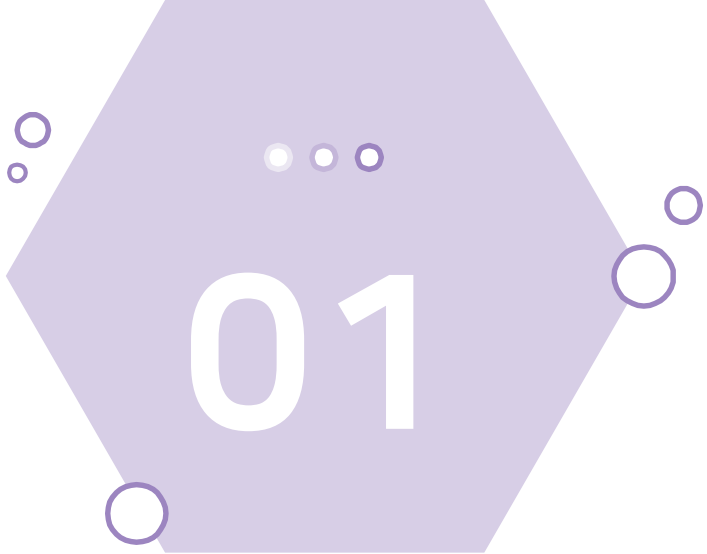
# 7강. 교착상태 II

방송대 컴퓨터과학과
김진욱 교수

# 목차




# 교착상태 회피

# 교착상태 회피

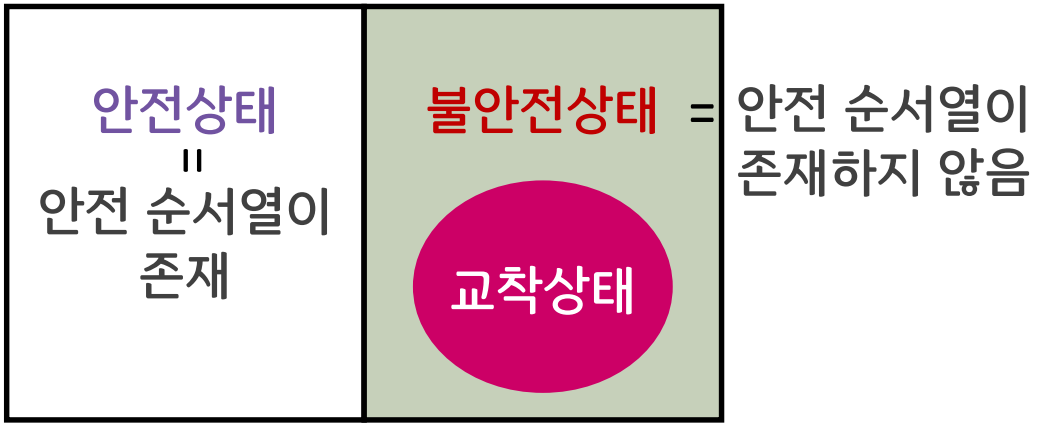
## 교착상태 회피

- 프로세스의 자원 사용에 대한 사전 정보를 활용하여 교착상태가 발생하지 않는 상태에 머물도록 하는 방법
  - » 사전 정보: 현재 할당된 자원, 가용상태의 자원, 프로세스들의 최대 요구량

## 프로세스의 상태 영역

### ✪ 안전상태

교착상태를 회피하면서 각 프로세스에게 그들의 최대 요구량까지 빠짐없이 자원을 할당할 수 있는 상태

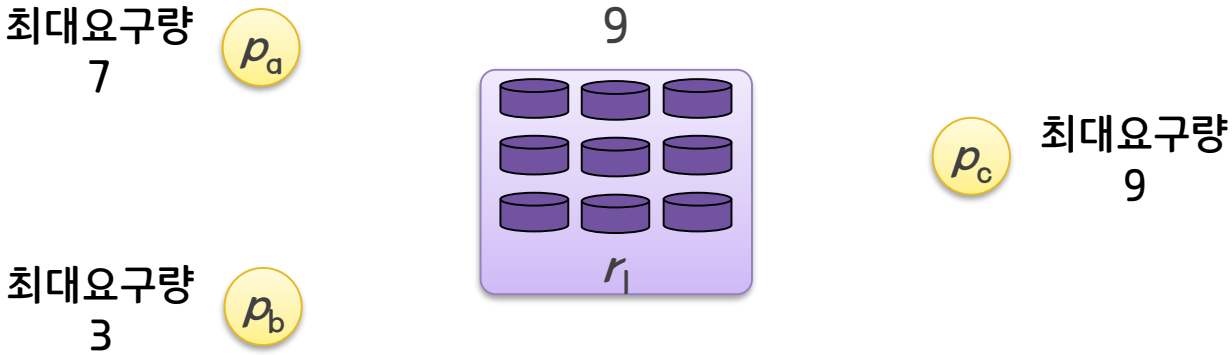


# 교착상태 회피



## 안전 순서열

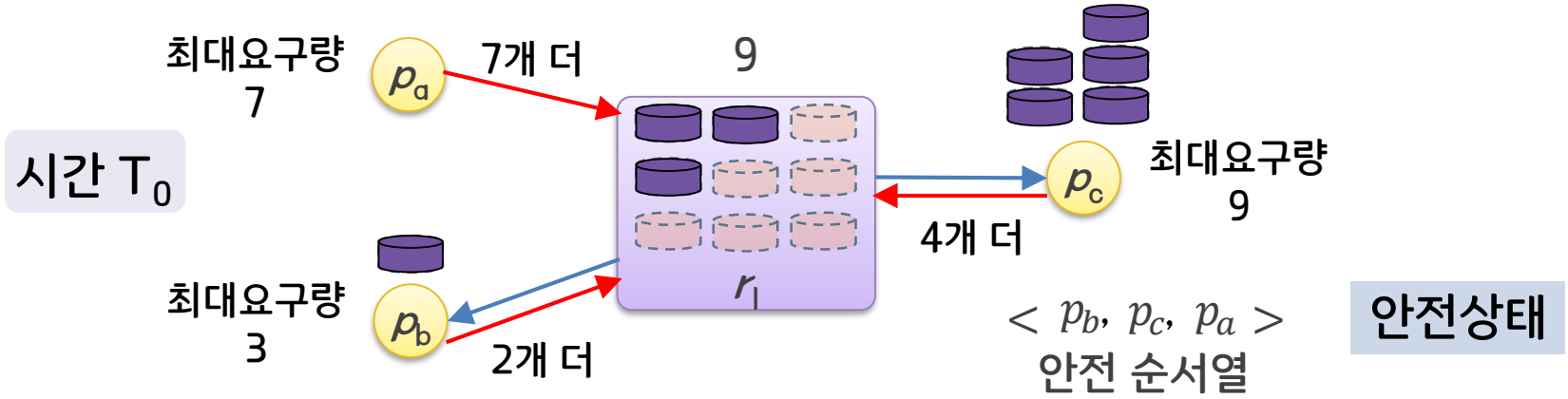
- 순서 있는 프로세스의 집합 $< p_1, p_2, \cdots, p_n >$
- 각 $p_i$에 대해, $p_i$가 추가로 요구할 수 있는 자원 소요량이 현재 가용 상태이거나 혹은 현재 가용인 자원에 $p_j$(단, $j < i$)에 할당된 자원까지 포함하여 할당 가능한 경우



# 교착상태 회피

## 안전 순서열

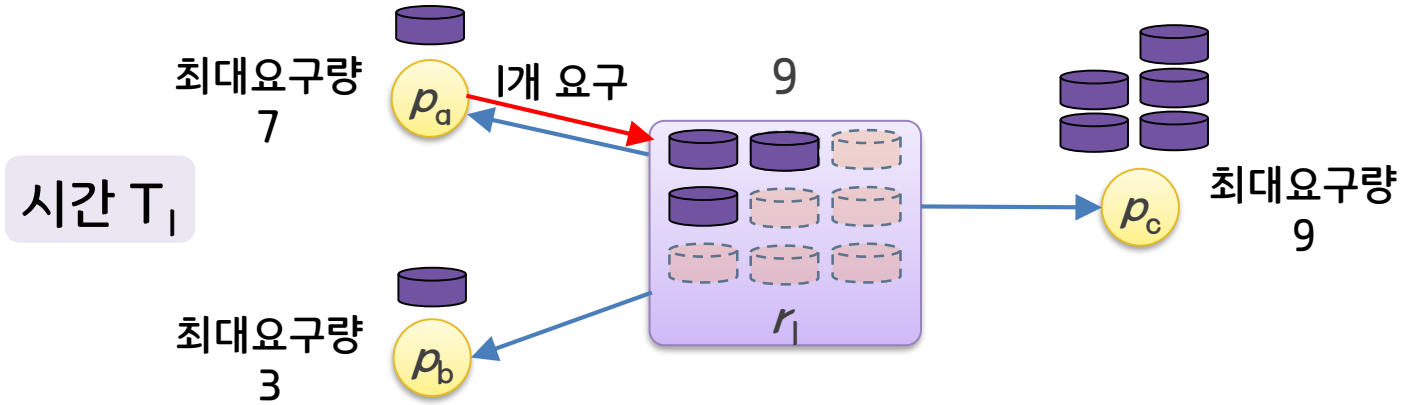
- 순서 있는 프로세스의 집합 $< p_1, p_2, \cdots, p_n >$
- 각 $p_i$에 대해, $p_i$가 추가로 요구할 수 있는 자원 소요량이 현재 가용 상태이거나 혹은 현재 가용인 자원에 $p_j$(단, $j < i$)에 할당된 자원까지 포함하여 할당 가능한 경우



# 교착상태 회피

## 안전 순서열

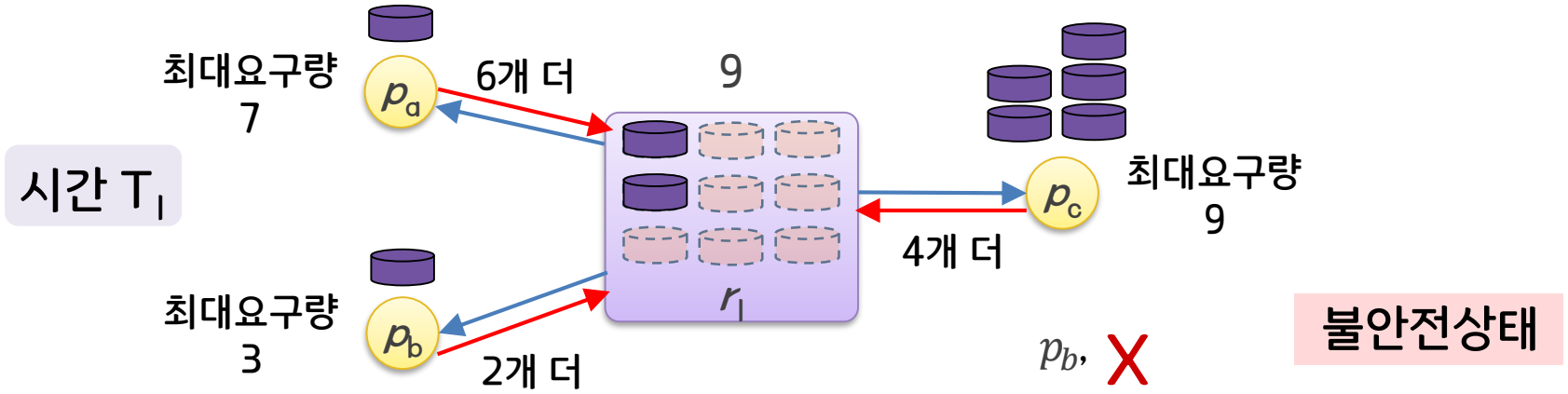
- 순서 있는 프로세스의 집합 $< p_1, p_2, \cdots, p_n >$
- 각 $p_i$에 대해, $p_i$가 추가로 요구할 수 있는 자원 소요량이 현재 가용 상태이거나 혹은 현재 가용인 자원에 $p_j$(단, $j < i$)에 할당된 자원까지 포함하여 할당 가능한 경우



# 교착상태 회피

## 안전 순서열

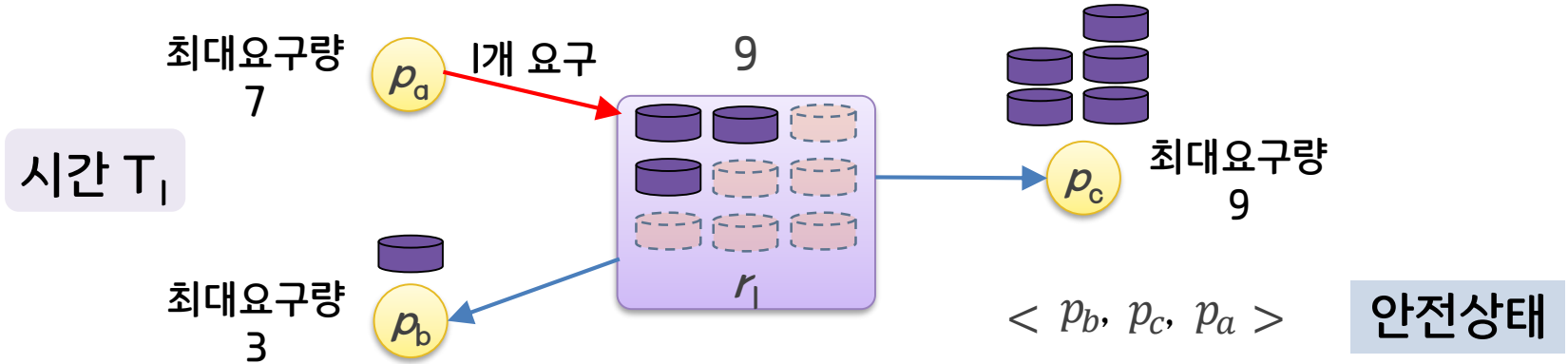
- 순서 있는 프로세스의 집합 $< p_1, p_2, \cdots, p_n >$
- 각 $p_i$에 대해, $p_i$가 추가로 요구할 수 있는 자원 소요량이 현재 가용 상태이거나 혹은 현재 가용인 자원에 $p_j$(단, $j < i$)에 할당된 자원까지 포함하여 할당 가능한 경우



# 교착상태 회피

## 안전 순서열

- 순서 있는 프로세스의 집합 $< p_1, p_2, \cdots, p_n >$
- 각 $p_i$에 대해, $p_i$가 추가로 요구할 수 있는 자원 소요량이 현재 가용 상태이거나 혹은 현재 가용인 자원에 $p_j$(단, $j < i$)에 할당된 자원까지 포함하여 할당 가능한 경우



# 교착상태 회피

- 교착상태는 불안전상태에서 발생
  - 불안전상태: 할당 과정에 따라 교착상태가 될 수도 있는 상태

- 프로세스가 가용상태의 자원을 요구하더라도
  안전상태를 유지하기 위해 프로세스는 대기상태가 될 수 있음

# 교착상태 회피 알고리즘



- 각 자원 유형의 단위자원이 여러 개일 경우
  - 은행원 알고리즘

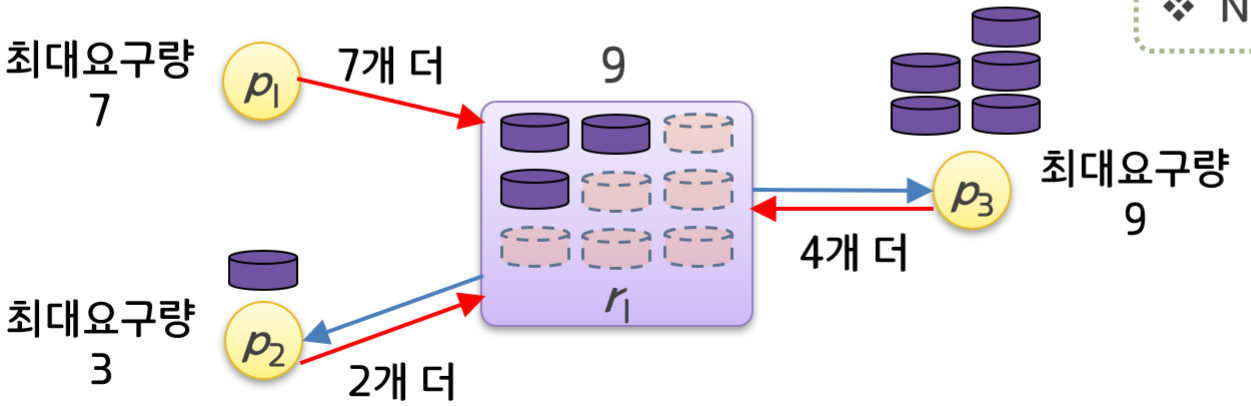
- 각 자원 유형의 단위자원이 하나밖에 없는 경우
  - 변형된 자원할당 그래프

# 각 자원 유형의 단위자원이 여러 개일 경우

## 은행원 알고리즘

- 자원을 요청받으면 그 자원을 할당해 주고 난 후의 상태를 계산해서 그것이 안전상태가 보장되는 경우에만 자원을 할당

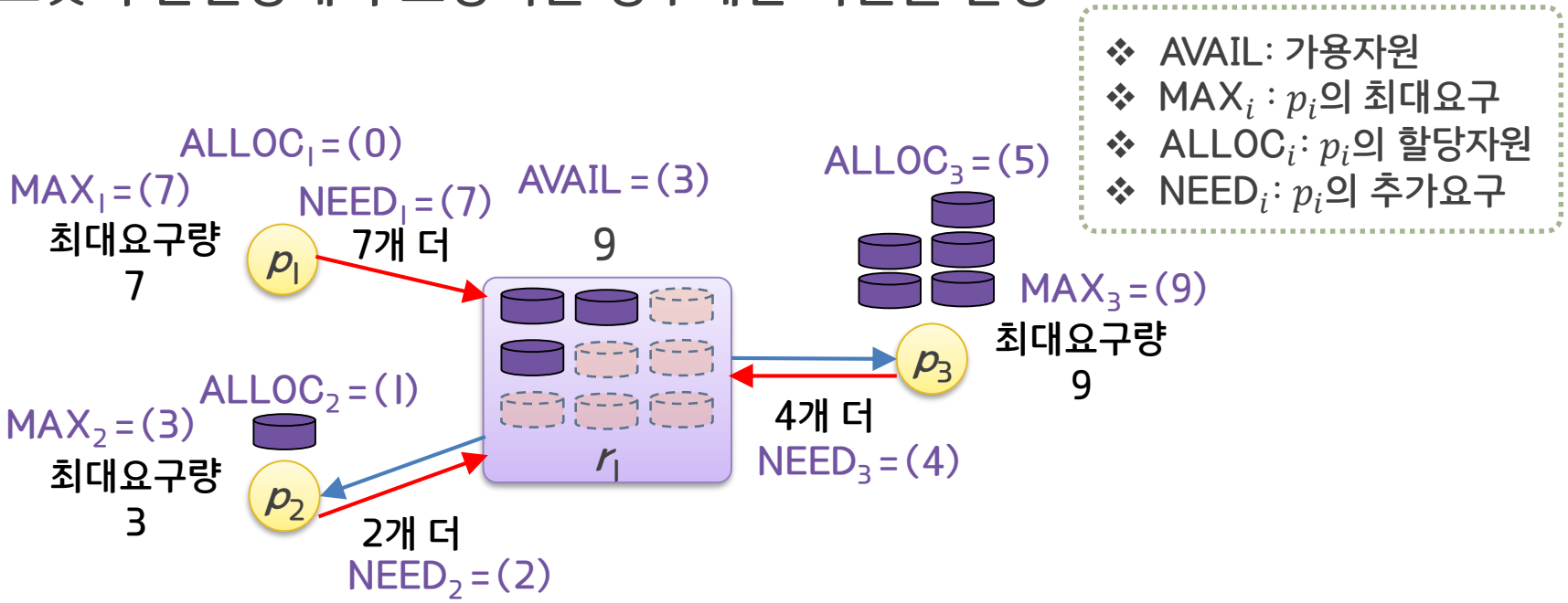
❖ AVAIL: 가용자원
❖ $MAX_i$ : $p_i$의 최대요구
❖ $ALLOC_i$: $p_i$의 할당자원
❖ $NEED_i$: $p_i$의 추가요구



# 각 자원 유형의 단위자원이 여러 개일 경우

## 은행원 알고리즘

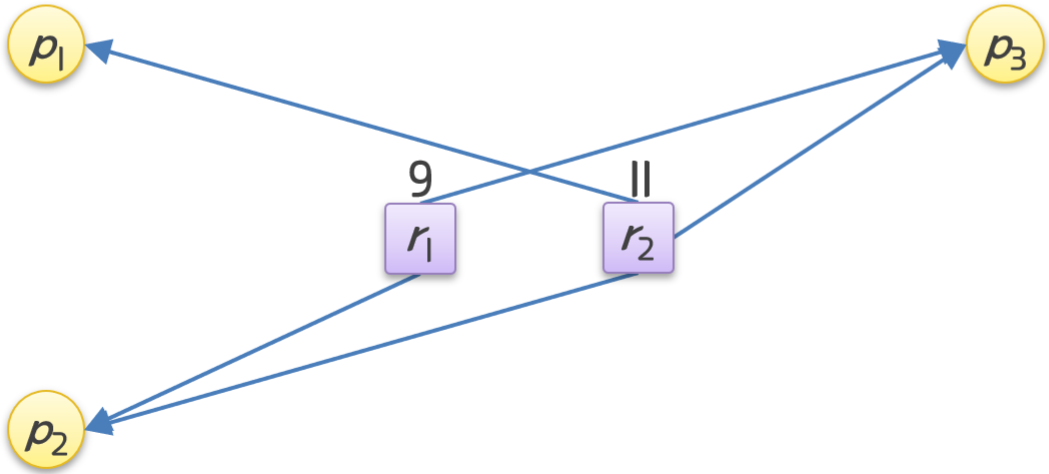
- 자원을 요청받으면 그 자원을 할당해 주고 난 후의 상태를 계산해서 그것이 안전상태가 보장되는 경우에만 자원을 할당



# 각 자원 유형의 단위자원이 여러 개일 경우

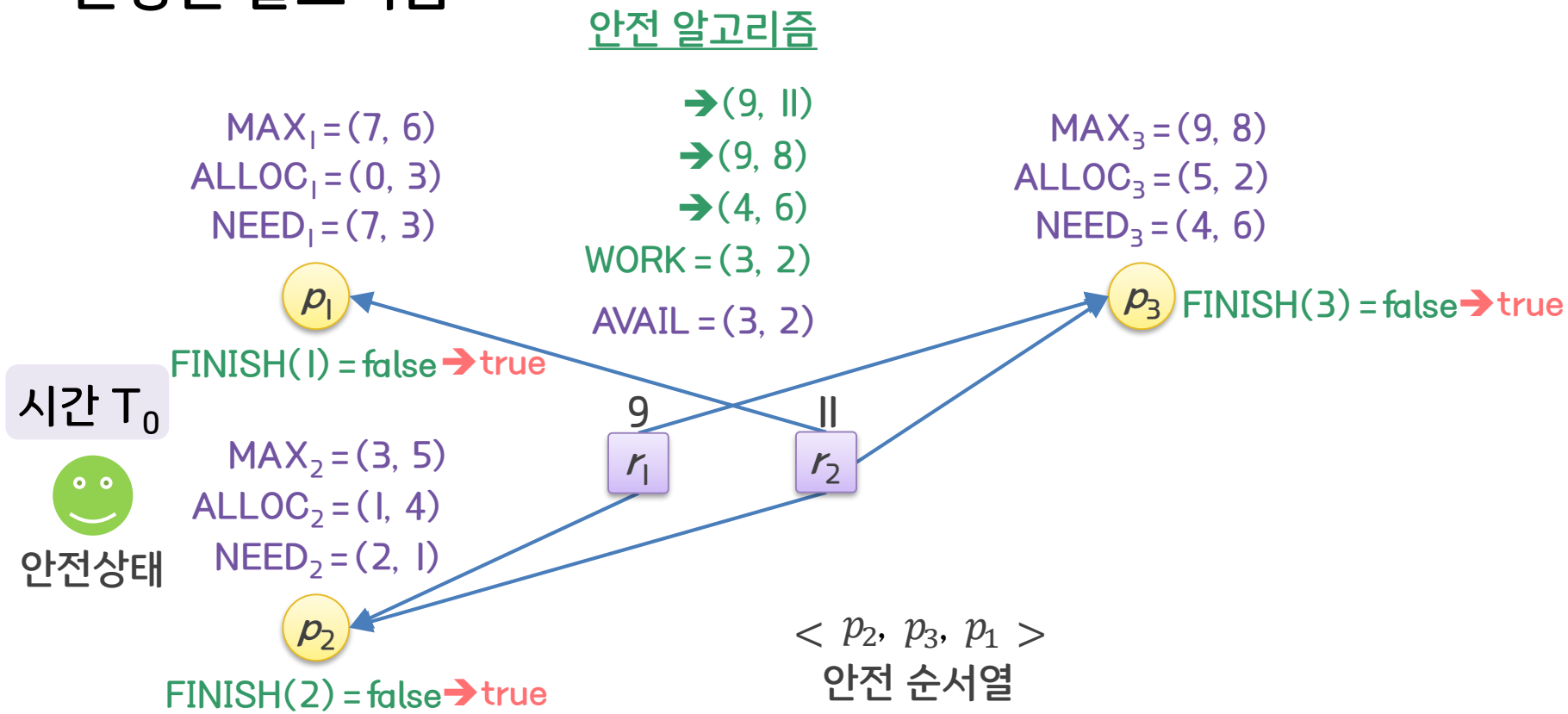
- 은행원 알고리즘

시간 $T_0$



# 각 자원 유형의 단위자원이 여러 개일 경우

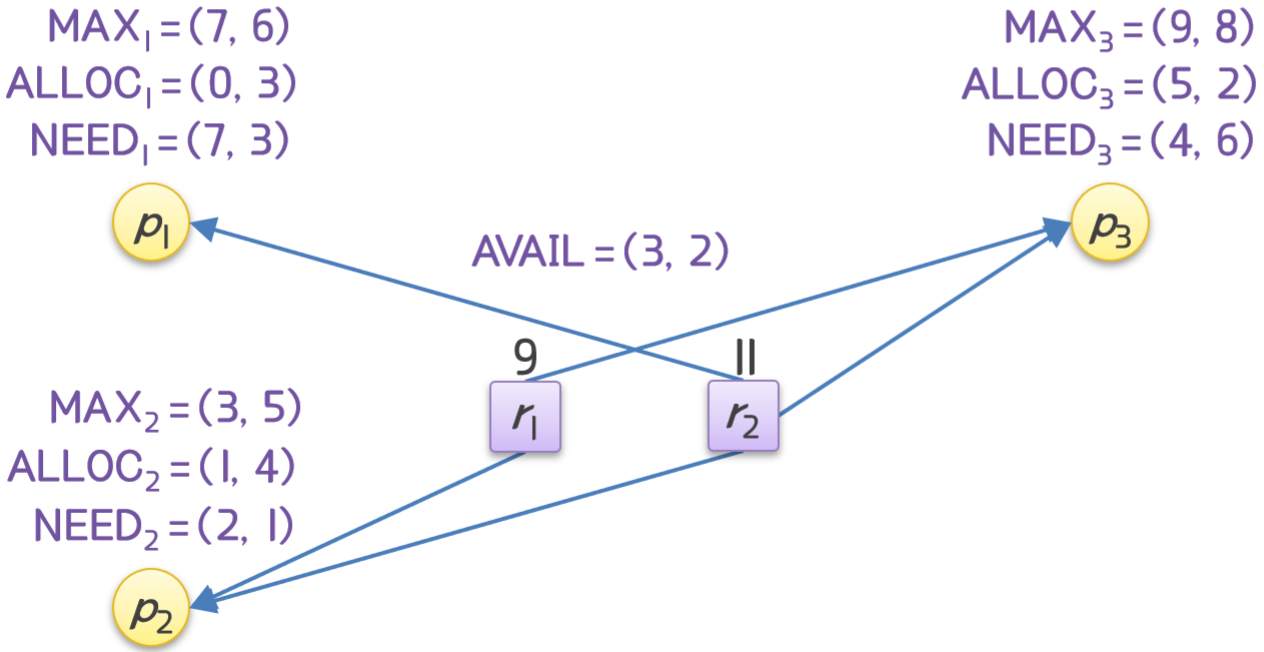
- 은행원 알고리즘



# 각 자원 유형의 단위자원이 여러 개일 경우

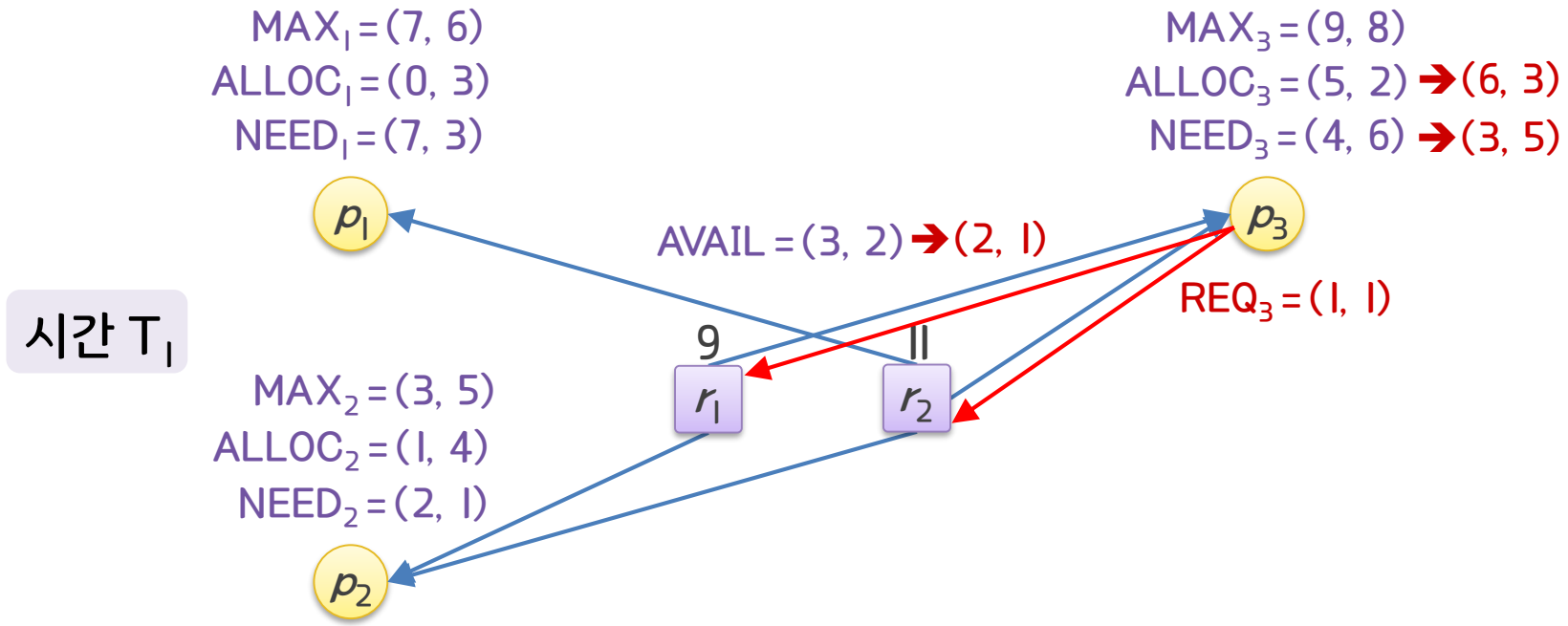
- 은행원 알고리즘

시간 $T_1$



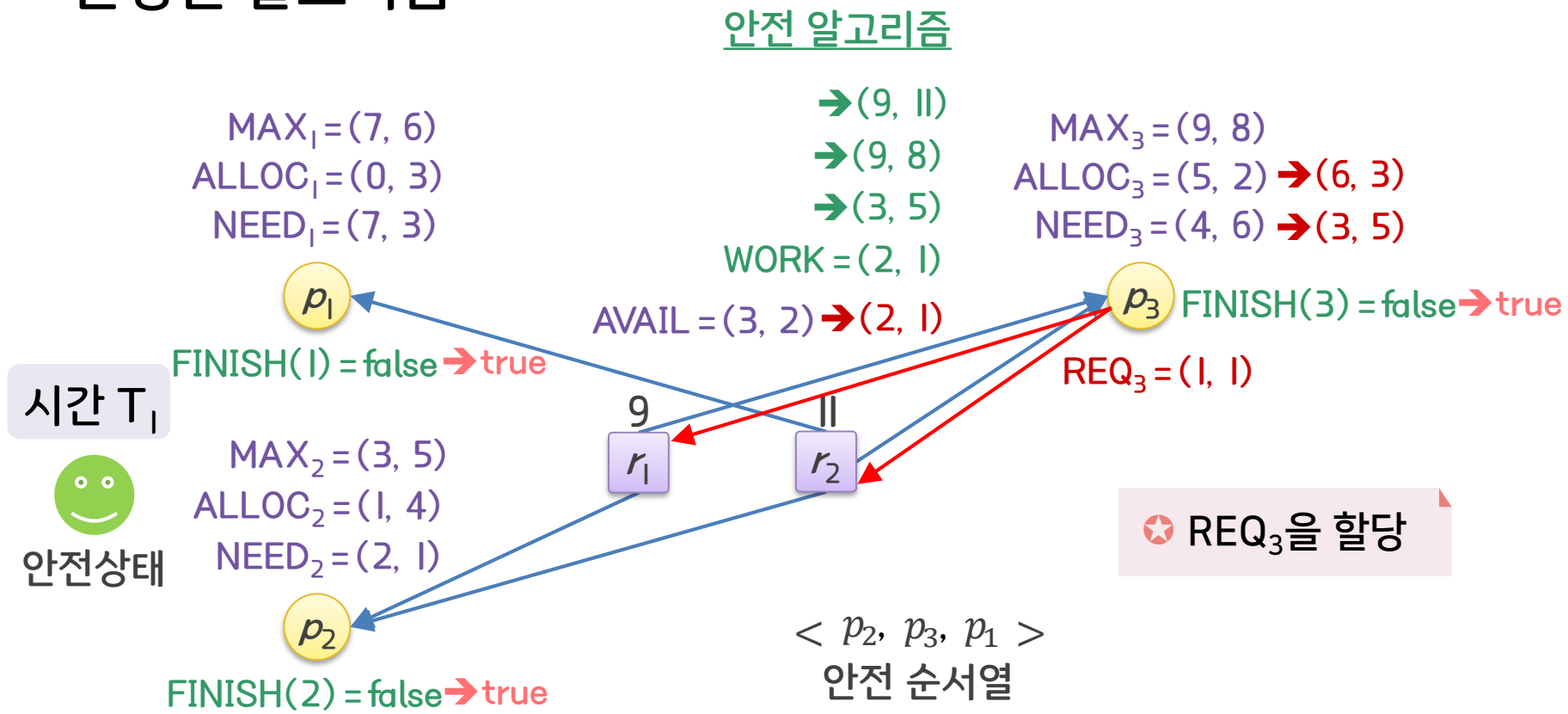
# 각 자원 유형의 단위자원이 여러 개일 경우

- 은행원 알고리즘



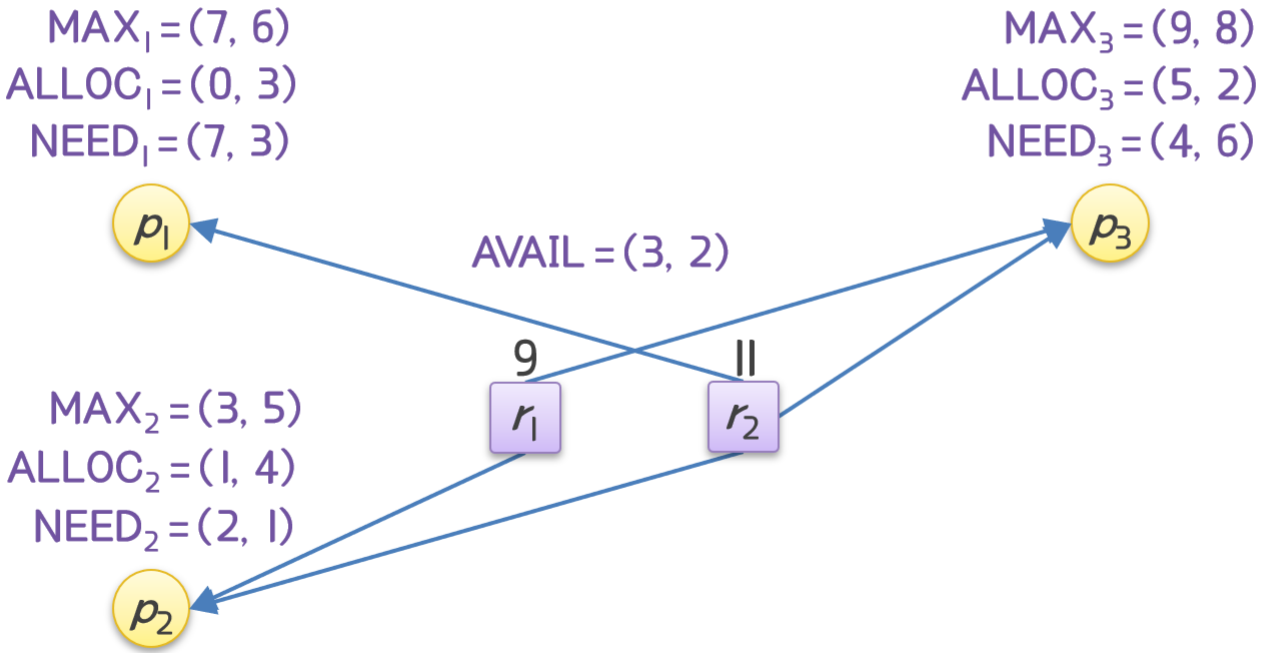
# 각 자원 유형의 단위자원이 여러 개일 경우

## 은행원 알고리즘



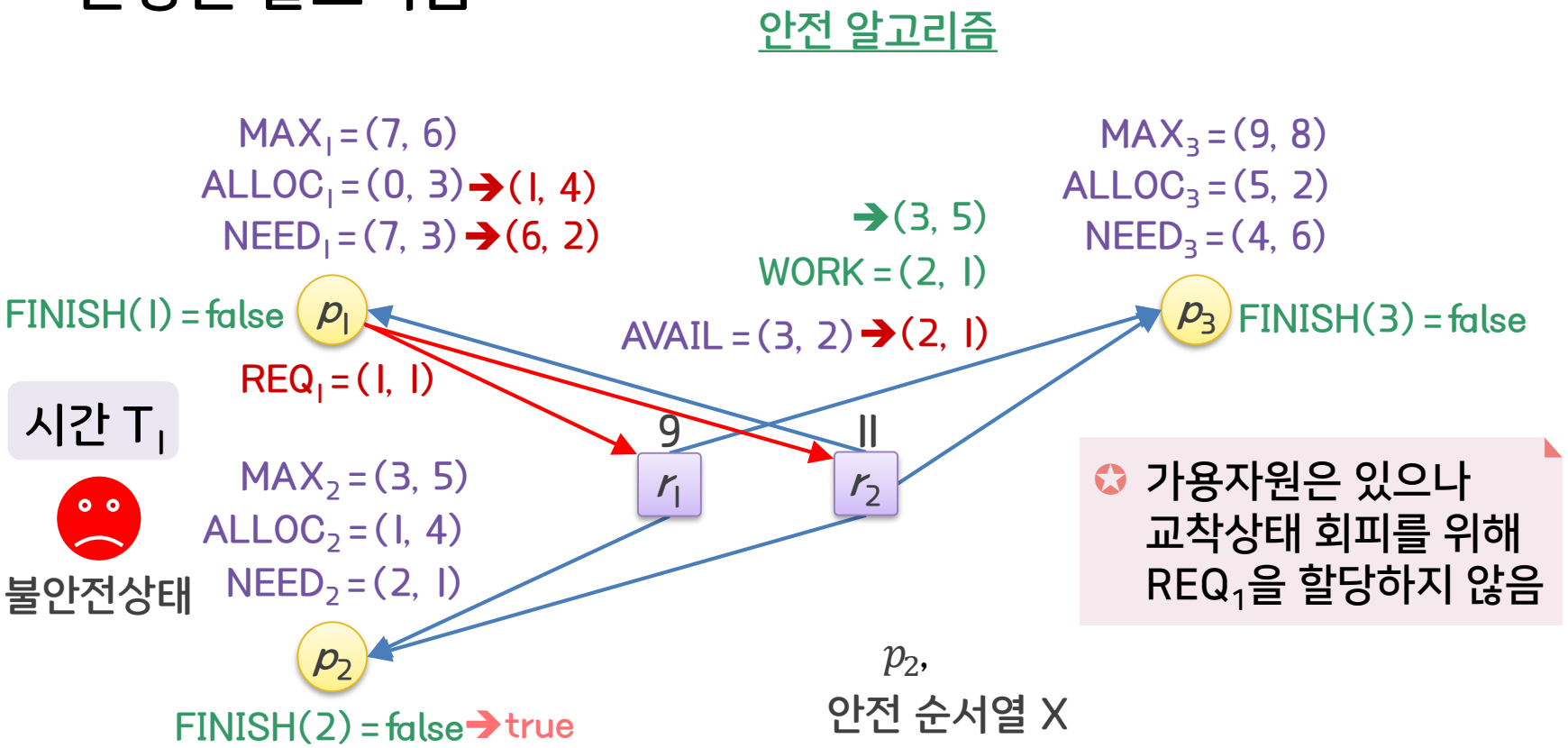
# 각 자원 유형의 단위자원이 여러 개일 경우

- 은행원 알고리즘



# 각 자원 유형의 단위자원이 여러 개일 경우

## 은행원 알고리즘



# 각 자원 유형의 단위자원이 여러 개일 경우

## 은행원 알고리즘

✪ 시간 복잡도: $O(mn^2)$

① $REQ_i \leq NEED_i$가 거짓이면 오류

② $REQ_i \leq AVAIL$이 거짓이면 $p_i$는 대기

③ $REQ_i \leq AVAIL$이면

③-1 다음과 같이 할당 후와 같은 상태를 만듦

$AVAIL \leftarrow AVAIL - REQ_i$

$ALLOC_i \leftarrow ALLOC_i + REQ_i$

$NEED_i \leftarrow NEED_i - REQ_i$

③-2 이 상태가 안전상태인지를 조사

③-3 안전상태이면 $REQ_i$를 할당

③-4 그렇지 않으면 프로세스를 대기상태로, 데이터 구조는 이전 상태로 복구

## 안전 알고리즘

① 길이가 각각 $m$, $n$인 WORK와 FINISH 초기화

$WORK \leftarrow AVAIL$

$FINISH(i) \leftarrow$ `false`, $i = 1, 2, \cdots, n$

② $FINISH(i)$ = false이고 $NEED_i \leq WORK$인 $i$ 찾기

그런 $i$가 없으면 go to ④

③ $WORK \leftarrow WORK + ALLOC_i$

$FINISH(i) \leftarrow$ `true`

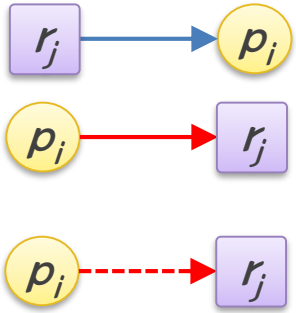
go to ②

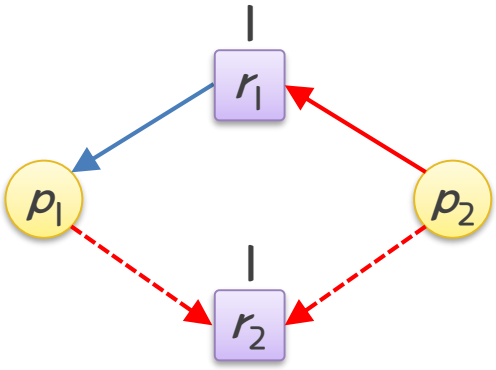
④ 모든 $i$에 대하여 $FINISH(i)$ = `true`이면 안전상태

# 각 자원 유형의 단위자원이 하나밖에 없는 경우

## 변형된 자원할당 그래프

- 할당간선 $(r_j, p_i)$ : 자원 $r_j$가 프로세스 $p_i$에 할당됨
- 요구간선 $(p_i, r_j)$ : 프로세스 $p_i$가 자원 $r_j$를 요구함
- 선언간선 $(p_i, r_j)$ : 앞으로 프로세스 $p_i$가 자원 $r_j$를 요구하게 될 것임



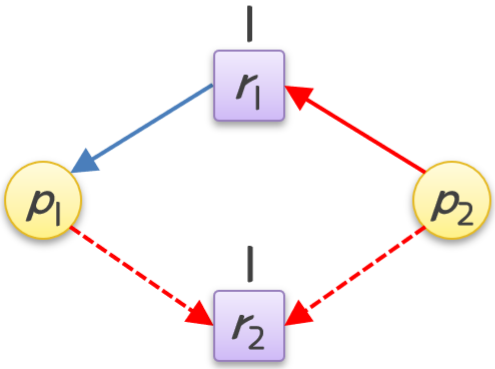



# 각 자원 유형의 단위자원이 하나밖에 없는 경우

## 변형된 자원할당 그래프

- 자원을 요청받으면 그 요구간선을 할당간선으로 변환하여도 사이클이 발생되지 않는 경우에만 자원을 할당

- $p_1$이 $r_2$를 요구하는 경우

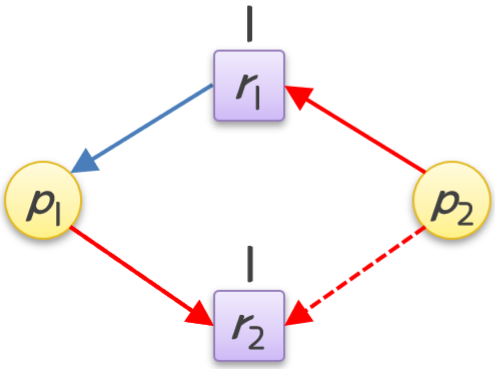


# 각 자원 유형의 단위자원이 하나밖에 없는 경우

## 변형된 자원할당 그래프

- 자원을 요청받으면 그 요구간선을 할당간선으로 변환하여도 사이클이 발생되지 않는 경우에만 자원을 할당

- $p_1$이 $r_2$를 요구하는 경우

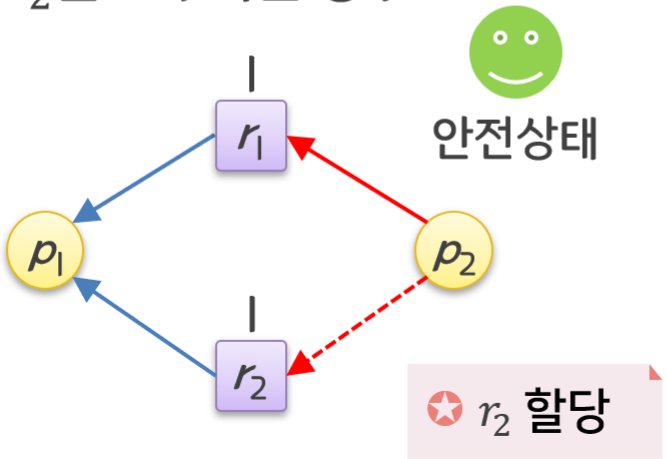


# 각 자원 유형의 단위자원이 하나밖에 없는 경우

## 변형된 자원할당 그래프

- 자원을 요청받으면 그 요구간선을 할당간선으로 변환하여도 사이클이 발생되지 않는 경우에만 자원을 할당

- $p_1$이 $r_2$를 요구하는 경우

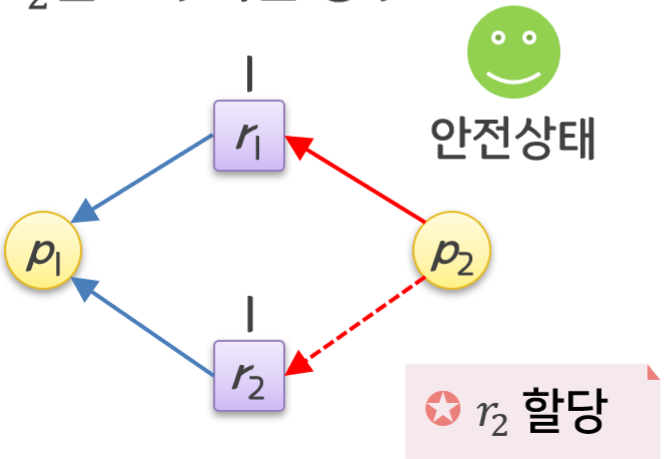


# 각 자원 유형의 단위자원이 하나밖에 없는 경우

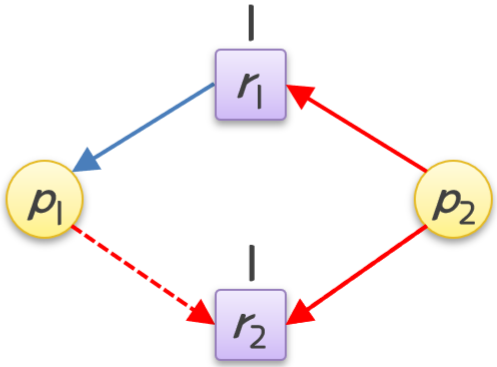
## 변형된 자원할당 그래프

- 자원을 요청받으면 그 요구간선을 할당간선으로 변환하여도 사이클이 발생되지 않는 경우에만 자원을 할당

- $p_1$이 $r_2$를 요구하는 경우




- $p_2$가 $r_2$를 요구하는 경우

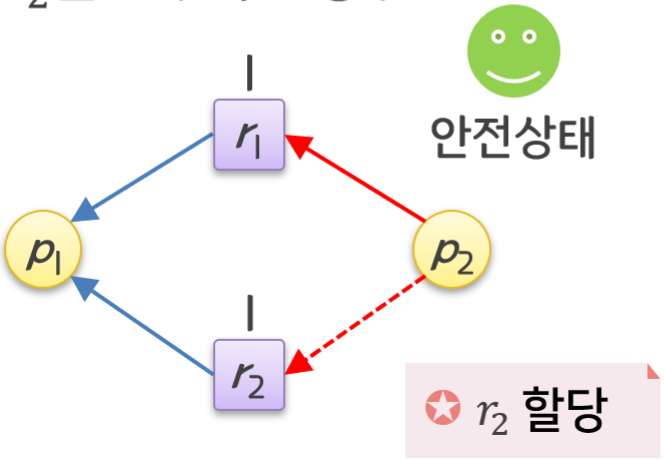


# 각 자원 유형의 단위자원이 하나밖에 없는 경우

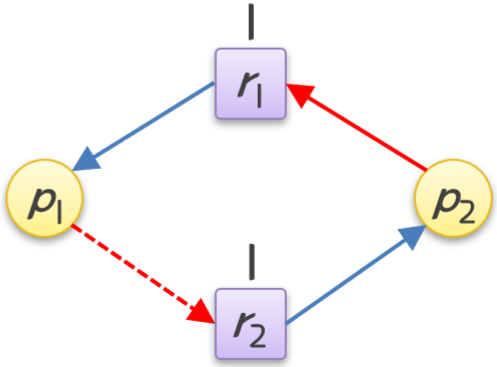
## 변형된 자원할당 그래프

- 자원을 요청받으면 그 요구간선을 할당간선으로 변환하여도 사이클이 발생되지 않는 경우에만 자원을 할당

- $p_1$이 $r_2$를 요구하는 경우




- $p_2$가 $r_2$를 요구하는 경우

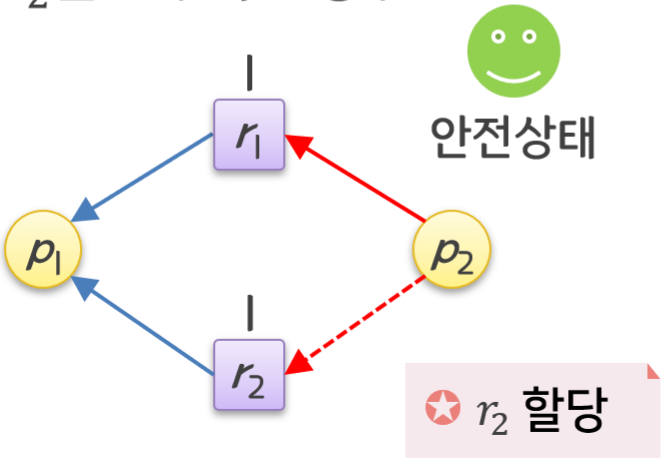


# 각 자원 유형의 단위자원이 하나밖에 없는 경우

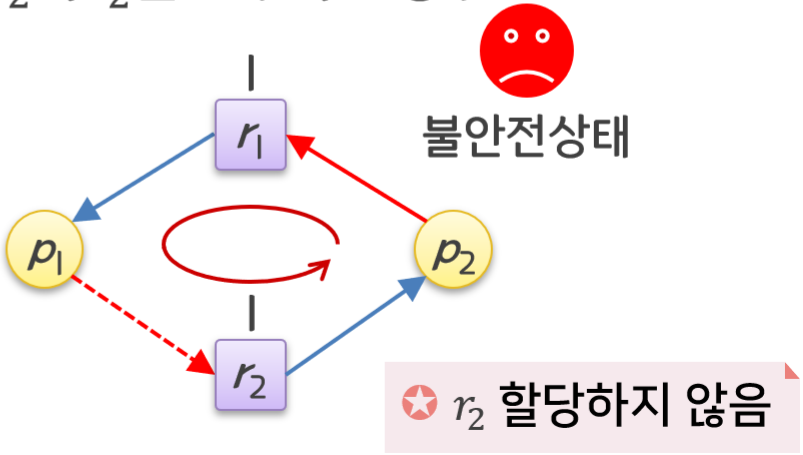
## 변형된 자원할당 그래프

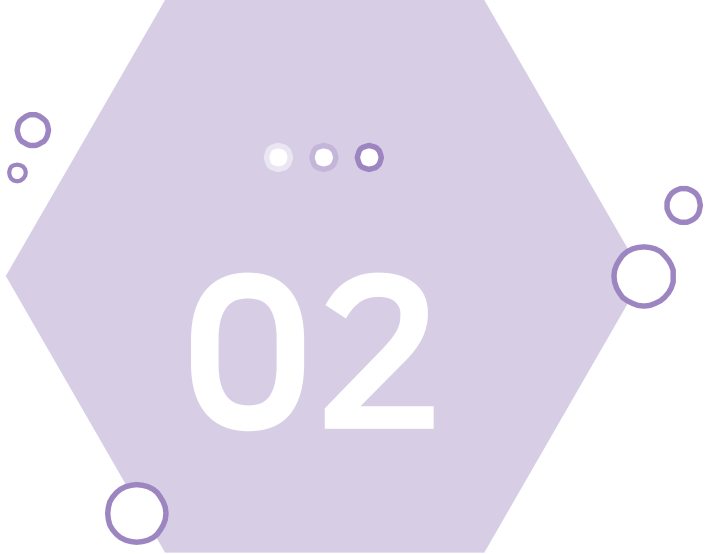
- 자원을 요청받으면 그 요구간선을 할당간선으로 변환하여도 사이클이 발생되지 않는 경우에만 자원을 할당

- $p_1$이 $r_2$를 요구하는 경우




- $p_2$가 $r_2$를 요구하는 경우






# 교착상태 탐지 및 복구

# 교착상태 탐지 및 복구



## 교착상태 탐지

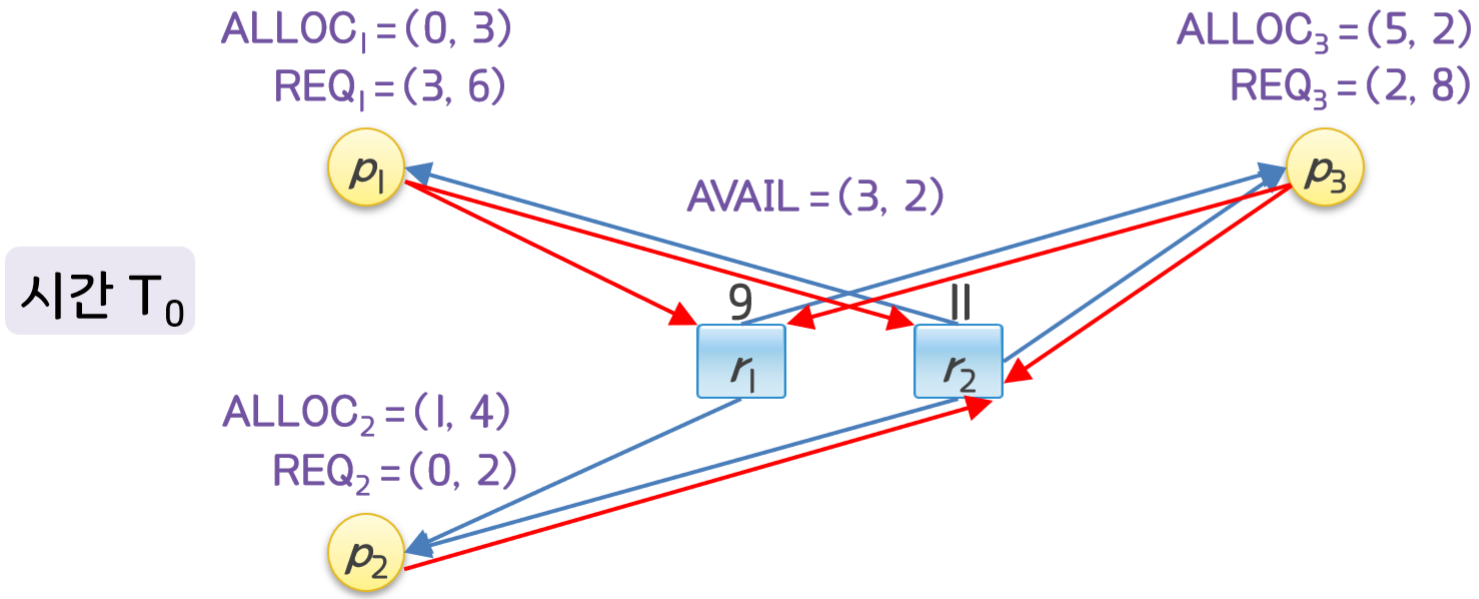
- 시스템의 교착상태 여부를 탐지하기 위해 주기적으로 상태 조사 알고리즘을 수행

## 교착상태 복구

- 교착상태가 탐지된 경우 복구조치에 들어감

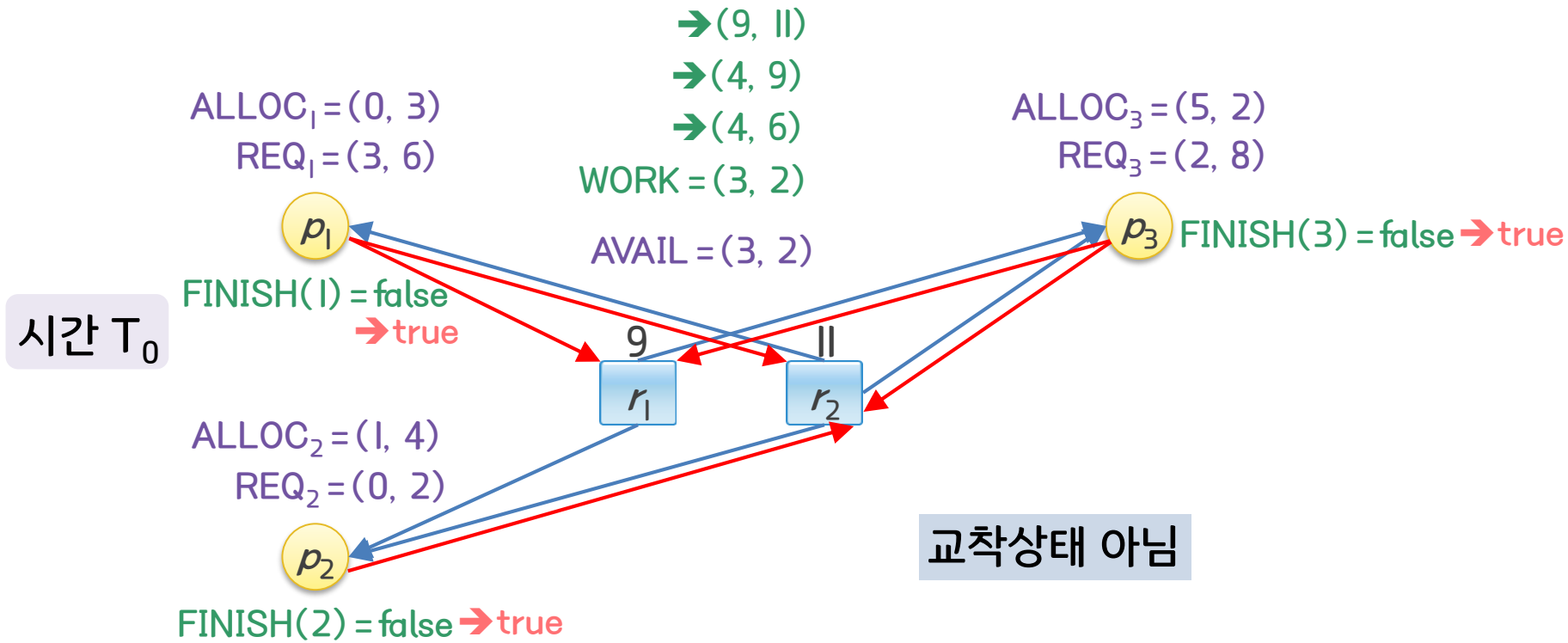
# 교착상태 탐지

- Shoshani와 Coffman 알고리즘



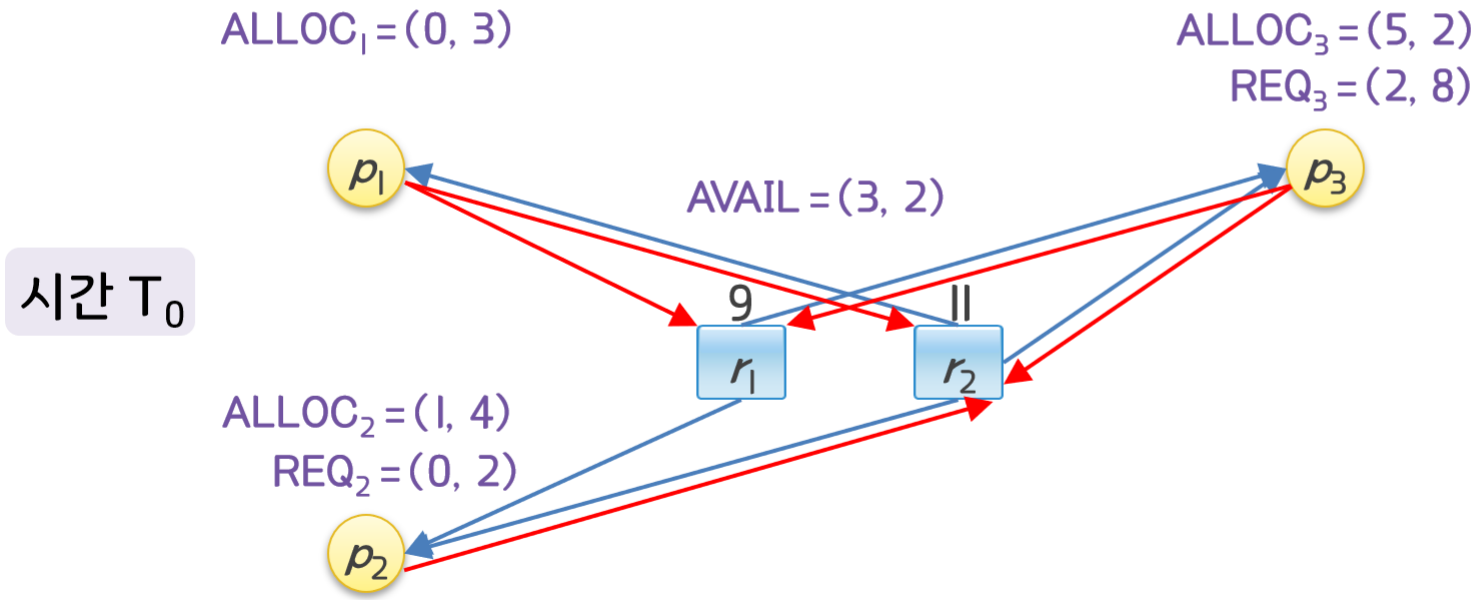
# 교착상태 탐지

- Shoshani와 Coffman 알고리즘



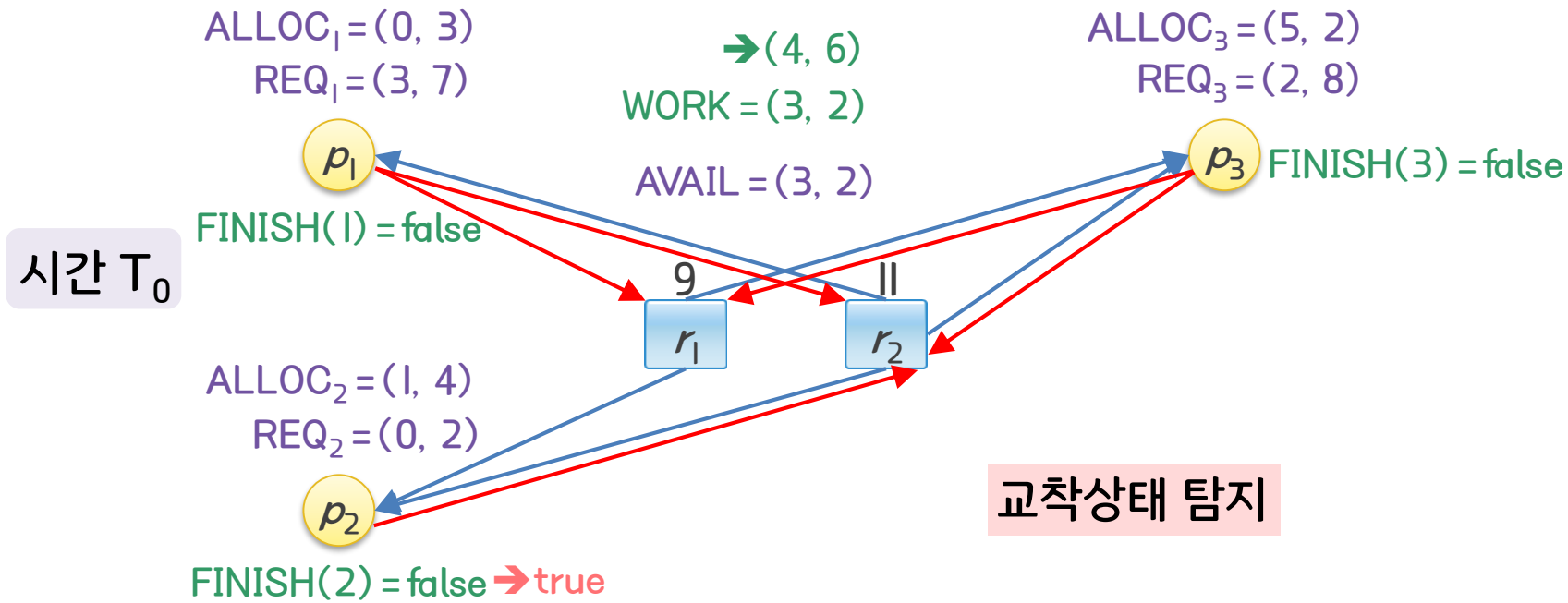
# 교착상태 탐지

- Shoshani와 Coffman 알고리즘



# 교착상태 탐지

- Shoshani와 Coffman 알고리즘



# 교착상태 탐지

## Shoshani와 Coffman 알고리즘

✪ 시간 복잡도: $O(mn^2)$

① 길이가 각각 $m$, $n$인 WORK와 FINISH 초기화

WORK ← AVAIL

$ALLOC_i \neq 0$이면 FINISH($i$) ← `false`

그렇지 않으면 FINISH($i$) ← `true`, $i$ = 1, 2, ⋯, $n$

② FINISH($i$) = `false`이고 $REQ_i \leq$ WORK인 $i$ 찾기

그런 $i$가 없으면 go to ④

③ WORK ← WORK + $ALLOC_i$

FINISH($i$) ← `true`

go to ②

④ 어떤 $i$에 대하여 FINISH($i$) = `false`이면 교착상태

- FINISH($i$) = `false`인 $p_i$는 교착상태임
- 알고리즘 수행 시점
  - » 즉시 받아들일 수 없는 할당요구가 있을 때
  - » 정해진 시간간격 또는 CPU 효율이 일정 수준 이하로 떨어질 때

# 교착상태 복구

## 복구의 주체

- 오퍼레이터: 교착상태 발생을 알려주면 수작업으로 복구
- 시스템: 자동적으로 복구

## 복구의 방법

- 교착상태 프로세스를 종료
- 교착상태 프로세스로부터 자원을 회수

# 교착상태 복구



## 프로세스 종료

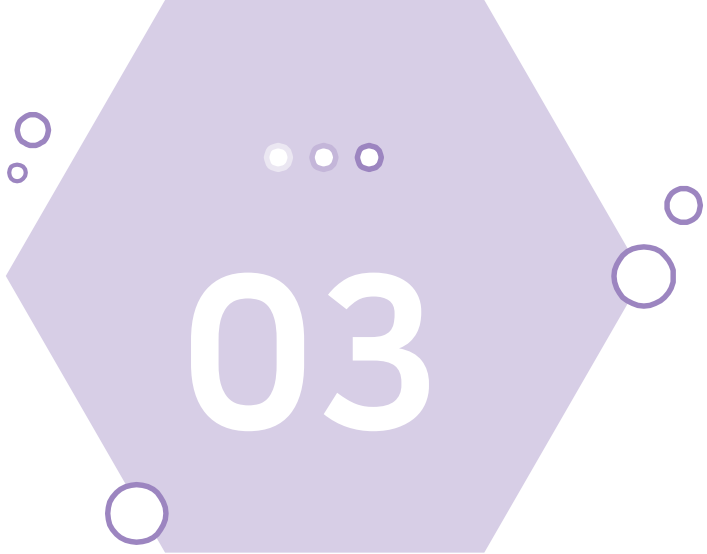
- 모든 교착상태 프로세스를 종료
  - ↪ 단점: 그동안 진행했던 내용들에 대한 복원 비용이 큼

- 사이클이 제거될 때까지 프로세스를 하나씩 종료
  - ↪ 단점: 종료 대상을 선택하기위한 비용, 매번 교착상태 재확인을 위한 비용

# 교착상태 복구



- 자원 회수
  - 사이클이 제거될 때까지 자원을 단계적으로 선점하여 다른 프로세스들에 할당
  - 고려사항: 희생자 선택, 복귀, 기아상태




# 복합적 접근방법

# 복합적 접근방법



- 방지, 회피, 탐지 및 복구를 복합적으로 사용
  - 자원을 유형에 따라 계층적으로 분류
  - 각 계층에 대하여 자원순서를 부여
  - 각 계층별로 방지, 회피, 탐지 및 복구 중 적절한 방법을 적용

강의를 마쳤습니다.

다음시간에는

# 8강. 메모리 관리