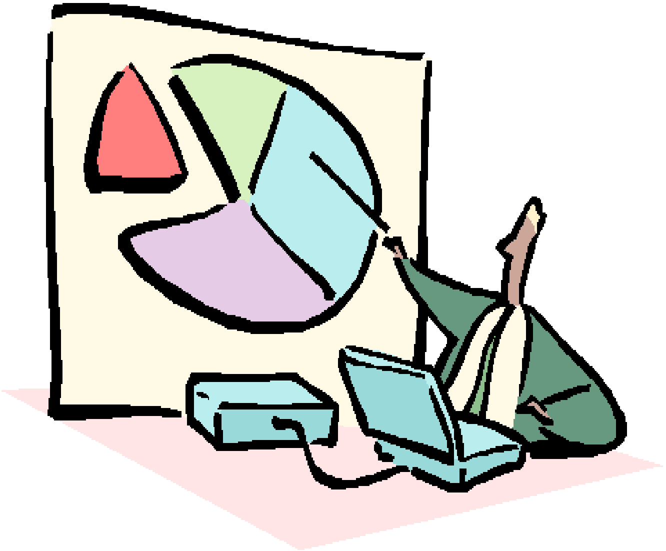
Internship Task Guideline: Tutorial FTL 2 design #1

Sang-Phil Lim @ VLDB Lab., SKKU

2012.01.06



Task Description: Tutorial FTL 2

- **Tutorial FTL 1 on the Jasmine platform**
  - NO Normal & Sudden POR support
  - NO Garbage collection operation
  - NO Runtime bad block management
- **Task: Tutorial FTL 2 implementation**
  - Normal POR support
  - 512Byte sector-level address mapping
    - For improving random write performance

Tutorial FTL 2 Design: Address mapping

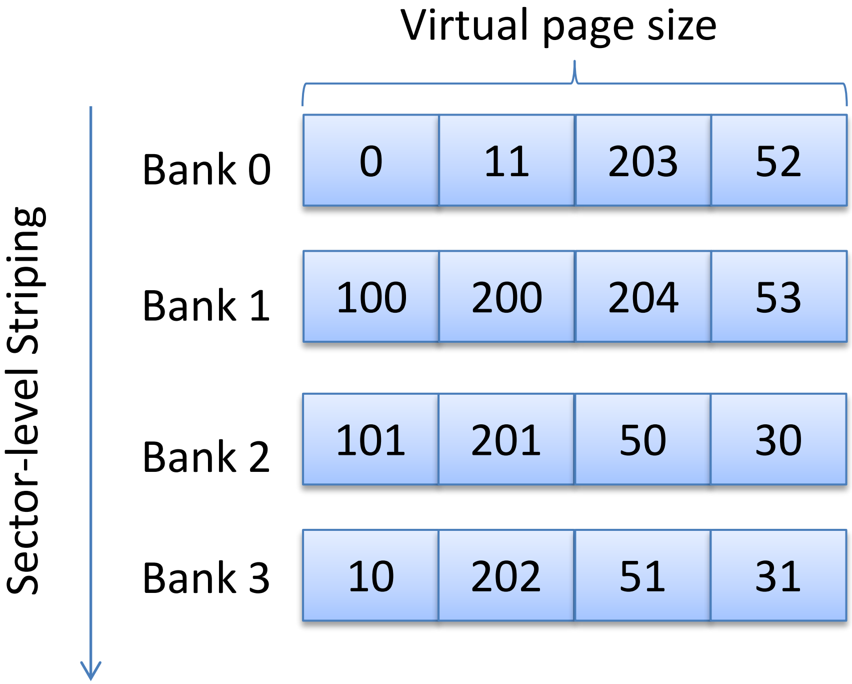
- **Problem statements**
  - Write commands are randomly queued into the SATA event queue
  - NOP of NAND flash memory is 1
  - To improve the random write performance, we should maximize the NAND flash utilization
- **Solution**
  - **Sector-level Data Striping in ‘Secondary Merge Buffer’**
  - We queue small size of write commands as possible as we can, and merge them for maximizing the NAND flash utilization as adopting the ‘Secondary Merge Buffer’
  - This buffer is managed by FTL

Tutorial FTL 2 Design: Address mapping

- **Sector-level Data Striping with 'Secondary Merge Buffer'**
  - Maximize write performance & NAND flash utilization
    - On demand free space allocation: Not fixed write position(i.e., bank #) of LSN
  - Interleaved read operation even in random read operations
    - If we strip the data in 'Sector-level' , the read performance could be maximized
    - NOTE: Need to verify the performance

Tutorial FTL 2 Design: Address mapping

- **Walkthrough: Handling write requests**
  - <W, 0, 1>, <W, 100, 2>, <W, 10, 2>, <W, 200,4>, <W,50,4>, <W,30,2>



Bank 0	0	11	203	52
Bank 1	100	200	204	53
Bank 2	101	201	50	30
Bank 3	10	202	51	31

2. When the buffer is full, call `ftl_write()`
 - alloc. new_vpn & write to NAND

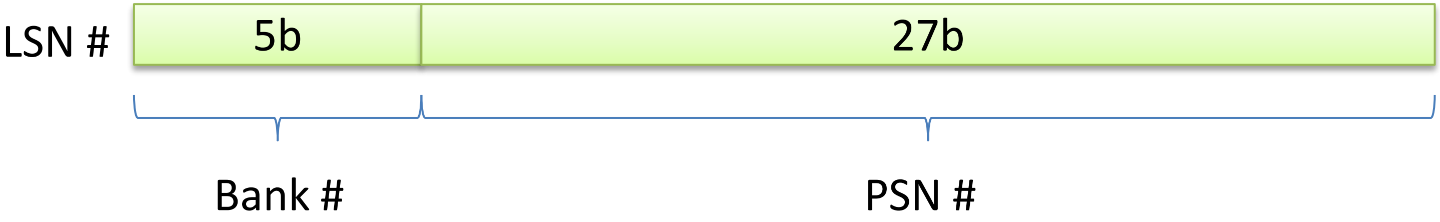
1. Write requests are queued in the 'Secondary Merge Buffer' first
 (including new data sending from SATA)

Tutorial FTL 2 Design: Address mapping

- **Research issues**
  - Read performance might be decreased
    - Consecutive LSNs could be written into the same bank
  - Buffer area size of 'Secondary Merge Buffer'
    - Note that the role of 'Secondary Merge Buffer' is not used to reduce the data quantity of write requests
    - Thus, the virtual page size * # of banks would be OK
  - As the secondary host data buffer cache (*optional*)
    - 'Secondary Merge Buffer' can be extended as a secondary buffer cache
    - In this case, buffer management algorithm is required (e.g., Hit case)

Tutorial FTL 2 Design: Address mapping

- **Address translation**
  - Basically, LSN to PSN
- **Memory layout of a sector-level address mapping table**



NOTE: 27bits for storing a PSN are reasonably sufficient (8192 blocks)

Tutorial FTL 2 Design: Mapping Table Caching

- **However, the size of sector-level address mapping table is too huge to store a whole mapping information in DRAM**
- **Thus, we need to cache a part of address mapping information on DRAM (or SRAM) in runtime**
- **Reference paper** *(recommended)*
  - Aayush Gupta et al, "DFTL: a flash translation layer employing demand-based selective caching of page-level address mappings", ASPROS '09
- **This is also connected with a 'Normal POR' issue**

Tutorial FTL 2 Design: Mapping Table Caching

- **Research Issues**
  - How much DRAM size should be reserved for caching the mapping table?
  - Caching mapping table size?
  - Which updated mapping information should be evicted (flushed) from the mapping cache pool?
    - Simply, a LRU policy is considerable

Tutorial FTL 2 Design: Garbage Collection

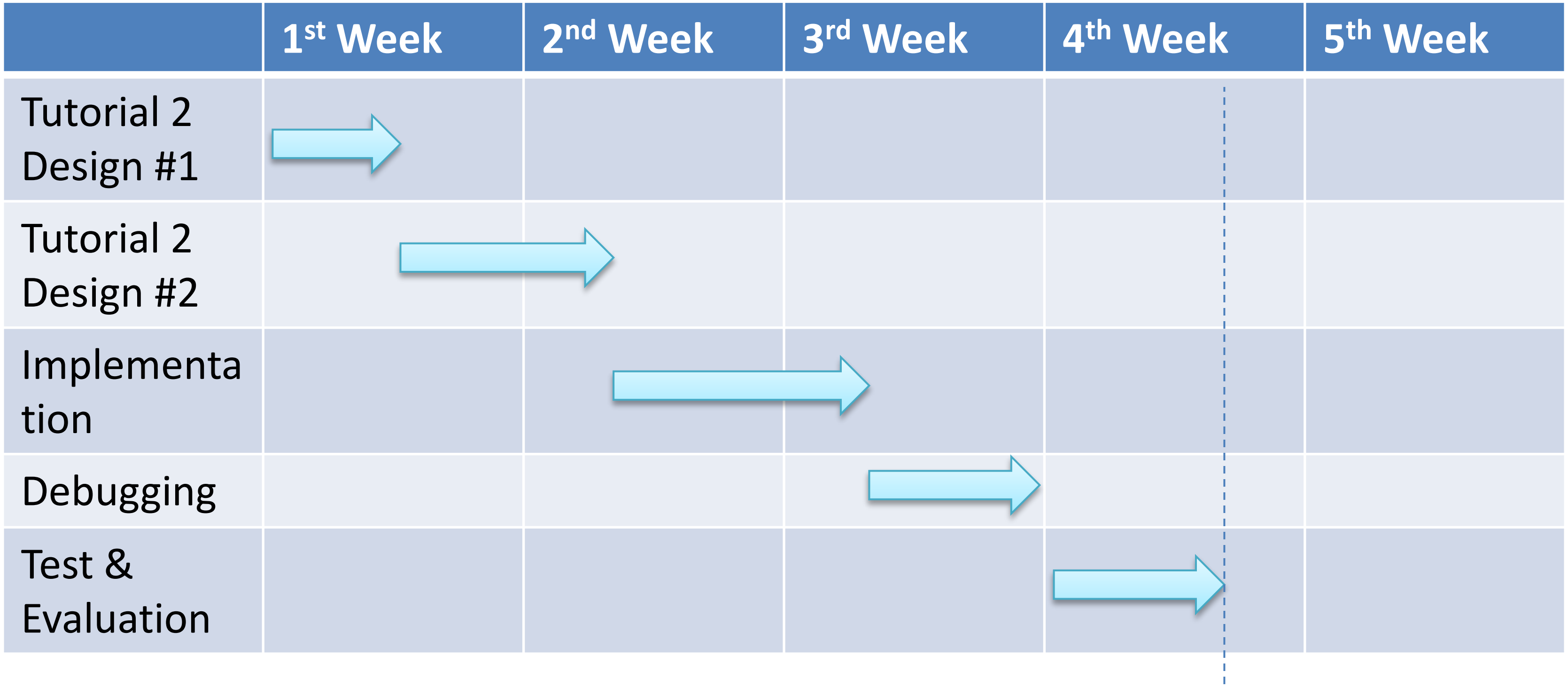
- **Garbage collection operation in Tutorial FTL2 is simple**
  - As same approach as in ‘Greedy FTL’ of the Jasmine firmware
- **Store a LSN list into the last page of each virtual block**
  - We can verify the valid ‘sector’s by comparing this info. with Sector-level mapping info.
  - When the GC is triggered, we need to load these info. from NAND

Tutorial FTL 2 Design: Garbage Collection

- **Research issues**
  - Page internal fragmentation
    - ***Most of sectors in a page are invalid. But the page is not invalid***
    - That means, in the virtual page-level, the invalidation ratio might be low
    - How can we reduce a percentage of the page internal fragmentation?
  - GC overhead
    - Sector-level GC
    - FC_COPYBACK vs. write-after-read op. (FC_COL_ROW_READ_OUT, FC_COL_ROW_IN_PROG)
    - There're consecutive or separated valid sectors in each pages

Task Schedule

The end of the internship: Feb 7, 2012



	1st Week	2nd Week	3rd Week	4th Week	5th Week
Tutorial 2 Design #1	→				
Tutorial 2 Design #2	→	→			
Implementation		→	→		
Debugging			→		
Test & Evaluation				→	

- Tutorial 2 Design #1: Conceptual design
- Tutorial 2 Design #2: Implementation(Functional) design