Automated Digital Forensics

Simson L. Garfinkel
Associate Professor, Naval Postgraduate School

June 2, 2011

http://simson.net/

NPS is the Navy's Research University.

Location: Monterey, CA

Campus Size: 627 acres

Students: 1500

- US Military (All 5 services)
- US Civilian (Scholarship for Service & SMART)
- Foreign Military (30 countries)

Schools:

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

Law enforcement & military agencies encounter substantial amounts of electronic media.

Typical media includes:

- Desktop & Laptop computers (hard drives)
- Cell phones (SIM chips, flash memory)
- iPods & MP3 music players
- GPS Devices

Typical sources includes:

- Domestic searches
- Border searches
- Media collected on the “battlefield”:
  - —*on combatants*
  - —*inside houses & apartments*
- Cyber security
  - —*victim systems*
  - —*attacker systems*
  - —*intermediaries*

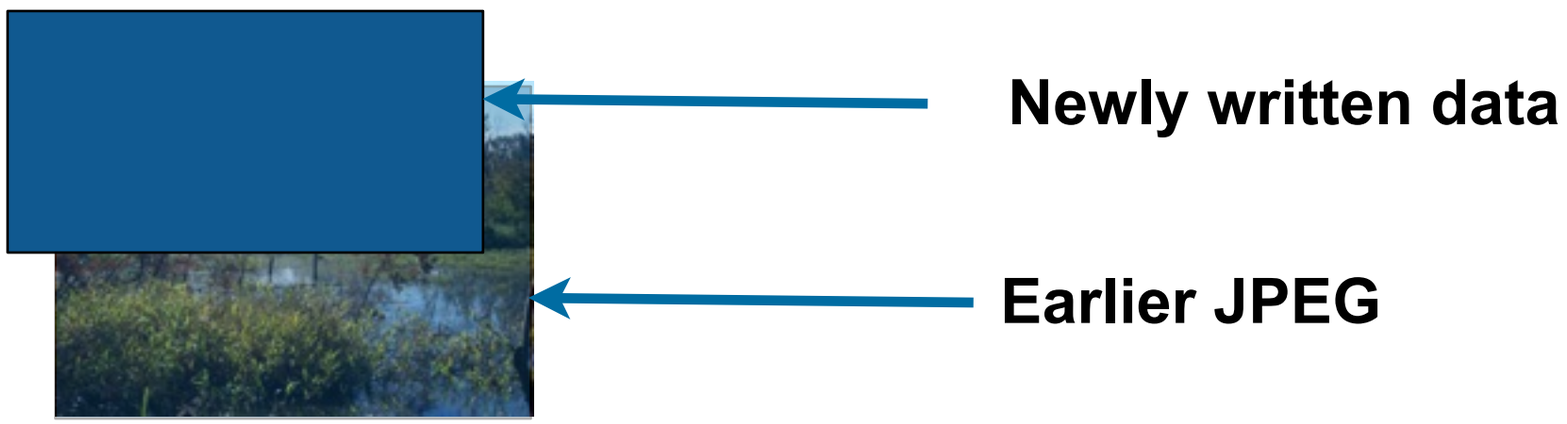
*Data **quality** makes digital forensics hard.*

Quality: any piece of data may be critical.

- Heterogeneity is a problem.
  - —*Address books*
  - —*Email*
  - —*Documents*
  - —*Photos*

Digital Forensics relies heavily on residual data.

- Slack space within files.
- Deleted Files
- Partially overwritten files
- Virtual memory fragments
- Hibernation files



- —*Residual data frequently reveals facts, motives, state-of-mind, or associations that the subject sought to hide from others.*
- —*CS lacks principled techniques for resolving incomplete data structures.*

*Data **quantity*** make digital forensics hard too!

Quantity: analysts have less time than the subject!

- User spent *years* assembling email, documents, etc.
- Analysts have days or hours to process it.

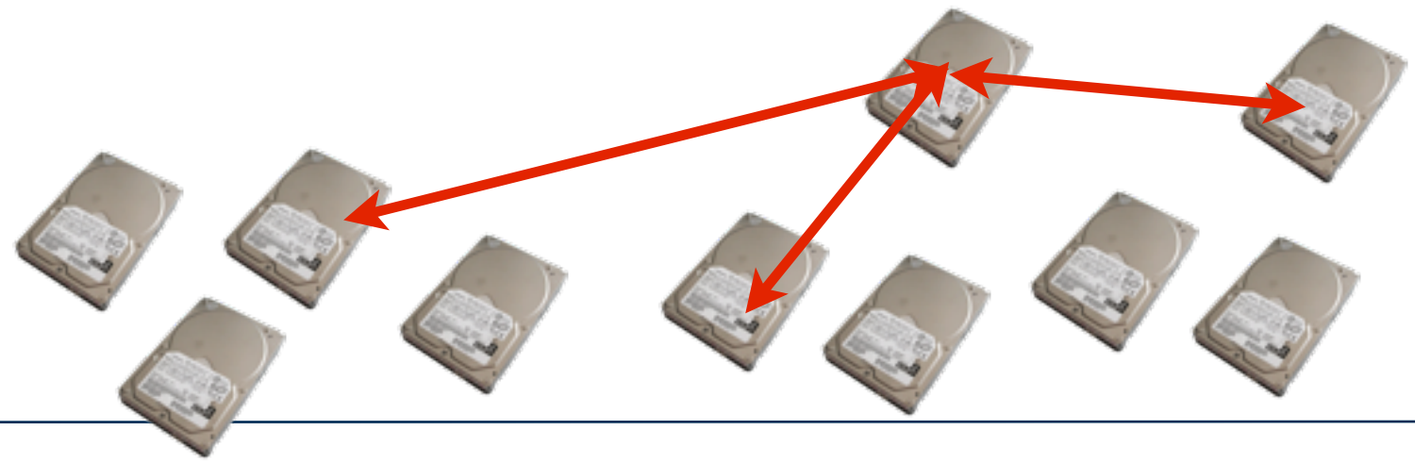
There is no resource advantage.

- Police analyze top-of-the-line systems … with top-of-the-line systems.
- National Labs have large-scale server farms … to analyze huge collections.

DF researchers must respond by developing new algorithms that:

—*Provide incisive analysis through cross-drive analysis.*

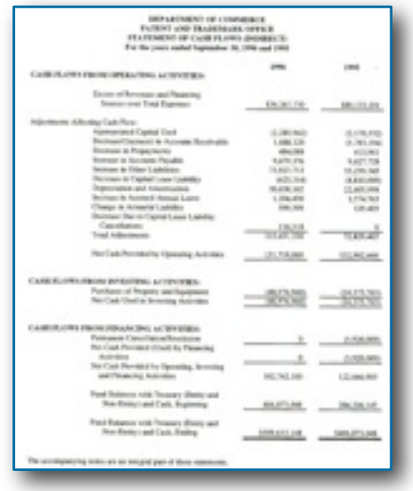
—*Operate autonomously on incomplete, heterogeneous datasets.*



Examinations can have multiple purposes!

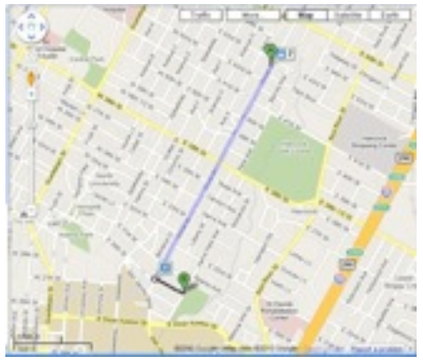
Examiner looks for ***evidence*** of a crime to support a ***conviction:***

- Financial records of fraud.
- Photographs of a murder.
- Threats sent by email.
- Evidence of a conspiracy.
- Child pornography.



Examiner looks for ***intelligence*** to support an ***investigation:***

- Associates & accomplices.
- Geographical locations.
- Tools, techniques, modus operandi, operating procedures



Examiner looks for ***artifacts*** of an ***intrusion:***

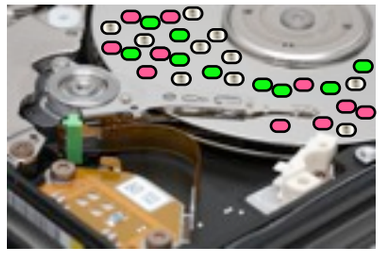
- New vulnerabilities and exploits.
- Evidence of how intrusion was done, who did it.

This talk introduces digital forensics and presents three research projects from my lab.

Introducing Digital Forensics

Histogram Analysis

Random sampling for high speed forensics



Creating Corpora for Digital Forensics

Introducing Digital Forensics

Data extraction is the first step of forensic analysis

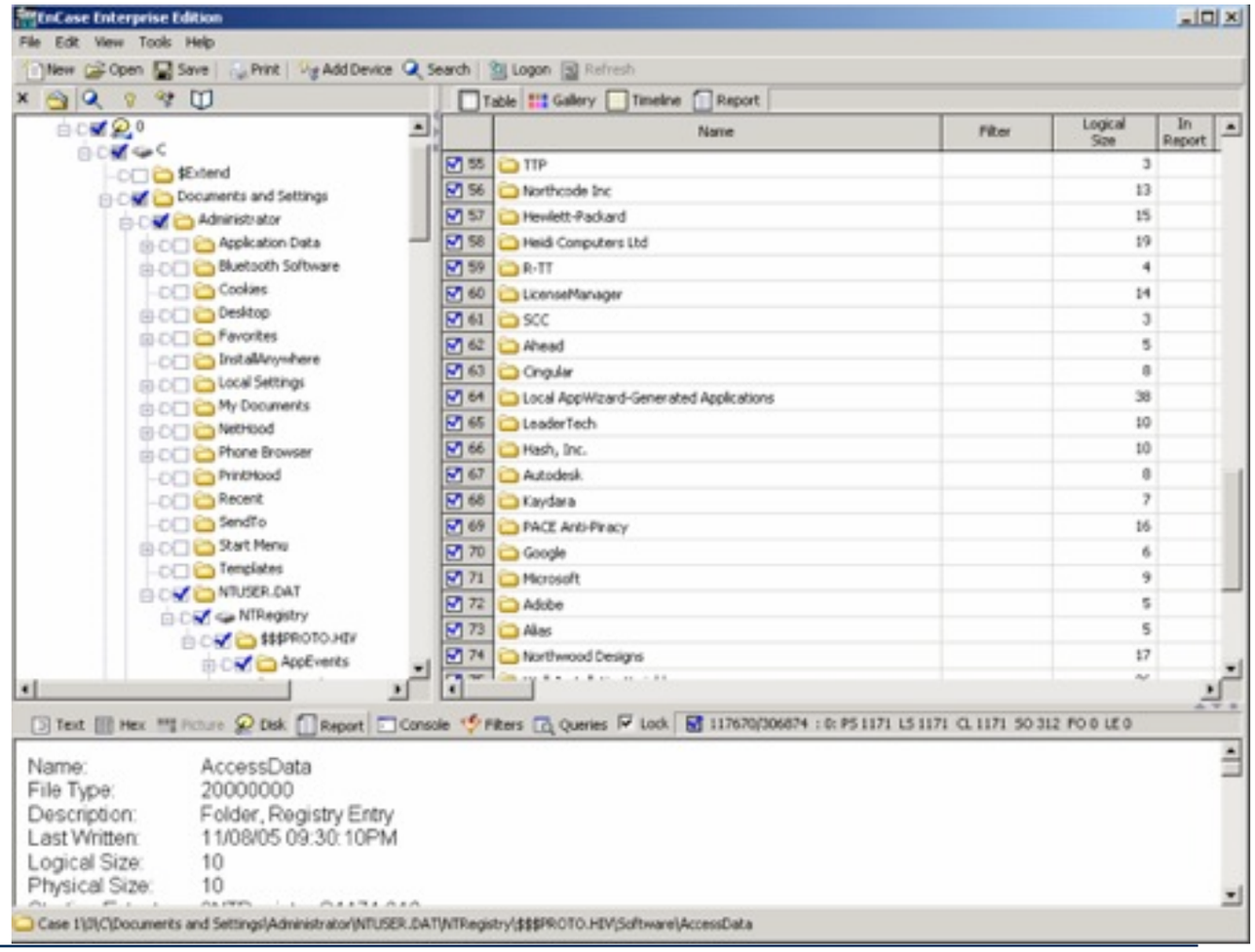
Imaging tools extract the data without modification.

- "Forensic copy" or "disk image."
- Original media is stored in an evidence locker.

After extraction, examiners use digital forensic tools to view the evidence.

Today's tools allow the examiner to:

- Display of *allocated* & *deleted* files.
- String search.
- Data recovery and *file carving*.
- Examining individual disk sectors in hex, ASCII and Unicode



The last decade was a "Golden Age" for digital forensics.

Widespread use of Microsoft Windows, especially Windows XP

Relatively few file formats:

- Microsoft Office (.doc, .xls & .ppt)
- JPEG for images
- AVI and WMV for video

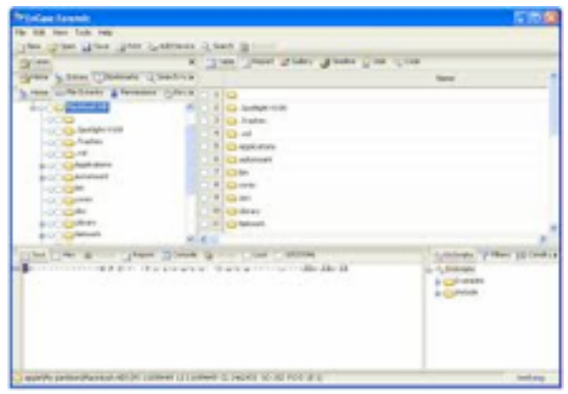
Most examinations confined to a single computer belonging to a single subject

Most storage devices used a standard interface.

- IDE/ATA
- USB

Uniformity gave us good tools and rapid growth.

Commercial tools:



EnCase®

Open Source Tools:

The Sleuth Kit

Content Extraction Toolkits:

Oracle Outside In Technology

Outside In Technology is a suite of software development kits (SDKs) that provides developers with a comprehensive solution to access, transform and control the contents of over 500 unstructured file formats. Each SDK within the suite is optimized to solve a particular problem but they are highly flexible and interoperable. Developers can quickly implement any combination of the Outside In SDKs to provide exactly the right functionality in their application while minimizing integration effort and code footprint. The SDKs offer a wide range of options to give the developer programmatic control of their workflow and output. Thorough documentation and sample applications with source code are included to further accelerate implementation.

Download | Documentation | Sample Code

Today there is a growing digital forensics crisis.

Much of the last decade's progress is quickly becoming irrelevant.

Tools designed to let an analyst find a file and take it into court...

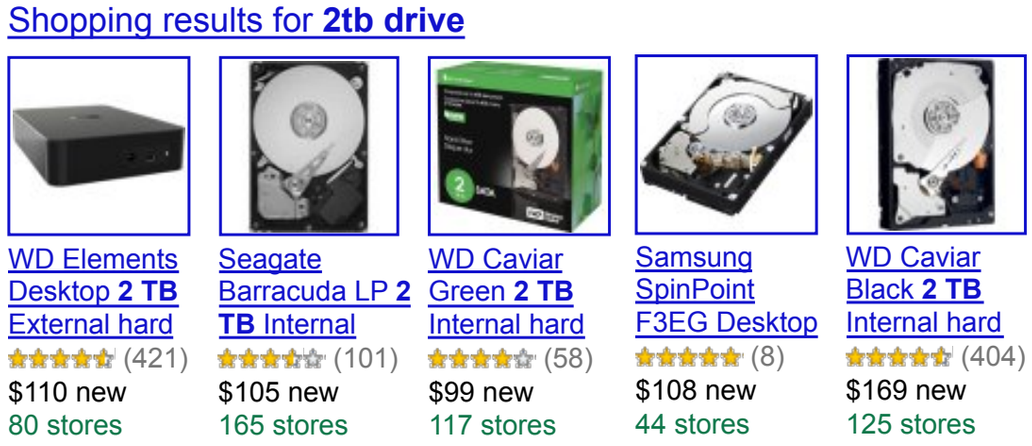
… don't scale to today's problems.

We have identified 5 key problems.

Problem 1 - Increased cost of extraction & analysis.

Data: too much and too complex!

- Increased size of storage systems.



- Cases now require analyzing multiple devices
  - *—2 desktops, 6 phones, 4 iPods, 2 digital cameras = 1 case*

- Non-Removable Flash

- Proliferation of operating systems, file formats and connectors
  - *—XFAT, XFS, ZFS, YAFFS2, Symbian, Pre, iOS,*

Consider FBI Regional Computer Forensic Laboratories growth:

- Service Requests: 5,057 (FY08) ➔ 5,616 (FY09) (+11%)
- Terabytes Processed: 1,756 (FY08) ➔ 2,334 (FY09) (+32%)

Problem 2 — RAM and malware forensics is really hard.

RAM Forensics—in its infancy

- RAM structures change frequently (no reason for them to stay constant.)
- RAM is constantly changing.

Malware is hard to analyze:

- Encryption; Conditional execution;
- Proper behiavor of most software is not specified.

Malware can hide in many places:

- On disk (in programs, data, or scratch space)
- BIOS & Firmware
- RAID controllers
- GPU
- Ethernet controller
- Motherboard, South Bridge, etc.
- FPGAs

Problem 3 — Mobile phones are really hard to examine.

Cell phones present special challenges.

- No standard connectors.
- No standard way to copy data out.
- Difficult to image & store cell phones without changing them.

How do we validate tools against thousands of phones?

- No standardized cables or extraction protocols.

NIST's *Guidelines on Cell Phone Forensics* recommends:

- "searching Internet sites for developer, hacker, and security exploit information."

How do we forensically analyze 100,000 apps?

Problem 4 — Encryption and Cloud Computing make it hard to get to the data

Pervasive Encryption — Encryption is increasingly present.

- TrueCrypt
- BitLocker
- File Vault
- DRM Technology

Cloud Computing — End-user systems won't have the data.

- Google Apps
- Microsoft Office 2010
- Apple Mobile Me

—But they may have residual data!

Problem 5 — Time is of the essence.

Most tools were designed to perform a complete analysis.

- Find all the files.
- Index all the terms.
- Report on all the data.
- Take as long as necessary!

Increasingly we are racing the clock:

- Police prioritize based on statute-of-limitations!
- Battlefield, Intelligence & Cyberspace operations require turnaround in days or hours.

My research focuses on three main areas:

Area #1: Data collection and manufacturing

- Large data sets of real data enable science. (+20TB)
- Small data sets of realistic data enable education, training and publishing. (<1TB)

Area #2: Bringing data mining and machine learning to forensics

- Breakthrough algorithms based on correlation and sampling
- Automated social network analysis (cross-drive analysis)
- Automated ascription of carved data

Area #3: Working above and below the files

- Most work to date is with files.
- Digital Forensics XML (DFXML)
  - —*Connecting tools.*
  - —*Representing applications, behaviors, users.*
- Forensics of bulk data

Emphasis on *building tools* and *working with practitioners*.

n=579 domexuser1@gmail.com
n=432 domexuser2@gmail.com
n=340 domexuser3@gmail.com
n=192 domexuser2@live.com
n=153 domexuser2@hotmail.com
n=146 domexuser1@hotmail.com
n=134 domexuser1@live.com
n=91 premium-server@thawte.com
n=70 talkback@mozilla.org
n=69 hewitt@netscape.com
n=54 DOMEXUSER2@GMAIL.COM
n=48 domexuser1%40gmail.com@imap.gmail.com
n=42 domex2@rad.li
n=39 lord@netscape.com
n=37 49091023.6070302@gmail.com

Stream-based forensics and histogram analysis with bulk_extractor

Data on hard drives can be divided into three categories:

Resident Data

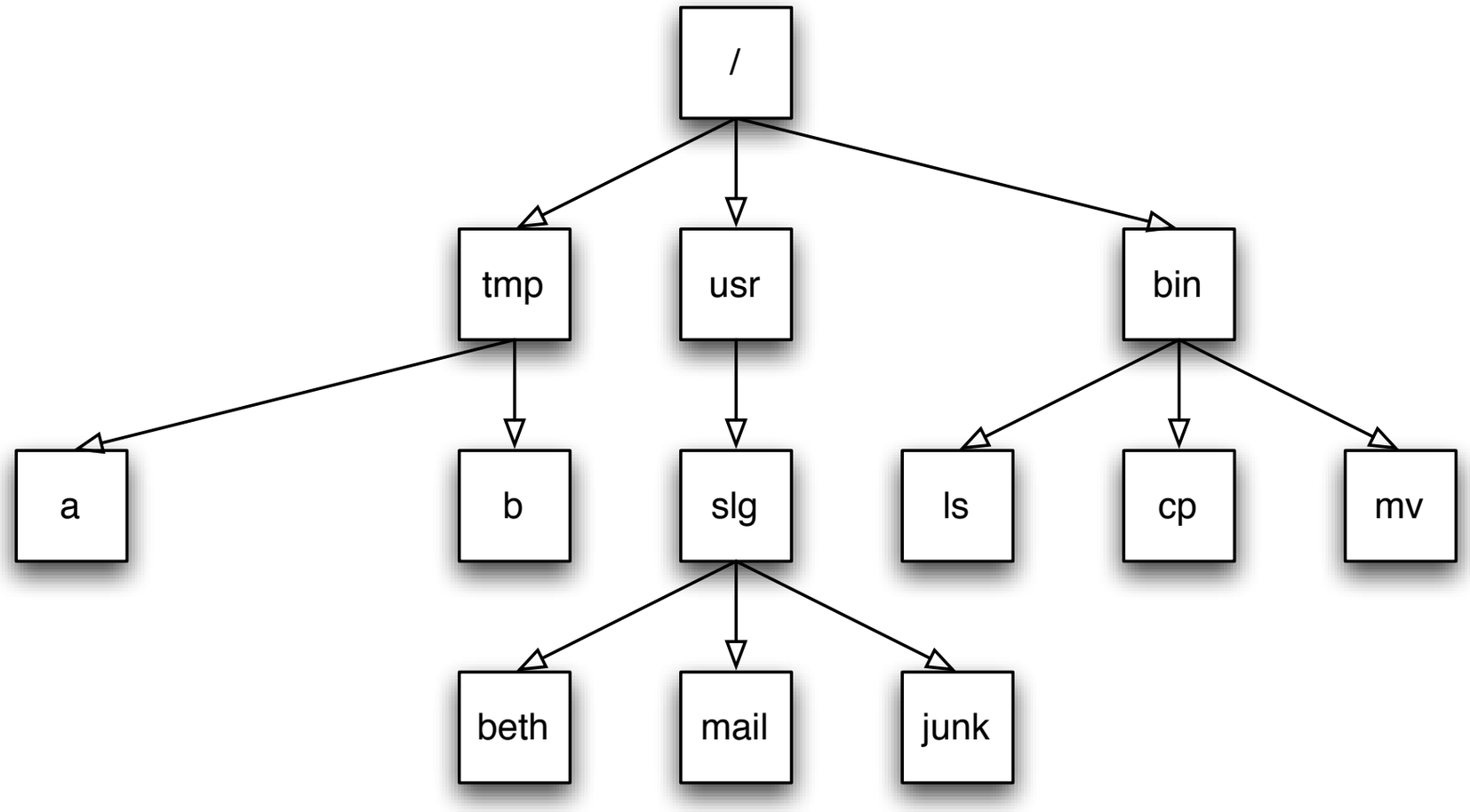
Deleted Data

No Data

user files
email messages
[temporary files]

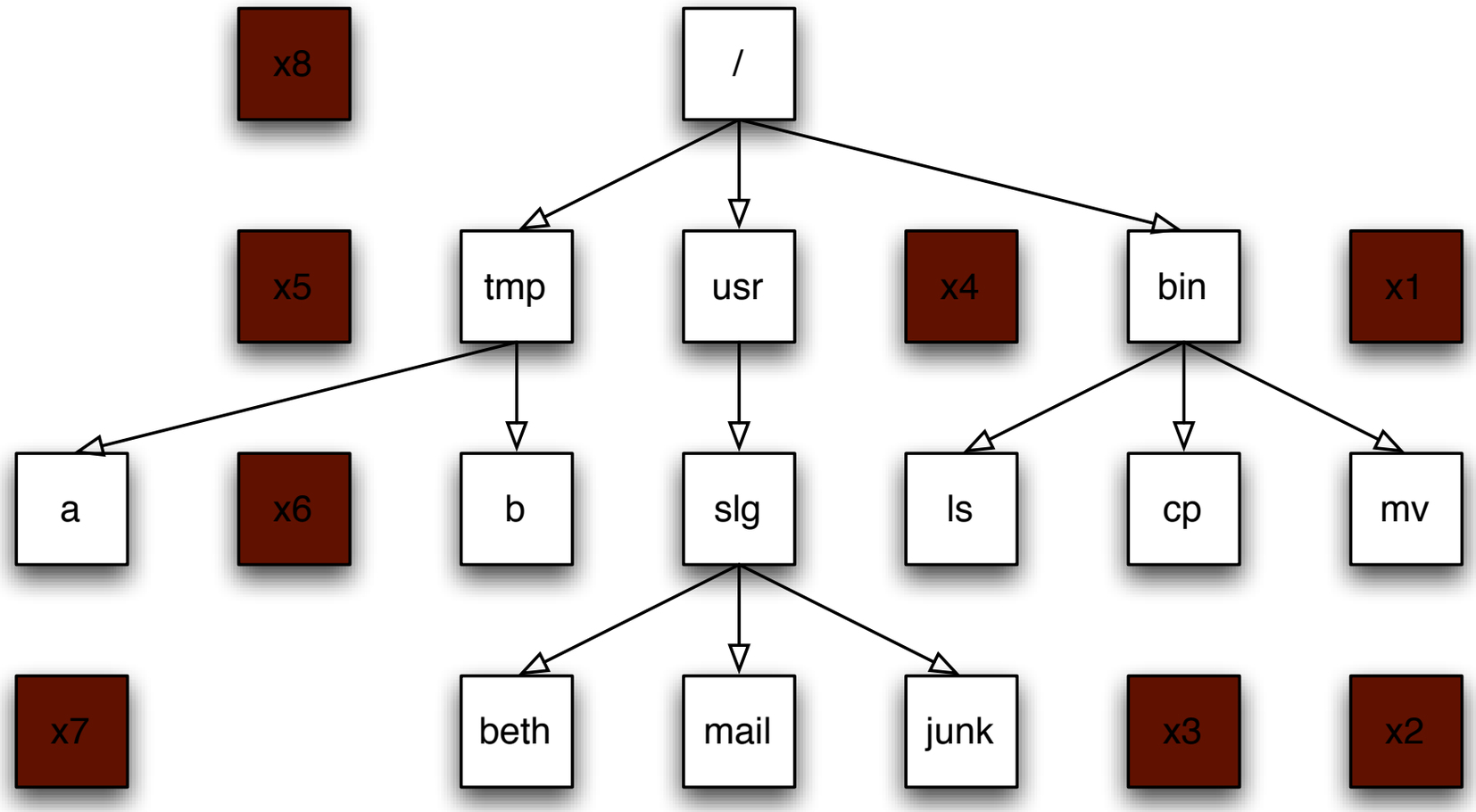
blank sectors

Resident data is the data you see from the root directory.



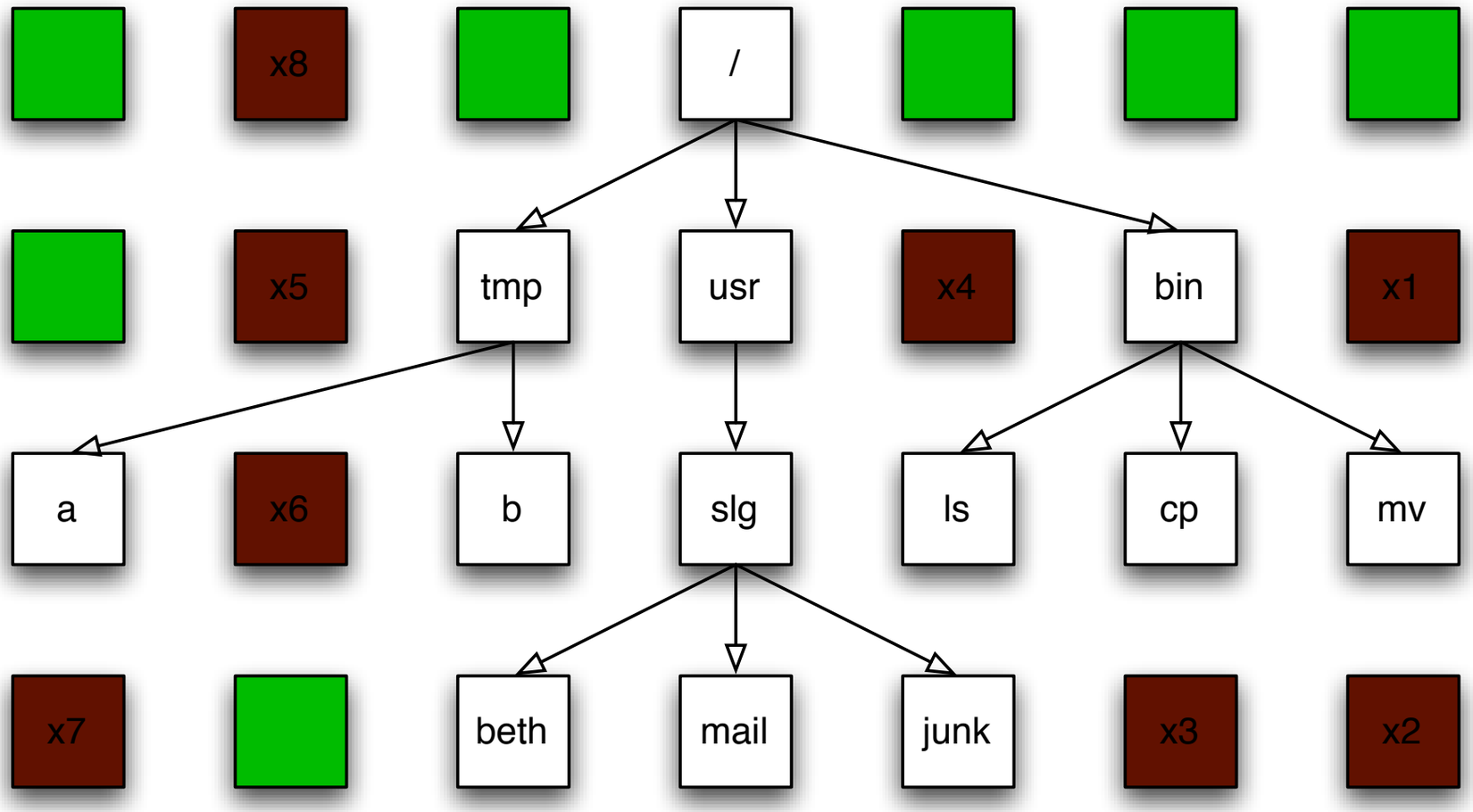
Resident Data

Deleted data is on the disk, but can only be recovered with forensic tools.



Deleted Data

Sectors with “No Data” are blank.



No Data

Today most forensic tools follow the same steps to analyze a disk drive.

Walk the file system to map out all the files (allocated & deleted).

For each file:

- Seek to the file.
- Read the file.
- Hash the file (MD5)
- Index file's text.

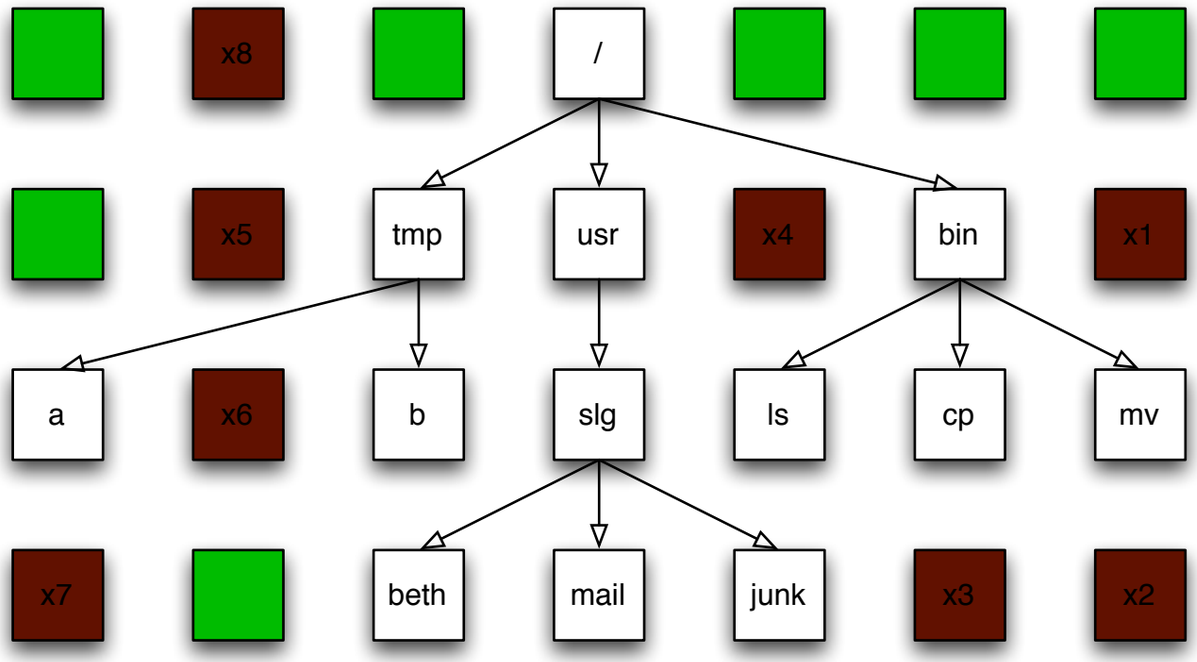
"Carve" space between files for other documents, text, etc.

Problem #1: Time

- 1TB drive takes 3.5 hours to read
  - —*10-80 hours to process!*

Problem #2: Completeness

- Lots of residual data is ignored.
  - —*Many investigations don't carve!*



Can we analyze a 1TB drive in 3.5 hours? (The time it takes to read the data.)

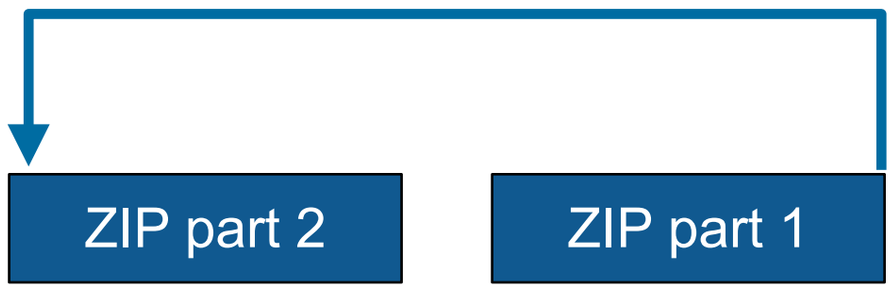
Stream-Based Disk Forensics:
Scan the disk from beginning to end; do your best.

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

Advantages:

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

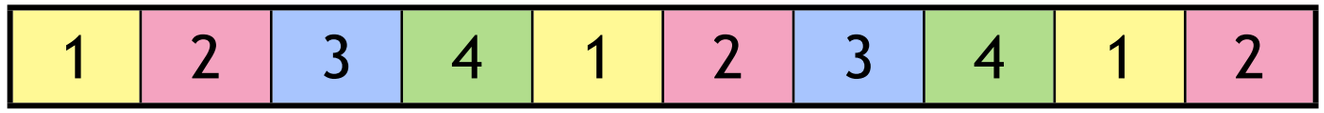
Disadvantages:



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

bulk_extractor: a high-speed disk scanner.

Written in C, C++ and Flex



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

Key Features:

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

Challenges:

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

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

email addresses from domexusers:

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

Histogram:

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

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

District Attorney filed charges against two individuals:

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

Defendants:

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

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

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

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

Sources of these addresses:

- Windows binaries
- SSL certificates
- Sample documents

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

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

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

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

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

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

Approach #3: Context-sensitive stop list.

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

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

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

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

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

Total stop list: 70MB (628,792 features)

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: Implemented as a set of C++ classes

Forensic Buffers and Path:

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

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

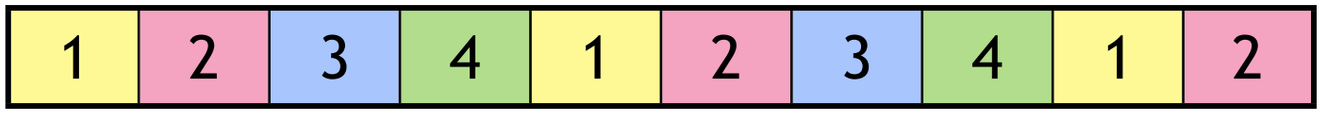
- feature_recorder — Holds output for each feature type

Plug-In Scanner System

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

bulk_extractor: Speed from multi threading

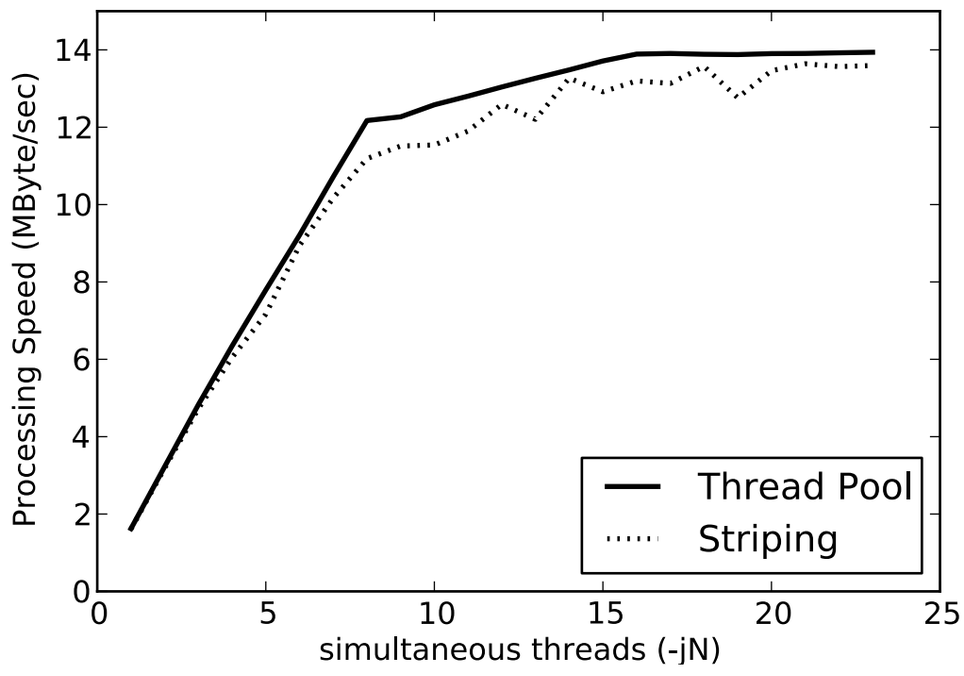
Primary thread:



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

Worker Threads:

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



bulk_extractor: Crash Protection

Every forensic tool crashes.

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

Crash Protection #1: “-c” option.

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

Crash Protection #2: restart.

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

Bulk_extractor's magic — opportunistic decompression

Most forensic tools recover:

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

bulk_extractor uses a different methodology:

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

This helps with:

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

Post-processing the feature files

The feature files are designed for easy, rapid processing.

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

bulk_diff.py: prepares difference of two bulk_extractor runs.

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

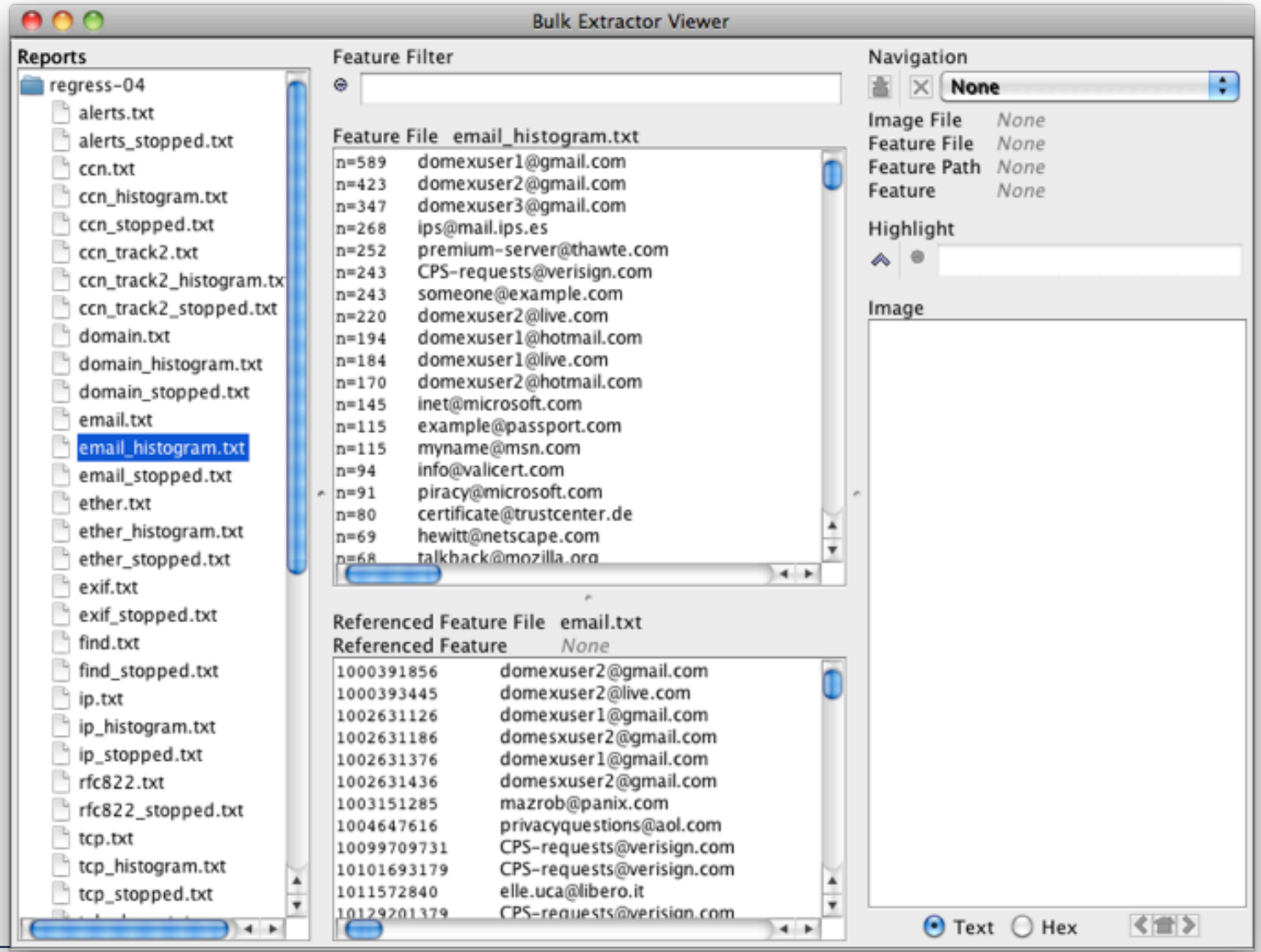
identify_filenames.py: Reports files responsible for features.

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

bulk_extractor GUI

100% Java

Uses bulk_extractor to view contents of compressed containers.



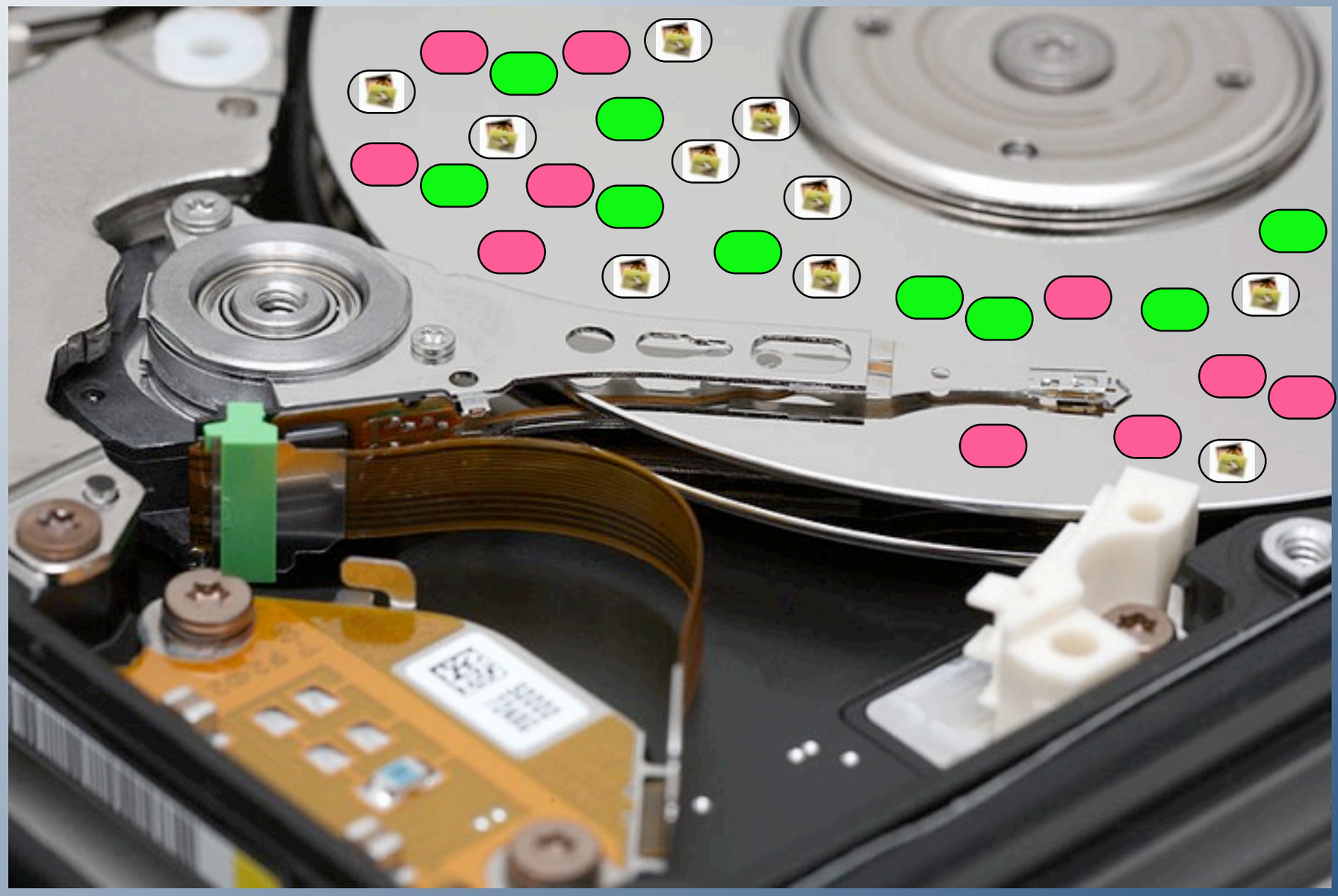
Future Work

Hash-based carving

Carving more file types

Lat/Long from GPS devices

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



Forensic 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

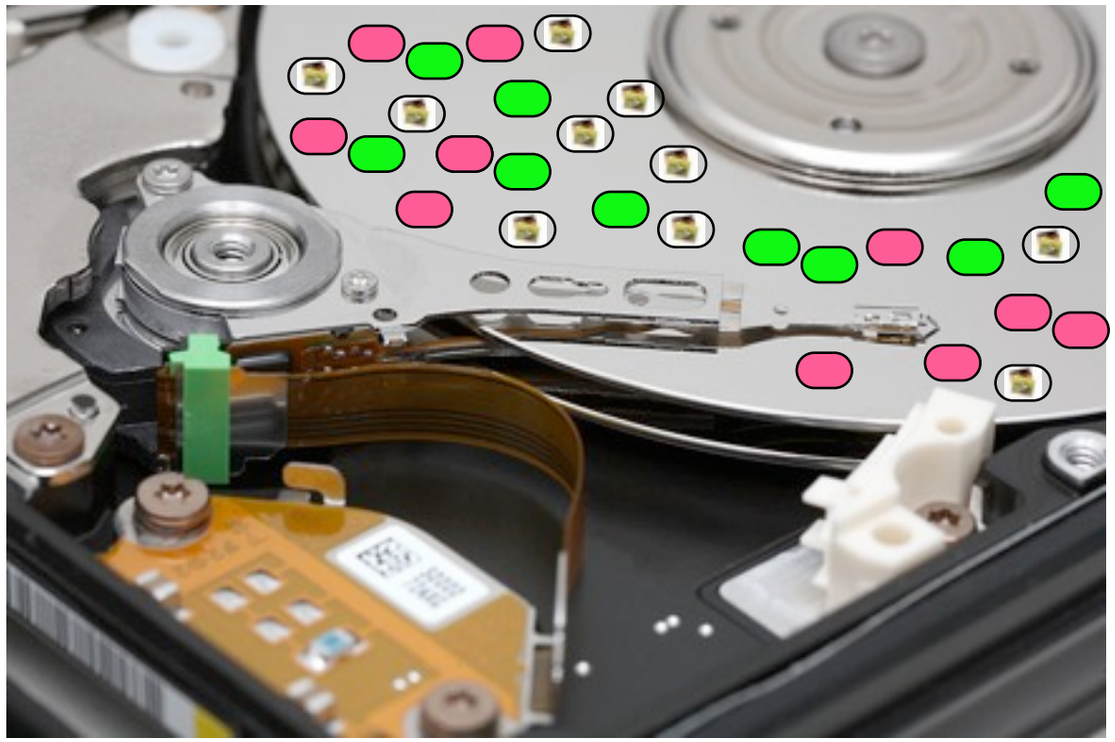
36 GB is a lot of data!

- ≈ 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:

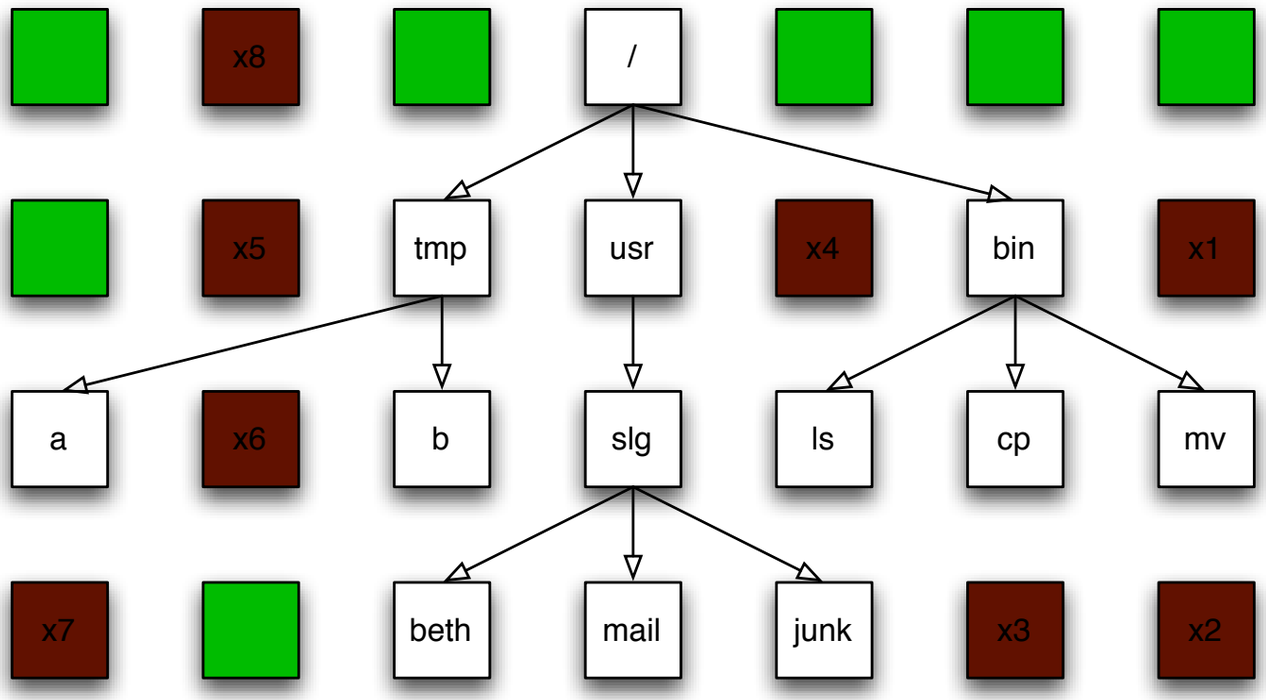
Hard drive contents can be predicted by sampling a few thousand sectors:

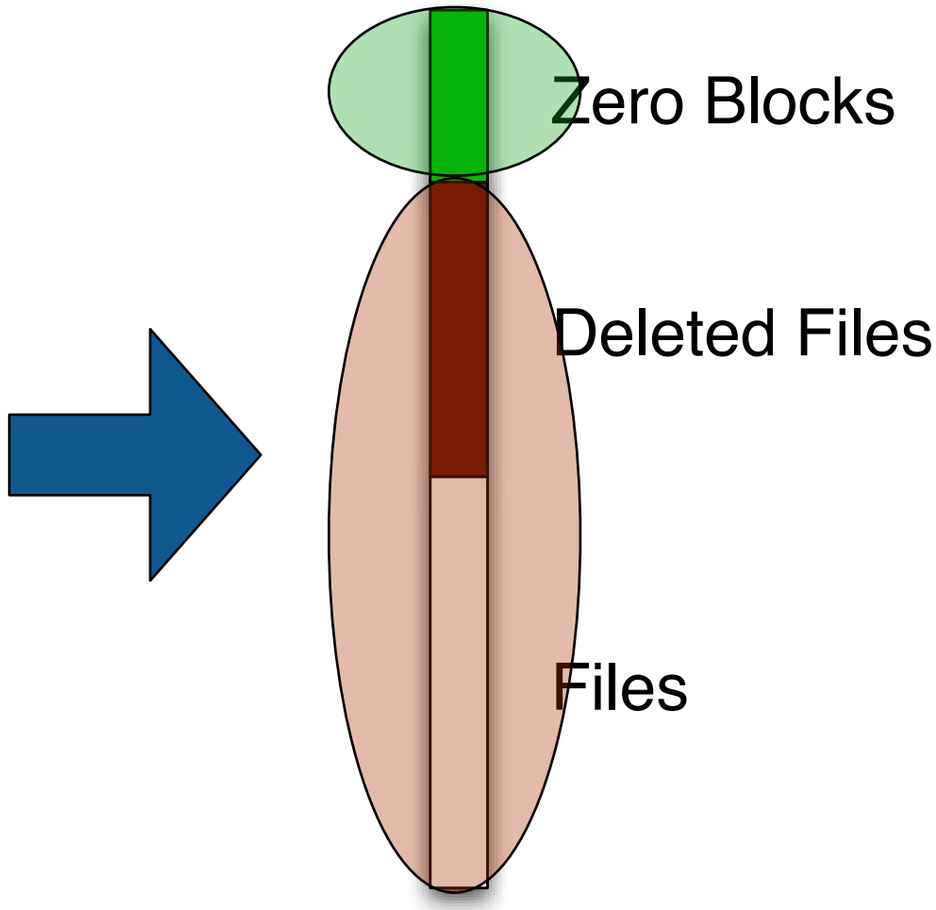


The challenge is identifying *likely voters.*

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

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





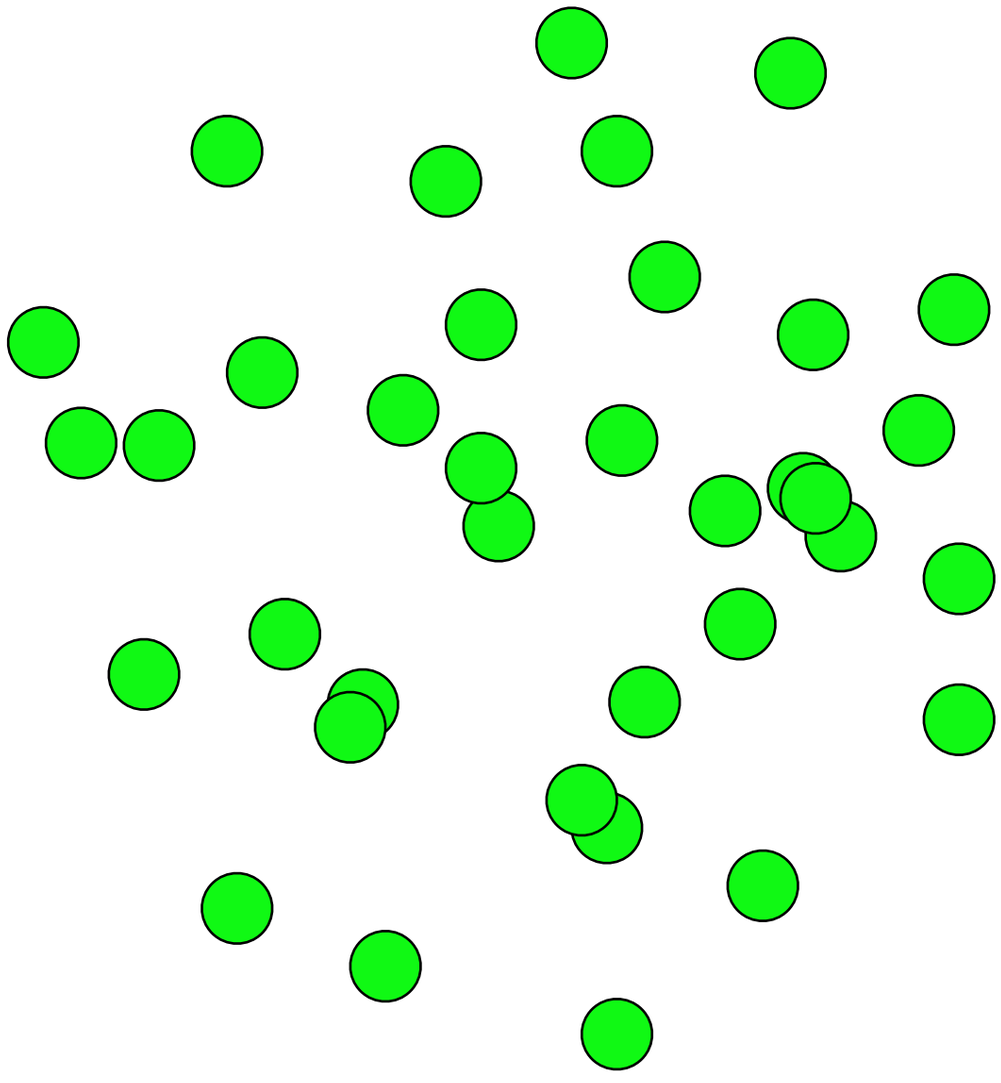
Simplify the problem.
Can we use statistical sampling to verify wiping?

Many organizations discard used computers.

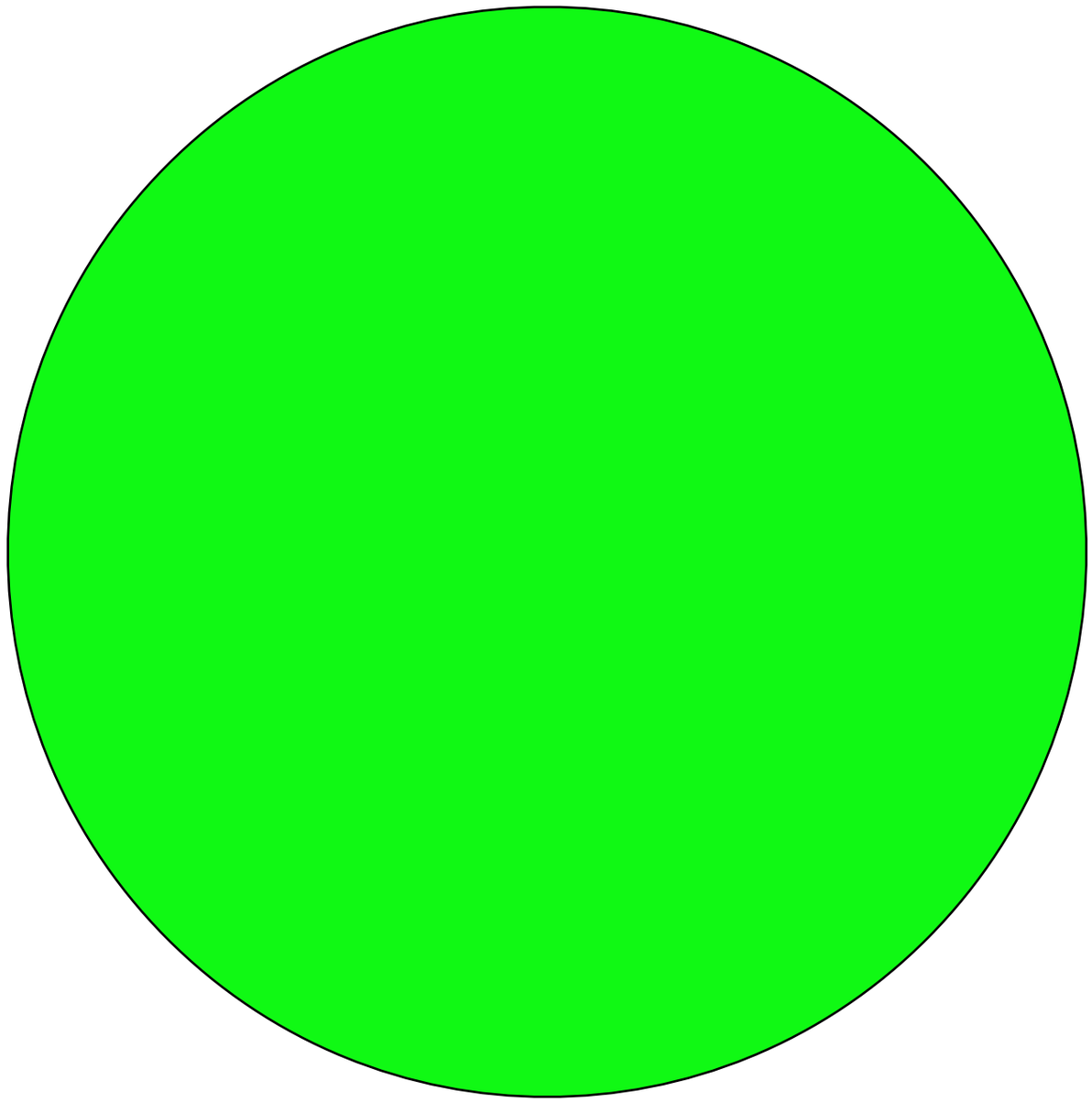
Can we verify if a disk is properly wiped in 5 minutes?

We read 10,000 randomly-chosen sectors … and they are all blank

We read 10,000 randomly-chosen sectors … and they are all blank



We read 10,000 randomly-chosen sectors … and they are all blank

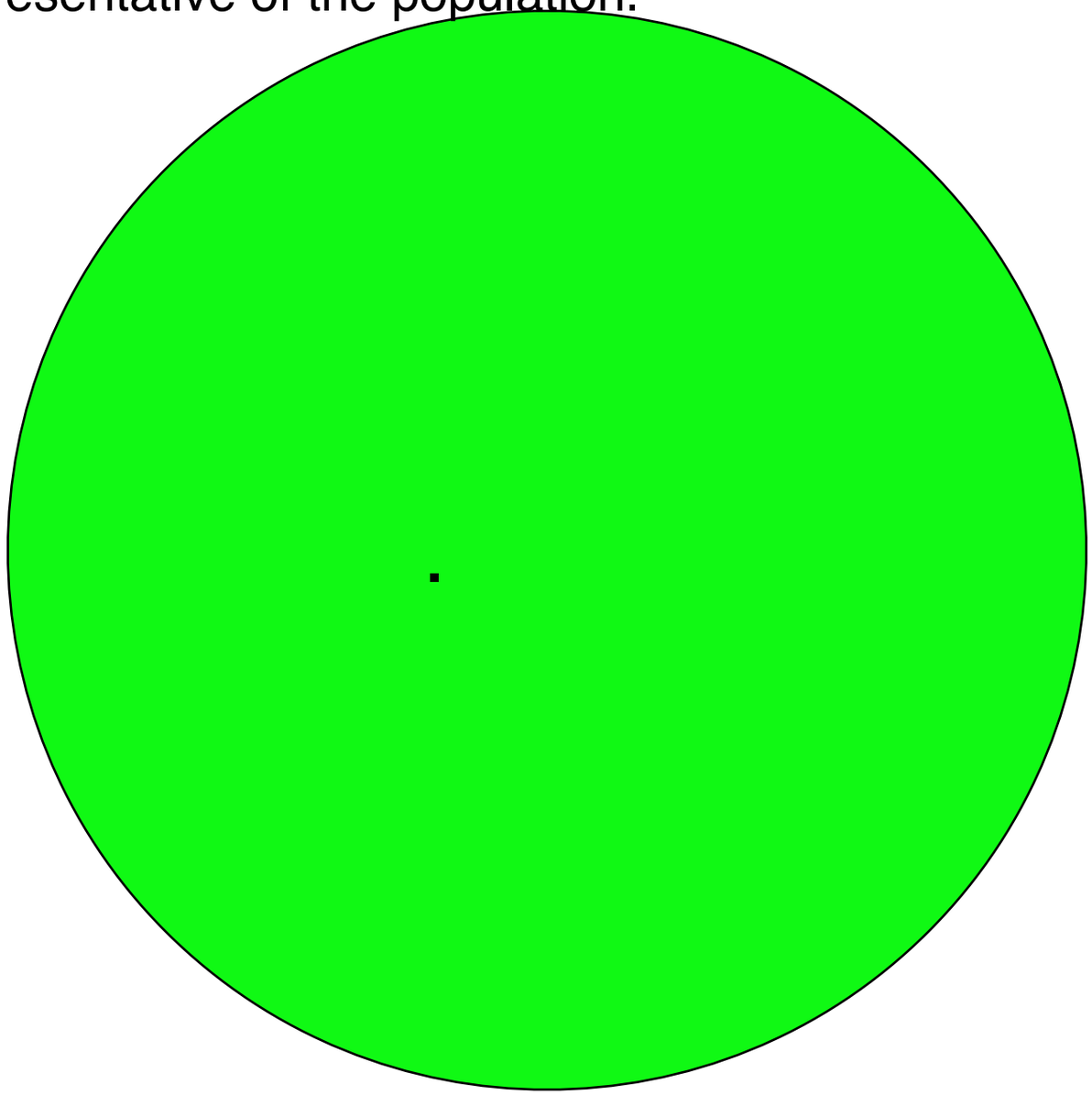


Chances are good that they are all blank.

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.

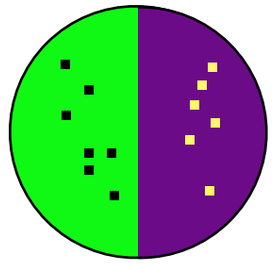


We will only find that 1 sector with exhaustive search.

What about other distributions?

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

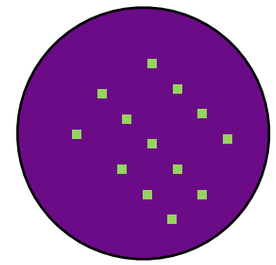
- The sampled frequency should match the distribution.
- *This is why we use random sampling.*



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 the disk has 10MB of data --- 20,000 non-zero sectors.

Read just 1 sector; the odds of finding a non-blank sector are:

$$\frac{200{,}000{,}000-20{,}000}{200{,}000{,}000} = 0.9999.$$

Read 2 sectors. The odds are:

$$\left(\frac{200{,}000{,}000-20{,}000}{200{,}000{,}000}\right)\left(\frac{199{,}999{,}999-20{,}000}{199{,}999{,}999}\right) = 0.99980001$$

first pick **second pick**

Odds we may have missed something

The more sectors picked, the less likely we are to miss all of the sectors that have non-NULL 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.

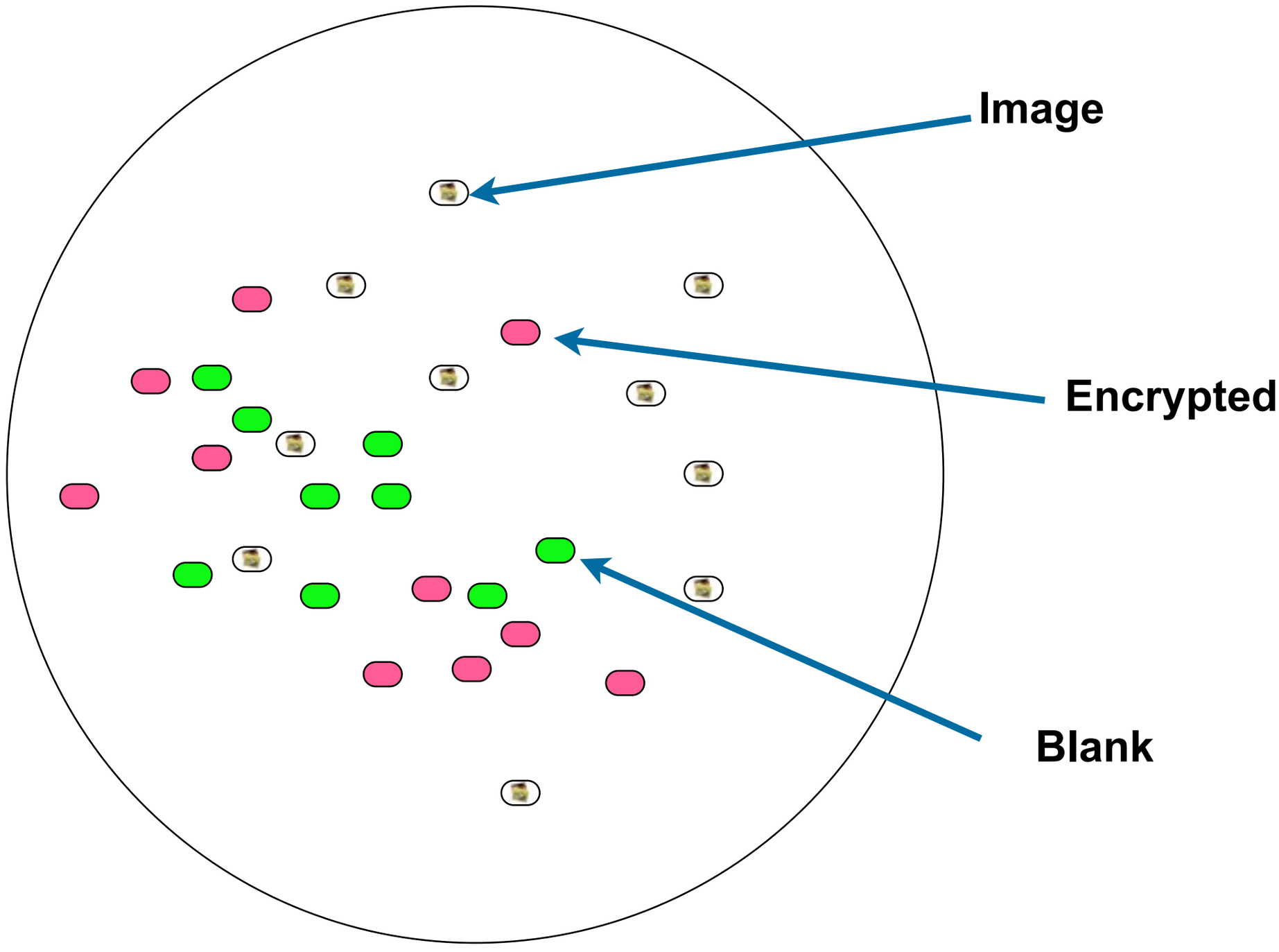
—So pick 500,000 random sectors. If they are all NULL, then the disk has p=(1-.00673) chance of having 10MB of non-NULL data.

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.

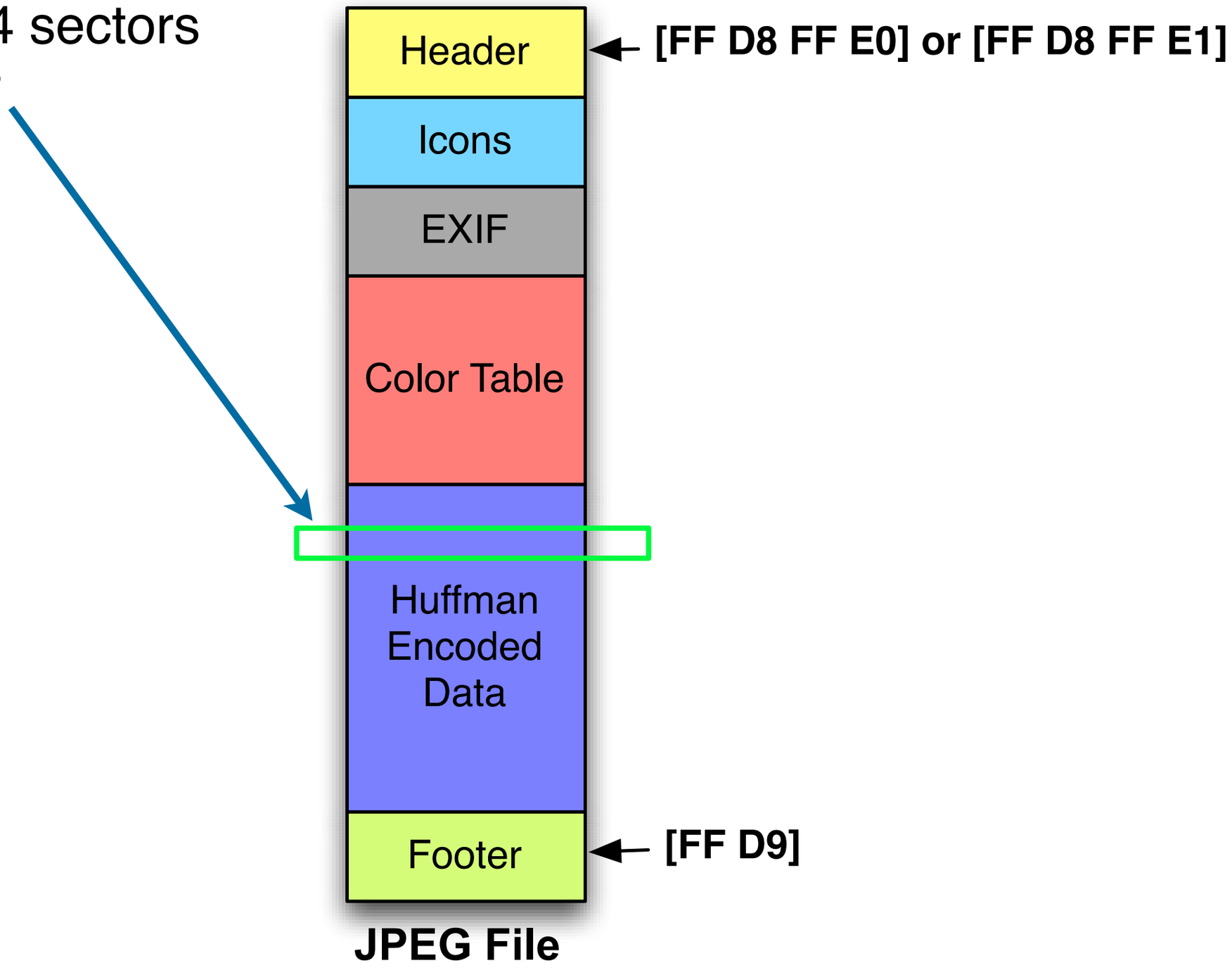
We can get a statistically significant sample in two minutes.



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?



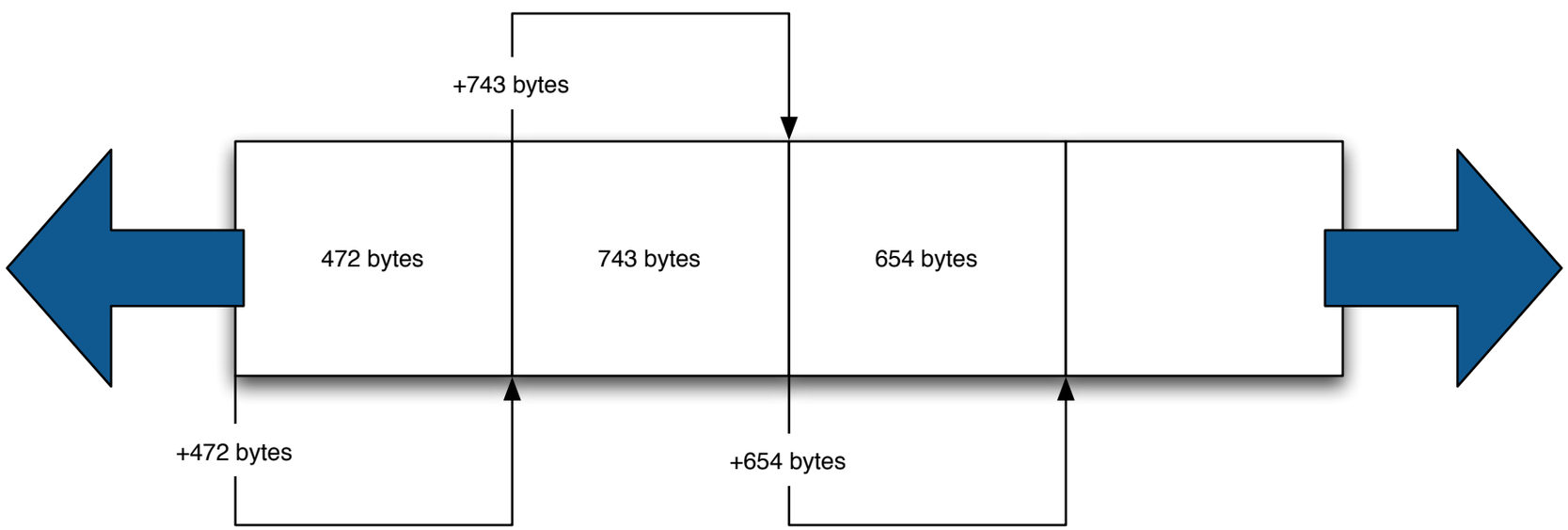
This is called the file fragment classification problem.

HTML files can be reliably detected with HTML tags

```
<body onload="document.getElementById('quicksearch').terms.focus()">
  <div id="topBar">
    <div class="widthContainer">
          <div id="skiplinks">
            <ul>
              <li>Skip to:</li>
```

MPEG files can be readily identified through framing

- Each frame has a header and a length.
- Find a header, read the length, look for the next header.



10 years of research on fragment identification…
… mostly using n-gram analysis (bigrams)

Standard approach:

- Get samples of different file types
- Train a classifier (typically k-nearest-neighbor or Support Vector Machines)
- Test classifier on "unknown data"

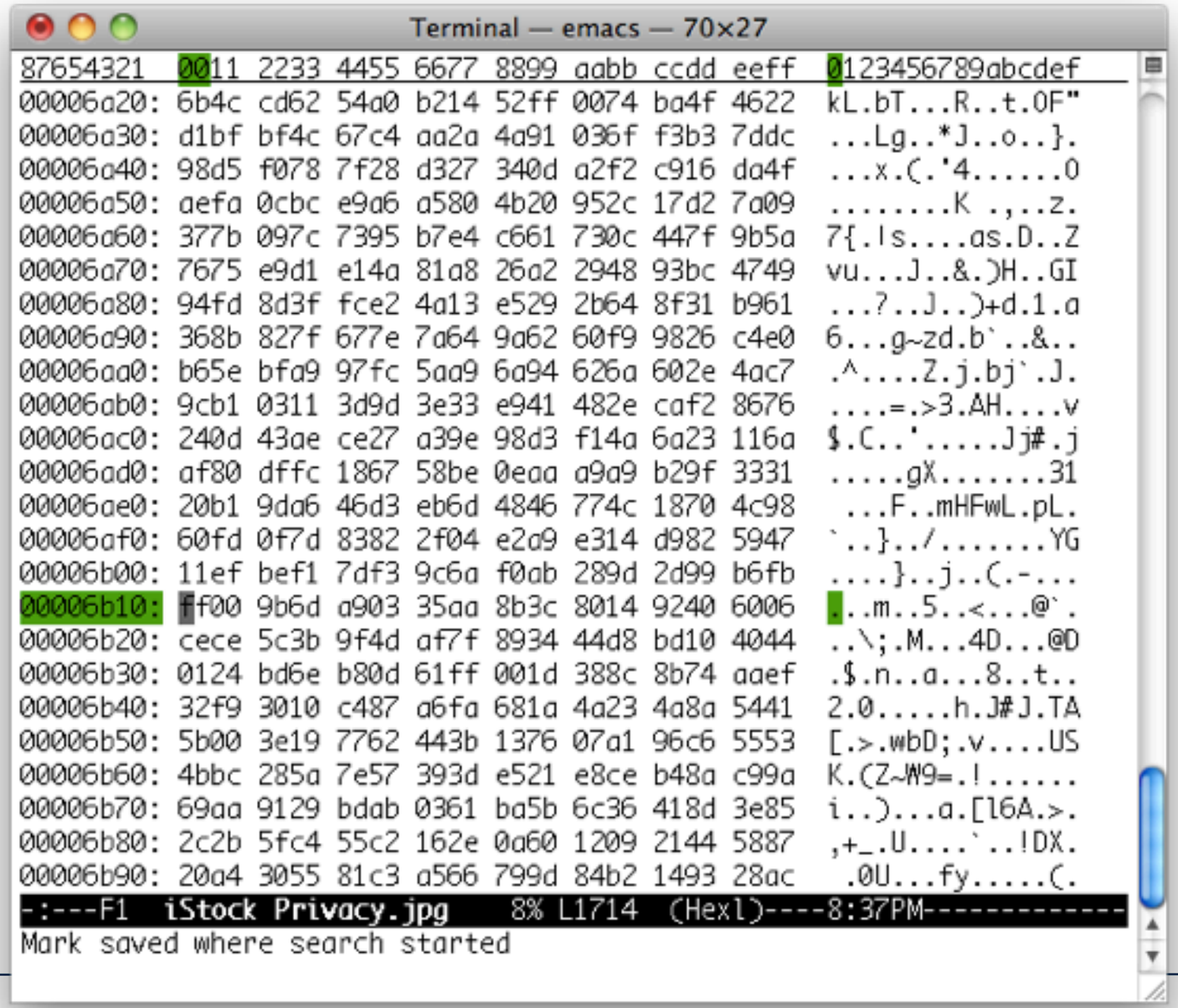
Examples:

- 2001 — McDaniel — "Automatic File Type Detection Algorithm"
  - —*header, footer & byte frequency (unigram) analysis (headers work best)*
- 2005 — LiWei-Jen et. al — "Fileprints"
  - —*unigram analysis*
- 2006 — Haggerty & Taylor — "FORSIGS"
  - —*n-gram analysis*
- 2007 — Calhoun — "Predicting the Type of File Fragments"
  - —*statistics of unigrams*

  - —*http://www.forensicswiki.org/wiki/File_Format_Identification*

Our approach: hand-tuned discriminators based on a close reading of the specification.

For example, the JPEG format "stuffs" FF with a 00.



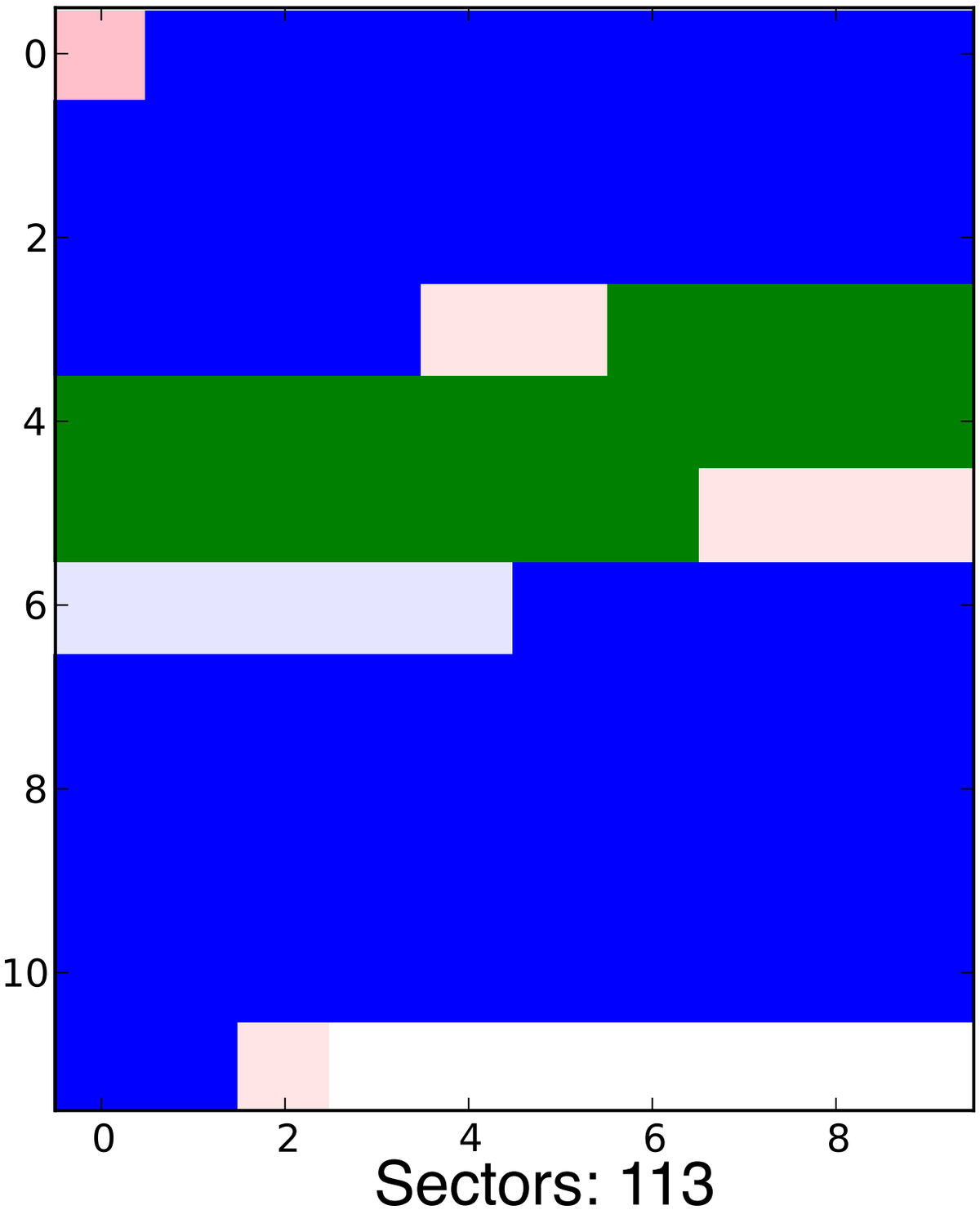
We built detectors to recognize the different parts of a JPEG file.

Bytes: 31,046

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

000897.jpg

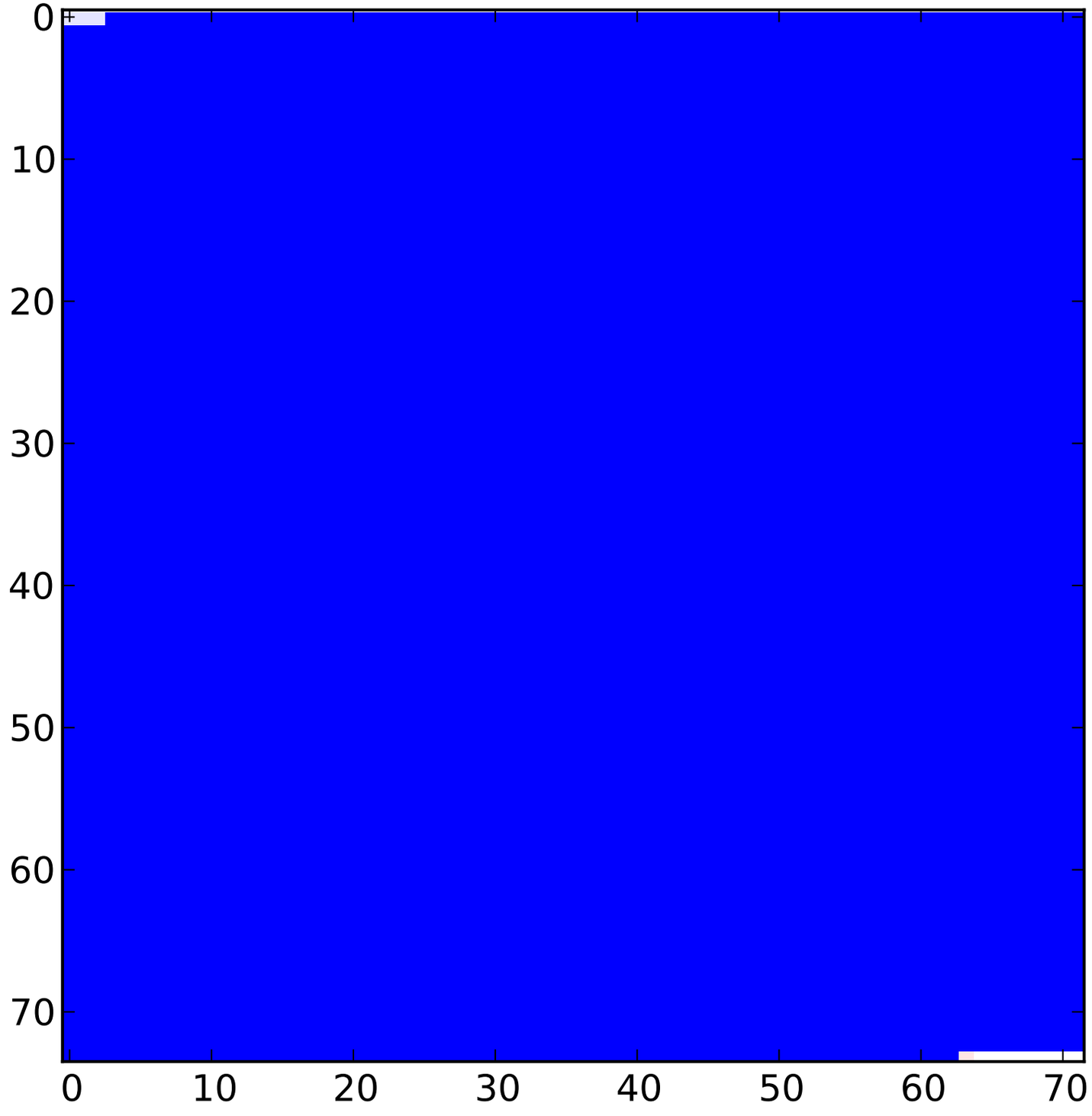
Bytes: 57596



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

000888.jpg

Bytes: 2,723,425



We developed five fragment discriminators.

JPEG — High entropy and FF00 pairs.

MPEG — Frames

Huffman-Coded Data — High Entropy & Autocorrelation

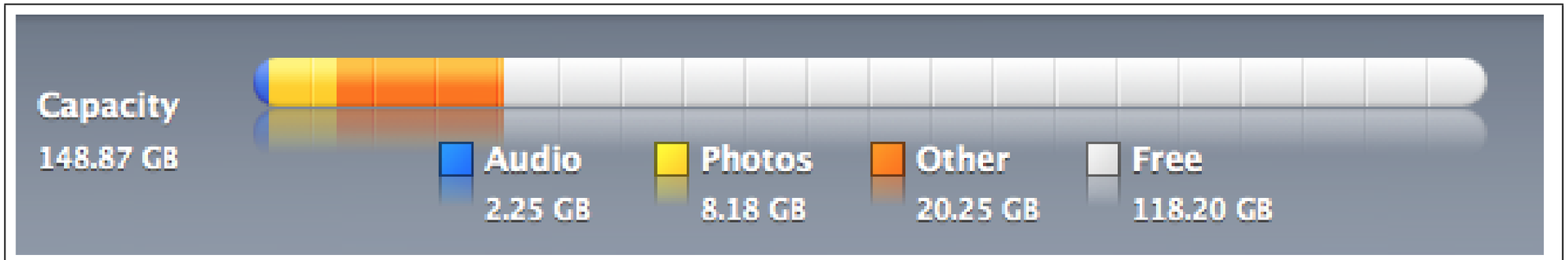
"Random" or "Encrypted" data — High Entropy & No autocorrelation

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

208 distinct 4096-byte block hashes

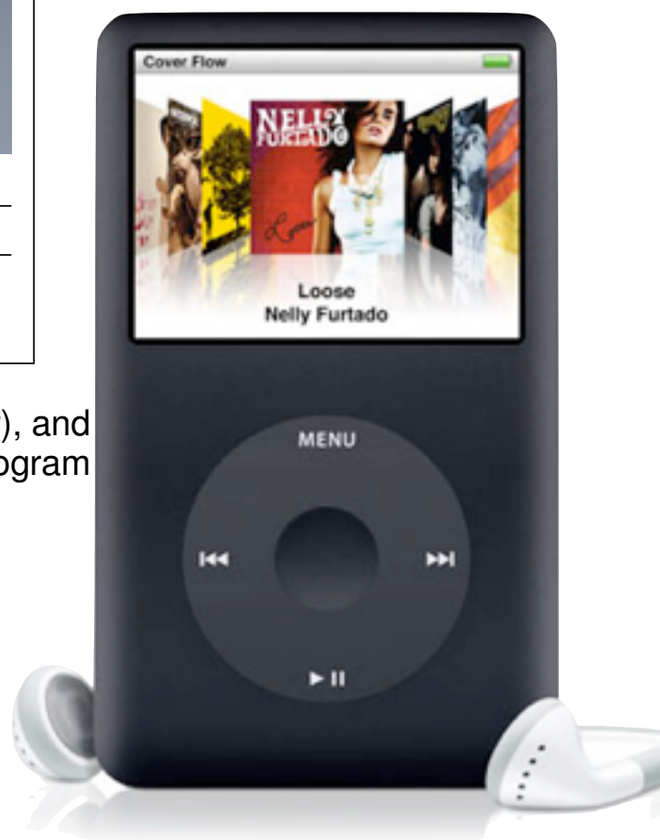
Combine random sampling with sector discrimination to obtain the forensic contents of a storage device.

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.



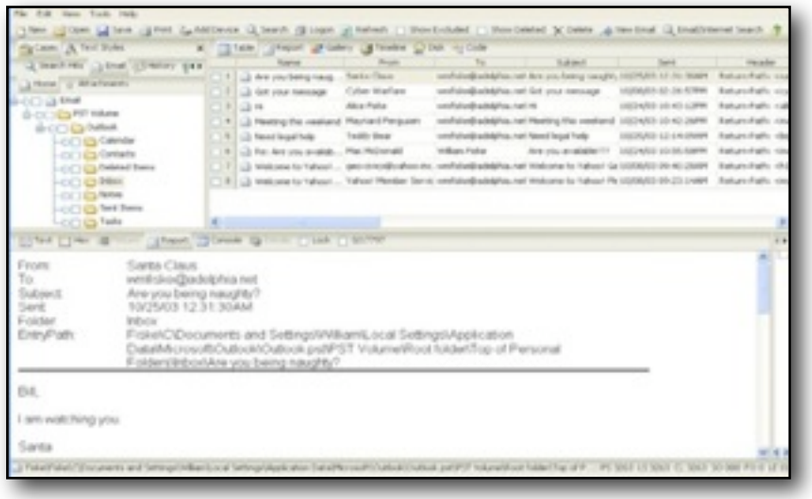
We accurately determined:

- % 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*

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.*

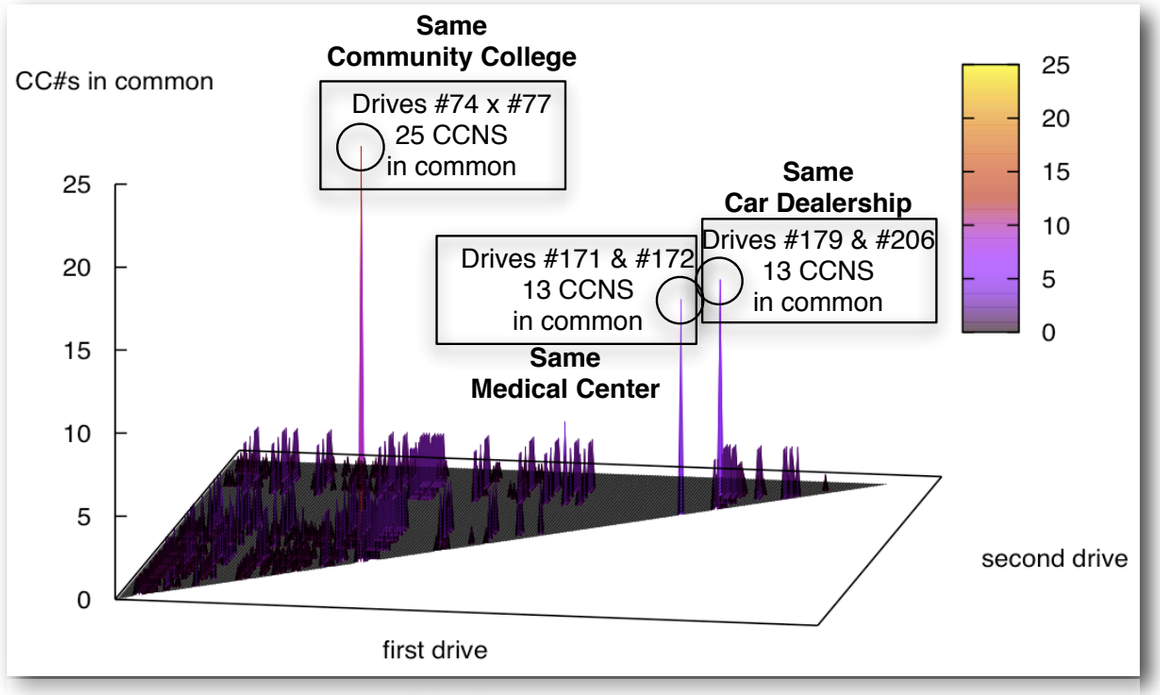
Evidence Reconstruction

- 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.

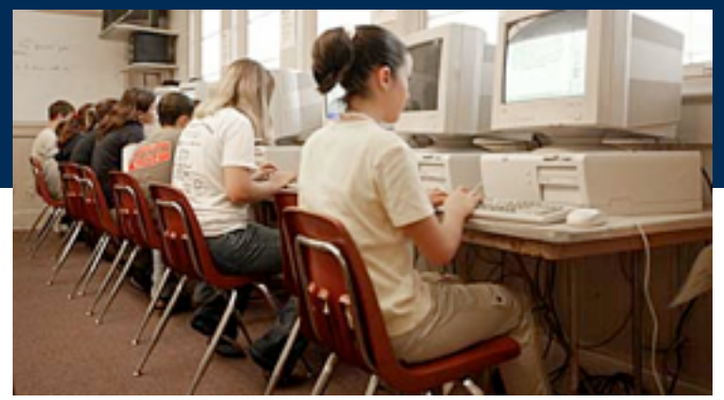
Why repeat some other scientist's experiment?

- 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.*

Digital Forensics education needs corpora too!



To teach forensics, we need real 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>

We manufacture data that 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*

We also acquire “real data” for use in research.

The Real Data Corpus (20TB)

- 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

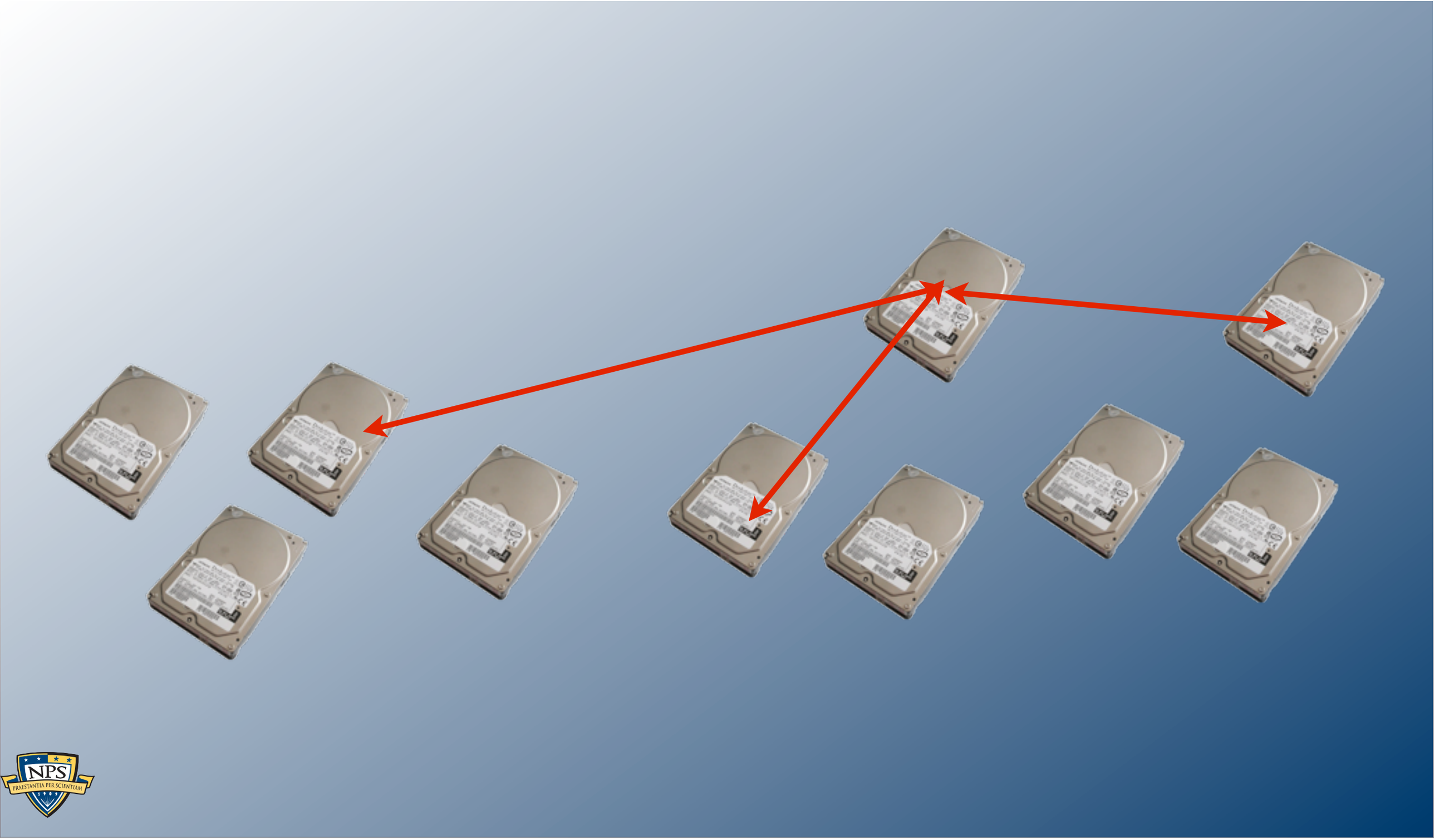
- Android Applications; Mobile Malware; etc.

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/

Real Data Corpus: Current Status

Country	HDs	Flash	Optical	GB (uncomp)
BA	7			38
CA	73	1		1,064
CE	1			82
CH	2			5
CN	143	568	98	3,627
DE	36	1		755
GR	13			27
IL	229	4		2,226
IN	317	66		19,540
MX	175			1,110
NZ	1			4
PS	98			957
TH	1			13
UA	22			55
	1,118	643	98	30,008



Where do we go from here?

There are a lot of fun projects to pursue.

Algorithms that work…

- With different kinds of data.
- At different resolutions & orders of magnitude.

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.

My 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.

In summary, 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

There are many opportunities to work outside CS

Need to engage with:

- Policy makers
- Police, Defense & Intelligence Communities.
- Defense Bar

Interesting legal issues.

- Data acquisition.
- Privacy
- Research Oversight (Institutional Review Boards.)

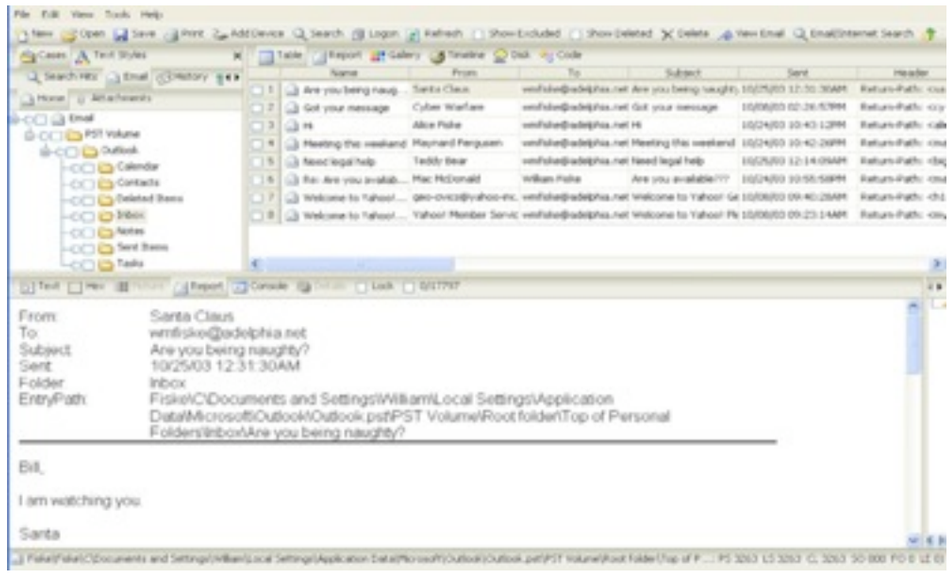
—*For more information, see http://www.simson.net/ or http://forensicswiki.org/*

Questions?

<DFXML>

Digital Forensics XML

Today's forensic tools are designed for performing forensic investigations.



Encase:
- GUI Closed Source
- EnScript

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

These tools are great for:

- File recovery
- Search

Not so great for automation, interoperability, or research.

Automation requires a forensics “language.” Standardized *formats* and *abstractions*.

Today we have limited formats and abstractions:

- Disk images — raw & EnCase E01 files
- Packet Capture files — BPF format
- Files — distributed as files or as ZIP for collections of files
- File Signatures — List of MD5 (or SHA1) hashes in hex with no context.
- “Selector Lists” — Lists of email address, CCNs, etc. (typically ASCII, rarely in Unicode)

We need new structured formats for distributing:

- Signatures Metrics (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

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

Today there is no good match between forensic tools and the needs of researchers.

Several of today's tools allow some degree of programmability:

- EnCase — EScript
- PyFlag — Flash Script & Python
- Sleuth Kit — C/C++

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.

Faced with this, a standard approach is to leverage the database:

- Extract everything into an SQL database.
- Use multiple SELECT statements to generate reports.
- SleuthKit 3.2 includes support for building SQLite databases
  - —*SQL schema are hard to extend.*

Digital Forensics XML:
An approach for standardizing forensic metadata

XML is well suited to forensics:

- We can represent a wide variety of data **today**.
- As our techniques improve, we can add new XML tags.
- More programmers speak XML than “forensics.”

Today we have XML tags to describe:

- Files and file metadata.
- Hash codes.
- Partitioning schemes.
- Application metadata.

— *We can use the same XML tags in many different applications.*

— *We can develop APIs to leverage the XML from python, perl, Java, etc.*

Example: <fileobject>

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

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

Simple example:

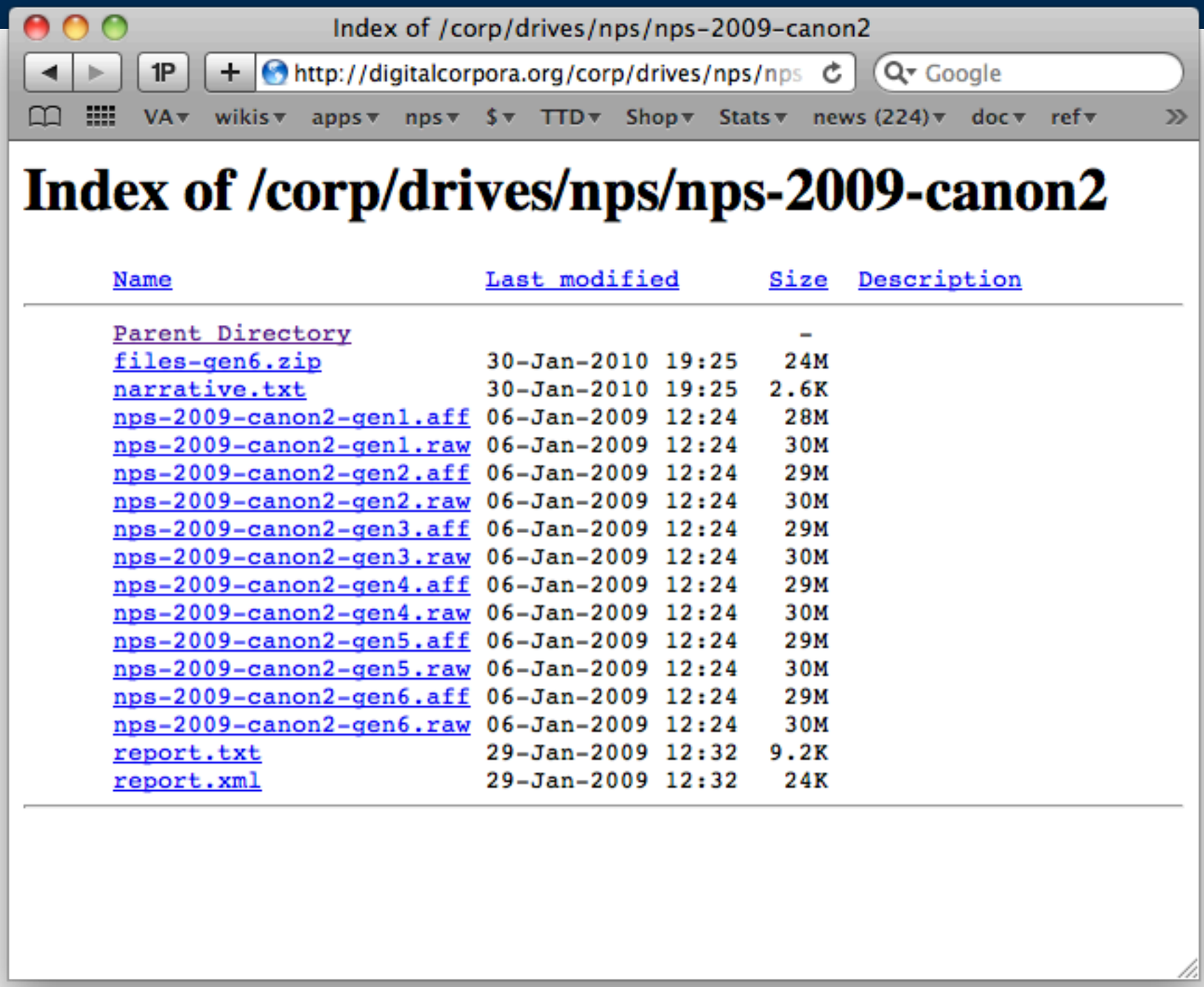
```
<fileobject>
<filename>samplefile.bin</filename>
<filesize>9014</filesize>
<mtime format='time_t'>1297835303.0</mtime>
<ctime format='time_t'>1297835303.0</ctime>
<atime format='time_t'>1299631657.0</atime>
<hashdigest type='MD5'>8dfcbdce6562602911990bdfd661415a</hashdigest>
<byte_runs>
  <run file_offset='0' len='9014' fs_offset='6553600' img_offset='6585856'/>
</byte_runs>
</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:

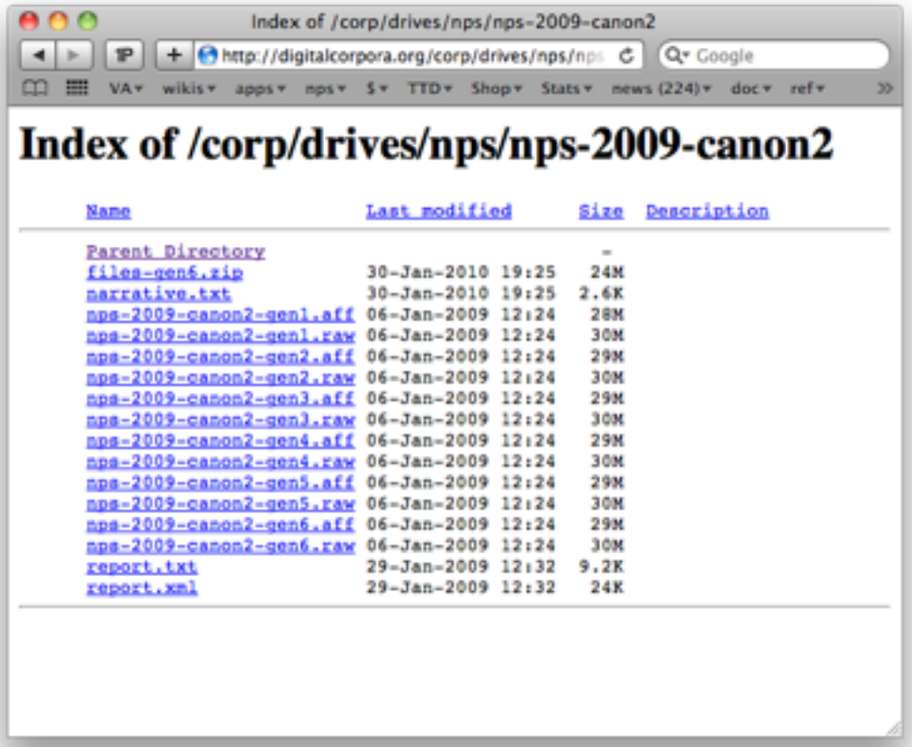
```
<?xml version='1.0' encoding='UTF-8'?>
<dfxml xmloutputversion='0.3'>
<metadata xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
  xmlns='http://afflib.org/fiwalk/'
  xmlns:dc='http://purl.org/dc/elements/1.1/'>

<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>
```

We use DFXML to describe what's in a disk image.



Researchers can quickly learn:

- Number of files on a disk, and their names.
- Human languages in use.
- Distribution of file types.
- Location of files on the disk



We have a growing list of tools that use DFXML

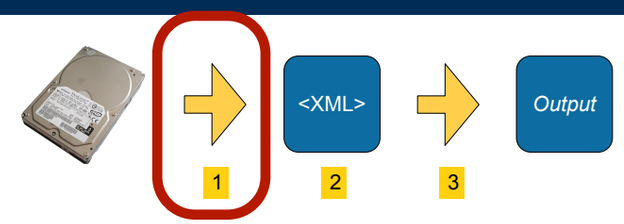
Generating DFXML:

- **fiwalk** — Creates DFXML from disk images. (Based on SleuthKit)
- **frag_find** — Hash-based carving; DFXML indicates where the files are in the disk image.
  - —*Used for malware detection, reassembling RAIDs, data exfiltration detection.*
- **dfxml_tool** — Generates DFXML hash lists from files.

Consuming DFXML:

- imap.py – Prints a prints a “map” of a disk image.
- iverify.py — Reports if the DFXML file matches a disk image.
- iredact.py — Removes or alters sensitive files in a disk image.
- iblkfind.py — Reports the file that maps to a given disk sector.
- idifference.py — Reports difference between two disk images.
- iexport.py — Exports the unallocated sectors.
- iextract.py — Extracts files of a given type.
- igrep.py — Reports the files in a disk image that match a string
- ihistogram.py — Fast histograms of the files on the disk
- ... and more

fiwalk extracts metadata from disk images.



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

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

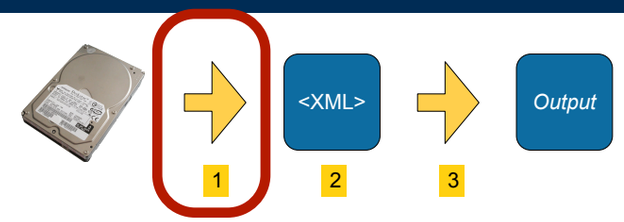
Features:

- 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)
- CSV (for spreadsheets)*

fiwalk provides limited control over extraction.



Include/Exclude criteria:

- Presence/Absence of file SHA1 in a Bloom Filter
- File name matching.

```
fiwalk -n .jpeg /dev/sda          # just extract the .jpeg files
```

File System Metdata:

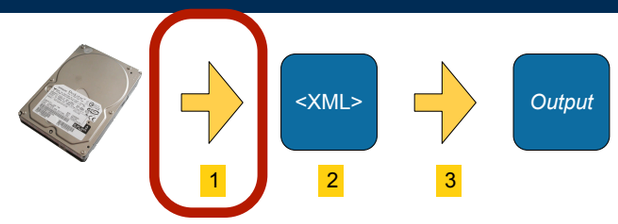
- -g — Report position of all file fragments
- -O — Do not report orphan or unallocated files

Full Content Options:

- -m — Report the MD5 of every file
- -1 — Report the SHA1 of every file
- -s *dir* — Save files to *dir*

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
```

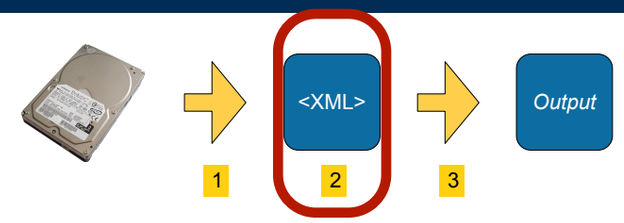
- *Plugins are run in a different process for safety.*
- *We have designed a native JVM interface which uses IPC and 1 process.*

Metadata extractors produce name:value pairs on STDOUT

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

Extracted metadata is automatically incorporated into output.

fiwalk produces four kinds of XML tags.



Provenance tags:

```
<fiwalk_version>0.4</fiwalk_version>
<Start_time>Mon Oct 13 19:12:09 2008</Start_time>
```

Per-Image tags:

```
<Imagefile>dosfs.dmg</Imagefile>
<volume startsector="512">
```

<volume> tags:

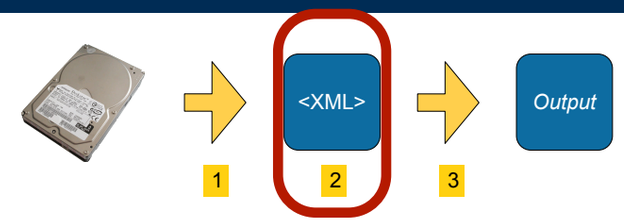
```
<Partition_Offset>512</Partition_Offset>
<block_size>512</block_size>
<ftype>4</ftype>
<ftype_str>fat16</ftype_str>
<block_count>81982</block_count>
```

<fileobject> tags:

```
<filesize>4096</filesize>
<partition>1</partition>
<filename>linedash.gif</filename>
<libmagic>GIF image data, version 89a, 410 x 143</libmagic>
```

<byte_runs> specifies data's physical location.

One or more <run> elements may be present:



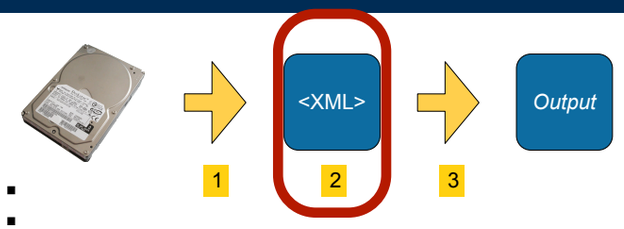

```
<byte_runs type='resident'>

  <run file_offset='0' len='65536'
       fs_offset='871588864' img_offset='871621120'/>

  <run file_offset='65536' len='25920'
       fs_offset='871748608' img_offset='871780864'/>

</byte_runs>
```

This file has two fragments:

- 64K starting at sector 1702385 (871621120 ÷ 512)
- 25,920 bytes starting at sector 1702697 (871780864 ÷ 512)

Additional XML attributes may specify compression or encryption.

XML incorporates the extracted metadata.



fiwalk metadata extractors produce name:value pairs:

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

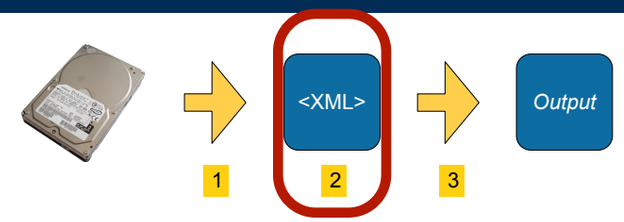
These are incorporated into XML:

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

— UTF-8 — Special characters are automatically escaped.

Resulting XML files can be distributed with images.

The XML file provides a key to the disk image:

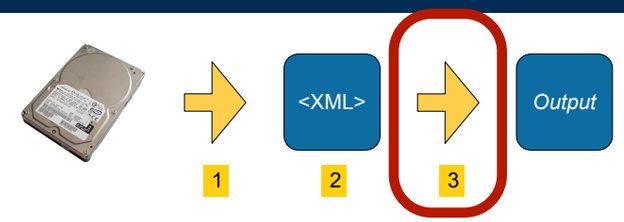



```
$ ls -l /corp/images/nps/nps-2009-domexusers/

-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.*

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.

Multiple interfaces:

- SAX callback interface

```
fiwalk_using_sax(imagefile, xmlfile, flags, callback)
```

  — *Very fast and minimal memory footprint*

- SAX procedural interface

```
objs = fileobjects_using_sax(imagefile, xmlfile, flags)
```

  — *Reasonably fast; returns a list of all file objects with XML in dictionary*

- DOM procedural interface

```
(doc,objs) = fileobjects_using_dom(imagefile, xmlfile, flags)
```

  — *Allows modification of XML that's returned.*

The SAX and DOM interfaces both return fileobjects!

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

Objects belong to one of two subclasses:

`fileobject_sax(fileobject)` — ***for the SAX interface***
`fileobject_dom(fileobject)` — ***for the DOM interface***

Both classes support the same interface:

- *fi.partition()*
- *fi.filename(), fi.ext()*
- *fi.filesize()*
- *fi.ctime(), fi.atime(), fi.crtime(), fi.mtime()*
- *fi.sha1(), fi.md5()*
- *fi.byteruns(), fi.fragments()*
- *fi.content()**

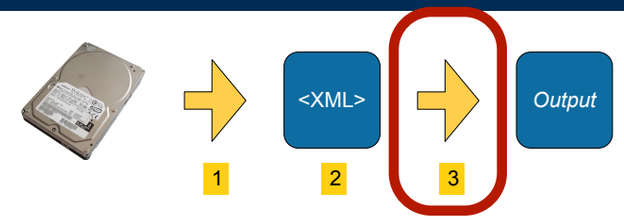
Example: igrep.py

```
import fiwalk

if __name__=="__main__":
    import sys

    from optparse import OptionParser
    parser = OptionParser()
    parser.usage = '%prog [options] image.iso  s1'
    parser.add_option("-d","--debug",help="debug",action="store_true")
    (options,args) = parser.parse_args()

if len(args)!=2:
    parser.print_help()
    sys.exit(1)

(image,data) = args

def process(fi):
    offset = fi.contents().find(data)
    if offset>0:
        print "%s (offset=%d)" % (str(fi),offset)

fiwalk.fiwalk_using_sax(imagefile=image),callback=process)
```

igrep.py in action

```
$ python igrep.py /corp/drives/nps/nps-2009-canon2/nps-2009-canon2-gen6.raw
Firmware

fileobject DCIM/100CANON/IMG_0044.JPG byte_runs: byterun[img_offset=114688;
file_offset=0 bytes=32768]  byterun[img_offset=1523712; file_offset=32768
bytes=32768]  byterun[img_offset=6356992; file_offset=65536 bytes=39659]
(offset=1228)

fileobject DCIM/100CANON/IMG_0042.JPG byte_runs: byterun[img_offset=147456;
file_offset=0 bytes=1361807]  (offset=1228)

fileobject DCIM/100CANON/IMG_0003.JPG byte_runs: byterun[img_offset=1835008;
file_offset=0 bytes=840101]  (offset=1228)

fileobject DCIM/100CANON/IMG_0043.JPG byte_runs: byterun[img_offset=1556480;
file_offset=0 bytes=278528]  byterun[img_offset=2686976; file_offset=278528
bytes=1474560]  byterun[img_offset=4407296; file_offset=1753088 bytes=786432]
byterun[img_offset=6062080; file_offset=2539520 bytes=294498]  (offset=1228)

fileobject DCIM/100CANON/IMG_0045.JPG byte_runs: byterun[img_offset=6406144;
file_offset=0 bytes=110686]  (offset=1228)
...
```

Example: Find and print all the files 15 bytes in length.



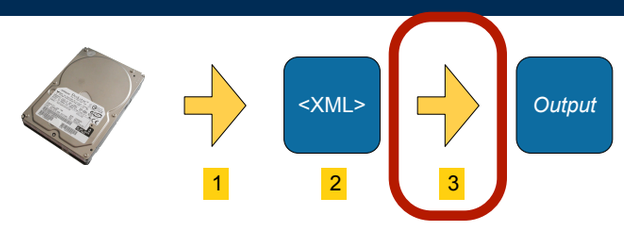
Using DOM interface:

```
import fiwalk

objs = fileobjects_using_sax(imagefile, xmlfile, flags)
for fi in filter(lambda x:x.filesize()==15, objs):
    print fi
```

(For the Python-impaired:)

```
import fiwalk
objs = fileobjects_using_sax(imagefile, xmlfile, flags)
for fi in objs:
    if fi.filesize()==15:
        print fi
```

The fileobject class allows direct access to file data.



byteruns() is an array of “runs.”

```
<byte_runs type='resident'>

  <run file_offset='0' len='65536'
       fs_offset='871588864' img_offset='871621120'/>

  <run file_offset='65536' len='25920'
       fs_offset='871748608' img_offset='871780864'/>

</byte_runs>
```

Becomes:

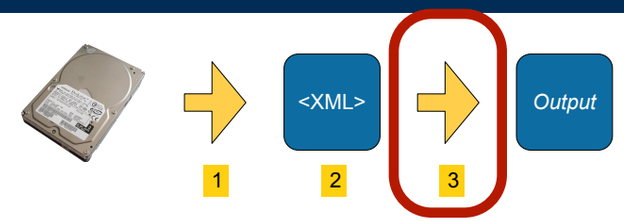
```
[byterun[offset=0; bytes=65536], byterun[offset=65536; bytes=25920]]
```

Each byterun object has:

```
run.start_sector()   – Starting Sector #
run.sector_count()
run.img_offset       - Disk Image offset
run.fs_offset        - File system offset
run.bytes            - number of bytes
run.content()        - content of file
```

The fileobject class allows direct access to file data.

byteruns() returns that array of “runs”
for both the DOM and SAX-based file objects.




```
>>> print fi.byteruns()
[byterun[offset=0; bytes=65536], byterun[offset=65536; bytes=25920]]
```

Accessor Methods:

- fi.contents_for_run(run) — *Returns the bytes from the linked disk image*
- fi.contents() — *Returns all of the contents*
- fi.file_present(imagefile=None) — Validates MD5/SHA1 to see if image has file
- fi.tempfile(calMD5,calcSHA1) — Creates a tempfile, optionally calculating hash

Publishing XML for disk images enables our remote exploitation methodology...

xmlrpc

Extract metadata in Boston.
Search from Monterey.
Just download what you need.

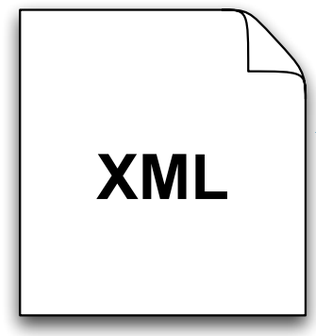
Publishing XML for disk images enables our remote exploitation methodology...

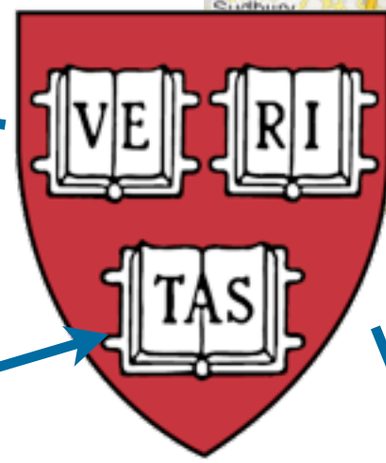
xmlrpc

AFF

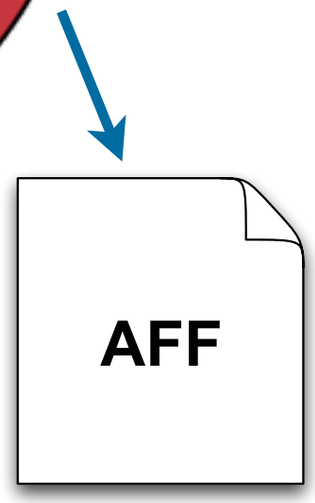
Extract metadata in Boston.
Search from Monterey.
Just download what you need.

Publishing XML for disk images enables our remote exploitation methodology...



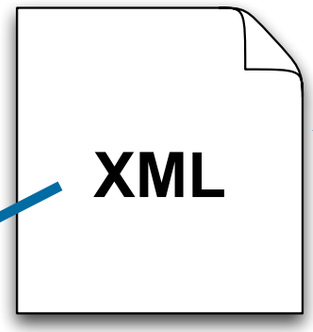


xmlrpc

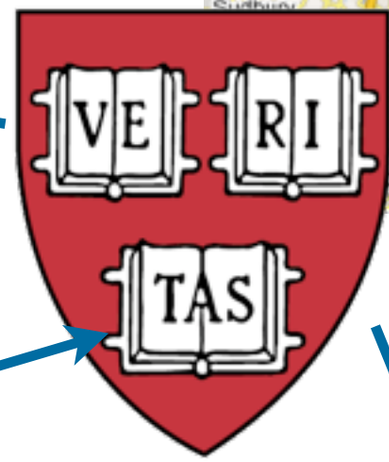


Extract metadata in Boston.
Search from Monterey.
Just download what you need.

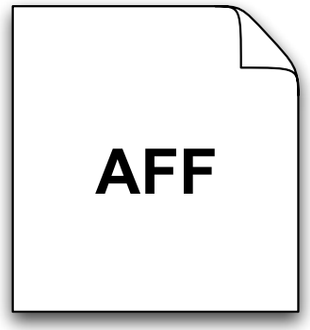
Publishing XML for disk images enables our remote exploitation methodology...



http

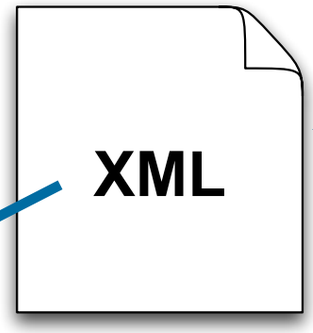


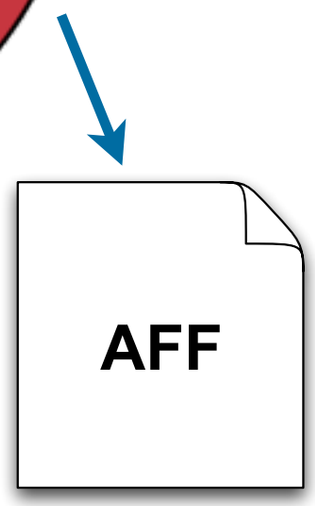
xmlrpc



Extract metadata in Boston.
Search from Monterey.
Just download what you need.

Publishing XML for disk images enables our remote exploitation methodology...





http

xmlrpc

Extract metadata in Boston.
Search from Monterey.
Just download what you need.

Question: how much time can we save in forensic analysis by processing files in *sector order?*

Currently, forensic programs process in directory order.

```
for (dirpath,dirnames,filenames) in os.walk("/mnt"):
    for filename in filenames:
        process(dirpath+"/"+filename)
```

Advantages of processing by sector order:

- Minimizes head seeks.

Disadvantages:

- Overhead to obtain file system metadata (but you only need to do it once).
- File fragmentation means you can't do a perfect job:

Using the architecture presented here, I performed the experiment.

Here's most of the program:

```
t0 = time.time()
fis = fiwalk.fileobjects_using_sax(imagefile)
t1 = time.time()
print "Time to get metadata: %g seconds" % (t1-t0)

print "Native order: "
calc_jumps(fis,"Native Order")
fis.sort(key=lambda(a):a.byteruns()[0].img_offset)
calc_jumps(fis,"Sorted Order")
```

With this XML framework, it took less than 10 minutes to write the program that conducted the experiment.

Answer: Processing files in sector order can improve performance *dramatically*.

	Unsorted	Sorted
Files processed:	23,222	23,222
backwards seeks	12,700	4,817
Time to extract metadata:	19 seconds	19 seconds
Time to read files:	441 seconds	38 seconds
Total time:	460 seconds	57 seconds

disk image: nps-2009-domexusers1

DFXML: Current Status

Working today:

- Programs for producing and consuming DFXML.
- A set of tags that can represent:
  - *Files & Metadata*
  - *Hashes*
  - *Time*
  - *Bloom Filters*

What we are working on:

- Bindings for languages other than python
- Use of DFXML in a cluster/HPC environment.
- Use of DFXML for digital archives and in other forensic communities.

What we need:

- Support from tool vendors (primary carvers)
- Support within the research community.