STORAGE SYSTEMS: FILE SYSTEMS

CS6410

Hakim Weatherspoon

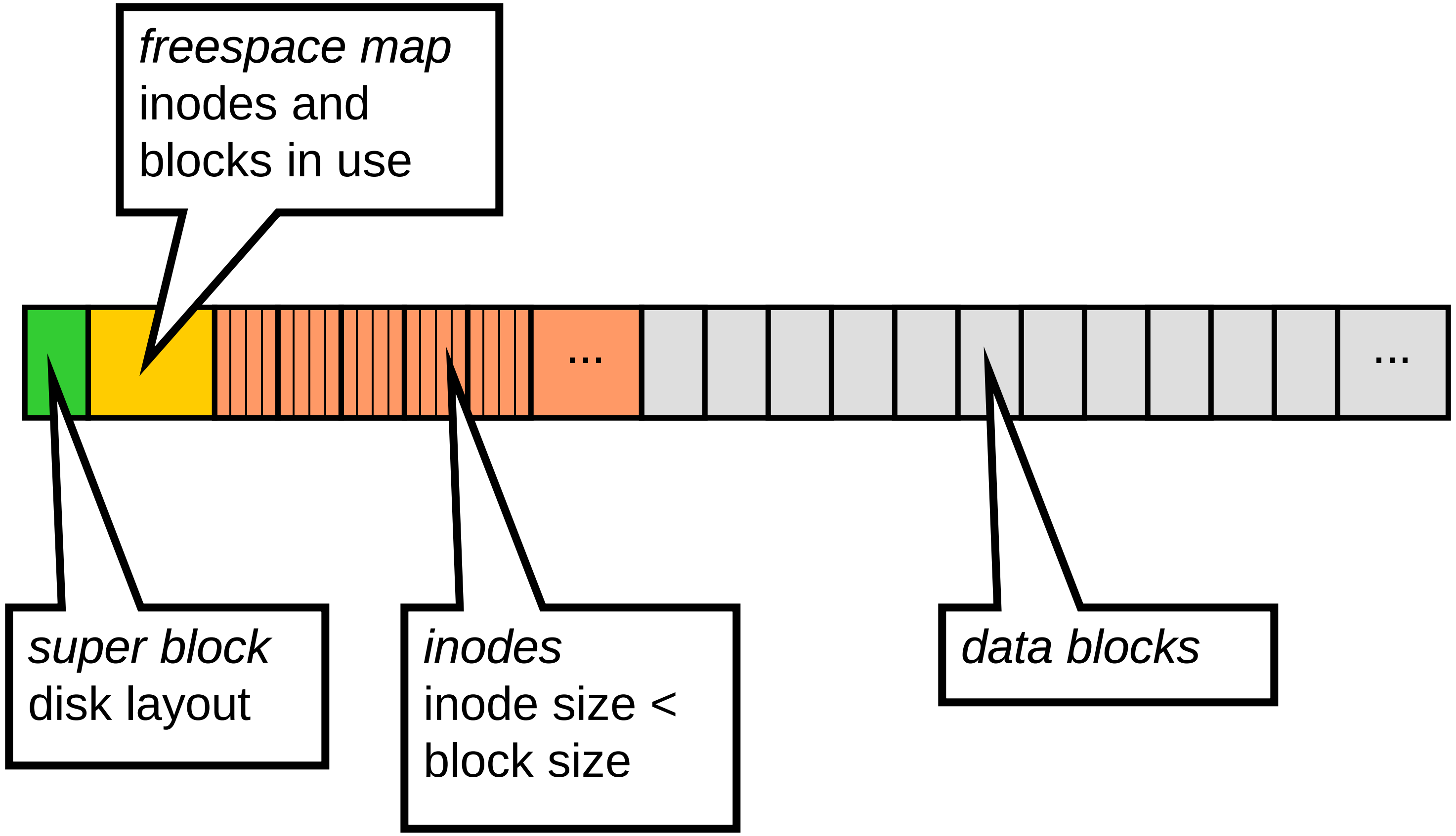
Plan for today

- Discuss Papers:
  - The Design and Implementation of a Log-Structured File System (LFS), Mendel Rosenblum and Ousterhout. SOSP, 1991.
  - Towards weakly consistent local storage systems (Yogurt), Ji-Yong Shin, Mahesh Balakrishnan, Tudor Marian, Jakub Szefer, Hakim Weatherspoon. SoCC 2016
- Historical Context:
  - UNIX File System (UFS)
  - UNIX Berkeley Fast File System (FFS)

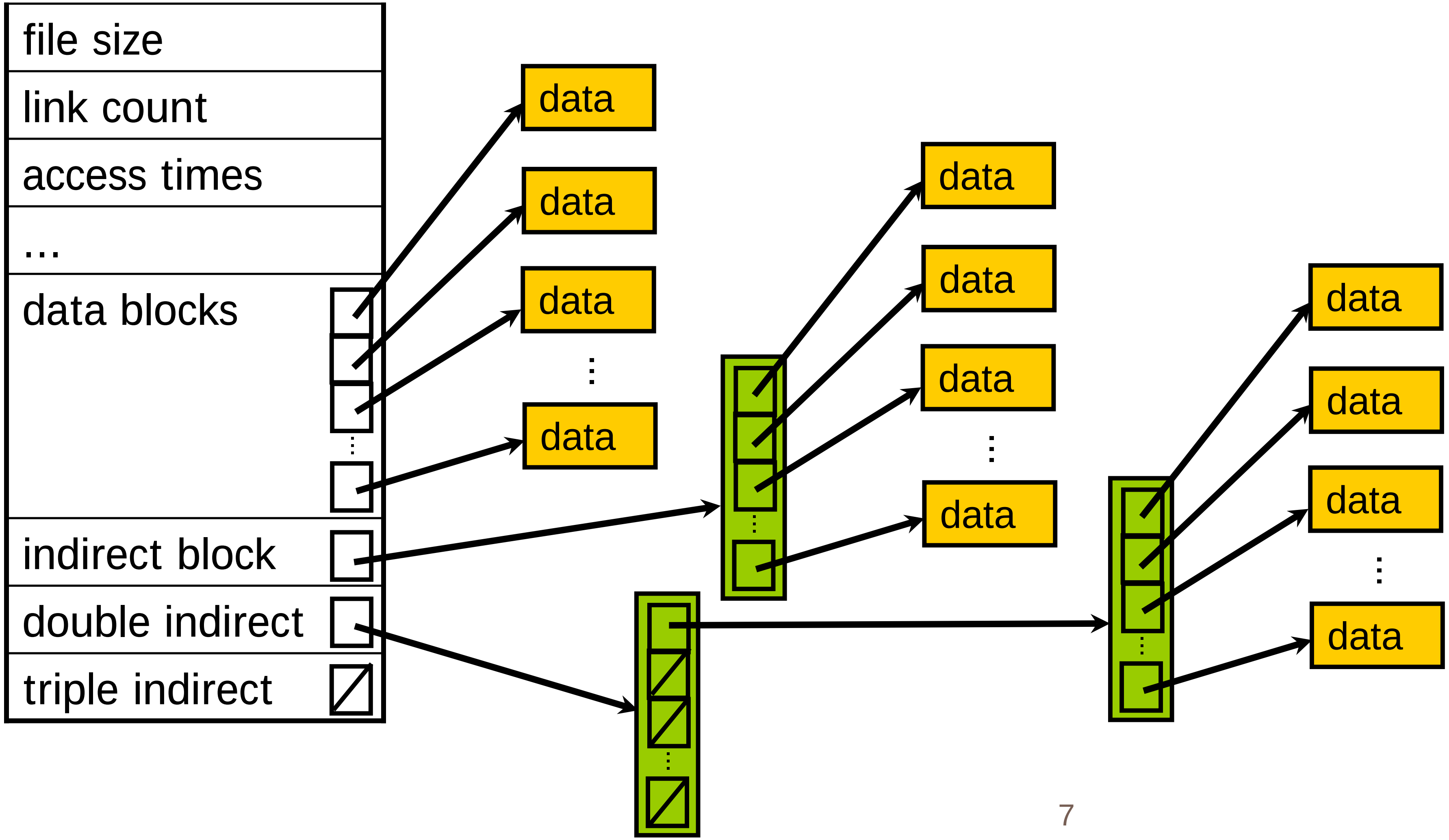
Background: Unix Fast File Sys

- Original UNIX File System (UFS)
  - Simple, elegant, but ***slow***
  - 20 KB/sec/arm; ~2% of 1982 disk bandwidth
- Problems
  - blocks too small
  - consecutive blocks of files not close together
    (random placement for mature file system)
  - i-nodes far from data
    (all i-nodes at the beginning of the disk, all data afterward)
  - i-nodes of directory not close together
  - no read-ahead

File system on disk



File representation



The Unix Berkeley Fast File System

- Berkeley Unix (4.2BSD)

- 4kB and 8kB blocks
  - (why not larger?)
  - Large blocks and small fragments
- Reduces seek times by better placement of file blocks
  - i-nodes correspond to files
  - Disk divided into cylinders
    - contains superblock, i-nodes, bitmap of free blocks, summary info
  - Inodes and data blocks grouped together
  - Fragmentation can still affect performance

FFS implementation

- Most operations do multiple disk writes
  - File write: update block, inode modify time
  - Create: write freespace map, write inode, write directory entry
- Write-back cache improves performance
  - Benefits due to high write locality
  - Disk writes must be a whole block
  - **Syncer process flushes writes every 30s**

FFS Goals

- keep dir in cylinder group, spread out different dir’s
- Allocate runs of blocks within a cylinder group, every once in a while switch to a new cylinder group (jump at 1MB).
- layout policy: global and local
  - global policy allocates files & directories to cylinder groups. Picks “optimal” next block for block allocation.
  - local allocation routines handle specific block requests. Select from a sequence of alternative if need to.

FFS locality

- don't let disk fill up in any one area
- paradox: for locality, spread unrelated things far apart
- note: FFS got 175KB/sec because free list contained sequential blocks (it did generate locality), but an old UFS had randomly ordered blocks and only got 30 KB/sec

FFS Results

- 20-40% of disk bandwidth for large reads/writes
- 10-20x original UNIX speeds
- Size: 3800 lines of code vs. 2700 in old system
- 10% of total disk space unusable

FFS Enhancements

- long file names (14 -> 255)
- advisory file locks (shared or exclusive)
  - process id of holder stored with lock => can reclaim the lock if process is no longer around
- symbolic links (contrast to hard links)
- atomic rename capability
  - (the only atomic read-modify-write operation, before this there was none)
- Disk Quotas
- Overallocation
  - More likely to get sequential blocks; use later if not

FFS crash recovery

- Asynchronous writes are lost in a crash
  - **Fsync** system call flushes dirty data
  - Incomplete metadata operations can cause disk corruption (order is important)
- FFS metadata writes are synchronous
  - Large potential decrease in performance
  - Some OSes cut corners

After the crash

- **Fsck** file system consistency check
  - Reconstructs freespace maps
  - Checks inode link counts, file sizes
- Very time consuming
  - Has to scan all directories and inodes

Perspective

- Features
  - parameterize FS implementation for the HW in use
  - measurement-driven design decisions
  - locality “wins”
- Flaws
  - measuremenets derived from a single installation.
  - ignored technology trends
- Lessons
  - Do not ignore underlying HW characteristics
- Contrasting research approach
  - Improve status quo vs design something new

The Design and Impl of a Log-structured File System

Mendel Rosenblum and John K. Ousterhout

- Mendel Rosenblum
  - Designed LFS, PhD from Berkeley
  - ACM Disseration Award Winner
  - Professor at Stanford, designed SimOS
  - Founder of VM Ware

- John Ousterhout
  - Professor at Berkeley 1980-1994
  - Created Tcl scripting language and TK platform
  - Research group designed Sprite OS and LFS
  - Now professor at Stanford after 14 years in industry

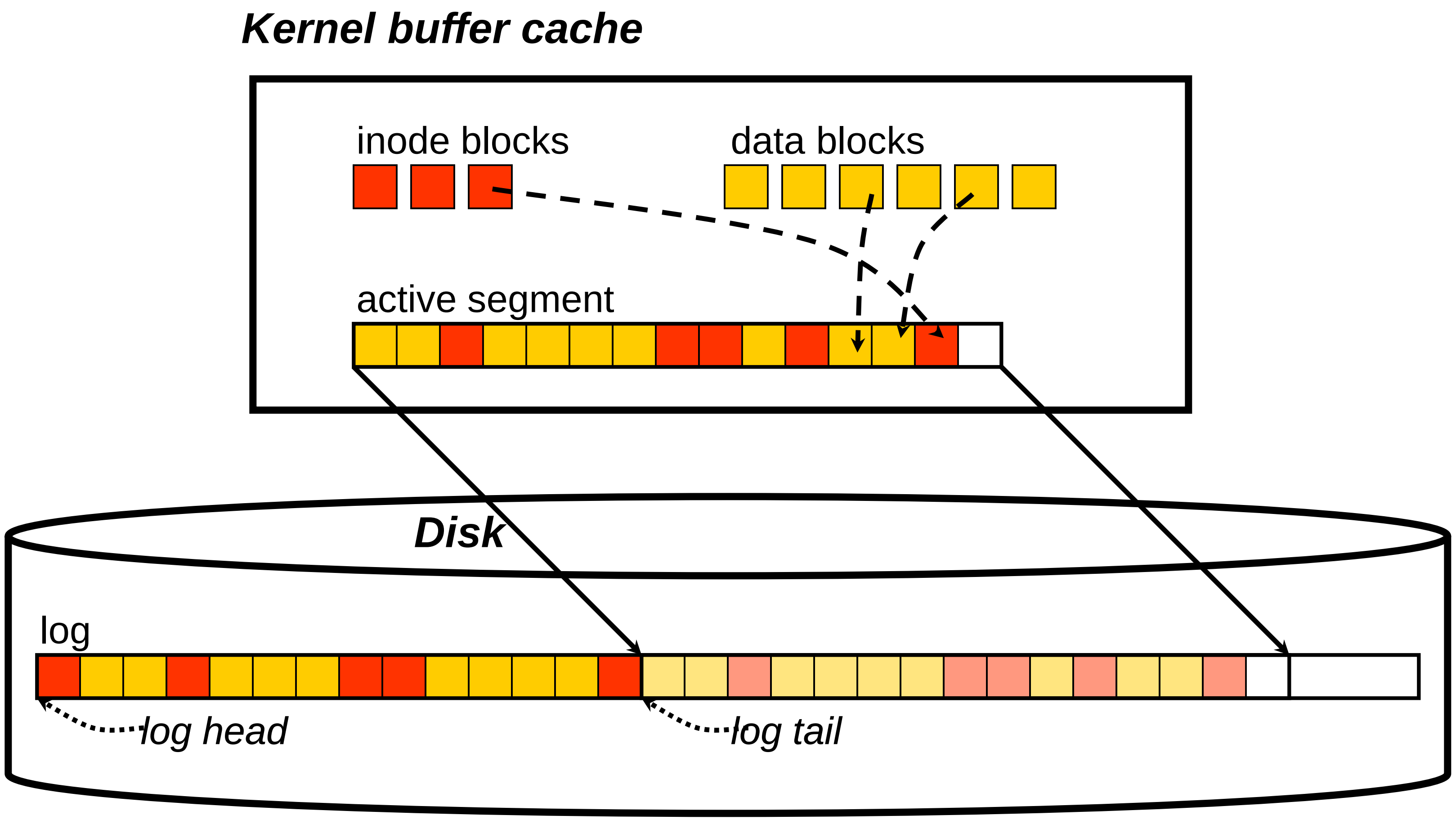
The Log-Structured File System

- Technology Trends
  - I/O becoming more and more of a bottleneck
  - CPU speed increases faster than disk speed
  - Big Memories: Caching improves read performance
  - Most disk traffic are writes
- Little improvement in write performance
  - Synchronous writes to metadata
  - Metadata access dominates for small files
  - e.g. Five seeks and I/Os to create a file
    - file i-node (create), file data, directory entry, file i-node (finalize), directory i-node (modification time).

LFS in a nutshell

- Boost write throughput by writing all changes to disk contiguously
  - Disk as an array of blocks, append at end
  - Write data, indirect blocks, inodes together
  - No need for a free block map
- Writes are written in *segments*
  - ~1MB of continuous disk blocks
  - Accumulated in cache and flushed at once
- Data layout on disk
  - "temporal locality" (good for writing)
    rather than "logical locality" (good for reading).
  - Why is this a better?
    - Because caching helps reads but not writes!

Log operation



LFS design

- Increases write throughput from 5-10% of disk to 70%
  - Removes synchronous writes
  - Reduces long seeks
- Improves over FFS
  - "Not more complicated"
  - Outperforms FFS except for one case

LFS challenges

- Log retrieval on cache misses
  - Locating inodes
- What happens when end of disk is reached?

Locating inodes

- Positions of data blocks and inodes change on each write
  - Write out inode, indirect blocks too!
- Maintain an inode map
  - Compact enough to fit in main memory
  - Written to disk periodically at *checkpoints*
    - Checkpoints (map of inode map) have special location on disk
    - Used during crash recovery

Cleaning the log: "Achilles Heel"

- Log is infinite, but disk is finite
  - Reuse the old parts of the log
- Clean old segments to recover space
  - Writes to disk create holes
  - Segments ranked by "liveness", age
  - Segment cleaner "runs in background"
- Group slowly-changing blocks together
  - Copy to new segment or "thread" into old

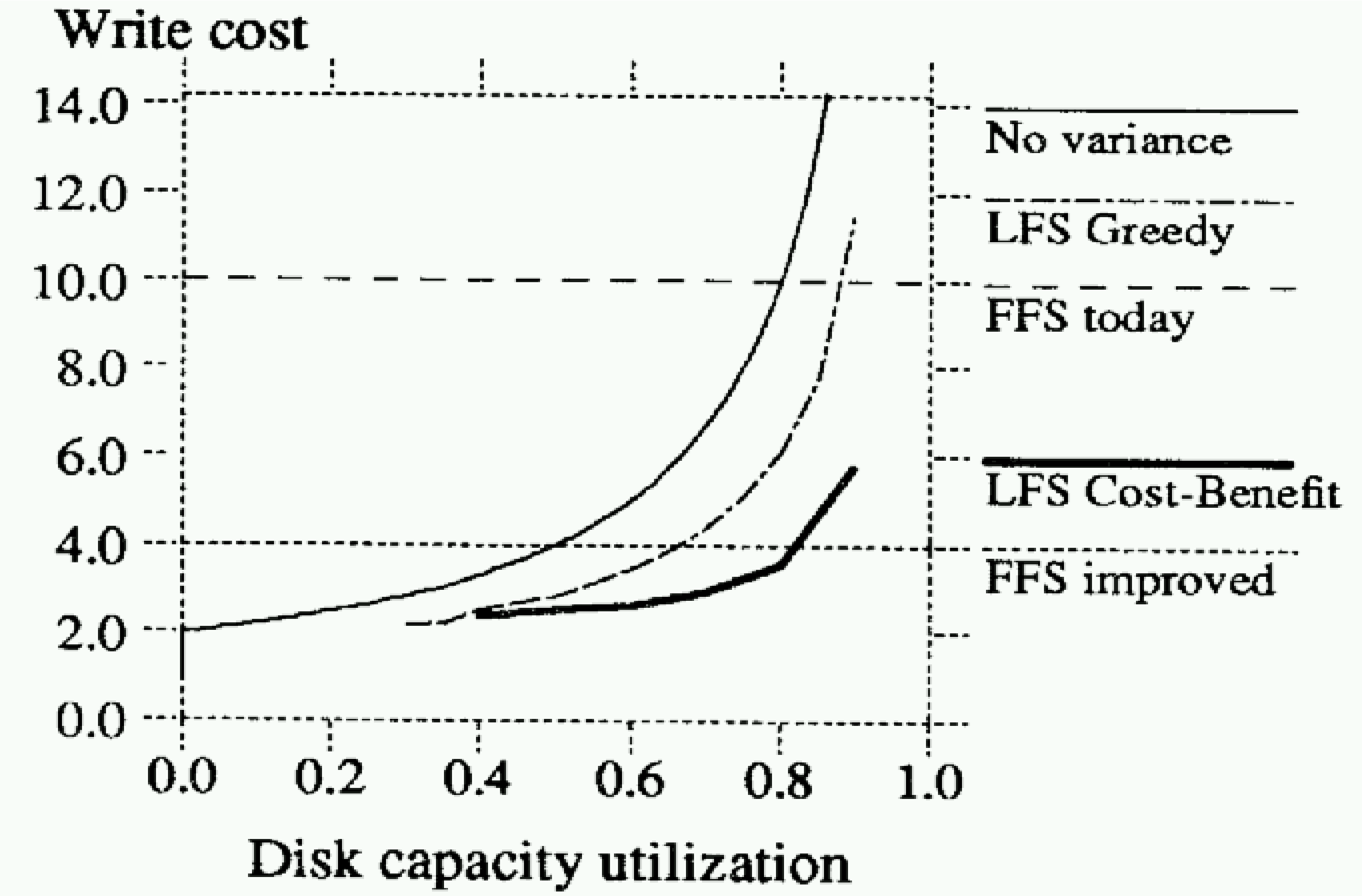
Cleaning policies

- Simulations to determine best policy
  - Greedy: clean based on low utilization
  - Cost-benefit: use age (time of last write)

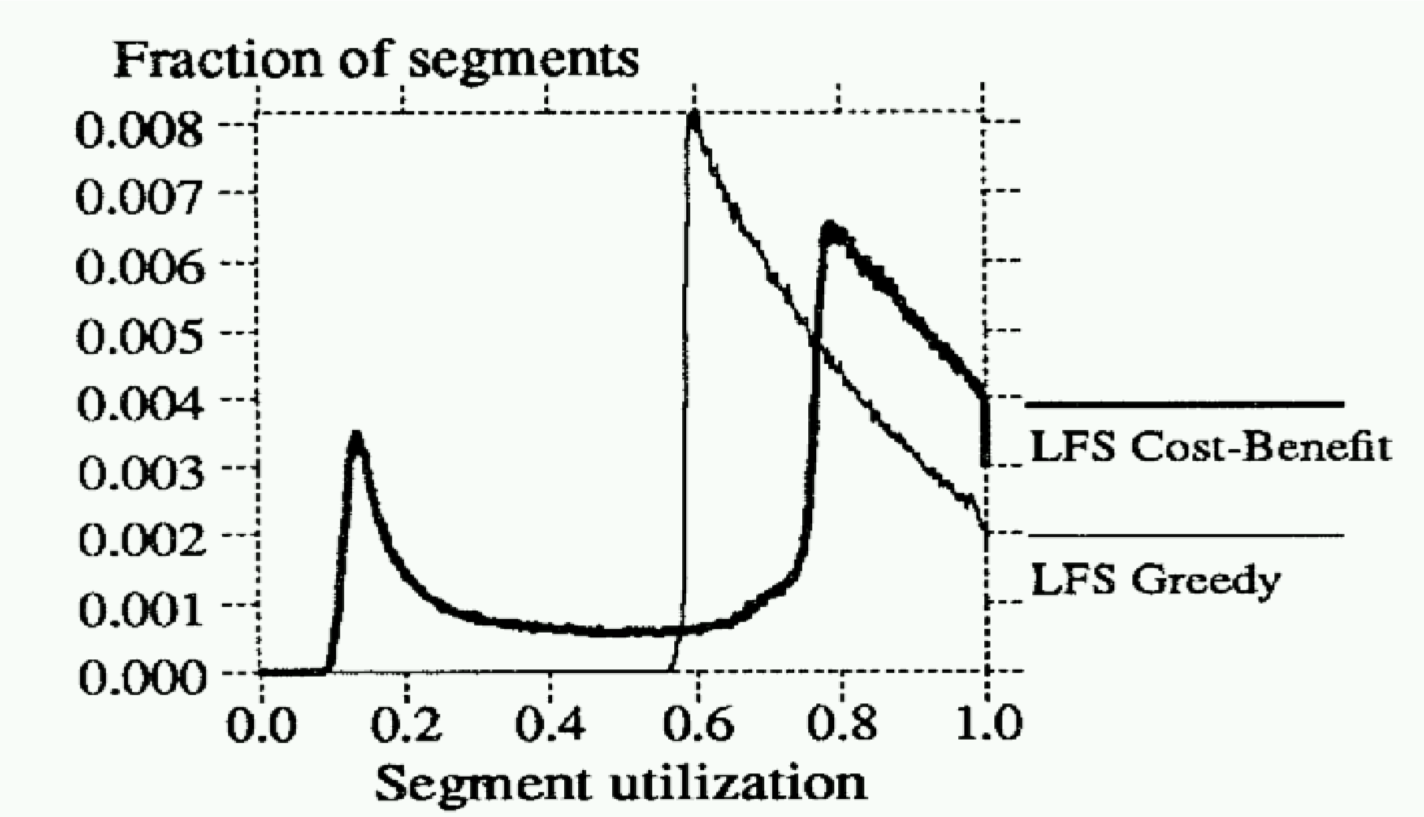
$$\frac{\texttt{benefit}}{\texttt{cost}} = \frac{\texttt{(free space generated)*(age of segment)}}{\texttt{cost}}$$

- Measure *write cost*
  - Time disk is busy for each byte written
  - Write cost 1.0 = no cleaning

Greedy versus Cost-benefit



Cost-benefit segment utilization



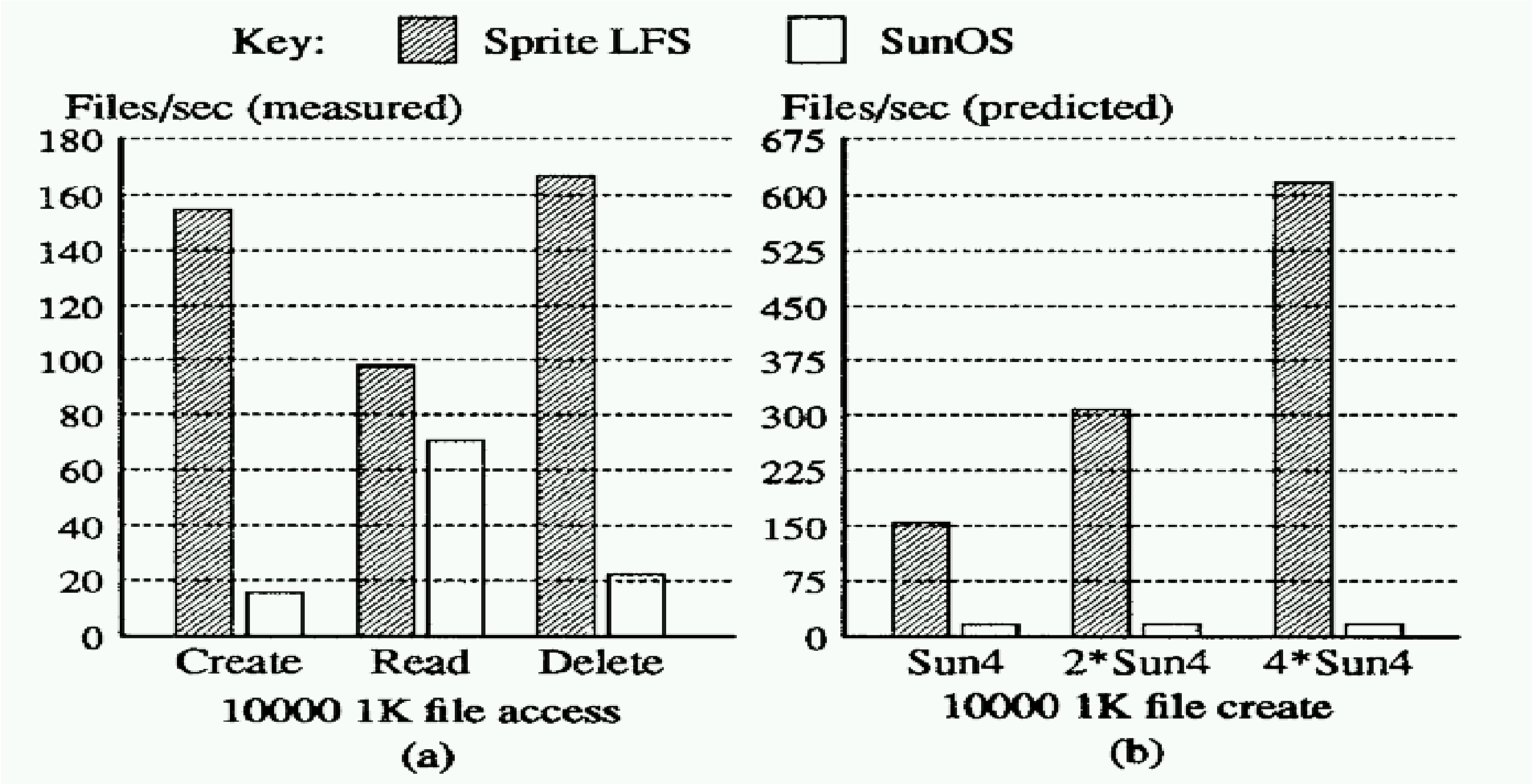
LFS crash recovery

- Log and checkpointing
  - Limited crash vulnerability
  - At checkpoint flush active segment, inode map
- No `fsck` required

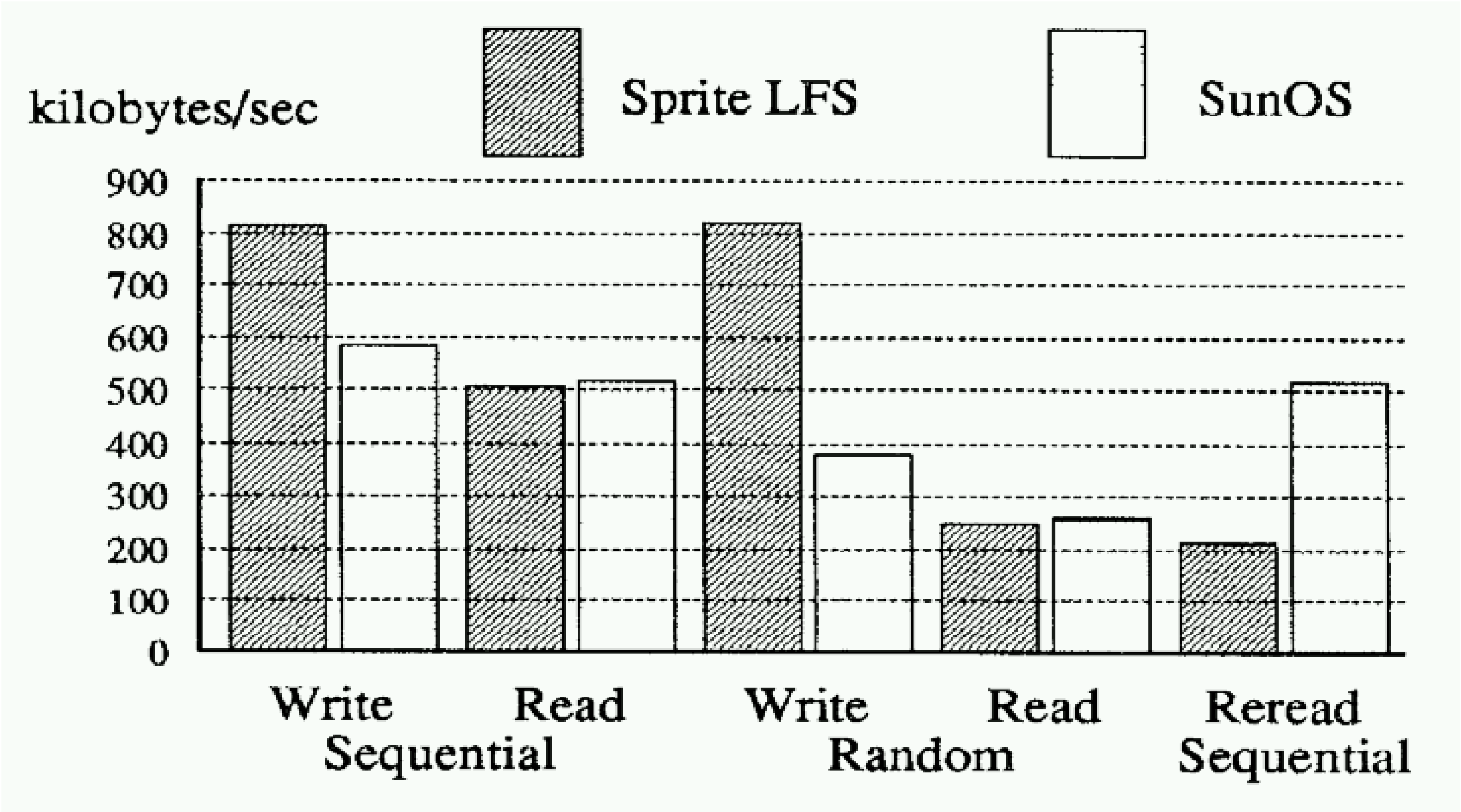
LFS performance

- Cleaning behaviour better than simulated predictions
- Performance compared to SunOS FFS
  - Create-read-delete 10000 1k files
  - Write 100-MB file sequentially, read back sequentially and randomly

Small-file performance



Large-file performance



Perspective

- Features
  - CPU speed increasing faster than disk => I/O is bottleneck
  - Write FS to log and treat log as truth; use cache for speed
  - Problem
    - Find/create long runs of (contiguous) disk space to write log
  - Solution
    - clean live data from segments,
    - picking segments to clean based on a cost/benefit function
- Flaws
  - Intra-file Fragmentation: LFS assumes entire files get written
  - If small files "get bigger", how would LFS compare to UNIX?
- Lesson
  - Assumptions about primary and secondary in a design
  - LFS made log the truth instead of just a recovery aid

Towards Weakly Consistent Local Storage Systems

Ji Yong Shin, Mahesh Balakrishnan, Tudar Marian, Jakub Szefer, Hakim Weatherspoon

- Ji Yong Shin
  - Student at Cornell, post-doc Yale
- Mahesh Balakrishnan
  - Student at Cornell, Researcher at Microsoft
  - Professor at Yale
- Tudor Marian
  - Student at Cornell, Researcher at Google
- Jakub Szefer
  - Professor at Yale
- Hakim Weatherspoon
  - Student at Berkeley, Professor at Cornell

Motivation

Devices	Throughput	Latency	Cost / GB
Registers	-	1 cycle	-
Caches	-	2-10ns	-
DRAM	10s of GB/s	100-200ns	$10.00
NVDIMM	10s of GB/s	100-200ns	$10.00
NVMM	10s of GB/s	800ns	$5.00
NVMe	2GB/s	10-100us	$1.40
SATA SSD	500MB/s	400us	$0.40
Disk	100MB/s	10ms	$0.05

- Heterogeneity is storage is increasing
  - Magnetic disks (hard disk drives), NAND-flash solid state drives (SSD), DRAM, NVRAM, hybrid drives, Intel 3DXpoint, Phase Change Memory (PCM)
  - Exhibit different characteristics
- Number of storage devices is increasing
  - **And is log-structured / multi-versioned**
- Local storage system starts to look like a distributed storage system

Research Question

- Can we make local storage system ***weakly consistent*** like a distributed storage system?
  - Weakly consistent means return stale (old versions) of data

StaleStores

- Single-node storage system that maintains and servers multi-versions
- Allows application to make performance consistency tradeoff
  - Higher performance and weak consistency for older version
  - Vs low performance and high consistency for latest version

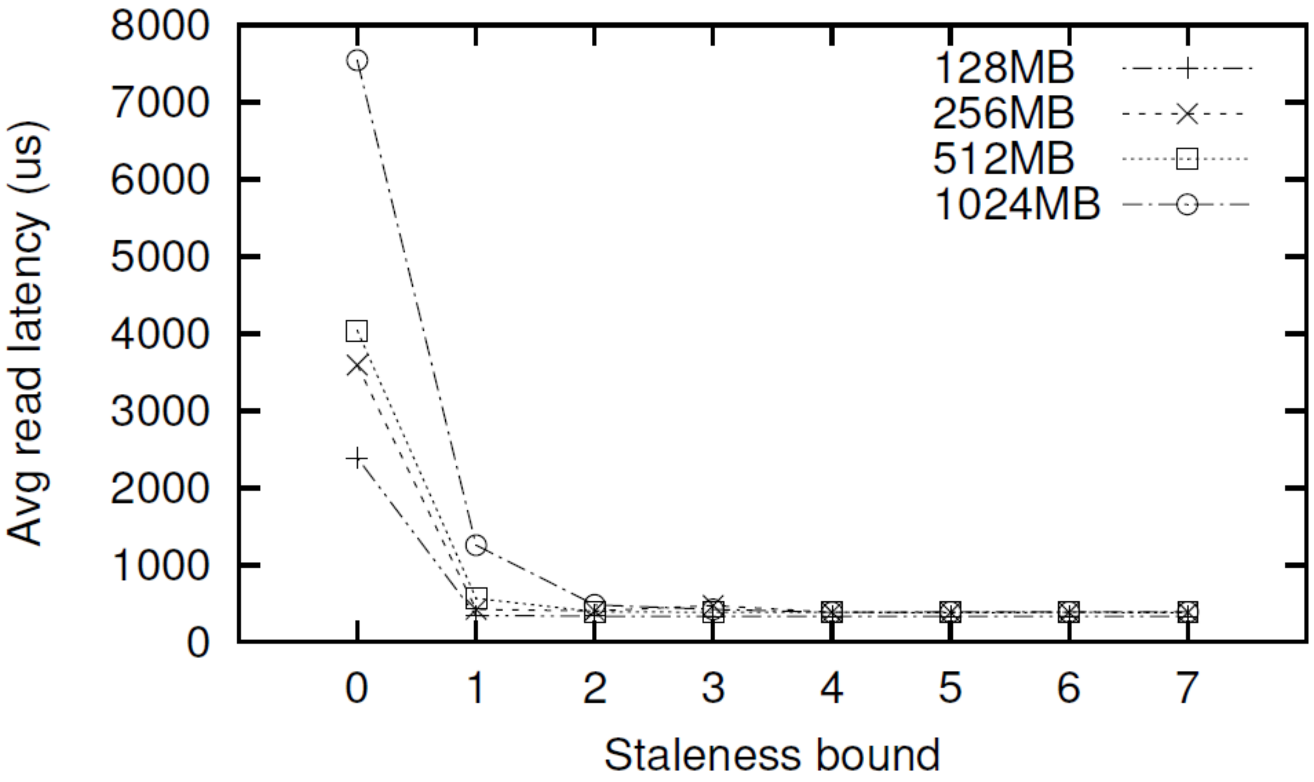
StaleStores

- Requirements:
  - Timestamped writes
  - Snapshot reads
  - Cost estimation
  - Version exploration

StaleStores

- Requirements:
  - Timestamped writes
  - Snapshot reads
  - Cost estimation
  - Version exploration
- API
  - Put
  - Get
  - GetCost
  - GetVersionedRange

Performance: Read Latency



Perspective

- Log-structured is a simple but power abstraction
  - Performance is high since seeks are reduced
  - However, performance suffers if disk nearly full
- Modern day resurrection of Log-Structrued
  - SSD and heterogeneity of storage
- Future log-structured storage systems
  - Trade-off consistency for performance

Next Time

- Read and write review:

- Survey paper due *next* Friday

- Check website for updated schedule

Next Time

- Read and write review:
  - **The Google file system**, Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung. *19th ACM symposium on Operating systems principles (SOSP)*, October 2003, 29--43.

  - **Spanner: Google's Globally Distributed Database**, James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation (OSDI'12)*, October 2012, 251--264.