# Digital Forensics Innovation: Searching A Terabyte of Data in 10 minutes

Simson L. Garfinkel
Associate Professor, Naval Postgraduate School

June 28, 2014

http://simson.net/

https://domex.nps.edu/deep/

# The Digital Evaluation and Exploitation (DEEP) Group: Research in “trusted” systems and exploitation.

## “Evaluation”

- Trusted hardware and software
- Cloud computing

## “Exploitation”

- MEDEX — “Media” — Hard drives, camera cards, GPS devices.
- CELEX — Cell phone
- DOCEX — Documents
- DOMEX — Document & Media Exploitation

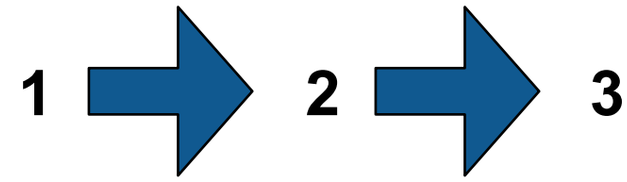
## Partners:

- Law Enforcement (FBI & Local)
- DHS (HSARPA; Video Games & Insider Threat)
- NSF (Courseware development)
- DoD


# Three principles underly our research:

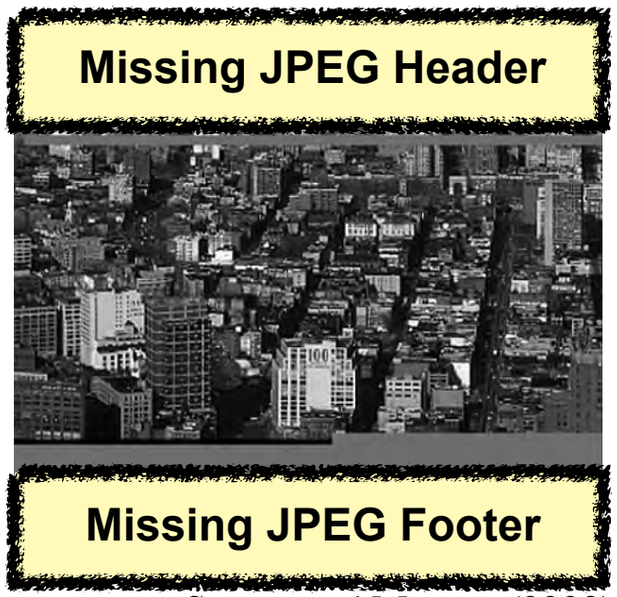
## 1. Automation is essential.

- Today most forensic analysis is done manually.
- We are developing techniques & tools to allow automation.




## 2. Concentrate on the invisible.

- It's *easy* to wipe a computer….
  — *but targets don't erase what they can't see.*
- So we look for:
  —*Deleted and partially overwritten files.*
  —*Fragments of memory in swap & hibernation.*
  —*Tool marks.*




Sencar and Memon (2009)

## 3. Large amounts of data is essential.

- Most research is based on search & recognition
  —*10x the data produces 10x the false-positives*
- We develop algorithms that work *better* with more data.

# Digital information is pervasive in today's society.

Many sources of digital information:

- Desktops, laptops, servers
- Mobile devices: phones & tablets
- Cars
- Internet Services (Cloud)





"Digital forensics" — the recovery, analysis & use of this information

- Law enforcement — Document a conspiracy (stock fraud; murder-for-hire; Silk Road)
- DOD — Identify members of a terrorist organization.
- Cyber investigations — Find and understand malware
- Ordinary people — Recover deleted files.

# We need digital forensics because digital devices are exceedingly complex.

Typical computer might have:

- Millions of files; dozens of applications
- Data from many different people
- Information in many different formats


Digital forensic tools allow for:

- Viewing hidden information / recovery of deleted files
- Determine *when* something happened in the computer's past
- Establish intent:
  - —*Recovered search terms*
  - —*User-generate content*
- Recover specific information critical to a case:
  - —Identity information (contacts, etc.)
  - —Contraband information *(stolen documents, criminal content)*

These tools can also be used for:

- Privacy auditing; software testing

# The digital forensics process makes *digital evidence* available for [legal] decisions

**Preparation: policy, training & tools**

**Collect & preserve evidence (devices)**

**Extract data**

**Analysis**

**Reporting & Testimony**

# There is also a digital forensics ecology

Students

Examiners

Reports


Commercial Tools

Schools & Training Programs

Seized devices

Open source Tools

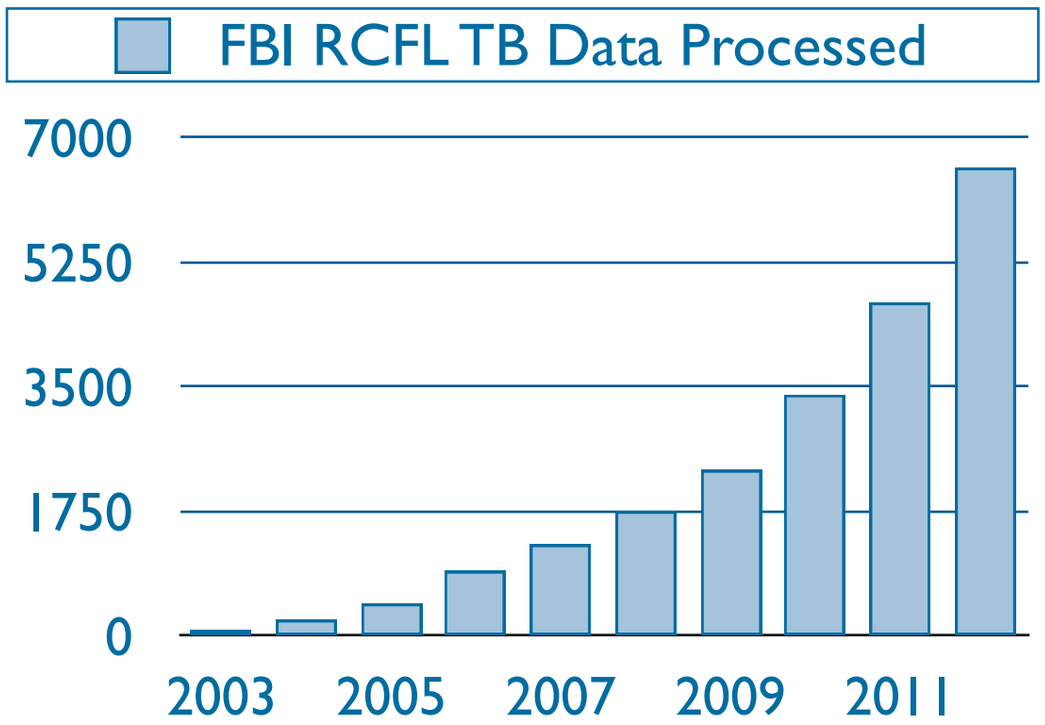

Research

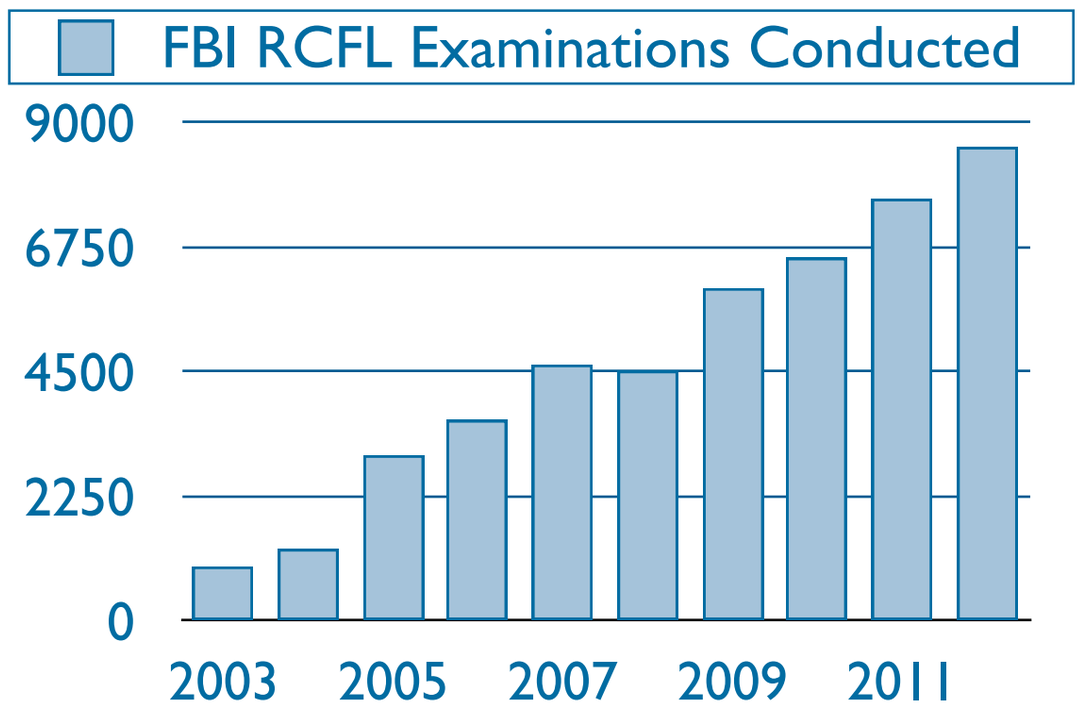
Ecology driver: computer crime & crimes with computers.

Ecology challenges: scale & diversity

# Data scale is a never-ending problem

Each year storage capacity & # investigations increases




Moore's works against digital forensics!

- Each year targets get larger.
- Cases are becoming more complex — multiple devices, web-based services.

We need *triage techniques* to identify the important data.

# Diversity: the second fundamental challenge

Analysts must be able to analyze any data found on any computer.





# Diversity produces fragmentation.
# Different data types, different applications, different tools.

**Mobile**

**iOS** **Android**

**“Feature”**
**“Burner”**

**Call records & contacts**
**User-generated data**
**Apps & Malware**

**Desktop/Laptop**

**Windows** **MacOS**

**User-generated data**
**Malware**

**Server & Private Cloud**

**Linux & Windows**

**Public Cloud**


**Business records**
**Social media**
**User-generated data**
**Malware**

**RAM analysis** **RAM analysis** **RAM analysis**

**Network Forensics: Data flow** **Network Forensics: Data flow**

# There are two approaches for addressing diversity.

## Device-specific techniques

- Solutions targeted at iOS, Android, Windows
- Pros:
  - —*Can recover highly specific data*
  - —*Highly targeted*
- Cons:
  - —*Requires substantial reverse engineering.*
  - —*Often works with only a specific version. Expensive to maintain*
- Examples: Celebrite (extraction), TaintDroid (analysis)

## Generic techniques

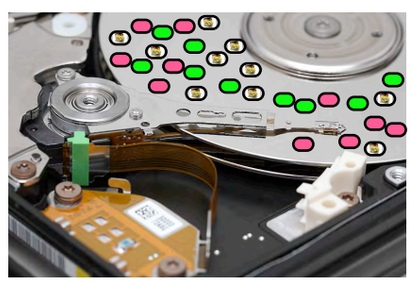
- Pros — Broadly applicable.
- Cons — Less targeted; may require substantial post-processing.
- Example: file hashing  (content-based search & ignore lists.)

# This talk presents two bulk data analysis techniques for triage and full-content analysis.

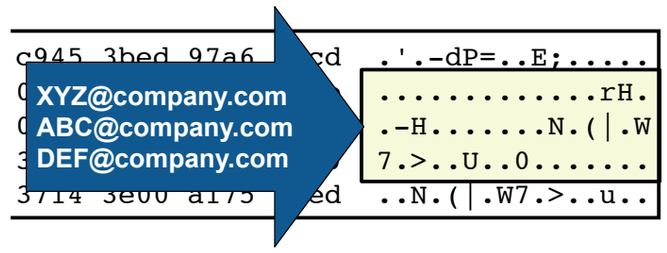
Introducing digital forensics and bulk data analysis.

✔

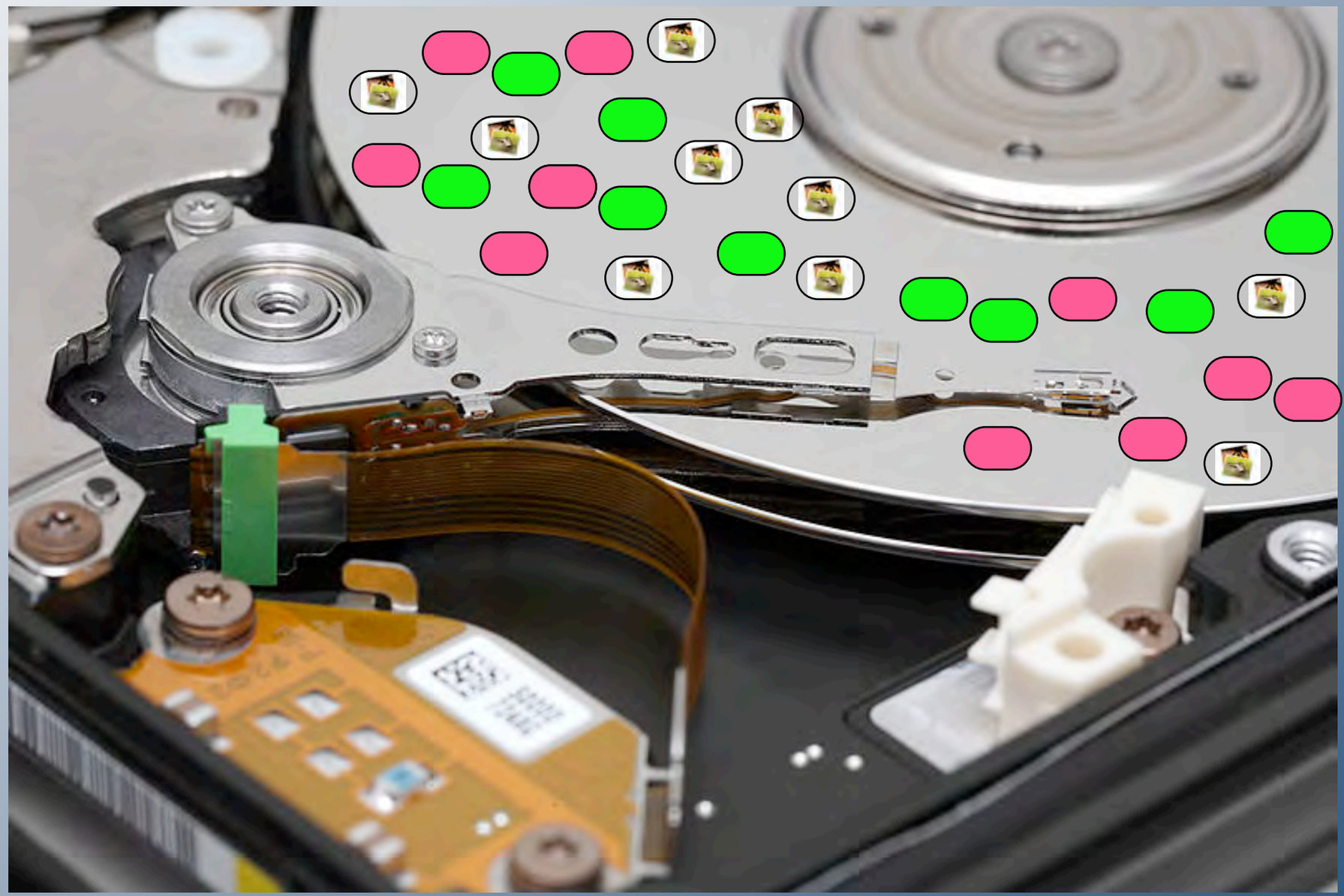
Sector hashing and random sampling for high speed forensics



Optimistic decoding finds “invisible” information




A research agenda ?




# High speed forensic analysis with random sampling and sector hashing

# Triage techniques are a promising approach for data overload

"Triage" prioritizes analysis & helps make go/no-go decisions.

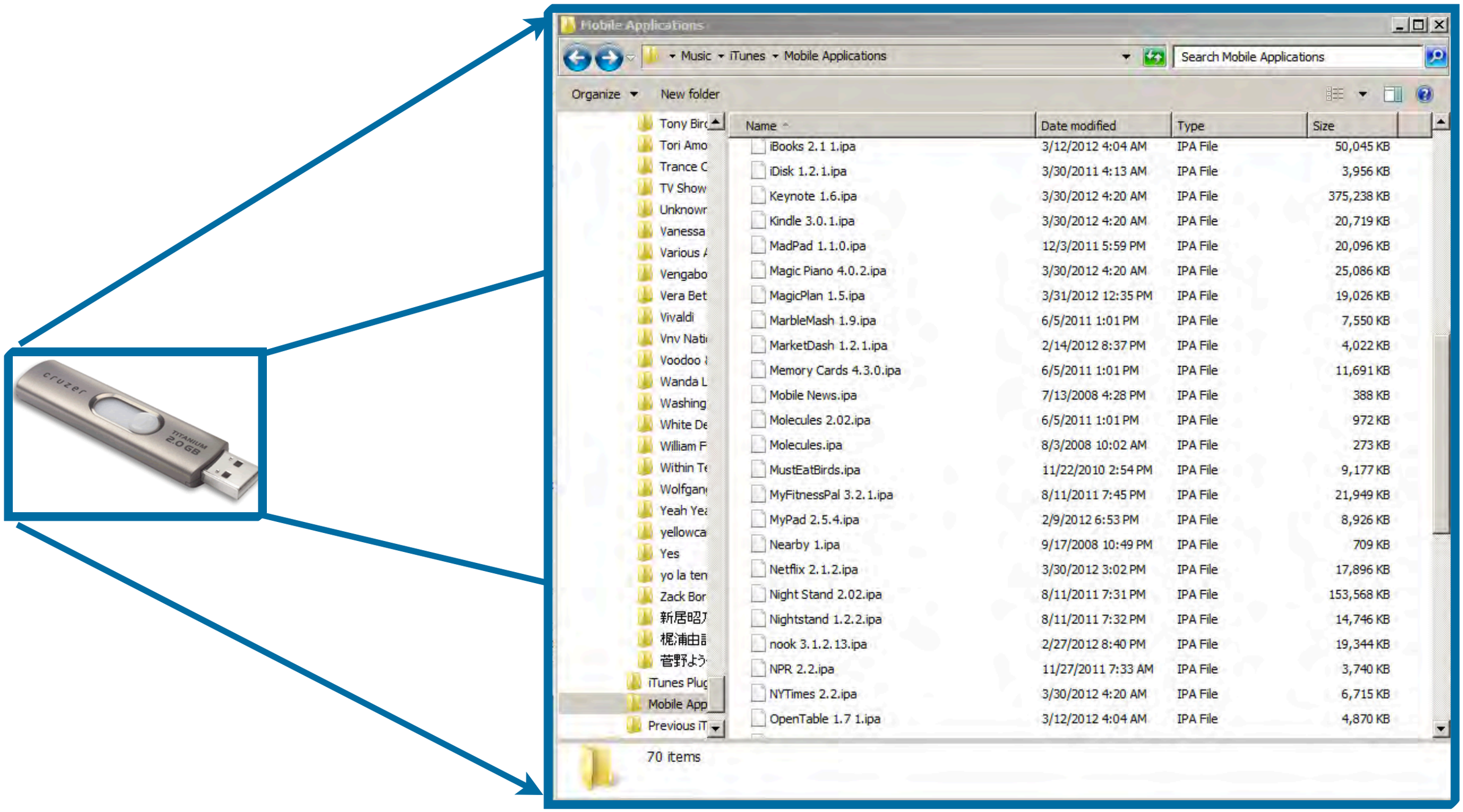
Examples:

- Deciding which devices to search at a border crossing
- Which computers to search in an organization
- Which devices to analyze first


Simplified triage problem:

- What can we learn about a 1TB drive in five minutes?
- Possible approaches:
  - —*Find & extract critical files. (Effective unless there is an active adversary)*
  - —*Randomly sample the drive, looking for relevant data.*

# We think of computers as devices with *files*.



# Storage devices organize data in *blocks* (*sectors*).

"hex dump:"

| hex words | | | | | | | | ASCII representation |
|---|---|---|---|---|---|---|---|---|
| a2b5 | bea7 | 692f | 5847 | a38a | dd53 | 082c | add5 | ....i/XG...S.,.. |
| a097 | 83a1 | ed96 | 26a6 | 3c69 | 3d0f | 750a | 2399 | ......&.<i=.u.#. |
| a2b5 | bea7 | 692f | 5847 | a38a | dd53 | 082c | add5 | ....i/XG...S.,.. |
| 5061 | b64c | 721d | 864b | 90b6 | b55f | bb04 | 735c | Pa.Lr..K..._..s\ |
| 9448 | 6730 | 5453 | df64 | 813e | b603 | 5795 | 2242 | .Hg0TS.d.>..W."B |
| e9c8 | 7454 | 7322 | 7cdc | b60e | 97af | 2f64 | 2728 | ..tTs"\|...../d'( |
| a097 | 83a1 | ed96 | 26a6 | 3c69 | 3d0f | 750a | 2399 | ......&.<i=.u.#. |
| a2b5 | bea7 | 692f | 5847 | a38a | dd53 | 082c | add5 | ....i/XG...S.,.. |
| 5061 | b64c | 721d | 864b | 90b6 | b55f | bb04 | 735c | Pa.Lr..K..._..s\ |
| 9448 | 6730 | 5453 | df64 | 813e | b603 | 5795 | 2242 | .Hg0TS.d.>..W."B |
| e9c8 | 7454 | 7322 | 7cdc | b60e | 97af | 2f64 | 2728 | ..tTs"\|...../d'( |
| 3cfb | 84bd | 2a84 | 2dfe | 50ea | 5935 | c349 | 1513 | <XYZ@COMPANY.COM |
| a9e9 | e92c | a3f8 | 6e46 | 0530 | 8a88 | c7a2 | 5d2b | ...,..nF.0....]+ |
| d89d | 77cc | fe1e | f637 | f3f3 | d0af | 1b47 | c09b | ..w....7.....G.. |
| a097 | 83a1 | ed96 | 26a6 | 3c69 | 3d0f | 750a | 2399 | ......&.<i=.u.#. |
| a2b5 | bea7 | 692f | 5847 | a38a | dd53 | 082c | add5 | ....i/XG...S.,.. |
| 5061 | b64c | 721d | 864b | 90b6 | b55f | bb04 | 735c | Pa.Lr..K..._..s\ |
| 9448 | 6730 | 5453 | df64 | 813e | b603 | 5795 | 2242 | .Hg0TS.d.>..W."B |
| e9c8 | 7454 | 7322 | 7cdc | b60e | 97af | 2f64 | 2728 | ..tTs"\|...../d'( |
| a097 | 83a1 | ed96 | 26a6 | 3c69 | 3d0f | 750a | 2399 | ......&.<i=.u.#. |
| a2b5 | bea7 | 692f | 5847 | a38a | dd53 | 082c | add5 | ....i/XG...S.,.. |
| 5061 | b64c | 721d | 864b | 90b6 | b55f | bb04 | 735c | Pa.Lr..K..._..s\ |
| 9448 | 6730 | 5453 | df64 | 813e | b603 | 5795 | 2242 | .Hg0TS.d.>..W."B |
| e9c8 | 7454 | 7322 | 7cdc | b60e | 97af | 2f64 | 2728 | ..tTs"\|...../d'( |
| a9e9 | e92c | a3f8 | 6e46 | 0530 | 8a88 | c7a2 | 5d2b | ...,..nF.0....]+ |
| d89d | 77cc | fe1e | f637 | f3f3 | d0af | 1b47 | c09b | ..w....7.....G.. |
| a097 | 83a1 | ed96 | 26a6 | 3c69 | 3d0f | 750a | 2399 | ......&.<i=.u.#. |
| a2b5 | bea7 | 692f | 5847 | a38a | dd53 | 082c | add5 | ....i/XG...S.,.. |
| 5061 | b64c | 721d | 864b | 90b6 | b55f | bb04 | 735c | Pa.Lr..K..._..s\ |

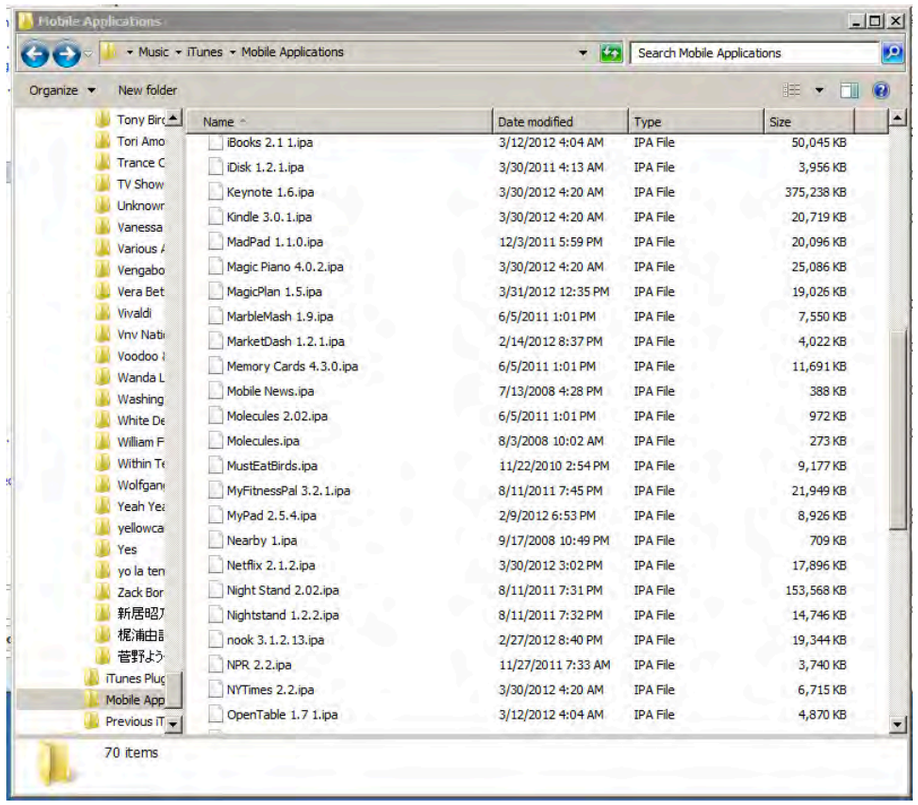

**A 64GB stick has 125 million sectors**

**512 bytes = 1 sector**

# Most digital forensics techniques process files.

Files are familiar:

- People are used to work with files
- Files fit well into the legal process
- Investigators can extract allocated files w/o special tools




"Bulk data" analysis complements file analysis:

- A lot of information is not in files!
  - —*RAM, swap, hibernation files*
- Files can be deleted & partially overwritten
- OS & Apps can be tricked into hiding information
  - —*e.g. "Android Anti-forensics: Modifying CyanogenMod," Karlsson & Glisson 2014*

# This approach combines bulk data analysis with random sampling to make a triage decision.

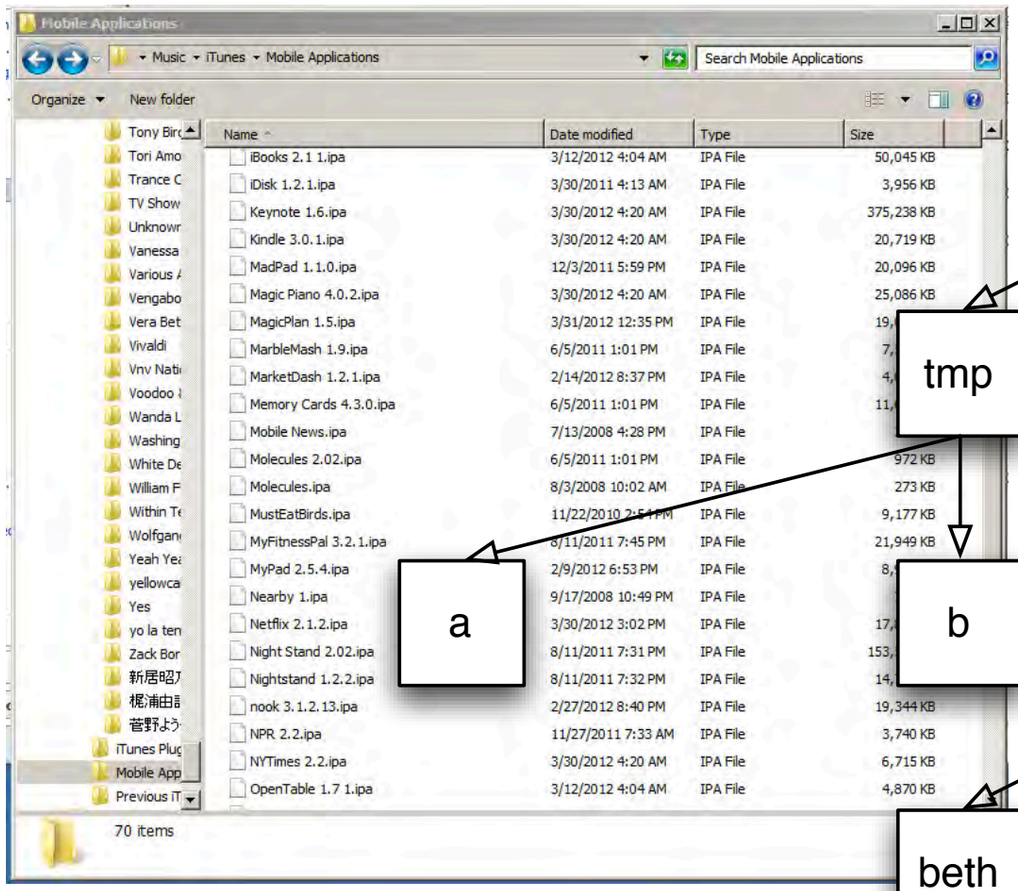
It takes 3.5 hours to read the entire drive:

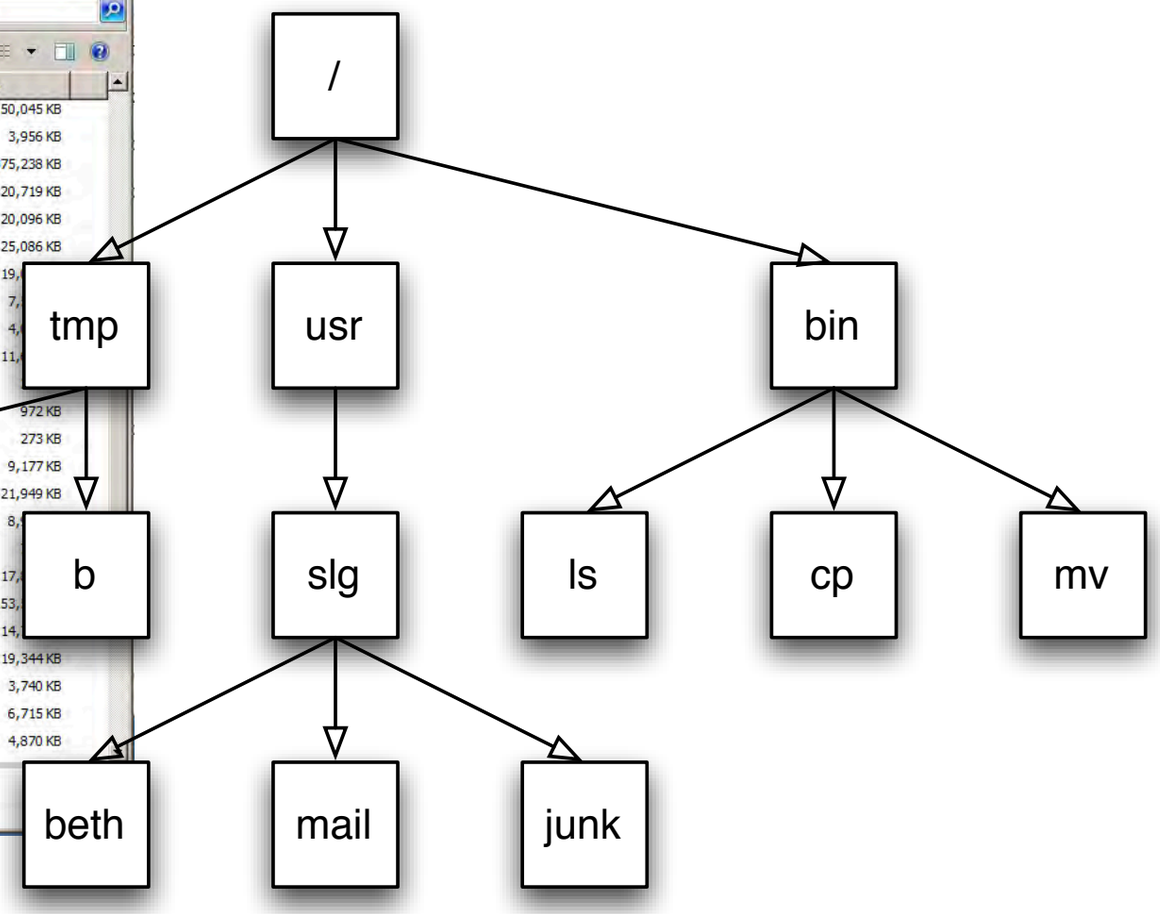
| |  |  |  |  |
|---|---|---|---|---|
| *Minutes* | *208* | *5* | *5* | *5* |
| *Data* | *1 TB* | *36 GB* | *6.5 GB* | *36 GB* |
| *# Seeks* | *1* | *1* | *100,000* | *500,000* |
| *% of data* | *100%* | *3.6%* | *0.65%* | *3.6%* |

In 5 minutes you can read:

- 36 GB in one strip
- HDD: 100,000 randomly chosen 64KiB strips (3 msec/seek)
- SSD: 500,000 randomly chosen 64KiB strips (0 msec/seek)

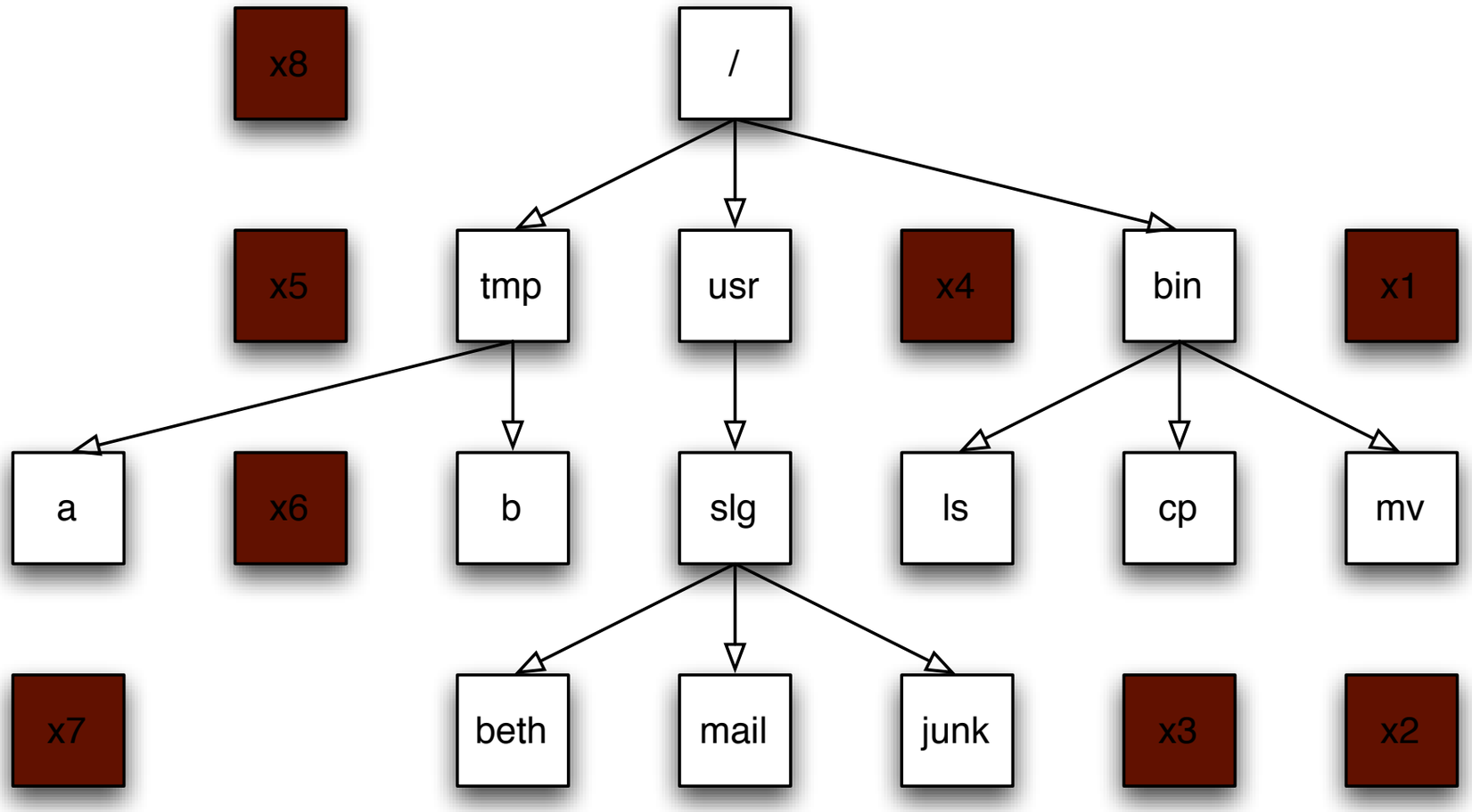
# All data on computers are stored in sectors.
# “Allocated files” are those that can be found from the root.



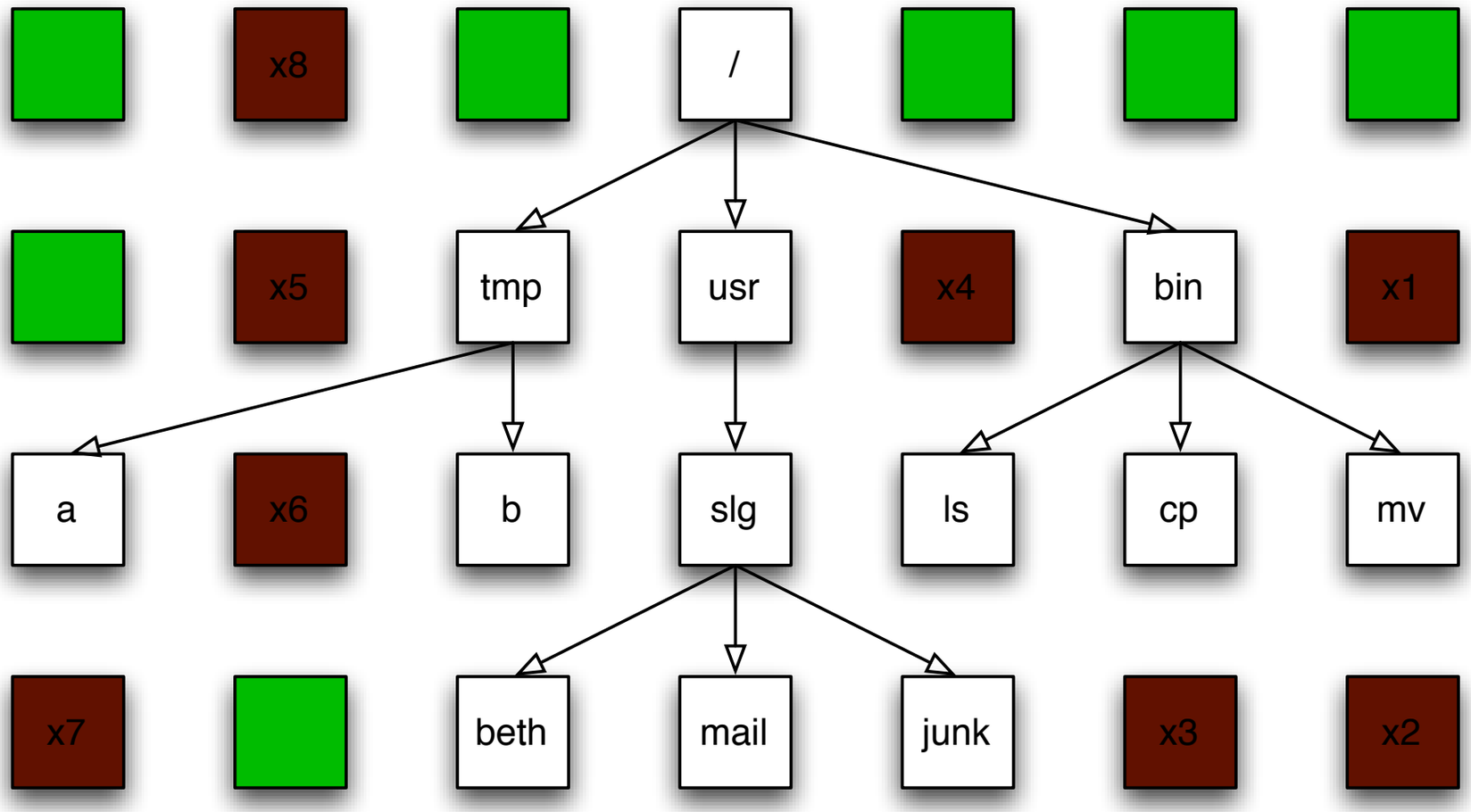

Allocated Files

# “Deleted data” are on the disk, but data can only be recovered with forensic tools.



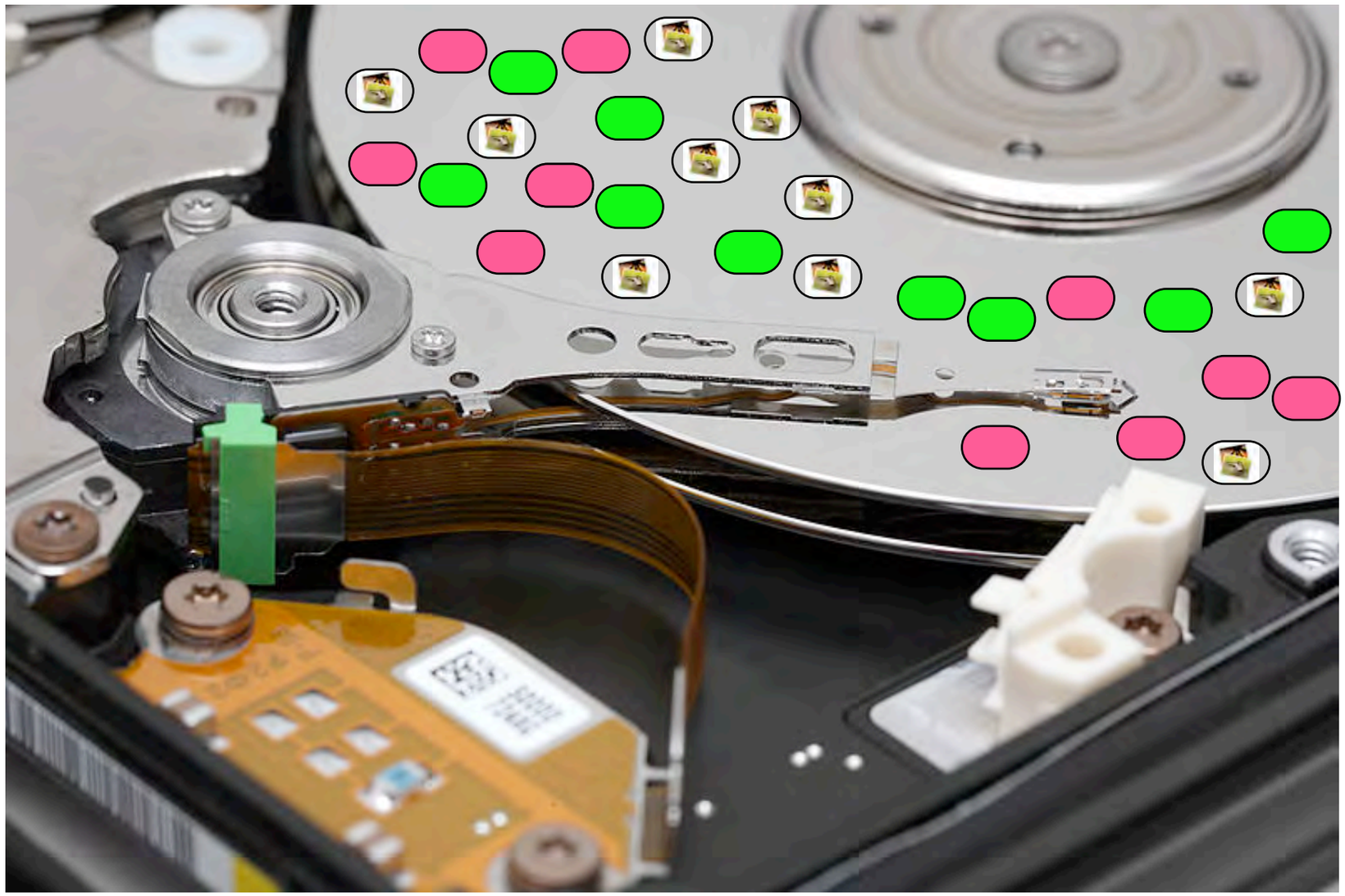

Deleted Data

# Some sectors are blank. They have “No data.”

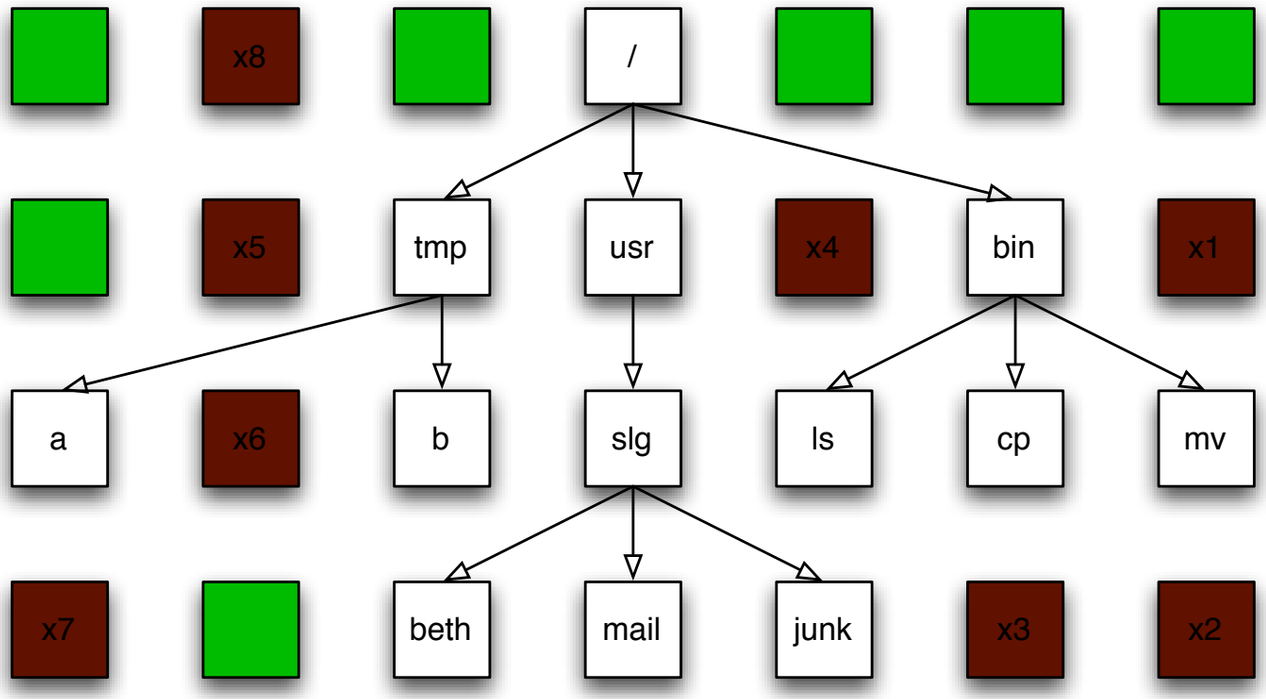


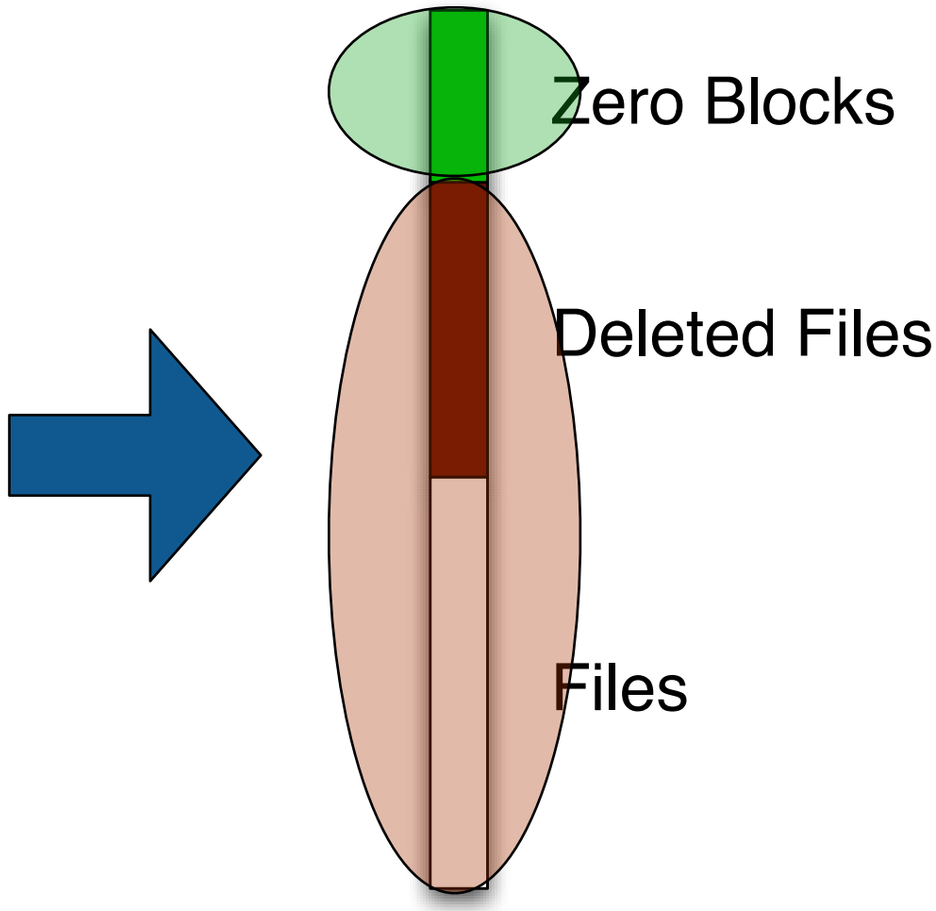

No Data

# Basic idea of random sampling:
# read random sectors and try to make sense of them.



# Sampling can’t distinguish *allocated* from *deleted* data.



# The Challenge for *forensic sampling*: interpreting each sector

"What data does this sector have?"

- Some sectors are easy to discern:

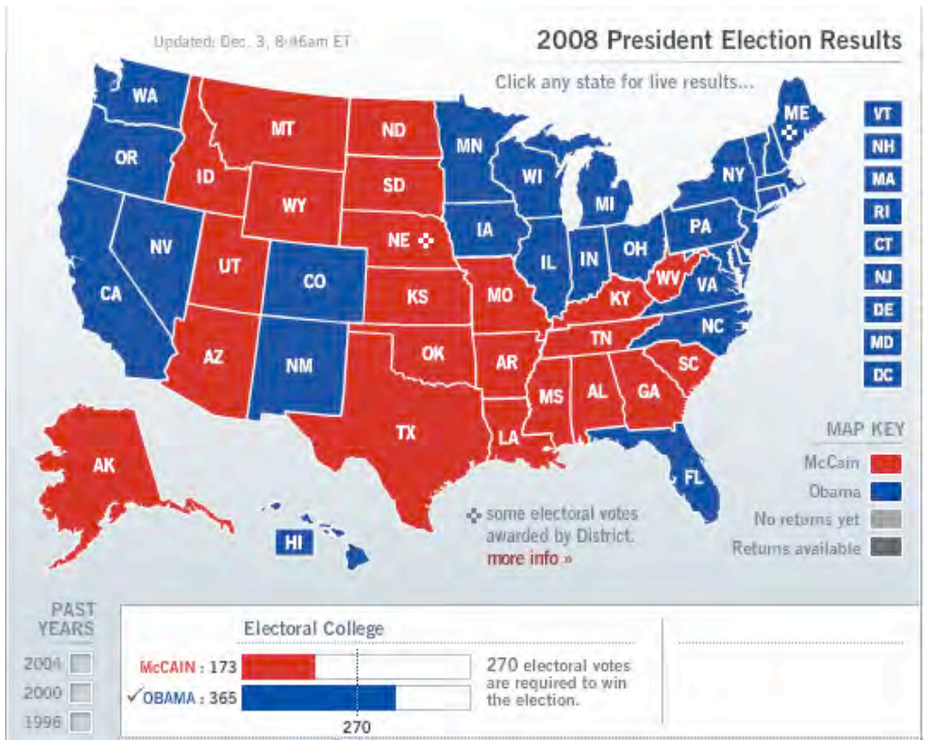
```
0000000: ffd8 ffe0 0010 4a46 4946 0001 0201 0048  ......JFIF.....H
0000010: 0048 0000 ffe1 1d17 4578 6966 0000 4d4d  .H......Exif..MM
0000020: 002a 0000 0008 0007 0112 0003 0000 0001  .*..............
0000030: 0001 0000 011a 0005 0000 0001 0000 0062  ...............b
0000040: 011b 0005 0000 0001 0000 006a 0128 0003  ...........j.(..
0000050: 0000 0001 0002 0000 0131 0002 0000 001b  .........1......
0000060: 0000 0072 0132 0002 0000 0014 0000 008d  ...r.2..........
0000070: 8769 0004 0000 0001 0000 00a4 0000 00d0  .i..............
0000080: 0000 0048 0000 0001 0000 0048 0000 0001  ...H.......H....
0000090: 4164 6f62 6520 5068 6f74 6f73 686f 7020  Adobe Photoshop
00000a0: 4353 2057 696e 646f 7773 0032 3030 353a  CS Windows.2005:
00000b0: 3035 3a30 3920 3136 3a30 313a 3432 0000  05:09 16:01:42..
00000c0: 0000 0003 a001 0003 0000 0001 0001 0000  ................
00000d0: a002 0004 0000 0001 0000 00c8 a003 0004  ................
00000e0: 0000 0001 0000 0084 0000 0000 0000 0006  ................
00000f0: 0103 0003 0000 0001 0006 0000 011a 0005  ................
```

- Some are hard:

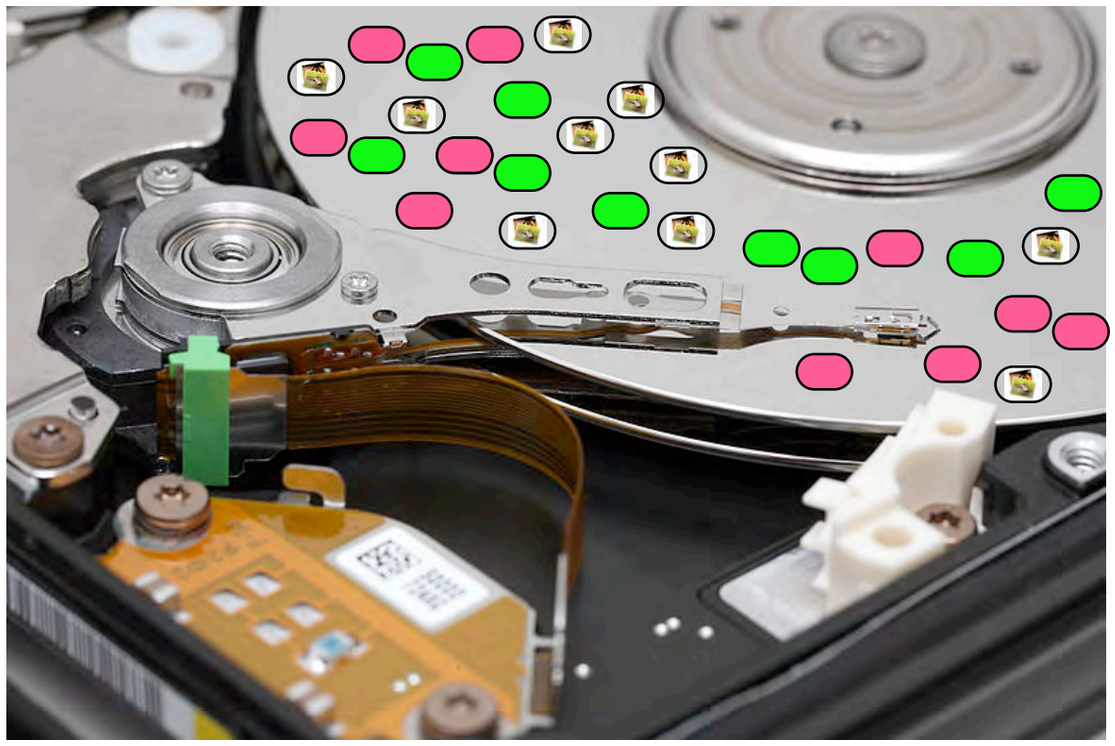
```
000a000: 0011 fa71 57f4 6f5f ddff 00bd 15fb 5dfd  ...qW.o_......].
000a010: a996 0fc9 dff1 ff00 b149 e154 97f4 efd5  .........I.T....
000a020: e3f5 7f47 71df 8ffb d5d7 da9e d87f c12f  ...Gq........../
000a030: f8ff 00d8 b1f4 b1f8 ff00 c57e ab7a ff00  ..........~.z..
```

# This is similar to other kinds of sampling.
# The goal is to predict the population from the sample.

US elections can be predicted by sampling thousands of households:




Hard drive contents can be predicted by sampling thousands of sectors:



The challenge is identifying *likely voters.*

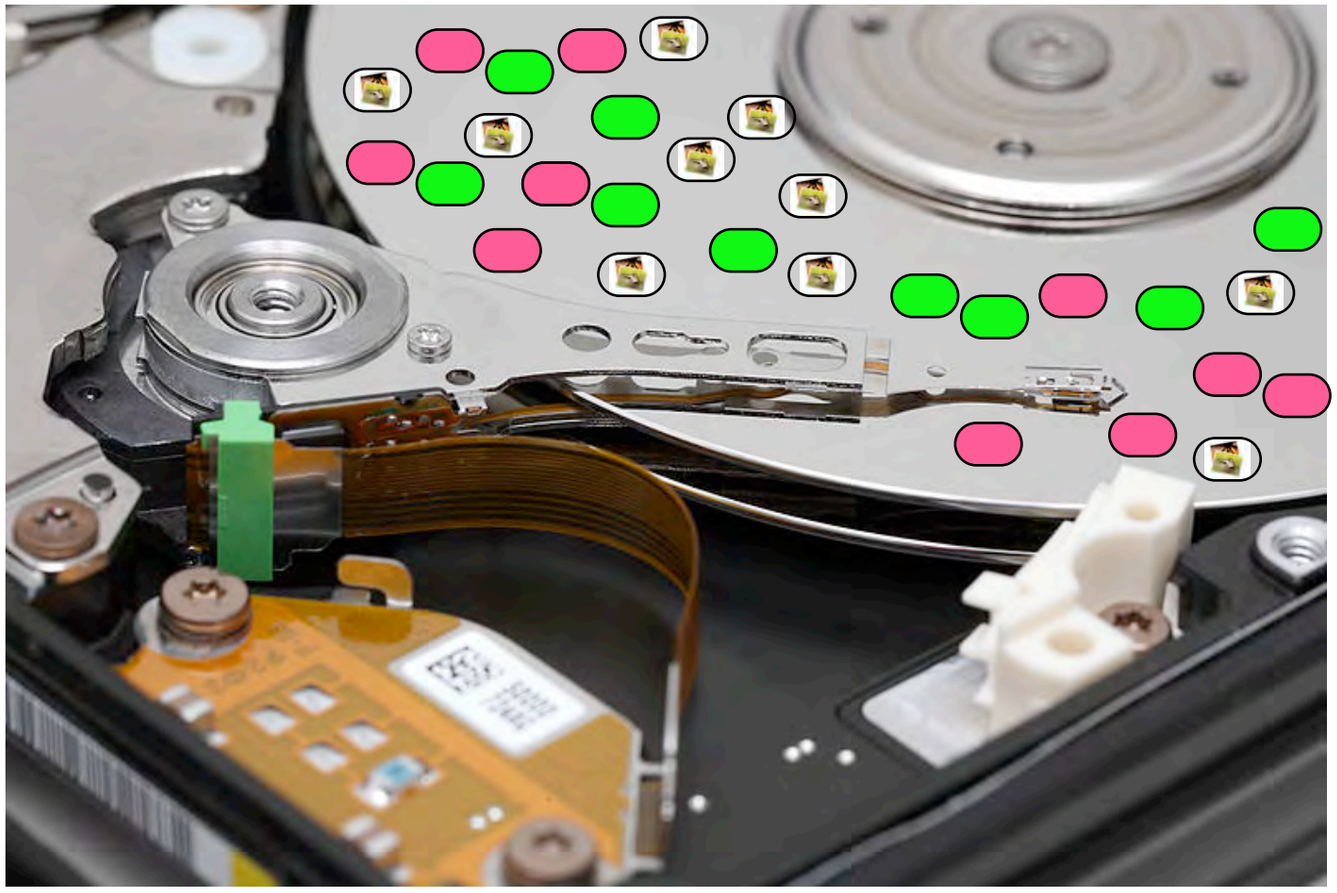
The challenge is *interpreting the content* of the sector that is sampled.

# Sampling can tell us about the content of the data

Sampling can tell us the proportion of...

—*blank sectors; video; HTML files; other data types...*

—*data with distinct signatures...*

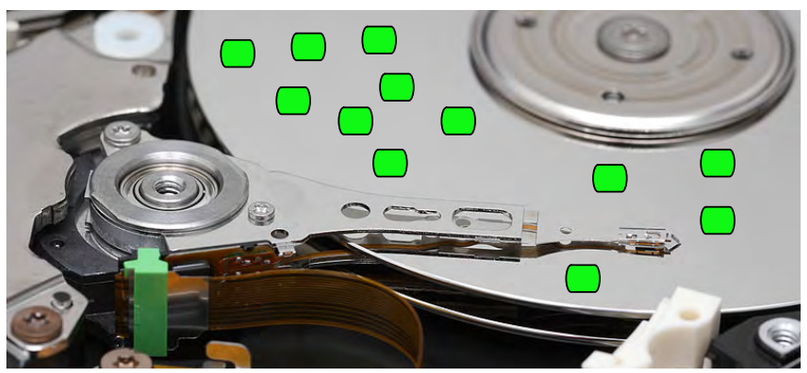


...provided we can identify the data type of each sector.

# Simplify the problem.
# Can we use statistical sampling to verify wiping?

Many organizations discard used computers.

Can we verify if a disk is properly wiped in 5 minutes?

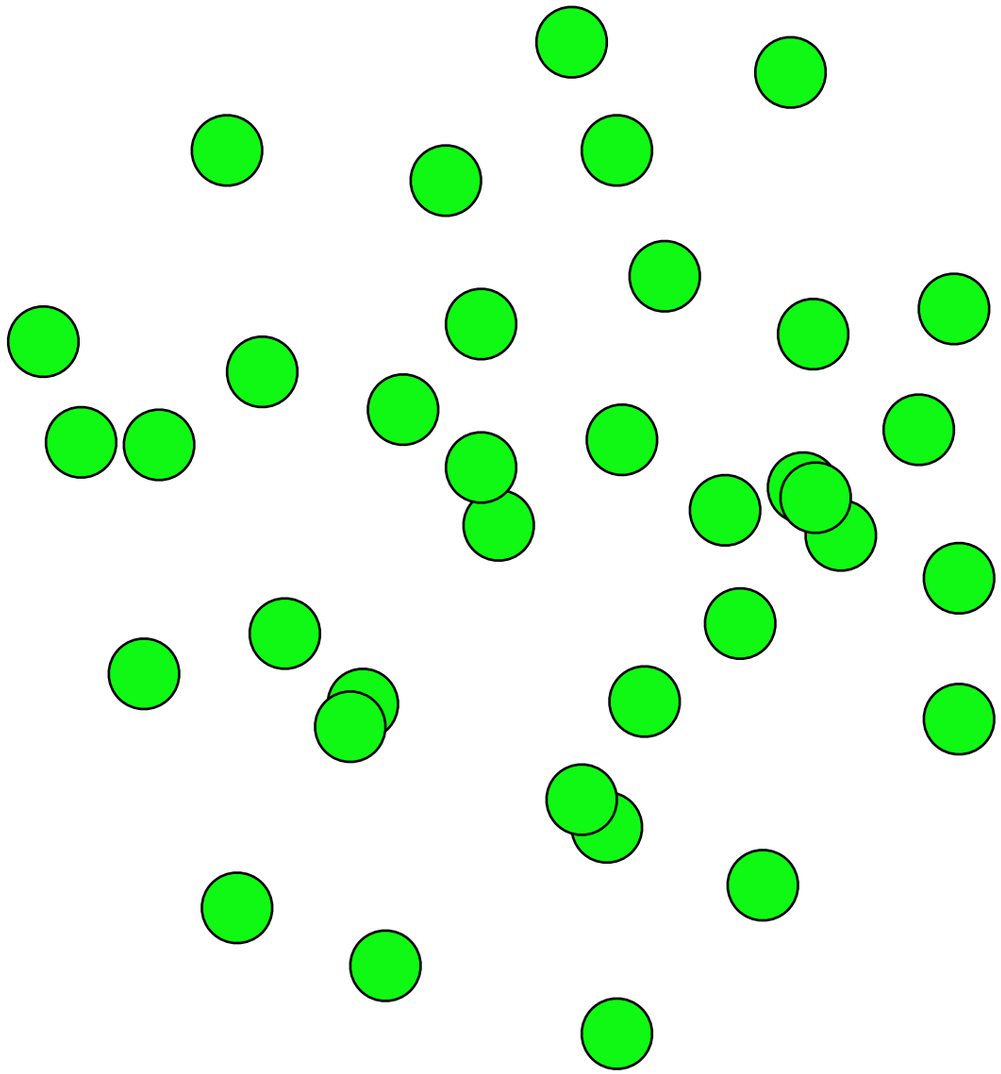


Simple solution:

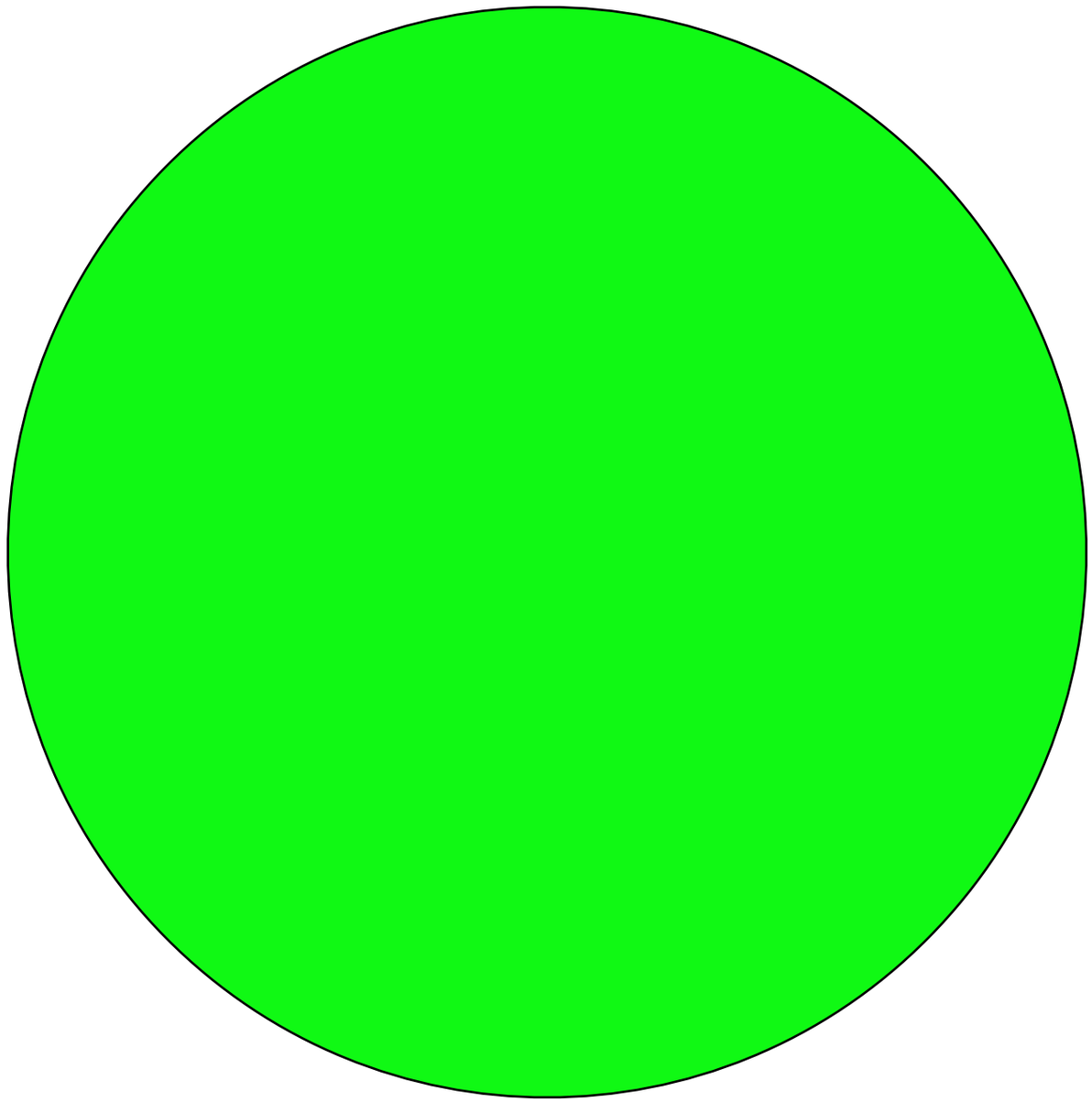
- 1. Read a random sector
  - —*If there is data, the drive is not wiped.*
- 2. Repeat until satisfied.

# A 1TB drive has 2 billion sectors.
# What if we read 10,000 and they are all blank?

# A 1TB drive has 2 billion sectors.
# What if we read 10,000 and they are all blank?



# A 1TB drive has 2 billion sectors.
# What if we read 10,000 and they are all blank?

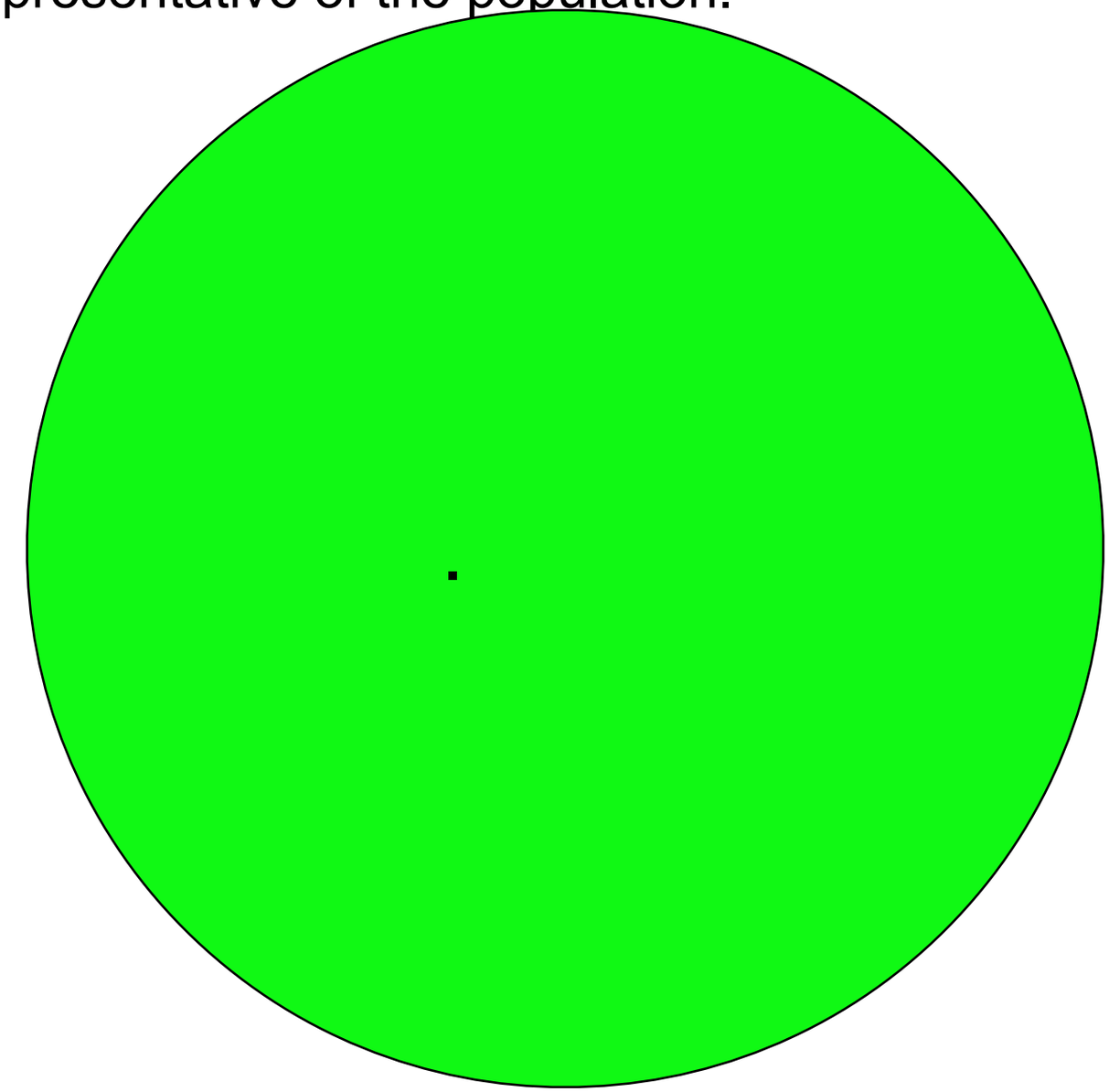


Chances are good that they are all blank.

# Random sampling *won't* find a single written sector.

If the disk has 1,999,999,999 blank sectors  (1 with data)

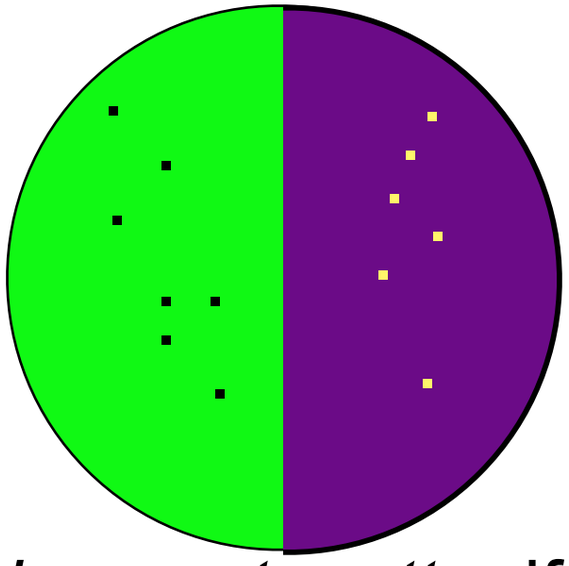
- The sample is representative of the population.



We will only find that 1 sector with exhaustive search.

# If half of the sectors are blank...

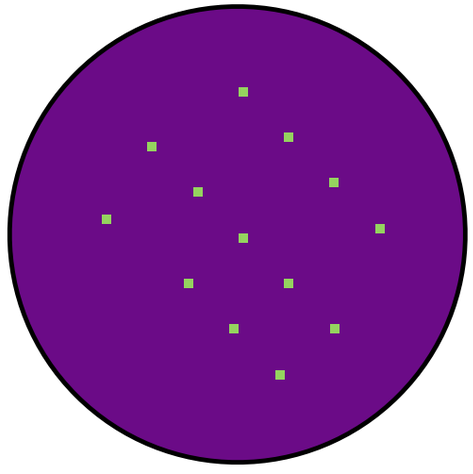
| Sectors | Blank | Data |
|---|---|---|
| Sampled | 5,000 (50%) | 5,000 (50%) |
| Total: | 1,000,000,000 (50%) | 1,000,000,000 (50%) |



The distribution of the data *does not matter* if sampling is random.

# What if the the sampled sectors *are the only blank sectors?*

| Sectors | Blank | Data |
|---|---|---|
| Sampled | 10,000 (100%) | 0 (0%) |
| Total: | 10,000 (0.0005%) | 1,999,990,000 (99%) |



If the the only sectors read are blank...

—*We are incredibly unlucky.*

—*Somebody has hacked our random number generator!*

# This is an example of the "urn" problem from statistics

Assume a 1TB disk has 10MB of data.

- 1TB = 2,000,000,000 = 2 Billion 512-byte sectors!
- 10MB = 20,000 sectors

Read just 1 sector; the odds that it is blank are:

$$\frac{2,000,000,000 - 20,000}{2,000,000,000} = .99999$$

# This is an example of the "urn" problem from statistics

Assume a 1TB disk has 10MB of data.

- 1TB = 2,000,000,000 = 2 Billion 512-byte sectors!
- 10MB = 20,000 sectors

Read just 1 sector; the odds that it is blank are:

$$\frac{2,000,000,000 - 20,000}{2,000,000,000} = .99999$$

Read 2 sectors. The odds that both are blank are:

$$\left(\frac{2,000,000,000 - 20,000}{2,000,000,000}\right)\left(\frac{1,999,999,999 - 20,000}{2,000,000,000}\right) = .99998$$

**first pick**

**second pick**

**Odds we may have missed something**

# The more sectors picked, the less likely we are to miss the data….

$$P(X = 0) = \prod_{i=1}^{n} \frac{((N - (i-1)) - M)}{(N - (i-1))} \qquad (5)$$

| Sampled sectors | Probability of not finding data |
|---|---|
| 1 | 0.99999 |
| 100 | 0.99900 |
| 1000 | 0.99005 |
| 10,000 | 0.90484 |
| 100,000 | 0.36787 |
| 200,000 | 0.13532 |
| 300,000 | 0.04978 |
| 400,000 | 0.01831 |
| 500,000 | 0.00673 |

**Table 1:** Probability of not finding any of 10MB of data on a 1TB hard drive for a given number of randomly sampled sectors. Smaller probabilities indicate higher accuracy.

| Non-null data | | Probability of not finding data |
|---|---|---|
| Sectors | Bytes | with 10,000 sampled sectors |
| 20,000 | 10 MB | 0.90484 |
| 100,000 | 50 MB | 0.60652 |
| 200,000 | 100 MB | 0.36786 |
| 300,000 | 150 MB | 0.22310 |
| 400,000 | 200 MB | 0.13531 |
| 500,000 | 250 MB | 0.08206 |
| 600,000 | 300 MB | 0.04976 |
| 700,000 | 350 MB | 0.03018 |
| 1,000,000 | 500 MB | 0.00673 |

**Table 2:** Probability of not finding various amounts of data when sampling 10,000 disk sectors randomly. Smaller probabilities indicate higher accuracy.

*—Pick 500,000 random sectors*

*—If are all NULL, the disk has p=(1-.00673) chance of having 10MB of non-NULL data*

*—The disk has a 99.3% chance of having less than 10MB of data*

# We sample 64KiB blocks instead of 512-byte sectors

## Sample with 64KiB “blocks” instead of 512-byte sectors.

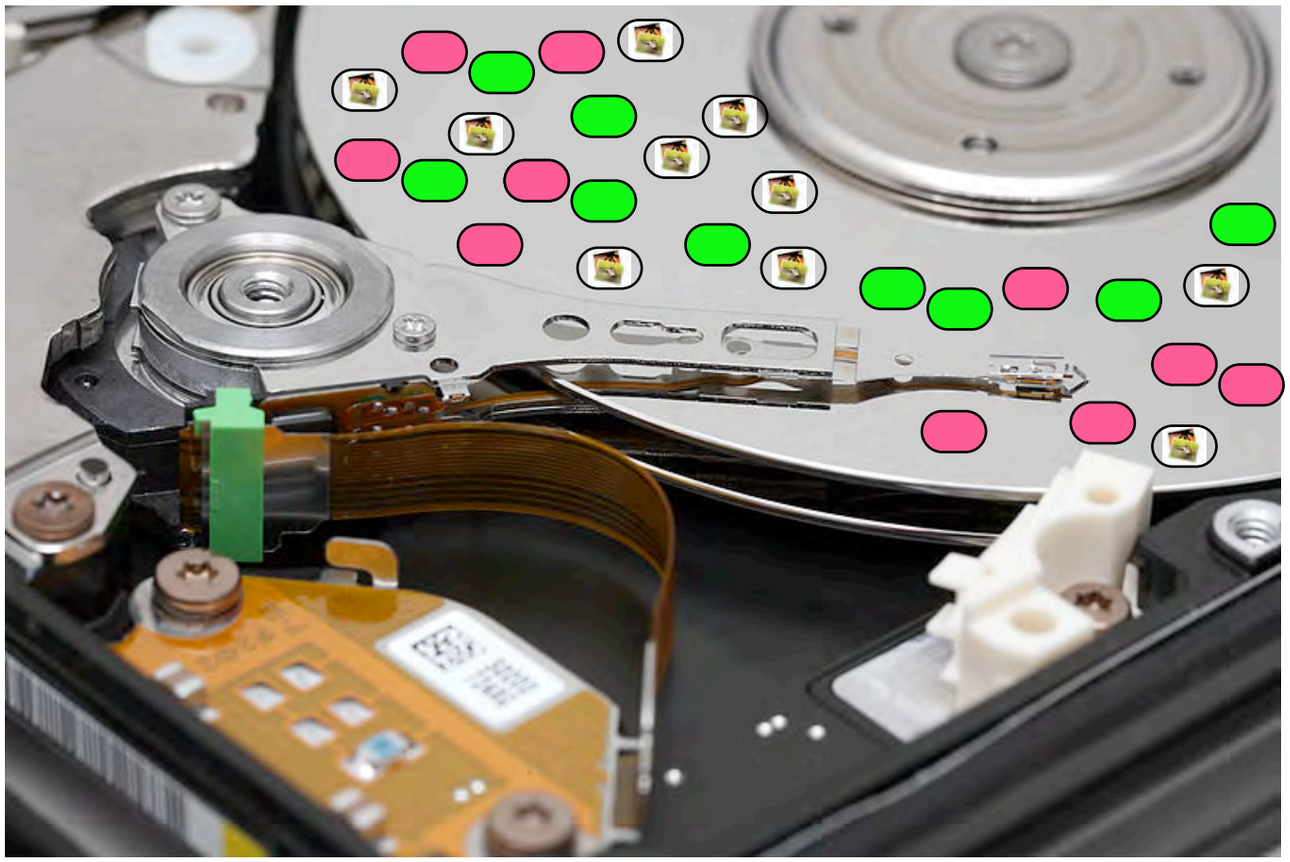
- It takes the same amount of time to read 65,536 bytes as 512 bytes
- Analyze 64KiB block with a 4KiB sliding window
- On a 1TB drive, there are 15,258,789 64KiB sections

## Identify data “type”

- Blank
- JPEG
- Video
- Encrypted

## Update results in real-time

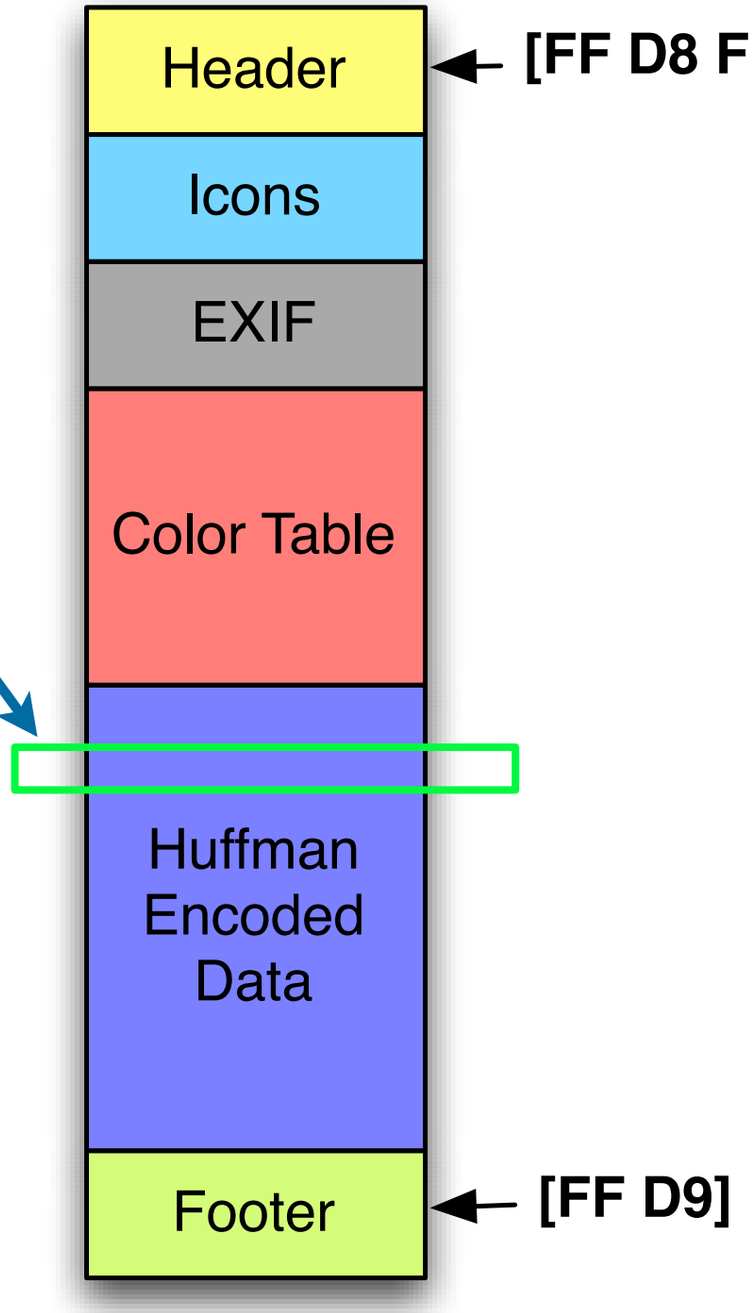
- Provides immediate feedback
- Catches important data faster
- Stop when analyst is satisfied.



# The challenge: identifying a file “type” from a fragment.

Can you identify a JPEG file from reading 4 sectors in the middle?

41,572 bytes

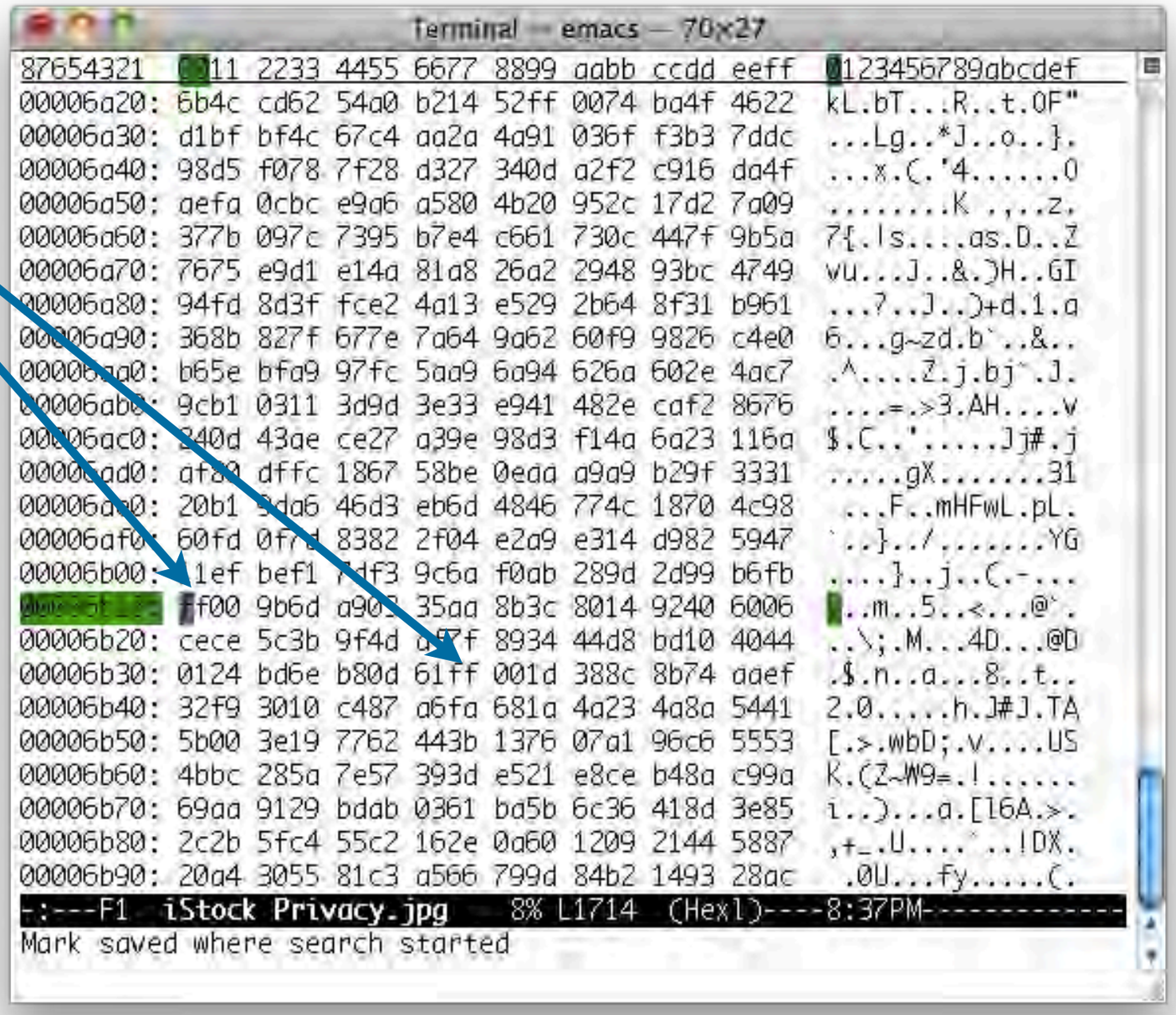



JPEG File

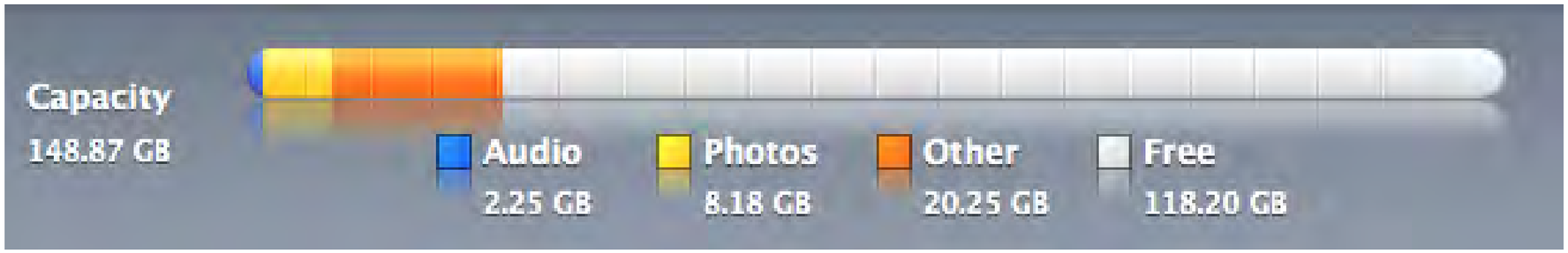
# Machine learning can identify file type of file “fragments.” Features: unigram & bigram frequency.

For example, JPEGs have many “FF00” bigrams

**FF00**



# With random sampling, we accurately predicted the storage allocations reported by iTunes




| | | | |
|---|---|---|---|
| Audio Data reported by iTunes: | 2.25 $GiB$ | 2.42 $GB$ | |
| MP3 files reported by file system: | | 2.39 $GB$ | |
| Estimated MP3 usage with random sampling : | | 2.49 $GB$ | 10,000 random samples |
| | | 2.71 $GB$ | 5,000 random samples |

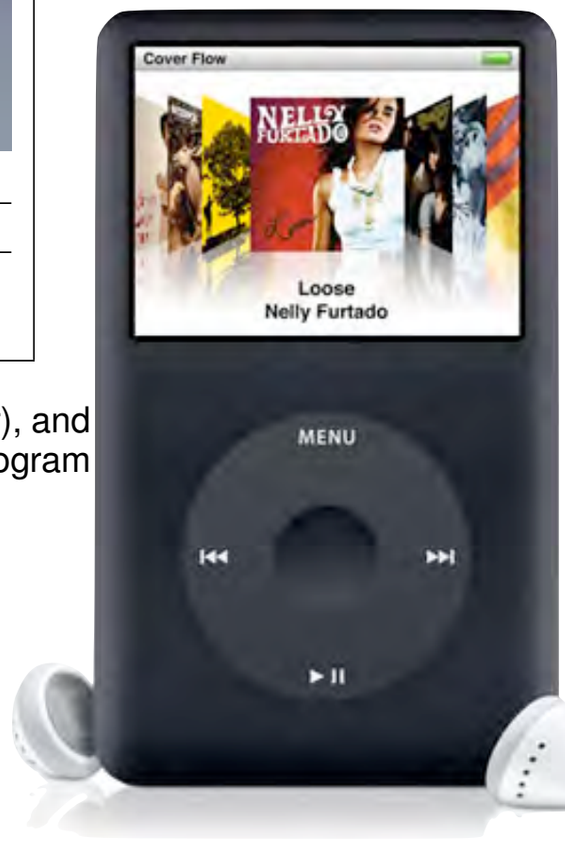


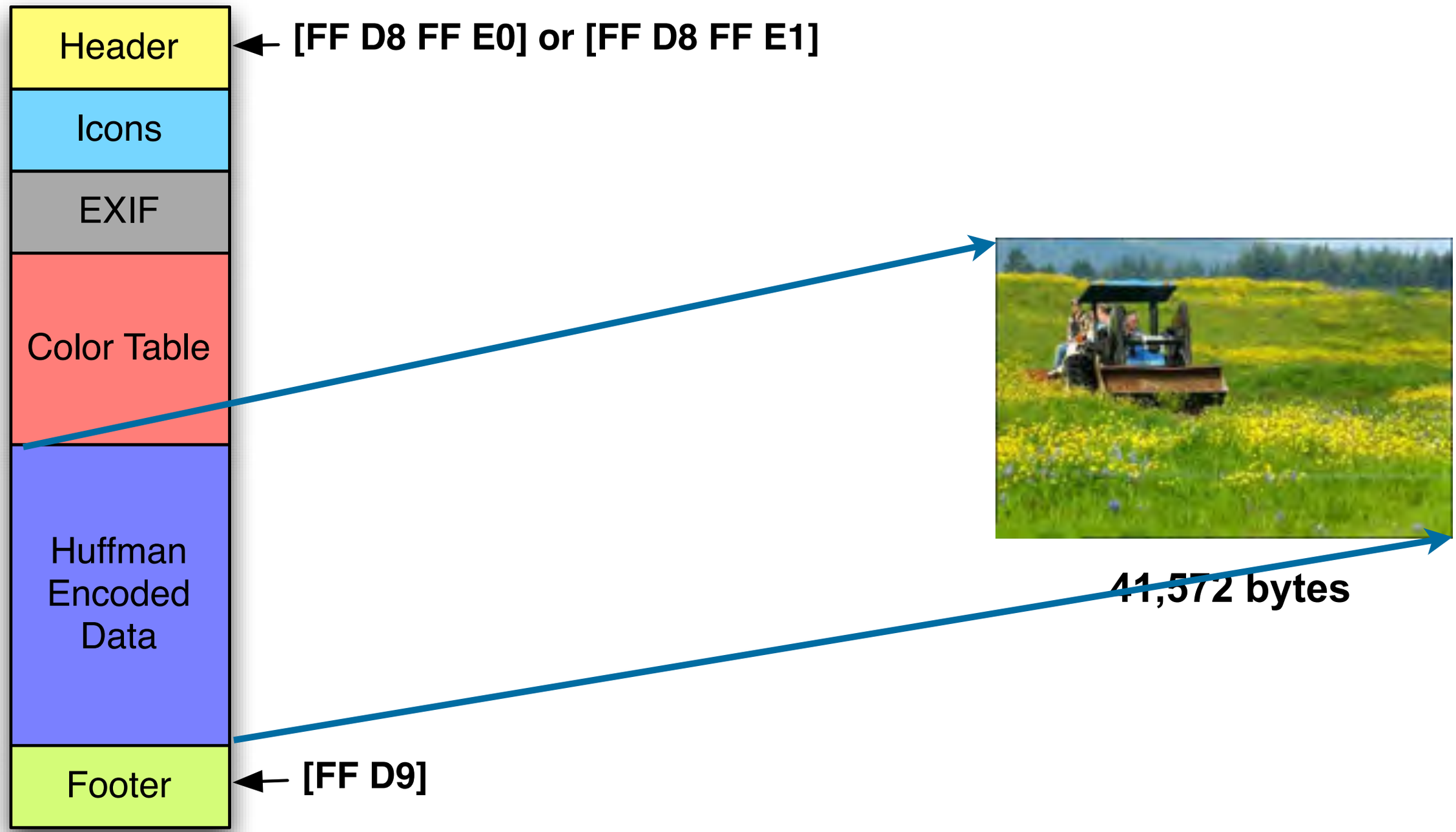

**Figure 1:** Usage of a 160GB iPod reported by iTunes 8.2.1 (6) (top), as reported by the file system (bottom center), and as computing with random sampling (bottom right). Note that iTunes usage actually in GiB, even though the program displays the “GB” label.

## We determined:

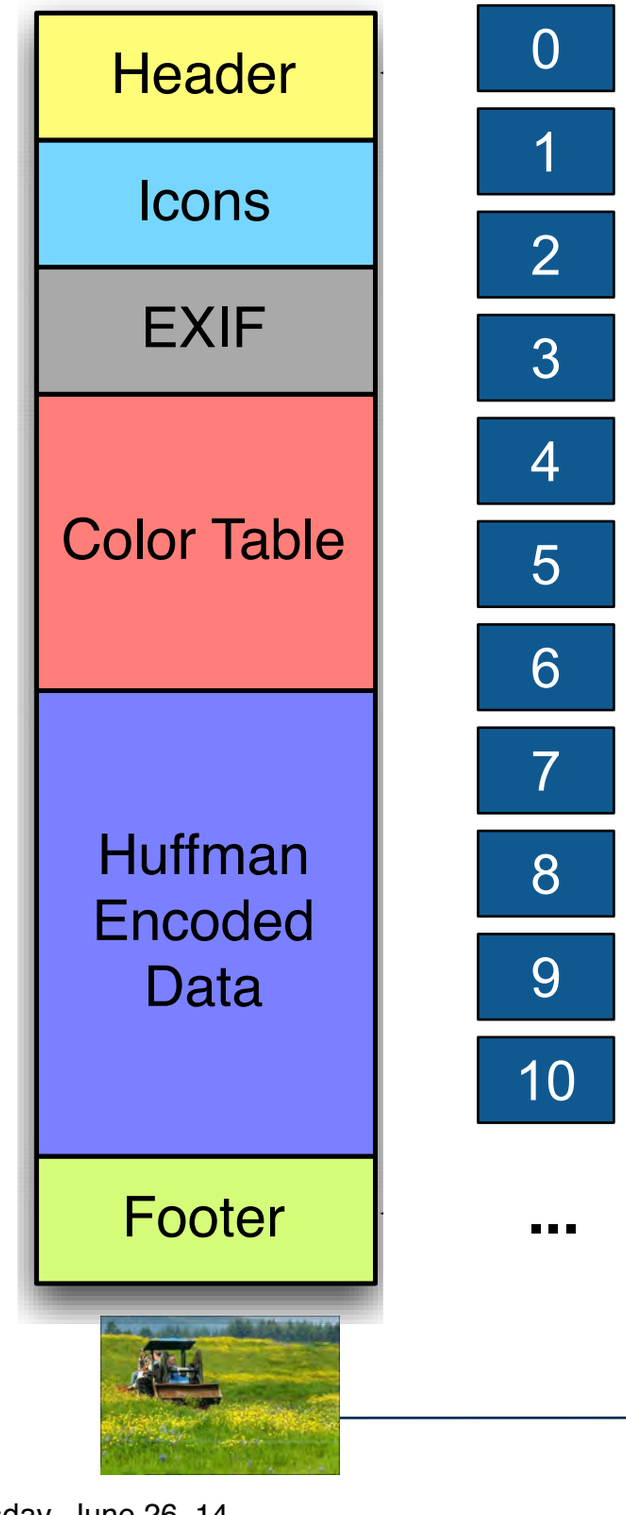
- % of free space; % JPEG; % encrypted

—*Simson Garfinkel, Vassil Roussev, Alex Nelson and Douglas White, Using purpose-built functions and block hashes to enable small block and sub-file forensics, DFRWS 2010, Portland, OR*

# The Huffman Encoded Data in a JPEG is high entropy.



# The 41K JPEG is a sequence of 88 blocks (each 512 bytes)




| Block # | Hex Values... |
|---|---|
| 0 | ffd8 ffe0 0010 4a46 4946 0001 0201 0048... |
| 1 | 0c0c 0c0c ffc0 0011 0800 6a00 a003 0122... |
| 2 | 4fa7 7567 ded2 cac5 8c82 2bf4 9e1c 23f9... |
| 3 | fafd 1527 e459 e934 c173 59ad 9234 f09f... |
| ... | ... |

# Each block has a cryptographic hash. Some are distinct, others are common.

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

Question: Do any of these hash values appear in other files?

# Should these block hashes be in other files?

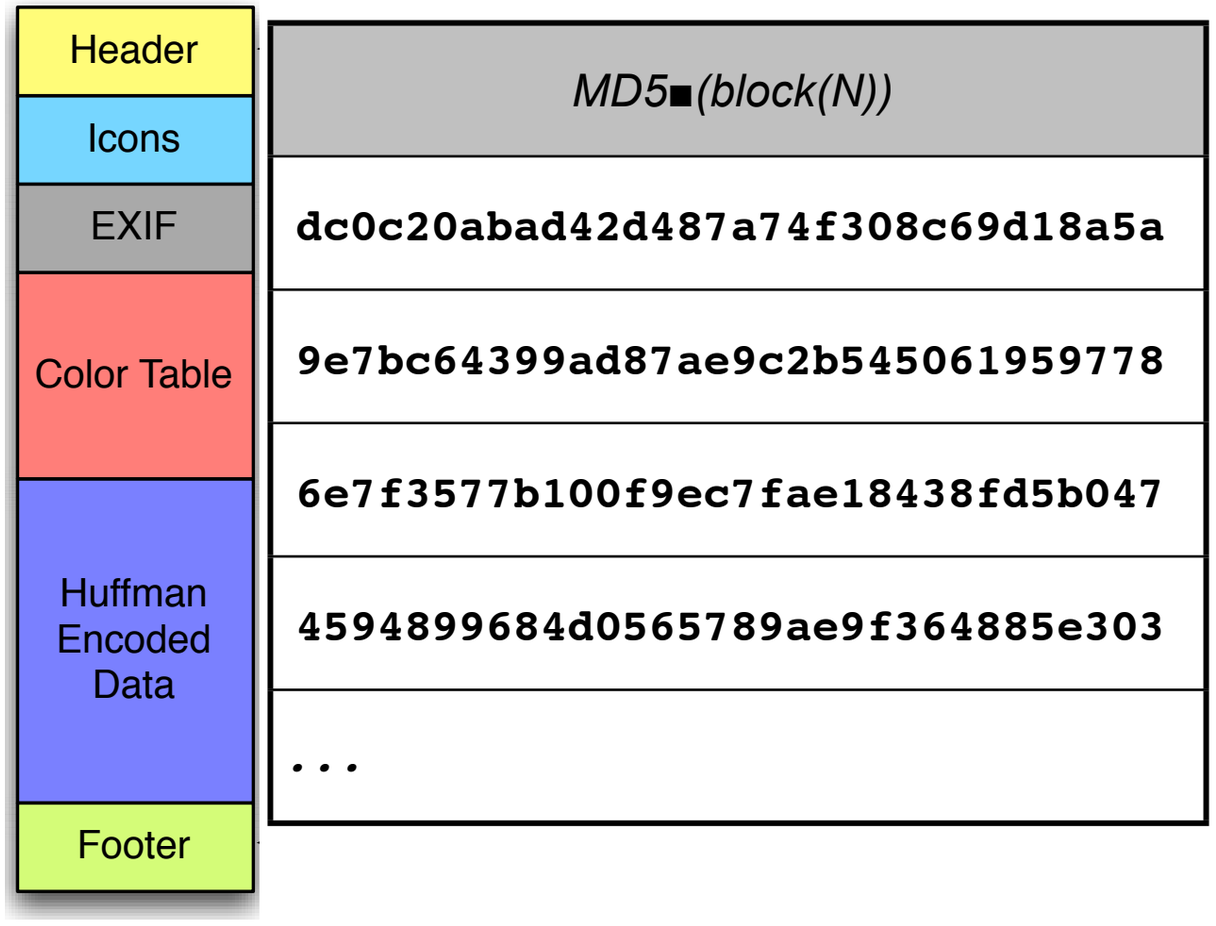
Specific byte sequences in high-entropy data are very rare.

- 512 bytes = $256^{512}$ =$10^{1,233}$ possible sectors

But metadata might be common:

- Specific headers
- Common color tables
- “all black”

We need to survey
a large samples of JPEGs
to find out which hashes are
common and which are distinct.

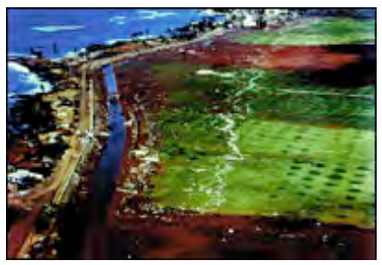


# GOVDOCS1: 1 million files from USG web servers

## Created by NPS DEEP lab in 2010

- ≈1 million heterogeneous files
  - —*Documents (Word, Excel, PDF, etc.); Images (JPEG, PNG, etc.)*
  - —*Database Files; HTML files; Log files; XML*
- Freely redistributable; 100s of different file types
- This database was surprising difficulty to collect, curate, and distribute:
  - —*Scale created data collection and management problems.*
  - —*Copyright, Privacy & Provenance issues.*

## Advantages: persistence & copyright

**<abstract>NOAA's National Geophysical Data Center (NGDC) is building high-resolution digital elevation models (DEMs) for select U.S. coastal regions. … </abstract>**

**<abstract>This data set contains data for birds caught with mistnets and with other means for sampling Avian Influenza (AI)….</abstract>**

# We examined sector hashes from ≃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 the 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 000107.jpg contained distinct hashes...

| 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 | **JPEG Header** |
| 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 | |
| 692527fa35782db8592486343 6d45d7f | offset 10240-10751 | 1 | |
| 76dbb9b469273d0e0e467a55728b7883 | offset 10752-11263 | 1 | |

# The middle of 000107.JPG appear elsewhere...

| 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 |
| 13d33222848d5b25e26aefb87dbdf294 | 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 |
| 51a50e6148d13111669218dc40940ce5 | 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 |

**Exif...**

This appears to be metadata that is shared with other JPEGs.

- Source: PhotoShop

# Example: Block 37 had 9198 collisions..
# The sector is filled with blank lines 100 characters long...

```
13d33222848d5b25e26aefb87dbdf294   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
0000100: 2020 2020 2020 2020 2020 2020 2020 2020
0000110: 2020 2020 2020 2020 2020 2020 2020 2020
0000120: 2020 2020 2020 2020 2020 2020 2020 2020
0000130: 2020 2020 2020 2020 2020 2020 2020 2020
0000140: 2020 2020 2020 2020 2020 200a 2020 2020                .
0000150: 2020 2020 2020 2020 2020 2020 2020 2020
0000160: 2020 2020 2020 2020 2020 2020 2020 2020
...
```

...

# Block 45 has 83 collisions.
# It also appears to contain EXIF metadata

```
51a50e6148d13111669218dc40940ce5   offset 23040-23551        83
$ 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
0000130: 4742 2063 6f6c 6f75 7220 7370 6163 6520  GB colour space
0000140: 2d20 7352 4742 0000 0000 0000 0000 0000  - sRGB..........
0000150: 002e 4945 4320 3631 3936 362d 322e 3120  ..IEC 61966-2.1
0000160: 4465 6661 756c 7420 5247 4220 636f 6c6f  Default RGB colo
0000170: 7572 2073 7061 6365 202d 2073 5247 4200  ur space - sRGB.
```

# Block 48 had 84 collisions..
# It appears to contain part of a JPEG color table...

```
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  ././.0501O.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
0000140: 4a4d 934d dc4e 254e 6e4e b74f 004f 494f  JM.M.N%NnN.O.OIO
0000150: 934f dd50 2750 7150 bb51 0651 5051 9b51  .O.P'PqP.Q.QPQ.Q
0000160: e652 3152 7c52 c753 1353 5f53 aa53 f654  .R1R|R.S.S_S.S.T
0000170: 4254 8f54 db55 2855 7555 c256 0f56 5c56  BT.T.U(UuU.V.V\V
0000180: a956 f757 4457 9257 e058 2f58 7d58 cb59  .V.WDW.W.X/X}X.Y
0000190: 1a59 6959 b85a 075a 565a a65a f55b 455b  .YiY.Z.ZVZ.Z.[E[
00001a0: 955b e55c 355c 865c d65d 275d 785d c95e  .[.\5\.\.]']x].^
```

# With blocks of 512 bytes and 4KiB, the vast majority of sectors had distinct hashes.

**Table 1. Incidence of singleton, paired, and common sectors in three file corpora.**

| No. of blocks | Govdocs | OpenMalware 2012 | 2009 NSRL RDS |
|---|---|---|---|
| **Block size: 512 bytes** | | | |
| Singleton | 911.4 M (98.93%) | 1,063.1 M (88.69%) | N/A |
| Pair | 7.1 M (.77%) | 75.5 M (6.30%) | N/A |
| Common | 2.7 M (.29%) | 60.0 M (5.01%) | N/A |

**Young, Foster, Garfinkel & Fairbanks, IEEE Computer, Dec. 2012**

# File systems align most files on sector boundaries. We match *file block hashes* with *disk sector hashes*.

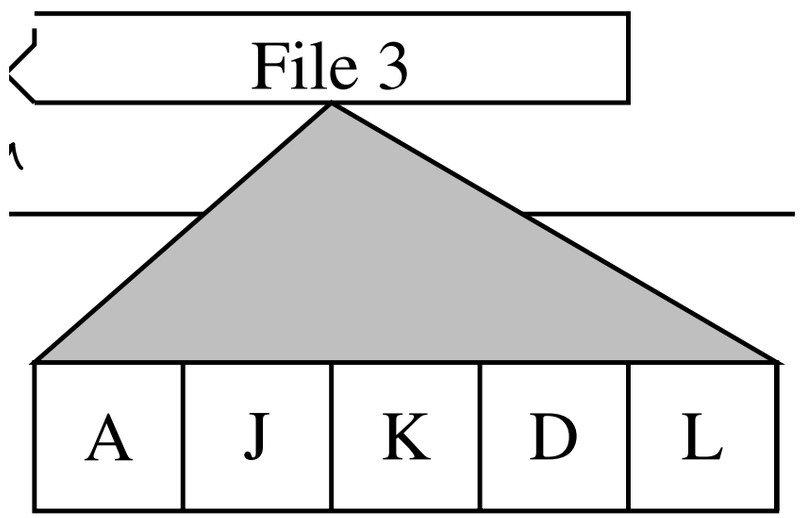
| Block # | Byte Range | MD5*(block(N)) |
|---|---|---|
| 0 | 0- 511 | **dc0c20abad42d487a74f308c69d18a5a** |
| 1 | 512-1023 | **9e7bc64399ad87ae9c2b545061959778** |
| 2 | 1024-1535 | **6e7f3577b100f9ec7fae18438fd5b047** |
| 3 | 1536-2047 | **4594899684d0565789ae9f364885e303** |
| 4 | ... | |

*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

# This means we can use distinct sectors to find known content.

## Method #1 — Random sampling

- Read & hash randomly chosen sectors.
- Lookup hash values in a database of block hashes.
- Distinct hash implies presence of files.
- Advantage: Can find presence of target content very quickly




## Method #2 — Full media sampling

- Read & hash every disk sector.
- Lookup hash values in a database of block hashes.
- Distinct hash imply presence of files.
- Advantage: Can find a single sector of target content
- Disadvantage: Requires a very fast database
  - —*1TB data in 208 minutes*
    - ≃ **80 Mbyte/sec**
    - ≃ **150,000 512-byte sectors/sec**
    - ≃ **150,000 database lookups/sec**

# Combining a Bloom filter & database, we can perform up to 2.7M TPS on low-cost hardware

**Table 2. Total transactions per second (TPS) for best execution.**

| Bloom filter | | | Database | | TPS at 1 M lookups | | TPS at 1,200 seconds | |
|---|---|---|---|---|---|---|---|---|
| *k* | *M* | Size | Strategy | Size | Present | Absent | Present | Absent |
| **100 million records** | | | | | | | | |
| 3 | 31 | 257 MiBytes | B-tree (preload) | 2.3 GiBytes | 35.3 K | 49.5 K | 161.3 K | 1.8 M |
| 3 | 31 | 257 MiBytes | B-tree | 2.3 GiBytes | 11.6 K | 565.8 K | 156.8 K | 2.3 M |
| 3 | 31 | 257 MiBytes | Hash map | 5.3 GiBytes | 13.9 K | 656.9 K | 641.9 K | 3.0 M |
| 3 | 31 | 257 MiBytes | Flat map | 2.2 GiBytes | 28.2 K | 746.9 K | 356.4 K | 2.6 M |
| 3 | 31 | 257 MiBytes | Red/black tree | 6.0 GiBytes | 12.9 K | 694.5 K | 187.0 K | 2.7 M |
| **1 billion records** | | | | | | | | |
| 3 | 34 | 2.1 GiBytes | B-tree (preload) | 23 GiBytes | 2.2 K | 6.1 K | 3.6 K | 23.1 K |
| 3 | 33 | 1.1 GiBytes | B-tree | 23 GiBytes | 2.6 K | 85.8 K | 3.7 K | 114.9 K |
| 3 | 33 | 1.1 GiBytes | Hash map | 57 GiBytes | – | – | 0.3 K | 3.1 K |
| 3 | 34 | 2.1 GiBytes | Flat map | 22 GiBytes | – | – | 0.4 K | 4.0 K |
| 3 | 33 | 1.1 GiBytes | Red/black tree | 60 GiBytes | – | – | 0.1 K | 1.4 K |

Hardware: 8GiB Laptop; 250GB external SSD.

—*"Distinct sector hashes for target file detection," Young, Garfinkel, Foster & Fairbanks, IEEE Computer, Dec. 2012*

# We have created “hashdb” (hash database) for creating and maintaining hash databases.

“Hashdb” is a C++ package that provides:

- hashdb library — creates, searches, and manages hash databases
- hashdb command — manually building and searching database
- scan_hashdb — A “bulk_extractor” scanner — search for known content in bulk data.

“scan_hashid” — integrates hashdb with bulk_extractor

Available on github:

- https://github.com/simsong/hashdb — the library
- https://github.com/simsong/bulk_extractor — bulk data research platform

Next steps:

- Testing hashdb in an operational environment.
- Create a hashdb with the sector hashes of every ingested device.

  —*64GB phone has 134M 512-byte sectors ≈ 6GB hashdb database*

# With hashdb, we can field-deploy a billion-row hash database for triage and exhaustive search.

## Use Case #1: Rapidly search for known content (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 Content | p (prob of missing content) |
|---|---|
| 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 content:

- Time to read data & search database: 208 minutes

Technique is file type and file system agnostic

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

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

```
a097 83a1 ed96 26a6 3c69 3d0f 750a 2399  ......&.<i=.u.#.
a2b5 bea7 692f 5847 a38a dd53 082c add5  ....i/XG...S.,..
5061 b64c 721d 864b 90b6 b55f bb04 735c  Pa.Lr..K..._..s\
9448 6730 5453 df64 813e b603 5795 2242  .Hg0TS.d.>..W."B
e9c8 7454 7322 7cdc b60e 97af 2f64 2728  ..tTs"|...../d'(
3cfb 84bd 2a84 2dfe 50ea 5935 c349 1513  <XYZ@COMPANY.COM
a9e9 e92c a3f8 6e46 0530 8a88 c7a2 5d2b  ...,..nF.0....]+
d89d 77cc fe1e f637 f3f3 d0af 1b47 c09b  ..w....7.....G..
```

# Optimistic Decoding

# In addition to looking for known content, forensic investigators search for “identity information.”

**XYZ@COMPANY.COM**

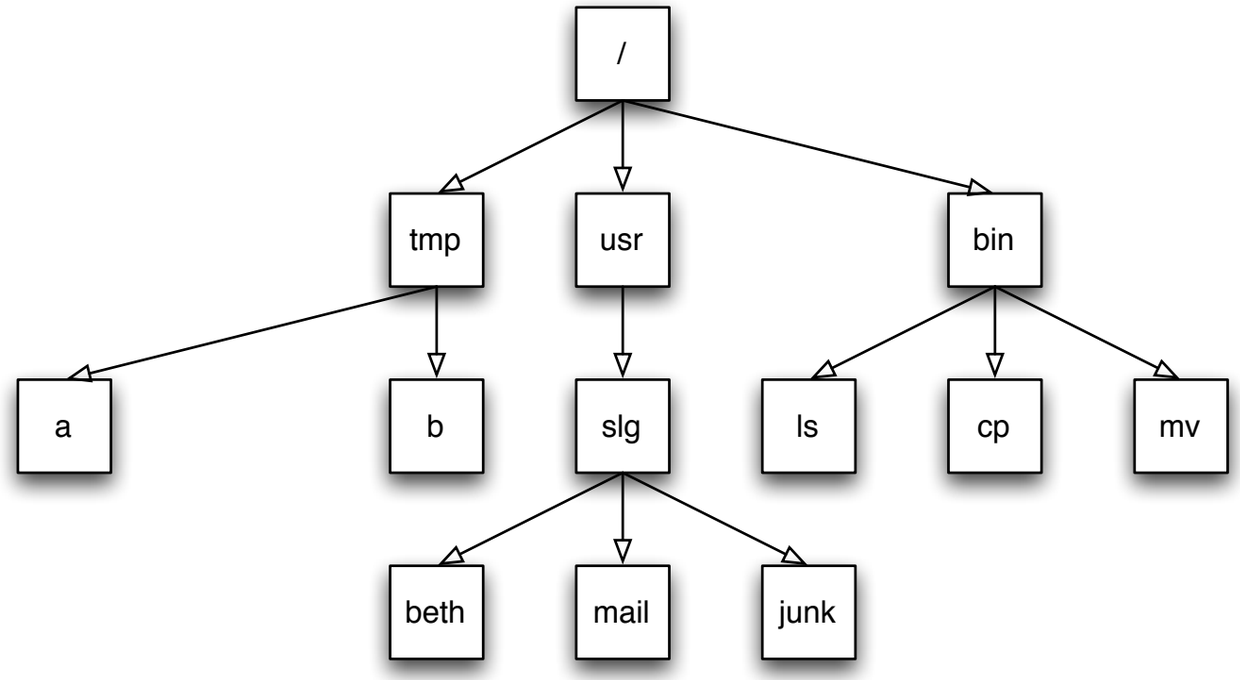
## Applications:

- Watch lists
- Identify known associates
- Inter-case correlation

# There are two ways to find email addresses on a drive.

## Approach #1:

- Extract text from every file.
- Scan the files with regular expressions



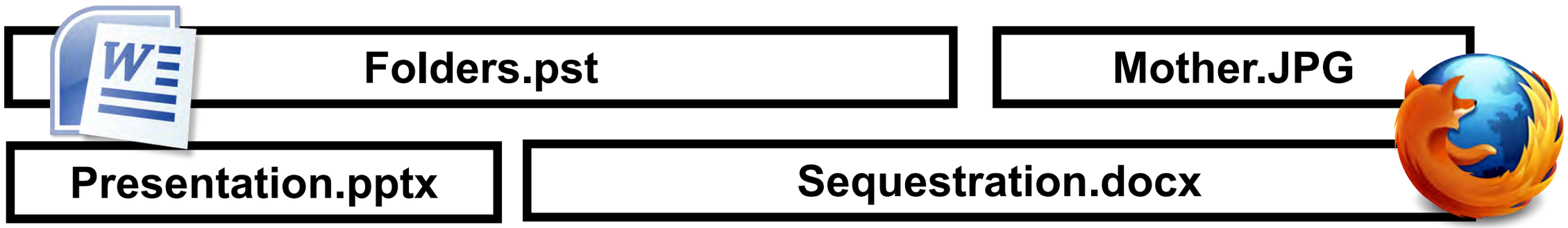

## Approach #2:

- Extract text from the “bulk data”
- Scan the text with regular expressions

```
$ cat /dev/disk1 | strings | grep '[a-zA-Z]+@[\-a-zA-Z._]+'
```

# Email addresses are extracted from *document files* by converting to text then scanning with regular expressions.

File ➳ Text ➳ RegEx ➳ Email Addresses



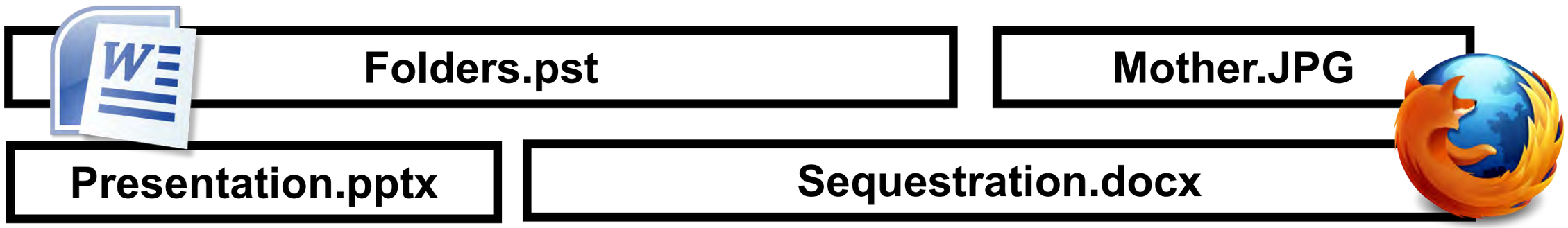

➳

TEXT TEXT TEXT
**XYZ@COMPANY.**
**COM** TEXT TEXT
TEXT TEXT

➳ **XYZ@COMPANY.COM**

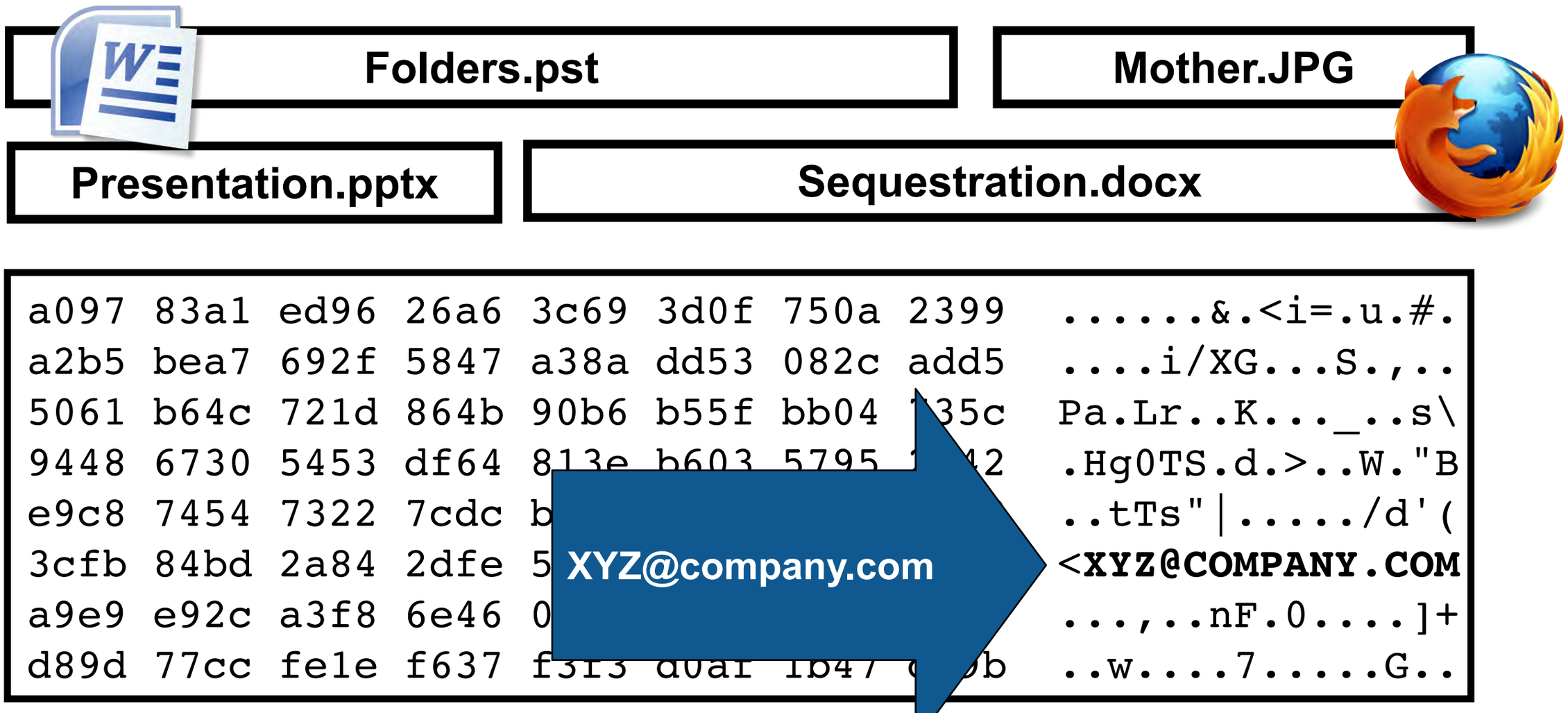
# Regular expressions can also extract email addresses from data not in files — “bulk data.”

[bulk data] ➳ RegEx ➳ Email Addresses




```
a097 83a1 ed96 26a6 3c69 3d0f 750a 2399   ......&.<i=.u.#.
a2b5 bea7 692f 5847 a38a dd53 082c add5   ....i/XG...S.,..
5061 b64c 721d 864b 90b6 b55f bb04 735c   Pa.Lr..K..._..s\
9448 6730 5453 df64 813e b603 5795 2242   .Hg0TS.d.>..W."B
e9c8 7454 7322 7cdc b60e 97af 2f64 2728   ..tTs"|...../d'(
3cfb 84bd 2a84 2dfe 50ea 5935 c349 1513   <XYZ@COMPANY.COM
a9e9 e92c a3f8 6e46 0530 8a88 c7a2 5d2b   ...,..nF.0....]+
d89d 77cc fe1e f637 f3f3 d0af 1b47 c09b   ..w....7.....G..
```

# It’s easy to see plain email addresses in bulk data.



# Challenge: email addresses can be encoded in many ways.

XYZ@company.com

- Unicode: “XYZ@company.com”

```
58 59 5a 40 63 6f 6d 70 61 6e 79 2e 63 6f 6d
```

- Base 16: “58595a40636f6d70616e792e636f6d0a”

```
3538 3539 3561 3430 3633 3666 3664 3730  58595a40636f6d70
3631 3665 3739 3265 3633 3666 3664 3061  616e792e636f6d0a
```

- Base 64: “WFlaQGNvbXBhbnkuY29tCg===”

```
5746 6c61 5147 4e76 6258 4268 626e 6b75  WFlaQGNvbXBhbnku
5932 3974 4367 3d3d 3d0a                 Y29tCg===.
```

- Compression: echo “XYZ@company.com” | compress | xxd

```
1f9d 9058 b268 0132 e64d 1b38 61dc e471  ...X.h.2.M.8a..q
51b0 8d02                                Q…
```

# Compression works by eliminating repeated sequences:

Computers use compression to save memory:

```
5859 5a40 636f 6d70 616e 792e 636f 6d20  XYZ@company.com
4142 4340 636f 6d70 616e 792e 636f 6d20  ABC@company.com
4445 4640 636f 6d70 616e 792e 636f 6d20  DEF@company.com
```

Compressed with “gzip:”

```
1f8b 0800 0000 0000 0203 8b88 8c72 48ce  .............rH.
cf2d 48cc abd4 03d2 0a8e 4ece 287c 1757  .-H.......N.(|.W
3714 3e00 b455 c1c5 3000 0000            7.>..U..0...
```

Compressed email addresses do not “look” like email addresses!

—*Forensic tools must decompress FIRST to identify compressed email addresses.*

# It’s hard to see compressed email address in bulk data.

```
e327 962d 6450 3d91 c945 3bed 97a6 a4cd   .'.-dP=..E;.....
1[illegible]b 0800 0000 0000 0203 8b88 8c72 48ce   .............rH.
[illegible]8cc abd4 03d2 0a8e 4ece 287c 1757   .-H.......N.(|.W
[illegible]714 3e00 b455 c1c5 3000 0000 0000 0000   7.>..U..0.......
0a8e 4ece 287c 1757 3714 3e00 a175 10ed   ..N.(|.W7.>..u..
```

**Folders.pst**

**Mother.JPG**

**Presentation.pptx**

**Sequestration.docx**

```
a097 83a1 ed96 26a6 3c69 3d0f 750a 2399   ......&.<i=.u.#.
a2b5 bea7 692f 5847 a38a dd53 082c add5   ....i/XG...S.,..
5061 b64c 721d 864b 90b6 b55f bb04 735c   Pa.Lr..K..._..s\
9448 6730 5453 df64 813e b603 5795 2242   .Hg0TS.d.>..W."B
[illegible]8 7454 7322 7cdc b60e 97af 2f64 2728   ..tTs"|...../d'(
[illegible]4bd 2a84 2dfe 50ea 5935 c349 1513   <XYZ@COMPANY.COM
[illegible]e92c a3f8 6e46 0530 8a88 c7a2 5d2b   ...,..nF.0....]+
d89d 77cc fe1e f637 f3f3 d0af 1b47 c09b   ..w....7.....G..
```

# It's hard to see compressed email address in bulk data.

```
e327 962d 6450 3d91 c945 3bed 97a6 ..cd   .'.-dP=..E;.....
1... 0800 0000 0000 0...           ....   ............rH.
    ...8cc abd4 03d2 0...          ....   .-H.......N.(|.W
    7714 3e00 b455 c1c5 3...       ....   7.>..U..0.......
    0a8e 4ece 287c 1757 3714 3e00 a175 ..ed   ..N.(|.W7.>..u..
```

XYZ@company.com
ABC@company.com
DEF@company.com

**Folders.pst**

**Mother.JPG**

**Presentation.pptx**

**Sequestration.docx**

```
a097 83a1 ed96 26a6 3c69 3d0f 750a 2399   ......&.<i=.u.#.
a2b5 bea7 692f 5847 a38a dd53 082c add5   ....i/XG...S.,..
5061 b64c 721d 864b 90b6 b55f bb04 735c   Pa.Lr..K..._..s\
9448 6730 5453 df64 813e b603 5795 2242   .Hg0TS.d.>..W."B
e...8 7454 7322 7cdc b60e 97af 2f64 2728   ..tTs"|...../d'(
    ...4bd 2a84 2dfe 50ea 5935 c349 1513   <XYZ@COMPANY.COM
    ...e92c a3f8 6e46 0530 8a88 c7a2 5d2b   ...,..nF.0....]+
    d89d 77cc fe1e f637 f3f3 d0af 1b47 c09b   ..w....7.....G..
```

# Example: Microsoft Word

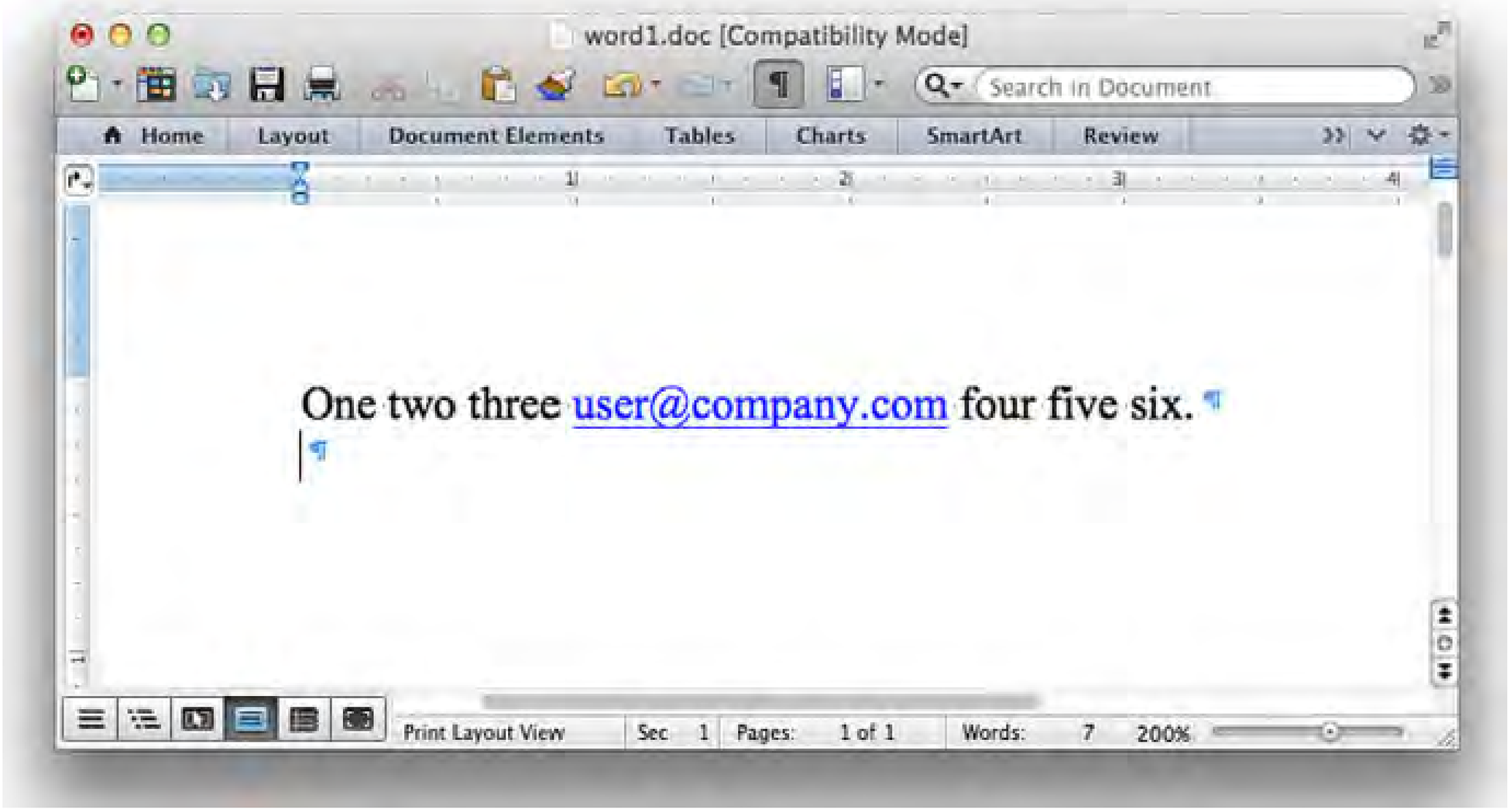
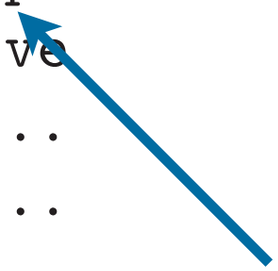



FIG. 1—*A Microsoft Word file containing a single sentence followed by a blank line.*

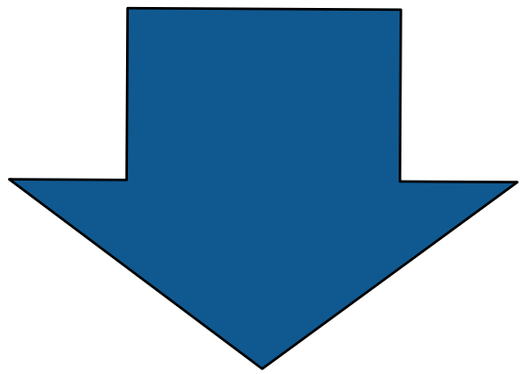
# Word's .doc format stores plain text (UTF-8 and UTF-16)

```
00000a00: 4f6e 6520 7477 6f20 7468 7265 6520 1320  One two three .
00000a10: 4859 5045 524c 494e 4b20 226d 6169 6c74  HYPERLINK ‘‘mailt
00000a20: 6f3a 7573 6572 4063 6f6d 7061 6e79 2e63  o:user@company.c
00000a30: 6f6d 2220 1475 7365 7240 636f 6d70 616e  om’’ .user@compan
00000a40: 792e 636f 6d15 2066 6f75 7220 6669 7665  y.com. four five
00000a50: 2073 6978 2e0d 0d00 0000 0000 0000 0000   six.............
00000a60: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000a70: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```



# Word's .docx format stores content as compressed XML

```
00000990: 0300 504b 0304 1400 0600 0800 0000 2100  ..PK.........!.
000009a0: ea76 7d78 d702 0000 c607 0000 1100 0000  .v}x............
000009b0: 776f 7264 2f64 6f63 756d 656e 742e 786d  word/document.xm
000009c0: 6ca4 55db 729b 3010 7def 4cff 81d1 7b0c  l.U.r.0.}.L...{.
000009d0: 7673 7198 e034 b7a6 79e8 3453 b7cf 1d19  vsq..4..y.4S....
000009e0: 0468 8cb4 1a49 98ba 5fdf 95b8 d889 ddd6  .h...I.._.......
000009f0: 495e 0c98 b367 cf9e 5d2d 1797 bf44 15ac  I^...g..]-...D..
00000a00: 9836 1c64 42c6 a388 044c a690 7159 24e4  .6.dB....L..qY$.
00000a10: c7f7 4f47 5312 184b 6546 2b90 2c21 6b66  ..OGS..KeF+.,!kf
```



**Uncompress**

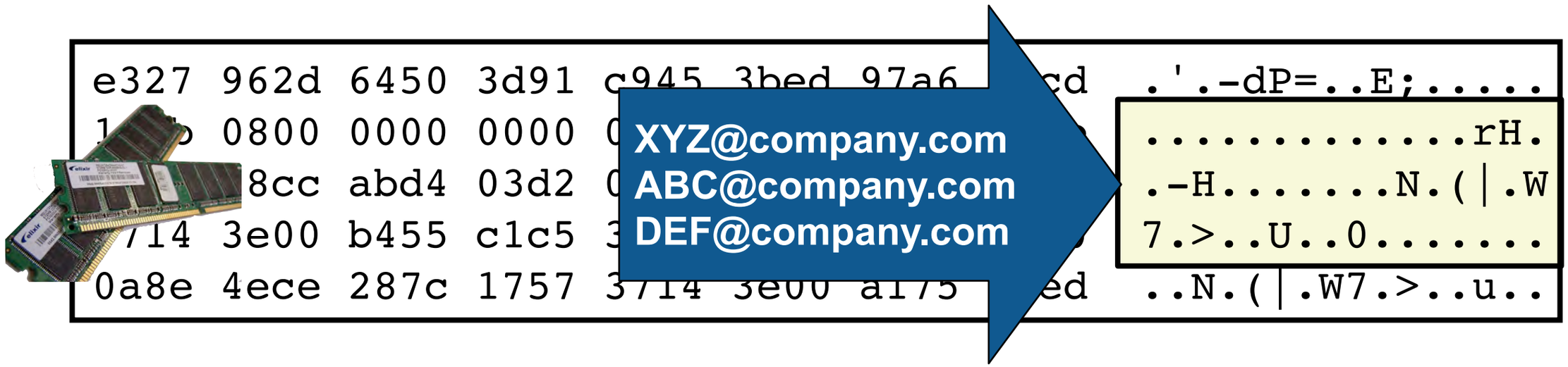
```
w:t></w:r><w:hyperlink r:id=''rId5'' w:history=''1''><w:r w:rsidRPr=
''004B377A''><w:rPr><w:rStyle w:val=''Hyperlink''/></w:rPr><w:t>user
@company.com</w:t></w:r></w:hyperlink><w:r><w:t
```

# PDFs generated by Word are compressed PDF streams

```
%PDF-1.3
%\304\345\362\345\353\247\363\240\320\304\306
4 0 obj
<< /Length 5 0 R /Filter /FlateDecode >>
stream
x^A\225\222\313n\2030^PE\367\376\212\2734\213:\266^C^FvU\252n
\272\251''Y\352\242\352\242BAi^U\240\201\246\217\277\257\237
\224''\224\250^B\311^^{4>\367\316^\\261\305^QB\332?+\344\252
@\277\303^CZ\254n^F\201j^@w\337P\231<\316d\352c\273)9r^\260R
\241j\260\321$\363\231a\321^MVZ^K\306!\240k<\202\336'\2702^U
@\333]bK\201\342=l>\343U^W\216^H\335\3671+\256H\360^D}\207[m
\240\355[^B^KTBqV\346\251\372e#^[\215Kl\247!^R\304\327\372k
\313&@\253\300\3305      q\324o\317\341\244\375f\223\313^Y^\
_\202\223Y\311\224JE\200#\316\270\363p\324\310\326\257^\364
\274^CR\311\377\2263}\250\235\324^I
\303^[@(FK\342\325^W\233v'^N\263\22
^M\305T\333\330P\241\314\320\244\30
^H1\261\261I;'\222\357\342=j?\243K1
^\\336\366^G\212q\250^D
endstream
endobj
```

```
q Q q 12 12 588 768 re W n /Cs1 cs 0 0 0 sc q 0.24 0 0 0.24 90 708.96
cm BT 50 0 0 50 0 0 Tm /TT1.0 1 Tf [ (O) -0.2 (ne) 0.2 ( t) 0.2 (w)
  -0.2 (o t) 0.2 (hre) 0.2 (e) 0.2 ( ) ] TJ ET Q 0 0 1 sc q 0.24 0 0
0.24 160.9746 708.96 cm BT 50 0 0 50 0 0 Tm /TT1.0 1 Tf [ (us) -0.2
  (e) 0.2 (r@) 0.1 (c) 0.2 (om) 0.2 (pa) 0.2 (ny.c) 0.2 (om) ] TJ ET Q
0 0 0 sc q 0.24 0 0 0.24 259.6641 708.96 cm BT 50 0 0 50 0 0 Tm /TT1.0
1 Tf ( ) Tj ET Q q 0.24 0 0 0.24 262.6641 708.96 cm BT 50 0 0 50 0 0
Tm /TT1.0 1 Tf [ (f) -0.5 (our f) -0.5 (i) 0.2 (ve) 0.2 ( s) -0.2 (i)
  0.2 (x.) ] TJ ET Q q 0.24 0 0 0.24 324.3281 708.96 cm BT 50 0 0 50 0
0 Tm /TT1.0 1 Tf ( ) Tj ET Q 0 0 1 sc 161.04 707.28 m 259.68 707.28 l
259.68 707.04 l 161.04 707.04 l h f 0 0 0 sc q 0.24 0 0 0.24 90 695.28
cm BT 50 0 0 50 0 0 Tm /TT1.0 1 Tf ( ) Tj ET Q Q
```

# Most digital forensic tools ignore compressed email addresses in bulk data.




## Today's tools ignore most kinds of encoding for bulk data:

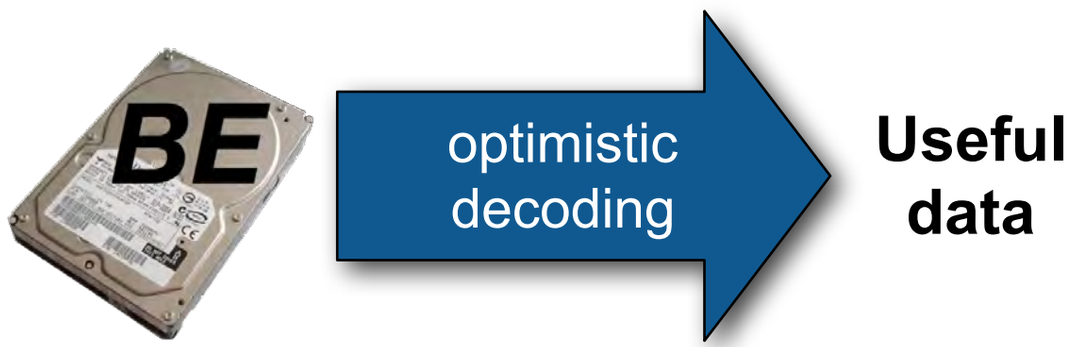
- Compression:
  - —*zlib (gzip, ZIP)*
  - —*RAR*
  - —*Windows Hibernation (Microsoft Xpress)*
- Simple obfuscation
  - —*ROT13, XOR(255)*

# bulk_extractor is a leading bulk data analysis tool.

High-performance digital forensics tool runs on Windows/Mac/Linux

Identifies and extracts a wide variety of formatted info in free-formatted data:

- Domain Names; Email addresses; URLs
- Search terms; Facebook IDs; JSON data
- KML files; VCARDs
- ZIP & RAR files; Carved JPEGs; EXIF data
- PCAP files; Ethernet Addresses; TCP/IP Connections; etc.
- ELF & PE headers; Windows Prefetch files; Windows LNK files



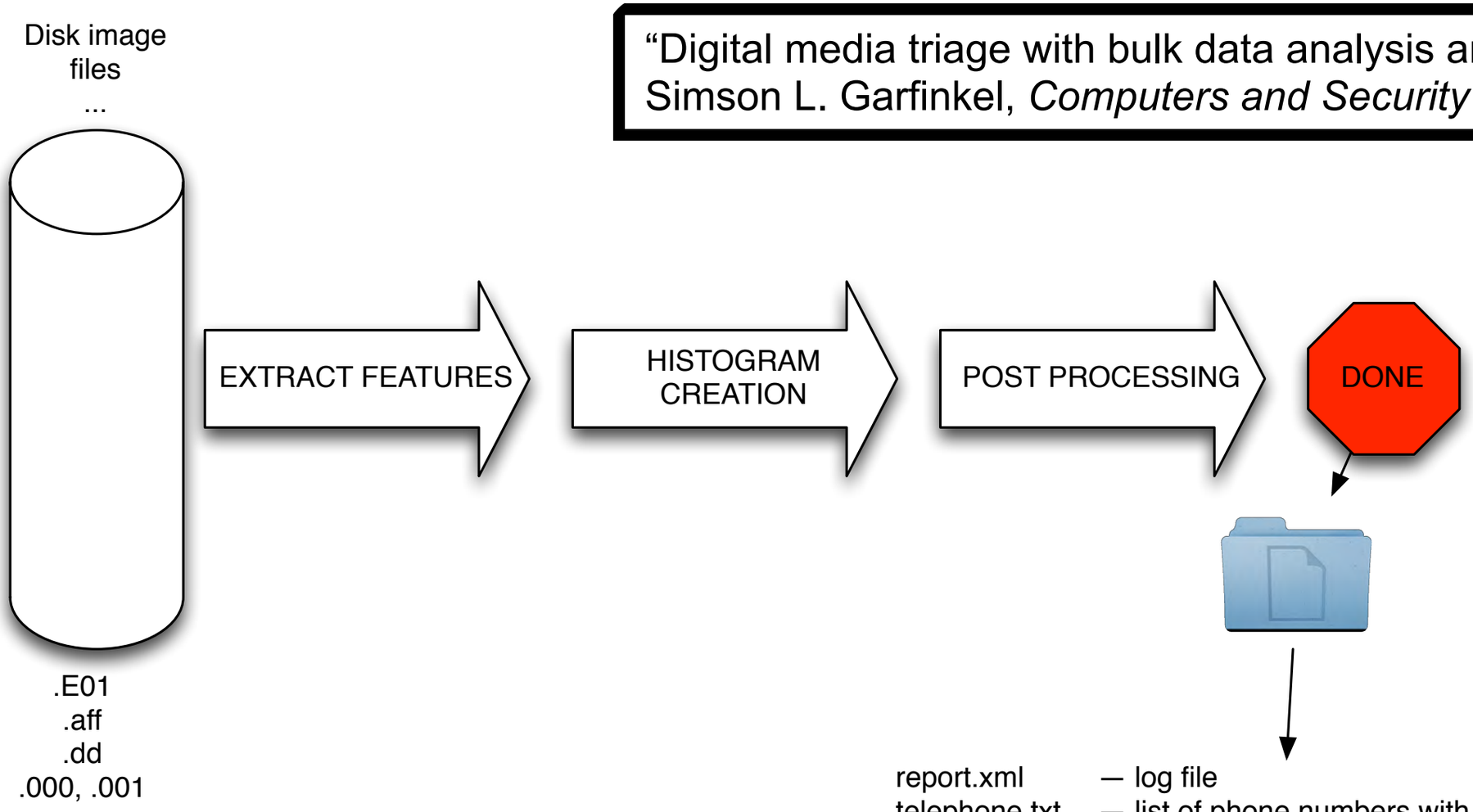

Uses “optimistic” decoding to check every block/byte for:

- Compression — RAR, ZIP, GZIP
- Encoding — BASE64 / BASE85
- Text extraction from PDF fragments (understands PDF compression & letter drawing commands)
- XOR-obfuscation

—*Digital media triage with bulk data analysis and bulk_extractor, Computers & Security 32 (2013)*

# bulk_extractor is our stream forensics program.
# BE finds and extracts “features” from bulk data.

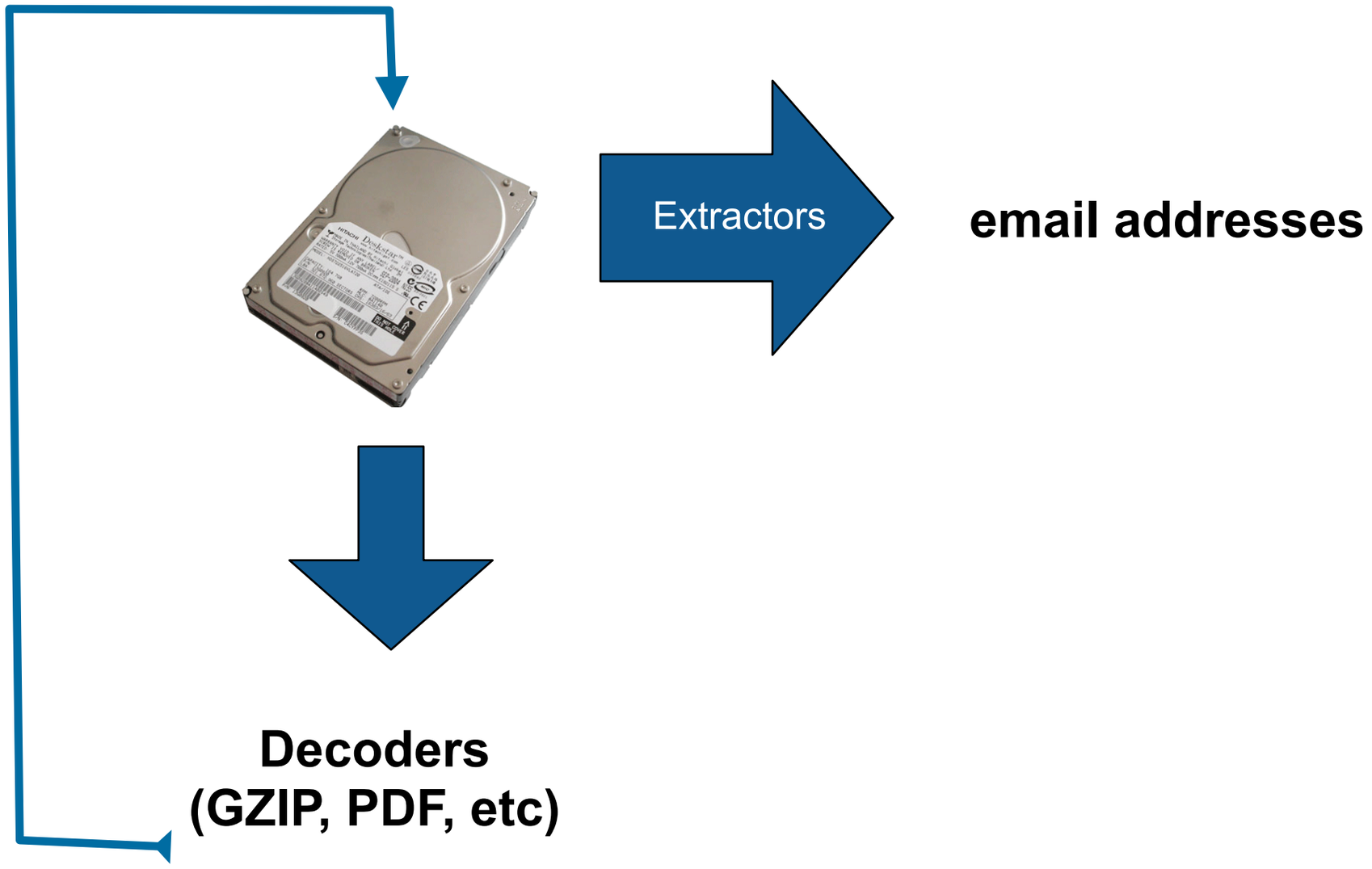
“Digital media triage with bulk data analysis and bulk_extractor,” Simson L. Garfinkel, *Computers and Security 32 (2013) 56-72*




## Output is a *directory* containing:

- feature files; histograms; carved objects
- Mostly in UTF-8; some XML
- Can be bundled into a ZIP file and process with bulk_extractor_reader.py

# bulk_extractor implements optimistic decoding.



# “Optimistic decoding” finds encoded email addresses by attempting to decode every byte with every algorithm.

Input sector:

```
e327 962d 6450 3d91 c945 3bed 97a6 a4cd  .'.-dP=..E;.....
1f8b 0800 0000 0000 0203 8b88 8c72 48ce  .............rH.
cf2d 48cc abd4 03d2 0a8e 4ece 287c 1757  .-H.......N.(|.W
3714 3e00 b455 c1c5 3000 0000 0000 0000  7.>..U..0.......
0a8e 4ece 287c 1757 3714 3e00 a175 10ed  ..N.(|.W7.>..u..
```

Optimistic decoding in theory:

```
try_decompress(buf[0:])
try_decompress(buf[1:])
try_decompress(buf[2:])
...
```

Optimistic decoding is computationally expensive

- It uses all the cores on a 64-core machine!
- Is it worthwhile?

# We used the “Real Data Corpus” to understand the potential of optimistic decoding in real investigations.

## The Real Data Corpus (70TB)

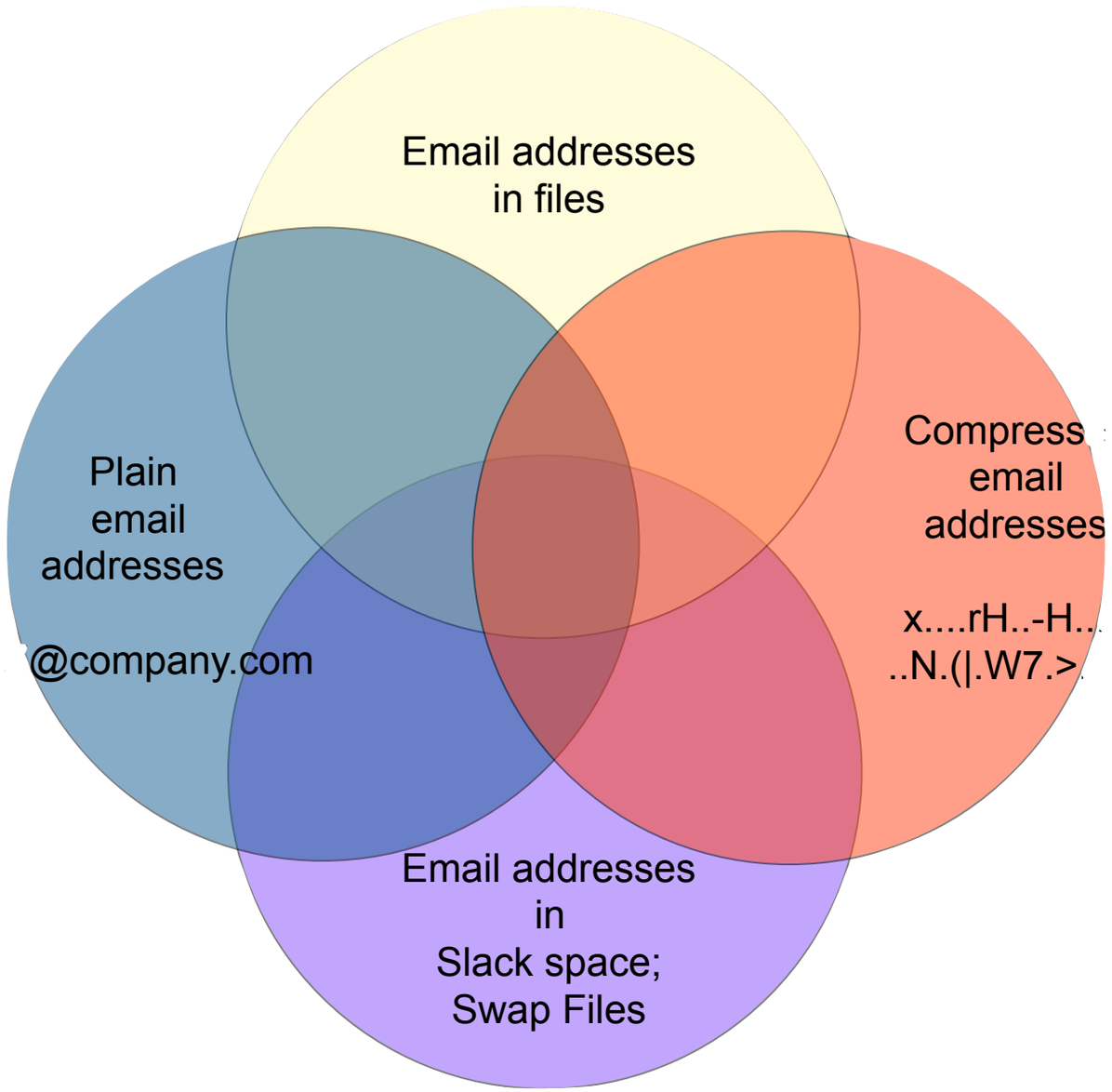
- Disks, camera cards, & cell phones purchased on the “secondary market” (used).
- Most contain data from previous users.
- Mostly acquire outside the US:
  - —*Canada, China, England, Germany, France, India, Israel, Japan, Pakistan, Palestine, etc.*
- Thousands of devices (HDs, CDs, DVDs, flash, etc.)

The problems we encounter obtaining, curating and exploiting this data mirror those of national organizations

—*Garfinkel, Farrell, Roussev and Dinolt, Bringing Science to Digital Forensics with Standardized Forensic Corpora, DFRWS 2009*
*http://digitalcorpora.org/*

# Survey of Encoded Non-File email addresses.

We searched 1646 used storage devices for email addresses that could only be recovered with *bulk data analysis* and *optimistic decoding*.



# Email addresses can be in files

## Files

- Documents
- Address book
- Email messages

**ABC@company.com**
**DEF@company.com**

## Browser Cache:

- Web mail
- Facebook Data

# Email addresses can be in non-file disk sectors

**ABC@company.com**
**DEF@company.com**

**XYZ@company.com**

**Swap Files**
**Hibernation Files**
**File fragments**

# The same email address may be *both* places.
# (A file that's read into RAM before the system hibernates.)

**ABC@company.com**
**DEF@company.com**

**Alice@net.net**

**Alice@net.net**

**XYZ@company.com**

**Swap Files**
**Hibernation Files**
**File fragments**

# This diagram represents email addresses on media.

**“file space”**

**ABC@company.com**
**DEF@company.com**

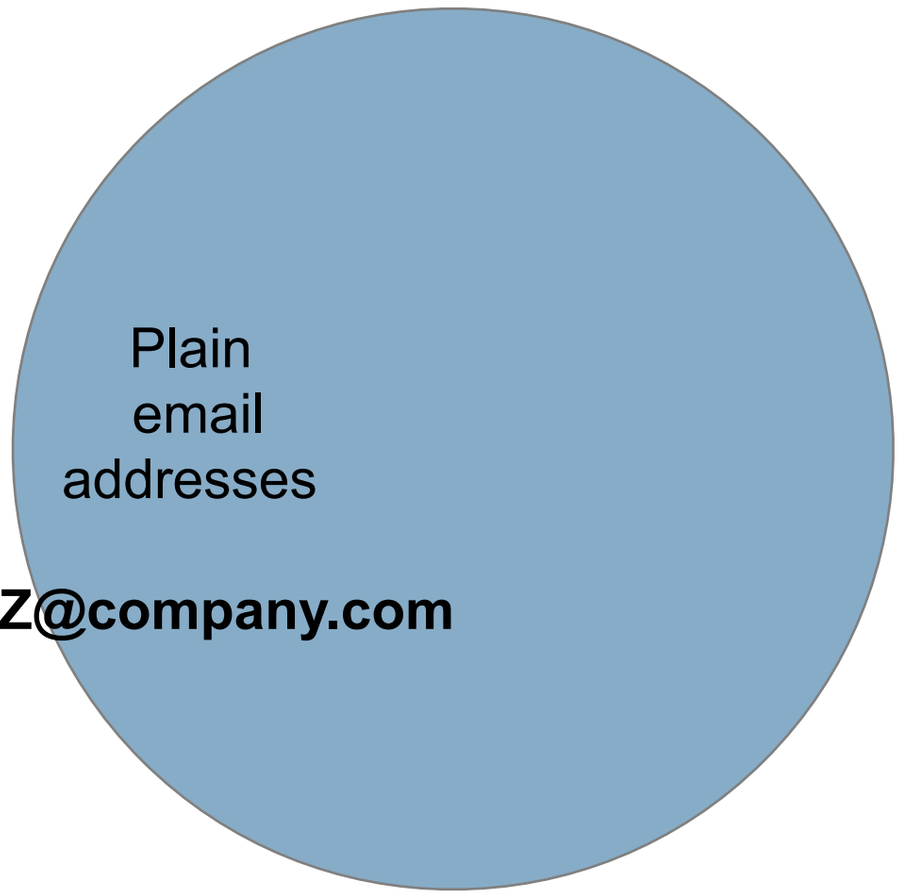
**Alice@net.net**

**XYZ@company.com**

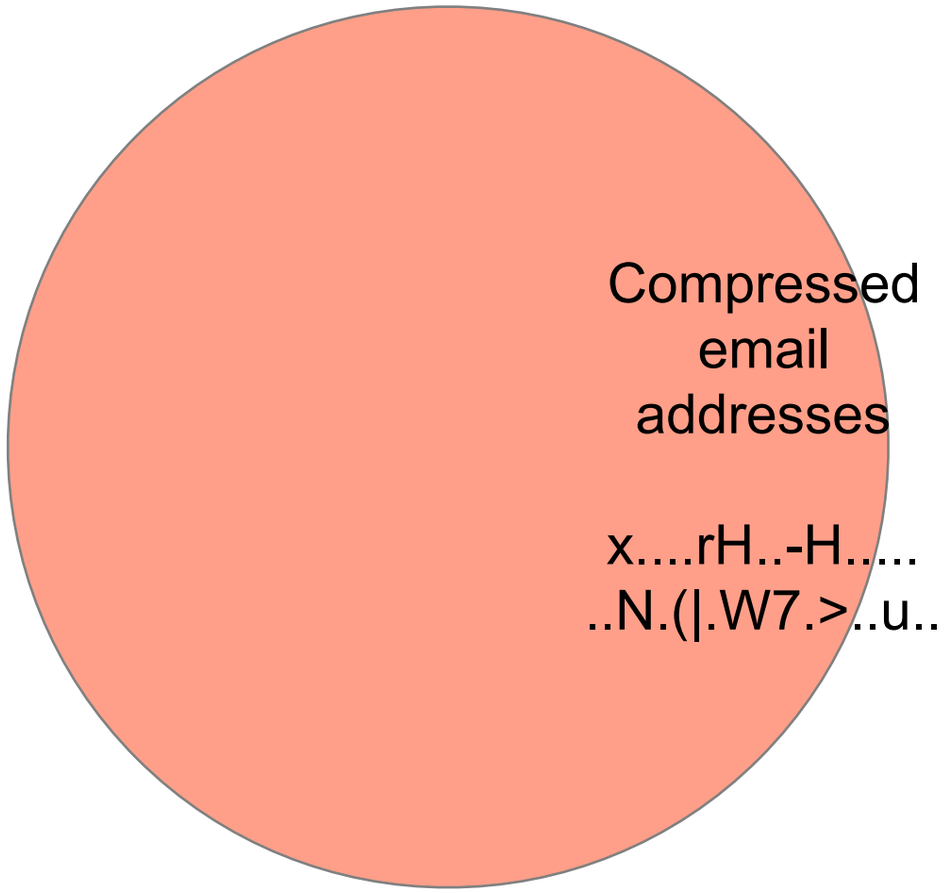
**“non-file space”**

**Swap Files**
**Hibernation Files**
**File fragments**

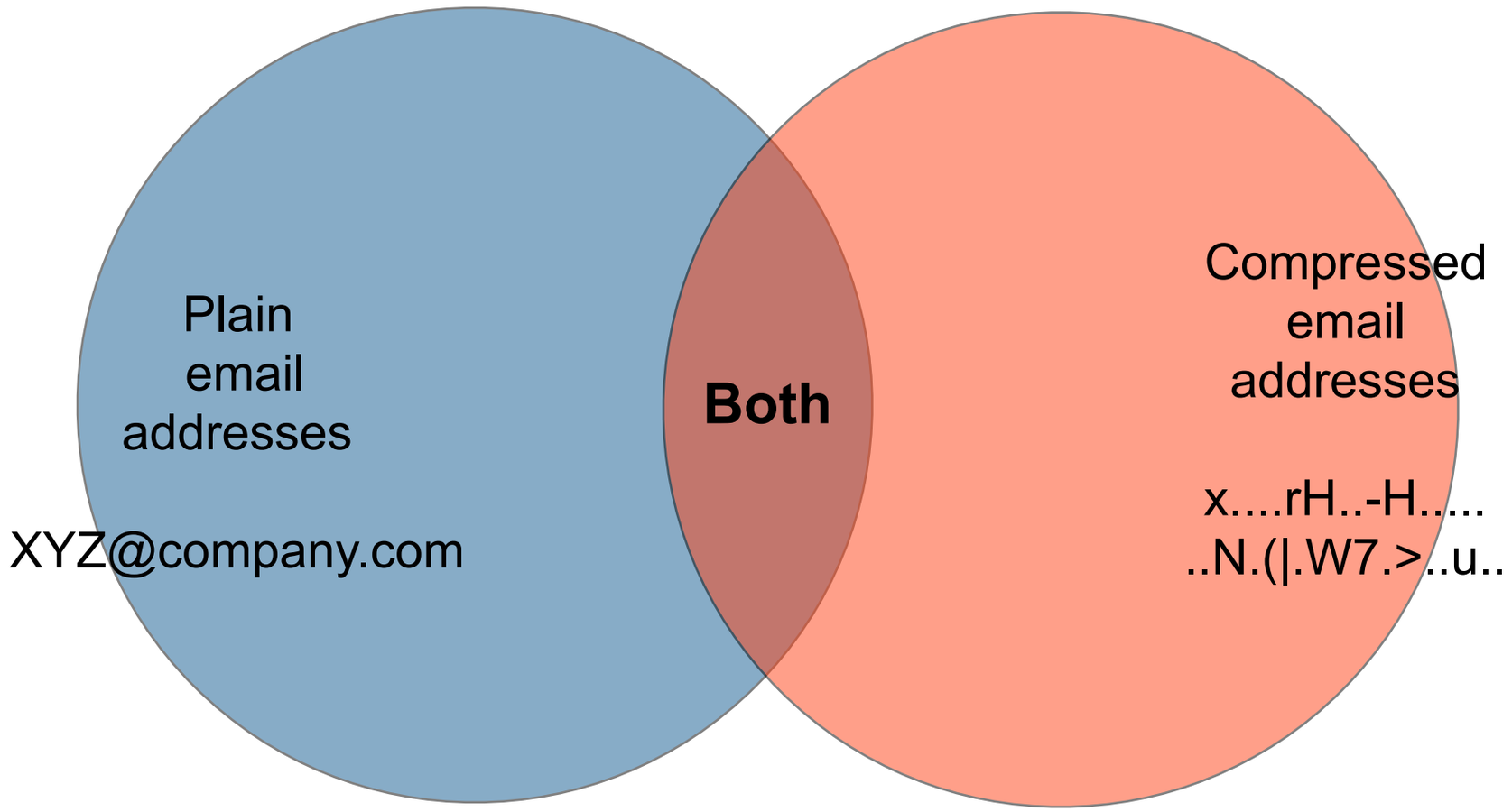
# Email addresses can be plain text.
# “XYZ@company.com”



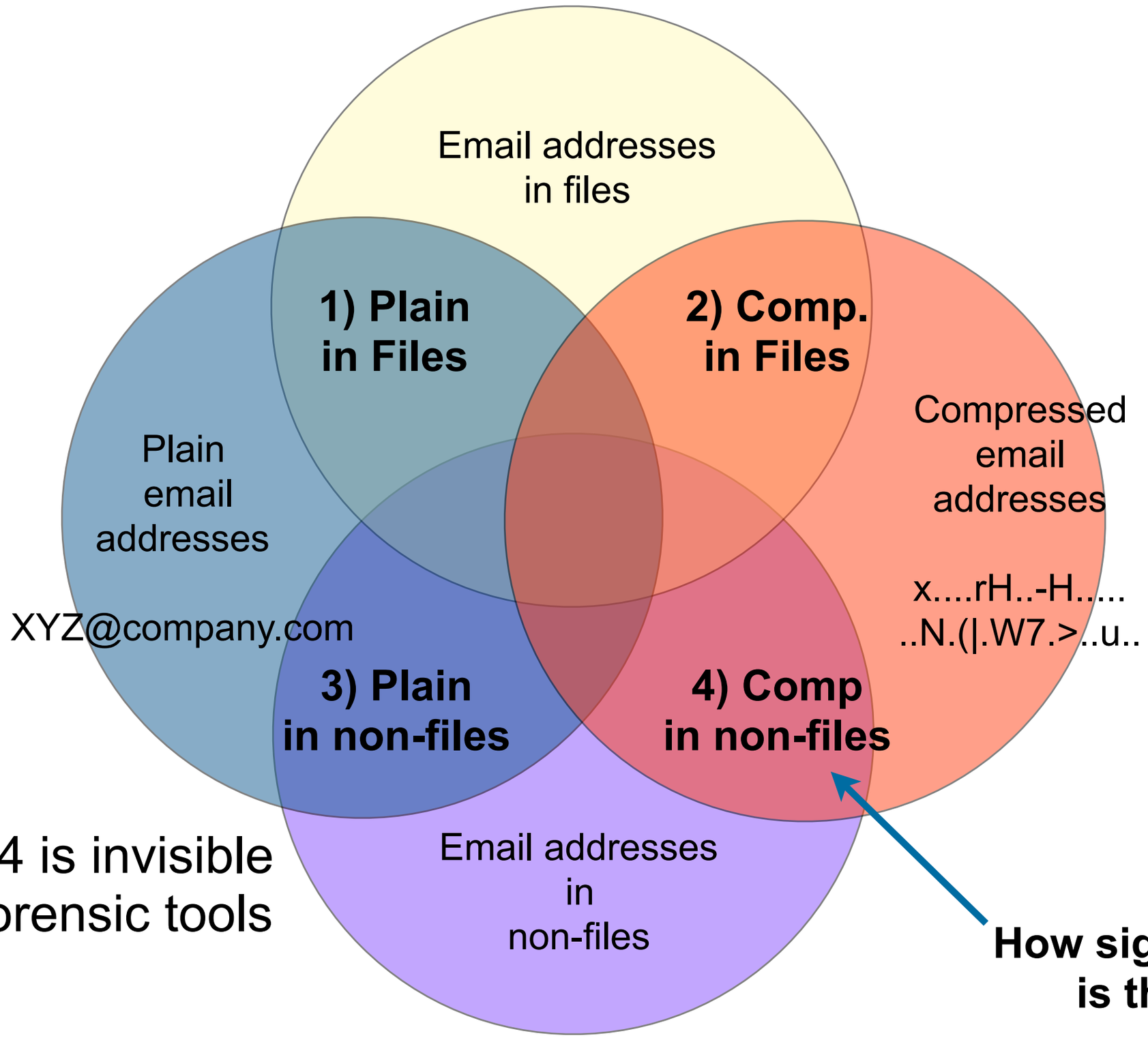
# Email addresses can be compressed or encoded. “x....rH..-H.....N.(|.W7.>..u..”



# Each address can be present *plain*, *compressed*, or both.



# There are four different conditions for an email address on the media.




Condition #4 is invisible to today's forensic tools

**How significant is this?**

# We processed the real data corpus with bulk_extractor

1646 used storage devices

Extractors

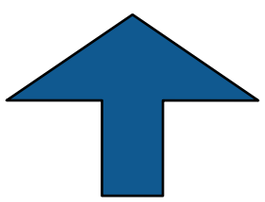
emails
encodings
maps

1646 sets of:

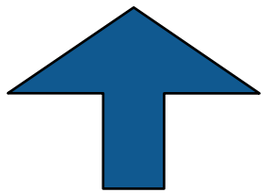
- extracted email addresses
- encoding of each email
- Map of files on device

# “Feature files” contain the extracted email addresses.

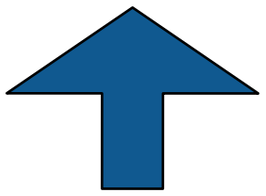
```
# UTF-8 Byte Order Marker; see http://unicode.org/faq/utf_bom.html
#
@
...
392175418    WindowsXP@gn.microsoft.com   Name=WindowsXP@gn.microsoft.com\015\012
...
3772517888-GZIP-28322  user@company.com  onterey-<nobr>user@company.com</nobr>
...
```



**Offset**



**Feature**



**Context**

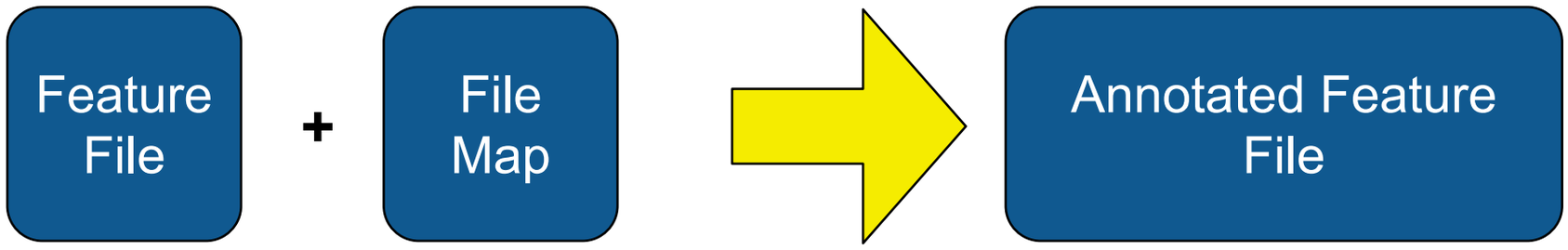
Plain text features have numeric offsets:

```
392175418
```

Compressed features will indicate the algorithm:

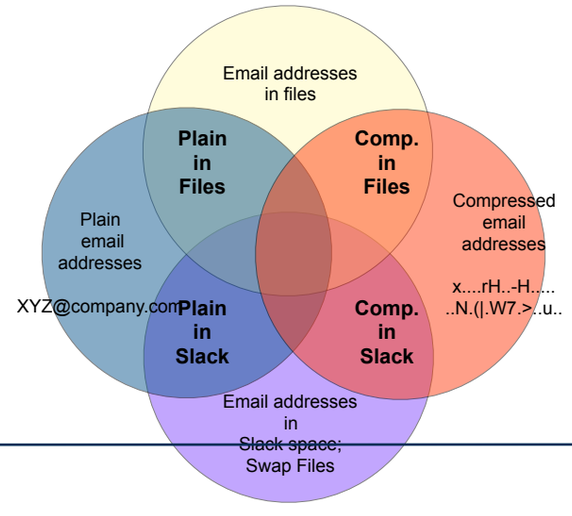
```
3772517888-GZIP-28322
```

# Post-processing with identify_files.py reveals file names

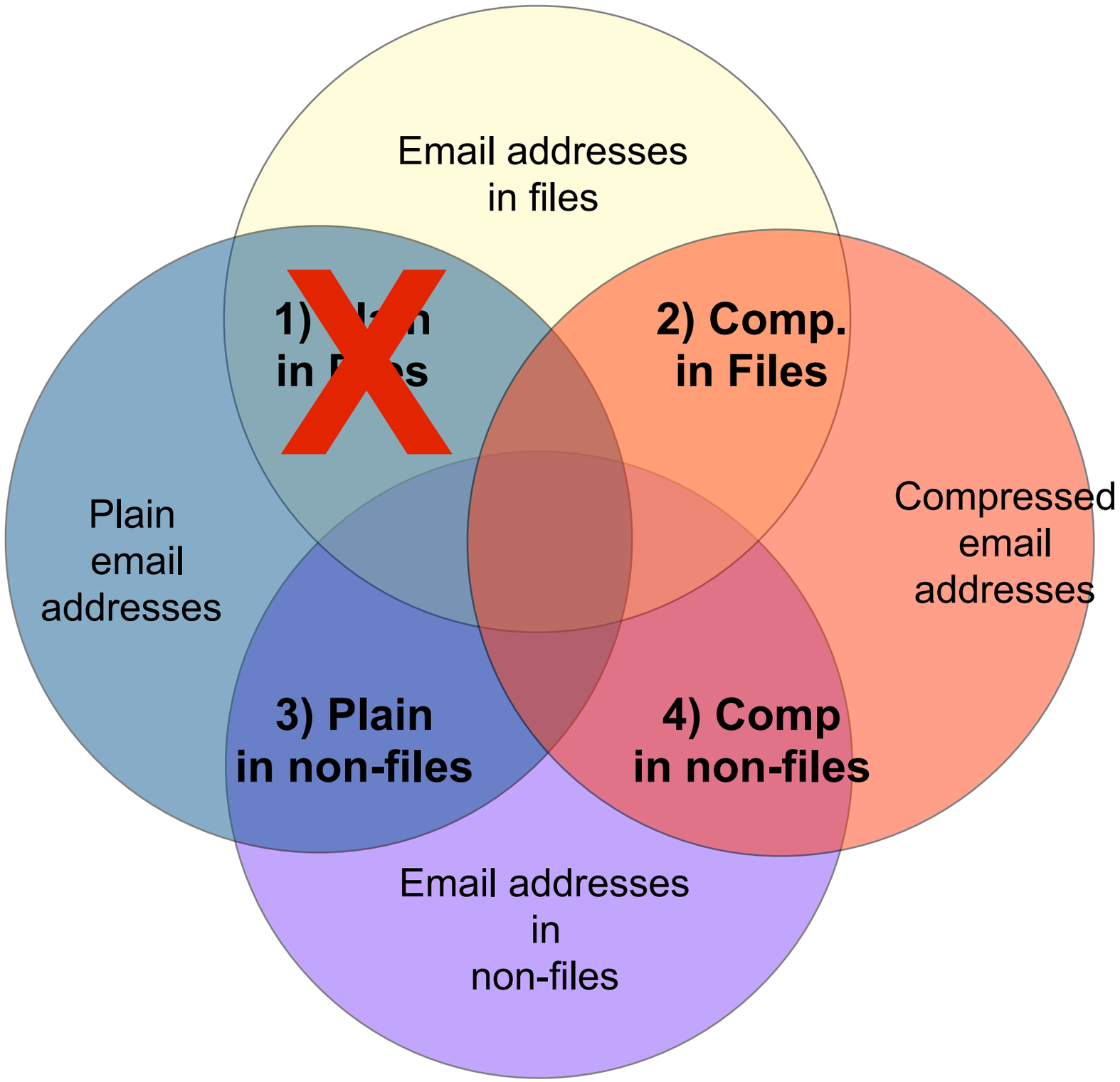



```
Offset:   392175418
Feature: WindowsXP@gn.microsoft.com
Context: \012[User]\015\012Name=WindowsXP@gn.microsoft.com
         \015\012Password=B@ji0
Filename:WINDOWS/system32/oobe/migx25a.dun
MD5:     2b00042f7481c7b056c4b410d28f33cf
```

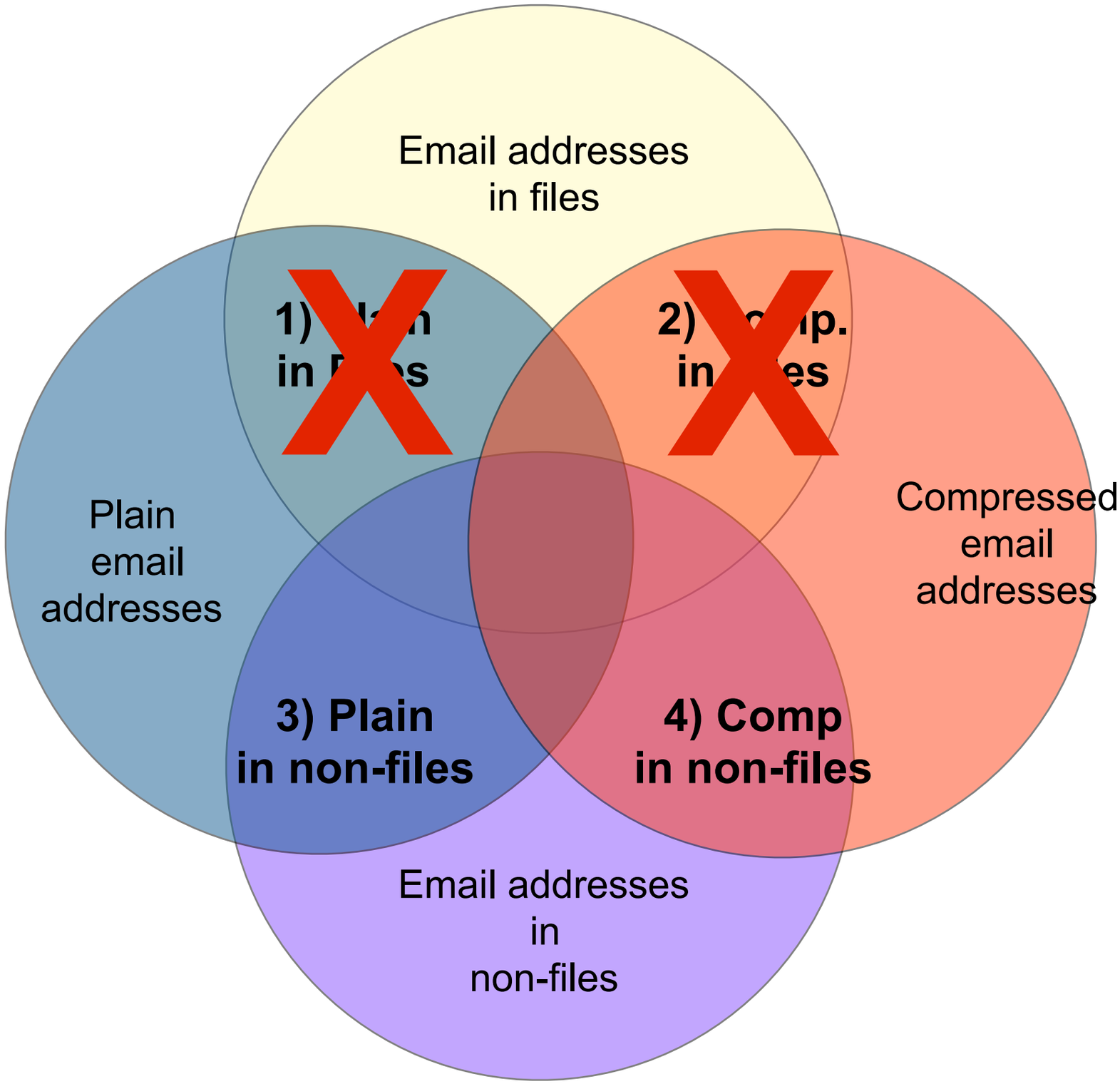
For each feature, we can determine if category #1, #2, #3 and #4!



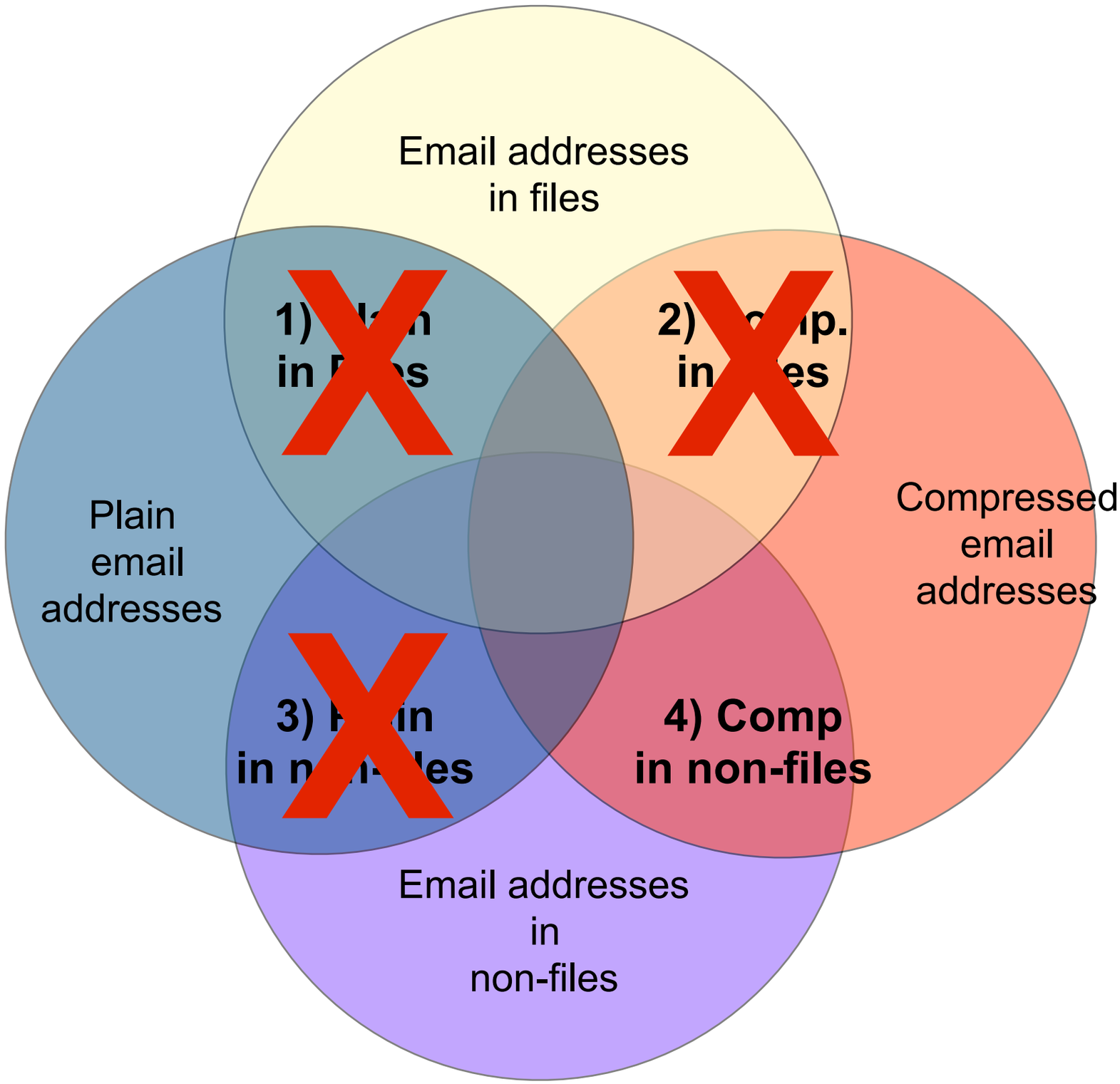
# For each drive:
# We removed every “plain” email address in an allocated file.



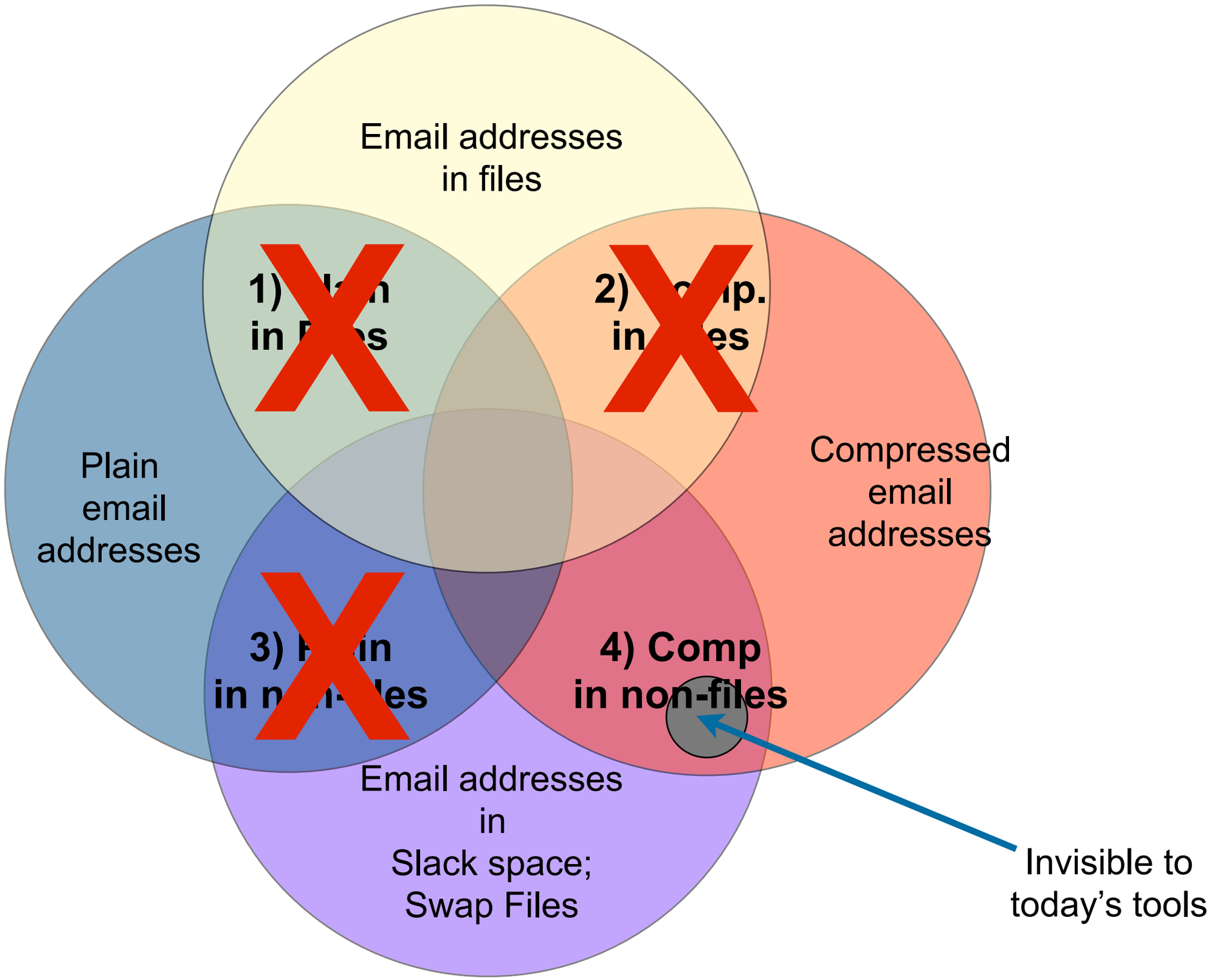
# ...Remove email addresses compressed and in files....



# ...Remove email addresses that are not compressed.



# ...those that remain are compressed and in non-file space.



# Email addresses were present with many different encodings

Example email addresses (sanitized)

| Encoding | Email Address (▪Sanitized) | Note |
|---|---|---|
| GZIP | ▪▪▪▪@▪▪▪▪▪.dk | PII |
| ZIP | ▪▪▪▪▪▪@desktopsidebar.com | PII |
| HIBER | ntIV@std.do | false positive |
| ZIP | ▪▪▪▪▪▪▪▪▪▪▪▪▪▪@digital.com | source code? |
| ZIP | pcg@goof.com | ECGS Compiler |
| ZIP | andrew@northwindtraders.com | MS Office Sample |
| ZIP | ActiveSh@eet.Na | false positive |
| GZIP | linux-ntfs-dev@lists.sourceforge.net | mailing list |

Questions:

- How common are compressed email addresses in unallocated space?
- Is this technique worth the effort?

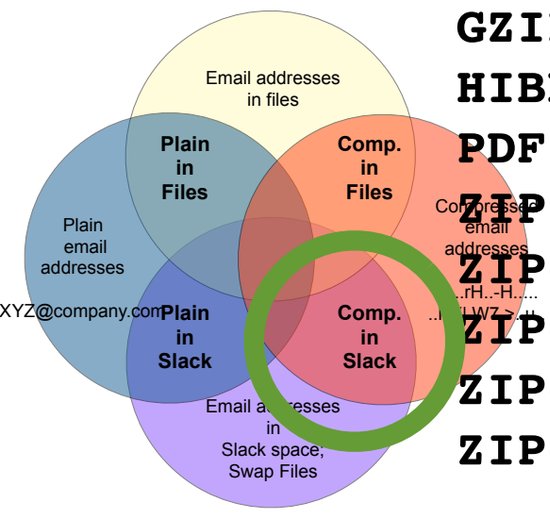
# Example: Drive IN10-0138

| Emails seen | count | 1) Plain in Files | 2) Comp. in Files | 3) Plain in non-files | 4) Comp in non-files |
|---|---|---|---|---|---|
| Cleartext | | 358 | -- | 5341 | -- |
| All Comp | | -- | 9 | -- | 135 |
| GZIP | 50 | | 14 | | 36 |
| HIBER | 39 | | 7 | | 32 |
| HIBER-GZIP | 23 | | | | 23 |
| PDF | 88 | | 1 | | 87 |
| ZIP | 28 | | 7 | | 21 |
| ZIP-PDF | 18 | | | | 18 |

135 out of 5700 email addresses are invisible to existing tools.

# 1,646 RDCdisk images that had intact file systems. Many email addresses existed only in Encoded NonFile.

| Coding | Drives | Emails | avg | max | σ |
|---|---|---|---|---|---|
| 1) Plain in files | 739 | 81,920 | 110 | 4,206 | 253 |
| 2) Comp in files | 355 | 19,711 | 55 | 5,454 | 388 |
| 3) Plain in non-files | 860 | 1,956,059 | 2,274 | 178,073 | 9,248 |
| 4) Comp in non-files | 474 | 165,481 | 349 | 59,376 | 2,889 |
| BASE64 Comp | 54 | 219 | 4 | 50 | 7 |
| BASE64-GZIP Comp | 2 | 64 | 32 | 37 | 5 |
| GZIP Comp | 234 | 66,195 | 282 | 9,103 | 981 |
| GZIP-BASE64 Comp | 7 | 44 | 6 | 11 | 3 |
| GZIP-GZIP Comp | 15 | 12,663 | 844 | 11,845 | 2,944 |
| GZIP-GZIP-BASE64 Comp | 2 | 38 | 19 | 30 | 11 |
| GZIP-GZIP-GZIP Comp | 4 | 58 | 14 | 38 | 14 |
| GZIP-GZIP-ZIP Comp | 1 | 12 | 12 | 12 | 0 |
| GZIP-PDF Comp | 5 | 38 | 7 | 30 | 11 |
| GZIP-ZIP Comp | 6 | 49 | 8 | 30 | 9 |
| HIBER Comp | 79 | 1,433 | 18 | 217 | 44 |
| PDF Comp | 162 | 2,352 | 14 | 238 | 31 |
| ZIP Comp | 388 | 85,252 | 219 | 59,369 | 3,025 |
| ZIP-BASE64 Comp | 5 | 30 | 6 | 13 | 5 |
| ZIP-BASE64-GZIP Comp | 2 | 65 | 32 | 38 | 5 |
| ZIP-GZIP Comp | 14 | 261 | 18 | 132 | 34 |
| ZIP-PDF Comp | 26 | 115 | 4 | 18 | 4 |




Some drives had more than 10,000 compressed email addrs.

# The encoding type reveals the email address source.

HIBER-GZIP were downloaded by HTTP and saved in Hibernation

Example:

```
...
…6464-HIBER-49691-GZIP-1526   groups-noreply@linkedin.com    3d\134"groups-noreply@linkedin.com
…6464-HIBER-49691-GZIP-2018   m▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪@gmail.com  3d\134"m▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪▪@gmail.co
…6464-HIBER-49691-GZIP-2128   sur▪▪▪▪▪▪▪1@gmail.com          3d\134"sur▪▪▪▪▪▪▪1@gmail.com\134"\
…6464-HIBER-49691-GZIP-2625   ▪▪▪▪▪▪▪.consultancy@gmail.com 3d\134"▪▪▪▪▪▪▪.consultancy@gmail.c
…6464-HIBER-49691-GZIP-2736   sur▪▪▪▪▪▪▪1@gmail.com          3d\134"sur▪▪▪▪▪▪▪1@gmail.com\134"\
…6464-HIBER-49691-GZIP-3186   san▪▪▪▪@▪▪▪▪▪▪▪▪▪▪.com        \134" "san▪▪▪▪@▪▪▪▪▪▪▪▪▪▪.com\134"\134u
…6464-HIBER-49691-GZIP-3685   Careers@▪▪▪▪▪▪bank.com         3d\134"Careers@▪▪▪▪▪▪bank.com\134"
…6464-HIBER-49691-GZIP-4124   par▪▪▪▪@team▪▪▪▪▪▪.com         3d\134"par▪▪▪▪@team▪▪▪▪▪▪.com\134"
…6464-HIBER-49691-GZIP-4149   u003epar▪▪▪▪@team▪▪▪▪▪▪.com   \134u003epar▪▪▪▪@team▪▪▪▪▪▪.com\13
…6464-HIBER-49691-GZIP-4607   d▪▪▪▪.▪▪▪▪▪@gmail.com         3d\134"d▪▪▪▪.▪▪▪▪▪@gmail.com\134"\
…6464-HIBER-49691-GZIP-4631   u003ed▪▪▪▪.▪▪▪▪▪@gmail.com    \134u003ed▪▪▪▪.▪▪▪▪▪@gmail.com\134
…6464-HIBER-49691-GZIP-5114   raj▪▪▪▪▪▪@bsnl.in             3d\134"raj▪▪▪▪▪▪@bsnl.in\134"\134u
…6464-HIBER-49691-GZIP-5558   kiran.▪▪▪@▪▪▪▪technology.com  3d\134"kiran.▪▪▪@▪▪▪▪technology.co
…6464-HIBER-49691-GZIP-5671   sur▪▪▪▪▪▪▪1@gmail.com          3d\134"sur▪▪▪▪▪▪▪1@gmail.com\134"\
...
```

- JSON object downloaded from Facebook by compressed HTTP
- In RAM, written to HIBER on disk when the system went into sleep.

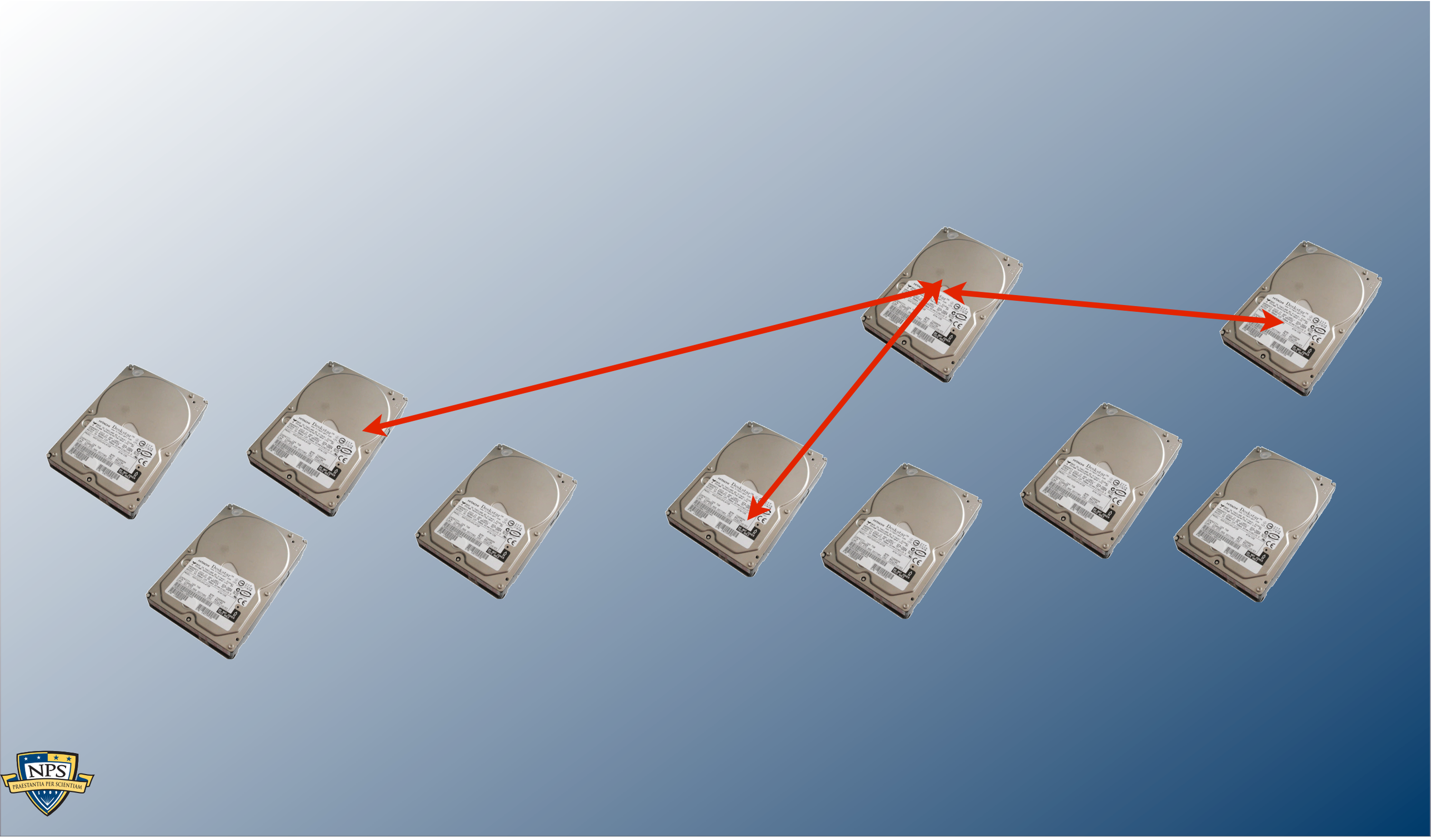
# Manual examination reveals the provenance of each type.

| Encoding | Source | Relevant? |
|---|---|---|
| BASE64 | Email messages sent as attachments | YES |
| BASE64-GZIP | Source code sent as attachments | NO |
| GZIP | HTML & JSON (Browser cache) | YES |
| HIBER | Program memory | YES |
| ZIP | Software Distributions | NO |
| ZIP-PDF | Archives of PDFs | YES |
| ZIP-ZIP-ZIP | WinZip Archives | NO |

## Conclusion:

- Optimistic decoding uncovers important information that is otherwise missed.

"The Prevalence of Encoded Digital Trace Evidence in Nonfile space of computer media," *Journal of Forensic Sciences*, to appear 2014



# Where do we go from here?
# A DF research agenda

# What's the DF research agenda?

## Engineering — embrace diversity & scale.

- Support the increasing number of data formats and encodings (master's projects)
- Support for new kinds of devices (SCADA, cars, etc.)
- Develop fault-tolerant high-performance architecture for data analysis
- Migration from desktop analysis to "cloud" analysis

  —*Analysis with Hadoop or Google App Engine*

## Science — better techniques.

- Approaches for finding data *relevant to the case at hand.*
- Approaches that work with *encrypted data* (storage, network, memory)
- Approaches for cross-case correlation. (Privacy-preserving anonymous matching.)

  —*Applications to law enforcement & digital humanities*
- Cloud-based acquisition & analysis

  —*How do we acquire and stabilize information?*
- Integration with *social network analysis*;

## Math and Science:

- Linguistics, Natural Language Processing & Machine Learning

# Please try our tools!

## bulk_extractor, a high-performance stream-based feature extractor

- https://github.com/simsong/bulk_extractor (dev tree)
- http://digitalcorpora.org/downloads/bulk_extractor (downloads)

—*Digital media triage with bulk data analysis and bulk_extractor, Computers & Security 32 (2013),*

## hashdb — high-performance database for sector hashes

- https://github.com/simsong/hashdb

## DFXML — An XML language for doing computer forensics

- provenance, file extraction, hashes and piecewise-hashes, registry values, etc.
- https://github.com/simsong/dfxml

—*Digital forensics XML and the DFXML toolset, Digital Investigation 8 (2012)*

## Data!

- http://digitalcorpora.org/

**Contact Information:**
**Simson L. Garfinkel**
**simsong@acm.org**
**http://simson.net/**