New Technology File System

JK Kim

@pr0neer

forensic-proof.com

proneer@gmail.com

Outline

1. NTFS

- ✓ Introduction
- ✓ Internals
- ✓ Features
- ✓ Example

NTFS Introduction

Security is a people problem…

NTFS Version

- 기존 FAT (File Allocation Table) 파일시스템은 개인용 운영체제(윈도우 3.1/95/98)를 위해 사용
- 윈도우 NT (New Technology)의 등장으로 서버용 운영체제에서 사용하기 위한 새로운 기능을 추가한 파일시스템 필요
- 윈도우 NT 이후 ~ 윈도우 7 까지 사용

버전	운영체제
1.0	Windows NT 3.1
1.1	Windows NT 3.5
1.2	Windows NT 3.51
3.0	Windows 2000
3.1 – 5.1	Windows XP
5.2	Windows Server 2003
6.0	Windows Vista, Windows Server 2008, Windows 7

NTFS Cluster Size

- 클러스터 크기가 4 KB가 넘을 경우 파일 압축을 지원하지 못함

볼륨 크기	Windows NT 3.51	Windows NT 4.0	Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7
7 MB – 512 MB	512 bytes	4 KB	4 KB
512 MB – 1 GB	1 KB	4 KB	4 KB
1 GB – 2 GB	2 KB	4 KB	4 KB
2 GB – 2 TB	4 KB	4 KB	4 KB
2 TB – 16 TB	Not Supported*	Not Supported*	4 KB
16 TB – 32 TB	Not Supported*	Not Supported*	8 KB
32 TB – 64 TB	Not Supported*	Not Supported*	16 KB
64 TB – 128 TB	Not Supported*	Not Supported*	32 KB
128 TB – 256 TB	Not Supported*	Not Supported*	64 KB
> 256 TB	Not Supported	Not Supported	Not Supported

* - MBR의 제한으로 지원하지 못하는 것을 의미한다.

http://support.microsoft.com/kb/140365

NTFS Size Limit

제한 항목	설 명	
최대 파일 크기	구조 상 최대 값	16 exabytes – 1KB (2^{64} bytes – 1KB)
	실제 구현된 최대값	16 terabytes – 64 KB (2^{44} bytes – 64 KB)
최대 볼륨 크기	구조 상 최대 값	2^{64} clusters - 1
	실제 구현된 최대 값	256 terabytes – 64 KB (2^{32} clusters – 1)
볼륨 당 파일 수	4,294,967,295 (2^{32} – 1)	

http://technet.microsoft.com/en-us/library/cc781134%28WS.10%29.aspx

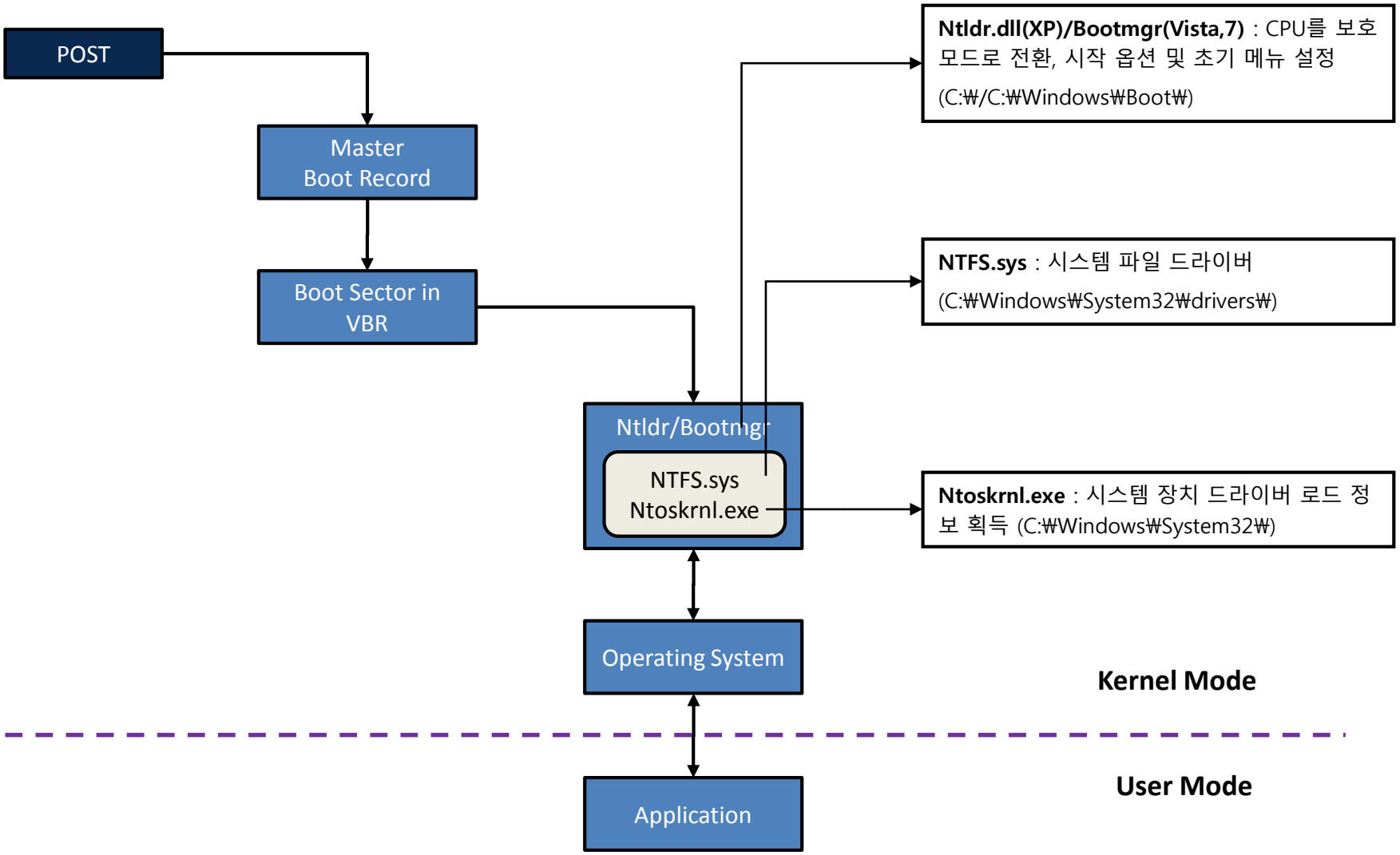
Features

특 징	설명
USN 저널	파일의 변경 내용을 기록 한 후 시스템 오류 발생으로 재부팅 될 경우 완료되지 않은 처리 작업을 롤백 (Rollback)
ADS (Alternate Data Stream)	다중 데이터 스트림 지원
Sparse 파일	파일 데이터가 대부분 0일 경우 실제 데이터 기록 없이 정보만 기록
파일 압축	LZ77 의 변형된 알고리즘을 사용하여 파일 데이터 압축 지원
VSS (Volume Shadow Copy) Service	윈도우 2003부터 지원된 기능으로 새롭게 덮어써진 파일 및 디렉터리의 백업본을 유지하여 복구 지원
EFS (Encrypting File System)	FEK (File Encryption Key)를 이용한 대칭키 방식의 파일 데이터 암호화 지원
Quotas	사용자 별 디스크 사용량 제한
유니코드 지원	다국어 지원 (파일, 디렉터리, 볼륨 이름 모두 유니코드 사용)
동적 배드 클러스터 할당	배드 섹터가 발생한 클러스터를 자동으로 재할당
대용량 지원	2 TB가 넘는 대용량 볼륨 지원

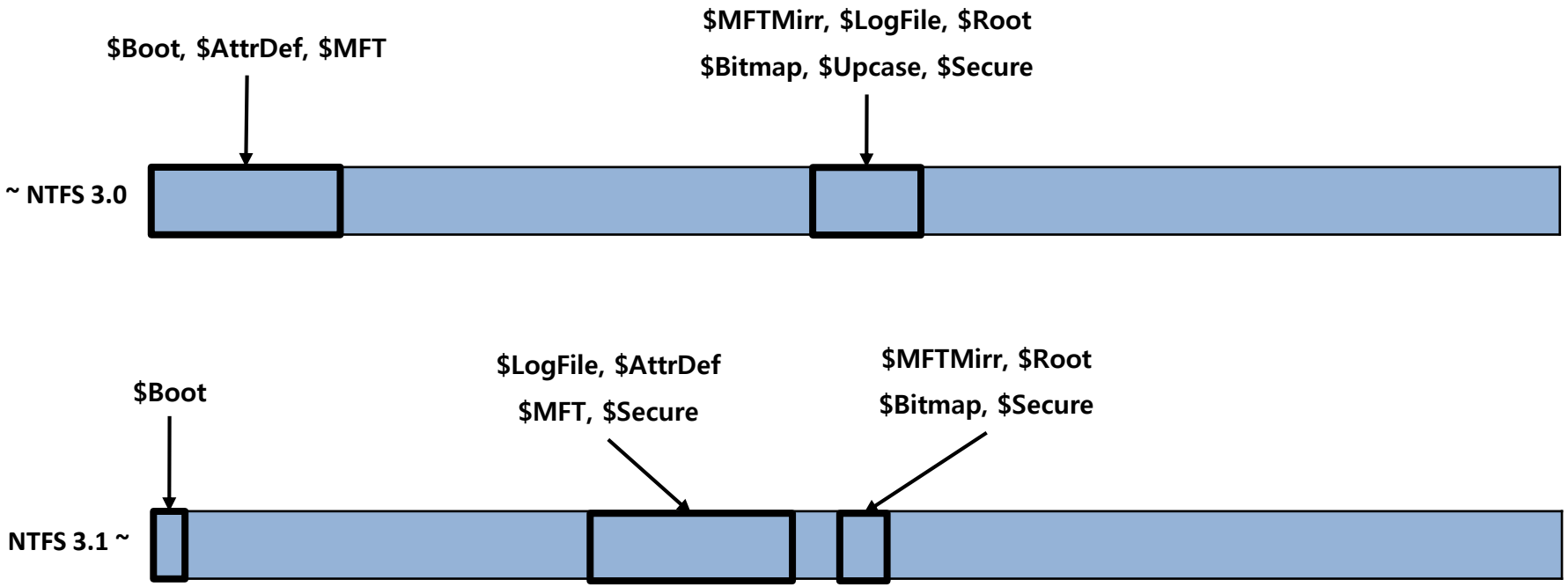
NTFS Internals

Security is a people problem…

NTFS Architecture



Layout


$Boot, $AttrDef, $MFT
$MFTMirr, $LogFile, $Root
$Bitmap, $Upcase, $Secure
~ NTFS 3.0
$Boot
$LogFile, $AttrDef
$MFT, $Secure
$MFTMirr, $Root
$Bitmap, $Secure
NTFS 3.1 ~

Structure

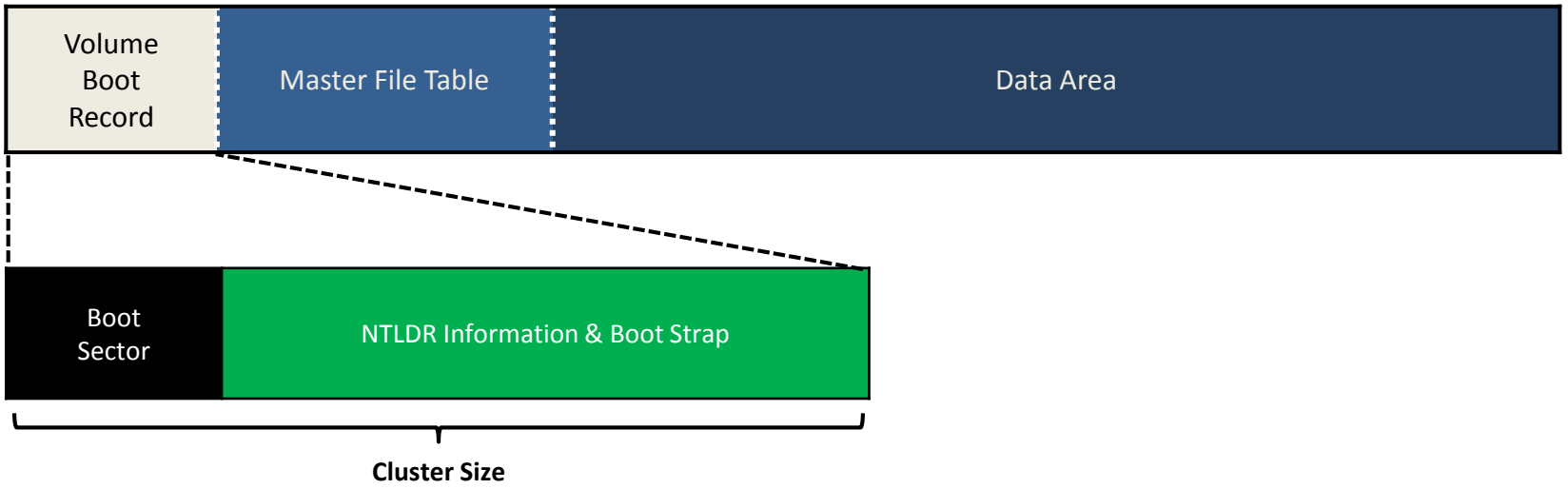
- 모든 데이터는 파일 형태로 관리 (관리 데이터, 디렉터리 등)
- VBR (Volume Boot Record)은 고정된 위치
- MFT는 일반적으로 VBR 이후에 존재하지만 데이터 영역 어느 곳에 와도 무관

VBR (Volume Boot Record)

- NTFS로 포맷된 파티션의 가장 앞부분에 위치하는 영역
- 부트 섹터, NTLDR 위치 정보, 추가적인 부트 코드 저장
- VBR의 크기는 클러스터 크기와 동일

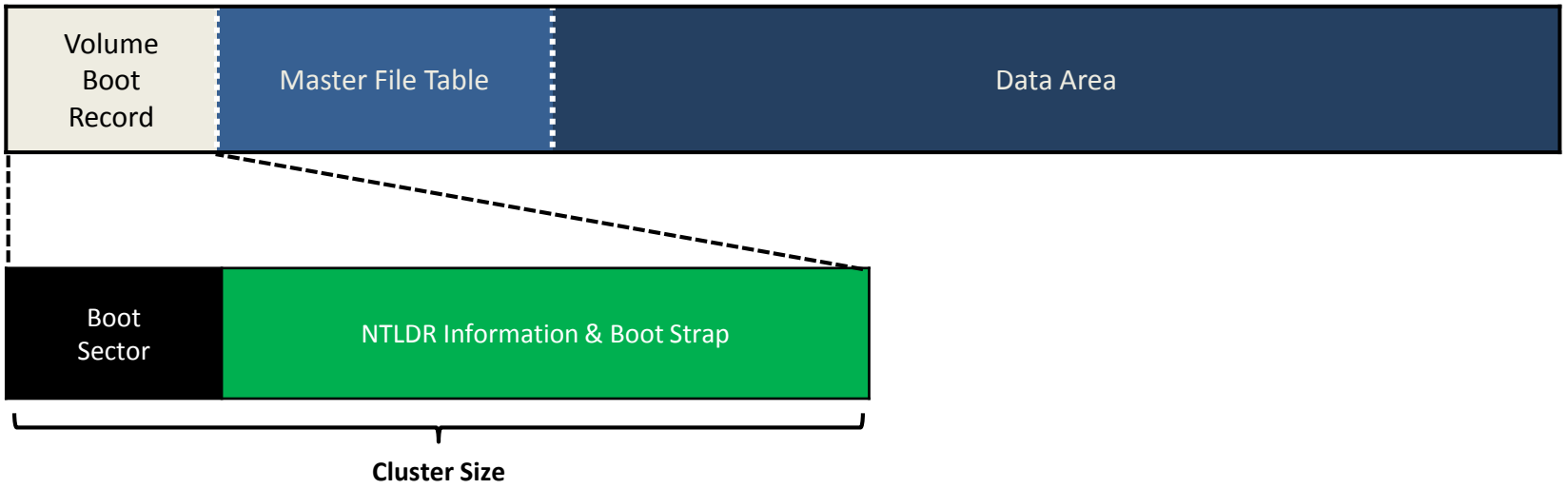
클러스터 크기	VBR 크기 (섹터)
512 bytes	1
1 KB	2
2 KB	4
4 KB	8

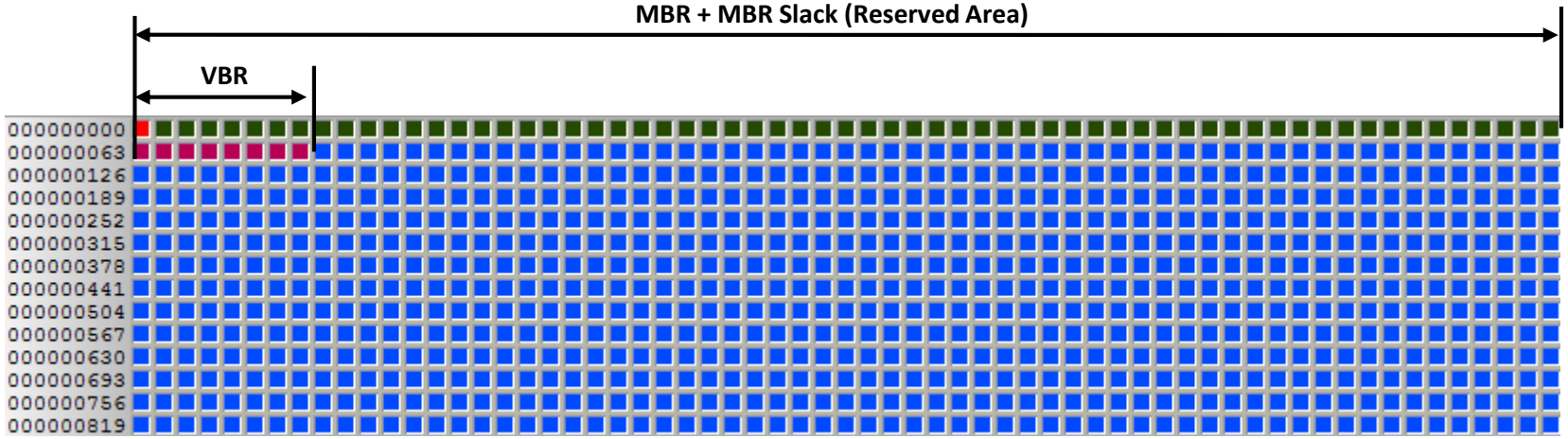
VBR (Volume Boot Record)



- 0 번 섹터 : 부트 섹터
- 나머지 섹터 : NTLDR 위치 및 추가적인 부트 코드 정보

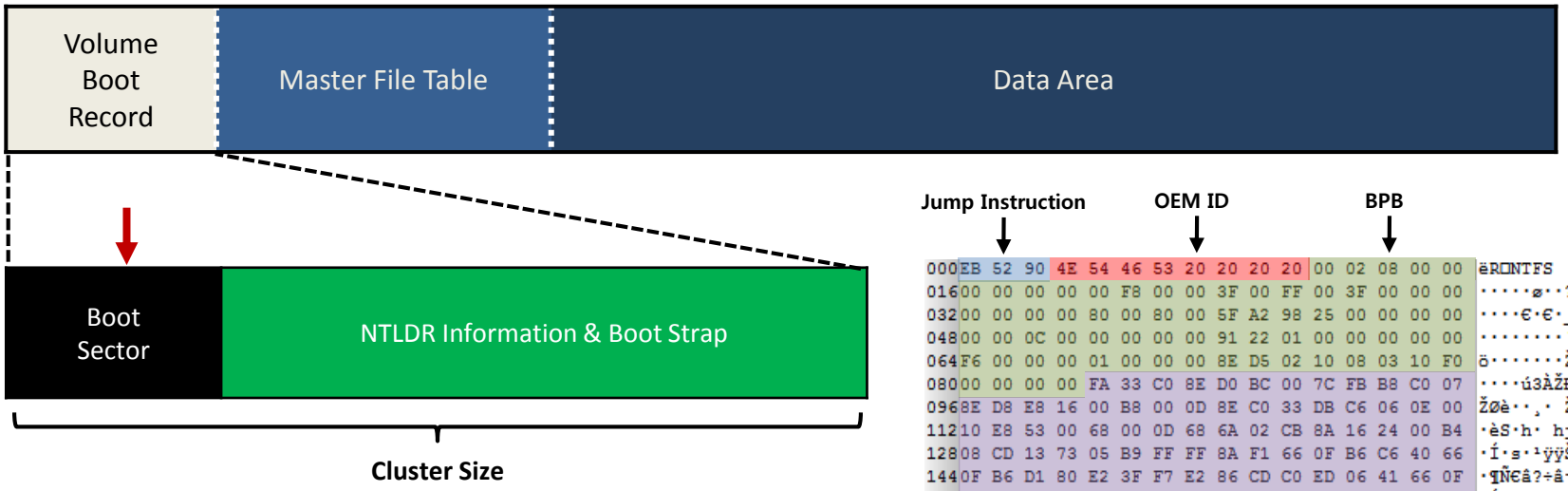
VBR (Volume Boot Record)



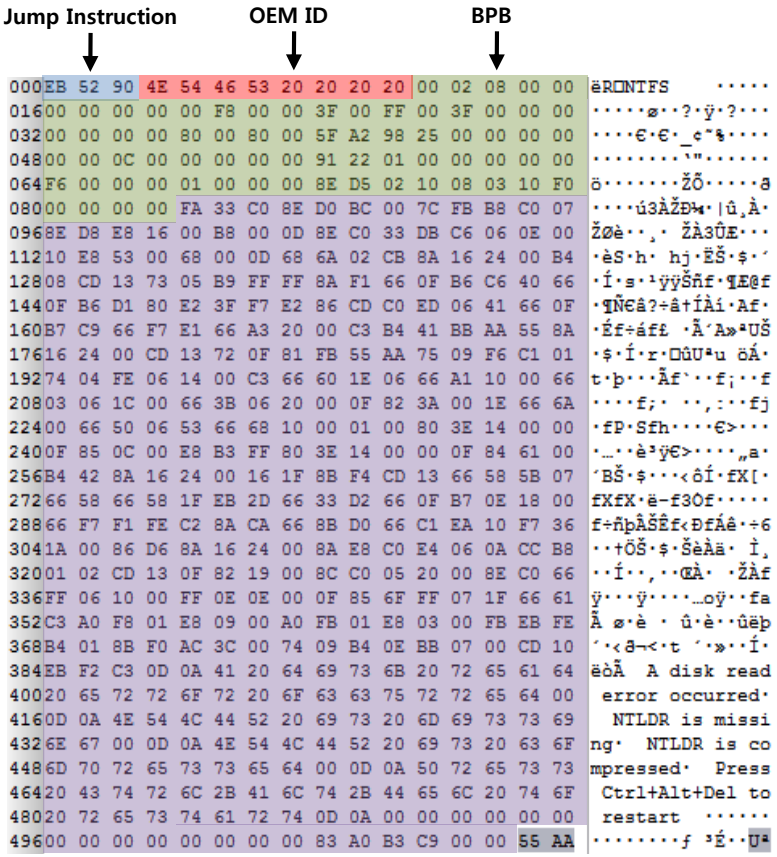


클러스터 크기가 4 K 인 경우

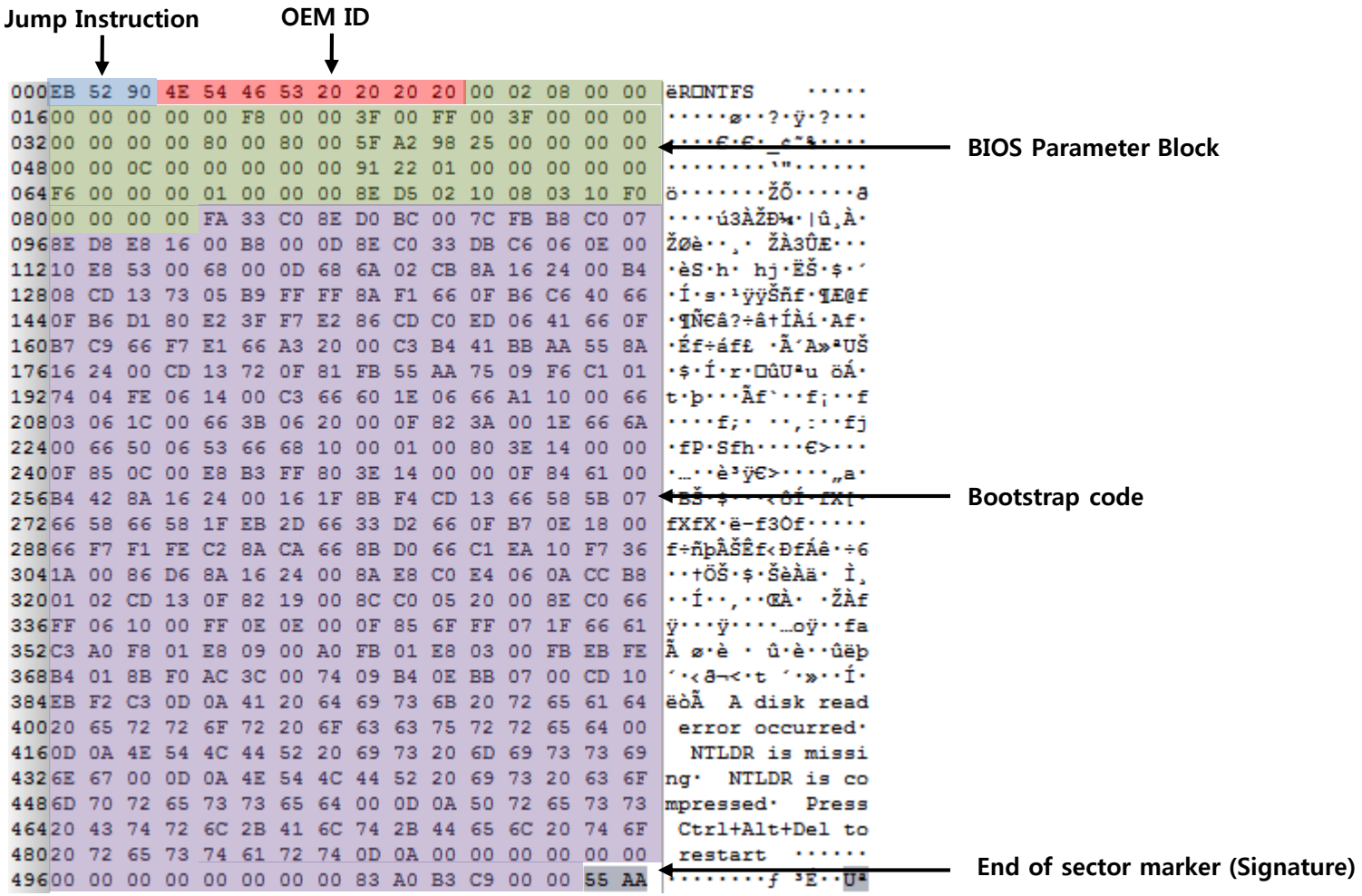
VBR (Volume Boot Record)



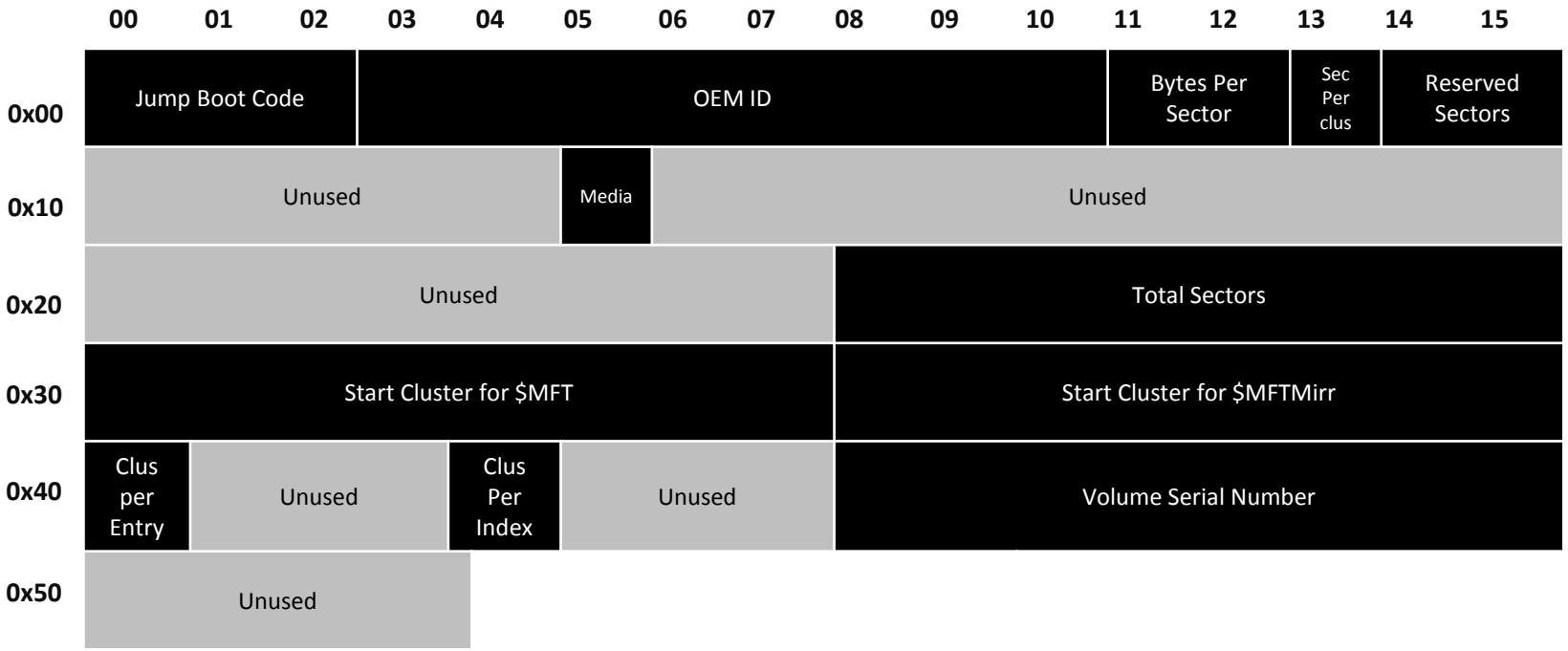
범위(10진수)	길이	설명
0 – 2	3 bytes	Jump Instruction
3 – 10	8 bytes	OEM ID
11 – 35	25 bytes	BPB
36 – 83	48 bytes	Extended BPB
84 – 509	426 bytes	Bootstrap code
510 – 511	2 bytes	End of sector marker (Signature)



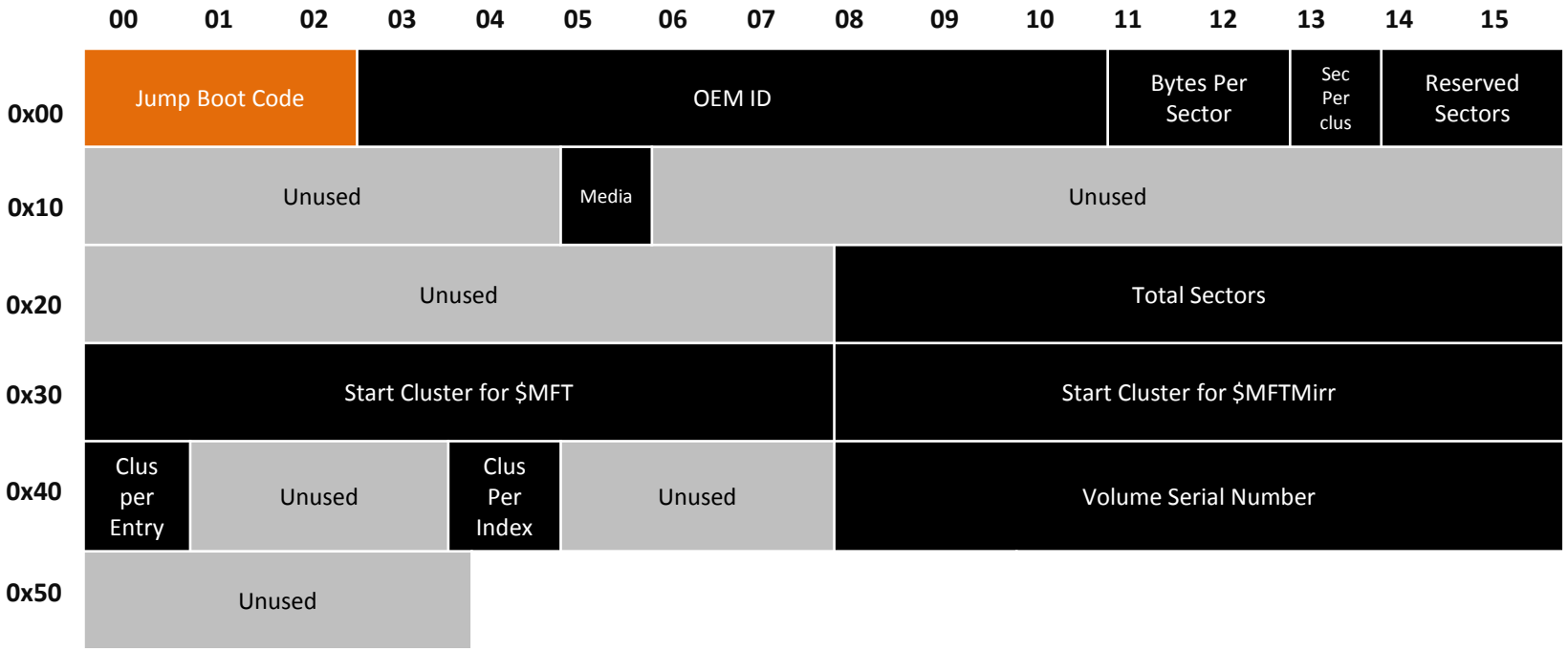
VBR (Volume Boot Record)



VBR (Volume Boot Record) ➔ BIOS Parameter Block

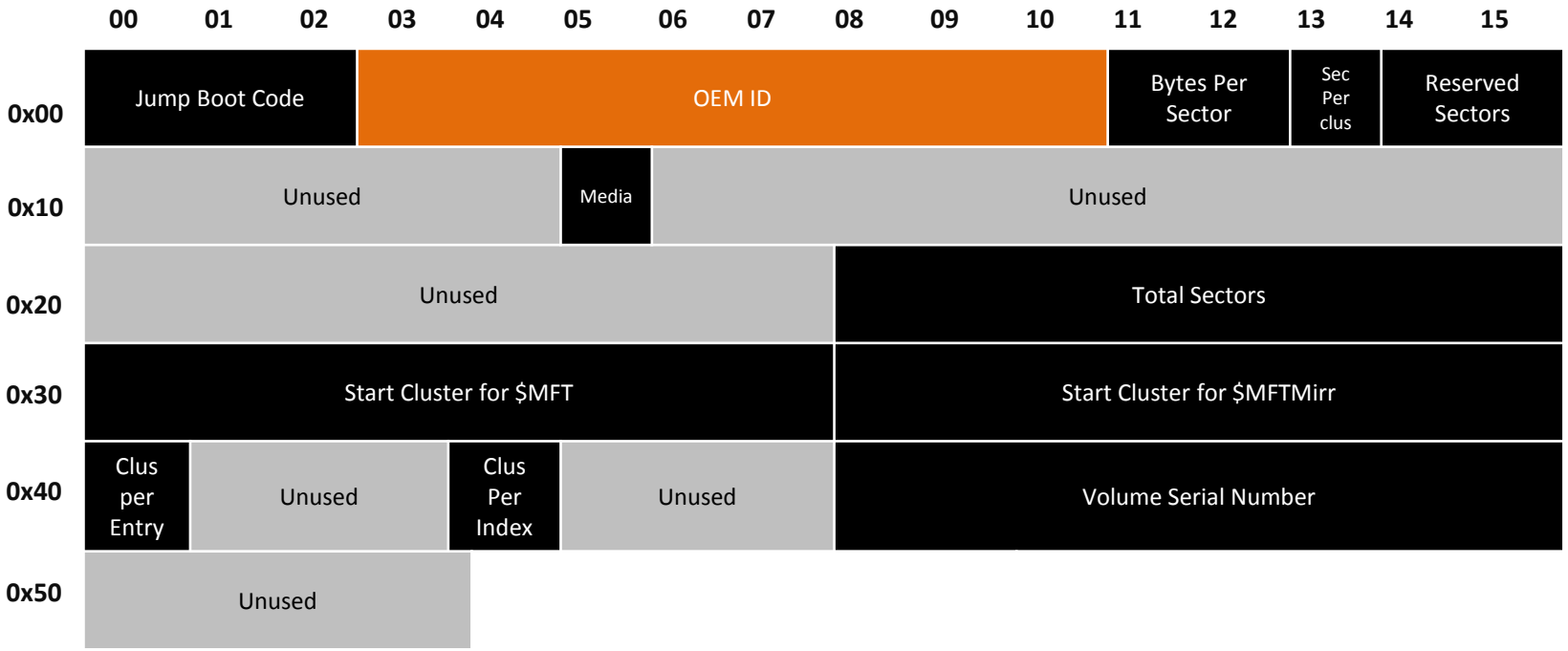


VBR (Volume Boot Record) ➔ BIOS Parameter Block



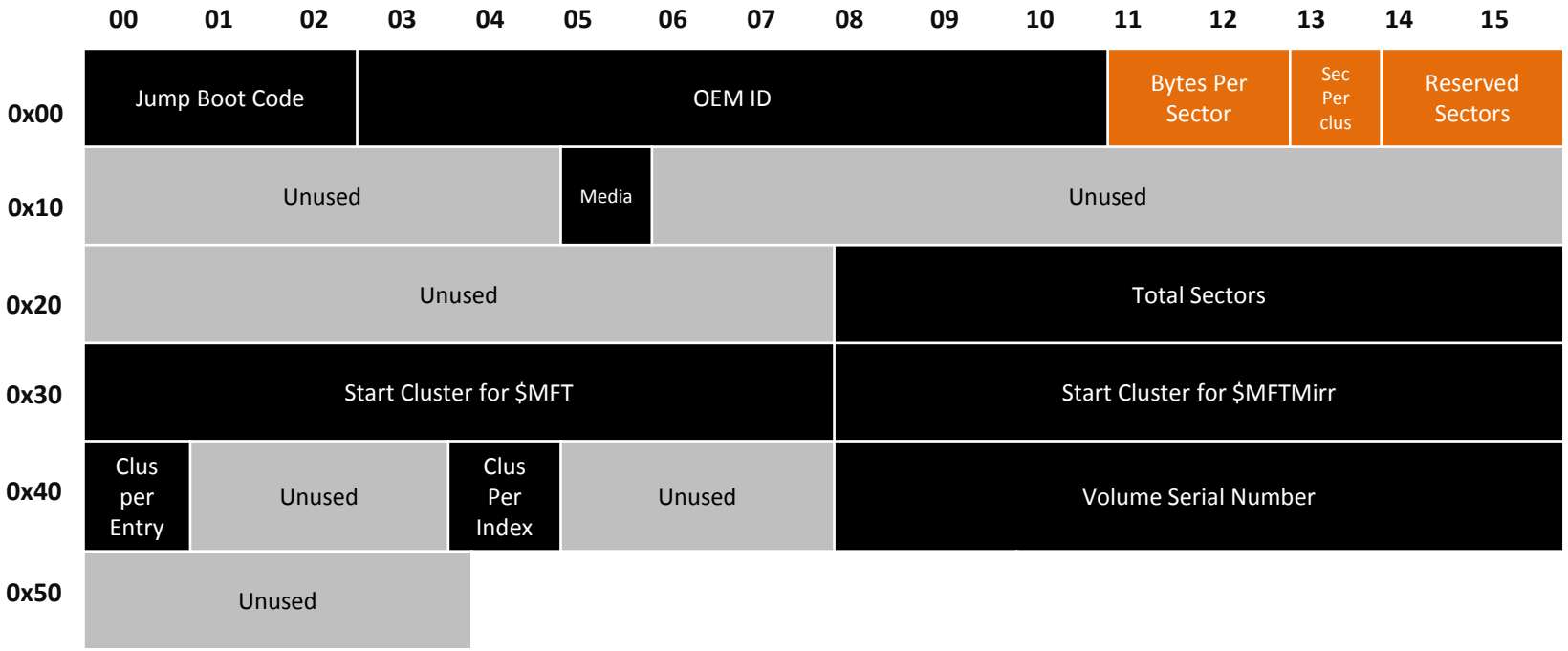
- **Jump Boot Code** : 부트 코드로 점프하기 위한 명령어 (0xEB5290)
  - EB58 : jmp 00000054
  - 90 : nop

VBR (Volume Boot Record) ➔ BIOS Parameter Block



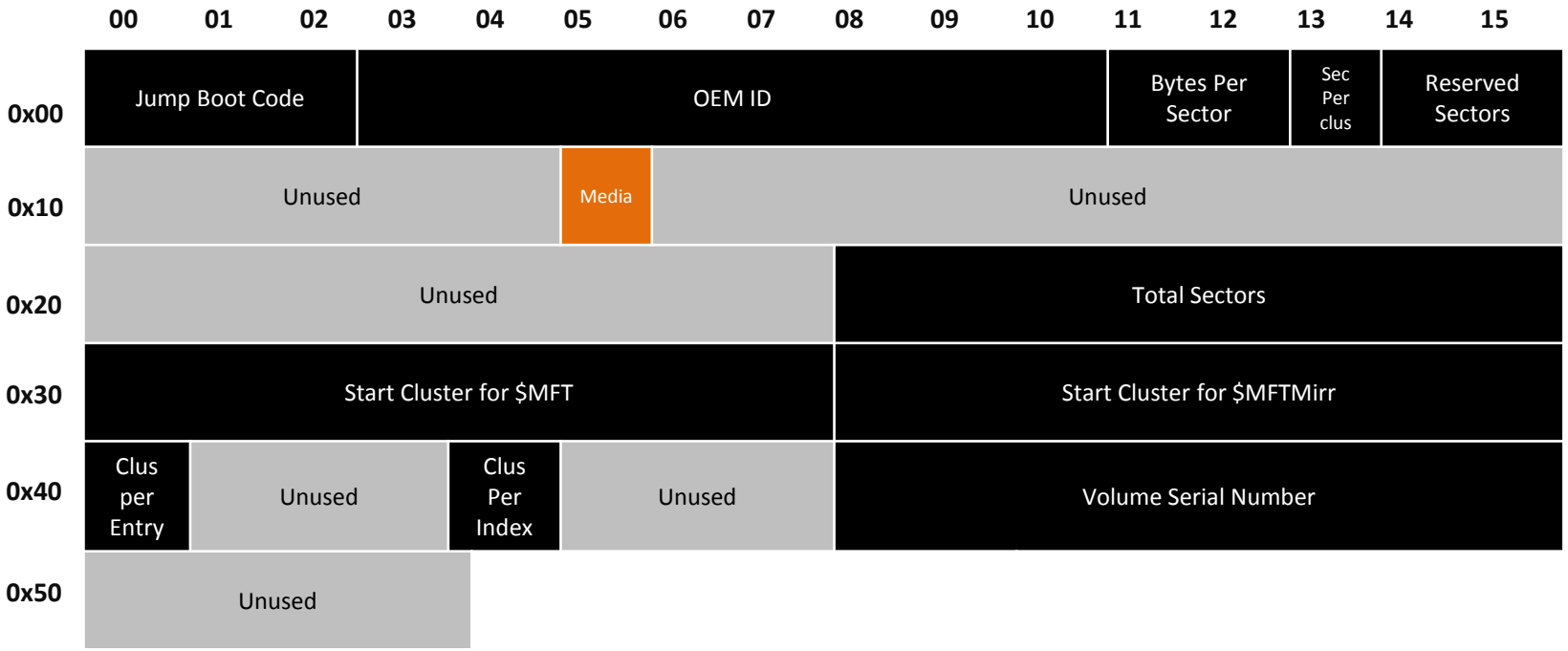
- **OEM ID** : 제조사 식별값
  - NTFS : "NTFS "

VBR (Volume Boot Record) ➔ BIOS Parameter Block



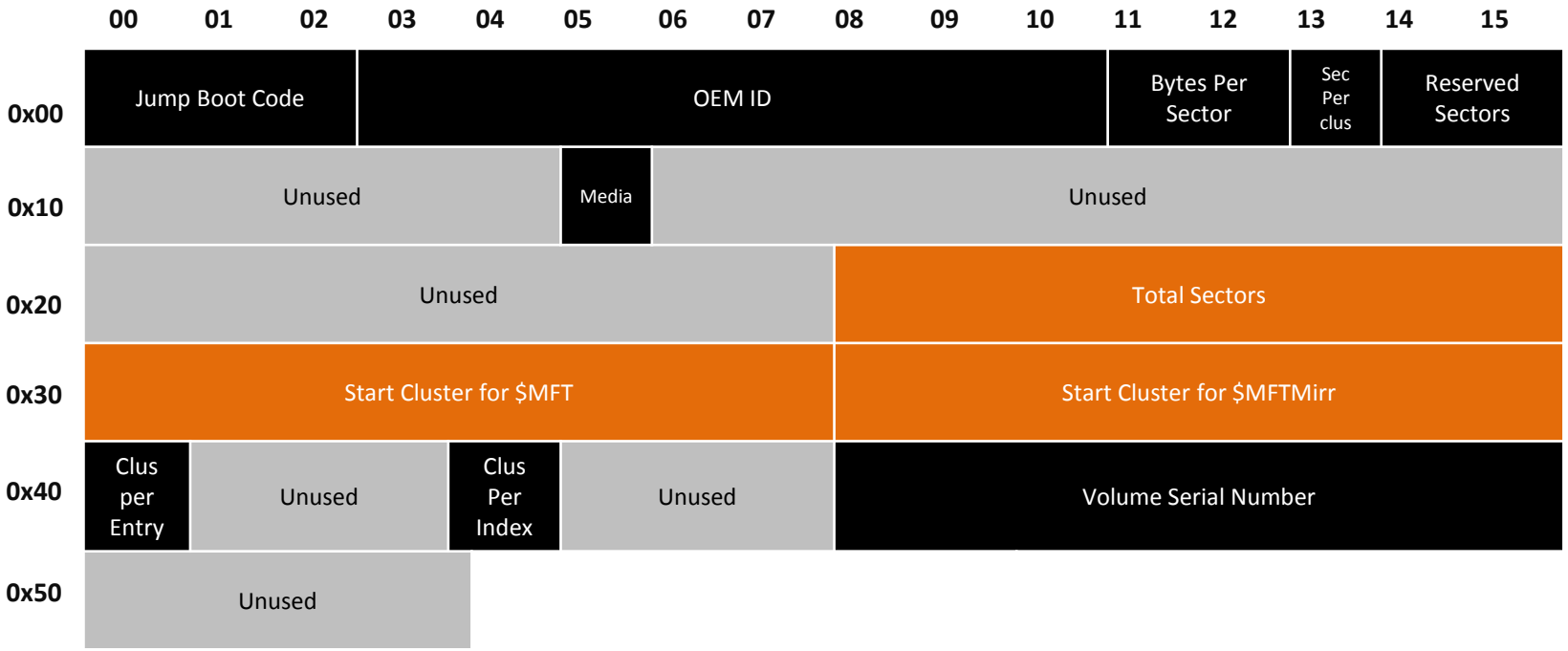
- **Bytes Per Sector :** 섹터 당 바이트 수
- **Sectors Per Cluster :** 클러스터 당 섹터 수 ➔ Bytes Per Sector X Sectors Per Cluster = 클러스터 크기
- **Reserved Sectors** : NTFS는 항상 파티션 맨 앞에 부트 섹터가 존재하므로 0x00

VBR (Volume Boot Record) ➔ BIOS Parameter Block



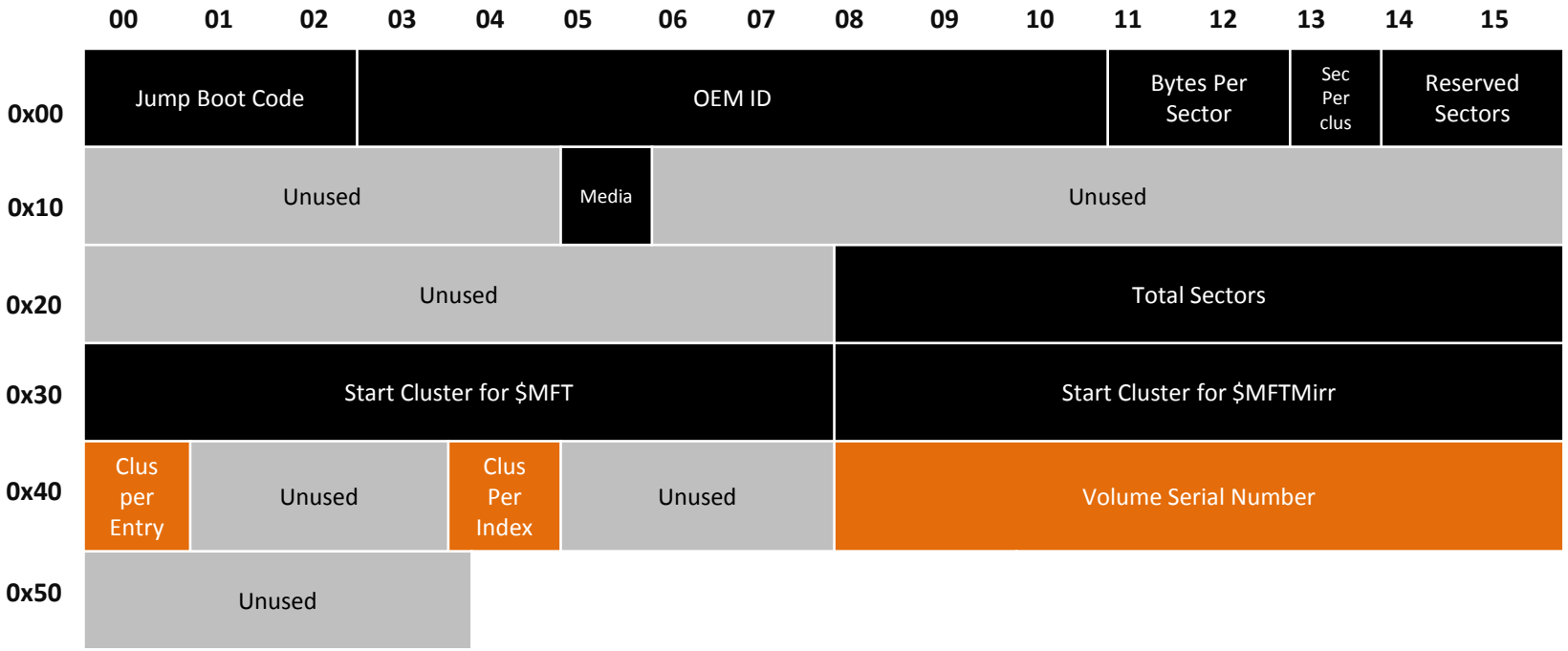
- **Media Descriptor**
  - 0xF8 : 고정식 디스크
  - 나머지 값은 플로피 디스크 구분 (현재 플로피 사용 X)

VBR (Volume Boot Record) ➔ BIOS Parameter Block



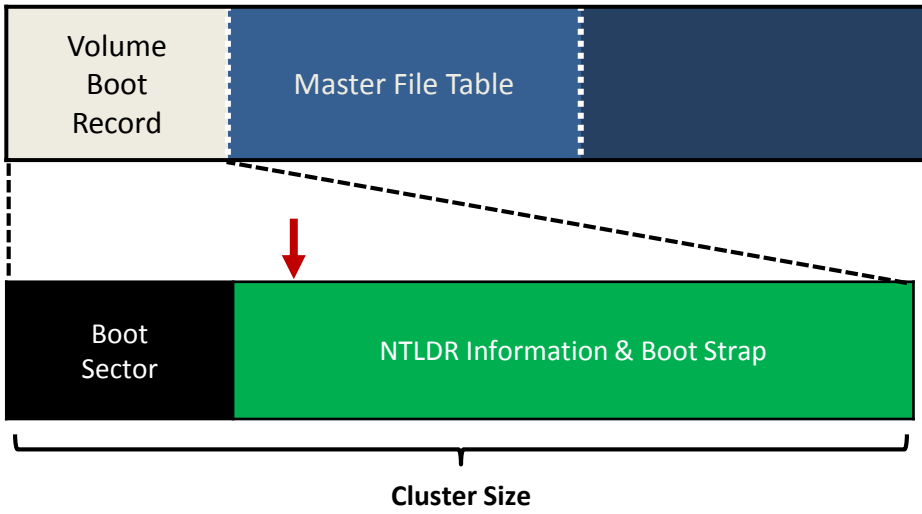
- **Total Sectors** : 해당 볼륨이 가지는 총 섹터 수
- **Start Cluster for $MFT** : $MFT의 LBA 주소
- **Start Cluster for $MFTMirr** : $MFTMirr의 LBA 주소

VBR (Volume Boot Record) ➔ BIOS Parameter Block



- **Clusters Per MFT Entry(Record)** : MFT Entry 크기
- **Clusters Per Index Buffer** : 디렉터리 공간을 할당하기 위해 사용되는 인덱스 버퍼의 크기
- **Volume Serial Number** : 볼륨 시리얼 번호

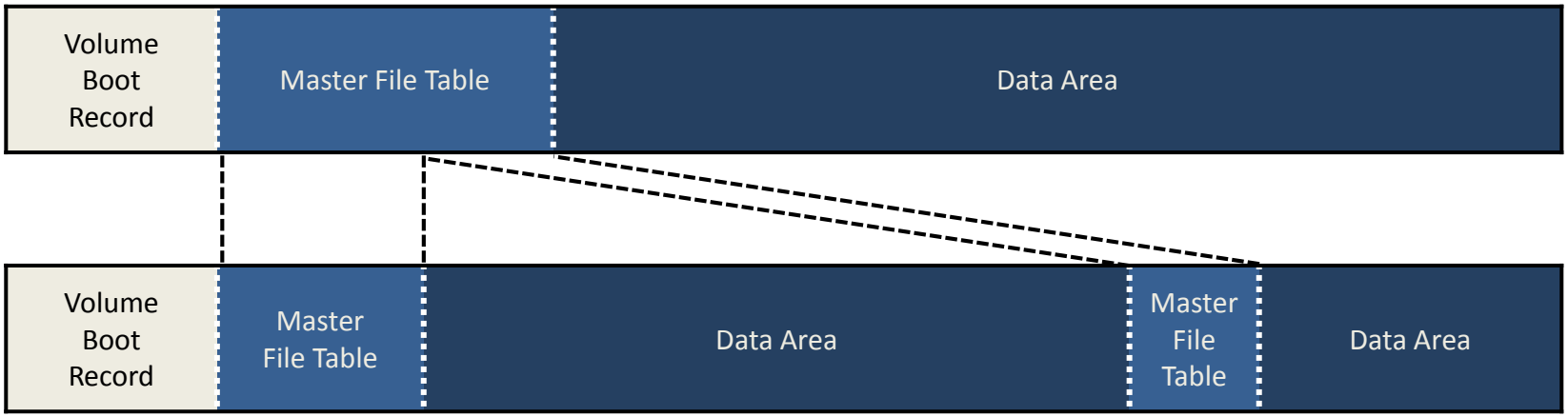
VBR (Volume Boot Record)




```
000 07 00 42 00 4F 00 4F 00 54 00 4D 00 47 00 52 00  ··B·O·O·T·M·G·R·
016 04 00 24 00 49 00 33 00 30 00 00 D4 00 00 00 24  ··$·I·3·0···Ô···$
032 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ····************
048 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ····************
064 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ····************
080 00 00 00 00 00 00 EB 22 90 90 05 00 4E 00 54 00  ·····ë"||··N·T·
096 4C 00 44 00 52 00 00 00 00 00 00 00 00 00 00 00  L·D·R···********
112 00 00 00 00 00 00 00 00 00 00 66 0F B7 06 0B 00  ··········f·····
128 66 0F B6 1E 0D 00 66 F7 E3 66 A3 52 02 66 8B 0E  f·¶···÷ãf£R·f‹··
144 40 00 80 F9 00 0F 8F 0E 00 F6 D9 66 B8 01 00 00  @·€ù····öÙf¸····
160 00 66 D3 E0 EB 08 90 66 A1 52 02 66 F7 E1 66 A3  ·fÓàë··f¡R·f÷áf£
176 66 02 66 0F B7 1E 0B 00 66 33 D2 66 F7 F3 66 A3  f·f·····f3Òf÷óf£
192 56 02 E8 95 04 66 8B 0E 4E 02 66 89 0E 26 02 66  V·è•·f‹·N·f‰·&·f
208 03 0E 66 02 66 89 0E 2A 02 66 03 0E 66 02 66 89  ··f·f‰·*·f··f·f‰
224 0E 2E 02 66 03 0E 66 02 66 89 0E 3E 02 66 03 0E  ·.·f··f·f‰·>·f··
240 66 02 66 89 0E 46 02 66 B8 90 00 00 00 66 8B 0E  f·f‰·F·f¸····f‹·
256 26 02 E8 83 09 66 0B C0 0F 84 5E FE 66 A3 32 02  &·èƒ·f·À·„^þf£2·
272 66 B8 A0 00 00 00 66 8B 0E 2A 02 E8 6A 09 66 A3  f¸ ···f‹·*·èj·f£
288 36 02 66 B8 B0 00 00 00 66 8B 0E 2E 02 E8 58 09  6·f¸°···f‹·.·èX·
304 66 A3 3A 02 66 A1 32 02 66 0B C0 0F 84 2B FE 67  f£:·f¡2·f·À·„+þg
320 80 78 08 00 0F 85 22 FE 67 66 8D 50 10 67 03 42  €x····…"þgf|P·g·B
336 04 67 66 0F B6 48 0C 66 89 0E 72 02 67 66 8B 48  ·gf·¶H·f‰·r·gf‹H
352 08 66 89 0E 6E 02 66 A1 6E 02 66 0F B7 0E 0B 00  ·f‰·n·f¡n·f·····
368 66 33 D2 66 F7 F1 66 A3 76 02 66 A1 46 02 66 03  f3Òf÷ñf£v·f¡F·f·
384 06 6E 02 66 A3 4A 02 66 83 3E 36 02 00 0F 84 1D  ·n·f£J·fƒ>6····„·
400 00 66 83 3E 3A 02 00 0F 84 CF FD 66 8B 1E 3A 02  ·fƒ>:····„Ïýf‹·:·
416 1E 07 66 8B 3E 4A 02 66 A1 2E 02 E8 E0 01 66 0F  ··f‹>J·f¡.·èà·f·
432 B7 0E 00 02 66 B8 02 02 00 00 E8 22 08 66 0B C0  ····f¸····è"·f·À
448 0F 85 16 00 66 0F B7 0E 5A 02 66 B8 5C 02 00 00  ·…··f···Z·f¸\···
464 E8 0C 08 66 0B C0 0F 84 42 0C 67 66 8B 00 1E 07  è··f·À·„B·gf‹···
480 66 8B 3E 3E 02 E8 3F 06 66 A1 3E 02 66 BB 20 00  f‹>>·è?·f¡>·f» ·
496 00 00 66 B9 00 00 00 00 66 BA 00 00 00 00 E8 E4  ··f¹····f°····èä
```

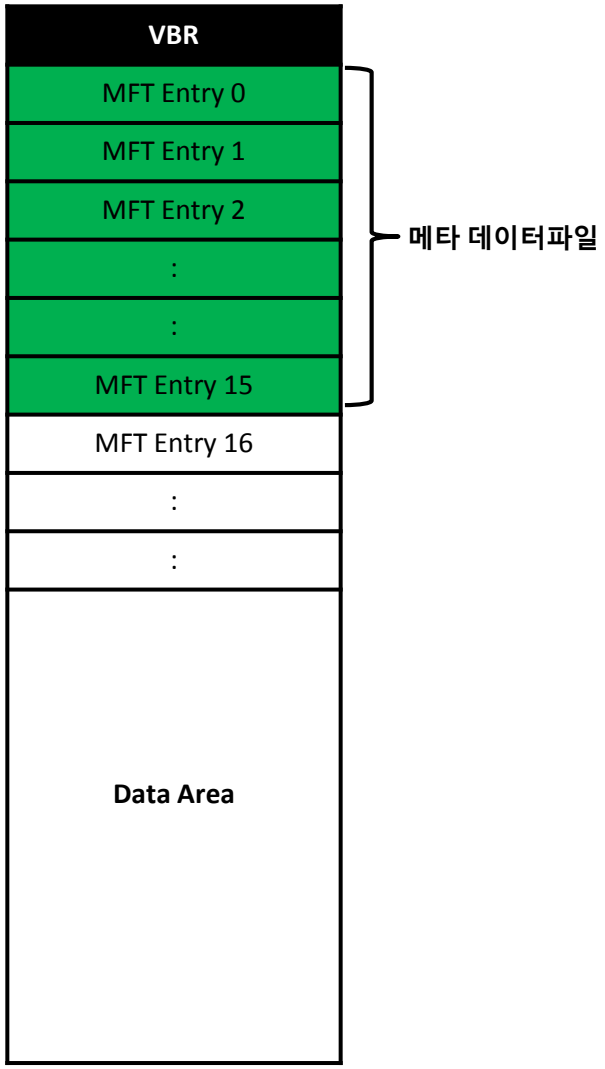
- NTFS 부트 로더의 위치 정보
- 윈도우 NT, 2000, XP : **NTLDR** (NT Loader)
  - C:₩
- 윈도우 Vista, 7 : **BOOTMGR** (Boot Manager)
  - C:₩Windows₩Boot₩

MFT (Master File Table)



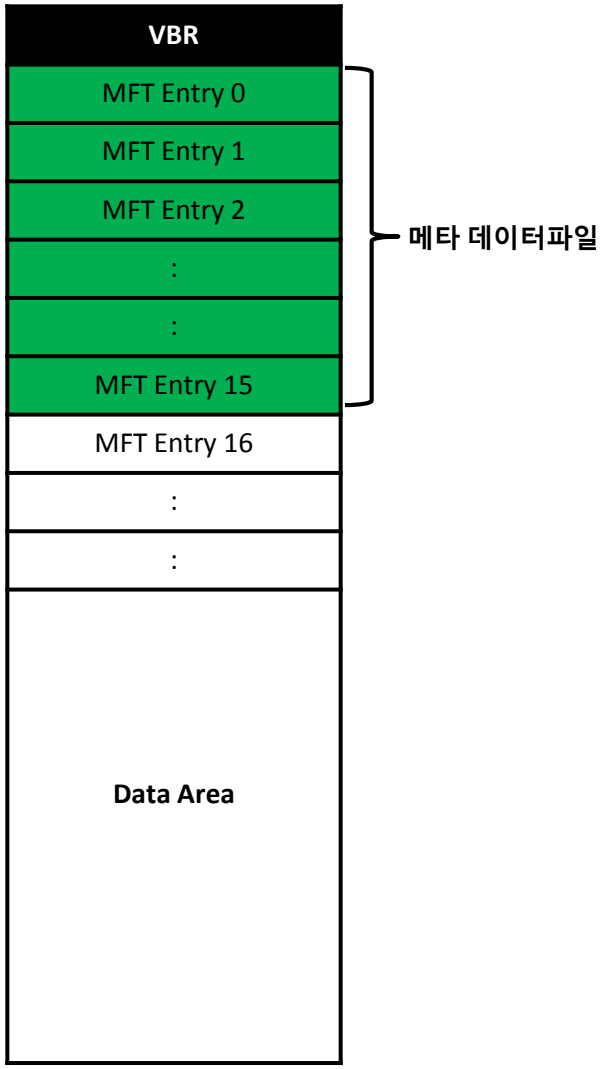
- NTFS는 파일, 디렉터리, 메타정보를 모두 파일 형태로 관리
- 각 파일의 위치, 속성, 시간정보, 이름, 크기 등의 메타정보는 MFT Entry라는 특별한 구조로 저장
- MFT(Master File Table)는 NTFS 상에 존재하는 모든 파일의 MFT Entry의 모음
- MFT 영역은 파일시스템 상의 파일 수에 따라 동적으로 할당
- 일반적으로 볼륨의 12.5% 정도가 MFT 영역으로 할당
- MFT Entry 0 ~ 15번은 파일시스템 생성 시 함께 생성되는 예약된(특별한 용도) 영역

MFT (Master File Table)



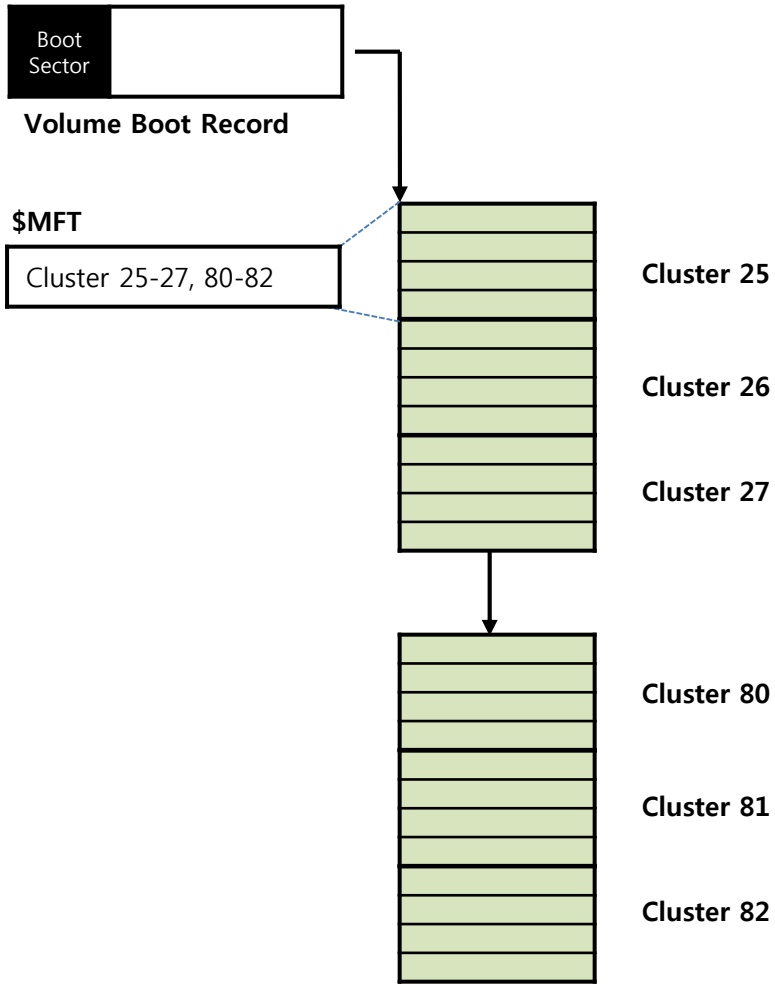
Entry 번호	Entry 이름	설명
0	$MFT	NTFS 상의 모든 파일들의 MFT Entry 정보
1	$MFTMirr	$MFT 파일의 일부 백업본
2	$LogFile	메타데이터의 트랜잭션 저널 정보
3	$Volume	볼륨의 레이블, 식별자, 버전 등의 정보
4	$AttrDef	속성의 식별자, 이름, 크기 등의 정보
5	.	볼륨의 루트 디렉터리
6	$Bitmap	볼륨의 클러스터 할당 정보
7	$Boot	볼륨이 부팅 가능할 경우 부트 섹터 정보
8	$BadClus	배드 섹터를 가지는 클러스터 정보
9	$Secure	파일의 보안, 접근 제어와 관련된 정보
10	$Upcase	모든 유니코드 문자의 대문자
11	$Extend	$ObjID, $Quota, $Reparse points, $UsnJrnl 등의 추가적인 파일의 정보를 기록하기 위해 사용
12 – 15		미래를 위해 예약
16 -		포맷 후 생성되는 파일의 정보를 위해 사용
-	$ObjId	파일 고유의 ID 정보 (Windows 2000 -)
-	$Quota	사용량 정보 (Windows 2000 -)
-	$Reparse	Reparse Point 에 대한 정보 (Windows 2000 -)
-	$UsnJrnl	파일, 디렉터리의 변경 정보 (Windows 2000 -)

MFT (Master File Table)

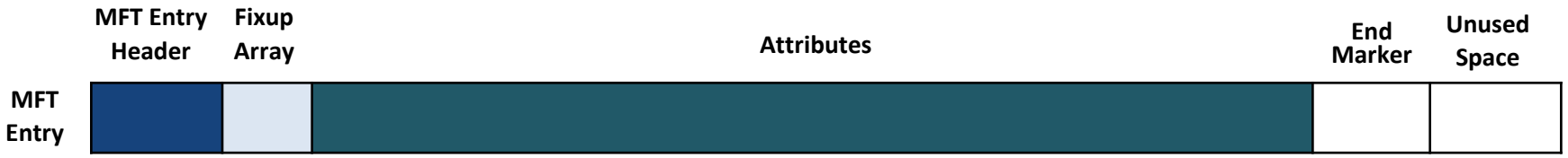


Entry 번호	Entry 이름	설명
0	$MFT	MFT에 대한 MFT Entry
1	$MFTMirr	$MFT 파일의 일부 백업본
2	$LogFile	메타데이터(MFT)의 트랜잭션 저널 정보
3	$Volume	볼륨의 레이블, 식별자, 버전 등의 정보
4	$AttrDef	속성의 식별자, 이름, 크기 등의 정보
5	.	볼륨의 루트 디렉터리
6	$Bitmap	볼륨의 클러스터 할당 정보
7	$Boot	볼륨이 부팅 가능할 경우 부트 섹터 정보
8	$BadClus	배드 섹터를 가지는 클러스터 정보
9	$Secure	파일의 보안, 접근 제어와 관련된 정보
10	$Upcase	모든 유니코드 문자의 대문자
11	$Extend	$ObjID, $Quota, $Reparse points, $UsnJrnl 등의 추가적인 파일의 정보를 기록하기 위해 사용
12 – 15		미래를 위해 예약
16 -		포맷 후 생성되는 파일의 정보를 위해 사용
-	$ObjId	파일 고유의 ID 정보 (Windows 2000 -)
-	$Quota	사용량 정보 (Windows 2000 -)
-	$Reparse	Reparse Point 에 대한 정보 (Windows 2000 -)
-	$UsnJrnl	파일, 디렉터리의 변경 정보 (Windows 2000 -)

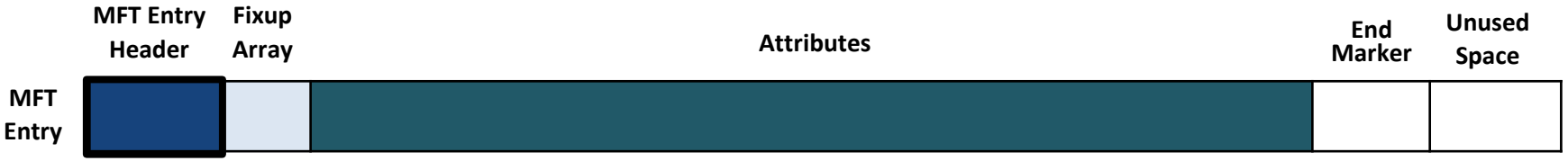
MFT (Master File Table)



MFT Entry

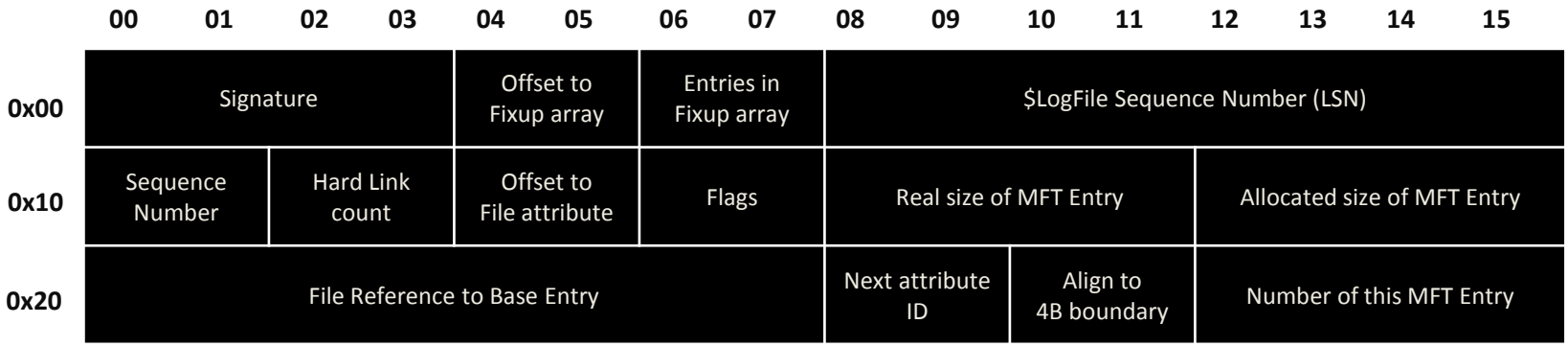

MFT Entry Header
Fixup Array
Attributes
End Marker
Unused Space
MFT Entry

MFT Entry ➔ MFT Entry Header

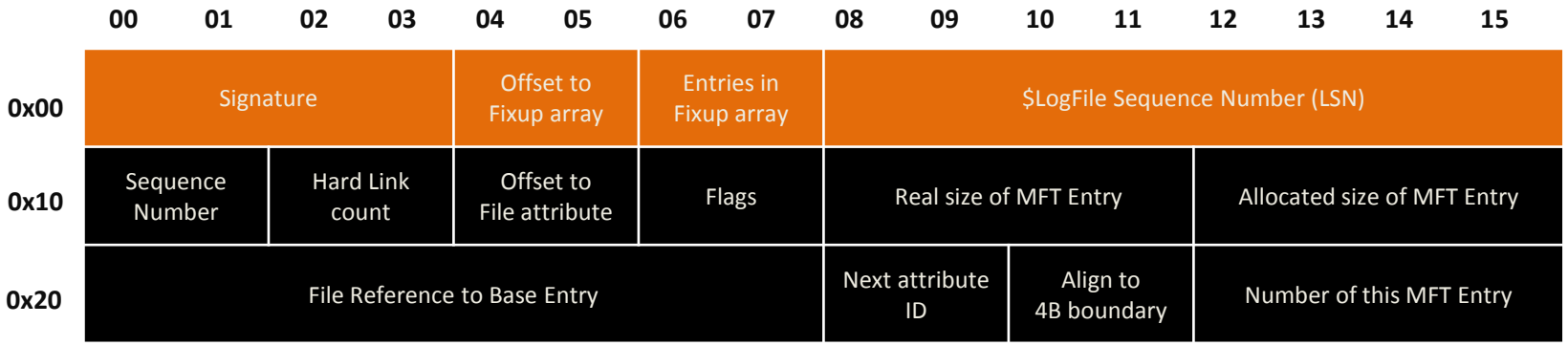


- MFT Entry의 메타정보를 담고 있는 48 바이트 크기의 영역

MFT Entry ➔ MFT Entry Header

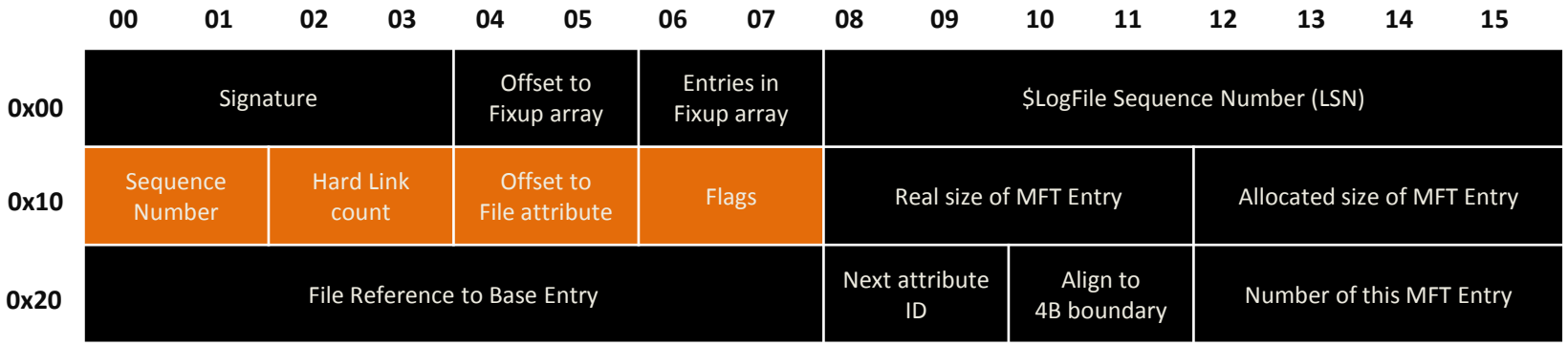


MFT Entry ➔ MFT Entry Header



- **Signature** : MFT Entry 시그니처 (“FILE”)
- **Offset to Fixup array** : Fixup 배열의 시작 위치
- **Number of entries in Fixup array** : Fixup 배열이 포함하는 항목 수
- **$LogFile Sequence Number (LSN)** : $LogFile에 존재하는 해당 파일의 트랜잭션 위치 값
  - MFT Entry가 변경될 때마다 갱신

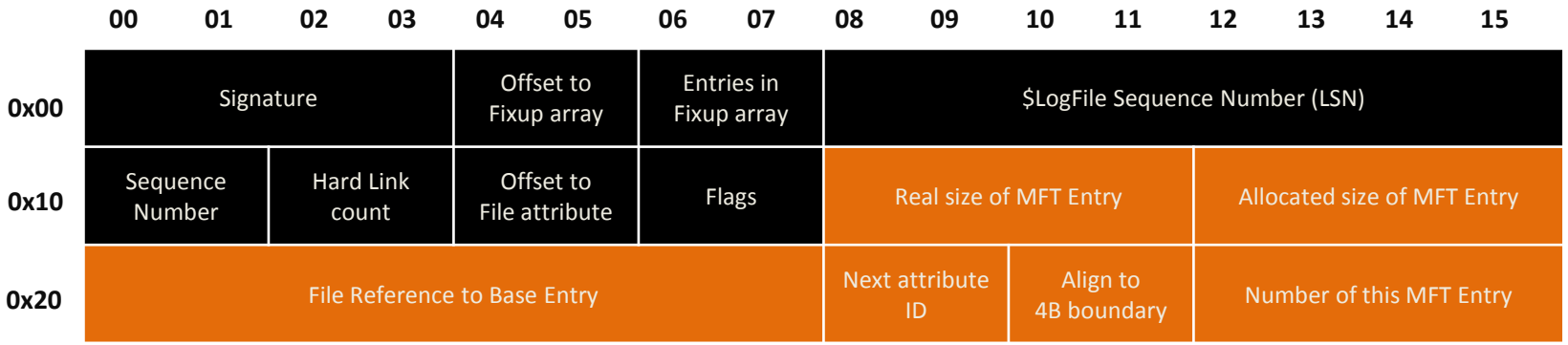
MFT Entry ➔ MFT Entry Header



- **Sequence Number** : 순서 번호로 MFT Entry 생성 후 할당/해제 시마다 1씩 증가
- **Link count** : 해당 MFT Entry에 연결된 하드 링크
- **Offset to File attribute** : 해당 Entry의 첫 번째 속성의 위치값
- **Flags** : MFT Entry 상태 정보 ➔ Flags 값이 0x00 일 경우는? 0x03일 경우는?

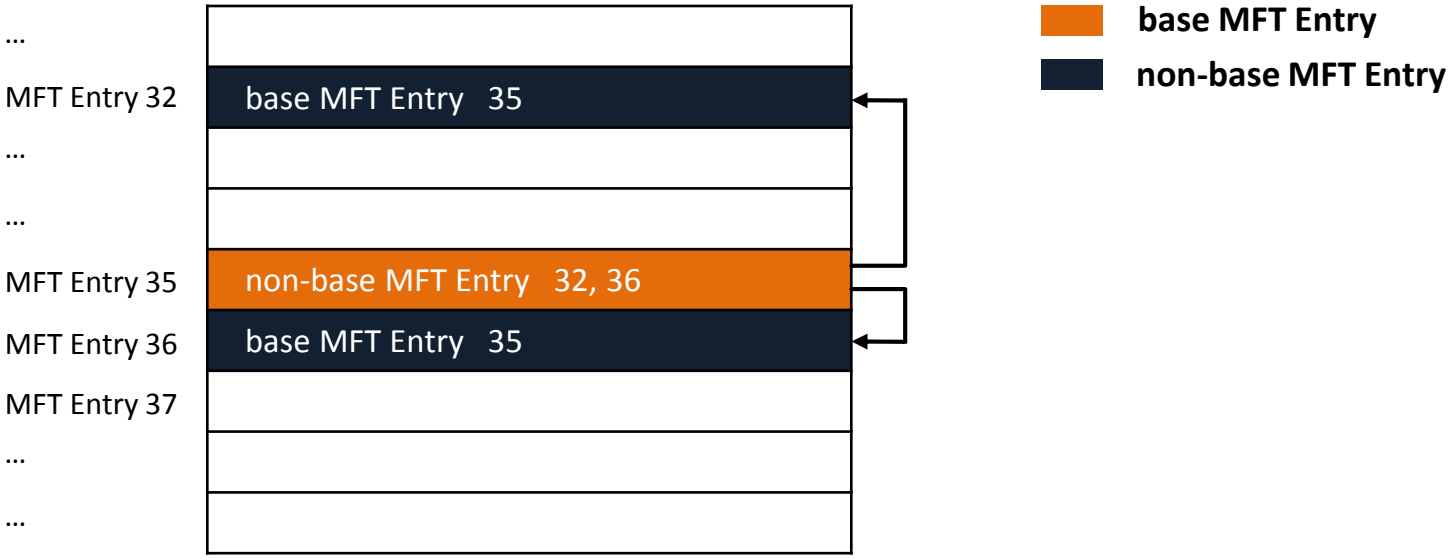
Flags 값	설명
0x01	사용 중
0x02	디렉터리
0x04	분석 안됨
0x08	분석 안됨

MFT Entry ➔ MFT Entry Header



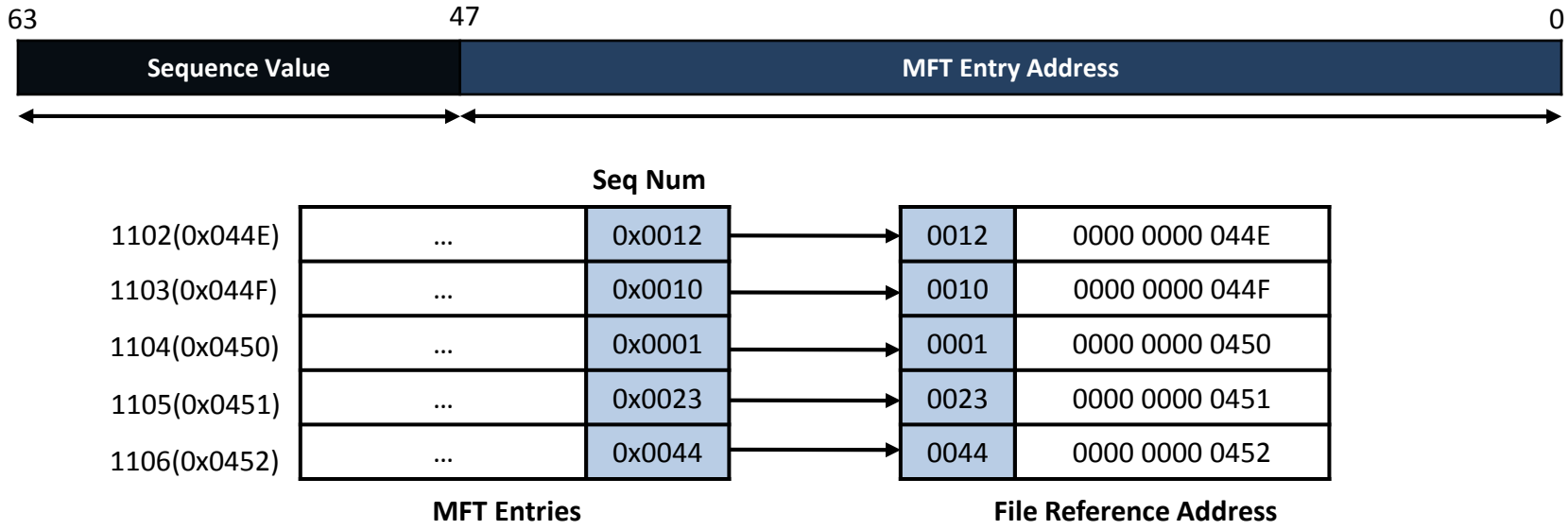
- **Real size of MFT Entry** : 실제 사용 중인 크기
- **Allocated size of MFT Entry** : MFT Entry에 할당된 크기 (1,024 바이트)
- **File Reference to Base Entry** : 해당 Entry의 non-base일 경우 자신의 base Entry의 주소값
- **Next attribute ID** : 다음 속성 ID (Entry 내에서만 유효한 값)
- **Align to 4B boundary (XP)**
- **Number of this MFT Entry (XP)**

Base & Non-base MFT Entry



- 파일 속성 내용이 클 경우 하나 이상의 MFT Entry 사용
- Base MFT Entry : 해당 파일의 첫 MFT Entry
- Non-base MFT Entry : 해당 파일의 나머지 MFT Entry

File Reference Address



- MFT Entry는 48 비트의 고유한 주소 값을 가짐
  - MFT Entry Address (MFT Entry Number)
- NTFS는 특정 MFT Entry 접근 시 48 비트 주소를 확장한 파일 참조 주소(File Reference Address) 사용
- **File Reference Address = Sequence Number + MFT Entry Address**
- 파일 참조 주소를 사용하는 이유는?

MFT Entry ➔ Fixup Array

- Fixup : "수리하다", "고치다" ➔ MFT Entry 데이터 무결성 판단
- MFT Entry는 1,024 바이트이므로 2개의 섹터 사용
- 각 섹터의 마지막 2바이트를 이용해 Fixup Array 구현
- MFT Entry 구조 외에도 다양한 구조에서 Fixup Arrary 사용 (클러스터 크기에 따라 2개 이상의 섹터 사용)

- FILE Records(MFT Entries) in the $MFT
- INDX Records in directories and other indexes
- RCRD Records in the $LogFile
- RSTR Records in the $LogFile

MFT Entry ➔ Fixup Array

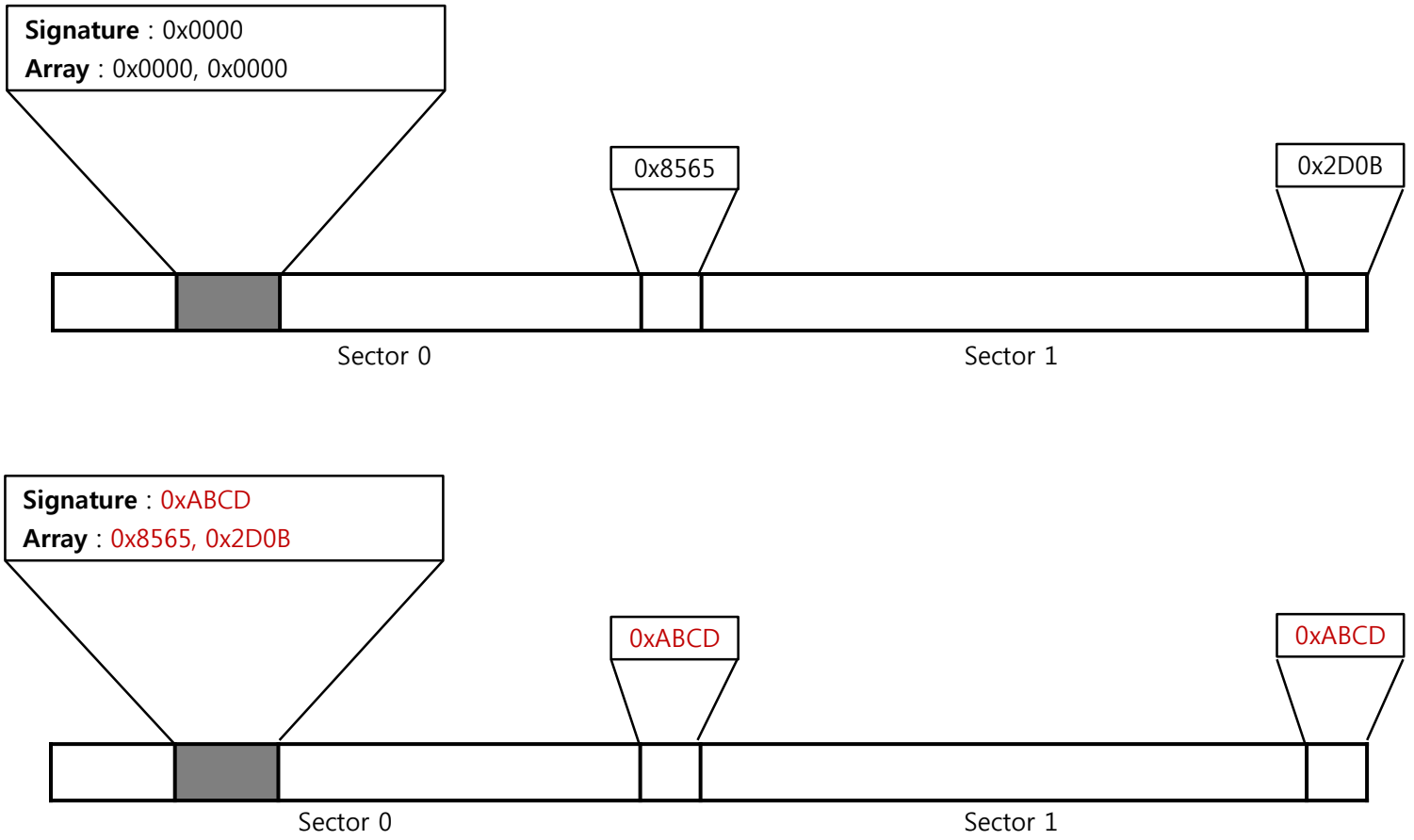
위치	데이터								설명
0x0000	… (48 Byte)								MFT Entry Header
0x0030	CD	AB	00	00	00	00	00	00	Fixup Array
…	…								
0x01F8	11	12	13	14	15	16	17	18	End of Sector 1
…	…								
0x03F8	21	22	23	24	25	26	27	28	End of Sector 2

원본 데이터

위치	데이터								설명
0x0000	… (82 Byte)								MFT Entry Header
0x0030	CD	AB	17	18	27	28	00	00	Fixup Array
…	…								
0x01F8	11	12	13	14	15	16	CD	AB	End of Sector 1
…	…								
0x03F8	21	22	23	24	25	26	CD	AB	End of Sector 2

Fixup이 적용된 데이터

MFT Entry ➔ Fixup Array



MFT Entry ➔ Attributes

MFT Entry ➔ Attributes

속성 식별값		속성이름	설명
16	0x10	**$STANDARD_INFORMATION**	파일의 생성.접근.수정 시간, 소유자 등의 일반적인 정보
32	0x20	**$ATTRIBUTE_LIST**	추가적인 속성들의 리스트
48	0x30	**$FILE_NAME**	파일 이름(유니코드), 파일의 생성.접근.수정 시간
64	0x40	**$VOLUME_VERSION**	볼륨 정보 (Windows NT 1.2 버전에만 존재)
64	0x40	**$OBJECT_ID**	16바이트의 파일, 디렉터리의 고유 값, 3.0 이상에서만 존재
80	0x50	**$SECURITY_DESCRIPTOR**	파일의 접근 제어와 보안 속성
96	0x60	**$VOLUME_NAME**	볼륨 이름
112	0x70	**$VOLUME_INFORMATION**	파일 시스템의 버전과 다양한 플래그
128	0x80	**$DATA**	파일 내용
144	0x90	**$INDEX_ROOT**	인덱스 트리의 루트 노드
160	0xA0	**$INDEX_ALLOCATION**	인덱스 트리의 루트와 연결된 노드
176	0xB0	**$BITMAP**	$MFT와 인덱스의 할당 정보 관리
192	0xC0	**$SYMBOLIC_LINK**	심볼릭 링크 정보 (Windows 2000+)
192	0xC0	**$REPARSE_POINT**	심볼릭 링크에서 사용하는 reparse point 정보 (Windows 2000+)
208	0xD0	**$EA_INFORMATION**	OS/2 응용 프로그램과 호환성을 위해 사용 (HPFS)
224	0xE0	**$EA**	OS/2 응용 프로그램과 호환성을 위해 사용 (HPFS)
256	0x100	**$LOGGED_UTILITY_STREAM**	암호화된 속성의 정보와 키 값 (Windows 2000+)

MFT Entry ➔ Attributes

속성 식별값		속성이름	설명
16	0x10	**$STANDARD_INFORMATION**	파일의 생성.접근.수정 시간, 소유자 등의 일반적인 정보
32	0x20	**$ATTRIBUTE_LIST**	추가적인 속성들의 리스트
48	0x30	**$FILE_NAME**	파일 이름(유니코드), 파일의 생성.접근.수정 시간
64	0x40	**$VOLUME_VERSION**	볼륨 정보 (Windows NT 1.2 버전에만 존재)
64	0x40	**$OBJECT_ID**	16바이트의 파일, 디렉터리의 고유 값, 3.0 이상에서만 존재
80	0x50	**$SECURITY_DESCRIPTOR**	파일의 접근 제어와 보안 속성
96	0x60	**$VOLUME_NAME**	볼륨 이름
112	0x70	**$VOLUME_INFORMATION**	파일 시스템의 버전과 다양한 플래그
128	0x80	**$DATA**	파일 내용
144	0x90	**$INDEX_ROOT**	인덱스 트리의 루트 노드
160	0xA0	**$INDEX_ALLOCATION**	인덱스 트리의 루트와 연결된 노드
176	0xB0	**$BITMAP**	$MFT와 인덱스의 할당 정보 관리
192	0xC0	**$SYMBOLIC_LINK**	심볼릭 링크 정보 (Windows 2000+)
192	0xC0	**$REPARSE_POINT**	심볼릭 링크에서 사용하는 reparse point 정보 (Windows 2000+)
208	0xD0	**$EA_INFORMATION**	OS/2 응용 프로그램과 호환성을 위해 사용 (HPFS)
224	0xE0	**$EA**	OS/2 응용 프로그램과 호환성을 위해 사용 (HPFS)
256	0x100	**$LOGGED_UTILITY_STREAM**	암호화된 속성의 정보와 키 값 (Windows 2000+)

MFT Entry ➔ Attributes

- 각 파일의 메타정보(이름, 시간정보, 속성, 내용 등)는 속성이라는 구조를 통해 표현
- 각 속성은 속성 헤더와 속성 내용을 가짐
- 파일 특성에 따라 다양한 속성을 가짐
- 속성은 크기에 따라 Resident와 Non-resident 속성으로 나뉨

NTFS Internals

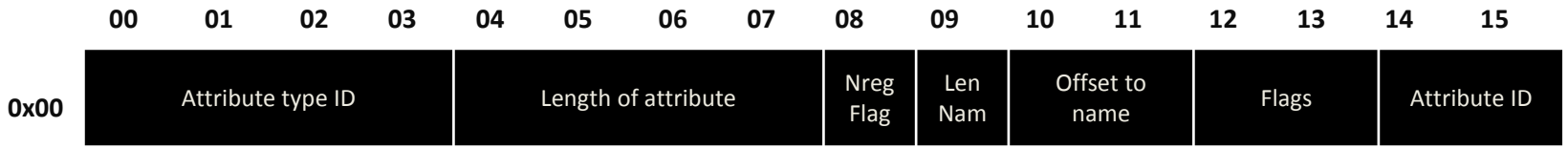
MFT Entry ➔ Attributes ➔ Resident & Non-resident Attribute

- **Resident 속성** : 속성 헤더 뒤에 바로 속성 내용이 저장 (MFT Entry 내부)
  - $STANDARD_INFORMATION, $FILE_NAME
- **Non-resident 속성** : 속성 내용이 너무 많이 별도의 클러스터에 저장 (MFT Entry 외부)
  - $ATTRIBUTE_LIST, $DATA
- $DATA 속성의 경우
  - 파일 크기 < 700 바이트 : Resident 속성
  - 파일 크기 > 700 바이트 : Non-resident 속성

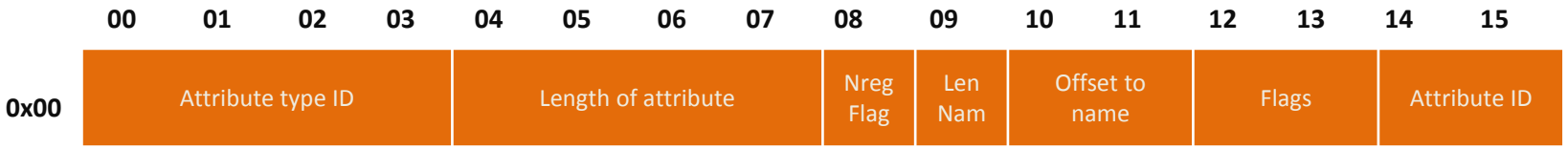
MFT Entry ➔ Attribute Header

- 각 속성은 속성의 메타정보 표현을 위해 속성 헤더를 가짐
- 속성 형식에 관계 없는 공통된 16 바이트 헤더 + 속성 형식(Resident/Non-resident)에 따라 추가적인 헤더

MFT Entry ➔ Common Attribute Header

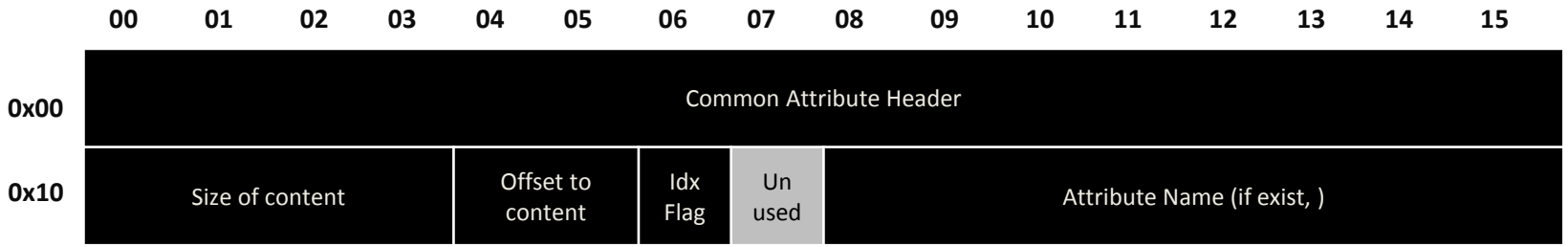


MFT Entry ➔ Common Attribute Header

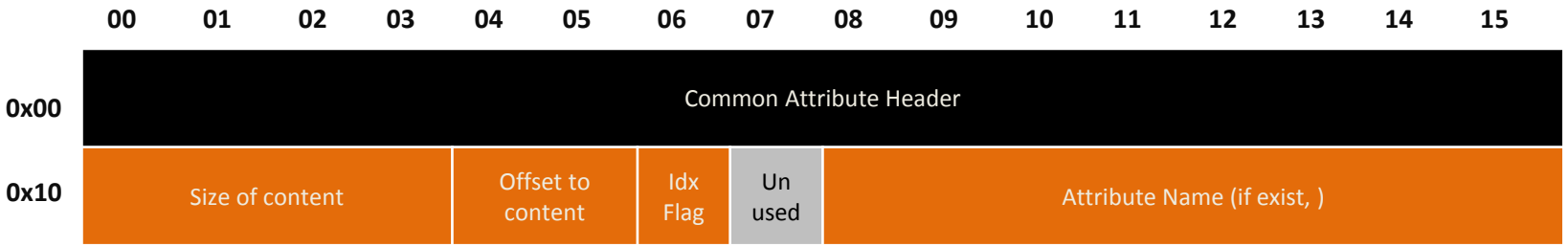


- **Attribute type identifier** : 속성 타입 식별값
- **Length of attribute** : 속성 헤더를 포함한 속성 전체 길이
- **Non-resident flag** : Non-resident 속성인지 여부 (0x01 값을 가진다면 Non-resident 속성)
- **Length of name** : 자신의 속성 이름 길이
- **Offset to name** : 속성 이름이 저장된 곳의 시작 위치
- **Flags** : 속성의 상태 표현
  - 0x0001 : 압축된 속성
  - 0x4000 : 암호화된 속성
  - 0x8000 : Sparse 속성
- **Attribute identifier** : 속성의 고유한 식별자로 MFT Entry에 같은 속성이 여러 개일 경우 구별하기 위해 사용

MFT Entry ➔ Resident Attribute Header

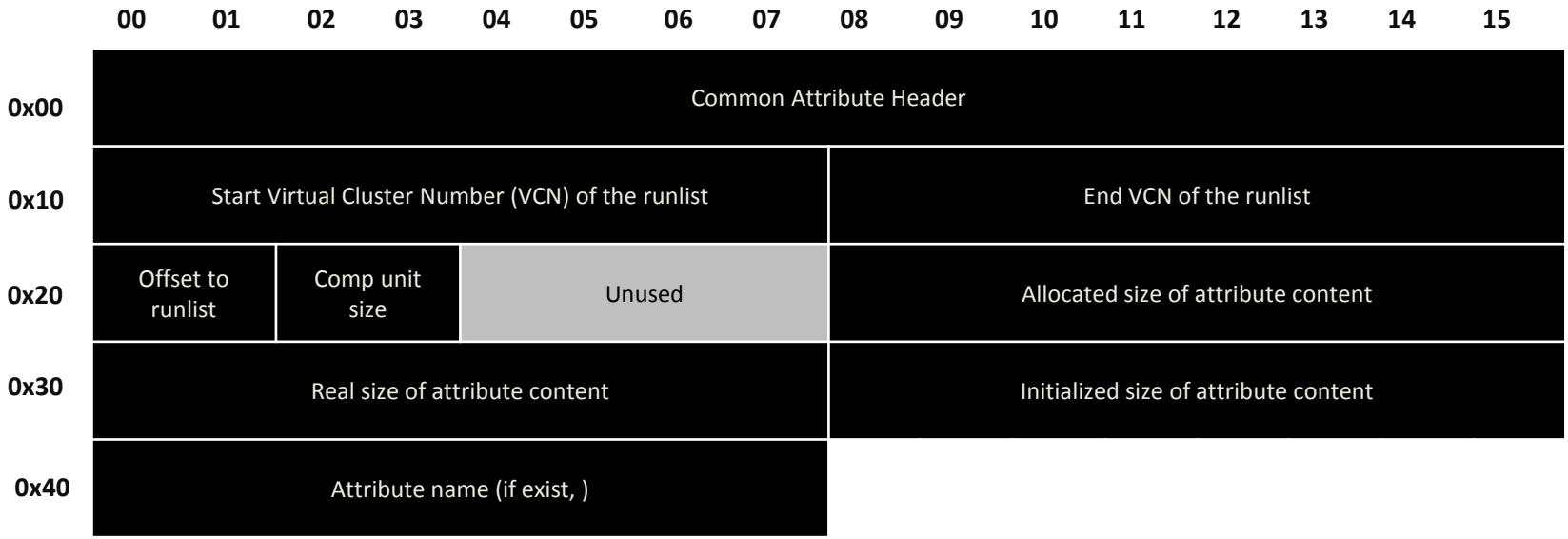


MFT Entry ➔ Resident Attribute Header

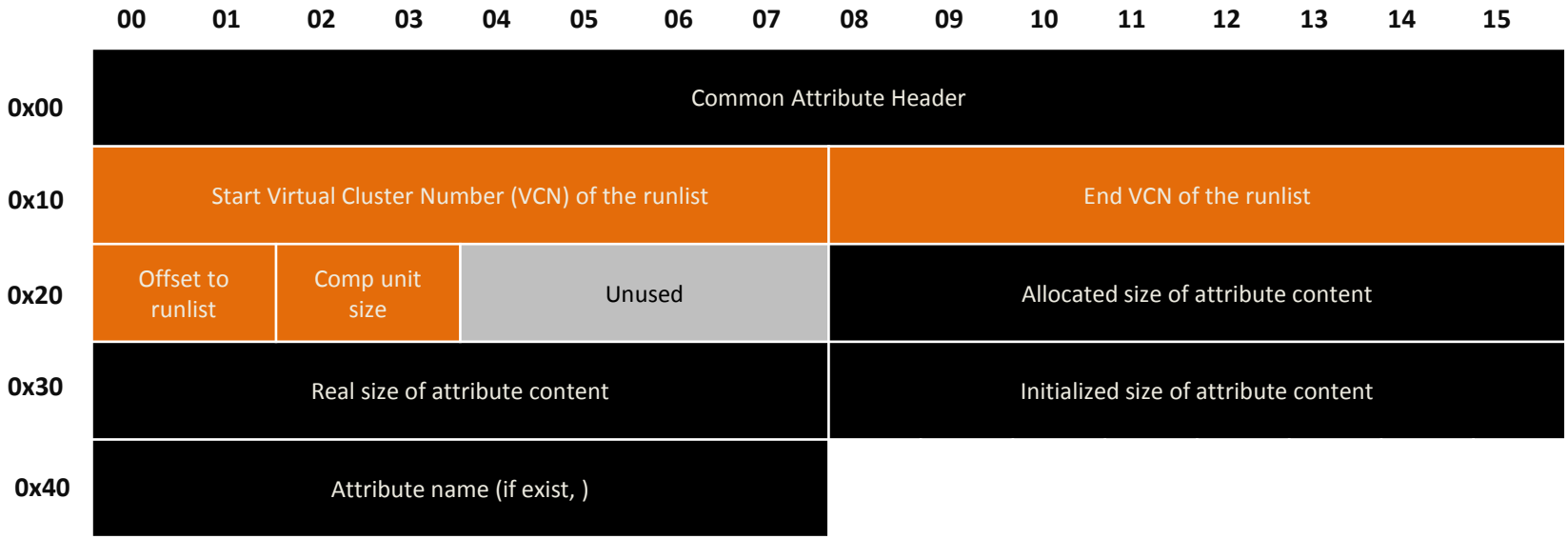


- **Size of content** : 헤더 뒤에 오는 속성 내용의 크기
- **Offset to content** : 속성 내용이 시작하는 곳의 위치
- **Indexed flag** : "1" 값을 가진다면 인덱스된 속성 ($FILE_NAME 속성은 "1" 로 설정)
- **Attribute name** : 속성 이름이 있는 경우 속성 이름, 없는 경우 바로 속성 내용

MFT Entry ➔ Non-resident Attribute Header

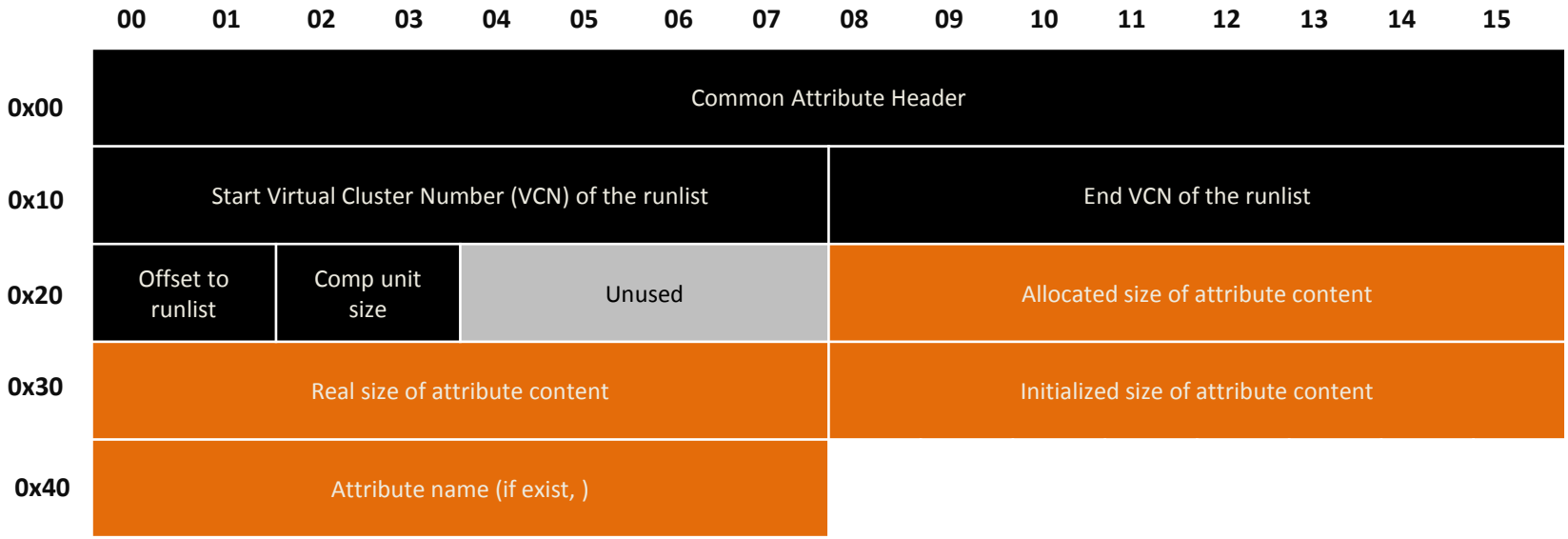


MFT Entry ➔ Non-resident Attribute Header



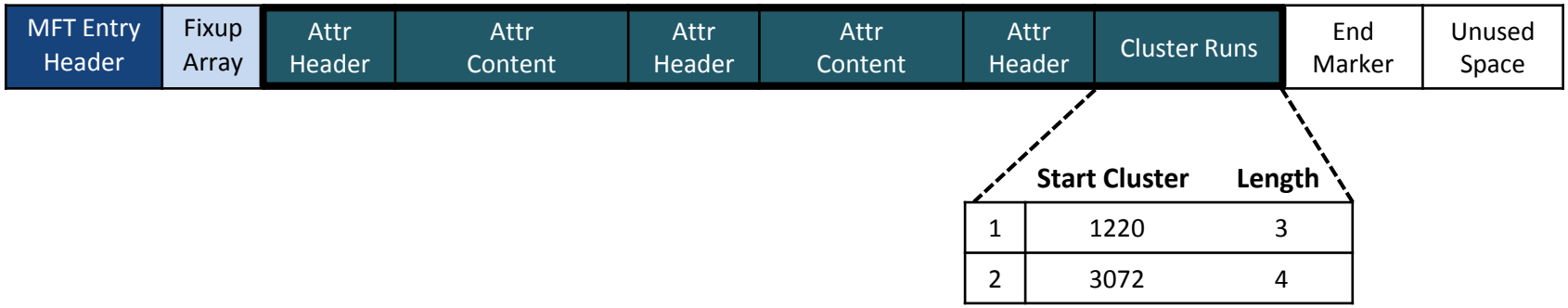
- **Start VCN of the runlist** : 속성 내용이 담긴 런리스트(runlist)의 시작 VCN
- **End VCN of the runlist** : 속성 내용이 담긴 런리스트(runlist)의 끝 VCN
- **Offset to runlist** : 속성 내부의 런리스트 시작 위치
- **Compression unit size** : 압축 속성일 경우 압축 단위

MFT Entry ➔ Non-resident Attribute Header

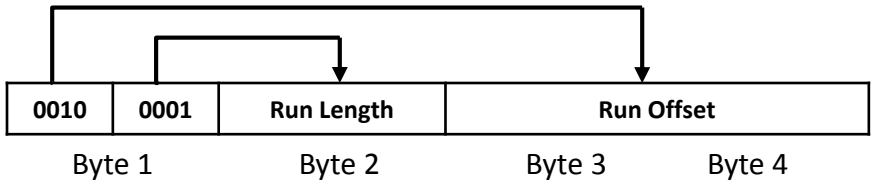


- **Allocated size of attribute content** : 속성 내용에 할당된 클러스터 크기 (클러스터 배수)
- **Real size of attribute content** : 속성 내용의 실제 크기
- **Initialized size of attribute content** : 속성 내용의 초기화된 크기
- **Attribute name** : 속성 이름이 있는 경우 속성 이름, 없는 경우 바로 속성 내용

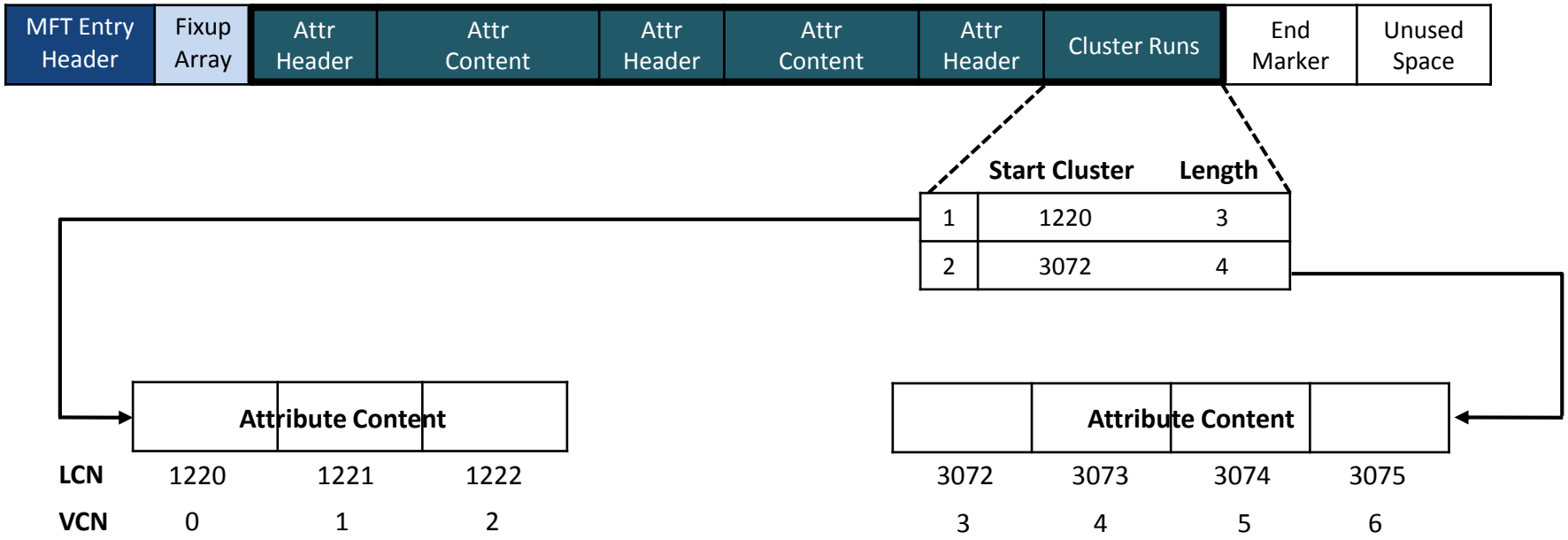
MFT Entry ➔ Cluster Runs



- Non-resident 속성의 경우 할당 받는 클러스터는 파일에 따라 수천~수만
- 비연속적으로 할당된 클러스터를 효과적으로 관리하기 위해 클러스터 런(Cluster Runs) 사용
- 클러스터 런은 런리스트(Runlist) 형태로 관리

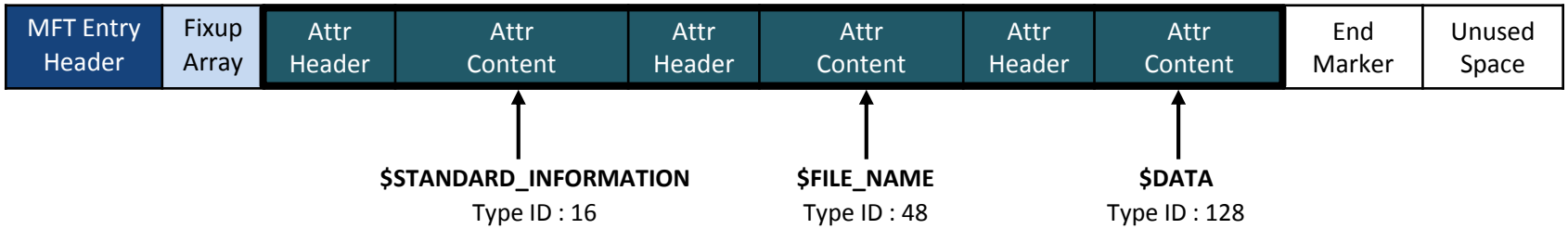


MFT Entry ➔ Logical Cluster Number (LCN) & Virtual Cluster Number (VCN)



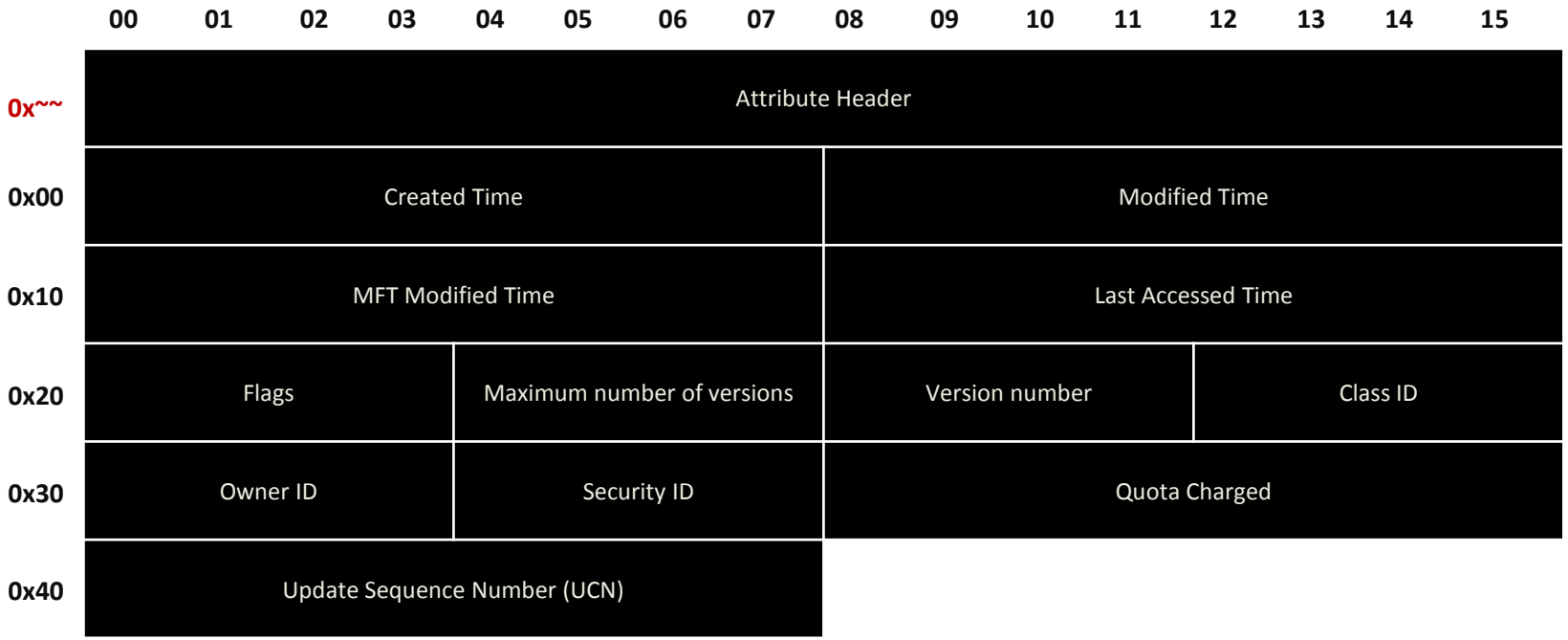
- **LCN** : 볼륨의 첫 번째 클러스터부터 순차적인 번호 (LBA?)
- **VCN** : 파일의 첫 번째 클러스터부터 순차적인 번호

MFT Entry ➔ Attribute Content



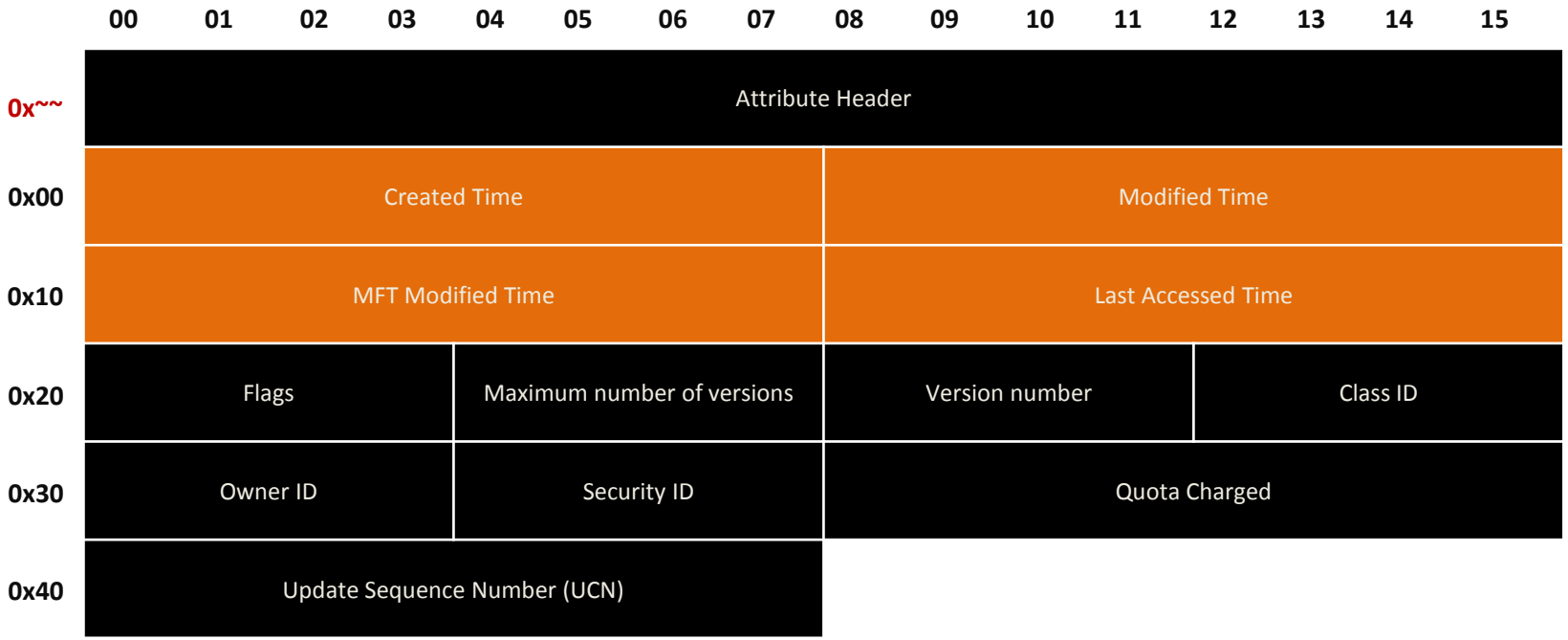
- 일반적인 파일은 3가지 속성을 가짐
  - **$STANDARD_INFORMATION** : 파일의 생성·접근·수정 시간, 소유자 등의 정보
  - **$FILE_NAME** : 파일 이름(유니코드), 파일의 생성·접근 ·수정 시간
  - **$DATA** : 파일 내용
- $ 이후 대문자가 온다면 속성 이름
- $ 이후 첫 글자만 대문자라면 메타데이터파일 ($MFT 제외)

MFT Entry ➔ Attributes ➔ $STANDARD_INFORMATION



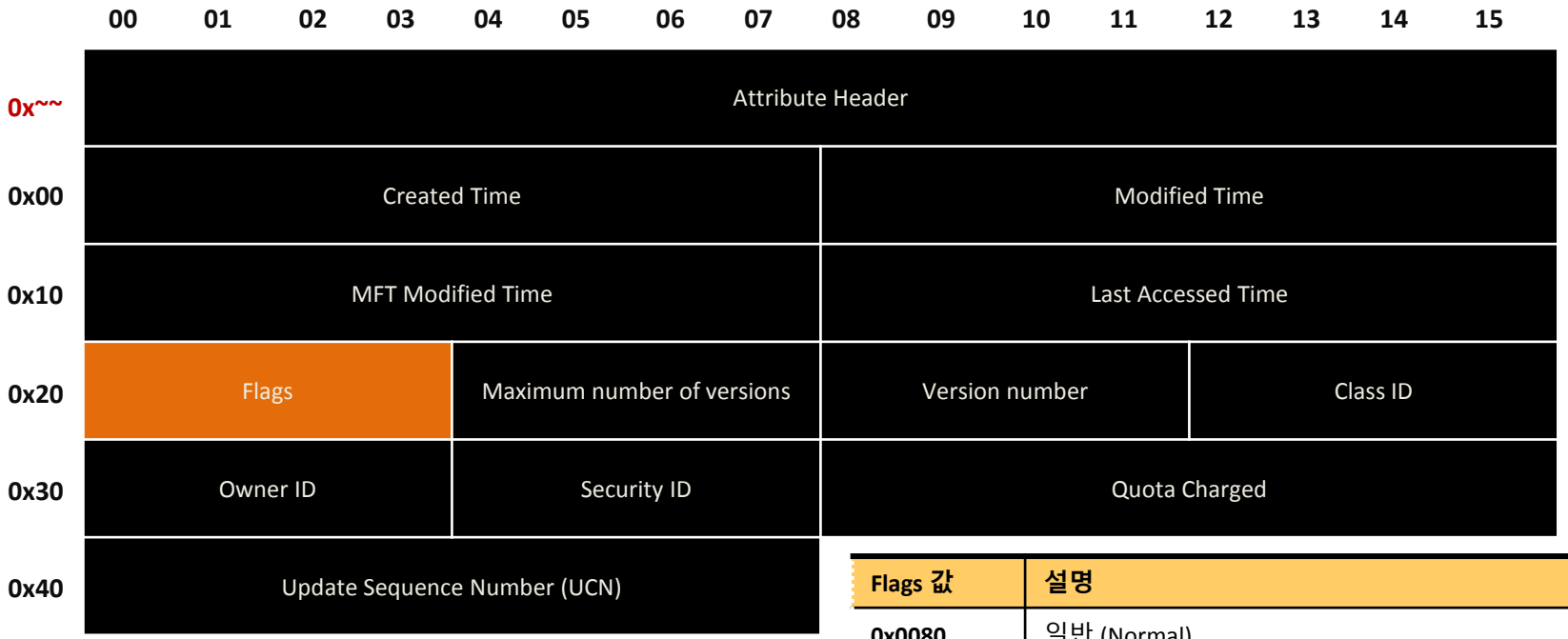
- 모든 파일에 존재하는 기본 속성으로 속성 식별값 16 (0x10)
- 속성 중 식별값이 가장 낮기 때문에 속성 구조에서 가장 처음에 위치
- 윈도우 NT에서는 48 바이트 크기, 2000 이상에서는 72 바이트

MFT Entry ➔ Attributes ➔ $STANDARD_INFORMATION



- **Windows 64-Bit Timestamp**
  - Created Time
  - Modified Time
  - MFT Modified Time
  - Last Accessed Time

MFT Entry ➔ Attributes ➔ $STANDARD_INFORMATION

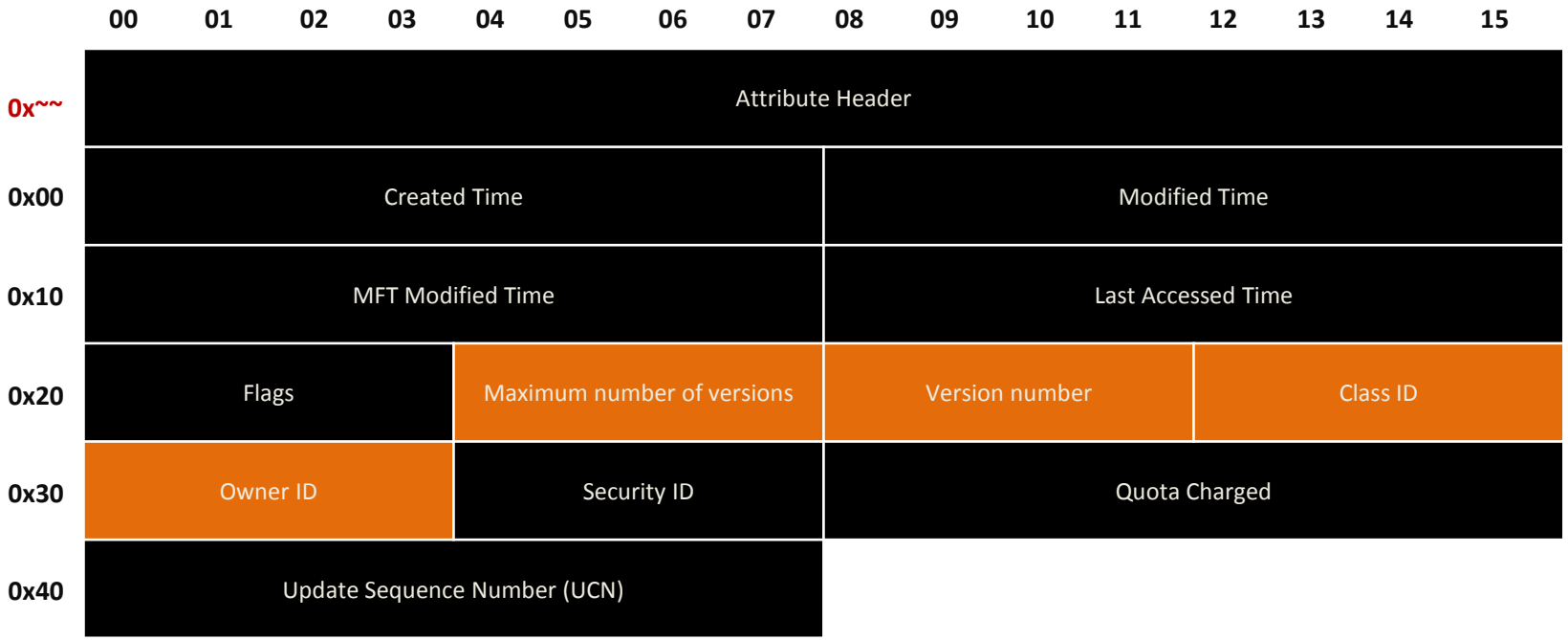


- **Flags**

Flags 값	설명
0x0001	읽기 전용 (Read Only)
0x0002	숨긴 파일 (Hidden)
0x0004	시스템 (System)
0x0020	아카이브 (Archive)
0x0040	장치 (Device)

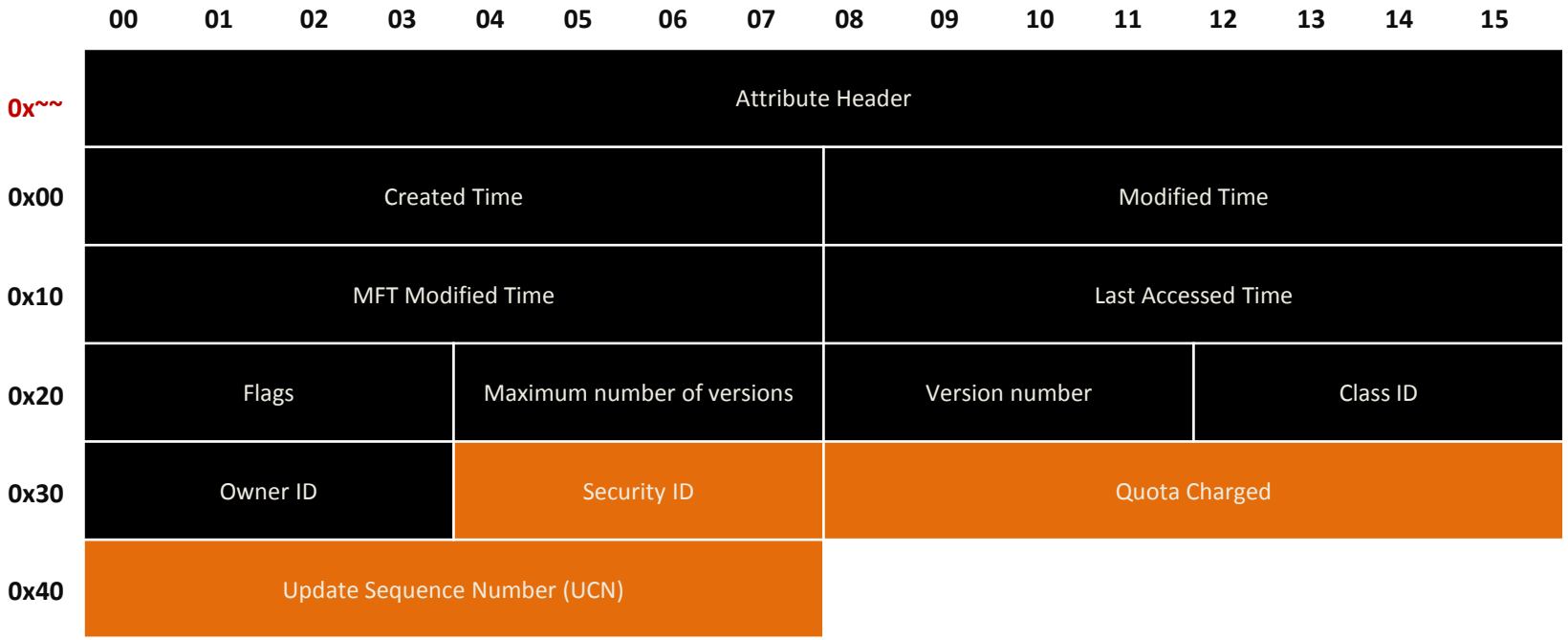
Flags 값	설명
0x0080	일반 (Normal)
0x0100	임시 (Temporary)
0x0200	Sparse 파일
0x0400	Reparse Point
0x0800	압축 (Compressed)
0x1000	오프라인 (Offline)
0x2000	인덱스 되지 않음
0x4000	암호화 (Encrypted)

MFT Entry ➔ Attributes ➔ $STANDARD_INFORMATION



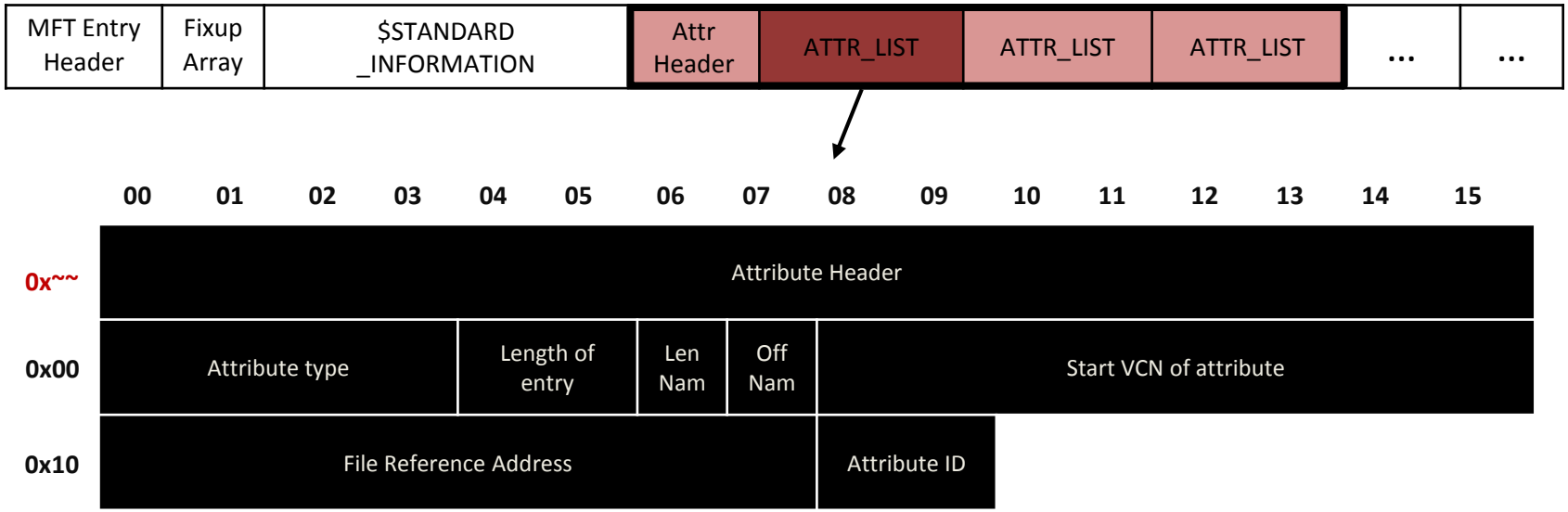
- **Maximum number of version** : 파일의 최대 버전값
- **Version number** : 파일의 버전
- **Class ID** : 인덱스된 클래스 ID
- **Owner ID** : 파일 소유자의 ID 값으로 $Quota 파일에서 인덱스로 사용

MFT Entry ➔ Attributes ➔ $STANDARD_INFORMATION



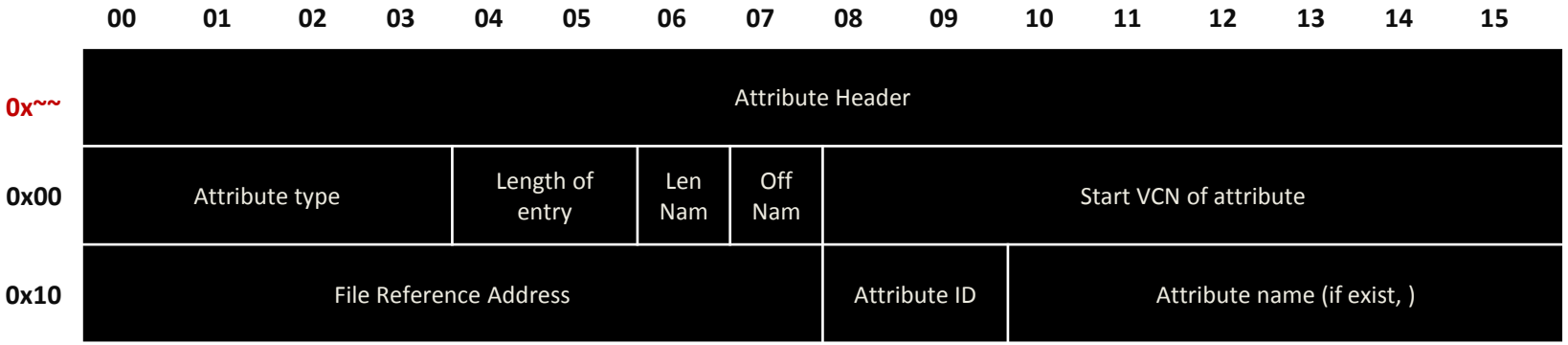
- **Security ID** : $Secure 파일의 인덱스로 사용되고 ACL 적용 시 사용
- **Quota Charged** : 사용자 할당량 중 해당 파일이 할당된 크기 (보통 파일 크기과 동일)
- **USN** : 파일의 USN 값으로 $UsnJrnl에서 인덱스로 사용

MFT Entry ➔ Attributes ➔ $ATTRIBUTE_LIST



- 속성 식별값 32 (0x20)으로 필수적인 속성은 아님
- 특정 파일의 속성이 1,024 바이트를 초과하는 경우 속성의 단편화
- 속성의 빠른 접근을 위해 사용하는 구조
- 여러 개의 속성 리스트 엔트리 사용

MFT Entry ➔ Attributes ➔ $ATTRIBUTE_LIST



- **Attribute type** : 속성 타입
- **Length of entry** : 엔트리 길이
- **Length of Name** : 이름 길이
- **Offset to Name** : 이름 시작 위치
- **Start VCN of attribute** : 속성 시작 VCN
- **File Reference Address** : 속성 위치의 파일 참조 주소
- **Attribute ID** : 속성 ID
- **Attribute name** : 속성 이름이 존재할 경우 속성 이름

MFT Entry ➔ Attributes ➔ $ATTRIBUTE_LIST (example)

- $Secure 메타데이터 파일의 $ATTRIBUTE_LIST 속성

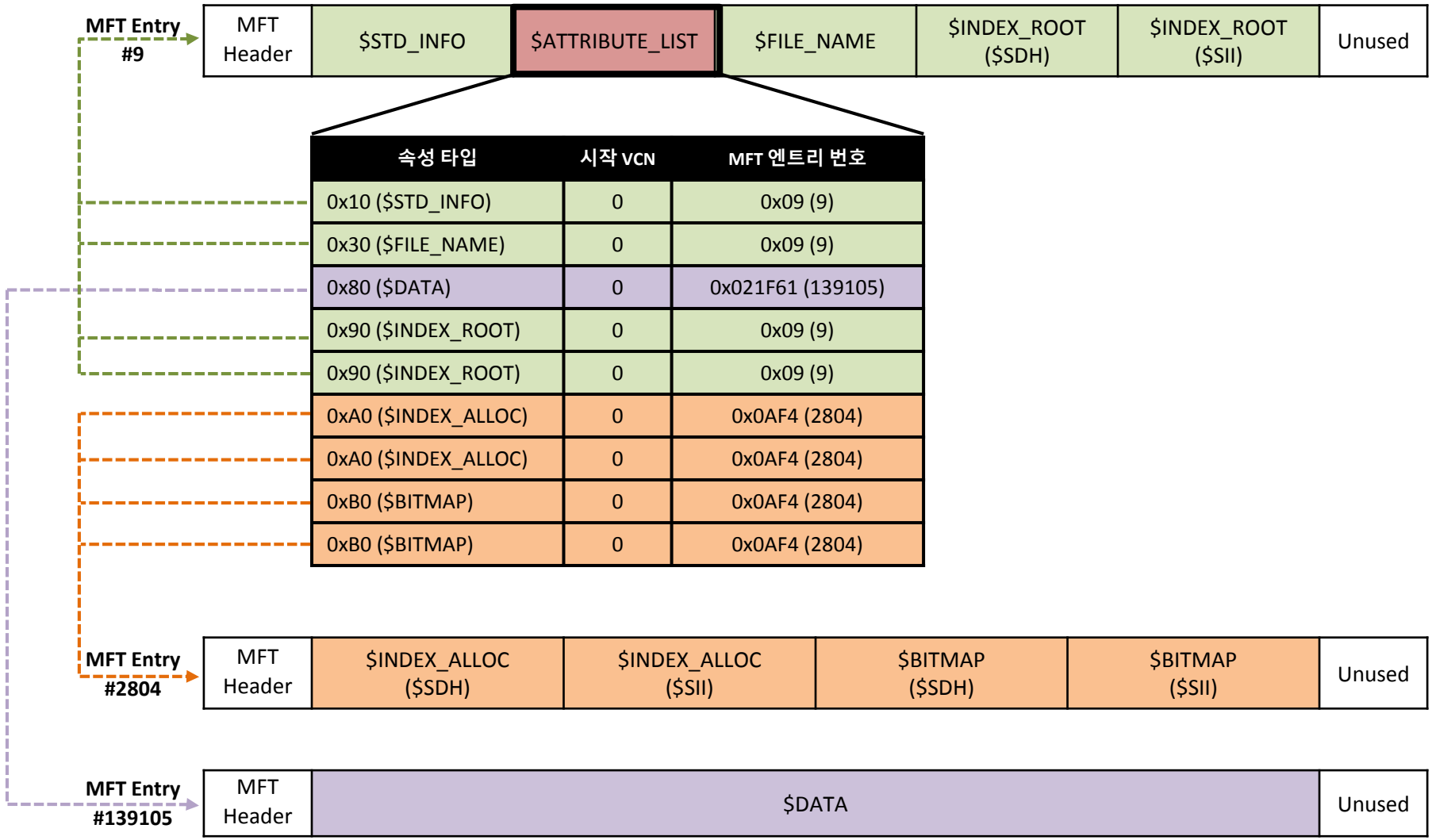
Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A61AA3000	10	00	00	00	20	00	00	1A	00	00	00	00	00	00	00	00
A61AA3010	09	00	00	00	00	00	09	00	00	00	6C	89	00	00	00	00
A61AA3020	30	00	00	00	20	00	00	1A	00	00	00	00	00	00	00	00
A61AA3030	09	00	00	00	00	00	09	00	07	00	00	00	DA	00	DC	00
A61AA3040	80	00	00	00	28	00	04	1A	00	00	00	00	00	00	00	00
A61AA3050	61	1F	02	00	00	00	82	00	00	00	24	00	53	00	44	00
A61AA3060	53	00	00	00	00	00	00	00	90	00	00	00	28	00	04	1A
A61AA3070	00	00	00	00	00	00	00	00	09	00	00	00	00	00	09	00
A61AA3080	10	00	24	00	53	00	44	00	48	00	00	00	00	00	00	00
A61AA3090	90	00	00	00	28	00	04	1A	00	00	00	00	00	00	00	00
A61AA30A0	09	00	00	00	00	00	09	00	11	00	24	00	53	00	49	00
A61AA30B0	49	00	00	00	00	00	00	00	A0	00	00	00	28	00	04	1A
A61AA30C0	00	00	00	00	00	00	00	00	F4	0A	00	00	00	00	01	00
A61AA30D0	02	00	24	00	53	00	44	00	48	00	6F	00	72	00	65	00
A61AA30E0	A0	00	00	00	28	00	04	1A	00	00	00	00	00	00	00	00
A61AA30F0	F4	0A	00	00	00	00	01	00	03	00	24	00	53	00	49	00
A61AA3100	49	00	69	00	61	00	73	00	B0	00	00	00	28	00	04	1A
A61AA3110	00	00	00	00	00	00	00	00	F4	0A	00	00	00	00	01	00
A61AA3120	04	00	24	00	53	00	44	00	48	00	72	00	61	00	6C	00
A61AA3130	B0	00	00	00	28	00	04	1A	00	00	00	00	00	00	00	00
A61AA3140	F4	0A	00	00	00	00	01	00	05	00	24	00	53	00	49	00
A61AA3150	49	00	00	00	6C	D6	F3	8F	00	00	00	00	00	00	00	00

MFT Entry ➔ Attributes ➔ $ATTRIBUTE_LIST (example)

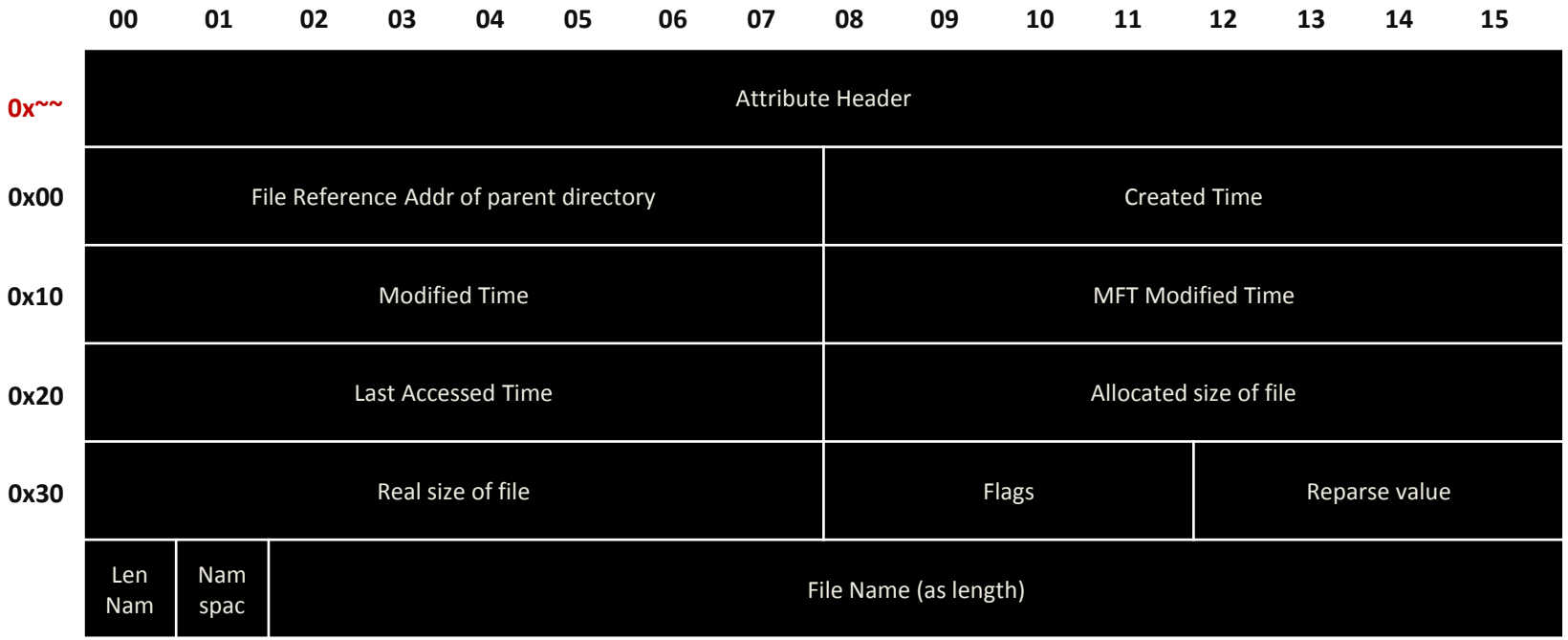
- $Secure 메타데이터 파일의 $ATTRIBUTE_LIST 속성

속성 타입	엔트리 길이	이름 길이	이름 위치	시작 VCN	속성 파일 참조 주소	속성 ID	속성 이름
0x10 ($STD_INFO)	0x0020	0x00	0x1A	0x~00	0x00090000 00000009	0x0000	-
0x30 ($FILE_NAME)	0x0020	0x00	0x1A	0x~00	0x00090000 00000009	0x0007	-
0x80 ($DATA)	0x0028	0x04	0x1A	0x~00	0x00820000 00021F61	0x0000	$SDS
0x90 ($INDEX_ROOT)	0x0028	0x04	0x1A	0x~00	0x00090000 00000009	0x0001	$SDH
0x90 ($INDEX_ROOT)	0x0028	0x04	0x1A	0x~00	0x00090000 00000009	0x0011	$SII
0xA0 ($INDEX_ALLOC)	0x0028	0x04	0x1A	0x~00	0x00010000 00000AF4	0x0002	$SDH
0xA0 ($INDEX_ALLOC)	0x0028	0x04	0x1A	0x~00	0x00010000 00000AF4	0x0003	$SII
0xB0 ($BITMAP)	0x0028	0x04	0x1A	0x~00	0x00010000 00000AF4	0x0004	$SDH
0xB0 ($BITMAP)	0x0028	0x04	0x1A	0x~00	0x00010000 00000AF4	0x0005	$SII

MFT Entry ➔ Attributes ➔ $ATTRIBUTE_LIST (example)

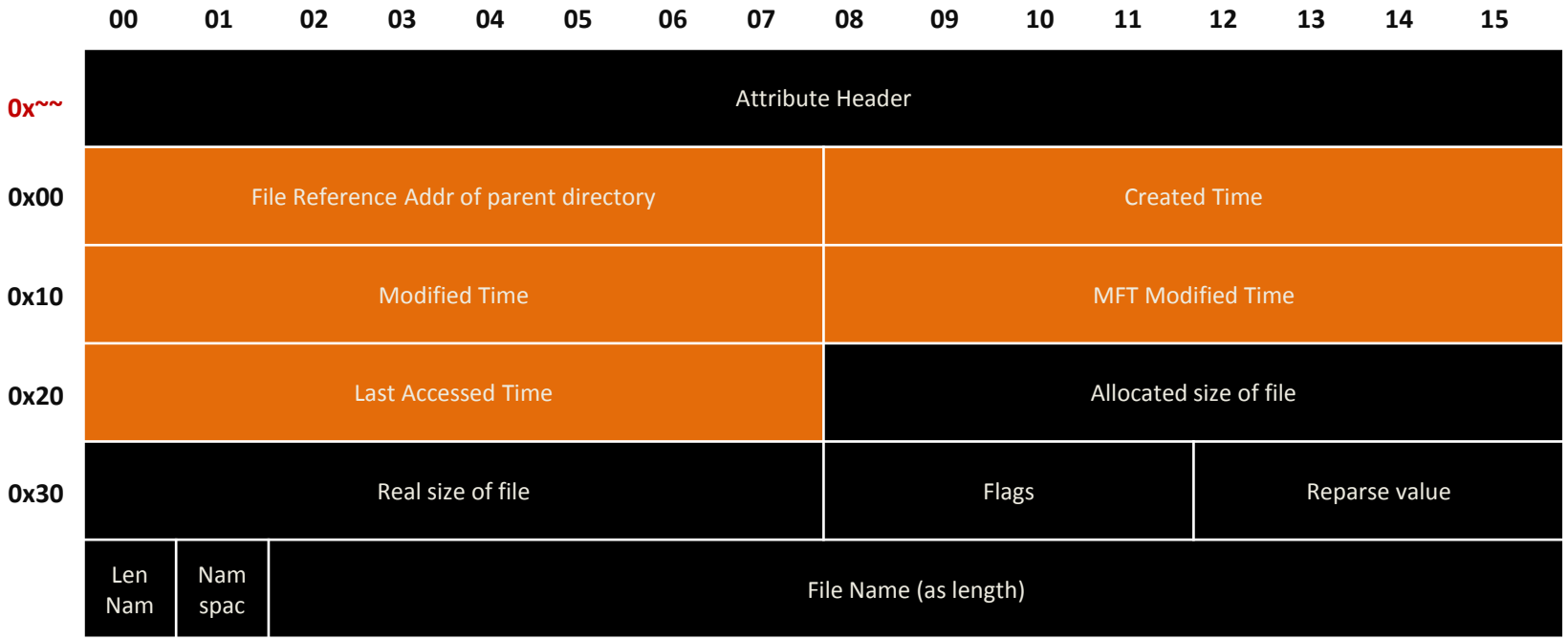


MFT Entry ➔ Attributes ➔ $FILE_NAME



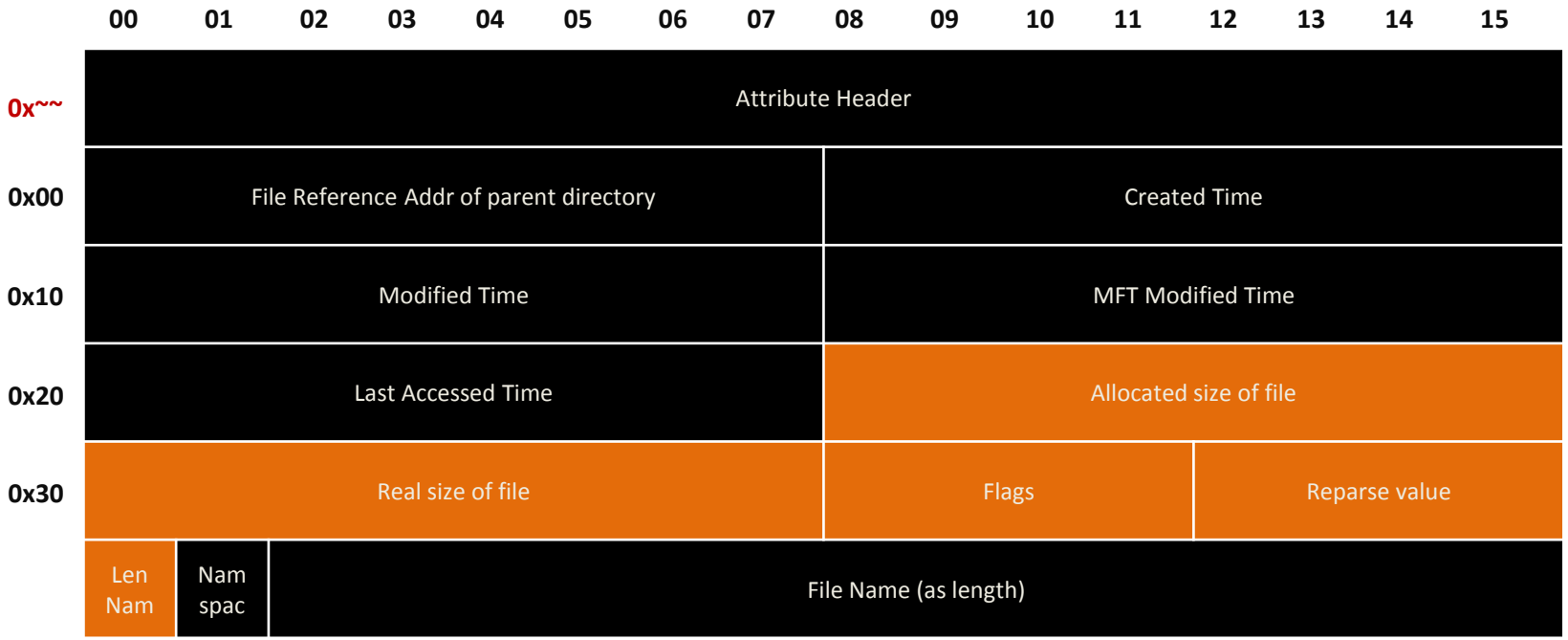
- 속성 식별값 48 (0x30)을 가지는 속성으로 파일의 이름 표현
- 유니코드(UTF-16)으로 인코딩되어 저장
- $FILE_NAME 속성은 MFT Entry 외에 $I30 인덱스(탐색을 위한 구조)에도 저장
- 일반적으로 파일 이름 변경을 제외하고는 $I30 인덱스의 $FILE_NAME 속성만 갱신

MFT Entry ➔ Attributes ➔ $FILE_NAME



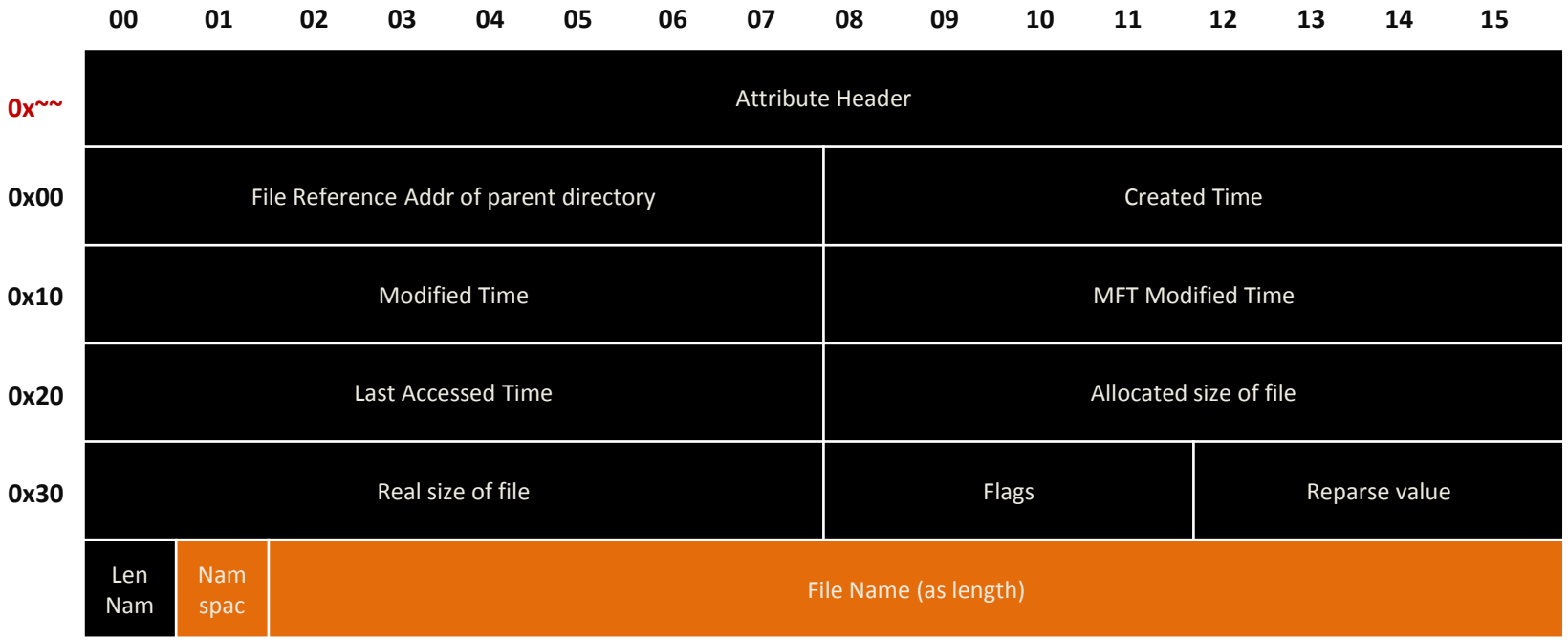
- File Reference Address of parent directory : 부모디렉터리의 파일 참조 주소
- **Windows 64-Bit Timestamp**
  - Created Time
  - Modified Time
  - MFT Modified Time
  - Last Accessed Time

MFT Entry ➔ Attributes ➔ $FILE_NAME



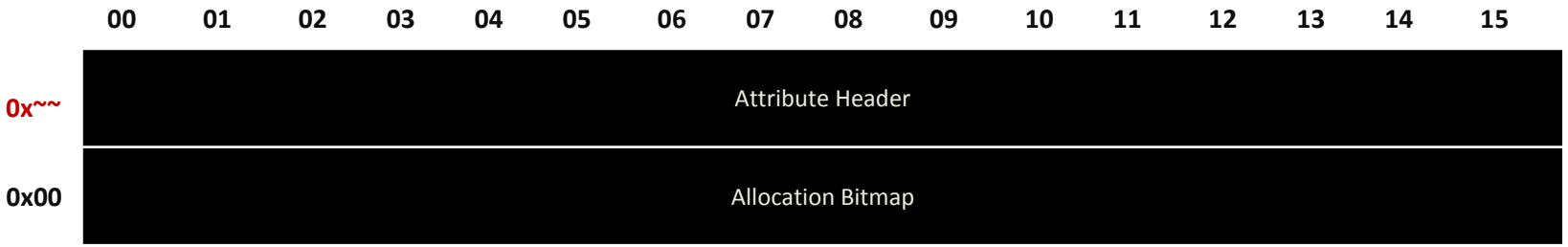
- **Allocated size of file** : 해당 파일이 할당된 크기 (클러스터 배수)
- **Real size of file** : 해당 파일의 실제 크기
- **Flags** : $STANDARD_INFORMATION 속성 플래그와 동일
- **Reparse value** : 해당 속성의 Reparse Point (해당 파일의 마운트, 심볼릭 링크 정보 등을 저장)
- **Length of Name** : 이름 길이

MFT Entry ➔ Attributes ➔ $FILE_NAME

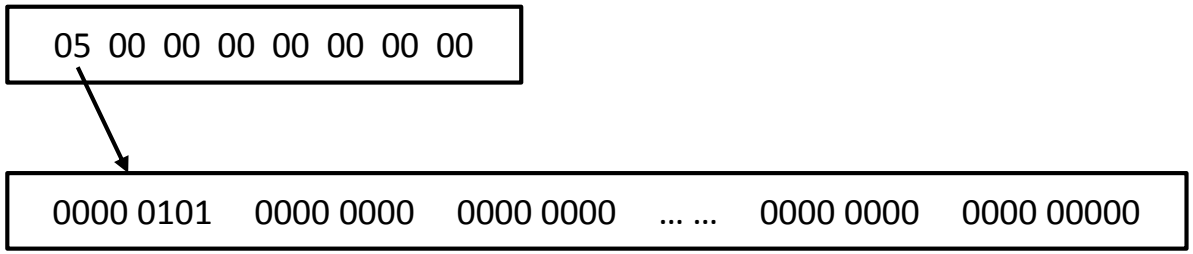


- **Namespace** : 이름의 표현 형식
  - POSIX, Win32, DOS, Win32&Dos
  - http://msdn.microsoft.com/en-us/library/aa365247%28VS.85%29.aspx
- **File Name** : 유니코드(UTF-16)으로 인코딩된 파일 이름

MFT Entry ➔ Attributes ➔ $BITMAP

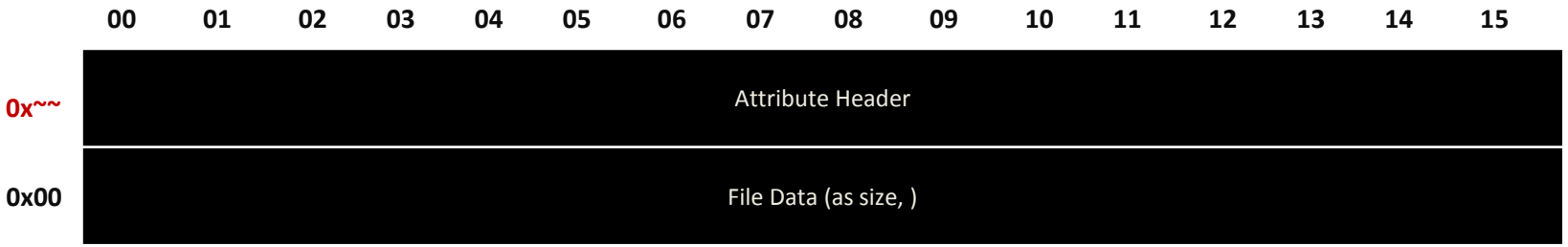


- 속성 식별값 176 (0xB0)을 가지는 속성으로 할당 정보 표현
- 할당 정보를 관리하는 데이터 ➔ **$MFT, $INDEX_ALLOCATION**



- 1 번째와 3 번째 MFT Entry가 사용 중임을 나타냄

MFT Entry ➔ Attributes ➔ $DATA



- 속성 식별값 180 (0x80)을 가지는 속성으로 파일 데이터 저장
- 속성 헤더 이후에 바로 속성 내용인 파일 데이터 스트림 위치
- 데이터의 크기에 따라 Resident 혹은 Non-resident 로 존재
- 대체 데이터 스트림(Alternate Data Stream)을 통해 2개 이상의 데이터 스트림 표현

MFT Entry ➔ Alternate Data Stream (ADS)

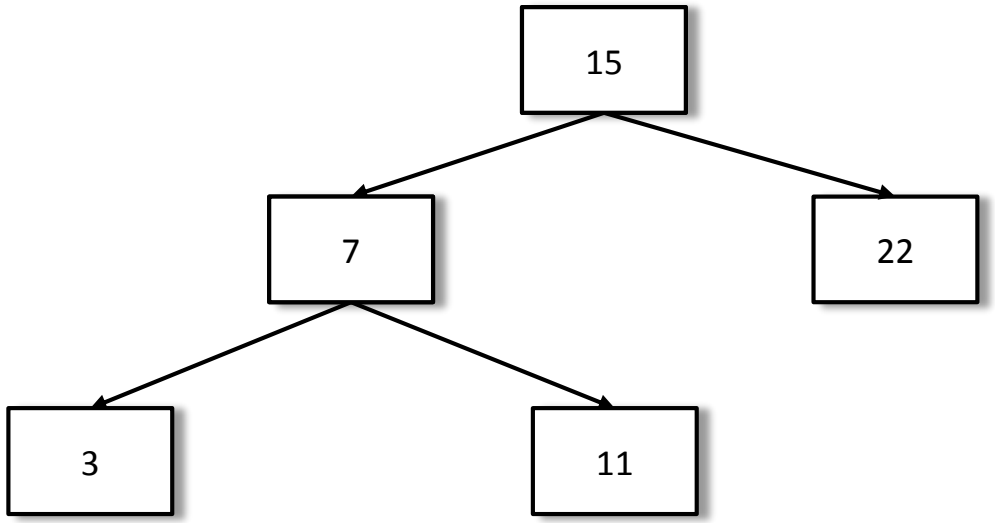
- $DATA 속성을 2 개 이상의 속성 스트림 지원 (윈도우 NT 3.1 버전부터 지원)
- NT(NTFS) 서버에서 Mac OS(HFS) 클라이언트를 지원하기 위해 추가
- 추가적인 속성 스트림(ADS)은 속성 이름을 통해 접근
- 결국, ADS는 반드시 속성 이름을 가지고 있어야 함 ➔ 기본 스트림은 없어도 무관
- ADS 속성은 파일 크기에 포함되지 않음
- ADS 용도
  - 파일의 요약 정보 저장
  - 영역 식별자(Zone Identifier)

MFT Entry ➔ Alternate Data Stream (ADS)

- **ADS 생성**
  - **~₩>** notepad proneer.txt:ads.txt
  - **~₩>** echo "This is contained ADS" > proneer.txt:ads2.txt
  - **~₩>** type c:₩windows₩system.ini > proneer.txt:ads3.ini
  - **~₩>** echo "This is attached to directory list" > :ads4.txt
- **ADS 확인**
  - 별도의 도구를 이용
  - **~₩>** more < proneer.txt:ads3.ini
- **ADS 삭제**
  - ADS가 존재하는 파일 삭제, 기본 스트림 복사 후 삭제
  - ADS를 지원하지 않는 파일시스템으로 복사
  - 별도의 도구를 이용하거나 무의미한 데이터로 대체
    - **~₩>** echo " " > proneer.txt:ads3.ini

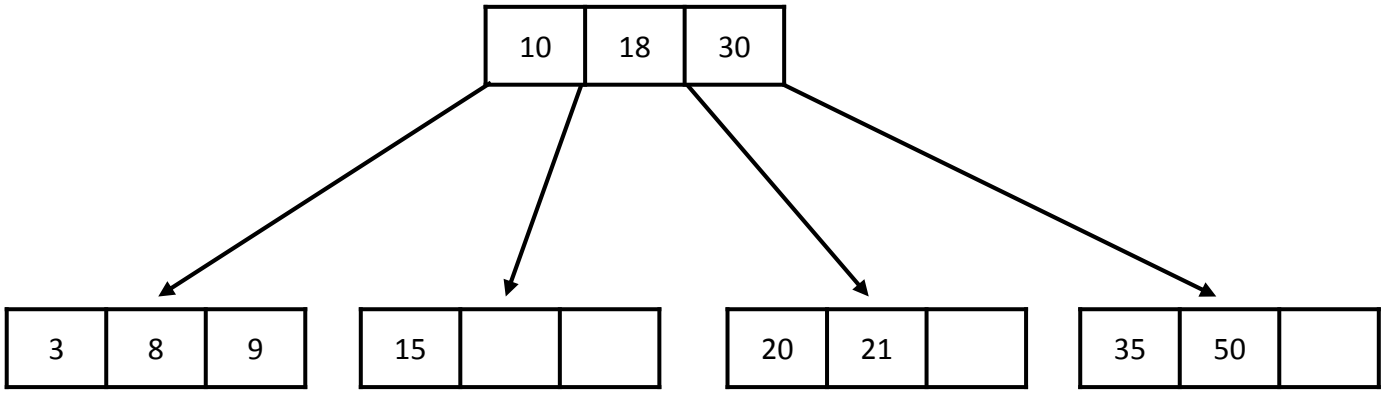
MFT Entry ➔ Indexes

- **Binary Tree (이진 트리)**
  - 한 노드가 최대 2개의 자식 노드를 갖는 트리
  - 노드의 왼쪽 서브트리의 모든 노드값은 키값보다 작음
  - 노드의 오른쪽 서브트리의 모든 노드값은 키값보다 큼



MFT Entry ➔ Indexes

- **B-Tree (B 트리)**
  - 차수가 m인 m-원 탐색트리
  - 루트와 단말 노드를 제외한 각 노드는 최소 m/2의 서브 트리를 가짐 (절반 이상이 채워져야 함)
  - 루트는 최소 2개의 서브 트리를 가짐
  - 모든 단말 노드는 같은 레벨



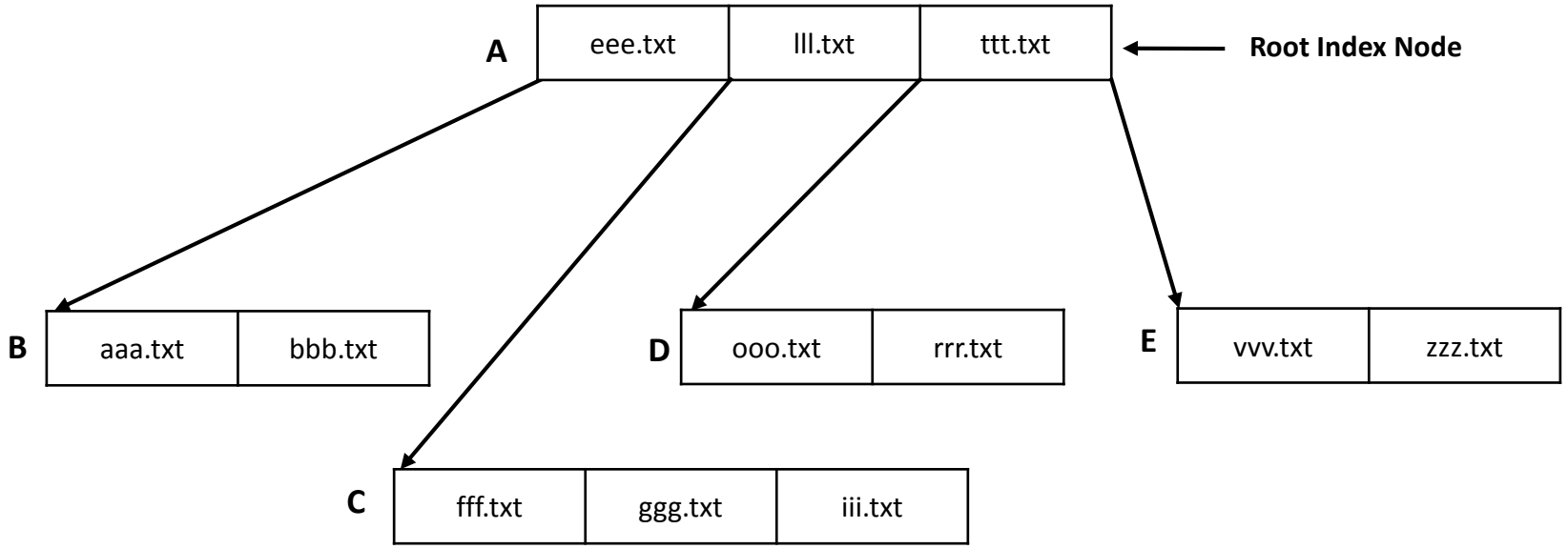
MFT Entry ➔ Indexes

- **인덱스 구조** : 빠르게 검색이 필요한 데이터는 인덱스 구조로 관리 (디렉터리 등)

인덱스 이름	인덱싱하는 데이터	위치
$I30	$FILE_NAME 속성	디렉터리의 MFT Entry
$SDH	Security Descriptors	$Secure 메타데이터 파일
$SII	Security ID	$Secure 메타데이터 파일
$O	Object ID	$ObjId 메타데이터 파일
$O	Owner ID	$Quota 메타데이터 파일
$Q	Quota	$Quota 메타데이터 파일

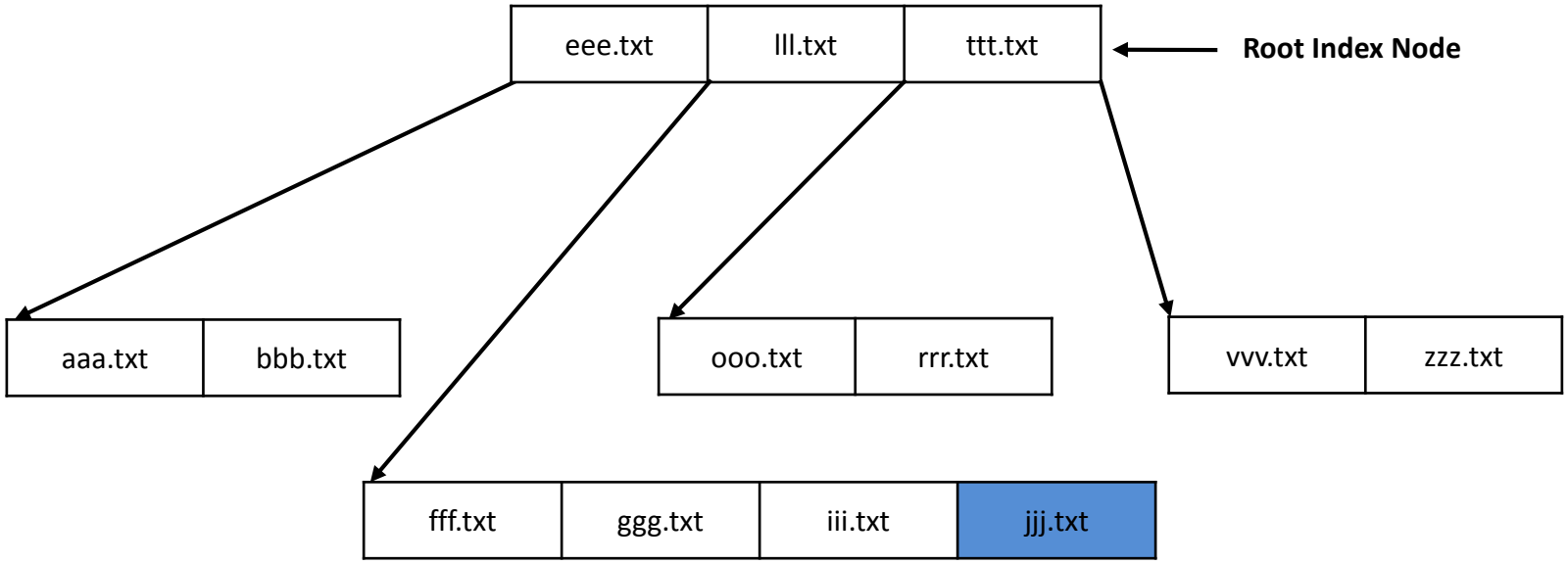
MFT Entry ➔ Indexes

- **B-Tree($I30)** : B 트리의 노드 값은 파일 이름 ($FILE_NAME 속성)



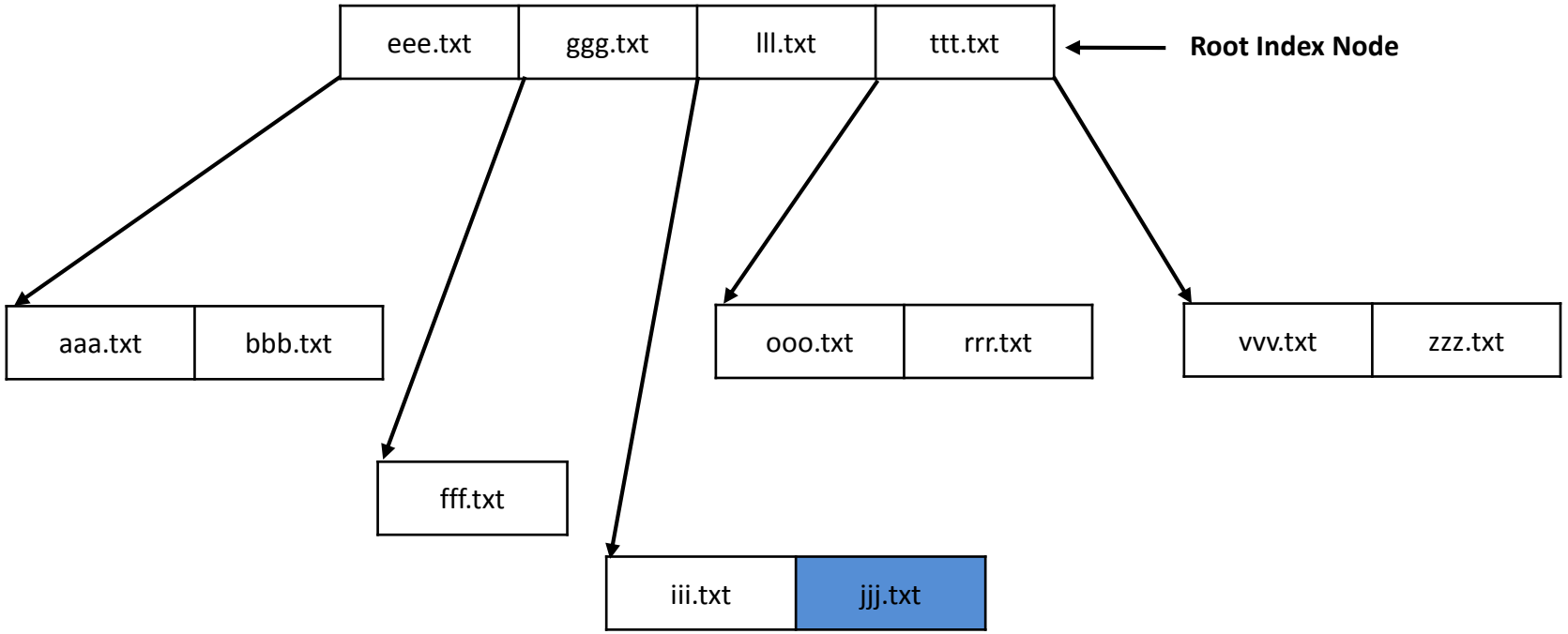
MFT Entry ➔ Indexes

- **B-Tree($I30)** : "jjj.txt" 파일 삽입 (1)



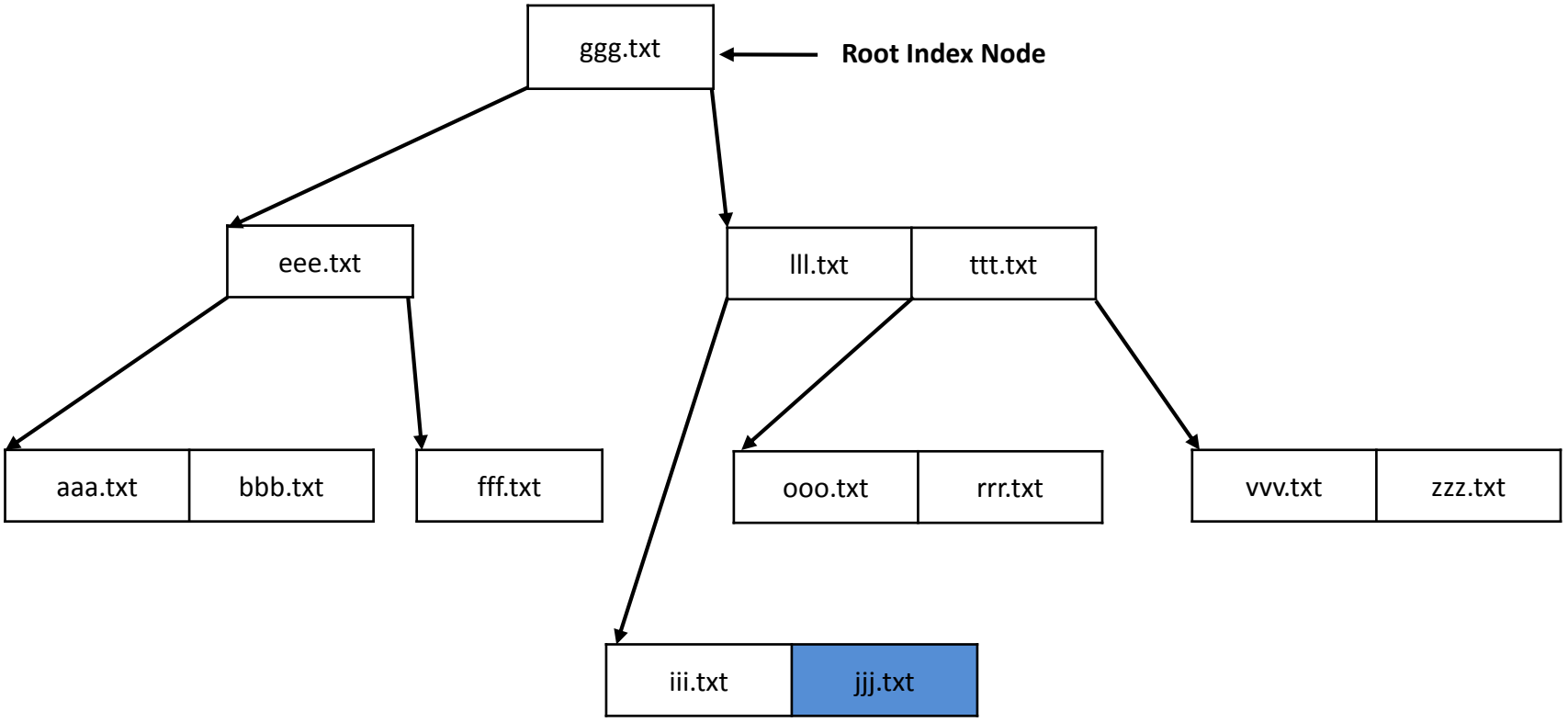
MFT Entry ➔ Indexes

- **B-Tree($I30)** : "jjj.txt" 파일 삽입 (2)



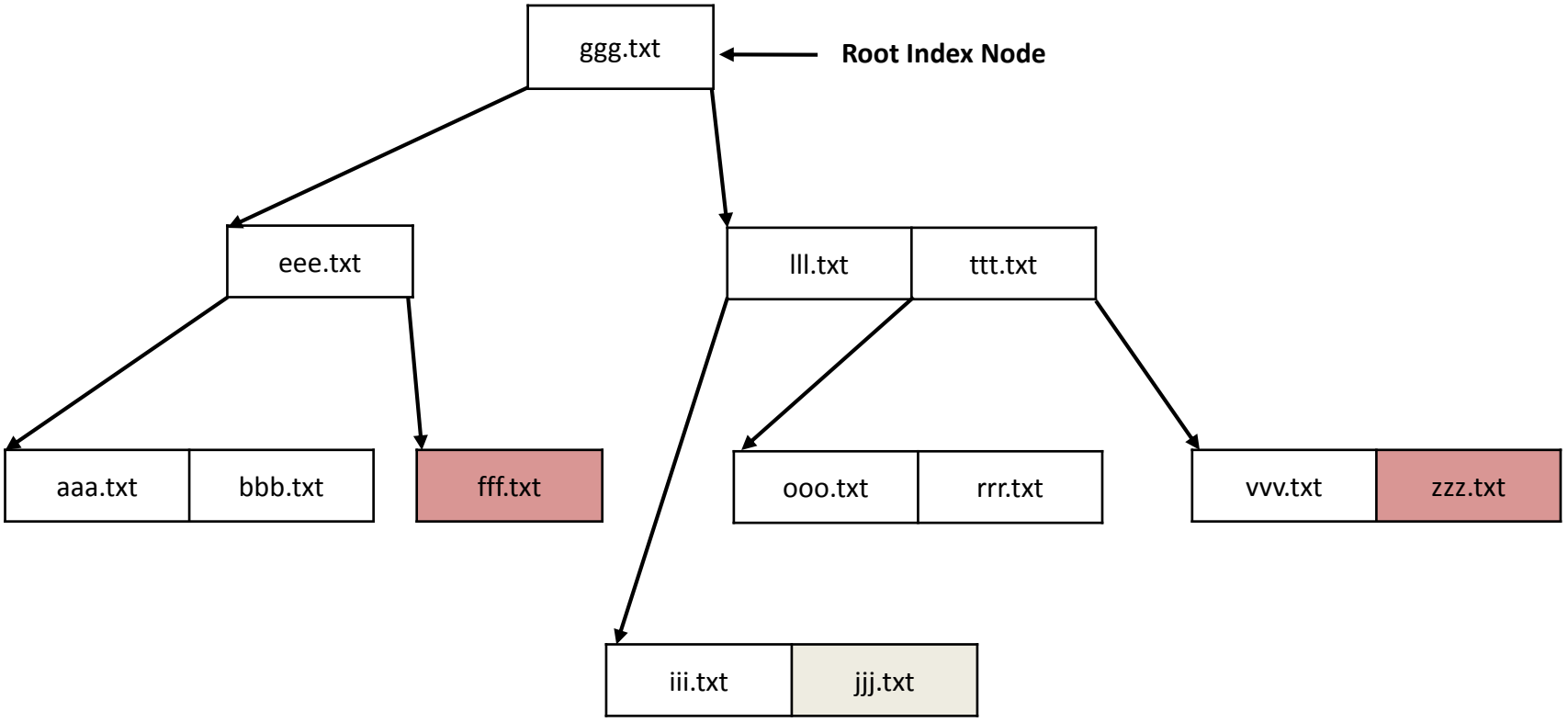
MFT Entry ➔ Indexes

- **B-Tree($I30)** : "jjj.txt" 파일 삽입 (3)



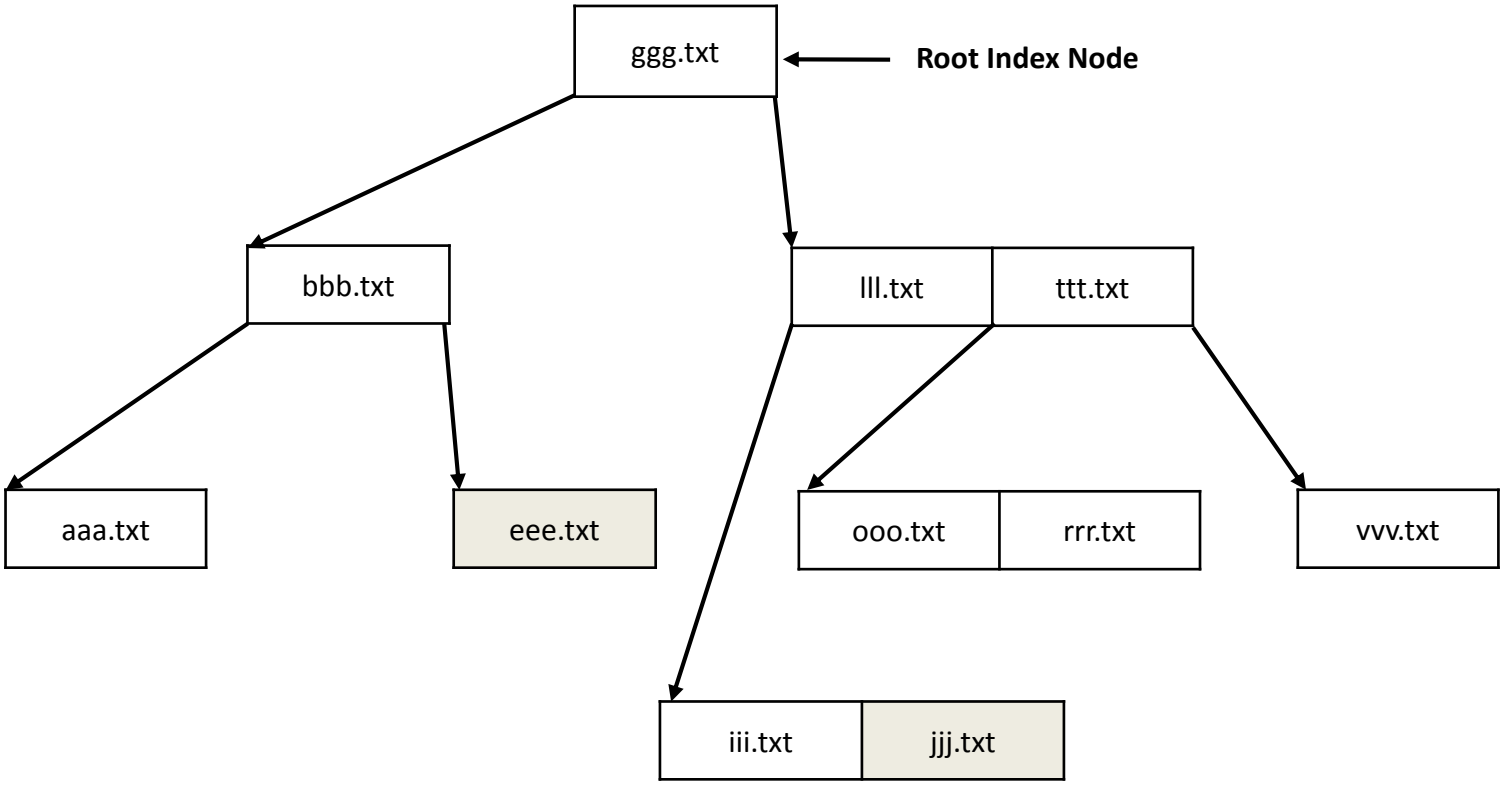
MFT Entry ➔ Indexes

- **B-Tree($I30)** : "fff.txt"와 "zzz.txt" 파일 삭제 (1)



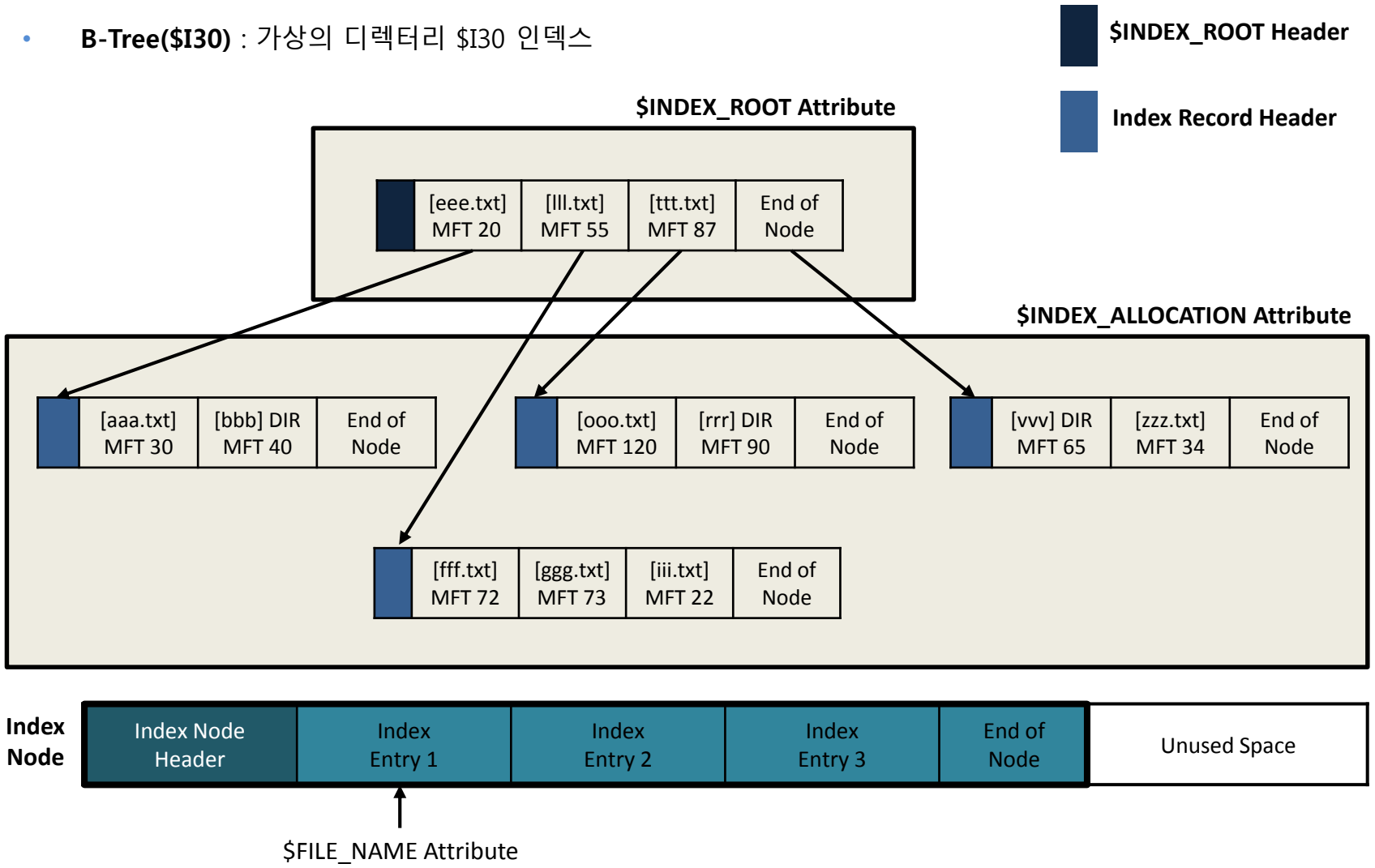
MFT Entry ➔ Indexes

- **B-Tree($I30)** : "fff.txt"와 "zzz.txt" 파일 삭제 (2)



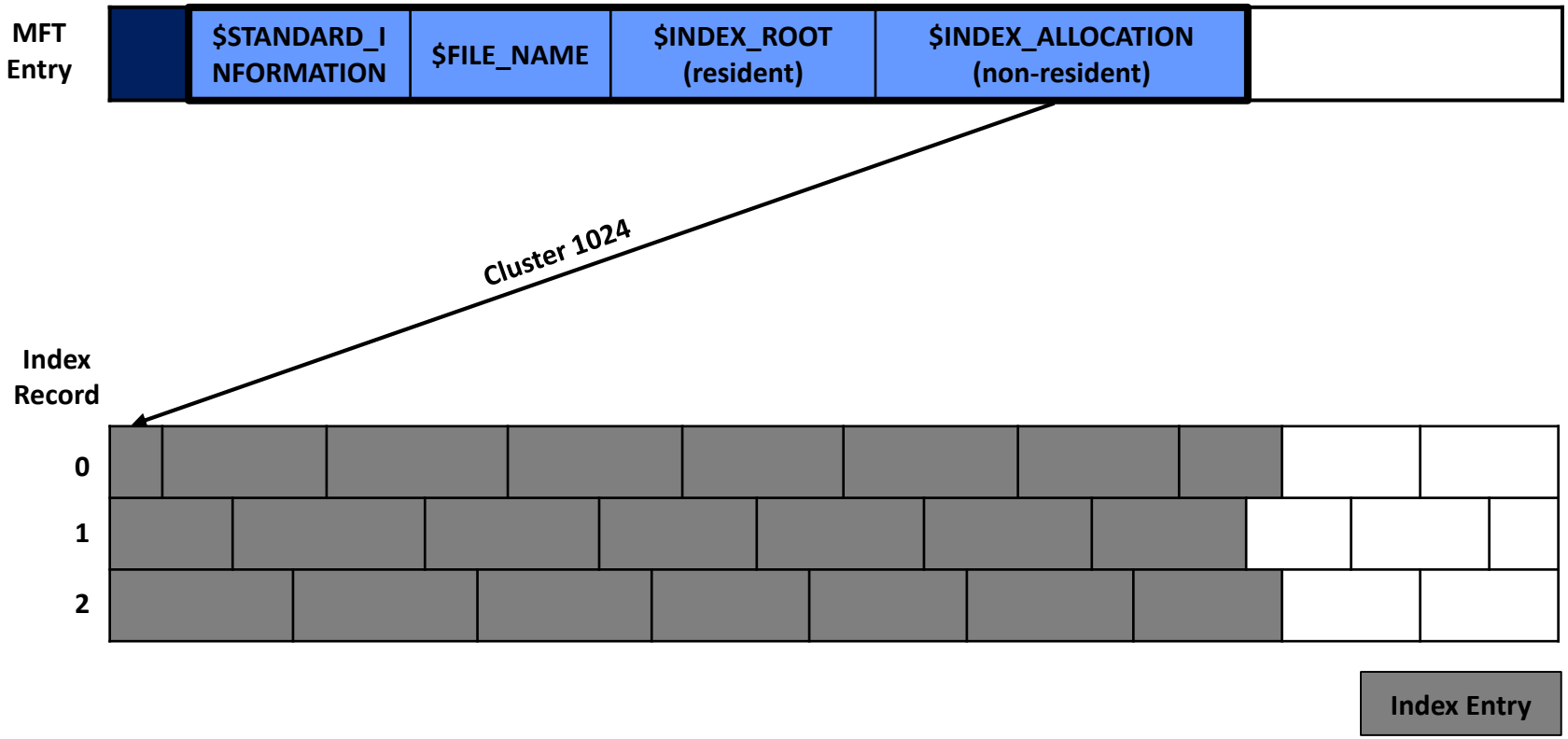
MFT Entry ➔ Indexes

- **B-Tree($I30)** : 가상의 디렉터리 $I30 인덱스

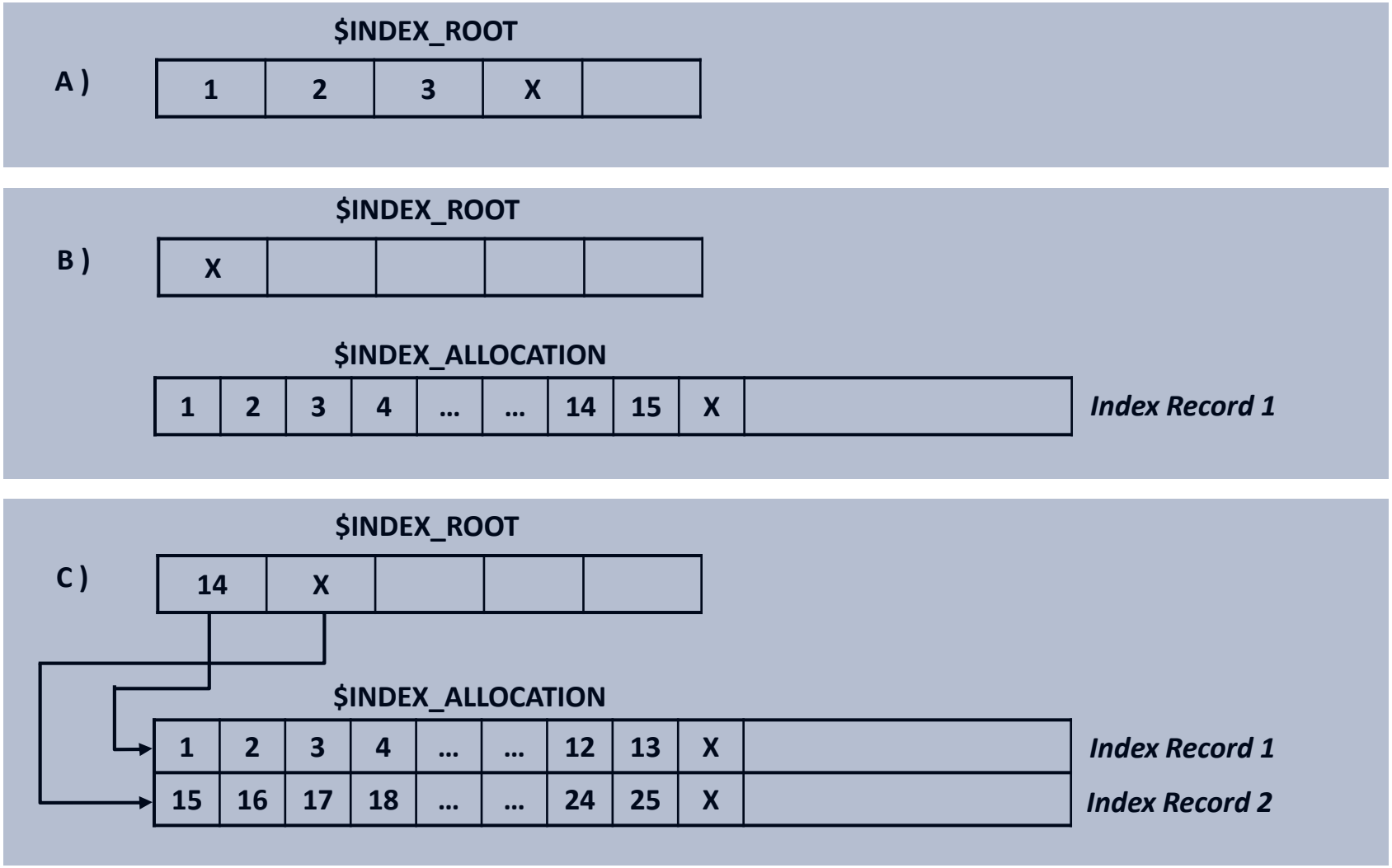


MFT Entry ➔ Indexes

- $INDEX_ROOT는 항상 Resident 속성
- 인덱스 항목이 많아지면 $INDEX_ALLOCATION 속성을 이용해 Non-resident로 구성

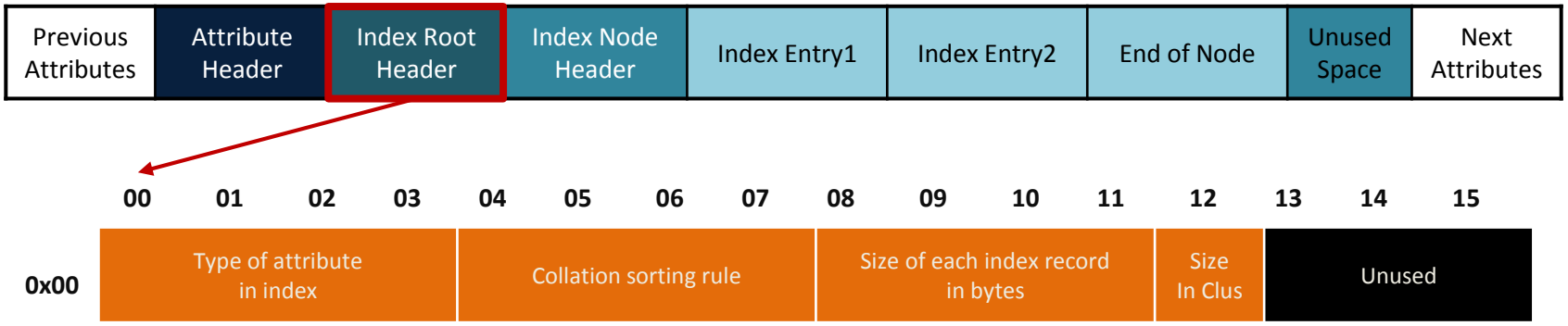


MFT Entry ➔ Indexes



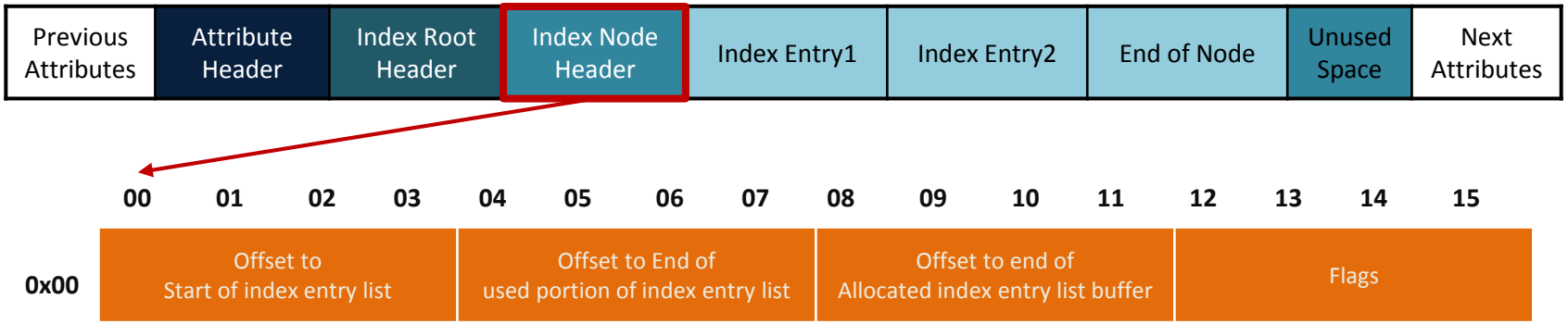
MFT Entry ➔ Attributes ➔ $INDEX_ROOT

MFT Entry ➔ Attributes ➔ $INDEX_ROOT



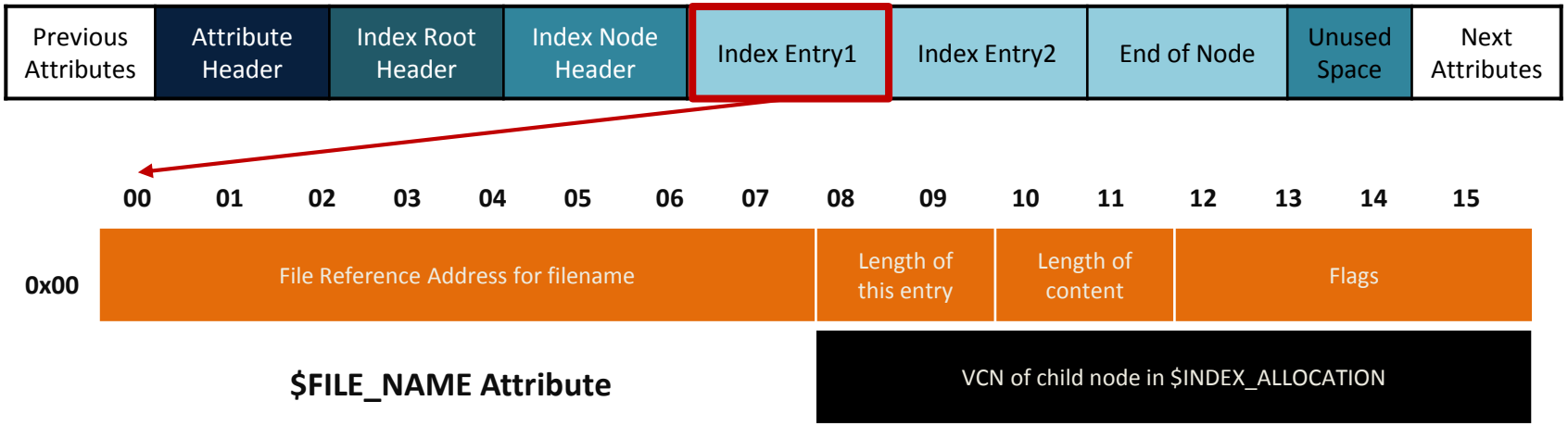
- **Type of attribute in index** : 인덱스 엔트리가 담고 있는 속성 식별값 (디렉터리의 경우 0x30, $FILE_NAME)
- **Collation sorting rule** : 인덱스 엔트리가 담고 있는 형식 (형식에 맞게 정렬됨)
- **Size of each index record in bytes** : $INDEX_ALLOCATION 속성이 가지는 인덱스 레코드의 바이트 크기
- **Size of each index record in Clusters** : $INDEX_ALLOCATION 속성이 가지는 인덱스 레코드의 클러스터 크기

MFT Entry ➔ Attributes ➔ $INDEX_ROOT



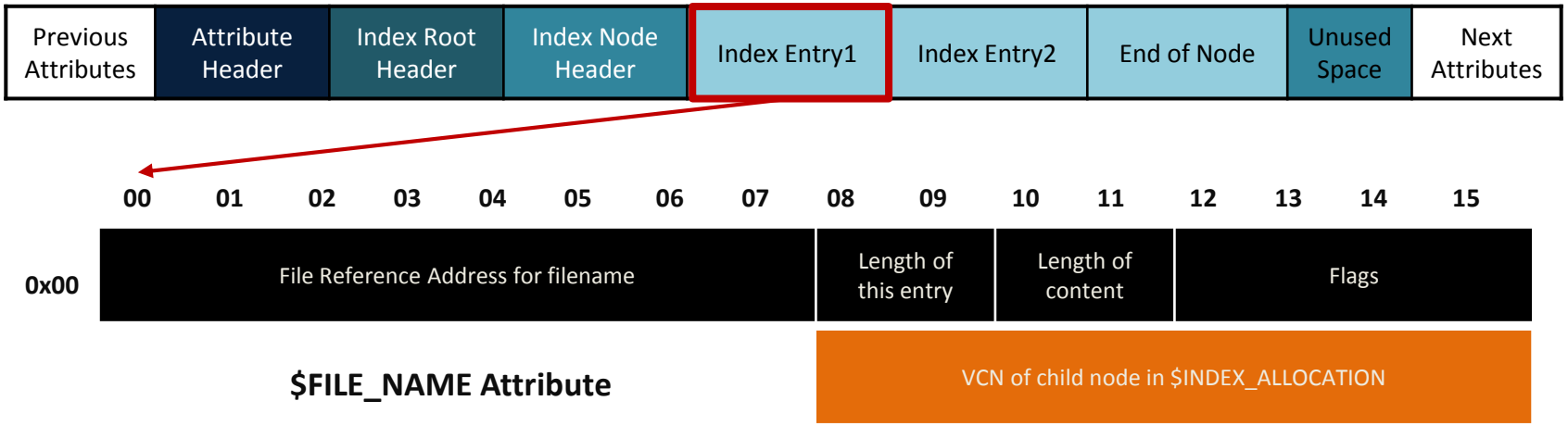
- **Offset to Start of index entry list** : 인덱스 엔트리 목록의 시작 위치
- **Offset to End of used portion of index entry list** : 인덱스 엔트리의 실제 크기 (인덱스 노드 헤더 포함)
- **Offset to end of Allocated index entry list buffer** : 인덱스 엔트리의 할당 크기 (인덱스 노드 헤더 포함)
- **Flags**
  - **0x00** : 인덱스 노드의 자식 노드가 없음
  - **0x01** : 인덱스 노드의 자식 노드가 있음

MFT Entry ➔ Attributes ➔ $INDEX_ROOT



- **File Reference Address for filename** : 해당 파일 및 디렉터리의 파일 참조 주소
- **Length of this entry** : 해당 인덱스 엔트리의 총 크기
- **Length of content** : 해당 인덱스 엔트리가 담고 있는 $FINE_NAME 속성의 크기
- **Flags**
  - **0x01** : 자식 노드가 존재
  - **0x02** : 노드의 마지막 엔트리

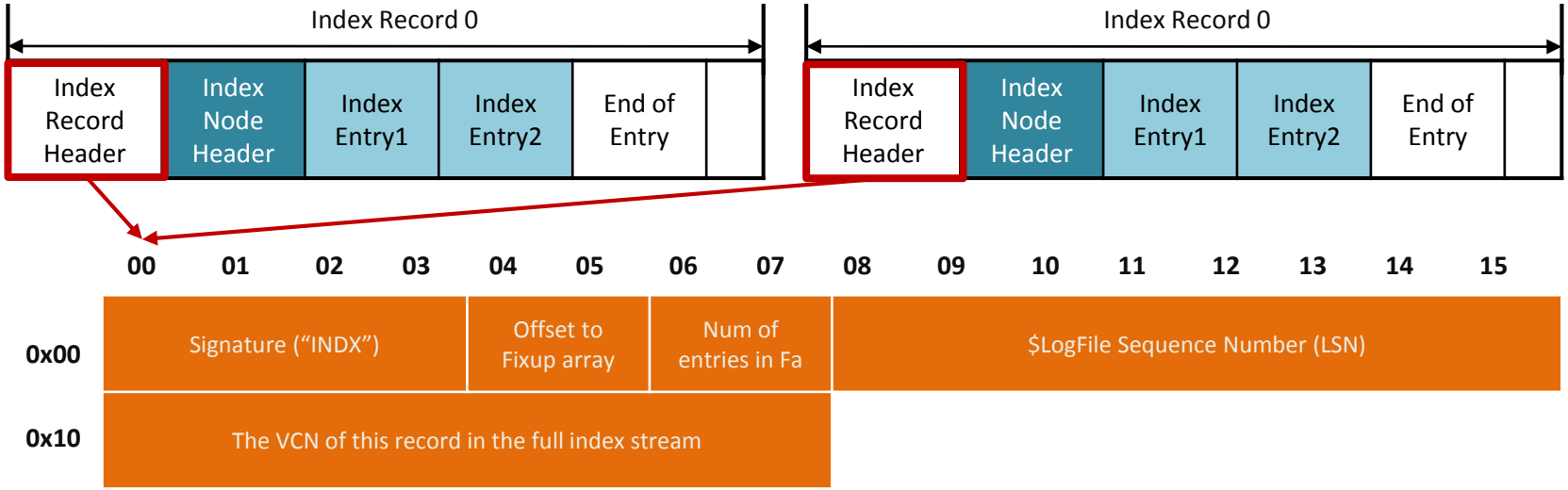
MFT Entry ➔ Attributes ➔ $INDEX_ROOT



- **VCN of child node in $INDEX_ALLOCATION** :

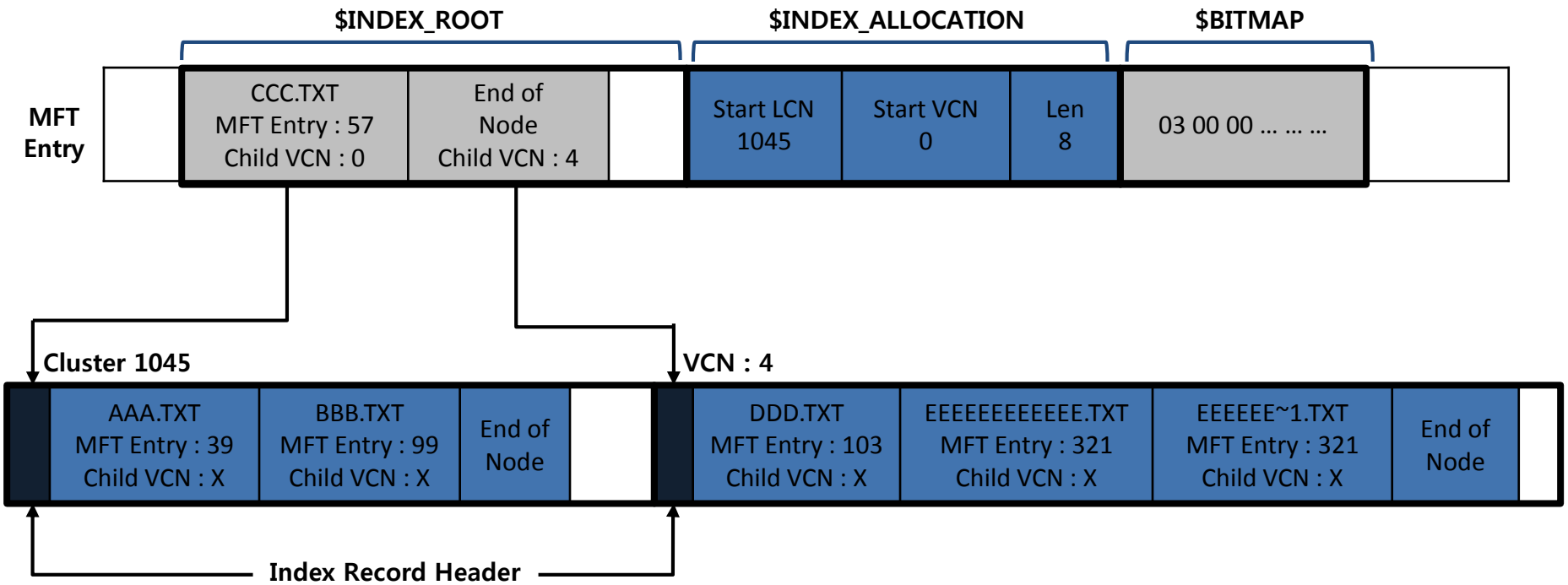
  해당 인덱스 엔트리가 자식 노드를 가지는 경우 $INDEX_ALLOCATION 속성에 위치한 자식 인덱스 노드의 위치

MFT Entry ➔ Attributes ➔ $INDEX_ALLOCATION



- **Signature :** $INDEX_ALLOCATION의 시그니처 (“INDX”)
- **Offset to Fixup array :** Fixup array의 위치
- **Number of entries in Fixup array :** Fixup array에 저장된 항목의 수
- **$LogFile Sequence Number (LSN) :** $LogFile에 존재하는 해당 파일의 트랜잭션 위치 값
- **The VCN of this record in the full index stream :** $INDEX_ALLOCATION속성에서 해당 인덱스 레코드가 저장된 위치

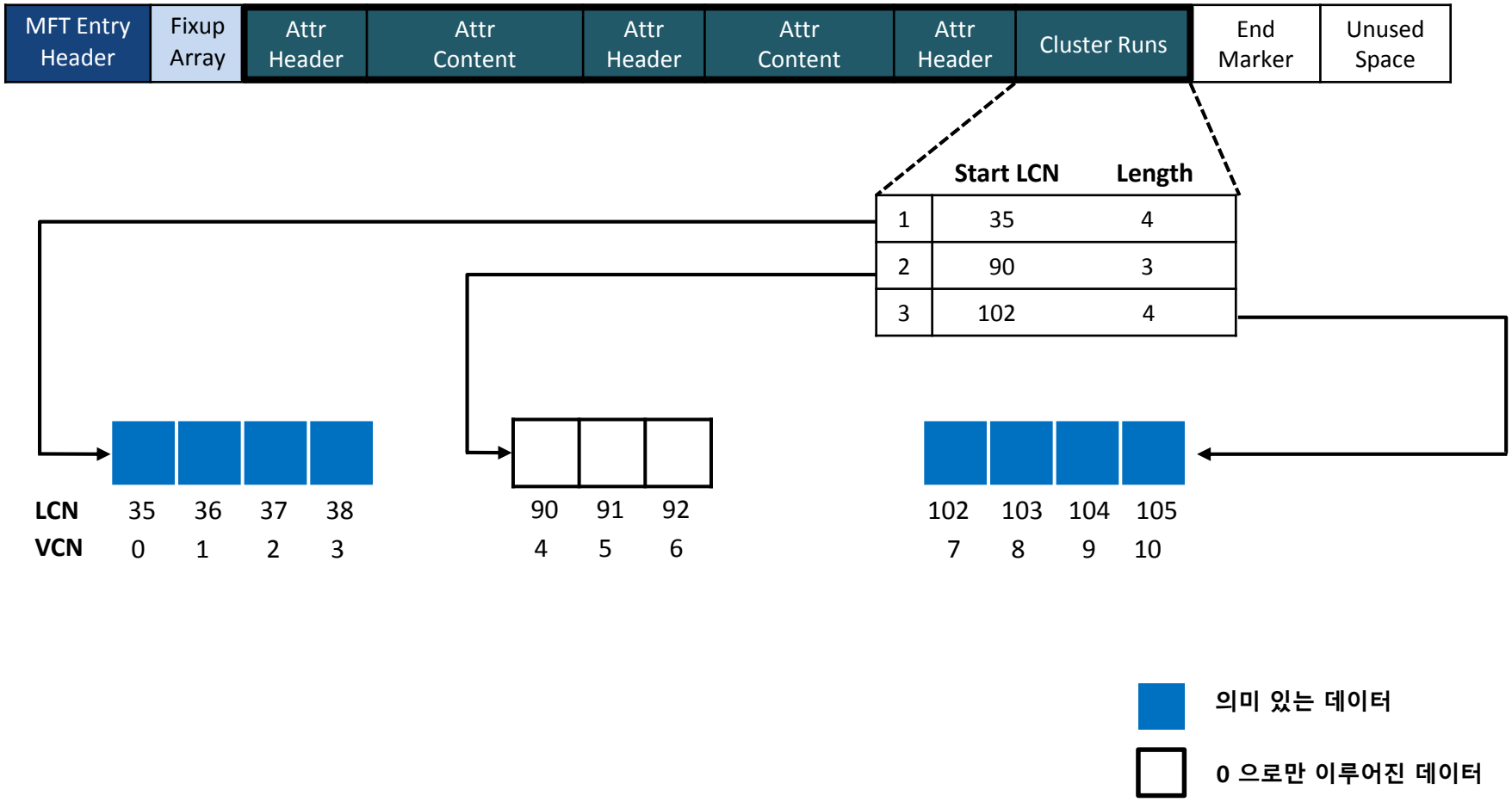
MFT Entry ➔ Attributes ➔ Index Structure



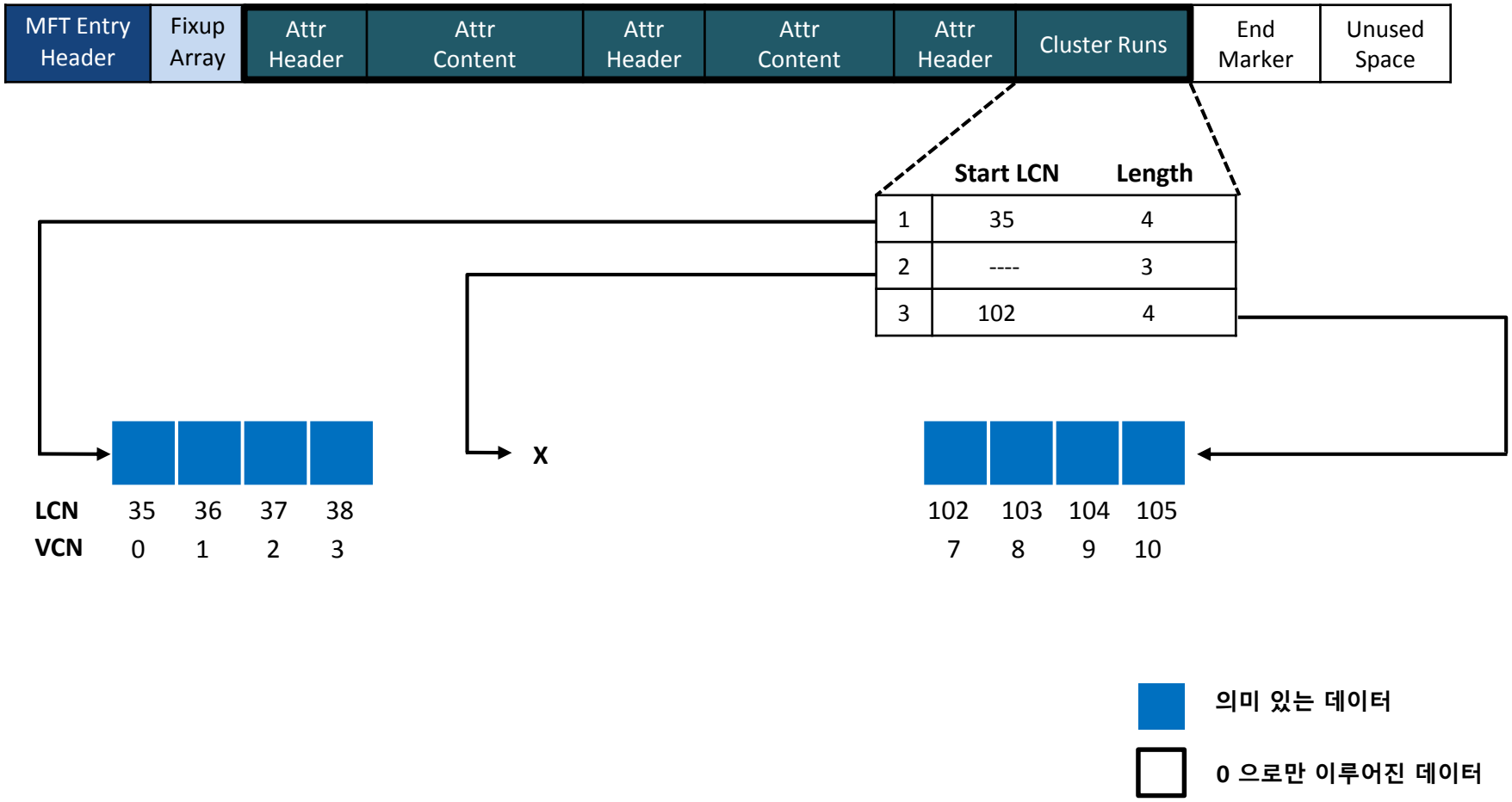
NTFS Features

Security is a people problem…

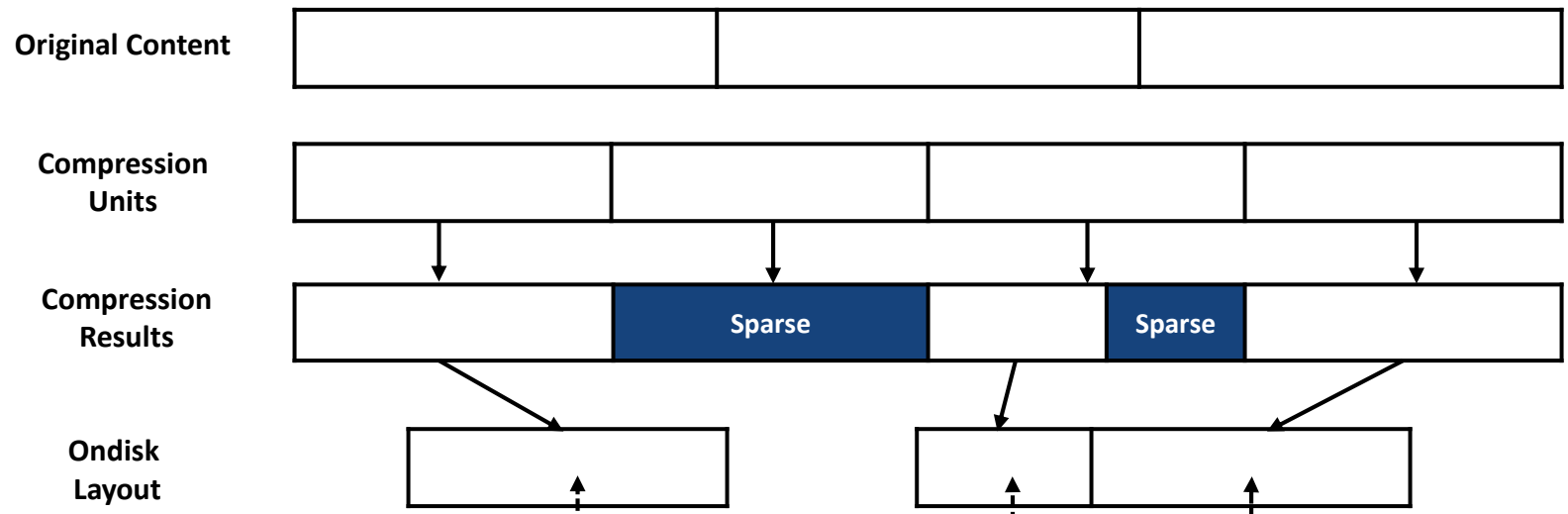
Sparse File (applied only $DATA, Non-resident) cont.



Sparse File (applied only $DATA, Non-resident)



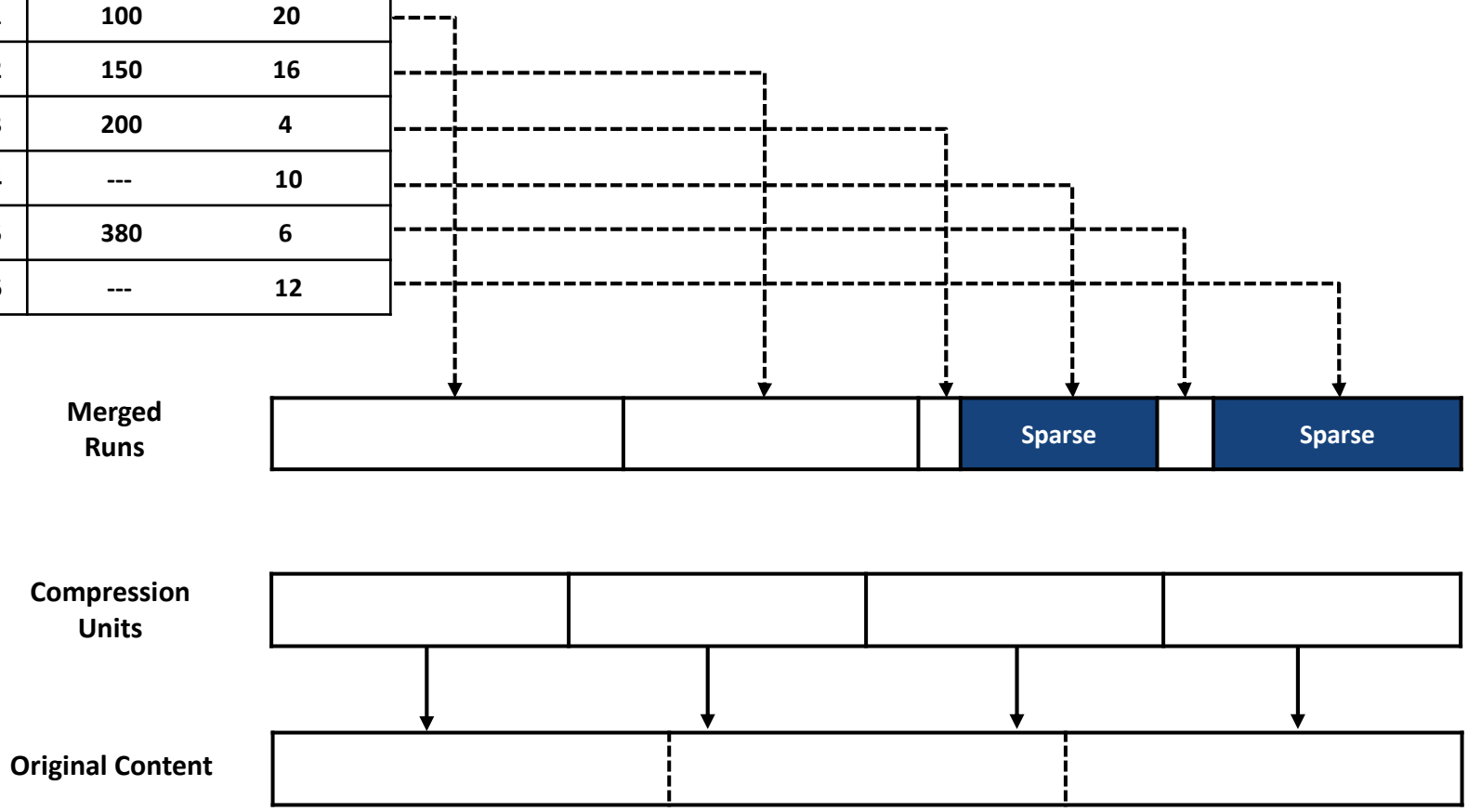
Compression (applied only $DATA, Non-resident)



	Start LCN	Length
1	100	16
2	----	16
3	150	10
4	----	6
5	160	16

Decompression (applied only $DATA, Non-resident)

	Start LCN	Length
1	100	20
2	150	16
3	200	4
4	---	10
5	380	6
6	---	12

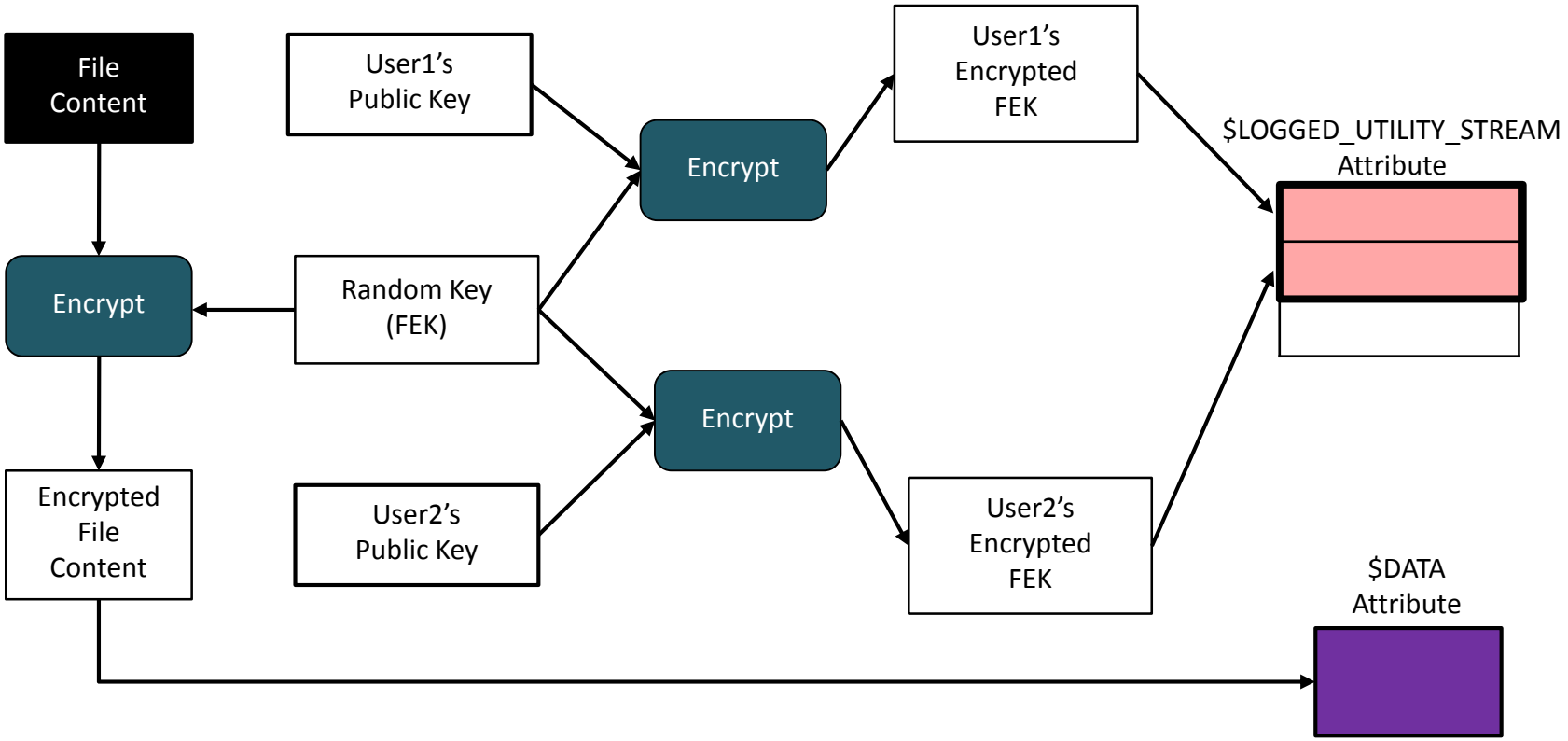


NTFS Features

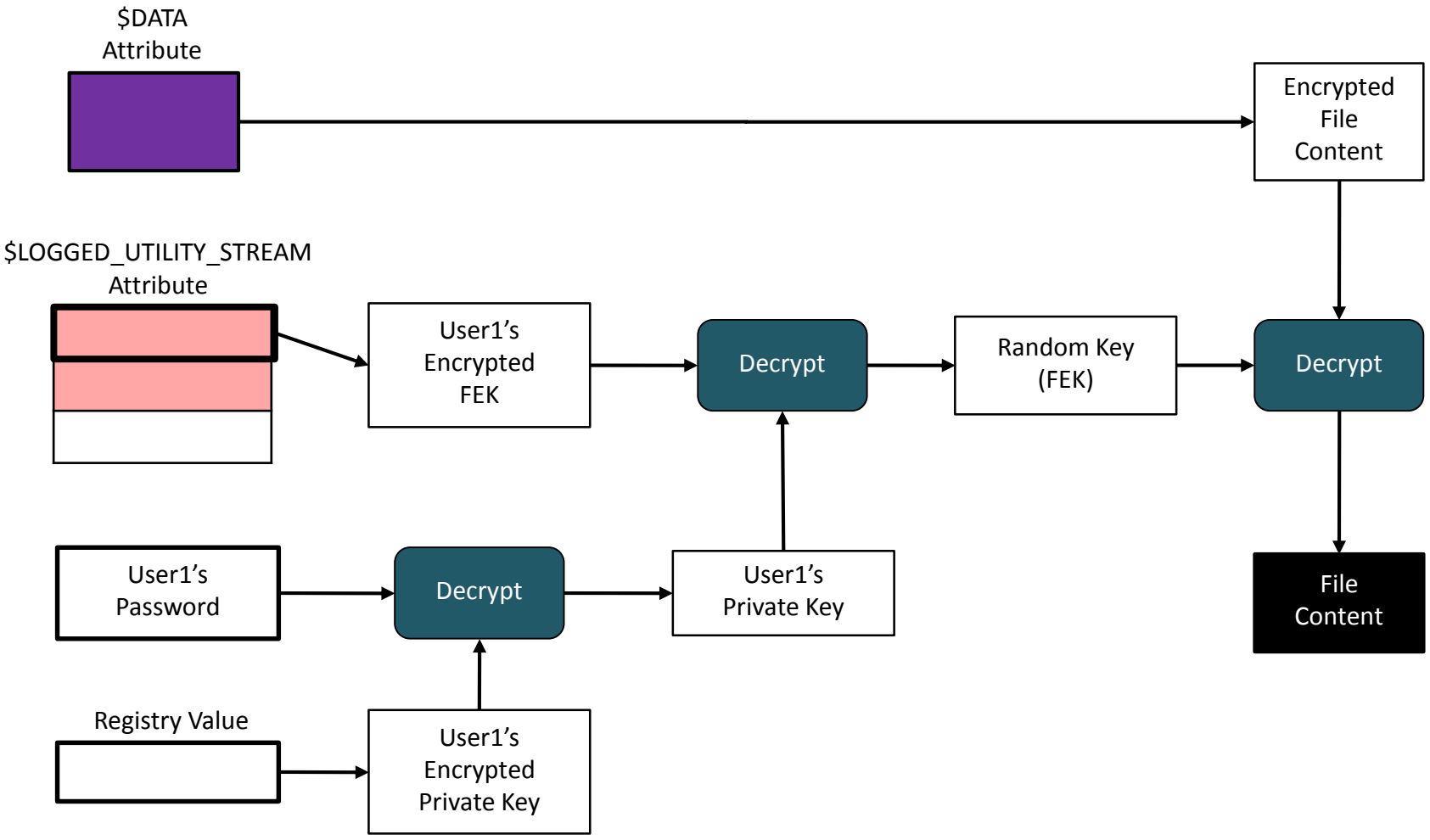
DESX	Win 2000
Triple DES	Win XP
AES	Win XP SP1, 2003

Encryption (applied only $DATA)

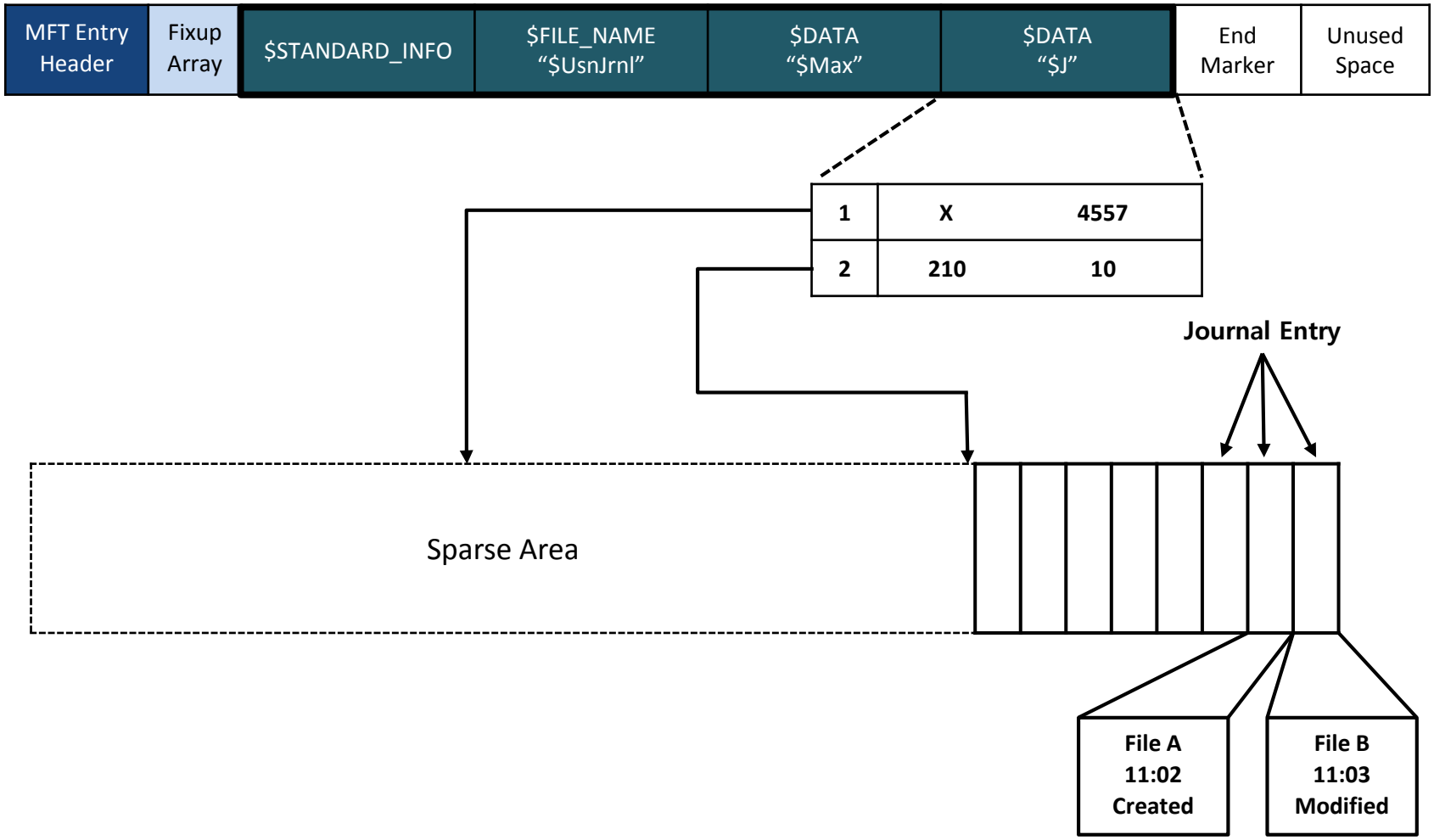
- **취약점** : EFS0.TMP



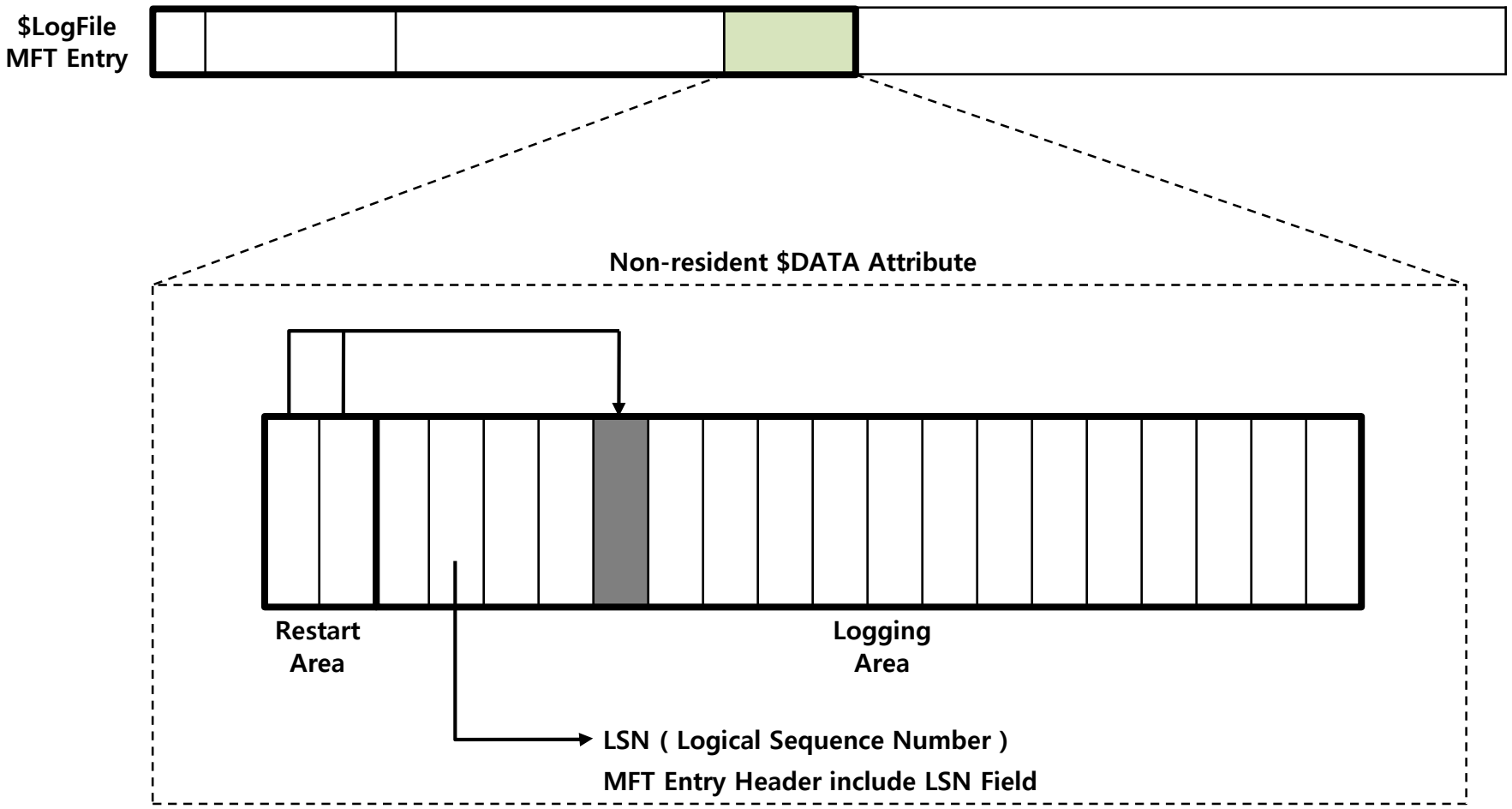
Decryption (applied only $DATA)



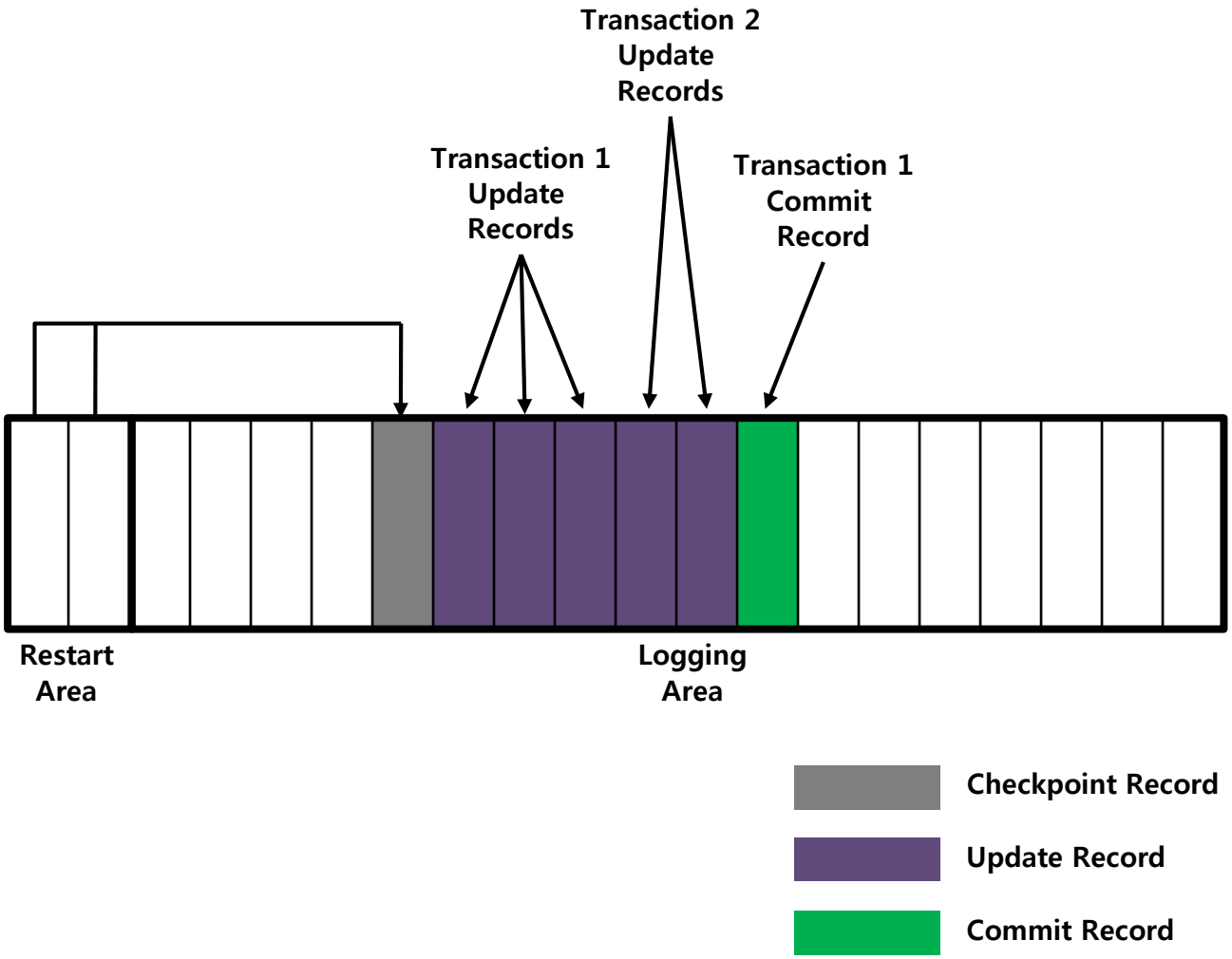
Change Journal ($UsnJrnl)



Logging File System Journaling ($LogFile) – MFT Entry 2



Logging File System Journaling ($LogFile) – MFT Entry 2

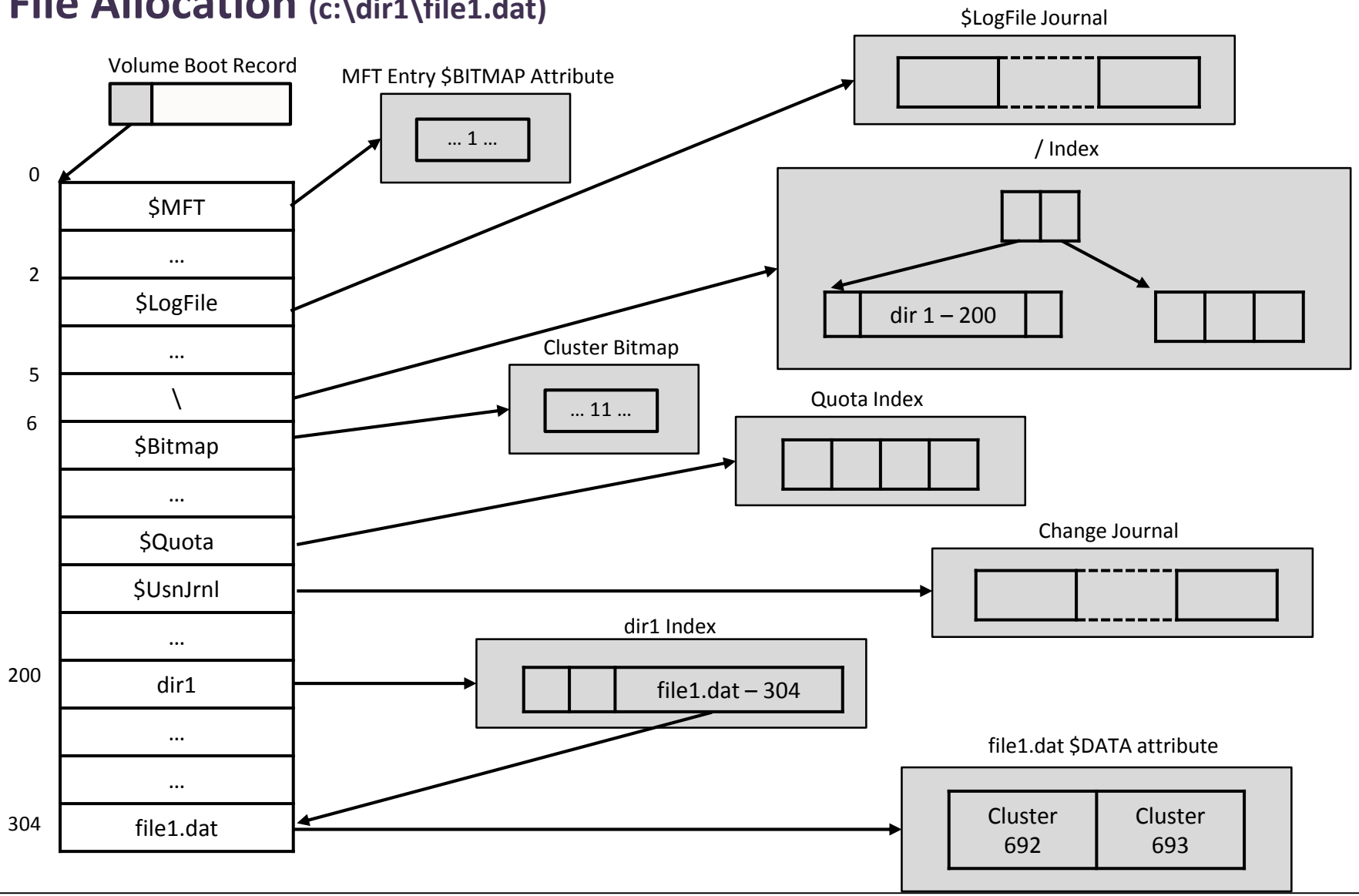


NTFS Example

Security is a people problem…

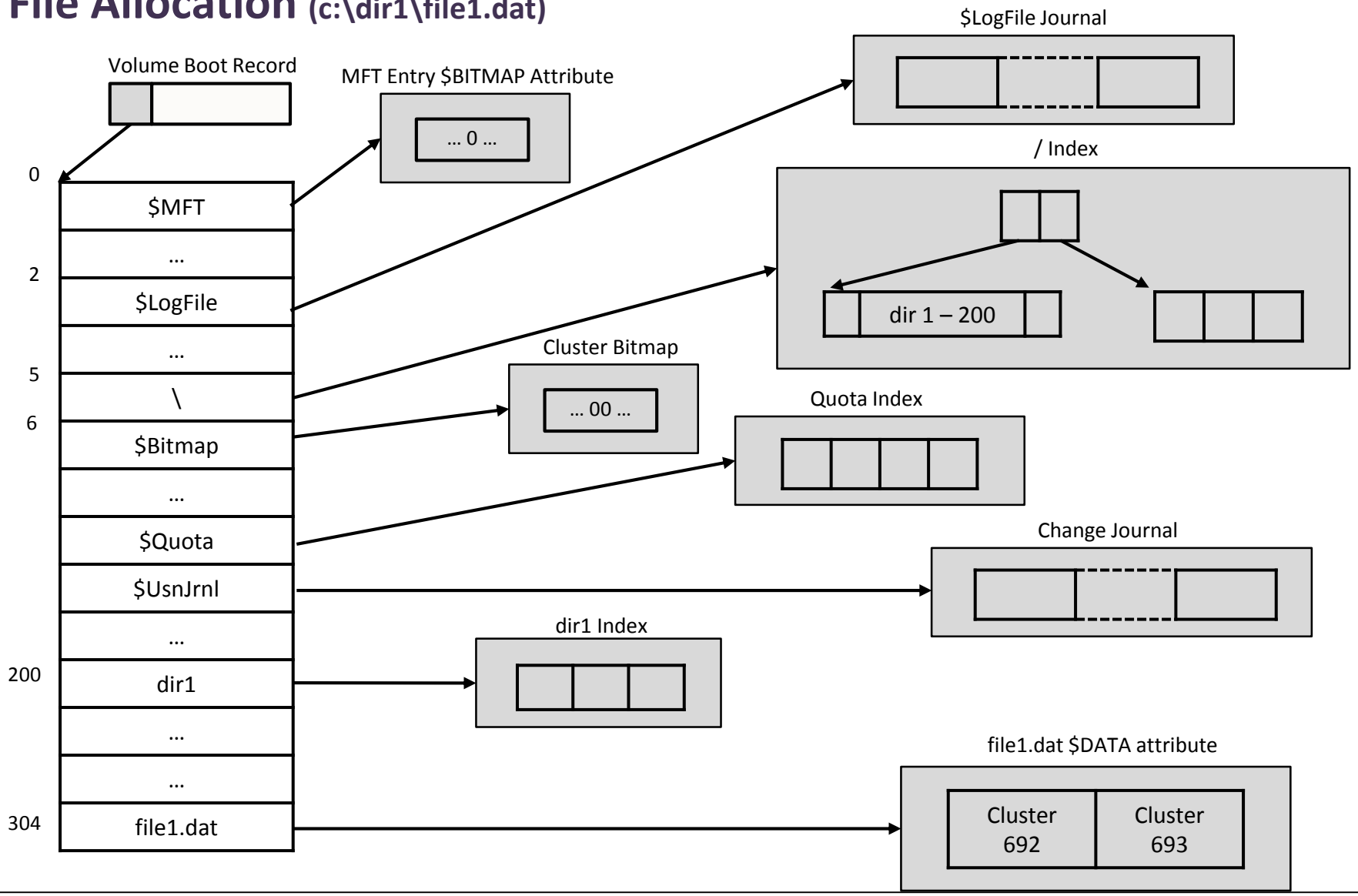
NTFS Example

File Allocation (c:\dir1\file1.dat)



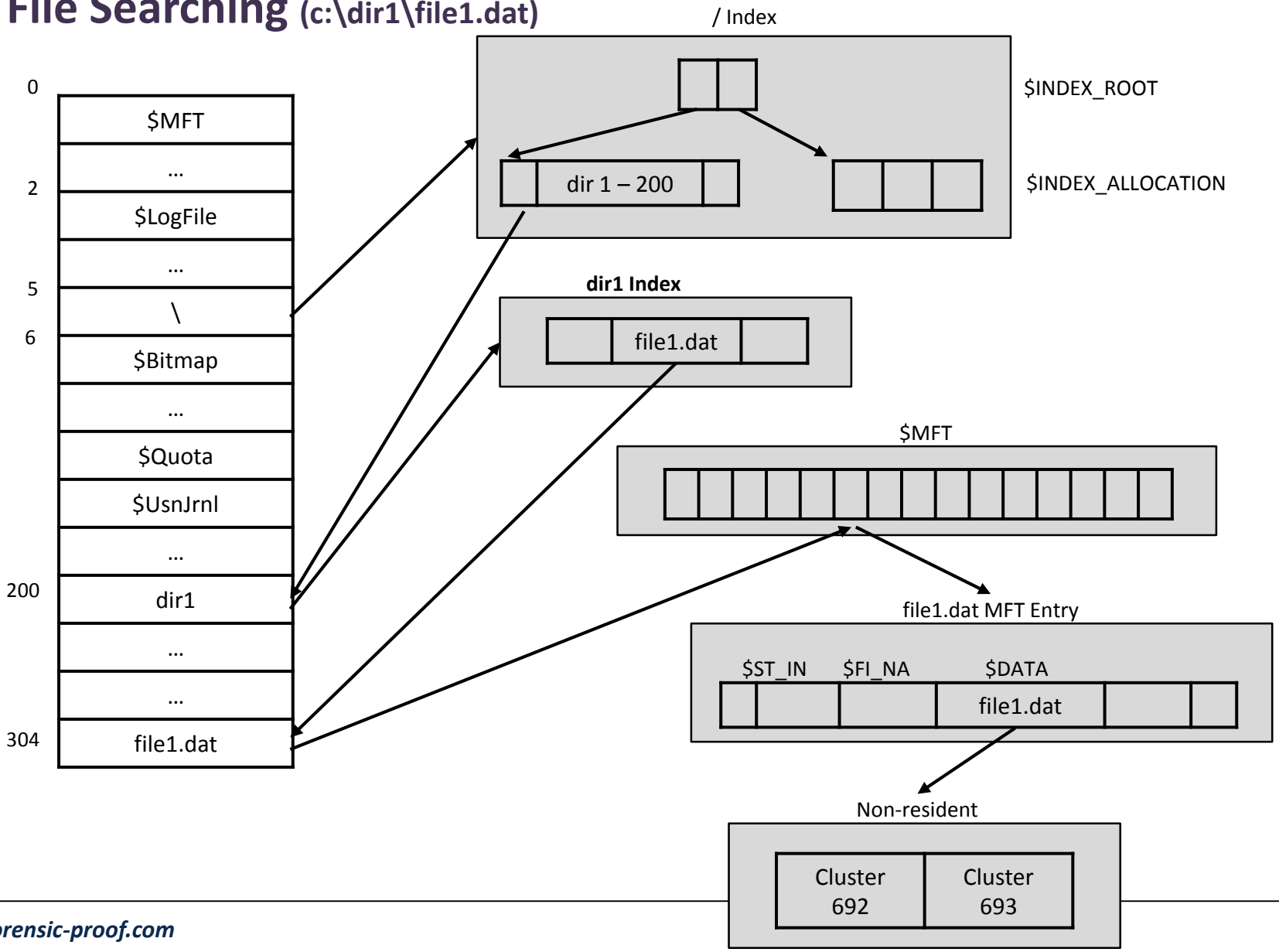
NTFS Example

File Allocation (c:\dir1\file1.dat)



NTFS Example

File Searching (c:\dir1\file1.dat)



Quiz !

Security is a people problem…

Quiz !

NTFS

- FAT32 ➔ NTFS로 변화하면서 추가된 기능은?
- NTFS의 VBR에서 부트 섹터의 역할은?
- NTFS의 VBR에서 2 번째 섹터의 역할은?
- $MFT 메타데이터 파일은 무엇인가?
- MFT Entry에서 Fixup Array를 사용하는 이유는?
- 파일 참조 주소(File Reference Address)에 MFT Entry 순서 번호(Sequence Number)가 포함되는 이유는?
- 클러스터 크기가 4KB인 경우 VBR의 크기는?

Quiz !

NTFS

- MFT Entry의 크기는?
- 파일 및 디렉터리의 시간 정보 위치는?
- $Bitmap 메타데이터 파일과 $BITMAP 속성의 차이는?
- $BadClus 메타데이터 파일이 저장하는 내용은?
- $MFT 크기가 가변적인 이유는?
- Resident 속성과 Non-resident 속성의 차이는?
- 기본적인 파일이 가지는 3가지 속성은?

Quiz !

NTFS

- $STANDARD_INFORMATION, $FILE_NAME 속성의 시간 정보 차이는?
- 파일을 빠르게 탐색하기 위한 구조는?
- Cluster Runs은 무엇인가?
- 숨긴 파일을 찾는 방법은?
- 암호화된 파일을 찾는 방법은?
- Alternate Data Stream (ADS) 의 용도는?
- ADS 속성을 가지는 파일 확인법은?

Quiz !

NTFS

- NTFS 비할당 클러스터 판별법은?
- 삭제된 파일 판별법은?
- 덮어써진 파일 판별법은?
- NTFS의 낭비되는 영역은?
- 연속적인 클러스터에 할당된 파일이 삭제된 경우 복구 방법은?
- 비연속적인 클러스터에 할당된 파일이 삭제된 경우 복구 방법은?

Question & Answer