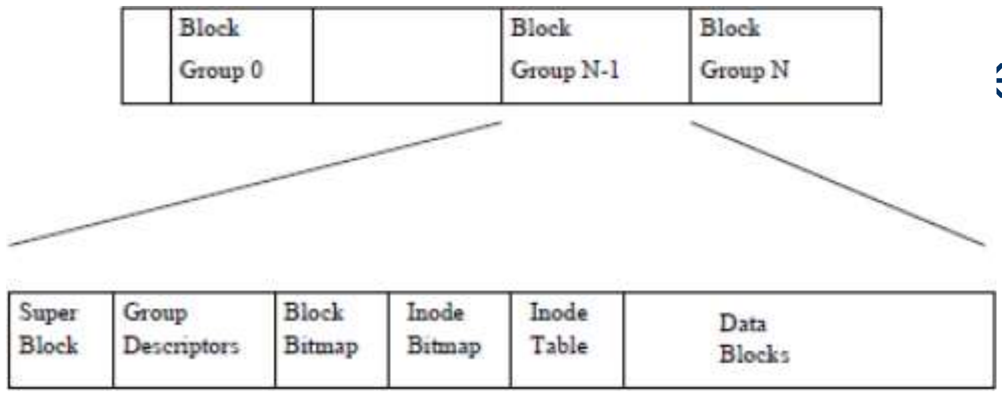
Implementations of File Systems

Last modified: 27.05.2021

Contents

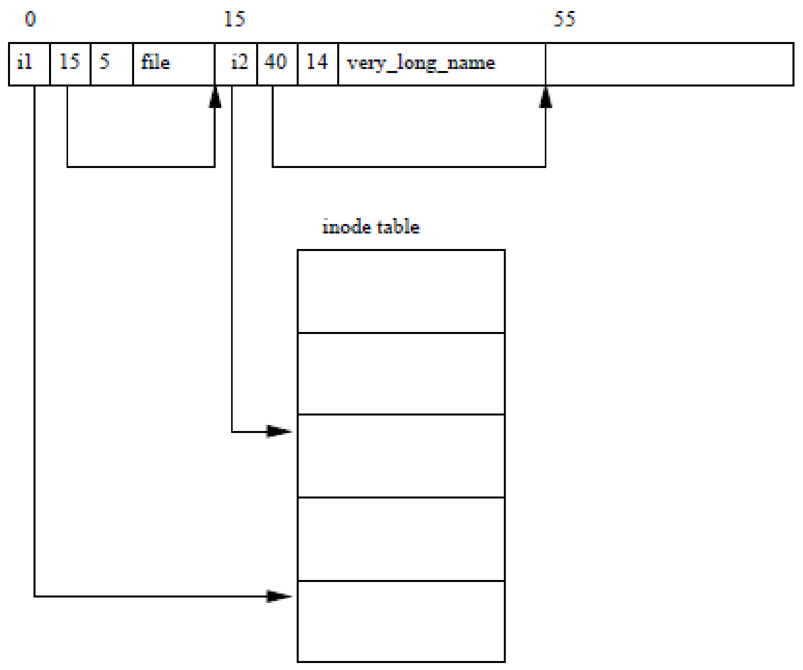
- Linux file systems
    - ext2, ext3, ext4, proc, swap
- MS Windows – FATxx, exFAT, NTFS
- ZFS/OpenZFS
- Disk management

Linux ext2

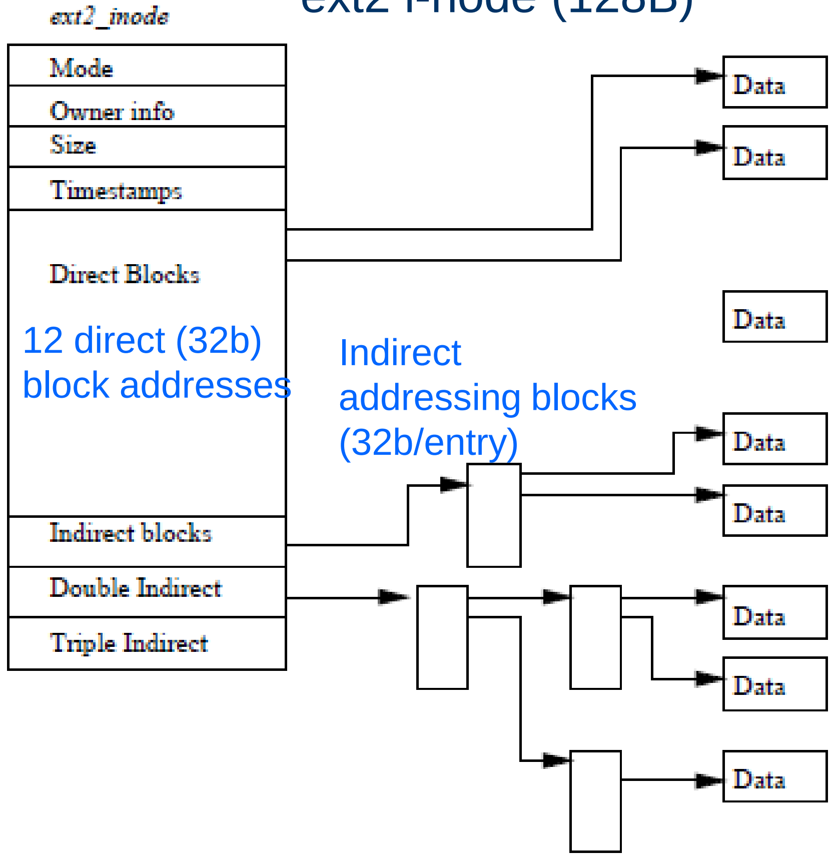


⇐ Physical layout of ext2 file system

ext2 directory



ext2 i-node (128B)



12 direct (32b) block addresses

Indirect addressing blocks (32b/entry)

textbook „Applied Operating Systems Concepts"

The Linux ext3 and ext4 File Systems

- **ext3** is standard on disk file system for Linux. Supersedes older **ext2** It keeps max. file size:16GiB-2TiB and of file system: 2-32TiB
  - Uses a mechanism similar to that of BSD Fast File System (FFS) for locating data blocks belonging to a specific file. The main differences concern disk allocation policies
    - In BSD UNIX FFS, the disk is allocated to files in blocks of 8Kb, with blocks being subdivided into fragments of 1Kb to store small files or partially filled blocks at the EOF
    - ext3 does not use fragments; it performs its allocations in smaller units (1, 2, 4 and 8 KB), the size depending on the total size of file system
    - ext3 uses cluster allocation policies designed to place logically adjacent blocks of a file into physically adjacent blocks on disk, so that it can submit an I/O request for several disk blocks as a single operation on a **block group**
    - Maintains bit map of free blocks in a **block group** (not a **cylinder group**), searches for free byte to allocate at least 8 blocks at a time (**pre-allocation**).
  - Directories can use hashed B-tree indexing.
- **ext4** - an ext3 extension, i.e. it is possible to mount ex2 and ext3 as ext4. Features:
  - Volumes up to 1 exbibyte (Eib), i.e. 2^{60} B, approx. 1.15 10^{18}B,
  - Files up to 16 tebibytes (TiB) 2^{40} B, approx. 17.6 TB (for 4 kB blocks)
  - Unlimited number of subdirs (ext3: 32000)
  - Transparent encryption
  - Extents are supported

Journaling

- ext3, ext4 implement **journaling**, with file system updates first written to a log file in the form of **transactions**
  - Once in log file, considered committed (file system operation successful)
  - Over time, log file transactions replayed over file system to put changes in place
- On system crash, some transactions might be in journal but not yet placed into file system
  - Must be completed once system recovers
  - No other consistency checking is needed after a crash (much faster than older methods)
- Improves write performance on hard disks by turning random I/O into sequential
- Levels of transactional journaling (see man ext4(5))
  - **Journal** (lowest risk). Both metadata and file contents are written to the journal before being committed to the main file system.
  - **Ordered** (medium risk, default). Only metadata is journaled; file contents are not, but it's guaranteed that file contents are written to disk before associated metadata is marked as committed in the journal.
  - **Writeback** (highest risk).. Only metadata is journaled; file contents are not.
- Journal can be stored on a separate device

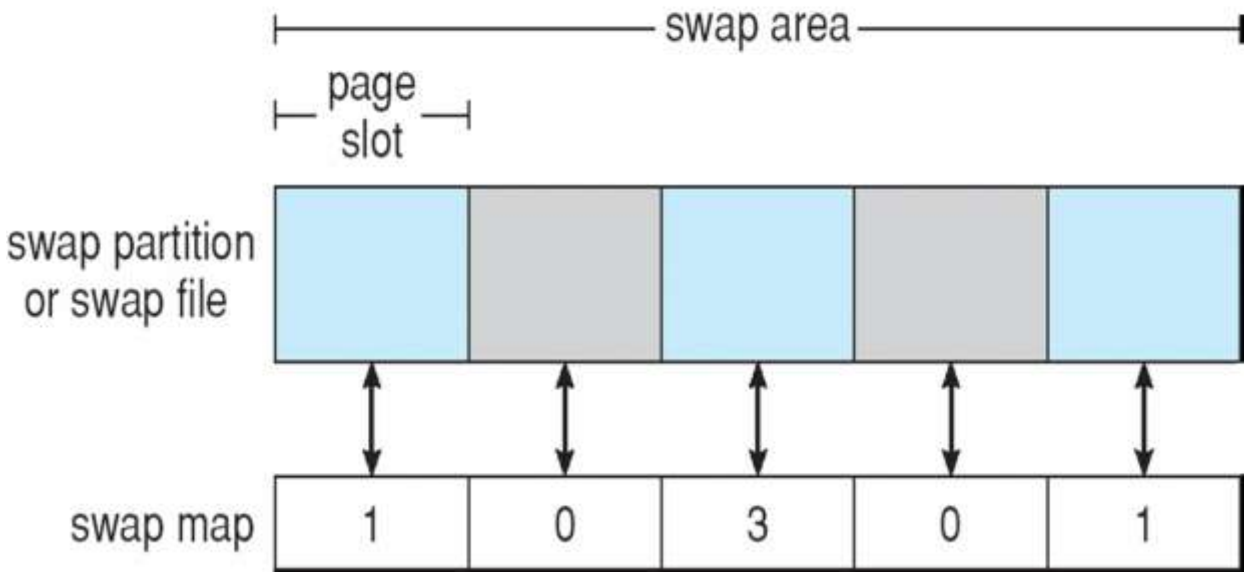
The Linux Proc File System

- The **proc file system** does not store data, rather, its contents are computed on demand according to user file I/O requests
- **proc** must implement a directory structure, and the file contents within; it must then define a unique and persistent **inode** number for each directory and files it contains
  - It uses this **inode** number to identify just what operation is required when a user tries to read from a particular file inode or perform a lookup in a particular directory **inode**. Information on process with given PID is available via i-nodes numbered e.g. as PID*2^16+id, where id determines type of information requested. Global information entries have i-node names corresponding to PID=0.
  - Kernel variables are available in /proc/sys directory
  - When data is read from one of these files, **proc** collects the appropriate information, formats it into text form and places it into the requesting process's read buffer

Swap-Space Management

- **Swapping** – strictly speaking moves entire process images between the main memory and swap space. In some operating systems swapping and paging are used interchangeable, reflecting the merging of these two concepts. Swap-space - disk space, which is used by virtual memory system, as an extension of main memory
- **Swap-space** can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition
- Swap-space management
  - 4.3BSD allocates swap space when process starts; holds text segment (the program) and data segment
  - Kernel uses swap maps to track swap-space use
  - Solaris 2 allocates swap space only when a page is forced out of physical memory, not when the virtual memory page is first created
  - Linux: swap space is used for swapping and paging (man mkswap(8), swapon(8))

Data Structures for Swapping on Linux Systems



- Swap space is only used for anonymous memory and for regions of memory shared by several processes
- One or more swap areas: files in other file systems or raw-swap-space partitions (man mkswap(8), swapon(8))
- Each swap area consist of 4kB page slots which hold swapped pages
- **Swap map**: an array of integer counters; value indicates the number of mappings to the swapped page

FAT

	FAT12	FAT16	FAT16B	FAT32
Cluster addr/ FAT entry sz	12b/ 12b	16b/ 16b	16b / 32b	28b / 32b
Max.vol./ cluster sz	32MB/ 8KB	32MB	2-16GB/ 32-256KB	2TB / 512B 16TB/64KB (4KB sectors)
Max. file	Vol.sz	Vol.sz	2/4GB-1	2/4GB-1
Max.files/cluster sz	4068/ 8KB	65536 / 32KB	65540/ 32KB	268173300/32KB

FAT partition logical structure

Reserved sectors
FAT #1
FAT #2
Root directory
Data area

- Directory entry (32B long):
  - File pointer: 8.3 format name, 1st data cluster nr, file length, attributes (RHSA), timestamp
  - Subdirectory pointer (atrib.:dir)
  - Volume label (attrib: label)
  - VFAT Long File Name (1 of max. 5 parts of a long file name: up to 13*2B chars)

MS exFAT

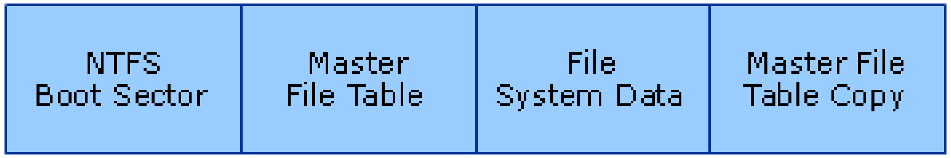
- Filesystem optimized for flash memory (def. FS for SDXC cards >=32GB); since 2019 internals disclosed (MS), 2013 – Samsung published GPL Linux driver.
- Features:
  - Max. file size: 6EiB (2^64-1);
  - Max. Vol. Size: 128PiB (2^57-1), recommended 512TiB (2^49-1)
  - 2796200 files/directory
  - Max. nr of files on volume: 2^32-1
  - Cluster size <= 32MiB
  - Free space bitmap
  - File and cluster pre-allocation
    - Single bit shows reservation state of each cluster
    - Single bit in the directory record indicates that the file is not fragmented (like a single extent); no need to use FAT
    - Files: valid space and allocated space
  - LFN (no 8.3); hash-based lookup (patented)
  - Single FAT, but metadata integrity through checksums

MS Windows NTFS

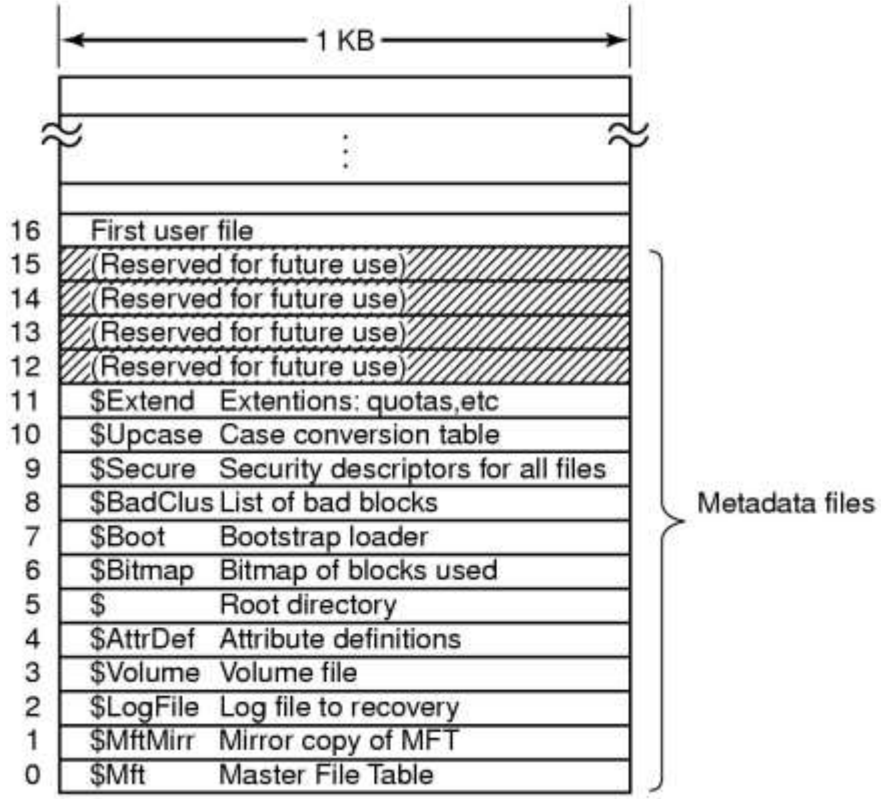
- The fundamental structure of the Windows file system (NTFS) is a ***volume***
  - Created by the Windows 7 disk administrator utility
  - Based on a logical disk partition
  - May occupy a portions of a disk, an entire disk, or span across several disks
- All ***metadata***, such as information about the volume, is stored in a regular file
- NTFS uses ***clusters*** as the underlying unit of disk allocation
  - A cluster is a number of disk sectors that is a power of two
  - Because the cluster size is smaller than for the 16-bit FAT file system, the amount of internal fragmentation is reduced
- Max. Volume size: theoret. 2^{64}-1 clusters, 1TiB=1024^3; Win10./1709: 8PB-2MB: <=Win10/1703: 256TiB-64KiB,max. file size: Win10./1709: 8PB-2MB' Win7: 16TiB-64KiB. Max. nr of files: 2^{32}-1.

NTFS internals

- Organization of an NTFS Volume



- NTFS uses logical cluster numbers (LCNs) as disk addresses
- The NTFS volume's metadata are all stored in files at the beginning of the volume.
  - The first file is MFT.
  - The second file, which is used during recovery if the MFT is damaged, contains a copy of the first 16 entries of the MFT.
  - The log is stored in the third metadata file ($LogFile).
  - Next files contain the volume file, attribute-definition table, root directory, bitmap of allocated and free clusters, boot file, bad-cluster file, change journal.



16	First user file	
15	(Reserved for future use)	
14	(Reserved for future use)	
13	(Reserved for future use)	
12	(Reserved for future use)	
11	$Extend	Extentions: quotas,etc
10	$Upcase	Case conversion table
9	$Secure	Security descriptors for all files
8	$BadClus	List of bad blocks
7	$Boot	Bootstrap loader
6	$Bitmap	Bitmap of blocks used
5	$	Root directory
4	$AttrDef	Attribute definitions
3	$Volume	Volume file
2	$LogFile	Log file to recovery
1	$MftMirr	Mirror copy of MFT
0	$Mft	Master File Table

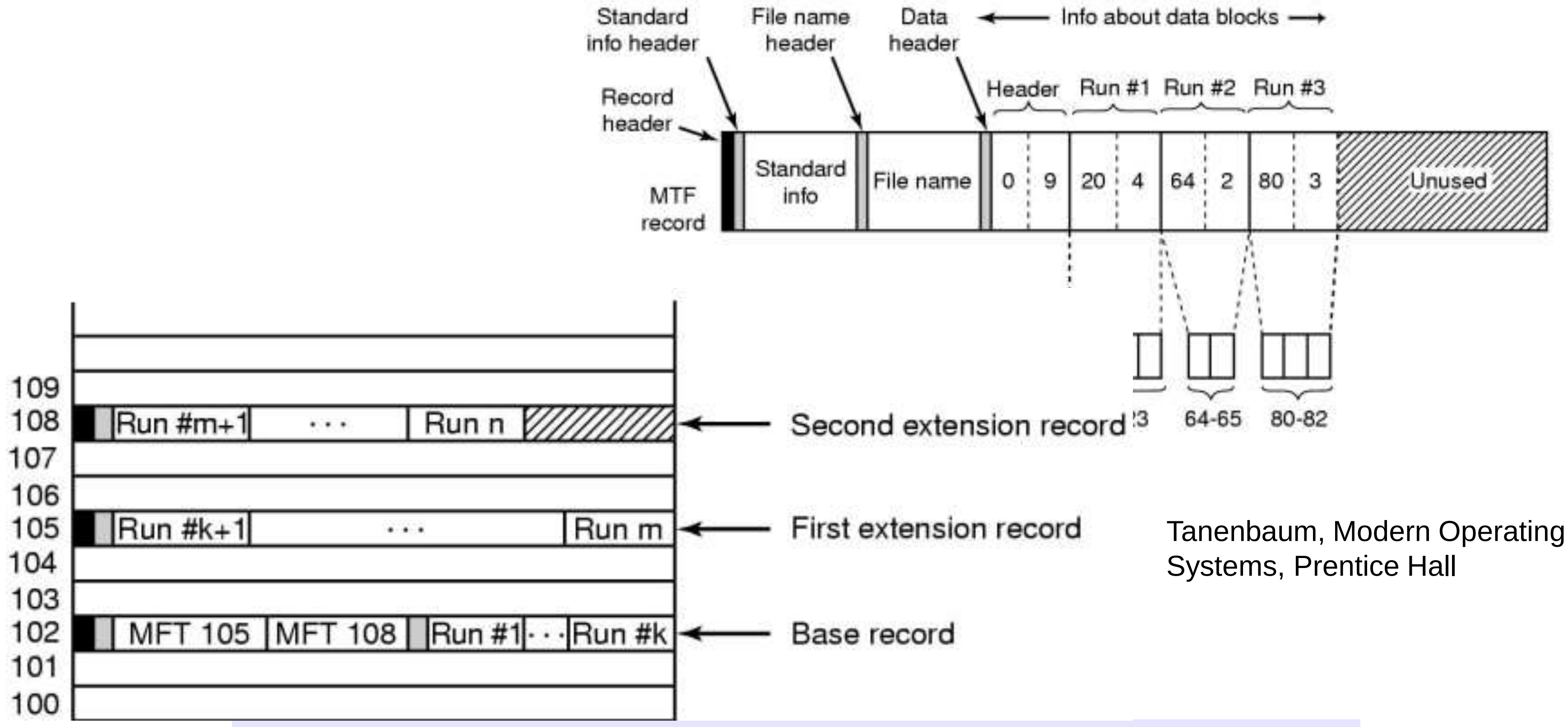
Tanenbaum, Modern Operating Systems, Prentice Hall

NTFS internals

- A file in NTFS is not a simple byte stream, as in MS-DOS or UNIX, rather, it is a **structured object consisting of attributes.** Attribute **can be added** as necessary using file-name-attribute syntax. **Unnamed attribute** refers to the bulk data of a file.
- Each file on an NTFS volume has a unique ID called a **file reference**.
  - 64-bit quantity that consists of a 48-bit file number and a 16-bit sequence number
  - Can be used to perform internal consistency checks

NTFS internals

- Every file in NTFS is described by one or more records (**base** and **extension records**) in an array stored in a special file called the **Master File Table** (MFT).
- Small attributes (**resident attributes**) are stored in the MFT record itself. For large attributes (such as large size unnamed bulk data) clusters are allocated for the data, and the cluster location is stored as **data runs** in the attribute.



Tanenbaum, Modern Operating Systems, Prentice Hall

NTFS internals - folders

- The NTFS name space is organized by a hierarchy of directories; the index root contains the top level of the **B+ tree**.
- Directory entry contains:
  - name
  - file reference
  - copy of the update timestamp and file size taken from the file’s resident attributes in the MFT (to speed up directory listing).

NTFS internals – links and such

- **Reparse point** – special file (since NTFS 3.0): contains reparse tag and data, that are interpreted by a filesystem filter identified by the tag (upon reparse point reference). Default tags:
  - **NTFS symbolic links** – files, containing a pathname of another file, perhaps on another file system
  - **Directory junction points** - links to directories on local drive
  - **NTFS volume mount points** - entry points to root directory of the mounted volume
- **Hard links** – directory entries, which share MFT record (and so file reference)

NTFS internals – advanced features

- To compress a file, NTFS divides the file's data into ***compression units***, which are blocks of 16 contiguous clusters.
- For sparse files, NTFS uses another technique to save space.
  - Clusters that contain all zeros are not actually allocated or stored on disk.
  - Instead, gaps are left in the sequence of virtual cluster numbers stored in the MFT entry for the file.
  - When reading a file, if a gap in the virtual cluster numbers is found, NTFS just zero-fills that portion of the caller's buffer.
- Windows implements the capability of bringing a NTFS volume to a known state and then creating a ***shadow copy*** that can be used to backup up a consistent view of the volume. Making a shadow copy of a volume is a form of ***copy-on-write*,** where blocks modified after the shadow copy is created are stored in their original form in the copy.
- Windows maintains **Update Sequence Number Journal** (USNJ) that records changes to files, streams and directories on the volume, as well as their various attributes and security settings. Applications can track changes to the volume.
- **Alternate data stream (ADS)** allow more than one data stream to be associated with a filename (a fork), using the format "filename:streamname"

File System — Recovery

- All file system data structure updates are performed inside transactions that are logged.
  - Before a data structure is altered, the transaction writes a log record that contains redo and undo information.
  - After the data structure has been changed, a commit record is written to the log to signify that the transaction succeeded.
  - After a crash, the file system data structures can be restored to a consistent state by processing the log records.
- This scheme does not guarantee that all the user file data can be recovered after a crash, just that the file system data structures (the metadata files) are undamaged and reflect some consistent state prior to the crash.
- Log handling is performed by the system ***log file service***.

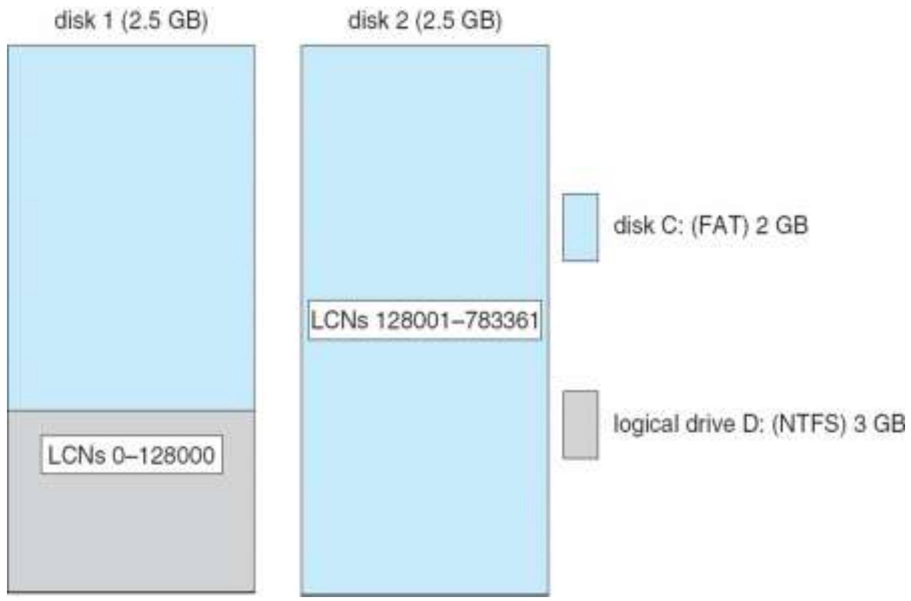
File System — Security

- Security of an NTFS volume is derived from the MS Windows (NT) object model.
- When a user logs on successfully, the system creates an **access token,**. with **security identifiers** (SID) that identify the user's account and any group accounts to which the user belongs; also a list of the privileges held by the user or the user's groups. Every process executed on behalf of this user will have a copy of this access token. The system uses this token to control access to securable objects.
- Each file object has **a security descriptor attribute** stored in an MFT record.
- The **security descriptor attribute** contains SIDs of the owner of the file and its main group; also two access control lists
  - **discretionary access control list** (DACL), defines exactly what type of interaction (e.g. reading, writing, executing or deleting) is allowed or forbidden by which user or groups of users.
  - **system access control list** (SACL), defines which interactions with the file or folder are to be audited and whether they should be logged when the activity is successful, failed or both.
- Access to a file is granted if there is no "**deny**" entry in its ACL, and there exists at least one "**allow**" entry.
- Access rights can be inherited from paremt directories.

Volume Management and Fault Tolerance

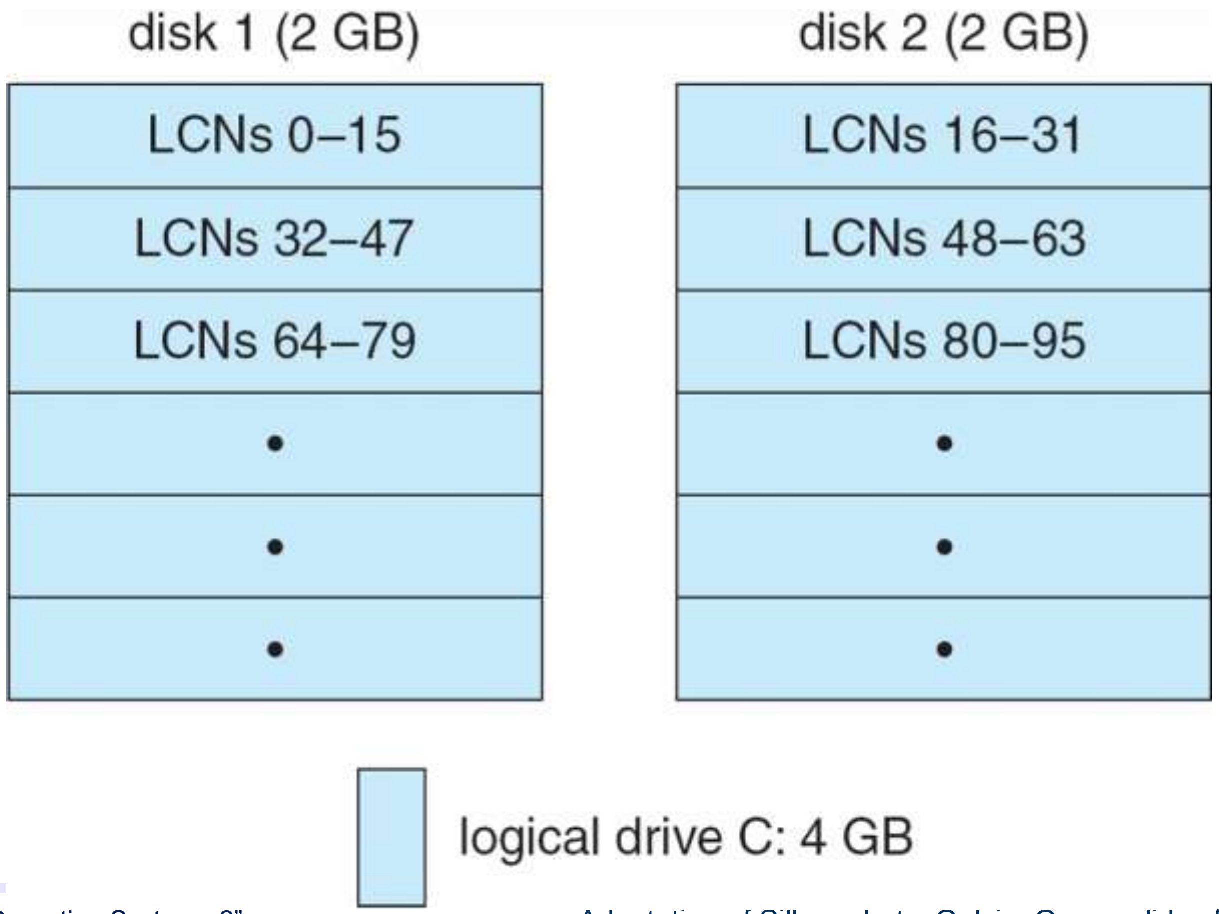
- **`FtDisk`**, the fault tolerant disk driver for Windows 7, provides several ways to combine multiple disk drives into one logical volume.
- To deal with disk sectors that go bad, FtDisk, uses a hardware technique called *sector sparing* and NTFS uses a software technique called *cluster remapping*.
- Logically concatenate multiple disks to form a large logical volume, a *volume set*

Example: volume set on 2 drives



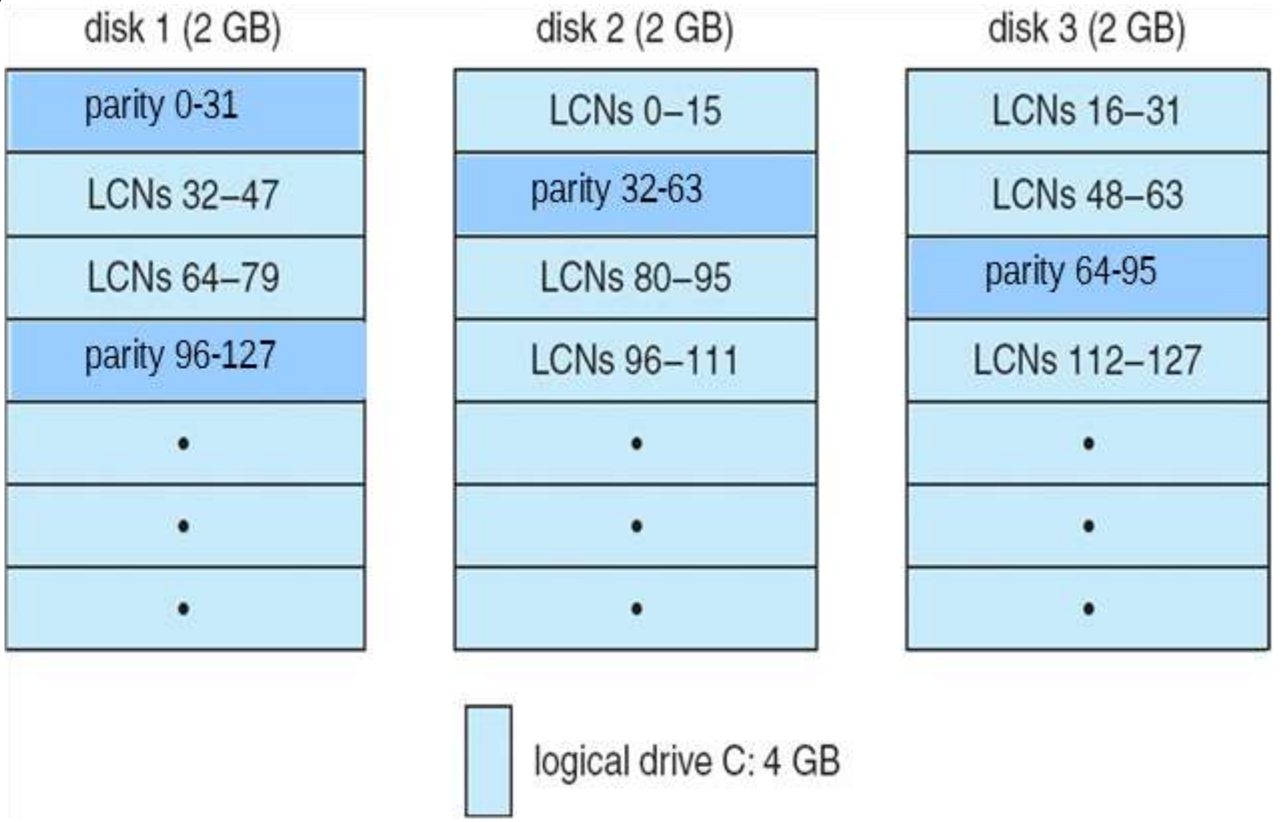
Stripe Set on Two Drives (RAID 0)

Interleaves multiple physical partitions in round-robin fashion to form a *stripe set* (also called RAID level 0, or "disk striping")
Advantage: possible improvement of access time



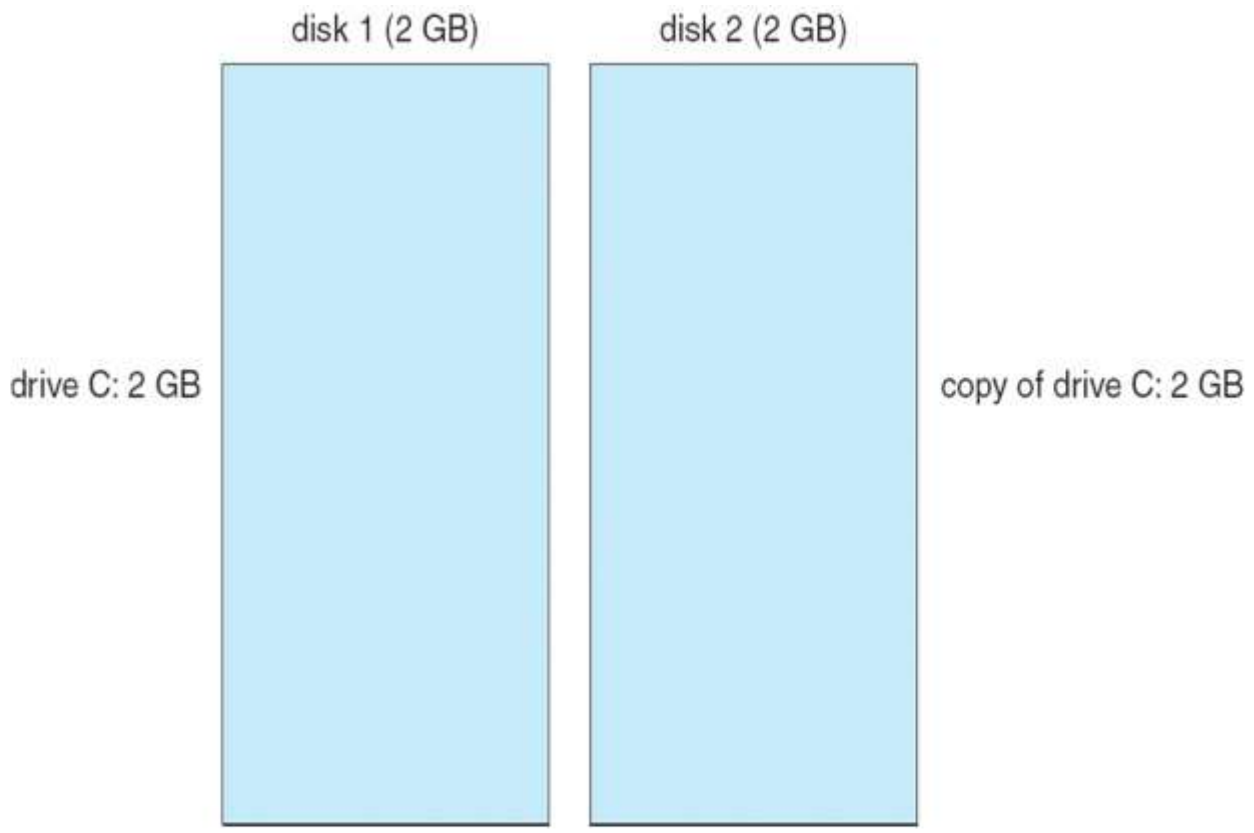
Stripe Set With Parity on Three Drives (RAID 5)

Variation of striping: ***stripe set with parity***, or RAID level 5. A parity stripe contains the byte-wise exclusive OR of the data stripes
Advantage: resistant to a single disk failure
Disadvantage: poor throughput for small file writes (read-modify-write cycles)



Mirror Set on Two Drives (RAID 1)

Disk mirroring, or RAID level 1, is a robust scheme that uses a ***mirror set*** — two equally sized partitions on two disks with identical data contents. Improvement of reliability, possibly in read througput



ZFS

ZFS - a combined file system and logical volume manager designed by Sun Microsystems. Currently developed as OpenZFS. ZFS is available on Linux via FUSE (see man: zfs-fuse(8)).

Some of the prominent features:

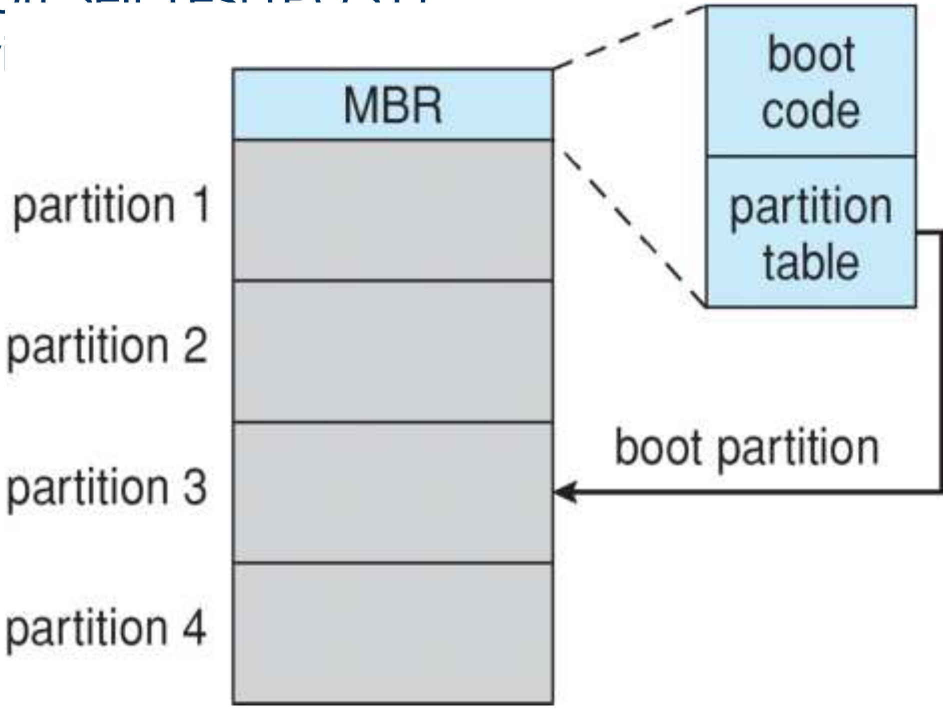
- support for high storage capacities; max. volume 256 zebibytes (2^{78}B), approx. 1,50 * 10^{23}B
- protection against data corruption,
- during writes, a block may be compressed, encrypted, checksummed and then **deduplicated**
- integration of filesystem and volume management,
- snapshots and copy-on-write clones,
- continuous integrity checking (checksums) and automatic repair; can handle whole disk failures, but also silent data corruption, disk firmware and driver errors
- RAID-Z1, RAID-Z2, RAID-Z3 (1,2 or 3 disks are allowed to fail – similar to RAID-5, RAID-6, RAID-7 but faster due to dynamic stripe width)

Disk Management

- **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information, plus data (usually 512B), plus error correction code (**ECC**)
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk
  - **Partition** the disk into one or more groups of cylinders, each treated as a logical disk
  - **Logical formatting** or "making a file system"
  - Methods such as **sector sparing** used to handle bad blocks
- When computer is turned on
  - A small **bootstrap** program initializes hardware
  - Then operating system is started, using boot blocks of boot partition

PC booting – MBR/BIOS

- When the PC is switched on and the power is steady:
- ROM BIOS (BIOS) initiates Power On Self Test (POST)
- The BIOS determines the "boot dev
- The BIOS loads the contents of the first physical sector of the hard disk into memory (the MBR) to location 7C00 through to 7DFF.
- The BIOS instructs the CPU to execute the MBR code by issuing a jump to location 7C00.
- MBR code which is Operating System *independent* finds in the MBR table an active (boot) partition and activates Operating System loader (can be OS specific) from boot blocks of that partition..



Note:

- The BIOS must run in 16-bit processor mode, and only has 1 MB of space to execute in.

GPT (GUID Partition Table)

MBR limitations:

- works with disks up to 2 TB in size.
- only supports up to four primary partitions — one of primary partitions can become “extended partition” so that logical partitions can be created inside it.

GPT features

- Every partition has a “globally unique identifier” GUID
- Multiple copies of table → recovery possible
- BIOS is replaced with **UEFI** (Unified Extensible Firmware Interface). UEFI can run in 32-bit or 64-bit mode. UEFI provides boot services (text and graphical consoles, bus, block and file services.) and runtime services include: date, time and NVRAM access).
- UEFI works with MBR (legacy BIOS mode) and GPT disks
- **ESP** (EFI system partition) is a partition on a data storage device that is used by computers adhering to the UEFI. When a computer is booted, UEFI firmware loads files stored on the ESP to start installed operating systems and utilities.
- An ESP needs to be formatted with a file system whose specification is based on the FAT file system and maintained as part of the UEFI specification.
- UEFI can support **Secure Boot**, which prevents loading of drivers or OS loaders that are not signed with an acceptable digital signature.