FAT v NTFS

- However, **while FAT is acceptable for most uses**, but:
  - very old
  - limited
  - relatively simplistic file system.
- FAT has
  - **few of the security**,
  - **capacity and reliability features**
- that are needed by high-end users, and especially, servers and workstations in a **corporate networking environment**.
- Recognizing that FAT was not a worthy basis upon which to build its new Windows NT operating system, Microsoft created the **New Technology File System, or NTFS.**
- The goals behind NTFS were to provide a flexible, adaptable, high-security and high-reliability file system, to help position Windows NT as a "serious" operating system for business and corporate users.

Overview and History of NTFS

- **NTFS is definitely "new" from** the standpoint that it is not based on the **old FAT** file system.
- However, **NTFS is not *entirely* new**, because some of its concepts were based on another file system that Microsoft was involved with creating: **HPFS**.
- **Before there was Windows NT, there was** OS/2.
- OS/2 was a joint project of Microsoft and IBM in the early 1990s; the two companies were trying to create the **next big success in the world of graphical operating systems**. They succeeded, to some degree, depending on how you are measuring success. :^) OS/2 had some significant technical accomplishments, **but suffered from marketing and support issues.**
- When they did this, they **borrowed many key concepts** from OS/2's native file system, **HPFS**, in creating NTFS.

NTFS goals

- **Reliability:** One important characteristic of a "**serious**" file system is that it **must be able to recover from problems without data loss resulting**. **NTFS implements specific features** to allow important **transactions to be completed** as an **integral whole**, to **avoid data loss**, and to **improve fault tolerance**.

- **Security and Access Control:** A major weakness of the FAT file system is that it includes **no built-in facilities** for **controlling access to folders or files on a hard disk**. **Without this control**, it is nearly impossible to implement applications and **networks that require security** and the ability to manage who can **read or write various data**.

- **Breaking Size Barriers:** In the early 1990s, FAT was limited to the **FAT16 version** of the file system, which only allowed partitions up to 4 GiB in size. **NTFS was designed** to **allow very large partition sizes**, in anticipation of **growing hard disk capacities**, as well as the use of **RAID arrays.**

NTFS goals

- **Storage Efficiency:** Again, at the time that NTFS was developed, most PCs used FAT16, which results in significant **disk space due to slack**. **NTFS avoids this problem by using a very different method of allocating space to files than FAT does**.

- **Long File Names: NTFS allows file names to be up to 255 characters**, instead of the 8+3 character limitation of conventional FAT.

- **Networking:** While networking is commonplace today, it was still in its relatively early stages in the PC world when Windows NT was developed. At around that time, businesses were just beginning to understand the importance of networking, and Windows **NT was given some facilities** to **enable networking on a larger scale**. (Some of the NT features that allow networking are not strictly related to the file system, though some are.)

NTFS Versions

- NTFS 1.0 or NTFS 3.1
- NTFS 1.1 / 4.0
- **NTFS 5.0: news**
- **1. Reparse Points: Files and directories** within the file system **can have actions associated with them**, so that when the file system object is **accessed in a particular way**, the action is carried out.
- **2. Improved Security and Permissions:** The **mechanisms for managing file system security** and **assigning permissions were improved.**
- **3. Change Journals: Disk volumes can be set to keep track** of **all operations performed** on the **files and directories they contain**.
- **4. Encryption:** NTFS 5.0 **allows you to encrypt files and automatically decrypt them** as they **are read.**

NTFS Versions

- **5. Disk Quotas: Administrators can track** how much disk space is being used by users or groups of users, and **even limit disk space** use if necessary.

- **6. Sparse File Support:** To save space on the disk, support was added for the **more efficient storage of *sparse files***, which are **large files that are mostly empty.**

- **7. Disk Defragmenter: Windows 2000 includes a disk defragmentation** program, **where Windows NT did not.** (Arguably, **this is an operating system feature of Windows 2000**, and not a file **system enhancement**, but I thought I'd mention it anyway, since it is obviously file system related.)

NTFS Architecture Overview

- **Virtually, every structure in NTFS is a file,**
- **including the structures used:**
  - **to manage the partition**
  - **maintain statistics**
  - **control information about the partition itself.**

- The **control information is stored in a set of special files** that are **created** when an **NTFS partition is first created**;
- these are **called metadata files** and **include such items** as:
  - **lists of files on the partition**
  - **volume information**
  - **cluster allocations, and so forth.**

- **One exception** to the "**everything is a file**" **rule** is the **partition boot sector**,
  - which **precedes the metadata files** on an **NTFS partition and**
  - **controls the most basic of NTFS operations**,
  - such as **loading the operating system**.

NTFS Architecture Overview-cont.

- **Every file in an NTFS partition is a collection** of **attributes**;
  - this even includes the **data** that the **file contains**, which is just **considered one** of **many attributes**.
  - **Other attributes include items such as** the **file's name and size**.
  - **This arrangement really** a **database-like setup—**
  - the operating system view **files as being objects** with **various characteristics**, and **manages them accordingly**.
  - This makes it **easy to manage files** and **add attributes if needed** in the **future.**
- Internally, **NTFS stores all files** (including metadata files) using a **cluster system--**each file is **broken into clusters**, each of which contain a binary number of 512-byte sectors.
- On the surface, this is **somewhat similar to how FAT stores data**,
- **but** the **implementation of clusters in NTFS is somewhat different**.

NTFS Volume Boot Sector

- The **NTFS volume boot sector begins** in the **first sector** of the **partition**, and consists of **2 different primary structures**. Again, these are similar to the structures in a FAT volume boot sector:

- **1. BIOS Parameter Block:** This is a block of data that contains **fundamental information about the volume itself**. This block identifies the volume as an NTFS partition, and includes such information as the **volume label and its size**. In addition, NTFS provides for an **extended BIOS parameter block**, which **contains additional information** about the **volume such** as the **location** of the **key metadata files**.

- **2. Volume Boot Code:** This is a **small block of program code** that **instructs the system on how to load the operating system**. With an NTFS volume, this code will be specific to Windows NT or 2000, whichever is installed on the system. **It will generally load NTLDR**, the NT loader program, and then transfer control to it to load the rest of the operating system. Note that **this code is also present** in the partition as a **system (metadata) file.**

NTFS System (Metadata) Files

Metadata File Name	File Name	MFT Record #	Description
Master File Table (MFT)	$MFT	0	This is the MFT itself. This seems to be a bit of a chicken-and-egg problem--how can a record in the MFT contain the MFT? :^) It doesn't. This first MFT record contains descriptive information *about* the MFT. This is consistent with how NTFS works--since the MFT itself is just "a file", so it also needs a record in the MFT!
Master File Table 2 (MFT2) or Master File Table Mirror	$MFTMirr	1	This is a *mirror* of the first 16 records of the real Master File Table. It is stored either in the middle of the partition (for Windows NT 3.5 and earlier) or the end of the partition (for Windows NT 4.0 and later). The mirror is a "backup" that is used to ensure that the first few records of the MFT, which of course describe the metadata files themselves, can be accessed in case of a disk error on the original MFT.
Log File	$LogFile	2	The transaction logging file for the volume. This is part of NTFS's file system recoverability feature.

NTFS System (Metadata) Files

Volume Descriptor	$Volume	3	Contains key information about the volume (partition) itself, such as its name, NTFS version, creation time, and so on. See the complete list of NTFS file attributes for more information.
Attribute Definition Table	$AttrDef	4	This table contains the names and descriptions of the various types of NTFS file attributes used on the volume. (It doesn't contain the attributes themselves, but rather descriptions of what the attributes mean. Remember--*meta*data.)
Root Directory / Folder	"." (single period)	5	This is a pointer to the root directory or folder of the volume.
Cluster Allocation Bitmap	$Bitmap	6	Contains a "map" showing which clusters on the volume are used and which are available for use.
Volume Boot Code	$Boot	7	This record contains a copy of the volume boot code (or a pointer to it). The volume boot code is also found in the volume boot sector.
Bad Cluster File	$BadClus	8	A list of all clusters on the volume that have been marked as "bad" (meaning, an error was detected on the volume somewhere in those clusters, so the file system wants to be sure not to use them again.)
Quota Table	$Quota	9	Table containing quota information, if disk quotas are being used on the volume. Only used for NTFS 5.0 or later.
Upper Case Table	$UpCase	10	Table containing information for converting file names to the Unicode (16-bit) file naming system for international compatibility.

Master File Table (MFT)

- **Probably** the **most important of** the **key system (metadata) files**
- that define an **NTFS volume**,
- **MFT is** the **place where information about every file** and **directory** on an **NTFS volume is stored**.
- The **MFT is in essence** a **relational database table**,
  - **containing various attributes about different files.**
- It acts as the "**starting point**" and central management feature of an NTFS volume--**sort of a "table of contents**" for the **volume**, if you will.
- It is **somewhat analog** to the **file allocation table in** a **FAT partition**, but **is much more than just a list of used and available clusters**.
- **When any file or directory is created on the NTFS volume**, a **record is created for it within** the **MFT**.
- The **size of each record in the MFT is equal** to the **cluster size** of the volume, but with a
  - **minimum** of 1,024 bytes **1K**
  - **maximum** of 4,096 **4K**

MFT cont.

- The **system uses these MFT records** to **store information** about the **file or directory**;
- **this information takes** the **form** of **attributes.**

- Since the **size of each MFT record is limited**,
- there are different ways that NTFS can store a file's attributes:
  - as either **resident attributes**
    - that **are stored within the MFT record**,
  - or **non-resident attributes**,
    - **stored either in additional MFT records**
    - or in **extents that lie outside the MFT**.

- Remember that under NTFS, there is **no special distinction between** the **data in a file and the attributes that describe the file—**
- the **data itself** is just the **contents** of the "**data attribute**".

MFT and small files

- This has an **interesting implication** for **small files**.
- **If the amount of space** required for
  - *all* of the attributes of a file,
  - including the data it contains,
  - **is smaller than the size of the MFT record**,
  - the **data attribute** will **be stored resident—**
- **within the MFT record itself.**

- Thus, **such files require no additional storage space on the volume**, and **also do not require separate accesses to the disk** to **check the MFT** and **then read the file**, **which improves performance**.

MFT and larger files

- **Larger files get more complicated**.
- As **additional attributes** **are added to** a **file**—
  - **either standard attributes defined by the system** or
  - **new ones created by the user**--and as the existing attributes are **expanded in size,**
  - **they may no longer fit into** the **MFT record** for the file.
- If this occurs, the **attributes** will be **moved out of the MFT** and be made **non-resident by** the **file system**.
- **Large files** will have **their data** stored as **external attributes**
- **Very large files** may even **get so large**
  - that the **attributes containing pointers** to the file **data**
  - **become external attributes themselves**!

MFT zone

- **As more files and directories are added to** the **file system**, it becomes necessary for NTFS to **add more records to the MFT**.
- **Since keeping** the **MFT contiguous on** the **disk improves performance**,
  - when an **NTFS volume is first set up**,
  - the **operating system reserves about 12.5% of** the **disk space immediately following the MFT**;
  - this is sometimes **called** the **"MFT Zone**".
- if there are enough entries placed in the MFT, as it expands it will use **up the "MFT Zone**".
- **When this happens**, the **operating system will automatically allocate more space elsewhere on** the **disk for the MFT.**
- **This allows the MFT to grow to a size limited only** by the **size of the volume**, but this **fragmentation** of the **MFT may reduce performance by increasing the number of reads required for some files**, and the **MFT cannot generally be defragmented**.

NTFS Partitions and Partition Sizes

- Under NTFS,
- the **maximum size of a partition (volume)** is in fact
- **2 to the 64th power.**

- This is equal to **16 binary exa-bytes**,
- or **18,446,744,073,709,551,616 bytes**.
- P Q T G M K
- **Does that seem large enough for your needs**? :^)
- Well, many problems with PC hard disks occurred when unintentional size limits were imposed by engineers who figured that, for example, "2 gigabytes ought to be enough".

- However, with **18 billion gigabytes** it would seem that you should be safe for a while in NTFS. :^)

NTFS Clusters and Cluster Sizes

- While **both FAT and NTFS use clusters,**
- **they use them in a very different way**, of course.

- This is due to the differences in the internal structures of the **two file systems.**

- Some of the **performance issues associated** with **very large FAT file system partitions** are **due to the fact**
- **that** the **file allocation tables** grow to a **very large size**,
- and **FAT was never created with extremely large volumes in mind**.

- In contrast, NTFS is **designed**
- **to be able to better handle the large internal structures**
- (such as the **MFT)** that **occur with large partitions**.

NTFS Clusters and Cluster Sizes (4K)

Partition Size Range (GiB)	Default Number of Sectors Per Cluster	Default Cluster Size (kiB)
<= 0.5	1	0.5
> 0.5 to 1.0	2	1
> 1.0 to 2.0	4	2
> 2.0 to 4.0	8	4
> 4.0 to 8.0	16	8
> 8.0 to 16.0	32	16
> 16.0 to 32.0	64	32
> 32.0	128	64

FAT 32

4 kiB for all partitions over 2.0 GiB, regardless of their size.
The reason for the difference between operating systems is perhaps a bit surprising: it has to do with NTFS's built-in file-based **compression**.

NTFS Directories and Files

- **From** an **external, structural perspective**,
- **NTFS generally employs** the **same methods for organizing files and directories as the FAT file system**
- This is usually called the **hierarchical or directory tree model.**
- The **"base" of the directory structure** is the **root directory**, which is **actually one** of the **key system metadata files** on an **NTFS volume**.

- While **NTFS is similar to FAT in its hierarchical structuring** of directories, it **is very different in how they are managed internally.**
- **One of** the **key differences** is that
- **In FAT volumes**,
  - directories are **responsible for storing most** of the key **information about files;**
  - the **files themselves contain only data**.
- **In NTFS**,
  - **files are collections of attributes**, so **they contain** their **own descriptive information, as well as their own data.**
  - An **NTFS directory pretty much stores only information about** the **directory itself**, not **about** the **files within the directory**.

NTFS directory record in MFT

- The **MFT record for** the **directory contains** the **following information** and **NTFS attributes:**
- **1. Header (H):**
  - **This is a set of low-level management data used by NTFS to manage the directory**.
  - **It includes sequence numbers used internally** by **NTFS** and **pointers to the directory's attributes and free space within the record**.
  - (Note that the **header is part of the MFT record but not an attribute**.)
- **2. Standard Information Attribute (SI):**
  - This attribute contains "standard" information stored **for all files and directories**.
  - **This includes fundamental properties** such as d**ate/time-stamps** for when the **directory was created, modified** and **accessed**.
  - **It also contains the "standard" attributes** usually associated with a file (such as whether the file is read-only, hidden, and so on.)
- **3. File Name Attribute (FN):**
  - **This attribute stores** the **name associated with** the **directory**.
  - **Note that** a **directory can have multiple file name attributes**,
  - **to allow** the **storage** of the "**regular" name of** the **file**, **along with** an **MS-DOS short filename** alias and also **POSIX-like hard links from multiple directories**. **See here for more on NTFS file naming**.

NTFS directory in MFT

- **4. Index Root Attribute:**
  - This **attribute contains** the **actual index of files contained within** the **directory**, or part of the index **if it is large**.
  - **If** the **directory is small, the entire index will fit within this attribute in** the **MFT;**
  - **if it is too large**, **some of the information is here** and the rest is stored in **external index buffer attributes**, as **described below**.
- **5. Index Allocation Attribute:**
  - **If a directory index is too large to fit** in the **index root attribute**,
  - the **MFT record** for the **directory will contain** an **index allocation attribute**,
  - **which contains pointers to index buffer entries containing** the **rest of** the **directory's index information.**
- **6. Security Descriptor (SD) Attribute:**
  - **This attribute contains security information** that **controls access** to the **directory** and **its contents**.
  - The **directory's Access Control Lists (ACLs)** and **related data are stored here.**

NTFS directories

- **small directories are stored entirely within their MFT entries**, **just like small files are.**
- **Larger ones have their information broken into multiple data records** that are referenced from the **root entry for the directory in the MFT**.
- NTFS uses a special way of **storing these index entries however**, compared to **traditional PC file systems**.
- **FAT**
- **FAT FS uses** a **simple linked-list** arrangement for **storing large directories**:
  - the **first few files are listed in** the **first cluster of** the **directory**,
  - and then **next files go into the next cluster**, **which is linked to** the **first**, and so on.
- **This is simple to implement**,
  - **but means that every time you look at the directory**
  - **you must scan it from start to end and**
  - **then sort it for presentation to the user**.
- It also makes it **time-consuming to locate individual files in the index**, especially with **very large directories**.

B trees

- **To improve performance, NTFS directories use** a **special data management structure** called a **B-tree**.
  - This is a **concept taken from relational database design**.
  - In brief terms, a **B-tree is a balanced storage structure that takes** the **form of trees**, where **data is balanced between branches of** the **tree**.
  - (Note that the **"B-tree" concept here refers** to a **tree of storage units** that **hold** the **contents of an individual directory**; it is a different concept entirely from that of the "directory tree", a logical tree of directories themselves.)
- **From** a **practical standpoint**, the **use of B-trees means that** the **directories** are **essentially "self-sorting".**
- **There is** a **bit more overhead involved when adding files to an NTFS directory**, because **they must be placed in this special structure**.
- However, the **payoff occurs when the directories are used**.
- The **time required to find a particular file under NTFS**
  - **is dramatically reduced compared to an unsorted linked-list structure—**
  - especially for **very large directories**.

NTFS Files and Data Storage

- Within NTFS,
- **all files are stored** in **pretty much** the **same way**:
- as a **collection of attributes**.

- **This includes the data in the file itself**, which is just another attribute: the "data attribute", technically.

- You may also wish to review **the discussion of NTFS attributes,** because understanding the difference between ***resident* and *non-resident* attributes** is important to making any sense at all of the rest of this page. ;^)
- The **way that data is stored in files in NTFS depends on** the **size of the file**.

File attributes

- The **core structure** of **each file is based on the following information** and **attributes that are stored for each file**
- **Header (H):**
  - The **header in the MFT** is a **set of low-level management data** used by **NTFS to manage the file**.
  - It **includes sequence numbers used internally** by **NTFS** and **pointers** to the **file's other attributes** and **free space within** the **record**.
  - (Note that the header is part of the MFT record but not an attribute.)
- **Standard Information Attribute (SI):**
  - This **attribute contains "standard" information stored** for **all files** and **directories.**
  - This includes **fundamental properties such as date/time-stamps** for when the **file was created, modified** and **accessed**.
  - It also contains the **"standard" FAT-like attributes** usually associated with a file (such as whether the file is **read-only, hidden**, and so on.)

File attributes

- **File Name Attribute (FN):**
  - This attribute stores the name associated with the file.
  - Note that a file can have multiple file name attributes, to allow the storage of the **"regular" name** of the **file**, along with an MS-DOS short filename alias and also POSIX-like hard links from multiple directories.
- **Data (Data) Attribute:**
  - **This attribute stores** the **actual contents of** the **file**.
- **Security Descriptor (SD) Attribute:**
  - This attribute contains **security information that controls access** to the **file**.
  - The file's **Access Control Lists (ACLs)** and related data are stored here.

Larger files

- If the file is too large for all of the attributes to fit in the MFT, NTFS begins a series of "expansions" that move attributes out of the MFT and and make them non-resident. The sequence of steps taken is something like this:
- 1. **First, NTFS will attempt to store the entire file in the MFT entry**, if possible. This will generally happen only for rather small files.
- 2. **If the file is too large to fit in the MFT record**, the **data attribute is made non-resident**. **The entry for** the **data attribute** in the **MFT contains pointers to data runs** (also called **extents**) **which are blocks of data stored** in **contiguous sections of** the **volume, outside** the **MFT**.
- 3. The **file may become so large that there isn't even room in the MFT record for the list of pointers in** the **data attribute**. If this happens, the list of data attribute pointers is **itself made non-resident**. Such a file will have no data attribute in its main MFT record; instead, a pointer is placed in the main MFT record to a **second MFT record that contains the data attribute's list of pointers to data runs.**
- 4. **NTFS will continue to extend this flexible structure if very large files are created**. **It can create multiple non-resident MFT records** if needed to store a **great number of pointers to different data runs**. Obviously, the larger the file, the **more complex the file storage structure becomes.**

Extents (data runs)

- The **data runs (extents)** **are where most file data** in an **NTFS volume is stored**.
- These **runs consist of blocks** of **contiguous clusters on** the **disk**.
- The **pointers in** the **data attribute(s)** for the **file contain**
  - **a reference to the start of the run**,
  - and also the **number of clusters in the run**
- The **start of each run is identified** using a **virtual cluster number** or **VCN.**
- The use of a "**pointer+length**" scheme means that under NTFS,
  - it **is not necessary to read each cluster** of the file
  - in **order to determine where the next one in the file is located**.
  - This method also **reduces fragmentation** of files compared to the **FAT setup**.

NTFS File Size

- **This flexible system allows files to be extended** in **size virtually indefinitely.**
- **In fact, under NTFS**, **there is no maximum file size.**
- A **single file can be made to take up** the **entire contents of a volume** (less the space **used for the MFT itself** and other **internal structures and overhead**.)
- **NTFS also includes some features** that **can be used to more efficiently store very large files**.
  - One is **file-based compression**, which can be used to let large files **take up significantly less space**.
  - **Another is support** for **sparse files**, which is especially well-suited for certain applications that **use large files** that **have non-zero data in only a few locations.**

Limits

- **Table 13.5 NTFS Size Limits**
- **Description** **Limit**
- Maximum file size — **Theory**: 16 exabytes minus 1 KB (2^{64} bytes minus 1 KB)
  Implementation: 16 terabytes minus 64 KB (2^{44} bytes-64 KB)
- Maximum volume size — Theory: 2^{64} clusters minus 1 cluster
  Implementation: 256 terabytes minus 64 KB (2^{32} clusters - 1 cluster)
- Files per volume — 4,294,967,295 (2^{32} minus 1 file)

- **Table 13.6 FAT32 Size Limits**
- **Description** **Limit**
- Maximum file size — 4 GB minus 1 byte (2^{32} bytes minus 1 byte)
- Maximum volume size — 32 GB (implementation)
- Files per volume — 4,177,920
- Maximum number of files and subfolders within a single folder — 65,534
  (The use of long file names can significantly reduce the number of available files and subfolders within a folder.)

- **Table 13.7 FAT16 Size Limits**
- **Description** **Limit**
- Maximum file size — 4 GB minus 1 byte (2^{32} bytes minus 1 byte)
- Maximum volume size — 4 GBFiles per volumeApproximately 65,536 (2^{16} files)
- Maximum number of files and folders within the root folder — 512 (Long file names can reduce the number of available files and folders in the root folder.)

NTFS File Naming

- The **following are** the **characteristics of regular file names** (and directory names as well) **in** the **NTFS file system**:
- **Length:** Regular file names **can be up to 255** characters in NTFS.
- **Case: Mixed case is allowed in NTFS file** names, and NTFS will preserve the mixed case, **but references to file names are case-insensitive.**
- **Characters:** Names can contain any characters, including spaces, **except the following** (which are reserved because they are generally used as file name or operating system delimiters or **operators**): **? " / \ < > * | :**
- **Unicode Storage:** All NTFS **file names are stored in** a **format called Unicode..**
  - **Unicode is an international, 16-bit character representation format** that allow for thousands of different characters to be stored.
  - Unicode is supported throughout NTFS.

Alias and file name attribute

- You may recall that when Windows 95's **VFAT** file system **introduced** long file names to Microsoft's consumer operating systems, it provided for an **aliasing feature**. The file system automatically creates a short file name ("8.3") alias of all long file names, for use by **older software written before long file names were introduced**. NTFS does something very similar. **It also creates a short file name alias for all long file names,** for compatibility with **older software**. (**If the file name given to the file or directory is short enough to fit within the "8.3", no alias is created**, since it is not needed). It's important to realize, however, that the similarities between VFAT and NTFS long file names are mostly superficial. Unlike the VFAT file system's implementation of long file names, **NTFS's** implementation is not a kludge added after the fact.

- NTFS was designed from the ground up to **allow for long file names.**

- File names are stored in the **file name attribute** for every file (or directory), in the **Master File Table.**
  - **In fact, NTFS supports the existence of multiple file name attributes** within each file's MFT record.
  - **One of these is used for the regular name of the file**,
  - and if a short MS-DOS alias file name is created, it goes in a **second file name** attribute.
  - **NTFS supports** the **creation of hard links**

NTFS File Attributes

- **All file (and directory) attributes**
  - **are stored in one of two different ways**,
  - **depending on** the **characteristics of** the **attribute—**
  - **especially, its size**.
- The following are the methods that NTFS will use to store attributes:
- **Resident Attributes: Attributes that require a relatively small amount** of storage space are stored directly within the file's primary **MFT record itself**. **These are called resident attributes**. **For example**, the **name of the file**, and its creation, modification and access date/time-stamps are resident for every file.
- **Non-Resident Attributes: If an attribute requires more space than is available within the MFT record**,
  - **it is not stored in that record,** obviously.
  - **Instead**, the **attribute is placed** in a **separate location**.
  - A **pointer is placed within the MFT** that **leads to the location of the attribute**.
  - This is called **non-resident attribute storage**.

External attributes

- In practice, **only** the **smallest attributes can fit into MFT records**, since the records are rather small.
- **Many other attributes will be stored non-resident**, **especially** the **data of** the **file**, which is also an **attribute**.
- **Non-resident storage** can itself take **2 forms**.
  - If the **attribute doesn't fit** in the **MFT but pointers to the data do fit**,
  - then the data is placed in a **data run**,
  - also called an **extent**, **outside the MFT**,
  - and a **pointer to the run is placed in the file's MFT record**.
  - In fact, an attribute can be stored in many different runs, **each with a separate pointer.**
- **If the file has so many extents** that **even the pointers to them won't fit**,
  - the **entire data attribute may be moved to an external attribute**
  - in a **separate MFT record entry,**
  - or **even multiple external attributes**.

NTFS system defined attributes

- **Attribute List:**
  - This is a **"meta-attribute":**
  - an **attribute that describes other attributes**.
  - If **it is necessary for an attribute** to be **made non-resident**,
  - **this attribute is placed in the original MFT record**
  - to **act as a pointer to** the **non-resident attribute**.
- **Bitmap:**
  - **Contains** the **cluster allocation bitmap**.
  - **Used by** the **$Bitmap metadata file.**
- **Data:**
  - Contains **file data**.
  - By default, **all the data** in a file is stored in a **single data attribute—**
  - even if that attribute is broken **into many pieces due to size**,
  - it is **still one attribute—**
  - **but there can be multiple data attributes for special applications**.

NTFS system defined attributes

- **Extended Attribute (EA) and Extended Attribute Information:**
  - These are **special attributes**
  - that are implemented for compatibility with **OS/2 use of NTFS partitions**.
  - They are not used by Windows NT/2000 to my knowledge.

- **File Name (FN):**
  - **This attribute stores** a **name associated with** a **file or directory**.
  - Note that a file or directory can have **multiple file name attributes**, **to allow** the **storage of the "regular" name of** the **file**,
  - along with an MS-DOS short filename alias and
  - also **POSIX-like hard links from multiple directories**.

NTFS system defined attributes

- **Index Root Attribute:**
  - This **attribute contains** the **actual index of files contained within** a **directory,**
  - or **part of the index if it is large**.
  - **If the directory is small**, the **entire index will fit within this attribute in** the **MFT;**
  - **if it is too large**, some of the information is here and the rest is stored in **external index buffer attributes**.

- **Index Allocation Attribute:**
  - If a **directory** index **is too large to fit** in the **index root attribute**,
  - the **MFT record for the directory will contain** an **index allocation attribute**,
  - **which contains pointers** to **index buffer entries containing** the **rest** of the **directory's index information**.

NTFS system defined attributes

- **Security Descriptor (SD): This attribute contains security information** that controls access to a file or directory. **Access Control Lists (ACLs**) and related data are stored in this attribute. File ownership and auditing information is also stored here.

- **Standard Information (SI):** Contains "standard information" for all files and directories. This includes fundamental properties such as **date/time-stamps for when the file was created, modified and accessed**. It also contains the **"standard" FAT-like attributes** usually associated with a file (such as whether the file is read-only, hidden, and so on.)

- **Volume Name, Volume Information, and Volume Version:**
  - These three attributes store
  - **key name, version and other information about** the **NTFS volume**.
  - Used by the $Volume **metadata file**.

NTFS Reparse Points

- **ability to create special file system functions** and
- **associate them with files or directories**.
- This enables the functionality of the NTFS file system to be **enhanced** and **extended dynamically.**
- **The feature is implemented using objects** that are **called reparse points**.
- **Use of reparse points begins with applications**.
- An application that wants to use the feature
  - stores data specific to the application-
  - which can be any sort of data at all—
  - into a reparse point.
- The **reparse point is tagged with** an **identifier**
  - **specific** to the **application**
  - and **stored with** the **file or directory**.
- A **special application-specific filter** (a driver of sorts)
  - is also associated with
  - the **reparse point tag type**
  - **and made known to the file system**.

Special reparse points

- **Symbolic Links:**
  - **Symbolic linking** allows you to **create a pointer** from one area of the directory structure
  - to the **actual location** of the **file elsewhere in** the **structure**.
  - a **symbolic link is a reparse point** that **redirect access from one file to another file.**

- **Junction Points:**
  - A **junction point** is **similar to a symbolic link**,
  - **but instead** of redirecting access from one file to another,
  - **it redirects access from one directory to another**.

Special reparse points

- **Volume Mount Points:**
  - A **volume mount point** is like a symbolic link or junction point,
  - but taken to the next level:
  - **it is used to create dynamic access to entire disk *volumes***.
  - For example, you can create volume mount points for removable hard disks or other storage media, or even use this feature to allow several different partitions (C:, D:, E: and so on) to **appear to the user as if they were all in one logical volume**.
  - **Windows 2000 can use this capability to break the traditional limit of 26 drive letters--using volume mount points**, you can access volumes without the need for a **drive letter for the volume**. This is useful for large CD-ROM servers that would otherwise require a separate letter for each disk (and would also require the user to keep track of all these drive letters!)
- **Remote Storage Server (RSS):**
  - This feature of Windows 2000 **uses a set of rules to determine**
  - **when to move infrequently used files on an NTFS volume to archive storage** (such as CD-RW or tape).
  - When it moves a file to "offline" or "near offline" storage in this manner,
  - RSS leaves behind reparse points that contain the instructions necessary
  - to access the archived files, if they are needed in the future.

NTFS Security and Permissions

- **General NTFS Security Concepts**
- access rights for files and directories **based** on **user** or **group** accounts.
- **There are 3 other important overall concepts** in **NTFS security**:
  - **object ownership**
  - **permission inheritance**
  - **auditing**.
- **Ownership** is a **special property right for NTFS objects** that gives file **owners** the **capability of granting permissions to others**.
-
- NTFS is also designed to **propagate permissions** down the hierarchy of the directory structure, under the control of the user.
- This **permission inheritance feature allows permissions to be assigned to groups of objects automatically**. It also allows permissions to be automatically **applied to new files that are created within an existing directory structure.**
- Finally, **auditing** allows **administrators to monitor changes to files or directories.**

Access Control Lists (ACLs) and Access Control Entries (ACEs)

- **Management** of **security** and access to NTFS objects
- **begins** in the **same place where everything else begins in NTFS**:
- in the **Master File Table (MFT).**

- The **MFT record** for every file and directory on an **NTFS volume contains** a **security descriptor (SD) attribute**. The name of this attribute makes rather clear what it contains: **information related to security and permissions for** the **corresponding object.**

- **One of the most important elements** within the security descriptor for any object is the **set of lists** within it, **which dictate which users may access the object, and in what manner.**

Access Control Lists (ACLs) and Access Control Entries (ACEs)

- These are called *access control lists* or *ACLs*.

- Every object in an NTFS partition **has 2 different types of access control lists**:
- **1. System Access Control List (SACL):**
  - **This ACL is managed** by the **system** (thus the name) and
  - is used to control **auditing** of **attempts to access** the **object.**

- **2. Discretionary Access Control List (DACL):**
  - This is the **"real" ACL. :^)**
  - Well, it is the one that most people are primarily concerned with,
  - because it is where permissions are stored that control what users and groups of users are allowed what type of access to the object.
  - If you hear someone refer to an object's ACL in the singular, this is the one they mean.

ACE

- **each entry in** an **ACL** **is called** an **access control entry** **or** **ACE**.

- **Each ACE contains:**
  - **an ID code** that identifies the **user or group to which the ACE applies**,
  - and then information about the specific **permission settings** that are to be applied to that user or group.

- **Many different ACEs can be placed into a list**,
- allowing the access of various types to be granted or denied for a variety of **different users and groups**.

- Some groups have special meaning, such as group "**Everyone**".

NTFS Permissions

- When **Windows NT** was built, **six different permission** types were created for NTFS objects.

Permission Type	Abbreviation Letter	Permission Granted For Files	Permission Granted For Folders
Read	R	Read file contents	Read folder contents
Write	W	Change file contents	Change folder contents (create new files or subfolders)
Execute	X	Execute (run) a program file	Traverse subfolder structures of folder
Delete	D	Delete file	Delete directory
Change Permissions	P	Change file's permission settings	Change folder's permission settings
Take Ownership	O	Take file ownership	Take folder ownership

- When **Windows 2000** was introduced, the six permission types above were "broken down" into **13 different permission components**, to allow for more "fine-tuned" control over different kinds of access.

Permission Components (Windows 2000 and Windows NT 4.0 SCM)	Permission Types (Windows NT)					
	Read (R)	Write (W)	Execute (X)	Delete (D)	Change Permissions (P)	Take Ownership (O)
Traverse Folder / Execute File			✓			
List Folder / Read Data	✓					
Read Attributes	✓		✓			
Read Extended Attributes	✓					
Create Files / Write Data		✓				
Create Folders / Append Data		✓				
Write Attributes		✓				
Write Extended Attributes		✓				
Delete Subfolders and Files						
Delete				✓		
Read Permissions	✓	✓	✓			
Change Permissions					✓	
Take Ownership						✓

Standard Permission Groups

- To **avoid the necessity**
- of **always setting low-level permissions**,
- **Windows** defines **standard permission groups.**

- Under the more advanced **Windows 2000 scheme**,
- there are **13 different permission components**,
- **which are collected into 6 different standard groups**

Standard Permission Group	Object Types Affected	Permission Types Granted (Applies Only To Appropriate Object Types)						Description
		Read (R)	Write (W)	Execute (X)	Delete (D)	Change Permissions (P)	Take Ownership (O)	
No Access	Folders or Files							Denies all access to the file or folder. The user can see the name of the object, but cannot do anything with it.
List	Folders Only	✓		✓				Users can see the list of files in the folder and traverse subfolders, but cannot view or execute files.
Read	Folders or Files	✓		✓				Users can read files and folders, execute files and traverse folders, but cannot change anything.
Add	Folders Only		✓	✓				Users can add files or subfolders to the folder, and can traverse subfolders, but cannot read or execute files.
Add & Read	Folders Only	✓	✓	✓				Users can add files or subfolders to the folder, and can read and execute files in the folder as well.
Change	Folders or Files	✓	✓	✓	✓			The user can read, write, execute or delete the file, or if applied to a folder, the files and subfolders within the folder. Note that this does *not* grant access to delete the folder itself. The user also cannot change permissions on the file or folder, or take ownership of it.
Full Control	Folders or Files	✓	✓	✓	✓	✓	✓	All permissions are granted. This also includes the special permission "Delete Subfolders and Files", which can only be given through the "Full Control" group under Windows NT.

Ownership and Permission Assignment

- **Every object** within the NTFS volume **has an owner**,
- which is a **user identified by the object as being** the **one who controls it.**

- **By default**, the **user who creates** a **file** or **folder becomes its owner.**

- The **significance of ownership** is that the owner of a file or folder always has the ability to **assign permissions for that object**.

- The owner can decide what permissions should be applied to the object, controlling others' access to the file or folder.

Ownership and Permission Assignment

- The two **special permissions** that are **associated with ownership** and **permission assignment are:**
  - "**Change Permissions**" **(P)**
  - "**Take Ownership**" **(O)**

- If a user is **granted the "Change Permissions" permission**, the user **can change the permission settings** for the **object even if he or she does not own it.**

- If a **user has "Take Ownership**" permission,
  - the **user has the ability to take over ownership of the resource**,
  - and of course,
  - once **it is owned the user can do anything he or she wants with** the **permissions**.

Static Permission Inheritance

- When you are using Windows NT and
- **create a new subfolder** or **file within a folder**,
- the new object is given a default set of permissions
- by **copying**
- **the current set of permissions** associated with the object's **parent folder.**

- This is called permission **inheritance,** or sometimes**, propagation**.

- Under NT's inheritance model, this only happens once, at the time the object is created. For this reason, conventional inheritance under NT is also called **static permission inheritance**,

- to distinguish it from the **dynamic inheritance used by Windows 2000**.

Dynamic Permission Inheritance and Advanced Inheritance Control

- This dynamic linking **method solves**
- the **two biggest problems with** the **static inheritance model.**

- First
  - any **changes** to the **parent folder**
  - are **automatically inherited by the child objects**.

- Second
  - **any changes that were made to** the **child object**
  - are **not destroyed** by this **automatic propagation**.

Windows 2000 advanced inheritance

- **Child Protection:**
  - The **main security properties dialog box** for **each object contains** a check box labeled "**Allow inheritable permissions from parent to propagate to this object**".
  - If the check in this box is **cleared**,
  - this **breaks** the **normal inheritance link between** this child and its parent (and **higher-level ancestors** as well).
  - When this is done, the child will **no longer dynamically inherit permissions** from **higher up in the directory tree**. Such a child object is said to be **protected from inheritance changes**.

- **Object Selection Control:**
  - **When changing permissions on a folder**,
  - **you can choose if the permissions** will **be applied** to
  - **any combination** of the **folder itself**, **files within it**,
  - or **subfolders within it.**

Windows 2000 advanced inheritance

- **Recursion Control:**
  - An option exists in the dialog box where individual permissions are assigned called "**Apply these permissions to objects and/or containers within this container only**".
  - The name of this option is **horribly confusing**.
  - What it means is that, if selected, permissions you choose are applied only to the folder's **immediate children**, but **not lower-level objects**.
  - So if this were chosen as we selected a permission for the "C:\Documents" folder in the example above, changes would propagate to "C:\Documents\Exec" but *not* "C:\Documents\Exec\Payroll-Projections", the item two levels down.

- **Forced Propagation:**
- An option called "**Reset permissions on all child objects and enable propagation of inheritable permissions**" is provided. This works the same way as the "**Replace Permissions on Subdirectories**" and "**Replace Permissions on Existing Files**" options from the older Windows NT static permission model.
- When selected, **NTFS will force propagation down to all child objects** and **remove any permissions that were directly assigned to those child objects**. This allows administrators to easily "**fix**" **permission problems in large directory structures.**

Permission Resolution

- Every time a user attempts a particular type of access to an object on NTFS, the **system must determine** if the access **should be allowed**.
- In theory, this is a simple matter of looking at the **access control lists** for the object, seeing what the permission settings are for the user, and determining if the desired **access is allowed**.
- Unfortunately, in reality, it's not this simple. :^)
- Since every object can have **many different permission settings**, it is possible that several different permission settings might apply to a particular object and access method.
- Furthermore, it is possible that these permission settings **might conflict**. When this occurs, the system must engage in a process of **resolving** the various permissions to determine which ones should **govern the access**.

Auditing

- **When auditing is enabled**, the **system can be set to keep track of certain events**.
  - When any of these events occur, the system will make an entry in a **special auditing log file** that can be read by administrators or others with the appropriate permission level.
  - Each entry will indicate the type of event, the date and time that it occurred, which user triggered the event, and other relevant information.

- **Auditing within NTFS is really just a small part** of the various auditing features offered by the **Windows NT** and **Windows 2000 operating systems.**
- These tools allow administrators to keep track of everything from logins, to the **use of printers, to system errors**. Within NTFS, auditable events are generally accesses of various types, roughly corresponding to the different types of **permissions**.
- **Auditing can be selected for files and for folders**, and can be selected for individual objects or **hierarchies of folders**, just like permissions can.

NTFS Reliability Features

- **NTFS was designed** as a **transaction-based or transactional file system**
- A **special activity log** **is maintained by the system**
- (in fact, it is one of NTFS's **metadata files).**

- **Every time a change is made** to any part of the volume,
- the **system records the change in** the **activity log**.

- **These changes include:**
  - **creation**
  - **deletion**
  - **or modification**
  - **of files or directories**

Transaction Recovery

- **When recovery is performed**,
  - the **file system examines** the **NTFS volume**,
  - **looking at** the **contents of** the **activity log**.

- **It scans all log entries back** to the **last checkpoint**, and **performs** a **three-pass recovery procedure**:
- **Analysis Pass: The system analyzes the contents** of the **log**
  - **to determine**
  - **what parts of** the **volume need** to be **examined**
  - **and/or corrected.**
- **Redo Pass:**
  - The system **"redoes" all completed transactions**
  - that **were recorded since the last checkpoint**.
- **Undo Pass:**
  - The **system "undoes" (or rolls back)** all **incomplete transactions to ensure file integrity.**

Change (USN) Journals

- Under Windows 2000, NTFS 5.0 partitions **can be set to keep track of changes to files and directories on** the **volume**, **providing** a **record** of **what was done to** the **various objects and when**.
- **When enabled**, the **system records all changes** made to the volume in the **Change Journal,** which is the name also used to describe the feature itself.
- Change Journals work in a fairly simple manner. **One journal is maintained for each NTFS volume**, and **it begins as an empty file**.
- Whenever a **change** **is made to the volume**, a **record is added to the file.**
- **Each record** **is identified** by a **64-bit Update Sequence Number** or **USN.** (In fact, Change Journals are sometimes called **USN Journals**.)
  - **each record** in the **Change Journal** **contains** **the** **USN**, the **name** of the file, and **information** about **what** the **change was**
  - **change Journal will contain an entry that indicates** that the **data was written**, but not the **contents of** the **data itself**.

Change (USN) Journals

- **For starters**, **it could be very useful** for **system-level utilities**.
- For example, **anti-virus programs** could make **use of change journals** to **detect unauthorized changes to files**.
- **Backup programs** could also make use of the **facility to determine** which files had changed since the last time a **backup was performed**.
- Programs that **perform system management** tasks such as **archival or replication** could also make **good use** of **this feature**.

Error Correction and Fault Tolerance

- **NTFS includes several fault tolerance features.**
- some of these capabilities are implemented through the use of the **NTFS fault-tolerant disk driver**, called "FTDISK"

- **Transactional Operation:**
  - The way that NTFS **handles transactions** as atomic units,
  - and allows **transaction recovery**,
  - are **key fault tolerance features** that I have described elsewhere in this section.
  - **Recovery is performed automatically whenever the system is started**.

- **Software RAID Support: NTFS partitions** can be set up to use **software RAID if the appropriate version of Windows NT or 2000 is used**. For more information, see the full discussion of **RAID.**

Error Correction and Fault Tolerance

- **Dynamic Bad Cluster Remapping:**
- When the fault-tolerant disk driver is used, the file system has the ability to automatically detect **when bad clusters have been encountered** during **read or write operations.**
-
- When a **bad cluster is found**, the **file system will automatically relocate** the **data from** the **bad location** and **mark** the **cluster bad so it will not be used in the future**.
- Now, the **FAT** file system includes the **ScanDisk** utility that can do this as well, but you **must run it manually-**-with **NTFS this can be done automatically**.
  - Furthermore, **ScanDisk can only identify clusters** that have **already gone bad**, at which point, **data may be lost**.
  - **FTDISK driver will actually read back data as it is written** (sometimes called a "**verify" operation**) ensuring that data is unlikely to be lost due to a bad cluster at the time of a write. (Bear in mind, however, that it is possible for an area of he disk to "go bad" between the time that the data is written and the time that it is read back.)

Volume Management and Fault Tolerance

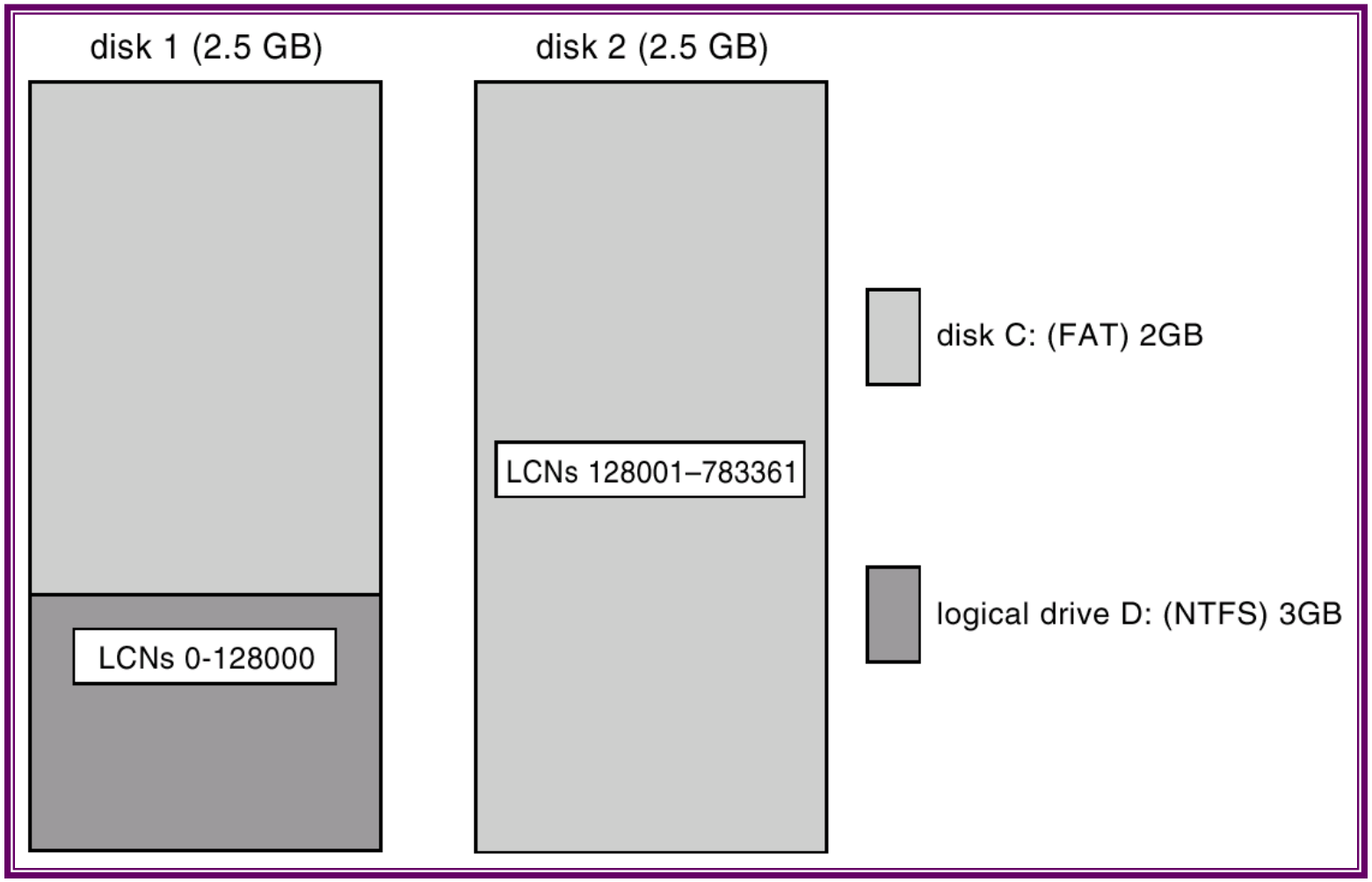
- FtDisk, the **fault tolerant disk driver for 2000**, provides several ways to combine multiple SCSI disk drives into one logical volume.
- **Logically concatenate** multiple disks to form a **large logical volume**, a ***volume set***.

- **RAID level 0:** Interleave multiple physical partitions in round-robin fashion to form a *stripe set* (also called **RAID level 0**, or "disk striping").
  - ☞ Variation: *stripe set with parity,* or RAID level 5.

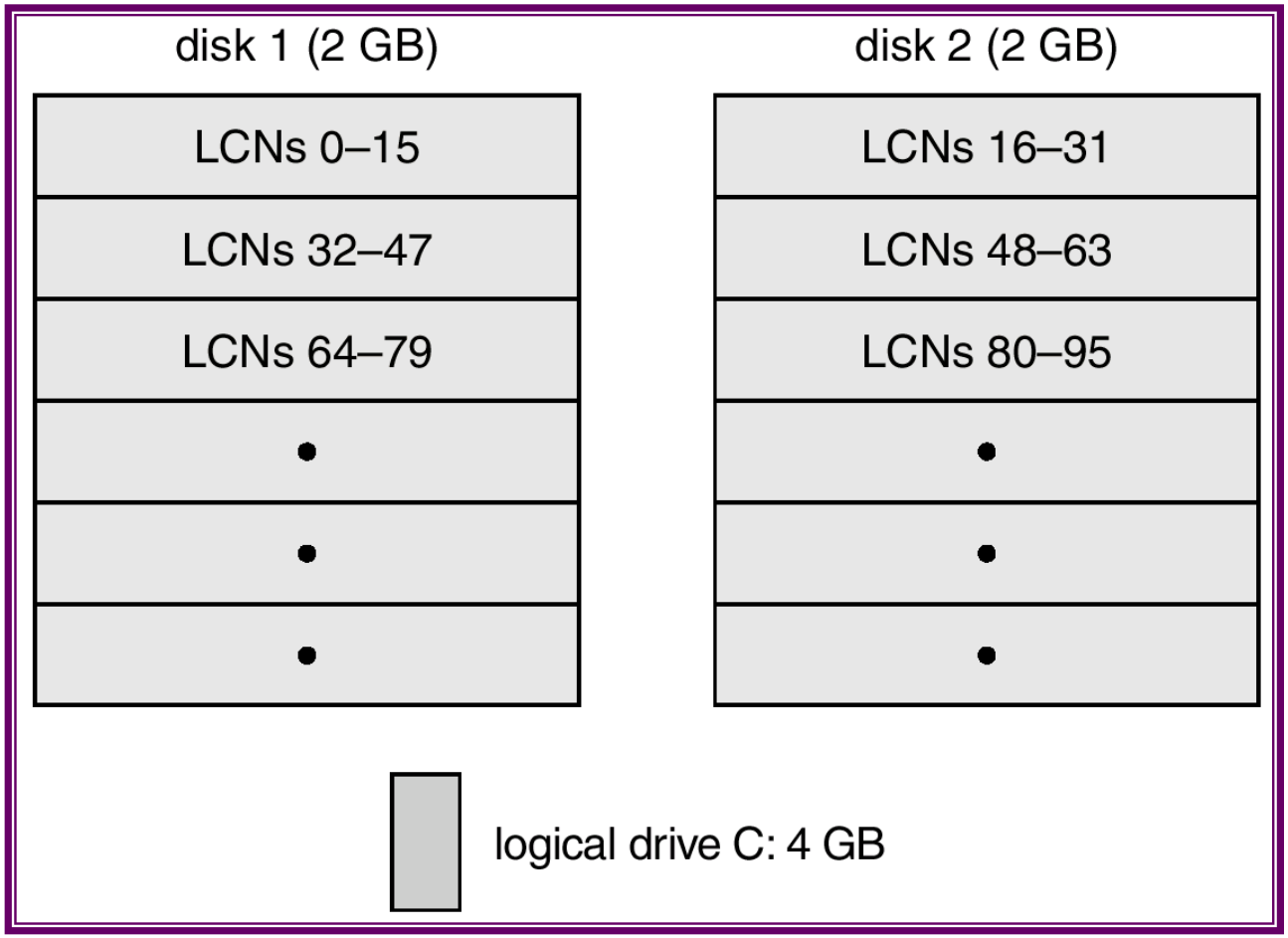
- **Disk mirroring**, or RAID level 1, is a robust scheme that uses a *mirror set* — two equally sized partitions on tow disks with identical data contents.

- 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***.

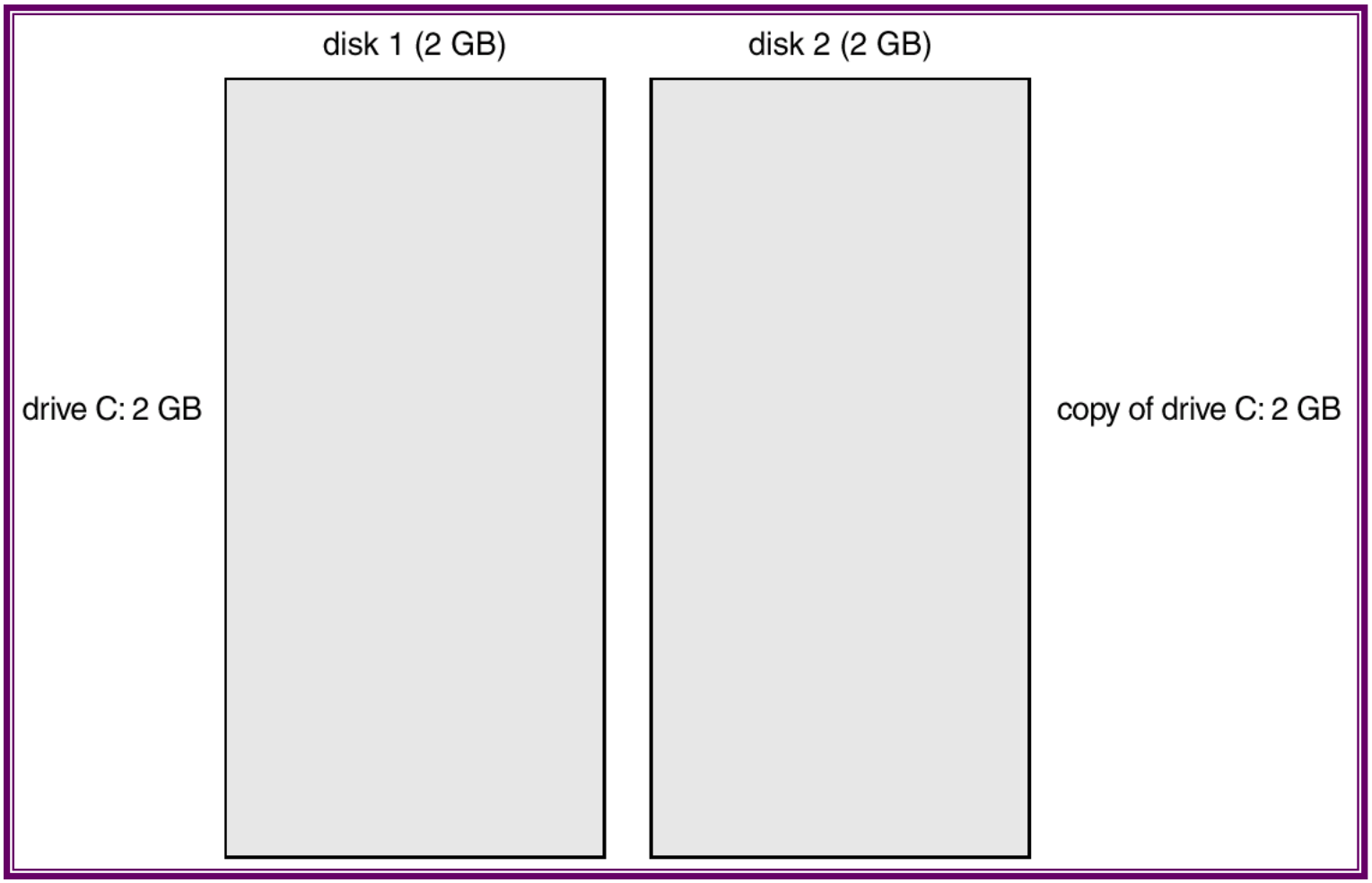
Volume Set On Two Drives



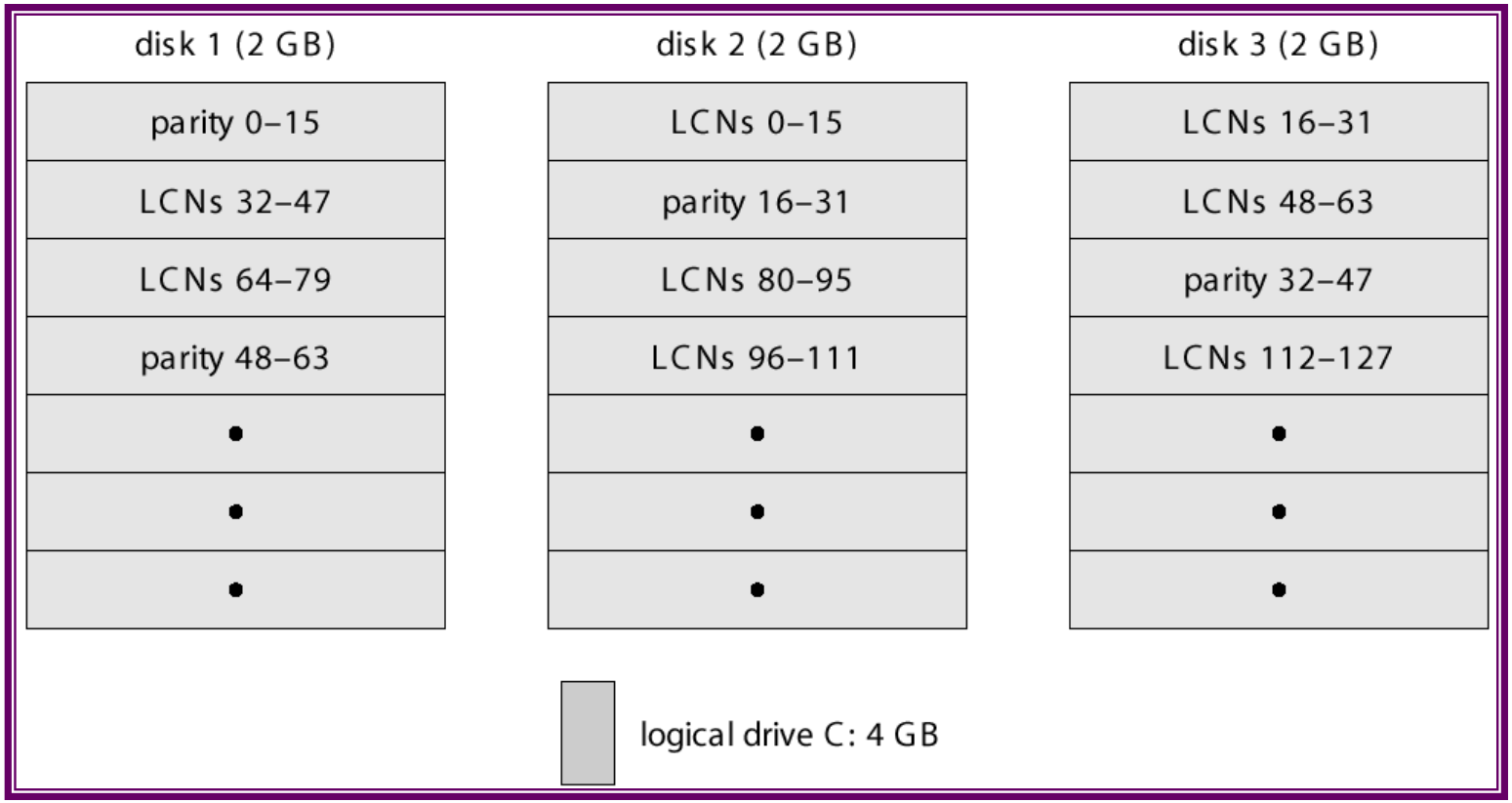
Stripe Set on Two Drives – RAID 0



Mirror Set on Two Drives – RAID 1



Stripe Set With Parity on Three Drives-RAID 5



Fragmentation and Defragmentation

- In fact, **due to their complexity, NTFS volumes suffer from a variety of different types of fragmentation.**
- Unlike FAT, where a **simple cluster allocation system is used**, NTFS uses the **Master File Table and a combination of resident and non-resident attributes to store files**.
- Due to **the flexible way that data is stored**, and that additional data storage areas are added as needed, the result can be pieces of data spread out over the volume, particularly when small files grow into large ones. Remember that while NTFS has a much better design than FAT, at its core it does still store data in **clusters.** The addition and removal of data storage *extents* causes much of the fragmentation of files and directories. As the **MFT grows**, it itself can become **fragmented, reducing** performance further.
- In its wisdom, **Microsoft decided to license** the **Diskeeper defragmenter technology and include it in Windows 2000**, so the operating system now includes a **built-in defragmenter**, though it is likely either less capable or slower than the full Diskeeper program sold by Executive Software

Other NTFS Features and Advantages

- NTFS offers a
  - superior architecture,
  - support for larger files,
  - security features such as access control and logging,
  - enhanced reliability

- Other features:
  - Compression
  - POSIX support
  - Encryption
  - Disk quota
  - Sparse file suport

Compression

- **built into NTFS is file-based compression**
  - ☞ that can be used to **compress individual files or folders**

- Under NTFS, you can easily compress one or more files or folders **by opening their properties** and telling the operating system to compress them.

- The **compression is handled by** the **operating system during writes**,
- and **decompression is automatic whenever** an **application needs to read** the **file.**

POSIX support

- POSIX is a **set of standards that** was **created to enhance** the **portability of applications between computer systems**
- There are actually a number of different POSIX standards.
- **NTFS specifically supports the POSIX.1 standard**, which is the standard for application program interfaces **based on the "C" programming language.**
- In practical terms, POSIX support is manifested most obviously in NTFS's support for special **file naming** provisions.
- For example, NTFS **allows for file names** to be **case-sensitive**, and also allows for **hard links to be established**, enabling multiple names for the same file within the file system. The "**last accessed**" date/time stamp for files is also part of **POSIX support.**

Encryption

- **encryption capability** in **NTFS 5.0**, as part of Windows 2000.
- This feature is called the **Encrypting File System or EFS**.
- Using EFS, it is possible to **encrypt important** data before storing it **on the NTFS partition**.
- **Without the proper decryption key, the data cannot be accessed**.
- This makes it impossible for anyone to easily access data stored on NTFS volumes by booting the PC with a floppy disk and using a disk sector editor, for example.
- It also offers some peace of mind to those who carry critically sensitive information around on notebook PCs, which are frequently lost--or "liberated", if you know what I mean...

Disk Quotas

- The **quota system implemented** in NTFS 5.0 is quite flexible and **includes many capabilities**:

- You have the ability to do the following:

  - **Set quotas on a per-user or per-volume basis**. This lets you limit space used on particular disks, or overall total space use for a person.

  - **Set a "limit" level** and a **"warning" level**, or both. The user is blocked from using any disk space above the "limit" level. He or she may use space beyond the "warning" level, but a warning will be generated.

  - **Monitor and log events** that **cause** a **user to go over** the **"limit"** or **"warning"** levels.

Sparse File Support

- To **improve the efficiency of storing sparse files** without necessitating the **use of compression**, a special feature was incorporated into NTFS under Windows 2000.

- A **file that is likely to contain many zeroes can be tagged as a sparse file**. **When this is done, a special attribute is associated with the file**, and it is stored in a different way from regular files.

- The **actual data in the files is stored on the disk**,
  - and **NTFS keeps track of where in** the **file each chunk of data belongs.**
  - The **rest** of the file's bytes-**--the zeroes--are not stored on disk.**

- NTFS seamlessly manages sparse files so that they **appear** to applications **like regular files**.

- So, for example, when an application asks to read a particular sequence of bytes from a sparse file, NTFS will **return both regular data and zeroes, as appropriate, just as it would for a regular file.**
- **The application can't** tell that NTFS is playing "storage games" on the disk.

NTFS Performance and Overhead Considerations

- The performance implications of NTFS features mean that using **NTFS may result in a slight decrease in performance compared** to the use of a simpler **file system such as FAT**.
- Here are a few tips and issues to keep in mind when considering how **to implement NTFS partitions**, which **may partially mitigate** this **performance impact**:

- **Use Only What You Need:**
- Some of the fancier features in NTFS can **impose more significant overhead penalties** on the system than others, both in terms of **processing power** and the use of **disk space**.
  - If you **need those features**, then in most cases they are **worth** the **cost**.
  - However, it makes sense to avoid using features that you don't need.
  - For example,
  - don't **compress frequently used files** if you have **lots of disk space**;
  - don't enable lots of **audit** logging unless you really require it, and so on.

NTFS Performance and Overhead Considerations

- **Use Adequate Hardware:**
  - Trying to implement a complex system on a low-powered system with a small, **slow hard disk** is probably asking for trouble.
  - The overhead associated with NTFS becomes less and less noticeable on **newer hardware** (which is part of what is allowing Microsoft to add even more new features to the file system.)
- **Watch The Cluster Size:**
  - While most people think of cluster size as being strictly a FAT file system issue, **NTFS also uses clusters**, and its performance is impacted by the **choice of cluster size**.
  - Cluster size is in turn affected by the choice of partition size. In addition, under Windows NT the cluster size is different if the partition was converted from FAT.
- **Fragmentation and Defragmentation:**
  - Contrary to a popular myth, NTFS partitions do become fragmented and performance can benefit from regular defragmenting.
  - Windows 2000 comes with a **defragmenter built into it**; for Windows NT you will need to use a third-party tool.

NTFS Partitioning Strategies

- here are a few specific points that you may want to **keep in mind when partitioning a system with NTFS:**
- **Limit the Number of Partitions:**
  - NTFS is designed to be able to gracefully handle **much larger partitions than FAT can**.
  - As a result, **you should avoid the temptation to "go overboard" and segment large disks or RAID arrays into many little volumes**.
  - Doing this makes the system more difficult to manage in most cases and results in not much improvement in performance.

- **Consider A Dedicated Operating System Partition:**
  - Many people who set up NTFS systems **use two partitions.**
  - **The "C:" (boot) volume is made smaller (a few gigabytes) and dedicated to the operating system and the installation of low-level utilities and administration tools.**
  - **The** other volume (normally "D:") is used for applications and data. This split may make administration easier and avoid problems associated with using too large a boot partition size with Windows NT.

NTFS Partitioning Strategies

- **Adjust Cluster Sizes If Needed:**
  - The default cluster size on an NTFS partition can be overridden, as described here.
  - You may wish to use larger or smaller clusters than the system would normally select, depending on your needs.
  - For example, you may want a **larger cluster size** for a partition that will be holding **large multimedia files**.
  - Do be aware that there can be consequences to going away from the default values. For example, NTFS compression will not work on a volume that uses clusters greater than 4 Kib in size.
- **Beware of Converting Partitions to NTFS Under Windows NT:**
  - Converting a partition from FAT to NTFS under Windows NT
  - results in the NTFS partition being assigned the **smallest possible cluster size, 512 bytes**.
  - This may cause a degradation in performance.
- **Multiple Operating System Compatibility:**
  - Some systems use multiple operating systems, some of which cannot natively read and write NTFS partitions. For example, Windows 9x/ME cannot read and write NTFS partitions.
  - On these machines, some people create one FAT partition, in addition to any NTFS partitions, for compatibility and file transfer purposes.
  - Note that this is not required for accessing files on an NTFS partition over a network; files on an NTFS partition can be shared across a network even if the network client's operating system cannot use NTFS locally.

NTFS vs. FAT

- Most frequently, the **question of NTFS vs. FAT**
  - is answered
  - by looking at the **advantages and disadvantages of NTFS**,
  - and **comparing it to the simpler FAT file system**.

- This can be made easier by assessing 3 **general questions**:
  - **1. Do you need the added features** and **capabilities that NTFS provides but FAT does not?**
  - **2. Are you willing to accept the additional hardware requirements necessary to use NTFS**, and to **deal with its drawabcks and limitations**?
  - **3. Can you invest the additional time and resources for proper administration of** an **NTFS system**?