One Big File Is Not Enough:

A Critical Evaluation of the Dominant Free-Space Sanitization Technique

Simson L. Garfinkel and David J. Malan
Division of Engineering and Applied Sciences
Harvard University

June 28, 2006

`rm` and `DEL` don't actually delete information.

The New York Times

Deleting is easy, but hard drive tells all

Investigators using digital forensic programs retrieve important evidence for court cases.

Eric A. Taub / New York Times

It was only a single digit in a 20-page Microsoft Word contract between two partners, but Scott Cooper earned his fee several years ago when he found it.

Cooper, a computer forensics expert, learned that the numeral "1" had been scrubbed in some later versions of this digital document.

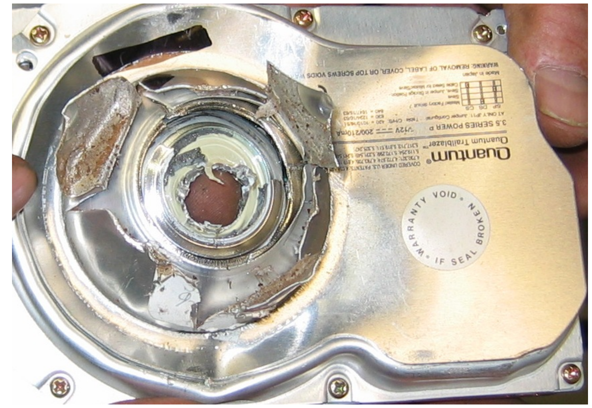
This gave his client, a partner in a software firm that had recently been sold, just a 5 percent rather than a 15 percent share in the company. If the change had gone undetected, the partner would have received $32 million rather than his rightful $96 million payout.

What the partner did not realize was that digital data rarely goes away, even when erased. "It is extremely difficult to completely delete all evidence from a hard drive," says John Colbert, the chief executive of Guidance Software, which makes a widely used program that helps retrieve digital evidence.

There are a variety of ways to prevent data recovery:

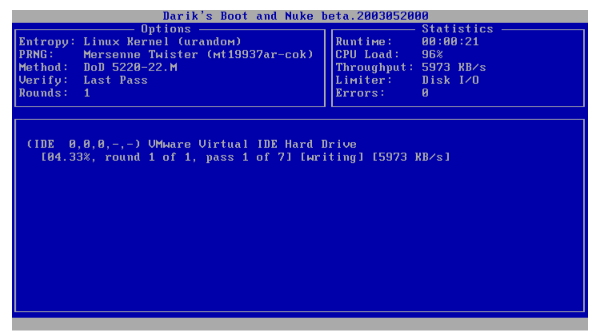
✘Physical Destruction

`http://edrsolutions.com/`



✘Overwrite every sector

`http://dban.sourceforge.net/`



✘Just use the disk. ["Understanding Data Lifetime via Whole System Simulation," Chow *et al.*, 2004]

Our research evaluates a common technique for selectively overwriting deleted data.

✔Create “one big file.” (64K writes)

✔(Create “little files.”) (64K + 512 byte writes)

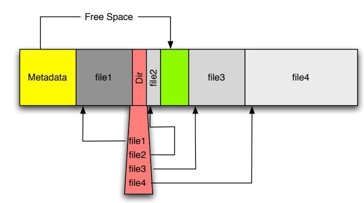
This approach is used by many disk sanitizers:

- Microsoft’s `CIPHER.EXE /W`
- The Apple Disk Utility
- Russinovich’s “SDelete”(`http://www.sysinternals.com/`)
- Tolvanen and Trant’s “Eraser” (`http://www.heidi.ie/eraser/`)

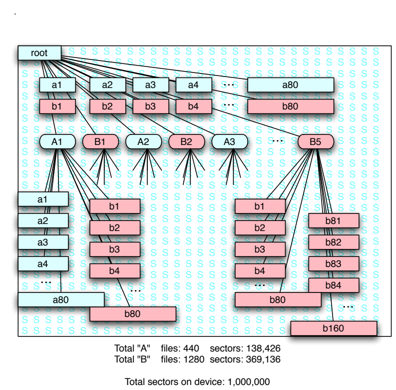
If the adversary can read blocks through the disk drive’s API, how effective is “one big file?”

Our paper evaluates the effectiveness of vendor tools and two “big file” approaches.

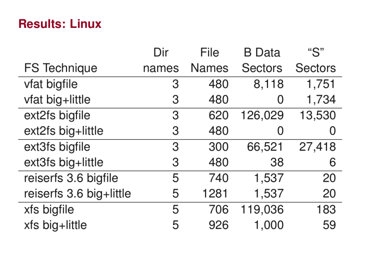
1. Slack space and free space.



2. Experiment



3. Results

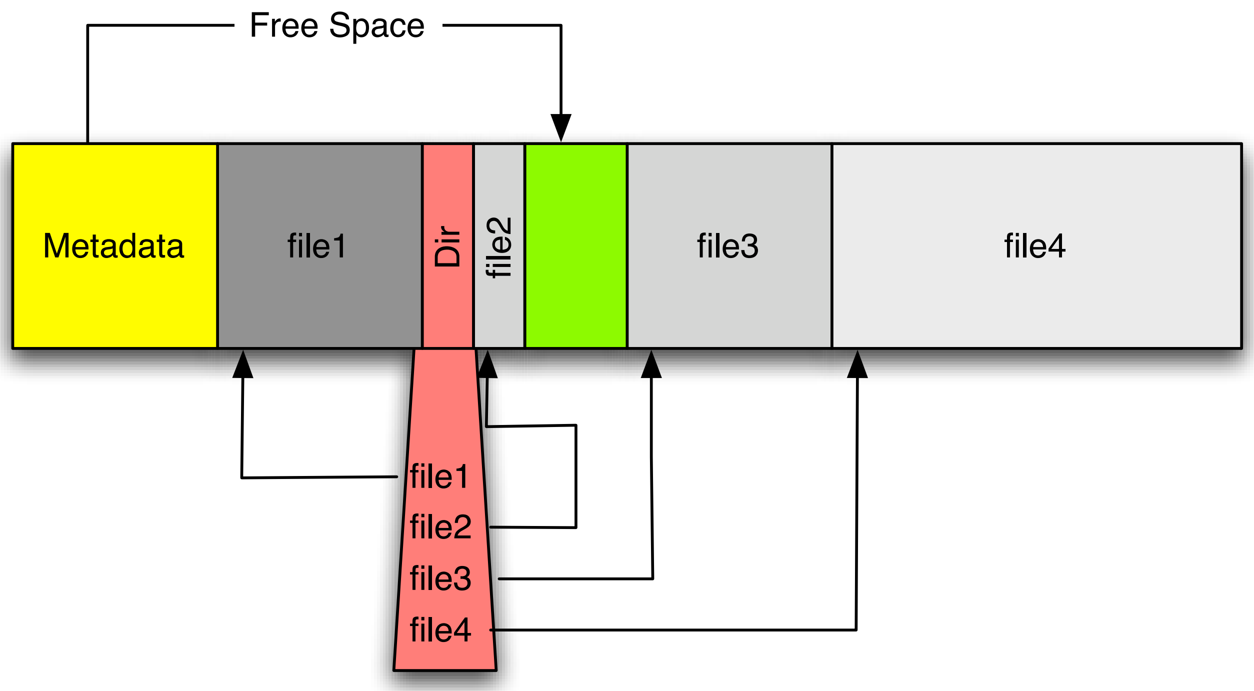


Results: Linux

FS Technique	Dir names	File Names	B Data Sectors	“S” Sectors
vfat bigfile	3	480	8,118	1,751
vfat big+little	3	480	0	1,734
ext2fs bigfile	3	620	126,029	13,530
ext2fs big+little	3	480	0	0
ext3fs bigfile	3	300	66,521	27,418
ext3fs big+little	3	480	38	6
reiserfs 3.6 bigfile	5	740	1,537	20
reiserfs 3.6 big+little	5	1281	1,537	20
xfs bigfile	5	706	119,036	183
xfs big+little	5	926	1,000	59

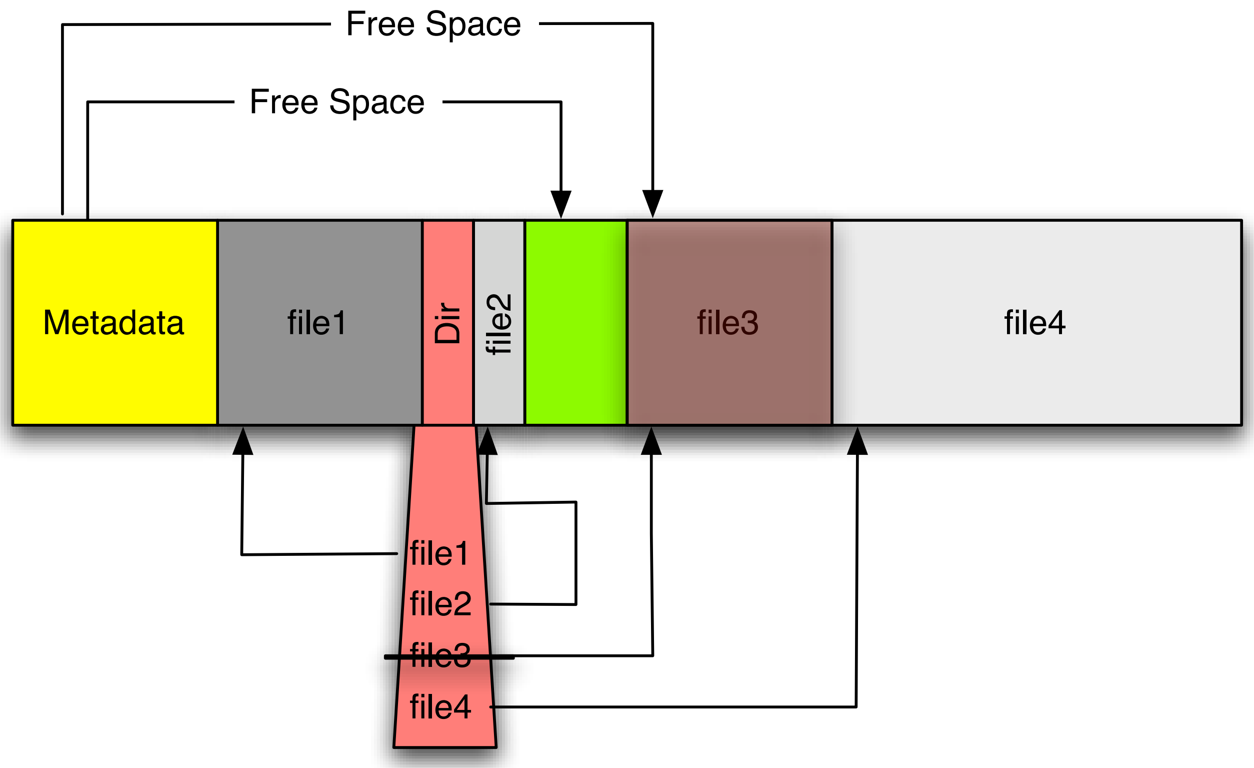
4. Improved design for file sanitization

The Free Space Sanitization Problem:



The hard drive has **metadata**, **file data**, **directory data**, and **free space**.

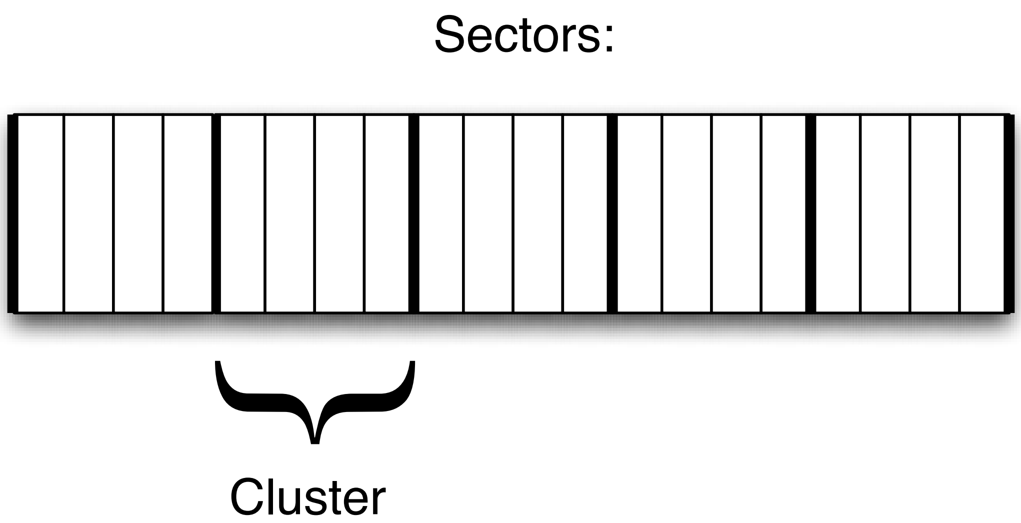
The Free Space Sanitization Problem:



% rm file3

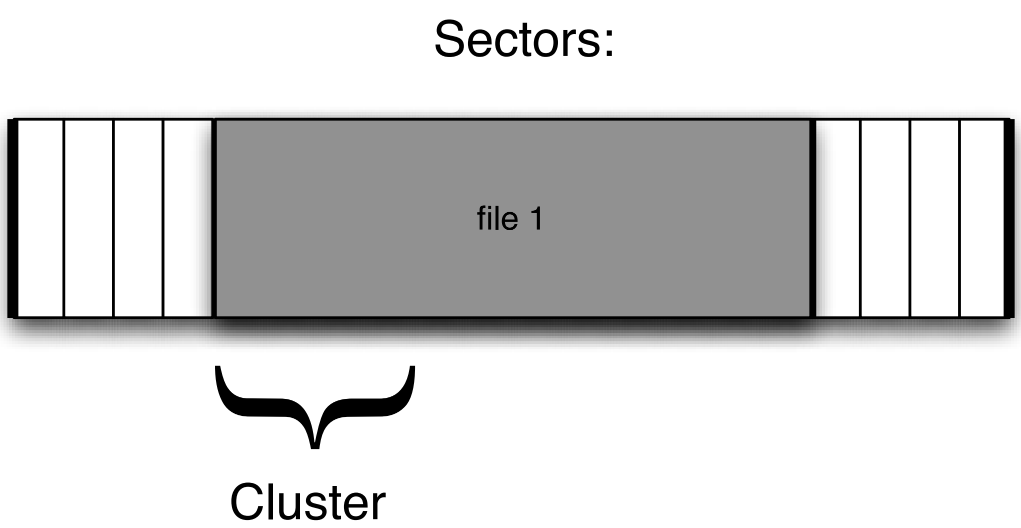
Deleting files deletes the directory entry but leaves the file's data.

The Slack Space Sanitization Problem:



Disks are read and written in *sectors* but allocated in *clusters.*

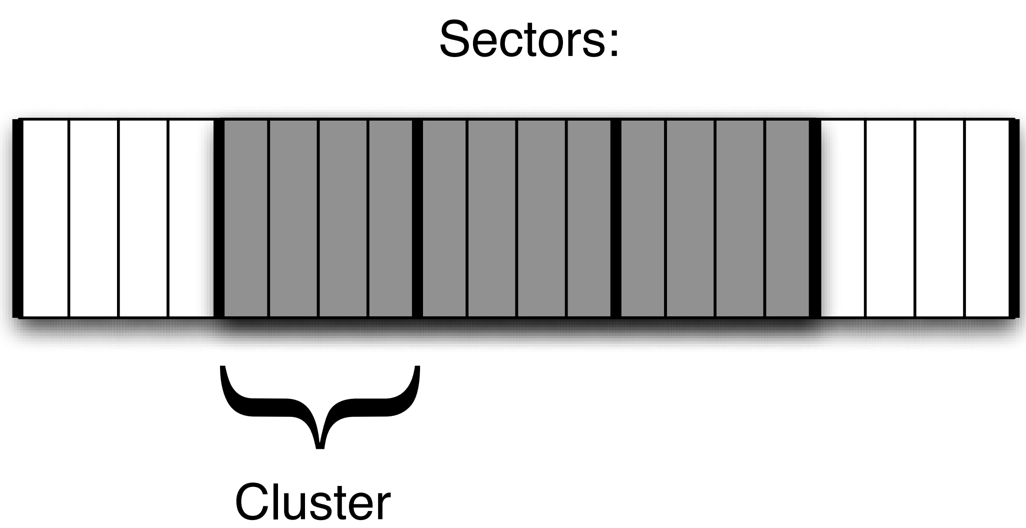
The Slack Space Sanitization Problem:



% download file1

Files can occupy an entire cluster.

The Slack Space Sanitization Problem:

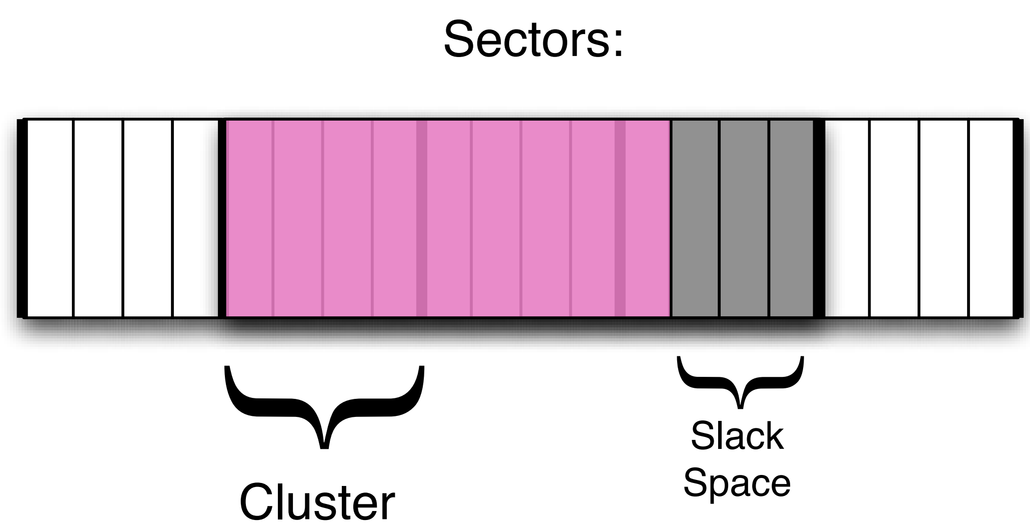


% download file1

% rm file1

When the file is deleted, the clusters are free for reallocation.

The Slack Space Sanitization Problem:



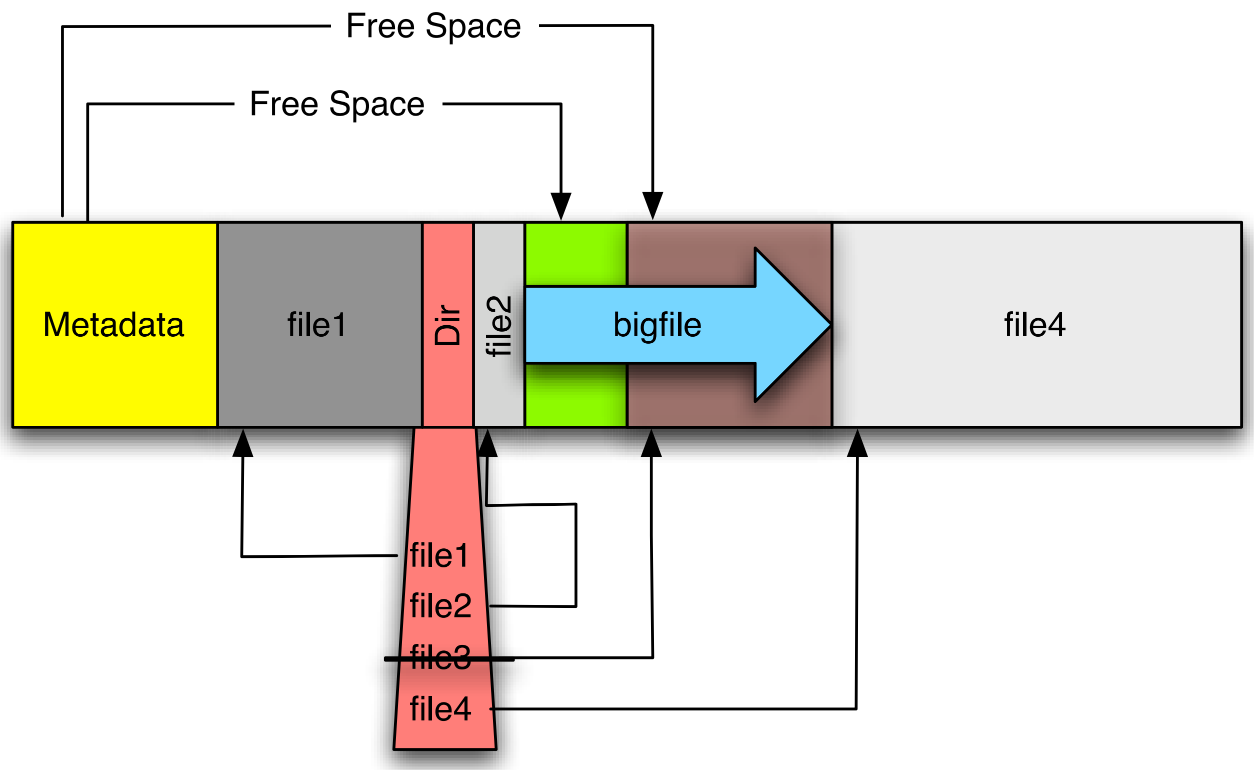
% download file1

% rm file1

% download file2

New files cannot access the slack space behind existing files.

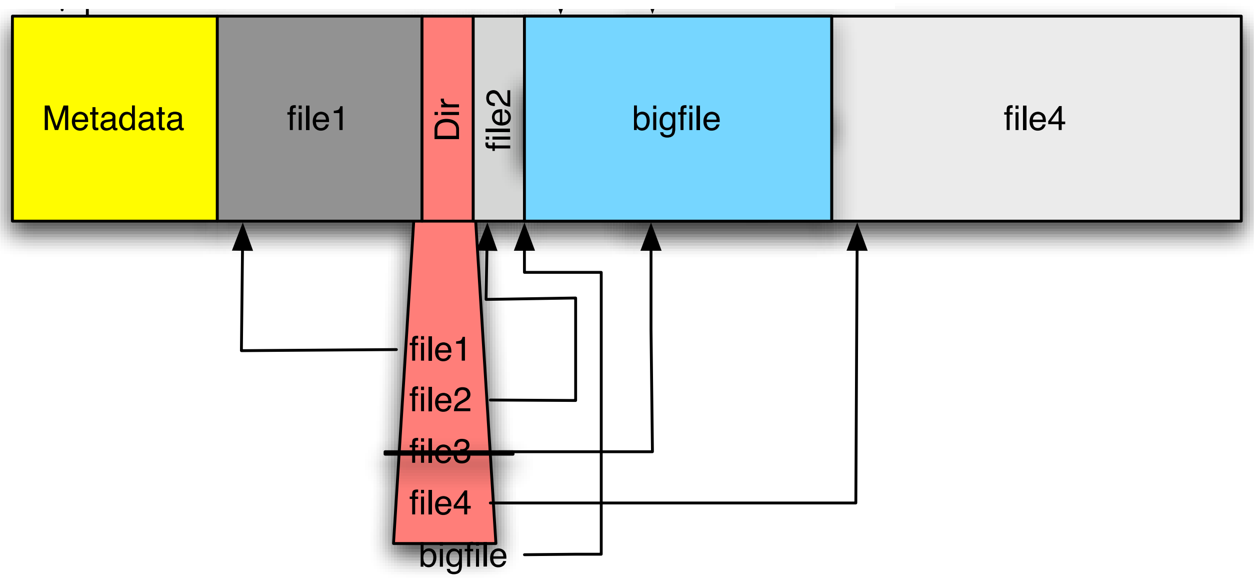
Writing a “big file” to the disk *should* overwrite the unallocated sectors.



% rm file3

% cp /dev/zero 'bigfile'

Writing a "big file" to the disk *should* overwrite the unallocated sectors . . .

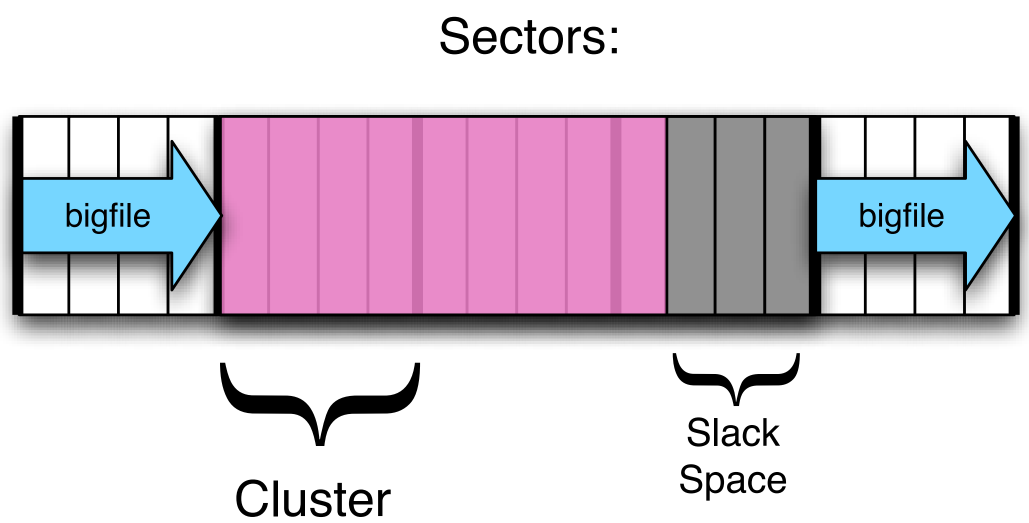


% rm file3

% cp /dev/zero 'bigfile'

. . . assuming that the "big file" can access all of the sectors.

We hypothesized that the "big file" could not access the slack space.



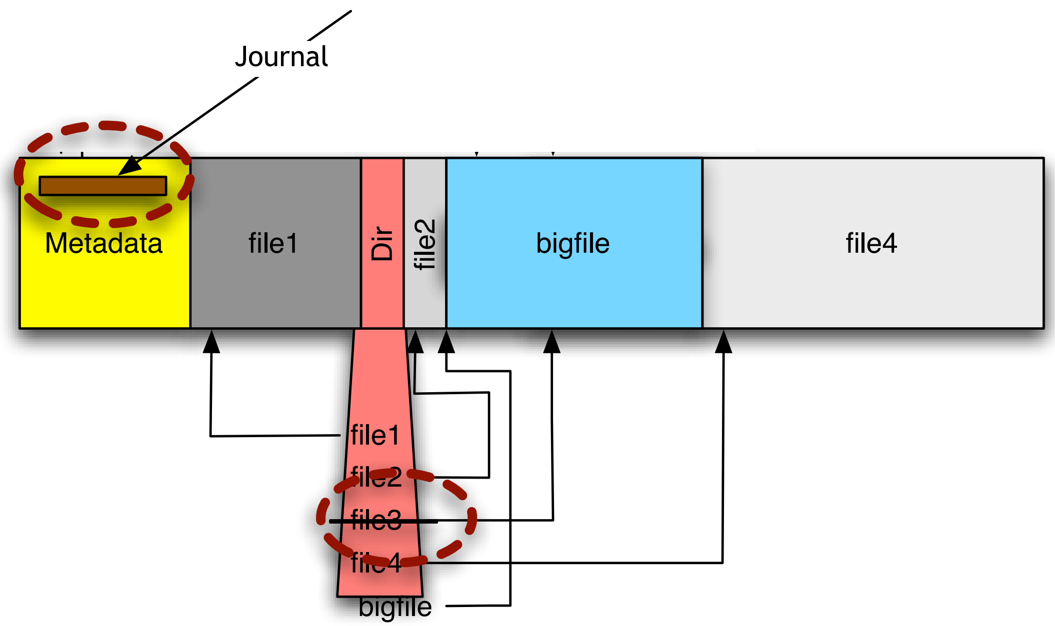
% download file1

% rm file1

% download file2

% cp /dev/zero bigfile

We also hypothesized that the “big file” could not access the metadata.

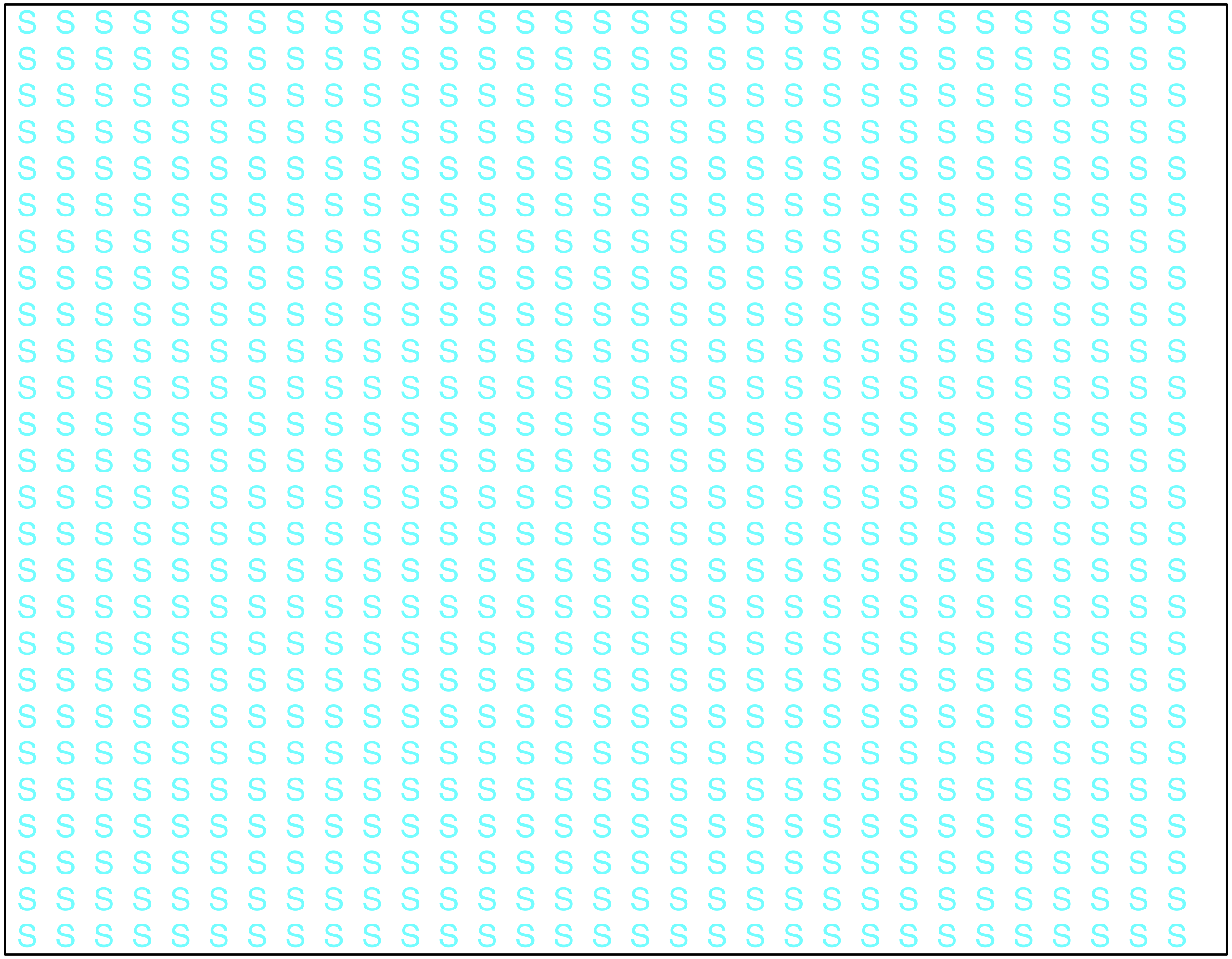


% rm file3

% cp /dev/zero 'bigfile'

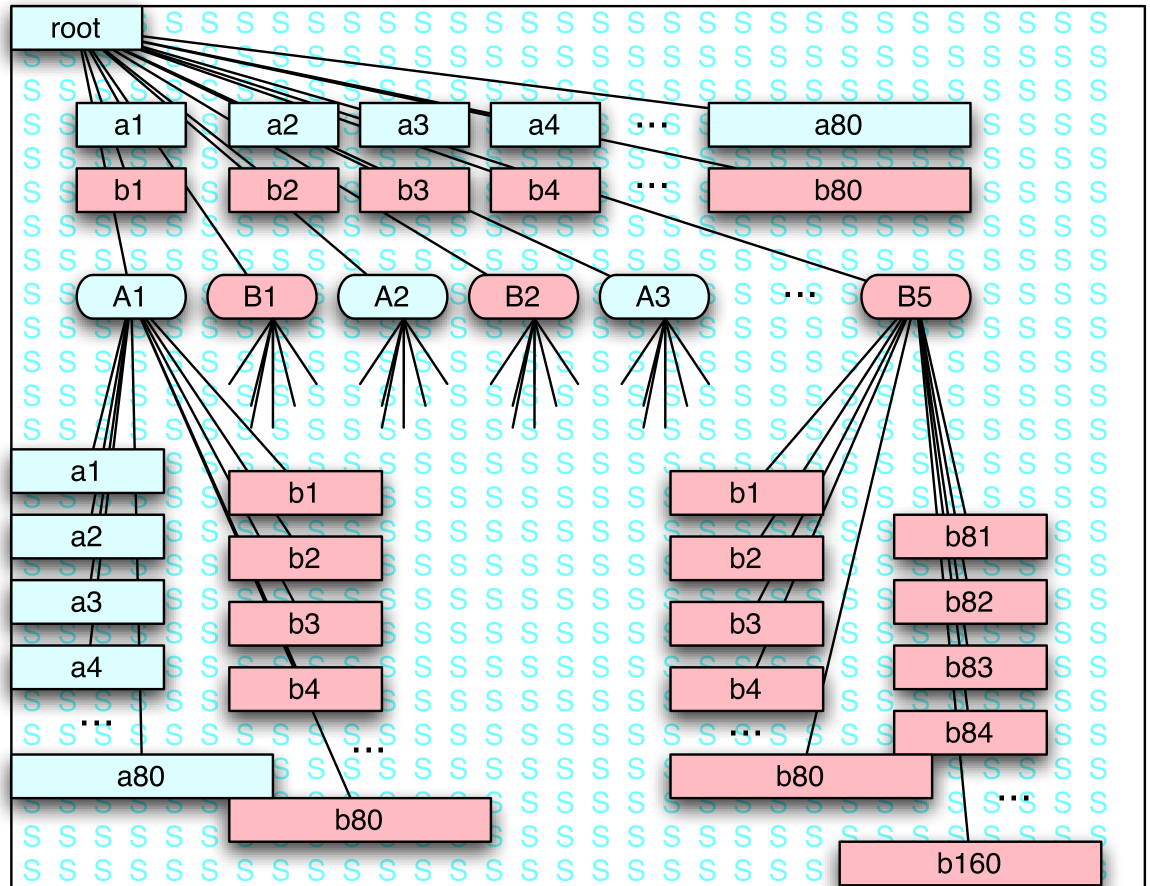
✔Old directory entries
✔Journals and Logfiles
✔Odd-sized clusters

Our experiment: Start with an "S" file.



This file contains all of the "slack space."

Create a set of "A" and "B" directories and files.

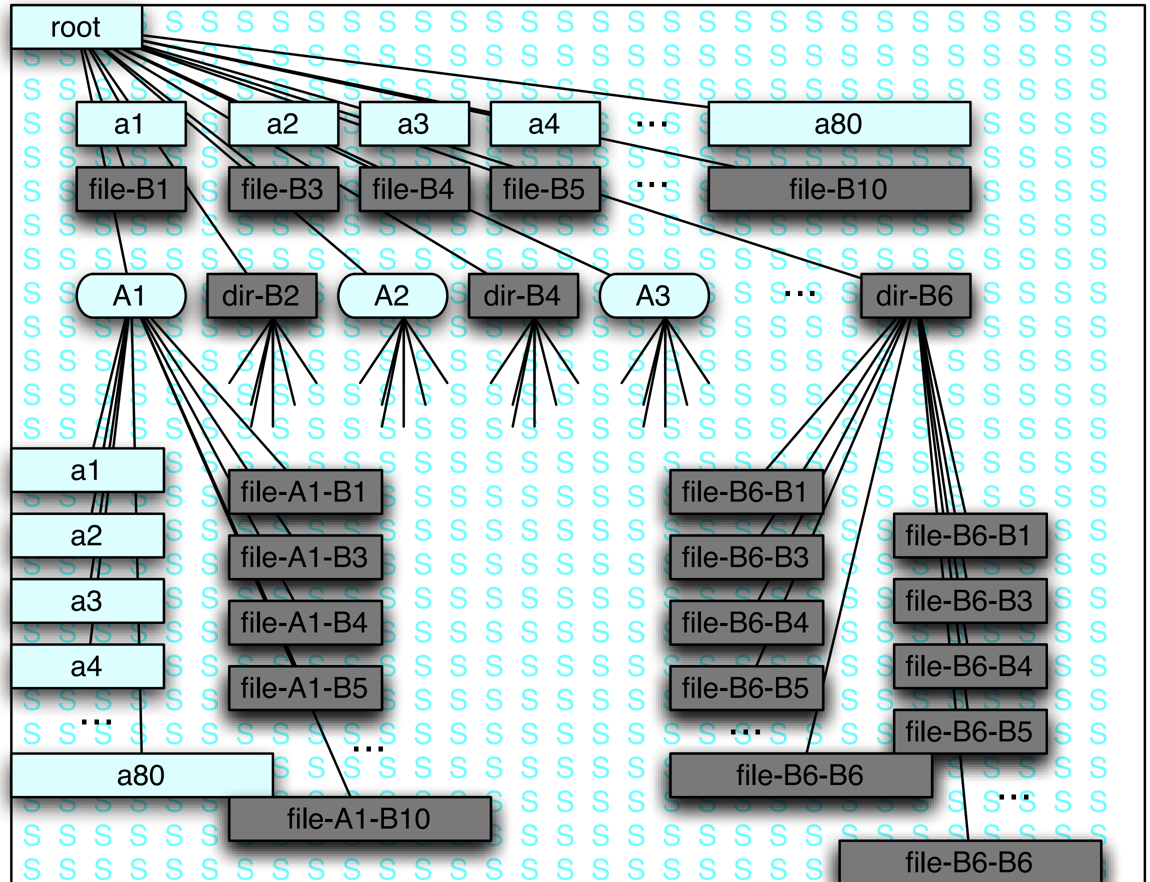


Total "A" files: 440 sectors: 138,426
Total "B" files: 1280 sectors: 369,136

Total sectors on device: 1,000,000

File sizes range from 113 bytes to 1.5MB

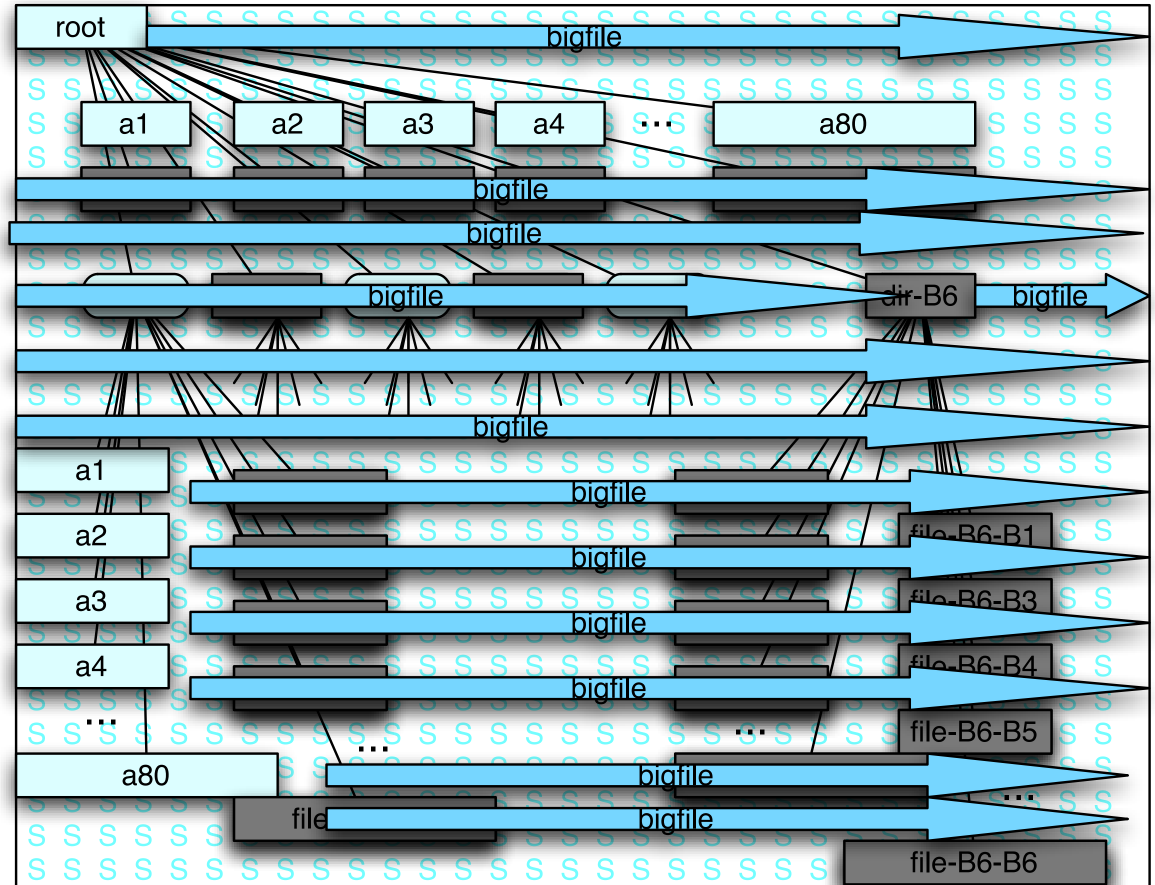
Delete the “B” files and directories.



Total "A"	files: 440	sectors: 138,426
Total "B"	files: 1280	sectors: 369,136

Total sectors on device: 1,000,000

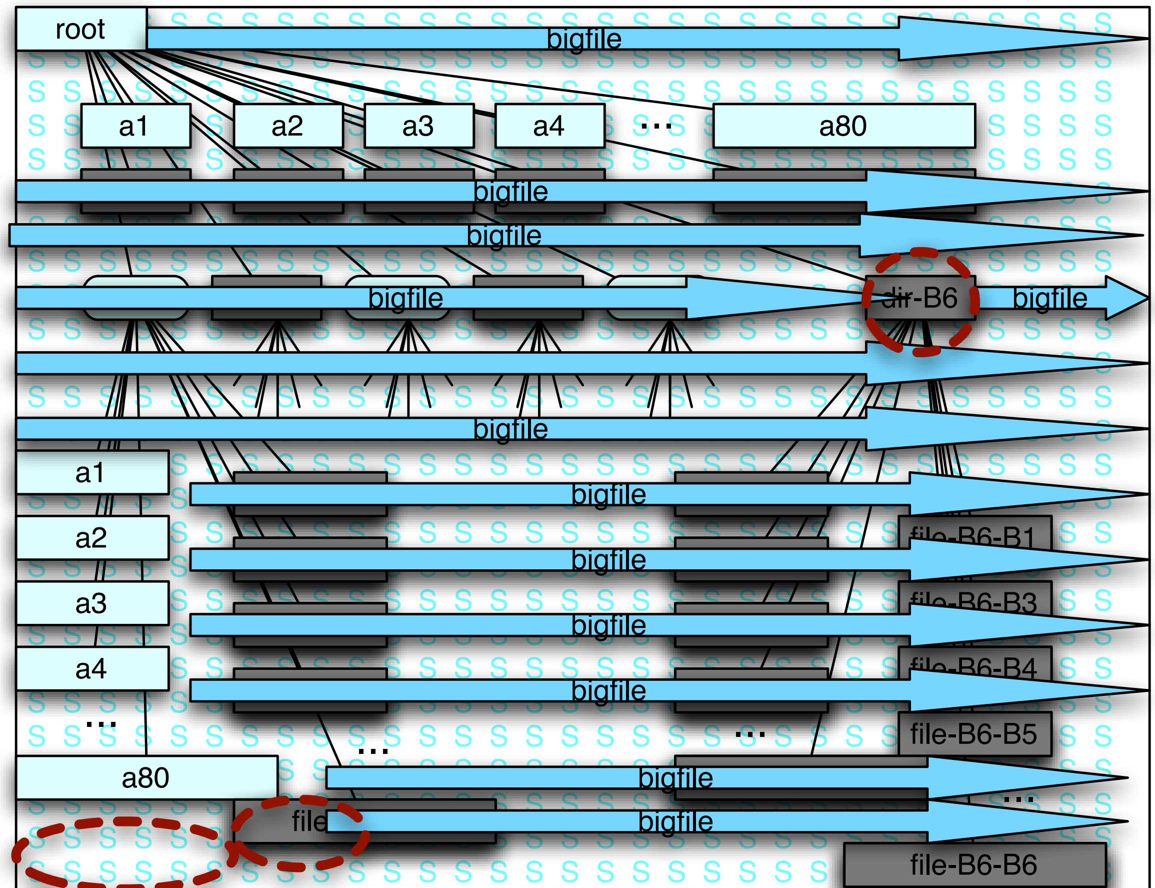
Create a big file.



Total "A" files: 440 sectors: 138,426
Total "B" files: 1280 sectors: 369,136

Total sectors on device: 1,000,000

Any "B" files that are found are not sanitized.



Total "A" files: 440 sectors: 138,426
Total "B" files: 1280 sectors: 369,136

Total sectors on device: 1,000,000

Any "S" sectors that are found are slack-space.

We investigated two techniques:
bigfile and big+little.

bigfile

1. Open a file.
2. Write 64KB chunks until writes fail.

big+little

1. Do “bigfile” technique.
2. Open a file; write 512B chunks until fail.
3. Repeat #2 until new files cannot be created.

We also evaluated vendor tools where possible.

Note: We are not discussing

- Recovering data from swap space.
- Physical remapping of sectors by the drive.
- Recovering overwritten data.

We don't consider these because they are not available through the drive API.

Results: FAT32

Technique (OS)	Dir names	File Names	“B” Data Sectors	“S” Sectors
bigfile (XP):	5	480	75	1,763
big+little:	5	480	0	1,734
`CIPHER.EXE`	5	480	0	1,734
bigfile (Mac OS)	5	1279	6	0
big+little	5	1279	0	0
Disk Utility	5	1278	0	0
bigfile (Linux)	5	1278	0	1,734
big+little	5	1278	0	1,734
bigfile (FreeBSD)	5	1278	16	56
big+little	5	1278	0	0

Sanitization is inconsistent between implementations.
All implementations leave file names.

Results: NTFS

Technique	Dir names	File Names	B Data Sectors	“S” Sectors
bigfile (XP)	5	1280	75	9
big+little:	5	1273	75	0
`CIPHER.EXE`	5	1273	65	0
Eraser	5	292	0	0
SDelete	5	1262	60	0

NTFS is harder to sanitize than FAT; tools are inconsistent.

Results: Linux

FS Technique	Dir names	File Names	B Data Sectors	“S” Sectors
fat bigfile	5	1278	0	1,734
fat big+little	5	1278	0	1,734
ext2fs bigfile	5	1278	6	0
ext2fs big+little	5	1278	0	0
ext3fs bigfile	5	1280	3,567	224
ext3fs big+little	5	1280	24	0
reiserfs 3.6 bigfile	5	1281	1,460	96
reiserfs 3.6 big+little	5	1281	1,460	96
xfs bigfile	5	801	1,004	44
xfs big+little	5	801	957	44

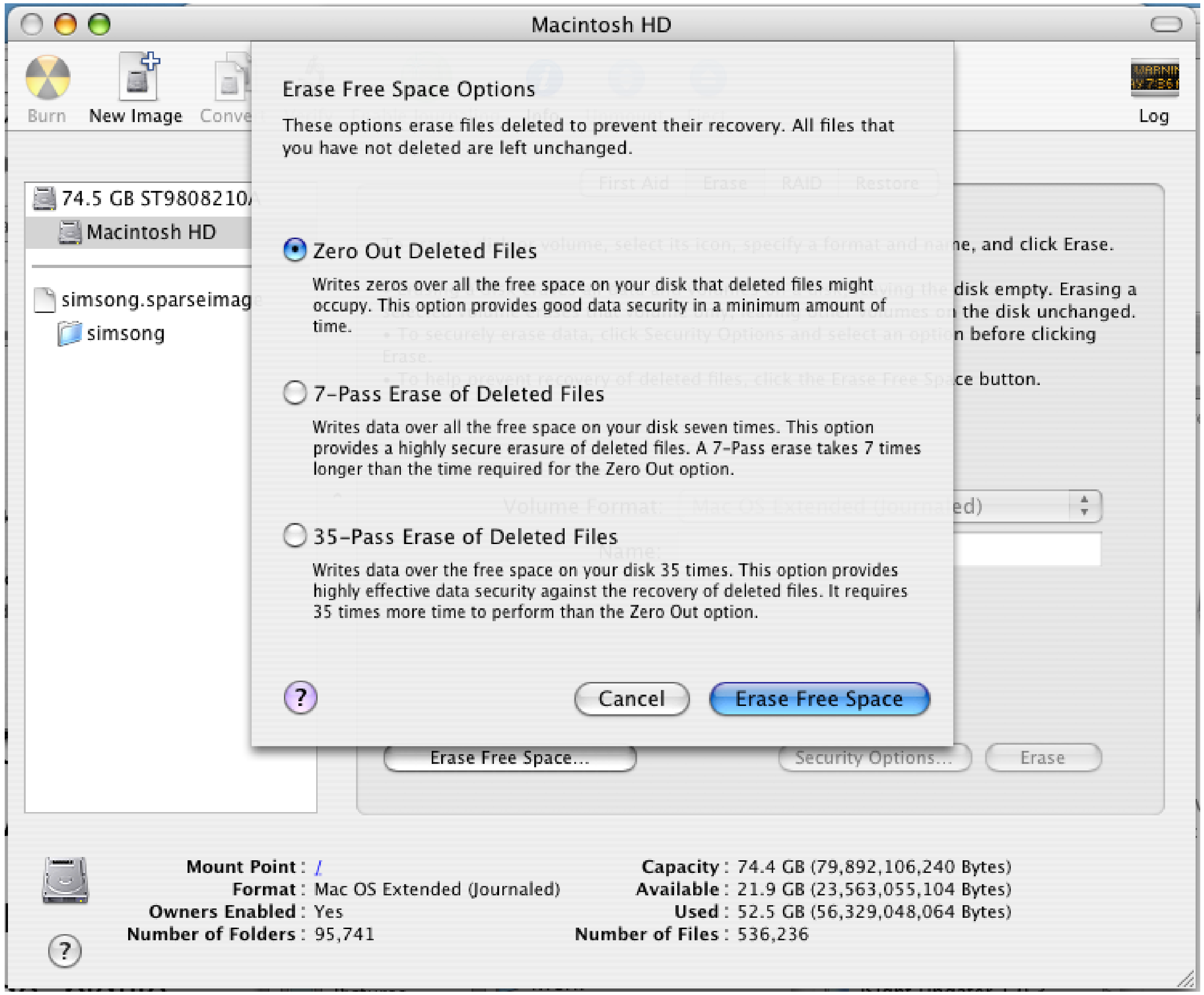
Journaled file systems are harder to sanitize.

Results: FreeBSD

FS Technique	Dir names	File Names	B Data Sectors	"S" Sectors
FAT bigfile	5	1278	16	56
FAT big+little	5	1278	0	0
UFS2 bigfile	5	1280	3,504	256
UFS2 big+little	5	1278	2,865	152

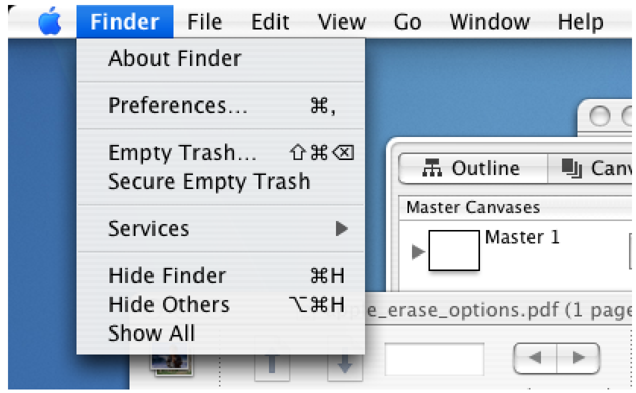
FAT is easier to sanitize than UFS.
"Little files" get many but not all sectors.

Mac OS provides an “Erase Free Space” feature.



EFS eliminated all user data, but left file names on journaled HFS.

Mac OS also provides a “Secure Empty Trash” feature.



SET works, but is slow. (7 overwrites!)

Comparison of “secure delete” approaches

FS Technique	Remnant Dir names	Remnant File Names	Remnant B Data Sectors
FAT SDelete	3	480	0
FAT Eraser	0	0	0
NTFS SDelete	5	1262	0
NTFS Eraser	5	294	0
Mac OS Secure Empty Trash	1	43	0

Better ways to sanitize:

Implement “clean delete” in:

- `ftruncate()`, `truncate()`, and `unlink()` (Linux and UNIX);
- `NtDeleteFile()` `NtSetInformationFile`(Windows).

Copy “allocated files” from drive A to drive B, then wipe A.

Background task that overwrites with NULs:

- All sectors on free list.
- All sectors in slack space.

  (Requires understanding of file system.)

Conclusions

One “big file” deletes nearly all of the “deleted” files, but:

... many file names and times are left behind.

... sometimes, complete files can be recovered.

Journaled file systems are harder to sanitize.

Vendor-provided tools appear to work through the user-level API and do not directly manipulate file system structures.

We believe that it is necessary to work at the file-system level to properly sanitize.