시스템 최신기술
(차량 제어 시스템 Week 2)

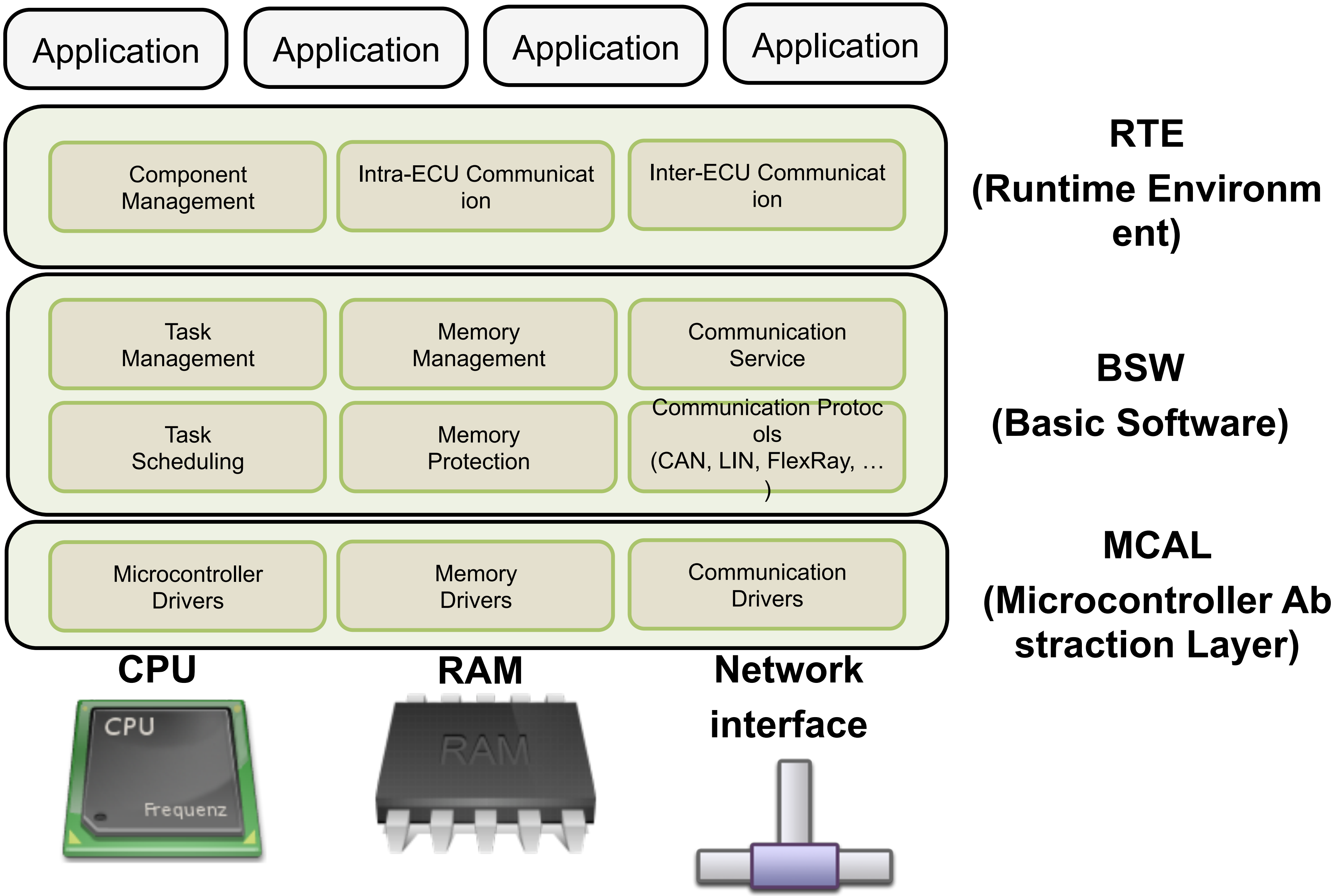
Prof. Jong-Chan Kim
Graduate School of Automotive Engineering
Kookmin University

Traditional Super Loop Method

- Create a single infinitely executing loop
- Do everything inside the loop

```
main()
{
        system_init();
        while(1) {
                do_A();
                do_B();
                do_C();
                …
        }
}
```

Automotive Control Software Architecture



MCAL & BSW

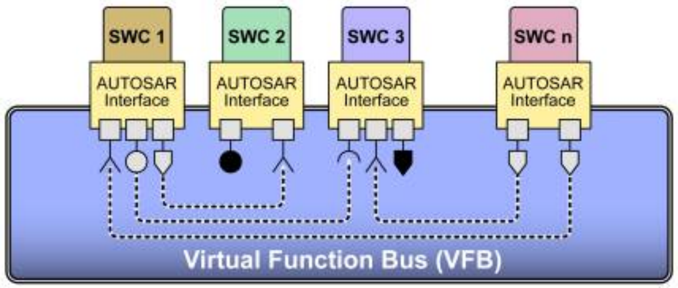
- RTOS
  - OS intended to serve real-time application requests - Wikipedia
- OSEK RTOS
  - Standard RTOS in the automotive industry
  - Not a product, just a specification
  - OSEK/VDX specification [LINK]

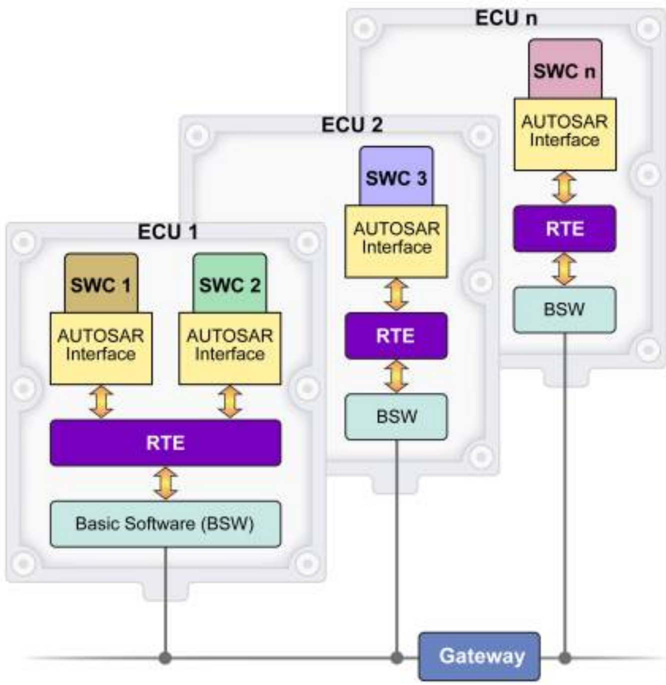
:= MCAL + BSW

RTE

- 단일 ECU내 혹은 ECU간 통신에 대해서 통합된 Communication Interface 제공
- SWC의 배치에 대한 freedom 제공

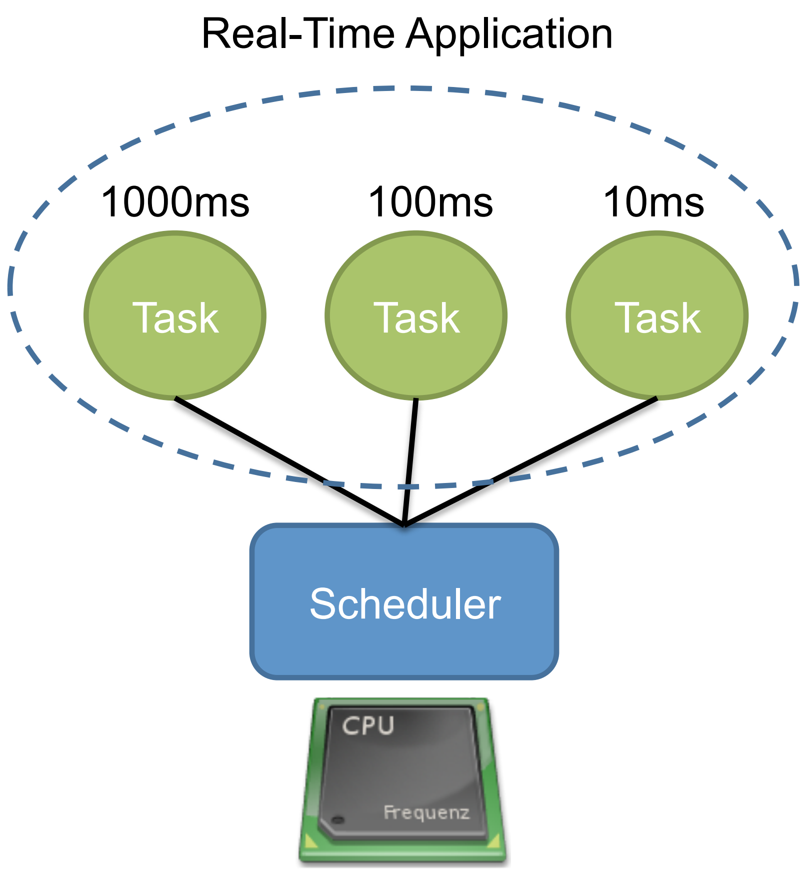
Design **Implementation**





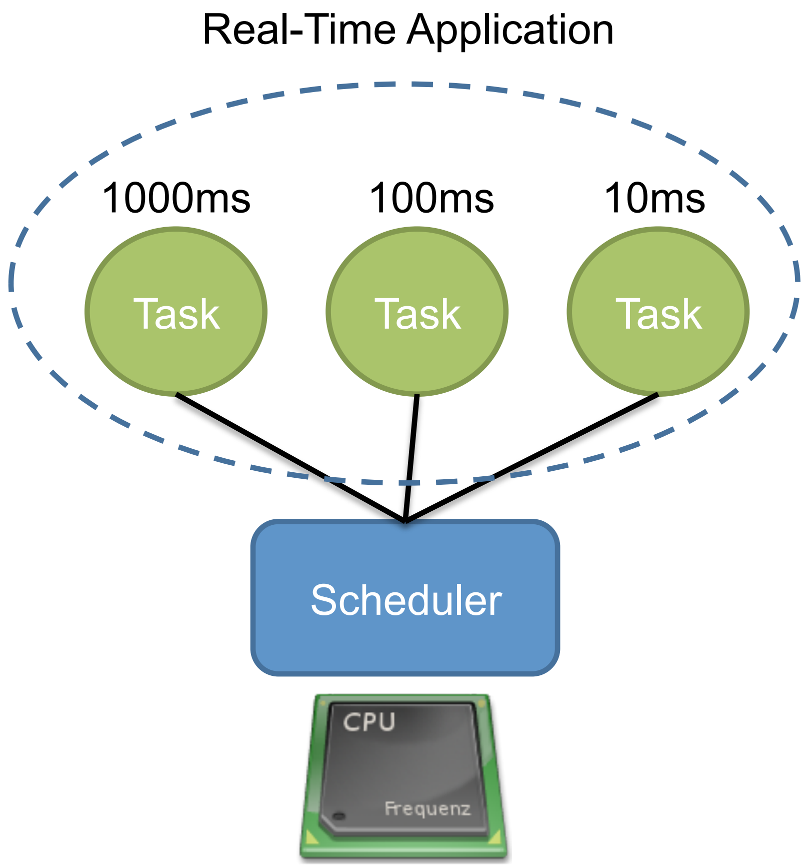
RTOS

- RTOS is in charge of guaranteeing the given real-time tasks' **deterministic** temporal behavior
- Task
  - Usually periodically invoked
  - Has a priority value
    - Fixed-priority scheduling
- Scheduler
  - Decides which task to run



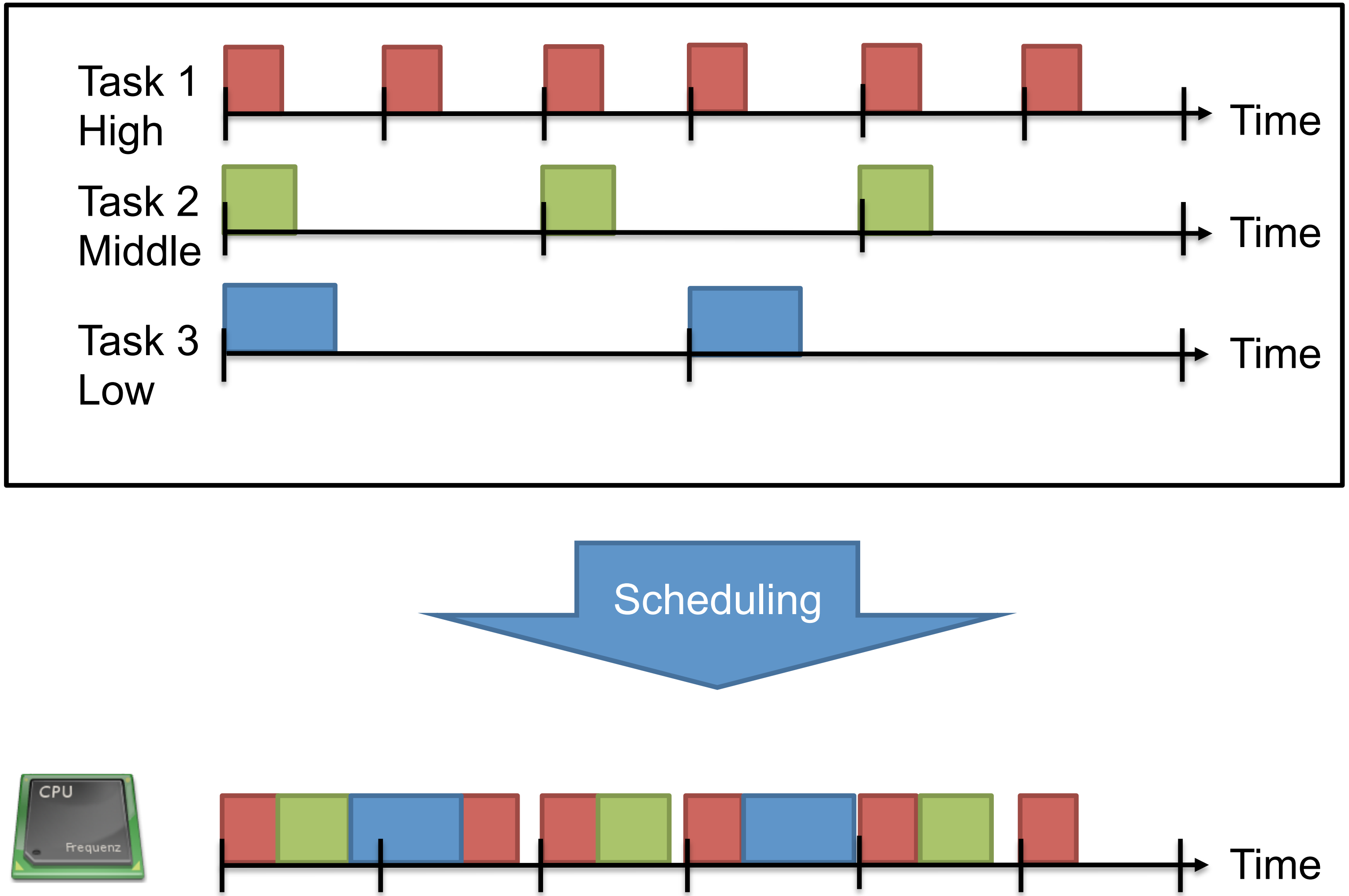
RTOS

```
DECLARE_TASK(task1, 1000, 1);
DECLARE_TASK(task2, 100, 2);
DECLARE_TASK(task3, 10, 3);
task1()
{
    …
}
task2()
{
    …
}
task3()
{
    …
}
main()
{
    init(task1);
    init(task2);
    init(task3);
    begin_os();
}
```



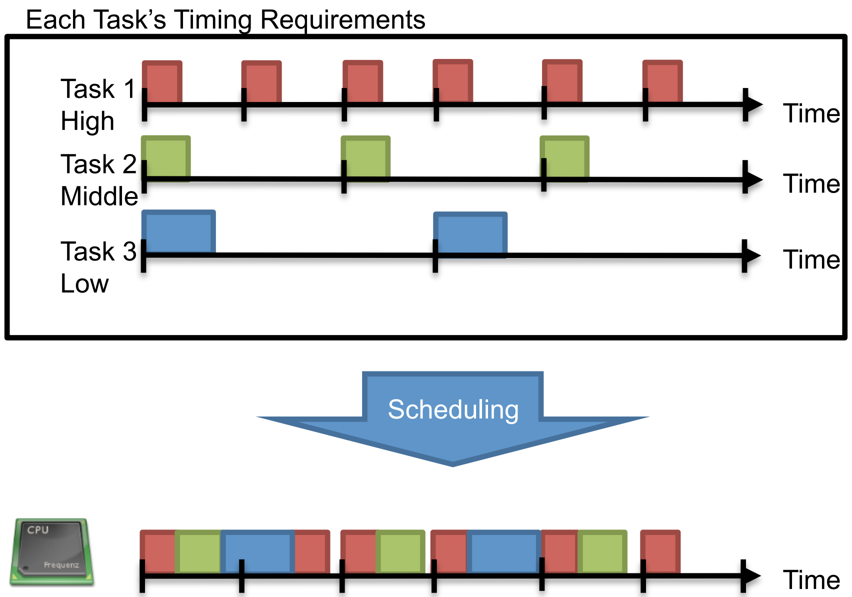
(Fixed-priority) Scheduling

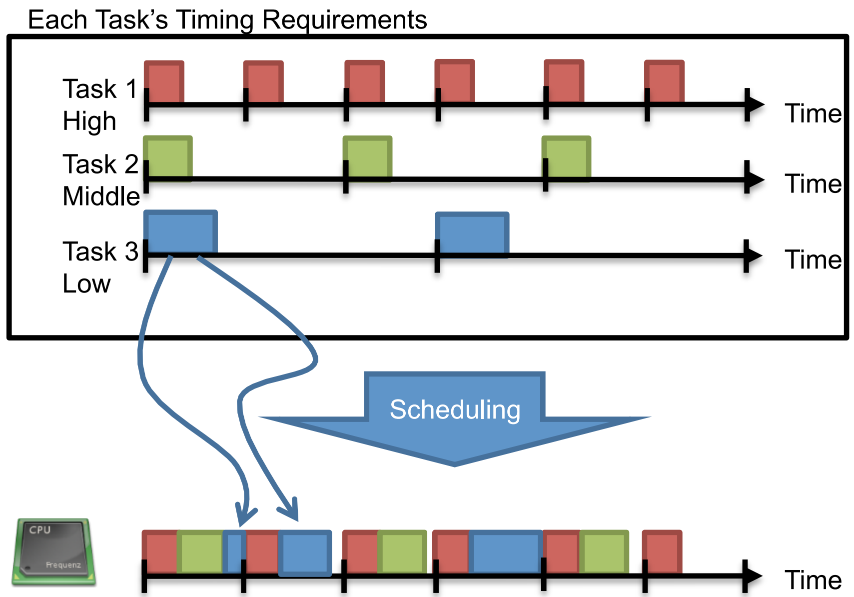
Each Task's Timing Requirements



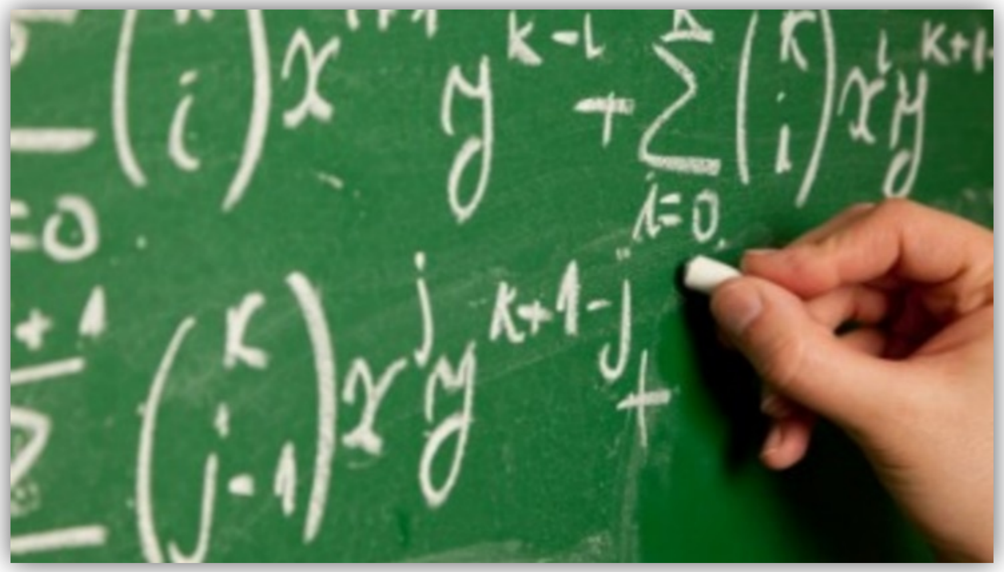
Scheduling

- Preemptive Scheduling
  - You can arbitrarily suspend the currently executing task
- Non-preemptive Scheduling
  - You have to wait for the current task to finish before starting a new task
- Cooperative Scheduling
  - Preemption allowed only at pre-defined time points





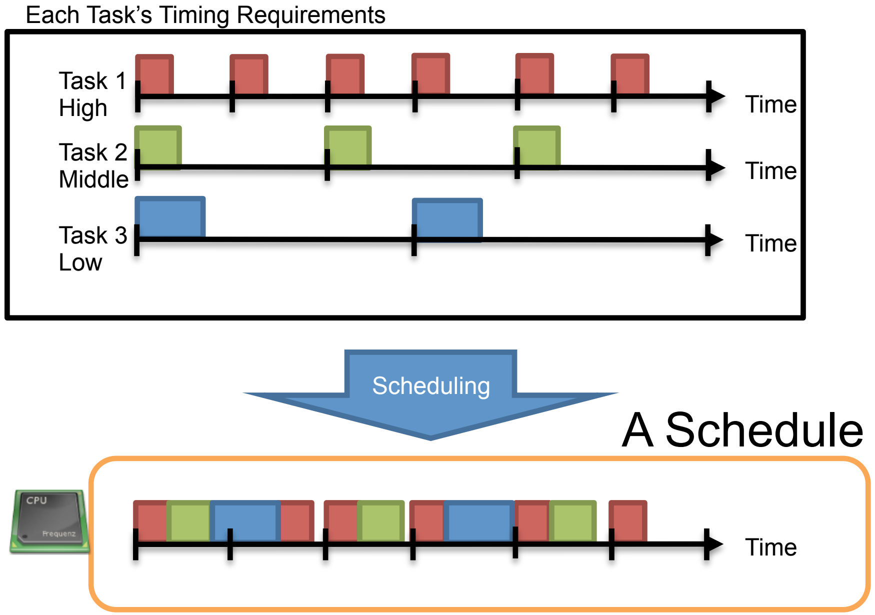
Scheduling Algorithm



Scheduling algorithm:
deploy work on computing resources in the time domain

Scheduling Algorithm

- Schedule
  - Assignment of jobs to the resource in the time domain



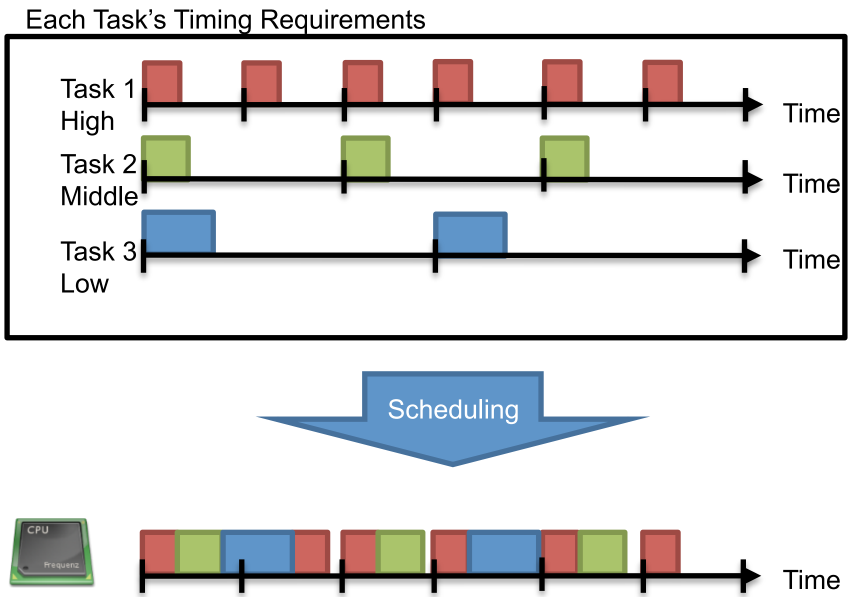
- Scheduler
  - A module that implements the scheduling algorithm

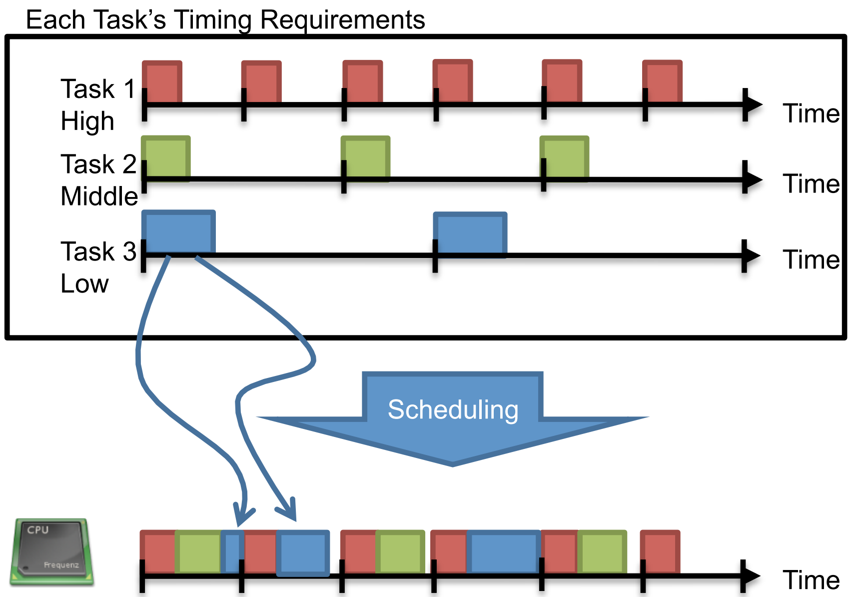
Scheduling Power

- Feasibility
  - A schedule is said to be **feasible** if all the tasks can be completed before their deadlines
- Schedulability
  - A set of tasks is said to be **schedulable** if there exists at least one algorithm that can produce a feasible schedule for the task set
  - A set of tasks is said to be **schedulable by a scheduling algorithm** if the scheduling algorithm produces a feasible schedule for the task set

Preemptive vs Non-preemptive

- Preemptive Scheduling
  - You can arbitrarily suspend the currently executing task
- Non-preemptive Scheduling
  - You have to wait for the current task to finish before starting a new task
- Cooperative Scheduling
  - Preemption allowed only at pre-defined time points





Event-triggered vs Time-triggered

- Event-triggered scheduling
  - At each event such as job release, job completion, scheduler is invoked to pick the next job to execute
  - On-line scheduling
- Time-triggered scheduling
  - Everything is fixed in the timeline before the system starts
  - Off-line scheduling

Our Assumed Workload

- We have a set of *N* periodic tasks
  - $\{T_1, T_2, \ldots, T_N\}$
- Each T_i is generally represented by a three-tuple (p_i, e_i, d_i)
  - p_i: period
  - e_i : WCET (Worst-Case Execution Time)
  - d_i: **hard** relative deadline
- When T_i is an implicit-deadline task
  - (p_i, e_i)

Our Assumed Resource

- For now, we assume a singlecore CPU
  - Active resource with speed notion
  - Shared by multiple periodic tasks
  - Preemptable
    - Most CPU provides context restore/save mechanisms to support preemptive scheduling

Scheduling for what?

- Non-real-time scheduling
  - Maximize average **performance**
    - Minimize response time
    - Maximize throughput
  - Ensure fairness
    - No starvation
- Real-time scheduling
  - Ensure **predictability**
    - Meet every job's deadline (HARD)
    - Maintain deadline meet ratio under a threshold (SOFT)

Real-time performance

Which one is better?

For a periodic task T = (100ms, 5ms)

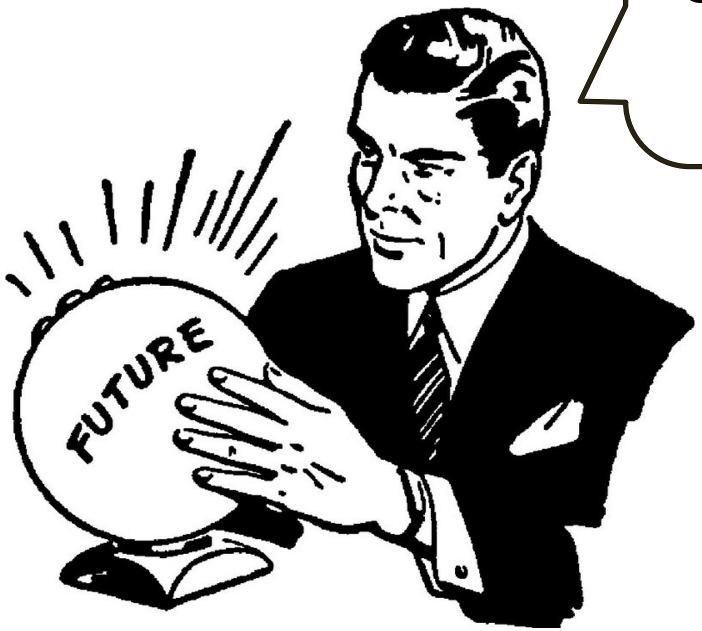
(1) Complete almost every job in 10ms

(2) Complete every job in 99ms

What is your answer? And why?

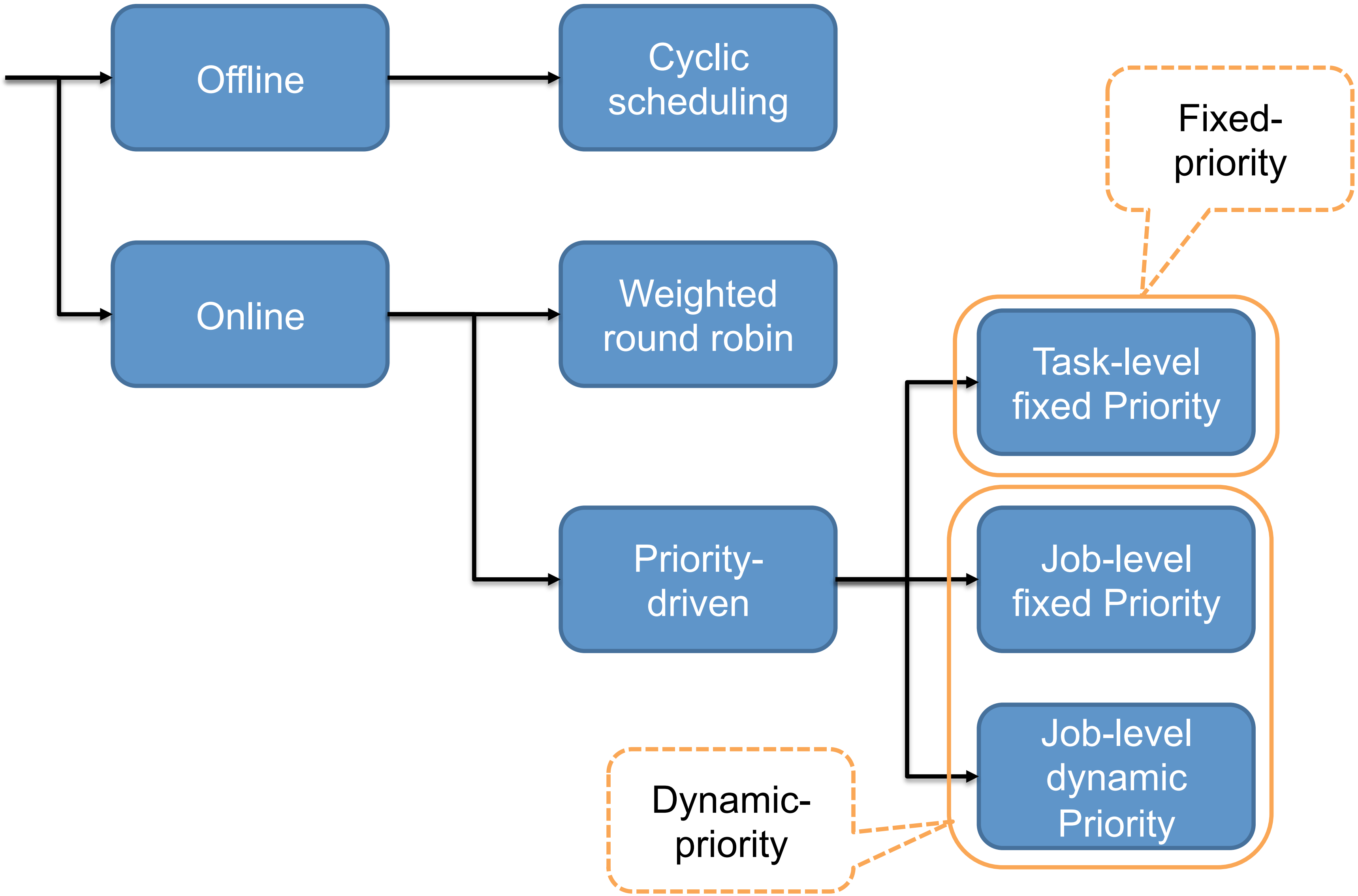
How can we know the future?

Periodic task m
odel plays as a
crystal ball



We know the future arrival pattern of each periodic task.
In that sense, we are **clairvoyant**.

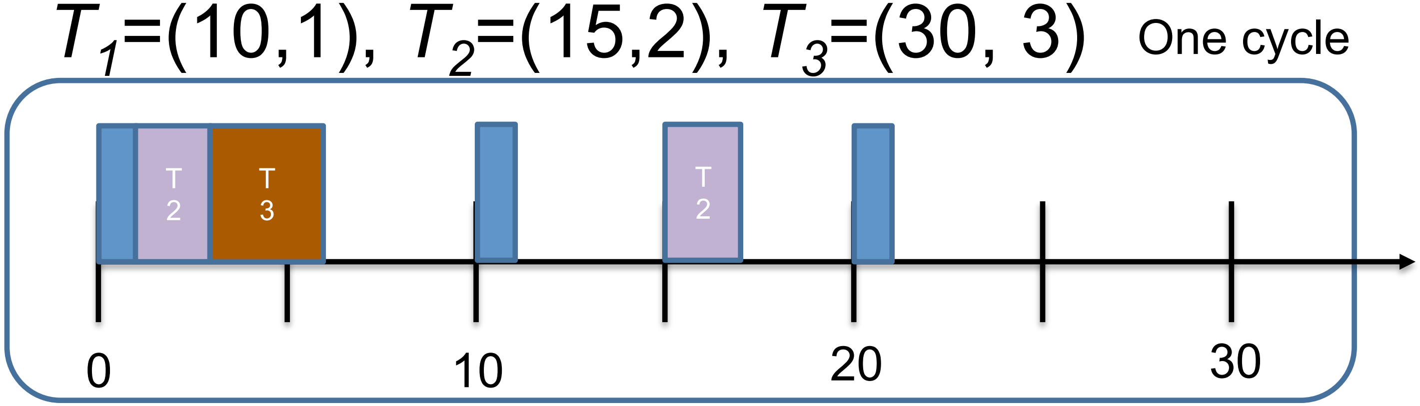
Typology of Real-Time Scheduling Algorithms



Cyclic Scheduling (1/2)

- Cyclic scheduling
  - Build a table listing jobs and their start times
  - Each job is executed in a non-preemptive way
- Cyclic executive
  - A simple SW module that dispatches jobs according to a given cyclic schedule
- Example

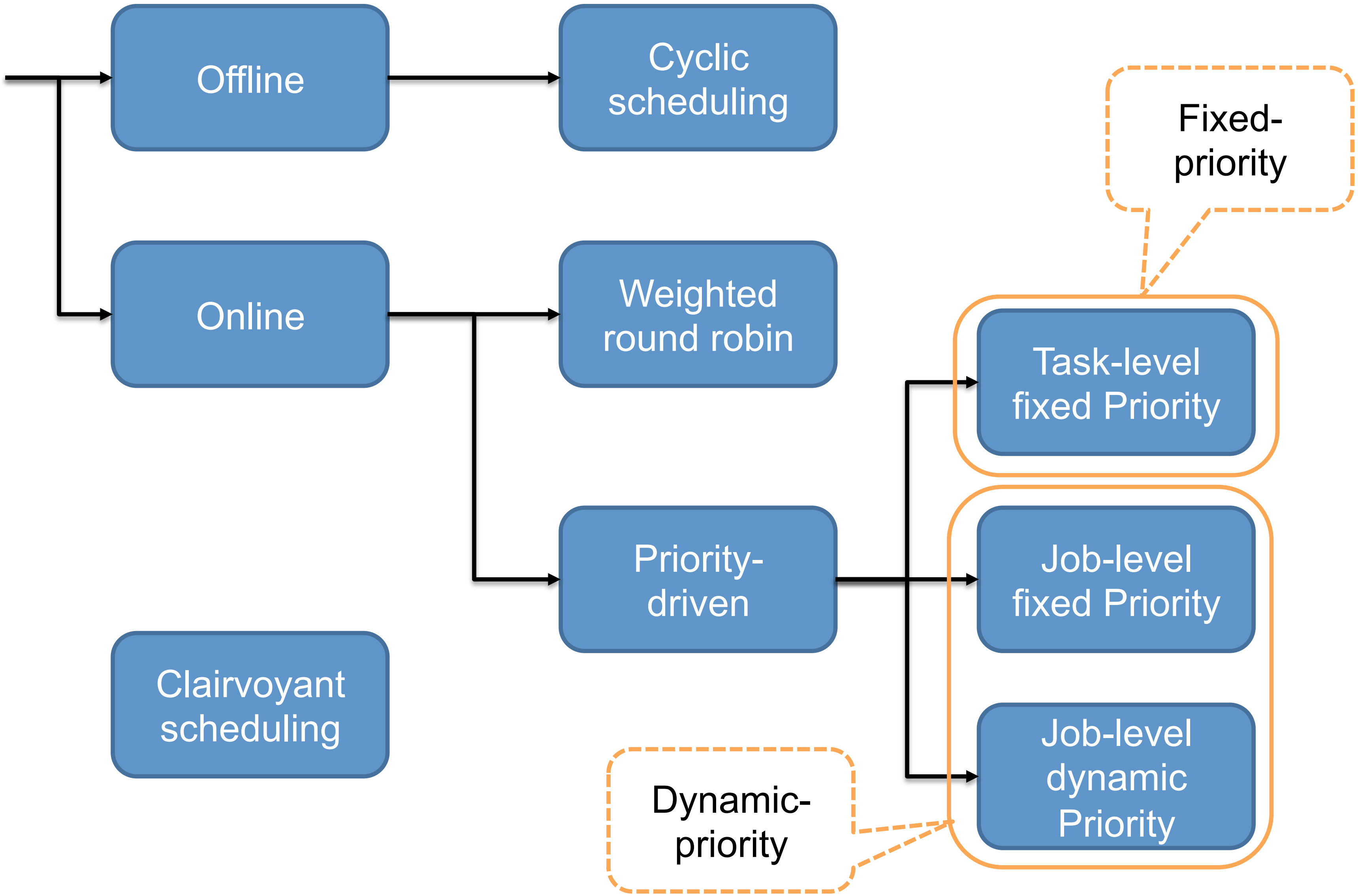
T_1=(10,1), T_2=(15,2), T_3=(30, 3)



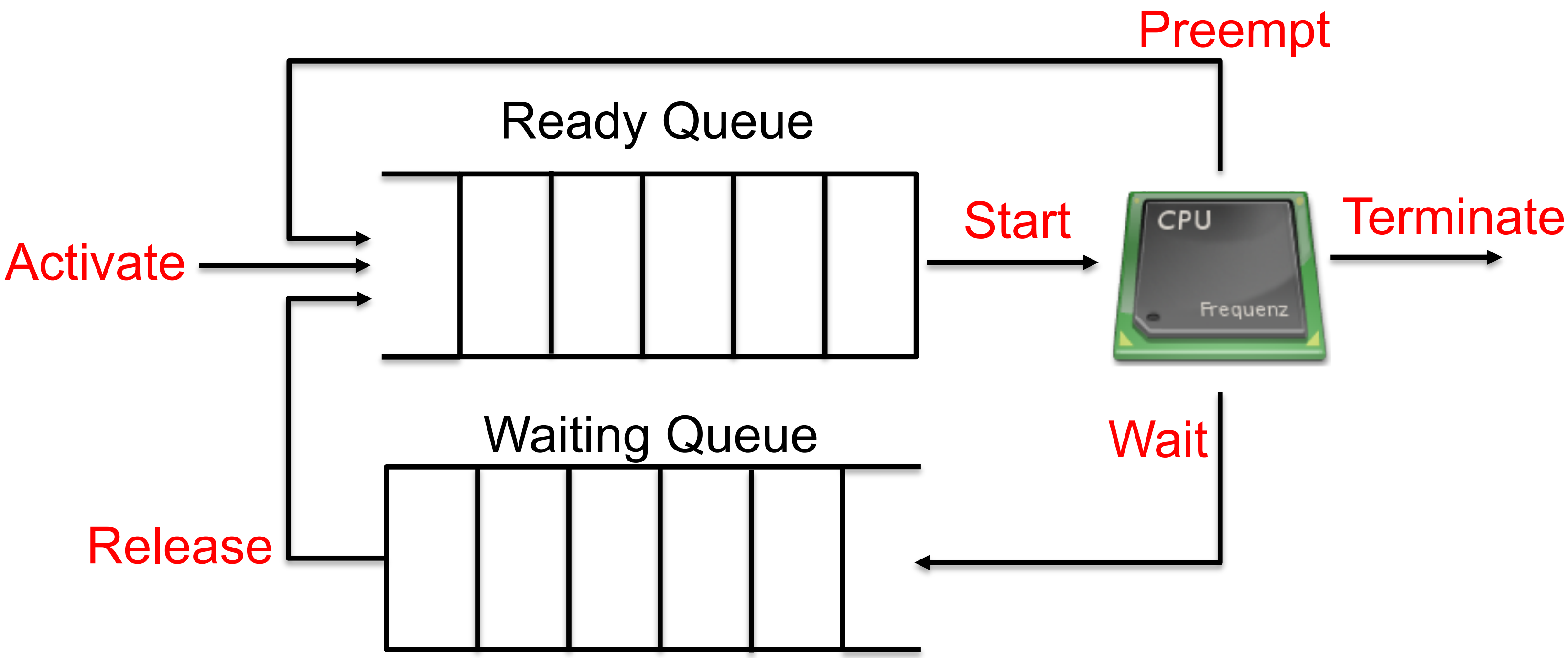
Cyclic Scheduling (2/2)

- Advantages
  - Simplicity: easy to implement
  - Predictability: system becomes totally deterministic
  - Small context switching overhead
- Disadvantages
  - Difficult: difficult to schedule
    - Building a table for 1,000 periodic tasks
    - How to guarantee it's optimal?
  - Not flexible to workload changes

Typology of Real-Time Scheduling Algorithms

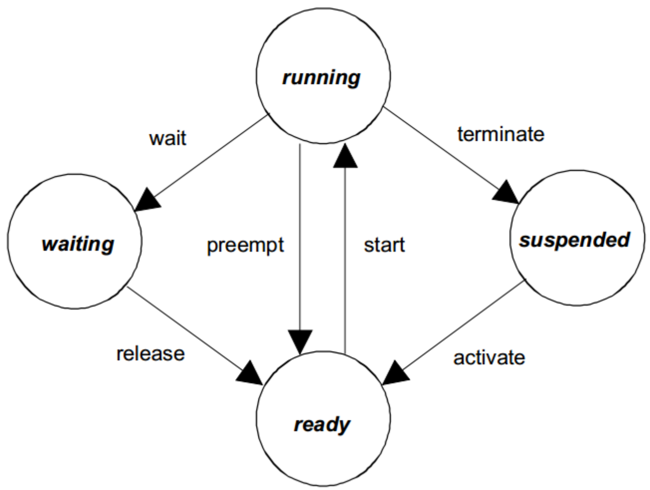


Online Scheduler (Task-level)

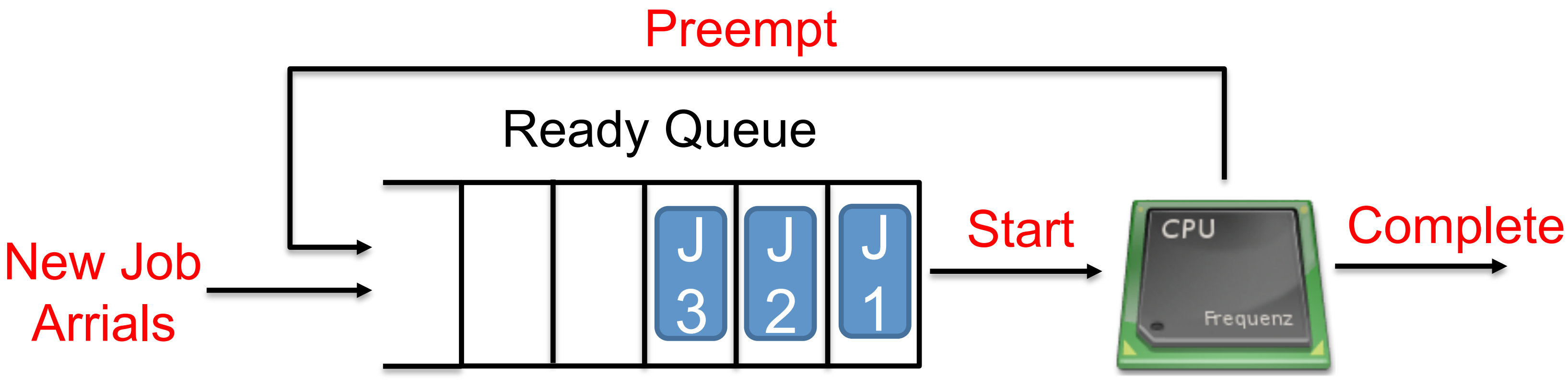


- Ready queue
  - Tasks ready to execute
- Waiting queue
  - Tasks waiting for certain events
    - Ex) I/O completion, periodic timer, …

OSEK Task Model



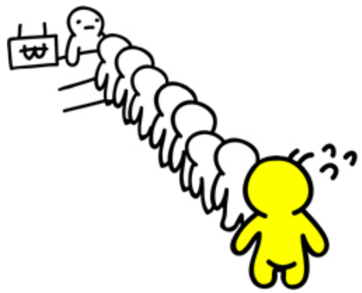
Online Scheduler (Job-level Simplified)

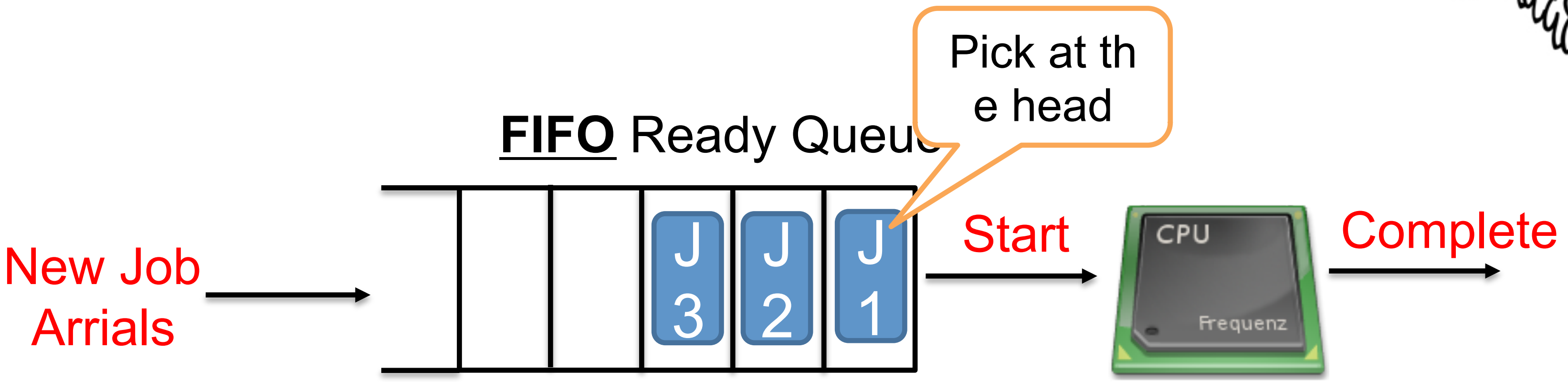


- Simply assume that every job never waits for something
  - No I/O
- Let's focus on
  - How to manage the ready queue
  - Which of the jobs to pick to run

FIFO (First-In, First-Out) Scheduling

- Also, First-Come, First-Served (FCFS)
- Start to complete (No preemption)





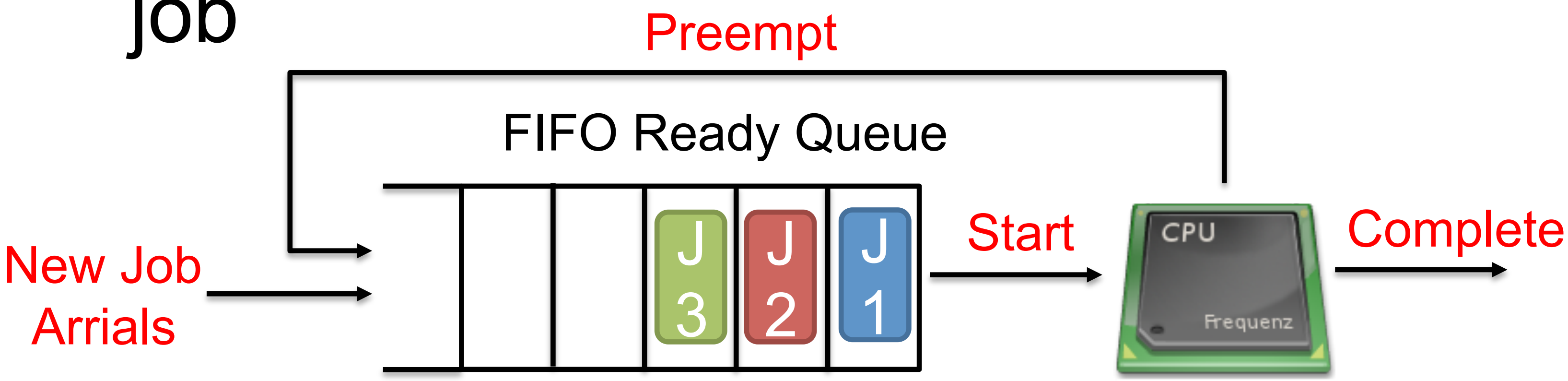
- Ready Queue
  – Single FIFO queue
- Job Dispatching
  – Pick the job at the head of the ready queue

Weighted Round-Robin Scheduling

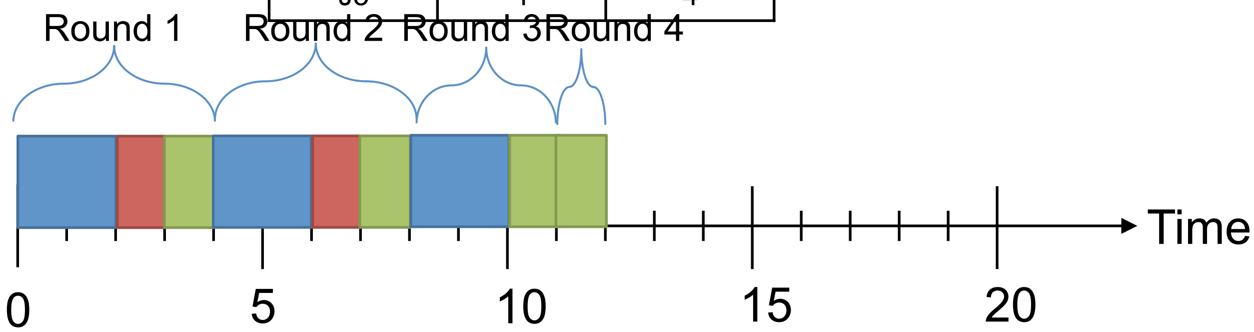
- More popular in time-sharing systems like UNIX or MS Windows
- Round-robin scheduling
  - Pick a job at the head of the FIFO ready queue
  - Execute the job only for a small unit time (time slice or time quantum)
  - If it does not finish within a time slice, put the job at the tail of the ready queue

Weighted Round-Robin Scheduling

- Different time slice size (weight) for each job

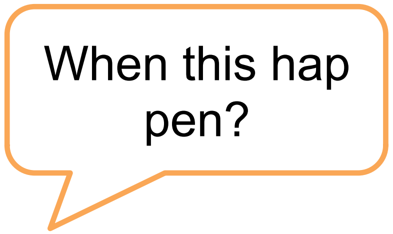


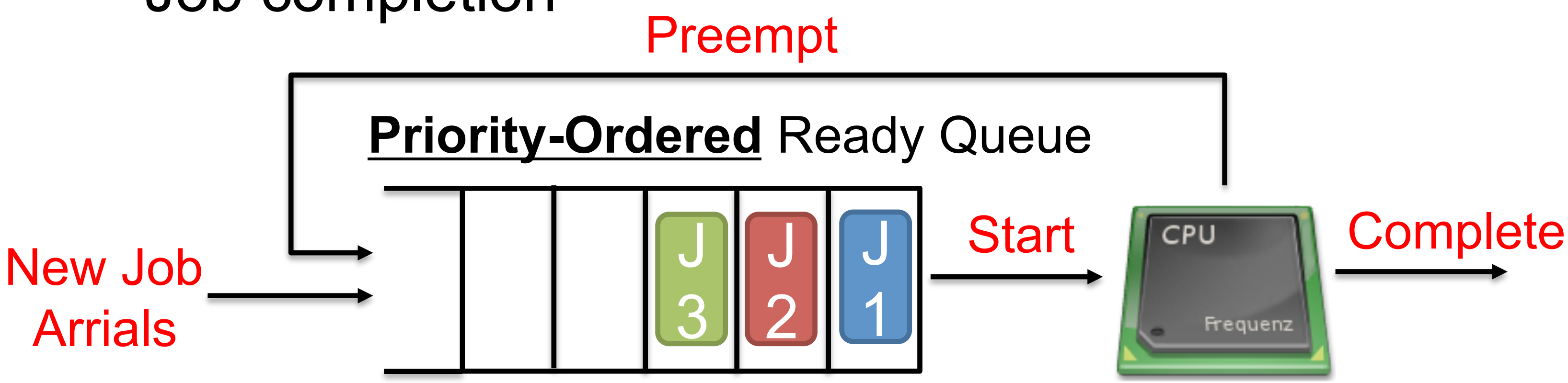
	Weight	Exe Time
J1	2	6
J2	1	2
J3	1	4



Priority-Driven Scheduling

- Each job has a priority
- Ready queue is ordered to the priorities (highest priority at the head)
- Always pick the highest-priority job
- Scheduling decisions are made on events such as
  - Job release
  - Job completion





Priority-Driven Scheduling

- Task-level Fixed Priority
  - RM (Rate Monotonic), DM (Deadline Monotonic), …

Which almost every RTOS is based on

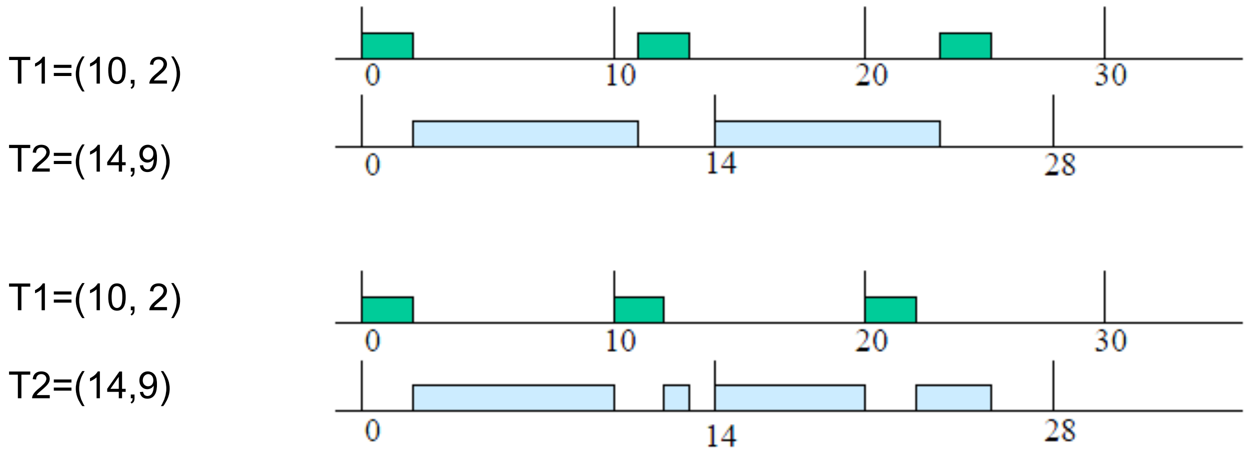
- Job-level Fixed Priority
  - EDF (Earliest Deadline First), LLF (Least Laxity First), FIFO, …
- Job-level Dynamic Priority
  - LST (Least Slack time First), …

In priority-driven scheduling, our focus is how to a ssign priorities to tasks or jobs at certain times

RM vs EDF

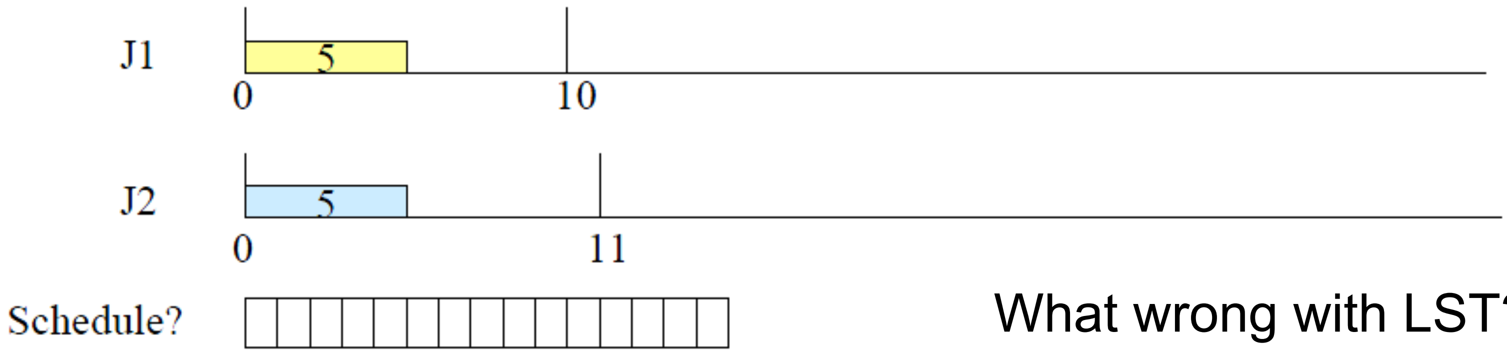
- RM (Rate Monotonic)
  - TFP
  - The higher rate, the higher priority
- EDF (Earliest Deadline First)
  - JFP
  - The shorter deadline, the higher priority

Which one is which?



LST (Least Slack Time)

- Slack time
  - The distance bet’n the deadline and the job completion time, assuming that the job get the processor
  - (d – t) – c
    - d: absolute deadline
    - t: current time
    - c: remaining execution time
  - When a job is executing, its slack ____________



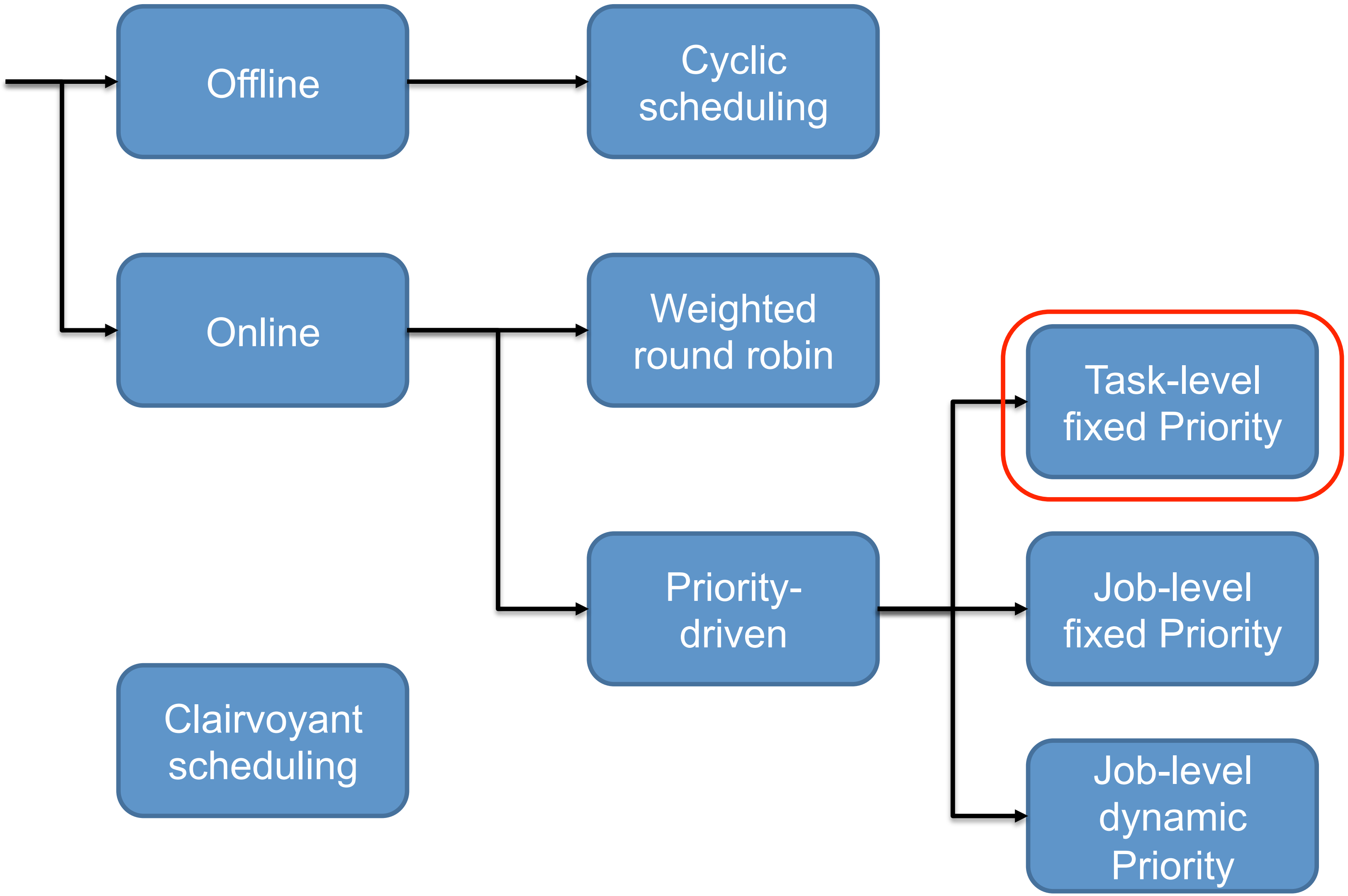
What wrong with LST?

Optimality

- A scheduling algorithm S is optimal under some given condition, if any algorithm can schedule a set of tasks, so c

- RM is an optimal TFP algorithm
- EDF is an optimal JFP algorithm

Typology of Real-Time Scheduling Algorithms



Note that

- Our workload
  - Periodic tasks
    - Each task is denoted by (p_i, e_i, d_i) or (p_i, e_i)
  - Tasks are independent
  - No aperiodic or sporadic task
- Our resource
  - Singlecore CPU
  - Preemptable at any time
  - Context switching overhead is negligible

Priority-Driven Scheduling

- At any time *t*, its highest priority ready job is given the CPU
- As a result, priority-driven scheduling is **work-conserving**
  - If there is any ready job, we cannot idle the CPU
- Scheduling decisions are made upon job releases and completions
  - What about WRR?

Fixed-Priority Scheduling

- Each task is associated with a fixed priority
- Due to its simplicity, most RTOSes support fixed-priority scheduling

OSTaskCreate()

```
INT8U OSTaskCreate(void   (*task)(void *pd),
                   void    *pdata,
                   OS_STK *ptos,
                   INT8U    prio);
```

Chapter	*File*	*Called from*	*Code enab*
4	OS_TASK.C	Task or startup code	OS_TASK_CR

OSTaskCreate() creates a task so it can be managed by _C/OS-II. Tasks can be cr
start of multitasking or by a running task. A task cannot be created by an ISR. A tas
infinite loop, as shown below, and must not return.

OSTaskCreate() is used for backward compatibility with µC/OS and when
OSTaskCreateExt() are not needed.

Depending on how the stack frame is built, your task has interrupts either enabled
check with the processor-specific code for details.

uCOS-II

VxWorks

taskInit()

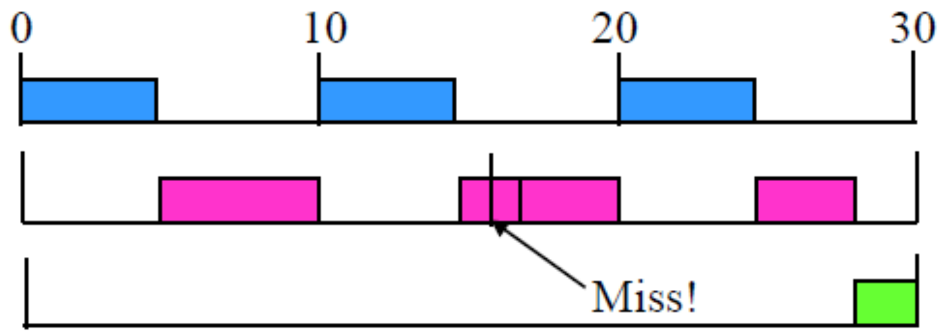
NAME *taskInit()* – initialize a task with a stack at a specified address

SYNOPSIS

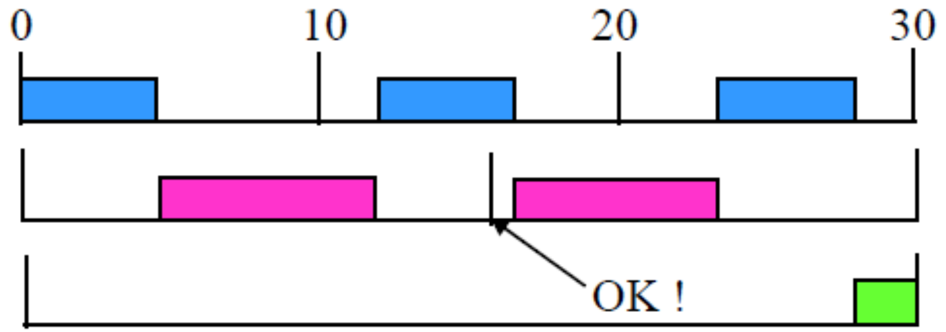
```
STATUS taskInit
    (
    WIND_TCB *  pTcb,        /* address of new task's TCB */
    char *      name,        /* name of new task (stored at pStackBase) */
    int         priority,    /* priority of new task */
    int         options,     /* task option word */
    char *      pStackBase,  /* base of new task's stack */
    int         stackSize,   /* size (bytes) of stack needed */
    FUNCPTR     entryPt,     /* entry point of new task */
    int         arg1,        /* first of ten task args to pass to func */
    int         arg2,
    int         arg3,
    int         arg4,
    int         arg5,
    int         arg6,
    int         arg7,
```

Fixed Priority vs Dynamic Priority

- $\{T_1=(p_1=10, e_1=4), T_2=(p_2=15, e_2=8), T_3=(p_3=30, e_3=2)\}$



Fixed Priority Schedule (RM)

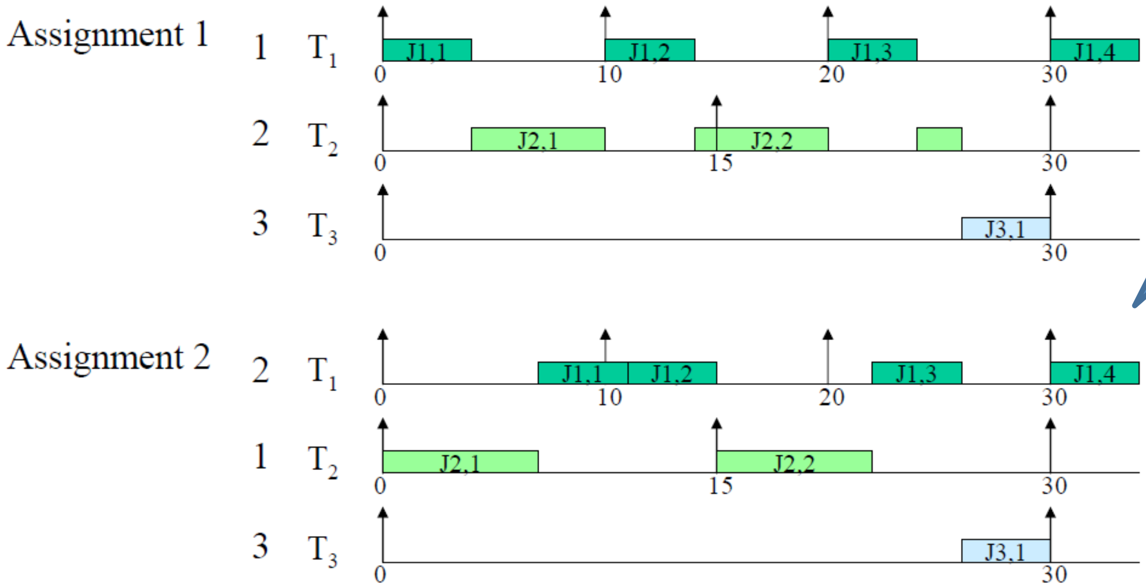


Dynamic Priority Schedule (EDF)

Fixed-Priority Scheduling

- Fundamental Problems
  - How to assign priorities to the given tasks?
  - How to analyze the schedulability of the tasks?

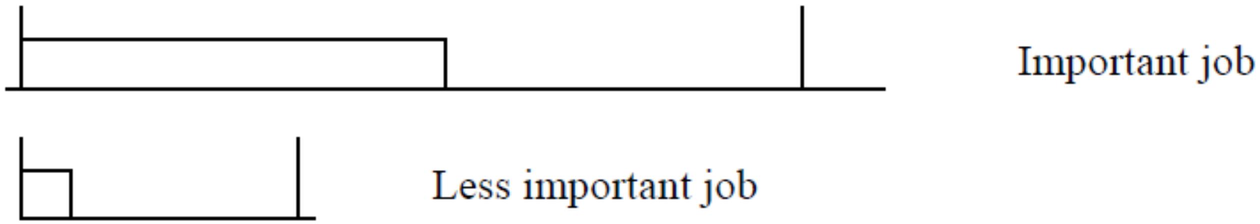
- $\{T_1=(p_1=10, e_1=4), T_2=(p_2=15, e_2=7), T_3=(p_3=30, e_3=4)\}$



How to check the schedulability without actually writing the scheduling diagram?

Priority vs Criticality

- Priority can or cannot reflect the criticality or functional importance of the task
- In the following example, giving the less important task higher priority results in both tasks meeting their deadlines



- Importance matters only when tasks cannot be scheduled (overload condition), not when they can be scheduled

Priority Assignments

- Random assignment
  - Poor performance

- Criticality ordered
  - T_1 is a brake control task
  - T_2 is a speed display task

- Urgency ordered
  - T_1 (100ms, 2ms)
  - T_2 (1000ms, 15ms)

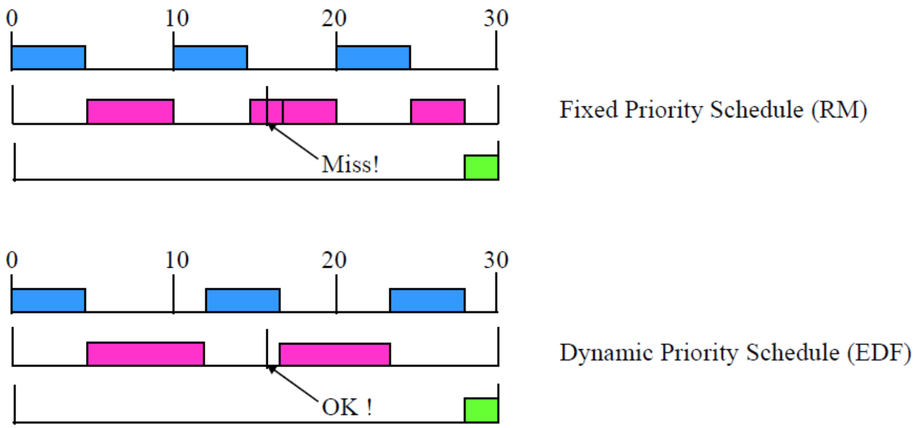
Optimal Priority Assignment Policies

- RM (Rate Monotonic) is an optimal static priority assignment for periodic tasks with implicit deadlines
  - The higher rate, the higher priority
- DM (Deadline Monotonic) is an optimal static priority assignment for periodic tasks with arbitrary deadlines
  - The shorter deadline, the higher priority

Optimality

- RM is optimal means that
  - If RM cannot schedule a task set, nobody else can.
- Recall the slide “Fixed Priority vs Dynamic Priority”

- $\{T_1=(p_1=10, e_1=4), T_2=(p_2=15, e_2=8), T_3=(p_3=30, e_3=2)\}$



- RM is optimal in the fixed-priority domain only when deadlines are equal to periods
- How to prove the optimality?

RM

- Now we know that RM is an optimal fixed-priority scheduling algorithm

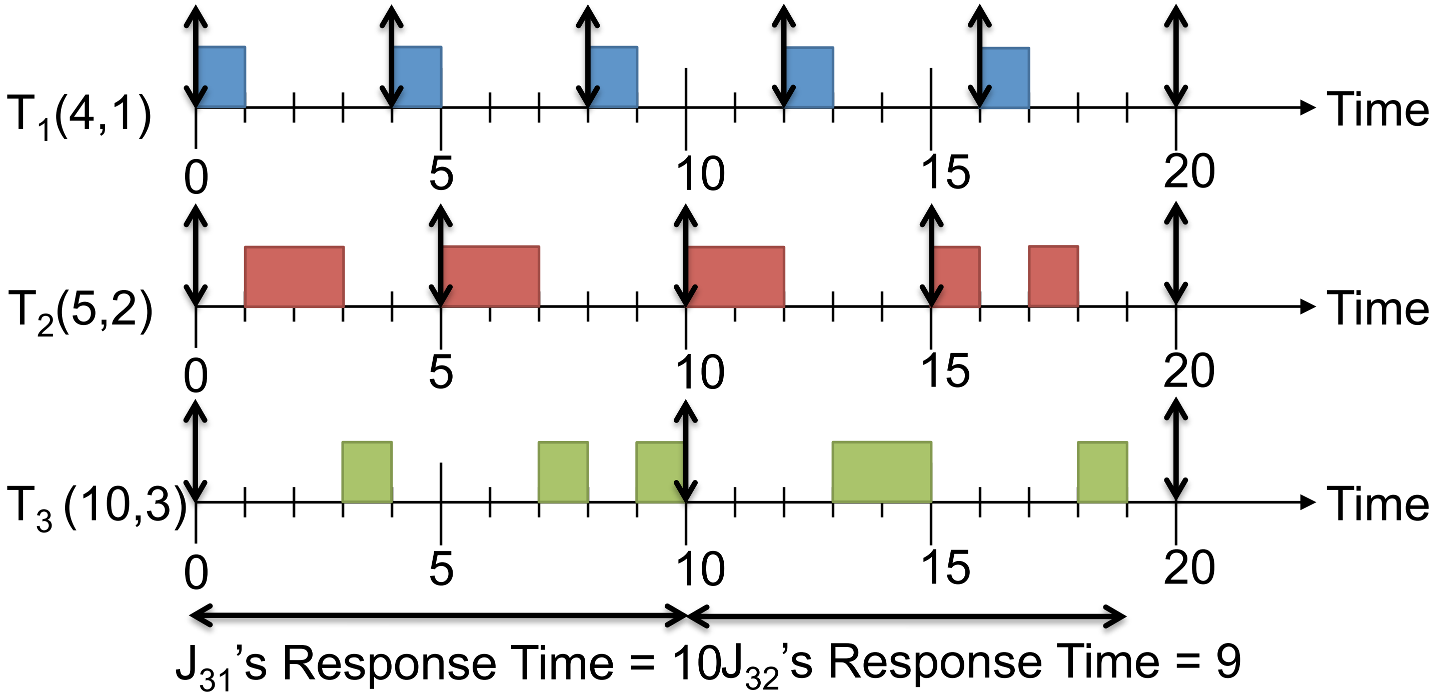
⇕

- If RM cannot schedule a workload, any fixed-priority scheduling algorithm cannot, either

- Our next question
  - How to check whether a given workload is schedulable?

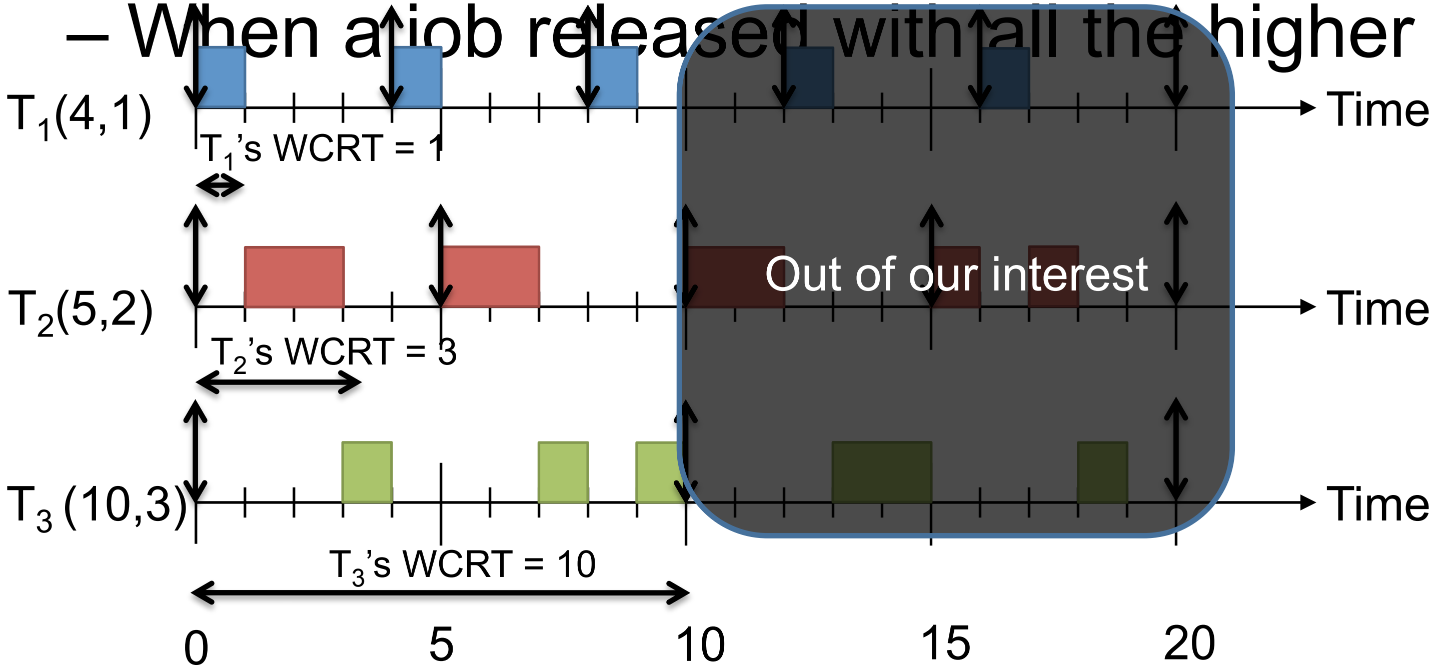
Response Time

- A job's response time
  - From its release time to its completion time



Worst-Case Response Time Analysis

- A Task's WCRT (Worst-Case Response Time)
  - The longest possible response time
- Occurs when?
  - When a job released with all the higher tasks



Schedulability Check

- Calculate every task T_i's WCRT r_i
- If every $r_i \le p_i = d_i$, then the workload is schedulable under RM
- To be honest, this method can be used for any fixed-priority scheduling algorithm with arbitrary deadlines
  - Why?

Wrap-up

- Now we know
  - **Why** RM is an optimal fixed-priority scheduling algorithm for periodic tasks with implicit deadlines
  - **How** to calculate the worst-case response times of given periodic tasks ➔ schedulability check

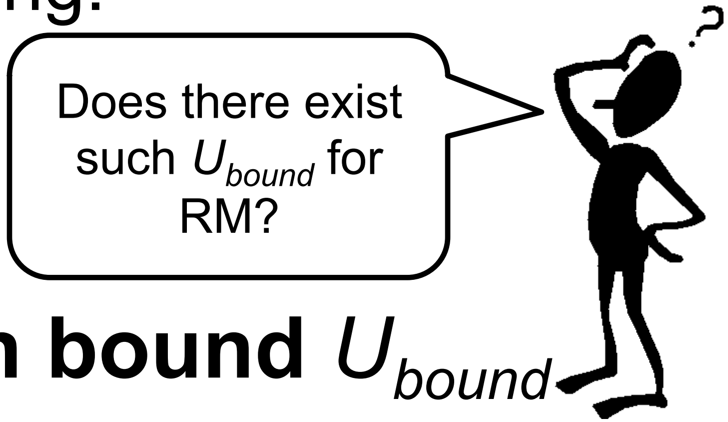
- 

➔ Utilization bound check

Utilization Bound

- Utilization
  - For a given set of periodic tasks $\{T_1, T_2, \ldots, T_N\}$ running on a CPU, its utilization U can be calculated as the following:

$$U = \sum_{1 \le i \le N} \frac{e_i}{p_i}$$

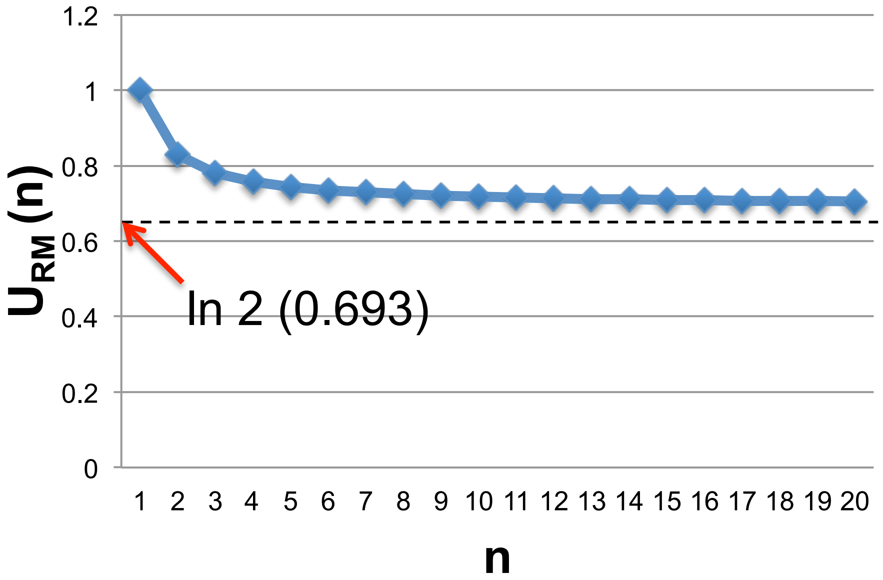


- Schedulable **utilization bound** U_{bound}
  - If a task set's utilization $U \le U_{bound}$, the task set is guaranteed to be schedulable

RM Utilization Bound

- For a system of *n* independent, preemptable periodic tasks with implicit deadlines, RM utilization bound

$$U_{RM}(n) = n(2^{1/n} - 1)$$



n	$U_{RM}(n)$
1	1
2	0.828427125
3	0.77976315
4	0.75682846
5	0.743491775
6	0.73477229
7	0.728626596
8	0.724061861
9	0.72053765
10	0.717734625

RM Utilization Bound Proof

Scheduling algorithms for multiprogramming in a **hard-real-time** environment
CL Liu, JW Layland - Journal of the ACM (JACM), 1973 - dl.acm.org
Abstract The problem of multiprogram scheduling on a single processor is studied from the viewpoint of the characteristics peculiar to the program functions that need guaranteed service. It is shown that an optimum fixed priority scheduler possesses an upper bound to **...**
8999회 인용 관련 학술자료 전체 101개의 버전 Web of Science: 1989 인용 저장

This paper actually opened the real-time systems field

Utilization Bound Check

- Only a sufficient condition
  - $U \le U_{RM} \rightarrow$ Schedulable
  - $U > U_{RM} \rightarrow$?

- Not an exact test like response time analysis. Then, why is it useful?
  - Simple one-shot solution
  - Good for online admission test

Questions