# 6th Slide Set
Operating Systems

Prof. Dr. Christian Baun

Frankfurt University of Applied Sciences
(1971–2014: Fachhochschule Frankfurt am Main)
Faculty of Computer Science and Engineering
christianbaun@fb2.fra-uas.de

## Learning Objectives of this Slide Set

- At the end of this slide set, you know/understand. . .
  - the **functions and basic terminology of file systems**
  - what **inodes** and **clusters** are
  - how **block addressing** works
  - the **structure** of selected file systems
  - an overview about **Windows file systems** and their characteristics
  - what **journaling** is and why it is used by many file systems today
  - how addressing via **extents** works
  - what **copy-on-write** is
  - how **defragmentation** works and when it makes sense to defragment

Exercise sheet 6 repeats the contents of this slide set which are relevant for these learning objectives
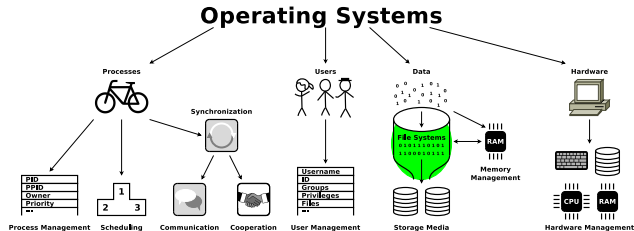
File Systems. . .

- organize the storage of files on data storage devices
  - Files are sequences of Bytes of any length which belongs together with regard to content
- manage file names and attributes (metadata) of files
- form a namespace
  - Hierarchy of directories and files
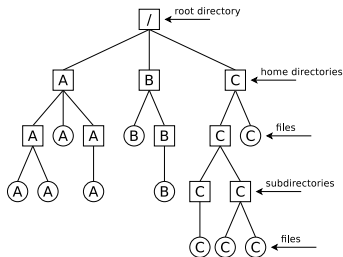
> - **Absolute path names**: Describe the complete path **from the root to the file**
> - **Relative path names**: All paths, which do **not begin with the root**

- are a layer of the operating system
  - Processes and users access files via their abstract file names and not via their memory addresses
- should cause only little overhead for metadata

Technical Principles of File Systems

- File systems address **clusters** and not blocks of the storage device
    - Each file occupies an integer number of clusters
    - In literature, the clusters are often called **zones** or **blocks**
        - This results in confusion with the sectors of the devices, which are in literature sometimes called blocks too
- The size of the clusters is essential for the efficiency of the file system
    - The smaller the clusters are. . .
        - Rising overhead for large files
        - Decreasing capacity loss due to internal fragmentation
    - The bigger the clusters are. . .
        - Decreasing overhead for large files
        - Rising capacity loss due to internal fragmentation

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

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

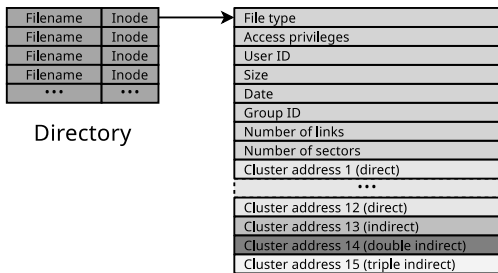- The cluster size can be specified, while creating the file system

Basic Terminology of Linux File Systems

In Linux: Cluster size ≤ 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

- The creation of a **file** causes the creation of an **Inode** (*index node*)
    - It stores a file's metadata, except the file name
        - Metadata are among others the size, UID/GID, permissions and date
    - Each inode has a unique inode number inside the file system
    - The inode contains references to the file's clusters
    - All Linux file systems base on the functional principle of inodes
- A **directory** is a file too (see slide 12)
    - Content: File name and inode number for each file in the directory
- The traditional working method of Linux file systems: **Block addressing**
    - Actually, the term is misleading because file systems always address clusters and not blocks (of the volume)
        - However, the term is established in literature since decades

Block Addressing using the Example ext2/3

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

Filename	Inode
Filename	Inode
Filename	Inode
Filename	Inode
...	...

Directory

File type
Access privileges
User ID
Size
Date
Group ID
Number of links
Number of sectors
Cluster address 1 (direct)
...
Cluster address 12 (direct)
Cluster address 13 (indirect)
Cluster address 14 (double indirect)
Cluster address 15 (triple indirect)

Inode

- If a file requires more clusters, these clusters are indirectly addressed
- Minix, ext2/3, 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 using the Example ext2/3

Directory

Inode

Default size in ext2: 128 Bytes
Default size in ext3/4: 256 Bytes
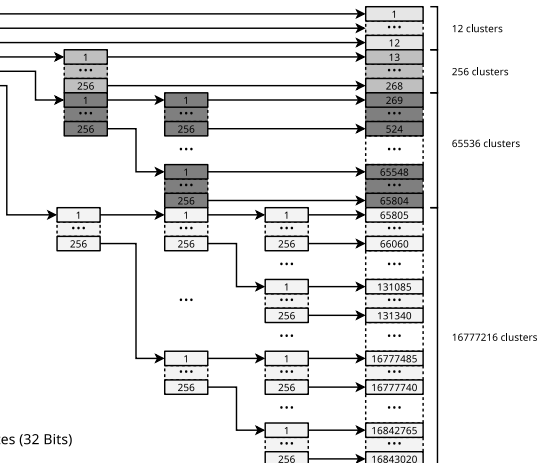
ext2/3 use 32-Bit cluster numbers
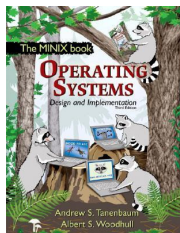ext4 uses 48-Bit cluster numbers

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

Minix

The Minix operating system   http://www.minix3.org

- Unix-like operating system
- Developed since 1987 by Andrew S. Tanenbaum for educational purposes
  https://www.youtube.com/watch?v=bx3KuE7UjGA
- Latest revision is 3.3.0 is from 2014
- Intel chipsets post-2015 run MINIX 3 internally as the software component of the Intel
  Management Engine
  https://www.zdnet.com/article/minix-intels-hidden-in-chip-operating-system/
  https://linuxnews.de/2017/11/minix-in-der-intel-management-engine/
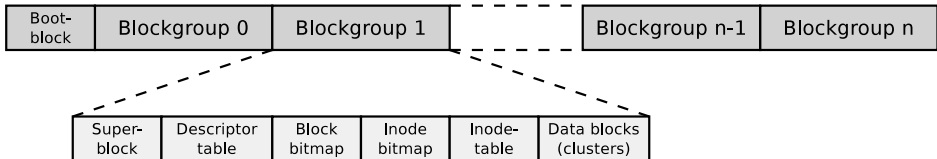 https://itsfoss.com/fact-intel-minix-case/

- Standard Linux file system until 1992
  - Not surprising, because Minix was the basis for the development of Linux
- The Minix file system causes low overhead
  - Useful applications „today": Boot floppy disks and RAM disks
- Storage is represented as a linear chain of equal-sized blocks (1-8 kB)
- A Minix file system contains just 6 areas
  - The simple structure makes it ideal for educational purposes

Minix File System Structure

Area 1	Area 2	Area 3	Area 4	Area 5	Area 6
Boot block (1 cluster)	Super block (1 cluster)	Inodes bitmap (1 cluster)	Cluster bitmap (1 cluster)	Inodes (15 clusters)	Data (remaining clusters)

- **Boot block**. Contains the boot loader that starts the operating system
- **Super block**. Contains information about the file system,
  - e.g. number of inodes and clusters
- **Inodes bitmap**. Contains a list of all inodes with the information, whether the inode is occupied (value: 1) or free (value: 0)
- **Clusters bitmap**. Contains a list of all clusters with the information, whether the cluster is occupied (value: 1) or free (value: 0)
- **Inodes**. Contains the inodes with the metadata
  - Every file and every directory is represented by at least a single inode, which contains the metadata
    - Metadata is among others the file type, UID/GID, access privileges, size
- **Data**. Contains the contents of the files and directories
  - This is the biggest part in the file system

ext2/3/4 File System Structure

Boot-block	Blockgroup 0	Blockgroup 1		Blockgroup n-1	Blockgroup n

Super-block	Descriptor table	Block bitmap	Inode bitmap	Inode-table	Data blocks (clusters)

- 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 is maintained in the respective block group
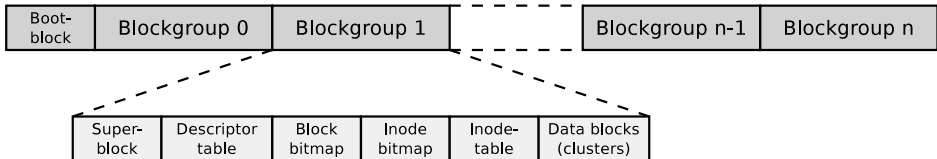

Maximum size of a block group: 8x cluster size in bytes

Example: If the cluster size is 4096 Bytes, each block group can contain up to 32768 clusters.
$\implies$ 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 close to the clusters they address
    - This reduces seek times and the degree of fragmentation
    - When using flash storage, the position of the data in the individual memory cells is irrelevant for the performance

ext2/3/4 Block Group Structure

Boot-block	Blockgroup 0	Blockgroup 1		Blockgroup n-1	Blockgroup n

Super-block	Descriptor table	Block bitmap	Inode bitmap	Inode-table	Data blocks (clusters)

- The first cluster of the file system contains the **boot block** (size: 1 kB)
  - It contains the boot manager, which starts the operating system
- Each block group contains a **copy of the super block**
  - This improves the data security
- 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** are each a single cluster big
  - 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
liesmich.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/liesmich.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 liesmich.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 27).

```
# 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 |liesmich.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 32 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 ⟹ 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 ⟹ 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 -- -- -- -- |liesmich.txt....|
```

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

Inode number: 02 fd 05 00 ⟹ 00 05 fd 02

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

Directory record length: 14 00 ⟹ 00 14 ⟹ 20 Bytes

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

File name length: 0c ⟹ 12 Bytes
File type: 01 ⟹ regular file

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 File Allocation Table (FAT) file system is based on the data structure of the same name
- The FAT (**File Allocation Table**) is a table of fixed size
- For each cluster in the file system, an entry exists in the FAT with the following information about the cluster:
    - Cluster is free or the storage medium is damaged at this point
    - Cluster is 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**)
    $\implies$ see slides 18 und 20

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**

Root Directory and FAT
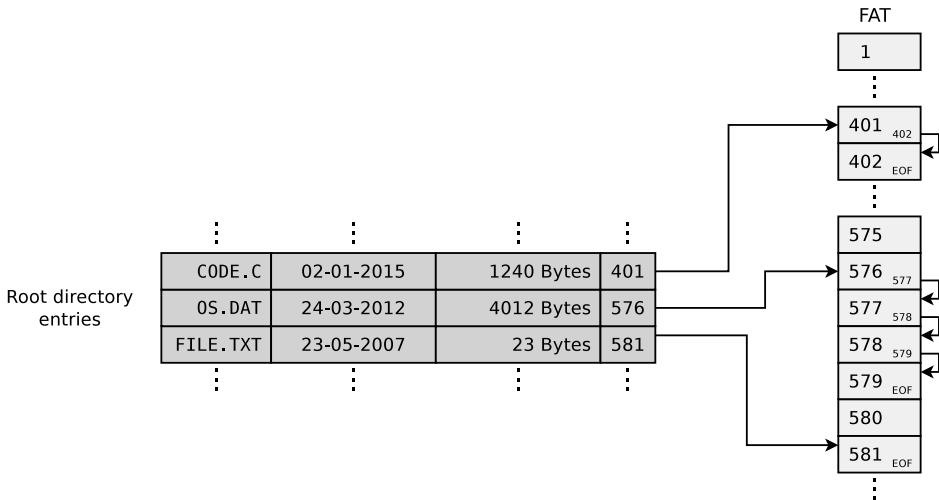
The topic FAT is well explained by. . .

● **Betriebssysteme**, *Carsten Vogt*, 1st edition, Spektrum Akademischer Verlag (2001), P. 178-179

Prof. Dr. Christian Baun – 6th Slide Set Operating Systems – Frankfurt University of Applied Sciences – WS2425   18/50

Structure of Root Directory Entries

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

Creation time
Creation date
Access date
Reserved
Modified date
Modified time
File size in Bytes (0 for directories)

File name (ASCII)
File extension
optional

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)

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

Why is 4 GB the maximum file size on FAT32?

Only 4 Bytes are available for specifying the file size.

Risk of File System Inconsistencies

- Typical problems of file systems based on a FAT:
    - lost clusters
    - cross-linked clusters
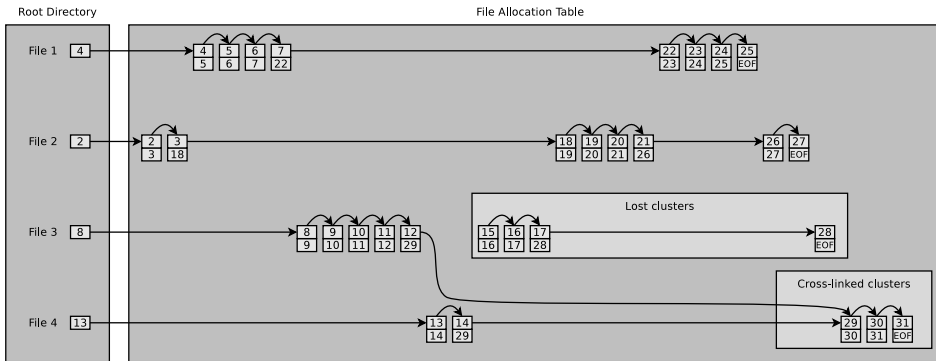
Source: http://www.sal.ksu.edu/faculty/tim/ossg/File_sys/file_system_errors.html

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

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

Used „today" only for DOS and Windows floppy disks

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
- Main field of application today: Mobile storage media ≤ 2 GB

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

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

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

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

- Implements Unicode character encoding

Long File Names (LFN)

- VFAT = great example for implementing a new feature + keeping backward compatibility
- LFNs (255 characters max) are split into max. 20 pseudo-directory records (see slide 25)
- File systems without LFN support ignore pseudo-directory records and show only the shortened name
- The first 4 bits of the **file attributes** field of VFAT records have value 1 (see slide 18)
- Special attribute: Upper/lower case is displayed, but ignored

Compatibility with MS-DOS and Windows up to v3.11

- VFAT and NTFS (see slide 38) store for every file a unique filename in 8.3 format
    - Operating systems without the VFAT extension ignore the pseudo-directory entries and only show the shortened file name
        - This way, Microsoft operating systems without NTFS and VFAT support can access files on NTFS partitions
- Challenge: **The short file names must be unique**
- Solution:
    - All special characters and dots inside the name are erased
    - All lowercase letters are converted to uppercase letters
    - Only the first 6 characters are kept
        - Next, a ~1 follows before the dot
    - The first 3 characters after the dot are kept and the rest is erased
    - If a file with the same name already exists, ~1 is replaced with ~2, etc.
- Example:
  A very long filename.test.pdf $\implies$ AVERYL~1.pdf

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)

A hex editor visualizes data in several ways

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

Some fundamentals. . .

- Hexadecimal system $\implies$ 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)

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 38)

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

Area 1	Area 2	Area 3	Area 4
Volume Boot Record (VBR)	Backup VBR	File Allocation Table (FAT)	Data (Cluster Heap)

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

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 needs to be checked by the consistency check**
- Solution: Collect the write operations in a journal
    $\implies$ **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 then 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

- http://www.ibm.com/developerworks/library/l-journaling-filesystems/index.html

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 (see slide 42) to the HDD/SSD
    - 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 implement 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 operations 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

Problem: Metadata Overhead

- Every inode at block addressing addresses a certain number of cluster numbers directly
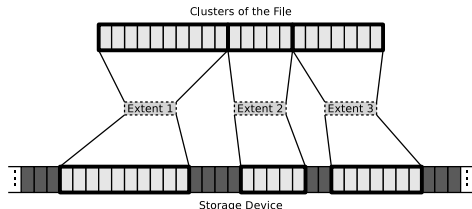- If a file requires more clusters, they are indirectly addressed
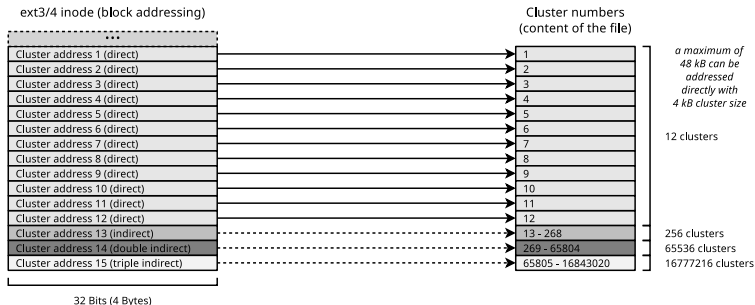
- This addressing scheme causes rising overhead with rising file size
- Solution: Extents

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 cluster 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)
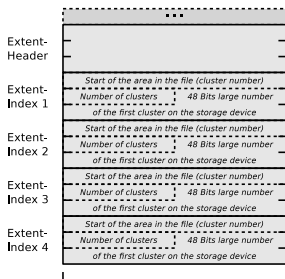 - Number of the first cluster on the storage device

- Result: Lesser overhead
- Examples: JFS, XFS, btrfs, NTFS, ext4
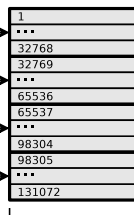
Extents using the Example ext4

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

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 10) 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
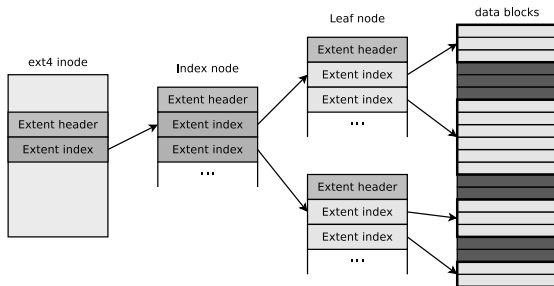
Helpful descriptions of Extents in ext4...

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 – **N**ew **T**echnology **F**ile **S**ystem

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 until SP3
- NTFS 3.0: Windows NT 4.0 since 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 ($\implies$ extents)
  - Maximum partition size: 256 TB ($\implies$ extents)
  - Security features on file and directory level
- Equal to VFAT. . .
  - implements NTFS file names up to 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 ≤ 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, 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)
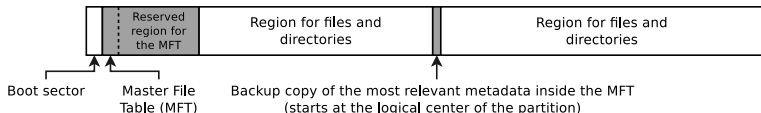
MFT record of a file (≤ 900 Bytes)

File attributes	File name	Permissions (Security Descriptor)	File contents

MFT record of a file with extents

(length of each MFT record: 1 kB)

File attributes	File name	Permissions (Security Descriptor)	Refecences to extents ("Data Runs")			
			Extent	VCN	Number of Clusters	LCN
			1	0	9	8
			2	9	8	23
			3	17	5	36

Storage medium

VCN: 0 1 2 3 4 5 6 7 8   9 10 11 12 13 14 15 16   17 18 19 20 21

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
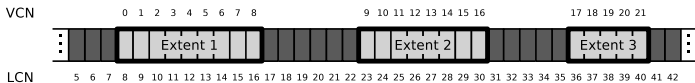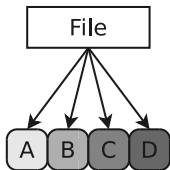

If an MFT entry refers to extents (so called „Data Runs"), it stores 3 values per extent

- 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)**
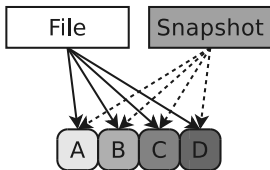

Also a directory is a file (MFT entry) whose file contents are the numbers of the MFT entries (files) associated with the directory

Most advanced Concept: Copy-on-write
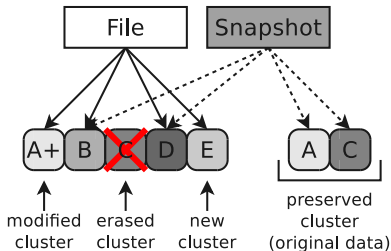

Before the snapshot   After the snapshot   After file modifications

- 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)

Accelerating Data Access with a Cache (1/2)

- Modern operating systems accelerate the access to stored data with a **Page Cache** (called *Buffer Cache*) in the main memory
  - If a file is requested for reading, the kernel first tries to allocate the file in the cache
    - If the file is not present in the cache, it is loaded into the cache
- The page cache is never as big as the amount of data on the system
  - That is why rarely requested files must be replaced
    - If data in the cache was modified, the modification must be passed down (written back) at some point in time
    - Optimal use of the cache is impossible because data accesses are non-deterministic (unpredictable)
- Modern operating systems usually do not pass down write accesses immediately ($\Longrightarrow$ **write-back**)
  - Benefit: Better system performance
  - Drawback: System crashes may cause inconsistencies

Accelerating Data Access with a Cache (2/2)

- DOS and Windows up to version 3.11 use the *Smartdrive* utility to implement a page cache
    - All later versions of Windows also contain a *Cache Manager* that implements a page cache
- Linux automatically buffers as much data as there is free space in main memory
    - The command `free -m` returns an overview of the memory usage
        - The `buff/cache` column indicates how much main memory is currently used for the page cache

```
$ free -m
            total      used      free    shared buff/cache  available
Mem:        15540      5774      1609       785       8156       8650
Swap:       10689         1     10688
```

Good sources regarding the page cache under Linux and how to empty it manually

https://askubuntu.com/questions/770108/what-do-the-changes-in-free-output-from-14-04-to-16-04-mean
http://www.thomas-krenn.com/de/wiki/Linux_Page_Cache
http://unix.stackexchange.com/questions/87908/how-do-you-empty-the-buffers-and-cache-on-a-linux-system
http://serverfault.com/questions/85470/meaning-of-the-buffers-cache-line-in-the-output-of-free

Howto implement Write-through mode? (1/3)

- The **Write-back** policy is standard in many hardware and operating system components (see slide sets 3, 5, and 6)
    - How is it possible to switch to **Write-through** write policy?

- In **Linux**, it is impossible to turn off the page cache globally
    - But it can be done on application and sometimes on file system level
        - However, it is only useful in some cases, like using an application with a dedicated cache in userspace (e.g., a DBMS), using memory-like storage devices or ramdisks or doing performance testing of devices

- The **Linux function call** `open` (`fcntl.h`) for creating or opening a file offers the flags `O_DIRECT` and `O_SYNC`
    - Setting them both specifies that write operations to this file bypass the Linux page cache and are immediately flushed to the drive

https://man7.org/linux/man-pages/man2/open.2.html
https://pubs.opengroup.org/onlinepubs/9699919799.2018edition/functions/open.html

Howto implement Write-through mode? (2/3)

- The **mount option** `-o sync` bypasses the page cache when using the file systems ext2, ext3, FAT, VFAT, and UFS (Universal Flash Storage)

https://linux.die.net/man/8/mount

- The **mount option** `dax` bypasses the page cache when using the file systems ext2, ext4, and XFS
  - Requires memory-like block devices like Persistent Memory (pmem)

https://www.kernel.org/doc/Documentation/filesystems/dax.txt
https://lwn.net/Articles/787233/

- File systems that make use of the **FUSE** (Filesystem in Userspace) module offer `direct-io` mode where the page cache is bypassed

https://www.kernel.org/doc/Documentation/filesystems/fuse-io.txt

Howto implement Write-through mode? (3/3)

- The **Windows** function call CreateFileA (fileapi.h) for creating or opening a file offers the flags FILE_FLAG_WRITE_THROUGH and FILE_FLAG_NO_BUFFERING
  - Setting them both specifies that write operations bypass the Windows system cache and are immediately flushed to the drive

https://devblogs.microsoft.com/oldnewthing/20210729-00/?p=105494
https://learn.microsoft.com/en-us/windows/win32/api/fileapi/nf-fileapi-createfilea

Additional information about how to enable write-through in different operating systems

https://github.com/ronomon/direct-io
https://ext4.wiki.kernel.org/index.php/Clarifying_Direct_IO%27s_Semantics

Fragmentation

- A cluster can only be assigned to a single file
  - If a file is bigger than a cluster, the file is split and stored in several clusters
    - **Fragmentation** means that logically related clusters are not located physically next to each other
  - Objective: Avoid frequent movements of the HDD's arms
    - If the clusters of a file are distributed over the HDD, the heads need to perform more time-consuming position changes when accessing the file
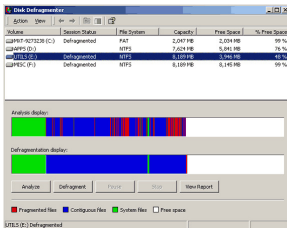   - For SSDs the position of the clusters is irrelevant for the latency
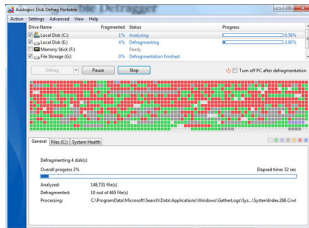

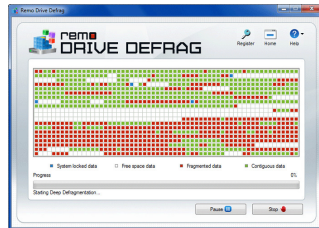Image source: http://windowsitpro.com   Image source: http://www.teknobites.com   Image s.: http://www.remosoftware.com

Defragmentation (1/3)

- These questions are frequently asked:
  - Why is it for Linux/UNIX not common to defragment?
  - Does fragmentation occur with Linux/UNIX?
  - Is it possible to defragment with Linux/UNIX?

- First of all, we need to answer: What do we want to achieve with **defragmentation**?
  - Writing data to a drive, always leads to fragmentation
    - The data is no longer contiguously arranged
  - A continuous arrangement would maximize the **continuous forward reading** of the data because no more seek times occur
  - Only if the seek times are huge, defragmentation makes sense
    - With operating systems, which use only a little amount of main memory for caching HDD accesses, high seek times are very negative

Discovery 1: Defragmentation accelerates mainly the continuous forward reading

Defragmentation (2/3)

- Single-tasking operating systems (e.g. MS-DOS)
  - Only a single application can be executed
    - If this application often hangs, because it waits for the results of read/write requests, this causes a poor system performance

> Discovery 2: Defragmentation may be useful for single-tasking operating systems.
> In practice, however, single-tasking operating systems are seldom used

- Multitasking operating systems
  - Multiple programs are executed at the same time
    - **Applications can almost never read large amounts of data in a row without other applications in between requesting r/w operations**
- In order to prevent programs, which are executed at the same time, from interfering with each other, operating systems read more data than is requested
  - The system reads a stock of data into the **cache**, even if no requests for these data exist

> Discovery 3: In multitasking operating systems, applications can almost never read large amounts of data in a row

Defragmentation (3/3)

- Linux systems automatically hold data in the cache, which is frequently accessed by the processes
    - **The impact of the cache greatly exceeds the short-term benefits, which can be achieved by defragmentation**

Discovery 4: Defragmenting has mainly a benchmark effect
Discovery 5: Enlarge the file system cache brings better results than defragmentation

- Defragmenting has mainly a **benchmark effect**
    - In practice, defragmentation (in Linux!) causes almost no positive impact
    - Tools like defragfs can be used for Linux file system defragmentation
        - Using these tools is often not recommended or useful

Helpful source of information: http://www.thomas-krenn.com/de/wiki/Linux_Page_Cache