15-712:
Advanced Operating Systems & Distributed Systems

The Design and Implementation of a Log-Structured File System

Prof. Phillip Gibbons

Spring 2023, Lecture 9

"The Design and Implementation of a Log-Structured File System"

Mendel Rosenblum, John K. Ousterhout 1992

- **Mendel Rosenblum** (UC Berkeley PhD, VMware, Stanford)
  - Co-founded VMware 1998, Chief Scientist until 2008
  - ACM Doctoral Dissertation Award, Mark Weiser Award, IEEE Info Storage Sys Award, ACM Software Sys Award
  - NAE, AAAS

- **John Ousterhout** (UC Berkeley, Industry, Stanford)
  - IEEE Info Storage Sys Award, ACM Software Sys Award, ACM Grace Murray Hopper Award
  - NAE
  - CMU PhD 1980

“The Design and Implementation of a Log-Structured File System”

Mendel Rosenblum, John K. Ousterhout 1992

SigOps HoF citation (2012):

The paper introduces log-structured file storage, where data is written sequentially to a log and continuously de-fragmented. The underlying ideas have influenced many modern file and storage systems like NetApp’s WAFL file systems, Facebook’s picture store, aspects of Google’s BigTable, and the Flash translation layers found in SSDs.

1992 Trends Prediction

- **Assumptions:**
  - CPU speeds increasing faster than disk speeds
  - Files are cached in main memory
  - Increasing memory sizes will make caches effective at satisfying read requests and buffering write requests
  - Crashes are infrequent & it's ok to lose seconds/minutes of work in each crash (use nonvolatile RAM for write buffer, if needed)

- **Thus, disk traffic will become dominated by writes**

- **Goal: "Design for file systems of the 1990s"**

Discussion: How accurate were these predictions?

Problems with Existing File Systems

- **Spread info around the disk, causing many small accesses**
  - Berkeley UNIX FFS: 5 seeks + I/O to create a new file
  - Small files achieve < 5% utilization of disk's raw bandwidth
- **Write synchronously (esp. directories & inodes)**
  - Slow

Log-Structured File System

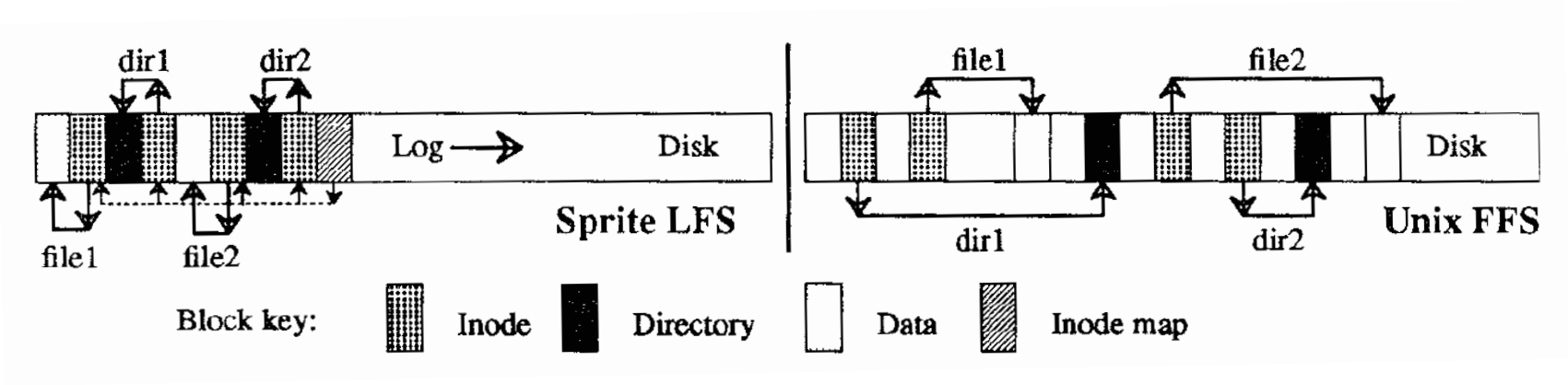
- **Log is the only structure on disk, written sequentially**
  - Avoids large overheads of disk seeks (important for short files)
  - Buffers writes and then writes to disk in a single disk write op
  - Contains indexing info for efficient reading

- **Most difficult challenge?**

  Need large extents of free space available for writing new data
  - Solution: Segments with a Segment Cleaner process

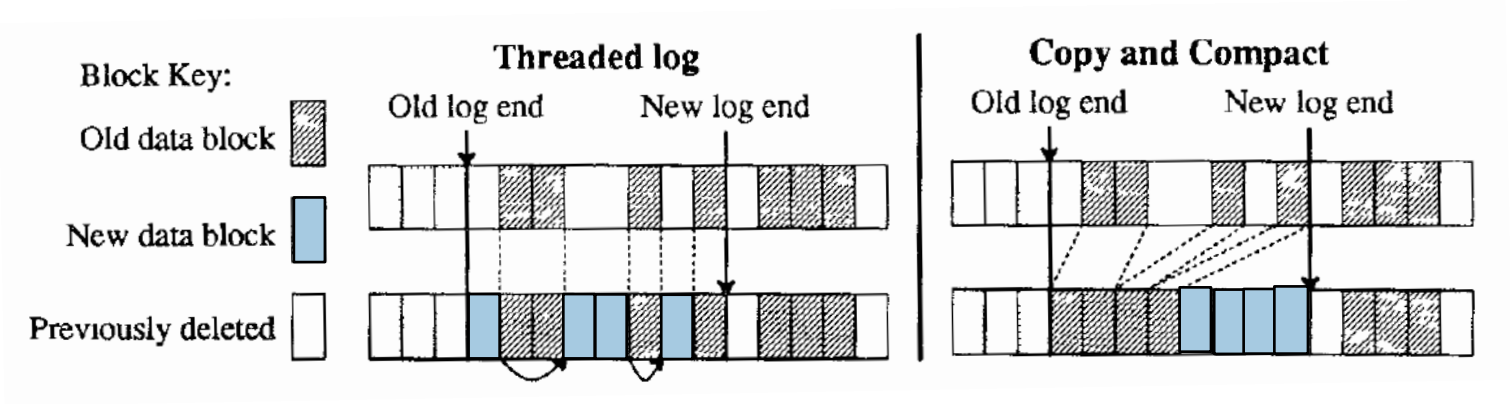
File Location

- **inode map: maintains current location in log of each inode**
  - Include file version number
  - Keep active portions of inode map in memory



Free Space Management: Segments

- **Threading free extents would cause severe fragmentation**
- **Copying is costly for long-lived files**



- **Segments**
  - Any given segment is written sequentially from beginning to end
  - Log is threaded on a segment-by-segment basis
  - Sprite LFS uses segments of size=512KB or 1 MB

Segment Cleaning Mechanism

- **Cleaning**
  - Read some segments into memory
  - Copy live data to a smaller number of clean segments
  - Reclaim original segments

- **Segment summary block: identifies each piece of info**
  - File number & block number for File data blocks
  - Uid = version number & inode number

- **Determine liveness by checking if file's inode or indirect block still refers to this block; otherwise block is dead**
  - No free-block list/bitmap, simplifying crash recovery

Crash Recovery

- **Checkpoints (every 30 secs – "probably much too short")**
  - Write all modified info to the log
  - Write a **checkpoint region** to fixed position on disk: addrs of all blocks in inode map & segment usage table, ptr to last segment written, current time

- **Roll-Forward from log**
  - New inode: update inode map
  - New data blocks w/o new inode: ignore
  - Adjust stats in segment usage table
  - LFS writes directory changes to log before corresponding directory block/inode (no directory changes during checkpoints)

Discussion: Summary Question #1

- **State the 3 most important things the paper says.** These could be some combination of their motivations, observations, interesting parts of the design, or clever parts of their implementation.

Segment Cleaning Policies

- **When?**
  - Run when # of clean segs < few 10s
- **On how many?**
  - Few 10s at a time until # clean > 50-100
- **Which segments?**
- **Grouping of live data**

} **Primary factors determining LFS performance**

- **Write cost: avg time disk is busy per byte of new data written, including all cleaning overhead**
  - E.g., write cost=10 means 10% of disk BW used for writing new data, 90% used for seeks, rotational latency, cleaning
  - With large segments, can ignore seeks & rotational latency

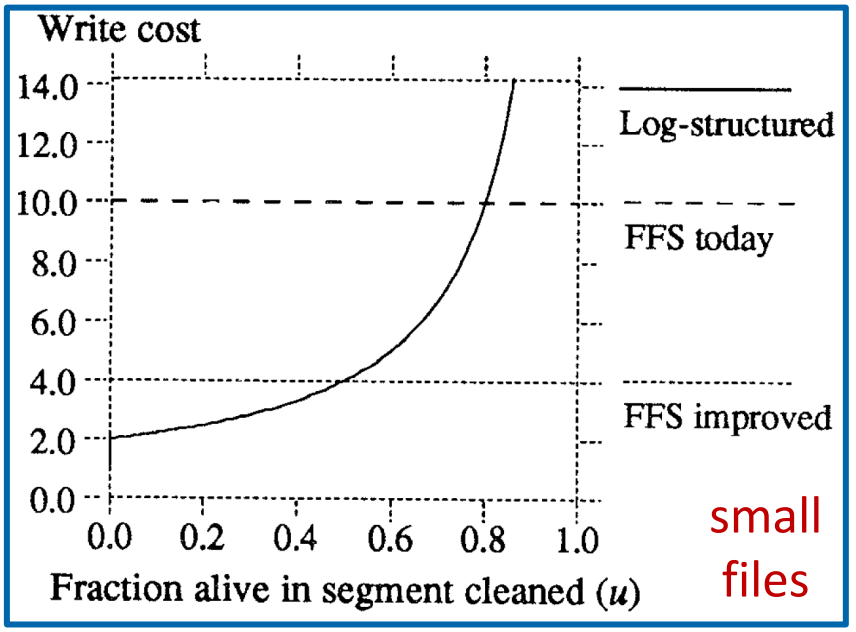
Analyzing Write Cost

- **Read N segs, write out $N * \mu$ segments of live data**
  - μ is utilization of the segments, $0 \leq \mu < 1$

$$\text{write cost} = \frac{\text{total bytes read and written}}{\text{new data written}}$$

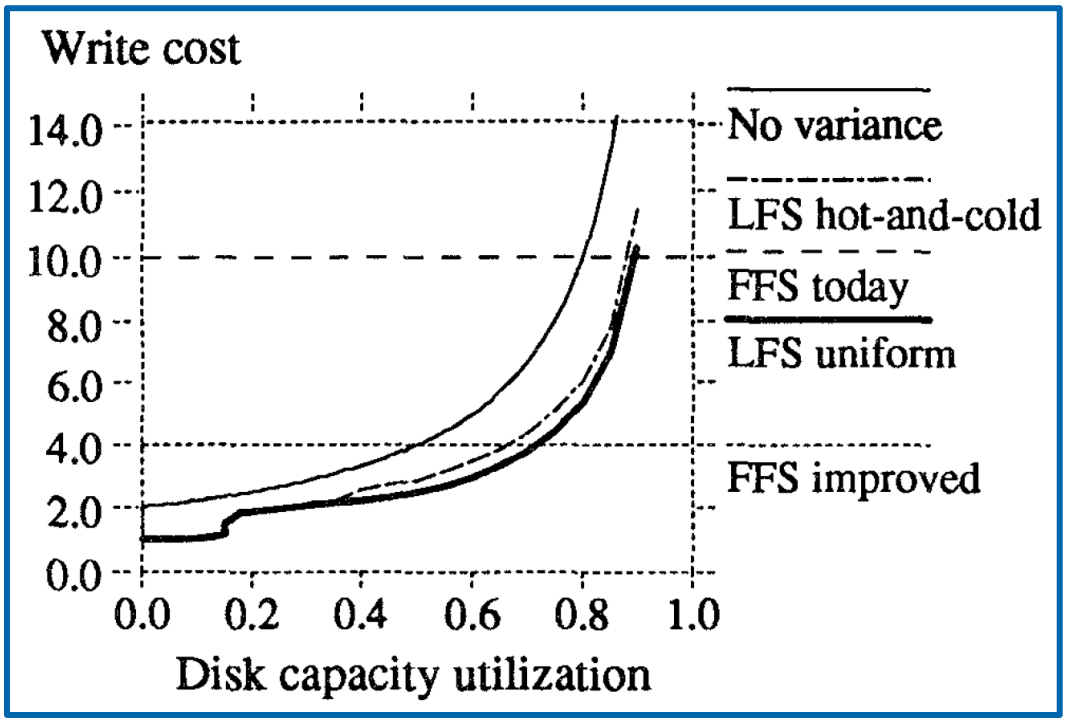
$$= \frac{\text{read segs} + \text{write live} + \text{write new}}{\text{new data written}}$$

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



- **Trade-off**
  - LFS performance "can be improved by reducing the overall utilization of the disk space"

Impact of Locality: A Surprise!



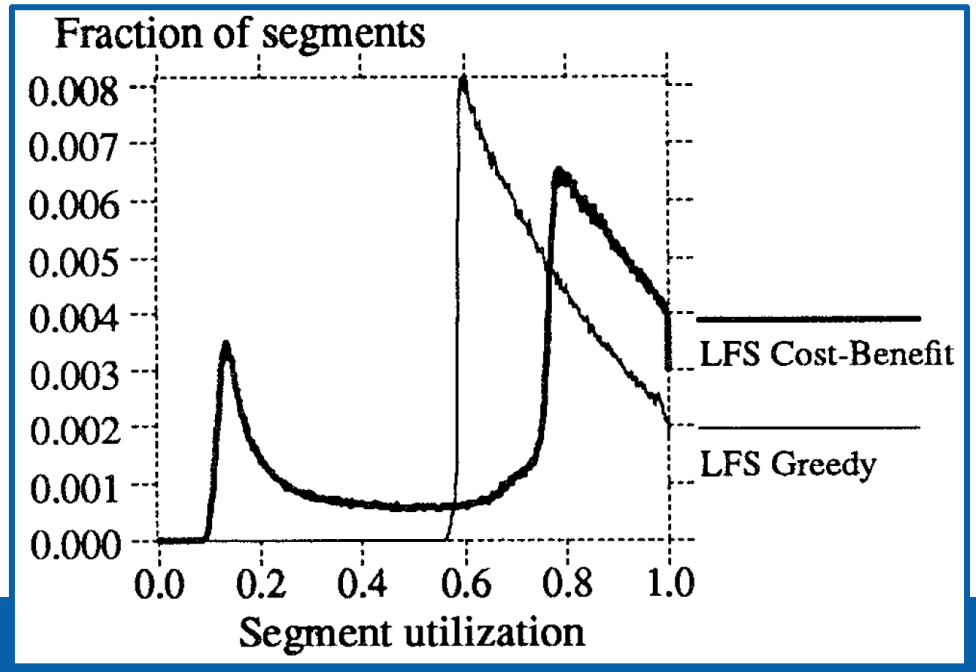
- **Simulator, 4KB files, select file to overwrite with new data**
  - Uniform: File selected uniformly at random
  - Hot-and-cold: 10% of files that are selected 90% of the time
  - Clean least utilized segment; Hot-and-cold writes out by age
- **Surprise: Locality & "better" grouping is worse! Why?**

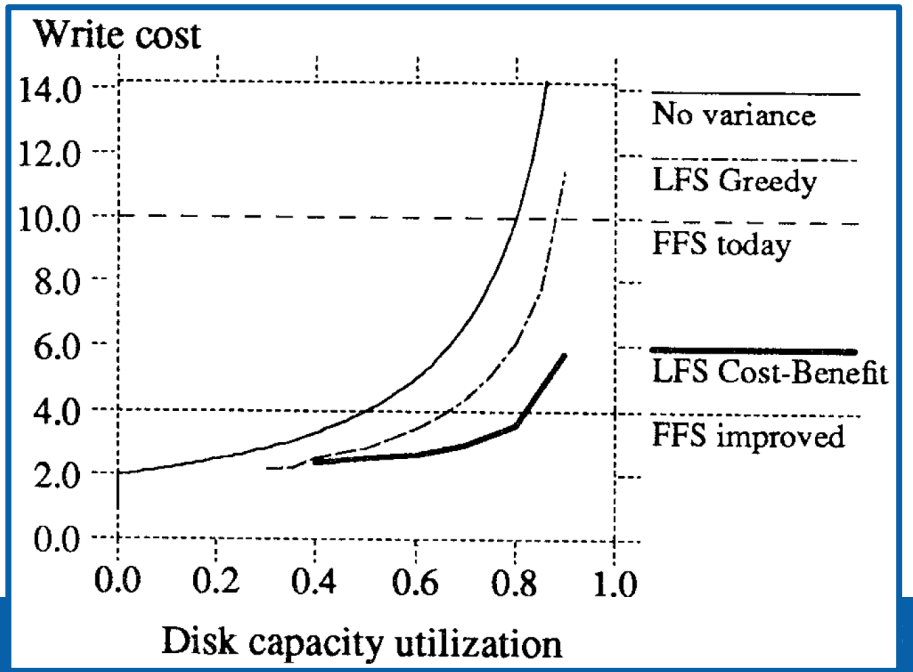
Cost-Benefit Policy

- **Problem: Cold segments tie up many free blocks for long time**
- **Solution: Factor in amount of time the space is likely to stay free**
  - Estimate as time since any block in segment was modified

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

Hot-and-cold distribution



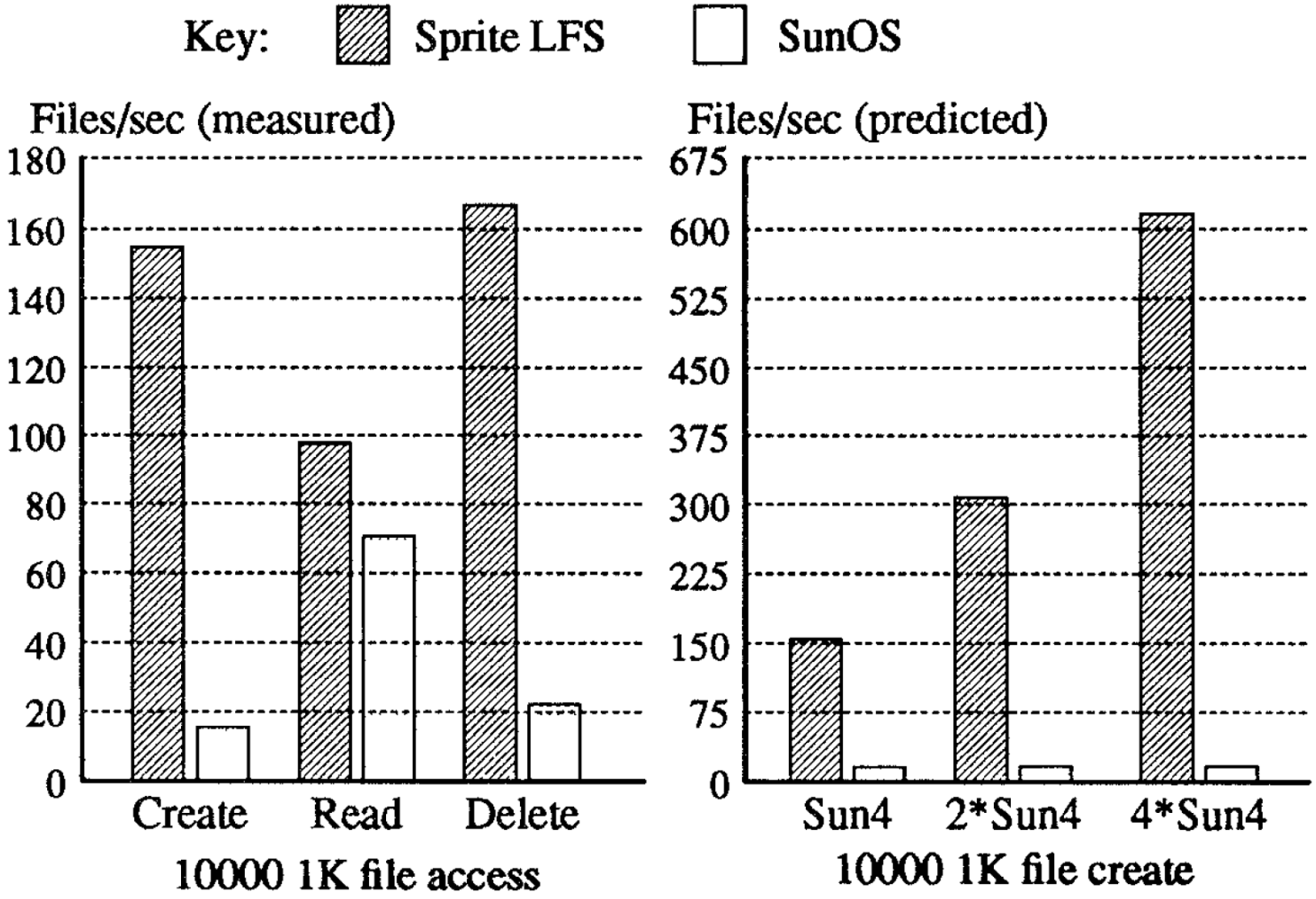


Discussion: Summary Question #2

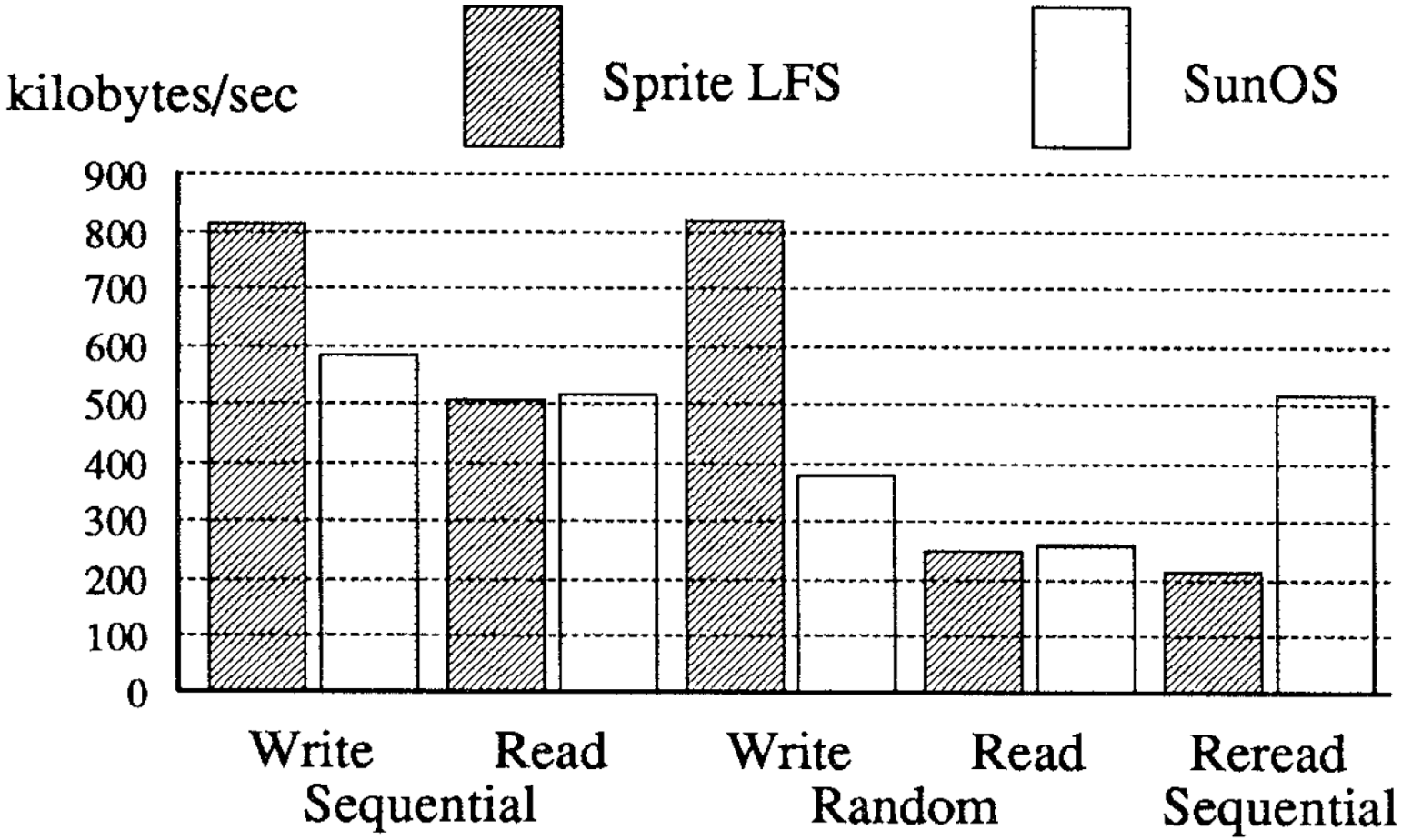
- **Describe the paper's single most glaring deficiency.** Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

Sprite LFS vs. SunOS (Unix FFS) – Best Case

- Sprite 4KB blocks, 1MB segments (no cleaning); SunOS 8KB blocks



100MB Files, Different Access Patterns



- Logical locality vs. Temporal locality

Cleaning Overheads

Table II. Segment Cleaning Statistics and Write Costs for Production File Systems.

Write cost in Sprite LFS file systems								
File system	Disk Size	Avg File Size	Avg Write Traffic	In Use	Segments		u	Write Cost
					Cleaned	Empty	Avg	
/user6	1280 MB	23.5 KB	3.2 MB/hour	75%	10732	69%	.133	1.4
/pcs	990 MB	10.5 KB	2.1 MB/hour	63%	22689	52%	.137	1.6
/src/kernel	1280 MB	37.5 KB	4.2 MB/hour	72%	16975	83%	.122	1.2
/tmp	264 MB	28.9 KB	1.7 MB/hour	11%	2871	78%	.130	1.3
/swap2	309 MB	68.1 KB	13.3 MB/hour	65%	4701	66%	.535	1.6

- **Sprite LFS is much better than simulation study predicts**
  - Large files that are written/deleted as a whole: greater locality
  - Cold segments much colder
  - Thus, highly bimodal segment utilization (empty & full)

Bottom Line

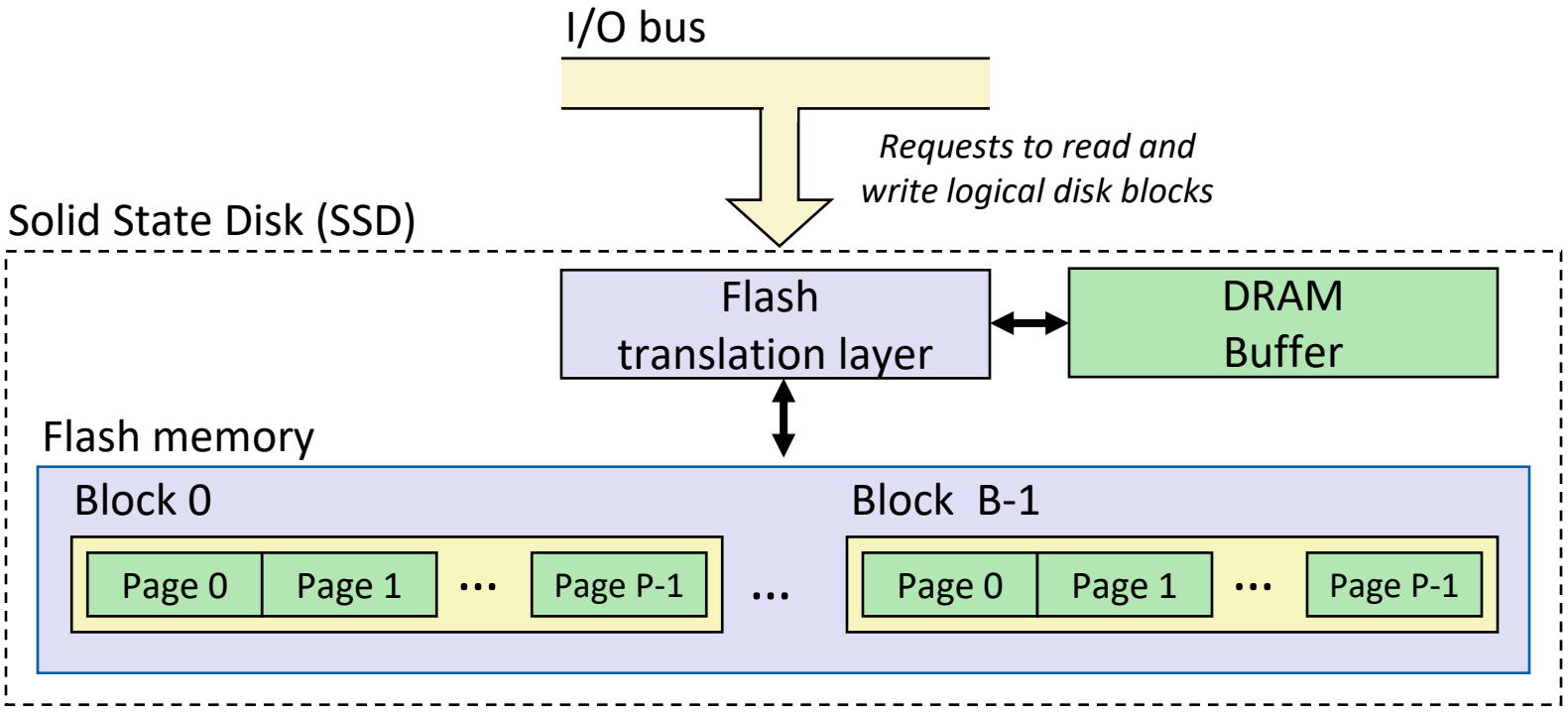
- **LFS can use disks an order of magnitude more efficiently**
  - Can use 70% of the disk bandwidth vs. 5-10% for Unix FFS

"This should make it possible to take advantage of several more generations of faster processors before I/O limitations once again threaten the scalability of computer systems."

How LFS Differs

- **From Garbage Collectors**
  - Sequential accesses necessary for high FS performance
  - Blocks belong to one file at a time: easier to identify garbage
- **From Databases**
  - Log is final repository for data, so must write entire blocks not deltas (compaction would hurt read performance)
  - Clean to reclaim log space vs. Delete after apply to data area
  - Simpler crash recovery since don't need to redo data updates
- **From FTLs on SSDs?**

Recall: Solid State Disks (SSDs)



- **Pages: 512KB to 4KB, Blocks: 32 to 128 pages**
- **Data read/written in units of pages.**
- **Page can be written only after its block has been erased.**
- **A block wears out after about 100,000 repeated writes.**

Discussion: Summary Question #3

- **Describe what conclusion you draw from the paper as to how to build systems in the future.** Most of the assigned papers are significant to the systems community and have had some lasting impact on the area.

Friday's Papers

File Systems and Disks (IV)

"A Case for Redundant Arrays of Inexpensive Disks (RAID)"

David A. Patterson, Garth Gibson, Randy H. Katz 1988

Optional Further Reading:

"Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?"

Bianca Schroeder, Garth A. Gibson 2007