File Systems: FFS and LFS

A Fast File System for UNIX
McKusick, Joy, Leffler, Fabry
TOCS 1984

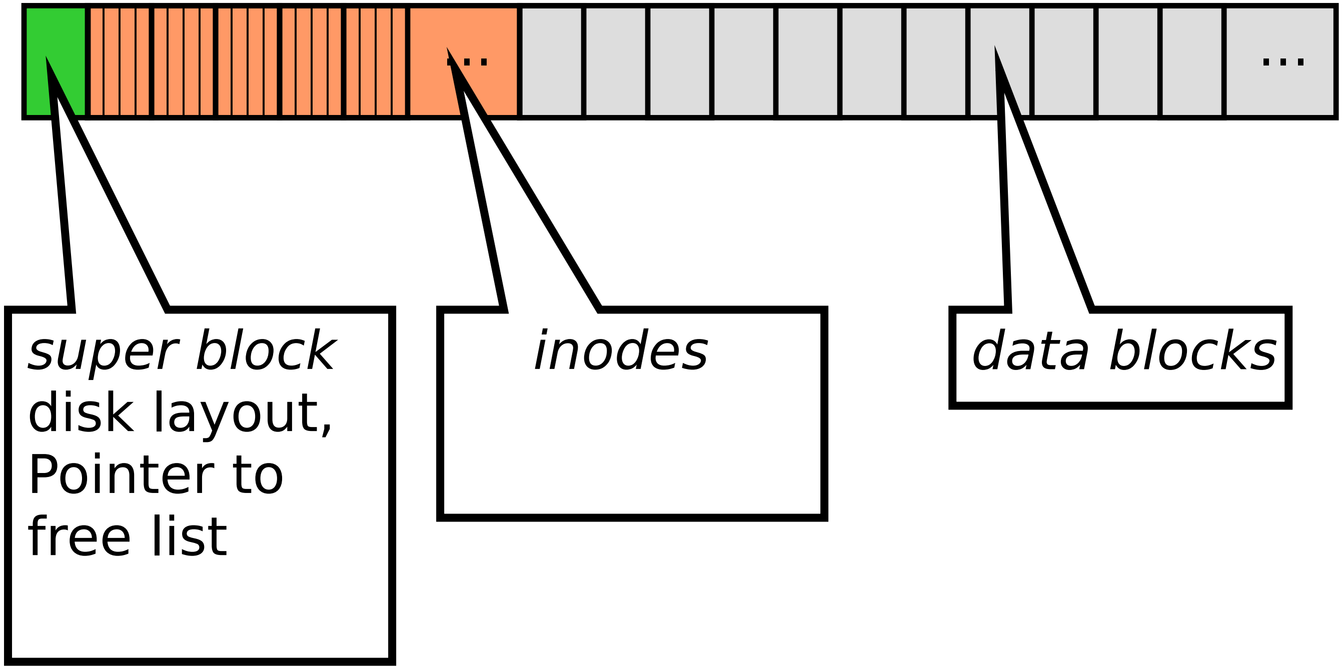
The Design and Implementation of a Log-Structured File System
Rosenblum and Ousterhout
SOSP 1991

Presented by: Dan Williams
(some slides borrowed from Ken Birman, Ben Atkin)

Overview

- Original UNIX file system
- Two systems to improve file system performance
  - Fast File System
  - Log-Structured File System
- Conclusions

File system on disk

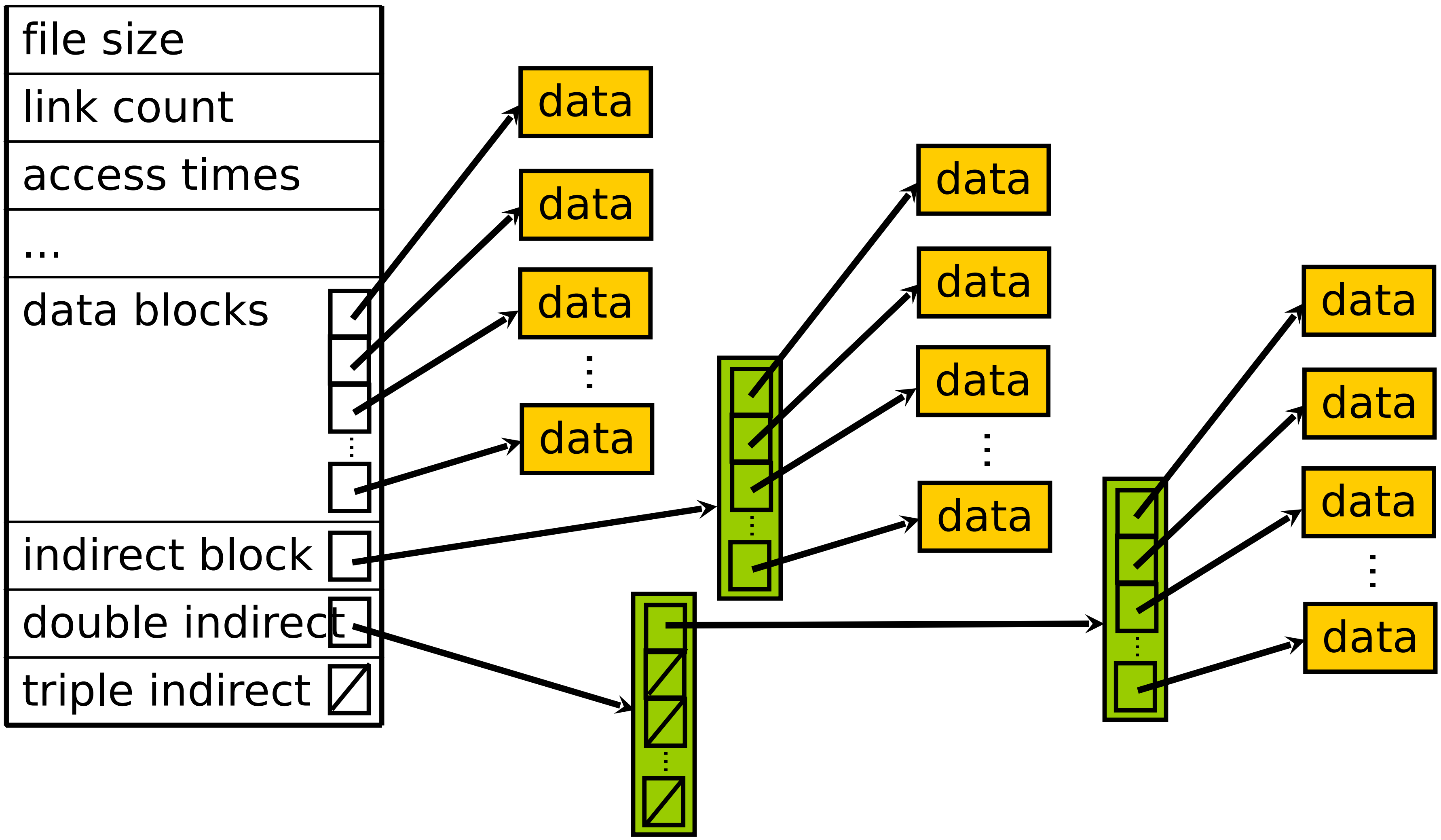


super block
disk layout,
Pointer to
free list

inodes

data blocks

Inodes


file size
link count
access times
...
data blocks
indirect block
double indirect
triple indirect
data
data
data
data
data
data
data
data
data
data
data
data

Performance Problems

- Lots of disk seeks
  - Inodes are far away from data blocks
- Lots of indirection
  - Small block size (512 bytes)
- Fragmentation
  - Even more disk seeks

Overview

- Original UNIX file system
- Two systems to improve file system performance
  - **Fast File System**
  - Log-Structured File System
- Conclusions

Fast File System

- Less disk seeking = better performance
- Become “disk-aware”
  - Better placement of data on disk
  - Cylinder groups
  - Layout policies
- Increase block size

Cylinder Groups

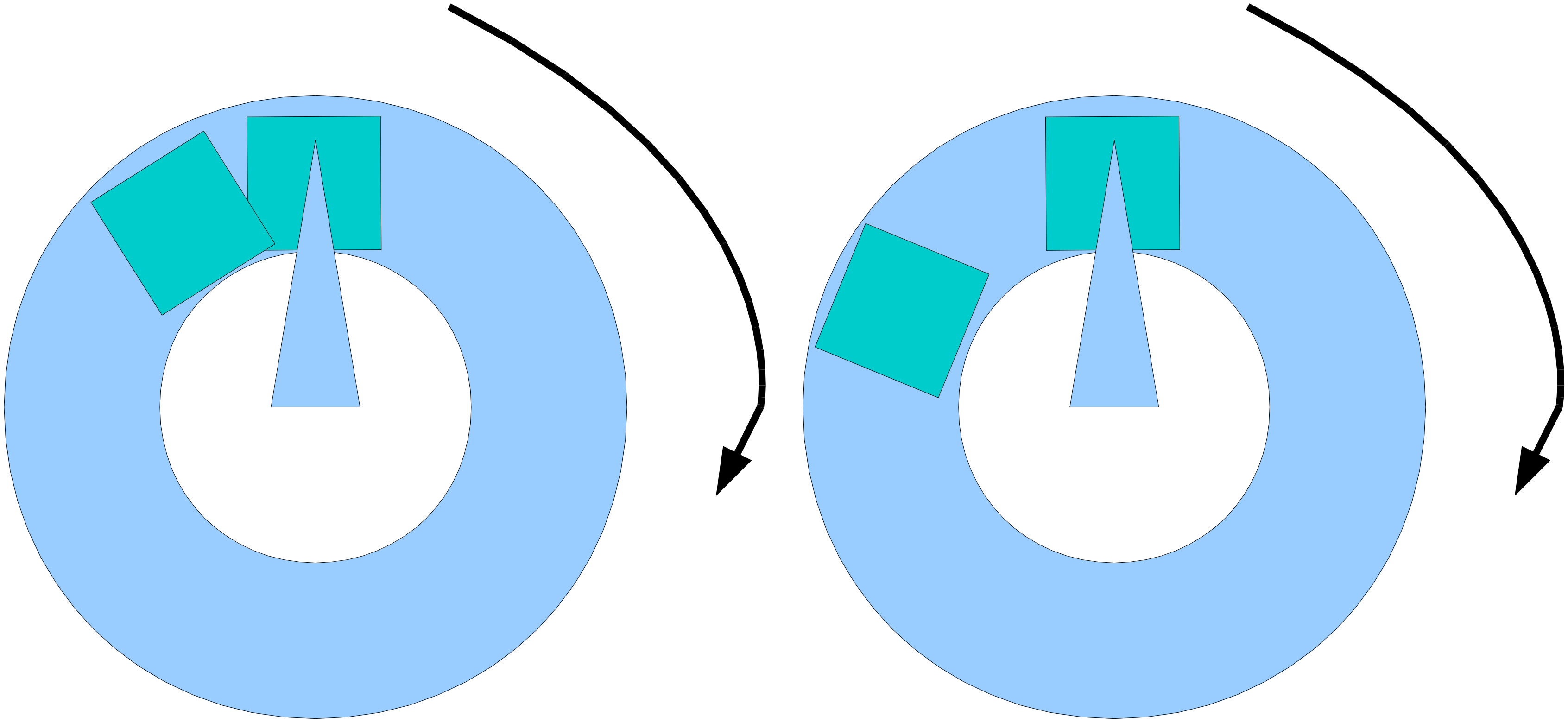
- Split disk into groups, each with
  - Super block copy (for redundancy)
  - inode/data block bitmap
  - Inodes
  - Data blocks
- Try not to seek very far

Layout Policies

- Keep related files together
  - Inodes for files in same directory
- Don't fill entire cylinder group with single large file

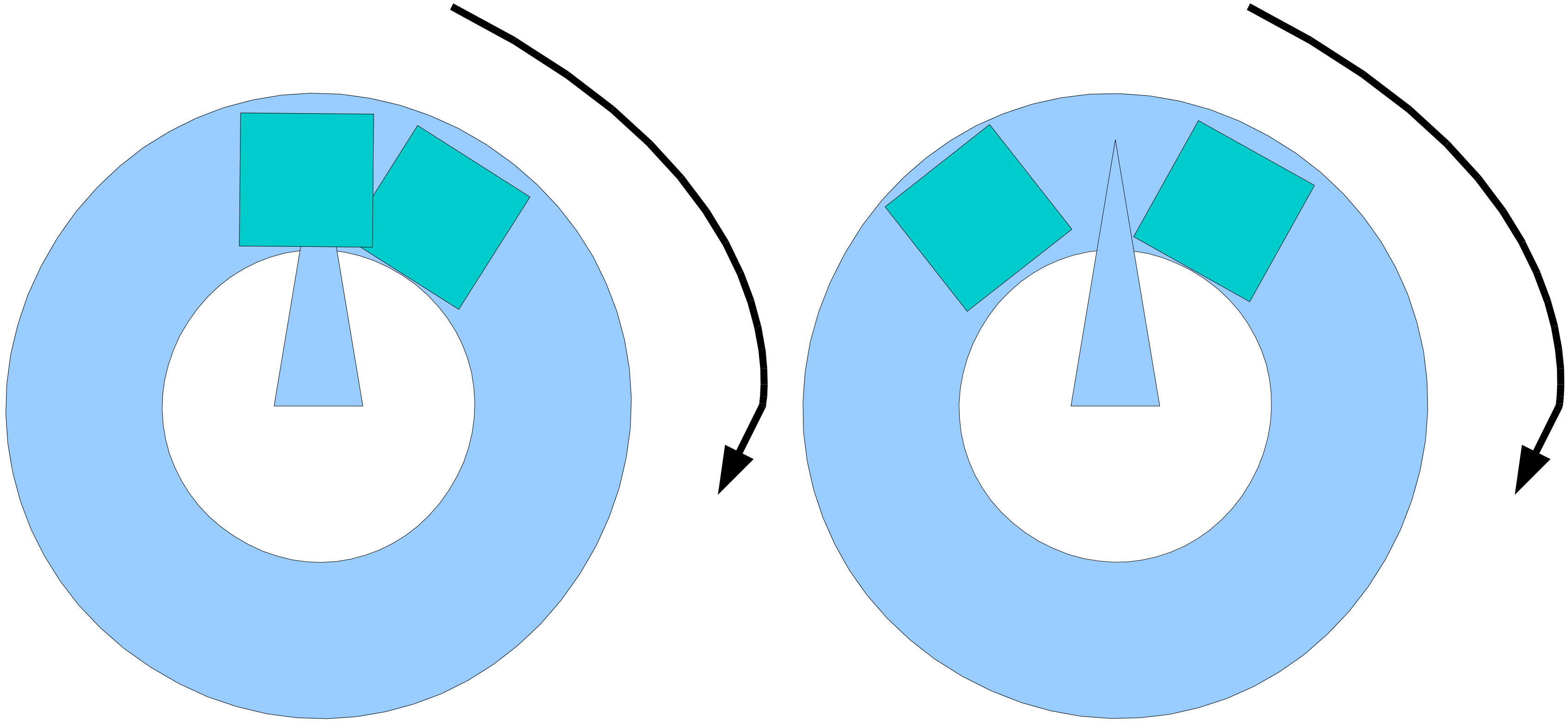
Rotation-aware placement

- Factor in time for disk spinning for layout
  - Time to service interrupts and wait for new transfer



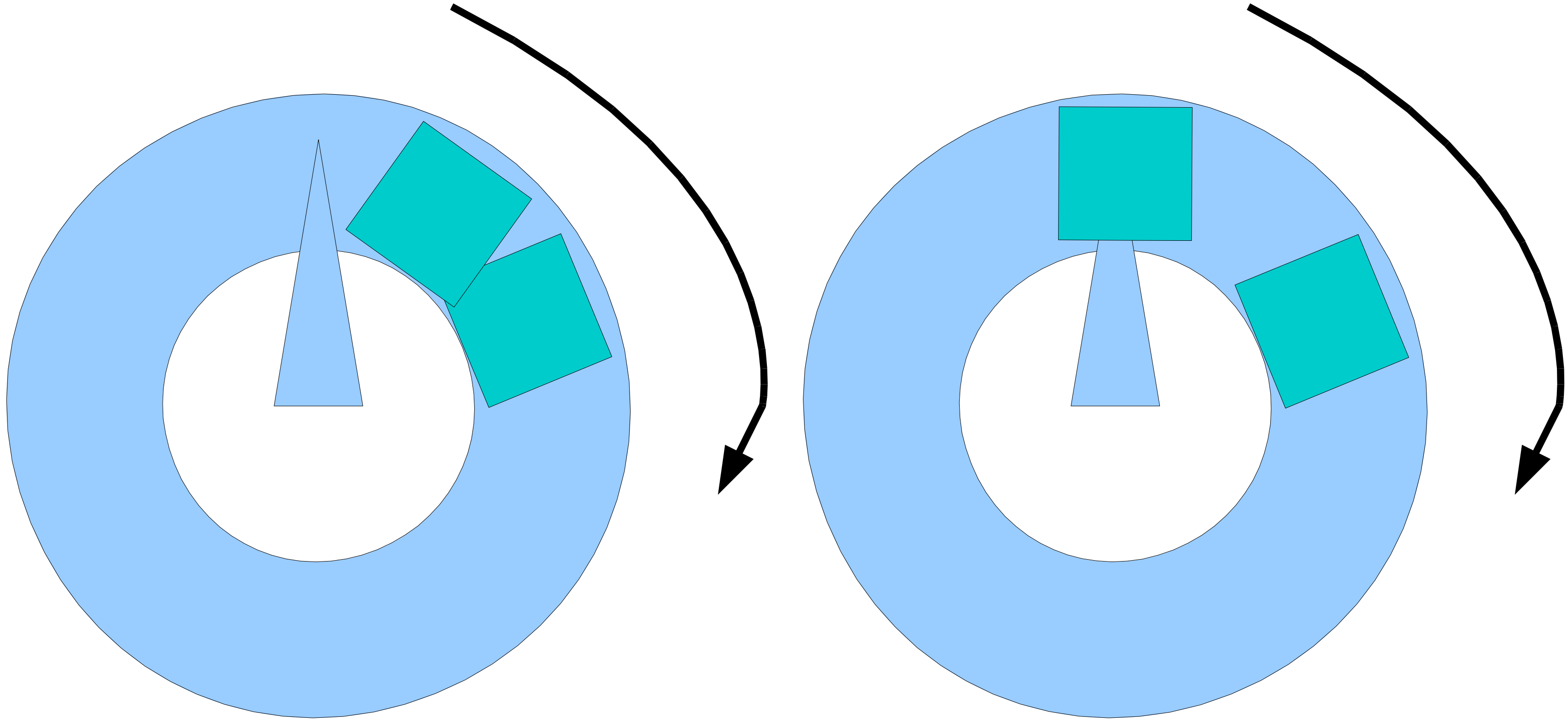
Rotation-aware placement

- Factor in time for disk spinning for layout
  - Time to service interrupts and wait for new transfer



Rotation-aware placement

- Factor in time for disk spinning for layout
  - Time to service interrupts and wait for new transfer



FFS Discussion

- Become disk aware
- Complex layout policies
- Rotation-aware doesn't make sense today
    - Disks have caches w/ read-ahead
- Still lots of small seeks for file updates
    - Not as good as sequential file access

Overview

- Original UNIX file system
- Two systems to improve file system performance
  - Fast File System
  - **Log-Structured File System**
- Conclusions

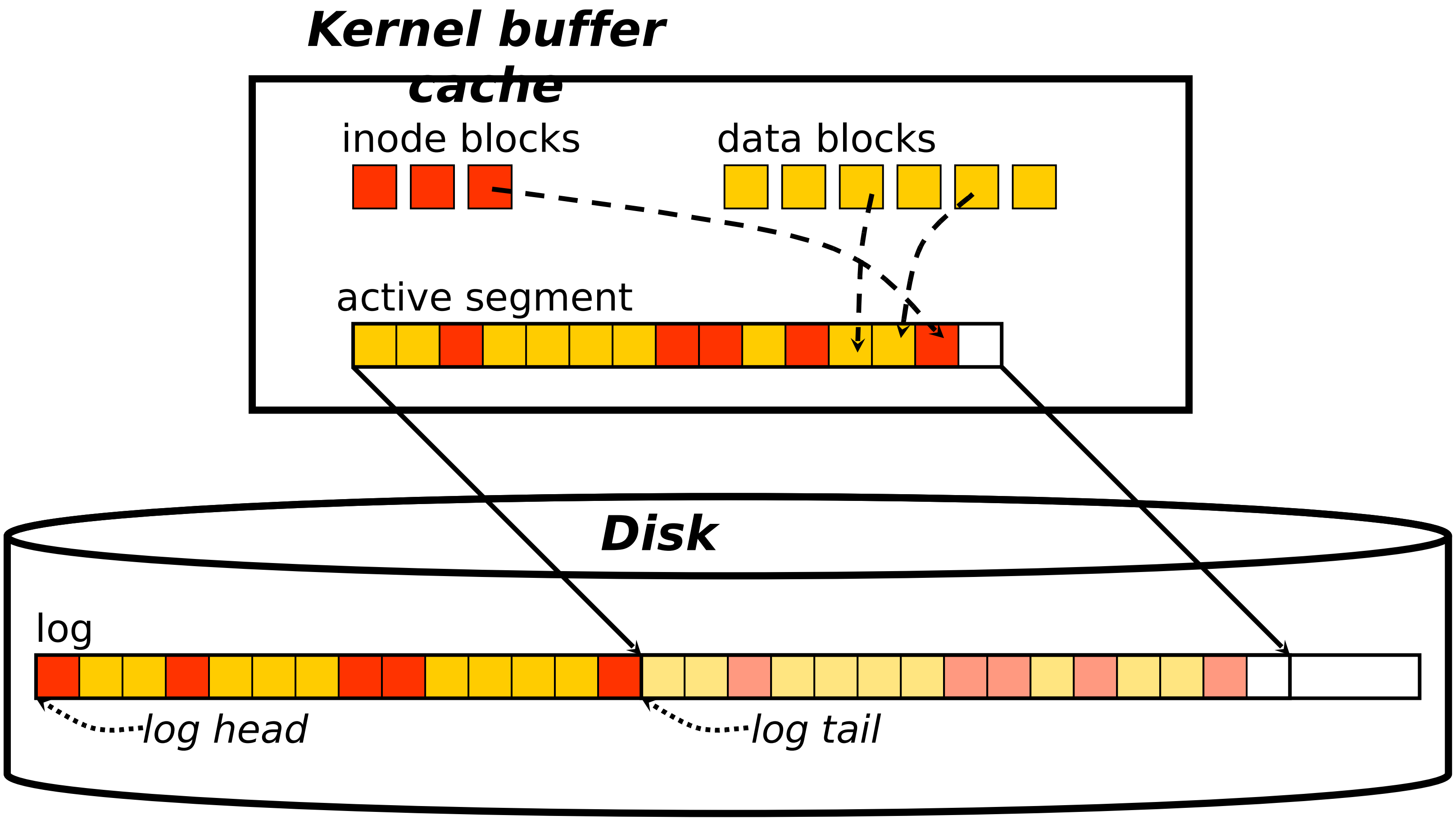
LFS Motivation

- More memory means bigger FS cache
  - Writes dominate I/O
  - Writes can be buffered and batched
- Random vs. sequential I/O gap increasing
  - Big win with sequential I/O
- Updates are expensive for traditional FS
  - Such as FFS

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

Log operation



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*

Cleaning the log

- 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

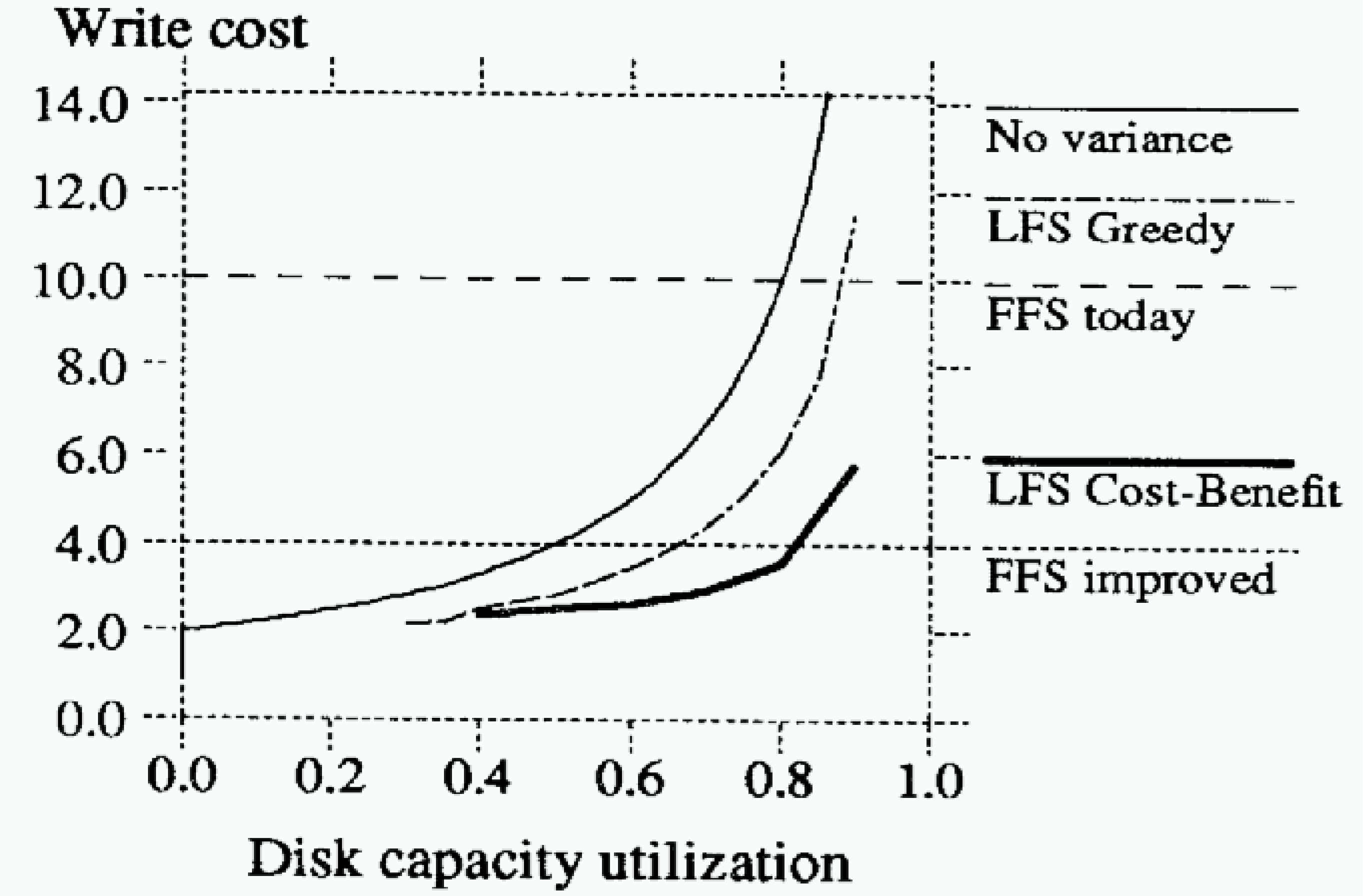
Cleaning policies

- Simulations to determine best policy
  - Greedy: clean based on low utilisation
  - Cost-benefit: use age (time of last write)
    - Hot vs. cold

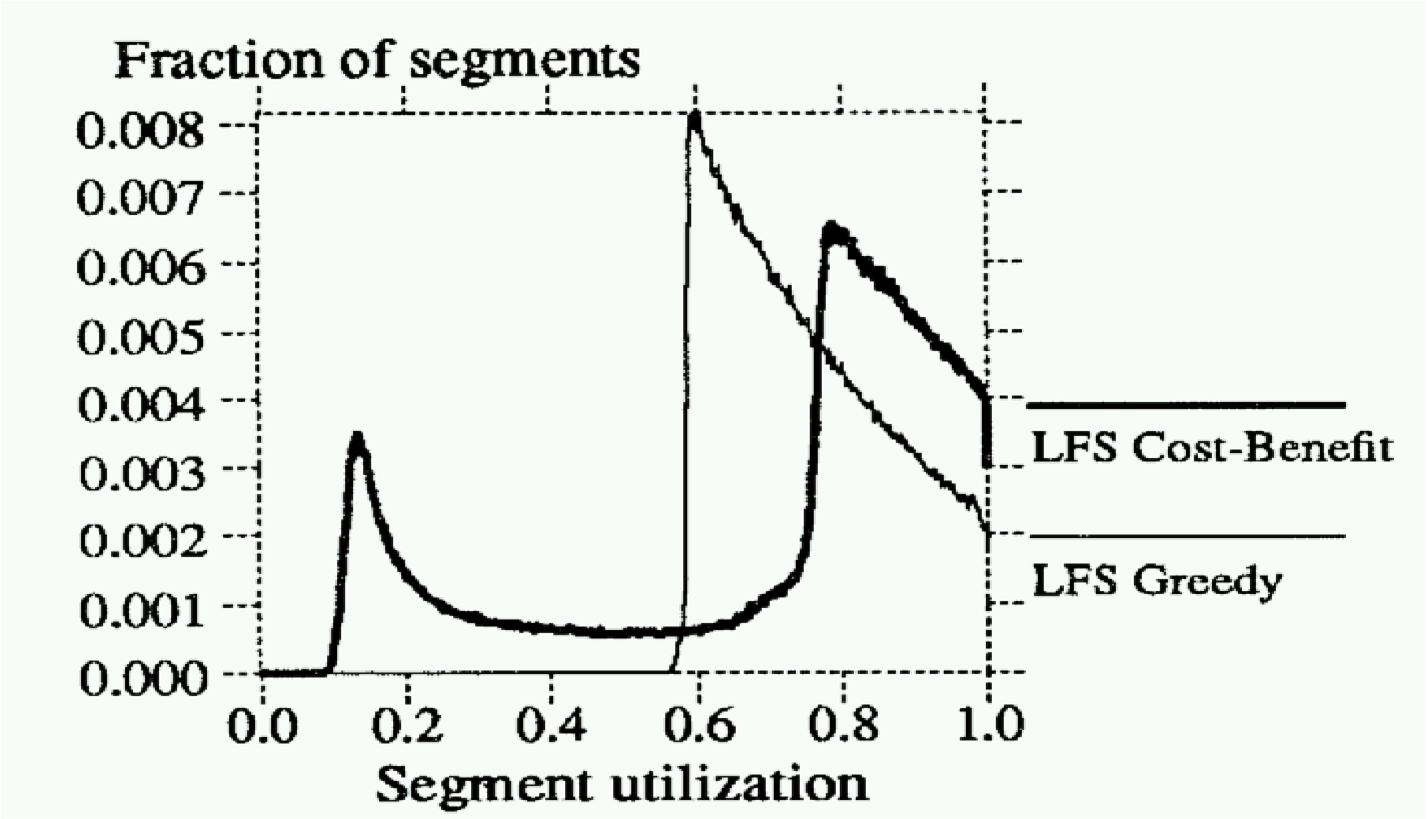
$$\frac{\texttt{benefit}}{\texttt{cost}} = \frac{\texttt{(free space generated)*(age of segment)}}{\texttt{cost}}$$

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

Greedy versus Cost-benefit



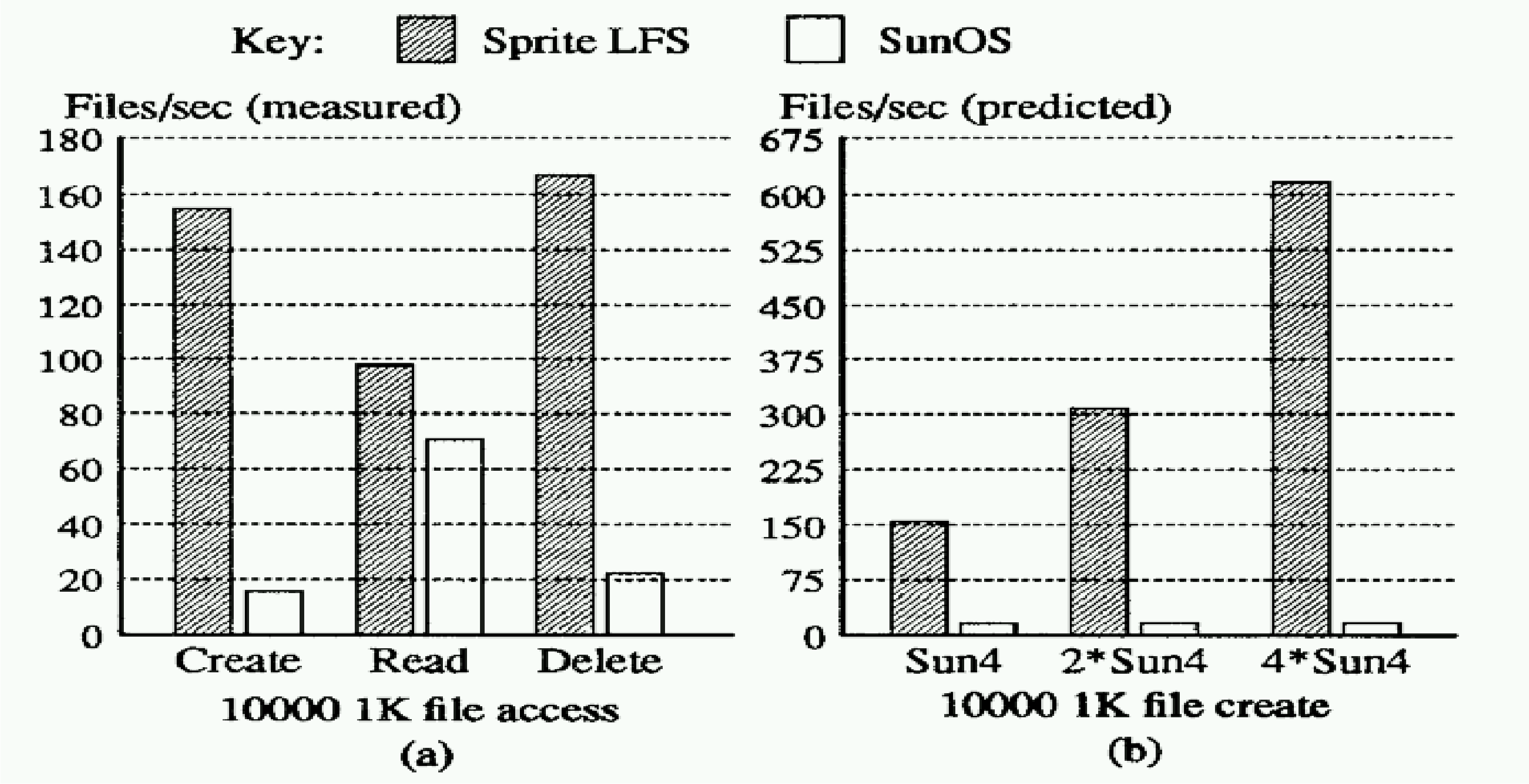
Cost-benefit segment utilisation



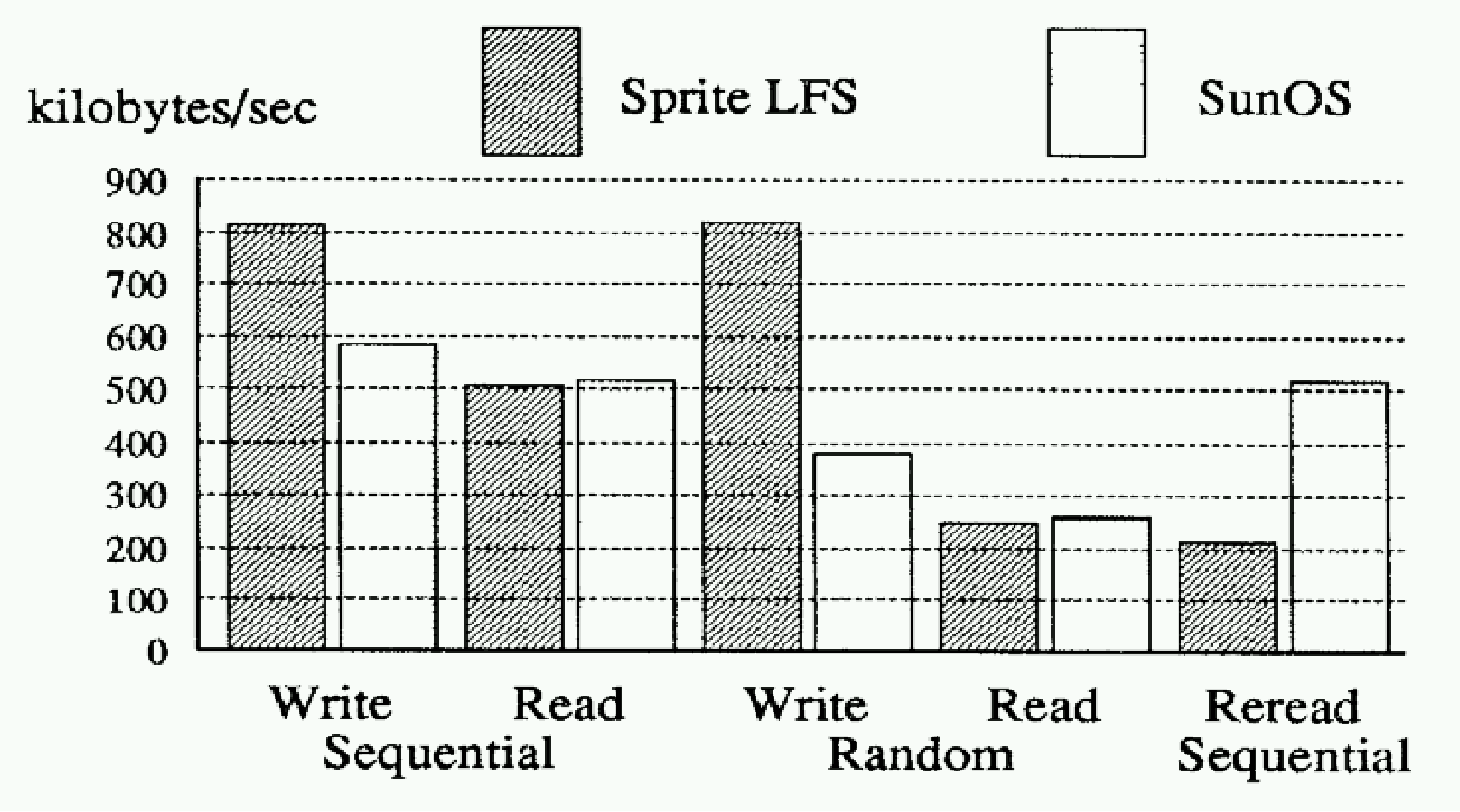
LFS performance

- 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



Summary

- Both increased performance and were influential
- FFS
  - Optimize existing FS
- LFS
  - Rethink FS
  - How expensive is cleaning?