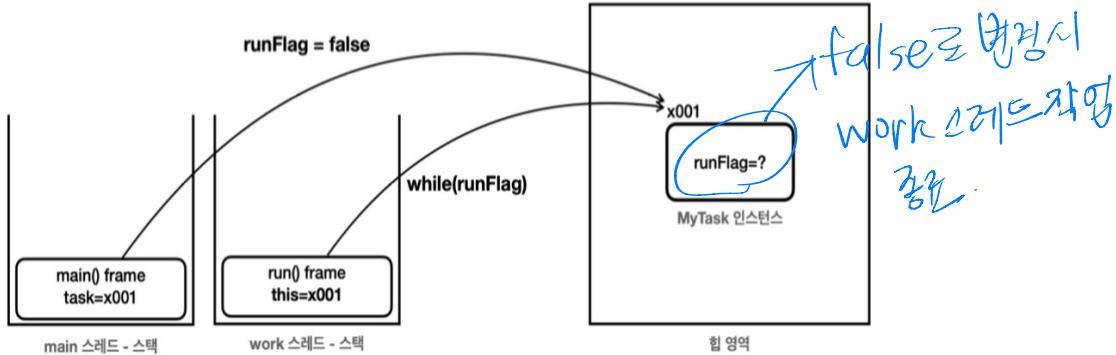


5장

· 멀티스레드 개시성

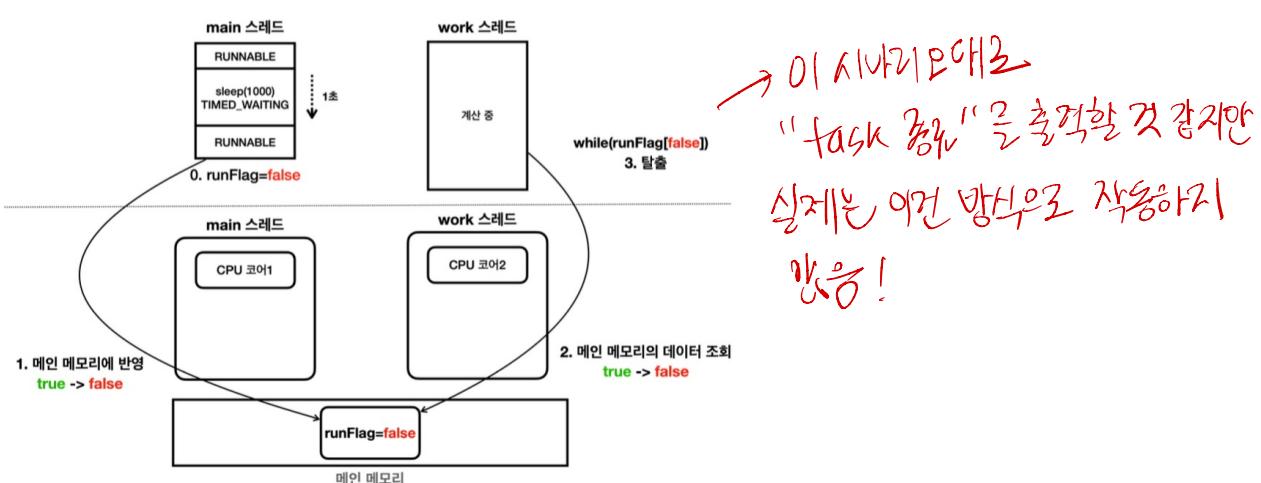
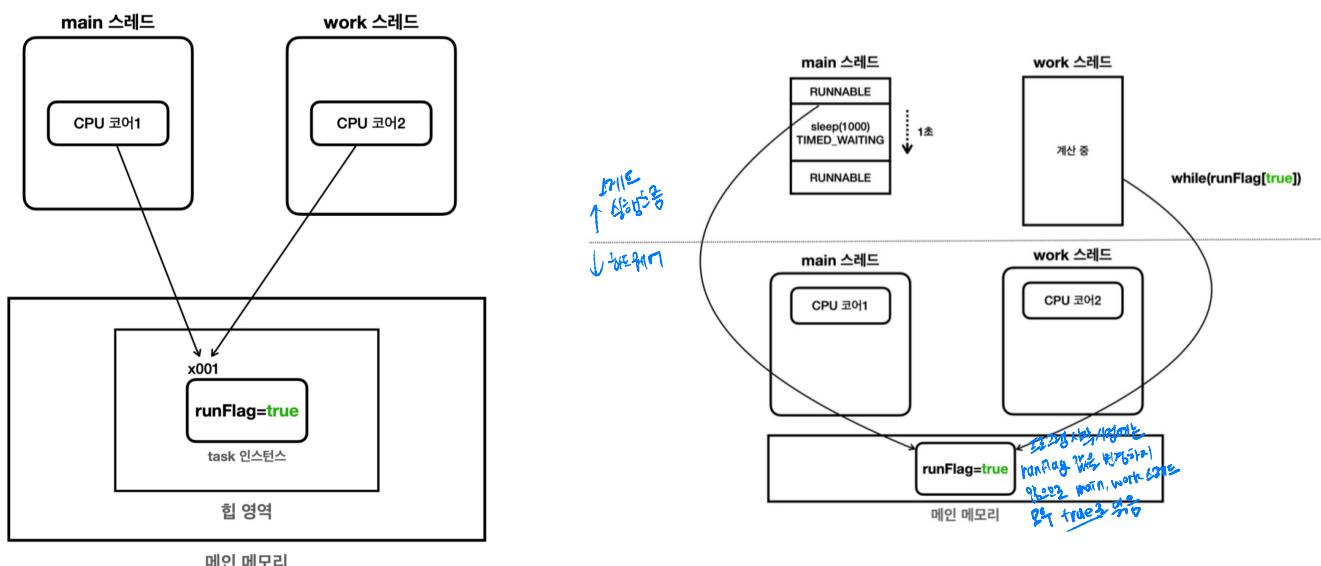
## Volatile

메모리 그림



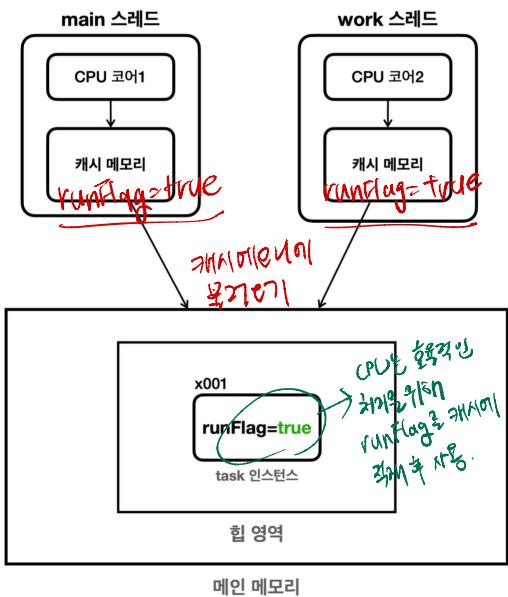
D/10/21 개시(성) 문제

<일반적으로 생각하는 메모리 접근방식>

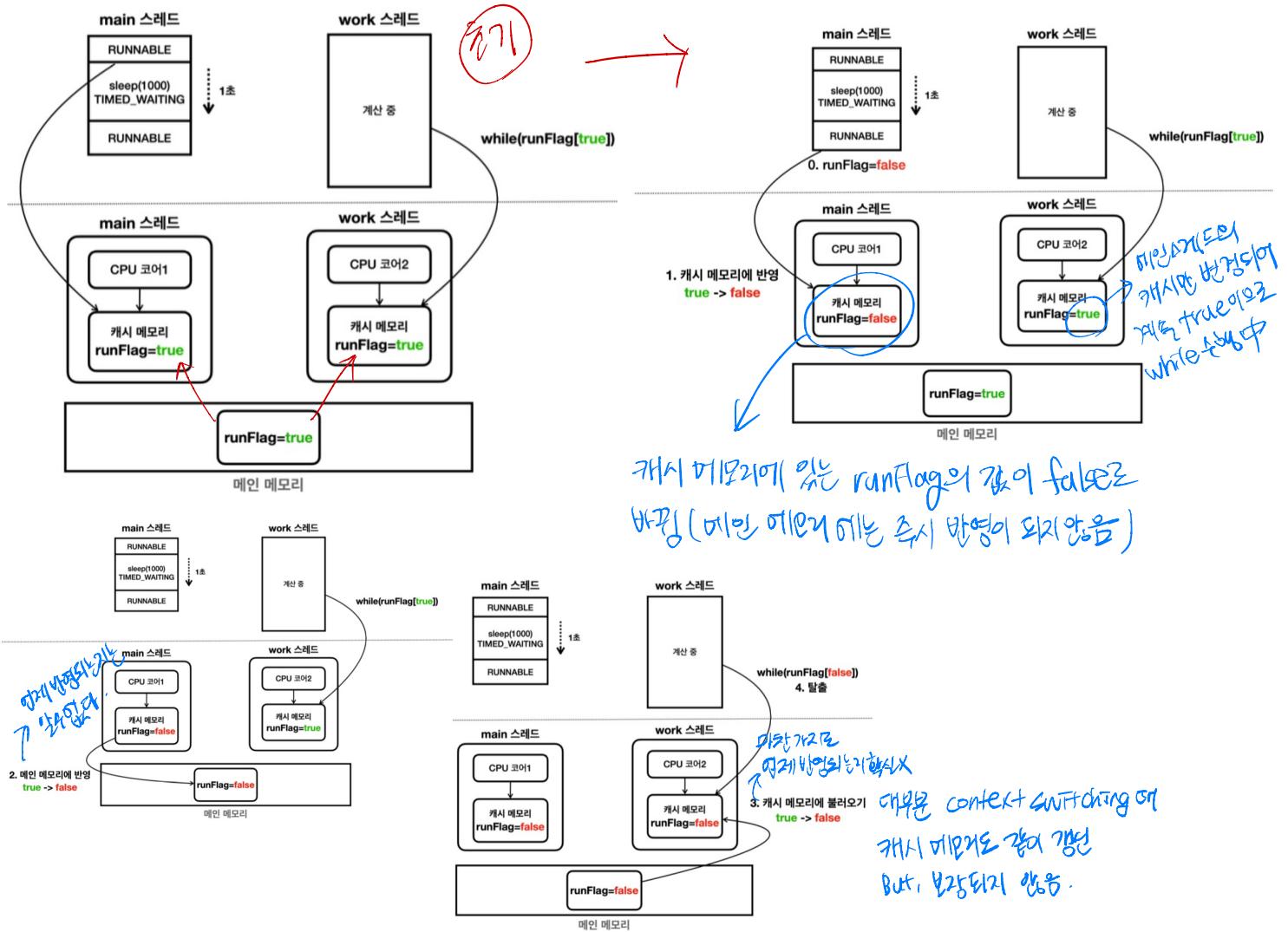


이 시나리오에서  
'task 종료'를 충족할 것 같지만  
실제는 이런 방식으로 작동하지  
않음!

# 〈상대 메모리 접근 방식〉 CPU는 처리 성능 개선을 위해 캐시 사용



매우 빠른 CPU 연산 속도를 자랑하며  
멀리 떨어진 메인 메모리 보다  
속도로 빠르고 가까운 캐시 메모리를 사용  
(코어 대신으로 캐시를 보유, 여러 코어가 공유하는  
캐시로 존재)



## 메모리 가시성 (memory visibility)

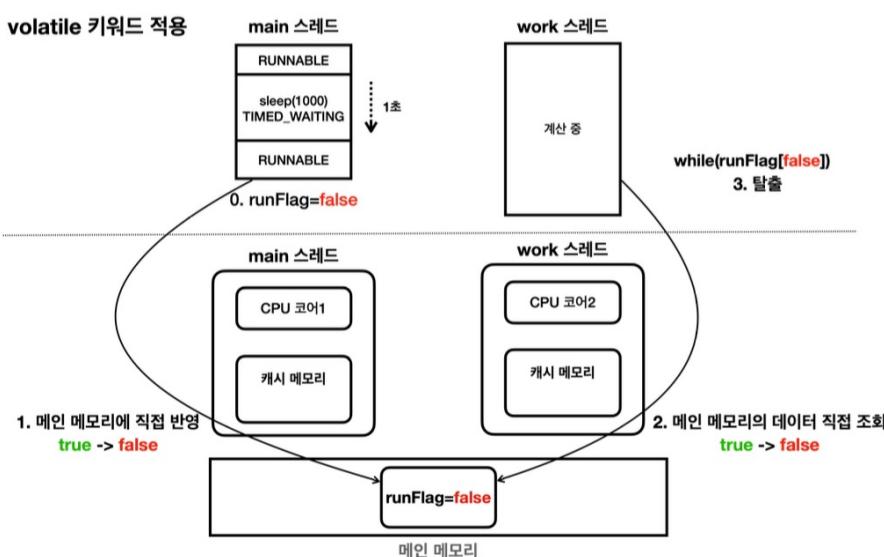
- (6) 멀티스레드 환경에서 한 스레드가 변경한 값이 다른 스레드에서  
언제 보이는지에 대한 문제  
↳ 그래서 쓰시보이게 하려면?

CPU 캐시에 캐시 사용시 성능은 개선되나, 대로는 멀티스레드  
에서 같은 시점에 정확히 같은 데이터를 보는것이 더 중요할수 있음

그래서 "Volatile" 키워드를 사용한다.

```
static class MyTask implements Runnable
    //boolean runFlag = true;
    volatile boolean runFlag = true;
```

runFlag는 더이상 캐시를 사용하지  
않고 read/write를 항상  
메인 메모리에 직접 접근하게 된다.



캐시 메모리를 채인 메모리에 반영하거나 빈 메모리의 변경 내용을 캐시 메모리에 다시 불리는 것은  
어제 발생할까? → CPU 설계, 실행 한정 차이有 → 즉시 반영되거나 나중에 되거나 평생 반영이 안될 수도...

- context switching 때 주로 경신되나 보장하지 않는다.
- 그래서 volatile 키워드를 쓰는 것이 학설하게 해결 가능.

## Java Memory Model

자바 프로그램이 어떻게 메모리에 접근하고 수정할 수 있는지 규정하여, 특히 멀티 스레드 프로그래밍에서  
스레드 간 상호작용을 정의한다.

핵심: 여러 스레드들의 작업 순서를 보장하는 happens-before 관계에 대한 정의

happens-before 관계: 자바 버전 1.5에서 스레드 간의 작업 순서를 정의하는 개념

ex) 작업 A가 작업 B보다 happens-before 관계에 있다면, 작업 A에서의 모든 메모리 변경 사항은  
작업 B에서 볼 수 있음 → 작업 A에서 변경된 내용은 작업 B 시작 전, 모두 메모리에 반영.

- happens-before 관계는 이름 그대로, 한 동작이 다른 동작보다 먼저 발생함을 보장한다.
- happens-before 관계는 스레드 간의 메모리 가시성을 보장하는 규칙이다.
- happens-before 관계가 성립하면, 한 스레드의 작업을 다른 스레드에서 볼 수 있게 된다.
- 즉, 한 스레드에서 수행한 작업을 다른 스레드가 참조할 때 최신 상태가 보장되는 것이다.

이 규칙은 따르면  
멀티 스레드 환경에서  
예상치 못한 동작  
허용 가능

happens-before 관계가 발생하는 경우

- 프로그램 순서 규칙 - 단일 스레드에서 프로그램의 순서대로 작성된 모든 명령은 happens-before 순서로 실행
- volatile 범주 규칙 - 한 스레드에서 volatile 범주에 대한 쓰기 작업은 해당 범주를 읽는 다른 스레드에 보임
  - ↳ volatile 범주 쓰기 작업이 범주를 읽는 것보다 happens-before 관계 위임
- 스레드 시작 규칙 - 한 스레드에서 Thread.start()를 호출하면, 해당 스레드 내의 모든 작업은 start() 호출 이후에 실행된 작업보다 happens-before 관계 위임

- 스레드 종료 규칙 - 한 스레드에서 `Thread.join()`을 호출하면 `join` 대상 스레드의 모든 작업은 `join()` 반복 후 작업한다 happens-before 관계를 가진다.
  - ↳ `Thread.join()` 호출전, `Thread`의 모든 작업이 완료되어야 하며 이 작업은 `join()` 한환 이후에 참조 가능
- 1 ~ 100 까지 더하는 sumTask 예제.
- 인터럽트 규칙 - 한 스레드에서 `Thread.interrupt()`을 호출하는 작업이 인터럽트된 스레드가 인터럽트를 강제하는 시점이 작업보다 happens-before 관계가 성립한다.
- 객체 생성 규칙 - 객체의 생성자는 마지막이 생성된 이후에만 다른 스레드에 의해 참조되도록 보장한다.
- 면티릭 규칙 - 한 스레드에서 `synchronized` 블록 종료후, 그 면티리를 얻는 모든 스레드는 해당 블록 내의 모든 작업을 볼 수 있다.
- 전이 규칙 - 만약 A가 B보다 happens-before 관계에 있고 B가 C보다 happens-before 관계에 있다면, A는 C보다 happens-before 관계에 있다.

## 〈비밀리 가시성 part 정리〉

Volatile, 블록 동기화 기법 (`synchronized`, `ReentrantLock`) 사용시 비밀리 가시성 문제가 발생하지 않는다.