NPS Tools for Automated Media Exploitation History and Current Projects

Simson L. Garfinkel
Associate Professor, Naval Postgraduate School

Sept 21, 2011,

http://simson.net/

NPS is the Navy's Research University.

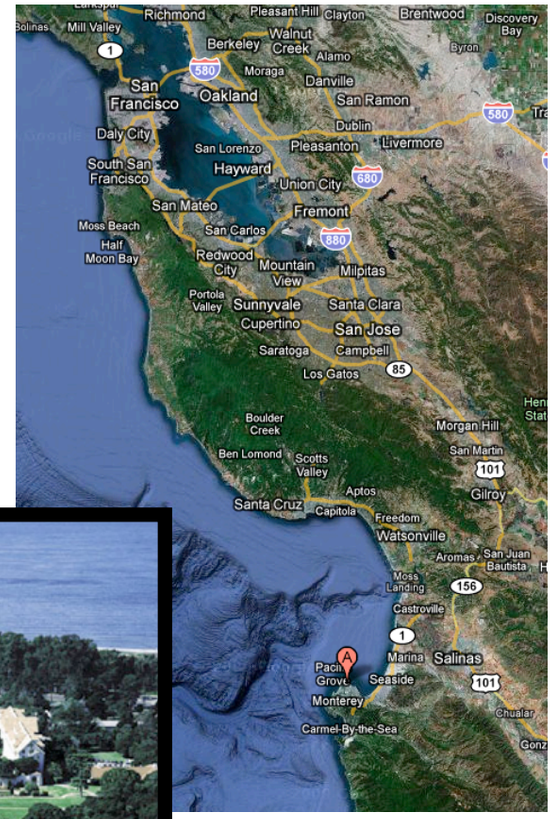
Location: Monterey, CA

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

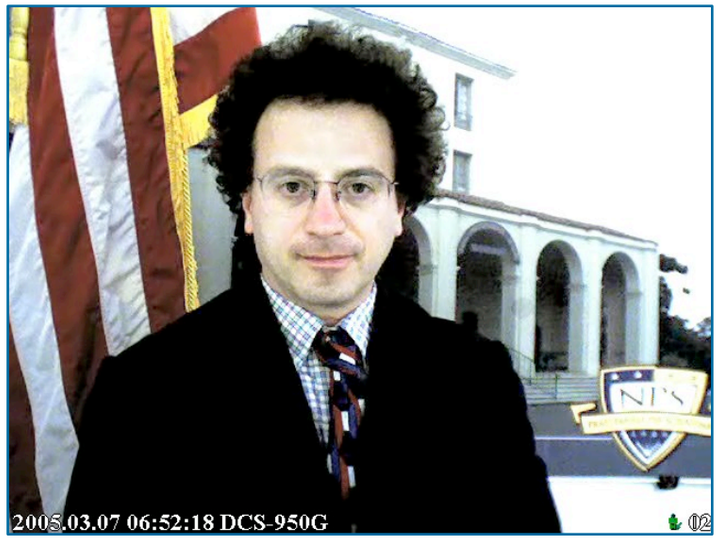


NCR Initiative:

- 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

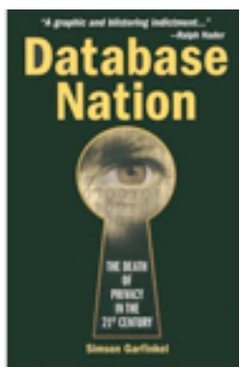
2010	PCS to National Capital Region
2006-	Joined NPS Faculty
2005-2006	Harvard University postdoc
2002-2005	MIT EECS PhD Program
1988-2004	Entrepreneur & Journalist

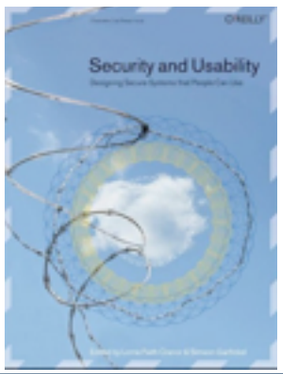


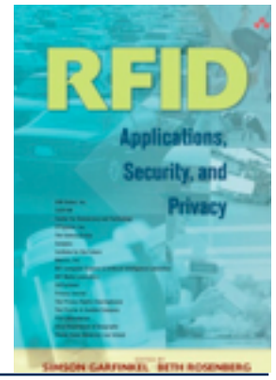
- Vineyard.NET, Broadband2Wireless,
- *Sandstorm Enterprises, Inc. (network forensics)*
- *Technology Review Magazine*
- *Chief Security Officer (CSO) Magazine (4 national awards)*
- *Boston Globe Columnist, 1997-2002*

1988-2011 Author & Inventor

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







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.

This talk presents tools-you-can-use for digital forensics… … and some tools that are currently under development.

Tool #1: bulk_extractor

- high-speed triage tool for all kinds of data

Tool #2: Smirk

- Facebook reconstruction tool
- Currently prototype; development funded through FY12

Tool #3: frag_find

- Hash-based carver
- Experimental but usable today; full “tool” by March 2012.

Tool #4: Random sampler.

- Completely research
- Hope to have a tool developed by end of FY12

Tool #1: bulk_extractor

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

0 → 1TB

3 hours, 20 min to *read* the data

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.

Primary Advantage: Speed

No disk seeking.

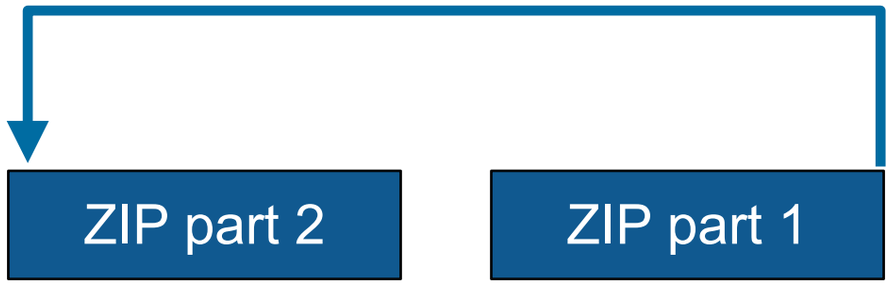
Potential to read and process at disk's maximum transfer rate.

Potential for intermediate answers.

Reads all the data — allocated files, deleted files, file fragments.

- Separate metadata extraction required to get the file names.

Primary Disadvantage: Completeness



Fragmented files won't be recovered:

- Compressed files with part2-part1 ordering (possibly .docx)
- Files with internal fragmentation (.doc but not .docx)

Fortunately, most files are *not* fragmented.

- Individual components of a ZIP file can be fragmented.

Most files that *are* fragmented have carvable internal structure:

- Log files, Outlook PST files, etc.

San Luis Obispo
Sincerely, California.

http://www.sanluisobispovacations.com/

A bulk_extractor Success Story

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.*

- Typically, it would take several days to conduct a proper forensic investigation.

bulk_extractor found actionable evidence in 2.5 hours!

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

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.

Faster than conventional tools.
Finds data that other tools miss.

Runs 2-10 times faster than EnCase or FTK *on the same hardware.*

- bulk_extractor is multi-threaded; EnCase 6.x and FTK 3.x have little threading.

Finds details invisible to other tools

- “Optimistically” decompresses and re-analyzes all data.
- Scans data in browser caches (downloaded with zip/gzip), even when deleted.
- Scans fragments of hibernation files.

Presents the data in an easy-to-understand report.

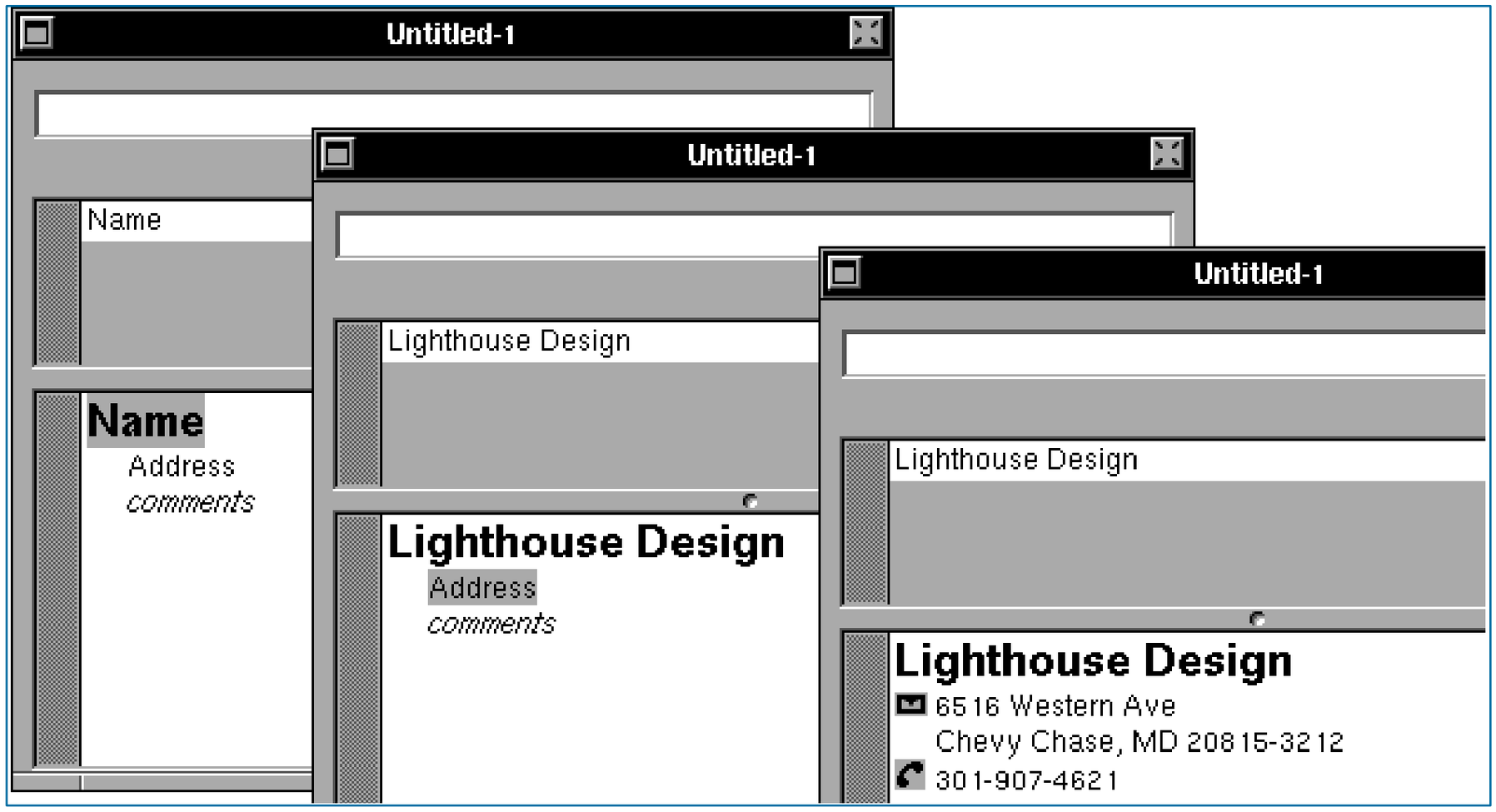
- Produces “histogram” of email addresses, credit card numbers, etc.
- Distinguishes primary user from incidental users.

Feeds data to cross-drive analysis tools.

History of bulk_extractor

bulk_extractor: 20 years in the making!

In 1991 I developed SBook, a free-format address book.

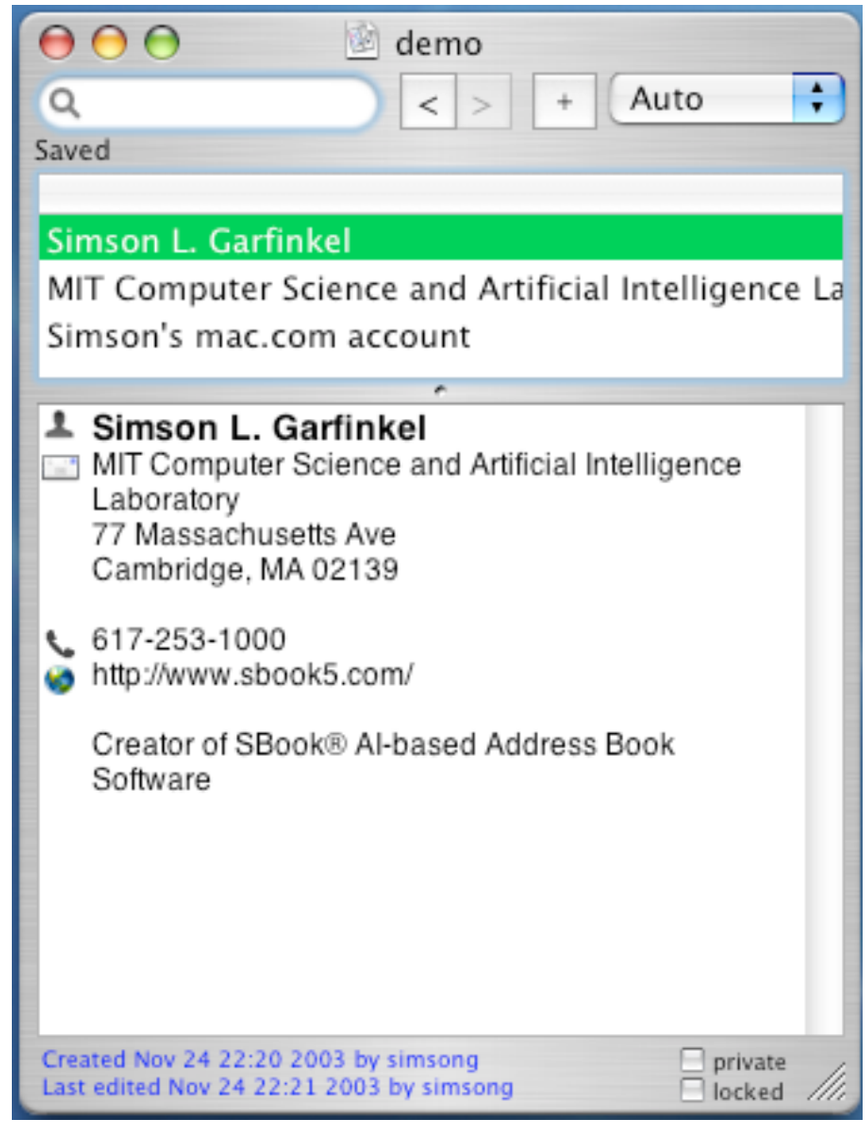


SBook used “Named Entity Recognition” to find addresses, phone numbers, email addresses *while you typed.*

Today we call this technology Named Entity Recognition

SBook's technology was based on:

- Regular expressions executed in parallel
  - —*US, European, & Asian Phone Numbers*
  - —*Email Addresses*
  - —*URLs*
- A gazette with more than 10,000 names:
  - —*Common "Company" names*
  - —*Common "Person" names*
  - —*Every country, state, and major US city*
- Hand-tuned weights and additional rules.



Implementation:

- 2500 lines of GNU flex, C++
- 50 msec to evaluate 20 lines of ASCII text.
  - —*Running on a 25Mhz 68030 with 32MB of RAM!*

In 2003, I bought 200 used hard drives

The goal was to find drives that had not been properly sanitized.

First strategy:

- DD all of the disks to image files
- run **strings** to extract printable strings.
- **grep** to scan for email, CCN, etc.
  - *VERY SLOW!!!!*
  - *HARD TO MODIFY!*

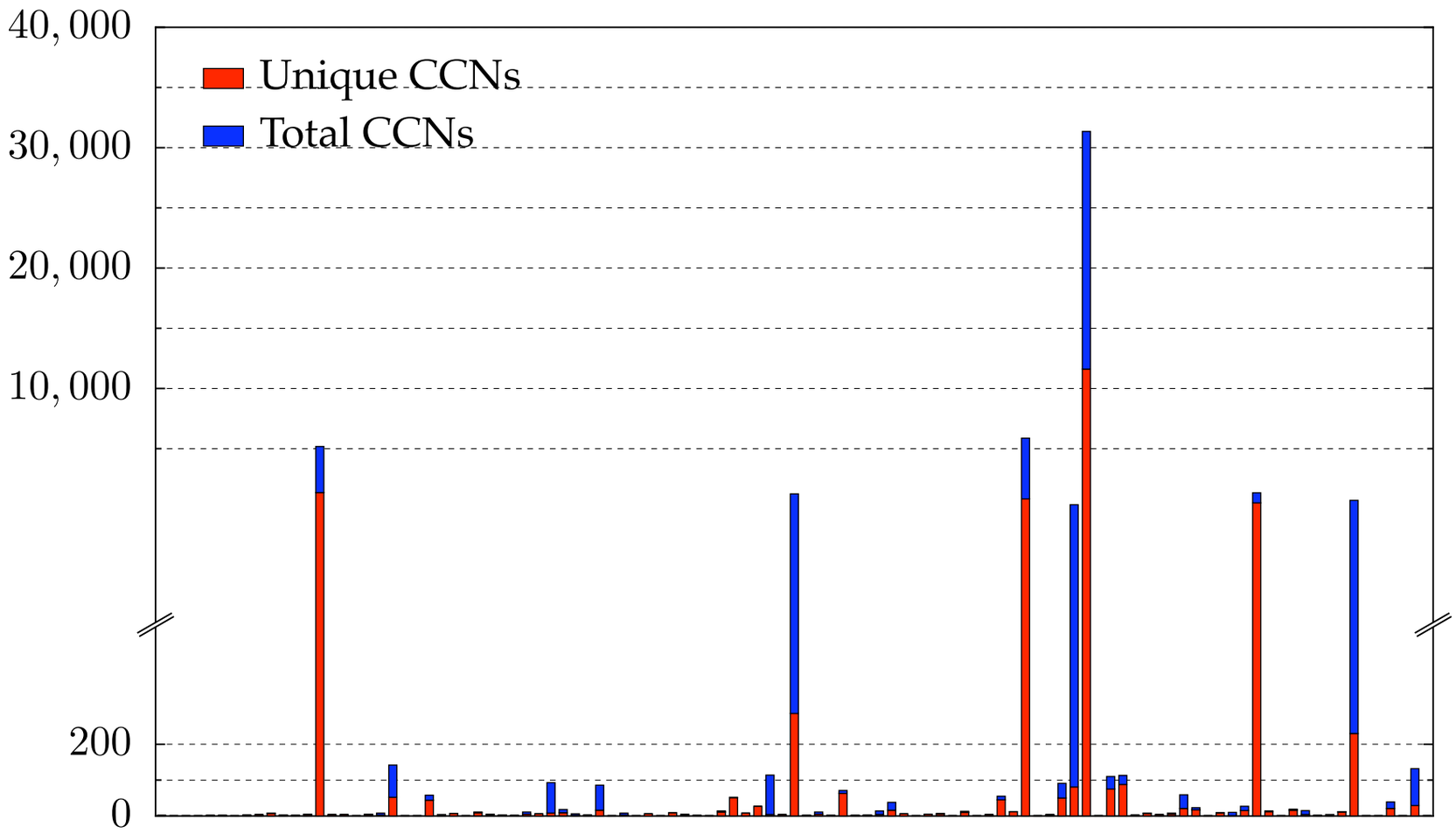
Second strategy:

- Use SBook technology!
- Read disk 1MB at a time
- Pass the *raw disk sectors* to flex-based scanner.
- Big surprise: scanner didn't crash!

Simple flex-based scanners required substantial post-processing to be useful

Techniques include:

- Additional validation beyond regular expressions (CCN Luhn algorithm, etc).
- Examination of feature “neighborhood” to eliminate common false positives.



The technique worked well to find drives with sensitive information.

Between 2005 and 2008, I interviewed law enforcement examiners regarding their use of forensic tools.

LE wanted a *highly automated* tool for finding:

- Email addresses
- Credit card numbers (including track 2 information)
- Google Search terms (extracted from URLs)
- Phone numbers
- GPS coordinates
- EXIF information from JPEGs
- All words that were present on the disk (for password cracking)

The tool had to:

- Run on Windows, Linux, and Mac-based systems
- Run with *no* user interaction
- Operate on raw disk images, split-raw volumes, E01 files, and AFF files
- Allow user to provide additional regular expressions for searches
- Automatically extract features from compressed data such as gzip-compressed HTTP
- Run at maximum I/O speed of physical drive
- Never crash

Starting in 2008, we made a series of limited releases.

- January 2008 — Created Subversion Repository
- April 2010 — Initial public release - 0.1.0
- May 2010 — Initial multi-threading release - 0.3.0
  - —*Each thread runs in its own process*
- Sept. 2010 — Stop lists - 0.4.0
- Oct. 2010 — Context-based stop-lists - 0.5.0
- Dec. 2010 — Switch to POSIX-based threads — 0.6.0
- Dec. 2010 — Support for WIndows HIBERFIL.SYS decompression — 0.7.0
- Jun. 2010 — First 1.0.0 Release (TODAY)

Tool capabilities result from substantial testing and user feedback.

Moving technology from the lab to the field has been challenging:

- 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.

Inside bulk_extractor

bulk_extractor: architectural overview

Written in C, C++ and GNU flex

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

Key Features:

- “Scanners” look for information of interest in typical investigations.
- Recursively re-analyzes compressed data.
- Results stored in “feature files”
- Multi-threaded

Java GUI

- Runs command-line tool and views results

bulk_extractor extracts “features” from disk images.

http://www.nps.edu/

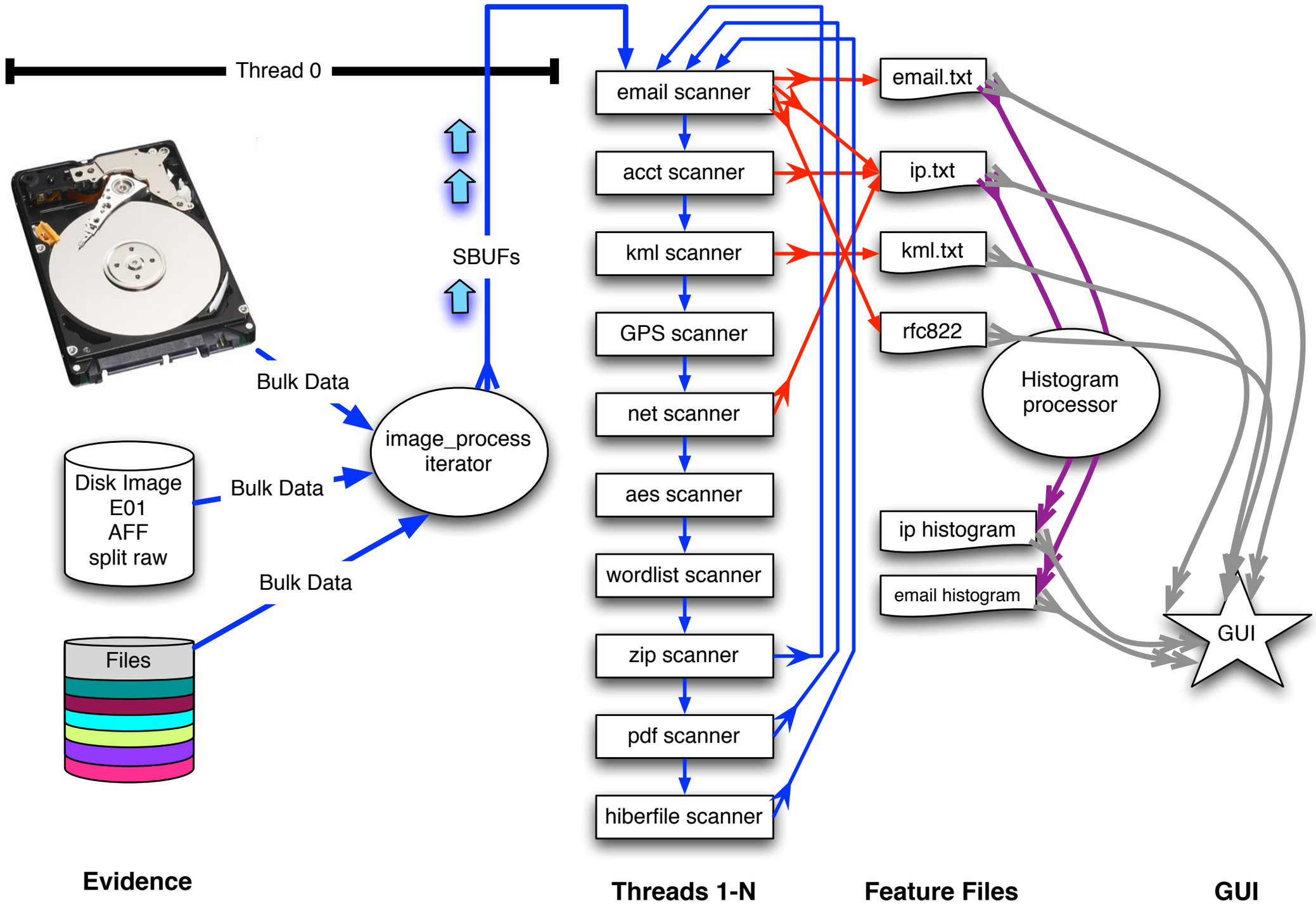
202-555-1212
user@domain.com

202-555-1212

http://www.nps.edu/

user@domain.com

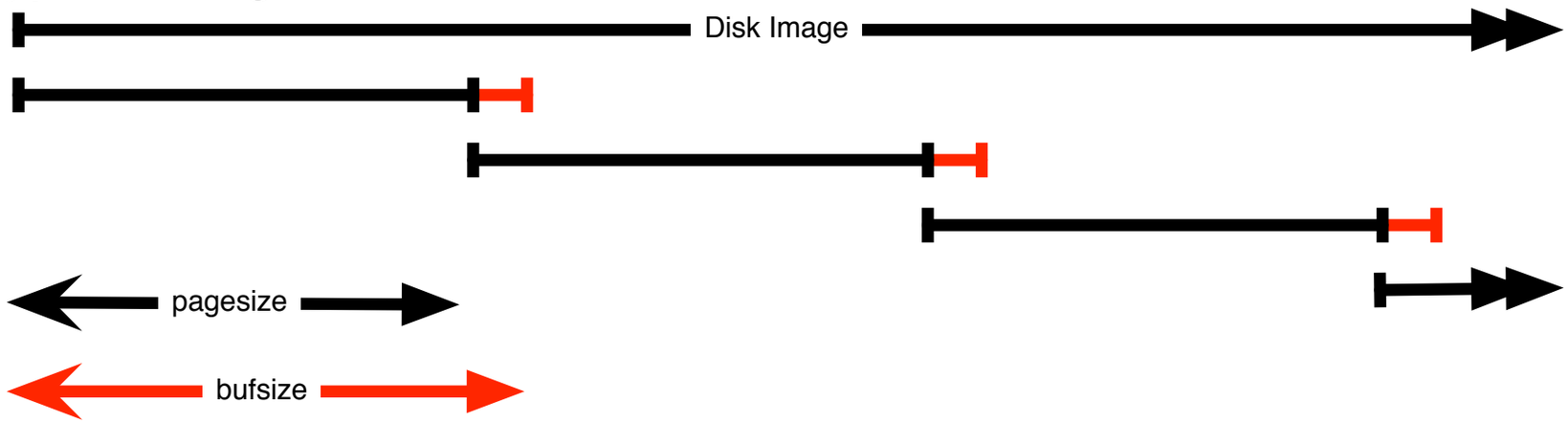
bulk_extractor: system diagram



The “pages” overlap to avoid dropping features that cross buffer boundaries.

The overlap area is called the *margin*.

- Each buffer is processed in parallel — they don’t depend on each other.
- Features start in the page but end in the margin are *reported*.
- Features that start in the margin are *ignored* (we get them later)
  - *Assumes that the feature size is smaller than the margin size.*
  - *Typical margin: 1MB*



Entire system is automatic:

- Image_process iterator makes **sbuf_t** buffers.
- Each buffer is processed by every scanner
- Features are automatically combined.

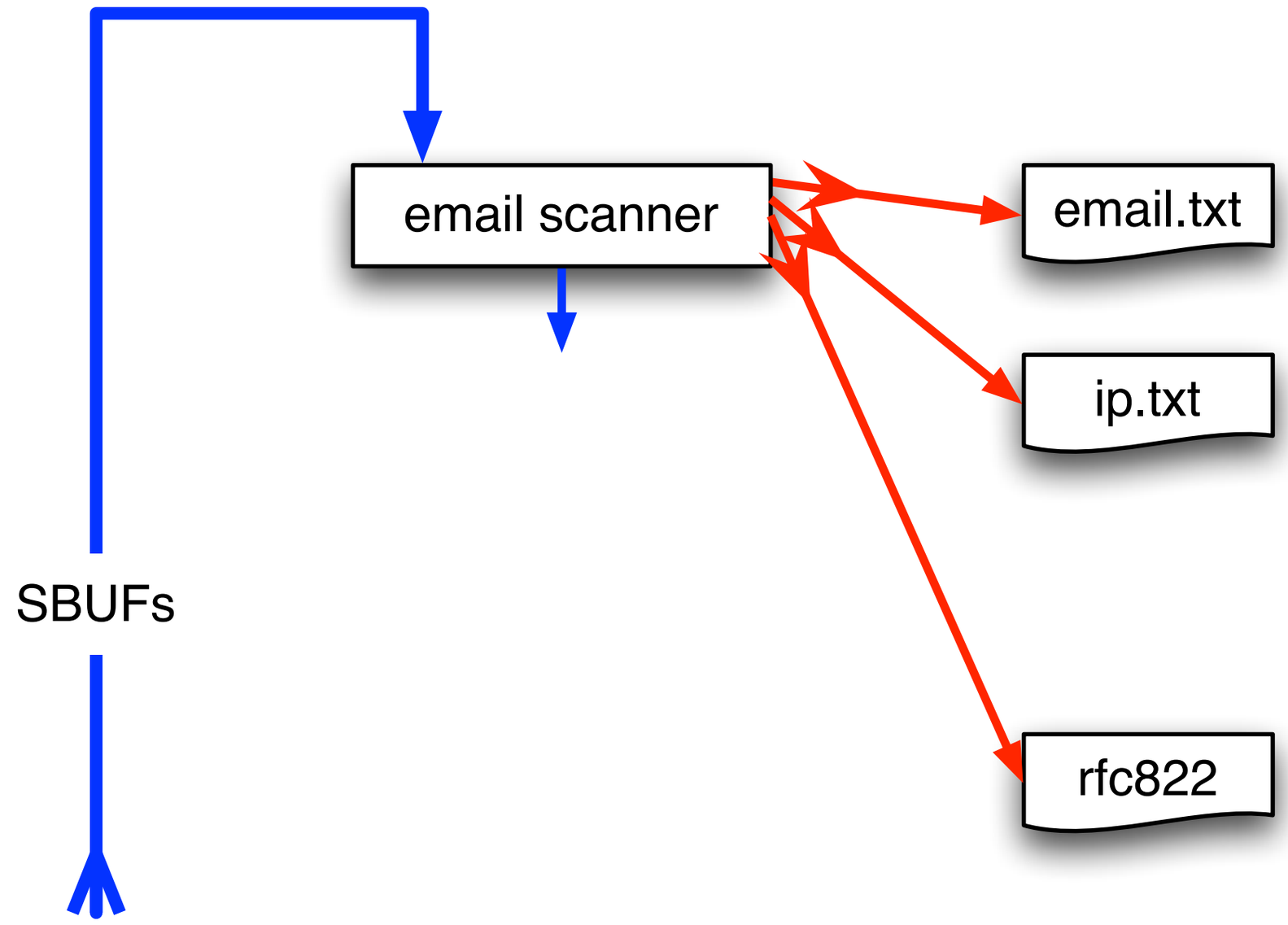
Scanners process an sbuf and extract features

scan_email is the email scanner.

- inputs: **sbuf** objects

outputs:

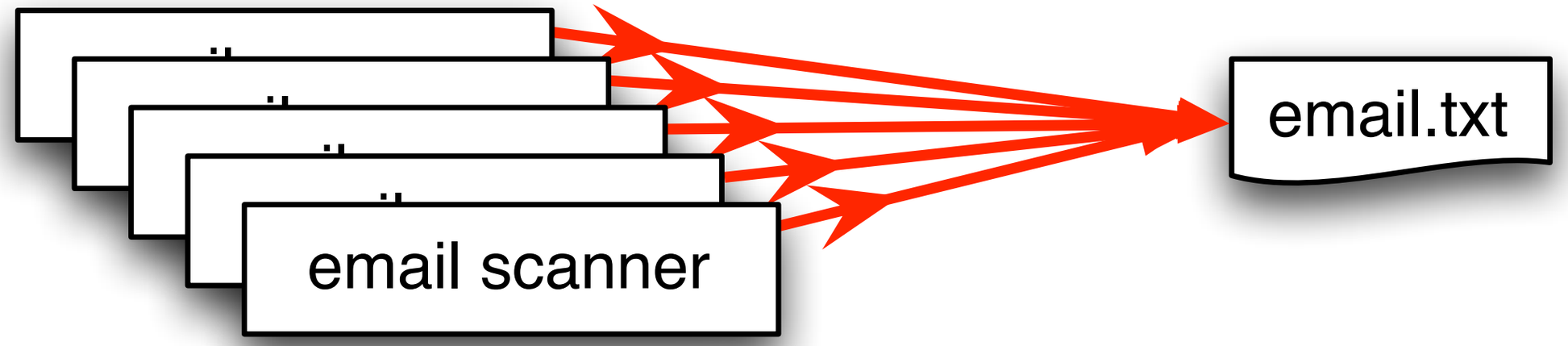
- **email.txt**
  - —*Email addresses*
- **rfc822.txt**
  - —*Message-ID*
  - —*Date:*
  - —*Subject:*
  - —*Cookie:*
  - —*Host:*
- **domain.txt**
  - —*IP addresses*
  - —*host names*



The *feature recording system* saves features to disk.

Feature Recorders store the features.

- Scanners are given a (feature_recorder *) pointer
- Feature recorders are *thread safe*.



Features are stored in a *feature file:*

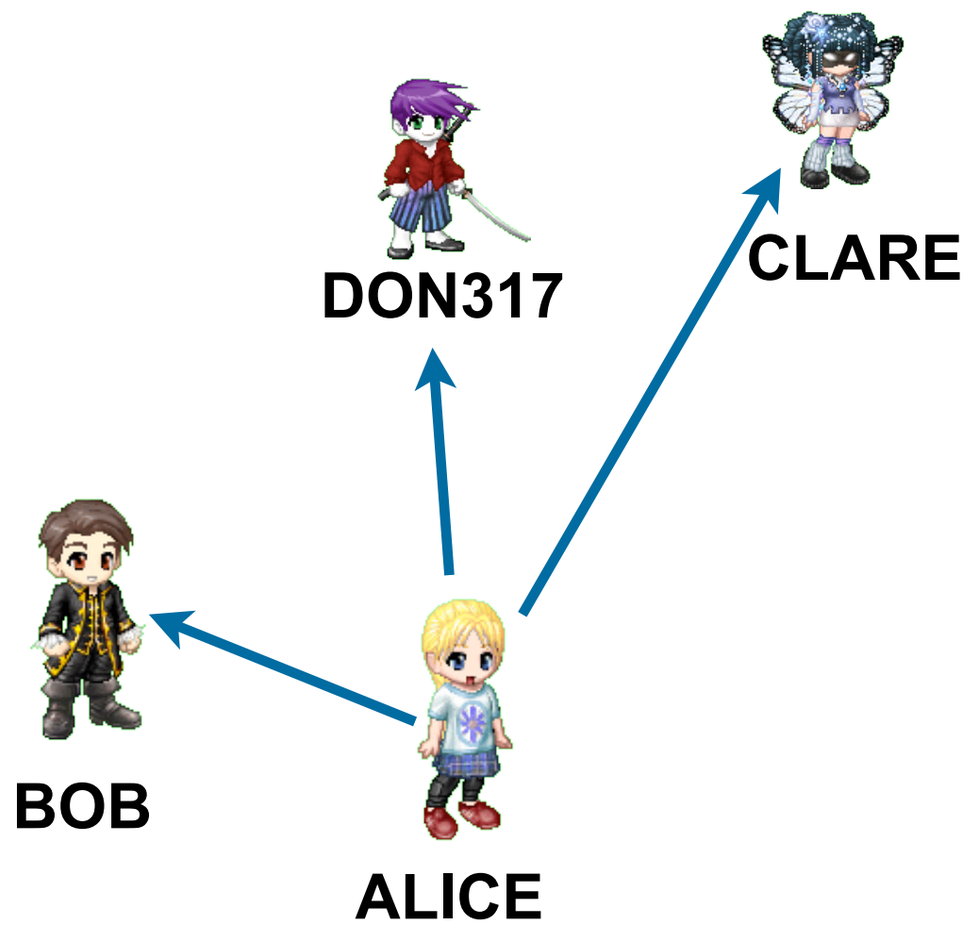
```
48198832  domexuser2@gmail.com    tocol>____<name>domexuser2@gmail.com/Home</name>____
48200361  domexuser2@live.com     tocol>____<name>domexuser2@live.com</name>____<pass
48413829  siege@preoccupied.net   siege) O'Brien <siege@preoccupied.net>_hp://meanwhi
48481542  danilo@gnome.org        Danilo __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("
```

offset **feature** **feature in evidence context**

Histograms are a powerful tool for understanding evidence.

Email histogram allows us to rapidly determine:

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



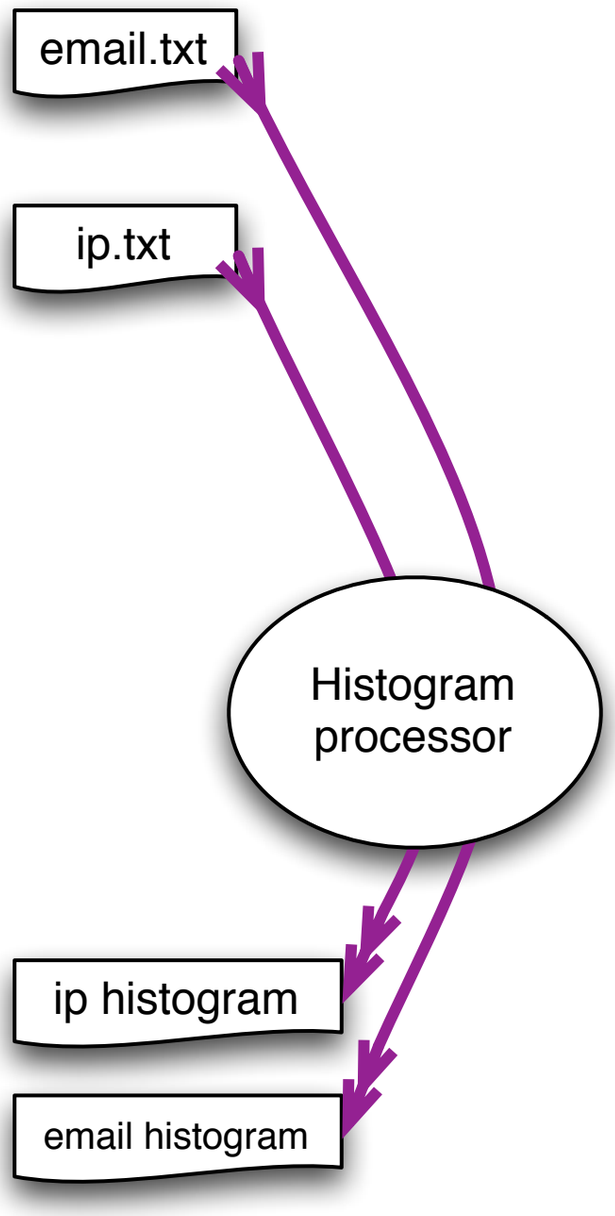
Drive #51 (Anonymized)

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

The feature recording system *automatically* makes histograms.

Simple histogram based on feature:

```
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
```



Based on regular expression extraction:

- For example, extract search terms with **.*search.*q=(.*)**

```
n=18     pidgin
n=10     hotmail+thunderbird
n=3      Grey+Gardens+cousins
n=3      dvd
n=2      %TERMS%
n=2      cache:
n=2      p
n=2      pi
n=2      pid
n=1      Abolish+income+tax
n=1      Brad+and+Angelina+nanny+help
n=1      Build+Windmill
n=1      Carol+Alt
```

bulk_extractor has *multiple* feature extractors. Each scanner runs in order. (Order doesn't matter.)

Scanners can be turned on or off

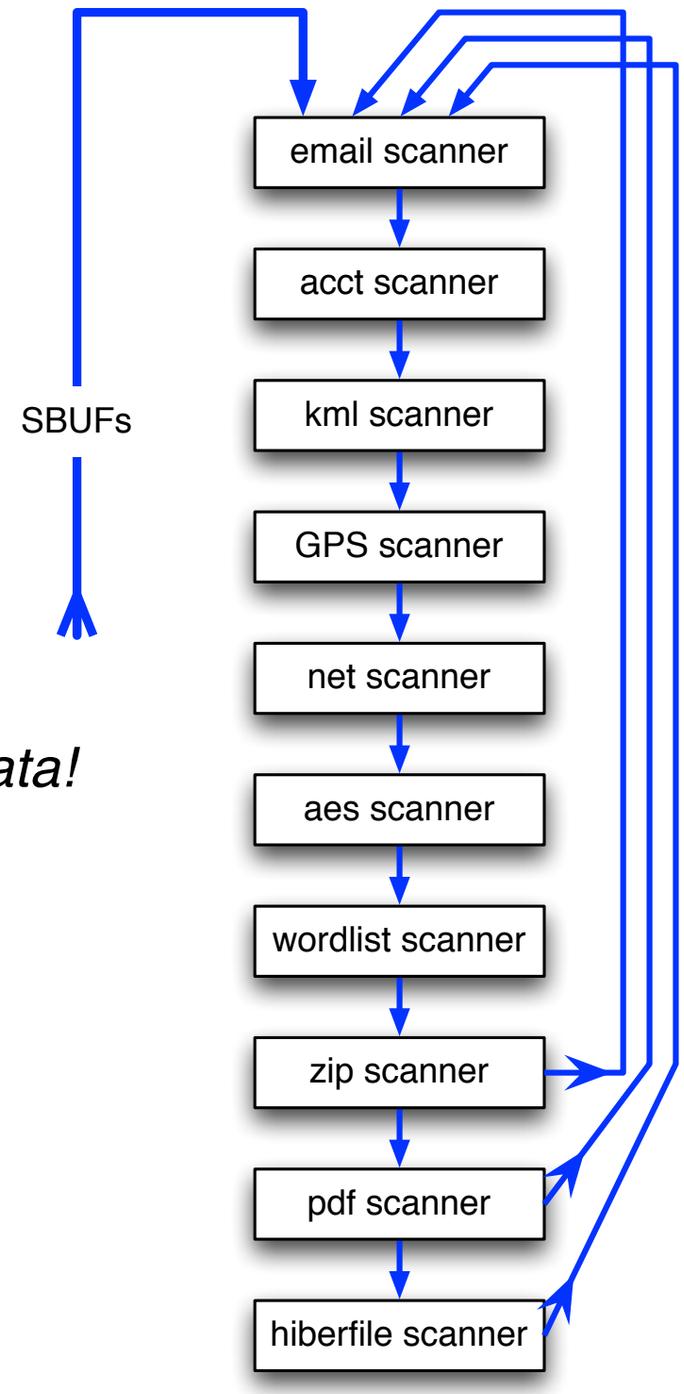
- Useful for debugging.
- AES key scanner is *very slow* (off by default)

Some scanners are *recursive.*

- *e.g.* scan_zip will find zlib-compressed regions
- An **sbuf** is made for the decompressed data
- The data is re-analyzed by the other scanners
  - *This finds email addresses in compressed data!*

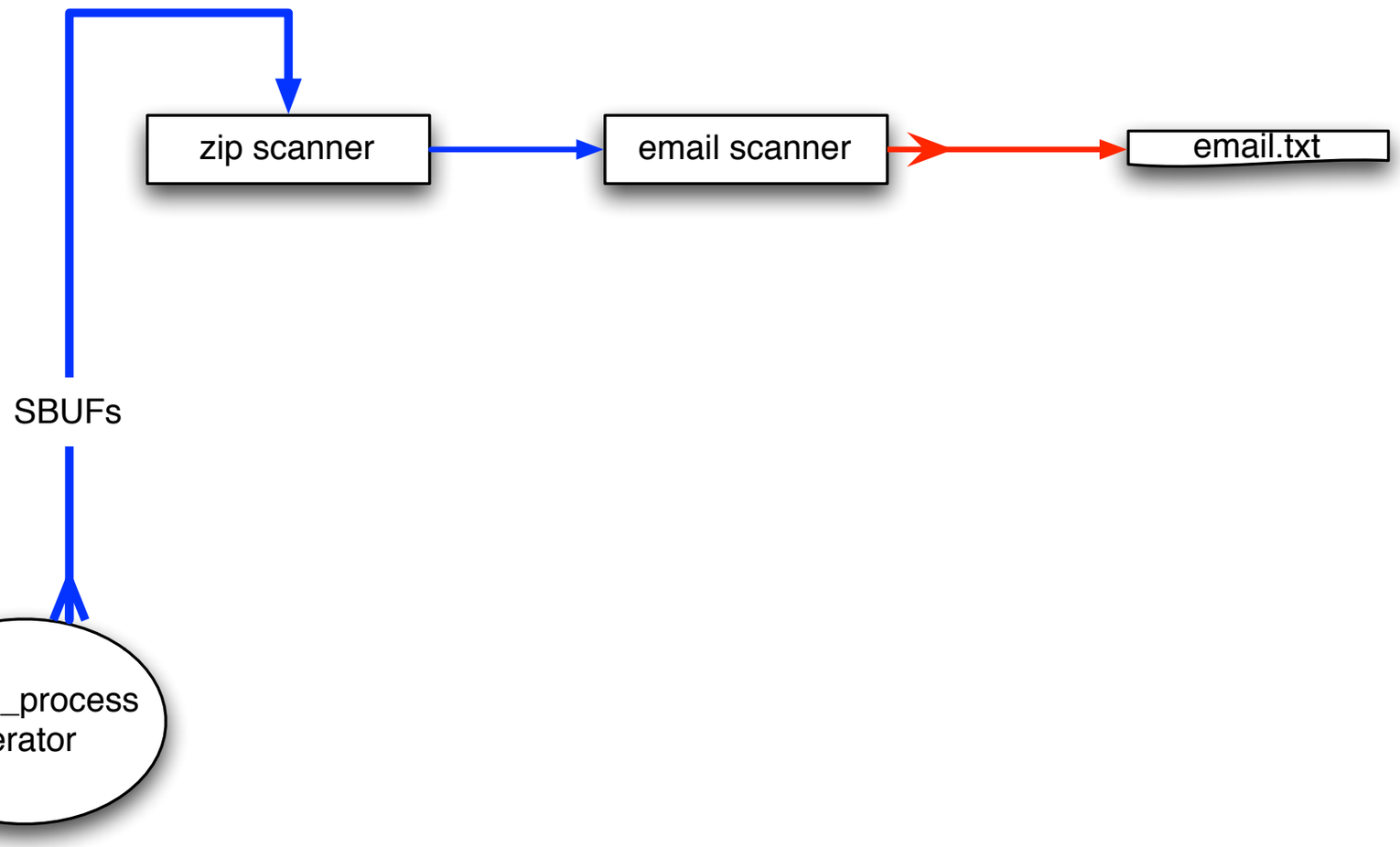
Recursion used for:

- Decompressing ZLIB, Windows HIBERFILE,
- Extracting text from PDFs
- Handling compressed browser cache data



Recursion requires a *new way* to describe offsets. bulk_extractor introduces the "forensic path."

Consider an HTTP stream that contains a GZIP-compressed email:

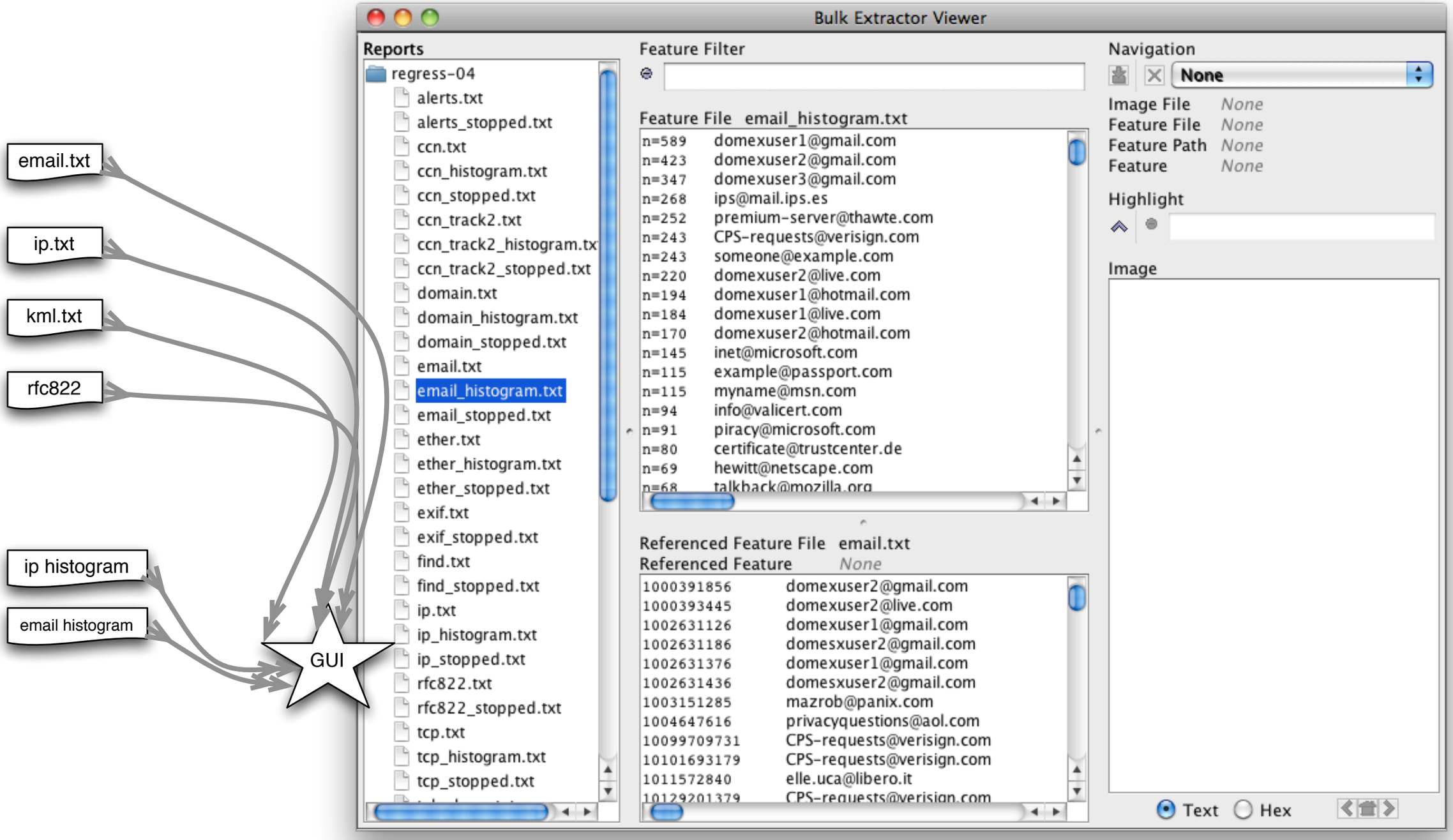


This is represented as:

```
1052168704-GZIP-3437    live.com    eMn='domexuser1@live.com';var srf_sDispM
1052168704-GZIP-3475    live.com    pMn='domexuser1@live.com';var srf_sPreCk
1052168704-GZIP-3512    live.com    eCk='domexuser1@live.com';var srf_sFT='<
```

GUI: 100% Java
Launches bulk_extractor; views results

Uses bulk_extractor to decode forensic path



Crash protection is provided with check pointing.

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.

Checkpointing — automatic restarting.

- 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 scanners

bulk_extractor 1.1 has 11 simple scanners

scan_accts — scans for “accounts”

- credit card numbers (validated), CCN Track 2 data, phone numbers, BitLocker recovery;

scan_aes — AES key files in swap and hibernation files

scan_email

- email addresses; URLs; domain names; IP addresses; Ethernet MAC addresses

scan_exif — EXIF records from JPEG files

scan_find — user-specified keyword scanner

scan_gps — XML records from Garmin GPS devices

scan_json — JSON from web pages

scan_kml — KML (Google Maps)

scan_net — IP packets; reconstructs pcap files

scan_winprefetch — Windows prefetch files

scan_wordlist — Constructs a word list for password cracking

bulk_extractor 1.1 has 4 recursive scanners

scan_base64 — decodes BASE64 encoded data

scan_hiberfile — decompresses Windows Hibernation files

scan_pdf — extracts text from PDF files

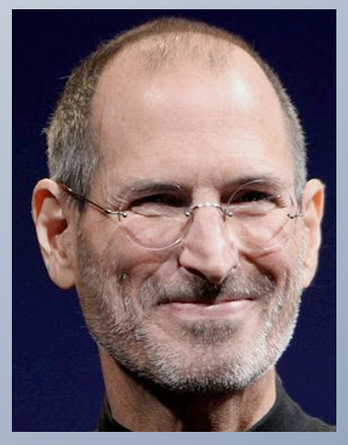
scan_zip — unzips ZIP files and components

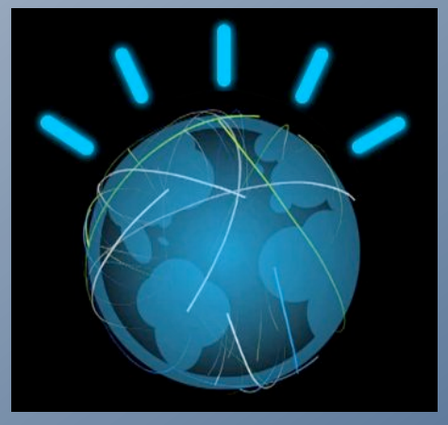
The Prefetch Scanner tells you some programs that have been run

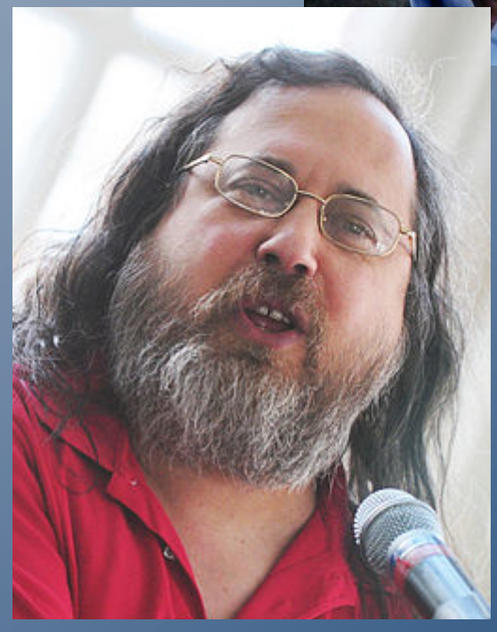
```
<prefetch>
  <header>
    <os>Windows Vista or Windows 7</os>
    <header_size>240</header_size>
    <filename>IEXPLORE.EXE</filename>
    <runs>53</runs>
    <atime>2011-02-07T13:03:24</atime>
  </header>
  <volume>
    <path>\DEVICE\HARDDISKVOLUME1</path>
    <serial_number>b46f6927</serial_number>
  </volume>
  <creation>2010-08-18T06:13:10</creation>
  <filenames>
    <file>\DEVICE\HARDDISKVOLUME1\WINDOWS\SYSTEM32\NTDLL.DLL</file>
    <file>\DEVICE\HARDDISKVOLUME1\WINDOWS\SYSTEM32\KERNEL32.DLL</file>
  </filenames>
  <dirnames>
    <dir>\DEVICE\HARDDISKVOLUME1\PROGRAM FILES</dir>
    <dir>\DEVICE\HARDDISKVOLUME1\PROGRAM FILES\COMMON FILES</dir>
    <dir>\DEVICE\HARDDISKVOLUME1\PROGRAM FILES\JAVA</dir>
    <dir>\DEVICE\HARDDISKVOLUME1\PROGRAM FILES\JAVA\JRE6</dir>
  </dirnames>
</prefetch>
```

Will recover deleted prefetch files.

Sample output







Suppressing False Positives

Modern operating systems are *filled* with email addresses.

Sources:

- 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 *stop list* commonly seen emails.

- Operating Systems have a LOT of emails. (FC12 has 20,584!)
- Problem: this approach gives Linux developers a free pass!

Approach #3: Context-sensitive stop list.

Instead of a stop list of features, use features+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 put the feature+context in the stop list.*

The “Stop List” entry is the feature+context.

- This ignores Linux developer email address in Linux binaries.
- The email address is reported if it appears in a different context.

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)

Sample from the stop list:

```
tzigkeit  <gord@gnu.ai.mit.edu>___* tests/demo          sl3/fedora12-64/domain.txt
tzigkeit  <gord@gnu.ai.mit.edu>___Reported by           sl3/fedora12-64/domain.txt
u-emacs-request@prep.ai.mit.edu (or the corresp         sl3/redhat54-ent-64/domain.txt
u:/pub/rtfm/" "/ftp@rtfm.mit.edu:/pub/usenet/" "        sl3/redhat54-ent-64/email.txt
ub/rtfm/" "/ftp@rtfm.mit.edu:/pub/usenet/" "            sl3/redhat54-ent-64/domain.txt
udson  <ghudson@mit.edu>',_    "lefty"                  sl3/redhat54-ent-64/domain.txt
ug-fortran-mode@erl.mit.edu__This list coll             sl3/redhat54-ent-64/domain.txt
uke Mewburn <lm@rmit.edu.au>, 931222_AC_ARG             sl3/fedora12-64/domain.txt
um _ *          kit@expo.lcs.mit.edu_ */_#ifndef _As    sl3/redhat54-ent-64/email.txt
um _ *          kit@expo.lcs.mit.edu_ */__#ifndef _A    sl3/redhat54-ent-64/email.txt
um _ *          kit@expo.lcs.mit.edu_ */__#ifndef _S    sl3/redhat54-ent-64/email.txt
```

The context-sensitive stop list prunes the OS-supplied features.

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

talkback@mozilla.org and other email addresses were *not* eliminated because they were present on the base OS installs.

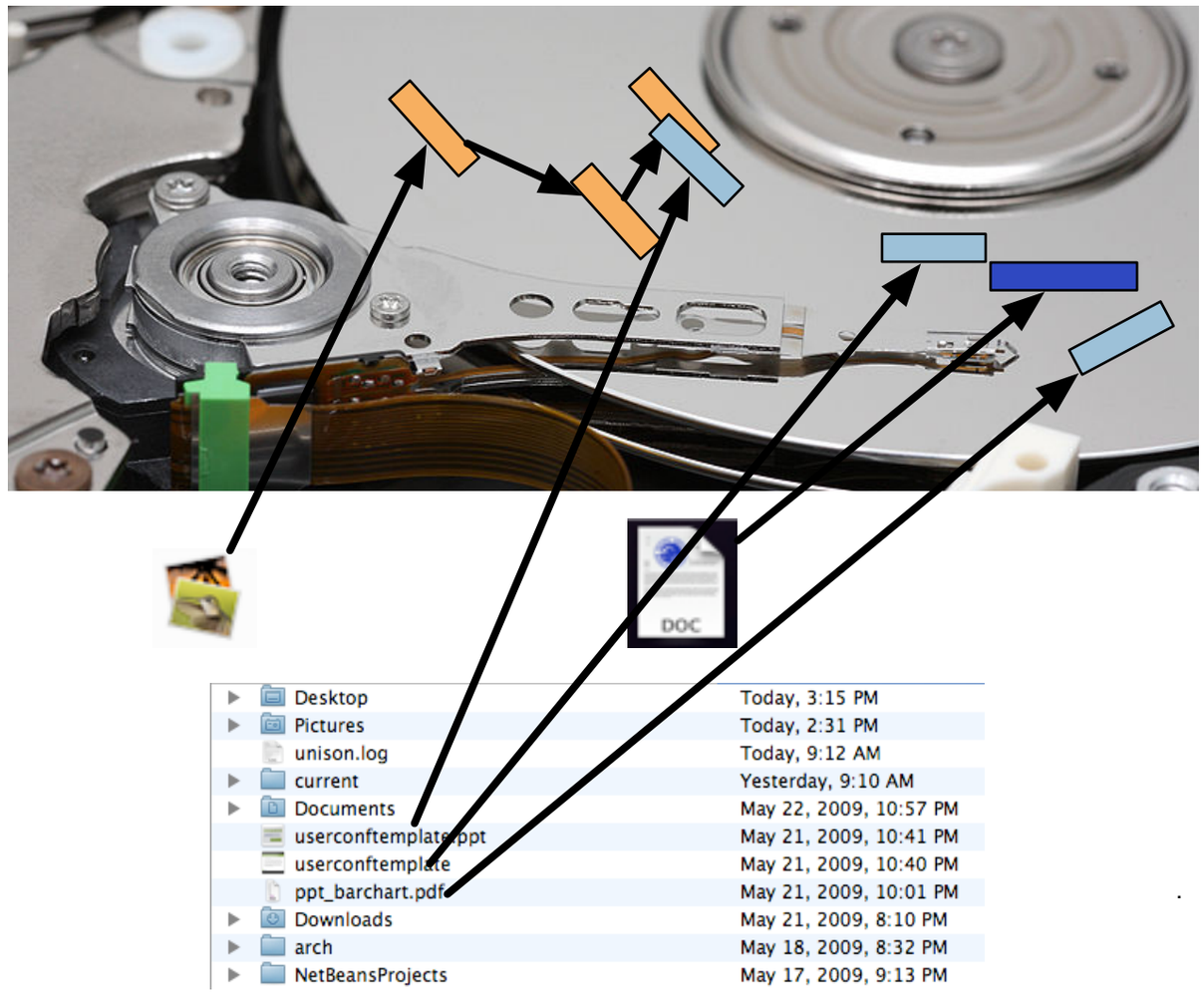
You can download the list today:

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

Extending bulk_extractor with Plug-ins

Filenames can be added through post-processing.

bulk_extractor reports the *disk blocks* for each feature.



To get the file names, you need to map the disk block to a file.

- Make a map of the blocks in DFXML with **fiwalk** (http://afflib.org/fiwalk)
- Then use **python/identify_filenames.py** to create an *annotated feature file.*

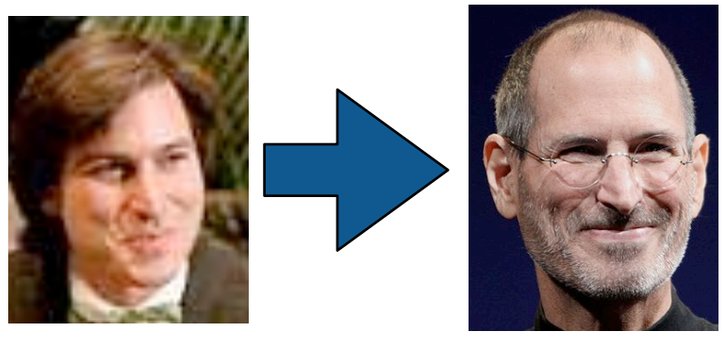
bulk_diff.py: compare two different bulk_extractor reports

The “report” directory contains:

- DFXML file of bulk_extractor run information
- Multiple feature files.

bulk_diff.py: create a “difference report” of two bulk_extractor runs.

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

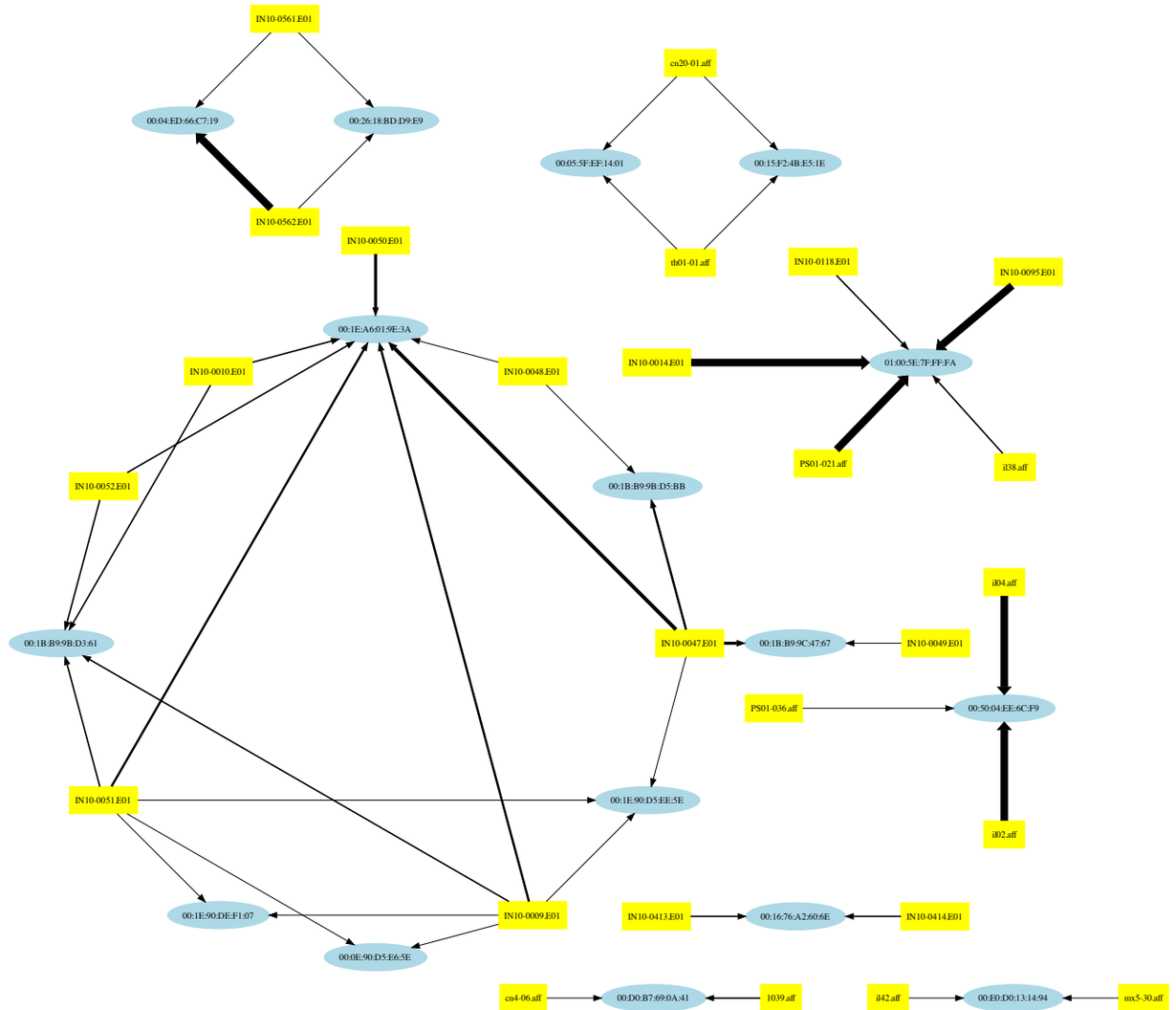


IP Carving and Network Reassembly plug-in

bulk_extractor extended to recognize and validate network data.

- Automated extraction of Ethernet MAC addresses from *IP packets in hibernation files.*

We then re-create the physical networks the computers were on:



C++ programmers can write C++ plugins

Plugins are distributed as *shared libraries*.

- Windows: **scan_bulk.DLL**
- Mac & Linux: **scan_bulk.so**

Plugins must support a single function call:

```
void scan_bulk(const class scanner_params &sp,
               const recursion_control_block &rcb)
```

- scanner_params — Describes what the scanner should do.
  - —*sp.sbuf* — *SBUF to scan*
  - —*sp.fs* — *Feature recording set to use*
  - —*sp.phase==0* — *initialize*
  - —*sp.phase==1* — *scan the SBUF in sp.sbuf*
  - —*sp.phase==2* — *shut down*
- recursion_control_block — Provides information for recursive calls.

The same plug in system will be used by a future version of **fiwalk**.

- The same plug-in will be usable with multiple forensic tools.

bulk_extractor is an open source program!
You can help make it better.

Better handling of text:

- MIME decoding (e.g. user=40localhost should be user@localhost)
- Improved handling of Unicode.

More scanners

- RAR & RAR2
- LZMA
- BZIP2
- MSI & CAB
- NTFS
- VCARD

Reliability and conformance testing.

GET PAID TO WORK ON BULK_EXTRACTOR: ASK ME HOW!

In conclusion, bulk_extractor is a powerful stream-based forensic tool.

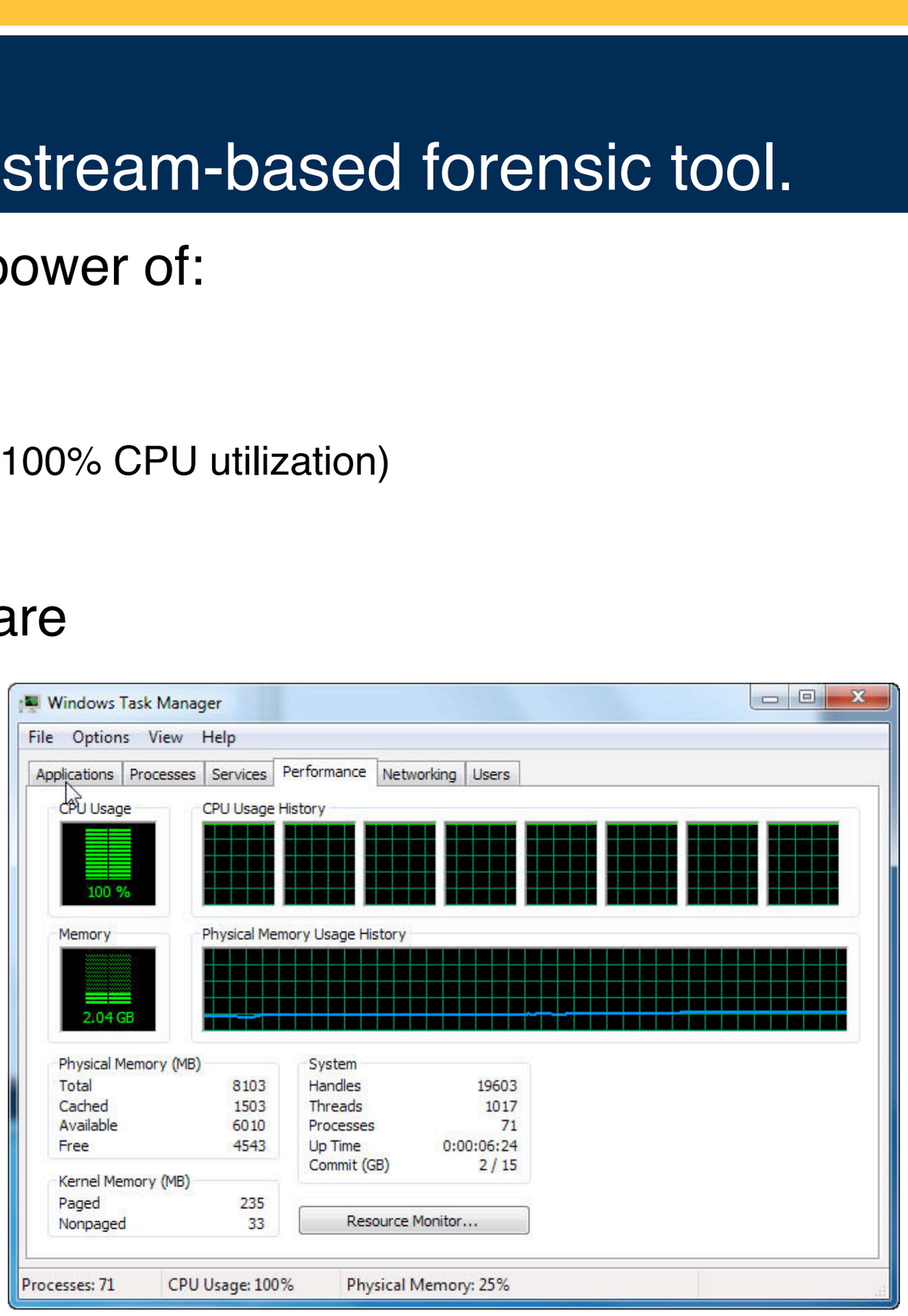
Bulk_extractor demonstrates the power of:

- Bulk data processing.
- Carving EVERYTHING
- Multi-threading (we can process data with 100% CPU utilization)

Bulk_extractor is 100% free software

- Public Domain (work of US Government)
- Please use the ideas in other programs!
  - *DFXML*
  - *Job Distribution*
  - *Forensic Path*
  - *SBUF*
- Let's keep the plug-in system consistent.
- Download from http://afflib.org/

Questions?

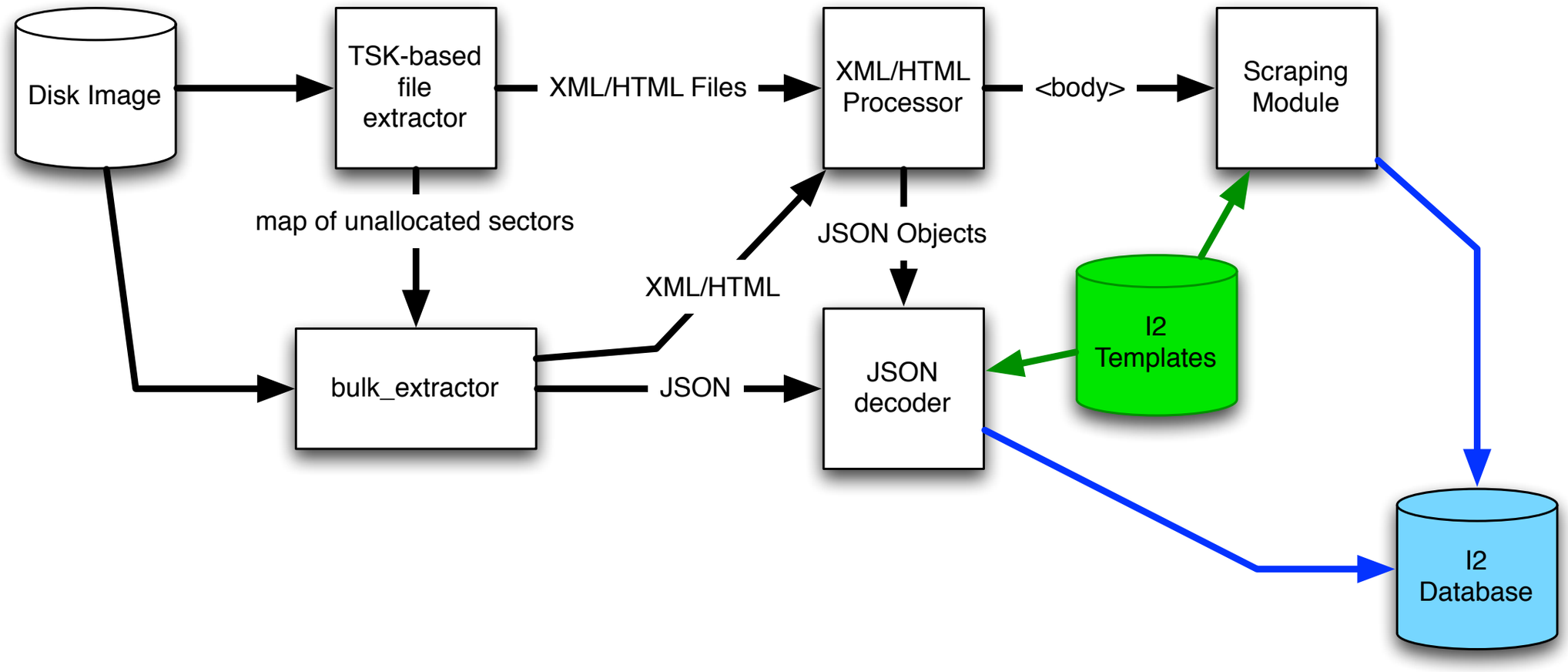


Tool #2: Smirk

Smirk is a tool for reconstructing social media information.

Phase 1 — Smirk scans the disk image for Facebook data

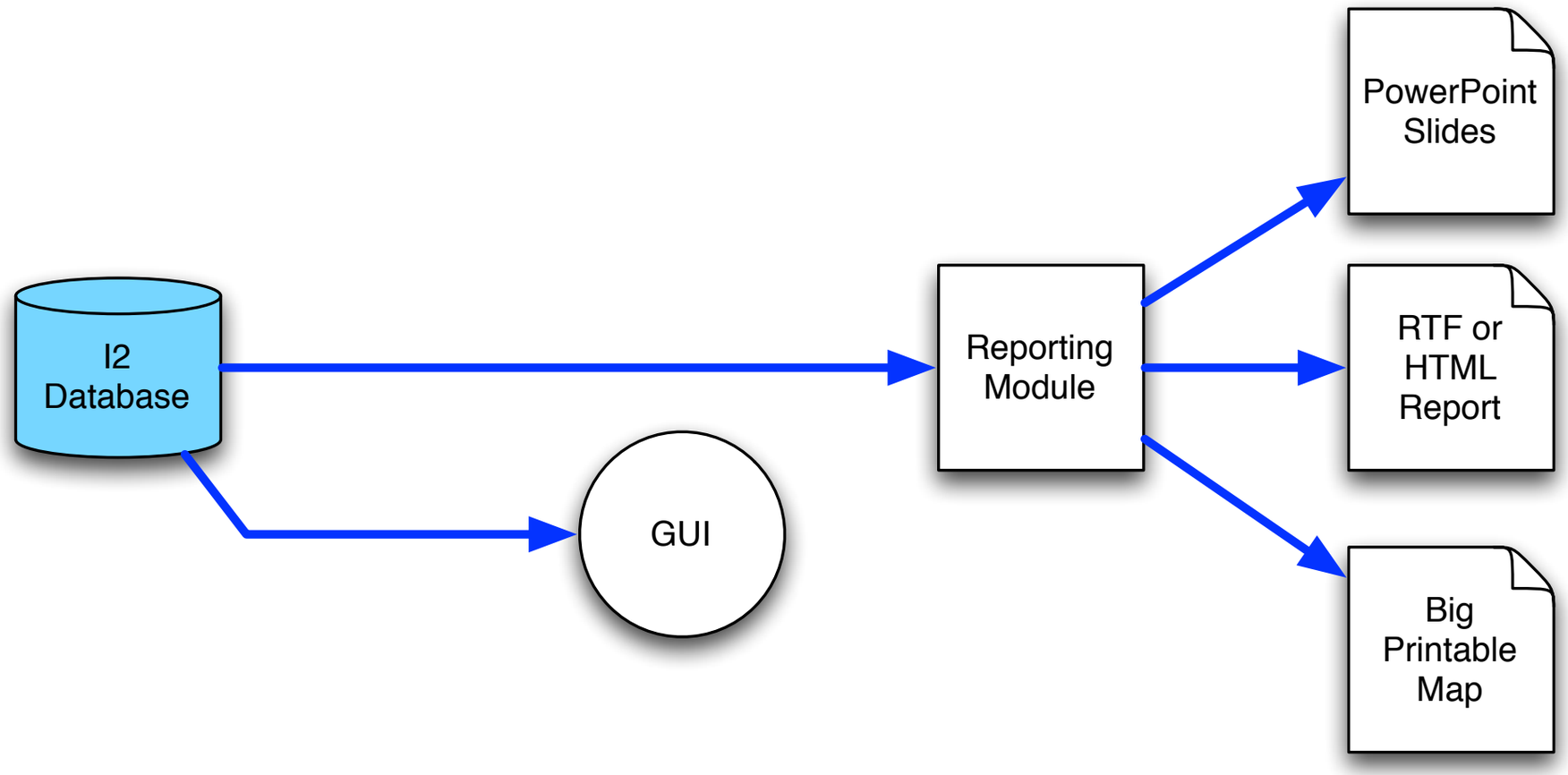
- Results stored in a database



Smirk is designed to be easy to use

Phase 2 — Smirk extracts data from the database

- Automatically produces PowerPoint and Excel reports.
- Will produce other reports (I2?) in the future.



SMIRK

Social Media Analysis Report

sample.vmdk

Generated 9/19/2011 6:22:58 PM

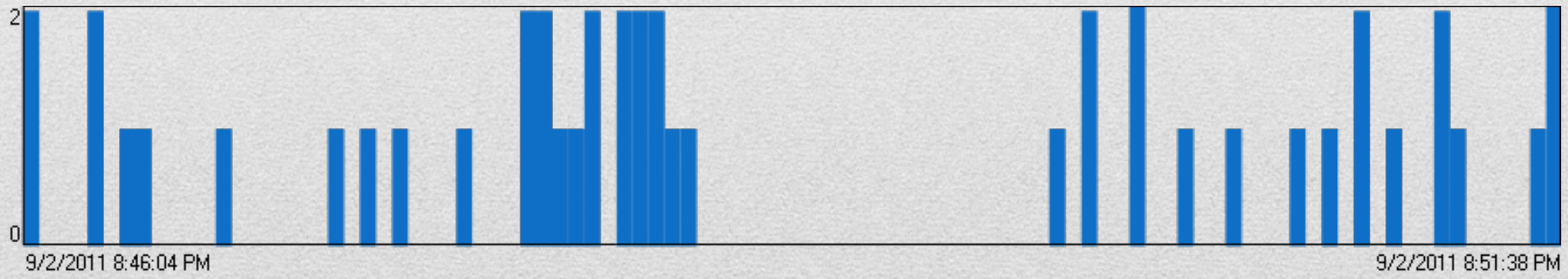
UNCLASSIFIED

Version
SMIRK 1.0 Prototype

Facebook Usage Summary

Facebook Accounts Used:	1
Earliest Facebook Usage:	2011-09-02 20:48:36Z
Latest Facebook Usage:	2011-09-02 20:52:39Z
Facebook URLs Visited:	47
Facebook Cached Items:	533
Facebook Chat Sessions:	2
Total Facebook Accounts Observed:	285

Facebook Access Over Time:



Version
SMIRK 1.0 Prototype

Active Facebook Account

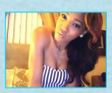
Facebook ID: 100001926917994

Name: Jason Peterson

Version

SMIRK 1.0 Prototype

Facebook Profile Views

	Profile	Time	URL (www.facebook.com/...)
	Jason Peterson [100001926917994]	2011-09-02 20:46:25Z	profile.php?id=100001926917994
	Susan Dillard [100001995672759]	2011-09-02 20:50:17Z	profile.php?id=100001995672759
	Surya Gupta [100002880500594]	2011-09-02 20:50:54Z	profile.php?id=100002880500594
	Ghulam Mustafa [100002861695001]	2011-09-02 20:51:12Z	profile.php?id=100002861695001
	Gabriela Correas [100002898984389]	2011-09-02 20:51:35Z	profile.php?id=100002898984389

Version

SMIRK 1.0 Prototype

Album Photos Viewed

Viewed: 2011-09-02 20:46:47Z
Owner: Susan Dillard [100001995672759]

Viewed: 2011-09-02 20:47:13Z
Owner: Susan Dillard [100001995672759]

Viewed: 2011-09-02 20:47:25Z
Owner: John Waters [100001686744296]

Viewed: 2011-09-02 20:47:54Z
Owner: Susan Dillard [100001995672759]

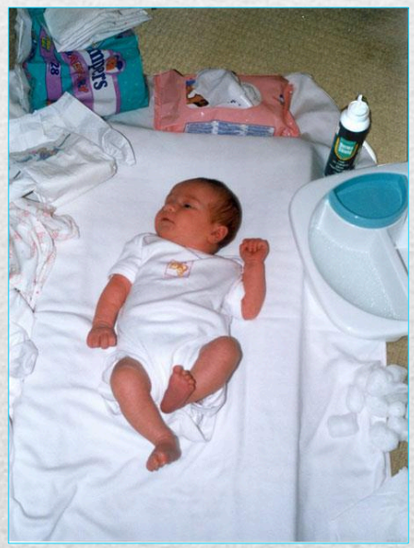
Version
SMIRK 1.0 Prototype

Album Photos Viewed

Viewed: 2011-09-02 20:47:55Z
Owner: Susan Dillard [100001995672759]

Viewed: 2011-09-02 20:47:56Z
Owner: Susan Dillard [100001995672759]

Viewed: 2011-09-02 20:47:58Z
Owner: Susan Dillard [100001995672759]



Viewed: 2011-09-02 20:48:01Z
Owner: Susan Dillard [100001995672759]

Album Photos Viewed

Viewed: 2011-09-02 20:48:03Z
Owner: Susan Dillard [100001995672759]

Viewed: 2011-09-02 20:48:06Z
Owner: Susan Dillard [100001995672759]

Viewed: 2011-09-02 20:48:07Z
Owner: Susan Dillard [100001995672759]

Viewed: 2011-09-02 20:48:13Z
Owner: John Waters [100001686744296]

Chat Session

	User	Time	Message
	Jason Peterson [100001926917994]	2011-09-02 20:46:36Z	'Hey Sue'
	Susan Dillard [100001995672759]	2011-09-02 20:46:47Z	'hey jason, whats up'
	Jason Peterson [100001926917994]	2011-09-02 20:46:59Z	'Are you coming to the meeting with the boss?'
	Jason Peterson [100001926917994]	2011-09-02 20:47:04Z	'It's at 10pm, under the bridge'
	Susan Dillard [100001995672759]	2011-09-02 20:47:56Z	'i might be late but i'll be there'
	Jason Peterson [100001926917994]	2011-09-02 20:49:10Z	'ok I'll see you then'
	Susan Dillard [100001995672759]	2011-09-02 20:49:13Z	'[illegible]'
	Jason Peterson [100001926917994]	2011-09-02 20:49:36Z	'you too!'

Version

SMIRK 1.0 Prototype

Other Facebook IDs on Media

[7824808]
Nick Curtis

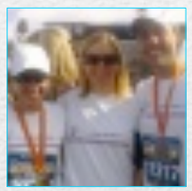
[542567752]
Tiff Illmatic Allen

[586519871]
Mo Greim

[627283604]
James Patterson

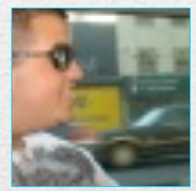
[695124950]
Kate Parnin

[695131311]
Aquib Ijaz



[725996649]
Chris Just

[763248194]
Derick Shomo



[1401910404]
Samer Mahmoud

[1686420084]
Cedric Bullock

[1820887915]
Nathan Foster

[5295171251]
John Mayer

[5678046685]
U2

[6295077515]
Sugarland

[6564142497]
Usher

Version
SMIRK 1.0 Prototype

Other Facebook IDs on Media

[6815841748]
Barack Obama

[6885814958]
Lil Wayne

[7126051465]
Katy Perry

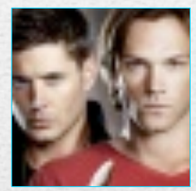
[8210451787]
Linkin Park

[8225281300]
Superbad

[8651510029]
Johnny Cash

[9328458887]
adidas Originals

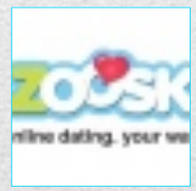
[9899376497]
John Cena - WWE Universe



[9991232322]
Supernatural

[10376464573]
Lady Gaga

[12073986982]
John Terry



[13065625020]
Zoosk

[15750194565]
Susan Tedeschi

[16100944031]
System of a Down

[16547831022]
Xbox

Version
SMIRK 1.0 Prototype

Classification

Case Name

Media Summary

Media Path:	c:\Program Files (x86)\Smirk\sample.vmdk
Processing Start Time:	2011-09-19 18:21:51Z
Processing Duration:	0h 1m 3s
Web Browsers:	Mozilla Firefox 5.0.1
Machine Accounts Used:	'FB User'

Smirk status: prototype available now

Smirk is an 18 month project.

Prototype 1.0 available today.

- http://simson.net/smrik

January 2012 — Alpha 1 release

March 2012 — Beta 1 release

August 2012 — Release 1.0

Tool #3: frag_find

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

Consider a disk sector with 512 bytes.

- There are $2^{512x8} \approx 10^{1,233}$ different sectors.
- A randomized sector is likely to be "distinct."
  - —*e.g. encryption keys, high-entropy data, etc.*

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.

Many JPEGs seem to contain distinct blocks.

Experimentally, we see that most JPEGs have distinct blocks…

208 distinct 4096-byte block hashes

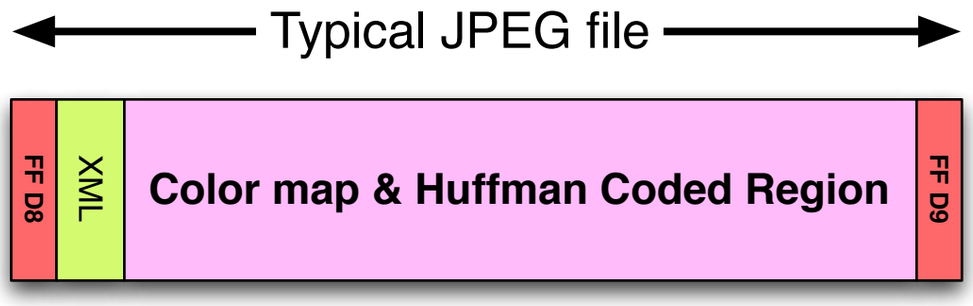
∩

204 distinct 4096-byte block hashes

=

0 block hashes in common

Even with JPEG headers, XML, and Color Maps:



In fact, many JPEGs also have sectors in common (whitespace).

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.

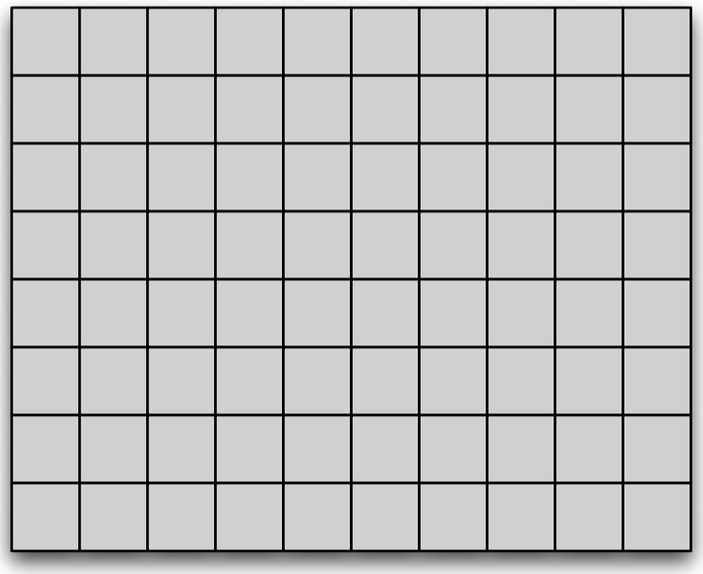
- *(Distinct Block Hypothesis #1:*
  - *If a block of data from a file is distinct, then a copy of that block found on a data storage device is evidence that the file was once present.)*

Hash-based carving uses distinct blocks to find scrambled content.

Input:

- 1 or more *Master Files*

- A disk image



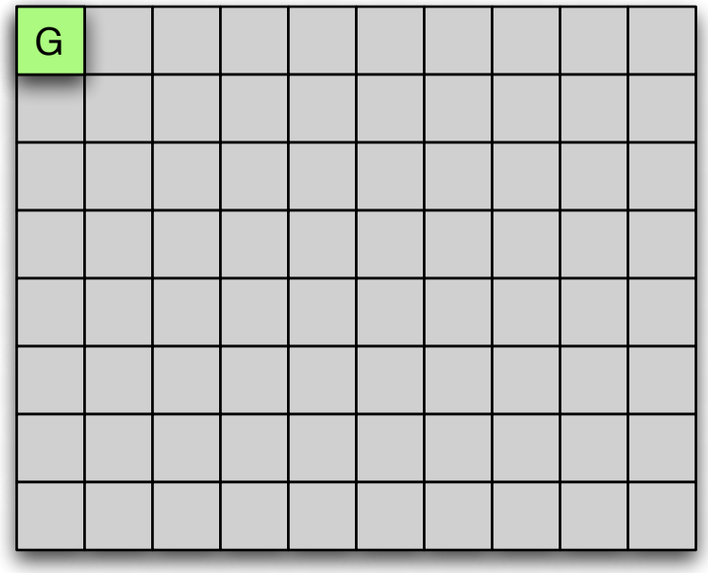
Hash-based carving finds the master file in the disk image.

Hash-based carving uses distinct blocks to find scrambled content.

Input:

- 1 or more *Master Files*

- A disk image



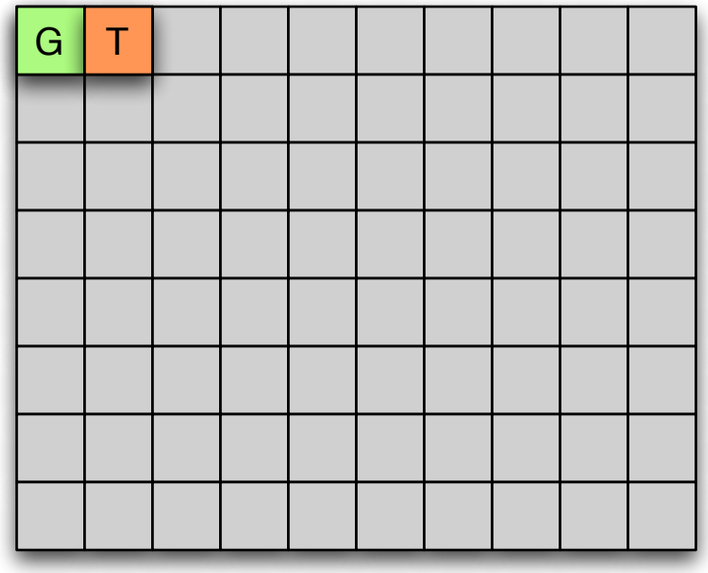
Hash-based carving finds the master file in the disk image.

Hash-based carving uses distinct blocks to find scrambled content.

Input:

- 1 or more *Master Files*

- A disk image



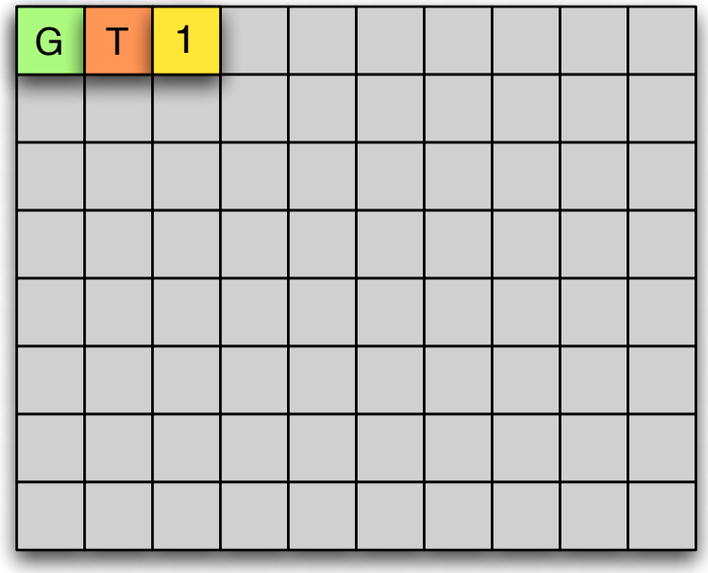
Hash-based carving finds the master file in the disk image.

Hash-based carving uses distinct blocks to find scrambled content.

Input:

- 1 or more *Master Files*

- A disk image



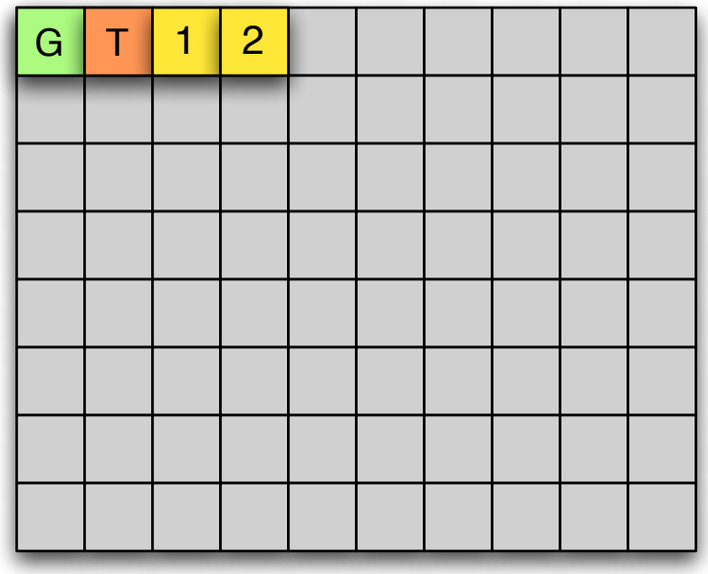
Hash-based carving finds the master file in the disk image.

Hash-based carving uses distinct blocks to find scrambled content.

Input:

- 1 or more *Master Files*

- A disk image



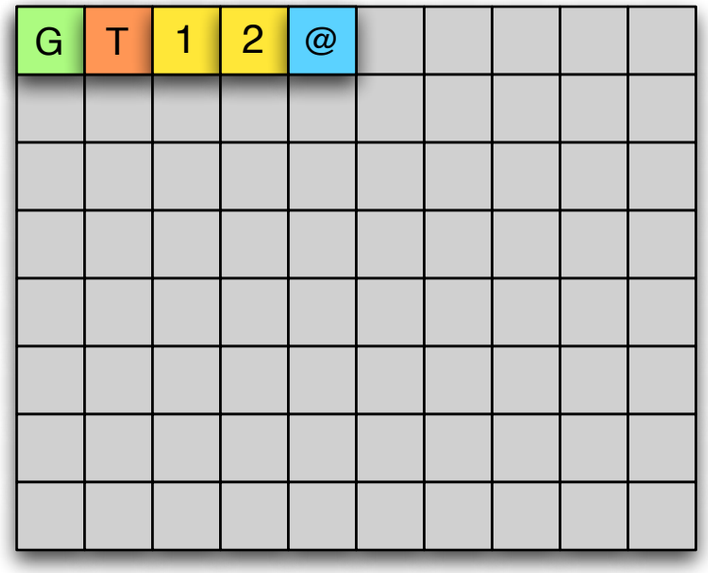
Hash-based carving finds the master file in the disk image.

Hash-based carving uses distinct blocks to find scrambled content.

Input:

- 1 or more *Master Files*

- A disk image



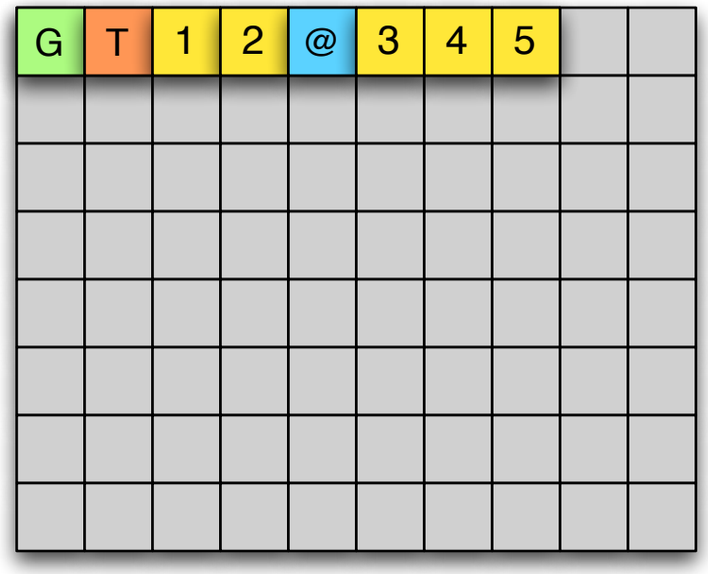
Hash-based carving finds the master file in the disk image.

Hash-based carving uses distinct blocks to find scrambled content.

Input:

- 1 or more *Master Files*

- A disk image

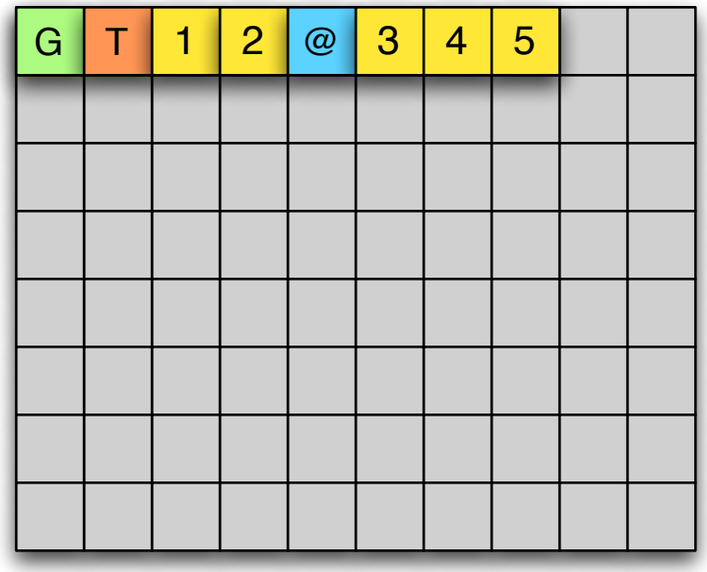


Hash-based carving finds the master file in the disk image.

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:

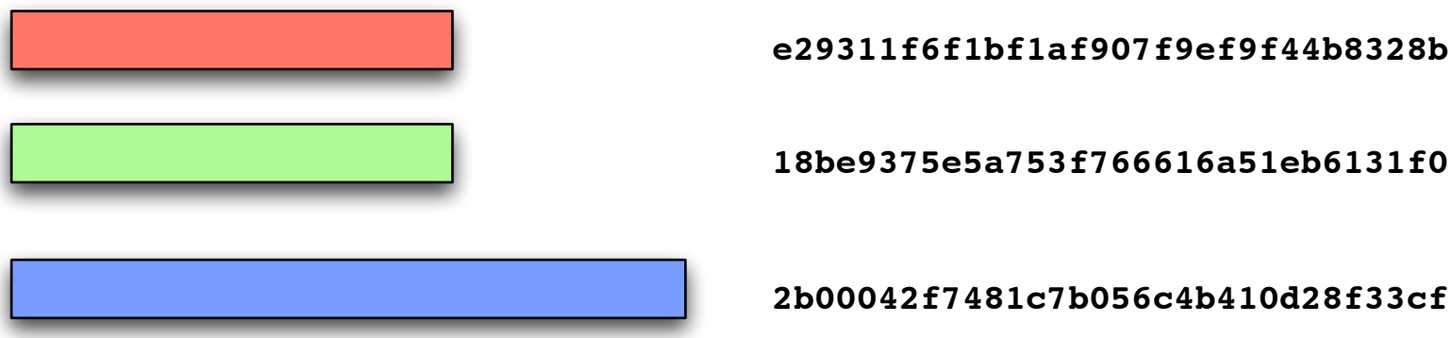
```
<fileobject>
  <filename>DCIM/100CANON/IMG_0001.JPG</filename>
  <byte_runs>
    <run file_offset='0' fs_offset='55808' img_offset='81920' len='855935'/>
      <hashdigest type='MD5'>b83137bd4ba4b56ed856be8a8e2dc141</hashdigest>
      <hashdigest type='SHA1'>03eaa4a5678542039c29a5ccf12b3d71ae96cbd2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

Uses:

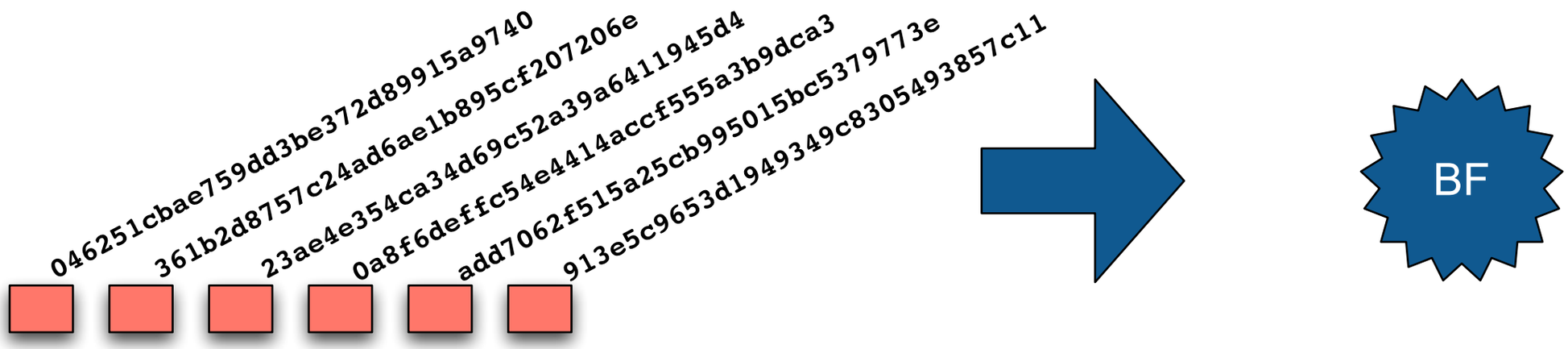
- Exfiltration of sensitive documents;
- Data Loss Detection; etc.
- Download from http://afflib.org/

Distinct Block Recognition can be used to find objectionable material.

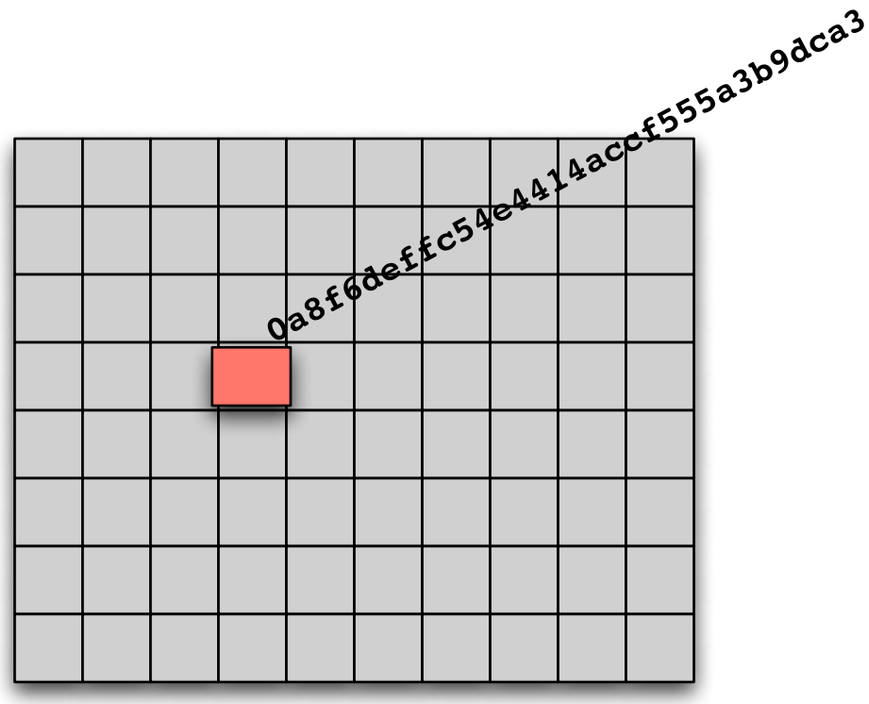
Objectionable materials are detected with hash sets.



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

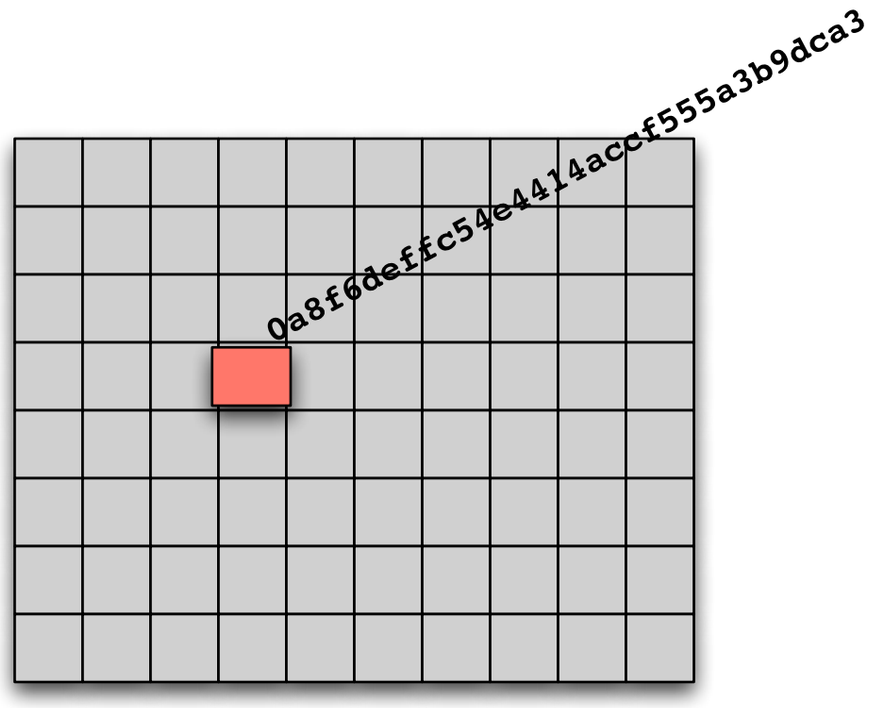


If a sector of an objectionable file is found on a drive...



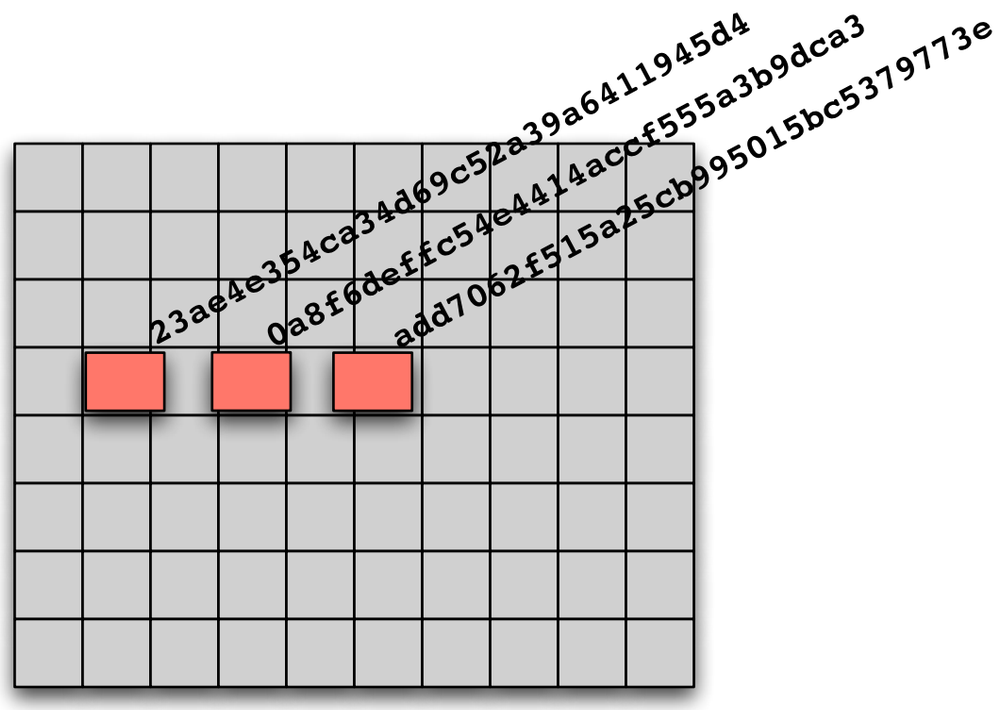
If a sector of an objectionable file is found on a drive...

Then either the entire file was once present…



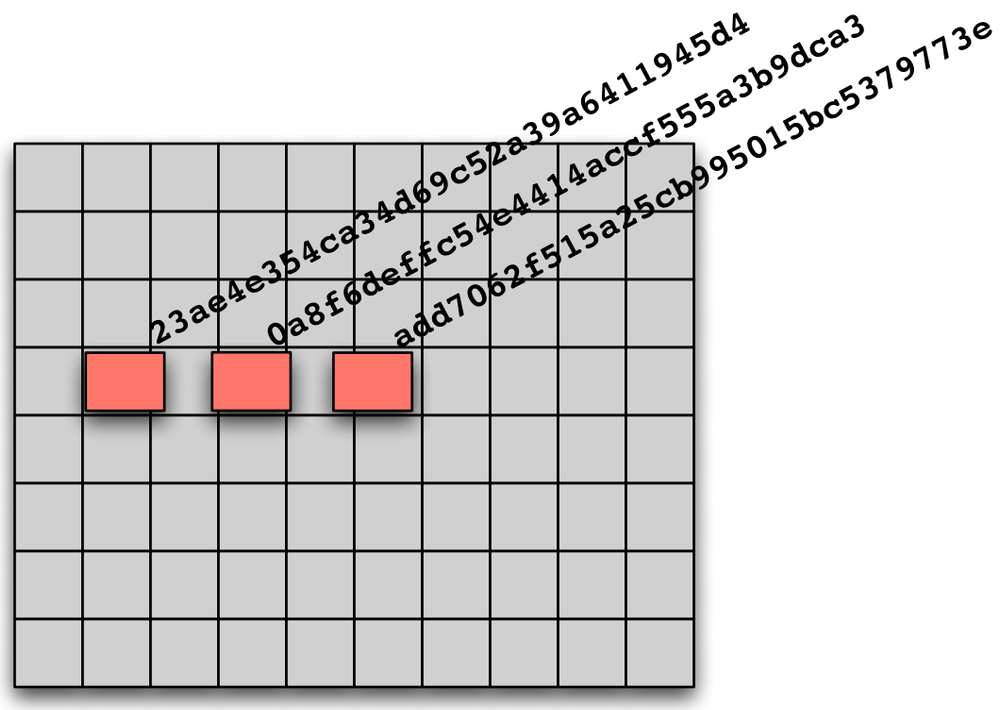
If a sector of an objectionable file is found on a drive...

Then either the entire file was once present…

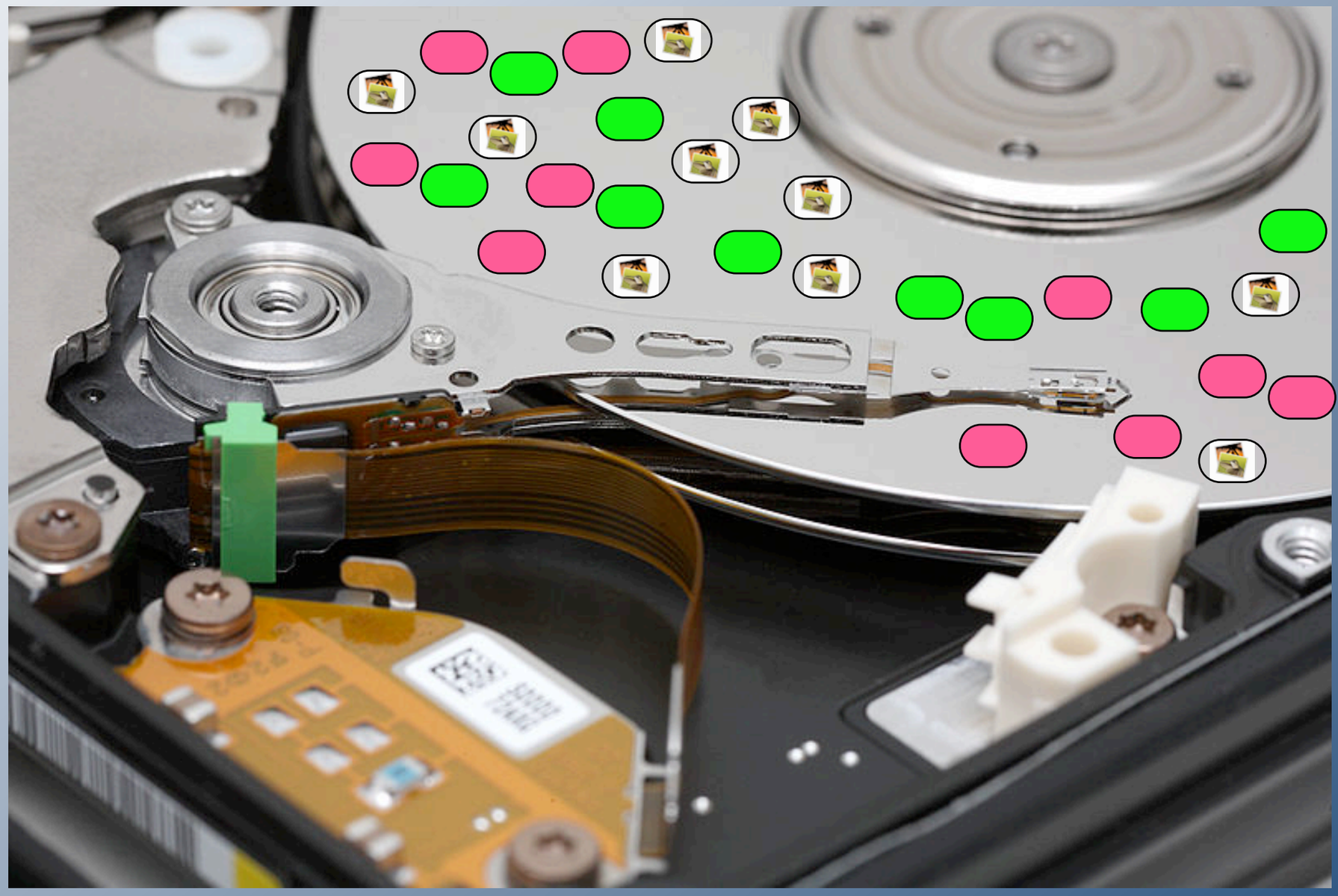


If a sector of an objectionable file is found on a drive...

Then either the entire file was once present…



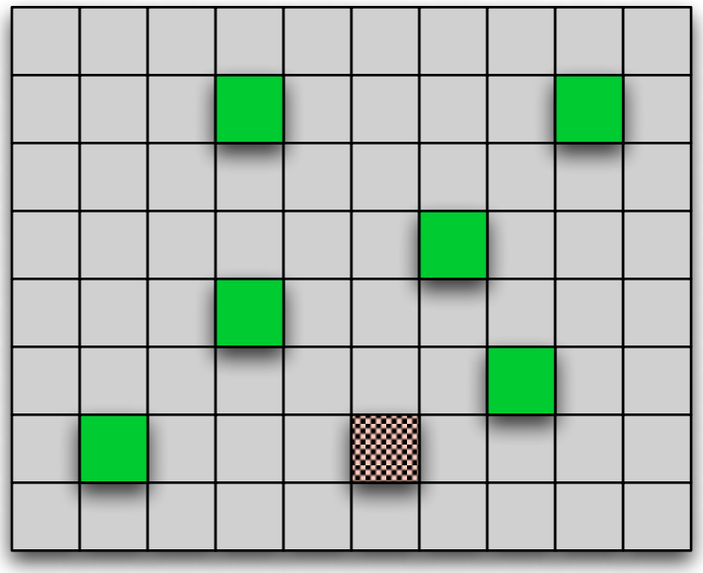
… or else the sector really isn't distinct.



Tool #4: Random Sampler

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 search a database.

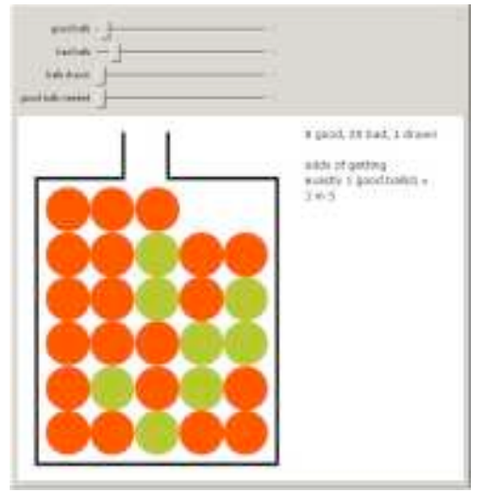
- A 1TB database of objectionable material has just 64 GiB of hashes.

Finding a match indicates the presence of objectionable material.

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.

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

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.

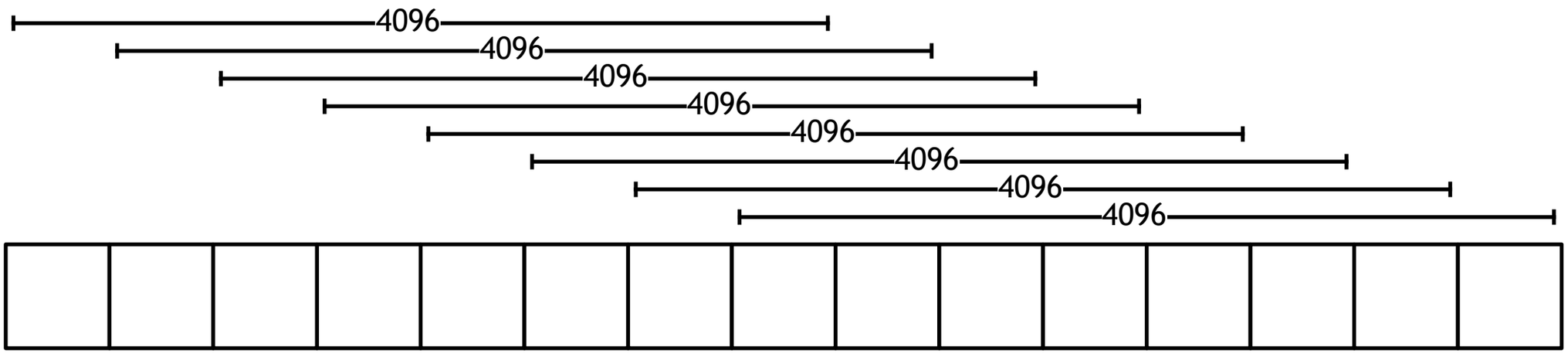
Increase efficiency with larger block size.

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!

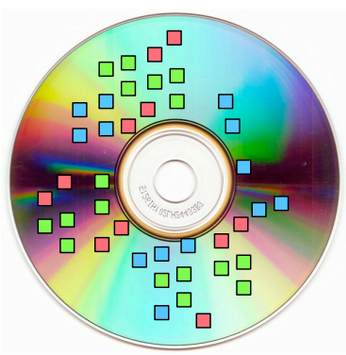
- We read 15 512-byte blocks.
- Then we compute 8 different 4096-byte block hashes.
- Each one is checked in the Bloom Filter



(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.

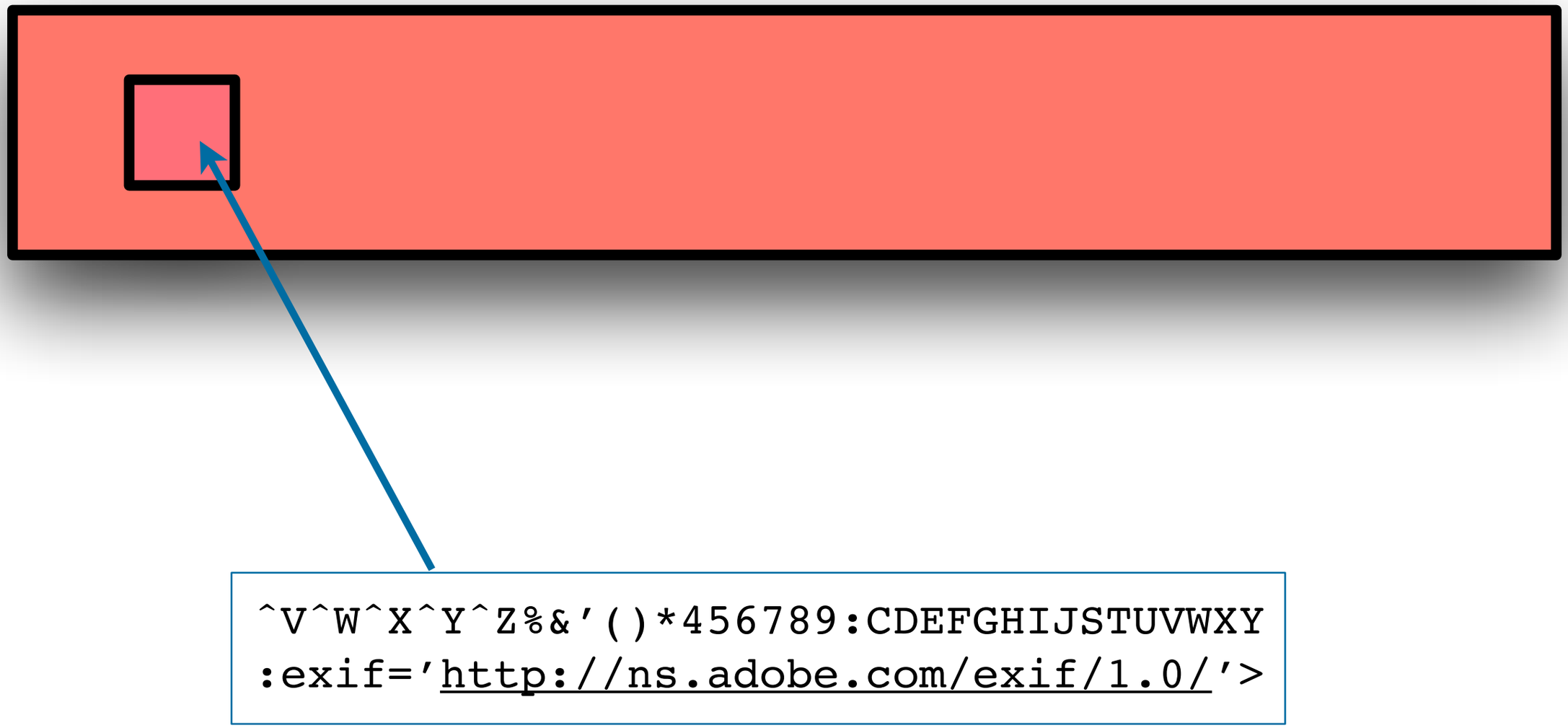
		
Minutes	208	5
Max Data Read	1 TB	24 GB



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

File fragment identification can make this technique more powerful.

File fragment identification is a well-studied problem.

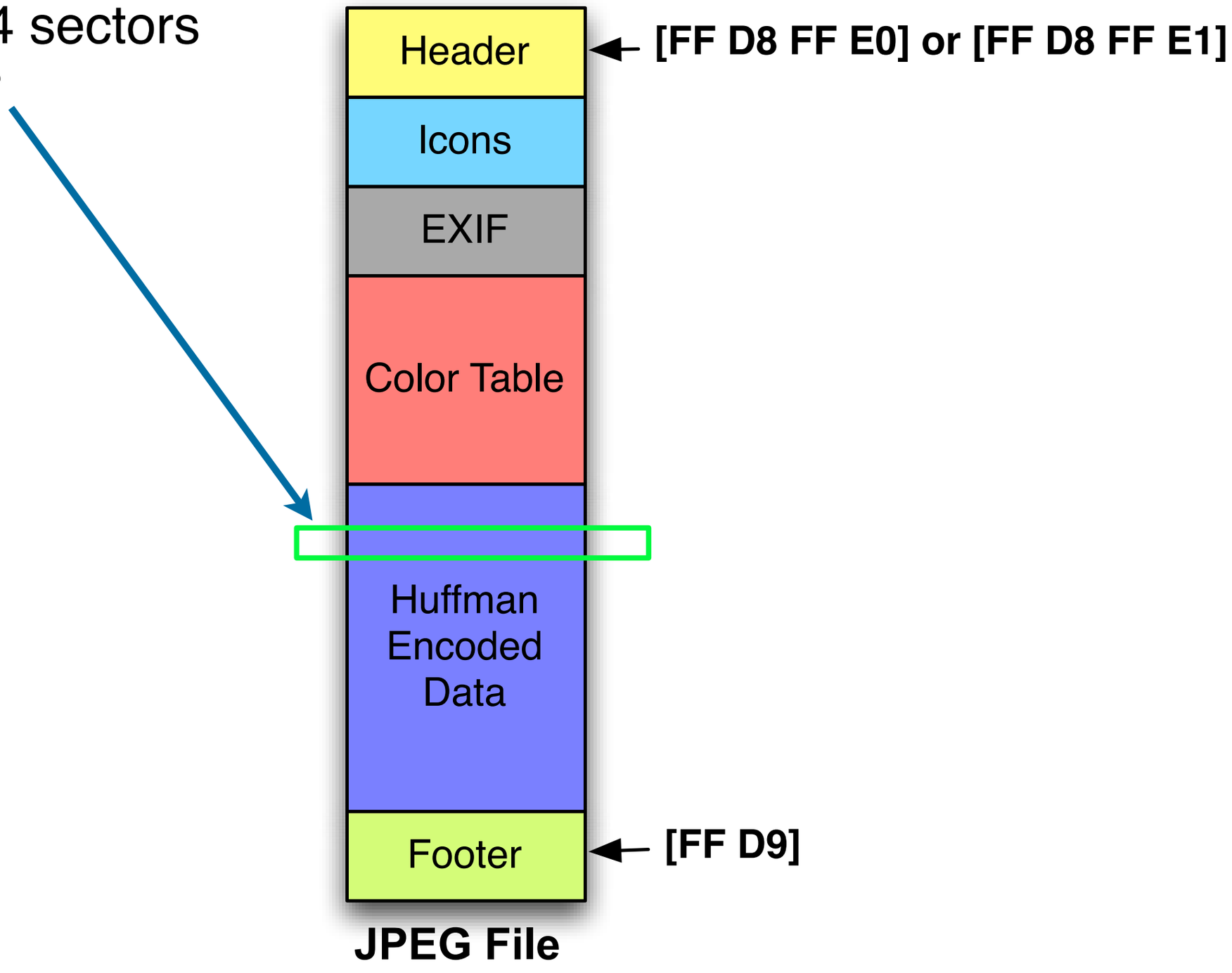


This fragment from a file is highly suggestive of a JPEG.

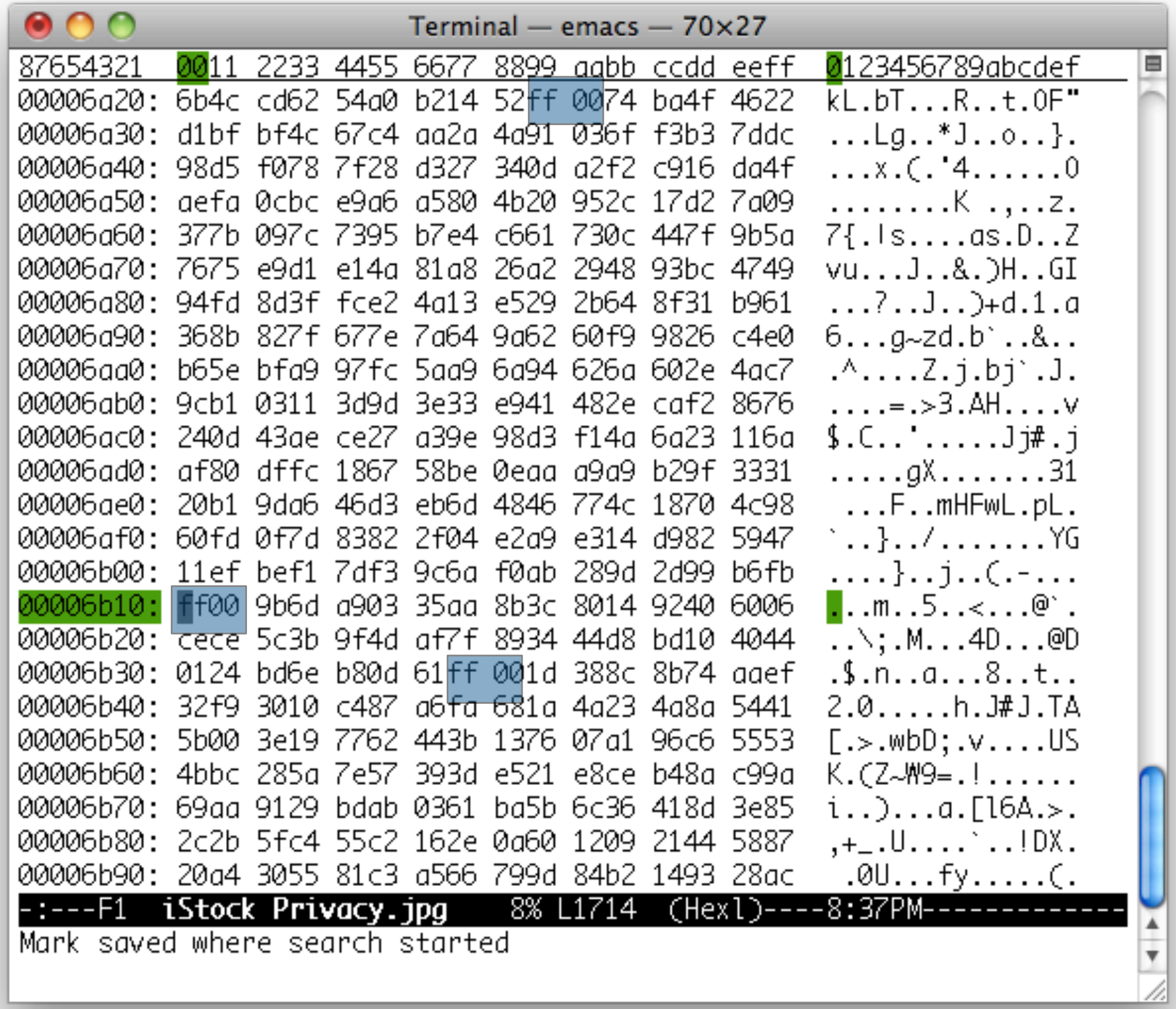
Other fragments can also contain JPEG “features.”

JPEG files contain significant internal structure. Most identification systems look at headers & footers.

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



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

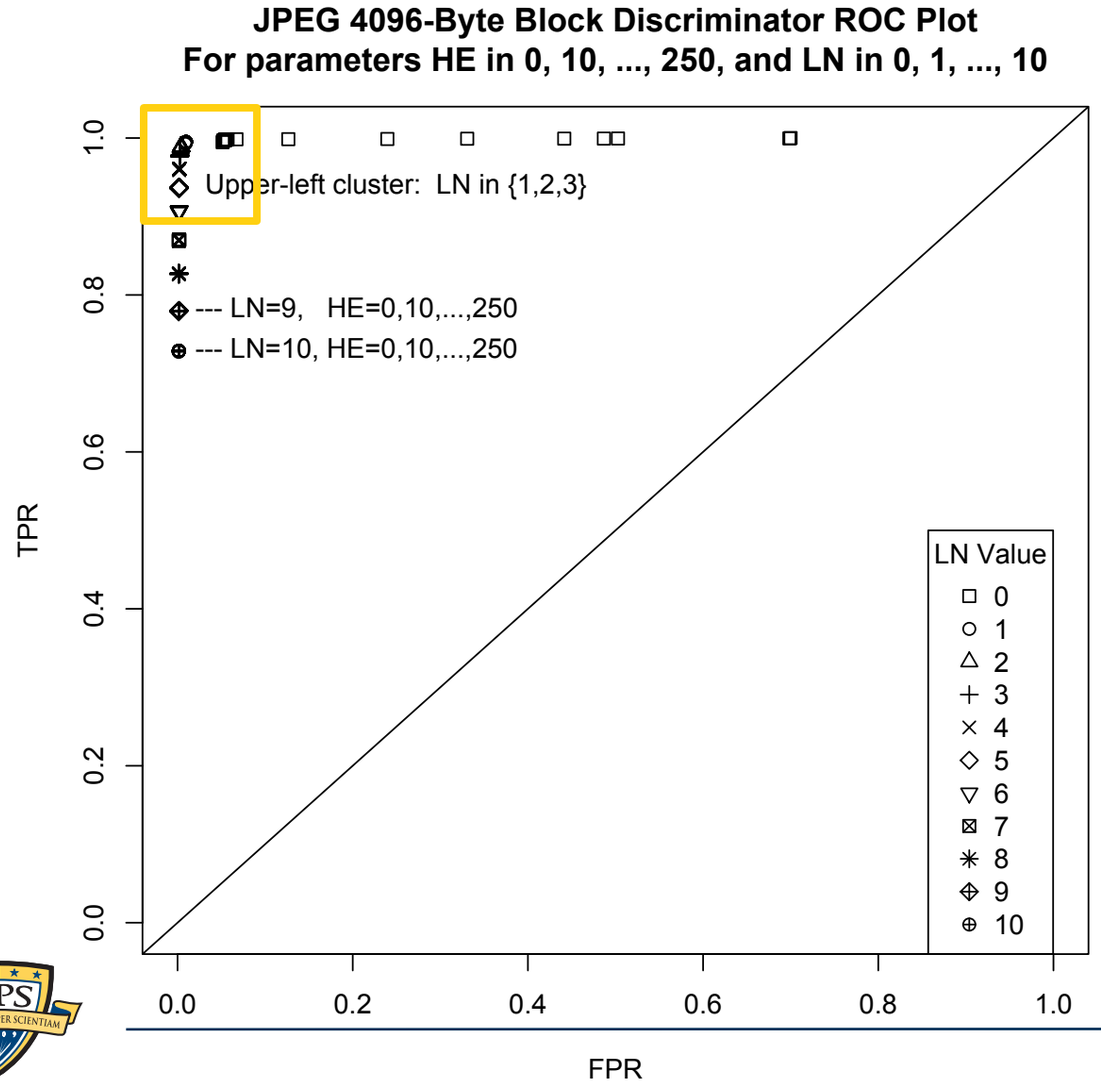


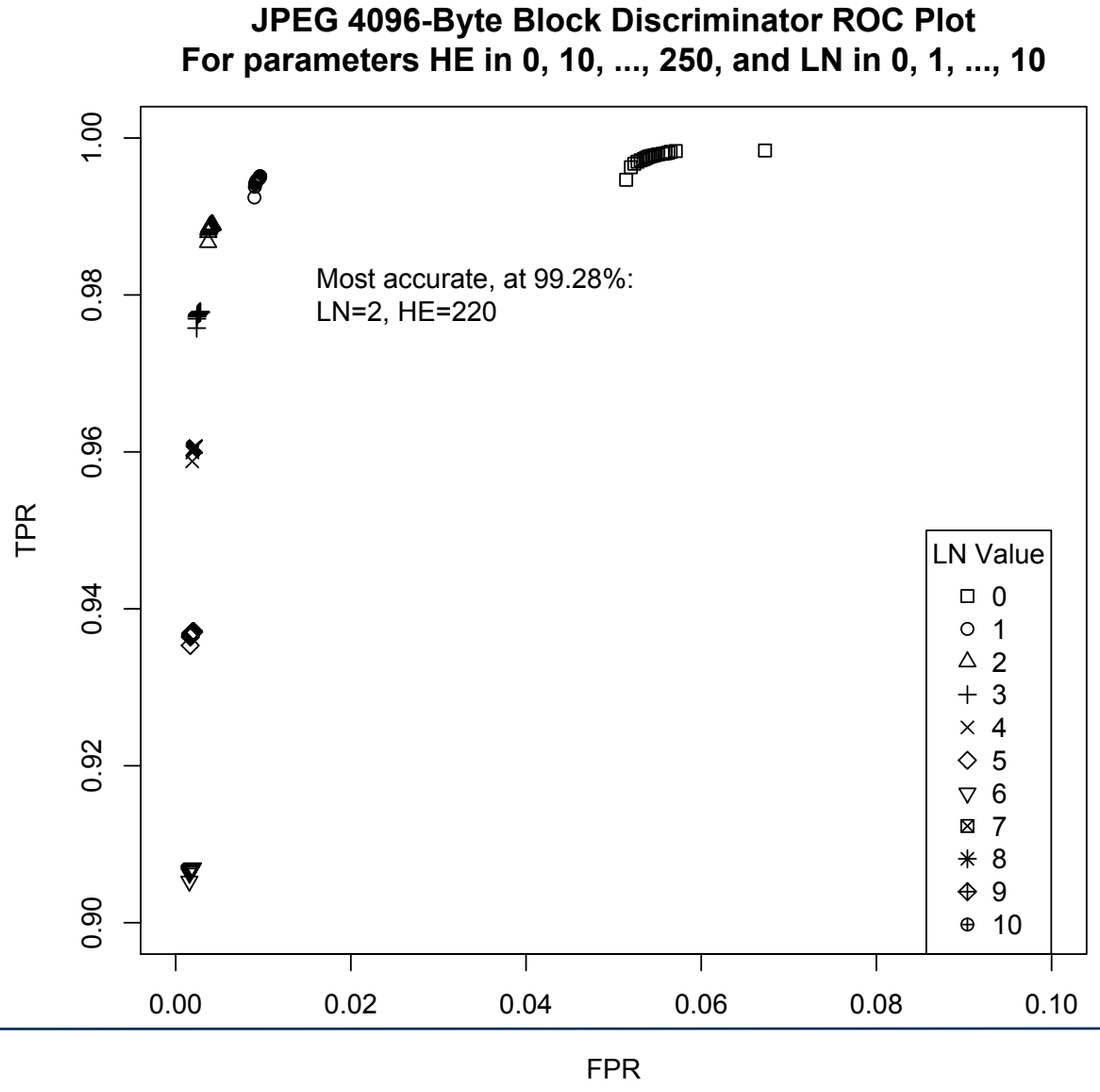
Our JPEG discriminator counts the number of FF00s.

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

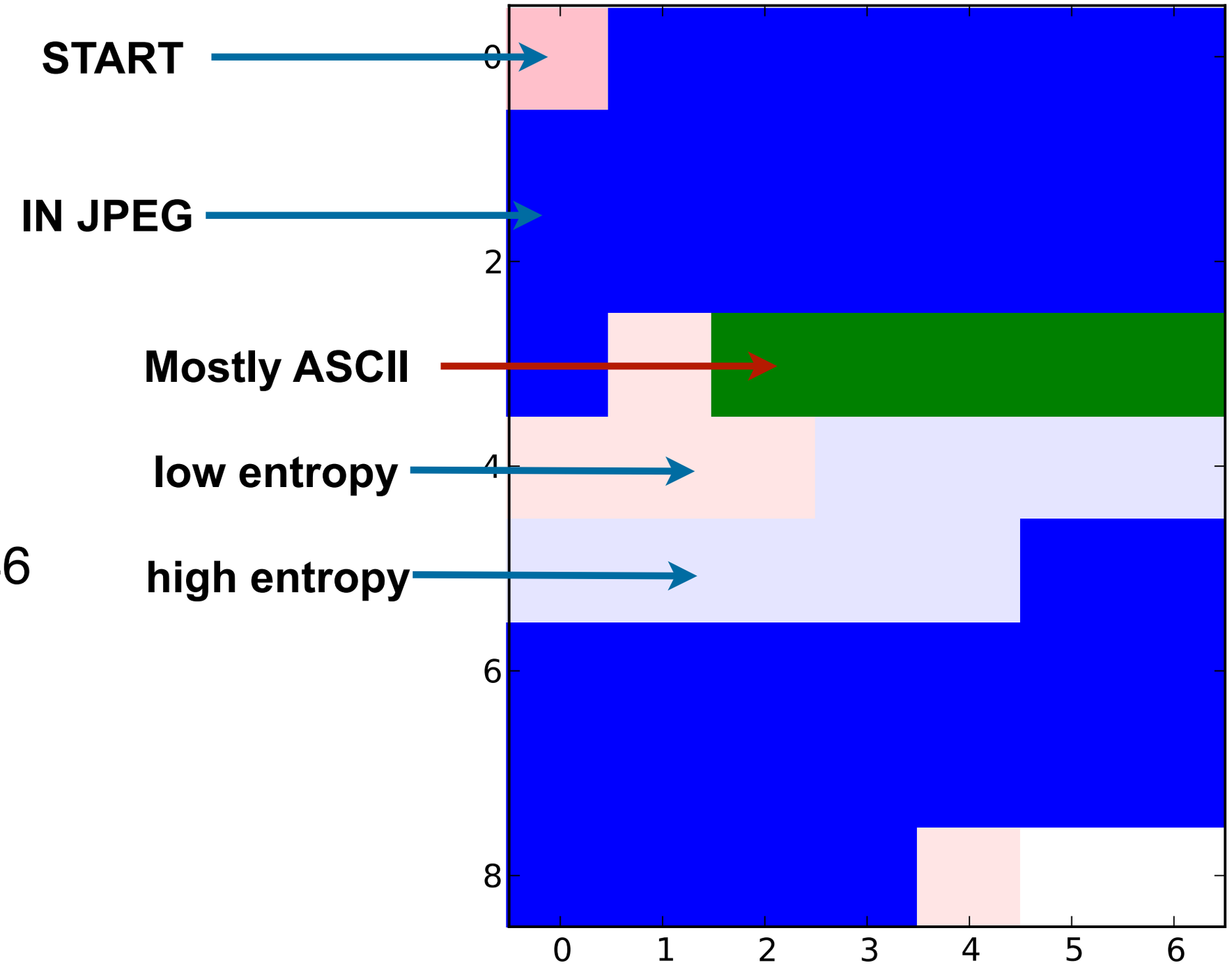
We perform a search to find the best values.





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

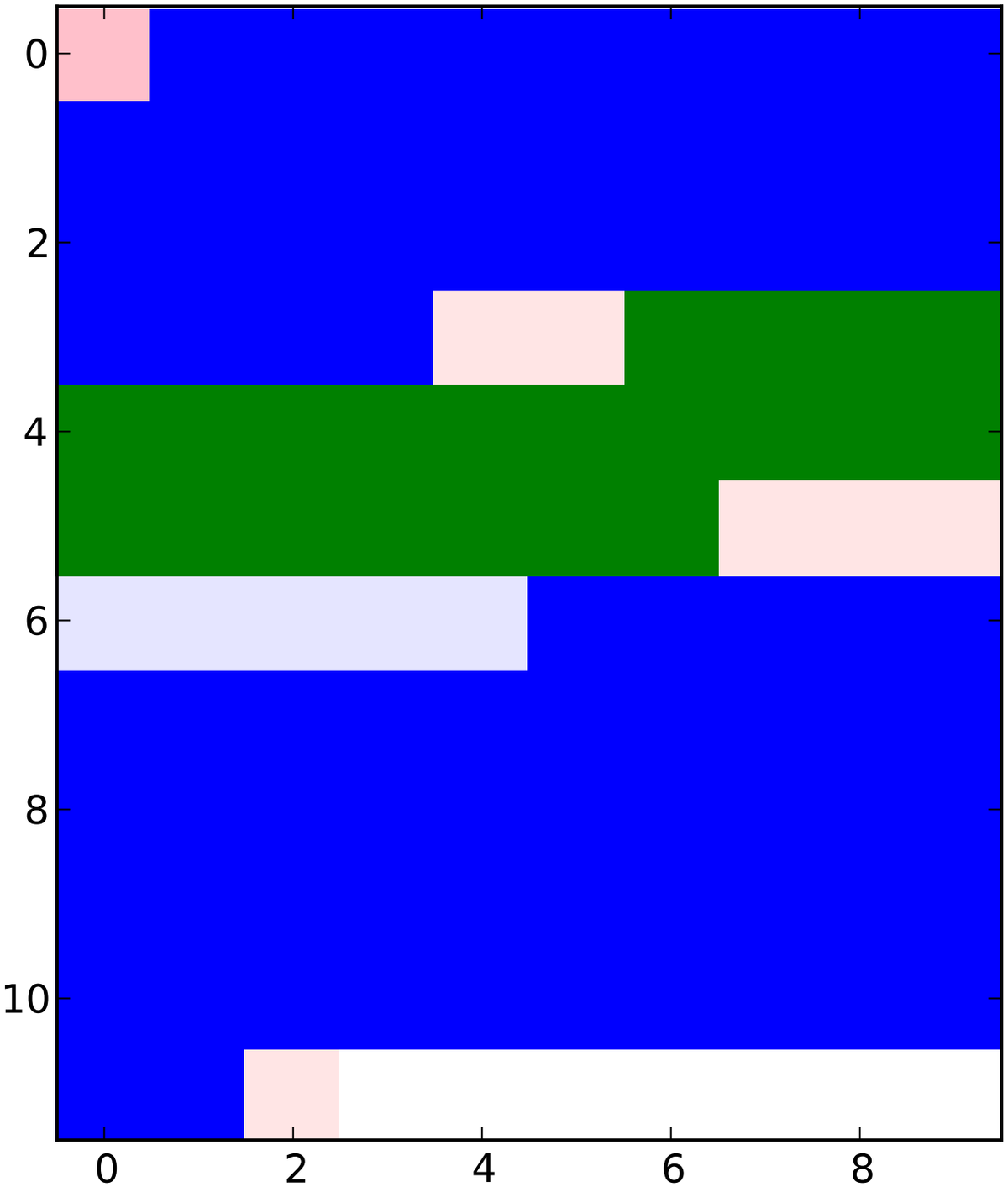
Bytes: 31,046



512-byte sectors: 61

000897.jpg

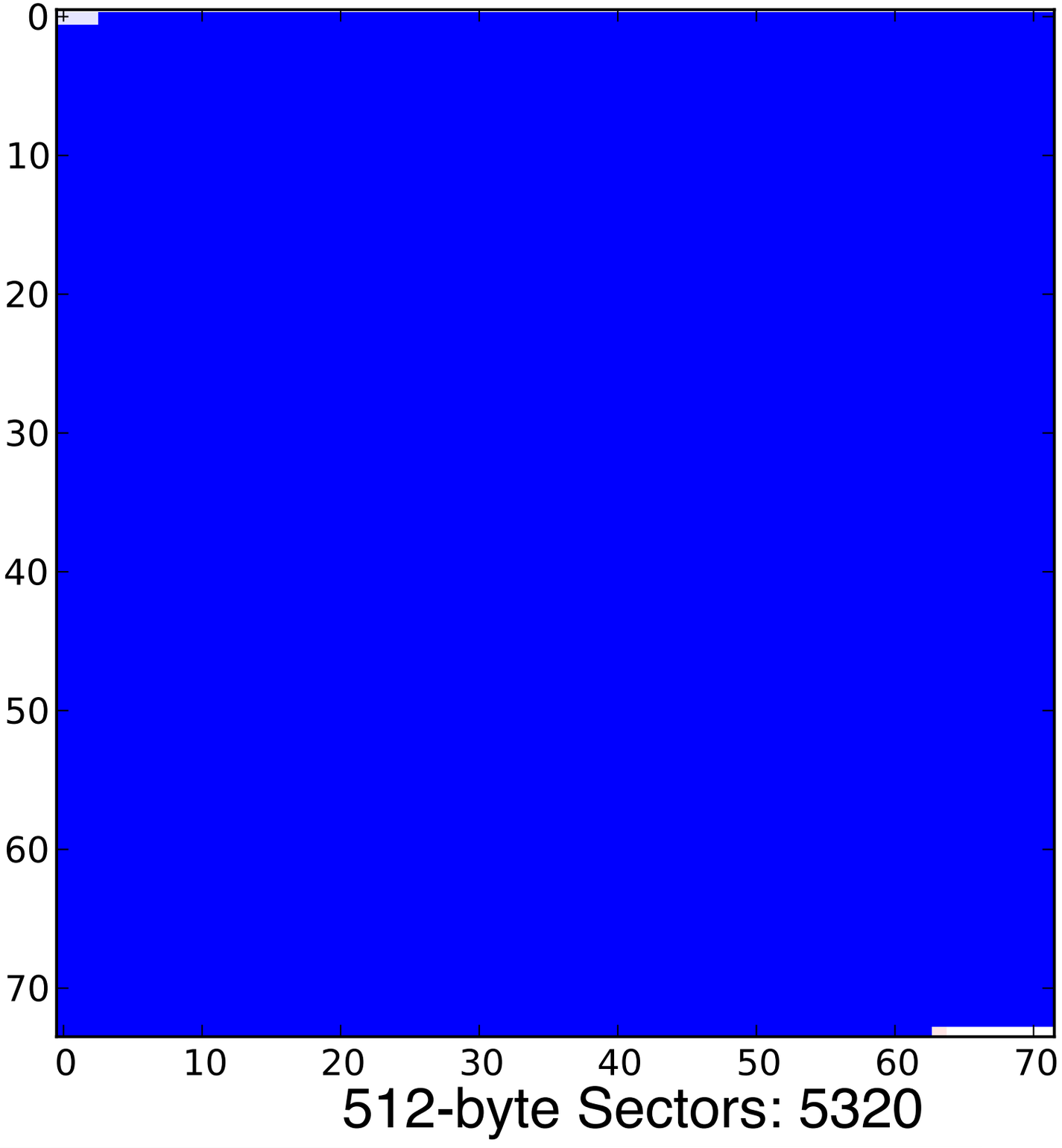
Bytes: 57,596



512-byte Sectors: 113

000888.pdf

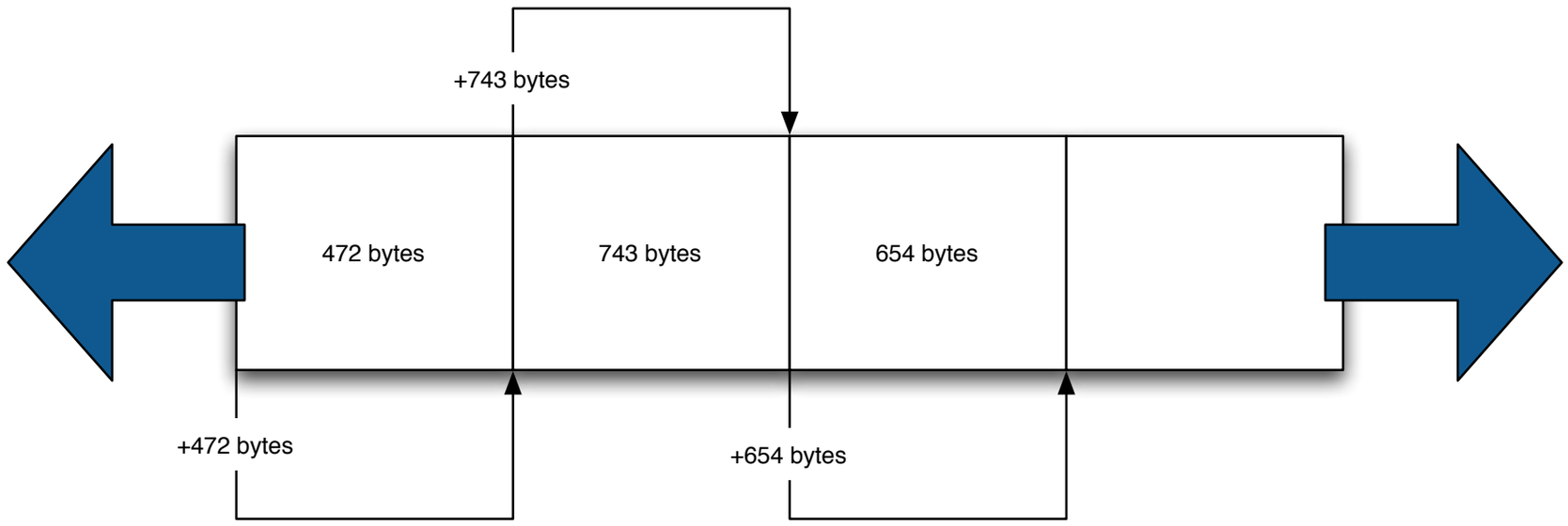
Bytes: 2,723,425



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.

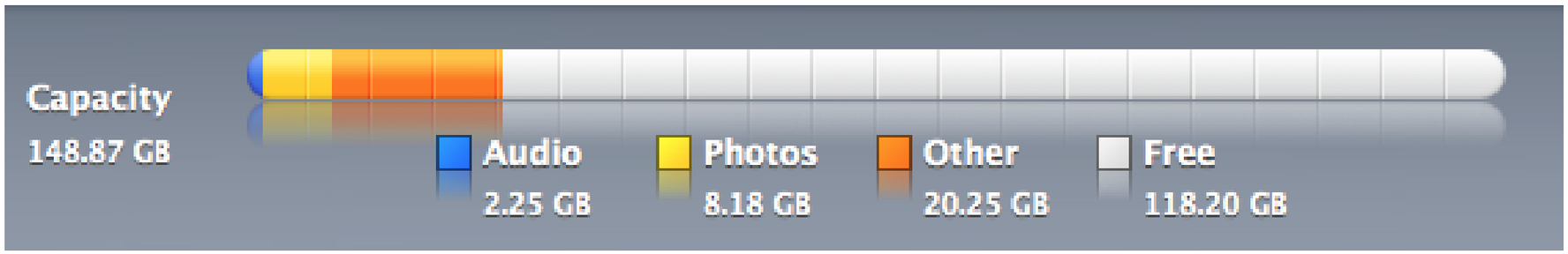


Our MP3 discriminator:

- 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.

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