

AI 활용 프로그래밍

Week 13: 쓰레드 & 미니게임

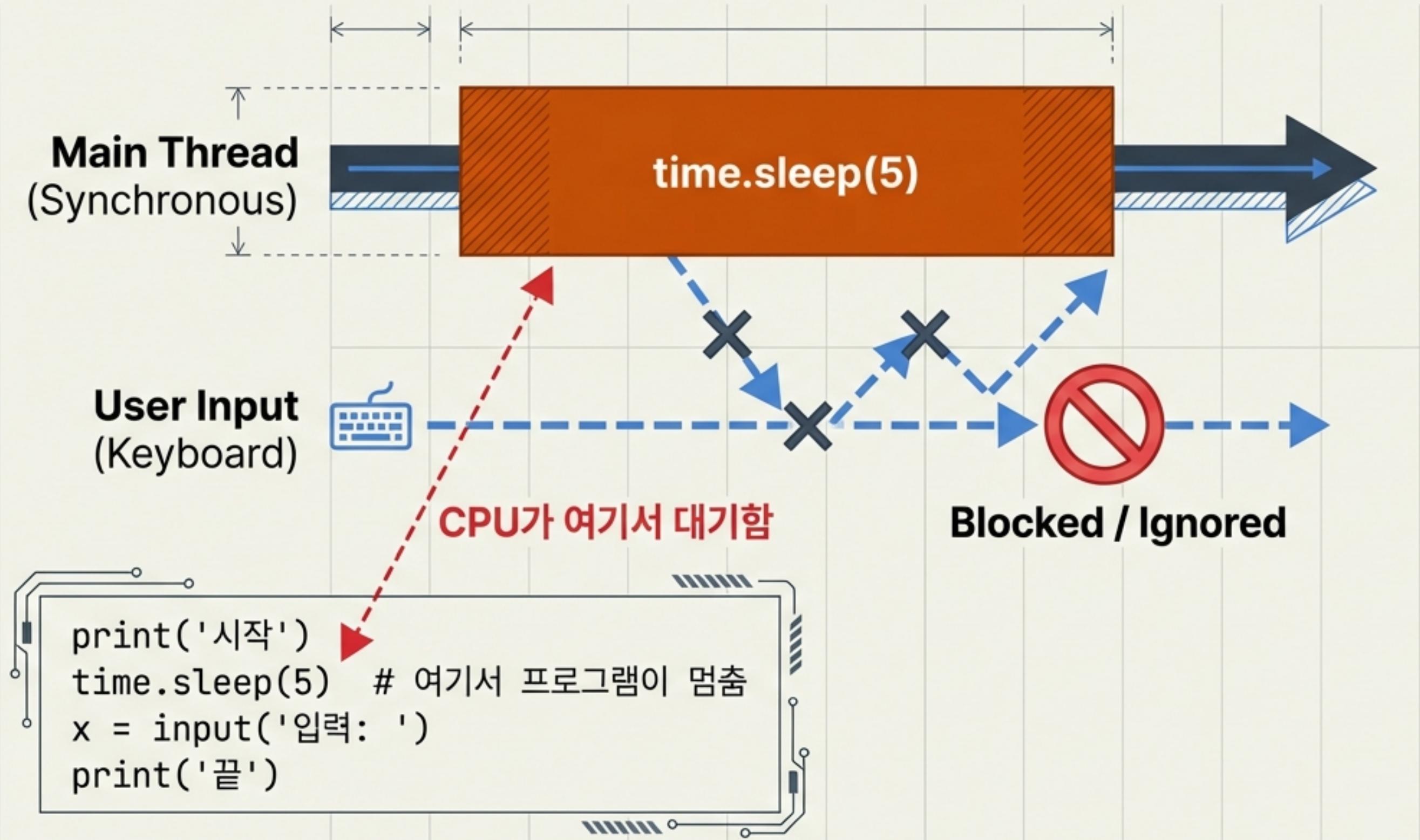
Blocking 로직에서 반응형 게임플레이로

2026-1학기 · 컴퓨터학부

Learning Objectives

1. 동시성(Concurrency)과 쓰레드(Thread)의 개념 이해
2. threading.Thread 생성, 실행(start), 동기화(join) 마스터
3. 미니게임에 타이머와 애니메이션 같은 '동시 실행' 요소 적용

워밍업: 타이머를 켜면 입력이 막힌다?



문제점

파이썬의 기본 실행 방식은 '동기 실행'입니다.
앞 줄이 끝날 때까지 다음 줄로 넘어가지 않습니다.

결과

5초 동안 프로그램은 '동결' 상태.
키보드 입력 불가.

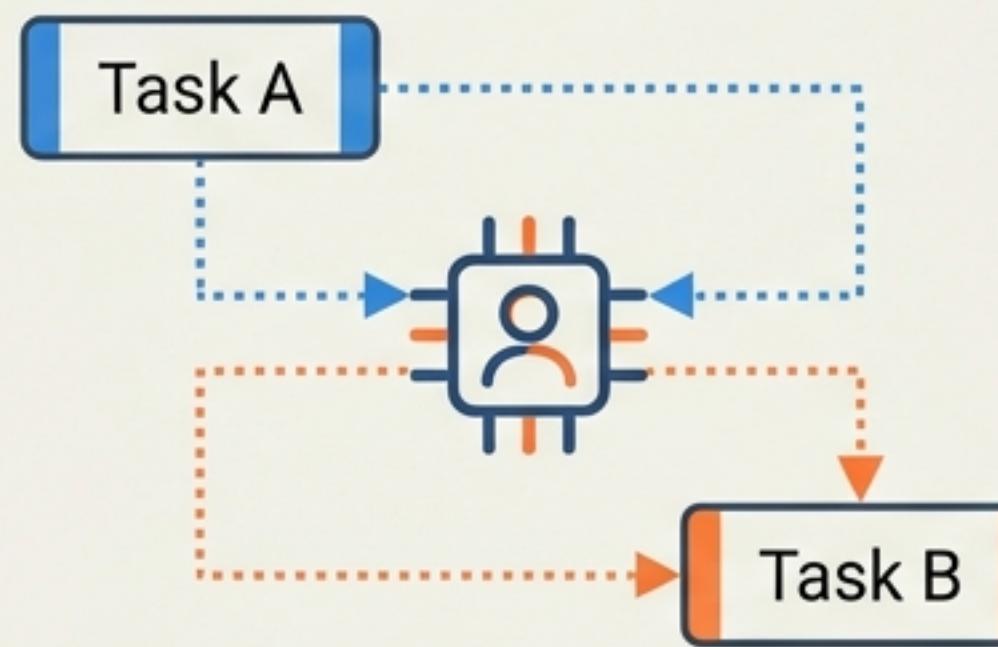
해결책

기다리는 동안 다른 일을 하려면
'쓰레드(Thread)'가 필요합니다.



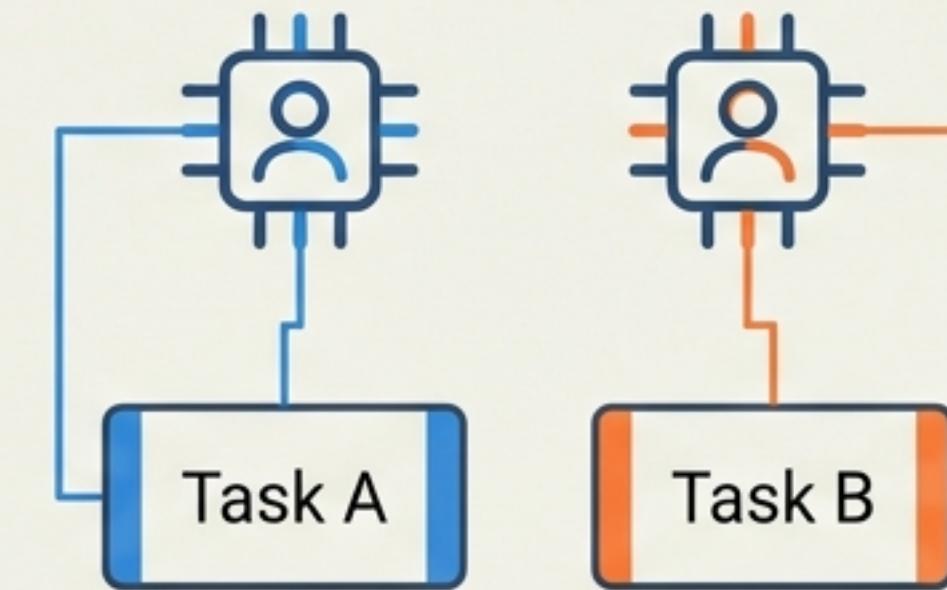
동시성(Concurrency) vs 병렬성(Parallelism)

동시성 (Single Core)



작업을 아주 빠르게 번갈아가며 처리
(Context Switching). 논리적 동시 실행.

병렬성 (Multi-Core)

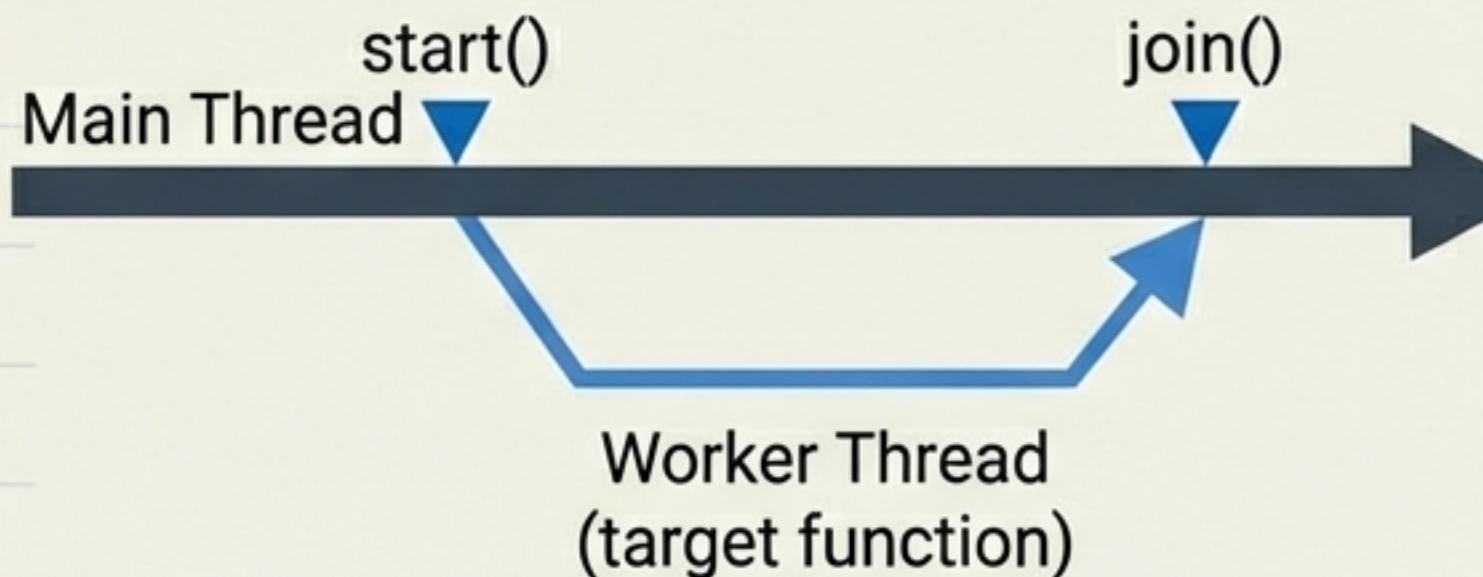


멀티 코어를 사용하여 물리적으로
동시에 처리.

파이썬의 Thread는 I/O 작업(파일 읽기, 네트워크 대기, sleep, 사용자 입력) 효율화에 강력합니다.
이번 주는 반응형 인터페이스를 위한 Threading에 집중합니다.

threading.Thread 기본 문법

Fork-Join Model



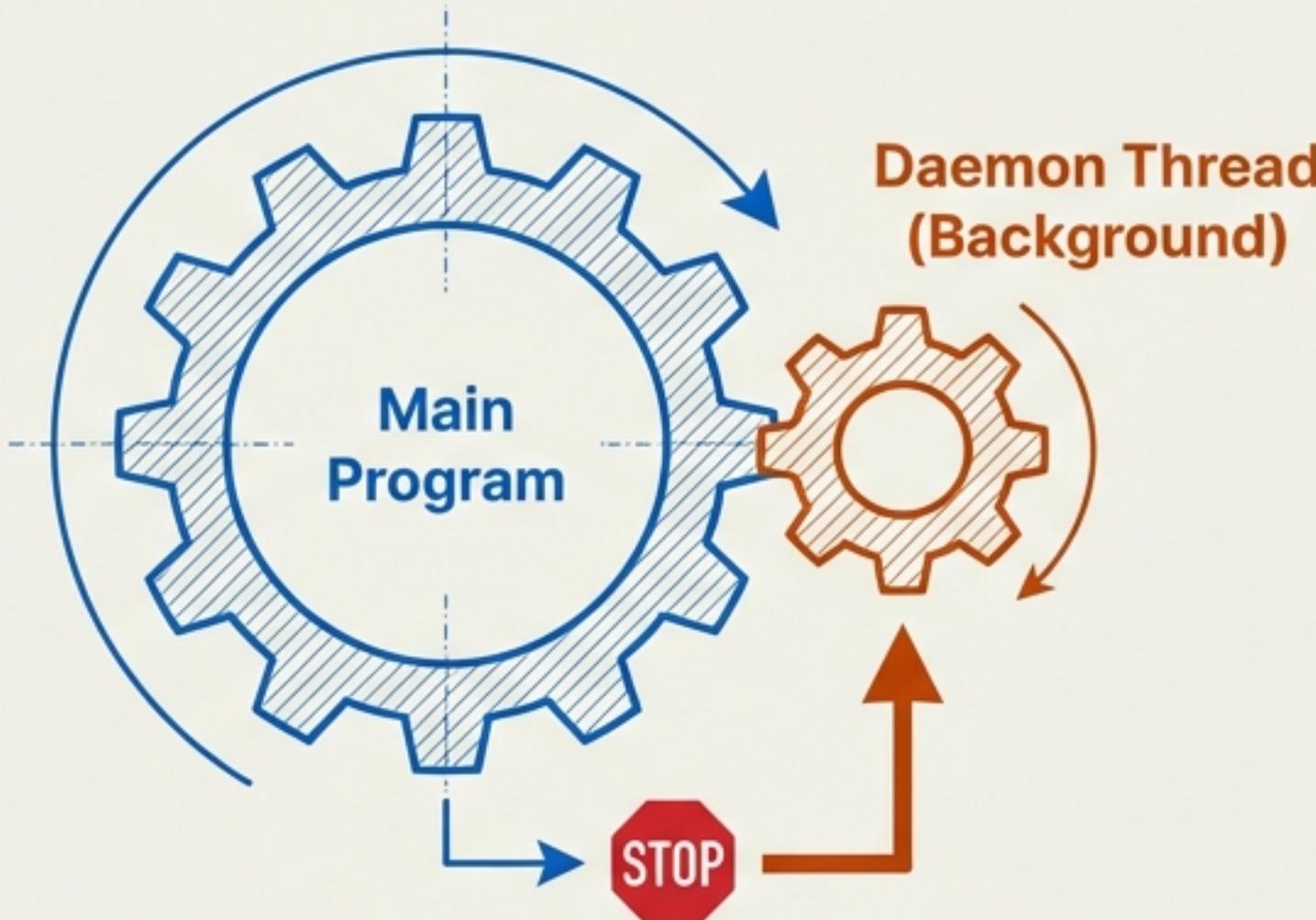
```
import threading  
  
def work():  
    print('작업 시작')  
  
t = threading.Thread(target=work) # 1. 객체 생성  
t.start() # 2. 실행 (분기)  
t.join() # 3. 대기 (합류)
```

핵심 용어 정의

- **target**: 별도의 흐름으로 실행할 함수 지정
- **start()**: 쓰레드 생성 및 즉시 실행
- **join()**: 작업이 끝날 때까지 메인 쓰레드가 대기

수명 주기 관리: Daemon 쓰레드

메인 프로그램 종료 시 함께 종료되는 보조 쓰레드



메인 프로그램이 멈추면, 데몬 쓰레드도 즉시 멈춤.

개념

메인 프로그램이 종료될 때, 함께 강제 종료되어야 하는 보조 작업.

코드

```
t = threading.Thread(target=spinner, daemon=True)
```

활용 사례

- 백그라운드 음악 재생
- 자동 저장
- 로딩 애니메이션



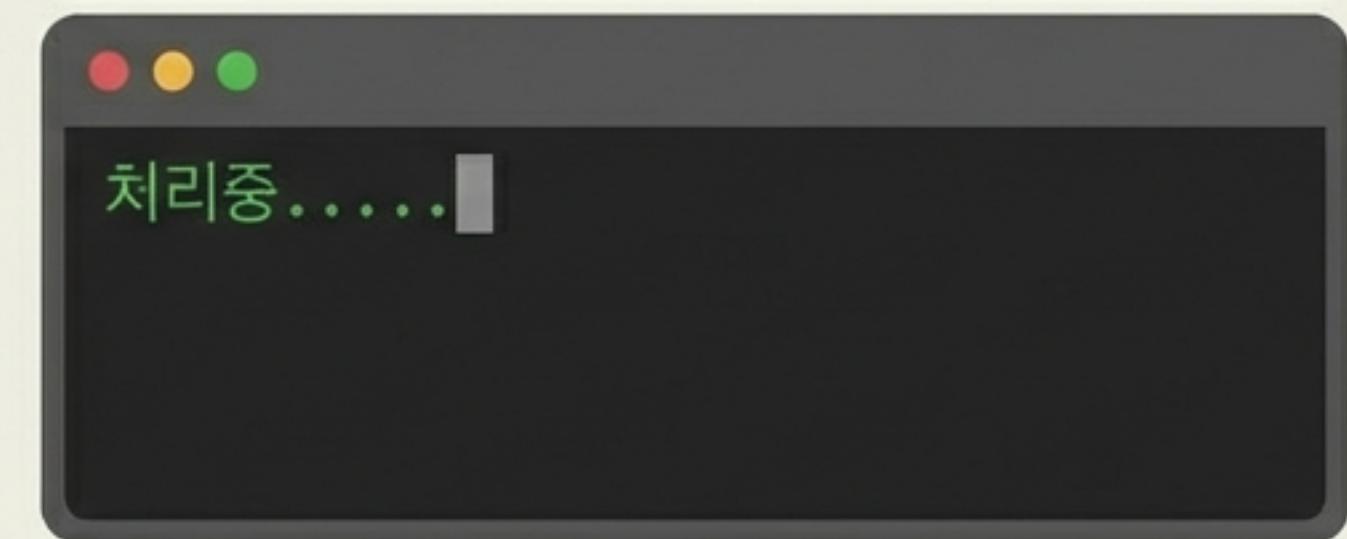
주의: 데이터 저장 등 중요한 작업 도중에 강제 종료될 수 있음.

실전 예제: 멈추지 않는 로딩 스피너

```
def spinner():
    while running:
        print('.', end='', flush=True)
        time.sleep(0.2)

running = True ..... # 데몬 쓰레드로 설정
t = threading.Thread(target=spinner, daemon=True)
t.start()

do_heavy_work() # 메인 작업 수행
running = False # 스피너 중지 .....
```



Remote Control Bridge:
"running" variable controls execution

Spinner Thread (Printing dots)

Main Thread (Heavy Calculation)

Remote Control Bridge:
"running" variable controls execution

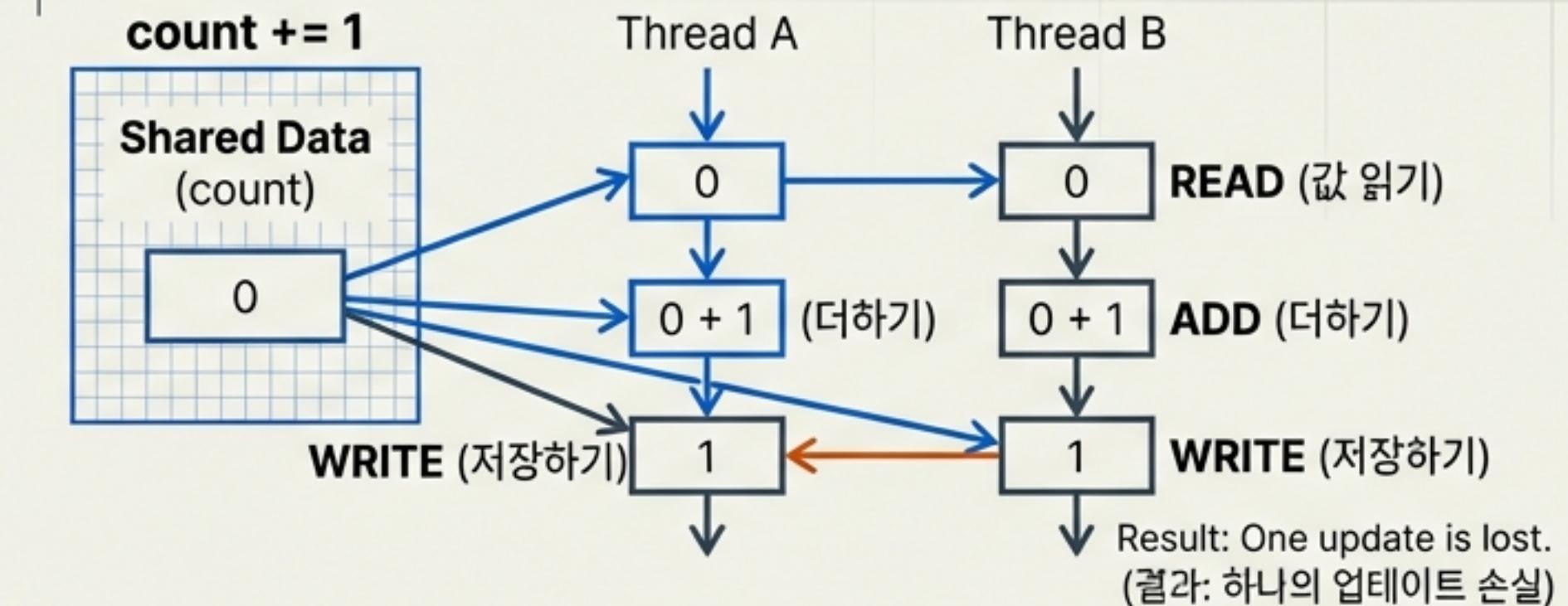
함정: 경쟁 상태 (Race Condition)

The Mystery

두 개의 쓰레드가 'count'를 100,000번씩 증가시킴.

- 예상 결과: 200,000
- 실제 결과: 145,921

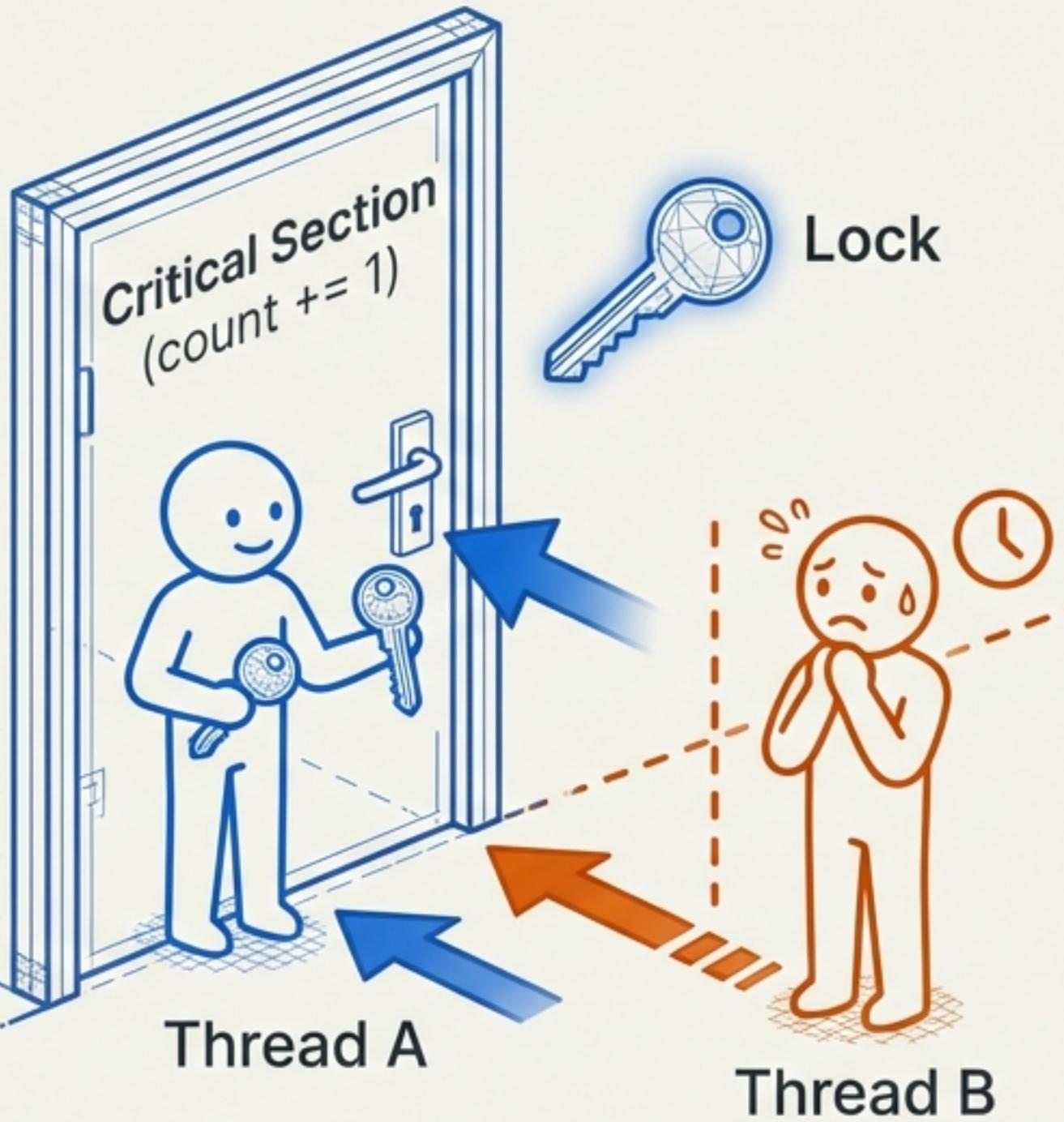
Micro-Operation Diagram



교훈

공유 데이터(Shared Data)에 대한 보호 없는 동시 접근은 데이터 손실을 유발합니다 (Lost Update).

보호막: Lock (Mutual Exclusion)



```
lock = threading.Lock()

def inc():
    global count
    for _ in range(100000):
        with lock: # 1. 잠금 획득 (Acquire)
            count += 1
        # 2. 임계 구역 (CriticalSection)
        # 3. 잠금 해제 (Release)
```

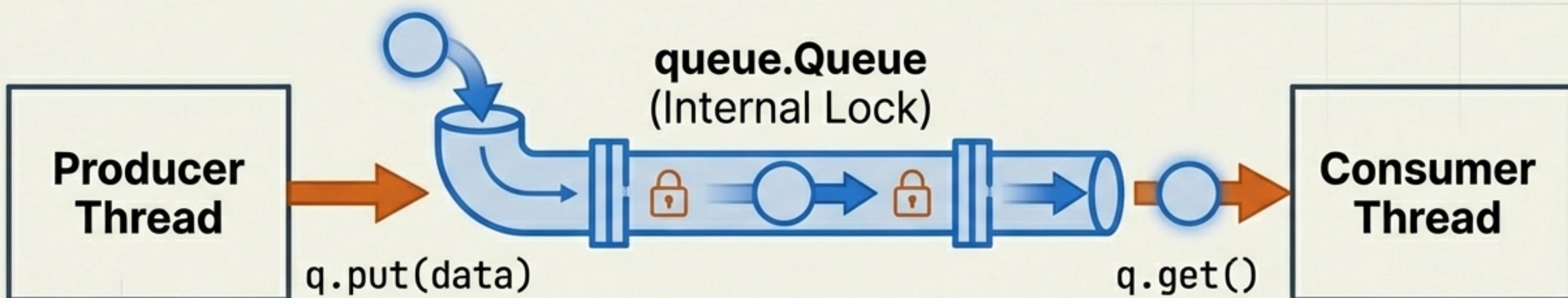
- 장점: 데이터 무결성 보장.
- 단점: 과도한 잠금은 성능 저하 (병렬성 감소).





안전한 통로: Queue (Thread-Safe Communication)

“메모리를 공유하여 통신하지 말고, 통신하여 메모리를 공유하라.”



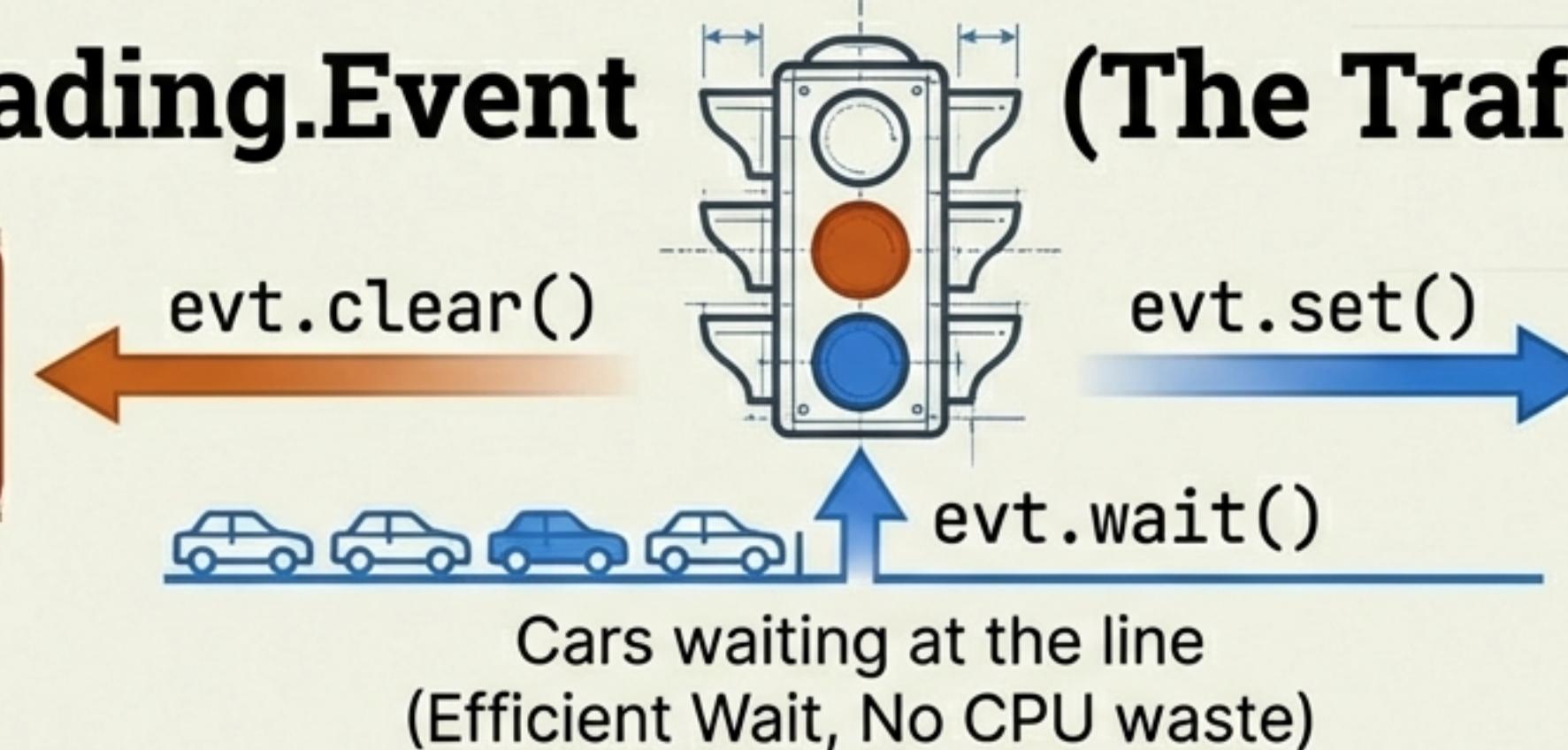
- `queue.Queue`는 내부적으로 Lock이 구현되어 있어 **Thread-safe**.
- 데이터 흐름이 명확하여 **디버깅이 용이**함.



고급 제어 도구: Event & Timer

threading.Event

(The Traffic Light)



threading.Timer (The Time Bomb)

```
t = threading.Timer(1.0, callback)
```

지정된 시간 후, 함수를 단 한 번 실행.

게임 제한 시간 초과 판정.





미니게임 설계: 반응 속도 테스트

- 1. 랜덤한 시간(2~5초) 대기.
- 2. 화면에 'GO!' 출력.
- 3. 사용자가 Enter를 누를 때까지의 시간(ms) 측정.



Challenge Note
Main Thread는 입력을 대기하고, Background Thread는 시간을 잡니다.



게임 스켈레톤 코드 (Skeleton Code)

```
import threading, time, random
go_event = threading.Event() # 신호 객체

def game_logic():
    time.sleep(random.randint(2, 5))
    print("GO!")
    go_event.set() # 신호 발송!

# 쓰레드 시작
threading.Thread(target=game_logic, daemon=True).start()

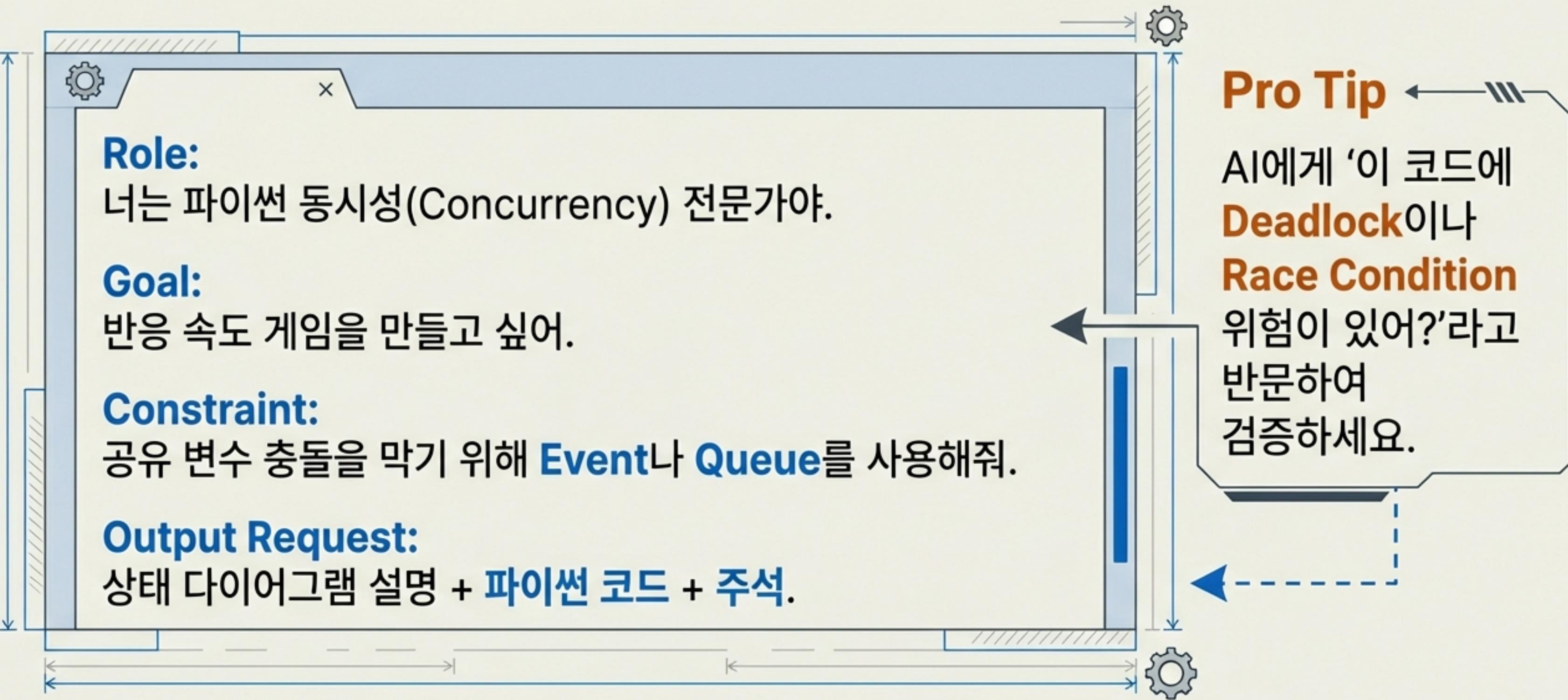
input("준비되면 엔터...")
# 메인 쓰레드는 여기서 대기하다가 신호를 받음
```

신호 발송!

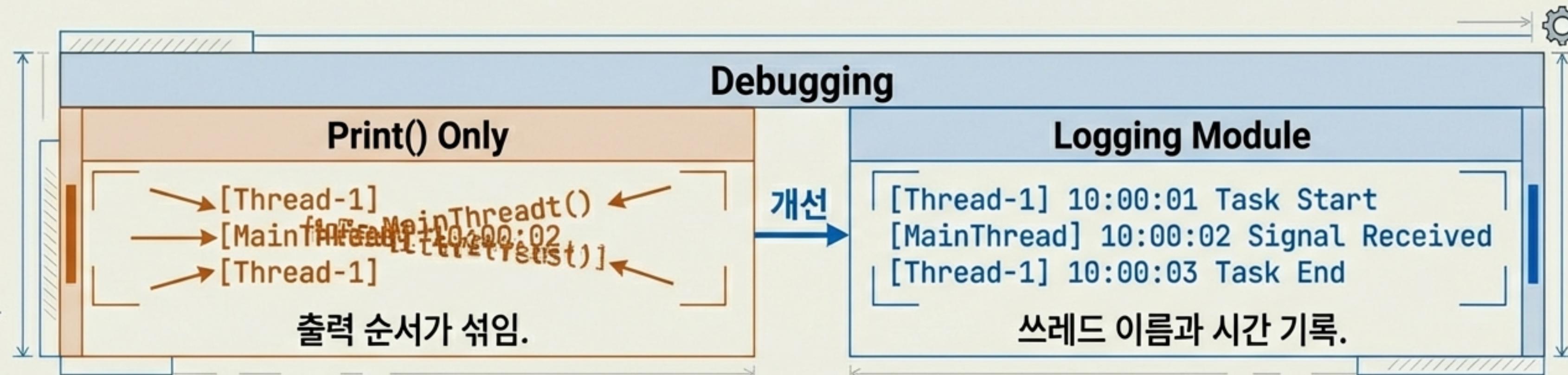
Key Concept
Logic(타이밍)과
UI(입력)의 분리.



With AI: 쓰레드 코딩을 위한 프롬프트 전략



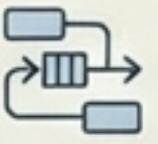
디버깅과 안전한 설계 팁



Best Practices List



공유 변수 최소화: 가능하면 지역 변수 사용.



구조 단순화: 복잡한 Lock보다 Queue 사용 권장.



데드락(Deadlock) 주의: 서로가 끝나기를 영원히 기다리는 상황 방지.

요약 및 미니 퀴즈

Thread

start/join으로
실행 및 동기화

Safety

Lock으로 보호,
Queue로 통신

Control

Event/Timer로
흐름 제어

미니 퀴즈

Q1. start() vs run()의 차이는?

새 쓰레드 생성 여부

Q2. 공유 자원을 보호하지 않을 때 발생하는 문제는?

Race Condition

Q3. Thread-safe한 통신 도구는?

Queue

다음 주: 나만의 창의적인 프로그램 발표.