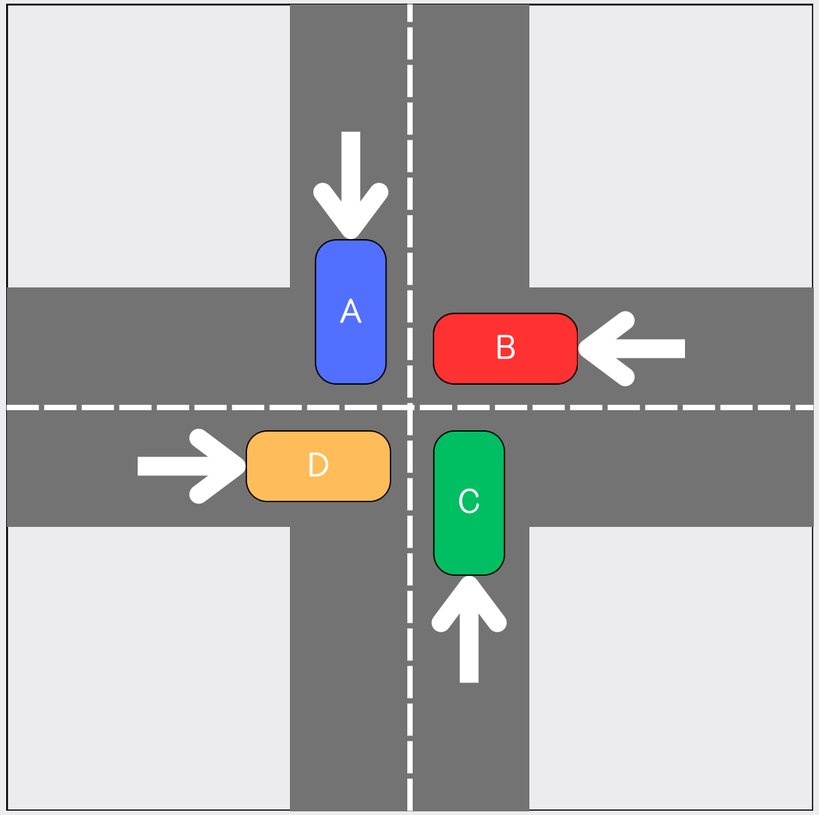
교착상태

Deadlock?



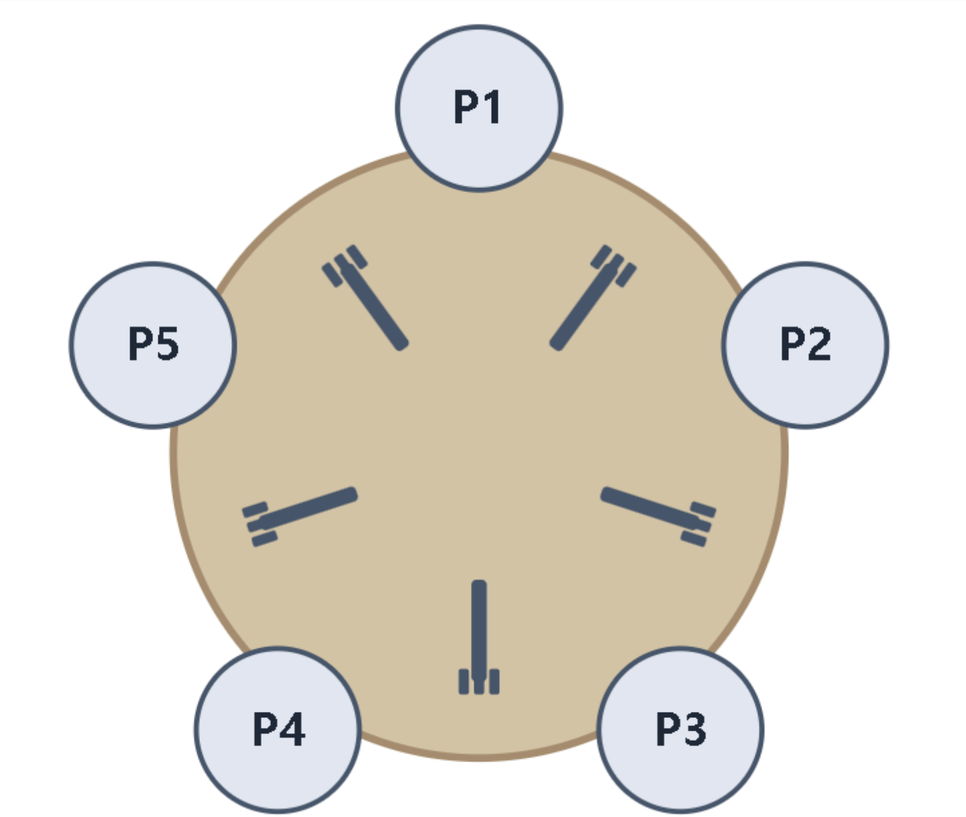
두 개 이상의 프로세스가 서로의 **자원**을 기다리며
무한히 대기하는 상태입니다

발생조건

- 상호배제 Mutual Exclusion
- 점유대기 Hold and Wait
- 비선점 No Preemption
- 순환대기 Circular Wait

모든 조건이
동시에 성립해야
데드락 발생

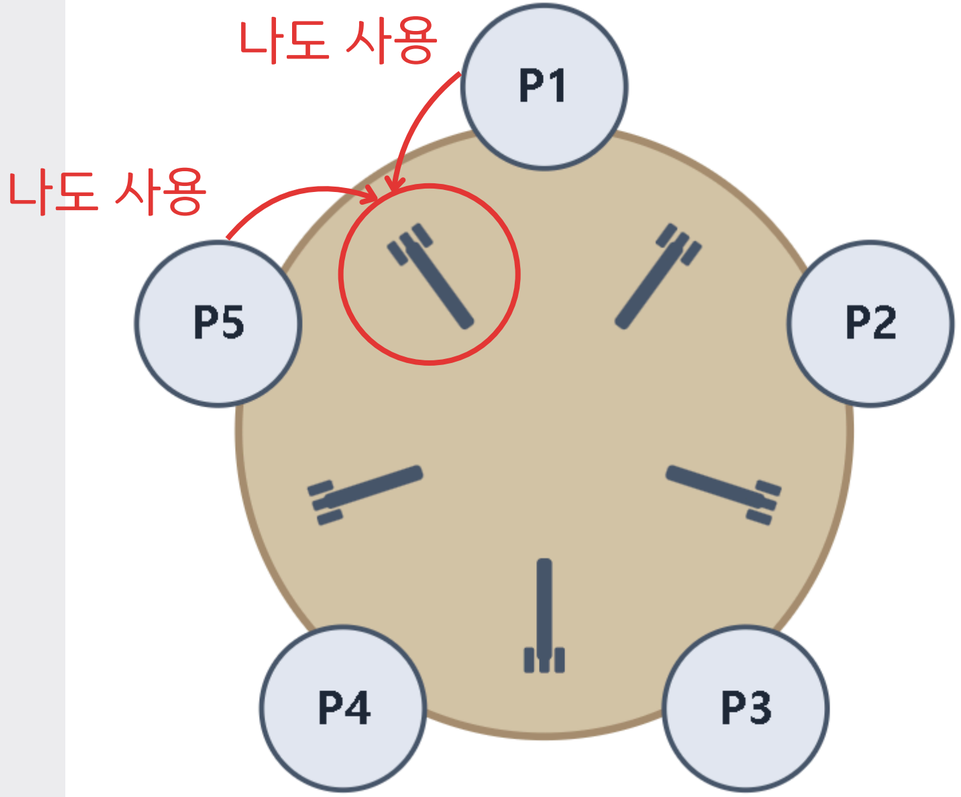
식사하는 철학자 문제




```
while (true) {
    think();                    // 생각한다

    pickLeft();                 // 왼쪽 포크를 집는다
    pickRight();                // 오른쪽 포크를 집는다

    eat();                      // 식사한다

    putRight();                 // 오른쪽 포크를 내려놓는다
    putLeft();                  // 왼쪽 포크를 내려놓는다
}
```

발생조건 - 상호배제

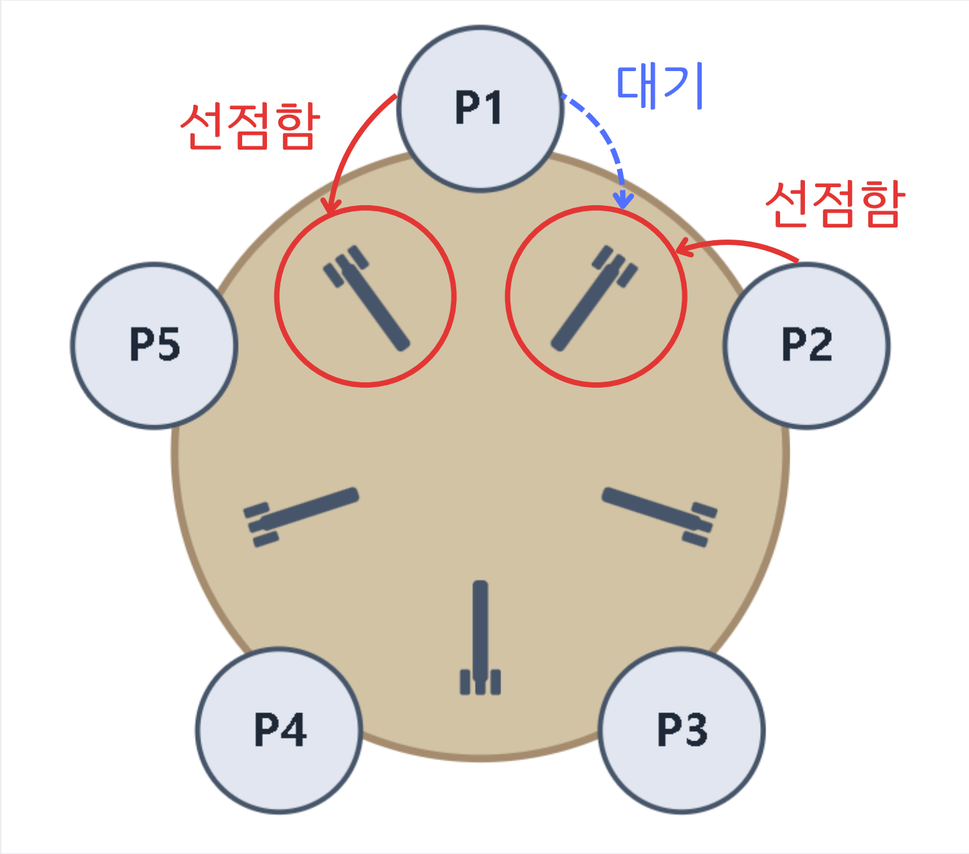
자원을 한 번에 하나의 프로세스/스레드만 사용 가능



포크 하나를 두 철학자가 동시에 사용할 수 없음

발생조건 - 점유대기

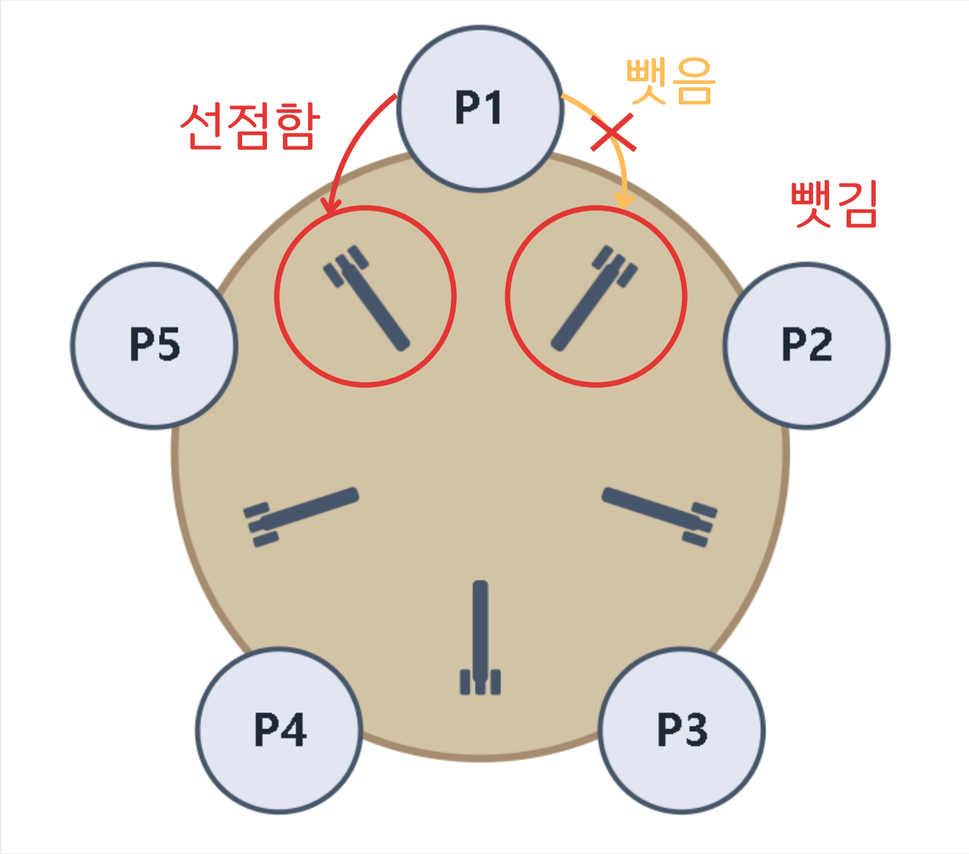
자원을 점유한 상태에서 다른 자원을 추가로 요청하며 대기



왼쪽 포크를 잡은 상태에서 오른쪽 포크를 요청하며 대기

발생조건 - 비선점

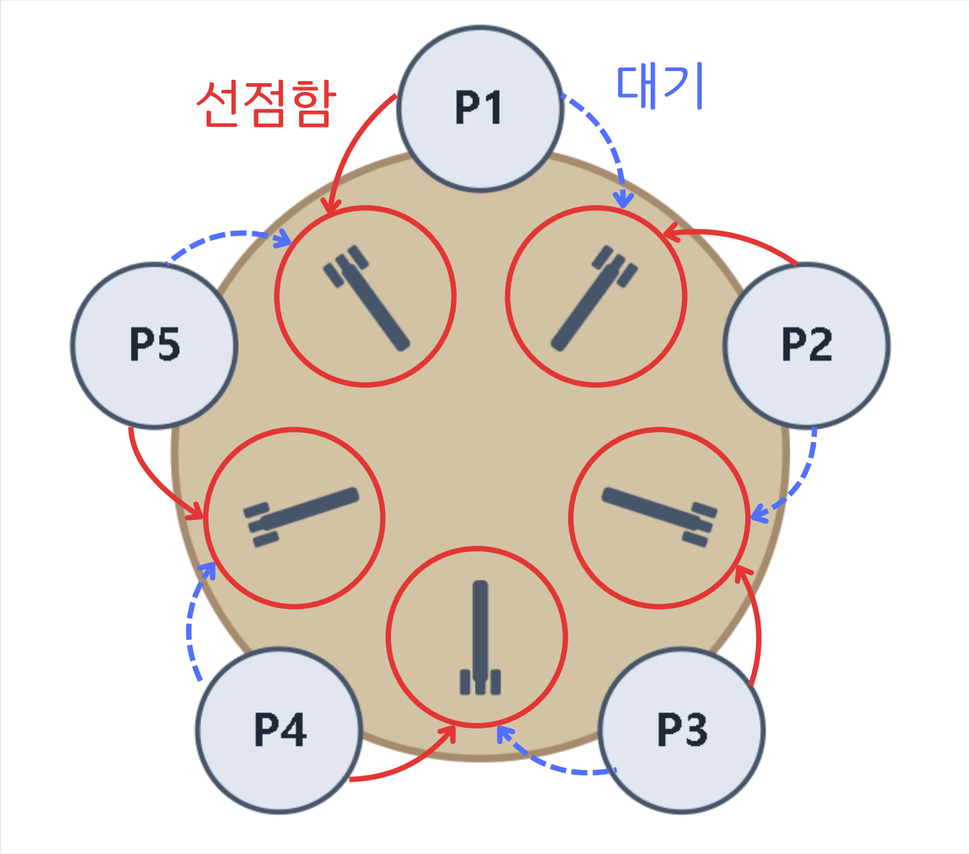
다른 프로세스의 자원을 강제로 빼앗을 수 없음



옆 철학자의 포크를 강제로 빼앗을 수 없음

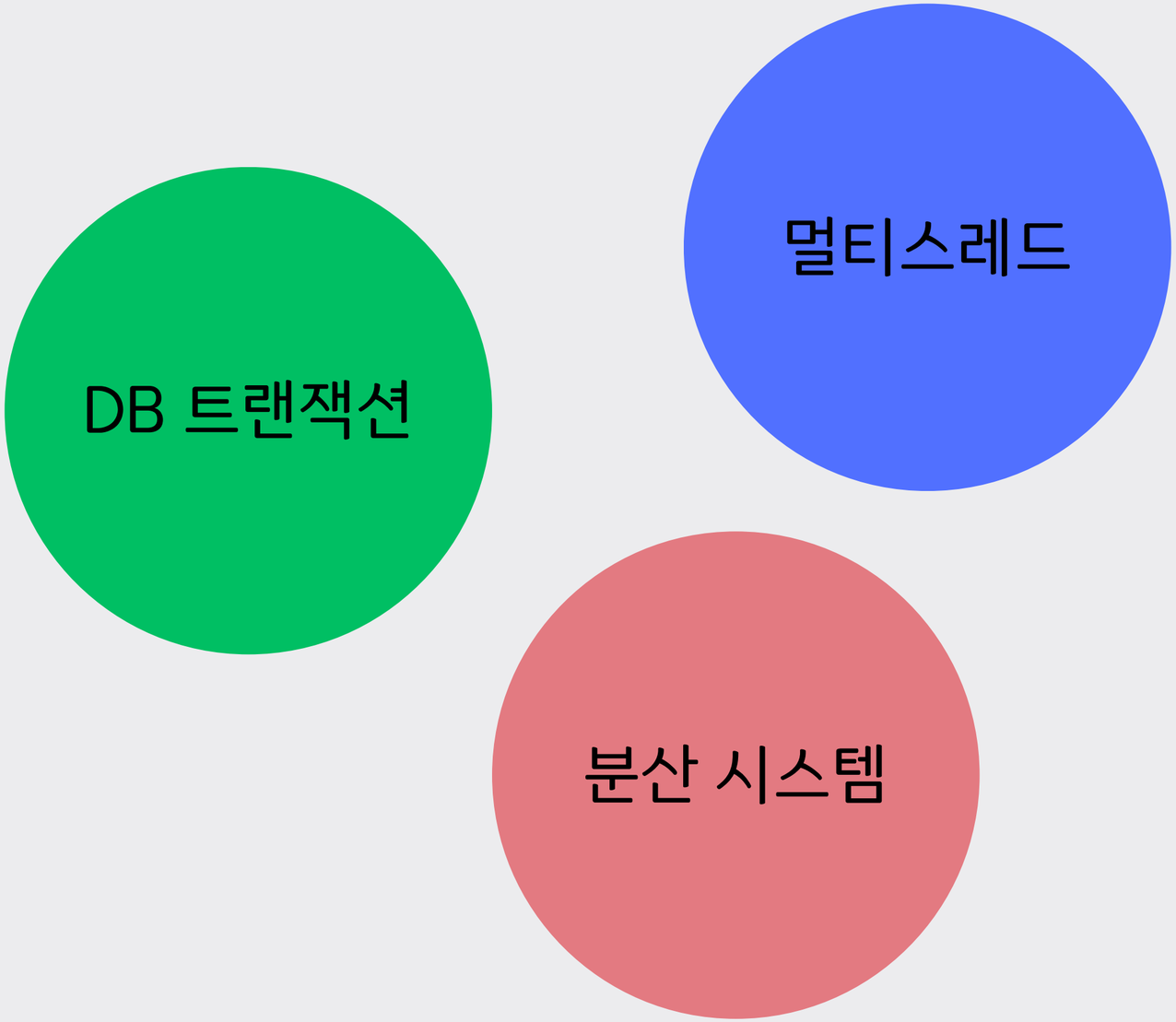
발생조건 - 순환대기

프로세스들이 순환 형태로 서로의 자원을 대기



P1→P2→P3→P4→P5→P1 원형 대기 고리 형성

실제 시스템에서 문제 발생



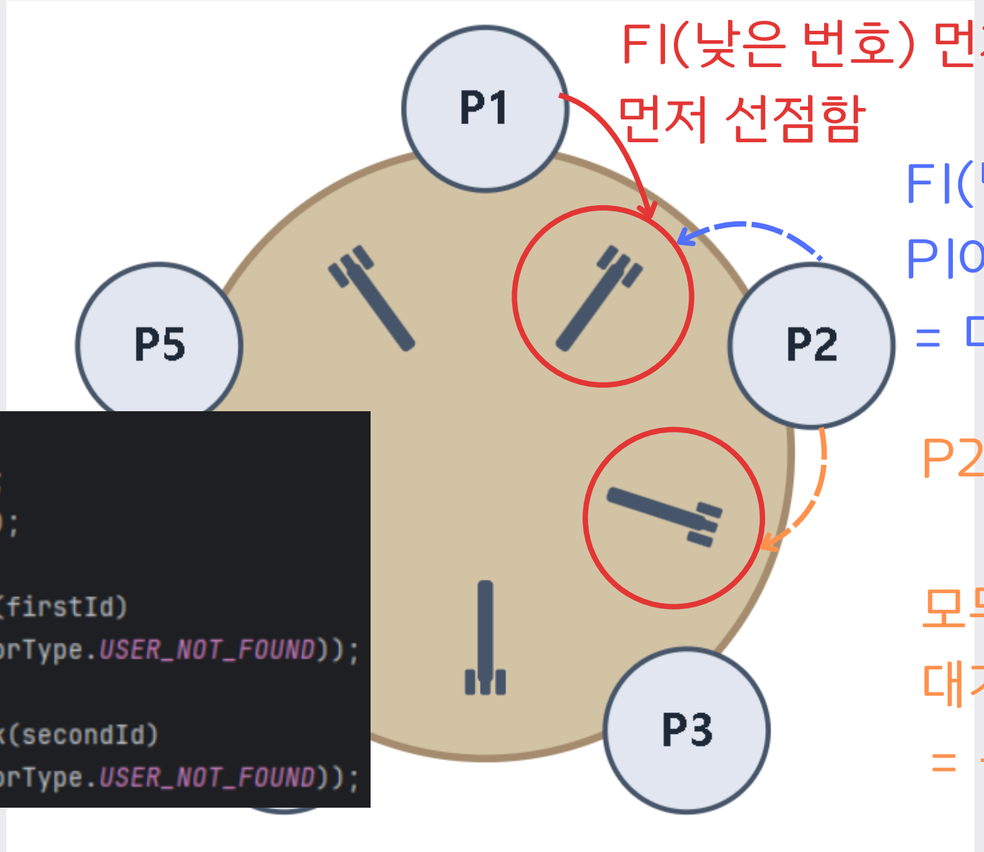
해결법

- 예방 Prevention
- 회피 Avoidance
- 탐지 Detection
- 무시 Ignorance

해결법 - 예방

4가지 조건 중 하나 이상을 원천적으로 제거

```
// User Id가 낮은 순으로 락 잠금
Long firstId = Math.min(followerId, followingId);
Long secondId = Math.max(followerId, followingId);

User firstUser = userRepository.findByIdWithLock(firstId)
        .orElseThrow(() -> new CoreException(ErrorType.USER_NOT_FOUND));

User secondUser = userRepository.findByIdWithLock(secondId)
        .orElseThrow(() -> new CoreException(ErrorType.USER_NOT_FOUND));
```



F1(낮은 번호) 먼저 시도
먼저 선점함

F1(낮은 번호) 먼저 시도
P1이 이미 보유
= 대기

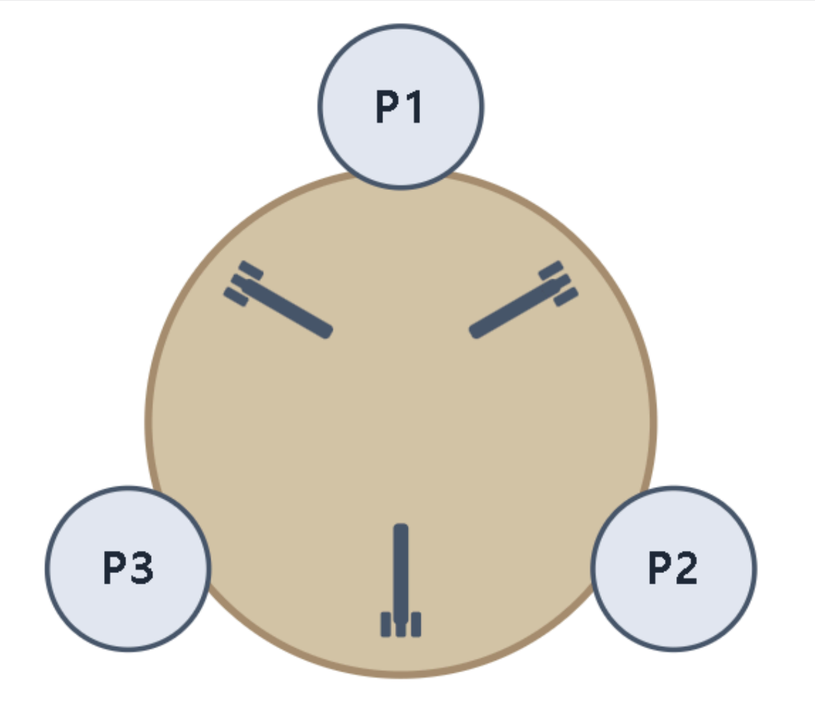
P2는 F1을 못 잡았으니 F2도 안잡음

모두 낮은 번호부터 집으면
대기 방향이 한쪽으로만 흐름
= 순환대기 조건이 깨짐

모든 철학자가 포크 번호가 낮은 것부터 집는 방식

해결법 - 회피

자원 할당 전 안전 상태인지 확인



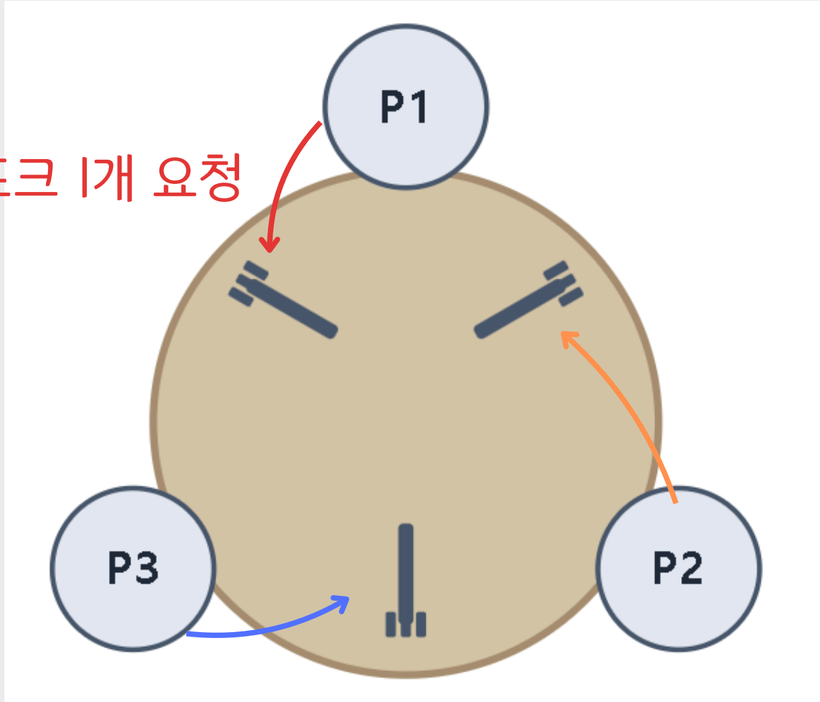
Available	현재 사용 가능한 자원 수
Max	각 프로세스가 최대로 필요한 자원 수
Allocation	각 프로세스가 현재 보유한 자원 수
Need	각 프로세스가 추가로 필요한 자원 수 (Max - Allocation)

포크 집기 전에 ”둘 다 가능한가?” 확인

해결법 - 회피

자원 할당 전 안전 상태인지 확인

P1이 포크 1개 요청

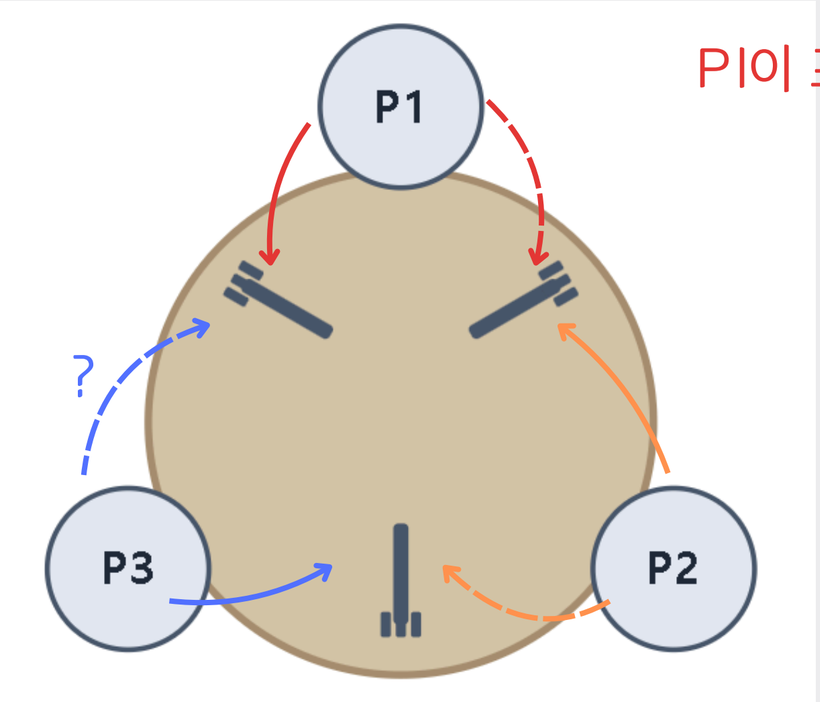


철학자	Max	Allocation	Need
P1	2	1	1
P2	2	1	1
P3	2	1	1

Available = 3 - 3 = 0개

해결법 - 회피

자원 할당 전 안전 상태인지 확인



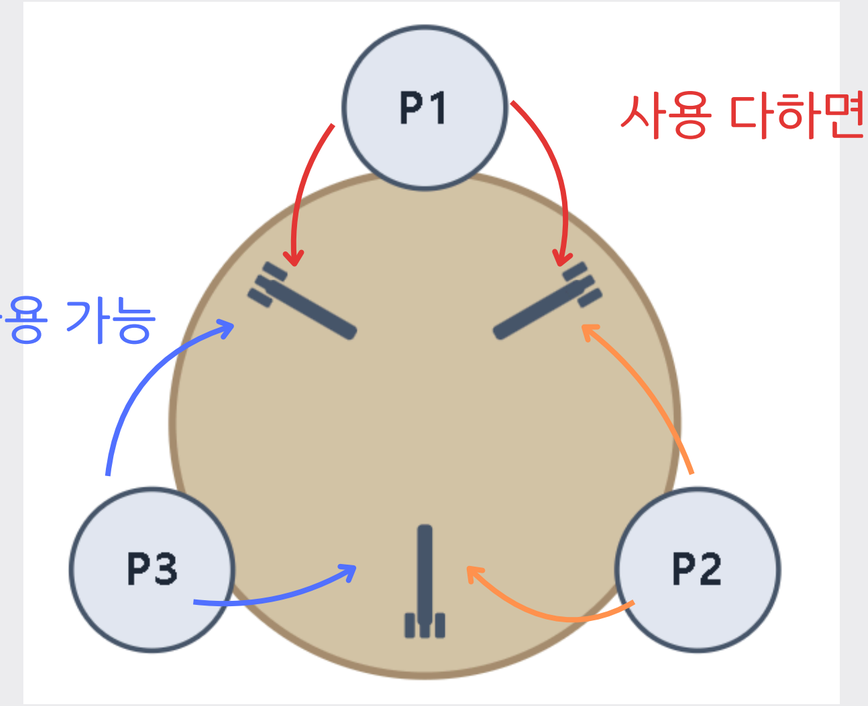
P1이 포크 1개 더 요청

철학자	Max	Allocation	Need
P1	2	1	1
P2	2	1	1
P3	2	1	1

Available = 3 - 6 = -3개??

해결법 - 회피

자원 할당 전 안전 상태인지 확인



사용 다하면 반납

반납 했으니 사용 가능

철학자	Max	Allocation	Need
P1	2	2	0
P2	2	1	1
P3	2	0	2

↓

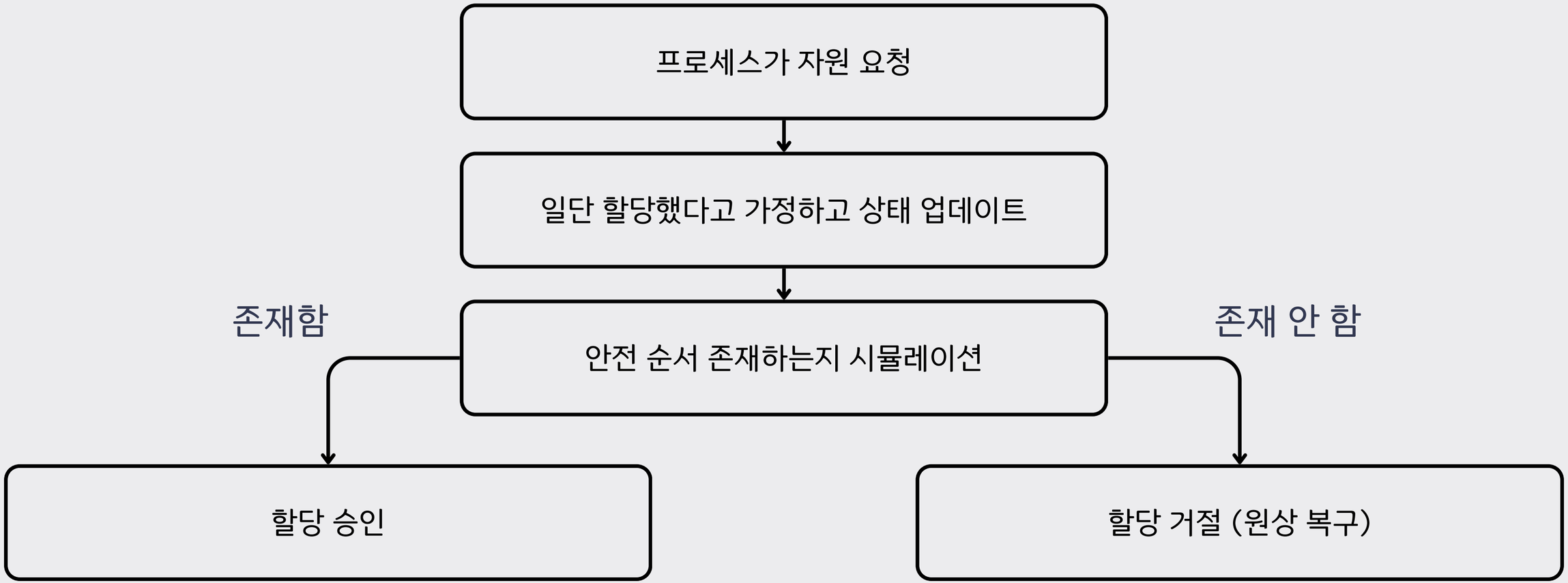
철학자	Max	Allocation	Need
P1	2	0	2
P2	2	2	0
P3	2	1	1

↓

철학자	Max	Allocation	Need
P1	2	0	2
P2	2	0	2
P3	2	2	0

해결법 - 회피

자원 할당 전 안전 상태인지 확인



해결법 - 회피

자원 할당 전 안전 상태인지 확인

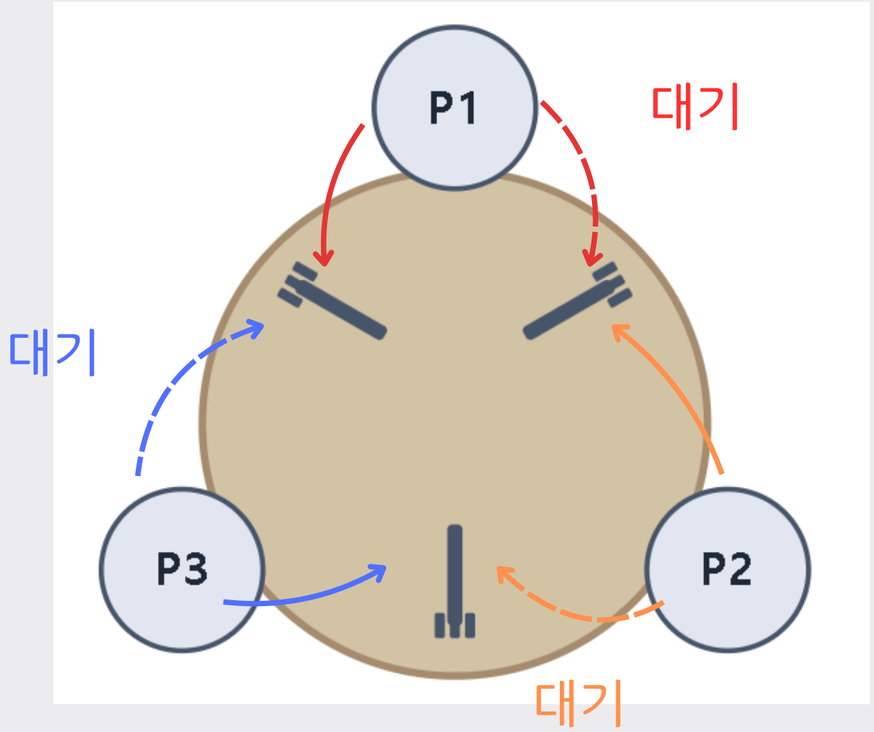
실제 실무에는 불가능 하지 않을까?

–

시스템의 모든 프로세스와 자원의 최대 요구량을 미리 알고 있어야하는 등
비현실적인 제약 조건과 복잡성이 존재하여 사용하지 못할거 같음

해결법 - 탐지

발생을 허용하되 주기적으로 탐지 후 복구



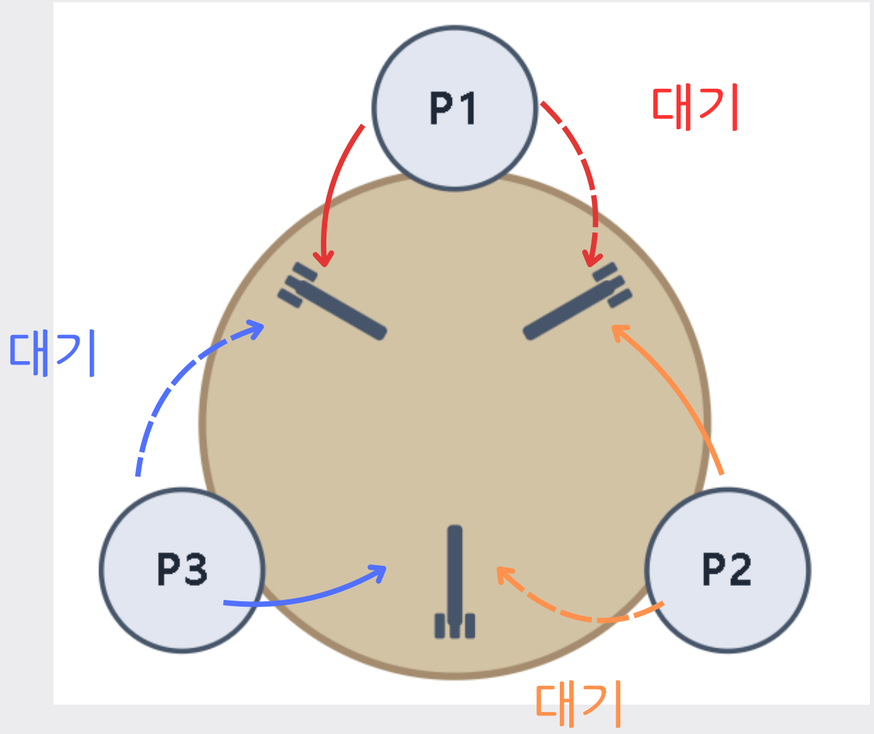
일단 발생하도록 **허용**

발견하면 **복구**

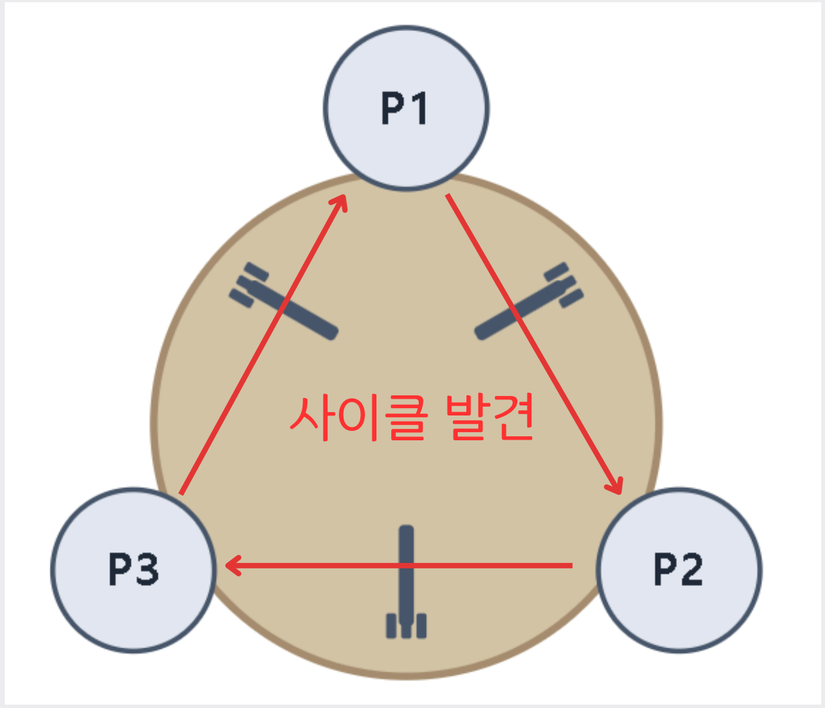
일정 시간 대기 후 포크 내려놓고 재시도

해결법 - 탐지

발생을 허용하되 주기적으로 탐지 후 복구

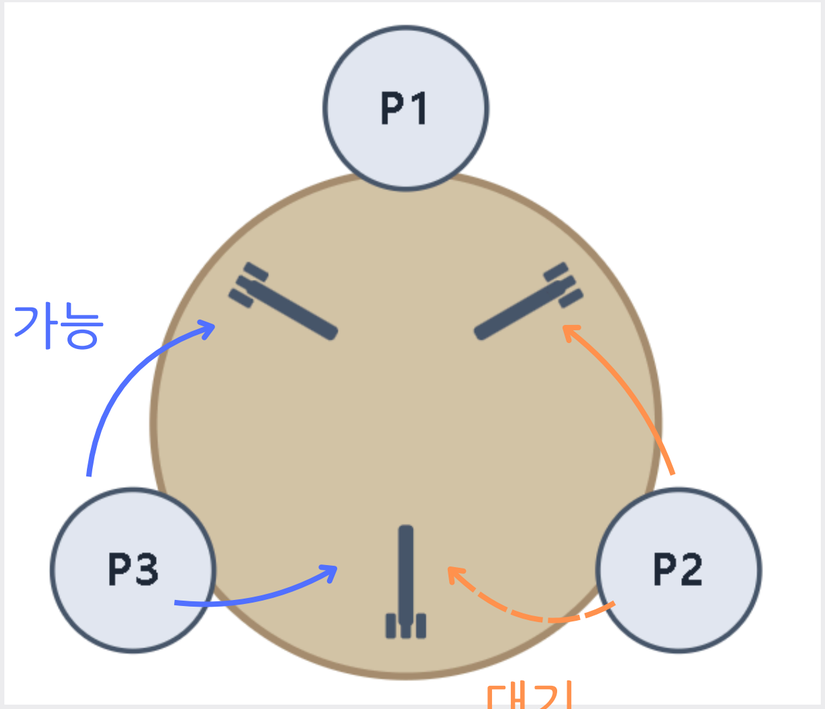


Wait-for 그래프



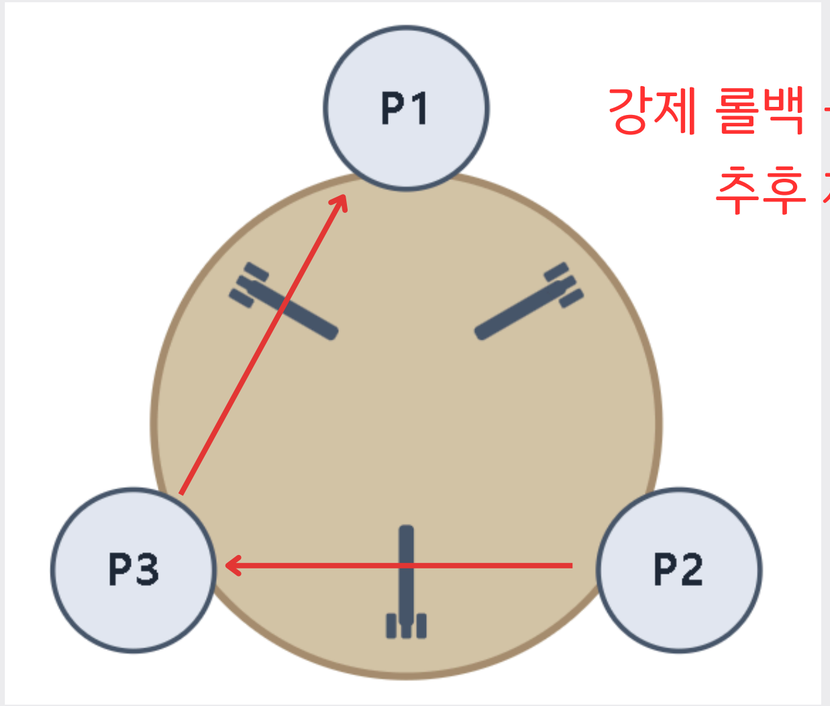
해결법 - 탐지

발생을 허용하되 주기적으로 탐지 후 복구



희생자 선택

(비용 낮은 것)



해결법 - 무시

발생 확률이 낮으면 무시

실제 서비스에는 불가능 하지 않을까?

–

24시간 운영 서비스에서 데드락 발생 시
사용자 불만/이탈 => 매출 손실

데드락? 그냥 무시하고, 발생하면 재부팅하면 되지

그럼 뭐 쓰라고

- 락 순서 고정
- tryLock + 타임 아웃
- DB는 InnoDB 자동 탐지
- 락 범위 잘 잡기

QnA