# 운영체제 – 스레드 동기화

# (Thread Synchronization)

1871227 임세진

https://youtu.be/75UEq6TaXq4

HSU 한성대학교 HANSUNG UNIVERSITY

CryptoCraft LAB
https://crypto.modoo.at

# Contents



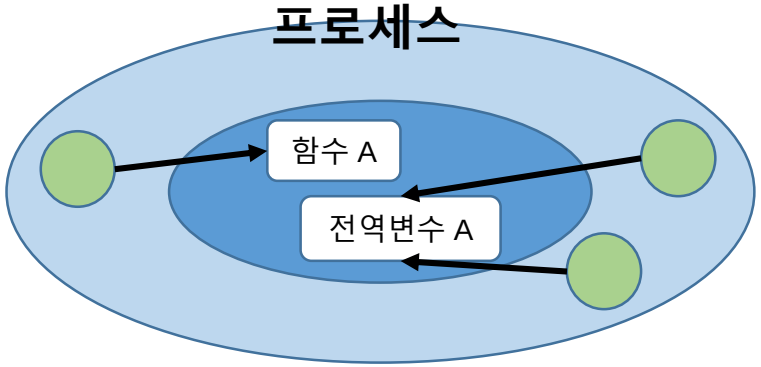
CryptoCraft LAB

# 01. 스레드 동기화의 필요성

# 01. 스레드 동기화의 필요성

- 스레드 : 프로세스보다 작은 크기의 운영체제 실행 단위

- 프로세스마다 정보를 관리하듯, 스레드마다 정보를 관리하여 생성, 소멸, 스케줄링 등 독립된 단위로 다룸




컨테이너 역할

- 프로세스는 스레드에게 공유 환경을 제공함

- 한 프로세스 내에 속한 스레드들은 프로세스 내에 작성된 함수를 호출할 수 있고 전역변수를 액세스할 수 있음

# 01. 스레드 동기화의 필요성

- 멀티태스킹 : 다수의 작업을 동시에 실행시키는 응용프로그램 작성 기법

- 병렬성을 높이고 실행시간 및 응답시간을 낮출 수 있음

- 문제점 : 다수의 작업(스레드)들이 공유 데이터에 동시에 접근하면 공유 데이터가 훼손되어 문제가 발생할 수 있음

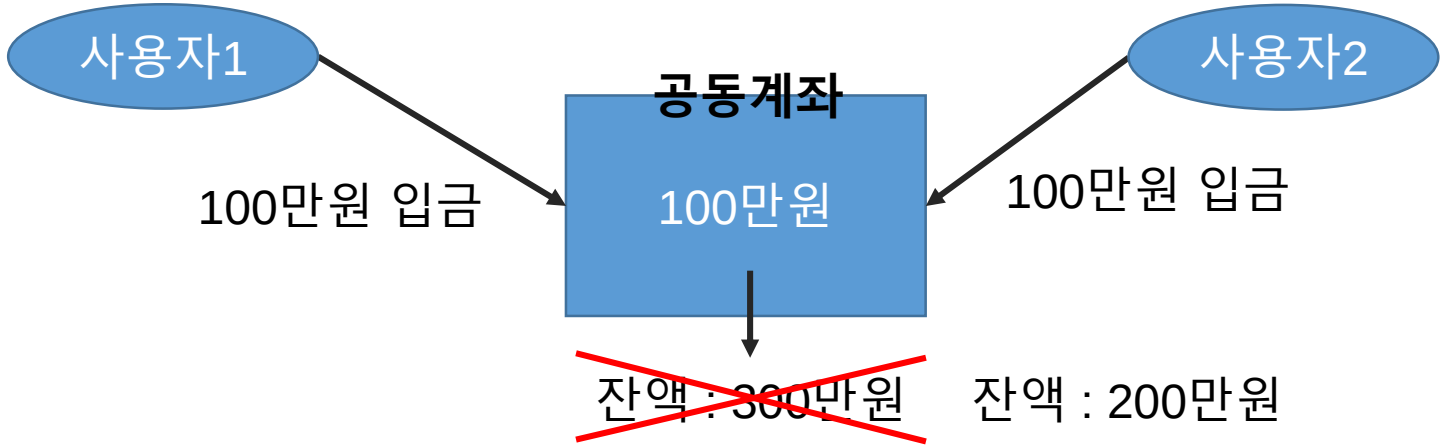
- 이러한 공유 데이터에 대한 동시 접근을 해결하는 방법이 동기화임

# 01. 스레드 동기화의 필요성

- 스레드 동기화 : 공유 데이터에 접근하고자 하는 다수의 스레드가 충돌없이 공유 데이터에 접근하기 위해 상호 협력하는 것

=> 한 스레드가 공유 데이터에 대해 배타적이고 독점적으로 접근하도록 함

- 공유  데이터 문제 (스레드 동기화가 없다면)

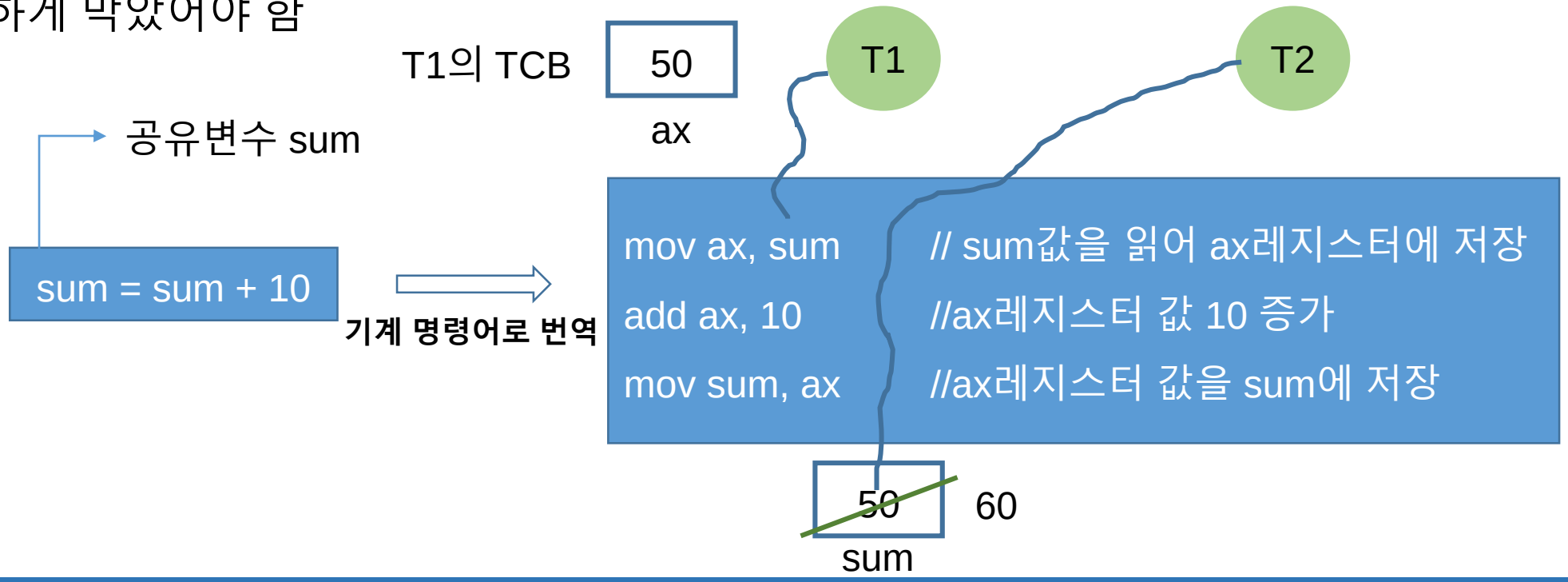


# 01. 스레드 동기화의 필요성

- 공유 데이터 문제의 원인

⇒ ‘sum = sum +10’ 이 하나의 CPU 명령이 아님

⇒ 한 스레드가 ‘sum = sum +10’의 실행을 완전히 마칠 때까지 다른 스레드가 이 코드를 실행하지 못하게 막았어야 함



# 01. 스레드 동기화의 필요성

- 공유 데이터 접근 문제의 해결책

- 문제의 핵심 : 여러 스레드가 공유 변수에 접근하려고 경쟁함

- 해결책 : 동기화

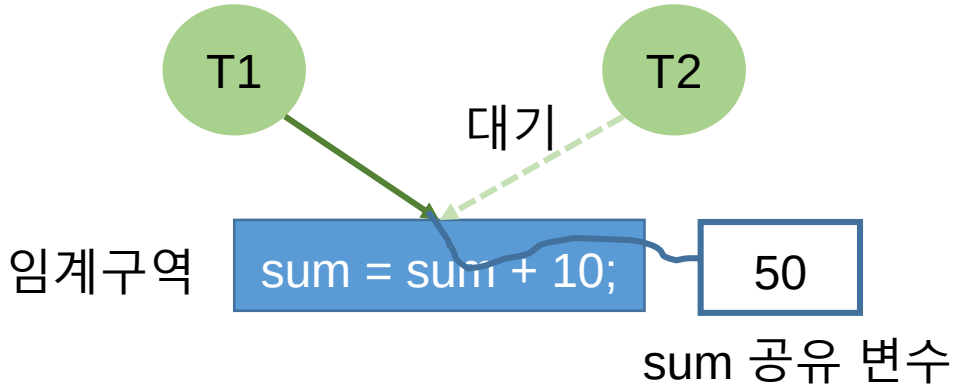
⇒ 한 순간에 하나의 스레드만 공유 데이터에 접근하도록 함

⇒ 한 스레드가 공유 데이터에 접근을 완전히 마칠 때까지 다른 스레드가 공유 데이터에 접근하지 못하도록 제어

# 01. 스레드 동기화의 필요성

- 임계구역 (critical section)

: 공유 데이터에 접근하는 프로그램 코드




- 상호배제 (mutual exclusion)

: 임계구역을 오직 한 스레드만 배타적 독립적으로 사용하도록 보장하는 기법

# 02. 상호배제
# (mutual exclusion)

# 02. 상호배제 (mutual exclusion)

- 공유 데이터를 다루는 프로그램 코드의 일반적인 구조

… 일반코드 …

**Entry 코드 (임계구역진입코드)**

**… 임계구역코드 …**

**Exit 코드 (임계구역진출코드)**

… 일반코드 …

- Entry code : 임계구역에 진입하기 전에 다른 스레드가 임계구역에 있는지 검사

⇒ 스레드가 없다면 다른 스레드가 들어오지 못하도록 조치를 취하고,

⇒ 스레드가 있다면 그 스레드가 임계구역을 벗어날 때까지 현재 스레드를 대기시킴

- Exit code : 대기 중인 스레드나 다른 스레드가 임계구역에 진입할 수 있도록 entry 코드에서 취한 조치를 푸는 코드
- 임계구역 코드 : 공유 데이터를 접근하는 코드를 포함하며, 한 스레드에 의해서만 배타적으로 사용되도록 보장되어야 하는 코드

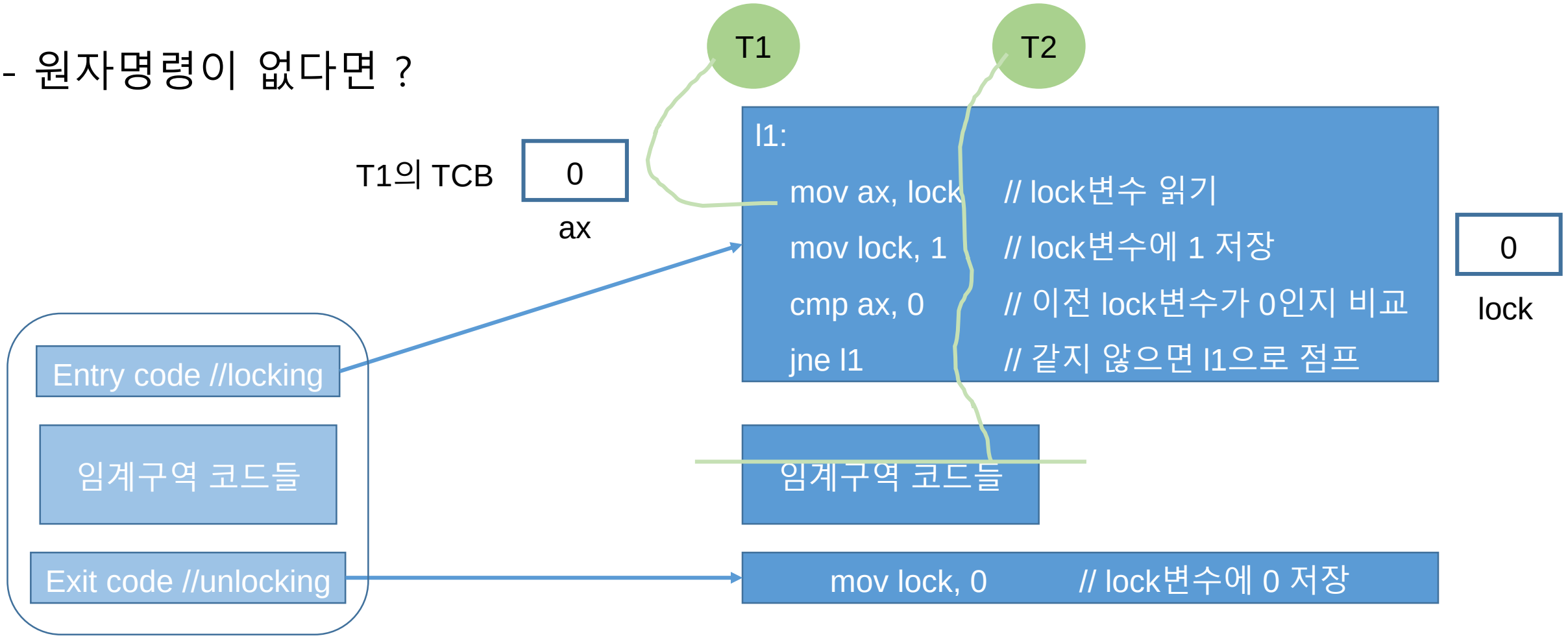
# 02. 상호배제 (mutual exclusion)

- 상호배제 구현
  - 소프트웨어적 방법 : Peterson’s 알고리즘 등
  - 하드웨어적 방법 : 인터럽트 서비스 금지, 원자 명령

- 원자 명령 (atomic instruction) : 상호배제를 위해 특별히 설계된 CPU명령

# 02. 상호배제 (mutual exclusion)

- 원자명령이 없다면 ?



# 02. 상호배제 (mutual exclusion)

- 해결법 : 원자 명령 (atomic instruction) 도입

앞의 문제가 생겼던 두 명령을 하나의 명령으로 만듦

# 03. 멀티스레드 동기화 기법

# 03. 멀티스레드 동기화 기법

- 스레드 동기화 기법
  - 락 (lock) 방식 : 뮤텍스 (mutex), 스핀락 (spinlock)
  - wait-signal 방식 : 세마포 (semaphore)

- 락 방식 : lock 변수를 두고 lock을 잠근 스레드만 임계구역에 진입하여 공유자원을 사용하도록 하는 기법
  - 뮤텍스 : 락을 소유하지 않은 스레드가 대기 큐에서 잠자면서 락이 풀리기를 기다리는 방식
  - 스핀락 : 락을 소유하지 않은 스레드가 락이 풀릴 때까지 무한히 락을 검사하는 방식

# 03. 멀티스레드 동기화 기법

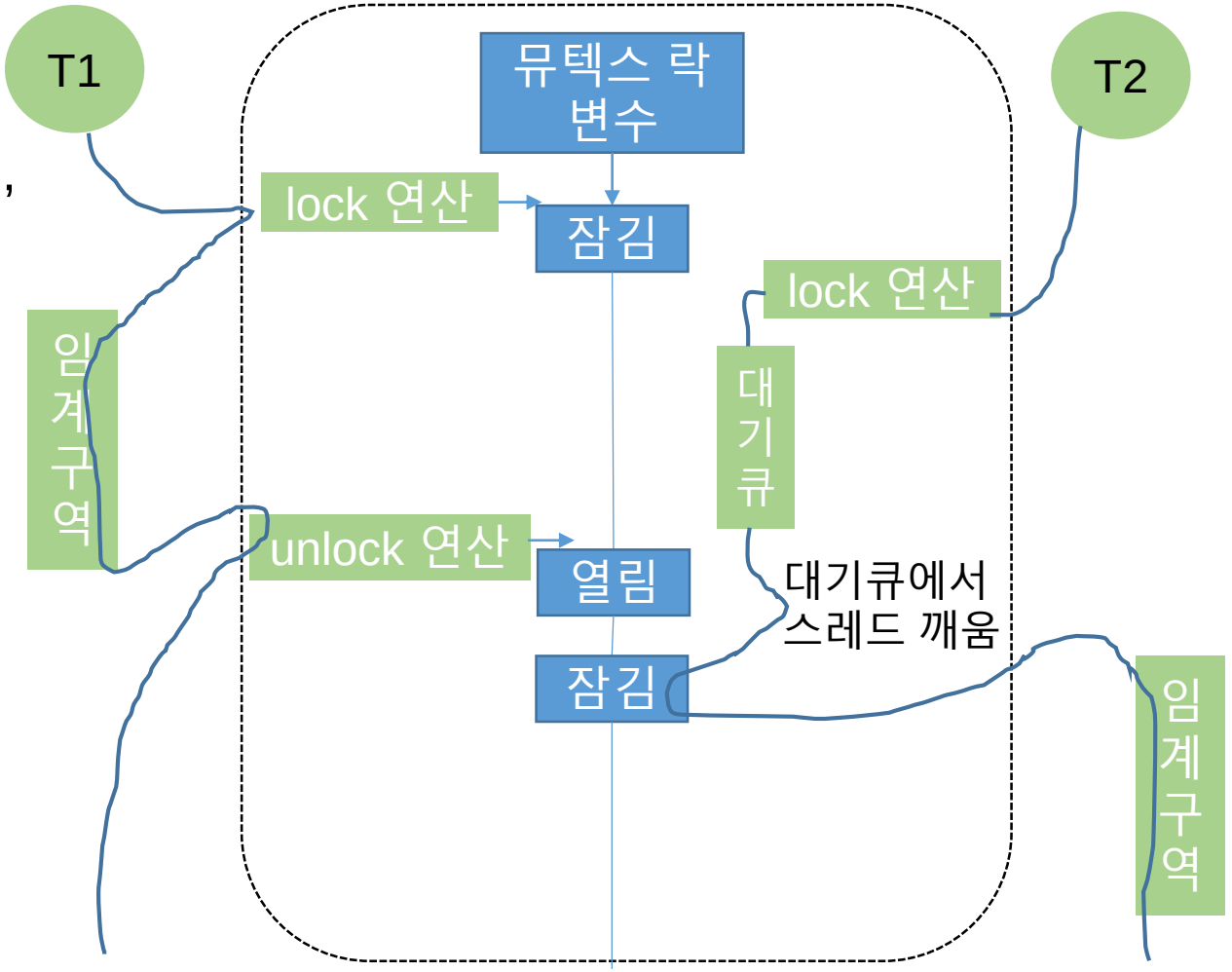
- 뮤텍스

: 임계구역의 실행시간이 짧을 경우,

락이 잠겨있는 시간보다 스레드가

잠자고 깨는데 걸리는 시간이 더

크기 때문에 비효율적임

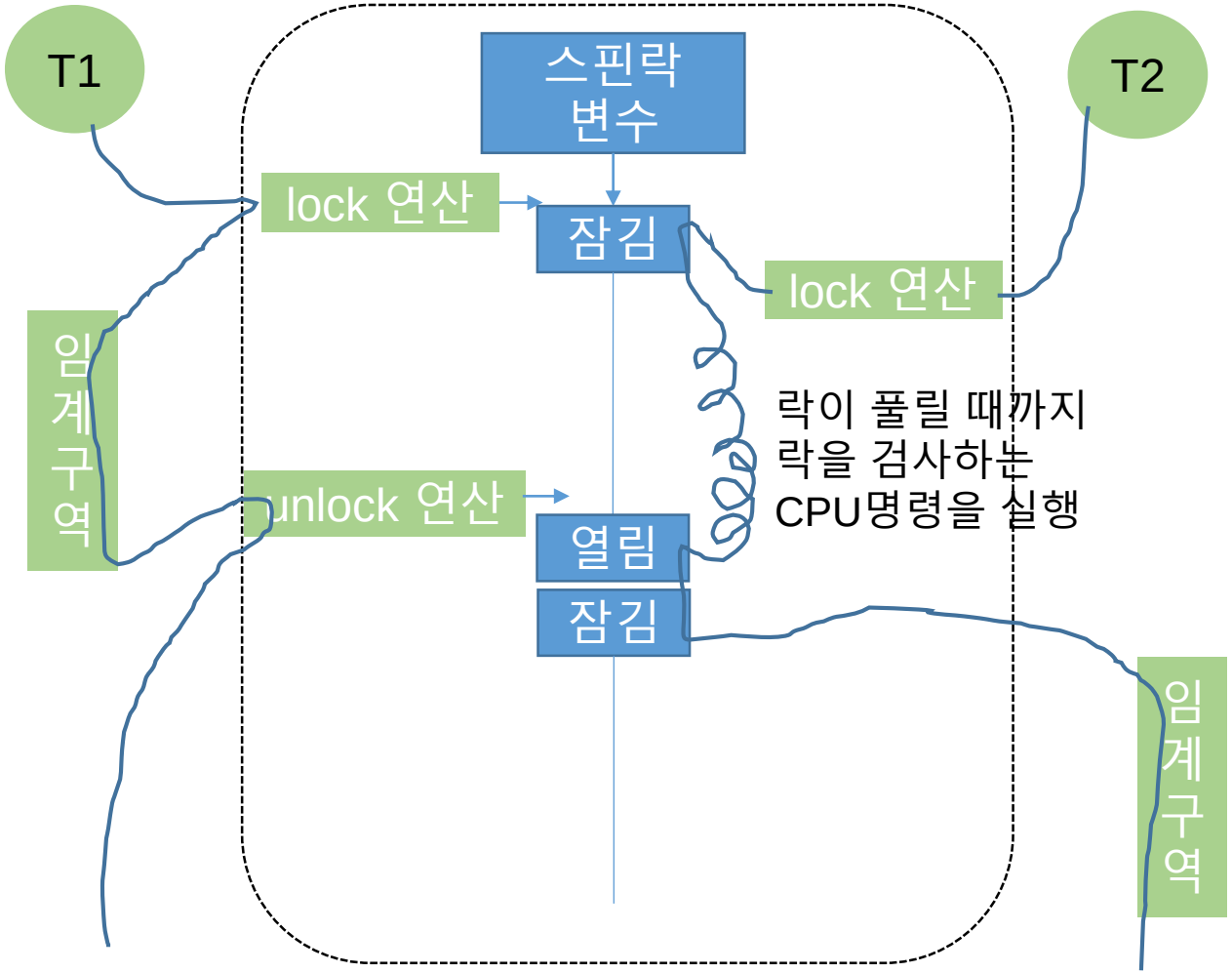


# 03. 멀티스레드 동기화 기법

- 스핀락

- 임계구역의 실행시간이 짧아 빨리 락이 풀리는 경우에 매우 효과적.

- 단일 CPU 운영체제에서는 비효율적인 방법.



# 03. 멀티스레드 동기화 기법

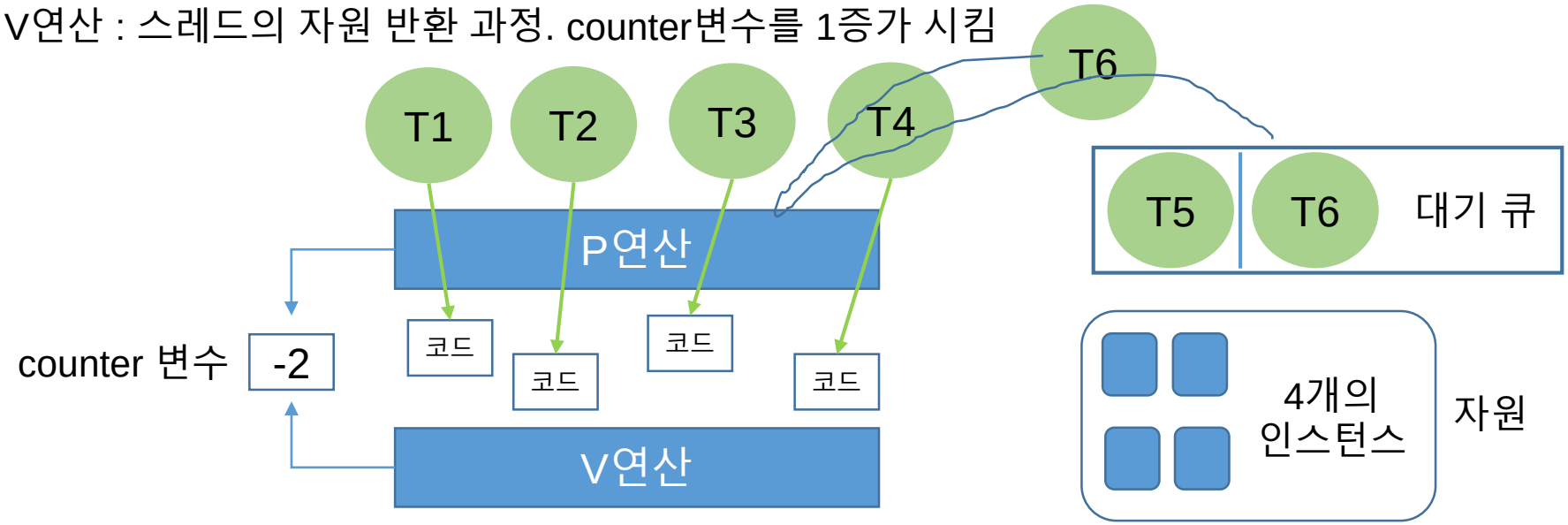
- 세마포 (Semaphore)

: 스레드가 동시에 사용할 수 있는 하나의 자원에 대해 스레드가 공유하도록 관리하는 기법

- 자원의 개수 < 스레드 개수 : 스레드 동기화 필요

- P연산 : 스레드의 자원 사용을 허가하는 과정. counter변수를 1감소 시킴

- V연산 : 스레드의 자원 반환 과정. counter변수를 1증가 시킴



# 03. 멀티스레드 동기화 기법

- 자원을 할당 받지 못할 경우의 동작 방식

- sleep-wait 세마포 : P연산에서 자원 사용을 허가 받지 못한 스레드는 대기 큐에서 잠을 잠

```
P연산 {
   counter--; //자원요청
   If (counter < 0) {
     현재 스레드를 대기큐에 삽입
   }
   자원 획득
}
```

```
V연산 {
   counter++; //자원반환
   If (counter <= 0) { //기다리는 스레드가 있으면
     대기 큐에서 한 스레드를 깨움
  }
   자원 획득
}
```

- busy-waiting 세마포 : P연산에서 사용가능한 자원이 생길 때까지 무한 루프를 돌며 체크

감사합니다☺