CUBRID

42 | SEOUL

42 Seoul & CUBRID

Double Write Buffer

Team Z
keokim(김건우) minjkim2(김민준) saoh(오세훈)

CUBRID

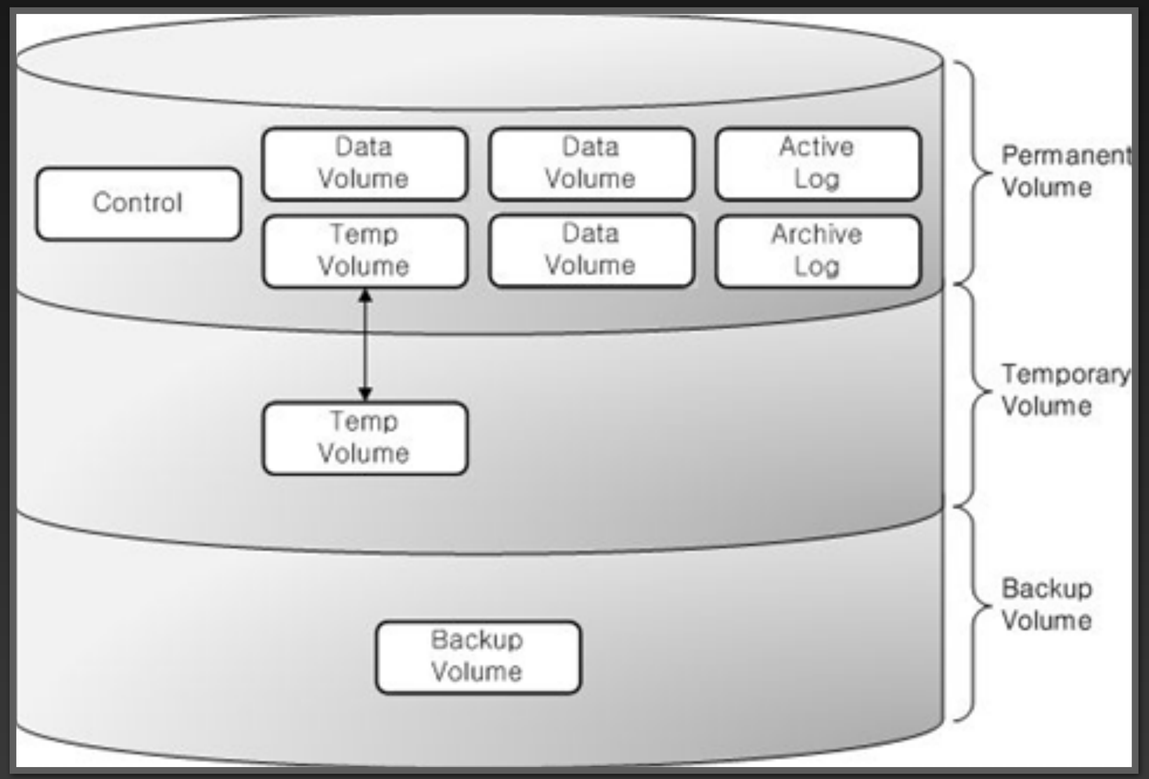
42 | SEOUL

데이터베이스 볼륨 구조

CUBRID 데이터베이스의 볼륨은 크게 **영구적 볼륨**, **일시적 볼륨**, **백업 볼륨**으로 분류한다.

➔ 영구적 볼륨은 영구적 데이터를 저장하지만 일시적 데이터도 저장할 수 있는 데이터 볼륨이 있다.

➔ DWB 파일이 여기에 해당한다.



CUBRID

42 | SEOUL

DWB 파일

DWB 파일은 partial write로 인한 I/O 에러를 방지하기 위하여 disk에 쓰여지는 데이터 페이지들의 복사본을 저장하는 공간이다.

➔ 데이터 파일에 쓰기 전에 buffer pool로 flush된 페이지를 쓰는 저장영역이다.

➔ 모든 데이터 페이지는 DWB에 먼저 쓰여지고 난 후 영구 데이터 볼륨에 있는 데이터 위치에 쓰여진다.

➔ 데이터베이스가 재시작될 때, partial write된 페이지들이 탐지되고 DWB에서 대응되는 페이지로 대치된다.

CUBRID

42 | SEOUL

DWB 기능

DB단의 I/O 작업은 16KB (page의 기본단위)
OS단의 I/O 작업은 4KB

Partial write : OS의 I/O 단위인 block과 DB 시스템의 page의 크기가 같지 않기 때문에 발생하는 문제.

DWB는 DB단의 논리적인 I/O 단위와 OS단의 실질적인 I/O 단위가 다름으로 인해 발생할 수 있는 page corrupt를 방지하기 위해 사용하는 기능이다.

CUBRID

42 | SEOUL

DB에서의 DWB

DB 시스템은 Page Replace를 위한 강제적 / 주기적 Flush 작업을 진행한다.

Flush는 DWB를 사용하지 않고 진행하거나 DWB를 사용하여 진행할 수 있다.

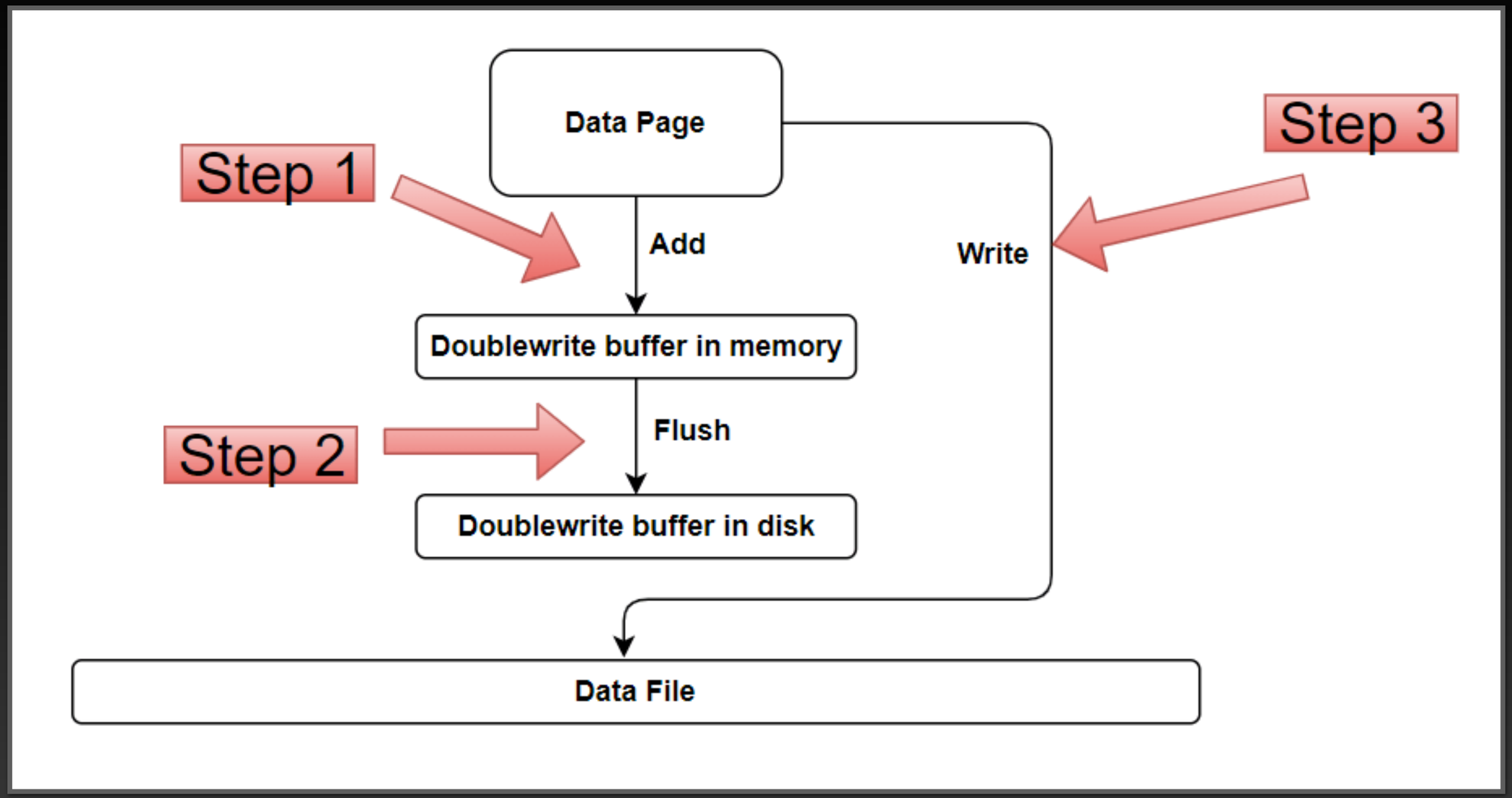
DWB를 사용하는 경우 system crash가 발생해서 일어난 partial write에 대해 page recovery가 가능하다.

DB 시스템이 DWB를 사용하는 방식

➔ Page를 저장할 공간 탐색 및 저장

➔ 특정 개수의 page를 disk로 한번에 flush

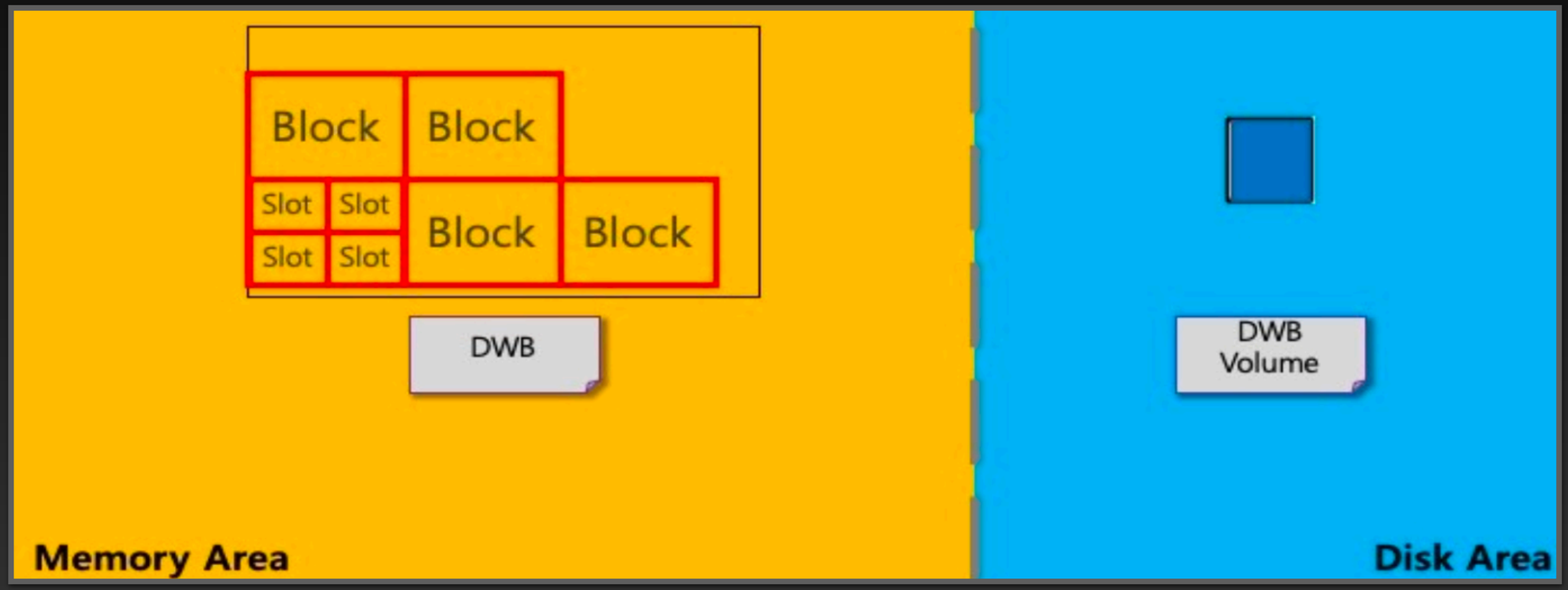
➔ Partial write가 일어난 DB 복구


Data Page
Step 1
Step 3
Add
Write
Doublewrite buffer in memory
Flush
Step 2
Doublewrite buffer in disk
Data File

CUBRID

42 | SEOUL

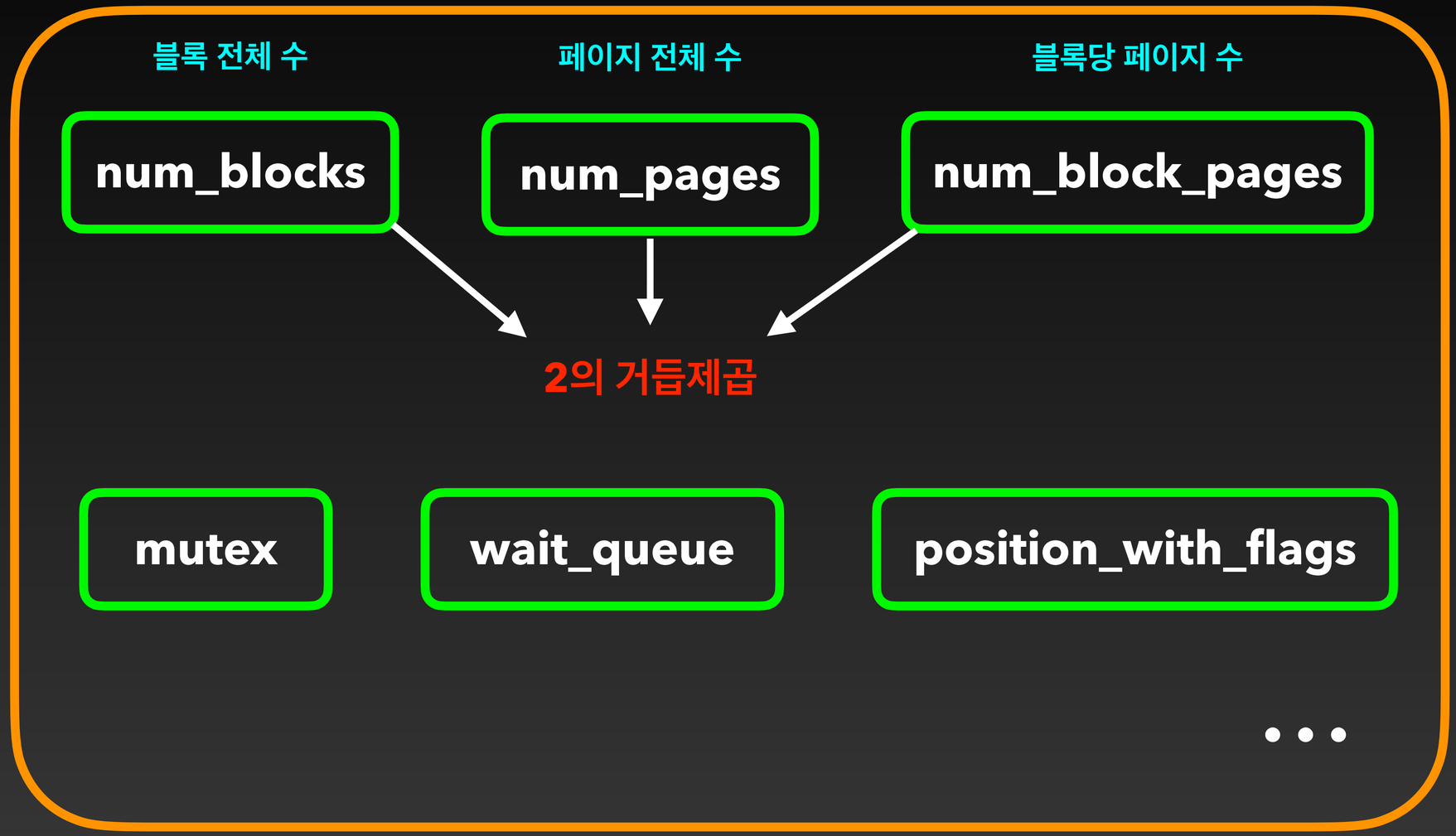
DWB 구조



CUBRID

42 | SEOUL

static DOUBLE WRITE BUFFER dwb_Global;



1 <= DWB_BLOCK <= 32
0.5MB <= DWB_SIZE <= 32MB

IO_DEFAULT_PAGE_SIZE : 16KB
num_pages = DWB_SIZE / IO_PAGE_SIZE
0.5MB / 16KB = 32
32MB / 16KB = 2048

32 <= num_pages <= 2048

num_block_pages = num_pages / num_blocks

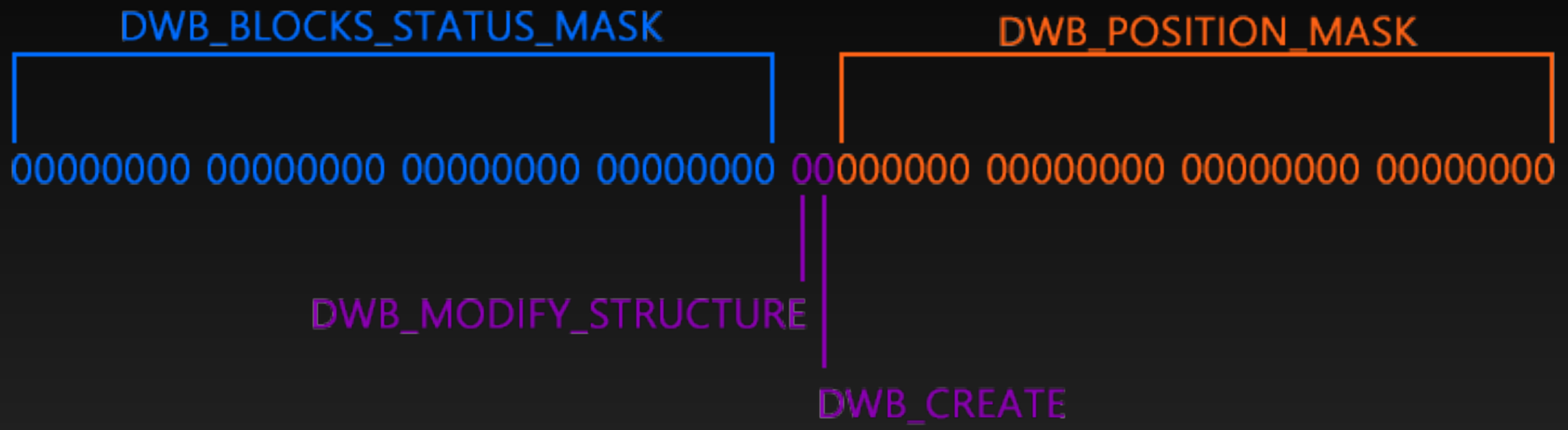
1 <= num_block_pages <= 2048

Slot의 개수 = num_block_pages

1 <= Slot 개수 <= 2048
1 <= log2_num_block_pages <= 11

1개의 Slot -> 1개의 Page

position_with_flags



DWB_BLOCKS_STATUS_MASK: 어떤 블록에 대해 SET이면 WRITE 중임을 나타냅니다.

DWB_POSITION_MASK: SLOT의 BLOCK내에서의 위치를 나타냅니다.

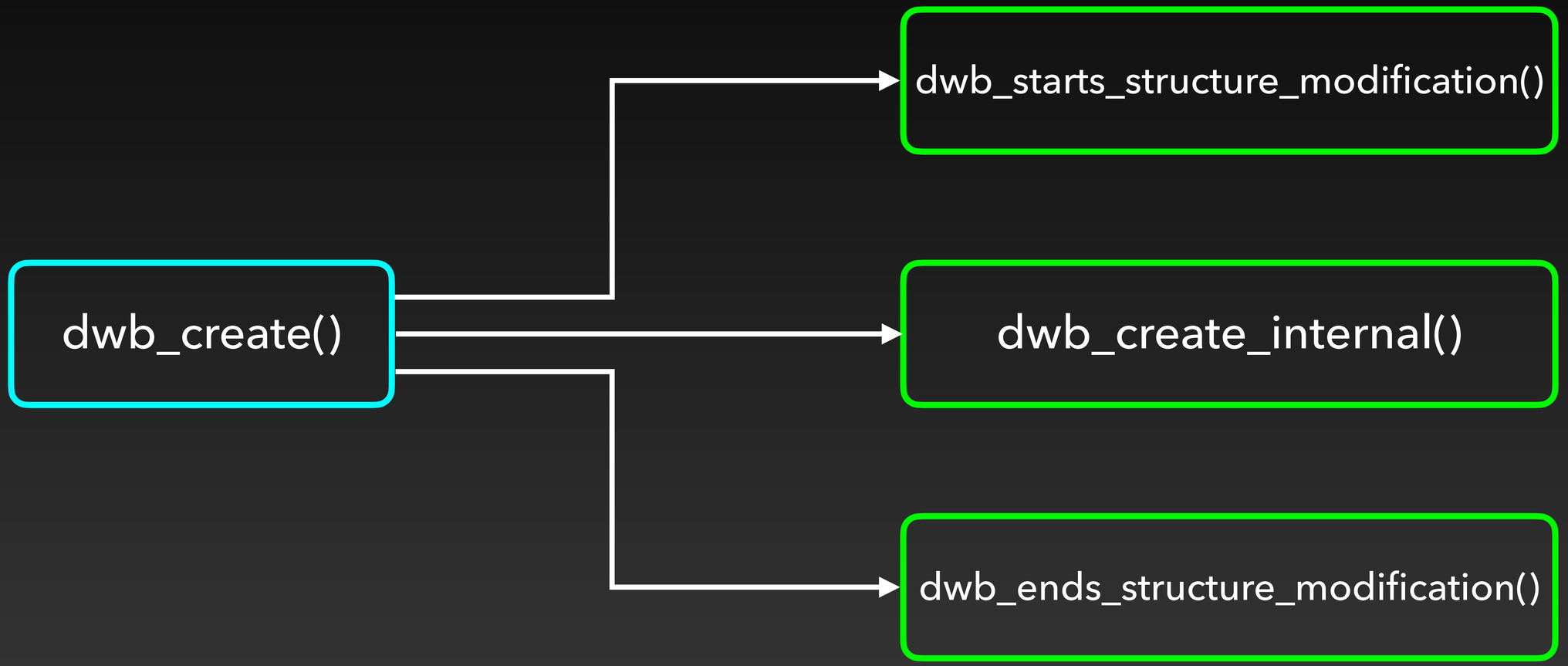
DWB_MODIFY_STRUCTURE: SET이면 DWB가 만들어지고 있음을 나타냅니다.

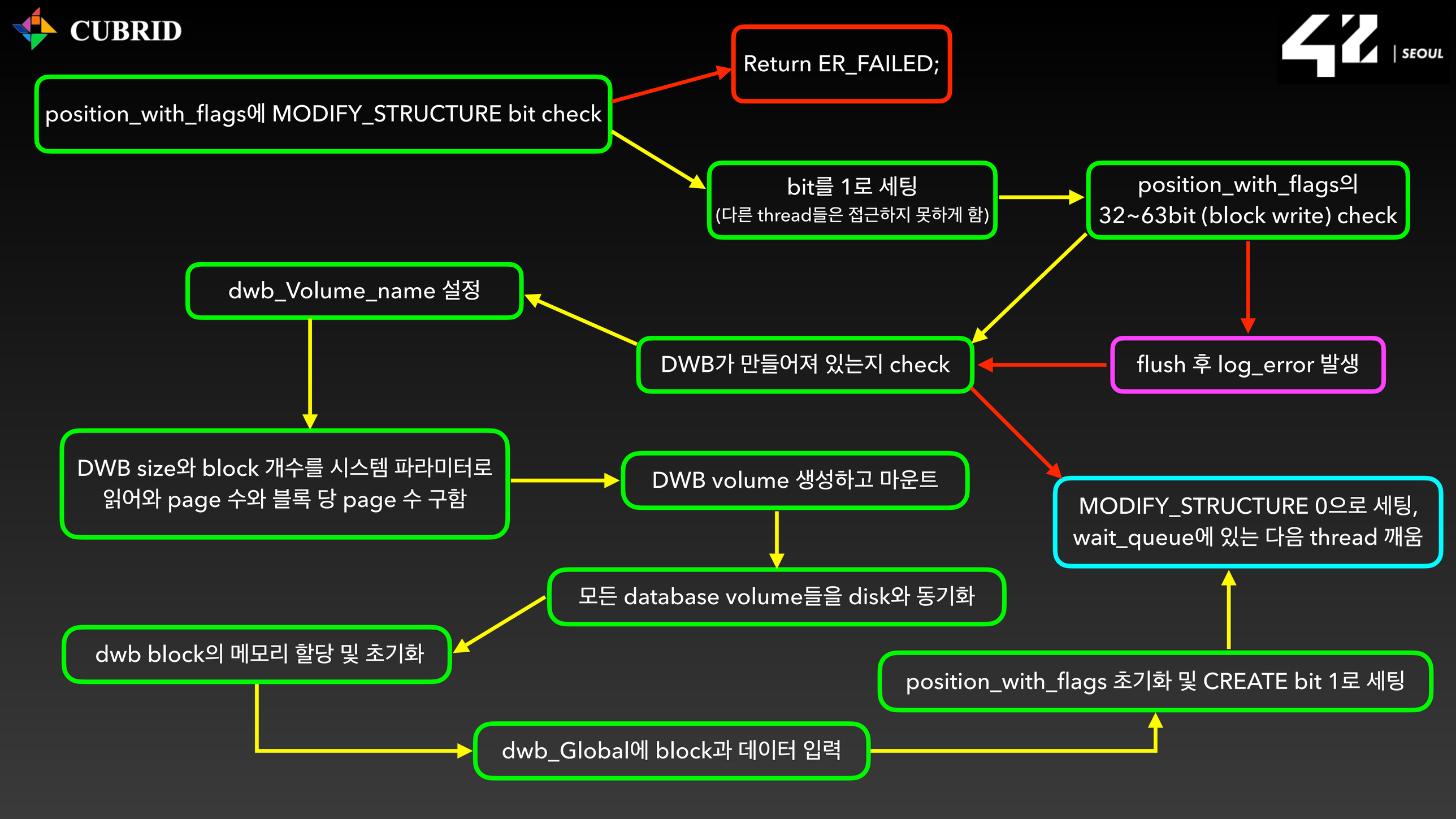
DWB_CREATE: SET이면 DWB가 만들어졌음을 나타냅니다.

CUBRID

42 | SEOUL

DWB Create




CUBRID
42 SEOUL
position_with_flags에 MODIFY_STRUCTURE bit check
Return ER_FAILED;
bit를 1로 세팅
(다른 thread들은 접근하지 못하게 함)
position_with_flags의
32~63bit (block write) check
dwb_Volume_name 설정
DWB가 만들어져 있는지 check
flush 후 log_error 발생
DWB size와 block 개수를 시스템 파라미터로
읽어와 page 수와 블록 당 page 수 구함
DWB volume 생성하고 마운트
MODIFY_STRUCTURE 0으로 세팅,
wait_queue에 있는 다음 thread 깨움
모든 database volume들을 disk와 동기화
dwb block의 메모리 할당 및 초기화
position_with_flags 초기화 및 CREATE bit 1로 세팅
dwb_Global에 block과 데이터 입력

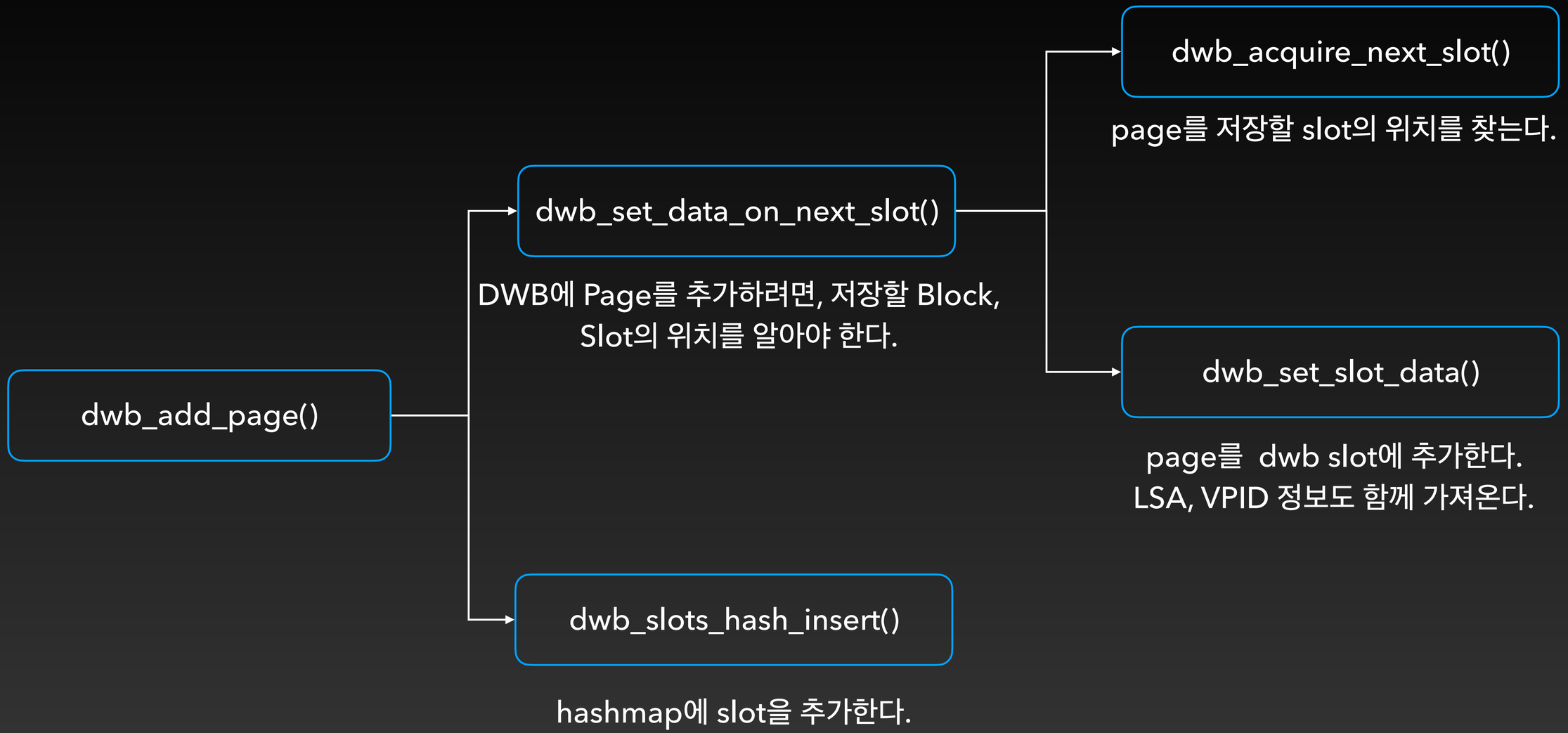
DWB에 page 추가

dwb_add_page()

DWB가 사용 가능하다면 DWB에 page를 추가하고(dwb_add_page())

사용 불가능 하다면 Disk에 Page를 바로 write.

Page는 메모리 영역에 존재.


dwb_add_page()
dwb_set_data_on_next_slot()
DWB에 Page를 추가하려면, 저장할 Block, Slot의 위치를 알아야 한다.
dwb_acquire_next_slot()
page를 저장할 slot의 위치를 찾는다.
dwb_set_slot_data()
page를 dwb slot에 추가한다.
LSA, VPID 정보도 함께 가져온다.
dwb_slots_hash_insert()
hashmap에 slot을 추가한다.

Slot의 위치 탐색

dwb_acquire_next_slot()

DWB의 어느 부분에 page를 저장할지 위치를 가져와야 한다.

- Slot의 위치는 어떻게 찾을까?
  - dwb_acquire_next_slot() 함수에서 slot의 위치를 가져온다.
  - DWB의 position_with_flags를 비트 연산을 사용해 Block 및 Slot의 index를 탐색, Slot의 시작주소를 얻는다.
  - Block의 첫번째 Slot에 Write작업을 할 경우, 해당 Block에 Writing을 시작했다는 flag를 세워준다.
  - position_with_flags 값을 다음 위치를 가르키게 증가시킨 후, dwb_Global 구조체의 position_with_flags에 저장한다.

position_with flags

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

DWB_CREATE

0000 0000 0000 0000 0000 0000 0000 0000 0100 0000 0000 0000 0000 0000 0000 0000

DWB_MODIFY_STRUCTURE

0000 0000 0000 0000 0000 0000 0000 0000 1000 0000 0000 0000 0000 0000 0000 0000

DWB_POSITION_MASK

0000 0000 0000 0000 0000 0000 0000 0000 0011 1111 1111 1111 1111 1111 1111 1111

DWB_BLOCKS_STATUS_MASK

1111 1111 1111 1111 1111 1111 1111 1111 0000 0000 0000 0000 0000 0000 0000 0000

position_with flags

0000 0000 0000 0000 0000 0000 0000 0000 0100 0000 0000 0000 0000 0010 0000 0110

DWB_GET_POSITION

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0010 0000 0110

DWB_GET_BLOCK_NO_FROM_POSITION

DWB_GET_POSITION(position_with_flags) >> (DWB_LOG2_BLOCK_NUM_PAGES)

ex) num_block_pages : 256 (>> 8)

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0010

➔ 2번째 Block

DWB_GET_POSITION_IN_BLOCK

DWB_GET_POSITION & DWB_BLOCK_NUM_PAGES - 1

DWB_GET_POSITION

0000 0000 0000 0000 0000 0000 0000 0000 0100 0000 0000 0000 0000 0010 0000 0110

DWB_BLOCK_NUM_PAGES (256) - 1

0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 1111 1111

DWB_GET_POSTION_IN_BLOCK

0000 0000 0000 0000 0000 0000 0000 0000 0100 0000 0000 0000 0000 0000 0000 0110

➔ 6번째 Slot

탐색 실패

dwb_acquire_next_slot()

!DWB_IS_CREATED

- DWB가 생성되지 않았을 경우, DWB가 flush하기 전에 삭제된 경우

DWB_IS_MODIFYING_STRUCTURE / DWB_IS_BLOCK_WRITE_STARTED

- wait_queue에서 대기할 수 없는경우

탐색 대기

dwb_acquire_next_slot()

- DWB_IS_MODIFYING_STRUCTURE (DWB 생성 / 삭제 중)
  - dwb_Global의 wait_queue에서 완료되기를 기다리고 start로 돌아가서 다시 시작
- Block에서 첫 번째 위치의 Slot인 경우
  - 다른 thread가 해당 Block에 write 작업중인 경우 Block 구조체 안의 wait queue에서 작업이 끝나기를 기다린다.
  - Write 작업 : 해당 Block에 대해서 slot 탐색 부터, DWB flush 사이의 작업을 진행 중.
  - 해당 block이 writing 상태가 끝날 때까지, work thread는 dwb_acquire_next_slot() 함수 내부 start부터 다시 시작한다.

Slot에 Page 저장

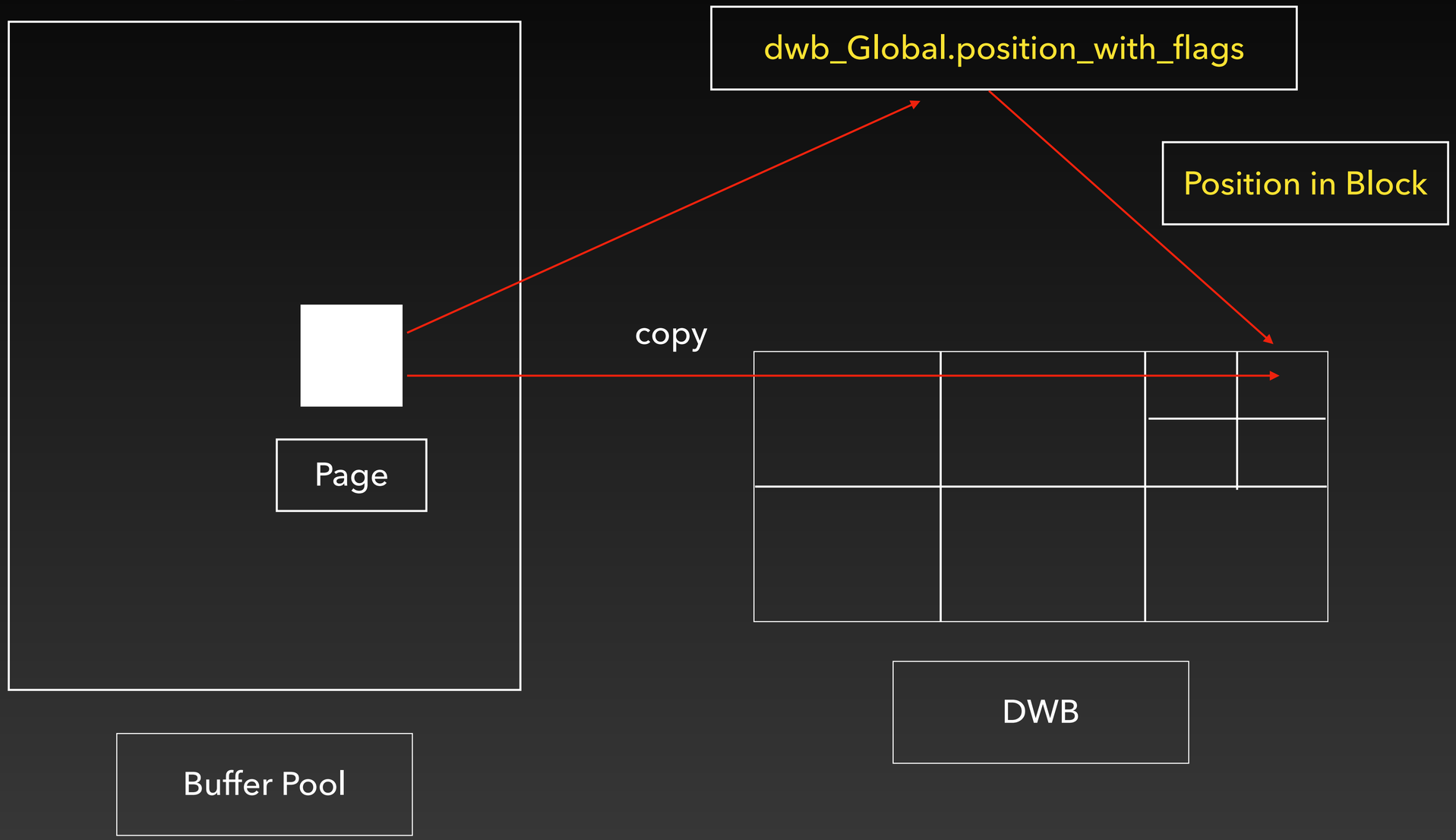
dwb_set_slot_data()

- 위에서 구한 Slot의 주소에 메모리에 있는 Page를 저장해야한다.
- memcpy() 함수를 이용해 저장을 한다.
- Page의 LSA, VPID 정보도 slot에 저장

CUBRID

42 | SEOUL

Memory Area



Page Cache

dwb_slots_hash_insert()

- dwb_Global 구조체 slots_hashmap
  - vpid 값을 key로, slots_hash_entry를 value로 사용한다.
  - Page replacement에서 Cache 역할을 한다. 찾고있는 Page가 Memory에 없을 때, DWB 에서 Page 를 찾는다. 이때 빠른 검색을 위해 이용한다.
  - DB에 Flush되면 hashmap에서 삭제된다.
- hashmap에 vpid 값이 있는지 확인하고, 없으면 vpid를 추가하고, slots_hash_entry를 가져온다.
- LSA를 비교하여 hashmap에 더 최신의 slot이 있는 경우를 제외하고, slots_hash_entry에 slot을 넣는다.

FLUSH

호출하는 주체

1. **dwb flush block daemon**
2. 직접

호출하는 경우

1. 한 **Block**의 **slot**이 가득 찼을 때
2. **DWB** 를 생성할때, 이미 존재하는 경우

Slot Ordering

qsort 로 정렬

정렬기준 : **vol id, page id, lsa page id, lsa offset**

정렬하는 이유

1. **Volume**들이 **VPID**순서로 정렬되어있다.
2. 같은 **Page** 는 **LSA** 를 비교해 최신 버전만 **Flush**

Slot Ordering

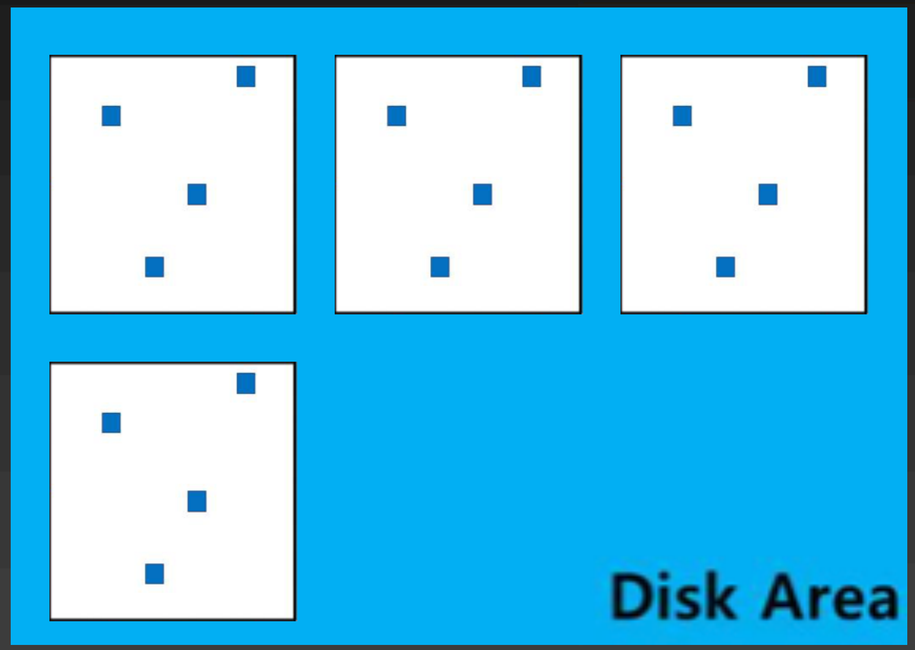
Vol id
Page id
> VPID

Page id
Offset
> Log_LSA

VPID

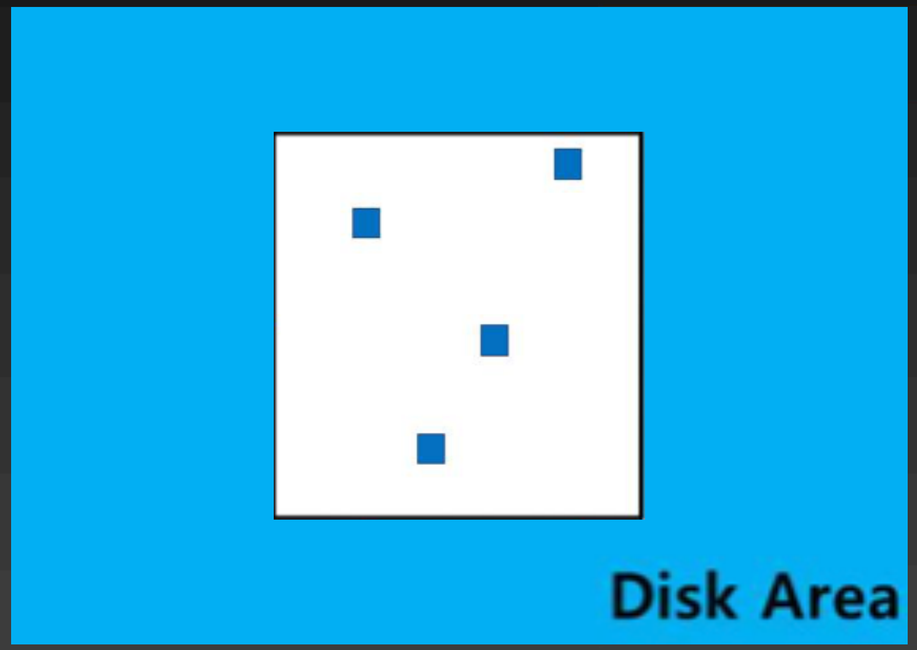
Vol id

Page id



Log_LSA

Page id
Offset



Slot Ordering

Volid : 0 Page id : 0 LSA id : 1 LSA off : 1	Volid : 1 Page id : 0 LSA id : 0 LSA off : 0	Volid : 0 Page id : 1 LSA id : 1 LSA off : 0	Volid : 0 Page id : 1 LSA id : 0 LSA off : 1	Volid : 0 Page id : 1 LSA id : 0 LSA off : 0	Volid : 0 Page id : 0 LSA id : 0 LSA off : 1	Volid : 0 Page id : 0 LSA id : 0 LSA off : 0	Volid : 0 Page id : 0 LSA id : 1 LSA off : 0

qsort 로 정렬

정렬기준 : **vol id, page id, lsa page id, lsa offset**

Slot Ordering

Volid : 0 Page id : 0 LSA id : 0 LSA off : 0	Volid : 0 Page id : 0 LSA id : 0 LSA off : 1	Volid : 0 Page id : 0 LSA id : 1 LSA off : 0	Volid : 0 Page id : 0 LSA id : 1 LSA off : 1	Volid : 0 Page id : 1 LSA id : 0 LSA off : 0	Volid : 0 Page id : 1 LSA id : 0 LSA off : 1	Volid : 0 Page id : 1 LSA id : 1 LSA off : 0	Volid : 1 Page id : 0 LSA id : 0 LSA off : 0

qsort 로 정렬

정렬기준 : **vol id, page id, lsa id, lsa offset**

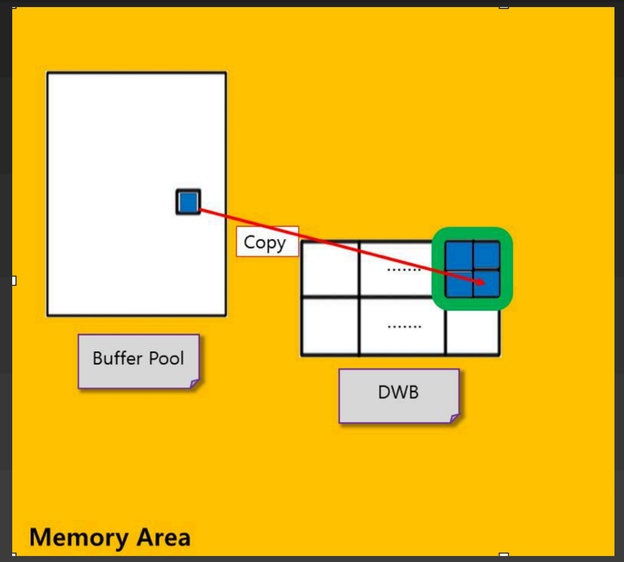
Slot Ordering

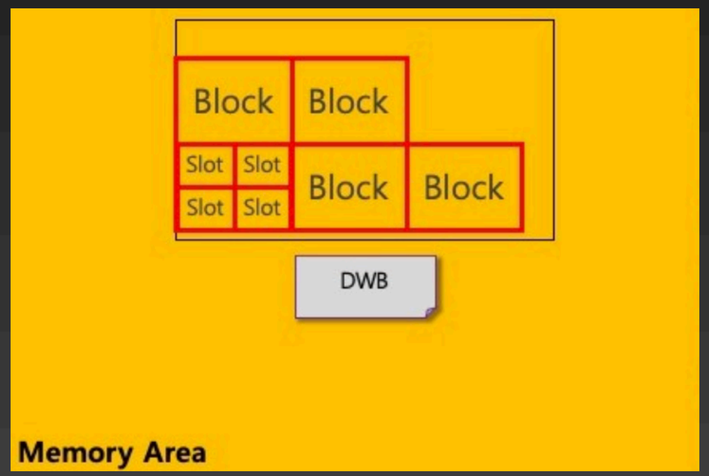
정렬된 Slot의 중복 Page 제거

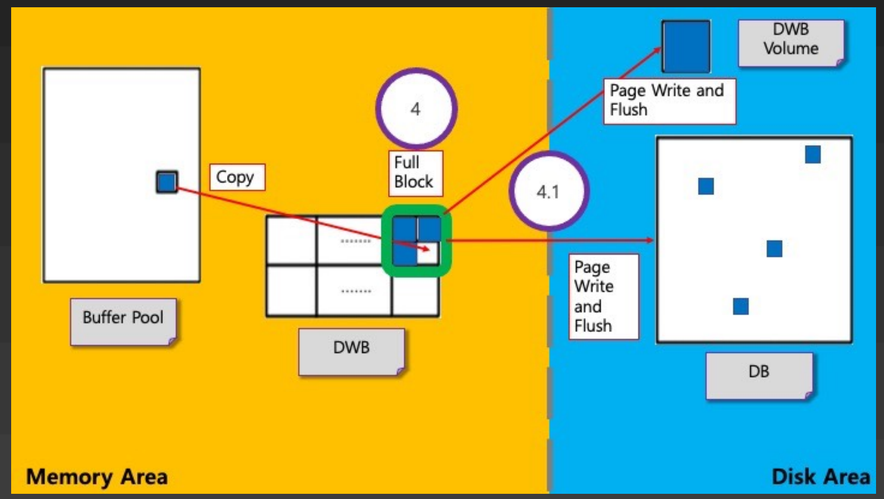
Volid : 0
Page id : 0
LSA id : 1
LSA off : 1

Volid : 0	Volid : 1
Page id : 1	Page id : 0
LSA id : 1	LSA id : 0
LSA off : 0	LSA off : 0

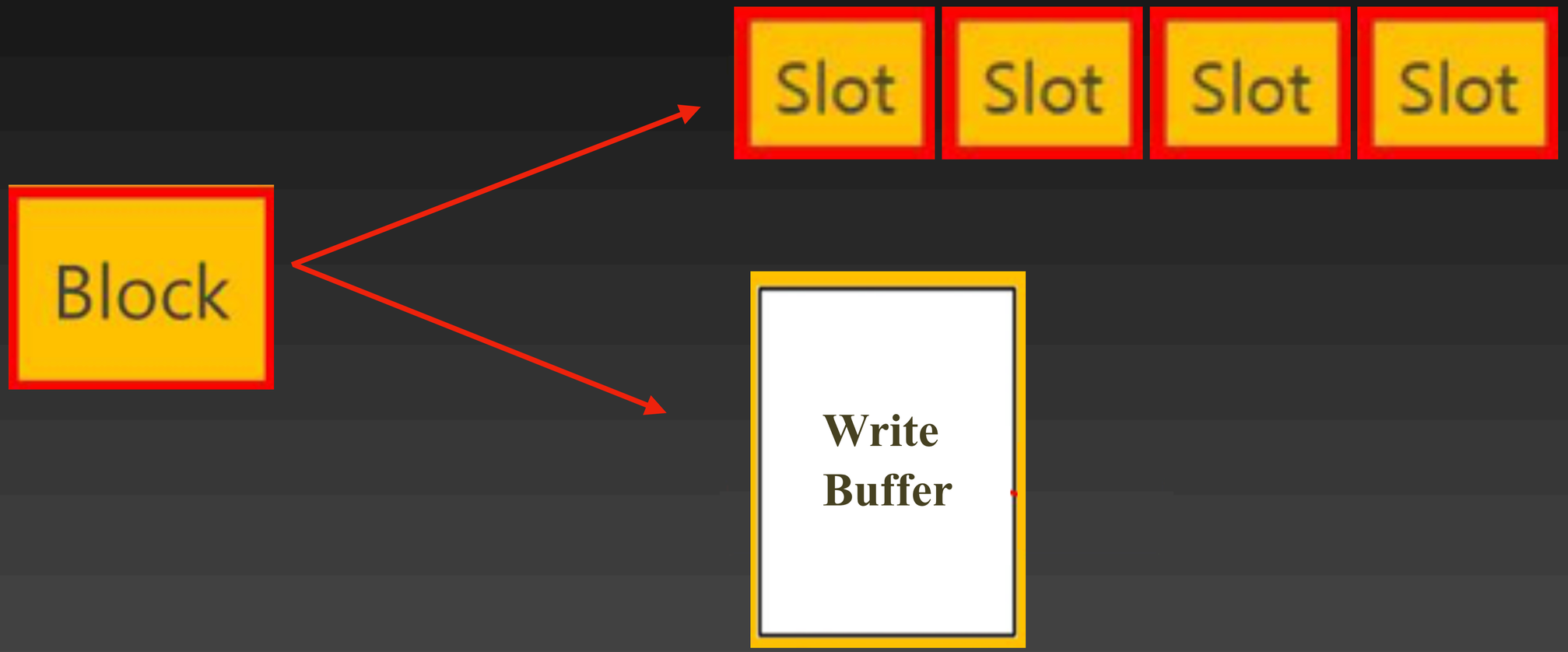
Block Structure



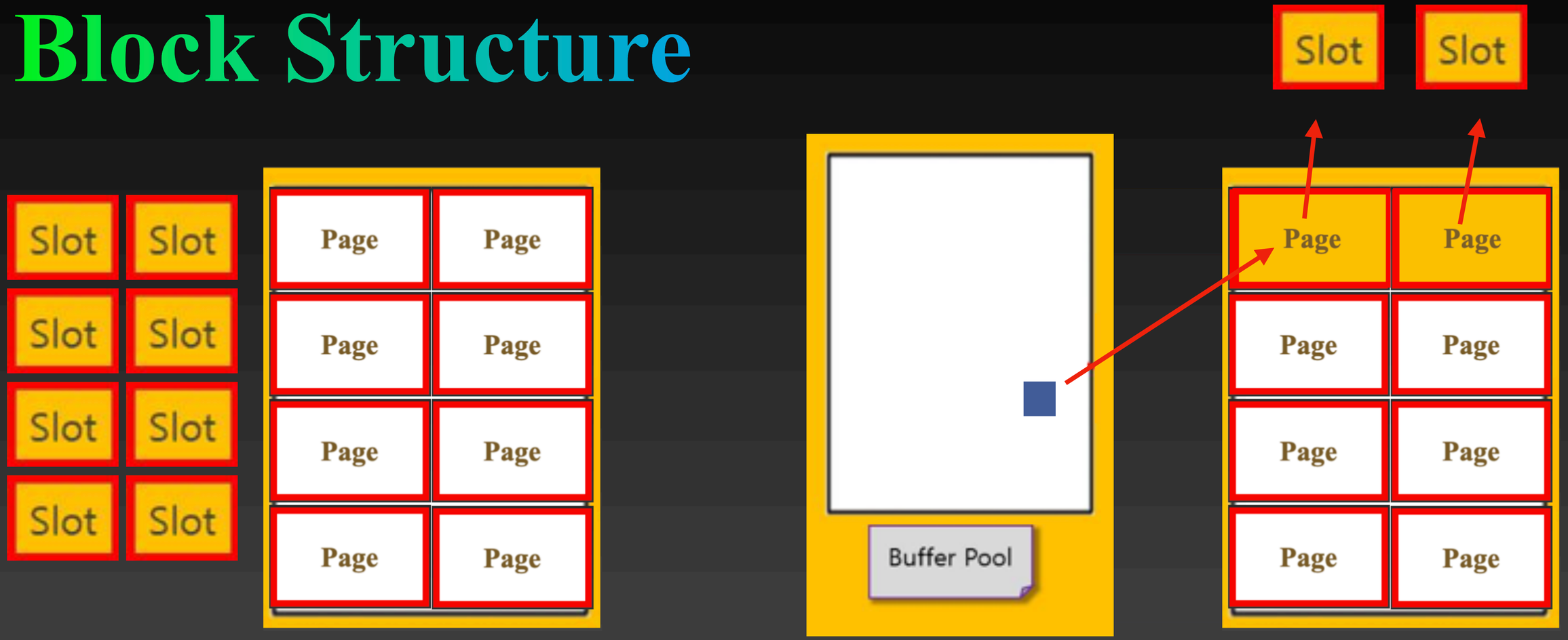




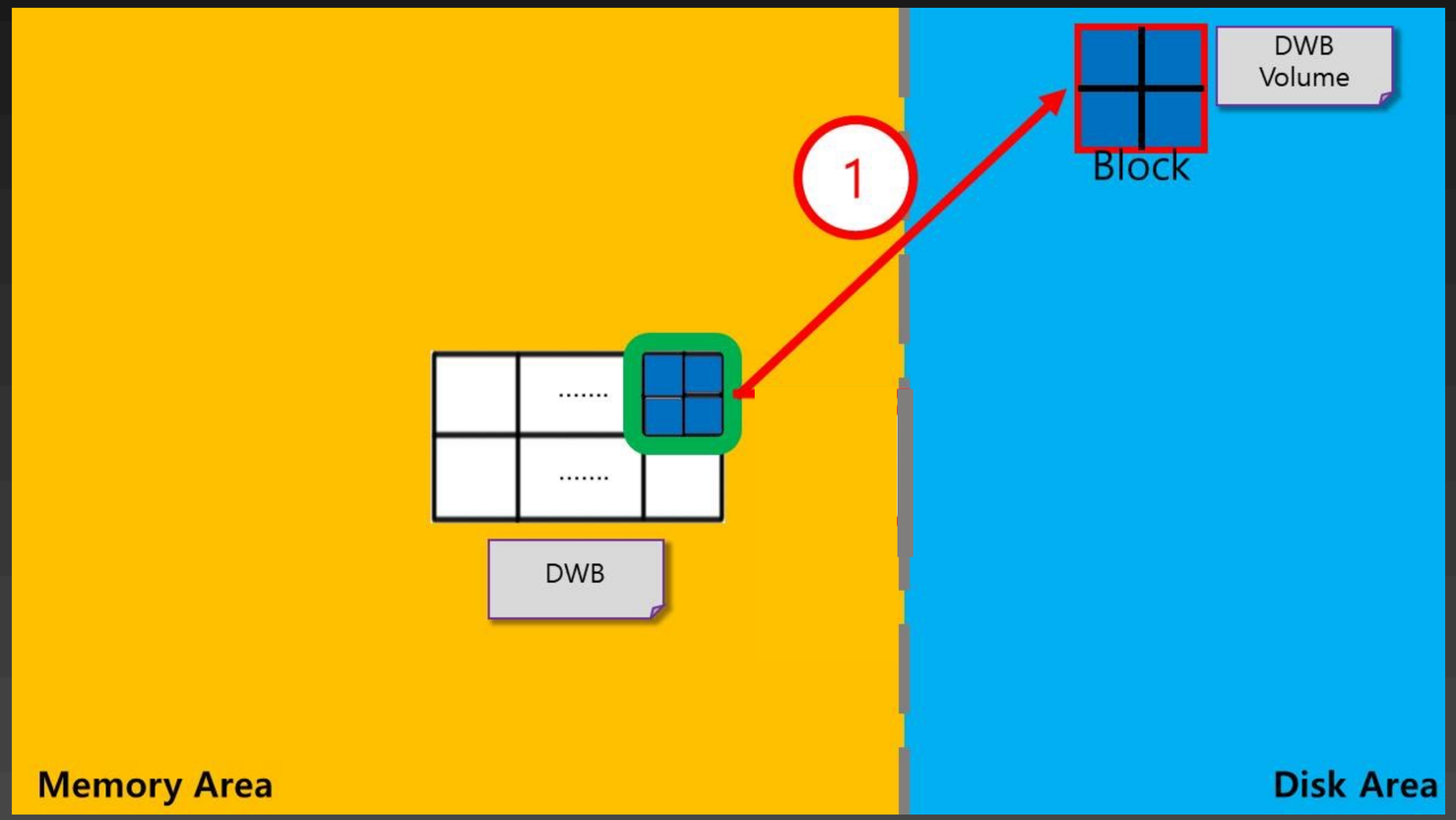
Block Structure



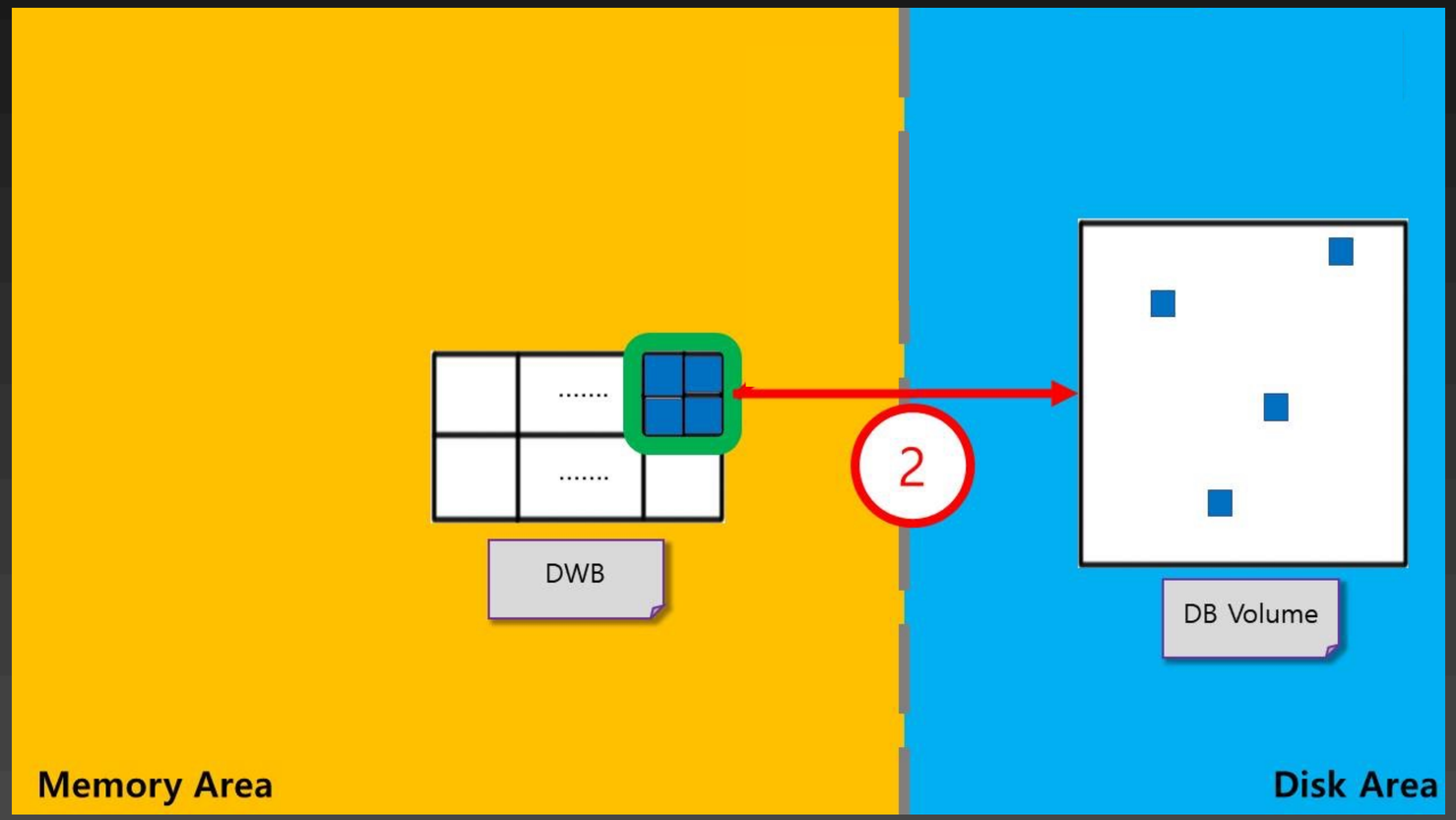
Block Structure



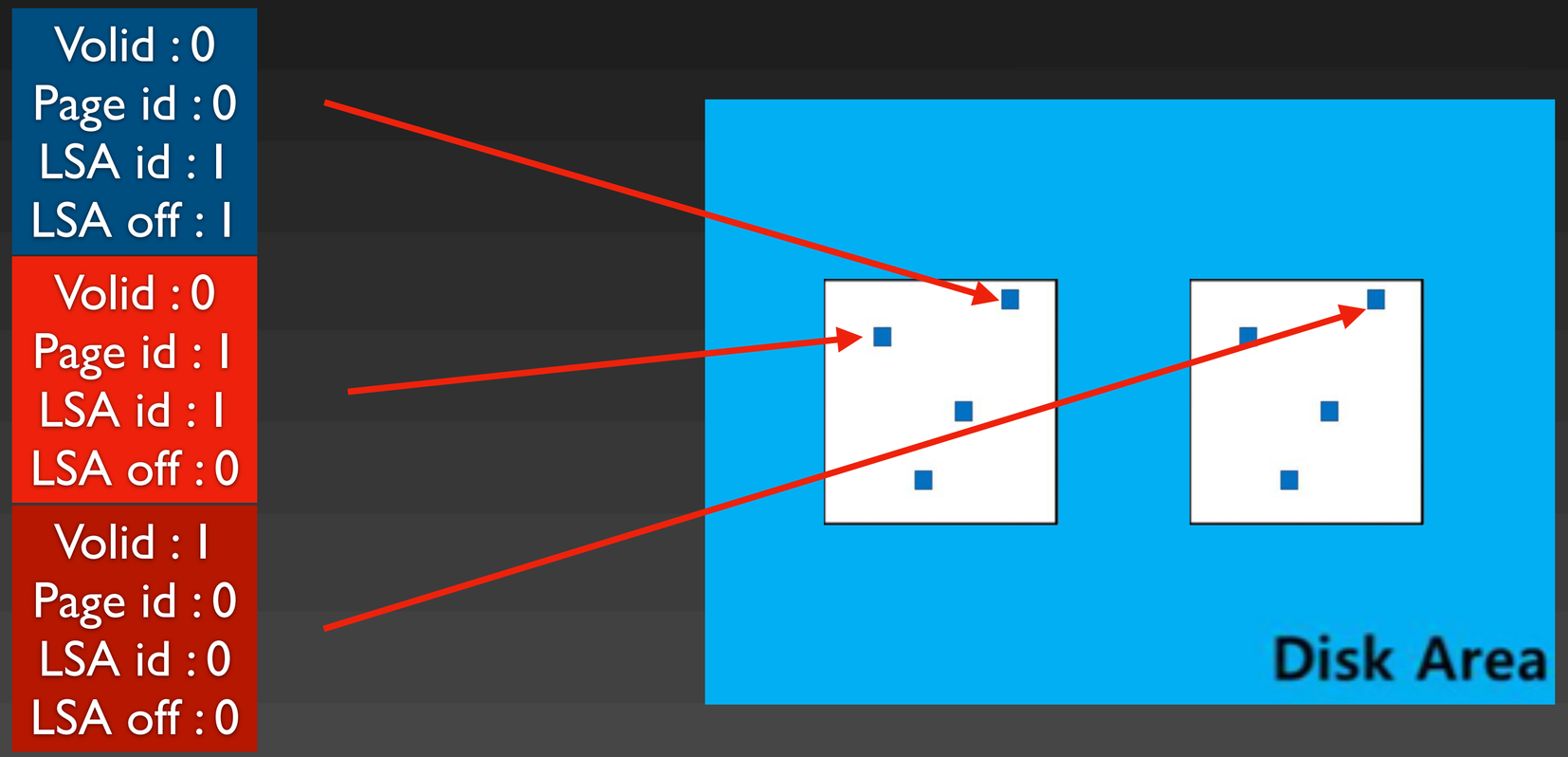
DWB Volume Flush



DB Volume Flush



DB Volume Flush



Sync Daemon

dwb flush block daemon가 호출하는 **daemon**

주기적으로 **DB**로 **Page**를 **flush**한다

사용하는 이유

dwb volume 에 데이터가 이미 저장되어 우선순위 작업이 아니기때문에

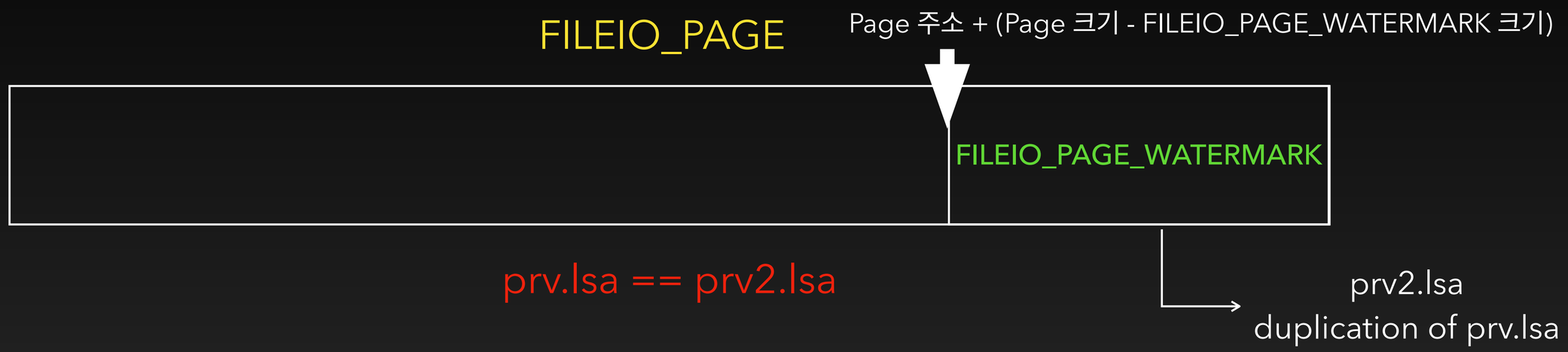
Page 복구

dwb_load_and_recover_pages()

CUBRID server가 재시작될 때, Disk의 DWB Volume을 이용해 복구를 진행한다.

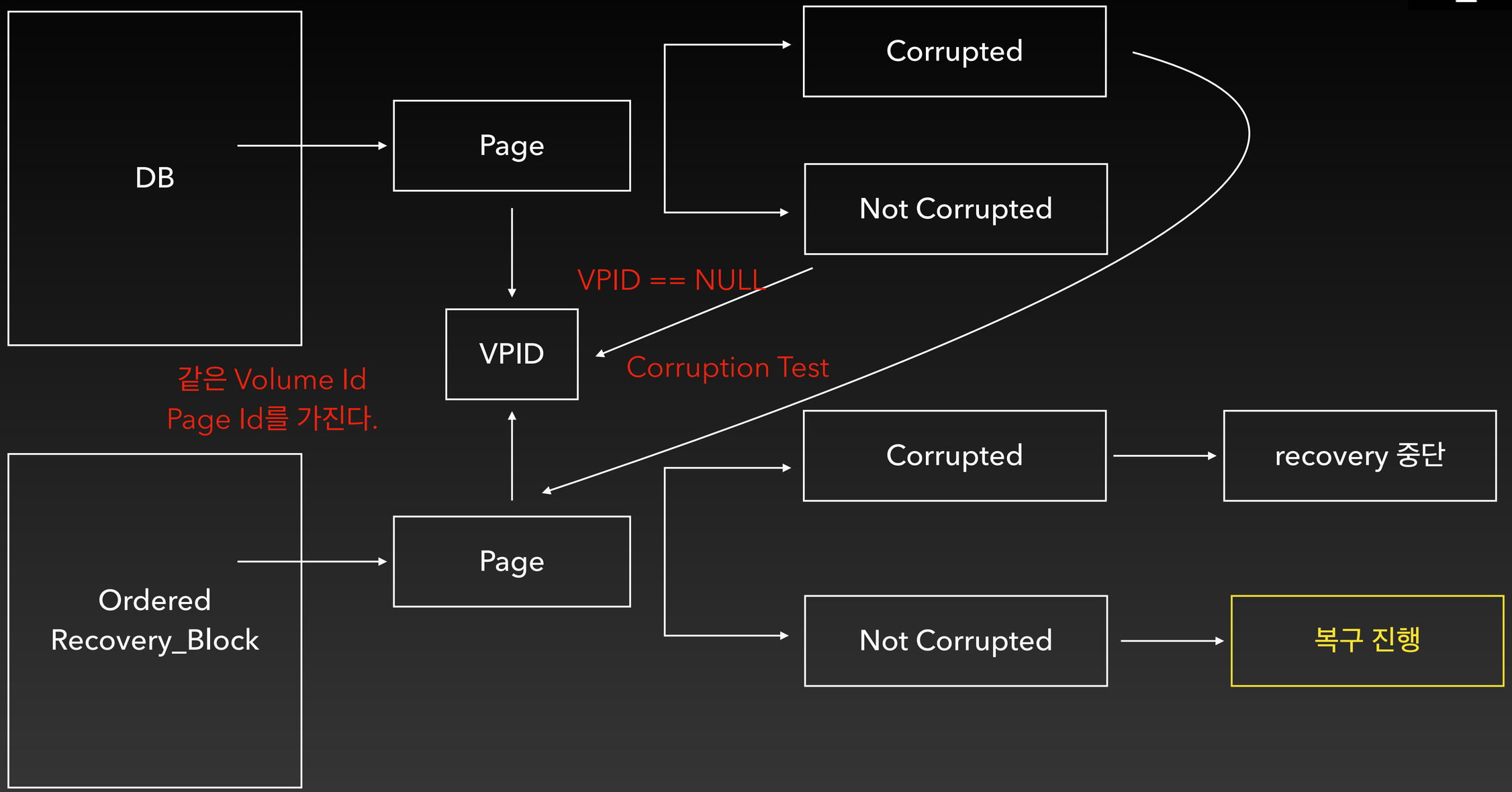
- Disk에 저장된 DWB Volume이 존재하는지 확인 후, mount 시킨다.
- 정상적인 DWB Volume은 2의 제곱승의 크기를 가진다. 이 경우에만 DWB를 이용해 복구를 진행하고, 그 외의 경우에는 복구를 하지 않는다.
- 복구를 위한 Block(rcv_block)을 메모리에 생성한다. 이 Block은 DWB Volume의 page의 갯수만큼 slot을 가지고 있다.
- DWB Volume의 page들을 rcv_block으로 읽어오고 slot의 VPID, LSA도 가져온다.
- rcv_block의 Slot들을 VPID, LSA 순으로 정렬한다. LSA 를 비교하여 더 최신의 LSA를 남긴다. 같은 LSA를 가질 경우, DWB에 마지막으로 Flush된 Slot의 LSA를 남긴다.

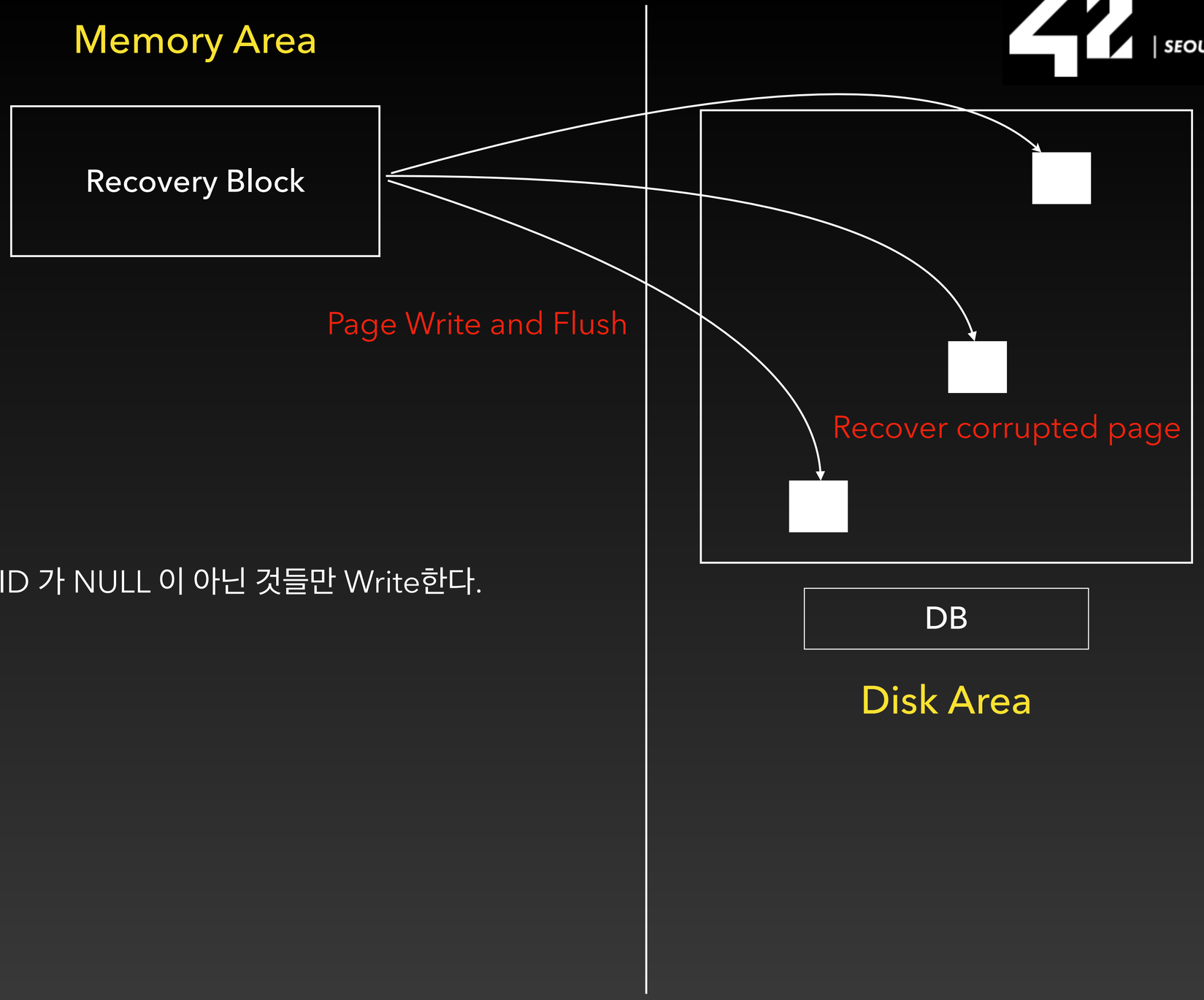
Page Corruption Test




```
struct fileio_page
{
  FILEIO_PAGE_RESERVED prv; /* System
  char page[1];          /* The user p

  // You cannot directly access prv2
  FILEIO_PAGE_WATERMARK prv2;   /* sy
};
```


DB
Page
Corrupted
Not Corrupted
VPID == NULL
VPID
Corruption Test
같은 Volume Id
Page Id를 가진다.
Ordered
Recovery_Block
Page
Corrupted
recovery 중단
Not Corrupted
복구 진행



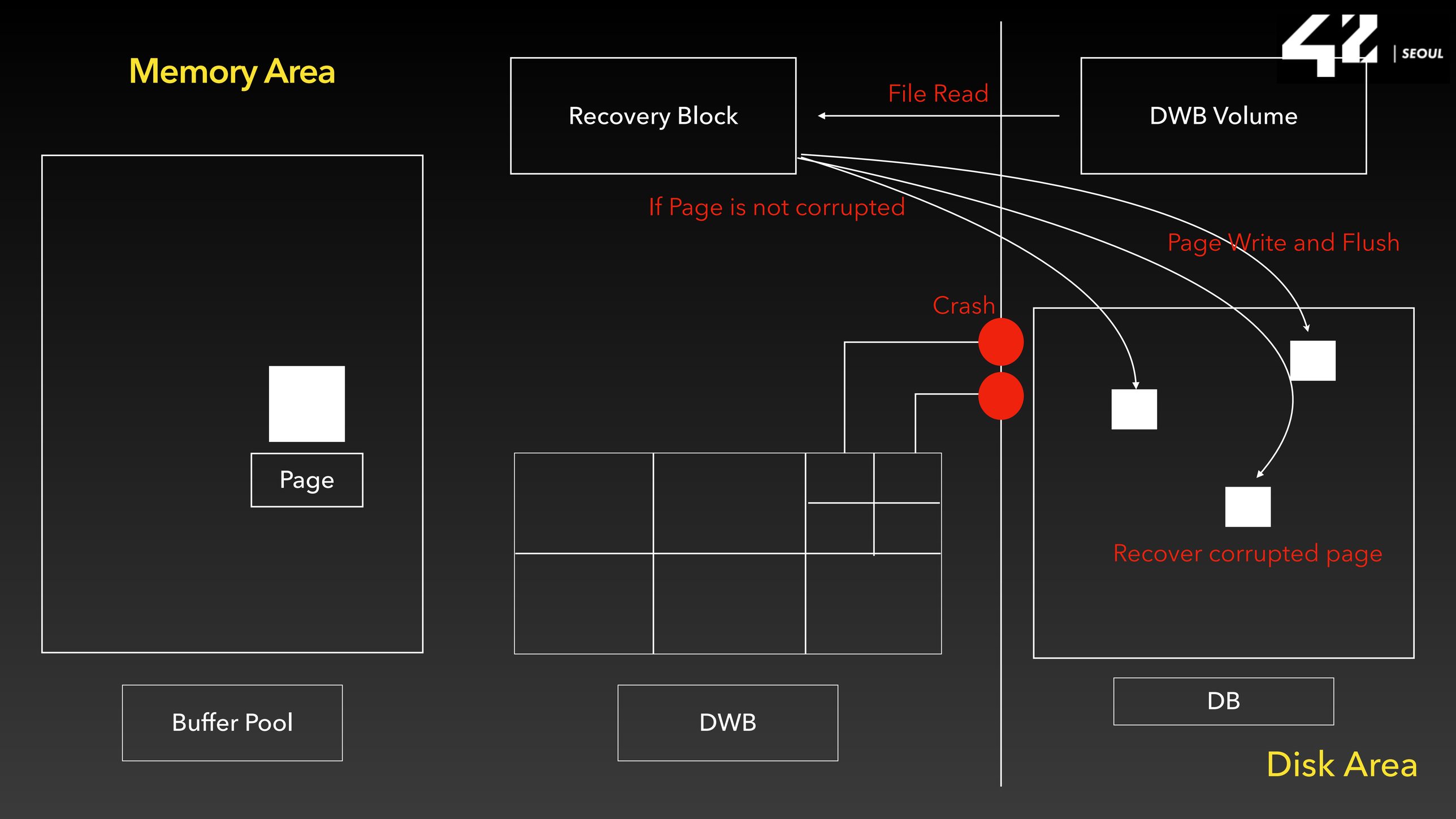
dwb_write_block()

➔ 정렬된 rcv_block을 DB에 Write, VPID 가 NULL 이 아닌 것들만 Write한다.

Flush 진행

DWB Volume dismount, Destroy

DWB_Create()


Memory Area
Recovery Block
File Read
DWB Volume
If Page is not corrupted
Page Write and Flush
Crash
Page
Recover corrupted page
Buffer Pool
DWB
DB
Disk Area