

# Greedy FTL

Prof. Jinkyu Jeong ([jinkyu@skku.edu](mailto:jinkyu@skku.edu))

TA -- Minwoo Ahn ([minwoo.ahn@csl.skku.edu](mailto:minwoo.ahn@csl.skku.edu))

TA -- Donghyun Kim ([donghyun.kim@csl.skku.edu](mailto:donghyun.kim@csl.skku.edu))

Computer Systems Laboratory

Sungkyunkwan University

<http://csl.skku.edu>

# Contents

- Intro. To Greedy FTL
  - Metadata management
  - Garbage collection
  - Misc. (Read & Write operation, Bad block management)

# Greedy FTL

- `./ftl_greedy`
  - `ftl.c`, `ftl.h`
  
- Page mapping FTL
  - Simple GC
    - GC policy: greedy
  - NPO (Normal Power Off) recovery

# Metadata Management

- Types of FTL metadata
  - L2P table
  - Free block count
  - Bad block bitmap
  - Valid page information
  - etc.
- FTL metadata in,
  - SRAM
    - Very fast, but only 96KB (code + data), volatile
  - DRAM
    - Fast but slower than SRAM, 64MB, volatile
  - NAND
    - Slow, 64GB, non-volatile

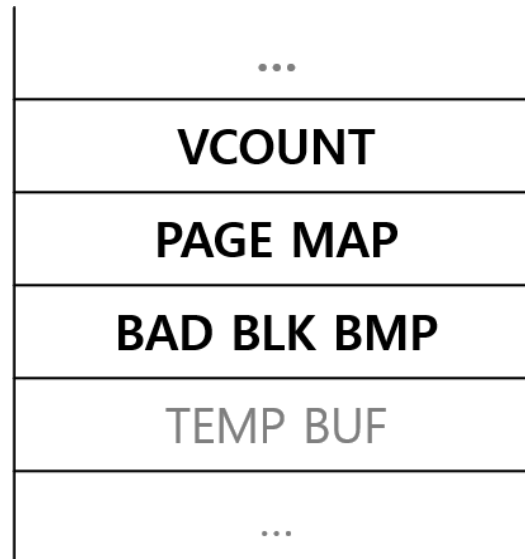
# FTL Metadata in SRAM

- 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 in DRAM

- ftl.h



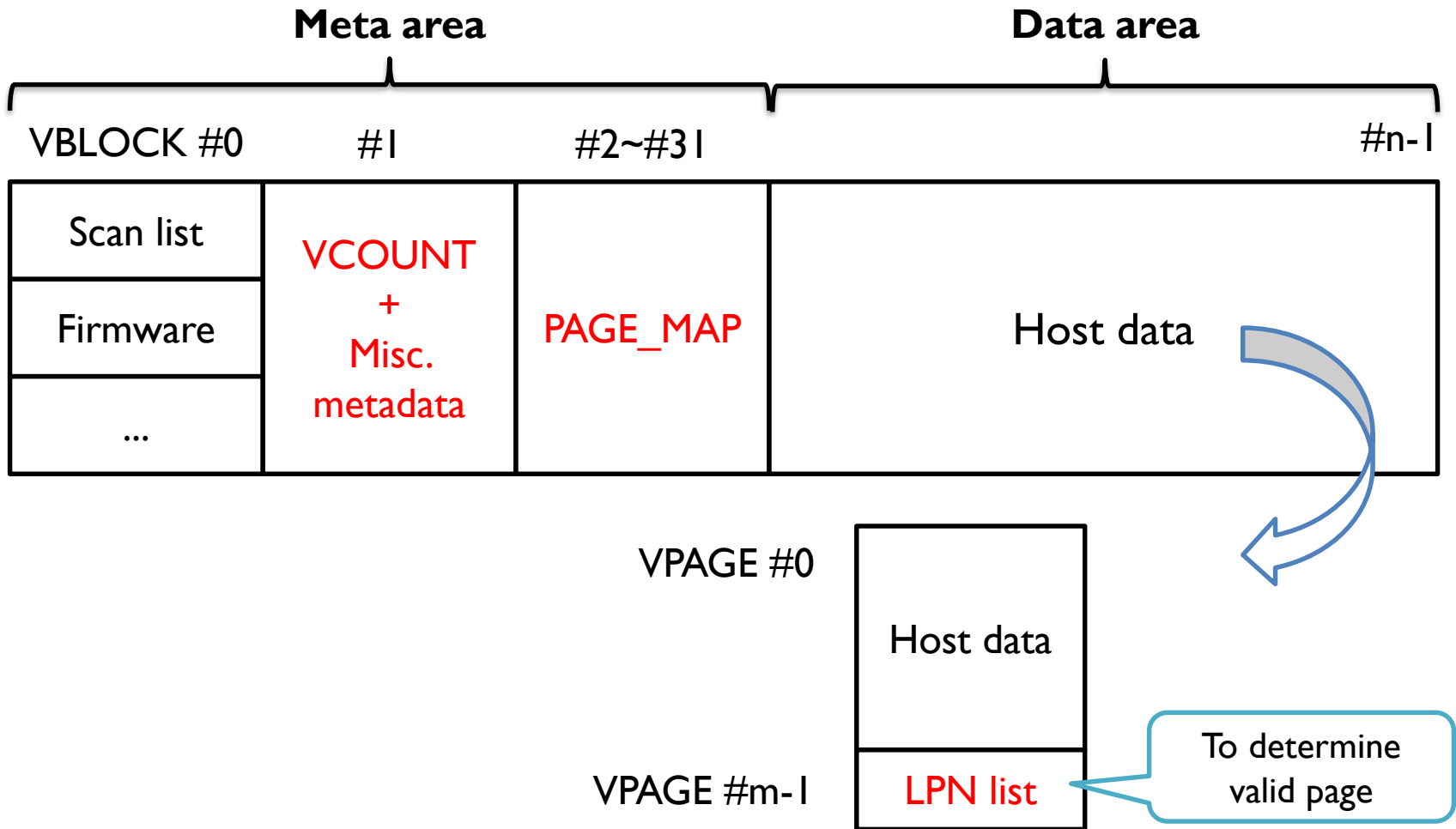
```
#define BAD_BLK_BMP_ADDR      (TEMP_BUF_ADDR + TEMP_BUF_BYTES)           // bitmap of initial bad blocks
#define BAD_BLK_BMP_BYTES    (((NUM_VBLKS / 8) + DRAM_ECC_UNIT - 1) / DRAM_ECC_UNIT * DRAM_ECC_UNIT)

#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)
```

# FTL Metadata in NAND

- Backup layout (“logical” view)



# FTL Metadata in NAND (cont'd)

- 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 ?
```



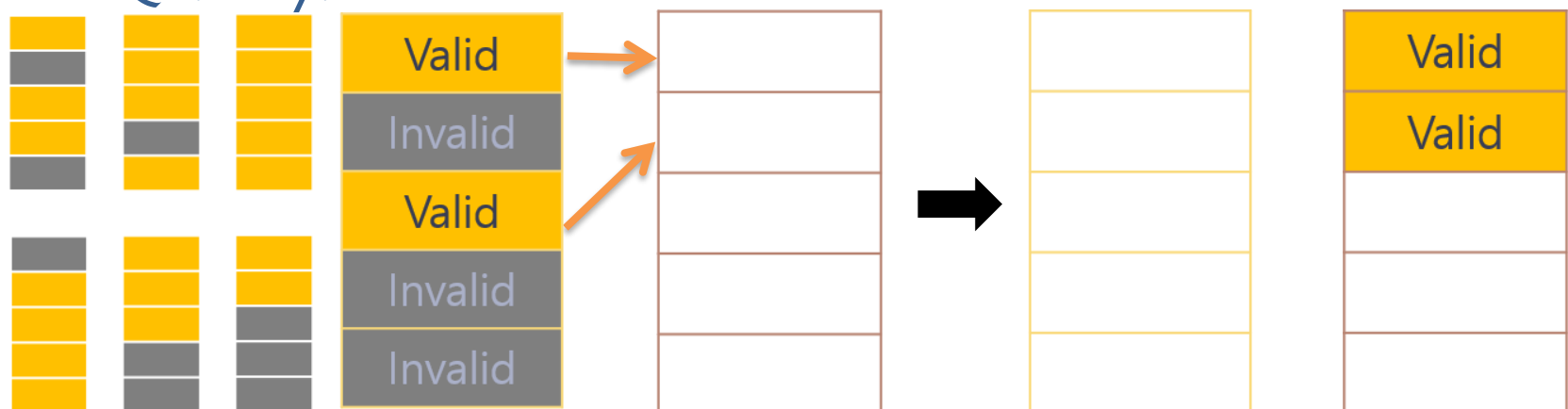
# Quiz #1

- Q1: Guess the reason why some data should be kept at DRAM, not SRAM?
- Q2: What kinds of metadata are copied into NAND at which function in Greedy FTL?
- Q3: Is it resistive against power crashes?

# Garbage Collection

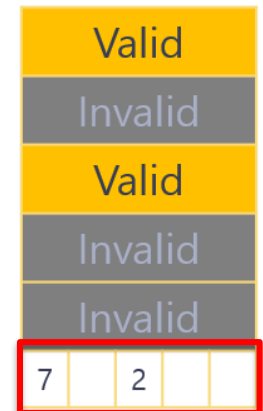
- GC Process
  - Select victim block
  - Copy valid pages to free block
  - Erase victim block
- GC policy: which block is chosen for victim?
  - Greedy: one with the minimum number of valid page

- Q4: Why?



# Garbage Collection (cont'd)

- Cannot use spare area in Jasmine
  - Used for own ECC
  - Instead, Greedy FTL does “reverse index”
- Reverse index
  - L2P table: LPN  $\rightarrow$  PPN
  - Reverse index: PPN  $\rightarrow$  LPN
  - Q5: When and why is it necessary?
  - Greedy FTL uses a end page of each block as reverse index



# Quiz #2

- Q6: When is GC triggered in greedy FTL?
- Q7: What is the disadvantage of Q6's policy?
- Q8: Summarize GC process of greedy FTL at code-level
- Send answer from Q1 to Q8
  - [donghyun.kim@cs.skku.edu](mailto:donghyun.kim@cs.skku.edu)
  - Due: Until next lab class
  - Answer each question in 1 sentence except for Q8

# Miscellaneous: 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 ?
```

# Miscellaneous: 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

# Miscellaneous: Bad Block Management

- 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

**Any Questions?**