데이터베이스 및 설계

Chap 10. 데이터베이스의 저장과 접근

2012.06.07.

오 병 우

컴퓨터공학과

kit 국립금오공과대학교
Kumoh National Institute of Technology

데이타베이스의 저장(1)

- 데이타베이스 저장장치
  - 데이타를 저장하는 방법과 접근에 영향
  - 직접 접근 저장 장치(DASD:Direct Access Storage Device)로 디스크 사용
- 디스크 접근시간(access time)
  - 헤드가 원하는 트랙에 있는 레코드를 찾아 전송하는데 걸리는 시간
  - 탐구 시간(seek time): 헤드가 트랙까지 이동하는 데 걸리는 시간
  - 회전지연 시간(rotational latency): 섹터가 헤드 밑까지 오는 시간
  - 데이타 전송 시간(transfer time)
  - 메인메모리 접근시간에 비해 느림
    - 약 10~30ms cf) 메인 메모리 : 약 10~100ns
    - 디스크 접근 횟수의 최소화가 가장 중요한 성능 개선 방법
    - 디스크에의 배치, 저장이 중요한 문제

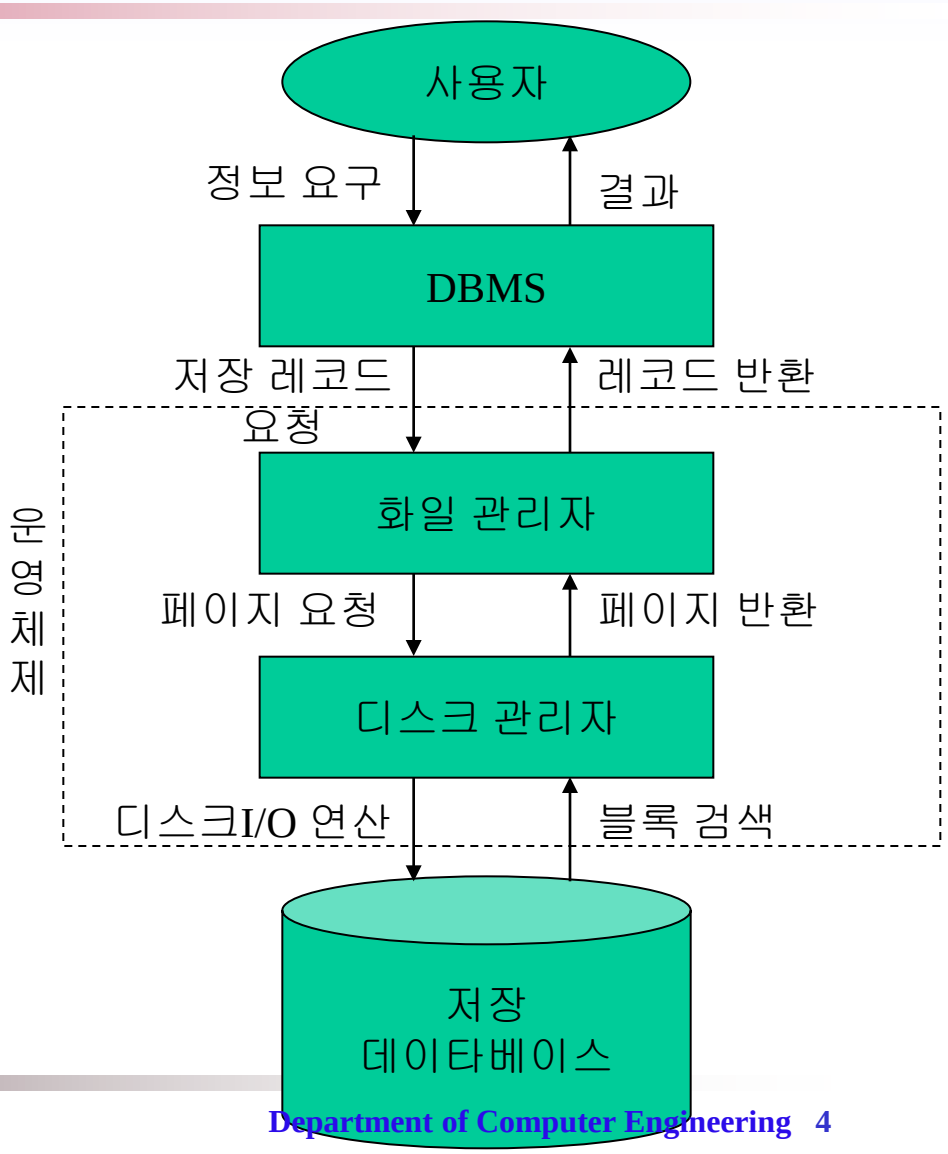
시험에 잘 나옴. 중요

데이타베이스의 저장(2)

- 저장구조(storage structure)
  - 디스크에 데이타가 배치되어 저장되어 있는 형태
  - 다양한 저장구조를 지원해야 좋은 시스템임
    - DB의 부분별로 적절한 저장
    - 성능요건 변경 시 저장 구조 변경
  - 데이타베이스의 물리적 설계
    - DB의 사용 방법, 응용, 응용 실행 빈도수에 따라 적절한 저장방식을 선정하는 과정

데이타베이스의 접근

- 데이타베이스의 일반적인 접근 과정
  - DBMS는 사용자가 요구하는 레코드를 결정
    - 화일 관리자에게 그 저장 레코드의 검색 요청
  - 화일 관리자는 그 저장레코드가 들어있는 페이지 결정
    - 디스크 관리자에게 그 페이지의 검색 요청
  - 디스크 관리자는 그 페이지의 물리적 위치 결정
    - 디스크에 입출력 명령을 내림



디스크 관리자(1)

- 기본 입출력 서비스 (basic I/O service) 모듈
  - 모든 물리적 I/O 연산에 대한 책임
  - 물리적 디스크 주소를 알고 있어야 함
  - 운영체제의 한 구성요소
- 화일 관리자 지원
  - 화일 관리자는 디스크를 일정 크기의 페이지로 구성된 페이지 세트들의 논리적 집단으로 취급
  - 데이타 페이지 세트와 하나의 자유공간 페이지 세트
  - 페이지 세트 : 유일한 페이지 세트 ID
  - 페이지 : 해당 디스크 내에서 유일한 페이지 번호를 가짐
- 디스크 관리
  - 페이지 번호 ← (사상) → 물리적 디스크 주소
    → 화일 관리자를 장비에서 독립
  - 화일 관리자의 요청에 따라 페이지 세트에 대한 페이지의 할당과 회수

디스크 관리자(2)

- 디스크 관리자의 페이지 관리 연산
  - 화일 관리자가 명령할 수 있는 연산
    1) 페이지 세트 S 로부터 페이지 P의 검색
    2) 페이지 세트 S 내에서 페이지 P 의 교체
    3) 페이지 세트 S 에 새로운 페이지 P 의 첨가
    (자유공간 페이지 세트의 빈 페이지를 할당)
    4) 페이지 세트 S 에서 페이지 P 를 제거
    (자유공간 페이지 세트로 반환)

- Notes
  - 화일관리자의 요청에 의한 연산 : 1), 2)
  - 디스크 관리자의 필요에 따른 연산 : 3), 4)

화일 관리자(1)

- DBMS가 디스크를 저장 화일들의 집합으로 취급할 수 있도록 지원
- 저장 화일(stored file)
  - 한 타입의 저장레코드 어커런스들의 집합
  - 한 페이지 세트는 하나 이상의 저장 화일을 포함
  - 화일 이름 또는 화일 ID로 식별
- 저장 레코드
  - 레코드번호 또는 레코드 ID(RID: Record Identifier)로 식별
  - 전체 디스크 내에서 유일
  - (페이지 번호, 오프셋(슬롯번호))
- OS의 한 구성요소 또는 DBMS와 함께 패키지화

화일 관리자(2)

- 화일 관리자의 화일 관리 연산
  - DBMS가 화일관리자에 명령할 수 있는 연산
    1) 저장 화일 f에서 저장 레코드 r의 검색
    2) 저장 화일 f에 있는 저장 레코드 r의 대체
    3) 저장 화일 f에 새로운 레코드를 첨가하고 새로운 레코드 RID, r을 반환
    4) 저장 화일 f에서 저장 레코드 r의 제거
    5) 새로운 저장 화일 f의 생성
    6) 저장 화일 f의 제거

페이지 세트와 화일

- 페이지 관리(page management)
  - 화일관리자가 물리적 디스크 I/O가 아닌 논리적인 페이지 I/O 로 관리할 수 있게끔 지원하는 디스크 관리자의 기능

- 예제 : 대학 데이타베이스
  - 레코드들의 논리적 순서는 그림에 있는 것과 같이 각각 학번, 과목번호, 학번-과목번호 순임
  - 저장 순서도 이 논리적 순서와 같음
  - 저장 화일들은 28개의 페이지로 구성된 페이지 세트에 저장
  - 각 레코드들은 하나의 페이지를 차지

대학 데이타베이스

	학번	이름	학년	학과
S1:	100	나수영	4	컴퓨터
S2:	200	이찬수	3	전기
S3:	300	정기태	1	컴퓨터
S4:	400	송병길	4	컴퓨터
S5:	500	박종화	2	산공

학생

	과목번호	과목이름	학점	담당교수
C1:	C123	프로그래밍	3	김성국
C2:	C312	자료 구조	3	황수관
C3:	C324	화일 구조	3	이규찬
C4:	C413	데이타베이스	3	이일로
C5:	E412	반 도 체	3	홍봉진

과목

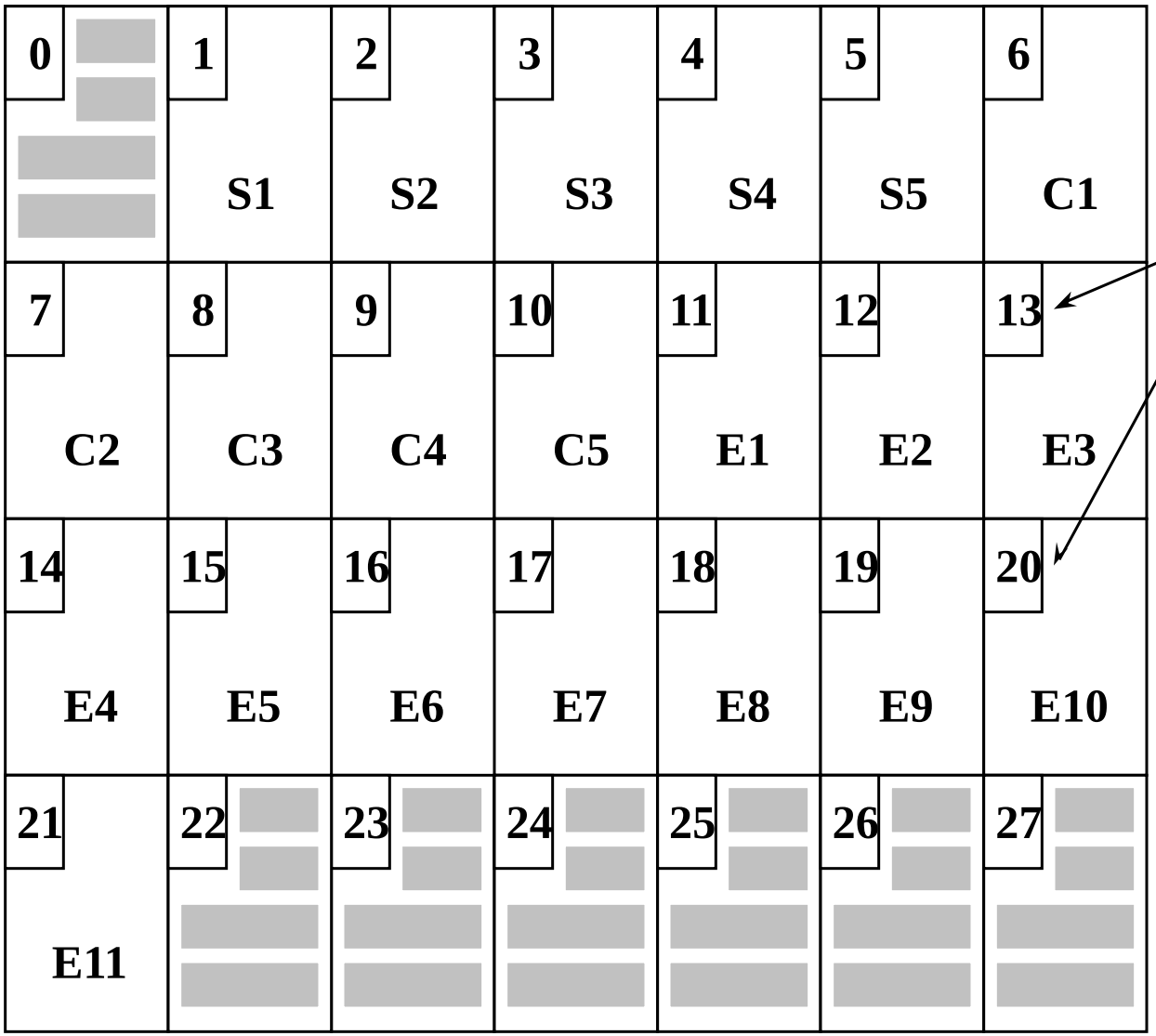
	학번	과목번호	성적
E1:	100	C413	A
E2:	100	E412	A
E3:	200	C123	B
E4:	300	C312	A
E5:	300	C324	C
E6:	300	C413	A
E7:	400	C312	A
E8:	400	C324	A
E9:	400	C413	B
E10:	400	E412	C
E11:	500	C312	B

등록

예제의 연산 과정 (1)

- 처음(빈 디스크) :
  - 하나의 자유 공간 페이지 세트만 존재(1 ~ 27)
  - 페이지 0 제외 : 디스크 디렉터리
- 파일 관리자 : 학생 화일에 있는 5개의 레코드 삽입
  - 디스크관리자 : 자유공간 페이지 세트의 페이지 1에서 5까지를 " 학생 페이지 세트" 라고 이름을 붙이고 할당
- 과목과 등록 화일에 대한 페이지 세트를 할당
  - 4개의 페이지 세트가 만들어짐
  - "학생"(1~5), "과목"(6~10), "등록"(11~21),"자유공간" 페이지 세트 (페이지 22~27)

대학 데이타베이스의 초기 적재 후의 디스크 배치도

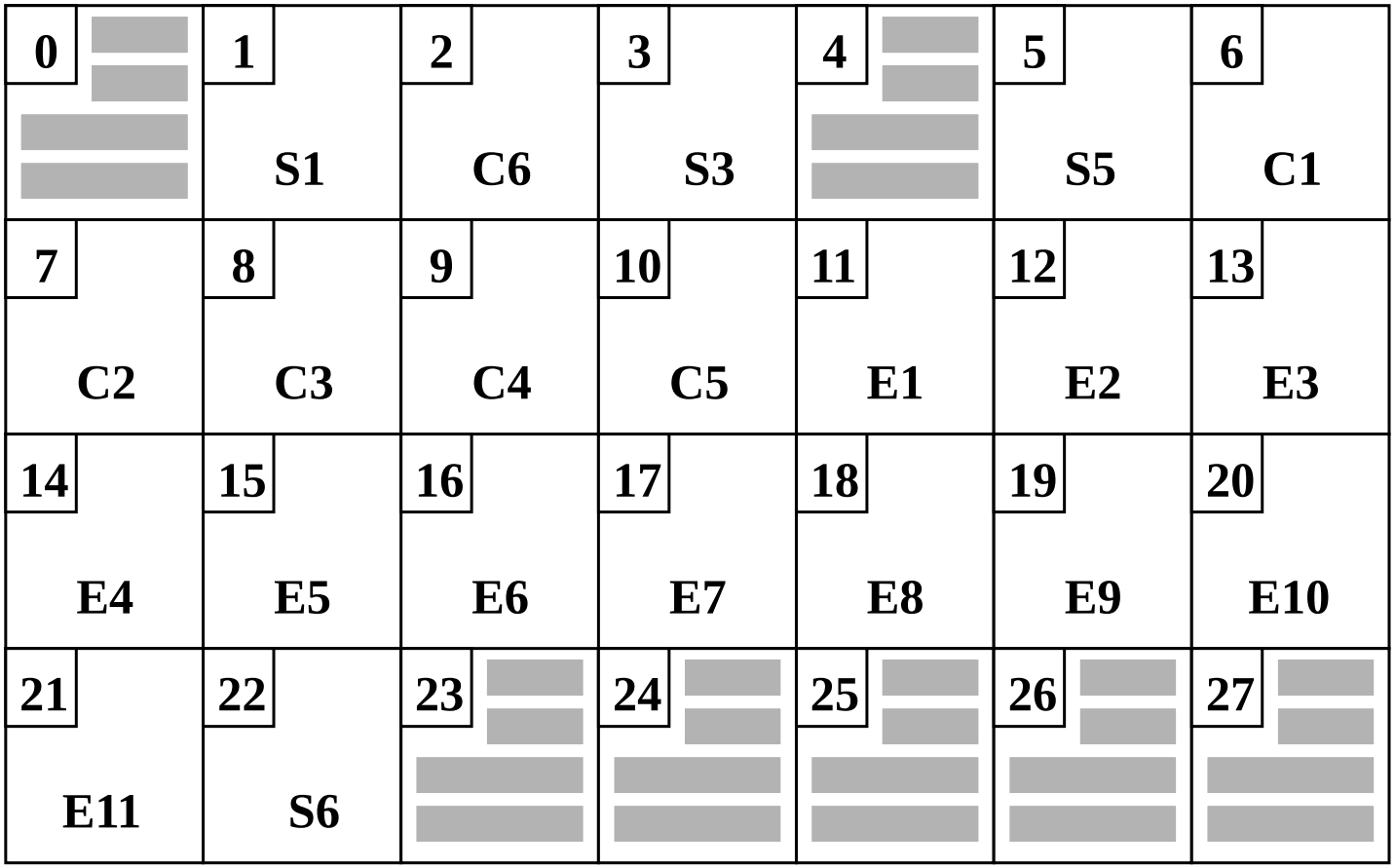


예제의 연산 과정 (2)

- 화일 관리자 : 새로운 학생 S6 (학번 600)을 삽입
  - 디스크 관리자 : 첫번째 자유 페이지 (페이지 22)를 자유공간 페이지 세트에서 찾아서 학생 페이지 세트에 첨가
- 화일 관리자 : S2 (학번 200)를 삭제
  - 디스크 관리자 : 이 레코드가 저장되어 있던 페이지 (페이지 2)를 자유공간 페이지 세트로 반납
- 화일 관리자 : 새로운 과목 C6 (E 515)를 삽입
  - 디스크 관리자 : 자유공간 페이지 세트에서 첫번째 자유페이지 (페이지 2)를 찾아서 과목 페이지 세트에 첨가
- 화일 관리자 : S4를 삭제
  - 디스크 관리자 : S4가 저장되어 있던 페이지 (페이지 4)를 자유공간 페이지 세트에 반납

삽입, 삭제 연산 뒤의 디스크 배치도

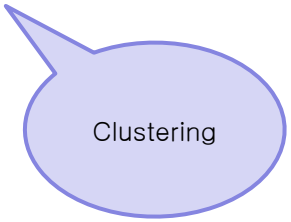
I : S6
D : S2
I : C6
D : S4



◈ 삽입, 삭제 연산 실행 후에는 페이지들의 물리적 인접성이 없어짐

포인터 표현 방법

- 한 페이지 세트에서 페이지의 논리적 순서가 물리적 인접으로 표현되지 않음
  - 페이지
    : 페이지 헤드 - 제어정보 저장
  - 포인터
    : 논리적 순서에 따른 다음 페이지의 물리적 주소
  - 다음 페이지 포인터는 디스크 관리자가 관리
    (화일 관리자는 무관)



- 페이지 헤드에 “다음 페이지” 포인터가 포함되어 있는 경우의 디스크 배치도

포인터 표현 방법(2)

- 디스크 디렉터리(페이지 세트 디렉터리)
  - 실린더 0, 트랙 0에 위치
  - 디스크에 있는 모든 페이지 세트의 리스트와 각 페이지 세트의 첫번째 페이지에 대한 포인터 저장

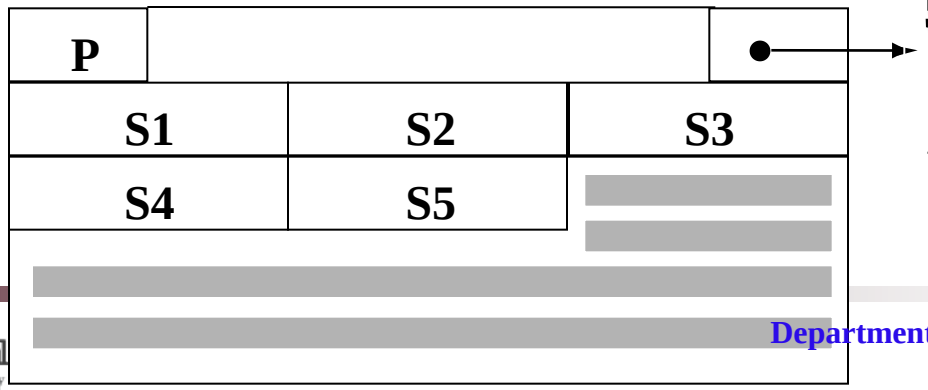
- 디스크 디렉터리 (페이지 0)

화일 관리자의 기능

- 저장 레코드 관리 (stored record management) 기능
  - DBMS가 페이지 I/O 에 대한 세부적인 사항에 대해 알 필요 없이 저장화일과 저장 레코드만으로 동작하게 함

- 예
  - 하나의 페이지에 여러 개의 레코드 저장
  - 학생 레코드에 대한 논리적 순서는 학번 순

1) 페이지 p에 5개의 학생레코드(S1~ S5)가 삽입되어 있다고 가정



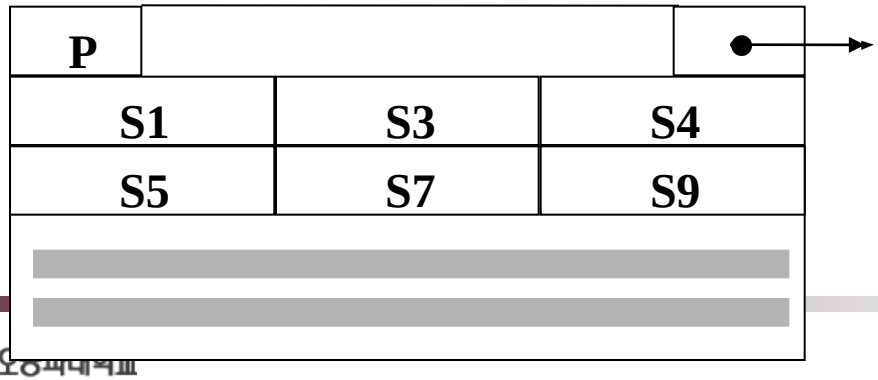
5개의 학생 레코드를 처음 적재한 페이지 P의 배치도

화일 관리자(2)

2) DBMS : 학생 레코드 S9(학번 900)의 삽입 요청
   - 페이지 p의 학생레코드 S5 바로 다음에 저장

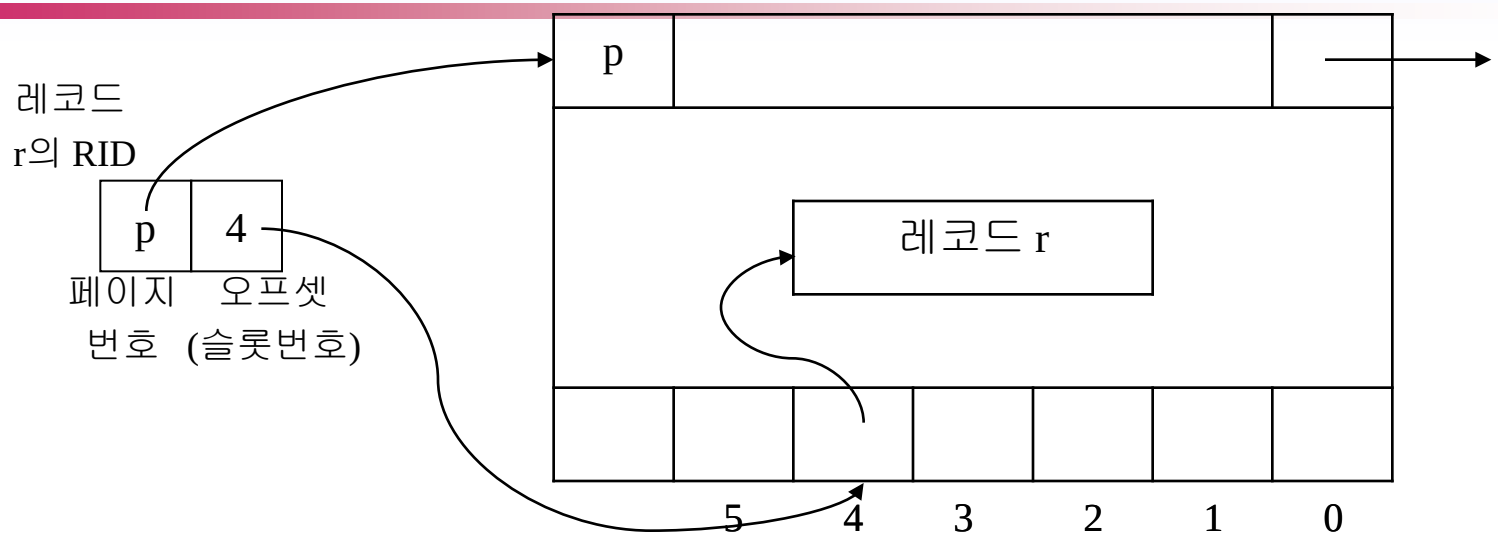
3) DBMS : 레코드 S2의 삭제 요청
   - 페이지 p에 있는 학생 레코드 S2를 삭제하고 뒤에 있는 레코드들을 모두 앞으로 당김

4) DBMS : 레코드 S7(학번 700)의 삽입 요청
   - 학생레코드 S5 다음에 들어가야 되므로 학생 레코드 S9를 뒤로 옮김



S2가 삭제되고 S9와 S7이 삽입된 후의 페이지 P 의 배치도

RID의 구현

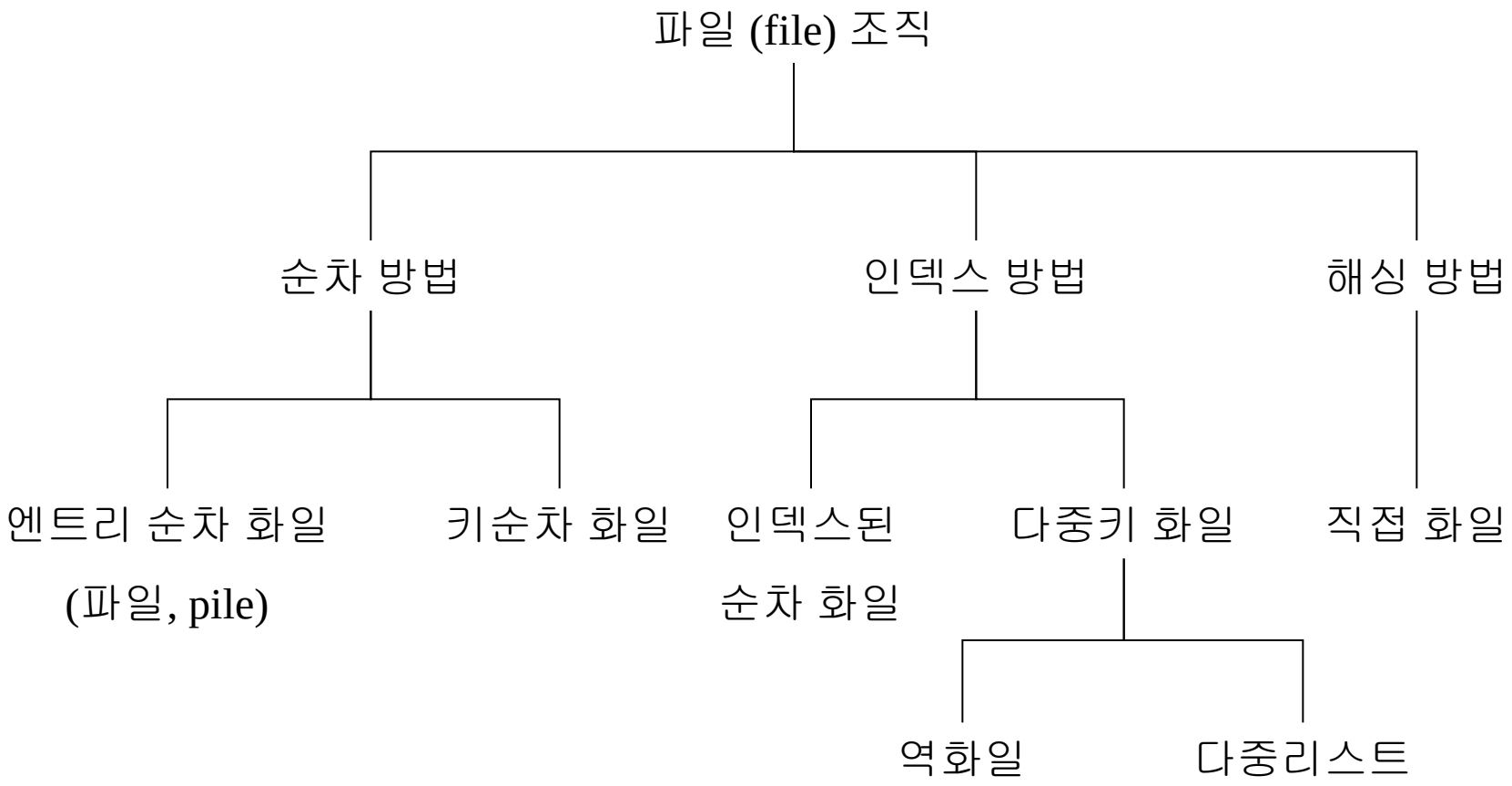


- RID = (페이지 번호 p, 오프셋(슬롯번호))
- 오프셋 = 페이지 내에서의 레코드 위치(byte)를 표현
- 레코드가 한 페이지 내에서 이동할 때마다 RID의 변경 없이 오프셋의 내용(포인터)만 변경
- 최악의 경우 두 번째 접근으로 원하는 레코드 검색가능
  - 해당 페이지가 오버플로가 되어 다른 페이지로 저장된 경우 두 번 접근

Note

- 화일에 속하는 모든 저장 레코드들을 순차적으로 접근 가능
  - 순차적 접근 : 페이지 세트내에서는 페이지 순서, 페이지 안에서는 레코드 순서
    - 레코드 순서 : RID의 오름차순(물리적 순차(physical sequence))
- 제어 정보
  - 저장 레코드에 저장된 정보 중, 시스템이 필요로 하는 정보
    - 화일의 ID
    - 레코드 길이(가변 길이 레코드의 경우)
    - 삭제 플래그(물리적 삭제를 하지 않는 경우)
  - 레코드 앞에 접두부(prefix)로 만들어 저장
  - 일반 사용자와 무관
- 사용자 데이타 필드
  - 저장 레코드에 저장된 정보 중, 사용자가 필요로 하는 정보
  - DBMS가 이용, 화일 관리자와 디스크 관리자는 알 필요 없음 (화일 관리자는 화일을 단순히 바이트 스트링으로 인식)

화일의 조직 방법


파일 (file) 조직
순차 방법
인덱스 방법
해싱 방법
엔트리 순차 화일
(파일, pile)
키순차 화일
인덱스된
순차 화일
다중키 화일
직접 화일
역화일
다중리스트

순차 방법

- 레코드들의 논리적 순서가 저장 순서와 동일
  - 힢(heap) 또는 파일(pile) : 엔트리 순차(entry-sequence) 화일
  - 일반적인 순차 화일 : 키 순차(key-sequence) 화일
- 레코드 접근 - 물리적 순서
- 화일 복사, 순차적 일괄 처리(batch processing) 응용

S4	S1	S2
S5	S3	

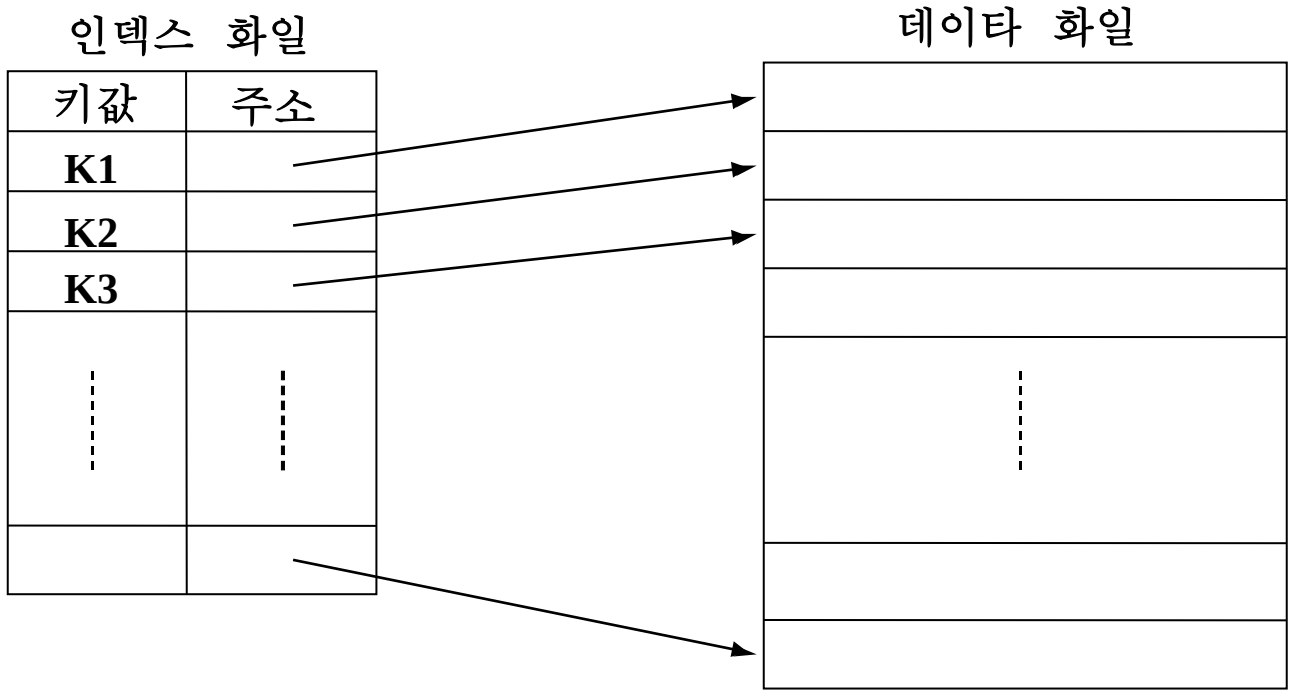
파일(엔트리순차) 화일

S1(100)	S2(200)	S3(300)
S4(400)	S5(500)	

키(학번)순차 화일

인덱스 방법

- 인덱스를 통해 데이타 레코드를 접근
- 인덱스된 화일(indexed file)의 구성
  - 인덱스 화일(index file)
  - 데이타 화일(data file)



인덱스된 순차 화일(indexed sequential file)

- 하나의 인덱스를 사용
- 순차 화일과 직접 화일을 결합한 형태

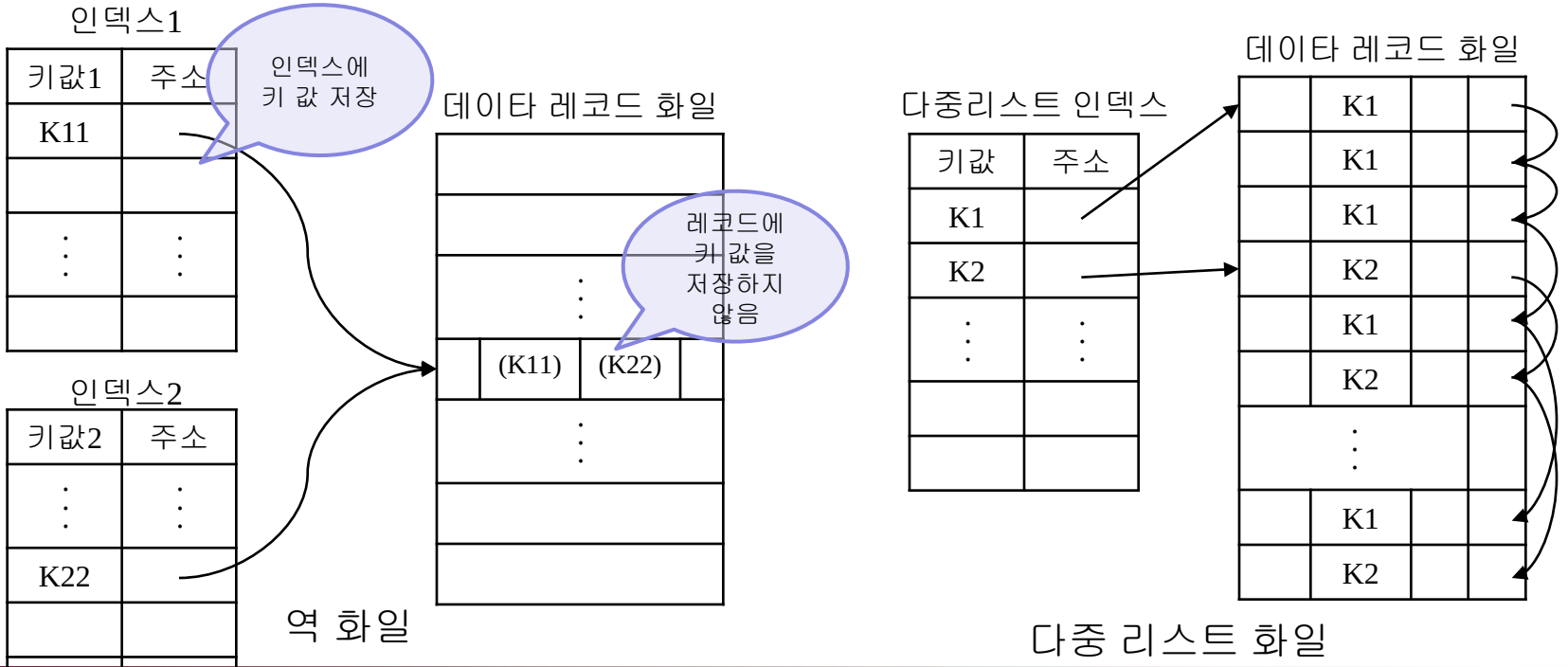
- 순차 데이타 화일(sequential data file)
  - 레코드가 순차적으로 정렬

    → 레코드 집합 전체에 대한 순차 접근 요구를 지원하는데 사용 (순차 접근 방법(sequential access method))
- 인덱스(index)
  - 레코드들에 대한 포인터를 가짐

    → 개별 레코드들에 대한 임의 접근 요구를 지원하는데 사용(직접 접근 방법(direct access method))
- ISAM (Indexed Sequential Access Method) File

다중 키 화일(multikey file)

- 데이타를 중복시키지 않으면서 여러 방법으로 데이타를 접근할 수 있는 다중 접근 경로를 제공
- 역 화일(inverted file)
  - 인덱스에 키 값과 레코드 주소를 저장하고, 레코드에서는 키 값을 저장하지 않음
- 다중 리스트 화일(multilist file)
  - 하나의 인덱스 값마다 하나의 데이타 레코드 리스트를 구축



인덱스의 구조 및 종류(1)

- 인덱스의 구조
  - <레코드 키 값, 레코드 주소(포인터)>
- 기본 인덱스(primary index)
  - 기본 키를 포함한 인덱스
- 보조 인덱스(secondary index)
  - 기본 인덱스 이외의 인덱스. 보통 보조키를 포함
- 집중 인덱스(clustered index)
  - 데이타 레코드의 물리적 순서가 인덱스 엔트리 순서와 동일하게 유지하도록 구성된 인덱스
  - 같은 인덱스 키 값을 가진 레코드는 물리적으로 인접하게 되어 검색이 효율적
- 비 집중 인덱스(unclustered index)
  - 집중 형태가 아닌 인덱스
  - 하나의 데이타 화일에 여러 개 생성 가능

인덱스의 구조 및 종류(2)

- 밀집 인덱스(dense index)
  - 데이타 레코드 하나에 대해 하나의 인덱스 엔트리가 만들어지는 인덱스
  - 역 인덱스(inverted index)는 보통 밀집 인덱스 형태로 만듬
- 희소 인덱스(sparse index)
  - 데이타 화일의 레코드 그룹 또는 데이타 블록에 하나의 엔트리가 만들어지는 인덱스

(1) B-트리

- 차수 m인 B-트리의 특성
  - 공백이거나 높이가 1이상인 m-원 탐색 트리(m-way search tree)
  - 루트와 리프(leaf)를 제외한 내부 노드는 최소 [m/2], 최대 m개의 서브트리를 가짐(적어도 [m/2]-1개의 키 값을 가짐)
  - 루트는 그 자체가 리프가 아닌 이상 적어도 두 개의 서브트리를 가짐
  - 모든 리프는 같은 레벨에 있음
    - 균형 트리

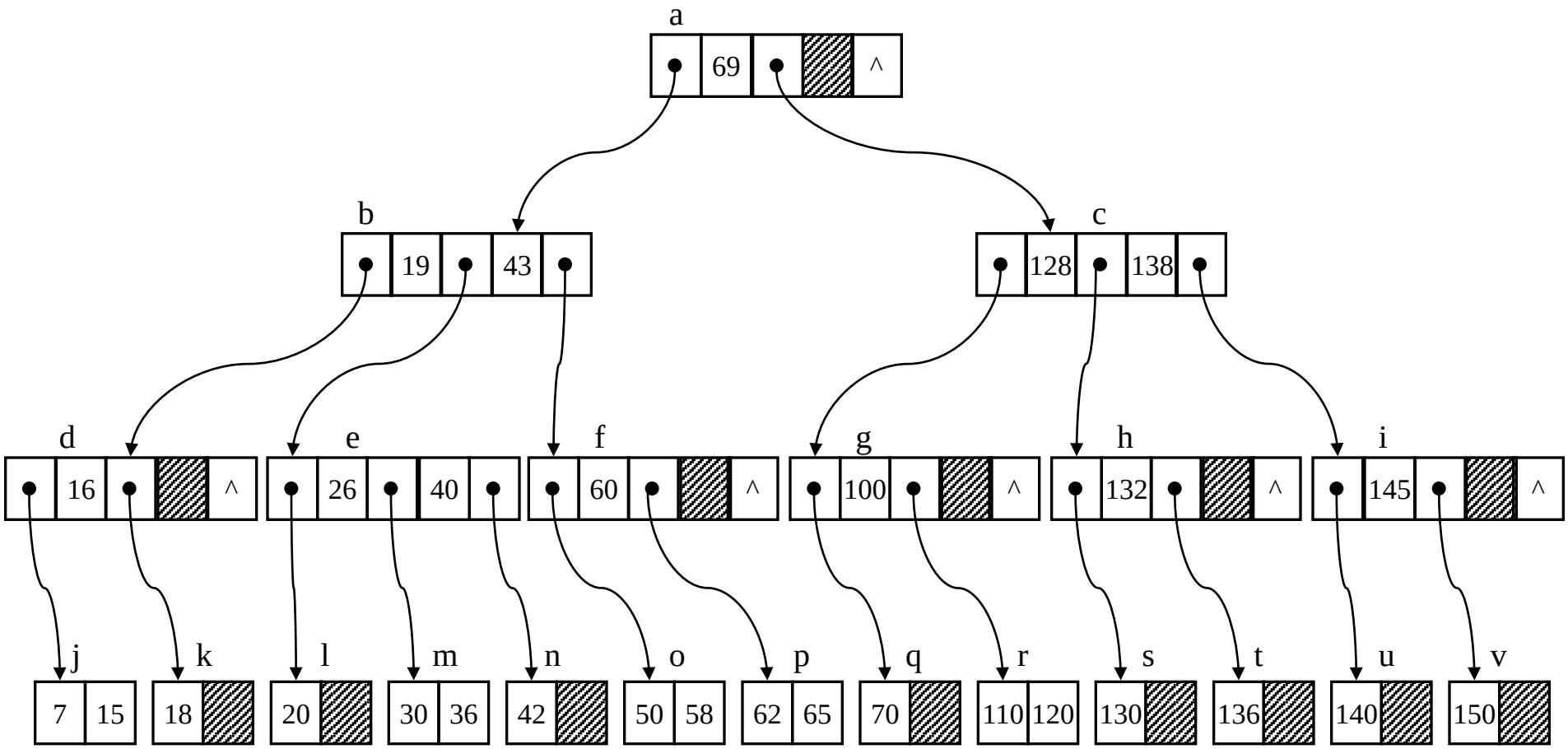
B-트리의 노드의 구조 및 특성

$< n, P_0, <K_1, A_1>, P_1, <K_2, A_2>, P_2, \ldots, P_{n-1}, <K_n, A_n>, P_n >$

- n : 키 값의 수 ($1 \le n < m$)
- $P_0, \ldots, P_n$: 서브트리에 대한 포인터
- $K_1, \ldots, K_n$: 키 값
  - 각 키 값(K_i)은 그 키 값을 가지고 있는 레코드에 대한 포인터 Ai를 함께 포함

- 한 노드 안에 있는 키 값들은 오름차순
- P_i ($0 \le i \le n-1$)가 지시하는 서브트리에 있는 모든 노드들의 모든 키 값들은 키 값 K_{i+1}보다 작음
- P_n이 지시하는 서브트리에 있는 노드들의 모든 키 값들은 K_n보다 큼
- P_i ($0 \le i \le n$)가 지시하는 서브트리들은 모두 m-원 서브 탐색 트리

3차 B-트리


a
69
^
b
19
43
c
128
138
d
16
^
e
26
40
f
60
^
g
100
^
h
132
^
i
145
^
j
7
15
k
18
l
20
m
30
36
n
42
o
50
58
p
62
65
q
70
r
110
120
s
130
t
136
u
140
v
150

연산(1)

- B-트리에서의 연산의 종류
  - 직접 탐색 : 키 값에 의한 분기
  - 순차 탐색 : 중위 순회
  - 삽입, 삭제 : 트리의 균형을 유지해야함
  - 분할 : 노드 오버플로 발생시
  - 합병 : 노드 언더플로 발생시
- 삽입
  - 리프노드
    - 빈 공간이 있는 경우 : 단순히 빈 공간에 삽입하면 됨
    - 오버플로

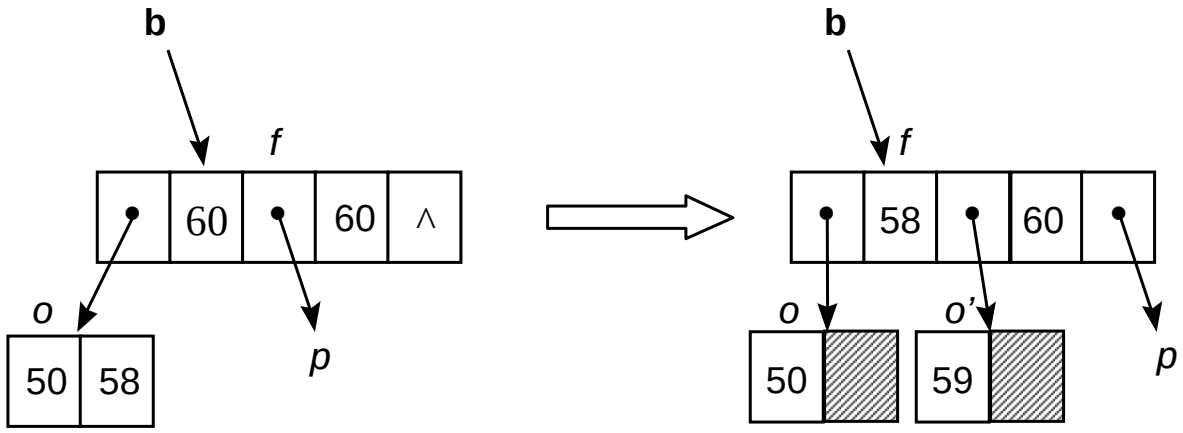
      1) 두 노드로 분열(split)

      2) $\lceil m/2 \rceil$ 째의 키 값 → 부모노드

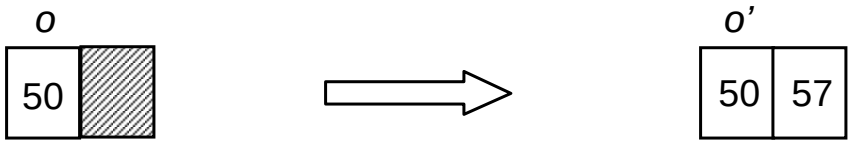
      3) 나머지는 반씩 나눔 (왼쪽, 오른쪽 서브트리)

삽입 예

- 59 삽입

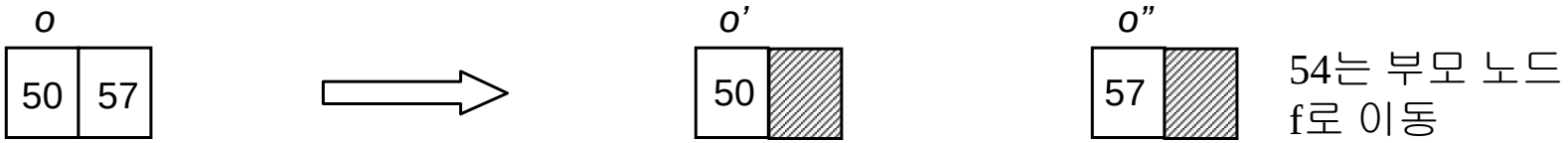


- 57 삽입

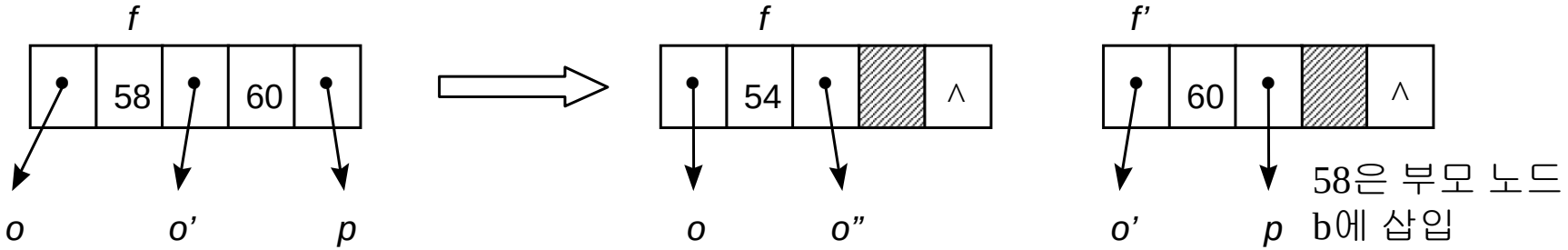


삽입 예

- 54의 삽입으로 노드 분할

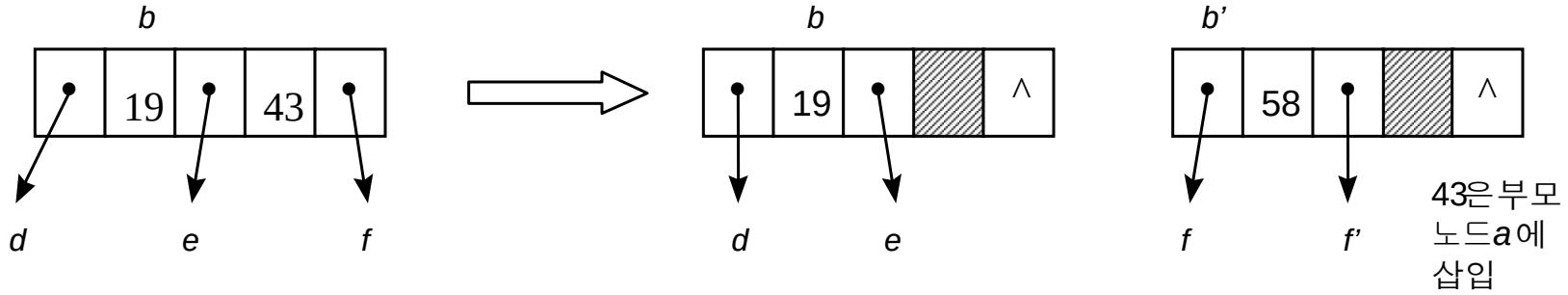


- 부모 노드 f에 54 삽입

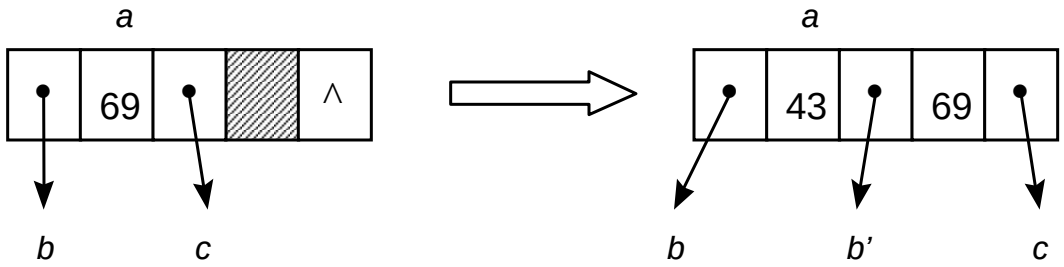


삽입 예

- 부모 노드 b에 58 삽입



- 부모 노드 a에 43 삽입

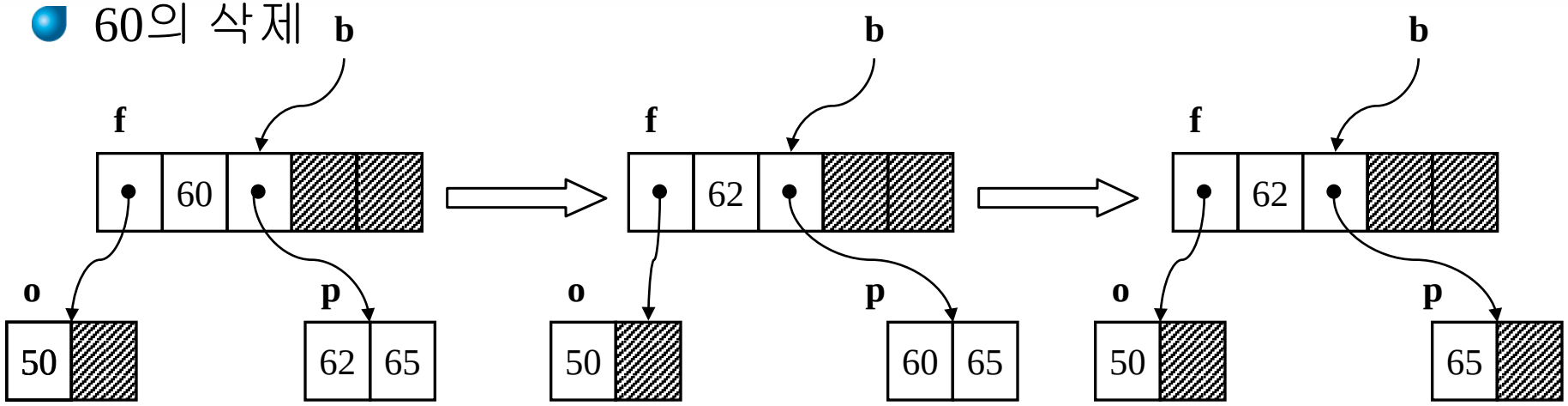


연산(2)

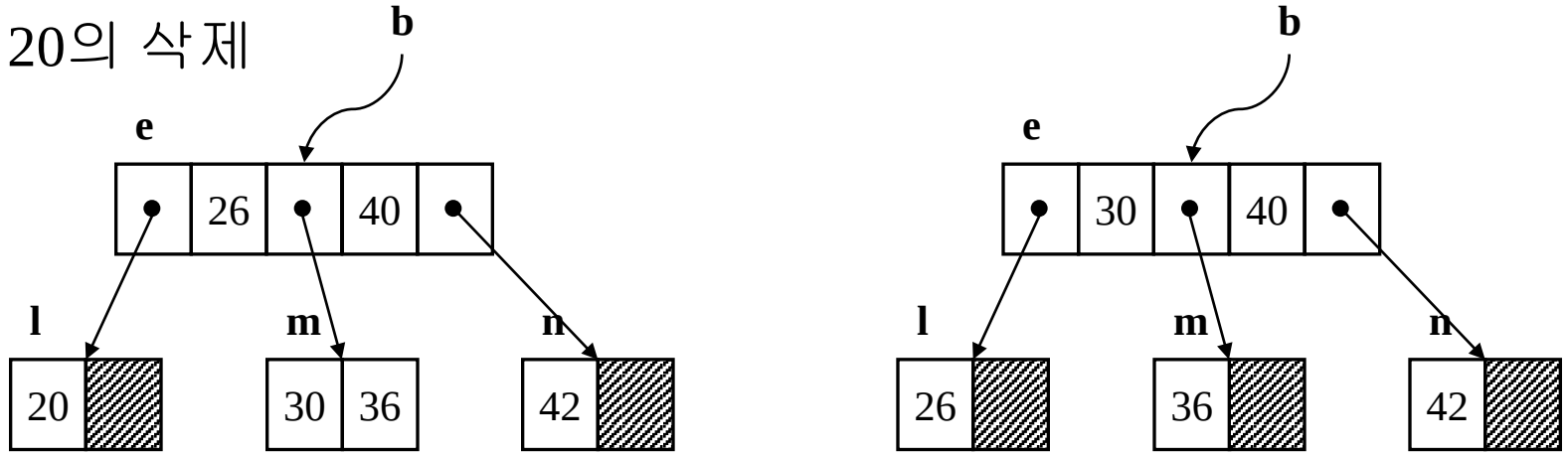
- 삭제
  - 리프노드
  - 삭제키가 리프가 아닌 노드에 존재
    - 후행키 값과 자리교환(후행키-항상 리프에)
    - 리프노드에서 삭제
  - 언더플로 : 키수 < [m/2] -1
    - 재분배 (redistribution)
      - 최소키 수 이상을 포함한 형제노드에서 이동
        (형제노드의 키 → 부모노드 → 언더플로 노드)
    - 합병 (merge)
      - 재분배 불가능시 이용
        (형제노드 + 부모노드의 키 + 언더플로 노드)

삭제 예

- 60의 삭제



- 20의 삭제

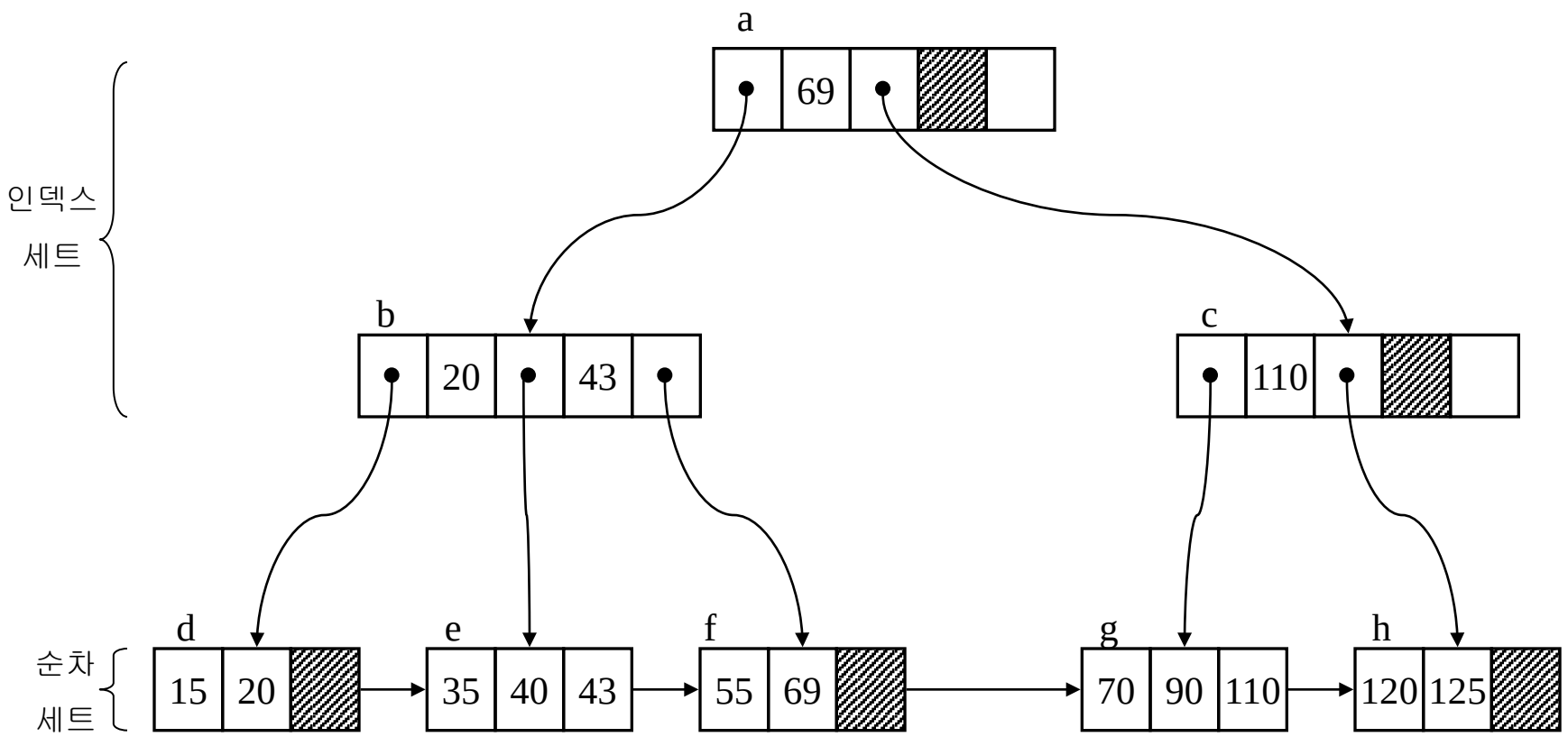


(2) B^+-트리

- 인덱스 세트 (index set)
  - 내부 노드
  - 리프에 있는 키들에 대한 경로 제공
  - 직접처리 지원

- 순차 세트 (sequence set)
  - 리프 노드
  - 모든 키 값들을 포함
  - 순차 세트는 순차적으로 연결
    → 순차처리 지원
  - 내부 노드와 다른 구조

차수가 3인 B+-트리


a
69
인덱스
세트
b
20
43
c
110
d
e
f
g
h
순차
세트
15
20
35
40
43
55
69
70
90
110
120
125

B^+-트리(2)

- 특성
  - 노드 구조

    $< n, P_0, K_1, P_1, K_2, P_2, \ldots, P_{n-1}, K_n, P_n >$

    - n : 키 값의 수 ($1 \le n < m$)
    - $P_0, \ldots, P_n$: 서브트리에 대한 포인터
    - $K_1, \ldots, K_n$: 키 값
  - 루트는 0 또는 2에서 m개 사이의 서브트리를 가짐
  - 루트와 리프를 제외한 모든 내부 노드는 최소 $\lceil m/2 \rceil$개, 최대 m개의 서브트리를 가짐
  - 리프가 아닌 노드에 있는 키 값의 수는 그 노드의 서브트리 수보다 하나 적음
  - 모든 리프 노드는 같은 레벨
  - 한 노드 안에 있는 키 값들은 오름차순

B^+-트리(3)

- Pi, $(0 \le i \le n-1)$가 지시하는 서브트리에 있는 노드들의 모든 키 값들은 키 값 K_{i+1}보다 작거나 같다.
- P_n이 지시하는 서브트리에 있는 노드들의 어떤 키 값도 키 값 K_n보다 크다.
- 리프 노드는 모두 링크로 연결되어 있다.
- 리프 노드의 구조

$$< q, <K_1, A_1>, <K_2, A_2>, \ldots , <K_q, A_q>, P_{next}>$$

  - A_i : 키 값 K_i를 가지고 있는 레코드에 대한 포인터
  - q : $\lceil m/2 \rceil \le q$
  - P_{next} : 다음 리프 노드에 대한 링크

B^+-트리(4)

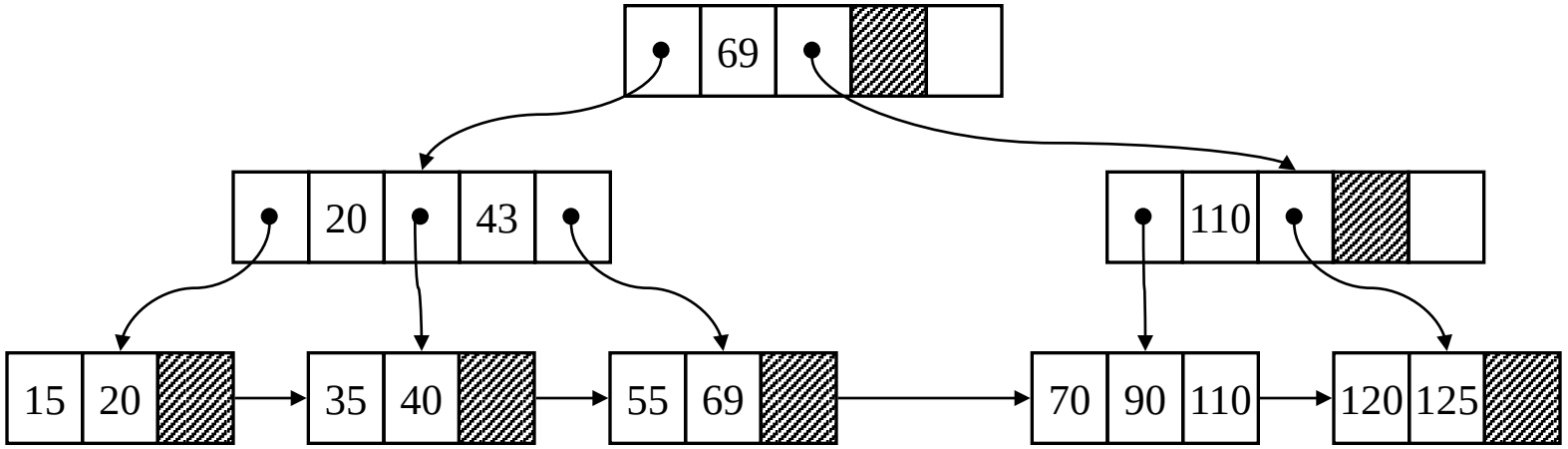
- B-트리와의 차이점
  - 인덱스 세트에 있는 키 값 : 리프 노드에 있는 키 값을 찾아가는 경로만 제공하기 위해서 사용
    - 인덱스 세트에 있는 키 값은 사실상 모두 순차 세트에 다시 나타남
  - 인덱스 세트의 노드와 순차 세트의 노드는 그 내부 구조가 서로 다름
    - 리프 노드 : 키 값과 이 키 값을 포인터가 함께 저장
    - 인덱스 세트에 있는 노드 : 키 값만 저장된다.
    - 노드에 저장할 수 있는 원소의 수도 서로 다름
  - 순차 세트의 모든 노드가 순차적으로 서로 연결된 연결 리스트
    - 순차 접근이 효율적

B^+-트리의 연산

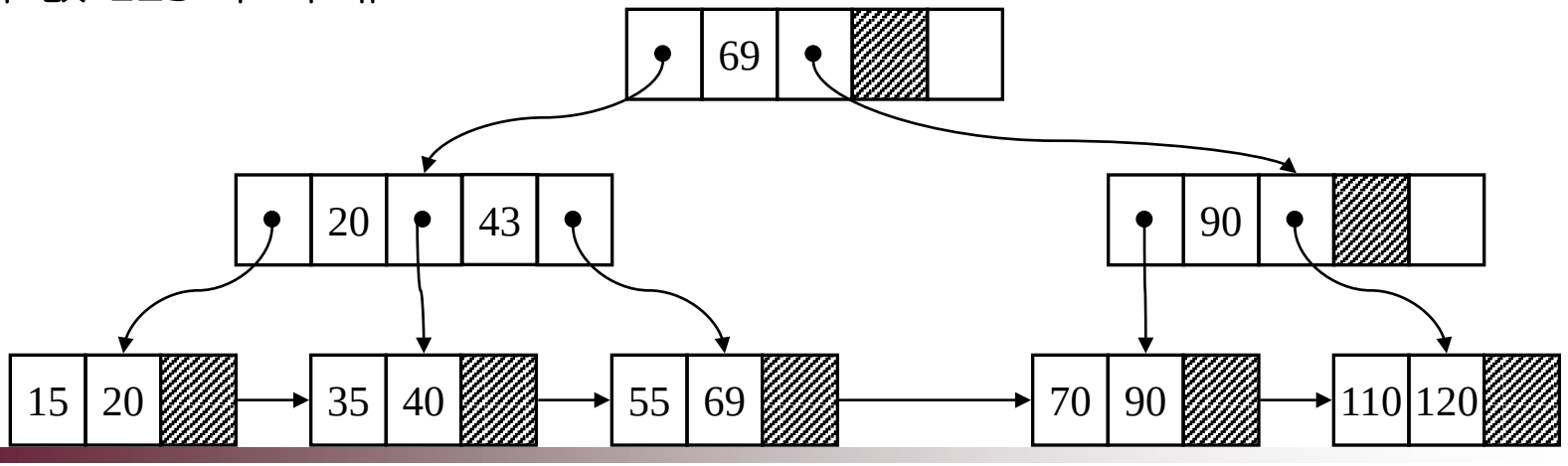
- 탐색
  - B^+-트리의 인덱스 세트 : m-원 탐색 트리와 같음
  - 리프에서 검색
- 삽입
  - B-트리와 유사
  - 리프의 오버플로우 (분할) → 부모 노드, 분할노드 모두에 키 값 존재
  - 순차 세트의 연결 리스트의 순차성 유지
- 삭제
  - 리프에서만 삭제 (재분배, 합병이 필요 없는 경우)
    - 분리자(separator)로 유지
      ∵ 다른 키 값을 탐색하는데 분리 키 값으로만 사용
  - 재분배: 인덱스 키 값 변화, 트리 구조 유지
  - 합병 : 인덱스의 키 값도 삭제

삭제 예

- B^+-트리에서 키 값 43을 삭제한 이후의 트리

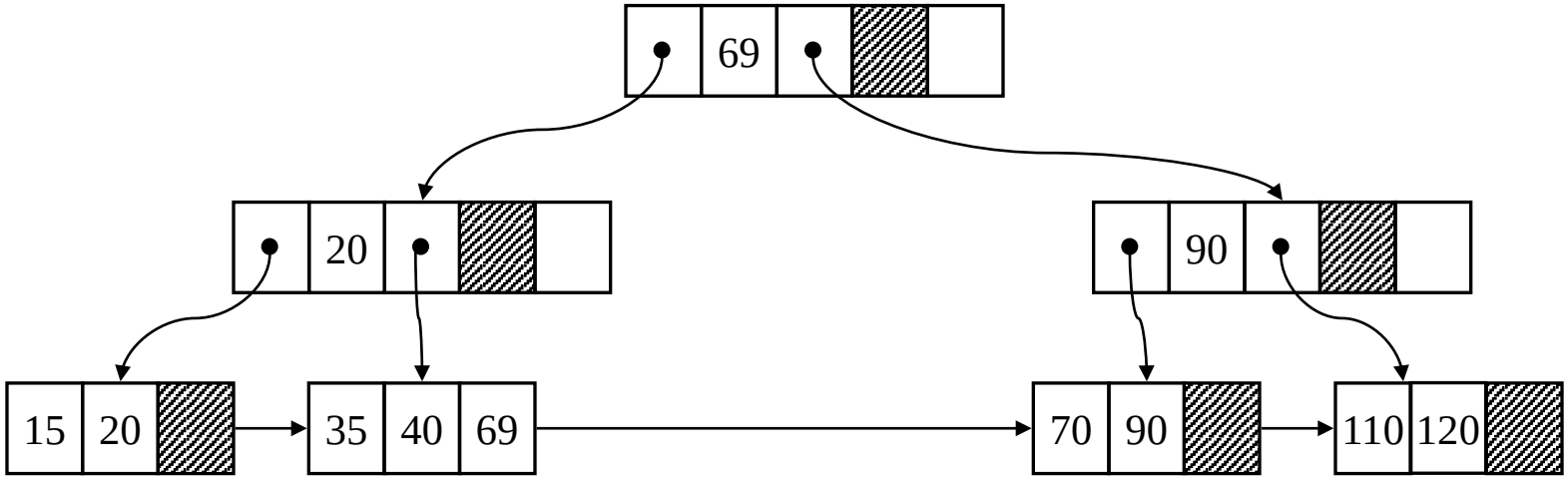


- 키 값 125의 삭제



삭제 예

- 키 값 55의 삭제

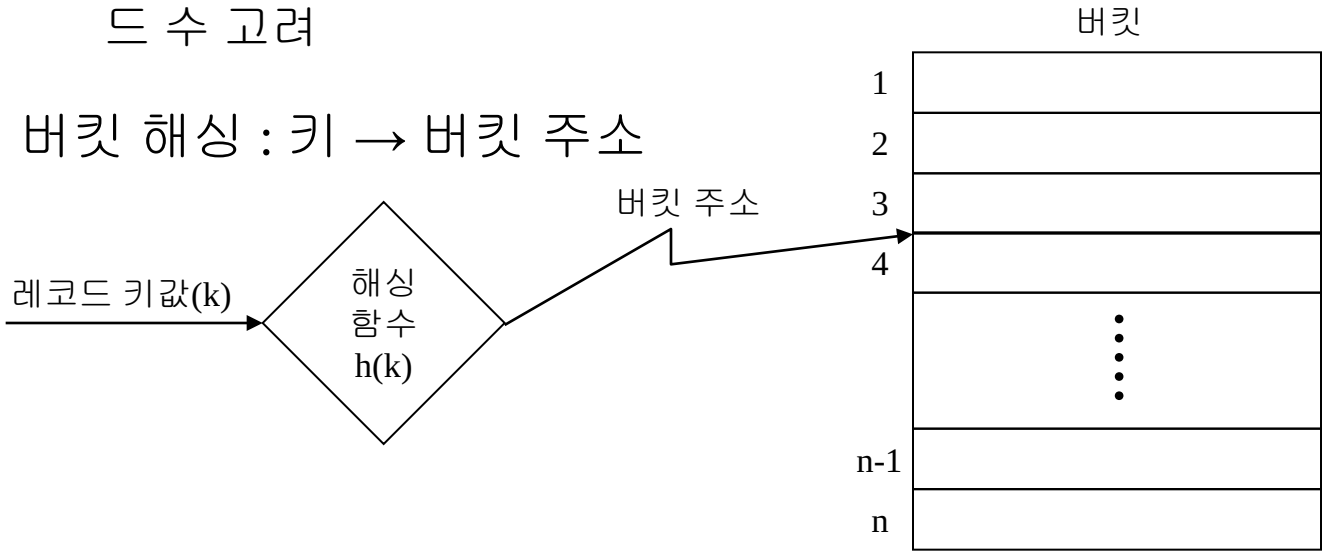


해싱 방법

- 다른 레코드 참조 없이 목표 레코드 직접 접근
  - 직접 화일(direct file)
- 키 값과 레코드 주소 사이의 관계 설정
- 해싱 함수(hashing function)
  - 키 값으로부터 주소를 계산
  - 사상 함수(mapping function) : 키 → 주소
  - 삽입, 검색에도 이용

(1) 버킷 해싱

- 버킷(bucket) : 하나의 주소를 가지면서 하나 이상의 레코드를 저장할 수 있는 화일의 한 구역
  - 버킷 크기 : 저장장치의 물리적 특성과 한번 접근으로 채취 가능한 레코드 수 고려
- 버킷 해싱 : 키 → 버킷 주소



- 충돌(collision) : 상이한 레코드들이 같은 주소(버킷)로 변환
  - 버킷 만원 - 오버플로우 버킷
  - 한번의 I/O가 추가됨

(2) 확장성 해싱(1)

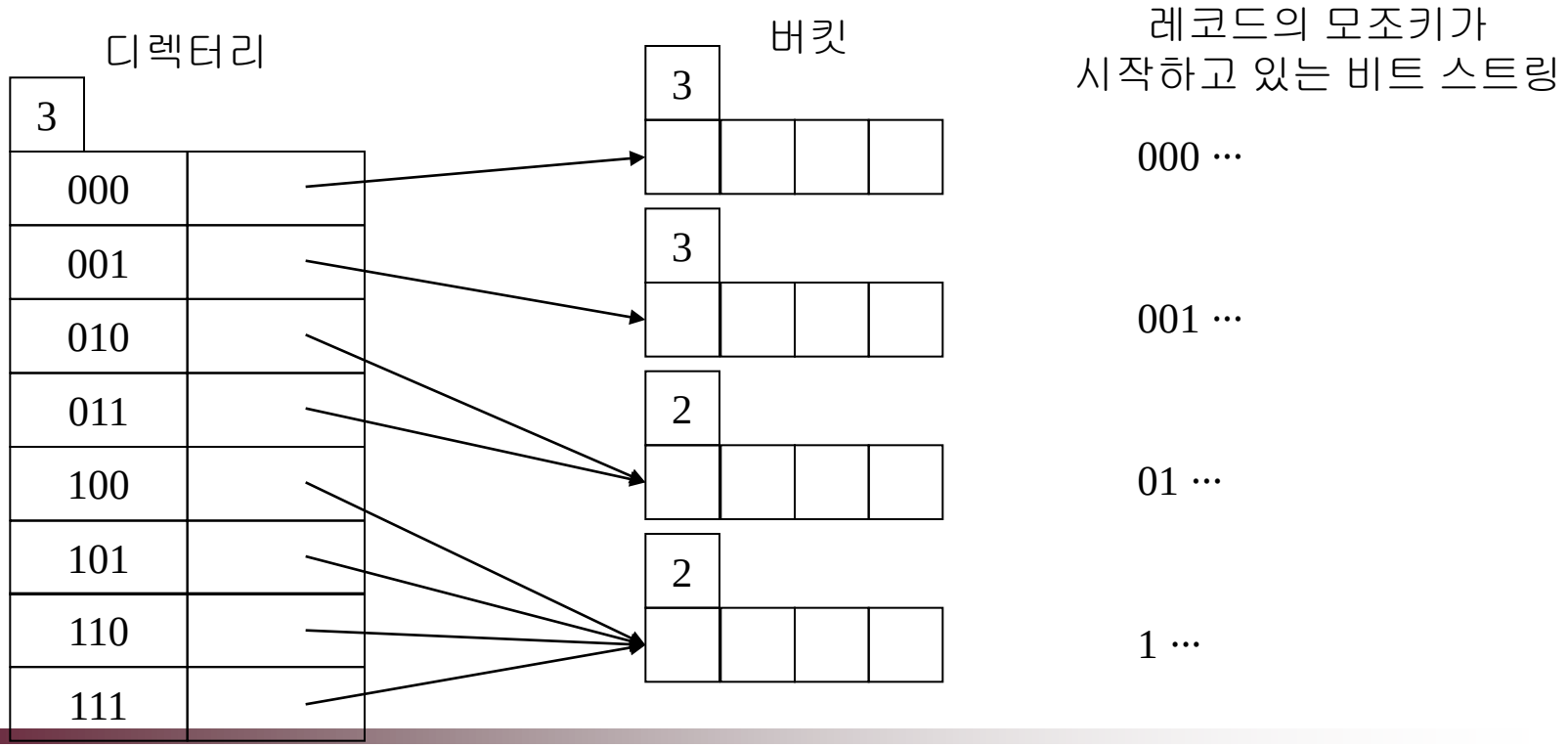
- 충돌 문제에 대처하기 위해 제안된 기법
- 특정 레코드 검색 - 1~2번의 디스크 접근

- 모조키(pseudokey)
  - 확장성 해싱 함수:
    키 값 → 일정 길이의 비트 스트링(모조키)
  - 모조키의 처음 d 비트를 인덱스로 사용

- 디렉터리
  - 헤더 : 정수값 d를 저장
    - d = 전역 깊이(global depth)
  - 버킷들을 지시하는 2^d 개의 포인터
  - 디스크에 저장

(2) 확장성 해싱(2)

- 버킷
  - 정수 값 p(≤ d)가 저장된 헤더 존재
    - p = 지역 깊이 (local depth)
      = 버킷에 저장된 레코드들의 모조키들이 처음부터 p비트까지 모두 동일



확장성 해싱의 연산 (1)

- 검색
  - 모조키의 처음 d비트를 디렉터리에 대한 인덱스로 사용
  - 접근된 디렉터리 엔트리는 목표 버킷에 대한 포인터를 제공

  - 검색 예
    1) 키 값 k → 모조키 101000010001
    2) 모조키의 처음 3비트(=d) 사용
       - 디렉터리의 6번째(101) 엔트리를 접근
    3) 엔트리는 4번째 버킷에 대한 포인터
       - 키 값 k를 가지고 있는 레코드가 저장되어있는 곳
       - p= 1 : 모조키 비트 1로 시작하는 레코드 저장

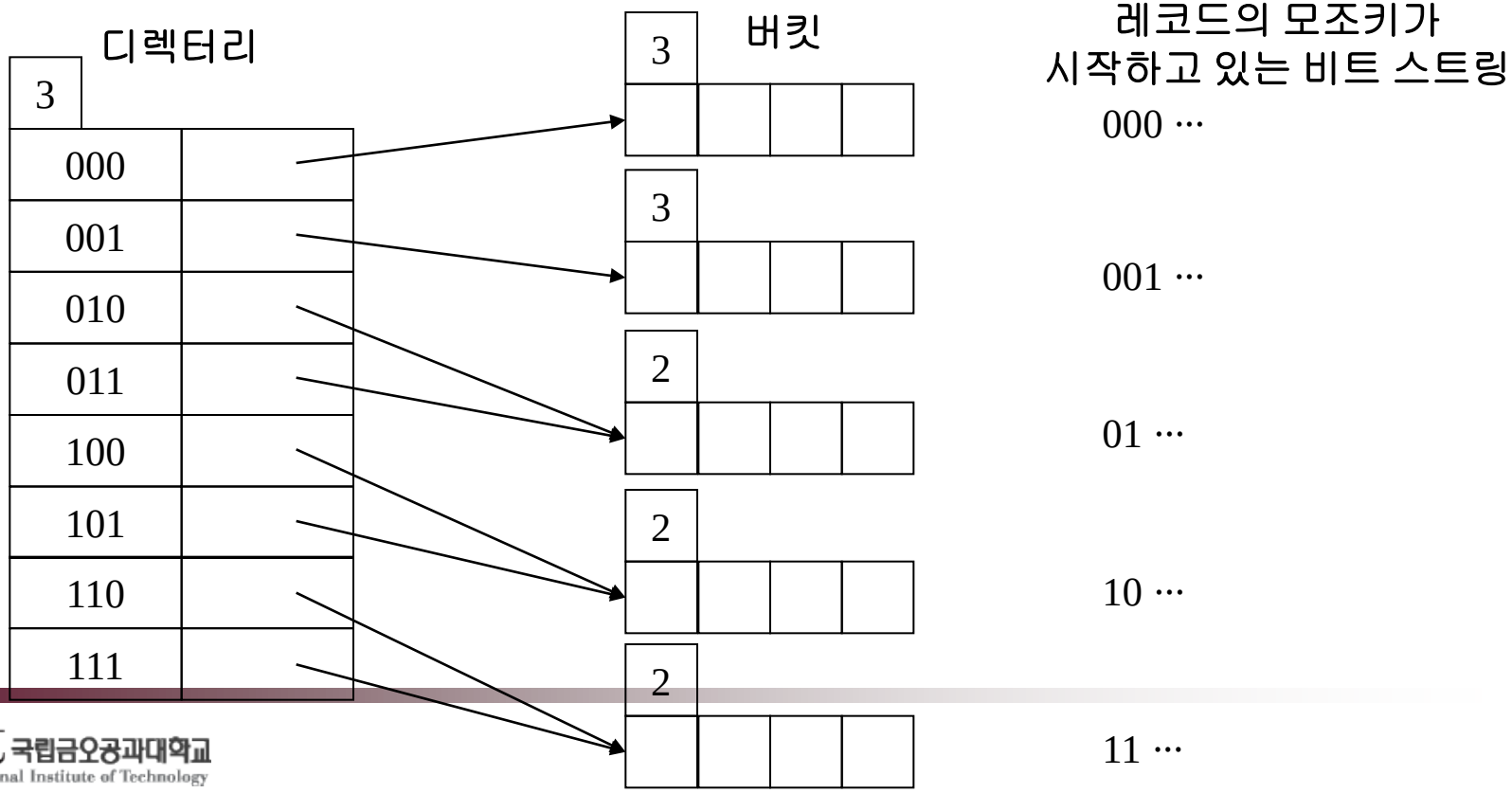
확장성 해싱의 연산 (2)

- 저장
  - 모조키의 처음 d 비트를 이용 디렉터리 접근
  - 포인터가 지시하는 버킷에 레코드 저장

  - 오버플로 처리(버킷이 만원일 때)
    - 새로운 버킷을 생성
    - 오버플로 버킷의 레코드들과 새로 저장할 레코드를 오버플로 버킷과 새로 할당한 버킷에 분산
      - 오버플로 버킷의 p+1값에 따라
    - 기존 버킷과 새로 할당된 버킷의 깊이 값 p는 모두 (p+1)로 설정
    - 디렉터리 오버플로(d< (p+1)인 경우)
      - 디렉터리 크기 증가(d 값을 1 증가시킴)
      - 포인터 조절

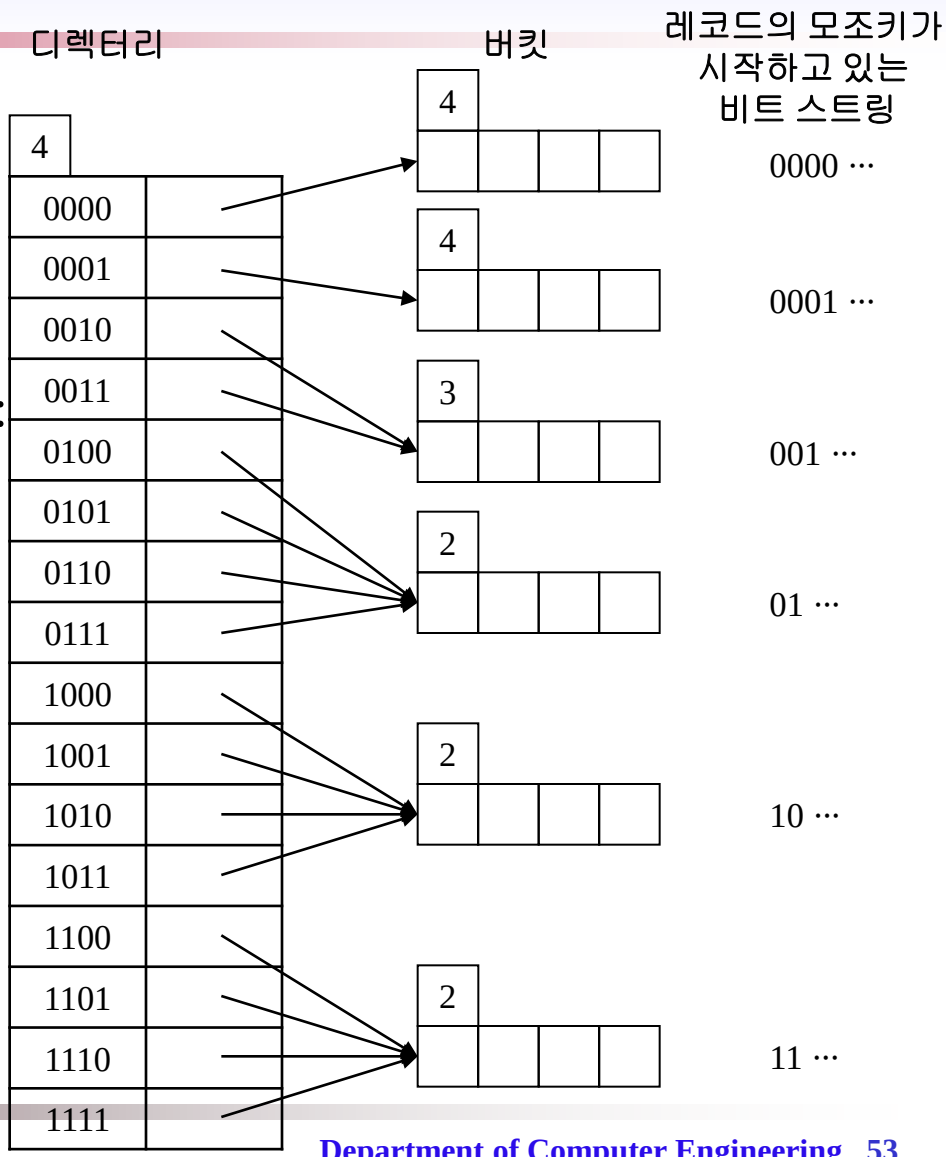
버킷 오버플로 예

- 버킷 4가 만원인 상태에서 모조키가 10으로 시작하는 레코드 삽입
- 버킷을 분할 : 빈 버킷 할당
  - 모조키 11로 시작되는 레코드를 새 버킷으로 이동
- 디렉터리의 110과 111의 포인터 값은 새 버킷을 지시하도록 변경
- 분할된 버킷의 깊이는 2로 증가



디렉터리 오버플로 예

- 첫번째 버킷(000)이 만원인 경우에 레코드 삽입
  - 전역 깊이(d)
    < (지역 깊이(p) + 1)
  - d를 증가시켜 디렉터리 확장 : 2배 증가
    - 빈 버킷을 할당받아 모조키가 0001로 시작되는 레코드를 이동
    - 디렉터리 헤더와 버킷 헤더에 있는 깊이 값 증가
    - 포인터 조정



확장성 해싱의 연산 (3)

- 삭제
  - 삭제할 레코드를 찾아 삭제
  - 한 버킷에 하나만 있는 레코드를 삭제하는 경우
    - 버디 버킷을 검사, 두 버디에 있는 레코드들이 한 버킷에 들어갈 수 있으면 합병
      - 버디 버킷(buddy bucket) : 똑같은 깊이 값(p)을 가지고 있고 그 모조 키들의 처음 (p－1)비트들은 모두 같은 버킷
    - 버킷의 새로운 깊이 값은 p－1
    - 모든 버킷들의 깊이 값이 디렉터리 깊이 값보다 작게 되면 디렉터리의 깊이를 하나 감소(d ← d-1)
      - 디렉터리 크기는 반으로 줄어듦
      - 포인터 엔트리들을 적절히 재조정