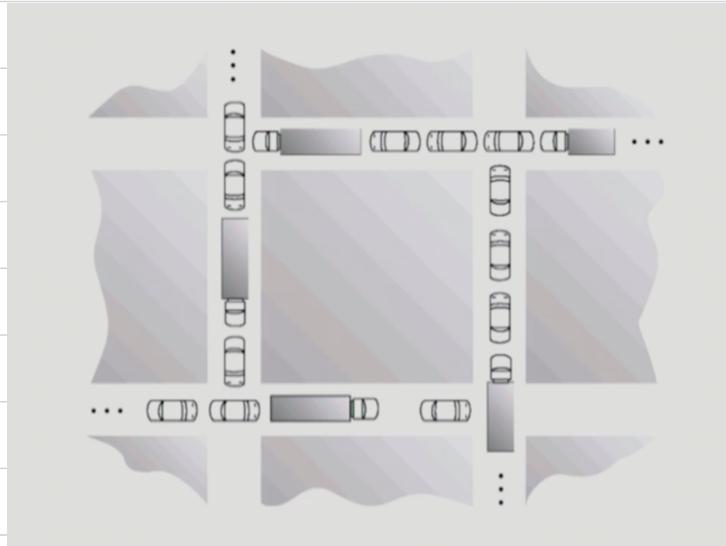


# Deadlock (교착 상태)



## The Deadlock Problem

### - Deadlock

- 일련의 프로세스들이 서로가 가진 자원을 기다리며 block된 상태

### - Resource

- 하드웨어, 소프트웨어 등을 포함하는 개념
- (예) I/O device, CPU cycle, memory space, Semaphore 등
- 프로세스가 자원을 사용하는 절차  
*\*Request, Allocate, Use, Release*

### - Deadlock Example 1

- 시스템에 2개의 tape drive가 있다.
- 프로세스 P<sub>1</sub>과 P<sub>2</sub> 각각이 하나의 tape drive를 보유한 채 다른 하나를 기다리고 있다.

### - Deadlock Example 2.

- Binary semaphores A and B

P <sub>1</sub> :	P <sub>2</sub> :
P(A);	P(B);
P(B);	P(A);

# Deadlock 발생의 4가지 조건

## • Mutual Exclusion (상호 배제)

✓ 매 순간 하나의 프로세스 만의 자원을 사용 할 수 있음

## • No preemption (비선점)

✓ 프로세스는 자원을 스스로 빼어놓을 뿐 강제로 빼앗기지 않음

## • Hold and Wait (보유 대기)

✓ 자원을 가진 프로세스가 다른 자원을 기다릴 때 보유 차운을 놓기 않고 계속 가지고 있음.

## • Circular wait (순환 대기)

✓ 자원을 대리는 프로세스간에 사이클이 형성 되어야 함.

✓ 프로세스  $P_0, P_1, \dots, P_n$ 이 있을 때

$P_0$ 은  $P_1$ 이 가진 자원을 대리함

$P_1$ 은  $P_2$ 가 가진 자원을 대리함

$P_2$ 은  $P_3$ 이 가진 자원을 대리함

$P_3$ 은  $P_0$ 이 가진 자원을 대리함

# Resource-Allocation Graph (자원 할당 그래프)

## - Vertex

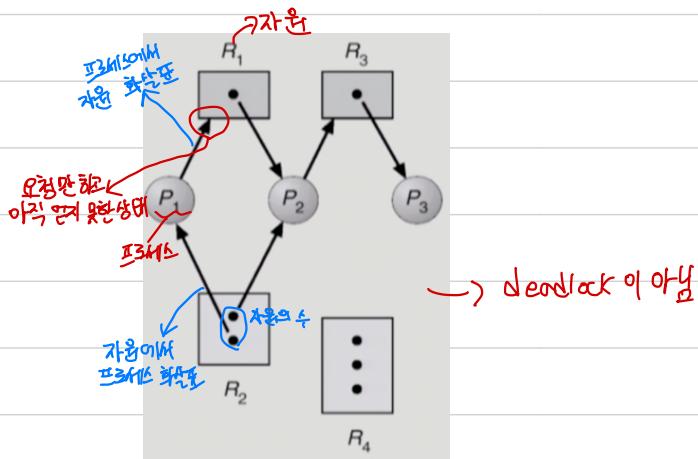
✓ Process  $P = \{P_1, P_2, \dots, P_n\}$

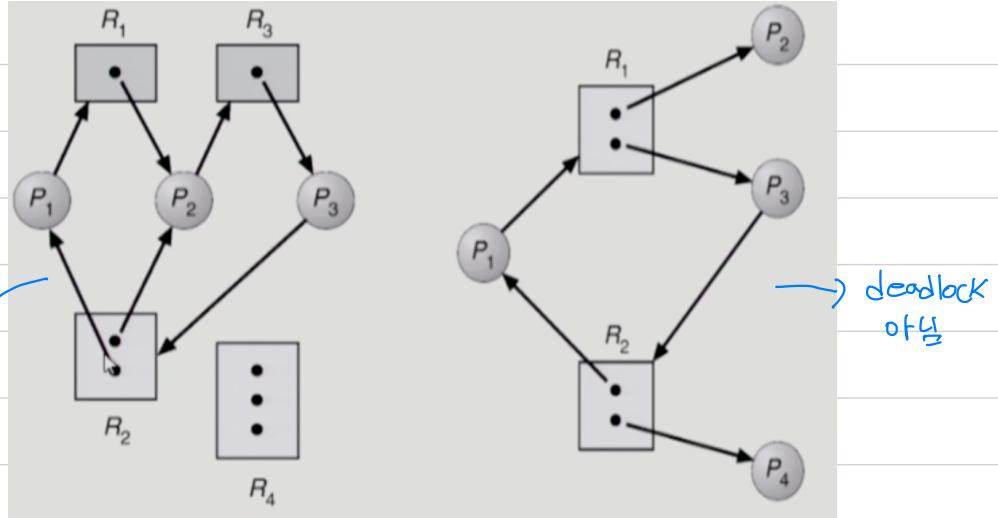
✓ Resource  $R = \{R_1, R_2, \dots, R_n\}$

## - Edge

✓ request edge  $P_i \rightarrow R_j$

✓ assignment edge  $R_j \rightarrow P_i$





- 그래프에 Cycle이 없으면 deadlock이 아니다.

- 그래프에 Cycle이 있으면

    v if only one instance per resource type, then deadlock

    v if several instances per resource type, possibility of deadlock

## Deadlock의 처리 방법

### - Deadlock Prevention

v 자원 할당 시 Deadlock의 4가지 필요 조건 중 어느 하나가 만족되지 않도록 하는 것

### - Deadlock Avoidance

v 자원 요청에 대한 부가적인 정보를 이용해서 deadlock의 가능성이 없는 경우에만 자원을 할당

v 시스템 state가 원래 state로 돌아올 수 있는 경우에만 자원 할당

### - Deadlock Detection and recovery

v Deadlock 발생은 허용하되 그에 대한 detection 무단을 두어 deadlock 발견시 recover

### - Deadlock Ignorance

v Dead lock을 시스템이 책임지지 않음

v UNIX를 포함한 대부분의 OS가 선택

    ↳ 사람이 알아서 해결

# Deadlock Prevention

## - Mutual Exclusion

- 公共资源는 안되는 자원의 경우 반드시 성립해야 함.

## - Hold and Wait

- 프로세스가 자원을 요청할 때 다른 어떤 자원도 가지고 있지 않아야 한다.
- 방법1. 프로세스 시작시 모든 필요한 자원을 할당 블록 하는 방법
- 방법2. 자원이 필요할 경우 보유 자원을 모두 놓고 다시 요청

## - No Preemption

- 프로세스가 어떤 자원을 기다려야 하는 경우 이미 보유한 자원이 선점됨
- 모든 필요한 자원을 얻을 수 있을 때 그 프로세스는 다시 시작된다.
- Starvation을 쉽게避免하고饥饿을 할 수 있는 자원에서 주로 사용 (CPU, memory)

## - Circular Wait

- 모든 자원 유형에 할당 순서를 정하여 정해진 순서대로만 자원 할당
- 예를 들어 순서가 3인 자원  $R_3$ 를 보유 중인 프로세스가 순서가 1인 자원  $R_1$ 을 할당받기 위해서는 우선  $R_3$ 을 Release해야 한다.

⇒ Utilization 저하, throughput 감소, starvation 문제

일어나지 않은 deadlock을  
마지 예방하기 때문에 상당히 비효율적

# Deadlock Avoidance

- 자원 요청에 대한 부가정보를 이용해서 자원 할당이 deadlock으로 봉쇄 안전(safe) 한지를 동적으로 조사해서 안전한 경우에만 할당
- 가장 단순하고 일반적인 모델은 프로세스들이 필요로 하는 각 자원별 최대 사용량을 미리 선언하도록 하는 방법론.

## • Safe state

- v 시스템 내의 프로세스들에 대한 **Safe sequence**가 존재하는 상태

## • Safe Sequence

- v 프로세스의 Sequence  $\langle P_1, P_2, P_3, \dots, P_n \rangle$  이 Safe 상태면  $P_i$  ( $1 \leq i \leq n$ ) 의 자원 요청이 "가용자원 + 모든  $P_j$  ( $j < i$ )의 모유자원"에 의해 충족되어야 함
  - 조건을 만족한다면 다음 방법으로 모든 프로세스의 수행을 보장
    - $P_i$ 의 자원 요청이 즉시 충족될 수 없으면 모든  $P_j$  ( $j < i$ )가 풀려갈 때까지 기다린다.
    - $P_{n+1}$ 이 풀려오면  $P_1$ 의 자원을 양도시켜 수행한다.

## • 시스템이 Safe state에 있으면

⇒ NO deadlock

## • 시스템이 Unsafe state에 있으면

⇒ Possibility of deadlock

## • Dead lock Avoidance

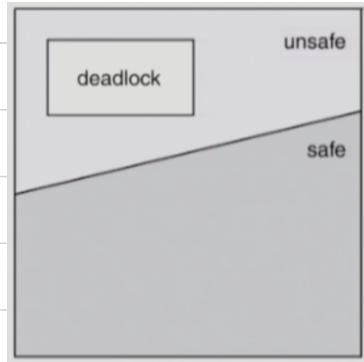
- v 시스템이 unsafe state에 들어가지 않는 것을 보장

- v 2가지 경우의 avoidance 알고리즘

- Single instance per resource types
    - Resource Allocation Graph Algorithm 사용

- Multiple instances per resource types

- Banker's Algorithm 사용



# Resource Allocation Graph Algorithm

## • Claim edge $P_i \rightarrow R_j$

- ✓ 프로세스  $P_i$  가 자원  $R_j$ 를 미래에 요청할 수 있음을 표시(점선으로 표시)
- ✓ 프로세스가 해당 자원 요청시 request edge로 바뀜(실선)
- ✓  $R_j$  가 Release하면 assignment edge는 다시 claim edge로 바뀐다.

• Request edge의 assignment edge 변경시 (점선을 포함하여) Cycle이 생기지 않는 경우에만 요청 자원을 할당한다.

• Cycle 생성 여부 조사시 프로세스의 수가 기일때  $O(n^2)$  시간이 걸린다.

