Greedy FTL

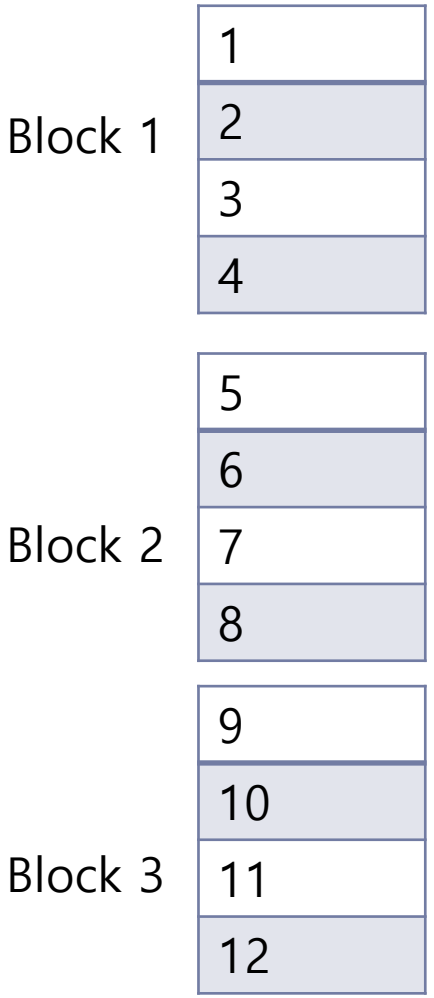
Sejun Kwon(sejun000@csl.skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

Contents

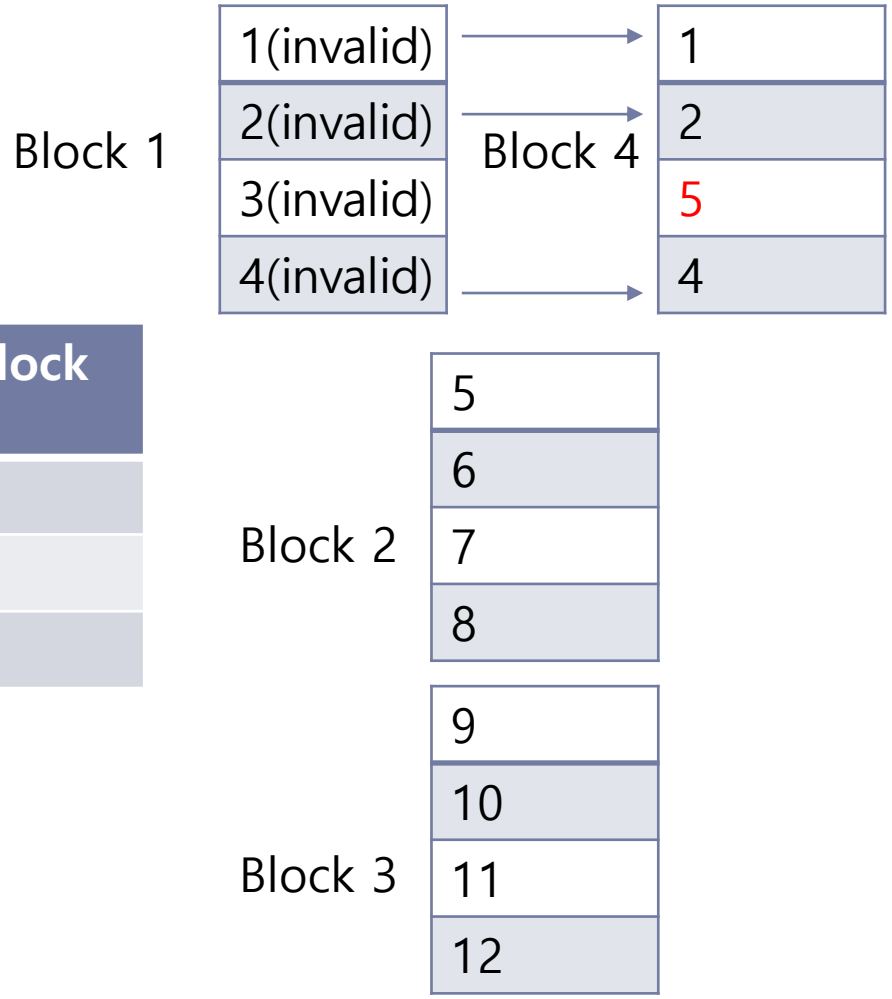
- Block Level Mapping
- Page Level Mapping
- Garbage Collection
- Power-Off Recovery
- Greedy FTL

Block Level mapping

LBN(Logical Block Number)	PBN(Physical Block Number)
1	1
2	2
3	3



Block Level mapping



LBN(Logical Block Number)	PBN(Physical Block Number)
1	4
2	2
3	3

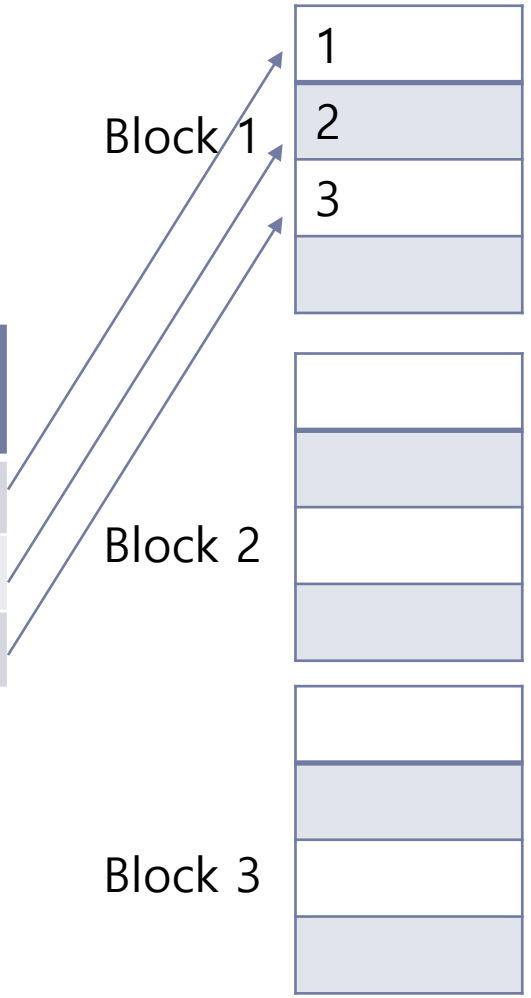
Block Level mapping

- Disadvantage
  - Data to copy is a lot.
  - Amount of Writes is heavy

- Advantage
  - Metadata
  - Cache hit(According to Hardware)

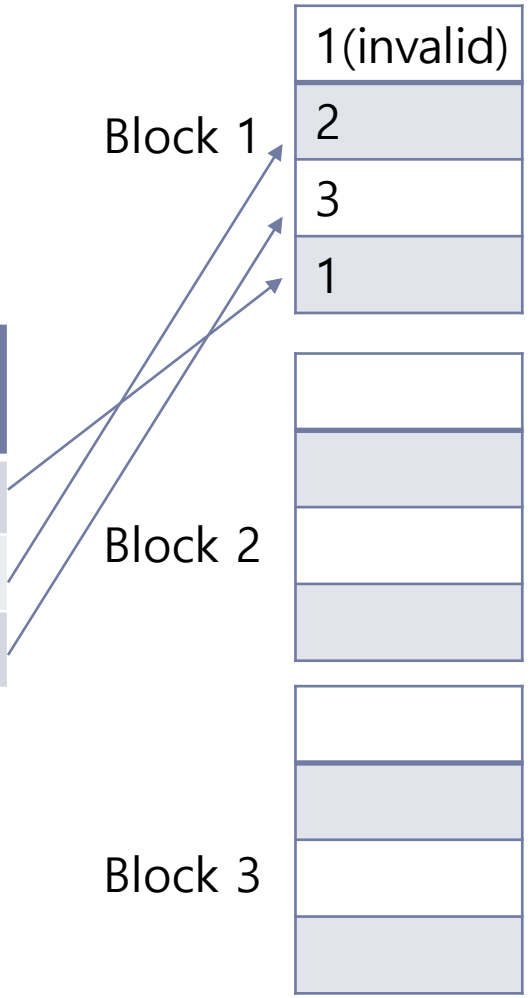
Page Level mapping

LPN(Logical Page Number)	PPN(Physical Page Number)
1	1
2	2
3	3



Page Level mapping

LBN(Logical Block Number)	PPN(Physical Block Number)
1	4
2	2
3	3



Page Level mapping

- Advantage
  - Amount of copy is low
  - Simple

- Disadvantage
  - Heavy metadata

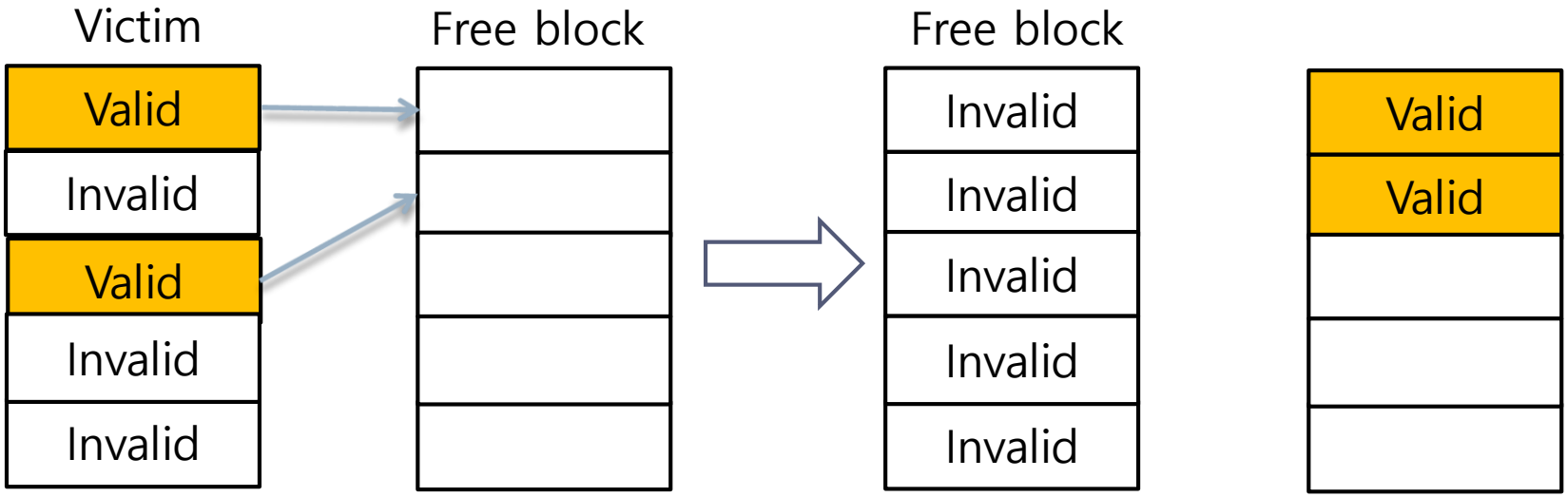
  (2048 vblocks/bank * 128 pages/vblock * 8 banks * 4 bytes = 8 MB)
  - If DRAM does not exist?

Greedy FTL

- Page Level Mapping
- Garbage Collection with “Greedy” policy

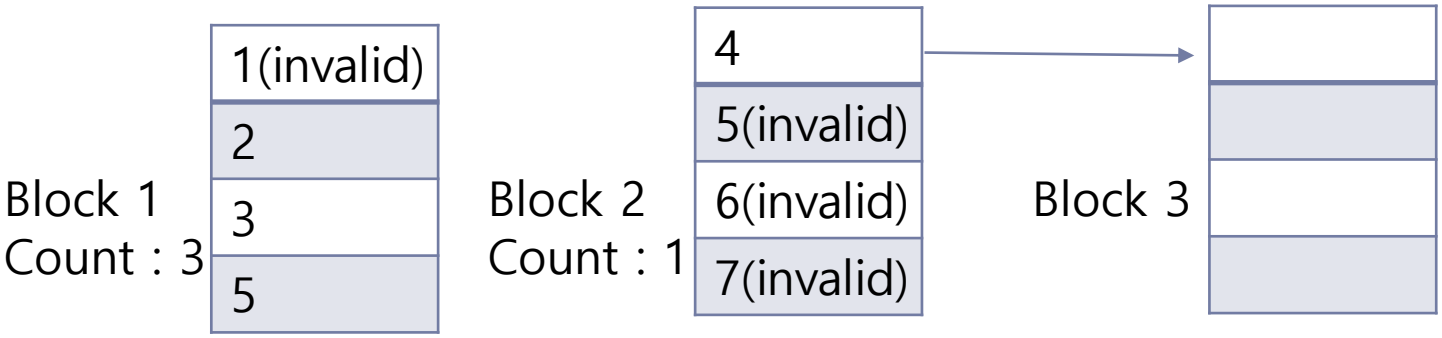
Garbage Collection

- Select victim block
- Copy valid pages to free block
- Erase victim block



Garbage Collection

- Victim block
  - Greedy FTL always select the block of minimum vaild count as a victim block.



FTL Metadata

- Data
  - Contents vs Metadata
- Types of FTL Metadata
  - Free block count
  - Page-level mapping table
  - Bad block bitmap table
  - Valid page information
  - …

Greedy FTL

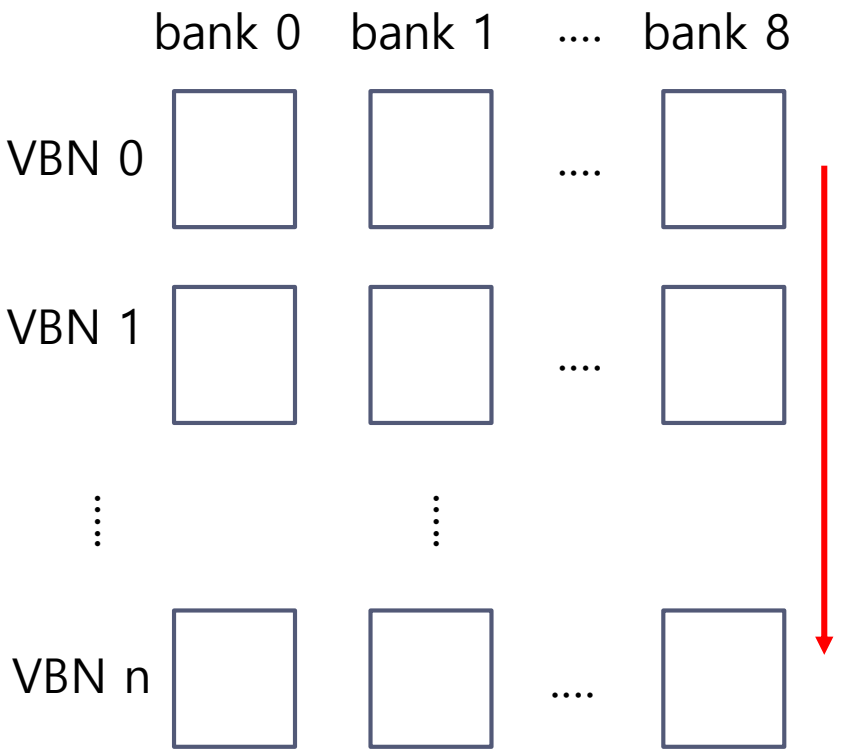
- ./ftl_greedy
  - ftl.c, ftl.h

- Page Mapping FTL
  - Support simple garbage collection
    - Victim selection policy : Greedy
  - Support normal power-off recovery

Write Process

LPN = LBA / SECTORS_PER_PAGE

BANK = LPN % NUM_BANKS



- Write data sequentially from the first block of each bank.

Metadata

- SRAM : very fast, only 96KB(Code + Data)
- DRAM : intermediate speed, 64MB
- NAND : slow, 64GB, 4MB per one vblock .

FTL Metadata

- SRAM Metadata
  - ./ftl_greedy/ftl.c

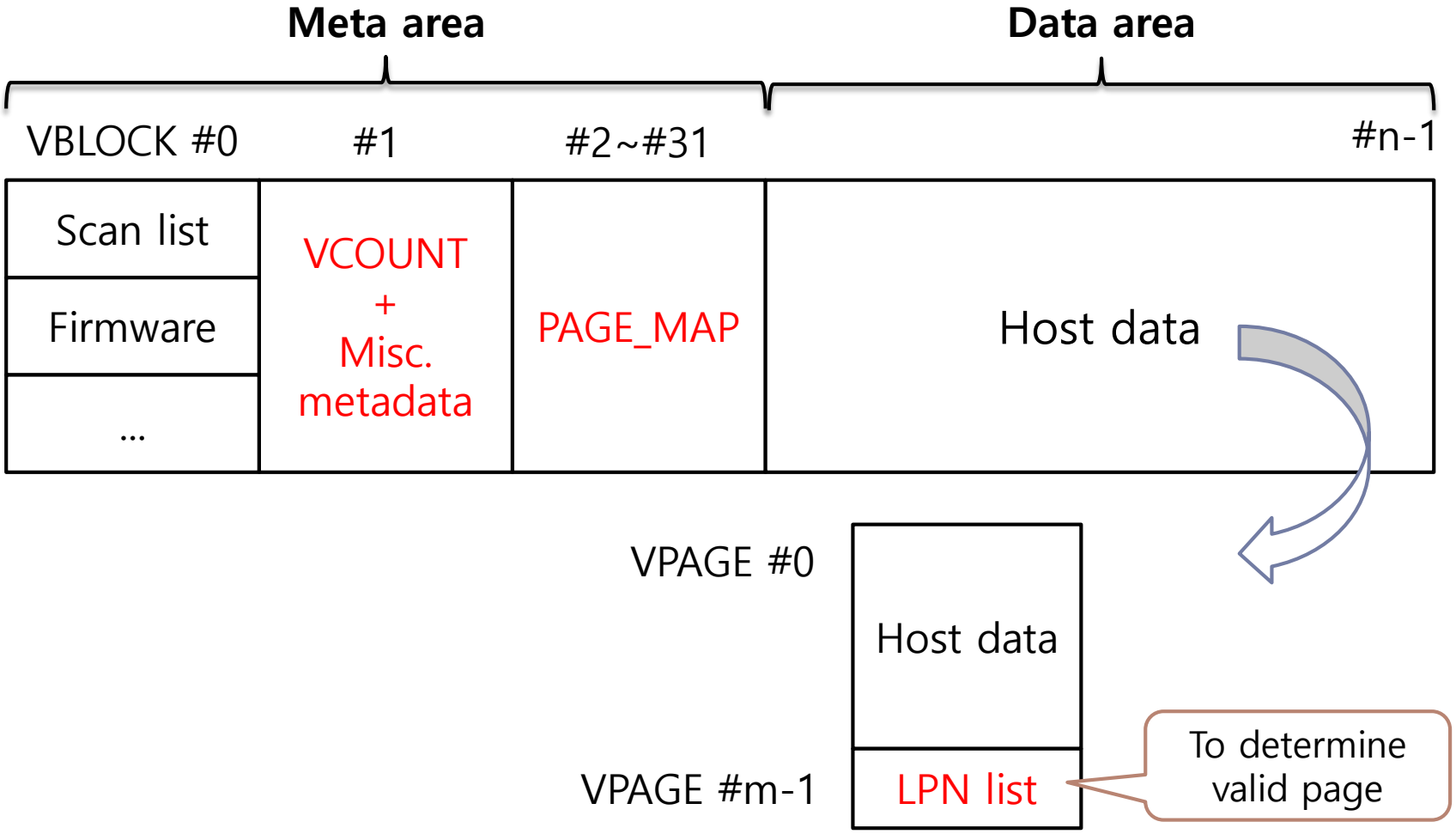
```
//----------------------------------
// metadata structure
//----------------------------------
typedef struct _ftl_statistics
{
    UINT32 gc_cnt;
    UINT32 page_wcount; // page write count
}ftl_statistics;

typedef struct _misc_metadata
{
    UINT32 cur_write_vpn; // physical page for new write
    UINT32 cur_miscblk_vpn; // current write vpn for logging the misc. metadata
    UINT32 cur_mapblk_vpn[MAPBLKS_PER_BANK]; // current write vpn for logging the age mapping info.
    UINT32 gc_vblock; // vblock number for garbage collection
    UINT32 free_blk_cnt; // total number of free block count
    UINT32 lpn_list_of_cur_vblock[PAGES_PER_BLK]; // logging lpn list of current write vblock for GC
}misc_metadata; // per bank

//----------------------------------
// FTL metadata (maintain in SRAM)
//----------------------------------
static misc_metadata  g_misc_meta[NUM_BANKS];
static ftl_statistics g_ftl_statistics[NUM_BANKS];
static UINT32         g_bad_blk_count[NUM_BANKS];
```

FTL Metadata

- DRAM Metadata
  - ./ftl_greedy/ftl.h

```
#define BAD_BLK_BMP_ADDR    (TEMP_BUF_ADDR + TEMP_BUF_BYTES)                  // bitmap of initial bad blocks
#define BAD_BLK_BMP_BYTES   (((NUM_VBLKS / 8) + DRAM_ECC_UNIT - 1) / DRAM_ECC_UNIT * DRAM_ECC_UNIT)

#define PAGE_MAP_ADDR       (BAD_BLK_BMP_ADDR + BAD_BLK_BMP_BYTES)            // page mapping table
#define PAGE_MAP_BYTES      ((NUM_LPAGES * sizeof(UINT32) + BYTES_PER_SECTOR - 1) \
                              / BYTES_PER_SECTOR * BYTES_PER_SECTOR)

#define VCOUNT_ADDR         (PAGE_MAP_ADDR + PAGE_MAP_BYTES)
#define VCOUNT_BYTES        ((NUM_BANKS * VBLKS_PER_BANK * sizeof(UINT16) + BYTES_PER_SECTOR - 1) \
                              / BYTES_PER_SECTOR * BYTES_PER_SECTOR)
```

NAND Layout


Meta area
Data area
VBLOCK #0
#1
#2~#31
#n-1
Scan list
Firmware
...
VCOUNT
+
Misc.
metadata
PAGE_MAP
Host data
VPAGE #0
Host data
VPAGE #m-1
LPN list
To determine
valid page

FTL Open

- ./ftl_greedy/ftl.c

```
void ftl_open(void)
{
    led(0);
    sanity_check();
    //--------------------------------------
    // read scan lists from NAND flash
    // and build bitmap of bad blocks
    //--------------------------------------
    build_bad_blk_list();

    //--------------------------------------
    // If necessary, do low-level format
    // format() should be called after loading scan lists, because format() calls is_bad_block().
    //--------------------------------------
    if (check_format_mark() == FALSE)
    {
        uart_print("do format");
        format();
        uart_print("end format");
    }
    // load FTL metadata
    else
    {
        load_metadata();
    }
    g_ftl_read_buf_id = 0;
    g_ftl_write_buf_id = 0;

    // This example FTL can handle runtime bad block interrupts and read fail (uncorrectable bit errors) interrupts
    flash_clear_irq();

    SETREG(INTR_MASK, FIRQ_DATA_CORRUPT | FIRQ_BADBLK_L | FIRQ_BADBLK_H);
    SETREG(FCONF_PAUSE, FIRQ_DATA_CORRUPT | FIRQ_BADBLK_L | FIRQ_BADBLK_H);

    enable_irq();
} ? end ftl_open ?
```

Bad Block

- ./ftl_greedy/ftl.h

```
static void build_bad_blk_list(void)
{
    UINT32 bank, num_entries, result, vblk_offset;
    scan_list_t* scan_list = (scan_list_t*) TEMP_BUF_ADDR;

    mem_set_dram(BAD_BLK_BMP_ADDR, NULL, BAD_BLK_BMP_BYTES);

    for (bank = 0; bank < NUM_BANKS; bank++)
    {
        SETREG(FCP_CMD, FC_COL_ROW_READ_OUT);

        SETREG(FCP_DMA_ADDR, (UINT32) scan_list);
        SETREG(FCP_DMA_CNT, SCAN_LIST_SIZE);
        SETREG(FCP_COL, 0);
        SETREG(FCP_ROW_L(bank), SCAN_LIST_PAGE_OFFSET);
        SETREG(FCP_ROW_H(bank), SCAN_LIST_PAGE_OFFSET);

        SETREG(FCP_ISSUE, NULL);
        while ((GETREG(WR_STAT) & 0x00000001) != 0);
        while (BSP_FSM(bank) != BANK_IDLE);

        ...

        g_bad_blk_count[bank] = 0;

        for (vblk_offset = 1; vblk_offset < VBLKS_PER_BANK; vblk_offset++)
        {
            if (mem_search_equ_dram(scan_list, sizeof(UINT16), num_entries + 1, vblk_offset)
                    < num_entries + 1)
            {
                g_bad_blk_count[bank]++;
                set_bit_dram(BAD_BLK_BMP_ADDR + bank*(VBLKS_PER_BANK/8 + 1), vblk_offset);
            }
        }
    } ? end for bank=0;bank<NUM_BANKS... ?
} ? end build_bad_blk_list ?
```

Block #0, Page #0 contains bad block list

Read Operation

```
void ftl_read(UINT32 const lba, UINT32 const num_sectors)
{
    UINT32 remain_sects, num_sectors_to_read;
    UINT32 lpn, sect_offset;
    UINT32 bank, vpn;

    lpn          = lba / SECTORS_PER_PAGE;
    sect_offset  = lba % SECTORS_PER_PAGE;
    remain_sects = num_sectors;

    while (remain_sects != 0)
    {
        if ((sect_offset + remain_sects) < SECTORS_PER_PAGE)
        {
            num_sectors_to_read = remain_sects;
        }
        else
        {
            num_sectors_to_read = SECTORS_PER_PAGE - sect_offset;
        }
        bank = get_num_bank(lpn); // page striping
        vpn  = get_vpn(lpn);
        CHECK_VPAGE(vpn);

        if (vpn != NULL)
        {
            nand_page_ptread_to_host(bank,
                                     vpn / PAGES_PER_BLK,
                                     vpn % PAGES_PER_BLK,
                                     sect_offset,
                                     num_sectors_to_read);
        }
        // The host is requesting to read a logical page that has never been written to.
        else
        {
            ...
        }
        sect_offset   = 0;
        remain_sects -= num_sectors_to_read;
        lpn++;
    } ? end while remain_sects! =0 ?
} ? end ftl_read ?
```

Write Operation

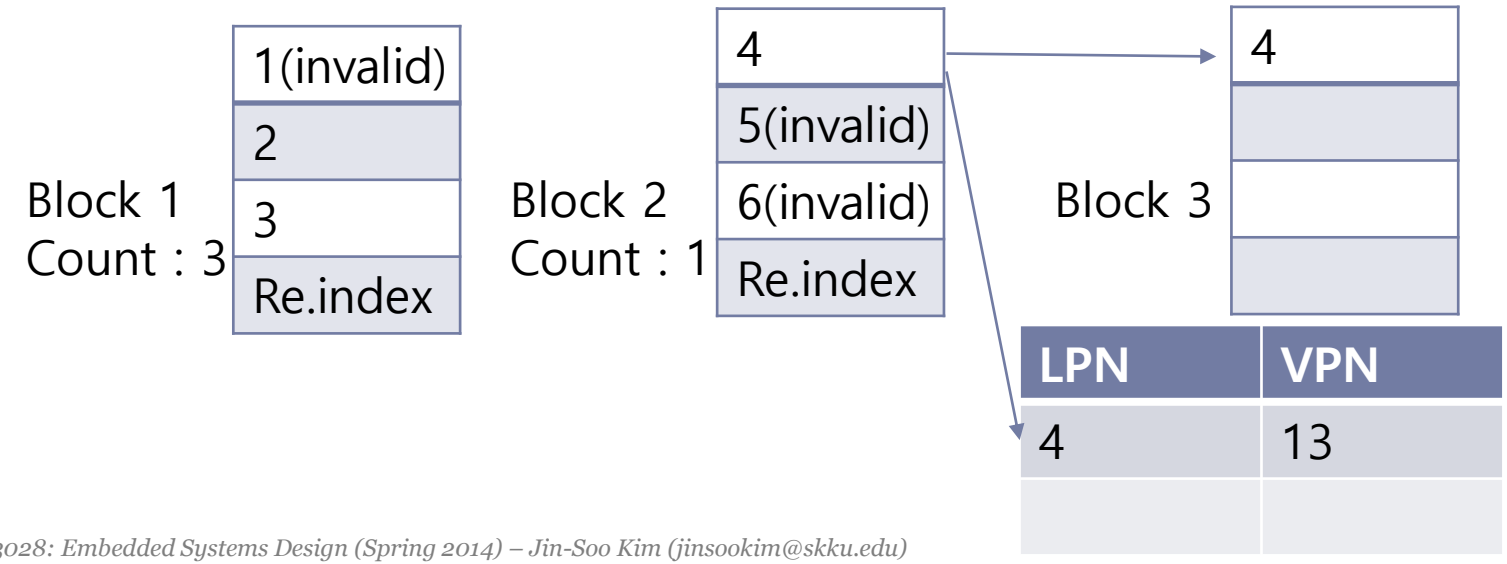
```
static void write_page(UINT32 const lpn, UINT32 const sect_offset, UINT32 const num_sectors)
{
    UINT32 bank, old_vpn, new_vpn;
    UINT32 vblock, page_num, page_offset, column_cnt;
    ...
    new_vpn  = assign_new_write_vpn(bank);
    old_vpn  = get_vpn(lpn);

    // if old data already exist,
    if (old_vpn != NULL)
    {
        ...
        if (num_sectors != SECTORS_PER_PAGE)
        {
            // read `left hole sectors'
            if (page_offset != 0)
            {
                nand_page_ptread(bank, vblock, page_num, 0, page_offset,
                                 WR_BUF_PTR(g_ftl_write_buf_id), RETURN_ON_ISSUE);
            }
            // read `right hole sectors'
            if ((page_offset + column_cnt) < SECTORS_PER_PAGE)
            {
                nand_page_ptread(bank, vblock, page_num, page_offset + column_cnt,
                                 SECTORS_PER_PAGE - (page_offset + column_cnt),
                                 WR_BUF_PTR(g_ftl_write_buf_id), RETURN_ON_ISSUE);
            }
        }
        // full page write
        page_offset = 0;
        column_cnt  = SECTORS_PER_PAGE;
        // invalid old page (decrease vcount)
        set_vcount(bank, vblock, get_vcount(bank, vblock) - 1);
    } ?end if old_vpn != NULL?
    vblock   = new_vpn / PAGES_PER_BLK;
    page_num = new_vpn % PAGES_PER_BLK;

    nand_page_ptprogram_from_host(bank, vblock, page_num, page_offset, column_cnt);

    // update metadata
    set_lpn(bank, page_num, lpn);
    set_vpn(lpn, new_vpn);
    set_vcount(bank, vblock, get_vcount(bank, vblock) + 1);
} ?end write_page?
```

Decrease valid page count

Update LPN list
Update page mapping table
Increase valid page count

Garbage Collection

- Reverse index
  - For remapping logcal page to physical page, reverse index(physical to logical) is necessary
  - Greedy FTL uses a bottom page of each block as reverse index.



Garbage Collection

- ./ftl_greedy/ftl.c

```
static UINT32 get_vt_vblock(UINT32 const bank)
{
    ASSERT(bank < NUM_BANKS);

    UINT32 vblock;

    // search the block which has mininum valid pages
    vblock = mem_search_min_max(VCOUNT_ADDR + (bank * VBLKS_PER_BANK * sizeof(UINT16)),
                                sizeof(UINT16),
                                VBLKS_PER_BANK,
                                MU_CMD_SEARCH_MIN_DRAM);

    ASSERT(is_bad_block(bank, vblock) == FALSE);
    ASSERT(vblock >= META_BLKS_PER_BANK && vblock < VBLKS_PER_BANK);
    ASSERT(get_vcount(bank, vblock) < (PAGES_PER_BLK - 1));

    return vblock;
}
```

Using memory utility, search victim block which has the minimum valid page count

Any Questions?