

# 42 Seoul & CUBRID

42 SEOUL & CUBRID

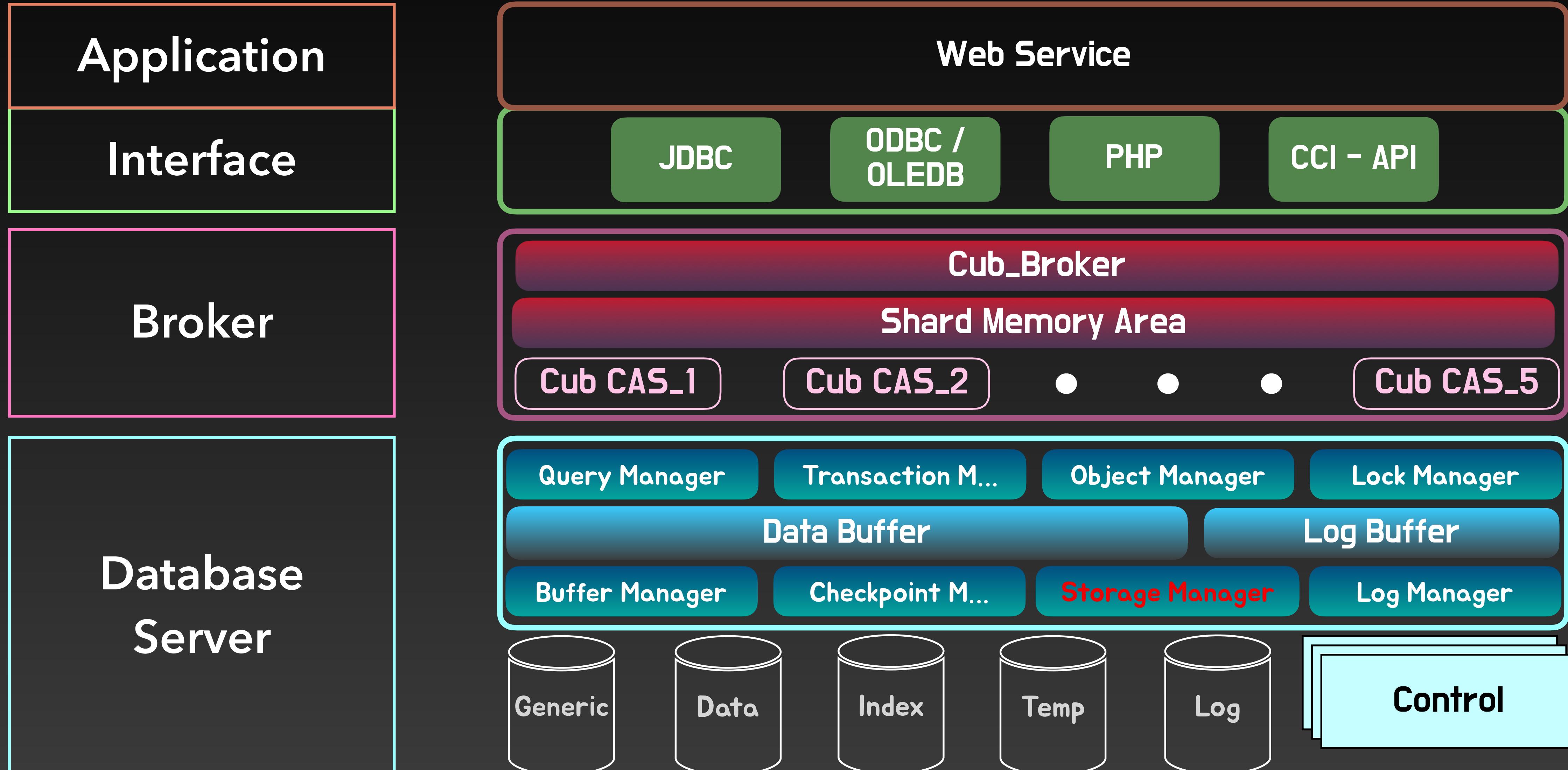
Phase 3  
Disk Manager

*bypark jolim seubaek jseo jinbekim samin*

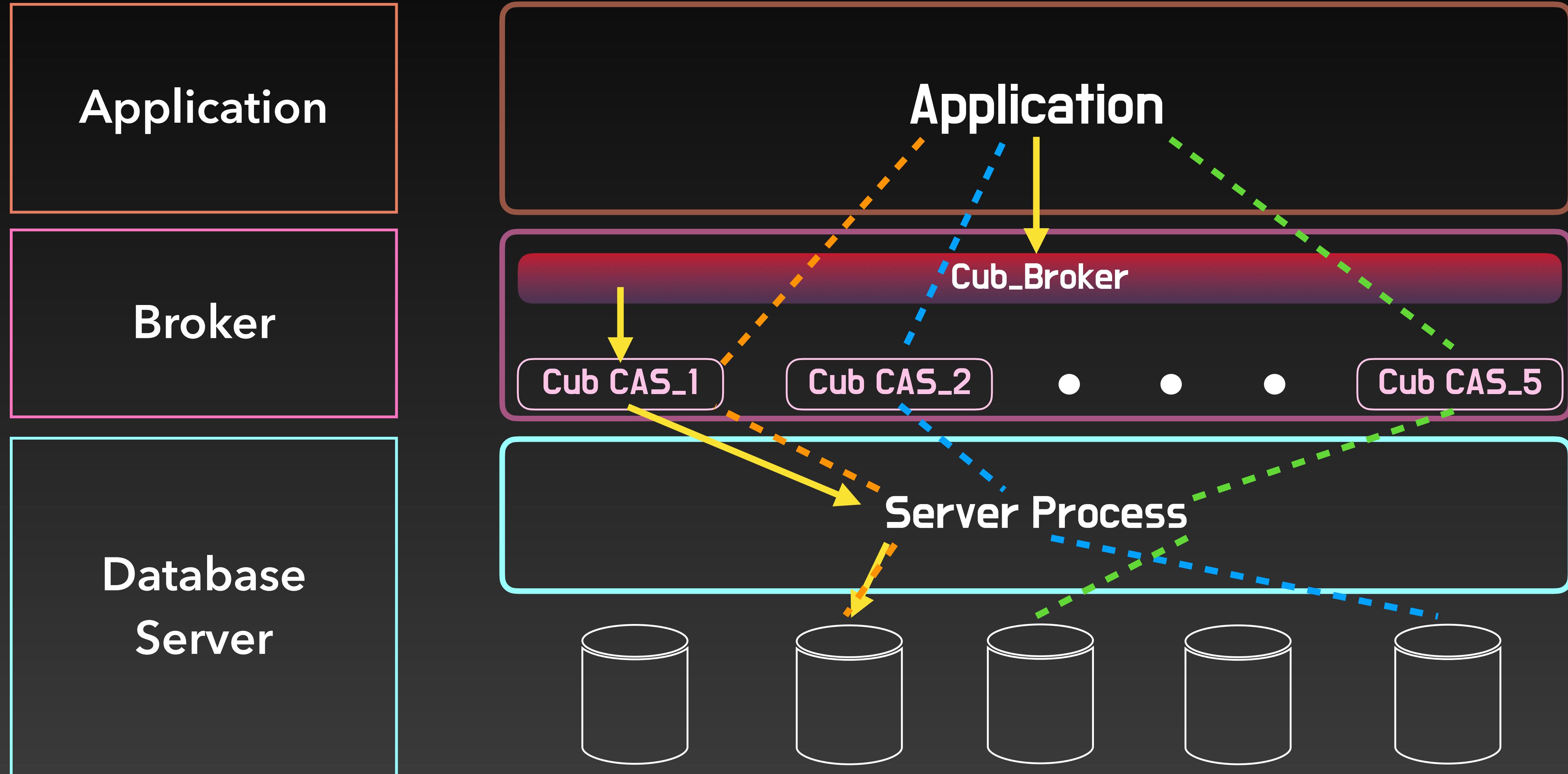
# Disk Manager

by bypark

# CUBRID 시스템 구조

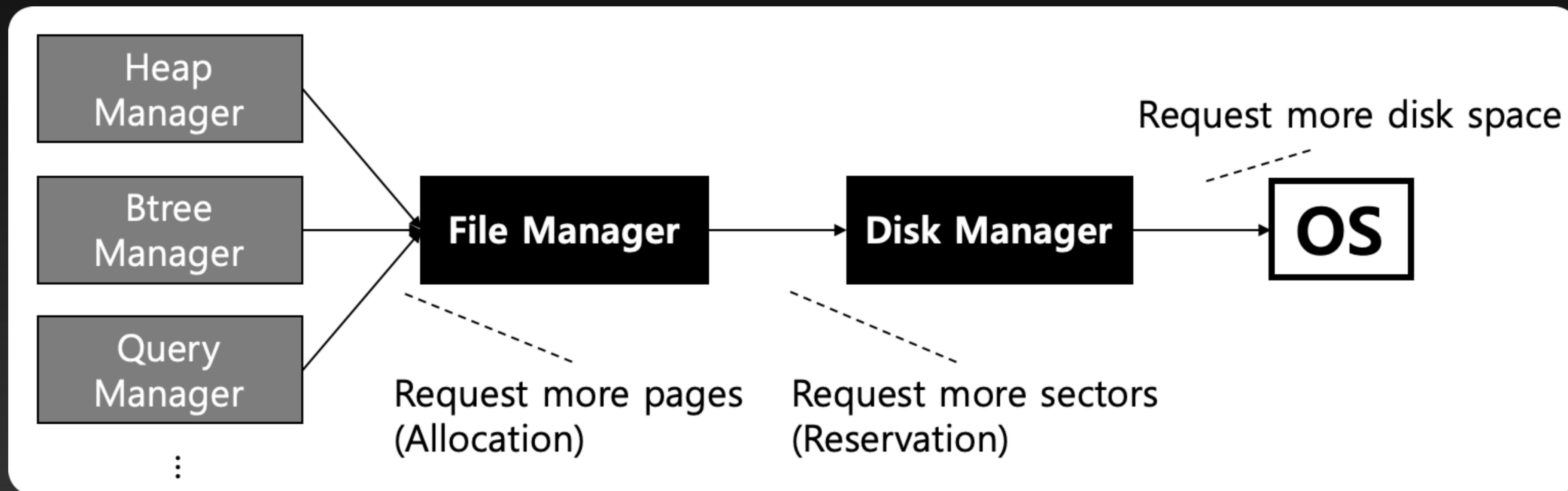


# CUBRID 시스템 구조



# Storage Manager

## Disk / File Manager



*Disk Manager* → 볼륨 공간 전체 관리, 섹터들의 예약여부 트래킹

disk\_\* prefix

*File Manager* → 파일 내에서의 페이지 할당 여부 트래킹

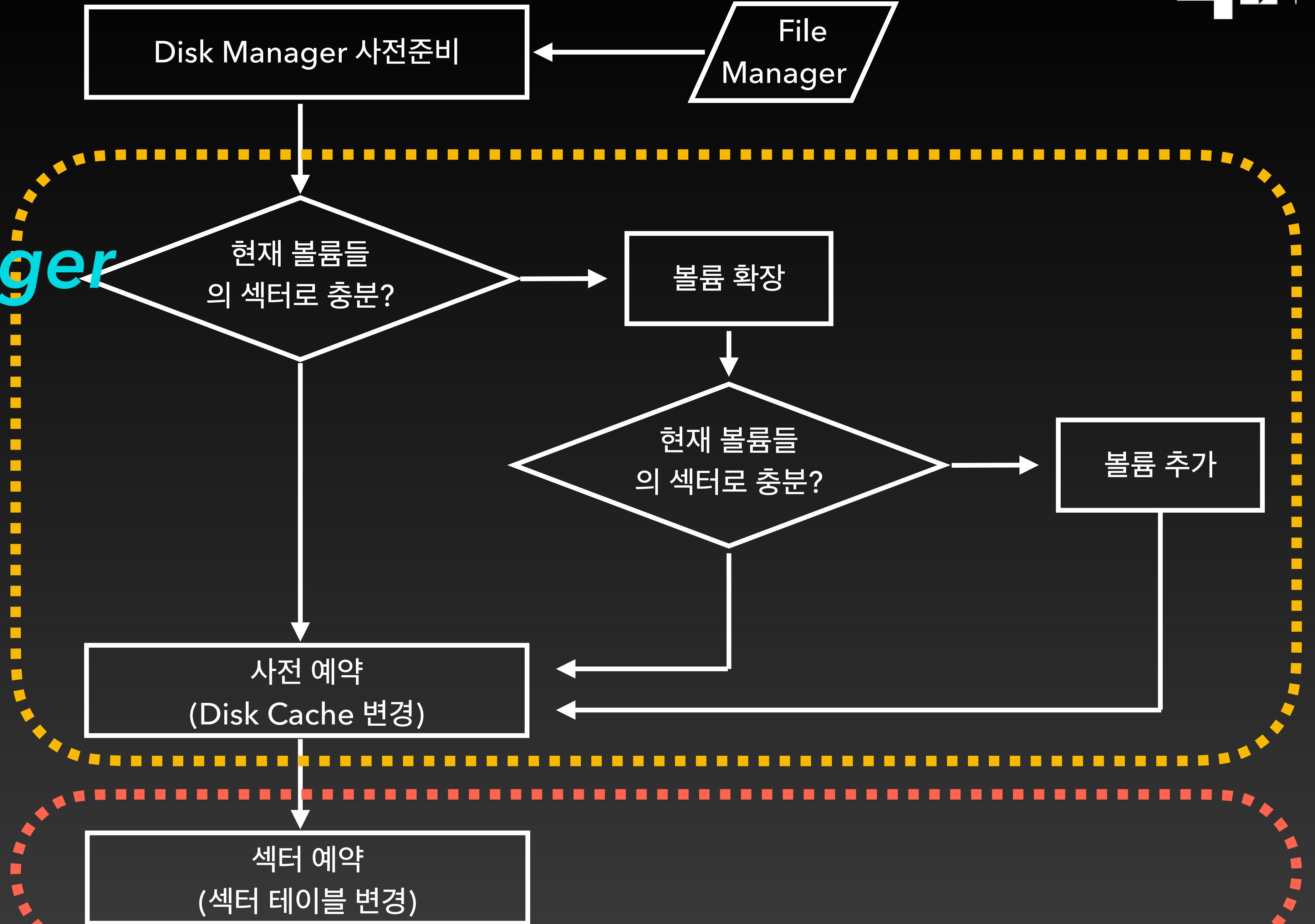
file\_\* prefix

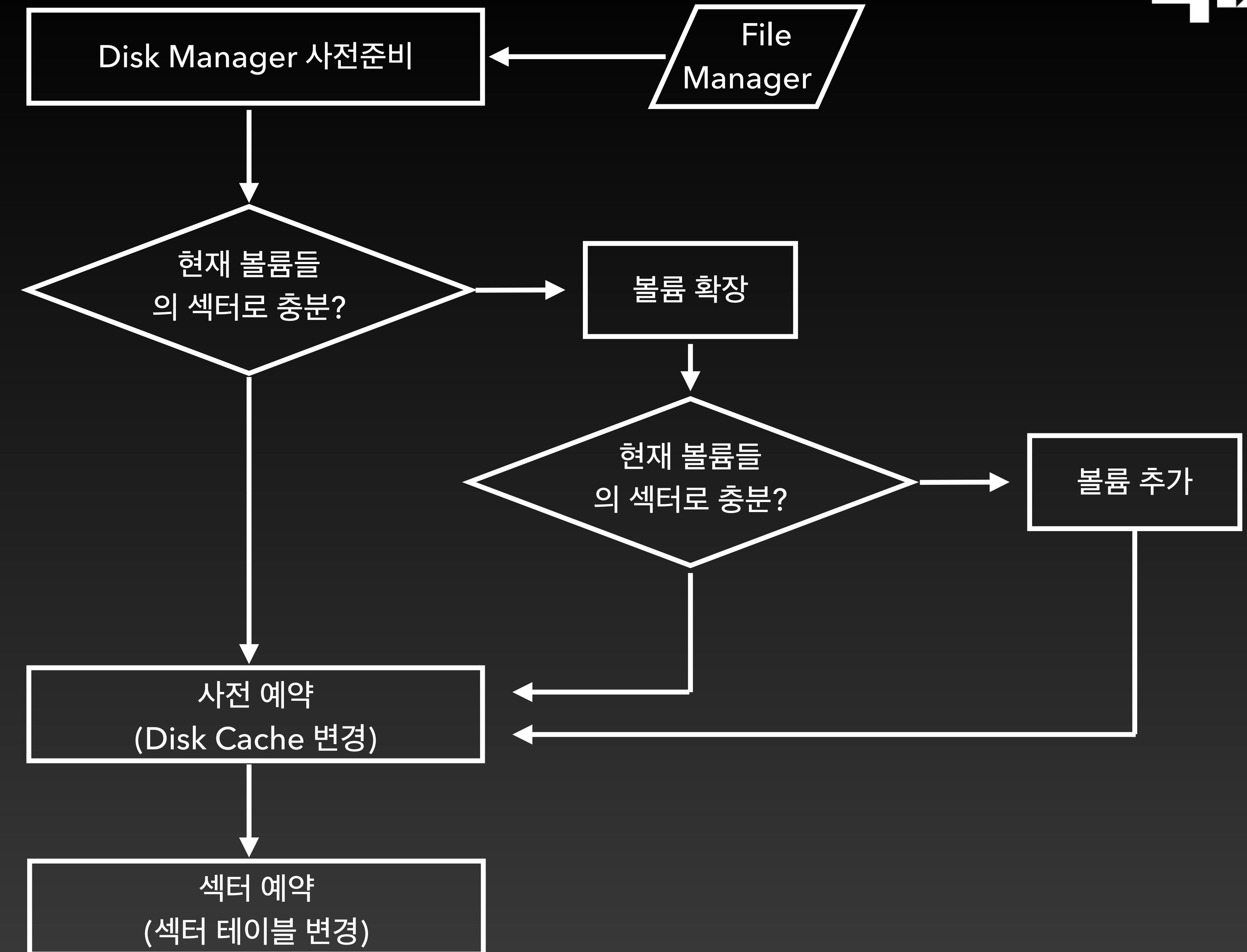
*Disk Manager* → 볼륨 공간 전체 관리, 섹터들의 예약여부 트래킹

## 2 Step Sector Reservation Disk Manager

## Disk Cache 사전예약

## 실제적인 예약표시 과정

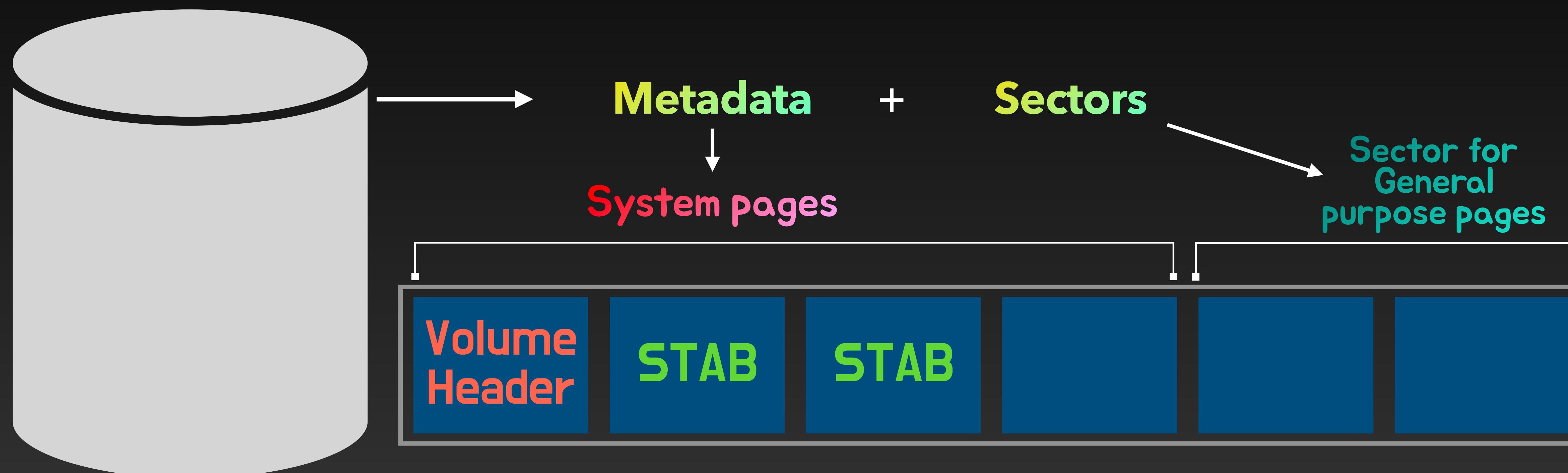




1. Volume 구조

2. Sector Reservation STRUCT

# Volume 구조

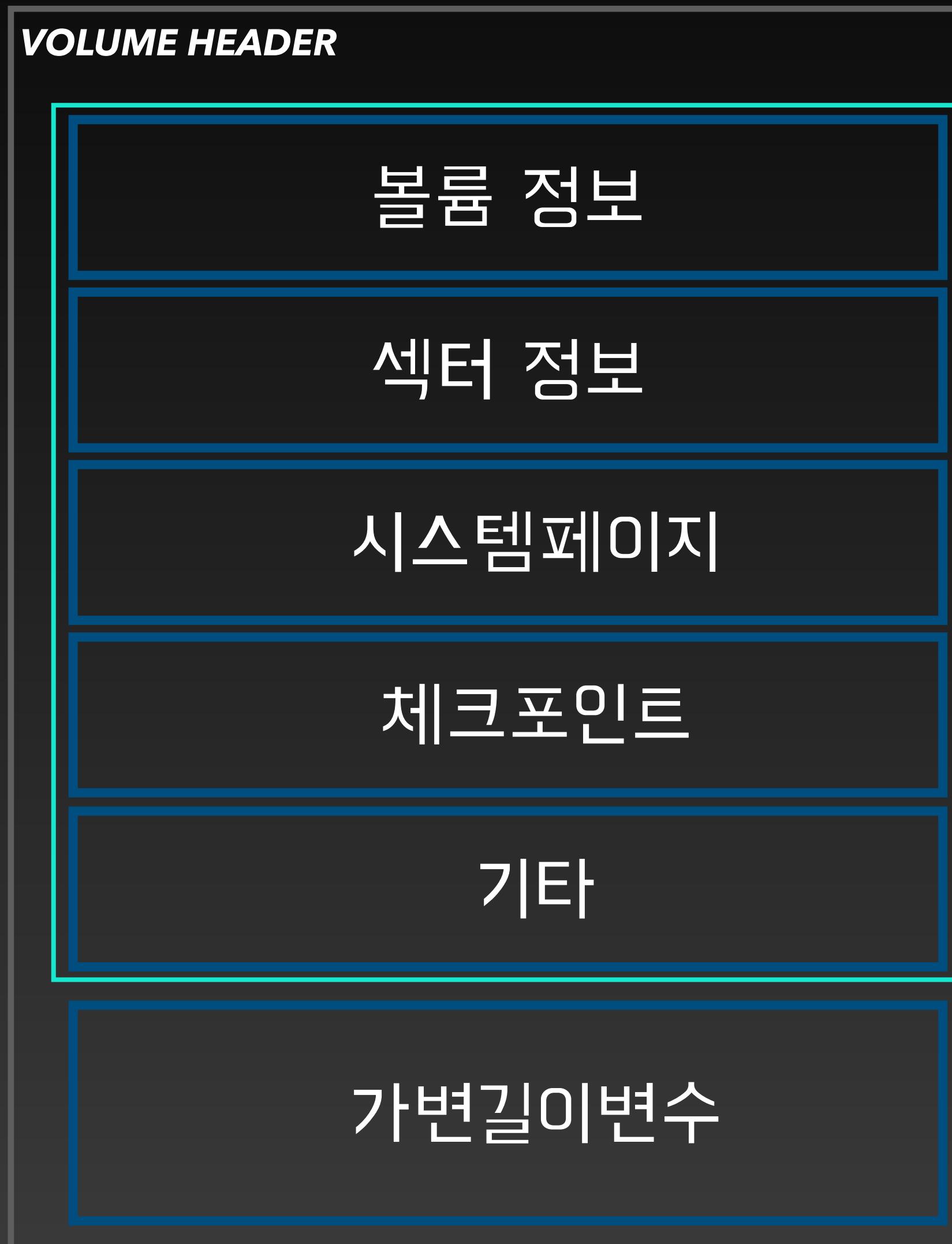


# Volume Header

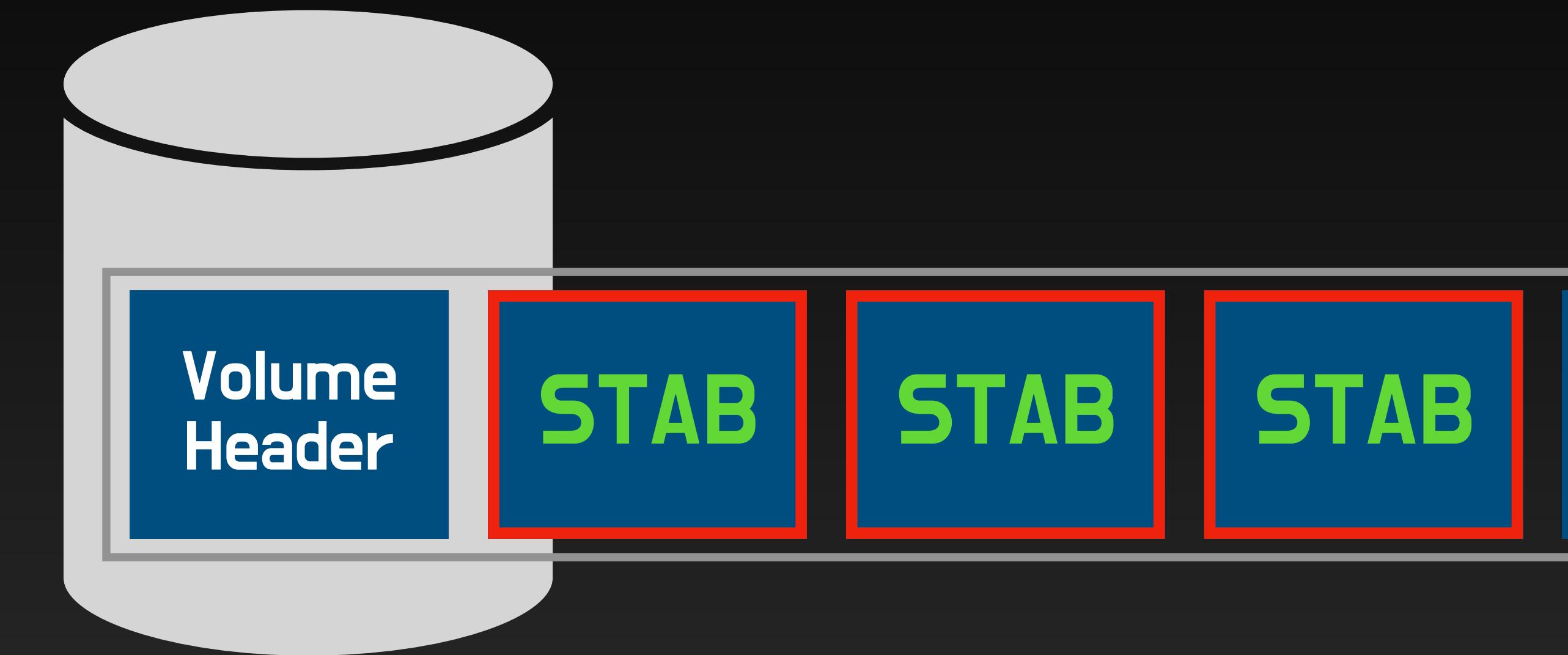


# STAB

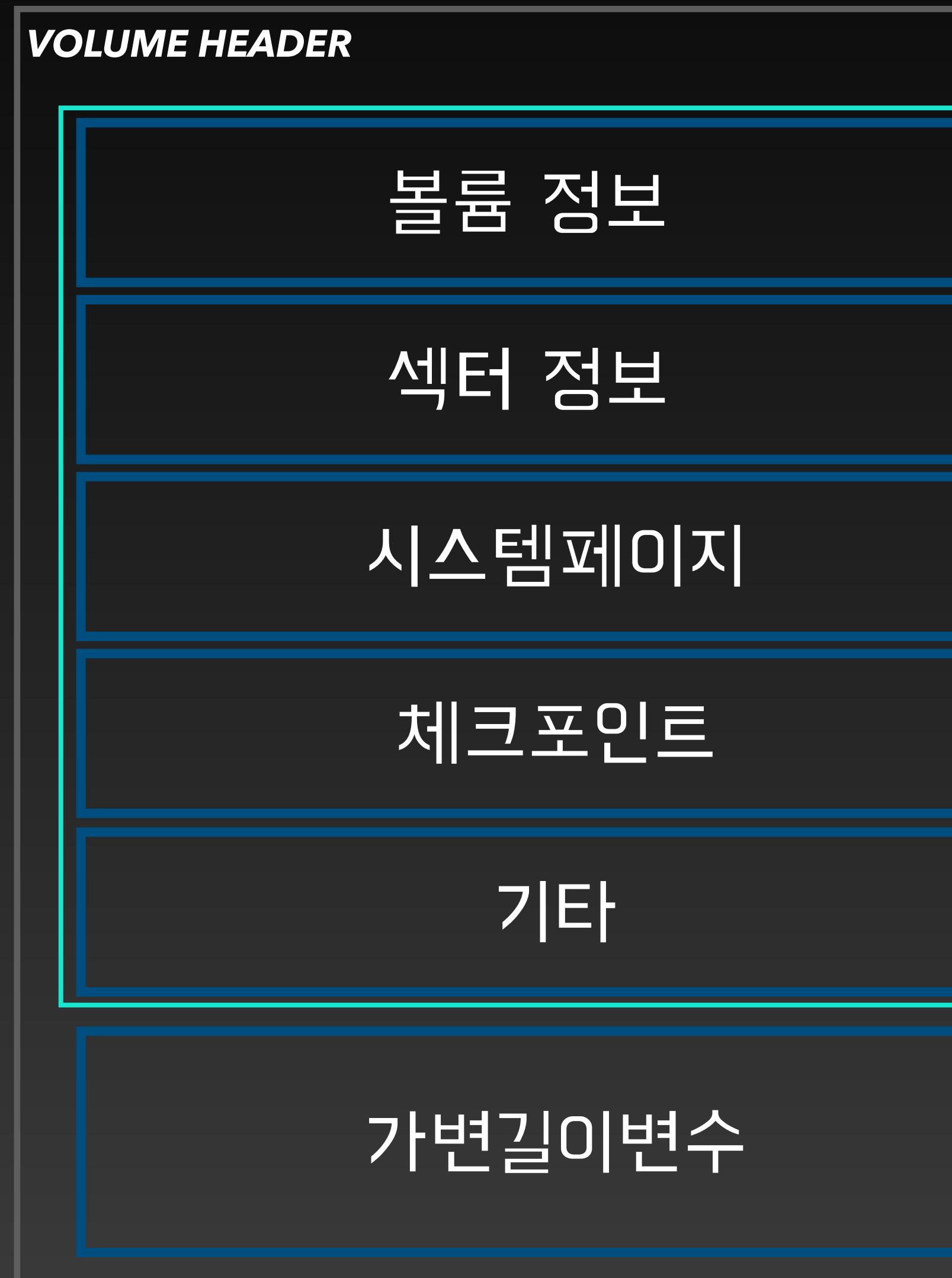
## Volume Header



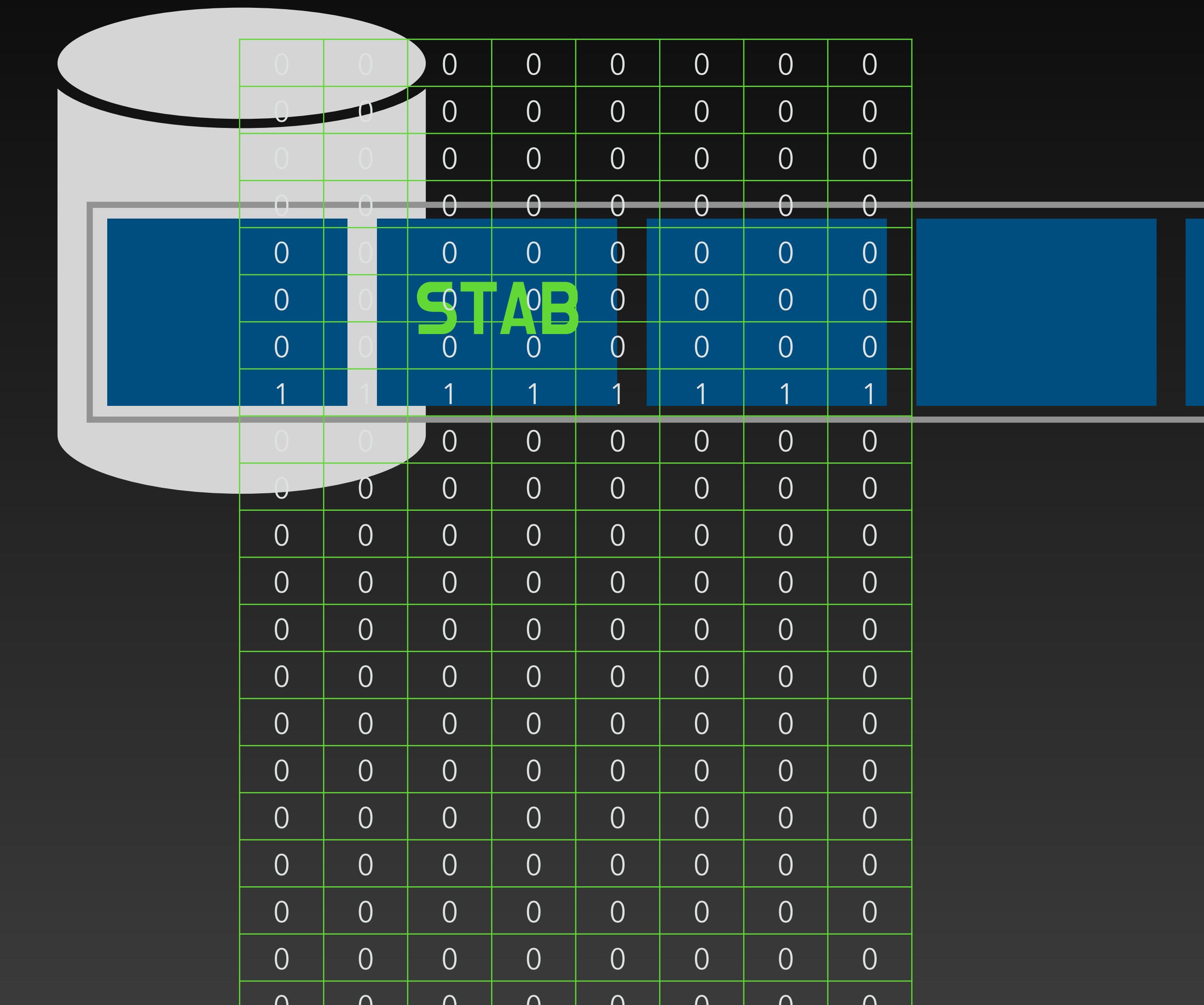
**STAB**



# Volume Header



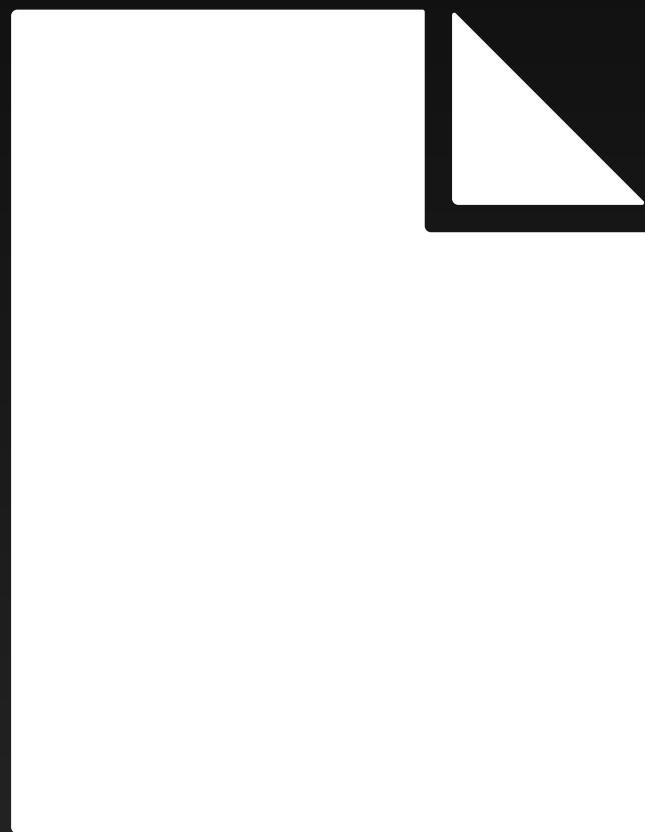
# STAB





# Sector Reservation STRUCT

**DISK\_RESERVE\_CONTEXT**



**DISK\_CACHE**



**DISK\_EXTEND\_INFO**

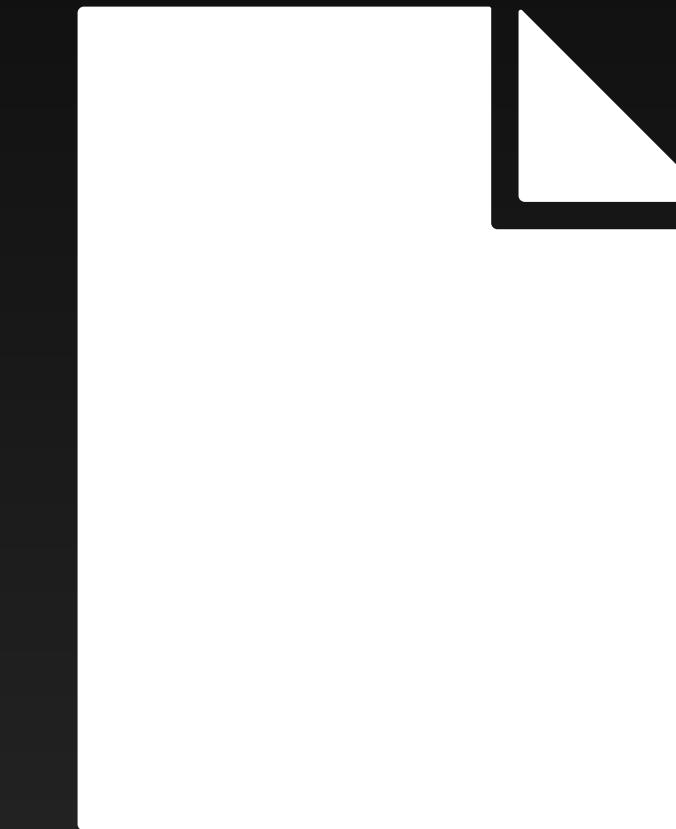


# Sector Reservation STRUCT

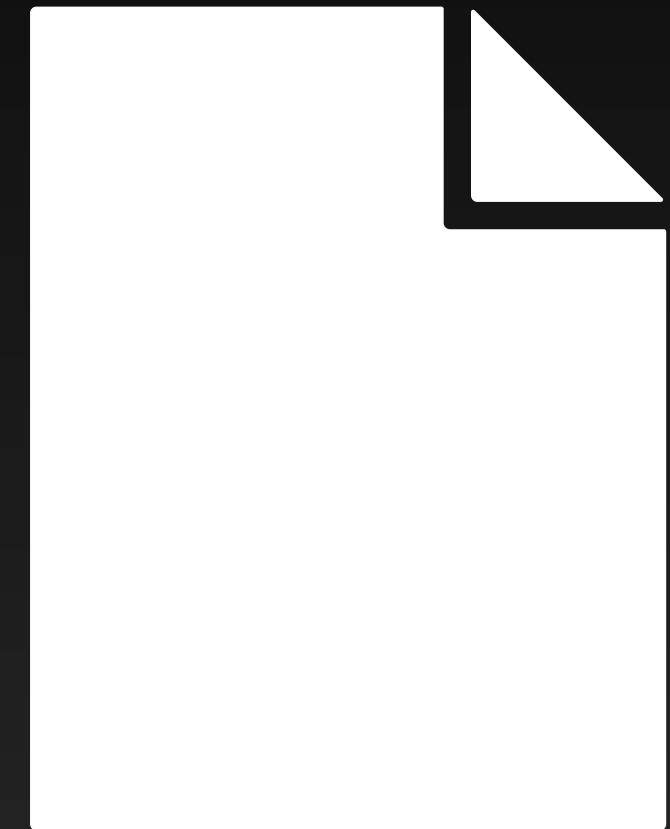
**DISK\_RESERVE\_CONTEXT**



**DISK\_CACHE**



**DISK\_EXTEND\_INFO**



## 요청 진행 상황

nset\_total : 예약 요청된 섹터수

VSID : 섹터의 배열 (예약과정의 최종산출물로 예약한 섹터들의 위치)

cache\_vol\_reserve : 볼륨별 사전예약 섹터 수(VOLID)의 배열

n\_cache\_vol\_reserve : 사전예약한 섹터들이 속한 볼륨의 수 (cache\_vol\_reserve 배열의 크기)

n\_cache\_reserve\_remaining : 현재 예약중인 볼륨에서의 남은 섹터 예약량을 저장

purpose : 예약 목적

# Sector Reservation STRUCT

**DISK\_RESERVE\_CONTEXT**



**요청 진행 상황**

nvols\_perm : 영구타입 볼륨의 개수

nvols\_temp : 임시타입 볼륨의 개수

disk\_cache\_volinfo : 볼륨별 목적과 가용 섹터 수 (이 볼륨별 가용 섹터를 기준으로 볼륨별 사전예약 수행)

perm\_purpose\_info : 영구목적 볼륨들 전체의 합산 섹터 정보와 확장 관련 정보를 지님

temp\_purpose\_info : 임시목적 볼륨들 전체의 합산 섹터 정보와 확장 관련 정보를 지님

영구목적과는 다르게 nset\_perm\_total/free변수를 추가로 지니고 있음

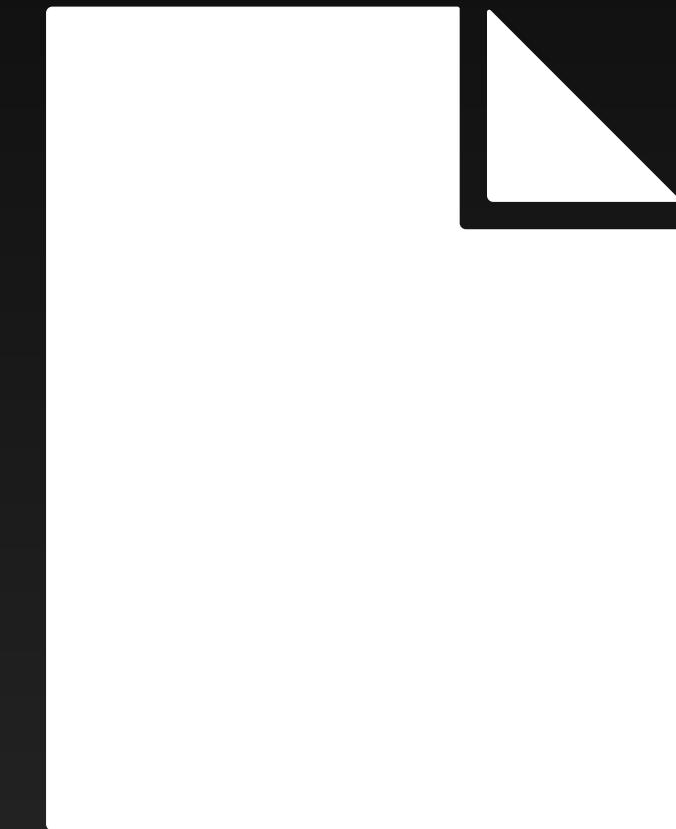
mutex\_extend : 볼륨 확장을 위한 뮤텍스

owner\_extend : mutex\_extend를 잡은 스레드의 아이디

**DISK\_CACHE**



**섹터 예약 정보**



**DISK\_EXTEND\_INFO**

# Sector Reservation STRUCT

**DISK\_RESERVE\_CONTEXT**



## 요청 진행 상황

nset\_free : 볼륨들의 합산 가용 섹터 수  
nset\_total : 볼륨들의 합산 전체 섹터 수  
nset\_max : 볼륨들의 합산 최대 섹터 수  
nset\_intention : 사전예약 시 공간부족으로 공간을 확보하려 할 때, 확보하려는 섹터 수  
(이 값보다는 크게 볼륨을 확장, 추가한다)

mutex\_reserve : 사전예약을 위한 뮤텍스

owner\_reserve : mutex\_reserve 뮤텍스를 잡은 쓰레드 ID

nset\_vol\_max : 볼륨 확장 시 최댓값 (볼륨 생성 시 볼륨 헤더의 nset\_max로 설정)

volid\_extend : 볼륨 확장시 auto\_extend 대상이 되는 볼륨  
(하나의 볼륨의 최댓값까지 확장되어야 새로운 볼륨이 생성되므로, 항상 마지막에 생성된 볼륨을 가리킨다)

voltype : 볼륨 타입

**DISK\_CACHE**



## 섹터 예약 정보

**DISK\_EXTEND\_INFO**

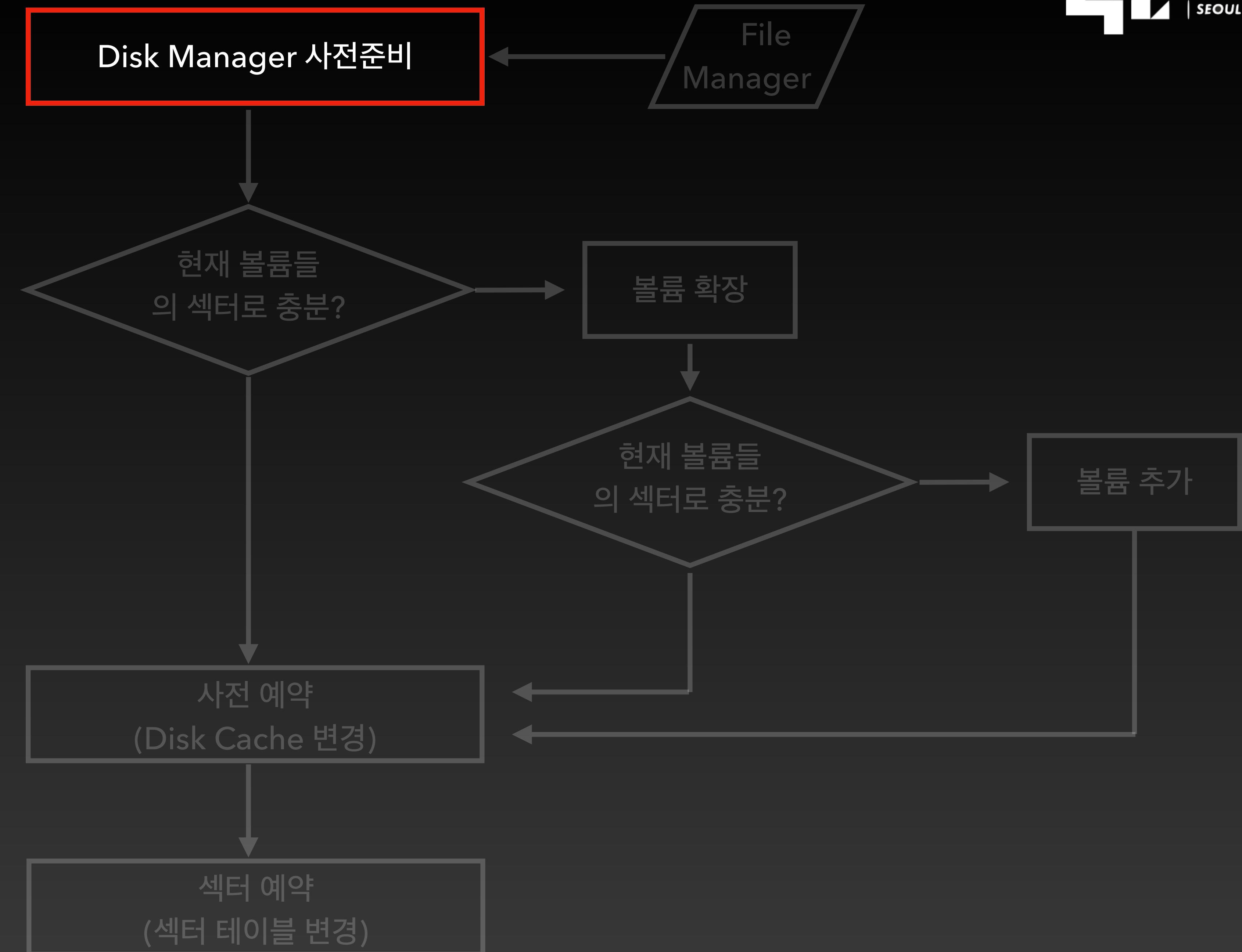


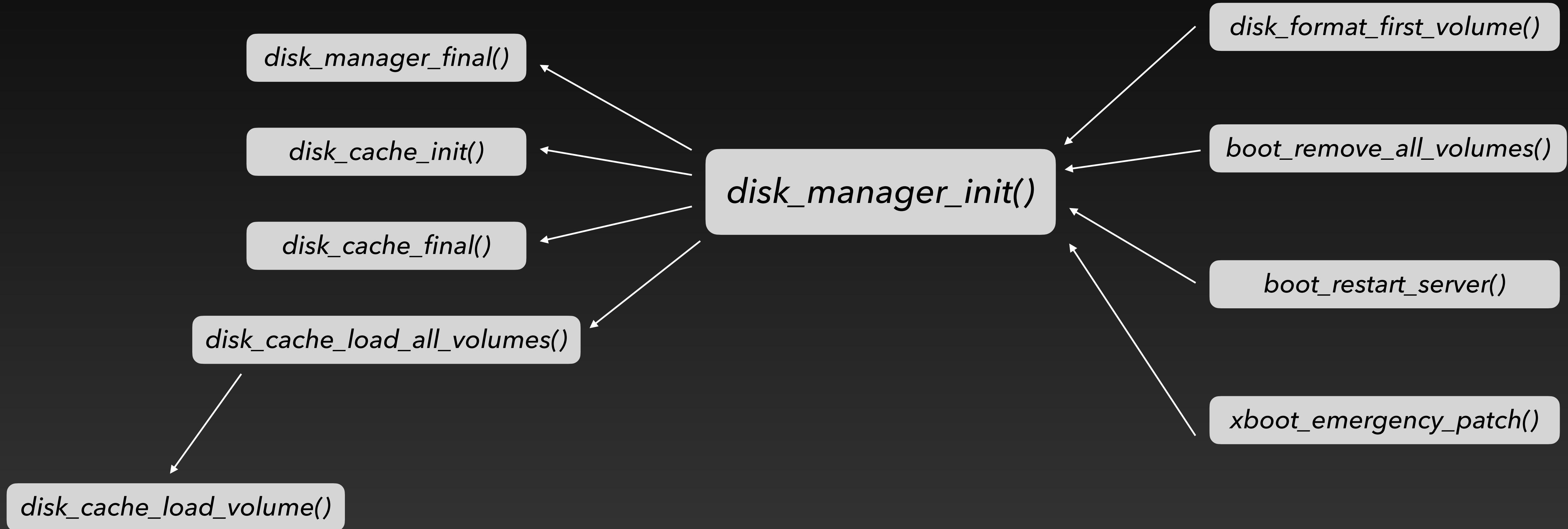
## 전체 볼륨들의 섹터 정보

# 사전 준비 (disk\_manager\_init)

by jolim

# Step 1 사전 준비





# 디스크 캐시 초기화 과정

멤버 변수의 초기화: disk\_cache\_init()

```
struct disk_cache
{
    int nvols_perm;           // 영구타입 볼륨의 수 ← 0
    int nvols_temp;           // 임시타입 볼륨의 수
    DISK_CACHE_VOLINFO vols[LOG_MAX_DBVOLID + 1]; // 볼륨 정보 배열

    DISK_PERM_PURPOSE_INFO perm_purpose_info; // 영구 목적 볼륨의 정보
    DISK_TEMP_PURPOSE_INFO temp_purpose_info; // 임시 목적 볼륨의 정보
};
```

# 디스크 캐시 초기화 과정

멤버 변수의 초기화: disk\_cache\_init()

```
typedef struct disk_perm_info DISK_PERM_PURPOSE_INFO;
```

```
struct disk_perm_info
```

```
{
```

```
    DISK_EXTEND_INFO extend_info;
```

```
};
```

```
typedef struct disk_temp_info DISK_TEMP_PURPOSE_INFO;
```

```
struct disk_temp_info
```

```
{
```

```
    DISK_EXTEND_INFO extend_info;
```

```
    DKNSECTS nsect_perm_free;
```

```
    DKNSECTS nsect_perm_total; ← 0
```

```
};
```

# 디스크 캐시 초기화 과정

멤버 변수의 초기화: disk\_cache\_init()

```
typedef struct disk_extend_info DISK_EXTEND_INFO;
struct disk_extend_info
{
    volatile DKNSECTS nsect_free;
    volatile DKNSECTS nsect_total;
    volatile DKNSECTS nsect_max;
    volatile DKNSECTS nsect_intention;
    pthread_mutex_t mutex_reserve;
    VOLID volid_extend;
    DB_VOLTYPE voltype;
};
```

0

pthread\_mutex\_init()

한 볼륨에 가능한 최대 섹터 개수

NULL\_VOLID == -1

DB\_PERMANENT\_VOLTYPE

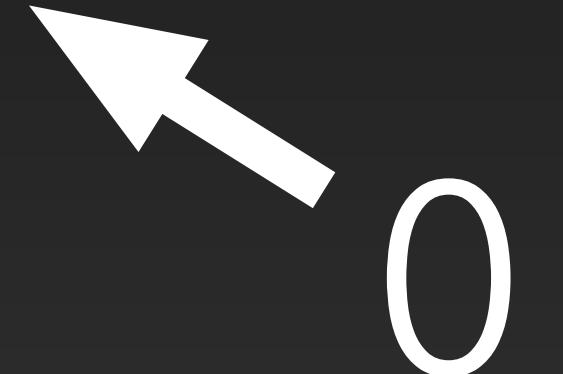
or

DB\_TEMPORARY\_VOL\_TYPE

# 디스크 캐시 초기화 과정

멤버 변수의 초기화: disk\_cache\_init()

```
typedef struct disk_cache_volinfo DISK_CACHE_VOLINFO;
struct disk_cache_volinfo
{
    DB_VOLPURPOSE purpose; ← DISK_UNKNOWN_PURPOSE
    DKNSECTS nsect_free;
};
```

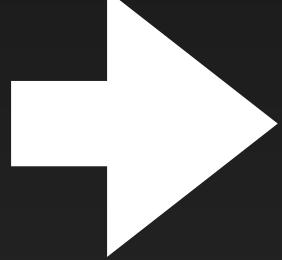


0

# 디스크 캐시 초기화 과정

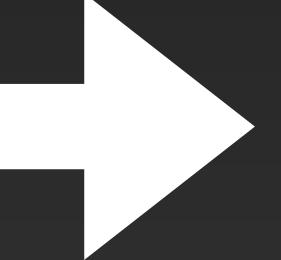
빈 섹터 수의 계산: disk\_volume\_boot()

```
if (볼륨 탑 == 임시 탑)
{
    return NO_ERROR;
}
```

if (볼륨 목적 == 임시 목적) 

시스템 페이지를 제외한 모든 섹터를 빈 섹터로 표시한다

```
{
    disk_stab_init (볼륨 헤더);
    빈 섹터 수 = 전체 섹터 수 - 마지막 시스템 페이지까지의 섹터 수 - 1;
}
```

else // (볼륨 목적 == 영구 목적) 

모든 유닛을 돌며 섹터 테이블에서 비어있는 섹터 수를 직접 센다

```
{
    빈 섹터 수 = 0;
    disk_stab_iterate_units_all (볼륨 헤더, disk_stab_count_free, &빈 섹터 수);
}
```

# 디스크 캐시 초기화 과정

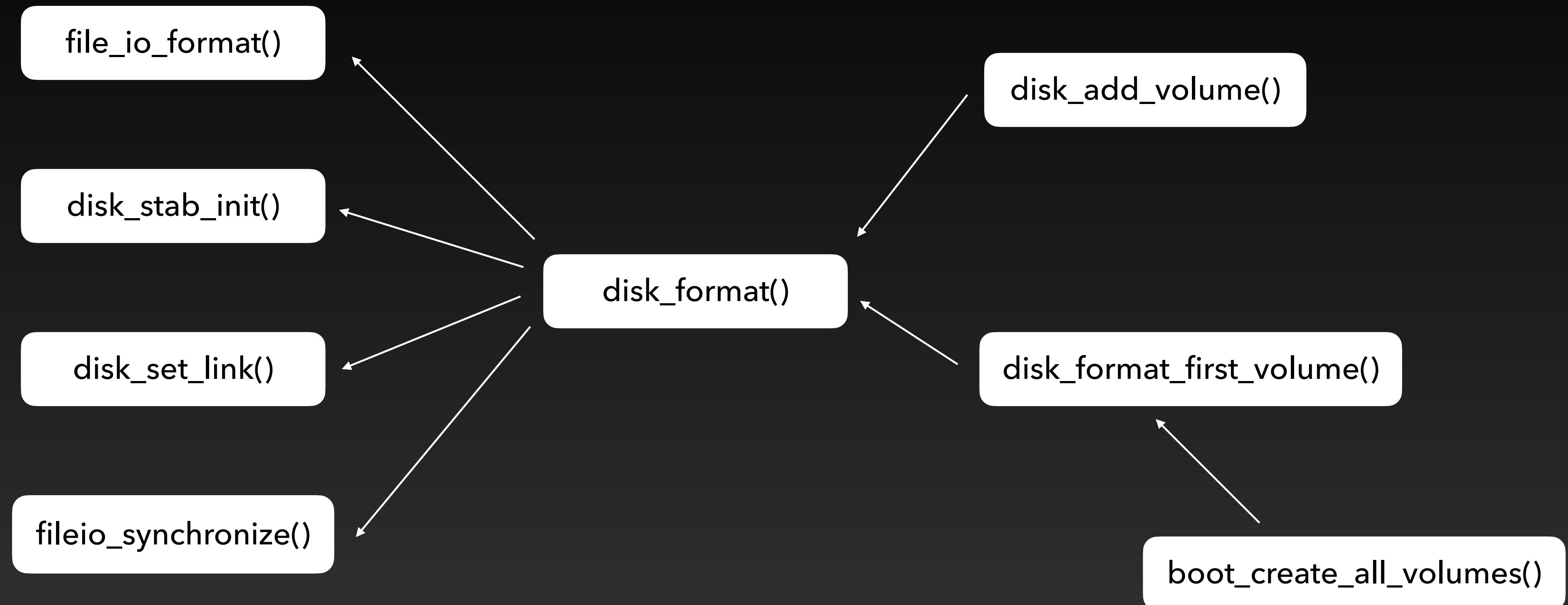
disk\_Cache의 초기화: disk\_cache\_load\_volume()

space\_info: disk\_volume\_boot()에서 얻어낸 하나의 볼륨의 빈 섹터 정보

```
disk_Cache->perm_purpose_info.extend_info.nsect_free += space_info.n_free_sects;  
disk_Cache->perm_purpose_info.extend_info.nsect_total += space_info.n_total_sects;  
disk_Cache->perm_purpose_info.extend_info.nsect_max += space_info.n_max_sects;
```

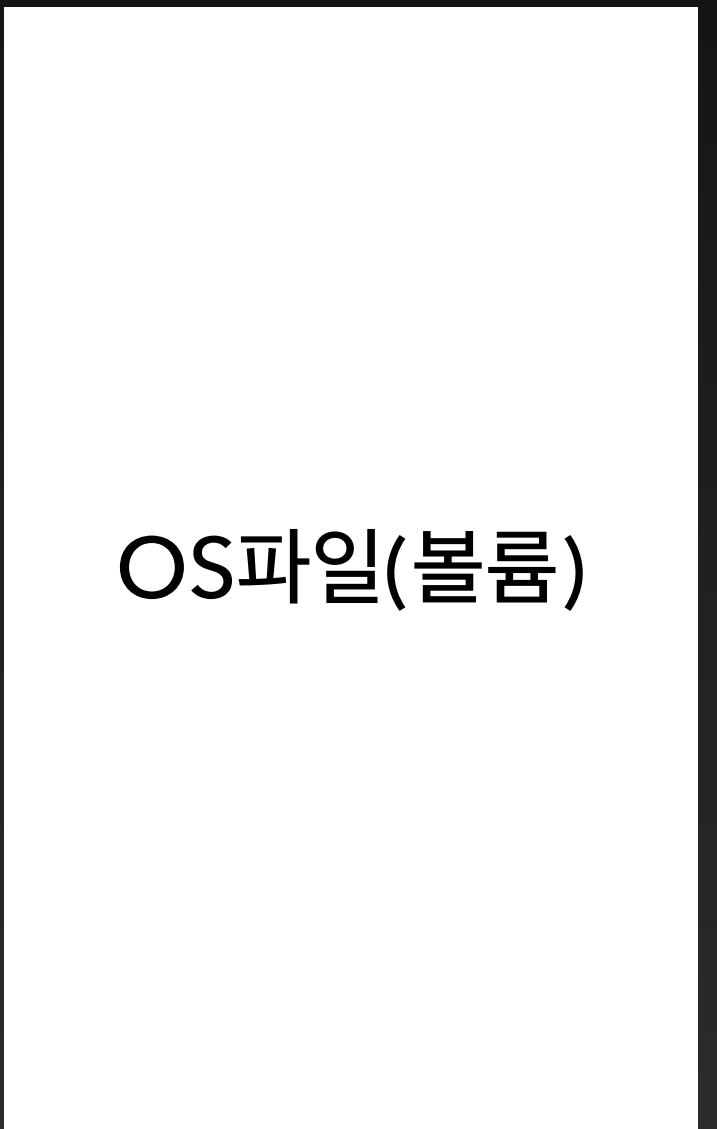
```
disk_Cache->vols[volid].nsect_free = space_info.n_free_sects;  
disk_Cache->vols[volid].purpose = vol_purpose;
```

```
disk_Cache->nvols_perm++;
```



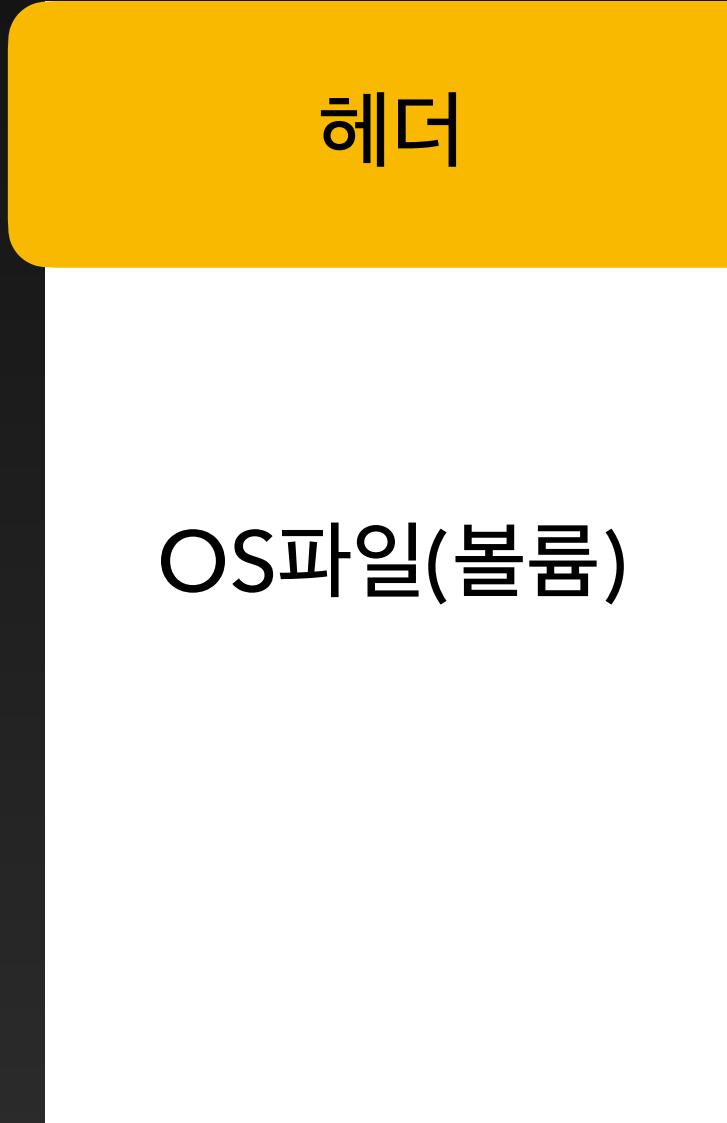
# 디스크 포맷 과정

file\_io\_format()  
1. OS파일 생성



OS파일(볼륨)

2. 볼륨 헤더 초기화



헤더

OS파일(볼륨)

# 디스크 포맷 과정

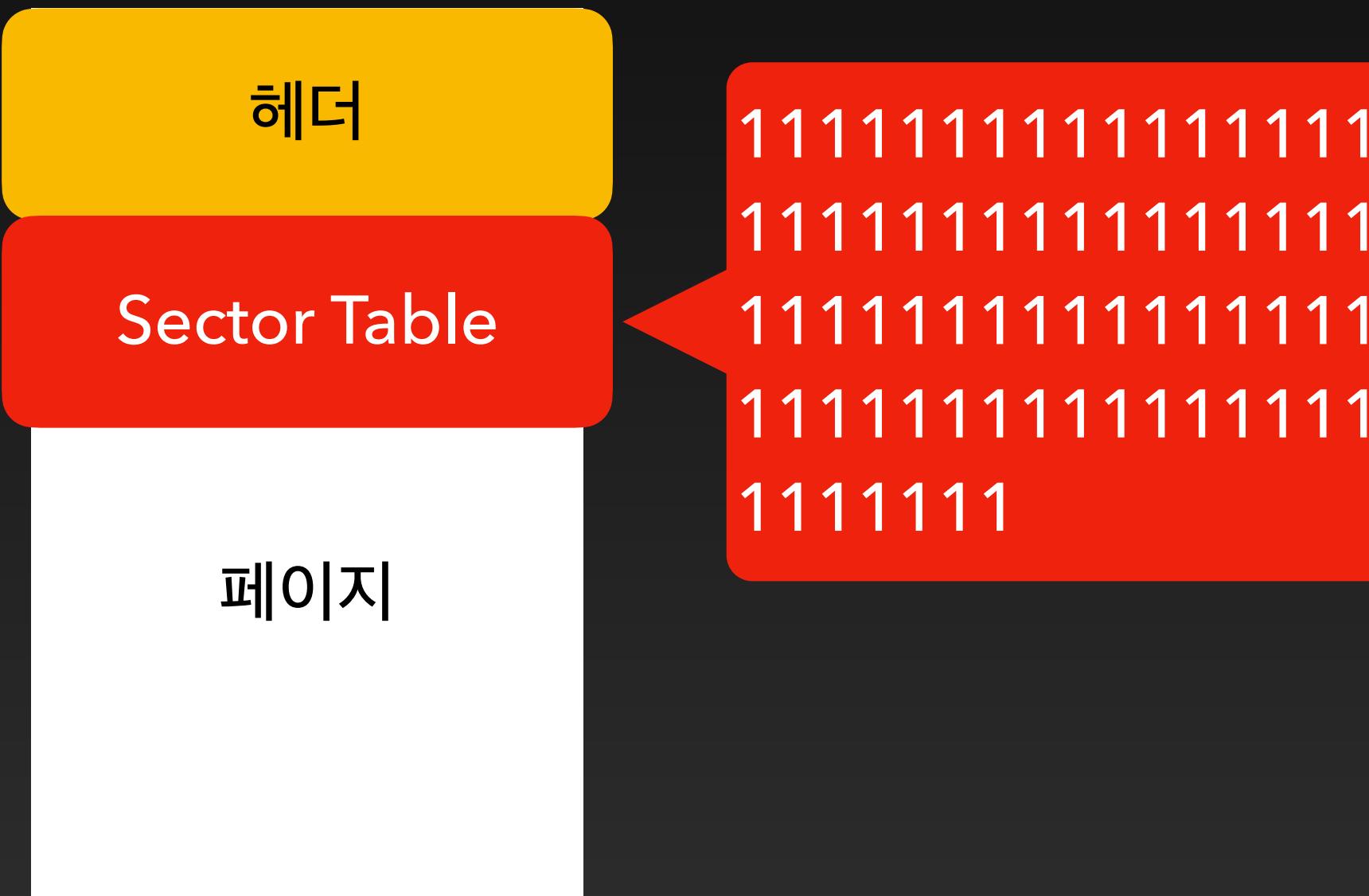
## 3. Variable length field 설정



# 디스크 포맷 과정

disk\_stab\_init()

## 4. Sector table 초기화



현재 페이지 수 만큼의 비트를 1로 채운다

# 디스크 포맷 과정

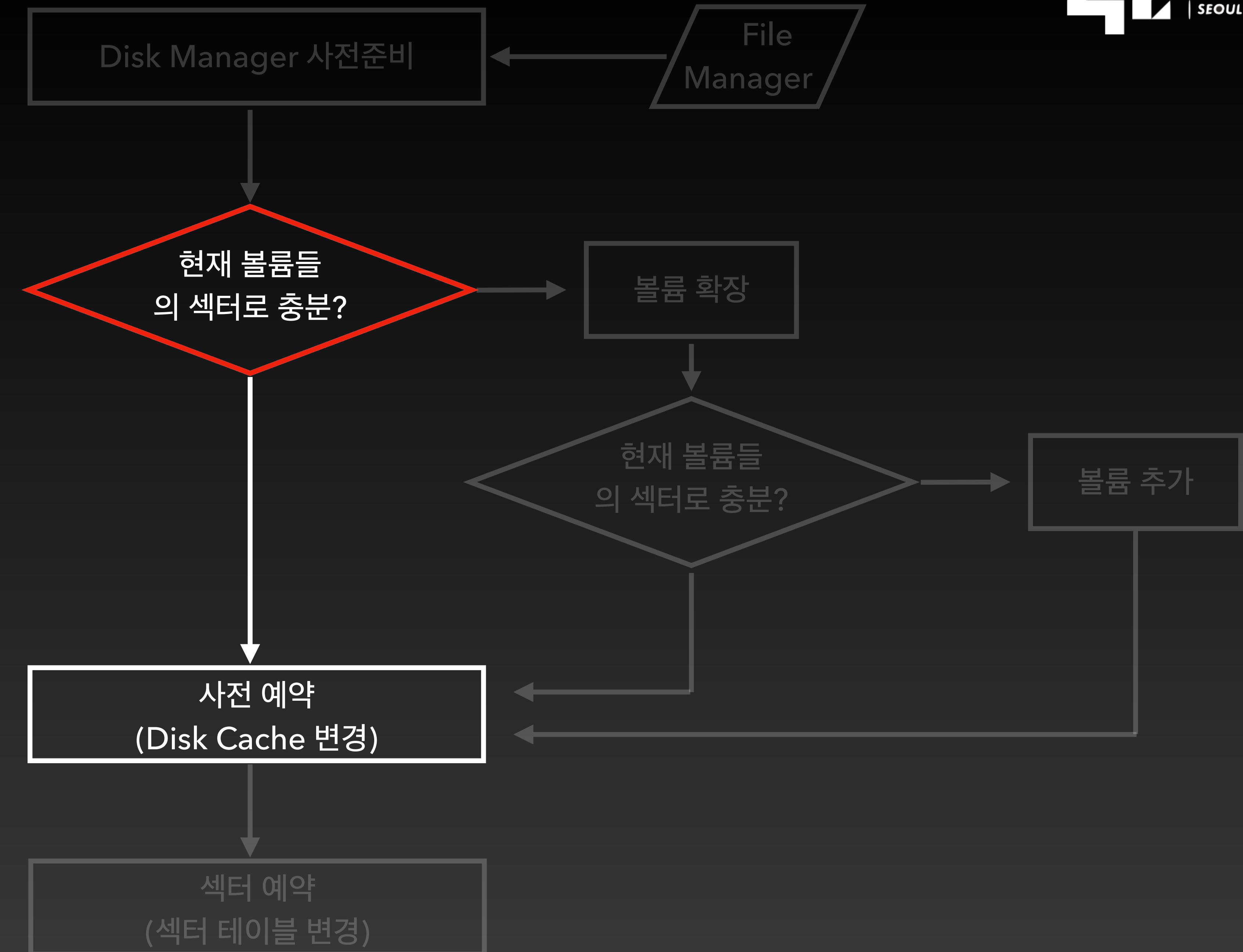


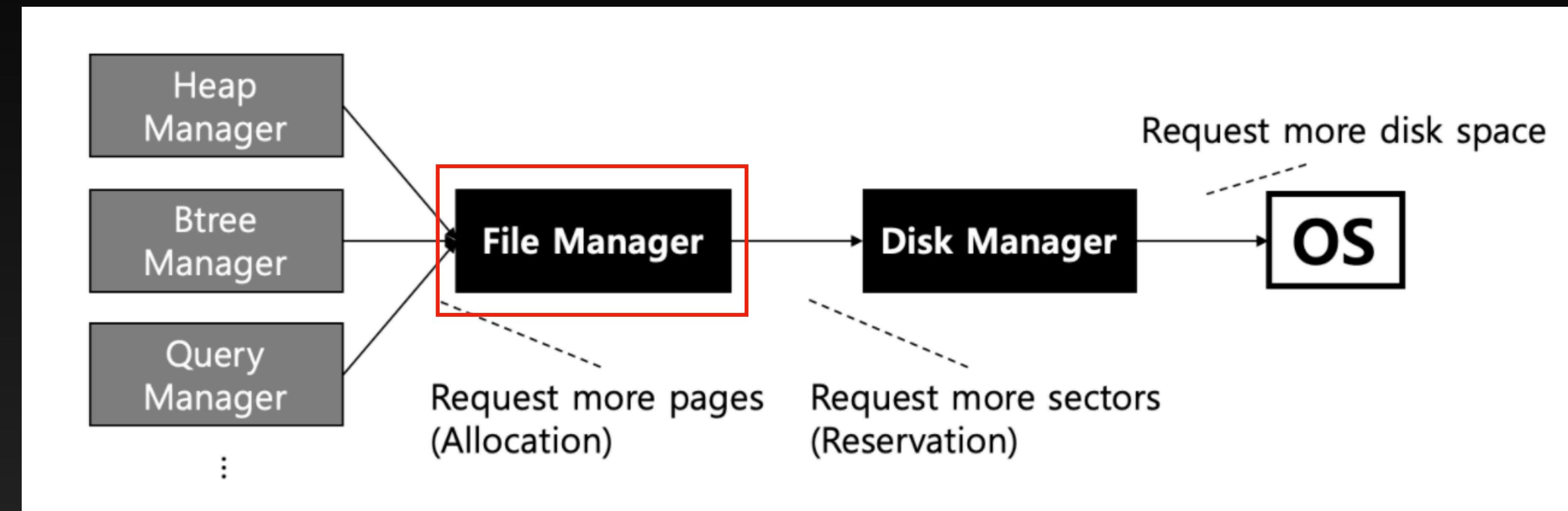
사전 예약

(disk\_reserve\_sectors)

by seubaek

# Step 1 사전 예약

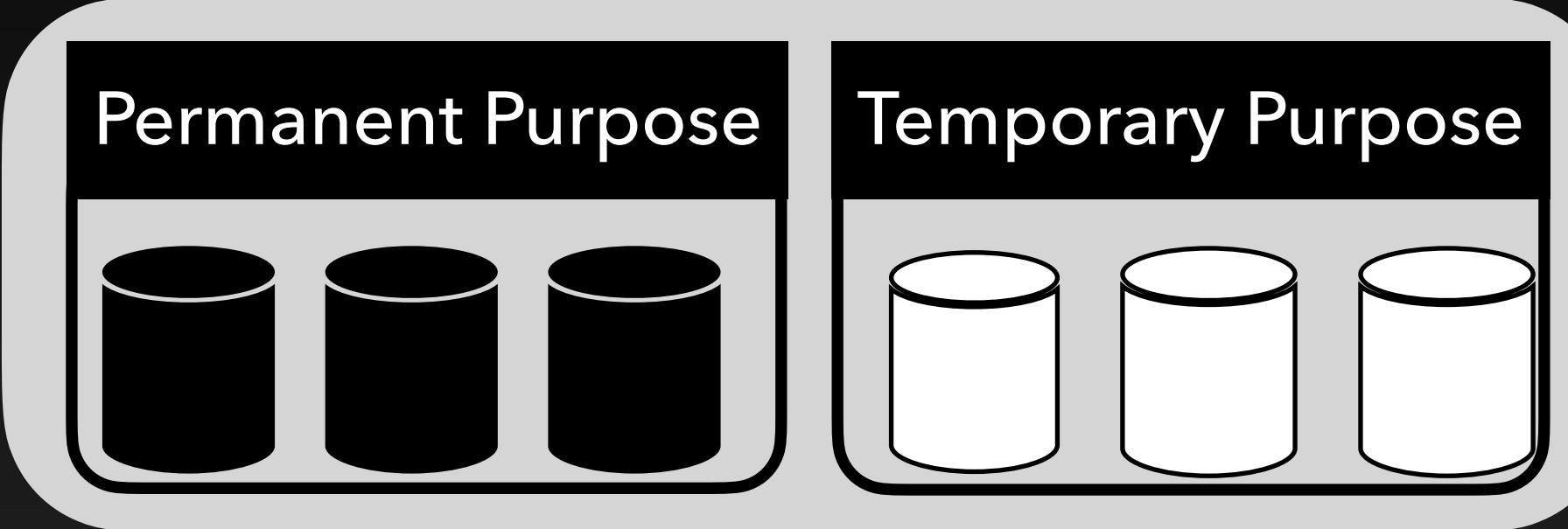




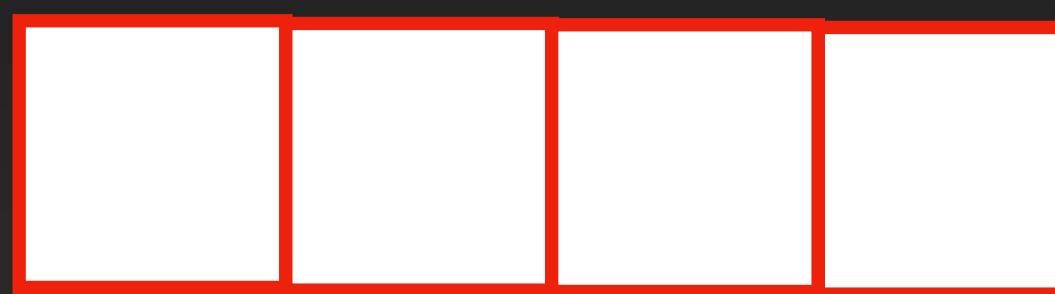
1. 새 파일 생성하는 경우 ( `file_create` )
2. 영구 파일에 섹터를 추가하는 경우 ( `file_perm_expand` )
3. 임시 파일에 새 페이지를 할당하는 경우 ( `file_temp_alloc` )

```
int
```

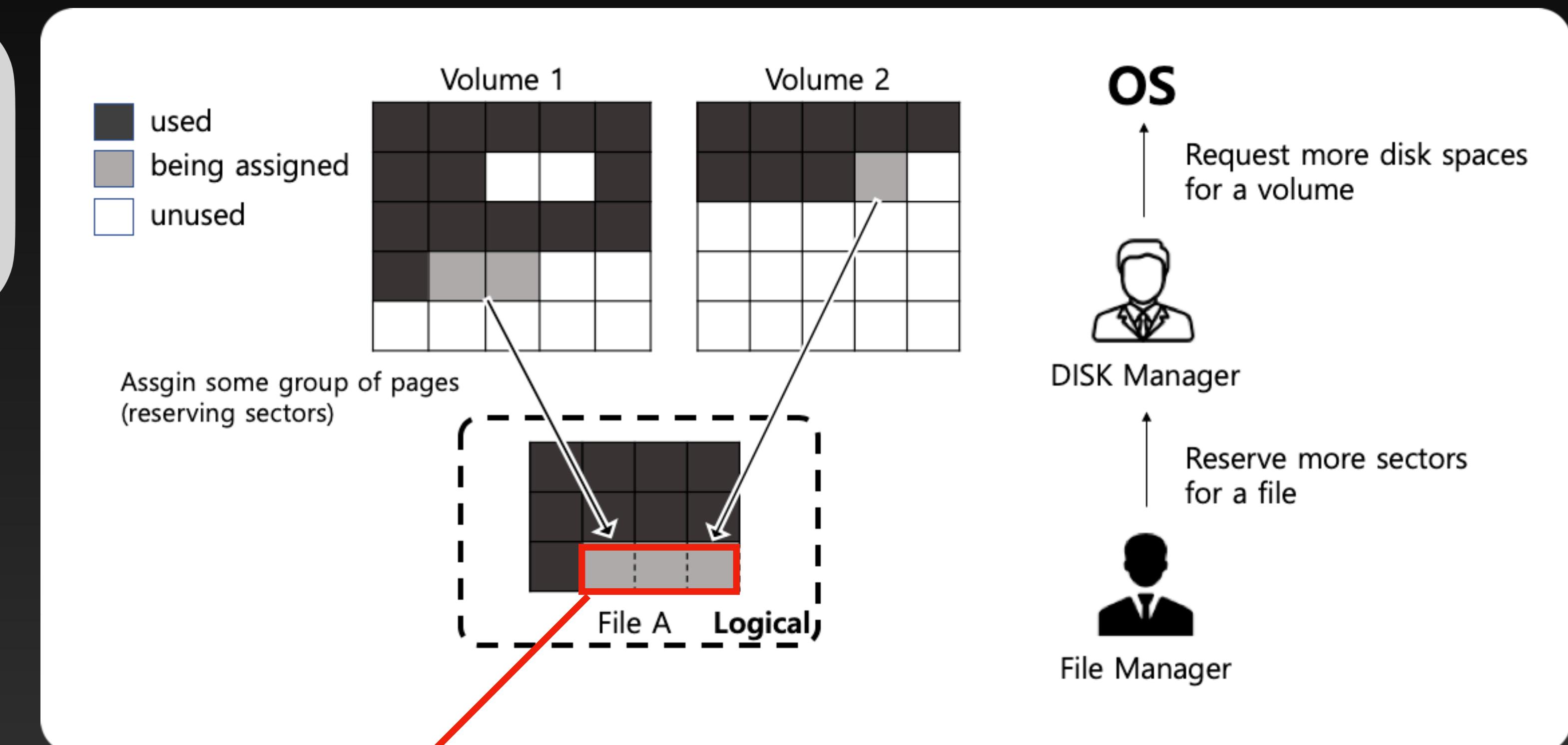
```
disk_reserve_sectors (THREAD_ENTRY * thread_p, DB_VOLPURPOSE purpose, VOLID volid_hint, int n_sectors,  
[| | | | | VSID * reserved_sectors)
```



어떤 목적의 볼륨인지?



섹터 예약 수 만큼의 빈 배열



몇개의 섹터가 필요한지?

## DISK\_RESERVE\_CONTEXT

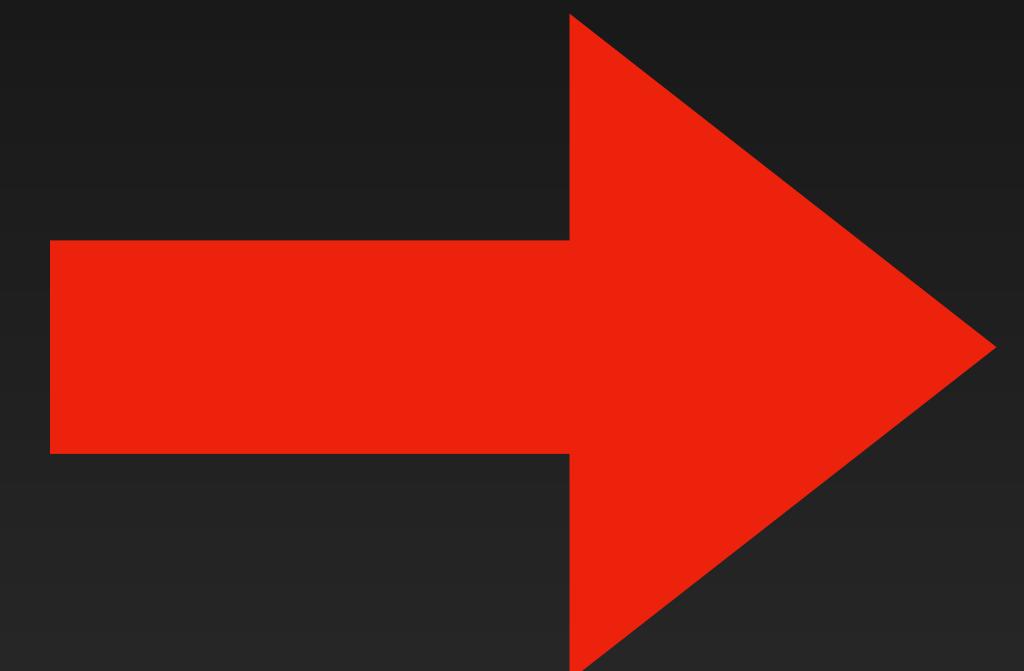


1. 사전 예약에 필요한 섹터의 수
2. 사전 예약한 섹터들이 속한 볼륨의 수
3. 예약한 섹터들의 위치 값이 담긴 배열
4. 남은 섹터량 변수 초기화
5. 볼륨의 목적

```
/* init context */
context.nsect_total = n_sectors;
context.n_cache_reserve_remaining = n_sectors;
context.vsidp = reserved_sectors;
context.n_cache_vol_reserve = 0;
context.purpose = purpose;
```

사전 예약 구조체 안 Value를 바탕으로 디스크 캐시를 초기화

**DISK\_RESERVE\_CONTEXT**



**DISK\_CACHE**



사전 예약 구조체 안 Value를 바탕으로 섹터 테이블을 변경

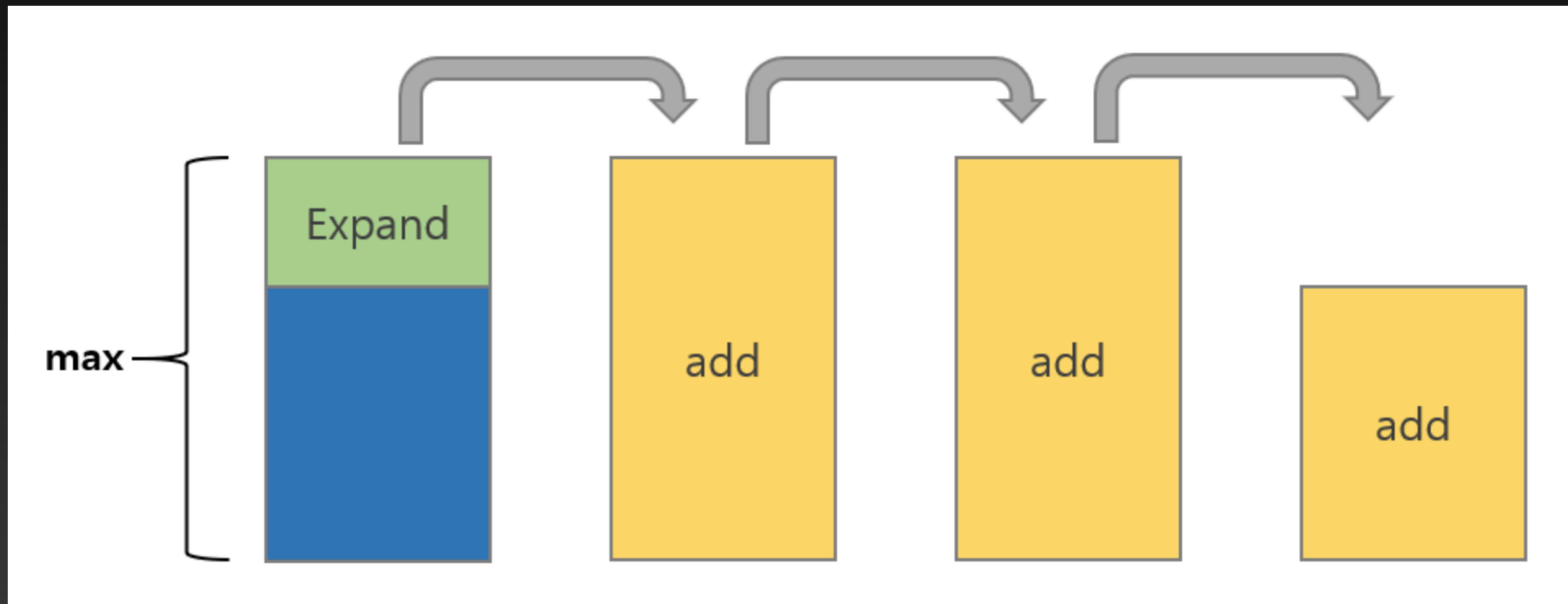
## DISK\_RESERVE\_CONTEXT



잔여 볼륨 별 가용 섹터 내에 예약이 완료되면 예약 과정을 종료



볼륨이 확장되거나 추가될 경우



## 디스크 동기화 체크

```
if (did_extend)
{
    /* safe-guard: catch inconsistencies early */
    assert (disk_check (thread_p, false) != DISK_INVALID);
}
```

볼륨을 확장 및 추가하는 과정에서 디스크 캐시의 확장 정보를 수정  
데이터의 일관성을 보장할 수 없다

## 1st. 디스크 캐시와 boot\_db\_parm 확인

### Boot\_db\_parm



```
nvolz_perm = xboot_find_number_permanent_volumes (thread_p);  
nvolz_temp = xboot_find_number_temp_volumes (thread_p);  
volid_perm_last = xboot_find_last_permanent (thread_p);  
volid_temp_last = xboot_find_last_temp (thread_p);
```

- 디스크에 존재하는 영구, 임시 볼륨의 수
- 디스크에 존재하는 영구, 임시 볼륨의 마지막 주소값

디스크 자체에 볼륨의 수가 맞지 않는 경우  에러 레이블로 이동

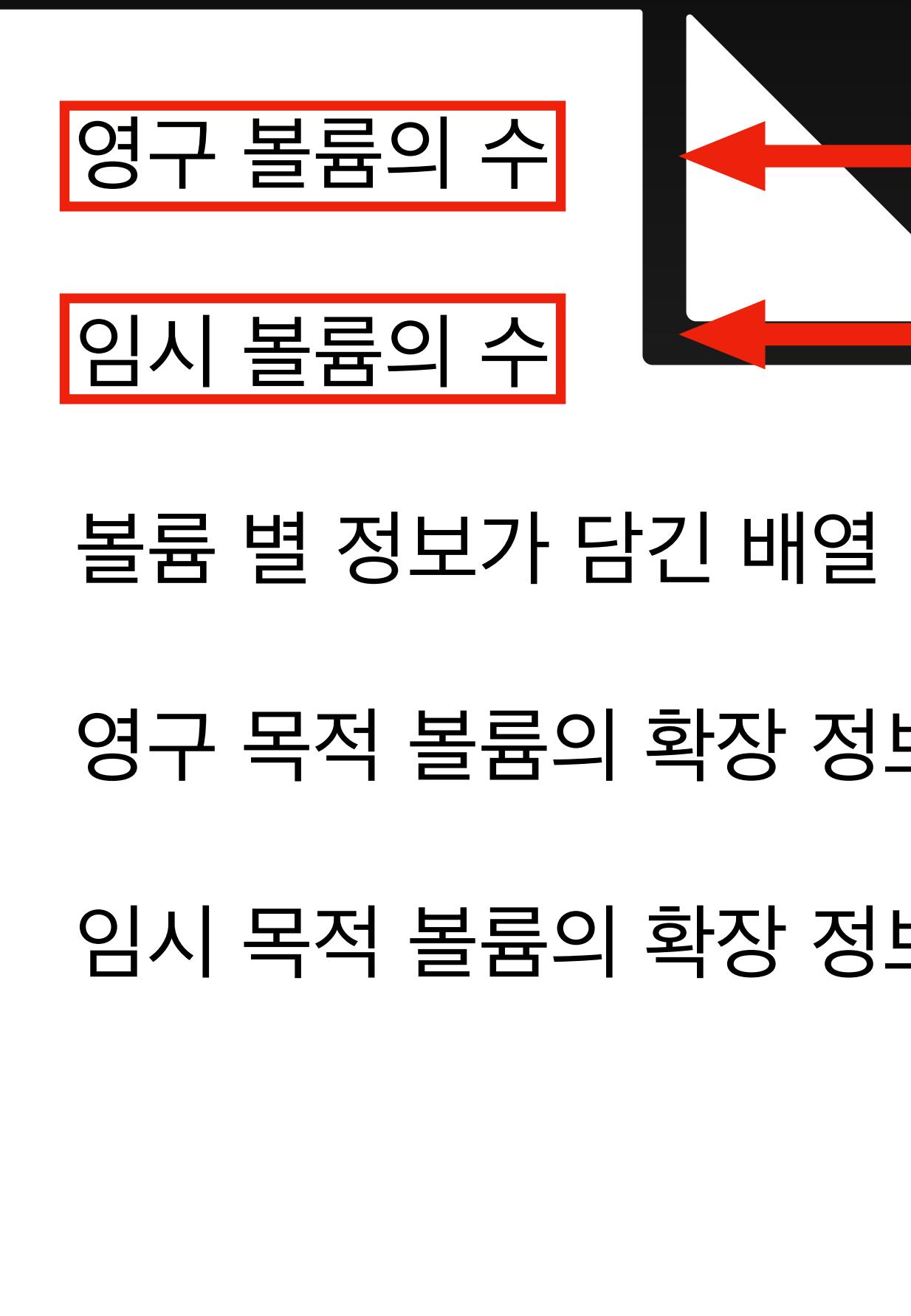
```
if (nvols_perm != volid_perm_last + 1)
{
    /* cannot repair */
    assert_release (false);
    csect_exit (thread_p, CSECT_DISK_CHECK);
    return DISK_INVALID;
}

if (nvols_temp > 0 && (nvols_temp != (LOG_MAX_DBVOLID - volid_temp_last + 1)))
{
    /* cannot repair */
    assert_release (false);
    csect_exit (thread_p, CSECT_DISK_CHECK);
    return DISK_INVALID;
}
```

## DISK\_CACHE



에러 레이블로 이동



볼륨 별 정보가 담긴 배열

영구 목적 볼륨의 확장 정보

임시 목적 볼륨의 확장 정보

디스크 캐시의 볼륨 별 개수와  
실질적인 볼륨 별 개수를 비교

nvols\_perm

nvols\_temp

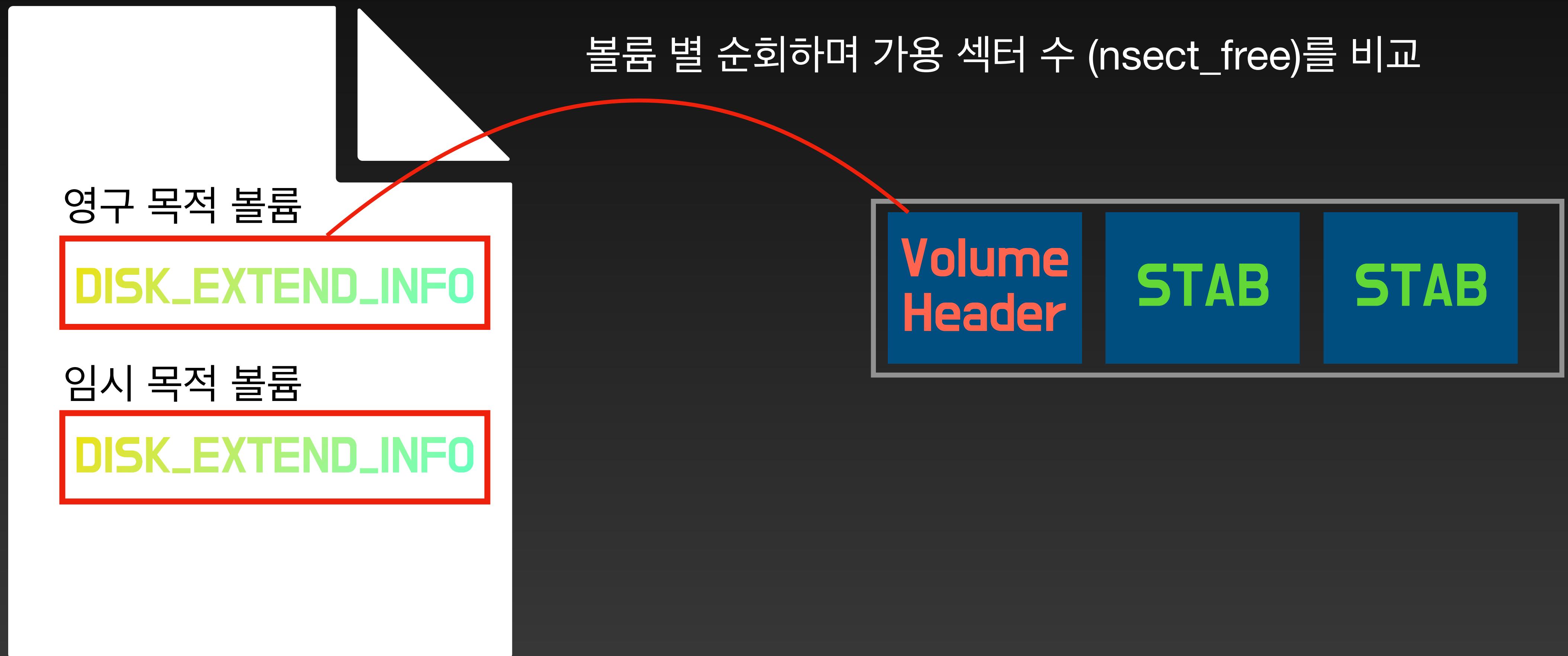
## 2nd. 디스크 캐시와 섹터 테이블의 일관성 확인

### DISK\_CACHE



에러 레이블로 이동

볼륨 별 순회하며 가용 섹터 수 (nsect\_free)를 비교



## ERROR 상황 발생 시 ( 디스크 캐시 )

사전 예약 구조체의 볼륨 별 사용 섹터 수를 바탕으로 디스크 캐시의 가용 섹터 수를 초기화

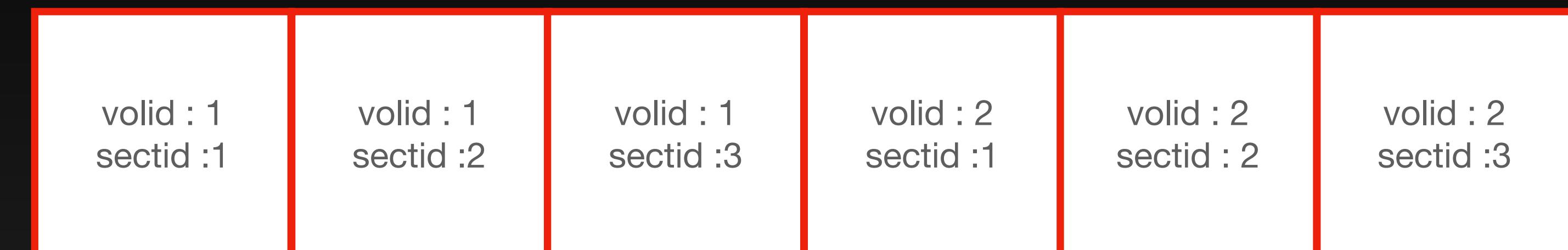
**DISK\_RESERVED\_CONTEXT**



**DISK\_CACHE**



## ERROR 상황 발생 시 ( 영구 볼륨 )



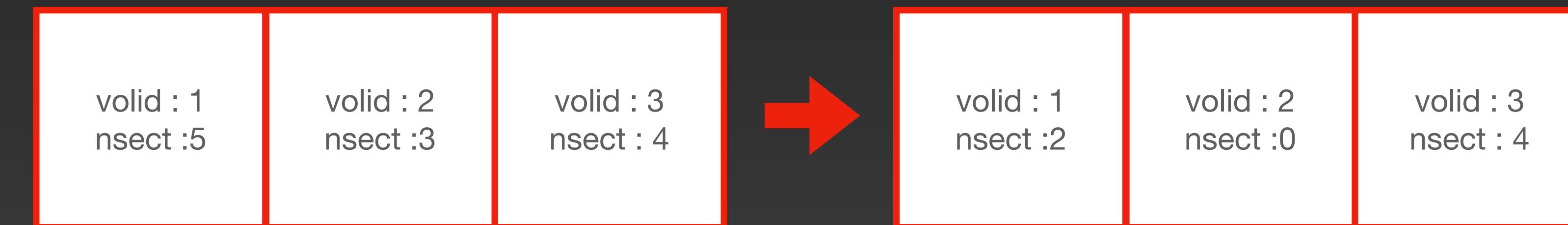
예약이 진행된 섹터 배열



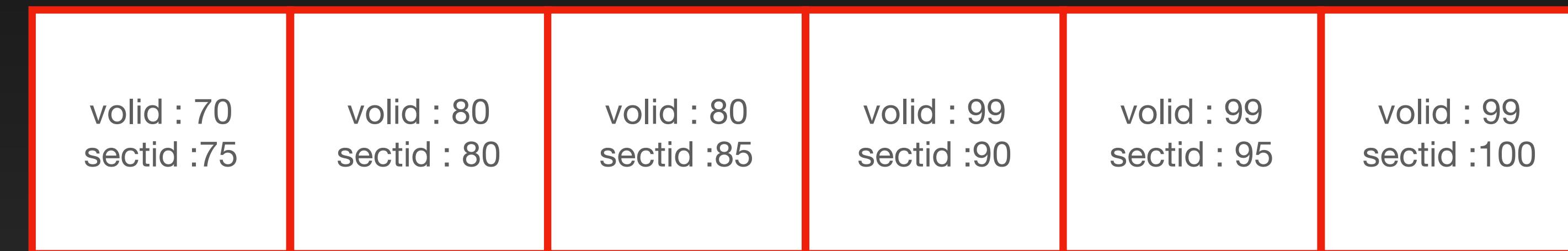
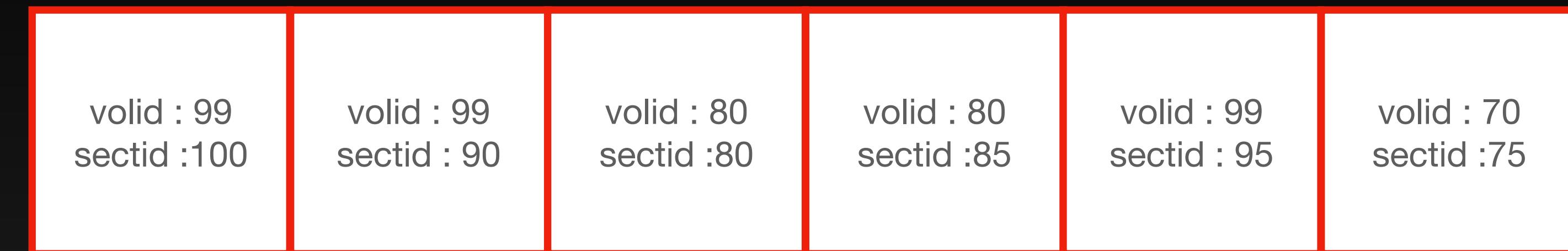
이중 루프를 순회

이미 디스크에 예약이 진행  
된 섹터의 수를 감소 진행

```
context.cache_vol_reserve[iter].nsect--;
```



## ERROR 상황 발생 시 ( 임시 볼륨 )



이중 루프를 순회

이미 디스크에 예약이 진행  
된 섹터의 수를 감소 진행

```
context.cache_vol_reserve[iter].nsect--;
```

## ERROR 상황 발생 시 ( 디스크 동기화 )

강제적 동기화를 진행

```
if (disk_check (thread_p, true) == DISK_INVALID)
{
    /* oh, it was bad */
    error_code = NO_ERROR;
    er_clear ();
    retried = true;
    goto retry;
}
```

## 1st. 디스크 캐시와 boot\_db\_parm 확인

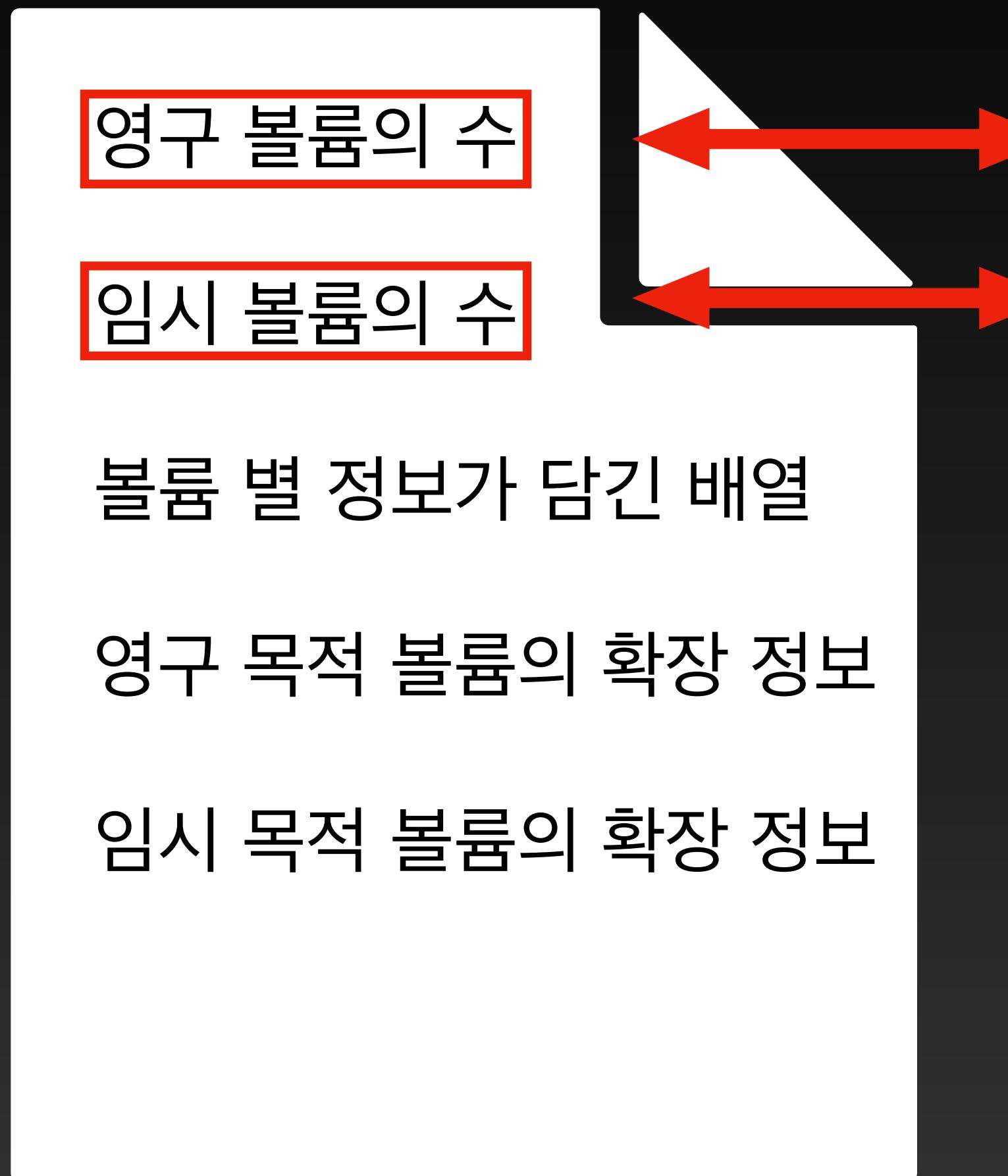
### Boot\_db\_parm



```
nvols_perm = xboot_find_number_permanent_volumes (thread_p);  
nvols_temp = xboot_find_number_temp_volumes (thread_p);  
volid_perm_last = xboot_find_last_permanent (thread_p);  
volid_temp_last = xboot_find_last_temp (thread_p);
```

- 디스크에 존재하는 영구, 임시 볼륨의 수
- 디스크에 존재하는 영구, 임시 볼륨의 마지막 주소값

## DISK\_CACHE



nvols\_perm

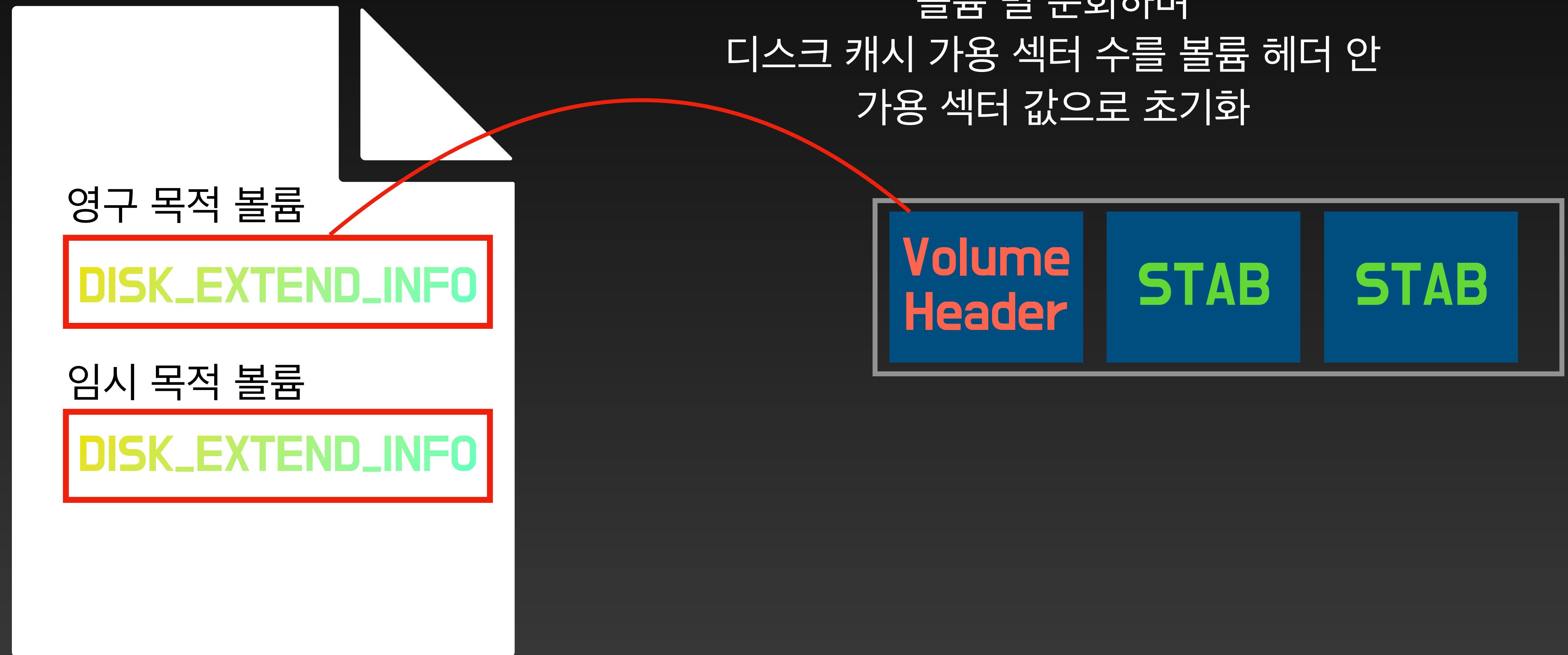
nvols\_temp

강제 동기화

```
|   |   if (repair)
|   {
|       disk_Cache->nvols_perm = nvols_perm;
|   }
|
|   |   if (repair)
|   {
|       disk_Cache->nvols_temp = nvols_temp;
|   }
```

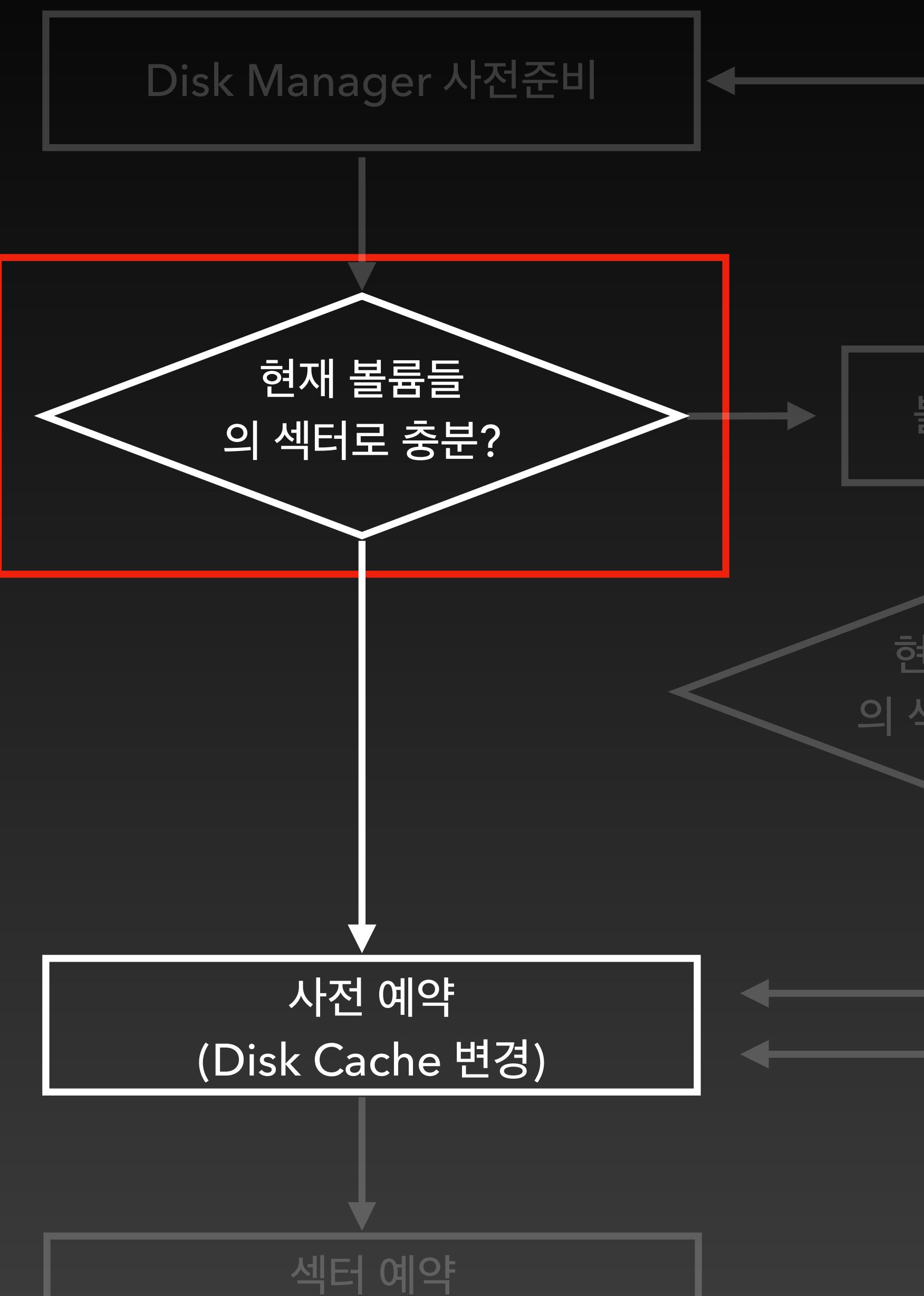
## 2nd. 디스크 캐시와 섹터 테이블의 일관성 확인

### DISK\_CACHE



동기화 과정에서 INVALID 값을 리턴하게 되면  
사전 예약 과정을 처음부터 다시 진행

```
retry:  
    // 디스크 로그에 기록  
    disk_log ("disk_reserve_sectors", "reserve %d sectors for %s.", n_sectors, disk_purpose_to_string (purpose));  
  
    // 로그 시스템 시작  
    log_sysop_start (thread_p);  
  
    /* we don't want to conflict with disk check */  
    // 크리티컬 섹션 진입을 위한 reader Lock을 획득  
    error_code = csect_enter_as_reader (thread_p, CSECT_DISK_CHECK, INF_WAIT);  
    // 에러 코드를 뱉으면 로그 시스템 정지  
    if (error_code != NO_ERROR)  
    {  
        ASSERT_ERROR ();  
        log_sysop_abort (thread_p);  
        return error_code;  
    }  
  
    // 사전예약 구조체 초기화  
    /* init context */  
    context.nsect_total = n_sectors;  
    context.n_cache_reserve_remaining = n_sectors;  
    context.vsidp = reserved_sectors;  
    context.n_cache_vol_reserve = 0;  
    context.purpose = purpose;
```

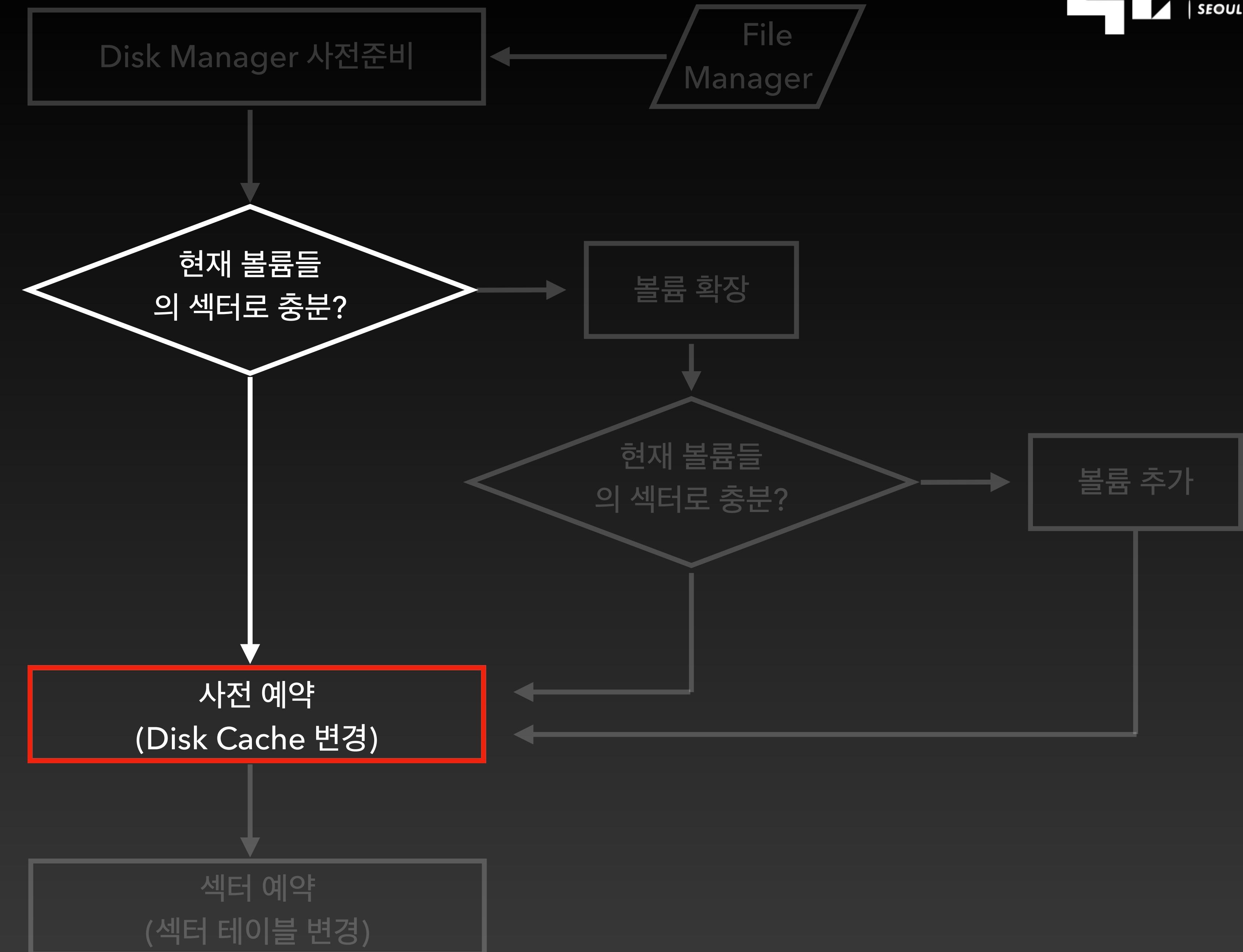


사전 예약

(disk\_reserve\_from\_cache)

by jseo

# Step 1 사전 예약



# 사전 예약

디스크의 섹터를 예약하여 STAB에 기록하기 이전에 수행

예약하려는 섹터들을 어떤 볼륨으로부터 얼마나 예약이 가능한지 맥락을 파악하는 과정

요구되는 섹터들이 현재 볼륨 만으로 충분하지 않다면, 사전 예약 과정에서 볼륨을 확장 혹은 추가하는 과정을 거침

# 사용되는 구조체

**DISK\_RESERVE\_CONTEXT**



요청 진행 상황

**DISK\_CACHE**



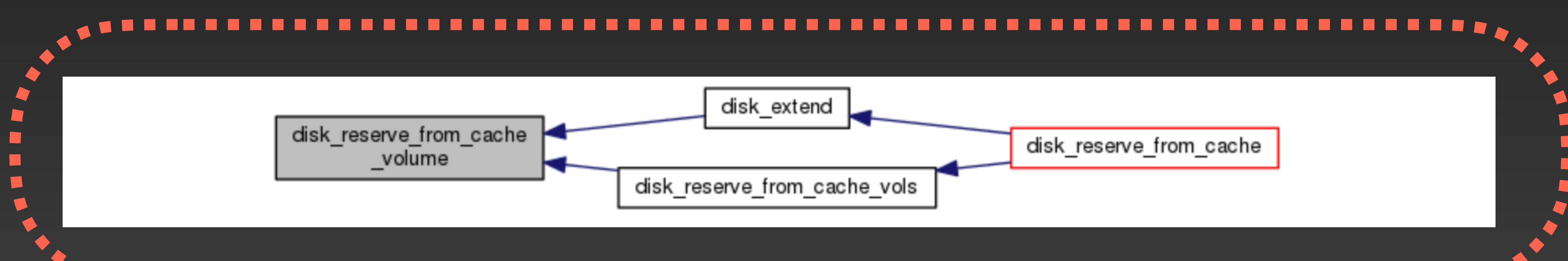
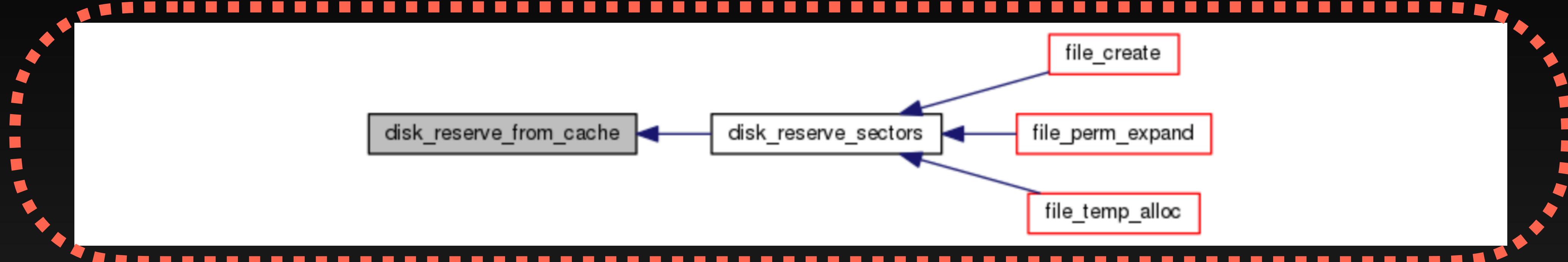
섹터 예약 정보

**DISK\_EXTEND\_INFO**



전체 볼륨들의  
섹터 정보

# 호출 흐름

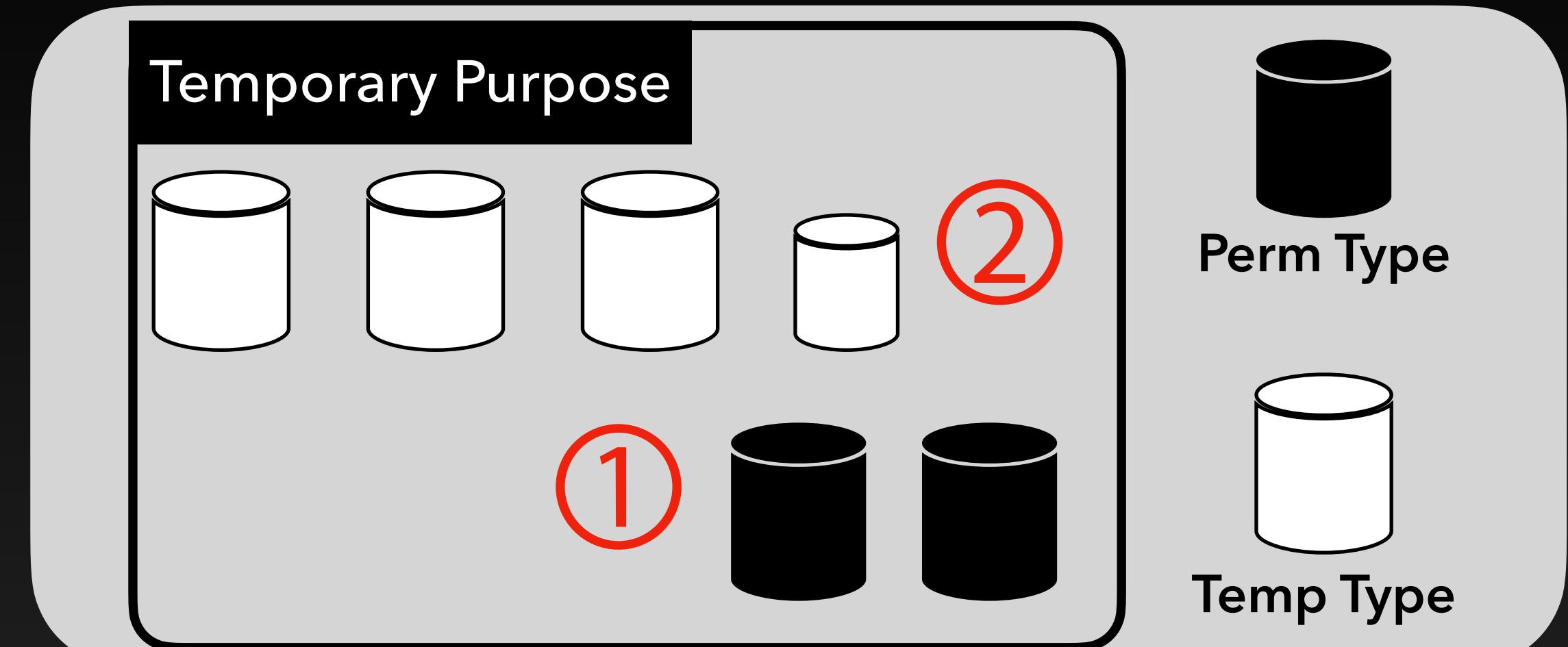


# 흐름 세부 사항 ①

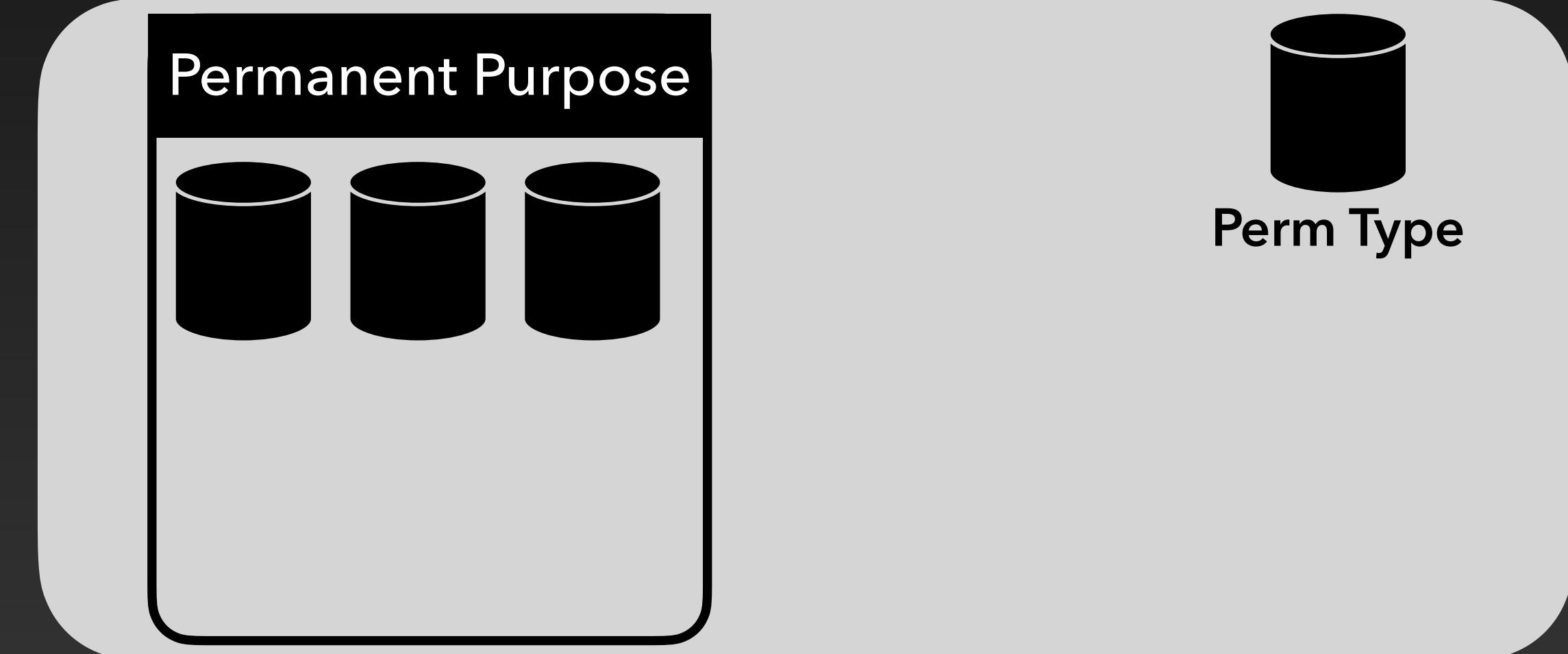
- 사전 예약 진행 이전에 예약을 진행하려는 맥락 (context)는 파악이 완료되어 있음
- 사전 예약은 디스크 캐쉬에 파악된 맥락을 반영하는 과정을 거침
- 디스크 매니저에서 단일로 존재하는 디스크 캐쉬가 반드시 초기화 되어 있어야 함
- 디스크 캐쉬가 초기화되어 존재한다면, 사전 예약을 진행하려는 볼륨 타입을 결정

# 흐름 세부 사항 ②

- 예약의 목적이 Temporary의 경우
  1. Permanent 탑입 볼륨 우선 진행 결정
  2. Temporary 탑입 볼륨 예약 진행 결정



- 예약 목적이 Permanent의 경우  
Permanent 탑입 볼륨 진행 결정



## 흐름 세부 사항 ③

- 예약을 진행하려는 목적에 따라 DISK\_EXTEND\_INFO가 결정되었다면, 예약을 바로 진행할 수 있는지 확인
- DISK\_EXTEND\_INFO에 해당되는 볼륨의 가용 섹터 수가 맥락의 예약을 진행해야 하는 섹터 수보다 큰지 확인
- 가용 섹터 수가 충분하다면, 사전 예약을 진행
- 가용 섹터 수가 충분하지 않다면, 디스크 확장 혹은 추가를 진행

# 흐름 세부 사항 ④

- 사전 예약을 진행할 수 있다고 판단하게 되면, `disk_reserve_from_cache_vols` 함수를 호출
- 위 함수로 `DISK_EXTEND_INFO`에 유지하고 있는 볼륨들을 탐색하여 예약이 가능한 볼륨을 대상으로 사전 예약을 진행
- 탐색은 사전 예약의 대상으로 하는 볼륨의 타입에 따라 탐색 시작 지점과 탐색 방향이 다름

- 볼륨 타입이 `Permanent`인 경우

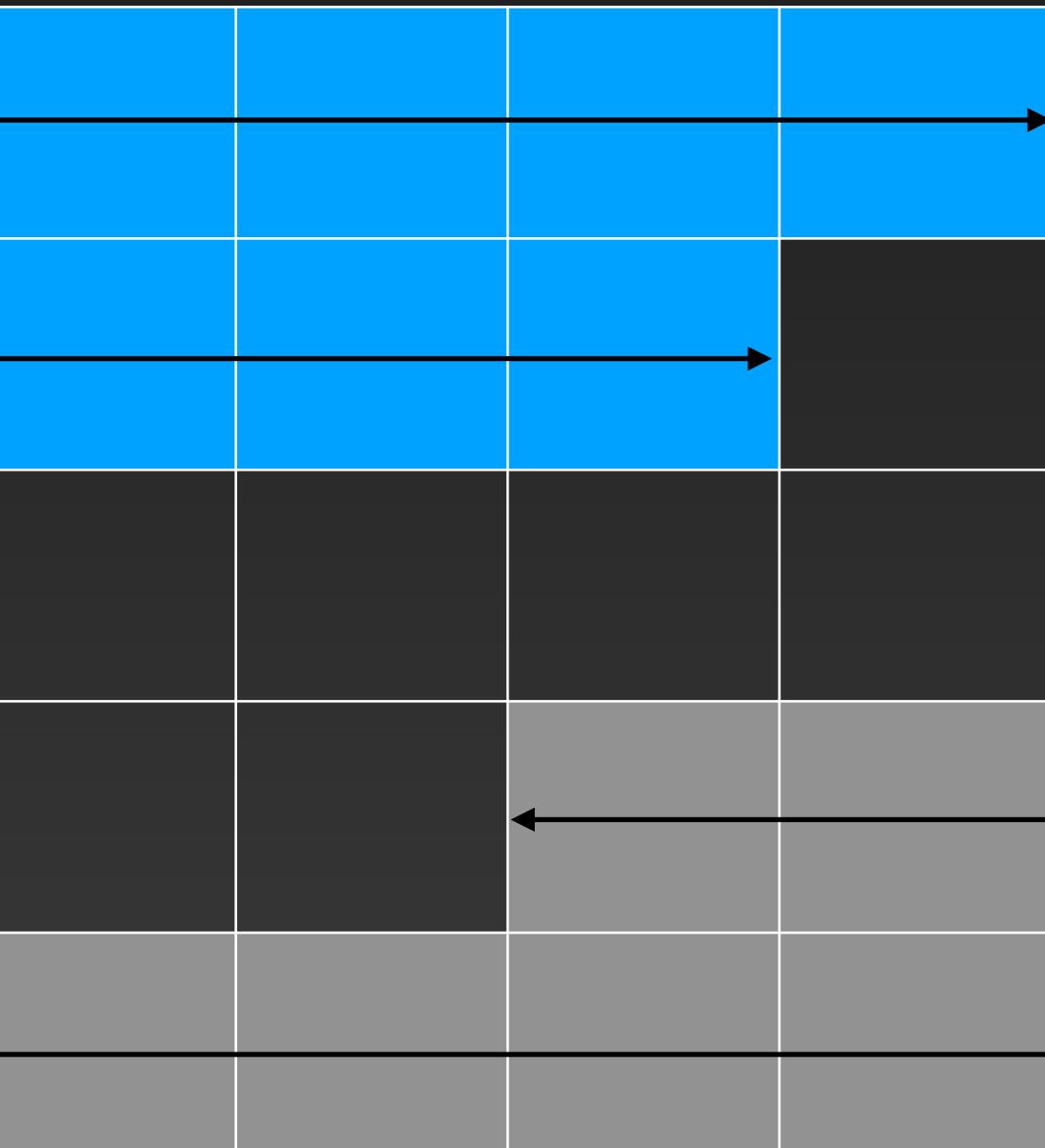
시작 인덱스는 0, 진행 방향은 +1, 끝 인덱스는 `Permanent` 타입 볼륨의 수까지

- 볼륨 타입이 `Temporary`인 경우

시작 인덱스 볼륨 기록 끝 지점, 진행 방향은 -1, 끝 인덱스는 `Temporary` 타입 볼륨의 수까지

`disk_Cache->vols`

`Permanent` 타입



`Temporary` 타입

# 흐름 세부 사항 ⑤

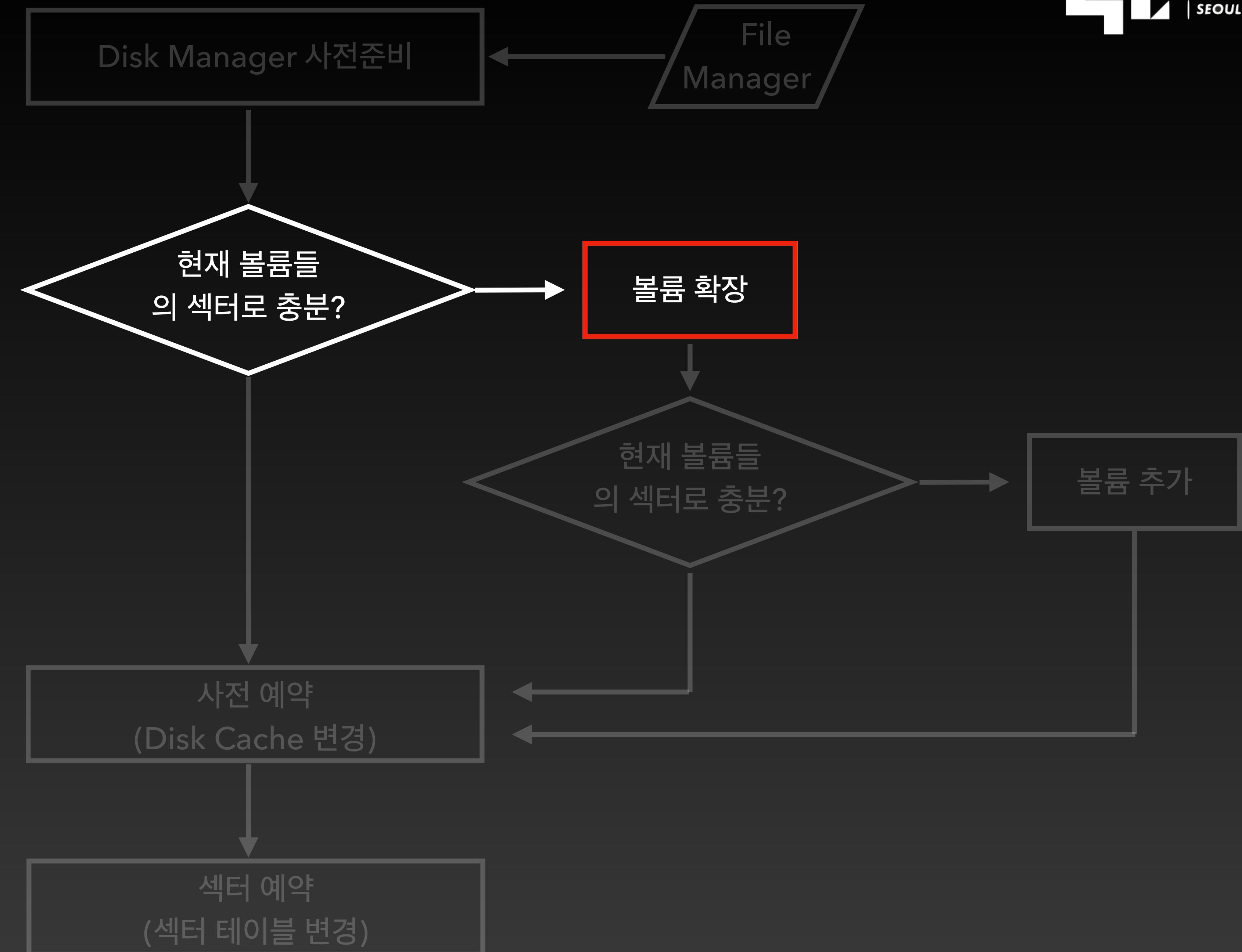
- `disk_reserve_from_cache_volume` 함수 호출을 통해 특정 볼륨을 대상으로 맥락의 예약해야 하는 섹터 수만큼 공간을 확보
- 공간을 확보한다는 것은 4가지 작업으로 이뤄짐
  1. 디스크 캐시에 사전 예약을 진행한 볼륨의 가용 공간 수를 감소
  2. DISK\_EXTEND\_INFO에서 유지하고 있는 전체적인 볼륨에서의 가용 섹터 수를 감소
  3. 맥락에 사전 예약을 진행한 볼륨과 얼만큼의 공간을 확보했는지 기록
  4. 맥락에 확보한 공간만큼 예약해야 하는 섹터 수를 감소

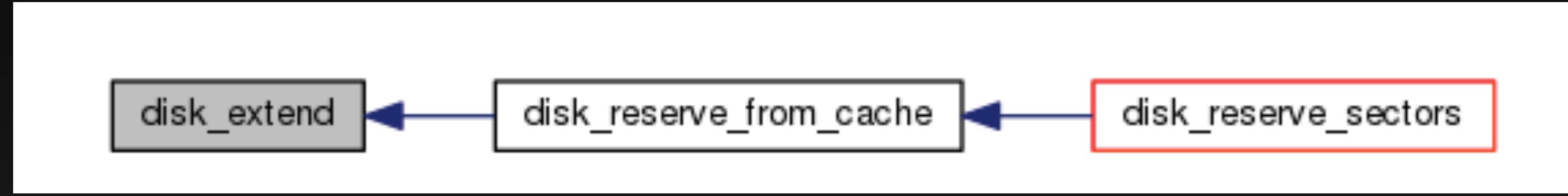
디스크 확장

disk\_extend

by samin

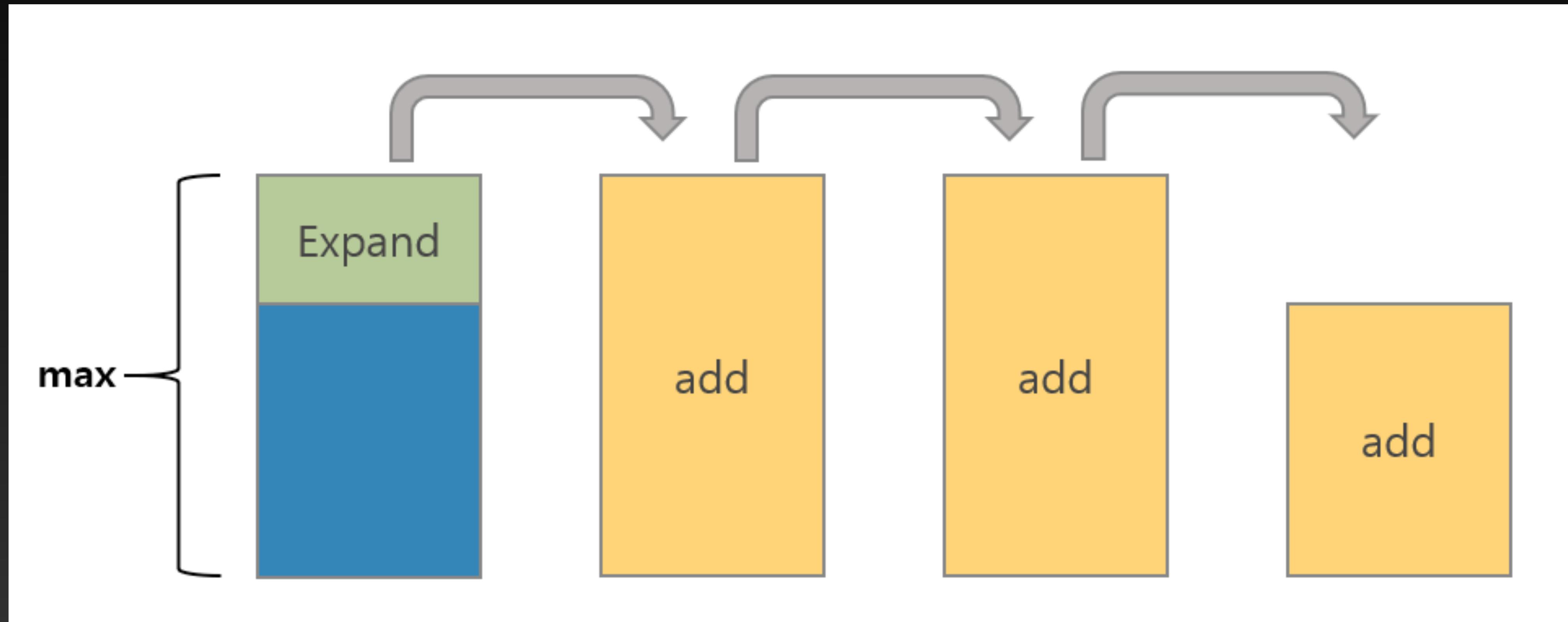
# Step 1 볼륨 확장



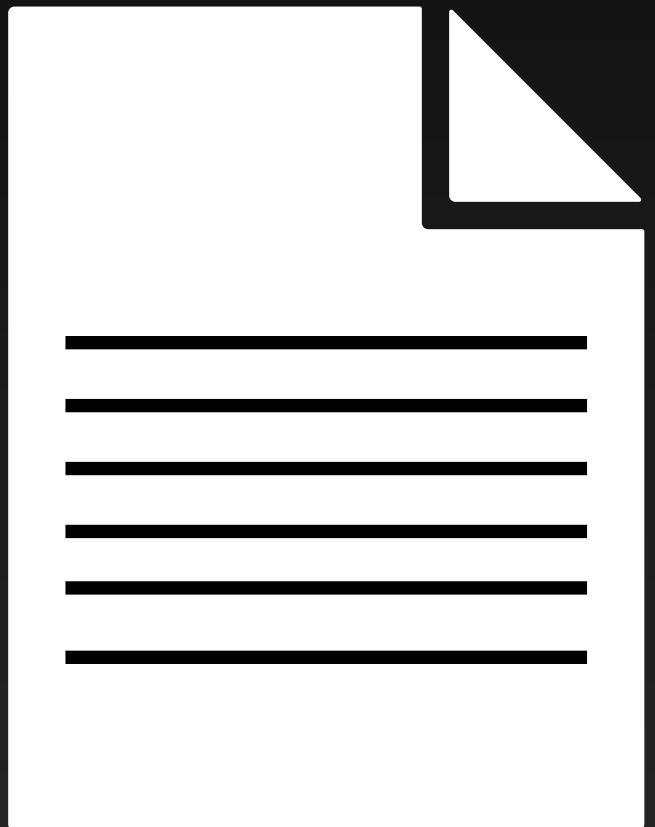


현재 상태로는 예약을 처리할수 없다면  
disk\_extend를 호출하여 공간 확보 후 사전 예약

# disk\_extend



# DISK\_EXTEND\_INFO



전체 볼륨들의  
섹터 정보

nset\_free : 볼륨들의 합산 가용 섹터 수  
nset\_total : 볼륨들의 합산 전체 섹터 수  
nset\_max : 볼륨들의 합산 최대 섹터 수  
nset\_intention : 사전예약 시 공간부족으로 공간을 확보하려 할 때, 확보하려는 섹터 수  
mutex\_reserve : 사전예약을 위한 뮤텍스  
owner\_reserve : mutex\_reserve 뮤텍스를 잡은 쓰레드 ID  
nset\_vol\_max : 볼륨 확장 시 최댓값  
volid\_extend : 볼륨 확장시  
auto\_extend 대상이 되는 볼륨  
(하나의 볼륨의 최댓값까지 확장되어야 새로운 볼륨이 생성되므로, 항상 마지막에 생성된 볼륨을 가리킨다)  
voltype : 볼륨 타입

# disk\_extend\_볼륨확장(extend) & 사전예약

1. 현재 볼륨의 확장 섹터 수 구함.
2. 볼륨 확장
  1. 볼륨 헤더 수정
  2. 새로운 볼륨의 페이지 개수 구하기
  3. 영구 볼륨일시 로그 남김
  4. 확장 fileio\_expand\_to
3. 확장된 섹터 수, 변수들에 적용
4. 확장한 볼륨에 사전예약 진행

# disk\_extend\_볼륨확장(extend) & 사전예약

1. 현재 볼륨의 확장 섹터 수 구함.

MIN (확보해야 할 총 섹터수,  
확장 가능한 최대 섹터 개수 - 현재 섹터의 개수)

# disk\_extend\_볼륨확장(extend) & 사전예약

## 2. 볼륨 확장 disk\_volume\_expand()

1. 볼륨 헤더 수정
2. 새로운 볼륨의 페이지 개수 구하기
3. 영구 볼륨일시 로그 남김
4. 확장 fileio\_expand\_to()

# disk\_extend\_볼륨확장(extend) & 사전예약

## 3. 확장된 섹터 수, 변수들에 적용

확보할 총 섹터 수 -= 확장해서 새로 생긴 섹터수

현재 섹터의 개수 += 확장해서 새로 생긴 섹터수

# disk\_extend\_볼륨확장(extend) & 사전예약

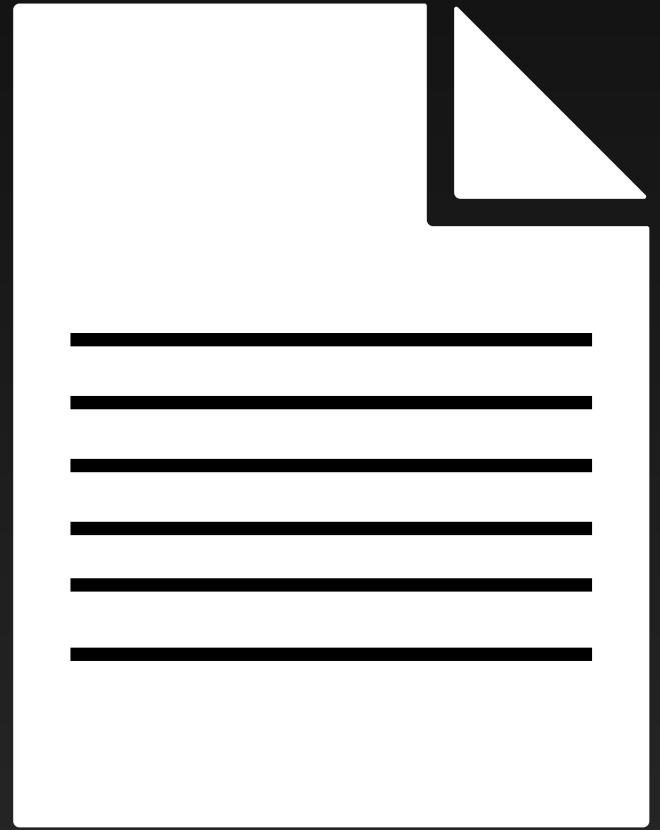
## 4. 확장한 볼륨에 사전예약 진행

disk\_reserve\_from\_cache\_volume()

# disk\_extend\_볼륨추가(add) & 사전예약

1. volext지역변수 초기화. (볼륨 생성에 필요한 정보 저장 변수)
2. 볼륨이 모두 확보 될때까지 볼륨 추가&사전예약 반복
3. 생성할 볼륨의 total값 저장
4. 볼륨 추가
5. 새 볼륨 추가로 생긴 섹터 수를 변수값에 적용
6. 추가한 볼륨에 사전예약 진행
7. 다음에 확장할 볼륨id 변수에 새 볼륨id저장

## DBDEF\_VOL\_EXT\_INFO



볼륨 추가 시 필요한  
정보 저장 구조체

```
const char *path; 볼륨이 생성될 경로, NULL이면 시스템 파라  
미터 값  
const char *name; 볼륨 명, NULL이면  
[db_name].ext[volid] 형식으로 생성  
const char *comments;  
int max_npaged; 생성하는 볼륨의 최대 페이지  
int extend_npaged;  
INT32 nsect_total; 생성 볼륨의 현재 섹터 수  
INT32 nsect_max; 볼륨이 확장할 때 가질 수 있는 최대 섹터 수  
int max_writesize_in_sec;  
DB_VOLPURPOSE purpose;  
DB_VOLTYPE voltype;  
bool overwrite;
```

# disk\_extend\_볼륨추가(add) & 사전예약

1. volext지역변수 초기화. (볼륨 생성에 필요한 정보 저장 변수)
2. 볼륨이 모두 확보 될때까지 볼륨 추가&사전예약 반복

# disk\_extend\_볼륨추가(add) & 사전예약

## 3. 생성할 볼륨의 total값 저장

total = 확보할 섹터 수 + 시스템 페이지 영역 섹터 수

total = MIN(볼륨이 확장할 때 가질 수 있는 최대 섹터 수 ,  
total)

total = MAX(total, 디스크 볼륨 섹터의 최소 값)

반올림 해준다

# disk\_extend\_볼륨추가(add) & 사전예약

## 4. 볼륨 추가

disk\_add\_volume()

# disk\_extend\_볼륨추가(add) & 사전예약

5. 새 볼륨 추가로 생긴 섹터 수를 변수값에 적용

확보할 섹터 수 -= 새로 확장한 섹터수

# disk\_extend\_볼륨추가(add) & 사전예약

## 6. 추가한 볼륨에 사전예약 진행

disk\_reserve\_from\_cache\_volume()

# disk\_extend\_볼륨추가(add) & 사전예약

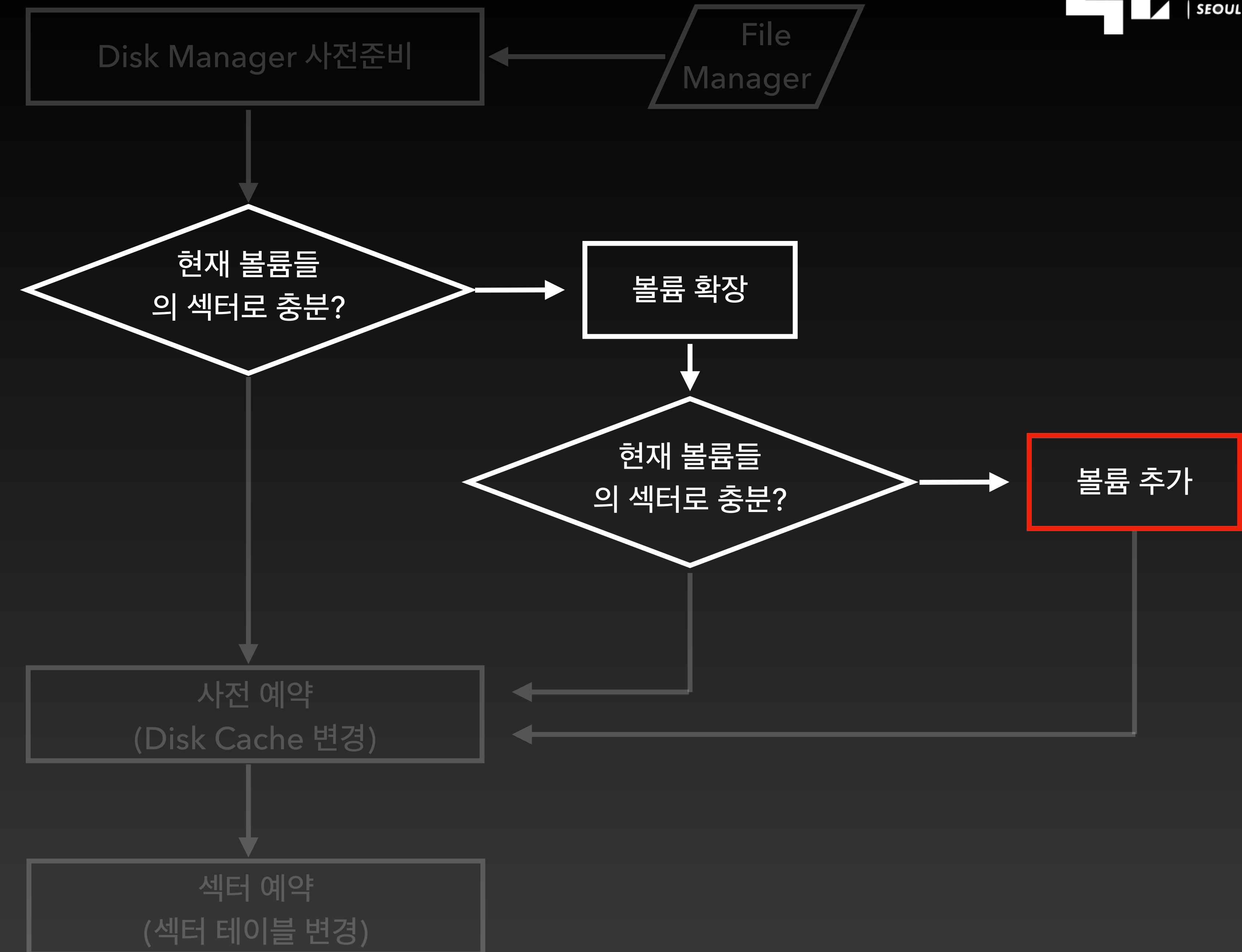
7. 다음에 확장할 볼륨id 변수에 새 볼륨id저장

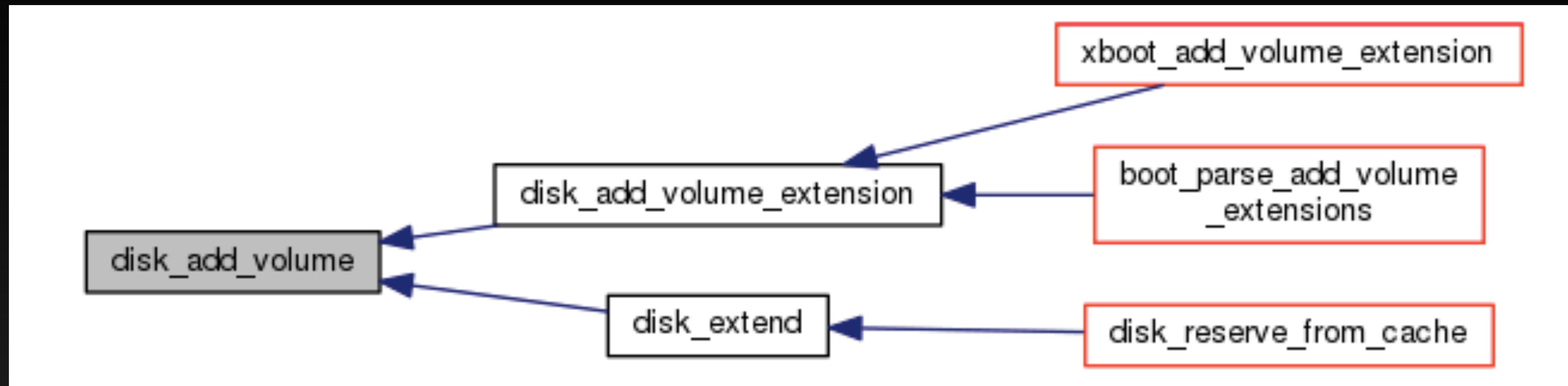
디스크 추가

disk\_add\_volume

by samin

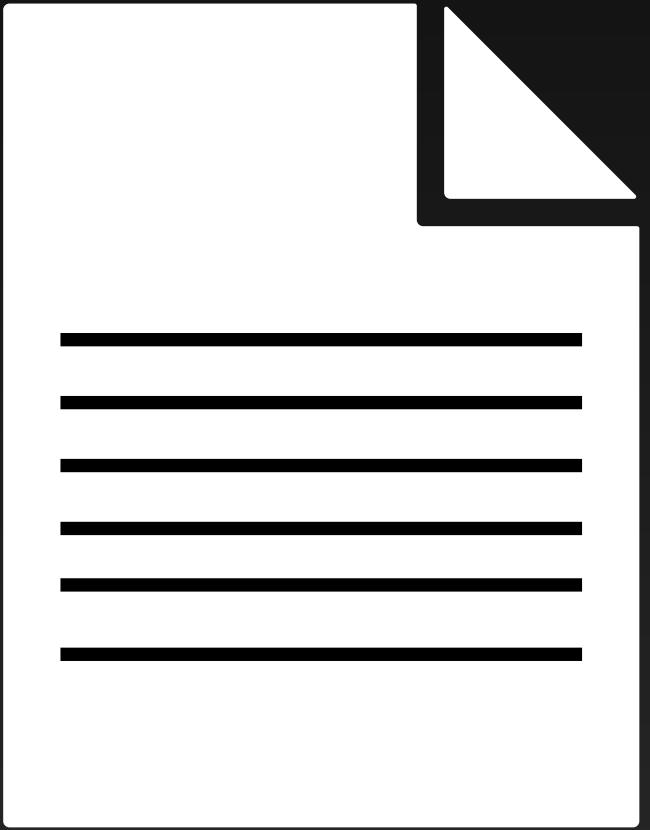
# Step 1 볼륨 추가





처음 데이터베이스 생성시 생성할 볼륨을 명시하거나,  
사용자가 임의로 볼륨을 추가하거나,  
사전예약시 볼륨을 최대크기까지 확장후 추가적인 볼륨이 필요시  
disk\_add\_volume 호출

## boot\_Db\_param



볼륨마다 있는 시스템  
힙 파일에 저장된 볼륨  
에 대한 파라미터들을  
지니고 있는 전역변수

...

VOLID nvols; 생성된 볼륨 수

VOLID temp\_nvols; 생성된 임시 볼륨의 수

VOLID last\_volid; 마지막 볼륨 식별자

VOLID temp\_last\_volid; 다음 임시 볼륨 식별자.

.....

# disk\_add\_volume

1. disk\_Cache 의 볼륨이 가득찼는지 검사
2. 새로 생성한 vol 의 fullname, volid 만들기
3. 시스템별 허용가능한 nsect 최대길이 구하기
4. 캐시에 볼륨 정보 추가(크기, 목적)
5. 볼륨 생성 disk\_format()
6. 영구타입볼륨이라면 볼륨 인포 파일(\_vinf) 업데이트. logpb\_add\_volume()
7. 새로운 볼륨정보를 boot\_Db\_param에 업데이트한다.boot\_dbparm\_save\_volume()

# disk\_add\_volume

## 1. disk\_Cache 의 볼륨이 가득찼는지 검사

디스크 캐시의 영구볼륨 + 임시볼륨 허용가능한 볼륨 id보다 작아야한다.

# disk\_add\_volume

2. 새로 생성한 vol 의 fullname, volid 만들기 ()

DBDEF\_VOL\_EXT\_INFO의 path, name 를 활용하여 fullname 구한다.

boot\_Db\_parm 을 활용하여 volid 추가

# disk\_add\_volume

3. 시스템별 허용가능한 nsect 최대길이 구하기

32bit

(INT\_MAX / (page\_size)) / DISK\_SECTOR\_NPAGES<64>

64bit

INT\_MAX / DISK\_SECTOR\_NPAGES<64>

# disk\_add\_volume

## 4. 캐시에 볼륨 정보 추가(크기, 목적)

볼륨의 탑입에 따라  
disk\_Cache의 탑입별 볼륨 크기 증가

볼륨 목적 대입

# disk\_add\_volume

5. 볼륨 생성

disk\_format()

# disk\_add\_volume

6. 영구타입볼륨이라면 볼륨 인포 파일(\_vinf) 업데이트.

logpb\_add\_volume()

# disk\_add\_volume

7. 새로운 볼륨정보를 boot\_Db\_param에 업데이트한다.

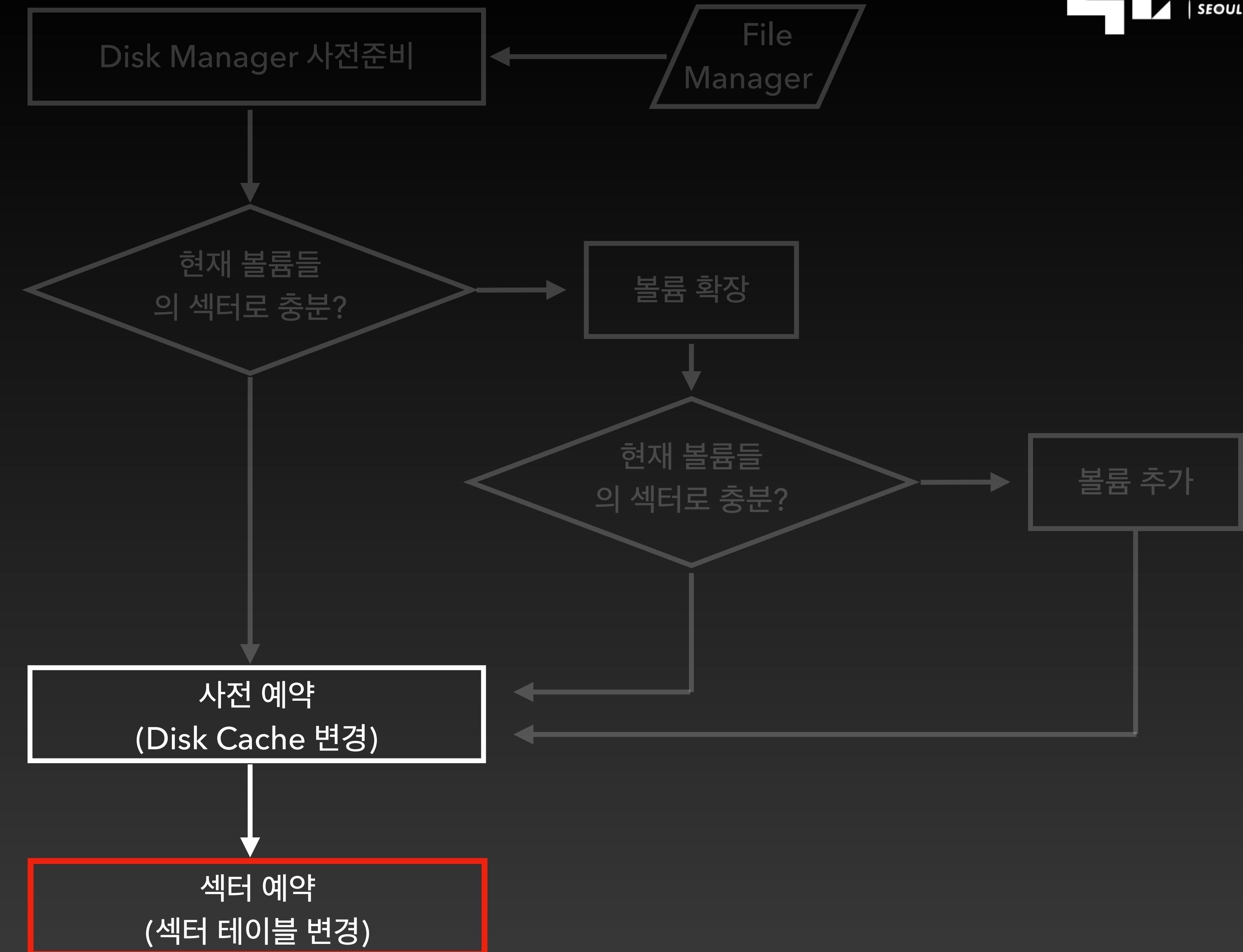
boot\_dbparm\_save\_volume()

섹터 예약

(disk\_reserve\_sectors\_in\_volume)

by jinbekim

## Step 2 섹터 예약



***struct disk\_reserve\_context***

```
int nsect_total;  
VSID *vsidp;  
DISK_CACHE_VOL_RESERVE cache_vol_reserve[VOLID_MAX];  
int n_cache_vol_reserve;  
int n_cache_reserve_remaining;  
DKNSECTS nsects_lastvol_remaining;  
DB_VOLPURPOSE purpose;
```

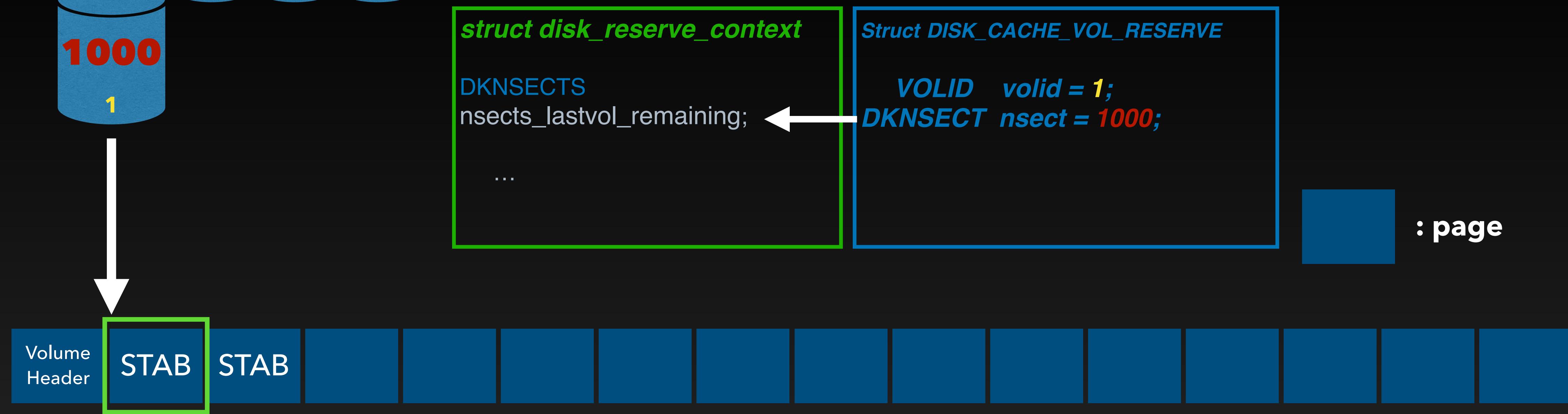
***Struct DISK\_CACHE\_VOL\_RESERVE***

```
VOLID volid = 1;  
DKNSECT nsect = 1000;
```

어떤 볼륨(volid)에서 얼마큼(nsect) 예약 하는지 정보 배열

배열의 길이는 얼마인지

현재 볼륨에서 얼마나 더 예약해야 하는지



# Vol1. STAB

1

*:disk\_stab\_unit*



# 1. *nsect\_lastvol\_remaining* >= 64

## ***struct disk\_reserve\_context***

DKNSECTS nsects\_lastvol\_remaining; // 1000

■

**Vol1. STAB**

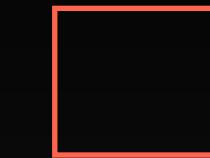
: disk\_stab\_unit

1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

**1. nsect\_lastvol\_remaining >= 64*****struct disk\_reserve\_context***

```
DKNSECTS nsects_lastvol_remaining; // 1000 - 64
```

```
...
```

**Vol1. STAB**: *disk\_stab\_unit*

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

**2. *nsect\_lastvol\_remaining < 64******struct disk\_reserve\_context***

DKNSECTS nsects\_lastvol\_remaining; // 14

...

**Vol1. STAB**: *disk\_stab\_unit*

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1

**2. *nsect\_lastvol\_remaining < 64******struct disk\_reserve\_context***

DKNSECTS nsects\_lastvol\_remaining; // 0

...

# Vol1. STAB

1

## **:disk\_stab\_unit**



## 3. *Unit* $\neq 0$

## ***struct disk\_reserve\_context***

DKNSECTS nsects\_lastvol\_remaining;

■

# Vol1. STAB

1

## **:disk\_stab\_unit**



## 3. *Unit* $\neq 0$

## ***struct disk\_reserve\_context***

DKNSECTS nsects\_lastvol\_remaining;

1

# Vol1. STAB

1

## **:disk\_stab\_unit**



## 3. *Unit* $\neq 0$

## ***struct disk\_reserve\_context***

## DKNSECTS nsects\_lastvol\_remaining;

■

# Vol1. STAB

1

## **:disk\_stab\_unit**



## 4. *Unit != 0 && !trailing*

## ***struct disk\_reserve\_context***

DKNSECTS nsects\_lastvol\_remaining;

■

# Bit count

# Vol1. STAB

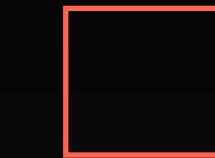
1

## :disk\_stab\_ur

*bit* = ~*b*

*bit* &= -1

이 후, 아래의 값과 &연산의 결과로 `offset_to_bit`를 알아내게 된

**Vol1. STAB**: *disk\_stab\_unit*

1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

***struct vsid****int32\_t sectid;*  
*Short valid;****struct disk\_reserve\_context****VSID \*vsidp;*

...



# Vol1. STAB

1

## **:disk\_stab\_unit**



## 5. *hint\_alloc*

## ***struct disk\_volume\_header***

## 볼륨 정보

## 섹터 정보

## **SECTID hint allocsect**

시스템페이지

체크포인트

기타

가변길이변수

Thank you