실시간 시스템 개념 및 실시간성 보장 기법

Jinkyu Lee

Dept. of Computer Science and Engineering,
Sungkyunkwan University (SKKU)

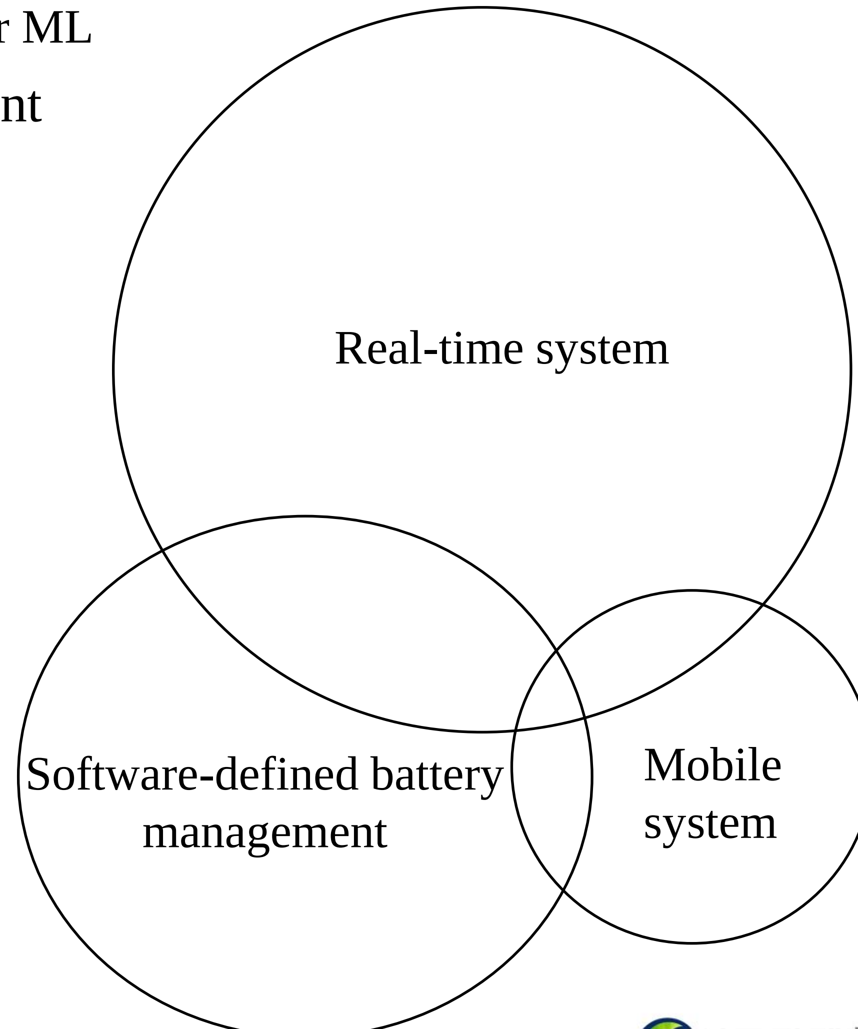
Instructor Information

- Jinkyu Lee 이진규
  - Associate Professor
    - Department of Computer Science and Engineering 소프트웨어학과, SKKU (2014.3~)
  - Postdoctoral Researcher
    - Department of EECS, University of Michigan
      - Advisor: Prof. Kang G. Shin
  - B.S., M.S., Ph.D.
    - Department of Computer Science 전산학부, KAIST
      - Advisor: Prof. Insik Shin & Ikjun Yeom

- Contact information
  - Email. jinkyu.lee@skku.edu
  - Website. https://rtclskku.github.io/website/jinkyulee.html
  - Real-Time Computing Lab@SKKU. https://rtclskku.github.io/website

Instructor Information

- Research areas
  - Real-time system
    - ML (Machine learning) for RT, RT for ML
  - Software-defined battery management
    - Battery management systems
  - Mobile system



Specialty:
Real-time scheduling

- System design and analysis with **timing guarantees**
- QoS support
- Resource management

Instructor Information

- Achievements: research
  - IEEE **RTSS** (Real-Time Systems Symposium)

    Top 1 RT conference, about 30 papers published per year
    - 19 papers published
  - IEEE **RTAS** (Real-Time and Embedded Technology and Application Symposium)

    Top 2 RT conference, about 30 papers published per year
    - 6 papers published
  - SCI(E) Journals
    - 53 papers, including 17 IEEE/ACM transactions/journals

- Achievements: professional activities
  - TPC (Technical Program Committee) co-chair: IEEE RTCSA 2020
  - TPC: IEEE **RTSS** 2018, 2020, 2021 and 2022
  - TPC: IEEE **RTAS** 2020, 2022 and 2023

Instructor Information

- Research example
  - How to capture users' grip for smartphones?

https://www.youtube.com/watch?v=FvQ87wmS6kk

SmartGrip: Grip Sensing System for Commodity Mobile Devices Through Sound Signals
Personal and Ubiquitous Computing 2020

Announcement

- 카메라를 켜주세요.
- Zoom 참여이름을 본인의 이름으로 셋팅해주세요.
- 과제: 오늘 수업이 끝나기 전까지 개인별로 질문 한개씩을 해주세요.

- 출석체크

Overall Schedule

- **실시간 시스템 개념**: 실시간 시스템의 개념을 설명하고, 실시간성 보장에 관한 기본적인 배경 지식을 공부함.
  - 09:00 – 10:15 **강의**
  - 10:30 – 11:45 **Case Study**
- 점심식사
  - 12:00 – 13:00
- **실시간성 보장 기법**: 주어진 마감 시간 내에 작업 완료를 “보장” 할 수 있는 기법들에 대해 공부함.
  - 13:00 – 14:15 **강의**
  - 14:30 – 15:45 **과제**
- 강의 노트 링크: https://rtclskku.github.io/website/papers/20230110_SSH.pdf

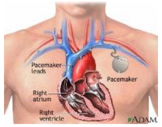
Real-Time System

- Systems that operate with time constraints (deadlines)
  - Important to produce accurate results **before deadlines**
  - Usually for safety-critical systems or mission-critical systems

Automotive

Avionics

Medical Devices



Robots

Military

Smartphones

Multimedia

Factory Automation

Real-Time System

- Systems that operate with time constraints (deadlines)
  - Important to produce accurate results **before deadlines**
  - Usually for safety-critical systems or mission-critical systems

From Honda

Real-Time Systems

- Definition
  - Systems whose correctness depend on their temporal aspects as well as their functional aspects

- Performance measure
  - Timeliness on timing constraints (deadlines)
  - Speed/average case performance are less significant.

- Key property
  - Predictability on timing constraints

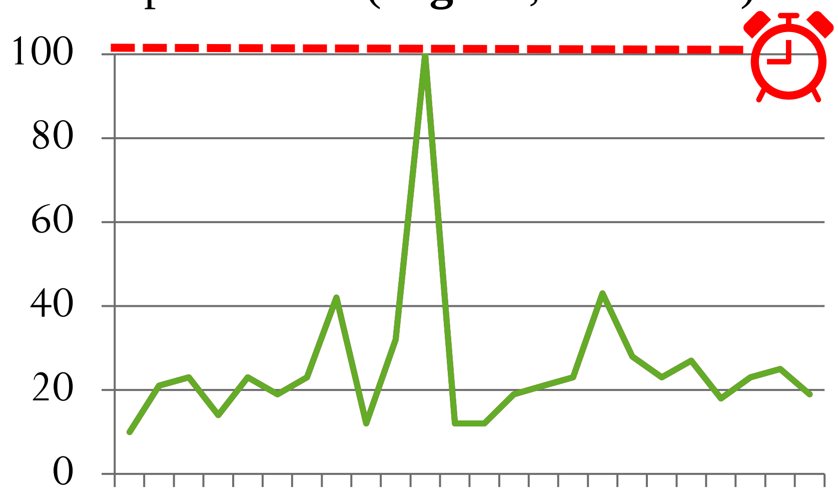
Misconceptions about Real-Time Systems

- Real-time ≠ fast
  - Rather predictable than fast
  - “A man drowned in a river with an average depth of 20 centimeters

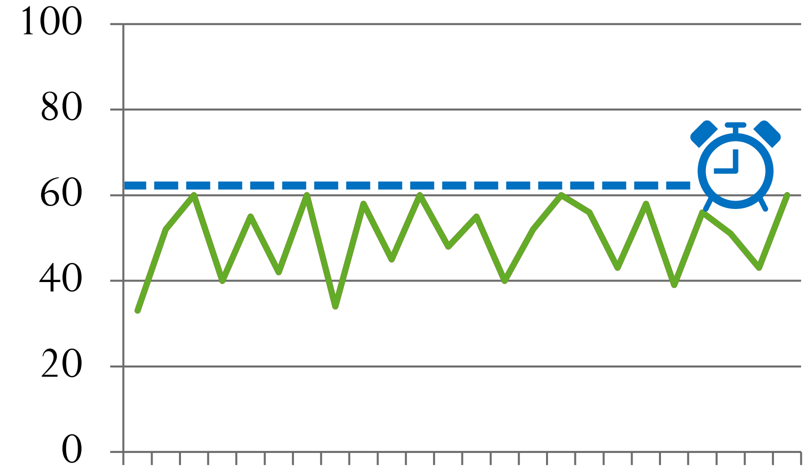
Real-Time System

- Systems that operate with time constraints (deadlines)
  - Important to produce accurate results **before deadlines**
  - Usually for safety-critical systems or mission-critical systems

response time (**avg=25**, **max=100**)



response time (**avg=50**, **max=60**)



Real-Time Systems: Hard vs. Soft

- Hard real-time systems
  - Disastrous or very serious consequences may occur if a deadline is missed
  - Validation is essential
    - Can all the deadlines be met, even under worst-case scenario?
  - Deterministic guarantees

- Soft real-time systems
  - Ideally, the deadline should be met for maximum performance. The performance degrades in case of deadline misses.
  - Best effort approaches / statistical guarantees

Real-Time Systems: *Research Topics*

Design *time-predictable* computing systems

\+

Timing guarantee and *analysis* of computing systems

Applications

Languages

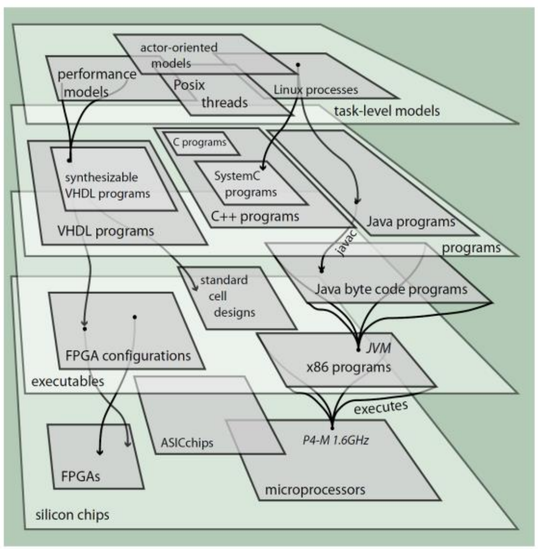
Operating Systems

Middleware

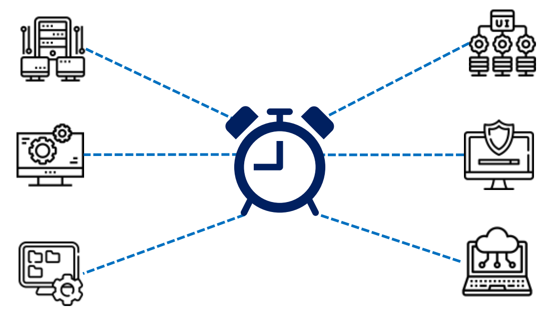
Architecture

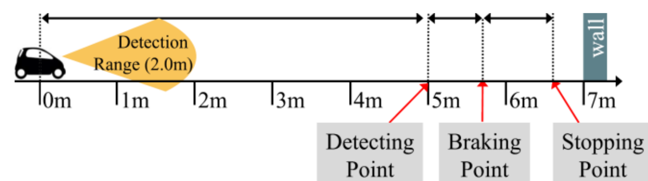
Hardware

…



Cyber Physical Systems: Design Challenges, IEEE ISORC, 2008.

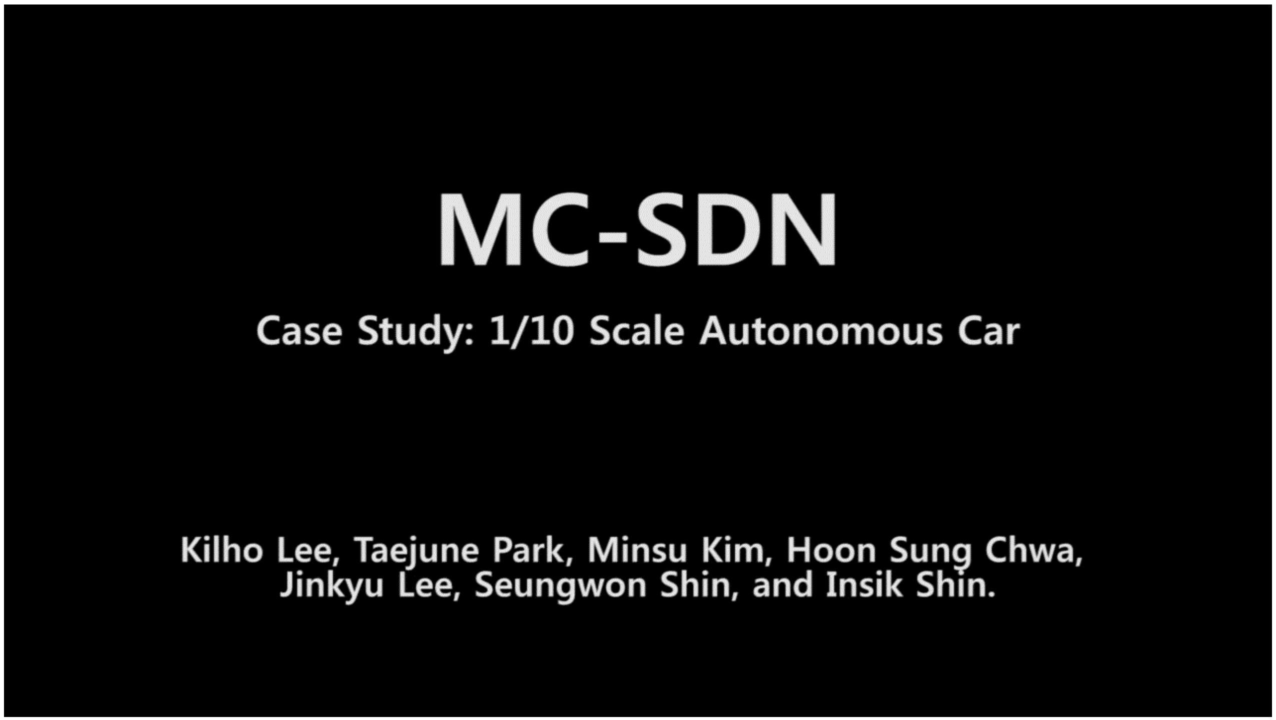




MC-SDN: Supporting Mixed-Criticality Real-Time Communication Using Software-Defined Networking, IEEE IoT Journal, 2019.

Real-Time Systems: *Research Topics*

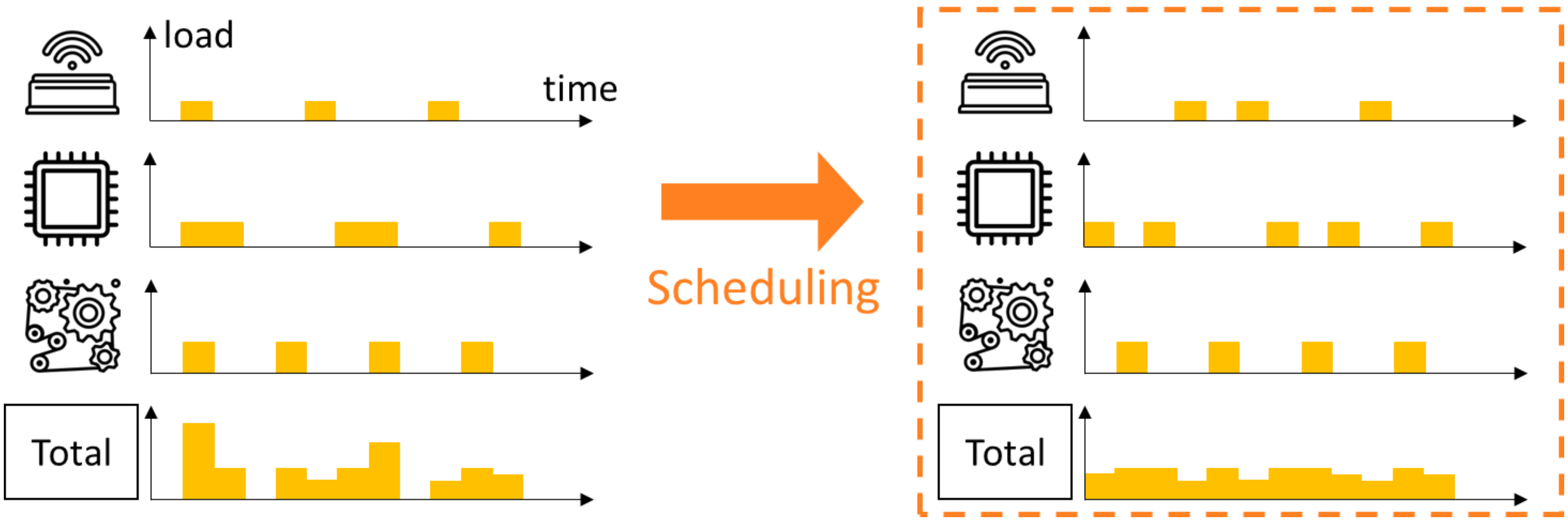
- Research example
  - How to provide timing guarantees for software-defined networks?



MC-SDN: Supporting Mixed-Criticality Real-Time Communication Using Software-Defined Networking
IEEE **RTSS** 2018

Real-Time Systems: *Research Topics*

- Research example
  - How to minimize battery aging while providing timing guarantees of real-time tasks?

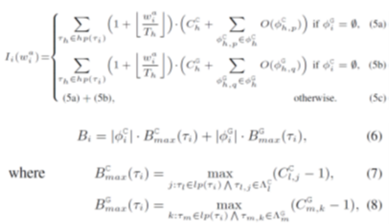


Battery Aging Deceleration for Power-Consuming Real-Time Systems
IEEE **RTSS** 2019

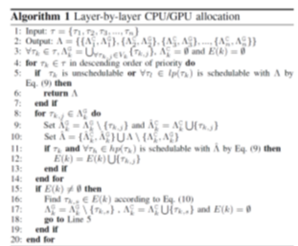
Real-Time Systems: *Research Topics*

- Research example
  - How to fully utilize CPU *and* GPU for time-predictable execution for DNN tasks?

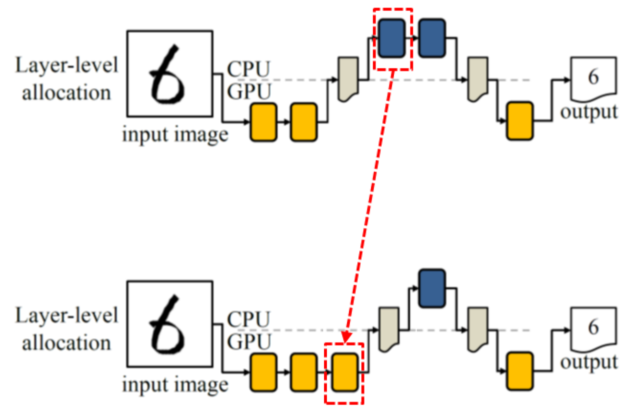
Timing analysis



Layer allocation



Timing guarantee, but accuracy loss



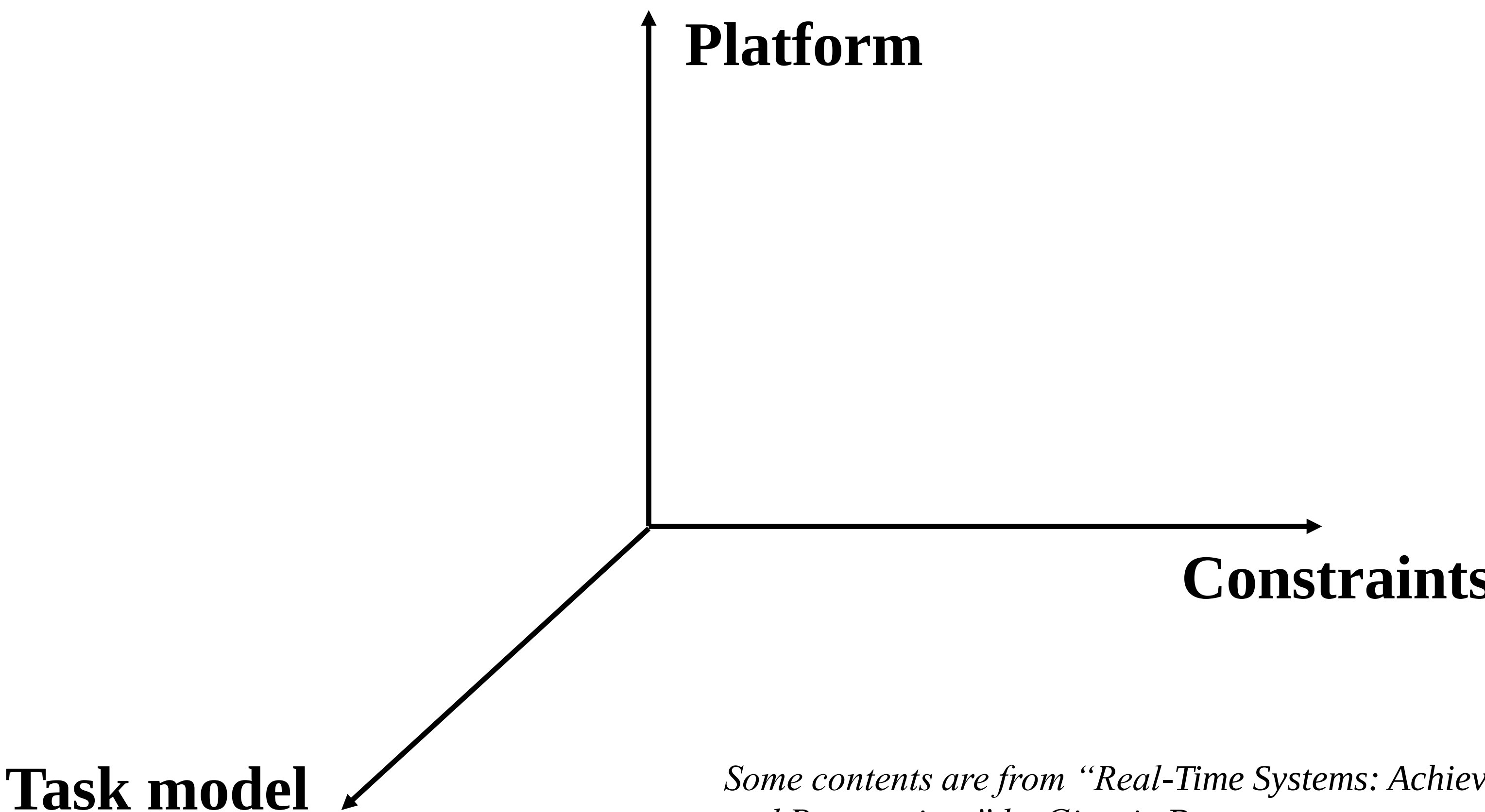
Online GPU/CPU re-allocation

Timing guarantee, and no accuracy loss

LaLaRAND: Flexible Layer-by-Layer CPU/GPU Scheduling for Real-Time DNN Tasks
IEEE **RTSS** 2021

Real-Time Scheduling

Schedule all real-time tasks
such that there is no single job deadline miss

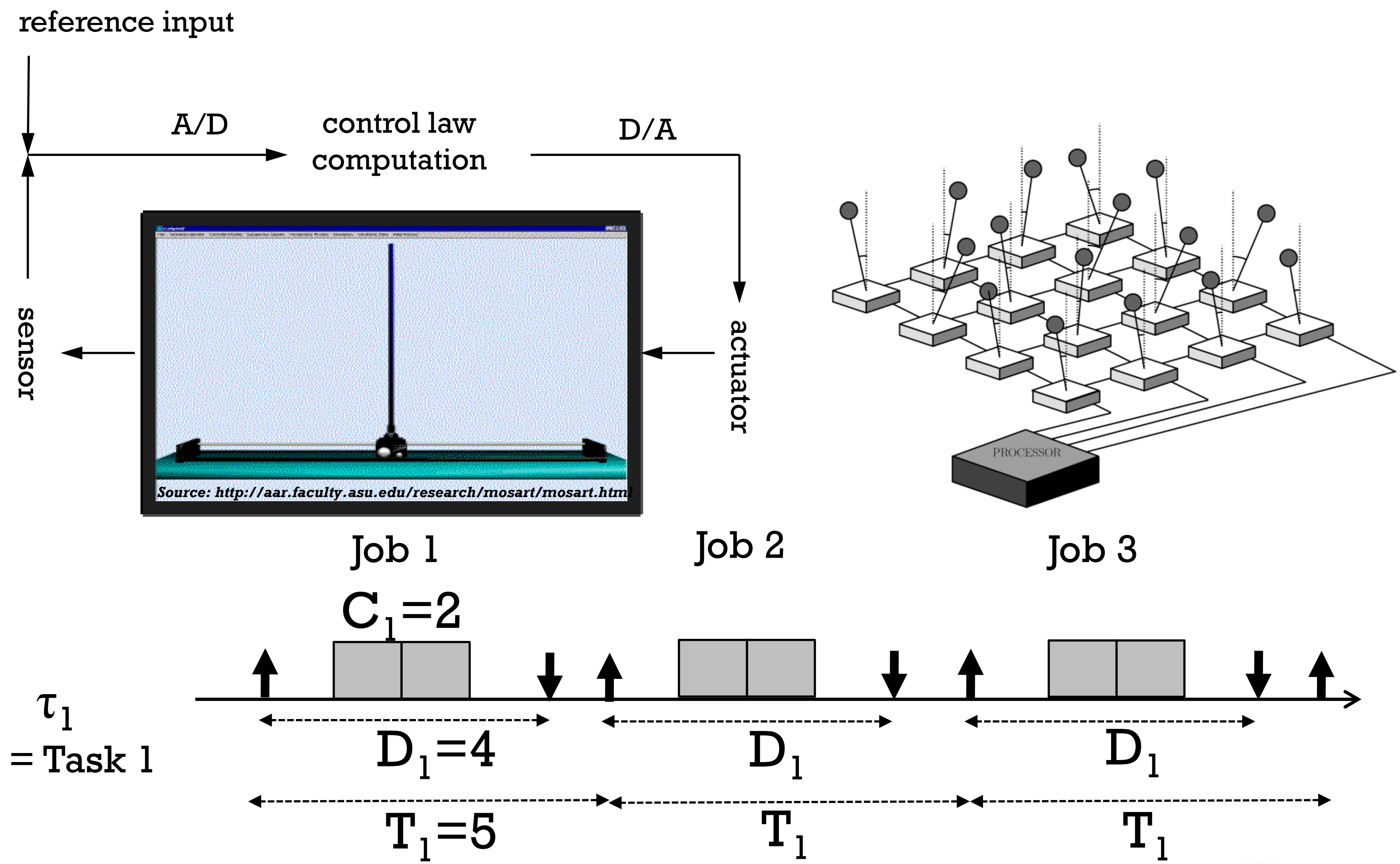


Some contents are from "Real-Time Systems: Achievements and Perspectives" by Giorgio Buttazzo.

Real-Time Scheduling

- Task models
  - Single job
  - Aperiodic
  - Sporadic
  - Periodic
  - DAG-structured
  - Gang-structured
  - …
- Constraints
  - Precedence
  - Shared resources
  - Self-suspensions
  - Mode changes
  - Fully preemptive
  - Fully non-preemptive
  - Limited preemptive
  - …
- Platforms
  - Uniprocessor
    - …
  - Multiprocessor
    - …
  - GPU + CPU
  - Distributed
  - …

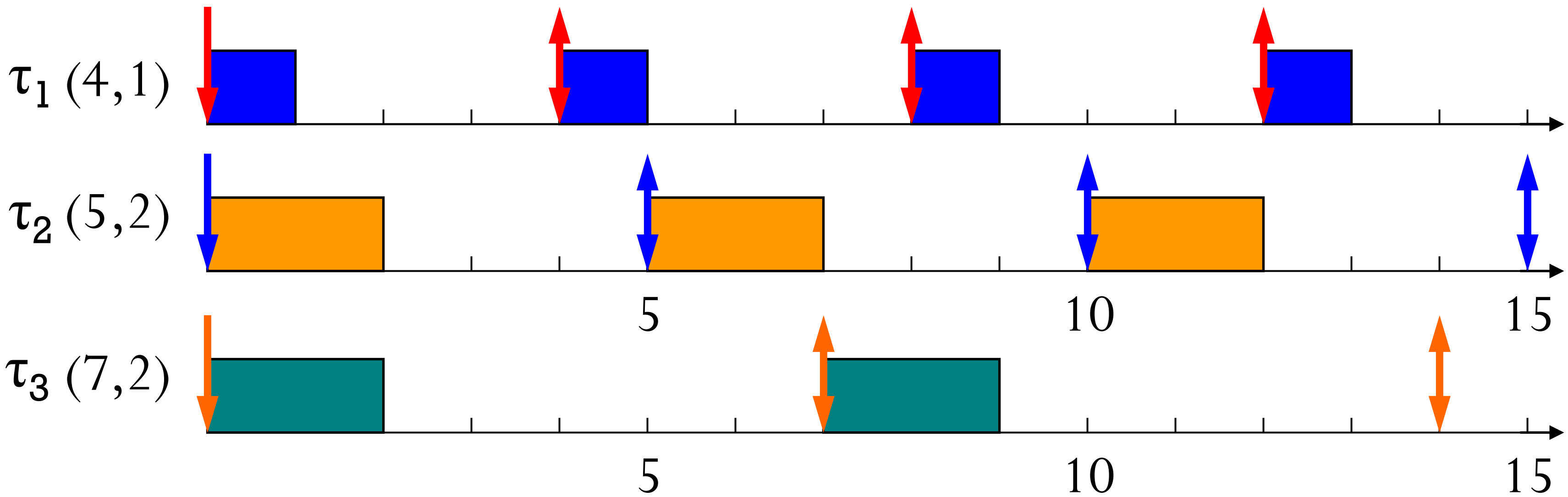
Real-Time Scheduling



Real-Time Scheduling

- Uniprocessor

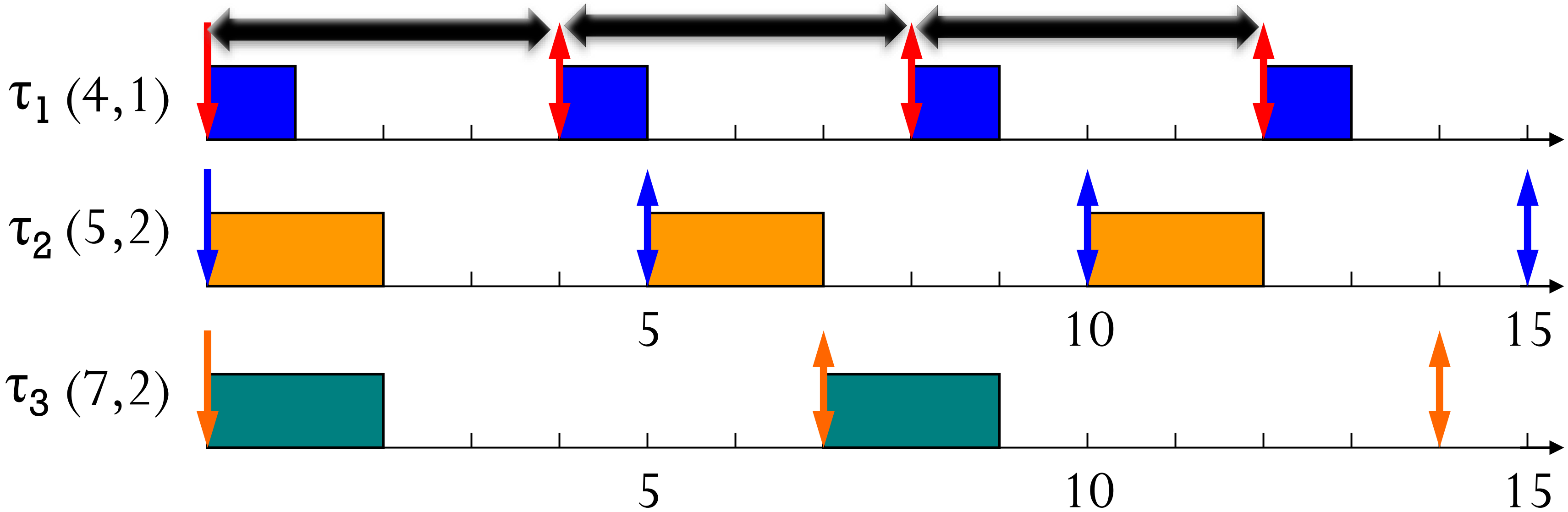
$T_1 = 4$, $D_1 = 4$, $C_1 = 1$



Real-Time Scheduling

- Uniprocessor **: at each time slot, only one job can be executed.**

$T_1=4$, $D_1=4$, $C_1=1$



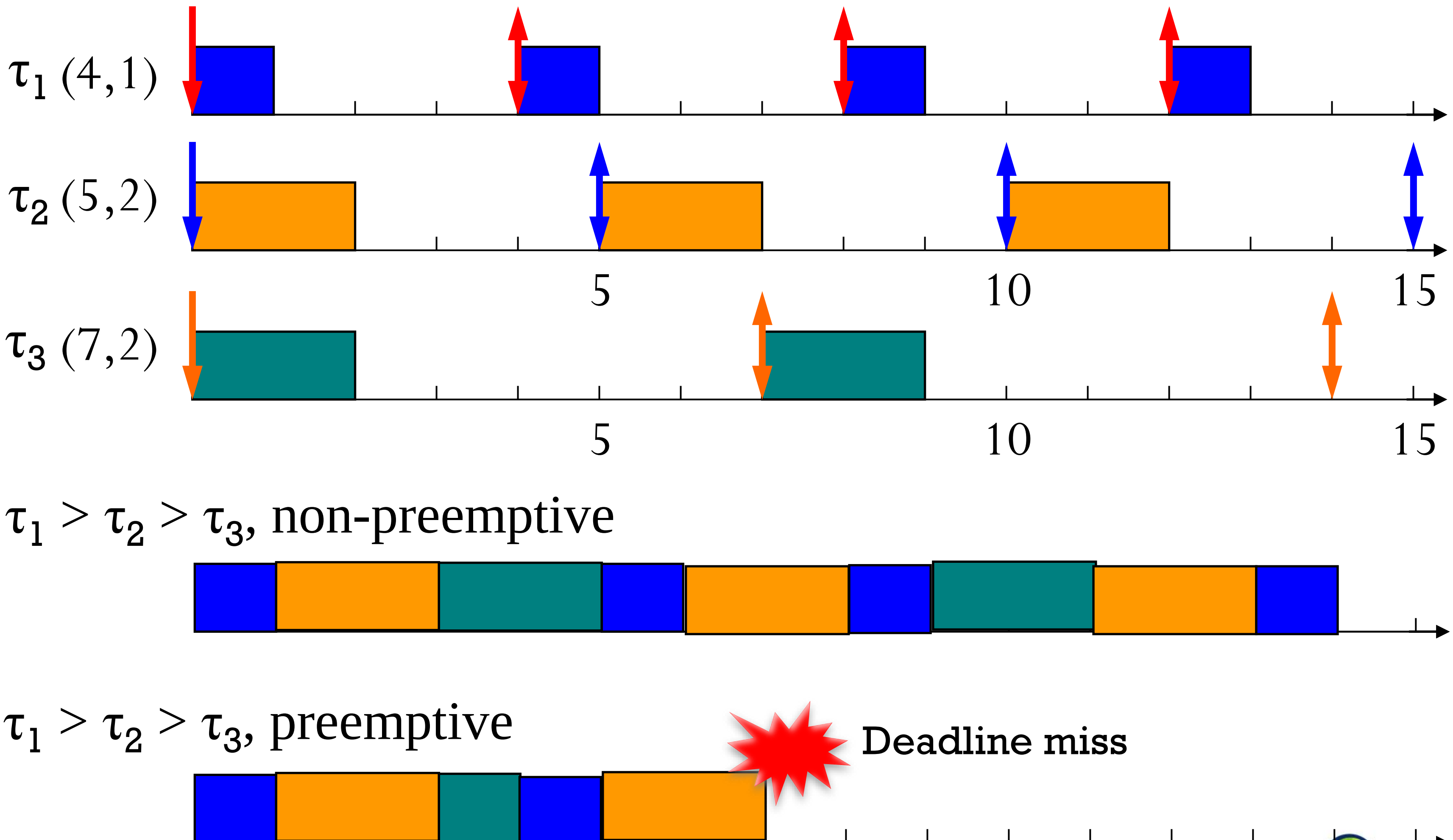
Schedule all real-time tasks such that there is no single job deadline miss

= Schedulable

Real-Time Scheduling

- Uniprocessor: **at each time slot, only one job can be executed.**

$T_1=4$, $D_1=4$, $C_1=1$



Real-Time Scheduling

- Task models
  - Single job
  - Aperiodic
  - Sporadic
  - **Periodic**
  - DAG-structured
  - Gang-structured
  - …
- Constraints
  - Precedence
  - Shared resources
  - Self-suspensions
  - Mode changes
  - **Fully preemptive**
  - **Fully non-preemptive**
  - Limited preemptive
  - …
- Platforms
  - **Uniprocessor**
    - …
  - Multiprocessor
    - …
  - GPU + CPU
  - Distributed
  - …

Real-Time Scheduling

- Optimization criteria

Schedule all real-time tasks such that there is no single job deadline miss

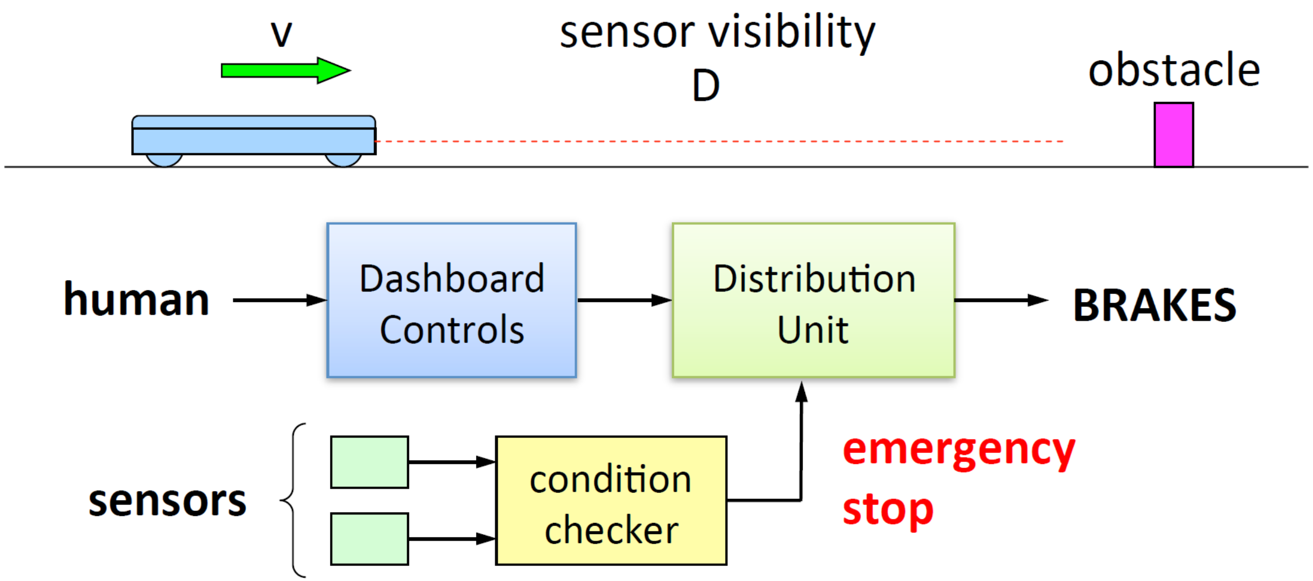
  - Feasibility
  - Response time
  - Maximum lateness
  - Utilization bandwidth
  - Energy consumption
  - Temperature
  - Number of processors

Real-Time Scheduling

- Industrial/practical usage of real-time scheduling
  - Rate Monotonic is used in most industrial settings.
  - Priority Inheritance is in most RT kernels.
  - Sporadic Server is specified in POSIX.
  - CBS is implemented in LINUX.
  - EDF is now supported by a few kernels
    - Erika Enterprise: certified OSEK and adopted by Magneti Marelli in next generation ECUs
    - Ada 2005 runtime support
    - Linux: SCHED_DEADLINE in mainline since June 2014
  - A lot of RT Tools are now available for
    - WCET estimation, schedulability analysis, scheduling simulation, formal verification, etc.

Real-Time Scheduling

- A practical example of real-time scheduling



GOAL: If an obstacle is detected, stop the train without hitting the obstacle.

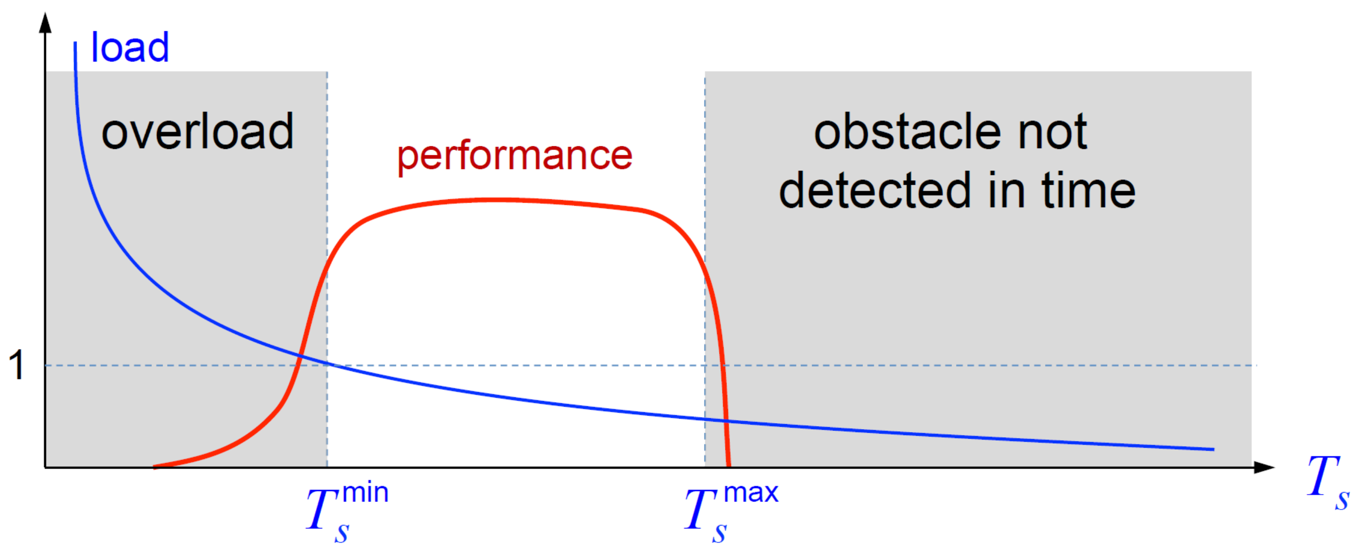
PROBLEM: Find the sampling periods of the sensors that guarantee the feasibility of the goal

From "Real-Time Systems: Achievements and Perspectives" by Giorgio Buttazzo

Real-Time Scheduling

- A practical example of real-time scheduling

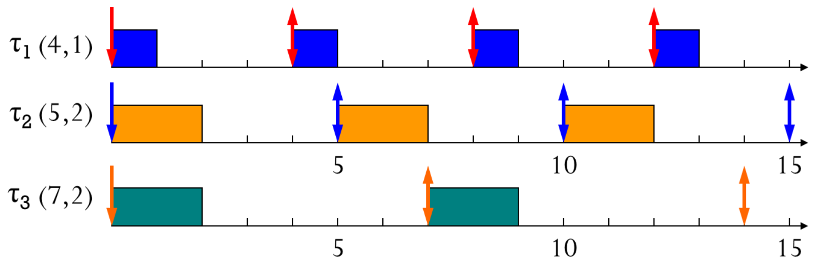
  - Tasks are scheduled by Rate Monotonic (implicit deadlines)
  - Let $\tau_s(C_s, T_s)$ be the periodic task devoted to sampling
  - Assume τ_s has the shortest period (highest priority).
  - Let U_{other} be the load of the other tasks

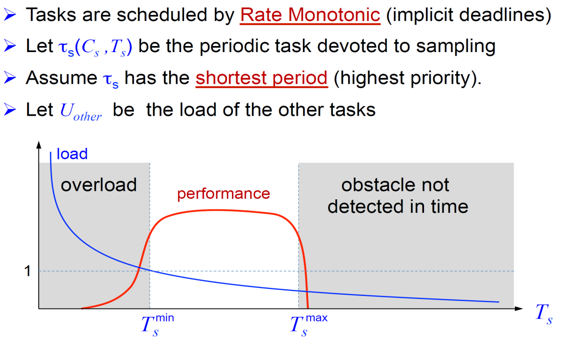


From "Real-Time Systems: Achievements and Perspectives" by Giorgio Buttazzo

Real-Time Scheduling

- A practical example of real-time scheduling





- What is the physical meaning of 1/4 + 2/5 + 2/7?
  - Utilization
  - What if the utilization is larger than 1.0?

From "Real-Time Systems: Achievements and Perspectives" by Giorgio Buttazzo

Real-Time Scheduling

- A practical example of real-time scheduling

The minimum period can be computed imposing the system schedulability by the Liu & Layland test:

The system is schedulable if $\frac{C_s}{T_s} + U_{other} \leq U_{\text{lub}}^{RM}$

that is $T_s \geq \frac{C_s}{U_{\text{lub}}^{RM} - U_{other}}$

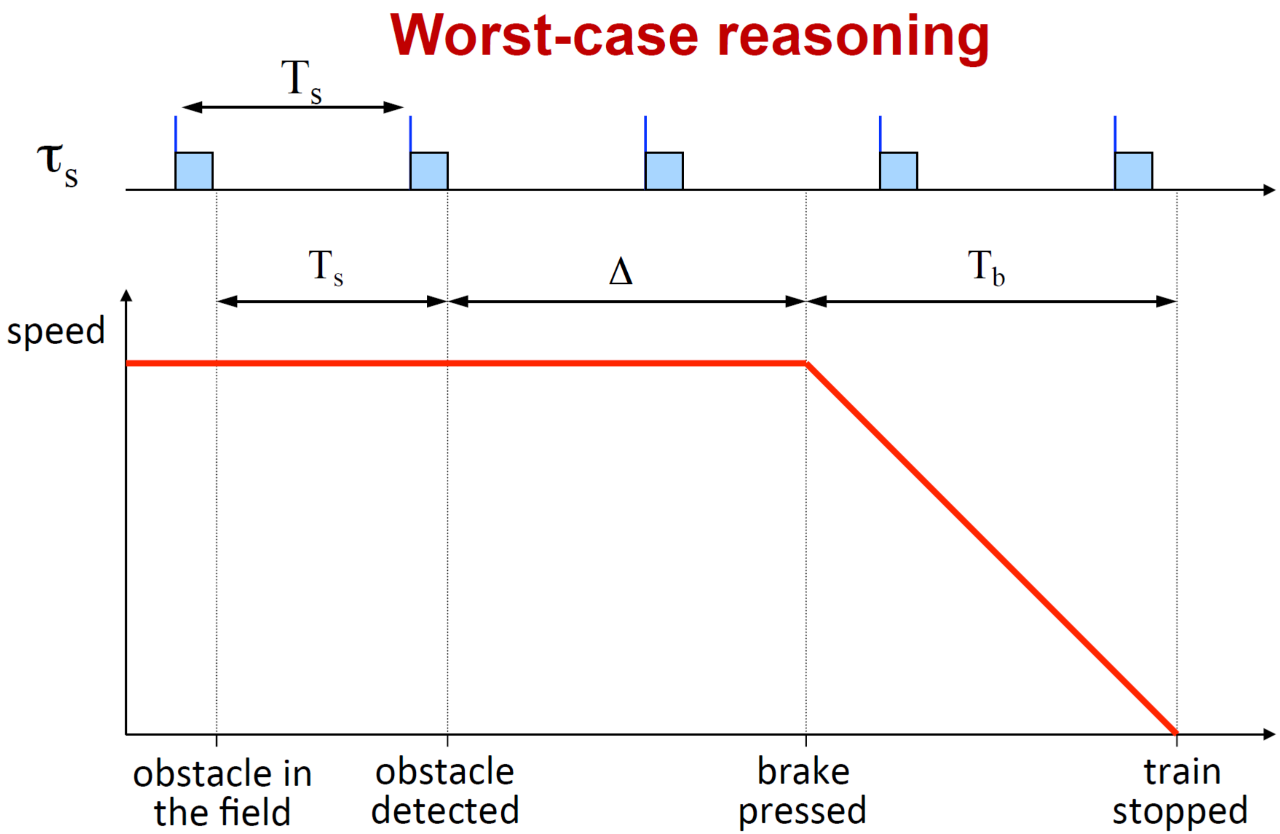
$$\boxed{T_s^{\min} = \frac{C_s}{U_{\text{lub}}^{RM} - U_{other}}}$$

- Example: 1/4 + 2/5 + 2/7 <= 0.7

From "Real-Time Systems: Achievements and Perspectives" by Giorgio Buttazzo

Real-Time Scheduling

- A practical example of real-time scheduling

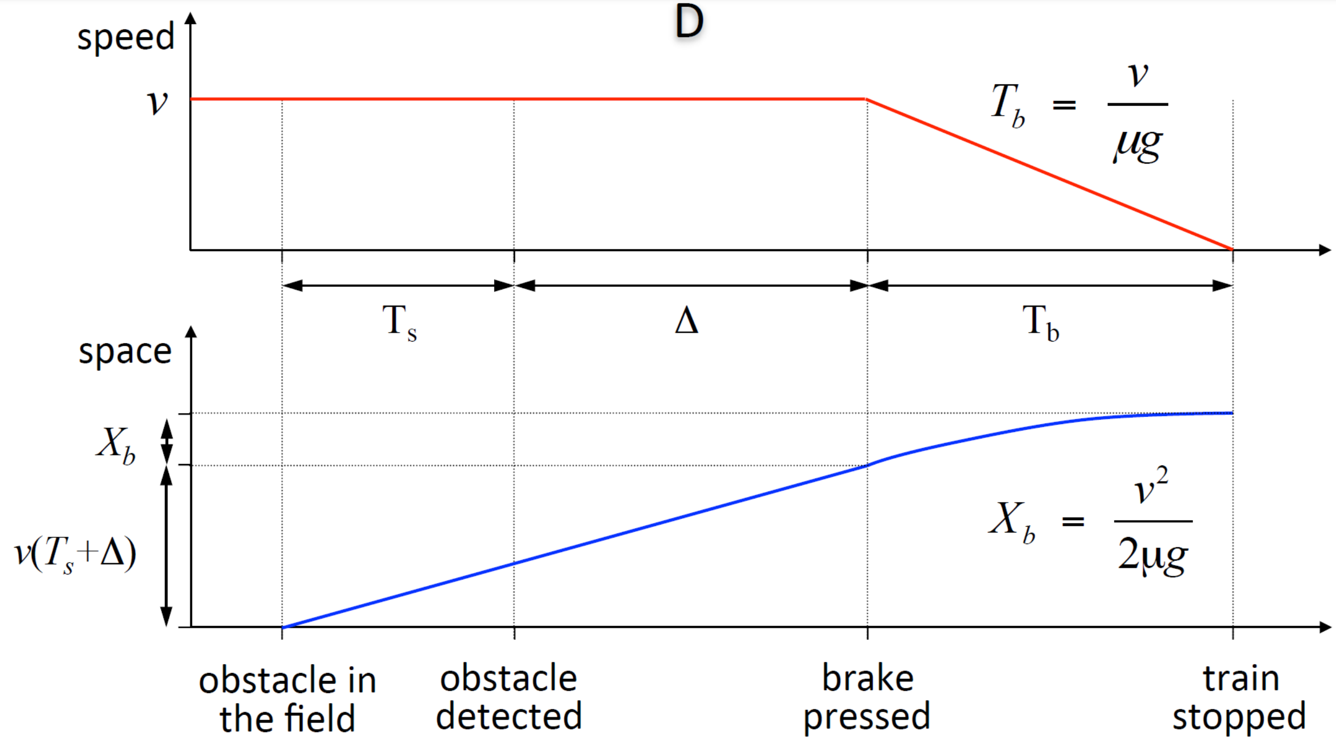


From "Real-Time Systems: Achievements and Perspectives" by Giorgio Buttazzo

Real-Time Scheduling

- A practical example of real-time scheduling

The space covered by the train in ($T_s+\Delta+T_b$) should not exceed D



From "Real-Time Systems: Achievements and Perspectives" by Giorgio Buttazzo

Real-Time Scheduling

- A practical example of real-time scheduling

$$v(T_s + \Delta) + X_b < D$$

$$X_b = \frac{v^2}{2\mu g}$$

$$v(T_s + \Delta) + \frac{v^2}{2\mu g} < D$$

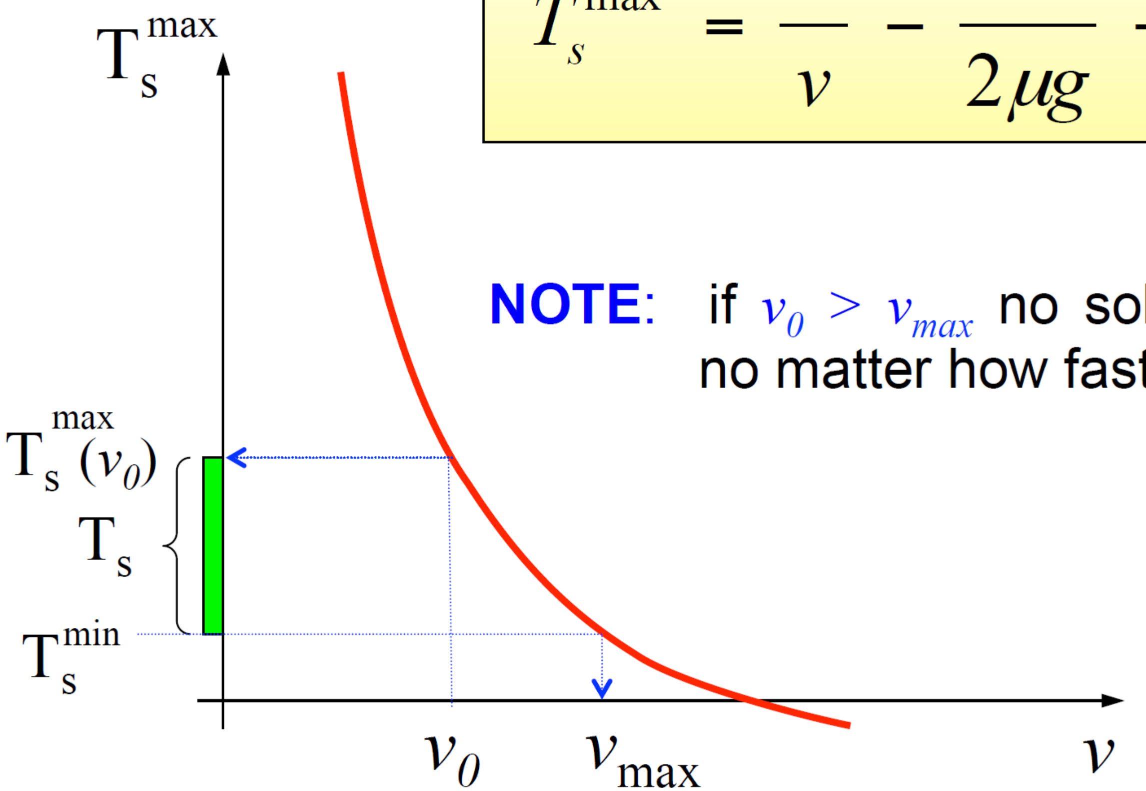
$$T_s < \frac{D}{v} - \frac{v}{2\mu g} - \Delta \rightarrow T_s^{\max}$$

From "Real-Time Systems: Achievements and Perspectives" by Giorgio Buttazzo

Real-Time Scheduling

- A practical example of real-time scheduling

$$T_s^{max} = \frac{D}{v} - \frac{v}{2\mu g} - \Delta$$

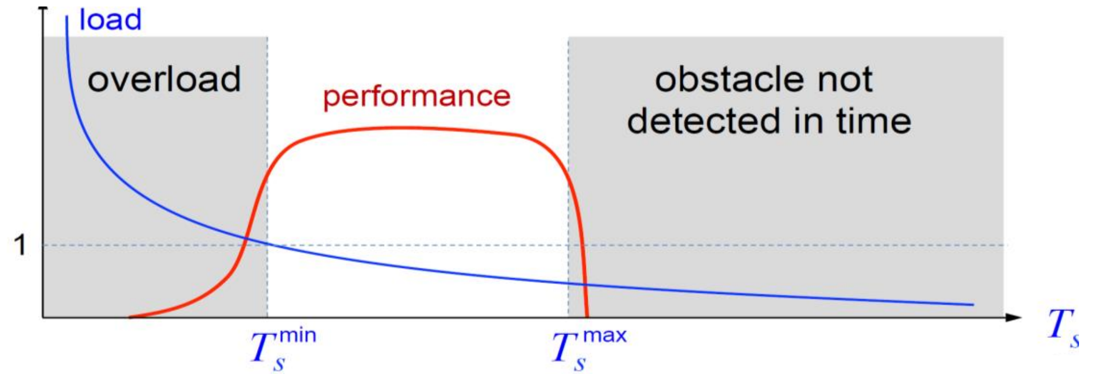


NOTE: if $v_0 > v_{max}$ no solution exists, no matter how fast the CPU is!

From "Real-Time Systems: Achievements and Perspectives" by Giorgio Buttazzo

Real-Time Scheduling

- A practical example of real-time scheduling



$$T_s^{\min} = \frac{C_s}{U_{\text{lub}}^{RM} - U_{other}}$$

$$T_s^{\max} = \frac{D}{v} - \frac{v}{2\mu g} - \Delta$$

Why this is possible?

The system is schedulable if $\frac{C_s}{T_s} + U_{other} \leq U_{\text{lub}}^{RM}$

Scheduling algorithm + Schedulability analysis

From "Real-Time Systems: Achievements and Perspectives" by Giorgio Buttazzo

Real-Time Scheduling

- Task model, constraints and platform are usually given.
- Then, how can we achieve timeliness of a real-time systems?

Schedule all real-time tasks such that there is no single job deadline miss

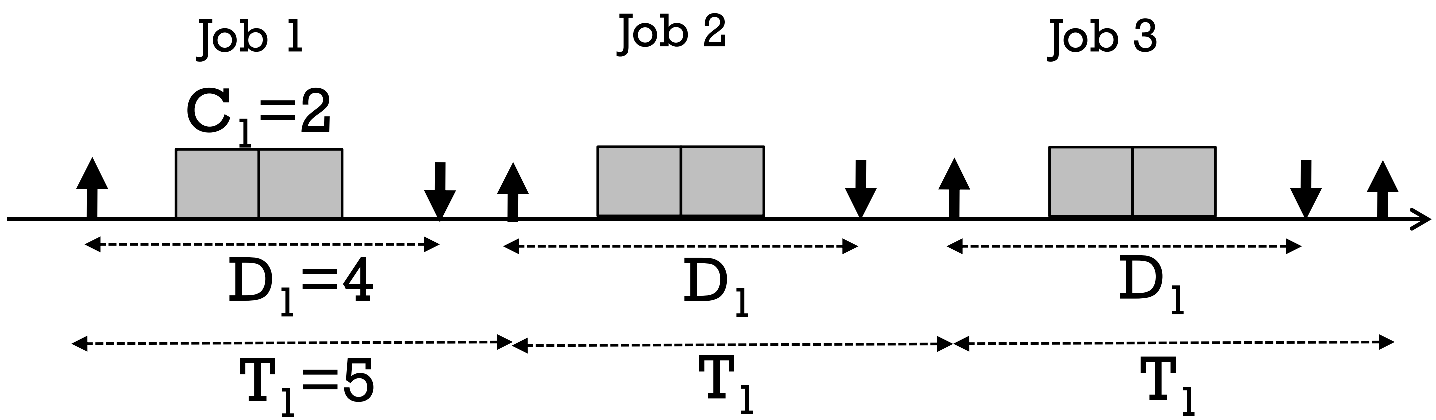
- Task models
  - Single job
  - Aperiodic
  - Sporadic
  - **Periodic**
  - DAG-structured
  - Gang-structured
  - …
- Constraints
  - Precedence
  - Shared resources
  - Self-suspensions
  - Mode changes
  - **Fully preemptive**
  - **Fully non-preemptive**
  - Limited preemptive
  - …
- Platforms
  - **Uniprocessor**
    - …
  - Multiprocessor
    - …
  - GPU + CPU
  - Distributed
  - …

Real-Time Scheduling

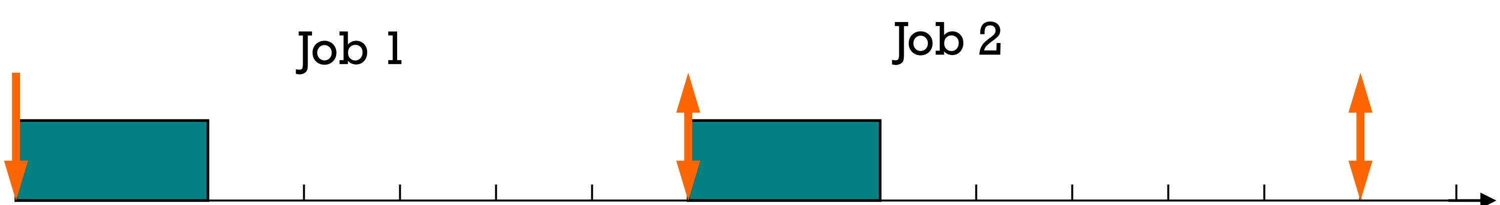
- From now on, we focus on the following *periodic/sporadic task model*
  - τ_i :Task i
    - T_i: the minimum separation / period
    - C_i: the worst-case execution time (WCET)
    - D_i: the relative deadline
  - Job: an instance of a task

(T_i, C_i, D_i)
Or
(T_i, C_i), implying $T_i = D_i$

$\tau_1(T_1,C_1,D_1)$
$\tau_1\ (5,2,3)$



$\tau_3(T_3,C_3)$
$\tau_3\ (7,2)$



Jinkyu Lee
Dept. of Computer Science and Engineering

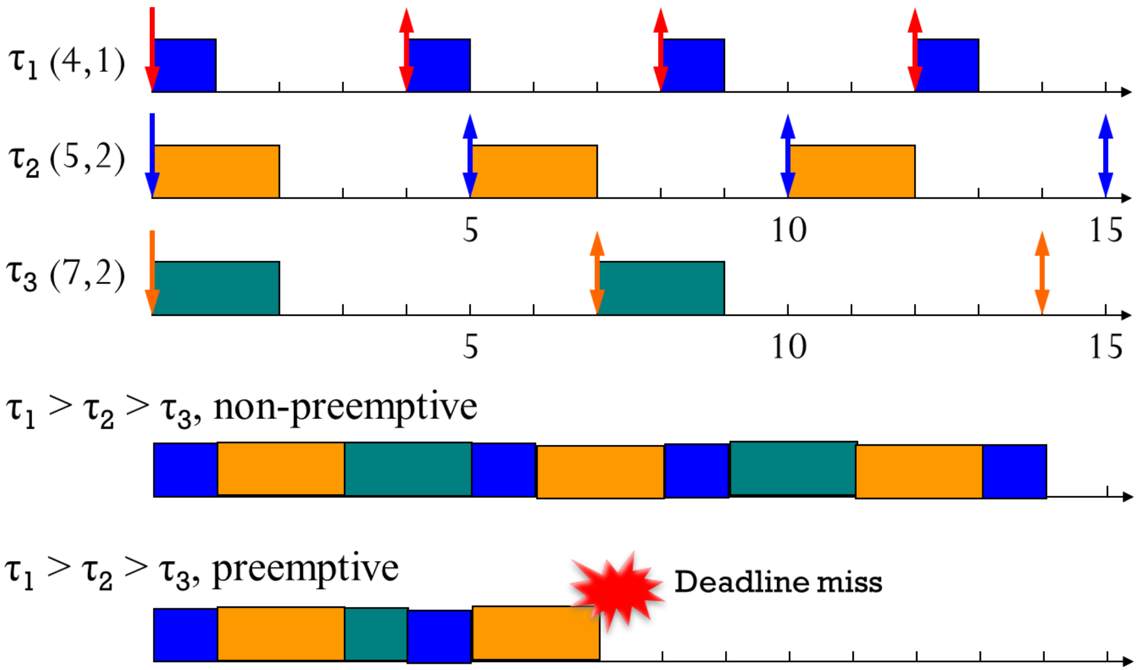
성균관대학교 SUNG KYUN KWAN UNIVERSITY

Real-Time Scheduling

- From now on, we focus on the following *periodic/sporadic task model*
  - τ_i :Task i
    - T_i: the minimum separation / period
    - C_i: the worst-case execution time (WCET)
    - D_i: the relative deadline
  - Where do these parameters come from?
    - T_i and D_i: usually from the target system (recall the hitting-obstacle example)
    - C_i: empirically measured, or code analysis
      - WCET analysis, under given platform

Real-Time Scheduling

- From now on, we focus on the *fully preemptive* constraint and a *uniprocessor platform*.
  - A higher-priority job can preempt a currently-executing lower-priority job.
  - At each time slot, only one job can be executed.



- Where do these constraints/platforms come from?
  - Usually from the target system and its design

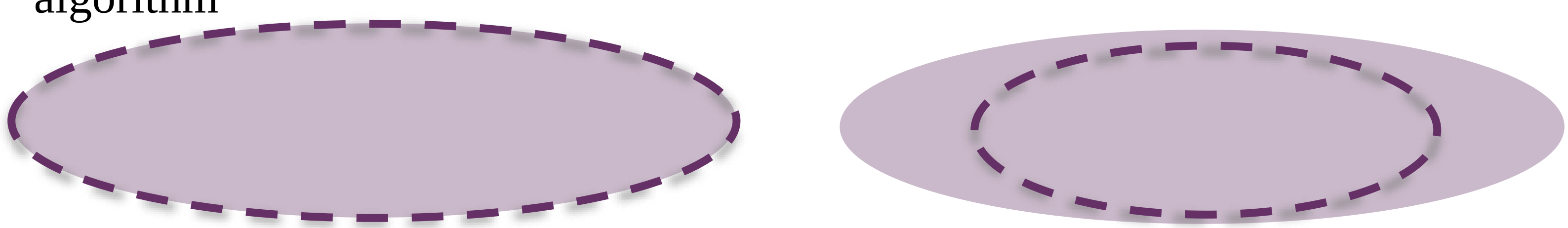
Real-Time Scheduling

- Scheduling goal

Schedule all real-time tasks such that there is no single job deadline miss

  - Under the periodic/sporadic task model, the fully preemptive constraint, and a uniprocessor platform

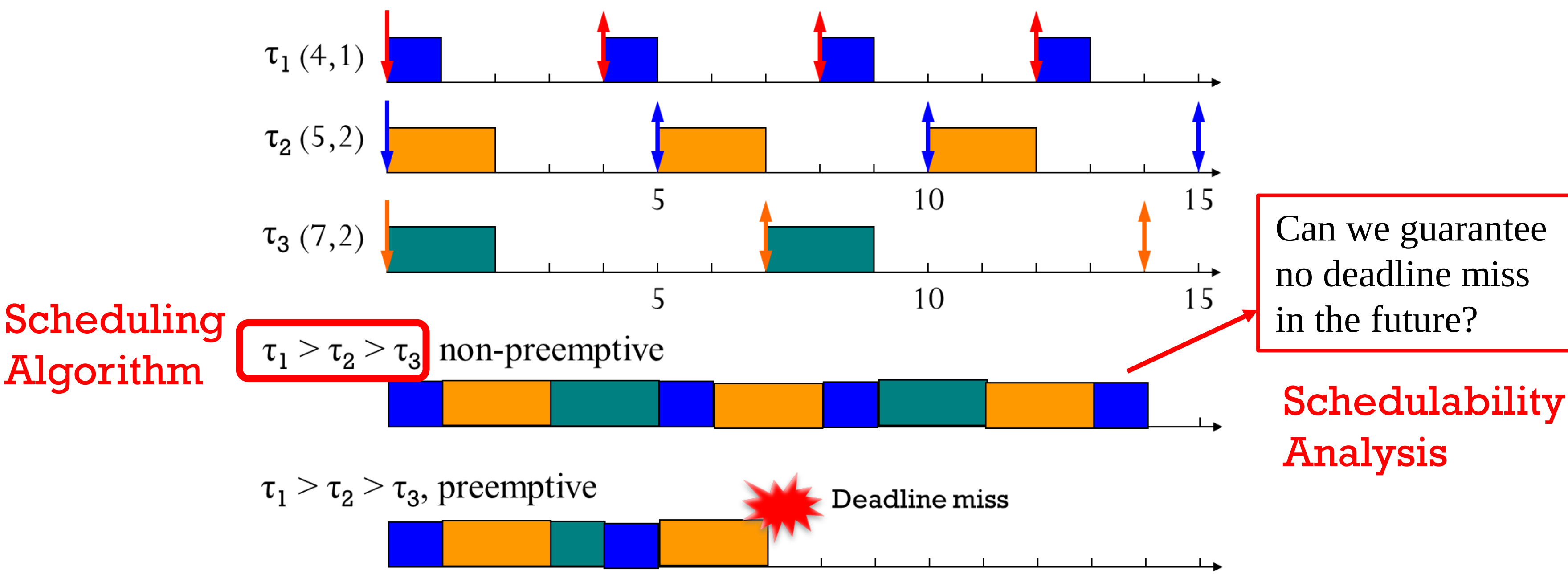
- Two main components for real-time scheduling
  - Scheduling algorithm
    - Determine when each job executes (which job to be executed)
  - Schedulability analysis
    - Guarantee no deadline miss of a given task set under the target scheduling algorithm



Exact (sufficient and necessary) **Only sufficient**

Real-Time Scheduling

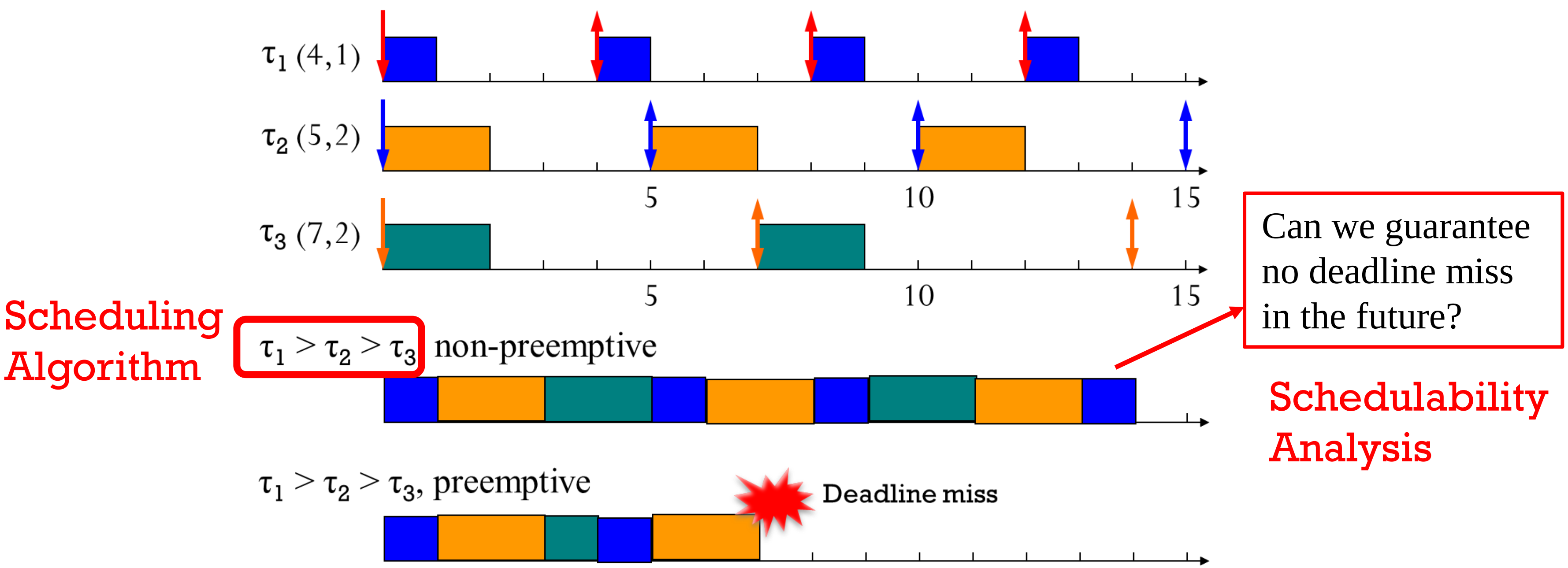
- Two main components for real-time scheduling
  - Scheduling algorithm
    - Determine when each job executes (which job to be executed)
  - Schedulability analysis
    - Guarantee no deadline miss of a given task set under the target scheduling algorithm



Jinkyu Lee
Dept. of Computer Science and Engineering

Real-Time Scheduling

- Why do we need schedulability analysis? Isn't it possible to guarantee schedulability simply by simulating a target set of tasks?
  - Is it possible to simulate all different combinations of offsets?
  - Can each simulation guarantee the schedulability even with actual execution times less than their corresponding WCETs?
  - Is it feasible to simulate a target set of tasks if their LCM is very large?



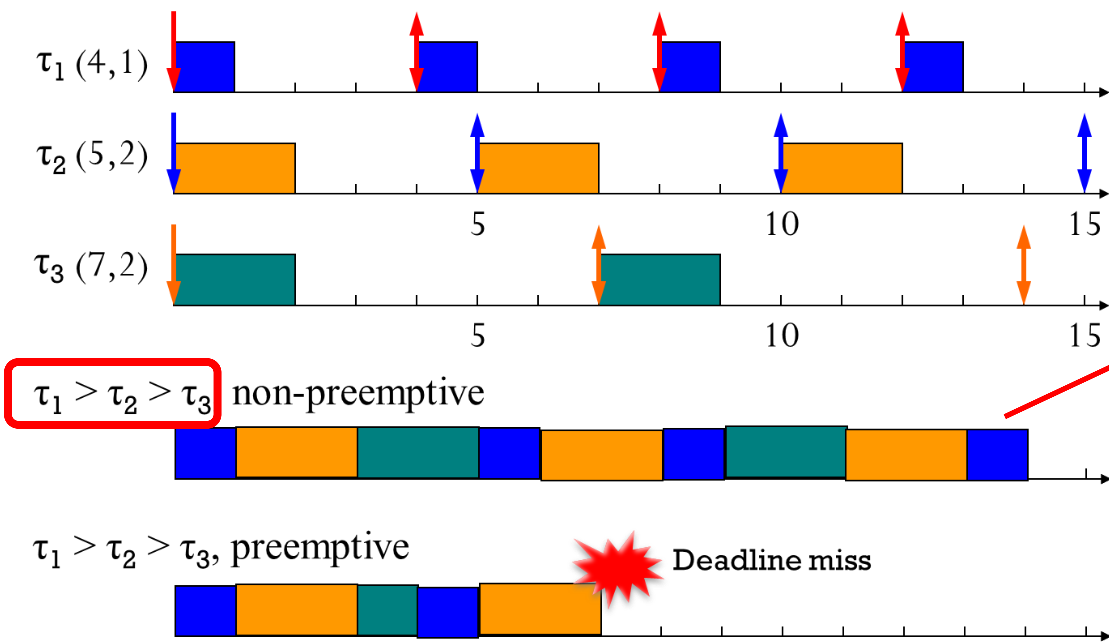
Real-Time Scheduling

- Which scheduling algorithm with which condition (i.e., schedulability analysis) guarantees that there is no single job deadline miss forever?
  - With any offset,
  - With any actual execution time,
  - With any arbitrary long period,

Example

The system is schedulable if $\frac{C_s}{T_s} + U_{other} \le U_{\mathrm{lub}}^{RM}$

Scheduling Algorithm

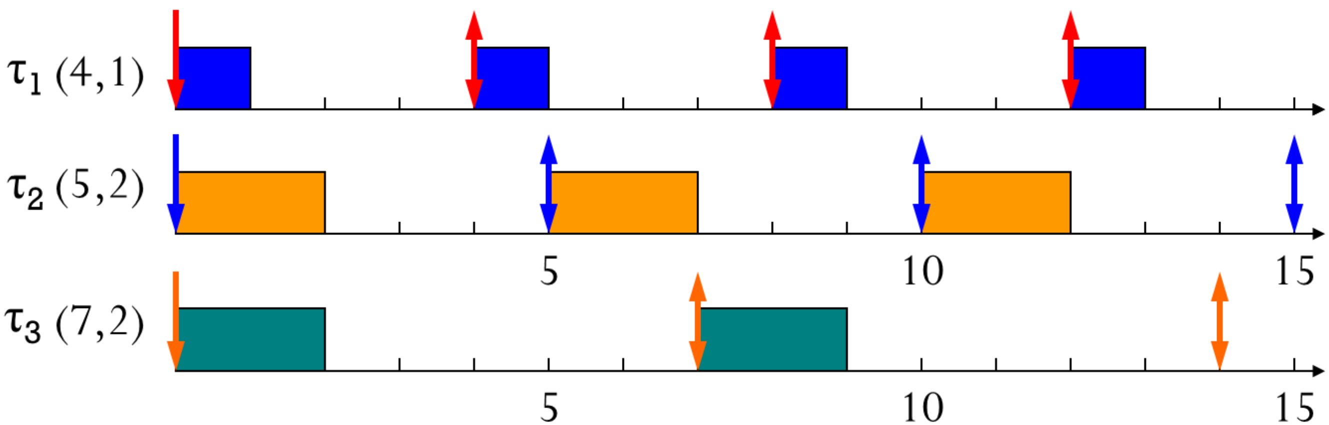


Can we guarantee no deadline miss in the future?

Schedulability Analysis

Real-Time Scheduling

- Let's focus on the most popular (simplest) setting.
  - A single processor
  - Hard deadline
  - Independent periodic tasks
  - Relative deadline = period ($D_i = T_i$)
  - Preemptable without any limit
  - No overhead for context switch

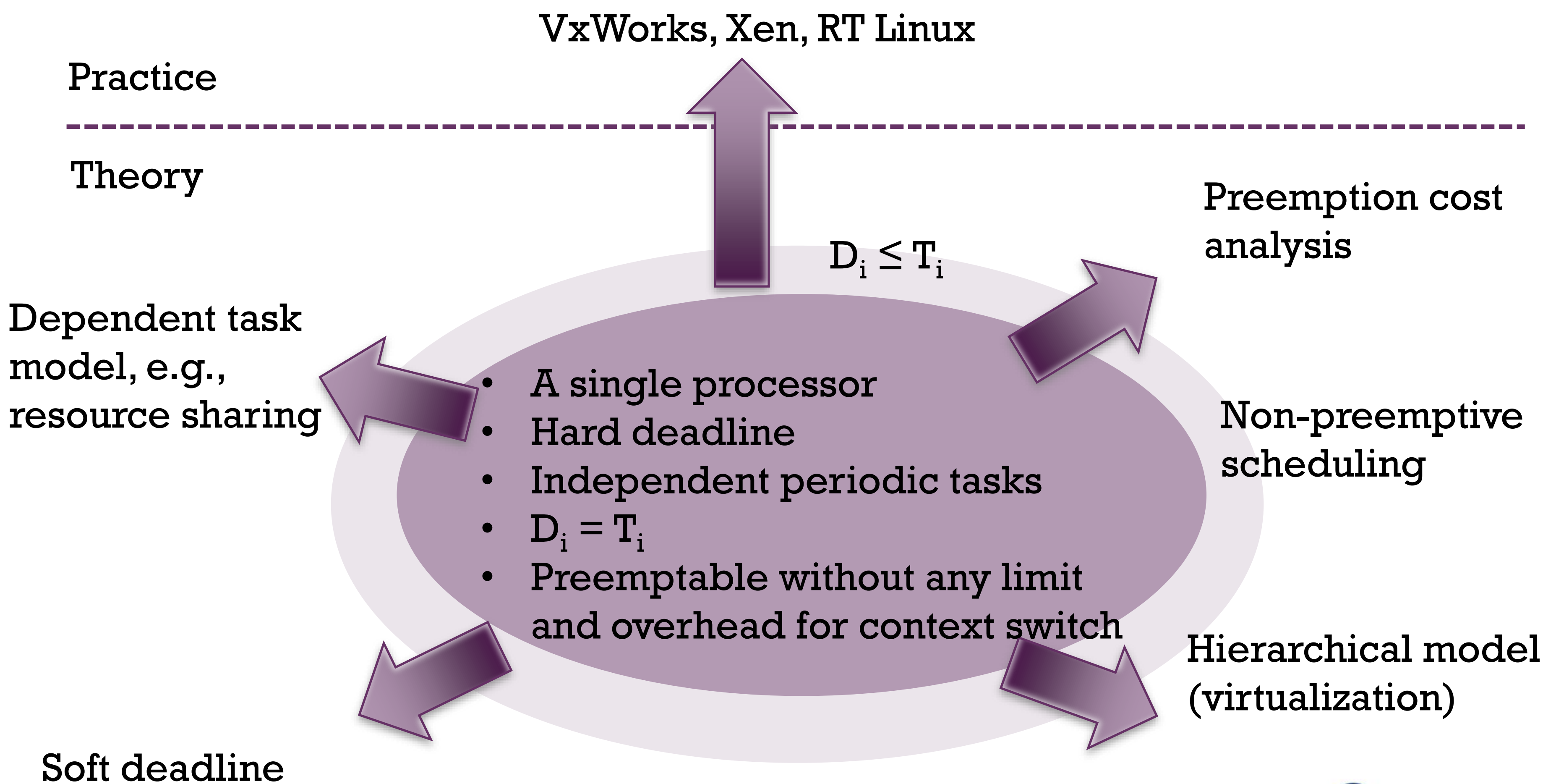


Jinkyu Lee
Dept. of Computer Science and Engineering

성균관대학교 SUNG KYUN KWAN UNIVERSITY

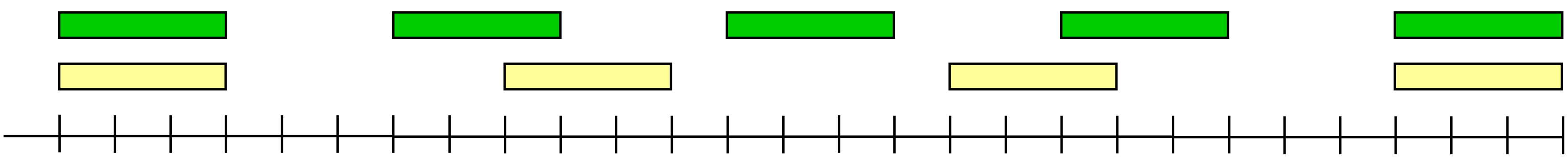
Real-Time Scheduling

- How can this setting be extended?



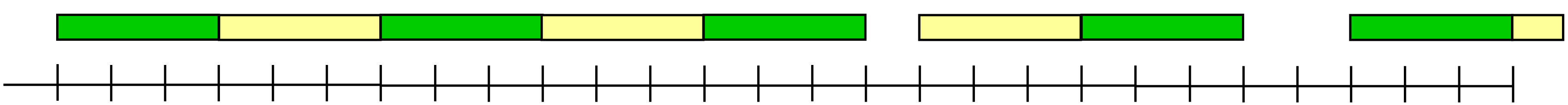
Clock-Driven Task Scheduling

- Clock-driven
  - a schedule determines (off-line) which job to be executed at each instant
  - static or *cyclic*
  - predictable and deterministic
  - scheduler: invoked by a timer
  - multiple *tables* for different operation modes



$T_1 = 6, C_1 = 3, D_1 = 6$
$T_2 = 8, C_2 = 3, D_2 = 8$

Major cycle = lcm (6,8) = 24

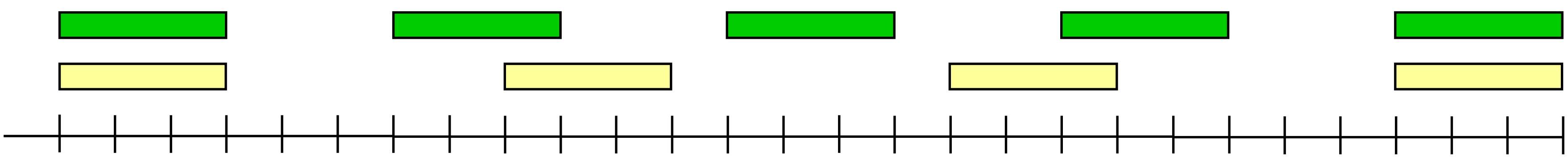


Pros and Cons of Cyclic Schedule

- Pros
  - Simple, table-driven, easy to validate (knows what's going on at any moment)
  - Fits well for *harmonic* periods and small system variations
  - Static schedule => deterministic, static resource allocation, *no preemption*
  - Small jitter
- Cons
  - Difficult to change (need to re-schedule all tasks)
  - Fixed released times for the set of tasks
  - Difficult to deal with different temporal dependencies
  - Scheduling algorithm may become complex (NP-hard)
  - Doesn't support aperiodic and sporadic tasks efficiently

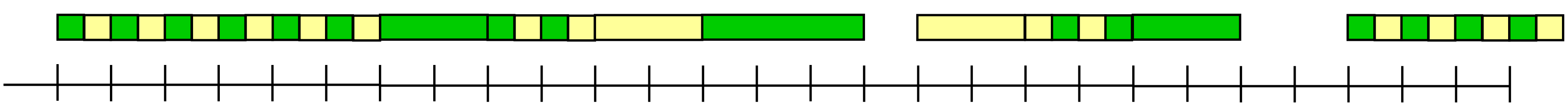
Round-Robin Task Scheduling

- Weighted Round-robin
  - Interleave job executions
  - Allocate a time slice to each job in the FIFO queue
  - Time slice may vary while sharing the processor
  - Good for pipelined jobs, e.g., network packets



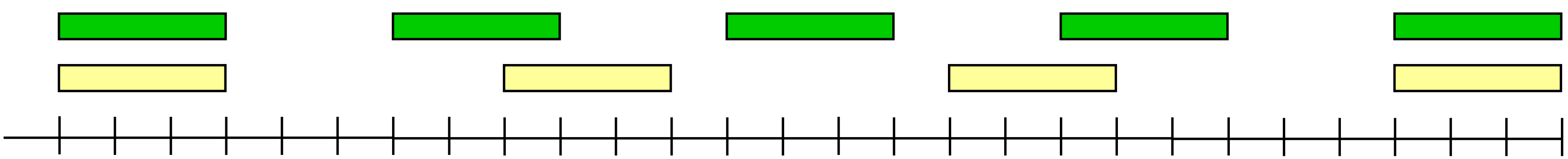
$T_1 = 6, C_1 = 3, D_1 = 6$
$T_2 = 8, C_2 = 3, D_2 = 8$

Major cycle = lcm (6,8) = 24



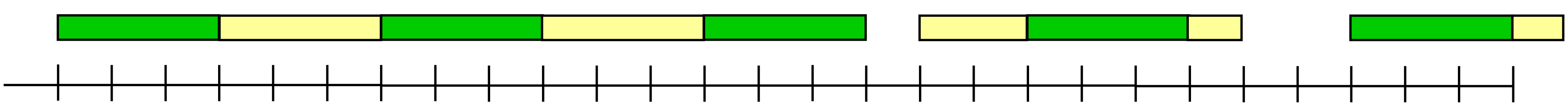
Priority-Driven Task Scheduling

- Priority-driven
  - The highest-priority job gets to run until completion or blocked
  - A processor is never idle if ready jobs are waiting (work-conserving)
  - Preemptive or non-preemptive
  - Priority assignment can be static or dynamic
  - Scheduler just looks at the priority queue for waiting jobs (list schedule)



$T_1 = 6, C_1 = 3, D_1 = 6$
$T_2 = 8, C_2 = 3, D_2 = 8$

Major cycle = lcm (6,8) = 24

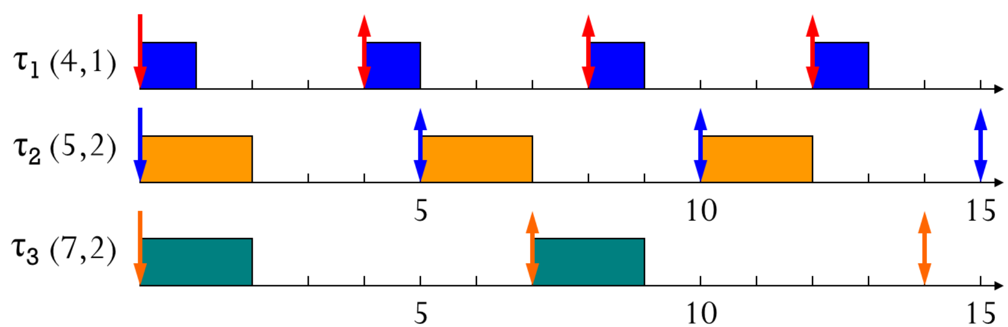


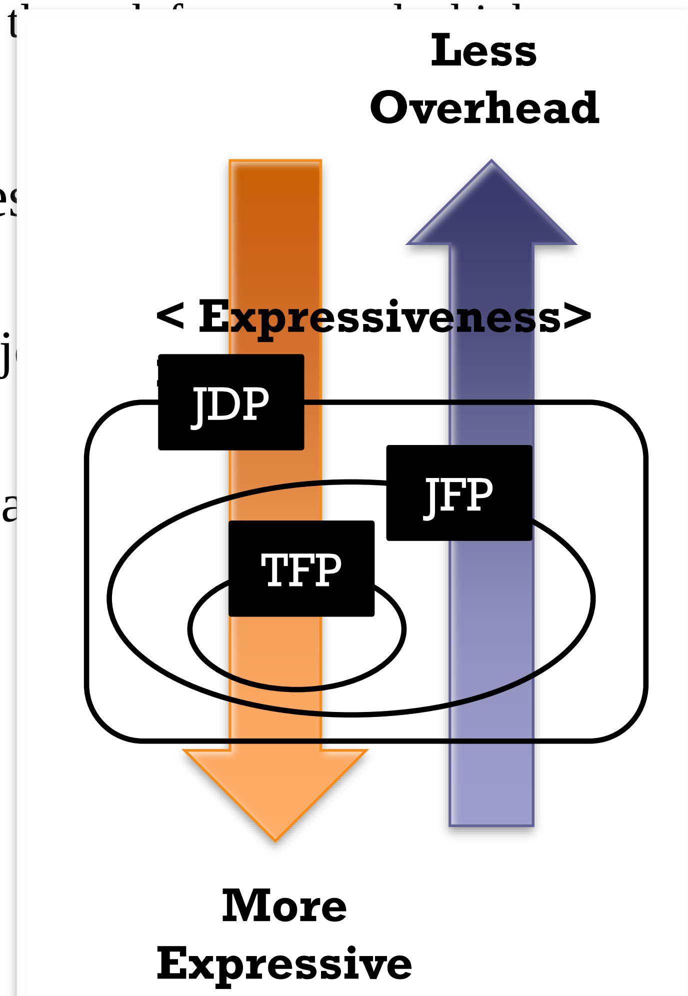
Priority-Driven Task Scheduling

- Why priority-driven scheduling algorithm?
  - Use priority to represent urgency/importance
  - Easy implementation of scheduler (compare task priorities and then dispatch tasks accordingly)
  - Tasks can be added or removed easily
  - No *direct control* of execution instant

Classification of Priority-Driven Scheduling

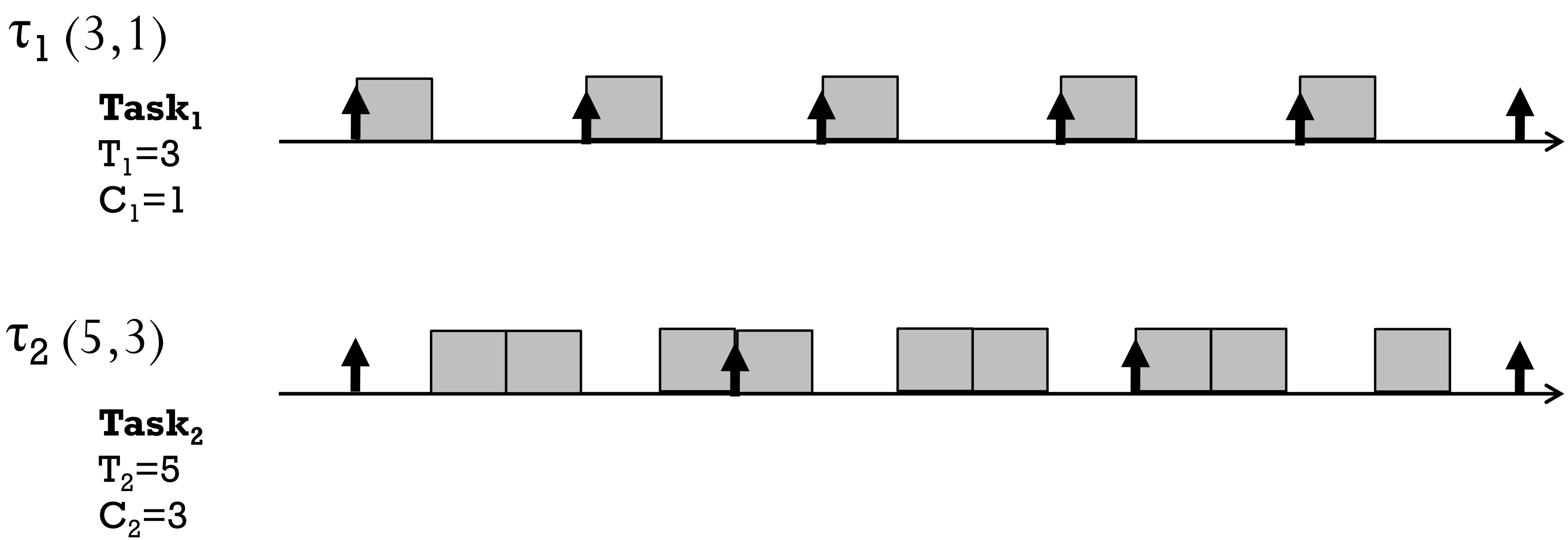
- **Task-level fixed-priority (TFP)**: all jobs of a periodic task have the same fixed priority
  - RM (Rate-monotonic) [Liu and Layland 1973] – the higher t
    its priority
- Task-level dynamic-priority: different prioritie
  periodic task
  - **Job-level fixed-priority (JFP)**: priority of each j
    - EDF (Earliest Deadline First) [Liu and Layland 1973]
  - **Job-level dynamic-priority (JDP)**: priority of ea
    time
    - LLF (Least Laxity First) [Leung 1989]





Classification of Priority-Driven Scheduling

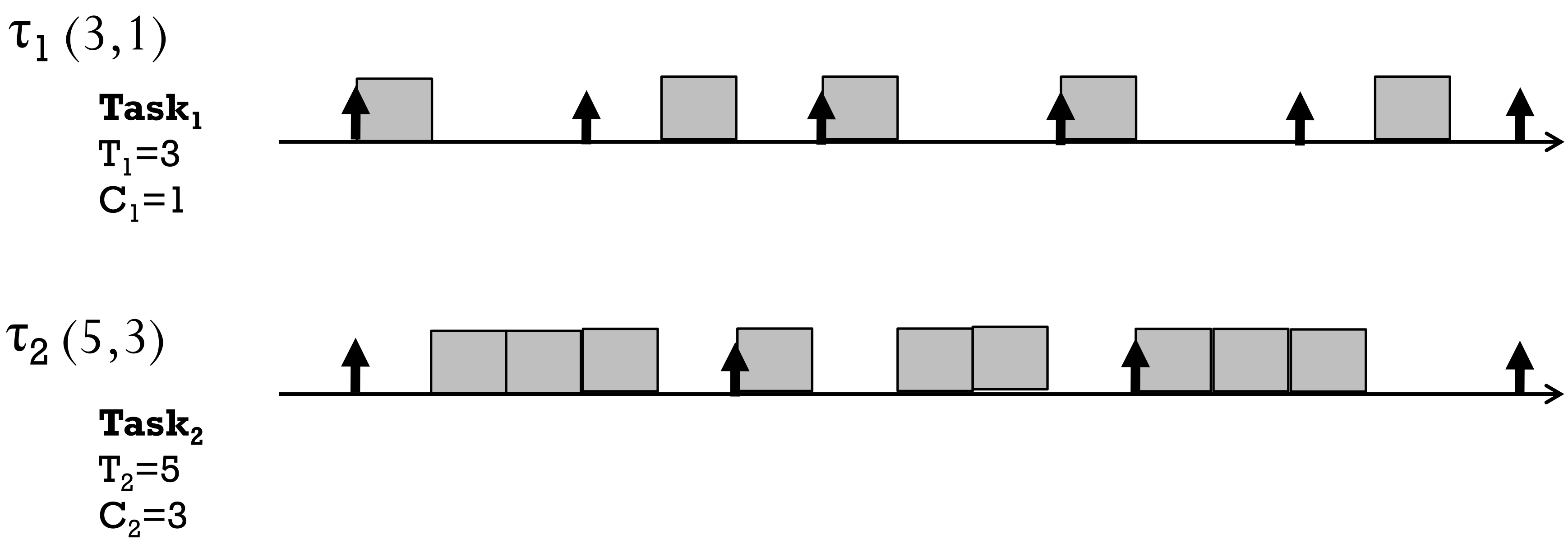
- **Task-level fixed-priority (TFP)**: all jobs of a periodic task have the same fixed priority
  - RM (Rate-monotonic) [Liu and Layland 1973] – the higher the task frequency, the higher its priority



Jinkyu Lee
Dept. of Computer Science and Engineering

Classification of Priority-Driven Scheduling

- **Job-level fixed-priority (JFP)**: priority of each job is fixed
  - EDF (Earliest Deadline First) [Liu and Layland 1973]



Jinkyu Lee
Dept. of Computer Science and Engineering

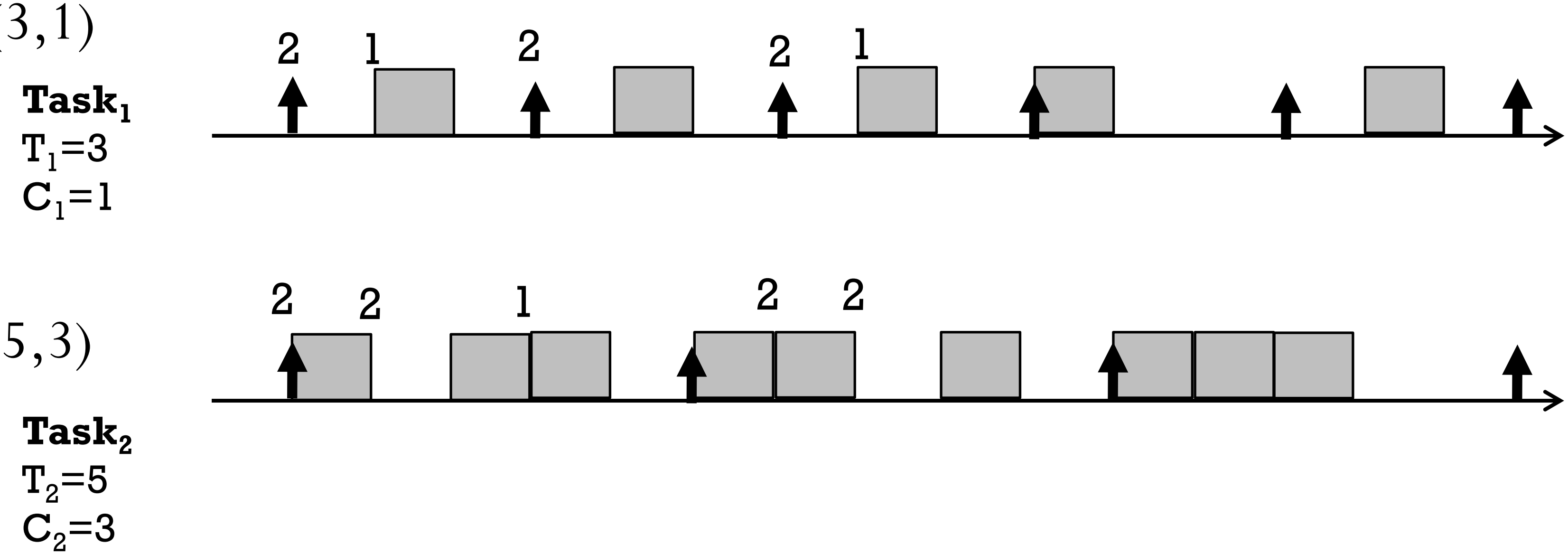
성균관대학교 SUNG KYUN KWAN UNIVERSITY

Classification of Priority-Driven Scheduling

- **Job-level dynamic-priority (JDP)**: priority of each job can change over time
  - LLF (Least Laxity First) [Leung 1989]

$\tau_1\ (3,1)$

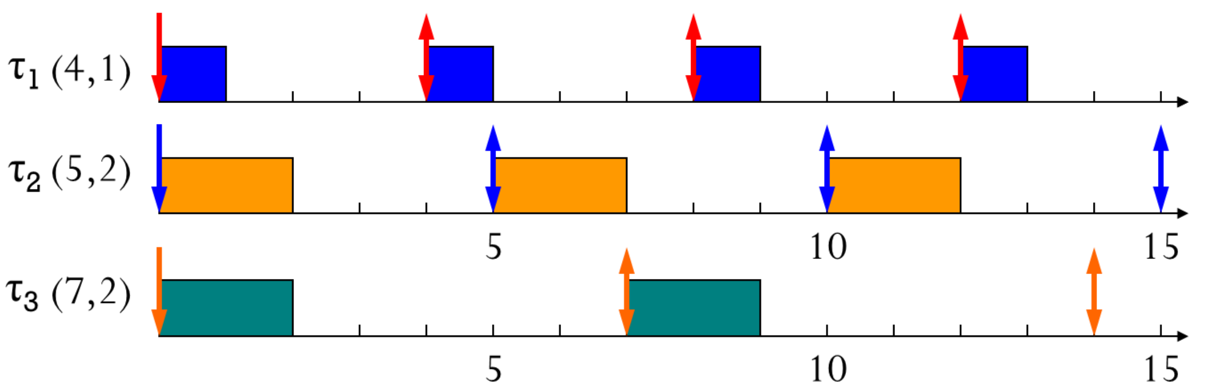
$\tau_2\ (5,3)$



Jinkyu Lee
Dept. of Computer Science and Engineering

Classification of Priority-Driven Scheduling

- Can we guarantee no deadline miss of all jobs of a given task set under each scheduling algorithm?
  - TFP: RM
  - JFP: EDF
  - JDP: LLF
- What is the maximum utilization?
  - $U = \sum C_i/T_i \leq 1.0$
    - Example: 1/4 + 2/5 + 2/7



- Can each scheduling algorithm achieve the maximum utilization?

Jinkyu Lee
Dept. of Computer Science and Engineering

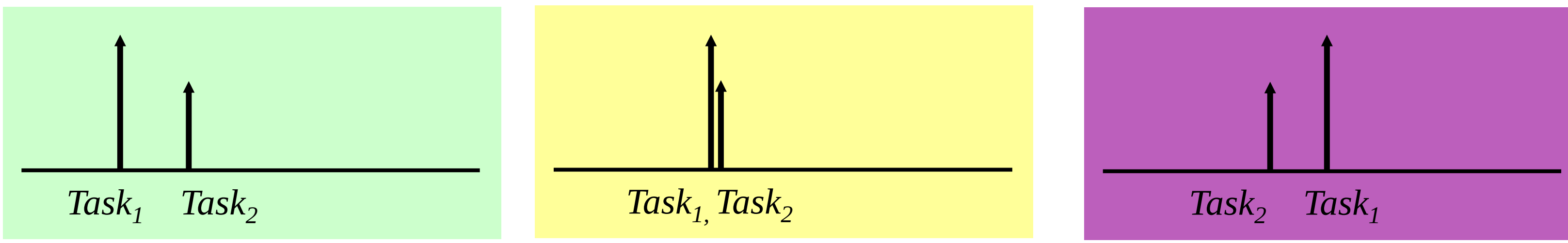
RM: Overview

- Critical instant under RM
- Utilization-based analysis
  - $U \le n(2^{1/n} - 1.0)$
    - $n \to \infty$, $n(2^{1/n} - 1.0) \to 0.69$
  - Only sufficient
- Response-time analysis
  - Exact
- Some applications
  - Schedulability with transient overhead, interrupts and blocking

Without loss of generality
$T_1 \le T_2 \le \ldots \le T_n$

RM: Critical Instant

- Critical instant of τ_i
  - A job of τ_i arriving at that instant has the largest response time
- Which τ_2 has the maximum response time?



- In-phase instant is critical under RM
  - All higher priority tasks are released at the same instant of $J_{i,c}$ (assume all jobs are completed before the next job is released)

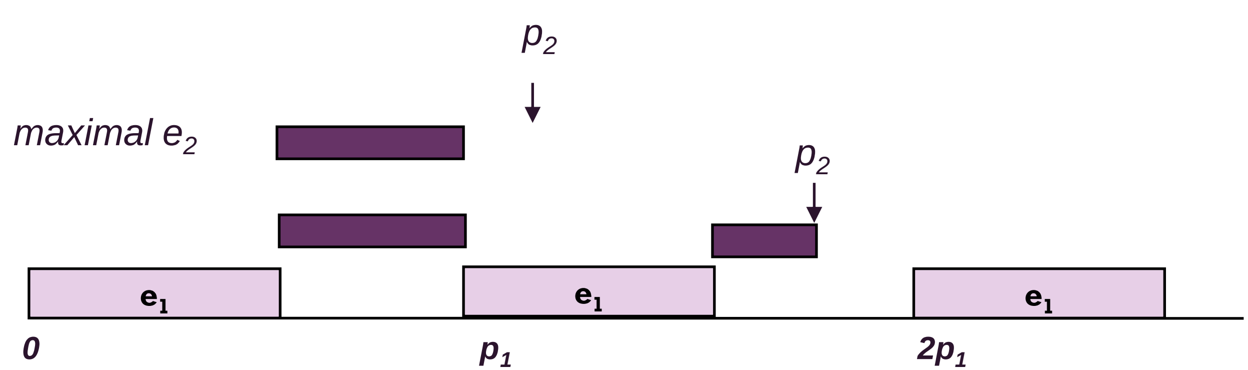
Why?

Jinkyu Lee
Dept. of Computer Science and Engineering

RM: Utilization-Based Analysis

Proof of $U \leq n(2^{1/n} - 1.0)$

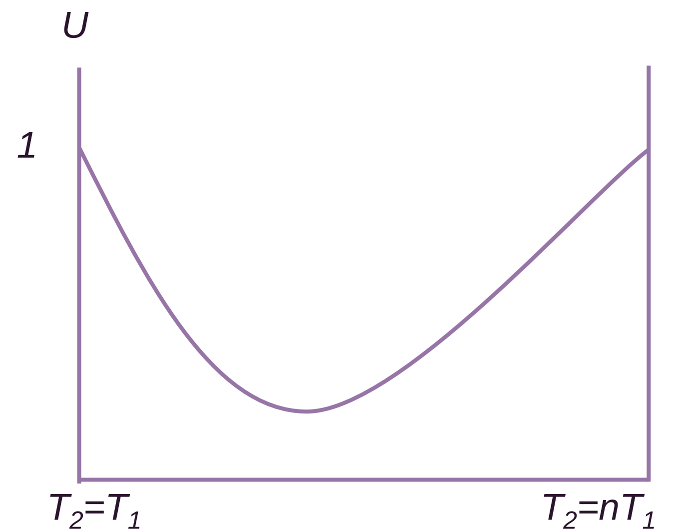
- Let's consider two tasks
  - $Task_2$ can only be executed when $Task_1$ is not in the system.
- Let $T_2 < 2T_1$. Determine the maximum schedulable C_2.
  - If $T_2 \leq T_1 + C_1$, $\max(C_2) = T_1 - C_1$
    - $U = C_1/T_1 + (T_1 - C_1)/T_2$
  - Else, $\max(C_2) = T_2 - 2C_1$
    - $U = C_1/T_1 + (T_2 - 2C_1)/T_2$



RM: Utilization-Based Analysis

- Given C_1, T_1, and T_2, plot U
- The minimum U occurs when
  - $T_2 = T_1+C_1$

  where $U = C_1/T_1+(T_1-C_1)/(T_1+C_1)$



- What is the minimum U?
  - Take the derivative w.r.t T_1 and set $dU/dT_1=0$
  - We will get $C_1=(2^{1/2}-1.0)T_1$ and $U=0.828$
- General cases with n

$$-C_1*T_1^{-2}-(T_1-C_1)*(T_1+C_1)^{-2} + (T_1+C_1)^{-1}=0$$

$$-C_1*T_1^{-2}+2C_1*(T_1+C_1)^{-2}=0$$

RM-schedulable if $U \leq n(2^{1/n} - 1.0)$

RM: Utilization-Based Analysis

- Example

	C_i	T_i	U_i
Task_1	20	100	0.200
Task_2	40	150	0.267
Task_3	100	350	0.286

- Total utilization is

$$.200 + .267 + .286 = .753 < U(3) = 3(2^{1/3} - 1.0) = .779$$

- The periodic tasks in the example are schedulable according to the UB test.

RM: Utilization-Based Analysis

- Example

	C_i	T_i	U_i
$Task_1$	40	100	0.400
$Task_2$	40	150	0.267
$Task_3$	100	350	0.286

- Total utilization is

$$.400 + .267 + .286 = .953 > U(3)=3(2^{1/3} - 1.0) = .779$$

- UB test cannot deem the task schedulable.
  - Try the exact analysis

RM: Response-Time Analysis

- Response time analysis
  - The duration between a job's release time and finishing time

Without loss of generality
$T_1 \le T_2 \le \ldots \le T_n$

$$a_{n+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_n}{T_j} \right\rceil C_j \quad \text{where} \quad a_0 = \sum_{j=1}^{i} C_j$$

A set of higher-priority tasks

  - Test terminates when $a_{n+1} = a_n$
  - Task *i* is schedulable if its response time is before its deadline: $a_n \le T_i$
    - a_n is the response time of Task_i
  - Can be applied to any TFP
    - *Not only RM, but also any TFP*

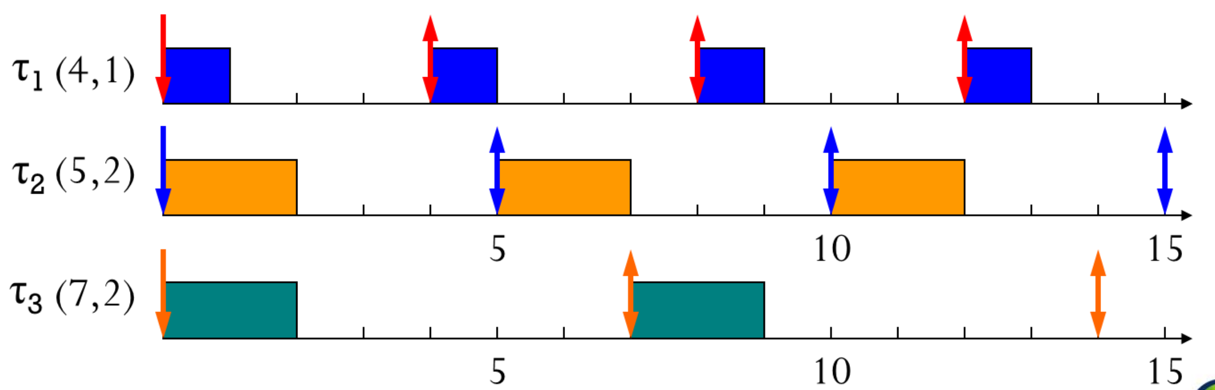
RM: Response-Time Analysis

Without loss of generality

$T_1 \leq T_2 \leq \ldots \leq T_n$

$$a_{n+1} = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_n}{T_j} \right\rceil C_j \quad \text{where} \quad a_0 = \sum_{j=1}^{i} C_j$$

A set of higher-priority tasks



Jinkyu Lee
Dept. of Computer Science and Engineering

성균관대학교
SUNG KYUN KWAN UNIVERSITY

RM: Response-Time Analysis

- How does this work? $Task_3$ is schedulable?

	C_i	T_i	U_i
$Task_1$	40	100	0.400
$Task_2$	40	150	0.267
$Task_3$	100	350	0.286

$$a_0 = \sum_{j=1}^{3} C_j = C_1 + C_2 + C_3 = 40 + 40 + 100 = 180$$

$$a_1 = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_0}{T_j} \right\rceil C_j = C_3 + \sum_{j=1}^{2} \left\lceil \frac{a_0}{T_j} \right\rceil C_j$$

$$= 100 + \left\lceil \frac{180}{100} \right\rceil (40) + \left\lceil \frac{180}{150} \right\rceil (40) = 100 + 80 + 80 = 260$$

Jinkyu Lee
Dept. of Computer Science and Engineering

RM: Response-Time Analysis

$$a_2 = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_1}{T_j} \right\rceil C_j = 100 + \left\lceil \frac{260}{100} \right\rceil (40) + \left\lceil \frac{260}{150} \right\rceil (40) = 300$$

$$a_3 = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{a_2}{T_j} \right\rceil C_j = 100 + \left\lceil \frac{300}{100} \right\rceil (40) + \left\lceil \frac{300}{150} \right\rceil (40) = 300$$

$a_3 = a_2 = 300$ Done!

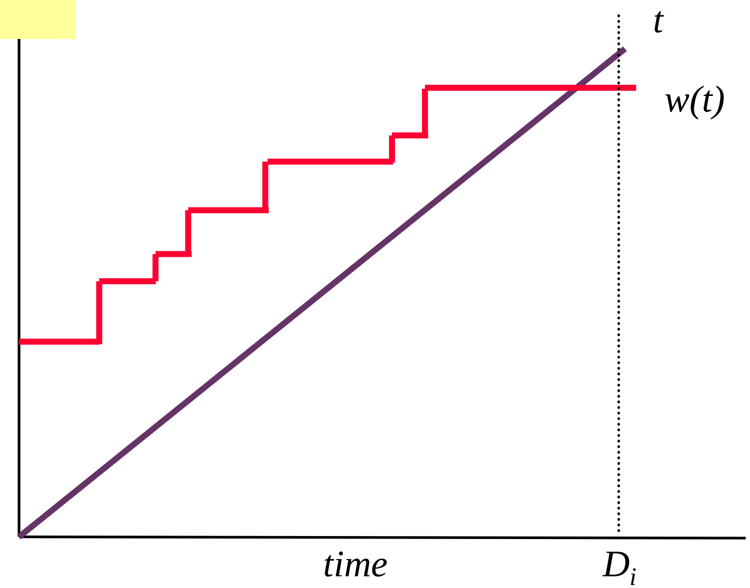
- Task is schedulable using RT test.
  - $a_3 = 300 < T_3 = 350$

RM: Response-Time Analysis

- Why does this work?
  - Time-demand analysis based on critical instant
  - If J_i is done at t, then the total work must be done in $[0,t]$ is (from J_i and all higher priority tasks): time demand function

$$w_i(t) = C_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{T_k} \right\rceil C_k$$

  - Can we find $t \leq D_i$ such that $w_i(t) \leq t$
    - Cannot check all $t \perp [0, d_i]$
    - Check all arrival instants and d_i



RM: Transient Overload

- Question: what if the task with a smaller period is not important to the underlying application?
- Answer: consider period transformation, period aggregation or period splitting
- Example: consider the following unschedulable task set

τ_i	C_i	EC_i (avg exec time)	T_i	comments
τ_1	20	10	100	
τ_2	30	25	150	
τ_3	80	40	210	Non-critical
τ_4	100	20	400	

RM: Transient Overload

- Solution 1: reduce $Task_3$'s priority by lengthening its period, possible only if $Task_3$'s relative deadline can be greater than its original period. In such a case, replace $Task_3$ by two tasks $Task_3^0$ and $Task_3^{00}$, each with period 420, WC exec times $C_3^0 = C_3^{00} = 80$, avg exec times $a_3^0 = a_3^{00} = 40$. $Task_3^0$ and $Task_3^{00}$ must be phased to be 210 time units apart. If the set $\{Task_1, Task_2, Task_3^0, Task_3^{00}, Task_4\}$ is RM-schedulable, done.

τ_i	C_i	EC_i (avg exec time)	T_i	comments
τ_1	20	10	100	
τ_2	30	25	150	
τ_3	80	40	210	Non-critical
τ_4	100	20	400	

RM: Transient overload

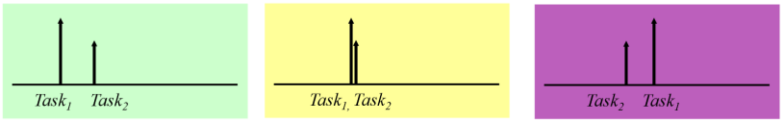
- Solution 2: increase $Task_4$'s priority by splitting each invocation into two: $Task_4^0$: $C_4^0 = C_4/2$, $a_4^0 = a_4/2$, $T_4^0 = T_4/2$
  - {$Task_1$, $Task_2$, $Task_4$} or {$Task_1$, $Task_2$, $Task_4^0$} are schedulable

τ_i	C_i	EC_i (avg exec time)	T_i	comments
τ_1	20	10	100	
τ_2	30	25	150	
τ_3	80	40	210	Non-critical
τ_4	100	20	400	

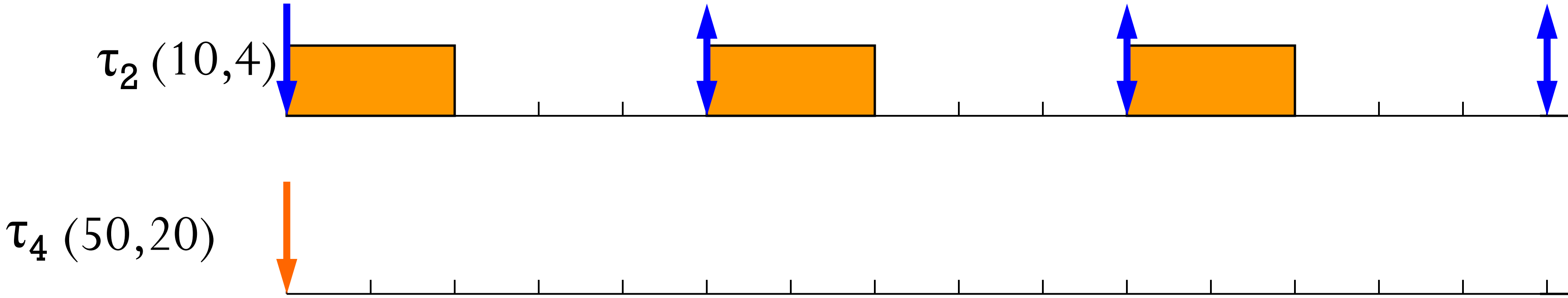
EDF: Schedulability Analysis

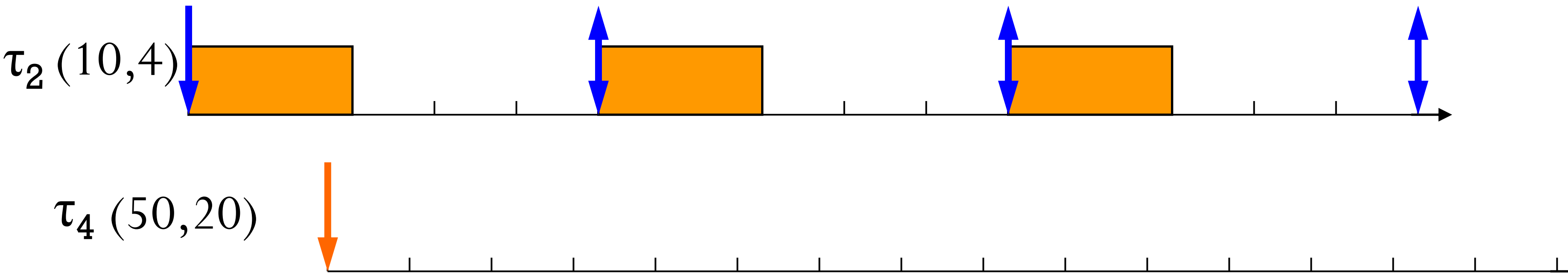
- RM's critical instants hold for EDF?
  - Why or why not?

- Critical instant of τ_i
  - A job of τ_i arriving at that instant has the largest response time
- Which τ_2 has the maximum response time?



- In-phase instant is critical under RM
  - All higher priority tasks are released at the same instant of $J_{i,c}$ (assume all jobs are completed before the next job is released)

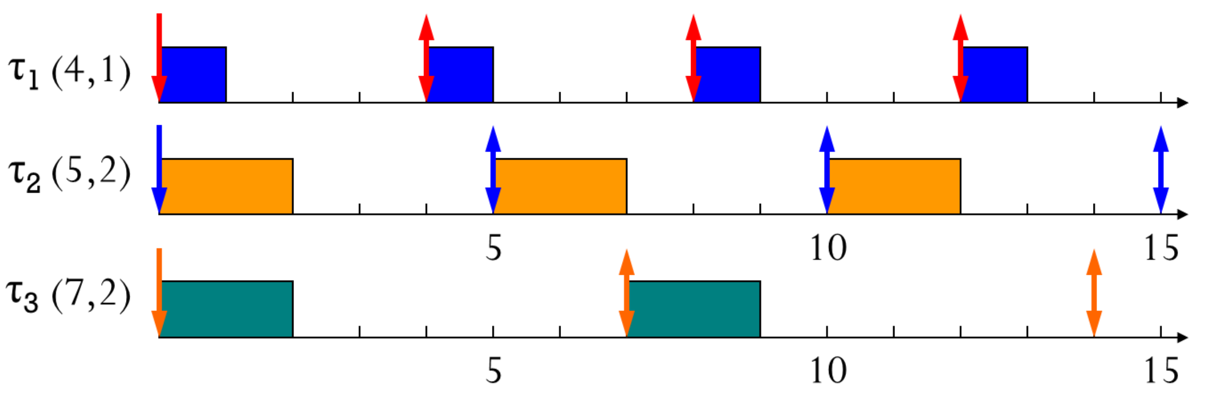




Jinkyu Lee
Dept. of Computer Science and Engineering

EDF: Schedulability Analysis

- EDF is schedulable iff U ≤ 1.0
  - U = 1/4 + 2/5 + 2/7
- How to prove?
  - Using busy interval



Jinkyu Lee
Dept. of Computer Science and Engineering

EDF: Schedulability Analysis

- Example
  - A digital robot with EDF schedule
    - Control loop: $C_c \leq 8$ms at 100Hz
    - BIST (Built-in-Self-Testing): $C_b \leq 50$ms
    - BIST can be done every 250ms
  - Add a telemetry task to send and receive messages with $C_t \leq 15$ms
    - If BIST is done every 1000ms
    - The telemetry task can have a relative deadline of 100ms
      - Sending or receiving must be separated by at least 100ms

$$u_c + u_b = \frac{8}{10} + \frac{50}{T_b} \leq 1$$

$$u_c + u_b + ut = \frac{8}{10} + \frac{50}{1000} + \frac{15}{D_t} \leq 1$$

Summary: $D_i = T_i$

- TFP: RM
  - $U \le n(2^{1/n} - 1.0)$
  - Response-time analysis (exact)
- JFP: EDF
  - $U \le 1$ (exact)
- JDP: LLF
  - $U \le 1$ (exact)
- The three algorithms are optimal for each category.
- All the results hold for sporadic tasks.

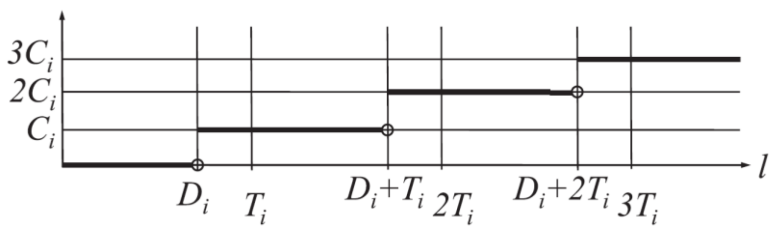
Extension: $D_i = T_i \rightarrow D_i \leq T_i$

- TFP: DM (Deadline Monotonic)
  - $\sum C_i/D_i \leq n(2^{1/n} - 1.0)$
  - Response-time analysis: $a_n \leq D_i$, still exact
- JFP: EDF
  - $\sum C_i/D_i \leq 1$
    - Not exact, but there exists another exact analysis.
- JDP: LLF
  - $\sum C_i/D_i \leq 1$
- The three algorithms are still optimal for each category.
- All the results hold for sporadic tasks.

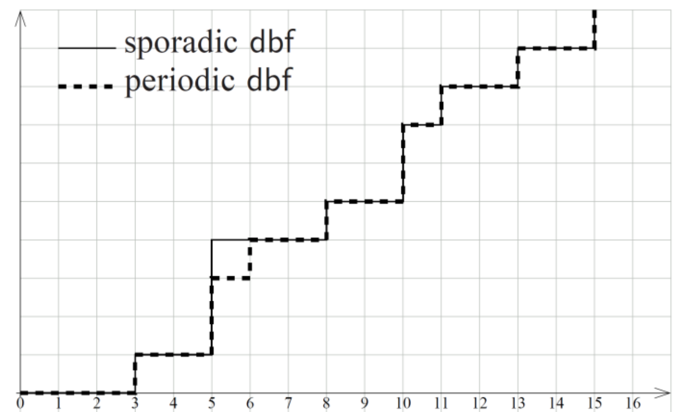
Extension: $D_i = T_i \rightarrow D_i \leq T_i$

- JFP: EDF
  - Any exact analysis?

$$\mathsf{DBF}_{\mathsf{EDF}}(\tau_i, l) \triangleq \left(\left\lfloor \frac{l - D_i}{T_i} \right\rfloor + 1 \right) \cdot C_i.$$



(a) $\mathsf{DBF}_{\mathsf{EDF}}(\tau_i, l)$



Extension: $D_i = T_i \rightarrow D_i \leq T_i$

- JFP: EDF
  - Any exact analysis?
    - $\sum$ DBF($Task_i$, t) $\leq$ t, for all t>0

$$\mathsf{DBF}_{\mathsf{EDF}}(\tau_i, l) \triangleq \left(\left\lfloor \frac{l - D_i}{T_i} \right\rfloor + 1 \right) \cdot C_i.$$

  - Proof
    - Busy interval

Where are we?

VxWorks, Xen, RT Linux

Practice

Theory

Preemption cost analysis

$D_i \leq T_i$

Dependent task model, e.g., resource sharing

- A single processor
- Hard deadline
- Independent periodic tasks
- $d_i = p_i$
- Preemptable without any limit and overhead for context switch

Non-preemptive scheduling

Hierarchical model (virtualization)

Soft deadline

How to Apply This to a Target System?

- Check
  - Task model
    - Any dependency?
    - Any shared resources?
    - (T, C, D) is given?
  - Platform
  - Constraints