임베디드시스템설계

스케줄링

Byungjin Ko
Department of Intelligent Robotics

운영체제에서 일어나는 다양한 스케줄링

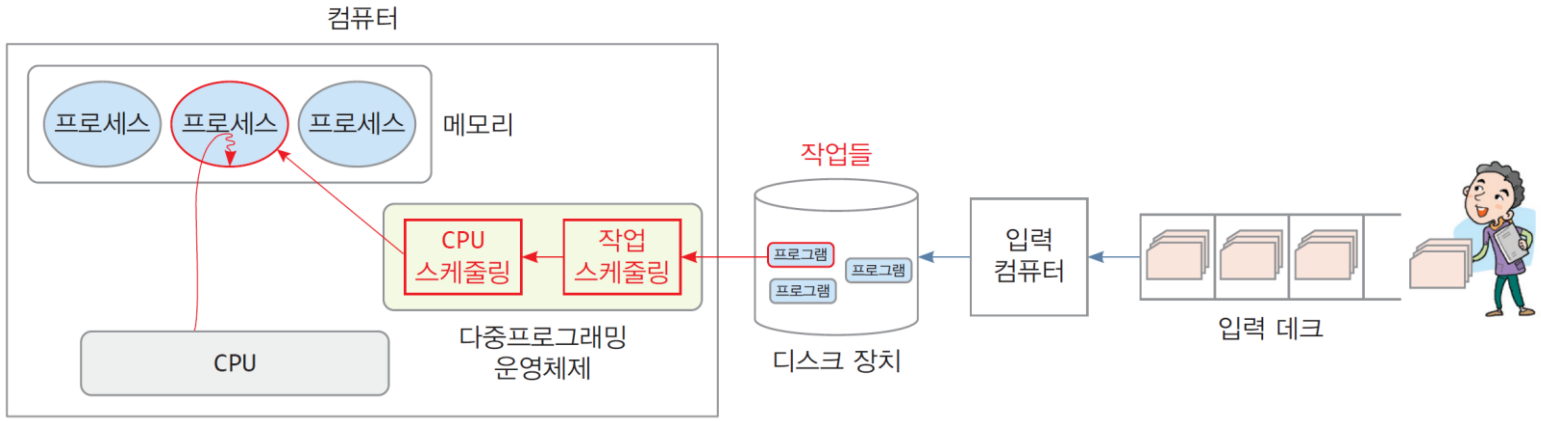
- 스케줄링은 왜 필요할까?
  - 자원에 대한 경쟁이 있는 곳에서 경쟁자 중 하나 선택 하는 과정 필요
  - 자원 : CPU, 디스크, 프린트, 파일, 데이터베이스 등

- 컴퓨터 시스템 내 다양한 스케줄링
  - 작업(job) 스케줄링
    - 대기중인 배치 작업(Job) 중 메모리에 적재할 작업 결정
  - CPU 스케줄링
    - 프로세스/스레드 중에 하나를 선택하여 CPU 할당
    - 오늘날 CPU 스케줄링은 스레드 스케줄링
  - 디스크 스케줄링
    - 디스크 장치 내에서 디스크 입출력 요청 중 하나 선택
  - 프린터 스케줄링
    - 프린팅 작업 중 하나 선택하여 프린터 할당

다중프로그래밍과 스케줄링

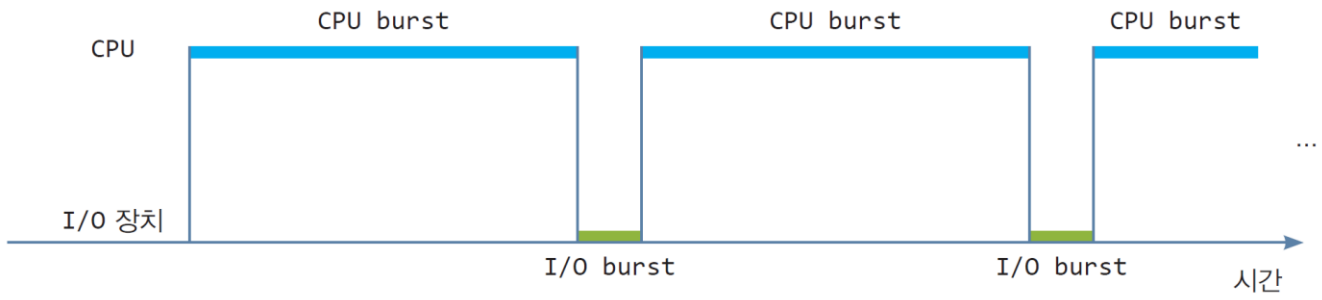
- 다중프로그래밍의 도입 목적 리뷰
  - CPU 유휴 시간 (idle time) 줄여 CPU 활용률 향상 목적
    - 프로세스가 I/O를 요청하면 다른 프로세스에게 CPU 할당

- 다중프로그래밍과 함께 2가지 스케줄링 도입
  - 작업 스케줄링(job scheduling)
    - 메모리에 적재된 프로세스가 종료하면 디스크에서 기다리는 작업 중 하나를 선택하여 메모리에 적재하는 과정을 뜻함
  - CPU 스케줄링(CPU scheduling)
    - 메모리에 적재된 작업 중 CPU에 실행시킬 프로세스를 선택하는 과정을 뜻함



CPU burst와 I/O burst

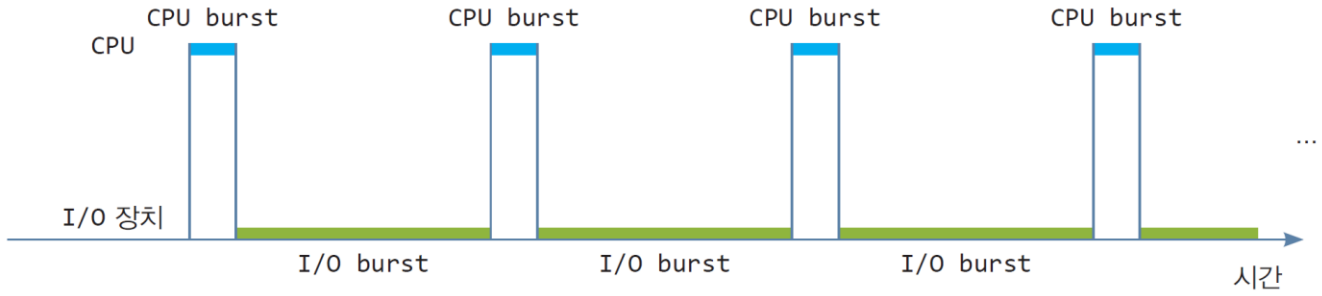
- 프로그램의 실행 특성
  - CPU 연산 작업과 I/O 작업(화면 출력, 키보드, 입력, 파일 입출력 등)이 순차적으로 섞여 있음
  - CPU-burst – I/O burst – CPU-burst – I/O burst의 반복 …



(a) CPU 집중 프로세스의 실행 특성

- CPU burst
  - 프로그램 실행 중 CPU 연산(계산 작업)이 연속적으로 실행되는 상황

- I/O burst
  - 프로그램 실행 중 I/O 장치의 입출력이 이루어지는 상황



(b) I/O 집중 프로세스의 실행 특성

CPU 스케줄링의 정의와 목표

- CPU 스케줄링
  - 정의
    - 실행을 기다리는 스레드 중 하나를 선택하는 과정
  - 기본 목표
    - CPU 활용률 향상
    - 컴퓨터 시스템 처리율 향상
  - 컴퓨터 시스템에 따라 CPU 스케줄링의 목표가 다를 수 있음

CPU 스케줄링의 기준(criteria)

- 스케줄링 알고리즘의 다양한 목표와 평가 기준 (1/2)
  - CPU 활용률(CPU utilization)
    - 전체 시간 중 CPU의 사용 시간 비율 (운영체제 입장)
  - 처리율(throughput)
    - 단위 시간당 처리하는 스레드 개수 (운영체제 입장)
  - 공평성(fairness)
    - CPU를 스레드들에게 공평하게 배분 (사용자 입장)
  - 응답시간(response time)
    - 사용자에 대한 응답 시간 최소화 (사용자 입장)

CPU 스케줄링의 기준(criteria)

- 스케줄링 알고리즘의 다양한 목표와 평가 기준 (2/2)
  - 대기시간(waiting time)
    - 스레드가 준비 리스트에서 CPU를 할당 받을 때까지 기다리는 시간을 최소화 (운영체제와 사용자 입장)
  - 소요 시간(turnaround time)
    - 프로세스(스레드)가 컴퓨터 시스템에 도착한 후(혹은 생성된 후) 완료될 때까지 걸린 시간 (사용자 입장)
  - 시스템 정책(policy enforcement) 우선
    - CPU 스케줄링이 시스템의 정책에 맞도록 이루어 져야함
    - 예) 실시간 시스템에서는 스레드가 완료 시한(deadline) 내에 이루어지도록 하는 정책
    - 예) 급여 시스템에서는 안전을 관리하는 스레드를 우선 실행하는 정책 등
  - 자원 활용률(resource efficiency)
    - CPU나 I/O 장치 등 자원이 놀지 않도록 자원 활용률을 극대화 하는 것

타임 슬라이스

- 대부분 운영체제에서는 하나의 스레드가 너무 오래 CPU를 사용하도록 허용하지 않음

- 타임 슬라이스와 스케줄링
  - 운영체제는 스레드가 CPU를 사용할 타임 슬라이스를 정하고 해당 시간 동안 CPU를 사용하게 함
  - 타임 슬라이스(time slice)
    - 스케줄된 스레드에게 한 번 할당하는 CPU 시간
      - 스레드가 CPU 사용을 보장받는 시간
    - 커널이 스케줄을 단행하는 주기 시간
      - 타이머 인터럽트의 도움을 받아 타임 슬라이스 단위로 CPU 스케줄링
    - 타임 퀀텀(time quantum), 타임 슬롯(time slot)이라고도 함

CPU 스케줄링이 실행되는 4가지 상황

- CPU 스케줄링은 언제 시행될까?

1. 스레드가 시스템 호출 끝에 I/O를 요청하여 블록될 때
    - 스레드를 블록 상태로 만들고 스케줄링
    - (CPU 활용률 향상 목적)

2. 스레드가 자발적으로 CPU를 반환할 때
    - yield() 시스템 호출 등을 통해 스레드가 자발적으로 CPU 반환
    - 커널은 현재 스레드를 준비 리스트에 넣고, 새로운 스레드 선택
    - (CPU의 자발적 양보)

3. 스레드의 타임 슬라이스가 소진되어 타이머 인터럽트 발생
    - (균등한 CPU 분배 목적)

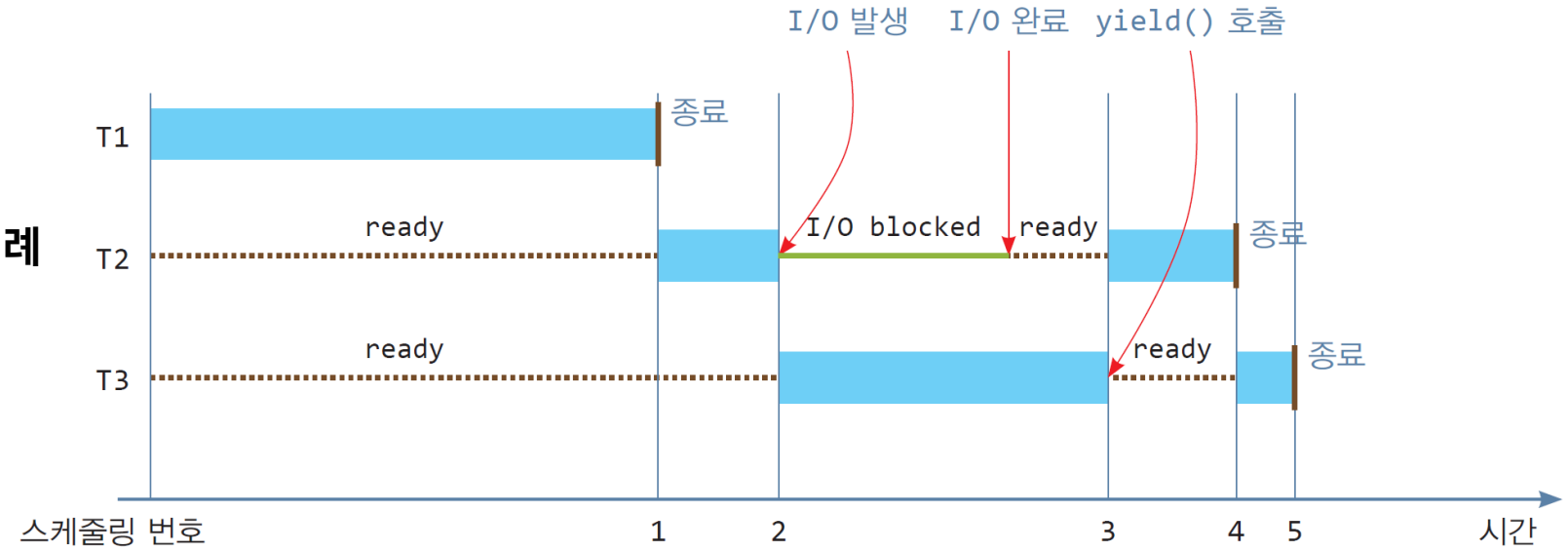
4. 더 높은 순위의 스레드가 요청한 입출력 작업 완료, 인터럽트 발생
    - 현재 스레드를 강제 중단(preemption)시켜 준비 리스트에 넣고
    - 높은 순위의 스레드를 깨워 스케줄링
    - (우선순위를 지키기 위한 목적)

선점 스케줄링과 비선점 스케줄링

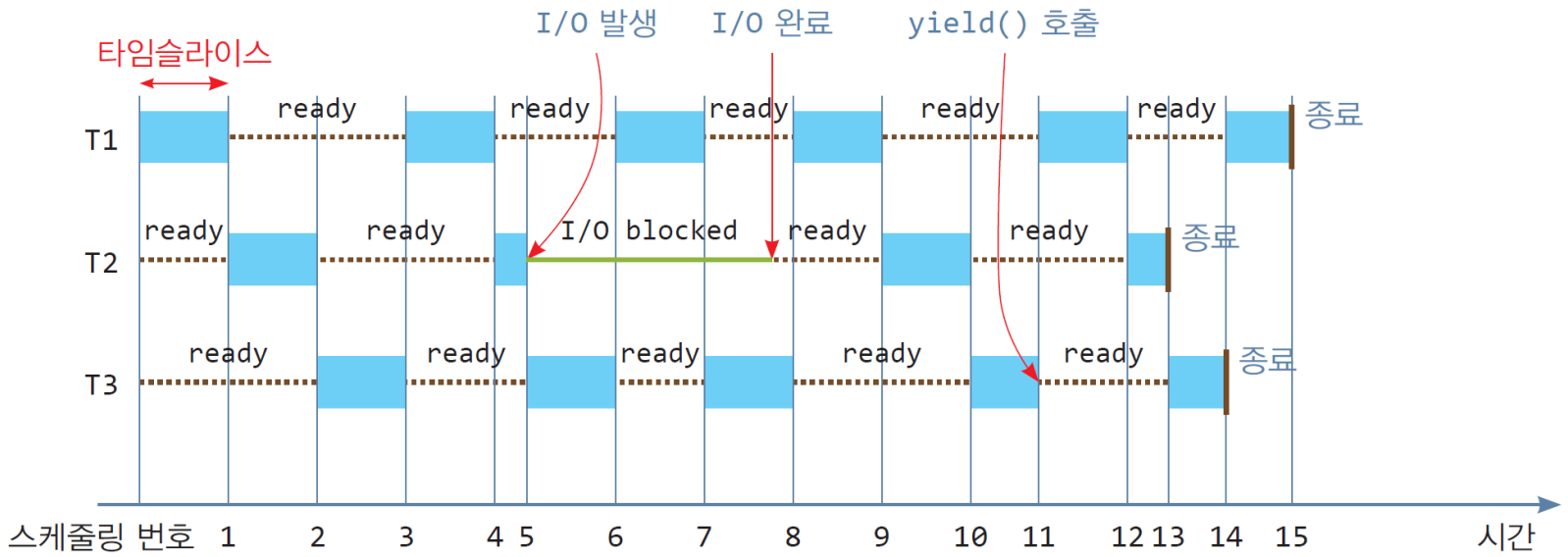
- 실행중인 스레드의 강제 중단 여부에 따른 CPU 스케줄링
  - **비선점 스케줄링**(non-preemptive scheduling) 타입
    - 현재 실행중인 스레드를 **강제로 중단시키지 않는** 타입
      - 스레드가 CPU를 할당 받아 실행을 시작하면, 완료되거나 CPU를 더 이상 사용할 수 없는 상황이 될 때까지 스레드를 강제 중단시키지 않음
    - 스케줄링 시점
      - CPU를 더 이상 사용할 수 없게 된 경우: I/O로 인한 블록 상태, sleep () 함수
      - 자발적으로 CPU 양보할 때: yield() 시스템 호출
      - 종료할 때
  - **선점 스케줄링**(preemptive scheduling) 타입
    - 현재 실행중인 스레드를 **강제 중단**시키고 다른 스레드 선택
    - 스케줄링 시점
      - 타임슬라이스가 소진되어 타이머 인터럽트가 발생될 때
      - 인터럽트나 시스템 호출 종료 시점에서, 더 높은 순위의 스레드가 준비 상태일 때
- 오늘날
  - 일부 실시간 임베디드 시스템 운영체제 – 비선점 스케줄링
  - 그 외 대부분의 운영체제 - 선점 스케줄링

비선점 스케줄링과 선점 스케줄링 비교

(a) 비선점 스케줄링 사례



(b) 선점 스케줄링 사례



기아와 에이징

- 기아(starvation)
  - 스레드가 스케줄링에서 선택되지 못한 채 오랜 동안 준비 리스트에 있는 상황
  - 사례
    - 우선순위를 기반으로 하는 시스템에서, 더 높은 순위의 스레드가 계속 시스템에 들어오는 경우
    - 짧은 스레드를 우선 실행시키는 시스템에서, 자신보다 짧은 스레드가 계속 도착하는 경우
  - 스케줄링 알고리즘 설계 시 기아 발생을 면밀히 평가
    - 기아가 발생하지 않도록 설계하는 것이 바람직함
- 에이징(aging)
  - 기아의 해결책
  - 스레드가 준비 리스트에 머무르는 시간에 비례하여 스케줄링 순위를 높이는 기법
    - 오래 기다릴 수는 있지만 언젠가는 가장 높은 순위에 도달하는 것 보장

기본적인 CPU 스케줄링 알고리즘들

- FCFS(First Come First Served)(비선점 스케줄링)
  - 도착한 순서대로 처리
- Shortest Job First(비선점 스케줄링)
  - 가장 짧은 스레드 우선 처리
- Shortest Remaining Time First(선점 스케줄링)
  - 남은 시간이 짧은 스레드가 준비 큐에 들어오면 이를 우선 처리
- Round-Robin(선점 스케줄링)
  - 스레드들을 돌아가면서 할당된 시간(타임 슬라이스)만큼 실행

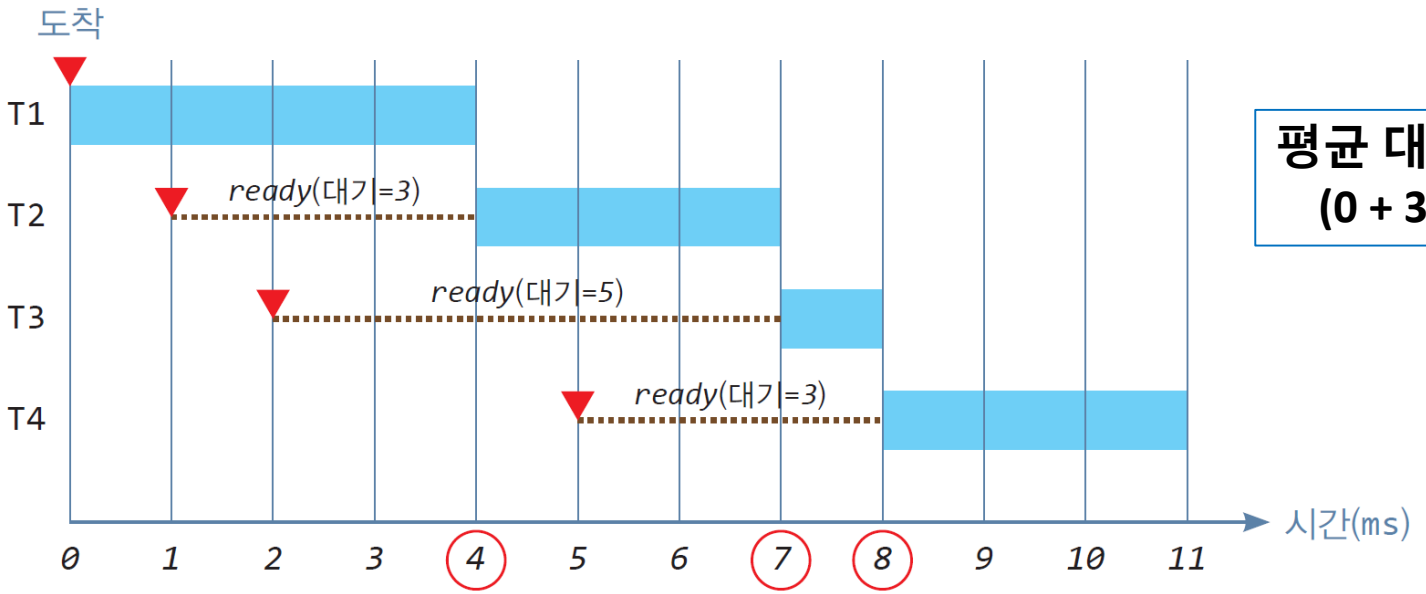
FCFS(First Come First Served)

- 알고리즘
  - 선입선처리
    - 먼저 도착한(큐의 맨 앞에 있는) 스레드 먼저 스케줄링
- 스케줄링 파라미터 : 스레드 별 큐 도착 시간
- 스케줄링 타입 : 비선점 스케줄링
- 스레드 우선순위 : 없음
- 기아 : 발생하지 않음
    - 스레드가 오류로 인해 무한 루프를 실행한다면, 뒤 스레드 기아 발생
- 성능 이슈
  - 처리율 낮음
  - 호위 효과(convoy effect) 발생
    - 긴 스레드가 CPU를 오래 사용하면, 늦게 도착한 짧은 스레드 오래 대기

FCFS 예

스레드	도착 시간	실행 시간(ms)
T1	0	4
T2	1	3
T3	2	1
T4	5	3

실행 시간 동안 입출력은 발생하지 않는다고 가정한다.



평균 대기 시간 :
(0 + 3 + 5 + 3)/4 = 11/4 = 2.75ms

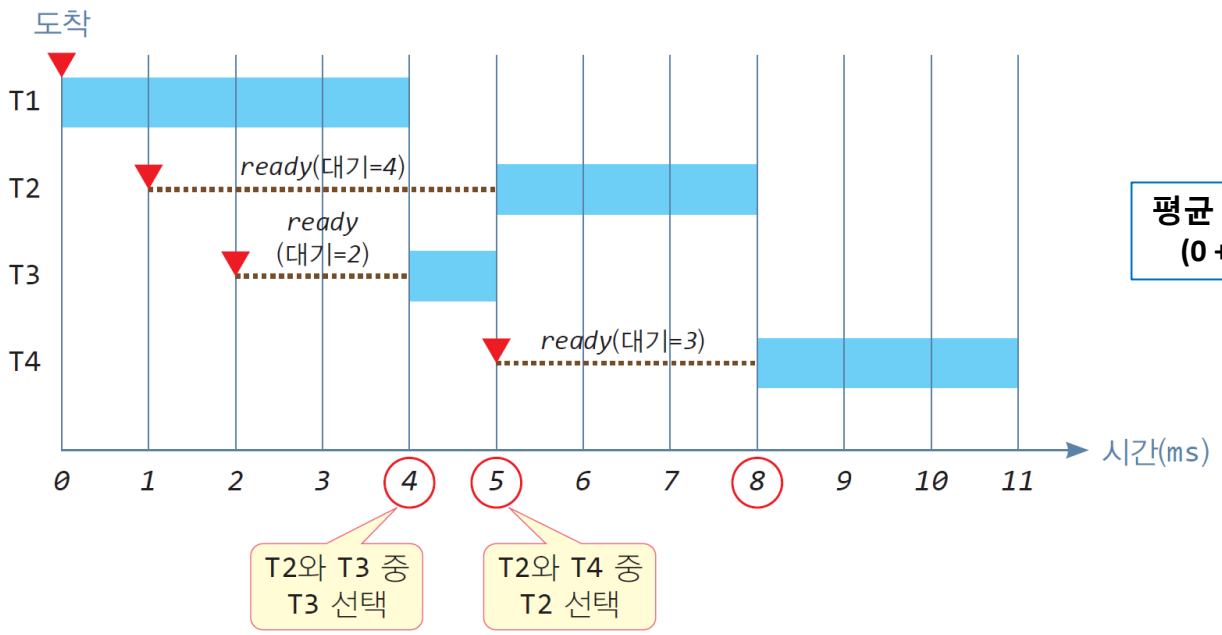
※ 빨간 원은 스케줄링이 일어나는 시점을 나타낸다.

SJF(Shortest Job First)

- 알고리즘
  - 최단 작업 우선 스케줄링
  - 실행 시간(예상 실행 시간)이 가장 짧은 스레드 선택해 평균 대기 시간을 최소화 하는데 목적
- 스케줄링 파라미터 : 스레드 별 예상 실행 시간
  - 스레드의 실행 시간을 정확히 예측하는 것은 불가능
- 스케줄링 타입 : 비선점 스케줄링
- 스레드 우선순위 : 없음
- 기아 : 발생 가능
  - 짧은 스레드가 계속 도착하면, 긴 스레드는 실행 기회를 언제 얻을 지 예측할 수 없음
- 성능 이슈
  - 가장 짧은 스레드가 먼저 실행되므로 평균 대기 시간 최소화
- 문제점
  - 실행 시간의 예측이 불가능하므로 현실에서는 거의 사용되지 않음

SJF 예

스레드	도착 시간	실행 시간(ms)
T1	0	4
T2	1	3
T3	2	1
T4	5	3



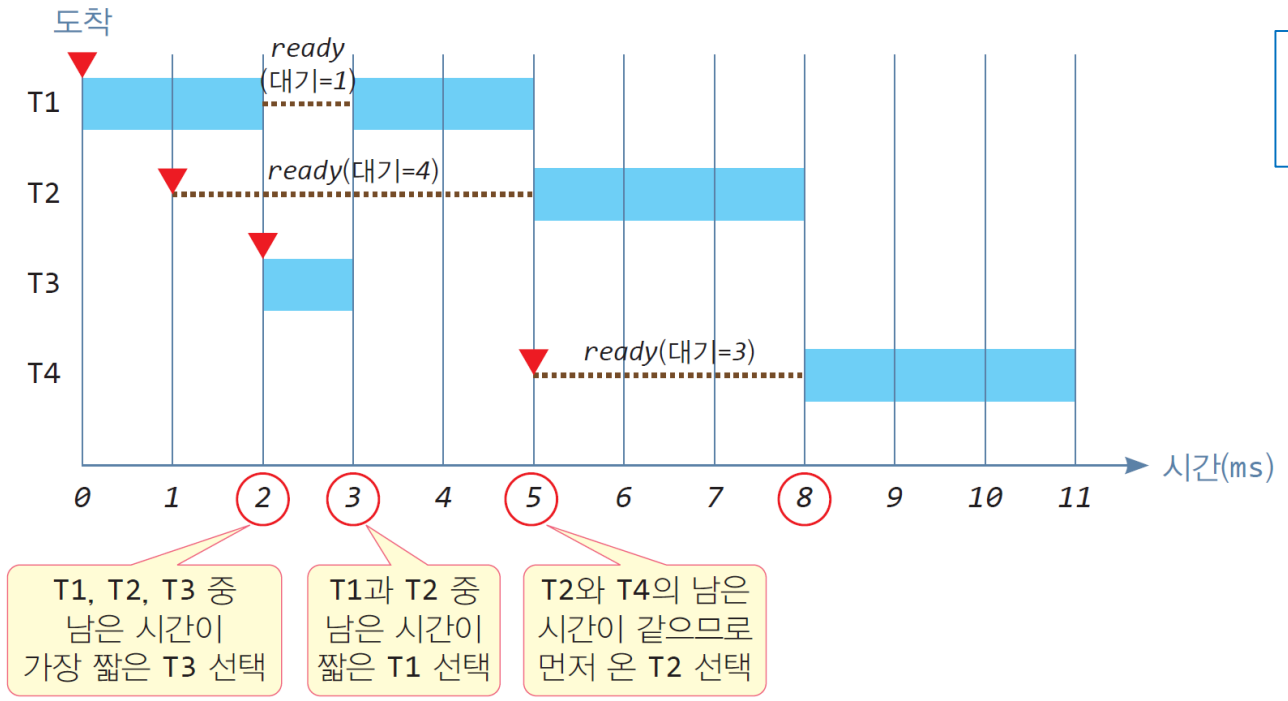
평균 대기 시간 :
(0 + 4 + 2 + 3)/4 = 9/4 = 2.25ms

SRTF(Shortest Remaining Time First)

- 알고리즘
  - 최소 잔여 시간 우선 스케줄링
  - 남은 실행 시간이 가장 짧은 스레드 선택
  - SJF의 선점 스케줄링 버전
    - 한 스레드가 끝나거나 실행 시간이 더 짧은 스레드가 도착할 때, 남은 실행 시간이 가장 짧은 스레드 선택
- 스케줄링 파라미터 : 스레드 별 예상 실행 시간과 남은 실행 시간 값
  - 이 시간을 정확히 예측하는건 불가능
- 스케줄링 타입 : 선점 스케줄링
- 스레드 우선순위 : 없음
- 기아 : 발생 가능
  - 짧은 스레드가 계속 도착하면, 긴 스레드는 실행 기회를 언제 얻을 지 모름
- 성능 이슈
  - 실행 시간이 가장 짧은 스레드가 먼저 실행되므로 평균 대기 시간 최소화
- 문제점
  - 실행 시간 예측이 불가능하므로 현실에서는 거의 사용되지 않음

SRTF 예

스레드	도착 시간	실행 시간(ms)
T1	0	4
T2	1	3
T3	2	1
T4	5	3



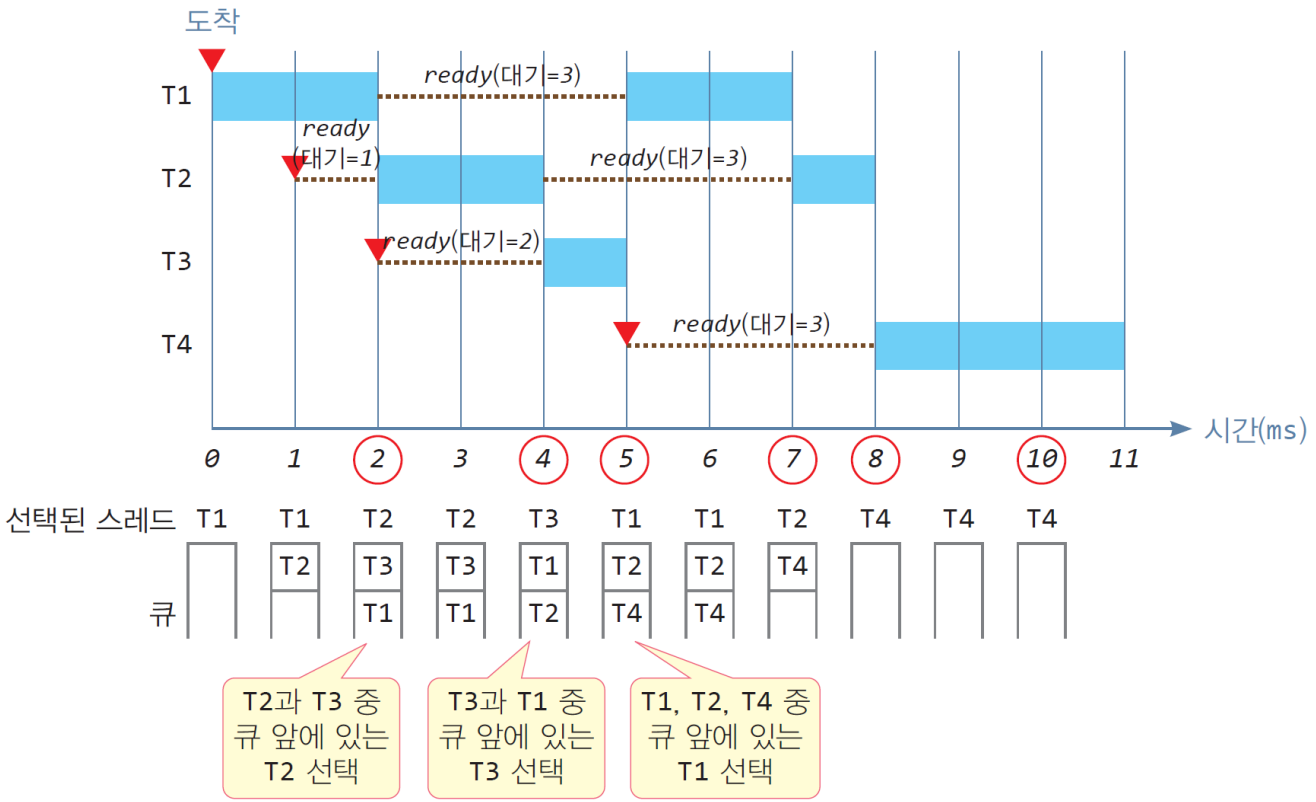
평균 대기 시간 :
(1 + 4 + 0 + 3)/4 = 8/4 = 2ms

RR(Round-Robin)

- 알고리즘
  - 스레드들에게 공평한 실행 기회를 주기 위해 큐에 대기중인 스레드들을 타임 슬라이스 주기로 돌아가면서 선택
- 스케줄링 파라미터 : 타임 슬라이스
- 스케줄링 타입 : 선점 스케줄링
- 스레드 우선순위 : 없음
- 기아 : 없음
  - 스레드의 우선순위가 없고, 타임 슬라이스가 정해져 있어, 일정 시간 후에 스레드는 반드시 실행
- 성능 이슈
  - 공평하고, 기아 현상 없고, 구현이 쉬움
  - 잦은 스케줄링으로 전체 스케줄링과 컨텍스트 스위칭에 대한 오버헤드 큼 (특히 타임 슬라이스가 작을 때 더욱 큼)

RR 예 (타임 슬라이스=2ms일 때)

스레드	도착 시간	실행 시간(ms)
T1	0	4
T2	1	3
T3	2	1
T4	5	3



평균 대기 시간 : (3 + 4 + 2 + 3)/4 = 12/4 = 3ms

스케줄링 : 6번(2, 4, 5, 7, 8, 10ms 때)

컨텍스트 스위칭 : 5번 발생(2, 4, 5, 7, 8ms 때)

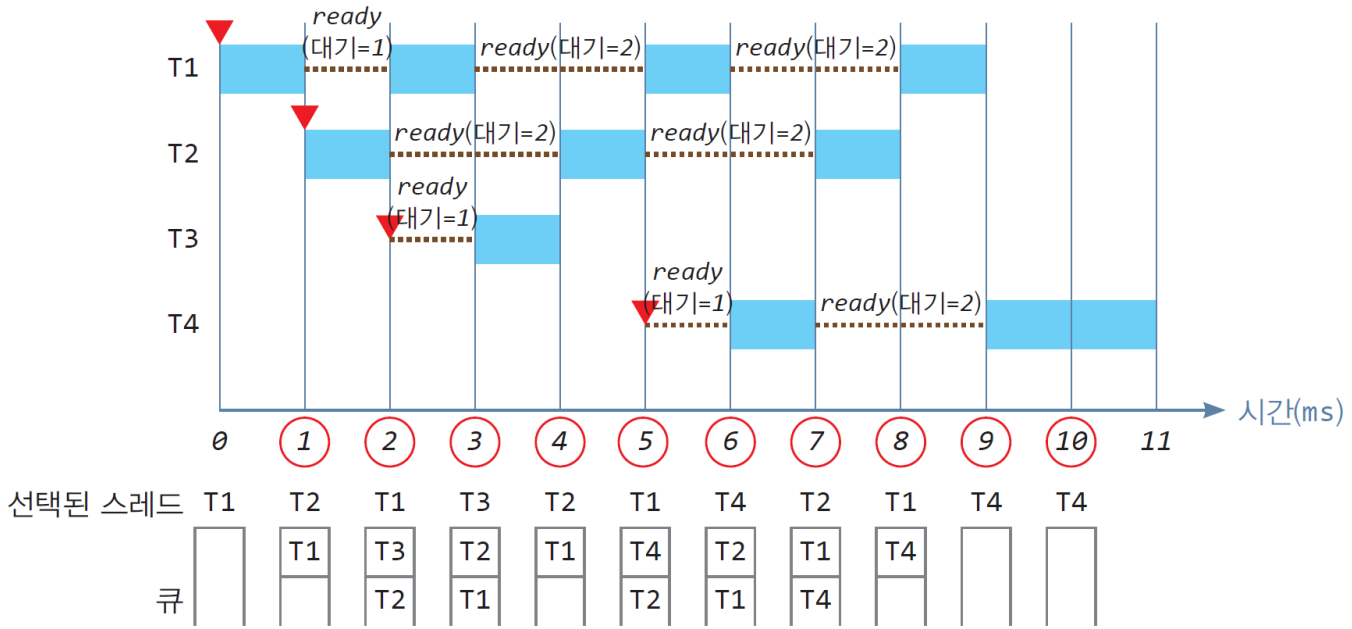
RR 예 (타임 슬라이스=1ms일 때)

스레드	도착 시간	실행 시간(ms)
T1	0	4
T2	1	3
T3	2	1
T4	5	3

평균 대기 시간 : (5 + 4 + 1 + 3)/4 = 13/4 = 3.25ms

스케줄링 : 10번(1, 2, 3, 4, 5, 6, 7, 8, 9, 10ms 시점)

컨텍스트 스위칭 : 9번(1, 2, 3, 4, 5, 6, 7, 8, 9ms 시점)



실시간 스케줄링

- Consider a control system in a future vehicle
  - Steering wheel sampled every 10 ms – wheel positions adjusted accordingly (computing the adjustment takes 4.5 ms of CPU time)
  - Breaks sampled every 4 ms – break pads adjusted accordingly (computing the adjustment takes 2ms of CPU time)
  - Velocity is sampled every 15 ms – acceleration is adjusted accordingly (computing the adjustment takes 0.45 ms)
  - For safe operation, adjustments must always be computed before the next sample is taken

Is it possible to ALWAYS compute
ALL adjustments IN TIME?

실시간 스케쥴링의 동작원리

Q. Will my real-time application really meet its timing constraints?

- Rate Monotonic (RM)

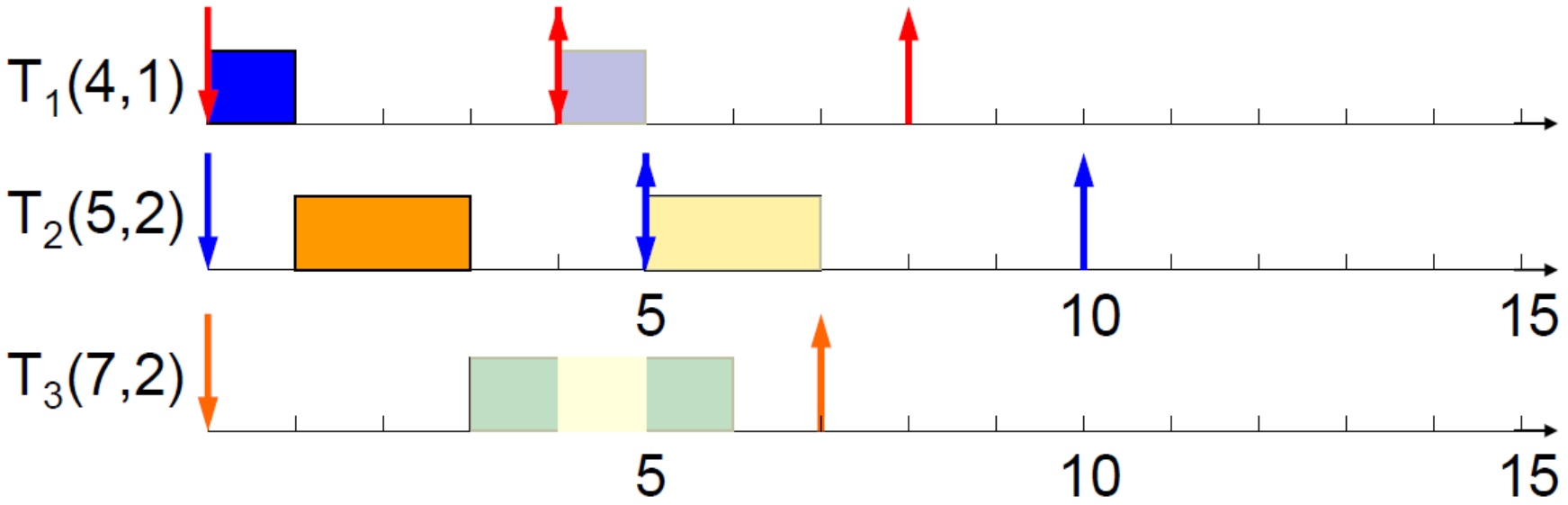
  Statically assign higher priorities to tasks with lower periods

- Earliest Deadline First (EDF)

  The earlier the deadline, the higher the priority

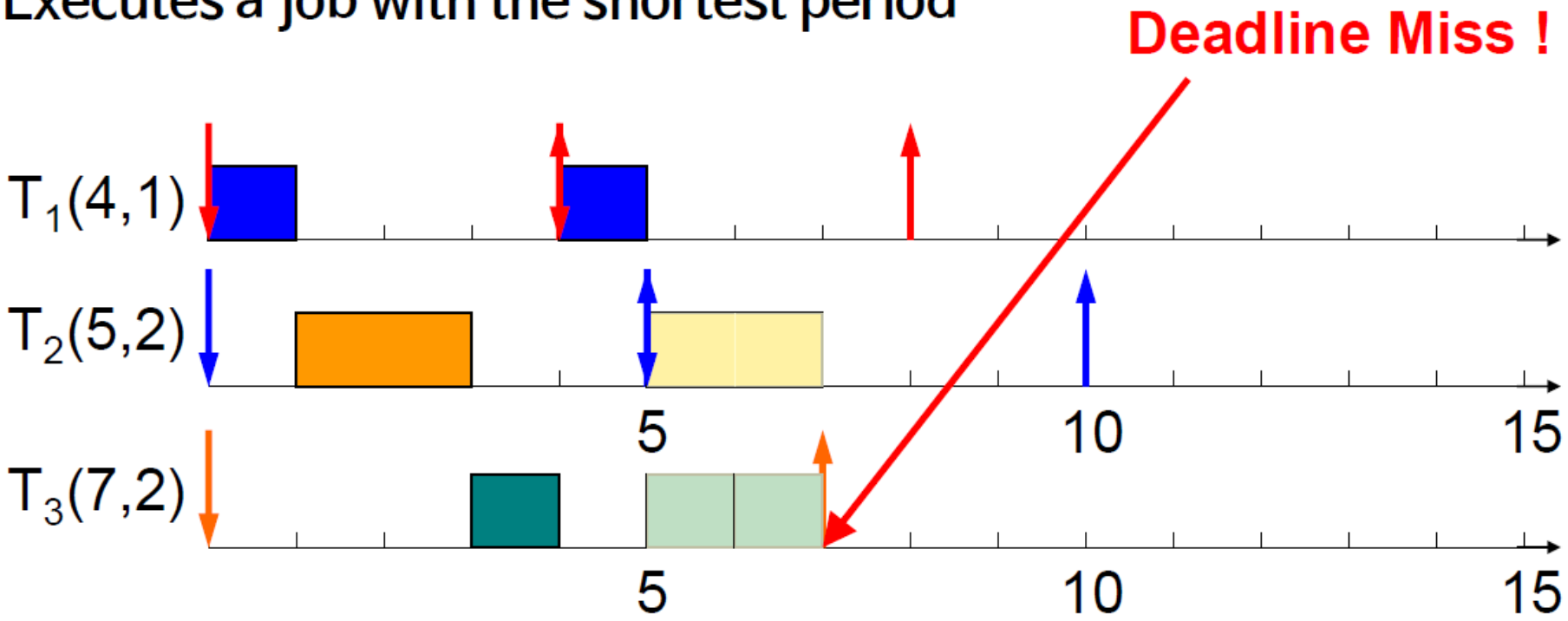
실시간 스케줄링: Rate-Monotonic Scheduling (1/2)

- Rate-Monotonic Scheduling (RMS)
  - Statically assigns priority according to period
  - A task with a shorter period has a higher priority
  - Executes a job with the shortest period



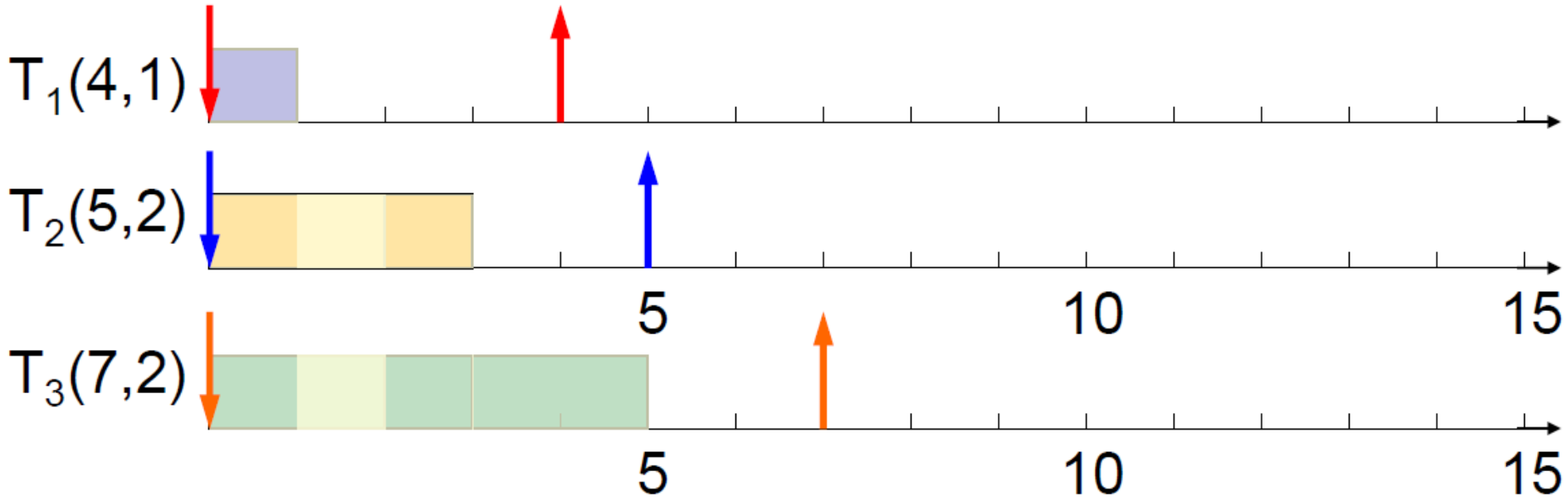
실시간 스케줄링: Rate-Monotonic Scheduling (2/2)

- Statically assigns priority according to period
- A task with a shorter period has a higher priority
- Executes a job with the shortest period



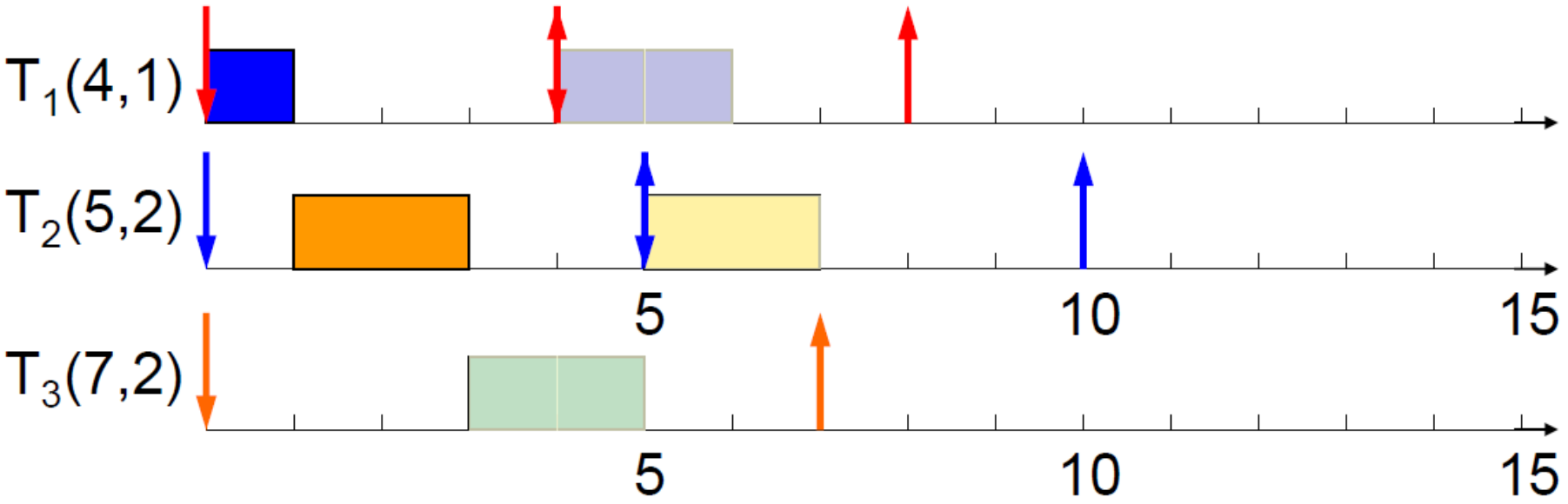
실시간 스케줄링: Earliest Deadline First (1/5)

- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline



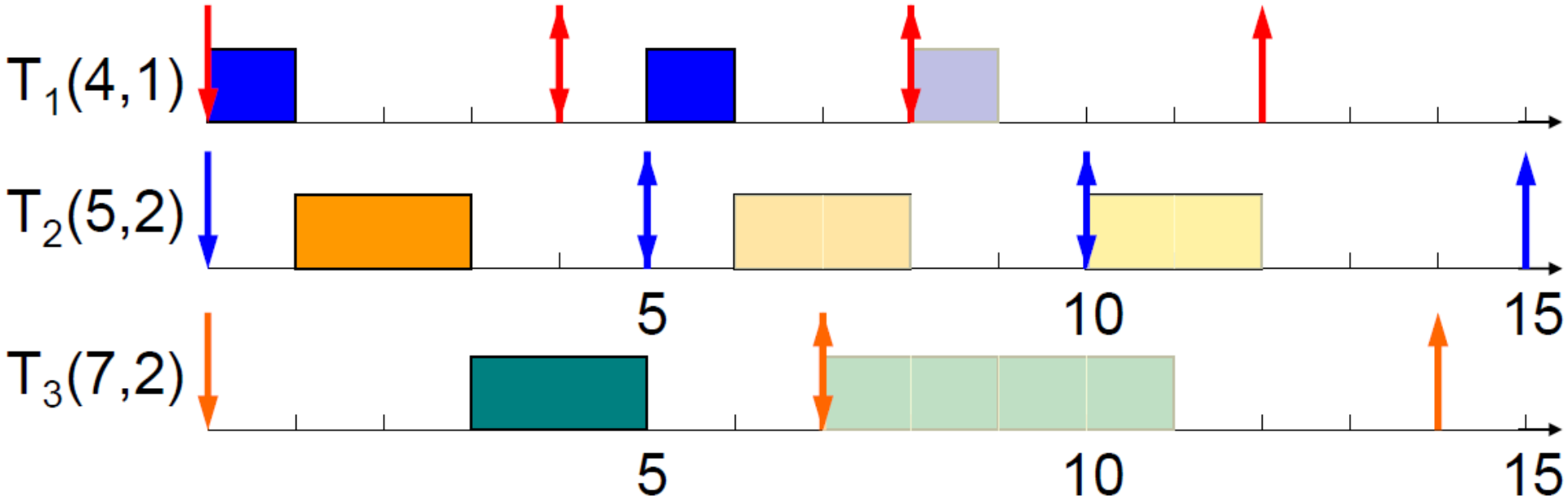
실시간 스케줄링: Earliest Deadline First (2/5)

- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline



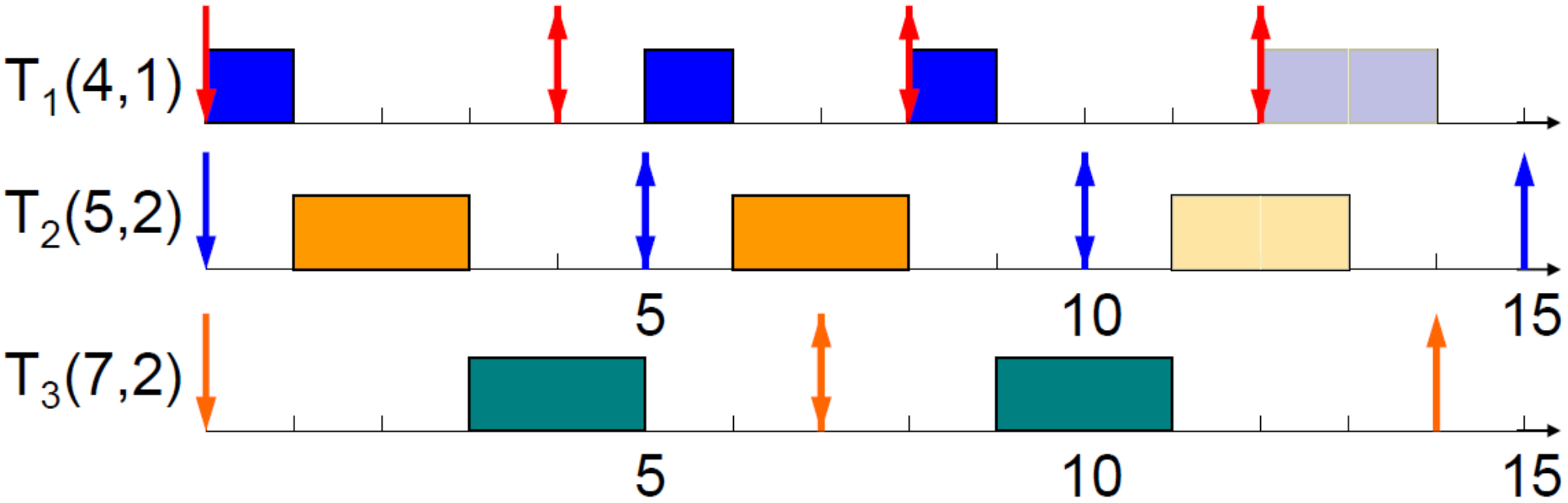
실시간 스케줄링: Earliest Deadline First (3/5)

- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline



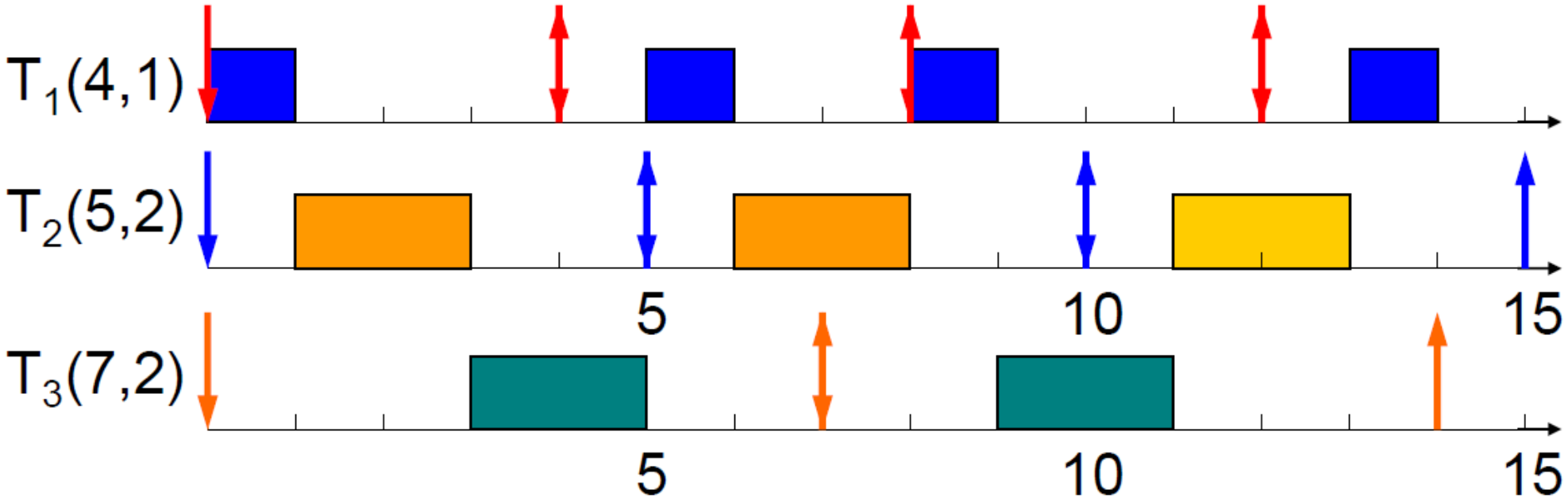
실시간 스케줄링: Earliest Deadline First (4/5)

- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline



실시간 스케줄링: Earliest Deadline First (5/5)

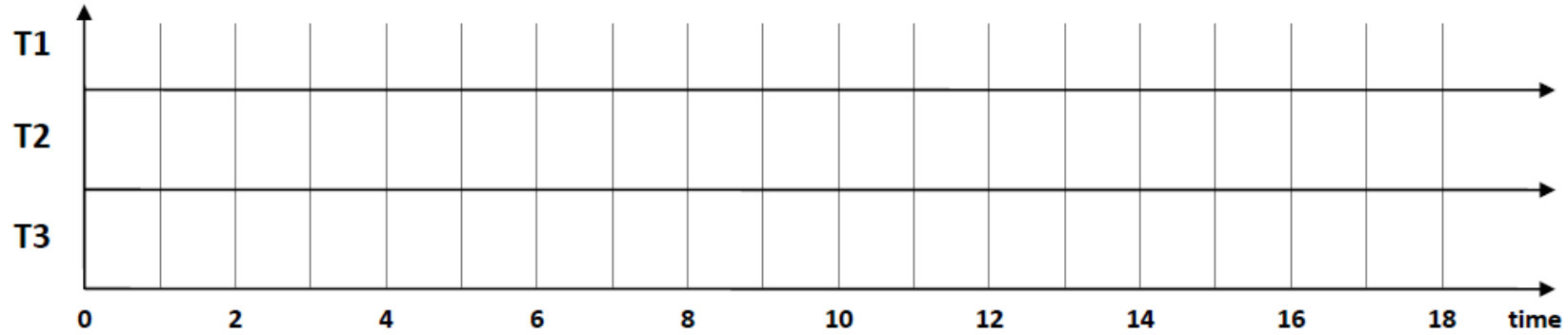
- Optimal dynamic priority scheduling
- A task with a shorter deadline has a higher priority
- Executes a job with the earliest deadline



실시간 스케줄링: Earliest Deadline First (예)

Task	Execution Time	Period
T1	1	3
T2	1	4
T3	2	7

RM



EDF