Project 3:
Directories

COP 4610 / CGS 5765

Principles of Operating Systems

Outline

- File/Directory Names
- Creating Directories
- Undelete
- Problem of orphaned data
- File removal walkthrough
- Directory removal walkthrough

File/Directory Names

- **<main_part (8 bytes)>.<extension (3 bytes)>**
  - Each part is padded with trailing spaces
- **DIR_Name[0]** may not equal **0x20**
- There is an implied "." character between the main part of the name and the extension part of the name that is not present in DIR_Name
- Lower case characters are not allowed in DIR_Name (i.e., in the record entry)
- Cannot contain
  - spaces (0x20)
  - . (0x2E)
  - …

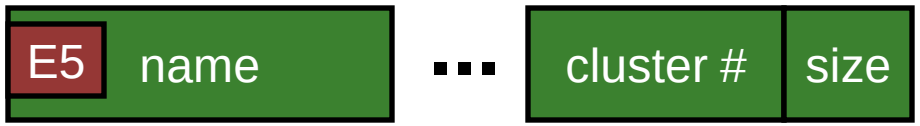
Creating Directories

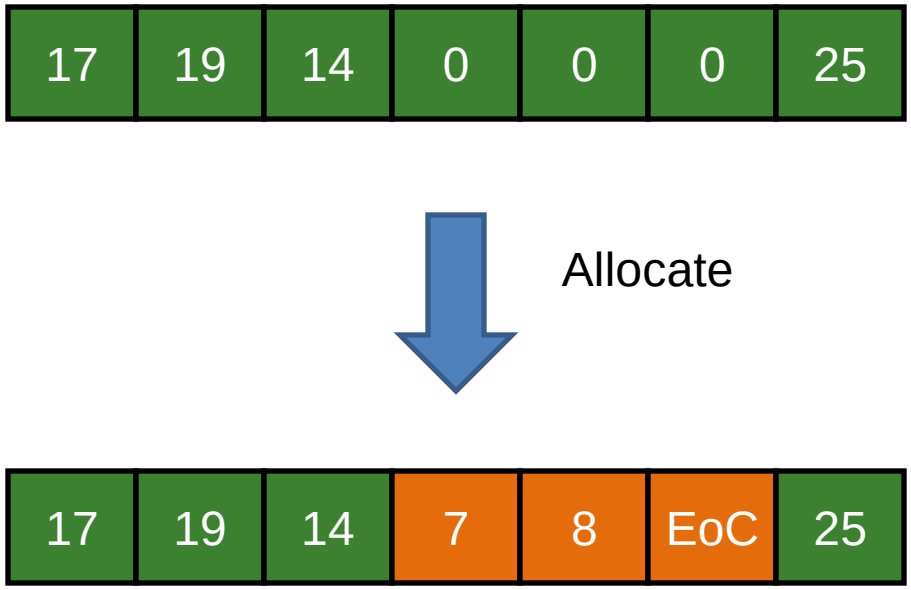
- **`ATTR_DIRECTORY`** bit set
- **`DIR_FileSize`** set to 0
- Allocate one cluster
  - Set **`DIR_FstClusLO`** and **`DIR_FstClusHI`**
  - EOC mark in FAT
- Create *dot* and *dotdot* special entries
  - *dot* – points to itself
  - *dotdot* – points to starting cluster of parent
  - root directory does not contain *dot* and *dotdot* files as the first two directory entries in the directory

UNDELETE

Undelete

- Find information not overwritten
- Example:
  - Find records marked with 0xE5
    - Cluster number
    - Size
    - Partial file name

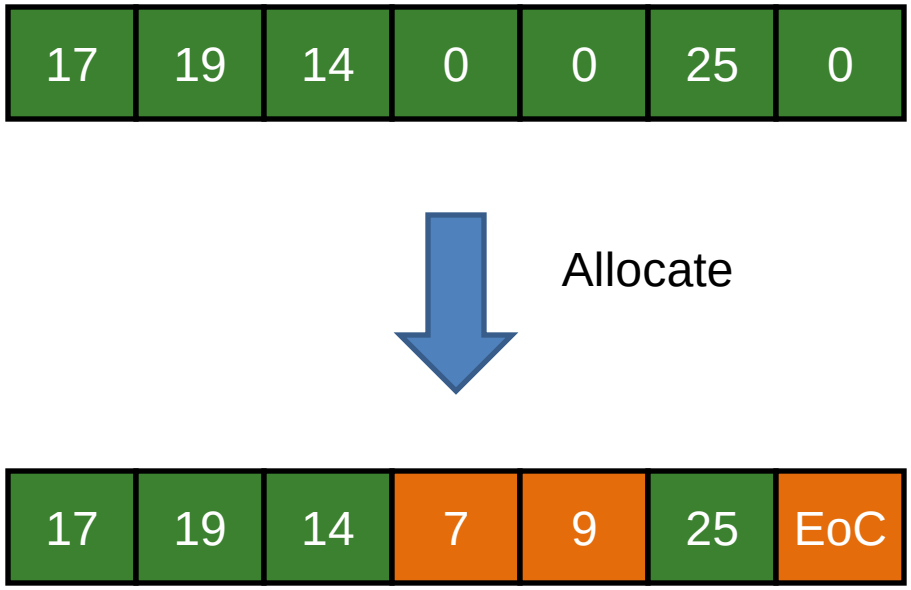



17
19
14
0
0
0
25
Allocate
17
19
14
7
8
EoC
25

Undelete

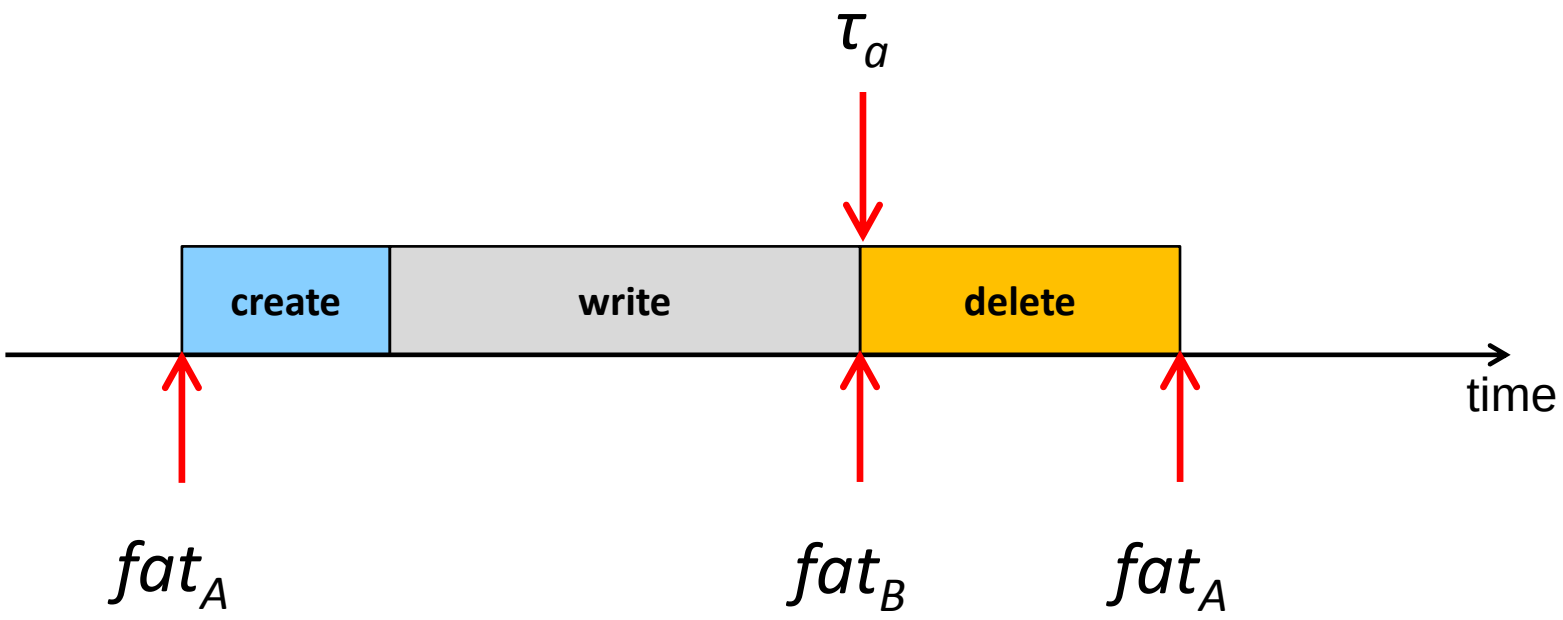
- Recover first cluster
- Subsequent clusters
  - Based on file size
  - Return next unallocated clusters
- Allocated clusters are very likely not part of the deleted file

Undelete



Example Sequence

fat_X – state of FAT



Undelete

- System state at τ_a
  - File data available
- What is deleted?
  - Assume
    - Cluster data is not deleted
    - FAT entries corresponding to file clusters are marked as free
    - First byte of directory record is overwritten with 0xE5
- File's data clusters
    - Set of clusters
      - Difference between fat_A and fat_B
    - Cluster order (linked list)
      - Same as allocation

ORPHANS

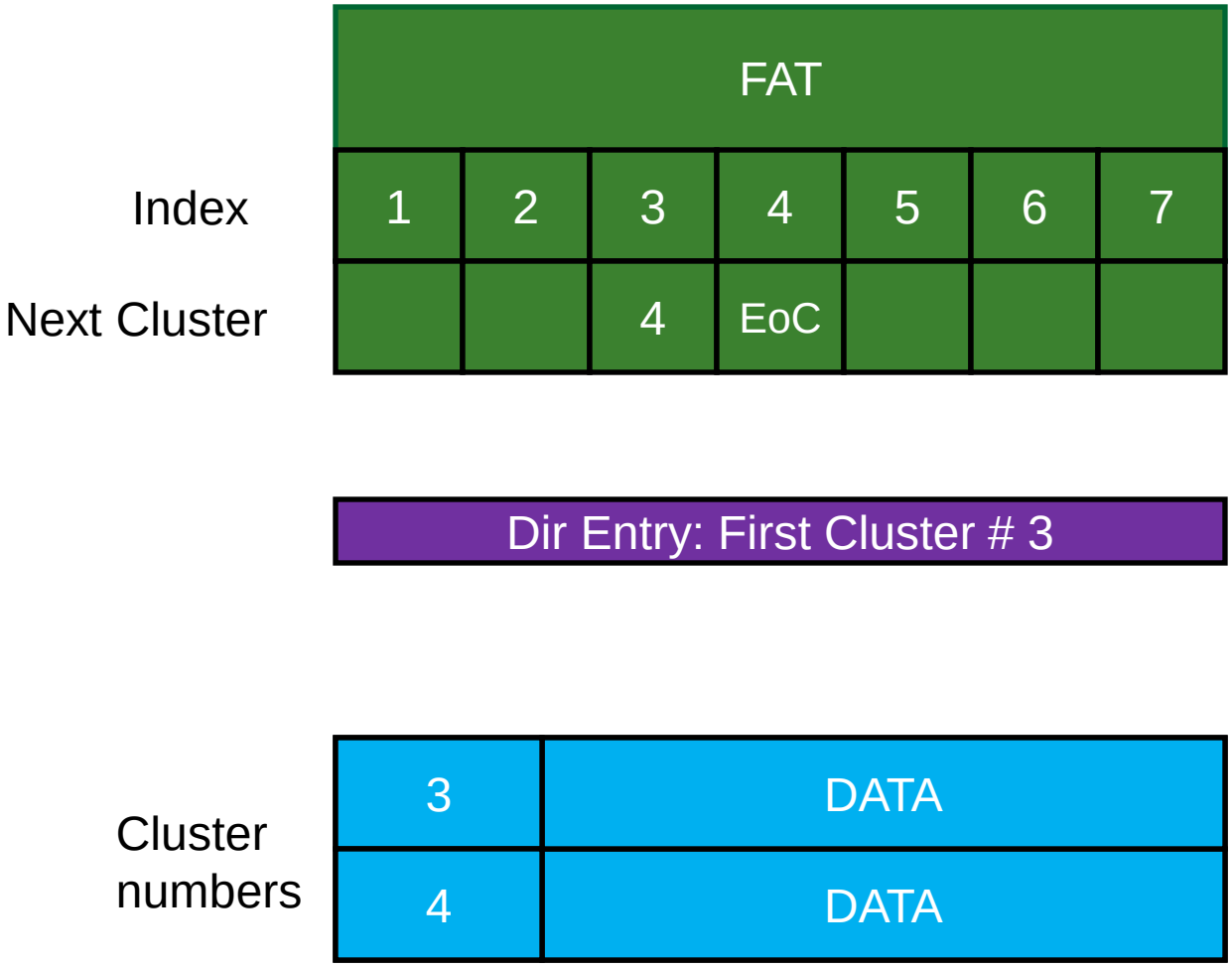
What is orphaned data?

- Orphaned data – data marked as valid but unreached through standard filesystem operations
- How could this ever happen?

Orphan Example

- Suppose we want to delete a file
- It has
  - *directory entry* with a first cluster number
  - Data clusters
  - FAT entries

Orphan Example

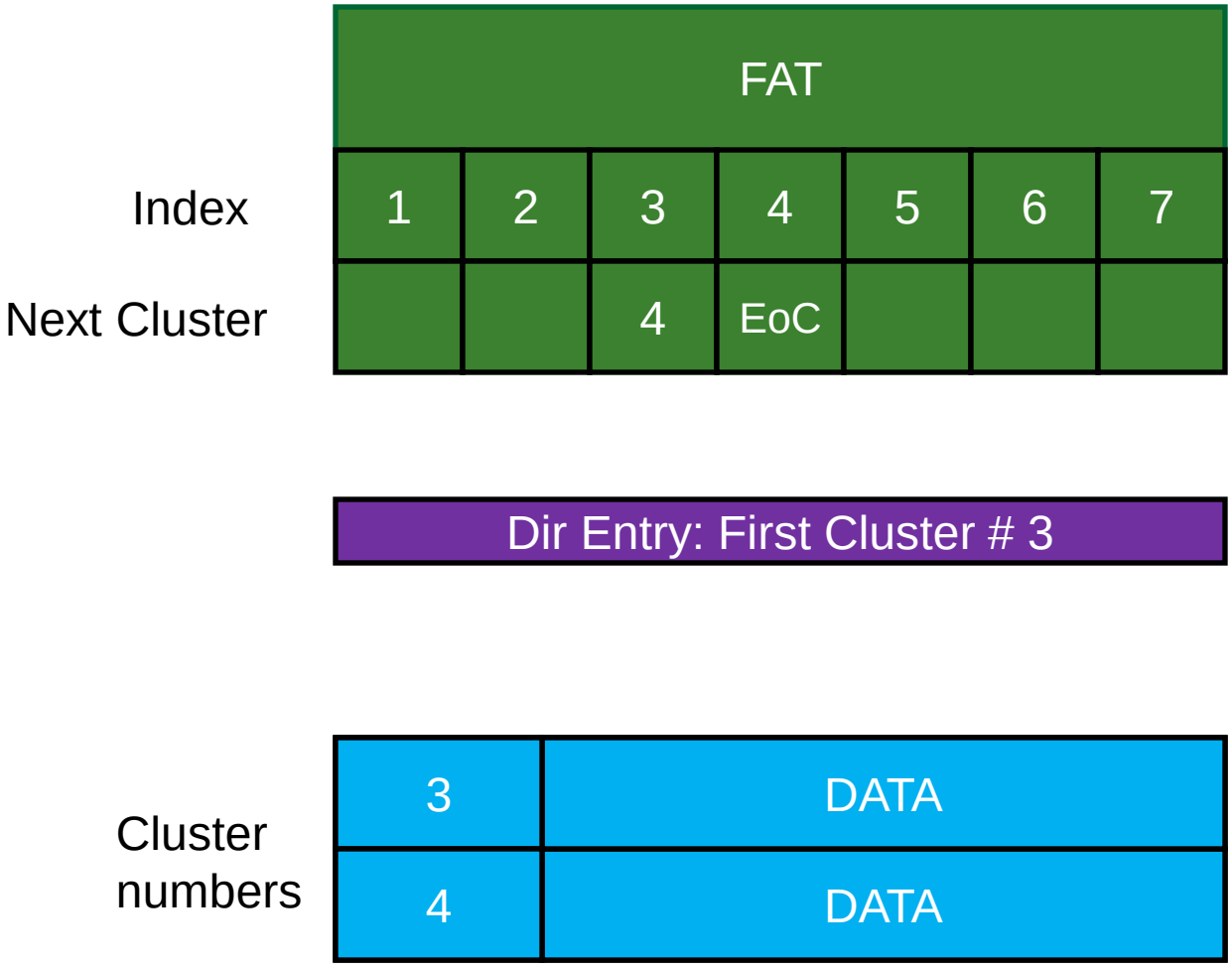


Our sample file starts at cluster 3 and continues to cluster 4.

Orphan Example

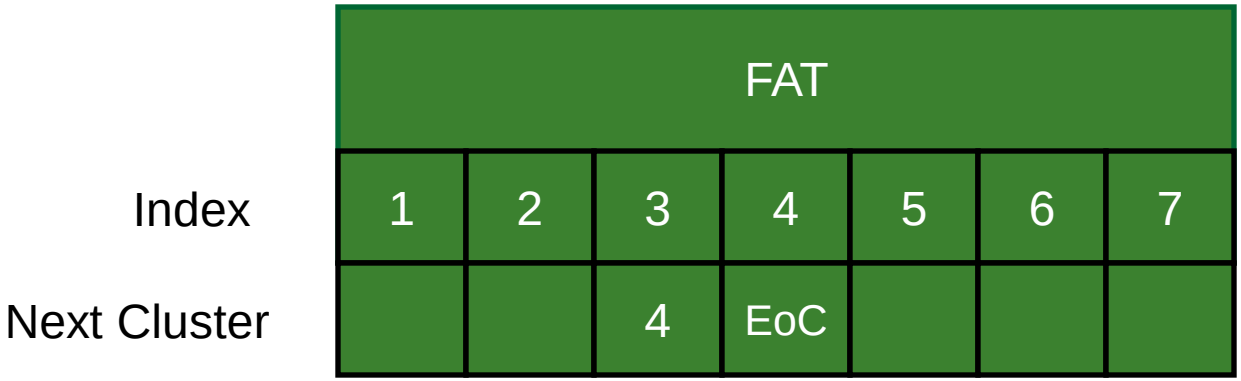
- Locating a file's contents begins by reading its directory entry contents
  - What if we start deleting there?

Orphan Example



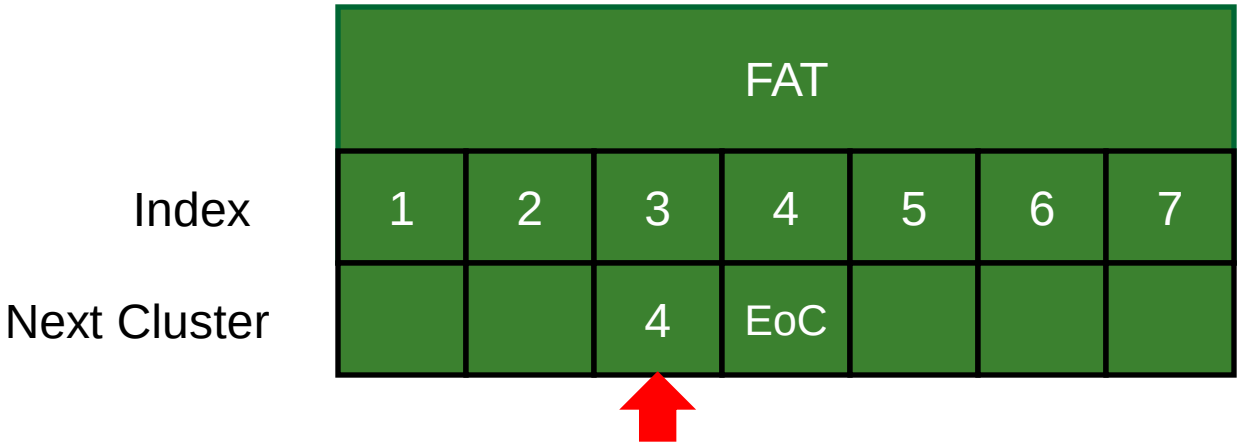
Step 1: Read the file's first cluster number into memory.

Orphan Example



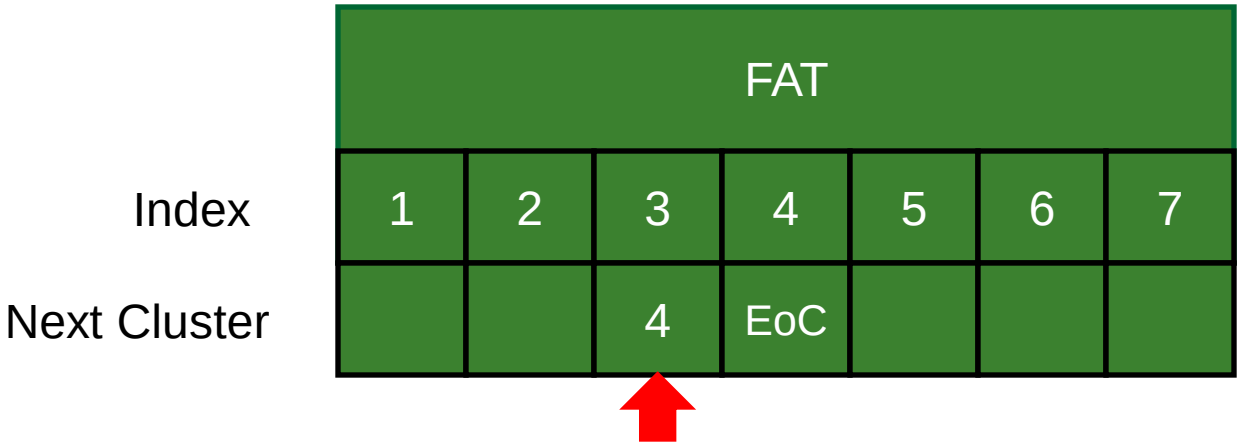
Step 2: Delete the file's directory entry.

Orphan Example



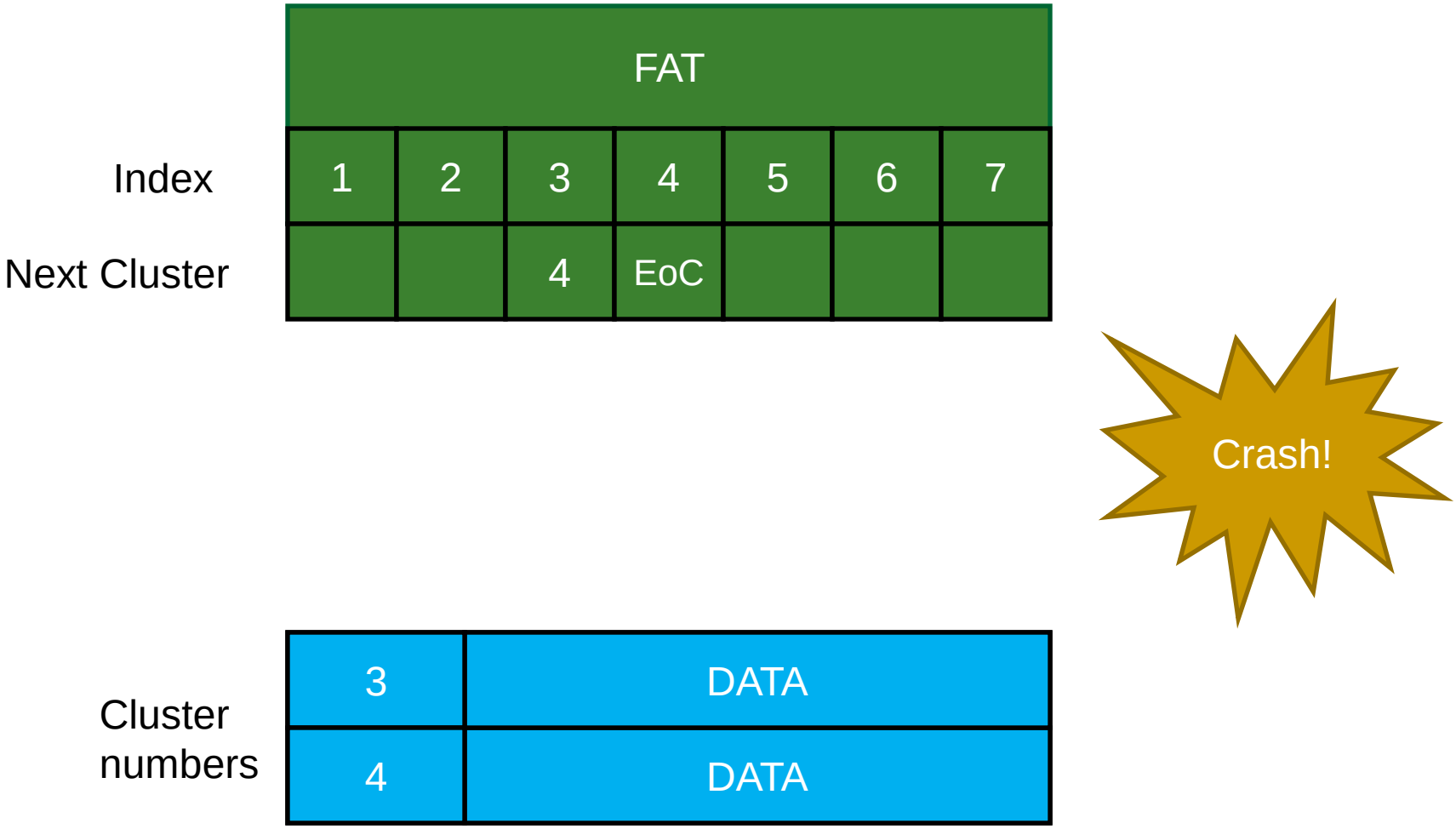
Step 3: Look up cluster 3 in the FAT.

Orphan Example



Step 4: Read the file's next cluster number into memory (4).

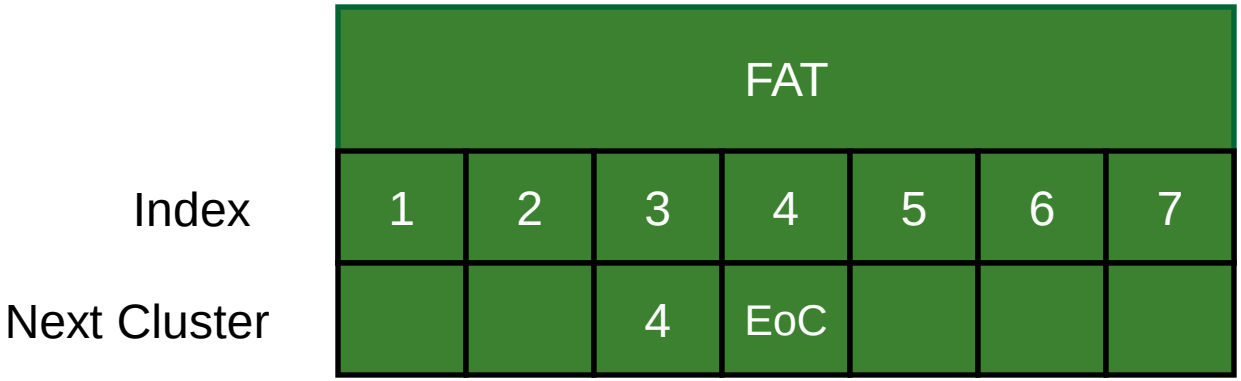
Orphan Example



Orphan Example

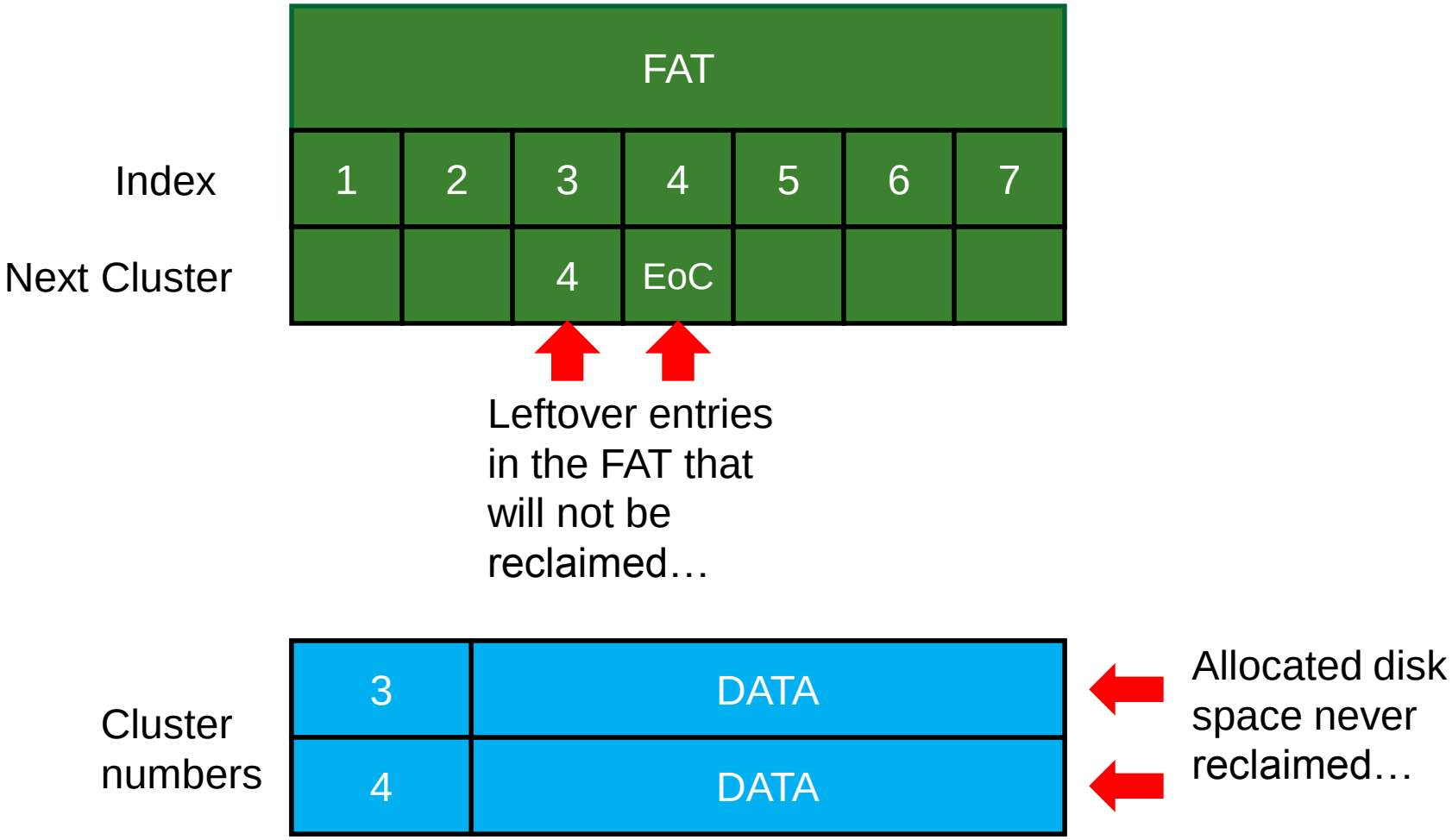
- System crashed requiring a reboot
- Is the data we were deleting reachable through filesystem operations?

Orphan Example



No starting point to reach remaining data of delete operation…

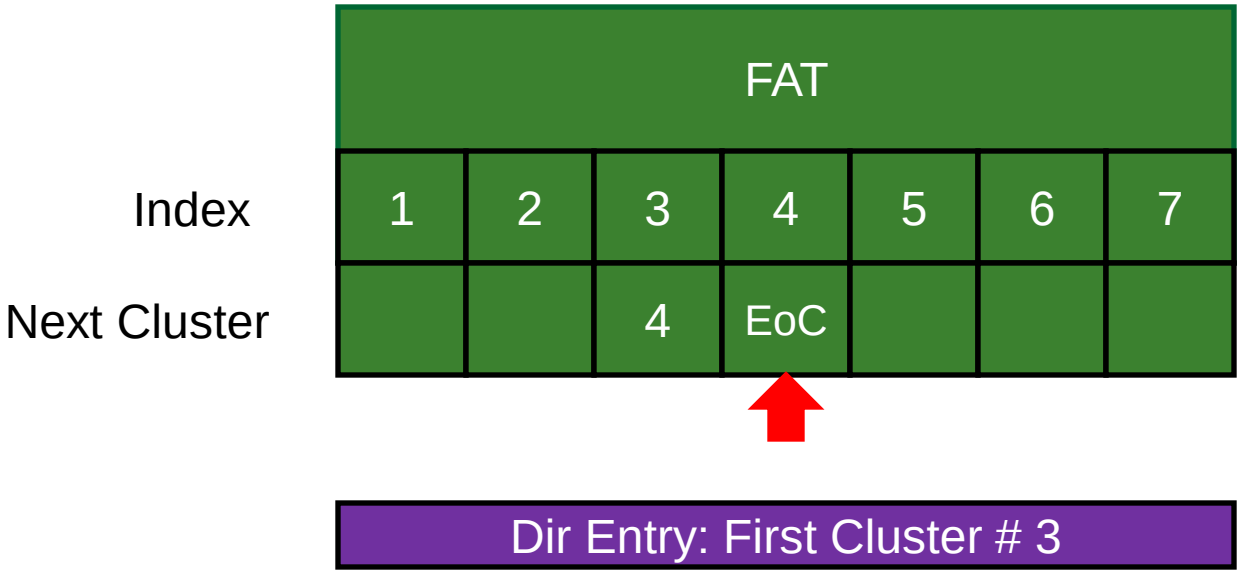
Orphan Example



Preventing Orphans

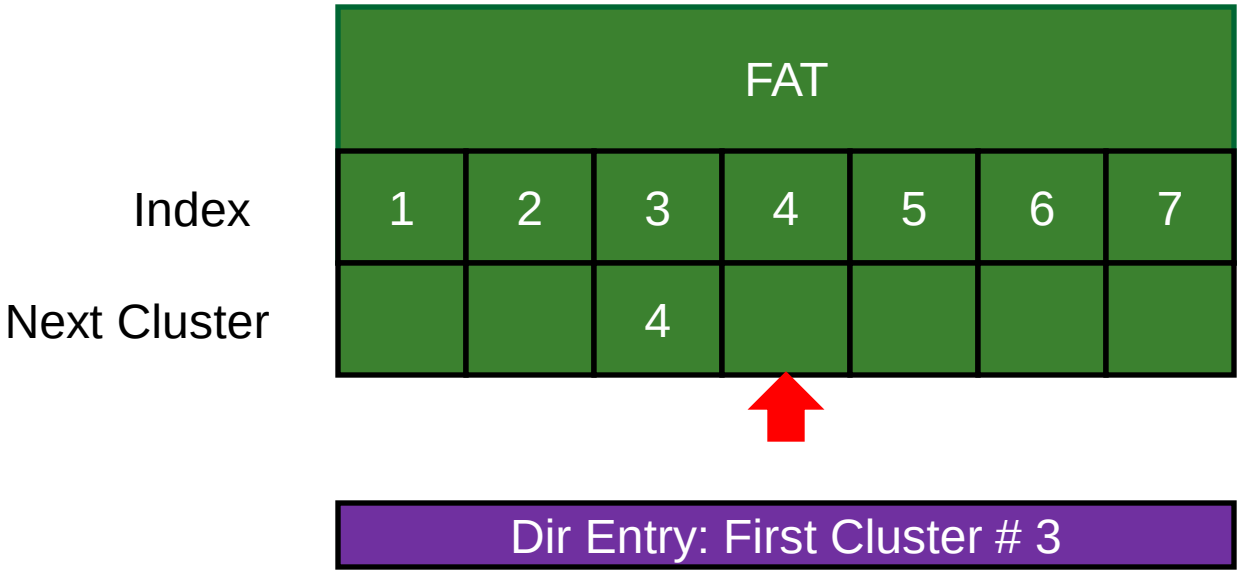
- How can we avoid the chance of orphans from a delete?
- Answer: delete starting with the end of the linked list (backwards)!

Preventing Orphans



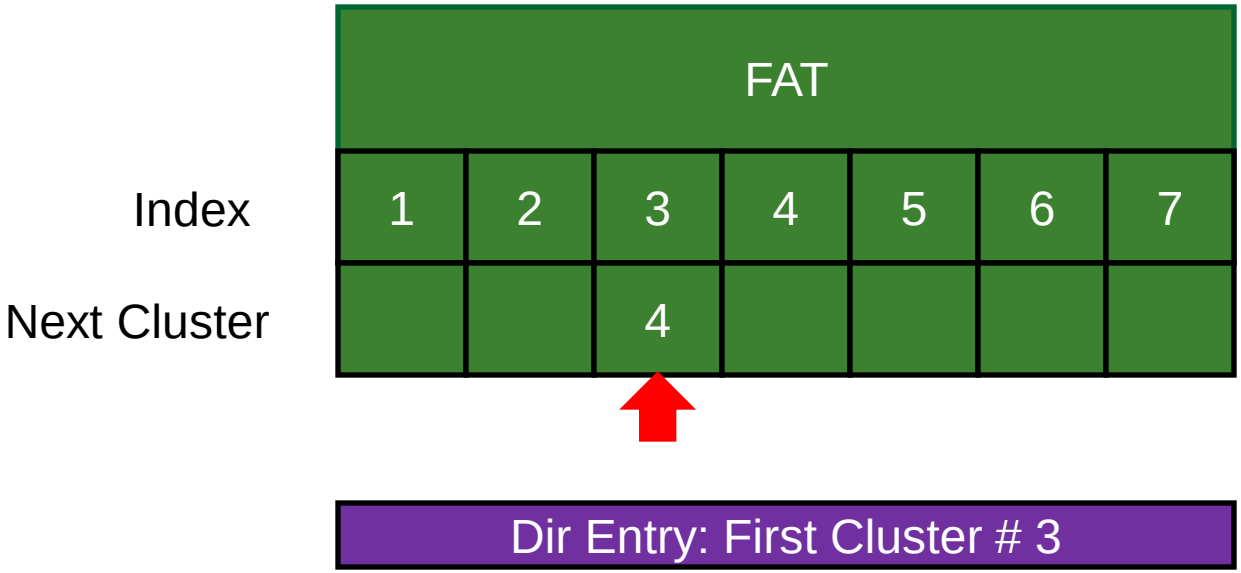
Step 1: Read through entire file until we find the last cluster entry for the file in the FAT

Preventing Orphans



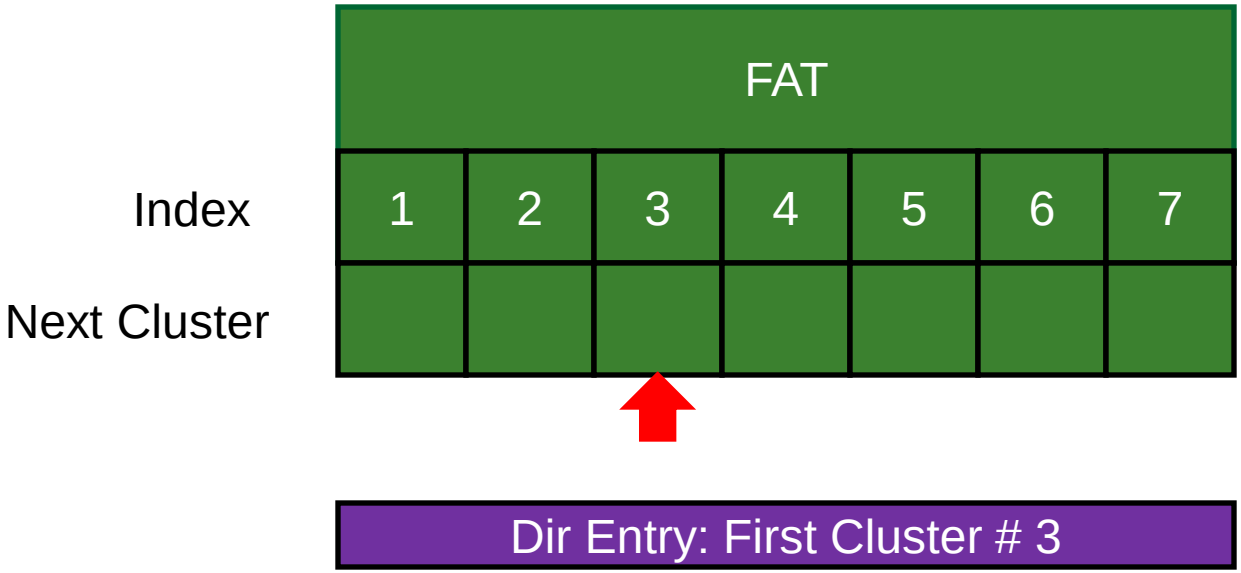
Step 2: Mark the last cluster as free. What happens if we crash here?

Preventing Orphans



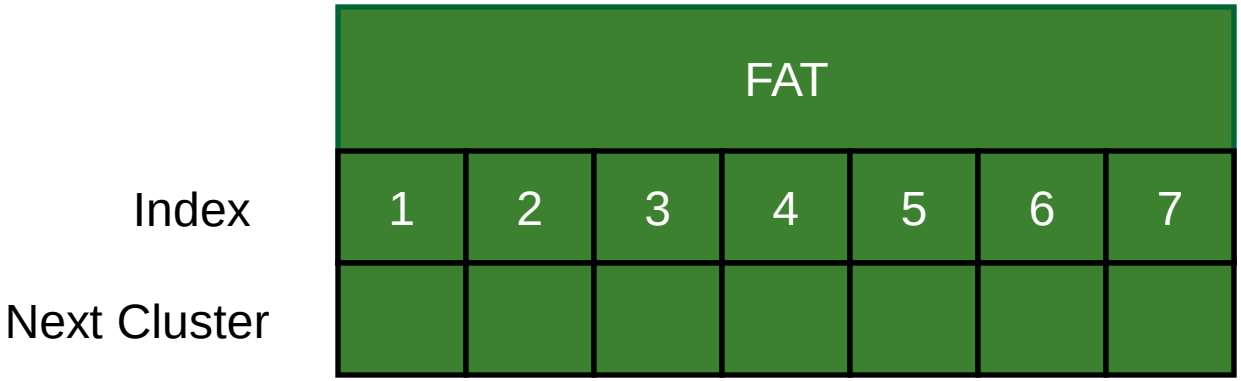
Step 3: Find the new last file cluster in the FAT.

Preventing Orphans



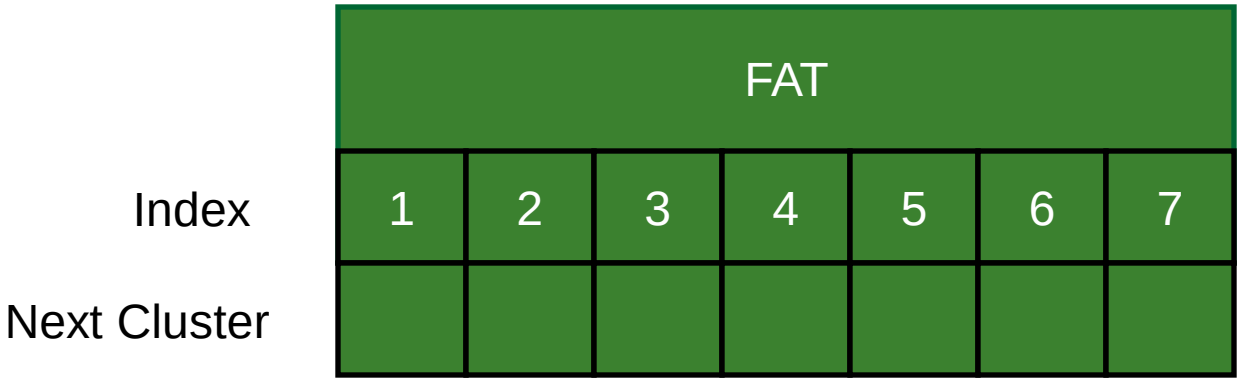
Step 4: Mark the last cluster as free.

Preventing Orphans



Step 5: Finally, if all the FAT entries for the file are marked free, delete the directory entry.

Preventing Orphans



Why don't we zero out the file's data?

File Data Leftovers

- Most file systems only update metadata upon deletion and leave old data as it was. Why?
  - Old data will just be overwritten later
  - Not accessible through filesystem operations
  - It can take a significant amount of time to zero over large amounts of file data
  - May cause extra wear on the device

File Data Leftovers

- File recovery utilities leverage this situation
  - Scans the file system for data clusters that are not currently allocated

FILE DELETION

File Deletion : rm

1. Check that the file to removed is a file and does exist
   - Cannot use this utility command to delete a directory
2. Seek to the last cluster entry in the FAT
3. Mark the last cluster entry in the FAT with the free mark of 0x00000000
4. Repeat 2 and 3 until there are no more cluster entries in the FAT
5. Delete the file's directory entry

Deleting a Directory Entry

- Can just mark the first byte in the directory entry to symbolize deletion
  - If DIR_Name[0] == 0xE5, then the directory entry is free (no file or directory name in this entry)
  - If DIR_Name[0] == 0x00, then the directory entry is free (same as for 0xE5), and there are no allocated directory entries after this one

rm Use Cases

- Successful rm

```
/DIR/] ls
. .. CONST.TXT EMPTY.TXT HELLO.TXT
/DIR/] rm HELLO.TXT
/DIR/]
```

- Unsuccessful rm

```
/DIR/] rm NOTHERE.TXT
Error: 'NOTHERE.TXT' does not exist
/DIR/]
```

DIRECTORY DELETION

Directory Deletion: rmdir

1. Check that directory to be removed is empty and is actually a directory
2. Go to step #2 for rm
   - Rest of directions just like deleting a file!

rmdir Use Cases

- Successful rmdir

```
/DIRS/] rmdir A
/DIRS/]
```

- Unsuccessful rmdir

```
/DIRS/] rmdir B
Error: directory not empty
/DIRS/]
```

rmdir Use Cases

- Unsuccessful rmdir

```
/DIR/] cd ..
/] rmdir FATINFO.TXT
Error: not a directory
/]
```