File System Archaeology

New techniques to help recover data from a FAT32 file system

Talk

- Data recovery
- New? Techniques
  - Reconstruct file system activity
- FAT32
  - Simple example
- Fragmented files
- Verification
- Exercise

Data Recovery

- Two main approaches
  - Use metadata
  - File carving
- Both approaches have problems with fragmentation

Data Recovery 2

- File Carving
  - Continually improving
    - New ideas to handle fragmented files
- But …
  - Slow
  - Doesn’t get metadata
    - File names, dates, length

New Techniques

- Use metadata to reconstruct file activity
  - Can reconstruct how files are laid out on disk
  - Can handle fragmentation
  - Don't need to read every cluster
    - Very efficient
- Can be used to supplement file carving techniques

New Techniques 2

- File validation strategies which don’t look at the file contents
- Reconstruction of FS data structures
- Knitting together fragmented directory clusters

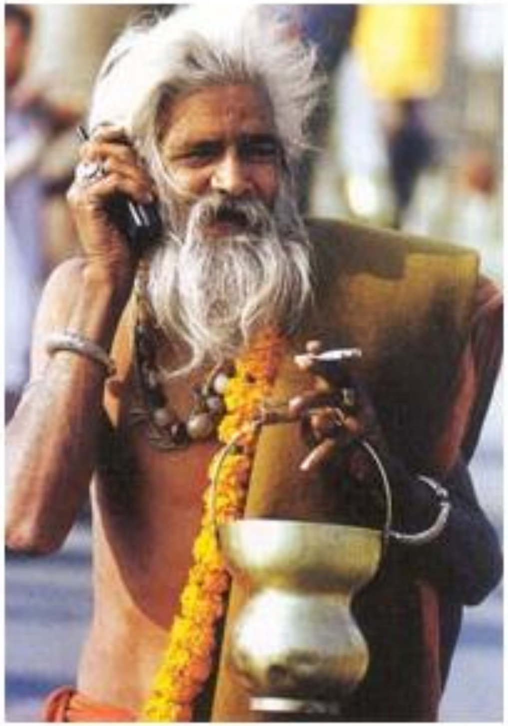
What?
Did you say FAT?
You mean that 30
year old file system.

FAT 32

- Portable and so used on
  - Phones
  - Cameras
  - USB keys

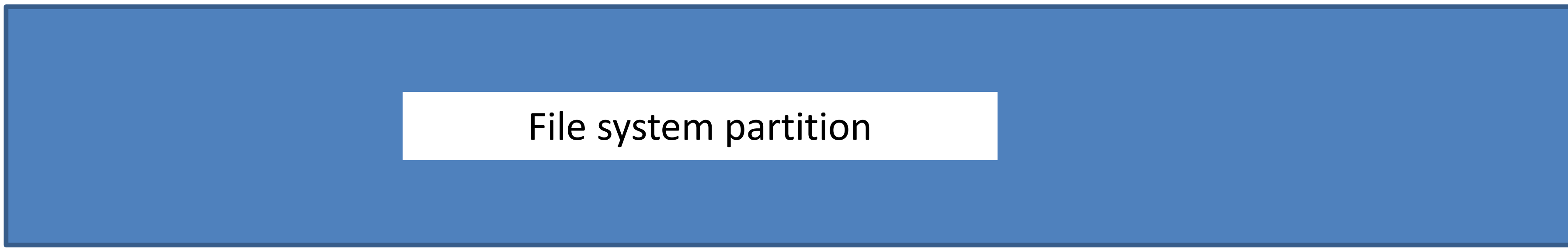
More cell phones than people

In 30 countries around the world, from Aruba to Italy to Hong Kong, mobile phone penetration has **past 100 percent**. Translation: the number of cell phone subscriptions has exceeded the size of the population. That's according to end-of-Q1-2006 data just released by London-based researcher Informa Telecom&Media. Here is the list:



Turks & Caicos Islands: 161.8%
Aruba: 150.8
Luxembourg: 140.7
Lithuania: 139.9
Cayman Islands: 136.4
Netherlands Antilles: 134.0
Grenada: 133.3
Israel: 125.9
Italy: 122.4
Cyprus: 121.5
Macau: 121.3
Bahrain: 117.8
Greece: 114.7
Czech Republic: 114.0
UAE: 113.9
Jersey: 113.6
Sweden: 112.5
Hong Kong: 110.8
UK: 110.1
Estonia: 108.6
Spain: 108.0
Austria: 107.3

FAT32 intro



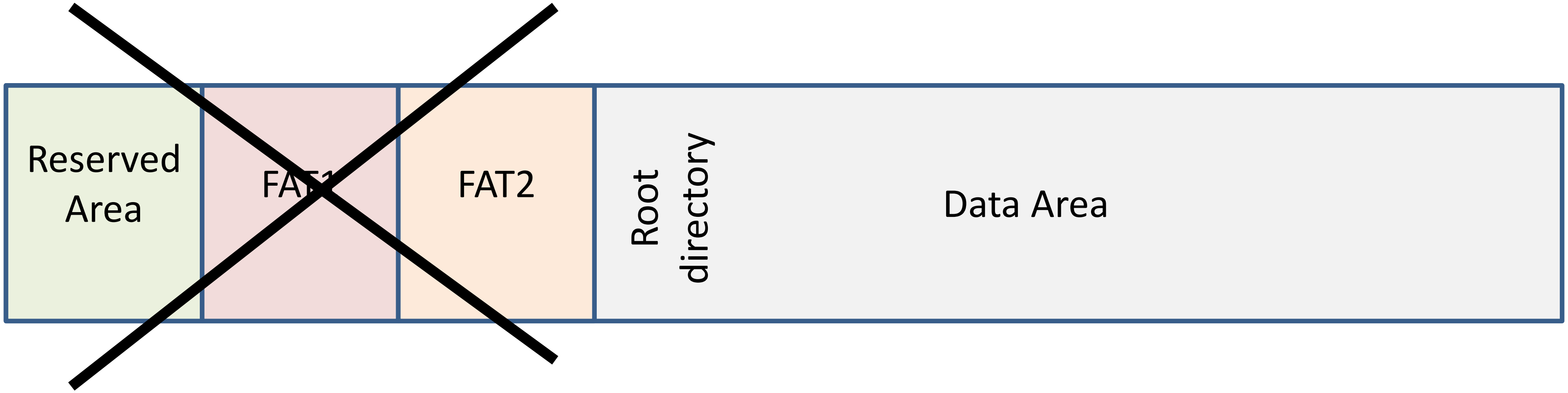
FAT32 intro

Formatting the disk initialises the data structures, creates a FAT and backup FAT and

FAT32 intro

For the purposes of this talk, we are not interested in anything other than the data area

FAT32 intro



For the purposes of this talk, we are not interested in anything other than the data area

FAT32 intro

For the purposes of this talk, we are not interested in anything other than the data area

FAT32 intro

The Data area is where all the file data and the directory information is held.

FAT32 intro

- Divided into clusters. (Say 4k, or 4096 bytes)
- Files are stored as a sequence of clusters
- Sequence information stored in the FAT

FAT32 intro

- Divided into clusters. (Say 4k, or 4096 bytes)
- Files are stored as a sequence of clusters
- Sequence information stored in the FAT

FAT32 intro

- Divided into clusters. (Say 4k, or 4096 bytes)
- Files are stored as a sequence of clusters
- Sequence information stored in the FAT
  - Oops … we’ve lost the FAT
    => we have to reconstruct the sequence

FAT32 intro

- Creating a file
  - Create an entry in the directory

FAT32 Typical Operations

- Creating a file (say 6k, requires 2 clusters)

FAT32 Typical Operations

- Creating a file (say 6k, requires 2 clusters)
  - Create an entry in the directory
    - Holds metadata name, dates, start, len

FAT32 Typical Operations

- Creating a file (say 6k, requires 2 clusters)
  - Create an entry in the directory
    - Holds metadata name, dates, start, len
  - Allocate clusters

FAT32 Typical Operations

- Creating a subdirectory
  - Create an entry in the root directory

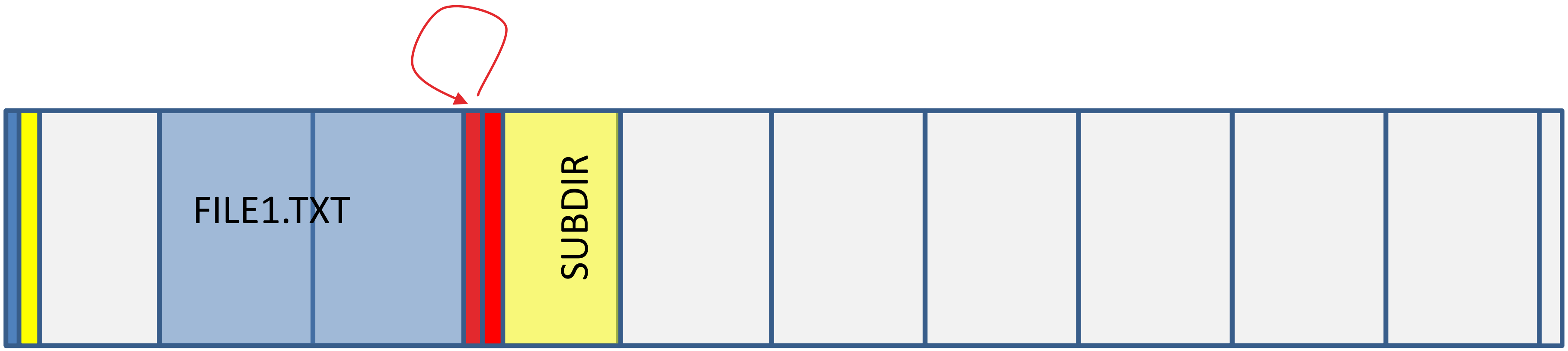
FAT32 Typical Operations

- Creating a subdirectory
  - Create an entry in the root directory
  - Allocate cluster

FAT32 Typical Operations

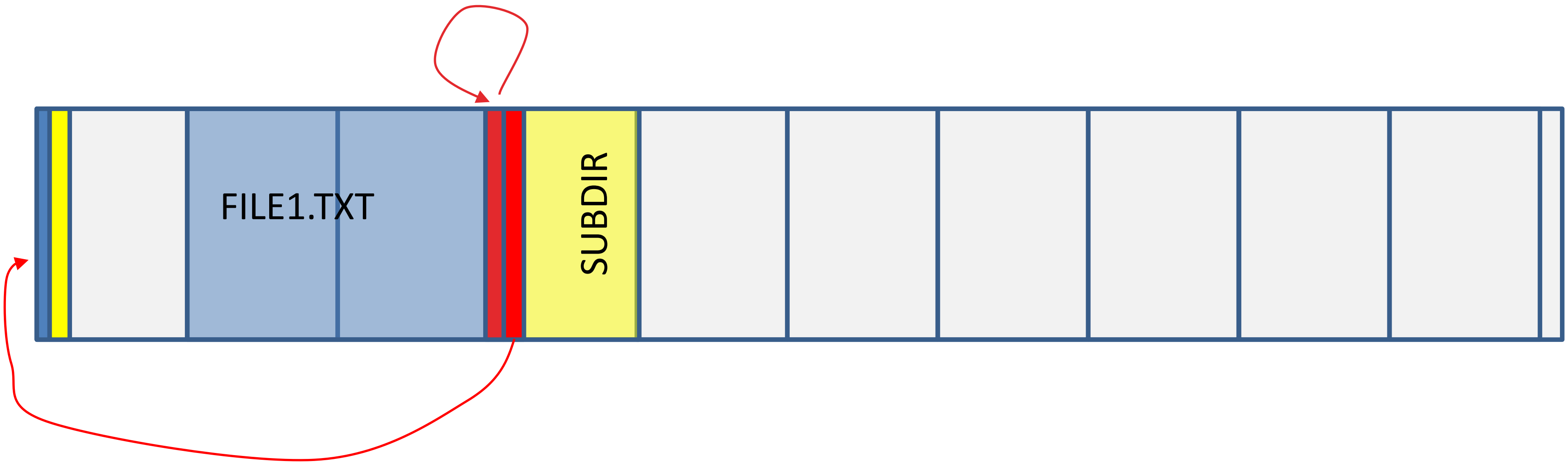
- Creating a subdirectory
  - Create an entry in the root directory
  - Allocate cluster
  - Create . and .. entries

FAT32 Typical Operations



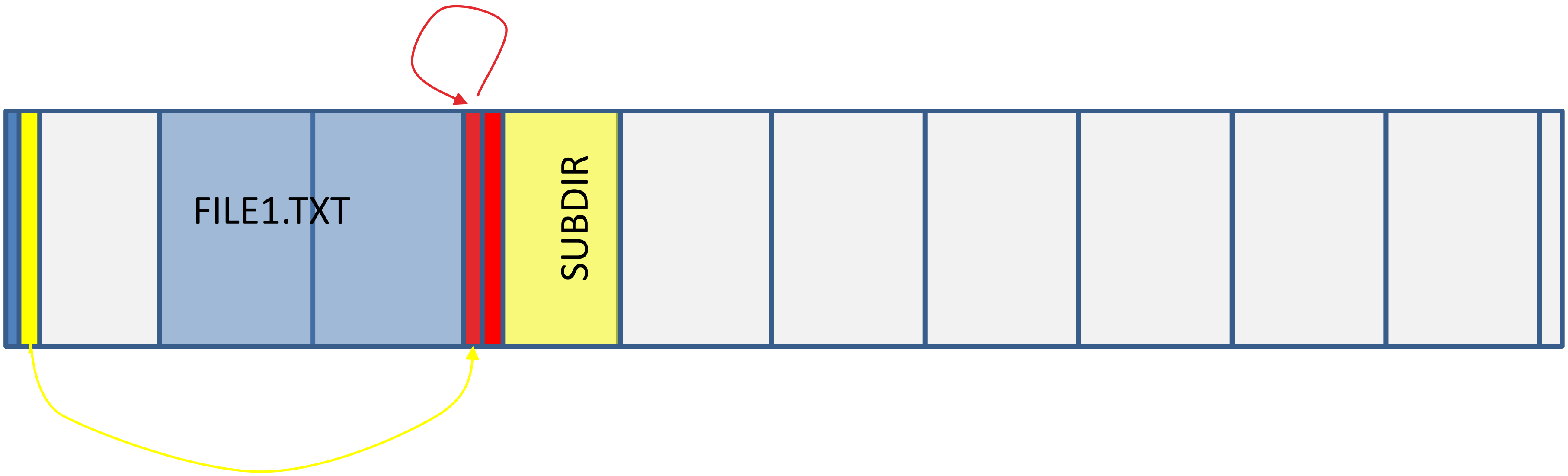
- The . entry points to itself

FAT32 Typical Operations



- The . entry points to itself
- The .. entry points to the parent directory (in this case, the root directory)

FAT32 Typical Operations

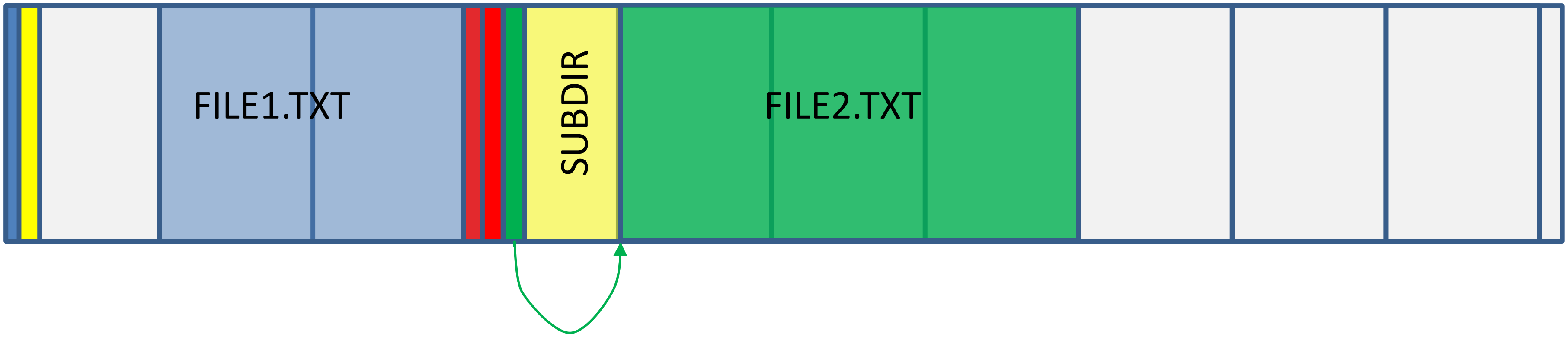


- Note that the . entry contains the same values as the subdirectory entry except for the name. Redundancy is good for forensics.

Allocation Strategy

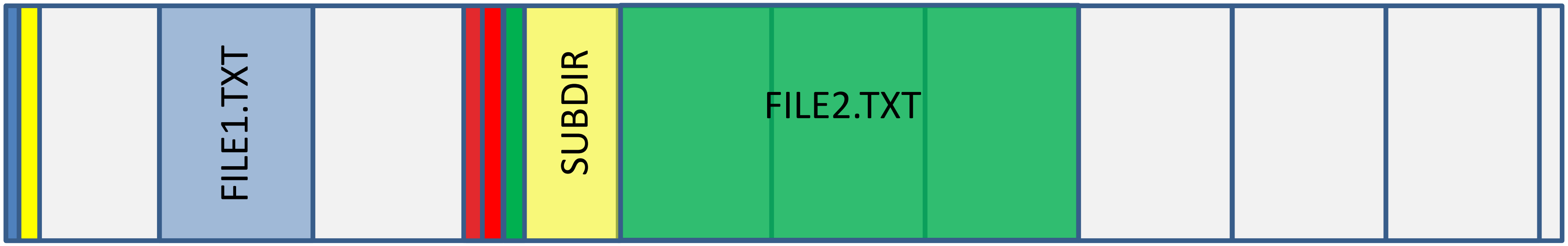
- The file system needs to allocate clusters
  - it is free to choose any free cluster
  - FAT32 supports next available
    - the next available cluster after the last one allocated

Allocation Strategy



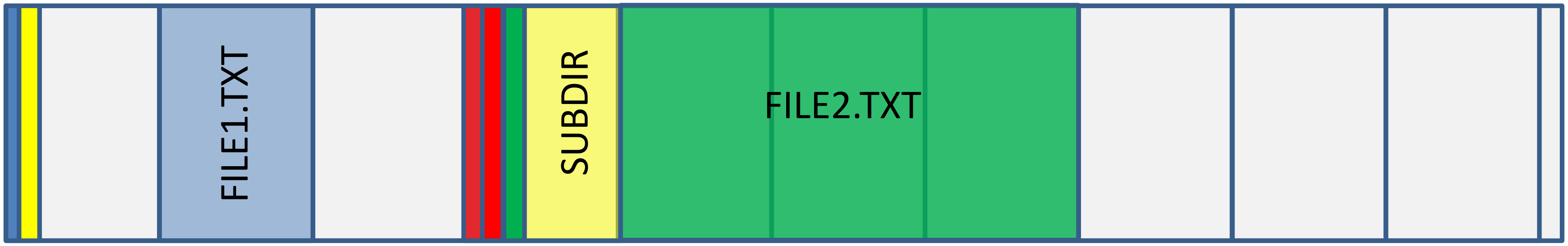
- A new file (3 clusters) is created in SUBDIR

Allocation Strategy



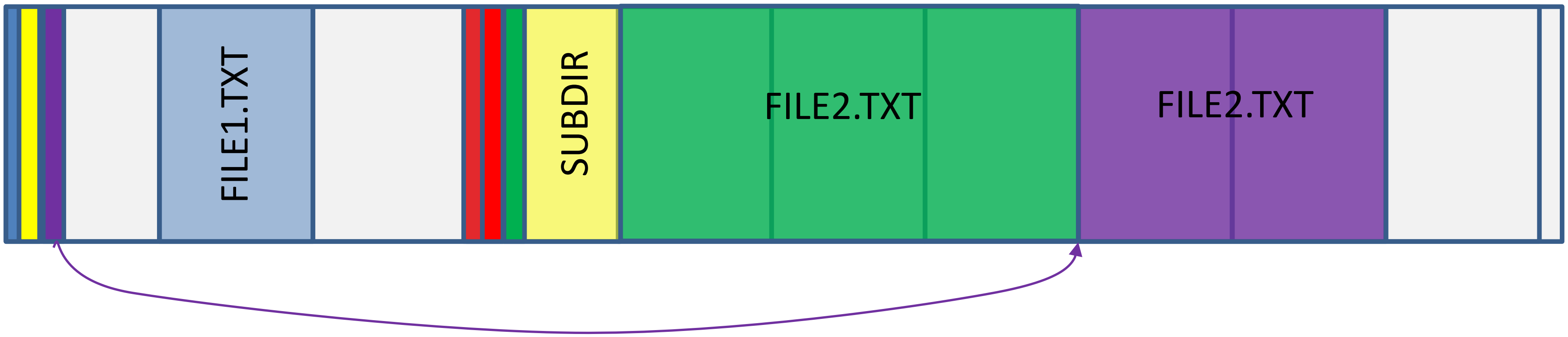
- A new file (3 clusters) is created in subdir
- FILE1.TXT is edited, reduced to 1 cluster

Allocation Strategy



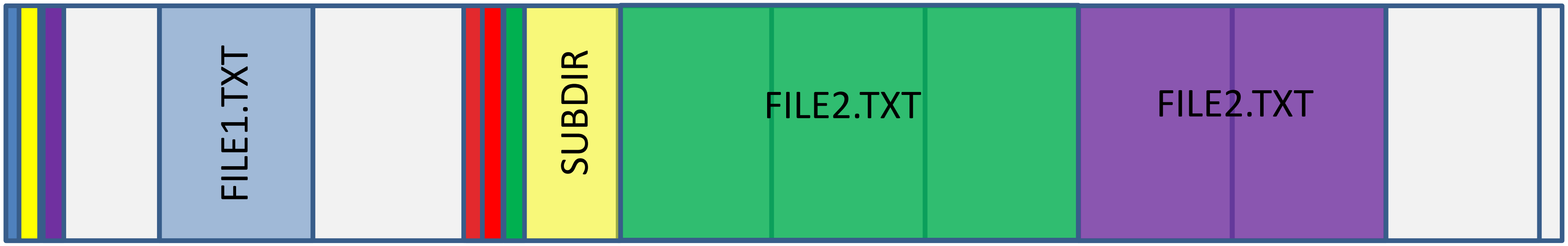
- A new file (3 clusters) is created in subdir
- FILE1.TXT is edited, reduced to 1 cluster
- FILE3.TXT is created in the root directory

Allocation Strategy



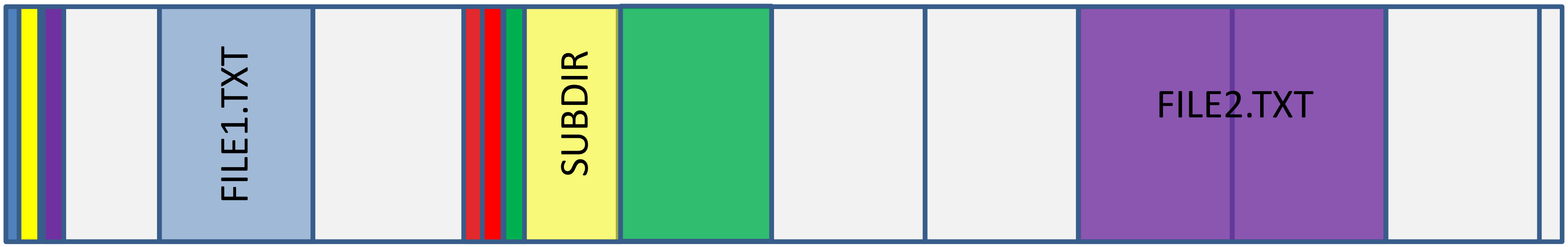
- A new file (3 clusters) is created in subdir
- FILE1.TXT is edited, reduced to 1 cluster
- FILE3.TXT is created in the root directory

Allocation Strategy



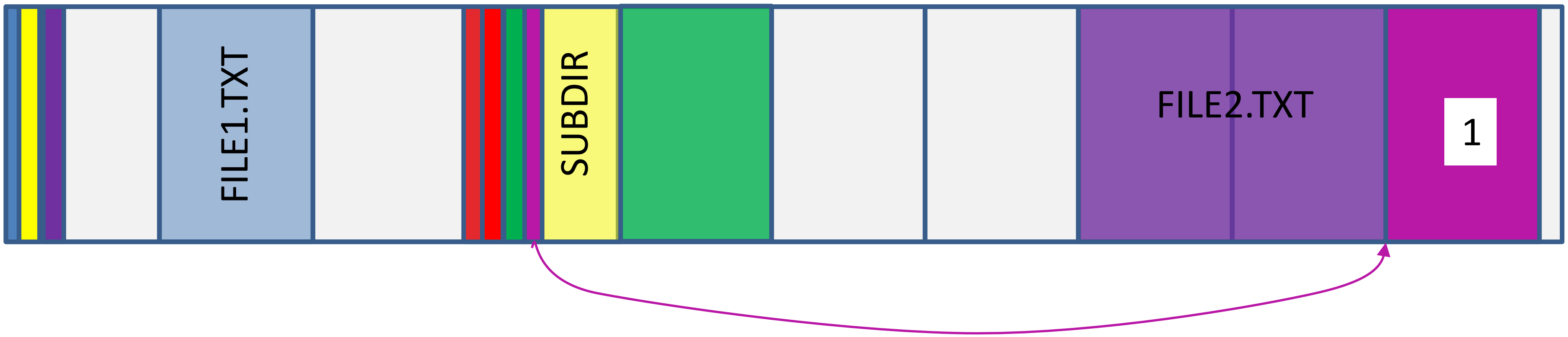
- A new file (3 clusters) is created in subdir
- FILE1.TXT is edited, reduced to 1 cluster
- FILE3.TXT is created in the root directory
- FILE2.TXT is reduced in size to 1 cluster

Allocation Strategy



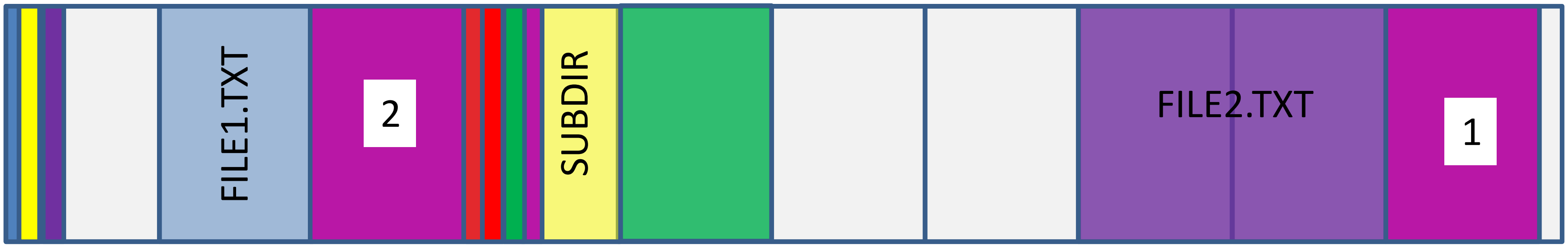
- A new file (3 clusters) is created in subdir
- FILE1.TXT is edited, reduced to 1 cluster
- FILE3.TXT is created in the root directory
- FILE2.TXT is reduced in size to 1 cluster

Allocation Strategy



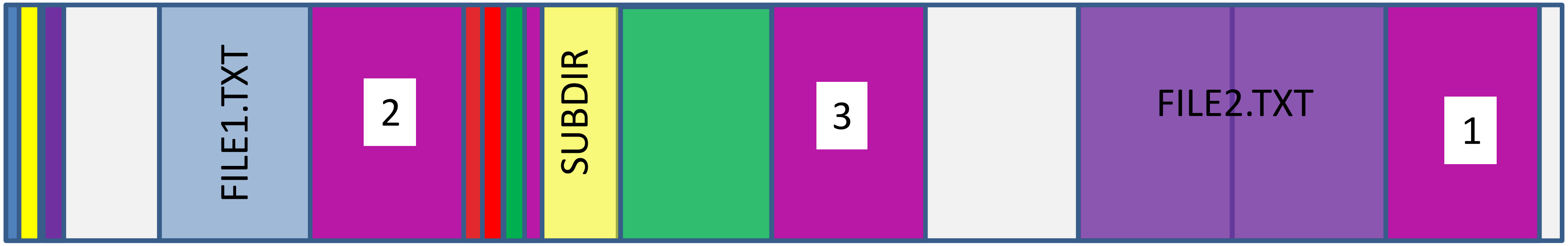
- A new file (3 clusters) is created in subdir
- FILE1.TXT is edited, reduced to 1 cluster
- FILE3.TXT is created in the root directory
- FILE2.TXT is reduced in size to 1 cluster
- FILE4.TXT (3 clusters) is created

Allocation Strategy



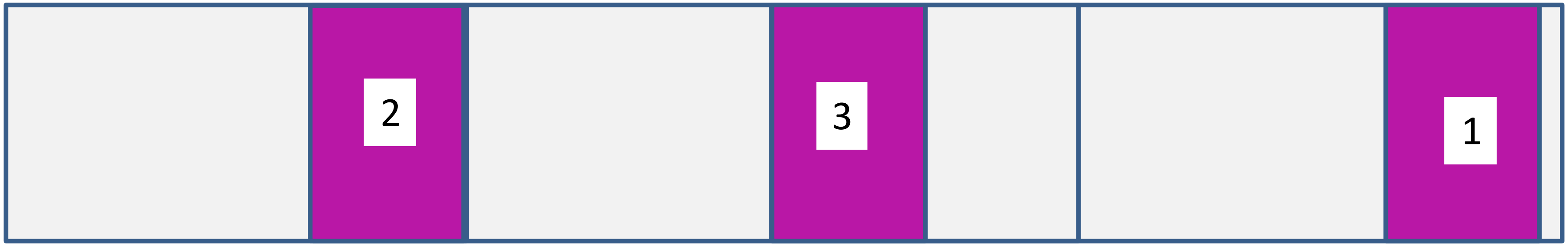
- A new file (3 clusters) is created in subdir
- FILE1.TXT is edited, reduced to 1 cluster
- FILE3.TXT is created in the root directory
- FILE2.TXT is reduced in size to 1 cluster
- FILE4.TXT (3 clusters) is created

Allocation Strategy



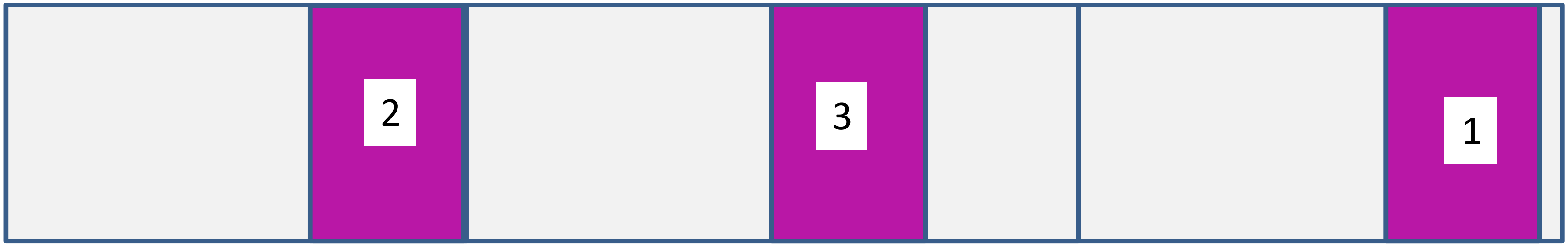
- A new file (3 clusters) is created in subdir
- FILE1.TXT is edited, reduced to 1 cluster
- FILE3.TXT is created in the root directory
- FILE2.TXT is reduced in size to 1 cluster
- FILE4.TXT (3 clusters) is created

Allocation Strategy



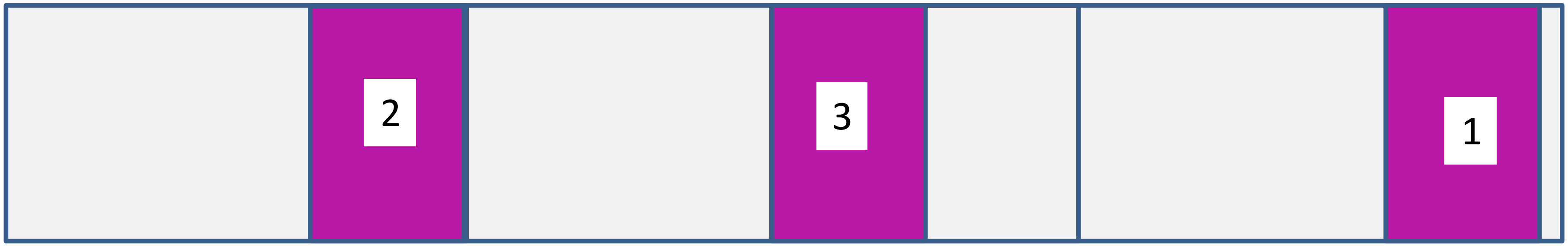
- FILE4.TXT is quite fragmented

Allocation Strategy



- FILE4.TXT is quite fragmented ...
  - but ...

Allocation Strategy



- FILE4.TXT is quite fragmented …
  - but …
  - if we know the sequence of operations and the allocation strategy, we can find out which clusters the files were allocated.

Data Recovery

- Find all the directory entries (assume they all still exist … only the FATs have been destroyed)
- Sort entries by creation time
- for each entry
  - use the start cluster and number of clusters to see where it would go.
  - when we come to FILE4.TXT it has only one place to go.

Problems

- Defragmentation
  - Let’s assume it doesn’t happen
- Deleted files
  - We don’t have the time they were deleted
- Deleted directories
  - Maybe overwritten and we lose many directory entries.

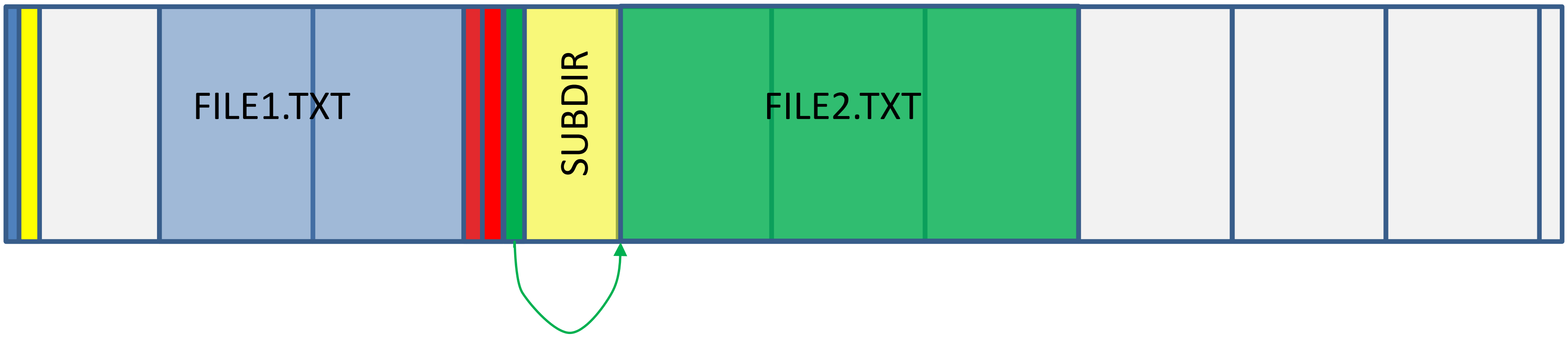
Problems 2

- Modification time
  - There’s only one and a file may be modified many times
  - It’s accurate to 2 seconds (creation time to 10ms)
  - A file may be modified and still occupy the same number of clusters
    - the last modification time is an upper bound

File Fragmentation

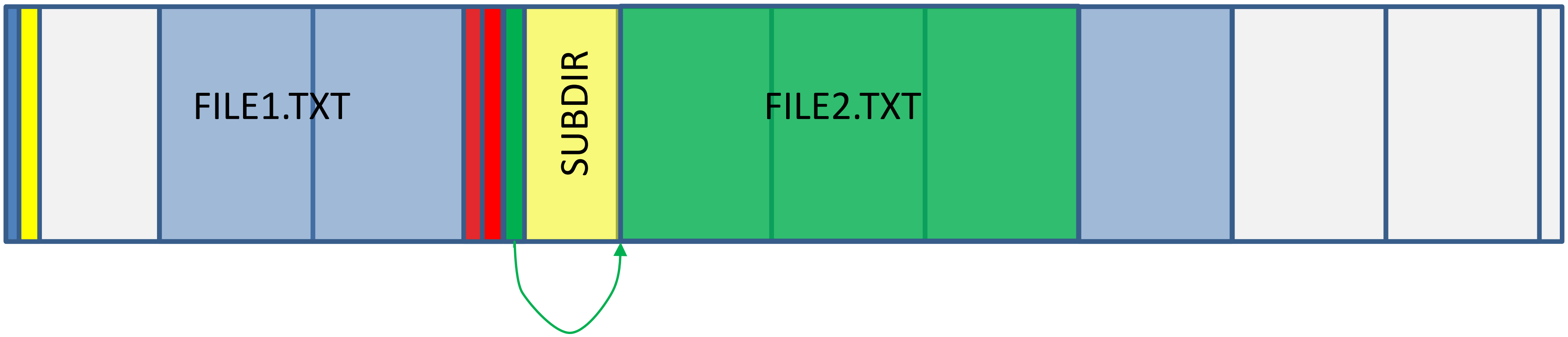
When a file is increased in size

Let’s go back in time



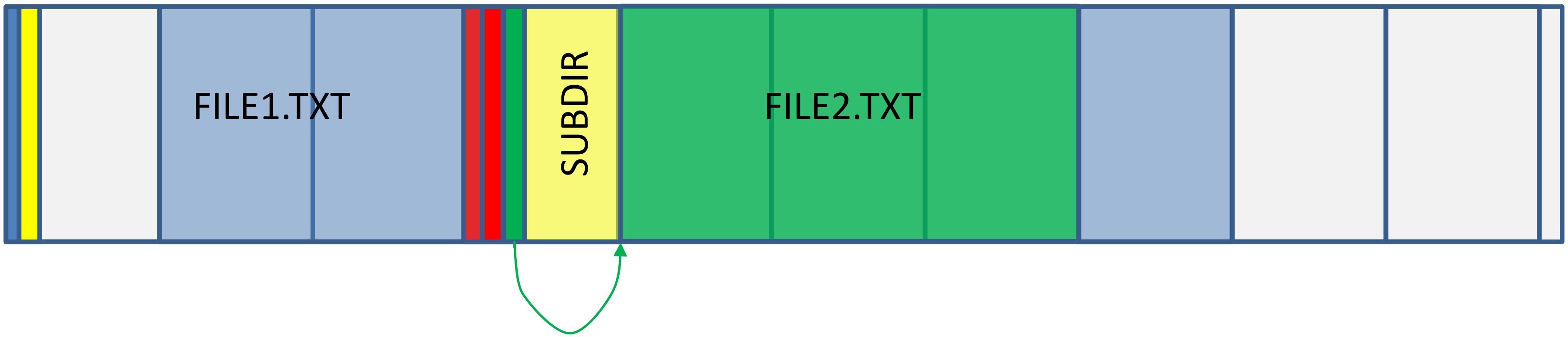
- FILE1.TXT is increased by one cluster

Let’s go back in time



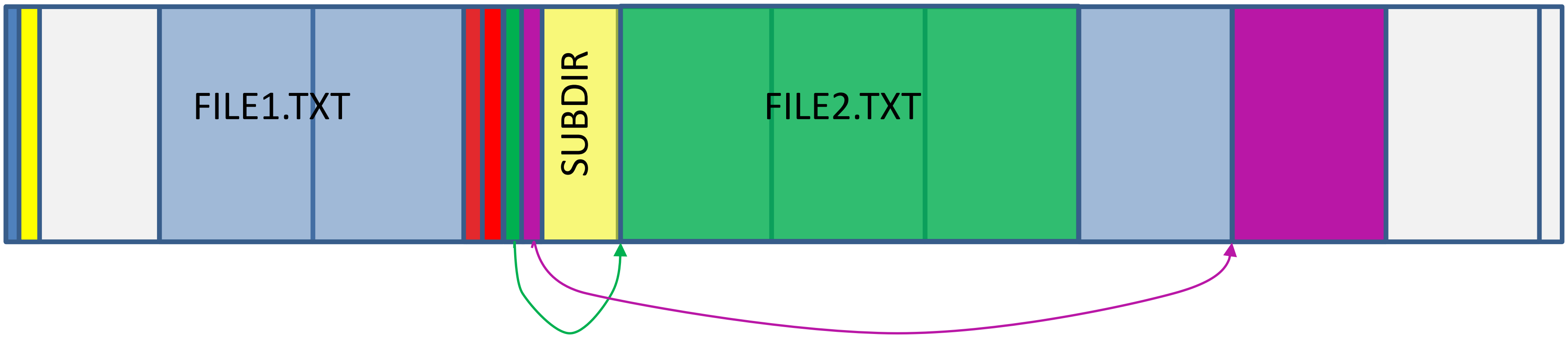
- FILE1.TXT is increased by one cluster

Let’s go back in time



- FILE1.TXT is increased by one cluster
- FILE5.TXT (1 cluster is created)

Let’s go back in time



- FILE1.TXT is increased by one cluster
- FILE5.TXT (1 cluster is created)
- And it is obvious where FILE1.TXT ends up.

If only there was a way to detect the end of a file

- This would help

Techniques

- Technique Number 1
  - Recover file system activity and use in conjunction with a known allocation strategy to recover file cluster layout.

Techniques

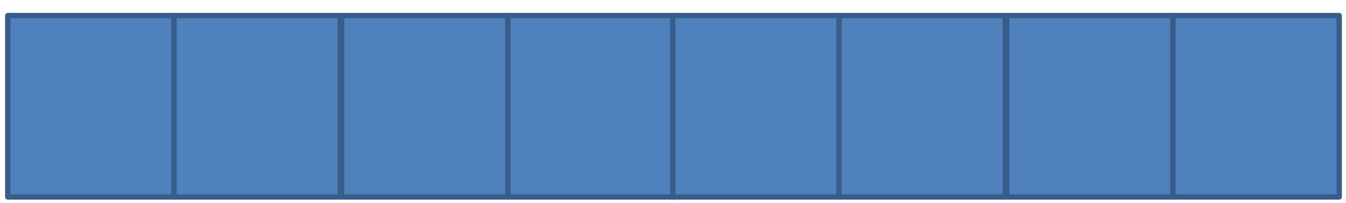
- Technique Number 2
  - We can use the redundancy of the . and .. directory entries to reliably determine cluster size and the start of the data area.
  - How?
  - we have two measures of distance. We know the byte offset where the . entry occurred and the parent entry. We also know the cluster number of each of these.
  - just divide the length by the length in clusters

Detecting the end of a file

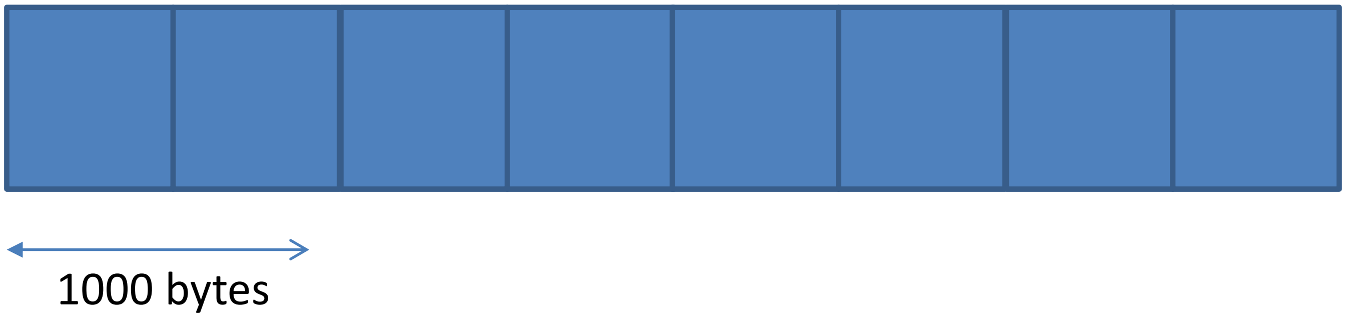
- RAM slack
  - Data written to the disk is always written in blocks of 512.
  - If a file of 1 byte is being created, just contains a single '?'
    - 512 bytes of memory is allocated
    - The first byte is set to the '?'
    - Remaining bytes are set to zero
- Idea of an ex-student (Anthony Walters)

Detecting end of file

- From the directory entry we know the length of the file
  - so we know how many bytes are in the last cluster
  - Let’s say there are 1000 bytes in the last cluster
- The cluster is, say 4K, that is 8 sectors in size

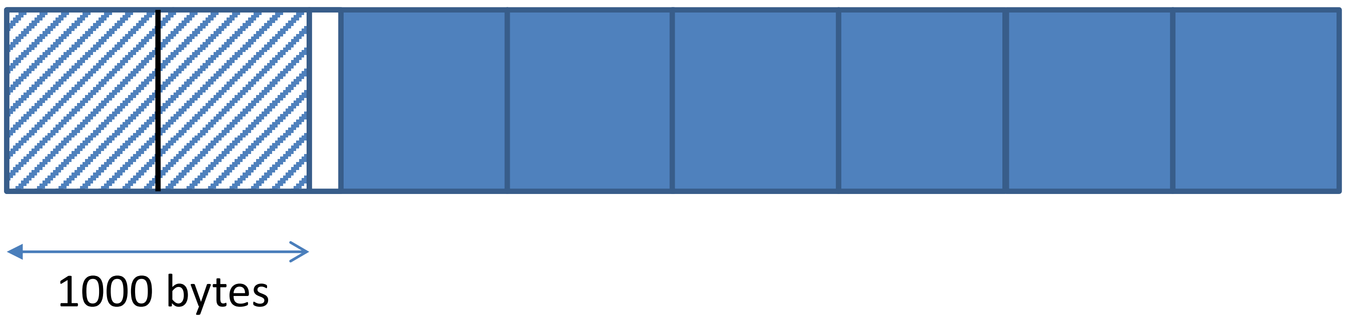


Detecting end of file



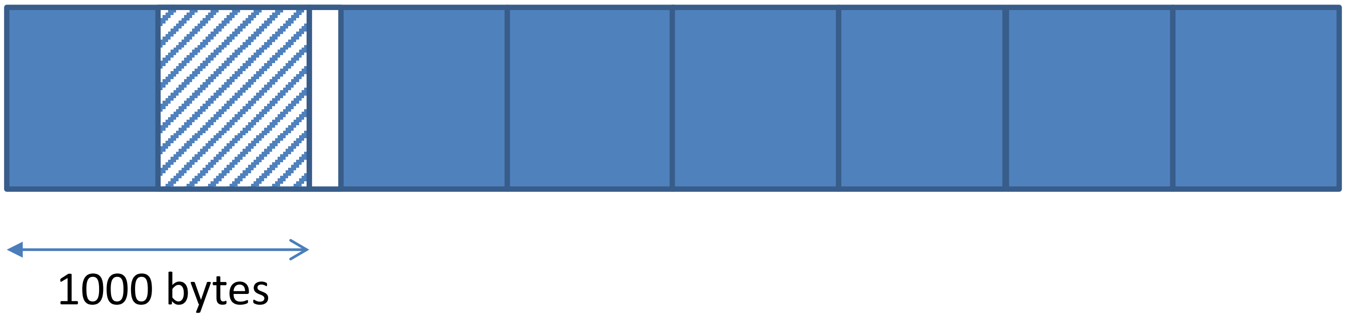
- We know that we are 1000 bytes into the last cluster
  - that is into the second sector
  - the end of that sector will contain zeros

Detecting end of file



- We know that we are 1000 bytes into the last cluster
  - that is into the second sector
  - the end of that sector will contain zeros

Detecting end of file



- So, we search for a cluster which has a second sector which contains the end of the file and 24 zeros

Problems

- Microsoft Common Document Format (CDF) files seem to occupy a whole number of sectors and so this technique won’t work with them.
- We get most confidence if the file ends with a non zero value.

Techniques

- Technique 1
  – Recover file system activity to identify file cluster layout.
- Technique 2
  – Reliably determine cluster size and the data area.
- Technique 3
  – Identify the end of files which do not occupy a whole number of sectors

Joining Directory Clusters

- A 4k cluster has room for 32 directory entries.
- A file usually requires more than 1 entry, in fact, tests show that typically a file requires 3.4 entries and so one cluster can hold 9 of these typical files.
- When a file is created and there isn't enough room in the directory a new cluster is added.

Long FileName entries

- Every file needs a base directory entry
- When a file is created which requires a long file name, then special entries are created (Long FileName or LFN entries).
- Tests show that 2.4 LFN entries are required for every file. Add the base directory entry to get 3.4
- The LFN entries have a checksum to tie them to the base directory entry.

Joining Directory Clusters

- The file directory entries may straddle a cluster.
  - We can then use the checksum to join the clusters together.
- We can also use common parent entries
- Failing that, we can use the file times in the directory to match

4 Techniques

- Technique 1
  - Recover file activity to identify file cluster layout.
- Technique 2
  - Reliably determine cluster size and the data area.
- Technique 3
  - Identify the end of files
- Technique 4
  - Joining up directory clusters

In Practice

Most files are not fragmented

- Simson Garfinkel used his huge corpus to determine that only 6% of files were fragmented. (http://simson.net/clips/academic/2007.DFRWS.pdf)
- We can quickly run through these and validate them and remove that part of the disk from further consideration
  - spend more time on the fragmented files.

Tying it all together

An Actual Example

- 4G FAT32 Flash key
- Originally formatted using XP
- ¾ filled with files
  - backed up, so mostly created
- Then formatted with OS X
  - same cluster size, but different data area
- A small number of file created occupying about 0.1% of the disk

Recovery

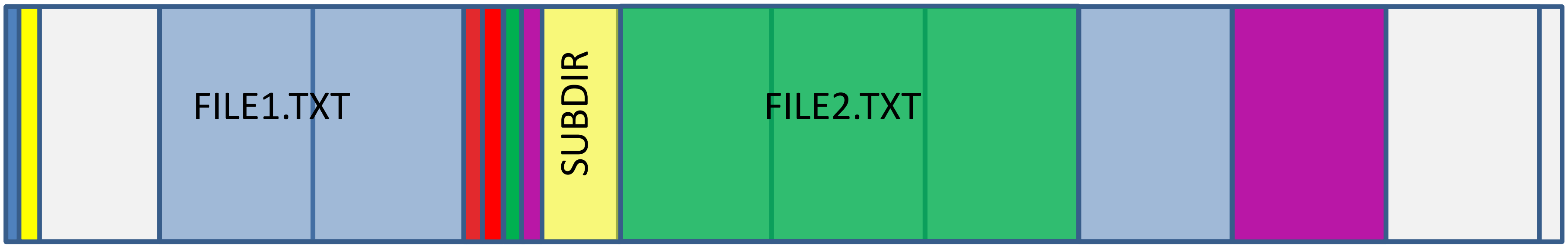
- We consider a block of files to be a set of files which are laid out sequentially
  - we can have more confidence that these files are not fragmented
- We recovered 2856 files occupying 1.6 GB bytes. The system found 65 file blocks with an average size of 61 files in a block. The largest block contained 424 files.
- Only the directories were fragmented

Efficiency

- This approach could be made very efficient. If you can verify a file by examining the last cluster, then you don’t need to examine every cluster on the disk.
- Every unfragmented file is an opportunity to avoid examining its clusters.

Exercise

Let’s go back in time



- FILE1.TXT is increased by one cluster
- FILE5.TXT (1 cluster is created)

Let’s go back in time

- FILE1.TXT is increased by one cluster
- FILE5.TXT (1 cluster is created)
- FILE2 is deleted
- When we look only at directory entries can we be sure that FILE1.TXT occupies those clusters

This?

Or this?

Code

- Code is at http://code.google.com/p/comebackfat/
- It is exploratory
- It is not efficient

Thank You