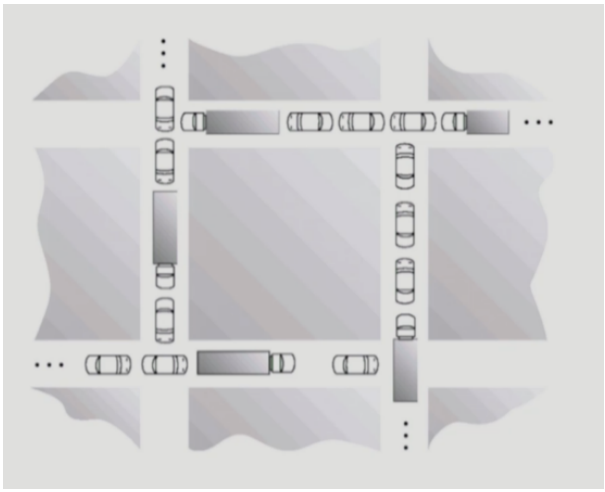
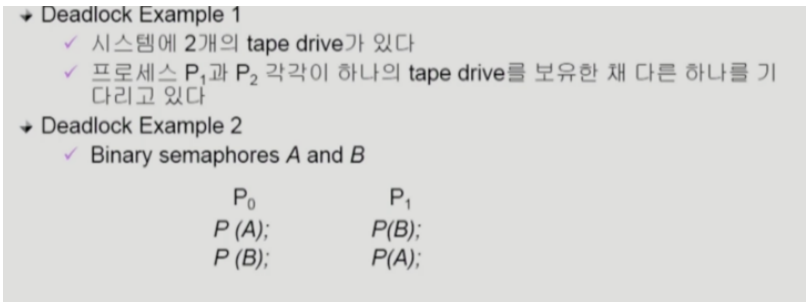
DeadLock (교착상태)

일련의 프로세스들이 서로가 가진 자원을 기다리며 Block 된 상태

자원 => 하드웨어/소프트웨어등 포함하는 모든 자원

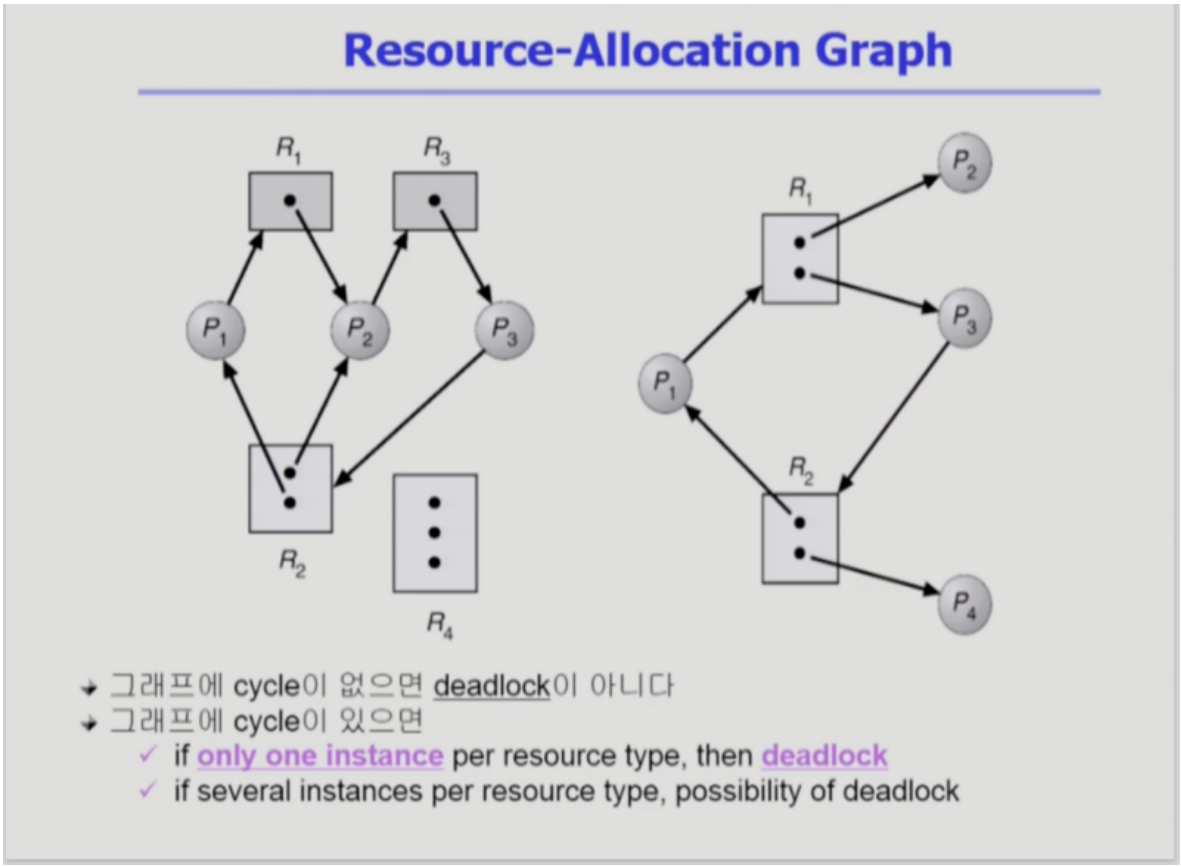




Deadlock 발생 조건 4 (필요충분조건)

- 상호배제(Mutual exclusion) - 프로세스 혼자만 독점적으로 자원을 사용해야함
- 비선점 (No Preemption) - 자원선점 후 빼앗기지 않음
- 보유대기 (Hold and wait) - 프로세스가 보유자원을 놓지 않고 추가적인 자원을 기다림
- 순환대기 (Circular wait) - 자원을 기다리는 프로세스간 사이클 형성

<순환대기 모형>



R -> P : 자원이 프로세스에 할당됨

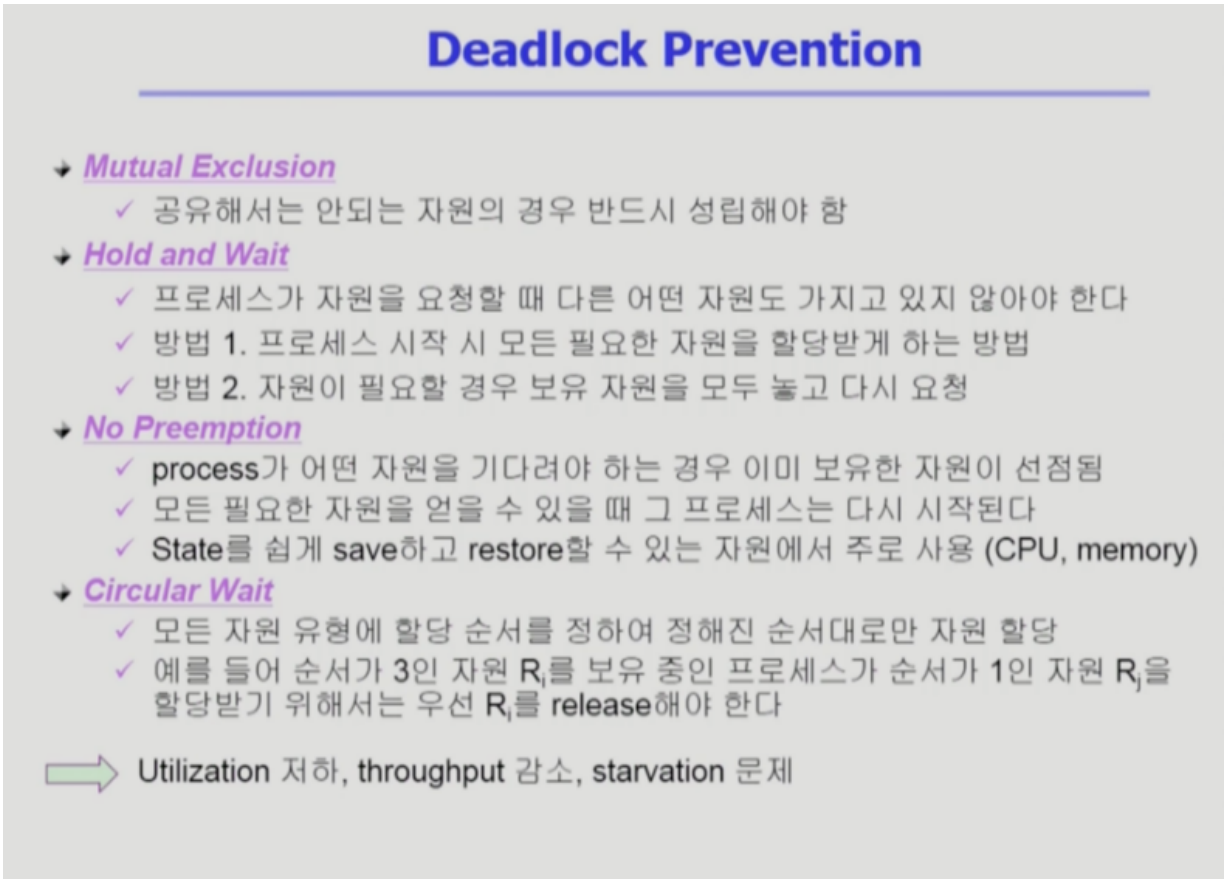
P - >R : 프로세스가 자원에 할당을 요청함

Deadlock 처리방법

- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection and recovery
- Deadlock Ignorance

1. DEADLOCK PREVENTION

= Deadlock이 발생하는 조건을 알고 미연에 방지함

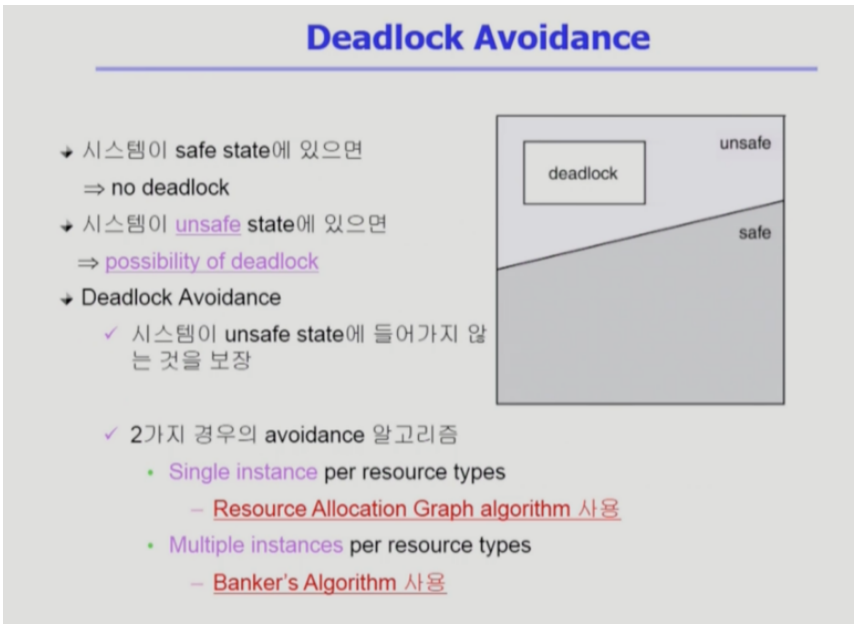


- circular wait ex) 1,3,5 자원이 필요한 경우 낮은 번호 부터 자원을 획득하는 순서를 정해놓음

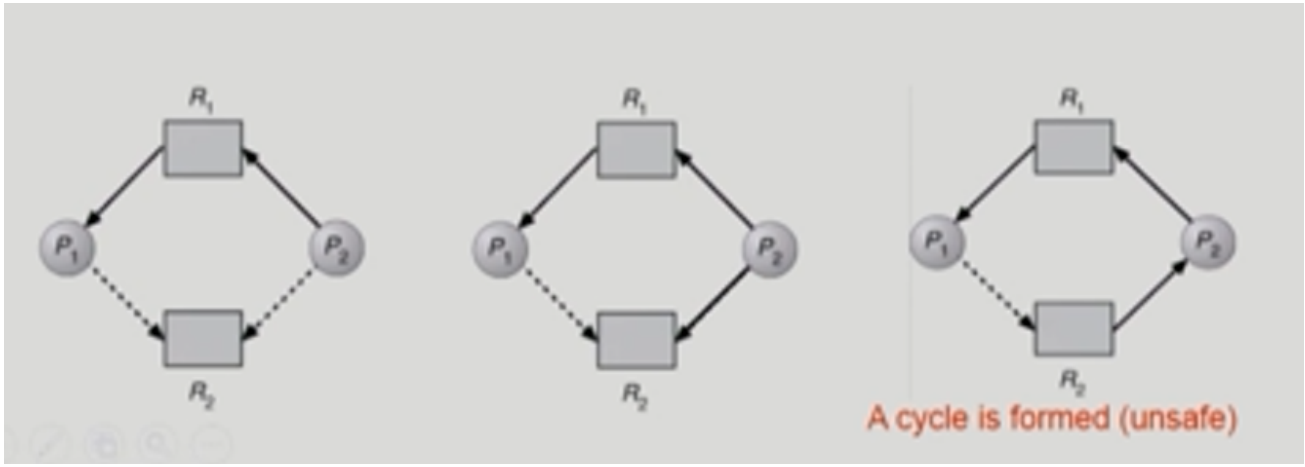
=> 미리 데드락을 방지하는 방법은 비효율적이다.

2. DEADLOCK AVOIDANCE

= 프로세스가 시작 될 때 프로세스 평생동안 쓸 자원의 최대량을 알고 자원할당을 결정함

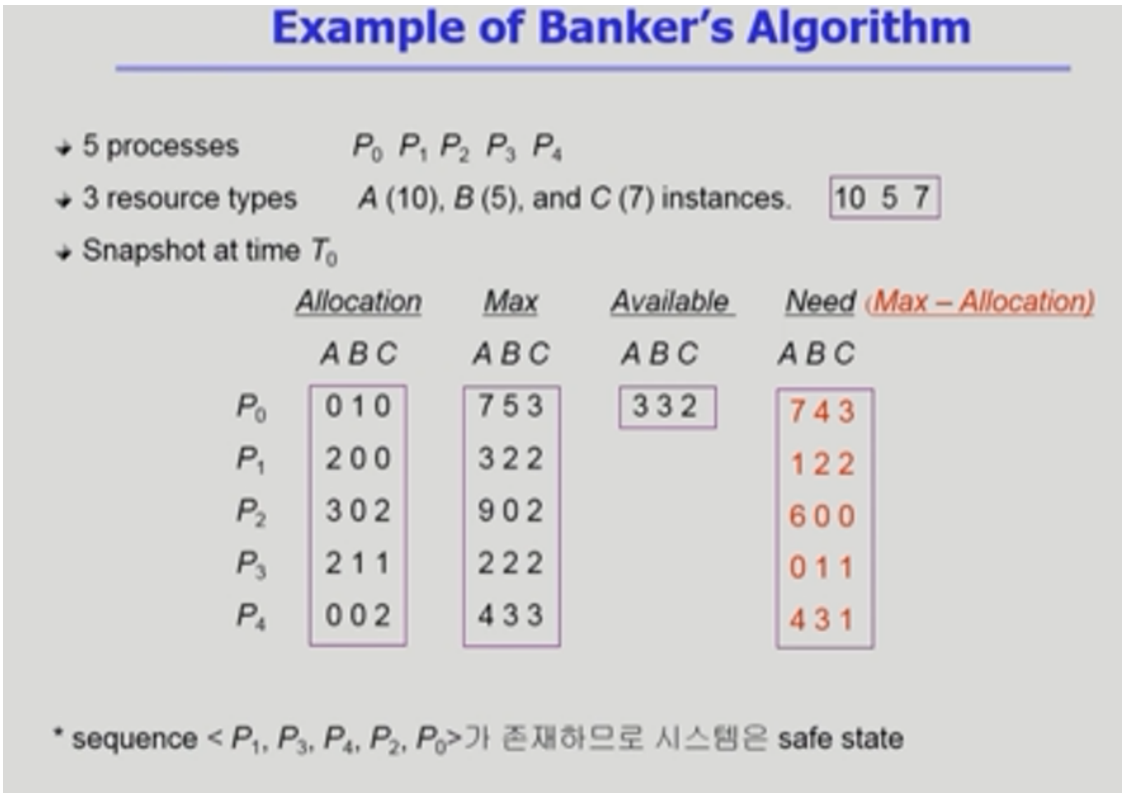


1. 자원당 하나의 인스턴스가 있는 경우



2. 자원당 여러개 인스턴스

Banker's Algorithm



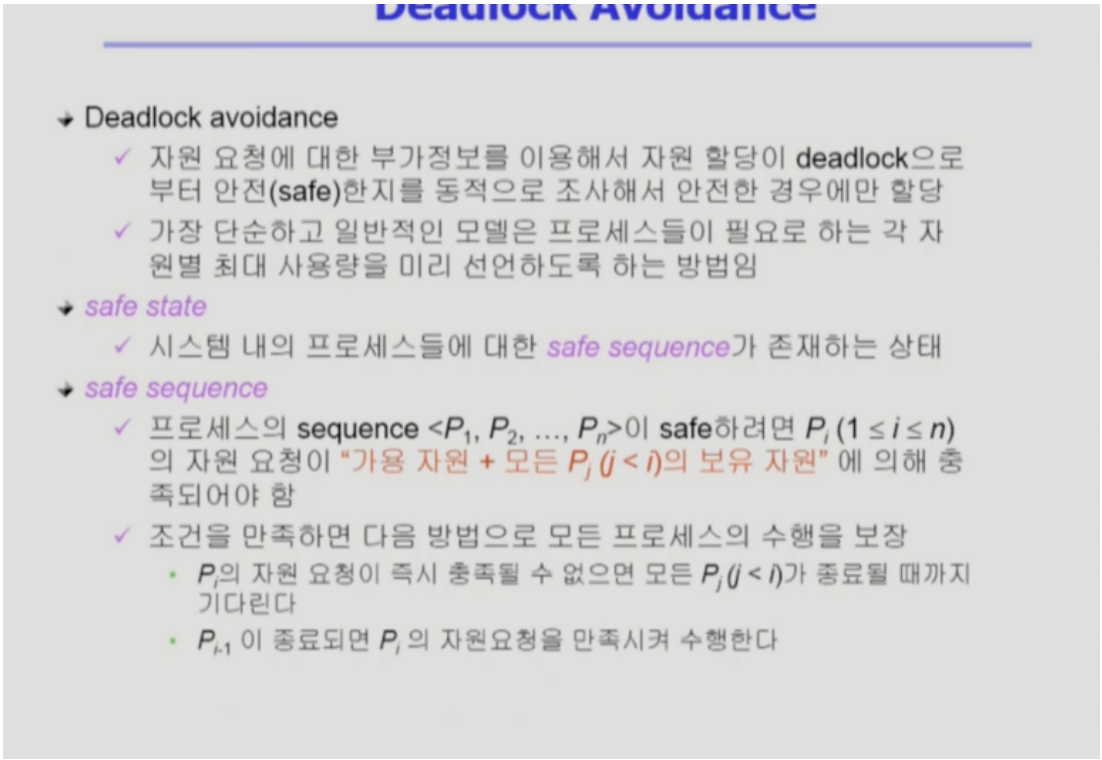
보수적, 최악의 상황을 가정하는 알고리즘

가용자원(Available)보다 많을 가능성이 있는 요청 (Need Max-Alllocation) 은 무조건 거부함

ex) available 332 일 때 P0가 1만 요청해도 안들어줌

가용자원이 남아도 최대가능요청자원을 넘어설 때까지 안줌

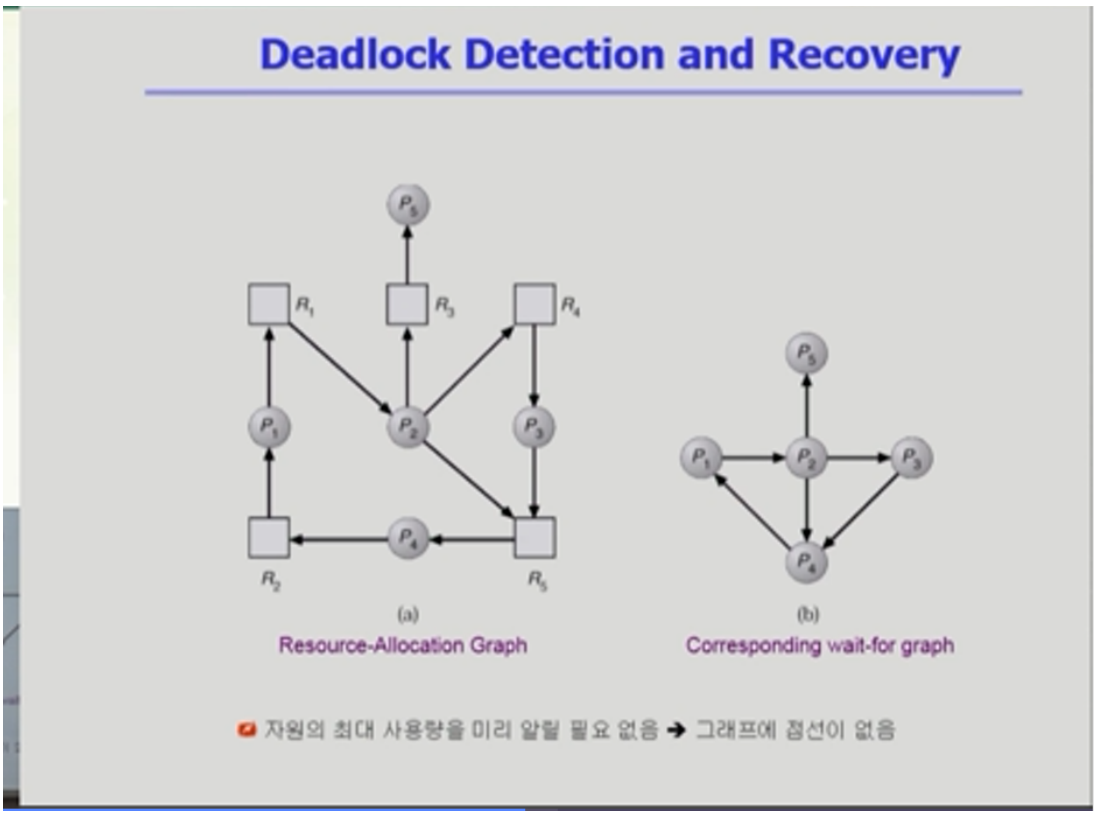
프로세스의 최대 요청자원을 순서대로 다 처리할수 있는 시퀀스가 존제하면 Safe state



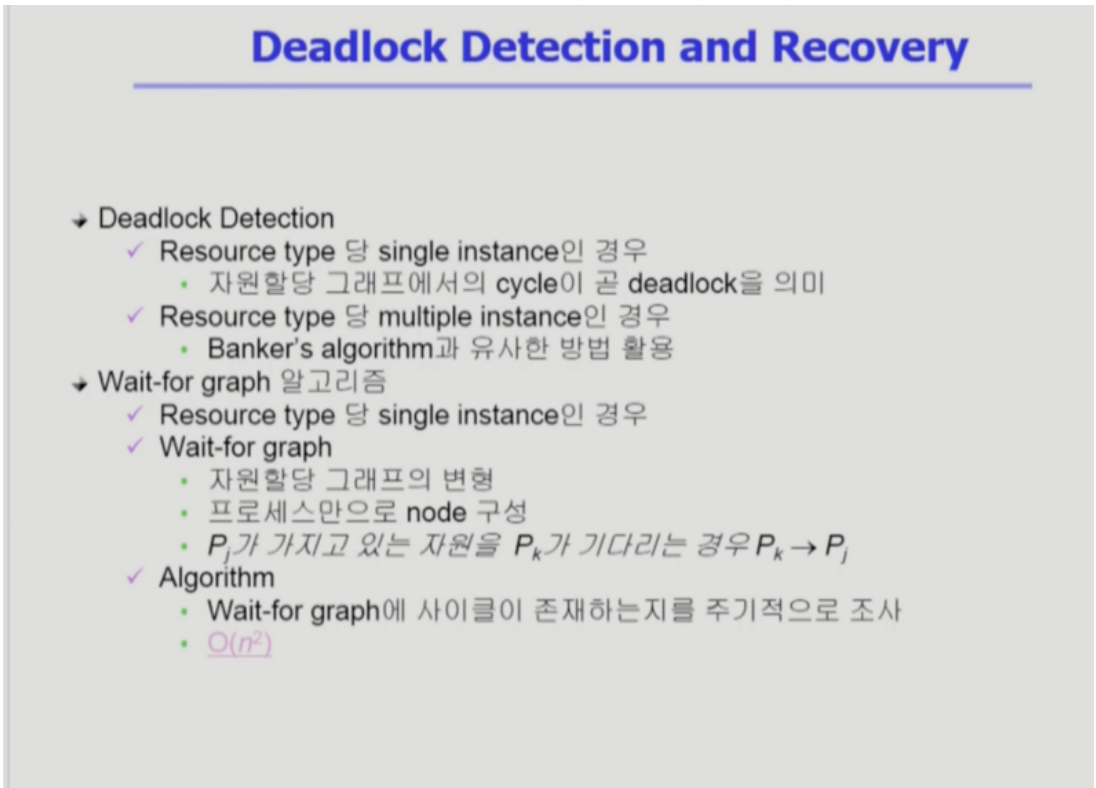
3. DEADLOCK DETECTION AND RECOVERY

1) Detection

- detection 은 그래프/뱅커스 둘다 이용해서 찾을 수 있음



- 그래프는 resource를 빼고 process들로만 연결한 축약형 그래프로도 표현가능



=> DFS/BFS 처럼 최대 O(n^2) 만큼 걸림

- 프로세스당 자원이 여러개 일 때

Deadlock Detection and Recovery

- Resource type 당 multiple instance인 경우
  - ✓ 5 processes: P_0 P_1 P_2 P_3 P_4
  - ✓ 3 resource types: A (7), B (2), and C (6) instances
  - ✓ Snapshot at time T_0:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

  - ✓ No deadlock: sequence <P_0, P_2, P_3, P_1, P_4> will work!

"*Request*"는 추가요청가능량이 아니라 현재 실제로 요청한 자원량을 나타냄

2) Recovery

Deadlock Detection and Recovery

- Recovery
  - ✓ ***Process termination***
    - Abort all deadlocked processes
    - Abort one process at a time until the deadlock cycle is eliminated
  - ✓ ***Resource Preemption***
    - 비용을 최소화할 victim의 선정
    - safe state로 rollback하여 process를 restart
    - Starvation 문제
      - 동일한 프로세스가 계속해서 victim으로 선정되는 경우
      - cost factor에 rollback 횟수도 같이 고려

- process termination - 프로세스 죽이기

1. 전부 죽이기

2. 하나씩 죽여보기

- Resource preemption

  희생양 프로세스를 선정하고 자원을 빼앗음

  이 때, Starvation 문제 (우선순위가 낮은 프로세스는 오랫동안 자원할당을 받지 못함)

  => 대기시간과 뺏긴 횟수등 다양한 요소를 고려해서 순서 정함

4. Deadlock Ignorance

Deadlock이 일어나든 말든 무시하는

prevention 은 오버헤드가 너무 큼

detection 역시 조금만 느려져도 detection 루틴이 실행되면 이것역시 시스템 오버헤드가 너무 큼

사용자에게 핸들링을 맡김