Operating Systems (A)
(Honor Track)

Lecture 19: File Systems (II)

Yao Guo (郭耀)

Peking University

Fall 2024

Acknowledgements: Prof. Xiangqun Chen & Tao Wang at PKU and Prof. Yuanyuan Zhou at UCSD

This Lecture

File Systems (II)

Fast File System (FFS)

Journaling File Systems

Log-structure File Systems

FAT

GFS

Fast File System (1980s)

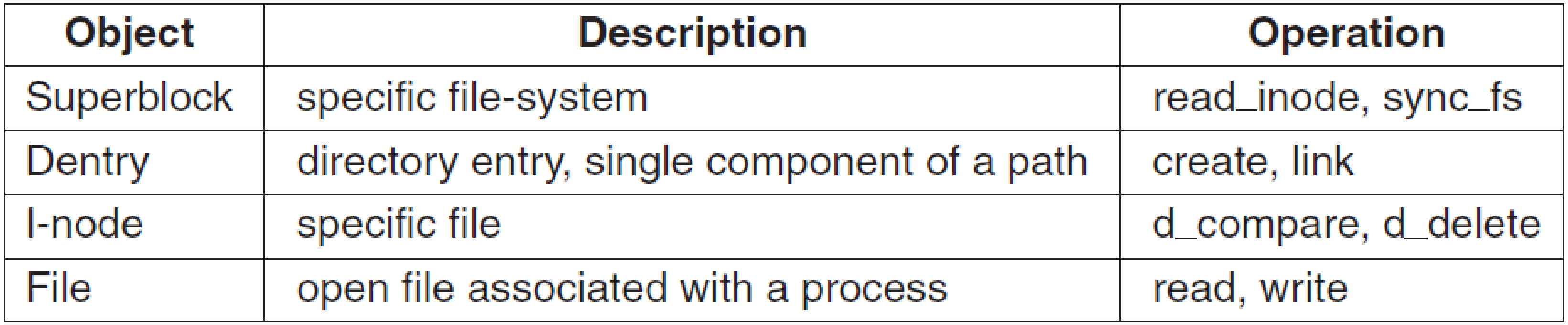
- Old Unix FS: performance degradation over time
- FFS: First disk-aware file system
  - Bitmaps
  - Locality groups
  - Rotated superblocks
  - Large blocks (4KB)
  - Fragments (inside each block)
  - Smart allocation policy
- FFS inspired modern files systems, including ext2 and ext3
- FFS also introduced several new features:
  - Long file names
  - Atomic rename
  - Symbolic links

The Linux File System

- MINIX
  - File names of 14 characters and file size of 64 MB
- ext
  - Allowed file names of 255 characters and files of 2 GB
  - Performance is worse than MINIX
- ext2
  - Long file names, long files, and better performance
- ext3
- ext4
- btrfs, xfs, zfs, etc.

The Linux VFS

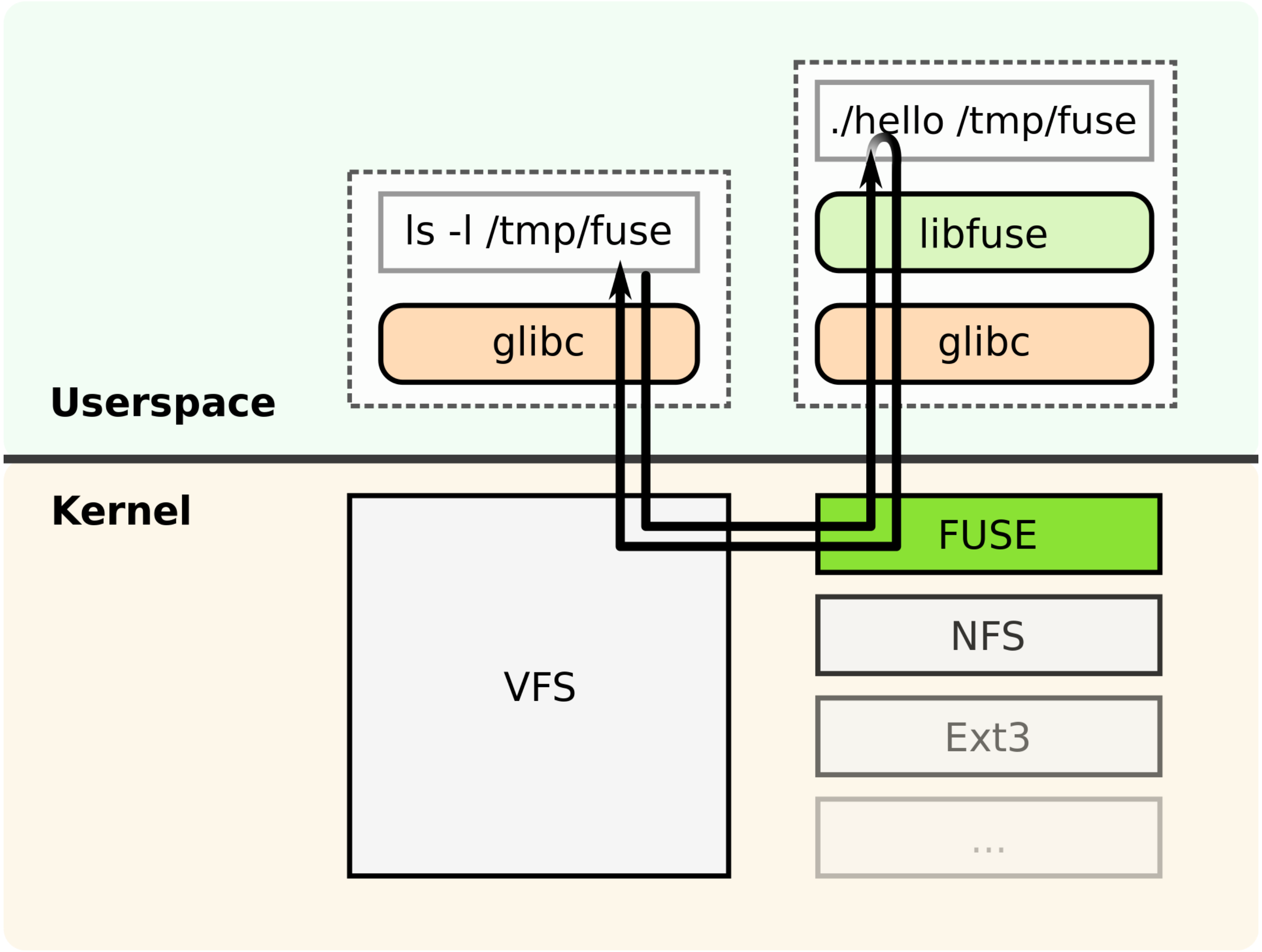
- Virtual File System (VFS)
  - Defines a set of basic file-system abstractions and the operations which are allowed on these abstractions
  - Supports multiple file systems



Object	Description	Operation
Superblock	specific file-system	read_inode, sync_fs
Dentry	directory entry, single component of a path	create, link
I-node	specific file	d_compare, d_delete
File	open file associated with a process	read, write

Figure 10-30. File-system abstractions supported by the VFS.

File System in Userspace (FUSE)



Data Redundancy

- □ Definition:

  if *A* and *B* are two pieces of data,
  and knowing *A* eliminates some or all values *B* could be,
  there is redundancy between *A* and *B*

- □ RAID examples:
  - ■ mirrored disk (complete redundancy)
  - ■ parity blocks (partial redundancy)
- □ File system examples:
  - ■ Superblock: field contains total blocks in FS
  - ■ Inodes: field contains pointer to data block
  - ■ Is there redundancy between these two types of fields? Why or why not?

File System Redundancy Example

Superblock: field contains total number of blocks in FS

DATA = N

Inode: field contains pointer to data block; possible DATA?

DATA in {0, 1, 2, …, N - 1}

Pointers to block N or after are invalid!

Total-blocks field has redundancy with inode pointers

Consistency Examples

Assumptions:

Superblock: field contains total blocks in FS.

DATA = 1024

Inode: field contains pointer to data block.

DATA in {0, 1, 2, …, 1023}

Scenario 1: Consistent or not?

Superblock: field contains total blocks in FS.

DATA = 1024

Inode: field contains pointer to data block.

DATA = 241

Consistent

Scenario 2: Consistent or not?

Superblock: field contains total blocks in FS.

DATA = 1024

node: field contains pointer to data block.

DATA = 2345

Inconsistent

Pros and CONs of Redundancy

Redundancy may improve:

- reliability
  - □ RAID-5 parity
  - □ Superblocks in FFS
- performance
  - □ RAID-1 mirroring (reads)
  - □ FFS group descriptor
  - □ FFS bitmaps

Redundancy hurts:

- capacity
- consistency
  - □ Redundancy implies certain combinations of values are illegal
  - □ Illegal combinations: inconsistency

How to fix Inconsistencies?

FSCK can be pronounced "F-S-C-K", "F-S-check", "fizz-check", "F-sack", "fisk", "fishcake", "fizik", "F-sick", "F-sock", "F-sek", "feshk", etc.

- Solution #1:
  - FSCK = file system checker
- Strategy:
  - After crash, scan whole disk for contradictions and “fix” if needed
  - Keep file system off-line until FSCK completes
- For example, how to tell if data bitmap block is consistent?

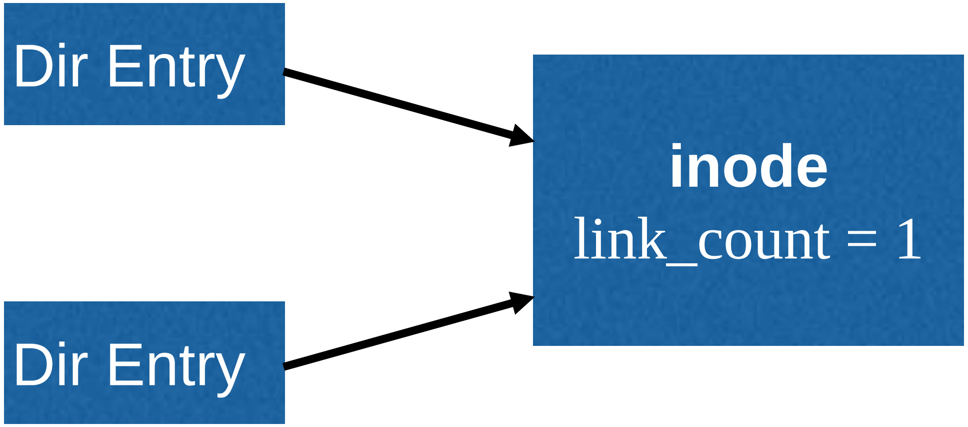
Read every valid inode+indirect block
If pointer to data block, the corresponding bit should be 1; else bit is 0

Fsck Checks

Hundreds of types of checks over different fields…

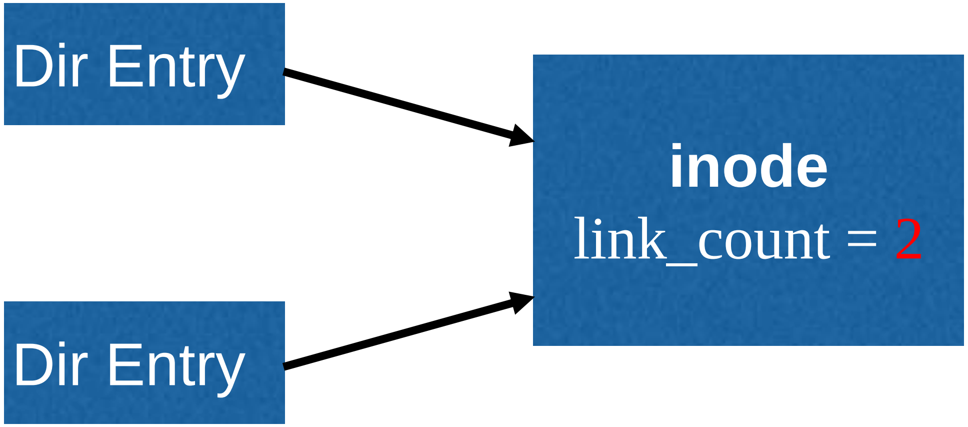
- Do superblocks match?
- Do directories contain “.” and “..”?
- Do number of dir entries equal **inode link counts**?
- Do different inodes ever point to **same block**?
- …

Link Count (example 1)



How to fix to have consistent file system?

Link Count (example 1)



Simple fix!

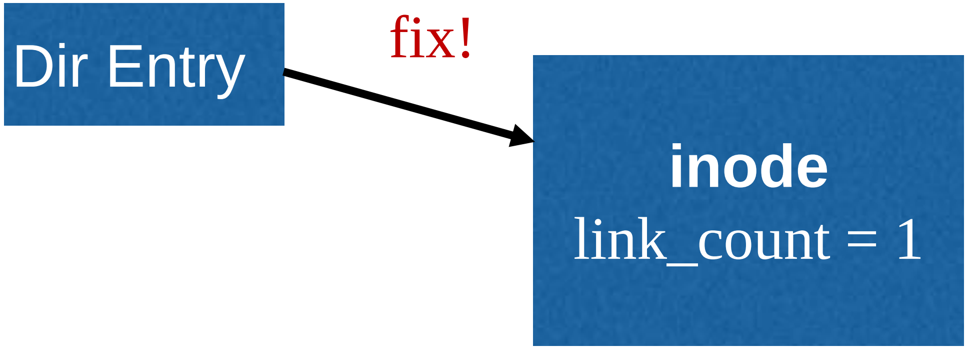
Link Count (example 2)

inode

link_count = 1

How to fix???

Link Count (example 2)



Dir Entry

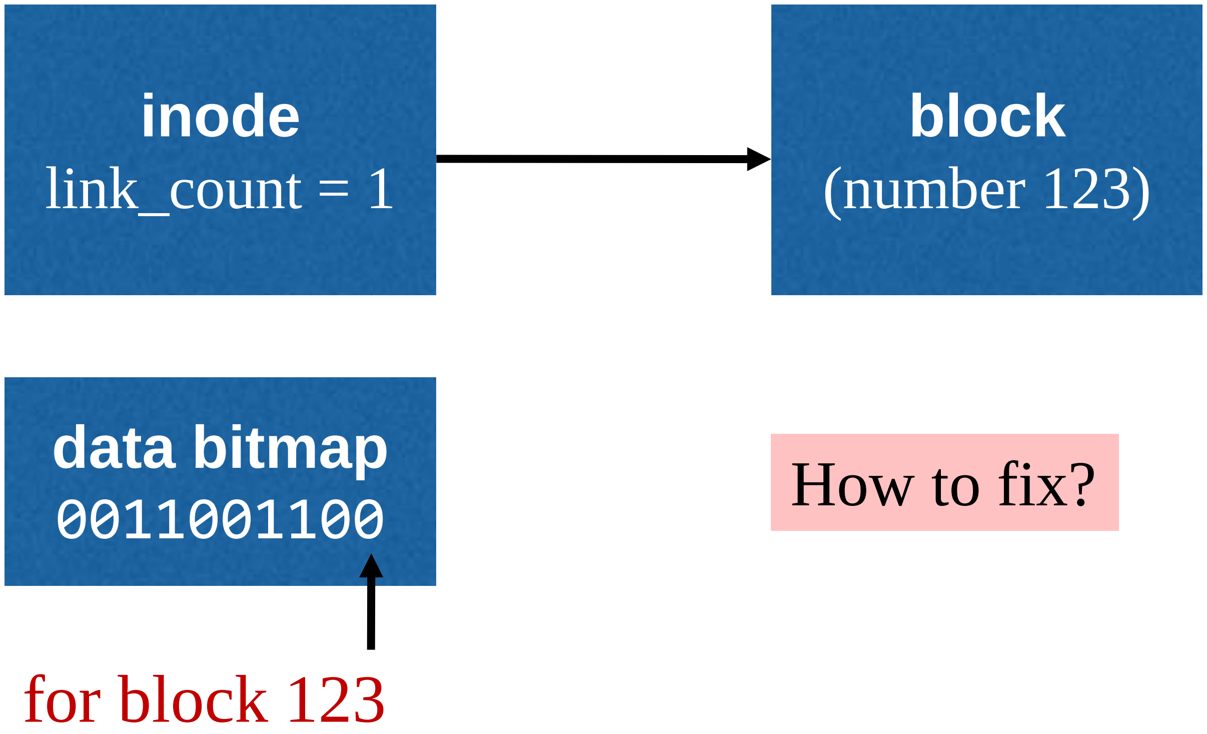
fix!

inode

link_count = 1

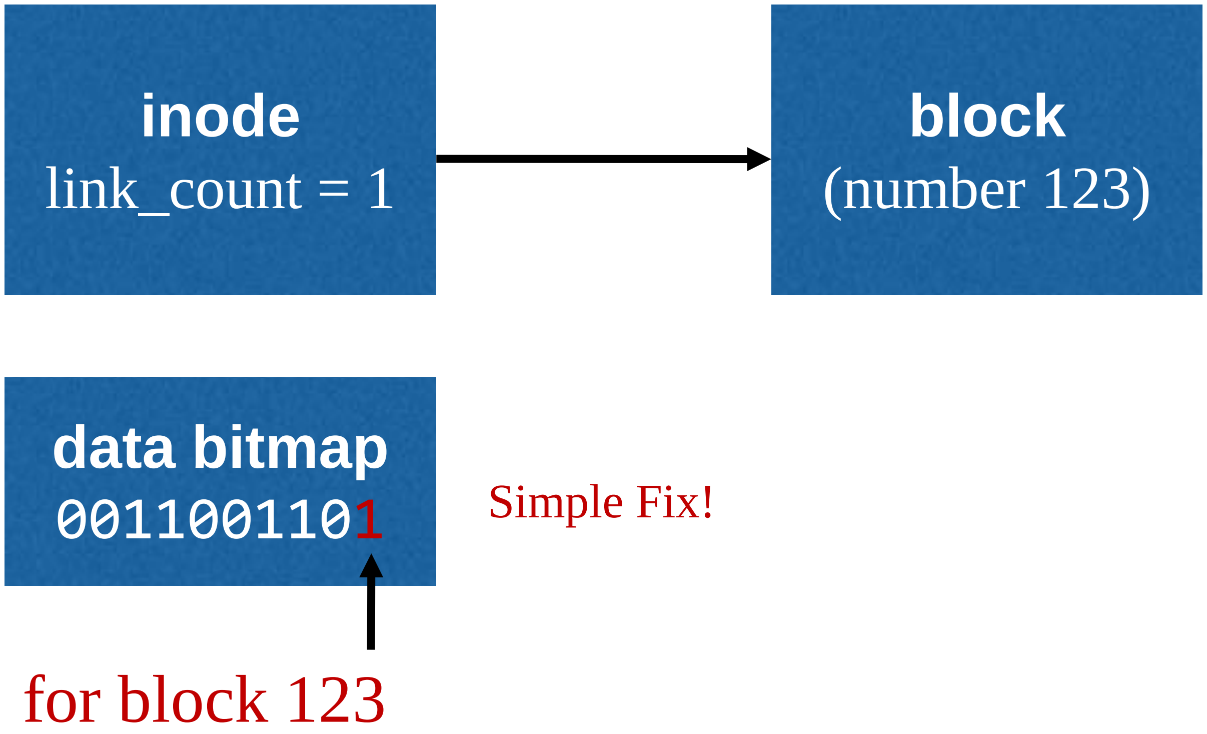
```
ls -l /
total 150
drwxr-xr-x  401 18432 Dec 31  1969 afs/
drwxr-xr-x.   2 4096  Nov  3 09:42 bin/
drwxr-xr-x.   5 4096  Aug  1 14:21 boot/
dr-xr-xr-x.  13 4096  Nov  3 09:41 lib/
dr-xr-xr-x.  10 12288 Nov  3 09:41 lib64/
drwx------.   2 16384 Aug  1 10:57 lost+found/
...
```

Data Bitmap



How to fix?

Data Bitmap



inode

link_count = 1

block

(number 123)

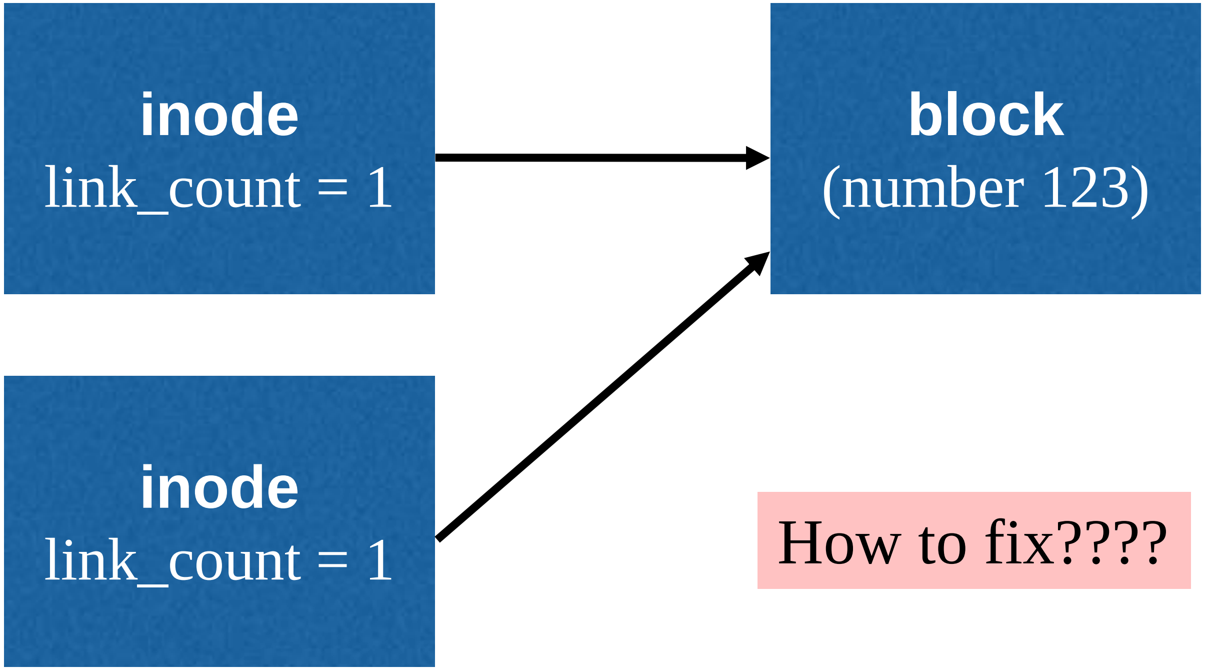
data bitmap

0011001101

Simple Fix!

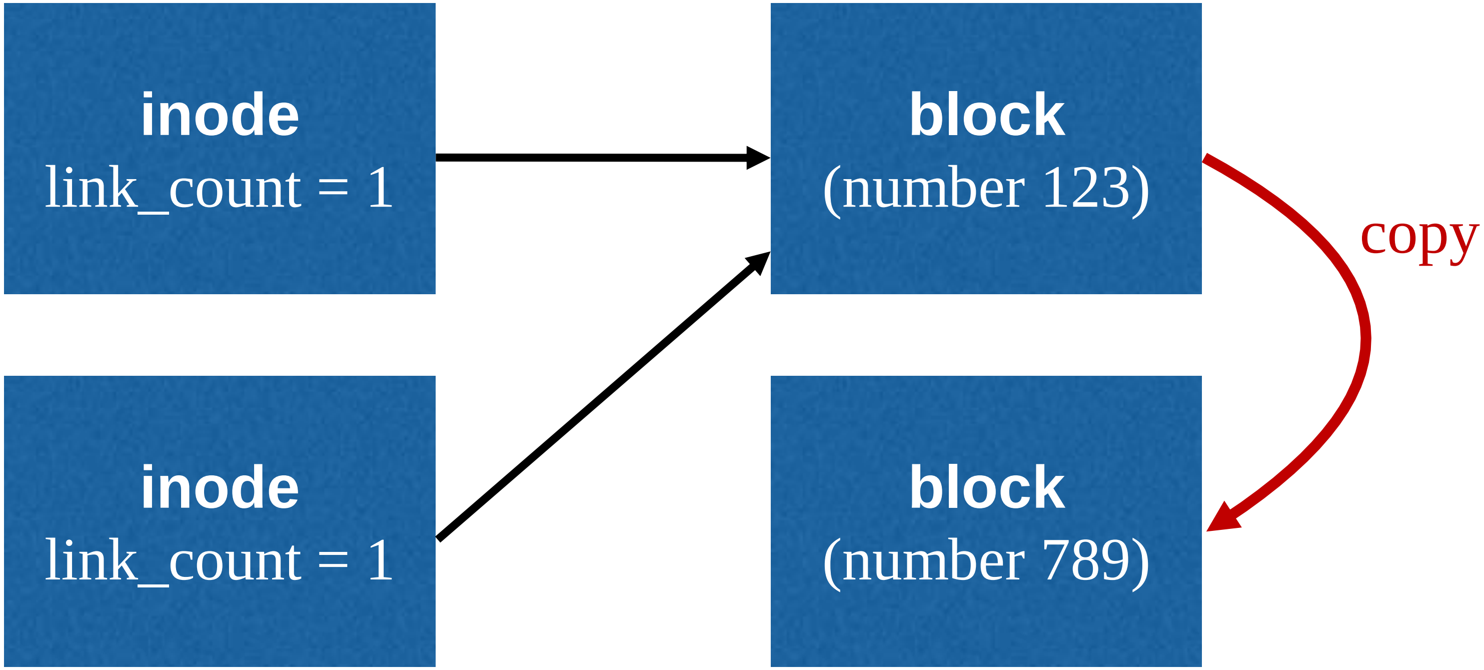
for block 123

Duplicate Pointers

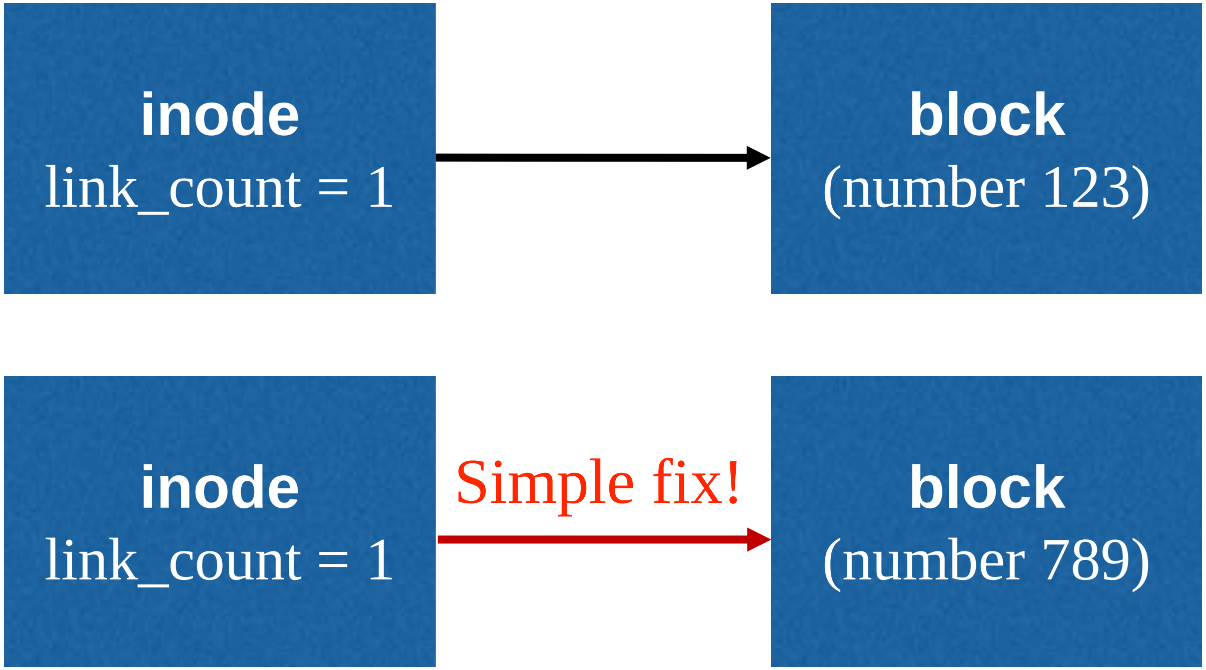


How to fix????

Duplicate Pointers

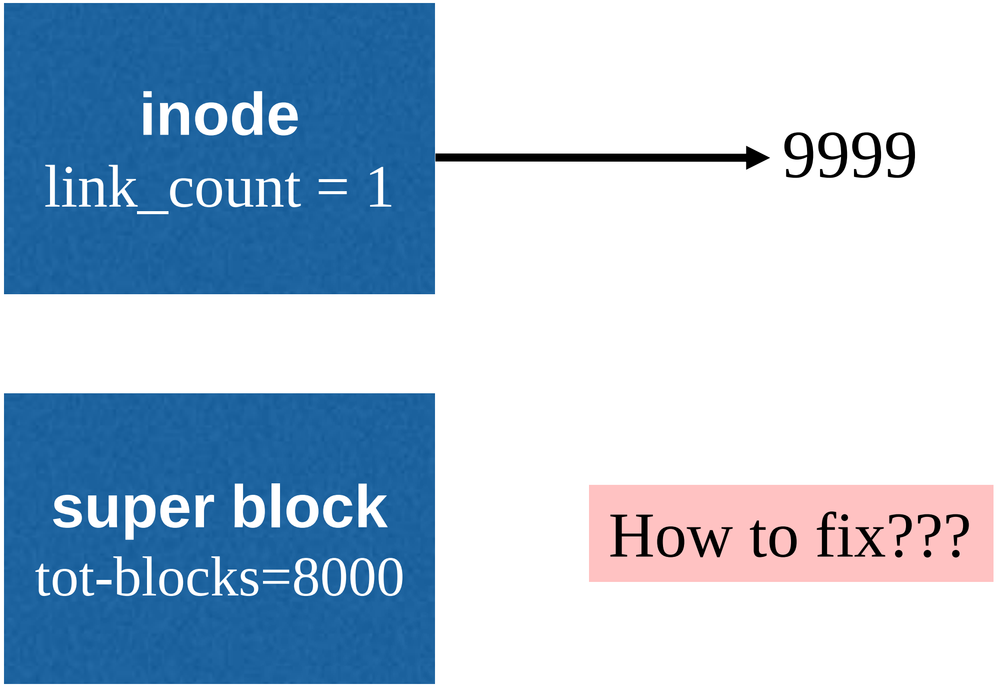


Duplicate Pointers



But is this correct?

Bad Pointer



inode
link_count = 1

9999

super block
tot-blocks=8000

How to fix???

Bad Pointer

inode

link_count = 1

Simple fix! (But is this correct?)

super block

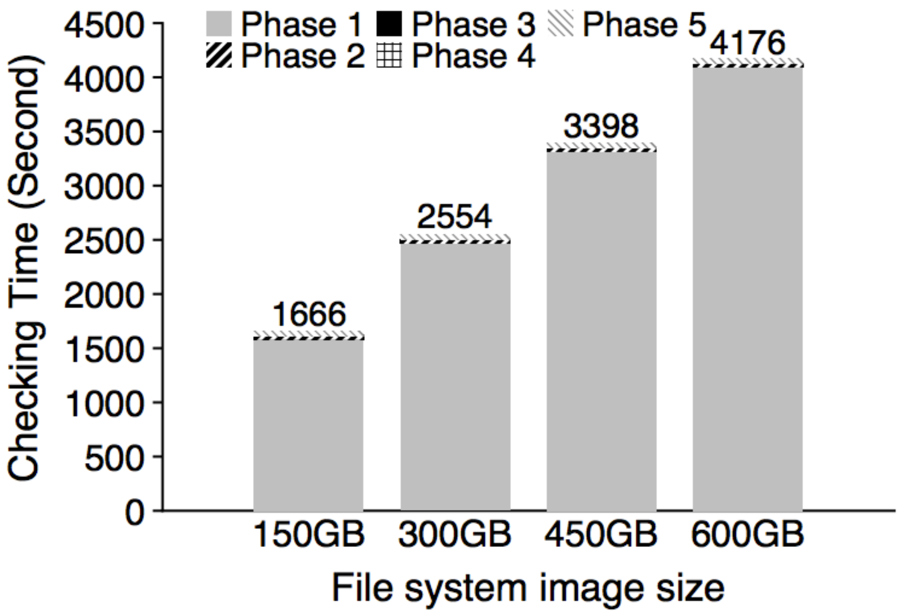
tot-blocks=8000

Problems with fsck

Problem 1:

- Not always obvious how to fix file system image
- Don’t know “correct” state, just consistent one
- Easy way to get consistency: reformat disk!

Problem 2: fsck is very slow



Checking a 600GB disk takes ~70 minutes

ffsck: The Fast File System Checker

Ao Ma, EMC Corporation and University of Wisconsin—Madison; Chris Dragga,
Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau, University of Wisconsin—Madison

Consistency Solution #2: Journaling

- **Goals**
  - Ok to do some **recovery work** after crash, but not to read entire disk
  - Don’t move file system to just any consistent state, get **correct** state
- **Strategy**
  - Atomicity
  - Definition of atomicity for **concurrency**
    - operations in critical sections are not interrupted by operations on related critical sections
  - Definition of atomicity for **persistence**
    - collections of writes are not interrupted by crashes; either (all new) or (all old) data is visible

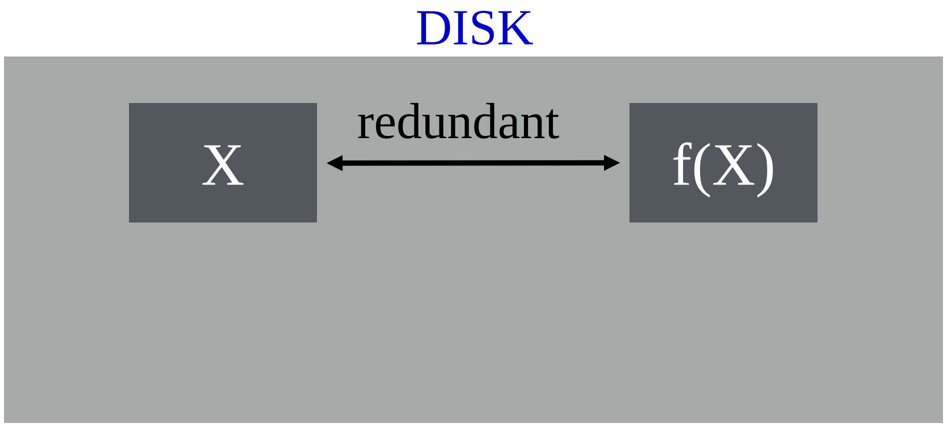
Journaling General Strategy

- Never delete ANY old data, until, ALL new data is safely on disk

- Ironically, adding redundancy to fix the problem caused by redundancy.

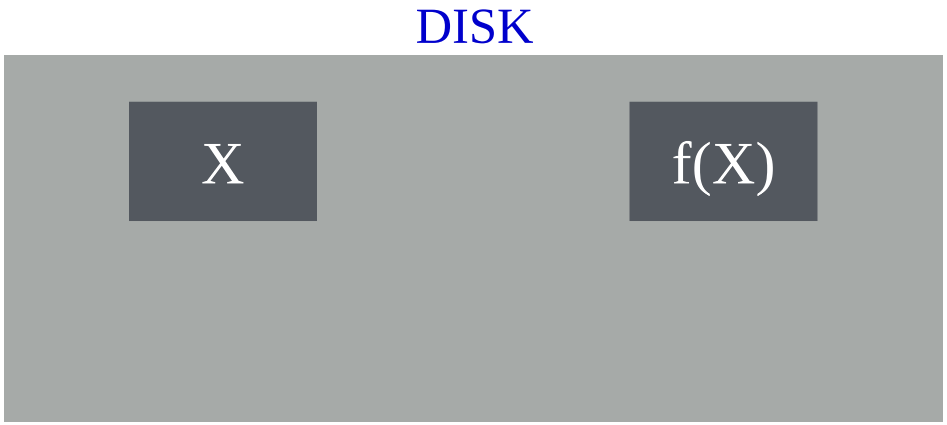
Fight Redundancy with Redundancy

Want to replace X with Y. Original:



Fight Redundancy with Redundancy

Want to replace X with Y. Original:

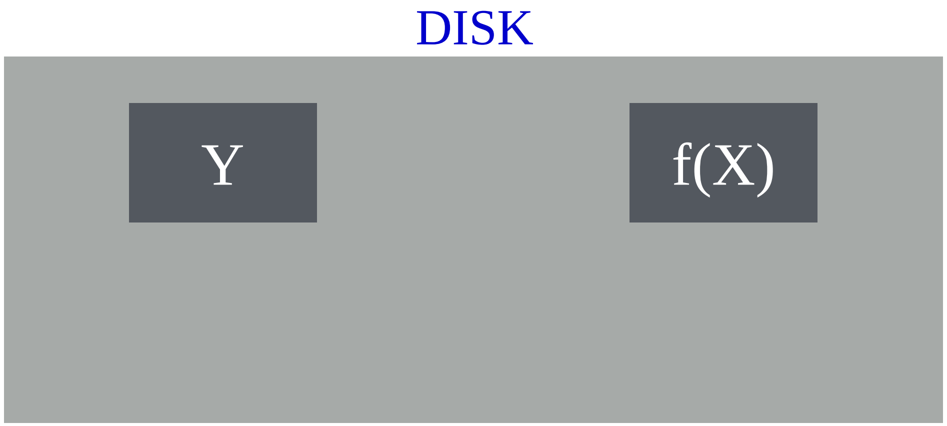


Good time to crash?

good time to crash

Fight Redundancy with Redundancy

Want to replace X with Y. Original:



DISK

Y

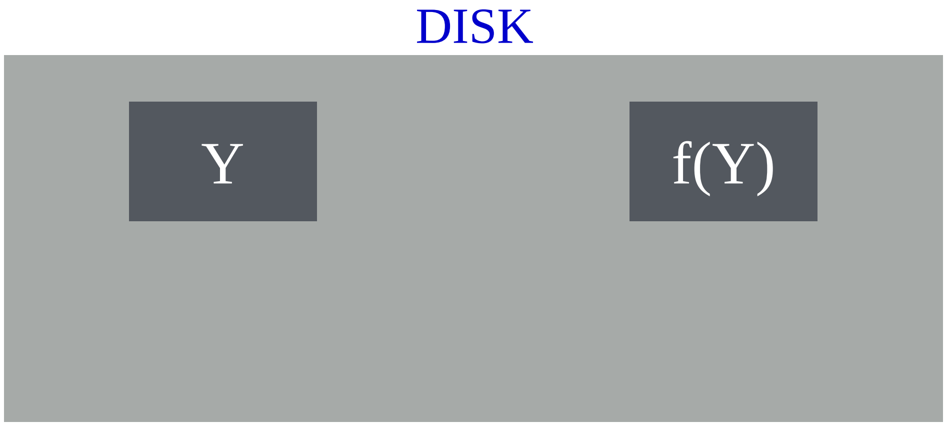
f(X)

Good time to crash?

bad time to crash

Fight Redundancy with Redundancy

Want to replace X with Y. Original:

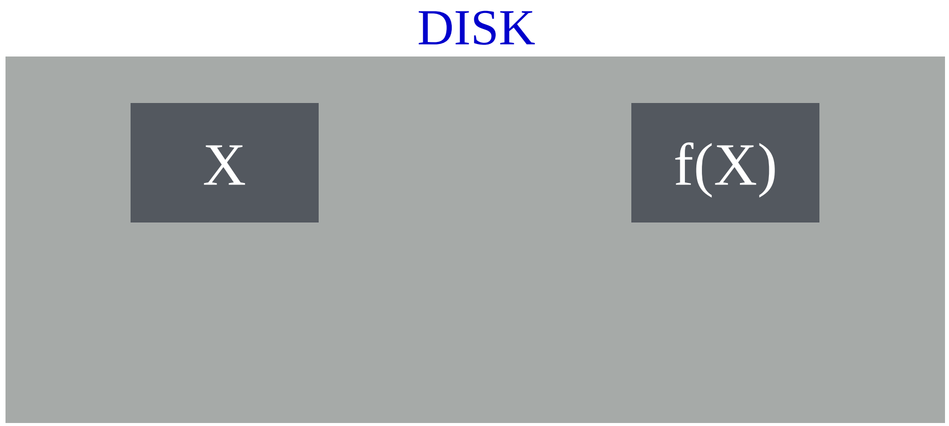


Good time to crash?

good time to crash

Fight Redundancy with Redundancy

Want to replace X with Y.



DISK

X

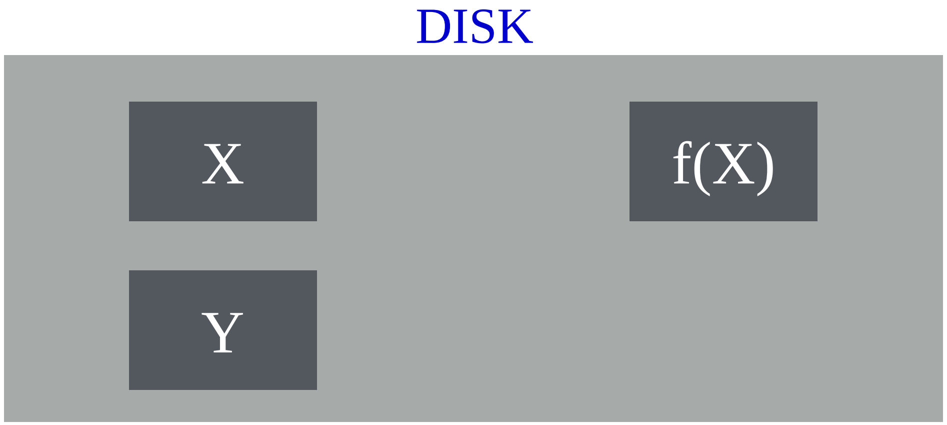
f(X)

Good time to crash?

good time to crash

Fight Redundancy with Redundancy

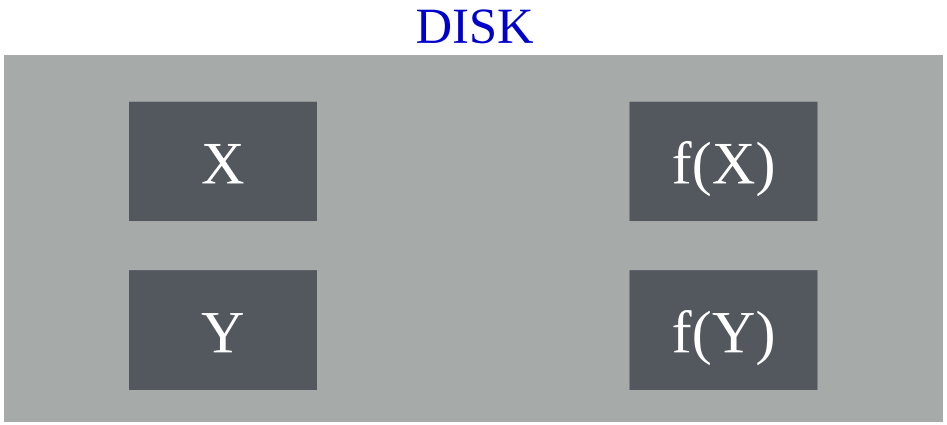
Want to replace X with Y. With journal:



good time to crash

Fight Redundancy with Redundancy

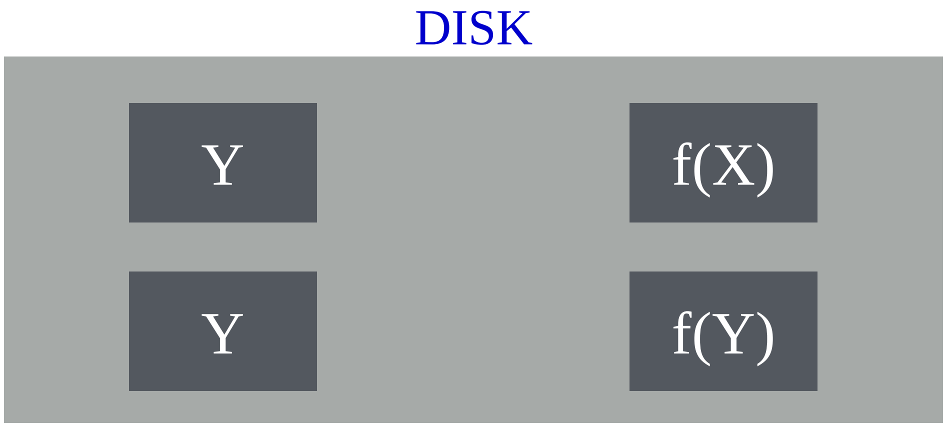
Want to replace X with Y. With journal:



good time to crash

Fight Redundancy with Redundancy

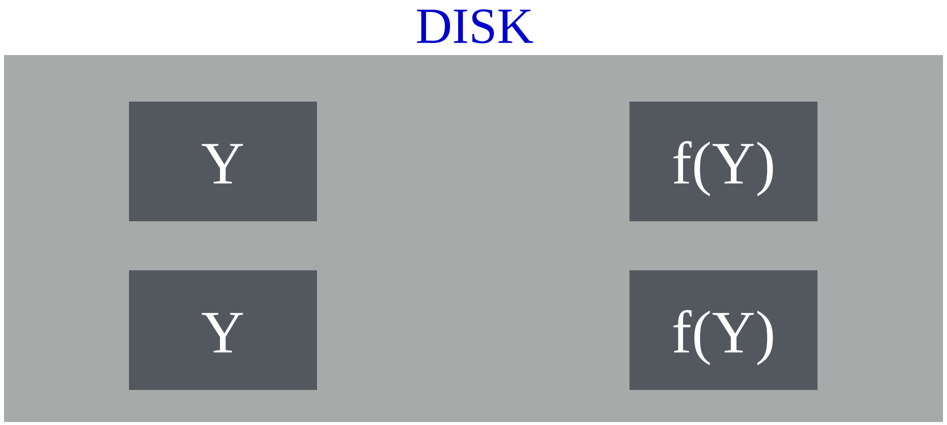
Want to replace X with Y. With journal:



good time to crash

Fight Redundancy with Redundancy

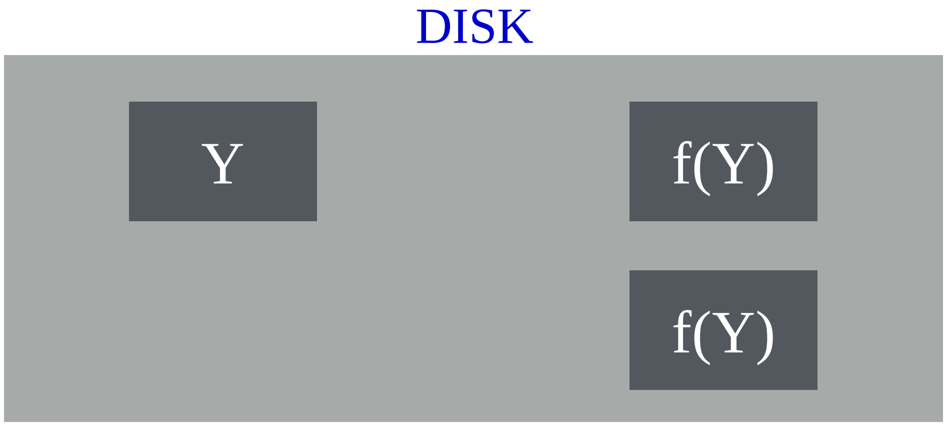
Want to replace X with Y. With journal:



good time to crash

Fight Redundancy with Redundancy

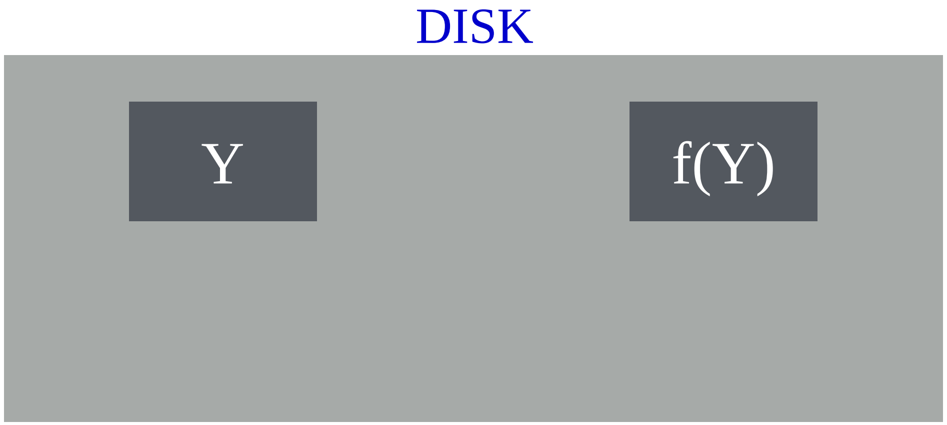
Want to replace X with Y. With journal:



good time to crash

Fight Redundancy with Redundancy

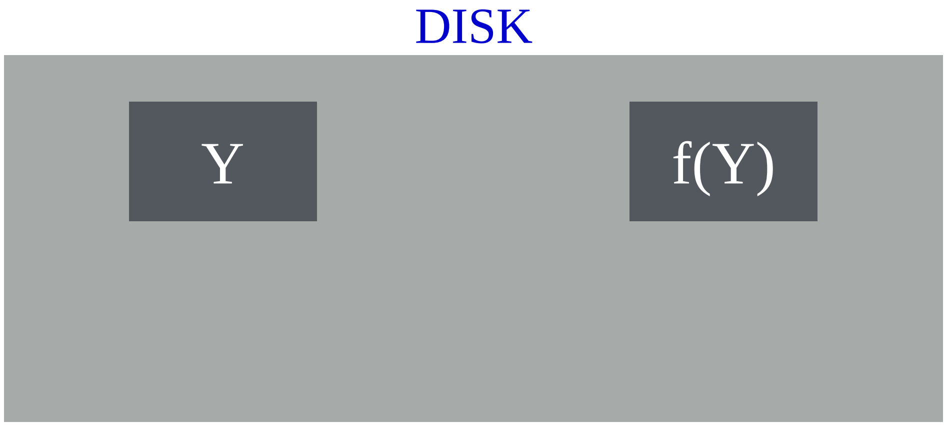
Want to replace X with Y. With journal:



good time to crash

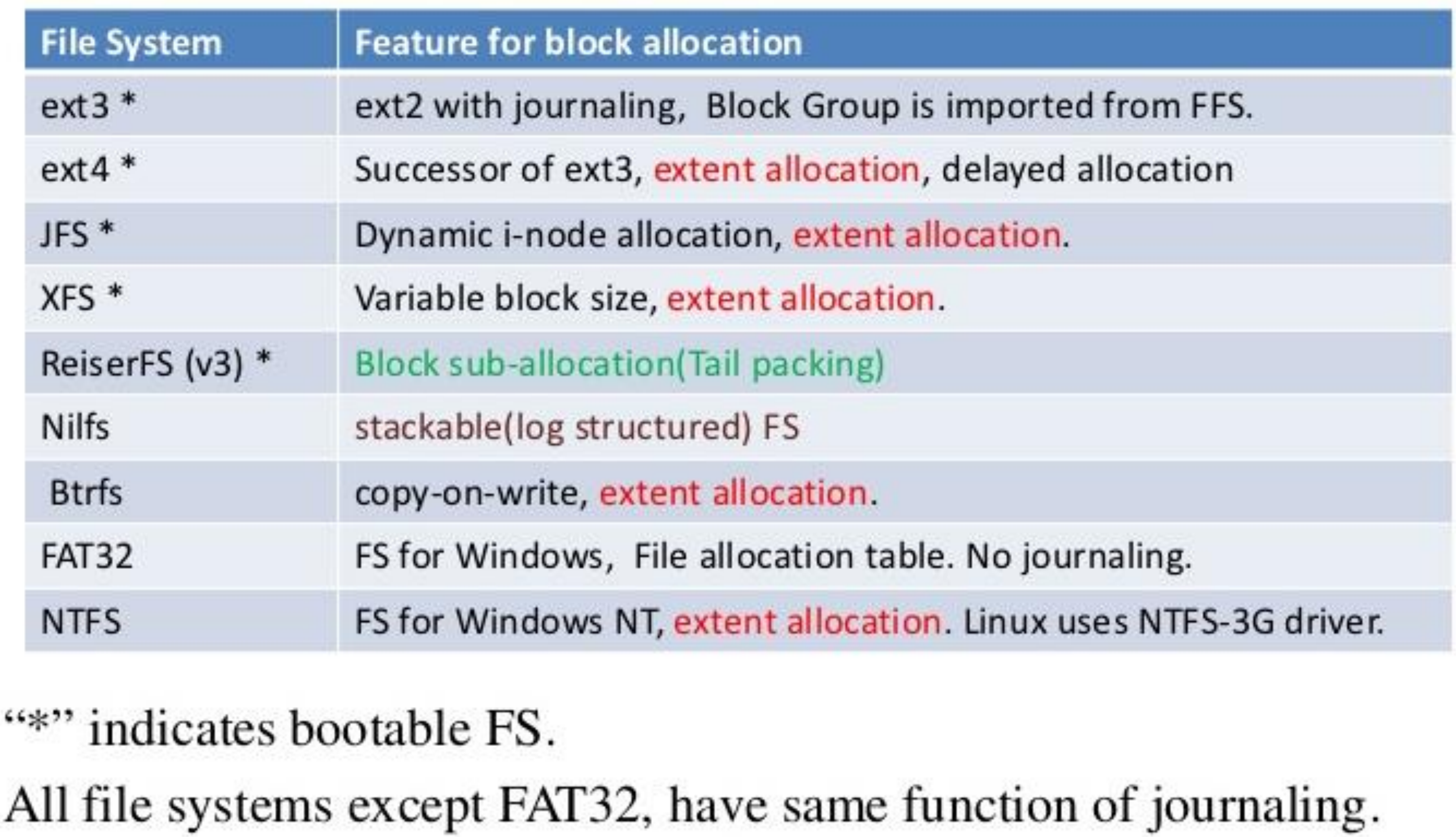
Fight Redundancy with Redundancy

Want to replace X with Y. With journal:



With journaling, it’s always a good time to crash!

Journaling is Widely Used



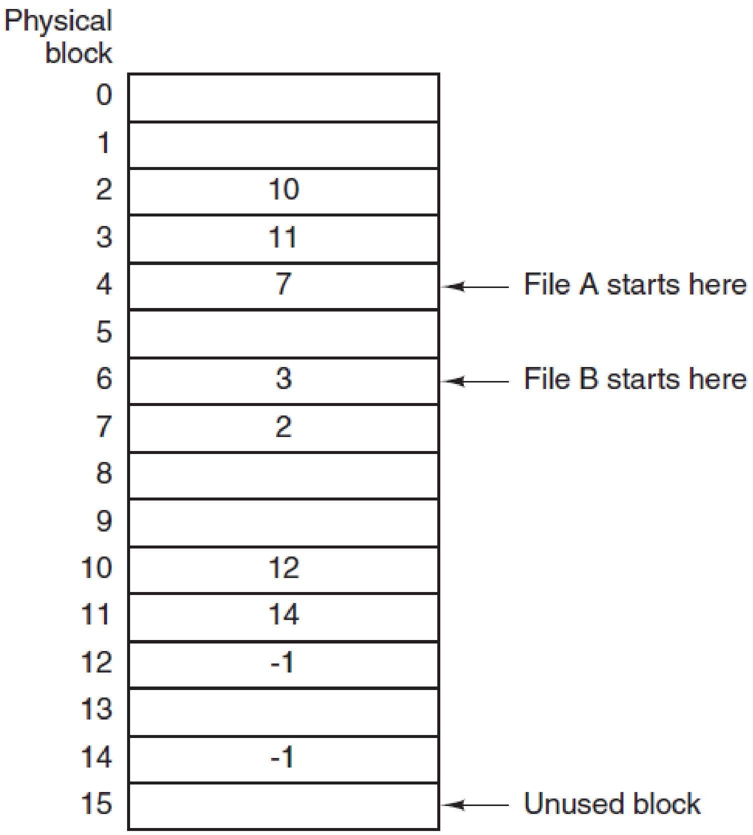
File System	Feature for block allocation
ext3 *	ext2 with journaling, Block Group is imported from FFS.
ext4 *	Successor of ext3, extent allocation, delayed allocation
JFS *	Dynamic i-node allocation, extent allocation.
XFS *	Variable block size, extent allocation.
ReiserFS (v3) *	Block sub-allocation(Tail packing)
Nilfs	stackable(log structured) FS
Btrfs	copy-on-write, extent allocation.
FAT32	FS for Windows, File allocation table. No journaling.
NTFS	FS for Windows NT, extent allocation. Linux uses NTFS-3G driver.

“*” indicates bootable FS.

All file systems except FAT32, have same function of journaling.

FAT

- File Allocation Table (FAT)
  - Used from MS-DOS to Windows ME
  - The entire table must be in memory



Physical block	
0	
1	
2	10
3	11
4	7
5	
6	3
7	2
8	
9	
10	12
11	14
12	-1
13	
14	-1
15	

Figure 4-12. Linked-list allocation using a file-allocation table in main memory.

FAT: Directory Entry

- Directory entry structure

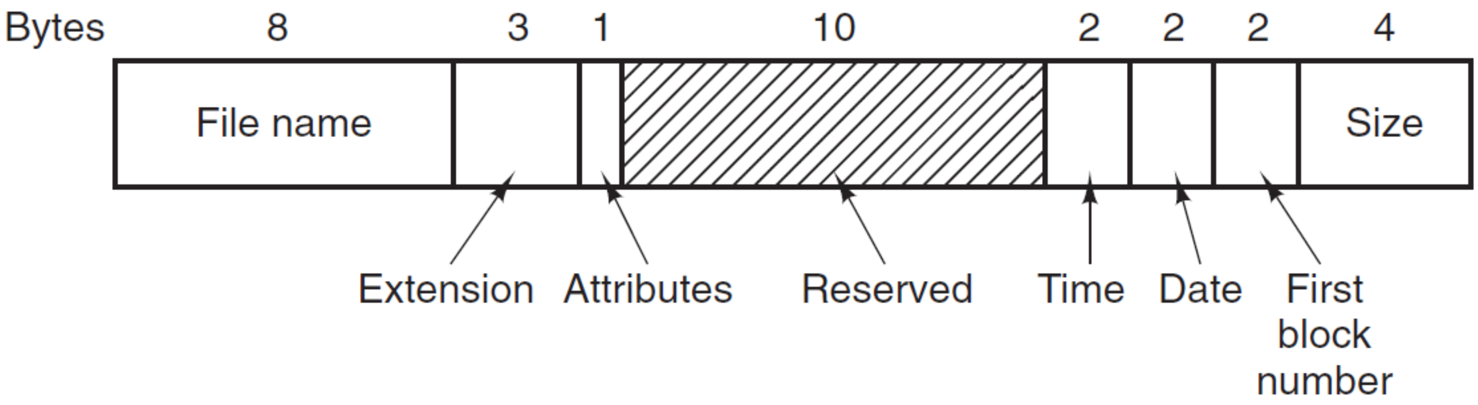


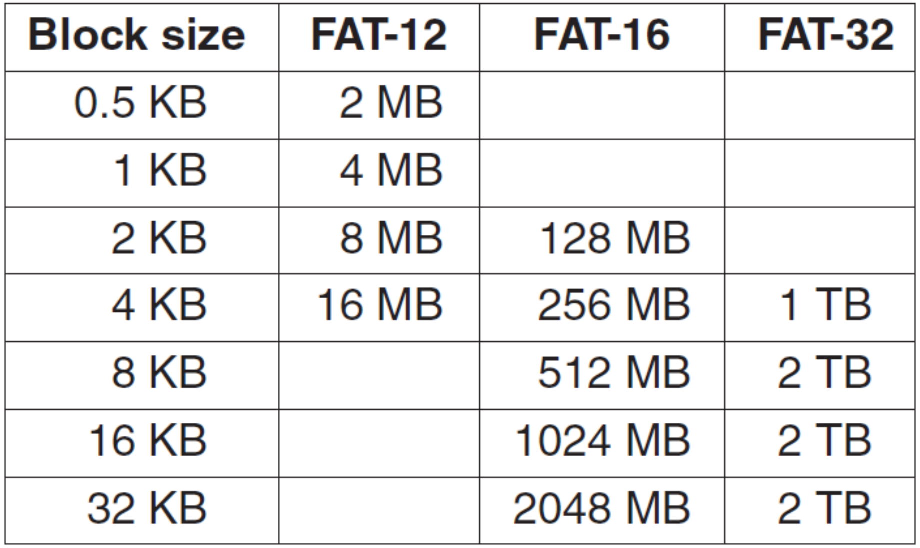
Figure 4-30. The MS-DOS directory entry.

FAT: Implementation

- Groups hard drive sectors into clusters
  - Increases performance by organizing blocks of sectors contiguously
- Maintains the relationship between files and clusters being used for the file
  - Clusters have two entries in the table
    - Current cluster information
    - Link to the next cluster or a special code indicating it is the last cluster
- Keeps track of writable clusters and bad clusters

Different FAT File Systems

- FAT-12, FAT-16, FAT-32
  - Uses different number of bits to address the blocks



Block size	FAT-12	FAT-16	FAT-32
0.5 KB	2 MB		
1 KB	4 MB		
2 KB	8 MB	128 MB	
4 KB	16 MB	256 MB	1 TB
8 KB		512 MB	2 TB
16 KB		1024 MB	2 TB
32 KB		2048 MB	2 TB

Figure 4-31. Maximum partition size for different block sizes. The empty boxes represent forbidden combinations.

Advantages of FAT

- Efficient use of disk space
  - Does not have to use contiguous space for large files
- File names (FAT32) can have up to 255 characters
- Easy to undelete files that have been deleted
  - When a file is deleted, the system places a hex value of E5h in the first position of the file name
  - File remains on drive and can be undeleted by providing the original letter in the undelete process

Disadvantages of FAT

- Overall performance slows down as more files are stored on the partition
- Hard drive can quite easily become fragmented
- Lack of security
  - NTFS provides access rights to files and directories
- File integrity problems
  - Lost clusters
  - Invalid files and directories
  - Allocation errors

NTFS

- Overcomes limitations of the FAT system
- Is a “journaling” file system
  - Keeps track of transaction performed and “rolls back” transactions if errors are found
- Uses a master file table (MFT) to store data about every file and directory on the volume
  - Similar to a database table with records for each file and directory
- Uses clusters and reserves blocks of space to allow the MFT to grow

Advantages of NTFS

- File access is very fast and reliable
- With the MFT, the system can recover from problems without losing significant amounts of data
- Security is greatly increased over FAT
- File encryption with EFS (Encrypting File System) and file attributes
- File compression
  - Process of reducing file size to save disk space

Disadvantages of NTFS

- Large overhead
  - Not recommended for volumes less than 4 GB
- Cannot access NTFS volumes from earlier OSes such as MS-DOS, Windows 5, or Windows 98

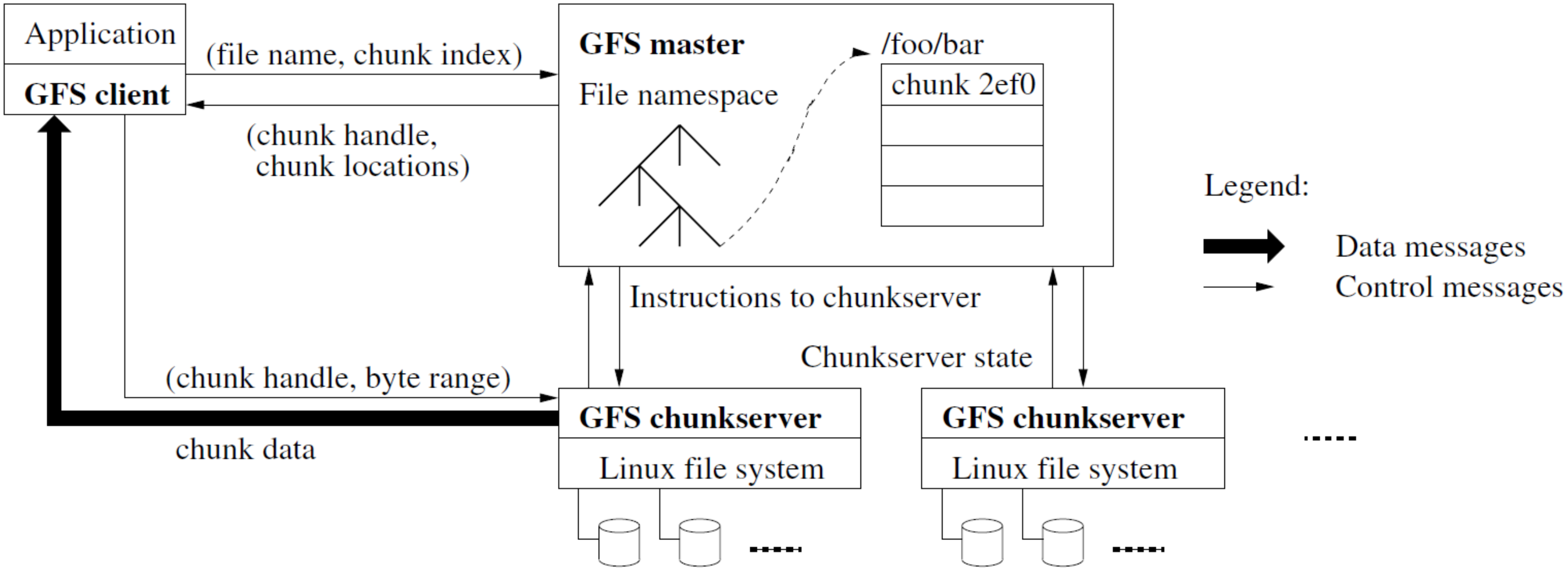
Google File System (GFS)

Motivation

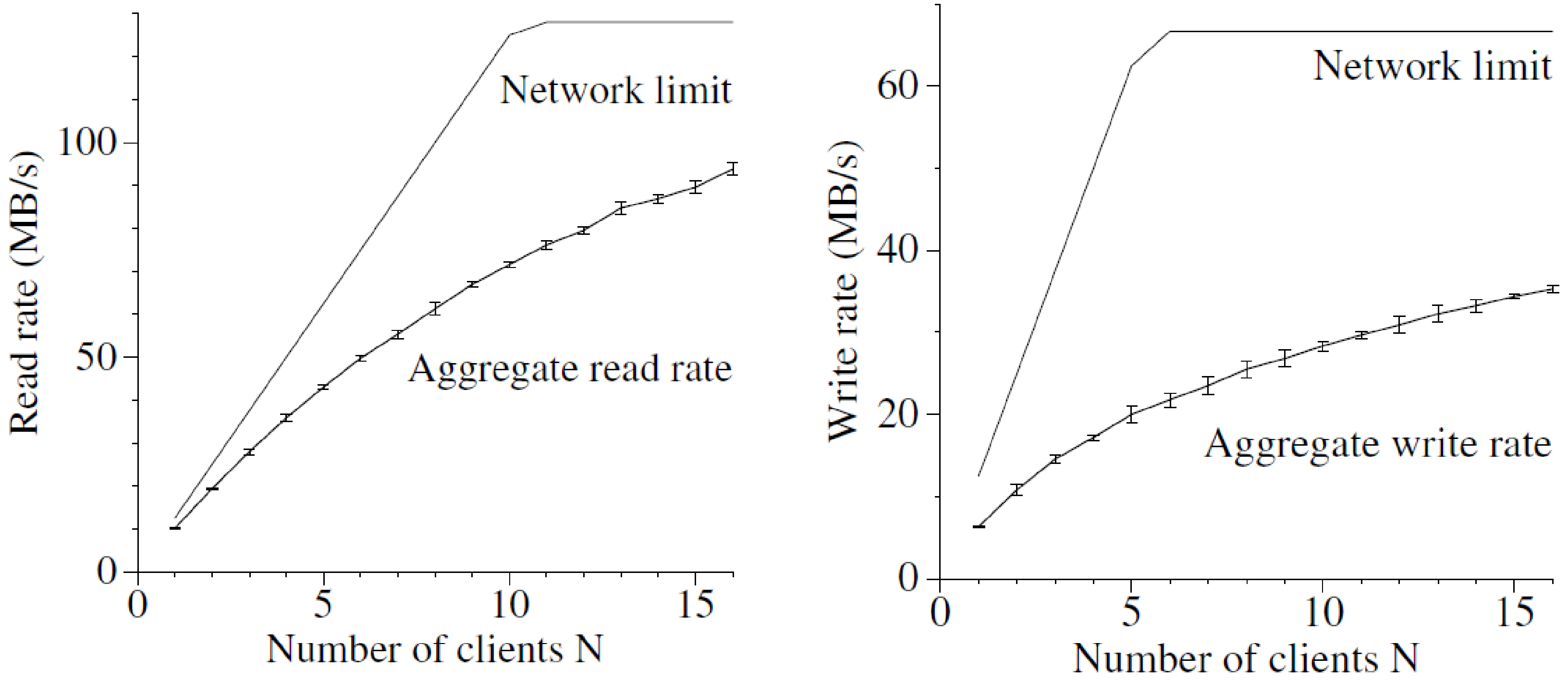
- Google workload characteristics
  - huge files (GBs); usually read in their entirety
  - almost all writes are appends
  - concurrent appends common
  - high throughput is valuable
  - low latency is not
- Computing environment:
  - 1000s of machines
  - machines sometimes fail (both permanently and temporarily)

GFS Architecture

- chunks: 64MB each



GFS Performance

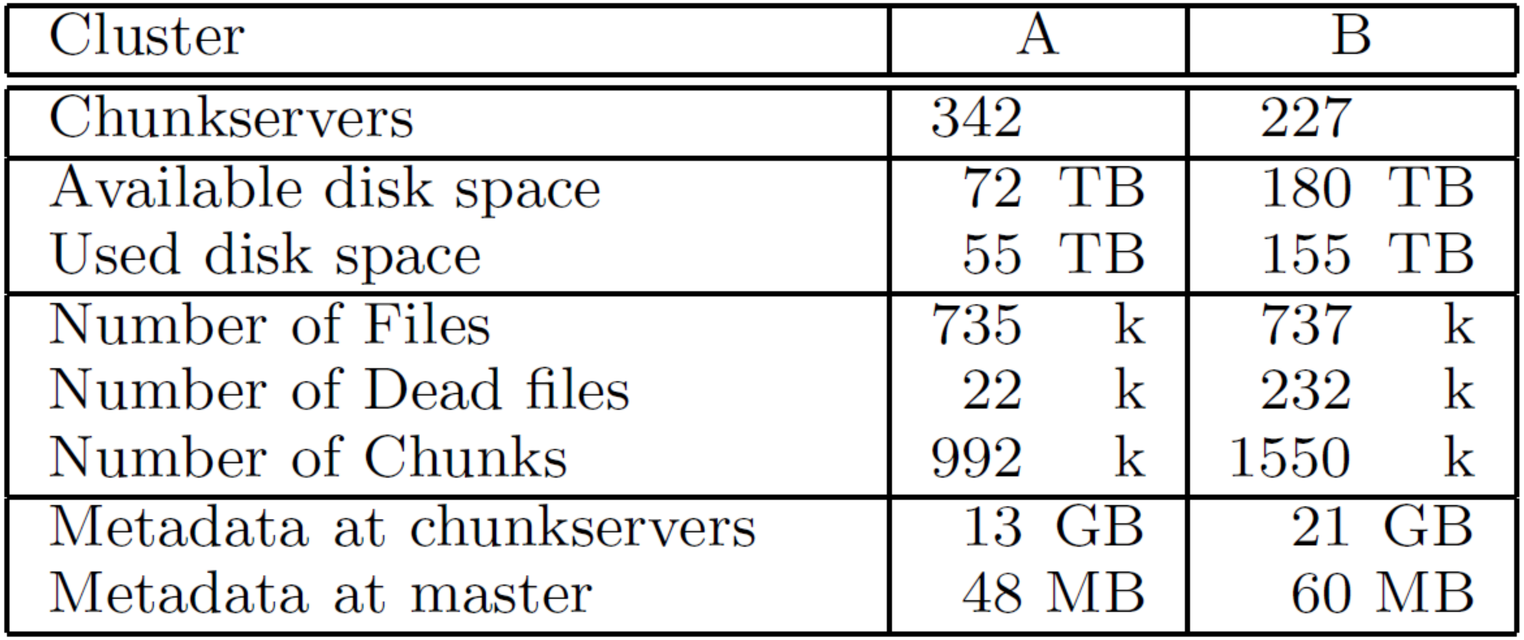


(a) Reads

(b) Writes

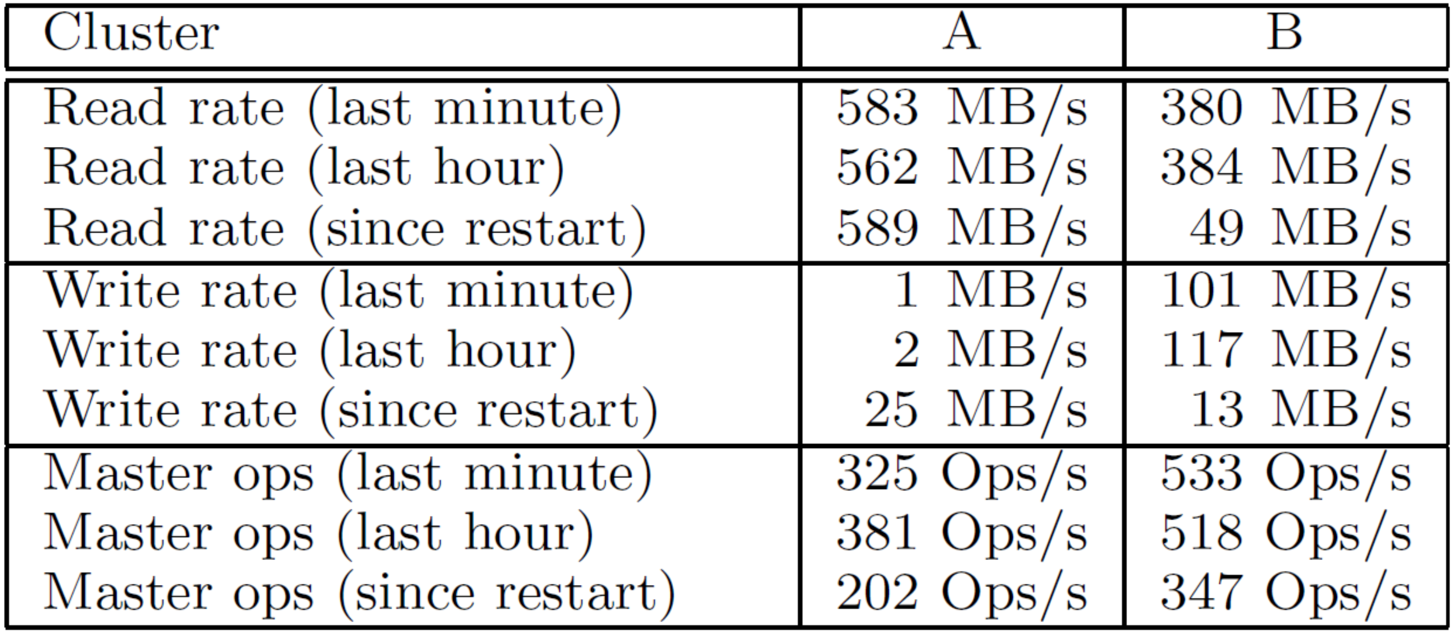
GFS Clusters

- □ Note: paper published in SOSP 2003
  - Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google file system”. In Proceedings of SOSP ‘03.



Cluster	A	B
Chunkservers	342	227
Available disk space	72 TB	180 TB
Used disk space	55 TB	155 TB
Number of Files	735 k	737 k
Number of Dead files	22 k	232 k
Number of Chunks	992 k	1550 k
Metadata at chunkservers	13 GB	21 GB
Metadata at master	48 MB	60 MB

GFS Performance



Cluster	A	B
Read rate (last minute)	583 MB/s	380 MB/s
Read rate (last hour)	562 MB/s	384 MB/s
Read rate (since restart)	589 MB/s	49 MB/s
Write rate (last minute)	1 MB/s	101 MB/s
Write rate (last hour)	2 MB/s	117 MB/s
Write rate (since restart)	25 MB/s	13 MB/s
Master ops (last minute)	325 Ops/s	533 Ops/s
Master ops (last hour)	381 Ops/s	518 Ops/s
Master ops (since restart)	202 Ops/s	347 Ops/s

Table 3: Performance Metrics for Two GFS Clusters

Summary

- File systems are evolving rapidly
- New file systems for new storage hardware and computing environments
  - Further reading:
    - FAST: USENIX Conference on File and Storage Technologies
- Next lecture
  - I/O