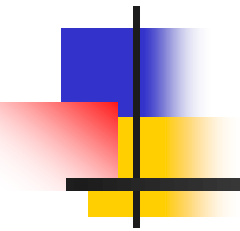
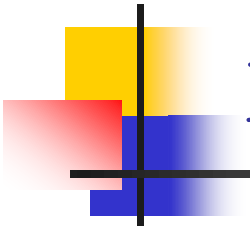


Embedded Software (Real-Time Operating System)

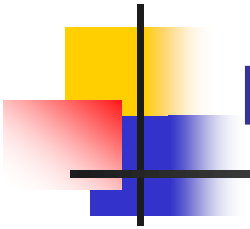
인하대학교 컴퓨터공학과

가을, 2018



개요

- 임베디드 SW 아키텍처
- 실시간 운영체제 스케줄링
- 실시간 스케줄링의 문제점과 해결책



Embedded (SW) Architectures

- **How do embedded systems manage multiple jobs?**
  - Reactions to input must be prompt.
  - Periodic tasks must be predictable.
  - How can we structure so that we can reason about latency?

- Basic Architectures
  - **Round-Robin (loop, trying everything)**
  - **Round-Robin with Interrupts (loop, but defer to high-priority work)**
  - **Function-Queue-Scheduling (schedule work units)**
  - **Real-time Operating System (let the OS suspend/resume tasks)**

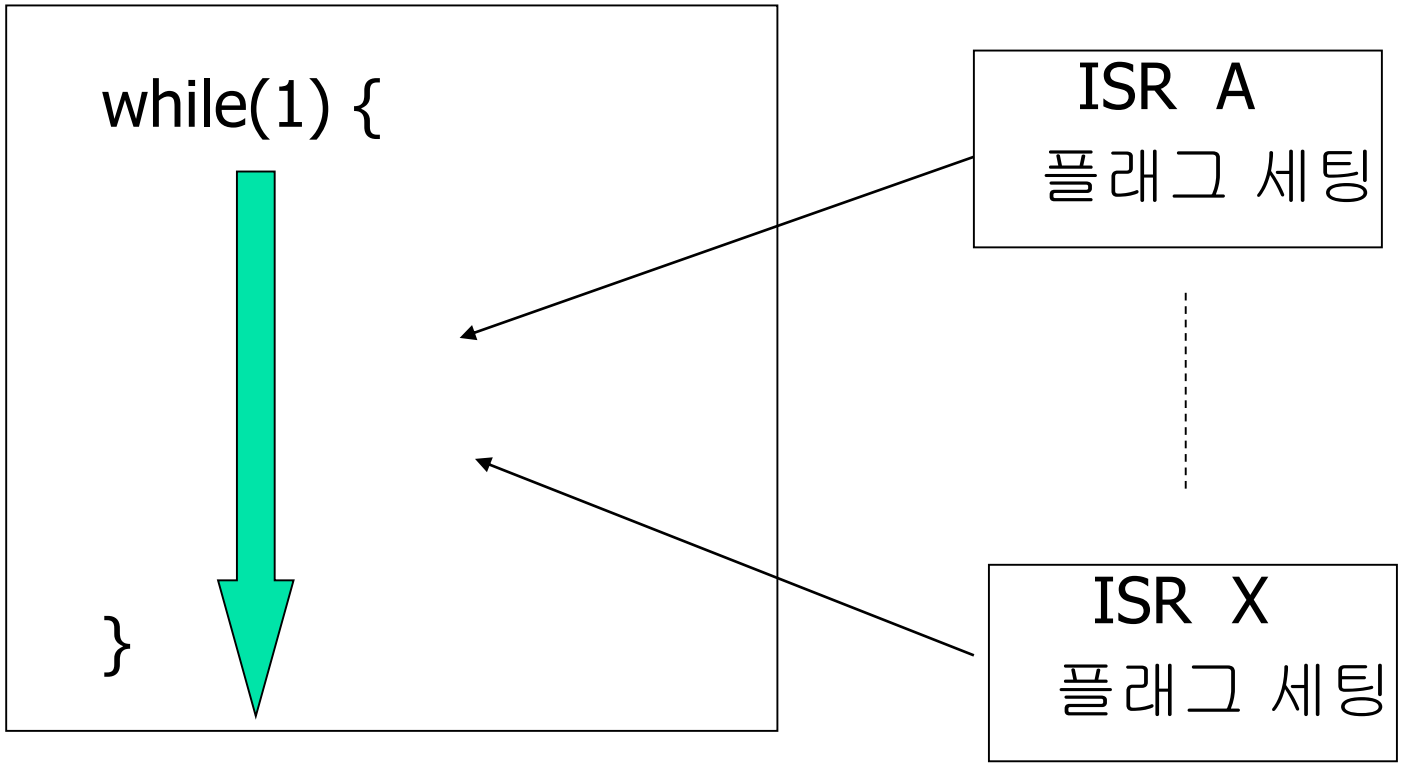
소프트웨어 구조 (1)

- RR(라운드 로빈)
  - Do everything in sequence
  - Write handler for each device
  - Infinite loop

```
while (1) -- for (;;)
{
    case 1:

    case 2:

          .
          .
          .

}
```

소프트웨어 구조 (2)

- RR with Interrupts
  - Important tasks are in ISR
  - Less important tasks are in while loop



소프트웨어 구조 (3)

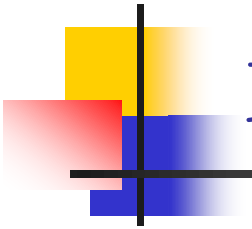
- 인터럽트를 포함한 라운드 로빈의 장단점
  - ISR로 구현된 중요한 task는 필요한 상황 발생시 (인터럽트 발생시) 바로 수행될 수 있음.(장점)
    - 구현이 비교적 간단하여 덜 중요한 다수의 작업과 한 두 개의 중요 작업이 필요한 시스템에 적합.
  - ISR 에 들어갈 device숫자가 늘어날 경우 starvation, livelock등의 문제가 발생(단점)
    - 일반 loop내부의 task의 경우 worst case estimation이 불가능함

소프트웨어 구조 (4)

- Function Queue Scheduling
  - Interrupt routine 이 수행 대기중인 function(들)의 포인터를 관리함
  - 포인터 큐에서 메인 스케줄 루틴이 다음 수행할 function을 선택함.
- 특징
  - Queue 는 FIFO가 아님
  - Task가 function으로 표현됨
  - 라운드로빈의 경우 task가 coding된 순서대로 수행되지만, FQS의 경우 순서가 가변적임.

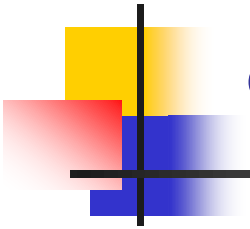
소프트웨어 구조 (5)

- **Real-time Operating System**
  - **OS가 task의 중단/재시작을 관리**
  - RR과 비교할 때
    - main함수를 loop로 구성할 필요가 없음
    - 단순한 multi-tasking으로 프로그래머 측면에서 관리용이
  - FQS와 비교할 때
    - Task switching 제공
    - 최우선순위 task의 worst cast waiting time이 ISR 처리 시간으로 축소
  - ISR과 task간의 shared variable을 대신하는 방법을 제공;
    - semaphore 등
    - shared data problem을 효율적으로 처리할 수 있음.



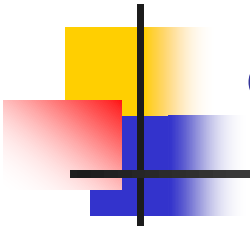
개요

- 임베디드 SW 아키텍처
- 실시간 운영체제 스케줄링
- 실시간 스케줄링의 문제점과 해결책



인터럽트 (1)

- 인터럽트
  - 비동기적 이벤트가 발생함을 CPU 에서 알려주는 하드웨어 매커니즘
  - ISR 로 문맥 전환 후
    - 가장 높은 우선 순위를 갖는 태스크가 다음 수행되도록 조정
- 인터럽트 지연 시간 (Interrupt latency)
  - 인터럽트에 반응해 ISR을 수행하기(전)까지의 시간
    - 인터럽트가 꺼져 있는 최대 시간 + 인터럽트 서비스 루틴을 수행하는 데 필요한 시간

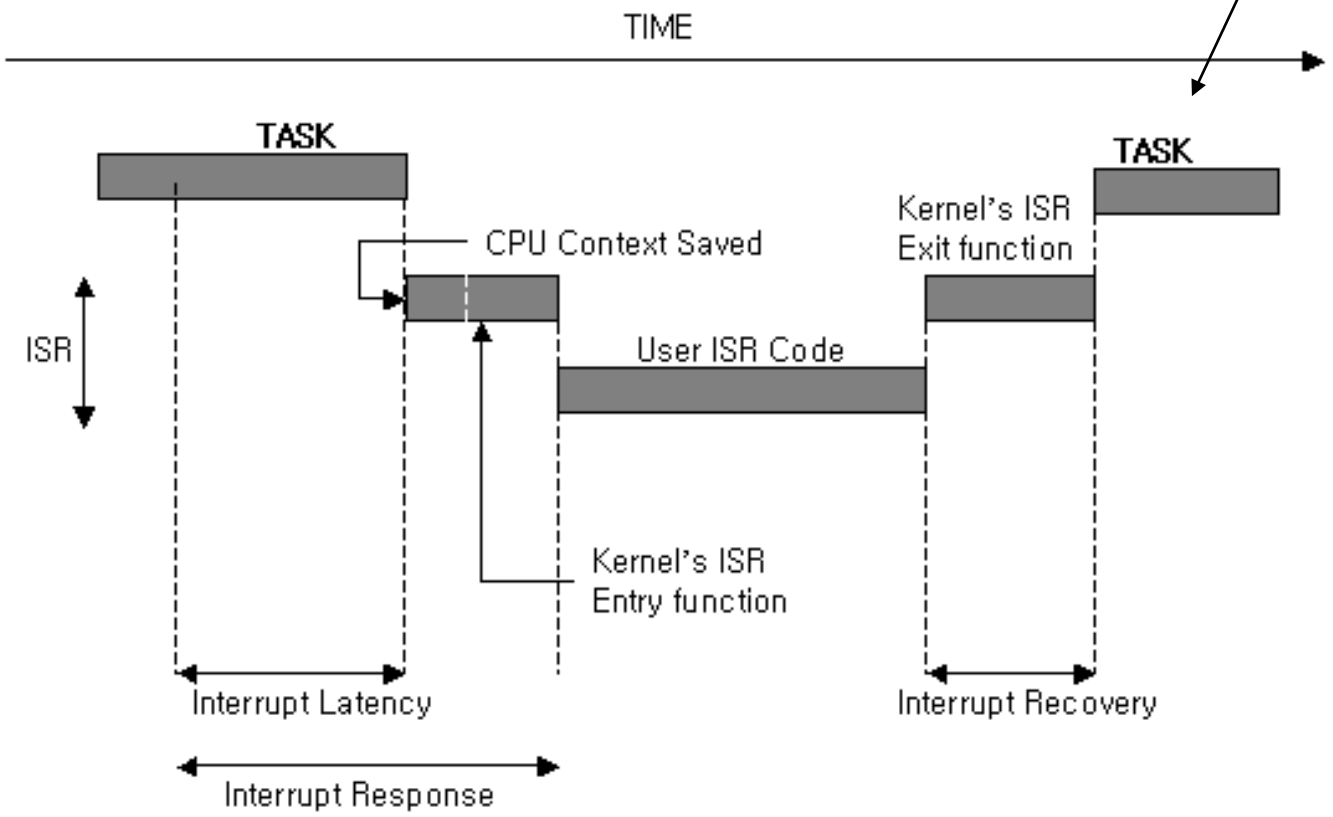


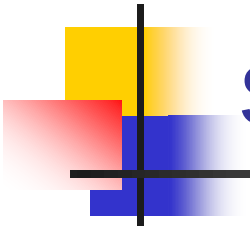
인터럽트 (2)

- 인터럽트 응답 시간 (Interrupt response)
  - 인터럽트에 반응해 서비스 루틴을 수행하기까지의 시간
    - 인터럽트 지연 시간 + CPU 문맥을 저장하는 시간 + 커널의 ISR 진입함수 수행하는데 걸리는 시간
- 인터럽트 복구 시간 (Interrupt recovery)
  - CPU 내용을 되돌리는 시간 + 가장 높은 우선 순위의 태스크를 선택하는 시간 + 가장 높은 우선 순위의 태스크로 돌아가는 시간

인터럽트 (3)

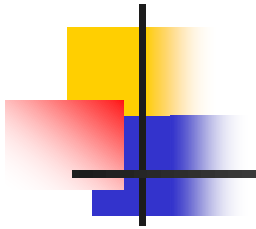
높은 우선 순위 태스크





Scheduler

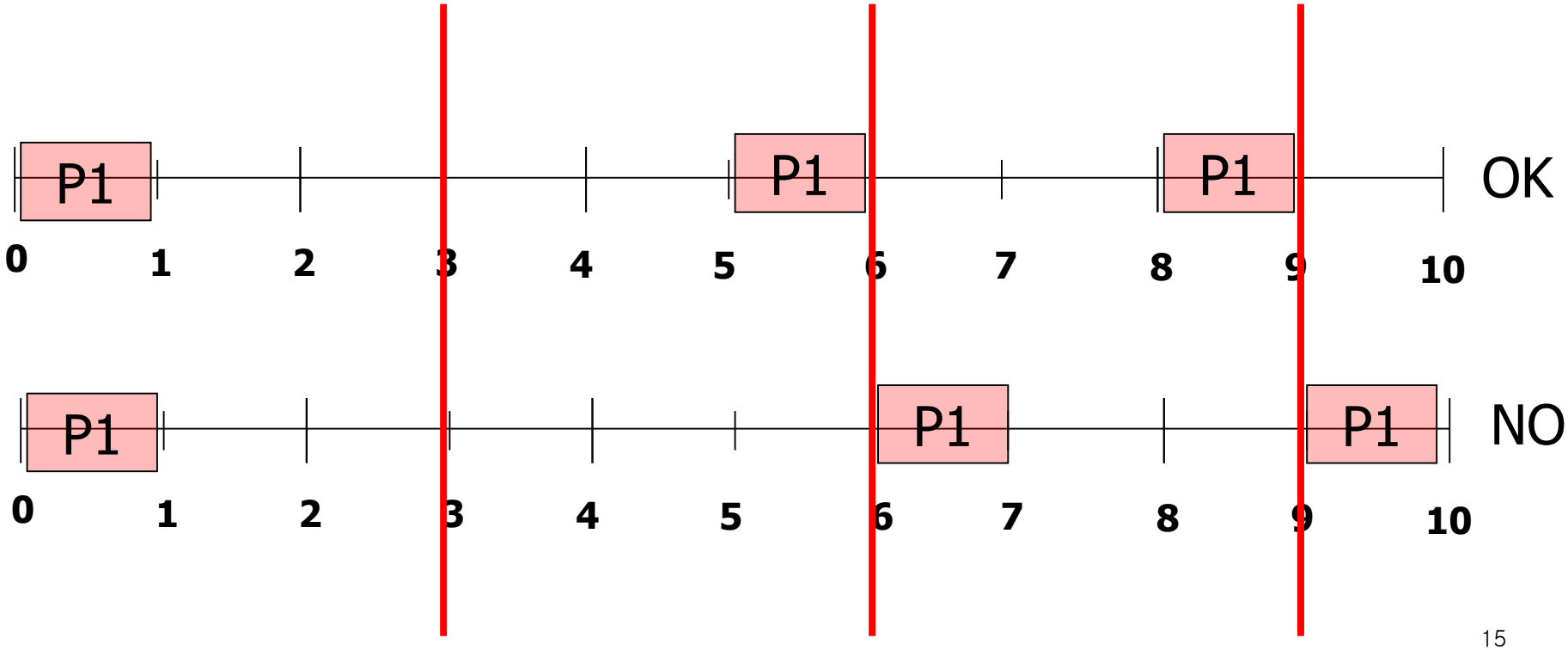
- Scheduler
  - <Examples>
    - FIFO
    - Shortest Job First (SJF)
    - Priority
    - Round robin
    - …

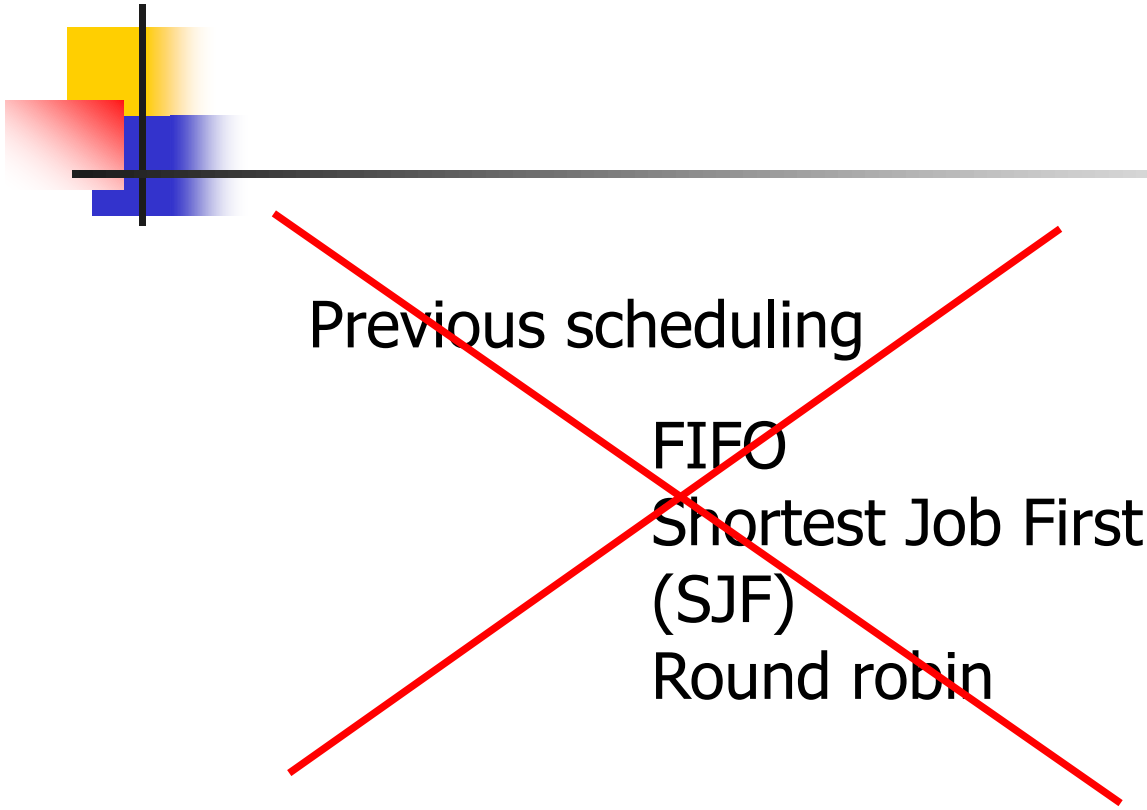


- In an RTOS
  - A scheduler should satisfy the real-time requirements of tasks
  - Real-time scheduling
    - Each task has its own deadline
    - The task needs to be finished before its deadline

Periodic task

Task	Period	Deadline	Computation time
τ_1	3	3	1
τ_2	4	4	1
τ_3	5	5	2



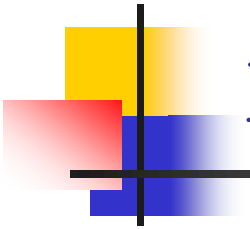


Previous scheduling

FIFO
Shortest Job First (SJF)
Round robin

Real-time scheduling

Rate-Monotonic Scheduling (RMS)
Earliest Deadline First scheduling (EDF)



개요

- 임베디드 SW 아키텍처
- 실시간 운영체제(RTOS) 스케줄링
  - RMS
  - EDF
- 실시간 스케줄링의 문제점과 해결책

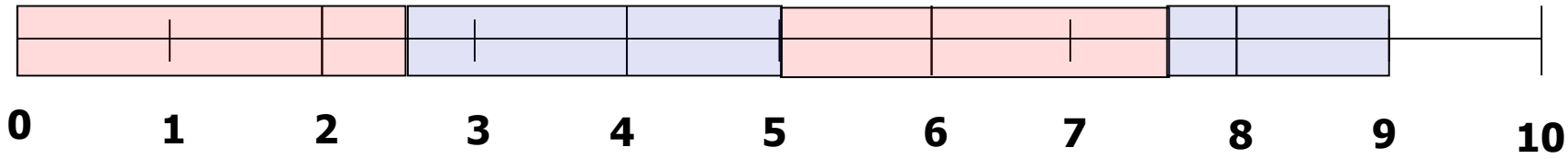
Rate Monotonic Scheduling - RMS

- It is a basically static priority scheduling algorithm
  - Assigns a priority to each task
  - Then, tasks are scheduled according to the priority given.
- Basically the RMS is designed for periodic tasks.
  - Assigns the priority of each task according to its period, so the shorter the period the higher the priority

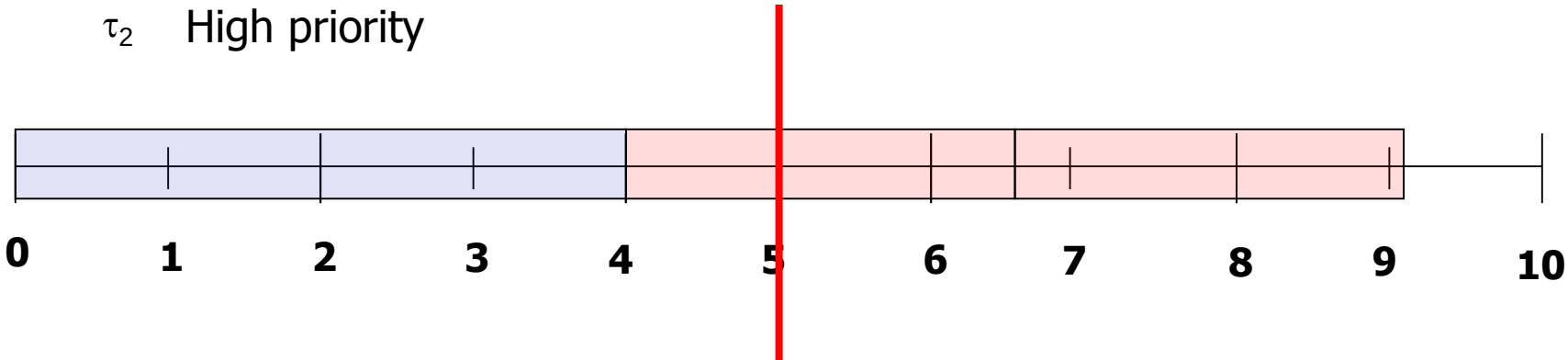
- For example, there are two tasks

Task	Period	Deadline	Computation time
τ_1	50	50	25
τ_2	100	100	40

τ_1 High priority



τ_2 High priority



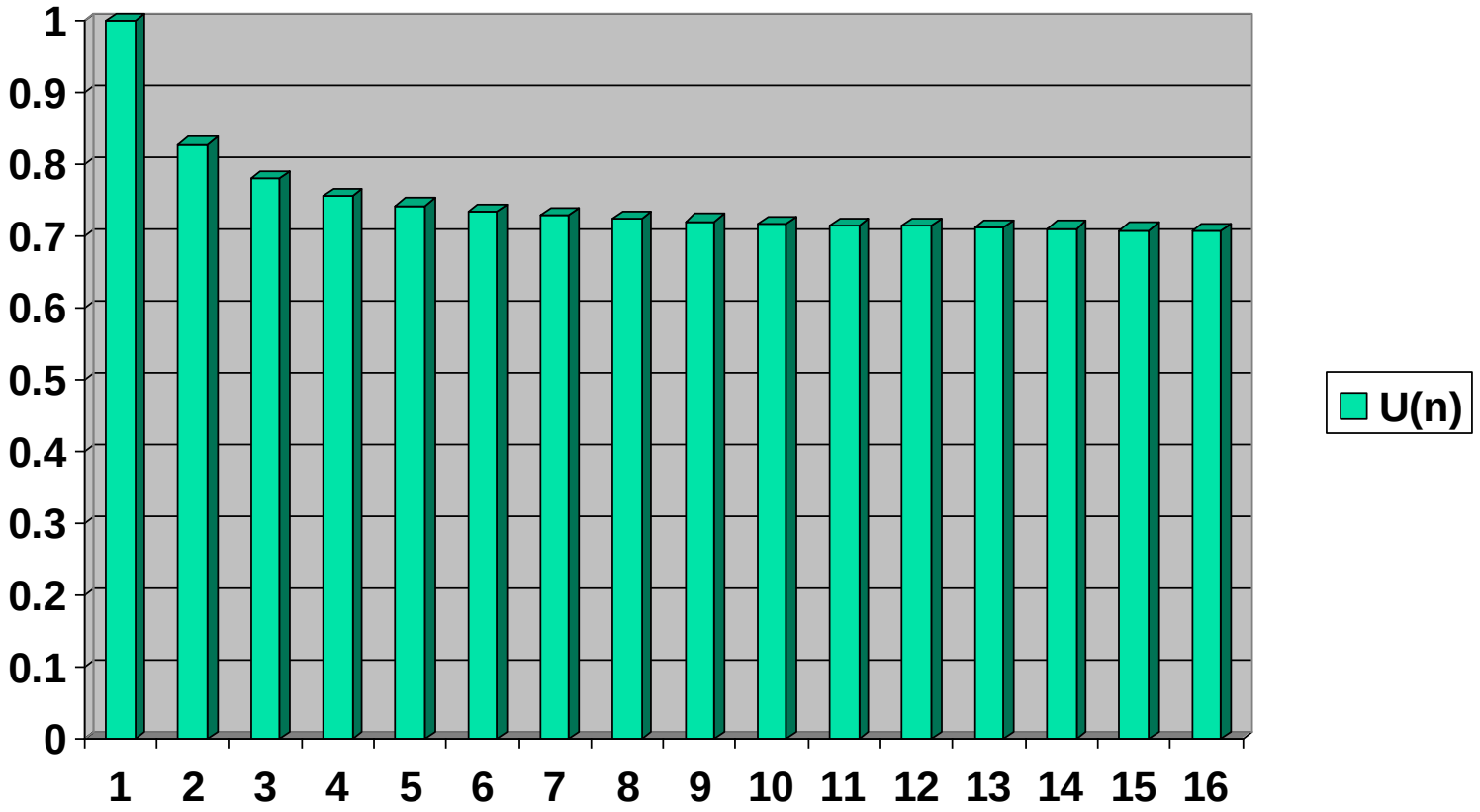
- Schedulibility test
  - "The task set is feasible if and only if all tasks have completed within their deadlines"

Worst-case computation time

Utilization $\longrightarrow U = \sum_{i=1}^{N} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$

Period

Then, the task set is feasible !



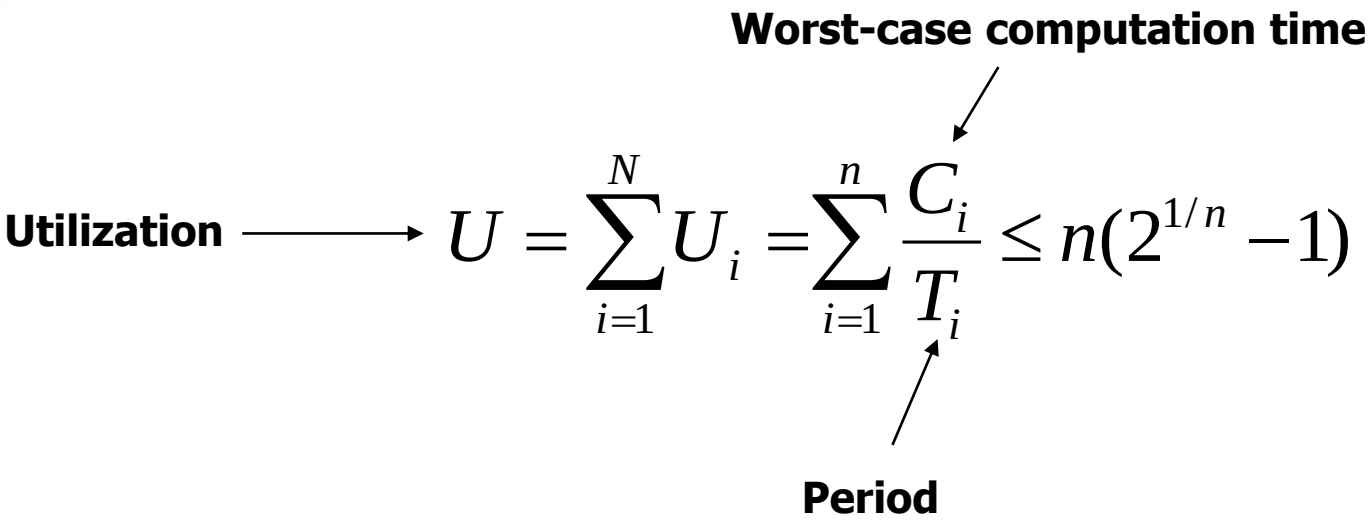
$$N \triangleright \infty \quad \Rightarrow \quad U \triangleright 0.693$$

Task	Period	Deadline	CT
τ_1	10	10	2
τ_2	25	25	5
τ_3	35	35	10

$$U = \sum_{i=1}^{N} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

U = 2/10 + 5/25 + 10/35 = 0.686.. < 0.779

$$3(2^{1/3} - 1)$$

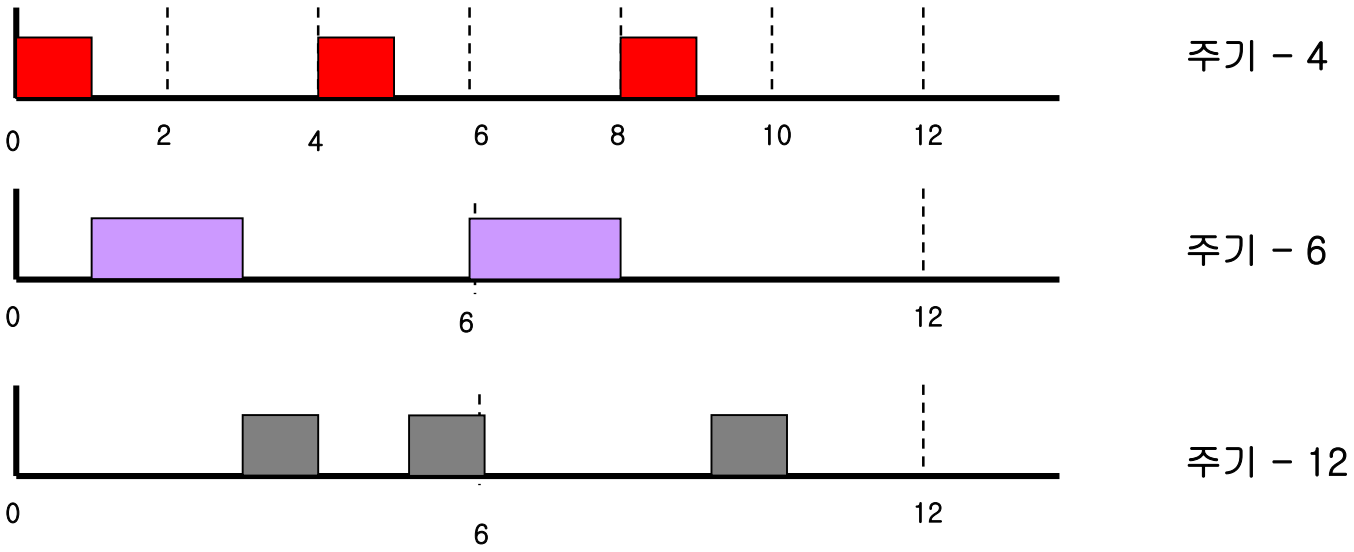


This inequality is not a necessary condition, but a sufficient condition

See the next slide

task	period	deadline	CT
τ_1	4	4	1
τ_2	6	6	2
τ_3	12	12	3

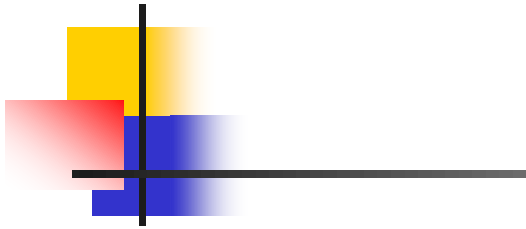
$$U = 1/4 + 2/6 + 3/12 = 0.83.. > 0.779$$



Feasible

Earliest Deadline First- EDF

- It is a basically dynamic priority scheduling algorithm
  - The priority of the task may change
  - It assigns priorities in order of deadline.
    - The highest-priority process is the one whose deadline is nearest in time
  - Clearly, priorities must be recalculated at every completion of a task

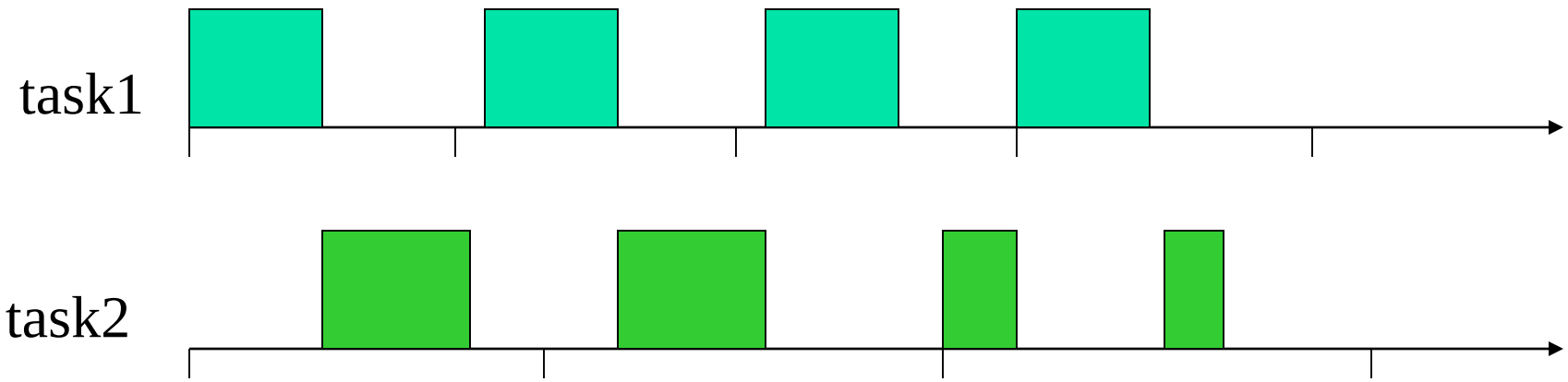


- Schedulability test

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

Sufficient and necessary condition

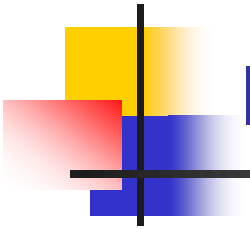
- Example



Task	Period	Deadline	CT
τ_1	3	3	1
τ_2	4	4	1
τ_3	5	5	2

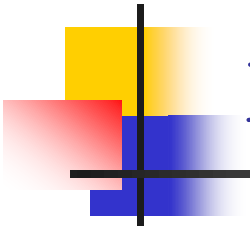
$$U = 1/3+1/4+2/5 = 0.98333333$$

timeline	Running task	Deadline
0		
1		
2		τ_1
3		τ_2
4		τ_3
5		τ_1
6		
7		τ_2



Introduced Scheduling Algorithm

- Assumptions
  - All tasks are periodic.
  - Tasks are independent
  - Each deadline ends with its period.
  - Static task execution time.
  - All tasks are preempt.
  - No scheduling overhead at all.



개요

- 임베디드 SW 아키텍처
- 실시간 운영체제(RTOS) 스케줄링
- 실시간 스케줄링의 문제점과 해결책

우선 순위 전도 현상 (1)

- 가정
  - 고정 우선 순위 선점형 스케줄링
- 정의
  - 높은 우선 순위의 태스크가 낮은 우선 순위의 태스크에 의해서 블럭킹되는 현상
- 언제 발생하는가?
  - 공유된 자원을 사용하면서 생기는 문제

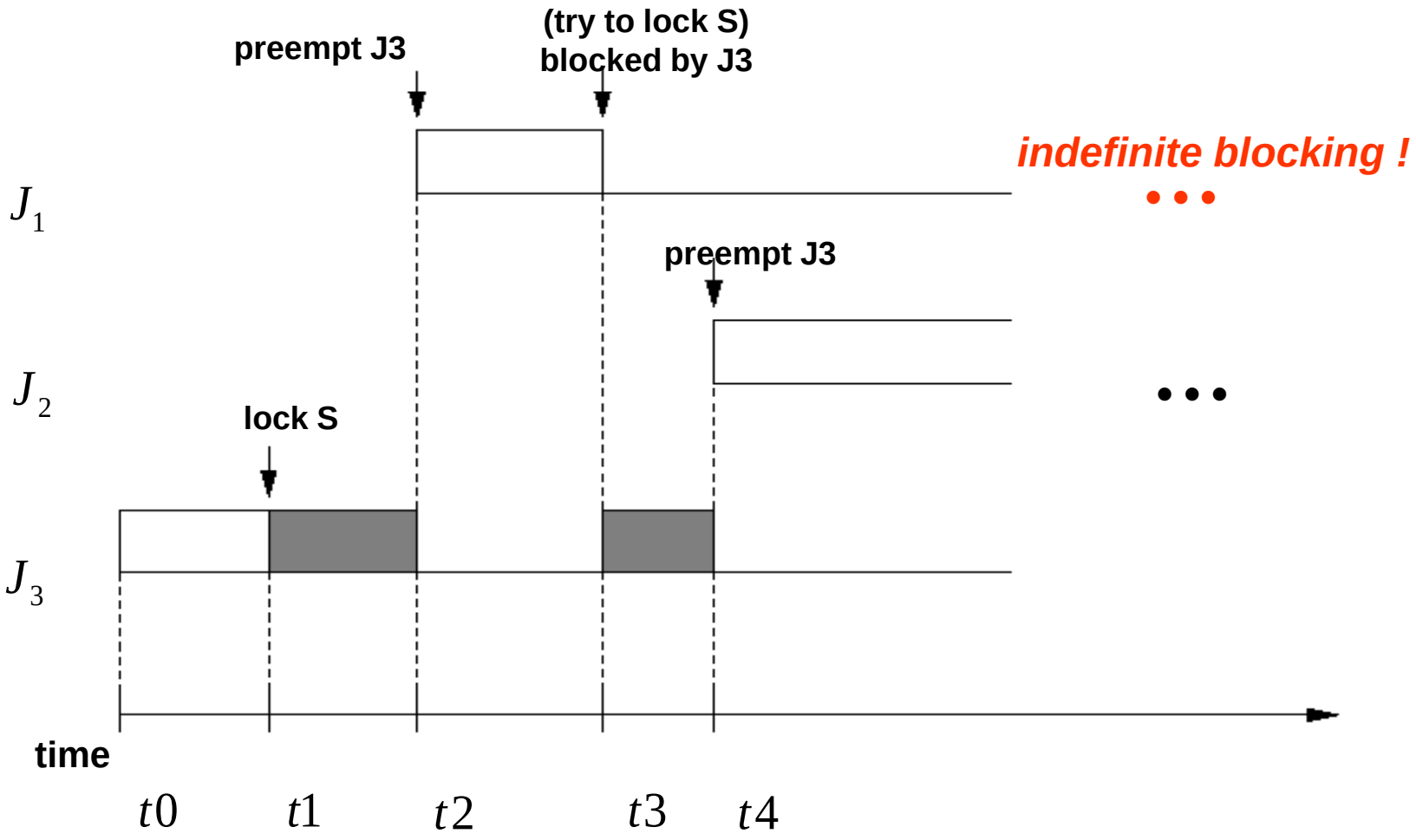
unavoidable blocking

우선 순위 전도 현상 (2)

- 문제점
  - 시나리오
    - 1. 낮은 우선 순위 태스크가 공유 자원에 접근한다.
    - 2. 높은 우선 순위의 태스크가 공유 자원을 접근하려고 하나, 락이 걸린 것을 확인해서 기다린다.
    - 3. 중간 우선 순위 태스크가 낮은 우선 순위의 태스크를 선점한다.

indefinite priority inversion

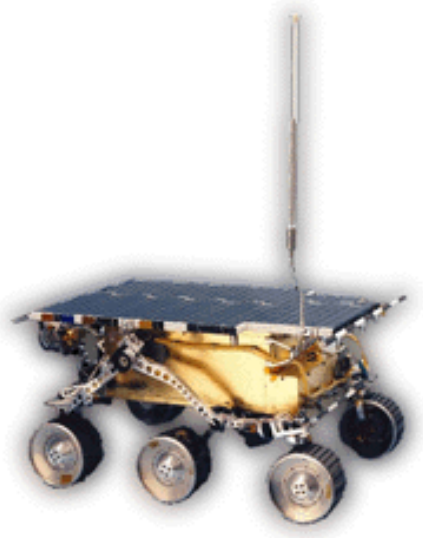
우선 순위 전도 현상 (3)


preempt J3
(try to lock S)
blocked by J3
indefinite blocking !
J1
preempt J3
J2
lock S
J3
time
t0
t1
t2
t3
t4

우선 순위 전도 현상 (3)

- 화성 탐사선 문제 (MARS pathfinder problem)

“ 화성 탐사선이 기상 자료를 수집하기 시작해서 얼마 지나지 않아서 우주선이 계속 리셋되는 현상이 생겼다”



소저너

우선 순위 전도 현상 (4)

- 화성 탐사선 문제 분석
  - 1. 운영체제
    - VxWorks RTOS
      - 선점형 우선 순위 기반 스케줄링
  - 2. “information bus”
    - 우주선의 서로 다른 요소에서 데이터를 주고 받는 일종의 공유 메모리 영역
    - 공유 자원 영역으로 세마포어를 통해서 상호 배제

우선 순위 전도 현상 (5)

- 화성 탐사선 문제 분석
  - 3. 태스크
    - 버스 관리 태스크
      - “information bus” 를 관리하는 태스크
      - 가장 높은 우선 순위
    - 데이터 수집 태스크
      - 기상 데이터를 수집하는 태스크
      - 가장 낮은 우선 순위
    - 통신 태스크
      - 중간 우선 순위

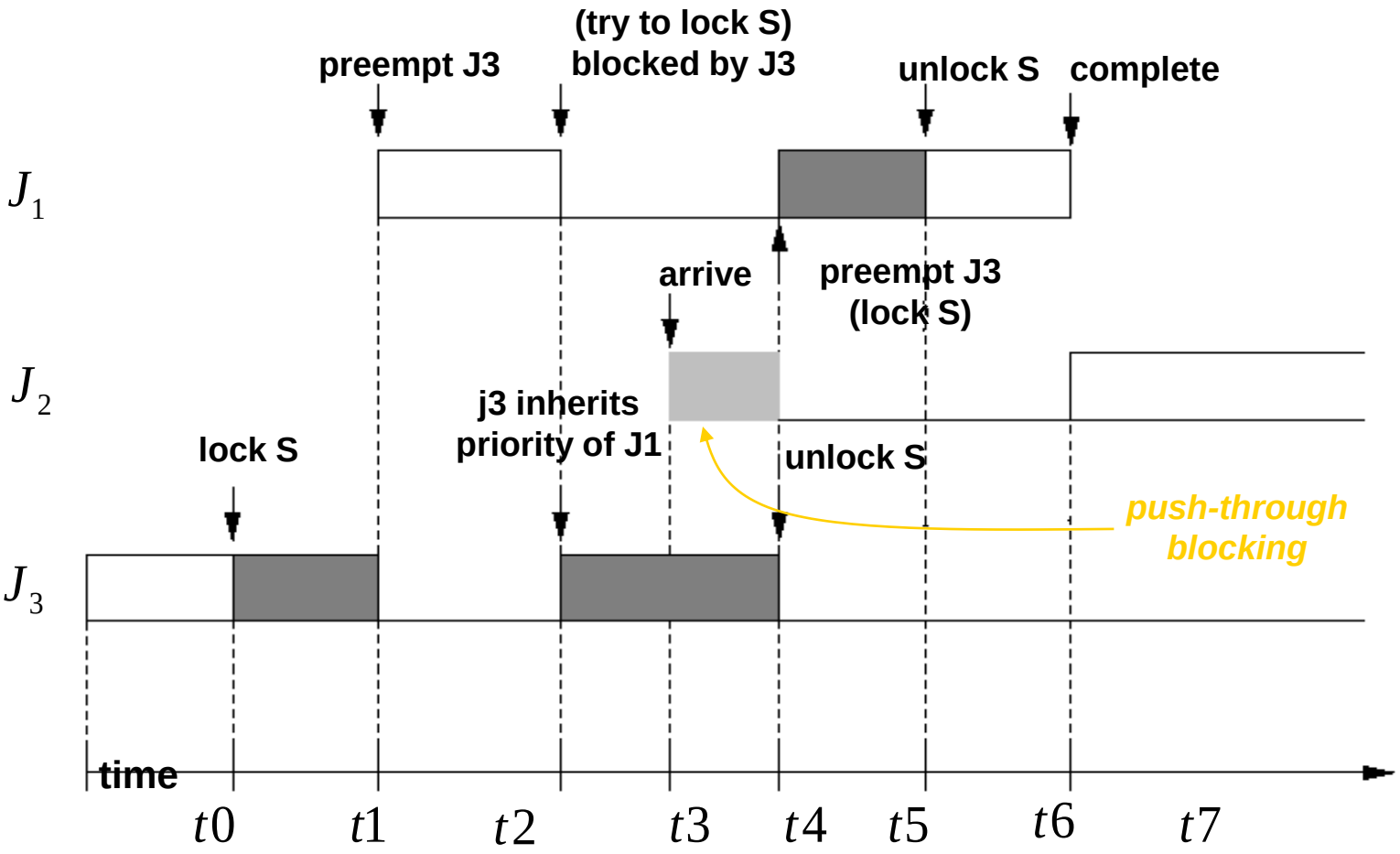
우선 순위 전도 현상 (6)

- 시나리오
  - 1. 데이터 수집 태스크 동작
    - Information bus 접근
  - 2. 버스 관리 태스크가 information bus 접근 시에 블록킹됨
    - 이유: 데이터 수집 태스크가 Information bus 접근함
  - 3. 비교적 수행 시간이 긴 통신 태스크가 데이터 수집 태스크를 선점함
  - 4. 버스 관리 태스크가 제대로 수행되지 않아서 watchdog timer가 시스템을 리셋시킴

우선 순위 계승 프로토콜 (1)

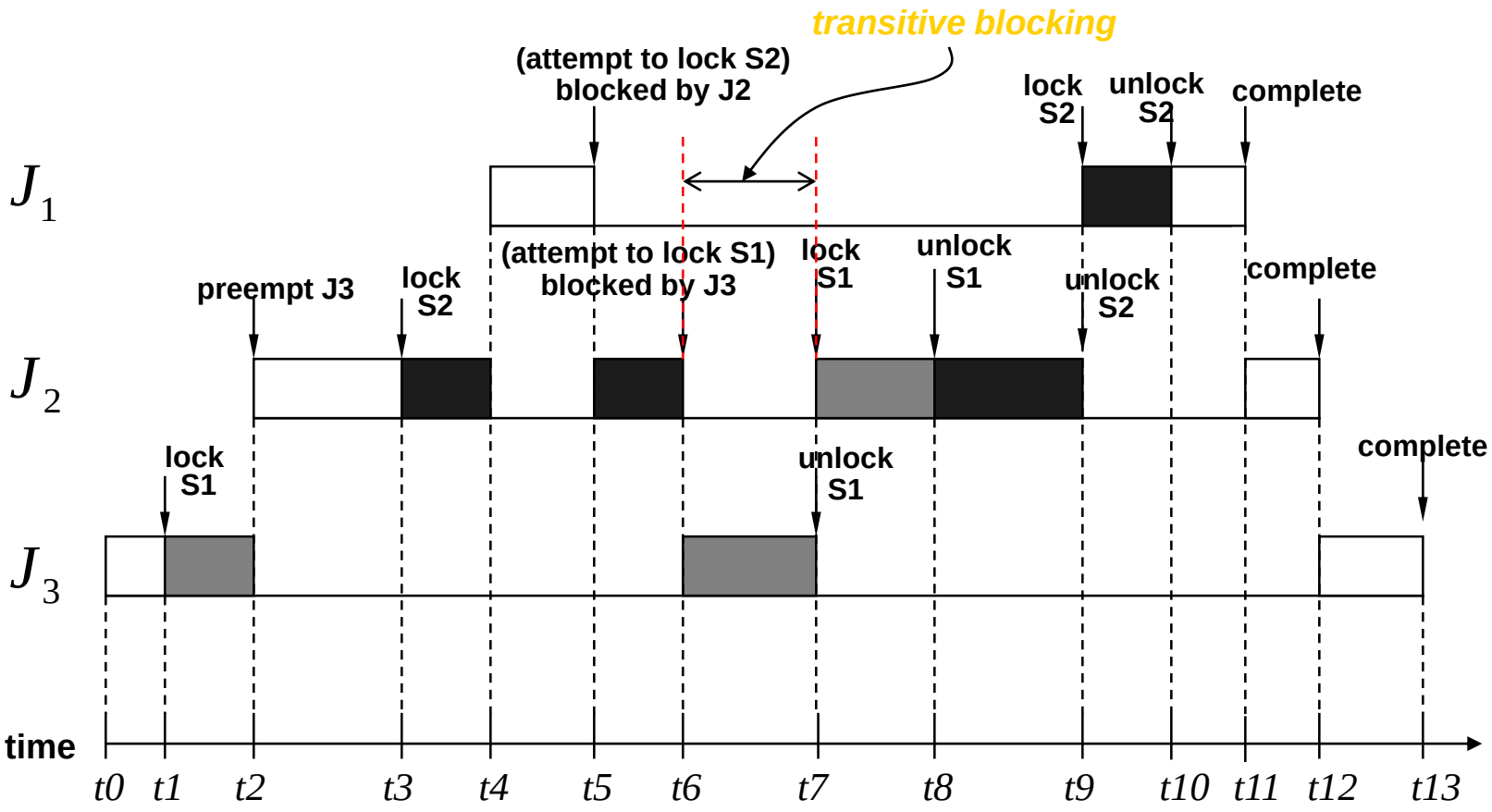
- 아이디어
  - 어떤 태스크가 높은 우선 순위를 갖는 태스크를 블록 시켰을 경우
    - 그 태스크는 그 태스크에 의해서 블록된 태스크 중 가장 높은 우선 순위를 갖는 태스크의 우선 순위를 상속받는다.
  - 이행적 성질
    - If J_3 blocks J_2 and J_2 blocks J_1, J_3 would inherit the priority of J_1 via J_2.
  - 언제 우선 순위를 되돌려 놓는가?
    - 임계 영역을 벗어날 때
- 특징
  - indefinite blocking 해결

우선 순위 계승 프로토콜 (2)



우선 순위 계승 프로토콜 (3)

- 이행 블록킹 (transitive blocking)

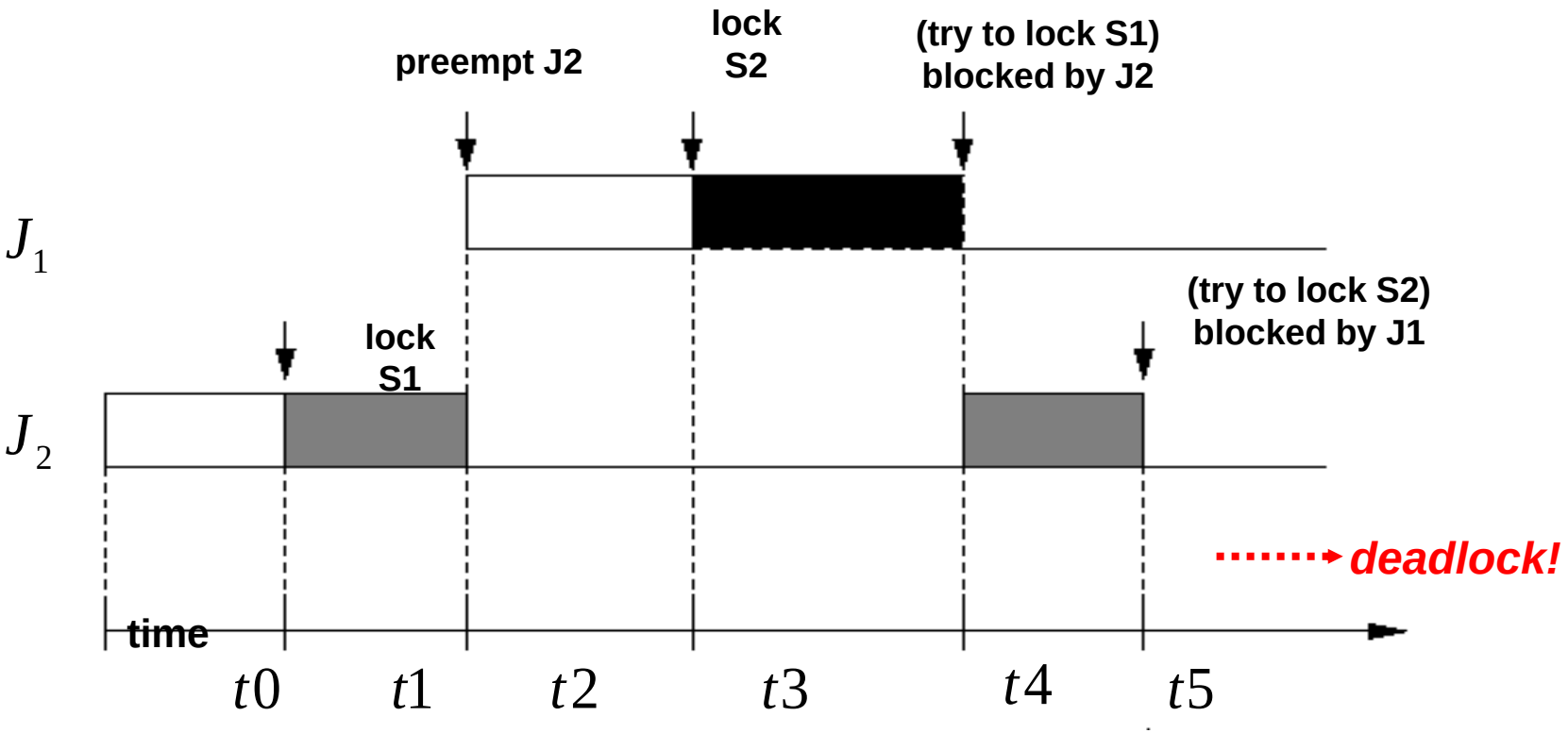


우선 순위 계승 프로토콜 (4)

$J_1 = \{..., P(S_2) ..., P(S_1),...,V(S_1) ...,V(S_2),...\}$

$J_2 = \{..., P(S_1) ..., P(S_2),...,V(S_2) ...,V(S_1),...\}$

crossing nested semaphores



우선순위 상한 프로토콜 (1)

- 아이디어
  - 자원의 우선 순위 상한 값
    - 해당 자원을 쓰려는 태스크 중 가장 높은 우선 순위를 갖는 태스크의 우선 순위
  - 어떤 특정 job J가 임계영역에 들어가기 위한 조건
    - J의 우선 순위가 현재 J 가 아닌 다른 태스크에 의해서 락이 걸려 있는 모든 자원의 우선 순위 상한 값 중 최대값 보다 커야 한다.
- 특징
  - 이행 블록킹 문제 해결
  - 데드락 해결

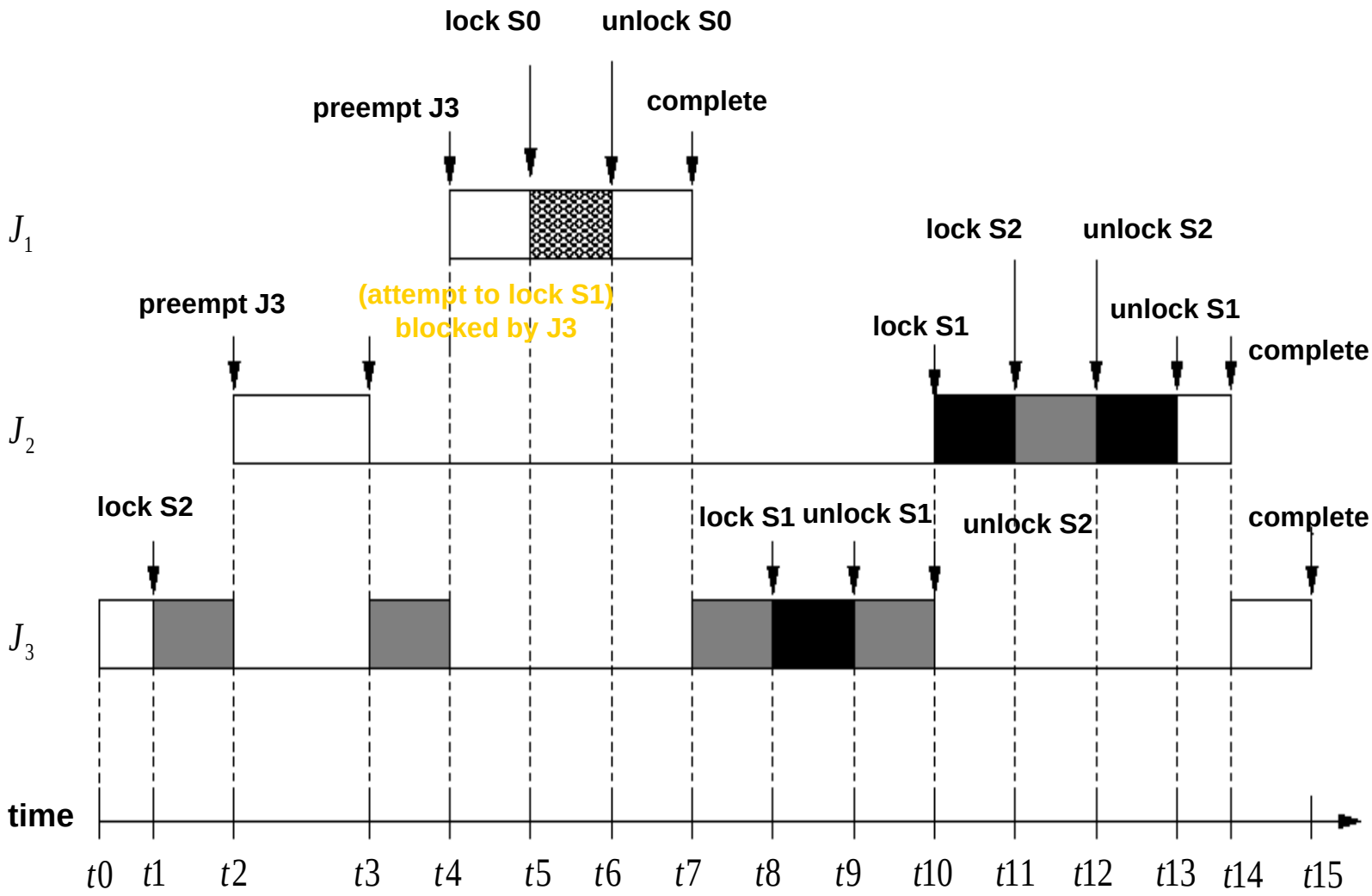
우선순위 상한 프로토콜 (2)

- 우선 순위 상한 값 계산
  - 시나리오

    - *J1* = {..., *P*(*S0*) ..., *V*(*S0*),...}
    - *J2* = {..., *P*(*S1*) ..., *P*(*S2*),...,*V*(*S2*) ...,*V*(*S1*),...}
    - *J3* = {..., *P*(*S2*) ..., *P*(*S1*),...,*V*(*S1*) ...,*V*(*S2*),...}
  - 각 자원의 우선 순위 상한 값
    - *S0; J1* 의 우선 순위
    - *S1; J2* 의 우선 순위
    - *S2; J2* 의 우선 순위

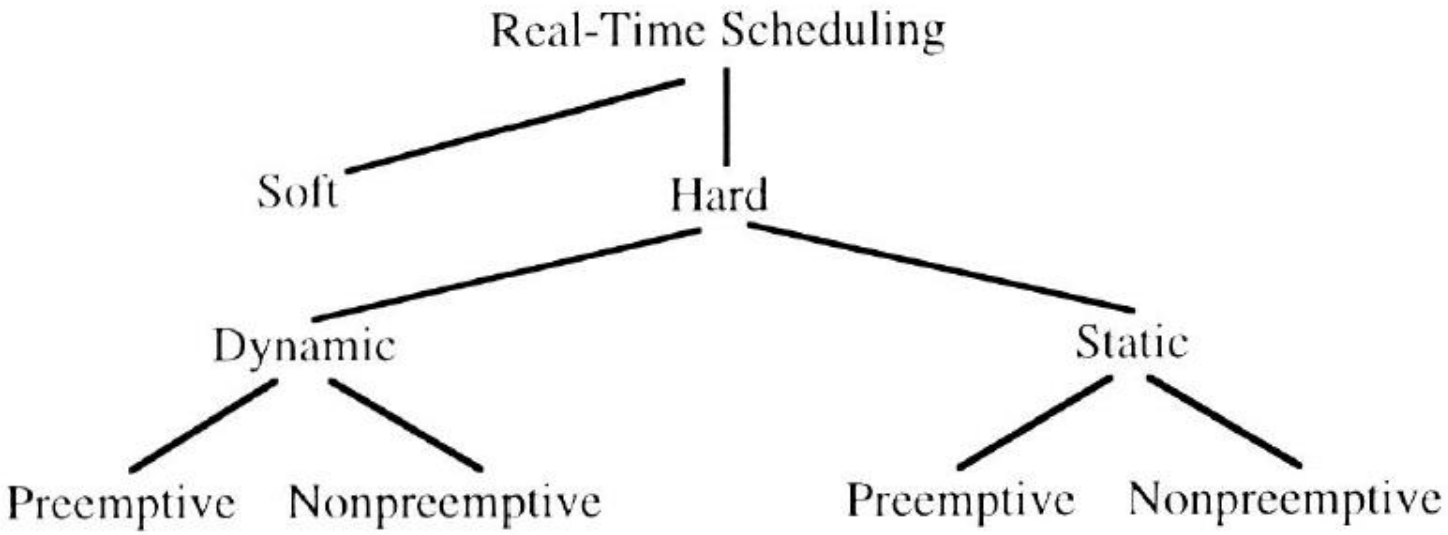
우선순위 상한 프로토콜 (3)

J1 의 우선 순위 > 시스템 priority ceiling



요약 및 결론

- 인터럽트
- 스케줄링
  - RMS (Rate Monotonic Scheduling)
  - EDF (Earliest Deadline First)



요약 및 결론

- 스케줄링 실적용 문제점
  - 우선 순위 전도 현상 (priority inversion)
  - Priority Inheritance Protocol (PIP)
  - Priority Ceiling Protocol (PCP)

- References

  - Sha L., R. Rajkumar, and J.P. Lehoczky. "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," IEEE Transactions on Computers, September 1990, p. 1175
  - Reeves, Glenn. "Re: What Really Happened on Mars?," Risks-Forum Digest, Volume 19: Issue 58, January 1998.