

NTFS File System

rapfer@gmail.com

이경식(nofate)

08/12/30

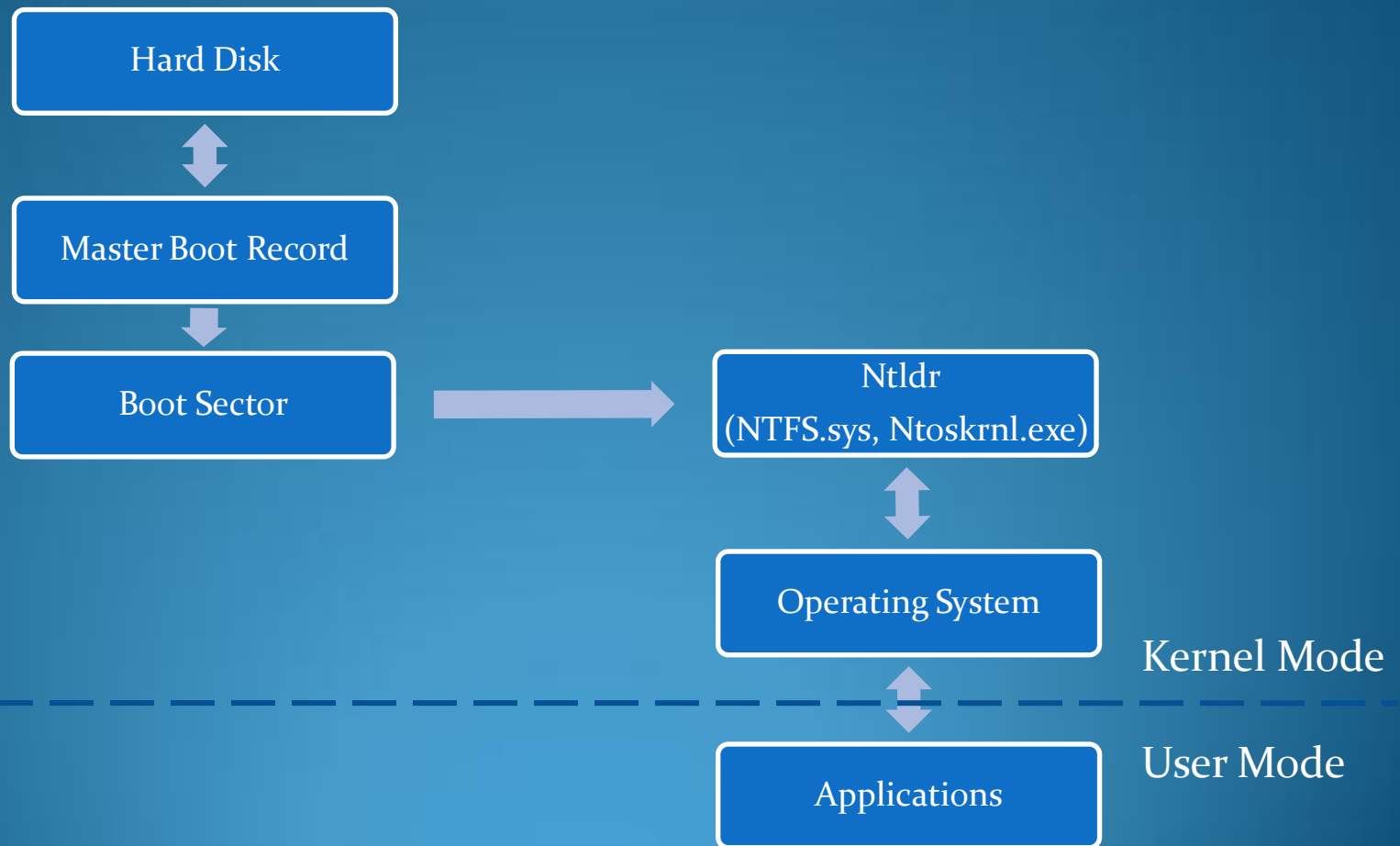
목차

1. NTFS Architecture
2. NTFS Boot Record
3. NTFS Master File Table
4. NTFS Attribute
5. Data Integrity and Recoverability with NTFS
6. NTFS Index

NTFS Architecture

NTFS에 대한 기본적인 내용과 기본 구조에 대해 알아보자

NTFS Architecture



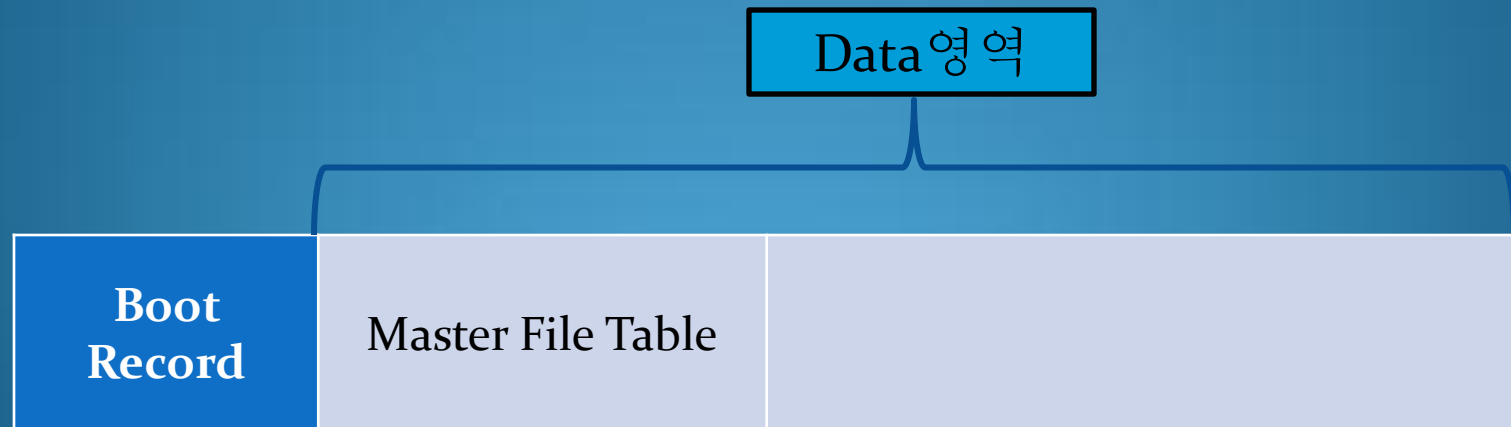
NTFS Architecture

Component	Component 설명
Hard Disk	하나이상의 파티션을 포함
Boot Sector	Ntdlr이 로드하는 파일시스템 구조와 볼륨레이아웃에 대한 정보를 저장하고 있는 부팅 가능한 파티션
Master Boot Record	메모리로 시스템 바이오스를 로드 하기 위한 실행 코드를 포함, 활성파티션과 부팅 가능한 파티션에 대한 정보를 가진다
NTLDR.dll	CPU를 보호모드로 변경하고, File system을 시작, Boot.ini를 읽는다.
NTFS.sys	NTFS System File Driver
Ntoskrnl.exe	시스템 디바이스 드라이버의 로드에 대한 정보
Kernel Mode	시스템의 모든 하드웨어와 메모리에 접근 가능한 레벨
User Mode	어플리케이션 구동이 가능한 레벨

NTFS Structure

- NTFS 구조

- 모든 데이터를 파일의 형태로 관리
 - MFT(Master File Table) 또한 데이터 영역
 - 즉, File System 관리 데이터도 파일로 존재
 - 성능과 안정성을 위해 실제로는 구조를 잡아놓음



NTFS Structure

- Boot Record

- Jump Code + OEM ID + BPB(BIOS Parameter Block)
- Windows의 부팅코드와 여러 설정 값을 가짐
- 손상 시 FileSystem 인식이 불가능
- NTFS시스템에서는 MFT의 위치를 가리키는 것이 주 목적이다.
- 볼륨 영역을 여러 개로 나누지 않으므로, FAT파일시스템에 비해 BR이 간단한 편

NTFS Structure

- Master File Table

- 볼륨에 존재하는 모든 파일과 디렉터리 정보를 가짐
- MFT Entry의 집합
 - MFT Entry 0~15번까지는 시스템파일용으로 예약
 - 예약된 부분을 meta data file이라 부름(\$ 표시)

- Data 영역

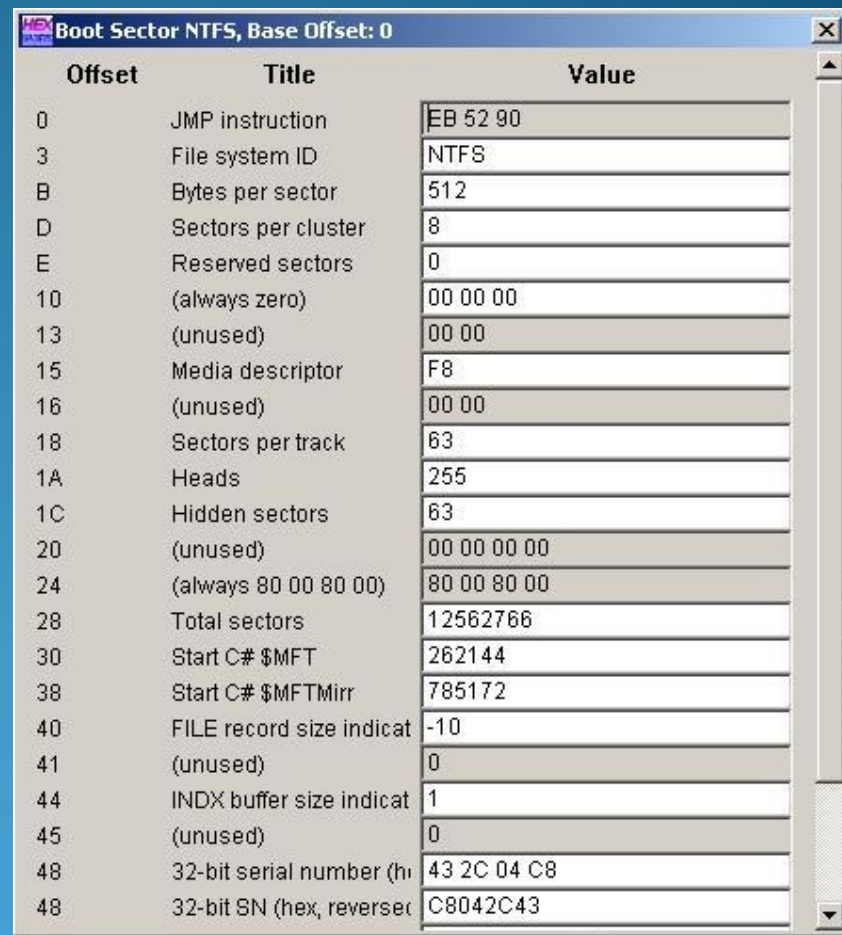
- 파일과 디렉터리를 담는 영역
- 클러스터 단위로 읽기/쓰기를 수행
- 볼륨 크기는 최대 4KB로 지정(압축 기능을 위해)

NTFS Boot Record

NTFS Boot Record에 대한 소개 및 분석

Boot Record

- 부팅코드와 여러 설정 값을 가짐
- WinHEX을 이용하여 원하는 파일 시스템에 대한 기본 구조를 확인
 - View → Template Manager
- Jump Boot Code(3byte) – EB 52 90
 - x86계열 부팅 시 이 항목을 읽어 해당 부트영역으로 점프
 - EB 52 – JMP 7C54
 - 90 – NOP
- OEM ID(8Byte) – ‘NTFS ‘
 - 제조사 또는 시스템의 이름을 기록

A screenshot of the WinHEX application showing the boot sector of an NTFS file system. The window title is 'Boot Sector NTFS, Base Offset: 0'. It displays a table with three columns: 'Offset', 'Title', and 'Value'. The data is as follows:

Offset	Title	Value
0	JMP instruction	EB 52 90
3	File system ID	NTFS
8	Bytes per sector	512
D	Sectors per cluster	8
E	Reserved sectors	0
10	(always zero)	00 00 00
13	(unused)	00 00
15	Media descriptor	F8
16	(unused)	00 00
18	Sectors per track	63
1A	Heads	255
1C	Hidden sectors	63
20	(unused)	00 00 00 00
24	(always 80 00 80 00)	80 00 80 00
28	Total sectors	12562766
30	Start C# \$MFT	262144
38	Start C# \$MFTMirr	785172
40	FILE record size indicat	-10
41	(unused)	0
44	INDX buffer size indicat	1
45	(unused)	0
48	32-bit serial number (h	43 2C 04 C8
48	32-bit SN (hex, reverser	C8042C43

Boot Record Analysis

- 0x0B - BytesPerSector(2Byte) – 0x00 20
 - Sector 당 바이트 수(4096바이트까지 가능)
- 0x0D - SectorPerCluster(1Byte) – 0x08
 - Cluster 당 Sector 수
- 0x0E - Reserved Sector(FAT시스템을 위해 존재) - 0x00 00

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00000000	EB	52	90	4E	54	46	53	20	20	20	20	00	02	08	00	00
00000010	00	00	00	00	00	F8	00	00	3F	00	FF	00	3F	00	00	00
00000020	00	00	00	00	80	00	80	00	4E	B1	BF	00	00	00	00	00
00000030	00	00	04	00	00	00	00	00	14	FB	0B	00	00	00	00	00
00000040	F6	00	00	00	01	00	00	00	43	2C	04	C8	67	04	C8	72
00000050	00	00	00	00	FA	33	C0	8E	D0	BC	00	7C	FB	B8	C0	07

Boot Record Analysis

- 0x15 - Media(1Byte) – 0xF8
 - DOS에서 사용하는 값, 0xF8 – Hard Disk, 0xF0 – FDD
- 0x28 - Total Sector(8Byte) – 가변
 - Volume의 총 Sector수
 - 저장장치의 총 섹터 수가 아님
 - 0x4EB1BF0000000000 → 12,562,766 sectors → ~6GB

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000000000	EB	52	90	4E	54	46	53	20	20	20	20	00	02	08	00	00
000000010	00	00	00	00	00	F8	00	00	3F	00	FF	00	3F	00	00	00
000000020	00	00	00	00	80	00	80	00	4E	B1	BF	00	00	00	00	00
000000030	00	00	04	00	00	00	00	00	14	FB	0B	00	00	00	00	00
000000040	F6	00	00	00	01	00	00	00	43	2C	04	C8	67	04	C8	72
000000050	00	00	00	00	FA	33	C0	8E	D0	BC	00	7C	FB	B8	C0	07

Boot Record Analysis

- 0x30 - Start of MFT(8Byte) – 가변
 - \$MFT의 시작 클러스터의 주소 - 0x040000
 - 이 값을 이용하여 MFT로 이동, 볼륨 분석을 하게 된다.
- 0x38 - Start of MFTMirr(8Byte) – 가변
 - MFT의 백업본의 시작 클러스터 주소 - 0xBFB14
 - MFT손상 시 이 항목을 이용하여 복구

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000000000	EB	52	90	4E	54	46	53	20	20	20	20	00	02	08	00	00
000000010	00	00	00	00	00	F8	00	00	3F	00	FF	00	3F	00	00	00
000000020	00	00	00	00	80	00	80	00	4E	B1	BF	00	00	00	00	00
000000030	00	00	04	00	00	00	00	00	14	FB	0B	00	00	00	00	00
000000040	F6	00	00	00	01	00	00	00	43	2C	04	C8	67	04	C8	72
000000050	00	00	00	00	FA	33	C0	8E	D0	BC	00	7C	FB	B8	C0	07

Boot Record Analysis

- 0x40 - MFT Entry Size(1Byte) – 0xF6
 - MFT Entry 크기 지정
 - 양수
 - MFT Entry는 해당 값만큼 클러스터를 사용
 - 음수
 - MFT Entry 크기가 클러스터 크기보다 작을 경우
 - MFT Entry 크기는 “ $2^{|MFT\ Entry\ Size|}$ ” 만큼의 Byte가 된다.
 - ex) 0xF6
 - $0xF6 \rightarrow -10 \rightarrow 2^{|-10|} \rightarrow 1,024\text{byte}$

Boot Record Analysis

- 0x44 - Index Record Size(1Byte) – 0x01
 - 인덱스 레코드의 크기
 - MFT Entry Size와 동일한 방식
- 0x48 - Serial Number(8Byte) – 가변
 - Volume Unique ID Number

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000000000	EB	52	90	4E	54	46	53	20	20	20	20	00	02	08	00	00
0000000010	00	00	00	00	00	F8	00	00	3F	00	FF	00	3F	00	00	00
0000000020	00	00	00	00	80	00	80	00	4E	B1	BF	00	00	00	00	00
0000000030	00	00	04	00	00	00	00	00	14	FB	0B	00	00	00	00	00
0000000040	F6	00	00	00	01	00	00	00	43	2C	04	C8	67	04	C8	72
0000000050	00	00	00	00	FA	33	C0	8E	D0	BC	00	7C	FB	B8	C0	07

NTFS Master File Table

NTFS의 MFT는 볼륨에 존재하는 모든 파일/디렉터리에 대한 정보를 담는 어떻게 보면 파일시스템의 가장 중요한 부분이라고 할 수 있다. 이번 장에서는 이 MFT에 대해 알아보자.

MFT

- 볼륨에 존재하는 모든 파일/디렉터리에 대한 정보를 담음
 - 파일/디렉터리의 이름, 생성시기, 크기, 위치, 소유자 등
- NTFS의 크기는 가변적
 - 초기 크기는 작지만 MFT의 크기가 커지면 줄어들지 않는다.
 - 즉, 많은 파일을 삭제하면 비어있는 MFT Entry가 많아짐

MFT

- MFT Entry

- MFT의 구성요소
- MFT Entry Header(42byte) + Attribute(나머지)
- 각 MFT Entry는 하나의 파일/디렉터리에 대한 정보를 가짐
- MFT Entry크기가 작으면, 여러 개에 나눠서 담는다.
 - 해당 MFT Entry들이 붙어 있을 필요는 없음.
- 0~15번까지의 MFT Entry는 Meta Data File이라고 불리며 파일시스템 관리파일로 예약
 - 구별을 위해 앞에 '\$'를 붙이고 첫 글자를 제외하고 전부 소문자(\$MFT제외)

NTFS MFT – Meta Data File

Entry Num	File Name	Description
0	\$MFT	Master File Table
1	\$MFTMirr	Master File Table mirror (4개의 record를 복제)
2	\$LogFile	Transaction Journal log
3	\$Volume	Volume Label, Version..
4	\$AttrDef	attribute names, numbers, and descriptions
5	.	The root folder
6	\$Bitmap	Showing free and unused clusters
7	\$Boot	Boot Record info
8	\$BadClus	Bad Cluster info
9	\$Secure	Security, Access info
10	\$Upcase	Lowercase character → Unicode uppercase character
11	\$Extend	NTFS extension file
12~15	Unused	Reserved Future use

MFT Entry Header

- 0x00 – Signature(4Byte) – ‘FILE’
 - MFT Entry에 문제발생 시 ‘BAAD’라는 값을 가진다.
- 0x02 – Fixup Array offset(2Byte) – 0x48
 - Fixup Array의 위치 아래 그림에서는 0x0030을 가리킴
- 0x04 – Fixup values count(2Byte) – 0x03
 - Fixup Array에 저장되어 있는 항목의 개수 – 0x04

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0400000000	46	49	4C	45	30	00	03	00	C8	59	B1	11	00	00	00	00
0400000010	01	00	01	00	38	00	01	00	98	01	00	00	00	04	00	00
0400000020	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00	00

MFT Entry Header - Fixup Array

- 해당 섹터의 데이터에 문제가 없는지 검사
- MFT Entry Header뒤쪽에 위치
- 각 End of Sector의 맨 뒤에 Signature(2byte)를 입력하고, Signature값 뒤에 원래 섹터 값을 저장
- 만약 MFT Entry Data Size 가 3 Sector라면
 - Count of Fixup Values 값은 4(Signature Sector까지 포함)
- Fixup Array를 사용하는 데이터
 - MFT Entries, INDEX Records, RCRD Records, RSTR Records

MFT Entry Header

- 0x08 - \$LogFile Sequence Number(8Byte) - 가변
 - \$LogFile에 기록된 해당 데이터의 수행관련 마지막 트랜잭션 위치
- 0x10 - Sequence Value(2Byte) - 가변
 - MFT Entry가 할당/할당해제 될 때마다 하나씩 증가
- 0x12 - Hard Link Count(2Byte) - 0x01
 - MFT에 연결된 하드링크 수, Base MFT Entry에서만 사용

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0400000000	46	49	4C	45	30	00	03	00	C8	59	B1	11	00	00	00	00
0400000010	01	00	01	00	38	00	01	00	98	01	00	00	00	04	00	00
0400000020	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00	00

MFT Entry Header

- 0x14 – First Attribute offset(2Byte) – 0x38
- 0x16 – Flags(2Byte) – 가변
 - MFT Entry의 상태, 속성 정보를 가짐.

0x01	해당 MFT Entry 사용 중
0x02	해당 MFT Entry는 디렉터리

- 0x18 – MFT Entry 사용 크기(4Byte) – 가변
- 0x1C – MFT Entry 할당 크기(4Byte) – 0x0400

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0400000000	46	49	4C	45	30	00	03	00	C8	59	B1	11	00	00	00	00
0400000010	01	00	01	00	38	00	01	00	98	01	00	00	00	04	00	00
0400000020	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00	00

MFT Entry Header

- 0x20 – Base MFT Entry 위치(8Byte) – 가변
 - 해당 MFT Entry가 Non-base MFT Entry의 경우 자신의 Base MFT Entry의 위치 값을 가진다.
 - 이 값은 File Reference Address의 형태를 가지게 된다.
- 0x28 – MFT Entry Attribute Type ID(2Byte) – 가변
 - MFT Entry내의 속성들이 가지는 고유 ID

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0400000000	46	49	4C	45	30	00	03	00	C8	59	B1	11	00	00	00	00
0400000010	01	00	01	00	38	00	01	00	98	01	00	00	00	04	00	00
0400000020	00	00	00	00	00	00	00	00	06	00	00	00	00	00	00	00

File Reference Address

- MFT Entry는 48bit Address를 가진다.
 - MFT Entry Address 또는 File Record Number라고도 불린다.
 - NTFS는 MFT Entry Address값을 그대로 사용하지 않음
 - → File Reference Address
- File Reference Address (64bit)
 - MFT Entry Header->Sequence Value + MFT Entry Address
 - ex) MFT Entry = 1024, Sequence Value = 0x20
 - → MFT Entry Address = 0x0000000000400
 - → File Reference Address = 0x00200000000000400

Sequence Value(16bit)

MFT Entry Address(48bit)

File Reference Address

- MFT Entry의 내용이 변했는지 판단하기 위함.
 - Sequence Value는 MFT Entry의 할당 상황이 바뀔 때마다 값이 증가
 - 즉, File Reference Addr의 상위 16비트 값이 바뀐다면, 해당 Entry에 자신이 찾던 파일을 담지 않음을 의미

Base/Non-base MFT Entry

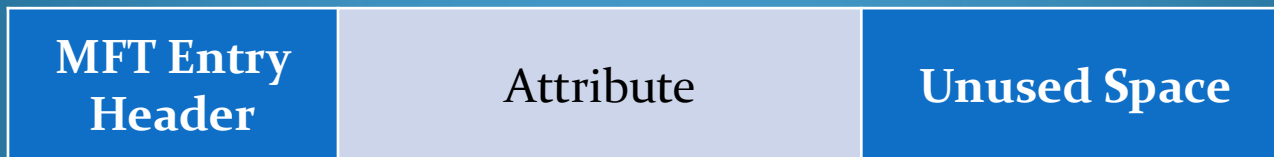
- 파일에 대한 정보가 많아 하나의 MFT에 담기 불가능할 경우 여러 개의 MFT Entry를 사용
- Base MFT Entry는 Non-base MFT Entry들을 가리킴
- Non-base MFT Entry들은 “File Reference to base MFT Entry”(0x20)라는 항목(MFT Header)을 통해 Base MFT Entry위치 값을 가진다.

Attribute

MFT Entry의 모든 정보는 속성으로 관리된다. 이번 장에서는 이 속성이란 것에 대해 한번 알아보도록 하자.

Attribute란?

- $\text{MFT Entry} = \text{MFT Entry Header} + \text{Attribute}$
- MFT Entry가 담고있는 파일 특성에 따라 속성이 바뀜
- 파일 생성시간, 파일 이름, 파일 내용 etc..



속성의 구성

- Attribute = Attribute Header + Attribute Content
 - 동일한 헤더구조 + 속성마다 다른 속성내용 형태
 - MFT Entry는 몇 개의 속성을 가지는지에 대한 정보를 가짐
- MFT Entry의 마지막 속성 뒤에는 End Marker를 넣어 준다.
 - End Marker = 0xFFFFFFFF

0xFFFFFFFF



MFT Entry Header	Attribute Header	Attribute Content	Attribute Header	Attribute Content	End Marker	Unused Space
------------------------	---------------------	----------------------	---------------------	----------------------	---------------	-----------------

속성의 종류

속성 타입 번호	속성 이름	설명
16	\$STANDARD_INFORMATION	최근 접근 시간, 생성 시간, 소유자, 보안 아이디 등.
32	\$ATTRIBUTE_LIST	속성을 찾기 위한 리스트
48	\$FILE_NAME	파일 이름(UNICODE)
64	\$VOLUME_VERSION	볼륨정보(Win NT)
64	\$OBJECT_ID	16byte로 이루어진 파일이나 디렉터리를 위한 고유 값(Win 2000이상)
80	\$SECURITY_DESCRIPTOR	File Access Control, 보안 속성
96	\$VOLUME_NAME	Volume name 관련 정보
112	\$VOLUME_INFORMATION	파일시스템의 버전과 여러 Flag를 가짐
128	\$DATA	파일의 내용

속성의 종류(계속)

속성 타입 번호	속성 이름	설명
144	\$INDEX_ROOT	인덱스 트리의 루트노드
160	\$INDEX_ALLOCATION	인덱스 트리와 연결된 노드
176	\$BITMAP	할당 정보를 관리
192	\$SYMBOLIC_LINK	Soft Link(NTFS 1.2)
192	\$REPARSE_POINT	Soft Link에서 사용하는 reparse 위치 정보를 가짐(NTFS 3.0↑)
208	\$EA_INFORMATION	OS/2 App과의 호환성을 위함
224	\$EA	OS/2 App과의 호환성을 위함
256	\$LOGGED_UTILITY_STREAM	암호화된 속성정보와 Key값을 가짐(NTFS 3.0↑)

속성의 종류

- MFT Entry와 Attribute Name은 MetaData 형식
 - Attribute의 경우엔 전부 대문자.
 - MFT Entry의 경우엔 첫 글자만 대문자를 가짐
 - 단, MFT Entry 0번인 \$MFT 파일은 전부 대문자로 기술
 - 속성 타입 번호가 낮은 순서대로 저장.
 - \$STANDARD_INFORMATION(ID : 16)가 제일 앞에 위치

\$STANDARD_INFORMATION(Type ID : 16)

MFT Entry Header	Attribute Header	Attribute Content	Attribute Header	Attribute Content	Unused Space
------------------	------------------	-------------------	------------------	-------------------	--------------

\$FILE_NAME(Type ID : 48)

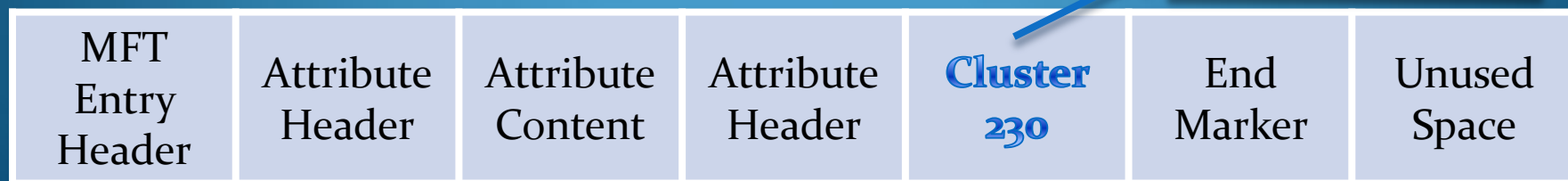
속성 저장 방식

- Resident

- 속성헤더 뒤쪽에 바로 속성 내용이 오는 형식
- MFT Entry내에 완전한 속성의 내용이 존재

- Non-Resident

- 속성 내용이 너무 커서 MFT Entry안에 넣지 못하는 경우
- ex) \$DATA
- 속성 내용을 따로 클러스터를 할당하여 저장
- Attribute Content는 어떤 클러스터에 할당되었는지만 저장
 - Cluster Run



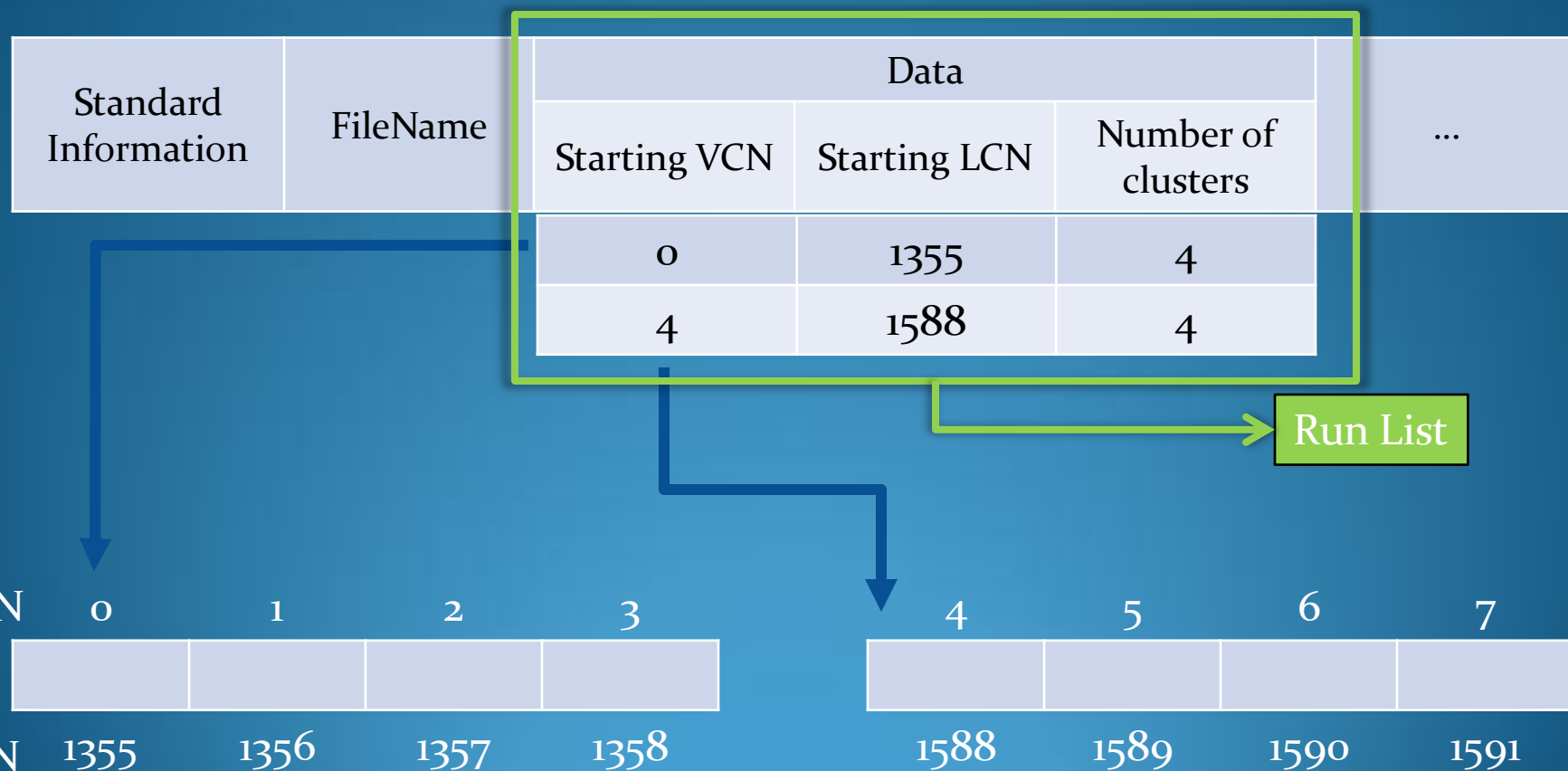
속성 저장 방식(Non-Resident)

- Cluster Run
 - Data Structure
 - 속성 내용이 어떤 클러스터에 할당되어 있는지 기록
 - 시작 클러스터와 길이를 기록
 - Run Data단위로 Start Cluster와 Length를 저장한다.
 - 데이터 분산 시 Run Data 크기가 커진다.

속성 저장 방식(Non-Resident)

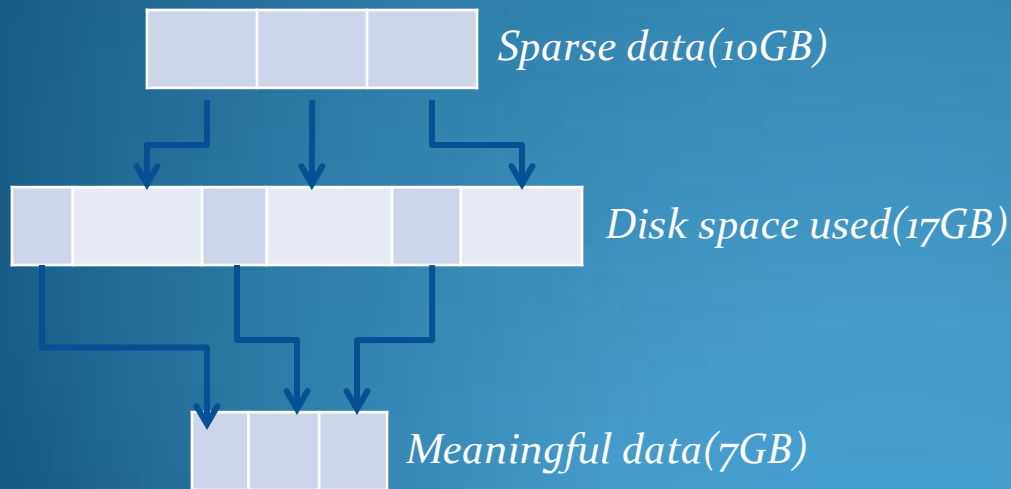
- LCN – Logical Cluster Number
 - Volume의 첫 번째 클러스터부터 순차적으로 지정되어 있는 주소
 - 중복주소는 존재하지 않음
- VCN – Virtual Cluster Number
 - 파일의 첫 번째 클러스터부터 순차적으로 지정
 - 파일의 첫 번째 클러스터 $VCN = 0$
 - 동일 파일에서만 중복되지 않음

속성 저장 방식(Non-Resident)

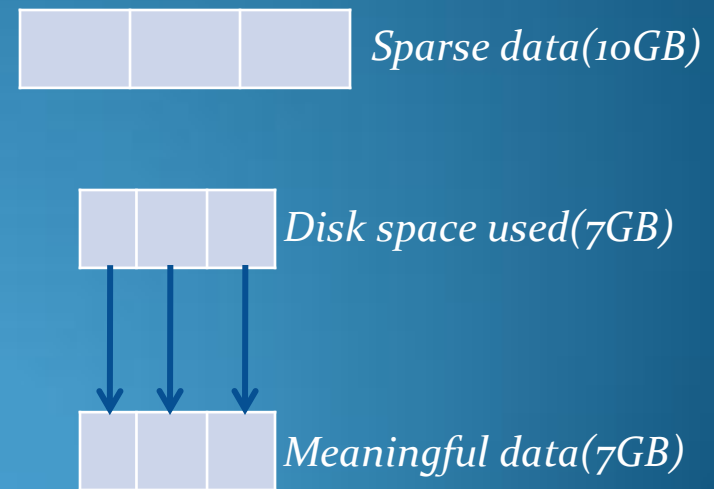


Sparse Data Storage

- \$DATA 속성에만 적용되는 형식
- 0으로 채워진 연속적인 클러스터에 대해 0으로 채워져 있다는 사실만을 기록, 실제로 저장하진 않음
- ex) DB 저장 방식

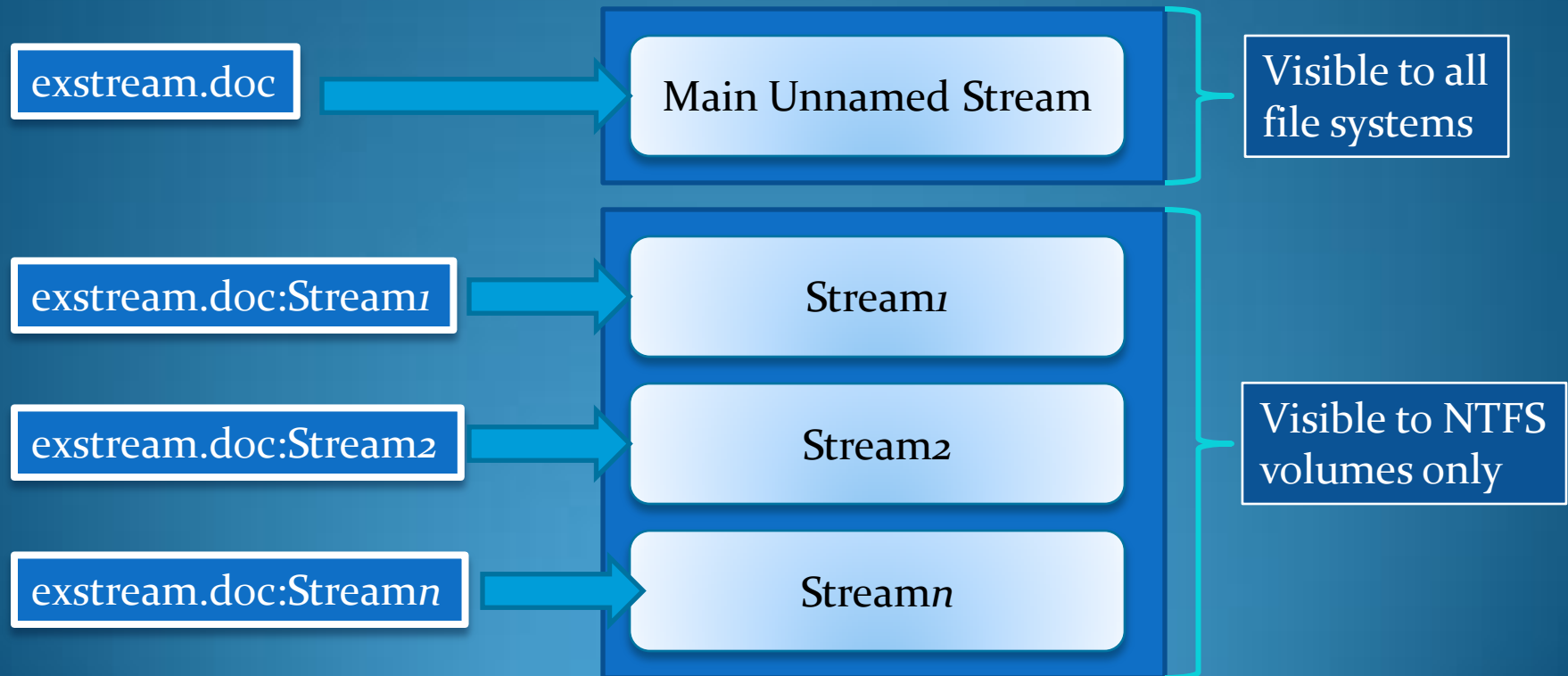


Without Sparse File Attribute Set



With Sparse File Attribute Set

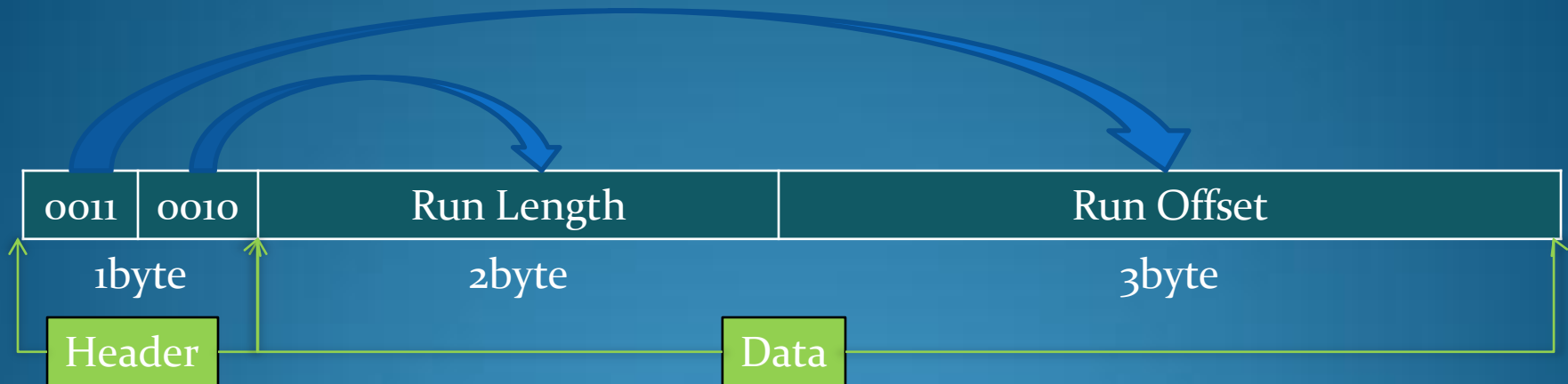
Unnamed and Named Streams



Run List

- Cluster Run은 처음 1byte가 각각 상위 4비트는 Run Offset항목의 길이 값, 하위 4비트는 Run Length항목의 길이 값을 가진다.
- 처음 Run Offset값을 제외하곤 전부 이전 Cluster Run의 값에 상대적이다.
 - Run Offset에는 음수가 들어갈 수 있다.
- Sparse형태의 Run List 가능.

Run List – Cluster Run



ex) 11 30 60 21 10 00 01 11 20 Eo 00

Ans)

Header : 1 1 , Run Length : 30 Run Offset : 60

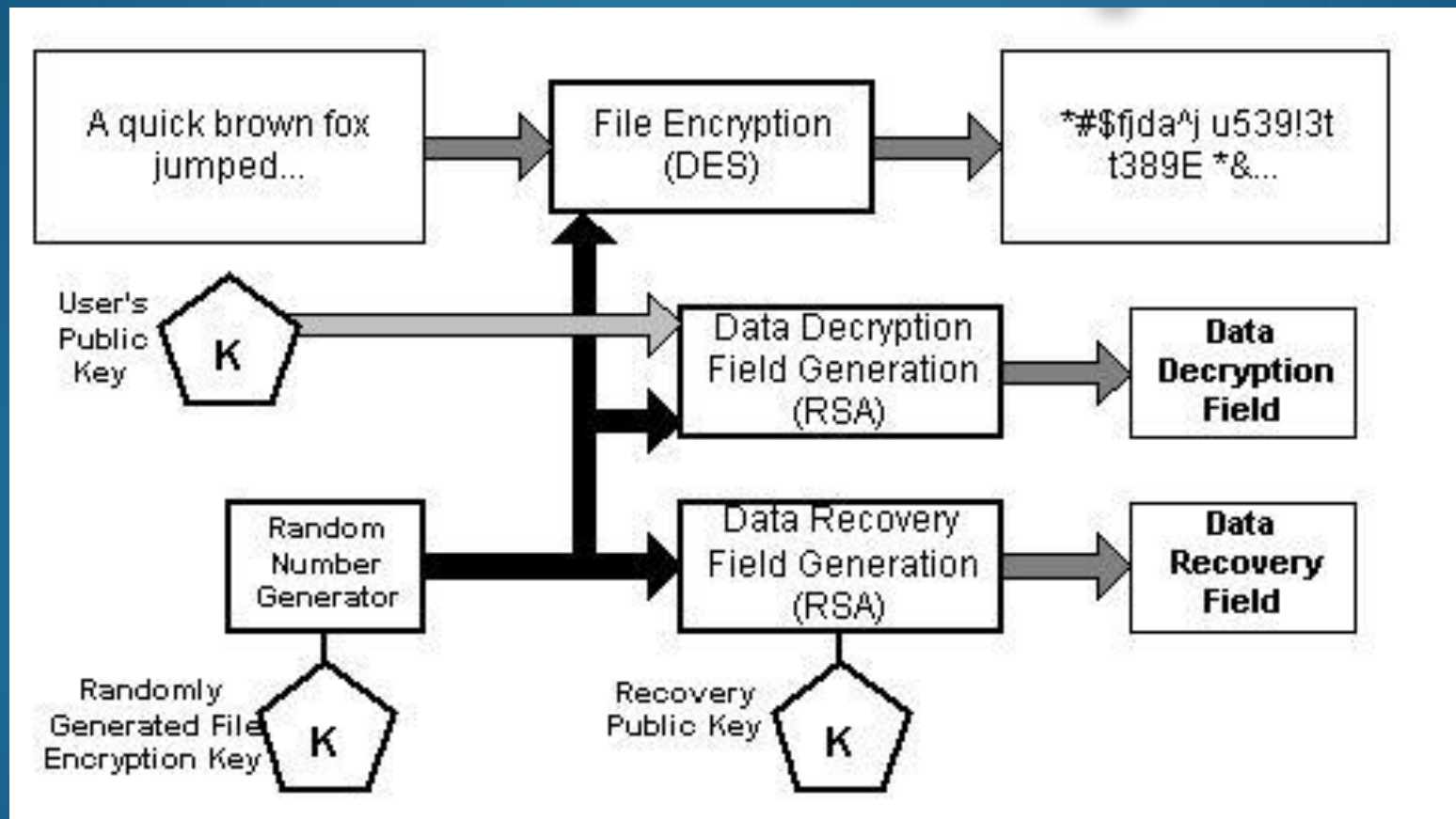
Header : 2 1 , Run Length : 10 Run Offset : 00 01

Header : 1 1 , Run Length : 20 Run Offset : Eo

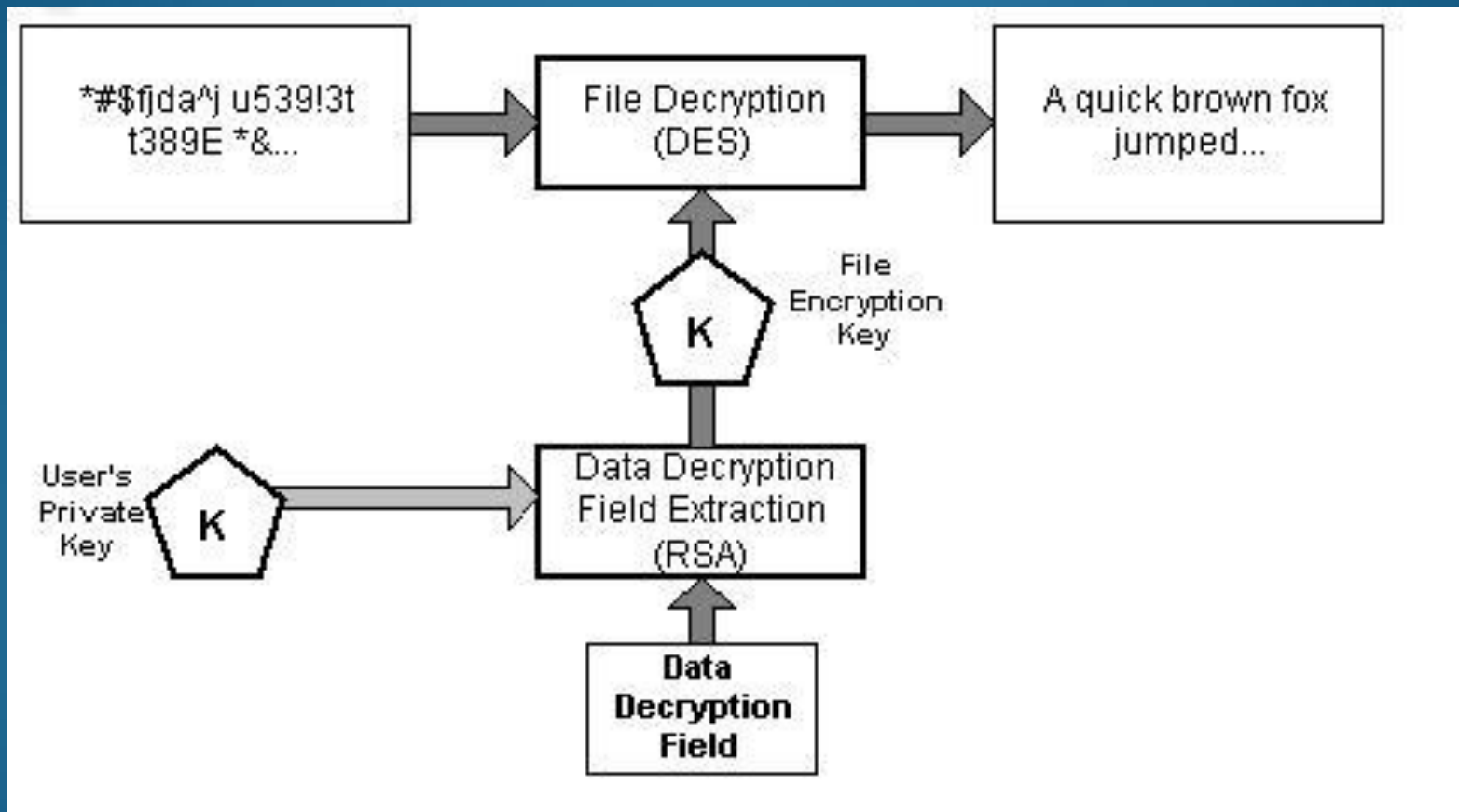
Encryption File System(EFS)

- EFS – Encrypted File System
 - DESX를 이용한 파일내용을 암호화
 - 암호화한 파일 내용을 \$DATA속성에 저장
 - 암호화에 사용된 Random Key는 사용자공개키로 암호화
 - 암호화된 FEK는 \$LOGGED_UTILITY_STREAM속성에 저장
 - 암호화 작업 도중 EFSO.TMP파일을 생성
 - 파일의 원형 데이터를 가짐
 - 그로 인해 암호화가 되더라도, 해당 파일 복구에 성공 시엔 쓸모가 없어진다.

Encryption process

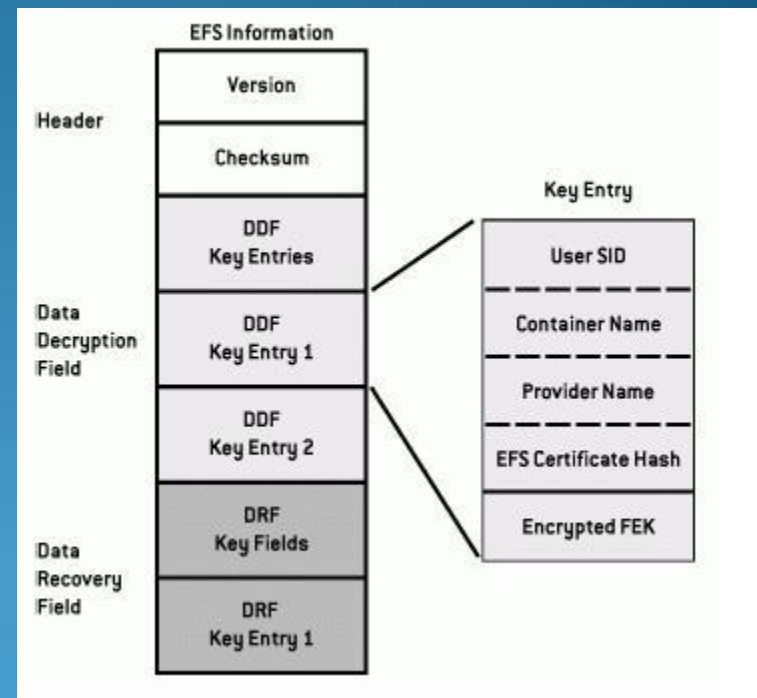


Decryption process



Encryption File System(EFS)

- DDF는 파일을 해독할 수 있는 유저마다 Entry가 생성
 - SID, 암호화 정보, FEK
- DEF는 데이터를 복구하는 각각의 방법을 위한 생성
 - 인증된 사용자가 데이터를 접근하는데 사용하는 FEK를 담고 있음.



Data Integrity and Recoverability with NTFS

NTFS는 데이터 무결성 보장 및 복구가 가능하도록 되어있다. 이번 장에서는 NTFS가 어떤식으로 이런 특징을 가지고 있는지 알아보도록 하자.

Data Recoverability

- NTFS는 일반적인 트랜잭션 로깅 과 복구 기술을 사용하여 볼륨의 일관성을 보장해 주는 파일복구 시스템을 가지고 있다.
- 디스크 실패 이벤트 발생 시
 - 로그 파일에 저장된 정보를 접근, 프로시저를 동작하여 복구
 - NTFS복구 프로시저는 일관적인 상태로 볼륨을 복구 시키는 정확성을 지니고 있다.
 - 트랜잭션 로깅은 작은 오버헤드를 가지고 있다.

Data Recoverability

- NTFS는 컴퓨터의 재부팅 실패 후, NTFS볼륨의 프로그램에 접근 시 “HDD recovery operations”를 이용하여 자동적으로 볼륨의 무결성을 지킨다.
- NTFS는 또한 볼륨의 배드섹터를 최소화 하기 위해 “cluster remapping”이라는 기술을 사용
- MBR이나 부트섹터에서 문제발생시엔 볼륨의 데이터에 액세스할 수 없음.

Recovering Data with NTFS

- NTFS는 시스템파일의 수정과 같은 I/O 명령을 관찰
- 트랜잭션은 완료를 제외한 disk failure 이벤트와 같은 문제가 발생 시엔 rolled back을 수행
- NTFS는 트랜잭션 로깅과 복구를 통해 볼륨의 오류가 없음을 보장한다.
- 모든 시스템파일은 system failure 후에 접근상태로 남게 된다. 그러나 유저데이터는 system failure 나 배드섹터와 같은 이유로 데이터가 손실될 수 있다.

Cluster Remapping

- bad sector error 발생 → cluster remapping
 - 배트섹터를 포함한 클러스터를 리매핑하고, 데이터에 새로운 클러스터를 할당
- 읽는 도중에 에러가 발생 시
 - NTFS는 에러를 발생시킨 프로그램을 리턴
 - 해당 데이터 손실
- 쓰는 도중에 에러가 발생 시
 - NTFS는 새로운 클러스터에 데이터를 작성
 - 해당 데이터는 손실되지 않음.

Cluster Remapping

- NTFS는 배드섹터를 포함한 클러스터의 주소를 이용하여, 해당 배드섹터를 재사용할 수 없도록 조치
- Cluster remapping은 백업을 수행하지 않는다.
- 해당 에러 타입은 Event Log를 통해 확인

NTFS Index

NTFS는 FAT File System보다 훨씬 효율적인 인덱싱 능력을 가지고 있다. 이번 장에서는 NTFS의 Index방식에 대해 알아보자.

Index?

- 파일시스템의 빠른 검색을 위해 인덱스 구조로 관리
 - ex) Directory
- Binary Tree의 변형인 B-Tree(Block-Tree) 사용

인덱스 이름	인덱싱 하는 데이터	존재하는 위치
\$I30	\$FILE_NAME attr	각 디렉터리의 MFT Entry
\$SDH	Security Descriptors	\$Secure Meta Data File
\$SII	Security IDs	\$Secure Meta Data File
\$O	Object IDs	\$ObjId Meta Data File
\$O	Owner IDs	\$Quota Meta Data File
\$Q	Quotas	\$Quota Meta Data File

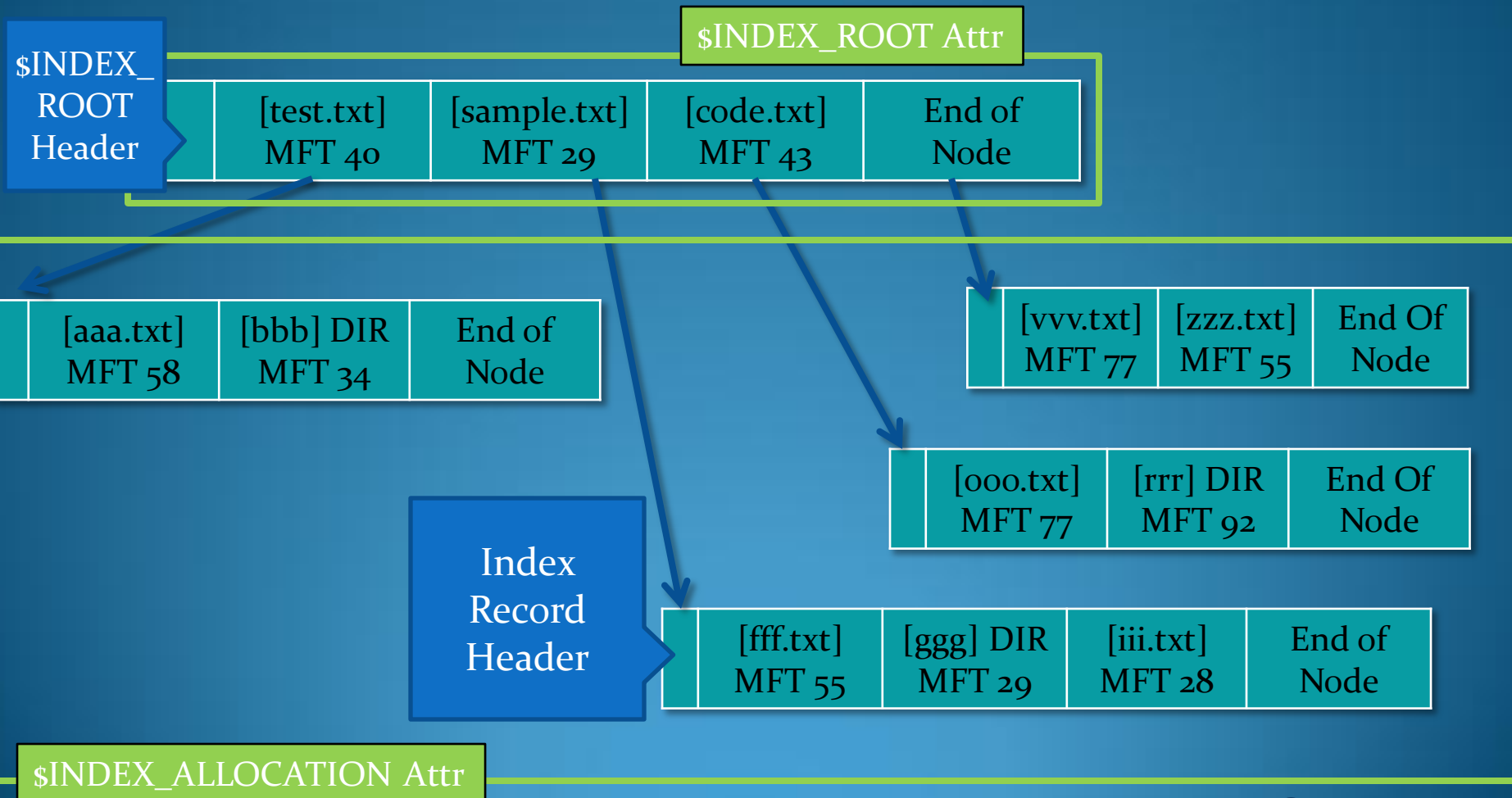
B-Tree

- 노드 하나에 여러 개의 Index Entry
- 항상 End Of Node를 가짐
- n 개의 Index Entry를 가진 Index Node는 $n+1$ 의 자식노드를 가짐
- 각각의 Index Entry는 MFT Entry 위치를 가짐.
 - → 해당 위치를 이용하여 자식 노드로 이동
- B-Tree는 MFT Record보다 디렉터리의 정보가 큰 경우에도 사용

Index Node

- Index Node Header + Index Entries
- Index Entry는 가변적
- 마지막 Entry는 End Of Node를 알림
 - 최소크기 : Index Node Header + End Of Node
- Index Node는 독립적으로 존재하지 않음.
 - \$INDEX_ROOT | \$INDEX_ALLOCATION attr에 속함
 - Index Node Header앞에 attr Header를 붙임
 - 일반적으로 Root Index Node가 \$INDEX_ROOT를 가짐

NTFS B-Tree





Question?

Reference

- IT Expert, 임베디드 개발자를 위한 파일시스템의 원리와 실습 – 한빛미디어
- Windows Internals (fourth edition)
- www.ntfs.com
- NTFS Document
 - (<http://info.21.lv/FAT%20Information/ntfsdoc/index.html>)
- How NTFS Works : Local File Systems – Microsoft
 - (<http://technet.microsoft.com/en-us/library/cc781134.aspx>)
- Computer Forensics Presentation File - Santa Clara Univ.
 - (http://www.cse.scu.edu/~tschwarz/coen152_05/PptPre/NTFSFS.ppt)
- 기타 수많은 웹사이트의 정보