bulk_extractor and the NPS toolset

Simson L. Garfinkel
Associate Professor, Naval Postgraduate School

May 17, 2012

http://domex.nps.edu/

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)

Schools:

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

NCR Initiative:

- 8 offices on 5th floor, 900N Glebe Road, Arlington
- Current staffing: 4 professors, 2 lab managers, 2 programmers, 4 contractors
- **OPEN SLOTS FOR .GOV PHDs!**

Monterey, CA

900N Glebe, Arlington, VA

NPS research: “Automated Media Exploitation”

Area #1: Bulk Data Analysis

- Feature extraction (bulk_extractor)
- Statistical techniques (random sampler)
- Similarity metrics (sdhash & sdtext)

Area #2: End-to-end automation of forensic processing

- Digital Forensics XML Toolkit (fiwalk, md5deep, etc.)
- Disk Image ⇛ Power Point (smirk)

Area #3: Data mining for digital forensics

- Automated analysis (cross-drive analysis)

Area #4: Creating Standardized Forensic Corpora

- Freely redistributable disk and memory images, packet dumps, files (digitalcorpora.org).

Stream-based forensics with 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.

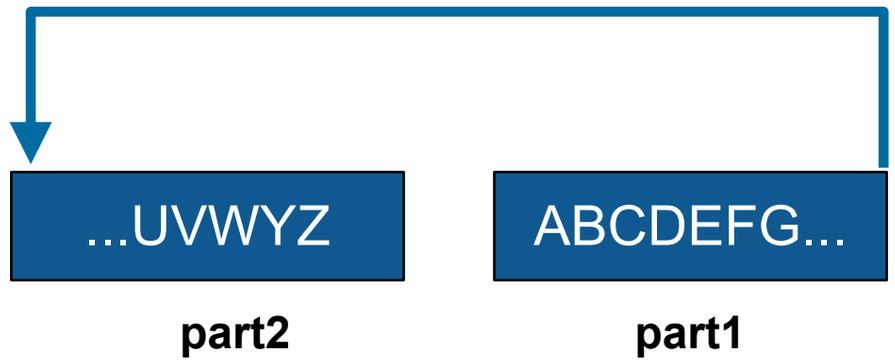
Easy to parallelize (“embarrassingly parallel”)

Results:

- Potential to read at maximum I/O transfer rates.
  - *Requires a lot of cores (24+).*
- Generates immediate answers

- Reads all the data — allocated files, deleted files, file fragments.
  - *Separate metadata extraction required to get the file names.*

Primary disadvantage: Fragmented files may not be recovered.



ZIP, GZIP & LZMA use *adaptive* compression algorithms.

- Part 1 required to decompress part 2.
- Also an issue for JPEG.

Fortunately, most files are *not* fragmented.

- Individual components of a ZIP can be recovered (e.g. `word/document.xml`)

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

- Log files, Outlook PST files, etc.

Our experience: bulk_extractor is *faster* and *finds data 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 email address, URLs, CCNs that other tools miss

- "Optimistically" decompresses and re-analyzes all data.
- Finds data in browser caches (downloaded with zip/gzip), and in many file formats.
- Carves IP packets, libpcap files, etc.

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

- Produces "histogram" of email addresses, credit card numbers, etc.
- Distinguishes primary user from incidental users.
- Produces pcap files from packets.

Uses:

- Triage & pre-processing.

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.*

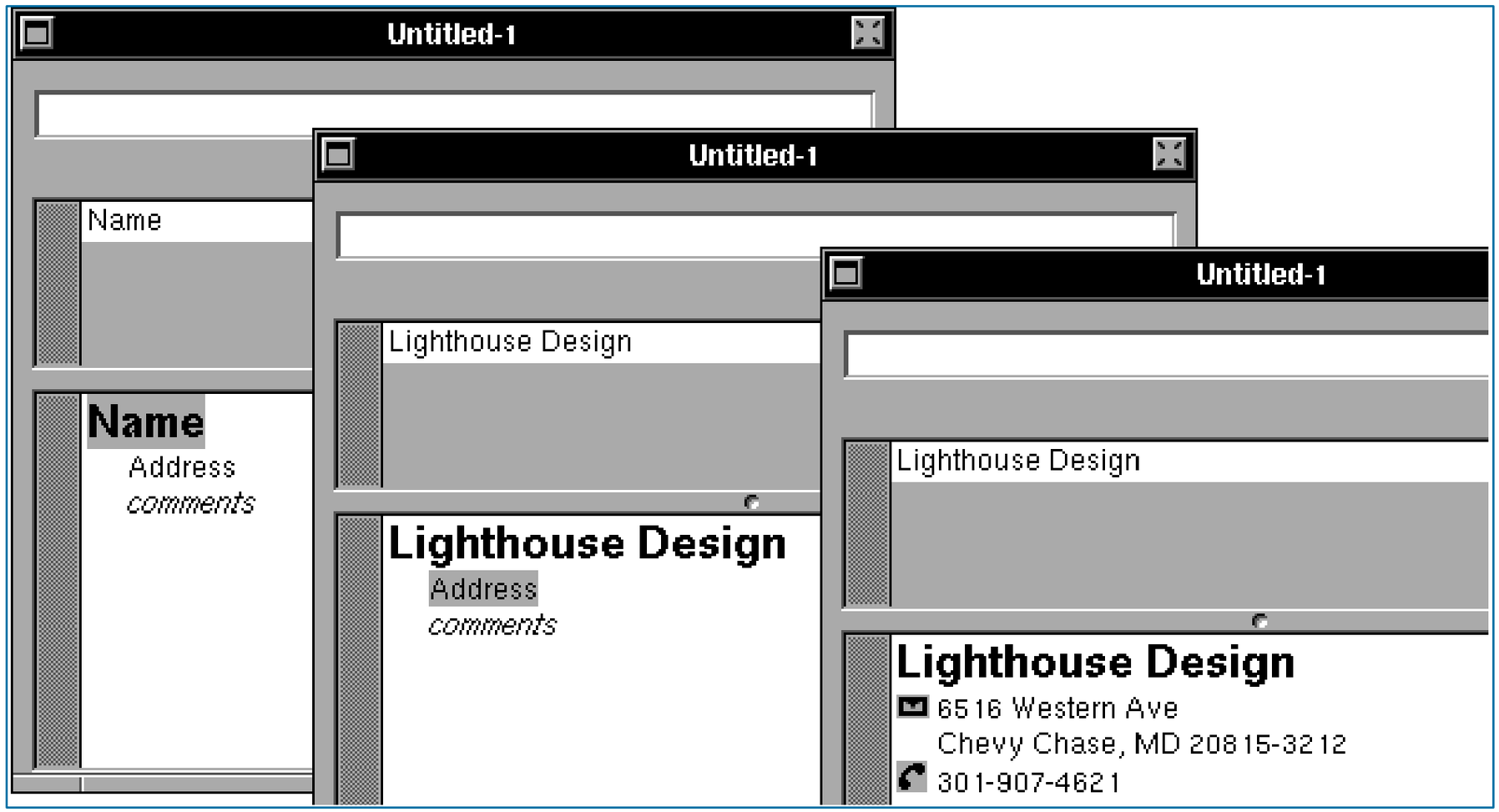
Bulk_extractor found:

- 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.

bulk_extractor: 20+ years of work

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



SBook automatically found 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!*

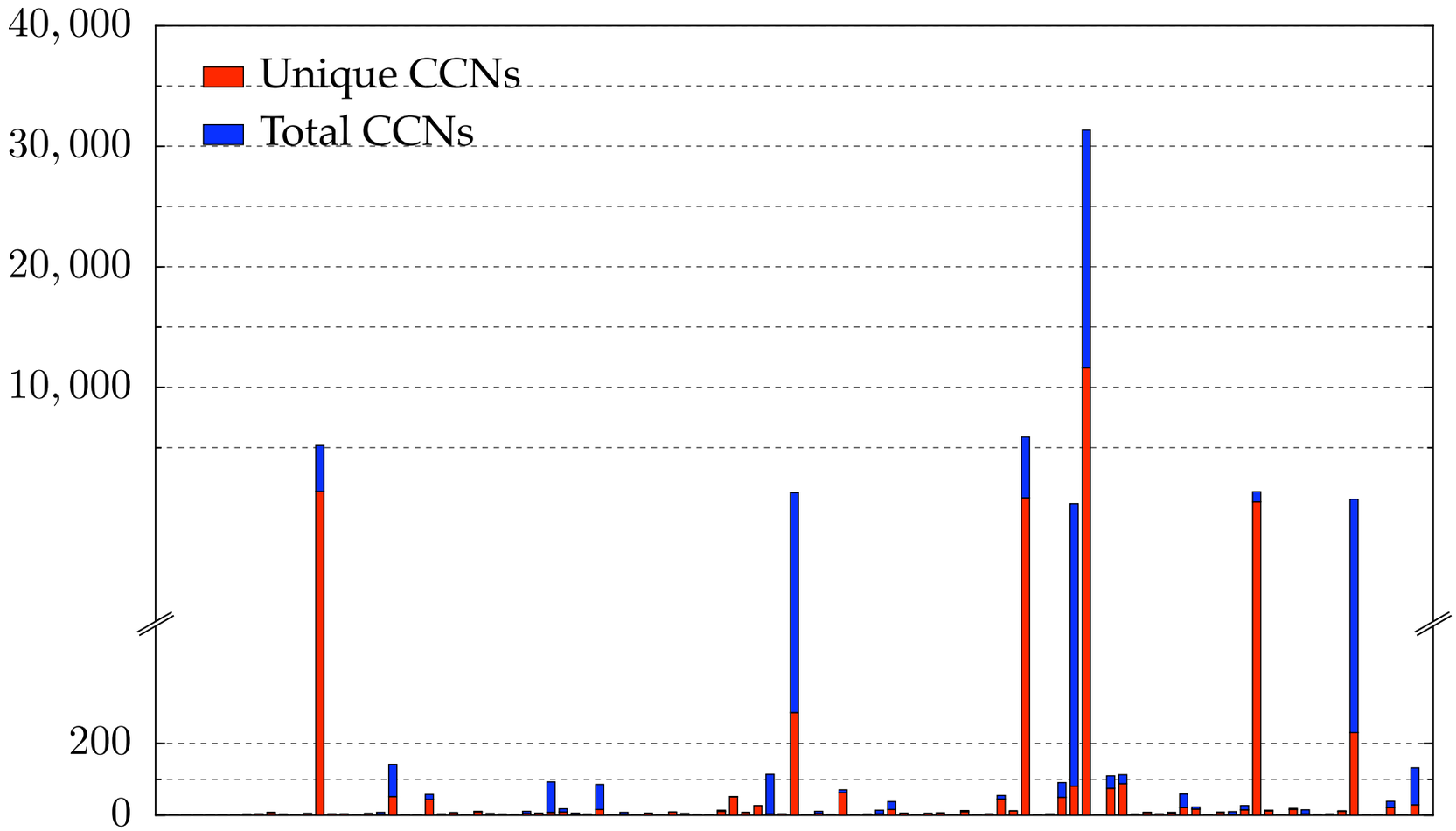
Second strategy:

- 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, we interviewed law enforcement regarding their use of forensic tools.

Law enforcement officers wanted a *highly automated* tool for finding:

- Email addresses & Credit card numbers (including track 2 information)
- Phone numbers, GPS coordinates & EXIF information from JPEGs
- Search terms (extracted from URLs)
- 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
- Run at maximum I/O speed of physical drive
- Never crash

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++, GNU flex and Java (GUI)

- Command-line tool.
- Linux, MacOS, Windows (compiled with mingw)
- BEViewer command-line tool and views results

Key Features:

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

http://www.nps.edu/

202-555-1212
user@domain.com

202-555-1212

http://www.nps.edu/

user@domain.com

bulk_extractor: system diagram

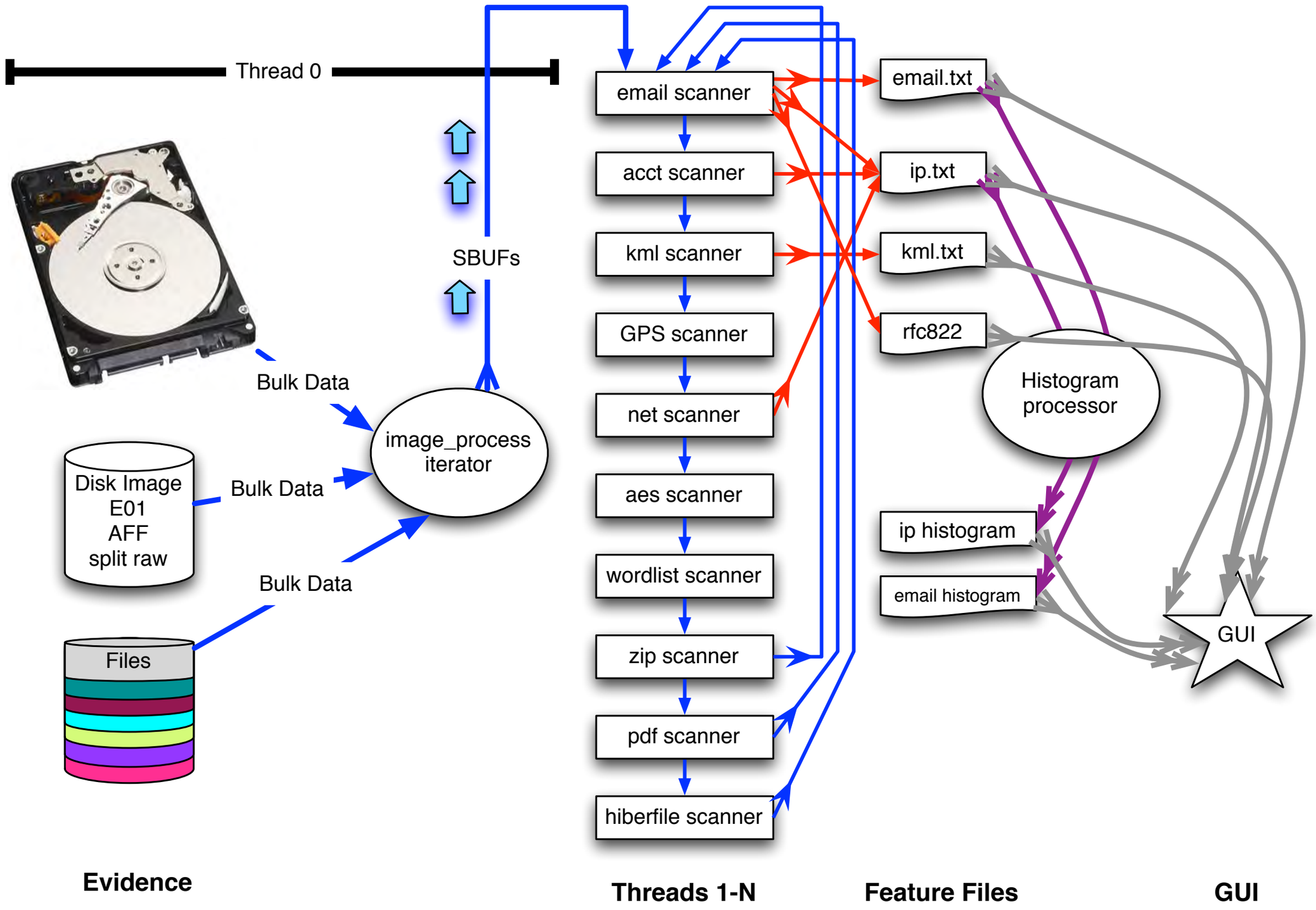
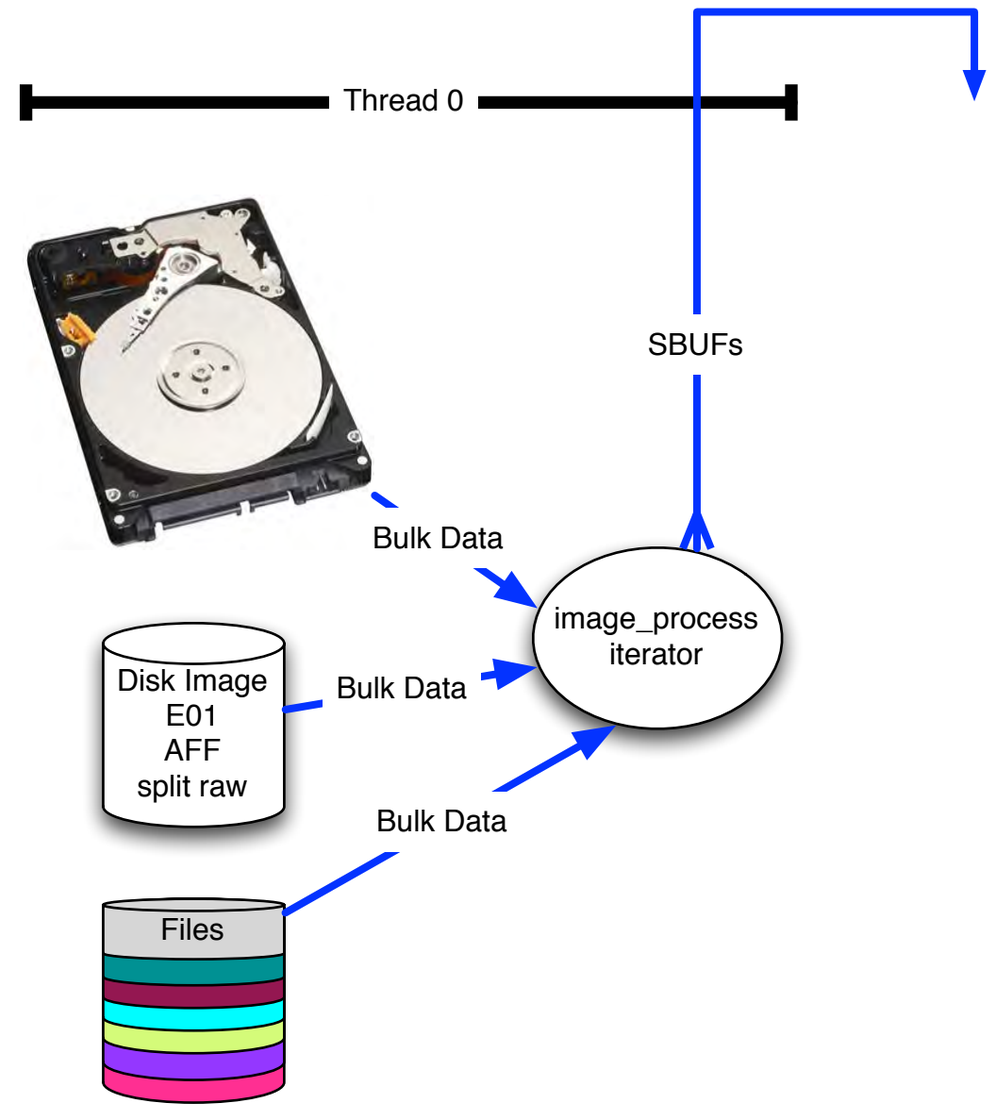


image processing
C++ iterator handles disks, images and files

Works with multiple disk formats.

- E01
- AFF
- raw
- split raw
- individual disk files



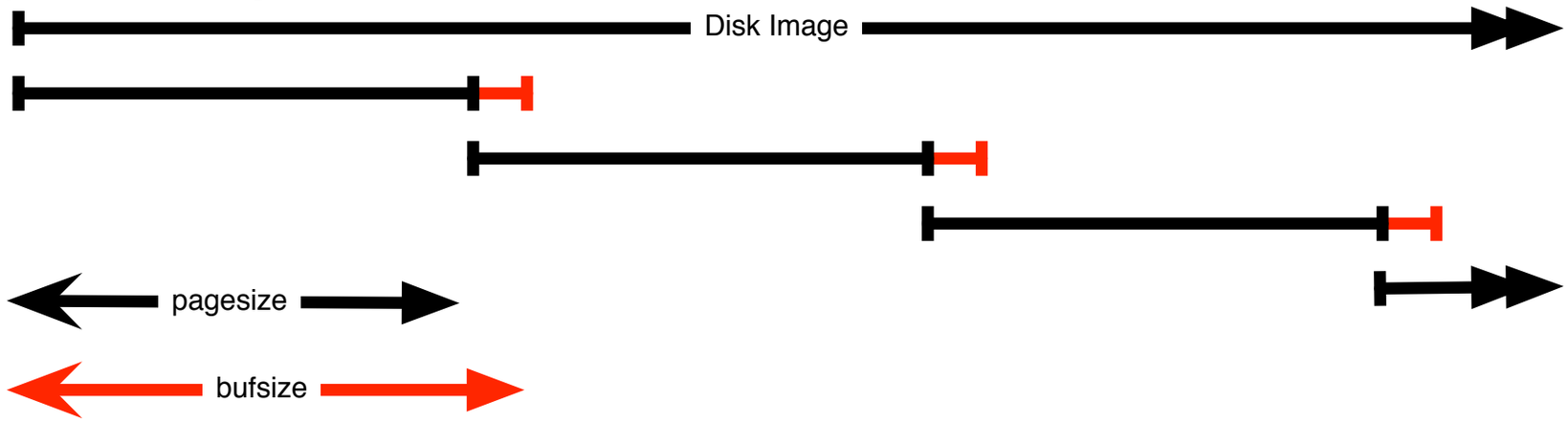
Evidence

We chop the 1TB disk into 65,536 x 16MiB “pages” for processing.

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

The overlap area is called the *margin*.

- Each sbuf can be 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.

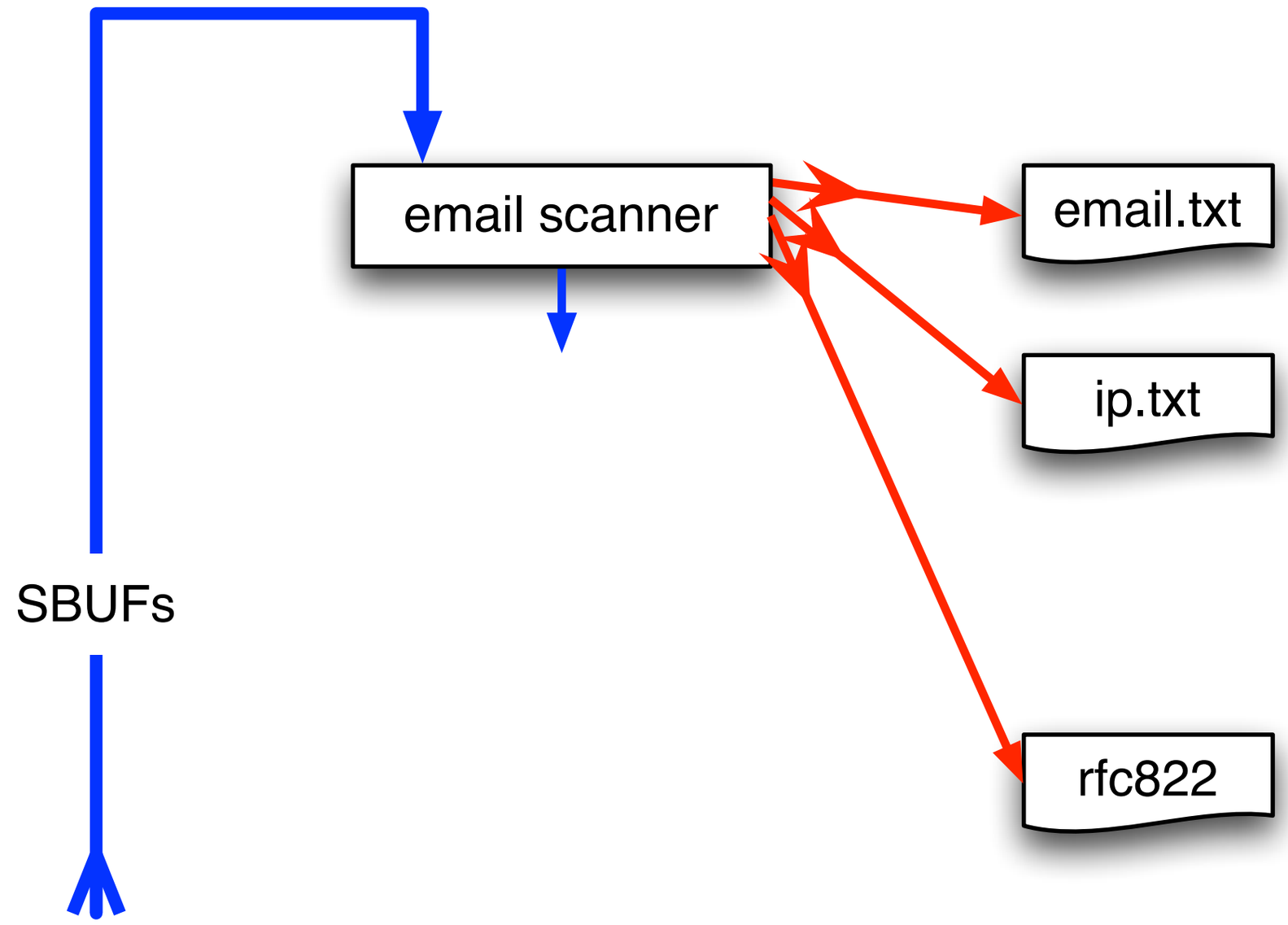
Scanners process each page and extract features

scan_email is the email scanner.

- inputs: **sbuf** objects

outputs:

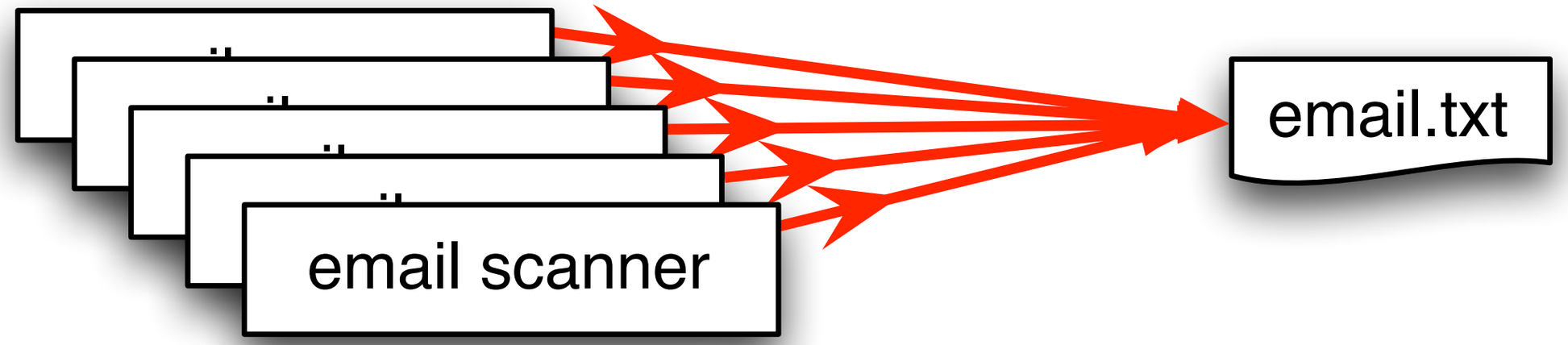
- **email.txt**
  - —*Email addresses*
- **rfc822.txt**
  - —*Message-ID*
  - —*Date:*
  - —*Subject:*
  - —*Cookie:*
  - —*Host:*
- **domain.txt**
  - —*IP addresses*
  - —*host names*



The *feature recording system* saves features to disk.

Feature Recorder objects store the features.

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



Features are stored in a *feature file:*

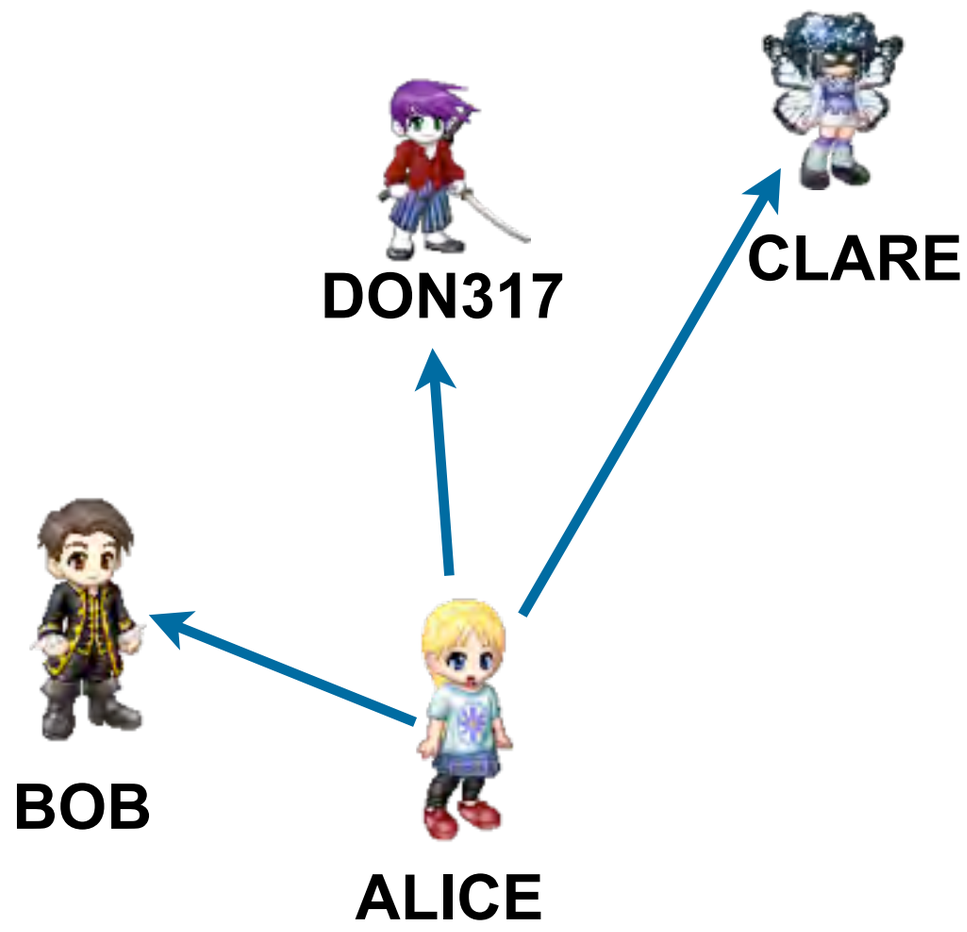
```
48198832  domexuser2@gmail.com    tocol>____<name>domexuser2@gmail.com/Home</name>____
48200361  domexuser2@live.com     tocol>____<name>domexuser2@live.com</name>____<pass
48413829  siege@preoccupied.net   siege) O'Brien <siege@preoccupied.net>_hp://meanwhi
48481542  danilo@gnome.org        Danilo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

Feature histograms are created at the end of processing. They are a powerful tool for understanding evidence.

Email address histogram allows us to rapidly determine:

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



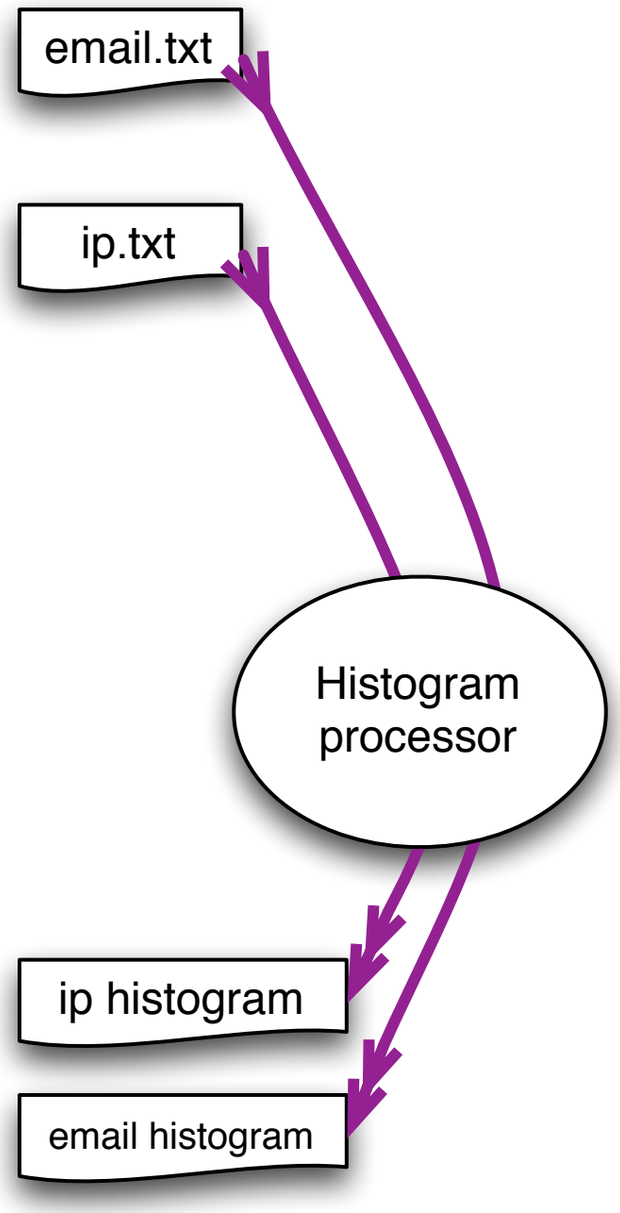
Drive #51 (Anonymized)

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

Histograms can be based on features or regular expression extracts from features.

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

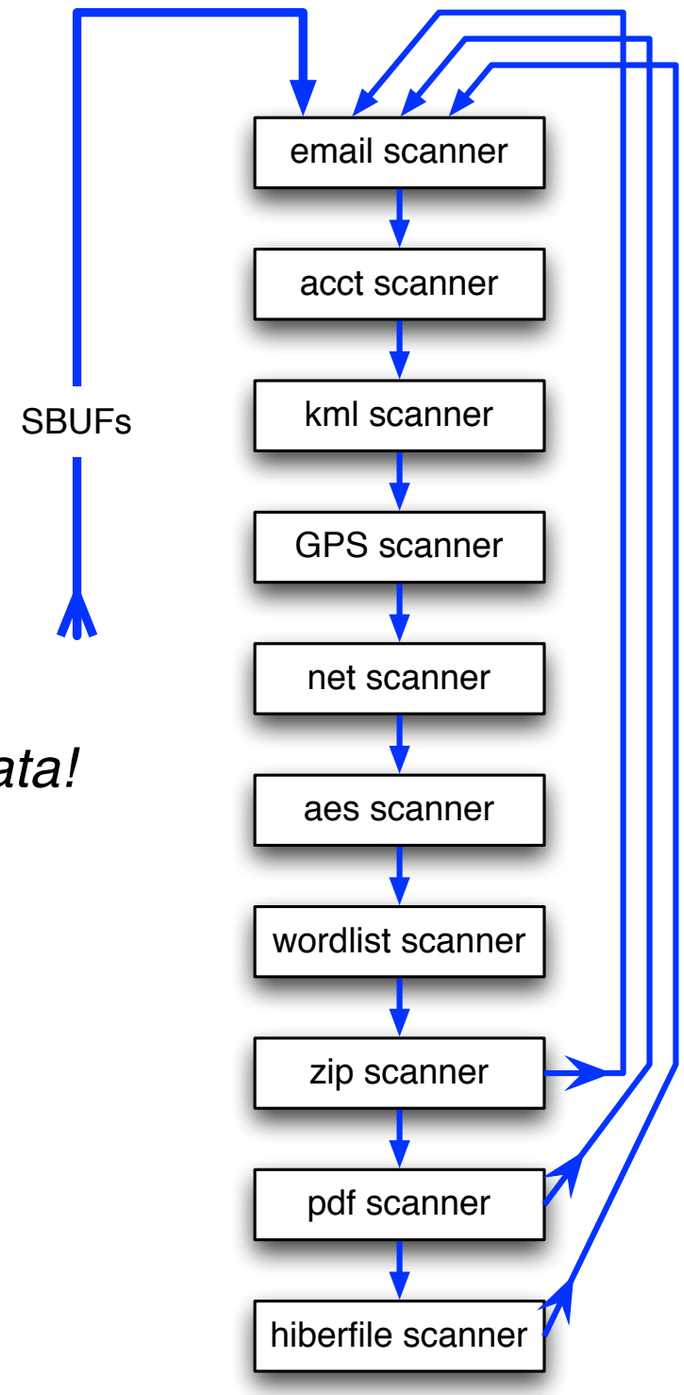
- Useful for debugging.
- AES key scanner is *very slow* (off by default)

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!*

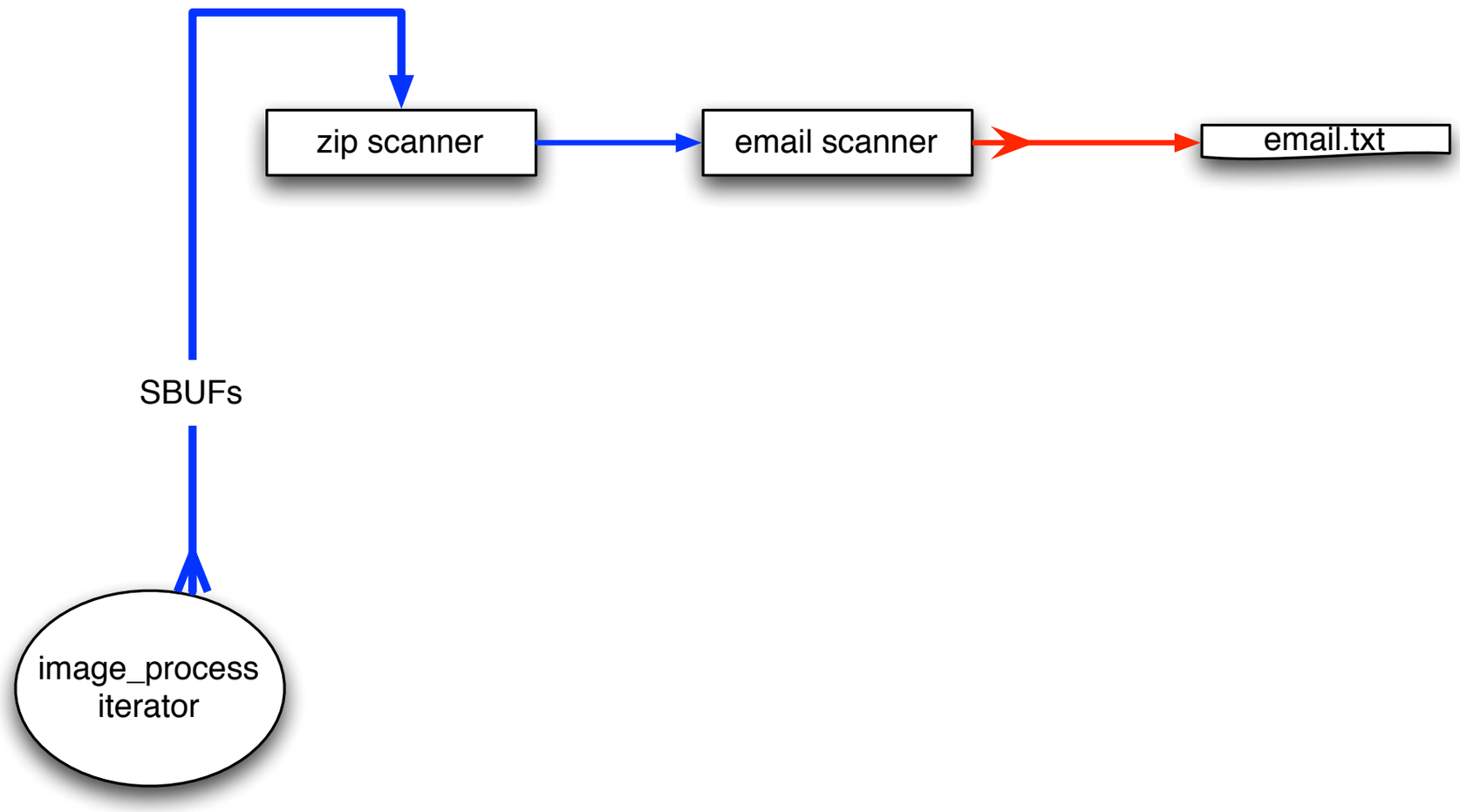
Recursion used for:

- 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."

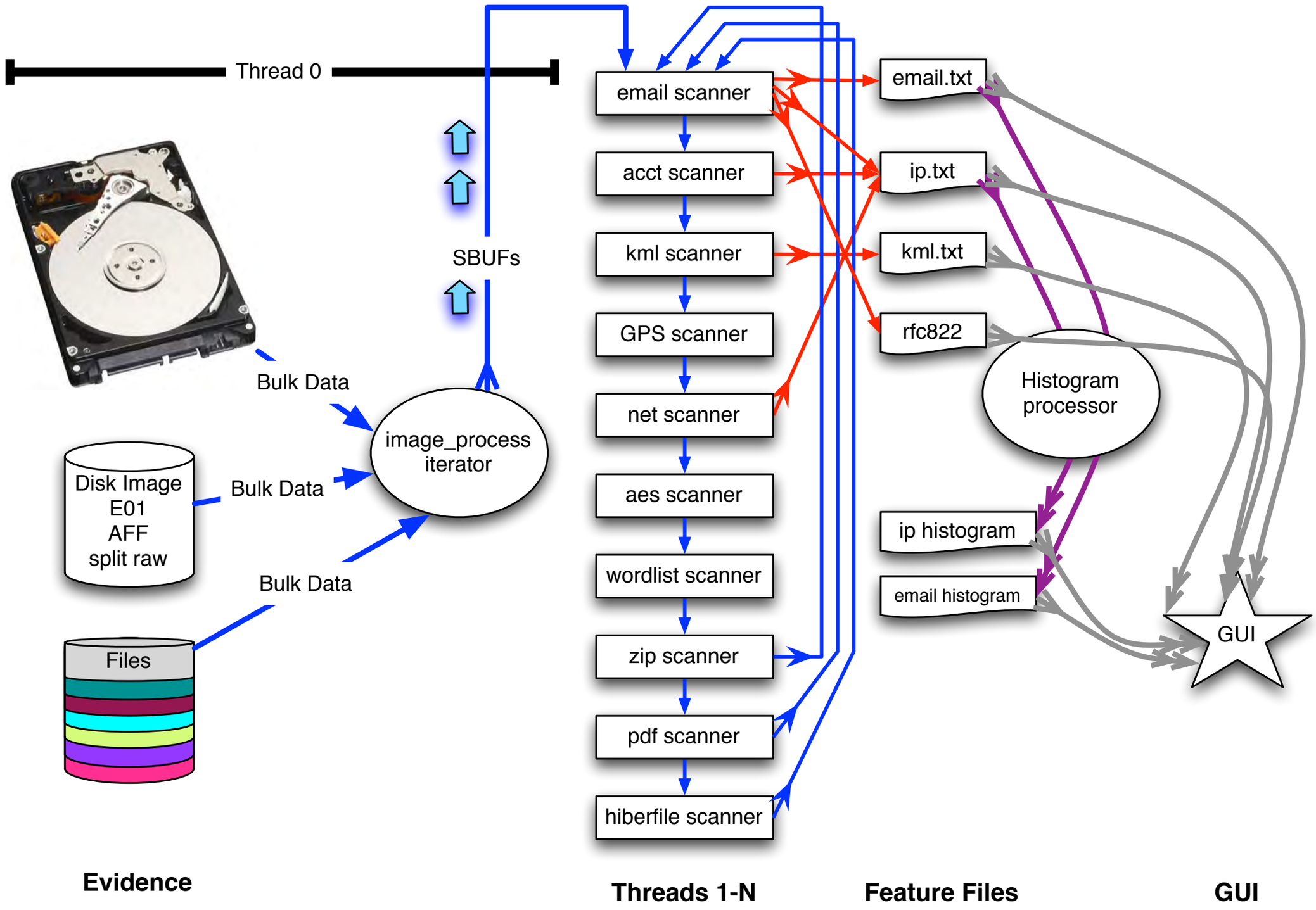
Consider an HTTP stream that contains a GZIP-compressed email:



We can represent this as:

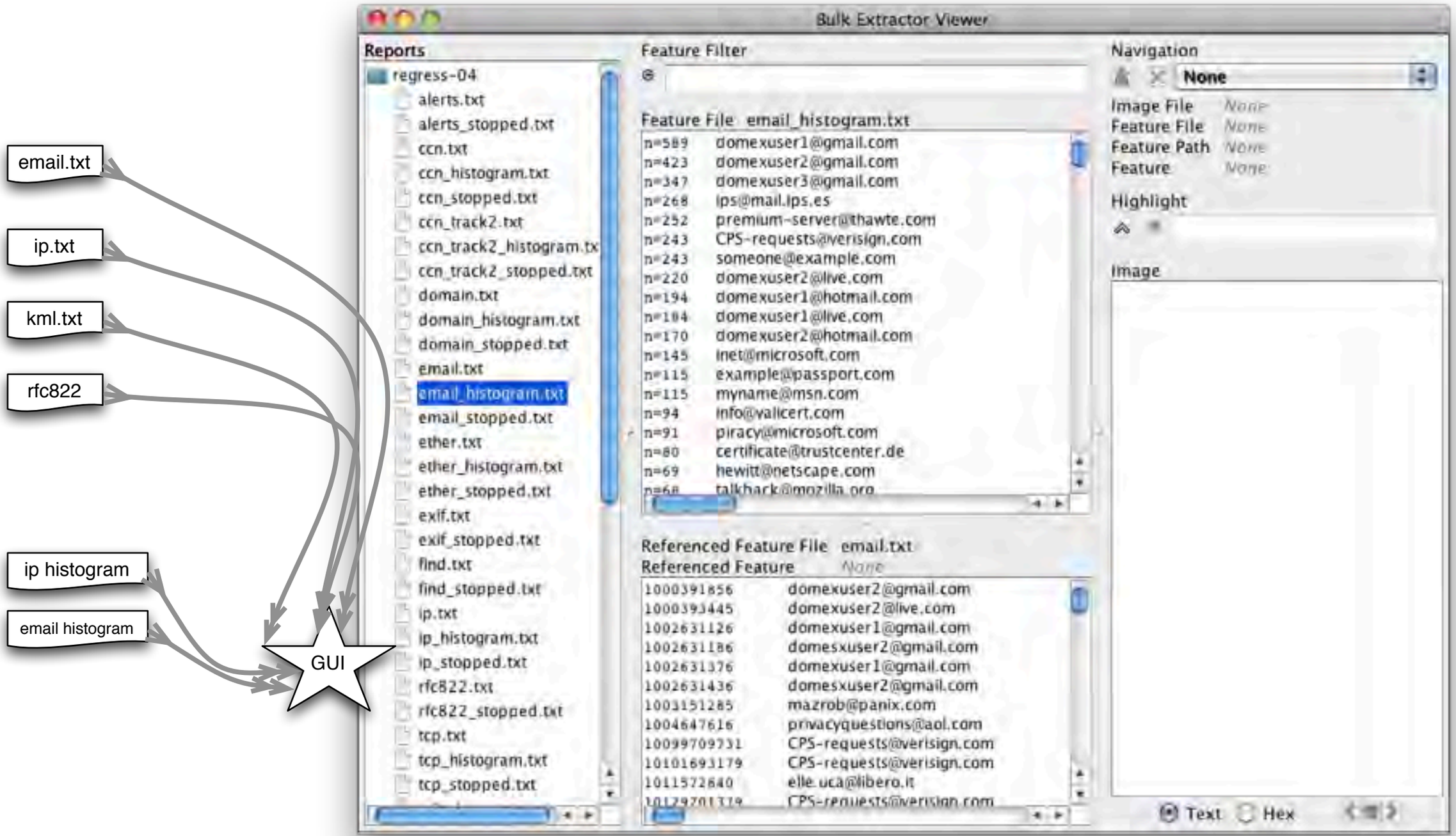
```
11052168704-GZIP-3437   live.com    eMn='domexuser1@live.com';var srf_sDispM
11052168704-GZIP-3475   live.com    pMn='domexuser1@live.com';var srf_sPreCk
11052168704-GZIP-3512   live.com    eCk='domexuser1@live.com';var srf_sFT='<
```

Integrated design, but compact.
15,500 lines of code; 33 seconds to compile on an i5



BEViewer: GUI runs on Windows, Mac & Linux Launches bulk_extractor; views results

Uses bulk_extractor to decode forensic path



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

bulk_extractor 1.2 uses feature files as the stop list.

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

Extending bulk_extractor with Plug-ins and post-processing.

Plugins written in C++

Plugins are distributed as *shared libraries*.

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

Plugins must support a single function call:

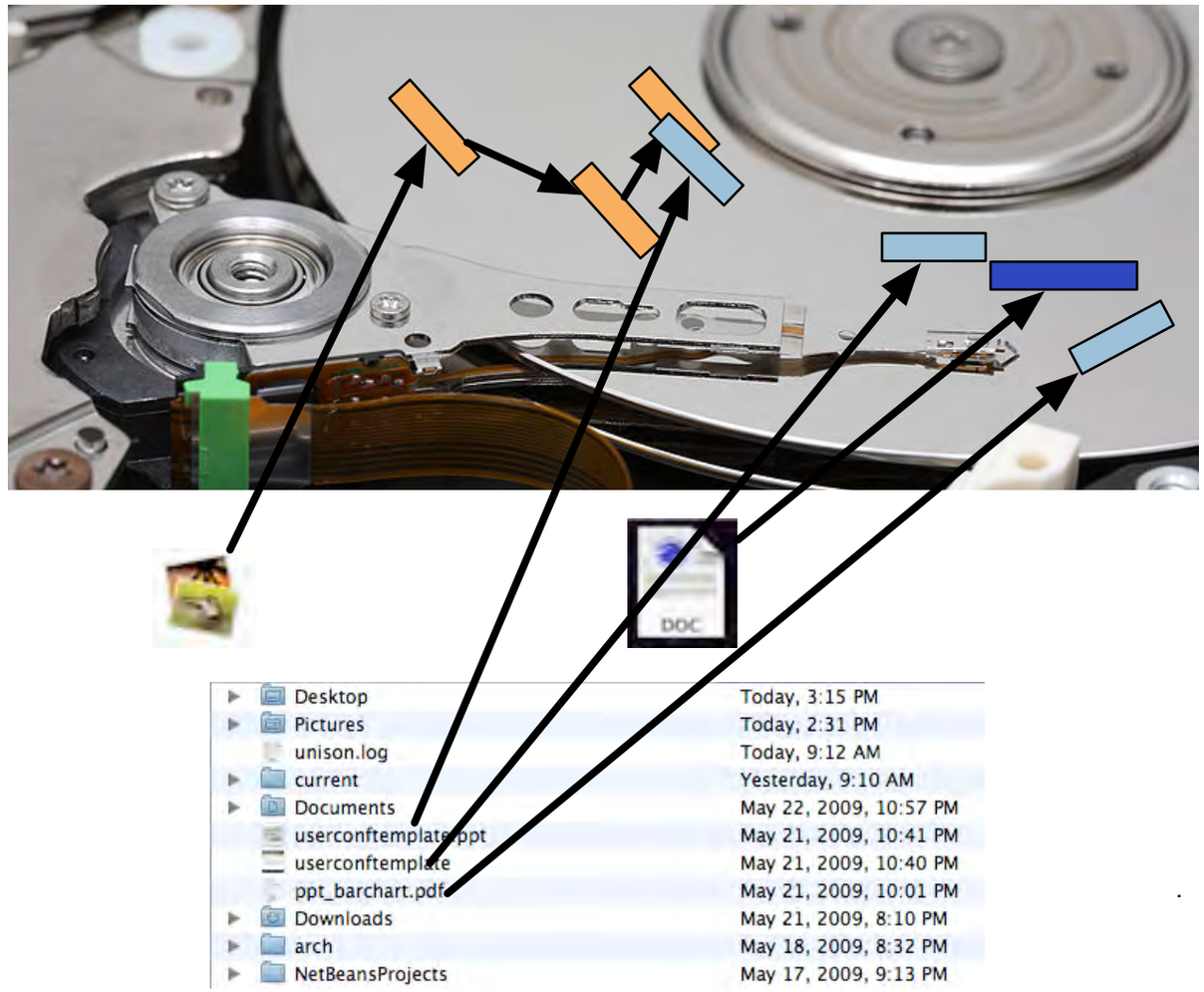
```
void scan_bulk(const class scanner_params &sp,
               const recursion_control_block &rcb)
```

- 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 **tcpflow**

identify_filenames.py: Determines the file name for each feature.

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.

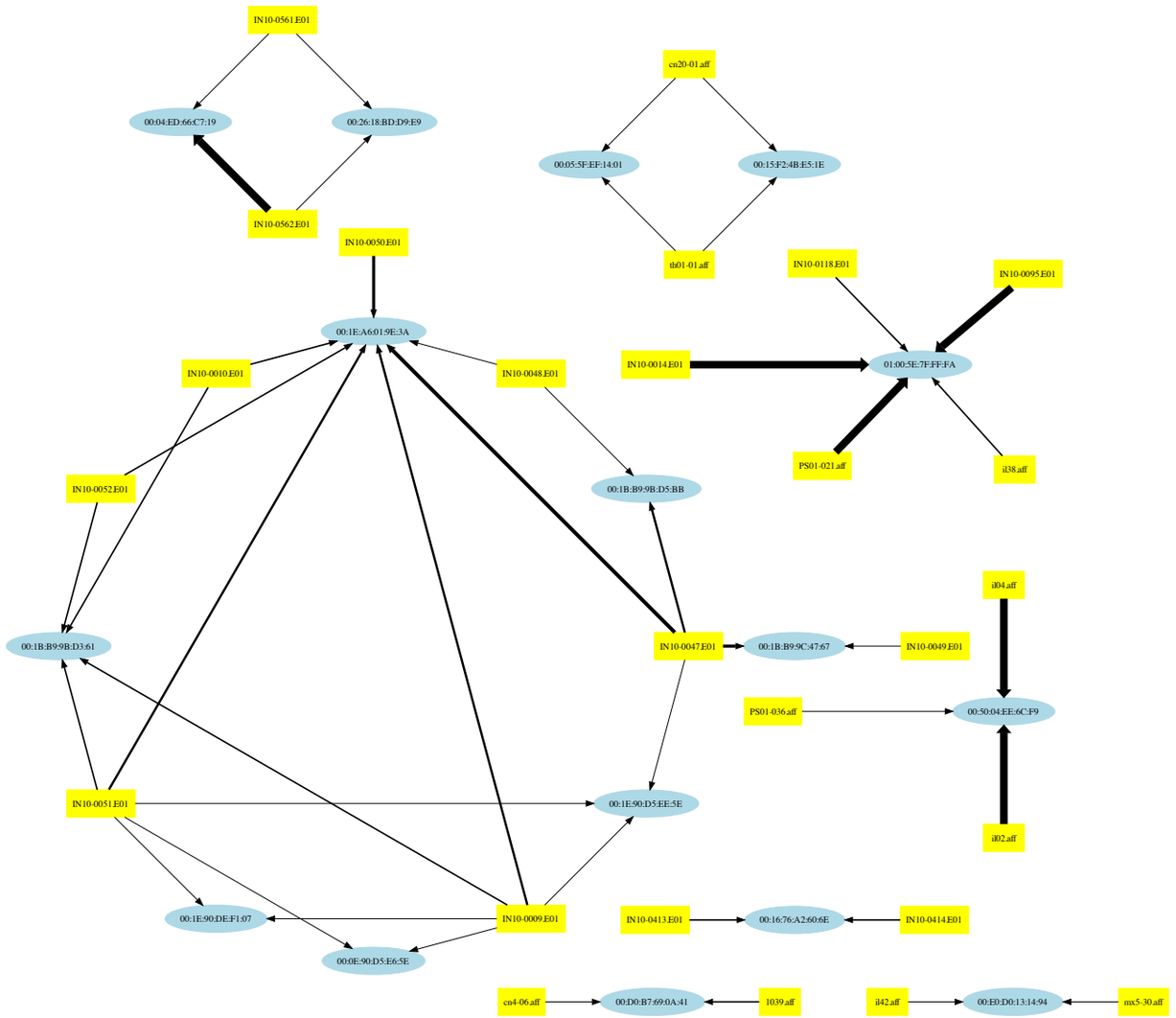
- Designed for timeline analysis.
- Developed with analysts.
- Reports “what’s changed.”
  - *Reporting “what’s new” turned out to be more useful.*
  - *“what’s missing” includes data inadvertently overwritten.*

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:



bulk_extractor: current status and future goals

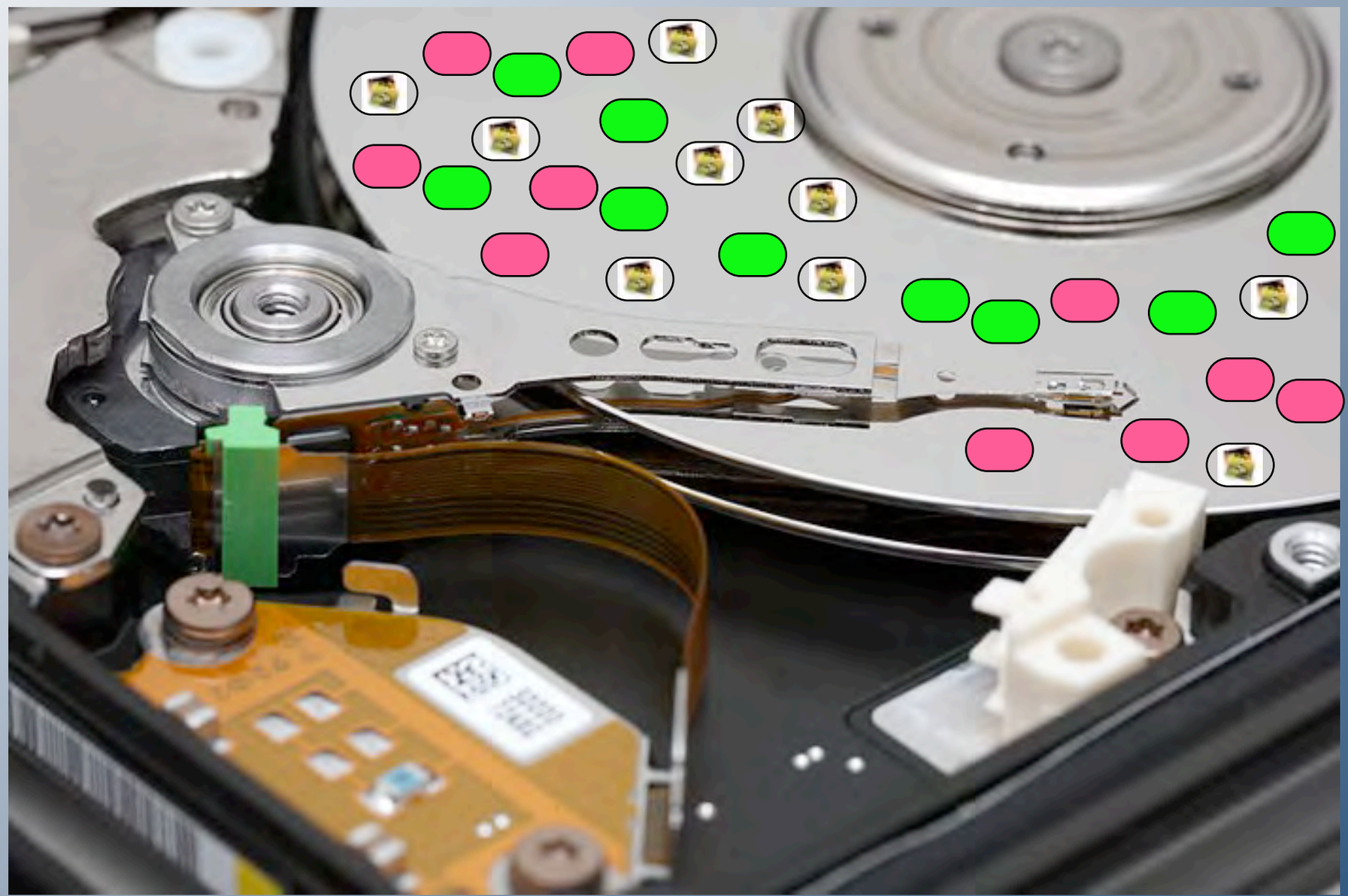
Scanners:

- accts
- aes
- base64
- ccns
- email headers
- exif
- find
- gps
- gzip
- net
- hiberfile
- httpheader
- json
- kml
- net
- pdf
- **vcard***
- winprefetch
- wordlist
- zip
- **windir***

Currently under development:

- bulk (detects encrypted data)
- RAR, RAR2
- LZMA
- BZIP
- Improved handling of Unicode.
- NTFS

Automated regression testing for new releases.



Random Sampling

Can we analyze a hard drive in five minutes?

US agents encounter a hard drive at a border crossings...

Searches turn up rooms filled with servers….

If it takes 3.5 hours to read a 1TB hard drive, what can you learn in 5 minutes?

		
Minutes	208	5
Max Data	1 TB	36 GB
Max Seeks		90,000

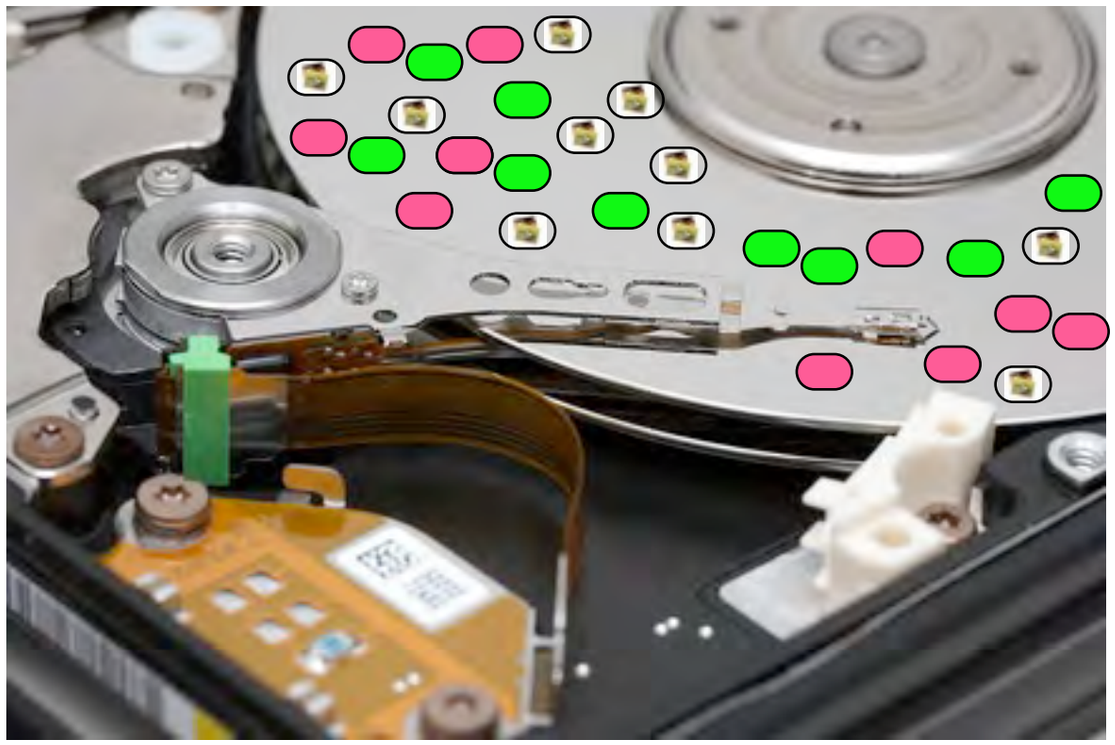
36 GB is a lot of data!

- ≈ 2.4% of the disk
- But it can be a *statistically significant sample*.

We can predict the statistics of a *population* by sampling a *randomly chosen sample.*

US elections can be predicted by sampling a few thousand households:

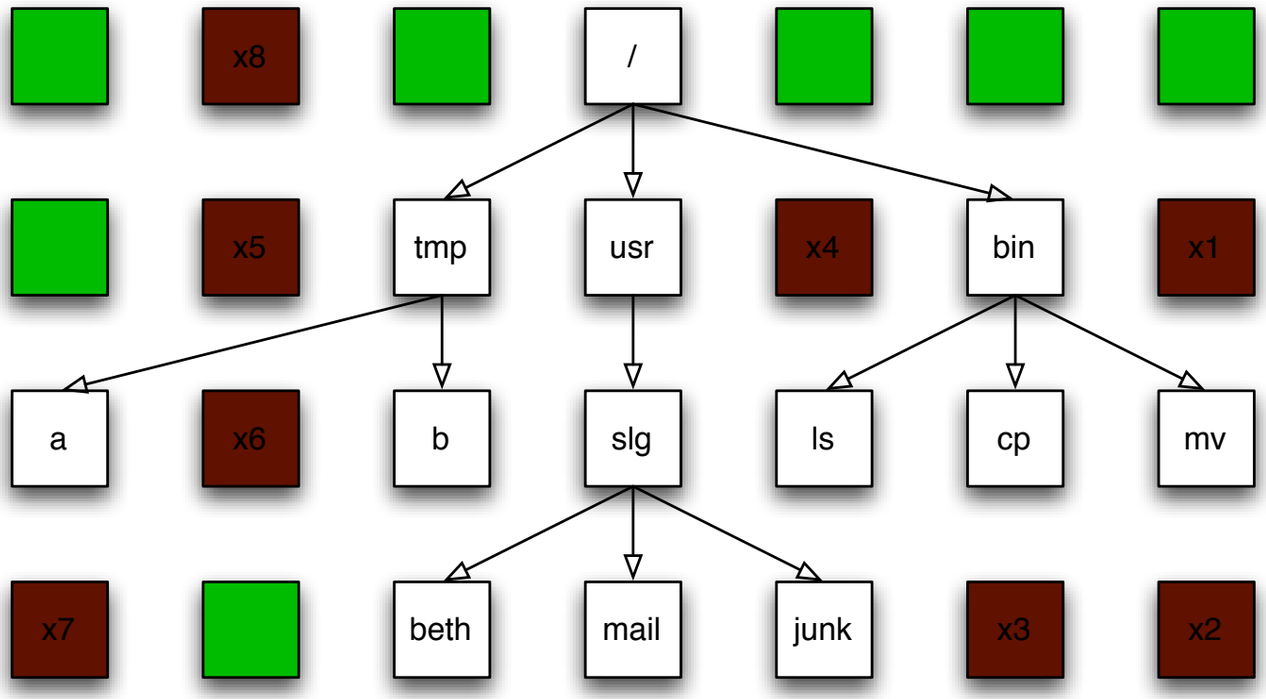
Hard drive contents can be predicted by sampling a few thousand sectors:

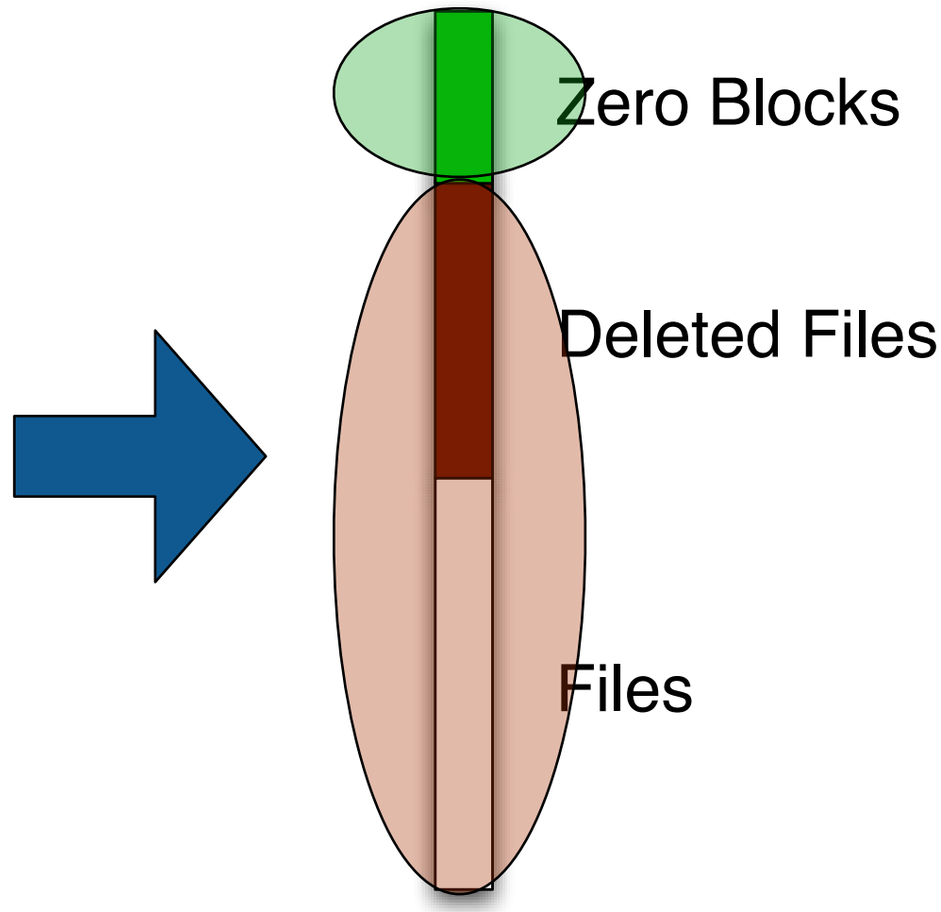


The challenge is identifying *likely voters.*

The challenge is *identifying the sectors* that are sampled.

Sampling can distinguish between "zero" and data. It can't distinguish between resident and deleted.

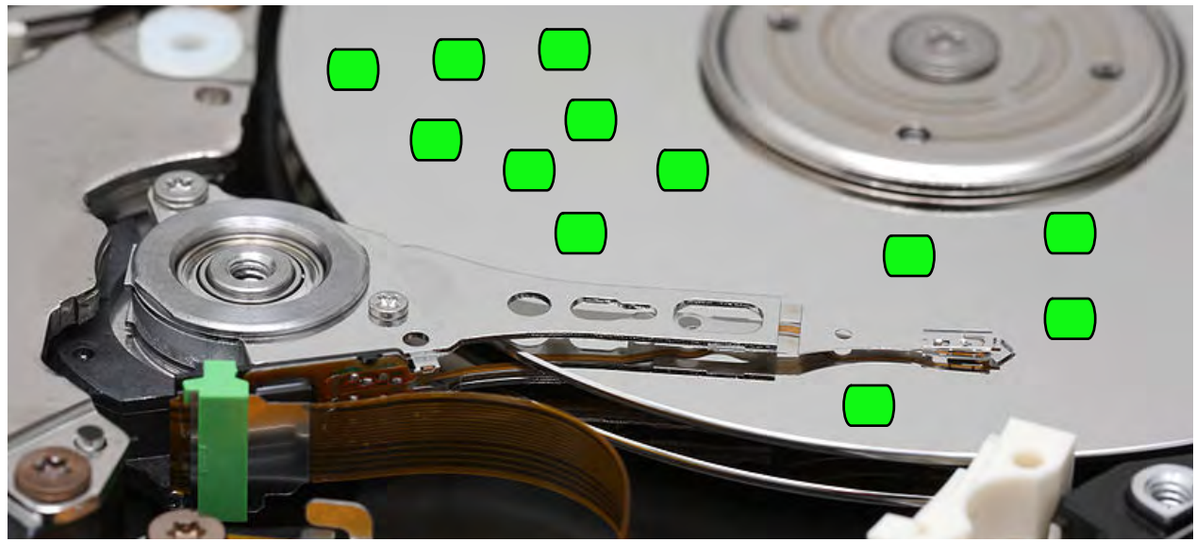




Simplify the problem.
Can we use statistical sampling to verify wiping?

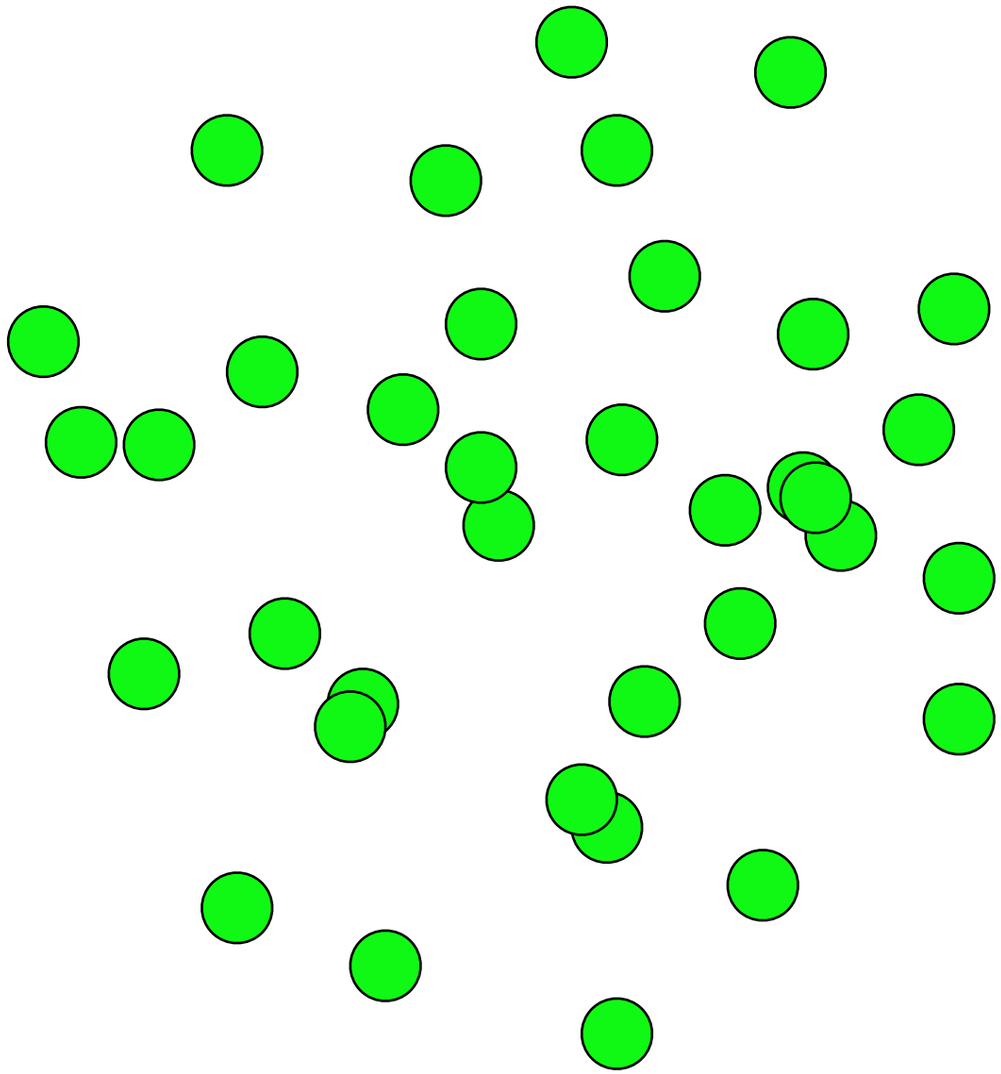
Many organizations discard used computers.

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

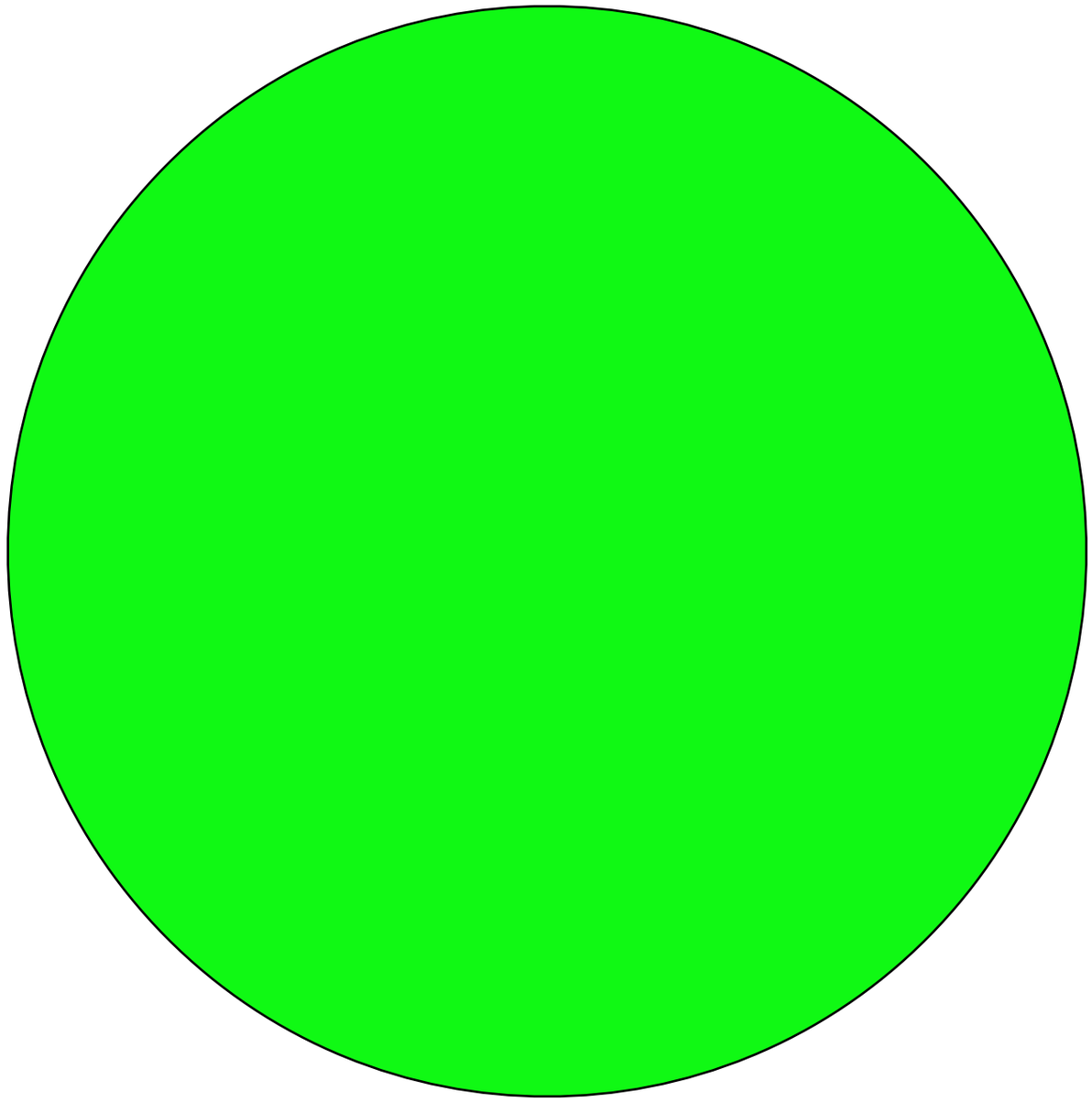


A 1TB drive has 2 billion sectors.
What if we read 10,000 and they are all blank?

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



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

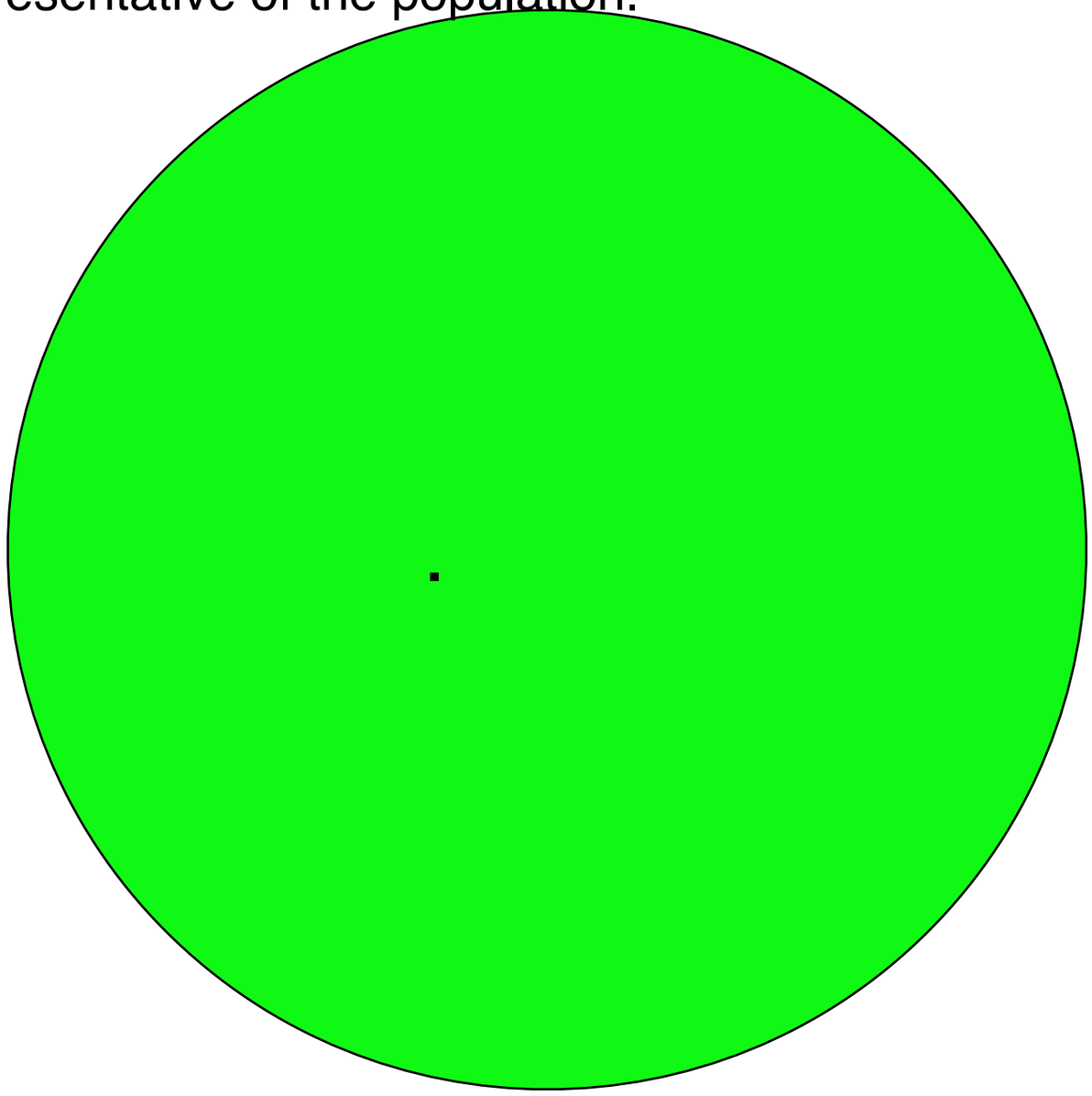


Chances are good that they are all blank.

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

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

- The sample is representative of the population.

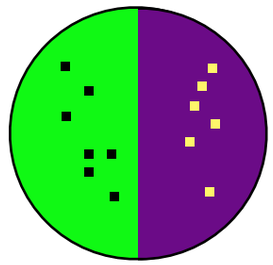


We will only find that 1 sector with exhaustive search.

What about other distributions?

If the disk has 1,000,000,000 blank sectors (1,000,000,000 with data)

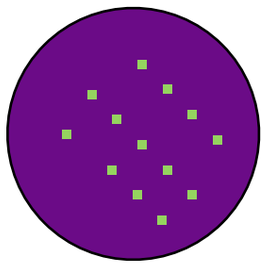
- The sampled frequency should match the distribution.
- *This is why we use random sampling.*



If the disk has 10,000 blank sectors (1,999,990,000 with data)

— and all these are the sectors that we read???

- We are incredibly unlucky.
- ***Somebody has hacked our random number generator!***



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

Assume a 1TB disk has 10MB of data.

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

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

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

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

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

first pick **second pick** **Odds we may have missed something**

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

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

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

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

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

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

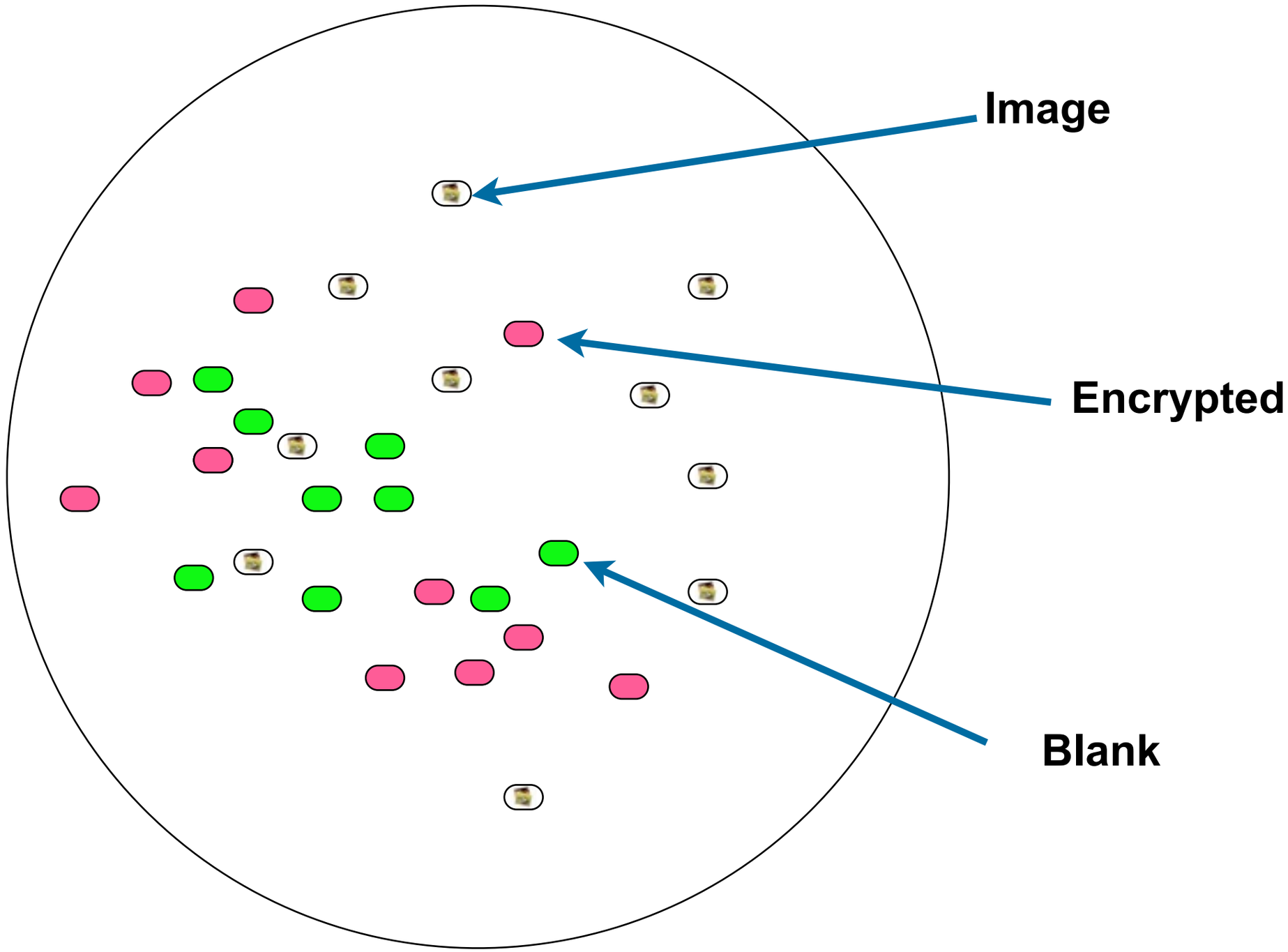
—So pick 500,000 random sectors. If they are all NULL, then the disk has p=(1-.00673) chance of having 10MB of non-NULL data.

We can use this same technique to calculate the size of the TrueCrypt volume on this iPod.

It takes 3+ hours to read all the data on a 160GB iPod.

- Apple bought very slow hard drives.

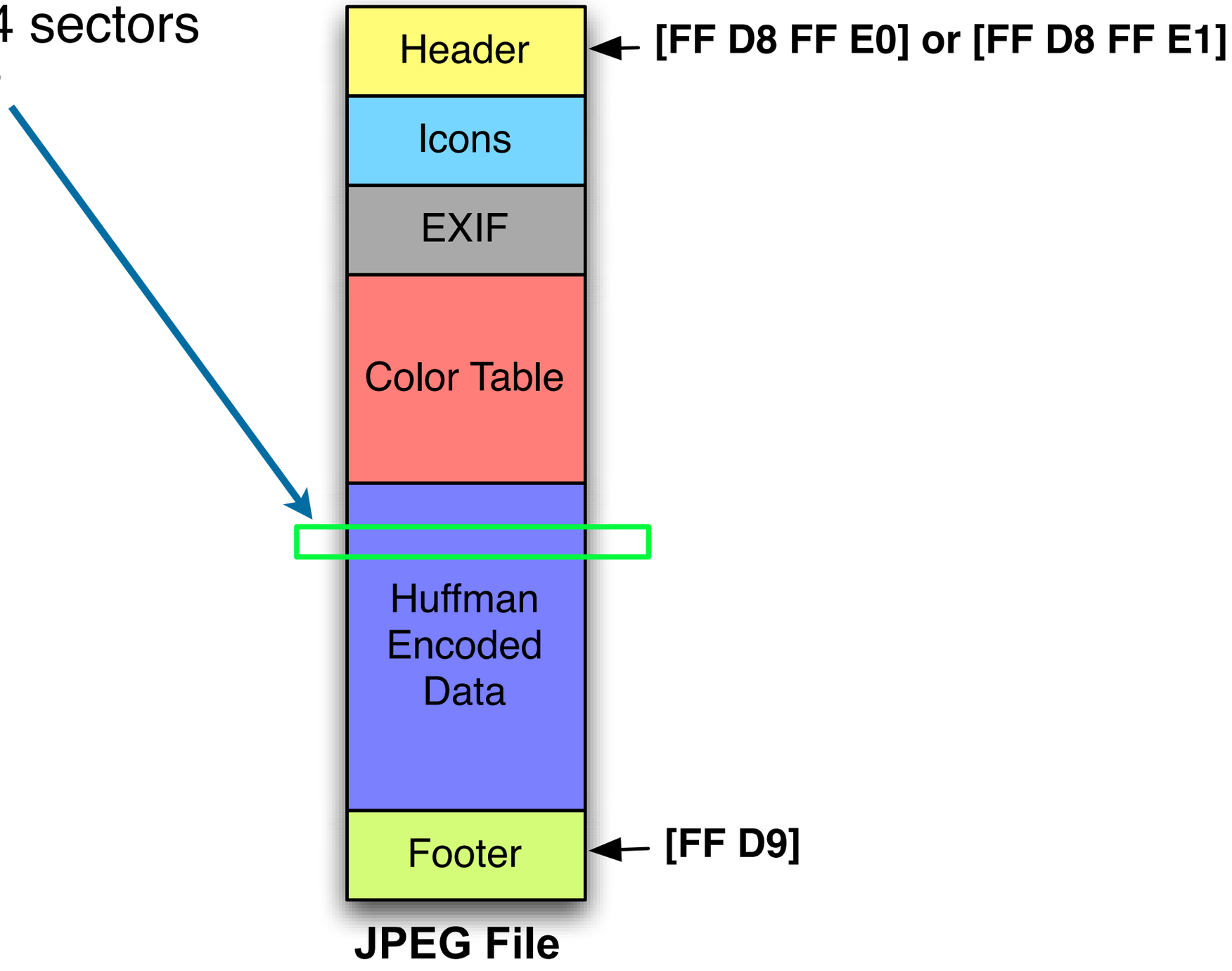
We can get a statistically significant sample in two minutes.



The % of the sample will approach the % of the population.

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

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



Approach #1: NPS has developed five special-purpose fragment discriminators.

JPEG — High entropy and FF00 pairs.

MPEG — Frames

Huffman-Coded Data — High Entropy & Autocorrelation

"Random" or "Encrypted" data — High Entropy & No autocorrelation

Distinct Data — a block from an image, movie, or encrypted file.

208 distinct 4096-byte block hashes

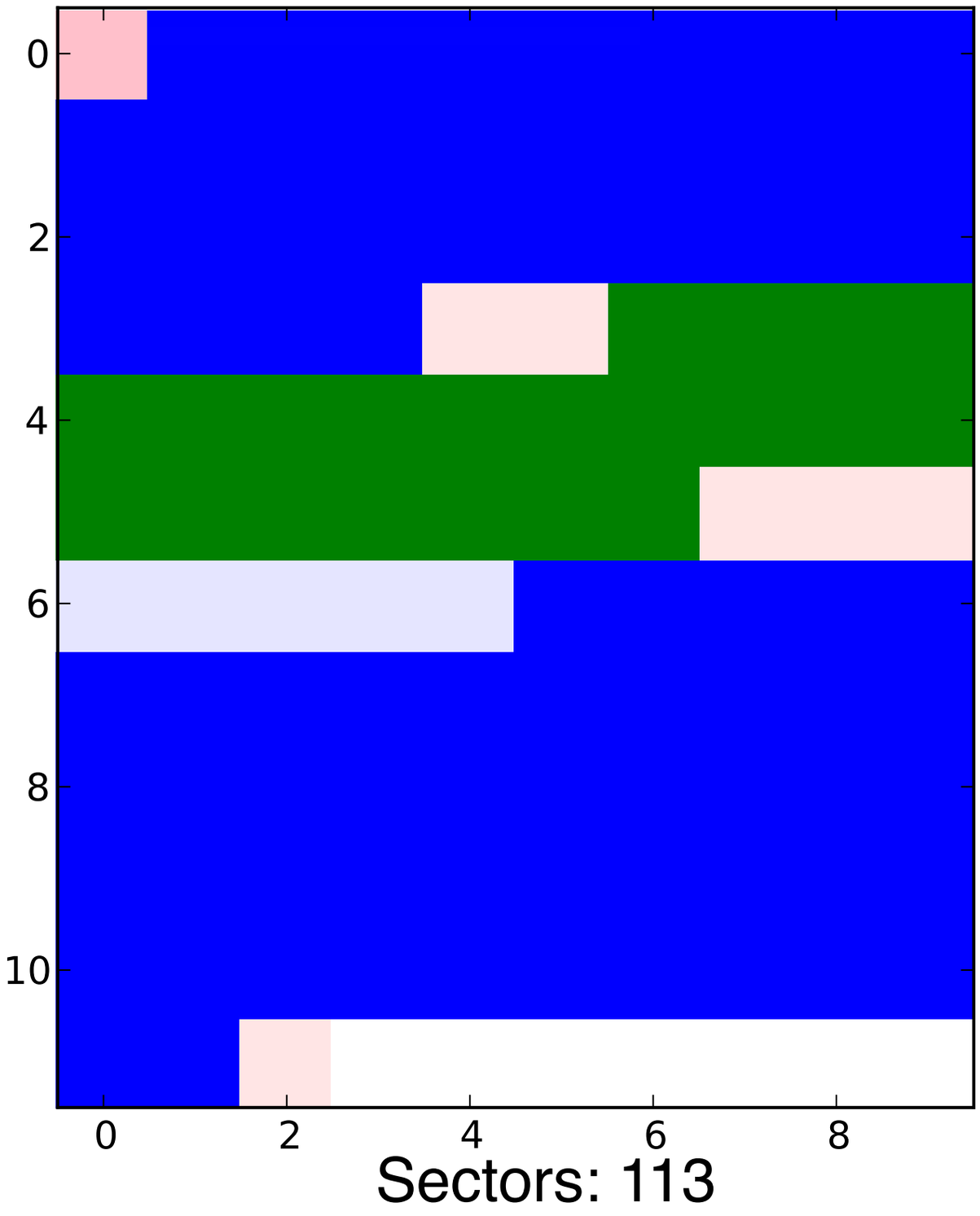
The JPEG detector understands different parts of the file.

Bytes: 31,046

Nearly 50% of this 57K file identifies as “JPEG”

000897.jpg

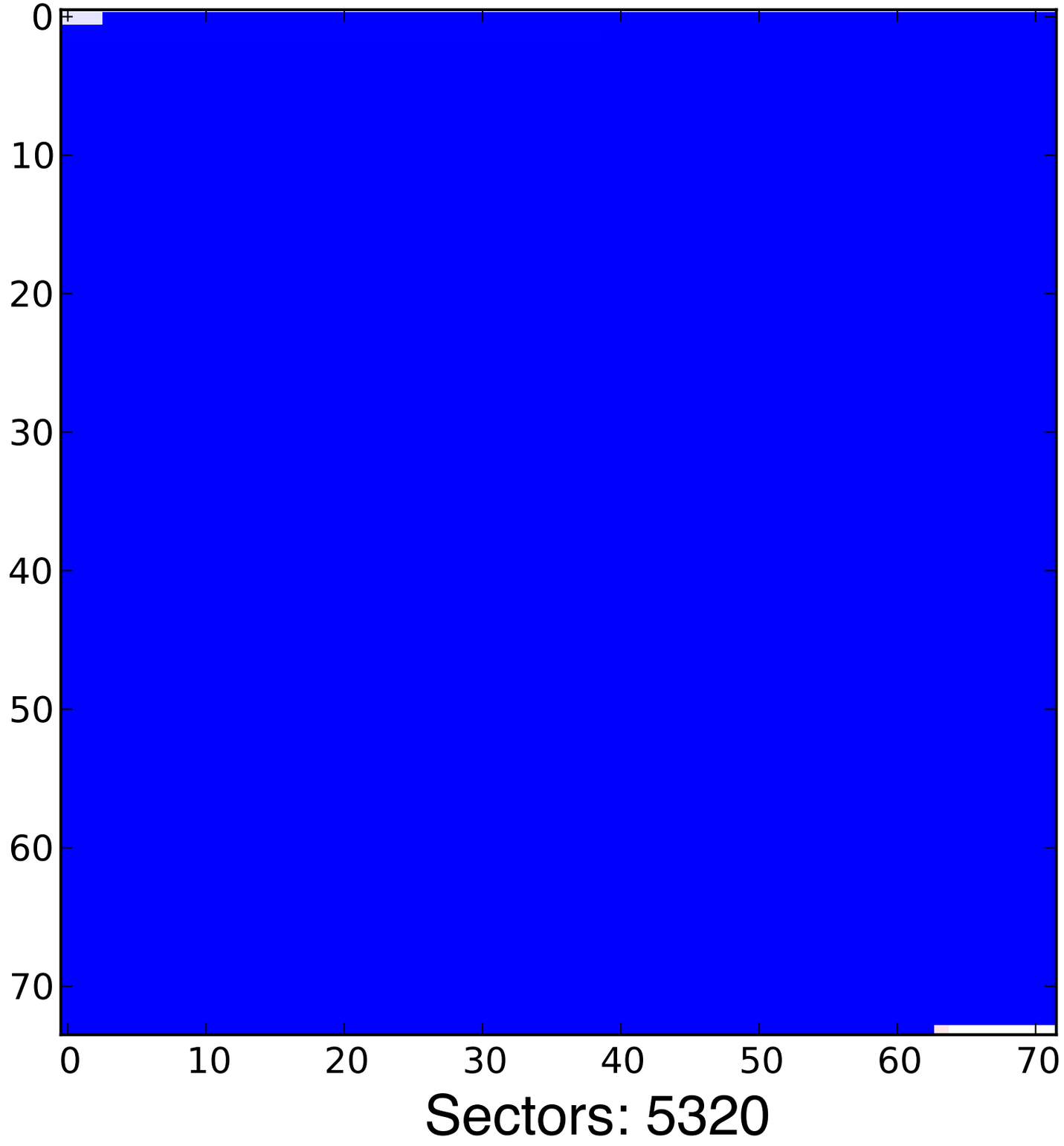
Bytes: 57596



Nearly 100% of this file identifies as “JPEG.”

000888.jpg

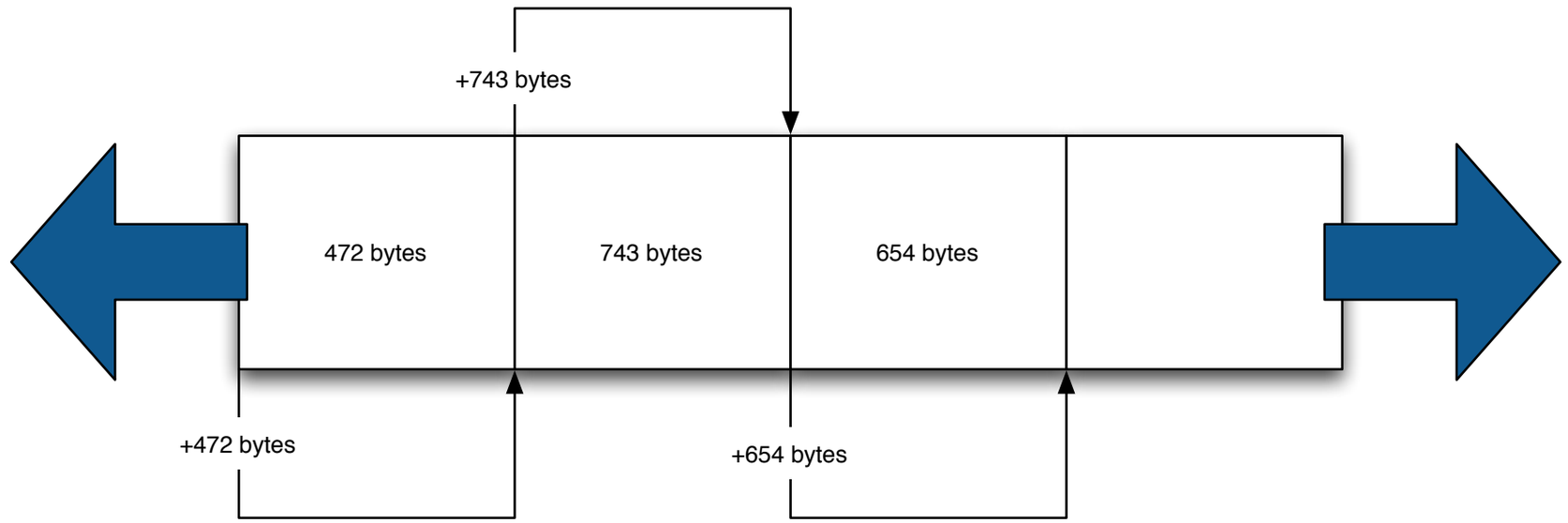
Bytes: 2,723,425



MPEG files can be readily identified with frame detection.

Each frame has a header and a length.

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

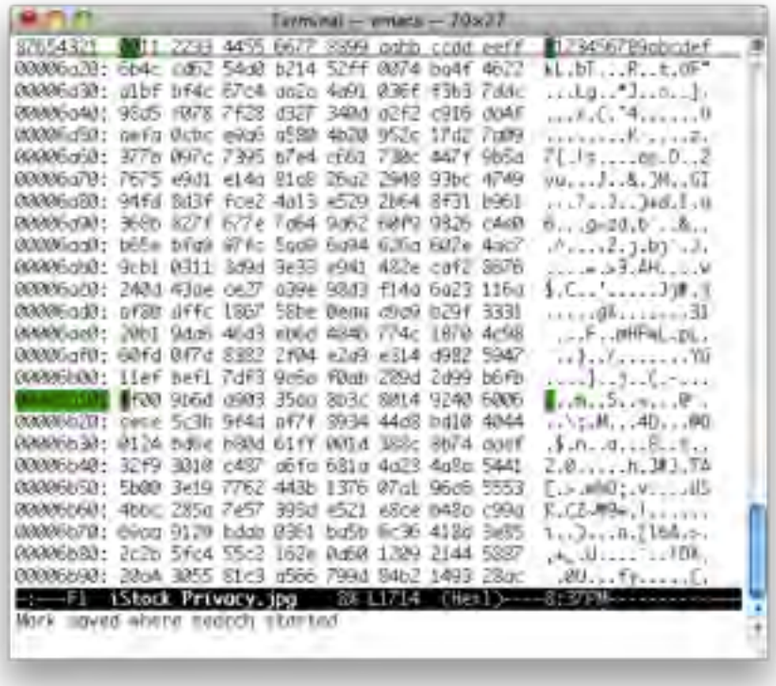


This approach can also reconstruct fragmented video.

Approach #2:
Bigram analysis with support vector machines.

“LIFT” — Learning to Identify File Types

- Machine learning / Support Vector Machines
- Developed at CMU (2009-2011) with DoD funding
- Abandoned by CMU; now maintained by NPS.

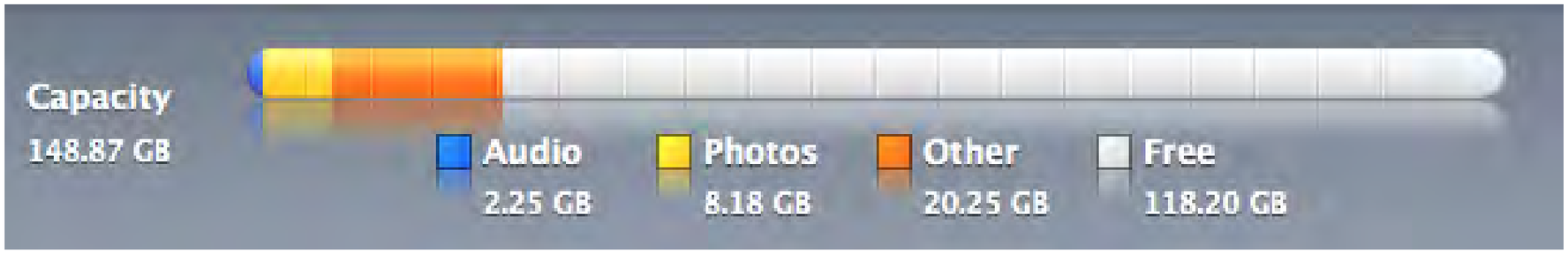


LIFT *should* be incorporated into bulk_extractor 1.3.

- Key obstacle: LIFT is not multi-threaded.

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

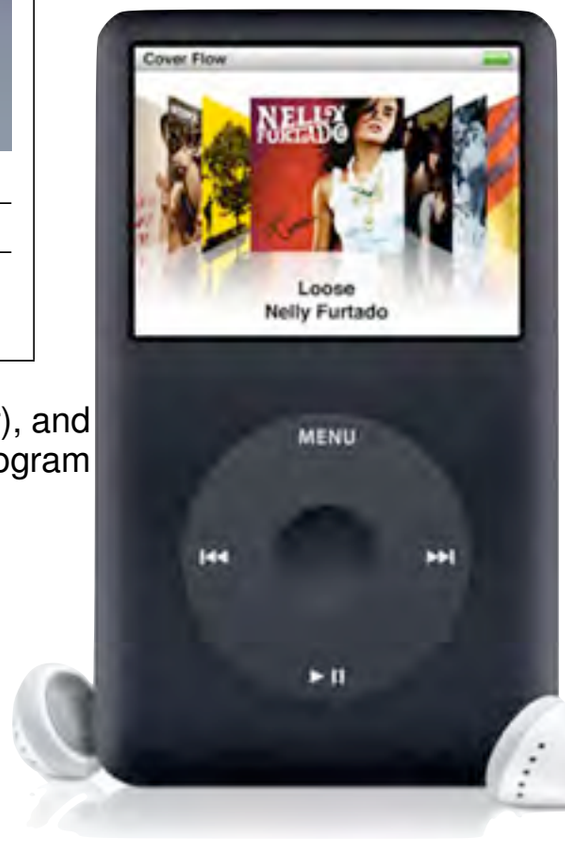


Figure 1: Usage of a 160GB iPod reported by iTunes 8.2.1 (6) (top), as reported by the file system (bottom center), and as computing with random sampling (bottom right). Note that iTunes usage actually in GiB, even though the program displays the “GB” label.

We accurately determined:

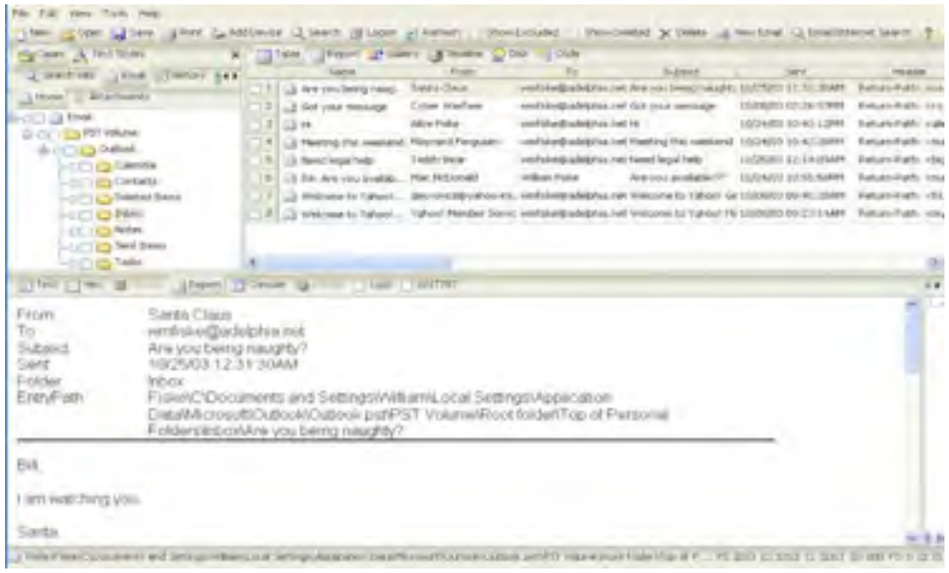
- % of free space; % JPEG; % encrypted

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

<dfxml>

<fileobject>
<creator>

Today's forensic tools are designed for investigations.



Encase:
- GUI Closed Source
- EnScript

SleuthKit:
- Command-line Open Source
- C/C++ API

These tools are great for:

- File recovery
- Search
- Looking at disk sectors

Not so great for automation, interoperability, or research.

Input formats are large and require frequent re-processing

Disk Images:

- Copying TBs takes time and is increasingly error-prone.
- File extraction has to be repeated (and needs to be repeatable)

PCAP processing — done differently by different tools.

- Sessionization
- Decompression
- Decryption

Hash Sets

- No systematic way for labeling or annotating.
- No easy way to expand beyond MD5.

Forensic processing (today) is (highly) tool dependent

EnCase:

- Windows GUI makes it difficult to create an end-to-end automated system.
- EnScript is not an industrial strength language
  - *Poorly documented*
  - *Not parallelized.*
  - *Limited to Windows*

SleuthKit:

- Widely used for processing on Linux / MacOS.
- Processing is not very high level — C++ / Python / Perl / Bash
- Limited support for some file systems.

Writing *programs* for these systems is hard:

- Many of the forensic tools are not designed for easy automation.
- Programming languages are *procedural* and *mechanism-oriented.*
- Data is separated from actions on the data.

Work product has limited use because of file formats

Microsoft Office Files (Word, Excel, PPT)

- Good for briefings.
- Excel has high value for analysts, but not for tool integration.

Adobe Portable Document Format

- Hard to recover data (“Dead Data”)
- Allows attaching files, but few people use this feature.

We rarely use these *output* as *input* for other tools.

Forensic outputs lack *provenance*.

Provenance is a “history of ownership or location.”

- Long history of provenance in the art community.
- Increasingly used in scientific computing.

In forensic processing, we would like to track:

- Where we got the data
  - *Sources*
  - *Classification levels*
- Tools that processed the data
  - *Version of the tool.*
  - *Compiler used to compile the tool.*
  - *Libraries linked with the tool.*
- Computer(s) on which processing was done.
  - *Operating System*

Although we track some of this now, we don’t do it in a way that is:

- Systematic
- Machine-readable

We need higher-level *formats* and *abstractions*.

Signatures

- parts of files
- n-grams
- piecewise hashes
- similarity metrics

File Metadata

- e.g. Microsoft Office document properties

File system metadata

- MAC times, etc.

Application Profiles

- e.g. collections of files that make up an application.

Internet and social network information

- Friends lists

Creating, testing, and adopting schema and formats is hard work.

Digital Forensics XML simplifies forensic processing.

DFXML can be used to annotate many forensic modalities

- Disk images
- Files within disk images
- Network Packets
- Hash sets

DFXML provides provenance:

- Tracks where data came from in a consistent, machine-readable manner.
- Tracks which tools were used and how they were made.

Integration:

- Tools can both generate and consume DFXML.

Dozens of people have been using DFXML since 2006

- Students at NPS — build projects without understanding EnScript, FTK or TSK
- Support has been added to numerous tools.

XML is an effective tool for annotating forensic outputs

XML has flexibility and good adoption:

- More programmers speak XML than “forensics.”
- Our tags are reasonably self-documenting
- Creating an XML for forensics was very straightforward.
- As our tools improve, we can add new XML tags.
  - *Namespaces can prevent tag conflicts.*
  - *Old tools will ignore tags they don’t understand.*

DFXML is an informal XML (at least for now)

- XML documents must be syntactically correct.
- “Validation is in the eye of the beholder.”
- Experience tells us which tags and constructs are useful.

Goals for DFXML

Complement existing forensic formats.

Be easy to generate

Be easy to ingest

Provide for human readability

Be free, open and extensible

Provide for scalability

Adhere to existing practices and standards

DFXML is in production.

We have XML tags that describe:

- Files, file metadata, application metadata
- Hash codes.
- Partitioning schemes.

We have toolkits:

- Python (2.7 & 3.2) modules for reading DFXML.
- C++ code for generating DFXML (including provenance)

We have support:

- fiwalk, bulk_extractor
- md5deep, sha1deep, hashdeep, etc.
- PhotoRec, StegCarver

<creator> provides provenance

```
<creator version='1.0'>
  <program>BULK_EXTRACTOR</program>
  <version>1.1.0_beta8</version>
  <build_environment>
    <compiler>GCC 4.2</compiler>
    <compilation_date>2011-11-19T23:27:21</compilation_date>
    <library name="afflib" version="3.6.9"/>
    <library name="libewf" version="20100805"/>
    <library name="exiv2" version="0.21.1"/>
  </build_environment>
  <execution_environment>
    <cpuid>
      <identification>GenuineIntel</identification>
      <family>6</family>
      <model>5</model>
      <stepping>5</stepping>
      <clflush_size>64</clflush_size>
      <nproc>16</nproc>
      <L1_cache_size>262144</L1_cache_size>
    </cpuid>
    <command_line>src/bulk_extractor -o dell1 /corp/drives/nist/nist-2004-
hacking/4Dell Latitude CPi.E01</command_line>
    <uid>501</uid>
    <username>simsong</username>
    <start_time>2011-11-20T04:34:27Z</start_time>
  </execution_environment>
```

<fileobject> is important for disk forensics

The DFXML <fileobject> tag describes information about a file.

- File name, size, and hash codes.
- Physical Location on the disk.
- Provenance

```
<fileobject>
  <filename>casper/filesystem.manifest-desktop</filename>
  <filesize>32672</filesize>
  <inode>651</inode>
  <meta_type>1</meta_type>
  <mode>511</mode>
  <nlink>1</nlink>
  <uid>0</uid>
  <gid>0</gid>
  <mtime>2008-12-29T01:33:32Z</mtime>
  <atime>2008-12-28T05:00:00Z</atime>
  <crtime>2008-12-29T01:33:32Z</crtime>
  <byte_runs>
   <byte_run file_offset='0' fs_offset='5577728' img_offset='5609984' len='32672'/>
  </byte_runs>
  <hashdigest type='md5'>bd1b0831fcba1f22eff2238da96055b6</hashdigest>
  <hashdigest type='sha1'>7e072af67f8d989cc8597487b948048ac3c7234</hashdigest>
</fileobject>
```

Multiple <fileobject>s can be used for a list of hashes

A hash list might be include metadata about the hashes, but lack timestamp and physical placement info:

```
<?xml version='1.0' encoding='UTF-8'?>
<dfxml xmloutputversion='0.3'>
<metadata xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns='http://afflib.org/fiwalk/'
  xmlns:dc='http://purl.org/dc/elements/1.1/'>

<classification>UNCLASSIFIED</classification>
<dc:type>Hash Set</dc:type>
<dfxml xmloutputversion='0.3'>

<fileobject>
<filename>demo1.bin</filename>
<filesize>1718</filesize>
<hashdigest type='MD5'>8e008247fde7bed340123f617db6a909</hashdigest>
</fileobject>

<fileobject>
<filename>demo2.bin</filename>
<hashdigest type='MD5'>c44293fdb35b6639bdffa9f41cf84626</hashdigest>
</fileobject>

</dfxml>
```

Additional annotations can be added as needed

We can start with a simple representation:

```
<fileobject>
  <filename>casper/filesystem.manifest-desktop</filename>
  <hashdigest type='md5'>bd1b0831fcba1f22eff2238da96055b6</hashdigest>
</fileobject>
```

...and add more later (e.g. another hash function):

```
<fileobject>
  <filename>casper/filesystem.manifest-desktop</filename>
  <filesize>32672</filesize>
  <hashdigest type='md5'>bd1b0831fcba1f22eff2238da96055b6</hashdigest>
  <hashdigest type='sha1'>7e072af67f8d989cc8597487b948048ac3c7234</hashdigest>
</fileobject>
```

<byte_runs> describes where the data is located.

```
<byte_runs>
  <byte_run offset="0"     img_offset="114688"  len="32768"/>
  <byte_run offset="32768" img_offset="1523712" len="32768"/>
  <byte_run offset="65536" img_offset="6356992" len="39659"/>
</byte_runs>
```

These can be annotated with hashes, too:

```
<byte_runs>
  <byte_run offset="0"     img_offset="114688"  len="32768">
     <hashdigest type="md5">e07910a06a086c83ba41827aa00b26ed</hashdigest>
  </byte_run>
  <byte_run offset="32768" img_offset="1523712" len="32768"/>
     <hashdigest type="md5">9a8ad92c50cae39aa2c5604fd0ab6d8c</hashdigest>
  </byte_run>
  <byte_run offset="65536" img_offset="6356992" len="39659"/>
     <hashdigest type="md5">2b00042f7481c7b056c4b410d28f33cf</hashdigest>
  </byte_run>
</byte_runs>
```

<byte_run> can have both `img_offset` and `fs_offset`:

- Redundant, but it makes processing easier!

tcpflow outputs <fileobject> for TCP/IP connections.

```
<fileobject>
  <filename>n8/074.125.019.164.00080-192.168.001.064.38853</filename>
  <filesize>6789</filesize>
  <tcpflow startime='2008-07-21T18:51:56.839358Z'
           endtime='2008-07-21T18:51:56.839358Z'
           src_ipn='74.125.19.164' dst_ipn='192.168.1.64'
           packets='14' srcport='80' dstport='38853' family='2' />
  <hashdigest type='MD5'>2df69a23d503ac7df18dee8ea6876eb8</hashdigest>
</fileobject>
```

- Now I can pull out of this <fileobject> the HTTP header:

```
<fileobject>
  <filename>n8/074.125.019.164.00080-192.168.001.064.38853</filename>
  <filesize>6789</filesize>
  <tcpflow startime='2008-07-21T18:51:56.839358Z'
           endtime='2008-07-21T18:51:56.839358Z'
           src_ipn='74.125.19.164' dst_ipn='192.168.1.64'
           packets='14' srcport='80' dstport='38853' family='2' />
  <hashdigest type='MD5'>2df69a23d503ac7df18dee8ea6876eb8</hashdigest>
  <byte_run file_offset='0' len='307'>
    <filename>n8/074.125.019.164.00080-192.168.001.064.38853-HTTP</filename>
    <hashdigest type='MD5'>74ae0b06d7a3adf9dbb9eb208905aae8</hashdigest>
  </byte_run>
</fileobject>
```

<msregistry> describes Windows registry values

```
<msregistry>
  <key name="HKEY_CURRENT_USER" root="1">
    <key name="Console">
      <mtime>2010-03-24T10:10:10-04:00Z</mtime>
      <value name="ColorTable00" type="REG_DWORD" value="0"/>
      <value name="ColorTable01" type="REG_DWORD" value="8388608"/>
...
      <value name="WindowSize" type="REG_DWORD" value="1638480"/>
      <value name="WordDelimiters" type="REG_DWORD" value="0"/>
    </key>
  </key>
</msregistry>
```

Key things that we can represent:

- Registry key Modification times.
- Physical location of registry entries within the hive or in unallocated space.
- Disconnected registry entries.

DFXML uses ISO8601 timestamps (e.g. 2010-03-24T10:10:10-04:00Z)

ISO 8601 has significant advantages over Epoch-based timestamps

- Represent times with or without time zones
- Can represent leap seconds
  - *1997-06-30T23:59:60Z*
  - *1998-12-31T23:59:60Z*
- Can represent clocks that are set incorrectly.
- Can represent systems with improper timezone handling.

Disadvantages:

- Timestamps are 20 bytes instead of 4 (or 8) bytes.
- Processing timestamps takes longer (but the added time is not significant).

Today there are several ways to use DFXML

Provenance <creator>

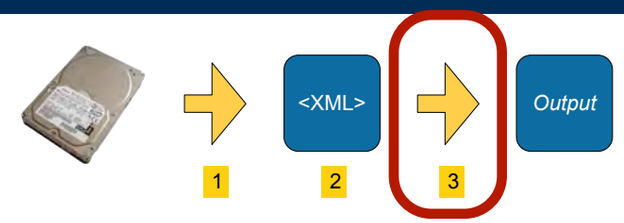
- **dfxml.h / dfxml.cpp**: DFXML C++ class (Mac, Linux & Windows)

File Objects <fileobject>

- **fiwalk** — uses SleuthKit to extract <fileobjects> from disk images
- **md5deep** (v4.0) — Hashes files and generates DFXML files
- **PhotoRec** — DFXML file reports where files were found
- **frag_find** — hash-based carving tool
- **tcpflow** — TCP/IP session reconstructor; outputs results in DFXML
- **bulk_extractor** — DFXML for provenance & plugins

DFXML can be consumed with programs written in Python

fiwalk.py and dfxml.py: Python modules for automated forensics.



Key Features:

- Can automatically run fiwalk with correct options if given a disk image
- Reads XML file if present (faster than regenerating)
- Creates and consumes **fileobject** objects.

High-performance processing with SAX callback interface:

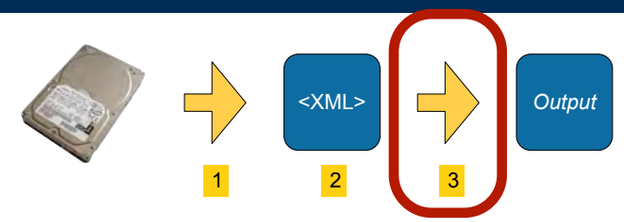
- Very fast and minimal memory footprint

Example: a program to print the files and sizes in a disk image:

```
import fiwalk,dfxml
def process(fi):
    if fi.filesize()<100:
        print fi.filename(),fi.filesize()
f = open("/corp/drives/nps/nps-2008-jean/nps-2008-jean.E01")
fiwalk.fiwalk_using_sax(imagefile=f,callback=process)
```

The program runs quite fast:

Sample program:




```
import fiwalk,dfxml
def process(fi):
    if fi.filesize()<100:
        print fi.filename(),fi.filesize()
f = open("/corp/drives/nps/nps-2008-jean/nps-2008-jean.E01")
fiwalk.fiwalk_using_sax(imagefile=f,callback=process)
```

Produces:

```
$ python x.py
$BadClus 0
$Extend/.. 56
$Extend/$ObjId 0
$Extend/$Quota 0
$Extend/$Reparse 0
$Secure 0
$Volume 0
...
Documents and Settings/Administrator/Cookies/administrator@ads.cnn[2].txt 96
Documents and Settings/Administrator/Cookies/administrator@c.msn[2].txt 68
...
Documents and Settings/Administrator/Cookies/administrator@www.msn[1].txt 85
...
```

“fi” is a Python fileobject class.

The Python **`dfxml.fileobject`** class is an easy-to-use abstract class for working with file system data.

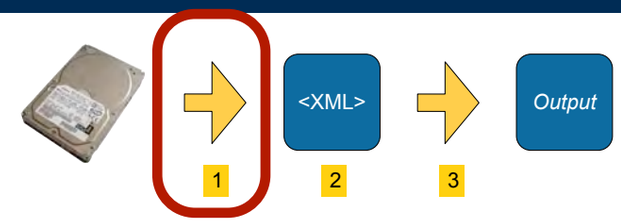
The class supports a simple interface:

```
fi.partition()
fi.filename(), fi.ext()
fi.filesize()
fi.uid(), fi.gid(), fi.metatype(), fi.mode()
fi.ctime(), fi.atime(), fi.crtime(), fi.mtime(), fi.dtime(), fi.times()
fi.sha1(), fi.md5()
fi.byteruns(), fi.fragments()
fi.content()
fi.tempfile()
```

Currently, you must look at the dfxml.py file for the documentation.

fiwalk extracts metadata from disk images.

fiwalk is a C++ program built on top of SleuthKit




```
$ fiwalk [options] -X file.xml imagefile
```

Features:

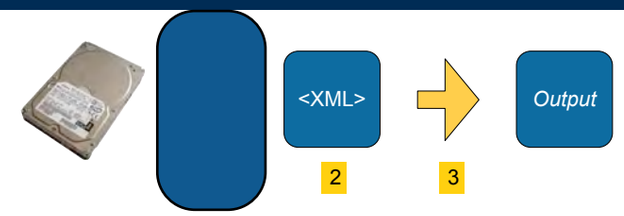
- Finds all partitions & automatically processes each.
- Handles file systems on raw device (partition-less).
- Creates a single output file with forensic data data from all.

Single program has multiple output formats:

- XML (for automated processing)
- ARFF (for data mining with Weka)
- "walk" format (easy debugging)
- SleuthKit Body File (for legacy timeline tools)

fiwalk has a plugable metadata extraction system.

Configuration file specifies Metadata extractors:



- Currently the extractor is chosen by the file extension.

```
*.jpg    dgi      ../plugins/jpeg_extract
*.pdf    dgi      java -classpath plugins.jar Libextract_plugin
*.doc    dgi      java -classpath ../plugins/plugins.jar word_extract
```

- Plugins are run in a different process for safety.

Metadata extractors produce name:value pairs on STDOUT

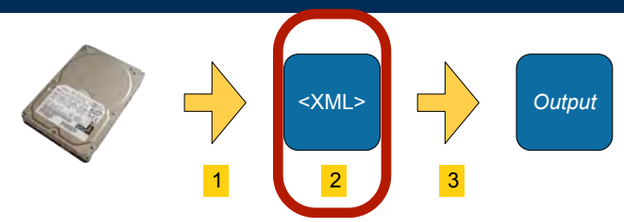
```
Manufacturer: SONY
Model: CYBERSHOT
Orientation: top - left
```

Extracted metadata is automatically incorporated into output:

```
<fileobject>
...
<Manufacturer>SONY</Manufacturer>
<Model>CYBERSHOT</Model>
<Orientation>top - left</Orientation>
...
</fileobject>
```

Resulting XML files can be distributed with images.

The XML file provides a key to the disk image:



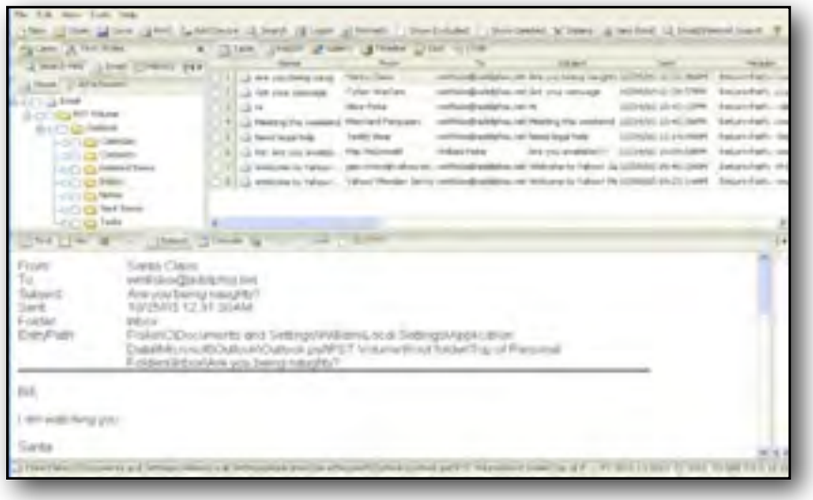

```
$ ls -l /corp/images/nps/nps-2009-domexusers/
-rw-r--r--  1 simsong  admin  4238912226 Jan 20 13:16 nps-2009-realistic.aff
-rw-r--r--  1 simsong  admin    38251423 May 10 23:58 nps-2009-realistic.xml
$
```

XML files:

- Range from 10K — 100MB.
  - *Depending on the complexity of the disk image.*
- Only have files & orphans that are identified by SleuthKit
  - *You can easily implement a "smart carver" that only carves unallocated sectors.*

Creating Forensic Corpora

Digital forensics is at a turning point.
Yesterday's work was primarily *reverse engineering*.



Key technical challenges:

- Evidence preservation.
- File recovery (file system support); Undeleting files
- Encryption cracking.
- Keyword search.

Today's work is increasingly *scientific.*

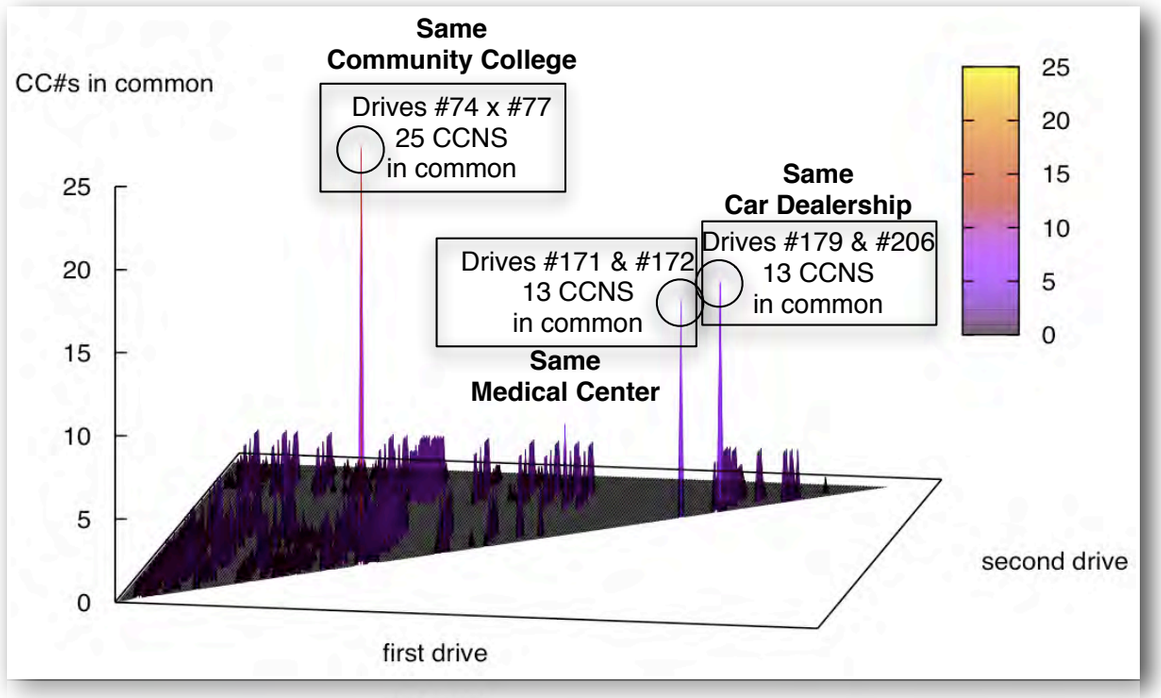
Evidence Reconstruction

- Files (fragment recovery carving)
- Timelines (visualization)

Clustering and data mining

Social network analysis

Sense-making



Science requires the *scientific process.*

Hallmarks of Science:

- Controlled and repeatable experiments.
- No privileged observers.

Why repeat some other scientist's experiment?

- Validate that an algorithm is properly implemented.
- Determine if ***your*** new algorithm is better than ***someone else's*** old one.
- (Scientific confirmation? — perhaps for venture capital firms.)

We can't do this today.

- People work with their own data
  - —*Can't sure because of copyright & privacy issues.*
- People work with "evidence"
  - —*Can't discuss due to legal sensitivities.*

We do science with “real data.”

The Real Data Corpus (30TB)

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

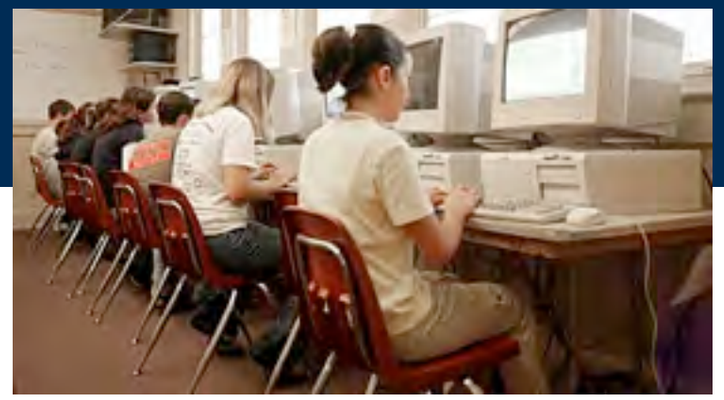
Mobile Phone Application Corpus

- Android Applications; Mobile Malware; etc.

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

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

Digital Forensics education needs fake data!



To teach forensics, we need complex data!

- Disk images
- Memory images
- Network packets

Some teachers get used hard drives from eBay.

- Problem: you don't know what's on the disk.
  - *Ground Truth.*
  - *Potential for illegal Material — distributing porn to minors is illegal.*

Some teachers have students examine other student machines:

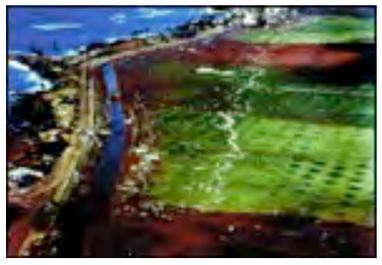
- Self-examination: students know what they will find
- Examining each other's machines: potential for inappropriate disclosure

We manufacture data that can be freely redistributed.

Files from US Government Web Servers (500GB)

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

Advantage over flickr & youtube: persistence & copyright

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

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

Our fake data can be freely redistributed.

Test and Realistic Disk Images (1TB)

- Mostly Windows operating system.
- Some with complex scenarios to facilitate forensics education.

—*NSF DUE-0919593*

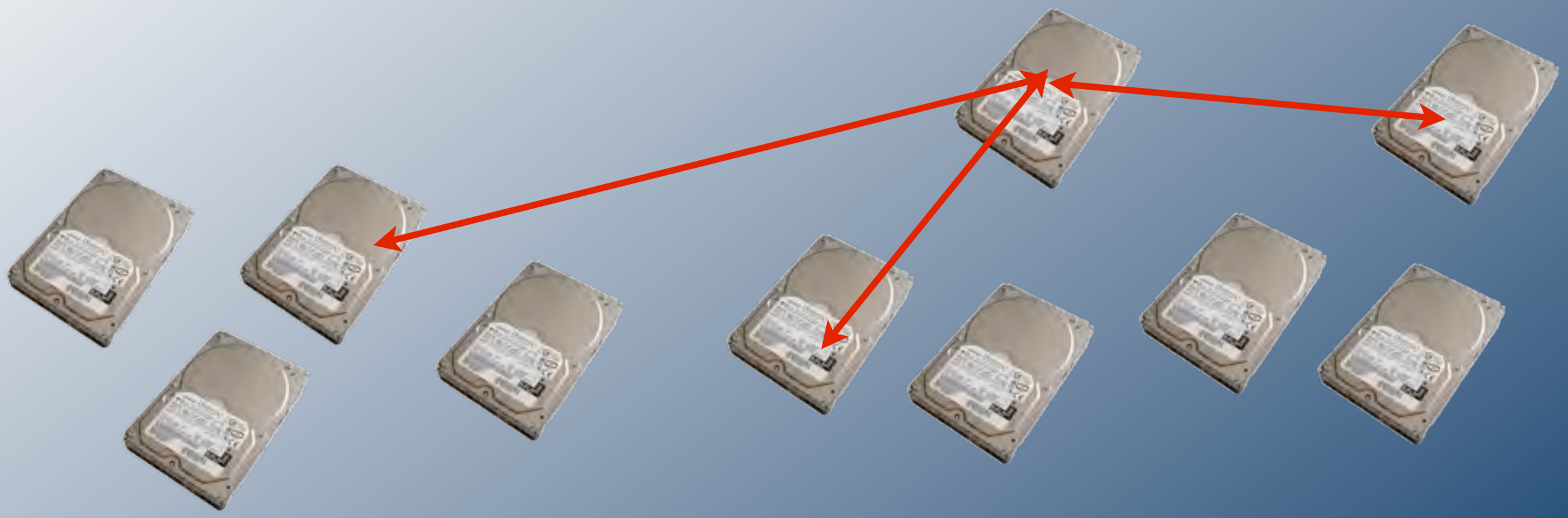
University harassment scenario

- Network forensics — browser fingerprinting, reverse NAT, target identification.
- 50MB of packets

Company data theft & child pornography scenario.

- Multi-drive correction.
- Hypothesis formation.
- Timeline reconstruction.

—*Disk images, Memory Dumps, Network Packets*



Where do we go from here?

There are many important areas for research

Algorithm development.

- Adopting to **different kinds of data**.
- **Different resolutions**
- **Higher Amounts (40TB—40PB)**

Software that can…

- Automatically identify outliers and inconsistencies.
- Automatically present complex results in simple, straightforward reports.
- Combine stored data, network data, and Internet-based information.

Many of the techniques here are also applicable to:

- Social Network Analysis.
- Personal Information Management.
- Data mining unstructured information.

Our challenges: innovation, scale & community

Most innovative forensic tools **fail when they are deployed**.

- Production data *much larger* than test data.
  - —*One drive might have 10,000 email addresses, another might have 2,000,000.*
- Production data *more heterogeneous* than test data.
- Analysts have less experience & time than tool developers.

How to address?

- Attention to usability & recovery.
- High Performance Computing for testing.
- Programming languages that are *safe* and *high-performance.*

Moving research results from lab to field is itself a research problem.

<u>In summary</u>, there is an urgent need for fundamental research in automated computer forensics.

Most work to date has been data recovery and reverse engineering.

- User-level file systems
- Recovery of deleted files.

To solve tomorrow's hard problems, we need:

- Algorithms that exploit large data sets (>10TB)
- Machine learning to find *outliers* and *inconsistencies.*
- Algorithms tolerant of data that is *dirty* and *damaged.*

Work in automated forensics is *inherently interdisciplinary.*

- Systems, Security, and Network Engineering
- Machine Learning
- Natural Language Processing
- Algorithms (compression, decompression, big data)
- High Performance Computing
- Human Computer Interactions