Hash-Based Carving: Searching media for files and file fragments with sector hashing

Simson L. Garfinkel

January 19, 2016

A bit about me

The Boston Globe
Sandstorm Enterprises

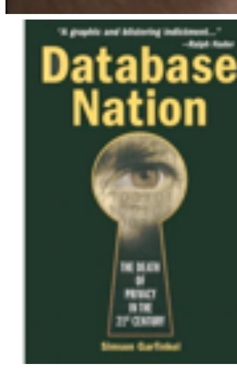
Tech Journalist:	**1985—2002**
Entrepreneur:	**1988—2002**
MIT PhD	**2002—2005**
Harvard	**2005—2006**

NPS PRAESTANTIA PER SCIENTIAM 1909

Associate Professor,
Naval Postgraduate School 2006—2015

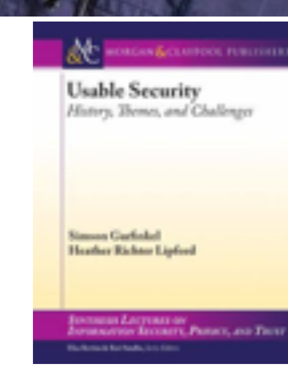
NIST National Institute of Standards and Technology U.S. Department of Commerce

Computer Scientist,
NIST 2015-



Privacy
2000

Internet of Things
2005



Security & Usability
2014

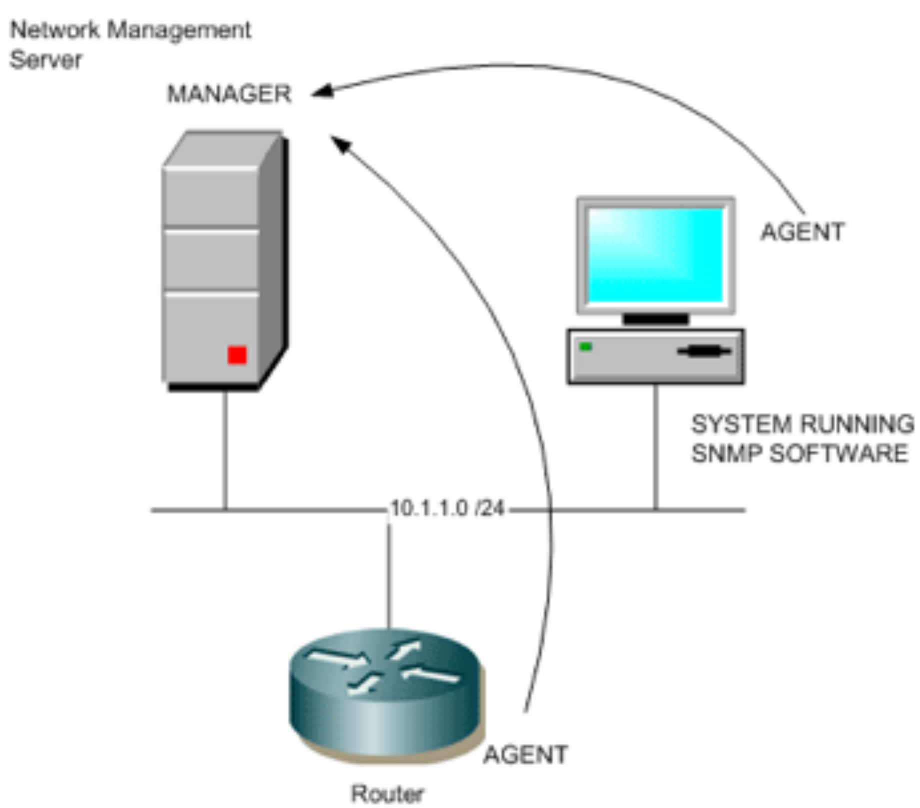
My current research:
Big Data, Privacy, and Usable Security

Release data without compromising privacy.

de-identification

differential privacy

Auditable tests for network protocols.



High DNSSEC Assurance Domain

Data overload:
A fundamental cyber & forensic challenge.

Boarder Crossings

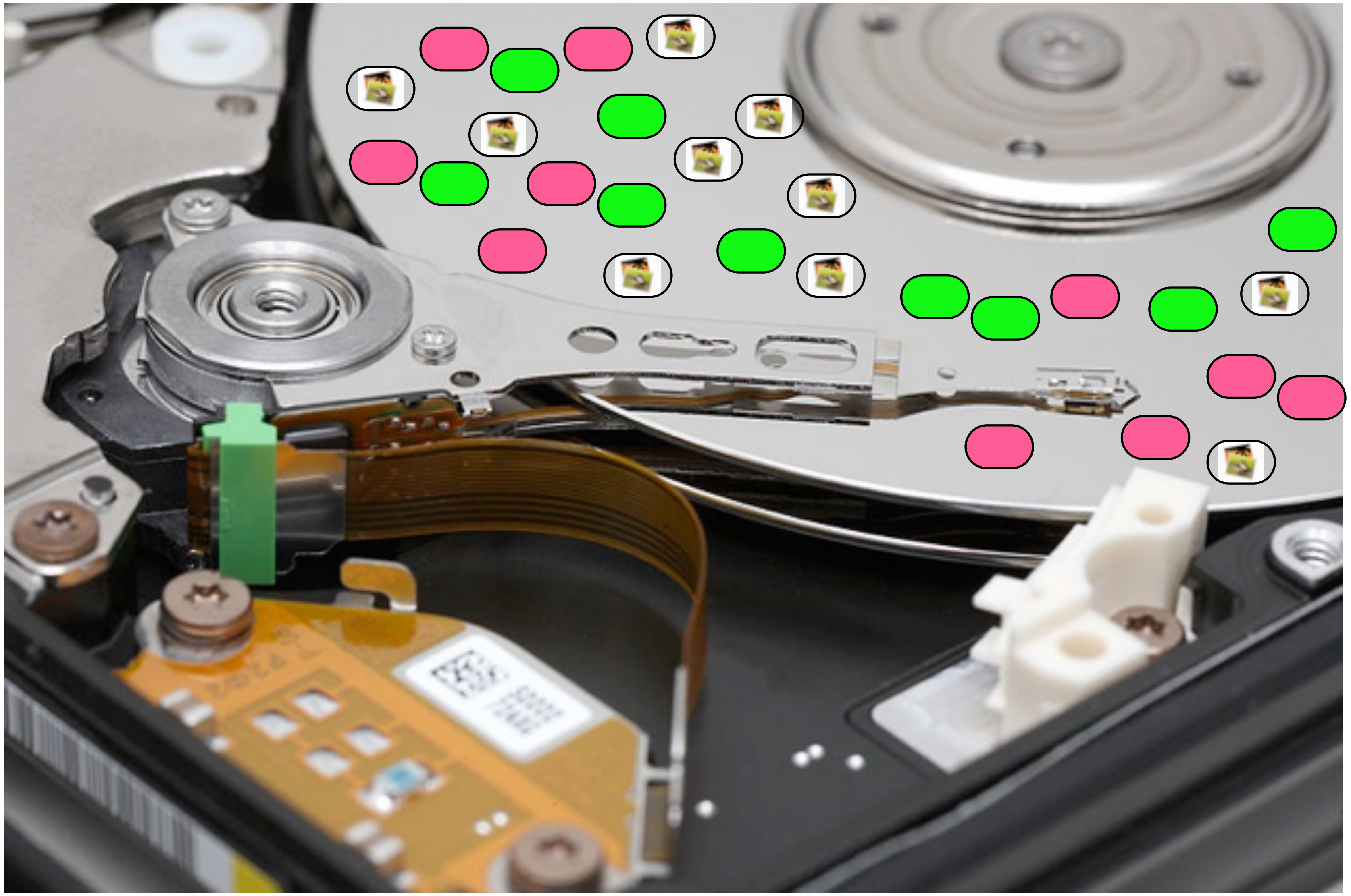
Disk:

Search & Seizure:

Cyber Security:

RAM:

Big idea (circa 2006): use random sampling to find “target” data.



It takes 3.5 hours to read a 1TB hard drive.

In 5 minutes you can read:

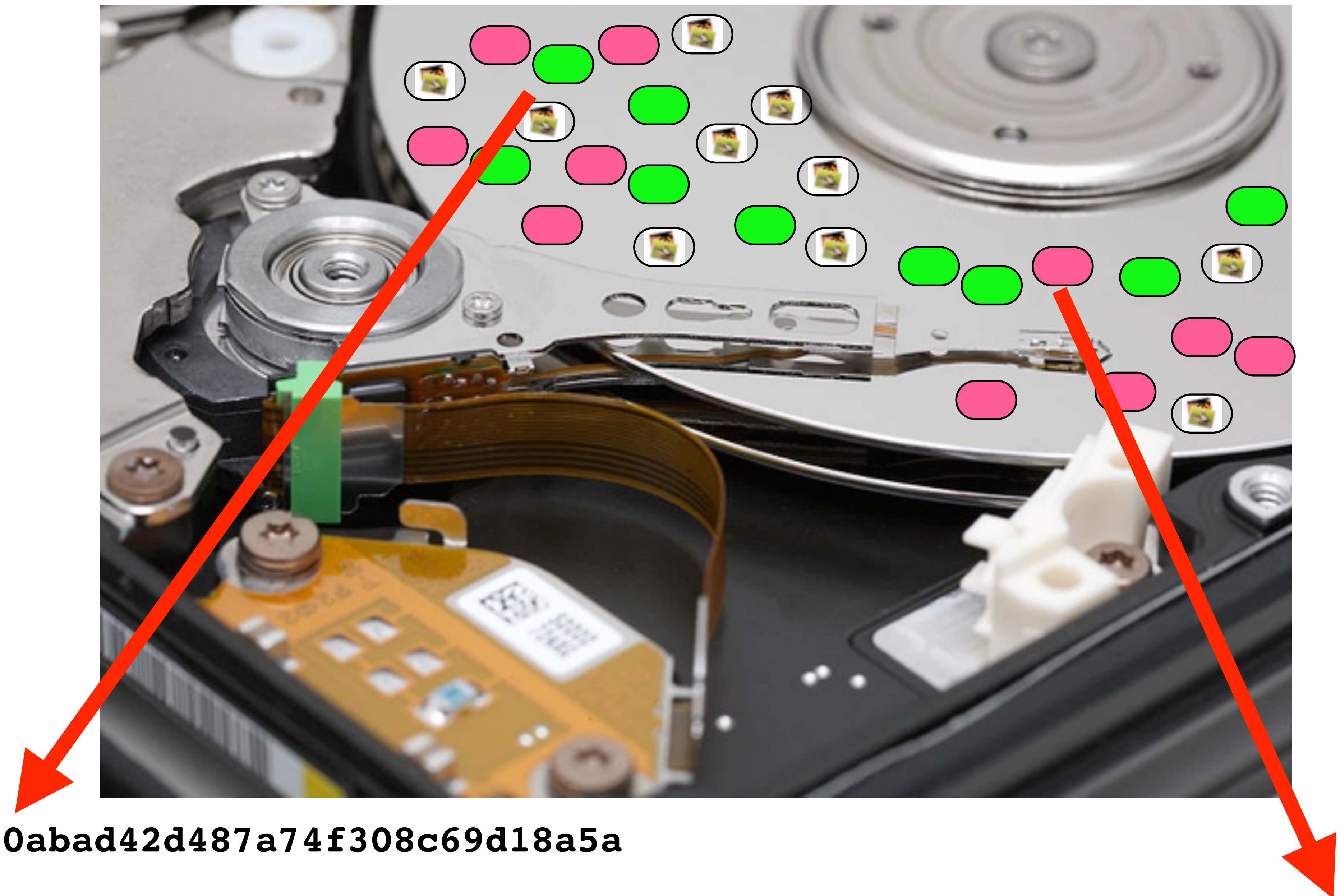
- 36 GB in one strip
- 100,000 randomly chosen 64KiB strips (assuming 3 msec/seek)

			
Minutes	208	5	5
Data	1 TB	36 GB	6.5 GB
# Seeks	1	1	100,000
% of data	100%	3.6%	0.65%

Problem: no easy way to find start & end of files

First approach:
“Block Hashes” to recognize fragments of data

1. We compute the cryptographic hash of randomly chosen blocks



`dc0c20abad42d487a74f308c69d18a5a`

`6e7f3577b100f9ec7fae18438fd5b047`

2. We search for those hashes in a database of “target block hashes”

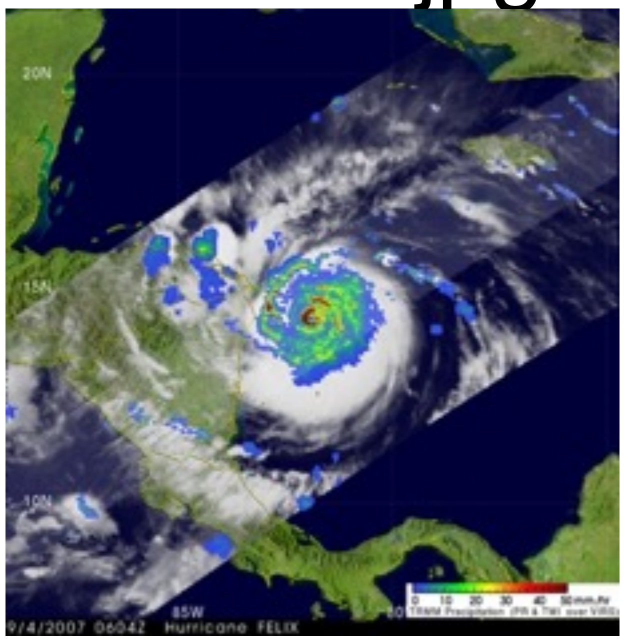
Review: Every file has a unique cryptographic hash

000107.jpg

41,572 bytes

`c996fe19c45bc19961d2301f47cabaa6`

000513.jpg



169,718 bytes

`759690467578b204d3c022330061a3eb`

000908.jpg

12,412 bytes

`244f4318543356c08c59baaa58951758`

Change 1 bit, the hash changes unpredictably.

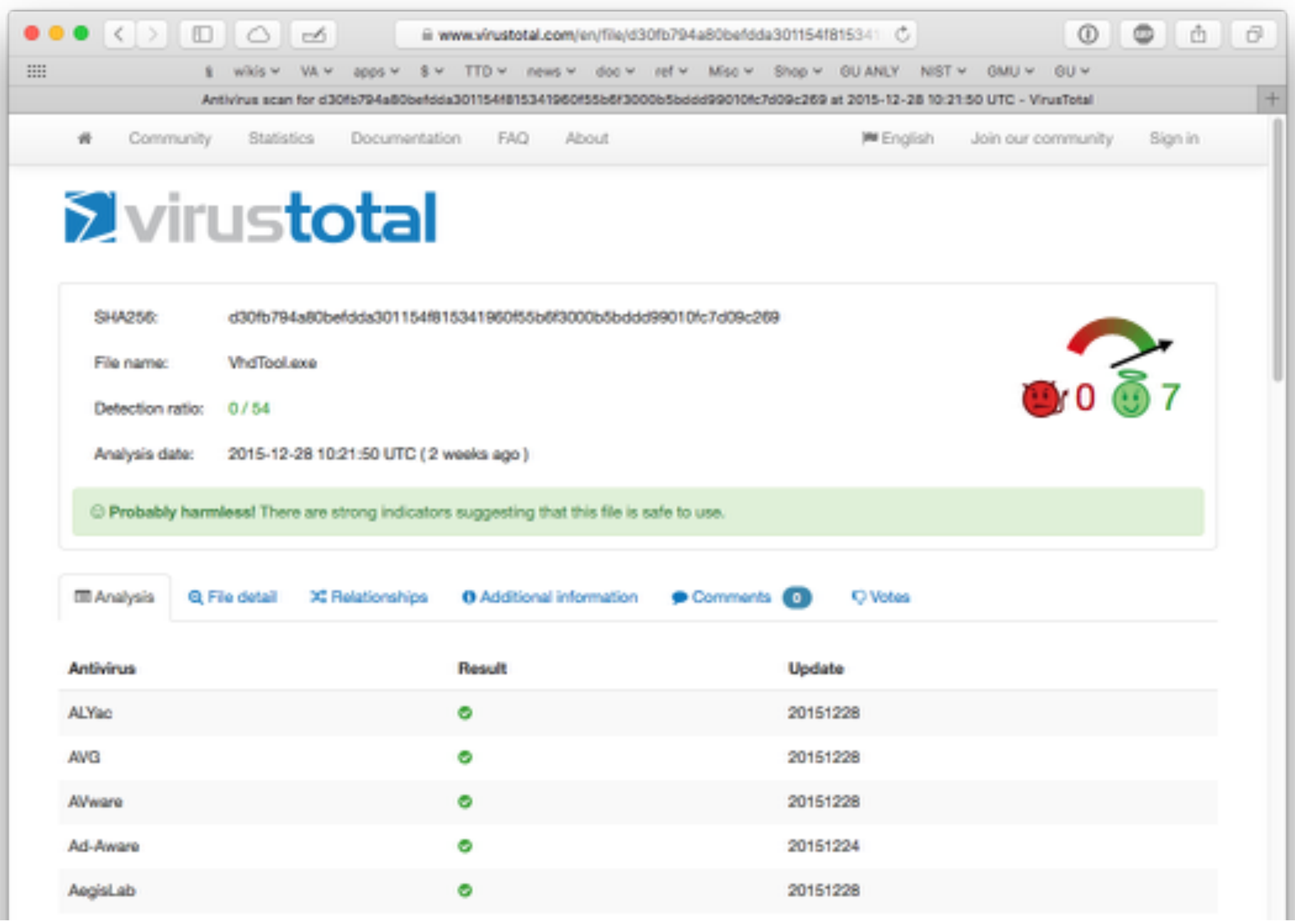
000107.jpg	000170.jpg*	000170.jpg**
		
41,572 bytes	41,572 bytes	41,572 bytes
`c996fe19c45bc19961d2301f47cabaa6`	`2b00042f7481c7b056c4b410d28f33cf`	`d16a4eb8e1cbb45eb4cb22d313b8813c`

Cybersecurity — hashes used to recognize files.

List of “good” files — Tripwire

List of “bad” files — Malware detection

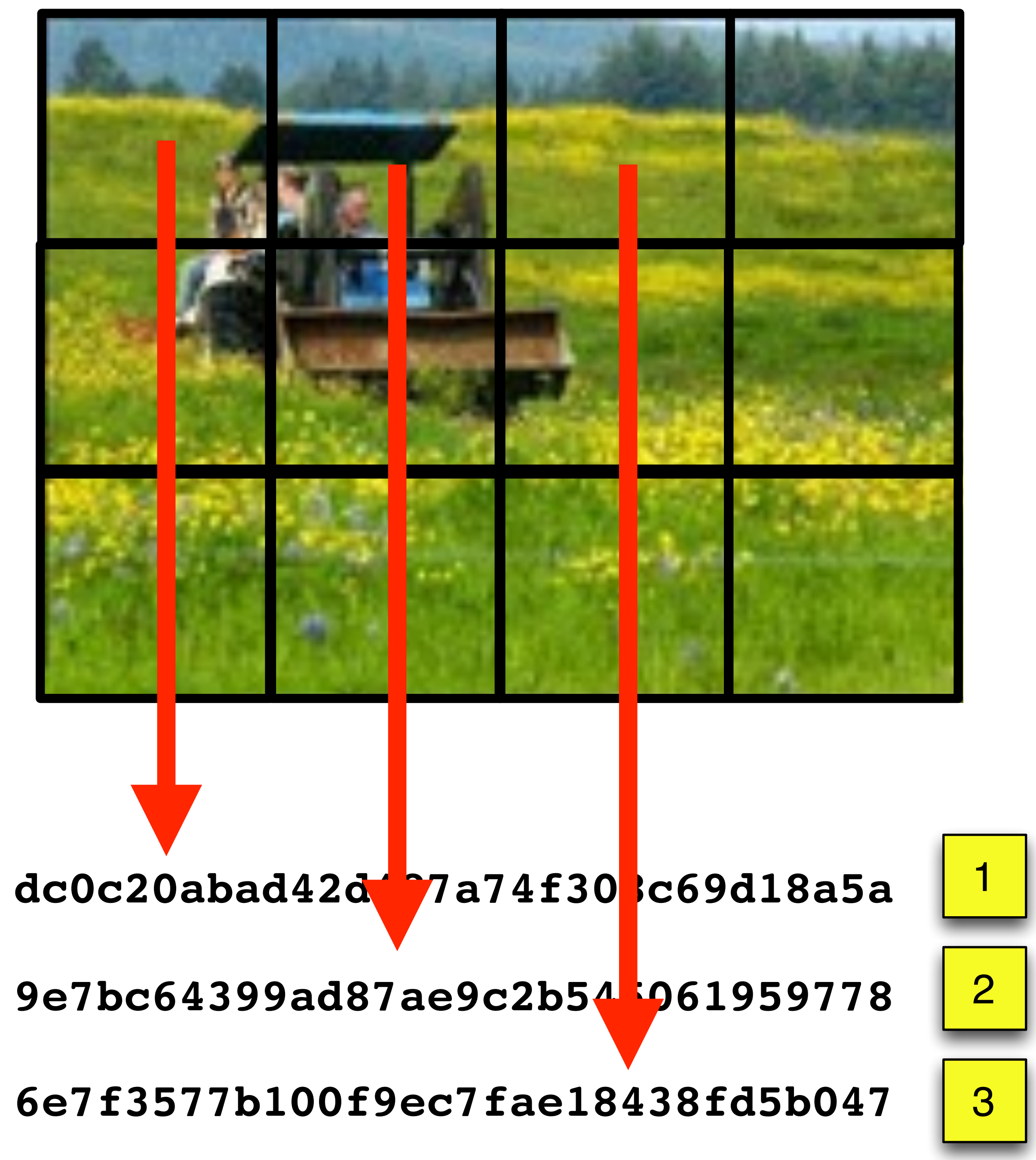
```
$ openssl md5 VhdTool.exe
MD5(VhdTool.exe)= 1b8be77e741cee1eb5fa3f9dac7c9ed1
```



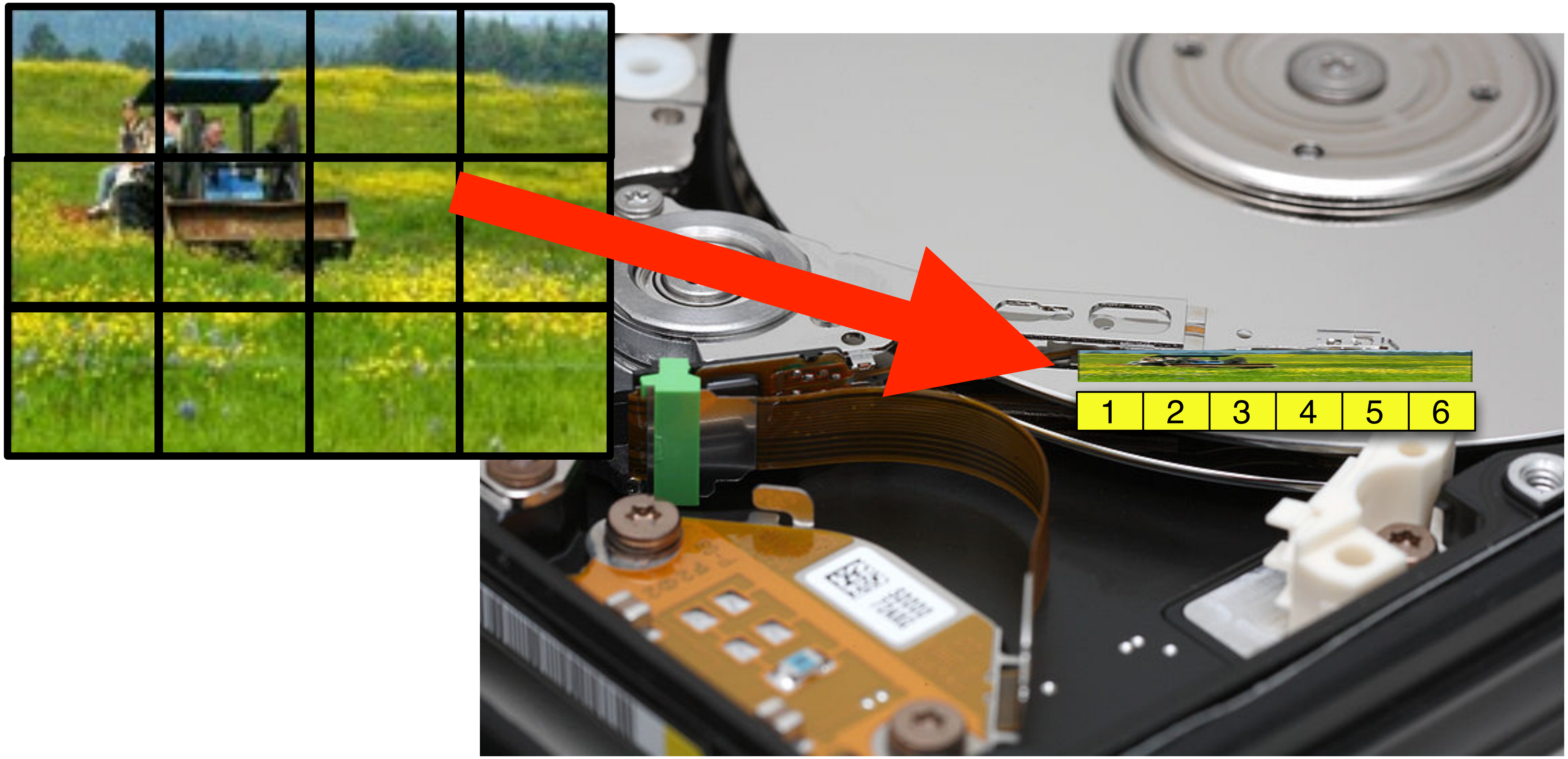
Every file can be also viewed as a sequence of blocks.

41,572 bytes ÷ 512 bytes/block = 81 blocks + 100 bytes
= 82 blocks
(w/ padding padding)

Each file block has its own hash.



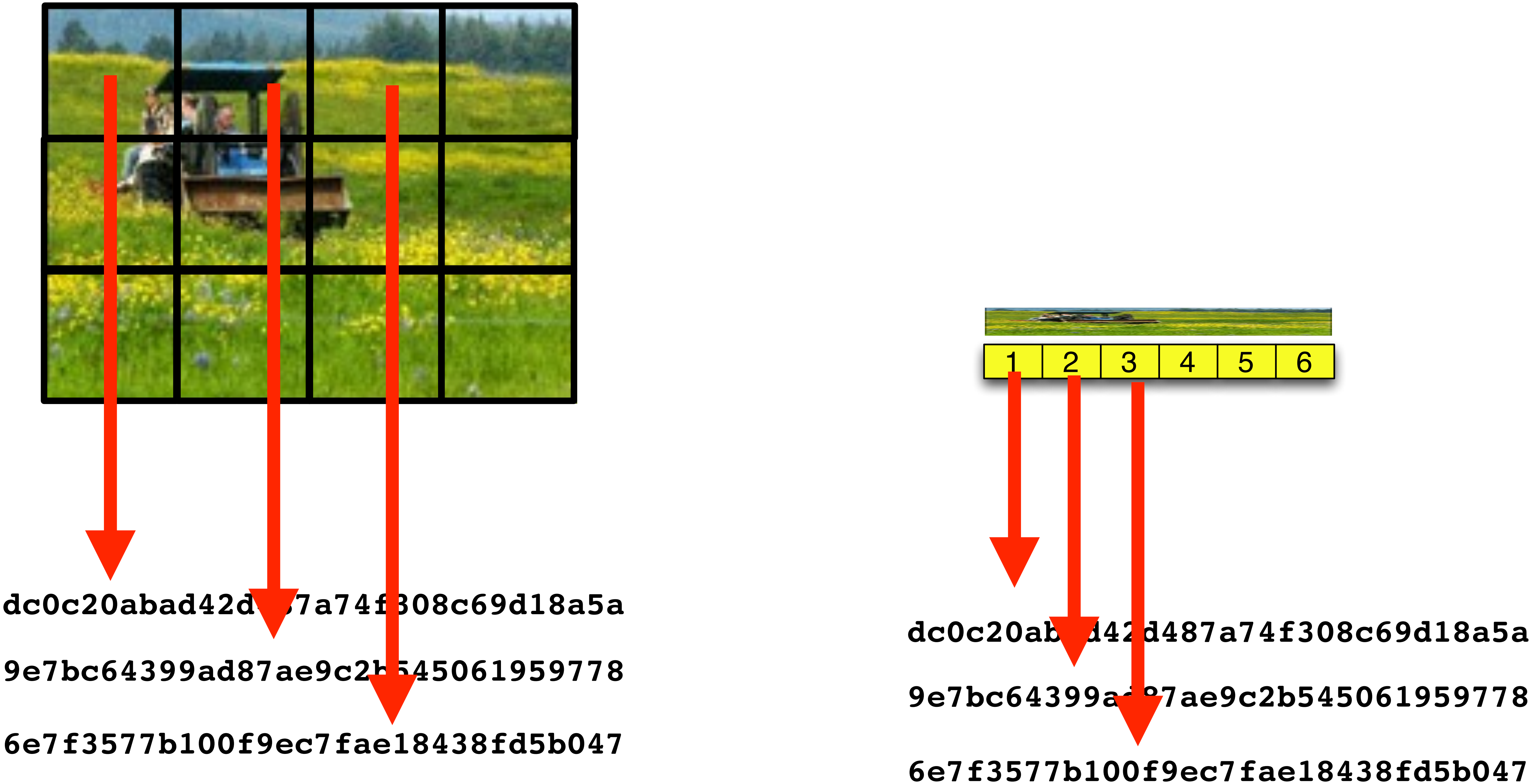
When a file is stored on a drive, file *blocks* are stored in disk *sectors*.



All modern file systems align files* on sector boundaries.

(*larger than 4KiB)

Key insight:
File block hashes are the same as disk sector hashes.



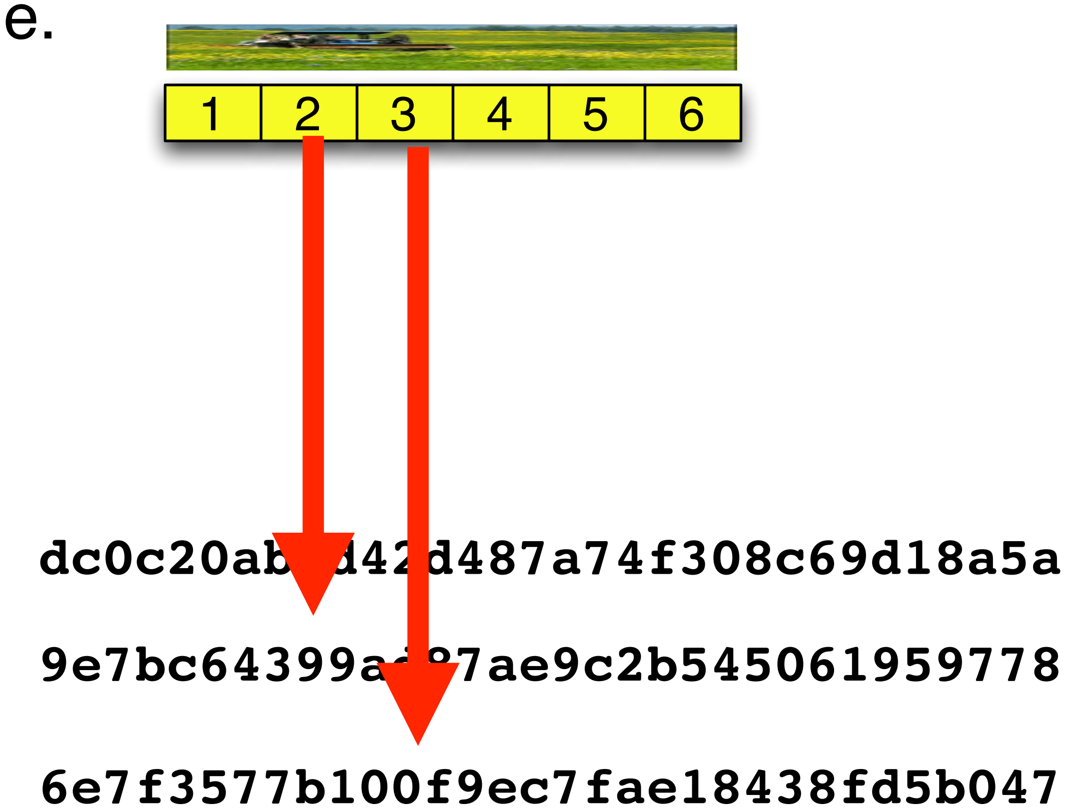
Block hashes *could* create huge capabilities.

#1 — High-speed search of target media with random sampling

- It takes 3.5 hours to search a 1TB hard drive.
- With random sampling, we could find “target data” within minutes.

#2 — Find invisible data

- Find fragments of files left in RAM or in storage.



Both applications require:

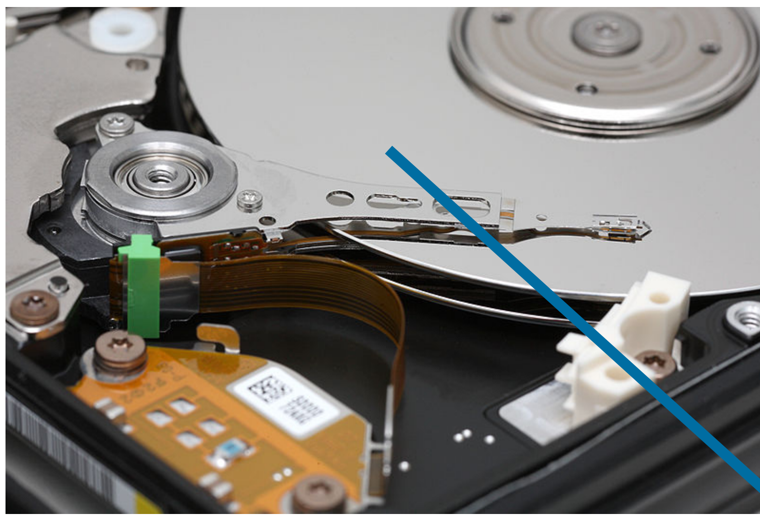
- Corpus of target data (1TB–10PB)
- Deployable billion-row database that can do 100,000 lookups/sec
- Data “identifiability”
  - —*block hashes must be distinct*

0000107.jpg:

Searching with block hashes:
the need for distinct data.

2005: Block-hashes can find files on a drive!

1. Hash every sector of the drive



2. Hash every sector of the target files

3. Look for matches

Block #	Byte Range	MD5*(block(N))
0	0- 511	dc0c20abad42d487a74f308c69d18a5a
1	512-1023	9e7bc64399ad87ae9c2b545061959778
2	1024-1535	6e7f3577b100f9ec7fae18438fd5b047
3	1536-2047	4594899684d0565789ae9f364885e303
4	...	

Using distinct sectors in media sampling and full media analysis to detect presence of documents from a corpus,
Kristina Foster, NPS Master's Thesis, 2012

2013: HashDB

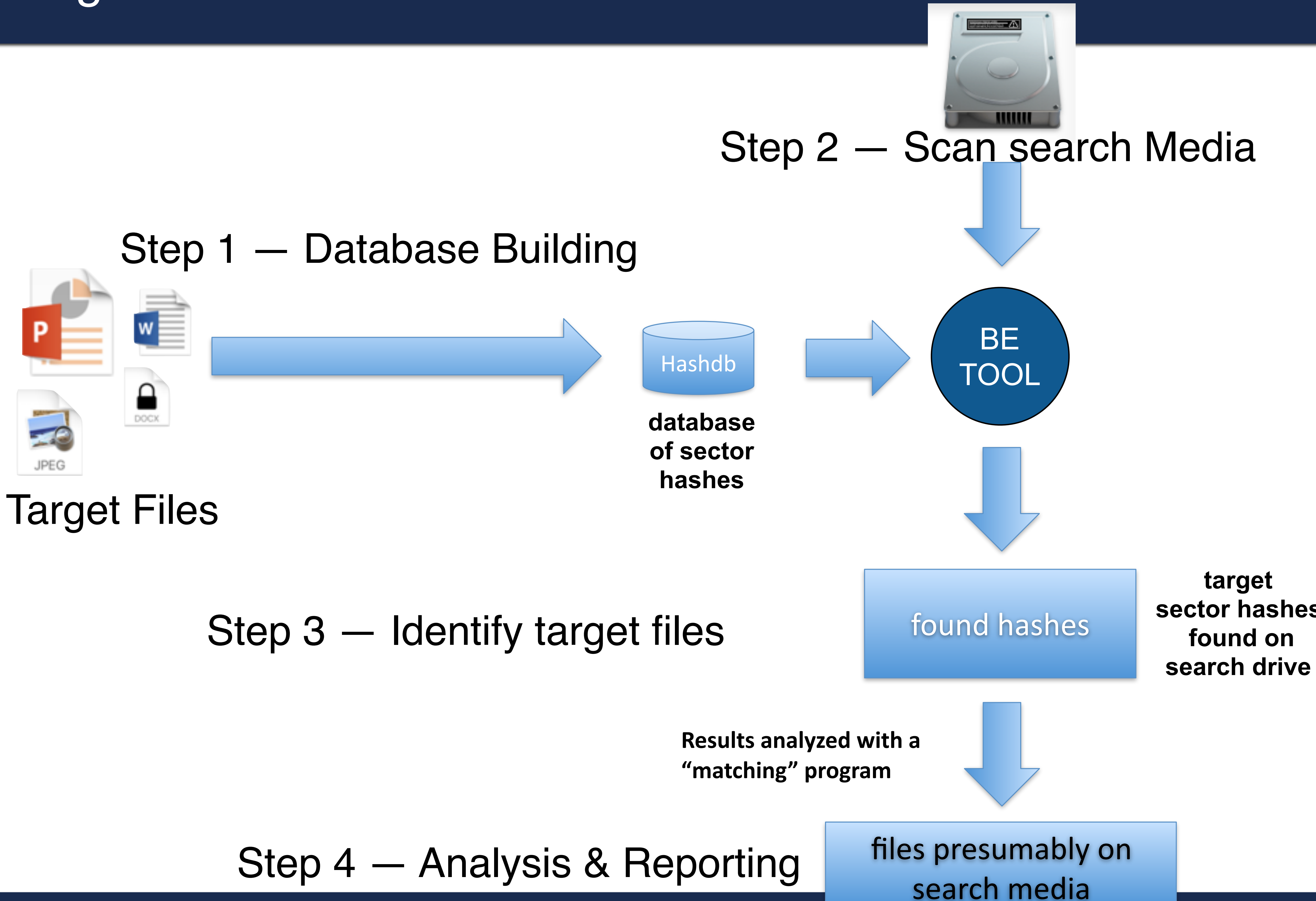
NPS created “hashdb”

- Stores 1 billion 128-bit hashes in 50GB file.
- For each hash, stores:
  - *Collection, Source File, Offset in File*
- 100,000 lookups/sec on SSD laptop
- Open Source C++ implementation

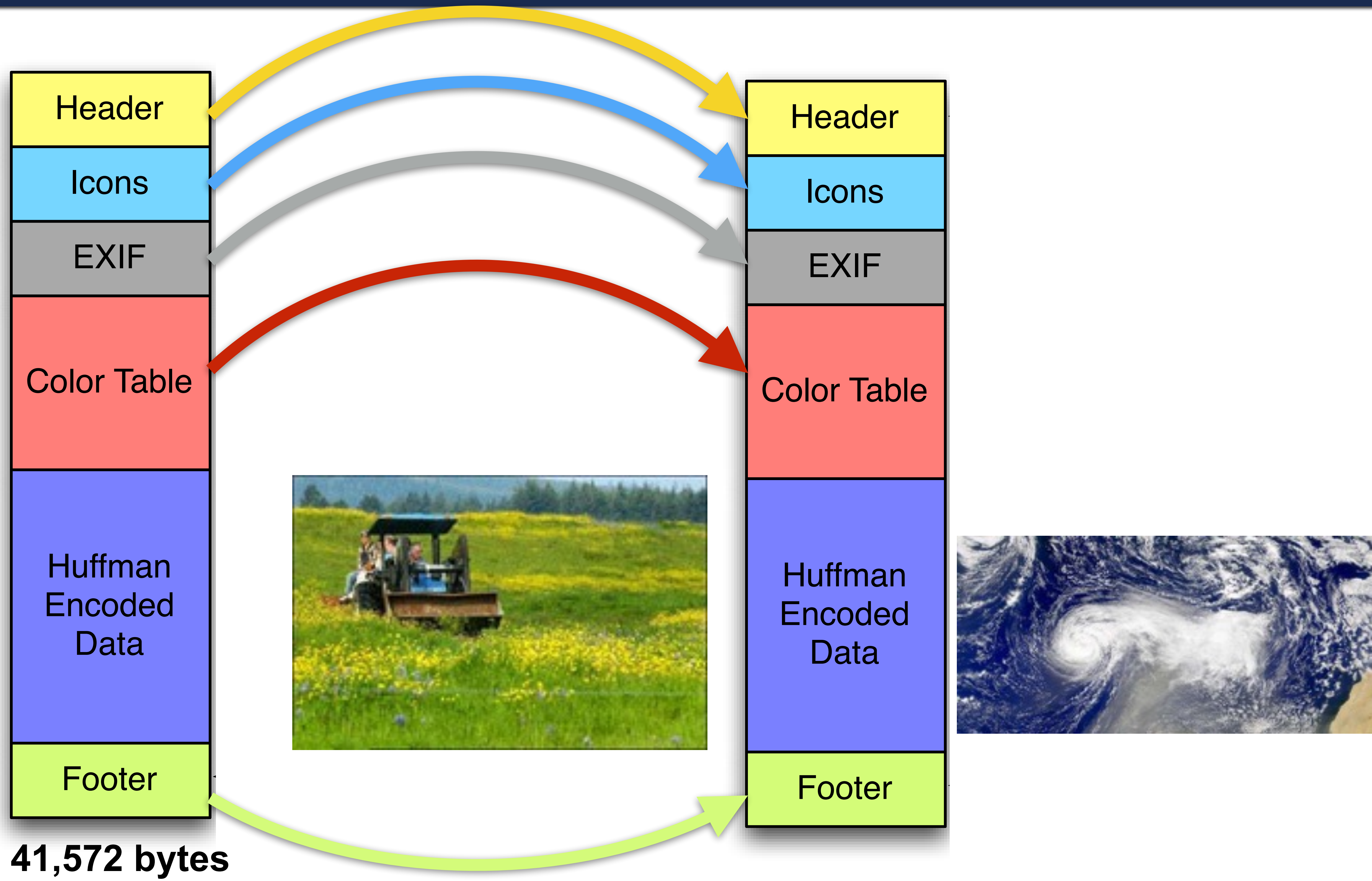
NPS integrated hashdb with “bulk_extractor”

- High-performance digital forensics tool.
- Deployed and used world-wide.
- Open Source.

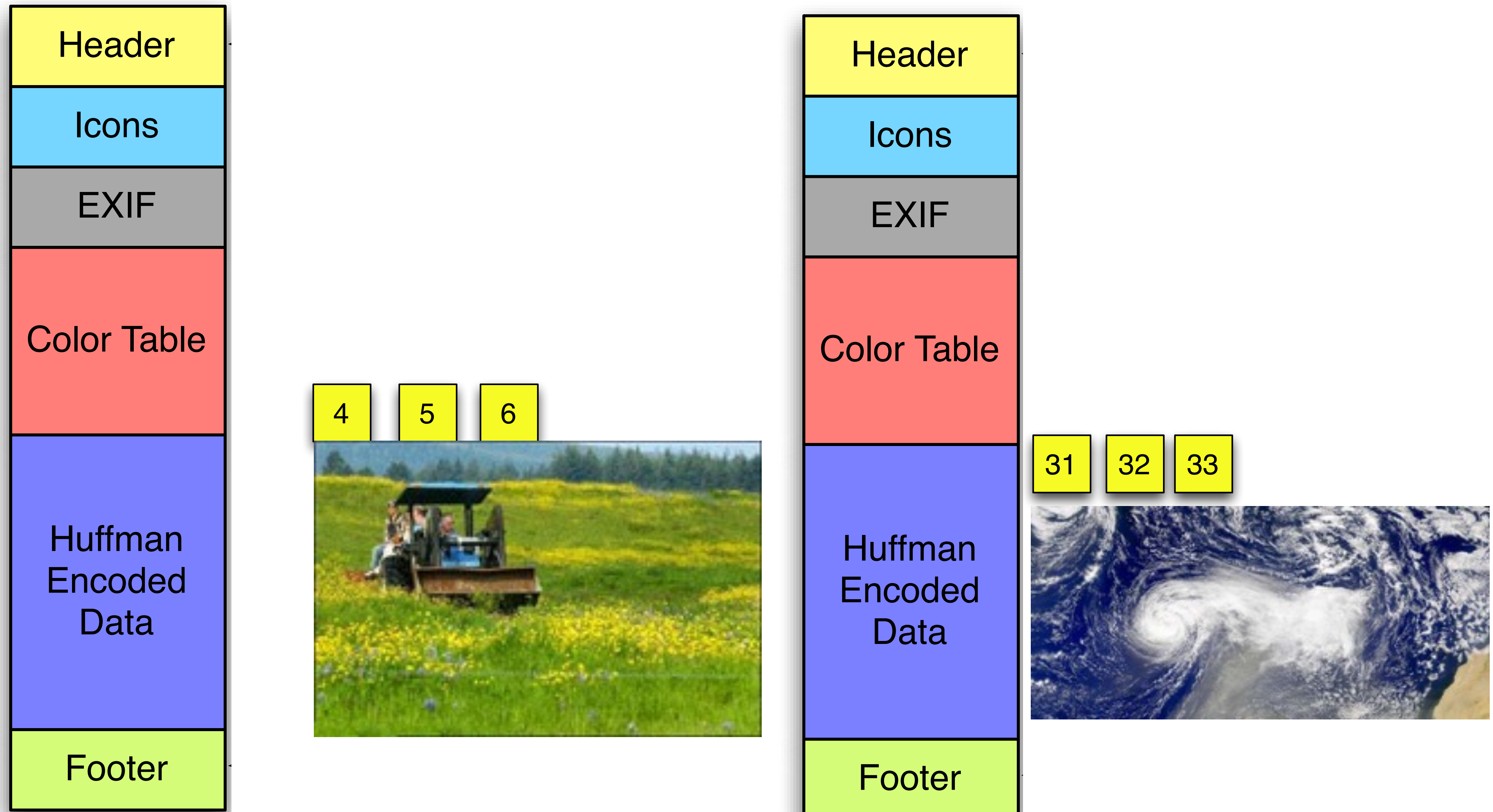
Target Architecture



Problem — files have internal structure. They are not bags of “high entropy.”

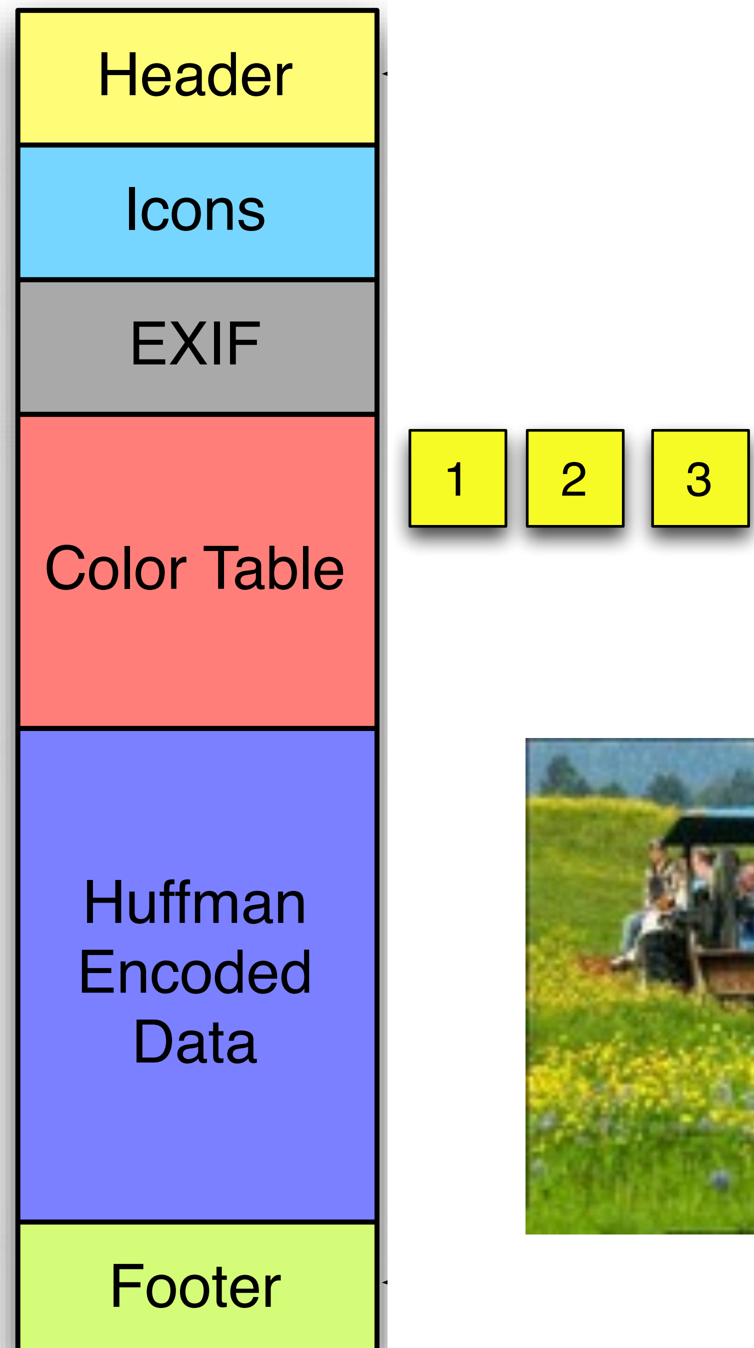


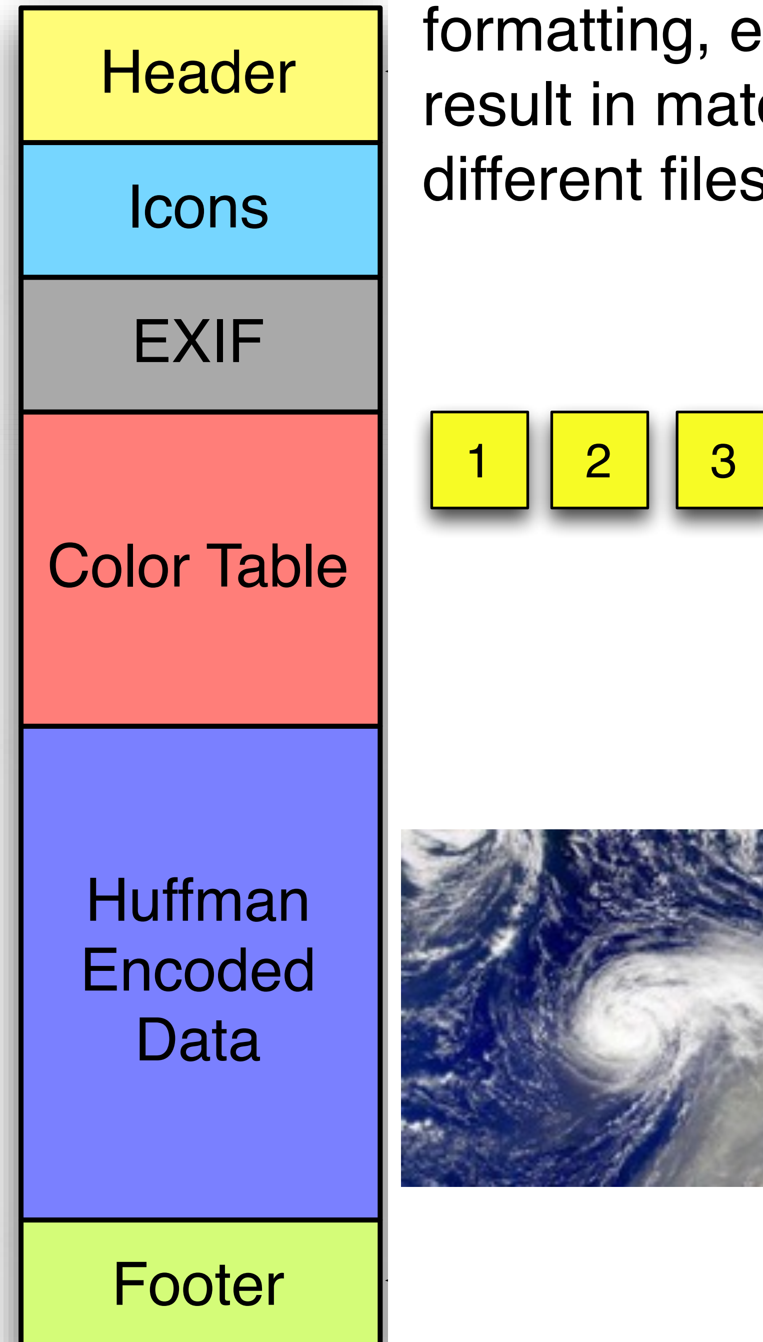
Some blocks are likely to be distinct for each file.



Different files will have different Huffman encoded areas.

Other blocks might occur in more than one file.

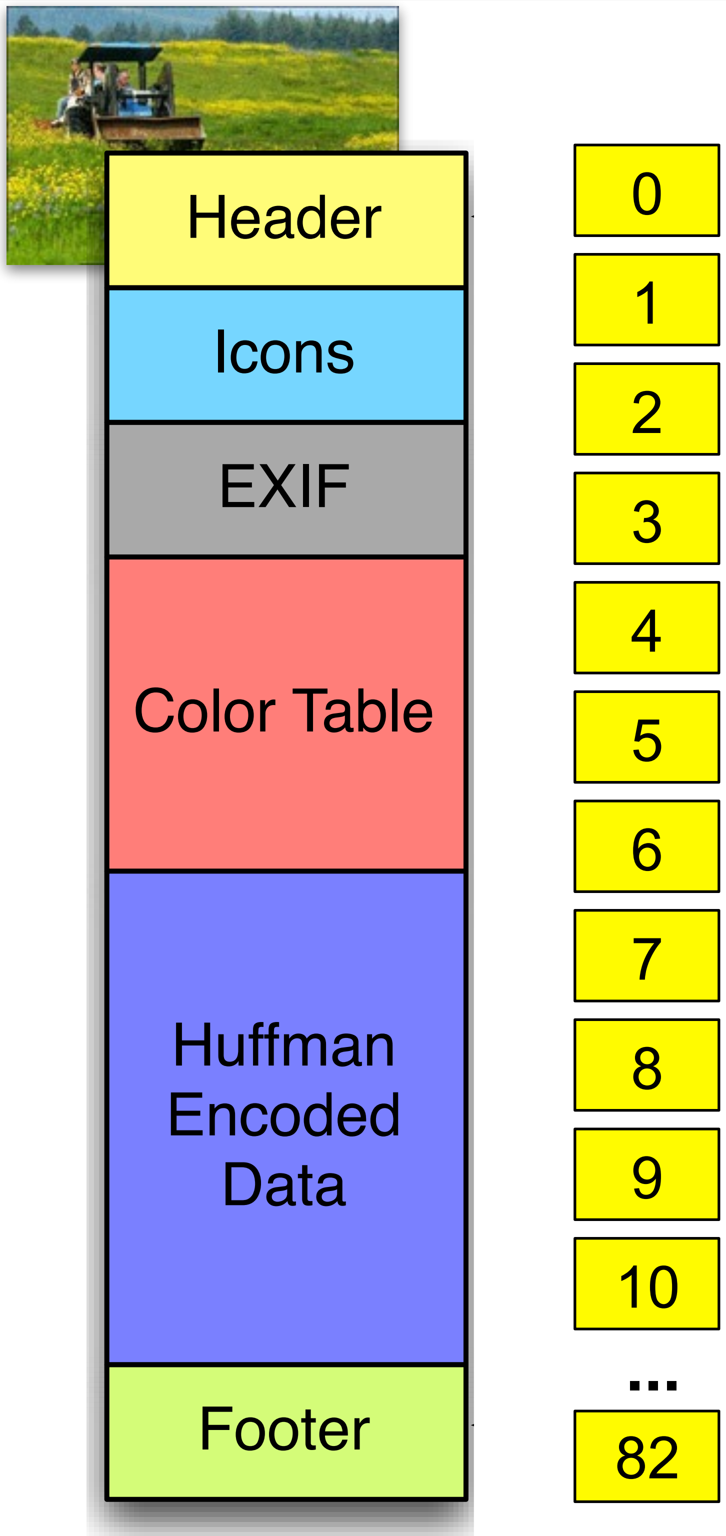




Common metadata, formatting, etc. might result in matches between different files

EXIF and color table are generated by the camera.

This 41K JPEG has 82 x 512B blocks.



Block #	MD5(Block(N))
0	dc0c20abad42d487a7 4f308c69d18a5a
1	9e7bc64399ad87ae9c 2b545061959778
2	6e7f3577b100f9ec7f ae18438fd5b047
3	4594899684d0565789 ae9f364885e303
. . .	. . .

We searched for these block hashes in a corpus of 4 million files.

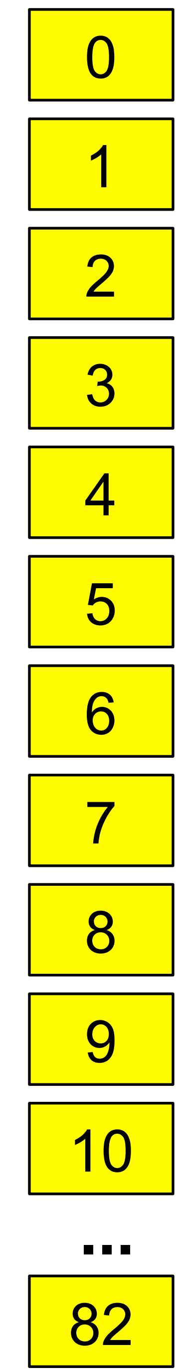
≃ 1 million in GOVDOCS1 collection

= 109,282 JPEGs (including 000107.jpg)

≃ 3 million samples of Windows malware

Results:

- Most of the block hashes in **000107.jpg** do not appear elsewhere in corpus.
- Some of the block hashes appeared in other JPEGs.
- None of the block hashes appeared in files that were not JPEGs



The beginning of the file has many distinct 512B blocks (distinct in our corpus of 100K JPEGs)

hash	location	count
dc0c20abad42d487a74f308c69d18a5a	offset 0-511	1
9e7bc64399ad87ae9c2b545061959778	offset 512-1023	1
6e7f3577b100f9ec7fae18438fd5b047	offset 1024-1535	1
4594899684d0565789ae9f364885e303	offset 1536-2047	1
4d21b27ceec5618f94d7b62ad3861e9a	offset 2048-2559	1
03b6a13453624f649bbf3e9cd83c48ae	offset 2560-3071	1
c996fe19c45bc19961d2301f47cabaa6	offset 3072-3583	1
0691baa904933c9946bbda69c019be5f	offset 3584-4095	1
1bd9960a3560b9420d6331c1f4d95fec	offset 4096-4607	1
52ef8fe0a800c9410bb7a303abe35e64	offset 4608-5119	1
b8d5c7c29da4188a4dcaa09e057d25ca	offset 5120-5631	1
3d7679a976b91c6eb8acd1bfa3414f96	offset 5632-6143	1
8649f180275e0b63253e7ee0e8fa4c1d	offset 6144-6655	1
60ebc8acb8467045e9dcbe207f61a6c2	offset 6656-7167	1
440c1c1318186ac0e42b2977779514a1	offset 7168-7679	1
72686172f8c865231e2b30b2829e3dd9	offset 7680-8191	1
fdff55c618d434416717e5ed45cb407e	offset 8192-8703	1
fcd89d71b5f728ba550a7bc017ea8ff1	offset 8704-9215	1
2d733e47c5500d91cc896f99504e0a38	offset 9216-9727	1
2152fdde0e0a62d2e10b4fecc369e4c6	offset 9728-10239	1
692527fa35782db85924863436d45d7f	offset 10240-10751	1
76dbb9b469273d0e0e467a55728b7883	offset 10752-11263	1

0
1
2
3
4
5
6
7
8
9
10
11
12

We thought that the header would be common, but we were wrong.!

The blocks in the middle of 000107.JPG were seen in *many* JPEGS in the corpus.

hash	location	count
9df886fdfa6934cc7dcf10c04be3464a	offset 14848-15359	1
95399e7ecc7ba1b38243069bdd5c263a	offset 15360-15871	1
ef1ffcdc11162ecdfedd2dde644ec8f2	offset 15872-16383	1
7eb35c161e91b215e2a1d20c32f4477e	offset 16384-16895	1
38f9b6f045db235a14b49c3fe7b1cec3	offset 16896-17407	1
edceba3444b5551179c791ee3ec627a5	offset 17408-17919	1
6bc8ed0ce3d49dc238774a2bdeb7eca7	offset 17920-18431	1
5070e4021866a547aa37e5609e401268	offset 18432-18943	14
13d3222848d5b25e26aefb87dbdf294	offset 18944-19455	9198
0dfcde85c648d20aed68068cc7b57c25	offset 19456-19967	9076
756f0bbe70642700aafb2557bf2c5649	offset 19968-20479	9118
c2c29016d3005f7a1df247168d34e673	offset 20480-20991	9237
42ff3d72b2b25f880be21fac46608cc9	offset 20992-21503	9708
b943cd0ea25e354d4ac22b886045650d	offset 21504-22015	9615
a003ec2c4145b0bc871118842b74f385	offset 22016-22527	9564
1168c351f57aad14de135736c06665ea	offset 22528-23039	7
51a50e6148d1311669218dc40940ce5	offset 23040-23551	83
365b122f53075cb76b39ca1366418ff9	offset 23552-24063	83
9ad9660e7c812e2568aaf063a1be7d05	offset 24064-24575	84
67bd01c2878172e2853f0aef341563dc	offset 24576-25087	84
fc3e47d734d658559d1624c8b1cbf2c1	offset 25088-25599	84
cb9aef5b7f32e2a983e67af38ce8ff87	offset 25600-26111	1

29
30
31
32
36
37
38
39
40
44
45
46
50

Block 37 was found in 9198 other files.
The sector is filled with blank lines 100 characters long...

```
13d3222848d5b25e26aefb87dbdf294    offset 18944-19455      9198

$ dd if=000107.jpg skip=18944 count=512 bs=1|xxd
0000000: 2020 2020 2020 2020 2020 2020 2020 2020
0000010: 2020 2020 2020 2020 2020 2020 0a20 2020              .
0000020: 2020 2020 2020 2020 2020 2020 2020 2020
0000030: 2020 2020 2020 2020 2020 2020 2020 2020
0000040: 2020 2020 2020 2020 2020 2020 2020 2020
0000050: 2020 2020 2020 2020 2020 2020 2020 2020
0000060: 2020 2020 2020 2020 2020 2020 2020 2020
0000070: 2020 2020 2020 2020 2020 2020 2020 2020
0000080: 200a 2020 2020 2020 2020 2020 2020 2020   .
0000090: 2020 2020 2020 2020 2020 2020 2020 2020
00000a0: 2020 2020 2020 2020 2020 2020 2020 2020
00000b0: 2020 2020 2020 2020 2020 2020 2020 2020
00000c0: 2020 2020 2020 2020 2020 2020 2020 2020
00000d0: 2020 2020 2020 2020 2020 2020 2020 2020
00000e0: 2020 2020 2020 0a20 2020 2020 2020 2020          .
00000f0: 2020 2020 2020 2020 2020 2020 2020 2020
```

37

This pattern comes from the "whitespace padding" of the XMP section.

- The whitespace can start on any byte offset, making collisions likely but not common

Block 45 was found in 83 other files. It appears to contain EXIF metadata

```
51a50e6148d1311669218dc40940ce5   offset 23040-23551  83
```

45

```
$ dd if=000107.jpg skip=23040 count=512 bs=1|xxd
0000000: 3936 362d 322e 3100 0000 0000 0000 0000  966-2.1.........
0000010: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000020: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000030: 0000 0000 0000 0000 0058 595a 2000 0000  .........XYZ ...
0000040: 0000 00f3 5100 0100 0000 0116 cc58 595a  ....Q........XYZ
0000050: 2000 0000 0000 0000 0000 0000 0000 0000   ...............
0000060: 0058 595a 2000 0000 0000 006f a200 0038  .XYZ ......o...8
0000070: f500 0003 9058 595a 2000 0000 0000 0062  .....XYZ ......b
0000080: 9900 00b7 8500 0018 da58 595a 2000 0000  .........XYZ ...
0000090: 0000 0024 a000 000f 8400 00b6 cf64 6573  ...$.........des
00000a0: 6300 0000 0000 0000 1649 4543 2068 7474  c........IEC htt
00000b0: 703a 2f2f 7777 772e 6965 632e 6368 0000  p://www.iec.ch..
00000c0: 0000 0000 0000 0000 0016 4945 4320 6874  ..........IEC ht
00000d0: 7470 3a2f 2f77 7777 2e69 6563 2e63 6800  tp://www.iec.ch.
00000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000100: 0000 0000 0000 0000 0000 0000 0064 6573  .............des
0000110: 6300 0000 0000 0000 2e49 4543 2036 3139  c........IEC 619
0000120: 3636 2d32 2e31 2044 6566 6175 6c74 2052  66-2.1 Default R
```

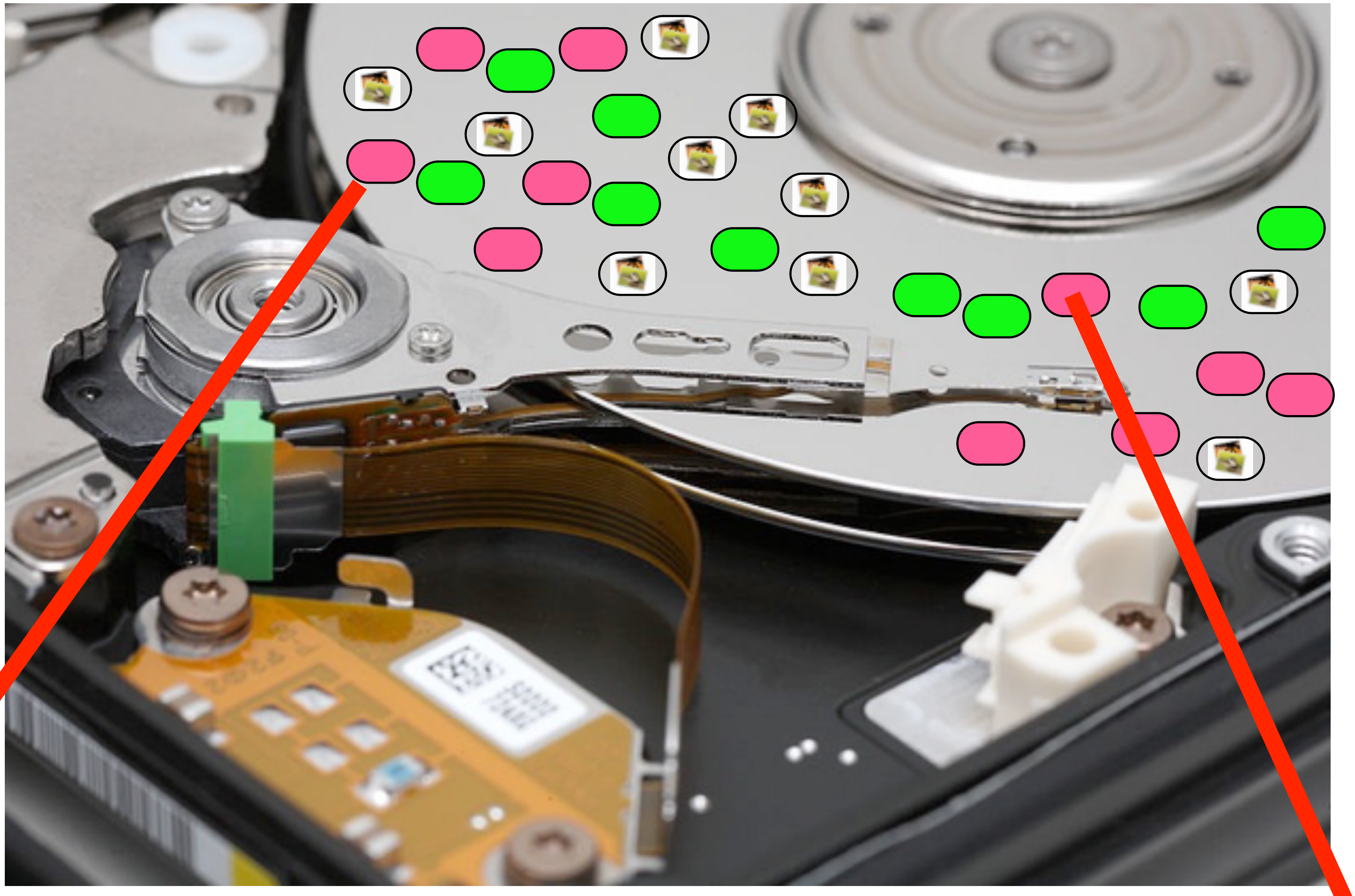
Block 48 was found in 84 other files.
It appears to contain part of a JPEG color table...

48

```
67bd01c2878172e2853f0aef341563dc   offset 24576-25087        84

$ dd if=000107.jpg skip=24576 count=512 bs=1 |xxd
0000000: 7a27 ab27 dc28 0d28 3f28 7128 a228 d429  z'.'.(.(?(q(.(.)
0000010: 0629 3829 6b29 9d29 d02a 022a 352a 682a  .)8)k).).*.*5*h*
0000020: 9b2a cf2b 022b 362b 692b 9d2b d12c 052c  .*.+.+6+i+.+.,.,
0000030: 392c 6e2c a22c d72d 0c2d 412d 762d ab2d  9,n,.,.-.-A-v-.-
0000040: e12e 162e 4c2e 822e b72e ee2f 242f 5a2f  ....L......./$/Z/
0000050: 912f c72f fe30 3530 6c30 a430 db31 1231  ././.050l0.0.1.1
0000060: 4a31 8231 ba31 f232 2a32 6332 9b32 d433  J1.1.1.2*2c2.2.3
0000070: 0d33 4633 7f33 b833 f134 2b34 6534 9e34  .3F3.3.3.4+4e4.4
0000080: d835 1335 4d35 8735 c235 fd36 3736 7236  .5.5M5.5.5.676r6
0000090: ae36 e937 2437 6037 9c37 d738 1438 5038  .6.7$7`7.7.8.8P8
00000a0: 8c38 c839 0539 4239 7f39 bc39 f93a 363a  .8.9.9B9.9.9.:6:
00000b0: 743a b23a ef3b 2d3b 6b3b aa3b e83c 273c  t:.:.;-;k;.;.<'<
00000c0: 653c a43c e33d 223d 613d a13d e03e 203e  e<.<.="=a=.=.> >
00000d0: 603e a03e e03f 213f 613f a23f e240 2340  `>.>.?!?a?.?.@#@
00000e0: 6440 a640 e741 2941 6a41 ac41 ee42 3042  d@.@.A)AjA.A.B0B
00000f0: 7242 b542 f743 3a43 7d43 c044 0344 4744  rB.B.C:C}C.D.DGD
0000100: 8a44 ce45 1245 5545 9a45 de46 2246 6746  .D.E.EUE.E.F"FgF
0000110: ab46 f047 3547 7b47 c048 0548 4b48 9148  .F.G5G{G.H.HKH.H
0000120: d749 1d49 6349 a949 f04a 374a 7d4a c44b  .I.IcI.I.J7J}J.K
0000130: 0c4b 534b 9a4b e24c 2a4c 724c ba4d 024d  .KSK.K.L*LrL.M.M
```

To make sector hashing useful, we can only use the hashes that are "distinct."



`dc0c20abad42d487a74f308c69d18a5a`

Probative hash: seen only in 1 file

`6e7f3577b100f9ec7fae18438fd5b047`

Non-probative: seen in many files

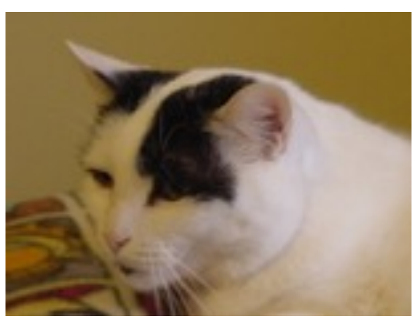
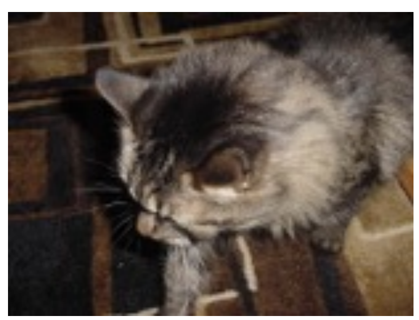
Question: how many files do we need to consider?

Experimental Setup

We performed a realistic test

Target files:

- “Monterey Kitty”
  - *—82 JPEGs, 2 QT movies, 4 MPEG4 files (201MB in total)*
- GOVDOCS1
  - *—≈1M files downloaded from US Government web sites*

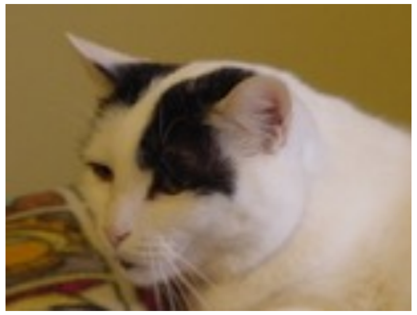



Search Media:

- M57-Patents — Scenario of a small business developed by NPS in 2009.
- jo-2009-11-20-oldComputer — disk image of person who had “kitty” materials.
  - *—13 GB disk image*

Experiment

1.Create hashdb database with Monterey Kitty & GOVDOCS1



+

1M files from
USG file servers

2.Use database to scan a "scenario" drive:

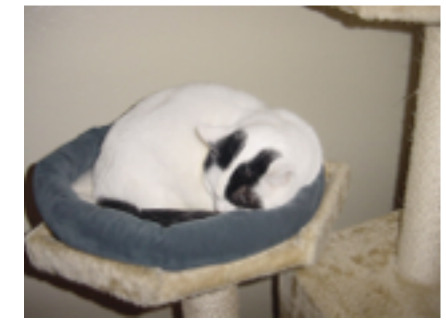
3.Hypothesis:

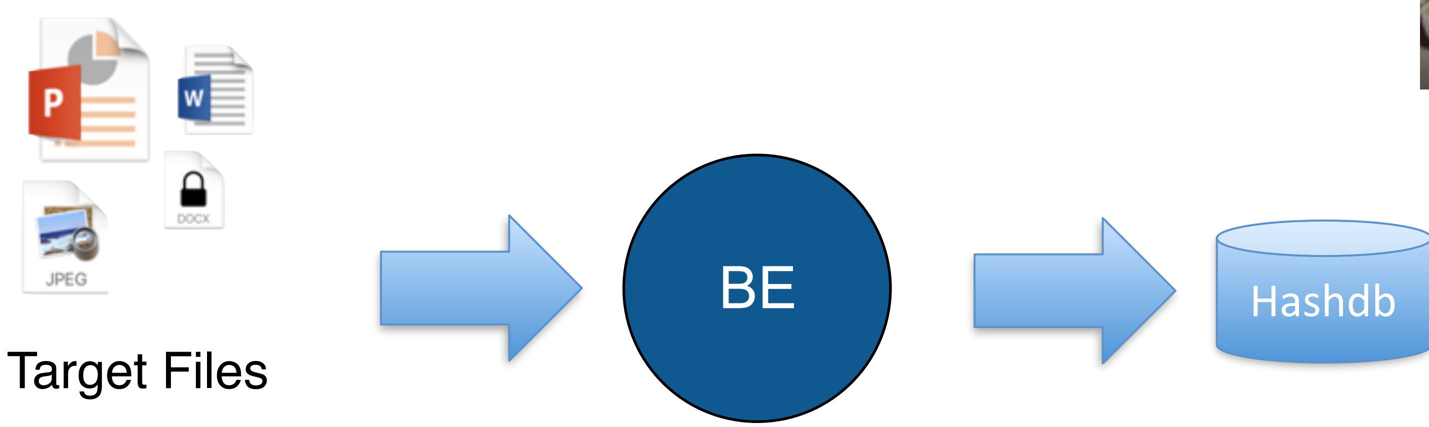
- If we found "distinct" blocks from a file, that file was on the test drive.
- We know the ground truth!

Step 1 — Database Building

Create hashdb database using bulk_extractor

- Monterey Kitty database: 50,206 hashes from 88 different files



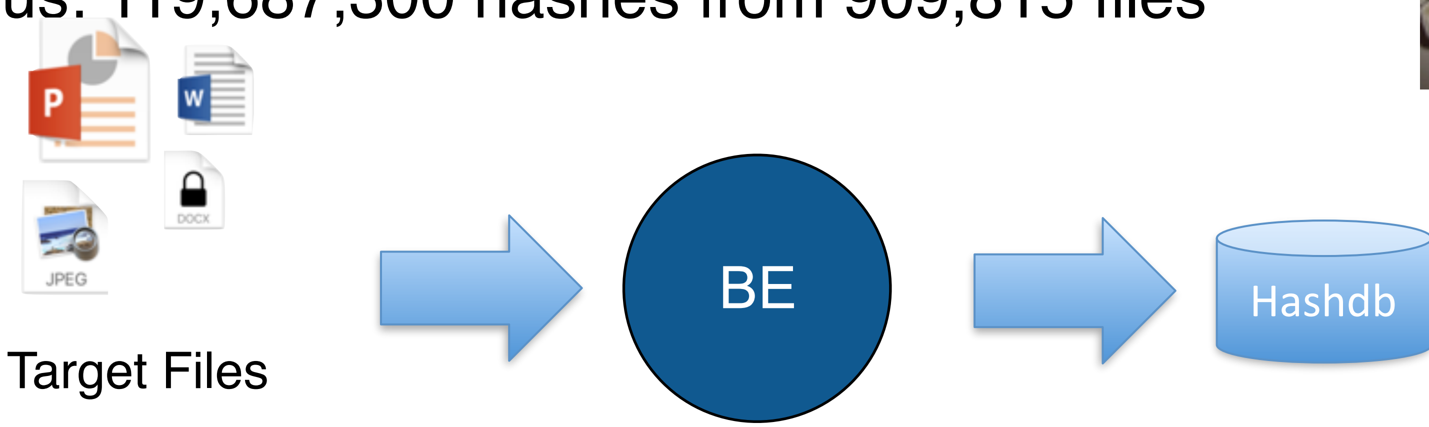


# times in DB	# of hashes
F Singleton	50,206
2 x	0
3 x	0
…	

Step 1 — Database Building

Create hashdb database using bulk_extractor

- Monterey Kitty database: 50,206 hashes from 88 different files
- GOVDOCS corpus: 119,687,300 hashes from 909,815 files



# times in DB	# of hashes
F Singleton	117,213,026
F F 2 x	514,238
F F F 3 x	60,317
…	
F F F 11,434	1 (“null”)

Step 2 — Media Scanning:

Input files: 16GB disk image

- 394 pages (6.3GB) x 32,768 overlapping 4KiB blocks per page.

Scan time: 116 seconds (64-core reference system)

- 111 K lookups/sec

Output — 33,847 matches found:

```
# Feature-Recorder: identified_blocks
# Filename: nps-2009-m57-patents-redacted/jo-2009-11-16.E01

86435328 736d99610d0097be78651ecdae4714bb {"count":39,"flags":"H"}

1231920640 90ccbdf24a74c8c05b94032b4ce1825d {“count":1,"flags":"H"}

1231924736 9403e1cac89e860b93570ac452d232a5 {“count":1}
```

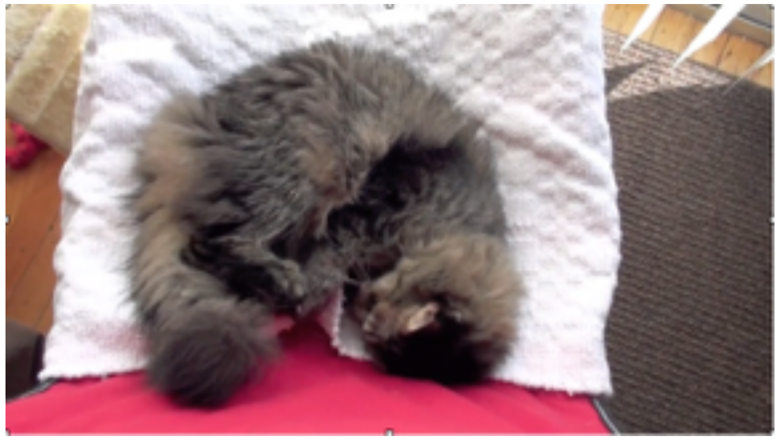
Step 3 — Analysis
What we found.

M57-Patents drives:

Can only be TiggerTheCat.m4v

- Found nearly all Kitty files
  - —*Found multiple copies*

  - —*In some cases, found all of a file <u>except</u> the first sector (that's good!)*



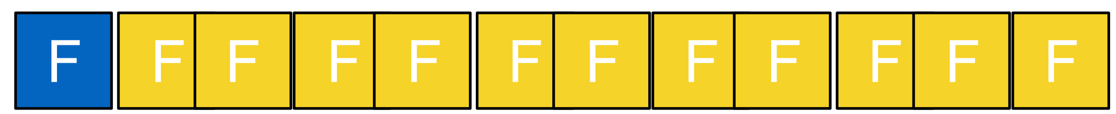
TiggerTheCat.m4v

We also found distinct blocks from GOVDOCS files on the M57 drive

M57-Patents drives:

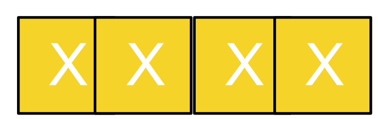
- Found nearly all Kitty files
  - *Found multiple copies*

  - *In some cases, found all of a file except the first sector (that's good!)*



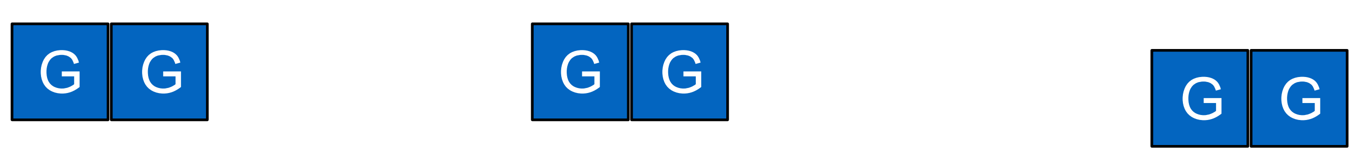
Distinct GOVDOCS files:

- *Found several complete files!* ***These files really were present! (fonts)***
- *Found several runs of distinct blocks from files* ***that were never present!***



- *Found many runs of common blocks.*

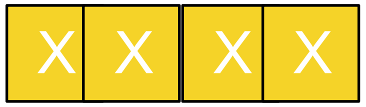
- *Frequently, we find common runs scattered:*



These are non-probative blocks

These blocks match files *that we know are not present.*

We thought they were distinct...
...because we had not looked at enough files!



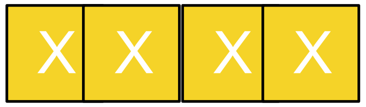
These are non-probative blocks

These blocks match files *that we know are not present.*

We thought they were distinct...
...because we had not looked at enough files!

These blocks were similar to the common blocks we had seen in 0000107.jpg:

- Incrementing binary numbers
- Whitespace
- Strange binary structures



We developed four tests for non-probative blocks.

1. The Ramp Test

- Detect and mark blocks with incrementing 4-byte binary numbers:

```
8102 0000 8202 0000 8302 0000 8402 0000
8502 0000 8602 0000 8702 0000 8802 0000
8902 0000 8a02 0000 8b02 0000 8c02 0000
8d02 0000 8e02 0000 8f02 0000 9002 0000
```

- These typically come from Microsoft Office Sector Allocation Tables.
  - *—They have a strong chance of appearing distinct…*
  - *—but they are algorithmically generated*

We developed four tests for non-probative blocks.

1. The Ramp Test

2. The White Space Test

- Any sector that is 3/4 white space is non-probative.
- Screens out whitespace in JPEGs and other files

```
0000000: 2020 2020 2020 2020 2020 2020 2020 2020
0000010: 2020 2020 2020 2020 2020 2020 0a20 2020                 .
0000020: 2020 2020 2020 2020 2020 2020 2020 2020
0000030: 2020 2020 2020 2020 2020 2020 2020 2020
0000040: 2020 2020 2020 2020 2020 2020 2020 2020
0000050: 2020 2020 2020 2020 2020 2020 2020 2020
0000060: 2020 2020 2020 2020 2020 2020 2020 2020
0000070: 2020 2020 2020 2020 2020 2020 2020 2020
0000080: 200a 2020 2020 2020 2020 2020 2020 2020    .
0000090: 2020 2020 2020 2020 2020 2020 2020 2020
```

We developed four tests for non-probative blocks.

1. The Ramp Test

2. The White Space Test

3. The 4-byte Histogram Test

- Suppresses sector if any 4-byte n-gram is present more than 256 times
- Usually catches white space test as well (but not always)

```
0000 6400 0000 01ff ffff 9c00 0000 0100
0000 6400 0000 01ff ffff 9c00 0000 0200
0000 0000 0000 0100 0000 6400 0000 01ff
ffff 9c00 0000 0100 0000 6400 0000 01ff
ffff 9c00 0000 0100 0000 6400 0000 01ff
ffff 9c00 0000 0100 0000 6400 0000 01ff
ffff 9c00 0000 0100 0000 6400 0000 01ff
ffff 9c00 0000 0100 0000 6400 0000 01ff
```

We developed four tests for non-probative blocks.

1. **The Ramp Test**
2. **The White Space Test**
3. **The 4-byte Histogram Test**
4. ~~**The Entropy Test**~~
    - ~~Mark as non-probative any block with entropy lower than a threshold~~
    - ~~Possibly use instead of “ad hoc” tests~~
    - Didn’t work as well

Evaluating the rules

Effectiveness at removing "distinct" non-probative blocks:

• Drive matches against database:	33,847
"Impossible" matches (source file not present):	677
# of blocks removed by ad hoc rules:	600
Effectiveness:	89%

Unfortunate removal of "non-probative" blocks in target data

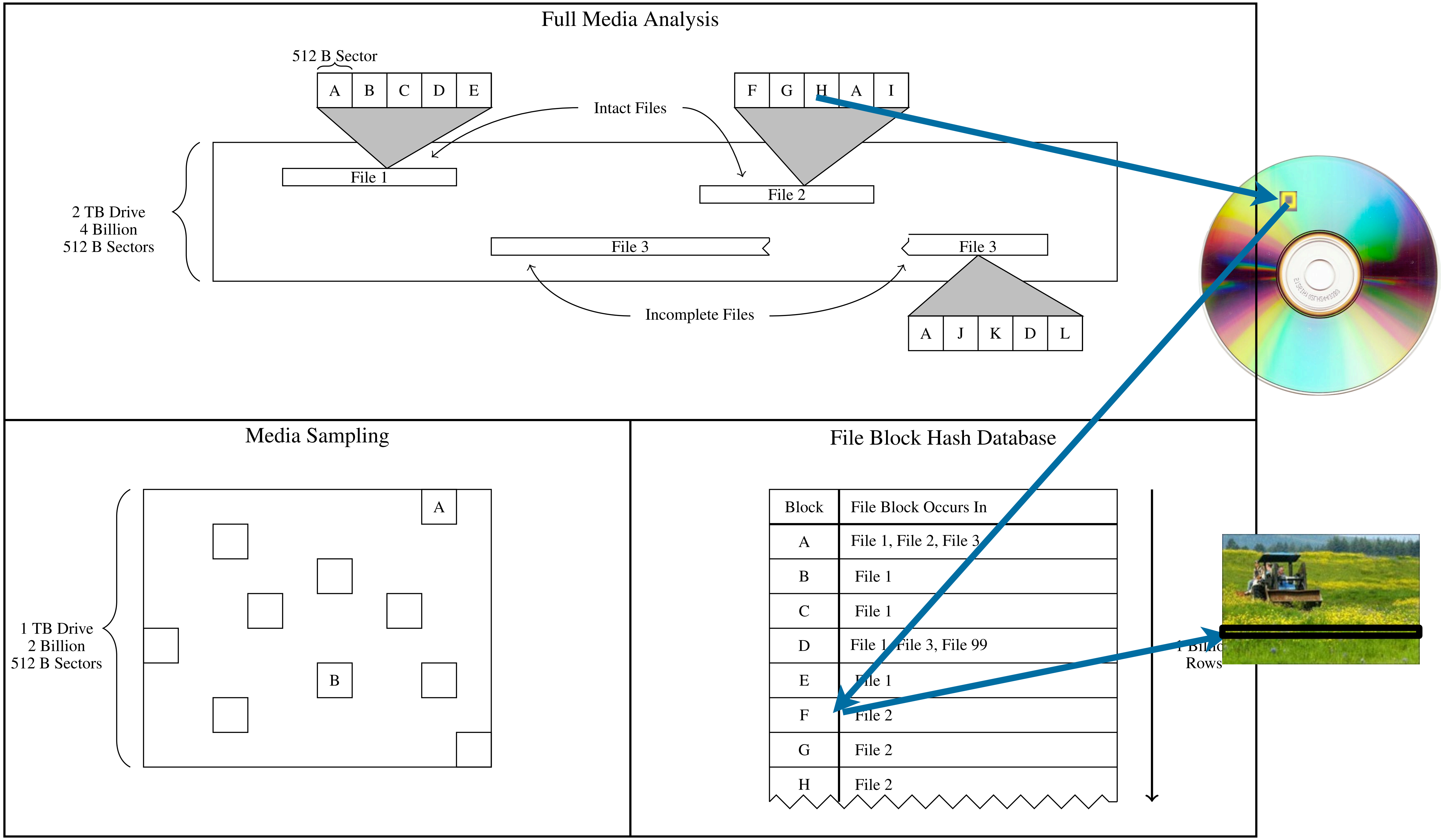
- # of distinct non-probative blocks n our target files: 126

Note: this work done with 4KiB blocks.

- Typical file has 15-500 blocks

Use Cases

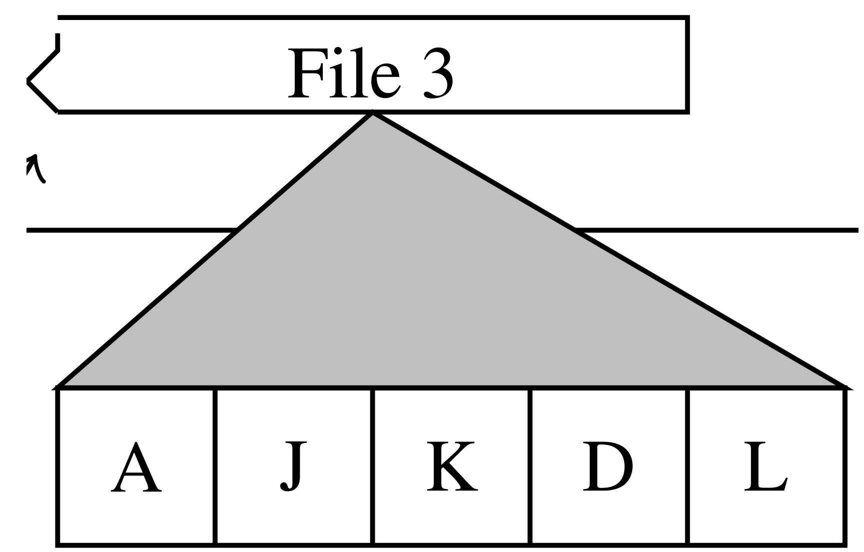
File systems align large files on sector boundaries. We hash file blocks and identify sectors that match.



Two uses cases — One key problem.

Case #1 — Random sampling

- Read & hash randomly chosen sectors.
- Lookup hash values in a database of block hashes.
- Distinct hash implies presence of files.



Case #2 — Full media sampling

- Read & hash every disk sector
- Lookup hash values in a database of block hashes.
- Distinct hash imply presence of files

Key problem: is the hash really "distinct?"

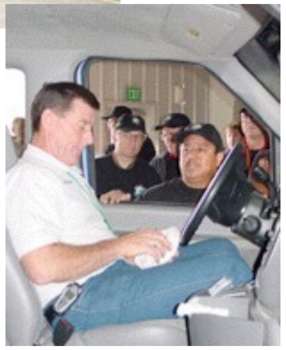
- Or is our corpus too small?
- Only way to tell — rules for identifying non-probative data.

Testing at scale shows this technique works.

Use Case #1: Rapidly search for known contraband:

- 1TB subject hard drive.
- 10 min x 60 min/sec x 1000 msec/sec / 3 msec/sample = 200,000 samples
- Searching for a sector from a corpus of 512GB
- 100% recognition of a single sector; 0% false positive rate

Amount of Contraband	p (prob of missing contraband)
5 MB	0.3654
10 MB	0.1335
15 MB	0.0488
20 MB	0.0178
25 MB	0.0065



Use Case #2: Find a single sector of known contraband:

- Time to read data & search database: 208 minutes

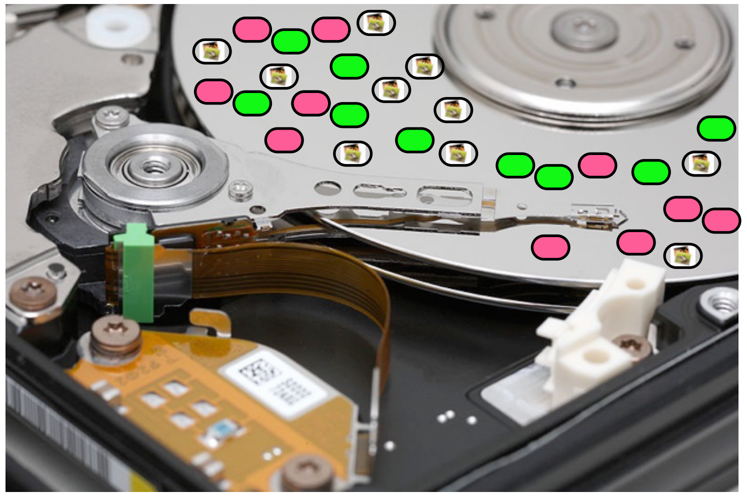
Technique is file type and file system agnostic

—*JPEG; Video; MSWord; Encrypted PDFs...*

—*provided data is not modified when copied or otherwise re-coded*

In summary:
Sector hashing works to identify unique content

We can spot a file from a single sector.



- Search a 1TB drive for 100MB of data in 5min
- Discover traces of a file after it's mostly overwritten.
- Works for disks & RAM

But...

- Requires "distinct" sector hashes — hashes linked with a *single file*.
- Some sectors *look distinct but aren't.*
  - —*You can never see enough content to make a "distinct" determination.*

We developed three rules for discarding "non-probative blocks."

- The rules work 89% of the time.
- These rules are heuristics for identifying binary data structures.