Chapter 12: 질의처리 (후반부-1)

개요

질의 비용산정

비용 산정을 위한 카타로그 정보

선택 연산

정렬

조인 연산

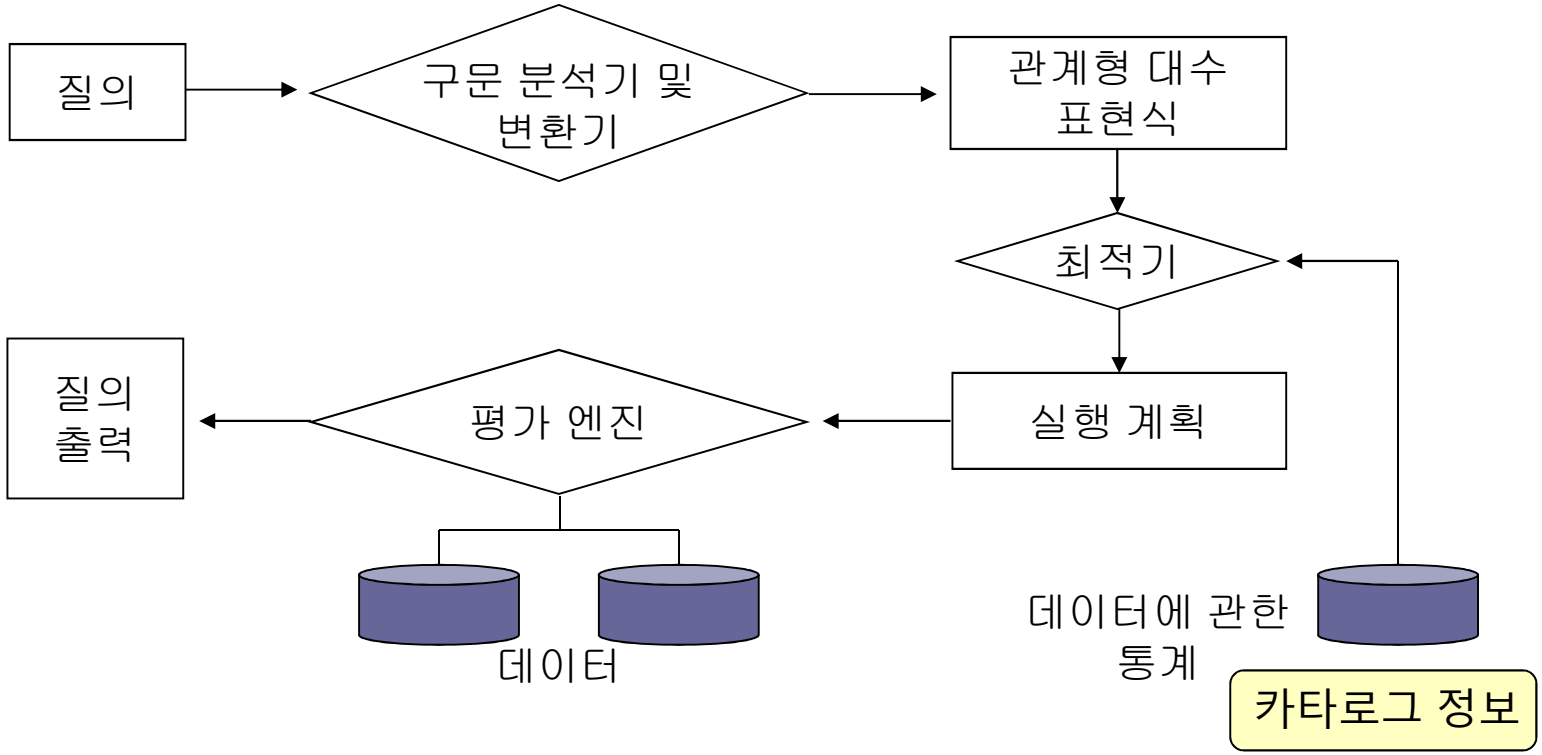
기타 연산

표현식의 평가

©Silberschatz, Korth and Sudarshan

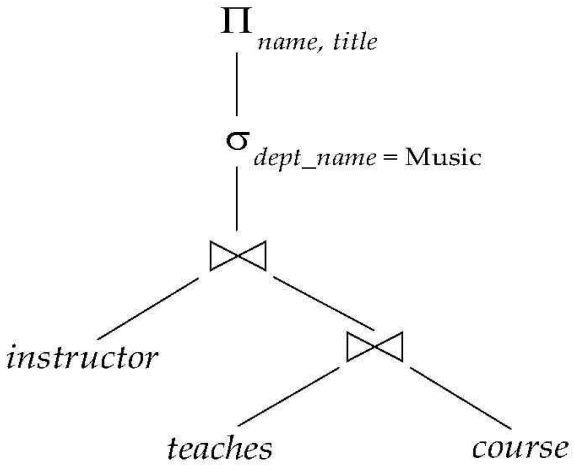
질의처리의 기본절차

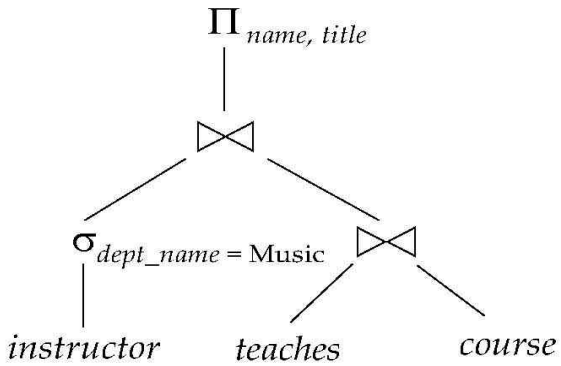
1. 구문 분석 및 변환
2. 최적화
3. 평가



질의 처리 비용 산정

```
SELECT name, title
FROM instructor NATURAL JOIN teaches NATURAL JOIN course
WHERE instructor.dept_name="Music";
```





일반적으로 디스크로부터의 블록 전송 수(**number of block transfers**)와 탐색 수 (**number of seeks**)가 평가의 실질적인 비용산정으로 사용된다.

t_T – time to transfer one block

t_S – time for one seek

b개의 블록 전송과 S번의 탐색 비용

$b * t_T + S * t_S$

질의 처리 비용 산정 (계속)

CPU 비용은 일반적으로 무시한다.

실시간 시스템에서는 CPU 비용을 산정하여야 한다.

이후, 비용 산정에 있어 디스크에 결과를 쓰기 연산하는 비용을 포함시키지 않는다고 가정한다.

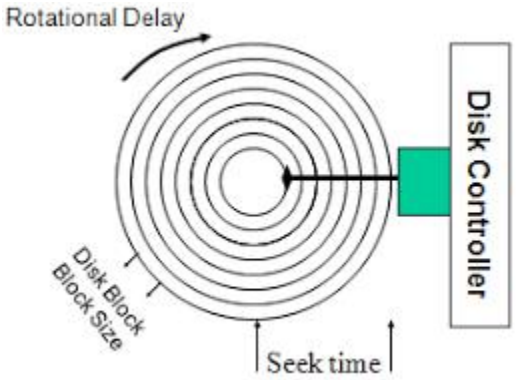
알고리즘의 비용은 많은 메모리를 가지면 디스크 액세스를 줄일 수 있으므로, 메인 메모리 내 버퍼의 크기에 따른다.

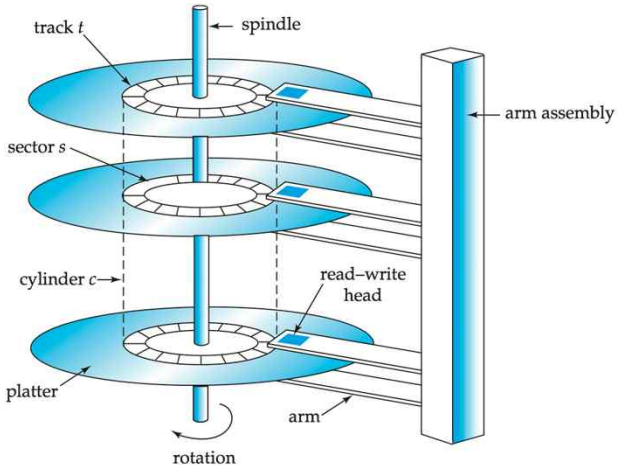
따라서, 비용을 산정할 때 메모리의 크기가 파라미터가 되어야 한다 : 흔히 최악의 경우의 비용을 사용한다.

비용 산정을 위한 카타로그 정보

instructor

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000





- n_r : 릴레이션 r내의 튜플의 수
- b_r : r의 튜플들을 내포하고 있는 블록의 수
- s_r : r의 한 튜플의 바이트 단위의 크기
- f_r : r의 블록킹 요인 – 즉, 한 블록에 들어가는 r의 튜플 수
- r의 튜플들이 파일내에 물리적으로 함께 저장되면, 다음과 같다.

$$b_r = \lceil \frac{n_r}{f_r} \rceil$$

- V(A, r) : 애트리뷰트 A에 대해 r에 나타나는 서로 다른 값의 수 ;

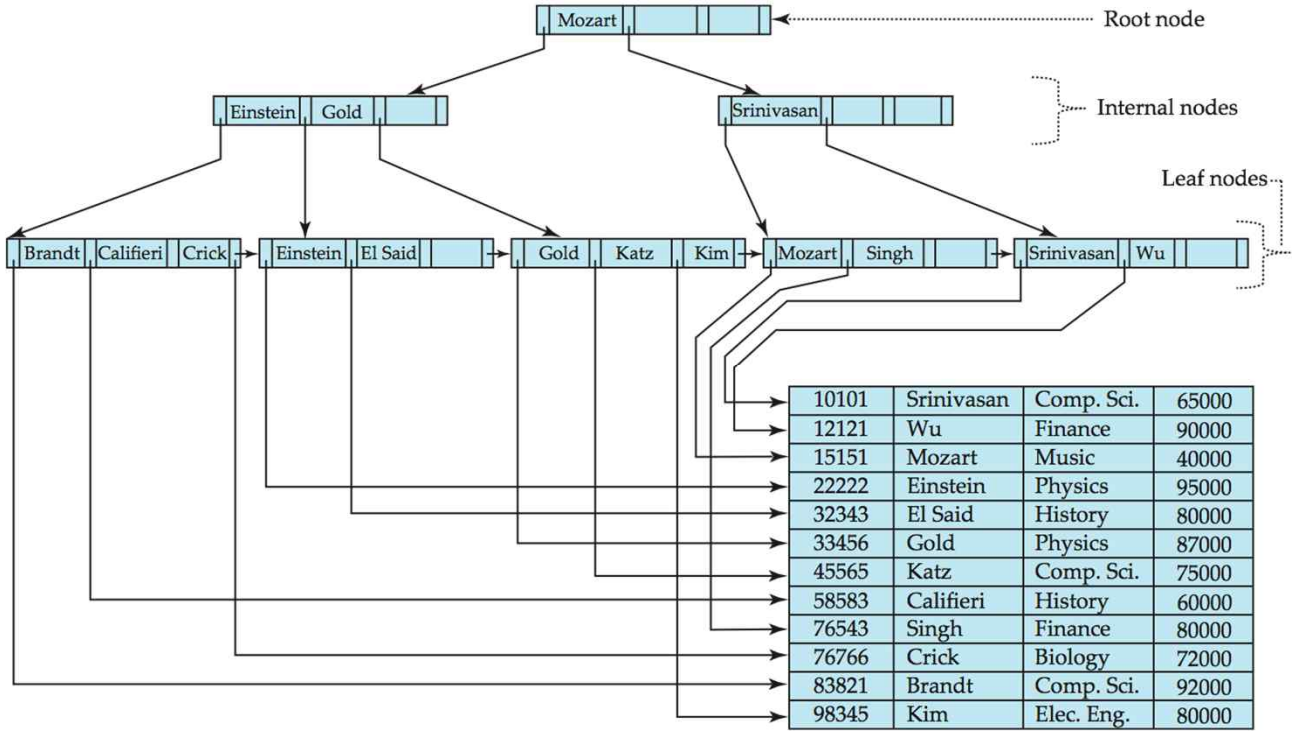
  $\Pi_A(r)$의 크기와 같다.

Example:
Block size 100byte로 가정
$n_{instructor}$ = 12
$b_{instructor}$ = 12
$s_{instructor}$ = 100 byte
$f_{instructor}$ = 1
$b_{instructor}$ = ceiling(12/1) = 12
V(name, instructor) = 12
V(ID, instructor) = 12
V(dept_name, instructor) = 7
. …

비용 산정을 위한 카타로그 정보 (계속)

- f_i : B^+ - 트리와 같은 트리 구조 인덱스에 있어, 인덱스 i의 내부 노드의 평균 전개
- h_i : 인덱스 i의 계층 수 – 즉, i의 높이이다.
  - (B^+ - 트리 같은) 인덱스에 있어, $h_i = \lceil \log_{f_i}(V(A, r)) \rceil$
  - 해쉬 인덱스에 있어 h_i는 1이다.

Example:
f_i=2
h_i=3



Example of B^+-Tree

선택 연산

ID	name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

(Q1) $\sigma_{salary>=75000}(instructor)$ (Q2) $\sigma_{name=Kim}(instructor)$

파일 스캔 (File scan)

Algorithm **A1** (선형 검색). 각 파일 블록을 스캔해 모든 레코드가 선택 조건을 만족하는지 여부를 테스트 한다.

산정 비용 = b_r block transfers + 1 seek

- b_r 은 릴레이션 r 의 블록수를 나타낸다.

선택이 키 애트리뷰트상에 행해지는 경우

(레코드를 찾으면 중단)

- cost = (b_r/2) block transfers + 1 seek

선형 검색은 다음에 조건에 관계없이 적용될 수 있다.

- 선택 조건 또는
- 파일 내 레코드의 순서, 또는
- 인덱스의 유무

Example:

- Cost = b_r = 12
- Cost = (b_r/2) = 6

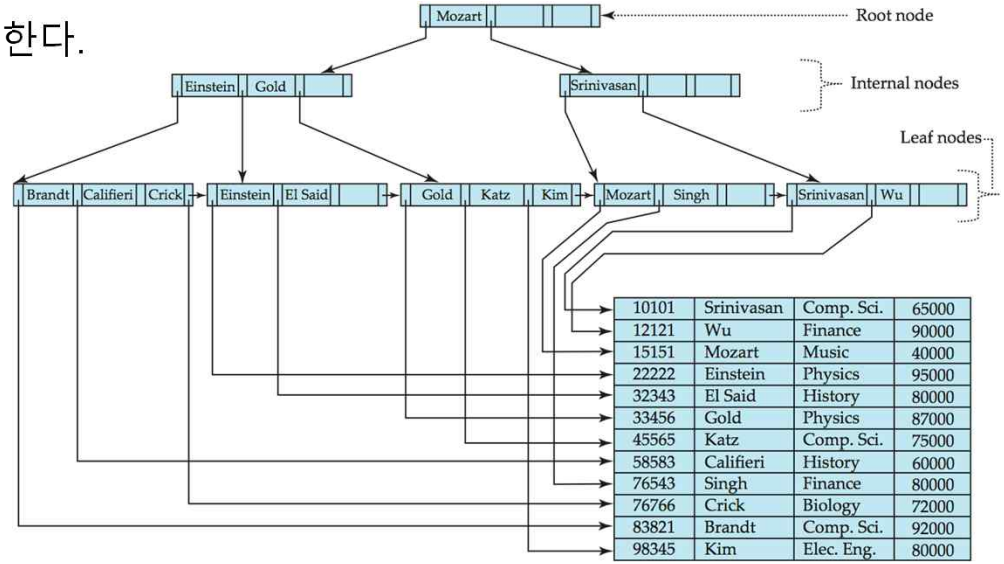
Note: 데이터가 정렬되어 저장되어 있지 않으므로 일반적으로 binary search는 적합한 방법이 아니다.

인덱스를 사용한 선택 연산

인덱스 스캔(Index scan) – 인덱스를 사용하는 검색 알고리즘

조건은 인덱스의 검색 키상에 있어야 한다.

Ex) $\sigma_{name=Kim}(instructor)$



A2 (후보 키상의 주 인덱스, 동등조건)

- 상응하는 조건을 만족하는 하나의 레코드를 검색한다.

  $Cost = (h_i + 1) * (t_T + t_S)$

Ex) Cost = 3+1=4

A3 (비 키상의 주 인덱스, 동등 조건)

- 여러 레코드를 검색한다. 레코드가 연속 블록에 위치하며, b개의 연속된 블록에해당한다고 가정.

  $Cost = h_i * (t_T + t_S) + t_S + t_T * b$

A4 (2차 인덱스의 검색 키상의 동등 조건).

검색 키가 후보 키이면 하나의 레코드를 검색한다.

- $Cost = (h_i + 1) * (t_T + t_S)$

검색 키가 후보 키가 아니면 여러 레코드를 검색한다 (각각은 서로 다른 블록에 있을 수 있다).

- Cost = $(h_i + n) * (t_T + t_S)$ Can be very expensive!

비교 연산을 내포한 선택연산

$\sigma_{A \le v}(r)$ 또는 $\sigma_{A \ge v}(r)$ 형태의 선택을 선형 검색, 이진 검색을 사용 하거나 다음과 같은 다양한 방식의 인덱스를 사용해 수행한다

A5 (**primary index, comparison**

A6 (**secondary index, comparison**).

복합 선택 연산

Conjunction: $\sigma_{\theta 1 \wedge \theta 2 \wedge \ldots \theta n}(r)$

Ex) $\sigma_{dept\_name=Comp.\ Sci.\ and\ salary\ <70000}(instructor)$

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

A7 (**conjunctive selection using one index**).

A8 (**conjunctive selection using composite index**).

A9 (**conjunctive selection by intersection of identifiers**).

Disjunction: $\sigma_{\theta 1 \vee \theta 2 \vee \ldots \theta n}(r)$.

A10 (**disjunctive selection by union of identifiers**).

Negation: $\sigma_{\neg \theta}(r)$

Use linear scan on file

정렬 (sorting)

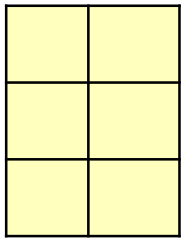
릴레이션에 인덱스를 구축하여 정렬된 순서로 릴레이션을 읽는데 인덱스를 사용한다. 각 튜플마다 하나의 디스크 블록 액세스가 야기될 수 있다.

메모리에 수용되는 릴레이션에 대해서는 퀵 정렬(quicksort)같은 기법이 사용될 수 있다. 메모리에 수용되지 않는 릴레이션에 대해서는 외부 정렬-합병(**external sort-merge**) 기법이 좋은 방법이다.

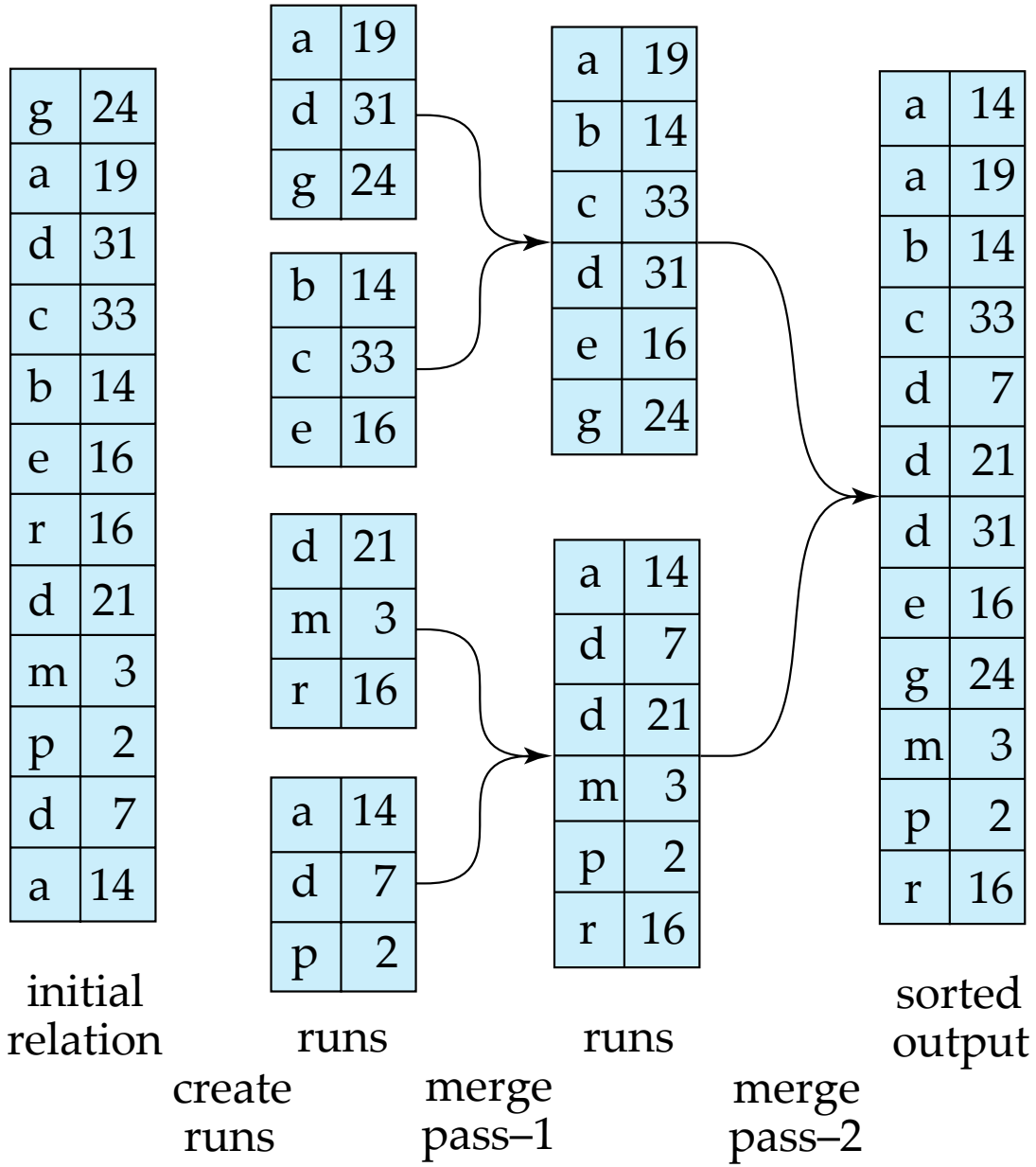
예제: 외부 정렬 – 합병

메모리 크기 (M) = 3 블록

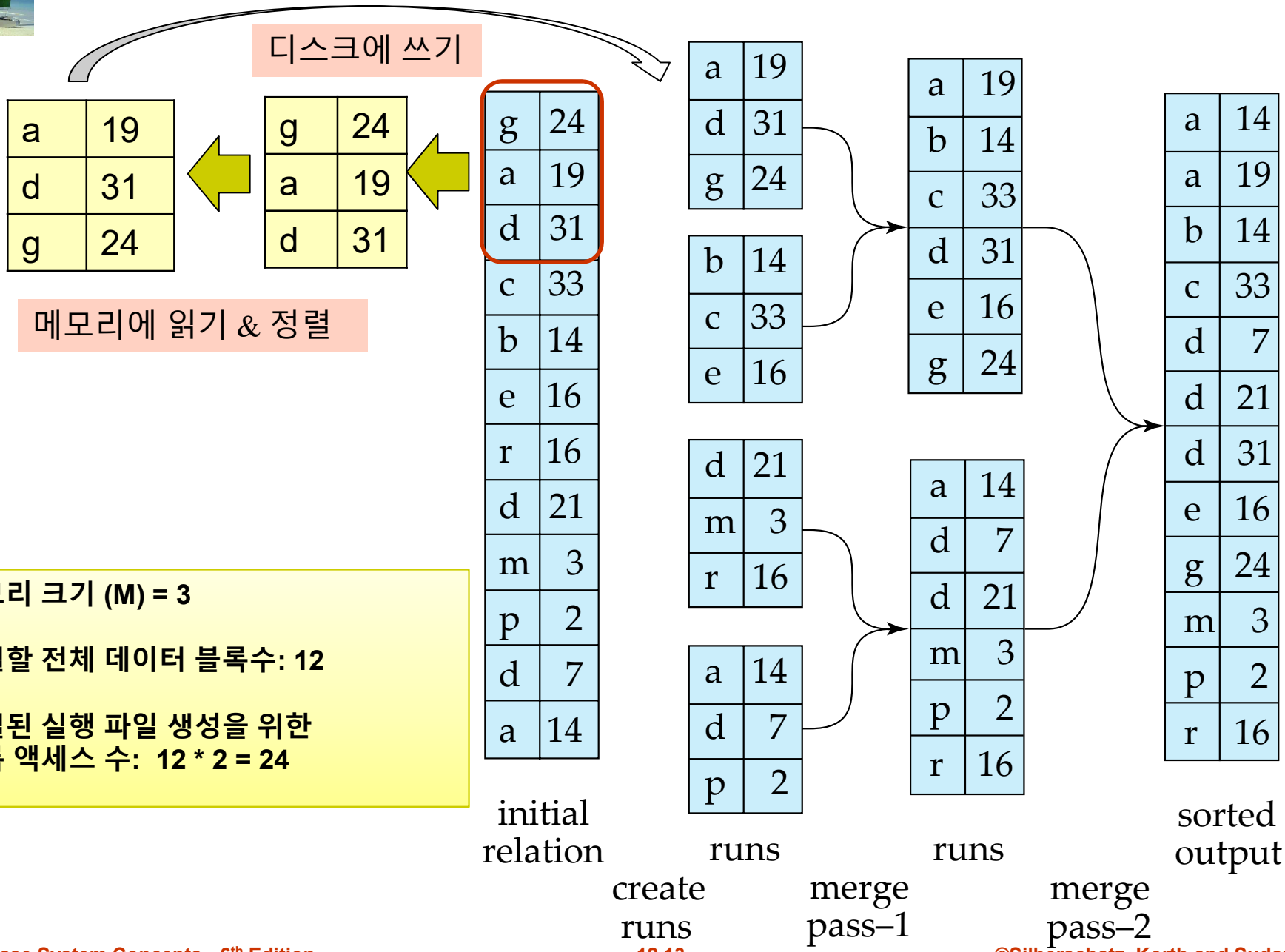
정렬할 전체 데이터 블록수: 12



메모리



예제: 외부 정렬 – 합병



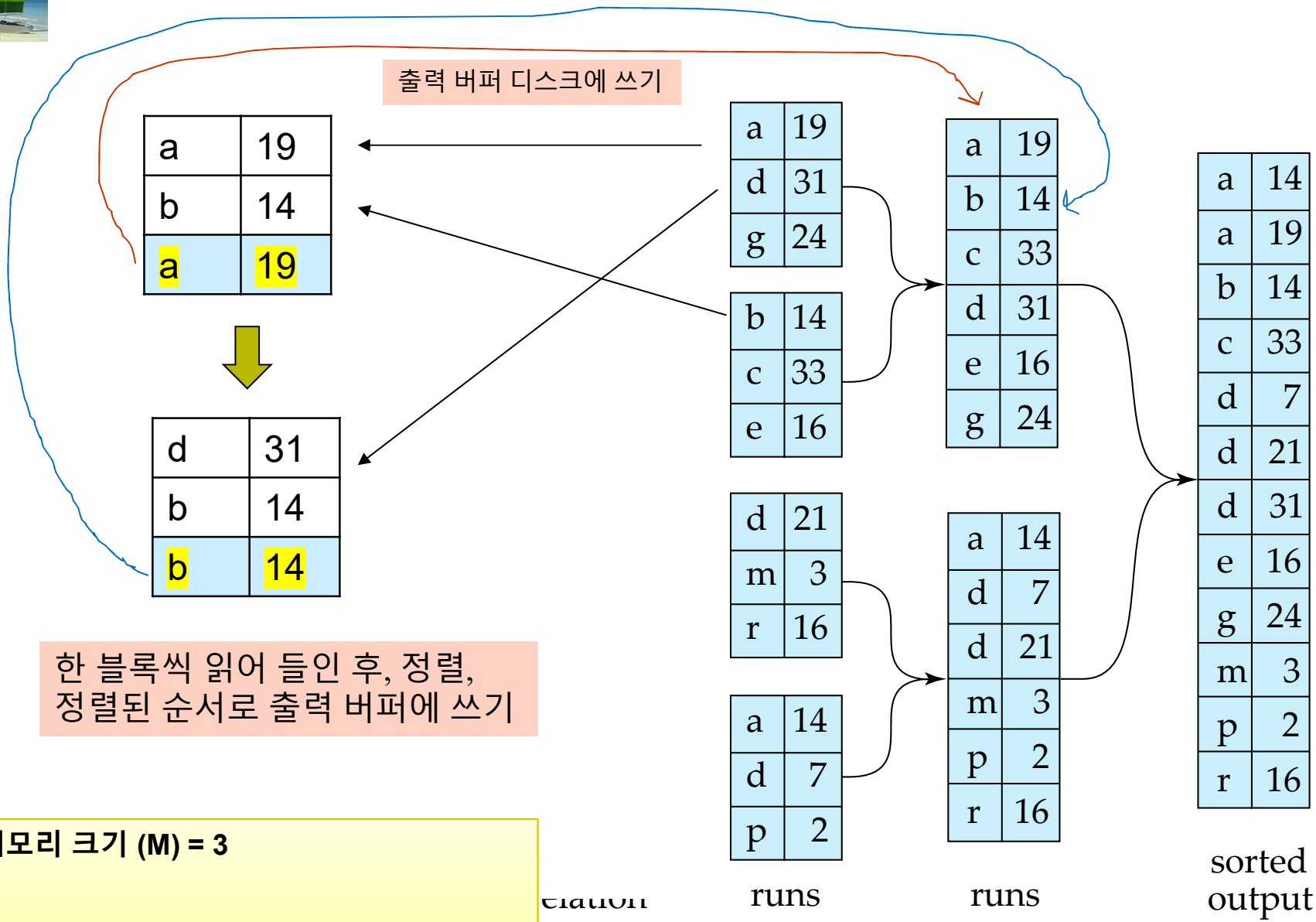
메모리 크기 (M) = 3

정렬할 전체 데이터 블록수: 12

정렬된 실행 파일 생성을 위한 블록 액세스 수: 12 * 2 = 24

©Silberschatz, Korth and Sudarshan

예제: 외부 정렬 – 합병

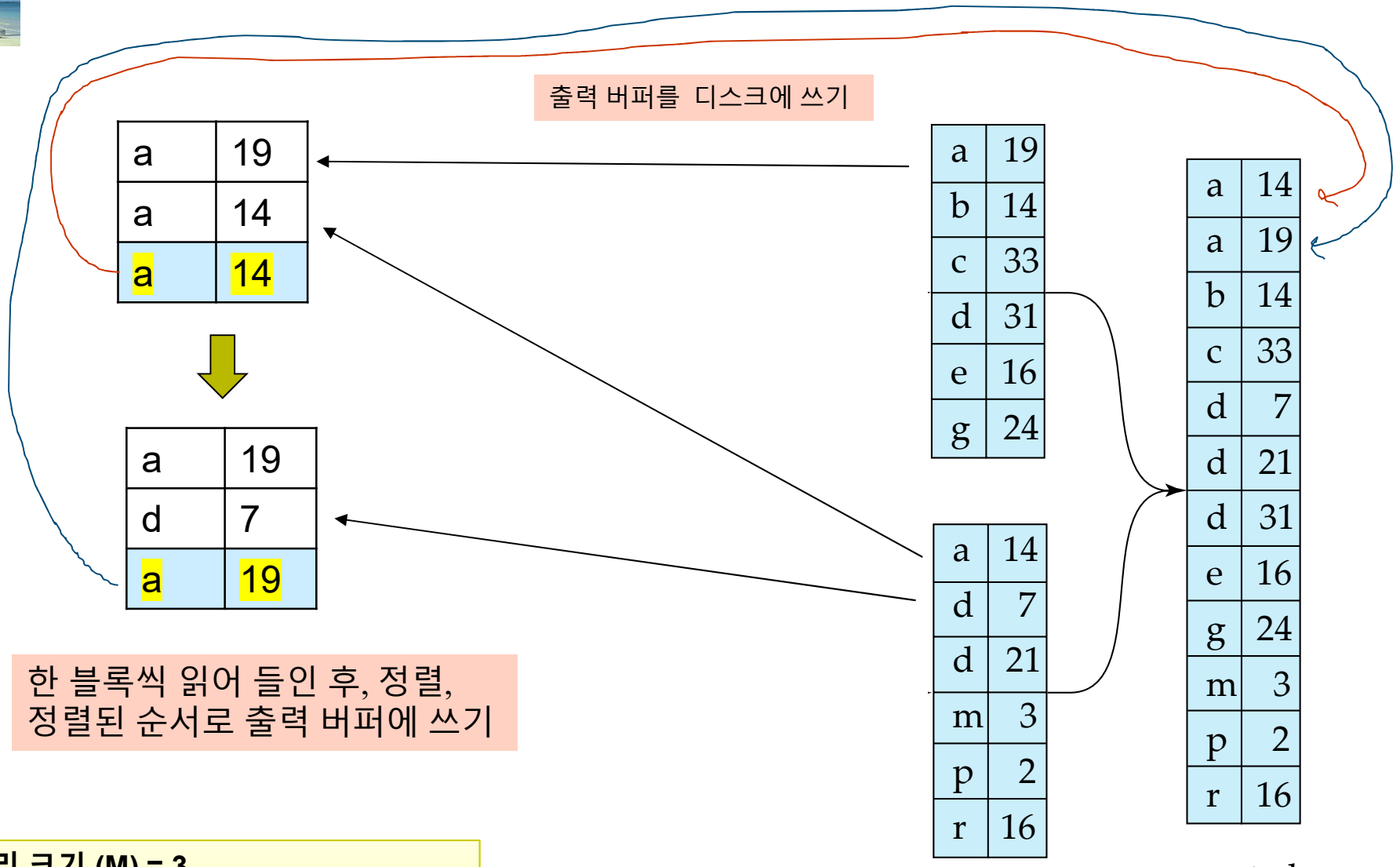


메모리 크기 (M) = 3

머지 1단계 블록 액세스 수: 12 * 2 = 24

©Silberschatz, Korth and Sudarshan

예제: 외부 정렬 – 합병



메모리 크기 (M) = 3

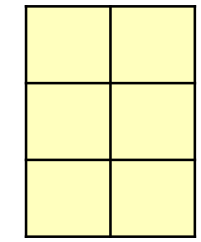
머지 2단계 블록 액세스 수: 12 * 2 = 24

©Silberschatz, Korth and Sudarshan

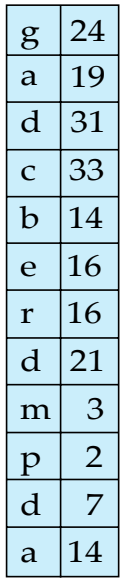
외부 정렬/합병 알고리즘

*M*을 페이지 단위의 메모리 크기라고 가정한다.

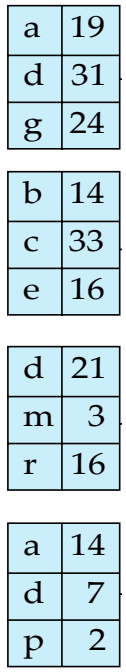
1. 다음과 같이 정렬된 실행 파일을 생성한다. 처음에 *i*를 0으로 한다.
   릴레이션의 끝까지 다음 과정을 반복해 수행한다.
   (a) 릴레이션의 M블록을 메모리에 읽어 들인다.
   (b) 메모리내에서 블록을 정렬한다.
   (c) 정렬된 데이터를 실행 파일 R_i에 기록하고 *i*를 증가 시킨다.
   이렇게 얻어진 마지막 *i* 값을 *N*이라고 하자.



메모리(M)



create runs

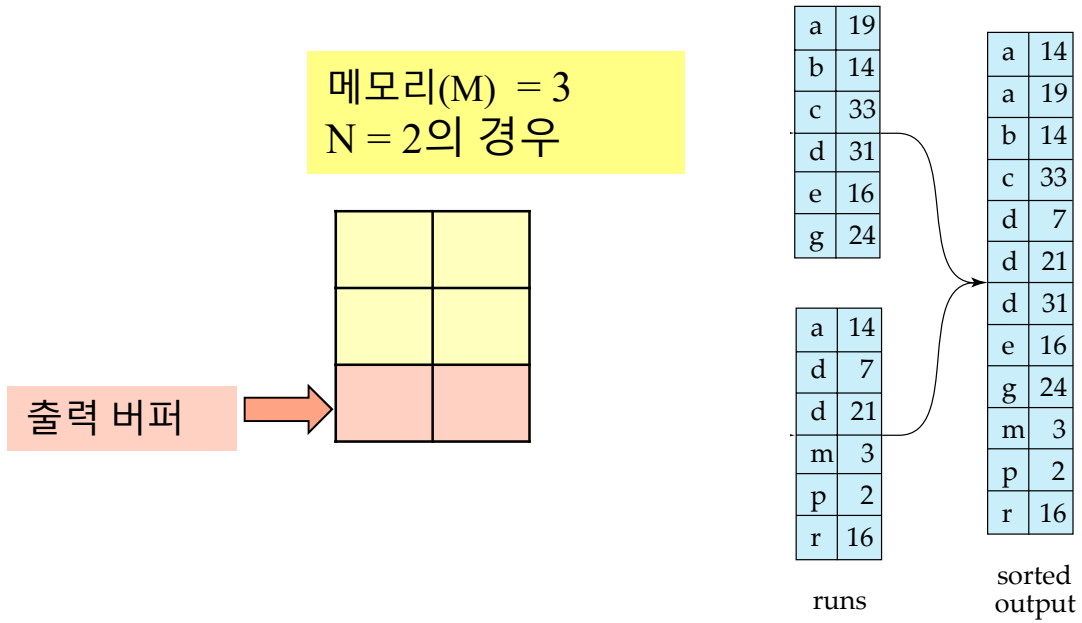


 ©Silberschatz, Korth and Sudarshan

외부 정렬 – 합병(계속)

2. 실행 파일을 합병한다 **(N-way merge)**. ; *N* < *M*이라 가정하자.

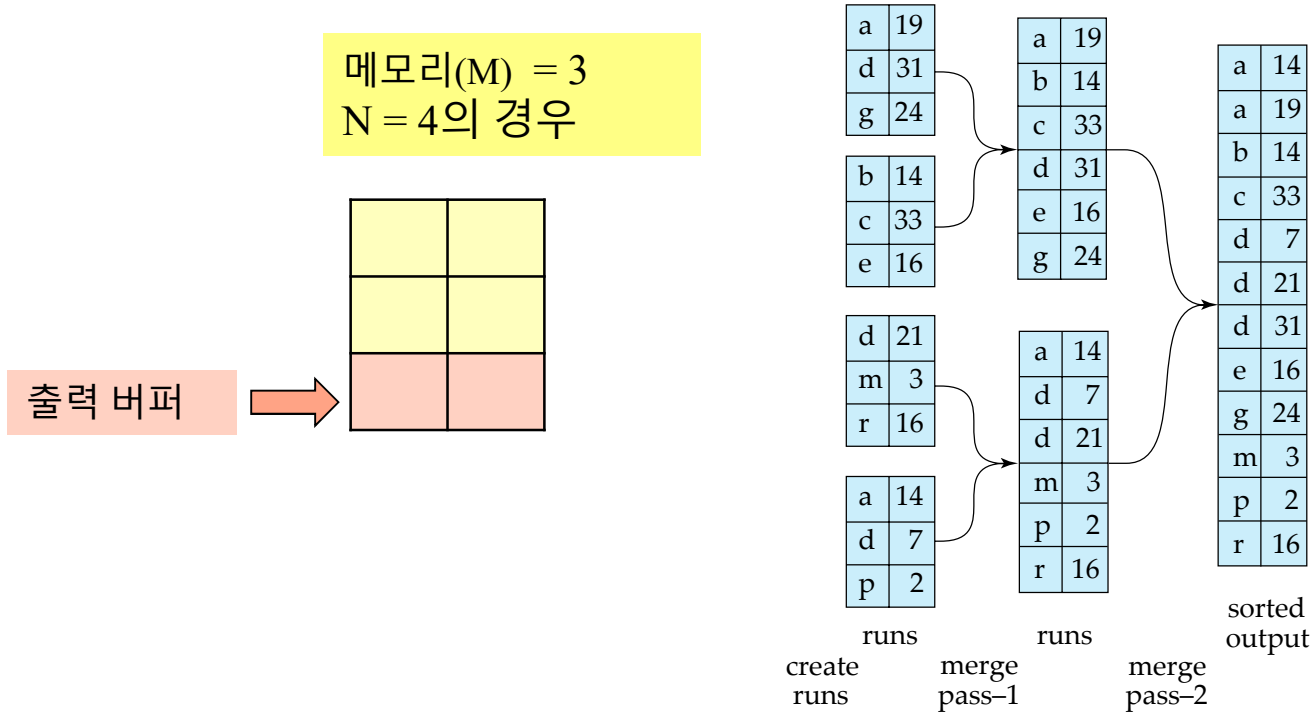
1. 단일 합병 절차에서는 입력 실행 파일을 버퍼링하는데 *N*개의 메모리 블록과 출력을 버퍼링하는데 1개의 블록을 사용한다. 우선 각 실행 파일의 첫 번째 블록을 버퍼에 읽어 들인다.
2. 모든 입력 버퍼 페이지가 빌때까지 다음 과정을 반복해 수행한다
   1. 각 버퍼에서 정렬된 순으로 첫 번째 레코드를 선택한다.
   2. 그 레코드를 출력버퍼에 기록한다. 출력버퍼가 다 차면
      그 내용을 디스크에 출력한다.
   3. 입력 버퍼 페이지에서 그 레코드를 삭제한다 ; 버퍼 페이지가 비면, 실행 파일의 다음 블록(있으면)을 버퍼에 읽어 들인다.



외부 정렬 – 합병(계속)

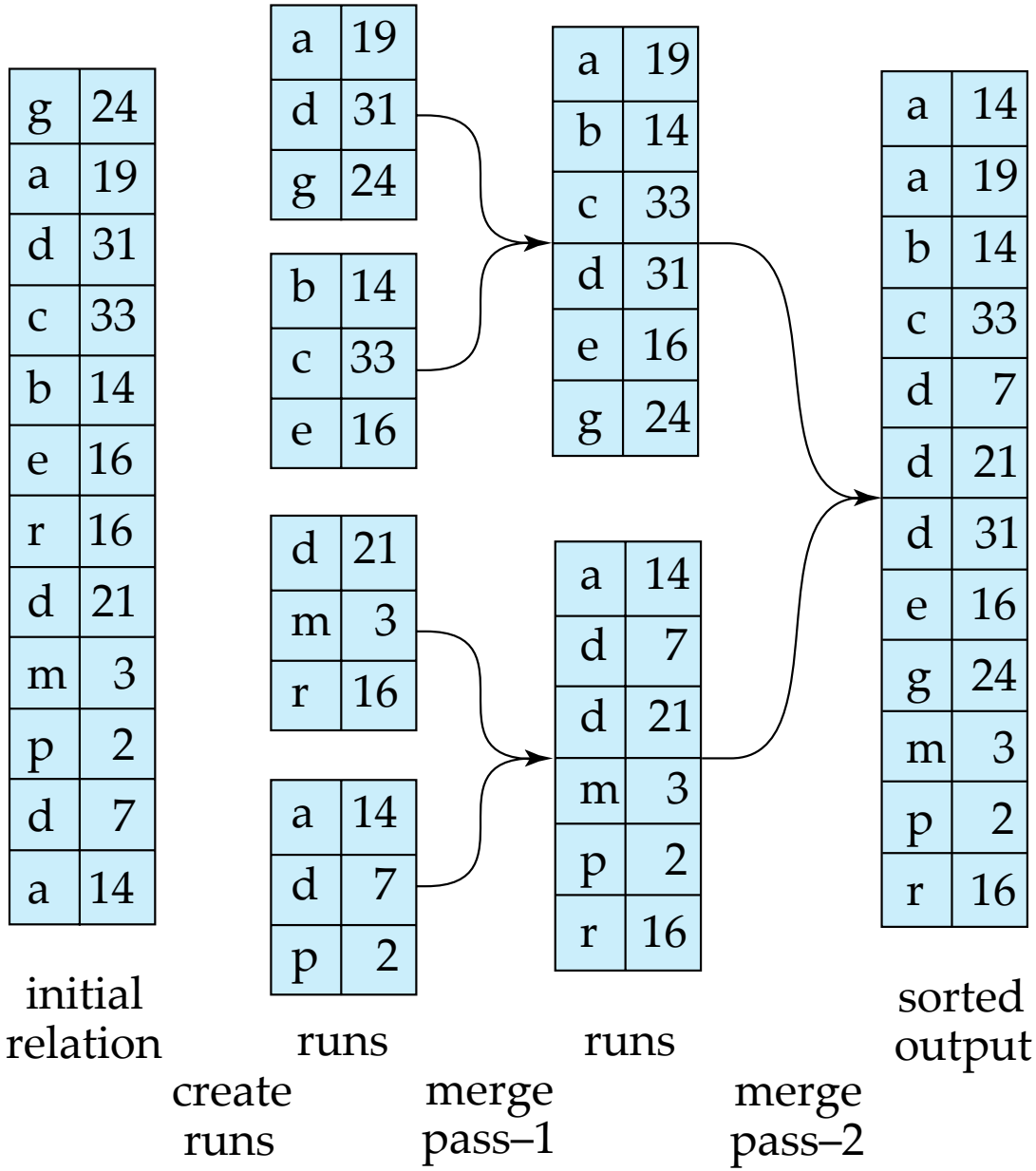
$N \geq M$ 이면, 여러 개의 합병 패스가 필요하다.

- 각 패스에서, $M-1$개의 실행 파일의 연속 그룹이 합병된다.
- 각 패스는 $M-1$의 단위로 실행 파일의 수를 감소시키고, 같은 $M-1$의 단위로 보다 긴 실행 파일을 생성한다.
  - E.g. M=11이고 90 개의 실행 파일이 있는 경우, 한 번의 패스는 실행 파일의 개수를 9개로 줄이는 효과를 얻는다. 또한 이 때 얻어진 각 실행 파일은 초기 사이즈의 10배에 해당하는 크기를 갖게 된다.
- 모든 실행 파일이 하나로 합병될 때까지 반복 패스가 수행된다.



©Silberschatz, Korth and Sudarshan

예제: 외부 정렬 – 합병



 ©Silberschatz, Korth and Sudarshan

외부 정렬 – 합병(계속)

비용 분석: (탐색 비용을 제외한 블록 전송 비용만 고려하는 경우)

- 각 패스를 포함해 초기 실행 파일 생성을 위한 디스크 액세스는 $2b_r$이다
  (마지막 패스는 예외인데, 결과를 디스크에 출력하지 않는다고 가정한다).
- 필요한 합병 패스의 총 수 : $(b_r/M)/(M-1)^n = 1$

$$n = \lceil \log_{M-1}(b_r / M) \rceil$$

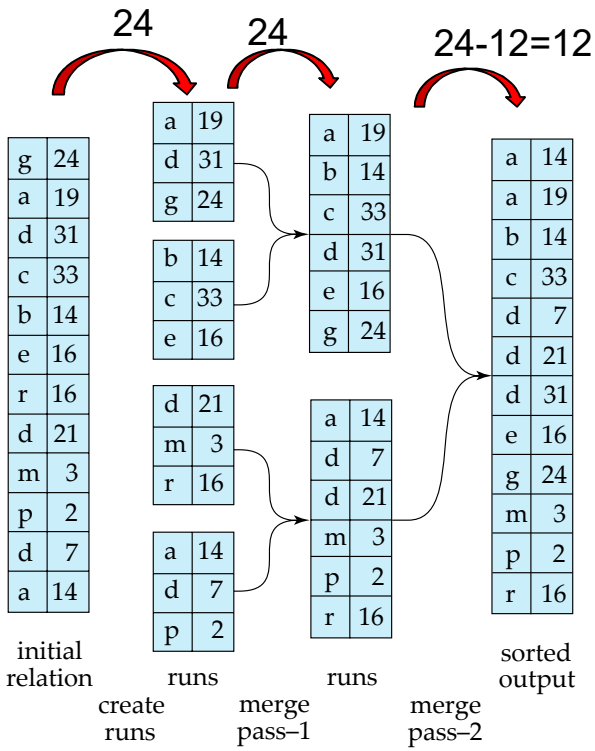
따라서, 외부 정렬을 위한 디스크 액세스의 총 수는 :

$$b_r(2\lceil \log_{M-1}(b_r / M) \rceil + 1)$$

메모리 크기 (M) = 3

정렬할 전체 데이터 블록수 b_r = 12

1. **정렬된 실행 파일 생성: 2* b_r= 24**
2. **합병 패스 수: $\log_2(12/3)=2$**
3. **디스크 액세스 총 수: 12(2*2+1) = 60**



Thank You