

Greedy FTL

Yeongjae Woo (yjwoo@csl.skku.edu)

Computer Systems Laboratory

Sungkyunkwan University

<http://csl.skku.edu>

Contents

- DRAM Controller
- FTL Metadata
- Garbage Collection
- Power-Off Recovery
- Greedy FTL

Schedule

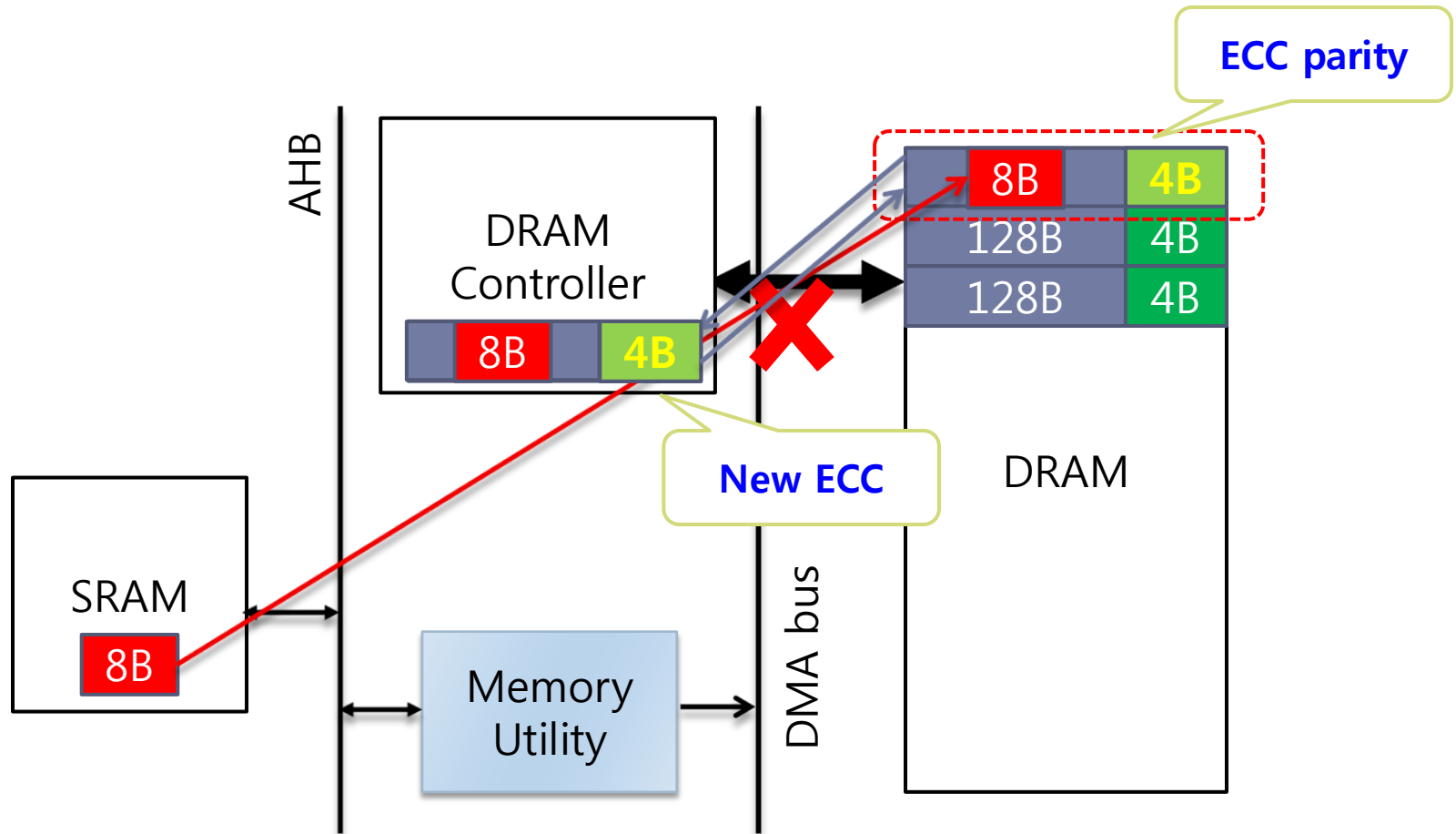
Date	Title
3/15 (Th)	Intro. to the Jasmine OpenSSD Platform
3/22 (Th)	Dummy FTL
3/29 (Th)	Iometer
4/5 (Th)	Tutorial FTL
4/12 (Th)	Greedy FTL, Reliability Issues
5/14 (Th)	Project #1 Presentation
5/17 (Th)	Project #2 Proposal
6/7 (Th)	Project #2 Progress Report
6/25 (M)	Project #2 Presentation

DRAM Controller

- **DRAM ECC**
 - Check and correct DRAM bit errors
 - 128 Byte + 4 Byte ECC Parity
- **Memory Utility**
 - Support data transmission from/to DRAM
 - SRAM to DRAM, DRAM to SRAM, DRAM to DRAM
 - `./include/mem_util.h`

DRAM Controller

- Write 8 Byte data from SRAM to DRAM



FTL Metadata

- Metadata
 - Userdata vs Metadata

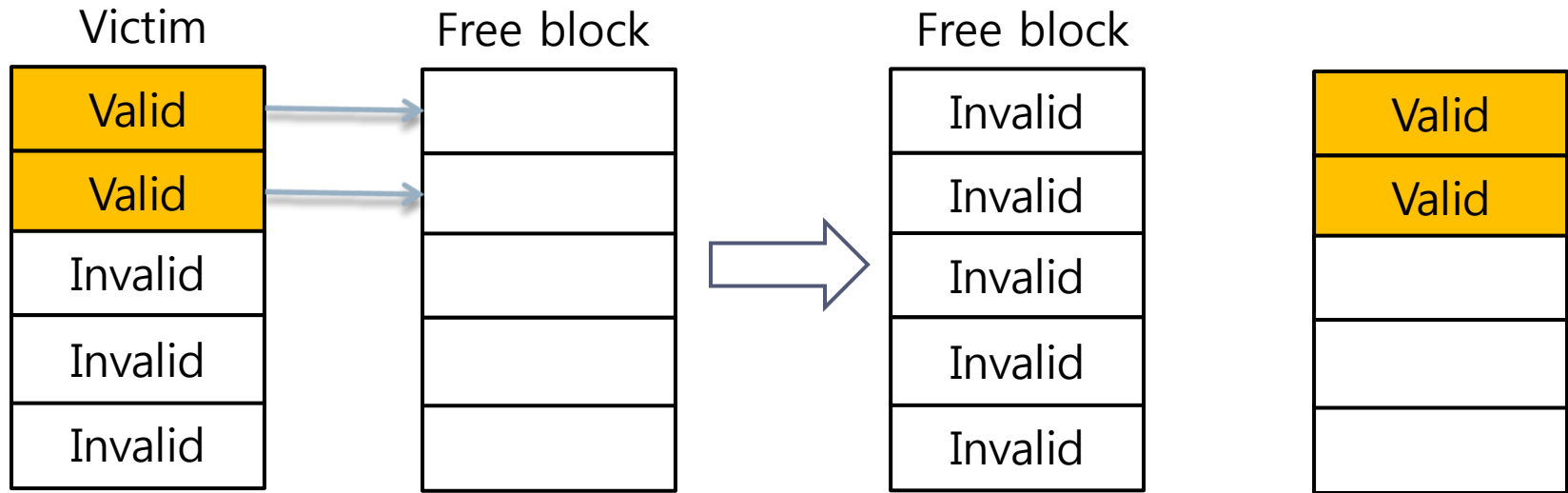
- Types of FTL Metadata
 - Free block count
 - Page-level mapping table
 - Bad block bitmap table
 - Valid page count information
 - ...

FTL Metadata

- Considerations
 - Where to locate
 - SRAM, DRAM, NAND
 - Performance impact
 - Metadata size
 - When to store into NAND
 - Can be reconstructed?

Garbage Collection

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



Power-Off Recovery

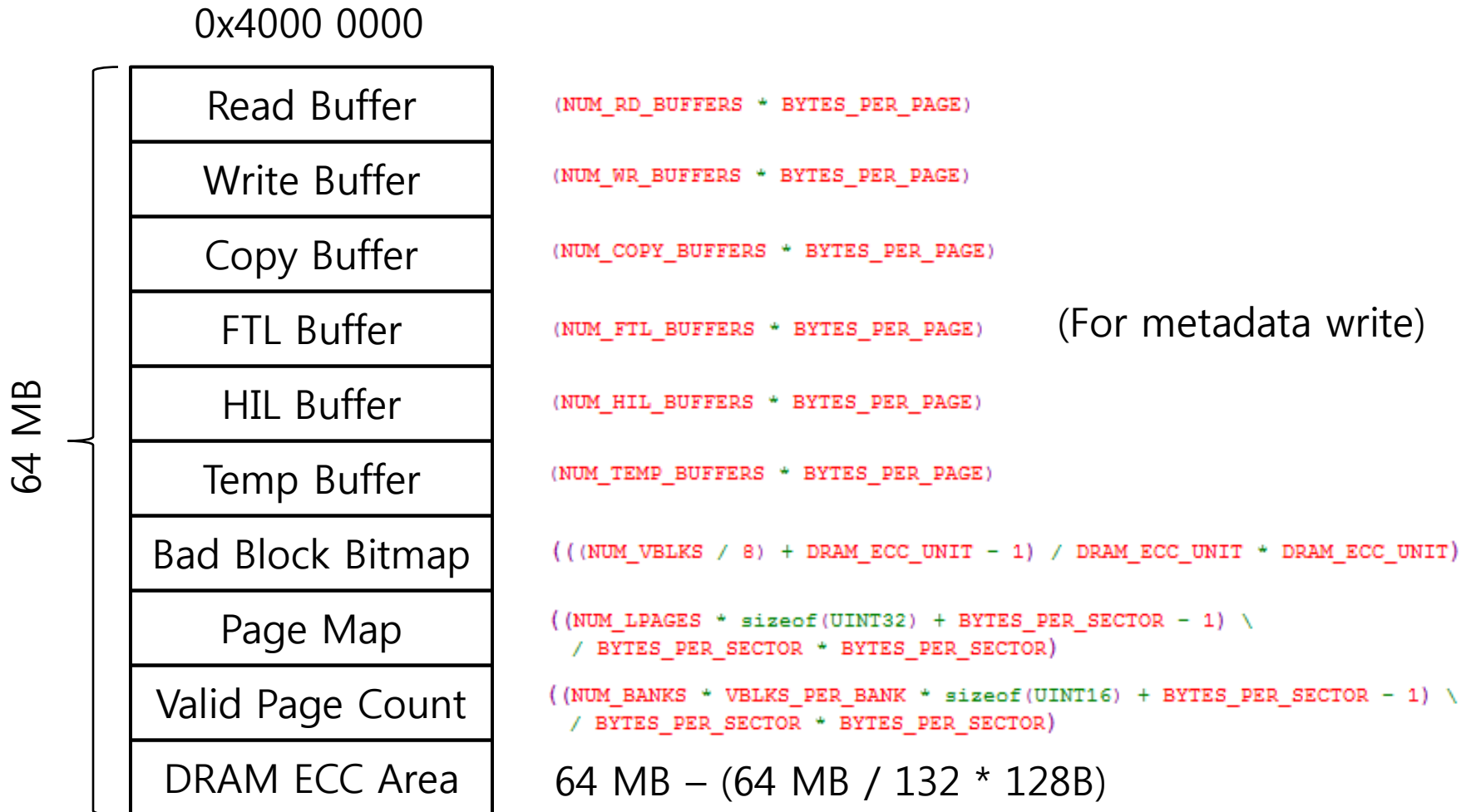
- Power-off leads the data loss
 - Userdata that resides in write buffer
 - Metadata that resides in SRAM and DRAM
 - Page-level mapping table
- Recover mapping table on next power-on
 - On program operation, store LPN into spare area
 - By scanning spare area, mapping table can be reconstructed

Greedy FTL

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

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

DRAM Layout



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_misblk_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

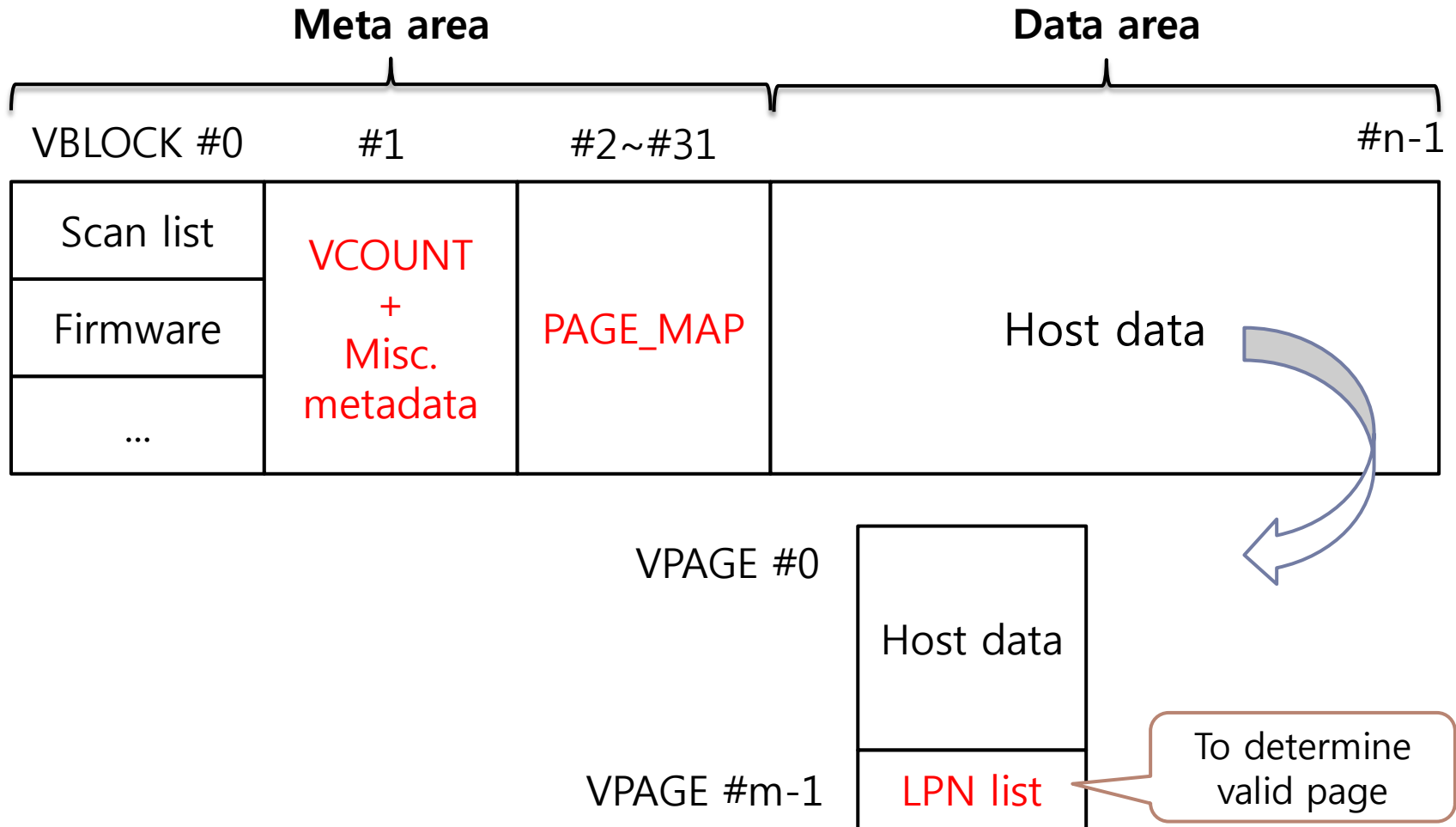
- DRAM Metadata
 - ./ftl_greedy/ftl.h

```
#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



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 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

- `./ftl_greedy/ftl.c`

```
void ftl_write(UINT32 const lba, UINT32 const num_sectors)
{
    UINT32 remain_sects, num_sectors_to_write;
    UINT32 lpn, sect_offset;

    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_write = remain_sects;
        }
        else
        {
            num_sectors_to_write = SECTORS_PER_PAGE - sect_offset;
        }
        // single page write individually
        write_page(lpn, sect_offset, num_sectors_to_write);

        sect_offset  = 0;
        remain_sects -= num_sectors_to_write;
        lpn++;
    }
} ? end ftl_write ?
```

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);

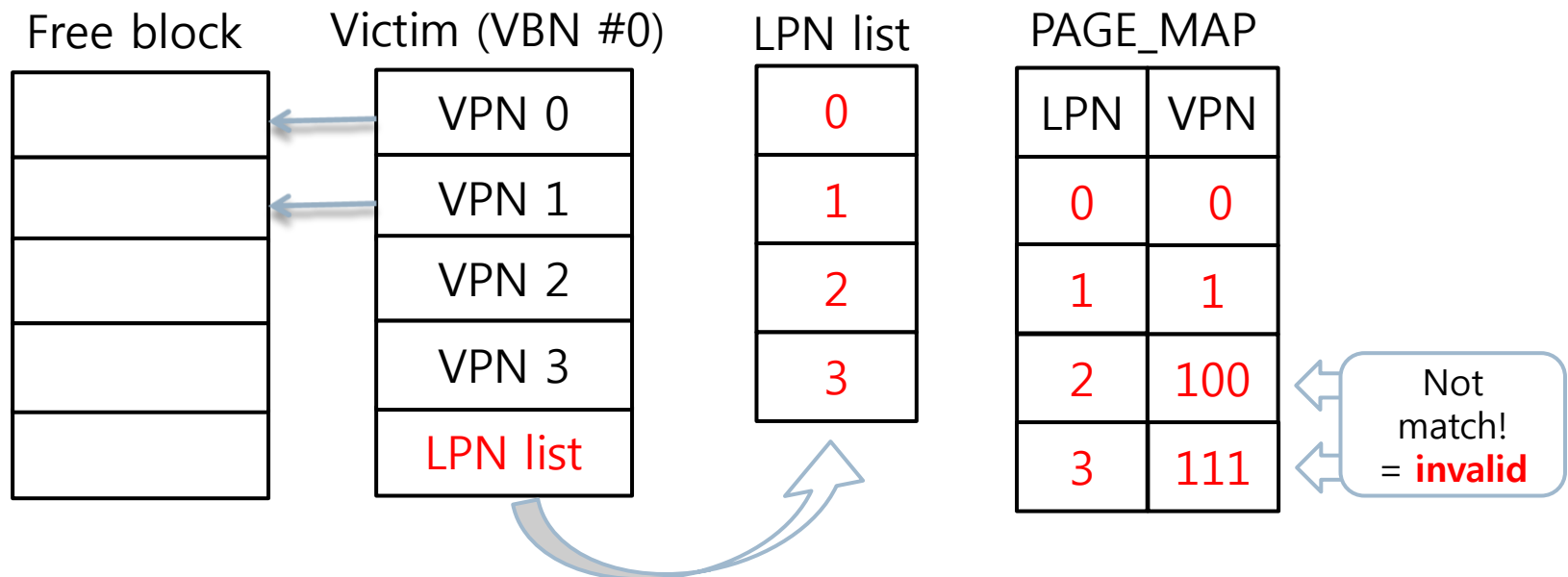
    // update metadata
    set_lpn(bank, page_num, lpn);
    set_vpn(lpn, new_vpn);
    set_vcount(bank, vblock, get_vcount(bank, vblock) + 1);
} // end write_page ?
```

Decrease valid page count

Update LPN list
Update page mapping table
Increase valid page count

Garbage Collection

- Greedy policy
 - Select victim block which has the maximum invalid pages



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 minimum 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_BLK_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

```

static void garbage_collection(UINT32 const bank)
{
    UINT32 src_lpn, vt_vblock, free_vpn, vcount, src_page, gc_vblock;

    vt_vblock = get_vt_vblock(bank); // get victim block
    vcount = get_vcount(bank, vt_vblock);
    gc_vblock = get_gc_vblock(bank);
    free_vpn = gc_vblock * PAGES_PER_BLK;

    // 1. load p2l list from last page offset of victim block (4B x PAGES_PER_BLK)
    nand_page_ptread(bank, vt_vblock, PAGES_PER_BLK - 1, 0,
                    ((sizeof(UINT32) * PAGES_PER_BLK + BYTES_PER_SECTOR - 1) / BYTES_PER_SECTOR),
                    FTL_BUF(bank), RETURN_WHEN_DONE);
    mem_copy(g_misc_meta[bank].lpn_list_of_cur_vblock, FTL_BUF(bank), sizeof(UINT32) * PAGES_PER_BLK);

    // 2. copy-back all valid pages to free space
    for (src_page = 0; src_page < (PAGES_PER_BLK - 1); src_page++)
    {
        // get lpn of victim block from a read lpn list
        src_lpn = get_lpn(bank, src_page);

        // determine whether the page is valid or not
        if (get_vpn(src_lpn) != ((vt_vblock * PAGES_PER_BLK) + src_page))
        {
            // invalid page
            continue;
        }

        // if the page is valid,
        // then do copy-back op. to free space
        nand_page_copyback(bank, vt_vblock, src_page,
                          free_vpn / PAGES_PER_BLK,
                          free_vpn % PAGES_PER_BLK);

        // update metadata
        set_vpn(src_lpn, free_vpn);
        set_lpn(bank, (free_vpn % PAGES_PER_BLK), src_lpn);

        free_vpn++;
    } // end for src_page=0;src_page<(... ?

    // 3. erase victim block
    nand_block_erase(bank, vt_vblock);

    // 4. update metadata
    set_vcount(bank, vt_vblock, VC_MAX);
    set_vcount(bank, gc_vblock, vcount);
    set_new_write_vpn(bank, free_vpn); // set a free page for new write
    set_gc_vblock(bank, vt_vblock); // next free block (reserve for GC)
    dec_full_blk_cnt(bank); // decrease full block count
} // end garbage_collection ?

```

Any Questions?