

On-disk filesystem structures

Jan van Wijk

Filesystem on-disk structures for
FAT, HPFS, NTFS, JFS, EXT2 and ReiserFS

FSYS - *software*

DFS *ee*

Presentation contents

- Generic filesystem architecture
 - FAT, File Allocation Table
 - HPFS, High Performance FileSystem
 - NTFS, New Technology FileSystem
 - JFS, Journalled File System (classic, bootable)
 - EXT2 and EXT3 Linux filesystems
 - ReiserFS, Linux filesystem

Who am I ?

Jan van Wijk

- Software Engineer, C, Rexx, Assembly
- Founded FSYS Software in 2001
- 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
- Home page: <http://www.dfsee.com>

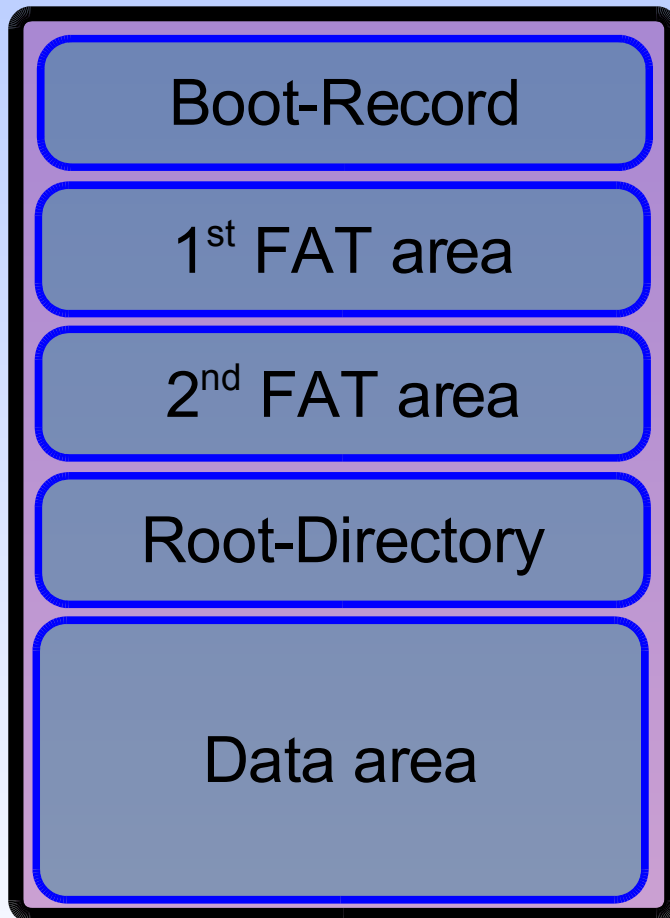
Information in a filesystem

- Generic volume information
 - Bootsector, superblocks, special files ...
- File and directory descriptive info
 - Directories, FNODEs, INODEs, MFT
 - Hierachy of files/directories
- Freespace versus used areas
 - Allocation-table, bitmap
- Used areas 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 harddisks
- Later expanded with sub-directory support to allow larger hierarchical filesystems
- Supported natively by the OS/2 kernel
- Enhancements in installable filesystems like FAT32.IFS and VFAT.IFS

FAT(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

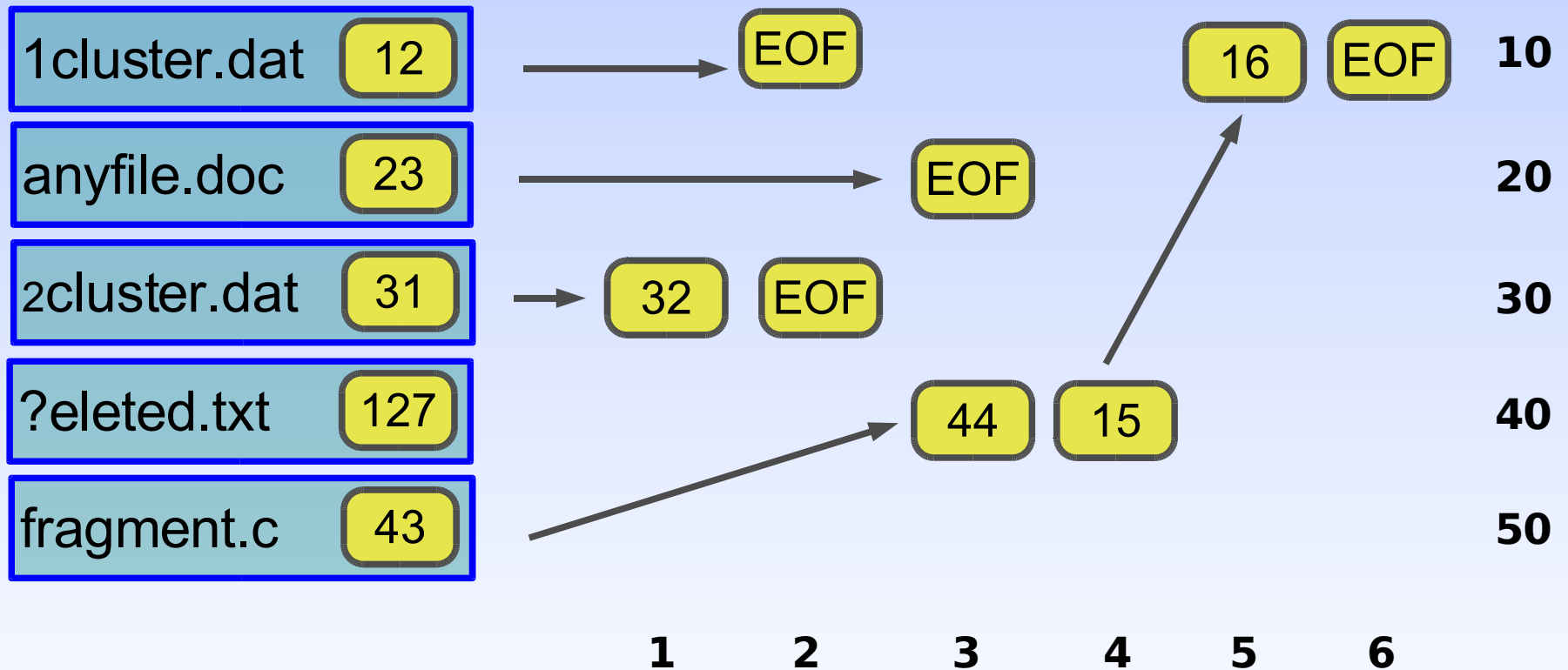
The Allocation Table

- The actual File Allocation Table has ONE value for every allocation unit (cluster):
 - 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'

FAT Allocation Chain

Directory entries

Part of the FAT area



FAT directory entries

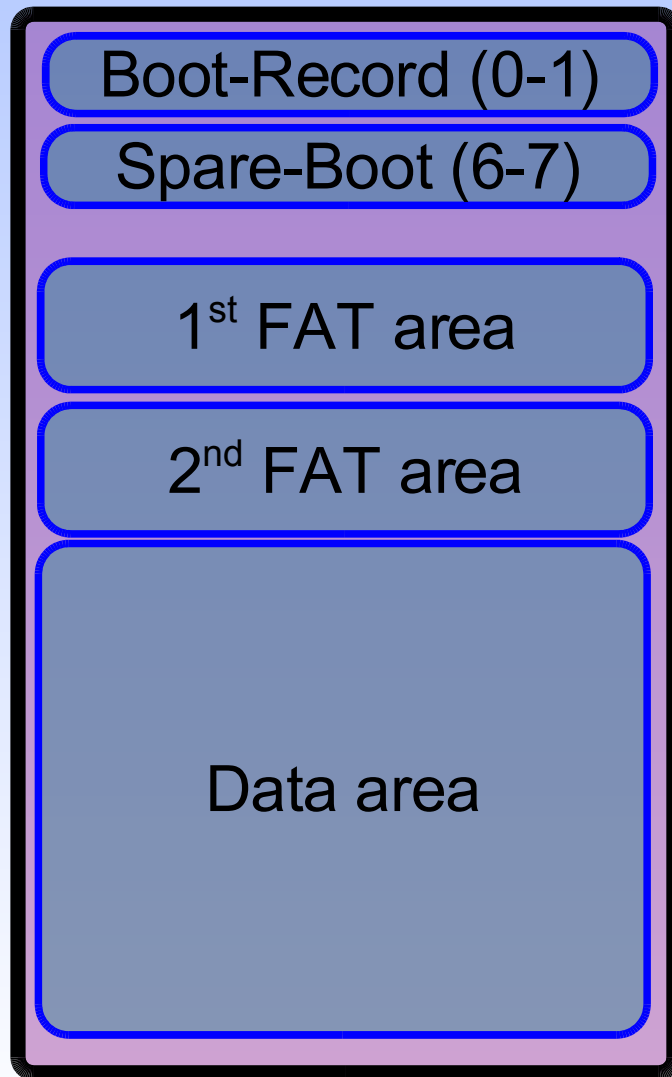
- A basic FAT directory entry contains:
 - 8 character BASE filename
 - 3 character file extension
 - 1 byte attribute with RO, System, Hidden etc
 - 4 byte date and time information
 - 2 bytes (16-bit) cluster-number for FIRST cluster
 - 4 bytes (32-bit) filesize, maximum value 2 Gb

- 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)

Common problems with FAT

- Combined file-allocation and freespace 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 marked FREE)
- OS/2 specific EA related problems:
 - stored in one huge file “EA DATA . SF”
 - Linked from an index in the FAT directory entry, can be damaged by other OS's or defragmenters

FAT32 Volume layout



- Bootsector, bootcode, label, geo and size info (BPB). Location of Root directory, freespace size
- File Allocation table, 32 bits for every cluster in the volume
- Exact duplicate of 1st FAT
- First data located at cluster 2 (usually the Root directory)
- Has clusters of filedata as well as clusters with directories

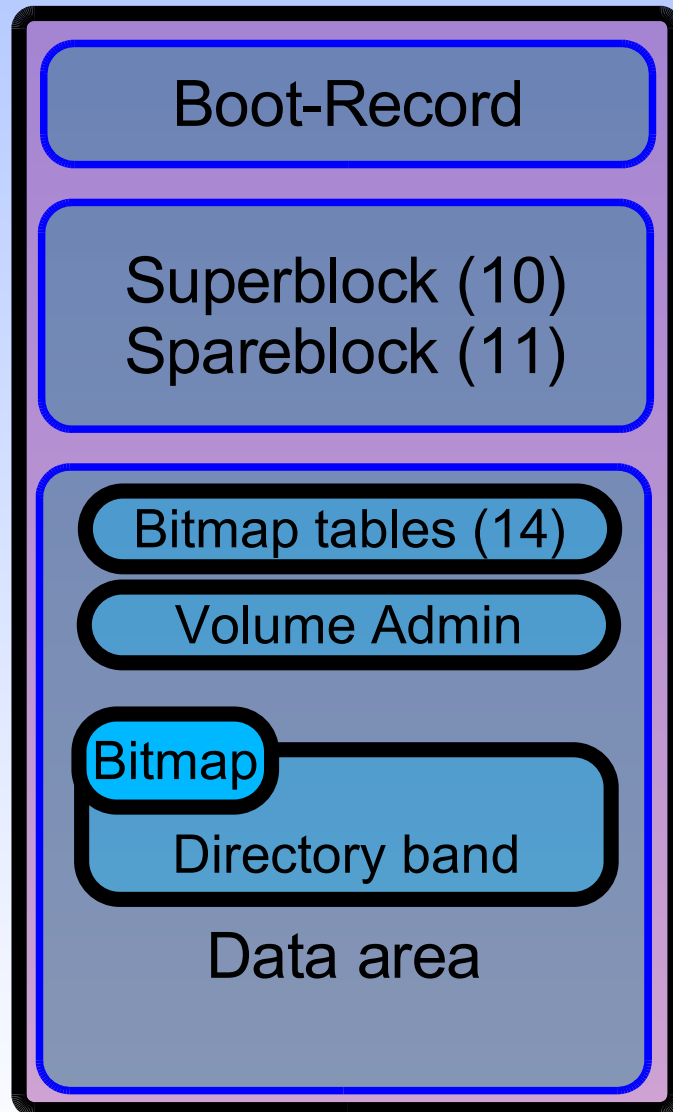
High Performance File System

- Designed by MS and IBM to overcome the shortcomings of the FAT filesystem
- Based on UNIX-like Fnodes and B-trees
- Designed for larger harddisks (> 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

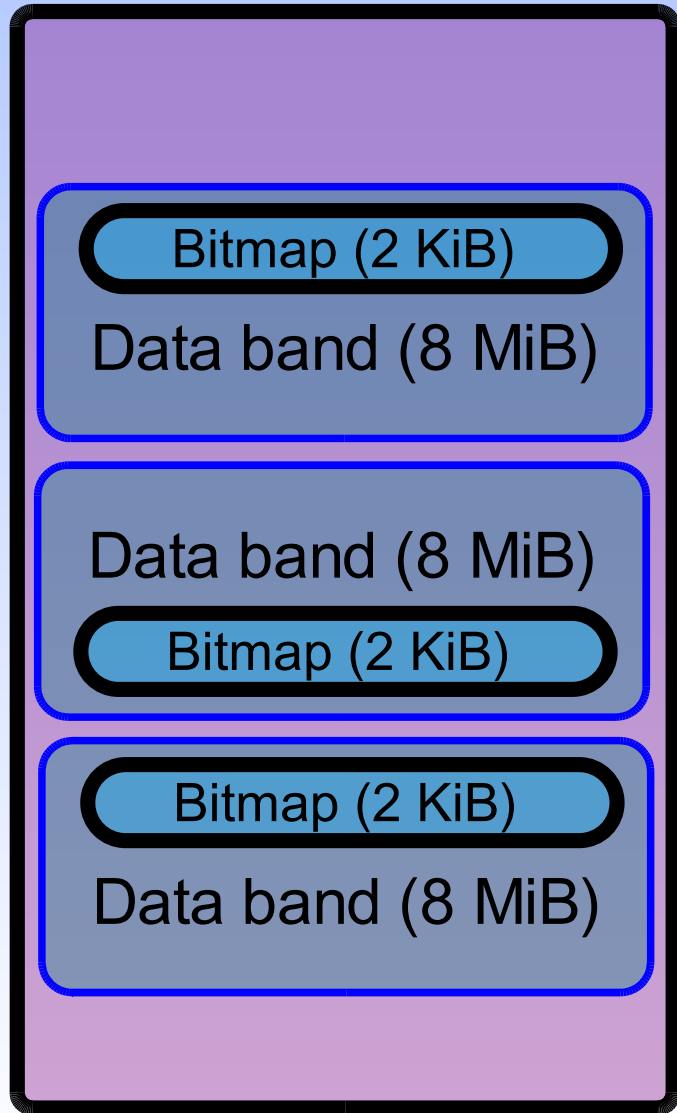
- FS-size upto 2 terabyte (2048 GiB) by design
- OS/2 implementation limit of 64 GiB due to shared cache design (5 bits of 32 for cache use)
- Allocation in single 512-byte sectors
- Filename maximum length of 254 characters
- Support for multiple codepages for filenames
- B-trees used for allocation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

HPFS Volume layout



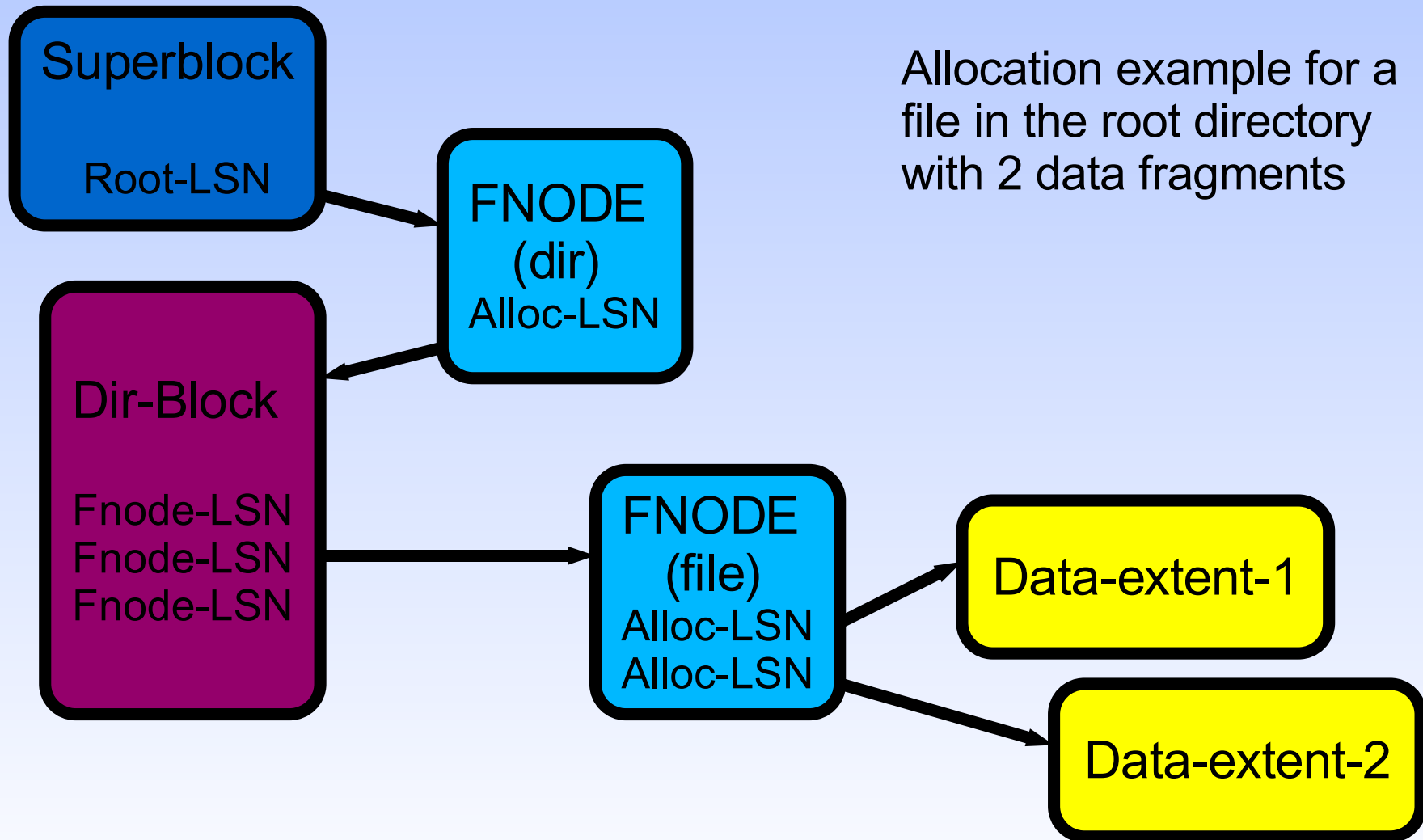
- Bootsector with HPFS bootcode
- Fixed volume-information pointer to Root-directory
- Variable volume-information
- Division in 8 MiB data bands
- Codepage, Hotfix, Spare etc
- Pre-allocated DIR-blocks, 1% in middle of volume (max 800 Mb)
- Separate Directory-BITMAP
- Filedata + extra allocation and directory blocks when needed

HPFS data-bands layout



- Data Bands:
 - Are of a FIXED size of 8 MiB (128 per gigabyte partition size)
 - Each have a freespace BITMAP that are located at the start or at the end (alternating) so they are back-to-back
 - Maximum UNFRAGMENTED filesize is almost 16 MiB

HPFS File allocation



HPFS Fnode layout

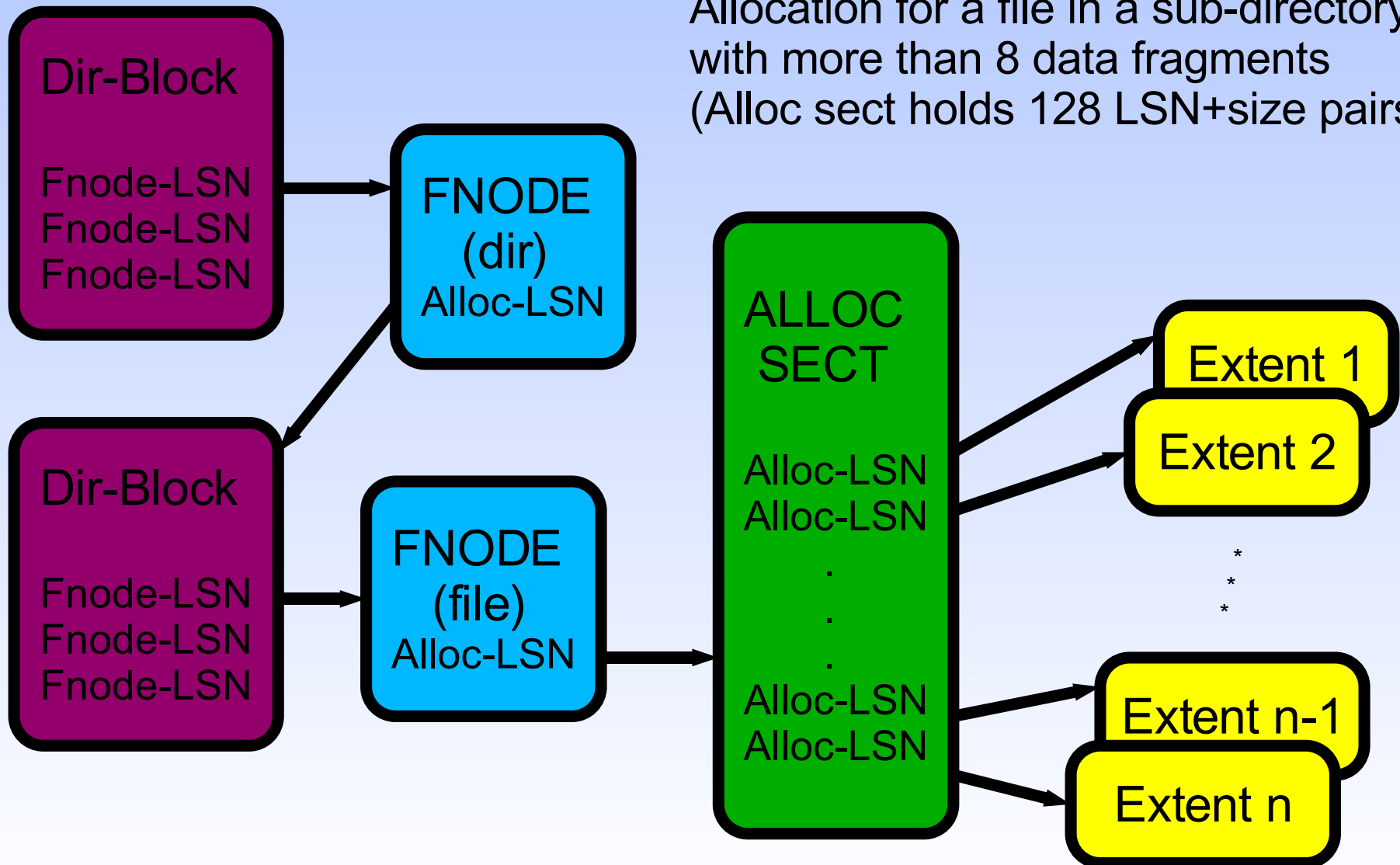
- An Fnode is 512 bytes with fixed size info:
 - Unique binary signature string 'ae 0a e4 f7'
 - Sectornumber (LSN) for Parent directory
 - First 15 characters of the filename (short name)
 - Length of filename, and length of the filedata
 - Type of the Fnode, either File or Directory
 - Allocation information, 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

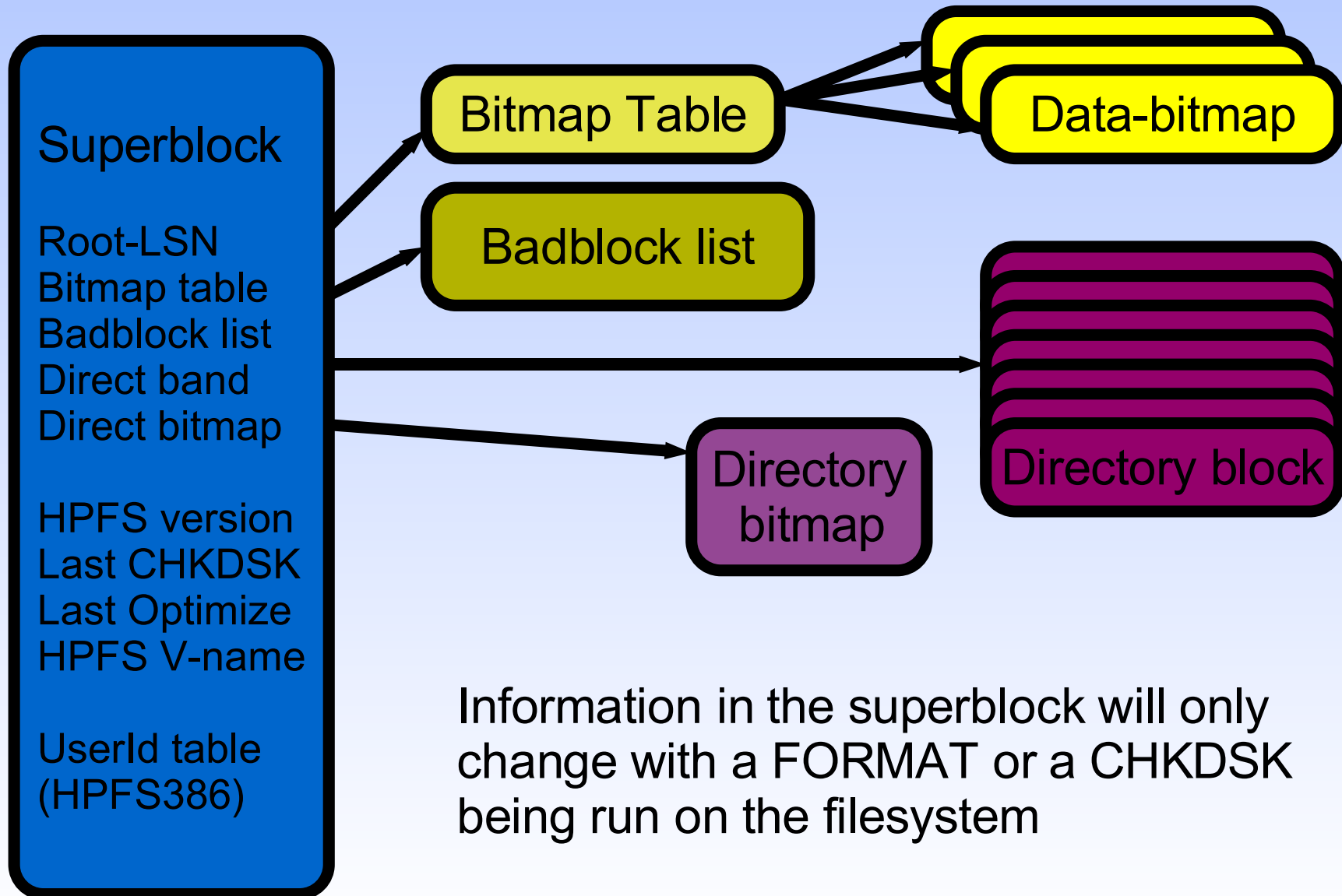
- A DirBlock is 2048 bytes with fixed size info:
 - Unique binary signature string 'ae 0a e4 77'
 - LSN for Parent and type Fnode or DirBlock (B-tree)
 - Sectornumber for THIS Directory-Block
 - Number of changes since creation of the block
- Then, variable sized Directory info with:
 - A B-tree 'down' 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

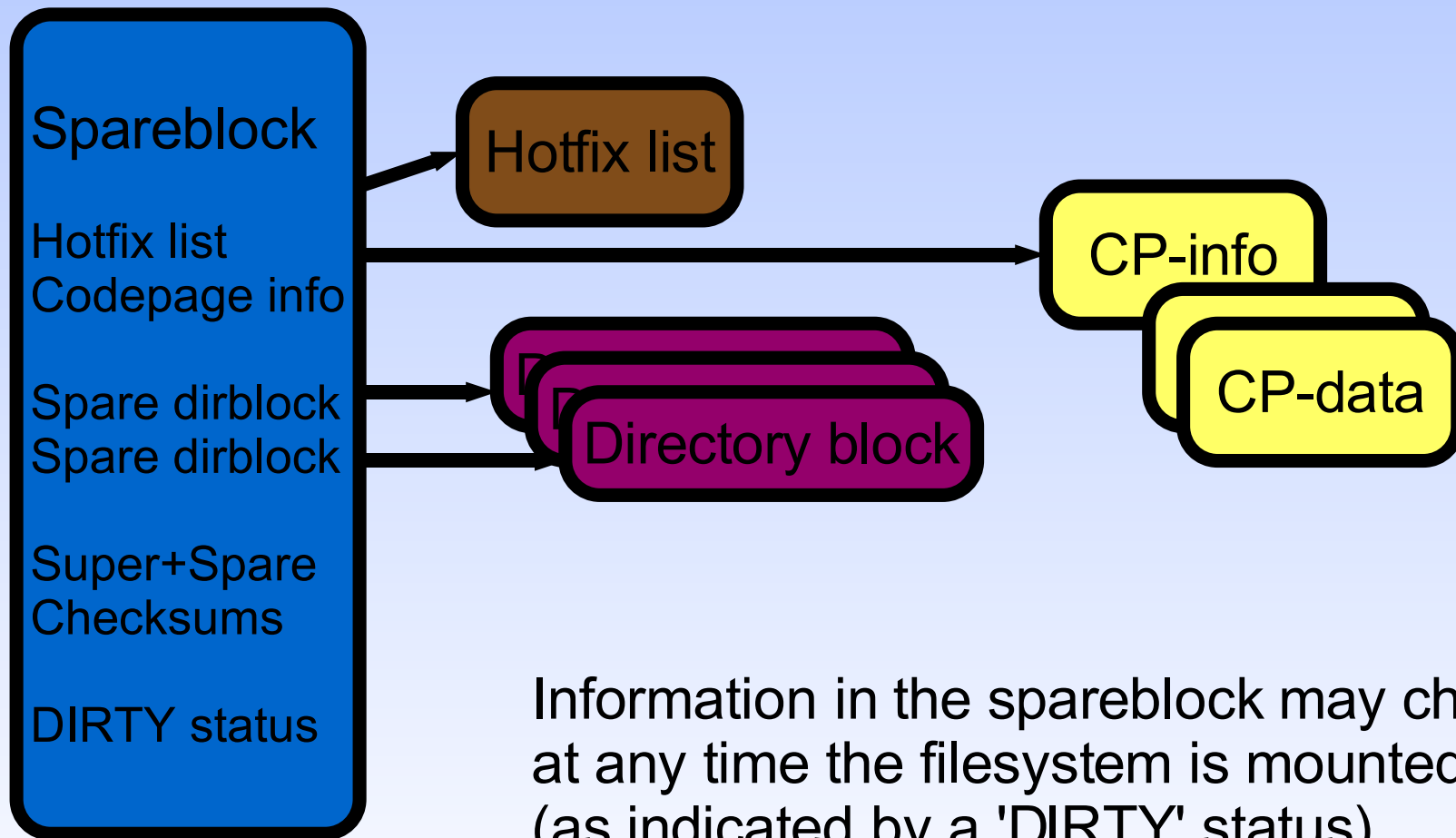
Allocation for a file in a sub-directory
with more than 8 data fragments
(Alloc sect holds 128 LSN+size pairs)



HPFS Superblock info



HPFS Spareblock info



New Technology File System

- Design started as new FS for OS/3 (32-bit OS/2) before that was renamed to Windows NT
- Organisation 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 represented by attribute values, with the data being the 'default data attribute'. Supports multiple data-streams for a single file.
- Has native support for OS/2 EA's (as MFT attribute)

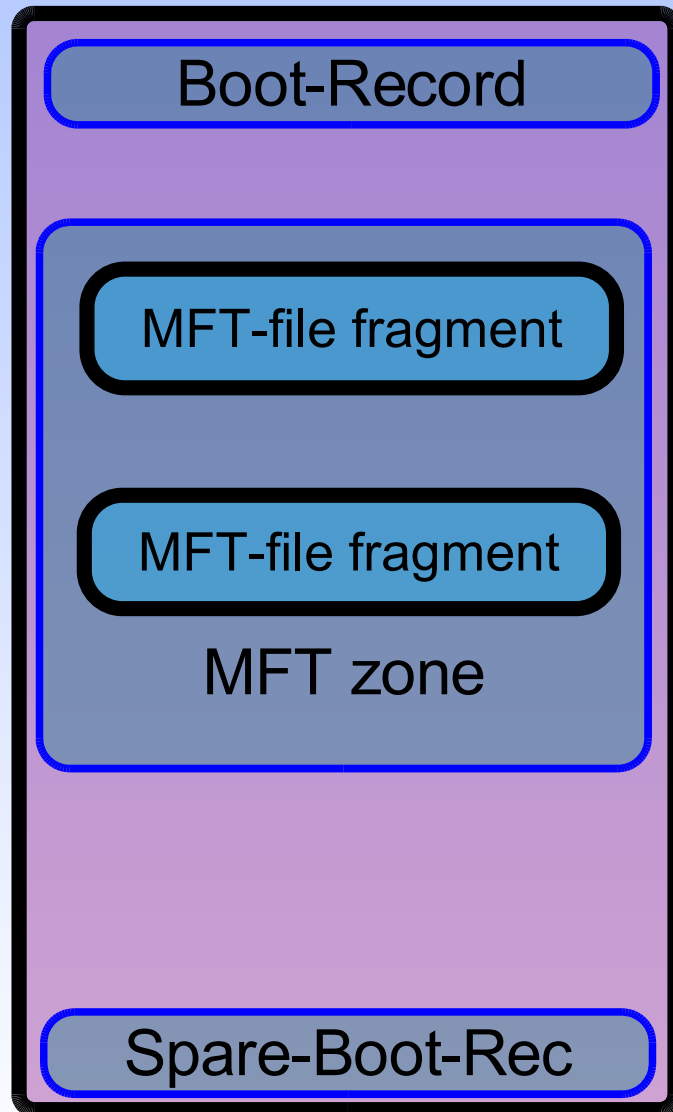
NTFS limits

- 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 2 KiB
 - May hold all data for small files. Larger attributes are stored externally, using runlists for the allocated space
- Filename of unlimited length, 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)
- 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 **journalling** filesystem
- File data itself is NOT part of the journal, so may get lost/damaged after a crash!

NTFS Volume layout



- Bootsector with NTFS bootcode
- 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, 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 logfile
- 3 = \$Volume Global volume information
- 4 = \$AttrDef Definitions for attribute values
- 5 = \ Root directory
- 6 = \$Bitmap Allocation bitmap
- 7 = \$Boot Bootrecord (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 upto 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

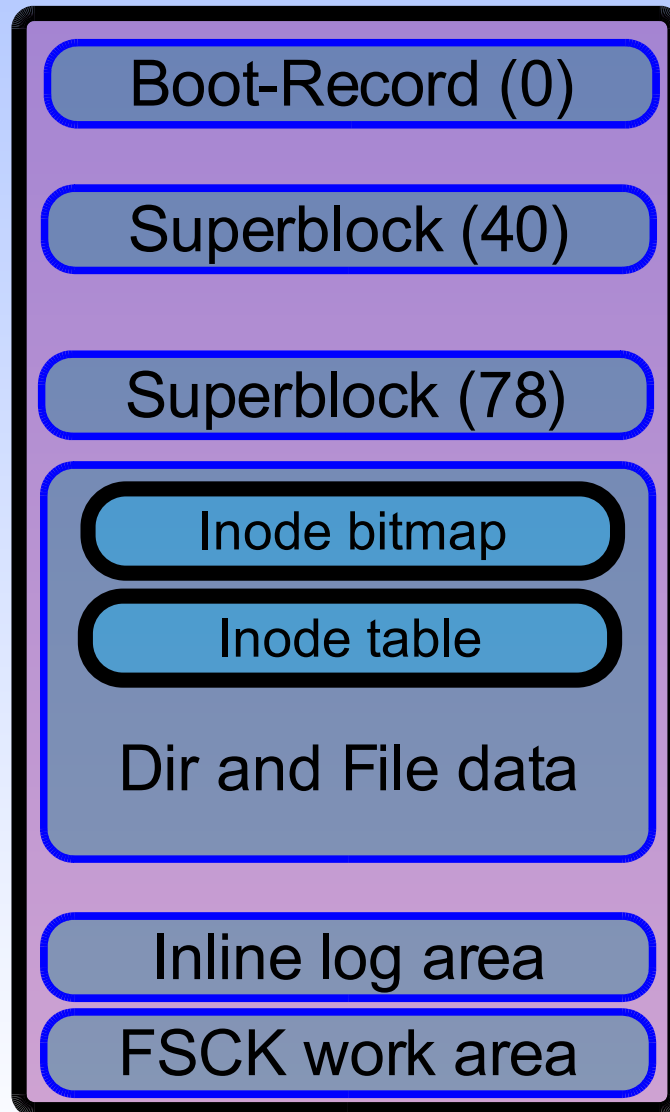
MFT attributes (from \$AttrDef)

- 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

Journalled File System

- Designed by IBM for its AIX operating system
- Based on UNIX-like structure with journalling 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
- Relies on LVM for some of its functionality

JFS Volume layout



- Bootsector, standard (label etc)
- JFS specific volume data with pointers to lots of info :-)
- Duplicate of main superblock
- Actual contents is grouped in 'aggregates' of fixed size
Layout of that to be refined
- The 'journal' file area
- Temporary space for CHKDSK

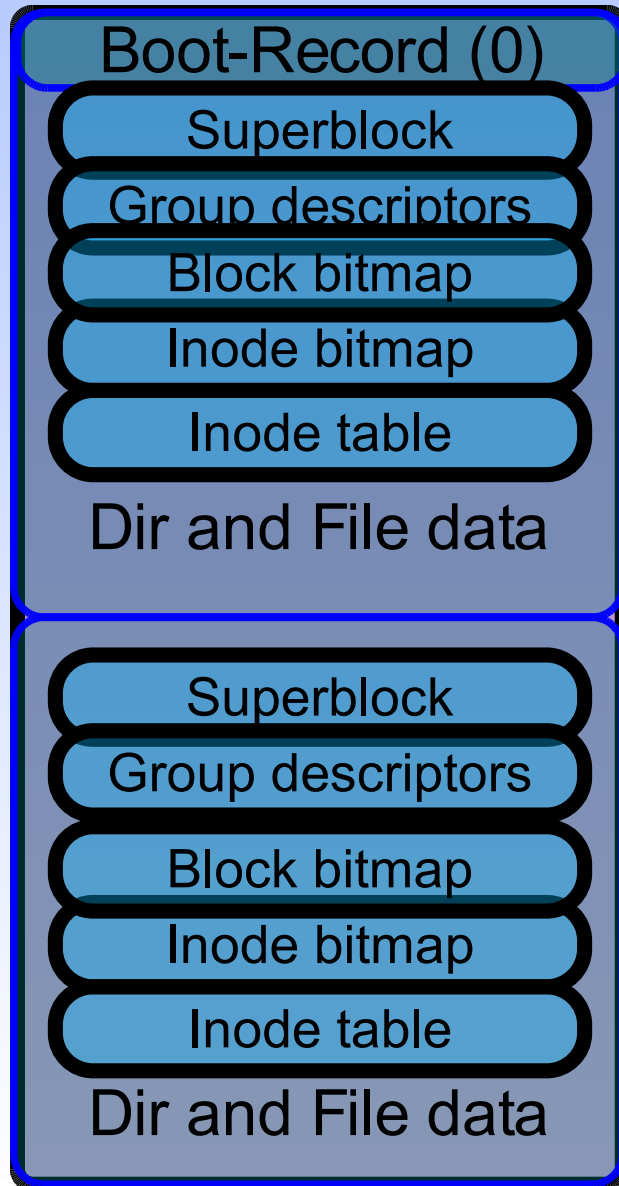
Extended 2nd FS, EXT2, EXT3

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

EXT2/3, 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 upto 12 blocks
 - Indirect, double indirect and triple-indirected blocks

EXT2/3 Volume layout

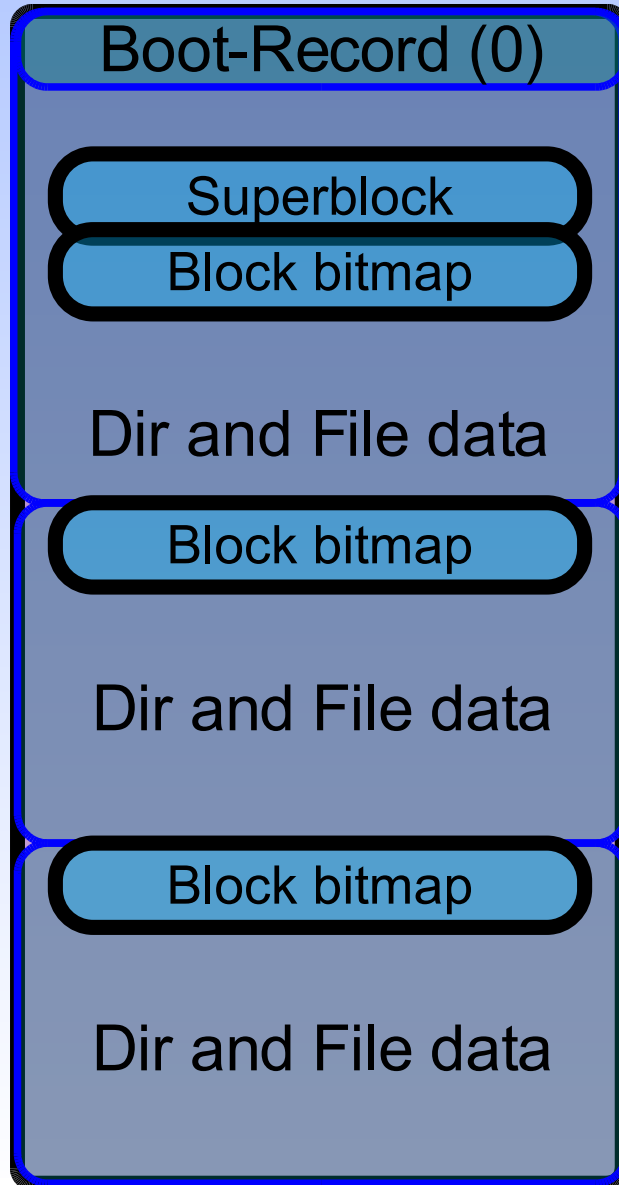


- Bootsector, normally empty may contain GRUB or LILO (is at start of the 1st block)
- Volume divided up in block-groups with identical layout, each having:
 - A superblock copy, can be sparse meaning that not every group has a copy of the superblock
 - Group description data
 - Allocation bitmap for this group
 - Usage bitmap for the inodes
 - Fixed size Inode table for the group
 - Rest of group are data blocks

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



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

On-disk filesystem structures

Questions ?

FSYS - *software*

DFSee