Greedy FTL

Prof. Dongkun Shin (dongkun@skku.edu)
TA – Junho Lee (crow6316@skku.edu)
TA – Somm Kim (sommkim@skku.edu)
Embedded Software Laboratory
Sungkyunkwan University
http://nyx.skku.ac.kr

Contents

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

- Sector-based Page mapping FTL simulator

Greedy FTL

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

- Page mapping FTL
  - Simple GC
    - GC policy: greedy
  - NPO (Normal Power Off) recovery
    - ftl_flush()
    - load_metadata()

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

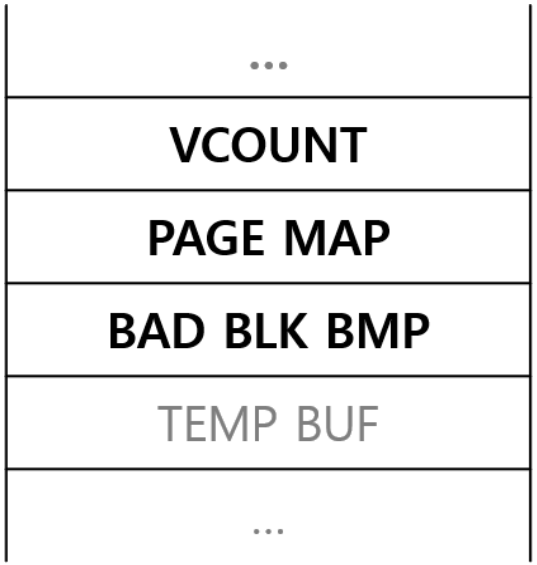
```
//----------------------------------
// metadata structure
//----------------------------------
typedef struct _ftl_statistics
{
    UINT32 gc_cnt;
    UINT32 page_wcount; // page write count
}ftl_statistics;

typedef struct _misc_metadata
{
    UINT32 cur_write_vpn; // physical page for new write
    UINT32 cur_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 in DRAM

- ftl.h



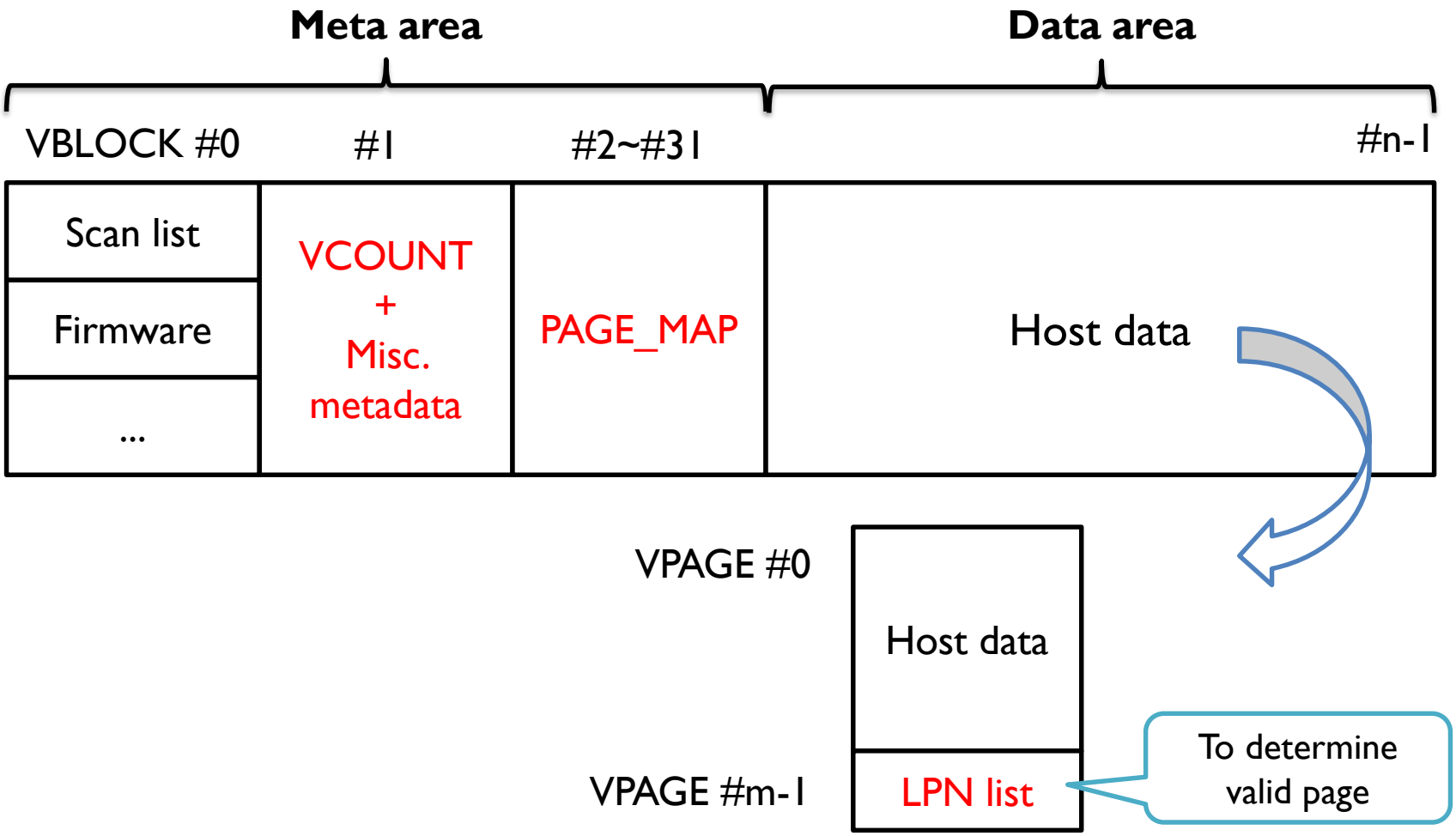

```
#define BAD_BLK_BMP_ADDR    (TEMP_BUF_ADDR + TEMP_BUF_BYTES)                  // bitmap of initial bad blocks
#define BAD_BLK_BMP_BYTES   (((NUM_VBLKS / 8) + DRAM_ECC_UNIT - 1) / DRAM_ECC_UNIT * DRAM_ECC_UNIT)

#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 ?
```

NPO (Normal Power Off) Test

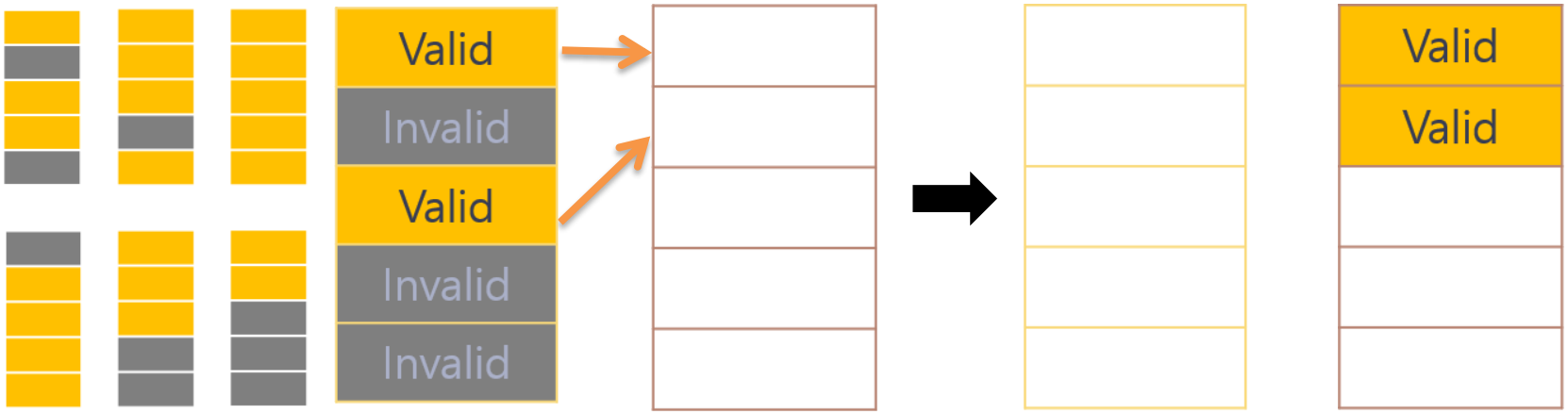
- Enable *check_format_mark()* function.
  - Modify ftl_open().

```
//---------------------------------------
// 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)
//if (TRUE)
{
    uart_print("do format");
    format();
    uart_print("end format");
}
```

- After connecting, insert your data, then turn Jasmine off and on.
  - Use "Safely Remove Hardware“
- Verify that data is preserved.

Garbage Collection

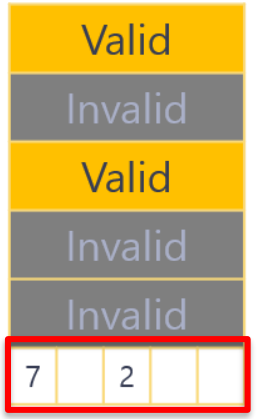
- GC Process
  - Select victim block
  - Copy valid pages to free block
  - Erase victim block
- GC policy: which block is chosen for victim?
  - Greedy: one with the minimum number of valid page



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 -> PPN
  - Reverse index: PPN -> LPN
  - Greedy FTL uses an end page of each block as reverse index



Modify Greedy FTL GC Policy

- Change the Jasmine Greedy FTL's Victim Block selection policy to Cost-benefit.

- Add statistics related variables for calculating age.

```
//----------------------------------
// 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];

// SATA read/write buffer pointer id
UINT32                g_ftl_read_buf_id;
UINT32                g_ftl_write_buf_id;

// WAF statics
UINT32                num_host_write;
UINT32                num_gc_write;
```

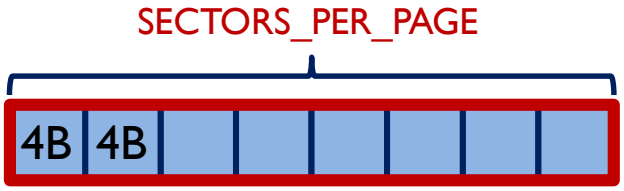
Sector-based
Page Mapping FTL Simulator

Lab 5 : Sector-based Page Mapping FTL Simulator

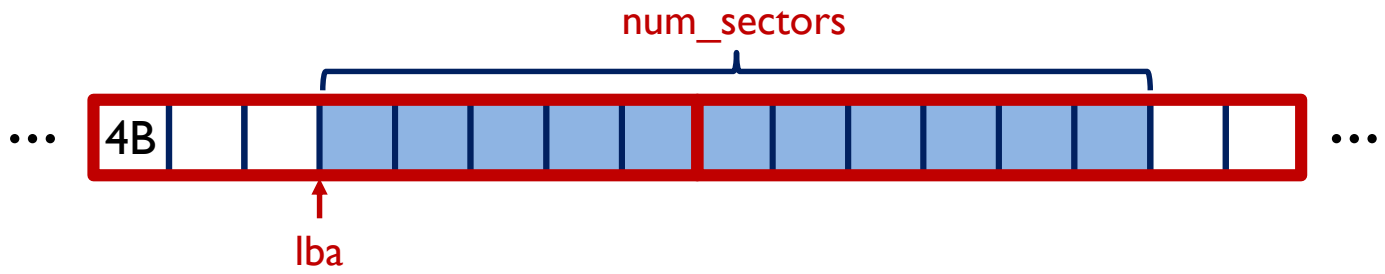
- Develop a page mapping FTL simulator with variable request size.
  - Simulate the operations of page mapping FTL
    - The operation of the FTL is similar to the page mapping FTL simulator at Lab 4
  - The host requests I/O based on sector
    - Variable size of write/read request
  - Assumption
    - Sector size: 4B
    - Page size: 32B(data area) + 4B(spare area)
    - Initial state of flash blocks: empty

Descriptions

- Page & Sector



- ex) ftl_write(lba, num_sectors, write_buffer)



ftl.c

- ftl_open()
- ftl_read(u32 lba, u32 num_sectors, u32 *read_buffer)
  - lba: start sector #
  - num_sectors: # of sectors
- ftl_write(u32 lba, u32 num_sectors, u32 *write_buffer)
  - lba: start sector #
  - num_sectors: # of sectors
  - You should add s.ftl_write for every nand_write call
- garbage_collection(u32 bank)
  - You should add s.gc_write for every nand_write call

- The unit of write count variable of 'struct ftl_stat' is sector.
- You can modify only ftl.c and nand.c.

Miscellaneous

- Recommended environment : Linux (Ubuntu is ok!)
  - You can do it in Windows, but be sure that your work also runs in Linux (I'll score all the works only in Ubuntu 16.04)
- Personal Project
- You should submit a report
  - Describe your code (Changes to your existing code)
  - Capture and analyze the WAF of Greedy policy and Cost-Benefit policy
    - 4 cases (Random/Hot Cold Workload & Greedy/Cost-Benefit policy)
    - HOT_RATIO: 96, N_RUNS: 100
- Submit to the icampus
  - Due: 10/23(Wed.) 23:59:59
  - File to submit: ftl_sim.c, ftl.h, ftl.c, nand.h, nand.c, Makefile, report.pdf
  - File name: $(STUDENT_ID).tar.gz
- Late penalty : -20% per day (Up to 3 days)

Any Questions?