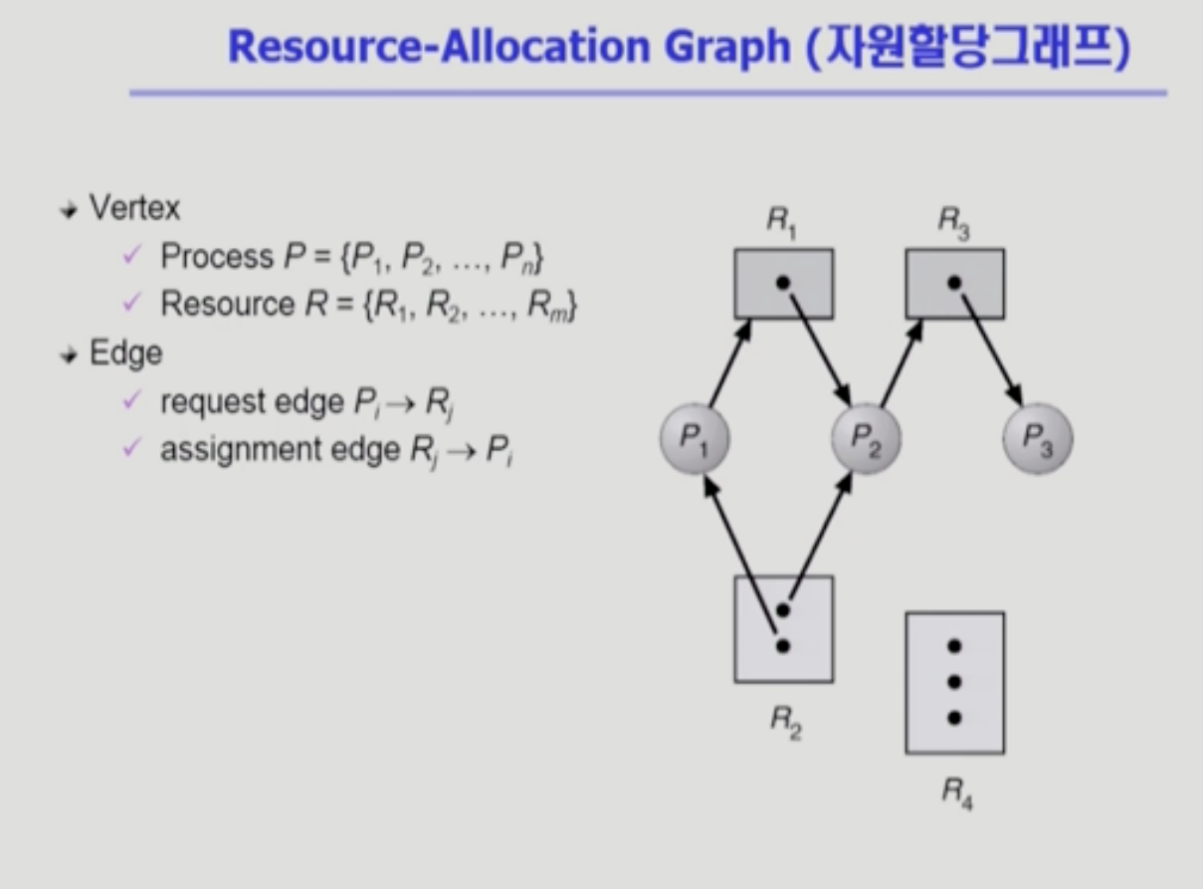
07 - Dead Lock

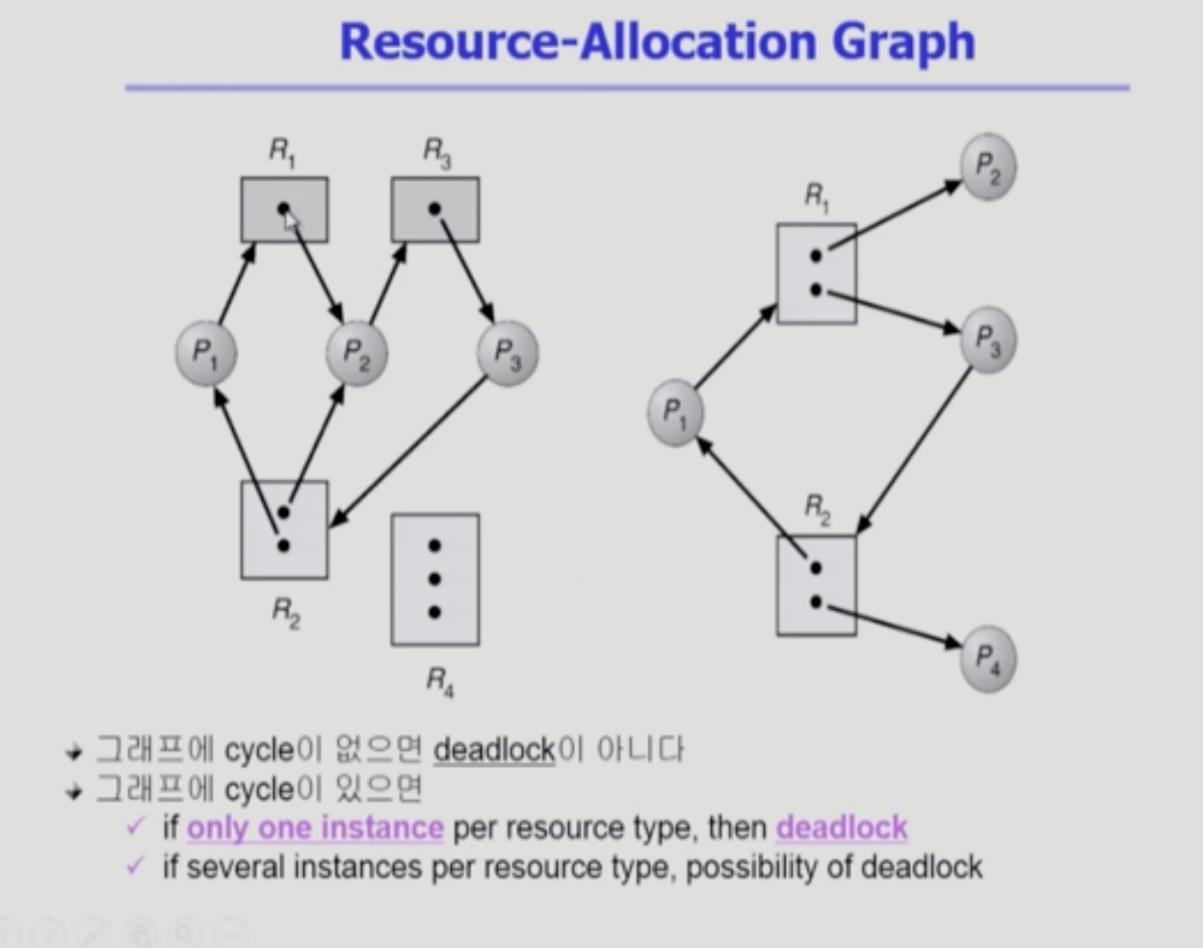
일련의 프로세스들이 서로가 가진 자원을 기다리며 blocking 상태

자원 : 하드웨어, 소프트웨어 등을 포함하는 개념 (semaphore 도 포함됨)

발생 조건

- Mutual exclusion - 상호배제
- No preemption - 비선점 (빼앗기지 않는다)
- Hold and wait - 보유 대기
- Circular wait - 순환 대기





- 사이클이 없으면 deadlock이 아님
- 사이클이 있더라도 두인스턴스가 있다면 아닐수도 있음 (오른쪽 그림은 아님)

처리방법

1. Deadlock prevention

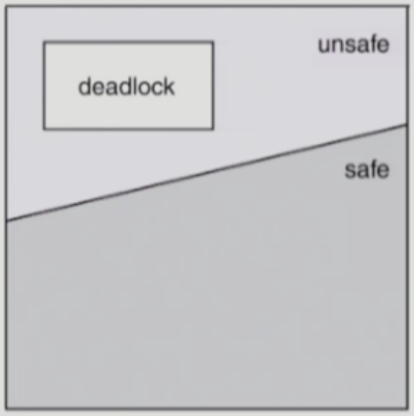
- ***Mutual Exclusion***
  - ✓ 공유해서는 안되는 자원의 경우 반드시 성립해야 함
- ***Hold and Wait***
  - ✓ 프로세스가 자원을 요청할 때 다른 어떤 자원도 가지고 있지 않아야 한다
  - ✓ 방법 1. 프로세스 시작 시 모든 필요한 자원을 할당받게 하는 방법
  - ✓ 방법 2. 자원이 필요할 경우 보유 자원을 모두 놓고 다시 요청
- ***No Preemption***
  - ✓ process가 어떤 자원을 기다려야 하는 경우 이미 보유한 자원이 선점됨
  - ✓ 모든 필요한 자원을 얻을 수 있을 때 그 프로세스는 다시 시작된다
  - ✓ State를 쉽게 save하고 restore할 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_i를 보유 중인 프로세스가 순서가 1인 자원 R_j을 할당받기 위해서는 우선 R_i를 release해야 한다

⇨ Utilization 저하, throughput 감소, starvation 문제

2. Deadlock Aviodance

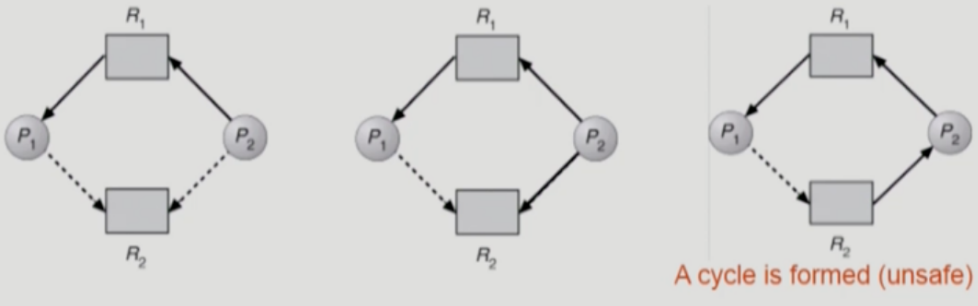
- Deadlock avoidance
  - ✓ 자원 요청에 대한 부가정보를 이용해서 자원 할당이 deadlock으로부터 안전(safe)한지를 동적으로 조사해서 안전한 경우에만 할당
  - ✓ 가장 단순하고 일반적인 모델은 프로세스들이 필요로 하는 각 자원별 최대 사용량을 미리 선언하도록 하는 방법임
- *safe state*
  - ✓ 시스템 내의 프로세스들에 대한 *safe sequence*가 존재하는 상태
- *safe sequence*
  - ✓ 프로세스의 sequence $<P_1, P_2, \ldots, P_n>$이 safe하려면 P_i $(1 \le i \le n)$의 자원 요청이 "가용 자원 + 모든 P_j $(j < i)$의 보유 자원" 에 의해 충족되어야 함
  - ✓ 조건을 만족하면 다음 방법으로 모든 프로세스의 수행을 보장
    - P_i의 자원 요청이 즉시 충족될 수 없으면 모든 P_j $(j < i)$가 종료될 때까지 기다린다
    - P_{i-1} 이 종료되면 P_i 의 자원요청을 만족시켜 수행한다

- 시스템이 safe state에 있으면
  ⇒ no deadlock
- 시스템이 unsafe state에 있으면
  ⇒ possibility of deadlock
- Deadlock Avoidance
  - ✓ 시스템이 unsafe state에 들어가지 않는 것을 보장



  - ✓ 2가지 경우의 avoidance 알고리즘
    - Single instance per resource types
      - Resource Allocation Graph algorithm 사용
    - Multiple instances per resource types
      - Banker's Algorithm 사용

- ***Claim edge*** $P_i \rightarrow R_j$
  - ✓ 프로세스 P_i가 자원 R_j를 미래에 요청할 수 있음을 뜻함 (점선으로 표시)
  - ✓ 프로세스가 해당 자원 요청시 request edge로 바뀜 (실선)
  - ✓ R_j가 release되면 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 n일 때 $O(n^2)$ 시간이 걸린다



자원 할당시 데드락 여부를 조사해서 발생할 수 있으면 자원을 할당하지 않음

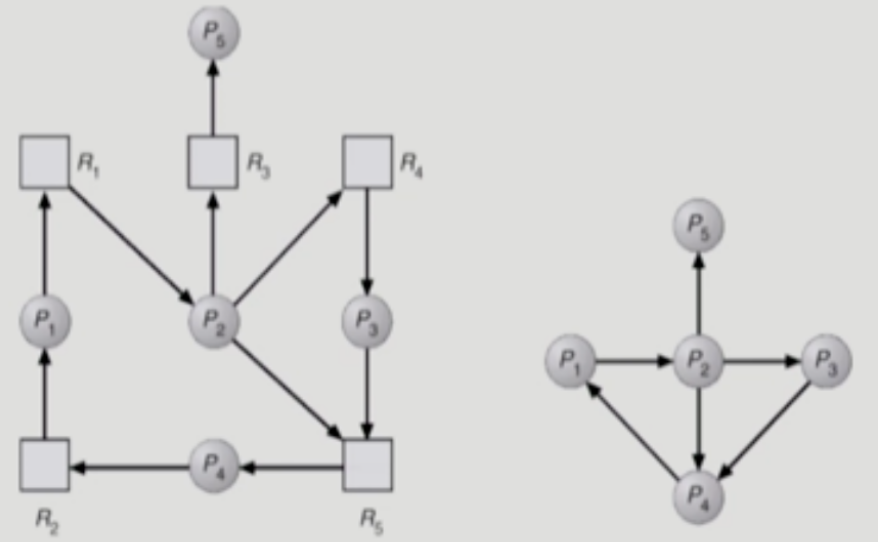
- 5 processes P_0 P_1 P_2 P_3 P_4
- 3 resource types A (10), B (5), and C (7) instances. 10 5 7
- Snapshot at time T_0

	Allocation	Max	Available	Need (Max – Allocation)
	A B C	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2	7 4 3
P_1	2 0 0	3 2 2		1 2 2
P_2	3 0 2	9 0 2		6 0 0
P_3	2 1 1	2 2 2		0 1 1
P_4	0 0 2	4 3 3		4 3 1

* sequence < P_1, P_3, P_4, P_2, P_0>가 존재하므로 시스템은 safe state

가용자원으로 처리되지 않는 경우 주지 않는 방식으로 함.

3. Deadlock Detection and recovery



자원의 최대 사용량을 미리 알릴 필요 없음 → 그래프에 점선이 없음

사이클이 있는 상태를 확인해서 복원하는 방식

- 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

- 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

4. Deadlock Ignorance - 대부분 OS가 이 방식을 채택함

- Deadlock이 일어나지 않는다고 생각하고 아무런 조치도 취하지 않음
  - ✓ Deadlock이 매우 드물게 발생하므로 deadlock에 대한 조치 자체가 더 큰 overhead일 수 있음
  - ✓ 만약, 시스템에 deadlock이 발생한 경우 시스템이 비정상적으로 작동하는 것을 사람이 느낀 후 직접 process를 죽이는 등의 방법으로 대처
  - ✓ UNIX, Windows 등 대부분의 범용 OS가 채택