

# Developing FTLs on the Jasmine OpenSSD Platform

Sang-Phil Lim ([lsfeel0204@gmail.com](mailto:lsfeel0204@gmail.com))

SKKU VLDB Lab.

2011-05-11

# Outline

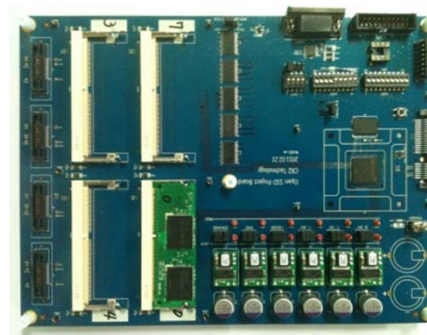
- **Getting Started**
- **Building & Installing Firmware**
- **Greedy FTL Implementation & Development Guide**
- **Debugging Guide**



# Getting Started

# Development Setup: Hardware

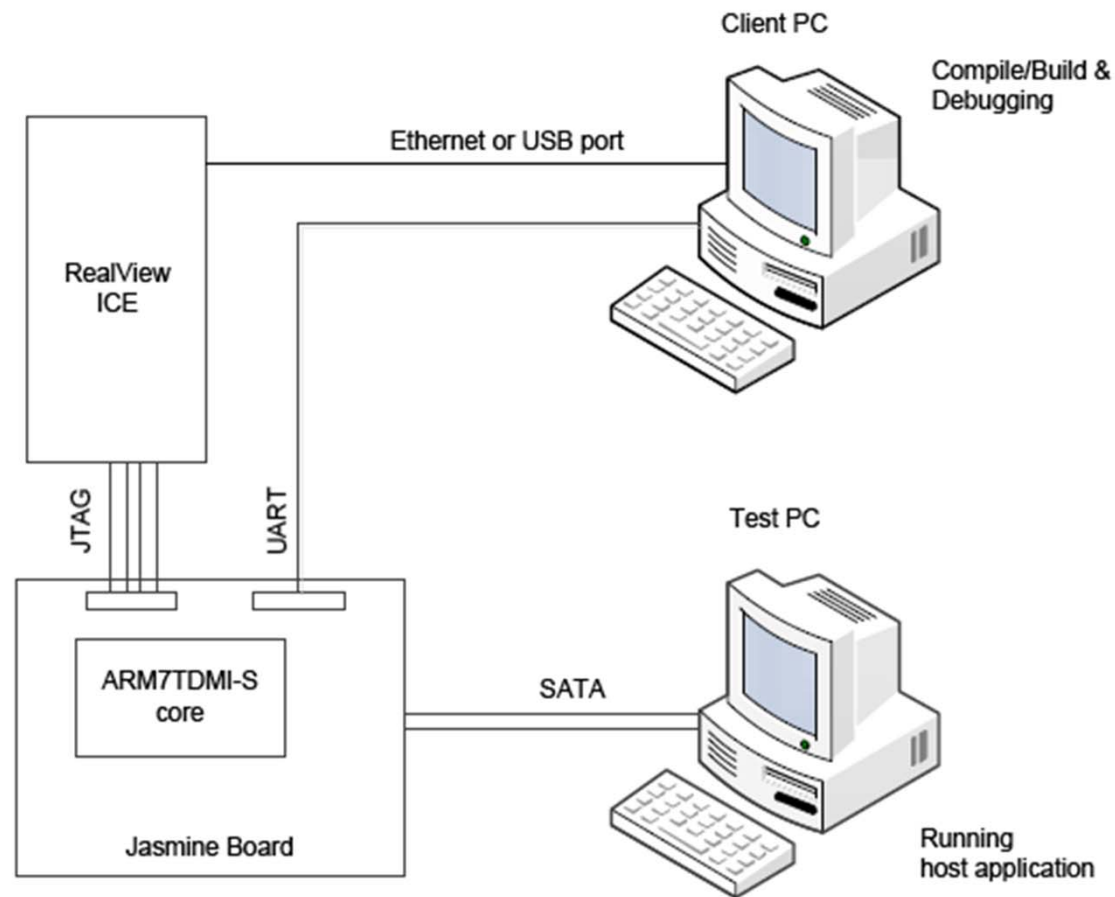
- Hardware Requirement



– For debugging (optional)



# Development Setup: Hardware



# Development Setup: Software

- **Jasmine Firmware**
  - Latest version: v1.0.3
  - Available from  
<http://www.openssd-project.org/wiki/Downloads>
- **RVDS(RealView Development Suite) 3.0 ↑ or Code Sourcery G++ Lite Edition for ARM (free)**
- **MS Visual Studio Express Free Edition 2010 (free)**

# Development Setup: Software

- **Installing toolchain**

- RVDS or Code Sourcery G++ Lite Edition for ARM
  - To build the firmware binary file (`firmware.bin`)
- MS Visual Studio Express Free Edition 2010
  - To build the firmware installer (`install.exe`)

- **Serial communication**

- Hyperterminal (BAUD\_115200/8/N/1/X)
- Configure on-board switches (SW 2,3,4)  
(please refer to Jasmine board schematics)



# Building & Installing Firmware



# Compile & Build Firmware

- **Setting compile options** (`./include/jasmine.h`)

```
OPTION_2_PLANE
OPTION_ENABLE_ASSERT
OPTION_FTL_TEST
OPTION_UART_DEBUG
OPTION_SLOW_SATA
OPTION_SUPPORT_NCQ
OPTION_REDUCED_CAPACITY
```

# Compile & Build Firmware

- **Build the firmware**

```
> cd ./build_gnu  
> build.bat
```

- **Compile the installer**

- Open `./installer/installer.sln` & Build
- Move `./installer/install.exe`  
to `./build_gnu`

# Install Firmware to the Jasmine Board

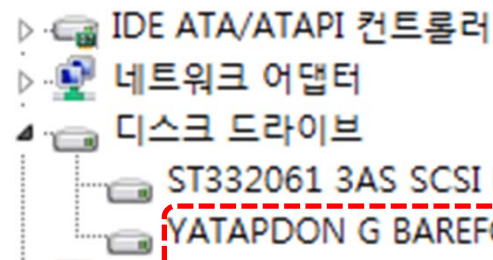
- Booting the Jasmine board as '*Factory mode*'



Factory Mode



Normal Mode  
(Default)



Device manager

- Install firmware

```
> ./build_gnu/install.exe
```

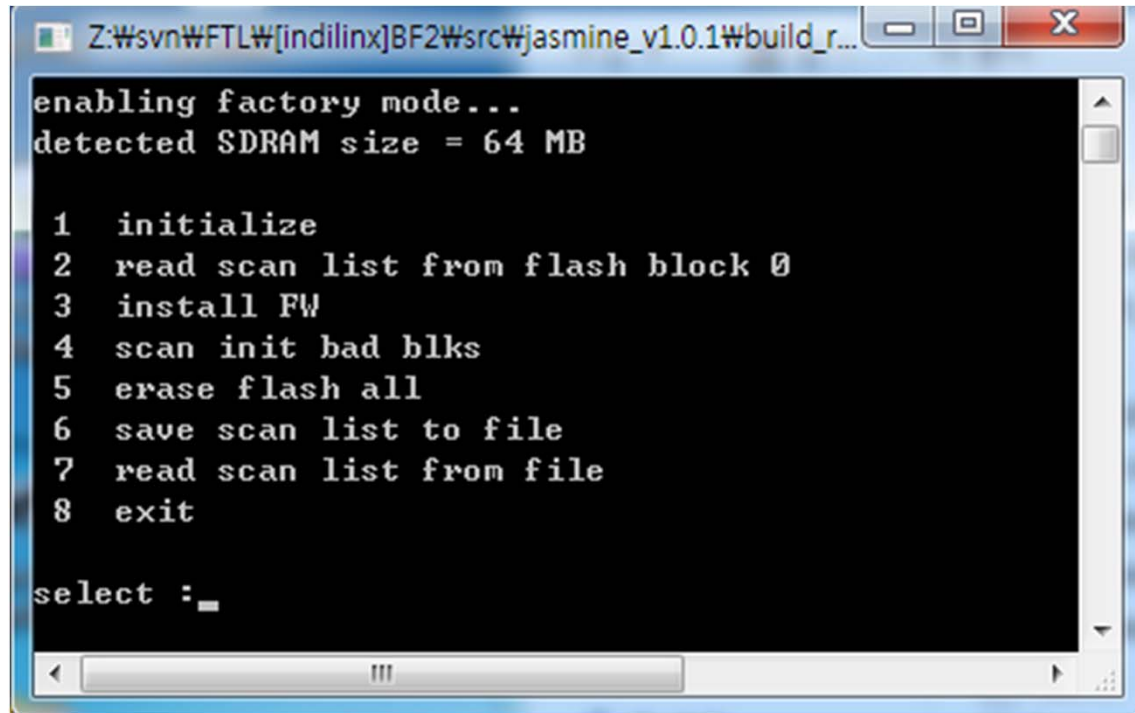
# Install Firmware to the Jasmine Board

- Install for the first time

1 – 2 – 6 – 3

- Reinstall

1 – 2 – 3



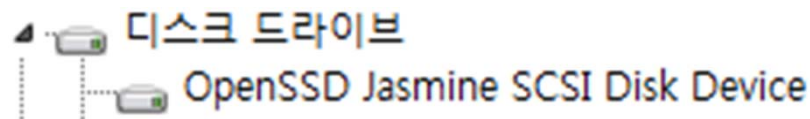
```
Z:\svn\FTLW[indilinx]BF2\src\jasmine_v1.0.1\build_r...
enabling factory mode...
detected SDRAM size = 64 MB

1  initialize
2  read scan list from flash block 0
3  install FW
4  scan init bad blks
5  erase flash all
6  save scan list to file
7  read scan list from file
8  exit

select :_
```

# Run Firmware

- **Booting the Jasmine board as '*Normal mode*'**
  - Unplug SATA cable
    - Jasmine would be busy doing internal low-level format
  - Plug SATA cable when LED at D4 position is lit
- **Now Jasmine is ready to process SATA commands**
- **Try to send IO requests to Jasmine board! ☺**





# Greedy FTL Implementation & FTL Development Guide

# FTL Implementation

Source file	Function	Description
./installer/installer.c	ftl_install_mapping_table	FTL 초기 메타데이터를 NAND 플래시에 기록하는 연산 수행. 펌웨어 설치 시 함께 호출됨
./ftl_[scheme]/ftl.c	ftl_open	FTL 초기화 과정 수행 - NAND 플래시로부터 메타데이터 로드 및 초기화 - VBLK #0 에 포맷 마크가 기록되어 있지 않을 경우, format 함수 호출
	format	VBLK #0 와 FTL 메타 영역을 제외한 나머지 블록들을 삭제 - 포맷 마크 기록
	ftl_read	사용자 데이터 읽기 처리
	ftl_write	사용자 데이터 쓰기 처리
	ftl_flush	SATA idle/standby time 에 주기적으로 메타데이터를 flush 하는 연산 수행

# Sample FTLs on the Jasmine Platform

- **Tutorial FTL** (developed by INDILINX)
  - Page-mapping FTL, but no garbage collection
- **Greedy FTL** (developed by SKKU VLDB Lab.)
  - Page-mapping FTL with simple garbage collection
  - Support Power-Off Recovery (*to appear*)
- **Dummy FTL**
  - Not a real FTL (*Not access NAND flash at all*)
  - For measuring SATA and DRAM speed



# Greedy FTL: FTL Open

```
void ftl_open(void)
{
    // check DRAM footprint
    sanity_check();
    // build bitmap of bad blocks
    build_bad_blk_list();

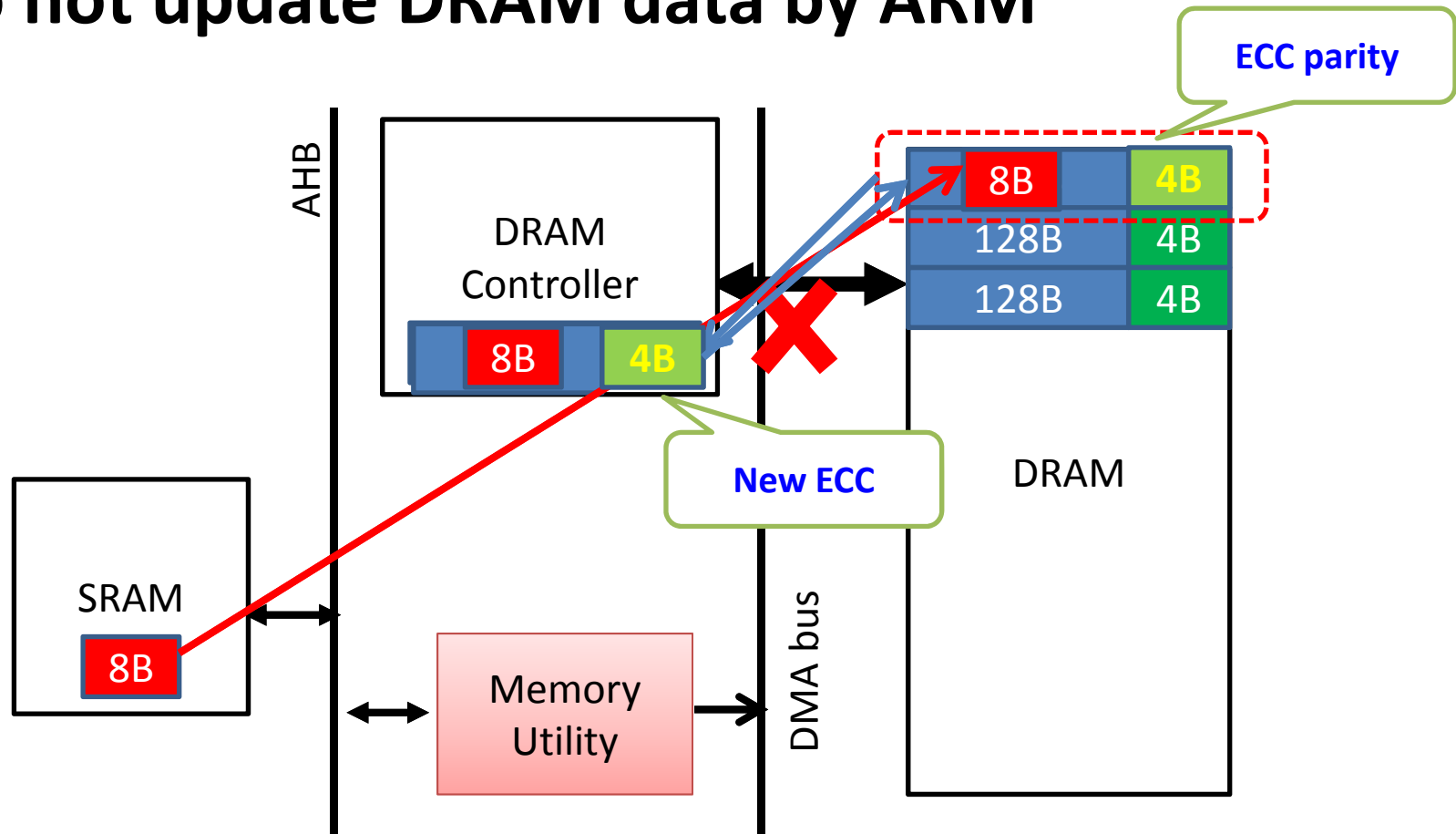
    if (check_format_mark() == FALSE) {
        format();
    }
    // load FTL metadata
    else {
        // load FTL metadata in SRAM/DRAM from nand
        load_metadata();
    }
    // init FTL & SATA buffer pointer
    g_ftl_read_buf_id = 0;
    g_ftl_write_buf_id = 0;
}
```

# Greedy FTL : FTL Metadata

- **DRAM metadata**
  - Page-level mapping table (PAGE\_MAP)
    - For Logical-to-Physical page address mapping
  - Bad block bitmap table (BAD\_BLK\_BMP)
  - Block valid count information (VCOUNT)
    - For victim block selection
- **SRAM metadata**
  - Misc. information
    - Page index pointer of meta & user area
    - Remain free block count
    - Etc.

# Guide #1 - Memory Utility

- Do not update DRAM data by ARM



# Guide #2 – DRAM Buffer Management

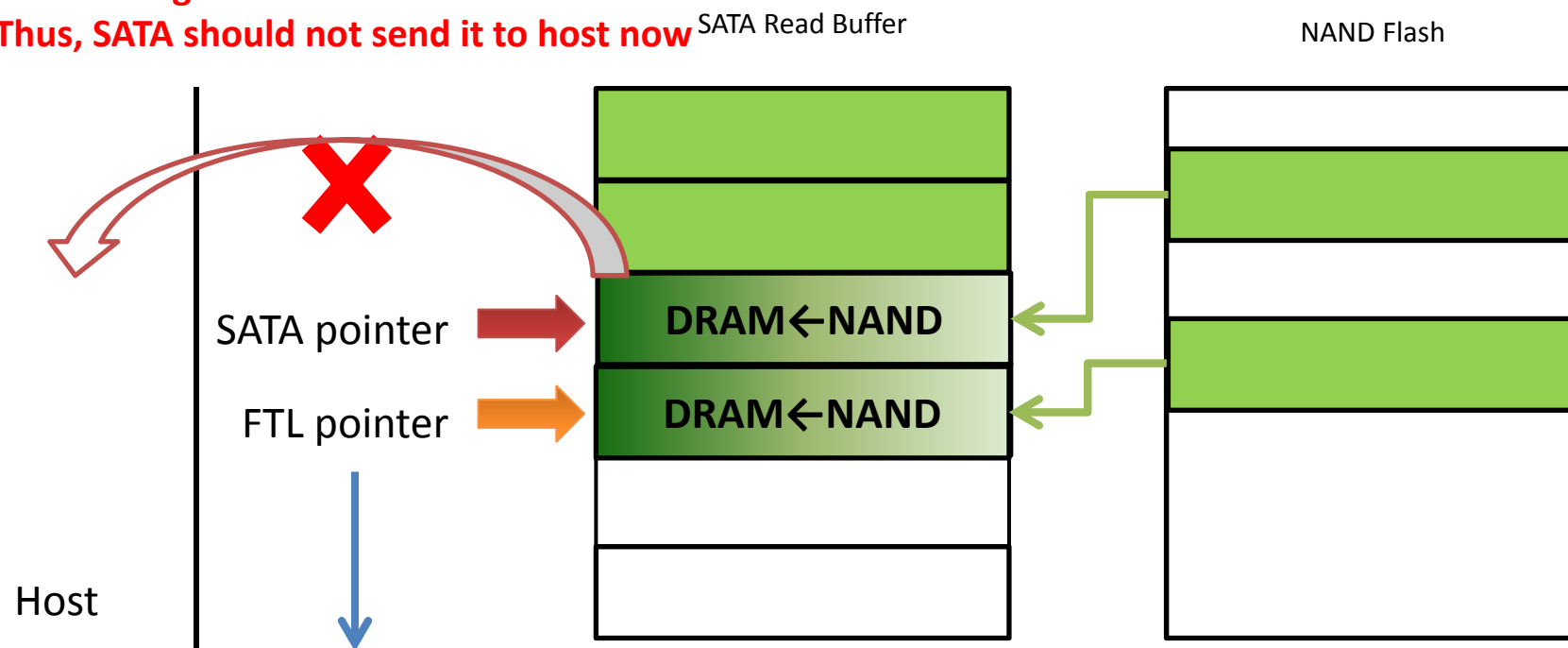
- **SATA read/write buffer**
  - Buffering the data to send to host or the data to program onto NAND
  - Circular FIFO
  - Controlled by **SATA/FTL/BM** buffer pointers
  - **Why do we need a BM buffer pointer?**
    - I/O Consistency issue
      - bandwidth gap between SATA and NAND
      - “SATA could send the wrong data to host”
      - “NAND could program the wrong host data”

## Guide #2 – DRAM Buffer Management (contd.)

- What if BM buffer pointer is not used?

Still reading data from NAND!

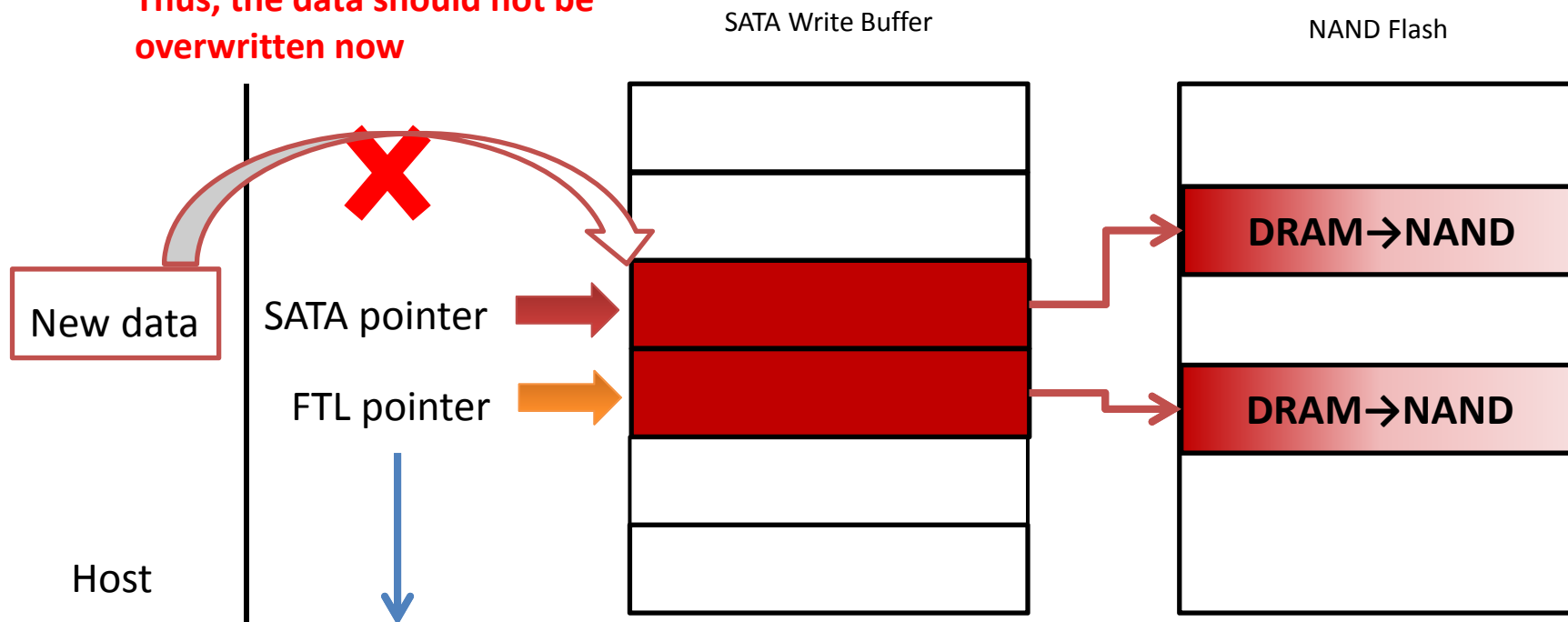
Thus, SATA should not send it to host now



## Guide #2 – DRAM Buffer Management (contd.)

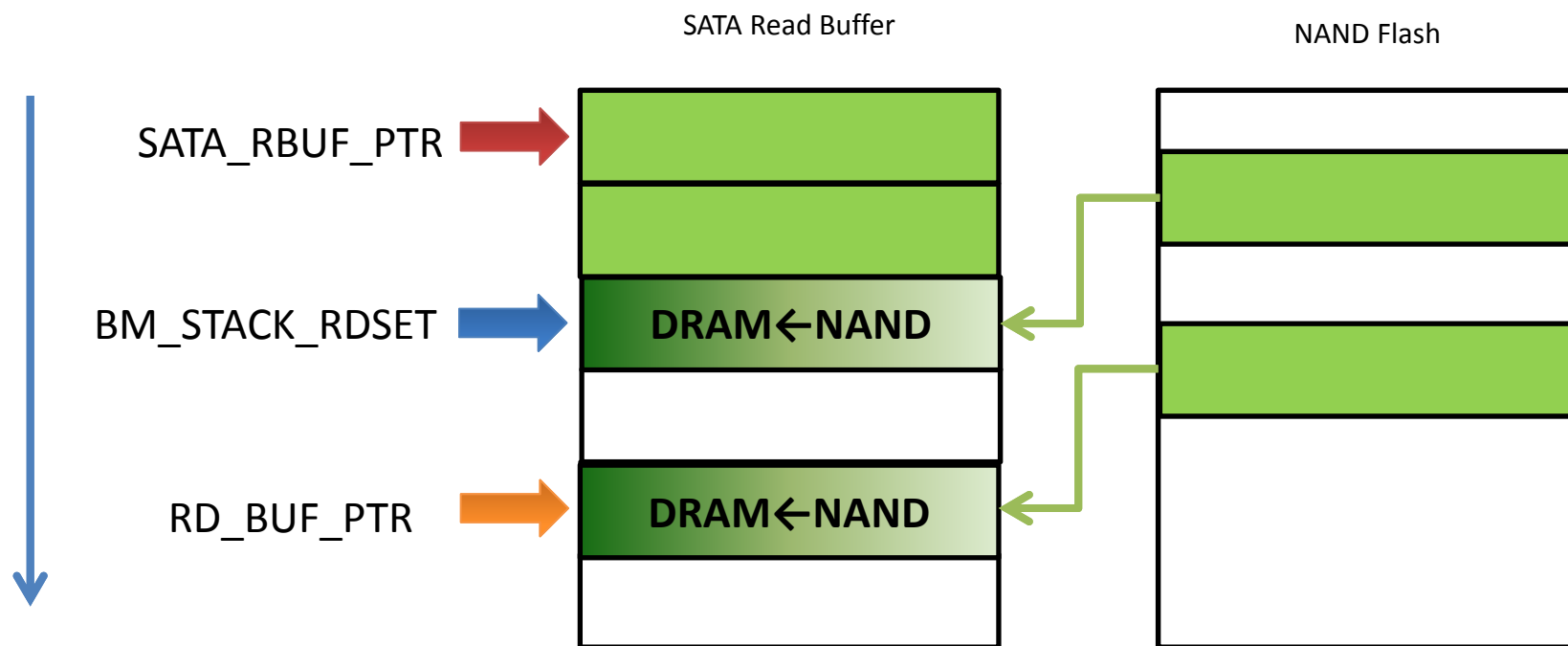
- What if BM buffer pointer is not used?

Still programming to NAND!  
Thus, the data should not be  
overwritten now



## Guide #2 – DRAM Buffer Management (contd.)

- SATA read/write buffer



# Greedy FTL : Read operation

```
void ftl_read(UINT32 const lba, UINT32 const num_sectors)
{
    lpn = lba / SECTORS_PER_PAGE;
    sect_offset = lba % SECTORS_PER_PAGE;
    remain_sects = num_sectors;

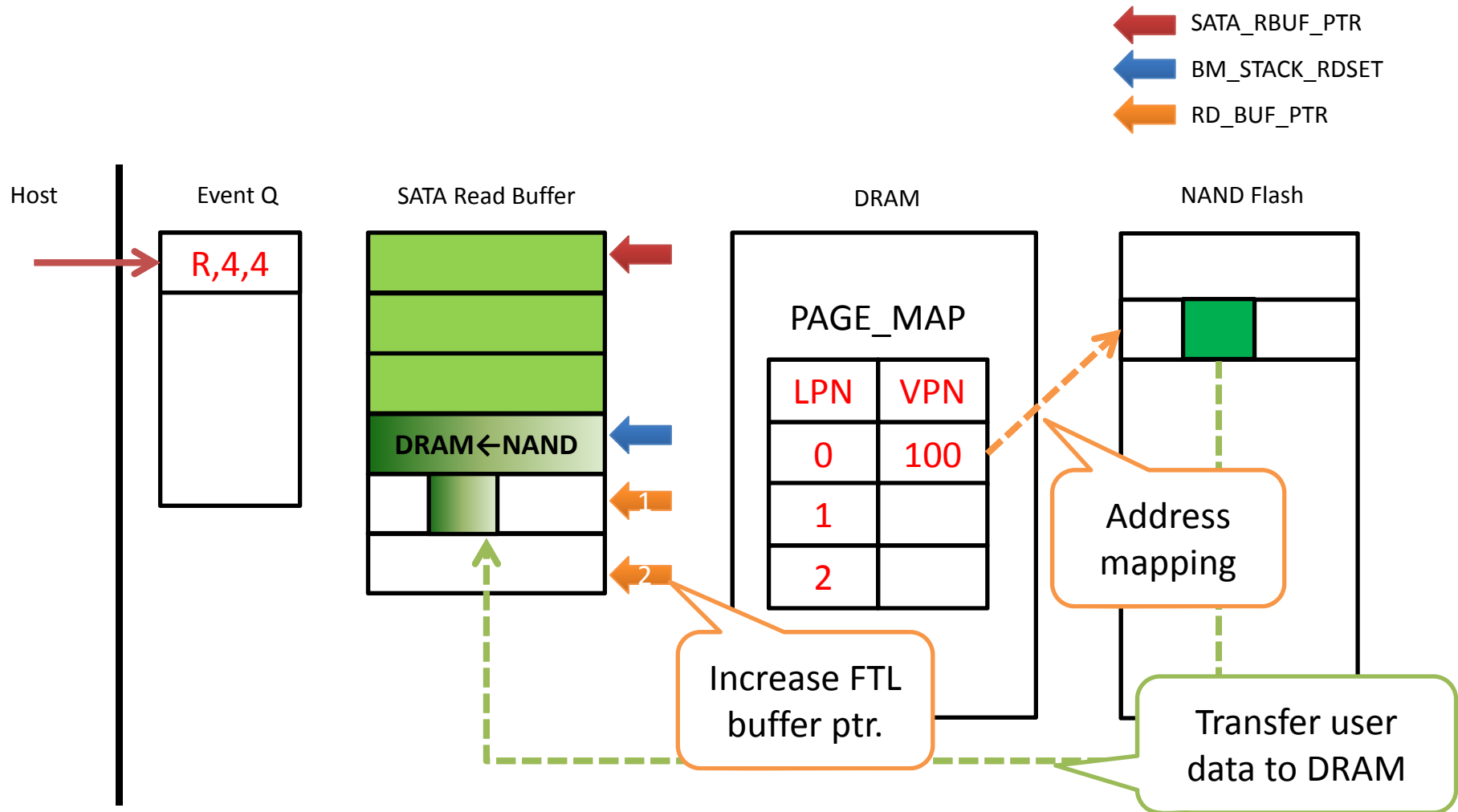
    while (remain_sects != 0) {
        if ((sect_offset + remain_sects) < SECTORS_PER_PAGE) {
            num_sectors_to_read = remain_sects;
        }
        else {
            num_sectors_to_read = SECTORS_PER_PAGE - sect_offset;
        }
        bank = get_num_bank(lpn); // virtual page-level striping
        vpn = get_vpn(lpn);       // address mapping
        ...
    }
}
```



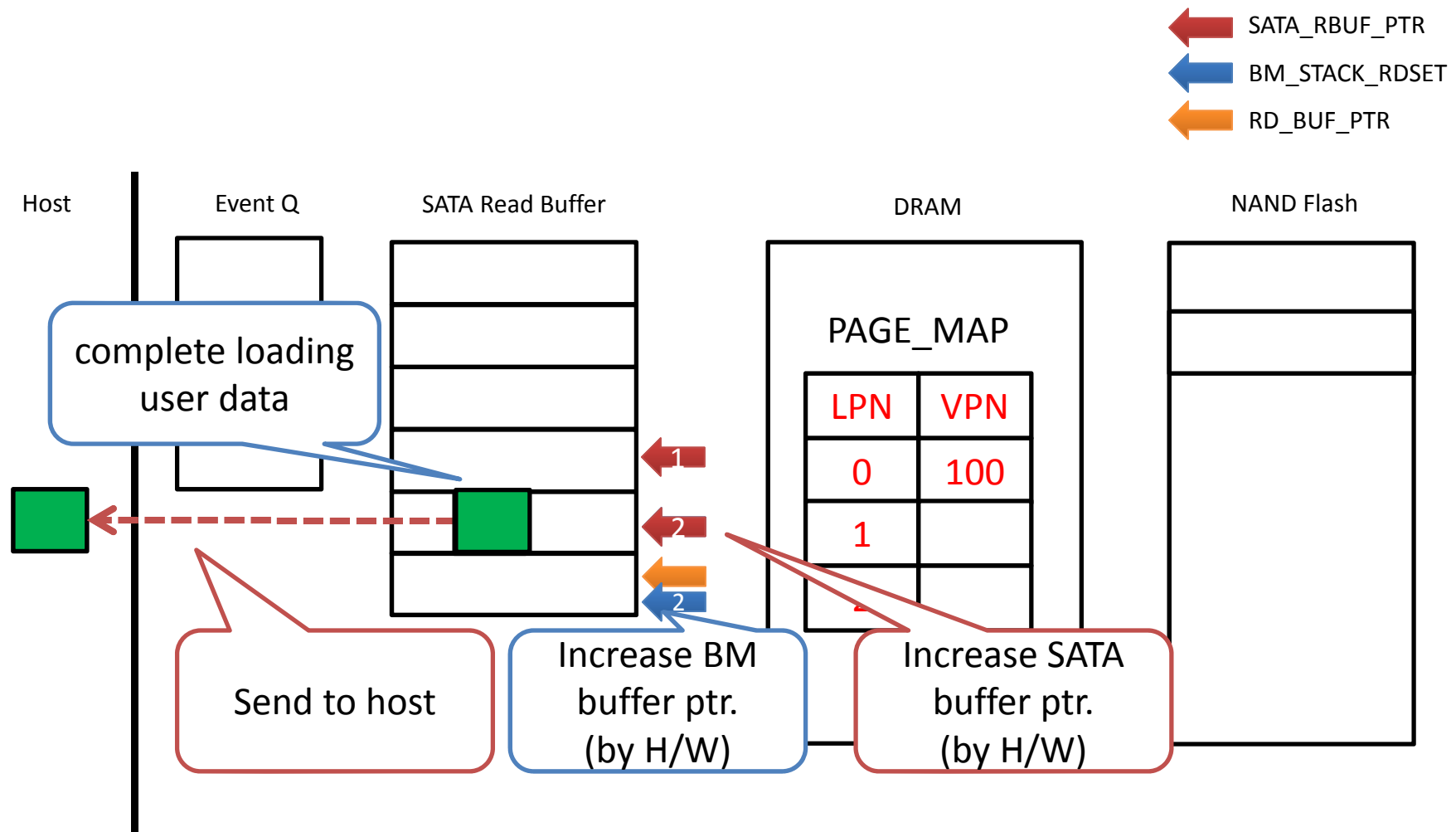
# Greedy FTL : Read operation (contd.)

```
...
// old data was already written
if (vpn != NULL) {
    // send read request to nand flash
    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 {
    // change Buffer Manager read limit pointer
    // change FTL SATA read pointer
}
sect_offset    = 0;
remain_sects -= num_sectors_to_read;
lpn++;
} // end of ftl_read function
```

# Greedy FTL : Read operation (contd.)



# Greedy FTL : Read operation (contd.)



# Greedy FTL : Write Operation

```
static void write_page(UINT32 const lpn, UINT32 const sect_offset,
UINT32 const num_sectors)
{
    bank          = get_num_bank(lpn);
    page_offset = sect_offset;
    column_cnt    = num_sectors;
    old_vpn       = get_vpn(lpn);           // address mapping
    new_vpn       = assign_new_write_vpn(bank); // get free vpage

    // if old data already exist,
    if (old_vpn != NULL) {
        // read `left hole sectors' [left*][new data][right]
        if (page_offset != 0)
            nand_page_ptread(..., RETURN_ON_ISSUE);
        // read `right hole sectors' [left][new data][right*]
        if ((page_offset + column_cnt) < SECTORS_PER_PAGE)
            nand_page_ptread(..., RETURN_ON_ISSUE);
        // invalid old page (decrease vcount)
        set_vcount(bank, vblock, get_vcount(bank, vblock) - 1);
    }
    ...
}
```

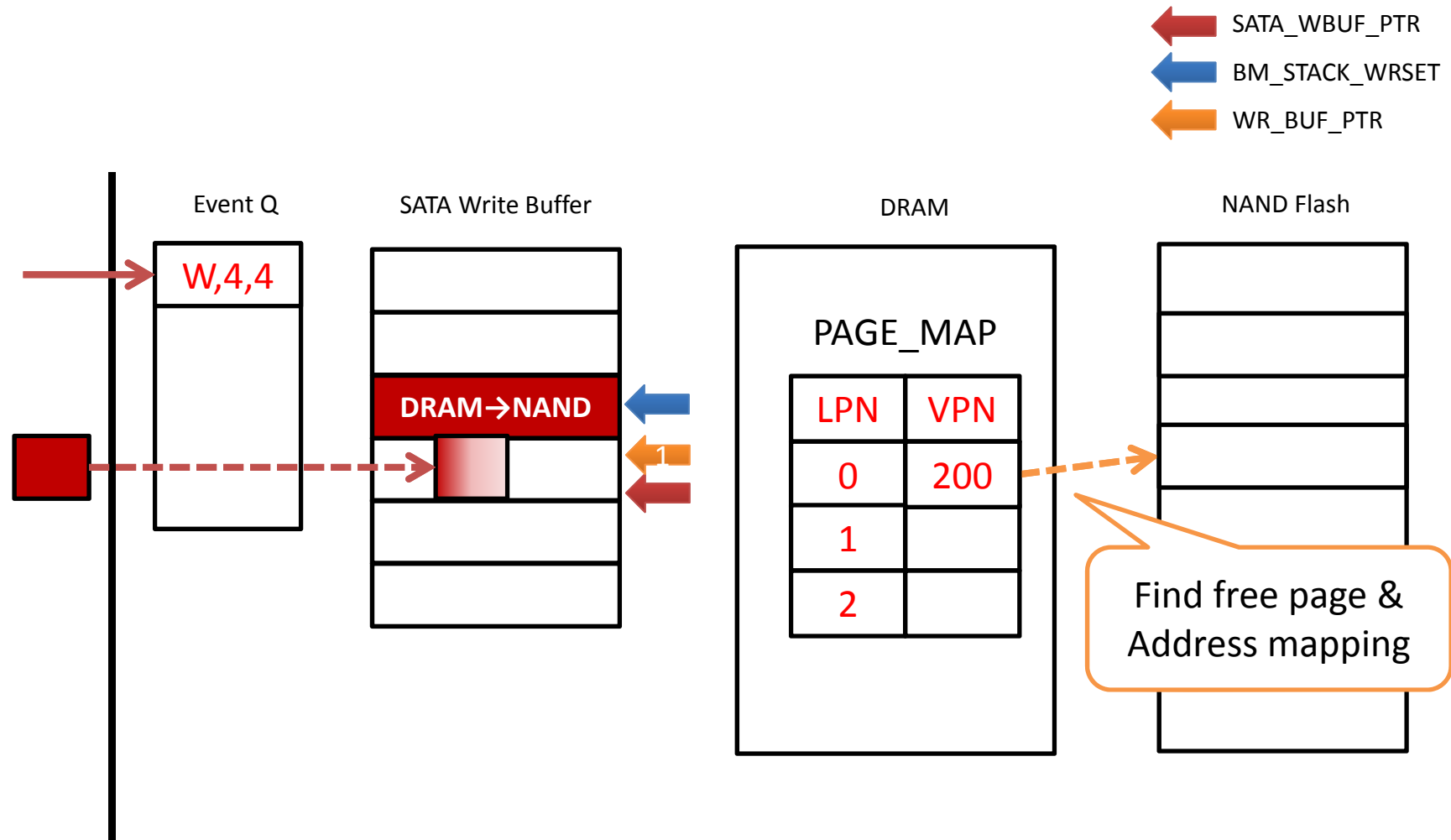
# Greedy FTL : Write Operation (contd.)

```
...
vblock    = new_vpn / PAGES_PER_BLK;
page_num  = new_vpn % PAGES_PER_BLK;

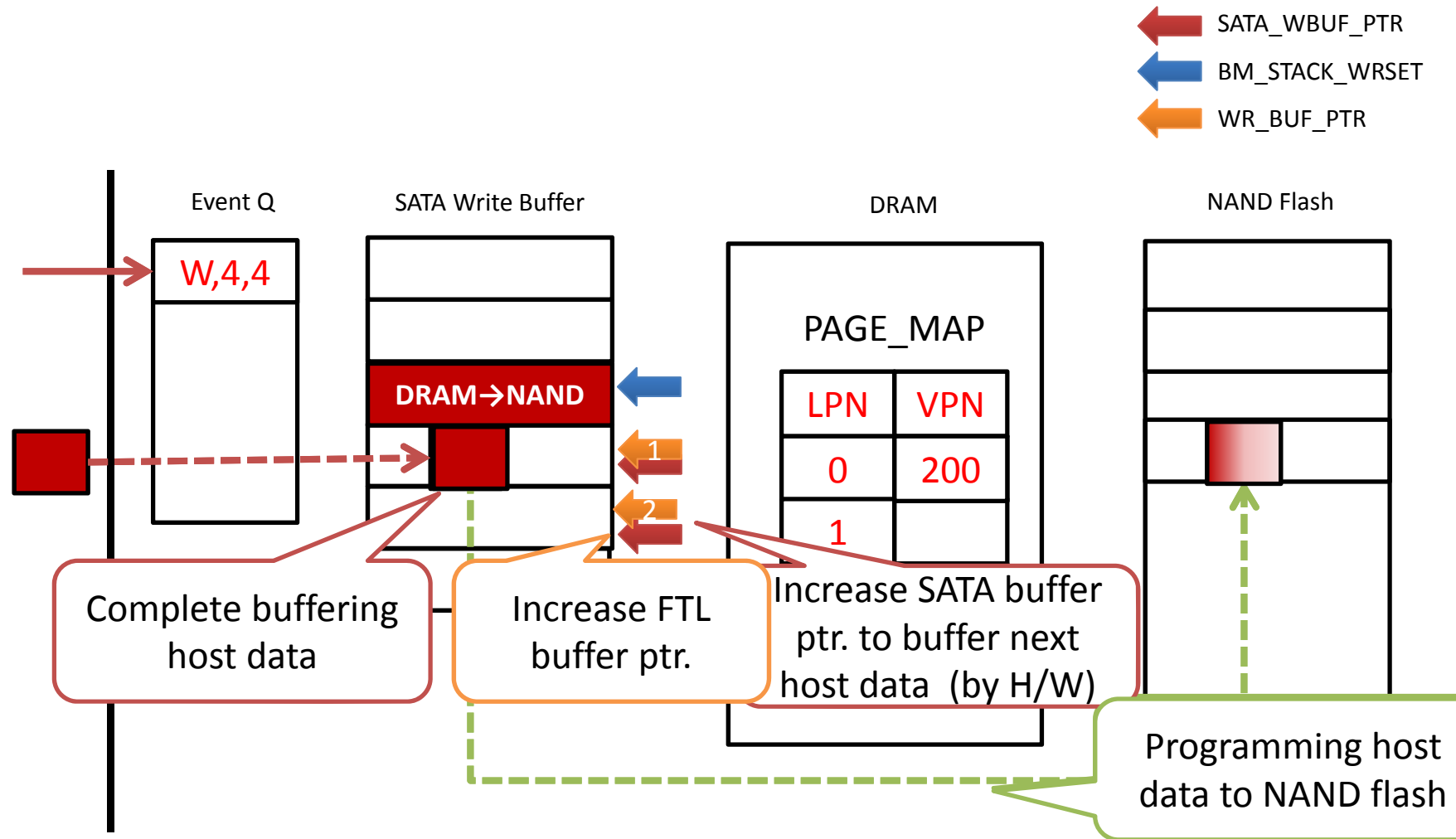
// write new data (RETURN_ON_ISSUE)
nand_page_ptprogram_from_host(bank,
                              vblock,
                              page_num,
                              page_offset,
                              column_cnt);

/* update metadata */
set_lpn(bank, page_num, lpn); // maintain lpn list for GC
set_vpn(lpn, new_vpn);        // address mapping table
// increase block valid page count
set_vcount(bank, vblock, get_vcount(bank, vblock) + 1);
} // end of write_page function (caller: ftl_write)
```

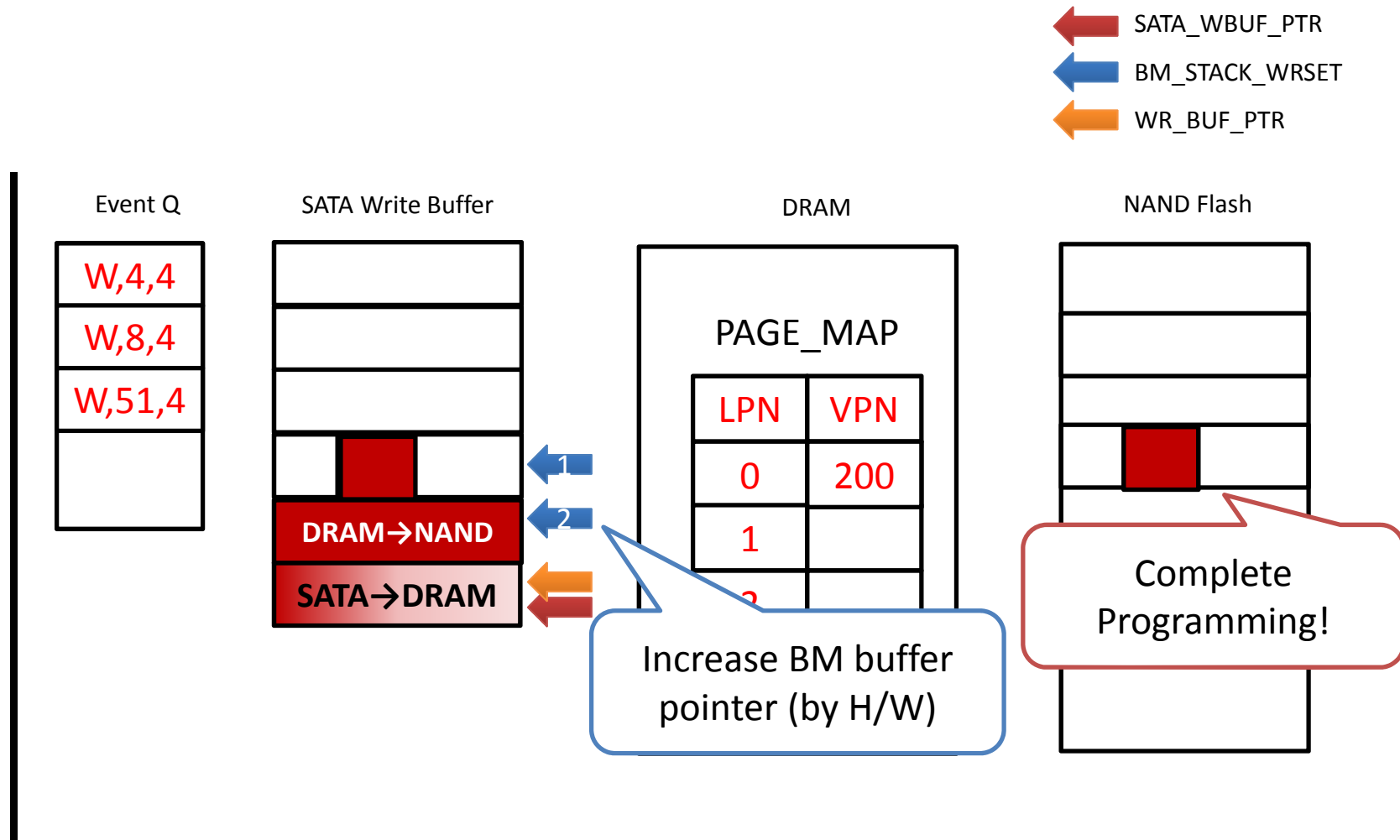
# Greedy FTL : Write Operation (contd.)



# Greedy FTL : Write Operation (contd.)



# Greedy FTL : Write Operation (contd.)

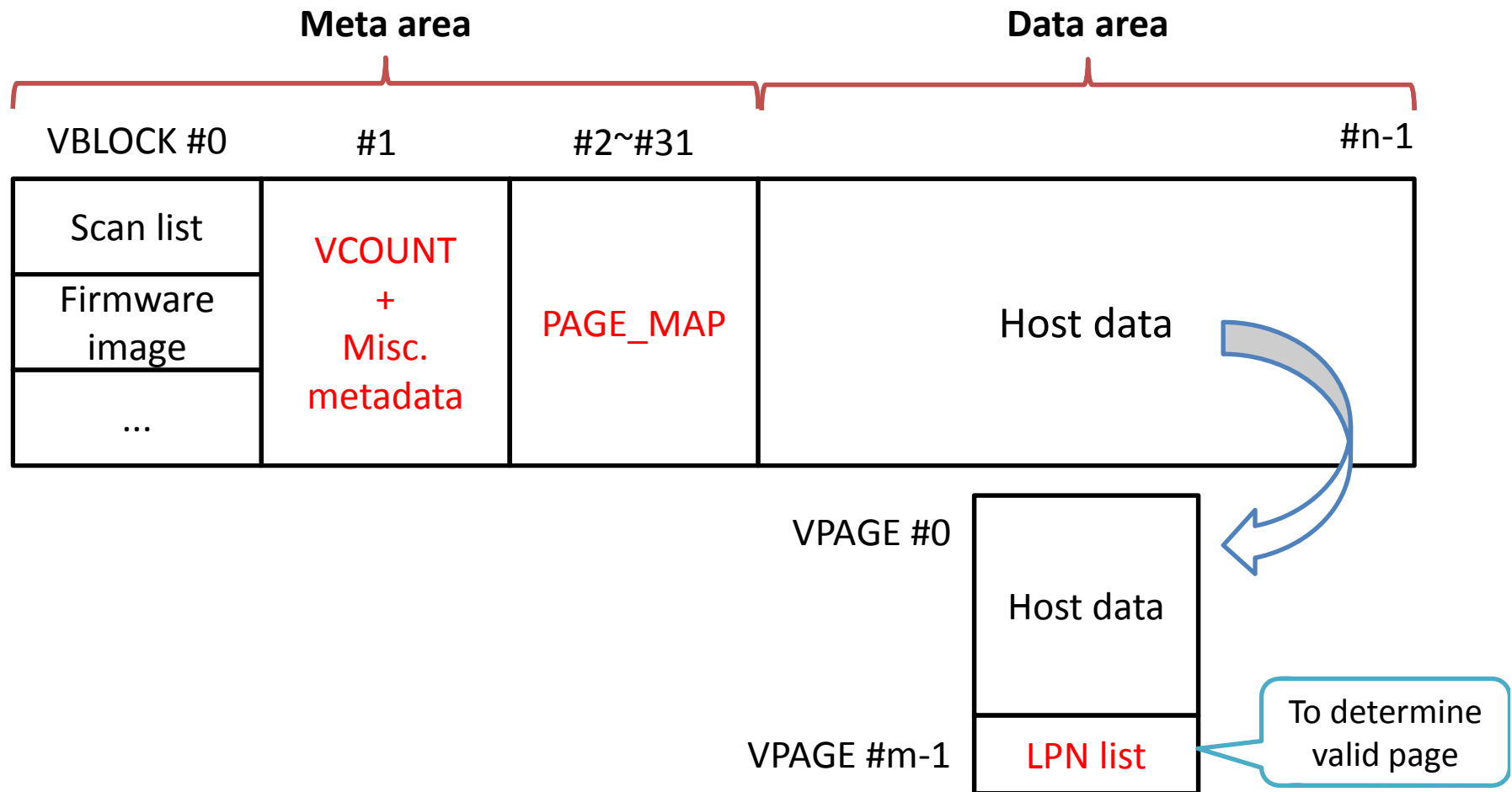




# Guide #3 – Flash Command

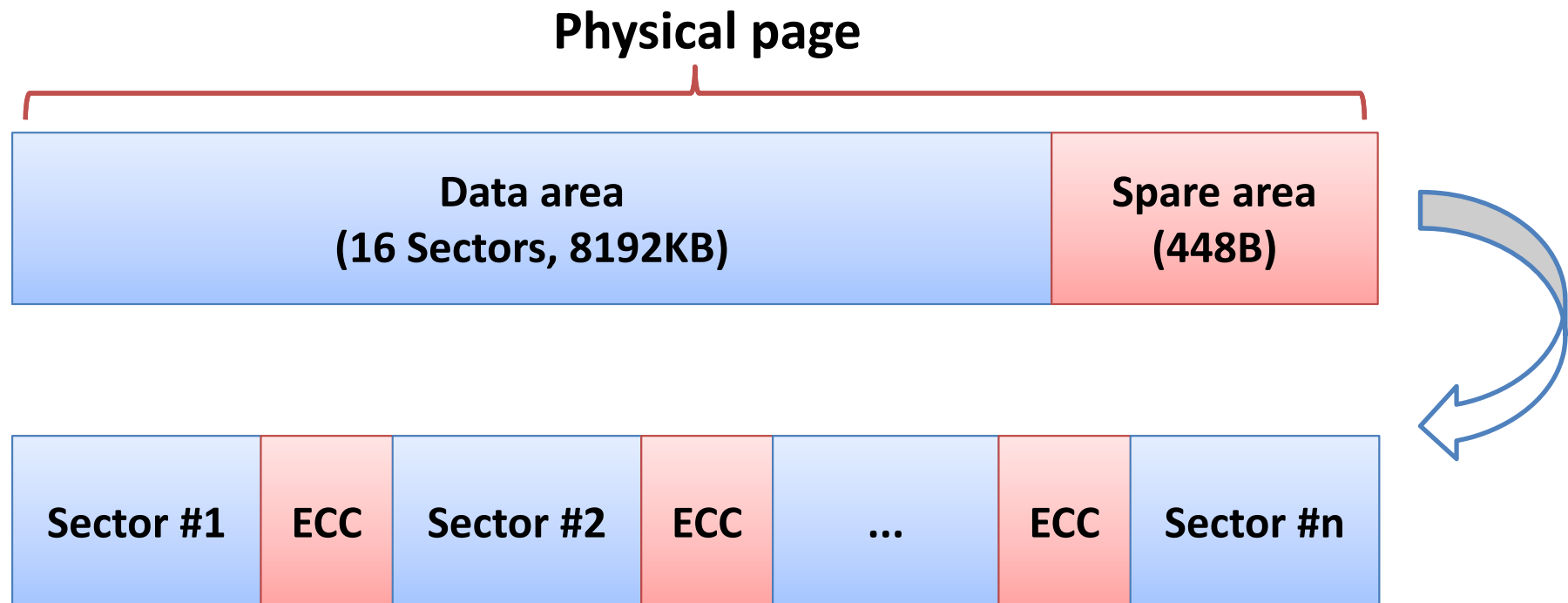
- **To send I/O command to NAND flash**
  - ① Programming FCP command manually
    - See tutorial FTL code ([./ftl\\_tutorial/ftl.c](#))
  - ② Use LLD interface ([./target\\_spw/flash\\_wrapper.c](#))
    - Easiest way to implement an FTL
    - If you want to make a high performance FTL, the first option is recommended

# Greedy FTL : NAND Configuration



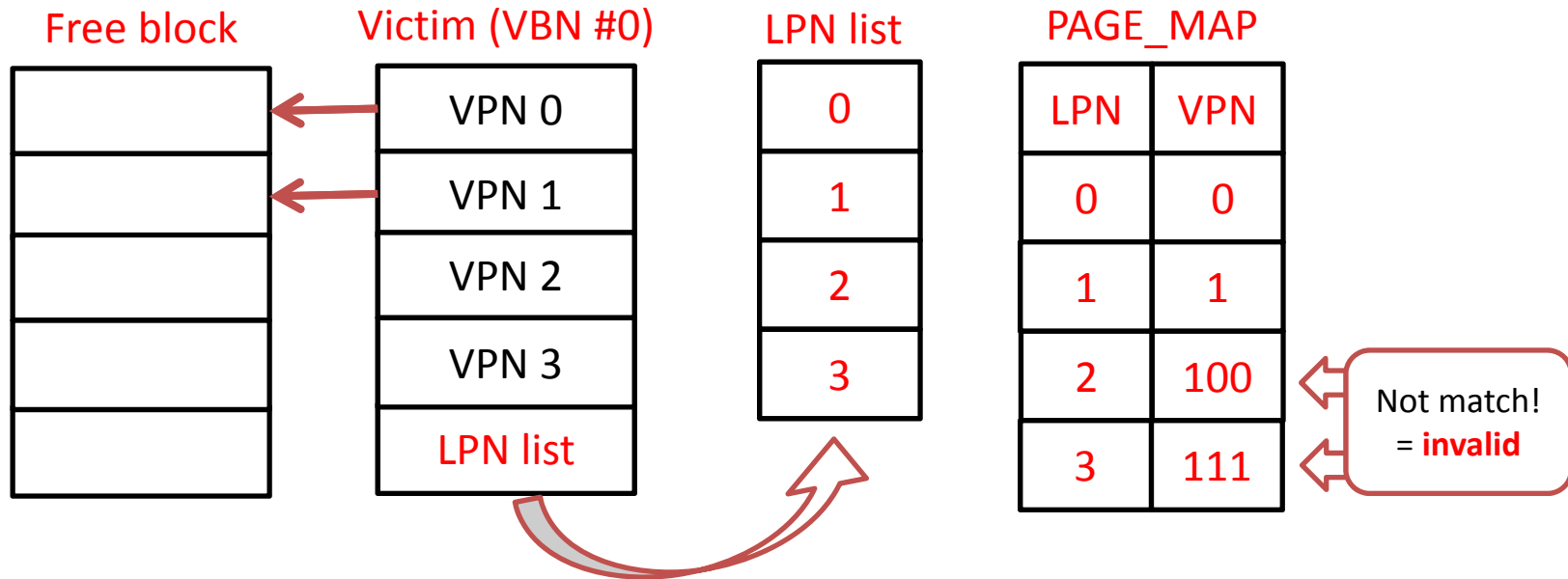
## Guide #4 - Restriction of Accessing NAND

- No support for accessing '**spare area**'



# Greedy FTL : Garbage Collection

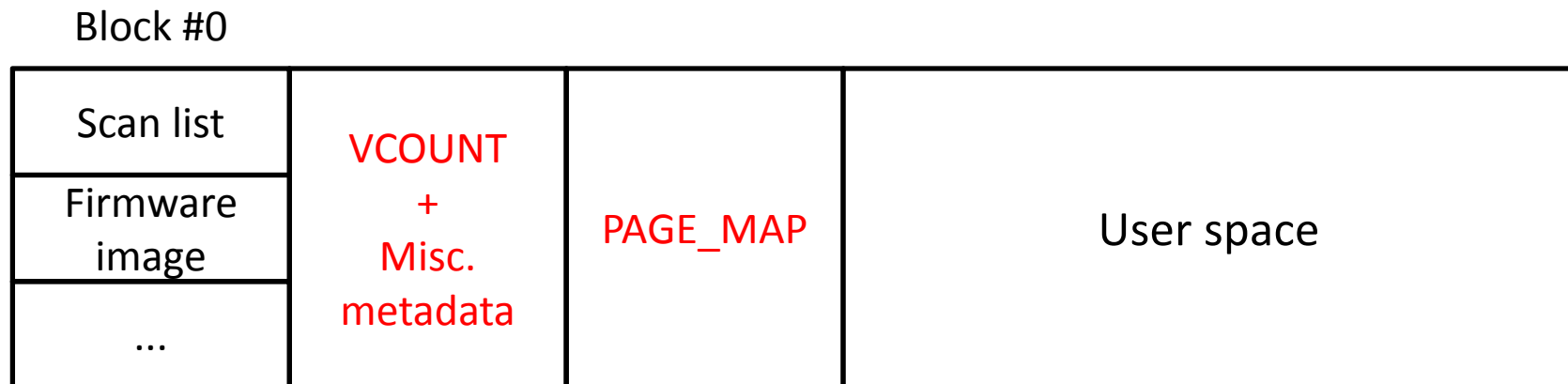
- If free blocks run out, a victim block is chosen based on the **'greedy policy'**
  - i.e., select the block containing min. valid pages



# Greedy FTL : POR(Power-Off Recovery)

- **Metadata logging**

- Flush all metadata instantly (by `ftl_flush`)
- @ SATA ready/idle/standby time



## Guide #5 – BSP interrupt

- Check `BSP_INTR` register (`ftl_isr`)

<code>FIRQ_CORRECTED</code>	<code>FIRQ_CRC_FAIL</code>	<code>FIRQ_MISMATCH</code>
<code>FIRQ_BADBLK_L</code>	<code>FIRQ_BADBLK_H</code>	<code>FIRQ_ALL_FF</code>
<code>FIRQ_ECC_FAIL</code>	<code>FIRQ_DATA_CORRUPT</code>	

- BSP contains the flash command which was last issued for debugging



# Debugging Guide

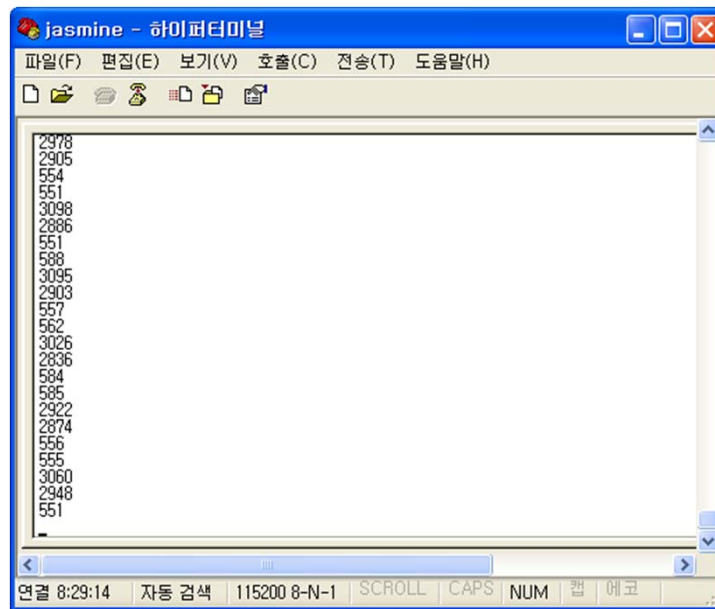
# Ways to Debug Your Code

- ① Serial communication (UART interface)
- ② ICE + RVD (JTAG interface)
- ③ On-board indicator (LED on Position D4)



# UART Debugging

- Memory/Register dump message
- You also can measure I/O response time using Timer function (`./target_spw/misc.c`)



# Debugging using ICE + RVD

- Line-by-line debugging

```
g_barrier = 0;  
while (g_barrier == 0);
```

To stop the  
firmware execution

- Check controller registers

Memory\_1

Start address

Columns

Data sizes

0x60000160

Auto column

4 bytes

	+0	+4	+8	+C	+10	+14	
0x60000160	0x0000000A	0x00000107	0x40000000	0x00000800	0x00000000	0x00000000	
0x60000178	0x00000000	0x00000000	0x00000000	0x00000000	0x00000016	0x00000000	
0x60000190	0x00000014	0x00000001	0x42F14000	0x00004000	0x00000000	0x0007FF80	0 . B . 0
0x600001A8	0x0007FF80	0x00000000	0x00000000	0x00000000	0x0000000F	0x00000000	
0x600001C0	0x00000014	0x00000001	0x42F14000	0x00004000	0x00000000	0x0007FF80	0 . B . 0
0x600001D8	0x0007FF80	0x00000000	0x00000000	0x00000000	0x00000010	0x00000000	
0x600001F0	0x00000014	0x00000001	0x42F14000	0x00004000	0x00000000	0x0007FF80	0 . B . 0
0x60000208	0x0007FF80	0x00000000	0x00000000	0x00000000	0x00000011	0x00000000	

0x60000160

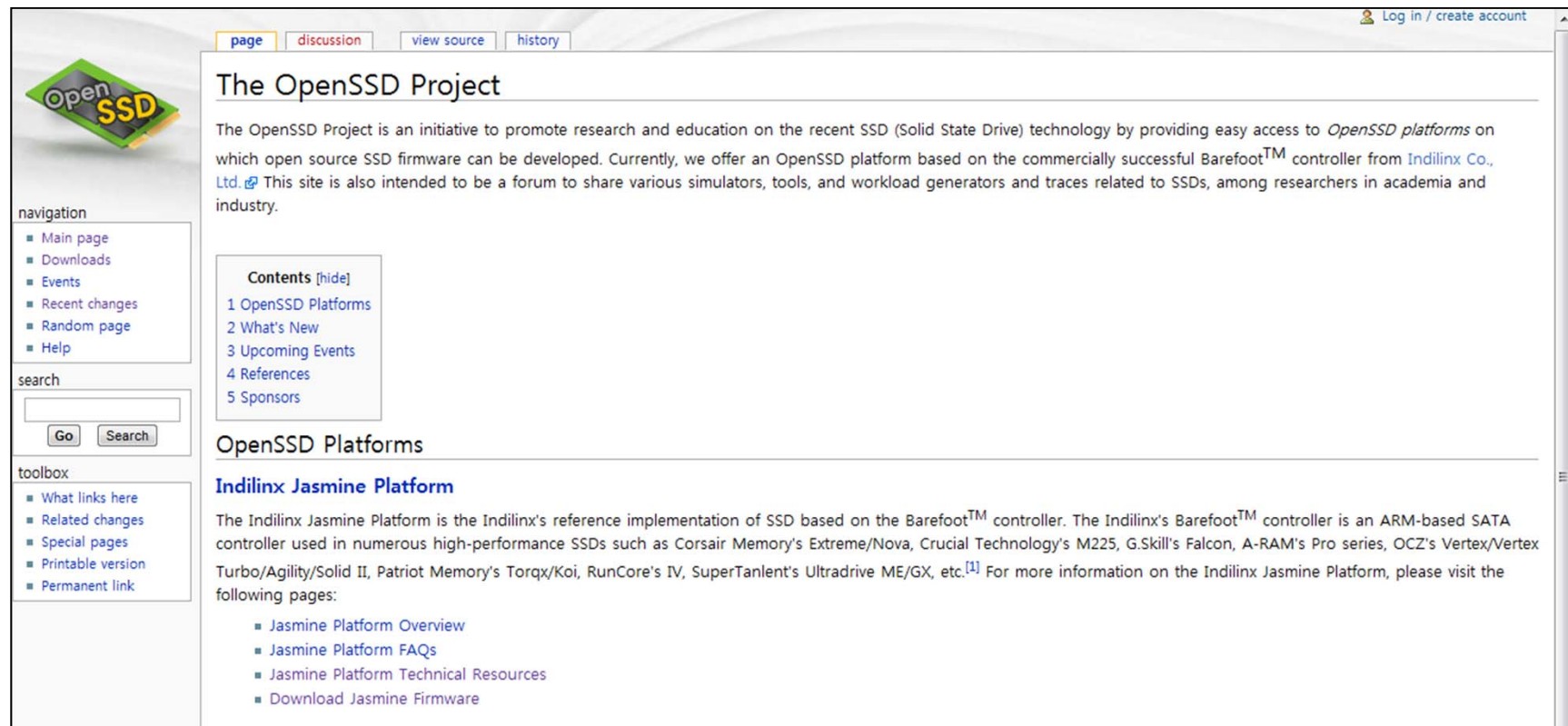
Last issued FC (BSP)

Last issued FC  
(BSP)

# Jasmine Technical Document

- You can download them from the OpenSSD Project homepage
  - <http://www.openssd-project.org>
- **Technical Reference Manual**
  - Jasmine Board Schematics
  - INDILINX Barefoot™ SSD Controller Specification
  - Jasmine Firmware Architecture
  - Jasmine Firmware Software Specification
- **FTL Developer's Guide**
  - FTL Porting Guide
  - Compile, Build & Install Firmware
  - Debugging Tips

# Participate in Our OpenSSD Activities!



The screenshot shows the OpenSSD Project website. At the top, there's a navigation bar with links: [page](#), [discussion](#), [view source](#), and [history](#). In the top right corner, there's a link for [Log in / create account](#). The main heading is "The OpenSSD Project". Below it, a paragraph describes the project: "The OpenSSD Project is an initiative to promote research and education on the recent SSD (Solid State Drive) technology by providing easy access to *OpenSSD platforms* on which open source SSD firmware can be developed. Currently, we offer an OpenSSD platform based on the commercially successful Barefoot™ controller from Indilinx Co., Ltd. This site is also intended to be a forum to share various simulators, tools, and workload generators and traces related to SSDs, among researchers in academia and industry." To the left of the main content, there's a sidebar with a navigation menu: [Main page](#), [Downloads](#), [Events](#), [Recent changes](#), [Random page](#), and [Help](#). Below the navigation menu is a search box with a "Go" button and a "Search" button. Further down is a "toolbox" section with links: [What links here](#), [Related changes](#), [Special pages](#), [Printable version](#), and [Permanent link](#). In the center, there's a "Contents [hide]" section with a list of links: [1 OpenSSD Platforms](#), [2 What's New](#), [3 Upcoming Events](#), [4 References](#), and [5 Sponsors](#). Below this is a section titled "OpenSSD Platforms" with a sub-heading "Indilinx Jasmine Platform". The text describes the Indilinx Jasmine Platform as the Indilinx's reference implementation of SSD based on the Barefoot™ controller. It mentions that the Indilinx's Barefoot™ controller is an ARM-based SATA controller used in numerous high-performance SSDs such as Corsair Memory's Extreme/Nova, Crucial Technology's M225, G.Skill's Falcon, A-RAM's Pro series, OCZ's Vertex/Vertex Turbo/Agility/Solid II, Patriot Memory's Torqx/Koi, RunCore's IV, SuperTalent's Ultradrive ME/GX, etc.<sup>[1]</sup> For more information on the Indilinx Jasmine Platform, please visit the following pages: [Jasmine Platform Overview](#), [Jasmine Platform FAQs](#), [Jasmine Platform Technical Resources](#), and [Download Jasmine Firmware](#).

# Thank you !