On-disk filesystem structures

Jan van Wijk

Filesystem on-disk structures for
FAT, HPFS, NTFS, JFS, EXTn and ReiserFS

FSYS - Software

DFSee

Presentation contents

- Generic filesystem architecture
  - (Enhanced) FAT(32), File Allocation Table variants
  - HPFS, High Performance FileSystem (OS/2 only)
  - NTFS, New Technology FileSystem (Windows)
  - JFS, Journaled File System (IBM classic or bootable)
  - EXT2, EXT3 and EXT4 Linux filesystems
  - ReiserFS, Linux filesystem

Who am I ?

Jan van Wijk

- Software Engineer, C, C++, Rexx, PHP, Assembly
- Founded FSYS Software in 2001, developing and supporting DFSee from version 4 to 16.x
- First OS/2 experience in 1987, developing parts of OS/2 1.0 EE (Query Manager, later DB2)
- Used to be a systems-integration architect at a large bank, 500 servers and 7500 workstations
- Developing embedded software for machine control and appliances from 2008 onwards

Home page: ***https://www.dfsee.com/***

Information in a filesystem

- Generic volume information
  - Boot sector, super blocks, special files ...
- File and directory descriptive info
  - Directories, FNODEs, INODEs, MFT-records
  - Tree hierarchy of files and directories
- Free space versus used areas
  - Allocation-table, bitmap
- Used disk-sectors for each file/directory
  - Allocation-table, run-list, bitmap

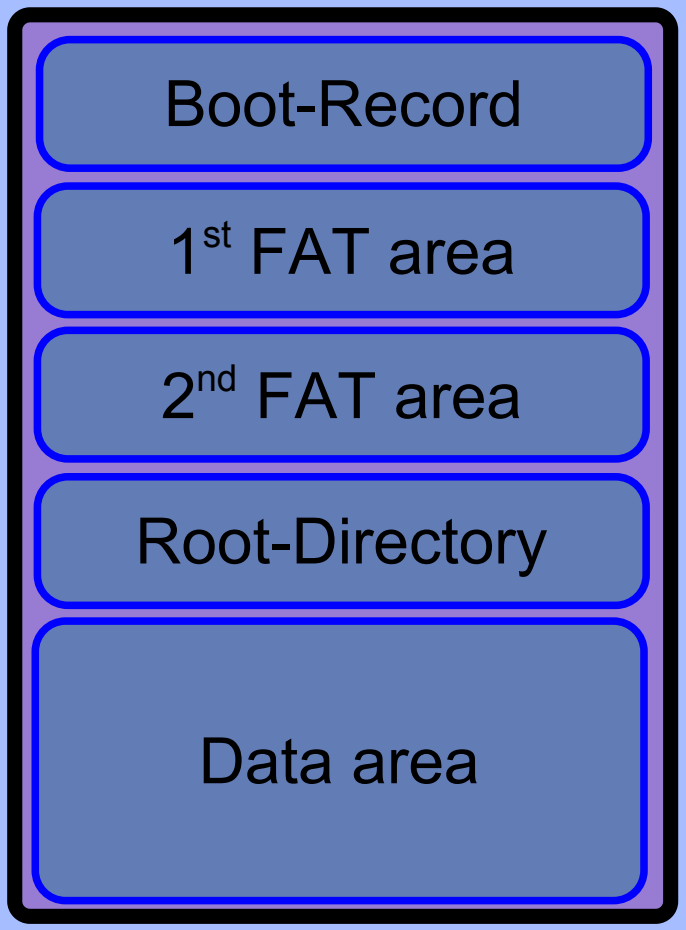
File Allocation Table

- The FAT filesystem was derived from older CPM filesystems for the first (IBM) PC
- Designed for diskettes and small hard disks
- Later expanded with subdirectory support to allow larger hierarchical filesystems

- Supported natively by the OS/2 kernel and almost any other modern operating system

- OS.2 (and Windows) enhancements usually implemented in installable filesystems like FAT32.IFS and VFAT.IFS

FAT(12/16) Volume layout



- Bootsector, bootcode, labels and geometry/size info (BPB)
- File Allocation table, 12/16 bits for every cluster in the volume
- Exact duplicate of 1st FAT

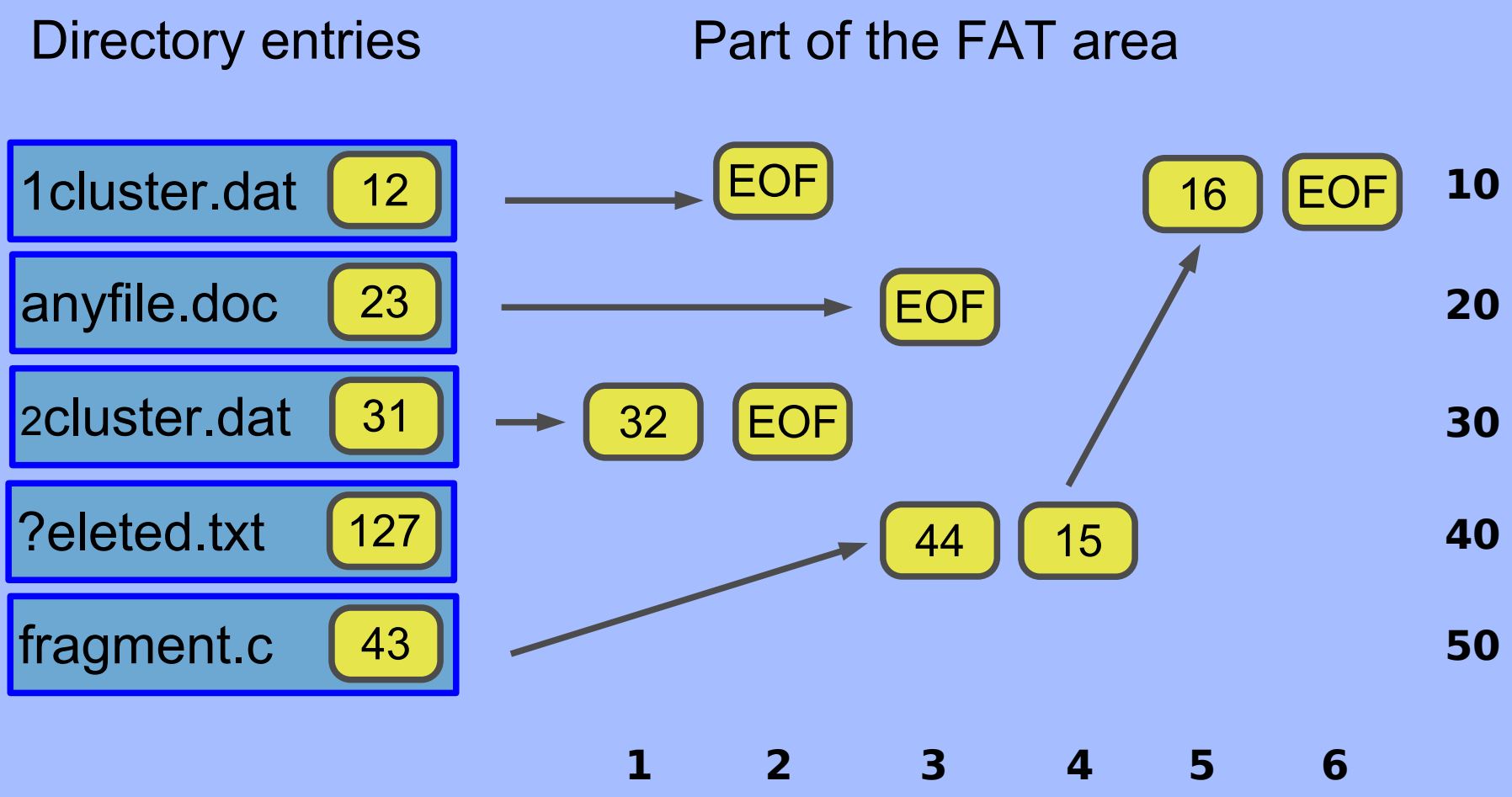
- Fixed size, fixed position

- First data located at cluster 2
- Has clusters of filedata as well as clusters with sub-directories

The Allocation Table

- The actual File Allocation Table has ONE value for every allocation unit (cluster), values are:
  - Free, the cluster is NOT in use, value is 0 (zero)
  - 2 .. max, location of the NEXT cluster in the chain
  - EOF, end of file, this is the last cluster in the chain
  - BAD, the cluster is unusable due to bad sectors
- Each value can be 12 bits, 16 bits or 32 bits depending on volume and cluster size.
- A directory entry points to the FIRST cluster of an 'allocation chain' representing each cluster used by this file or directory, ending in an EOF

FAT Allocation Chain



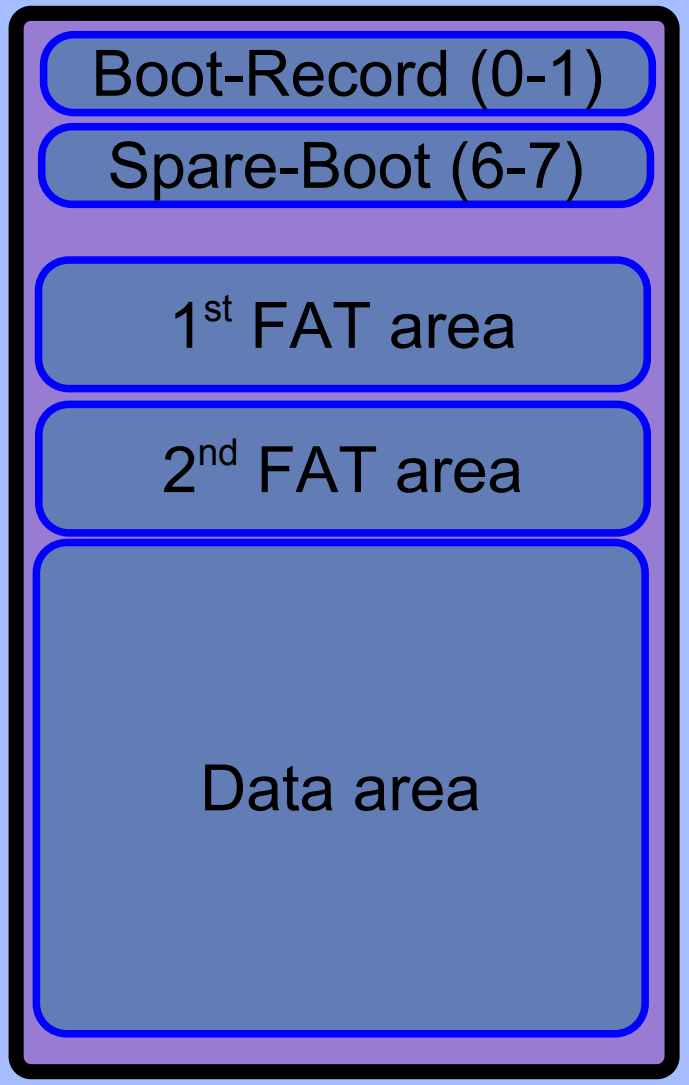
FAT(32) directory entries

- A basic FAT directory entry contains:
  - 8 character BASE filename
  - 3 character file extension
  - 1 byte attribute with RO, System, Hidden etc
  - 4 byte date and time information
  - 2 bytes (16-bit) cluster number for FIRST cluster
  - 4 bytes (32-bit) file size, maximum value 2 Gb
- OS/2, FAT32 and VFAT may add:
  - 2 bytes index value to OS2 extended-attributes
  - 2 bytes extra cluster number, making it 32-bit
  - Extra create/access date and time fields (VFAT)
  - Long Filename, storing a UNICODE filename up to 255 characters in entries preceding the regular one (used in FAT32, and possibly VFAT)

Common problems with FAT

- Combined file-allocation and free space administration (no redundancy) may cause:
  - Lost clusters, allocated but no directory link
  - Cross-links, clusters that are in more than 1 chain
  - Undelete will be UNRELIABLE for fragmented files because the cluster allocation is unknown after the file is erased. (clusters will be marked FREE)
- OS/2 specific EA related problems:
  - stored in one huge file “EA DATA . SF”
  - The EA's for each file take up a full cluster in that file
  - Linked from an index in the FAT directory entry, can be damaged by other OS's or defragmenters

FAT32 Volume layout



- Boot sector, boot code, label, geometry and size info (BPB). Location of Root directory, free space size
- File Allocation table, 32 bits for every cluster in the volume
- Exact duplicate of 1st FAT

- First data is located at cluster 2 (and often is the Root directory)
- Has clusters of file data as well as clusters with directories

- Windows implementation limit: only 28 bits of the 32 are used

Enhanced/extended FAT filesystem

- Designed for HUGE removable media and fast writing of large files (Video, Photo)
- Uses a separate allocation BITMAP file
- FAT entries are valid only for the fragmented files!
- Does NOT have a 'short' 8.3 filename!
- A journaled version (TexFAT) exists too, probably Win-CE (embedded use) only
- Mandatory on SD-cards over 32Gb (SDXC)

Enhanced FAT features and limits

- 64-bit partition/file size, allows HUGE files
- Uses all 32 bits for FAT entries (FAT32: 28)

- Sector size 512 to 4096 bytes supported
  - Sector and Cluster size are recorded in boot-area

- Cluster size up to 32 MiB (FAT32: 64 KiB)
  - Larger clusters means smaller bitmap/FAT => FASTER

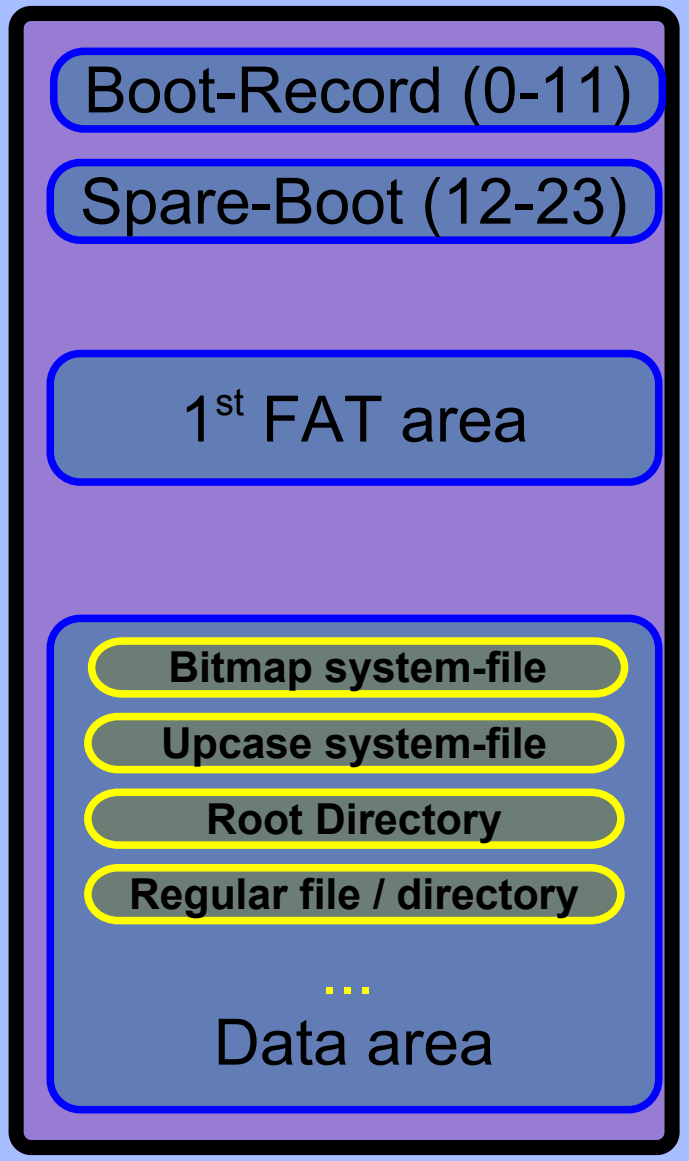
- Create, Modify and Access time, in mSec

- Max 256 MiB directories (> 2 million files)
  - Directory entries can include a 'name-hash' to speed up searching in huge directories dramatically

Enhanced FAT directory entries

- NOT compatible with other FAT filesystems!
- Each 32-byte entry has a specific 'type'
- Files/Directories have multiple entries:
  - FILE entry: Attributes and date/time info, checksum
  - STREAM entry: name-length, file sizes, first-cluster
  - NAME entry: Name (fragment) up to 15 unicode chars
- Several other special purpose types exist:
  - LABEL entry: Volume label, up to 11 unicode chars
  - BITMAP entry: Flags + Cluster for Bitmap system file
  - UPCASE entry: Flags + Cluster for Upcase system file
  - VOLGUID entry: 16 bytes for a Volume GUID string
  - PADDING entry: TexFAT (journaled), Win-CE only?
  - ACT entry: Access Control Table, Win-CE only

Enhanced FAT Volume layout



- Boot sector, boot code, (cluster) size info. Root directory cluster, OEM-area, boot checksum
  (Followed by FAT-area 'alignment gap')
- File Allocation table, 32 bits for every cluster in the volume
- 2nd FAT optional, duplicate of 1st
  (Followed by data area 'alignment gap')
- First data located at cluster 2
- Has clusters of (system) file data and clusters with directories
- Bitmap and Upcase table located in data area as 'system files'
- Bitmap, Upcase, Root directory can be located anywhere!

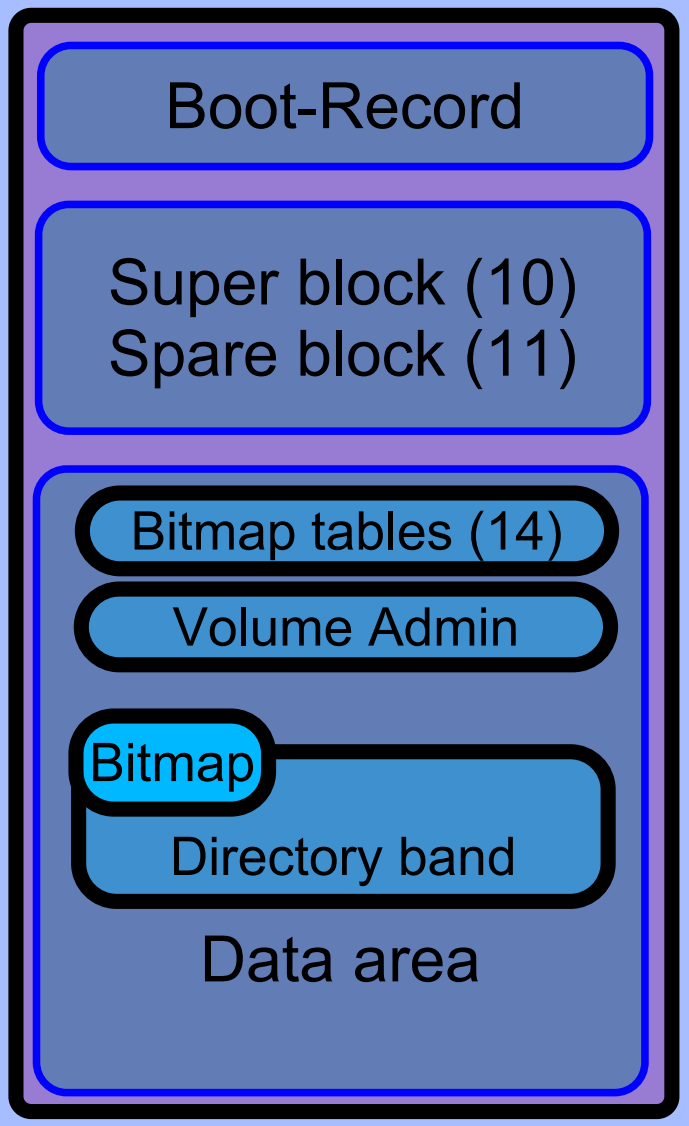
High Performance File System

- Designed by MS and IBM to overcome the shortcomings of the FAT filesystem
- Based on UNIX-like Fnodes and B-trees
- Designed for larger hard disks (> 100 MiB)
- More redundancy, less sensitive to crashes
- B-trees, fragmentation is less of a problem
- Implemented as Installable Filesystem with dedicated caching (HPFS.IFS, HPFS386.IFS)

HPFS Features and limits

- FS-size up to 2 terabyte (2048 GiB) by design
- OS/2 implementation limit of 64 GiB due to shared cache design (5 bits of 32 for cache use)
- Allocation in single 512-byte sectors
- Filename maximum length of 254 characters
- Support for multiple codepages for filenames
- B-trees used for allocation and directories
- Multi-level cache: Paths, Directories and Data

HPFS Volume layout



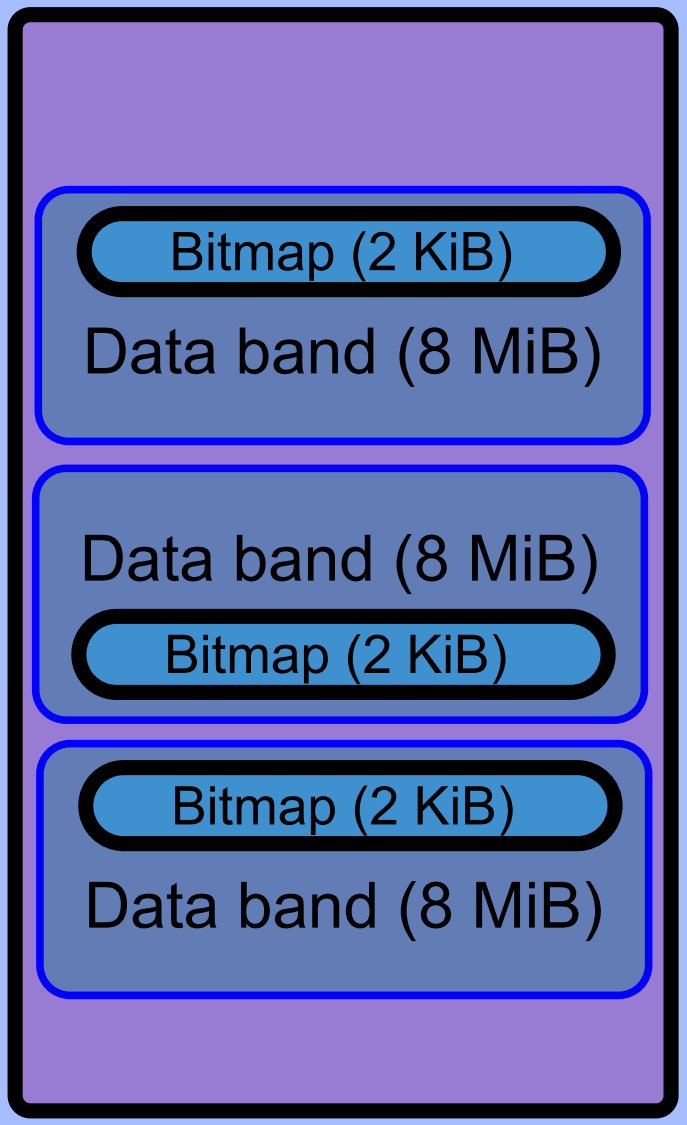
- Boot sector with HPFS boot code
- Fixed volume information pointer to Root directory
- Variable volume information

- Division in 8 MiB data bands
- Codepage, Hotfix, Spare etc

- Preallocated DIR-blocks, 1% in middle of volume (max 800 Mb)
- Separate Directory BITMAP

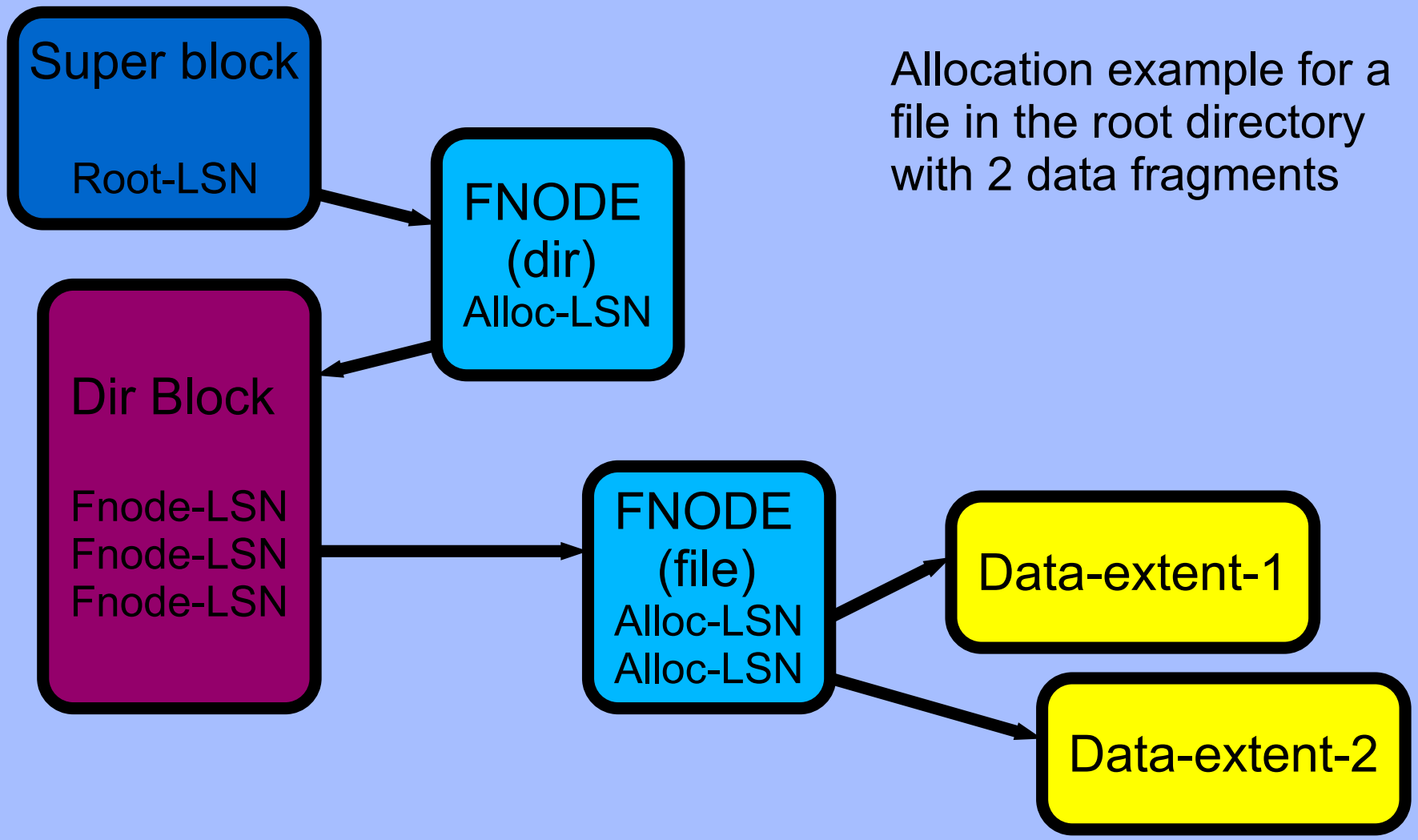
- File data + extra allocation and directory blocks when needed

HPFS data-bands layout



- Data Bands:
  - Area of a FIXED size of 8 MiB (128 per gigabyte partition size)
  - Each has a free space BITMAP that is located at the start or at the end (alternating) so they are back-to-back
  - Maximum UNFRAGEMENTED file size is limited to nearly 16 MiB because of the bitmaps located within each band

HPFS File allocation



Allocation example for a file in the root directory with 2 data fragments

HPFS Fnode layout

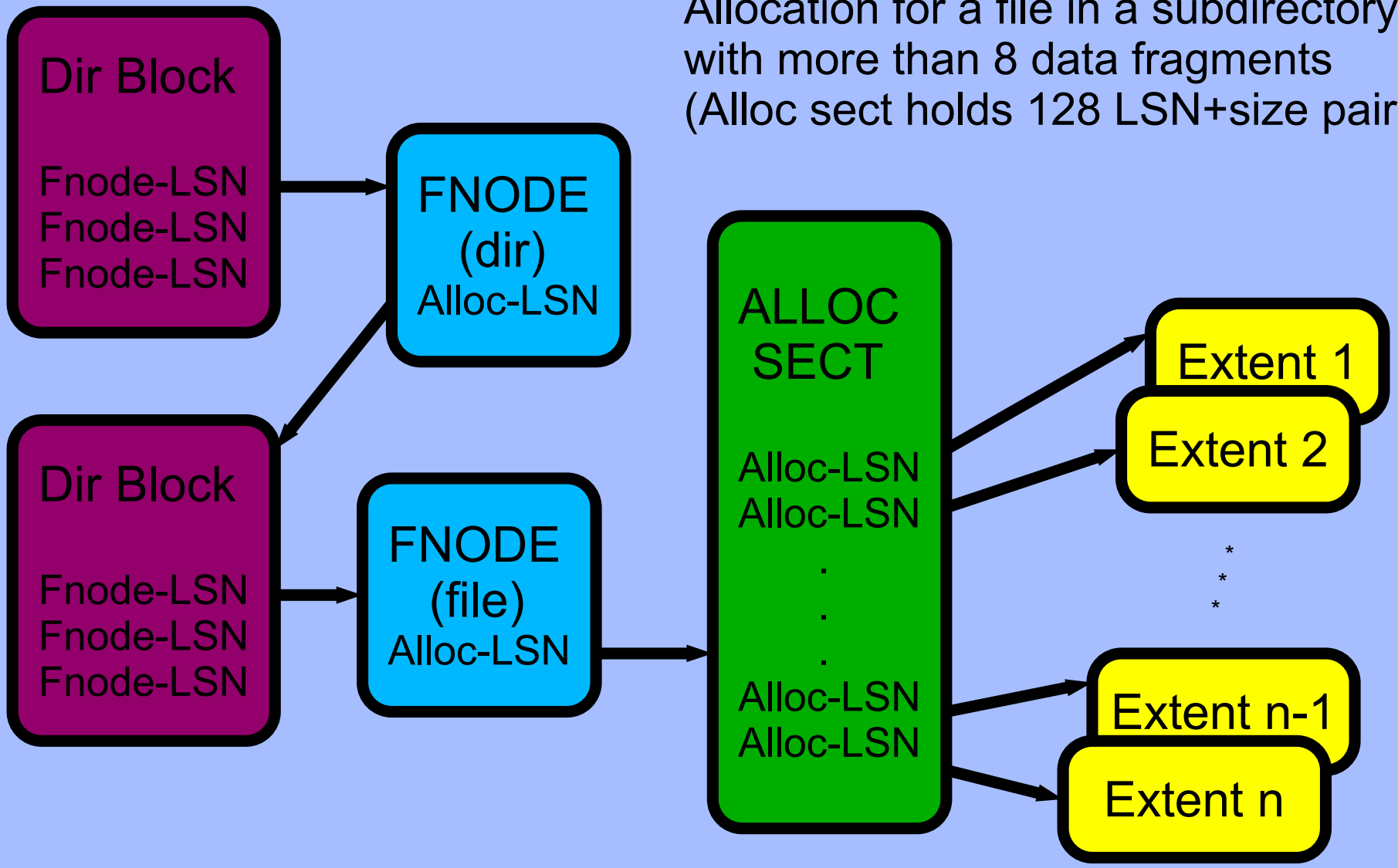
- An Fnode is 512 bytes with fixed size info:
  - Unique binary signature string 'ae 0a e4 f7'
  - Sector number (LSN) for Parent directory
  - First 15 characters of the filename (for undelete)
  - Length of filename, and length of the file data
  - Type of the Fnode, either File or Directory
  - Allocation information, max of 8 LSN+size pairs
  - DASD limits (user quota, HPFS386 only)
- Then, variable sized info may be present, either in the Fnode itself or externally:
  - Extended-attribute data (.longname, .icon etc)
  - Access Control Lists (HPFS386 only)

HPFS DirBlock layout

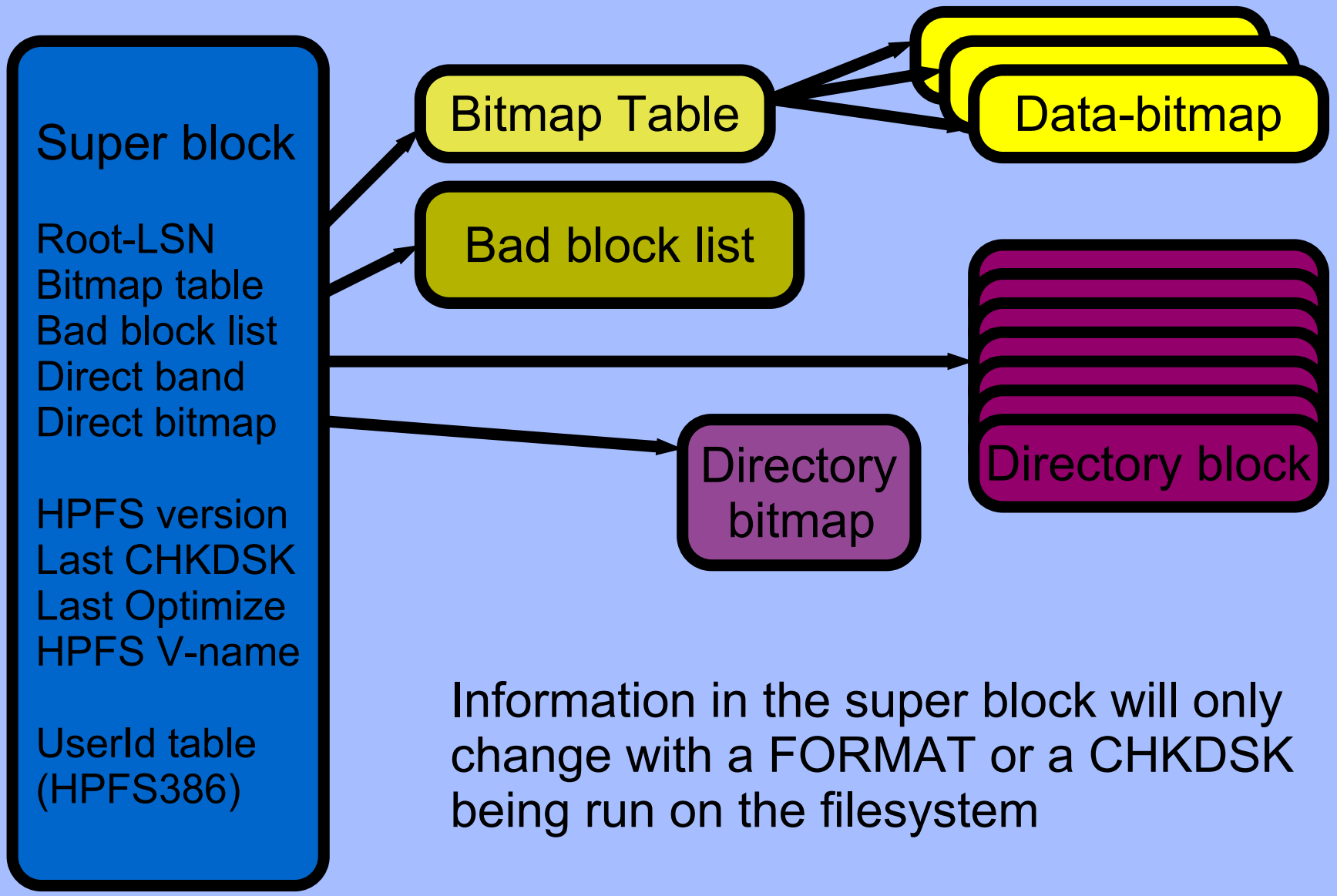
- A DirBlock is 2048 bytes with fixed size info:
  - Unique binary signature string 'ae 0a e4 77'
  - LSN for Parent and type Fnode or DirBlock (B-tree)
  - Sector number for THIS Directory-Block
  - Number of changes since creation of the block
- Then, variable sized Directory info with:
  - A B-tree 'down' pointer (DirBlock LSN), OR
  - Three date/time fields creation, modify, last access
  - The standard (FAT, SHRA) attributes
  - File data length and extended-attribute length
  - Codepage number to use with the filename
  - Variable sized filename, max 254 characters

HPFS Fragmented File

Allocation for a file in a subdirectory
with more than 8 data fragments
(Alloc sect holds 128 LSN+size pairs)

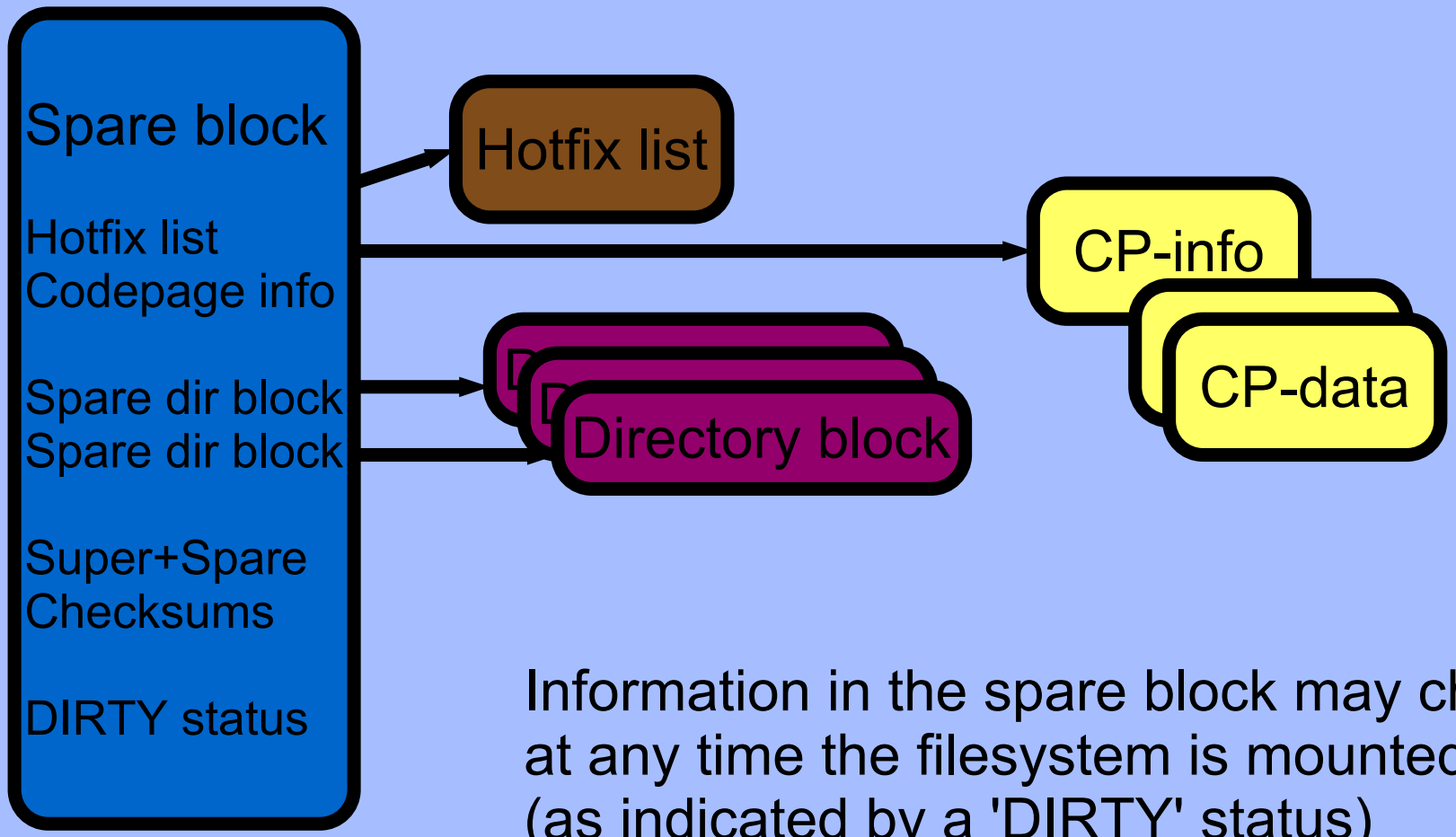


HPFS Superblock info



Information in the super block will only change with a FORMAT or a CHKDSK being run on the filesystem

HPFS Spareblock info



Information in the spare block may change at any time the filesystem is mounted (as indicated by a 'DIRTY' status)

New Technology File System

- Design started as new FS for OS/3 (32-bit OS/2) before that was renamed to Windows NT
- Organization is like a database, everything, including the FS administration itself is a FILE represented by an entry in the Master File Table (MFT)
- Can handle extreme sizes due to 64 bit values used
- All data is represented by attribute values, with the file data being the 'default data attribute'.
  Supports multiple named data streams for a single file.
- Has native support for OS/2 EA's (as MFT attribute)

NTFS limits

- FS-size upto 2^64 clusters by design
  - Some tools limited to 2048 GiB due to use of 32 bits for sector or cluster numbers
- Allocation in clusters of typically 8 sectors
- MFT record typical size is 1 KiB
  - May hold all data for small files. Larger attributes are stored externally, using runlists for the allocated space
- Filename of unlimited length, but limited by the OS itself to a length of 254 characters

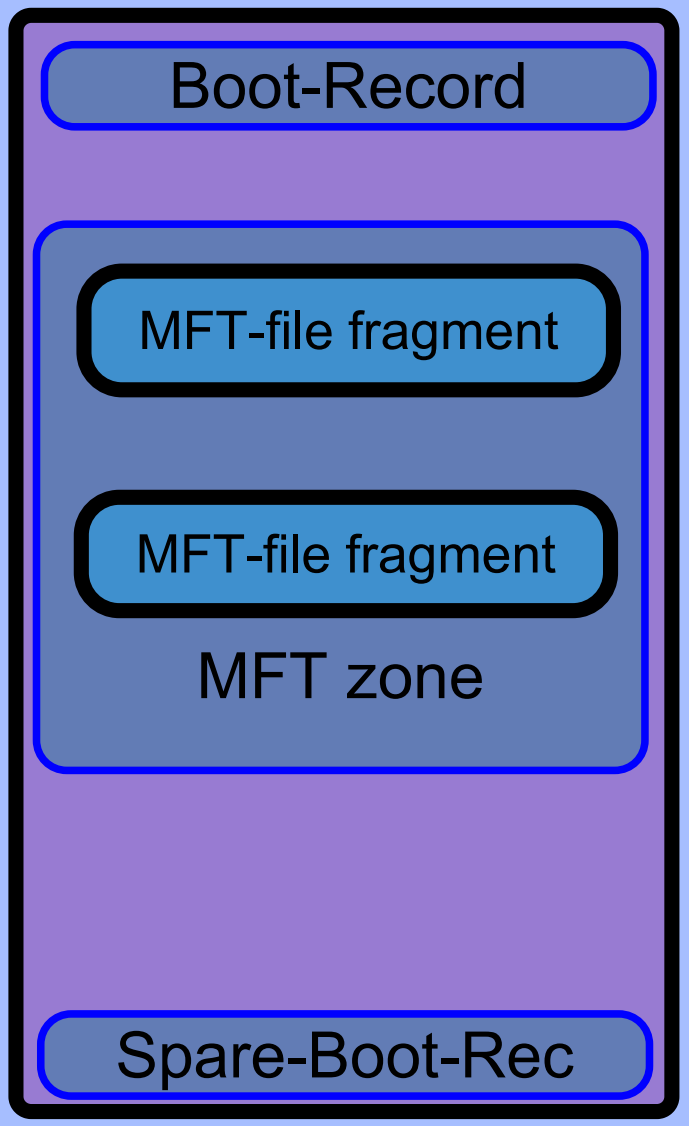
NTFS Features

- Uses UNICODE for filenames to allow for any character set (like codepages in HPFS)
- Usually includes a second ASCII 8.3 name too

- The FS keeps a transaction-LOG of all changes to the FS-structures to allow quick recovery and guarantee a consistent filesystem.

  - This makes it a ***journaling*** filesystem
  - File data itself is NOT part of the journal, so may get lost/damaged after a crash!

NTFS Volume layout



- Boot sector with NTFS boot code
- Some fixed volume-information, pointer to MFT and MFT-spare
- MFT zone is reserved to reduce fragmentation of the MFT, but will be used for data if FS gets full
- MFT itself is a regular file, so CAN and WILL get fragmented
- Rest of space is for all external attributes (file data), not stored in the MFT records themselves ...

NTFS special files

- 0 = $MFT — Main MFT file, all files/dirs
- 1 = $MFTmirr — Mirror MFT file, 1st 4 entries
- 2 = $LogFile — Journalling log file
- 3 = $Volume — Global volume information
- 4 = $AttrDef — Definitions for attribute values
- 5 = \ — Root directory
- 6 = $Bitmap — Allocation bitmap
- 7 = $Boot — Boot record (8 KiB at sect 0)
- 8 = $BadClus — Bad cluster administration
- 9 = $Secure — Global Security information
- A = $Upcase — Collating and uppercase info
- B = $Extend — Extended info (NTFS 5, XP)

MFT special file remarks

- Special files upto MFT-A are fixed, and standard
- MFT B represents a directory with (for XP):
  - $ObjId — Object identification data
  - $Quota — User space restriction data
  - $Reparse — Reparse points, aliases in the filesystem, much like Unix/Linux soft-links (or WPS shadows)
- MFT numbers up to around 1A are reserved for system file use by the FS itself, after that the first user files will appear

MFT record layout

- The MFT record is of a fixed size (1 KiB) that starts with a fixed header containing:
  - Unique signature string 'FILE'
  - Sequence, generation and 'fixup' information
  - Offset to first dynamic attribute in the record (0x38)
  - Type of the MFT-record, either File or Directory
- After this a dynamic list of variable sized attributes follows, these can be either:
  - Internal (Self contained) when small
  - External, using an allocation run-list pointing to one or more clusters being used for the data
  - Sparse, like external, but with 'empty' parts that do NOT take up any space yet, on the disk itself

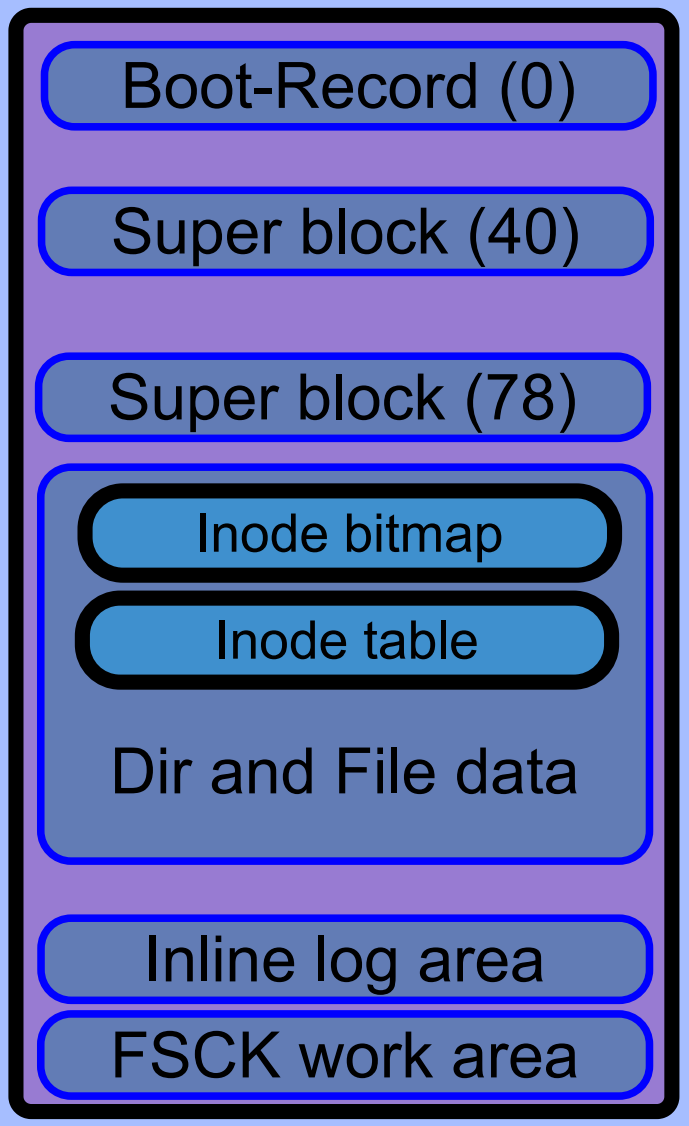
MFT attributes (from $AttrDef)

- 10 = $STANDARD_INFORMATION
- 20 = $ATTRIBUTE_LIST (group of attributes)
- 30 = $FILE_NAME
- 40 = $OBJECT_ID
- 50 = $SECURITY_DESCRIPTOR
- 60 = $VOLUME_NAME
- 70 = $VOLUME_INFORMATION
- 80 = $DATA (default or named data stream)
- 90 = $INDEX_ROOT (B-tree root, directories)
- A0 = $INDEX_LOCATION
- B0 = $BITMAP
- C0 = $REPARSE_POINT
- D0 = EA_INFORMATION
- E0 = EA (actual OS/2 extended attribute data)
- 100 = LOGGED_UTILITY_STREAM

Journaled File System

- Designed by IBM for its AIX operating system
- Based on UNIX-like structure with journaling and multiple storage area capabilities
- Ported to an OS/2 IFS by IBM to allow huge expandable filesystems with good performance and journalling (fast crash recovery)
- Port released as 'open source' for Linux too
  (Note: With a few additions, not 100% compatible!)
- Relies on LVM for some of its functionality

JFS Volume layout



- Boot sector, standard (label etc)
- JFS specific volume data with pointers to lots of info :-)
- Duplicate of main super block
- Actual contents is grouped in 'aggregates' of fixed size holding Inode tables and file data
- The 'journal' file area
- Temporary space for CHKDSK

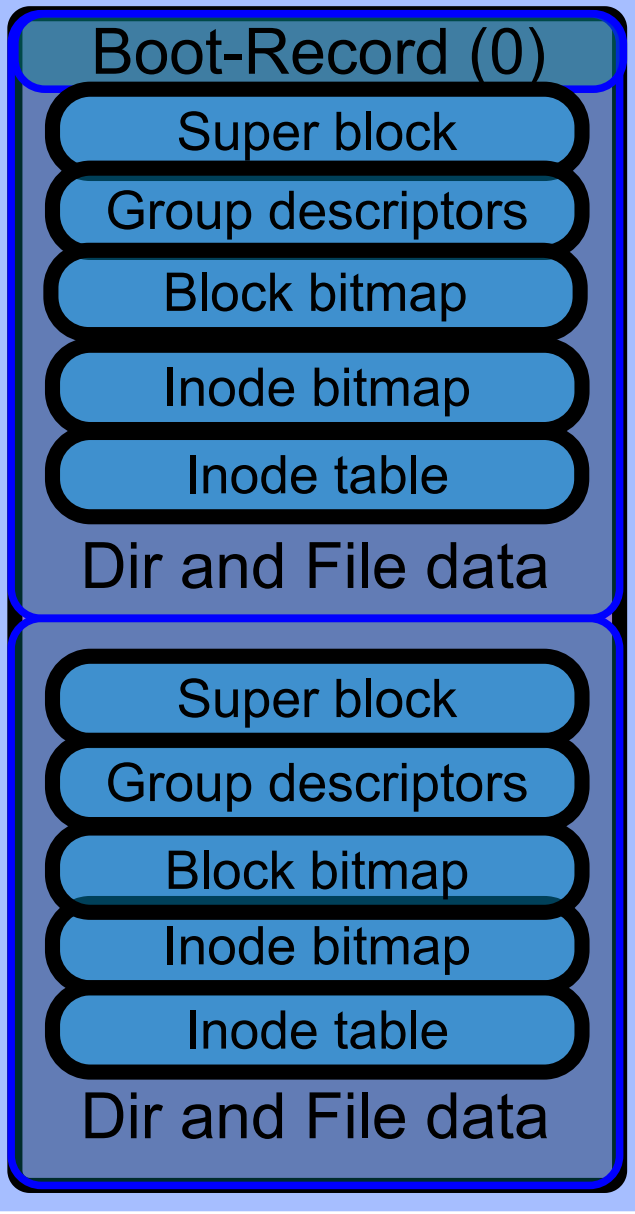
Extended 2nd FS, Ext2, Ext3, Ext4

- Designed by the Linux community, based on UNIX-like structures with many optimizations for speed and several new features
- No current port for OS/2 (LVM compatible)
- Like JFS and other Unix derivates, there is NO redundant filename info in the Inodes, making file recovery much more difficult.
- EXT3 adds a journalling file to EXT2
- EXT4 adds many features, and raises size limits

EXT2/3/4, Directories and Inodes

- Directories are ordinary files, containing a mapping between filenames and Inodes.
- There can be more than one directory entry pointing to the SAME Inode! (hard links)
- The Inode contains file attributes including ownership and a lists of allocated blocks.
  - 12 direct blocks, for files of up to 12 blocks
  - Indirect, double indirect and triple-indirected blocks
  - Ext4 may also use 'extents', much like runlists

EXT2/3/4 Volume layout

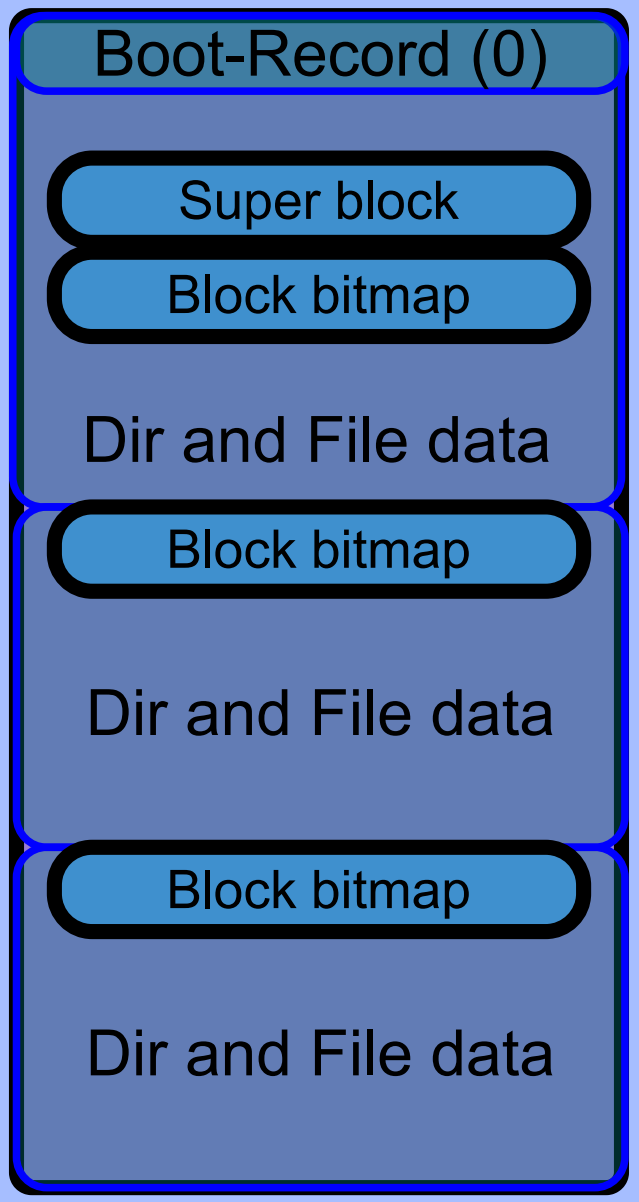


- Boot sector, normally empty may contain GRUB(2) or LILO (is at start of the 1st block)
- Volume divided up in block-groups with identical layout, each having:
  - A super block copy, can be sparse, with less copies of the super block
  - Group description data
  - Allocation bitmap for this group
  - Usage bitmap for the inodes
  - Fixed size Inode table for the group
  - Rest of group are data blocks
  - Ext4 may concentrate part of the info in specific groups, for performance and to allow larger contiguous files

ReiserFS

- Designed by Hans Reiser
- Based on a database model using a single large tree of information 'nodes'.
- The keys for the nodes uniquely identify them and also determine the sequence in the file
- Space efficient since the nodes are variable in size, and blocks can be filled up to 100% (blocks may contain data for multiple files)
- Reiser includes a journalling mechanism

ReiserFS Volume layout



- Boot sector, normally empty my contain GRUB or LILO
  - (is at start of the 1st block)
- There is just ONE super block
- Volume divided up in equal sized chunks, that can be described with a bitmap of exactly ONE block
  - (32768 blocks for 4Kb block size)
- Rest of the blocks contain tree nodes and leaves, with keys and data areas that contain directory and file data for the volume.

Questions ?