Drive Analysis in a Flash

Simson L. Garfinkel, Ph.D.

Associate Professor

Naval Postgraduate School

http://simson.net/

slgarfin@nps.edu

This presentation is about research being performed at the Naval Postgraduate School

Location: Monterey, CA

Campus Size: 627 acres

1500 Students:

- ⇨US Military (All 5 services)
- ⇨US Civilian (SFS & SMART)
- ⇨Foreign Military (30 countries)

4 Schools:

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

Hard drive forensics is facing the I/O Barrier.

Seagate Barracuda 7200.11 SATA 3Gb/s 1.5-TB Hard Drive

- ⇨7200 rpm
- ⇨120 MB/s sustained data rate
- ⇨32-MB cache
- ⇨$125.90 (including shipping)

1,500,301,910,016 bytes
------------------------------- = 12,502 seconds = 3 hours, 28 min
120,000,000 bytes/s

Disk performance is determined by data transfer rate and by seek time.

The Seagate Barracuda has a "random read seek time" of <8.5 msec.

Seek time = travel time + stabilization time.

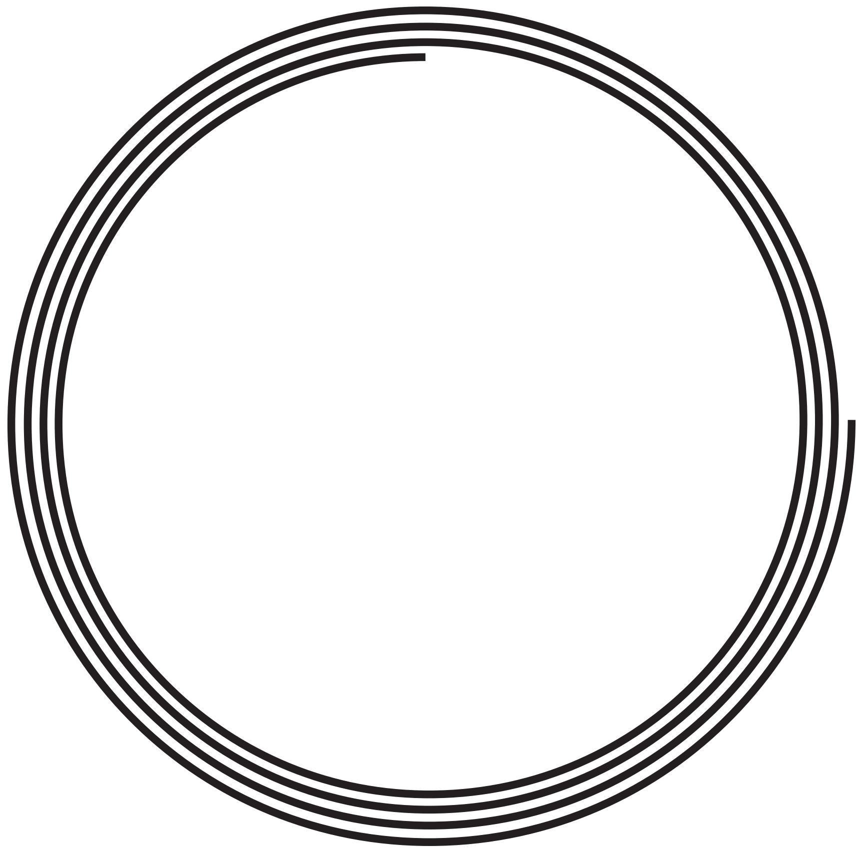
1 sec ÷ 8.5 msec/seek ≈ 1200 seeks/sec

Reading all of the data in 3.5 hours requires minimizing seeks.

Start at sector #0

End at sector #2,930,277,167

208 minutes x 7200 RPM = 1,500,300 revolutions

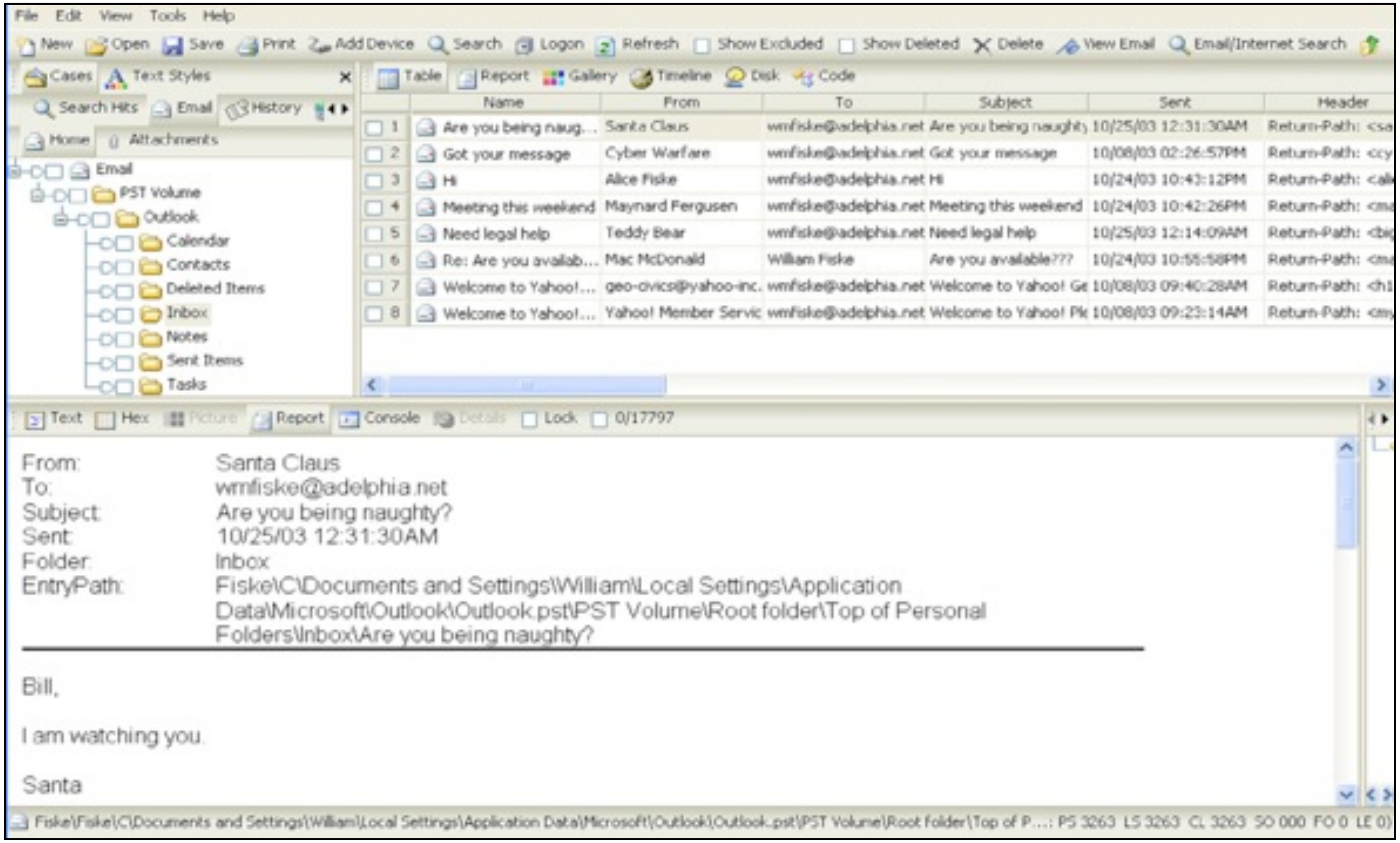


Unfortunately, data on hard drives is not laid out in consecutive sectors.

Most forensic programs read file-by-file:

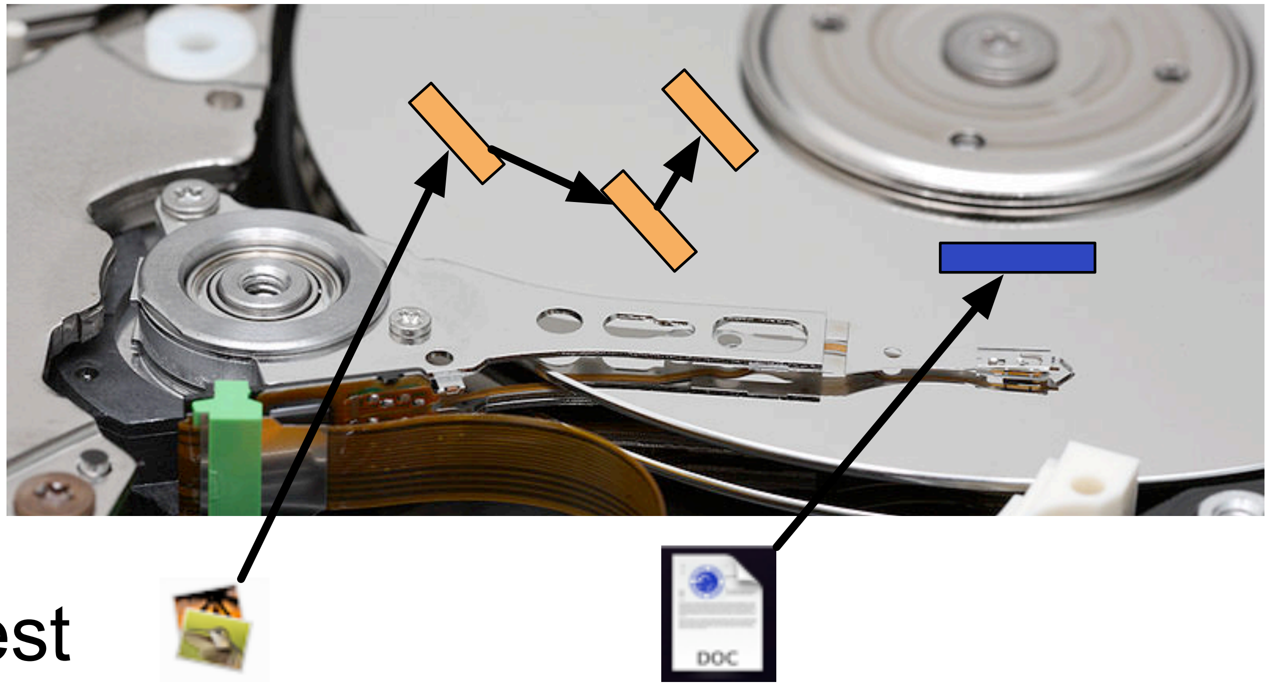
Guidance Software's EnCase is designed to:

- ⇨Recover deleted photos & documents.
- ⇨Recover email messages



Information is stored in different areas of the disk

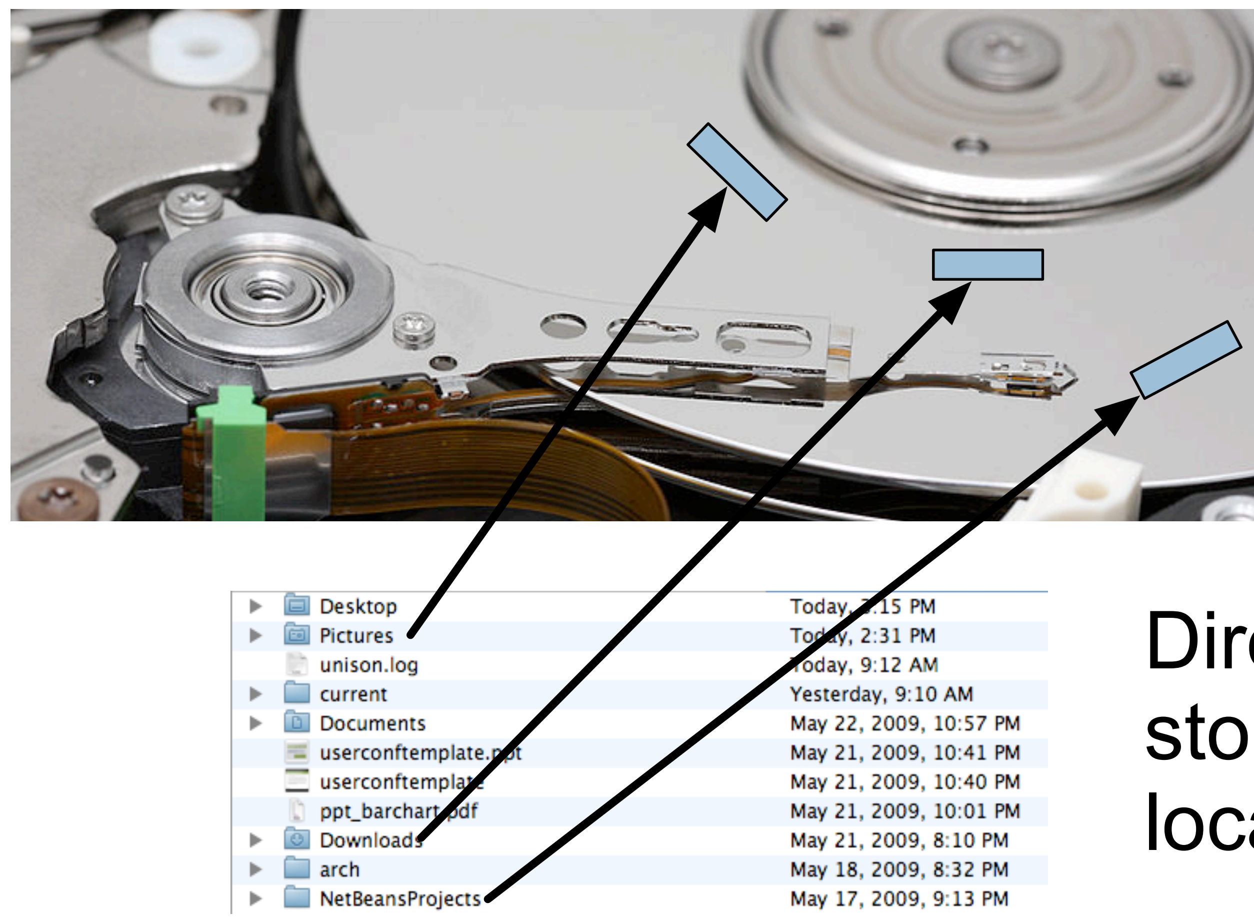
Information is stored in different areas of the disk



Documents of interest are stored in many areas.

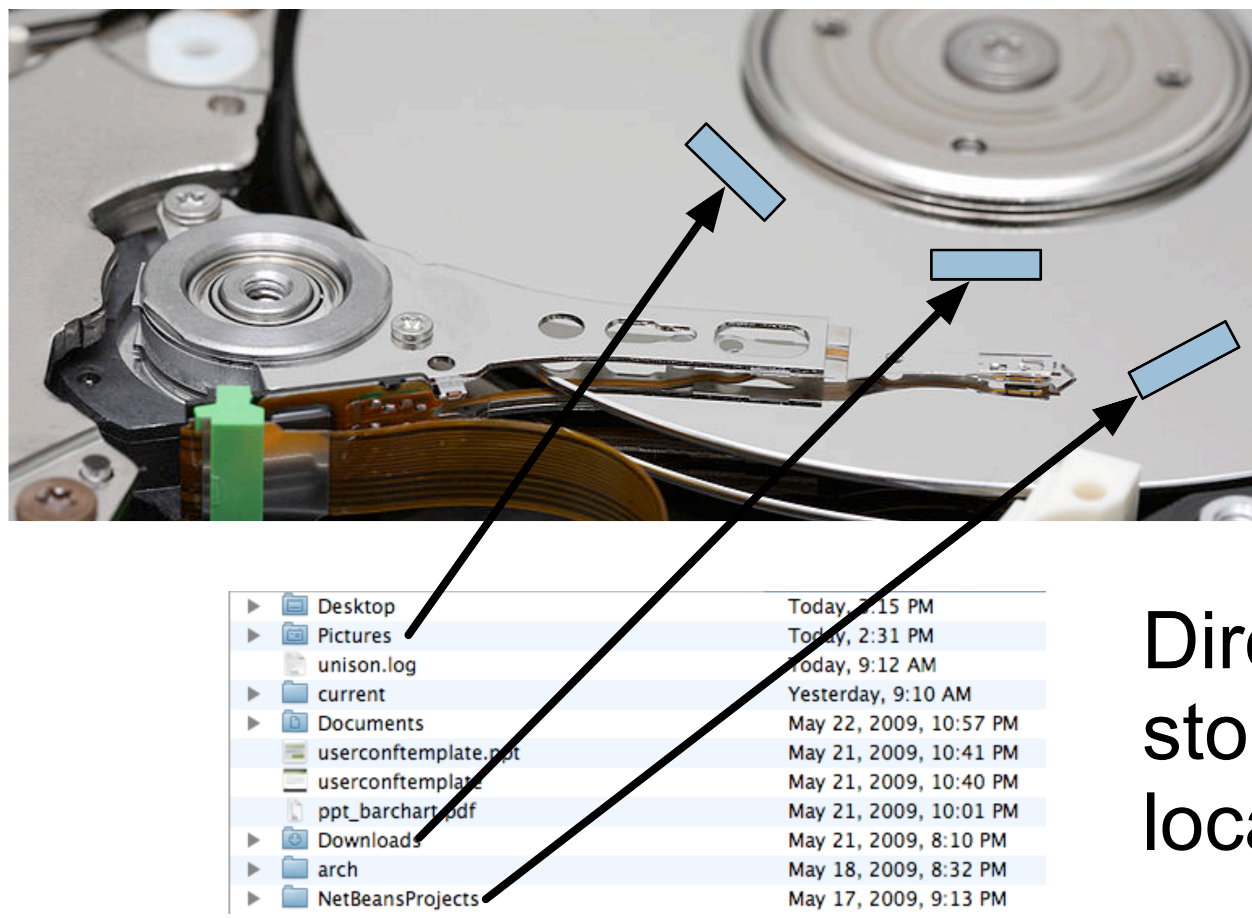
Some documents are "fragmented" into multiple locations

Information is stored in different areas of the disk



Directories are stored in many locations.

Information is stored in different areas of the disk

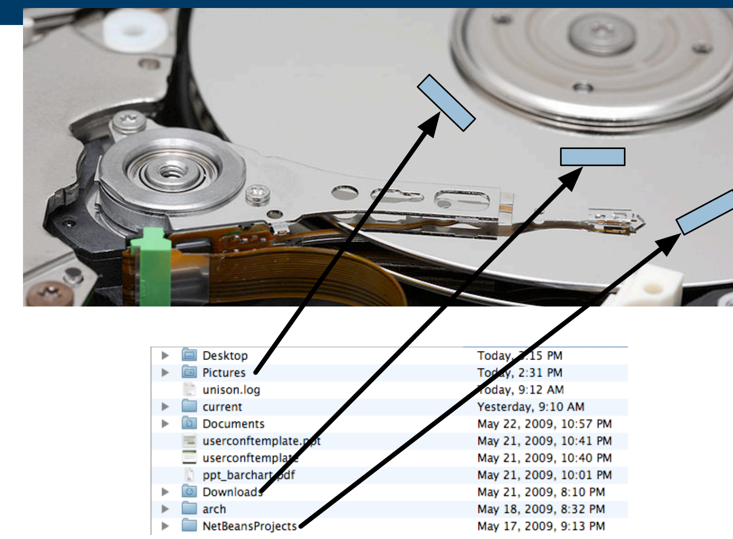


Directories are stored in many locations.

Because data is stored in different locations, it can take 10-30 hours to ingest a large disk.

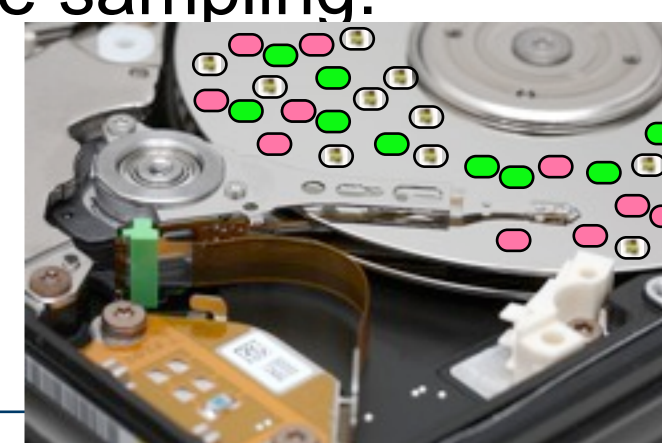
This talk presents two approaches for speeding disk forensics.

How is data arranged on a hard disk?



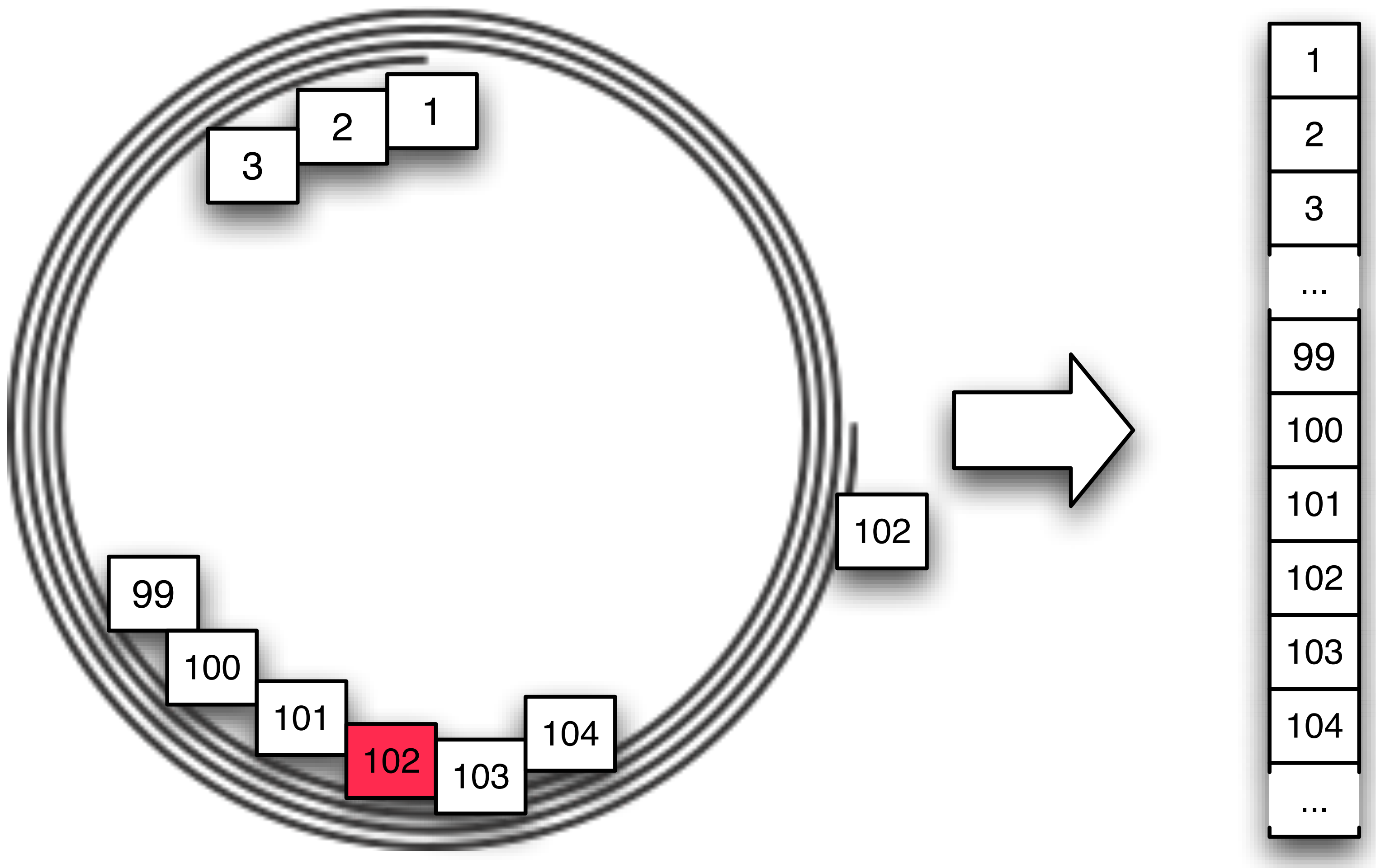
Approach #1: Speeding traditional forensics by considering disk order.

Approach #2: "Instant disk forensics" with drive sampling.



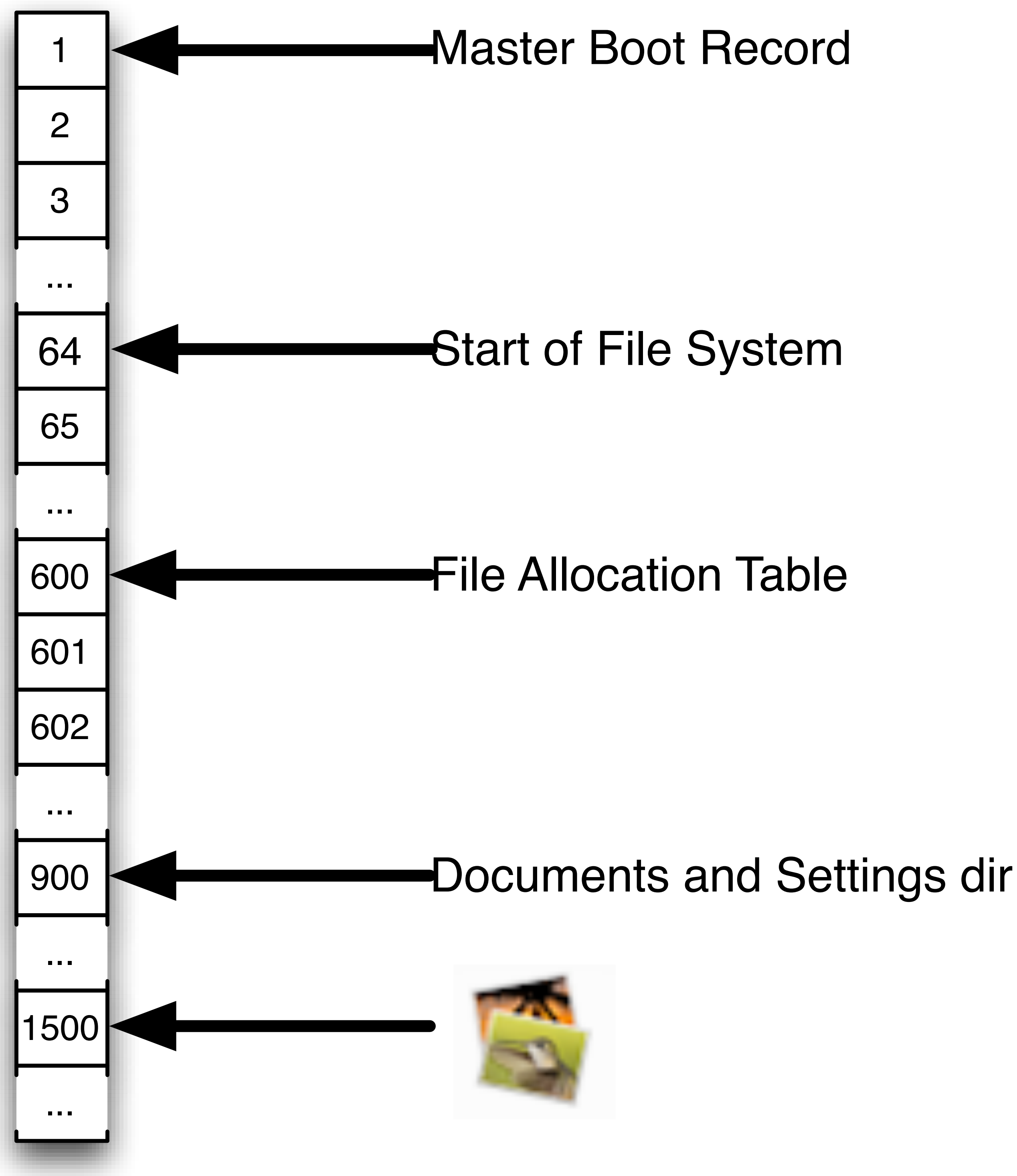
Speeding traditional disk forensics.

Modern disk drives address sectors by *logical block number.*



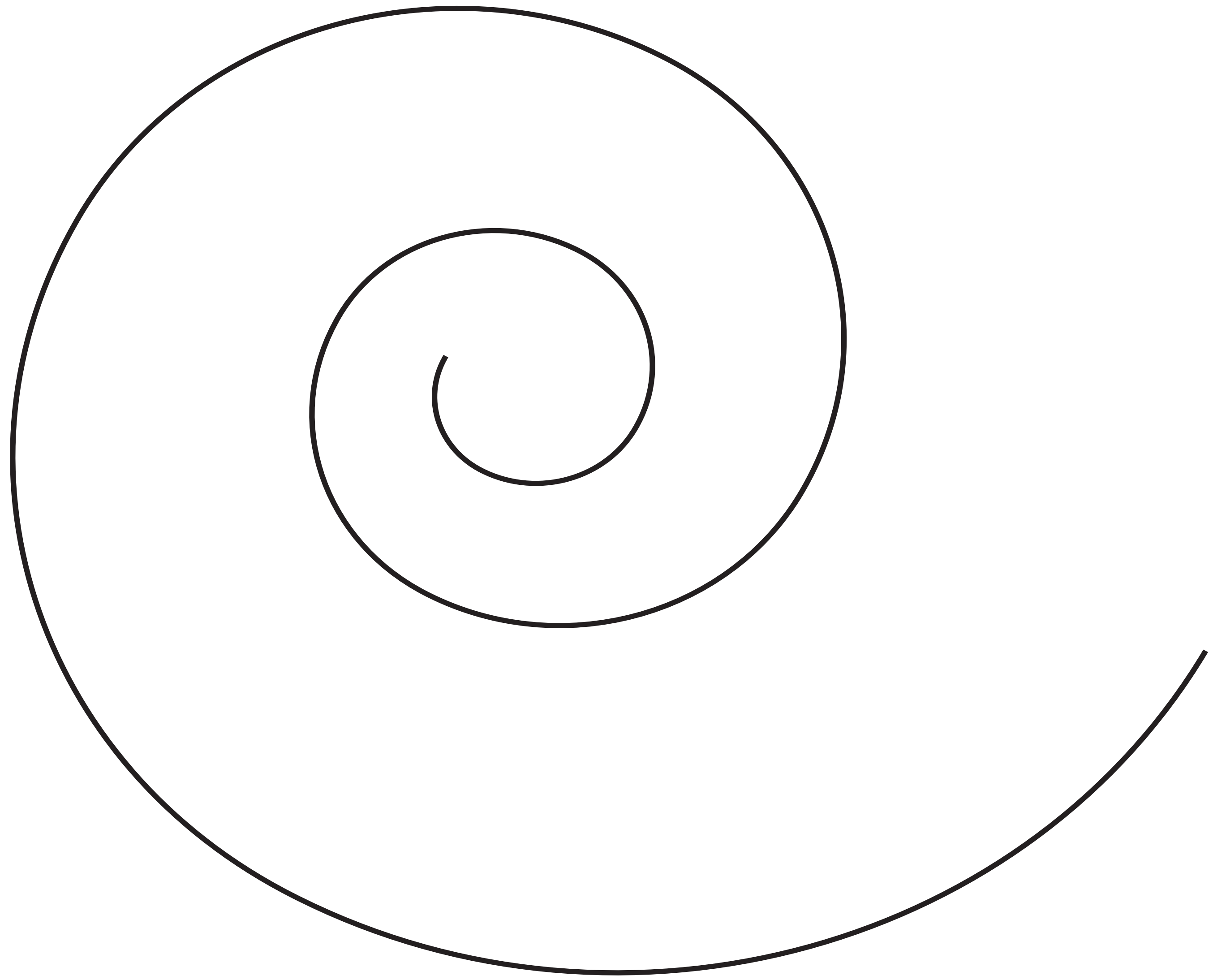
Bad sectors are transparently remapped.

Different parts of the file system are stored at different blocks.



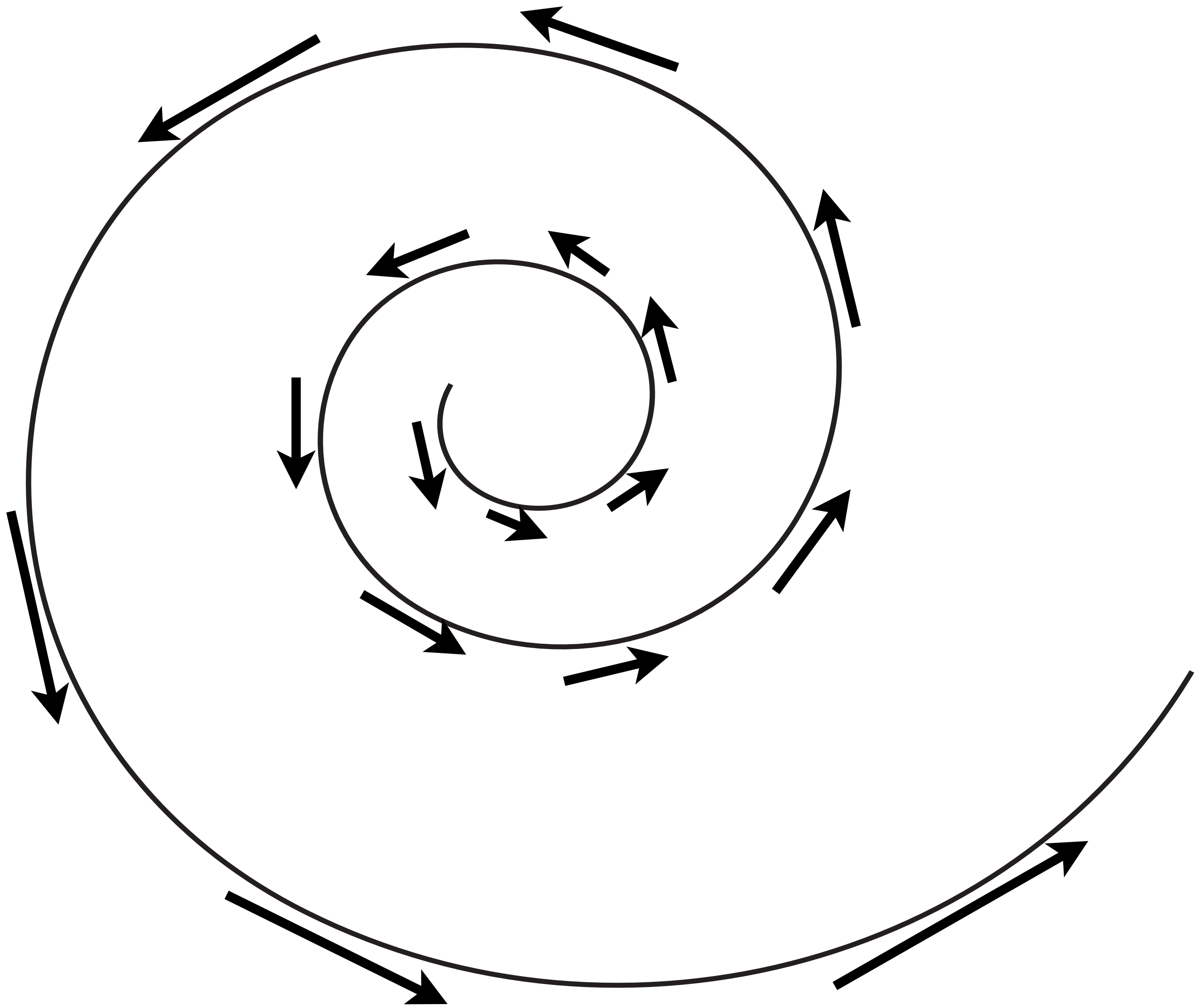
Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory*.

Imaging is done at maximum disk transfer rate:



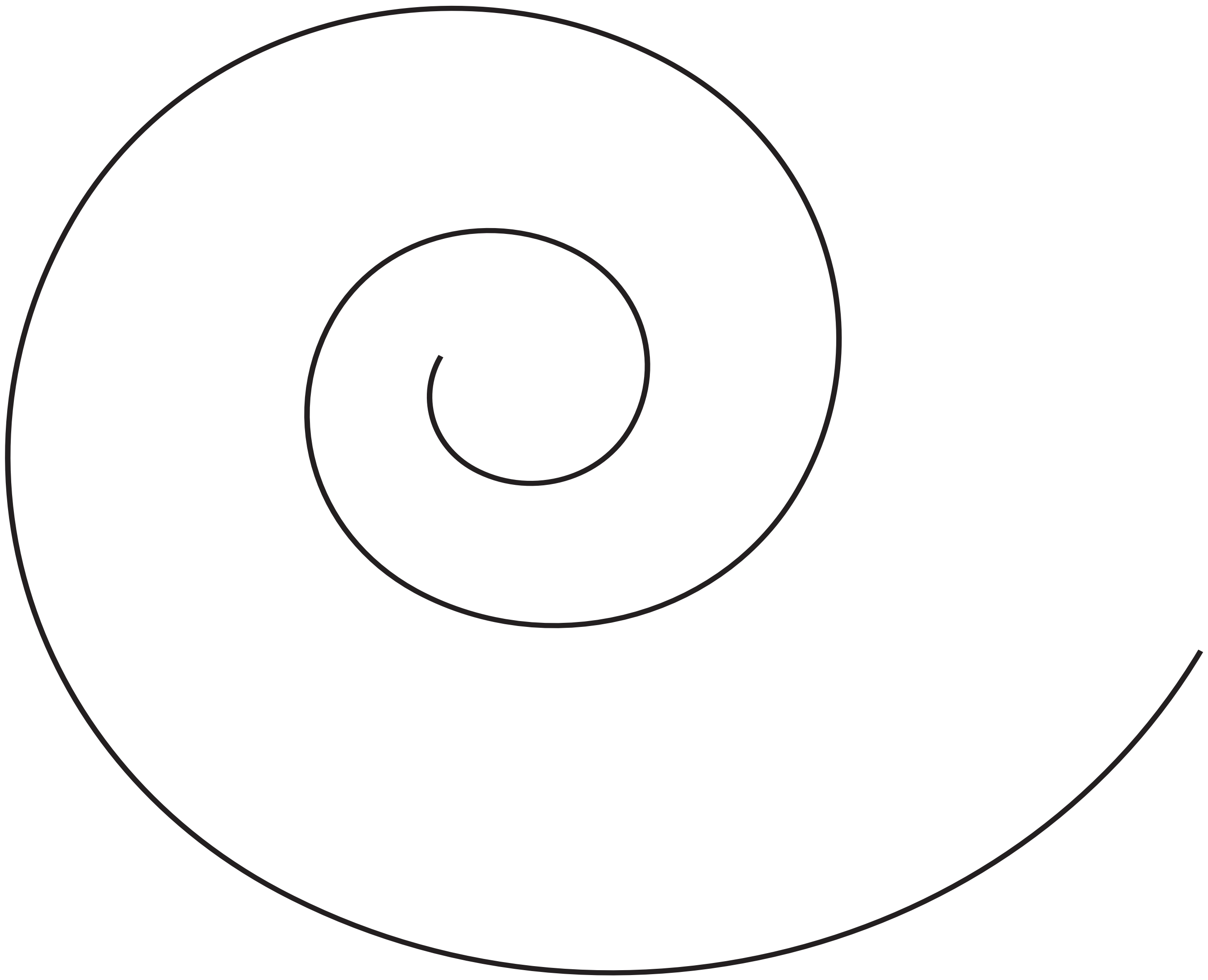
Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory*.

Imaging is done at maximum disk transfer rate:



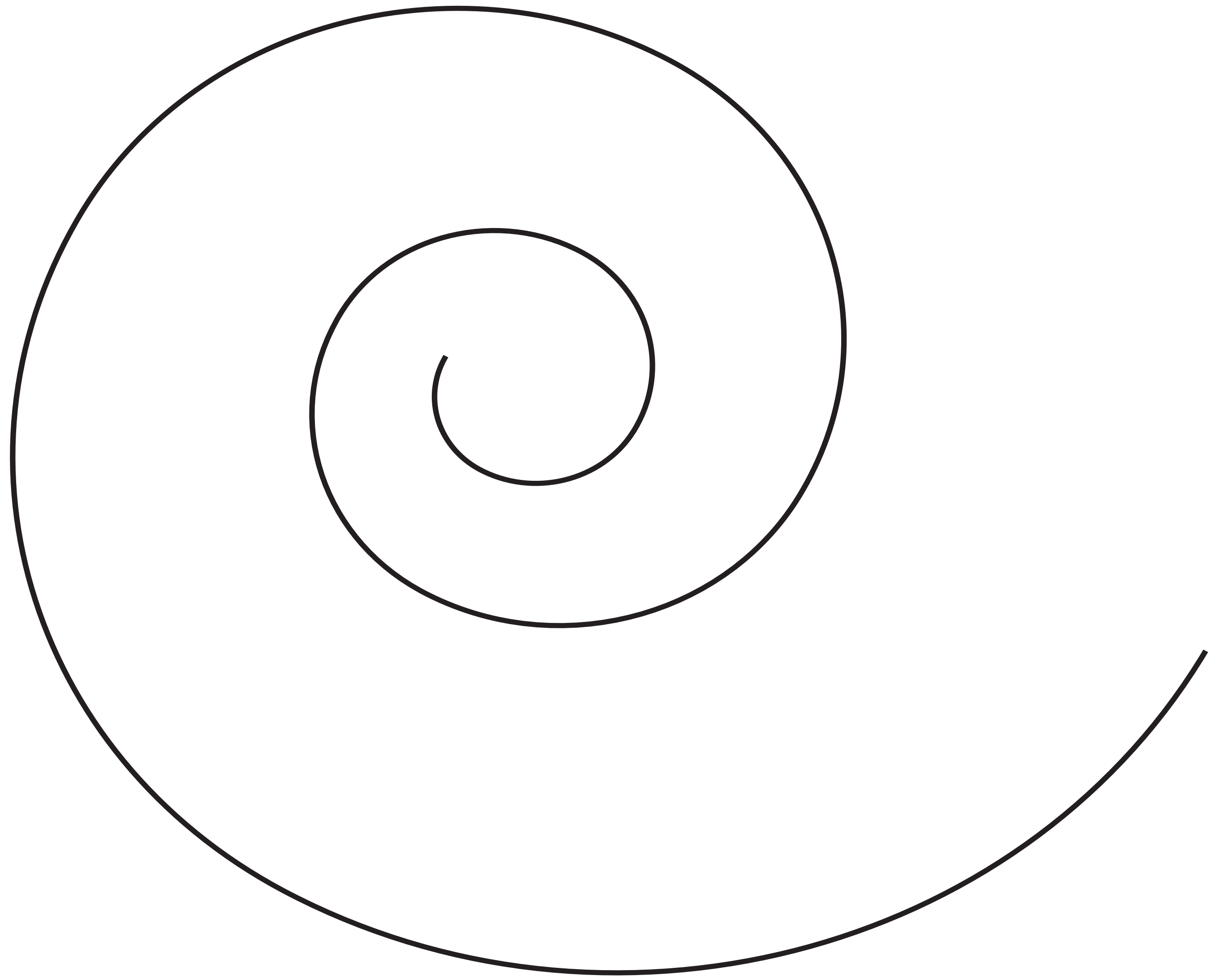
Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory*.

Imaging is done at maximum disk transfer rate:



Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory.*

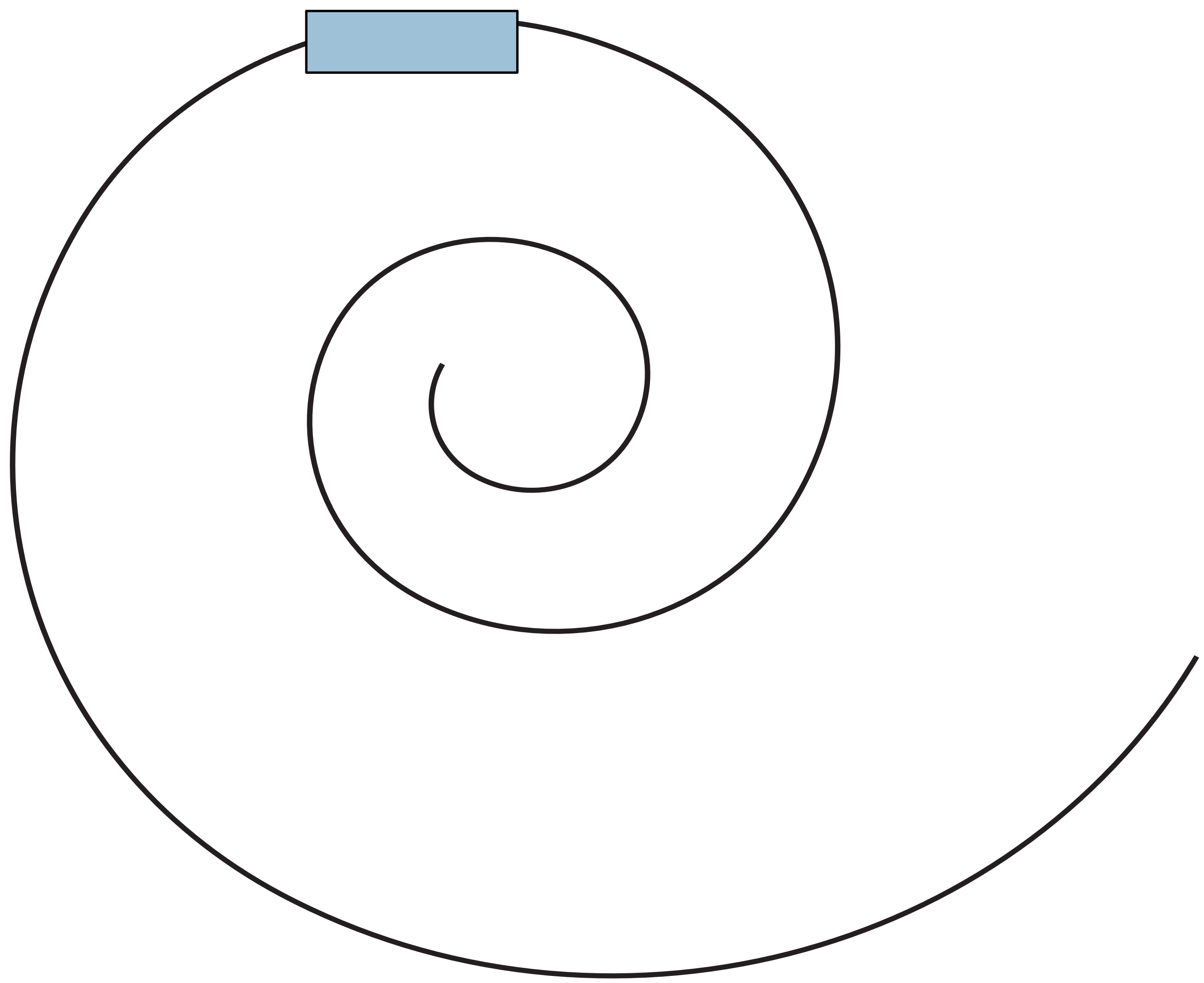
Imaging is done at maximum disk transfer rate:



But ingesting is done directory-by-directory.

Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory.*

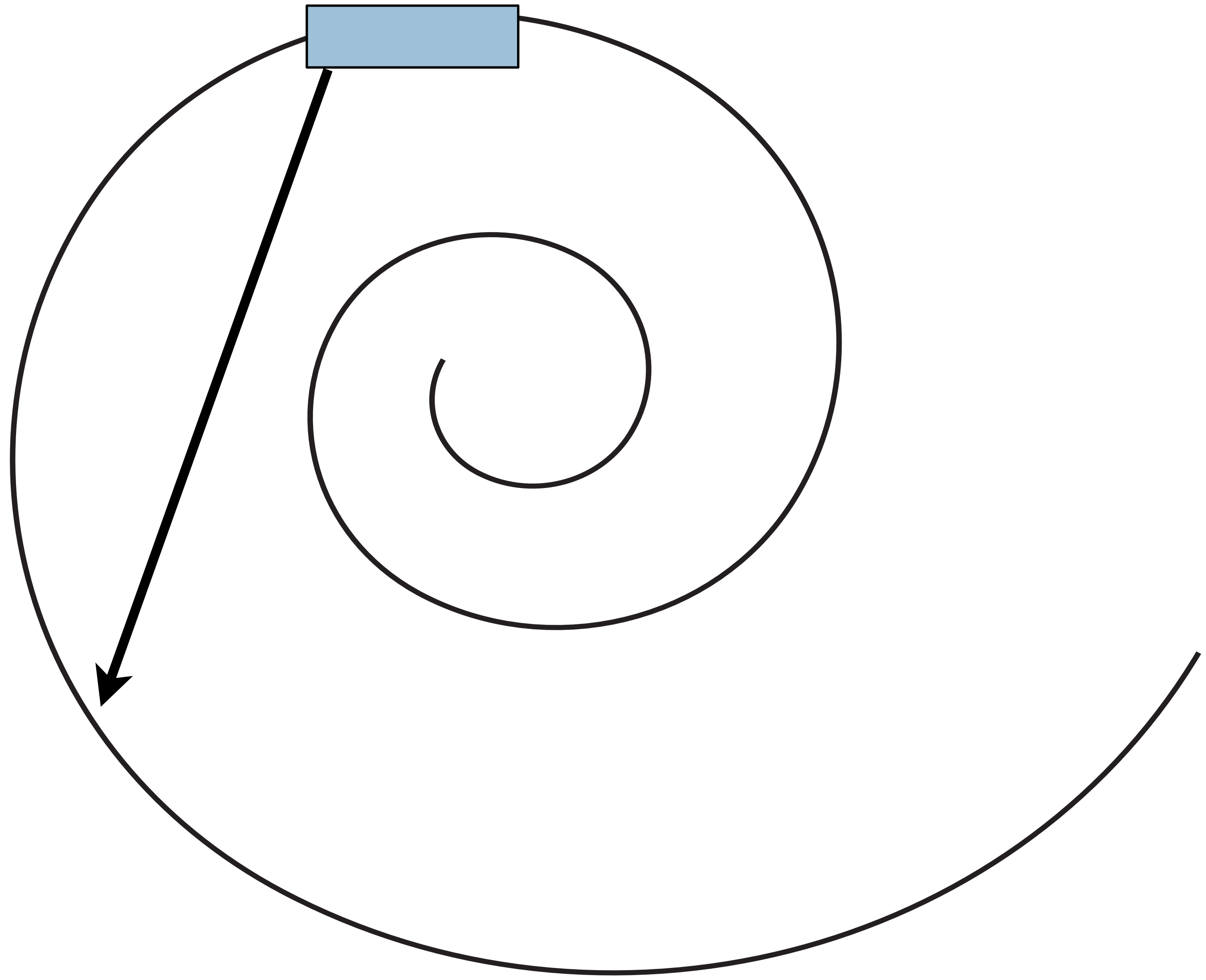
Imaging is done at maximum disk transfer rate:



But ingesting is done directory-by-directory.

Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory*.

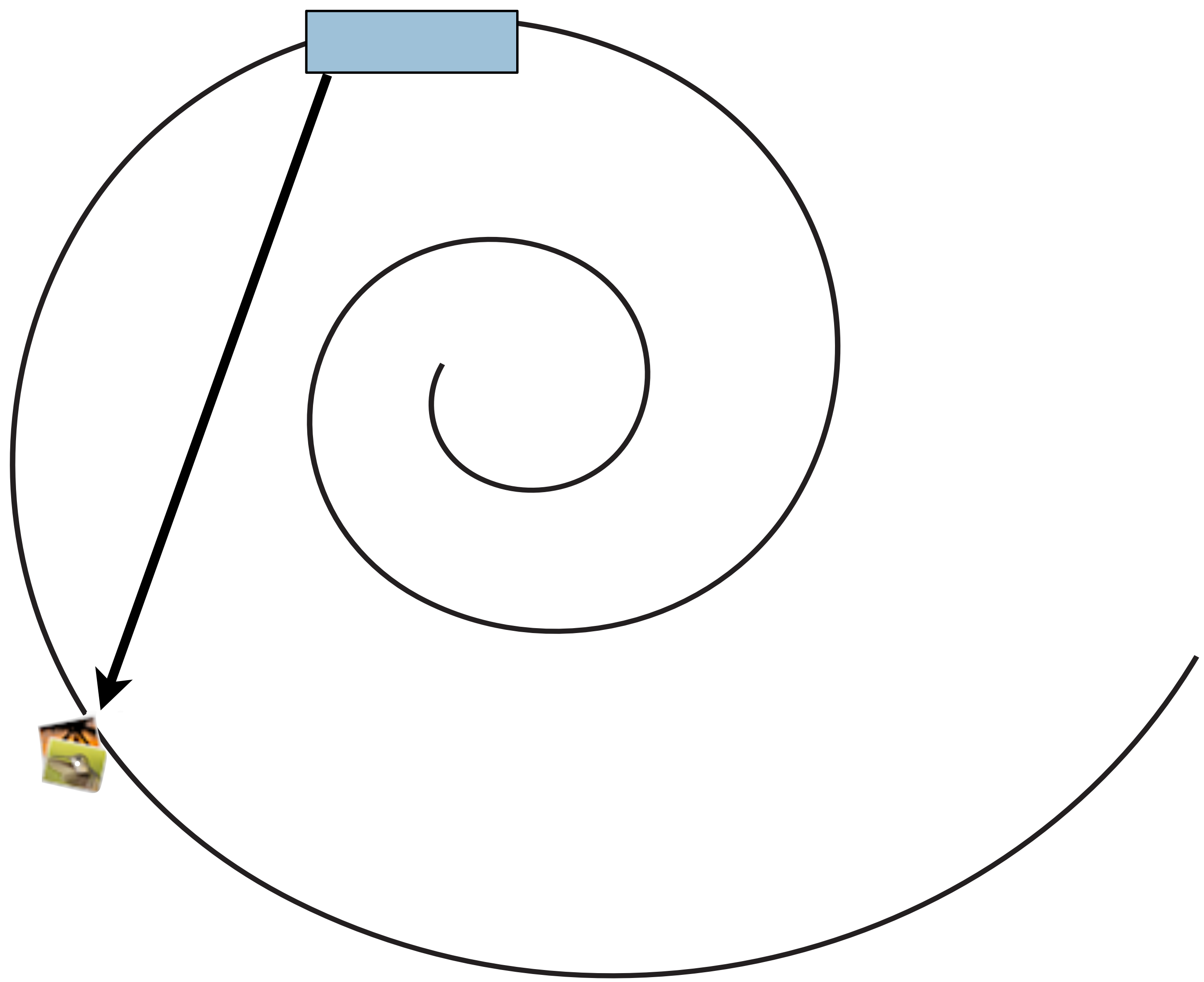
Imaging is done at maximum disk transfer rate:



But ingesting is done directory-by-directory.

Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory.*

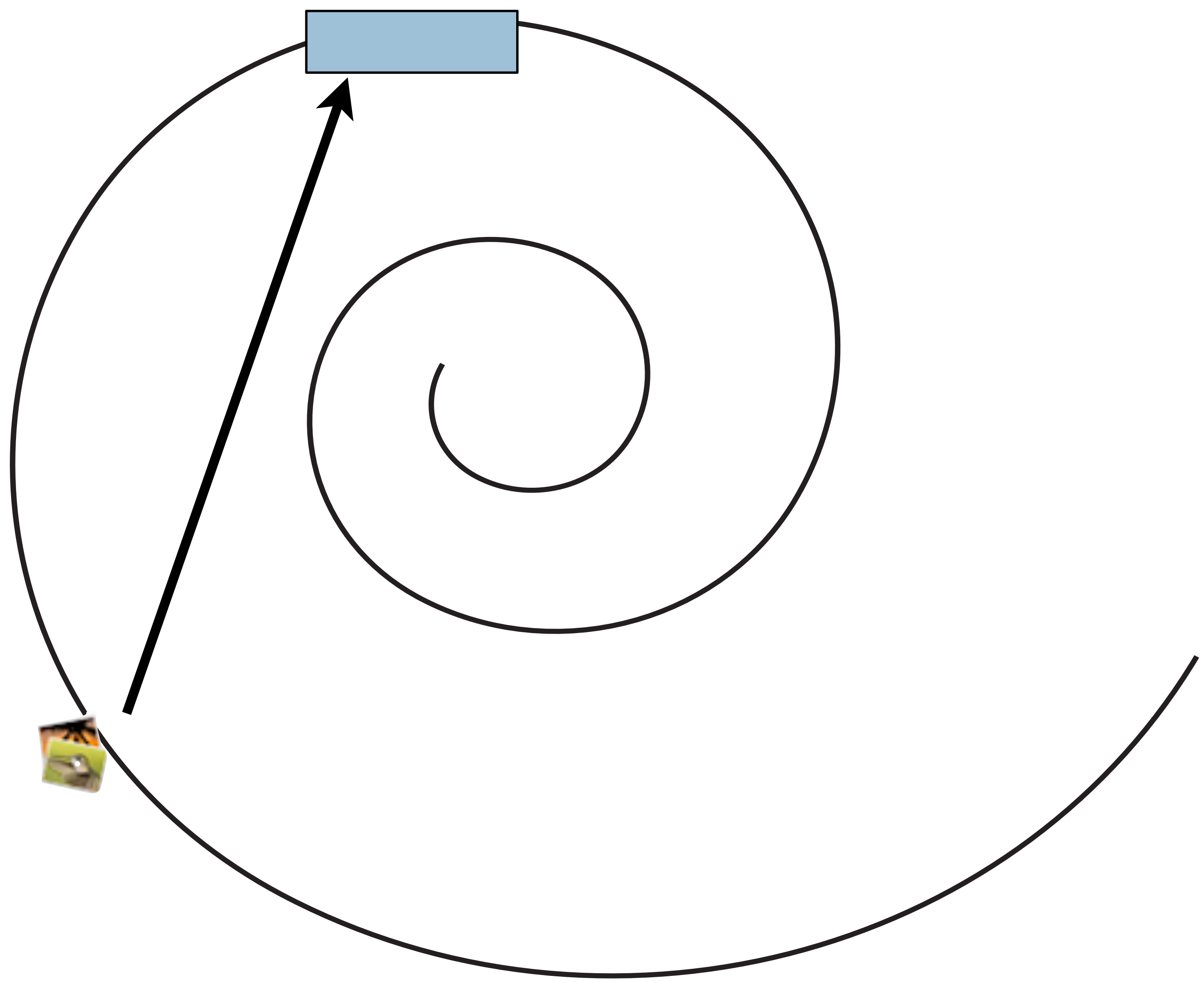
Imaging is done at maximum disk transfer rate:



But ingesting is done directory-by-directory.

Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory.*

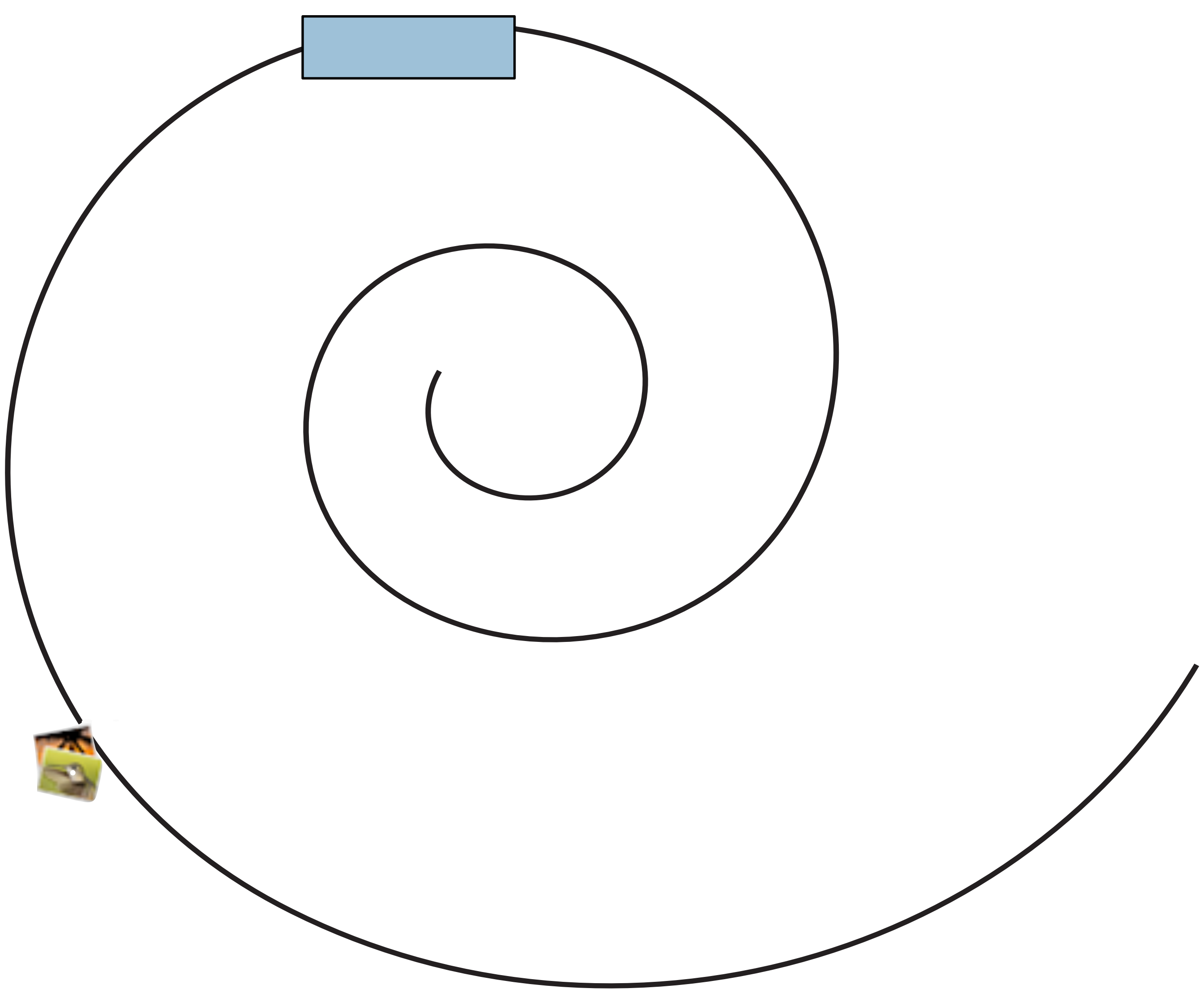
Imaging is done at maximum disk transfer rate:



But ingesting is done directory-by-directory.

Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory.*

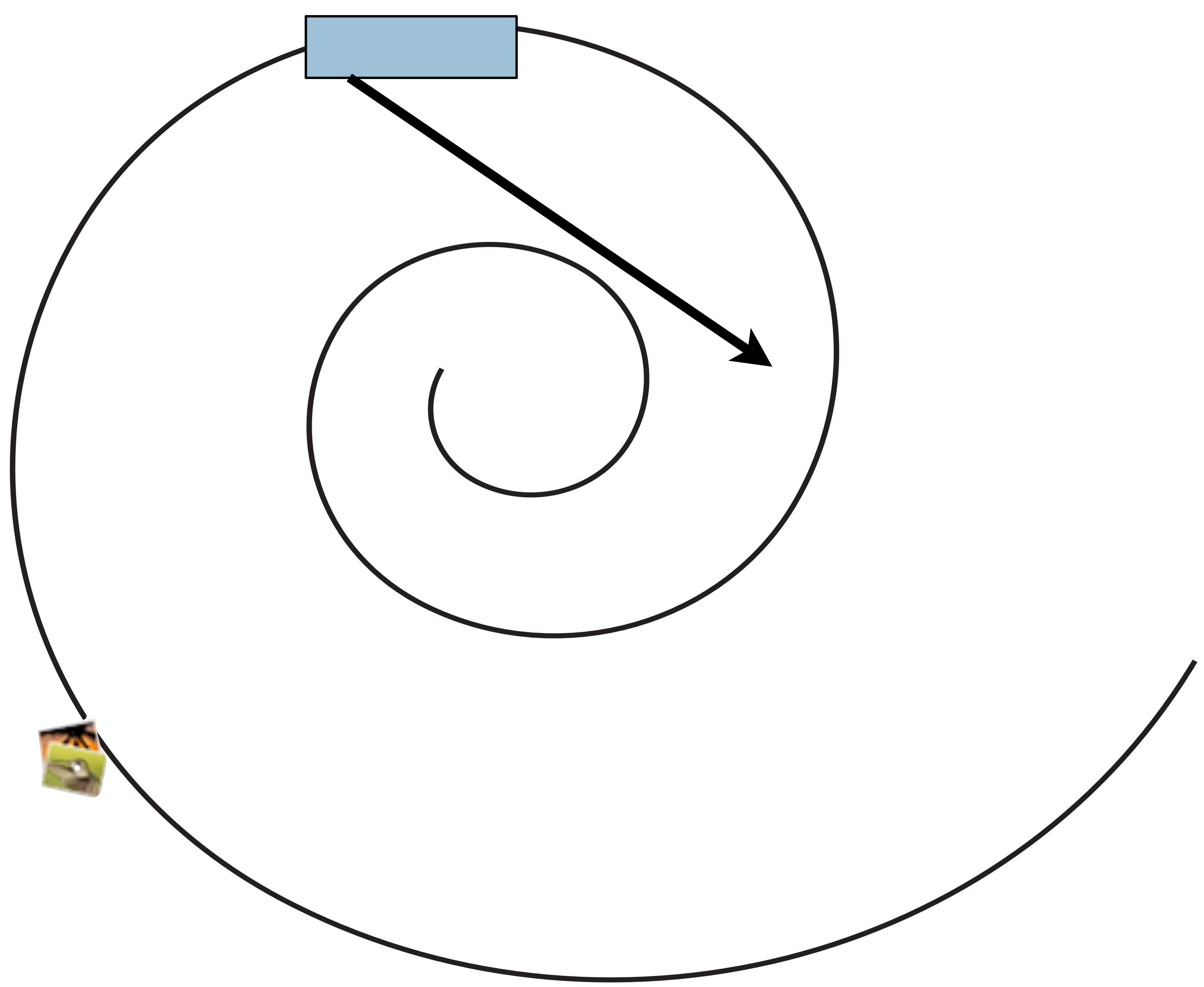
Imaging is done at maximum disk transfer rate:



But ingesting is done directory-by-directory.

Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory.*

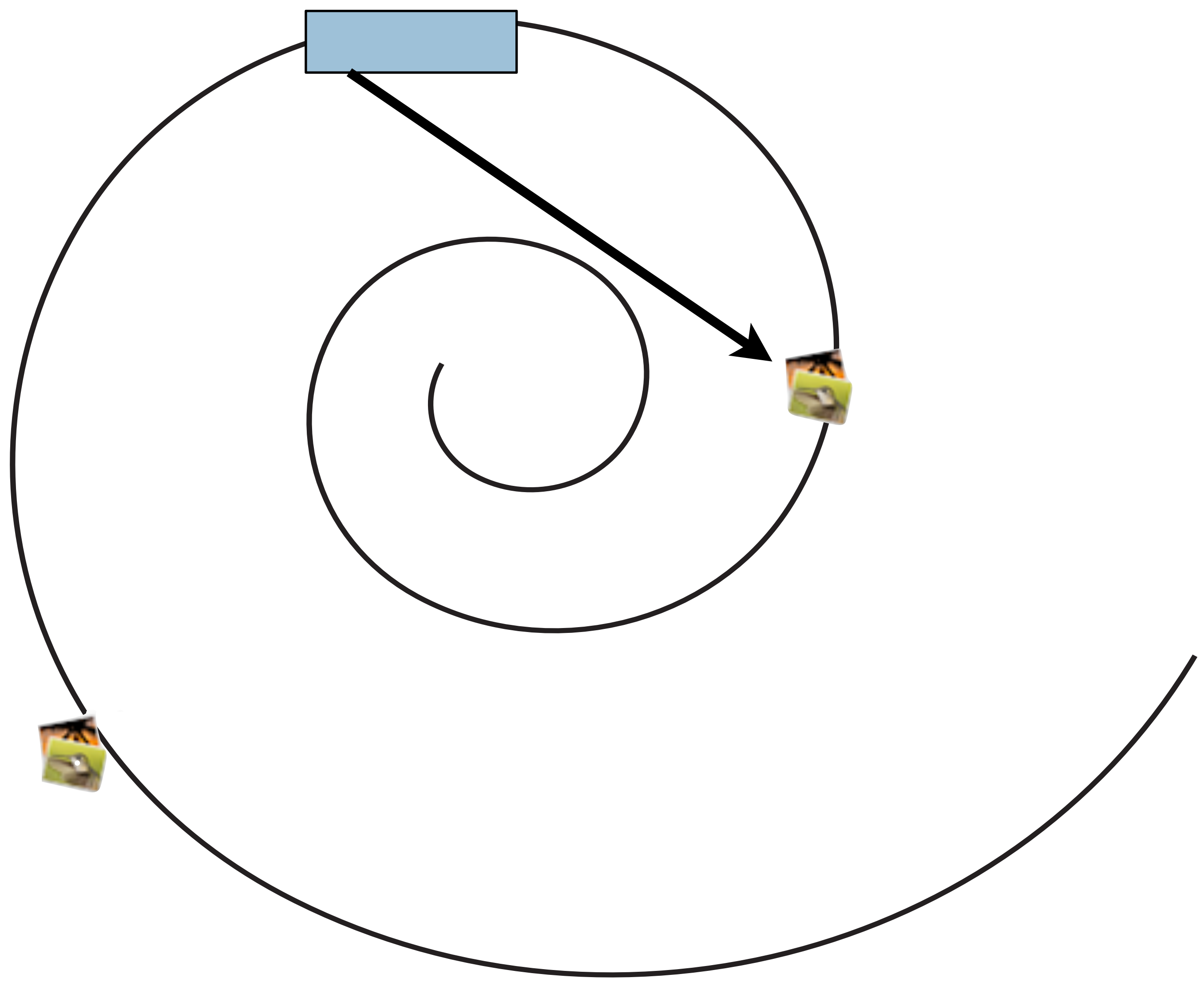
Imaging is done at maximum disk transfer rate:



But ingesting is done directory-by-directory.

Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory.*

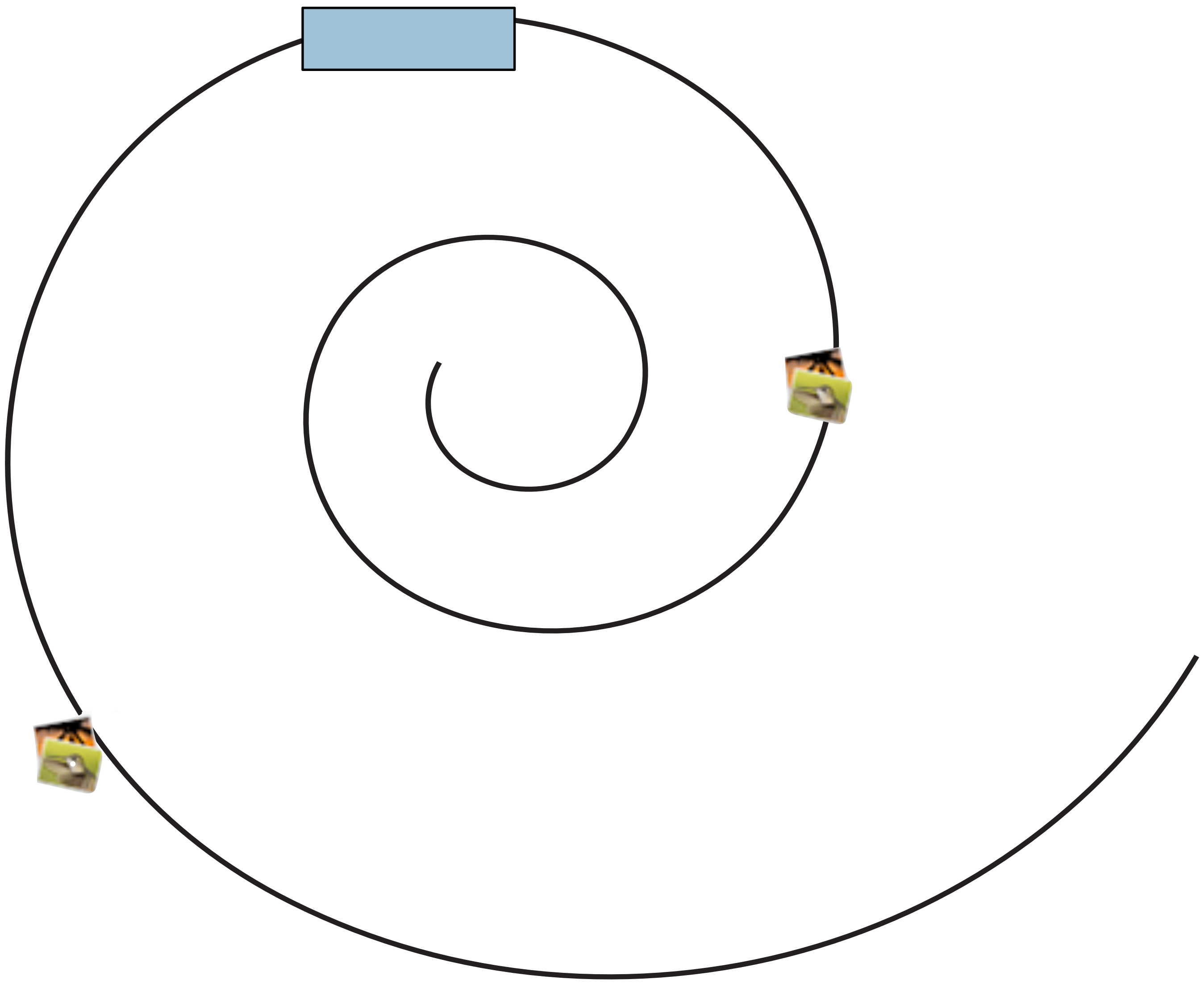
Imaging is done at maximum disk transfer rate:



But ingesting is done directory-by-directory.

Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory.*

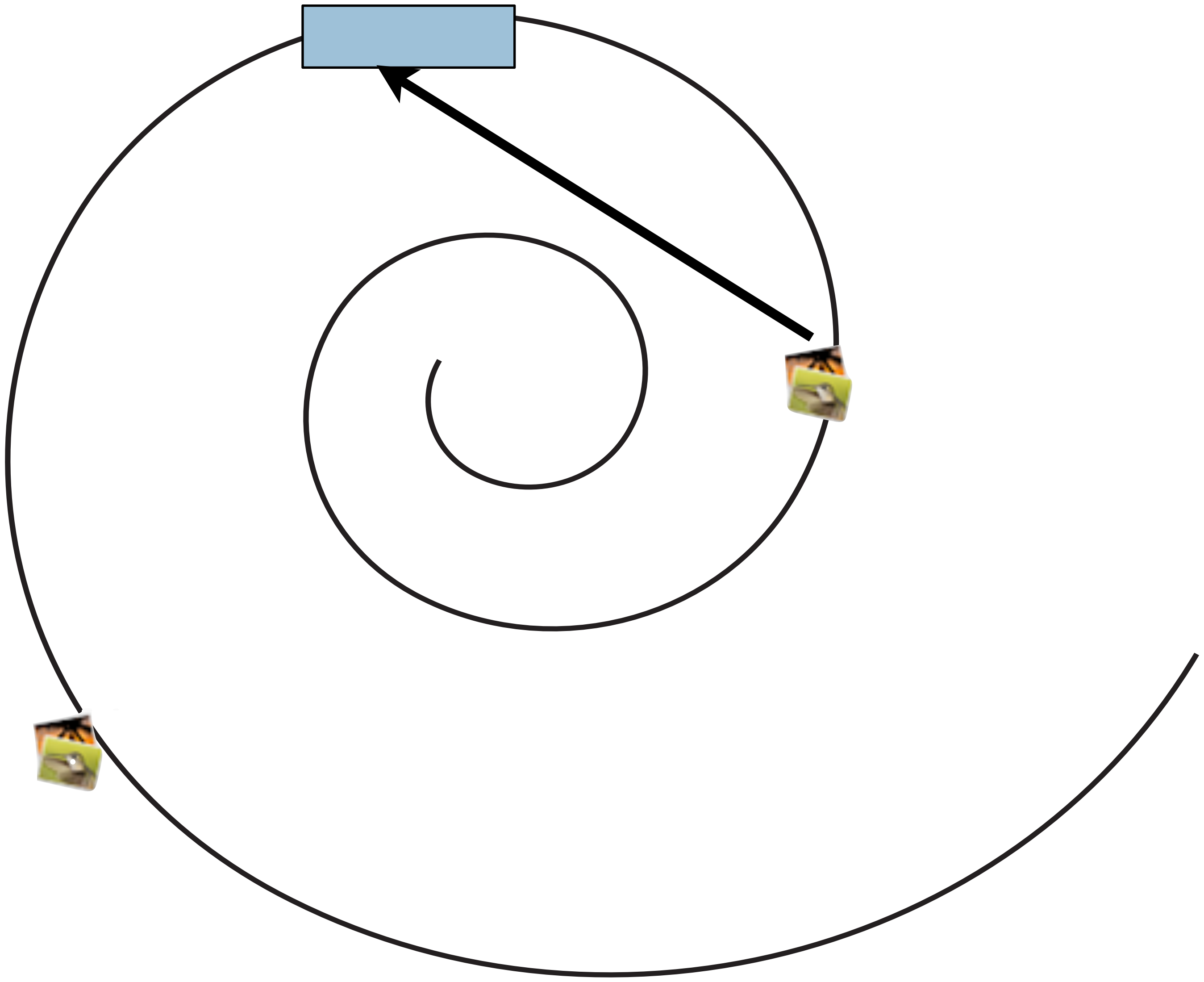
Imaging is done at maximum disk transfer rate:



But ingesting is done directory-by-directory.

Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory.*

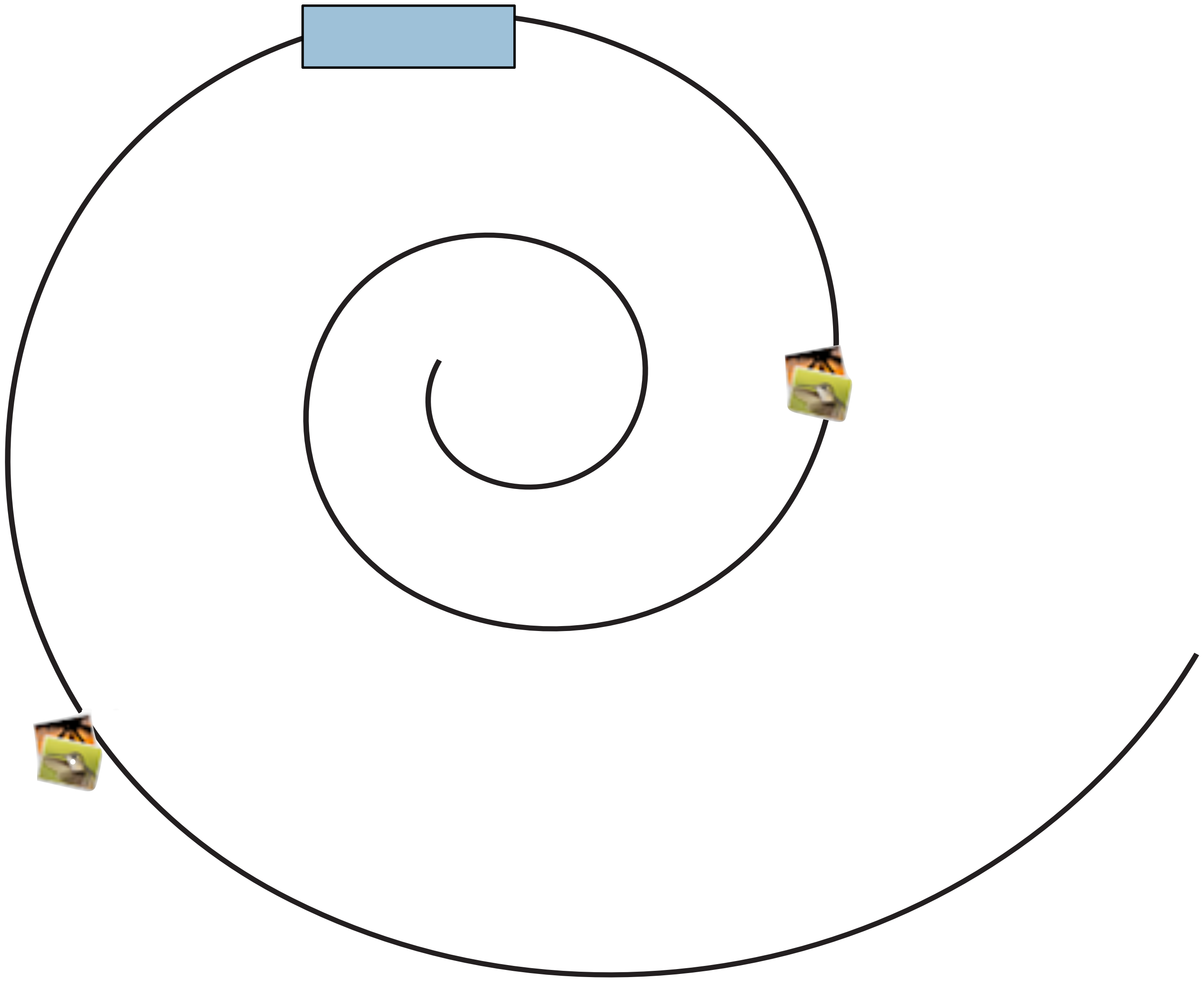
Imaging is done at maximum disk transfer rate:



But ingesting is done directory-by-directory.

Most forensic programs *image* the disk, then *ingest* the image *directory-by-directory.*

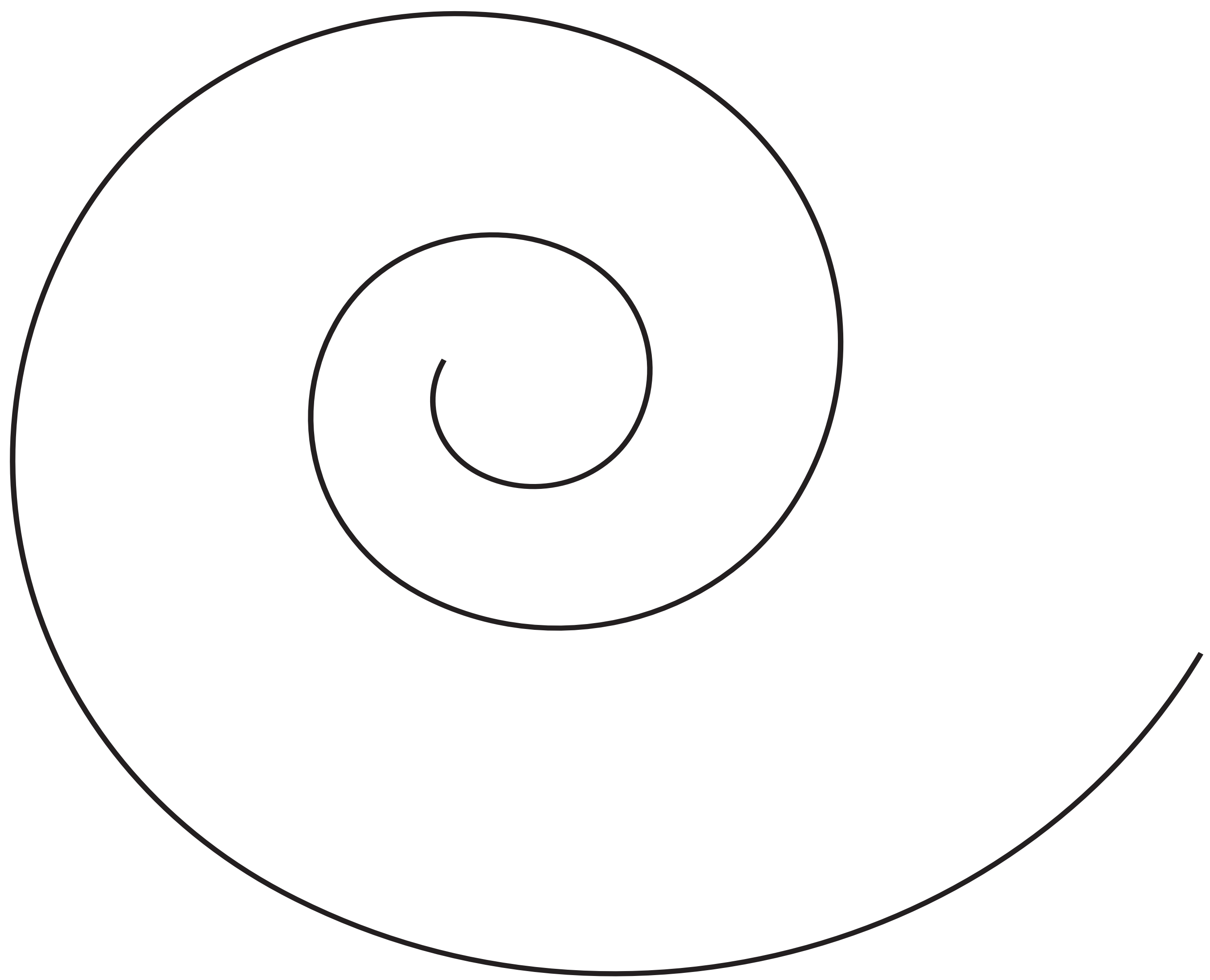
Imaging is done at maximum disk transfer rate:



But ingesting is done directory-by-directory.

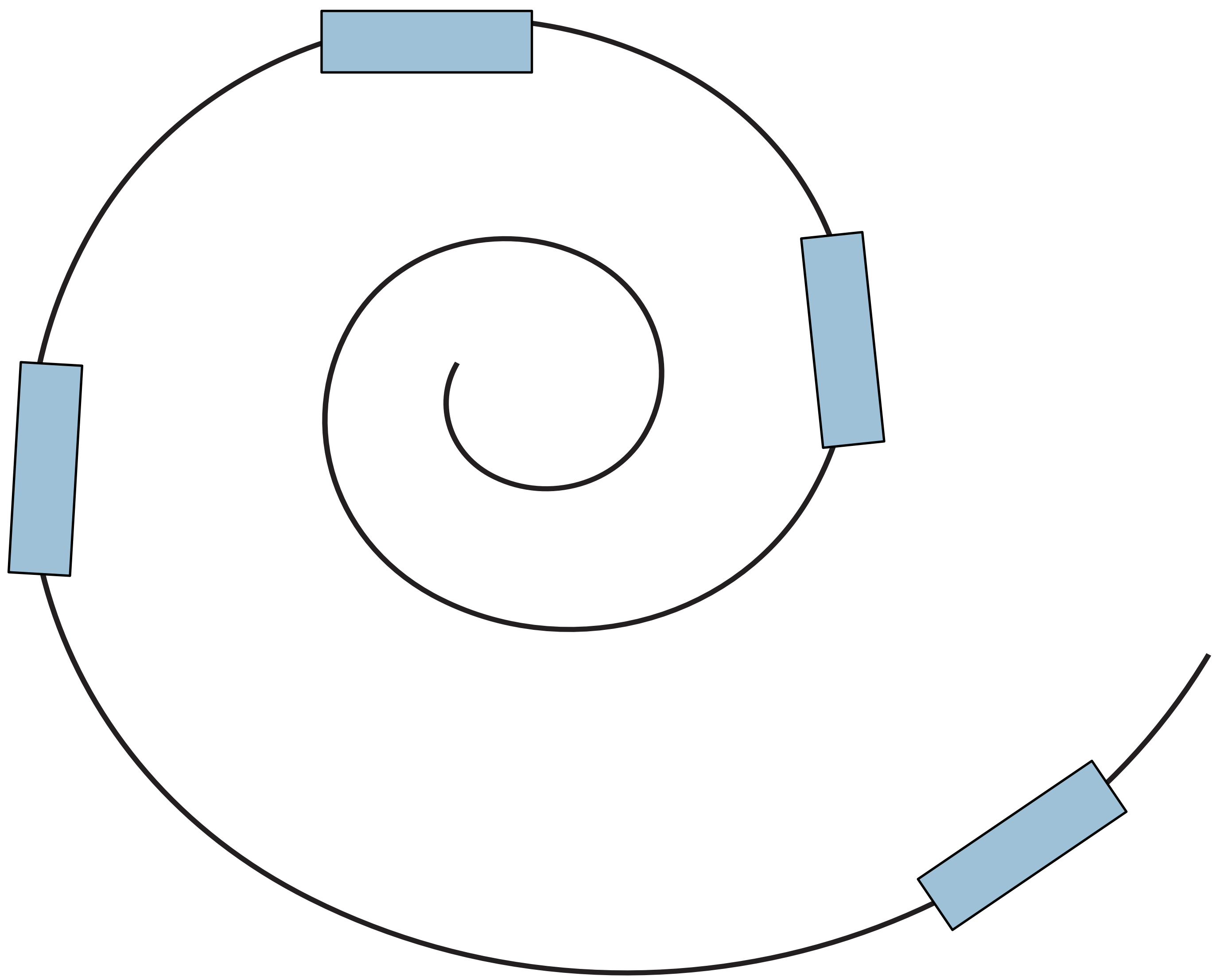
There is room for improvement!

1 - Find all of the directories



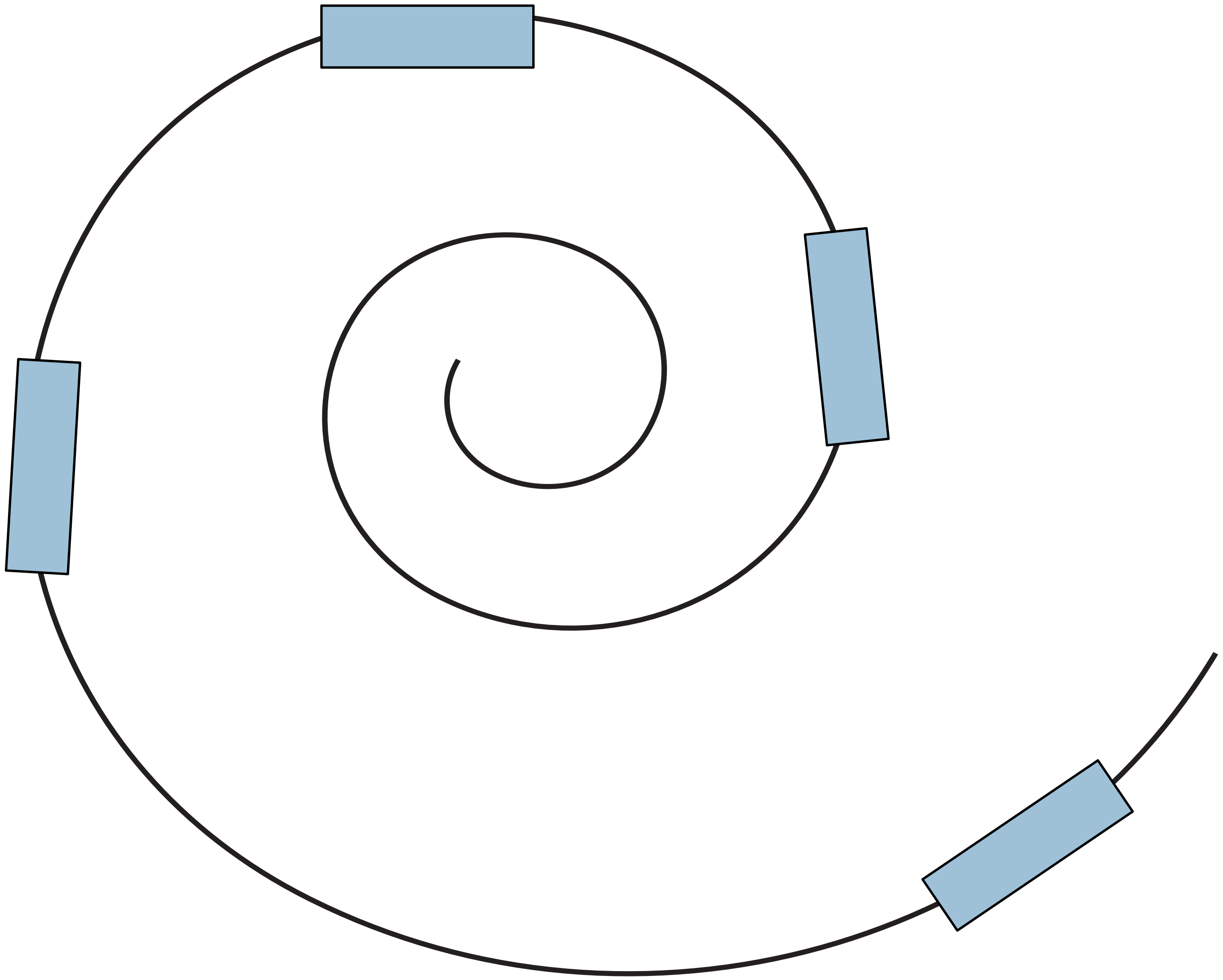
There is room for improvement!

1 - Find all of the directories



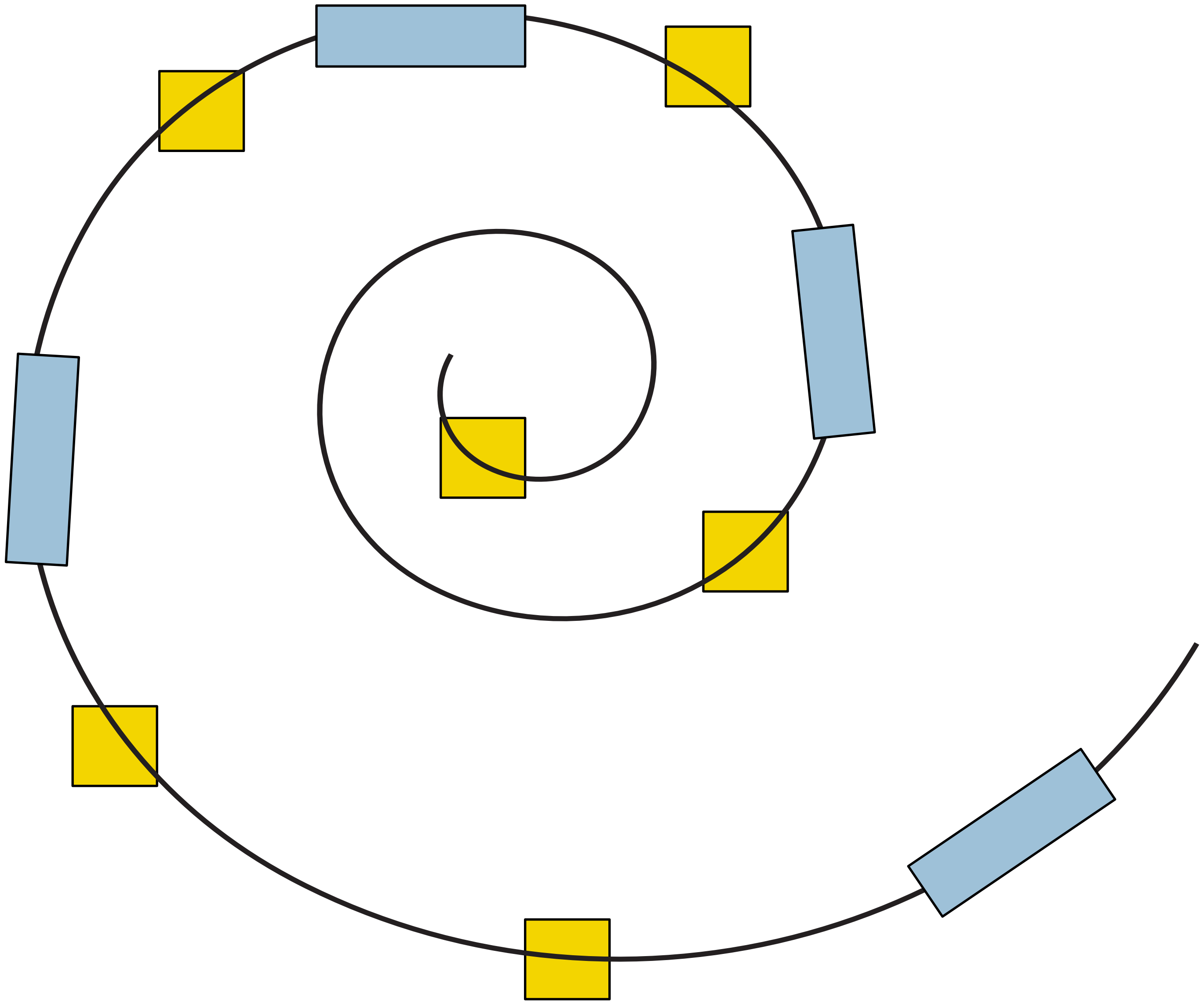
There is room for improvement!

1 - Find all of the directories
2 - Locate all the files



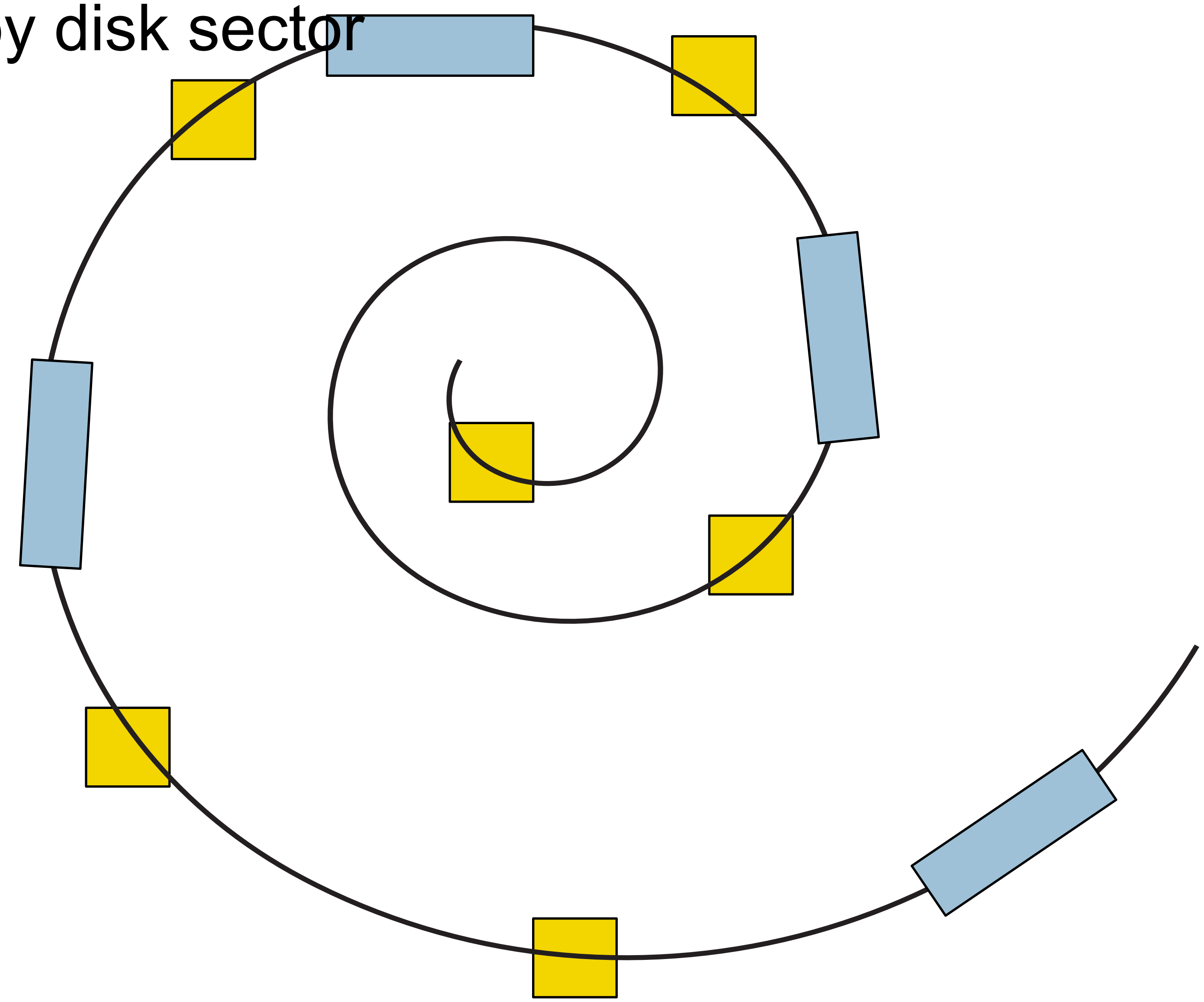
There is room for improvement!

1 - Find all of the directories
2 - Locate all the files



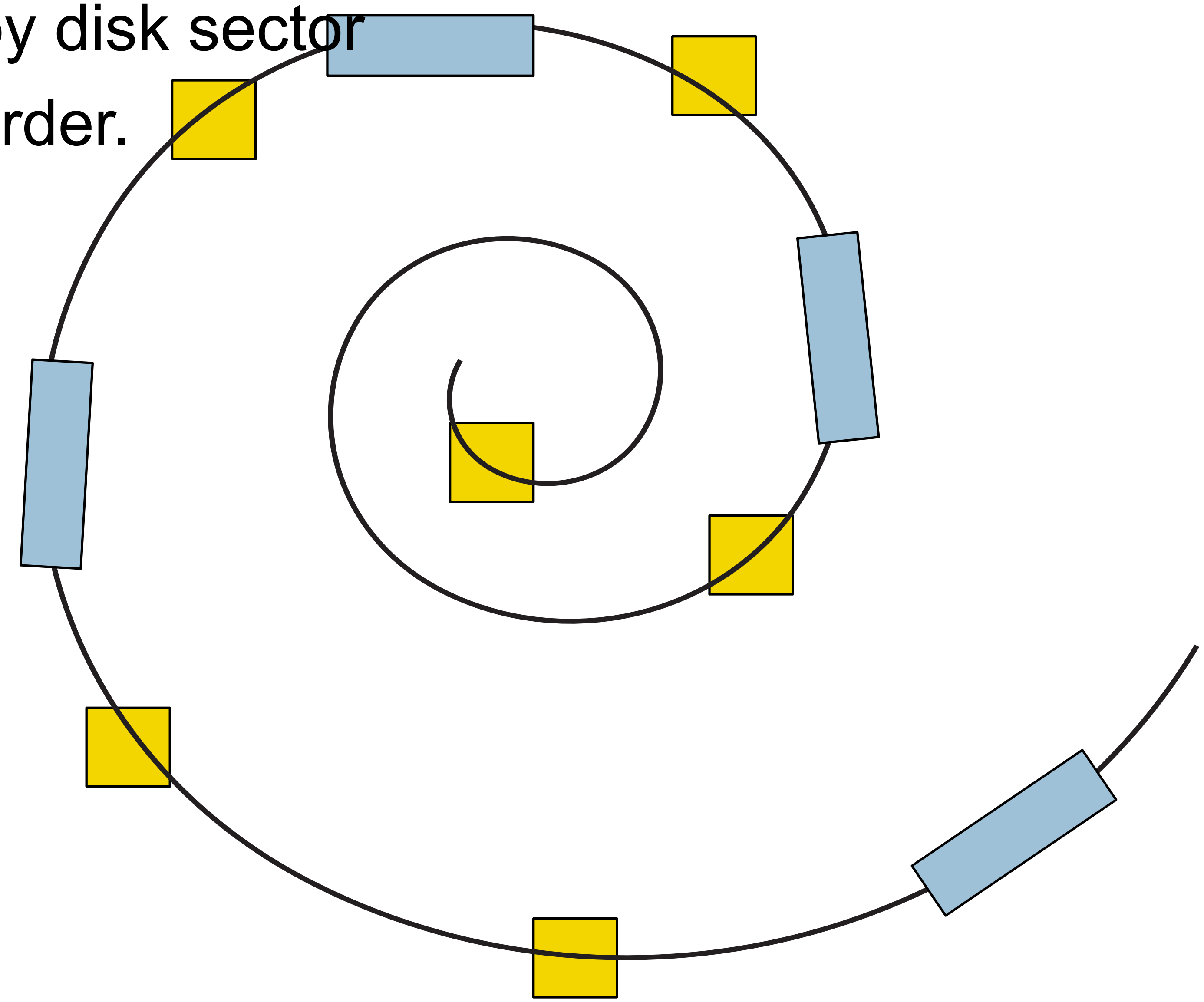
There is room for improvement!

1 - Find all of the directories
2 - Locate all the files
3 - Sort the files by disk sector



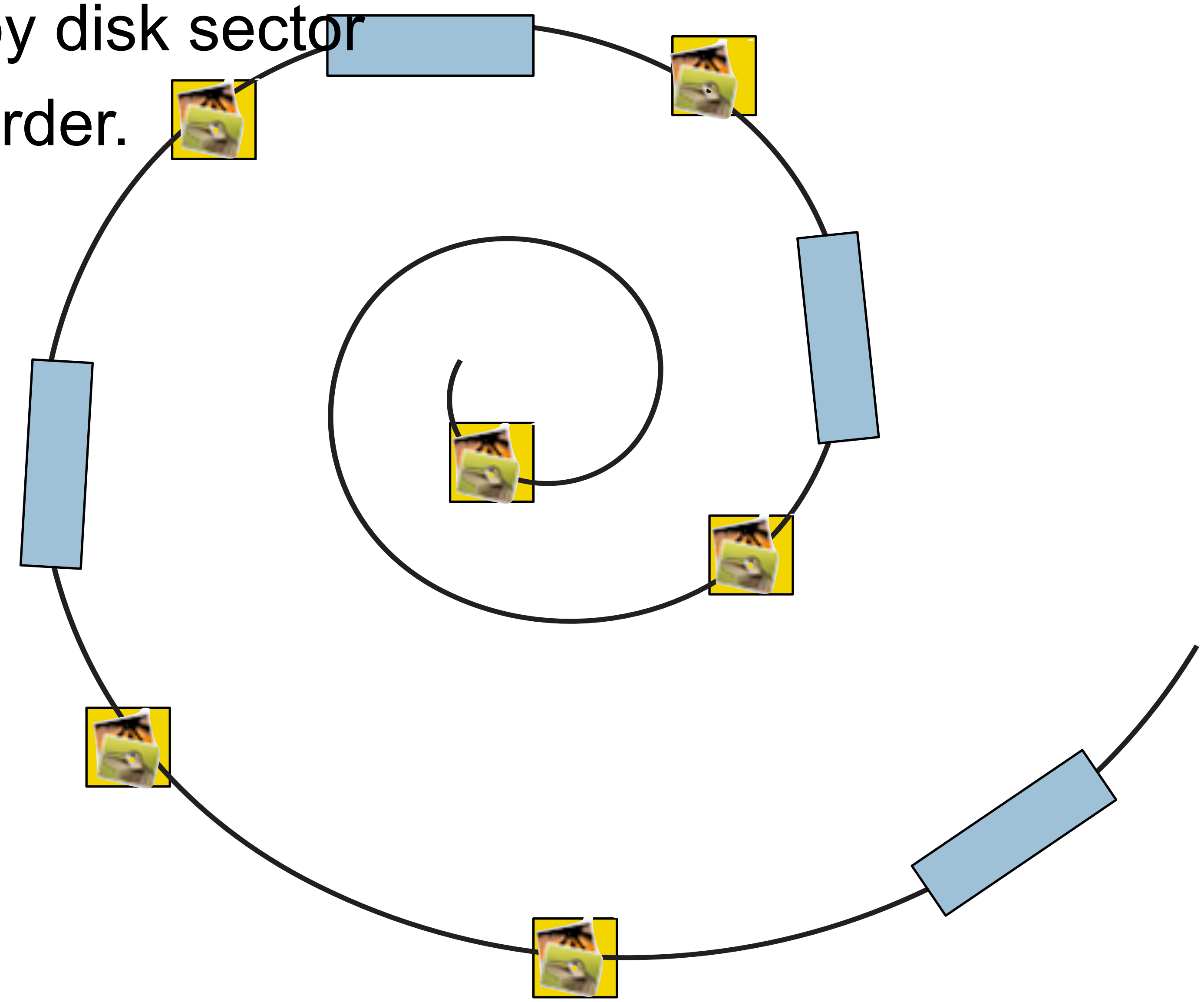
There is room for improvement!

1 - Find all of the directories
2 - Locate all the files
3 - Sort the files by disk sector
4 - Read in disk order.



There is room for improvement!

1 - Find all of the directories
2 - Locate all the files
3 - Sort the files by disk sector
4 - Read in disk order.



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)
```

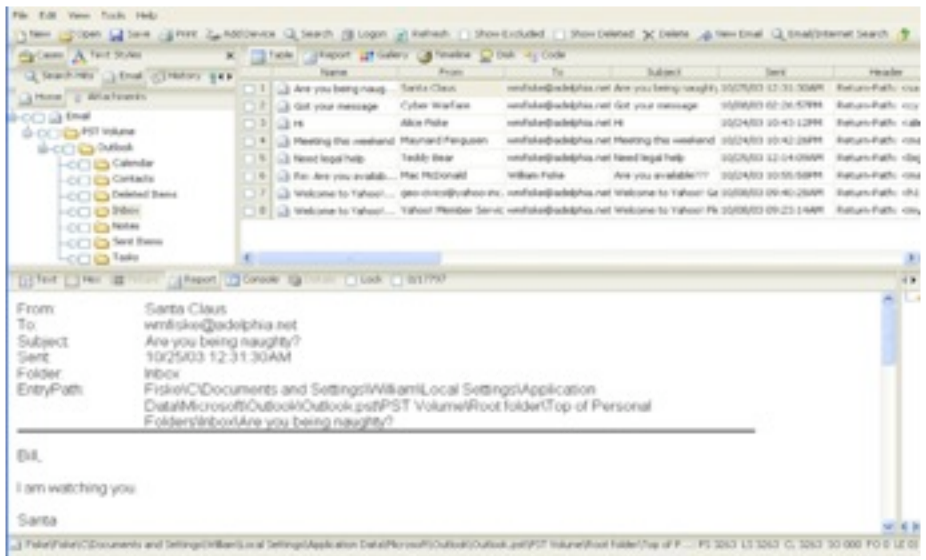
Advantages of processing by sector order:

- ⇨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:

Unfortunately, today's forensic tools are designed for performing forensic investigations.



Encase:
- GUI Closed Source

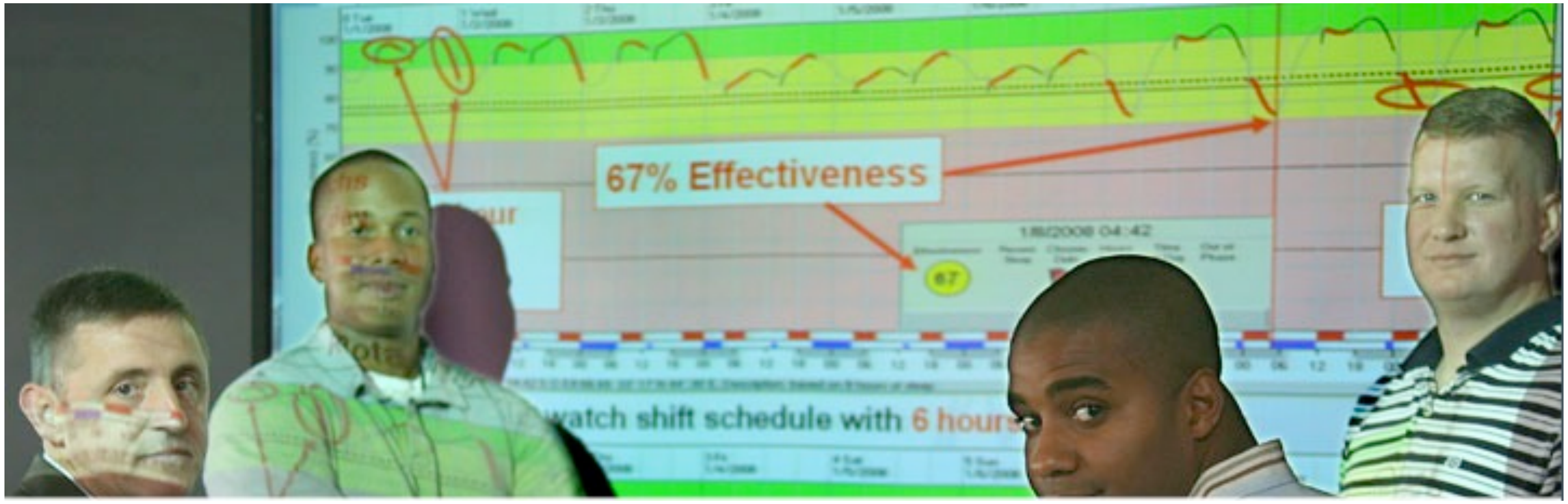
SleuthKit:
- Command-line Open Source

These tools are great for:

- ⇨FIle recovery
- ⇨Search

These tools were not created for research or automation.

Students (and researchers) need an easy-to-program environment for conducting forensic experiments.



It's *hard* to work with forensic data — All the details matter

- ⇨Many different file systems.
- ⇨Many different file types.

Good research requires working with large data sets.

- ⇨Even small "pilot studies" should be tested on multiple data sources.
- ⇨Otherwise, you aren't doing research on forensics — you are researching a particular object.

But 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++

But *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.

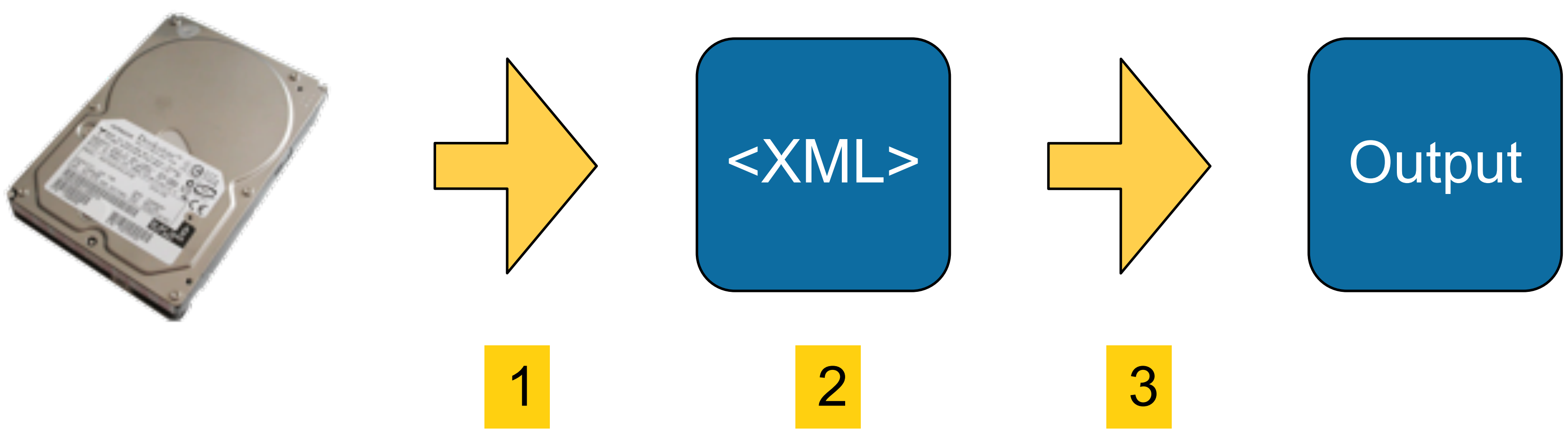
Faced with this, a standard approach is to leverage the database:

- ⇨Extract everything into an SQL database.
- ⇨Use multiple SELECT statements to generate reports.

We have developed a new approach for automated forensic analysis and research

The approach breaks forensic processing into three key parts:

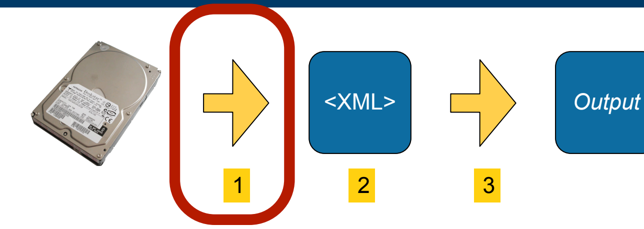
1. Extraction of forensic metadata.
2. Representation of the extracted metadata.
3. Processing.



You can start using this framework today.

You can easily expand it.

The framework is based on **fiwalk**, a too that extracts metadata from disk images.

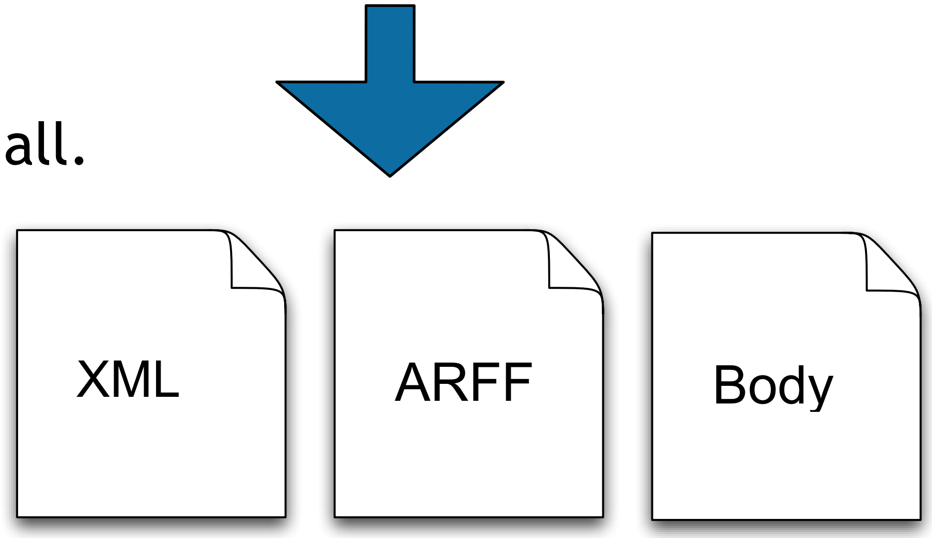


fiwalk is a C++ program built on top of SleuthKit

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

Features:

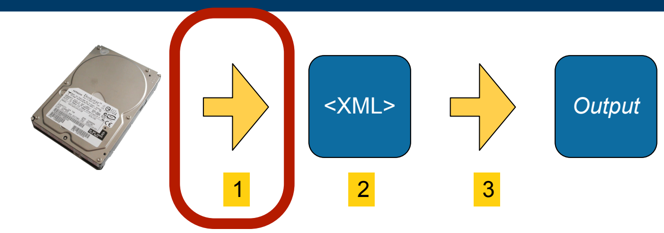
- ➩Finds all partitions & automatically processes each.
- ➩Handles file systems on raw device (partition-less).
- ➩Creates a *single output file* with forensic data data from all.



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
```

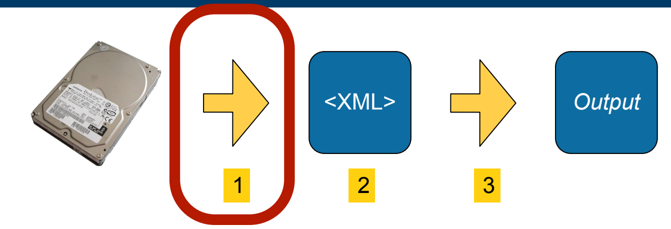
File System Metdata:

- ➪-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*

XML is ideally suited for representing forensic data.



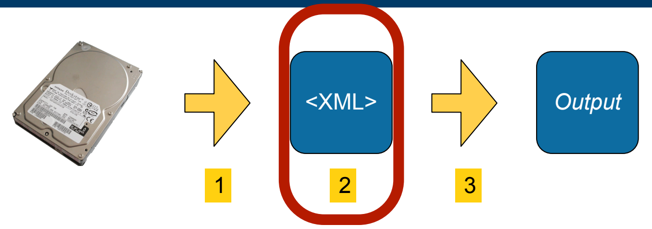
Forensic data is tree-structured.

- Case > Devices > Partitions > Directories > Files

- Files
  - *file system metadata*
  - *file meta data*
  - *file content*

- Container Files (ZIP, tar, CAB)
  - *We can exactly represent the container structure*
  - *PyFlag does this with "virtual files"*
  - *No easy way to do this with the current TSK/EnCase/FTK structure*
  - *(Note: Container files not currently implemented.)*

fiwalk produces three kinds of XML tags.



Per-Image tags

```
<fiwalk> – outer tag
<fiwalk_version>0.4</fiwalk_version>
<Start_time>Mon Oct 13 19:12:09 2008</Start_time>
<Imagefile>dosfs.dmg</Imagefile>
<volume startsector="512">
```

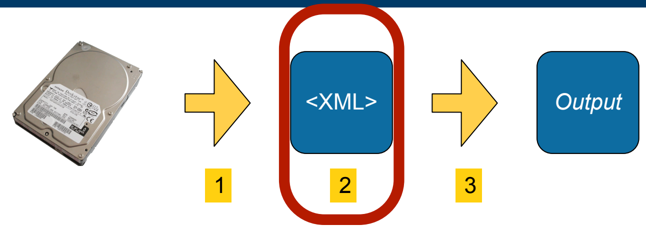
Per <volume> tags:

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

Per <fileobject> tags:

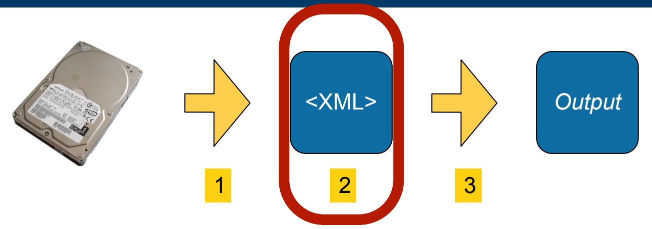
```
<filesize>4096</filesize>
<partition>1</partition>
<filename>linedash.gif</filename>
<libmagic>GIF image data, version 89a, 410 x 143</libmagic>
```

fiwalk XML example




```
<fileobject>
<filename>WINDOWS/system32/config/systemprofile/「开始」菜单/程序/附件/_rf55.tmp</
filename>
<filesize>1391</filesize>
<unalloc>1</unalloc>
<used>1</used>
<mtime>1150873922</mtime>
<ctime>1160927826</ctime>
<atime>1160884800</atime>
<fragments>0</fragments>
<md5>d41d8cd98f00b204e9800998ecf8427e</md5>
<sha1>da39a3ee5e6b4b0d3255bfef95601890afd80709</sha1>
<partition>1</partition>
<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>
</fileobject>
```

<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>
```

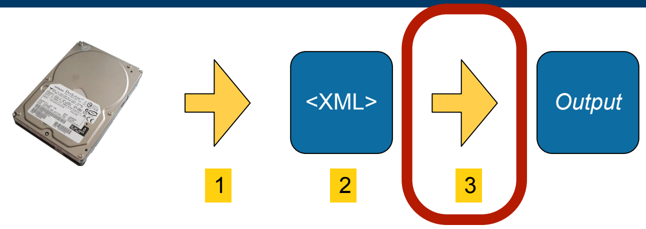
This file has two fragments:

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

Additional XML attributes may specify compression or encryption.

- ⇨Note: Currently <byte_runs> not provided for compressed or MFT-resident files.

fiwalk.py: a Python module for automated forensics.



Key Features:

- ⇨Automatically runs fiwalk with correct options if given a disk image
- ⇨Reads XML file if present (faster than regenerating)
- ⇨Creates **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)
```

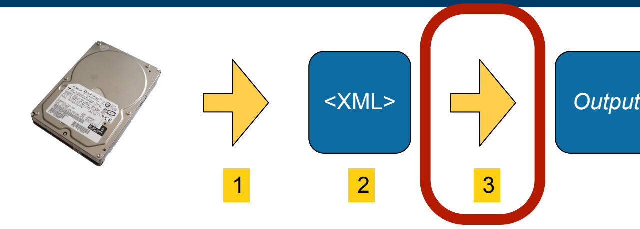
  – *Reasonably fast; returns a list of all file objects with XML in dictionary*

- ⇨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()**

Using the framework, we 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 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

Instant Drive Forensics with Statistical Sampling

Research Question:
Is it possible to analyze a hard drive in a minute?

What if US agents encounter a hard drive at a border crossing?

Or if a room filled with computers turns up on a search?

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

		
Minutes	208	1
Max Data	1.5 TB	7.2 GB
Max Seeks	15 million	72,000

7.2 GB is a lot of data!

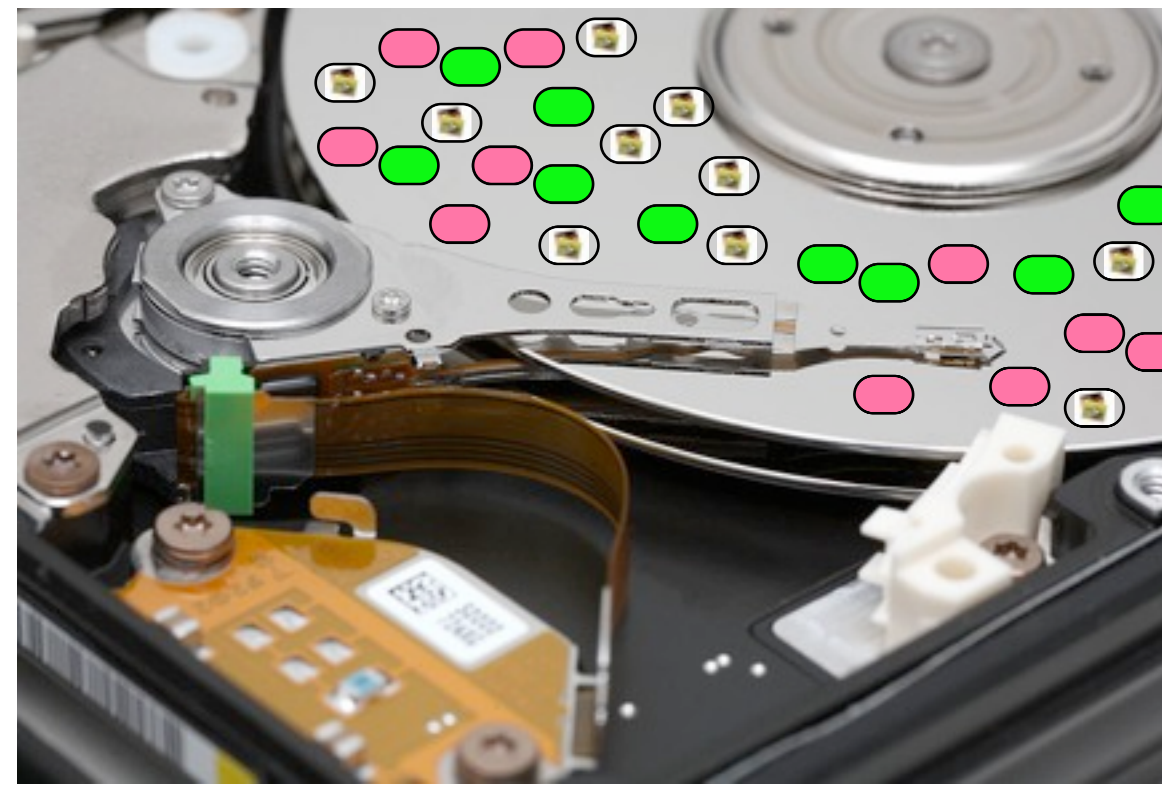
- ⇨≈ 0.48% of the disk
- ⇨But it can be a statistically significant sample.

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

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

The challenge is making sure that sample matches *likely voters.*

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



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

The accuracy of a random sample can be computed from *p* (accuracy) and *n* (sample size), if n< 5%

$$\text{Standard error} = \sqrt{\frac{p(1-p)}{n}}$$

Example:

- ⇨Sample 10,000 sectors and find 30% blank, standard error ≈ 4.5%
- ⇨Sample 100,000 sectors and find 15% are video, standard error ≈ 1.1%
- ⇨Sample 1,000,000 sectors and find 5% encrypted, standard error ≈ .02%

Caveats:

- ⇨The statistics are for the disk as a whole, not for the files.
- ⇨We *must* be able to identify the content of sectors.

Sectors on hard drives can be divided into three categories:

Resident Data

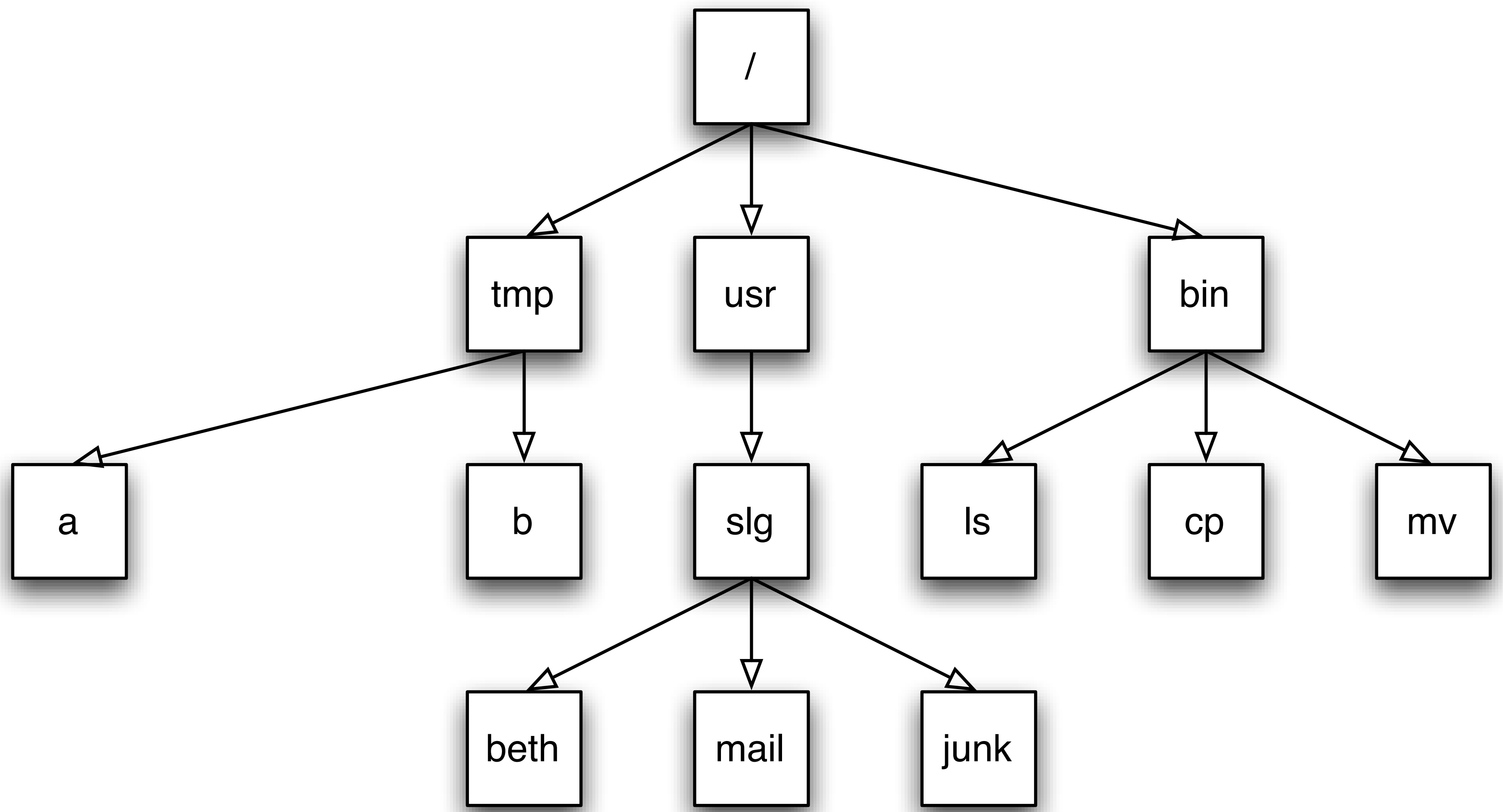
Deleted Data

user files
email messages
[temporary files]

Uninteresting Data

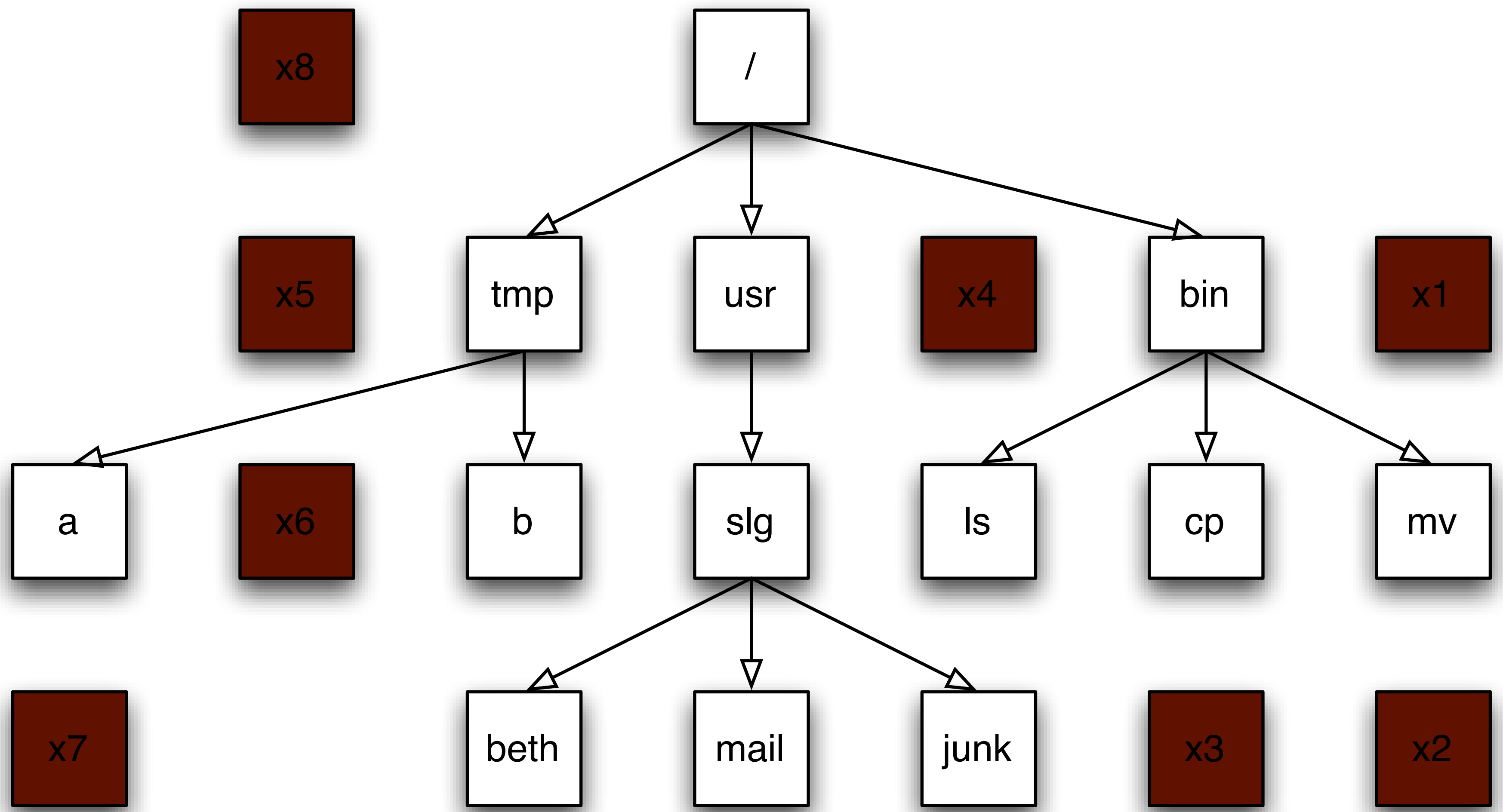
blank sectors [OS files]

Data on a hard drive is arranged in sectors



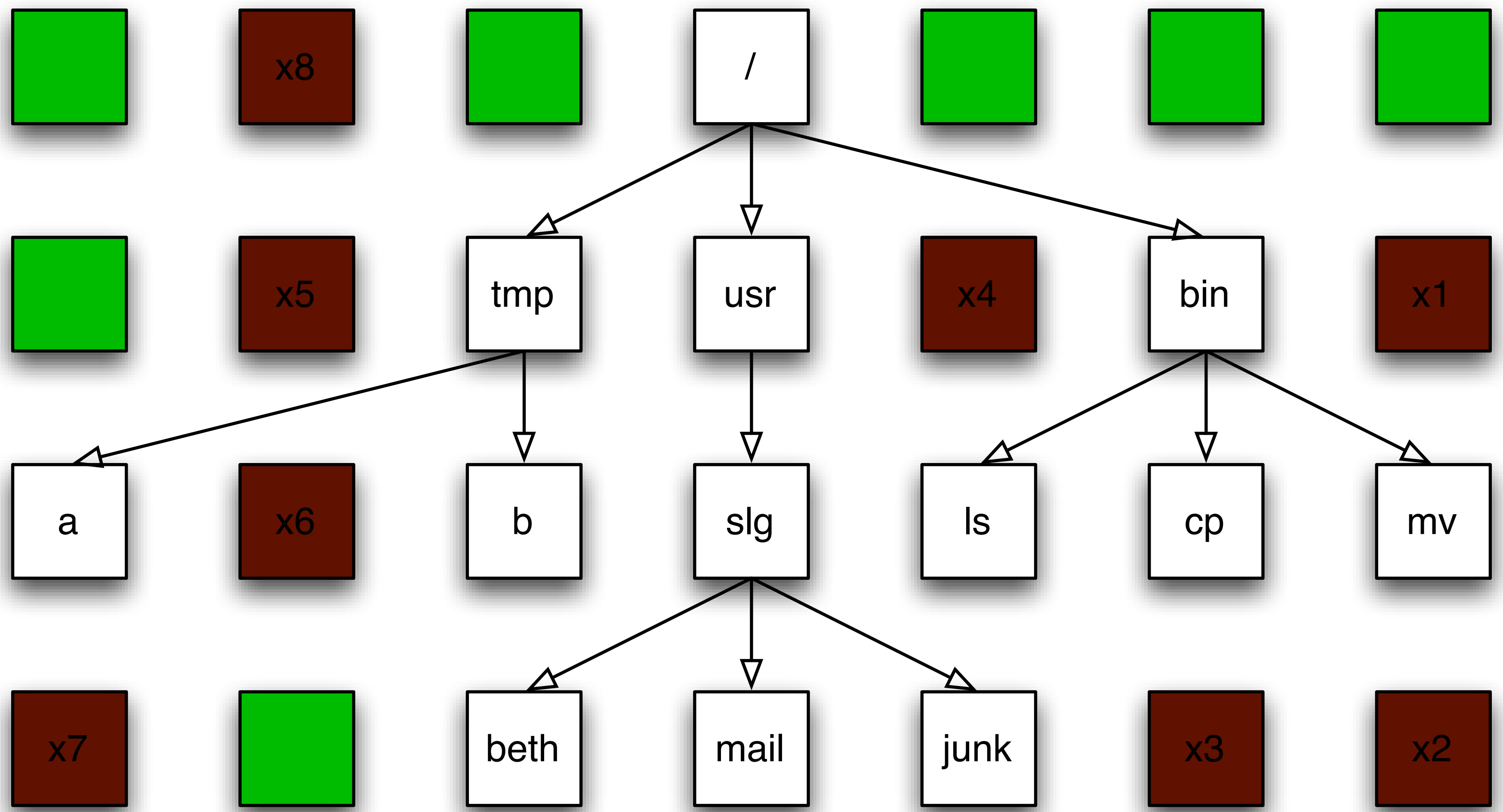
Resident Data = data visible to the user

Data on a hard drive is arranged in sectors



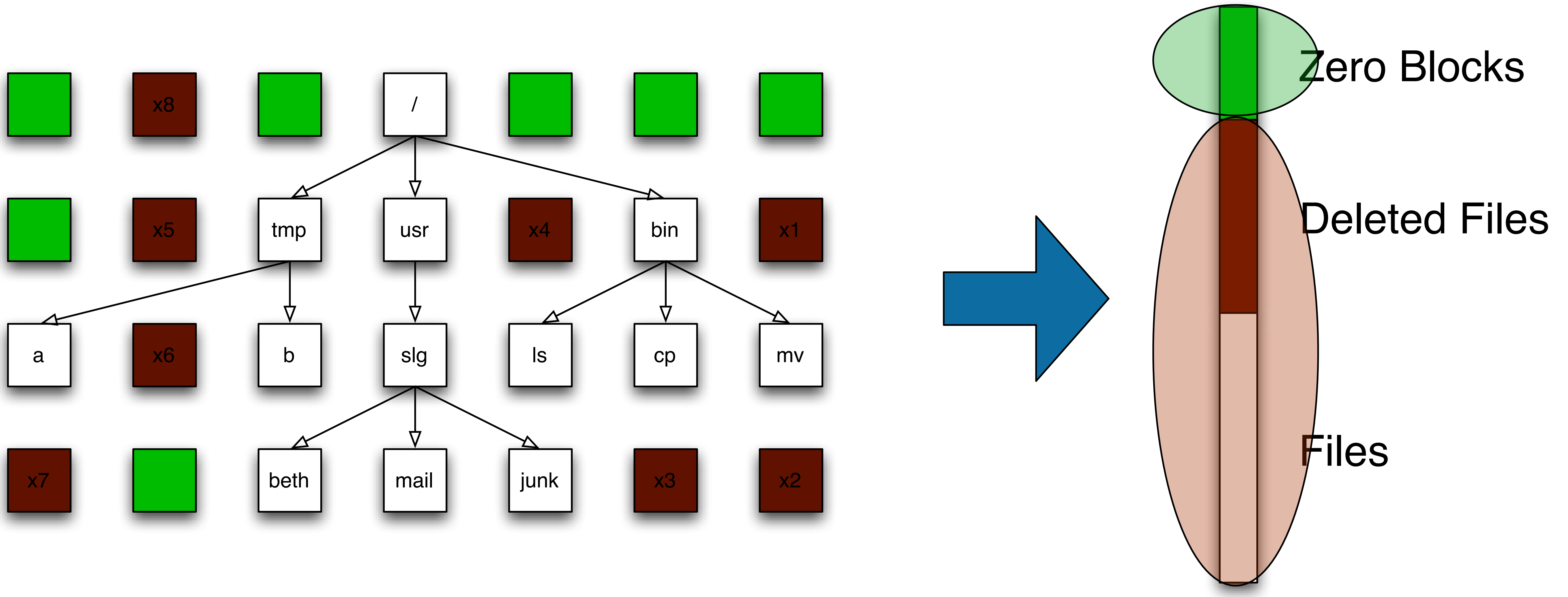
Deleted Data = files that were deleted.

Data on a hard drive is arranged in sectors



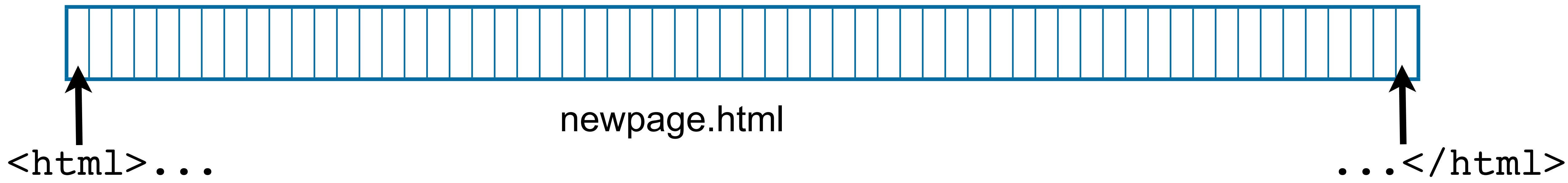
Uninteresting Data = never written (or wiped clean)

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



Can we classify files based on a sector?

A file 30K consists of 60 sectors:



Many file types have characteristics headers and footer:

	header	footer
HTML	`<html>`	`</html>`
JPEG	`<FF><D8><FF><E0>` `<00><10>JFIF<00>`	`<FF><D9>`
ZIP	`PK<03><0D>`	`<00><00><00><00>`

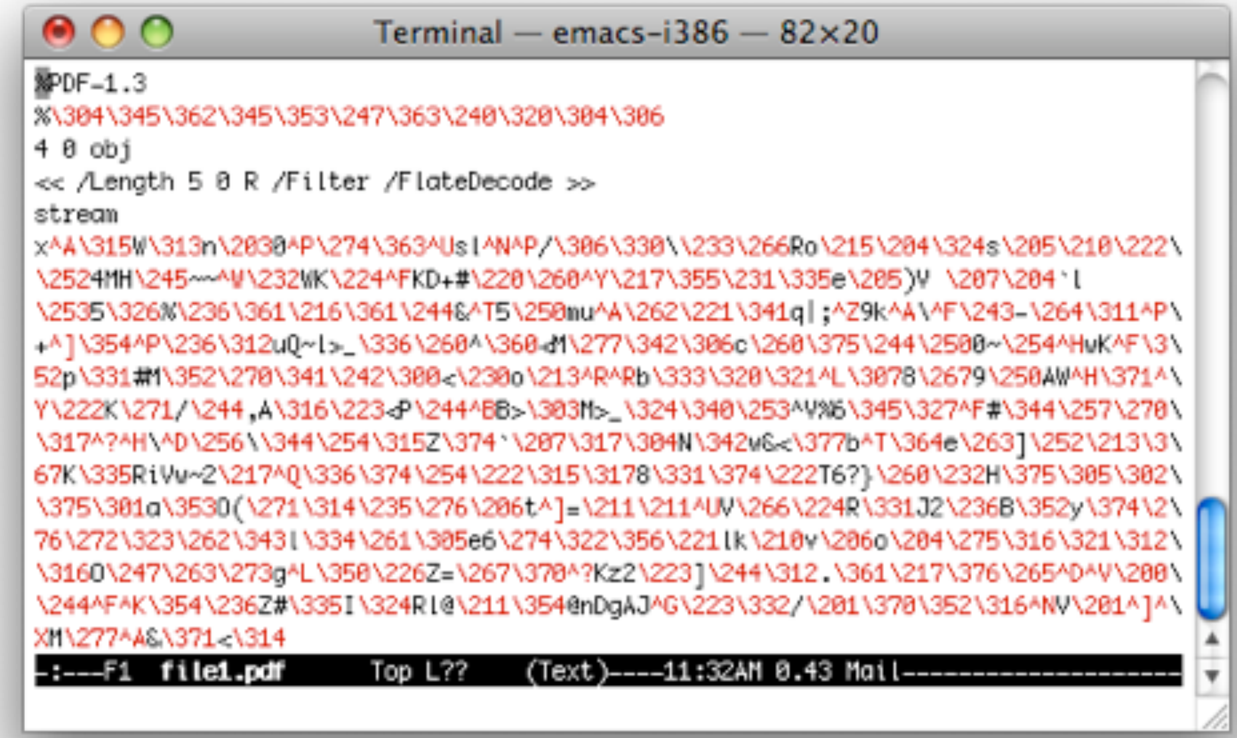
But what about the file in the middle?

Many files are "container" files.
Classifying sectors from these files will classify

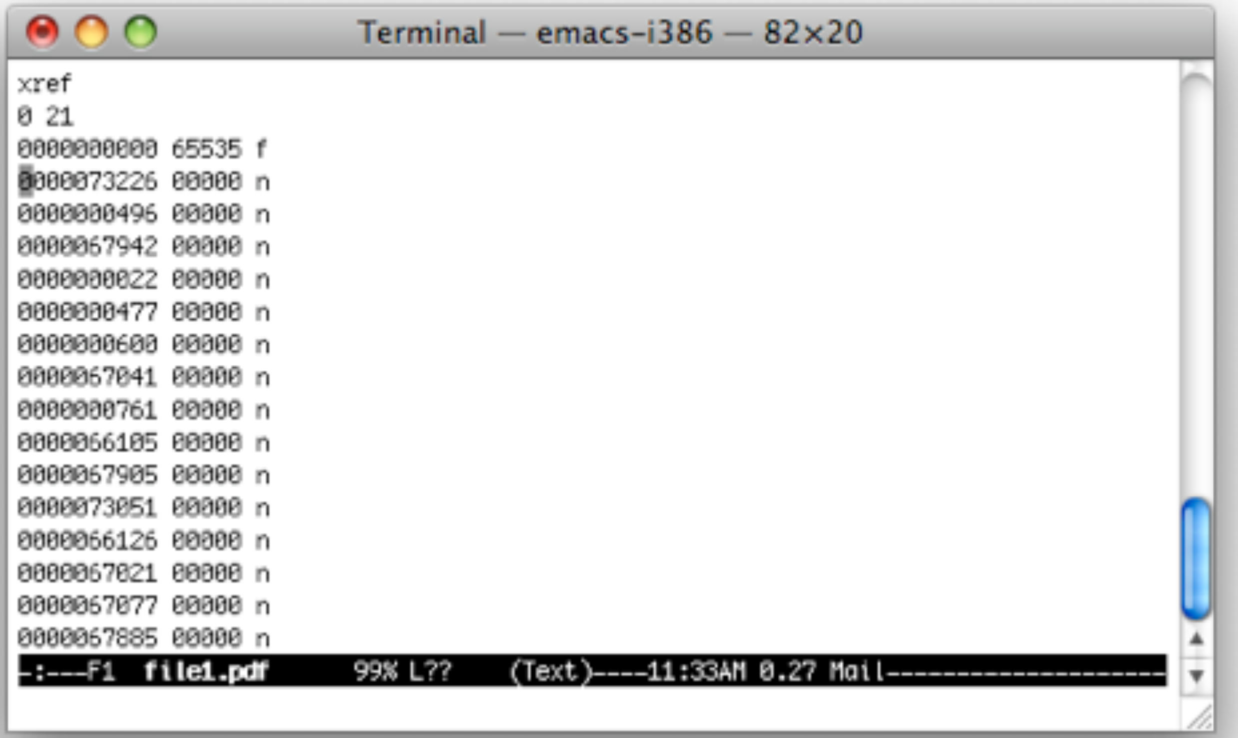
The PDF file format consists of:

- ⇨PDF header
- ⇨PDF xref table (a directory of objects in the PDF file)
- ⇨PDF objects (T/F; Numbers; Strings; Names; Arrays; Dicts; Streams; Null)

PDF header:

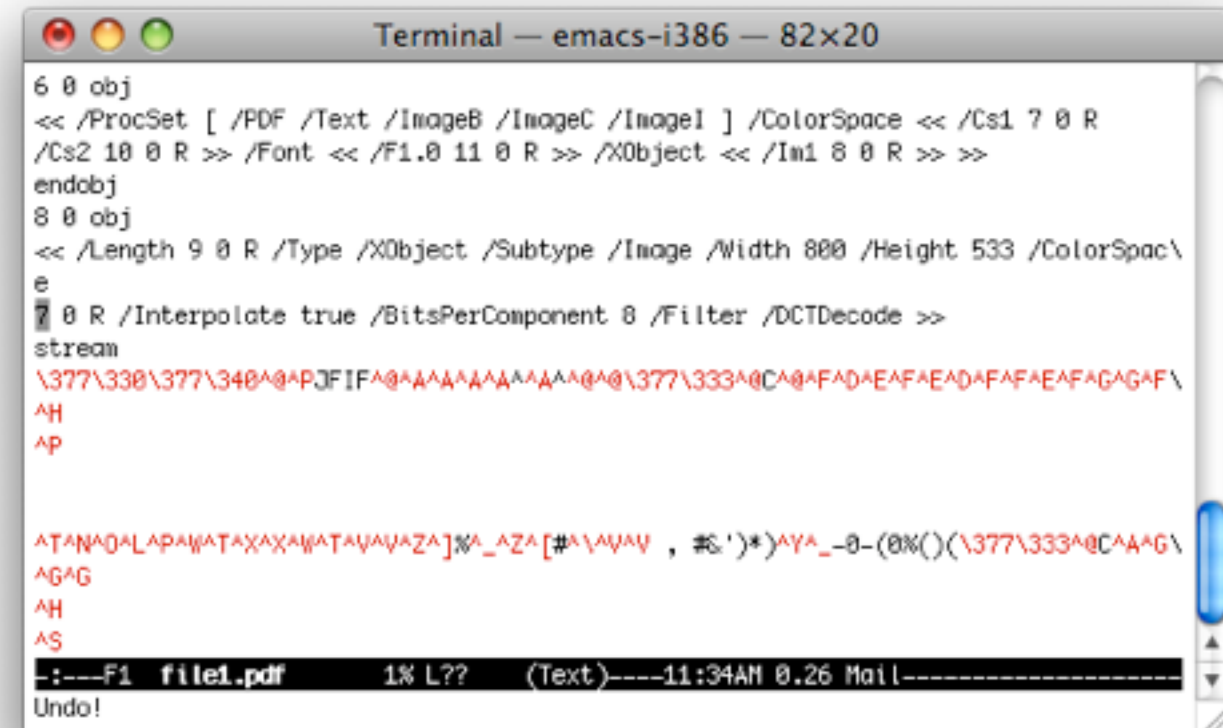


PDF xref:

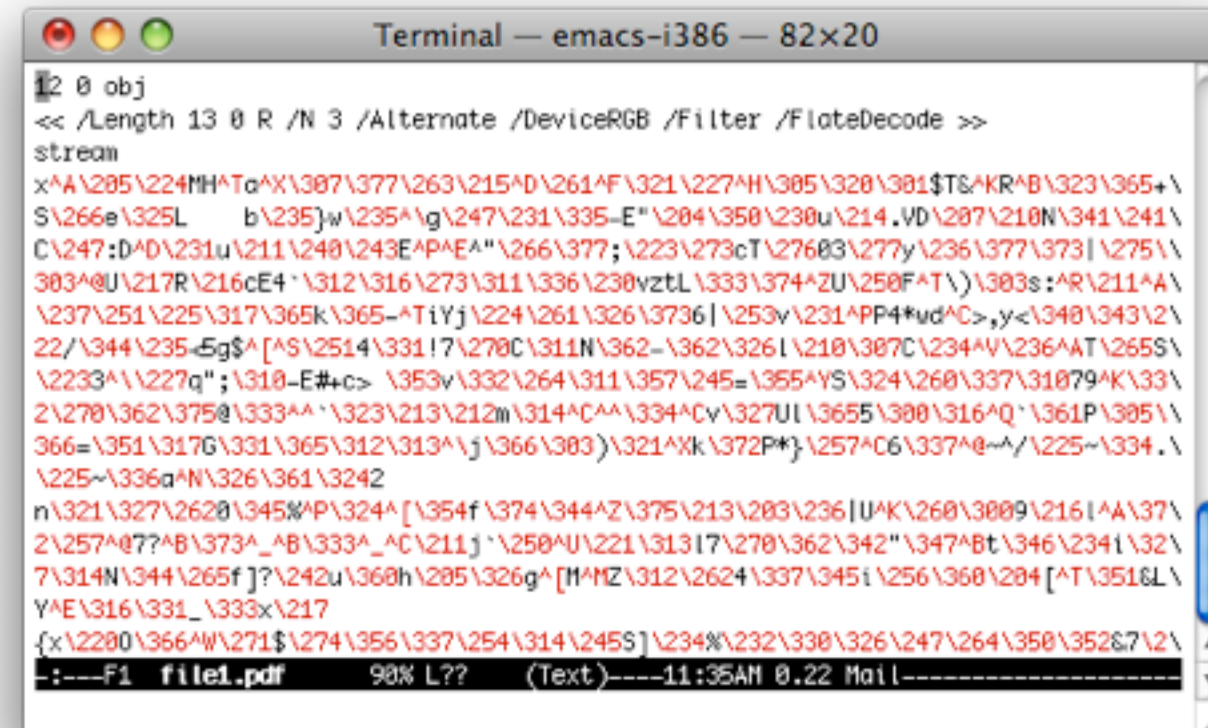


Individual "objects" in a PDF file may hold JPEGs, Text, FAX pages, JavaScript, and other content.

PDF JPEG:



PDF Text:



This 74166-byte PDF file has 144 sectors and 20 objects.

- ➪Compressed Text
- ➪Numbers (4)
- ➪Media Box (margins)
- ➪ColorSpace
- ➪ColorSpace
- ➪JPEG
- ➪Compressed Text
- ➪Array [/ICCBased 12 0 R]
- ➪Text
- ➪Array [/ICCBased 14 0 R]
- ➪Media Box
- ➪Page Count
- ➪Compressed Text
- ➪Font Table
- ➪Font Descriptor
- ➪Metadata

This is some text This is some text

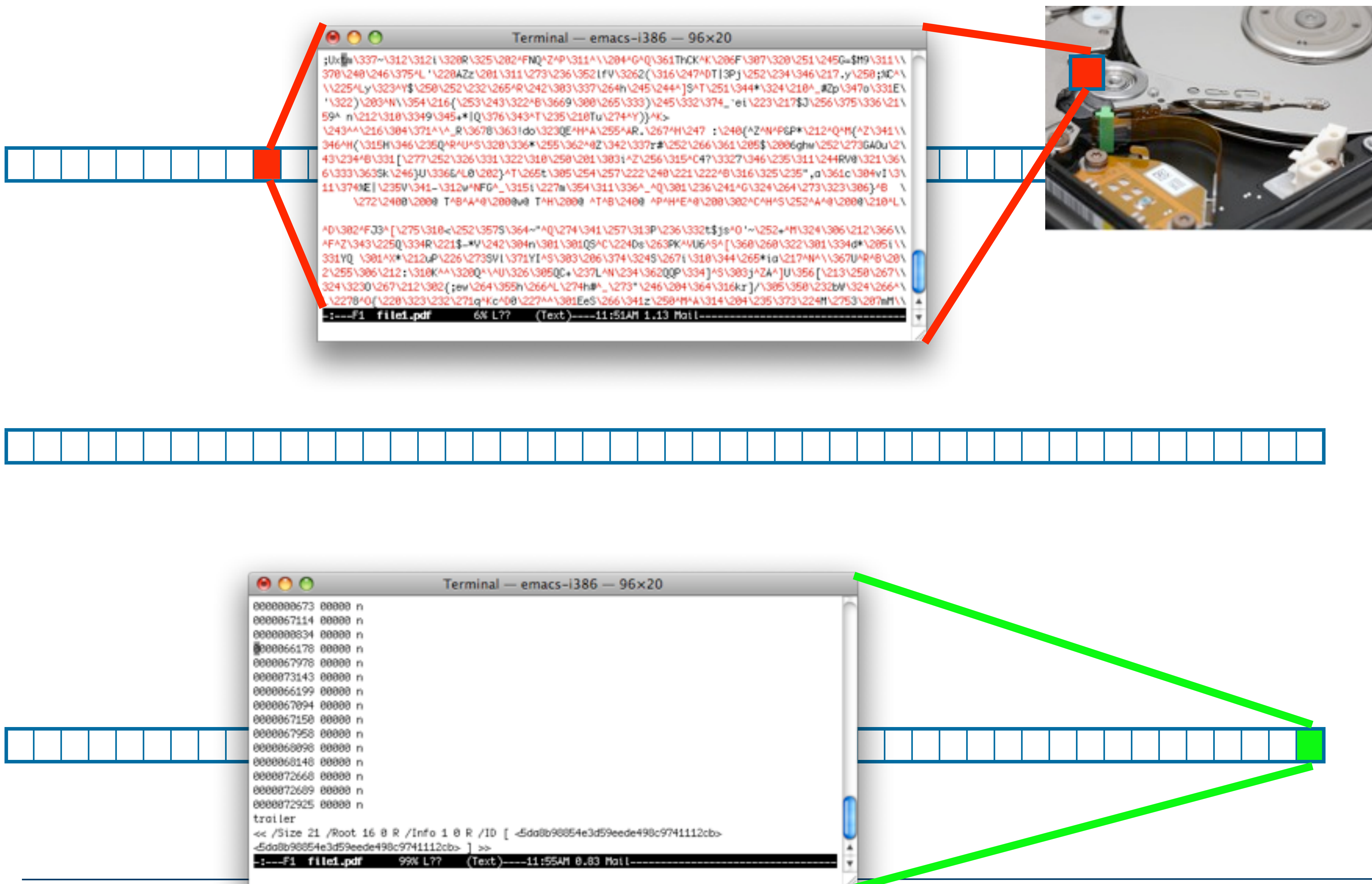
This is some text

This is some text

This is some text

This is some text

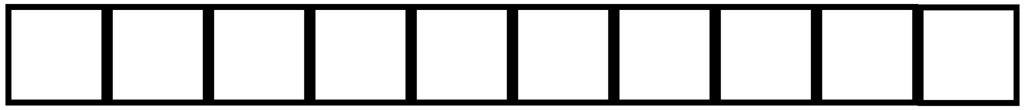
Some sectors are characteristically PDF data, others are just JPEGs or compressed text.



Most files on the hard drive are not fragmented. JPEGs in PDFs can be identified by scanning backwards.

In previous research, we found that only 15% of forensically interesting files are fragmented [Garfinkel 2007].

Therefore, we can use a sector's *context* to assist in identification:



Most files on the hard drive are not fragmented. JPEGs in PDFs can be identified by scanning backwards.

In previous research, we found that only 15% of forensically interesting files are fragmented [Garfinkel 2007].

Therefore, we can use a sector's *context* to assist in identification:

Most files on the hard drive are not fragmented. JPEGs in PDFs can be identified by scanning backwards.

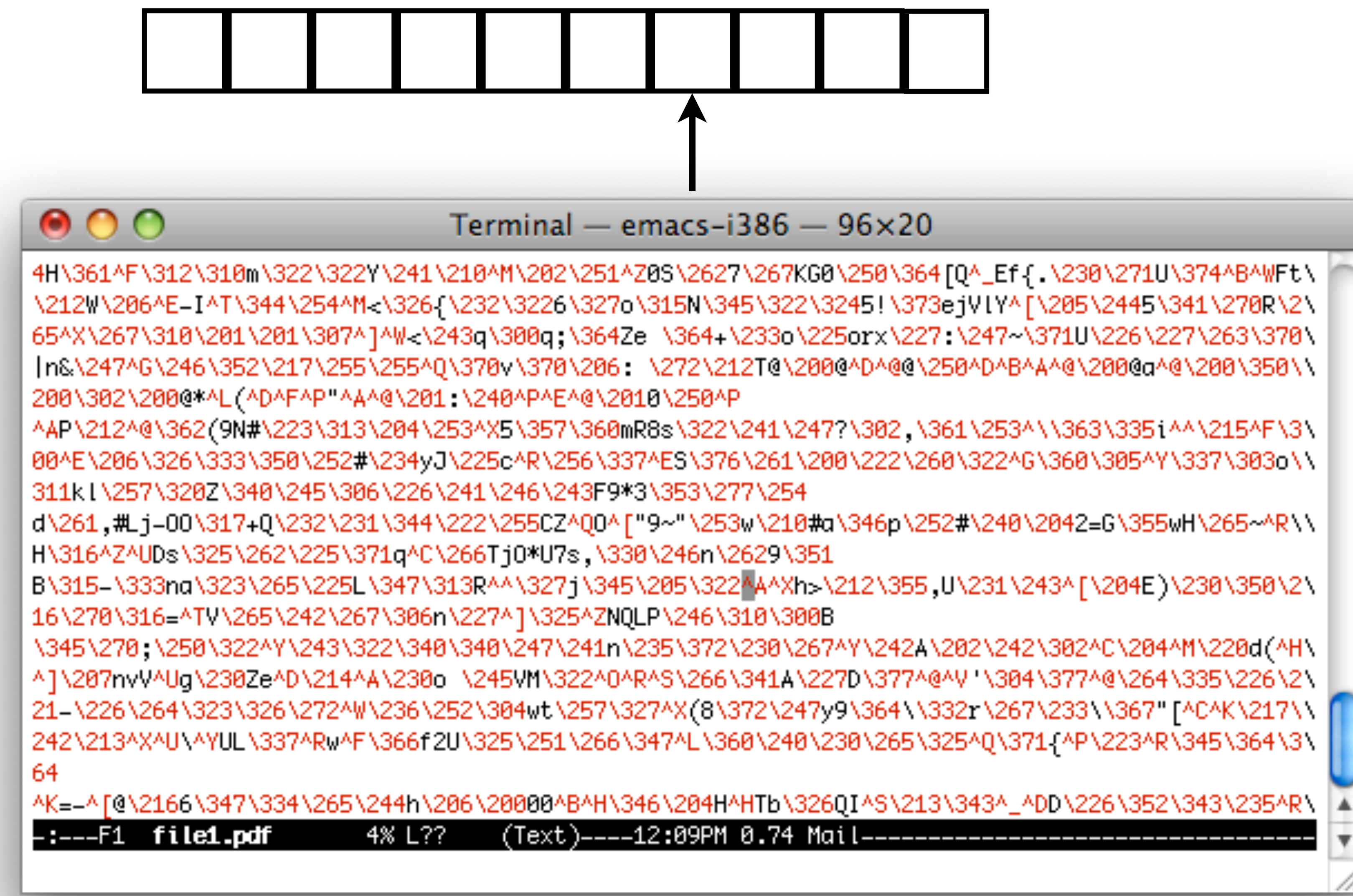
In previous research, we found that only 15% of forensically interesting files are fragmented [Garfinkel 2007].

Therefore, we can use a sector's *context* to assist in identification:

Most files on the hard drive are not fragmented. JPEGs in PDFs can be identified by scanning backwards.

In previous research, we found that only 15% of forensically interesting files are fragmented [Garfinkel 2007].

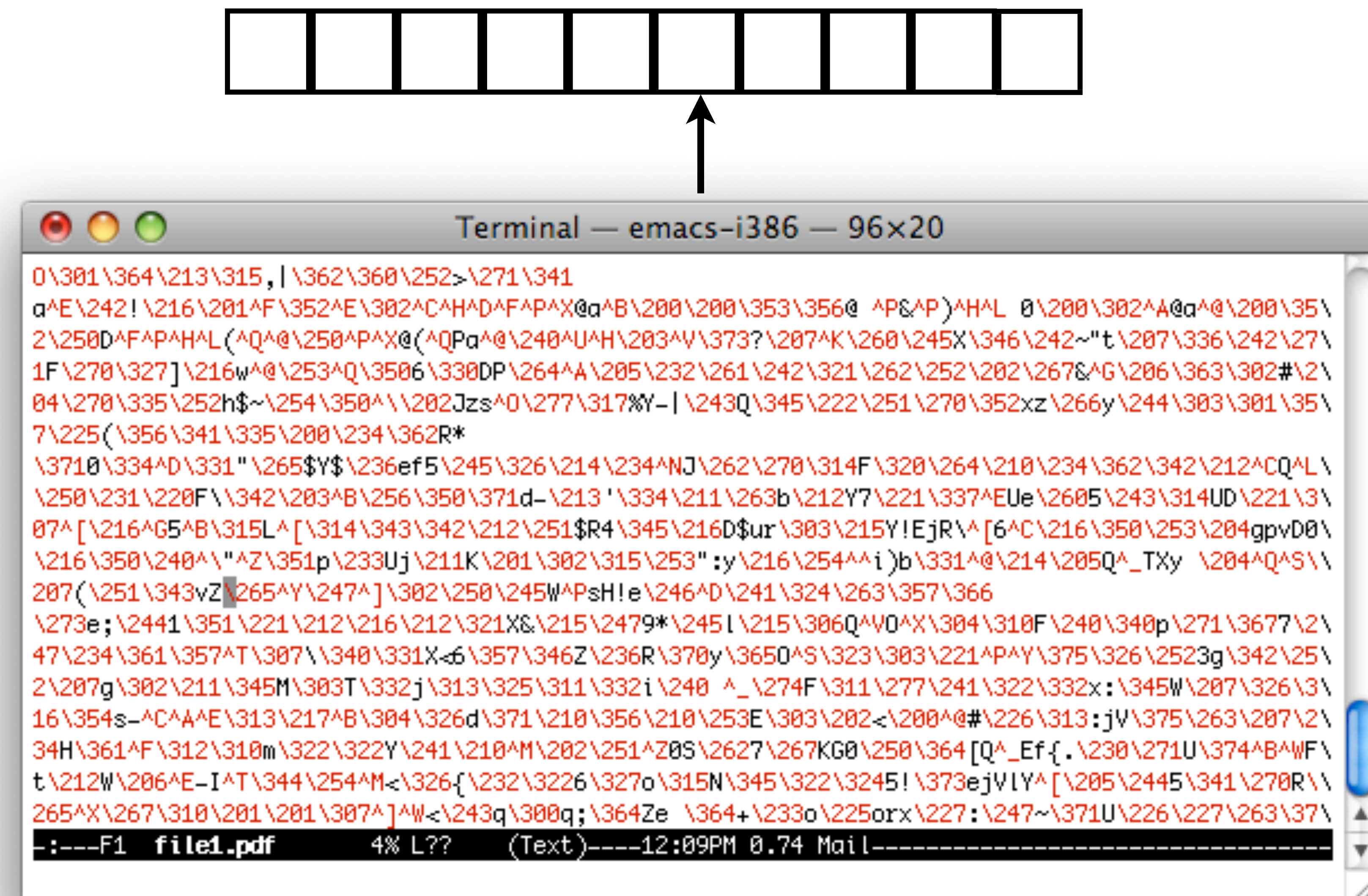
Therefore, we can use a sector's *context* to assist in identification:



Most files on the hard drive are not fragmented. JPEGs in PDFs can be identified by scanning backwards.

In previous research, we found that only 15% of forensically interesting files are fragmented [Garfinkel 2007].

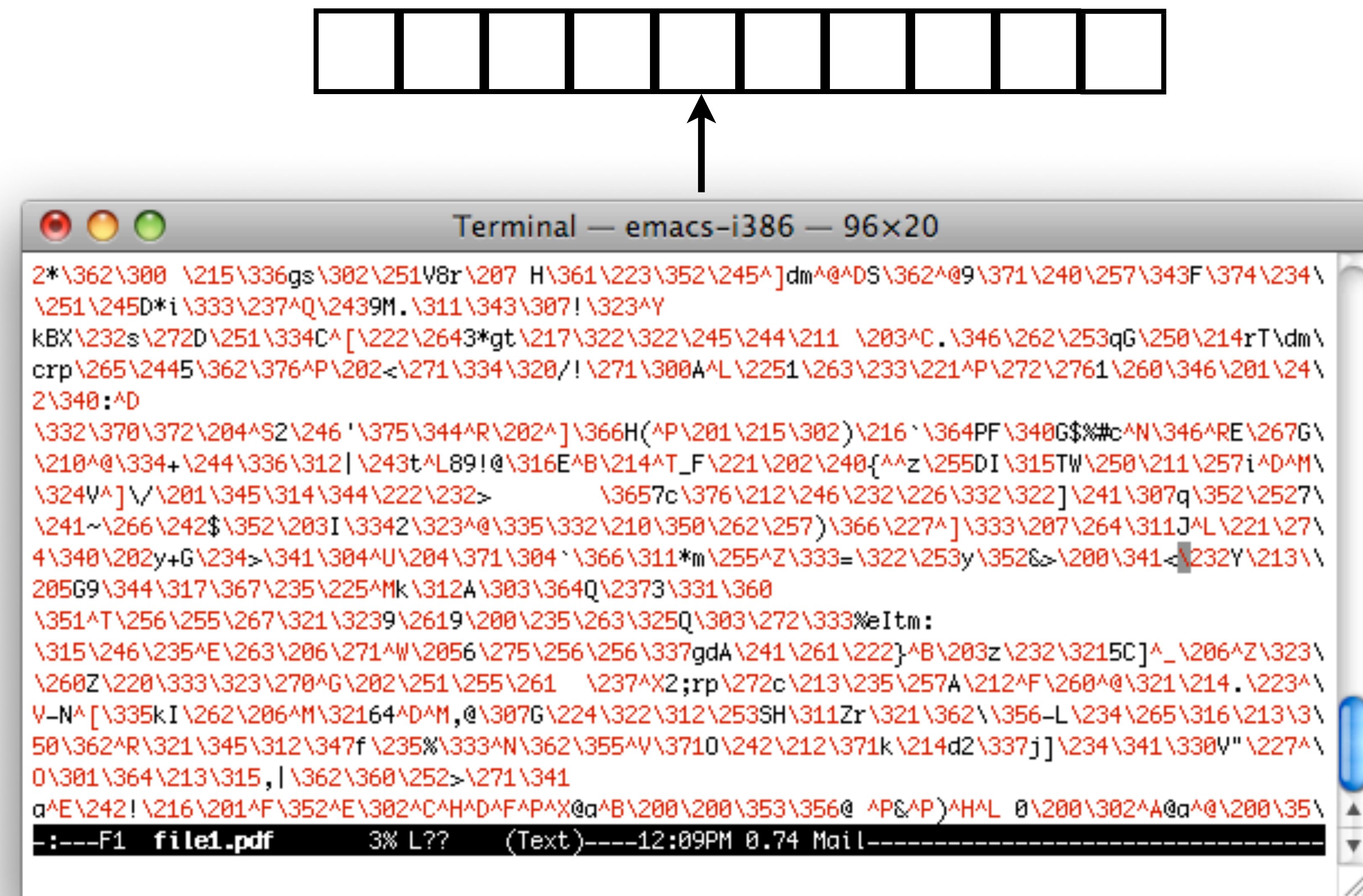
Therefore, we can use a sector's *context* to assist in identification:



Most files on the hard drive are not fragmented. JPEGs in PDFs can be identified by scanning backwards.

In previous research, we found that only 15% of forensically interesting files are fragmented [Garfinkel 2007].

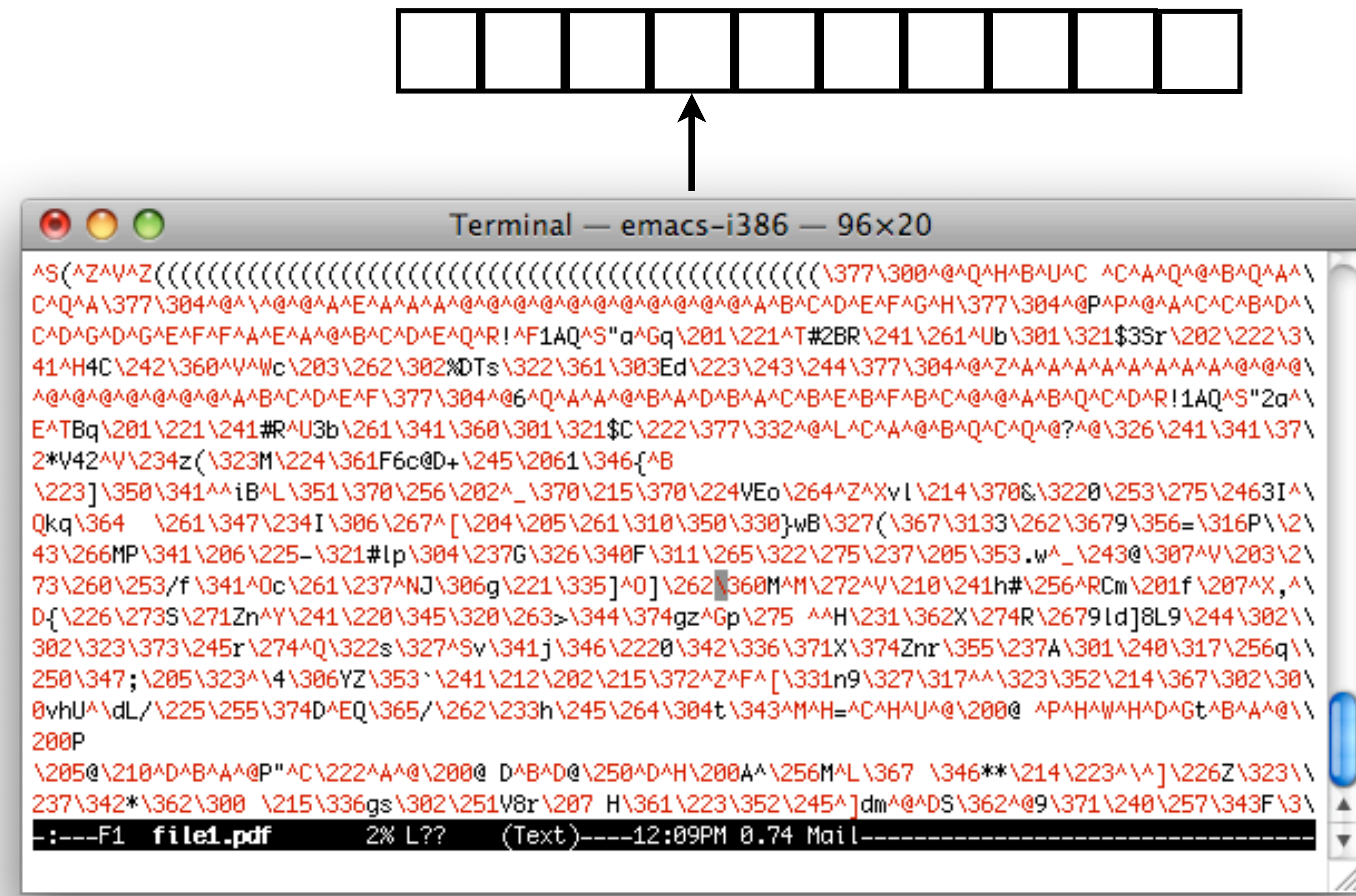
Therefore, we can use a sector's *context* to assist in identification:



Most files on the hard drive are not fragmented. JPEGs in PDFs can be identified by scanning backwards.

In previous research, we found that only 15% of forensically interesting files are fragmented [Garfinkel 2007].

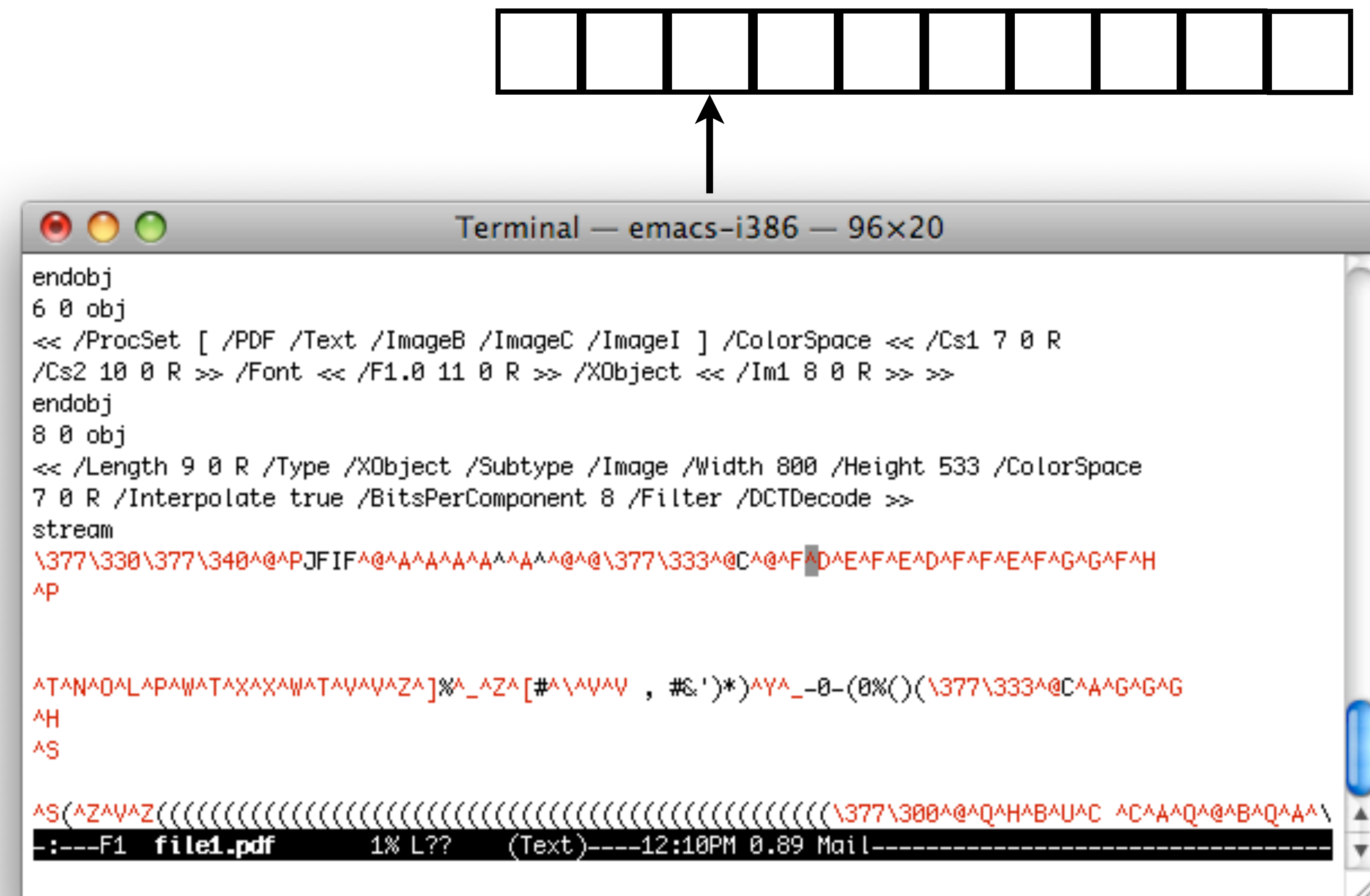
Therefore, we can use a sector's *context* to assist in identification:



Most files on the hard drive are not fragmented. JPEGs in PDFs can be identified by scanning backwards.

In previous research, we found that only 15% of forensically interesting files are fragmented [Garfinkel 2007].

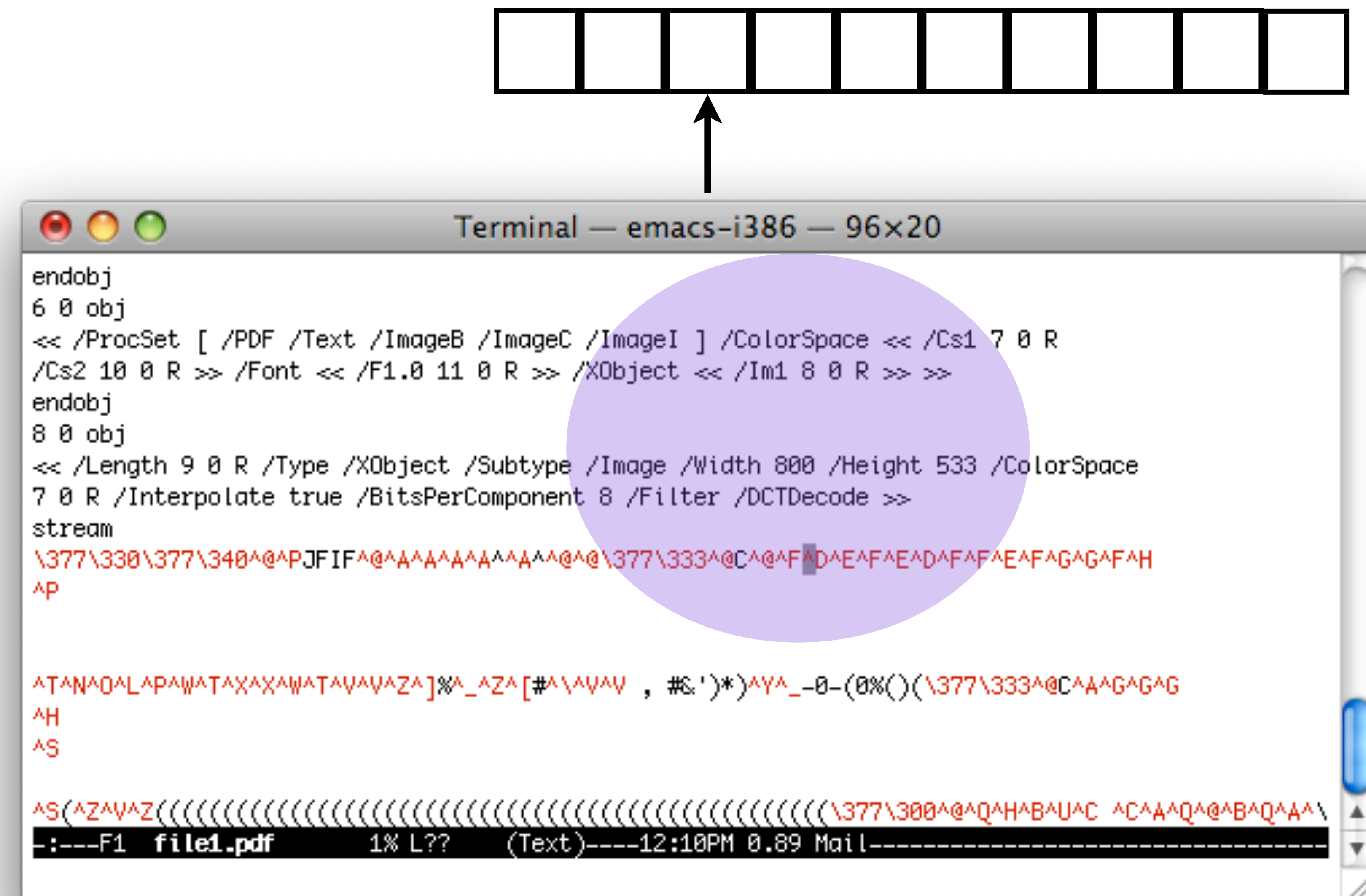
Therefore, we can use a sector's *context* to assist in identification:



Most files on the hard drive are not fragmented. JPEGs in PDFs can be identified by scanning backwards.

In previous research, we found that only 15% of forensically interesting files are fragmented [Garfinkel 2007].

Therefore, we can use a sector's *context* to assist in identification:



Sectors can also be identified from statistical properties.

File type	Identified By
NULL	direct examination.
HTML	n-gram analysis
JPEG	High-entropy with markers
ZIP	High-entropy that's not encrypted
Encrypted	High-entropy that passes encryption tests

Using sector identification, we can identify the *content* of a hard drive within 10 seconds (after it spins up).

Time to read 10,000 randomly chosen 64K runs: 4.4 seconds

Identifiable:

- ⇨Blank sectors
- ⇨JPEGs
- ⇨Encrypted data
- ⇨HTML

Standard error:

- ⇨0.1% for 10% determination
- ⇨5% for 50% determination (max error)

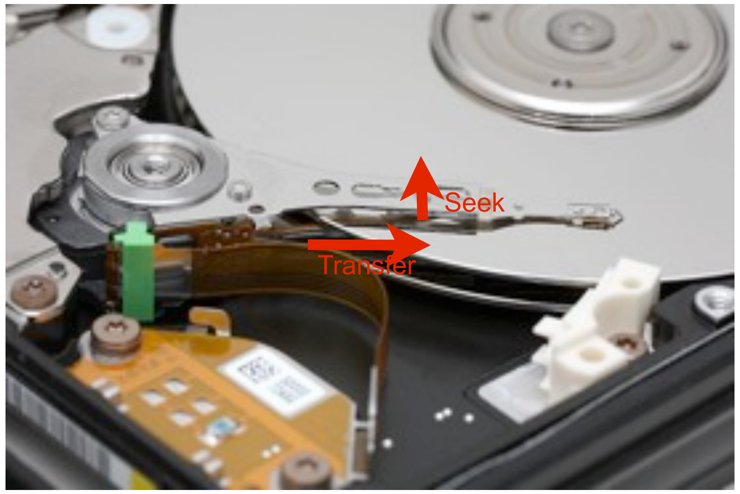
Sample report:

- ⇨10% encrypted; 30% JPEG; 50% MP3

Conclusion:
We can dramatically speed traditional forensics.

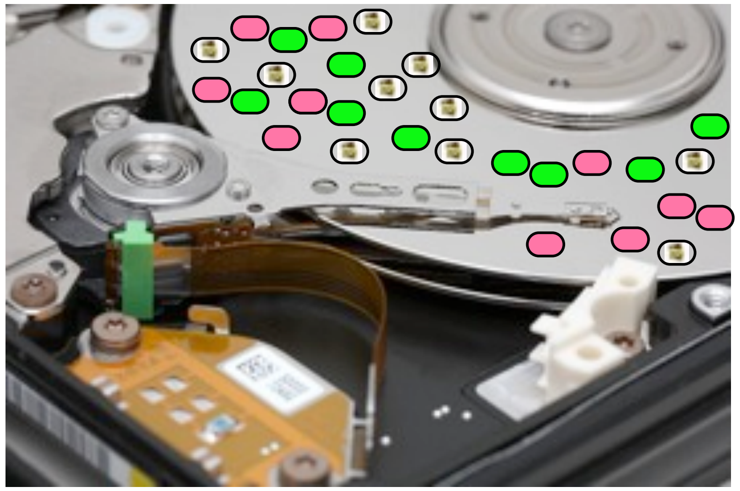
Traditional forensics:

- ➪Speed processing by sorting files by sector order.
- ➪Irrelevant for flash storage.
- ➪Works today.



Statistical Sampling:

- ➪Determine the content of a disk in 10 seconds.
- ➪Full content, not file content.
- ➪Research today.



For further information, see http://simson.net/

Question?