Operating Systems

File Systems

Prof. Dr. Oliver Hahm

Frankfurt University of Applied Sciences
Faculty 2: Computer Science and Engineering
oliver.hahm@fb2.fra-uas.de
https://teaching.dahahm.de

January 30, 2024

Agenda

Agenda

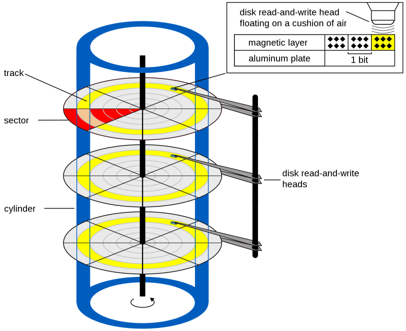
Hard Disk Drives

- HDDs are approx. 100 times less expensive per bit versus main memory and they offer approx. 100 times more capacity
  - **Drawback:** Accessing data on HDDs is approx. 1000 times slower
- Reason for the poorer access time:
  - HDDs are mechanical devices
    - They contain one or more discs, rotating with 4200, **5400**, **7200**, 10800, or 15000 revolutions per minute (RPM)
- For each side of each disc (**platter**), an arm exists with a **read-and-write head**
  - The **read-and-write head** is used to detect and modify the magnetization of the material
  - The distance between disk and head is approx. 20 nm ($20 * 10^{-9} m$)
- Also, HDDs have a **cache** (usually $\leq$ 32 MB)
  - This cache buffers read and write operations

Logical Structure of Hard Disk Drives (1/2)

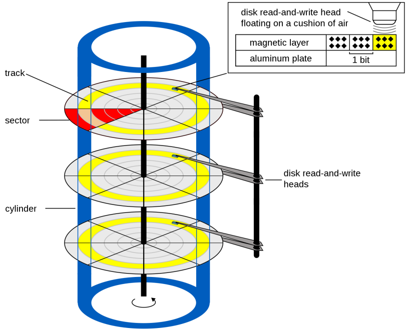
- The surfaces of the **platters** (discs) are magnetized in circular **tracks** by the heads
- All tracks on all disks at a specific arm position are part of a **cylinder**
- The tracks are divided into logical units (segments of a circle), which are called **blocks** or **sectors**
    - Typically, a sector contains 512 bytes payload
    - Sectors are the smallest addressable units of HDDs

Image source (structure): Henry Mühlpfordt. Wikimedia (CC-BY-SA-1.0)
Image source (HDD): `purepng.com` (CC0)


disk read-and-write head floating on a cushion of air
magnetic layer
aluminum plate
1 bit
track
sector
cylinder
disk read-and-write heads

Logical Structure of Hard Disk Drives (2/2)

- If data need be modified, the entire sector must be read and rewritten
- Today, software addresses **clusters**
  - Clusters are groups of sectors with a fixed size, e.g., 4 or 8 kB
  - In file systems of modern operating systems, clusters are the smallest addressable unit of HDDs


disk read-and-write head floating on a cushion of air
magnetic layer
aluminum plate
1 bit
track
sector
cylinder
disk read-and-write heads

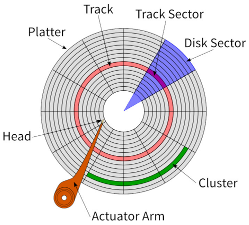


Image source (structure): Henry Mühlpfordt. Wikimedia (CC-BY-SA-1.0)
Image source (platter): Tim Bielawa. The Linux Sysadmins Guide to Virtual Disks (CC-BY-SA-4.0)

Logical Block Addressing (LBA)

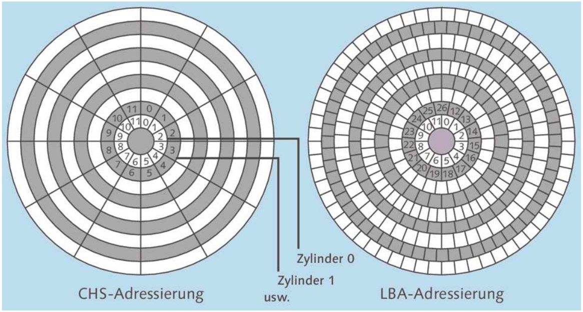


Image source

IT-Handbuch für Fachinformatiker. Sascha Kersken. 6th edition. Rheinwerk Verlag

- When CHS addressing is used, all tracks contain the same number of sectors
  - Each sector stores stores 512 bytes of payload
- Drawback: Storage capacity is wasted, because the data density decreases from the inner tracks to the outer tracks
- When LBA is implemented, this drawback does not exist

Required Time to access Data on HDDs

- The access time is an important performance factor
- Two factors influence the access time of HDDs
  1. Average Seek Time
     - The time that it takes for the arm to reach a desired track
     - Is for modern HDDs between 5 and 15 ms
  2. Average Rotational Latency Time
     - Delay of the rotational speed, until the required disk sector is located under the head
     - Depends entirely on the rotational speed of the disks
     - Is for modern HDDs between 2 and 7.1 ms

$$\text{Average Rot. Lat. Time [ms]} = \frac{1000\frac{[\text{ms}]}{[\text{sec}]} \times 60\frac{[\text{sec}]}{[\text{min}]} \times 0.5}{\frac{\text{revolutions}}{[\text{min}]}} = \frac{30{,}000\frac{[\text{ms}]}{[\text{min}]}}{\frac{\text{revolutions}}{[\text{min}]}}$$

Why does the equation contain 0.5 ?

Once the head has reached the right track, on average a half rotation of the disk must be waited for the correct sector to be under the head $\Longrightarrow$ Average Rotational Latency Time = half Rotational Latency Time

Solid State Drives (SSD)

- Sometimes called Solid State Disks
- Do not contain moving parts
- Benefits:
  - Fast access time
    - Data location is less important
  - Low power consumption
  - No noise generation
  - Mechanical robustness
  - Low weight
- Drawbacks:
  - Higher price compared with HDDs of the same capacity
  - Secure delete or overwrite is hard to implement
  - Limited number of program/erase cycles

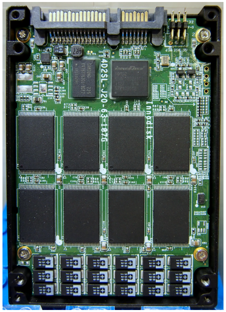
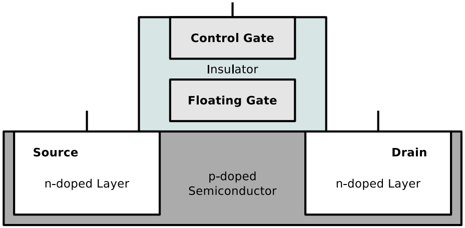


Image (SSD): Thomas Springer. Wikimedia (CC0)

Image (HDD): Erwan Velu. Wikimedia (CC-BY-SA-1.0)

Functioning of Flash Memory

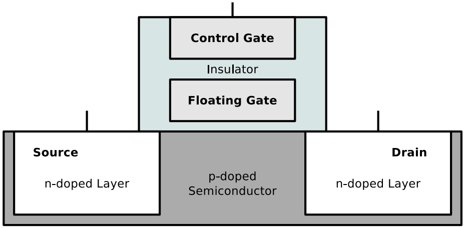
- Data is stored as electrical charges
- In contrast to main memory, no electricity is required to keep the data



- Each flash memory cell is a transistor and has 3 connectors
  - **Gate** = control electrode
  - **Drain** = electrode
  - **Source** = electrode
- The floating gate stores electrons (data)
  - Completely surrounded by an insulator
  - Electrical charge remains stable for years

Reading Data from Flash Memory Cells

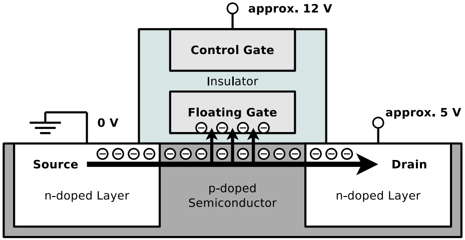
- A positively doped (p) semiconductor separates the 2 negatively doped (n) electrodes drain and source
  - Equal to a npn transistor, the npn passage is not conductive without a base current



- Above a certain positive voltage (5V) at the gate (threshold) a n-type channel is created in the p-area
  - Current can flow between source and drain through this channel
- If the floating gate contains electrons, the threshold is different
  - A higher positive voltage at the gate is required, so that current can flow between source and drain
    - This way the stored value of the flash memory cell is read out

Writing Data into Flash Memory Cells

- Data is stored inside flash memory cells by using Fowler-Nordheim tunneling



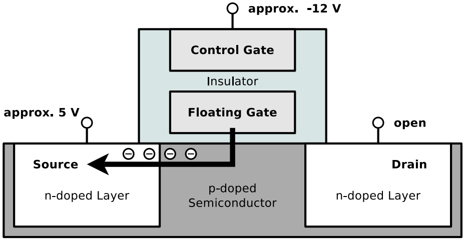
- A positive voltage (5V) is applied to the control gate
  - As a result, electrons can flow between source and drain
- If the high positive voltage is sufficient high (6 to 20V), some electrons are tunneled ($\Longrightarrow$ Fowler-Nordheim tunneling) through the insulator into the floating gate
- This method is also called Channel Hot Electron Injection

Recommended Source

Flash memory. *Alex Paikin*. 2004. `http://www.hitequest.com/Kiss/Flash_terms.htm`

Erasing Data in Flash Memory Cells

- For erasing a flash memory cell, a negative voltage (-6 to -20V) is applied at the control gate
    - As a result, electrons are tunneled in the reverse direction from the floating gate



- The insulating layer, which surrounds the floating gate, suffers from each erase cycle
    - At some point the insulating layer is no longer sufficient to hold the charge in the floating gate
    - For this reason, flash memory survives only a limited number of program/erase cycles

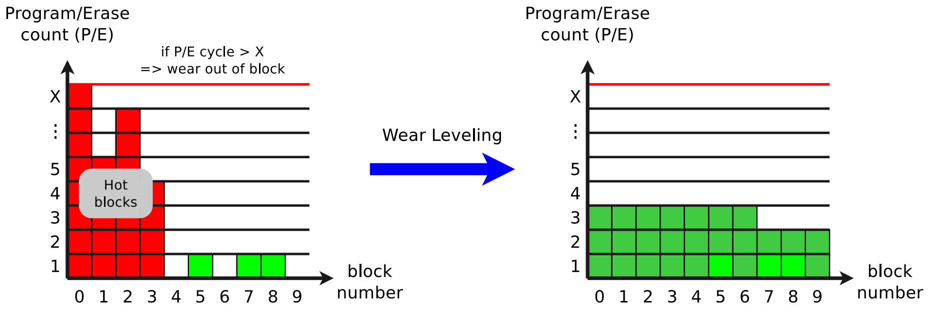
Functioning of Flash Memory

- Memory cells are connected to blocks and (depending on the structure also in) pages
  - A block always contains a fixed number of pages
  - Write/erase operations can only be carried out for entire pages or blocks
    $\Longrightarrow$ Write and erase operations are more complex than read operations
  - If data in a page need to be modified, the entire block must be erased
    1. To do this, the block is copied into a buffer memory (cache)
    2. Inside the cache, the data is modified
    3. Next, the block is erased from the flash memory
    4. Finally, the modified block is written into the flash memory
- 2 types of flash memory exist:
  - NOR memory (just blocks)
  - NAND memory (blocks and pages)

The circuit symbol indicates the way, the memory cells are connected

This influences the capacity and access time (latency)

Wear Leveling



- Wear leveling algorithms evenly distribute write operations
- File systems, which are designed for flash memory, and therefore minimize write operations, are e.g., JFFS, JFFS2, YAFFS and LogFS
  - JFFS contains its own wear leveling algorithm
    - This is often required in embedded systems, where flash memory is directly connected

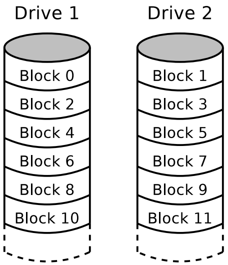
Agenda

Redundant Array of independent Disks (RAID)

- The performance of CPUs, cache, and main memory is growing faster than the data access time (*latency*) of HDDs and even SSDs
- The latency of **SSDs** is $\leq 1\,\mu$s $\Longrightarrow \approx$ factor 100 better than HDDs
- Further challenge
  - Storage drives can fail $\Longrightarrow$ risk of data loss
- Enhance latency and reliability of HDDs and SSDs $\Longrightarrow$ **RAID**
- One way to avoid the given limitations in terms of speed, capacity and reliability, is the parallel use multiple components
- A RAID consists of multiple drives (HDDs or SSDs)
  - For users and their processes, a RAID behaves like a single large drive
- Data is distributed across the drives of a RAID system
  - The RAID level specifies how the data is distributed
    - The most commonly used RAID levels are RAID 0, RAID 1 and RAID 5

RAID 0 – Striping – Acceleration without Redundancy

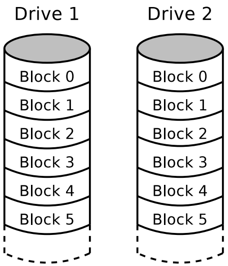
- No redundancy
  - Increases only the data rate
- Drives are partitioned into blocks of equal size
- If read/write operations are big enough ($>$ 4 or 8 kB), the operations can be carried out in parallel on multiple drives or on all drives



- In the event of a drive failure, not the entire data can be reconstructed
  - Only small files, which are stored entirely on the remaining drives, can be reconstructed (in theory)
- RAID 0 should only be used when data safety is irrelevant or redundancy is achieved by other means

RAID 1 – Mirroring

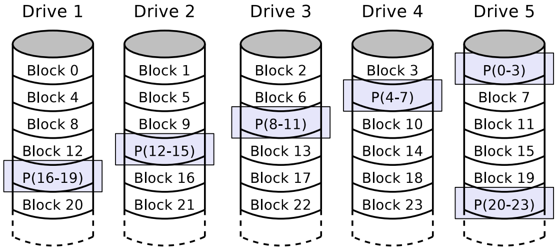
- At least 2 drives of the same capacity store identical data
  - If the drives are of different sizes, RAID 1 provides only the capacity of the smallest drive
- Failure of a single drive does not cause any data loss
  - Reason: The remaining drives store the identical data
- A total loss occurs only in case of the failure of all drives



- Any change of data is written on all drives
- Not a backup replacement
  - Corrupted file operations or virus attacks take place on all drives
- The read performance can be increased by intelligent distribution of requests to the attached drives

RAID 5 – Block-level Striping with distributed Parity Information

- Payload and parity information are distributed to all drives
- Benefits:
  - High throughput
  - High security level against data loss
  - No bottleneck



P(16-19) = block 16 XOR block 17 XOR block 18 XOR block 19

Summary of the RAID Levels

If you want...

the best performance and don't care about availability $\Longrightarrow$ RAID 0
the best availability and don't care about performance $\Longrightarrow$ RAID 1
a combination of performance and availability $\Longrightarrow$ RAID 5 or RAID 6

RAID	n (number of drives)	k (net capacity)	Allowed to fail	Performance (read)	Performance (write)
0	≥ 2	n	0 (none)	$n * X$	$n * X$
1	≥ 2	1	$n - 1$ drives	$n * X$	X
5	≥ 3	$n - 1$	1 drive	$(n-1) * X$	$(n-1) * X$

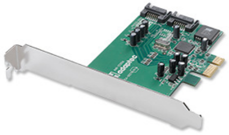
- X is the performance of a single drive during read or write
- The maximum possible performance in theory is often limited by the controller and the computing power of the CPU

If the drives of a RAID 1 have different capacities, the net capacity of a RAID 1 is equal to the capacity of its smallest drive

Hardware/Host/Software RAID (1/2)

Image Source: Adaptec

Adaptec SATA RAID 2410SA


Adaptec SATA II RAID

1220SA

- Hardware RAID
  - A RAID controller with a processor does the calculation of the parity information and monitors the state of the RAID

Benefit:	Operating system independent
	No additional CPU load
Drawback:	High price (approx. EUR 200)

- Host RAID
  - Either an inexpensive RAID controller or the chipset provide the RAID functionality
  - Usually only supports RAID 0 and RAID 1

Benefit:	Operating system independent
	Low price (approx. EUR 50)
Drawback:	Additional CPU load
	Possible dependence of rare hardware

Hardware/Host/Software RAID (2/2)

- Software RAID
  - Linux, Windows and MacOS allow to connect drives to a RAID without a RAID controller

Benefit:	No cost for additional hardware
Drawback:	Operating system dependent
	Additional CPU load

Example Usage

- Create a RAID 1 (`md0`) with the partitions `sda1` and `sdb1`:

```
mdadm --create /dev/md0 --auto md --level=1
      --raid-devices=2 /dev/sda1 /dev/sdb1
```

- Obtain information about any software RAID in the system:

```
cat /proc/mdstat
```

- Obtain information about a specific software RAID (`md0`):

```
mdadm --detail /dev/md0
```

- Remove partition `sdb1` and add partition `sdc1` to the RAID:

```
mdadm /dev/md0 --remove /dev/sdb1
mdadm /dev/md0 --add /dev/sdc1
```

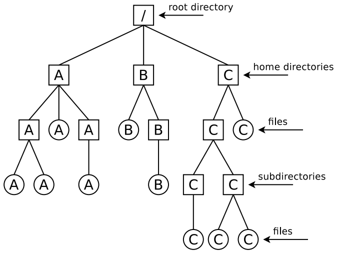
Agenda

Storing Files

How would you store files on a persistent storage device?

File Systems. . .

- organize the storage of files on data storage devices[1]
- manage file names and attributes (**metadata**) of files
- form a namespace, i.e., a hierarchy of directories and files
  - a namespace may comprise multiple file systems



- Absolute path names: Describe the complete path from the root to the file
- Relative path names: All paths, which do not begin with the root

- are a layer of the operating system
  - Processes and users access files via their abstract file names and not via their memory addresses

[1]Files are (semantically related) sequences of bytes

File System Structure

- Physical storage devices (**HDDs**, **SSDs**, or RAID arrays) can be combined into logical storage devices
- Each storage device may contain one or multiple file systems
- A file system may span over multiple storage devices
- Each file system has a **root directory**
- Directories form a tree-like structure
  - Paths can be specified in a relative or absolute form
  - Absolute paths start at the root directory
- On UNIX-like systems multiple file systems are mounted as sub-trees into a Virtual File System (VFS)
  - The **root directory** is denoted as /

Technical Principles of File Systems

- File systems address **clusters** of the storage device
  - Each file occupies an integer number of clusters
  - In literature, clusters are often called zones or blocks
- The size of the clusters is essential for the efficiency of the file system
  - The smaller the clusters are. . .
    - the bigger the overhead for large files
    - the bigger the usable capacity due to less internal fragmentation
  - The bigger the clusters are. . .
    - the smaller the overhead for large files
    - the smaller the usable capacity due to more internal fragmentation

The bigger the clusters, the more memory is lost due to internal fragmentation

- File size: 1 kB. Cluster size: 2 kB $\Longrightarrow$ 1 kB gets lost
- File size: 1 kB. Cluster size: 64 kB $\Longrightarrow$ 63 kB get lost!

- The cluster size can be specified while creating the file system

Agenda

Basic Terminology of Unix File Systems

In Unix: Cluster size $\leq$ size of memory pages (page size)

- The page size depends on the architecture
- x86 = 4 kB, Alpha and UltraSPARC = 8 kB, Apple Silicon = 16 kB, IA-64 = 4/8/16/64 kB

- Unix file systems are based on so called **inodes (*index nodes*)**
- The creation of a **file** causes the creation of an inode
  - It stores a file's metadata, except the file name
    - Metadata contain, e.g., size, (group) owner, permissions, and date
  - Each inode has a unique inode number inside the file system
  - The inode contains references to the file's clusters
  - More than one file name may refer to the same inode (hardlink)
- A **directory** is a file, too (see slide 37)
  - Content: File name and inode number for each file in the directory
- The traditional working method of Unix file systems: **Block addressing**

Attention!

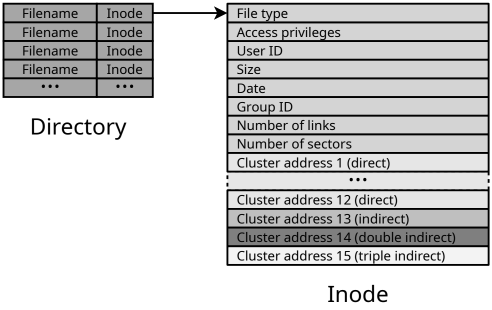
A **block** in Unix' terminology is a **cluster** on a hard disk

File Deletion

We have seen what happens on file creation.
But what about file deletion?

Block Addressing

- Each inode directly stores the numbers of up to 12 clusters



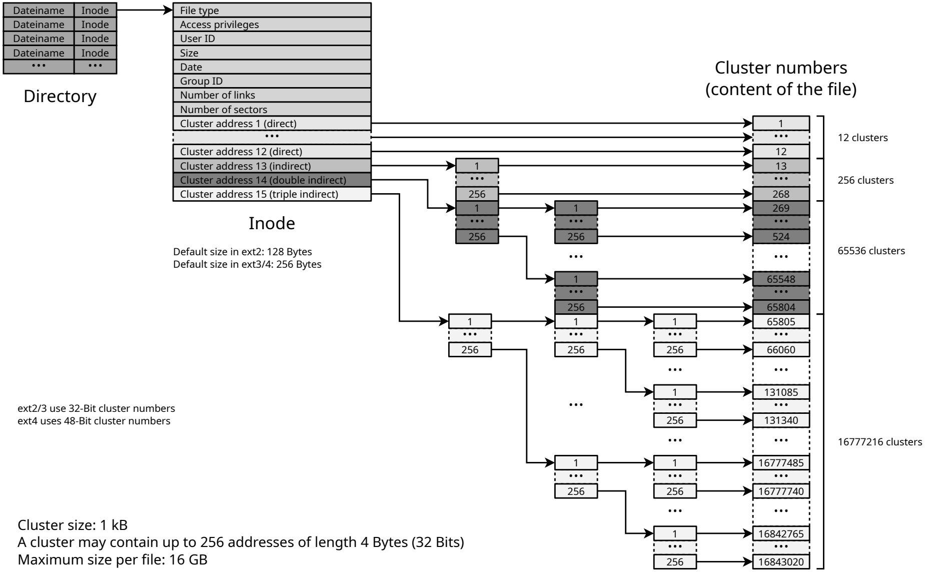
- If a file requires more clusters, these clusters are indirectly addressed
- ext2/3/4, ReiserFS and Reiser4 implement block addressing

Good explanation

`http://lwn.net/Articles/187321/`

- Scenario: No more files can be created in the file system, despite the fact that sufficient capacity is available
- Possible explanation: No more inodes are available
- The command `df -i` shows the number of existing inodes and how many are still available

Direct and indirect Addressing

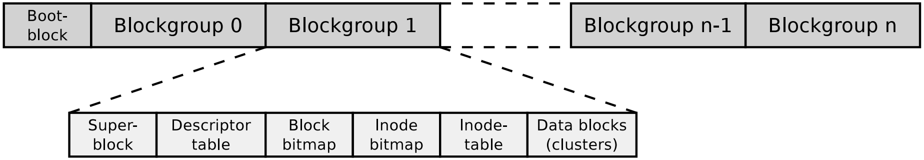


Default size in ext2: 128 Bytes
Default size in ext3/4: 256 Bytes

ext2/3 use 32-Bit cluster numbers
ext4 uses 48-Bit cluster numbers

Cluster size: 1 kB
A cluster may contain up to 256 addresses of length 4 Bytes (32 Bits)
Maximum size per file: 16 GB

ext2/3/4 File System Structure



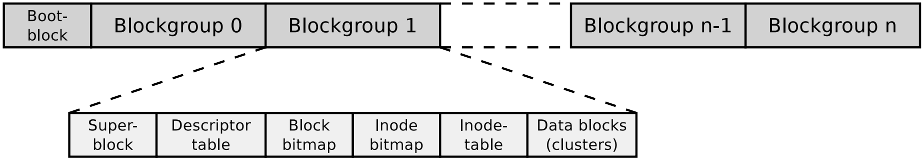
- The clusters of the file system are combined to **block groups** of the same size
  - The information about the metadata and free clusters of each block group are maintained in the respective block group

Maximum size of a block group: 8x cluster size in bytes

Example: If the cluster size is 4096 Bytes, each block group can contain up to 32768 clusters.
$\Longrightarrow$ The maximum block size is 32768 clusters $\times$ 4096 Bytes cluster size = 134,217,728 Bytes = 131,072 kB = 128 MB

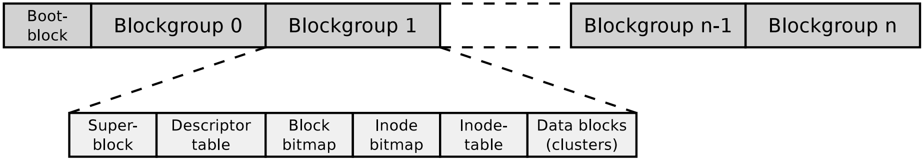
- Benefit of block groups (when using HDDs): inodes (metadata) are physically located closely to the referenced clusters

ext2/3/4 Block Group Structure (1/2)



- The first cluster of the file system contains the **boot block** (size: 1 kB)
  - It contains the boot manager, which starts the operating system
- **Super block**. Contains information about the file system, e.g.,
  - reference to the root directory
  - number of inodes and clusters
- Each block group contains a **copy of the super block**
  - This improves the data security

ext2/3/4 Block Group Structure (2/2)



- The **descriptor table** contains among others:
  - The cluster numbers of the block bitmap and inode bitmap
  - The number of free clusters and inodes in the block group
- **Block bitmap** and **inode bitmap** each have the size of a single cluster
  - They contain the information, which clusters and inodes in the block group are occupied
- The **inode table** contains the inodes of the block group
- The remaining clusters of the block group can be used for the **data**

Analyze File Systems (1/3)

We already know. . .

Directories are files that store file names and inode numbers

Let's take a look inside. . .

With the specification of the file system and some tools, the individual fields of the directory records can be examined. For example the record of `README.txt`

```
# lsblk | grep sda1
sda              8:0     1  29,3G  0 disk
sda1           8:1     1  29,3G  0 part
# mkfs.ext4 /dev/sda1
# mkdir /mnt/test
# mount -t ext4 /dev/sda1 /mnt/test/
# df -h | grep test
/dev/sda1          29G     45M   28G    1% /mnt/test
# ls -l /mnt/test
insgesamt 16
drwx------ 2 root root 16384 Sep 14 09:38 lost+found
# mkdir /mnt/test/testfolder
# echo "Betriebssysteme" > /mnt/test/testfolder/README.txt
# echo "OpSys" > /mnt/test/testfolder/file2.txt
# echo "12345" > /mnt/test/testfolder/anotherfile.dat
# touch /mnt/test/testfolder/empty_file
# ls -lai /mnt/test/testfolder/
insgesamt 20
392449 drwxr-xr-x 2 root root 4096 Sep 14 09:59 .
2 drwxr-xr-x 4 root root 4096 Sep 14 09:46 ..
392452 -rw-r--r-- 1 root root    6 Sep 14 09:58 anotherfile.dat
392453 -rw-r--r-- 1 root root    0 Sep 14 09:59 empty_file
392451 -rw-r--r-- 1 root root    6 Sep 14 09:47 file2.txt
392450 -rw-r--r-- 1 root root   16 Sep 14 09:47 README.txt
```

Analyze File Systems (2/3)

We analyze the contents of `testfolder.out` with command line tools like `hexdump` or `od` (octal dump). Alternatively, it is possible to use a graphical tool such as `wxHexEditor` (see slide 52).

```
# debugfs /dev/sda1
debugfs 1.44.5 (15-Dec-2018)
debugfs:  imap <392449>
Inode 392449 is part of block group 48
located at block 1572896, offset 0x0000
debugfs:  dump testfolder testfolder.out
debugfs:  quit
# ls -l testfolder.out
-rw-r--r-- 1 root root 4096 Sep 14 10:00 testfolder.out
# hexdump -C testfolder.out
00000000  01 fd 05 00 0c 00 01 02  2e 00 00 00 02 00 00 00  |................|
00000010  0c 00 02 02 2e 2e 00 00  02 fd 05 00 14 00 0c 01  |................|
00000020  6c 69 65 73 6d 69 63 68  2e 74 78 74 03 fd 05 00  |README.txt......|
00000030  14 00 09 01 66 69 6c 65  32 2e 74 78 74 00 00 00  |....file2.txt...|
00000040  04 fd 05 00 18 00 0f 01  61 6e 6f 74 68 65 72 66  |........anotherf|
00000050  69 6c 65 2e 64 61 74 00  05 fd 05 00 9c 0f 0a 01  |ile.dat.........|
00000060  65 6d 70 74 79 5f 66 69  6c 65 00 00 00 00 00 00  |empty_file......|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
...
```

A hex editor visualizes data in several ways

- 1st column: Number of previous bytes $\Longrightarrow$ offset or position indicator (address) inside the file in hexadecimal representation
- 2nd column: Bytes of the line in hexadecimal representation
- 3rd column: Bytes of the line in ASCII representation

Some fundamentals...

- Hexadecimal system $\Longrightarrow$ base 16
- 1 hexadecimal digit represents 4 bits
- 2 hexadecimal digits represent 1 byte

Analyze File Systems (3/3)

https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout

Offset	Size	Name	Description
0x0	4 Bytes	inode	Number of the inode that this directory record points to
0x4	2 Bytes	record length	Directory record length
0x6	1 Byte	name length	Length of the file name
0x7	1 Byte	file type	0x0 = unknown, 0x1 = regular file, 0x2 = directory, 0x3 = character-device special file, 0x4 = block-device special file, 0x5 = FIFO (named pipe), 0x6 = socket, 0x7 = symbolic link
0x8		character string	File name

```
00000010  -- -- -- -- -- -- -- --  02 fd 05 00 14 00 0c 01  |................|
00000020  6c 69 65 73 6d 69 63 68  2e 74 78 74 -- -- -- --  |README.txt......|
```

Inode number: 02 fd 05 00 $\Longrightarrow$ 00 05 fd 02

```
$ printf "%d\n" 0x0005fd02
392450
```

Directory record length: 14 00 $\Longrightarrow$ 00 14 $\Longrightarrow$ 20 Bytes

```
$ printf "%d\n" 0x0014
20
```

File name length: 0c $\Longrightarrow$ 12 Bytes File type: 01 $\Longrightarrow$ regular file

Information about Big-Endian vs. Little-Endian Byte Order

x86 processors use the little-endian byte order. If a data field is several bytes long, the least significant byte (LSB) is in the first position, i.e. on the lowest memory address. The more significant bytes are on the following memory addresses

Agenda

File Allocation Table (FAT)

The FAT file system was released in 1980 with QDOS, which was later renamed to MS-DOS

QDOS = Quick and Dirty Operating System

- The **FAT file system** is based on the data structure of the same name
- The **FAT (File Allocation Table)** is a table of fixed size
- The FAT contains an entry for each cluster in the file system, containing information whether the cluster is. . .
  - . . . free
  - . . . the storage medium is damaged at this point
  - . . . occupied by a file
    - In this case it stores the address of the next cluster, which belongs to the file or it is the last cluster of the file
- The clusters of a file are a linked list (cluster chain)

  $\Longrightarrow$ see slides 45 und 46

FAT File System Structure (1/2)

Area 1	Area 2	Area 3	Area 4	Area 5	Area 6
Boot block	Reserved blocks	FAT1	FAT2	Root directory	Data region

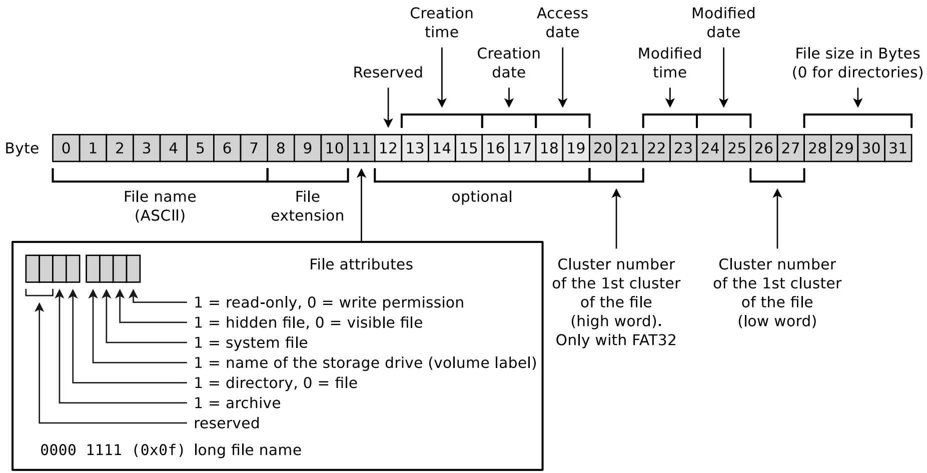
- The **boot sector** contains executable x86 machine code, which starts the operating system, and information about the file system:
  - Block size of the storage device (512, 1024, 2048 or 4096 Bytes)
  - Number of blocks per cluster
  - Number of blocks (sectors) on the storage device
  - Description (name) of the storage device
  - Description of the FAT version
- Between the boot block and the first FAT, optional **reserved blocks** may exist, e.g., for the boot manager
  - These clusters can not be used by the file system

FAT File System Structure (2/2)

Area 1	Area 2	Area 3	Area 4	Area 5	Area 6
Boot block	Reserved blocks	FAT1	FAT2	Root directory	Data region

- The **File Allocation Table** (FAT) stores a record for each cluster in the file system, which informs, whether the cluster is occupied or free
  - The FAT's consistency is essential for the functionality of the file system
    - Therefore, usually a copy of the FAT exists, in order to have a complete FAT as backup in case of a data loss
- In the **root directory**, every file and every directory is represented by an entry:
  - With FAT12 and FAT16, the root directory is located directly behind the FAT and has a fixed size
    - The maximum number of directory entries is therefore limited
  - With FAT32, the root directory can reside at any position in the data region and has a variable size
- The last region contains the actual **data**

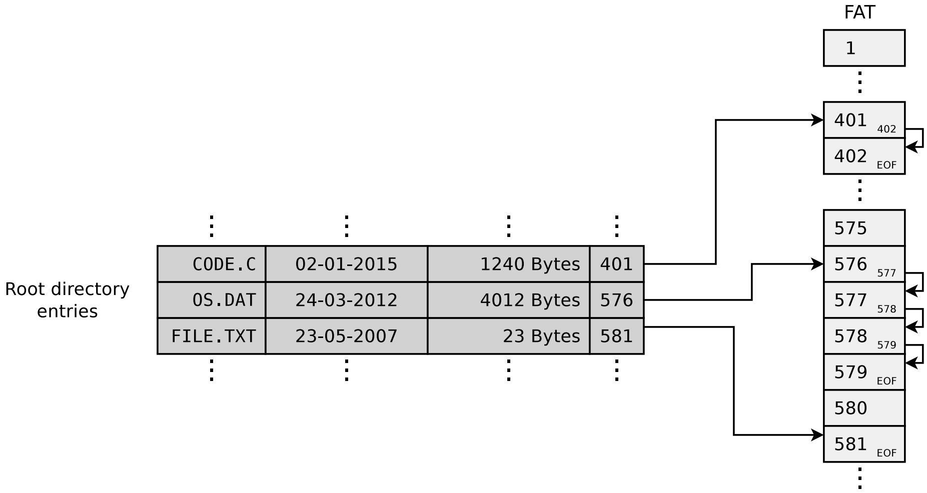
Structure of Root Directory Entries


Creation time
Access date
Modified date
Creation date
Modified time
File size in Bytes (0 for directories)
Reserved
Byte
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
File name (ASCII)
File extension
optional
File attributes
1 = read-only, 0 = write permission
1 = hidden file, 0 = visible file
1 = system file
1 = name of the storage drive (volume label)
1 = directory, 0 = file
1 = archive
reserved
0000 1111 (0x0f) long file name
Cluster number of the 1st cluster of the file (high word). Only with FAT32
Cluster number of the 1st cluster of the file (low word)

Why is 4 GB the maximum file size on FAT32?

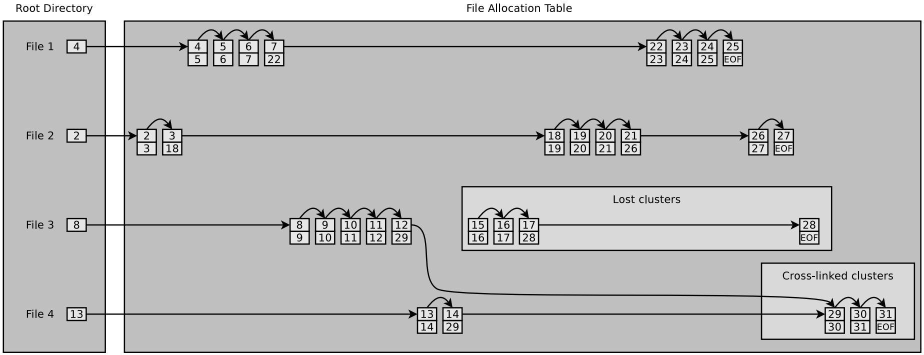
Only 4 Bytes are available for specifying the file size.

Root Directory and FAT



Risk of File System Inconsistencies

- Typical problems of file systems based on a FAT:
  - lost clusters
  - cross-linked clusters



FAT12

Released in 1980 with the first QDOS release

- Length of the cluster numbers: 12 bits
  - Up to $2^{12} = 4096$ clusters can be addressed
- Cluster size: 512 Bytes to 4 kB
- Supports storage media (partitions) up to 16 MB

$2^{12} * 4\,\text{kB cluster size} = 16384\,\text{kB} = 16\,\text{MB maximum file system size}$

- File names are supported only in 8.3 format
  - Up to 8 characters can be used to represent the file name and 3 characters for the file name extension

FAT16

- Released in 1983 because it was foreseeable that an address space of 16 MB is insufficient
- Up to $2^{16} = 65524$ clusters can be addressed
  - 12 clusters are reserved
- Cluster size: 512 Bytes to 256 kB
- File names are supported only in 8.3 format

Sources:
http://support.microsoft.com/kb/140365/de
http://secrets.mysfyts.com/index.asp?Page=Fat
http://web.allensmith.net/Storage/HDDlimit/FAT16.htm

Partition size	Cluster size
up to 31 MB	512 Bytes
32 MB - 63 MB	1 kB
64 MB - 127 MB	2 kB
128 MB - 255 MB	4 kB
256 MB - 511 MB	8 kB
512 MB - 1 GB	16 kB
1 GB - 2 GB	32 kB
2 GB - 4 GB	64 kB
4 GB - 8 GB	128 kB
8 GB - 16 GB	256 kB

The table contains default cluster sizes of Windows 2000/XP/Vista/7/8/10. The cluster size can be manually specified during the file system creation

Some operating systems (e.g., MS-DOS and Windows 95/98/Me) do not support clusters > 32 kB

Some operating systems (e.g., Windows 2000/XP/7/8/10) do not support clusters > 64 kB

FAT32

- Released in 1997 because of the rising HDD capacities and because clusters > 32 kB waste a lot of storage
- Size of the cluster numbers records in the FAT: 32 Bits
  - 4 Bits are reserved
  - Therefore, only $2^{28} = 268,435,456$ clusters can be addressed
- Cluster size: 512 Bytes to 32 kB
- Maximum file size: 4 GB
  - Reason: Only 4 Bytes are available for indicating the file size
- Main field of application today: Mobile storage media > 2 GB

Partition size	Cluster size
up to 63 MB	512 Bytes
64 MB - 127 MB	1 kB
128 MB - 255 MB	2 kB
256 MB - 511 MB	4 kB
512 MB - 1 GB	4 kB
1 GB - 2 GB	4 kB
2 GB - 4 GB	4 kB
4 GB - 8 GB	4 kB
8 GB - 16 GB	8 kB
16 GB - 32 GB	16 kB
32 GB - 2 TB	32 kB

The table contains default cluster sizes of Windows 2000/XP/Vista/7/8/10. The cluster size can be manually specified during the file system creation

Sources: `http://support.microsoft.com/kb/140365/de`

Longer File Names by using VFAT

- VFAT (Virtual File Allocation Table) was released in 1997
  - Extension for FAT12/16/32 to support long filenames
- Because of VFAT, Windows supported for the first time...
  - file names that do not comply with the 8.3 format
  - file names up to a length of 255 characters
- Uses the Unicode character encoding

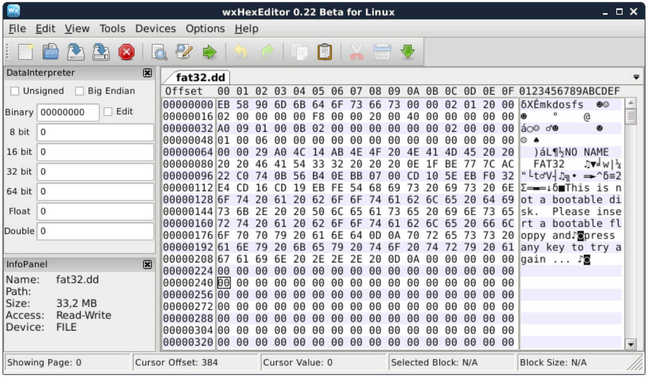
Long file names – Long File Name Support (LFN)

- VFAT is a good example for implementing a new feature without losing backward compatibility
- Long file names (up to 255 characters) are distributed to max. 20 pseudo-directory records
- File systems without Long File Name support ignore the pseudo-directory records and show only the shortened name
- For a VFAT records in the FAT, the first 4 bit of the **file attributes** field have value 1 (see slide 45)
- Special attribute: Upper/lower case is displayed, but ignored

Analyze FAT File Systems (1/3)

```
# dd if=/dev/zero of=./fat32.dd bs=1024000 count=34
34+0 Datensätze ein
34+0 Datensätze aus
34816000 Bytes (35 MB) kopiert, 0,0213804 s, 1,6 GB/s
# mkfs.vfat -F 32 fat32.dd
mkfs.vfat 3.0.16 (01 Mar 2013)
# mkdir /mnt/fat32
# mount -o loop -t vfat fat32.dd /mnt/fat32/
# mount | grep fat32
/tmp/fat32.dd on /mnt/fat32 type vfat (rw,relatime,fmask=0022,dmask=0022,codepage=437,iocharset=utf8,
  shortname=mixed,errors=remount-ro)
# df -h | grep fat32
/dev/loop0       33M     512   33M    1% /mnt/fat32
# ls -l /mnt/fat32
insgesamt 0
# echo "Betriebssysteme" > /mnt/fat32/liesmich.txt
# cat /mnt/fat32/liesmich.txt
Betriebssysteme
# ls -l /mnt/fat32/liesmich.txt
-rwxr-xr-x 1 root root 16 Feb 28 10:45 /mnt/fat32/liesmich.txt
# umount /mnt/fat32/
# mount | grep fat32
# df -h | grep fat32
# wxHexEditor fat32.dd
```

Analyze FAT File Systems (2/3)


wxHexEditor 0.22 Beta for Linux
File Edit View Tools Devices Options Help
DataInterpreter
Unsigned
Big Endian
Edit
InfoPanel
fat32.dd

```
Binary 00000000
8 bit  0
16 bit 0
32 bit 0
64 bit 0
Float  0
Double 0
```

```
Name:    fat32.dd
Path:
Size:    33,2 MB
Access:  Read-Write
Device:  FILE
```

```
Offset   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF
00000000 EB 58 90 6D 6B 64 6F 73 66 73 00 00 02 01 20 00 δXÉmkdosfs  ☻☺
00000016 02 00 00 00 00 F8 00 00 20 00 40 00 00 00 00 00 ☻    °     @
00000032 A0 09 01 00 0B 02 00 00 00 00 00 00 02 00 00 00 á○☺ ♂☻      ☻
00000048 01 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00 ☺ ♠
00000064 00 00 29 A0 4C 14 AB 4E 4F 20 4E 41 4D 45 20 20   )áL¶½NO NAME
00000080 20 20 46 41 54 33 32 20 20 20 0E 1F BE 77 7C AC   FAT32   ♫▼╛w|¼
00000096 22 C0 74 0B 56 B4 0E BB 07 00 CD 10 5E EB F0 32 "└t♂V┤♫╗• ═►^δ≡2
00000112 E4 CD 16 CD 19 EB FE 54 68 69 73 20 69 73 20 6E Σ═▬═↓δ■This is n
00000128 6F 74 20 61 20 62 6F 6F 74 61 62 6C 65 20 64 69 ot a bootable di
00000144 73 6B 2E 20 20 50 6C 65 61 73 65 20 69 6E 73 65 sk.  Please inse
00000160 72 74 20 61 20 62 6F 6F 74 61 62 6C 65 20 66 6C rt a bootable fl
00000176 6F 70 70 79 20 61 6E 64 0D 0A 70 72 65 73 73 20 oppy and♪◙press
00000192 61 6E 79 20 6B 65 79 20 74 6F 20 74 72 79 20 61 any key to try a
00000208 67 61 69 6E 20 2E 2E 2E 20 0D 0A 00 00 00 00 00 gain ... ♪◙
00000224 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000240 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000256 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000272 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000288 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000304 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000320 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
Showing Page: 0   Cursor Offset: 384   Cursor Value: 0   Selected Block: N/A   Block Size: N/A
```

A hex editor visualizes data in several ways

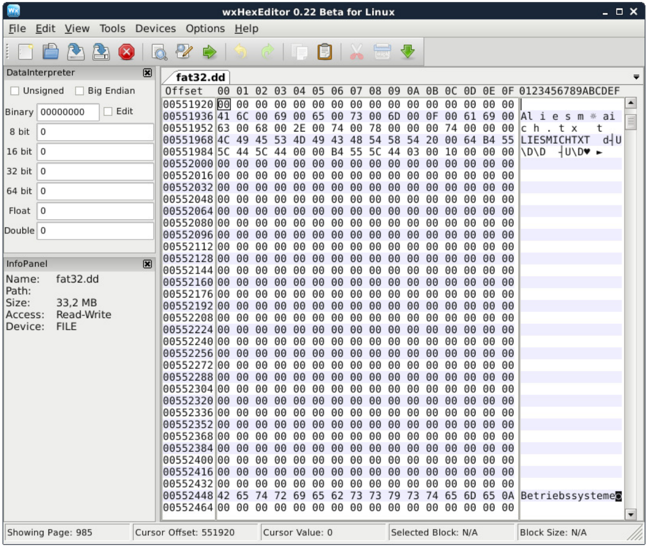
- 1st column: Number of previous bytes $\Longrightarrow$ offset
- 2nd column: Bytes of the line in hexadecimal representation
- 3rd column: Bytes of the line in ASCII representation

Some fundamentals. . .

- Hexadecimal system $\Longrightarrow$ base 16
- 1 hexadecimal digit represents 4 bits
- 2 hexadecimal digits represent 1 byte

http://dorumugs.blogspot.de/2013/01/file-system-geography-fat32.html
http://www.win.tue.nl/~aeb/linux/fs/fat/fat-1.html

Analyze FAT File Systems (3/3)


wxHexEditor 0.22 Beta for Linux
File Edit View Tools Devices Options Help
DataInterpreter
Unsigned
Big Endian
Edit
InfoPanel
fat32.dd

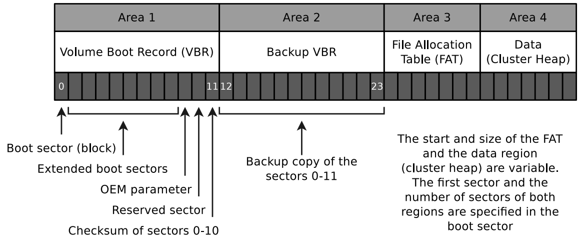
```
Binary  00000000
8 bit   0
16 bit  0
32 bit  0
64 bit  0
Float   0
Double  0

Name:    fat32.dd
Path:
Size:    33,2 MB
Access:  Read-Write
Device:  FILE
```

```
Offset   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 0123456789ABCDEF
00551920 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00551936 41 6C 00 69 00 65 00 73 00 6D 00 0F 00 61 69 00 Al i e s m ☼ ai
00551952 63 00 68 00 2E 00 74 00 78 00 00 00 74 00 00 00 c h . t x   t
00551968 4C 49 45 53 4D 49 43 48 54 58 54 20 00 64 B4 55 LIESMICHTXT  d┤U
00551984 5C 44 5C 44 00 00 B4 55 5C 44 03 00 10 00 00 00 \D\D  ┤U\D♥ ►
00552000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552016 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552032 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552048 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552064 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552096 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552112 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552128 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552144 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552176 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552192 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552208 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552224 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552240 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552256 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552272 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552288 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552304 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552320 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552336 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552352 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552368 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552384 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552400 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552416 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552432 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00552448 42 65 74 72 69 65 62 73 73 79 73 74 65 6D 65 0A Betriebssysteme◙
00552464 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
Showing Page: 985 | Cursor Offset: 551920 | Cursor Value: 0 | Selected Block: N/A | Block Size: N/A
```

exFAT

- Released in 2006 (usage is royalty-free since 2019)
- Up to $2^{32} = 4,294,967,296$ clusters can be addressed
- Cluster size: 512 Bytes to 64 MB
- Maximum file size: 16 EB (2^{64} Bytes)
- Main field of application: mobile flash memory (> 32 GB)
  - Fewer write operations than file systems with a journal (e.g., NTFS $\Longrightarrow$ slide 60)


Area 1
Area 2
Area 3
Area 4
Volume Boot Record (VBR)
Backup VBR
File Allocation Table (FAT)
Data (Cluster Heap)
0
11
12
23
Boot sector (block)
Extended boot sectors
OEM parameter
Reserved sector
Checksum of sectors 0-10
Backup copy of the sectors 0-11
The start and size of the FAT and the data region (cluster heap) are variable. The first sector and the number of sectors of both regions are specified in the boot sector

In contrast to the other FAT file system versions, the root directory does not have a fixed position. It is located within the data area and usually does not reside there in one piece, but is fragmented.

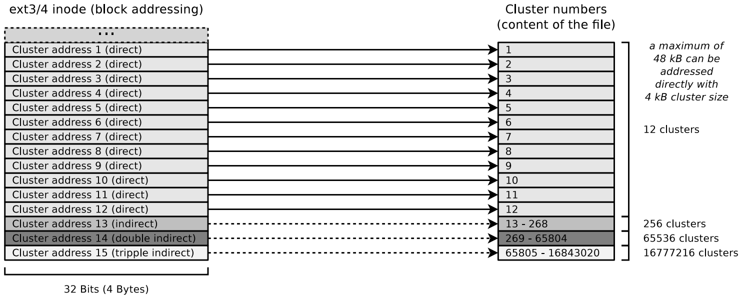
Partition size	Cluster size
up to 256 MB	4 kB
256 MB - 32 GB	32 kB
32 GB - 256 TB	128 kB

The table contains default cluster sizes of Windows 2000/XP/Vista/8/10. The cluster size can be manually specified during the file system creation
`https://support.microsoft.com/de-de/kb/140365`

Agenda

Problem: Metadata Overhead

- Every inode at block addressing addresses a certain number of cluster numbers directly
- If a file requires more clusters, they are indirectly addressed

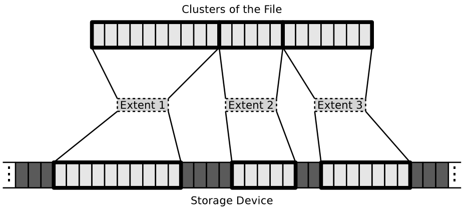


ext3/4 inode (block addressing)
...
Cluster address 1 (direct)
Cluster address 2 (direct)
Cluster address 3 (direct)
Cluster address 4 (direct)
Cluster address 5 (direct)
Cluster address 6 (direct)
Cluster address 7 (direct)
Cluster address 8 (direct)
Cluster address 9 (direct)
Cluster address 10 (direct)
Cluster address 11 (direct)
Cluster address 12 (direct)
Cluster address 13 (indirect)
Cluster address 14 (double indirect)
Cluster address 15 (tripple indirect)
32 Bits (4 Bytes)
Cluster numbers (content of the file)
1
2
3
4
5
6
7
8
9
10
11
12
13 - 268
269 - 65804
65805 - 16843020
a maximum of 48 kB can be addressed directly with 4 kB cluster size
12 clusters
256 clusters
65536 clusters
16777216 clusters

- This addressing scheme causes rising overhead with rising file size
- Solution: Extents

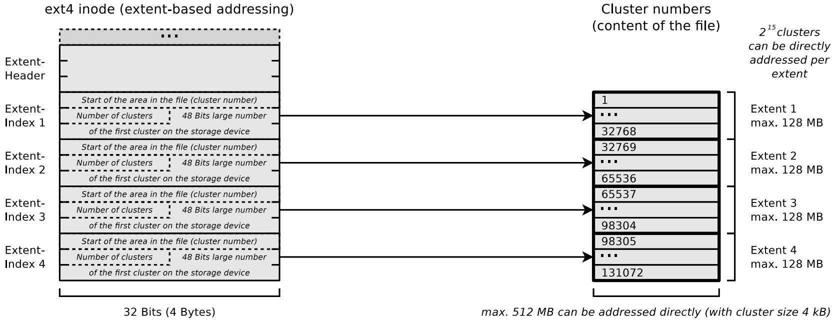
Extent-based Addressing

- Inodes do not address individual clusters, but instead create large areas of files to areas of contiguous blocks (**extents**) on the storage device
- Instead of many individual clusters numbers, only 3 values are required:
  - Start (cluster number) of the area (extent) in the file
  - Size of the area in the file (in clusters)
  - Number of the first cluster on the storage device
- Result: Lesser overhead
- Examples: JFS, XFS, btrfs, NTFS, ext4



Extents using the Example ext4

- With block addressing in ext2/3, each inode contains 15 areas with a size of 4 Bytes each ($\Longrightarrow$ 60 Bytes) for addressing clusters
- ext4 uses this 60 Bytes for an extent header (12 Bytes) and for addressing 4 extents (12 Bytes each)

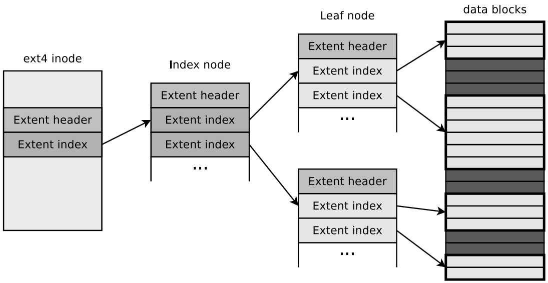

ext4 inode (extent-based addressing)
...
Extent-Header
Extent-Index 1
Extent-Index 2
Extent-Index 3
Extent-Index 4
Start of the area in the file (cluster number)
Number of clusters
48 Bits large number
of the first cluster on the storage device
32 Bits (4 Bytes)
Cluster numbers (content of the file)
2^{15} clusters can be directly addressed per extent
1
...
32768
32769
65536
65537
98304
98305
131072
Extent 1 max. 128 MB
Extent 2 max. 128 MB
Extent 3 max. 128 MB
Extent 4 max. 128 MB
max. 512 MB can be addressed directly (with cluster size 4 kB)

Extents cannot become larger than 128 MB (2^{15} bits) because ext4, just like its predecessors ext2 and ext3, organizes the file system clusters into so-called block groups (see slide 34) with a maximum size of 128 MB.

Maximum partition size of ext4: 2^{48} cluster numbers $\times$ 4096 Byte cluster size = 1 Exabyte

Benefit of Extents using the Example ext4

- With a maximum of 12 clusters, an ext3/4 inode can directly address 48 kB (at 4 kB cluster size)
- With 4 extents, an ext4 inode can directly address 512 MB
- If the size of a file is $>$ 512 MB, ext4 creates a tree of extents
    - The principle is analogous to indirect block addressing



Helpful descriptions of Extents in ext4...

https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout#Extent_Tree
https://www.sans.org/blog/understanding-ext4-part-3-extent-trees/
https://metebalci.com/blog/a-minimum-complete-tutorial-of-linux-ext4-file-system/
An analysis of Ext4 for digital forensics: https://www.sciencedirect.com/science/article/pii/S1742287612000357

NTFS – New Technology File System

Several different versions of the NTFS file system exist

- NTFS 1.0: Windows NT 3.1 (released in 1993)
- NTFS 1.1: Windows NT 3.5/3.51
- NTFS 2.x: Windows NT 4.0 bis SP3
- NTFS 3.0: Windows NT 4.0 ab SP3/2000
- NTFS 3.1: Windows XP/2003/Vista/7/8/10

Recent versions of NTFS offer additional features as. . .

- support for quotas since version 3.x
- transparent compression
- transparent encryption (Triple-DES and AES) since version 2.x

- Cluster size: 512 Bytes to 64 kB
- NTFS offers, compared with its predecessor FAT, among others:
  - Maximum file size: 16 TB ($\Longrightarrow$ extents)
  - Maximum partition size: 256 TB ($\Longrightarrow$ extents)
  - Security features on file and directory level
- Equal to VFAT. . .
  - implements NTFS file names up a length of 255 Unicode characters
  - implements NTFS interoperability with the MS-DOS operating system family by storing a unique file name in the format 8.3 for each file

Structure of NTFS (1/2)

- The file system contains a Master File Table (MFT)
  - It contains the references of the files to the clusters
  - Also contains the metadata of the files (file size, file type, date of creation, date of last modification and possibly the file content)
    - The content of small files $\leq$ 900 Bytes is stored directly in the MFT

Source: How NTFS Works. Microsoft. 2003. `https://technet.microsoft.com/en-us/library/cc781134(v=ws.10).aspx`

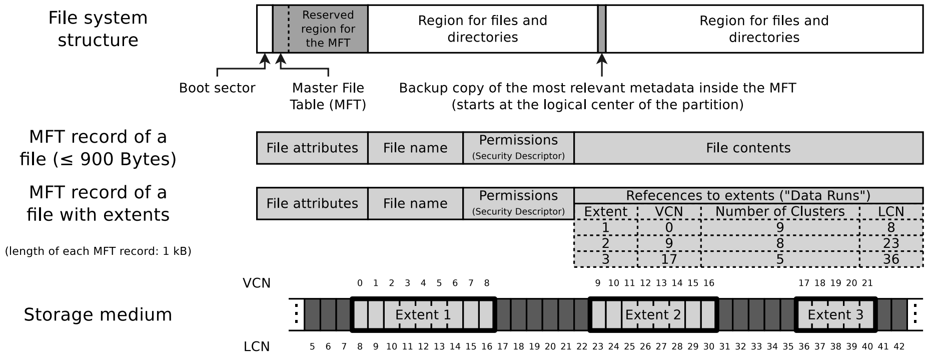
- When a partition is formatted as, a fixed space is reserved for the MFT
  - 12.5% of the partition size is reserved for the MFT by default
  - If the MFT area has no more free capacity, the file system uses additional free space in the partition for the MFT
    - This may cause fragmentation of the MFT (but has no negative effects for flash memory)

Partition size	Cluster size
< 16 TB	4 kB
16 TB - 32 TB	8 kB
32 TB - 64 TB	16 kB
64 TB - 128 TB	32 kB
128 TB - 256 TB	64 kB

The table contains default cluster sizes of Windows 2000/XP/Vista/7/8/10. The cluster size can be specified when the file system is created

Source: `http://support.microsoft.com/kb/140365/de`

Structure of NTFS (2/2)


File system structure
Reserved region for the MFT
Region for files and directories
Region for files and directories
Boot sector
Master File Table (MFT)
Backup copy of the most relevant metadata inside the MFT (starts at the logical center of the partition)
MFT record of a file (≤ 900 Bytes)
File attributes
File name
Permissions (Security Descriptor)
File contents
MFT record of a file with extents
File attributes
File name
Permissions (Security Descriptor)
Refecences to extents ("Data Runs")
(length of each MFT record: 1 kB)
VCN
0 1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
17 18 19 20 21
Storage medium
Extent 1
Extent 2
Extent 3
LCN
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42

Extent	VCN	Number of Clusters	LCN
1	0	9	8
2	9	8	23
3	17	5	36

If an MFT entry refers to extents (so called „Data Runs“), it stores 3 values per extent

- Start (cluster number) of the area (extent) in the file $\Longrightarrow$ Virtual Cluster Number (VCN)
- Size of the area in the file (in clusters) $\Longrightarrow$ number of clusters
- Number of the first cluster on the storage device $\Longrightarrow$ Logical Cluster Number (LCN)

Also a directory is a file (MFT entry) whose file contents are the numbers of the MFT entries (files) associated with the directory

Agenda

Problem: Write Operations

- If files or directories are created, relocated, renamed, erased, or modified, write operations in the file system are carried out
  - Write operations shall convert data from one consistent state to a new consistent state
- If a failure occurs during a write operation, the consistency of the file system must be checked
  - If the size of a file system is multiple GB, the consistency check may take several hours or days
  - Skipping the consistency check, may cause data loss
- Objective: Narrow down the data, which need to be checked by the consistency check
- Solution: Collect the write operations in a journal
  $\Longrightarrow$ **Journaling file systems**

Journaling File Systems

- Implement a journal, where write operations are collected before being committed to the file system
  - At fixed time intervals, the journal is closed and the write operations are carried out
- Advantage: After a crash, only the files (clusters) and metadata must be checked, for which a record exists in the journal
- Drawback: Journaling increases the number of write operations, because modifications are first written to the journal and next carried out
- 2 variants of journaling:
  - **Metadata journaling**
  - **Full journaling**

Helpful descriptions of the different journaling concepts. . .

- Analysis and Evolution of Journaling File Systems, *Vijayan Prabhakaran, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau*, 2005 USENIX Annual Technical Conference, `http://www.usenix.org/legacy/events/usenix05/tech/general/full_papers/prabhakaran/prabhakaran.pdf`

Metadata Journaling and Full Journaling

- **Metadata journaling** (*Write-Back*)
  - The journal contains only metadata (inode) modifications
    - Only the consistency of the metadata is ensured after a crash
  - Modifications to clusters are carried out by `sync()` ($\Longrightarrow$ write-back)
    - The `sync()` system call commits the page cache, that is also called = buffer cache to the HDD/SDD
  - Advantage: Consistency checks only take a few seconds
  - Drawback: Loss of data due to a system crash is still possible
  - Optional with ext3/4 and ReiserFS
  - NTFS and XFS provides only metadata journaling
- **Full journaling**
  - Modifications to metadata and clusters of files are written to the journal
  - Advantage: Ensures the consistency of the files
  - Drawback: All write operation must be carried out twice
  - Optional with ext3/4 and ReiserFS

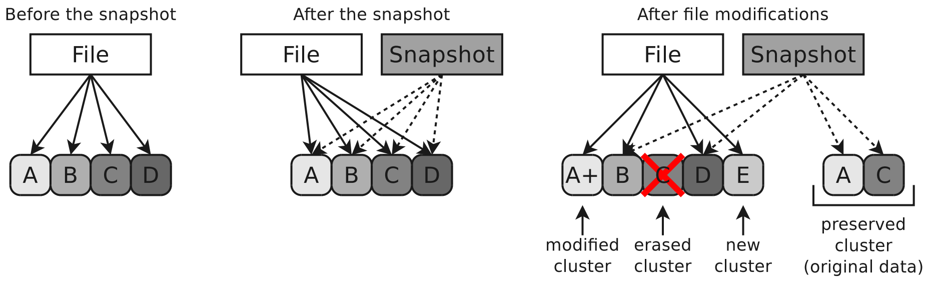
The alternative is therefore high data security and high write speed

Compromise between the Variants: Ordered Journaling

- Most Linux distributions use by default a compromise between both variants
- Ordered journaling
  - The journal contains only metadata modifications
  - File modifications are carried out in the file system first and next the relevant metadata modifications are written into the journal
  - Advantage: Consistency checks only take a few seconds and high write speed equal to journaling, where only metadata is journaled
  - Drawback: Only the consistency of the metadata is ensured
    - If a crash occurs while incomplete transactions in the journal exist, new files and attachments get lost because the clusters are not yet allocated to the inodes
    - Overwritten files after a crash may have inconsistent content and maybe cannot be repaired, because no copy of the old version exists
  - Examples: Only option when using JFS, standard with ext3/4 and ReiserFS

Interesting:
`https://www.heise.de/newsticker/meldung/Kernel-Entwickler-streiten-ueber-Ext3-und-Ext4-209350.html`

Most advanced Concept: Copy-on-write



- Write operations do not modify/erase file system contents
  - Modified data is written into free clusters
  - Afterward, the metadata is modified for the new file
- Older file versions are preserved and can be restored
  - Data security is better compared with journaling file systems
  - Snapshots can be created without delay (just metadata modification)
- Examples: ZFS, btrfs and ReFS (Resilient File System)

You should now be able to answer the following questions:

- How are Hard Disk Drives structured and how do they work?
- What are the characteristics and functioning of Solid State Drives?
- What is the purpose of Redundant Array of Independent Disks (RAID)?