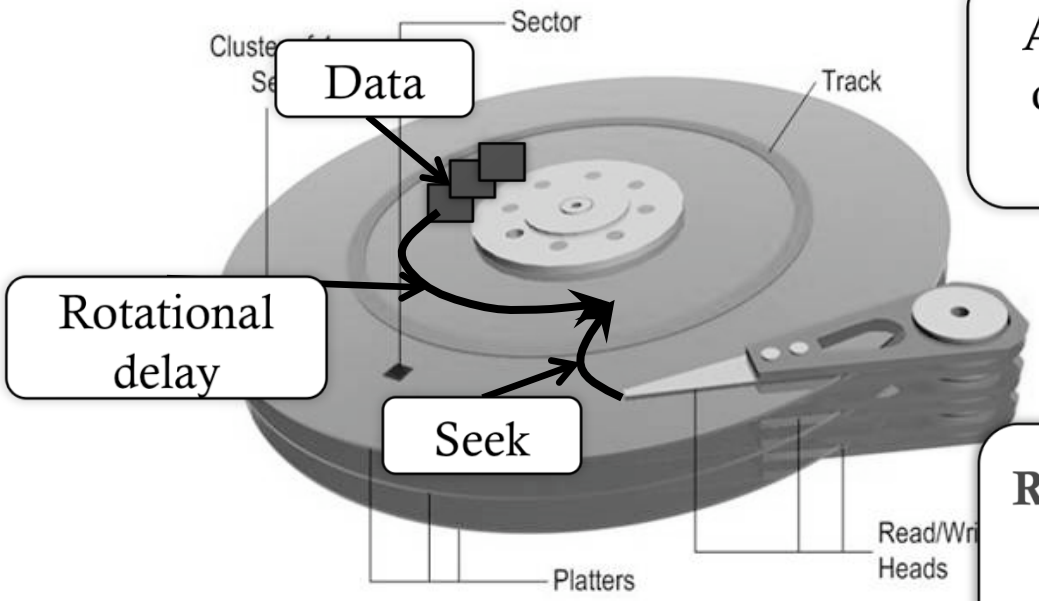
Operating Systems ECE344

Lecture 13: Solid-State Drive

Ding Yuan

Review

- Disk



Access data sequentially: only suffer one seek and rotational delay

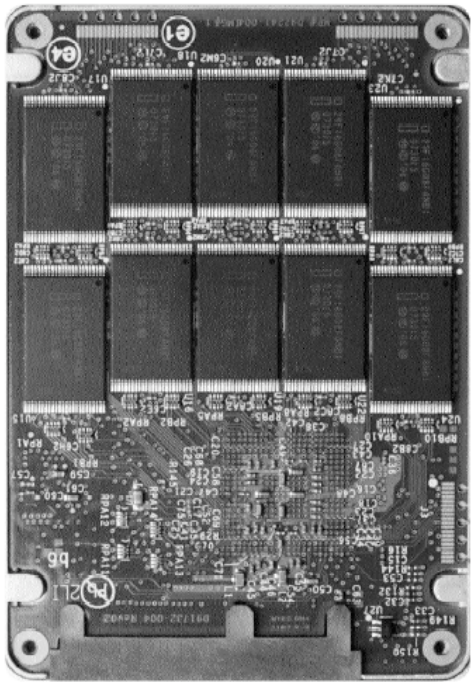
Random disk access: suffers one seek and rotational delay every time!

Fundamental Limitation with HD

- Mechanical
  - Cannot be made too fast
  - Latency in milliseconds

Solid-State Drive (SSD)

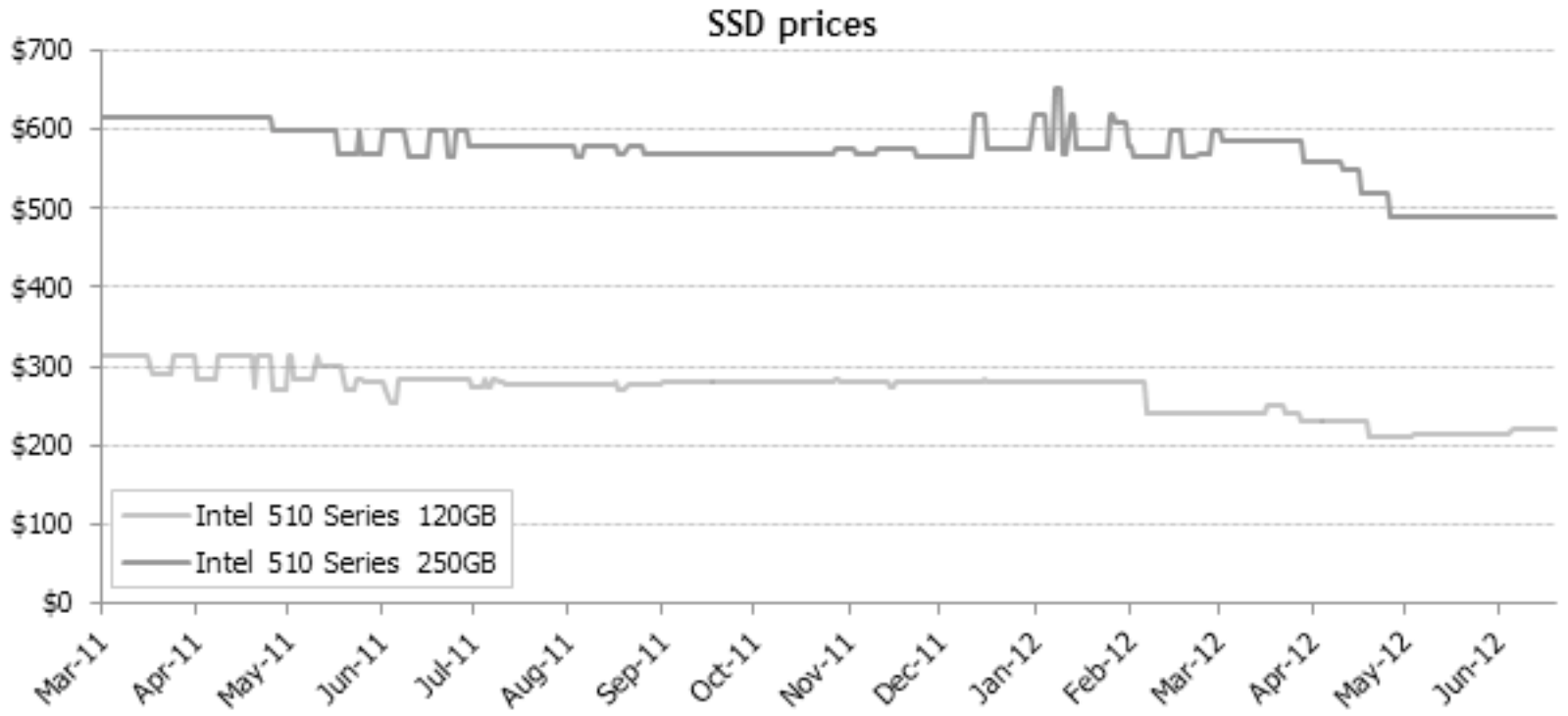
- Nonvolatile memory (Floating-gate transistors)
  - This gives the device the "solid-state" name
  - Most common media: NAND Flash memory



SSD: 2013

- Intel 520 Cherryville (laptop)
  - Capacity: 240 GB
    - Compare to HD: TB
  - Sequential R/W: 550/520 MB/s
    - Compare to HD: 122 MB/s
  - Latency
    - Read: 0.04 ms
    - Write: 0.2 ms
    - Compare to HD: tens of ms
  - $1/GB
    - Compare to HD: 0.06/GB

Price declining



Limitations

- Random write performance
  - For write, need to first erase a large of pages (32-64 pages), and reprogram
- Burnout
  - Each cell has limited erase/program cycles
    - Range from 1,000 – 100,000 writes

How do SSD's characteristics impact FS design?

- Characteristics of SSD
  - No mechanical components --- data location won't make difference
    - Optimizations on data location are no longer useful
  - Random write performance is BAD
    - Avoid random writes!
  - Limited "write" cycles for each cell
    - Spread the writes across the entire SSD

Log-structured File System

- Many popular Flash File Systems are log-structured File System
  - Linux JFFS/JFFS2/YAFFS/LogFS…
- The "output" (interface) of LFS is the same
  - File, directories
    - Same logical storage
- Data layout is very different

LFS Approach

- Treat the drive as a single log for appending
  - Collect writes in disk cache, write out entire collection in one large I/O request
  - All info written to drive is appended to log
    - Data blocks, attributes, inodes, directories, etc.
- Simple, eh?
  - Alas, only in abstract

LFS Challenges

- LFS has two challenges it must address for it to be practical
  1. Locating data written to the log
     - FFS places files in a location, LFS writes data "at the end"
  2. Managing free space on the disk
     - Disk is finite, so log is finite, cannot always append
     - Need to recover deleted blocks in old parts of log

LFS: Locating Data

- FFS uses inodes to locate data blocks
  - Directories contain locations of inodes
- LFS appends inodes to end of the log just like data
  - Makes them hard to find
- Approach
  - Use another level of indirection: Inode maps
  - Inode maps map inode #s to inode location
  - Location of inode map blocks kept in checkpoint region
  - Checkpoint region has a fixed location
  - Cache inode maps in memory for performance

LFS Layout

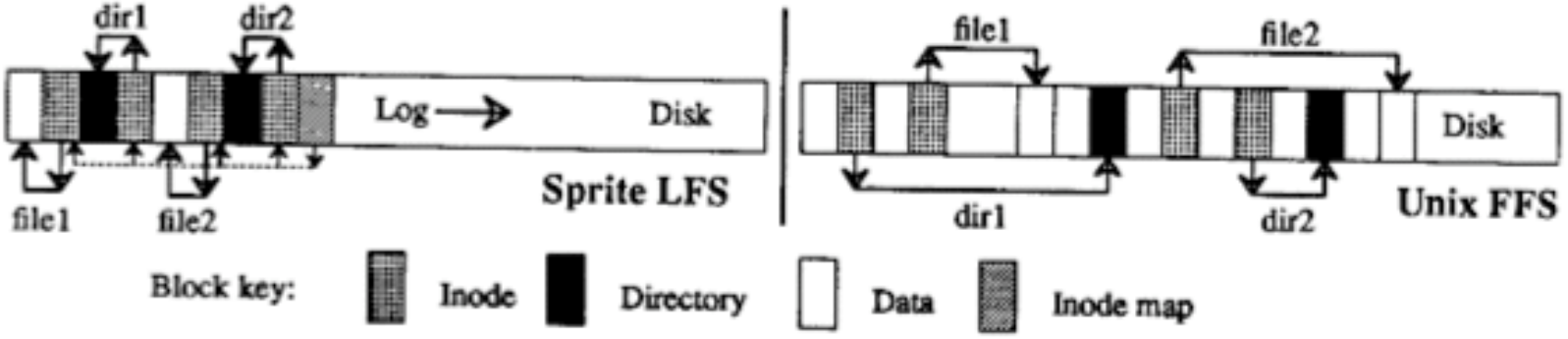


Fig. 1. A comparison between Sprite LFS and Unix FFS. This example shows the modified disk blocks written by Sprite LFS and Unix FFS when creating two single-block files named dir1/file1 and dir2/file2. Each system must write new data blocks and inodes for file1 and file2, plus new data blocks and inodes for the containing directories. Unix FFS requires ten nonsequential writes for the new information (the inodes for the new files are each written twice to ease recovery from crashes), while Sprite LFS performs the operations in a single large write. The same number of disk accesses will be required to read the files in the two systems. Sprite LFS also writes out new inode map blocks to record the new inode locations

LFS: Free Space Management

- LFS append-only quickly runs out of disk space
  - Need to recover deleted blocks
- Approach:
  - Fragment log into segments
  - Reclaim space by cleaning segments
    - Read segment
    - Copy live data to end of log
    - Now have free segment you can reuse
- Cleaning is a big problem
  - Costly overhead

Why LFS is a better fit for SSD?

- Characteristics of SSD
  - No mechanical components --- data location won't make difference
    - LFS does not try to group data of the same file together
  - Random write performance is BAD
    - No random write, only large write (to the end of log)
  - Limited "write" cycles for each cell
    - Always write to a different location (end of log)
    - No in-place write

LFS is not new

- Invented long before the advent of SSD
  - Published in SOSP'91
- Hard Disk can also benefit
  - Why?

Further reading

- Anatomy of a Solid-State Drive
  - http://queue.acm.org/detail.cfm?id=2385276
- JFFS: The Journalling Flash File System
  - http://sourceware.org/jffs2/jffs2-html/
- The Design and Implementation of a Log-Structured File System
  - http://www.stanford.edu/~ouster/cgi-bin/papers/lfs.pdf