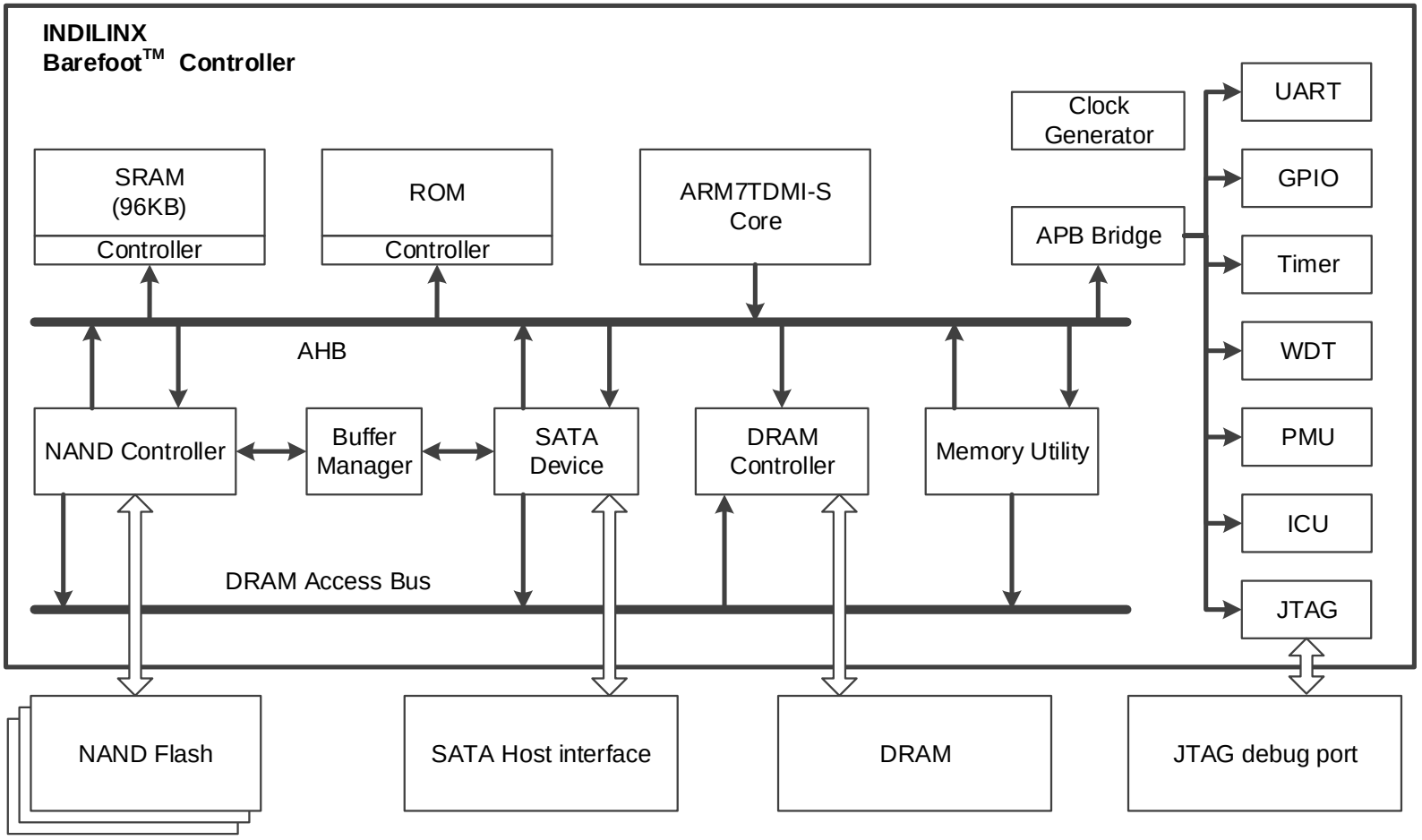
Dummy FTL

Prof. Jinkyu Jeong (Jinkyu@skku.edu)
TA -- Minwoo Ahn (minwoo.ahn@csl.skku.edu)
TA -- Donghyun Kim (donghyun.kim@csl.skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

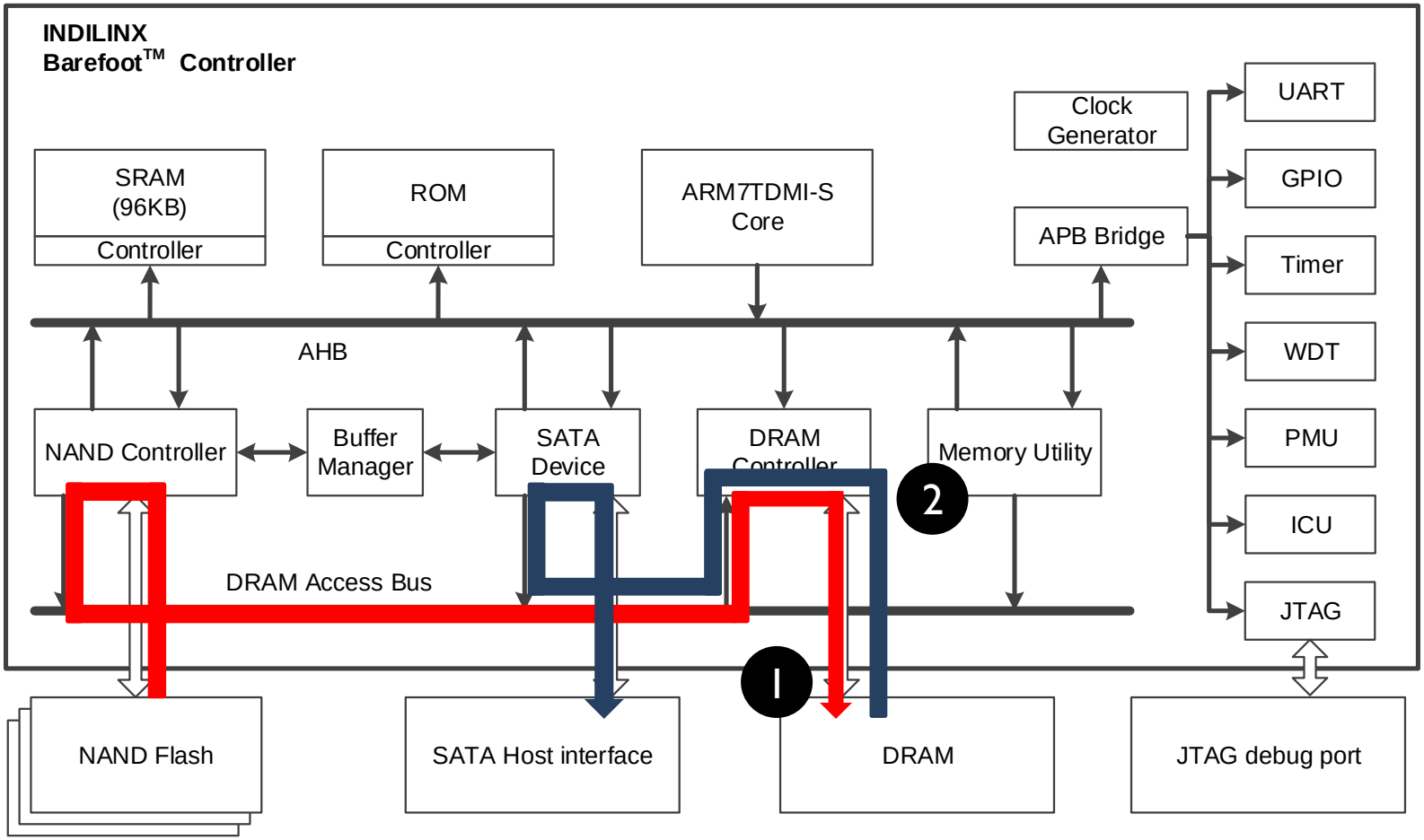
Contents

- Deep dive to the Jasmine & codes
  - Read / write command flow
  - Host interface layer
    - SATA controller
    - Buffer manager
  - Flash translation layer
- Intro. to DummyFTL
  - Request handling in Dummy FTL (code-level)

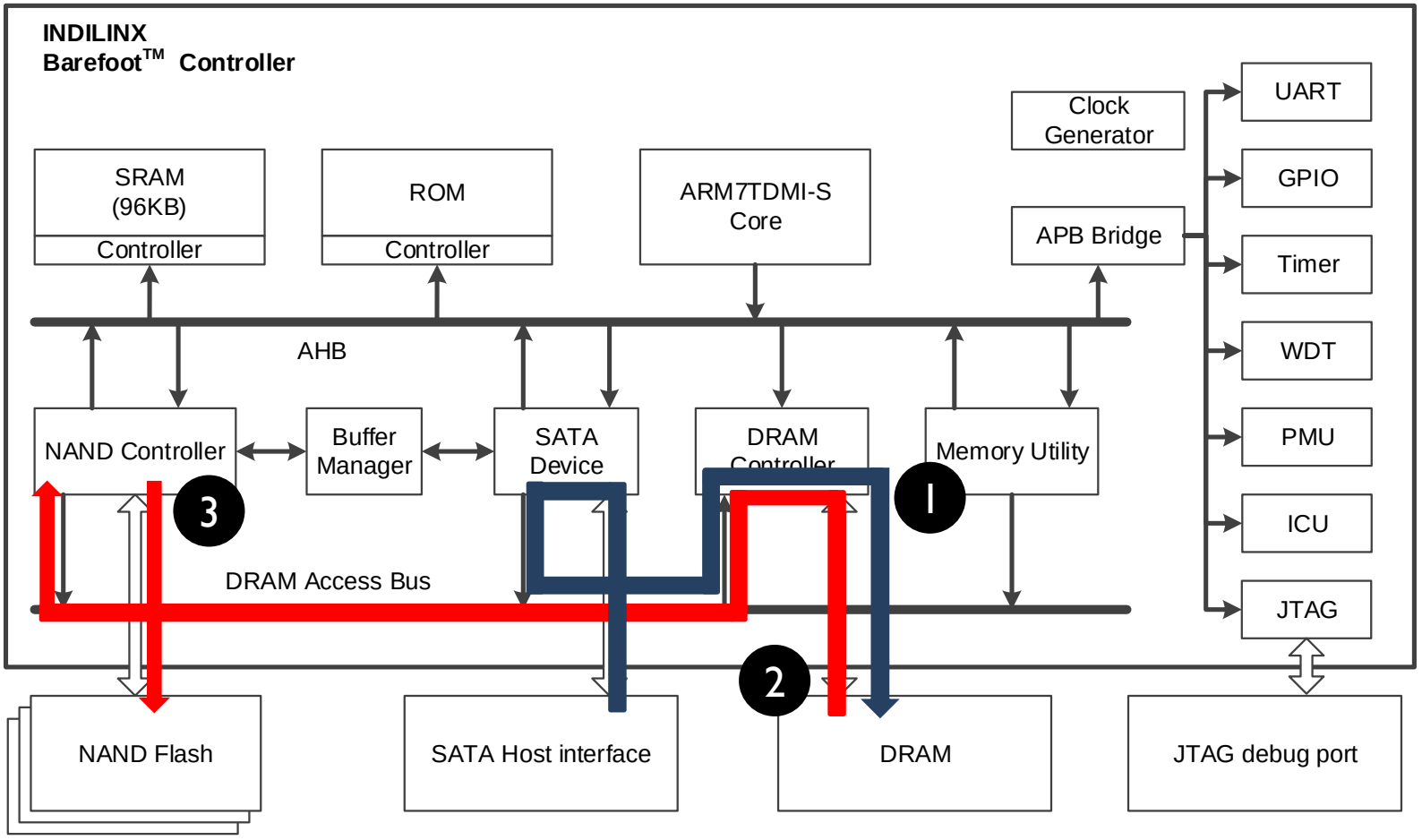
Hardware Architecture



Read Command Flow

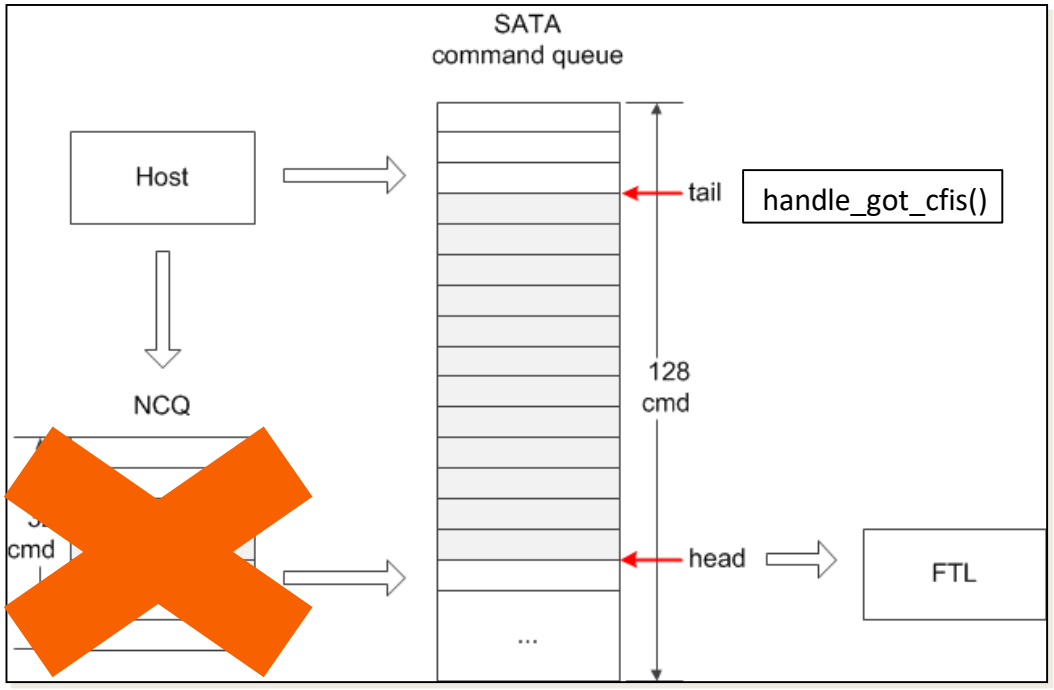


Write Command Flow



SATA Controller

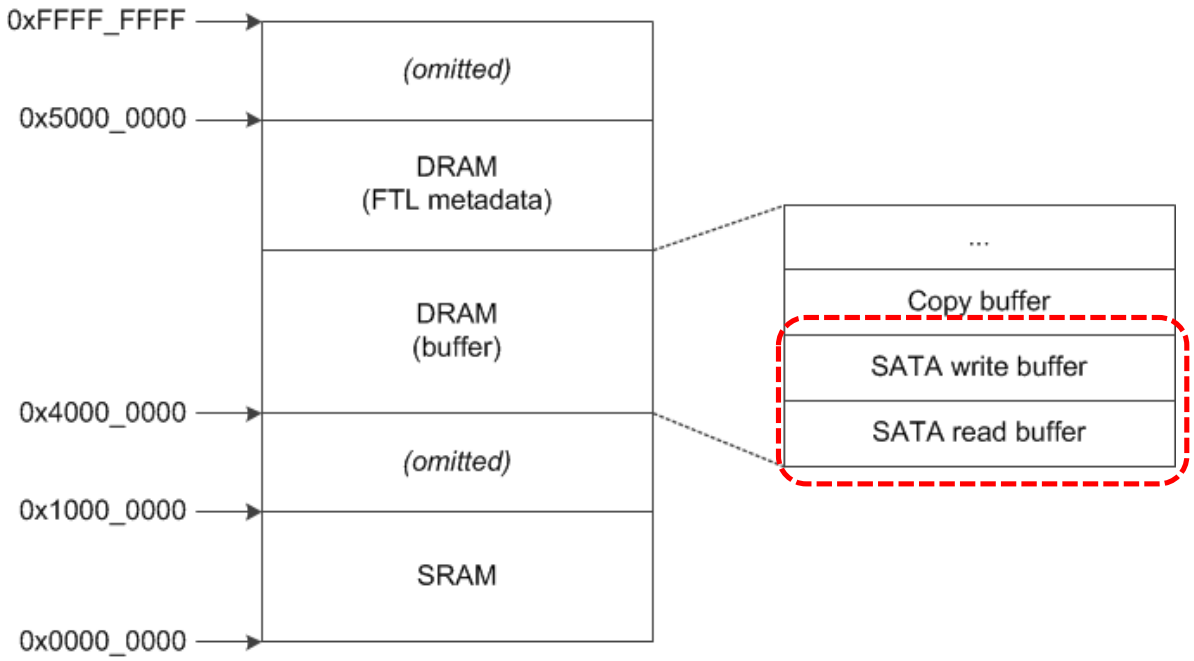
- SATA event queue
  - 128 slots for SATA command
  - Inserted by ISR
  - Deleted by FTL
- Queue policy
  - FIFO
    - Read latency suffers
  - Read first
    - RAW hazard
    - History log by H/W



Disabled in Jasmine

Buffer Manager

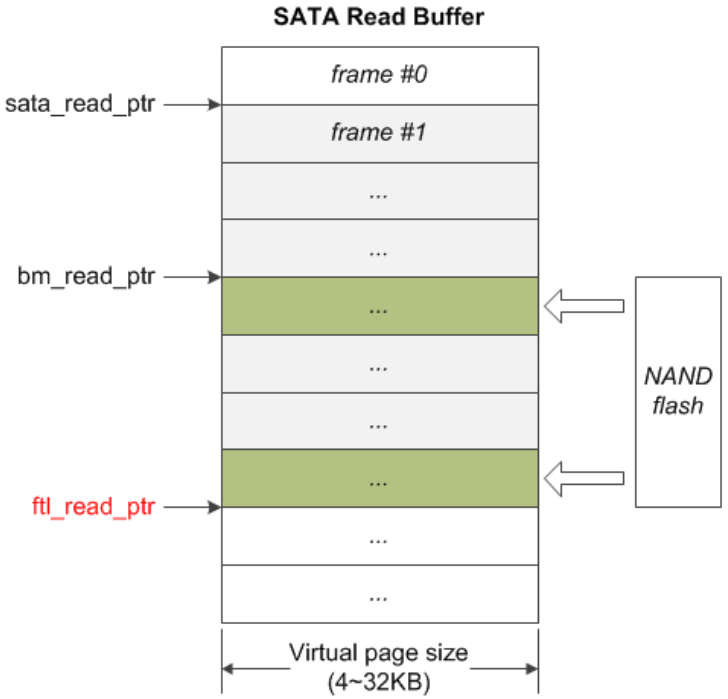
- SATA data is buffered in DRAM
- Memory map of Jasmine board

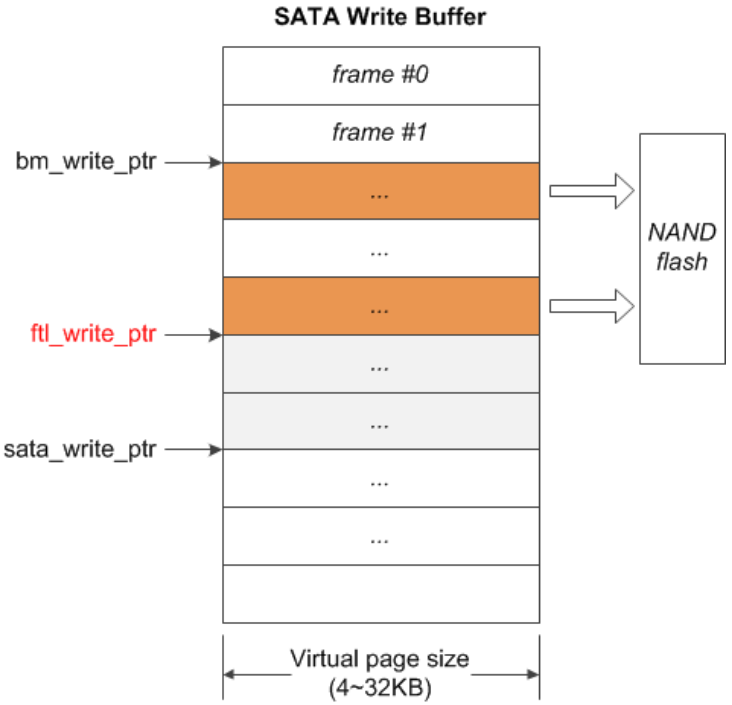


Buffer Manager (cont'd)

- ftl_read_ptr, ftl_write_ptr
  - Transfer data from / to NAND
- sata_read_ptr / sata_write_ptr
  - Transfer data to / from host

Q. Why the order of bm_ptr is different in read and write?





Triggering & Initializing FTL

- ./target_spw/init_gnu.s
  - Call init_jasmine()
  - Call Main()
- init_jasmine()
  - Initialize H/W configurations
- Main()
  - FTL top level loop
  - ./sata/sata_main.c

```
void Main(void)
{
    while (1)
    {
        if (eventq_get_count())
        {
            CMD_T cmd;

            eventq_get(&cmd);

            if (cmd.cmd_type == READ)
            {
                ftl_read(cmd.lba, cmd.sector_count);
            }
            else
            {
                ftl_write(cmd.lba, cmd.sector_count);
            }
        }
        else if (g_sata_context.slow_cmd.status == SLOW_CMD_STATUS_PENDING)
        {
            void (*ata_function)(UINT32 lba, UINT32 sector_count);

            slow_cmd_t* slow_cmd = &g_sata_context.slow_cmd;
            slow_cmd->status = SLOW_CMD_STATUS_BUSY;

            ata_function = search_ata_function(slow_cmd->code);
            ata_function(slow_cmd->lba, slow_cmd->sector_count);

            slow_cmd->status = SLOW_CMD_STATUS_NONE;
        }
        else
        {
            // idle time operations
        }
    } ? end while 1 ?
} ? end Main ?
```

Dummy FTL

- ./ftl_dummy
  - ftl.c, ftl.h
- Dummy FTL is not a real FTL
  - No access to NAND flash
  - Neither stores nor retrives any data
- Why Dummy FTL?
  - To simply measure the SATA & DRAM speed

How to Enable Dummy FTL

- ./build_gnu/Makefile

Dummy FTL: Read Handling

- ./ftl_dummy/ftl.c

```
void ftl_read(UINT32 const lba, UINT32 const total_sectors)
{
    UINT32 num_sectors_to_read;

    UINT32 lpage_addr       = lba / SECTORS_PER_PAGE;    // logical page address
    UINT32 sect_offset      = lba % SECTORS_PER_PAGE;    // sector offset within the page
    UINT32 sectors_remain   = total_sectors;

    while (sectors_remain != 0) // one page per iteration
    {
        if (sect_offset + sectors_remain < SECTORS_PER_PAGE)
        {
            num_sectors_to_read = sectors_remain;
        }
        else
        {
            num_sectors_to_read = SECTORS_PER_PAGE - sect_offset;
        }

        UINT32 next_read_buf_id = (g_ftl_read_buf_id + 1) % NUM_RD_BUFFERS;

        while (next_read_buf_id == GETREG(SATA_RBUF_PTR));   // wait if the read buffer is full (slow host)

        SETREG(BM_STACK_RDSET, next_read_buf_id);    // change bm_read_limit
        SETREG(BM_STACK_RESET, 0x02);                // change bm_read_limit

        g_ftl_read_buf_id = next_read_buf_id;

        sect_offset = 0;
        sectors_remain -= num_sectors_to_read;
        lpage_addr++;
    } ? end while sectors_remain! =0 ?
} ? end ftl_read ?
```

Dummy FTL: Write Handling

- ./ftl_dummy/ftl.c

```
void ftl_write(UINT32 const lba, UINT32 const total_sectors)
{
    UINT32 num_sectors_to_write;

    UINT32 sect_offset = lba % SECTORS_PER_PAGE;
    UINT32 remain_sectors = total_sectors;

    while (remain_sectors != 0)
    {
        if (sect_offset + remain_sectors >= SECTORS_PER_PAGE)
        {
            num_sectors_to_write = SECTORS_PER_PAGE - sect_offset;
        }
        else
        {
            num_sectors_to_write = remain_sectors;
        }

        while (g_ftl_write_buf_id == GETREG(SATA_WBUF_PTR));     // bm_write_limit should not outpace SATA_WBUF_PTR

        g_ftl_write_buf_id = (g_ftl_write_buf_id + 1) % NUM_WR_BUFFERS;     // Circular buffer

        SETREG(BM_STACK_WRSET, g_ftl_write_buf_id); // change bm_write_limit
        SETREG(BM_STACK_RESET, 0x01);               // change bm_write_limit

        sect_offset = 0;
        remain_sectors -= num_sectors_to_write;
    } ? end while remain_sectors! =0 ?
} ? end ftl_write ?
```

To End up Today Class

- Q. Why the order of bm_ptr is different in read and write?

- Q. What is for 'copy buffer' of buffer region in DRAM? (#7 slide)
  - Hint) http://www.openssd-project.org/mediawiki/images/Jasmine_Tech_Ref_Manual_v.1.4.pdf

- Answer each question briefly
- Email me with your own answers in each group
  - Due: 12:00(정오), 10/11
  - donghyun.kim@csl.skku.edu

Any Questions?