Future Work

- Different Cleaners.
- Assess disk utilization vs. performance for LFS in TP1-like benchmarks.
- Try to make FFS recover quickly (do inode and block allocation in batches).
- Figure out if LFS is really viable.
- Papers available via anonymous ftp:

  toe.cs.berkeley.edu:pub/personal/margo/

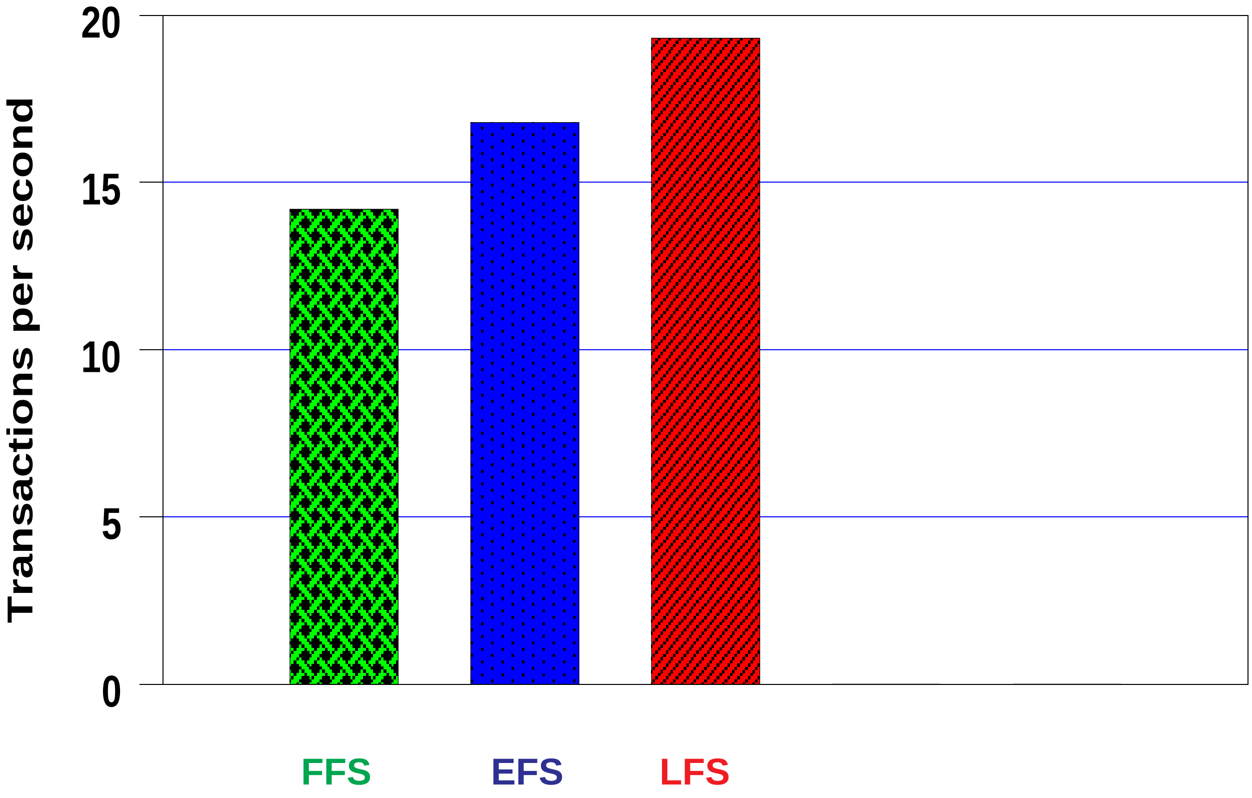
  thesis.ps.Z

  usenix.1.93.Z

Conclusions

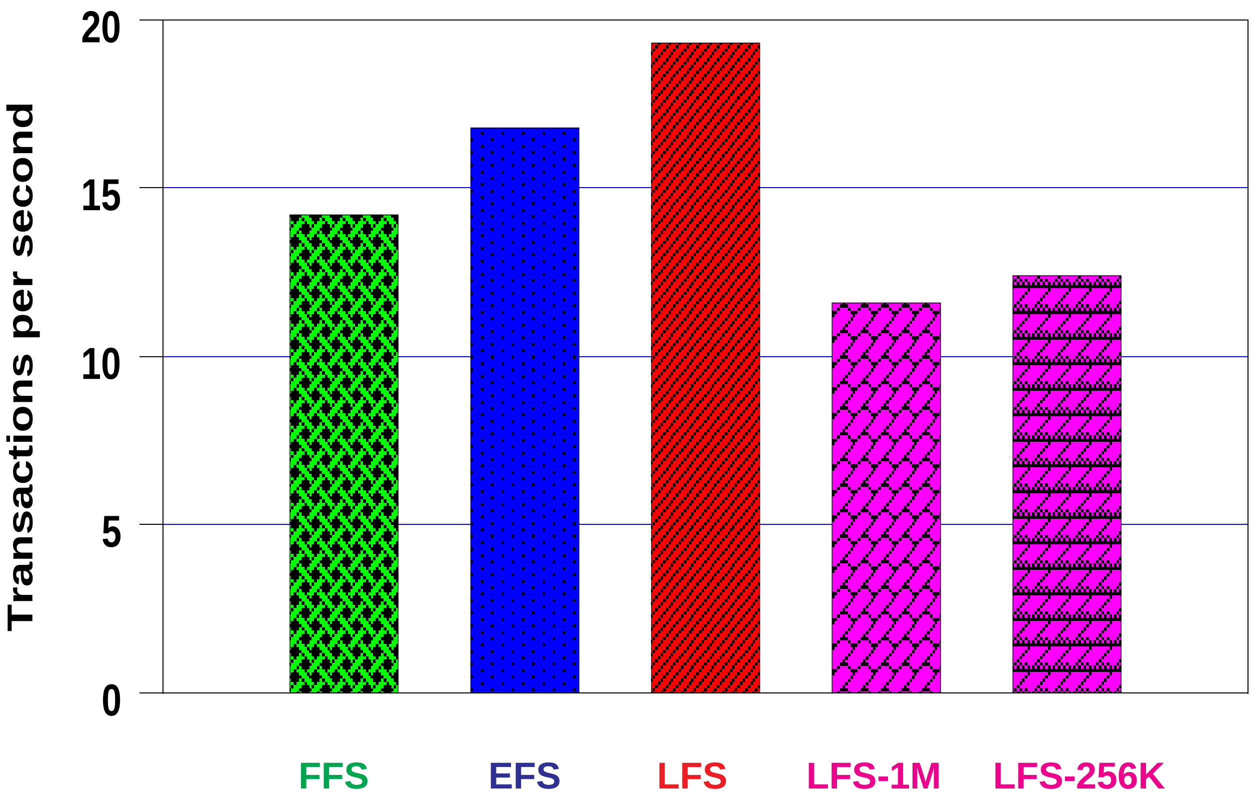
- Garbage Collection: Consider it harmful!
- Asynchronous directory operations are good.
- Clustering is good.
- Clustering writes of different files, not obviously such a win.
- FFS is remarkably flexible and robust.

TP1 Performance



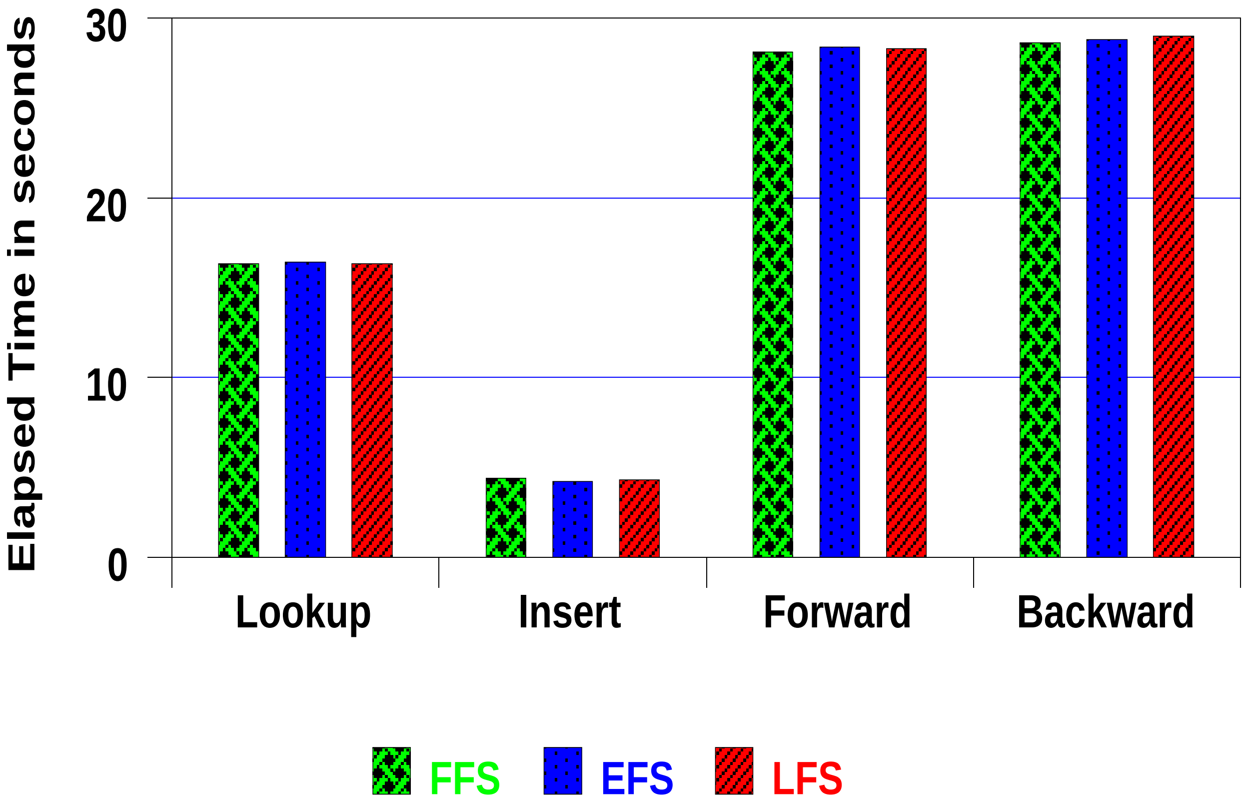
CONCLUSIONS

TP1 Performance



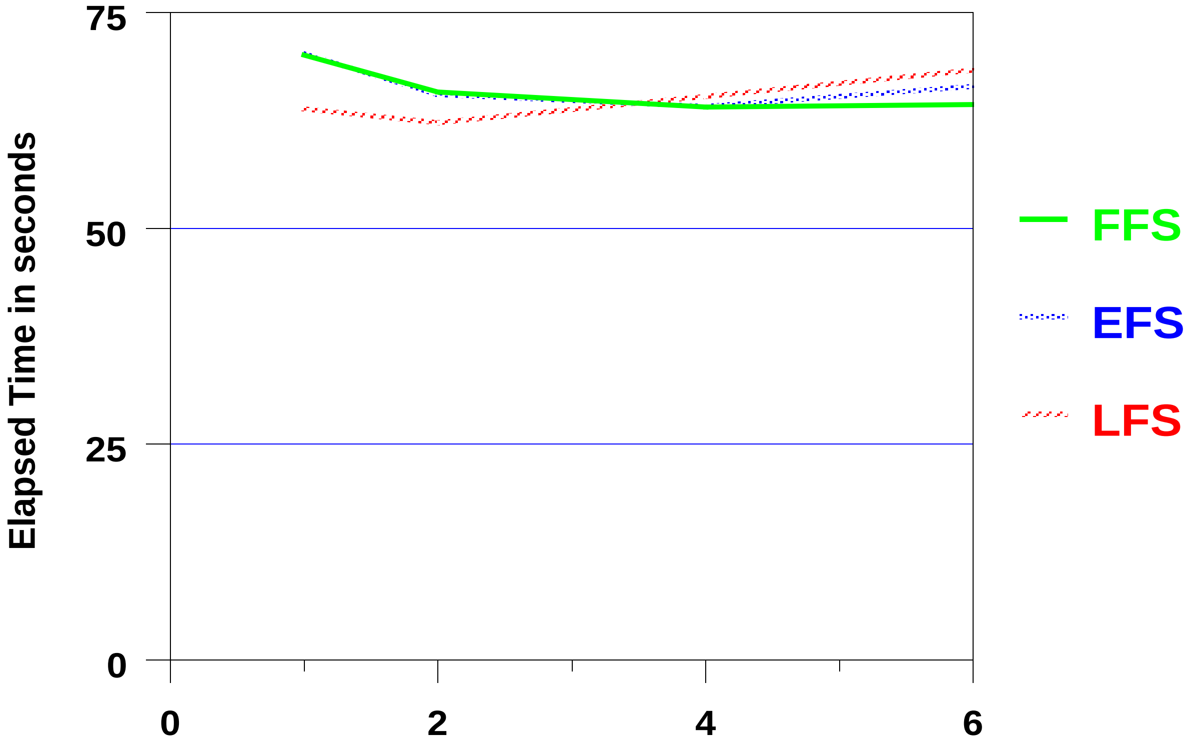
PERFORMANCE

OO1 Performance

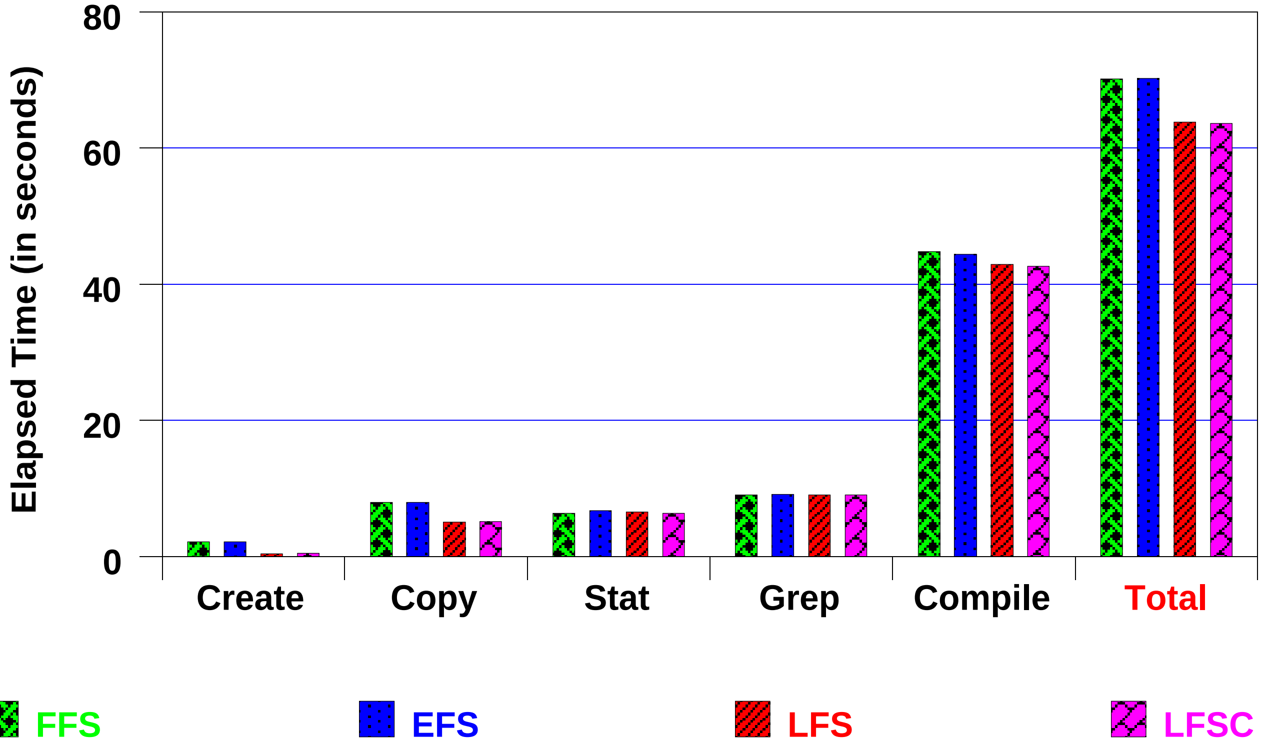


PERFORMANCE

Multi-User Andrew Performance

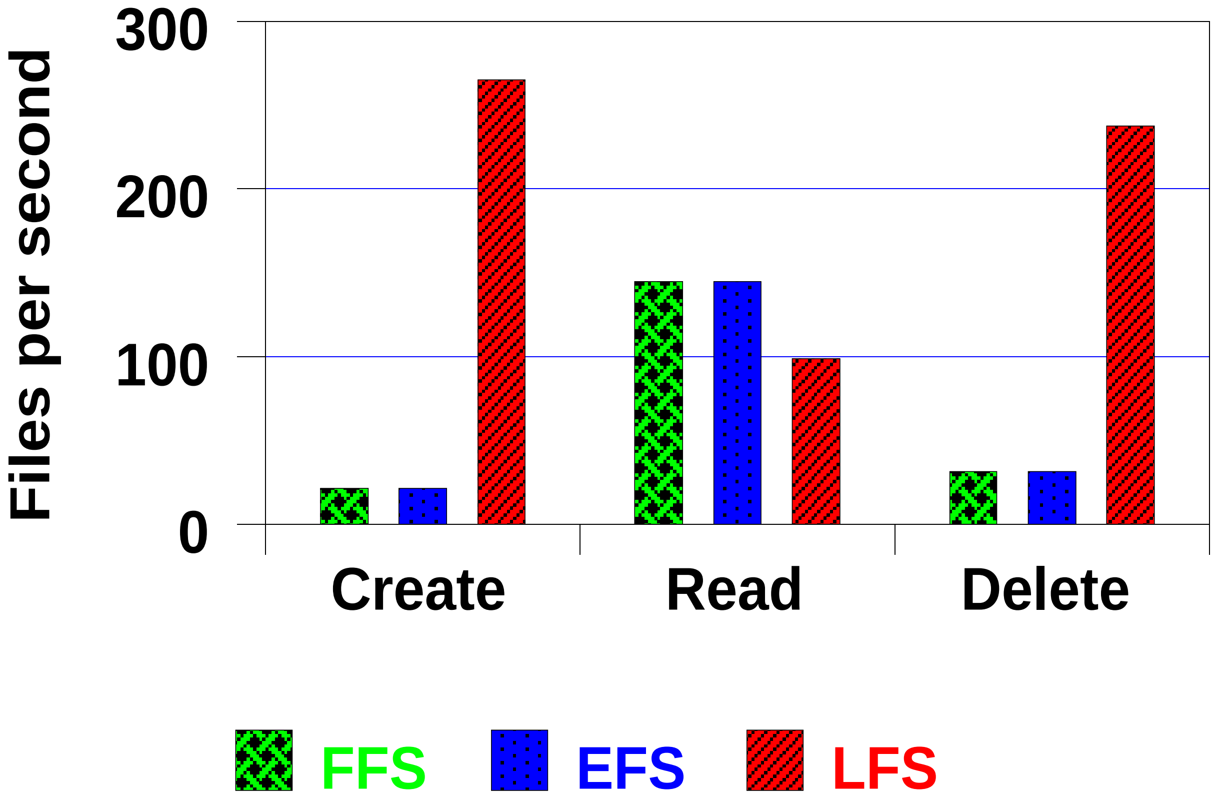


Single-User Andrew Performance



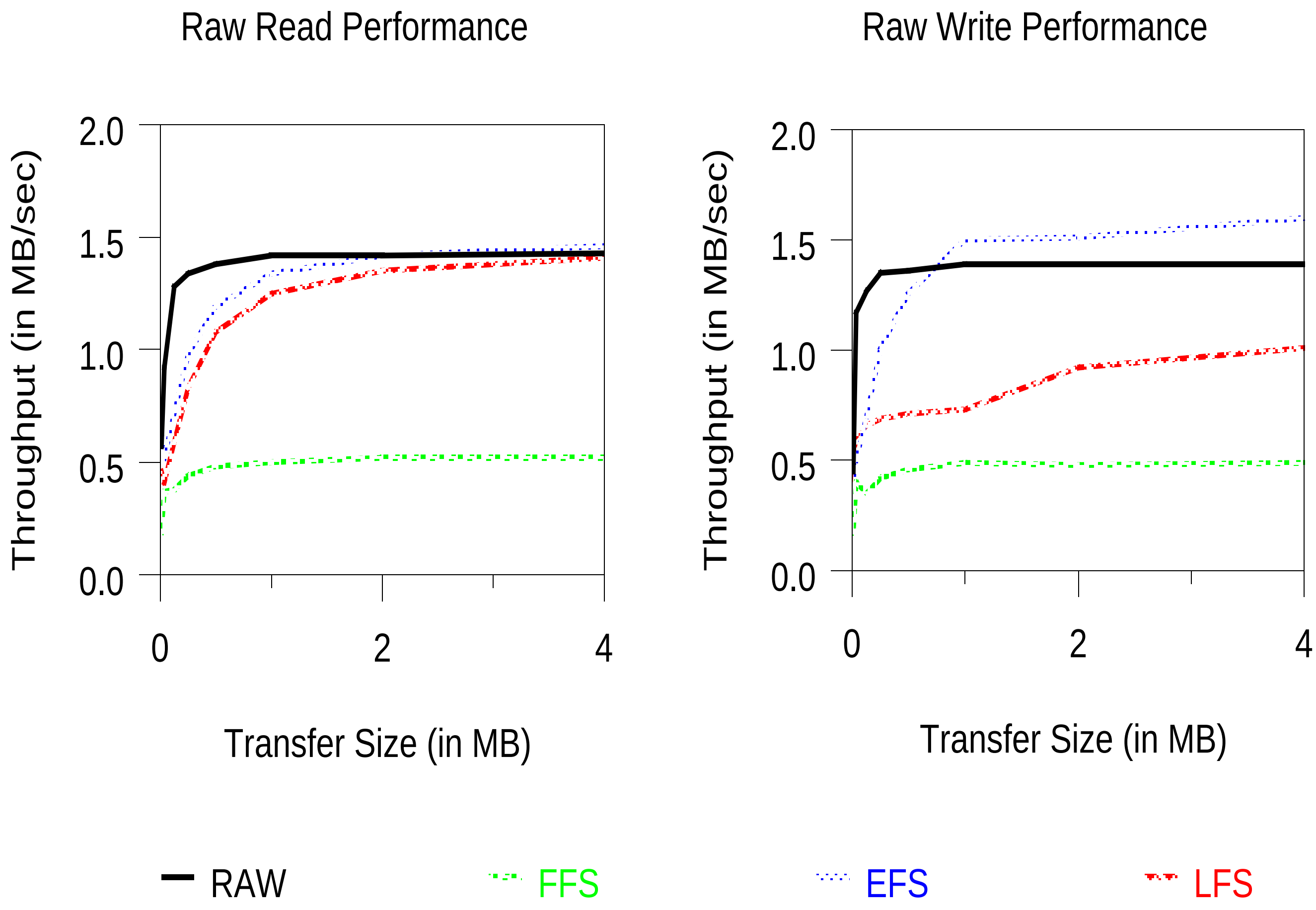
PERFORMANCE

Small File Performance



PERFORMANCE

Raw Performance



Performance

- **Compare three systems:**

  LFS: BSD Log-Structured File System

  FFS: Standard BSD Fast File System

  EFS: FFS with clustering turned on and *maxcontig* set so that cluster is 64K (maximum allowed by our controller).

- **HP9000/380 (25 Mhz 68040)**
- **SCSI SD97560 (13 ms average seek, 15.0 ms rotation, 1.6 MB/sec maximum bus bandwidth).**

Read-Ahead: Pleasures and Pitfalls

- **Sequential case easy: get nearly 100% of I/O bandwidth.**
- **Problem: How much do you read-ahead?**
- **Consider reading 8K logical pages on a 4K file system.**
- **Placing read-ahead blocks on regular queue can cause cache thrashing**

Clustering in the Fast File System

Extent-like Performance from a UNIX File System

Larry McVoy, Steve Kleiman
Proceedings 1991 Usenix Technical Conference
January 1991

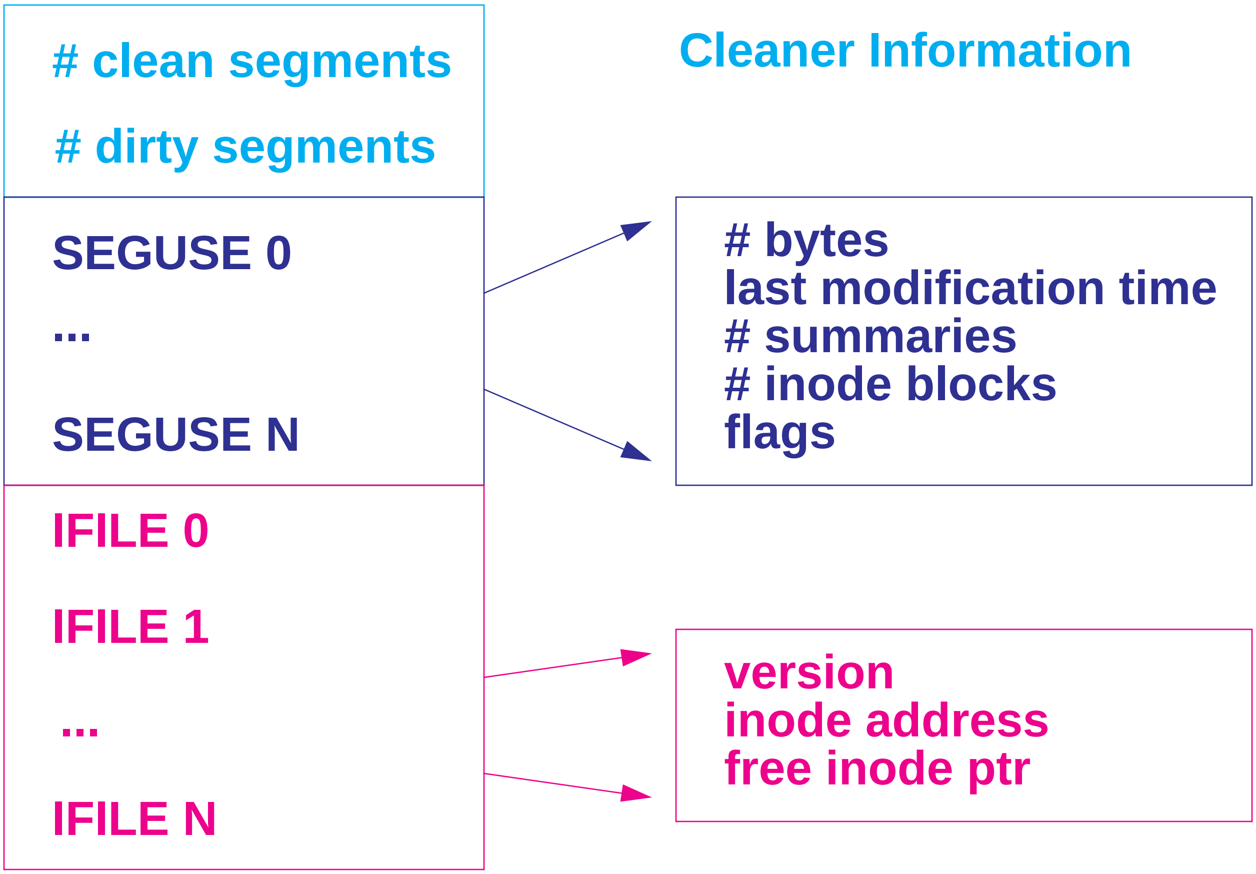
- **Set *maxcontig* high (a track or maximal unit to controller).**
- **Read/Write clusters of contiguous blocks.**
- **350 additional lines to FFS.**

Comparison to FFS

FFS	LFS
Replicated Superblock	Replicated Superblock
Cylinder Groups	Segments
Inode Bitmaps	Inode Map
Block Bitmaps	Segment Summaries
	Segment Usage Table

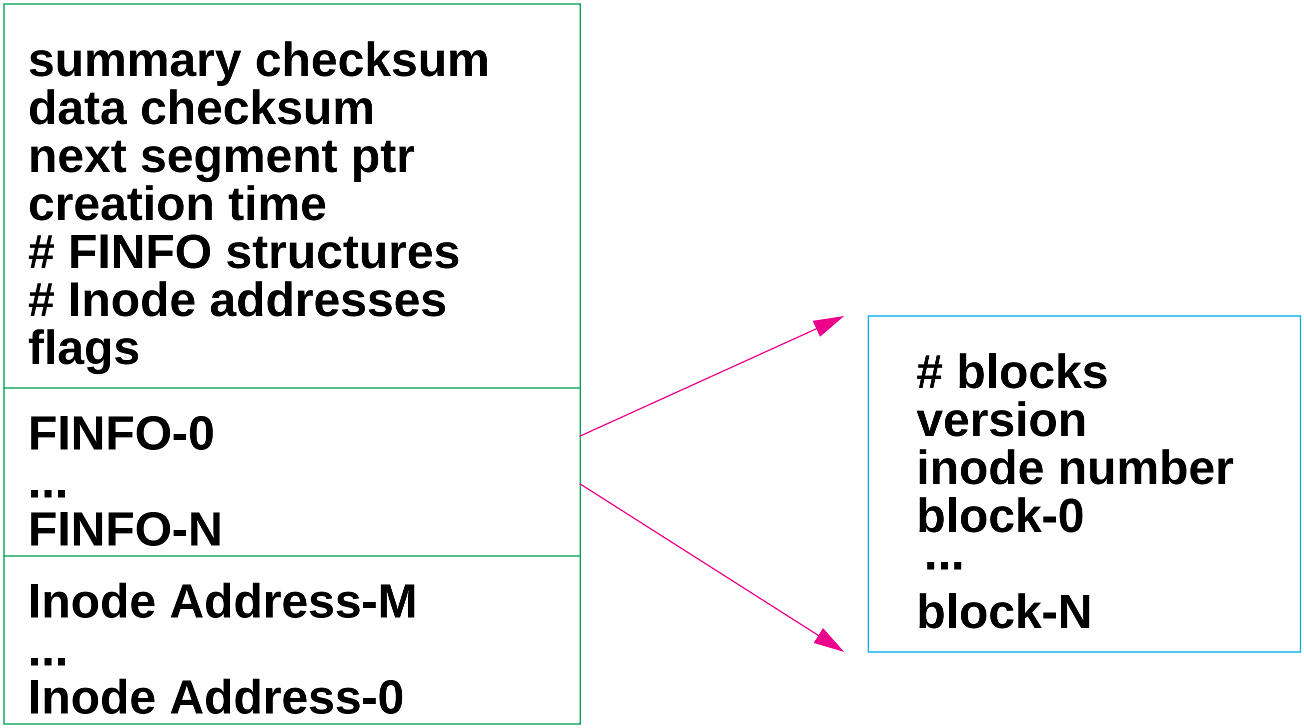
CLUSTERED FFS

The Ifile

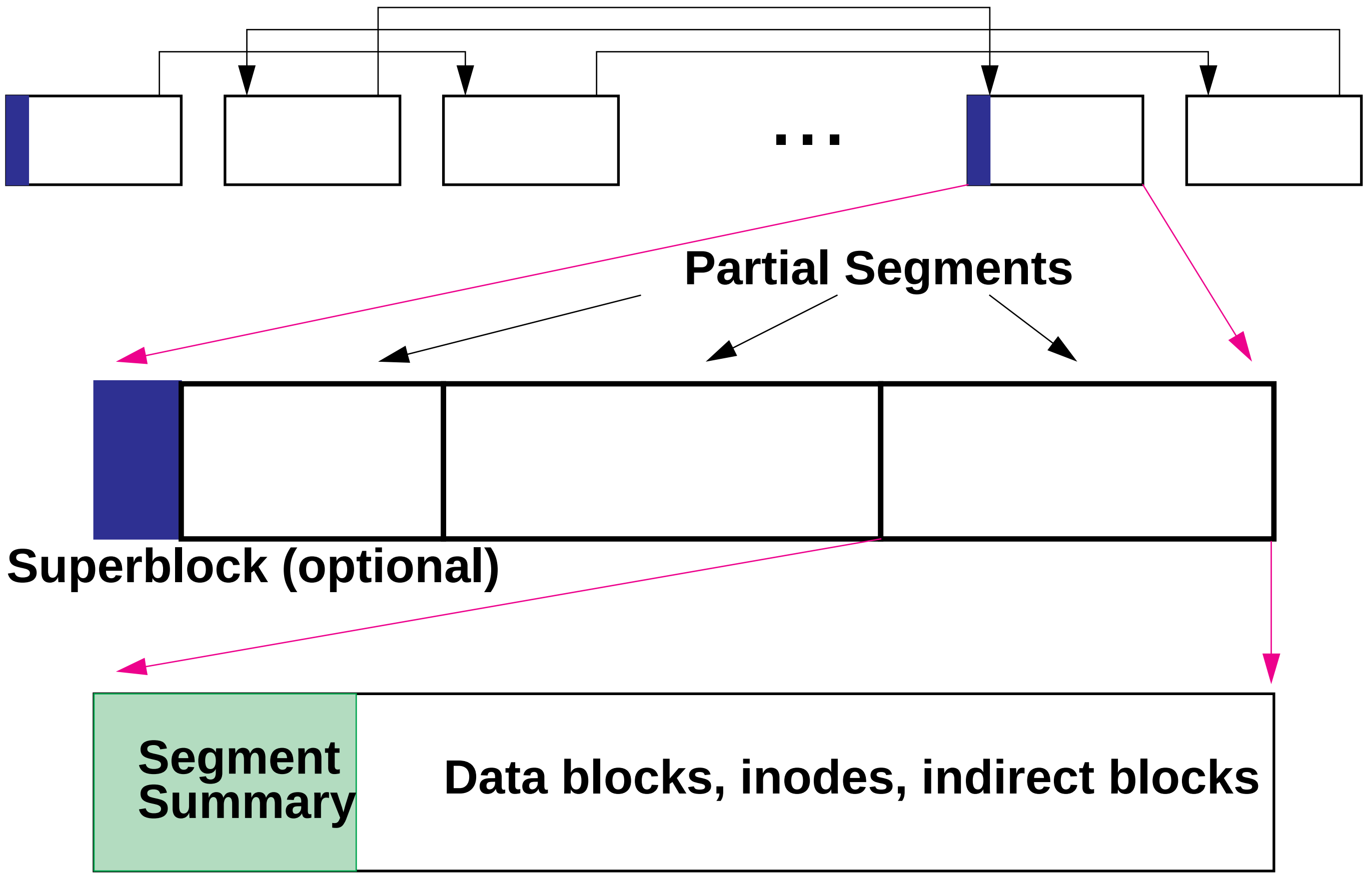


clean segments
dirty segments
Cleaner Information
SEGUSE 0
...
SEGUSE N
bytes
last modification time
summaries
inode blocks
flags
IFILE 0
IFILE 1
...
IFILE N
version
inode address
free inode ptr

Segment Summary



Segments



...
Partial Segments
Superblock (optional)
Segment Summary
Data blocks, inodes, indirect blocks

New Data Structures

- Inodes no longer in fixed locations.

  Introduce inode map to locate inodes.

- Segments must be self-identifying.

  Use segment summary blocks to identify blocks.

- Must know which segments are in use.

  Maintain segment usage table.

Data Structures

- Segments
- Partial Segments
- Segment Summary Blocks
- FINFO Structures
- IFILE
- Cleaner Info
- Segment Usage Structure
- Inode Map

Inode Allocation

- Sprite: Inode map is a sparse array.

  Directories allocated randomly.

  Files allocated by searching sequentially after directory.

  + Clustering in IFILE

  - Linear searching.

- BSD: Maintain free inodes in linked list.

  + Fast allocation.

  - No clustering in IFILE.

Directory Operations

- **Sprite: Maintains additional on-disk data structure to perform write-ahead logging.**
- **BSD: Uses “segment-batching” to guarantee ordering of directory operations.**

Sprite writes less data.

BSD avoids extra on-disk structure.

Roll forward simpler in BSD.

Does anyone really care???

BSD-LFS

The Inode Map and Segment Usage Table

- **Sprite: Special kernel memory structures**
- **BSD: Stored in regular IFILE (read-only to applications; written by the kernel).**

  Simplifies kernel.

  Provides information to cleaner.

BSD-LFS

Free Block Management

- Sprite: does not check disk utilization until block is written to disk.

  Can accept writes for which there is no disk space!

- BSD does two forms of accounting:

  Free blocks: blocks on disk that do not contain valid data.

  Writable blocks: clean segments available for writing.

Memory Usage

- Sprite reserves large portions of memory
  - 2 staging buffers
  - one segment system-wide for cleaning
  - 1/3 of buffer cache reserved read-only
- BSD uses normal buffer pool buffers, allocates space dynamically when necessary
- Cleaner competes for virtual space.

The Cleaner

- Sprite: Kernel process

  Single process cleans all file systems

  Kernel memory reserved for cleaner

- BSD: Cleaner runs as user process

  Reads IFILE

  Uses system calls to get block addresses and write out cleaned blocks

  Competes for VM with other processes

BSD-LFS

Design Changes

- The Cleaner
- Memory Usage
- Free Block Management
- The Inode Map and Segment Usage Table
- Directory Operations
- Inode Allocation

4.4BSD-LFS

An Implementation of a Log-Structured File System for UNIX

Margo Seltzer, Keith Bostic, Kirk McKusick, Carl Staelin
Proceedings Usenix Technical Conference
January 1993

- **New design and implementation**
- **Merged into vfs/vnode framework.**
- **60% code shared with FFS.**
- **Data structures similar to FFS.**

Sprite-LFS

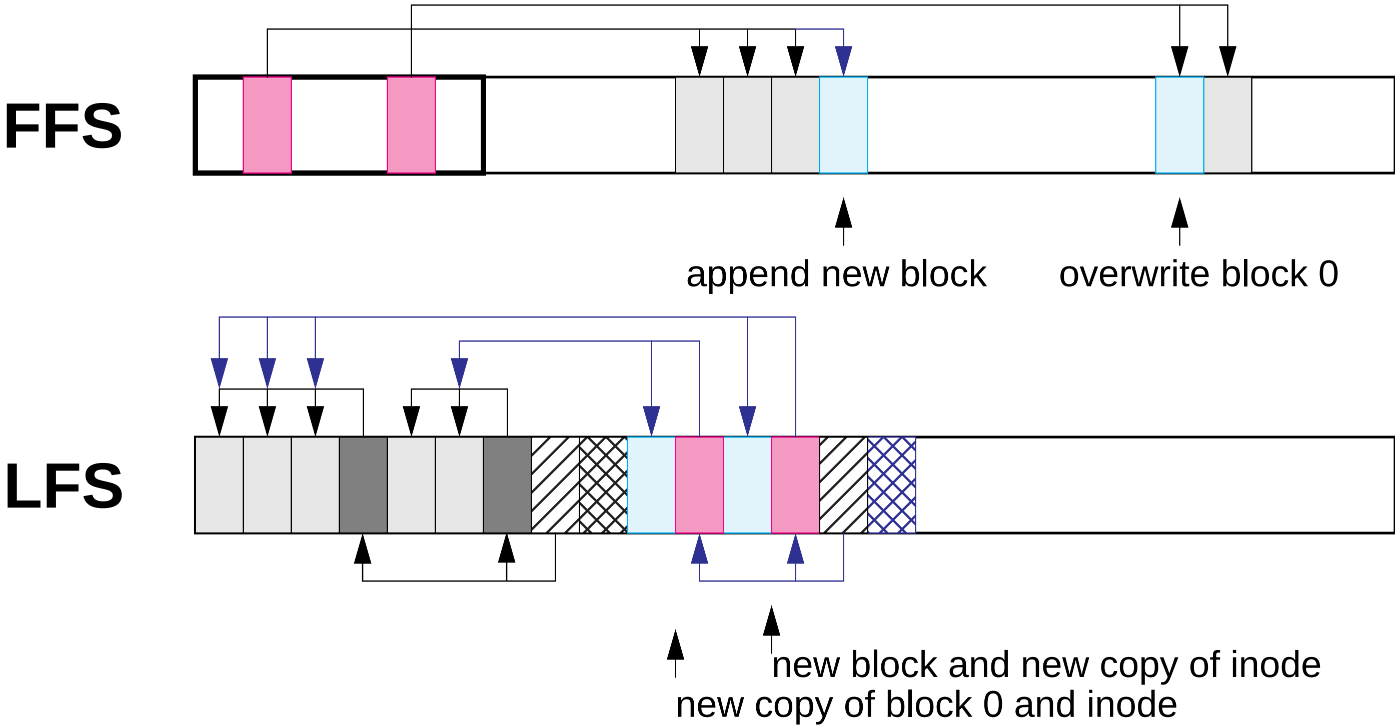
The Design and Implementation of a Log-structured File System

Mendel Rosenblum
Operating Systems Review
October 1991

- **Runs on the Sprite experimental operating system.**
- **LFS Running since 1990.**
- **10 Active file systems including home directories, source tree, executables, and swap.**

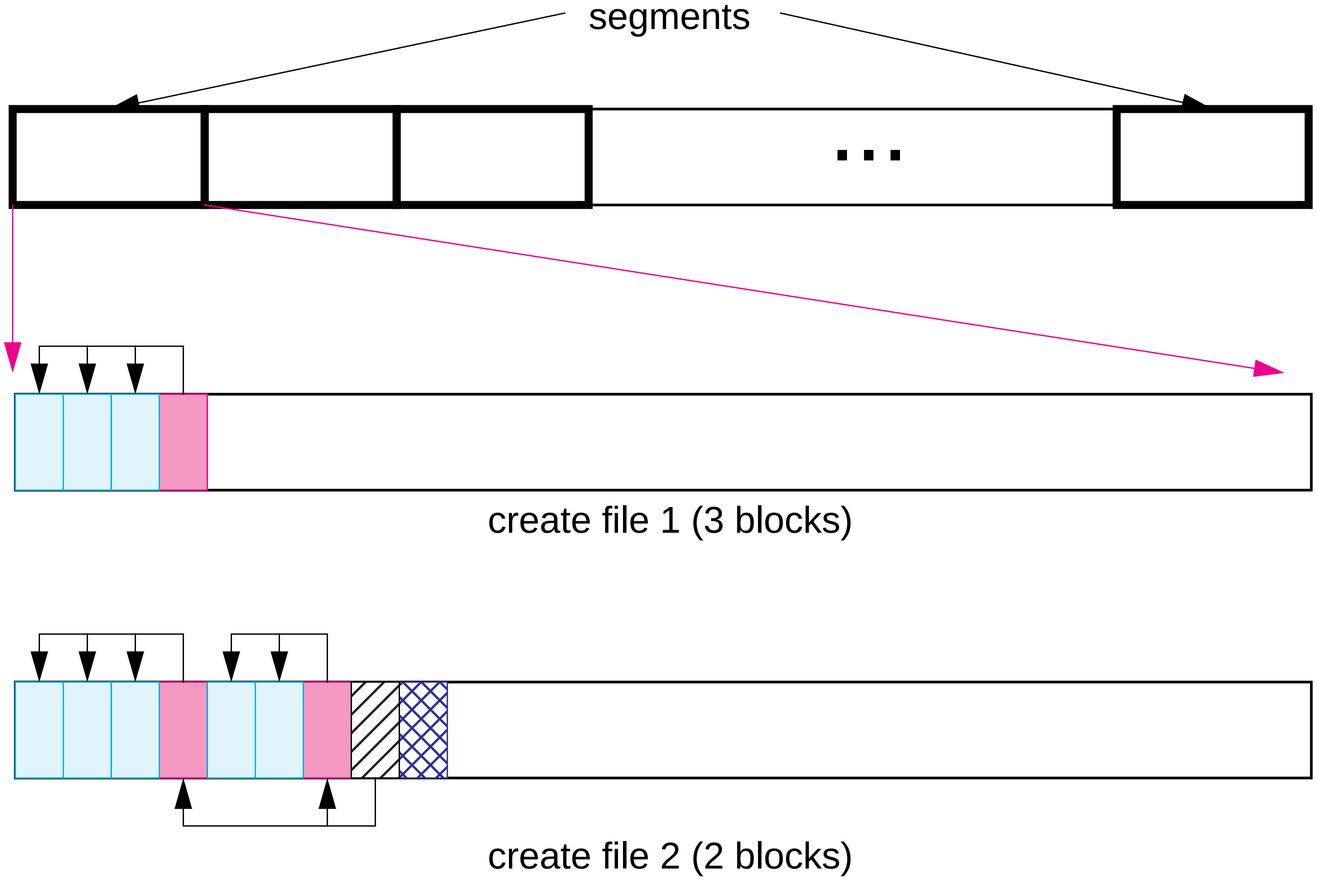
Extending or Modifying Files

- Update block 0 in file 2
- Append a block to file 1



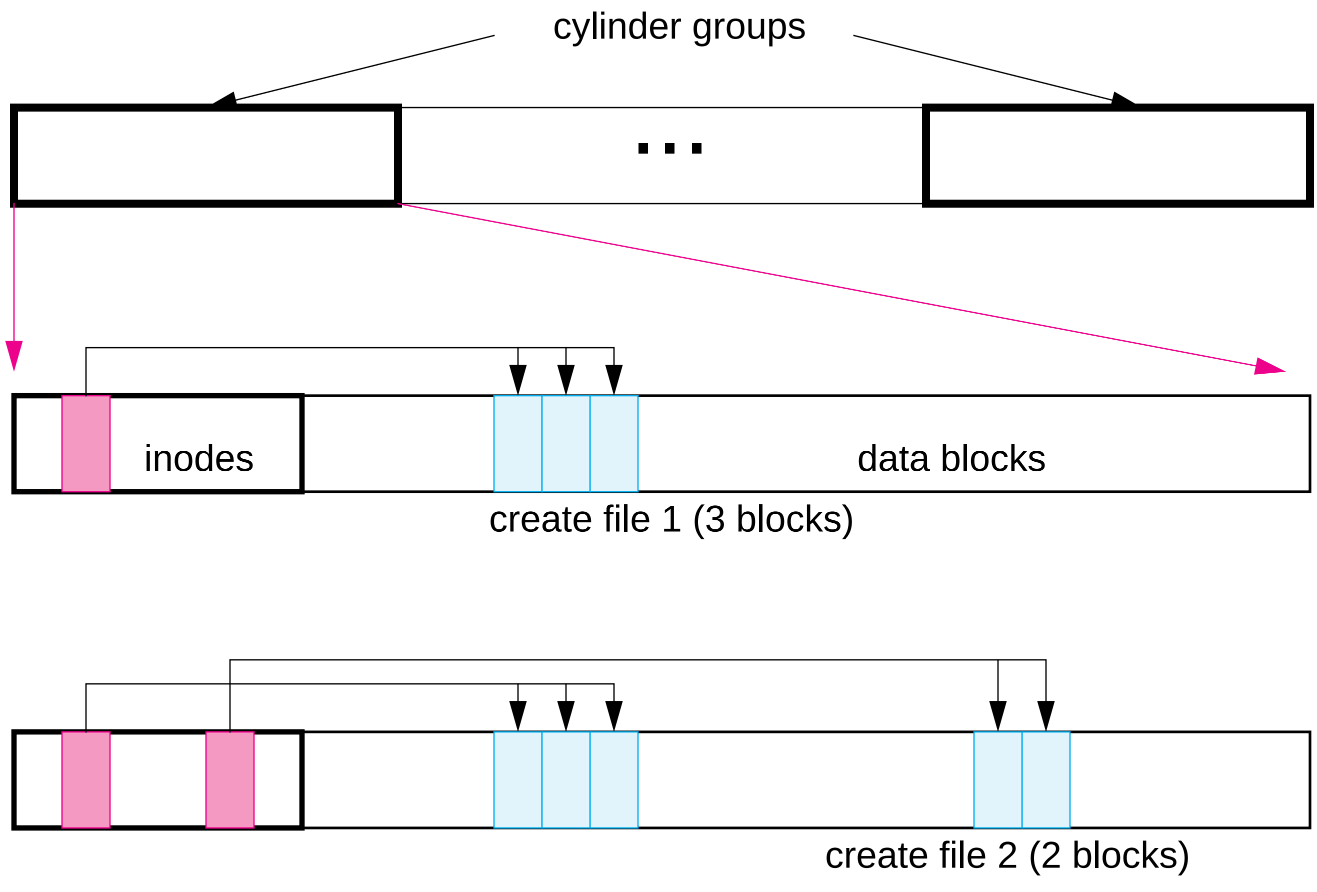
OVERVIEW

Allocation (LFS)



OVERVIEW

Allocation (FFS)



OVERVIEW

Log-Structured File Systems

Beating the I/O Bottleneck: A Case for Log-Structured File Systems

John Ousterhout, Fred Douglis
Operating Systems Review
January 1989

- **Make all writes sequential.**
- **Avoid synchronous operations.**
- **Use garbage collection to reclaim space.**
- **Use database recovery techniques.**

Outline

- An Overview of Log-Structured File Systems
- BSD-LFS Design
- Data Structures
- Clustering in the Fast File System
- Performance
- Conclusions

Project

- **This is work done at Berkeley with the Computer Systems Research Group.**
- **Collaborators:**

    Keith Bostic

    Kirk McKusick

    Carl Staelin

4.4BSD-LFS

Design, Implementation & Performance

Margo Seltzer

Harvard University

Division of Applied Sciences