Automated Media Exploitation Research: History and Current Projects

Simson L. Garfinkel
Associate Professor, Naval Postgraduate School

June 3, 2011

http://simson.net/

NPS is the Navy's Research University.

Location: Monterey, CA

Students: 1500

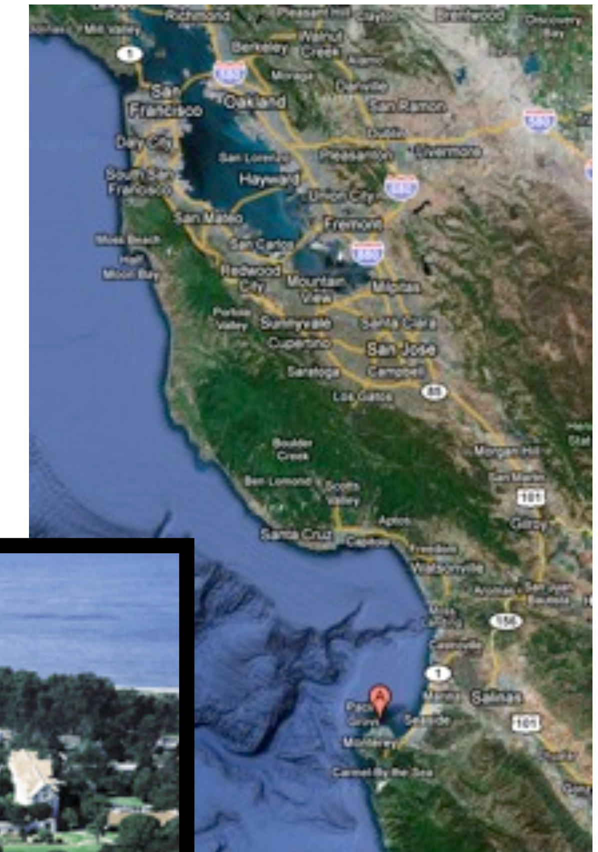
- US Military (All 5 services)
- US Civilian (Scholarship for Service & SMART)
- Foreign Military (30 countries)
- *All students are fully funded*

Schools:

- Business & Public Policy
- Engineering & Applied Sciences
- Operational & Information Sciences
- International Graduate Studies

NCR Initiative:

- 8 offices on 5th floor, 900N Glebe Road, Arlington
- FY12 plans: 4 professors, 2 postdocs
- Recruiting: Government employees for MS & PHDs



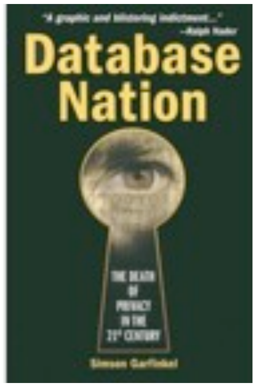
Simson Garfinkel
Associate Professor, Department of Computer Science

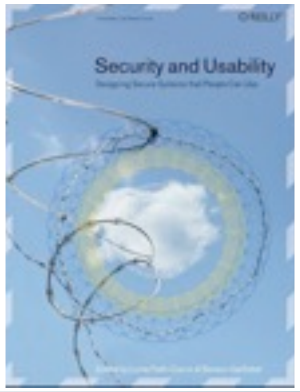
2010	PCS to National Capital Region
2006-	Joined NPS Faculty
2005-2006	Harvard University postdoc
2002-2005	MIT EECS PhD Program
1988-2004	Entrepreneur & Journalist

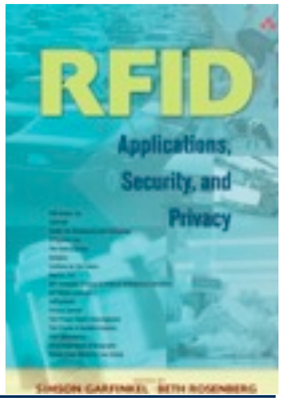
- Vineyard.NET, Broadband2Wireless,
- *Sandstorm Enterprises, Inc. (network forensics)*
- *Technology Review Magazine*
- *Chief Security Officer (CSO) Magazine (4 national awards)*
- *Boston Globe Columnist, 1997-2002*

1988-2011	Author & Inventor

- 14 books
- 6 US patents
- 45 journal articles & conference papers







NPS Team Members

Dr. Simson L. Garfinkel

Dr. Joel Young

Dr. Robert Beverly

Dr. Mathias Kölsch

Dr. Bret Michael

Dr. Neil Rowe

Scott Cote • Adam Russell • Bruce Allen

Major Research in Computer Forensics, 2006-2011

Corpus Creation, Management and Large-Scale Analysis

- For research, tool testing, and tool testing.
- Real Data Corpus — Real data from around the world.
- Realistic Corpus — Manufactured data.
- Cross-Drive Analysis — Datamining organizations

Forensics File Formats, Data Representation and Automation

- AFF — Advanced Forensic Format — Interoperability, Expandability & Encryption
- DFXML — Digital Forensics XML

Carving Research

- Fragment Recovery Carving
- Multiuser Carved Data Ascription Problem
- Bulk_Extractor — Parallelized Carving with Named Entity Extraction
- Hash-based Carving

Forensic Acceleration

- High-speed MD5/SHA/AES implementations.
- Random Sampling of Files and Blocks
- Bloom Filters for Searching

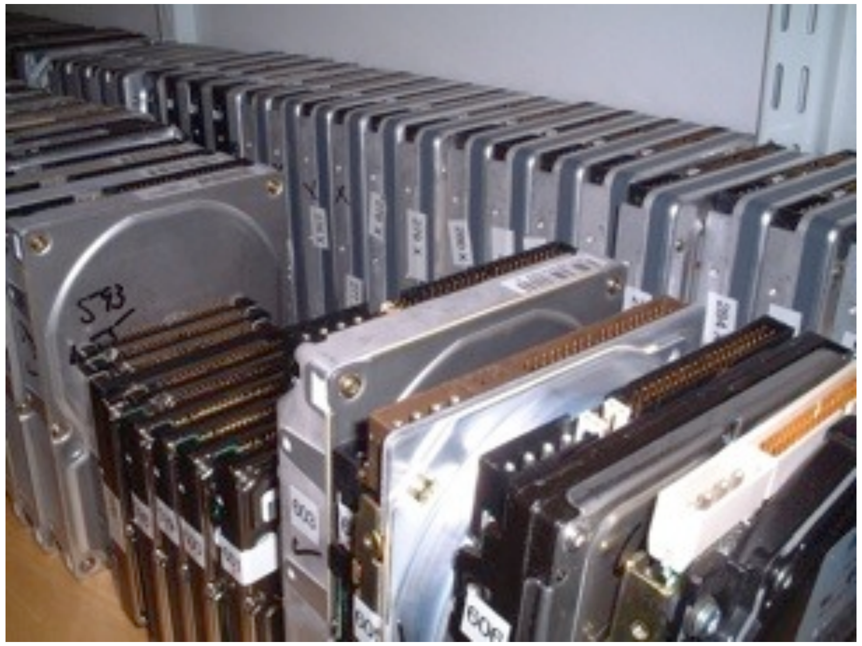
Collaborative Research:

- Profiling and Outlier Analysis
- Similarity Research

Current NPS research thrusts

Area #1: End-to-end automation of forensic processing

- Digital Forensics XML Toolkit
- Tool integration; automated metadata extraction



Area #2: Bringing data mining to forensics

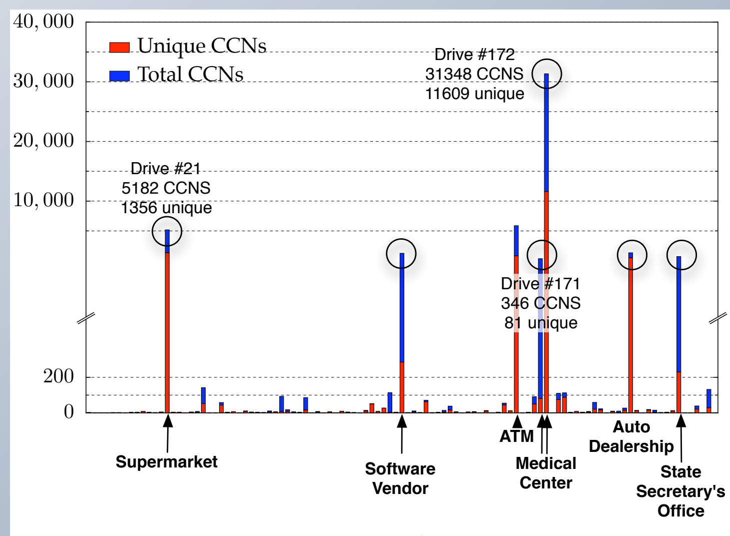
- Automated social network analysis (cross-drive analysis)
- Automated ascription of carved data
- Novel VIDEX and IMINT

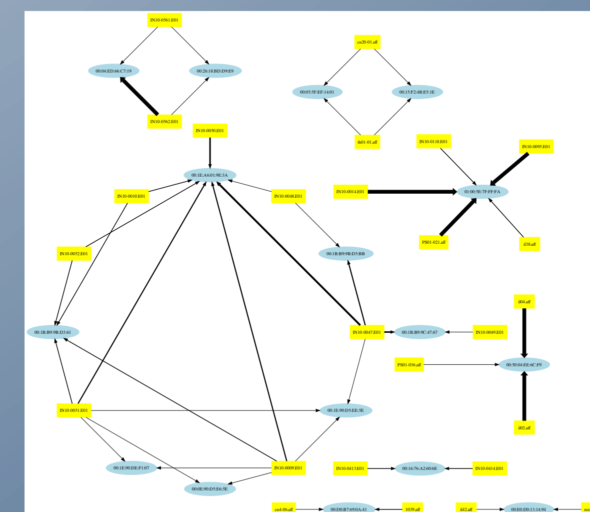
Area #3: Bulk Data Analysis

- Statistical techniques (sub-linear algorithms)
- Similarity Metrics;

Area #4: Creating Standardized Forensic Corpora

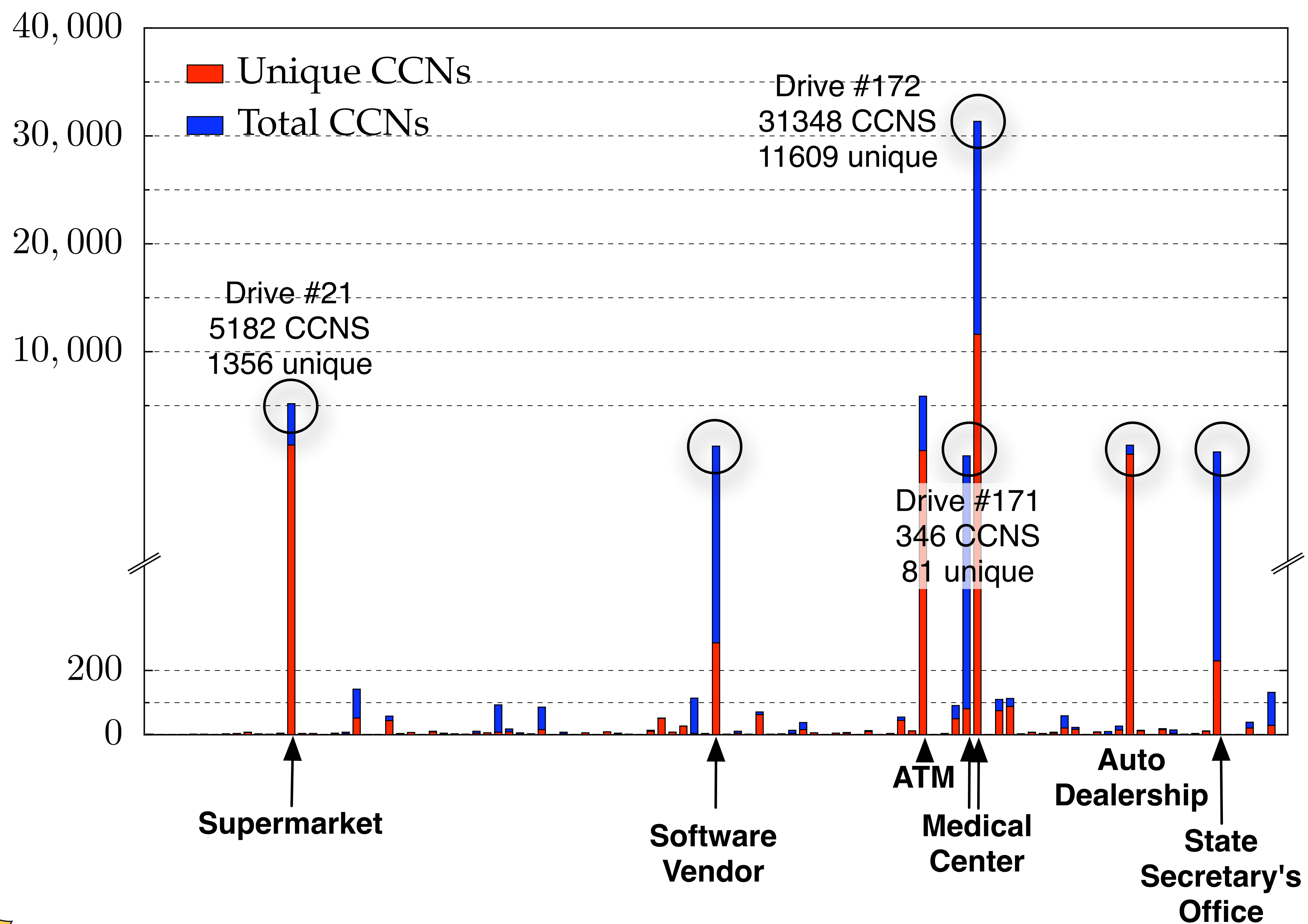
- Freely redistributable disk and memory images, packet dumps, file collections.





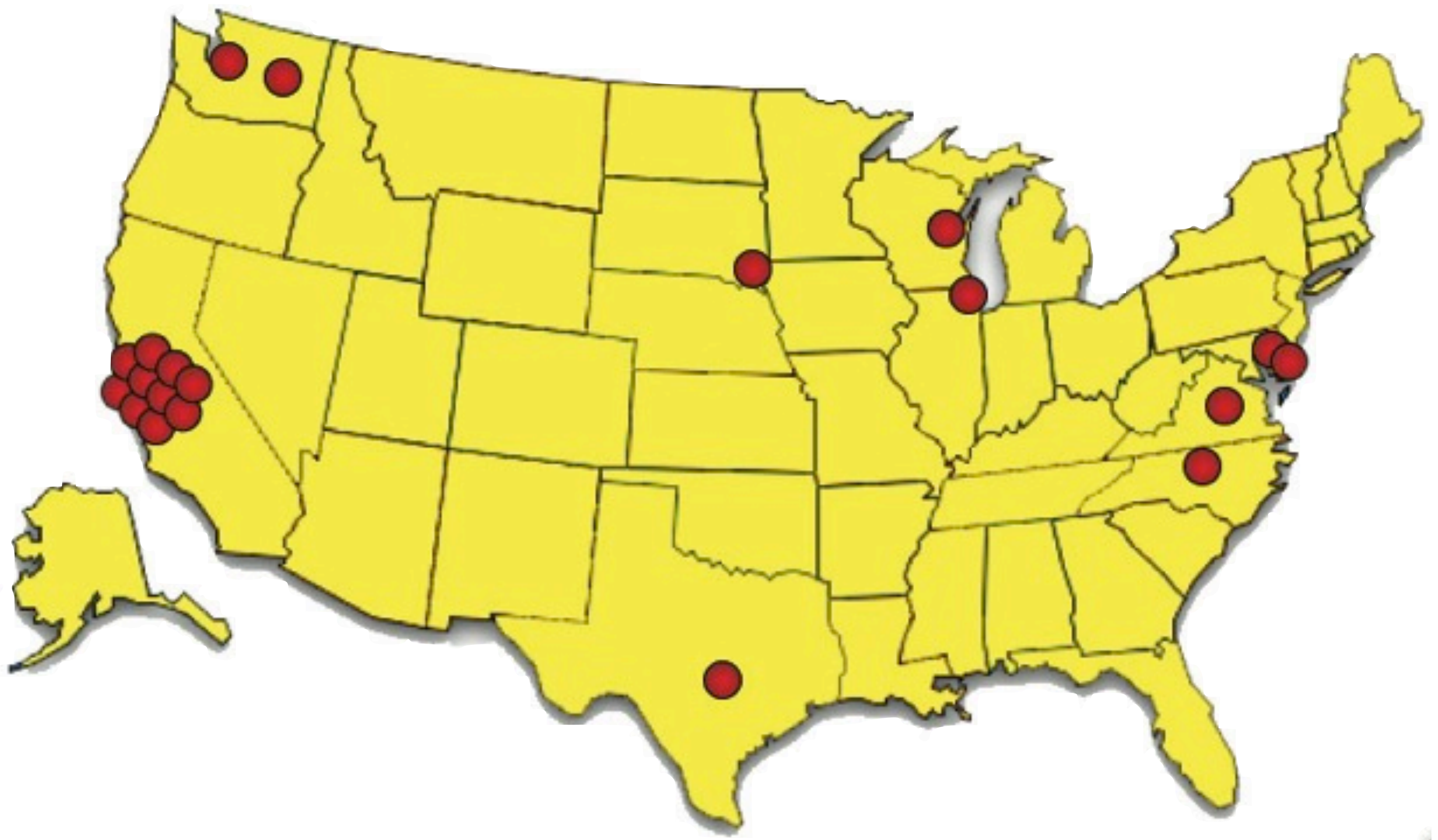
Research Synergy

2005: Triage by CCN Frequency



Manual Geolocation of hard drives.

Design Principles and Patterns for Computers that are Simultaneously Secure and Usable, MIT PhD Thesis, 2005

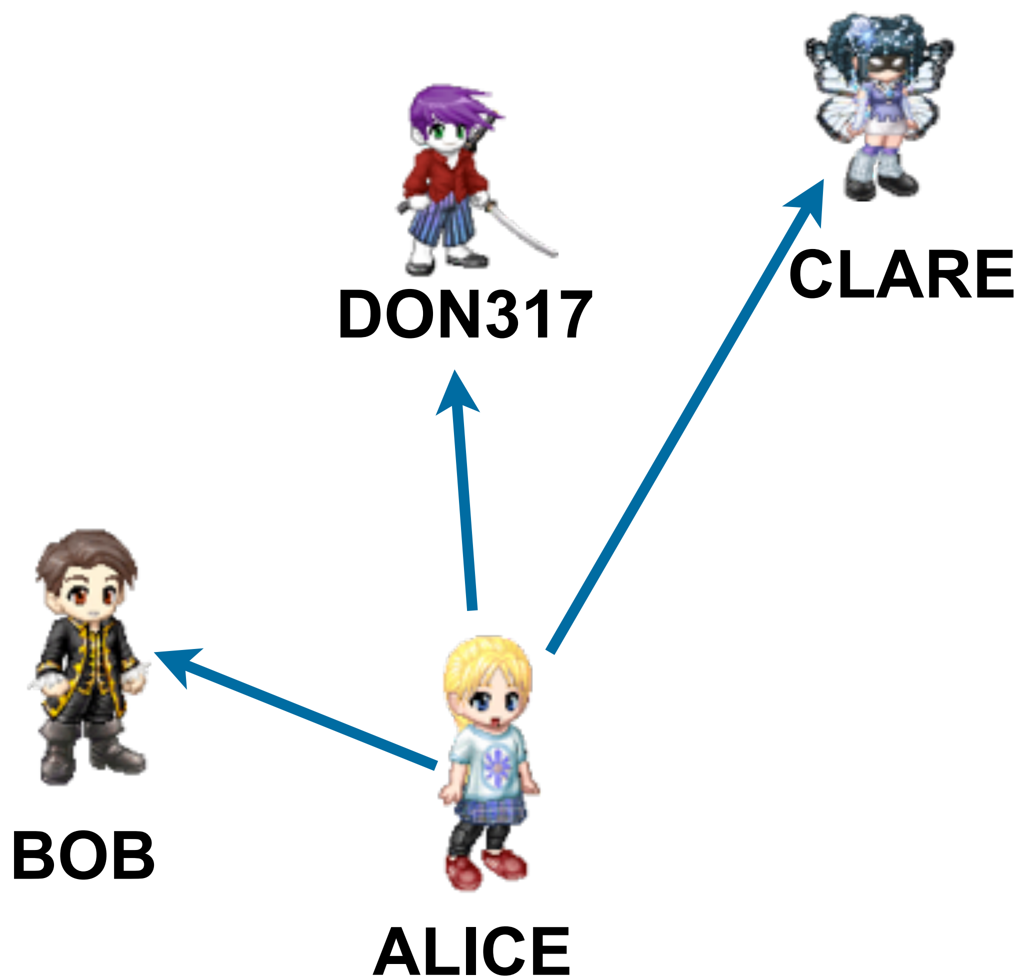


While tracing back the drives,
I discovered *drive attribution* via histogram analysis.

Histograms are powerful

Email histogram allows us to rapidly determine:

- Drive's primary user
- User's organization
- Primary correspondents
- Other email addresses

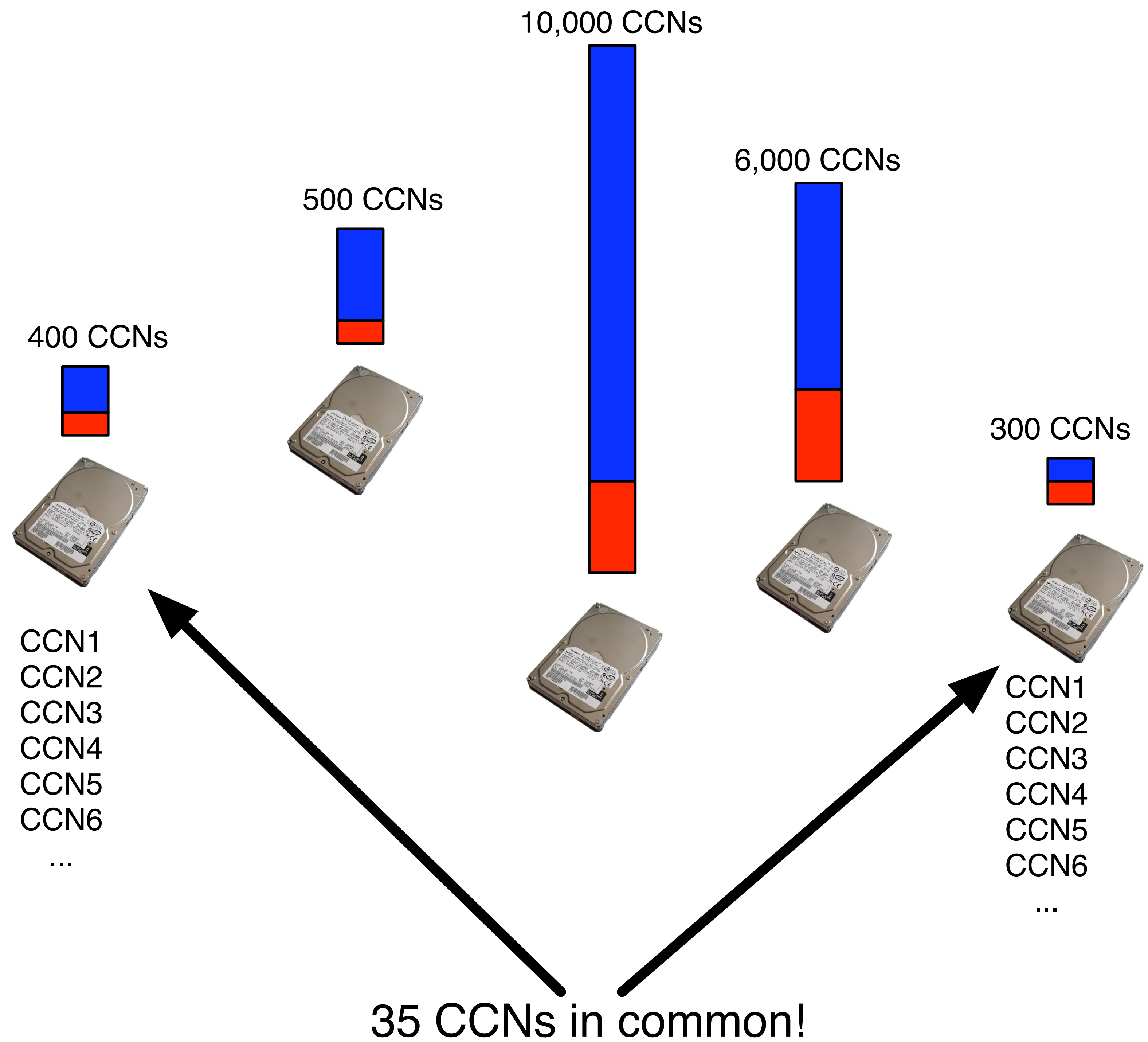


Drive #51 (Anonymized)

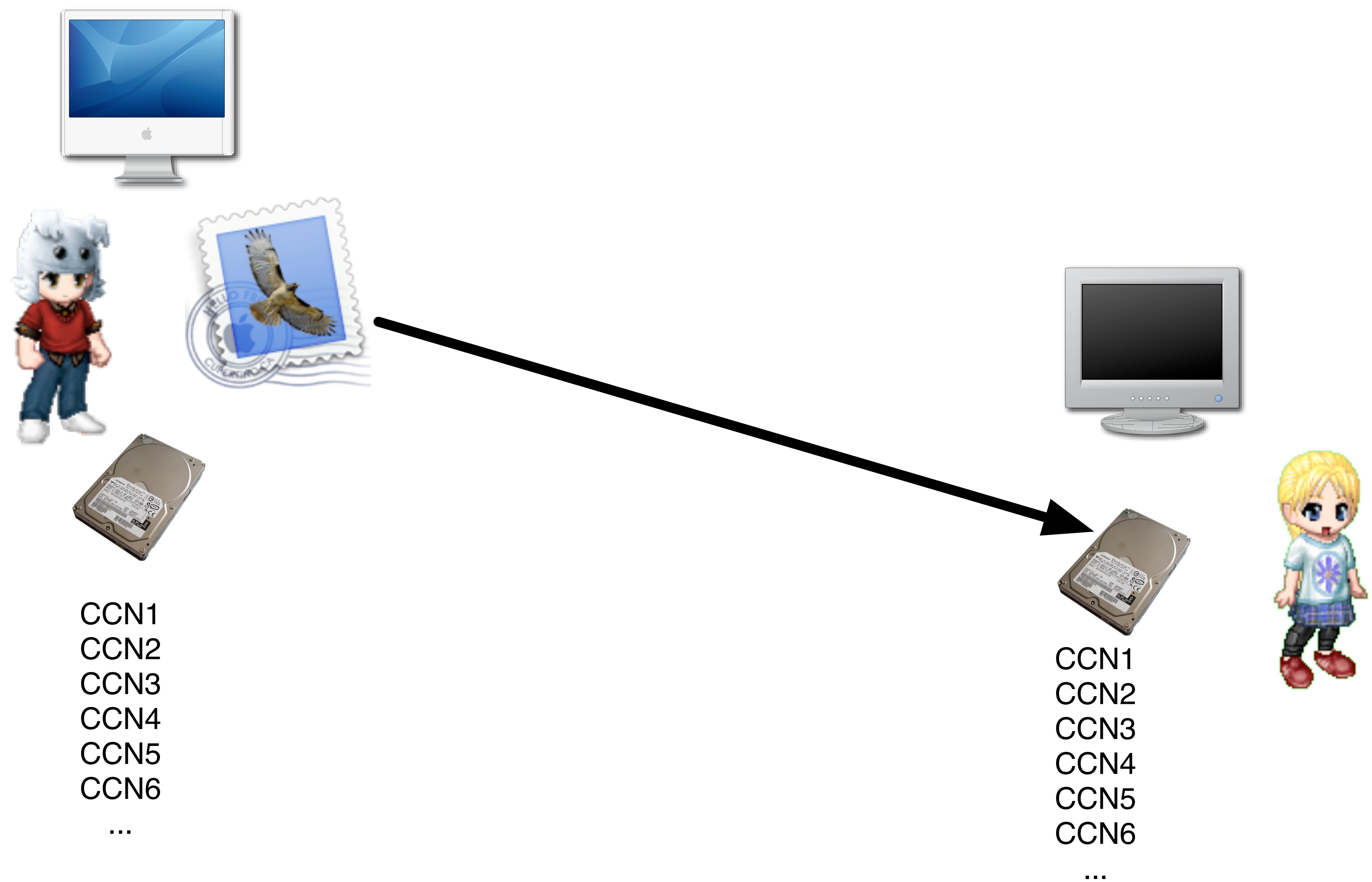
ALICE@DOMAIN1.com	8133
BOB@DOMAIN1.com	3504
ALICE@mail.adhost.com	2956
JobInfo@alumni-gsb.stanford.edu	2108
CLARE@aol.com	1579
DON317@earthlink.net	1206
ERIC@DOMAIN1.com	1118
GABBY10@aol.com	1030
HAROLD@HAROLD.com	989
ISHMAEL@JACK.wolfe.net	960
KIM@prodigy.net	947
ISHMAEL-list@rcia.com	845
JACK@nwlink.com	802
LEN@wolfenet.com	790
natcom-list@rcia.com	763

2005: Cross-drive analysis.

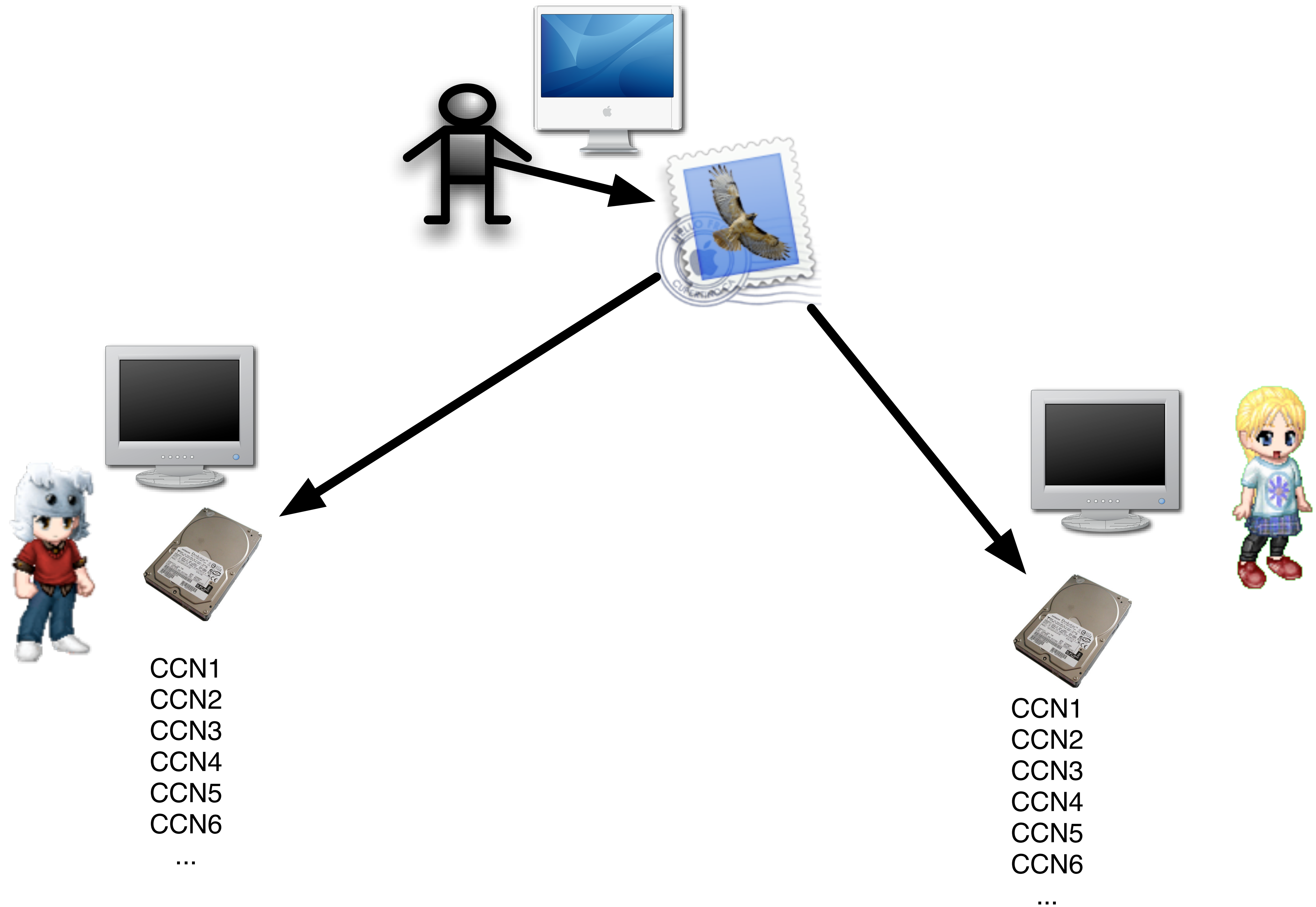
Why would two drives have 35 credit card numbers in common?



Scenario #1:
Owner of one drive sent a message to another drive.



Scenario #2:
Both drives received a message from a third party.



Cross Drive Analysis (CDA) computes the correlation matrix of the distinct ("pseudo-unique") information.

first drive

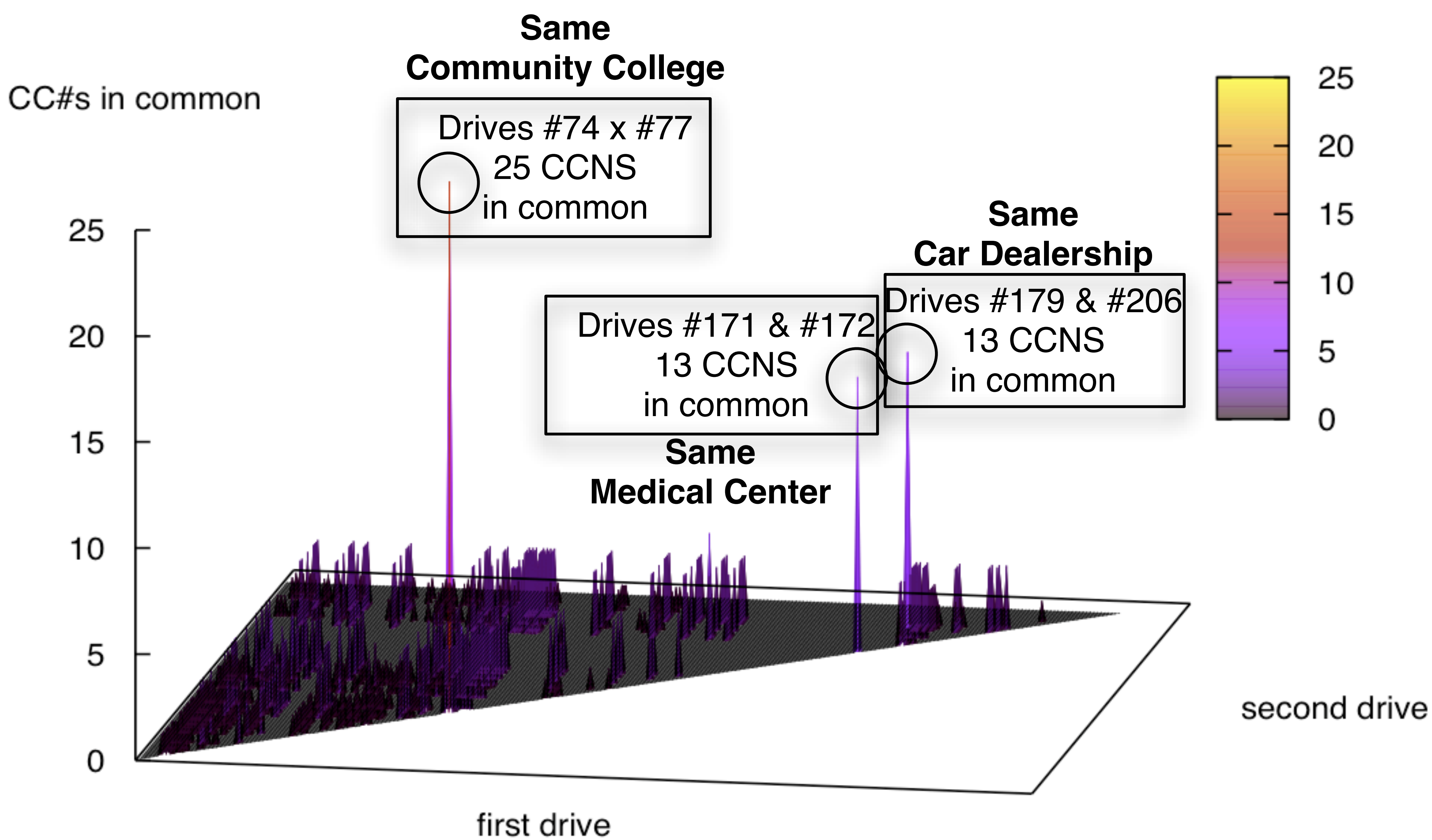
second drive

2	0	0	4	0
1	1	3	**35**	
0	3	1		
1	0			
4				

High correlation indicates likely extrinsic relationship

We correlated 250 drives and automatically found drives belonging to the same organization.

The algorithm *finds* the correlation selectors.

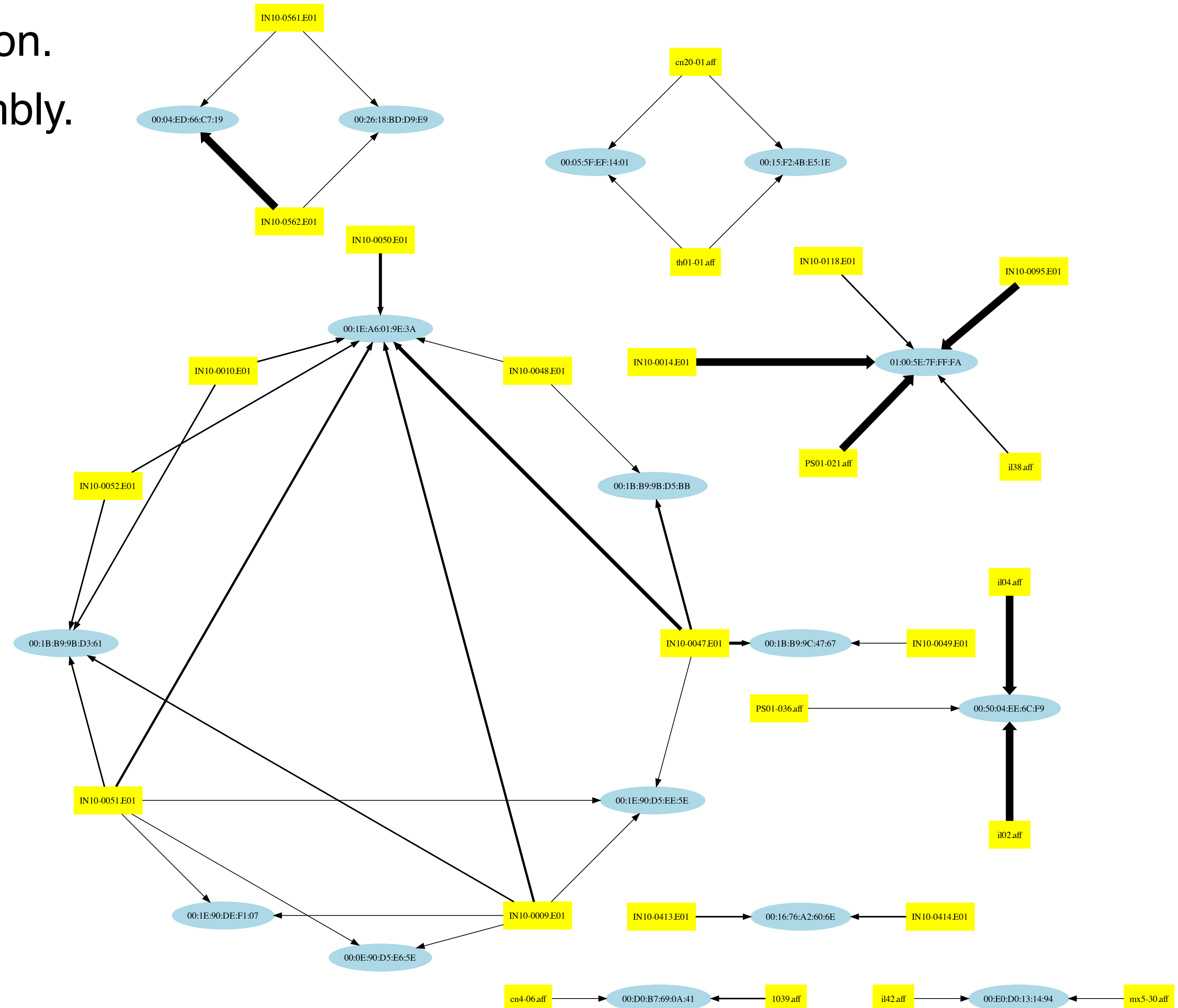


"Forensic Feature Extraction and Cross-Drive Analysis," 2006

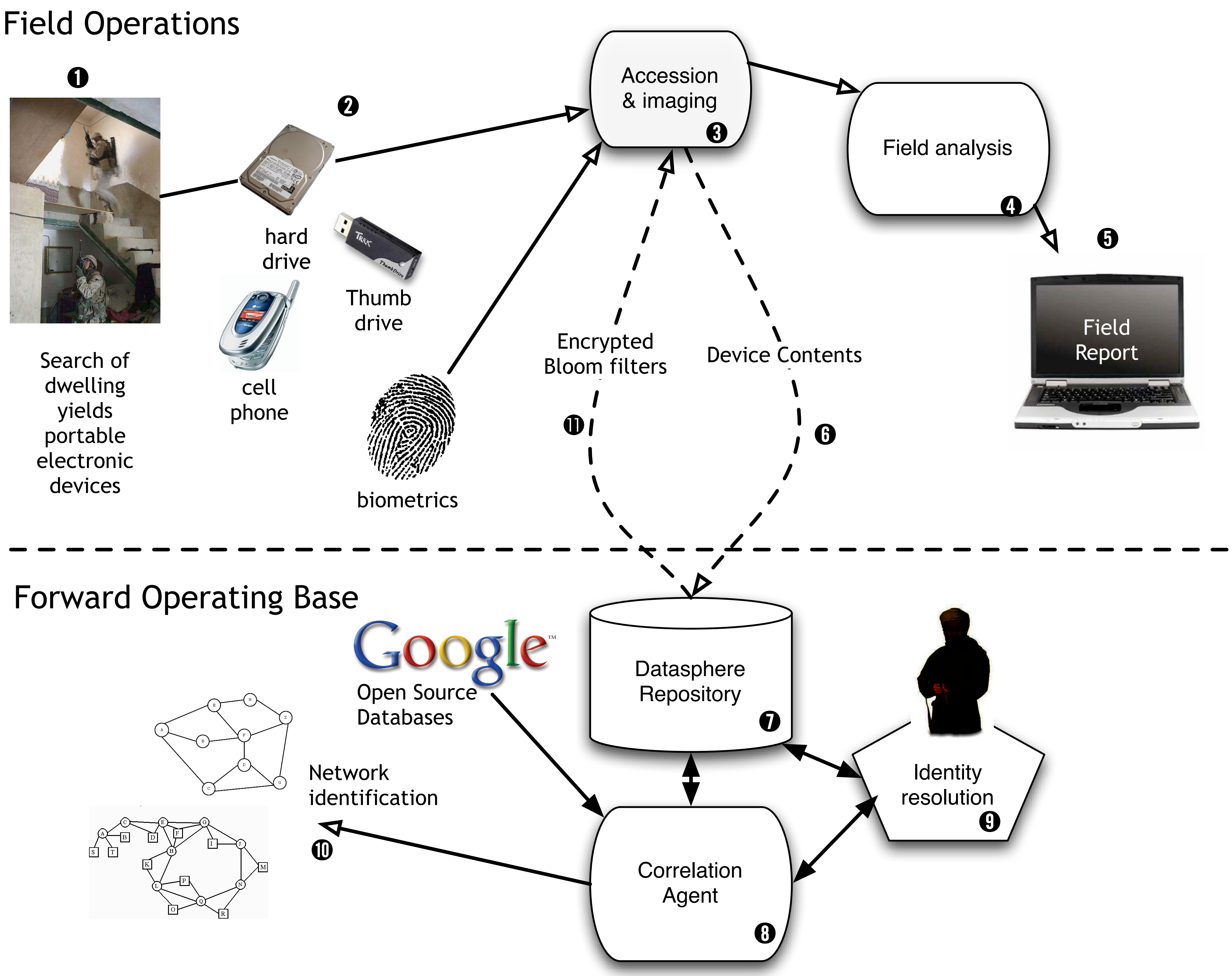
2011: IP Carving and Network Reassembly

bulk_extractor extended to recognize and validate network data.

- Automated extraction of Ethernet MAC addresses (client & server)
- Large-scale corpus correlation.
- Automated network reassembly.



NPS Big Vision for Automated Forensics



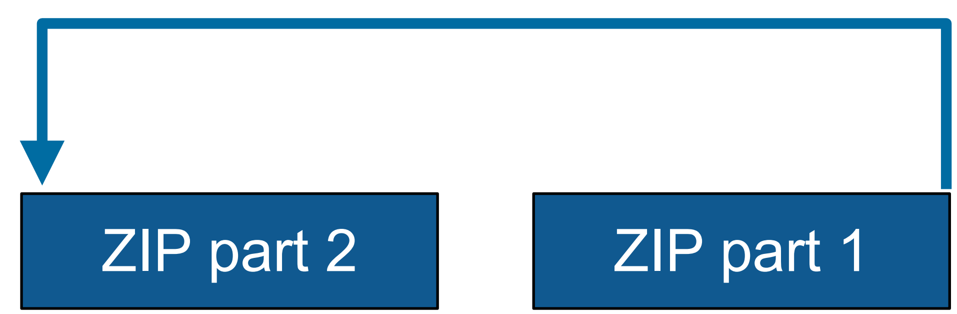
Stream-based forensics with bulk_extractor

Stream-Based Disk Forensics:
Scan the disk from beginning to end; do your best.

1. Read all of the blocks in order.
2. Look for information that might be useful.
3. Identify & extract what's possible in a single pass.

Advantages:

- No disk seeking.
- Read the disk at maximum transfer rate.
- Reads *all the data* — allocated files, deleted files, file fragments.



Disadvantages:

- Fragmented files won't be recovered:
  - — *Compressed files with part2-part1 ordering*
  - — *FIles with internal fragmentation (.doc)*
- A pass through the file system is needed to map contents to file names.

bulk_extractor: a high-speed disk scanner.

1	2	3	4	1	2	3	4	1	2

Written in C, C++ and Flex

- Command-line tool.
- Linux, MacOS, Windows (compiled with mingw)

Key Features:

- Uses regular expressions and rules to scan for:
  - *email addresses; credit card numbers; JPEG EXIFs; URLs; Email fragments.*
- Recursively re-analyzes ZIP components.
- Produces a histogram of the results.
- Multi-threaded.
  - *Disk is "striped" into pages*
  - *Results stored in mostly-ordered “feature files”*

Challenges:

- Must work with evidence files of *any size* and on *limited hardware.*
- Users can't provide their data when the program crashes.
- Users are *analysts* and *examiners*, not engineers.

bulk_extractor output: text files of "features" and context.

email addresses from domexusers:

```
48198832  domexuser2@gmail.com     tocol>____<name>domexuser2@gmail.com/Home</name>____
48200361  domexuser2@live.com      tocol>____<name>domexuser2@live.com</name>____<pass
48413829  siege@preoccupied.net    siege) O'Brien <siege@preoccupied.net>_hp://meanwhi
48481542  danilo@gnome.org         Danilo __egan <danilo@gnome.org>_Language-Team:
48481589  gnom@prevod.org          : Serbian (sr) <gnom@prevod.org>_MIME-Version:
49421069  domexuser1@gmail.com     server2.name", "domexuser1@gmail.com");__user_pref("
49421279  domexuser1@gmail.com     er2.userName", "domexuser1@gmail.com");__user_pref("
49421608  domexuser1@gmail.com     tp1.username", "domexuser1@gmail.com");__user_pref("
```

Histogram:

```
n=579   domexuser1@gmail.com
n=432   domexuser2@gmail.com
n=340   domexuser3@gmail.com
n=268   ips@mail.ips.es
n=252   premium-server@thawte.com
n=244   CPS-requests@verisign.com
n=242   someone@example.com
```

bulk_extractor success:
City of San Luis Obispo Police Department, Spring 2010

District Attorney filed charges against two individuals:

- Credit Card Fraud
- Possession of materials to commit credit card fraud.

Defendants:

- arrested with a computer.
- Expected to argue that defends were unsophisticated and lacked knowledge.

Examiner given 250GiB drive *the day before preliminary hearing.*

- In 2.5 hours Bulk Extractor found:
  - *Over 10,000 credit card numbers on the HD (1000 unique)*
  - *Most common email address belonged to the primary defendant (possession)*
  - *The most commonly occurring Internet search engine queries concerned credit card fraud and bank identification numbers (intent)*
  - *Most commonly visited websites were in a foreign country whose primary language is spoken fluently by the primary defendant.*
- Armed with this data, the DA was able to have the defendants held.

Eliminating false positives:
Many of the email addresses come with Windows!

Sources of these addresses:

- Windows binaries
- SSL certificates
- Sample documents

n=579	domexuser1@gmail.com
n=432	domexuser2@gmail.com
n=340	domexuser3@gmail.com
n=268	**ips@mail.ips.es**
n=252	**premium-server@thawte.com**
n=244	**CPS-requests@verisign.com**
n=242	**someone@example.com**

It's important to suppress email addresses not relevant to the case.

Approach #1 — Suppress emails seen on many other drives.

Approach #2 — Stop list from bulk_extractor run on clean installs.

Both of these methods *white list* commonly seen emails.

- Operating Systems have a LOT of emails. (FC12 has 20,584!)
- Should we give the Linux developers a free past?

Approach #3: Context-sensitive stop list.

Instead of extracting just the email address, extract the context:

- Offset: **351373329**
- Email: `zeeshan.ali@nokia.com`
- Context: `ut_Zeeshan Ali <zeeshan.ali@nokia.com>, Stefan Kost <`

- Offset: **351373366**
- Email: `stefan.kost@nokia.com`
- Context: `>, Stefan Kost <stefan.kost@nokia.com>____________sin`

— *Here "context" is 8 characters on either side of feature.*

We created a context-sensitive stop list for Microsoft Windows XP, 2000, 2003, Vista, and several Linux.

Total stop list: 70MB (628,792 features; 9MB ZIP file)

Applying it to domexusers HD image:

- # of emails found: 9143 ➔ 4459

without stop list		with stop list	
n=579	domexuser1@gmail.com	n=579	domexuser1@gmail.com
n=432	domexuser2@gmail.com	n=432	domexuser2@gmail.com
n=340	domexuser3@gmail.com	n=340	domexuser3@gmail.com
n=268	ips@mail.ips.es	n=192	domexuser2@live.com
n=252	premium-server@thawte.com	n=153	domexuser2@hotmail.com
n=244	CPS-requests@verisign.com	n=146	domexuser1@hotmail.com
n=242	someone@example.com	n=134	domexuser1@live.com
n=237	inet@microsoft.com	n=91	premium-server@thawte.com
n=192	domexuser2@live.com	n=70	talkback@mozilla.org
n=153	domexuser2@hotmail.com	n=69	hewitt@netscape.com
n=146	domexuser1@hotmail.com	n=54	DOMEXUSER2@GMAIL.COM
n=134	domexuser1@live.com	n=48	domexuser1%40gmail.com@imap.gmail.com
n=115	example@passport.com	n=42	domex2@rad.li
n=115	myname@msn.com	n=39	lord@netscape.com
n=110	ca@digsigtrust.com	n=37	49091023.6070302@gmail.com

http://afflib.org/downloads/feature_context.1.0.zip

bulk_extractor: Implemented as a set of C++ classes

Forensic Buffers and Path:

- sbuf_t — Holds data, margin, and forensic path of each page.
- pos0_t — Path of byte at sbuf[0]

```
100                   Offset at 100 bytes.
100-GZIP-500          At offset 100, GZIP compressed, 500 bytes further in
```

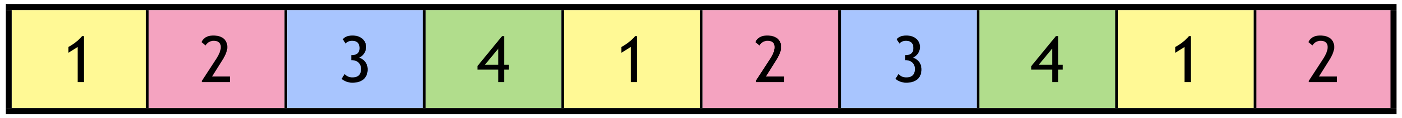
- feature_recorder — Holds output for each feature type

Plug-In Scanner System

- Each scanner is a C++ function that can be linked or loaded at run-time(*)
- Simple scanners look for features in bulk data and report them
  - *scan_accts, scan_aes, scan_bulk, scan_ccns2, scan_email, scan_exif, scan_find, scan_headers, scan_net, scan_wordlist*
- Scanners can instantiate files:
  - *scan_kml*
- Scanners can be recursive.
  - *scan_base64, scan_gzip, scan_hiberfile, scan_pdf, scan_zip*

bulk_extractor: Speed from multi threading

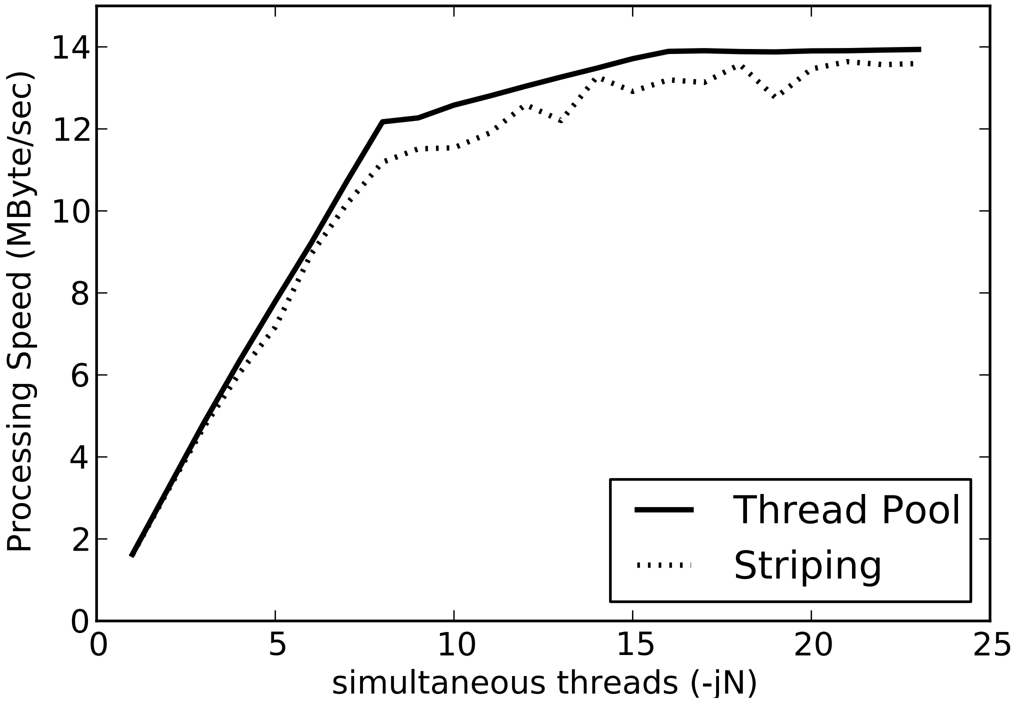
Primary thread:



- Iterator reads “pages” of forensic data and passes each page to a “worker.”
- Iterators available for:
  - *raw & splitraw files*
  - *AFF, E01*
  - *Directory Hierarchies.*
- MD5 is computed automatically as data is read (source validation).
- Generates DFXML file with:
  - *Tool compile and runtime provenance.*
  - *Status reports of what is found, errors, etc.*

Worker Threads:

- One per core.
- Automatically figures out how many cores you have.



bulk_extractor: Crash Protection

Every forensic tool crashes.

- Tools routinely used with data fragments, non-standard codings, etc.
- Evidence that makes the tool crash typically cannot be shared with the developer.

Crash Protection #1: “-c” option.

- Catches Unix/Windows signals for invalid memory access.
- Abandons current page; goes to next.
- Danger: invalid memory access may corrupt other operations.
  - *So it’s not the default!*

Crash Protection #2: restart.

- Bulk_extractor checkpoints current page in the file config.cfg
- After a crash, just hit up-arrow and return; bulk_extractor restarts at next page.

Bulk_extractor's magic — opportunistic decompression

Most forensic tools recover:

- allocated files
- "deleted" files
- carving of unallocated area.

bulk_extractor uses a different methodology:

- Carving and Named Entity Recognition
- Identification, Decompression and Re-Analysis of compressed data.

This helps with:

- hibernation files and fragments (hibernation files move around)
- swap file fragments
- browser cache fragments (gzip compression)

Post-processing the feature files

The feature files are designed for easy, rapid processing.

- Tab-Delimited
  - *path, feature, context*
- Text (UTF-8)

bulk_diff.py: prepares difference of two bulk_extractor runs.

- Designed for timeline analysis.
- Developed with analysts.
- Reports “what’s changed.”
  - *Actually, “what’s new” turned out to be more useful.*
  - *“what’s missing” includes data inadvertantly overwritten.*

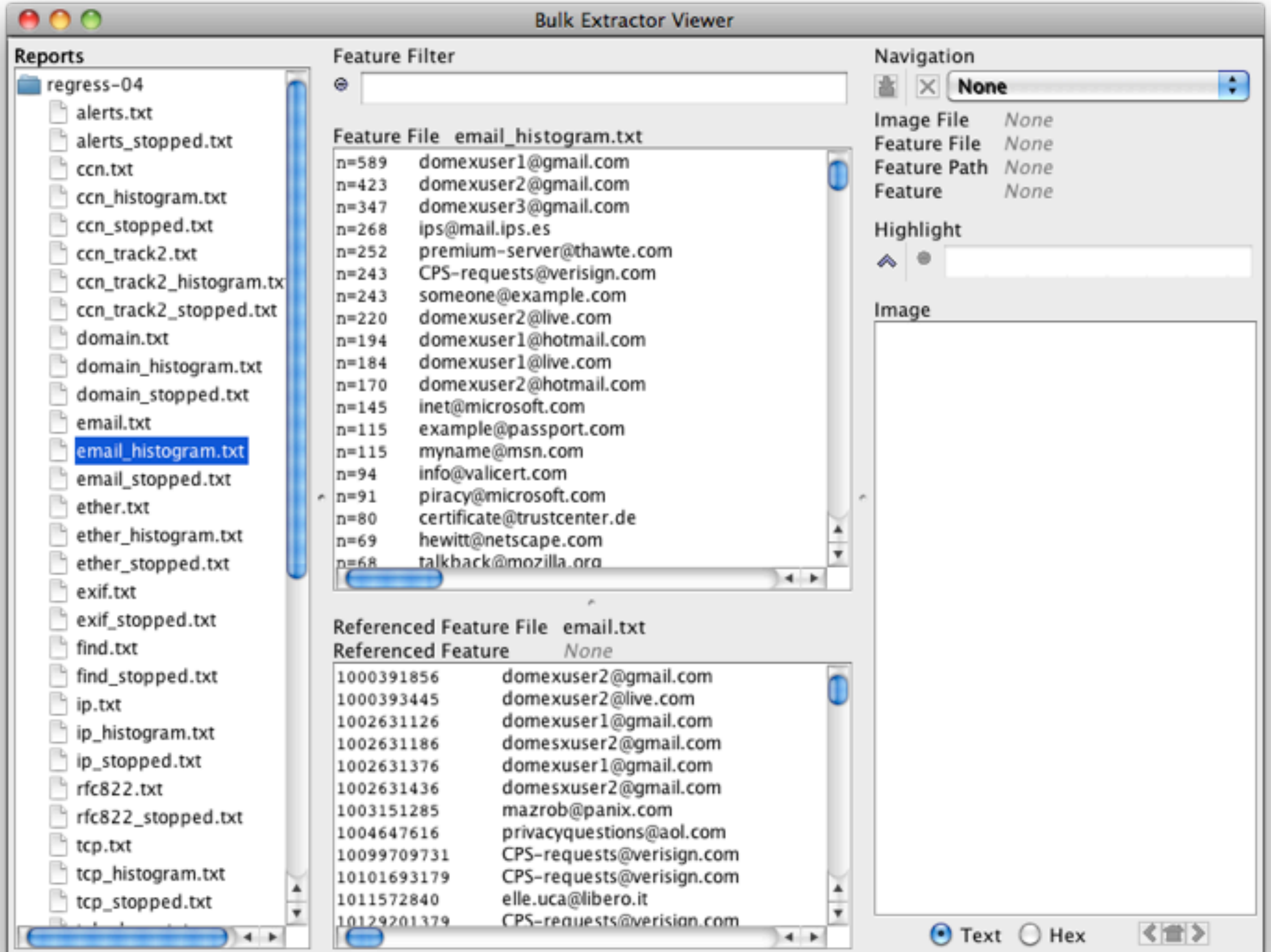
identify_filenames.py: Reports files responsible for features.

- Requires DFXML run (fiwalk) for disk image.
- Currently a two-step process; could be built in to bulk_extractor

bulk_extractor GUI

100% Java

Uses bulk_extractor to view contents of compressed containers.



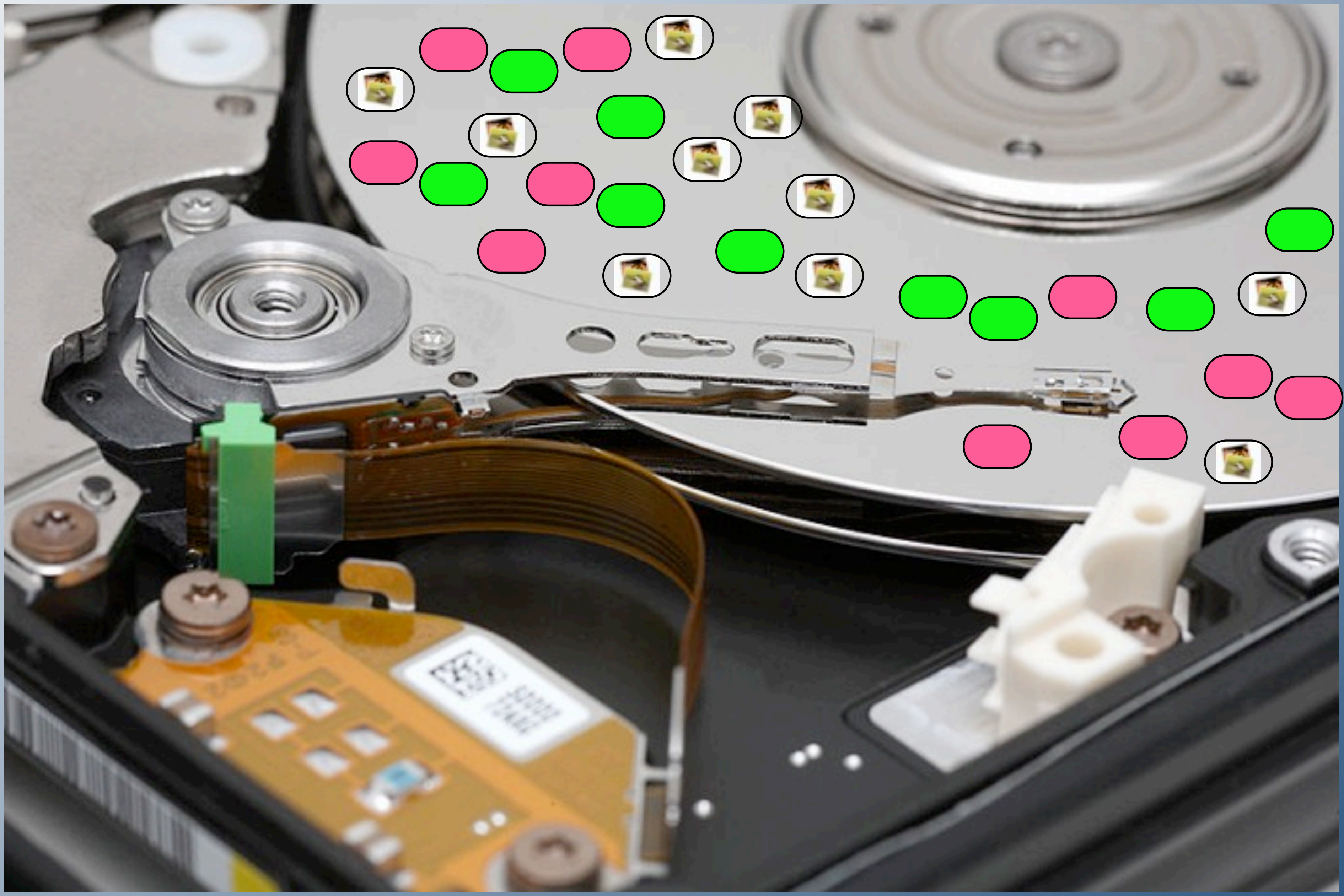
Future Work

Hash-based carving

Carving more file types

Lat/Long from GPS devices


```
<trkpt lat="38.848029" lon="-77.067389"><ele>16.00</
ele><time>2010-09-23T00:07:52Z</
time><extensions><gpxtpx:TrackPointExtension><gpxtpx:speed
>5.49</gpxtpx:speed><gpxtpx:course>0.00</gpxtpx:course></
gpxtpx:TrackPointExtension></extensions></trkpt><trkpt
lat="38.848076" lon="-77.067399"><ele>16.48</
ele><time>2010-09-23T00:07:53Z</
time><extensions><gpxtpx:TrackPointExtension><gpxtpx:speed
>5.49</gpxtpx:speed><gpxtpx:course>14.12</gpxtpx:course></
gpxtpx:TrackPointExtension></extensions></trkpt>
```



Part 1:
Distinct Block Recognition

Distinct block: a block of data that does not arise by chance more than once.

Consider a disk sector with 512 bytes.

- There are $2^{512\text{x}8} \approx 10^{1,233}$ different sectors.
- A randomized sector is likely to be "distinct."
  - — *e.g. encryption keys, high-entropy data, etc.*

A3841FBC3
84817DEF3
8239FF938
419893FF3

Distinct Block Hypothesis #1:

- If a block of data from a file is distinct, then a copy of that block found on a data storage device is evidence that the file was once present.

Distinct Block Hypothesis #2:

- If a file is known to have been manufactured using some high-entropy process, and if the blocks of that file are shown to be distinct throughout a large and representative corpus, then those blocks can be treated as if they are distinct.

What kinds of files are likely to have distinct blocks?

A bock from a JPEG image should be distinct.

208 distinct 4096-byte block hashes

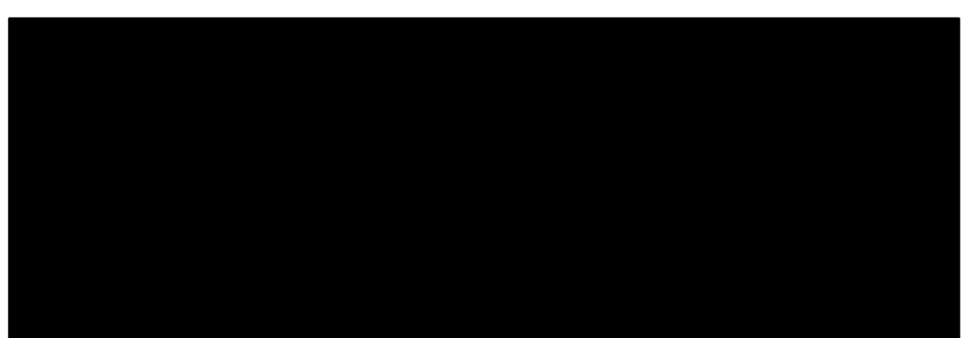
"You cannot step twice into the same river."

— *Heraclitus*

"You cannot step twice into the same sunny day."

— *Distinct Block Hypothesis*

… Unless the image is all black.



Probably no distinct blocks.

Even though JPEGs may *look* similar, they do not contain identical blocks.

208 distinct 4096-byte block hashes

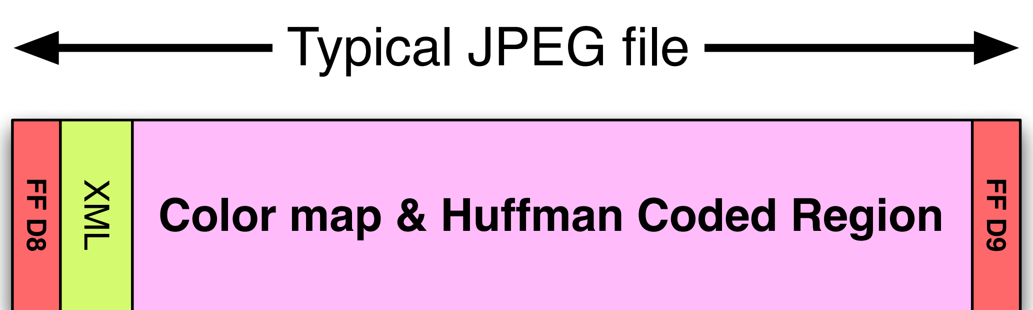
∩

204 distinct 4096-byte block hashes

=

0 block hashes in common

Even with JPEG headers, XML, and Color Maps:



Other kinds of files likely have distinct blocks as well.

Files with high entropy:

- Multimedia files (Video)
- Encrypted files.
- Files with *original* writing.
- Files with just a few characters "randomly" distributed
  - *There are 10^{33} ($^{512!}/_{500!}$) different sectors with 500 NULLs and 12 ASCII spaces!*

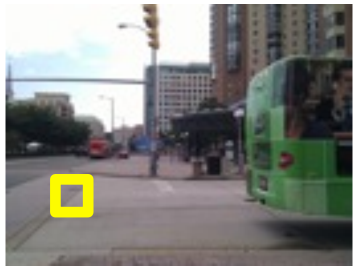
What kinds of files won't have distinct blocks?

- Those that are filled with a constant character.
- Simple patterns (`00 FF 00 FF 00 FF`)

Modern file systems align files on sector boundaries.

Place a file with distinct blocks on a disk.

- Distinct disk blocks => Distinct disk sectors.



208 distinct 4096-byte block hashes

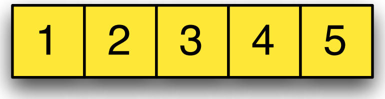
So finding a distinct block on a disk is evidence that the file was present.

- *(Distinct Block Hypothesis #1:*
  - *If a block of data from a file is distinct, then a copy of that block found on a data storage device is evidence that the file was once present.)*

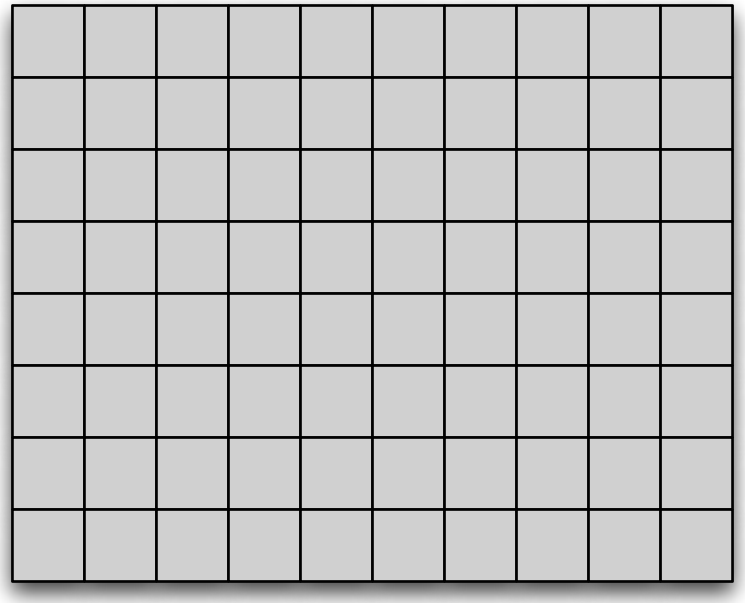
Hash-based carving

Input:

- 1 or more *Master Files*



- A disk image



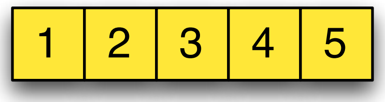
Algorithm:

- Hash each sector of each master file.
  - *Store hashes in a map[].*
- Hash each sector of the disk image.
  - *Check each sector hash against the map[]*
  - *If a sector hash matches multiple files, choose the file that creates the longer run.*

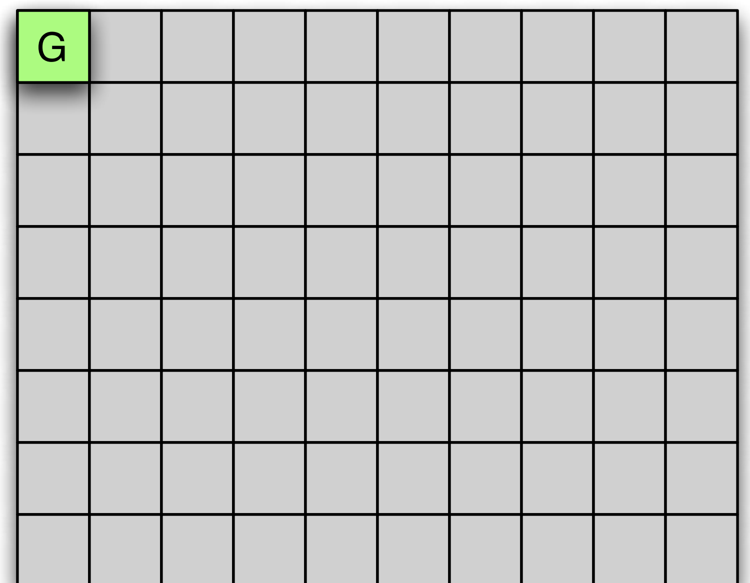
Hash-based carving

Input:

- 1 or more *Master Files*



- A disk image



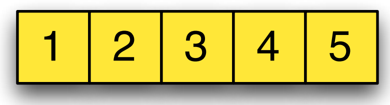
Algorithm:

- Hash each sector of each master file.
    — *Store hashes in a map[].*
- Hash each sector of the disk image.
    — *Check each sector hash against the map[]*
    — *If a sector hash matches multiple files, choose the file that creates the longer run.*

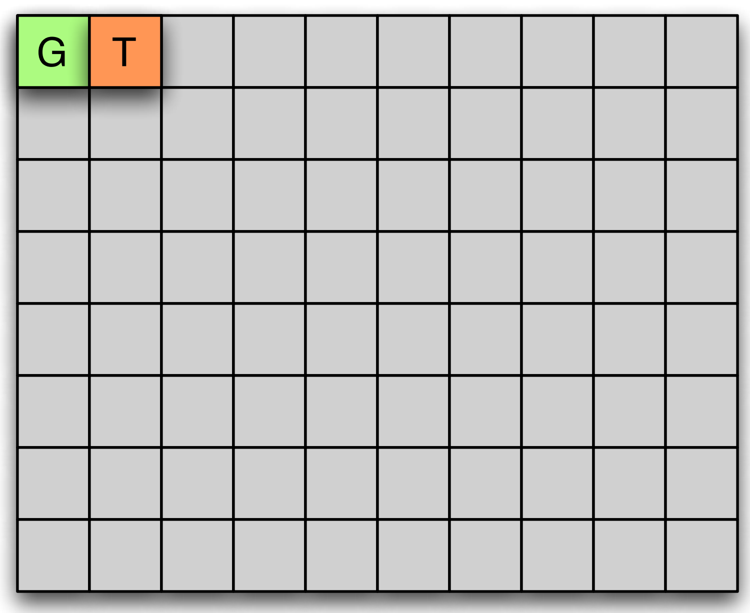
Hash-based carving

Input:

- 1 or more *Master Files*



- A disk image



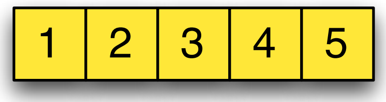
Algorithm:

- Hash each sector of each master file.
  — *Store hashes in a map[].*
- Hash each sector of the disk image.
  — *Check each sector hash against the map[]*
  — *If a sector hash matches multiple files, choose the file that creates the longer run.*

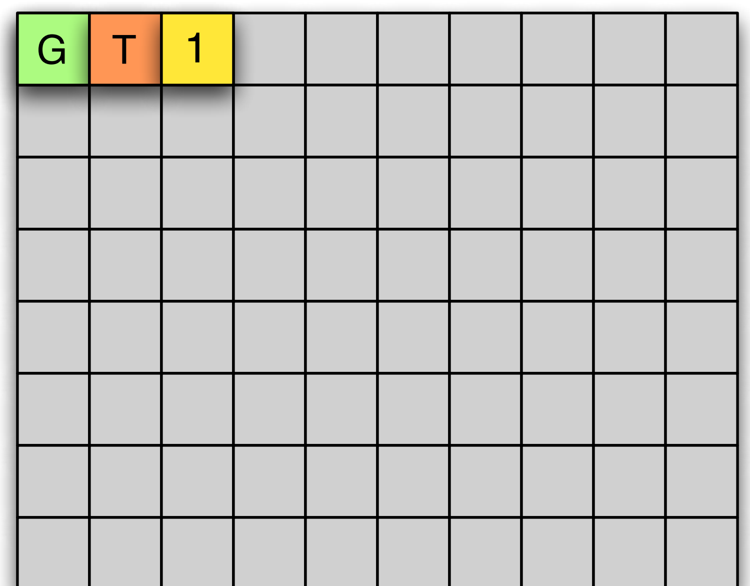
Hash-based carving

Input:

- 1 or more *Master Files*



- A disk image



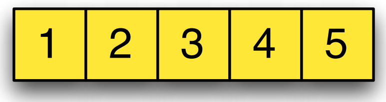
Algorithm:

- Hash each sector of each master file.
  - *Store hashes in a map[].*
- Hash each sector of the disk image.
  - *Check each sector hash against the map[]*
  - *If a sector hash matches multiple files, choose the file that creates the longer run.*

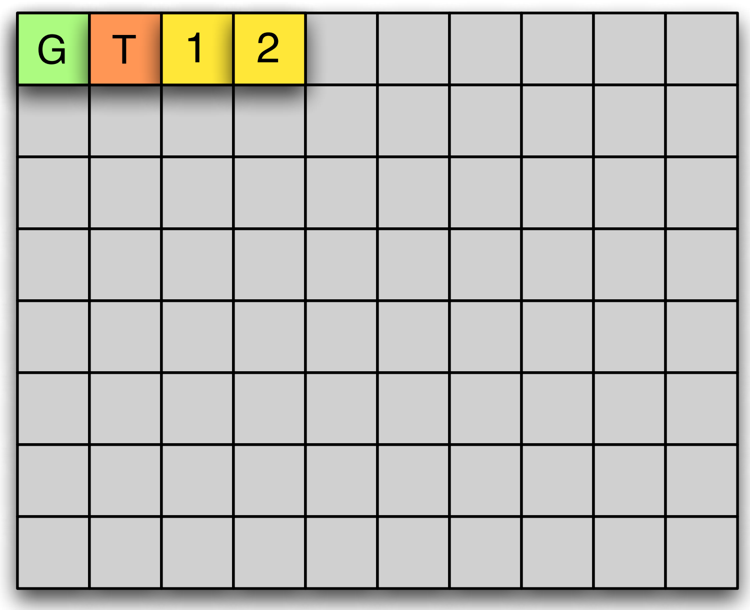
Hash-based carving

Input:

- 1 or more *Master Files*



- A disk image



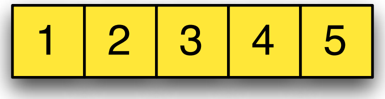
Algorithm:

- Hash each sector of each master file.
  — *Store hashes in a map[].*
- Hash each sector of the disk image.
  — *Check each sector hash against the map[]*
  — *If a sector hash matches multiple files, choose the file that creates the longer run.*

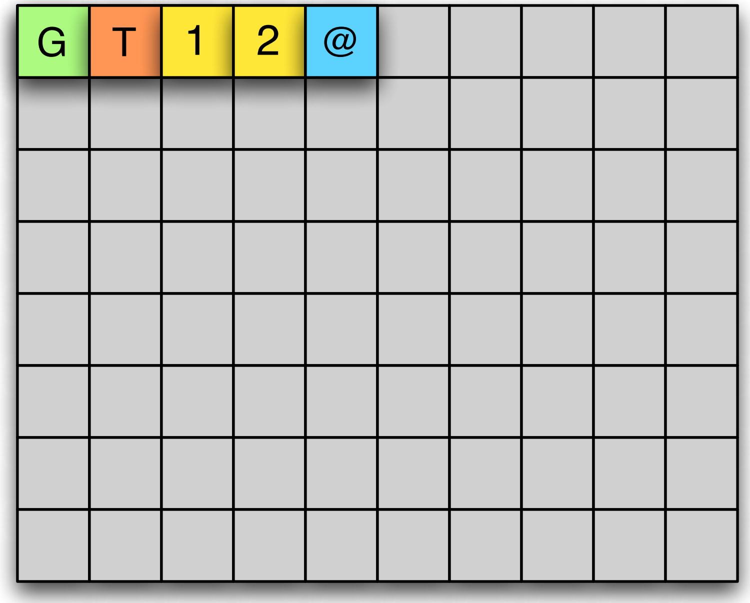
Hash-based carving

Input:

- 1 or more *Master Files*



- A disk image



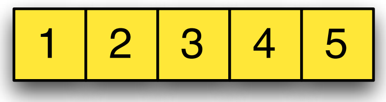
Algorithm:

- Hash each sector of each master file.
  - *Store hashes in a map[].*
- Hash each sector of the disk image.
  - *Check each sector hash against the map[]*
  - *If a sector hash matches multiple files, choose the file that creates the longer run.*

Hash-based carving

Input:

- 1 or more *Master Files*



- A disk image

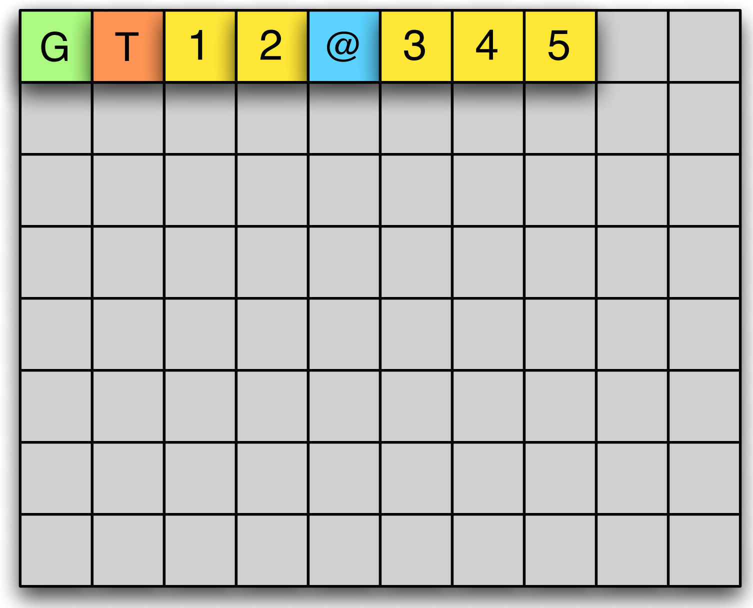
Algorithm:

- Hash each sector of each master file.
  - *Store hashes in a map[].*
- Hash each sector of the disk image.
  - *Check each sector hash against the map[]*
  - *If a sector hash matches multiple files, choose the file that creates the longer run.*

frag_find is a high-performance hash-based carver.

Implementation:

- C++
- Pre-filtering with NPS Bloom package.
  - — *All sector hashes are put in a Bloom Filter*
  - — *Block size = Sector Size = 512 bytes*



Output in Digital Forensics XML:

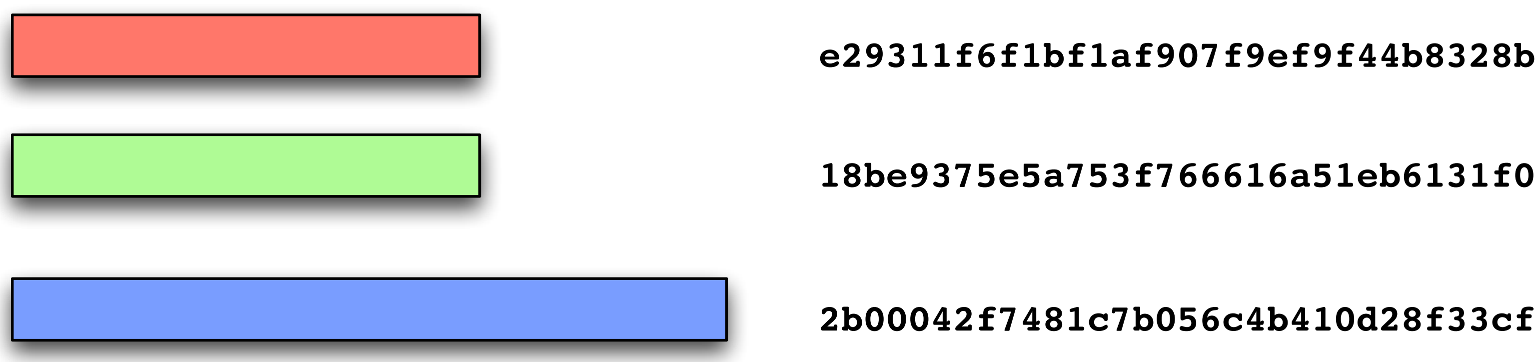
```
<fileobject>
  <filename>DCIM/100CANON/IMG_0001.JPG</filename>
  <byte_runs>
    <run file_offset='0' fs_offset='55808' img_offset='81920' len='855935'/>
      <hashdigest type='MD5'>b83137bd4ba4b56ed856be8a8e2dc141</hashdigest>
      <hashdigest type='SHA1'>03eaa4a5678542039c29a5ccf12b3d71ae96cbd2</hashdigest>
  </byte_runs>
</fileobject>
```

Uses:

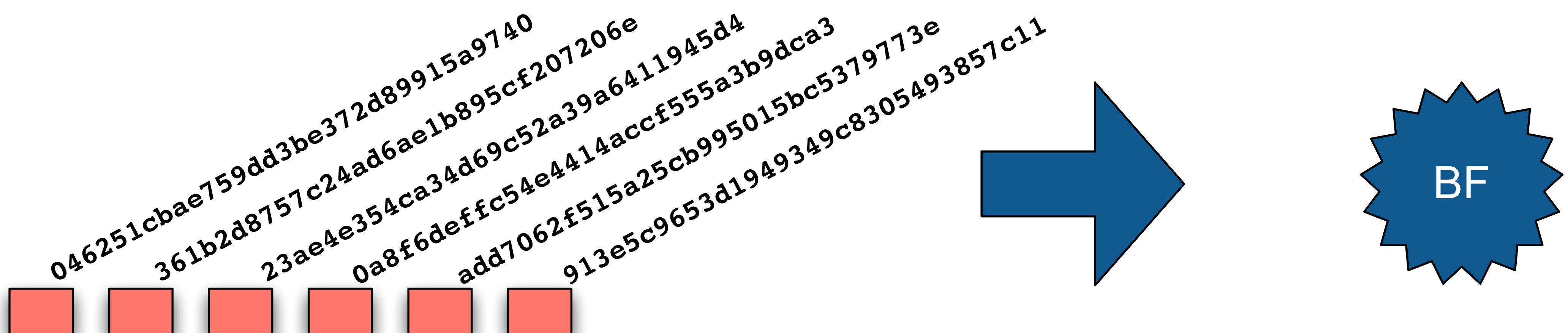
- Exfiltration of sensitive documents;
- Data Loss Detection; etc.
- Download from http://afflib.org/

Distinct Block Recognition can be used to find objectionable material.

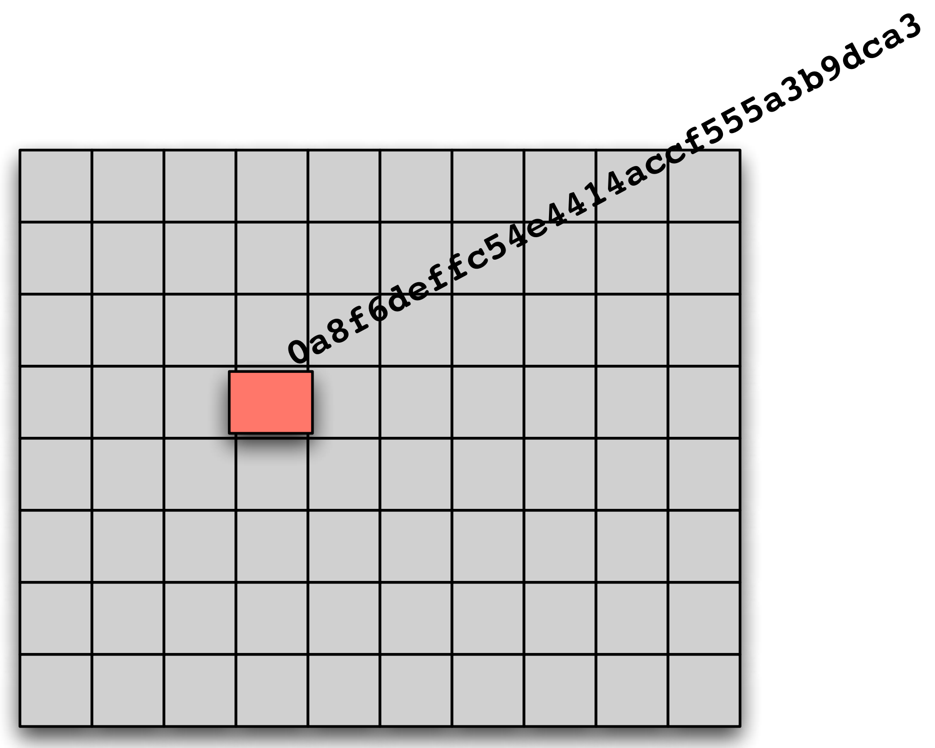
Currently objectionable materials are detected with hash sets.



With the block-based approach, each file is broken into blocks, and each block hash is put into a Bloom Filter:

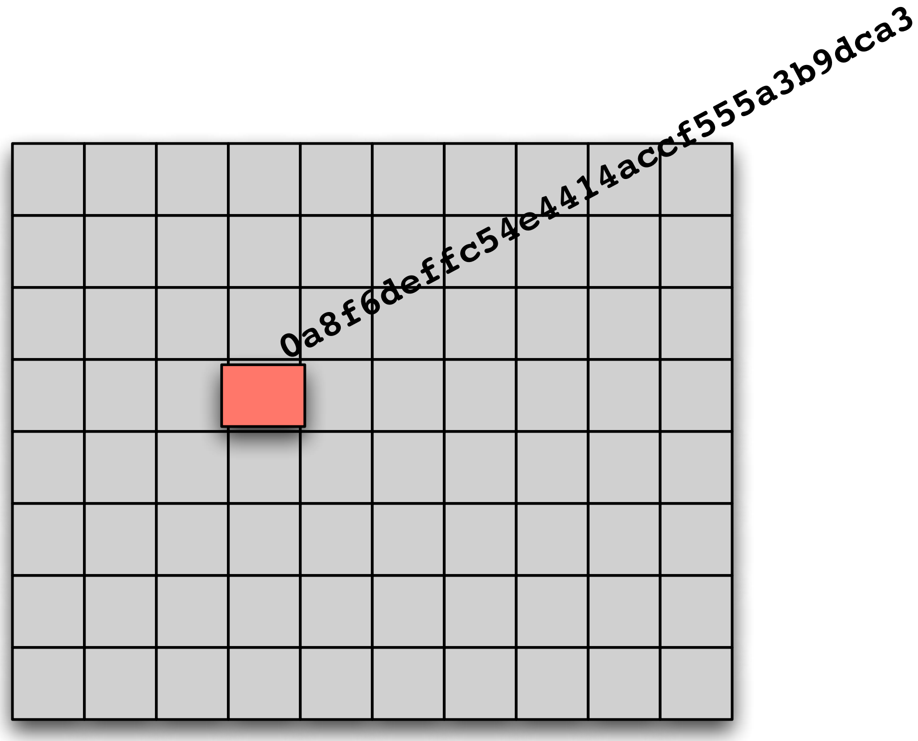


If a sector of an objectionable file is found on a drive...



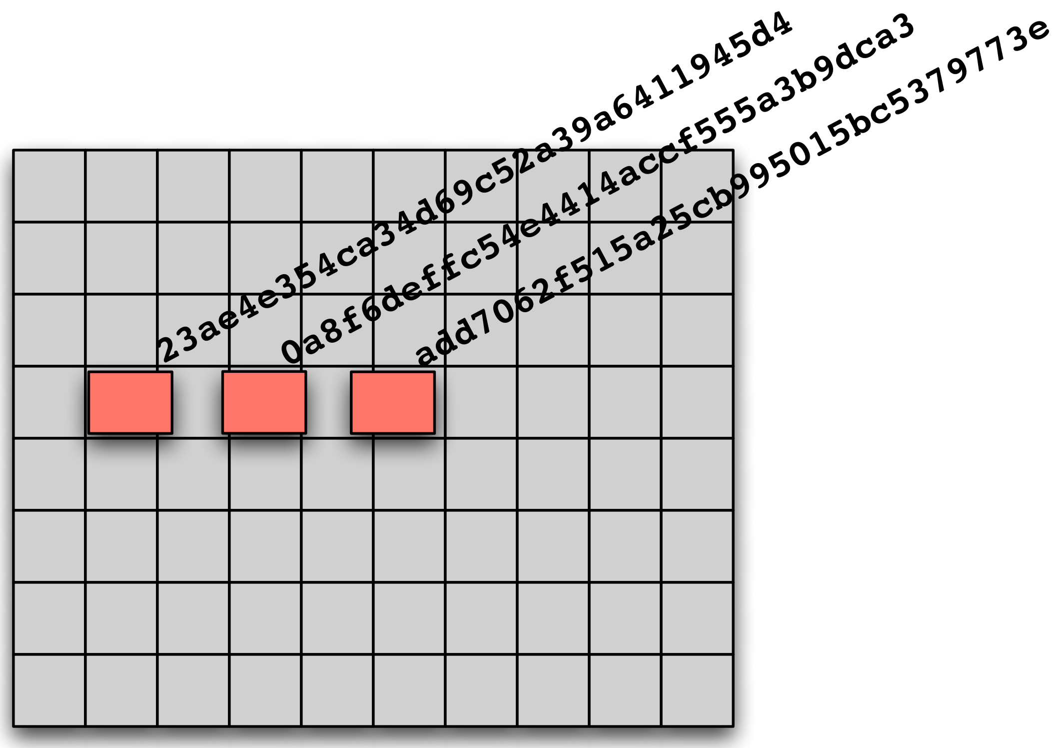
If a sector of an objectionable file is found on a drive...

Then either the entire file was once present…



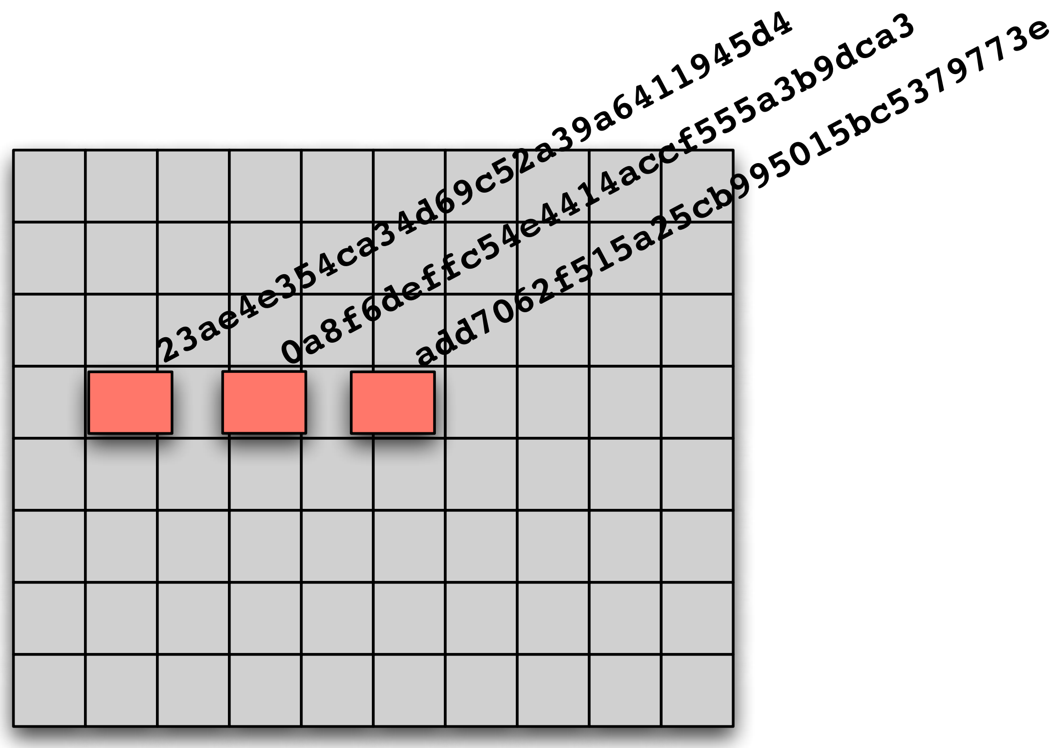
If a sector of an objectionable file is found on a drive...

Then either the entire file was once present…



If a sector of an objectionable file is found on a drive...

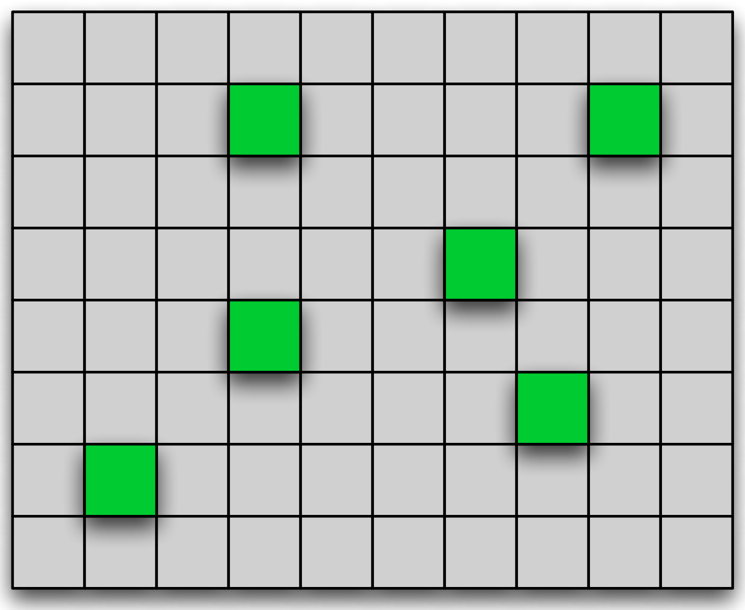
Then either the entire file was once present…



… or else the sector really isn't distinct.

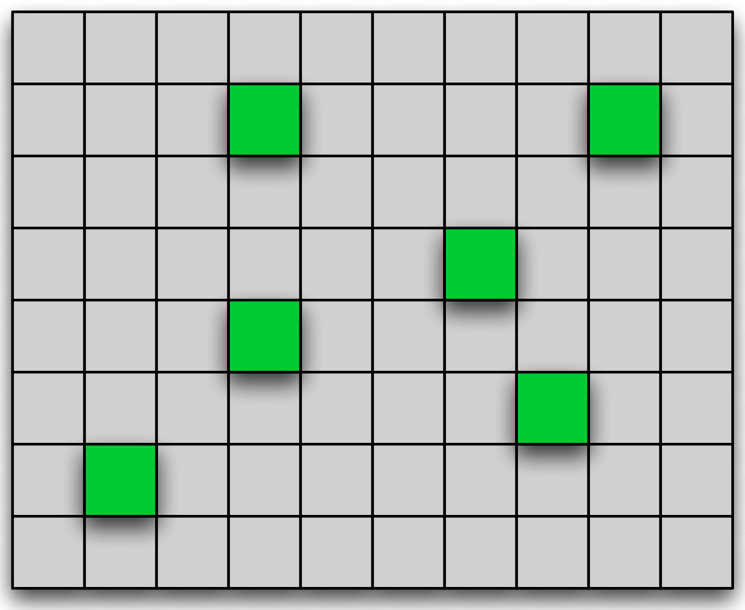
Random sampling can rapidly find the presence of objectionable material on a large storage device.

1TB drive = 2 billion 512-byte sectors.



Random sampling can rapidly find the presence of objectionable material on a large storage device.

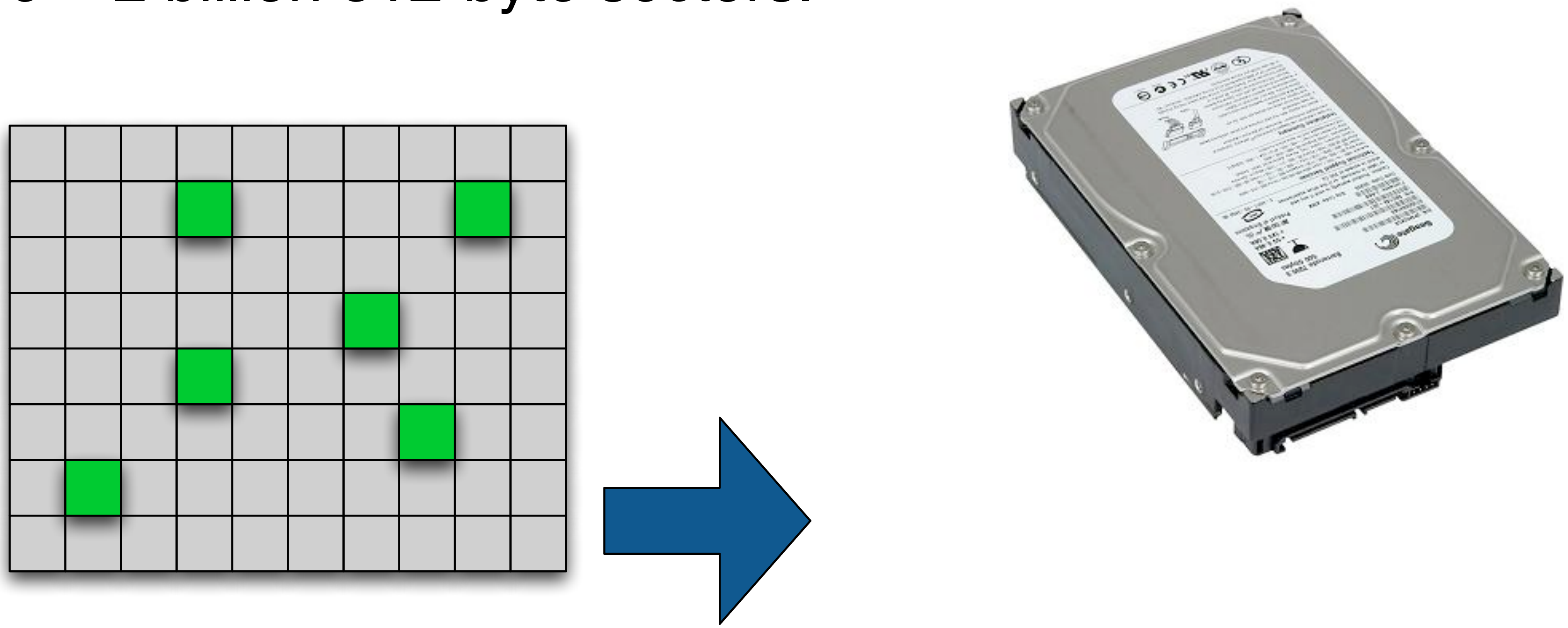
1TB drive = 2 billion 512-byte sectors.



We can pick random sectors, hash them, and probe the Bloom Filter.

Random sampling can rapidly find the presence of objectionable material on a large storage device.

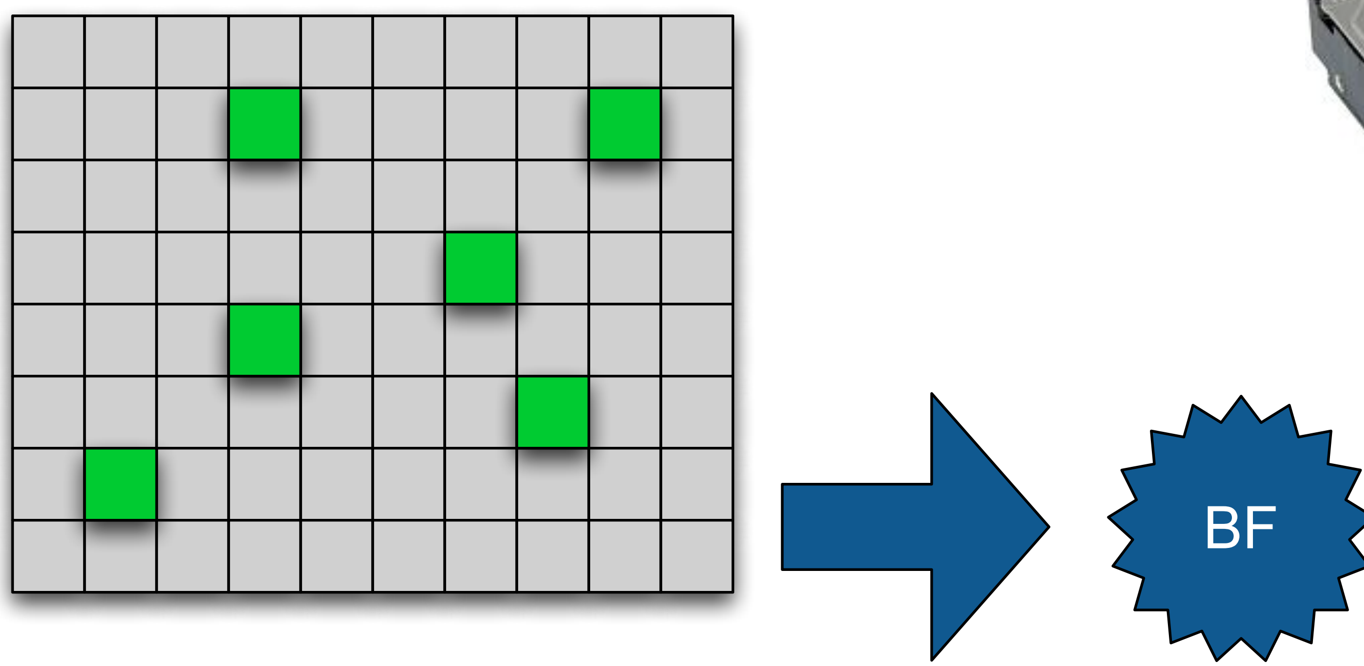
1TB drive = 2 billion 512-byte sectors.



We can pick random sectors, hash them, and probe the Bloom Filter.

Random sampling can rapidly find the presence of objectionable material on a large storage device.

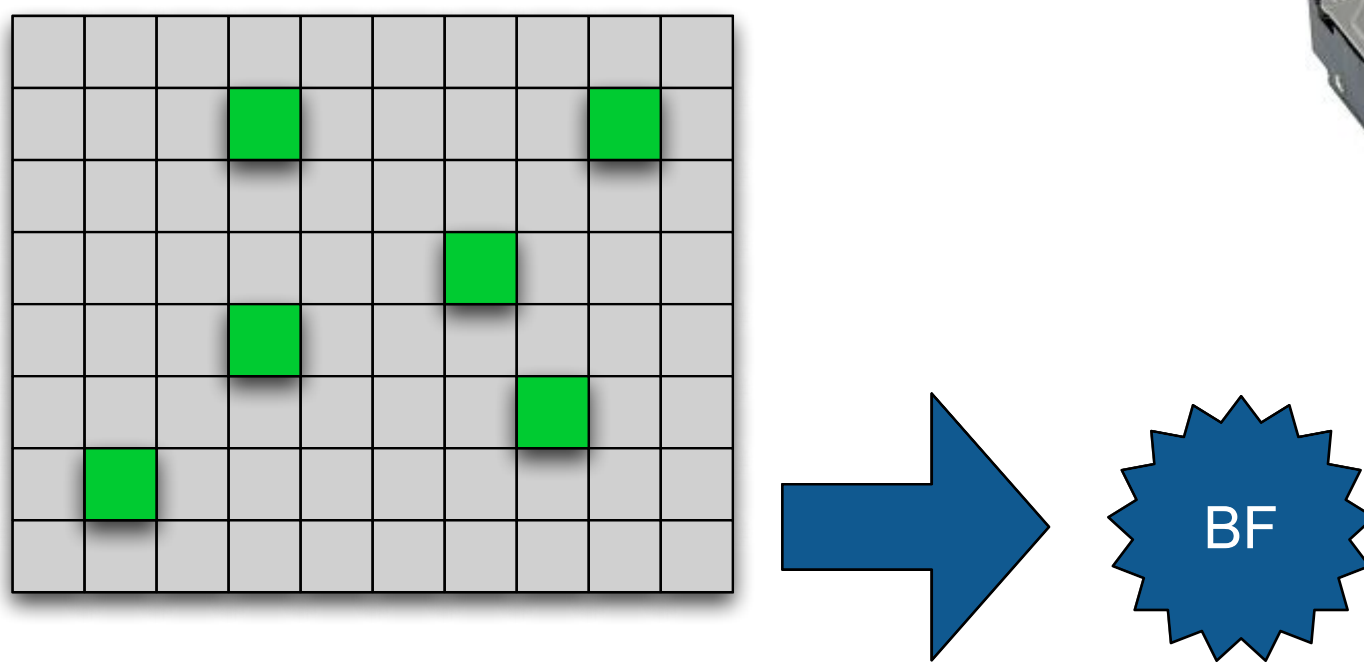
1TB drive = 2 billion 512-byte sectors.



We can pick random sectors, hash them, and probe the Bloom Filter.

Random sampling can rapidly find the presence of objectionable material on a large storage device.

1TB drive = 2 billion 512-byte sectors.

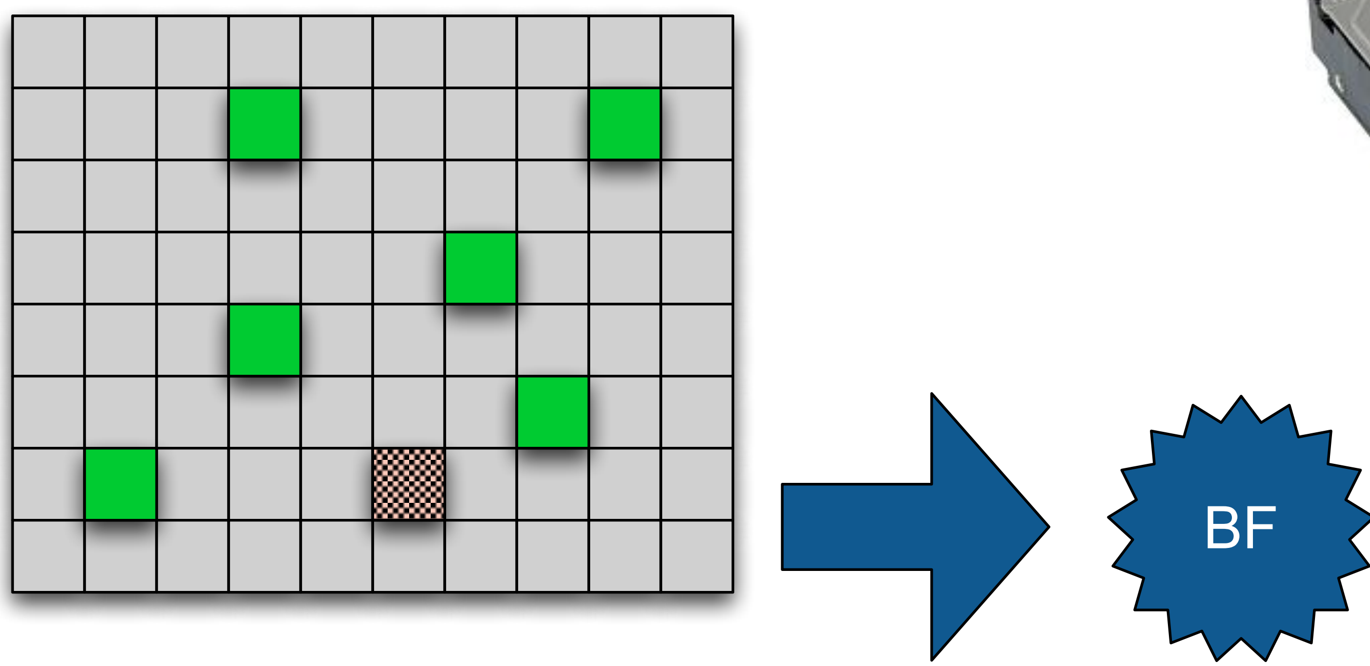


We can pick random sectors, hash them, and probe the Bloom Filter.

Finding a match indicates the presence of objectionable material.

Random sampling can rapidly find the presence of objectionable material on a large storage device.

1TB drive = 2 billion 512-byte sectors.



We can pick random sectors, hash them, and probe the Bloom Filter.

Finding a match indicates the presence of objectionable material.

Advantages of block recognition with Bloom Filters

Speed & Size:

- Can store billions of sector hashes in a 4GB object.
- False positive rate can be made very small.

Security:

- File corpus can't be reverse-engineered from BF
- BF can be encrypted for further security.

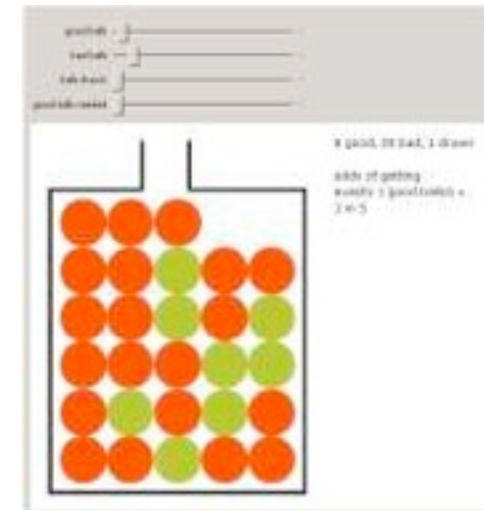
False positive rate:

- $m = 2^{32}$ $k = 4$ n=80 million $p < .0000266$ (512MiB BF)

The odds of finding the objectionable content depends on the amount of content and the # of sampled sectors.

Sectors on disk: 2,000,000,000 (1TB)

Sectors with bad content: 200,000 (100 MB)



Chose one sector. Odds of missing the data:

- (2,000,000,000 - 200,000) / (2,000,000,000) = 0.9999
- You are *very likely* to miss one of 200,000 sectors if you pick just one.

Chose two sectors. Odds of missing the data on both tries:

- 0.999 * (1,999,999,999-200,000) / (1,999,999,999) = .9998
- You are still *very likely* to miss one of 200,000 sectors if you pick two…
- … but a little less likely

Increasing # of samples *decreases* the odds of missing the data.

- The "Urn Problem" from statistics.

The more sectors picked, the less likely you are to miss *all* of the sectors that have objectionable content.

$$p = \prod_{i=1}^{n} \frac{((N-(i-1))-M)}{(N-(i-1))} \quad (1)$$

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.

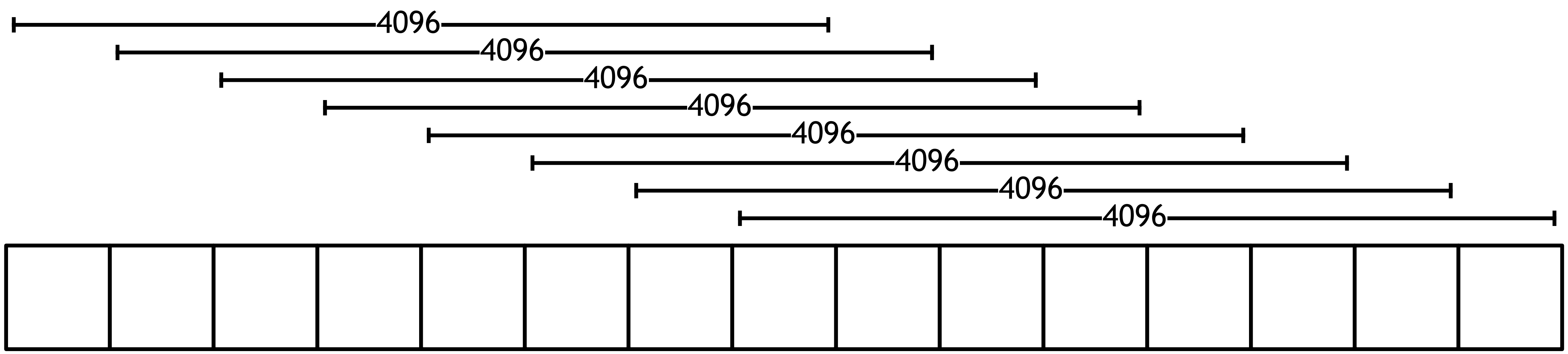
Increase efficiency with larger block size.

We use 4096-byte blocks instead of 512-byte sectors.

- Bloom Filter utilization is ⅛; we can hold 8x the number of hashes!
- Takes the same amount of time to read 4096 bytes as to read 512 bytes
- Most file fragments are larger than 4096 bytes.

But file systems do not align on 4096-byte boundaries!

- We read 15 512-byte blocks.
- Then we compute 8 different 4096-byte block hashes.
- Each one is checked in the Bloom Filter

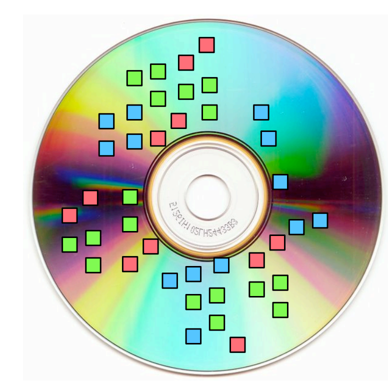


(We can read 64K and trade off I/O speed for CPU speed.)

With this approach, we can scan a 1TB hard drive for 100MB of objectionable material in 2-5 minutes.

		
Minutes	208	5
Max Data Read	1 TB	24 GB

We lower the chance of missing the data to p<0.001



```
^V^W^X^Y^Z%&'()*456789:CDEFGHIJSTUVWXY
:exif='http://ns.adobe.com/exif/1.0/'>
```

```
New York, September 2008^M\223Security
Metrics: What can you test?\224, Veri
fy 2007 International Software Testing
Conference, Arlington, Virginia, Octo
ber 2007.^M\223Attacks and Countermeas
```

Part 2:
Fragment Type Discrimination

File fragment identification is a well-studied problem.

```
^V^W^X^Y^Z%&'()*456789:CDEFGHIJSTUVWXY
:exif='http://ns.adobe.com/exif/1.0/'>
```

This fragment from a file is highly suggestive of a JPEG.

Prior academic work has stressed machine learning.

Algorithm:

- Collect *training* data.
- Extract a feature. (Typically byte-frequency distribution or n-grams.)
- Build a machine learning classifier. (KNN, SVN, etc.)
- Apply classifier to previously unseen *test data.*

Much of this work had problems:

- Many machine learning schemes were actually header/footer recognition.
  - *Well-known n-grams in headers dominated results.*
  - *Some techniques grew less accurate as analyzed more of a file!*
- Container File Complexity:
  - *Doesn't make sense to distinguish PDF from JPEG (if PDFs contain JPEGs.)*

We introduce three advances to this problem.

#1 — Rephrase problem as "discrimination," not recognition.

- Each discriminator reports likely presence or absence of a file type in [BUF]
- Thus, a fragment can be *both* JPEG and ZIP

#2 — Purpose-built functions.

- Develop specific discriminators for specific file types.
- Tune the features with *grid search*.

We've created three discriminators:

- JPEG discriminator
- MP3 discriminator
- Huffman-Coded Discriminator

JPEGs:
Most FFs are followed by 00 due to “byte stuffing.”

Terminal — emacs — 70×27

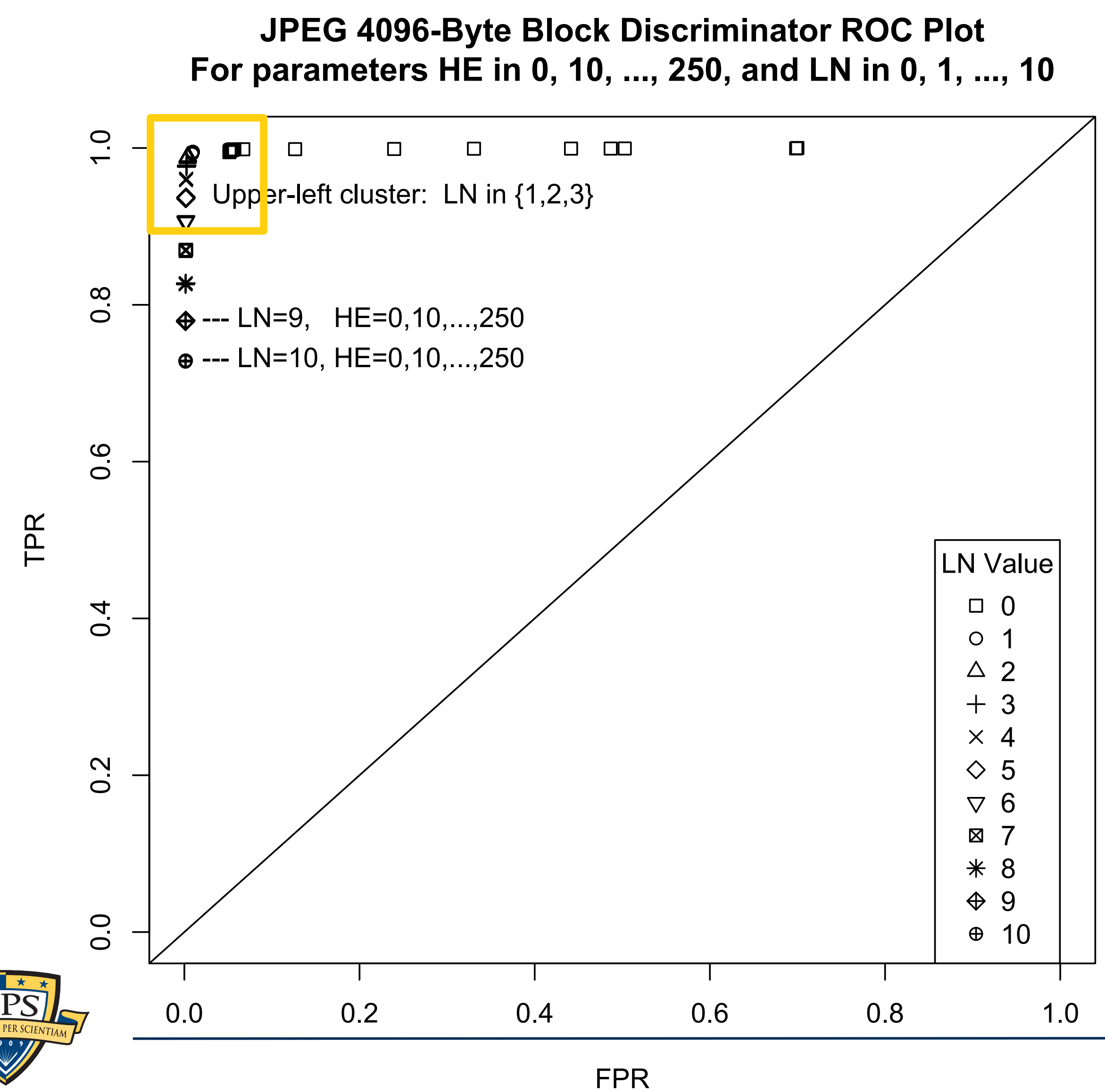
```
87654321  0011 2233 4455 6677 8899 aabb ccdd eeff  0123456789abcdef
00006a20: 6b4c cd62 54a0 b214 52ff 0074 ba4f 4622  kL.bT...R..t.OF"
00006a30: d1bf bf4c 67c4 aa2a 4a91 036f f3b3 7ddc  ...Lg..*J..o..}.
00006a40: 98d5 f078 7f28 d327 340d a2f2 c916 da4f  ...x.(.'4......O
00006a50: aefa 0cbc e9a6 a580 4b20 952c 17d2 7a09  ........K .,..z.
00006a60: 377b 097c 7395 b7e4 c661 730c 447f 9b5a  7{.|s....as.D..Z
00006a70: 7675 e9d1 e14a 81a8 26a2 2948 93bc 4749  vu...J..&.)H..GI
00006a80: 94fd 8d3f fce2 4a13 e529 2b64 8f31 b961  ...?..J..)+d.1.a
00006a90: 368b 827f 677e 7a64 9a62 60f9 9826 c4e0  6...g~zd.b`..&..
00006aa0: b65e bfa9 97fc 5aa9 6a94 626a 602e 4ac7  .^....Z.j.bj`.J.
00006ab0: 9cb1 0311 3d9d 3e33 e941 482e caf2 8676  ....=.>3.AH....v
00006ac0: 240d 43ae ce27 a39e 98d3 f14a 6a23 116a  $.C..'.....Jj#.j
00006ad0: af80 dffc 1867 58be 0eaa a9a9 b29f 3331  .....gX.......31
00006ae0: 20b1 9da6 46d3 eb6d 4846 774c 1870 4c98   ...F..mHFwL.pL.
00006af0: 60fd 0f7d 8382 2f04 e2a9 e314 d982 5947  `..}../.......YG
00006b00: 11ef bef1 7df3 9c6a f0ab 289d 2d99 b6fb  ....}..j..(.-...
00006b10: ff00 9b6d a903 35aa 8b3c 8014 9240 6006  ...m..5..<...@`.
00006b20: cece 5c3b 9f4d af7f 8934 44d8 bd10 4044  ..\;.M...4D...@D
00006b30: 0124 bd6e b80d 61ff 001d 388c 8b74 aaef  .$.n..a...8..t..
00006b40: 32f9 3010 c487 a6fa 681a 4a23 4a8a 5441  2.0.....h.J#J.TA
00006b50: 5b00 3e19 7762 443b 1376 07a1 96c6 5553  [.>.wbD;.v....US
00006b60: 4bbc 285a 7e57 393d e521 e8ce b48a c99a  K.(Z~W9=.!......
00006b70: 69aa 9129 bdab 0361 ba5b 6c36 418d 3e85  i..)...a.[l6A.>.
00006b80: 2c2b 5fc4 55c2 162e 0a60 1209 2144 5887  ,+_.U....`..!DX.
00006b90: 20a4 3055 81c3 a566 799d 84b2 1493 28ac   .0U...fy.....(.
-:---F1  iStock Privacy.jpg    8% L1714   (Hexl)----8:37PM------------
Mark saved where search started
```

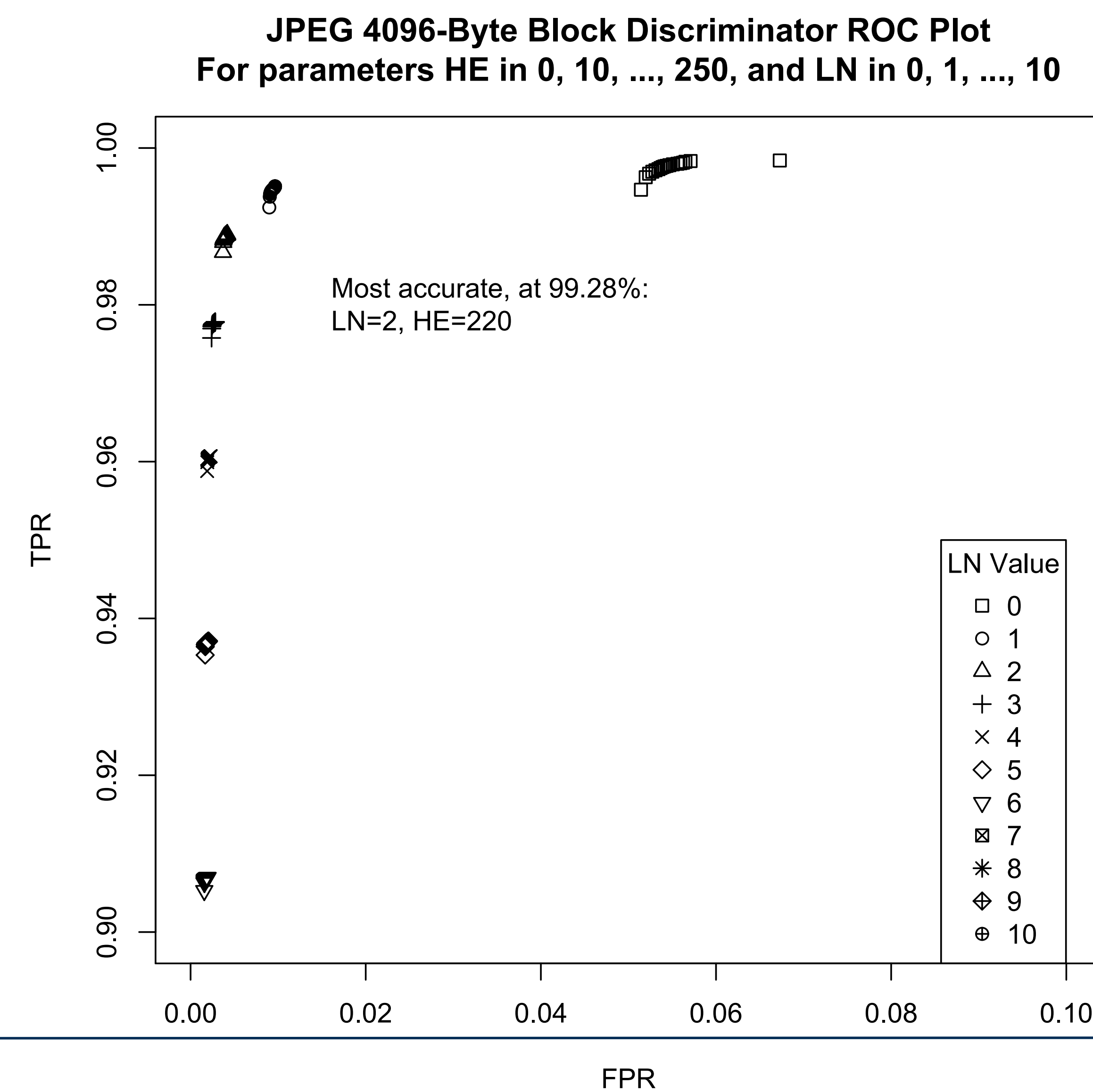
Our JPEG discriminator counts the number of FF00s.

Two tunable parameters:

- High Entropy (HE) - The minimum number of distinct byte values in the 4096-byte buffer.
- Low FF00 N-grams (LN) - The minimum number of <FF><00> byte pairs

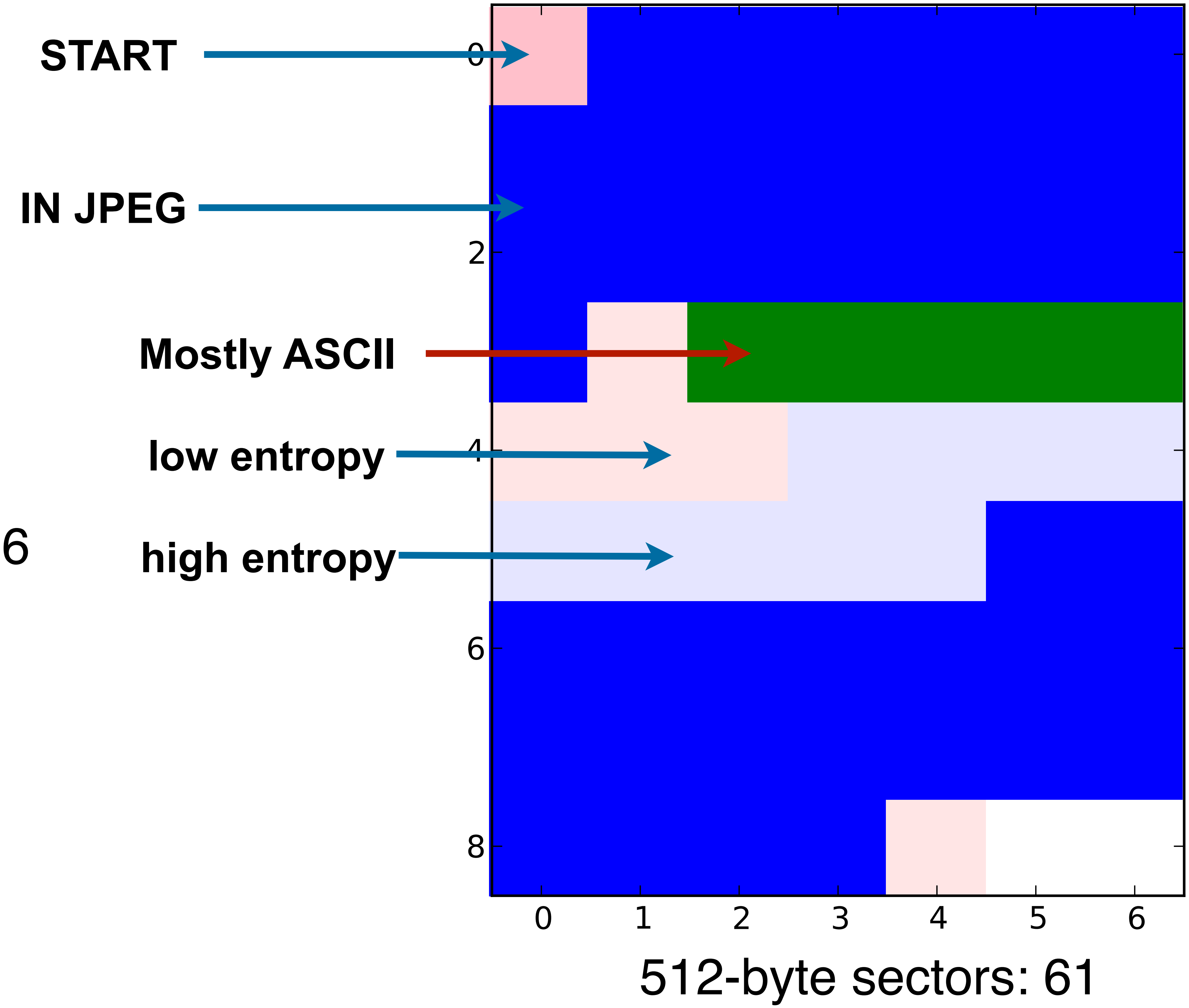
We perform a grid search with a variety of possible values.





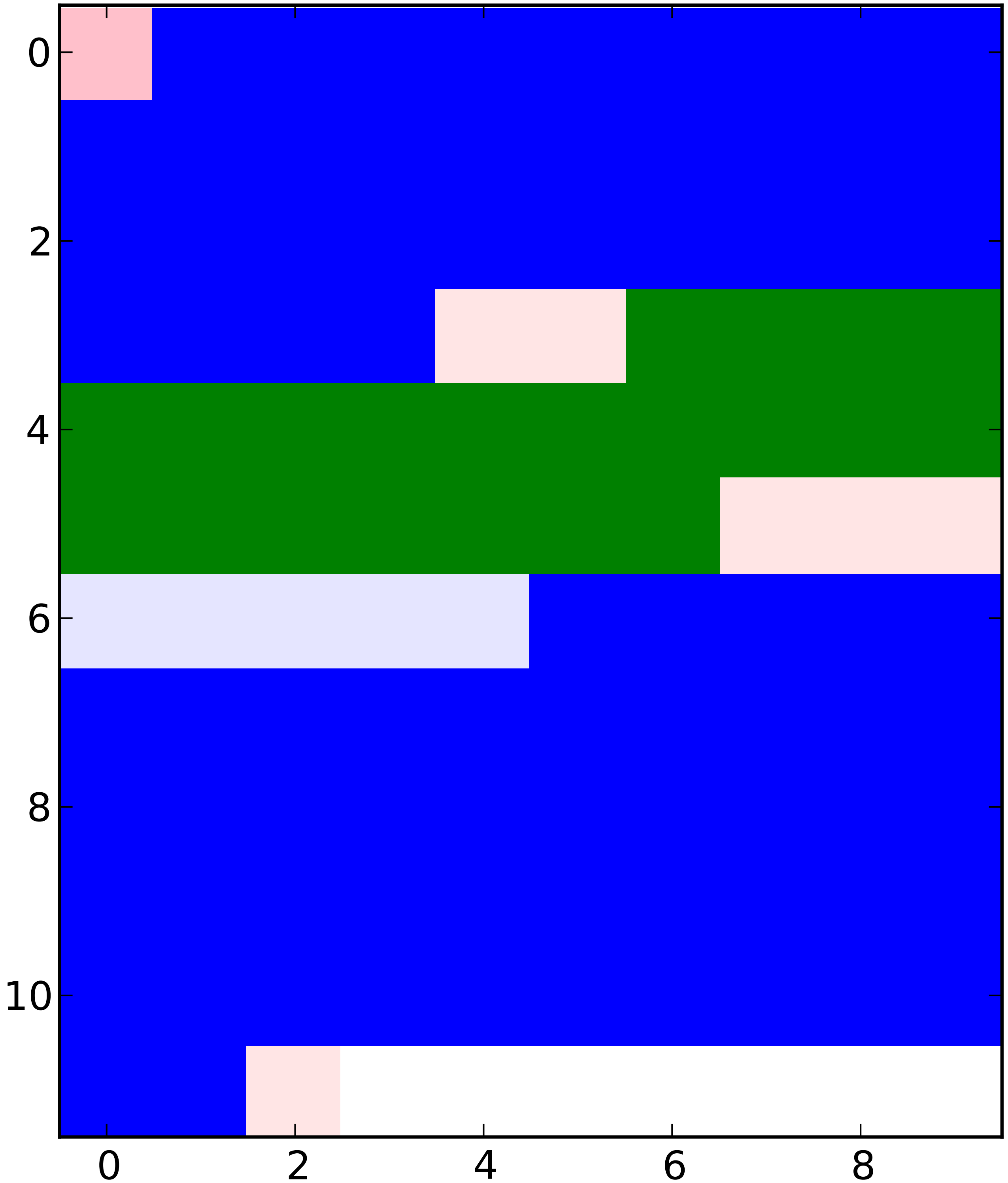
These maps of JPEG blocks show the accuracy.
000109.jpg

Bytes: 31,046



000897.jpg

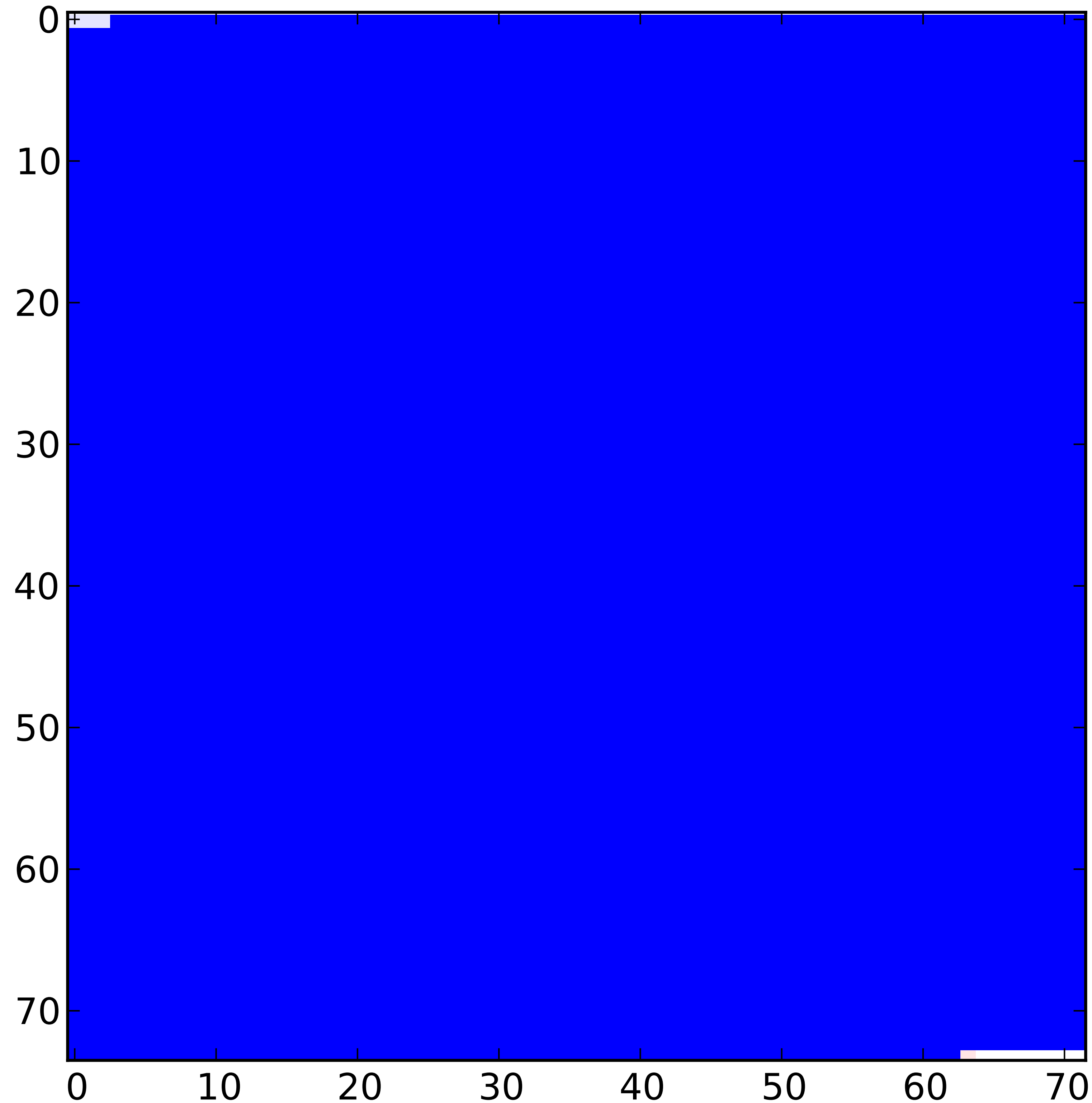
Bytes: 57,596



512-byte Sectors: 113

000888.pdf

Bytes: 2,723,425

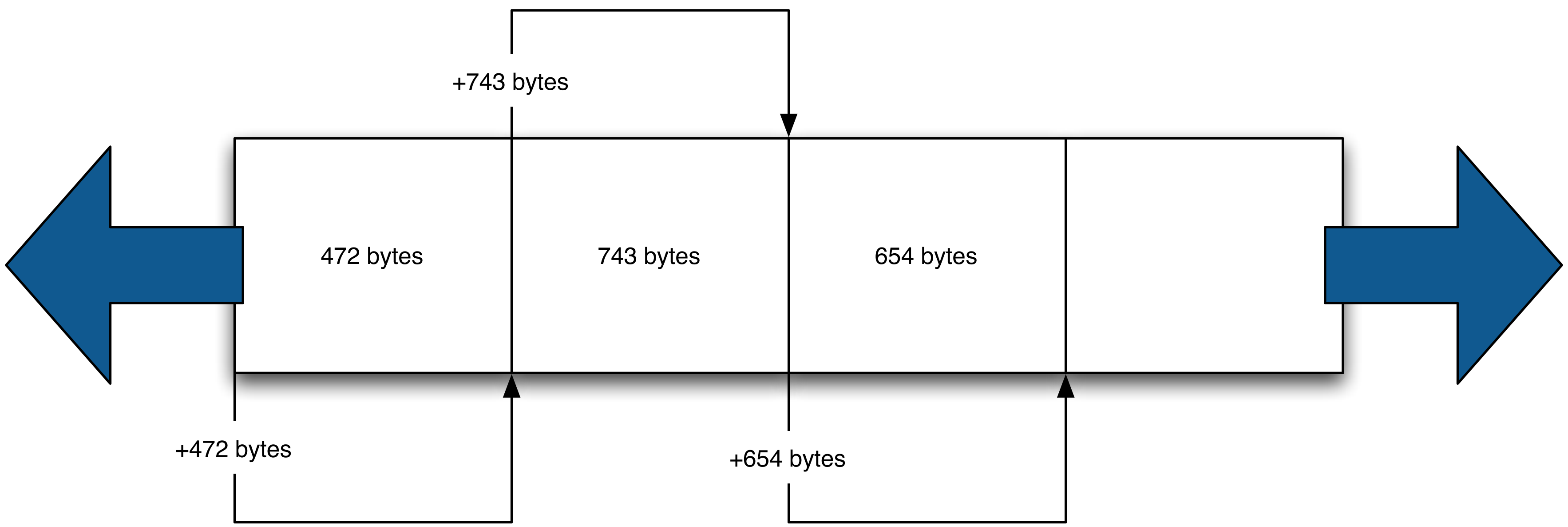


512-byte Sectors: 5320

The MPEG classifier uses the frame chaining approach.

Each frame has a header and a length.

Find a header, read the length, look for the next header.



Our MP3 discriminator:

- Frame header starts with a string of 11 sync bits
- Sanity-check bit rate, sample rate and padding flag.
- FrameSize = 144 x BitRate / (SampleRate + Padding)
- Skip to next Frame and repeat.
- Chain Length (CL) = 4 produced 99.56% accuracy with 4K buffer.

The Huffman-Encoding detector is based on autocorrelation.

Huffman-coding is a variable-length bit-level code.

- Symbols may be any number of bits.
- More frequent symbols are shorter.
- Hard to distinguish from random data.

Hypothesis:

- Common symbols will occasionally line up in successive bytes.

"alan" = 01011001 01000100

- If we perform an *autocorrelation*, common symbols will self-align more often than by chance, producing more 0s:

```
  01011001
⊕ 01000100
==========
  00011101
```

- With random (or encrypted) data, autocorrelation should not significantly change the statistics.

Char	Freq	Code
space	7	111
a	4	010
e	4	000
f	3	1101
h	2	1010
i	2	1000
m	2	0111
n	2	0010
s	2	1011
t	2	0110
l	1	11001
o	1	00110
p	1	10011
r	1	11000

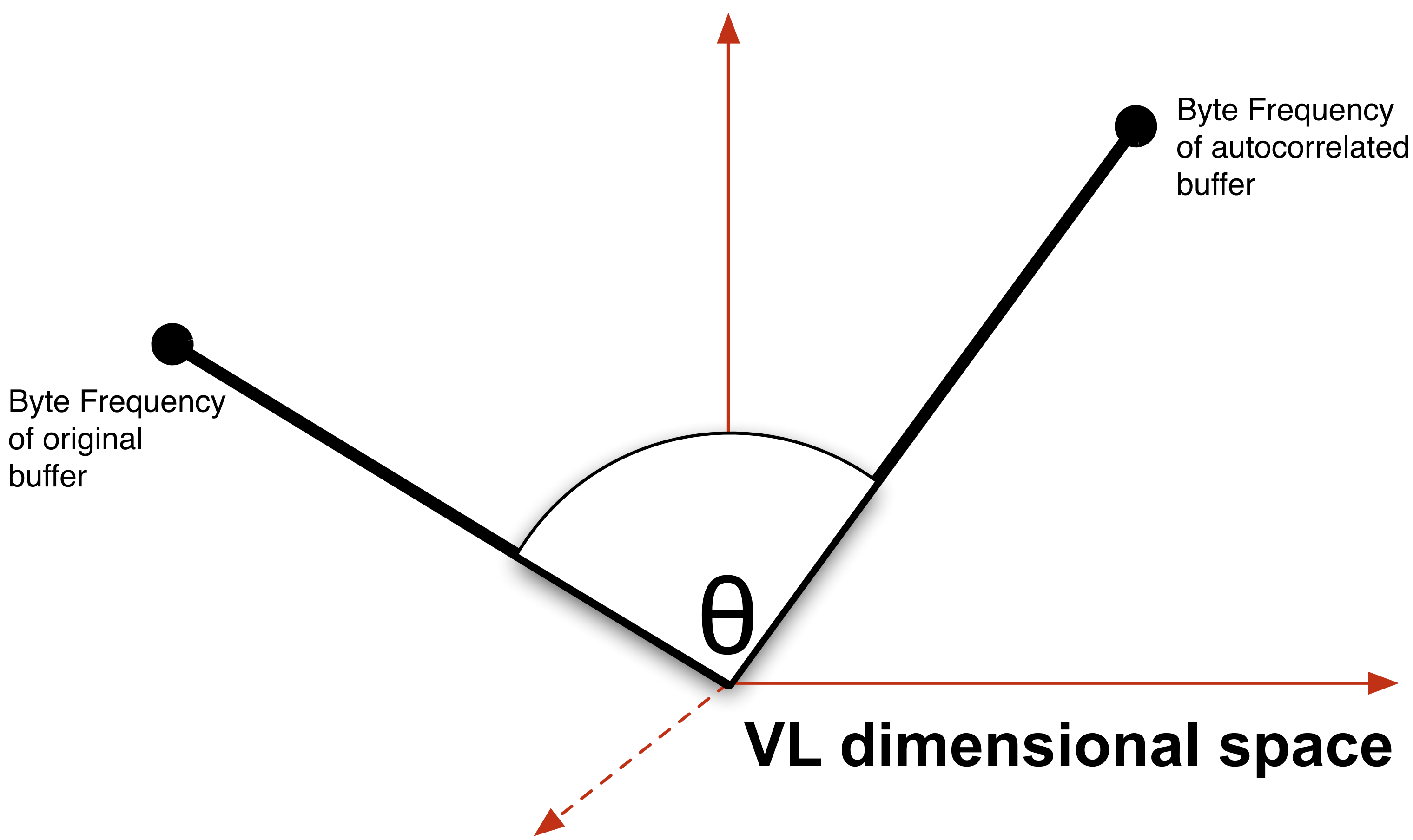
Our approach computes the cosine similarity of the byte-frequency distribution in multi-dimensional space

Two tunable parameters:

- VL - Vector Length - The number of dimensions to consider (this is VL=3)
- MCV - Minimum Cosine Value - if cos(θ) < MCV, data is deemed to be Huffman.

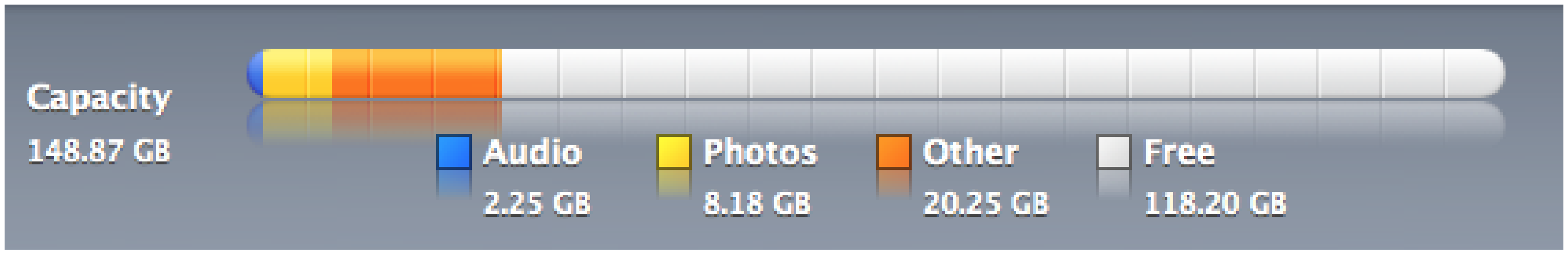
Best Results:

- 16KiB-block discriminator:
- 66.6% accurate,
- TPR 48.0%,
- FPR 0.450%.
- VL=250,
- MCV=0.9996391245556134.



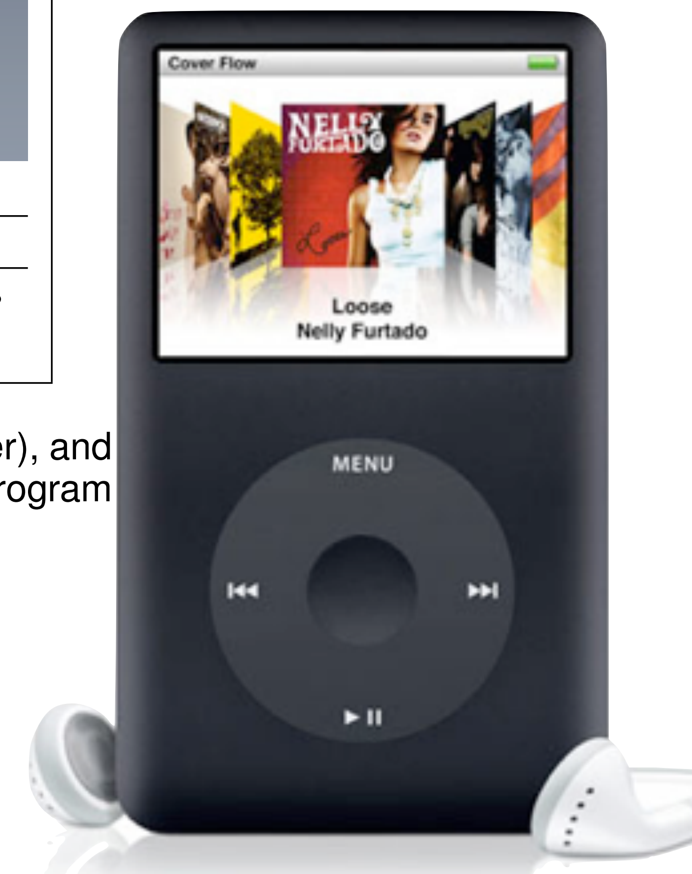
Combine random sampling with sector discrimination to obtain the forensic contents of a storage device.

Our numbers from sampling are similar to those 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 could accurately determine:

- Amount of free space
- Amount of JPEG
- Amount of MPEG