Jin-Soo Kim
(*jinsoo.kim@snu.ac.kr*)
Systems Software &
Architecture Lab.
Seoul National University

Spring 2024

Log-Structured File System

(M. Rosenblum and J. K. Ousterhout, SOSP 1991)

Technology Trends

- CPU speed is increasing at an exponential rate
- Main memory size is increasing at an exponential rate
- Disk technology is also improving rapidly, in the areas of cost and capacity rather than performance
  - Disk bandwidth can be improved substantially with the use of disk arrays and parallel-head disks
  - No major improvements for access time

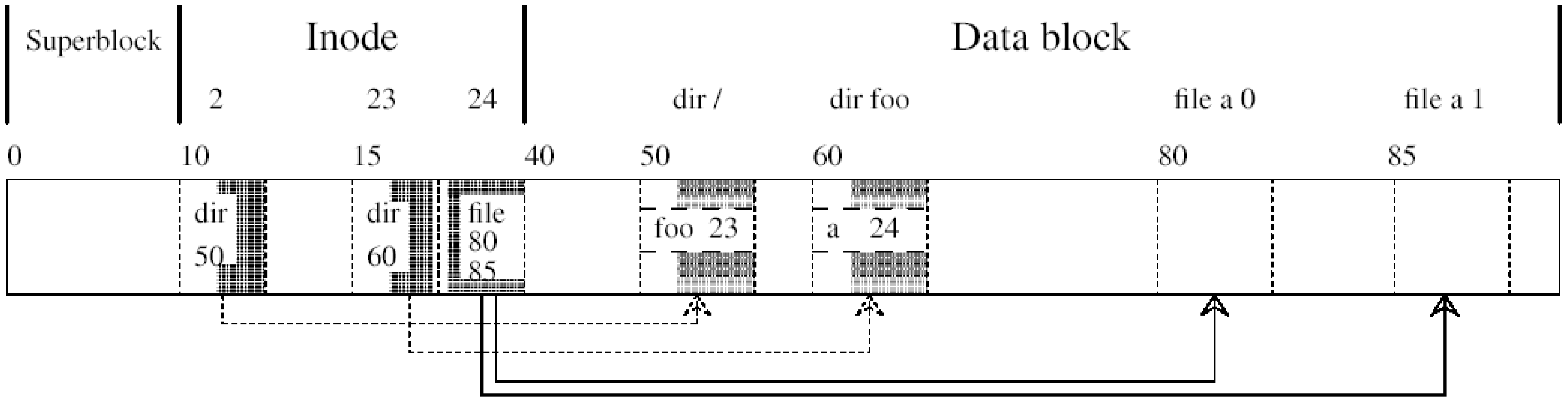
Workloads Trends

- Office and engineering applications tend to be dominated by accesses to small files
  - Small random disk I/Os
  - File creation and deletion times are dominated by updates to file system metadata

- Larger main memories make larger file caches possible
  - Disk traffic will become more and more dominated by __________
  - They can serve as write buffers; large numbers of modified disk blocks can be collected

FFS Example

- Reading "/foo/a"



Problems

- Too many ________ accesses
  - The inode for a file and the directory entry containing the file's name are separated from the file's contents
  - Several disk I/Os needed to create a new file

- The disk traffic is dominated by ___________ metadata writes
  - Directories and inodes are written synchronously
  - Defeat the potential use of the file caches as a write buffer

Sprite LFS

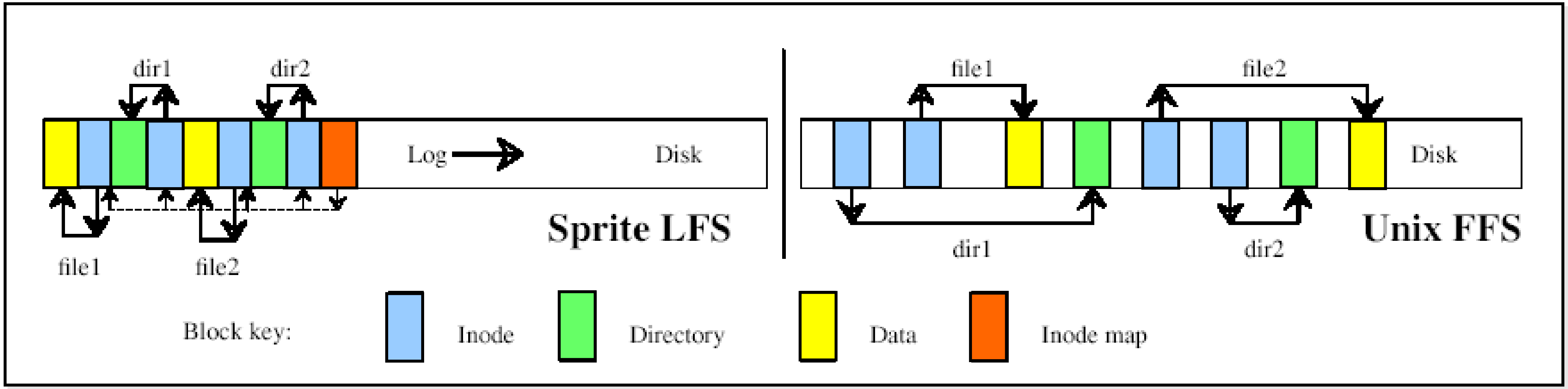
- Buffer a sequence of file system changes in the file cache
- Write all the changes to disk in a sequential structure called the _______

- Improves the write performance by eliminating almost all seeks
- The sequential nature of logs permits much faster _______________

- How to retrieve information from the log?
- How to manage the free space on disk?

Inode

- The inode structure is same
  - File attributes
  - Disk addresses of the first ten blocks
  - Indirect and double indirect blocks for handling large files
- LFS inodes are not in fixed locations on disk



Inode Map

- Maintain the current location of each inode
  - An array indexed by the file's i-number
  - <the current inode address, flags, version number, the last access time of the file>
  - The maximum number of inode map entries is fixed
    (BSD LFS uses inode file: the maximum number of inodes can grow)

- Divided into blocks that are written to the log much like file data blocks
  - The locations of the inode map blocks are kept in a fixed ______________ on disk
  - Frequently accessed blocks of the inode map are cached
  - Hit rate for inode map reference: 99.1% ~ 99.9%
    (On average only 15% of the inode map blocks is cached)

Segments

- The disk is divided into fixed-size segments
  - All live data must be copied out of a segment before the segment can be rewritten
  - The log can be ____________ through clean segments
- Segments should be large enough
  - The transfer time to read/write a whole segment should be much greater than the cost of a ____________
  - 512KB or 1MB in Sprite LFS
- Segment cleaning
  - Generates clean segments from segments containing live blocks

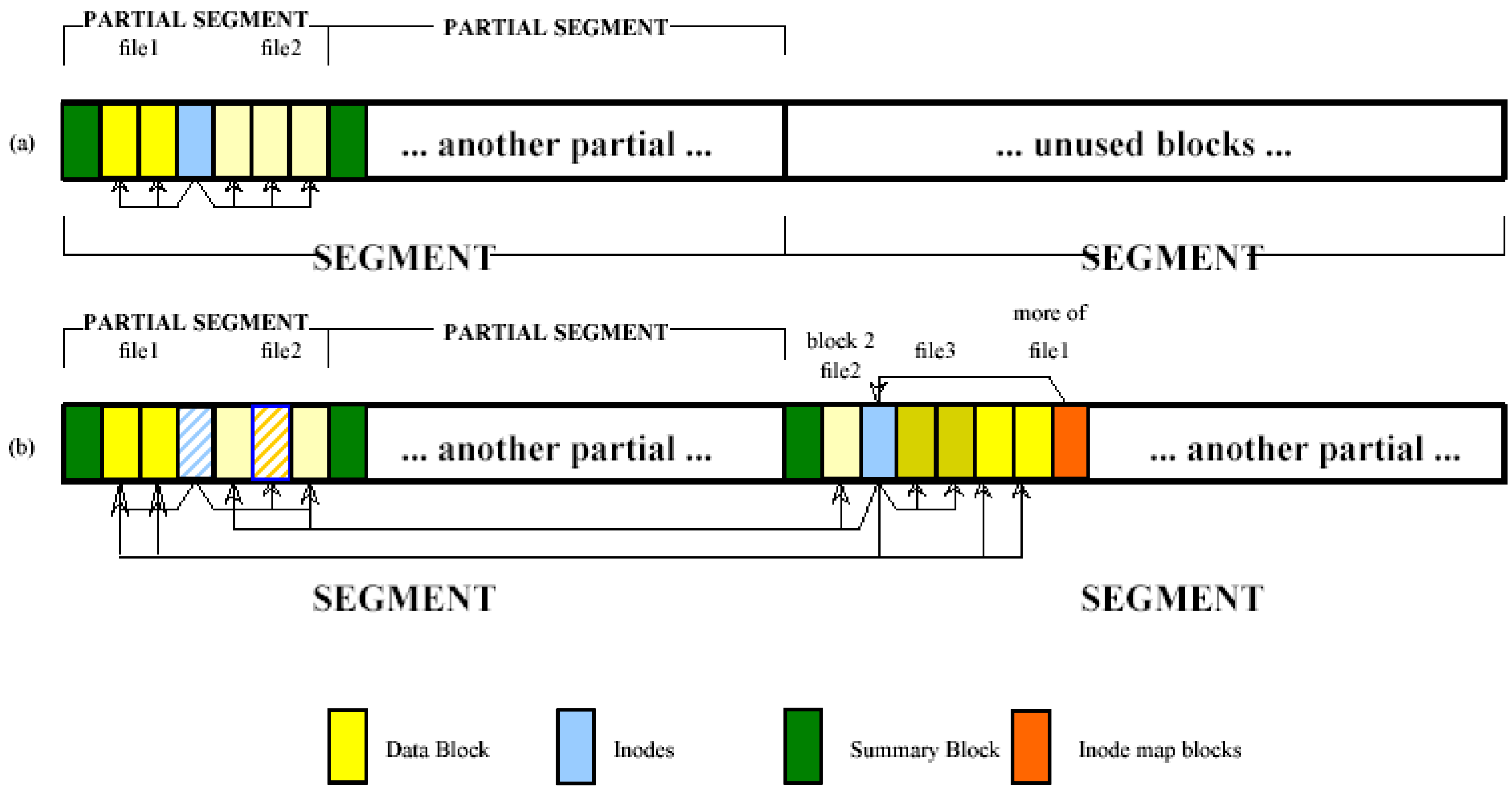
Segment Summary Block

- Used to identify live blocks
  - <file's i-number, version number, block number within the file> for each data block
  - Used to find and update the file's inode to reflect the block's new location during segment cleaning

- A block is …
  - Live if the block is still pointed by ____________________
    - No free-block list or bitmap needed → save space, simplify crash recovery
  - Not live if the file's version number (in the segment summary block) does not match the version number stored in __________________
    - The version number is incremented whenever the file is deleted or truncated to length zero
    - No need to examine the file's inode to discard such a block

Example

- File 2 modified, file 3 created, and two blocks append to file 1



Write Cost

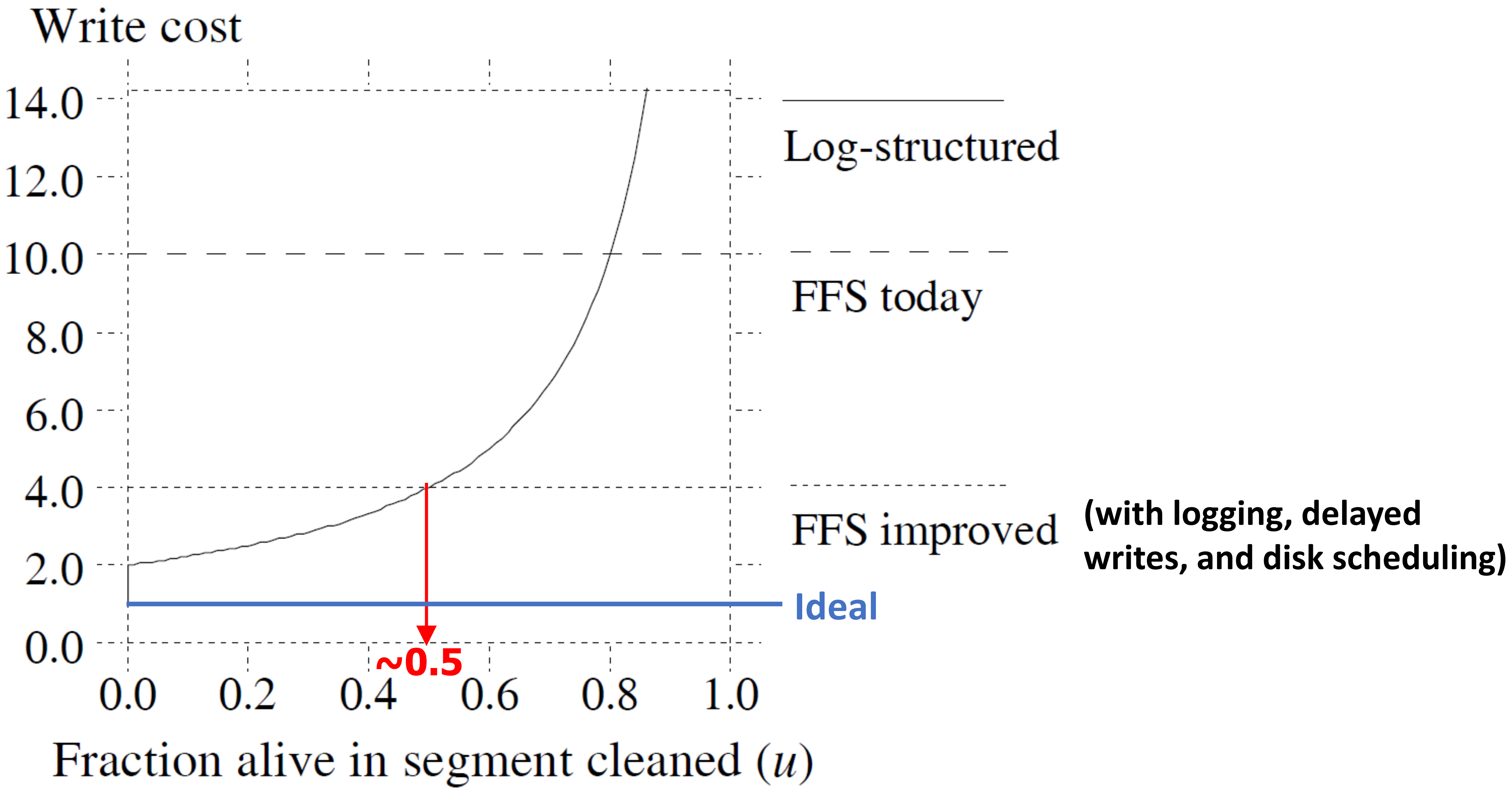
- The performance metric for cleaning policies
  - The average amount of time the disk is busy per byte of new data written, including all the cleaning overheads

$$\textit{write cost} = \frac{\textit{total bytes read and written}}{\textit{new data written}} = \frac{\textit{read segs} + \textit{write live} + \textit{write new}}{\textit{new data written}}$$

$$= \frac{N + N * u + N * (1 - u)}{N * (1 - u)} = \frac{2}{1 - u}$$

  - Utilization *u*: the fraction of data still live in segments
  - Seek and rotational latency are negligible in LFS
  - Normalized to the ideal write time (no cleaning, no seek time or rotational delay)

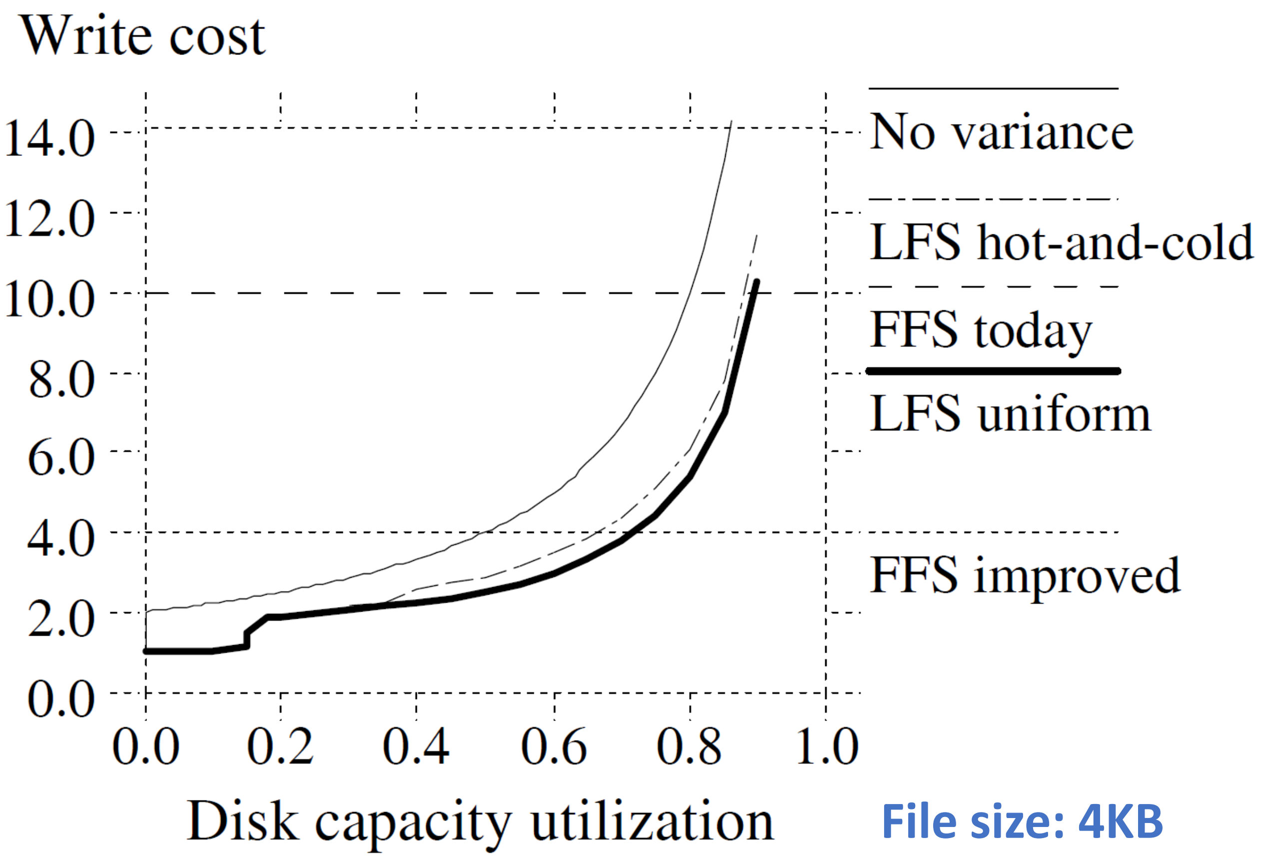
Utilization vs. Write Cost



Greedy Policy

- Choose the ___________ segments

- LFS Uniform
  - Uniform file access patterns
- LFS Hot-and-cold
  - 10% of files are accessed 90% of the time
  - Sorts the live data by age before writing it out again
- No variance
  - All segments always have exactly the same utilization



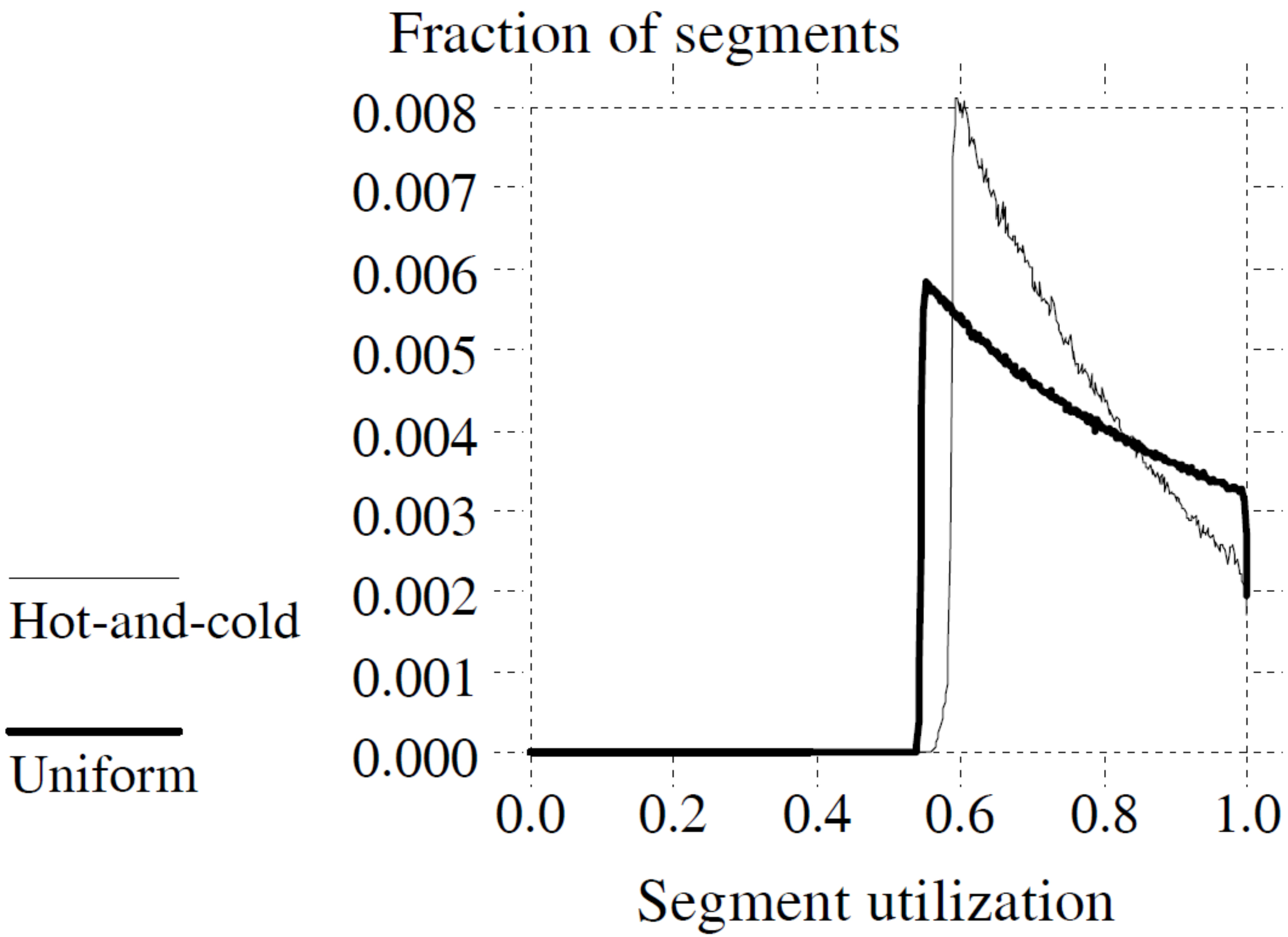
File size: 4KB

Greedy Policy: Analysis

- "Hot-and-cold" is worse than "Uniform"!
  - ***What's wrong?***

?

Disk capacity utilization: 75%



Cost-Benefit Policy

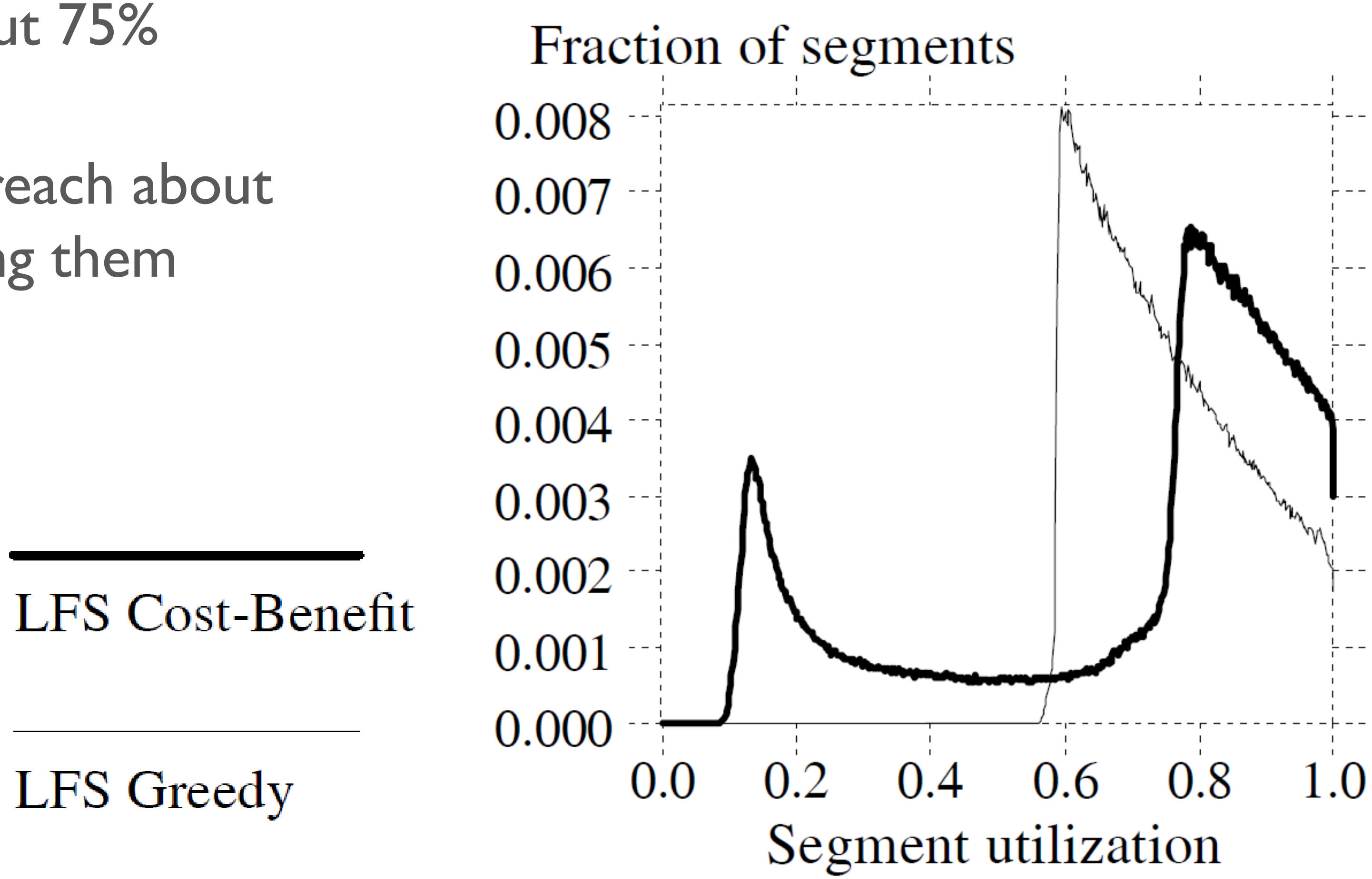
- Choose the segment with the __________ ratio of benefit to cost

$$\frac{benefit}{cost} = \frac{free\ space\ generated\ *\ age\ of\ data}{cost} = \frac{(1-u)*age}{1+u}$$

- Free space in a _______ segment is more valuable than free space in a _______ segment
  - Once a cold segment has been cleaned, it will take a long time before it reaccumulates the unusable free space
  - It is less beneficial to clean a hot segment because the data will likely die quickly
  - The most recent modified time of any block in the segment is used as an estimate of how long the space is likely to stay free

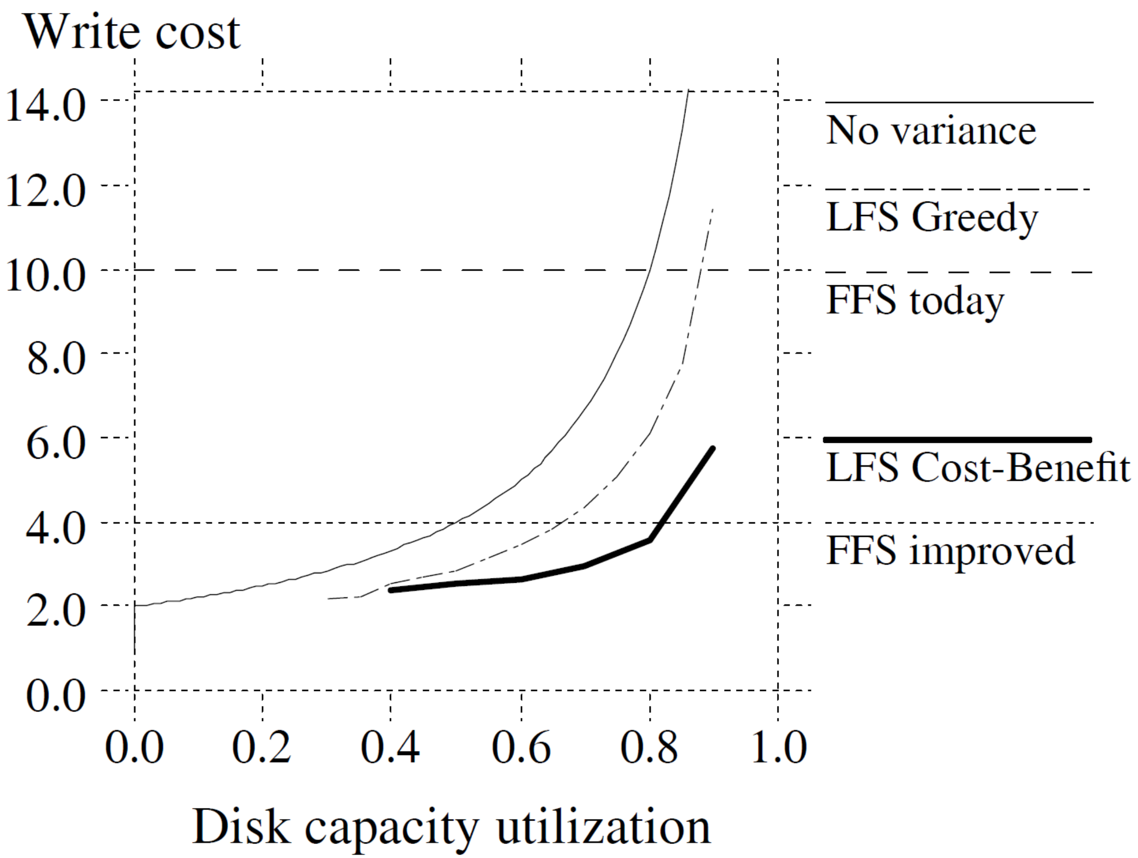
Cost-Benefit Policy: Goal

- Produces the ___________ distribution of segments
  - Cleans cold segments at about 75% utilization
  - But waits until hot segment reach about 15% utilization before cleaning them



Cost-Benefit Policy: Results

- LFS outperforms the best possible Unix FFS even at relatively high disk capacity utilization



Segment Usage Table

- Used to guide in the selection of segments to clean and to keep track of clean segments

- For each segment, segment usage table describes
  - The state of the segment (clean/dirty)
  - The number of live bytes in the segment
  - An estimate of the age of the youngest block in the segment (using the modified time for a file)

- The segment usage table is broken into blocks that can be cached in the file cache

Checkpoints

- Creating a checkpoint
  - Write out all modified information to the log
    - File data blocks, indirect blocks, inodes, blocks of the inode map, segment usage table
  - Write a checkpoint region to a special fixed position on disk
    - The addresses of all the blocks in _______________ and _______________
    - A pointer to the last segment written
    - The current time (in the last block of the checkpoint region)

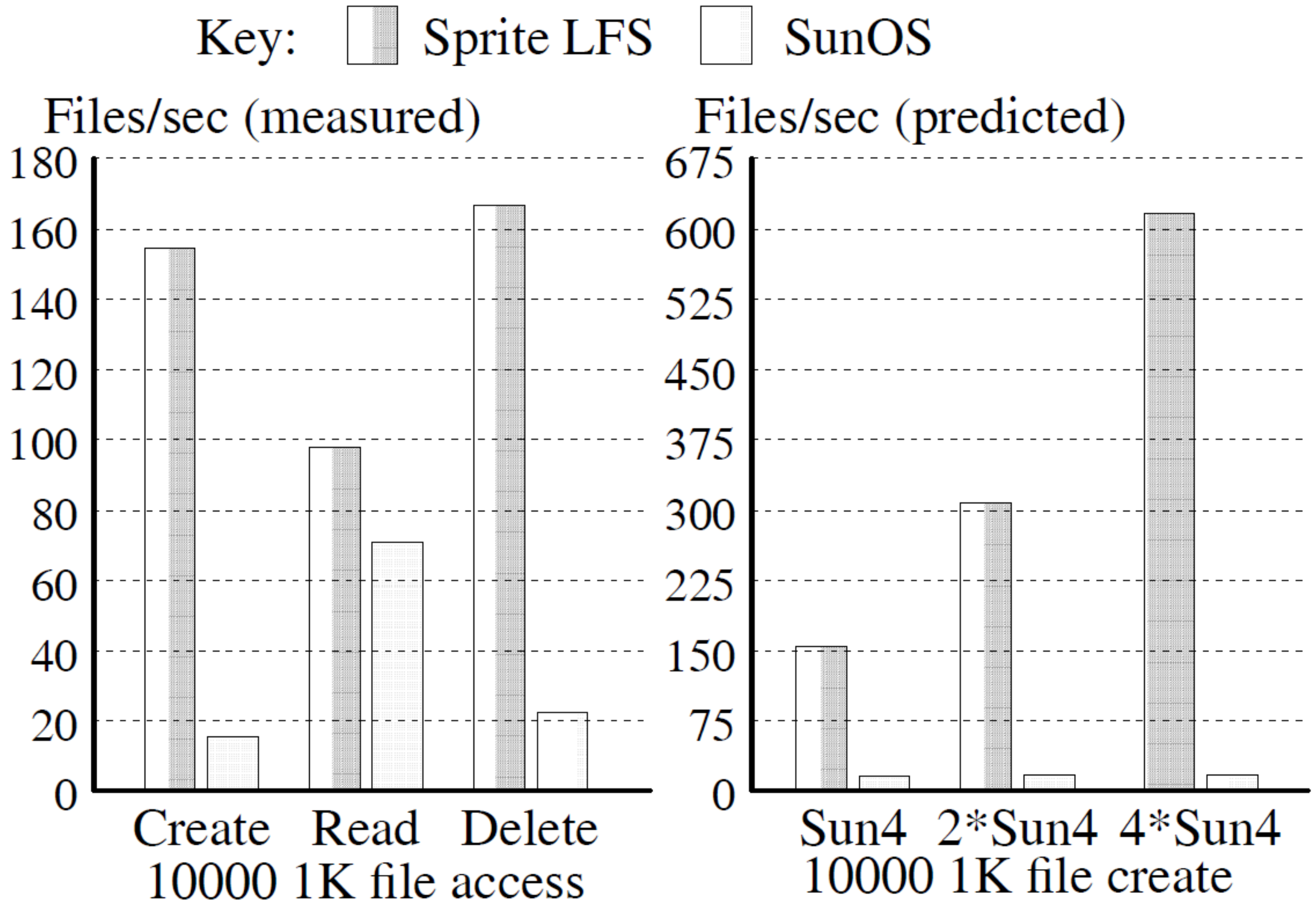
- Handling a crash during a checkpoint operation
  - Use two checkpoint regions
  - Alternate checkpoint operations between them
  - If the checkpoint fails, the time will not be updated

Roll-forward

- Recover inode map using segment summary blocks
  - The presence of a new inode → update the inode map
  - Data blocks without a new copy of the file's inode → ignored
- Adjust the utilizations in the ____________________
  - Maybe increased due to the live data left after roll-forward
  - Maybe decreased due to file deletions and overwrites
- Restore consistency between directory entries and inodes
  - ____________________ for each directory change

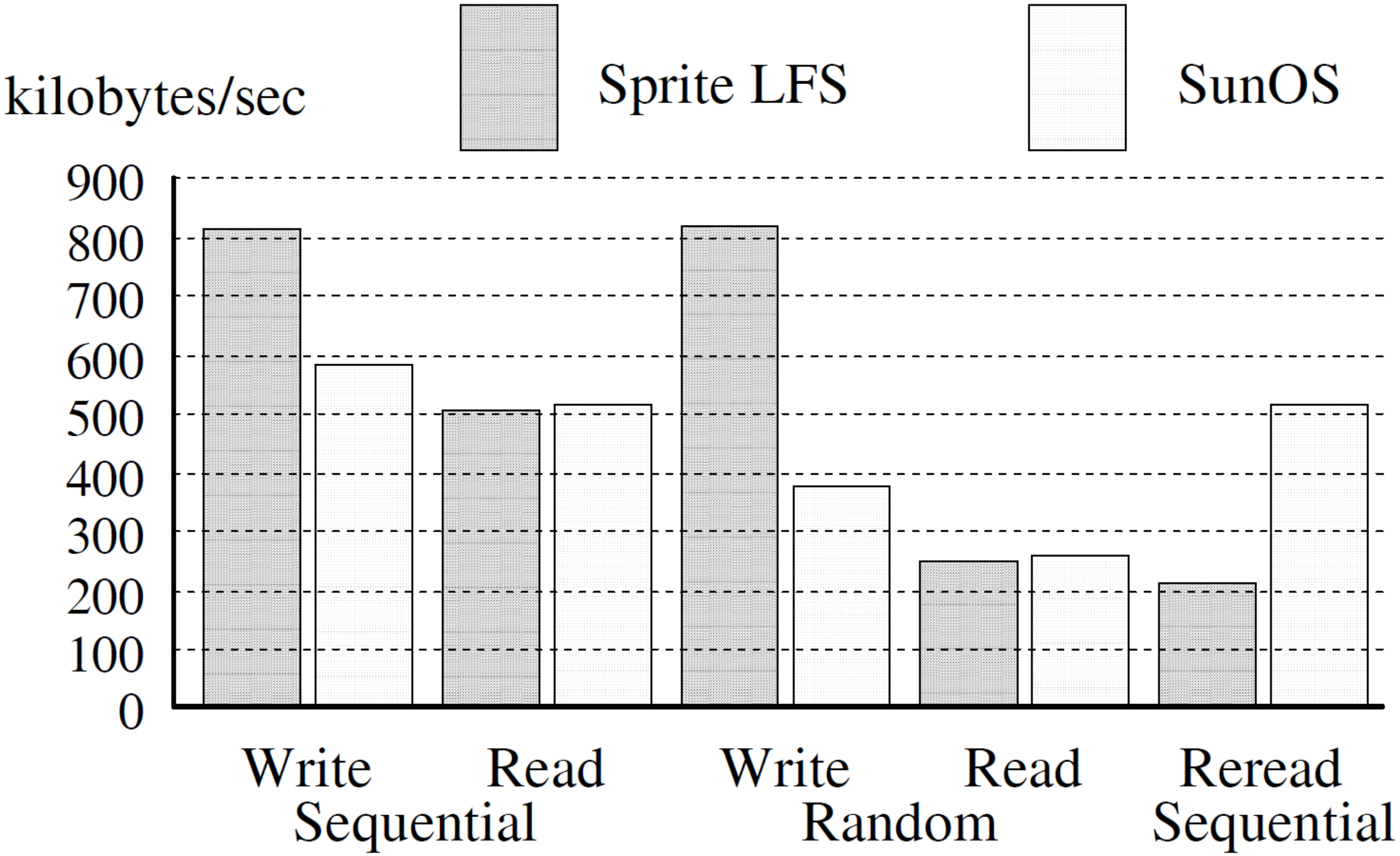
Small-file Performance

- 10x performance for the creation and deletion of small files



Large-file Performance

- Terrible sequential read after random write



Major Data Structures

Name	Purpose	Location
Inode	Locates blocks of file, holds protection bits, modify time, etc.	Log
Inode map	Locates position of inode in log, holds time of last access and version number	Log
Indirect block	Locates blocks of large files	Log
Segment summary	Identifies contents of segments (file number and offset for each block)	Log
Segment usage table	Counts live bytes still left in segments, stores last write time for data in segments	Log
Superblock	Holds static configuration information such as number of segments and segment size	Fixed
Checkpoint region	Locates blocks of inode map and segment usage table, identifies last checkpoint in log	Fixed
Directory change log	Records directory operations to maintain consistency of reference counts in inodes	Log

Summary: FFS vs. LFS

	FFS	LFS
Assign disk addresses	Block creation	Segment write
Allocate inodes	Fixed location	Appended to log
Max number of inodes	Statically determined	Statically determined or Grows dynamically
Map inode numbers to disk addresses	Static address	Lookup in inode map
Maintain free space	Bitmaps	Cleaner Segment usage table
Make file system state consistent	fsck	Checkpoints and Roll-forward

Seltzer vs. Ousterhout Debate

- Margo Seltzer implemented LFS on BSD and published a paper
  - M. Seltzer et al., "An Implementation of a Log-Structured File System for UNIX", *Proc. Winter 1993 USENIX Conference*.
- A Critique of Seltzer's 1993 USENIX Paper (Ousterhout)
- Seltzer published revised paper
  - M. Seltzer et al., "File System Logging Versus Clustering: A Performance Comparison," *Proc. Winter 1995 USENIX Conference*.
- A Critique of Seltzer's LFS Measurements (Ousterhout)
- A Response to Ousterhout's Critique of LFS Measurements (Seltzer)
- A Response to Seltzer's Response (Ousterhout)