



**TECHNISCHE
UNIVERSITÄT
DRESDEN**

Fakultät Informatik Institut für Systemarchitektur, Professur für Betriebssysteme

INFLUENTIAL WORK ON FILE SYSTEMS

CARSTEN WEINHOLD

Introduction and Terminology

A Fast File System for UNIX

Marshall Kirk McKusick, William N. Joy, Samuel J. Leffer, Robert S. Fabry
ACM Transactions on Computer Systems (TOCS), Vol. 2 Issue 3, Aug. 1984

Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem

Marshall Kirk McKusick, Gregory Ganger
Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference

The Design and Implementation of a Log-structured File System

Mendel Rosenblum, John K. Ousterhout
Proceedings of the 1991 Symposium on Operating System Principles (SOSP)

A Case for Redundant Arrays of Inexpensive Disks (RAID)

David A. Patterson, Garth Gibson, Randy H. Katz
SIGMOD '88 Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data

Introduction and Terminology

A Fast File System for UNIX

Marshall Kirk McKusick, William N. Joy, Samuel J. Leffer, Robert S. Fabry
ACM Transactions on Computer Systems (TOCS), Vol. 2 Issue 3, Aug. 1984

Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem

Marshall Kirk McKusick, Gregory Ganger
Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference

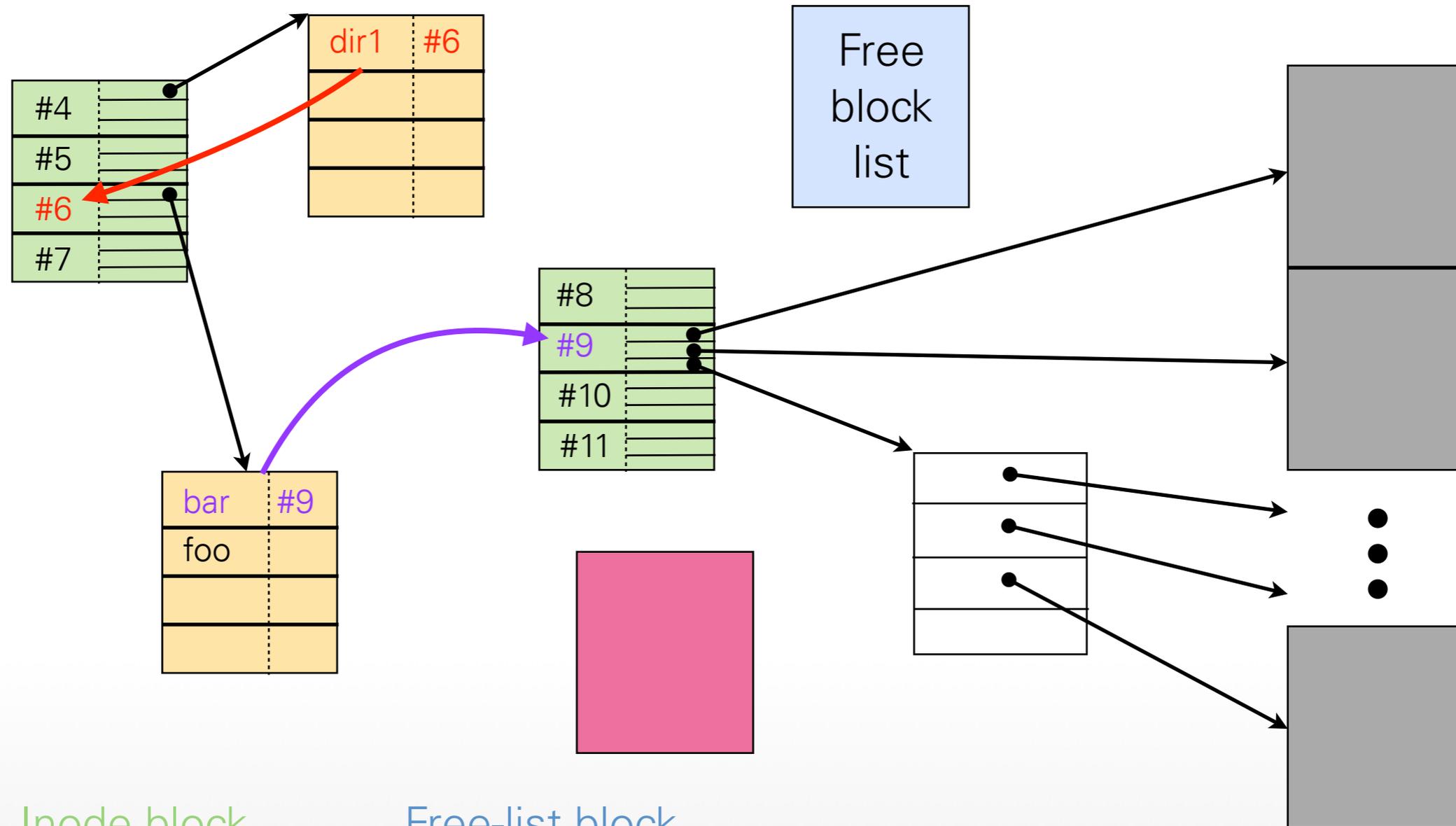
The Design and Implementation of a Log-structured File System

Mendel Rosenblum, John K. Ousterhout
Proceedings of the 1991 Symposium on Operating System Principles (SOSP)

A Case for Redundant Arrays of Inexpensive Disks (RAID)

David A. Patterson, Garth Gibson, Randy H. Katz
SIGMOD '88 Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data

- Mapping of high-level objects (files) to storage locations (blocks)
- Mapping encoded in metadata:
 - Hierarchy of directories: directory entries map filenames to inodes
 - Inodes contain file attributes and pointers
 - Pointers specify blocks with file content
 - Status of inodes and blocks: free / in use
 - Metadata is stored in blocks as well

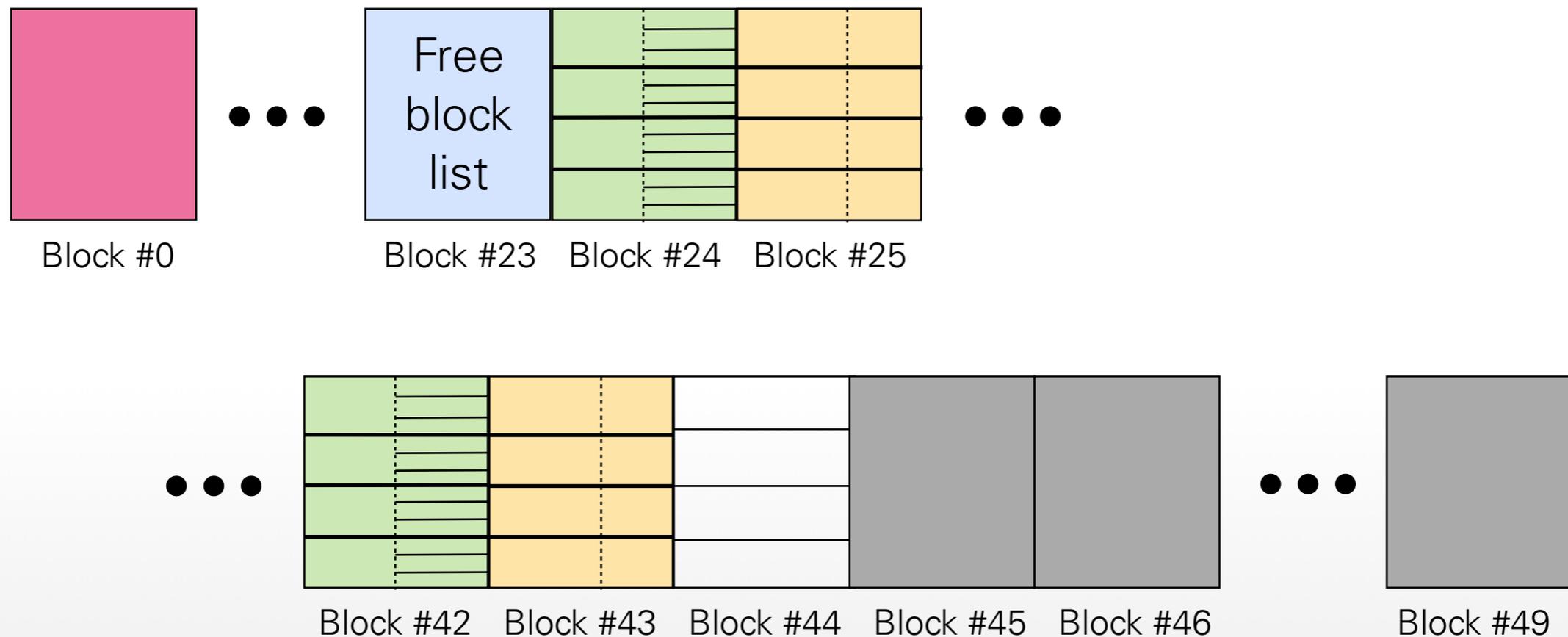


Inode block
 Directory block
 Indirect block

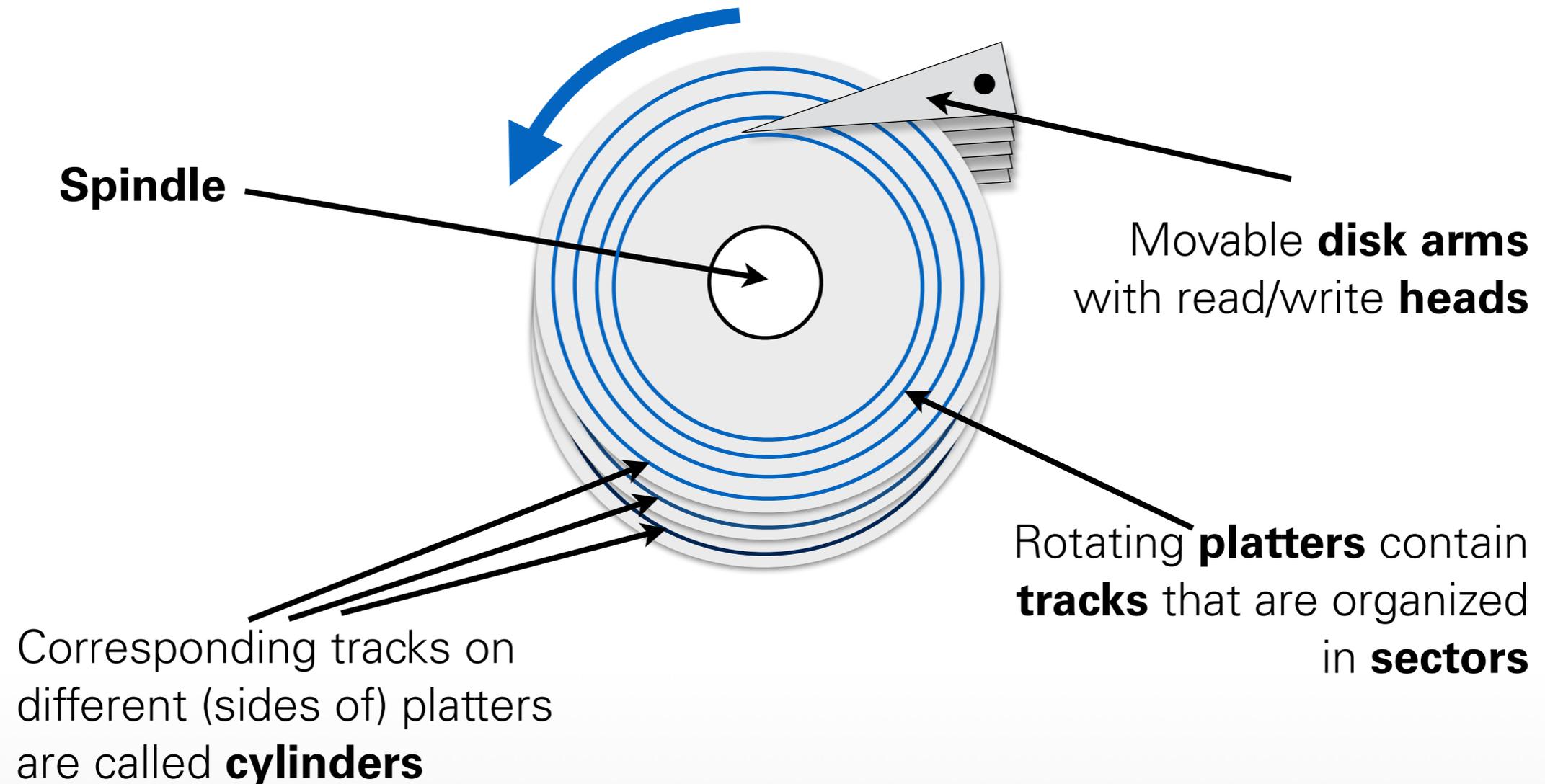
Free-list block
 Data block
 Super block

Blocks on disk are addressed by block number

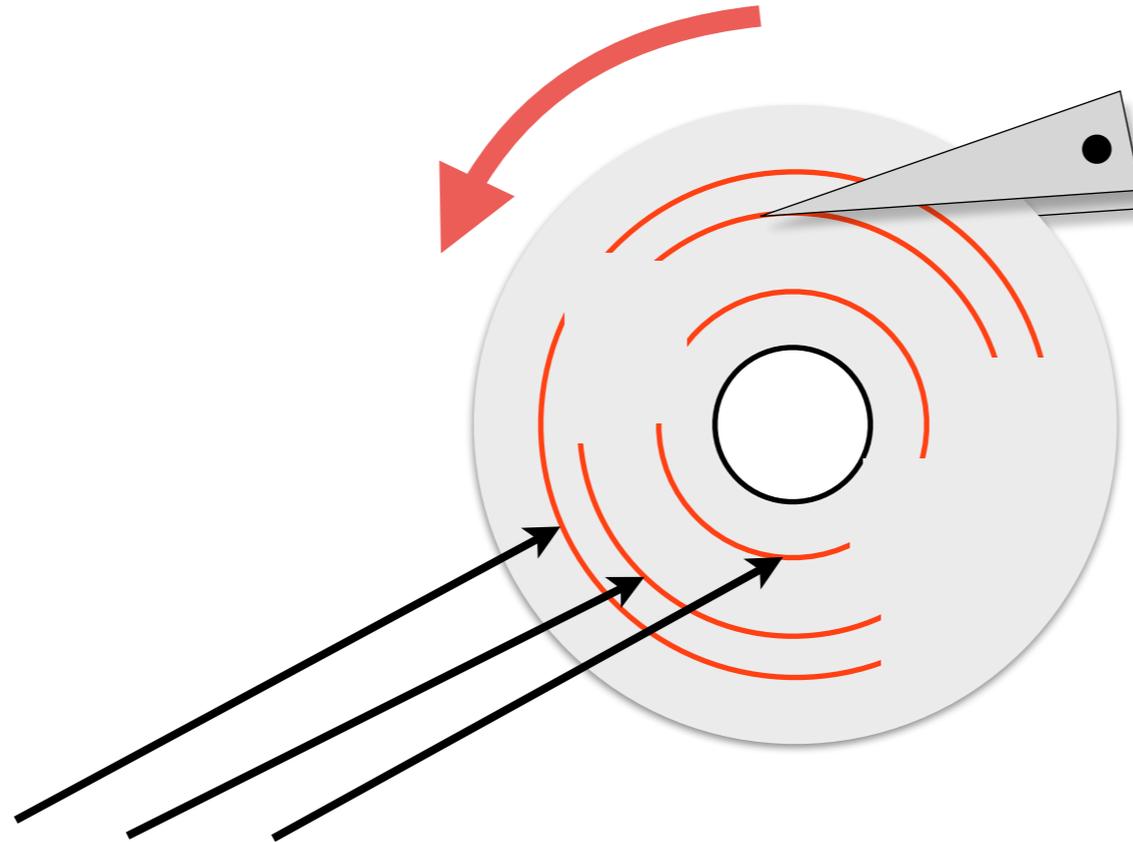
Data transfers of file-system data structures is done in blocks



DISK ORGANIZATION

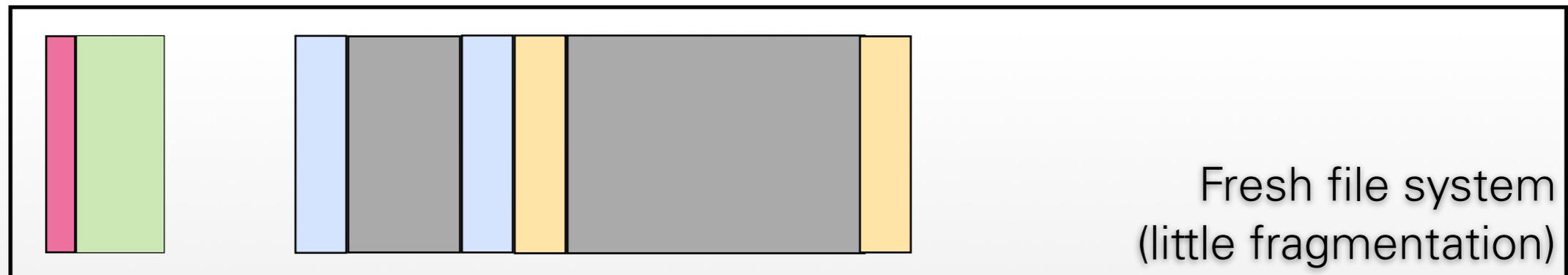


Disk space can also be **partitioned** into consecutive groups of cylinders (each partition can host one file system)



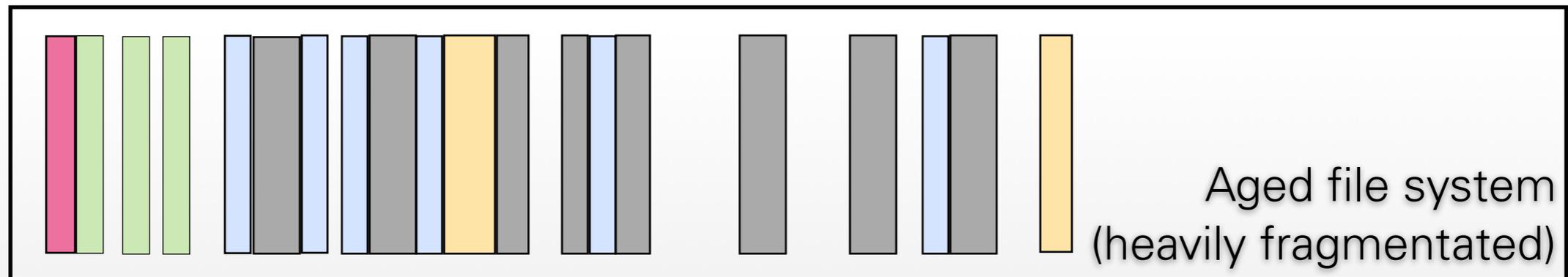
- **Inter-dependent data** and **metadata** often spread across different sectors, tracks, or cylinders
- **Read** and **write operations** can cause **disk-arm movement** to position **heads** over tracks
- **Rotational delay** until requested sectors pass under head

- Small block size (512 bytes, later 1024 bytes^[1])
- Free-block tracking based on linked list
- Fixed inode area at start of partition (e.g., first 4 MB of 150 MB partition reserved for inodes)



Super Block / Inode / Directory block / Free-block linked list / Data Block

- Fragmentation: 80 percent performance loss
- Long seeks between inode and blocks
- Inodes of files in same dir not grouped
- Small block size causes low throughput



Super Block / Inode / Directory block / Free-block linked list / Data Block

Introduction and Terminology

A Fast File System for UNIX

Marshall Kirk McKusick, William N. Joy, Samuel J. Leffer, Robert S. Fabry
ACM Transactions on Computer Systems (TOCS), Vol. 2 Issue 3, Aug. 1984

Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem

Marshall Kirk McKusick, Gregory Ganger
Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference

The Design and Implementation of a Log-structured File System

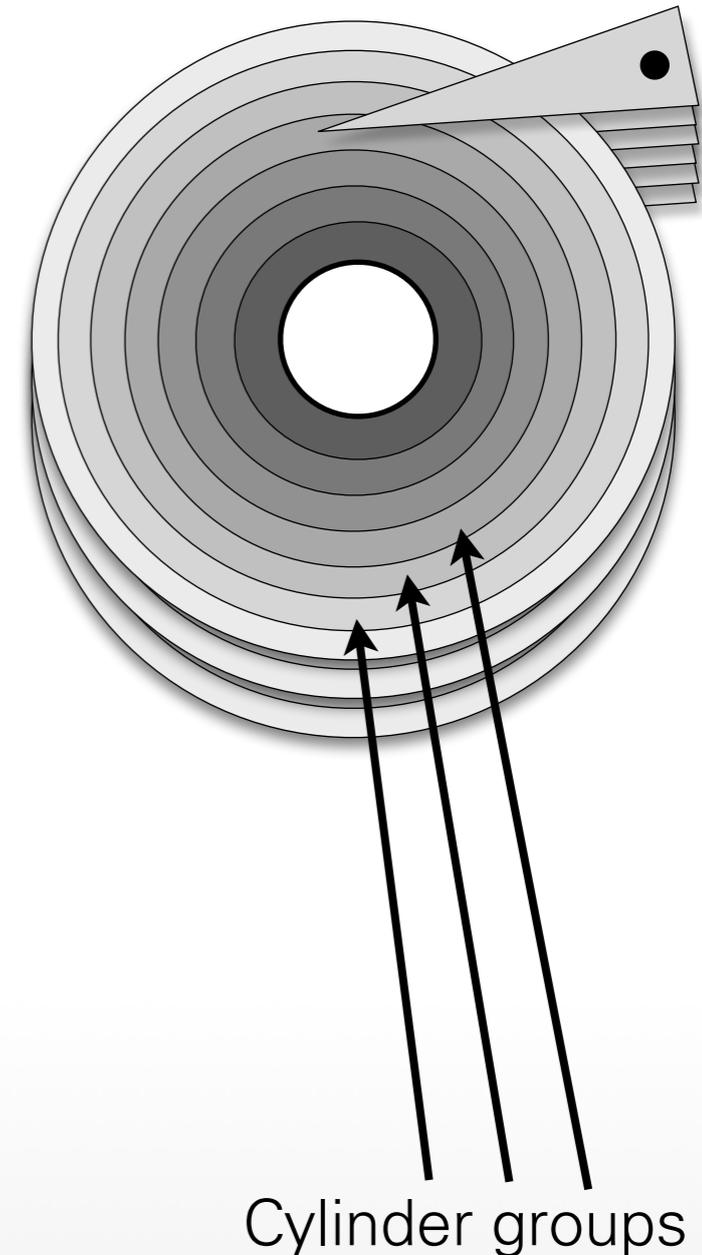
Mendel Rosenblum, John K. Ousterhout
Proceedings of the 1991 Symposium on Operating System Principles (SOSP)

A Case for Redundant Arrays of Inexpensive Disks (RAID)

David A. Patterson, Garth Gibson, Randy H. Katz
SIGMOD '88 Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data

- Goals and achievements of FFS:
 - Stable performance also for aged file systems
 - Parameterization to adapt to different hardware
 - New functionality
- Lives on in *BSD (still named UFS)
- Linux Ext2 is an incompatible clone of FFS

- FFS used **larger block size**
(configurable, min. 4096 bytes)
- FFS introduced **cylinder groups**:
 - Much smaller than partition
 - Fewer potentially long seeks:
 - **Array of inodes**, closer to data
 - **Allocation bitmap** for blocks
(cylinder group is preferred location for files in same dir)
 - Redundant **bookkeeping** information on **different platters**



- Larger blocks and inodes in each cylinder group enable faster throughput
- Problem to avoid: internal fragmentation (small files may not fill up entire blocks)

Space used	% waste	Organization
775.2 Mb	0.0	Data only, no separation between files
807.8 Mb	4.2	Data only, each file starts on 512 byte boundary
828.7 Mb	6.9	Data + inodes, 512 byte block UNIX file system
866.5 Mb	11.8	Data + inodes, 1024 byte block UNIX file system
948.5 Mb	22.4	Data + inodes, 2048 byte block UNIX file system
1128.3 Mb	45.6	Data + inodes, 4096 byte block UNIX file system

Disk space used for real file system on a (back then) contemporary time-sharing system [1]

- FFS introduces **fragments**:
 - Split blocks into smaller units (512, 1024, ... bytes)
 - Free space in cylinder group tracked at fragment level using bitmap:

Bits in map	XXXX	XX00	00XX	0000
Fragment numbers	0-3	4-7	8-11	12-15
Block numbers	0	1	2	3

Allocation bits (example from [1])

- **In example:** block 3 available, block 1 and 2 hold fragments only
- Also: **fewer, larger blocks** require **fewer pointers** in intermediate blocks

- **Inode allocation:**
 - Directory inodes: distribute across cylinder groups
 - File inode: same cylinder group as parent dir
- **Clustering** reduces I/O:
 - At most 16 disk transfers for maximum of 2048 inodes per cylinder group
 - Old Unix file system: one disk request for each inode in the worst case

- **Global layout policy:**
 - Put blocks in same cylinder group as inode
 - Large files: after first indirect block (and then after each MB) put blocks into other cylinder group
- **Local layout policy:**
 - 1) Rotationally closest block location, same cylinder (skip over n blocks to minimize rotational delay, e.g., due to slow CPU, interrupt handling, ...)
 - 2) Same cylinder group
 - 3) Other cylinder group (find via quadratic hashing)
 - 4) Exhaustive search in all cylinder groups
- **Reduce fragmentation:** 5% block reserve (super user only)



PERFORMANCE (IN 1984)

Type of File System	Processor and Bus Measured	Speed	Read Bandwidth	% CPU
old 1024	750/UNIBUS	29 Kbytes/sec	29/983 3%	11%
new 4096/1024	750/UNIBUS	221 Kbytes/sec	221/983 22%	43%
new 8192/1024	750/UNIBUS	233 Kbytes/sec	233/983 24%	29%
new 4096/1024	750/MASSBUS	466 Kbytes/sec	466/983 47%	73%
new 8192/1024	750/MASSBUS	466 Kbytes/sec	466/983 47%	54%

Read performance of FFS vs its predecessor [1]

Type of File System	Processor and Bus Measured	Speed	Write Bandwidth	% CPU
old 1024	750/UNIBUS	48 Kbytes/sec	48/983 5%	29%
new 4096/1024	750/UNIBUS	142 Kbytes/sec	142/983 14%	43%
new 8192/1024	750/UNIBUS	215 Kbytes/sec	215/983 22%	46%
new 4096/1024	750/MASSBUS	323 Kbytes/sec	323/983 33%	94%
new 8192/1024	750/MASSBUS	466 Kbytes/sec	466/983 47%	95%

Write performance of FFS vs its predecessor [1]

- **Long filenames:** variable-length dentries, think
`struct { int inode; int size; int nlen; char name[0]; }`
- **Advisory file locking**
- **Symbolic links**
- **Rename**
- **Quotas**

Introduction and Terminology

A Fast File System for UNIX

Marshall Kirk McKusick, William N. Joy, Samuel J. Leffer, Robert S. Fabry
ACM Transactions on Computer Systems (TOCS), Vol. 2 Issue 3, Aug. 1984

Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem

Marshall Kirk McKusick, Gregory Ganger
Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference

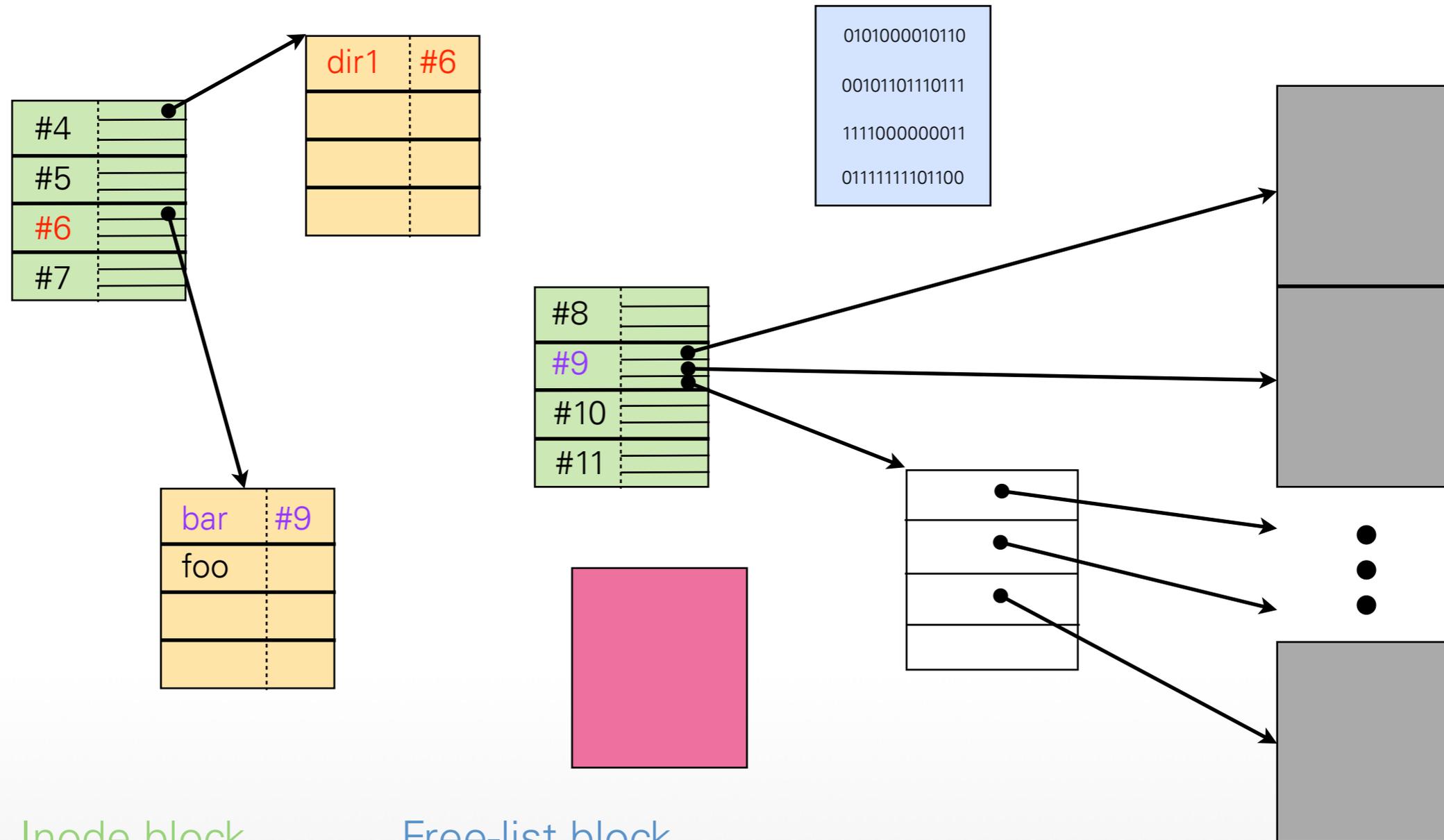
The Design and Implementation of a Log-structured File System

Mendel Rosenblum, John K. Ousterhout
Proceedings of the 1991 Symposium on Operating System Principles (SOSP)

A Case for Redundant Arrays of Inexpensive Disks (RAID)

David A. Patterson, Garth Gibson, Randy H. Katz
SIGMOD '88 Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data

- FFS in 1984: **asynchronous writes**^[1], but ...
- ... **Interdependencies** in file system structures:
 - Directory entries point to inodes
 - Inodes point to data blocks or indirect blocks
 - Bitmaps mark inodes or blocks as free / in use
- High-level operations often require multi-block updates (e.g., inode, dentry, bitmap, ...)
- Partial updates can cause inconsistencies

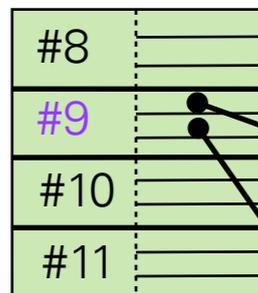
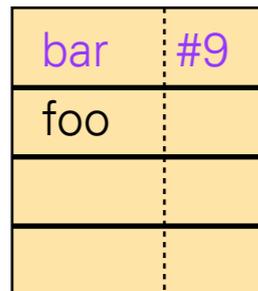
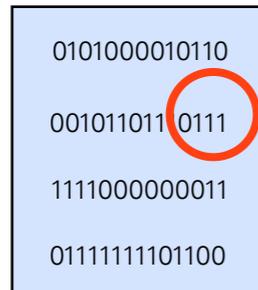


Inode block
 Directory block
 Indirect block

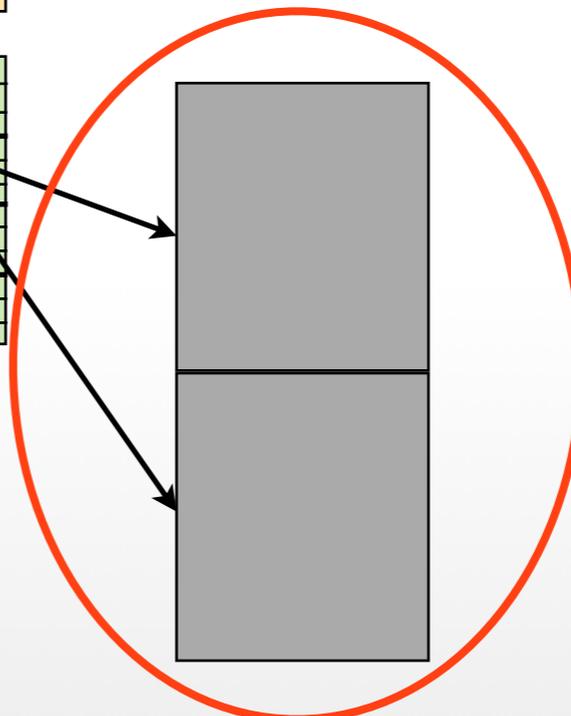
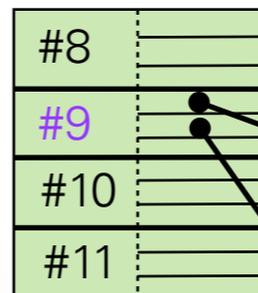
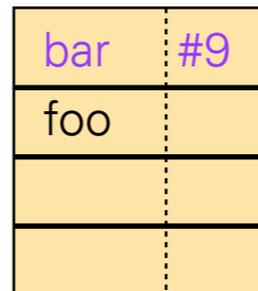
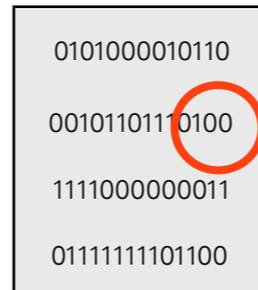
Free-list block
 Data block
 Super block

INVALID BLOCK BITMAP

Buffer Cache



Hard Disk



Inconsistency: data blocks, inode, and directory block have been written, but allocation info in bitmap block is out of data

Risk of data loss due to overwriting!

CRASH

Inode Block / Directory block / Indirect Block / Bitmap Block / Data Block

Buffer Cache

```

0101000010110
00101101110111
1111000000011
01111111101100
    
```

bar	#9
foo	

#8	
#9	
#10	
#11	



Hard Disk

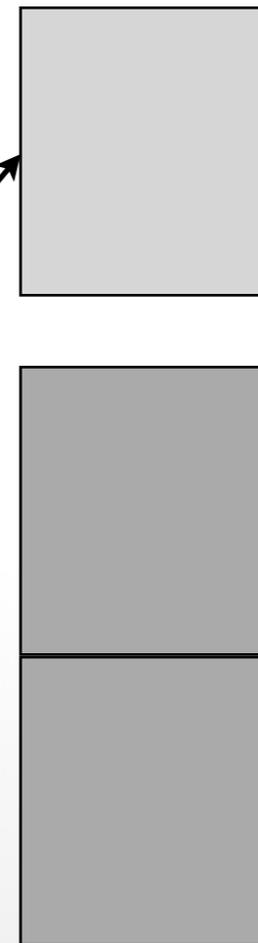
```

0101000010110
00101101110111
1111000000011
01111111101100
    
```

bar	#9
foo	

#8	
#9	
#10	
#11	

CRASH



Inconsistency:
bitmap, directory,
and data blocks
written, but inode
block not updated.

**Wrong/deleted
file is referenced!**

In-use / free
status of blocks is
incorrect.

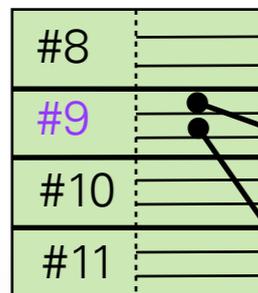
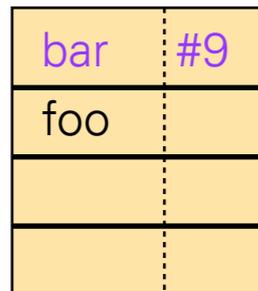
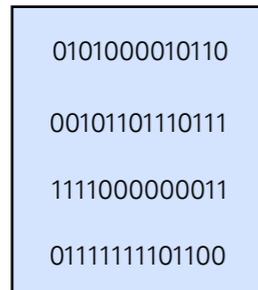
Inode Block / Directory block / Indirect Block / Bitmap Block / Data Block

- **Critical:** loss of previously existing data
 - Pointers to wrong inodes or data blocks containing data of other (or deleted) files
 - Used block or inode marked as free
 - Inode's reference counter too small
- **Non-critical:** temporary resource leaks
 - Free block or inode marked as used
 - Inode's reference counter too high
 - Data block (*or inode*) already persistent, but block pointer (*or directory entry*) not written yet

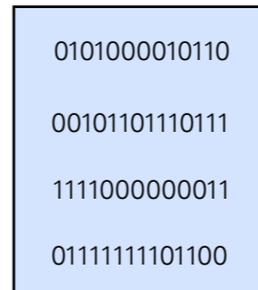
- **Valid pointers:** do not point to a data structure that has not been initialized yet
- **Reuse:** do not reuse a data structure, unless all old pointers to it have been invalidated
- **Reachability:** do not invalidate a pointer to a valid resource, before a new pointer to it has been set

SAFE WRITE ORDER

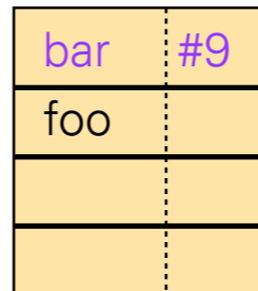
Buffer Cache



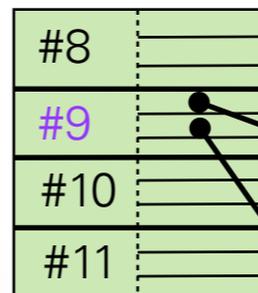
Hard disk



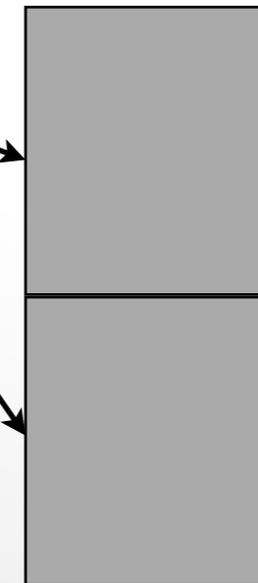
(1)a



(3)



(2)



(1)b

(1)c

Order of writes:

(1) Bitmap and data blocks (a-c)

[Write Barrier]

(2) Inode block

[Write Barrier]

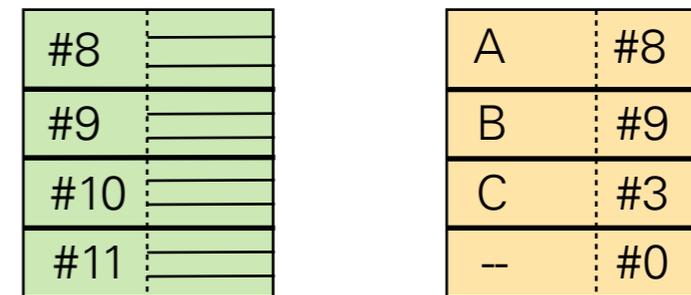
(3) Directory block

Inode Block / Directory block / Indirect Block / Bitmap Block / Data Block

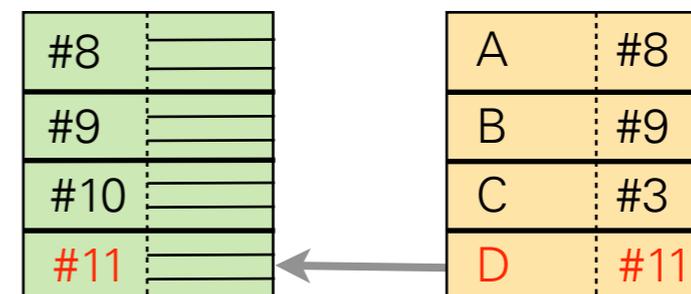
- **Idea:** group individual phases of large number of safe-write sequences into large batches
 - Accumulate modifications in buffer cache
 - When writing, takes all dependencies among blocks into account (serialize writes into batches)
- **Implementation:**
 - Dependency structures for each cache block
 - Metadata (e.g., directory -> inode)
 - Data (inode / indirect block -> data block)
 - Problem: cycles in dependency graph

- State of inode and directory blocks in buffer cache:
 - a) No dependency
 - b) Inode block must be initialized before writing directory block
 - c) Inode pointer in directory entry must be invalidated before inode can be freed
- Cyclic dependency!

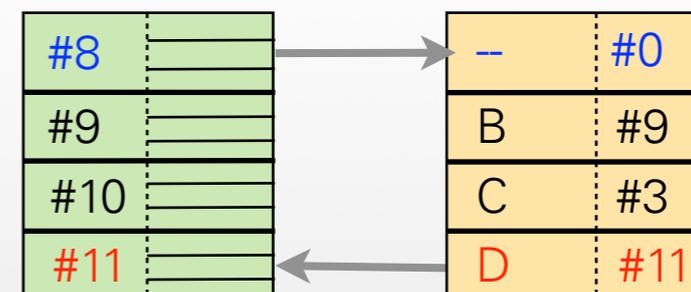
a) No modification



b) File D created



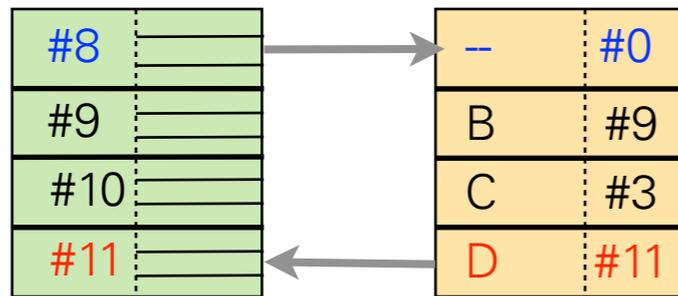
c) File A deleted



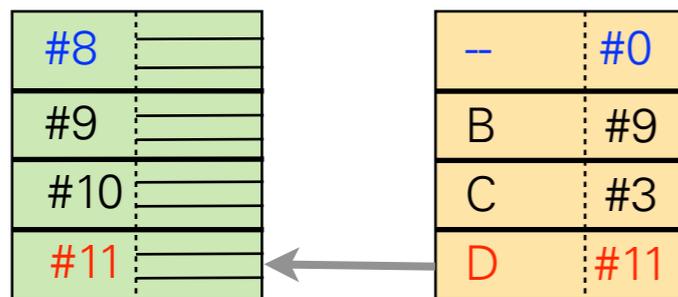
Source: [2]

BREAKING THE CYCLCLE

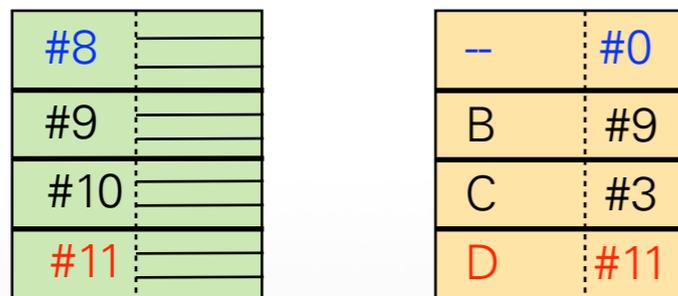
Buffer Cache



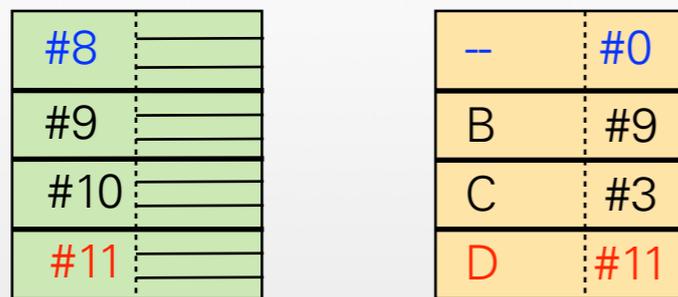
a) Metadata blocks modified in cache



b) Directory block written (but without new dir entry for D)

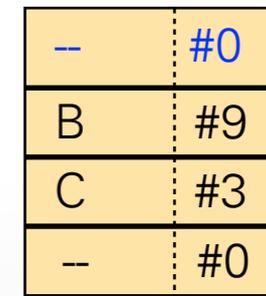
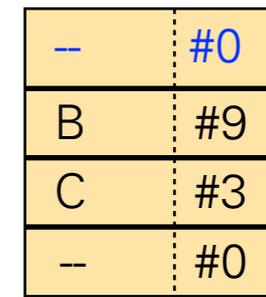
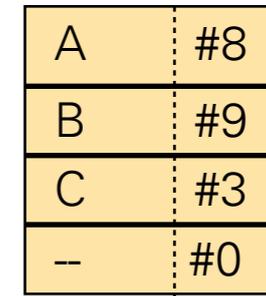


c) Inode block written



d) Directory block written again, this time including D

Hard disk



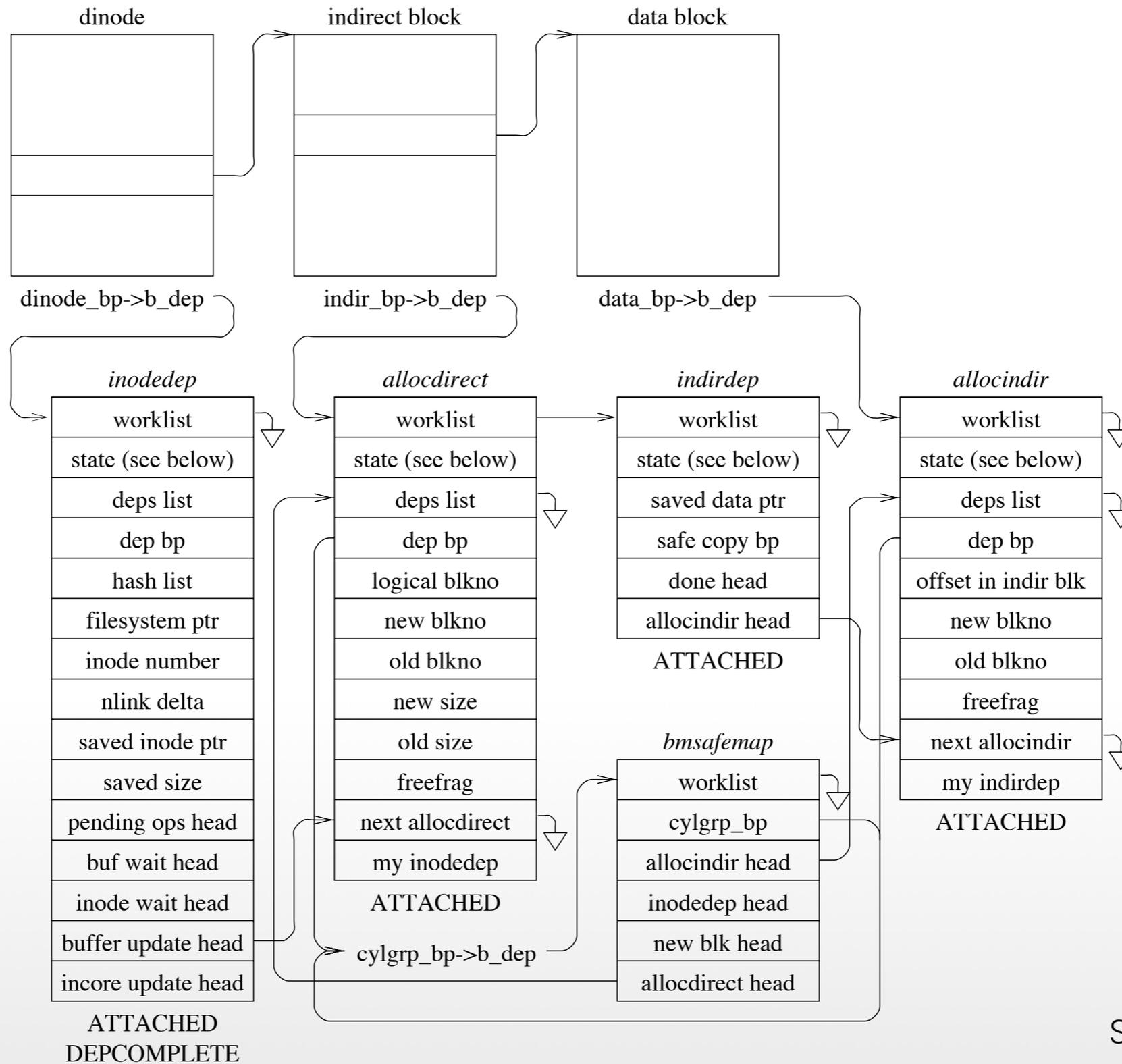
Source: [2]

- Goal: can write modified blocks at any time
- Soft-updates code inspects dependency structures before write operation:
 - Consistent / isolated modifications can be written immediately without side effects
 - Modifications with unsatisfied dependencies are rolled back temporarily for block-write operation
 - When consistent version of block is been written, re-apply temporarily removed modifications
- There is a dependency structure for each block modification (chained up in list linked at buffer head)

Name	Function	Associated Structures
bmsafemap	track bitmap dependencies (points to lists of dependency structures for recently allocated blocks and inodes)	cylinder group block
inodedep	track inode dependencies (information and list head pointers for all inode-related dependencies, including changes to the link count, the block pointers, and the file size)	inode block
allocdirect	track inode-referenced block (linked into lists pointed to by an inodedep and a bmsafemap to track inode's dependence on the block and bitmap being written to disk)	data block or indirect block or directory block
indirdep	track indirect block dependencies (points to list of dependency structures for recently-allocated blocks with pointers in the indirect block)	indirect block
allocindir	track indirect block-referenced block (linked into lists pointed to by an indirdep and a bmsafemap to track the indirect block's dependence on that block and bitmap being written to disk)	data block or indirect block or directory block
pagedep	track directory block dependencies (points to lists of diradd and dirrem structures)	directory block
diradd	track dependency between a new directory entry and the referenced inode	inodedep and directory block
mkdir	track new directory creation (used in addition to standard diradd structure when doing a mkdir)	inodedep and directory block
dirrem	track dependency between a deleted directory entry and the unlinked inode	first pagedep then tasklist
freefrag	tracks a single block or fragment to be freed as soon as the corresponding block (containing the inode with the now-replaced pointer to it) is written to disk	first inodedep then tasklist
freeblks	tracks all the block pointers to be freed as soon as the corresponding block (containing the inode with the now-zeroed pointers to them) is written to disk	first inodedep then tasklist
freefile	tracks the inode that should be freed as soon as the corresponding block (containing the inode block with the now-reset inode) is written to disk	first inodedep then tasklist

Source: [2]

UPDATE DEPENDENCIES



- **ATTACHED:**

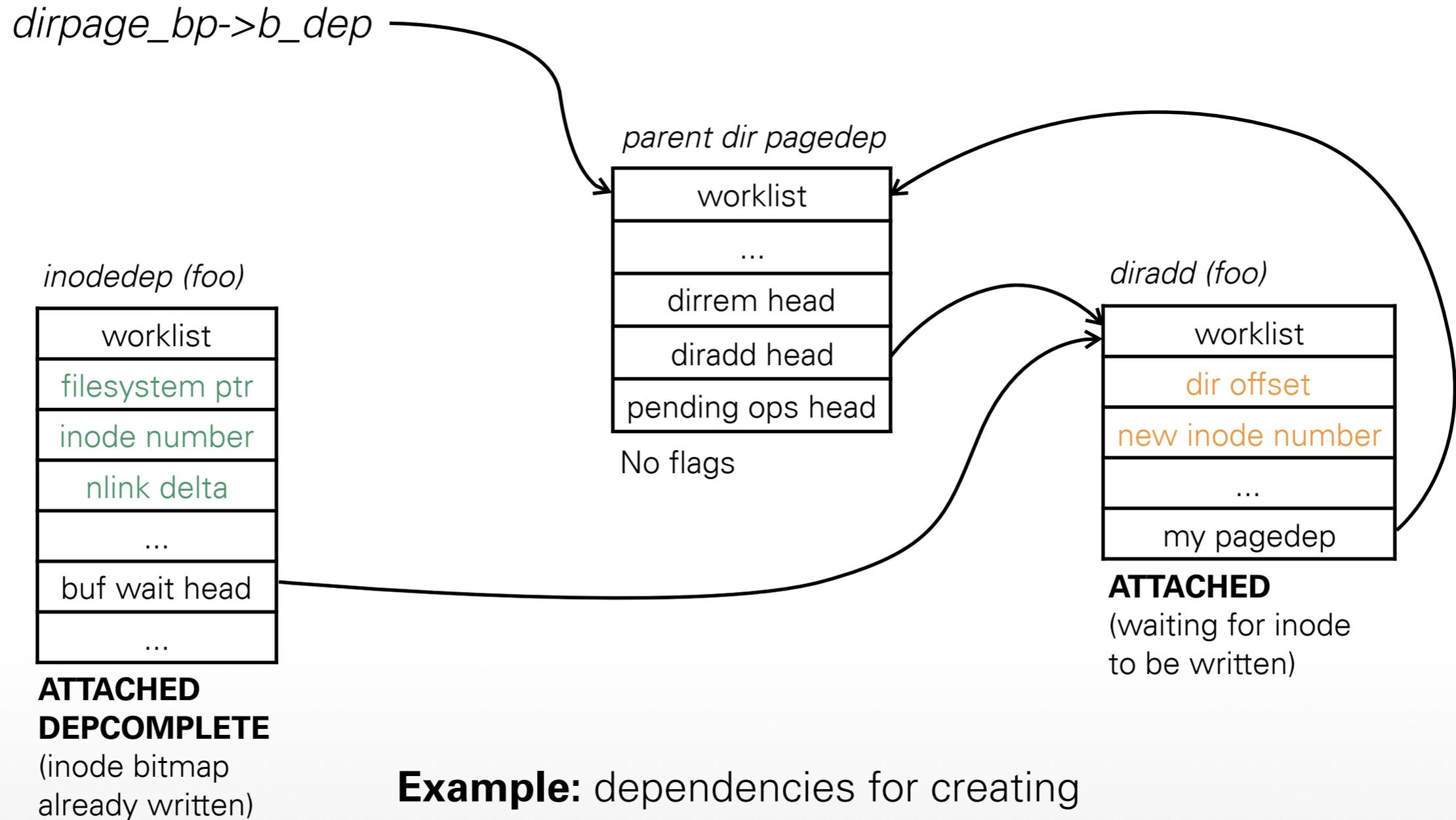
- Buffer is not being written at the moment
- For rollback: clear ATTACHED flag to indicate that modification must be re-applied after write

- **DEPCOMPLETE:**

- Modification safe for writing, no need to roll back

- **COMPLETE:**

- Modification has been written to disk
- Deallocation: only after all flags are set



Example: dependencies for creating a new directory entry (simplified version of presentation in [2])

Software development benchmark:

- 75 percent fewer writes
- 21 percent shorter run time

Filesystem Configuration	Disk Writes		Running Time
	Sync	Async	
Normal	162,410	39,924	2hr, 12min
Asynchronous	0	38,262	1hr, 44min
Soft Updates	1124	48,850	1hr, 44min

Time to build and install FreeBSD [2]

Filesystem Configuration	Disk Writes	
	Sync	Async
Normal	1,877,794	1,613,465
Soft Updates	118,102	946,519

Mail server operation [2]

Mail server workload:

- Writes per second dropped from 40 to 12 per send (with sync:async ratio of 1:1 and 1:8, respectively)
- Mail-handling capacity tripled

- Soft updates were introduced in 4.4BSD
- Huge improvement over synchronous writes
- Still in use today:
 - FreeBSD
 - OpenBSD

Introduction and Terminology

A Fast File System for UNIX

Marshall Kirk McKusick, William N. Joy, Samuel J. Leffer, Robert S. Fabry
ACM Transactions on Computer Systems (TOCS), Vol. 2 Issue 3, Aug. 1984

Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem

Marshall Kirk McKusick, Gregory Ganger
Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference

The Design and Implementation of a Log-structured File System

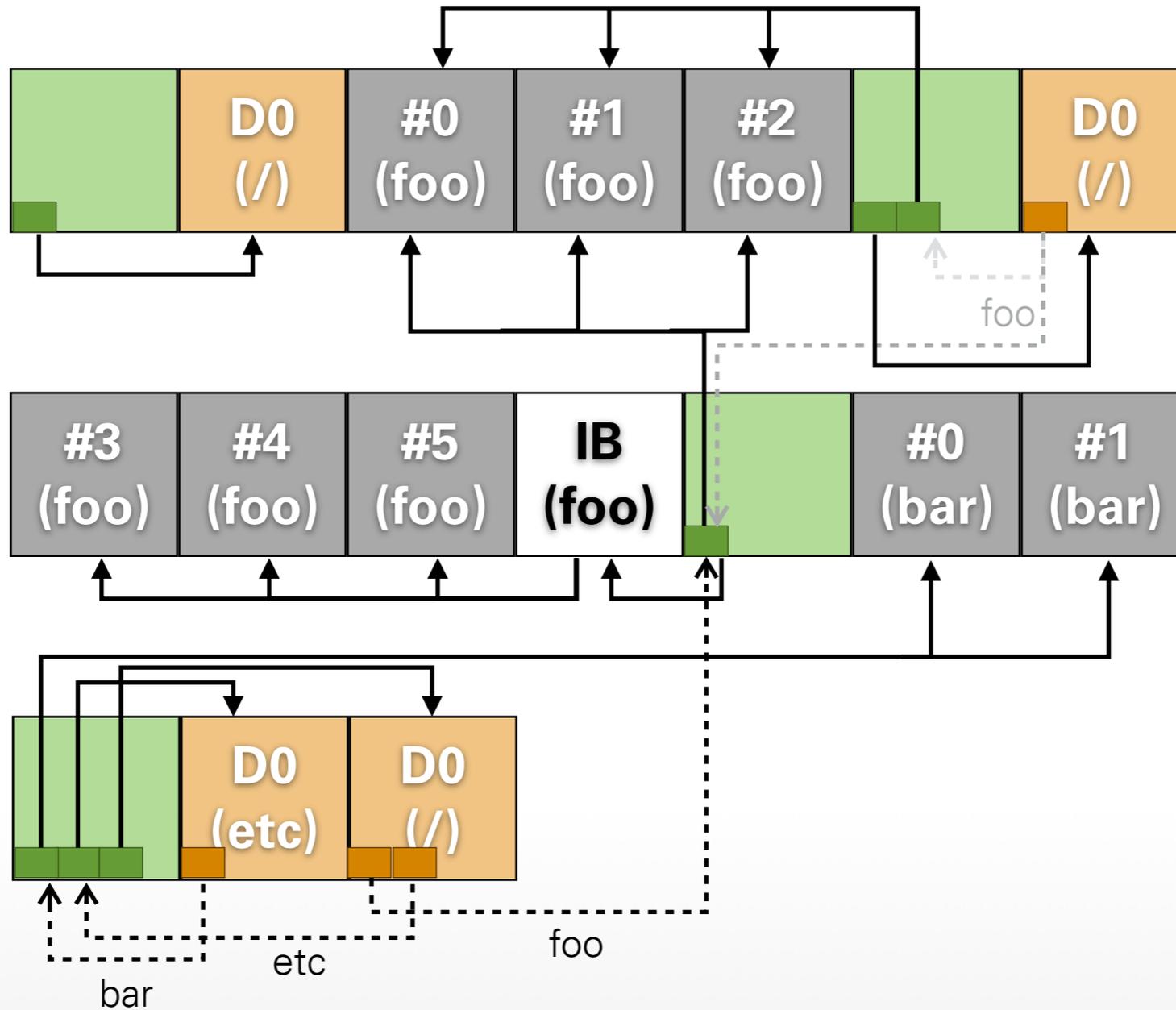
Mendel Rosenblum, John K. Ousterhout
Proceedings of the 1991 Symposium on Operating System Principles (SOSP)

A Case for Redundant Arrays of Inexpensive Disks (RAID)

David A. Patterson, Garth Gibson, Randy H. Katz
SIGMOD '88 Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data

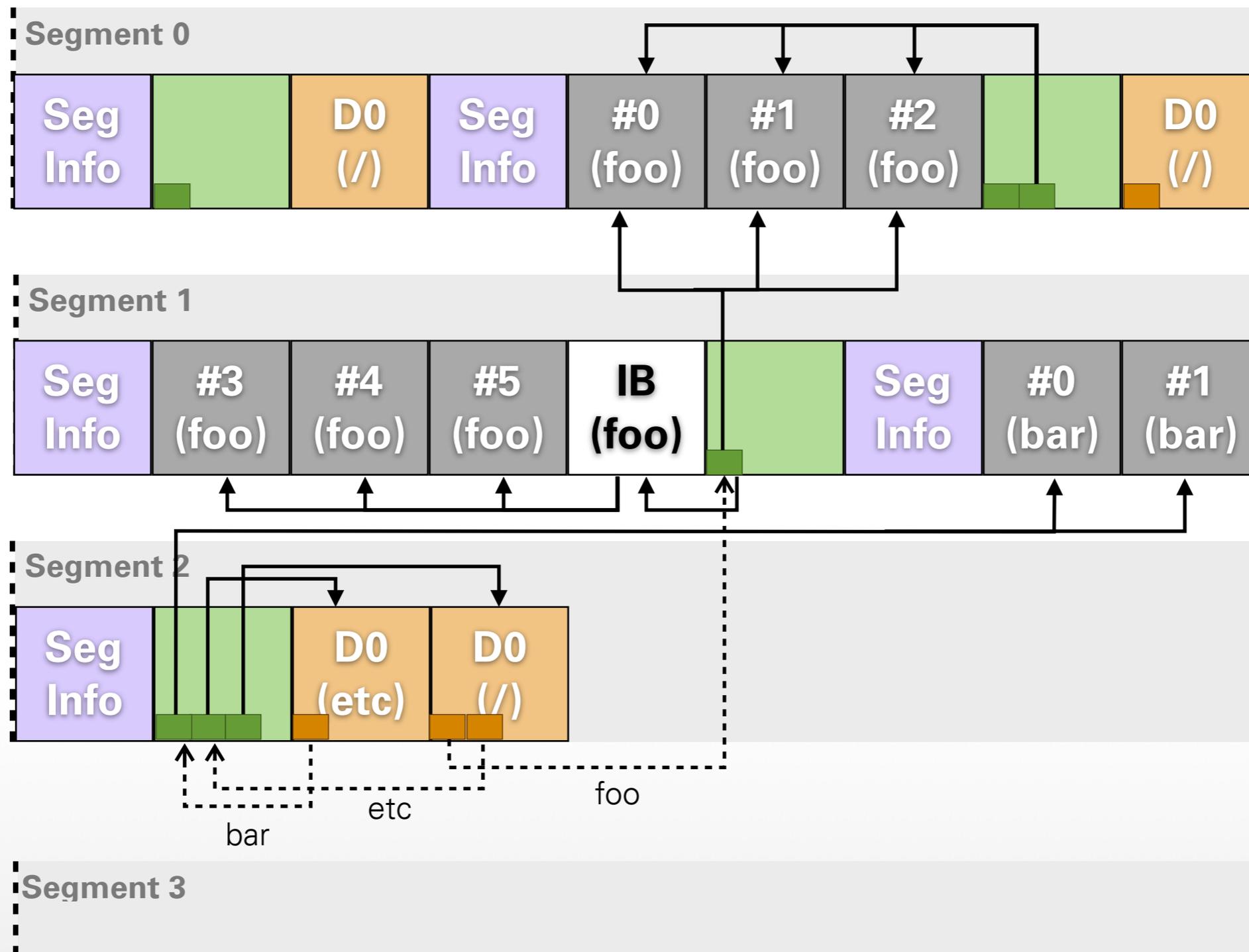
- **Idea:** treat entire file system as a log
 - Write data and metadata blocks linearly
 - No overwriting, append new block versions to log
 - Newest versions of all blocks contain latest file-system state
- Additional features:
 - Log contains consistent versions of old file-system state
 - Multiple snapshots of file-system state possible

LFS OPERATION



Inode Block / Directory Block / Indirect Block / Data Block

SEGMENTS IN AN LFS



Inode Block / Directory Block / Indirect Block / Data Block / Summary Block

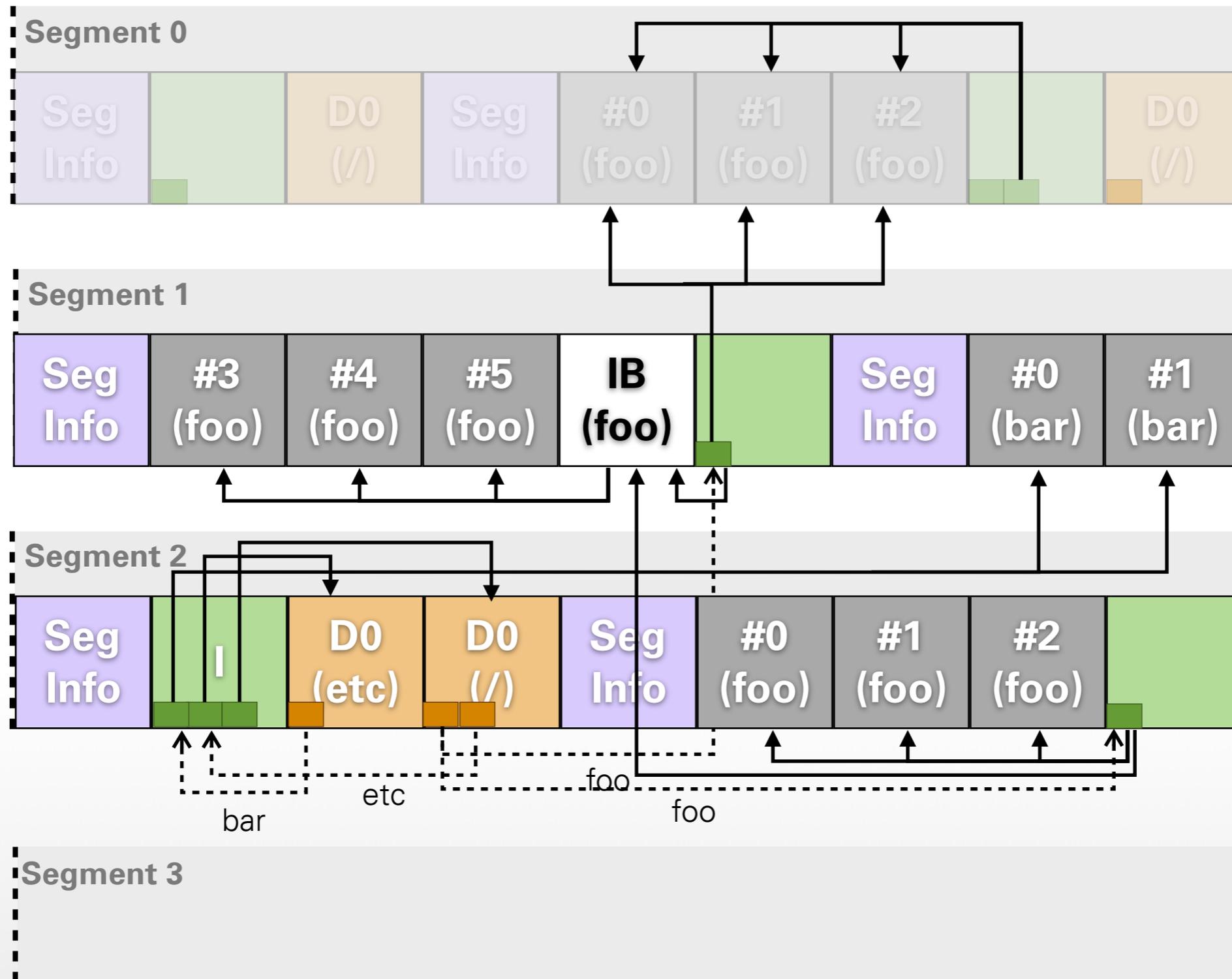
- Block address space portioned into **segments**
- Segments contain **data** and **metadata** blocks
- **Summary blocks** describe block types:
 - File / directory blocks: **(Inode,Block#)**
 - **Inode blocks**: new / modified inodes
 - **Inode-map blocks**: pointers to inodes in log
- Inode map **decouples** inode pointers in directory entries from physical location in log (prevents trickle-up effect up to root directory)

- All data and metadata in log ... **but:** full scan of entire log too expensive
- **Better:** regularly checkpoint current state of metadata in compact form
- **2 checkpoint areas** at fixed position:
 - Consistent version of all blocks from inode map + segment usage table
 - Timestamp + pointer to last written segment
 - Checksum to validate consistency and completeness of checkpoint area

- **Replay from last checkpoint**
 1. Choose latest consistent checkpoint area
 2. Reinitialize inode map in kernel memory
 3. Determine position of last segment written before checkpoint
 4. Roll-forward from last pre-checkpoint segment, redo all changes as found in log
 5. Latest consistent file system has been recovered

- **Problem:** Log cannot grow indefinitely
- **Solution 1:** free old blocks, thread into log
 - (+) Simple and fast free operation
 - (-) Fragmentation grows
- **Solution 2:** clean complete segment
 - (+) Fragmentation does not increase
 - (-) Live blocks in segment must be copied into new segment
- LFS as described in [3] implements solution 2

SEGMENT CLEANING



- **Cleaner process** searches for partially live segments in background
- Starts and stops based on thresholds (low / high watermark of free segments)
- There is no block-allocation table or free list; how to identify obsolete blocks?
 1. **Find pointer** to block in inode or indirect block
 2. **No valid pointer found** -> block is **obsolete**
- Validation efficiency?

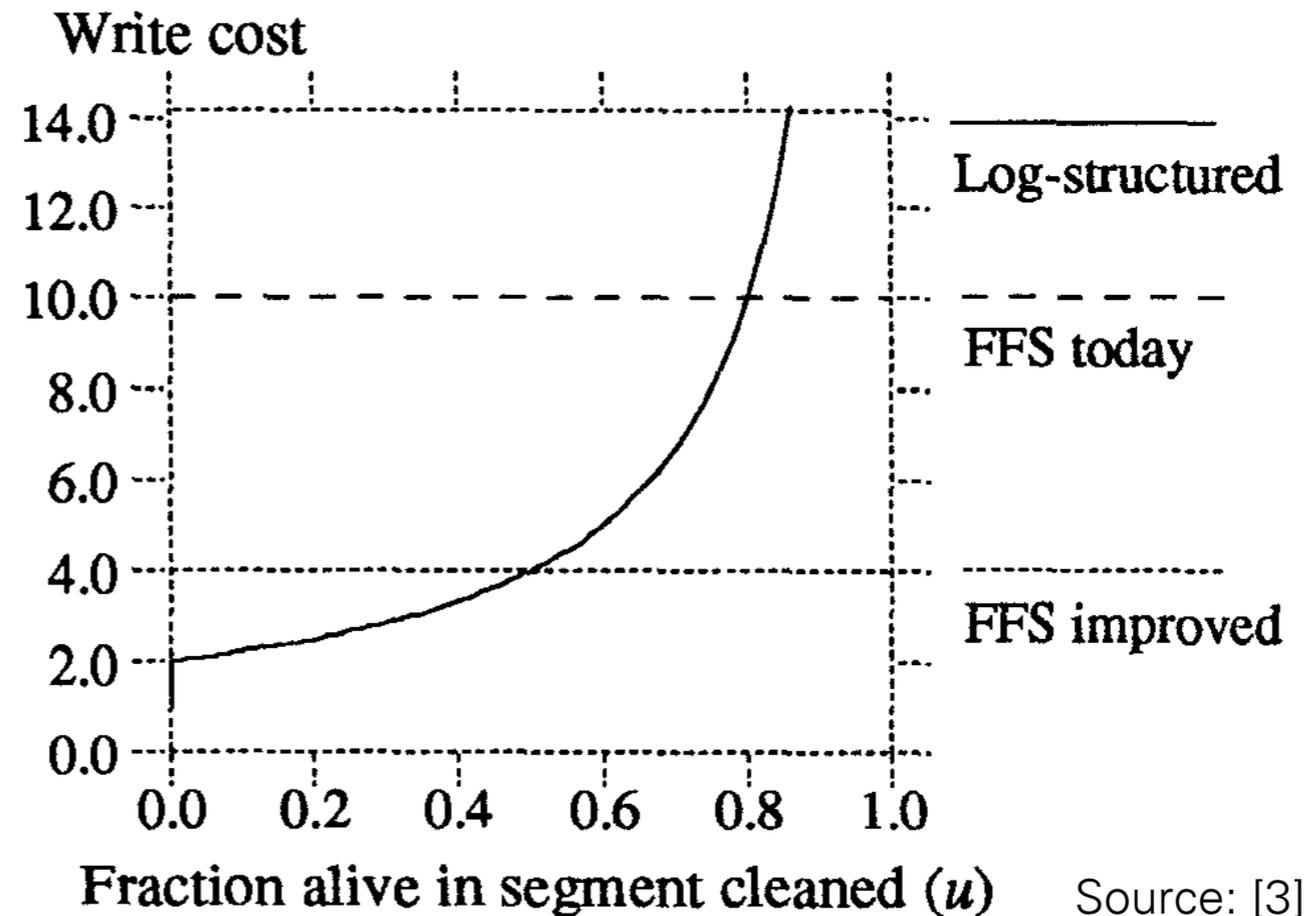
- LFS keeps statistics about each segment:
 - **Versions** of blocks stored in segment
 - Last **modification time** of any block in segment
 - Number of **live bytes** (utilization)
- Blocks containing segment-usage table are written to log (positions put in checkpoint area)
- **Versions numbers:**
 - Compare to inode versions to find obsolete blocks, still need to check indirect blocks
 - Inode versions recorded in inode map

- Segment-usage table helps with cost-benefit cleaning policy
- **Age + utilization** of segment enable classification into **hot** and **cold** segments
- Surprising result:
 - Clean cold segments even with high utilization (e.g., 75%)
 - Leave hot segments alone until utilization drops to much lower value (e.g., 15%)
- Details and simulation results in [3]

SMALL-FILE WRITE COST

Garbage collection must be factored into write cost

Worst case: steady state where garbage collection is constantly required



Source: [3]

$$\begin{aligned}
 \text{write cost} &= \frac{\text{total bytes read and written}}{\text{new data written}} \\
 &= \frac{\text{read segs} + \text{write live} + \text{write new}}{\text{new data written}} \\
 &= \frac{N + N*u + N*(1-u)}{N*(1-u)} = \frac{2}{1-u}
 \end{aligned}$$

Source: [3]

Variables:

- **N** ... number of segments
- **u** ... utilization of segments
- **N*(1-u)** ... free space generated, maximum amount of new data that can be written

- Suitability for different types of storage:
 - **Rotating disks:**
 - Excellent write performance with sufficient amount of clean segments on a minority of storage systems with rotating disks
 - Prone to fragmentation, can hurt read performance
 - **Solid-state / flash-based storage:**
 - Highly suitable, fragmentation has little impact
 - Does not require sophisticated flash-translation layer
- Real-world usage:
 - YAFFS2, JFFS, NILFS2, F2FS, ... for flash
 - Similar algorithms exist in flash translation layers

Introduction and Terminology

A Fast File System for UNIX

Marshall Kirk McKusick, William N. Joy, Samuel J. Leffer, Robert S. Fabry
ACM Transactions on Computer Systems (TOCS), Vol. 2 Issue 3, Aug. 1984

Soft Updates: A Technique for Eliminating Most Synchronous Writes in the Fast Filesystem

Marshall Kirk McKusick, Gregory Ganger
Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference

The Design and Implementation of a Log-structured File System

Mendel Rosenblum, John K. Ousterhout
Proceedings of the 1991 Symposium on Operating System Principles (SOSP)

A Case for Redundant Arrays of Inexpensive Disks (RAID)

David A. Patterson, Garth Gibson, Randy H. Katz
SIGMOD '88 Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data

"Gordon Bell said that between 1974 and 1984, single chip computers improved performance improvement by 40% per year [...]."

"Magnetic disk technology has doubled capacity and halved price every three years in line with the growth rate of semiconductor memory. [...]"

"In contrast to primary memory technologies, the performance of single large expensive magnetic disks (SLED) has improved at a modest rate. These mechanical devices are dominated by the seek and the rotation delays: from 1971 to 1981, the raw seek time for a high-end IBM disk improved by only a factor of two while the rotation time did not change."

Amdahl's Law:

$$S = \frac{1}{(1-f) + f/k}$$

Projection: Application spends 10% of execution time doing I/O, net speedup with faster processors will be:

- 10x faster CPU -> 5x overall speedup
- 100x faster CPU -> 10x overall speedup

Source: [4]

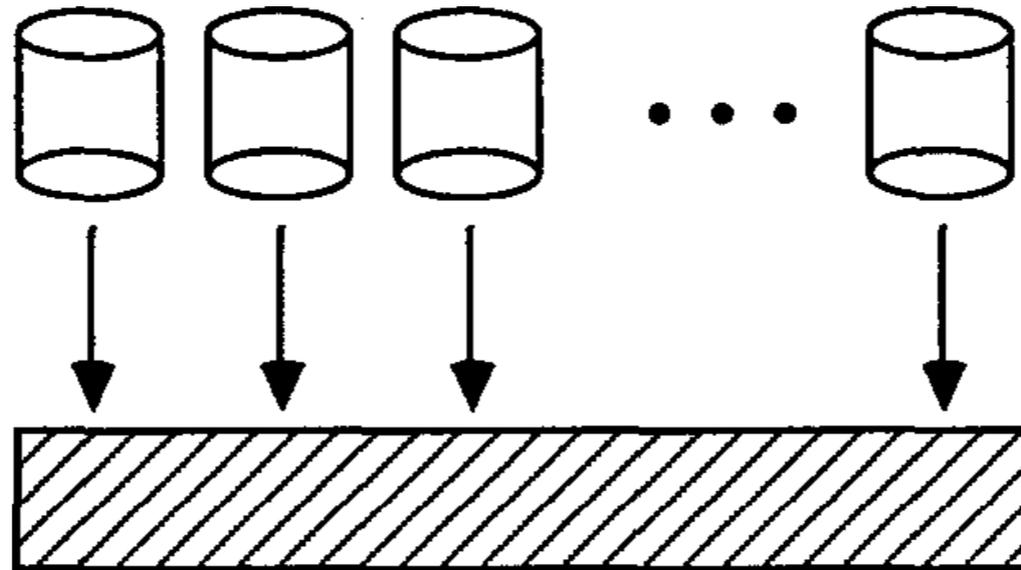


COST AND PERFORMANCE

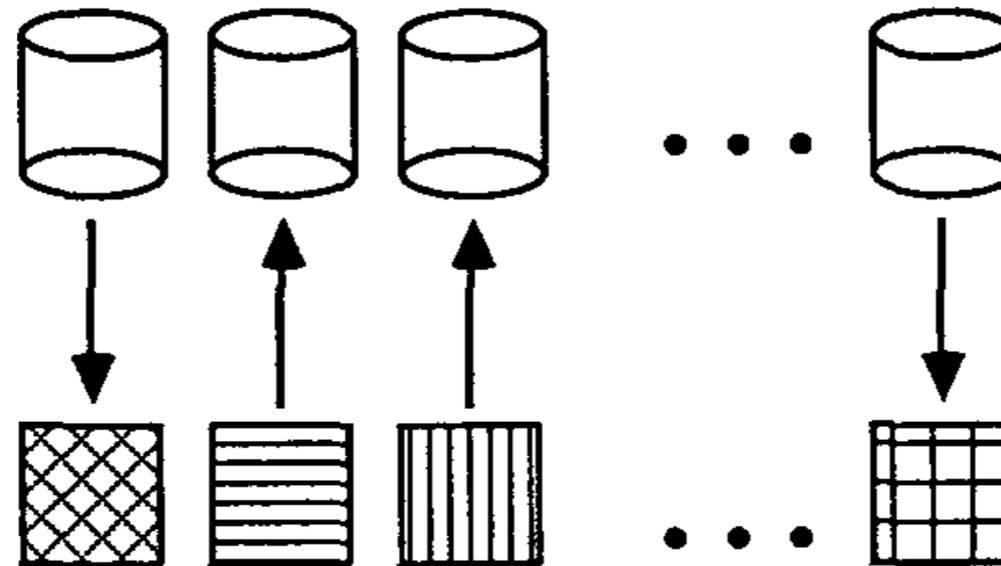
<i>Characteristics</i>	<i>IBM 3380</i>	<i>Fujitsu M2361A</i>	<i>Connors CP3100</i>	<i>3380 v. 2361 v 3100 3100</i>	
				<i>(>1 means 3100 is better)</i>	
Disk diameter (inches)	14	10.5	3.5	4	3
Formatted Data Capacity (MB)	7500	600	100	.01	.2
Price/MB(controller incl.)	\$18-\$10	\$20-\$17	\$10-\$7	1-2.5	1.7-3
MTTF Rated (hours)	30,000	20,000	30,000	1	1.5
MTTF in practice (hours)	100,000	?	?	?	?
No. Actuators	4	1	1	.2	1
Maximum I/O's/second/Actuator	50	40	30	.6	.8
Typical I/O's/second/Actuator	30	24	20	.7	.8
Maximum I/O's/second/box	200	40	30	.2	.8
Typical I/O's/second/box	120	24	20	.2	.8
Transfer Rate (MB/sec)	3	2.5	1	.3	.4
Power/box (W)	6,600	640	10	660	64
Volume (cu. ft.)	24	3.4	.03	800	110

Source: [4]

READ/WRITE PATTERNS



*(a) Single Large or "Grouped" Read
(1 read spread over G disks)*

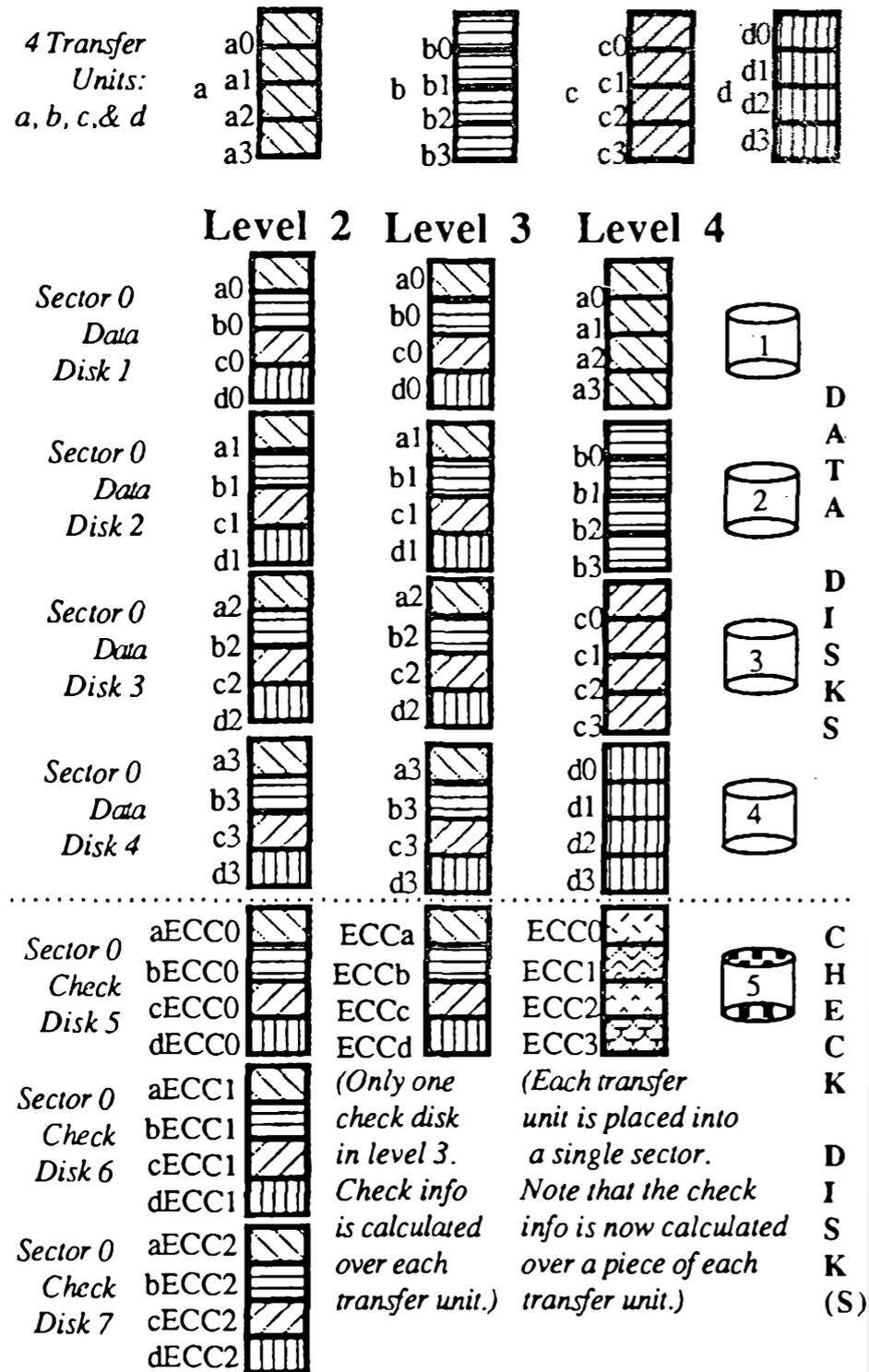


*(b) Several Small or Individual Reads and Writes
(G reads and/or writes spread over G disks)*

Source: [4]

- RAID1: Mirrored Disks
- RAID2: Hamming Code for ECC
- RAID3: Single Check Disk per Group
- RAID4: Independent Reads/Writes
- RAID5: No Single Check Disk

PLACEMENT: RAID2/3/4



For RAID4, small writes become more efficient than for RAID2/3. Parity information can be updated as follows:

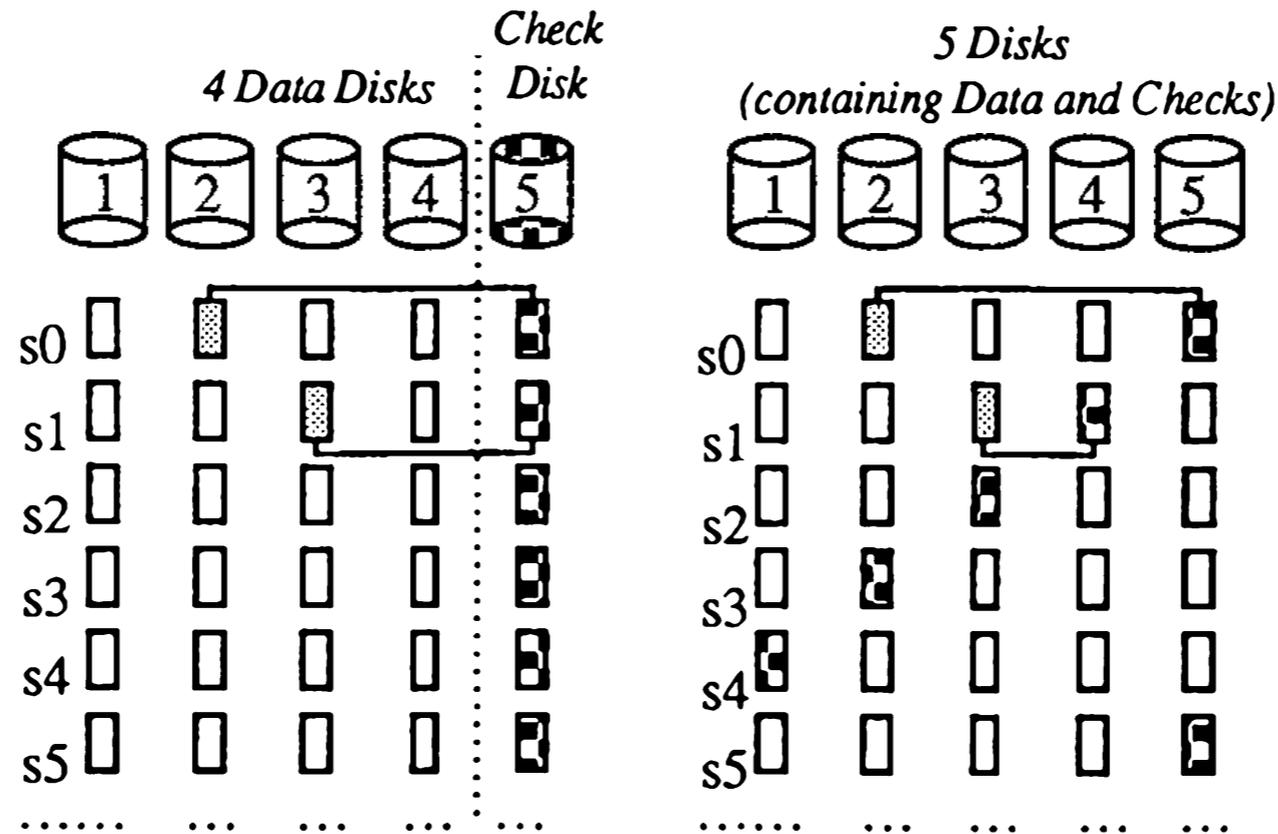
$$\text{new parity} = (\text{old data} \text{ xor } \text{new data}) \text{ xor } \text{old parity}$$

Thus, old data and old parity are read (in parallel), then updated data and parity sectors are written (also in parallel).

But, single check disk is bottleneck!

Source: [4]

PLACEMENT: RAID4/5



(a) Check information for Level 4 RAID for $G=4$ and $C=1$. The sectors are shown below the disks. (The checked areas indicate the check information.) Writes to s_0 of disk 2 and s_1 of disk 3 imply writes to s_0 and s_1 of disk 5. The check disk (5) becomes the write bottleneck.

(b) Check information for Level 5 RAID for $G=4$ and $C=1$. The sectors are shown below the disks, with the check information and data spread evenly through all the disks. Writes to s_0 of disk 2 and s_1 of disk 3 still imply 2 writes, but they can be split across 2 disks: to s_0 of disk 5 and to s_1 of disk 4.

Source: [4]

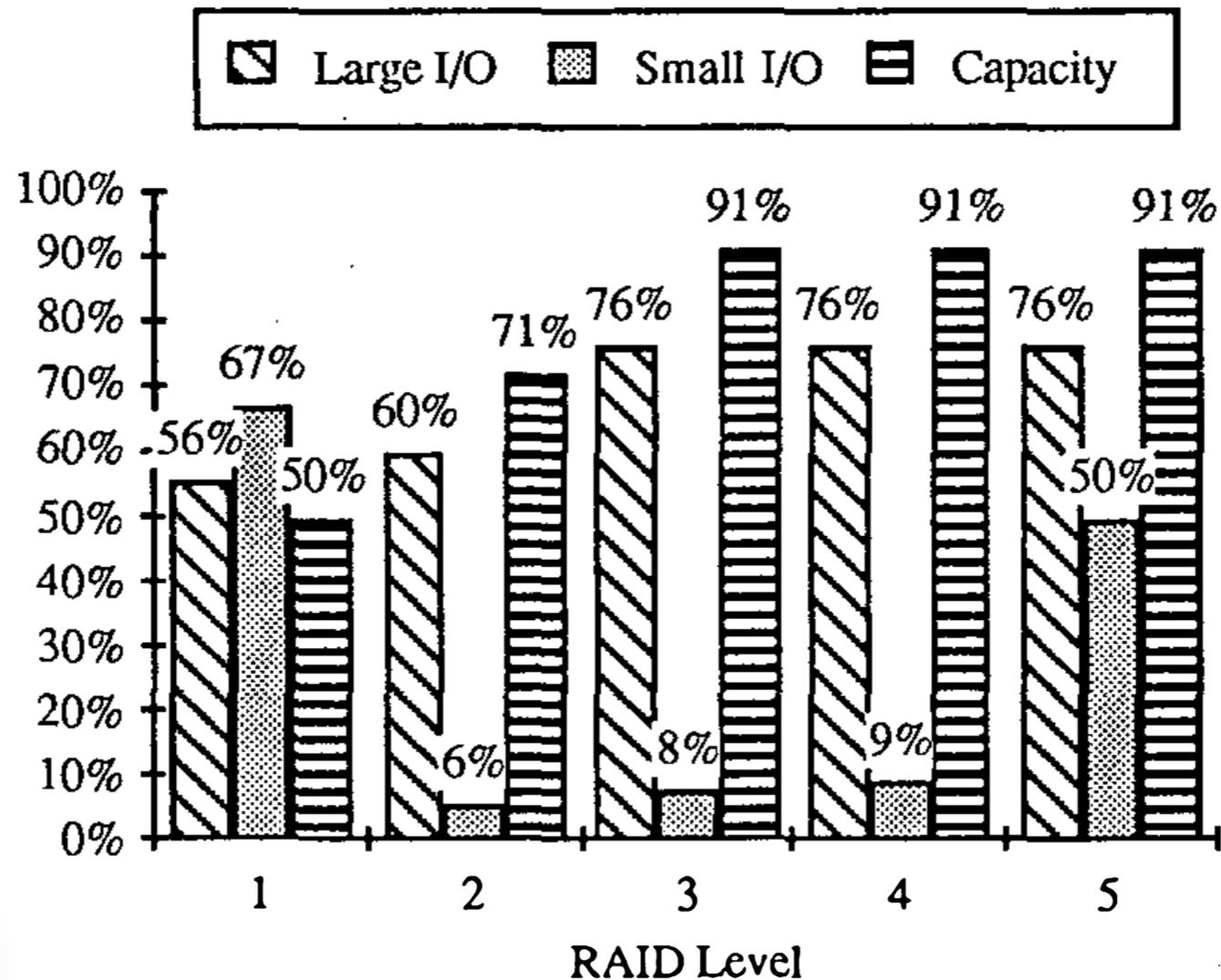


Figure 5. Plot of Large (Grouped) and Small (Individual) Read-Modify-Writes per second per disk and useable storage capacity for all five levels of RAID ($D=100$, $G=10$). We assume a single S factor uniformly for all levels, with $S=1.3$ where it is needed.

<i>Characteristics</i>	<i>RAID 5L (100,10) (CP3100)</i>	<i>SLED (IBM 3380)</i>	<i>RAID v. SLED (>1 better for RAID)</i>	<i>RAID 5L (10,10) (CP3100)</i>	<i>SLED (Fujitsu M2361)</i>	<i>RAID v. SLED (>1 better for RAID)</i>
Formatted Data Capacity (MB)	10,000	7,500	1.33	1,000	600	1.67
Price/MB (controller incl.)	\$11-\$8	\$18-\$10	2.2-.9	\$11-\$8	\$20-\$17	2.5-1.5
Rated MTTF (hours)	820,000	30,000	27.3	8,200,000	20,000	410
MTTF in practice (hours)	?	100,000	?	?	?	?
No. Actuators	110	4	22.5	11	1	11
Max I/O's/Actuator	30	50	.6	30	40	.8
Max Grouped RMW/box	1250	100	12.5	125	20	6.2
Max Individual RMW/box	825	100	8.2	83	20	4.2
Typ I/O's/Actuator	20	30	.7	20	24	.8
Typ Grouped RMW/box	833	60	13.9	83	12	6.9
Typ Individual RMW/box	550	60	9.2	55	12	4.6
Volume/Box (cubic feet)	10	24	2.4	1	3.4	3.4
Power/box (W)	1100	6,600	6.0	110	640	5.8
Min. Expansion Size (MB)	100-1000	7,500	7.5-75	100-1000	600	0.6-6

Table VII. Comparison of IBM 3380 disk model AK4 to Level 5 RAID using 10 Conners & Associates CP 3100s disks and a group size of 10 and a comparison of the Fujitsu M2361A "Super Eagle" to a level 5 RAID using 10 inexpensive data disks with a group size of 10. Numbers greater than 1 in the comparison columns favor the RAID.

[1] *"A Fast File System for UNIX"*, Marshall Kirk McKusick, William N. Joy, Samuel J. Leffer, Robert S. Fabry, ACM Transactions on Computer Systems (TOCS), Vol. 2 Issue 3, Aug. 1984

[2] *"Soft updates: a Technique for Eliminating Most Synchronous Writes in the Fast Filesystem"*, Marshall Kirk McKusick und Gregory R. Ganger, ATEC '99 Proceedings of the annual conference on USENIX Annual Technical Conference, 1999

[3] *"The Design and Implementation of a Log-Structured File System"*, Mendel Rosenblum und John K. Ousterhout, ACM Transactions on Computer Systems (TOCS) Volume 10 Issue 1, Februar 1992

[4] *"A Case for Redundant Arrays of Inexpensive Disks (RAID)"*, David A. Patterson, Garth Gibson, Randy H. Katz, SIGMOD '88 Proceedings of the 1988 ACM SIGMOD International Conference on Management of Data