CSCI3150 Assignment 3

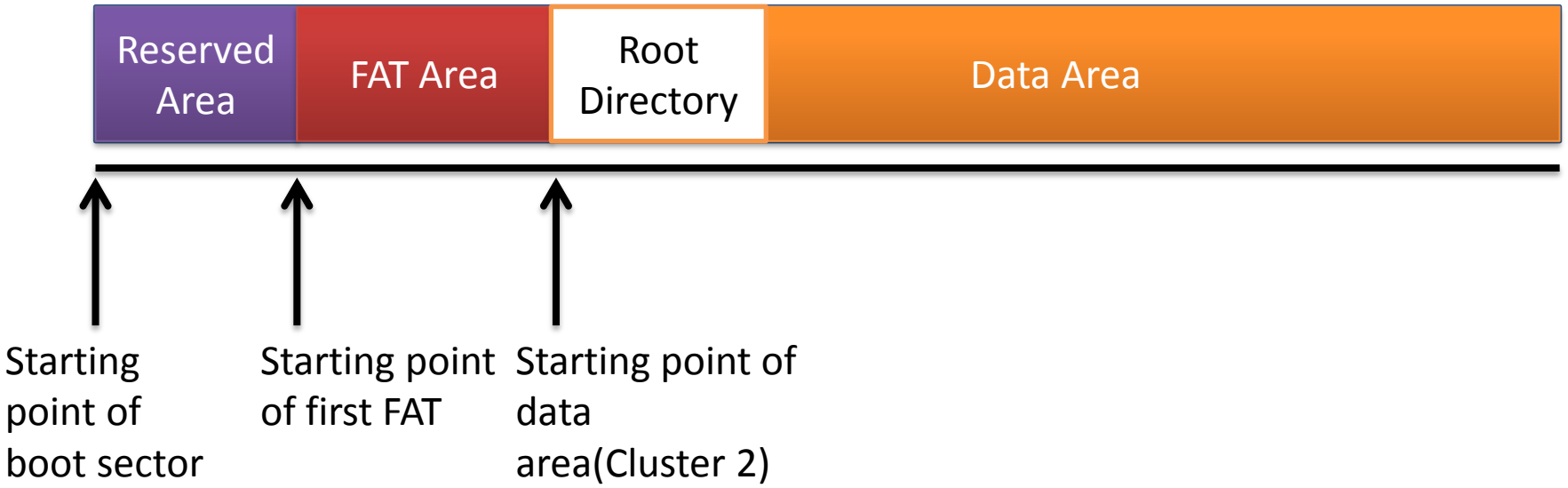
LI Runhui

Outline

- Structure of FAT32
  - Reserved Area
  - FAT Area
  - Data Area
- How to Recover

Structure of FAT32

- Three areas in FAT 32 file system



Reserved Area

- Boot Sector is Stored in This Area
  - It is very important in this assignment
  - A 90-byte pre-defined data structure
  - How to get it:
    - Define a same data structure
    - Read 90 bytes from the beginning of the file system
    - Now you can extract useful information from the data structure

Data Structure of Boot Sector

```
#pragma pack(push,1)
struct BootEntry {
    unsigned char BS_jmpBoot[3];    /* Assembly instruction to jump to boot code */
    unsigned char BS_OEMName[8];    /* OEM Name in ASCII */
    unsigned short BPB_BytsPerSec;  /* Bytes per sector. Allowed values include 512,
                                       1024, 2048, and 4096 */
    unsigned char BPB_SecPerClus;   /* Sectors per cluster (data unit). Allowed values
                                       are powers of 2, but the cluster size must be 32KB
                                       or smaller */
    unsigned short BPB_RsvdSecCnt;  /* Size in sectors of the reserved area */
    unsigned char BPB_NumFATs;      /* Number of FATs */
    unsigned short BPB_RootEntCnt;  /* Maximum number of files in the root directory for
                                       FAT12 and FAT16. This is 0 for FAT32 */
    unsigned short BPB_TotSec16;    /* 16-bit value of number of sectors in file system */
    unsigned char BPB_Media;        /* Media type */
    unsigned short BPB_FATSz16;     /* 16-bit size in sectors of each FAT for FAT12 and
                                       FAT16. For FAT32, this field is 0 */
    unsigned short BPB_SecPerTrk;   /* Sectors per track of storage device */
    unsigned short BPB_NumHeads;    /* Number of heads in storage device */
    unsigned long BPB_HiddSec;      /* Number of sectors before the start of partition */
    unsigned long BPB_TotSec32;     /* 32-bit value of number of sectors in file system.
                                       Either this value or the 16-bit value above must be
                                       0 */
```

Data Structure of Boot Sector

```
    unsigned long BPB_FATSz32;       /* 32-bit size in sectors of one FAT */
    unsigned short BPB_ExtFlags;     /* A flag for FAT */
    unsigned short BPB_FSVer;        /* The major and minor version number */
    unsigned long BPB_RootClus;      /* Cluster where the root directory can be
                                        found */
    unsigned short BPB_FSInfo;       /* Sector where FSINFO structure can be
                                        found */
    unsigned short BPB_BkBootSec;    /* Sector where backup copy of boot sector is
                                        located */
    unsigned char BPB_Reserved[12];  /* Reserved */
    unsigned char BS_DrvNum;         /* BIOS INT13h drive number */
    unsigned char BS_Reserved1;      /* Not used */
    unsigned char BS_BootSig;        /* Extended boot signature to identify if the
                                        next three values are valid */
    unsigned long BS_VolID;          /* Volume serial number */
    unsigned char BS_VolLab[11];     /* Volume label in ASCII. User defines when
                                        creating the file system */
    unsigned char BS_FilSysType[8];  /* File system type label in ASCII */
};
#pragma pack(pop)
```

FAT Area

- Maybe More Than 1 FATs
- How to Access the 1st FAT?
  - Remember the struct BootEntry?
  - Locates behind the reserved area
  - Use the information you extracted from boot sector
- How to Access the n-th FAT?
  - Locate right behind the (n-1)-th FAT

Data Area

- Locates right behind the FAT area
- Cluster: basic data unit of data area
  - Contiguous collection of sectors
- In an FAT 32 file system, there is no Cluster 0 and Cluster 1. The first Cluster in data area is Cluster2.

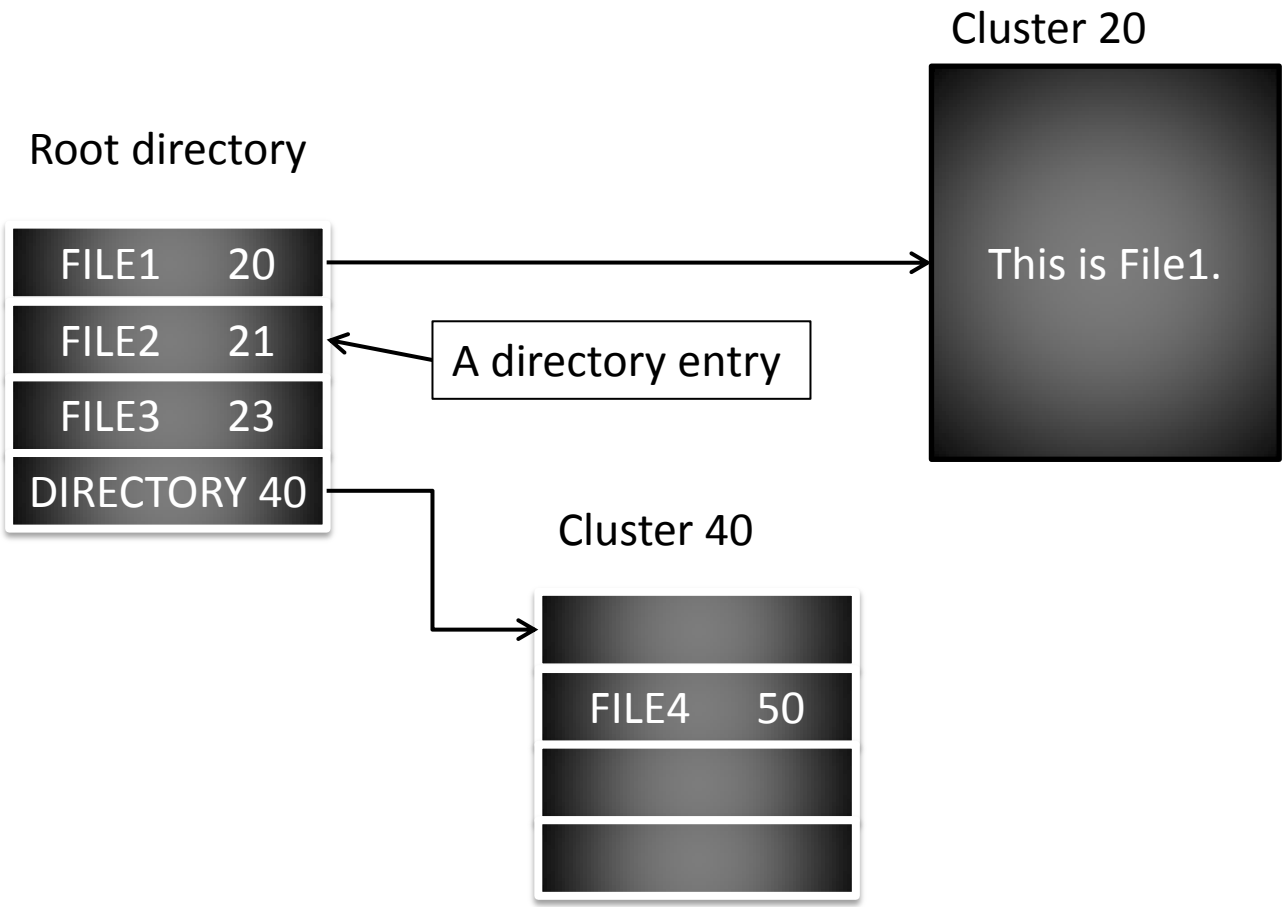
Root Directory

- How to Access Root Directory
  - In boot sector, BPB_RootClus shows where the root directory locates.
  - e.g. if BPB_RootClus=2, then, root directory locates in cluster 2
  - Since you know the number of FATs and the size of each FAT (can get from struct BootEntry), you can locate the root directory
  - Usually locates at cluster 2

Directory

- Consist of some directory entries
- Directory Entry:
  - Each directory entry represents a file/subdirectory in this directory.
  - Containing some Information about file name, file length, the beginning cluster, etc.

Directory Entry



Directory Entry

- Actually, Directory Entry Contains More Information.
  - It is a pre-defined Data Structure(like the boot sector)
  - Define a same data structure in your program
  - Get the directory entry from the file system
  - Now you can extract some useful information about the file.

Directory Entry

```
#pragma pack(push,1)
struct DirEntry
{
    unsigned char DIR_Name[11];     /* File name */
    unsigned char DIR_Attr;         /* File attributes */
    unsigned char DIR_NTRes;        /* Reserved */
    unsigned char DIR_CrtTimeTenth; /* Created time (tenths of second) */
    unsigned short DIR_CrtTime;     /* Created time (hours, minutes, seconds) */
    unsigned short DIR_CrtDate;     /* Created day */
    unsigned short DIR_LstAccDate;  /* Accessed day */
    unsigned short DIR_FstClusHI;   /* High 2 bytes of the first cluster address */
    unsigned short DIR_WrtTime;     /* Written time (hours, minutes, seconds */
    unsigned short DIR_WrtDate;     /* Written day */
    unsigned short DIR_FstClusLO;   /* Low 2 bytes of the first cluster address */
    unsigned long DIR_FileSize;     /* File size in bytes. (0 for directories) */
};
#pragma pack(pop)
```

Directory Entry

- File name
  - We only deal with short name
  - At most 8 bytes file name and 3 bytes extension
  - Only capital letters
  - Spaces other than string-terminal('\0')
  - Please refer to the specification for more information

FILE.EXT

Directory Entry

- Directory Attributes(DIR_Attr). Every bit has its meaning
  - 0000 0001 Read-only
  - 0000 0010 Hidden
  - 0000 0100 System
  - 0000 1000 Volume Label
  - 0001 0000 Directory
  - 0010 0000 Archival
  - 0000 1111 Long file Name
- e.g. a read-only hidden directory
  - (0x01)|(0x02)|(0x10) = (0x13) =$(0001\ 0011)_2$

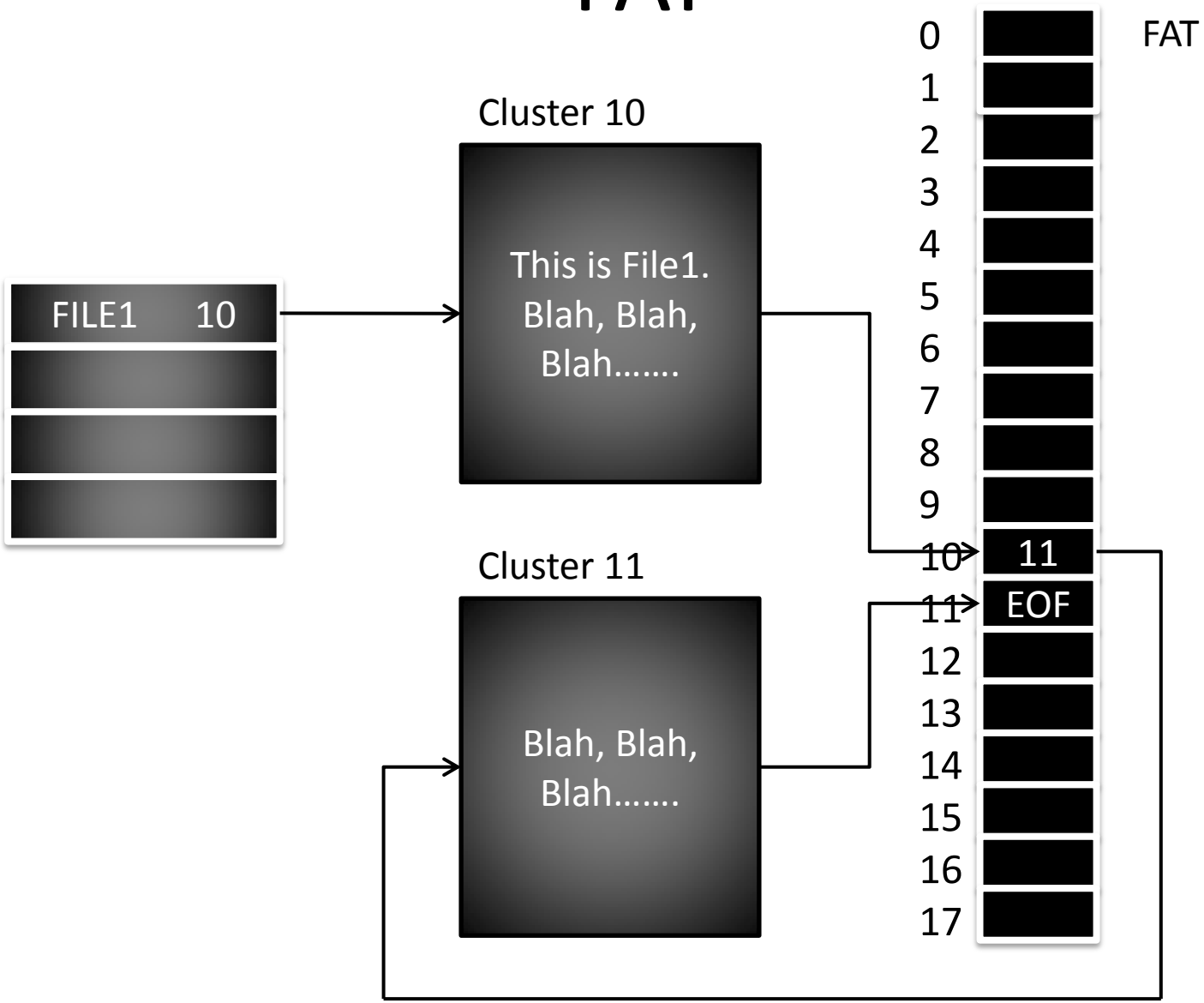
What if Large File…

- Large file means that the file size is larger than the size of one cluster
- The directory entry only tell the us the address of the first cluster
- Solution: using FAT(File Allocation Table)

FAT

- File Allocation Table
  - Can be viewed as an integer array of cluster index
  - Every entry in FAT is a 32-bit element
  - Shows where the next cluster locates
    - e.g. If FAT[10]=11, it means that the index of the next cluster is 11.
  - FAT[0] and FAT[1] is useless for this assignment. Just start from FAT[2].

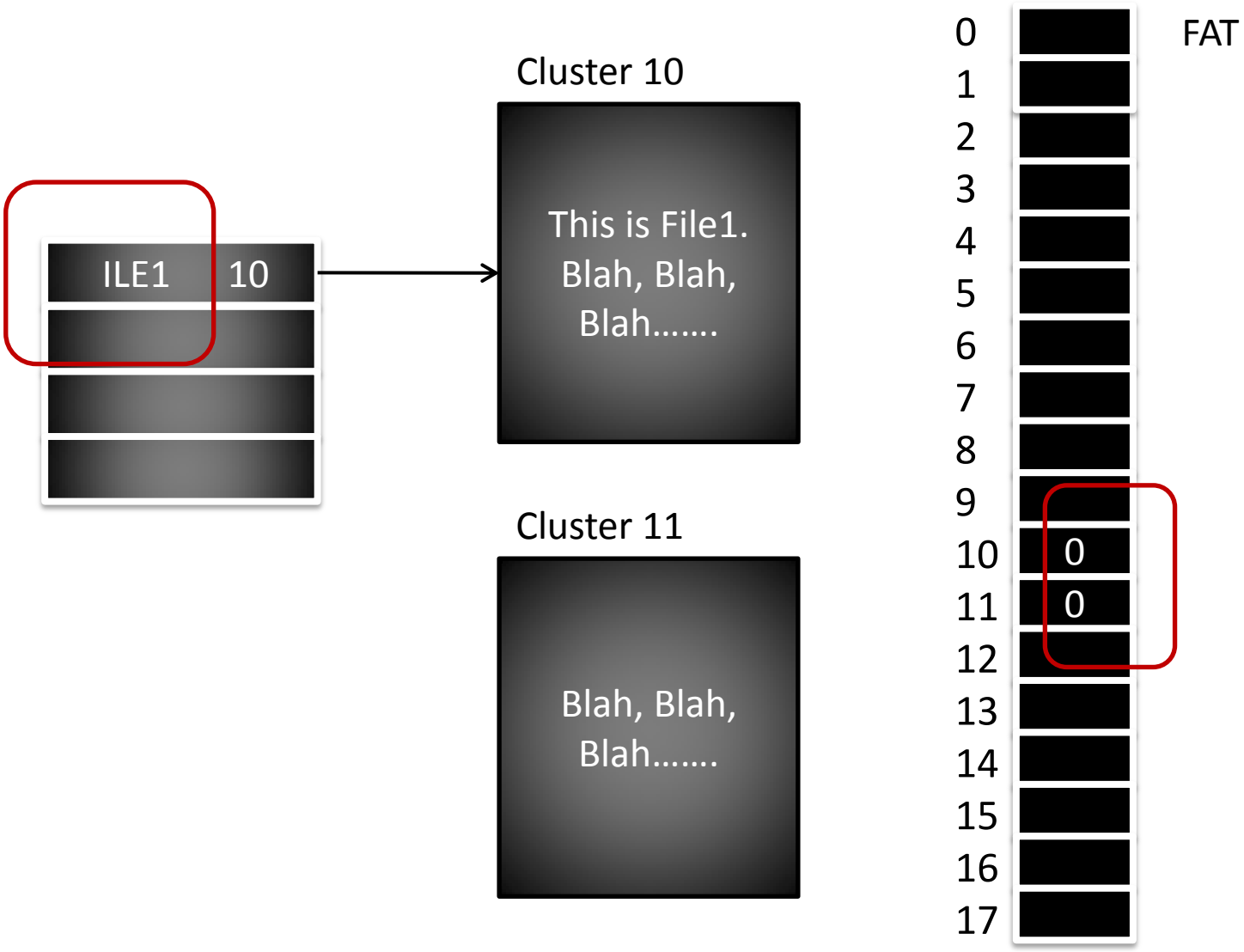
FAT


FILE1 10
Cluster 10
This is File1.
Blah, Blah,
Blah…….
Cluster 11
Blah, Blah,
Blah…….
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
FAT
11
EOF

What Happens After a Deletion

- Directory Entry
  - The first letter of the file name will be changed to 0xe5
- FAT
  - For every cluster c in the deleted file. FAT[c] is set to 0
- Everything else will stay the same: file size, first cluster, DIR_Attr, etc.

After Deletion



How to Do the Recovery

- Undo the Changes—File Name
  - The first letter of the file name is changed.
  - Change it back!
  - File name is provided by the user, so you need to go through all the entries in the root directory. For every entry(except subdirectories and long file names), compare the file name.
  - When you compare, remember do not compare the first letter.

How to Do the Recovery

- Undo the Changes—FAT
- After deletion, all involving FAT entries are changed to 0. So, just change them back.
  - For Small Files (smaller than 1 cluster)
    - Just change the corresponding entry in FAT to EOF

How to Do the Recovery

- Undo the Changes—FAT
  - For large files
    - Only consider files locating in the contiguous clusters.
    - e.g. A file of two clusters with the starting cluster to be cluster 10, then the two clusters are cluster 10 and cluster 11.
    - After deletion. FAT[10]=0 and FAT[11]=0. Change it back to FAT[10]=11 and FAT[11]=EOF.

How to Do the Recovery

- Undo the Changes—FAT
- Remember to Modify All the FATs.
  - Actually, it seems OK to only modify the first FAT.

How to Do the Recovery

- Undo the Changes—File Name(Advanced)
  - There may be more than one file names that match the provided name.
  - e.g. ABC.A and BBC.A will all be changed to (0xe5)BC.A
  - Your program should be able to detect ambiguous file names
  - Further, you can use MD5 value to decide which file is the expected file.

How to Do the Recovery

- Undo the Changes—File Name(Advanced)
  - MD5 value is also provided by the user
  - For every candidate, read the content into a buffer, calculate the MD5 value and compare the value with the value provided by the user.
  - # include <openssl/md5.h>
  - MD5(char* str, int length, char* result);

Our Test Cases will be Simple

- Only regular files in the root directory
- Will not use long file name
- You do not need to change FSINFO in the boot sector

Some Hints

- When test your program. DO NOT
  - Mount the file system
  - Write the file
  - Remove the file
  - Unmount the file system
  - Run your program for recovery.
- This will cause some strange problems.

Some Hints

- The correct order should be:
  - Mount the file system
  - Write the file
  - Unmount the file system
  - Mount the system
  - Remove the file
  - Unmount the file system
  - Run your program for recovery.

Some Hints

- I know some of you may be interested in making your OS special. But …
  - If you use a 64-bit OS, things may not work out. The unsigned long is 64-bit instead of 32-bit. So you may not be able to correctly read the boot sector and the directory entries.
  - If you have compiled the kernel by yourself. There may be some problem.(It happens to me, actually)
  - Just use the VMs, they should be fine.

DEMO

- That’s it! Have fun!

Let’s ROCK!!