MBR/GPT/FILE SYSTEM

JK Kim

@pr0neer

forensic-proof.com

proneer@gmail.com

개요

1. 저장장치 구조

2. MBR (Master Boot Record)

3. GPT (GUID Partition Table)

4. 파일시스템

저장장치 구조

저장장치 추상적 구조

저장장치 구조

저장장치 추상적 구조

- **MBR (Master Boot Record)**
  - 모든 저장장치의 가장 처음에 존재하는 구조
  - 최근에는 MBR의 단점을 보완한 GPT (GUID Partition Table)가 사용됨

- **윈도우 시스템 부팅 절차는?**

저장장치 추상적 구조

- **MBR Slack**
  - 저장장치의 시작인 MBR과 볼륨의 시작인 VBR 사이에 존재하는 낭비되는 공간
  - 부트킷, 랜섬웨어 등의 악성코드가 **악용** vs. 보안솔루션 등이 **선용**
  - **윈도우 XP/2K3**
    - ✓ 63섹터 (FDISK 트랙 할당 방식)
  - **윈도우 Vista/7/8**
    - ✓ 2,048섹터 (1MiB 할당 방식)

저장장치 추상적 구조

- **VBR**
  - 볼륨의 시작에 위치하는 구조로 볼륨의 클러스터 크기만큼 할당
  - 파일시스템의 메타 정보(BPB) + 부트로더 로딩 코드
  - 볼륨의 부트로더를 로딩하여 운영체제를 부팅시키는 코드

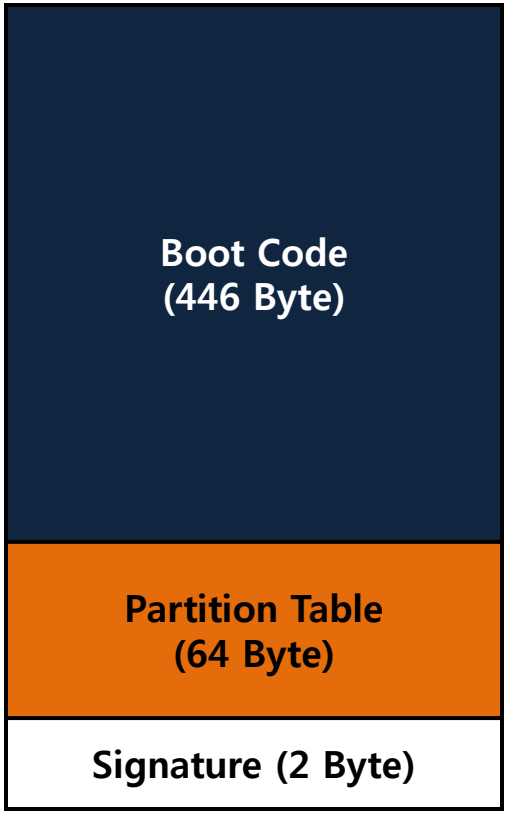
저장장치 추상적 구조

- **Volume Data**
  - 파일시스템에 의해 할당된 볼륨 데이터
  - [메타데이터 + 파일데이터]로 구성

MBR(Master Boot Record)

MBR 구조

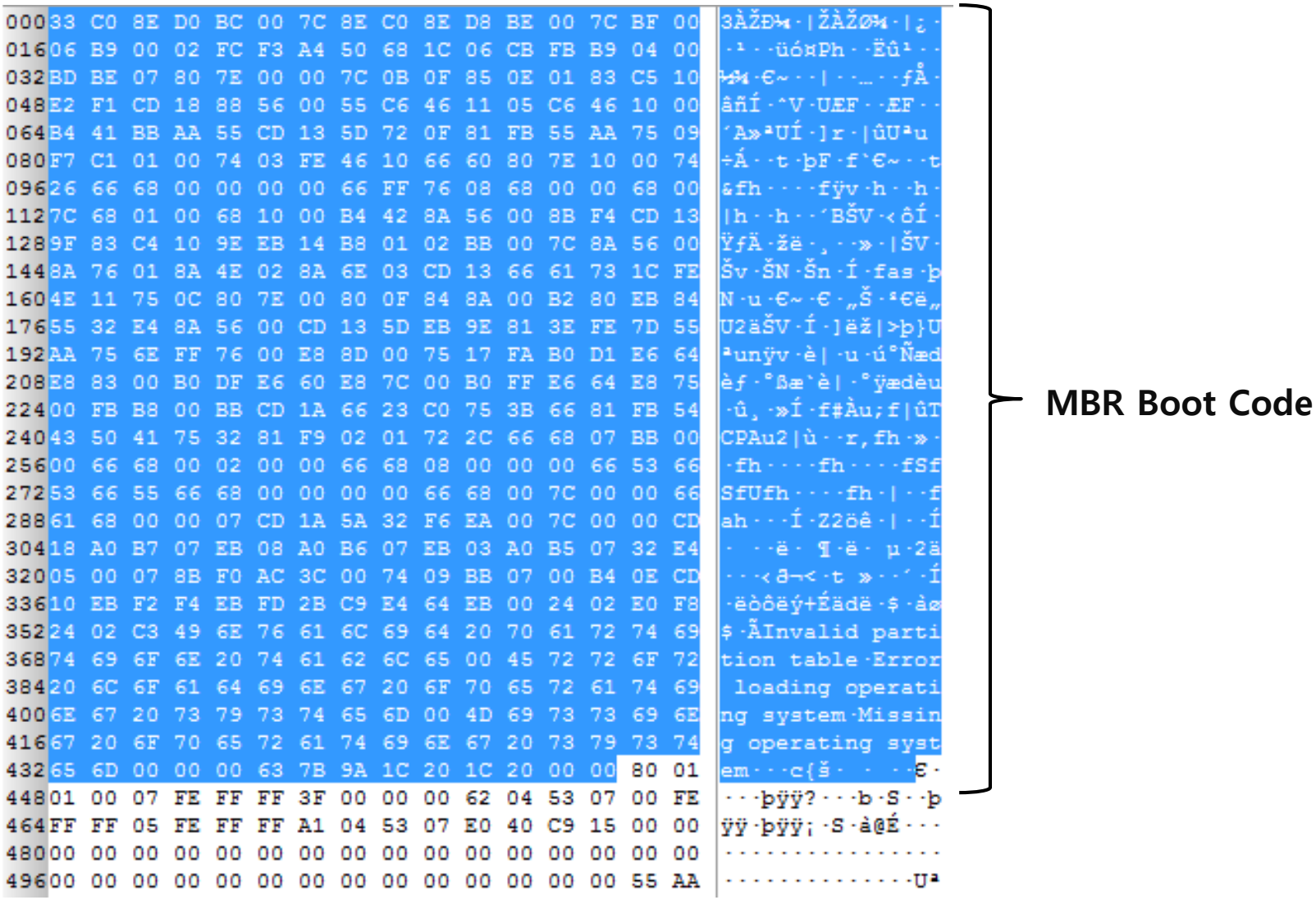
- 저장장치 첫 번째 섹터 (LBA 0)에 위치하는 512 바이트 크기의 영역
- 부트 코드와 파티션 테이블로 구성



MBR 데이터 구조

범 위		설명	크기
10 진수	16 진수		
0 – 445	0x0000 – 0x01BD	부트 코드	446 bytes
446 – 461	0x01BE – 0x01CD	파티션 테이블 엔트리 #1	16 bytes
462 – 477	0x01CE – 0X01DD	파티션 테이블 엔트리 #2	16 bytes
478 – 493	0x01DE – 0x01ED	파티션 테이블 엔트리 #3	16 bytes
494 – 509	0x01EE – 0x01FD	파티션 테이블 엔트리 #4	16 bytes
510 – 511	0x01FE – 0x01FF	시그니처 (0x55AA)	2 bytes

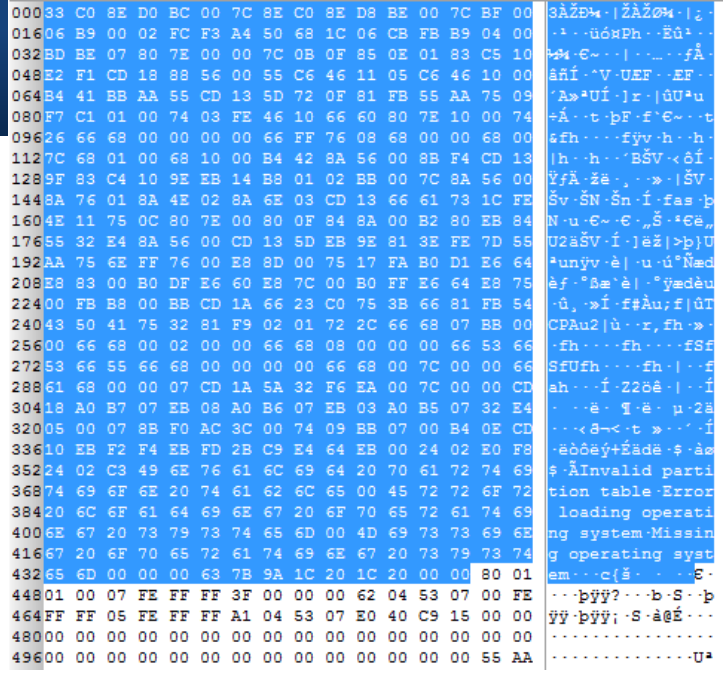
MBR 부트 코드

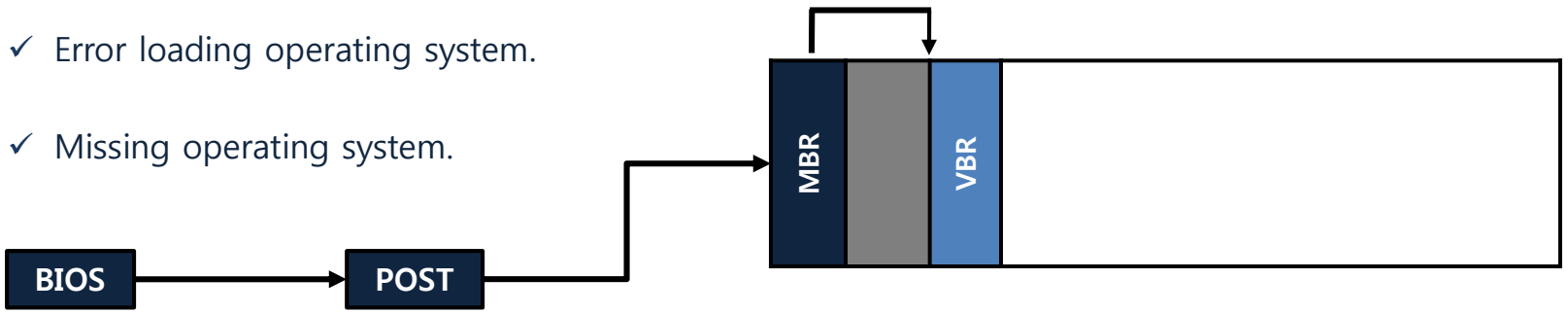


MBR Boot Code

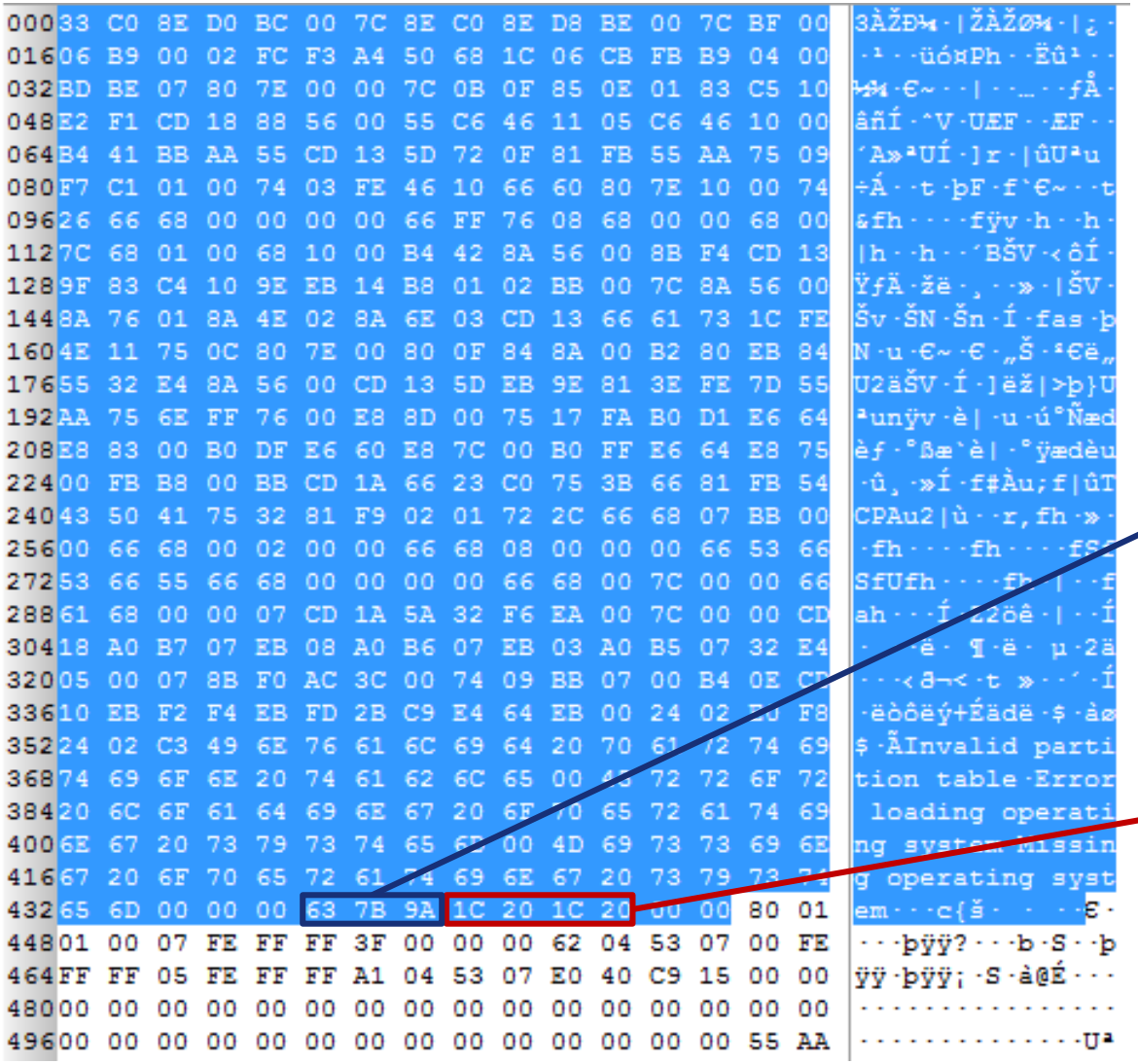
MBR 부트 코드

- 부팅 시 POST 과정 후 저장매체 첫 섹터 호출
- 첫 섹터인 MBR은 자신의 부트 코드 수행
- **부트 코드 역할**
  - MBR 파티션 테이블에서 부팅 가능한 파티션 검색
  - 부팅 가능한 파티션이 있을 경우, 해당 파티션의 VBR로 점프
  - 부팅 가능한 파티션이 없을 경우, 오류 메시지 출력
    - ✓ Invalid partition table.
    - ✓ Error loading operating system.
    - ✓ Missing operating system.





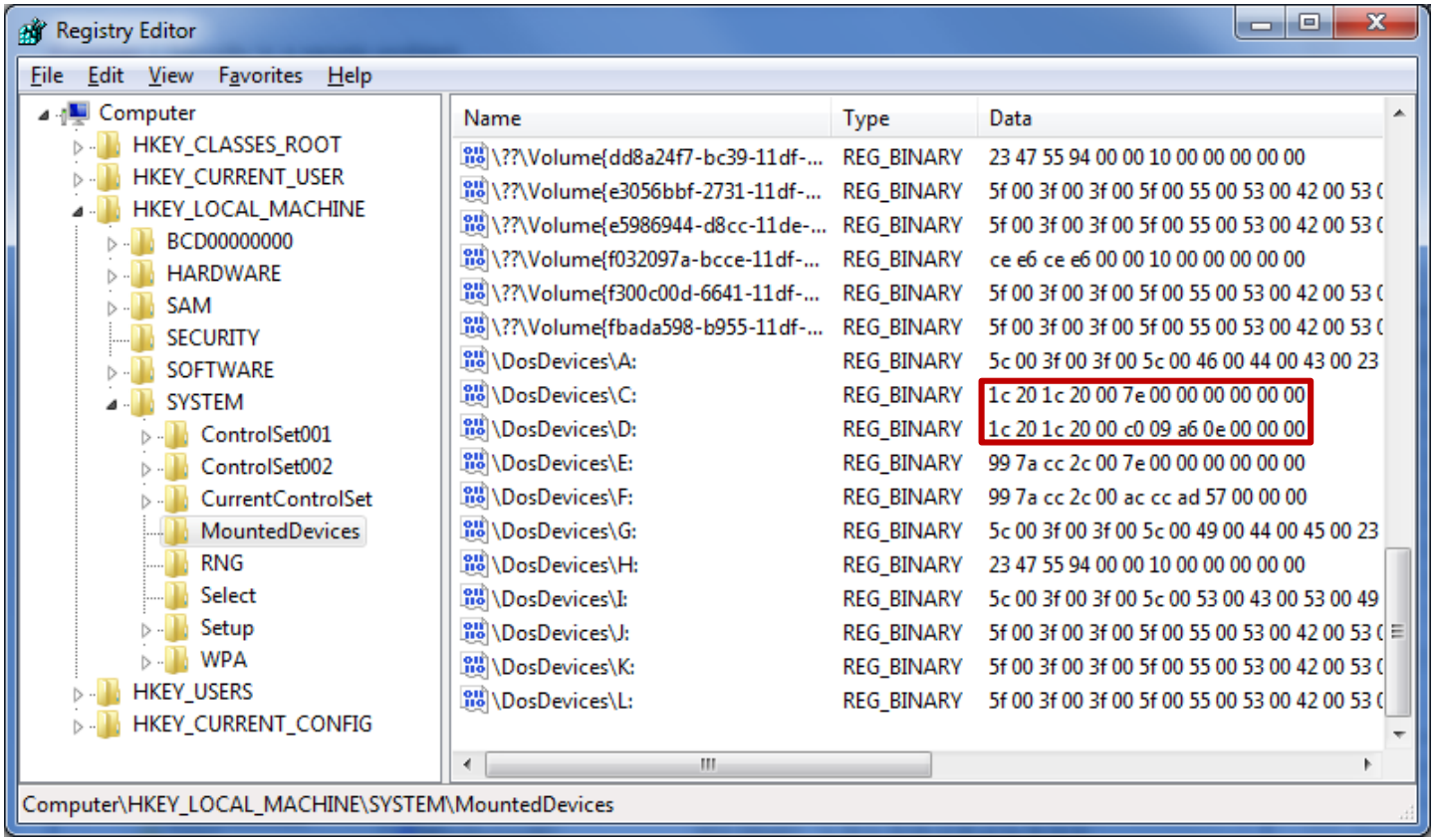
MBR 부트 코드 ➔ Device GUID



MBR 부트 코드 ➔ Device GUID

- 장치가 마운트되면 레지스트리에 장치 GUID(Globally Unique ID) 저장

- HKEY_LOCAL_MACHINE₩SYSTEM₩MountedDevices



MBR 부트 코드 ➔ Device GUID

- **HKEY_LOCAL_MACHINE₩SYSTEM₩MountedDevices**
  - ₩Dos₩Devices₩C: ➔ **1c 20 1c 20** 00 7e 00 00 00 00 00 00
  - ₩Dos₩Devices₩D: ➔ **1c 20 1c 20** 00 c0 09 a6 0e 00 00 00
  - 나머지 8바이트는 각 파티션의 시작 섹터 위치

MBR 파티션 테이블

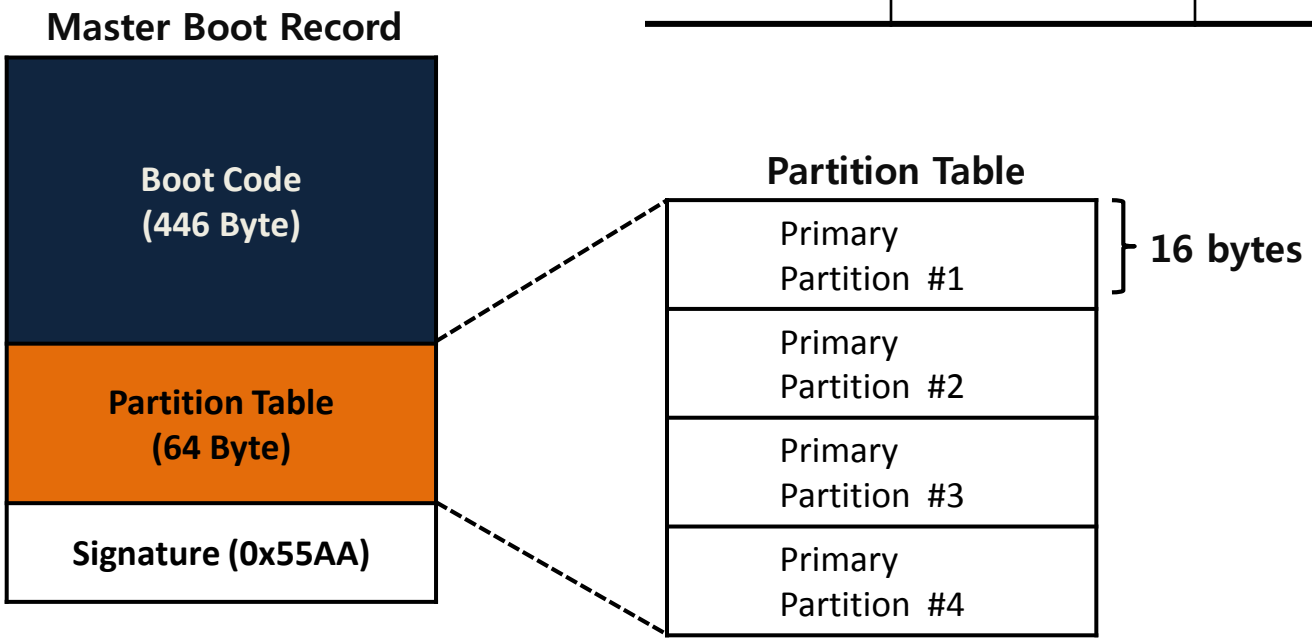
```
000 33 C0 8E D0 BC 00 7C FB 50 07 50 1F FC BE 1B 7C  3ÀŽÐ¼·|ûP·P·ü¾·|
016 BF 1B 06 50 57 B9 E5 01 F3 A4 CB BD BE 07 B1 04  ¿··PW¹å·ó¤Ë½¾·±·
032 38 6E 00 7C 09 75 13 83 C5 10 E2 F4 CD 18 8B F5  8n·| u·ƒÅ·âôÍ·‹õ
048 83 C6 10 49 74 19 38 2C 74 F6 A0 B5 07 B4 07 8B  ƒÆ·It·8,tö µ·´·‹
064 F0 AC 3C 00 74 FC BB 07 00 B4 0E CD 10 EB F2 88  ð¬<·tü»··´·Í·ëòˆ
080 4E 10 E8 46 00 73 2A FE 46 10 80 7E 04 0B 74 0B  N·èF·s*þF·€~··t·
096 80 7E 04 0C 74 05 A0 B6 07 75 D2 80 46 02 06 83  €~··t· ¶·uÒ€F··ƒ
112 46 08 06 83 56 0A 00 E8 21 00 73 05 A0 B6 07 EB  F··ƒV ·è!·s· ¶·ë
128 BC 81 3E FE 7D 55 AA 74 0B 80 7E 10 00 74 C8 A0  ¼|>þ}Uªt·€~··tÈ
144 B7 07 EB A9 8B FC 1E 57 8B F5 CB BF 05 00 8A 56  ·ë©‹ü·W‹õË¿··ŠV
160 00 B4 08 CD 13 72 23 8A C1 24 3F 98 8A DE 8A FC  ·´·Í·r#ŠÁ$?˜ŠÞŠü
176 43 F7 E3 8B D1 86 D6 B1 06 D2 EE 42 F7 E2 39 56  C÷ã‹Ñ†Ö±·ÒîB÷â9V
192 0A 77 23 72 05 39 46 08 73 1C B8 01 02 BB 00 7C   w#r·9F·s·¸··»·|
208 8B 4E 02 8B 56 00 CD 13 73 51 4F 74 4E 32 E4 8A  ‹N·‹V·Í·sQOtN2äŠ
224 56 00 CD 13 EB E4 8A 56 00 60 BB AA 55 B4 41 CD  V·Í·ëäŠV·`»ªU´AÍ
240 13 72 36 81 FB 55 AA 75 30 F6 C1 01 74 2B 61 60  ·r6|ûUªu0öÁ·t+a`
256 6A 00 6A 00 FF 76 0A FF 76 08 6A 00 68 00 7C 6A  j·j·ÿv ÿv·j·h·|j
272 01 6A 10 B4 42 8B F4 CD 13 61 61 73 0E 4F 74 0B  ·j·´B‹ôÍ·aas·Ot·
288 32 E4 8A 56 00 CD 13 EB D6 61 F9 C3 49 6E 76 61  2äŠV·Í·ëÖaùÃInva
304 6C 69 64 20 70 61 72 74 69 74 69 6F 6E 20 74 61  lid partition ta
320 62 6C 65 00 45 72 72 6F 72 20 6C 6F 61 64 69 6E  ble·Error loadin
336 67 20 6F 70 65 72 61 74 69 6E 67 20 73 79 73 74  g operating syst
352 65 6D 00 4D 69 73 73 69 6E 67 20 6F 70 65 72 61  em·Missing opera
368 74 69 6E 67 20 73 79 73 74 65 6D 00 00 00 00 00  ting system·····
384 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ················
400 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ················
416 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ················
432 00 00 00 00 00 2C 44 63 99 7A CC 2C 00 00 80 01  ·····,Dc™zÌ,··€·
448 01 00 07 FE FF FF 3F 00 00 00 D8 E5 D6 2B 00 00  ···þÿÿ?···ØåÖ+··
464 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ················
480 C1 FF 05 FE FF FF 17 E6 D6 2B 2A 66 61 0E 00 00  Áÿ·þÿÿ·æÖ+*fa···
496 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA  ··············Uª
```

Partition Table

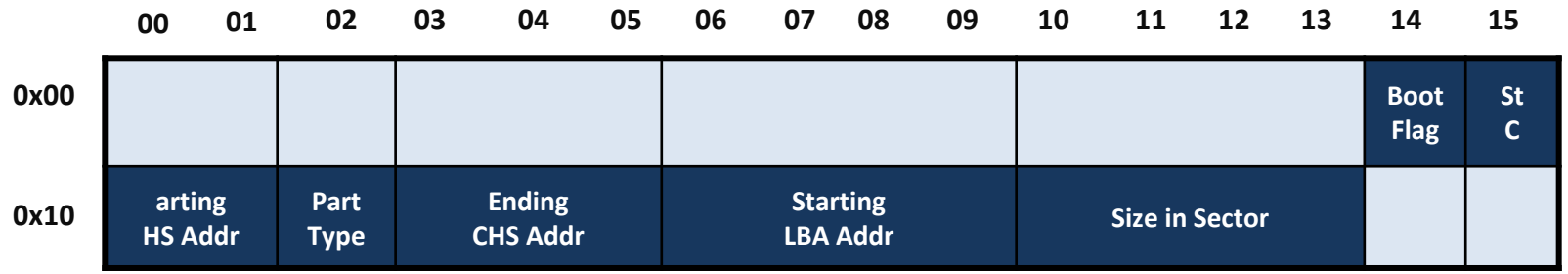
MBR 파티션 테이블

- *Primary Partition*
- *Extended Partition*
- *Logical Partition*

위치	크기	설명
0	446	Boot Code
446	16	Partition #1
462	16	Partition #2
478	16	Partition #3
494	16	Partition #4
510	2	Signature (0x55AA)

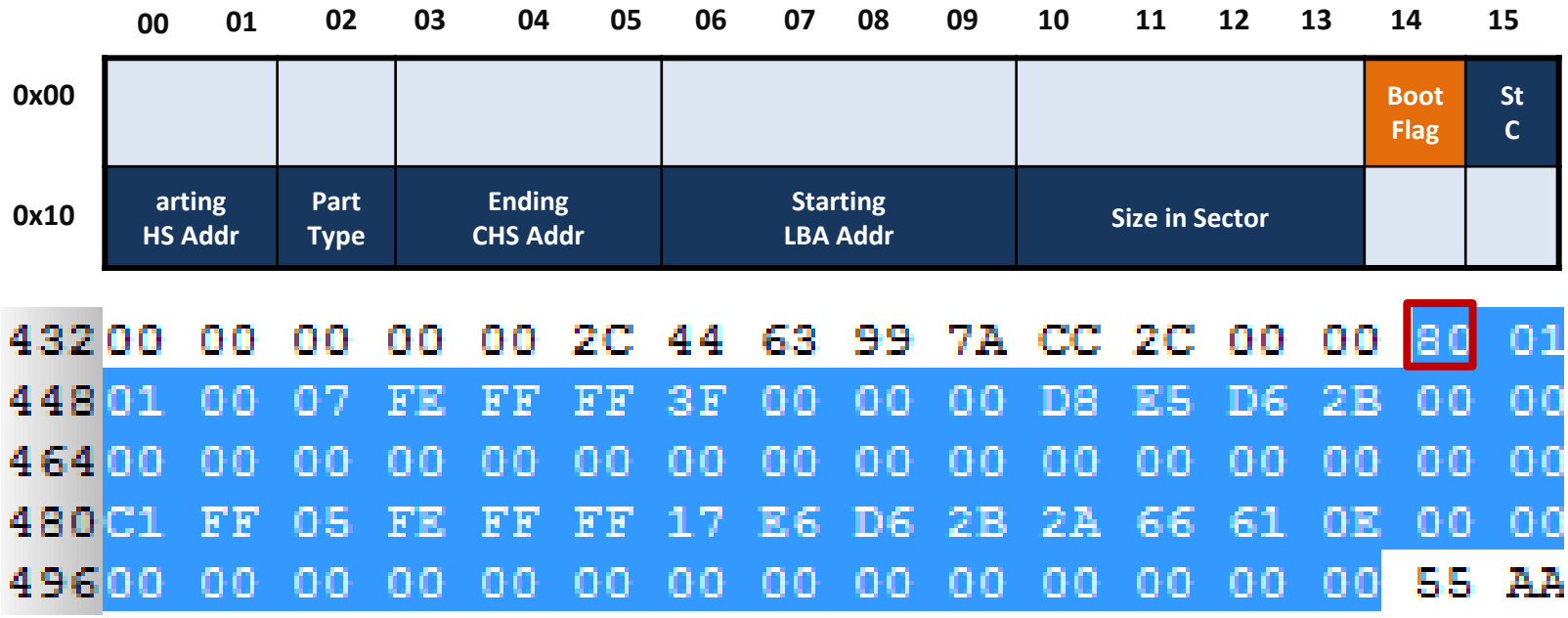


MBR 파티션 테이블



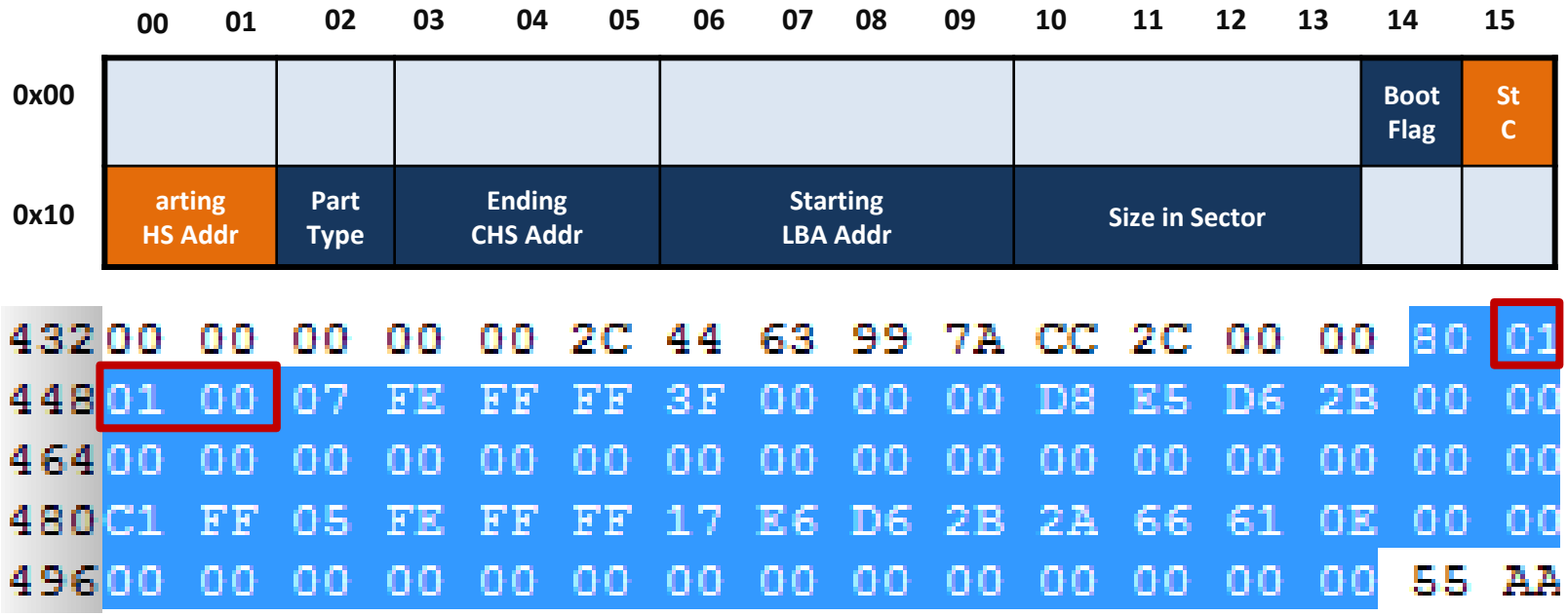

```
432 00 00 00 00 00 2C 44 63 99 7A CC 2C 00 00 80 01
448 01 00 07 FE FF FF 3F 00 00 00 D8 E5 D6 2B 00 00
464 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
480 C1 FF 05 FE FF FF 17 E6 D6 2B 2A 66 61 0E 00 00
496 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA
```

MBR 파티션 테이블



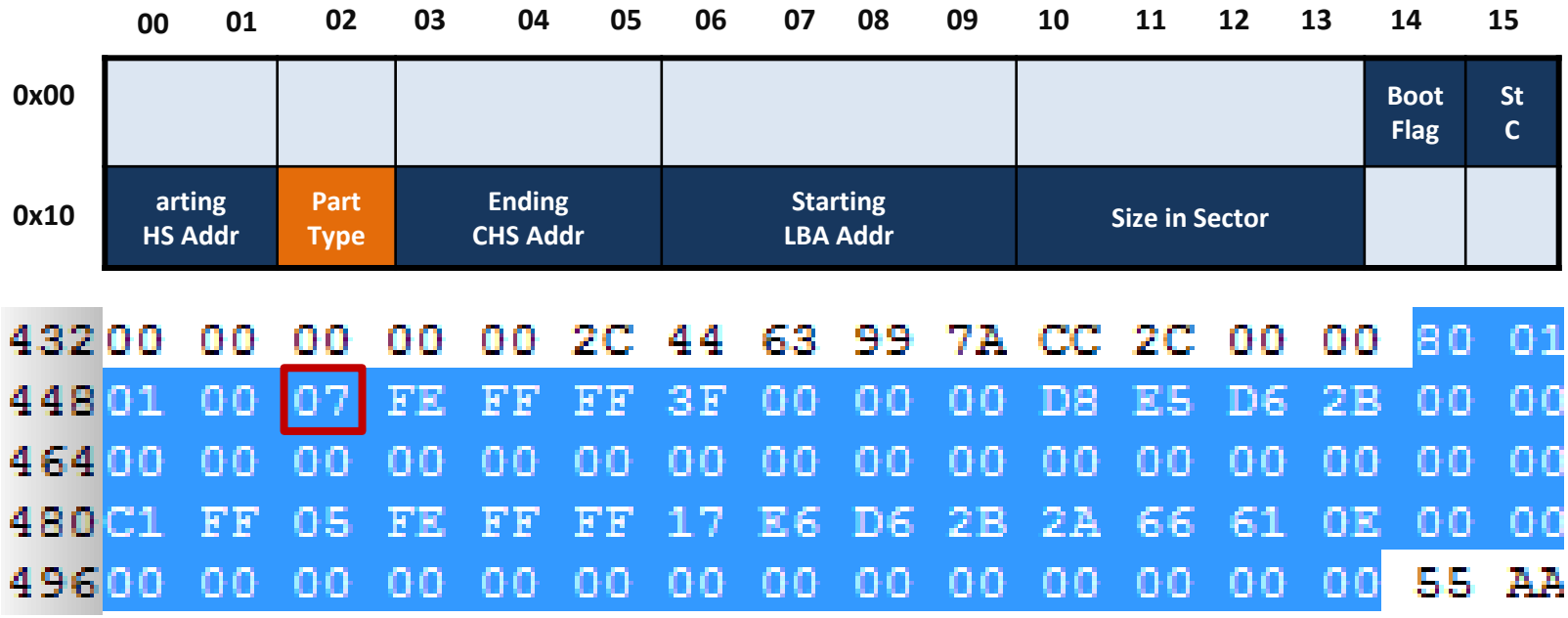
- **부트 플래그(Boot Flag) : 부팅 가능한 저장매체인지를 여부**
  - **0x80** : 부팅 가능
  - **0x00** : 부팅 불가능

MBR 파티션 테이블



- **시작 CHS 주소 (Starting CHS Address)**
  - 주소지정방식이 CHS일 경우 파티션의 시작 위치

MBR 파티션 테이블



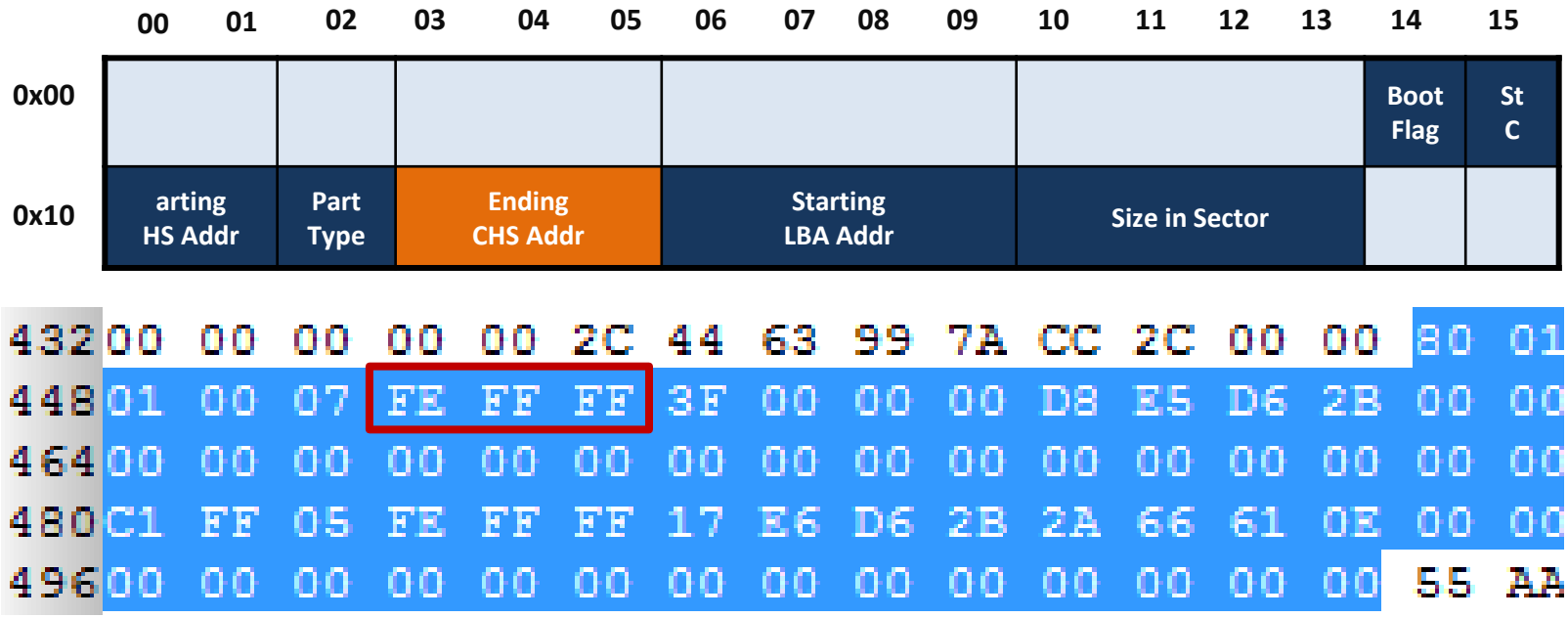
- **파티션 유형 (Partition Type) : 해당 파티션의 유형 (0x00 – 0xFF)**
  - 0x07
  - 파티션 유형 값 조작으로 숨긴 파티션 생성

MBR 파티션 테이블

- **파티션 유형 (Partition Type)**

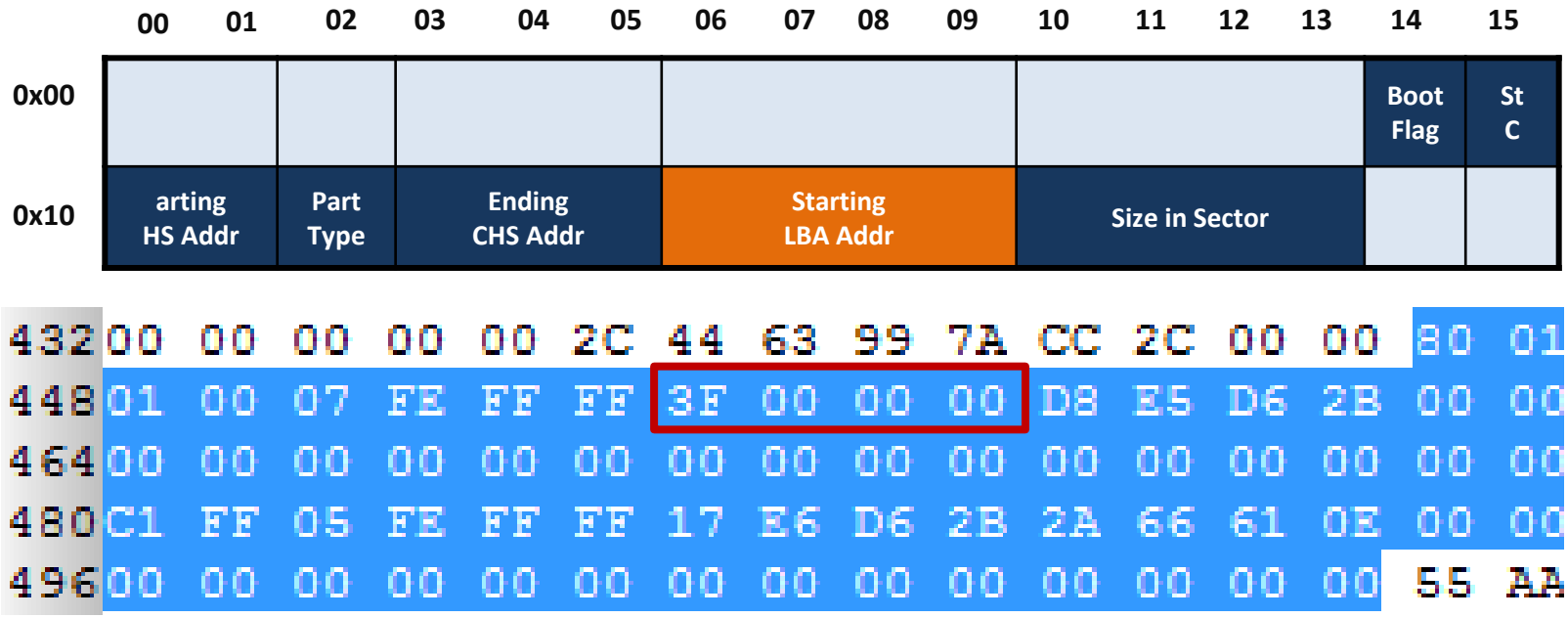
값 (16진수)	설명
00h	Empty
01h	DOS 12-bit FAT, CHS
02h	XENIX root file system, CHS
03h	XENIX /usr file system (obsolete)
04h	DOS 16-bit FAT (up to 32M), CHS
05h	DOS 3.3+ extended partition, CHS
06h	DOS 3.31+ Large File System (16-bit FAT, over 32M), CHS
07h	Advanced Unix, exFAT, NTFS
08h	OS/2 (V1.0 – 1.3 only), AIX bootable partition, Commodore DOS, DELL partition spanning multiple drives
09h	AIX data partition
0Ah	OPUS, Coherent swap partition, OS/2 Boot Manager
0Bh	Windows 95 with 32-bit FAT, CHS
0Ch	Windows 95 with 32-bit FAT (using LBA-mode INT 13 extensions), LBA
0Dh	-
…	…
FEh	LANstep, IBM PS/2 IML
FFh	XENIX bad block table

MBR 파티션 테이블



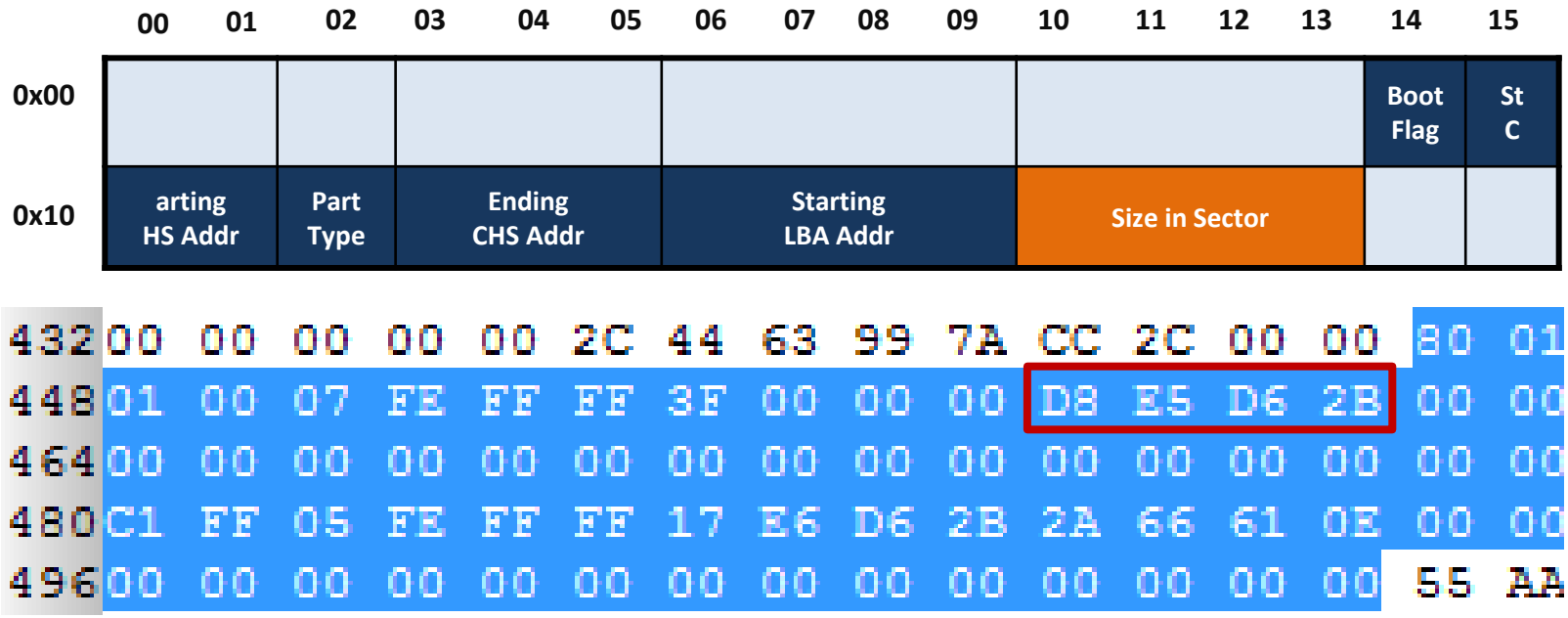
- **마지막 CHS 주소 (Ending CHS Address)**
  - 주소지정방식이 CHS일 경우 파티션의 끝 위치

MBR 파티션 테이블



- **시작 LBA 주소 (Starting LBA Address)**
  - 주소지정방식이 LBA일 경우, 파티션의 시작 섹터 위치
  - **윈도우 XP/2003 이전** : 63 섹터 (DOS 호환 영역, MBR Slack)
  - **윈도우 Vista 이후** : 2048 섹터

MBR 파티션 테이블

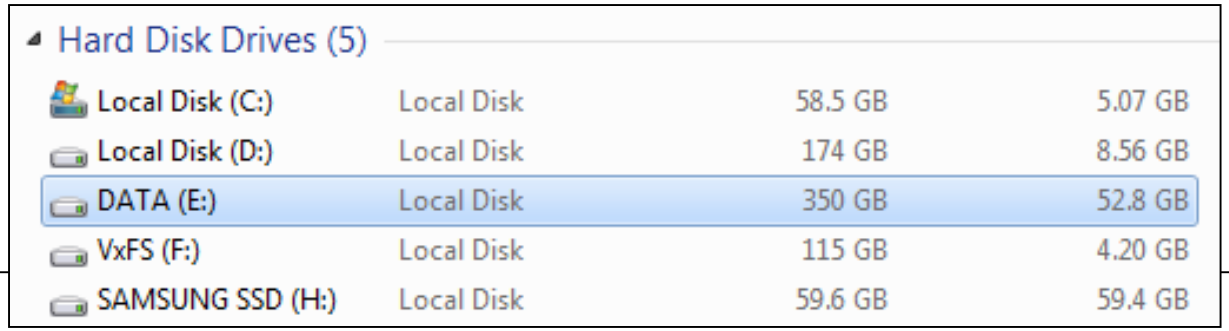


- **파티션 섹터 수 (Size in Sector) : 파티션(LBA)에 할당된 섹터의 총 수**
  - 0x2BD6E5D8 X 512 (sector size) = 376,577,961,984 (350 GB)
- **파티션 테이블이 인식할 수 있는 최대 파티션 크기는?**
  - 2^{32} (4,294,967,295) X 512 = 2,199,023,255,552 = **2 TB**

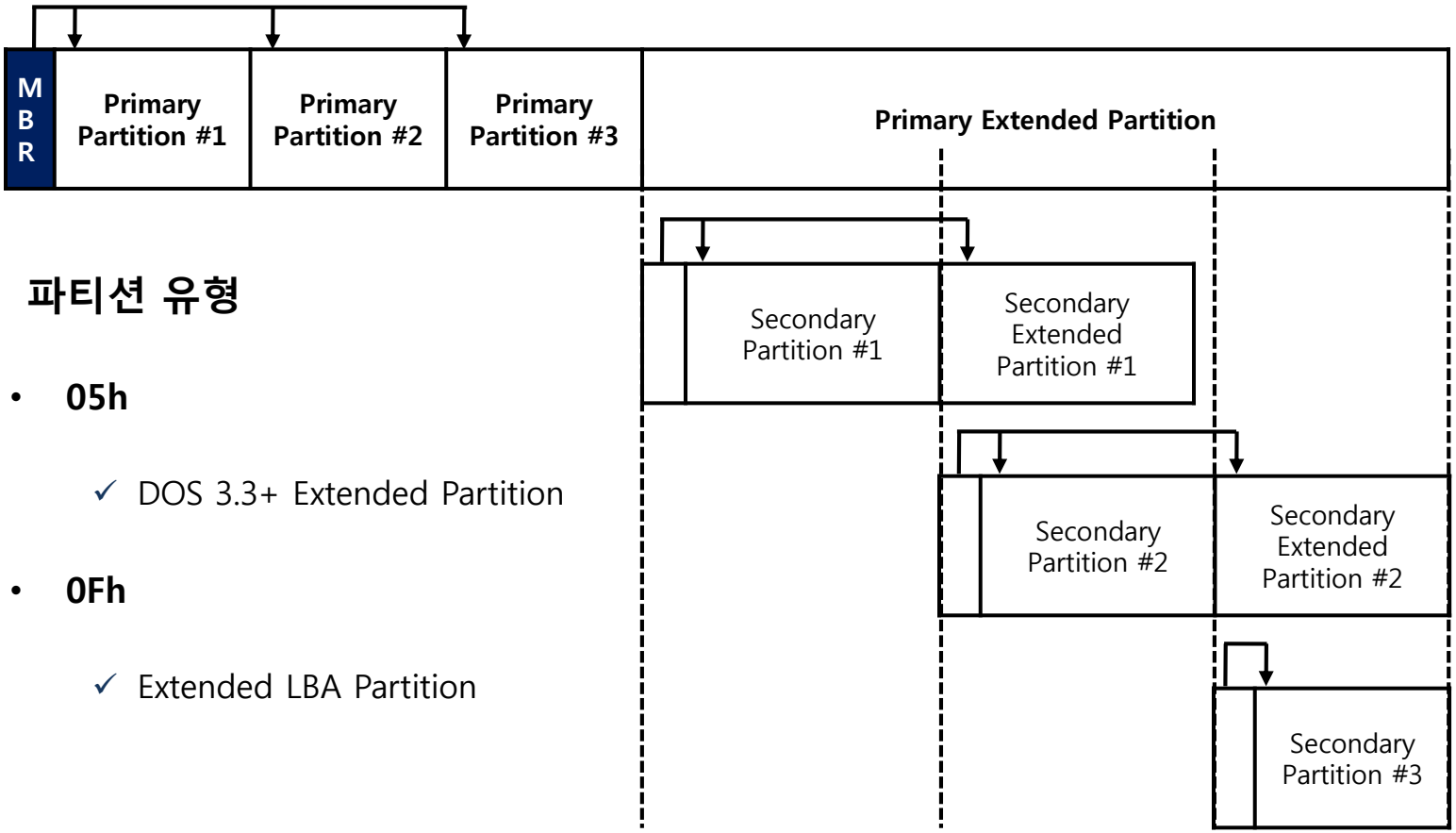
MBR 파티션 테이블

```
432 00 00 00 00 00 2C 44 63 99 7A CC 2C 00 00 80 01
448 01 00 07 FE FF FF 3F 00 00 00 D8 E5 D6 2B 00 00
464 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
480 C1 FF 05 FE FF FF 17 E6 D6 2B 2A 66 61 0E 00 00
496 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 AA
```

Partition	Boot Flag	Starting CHS Address	Partition Type	Ending CHS Address	Starting LBA Address	Size in Sector
#1	0x80	0x000101	0x07	0xFFFFFE	0x0000003F (63)	0x2BD6E5D8 (735,503,832; 350 GB)
#2	0x00	0x000000	0x00	0x000000	0x00000000 (00)	0x00000000
#3	0x00	0xFFC100	0x05	0xFFFFFE	0x2BD6E617 (735,503,895)	0x0E61662A (241,264,170; 115 GB)
#4	0x00	0x000000	0x00	0x000000	0x00000000 (00)	0x00000000



MBR 파티션 테이블



- **파티션 유형**
  - **05h**
    - ✓ DOS 3.3+ Extended Partition
  - **0Fh**
    - ✓ Extended LBA Partition

실습

- MBR 구조 확인하기!!!

GPT(GUID Partition Table)

GPT 소개

- MBR 파티션 테이블의 파티션 용량 제약 ➔ 2TB
- 인텔에서 BIOS의 대체 수단으로 ESI(Extensible Firmware Interface) 표준 제안
- 개선된 EFI 펌웨어에서 지원하는 파티션 테이블 형식 ➔ GPT
- 단순한 파티션 테이블 외에 다양한 디스크 정보 저장
- 1980년 대 : MBR 파티션 발표
- 1990년대 후반 : GPT 파티션 개발

GPT 소개

- 128개의 주(primary) 파티션 생성 가능 (MBR은 4개만 가능)
- 대용량의 볼륨 지원
- MBR 파티션 최대 크기 : (0xFFFF FFFF) = **2 TB (2^{40})**
- GPT 파티션 최대 크기 : (0xFFFF FFFF FFFF FFFF) = **8 ZB (20^{70})**
- CRC (cyclical Redundancy Check)를 이용해 파티션 테이블 보호
- x64 기반의 플랫폼에서 사용 가능
- GPT의 중요 데이터 구조는 볼륨의 끝에 복제본 저장 ➔ 장애 복구 가능

EFI (Extensible Firmware Interface)

- 운영체제와 하드웨어 펌웨어 사이의 새로운 인터페이스
- BIOS (Basic Input/Output System) 대체
- 초기에는 인텔에서 개발, 현재는 통합 (Unified) EFI로 발전
- **주요 특징**
  - GUI 인터페이스
  - 마우스 사용 가능
  - Pre-OS 소프트웨어 구동 가능
    - ✓ 시스템 복구, 인터넷 브라우저 등
  - 네트워크 기능
  - 다국어(한국어 포함) 지원

GPT 지원

- 윈도우 32비트

운영체제	플랫폼	읽기/쓰기 지원	부트 지원
Windows XP	IA-32	No	No
Windows Server 2003	IA-32	No	No
Windows Server 2003 SP1	IA-32	**YES**	No
Windows Vista	IA-32	**YES**	No
Windows Server 2008	IA-32	**YES**	No
Windows 7	IA-32	**YES**	No
Windows 8	IA-32	**YES**	No

http://en.wikipedia.org/wiki/GUID_Partition_Table

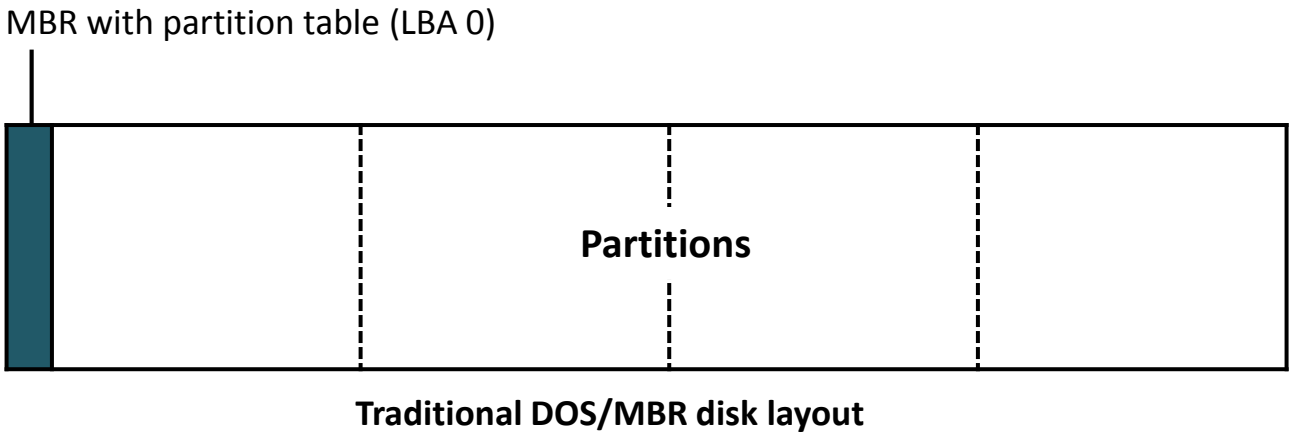
GPT 지원

- 윈도우 64비트

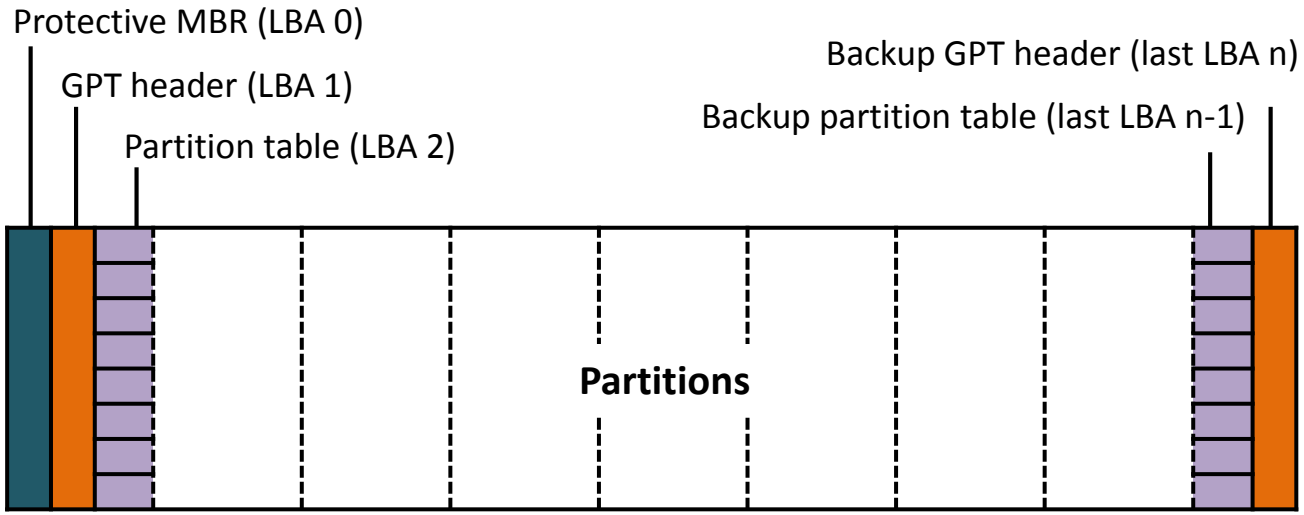
운영체제	플랫폼	읽기/쓰기 지원	부트 지원
Windows XP Pro x64 Edition Windows Server 2003	x64	**YES**	No
Windows Server 2003	IA-64	**YES**	**YES**
Windows Vista	x64	**YES**	Requires UEFI
Windows Server 2008	x64	**YES**	Requires UEFI
Windows Server 2008	IA-64	**YES**	**YES**
Windows 7 Windows Server 2008 R2	x64	**YES**	Requires UEFI
Windows Server 2008 R2	IA-64	**YES**	**YES**
Windows 8	x64	**YES**	Requires UEFI

http://en.wikipedia.org/wiki/GUID_Partition_Table

GPT 구조

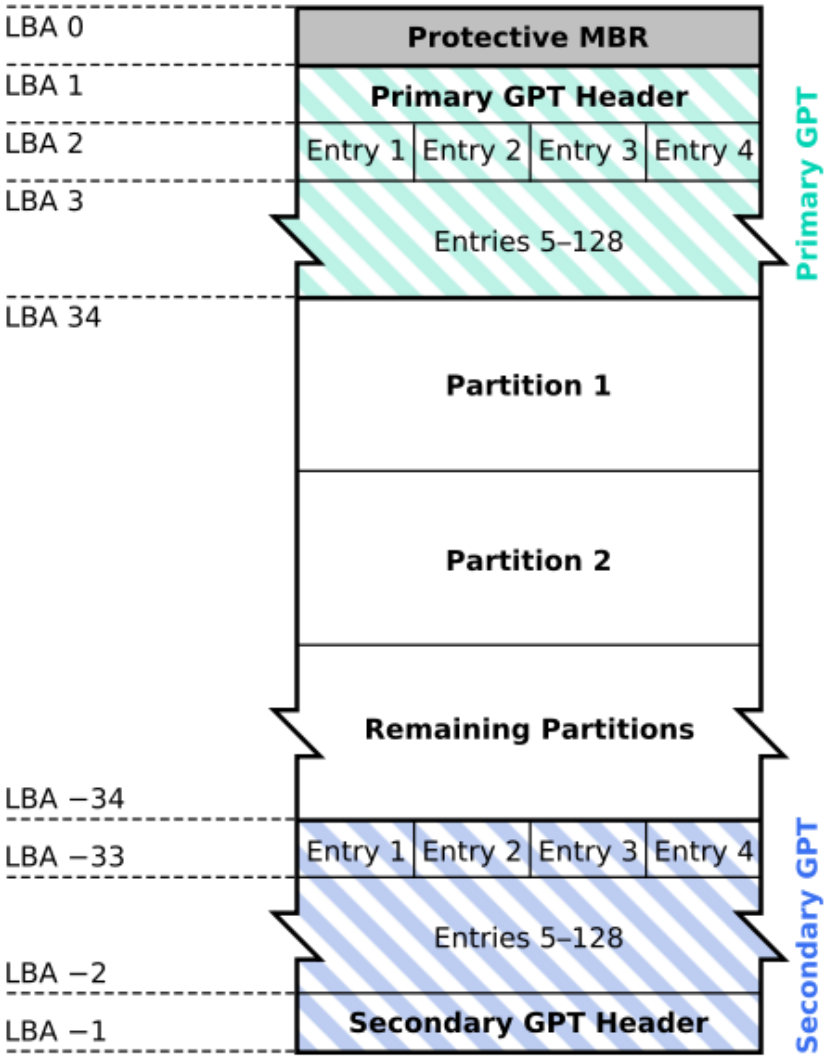


Traditional DOS/MBR disk layout



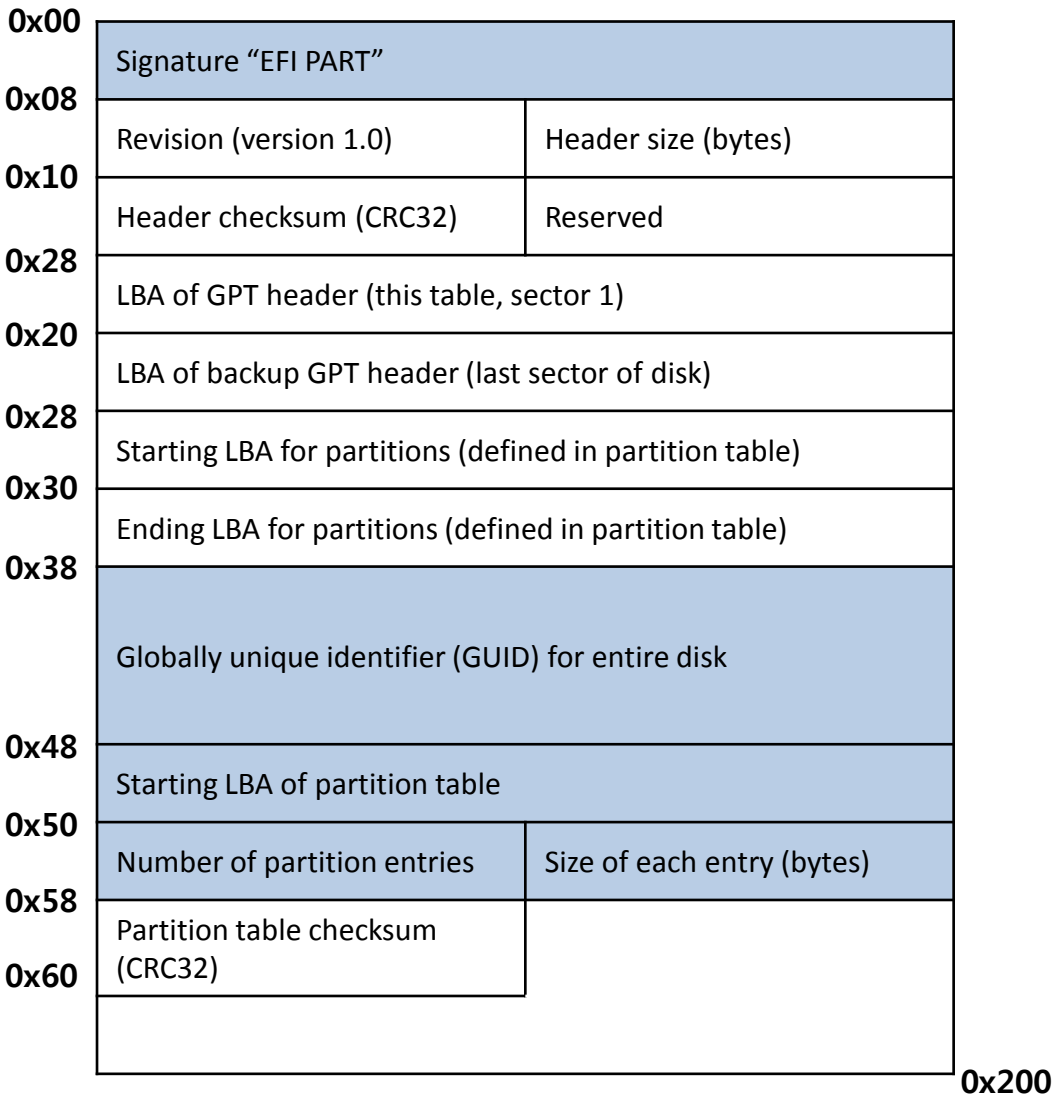
GPT disk layout

GPT 구조

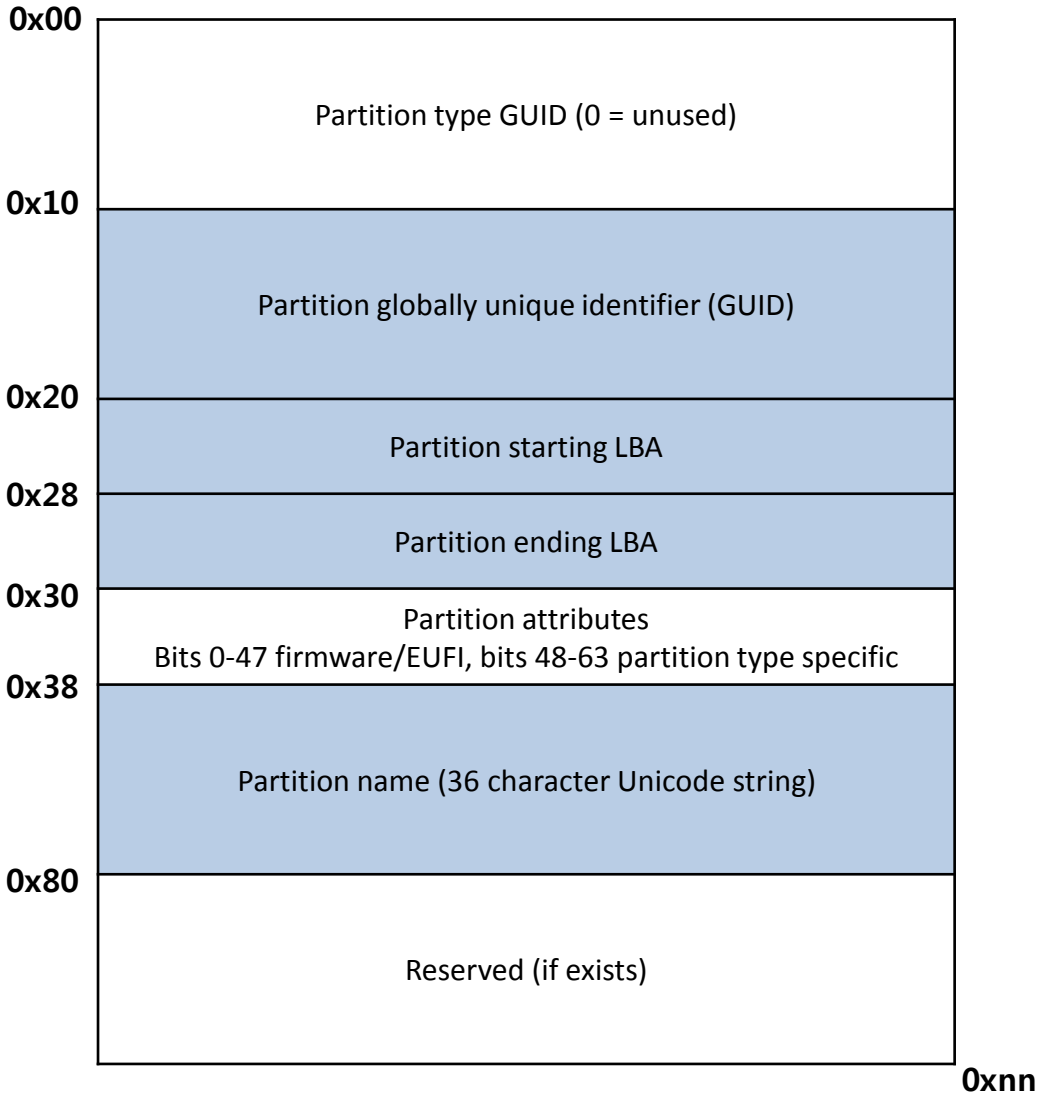


- **Protective MBR**
  - ✓ 기본 MBR과 호환
- **GPT Header**
- **GPT Entry**
  - ✓ 최대 128개 파티션
- **Backup**
  - ✓ GPT Header
  - ✓ GPT Entry

GPT 구조 ➔ GPT Header



GPT 구조 ➔ GPT Entry



파일시스템

파일시스템을 왜 사용하는가?

- 데이터는 파일 형태로 저장장치에 저장
- 저장장치 공간이 커질 수록 파일 수 증가 ➔ 파일시스템 필요
- 압축, 암호화, 저널, 동적 할당, 다국어 지원 등 추가기능 지원

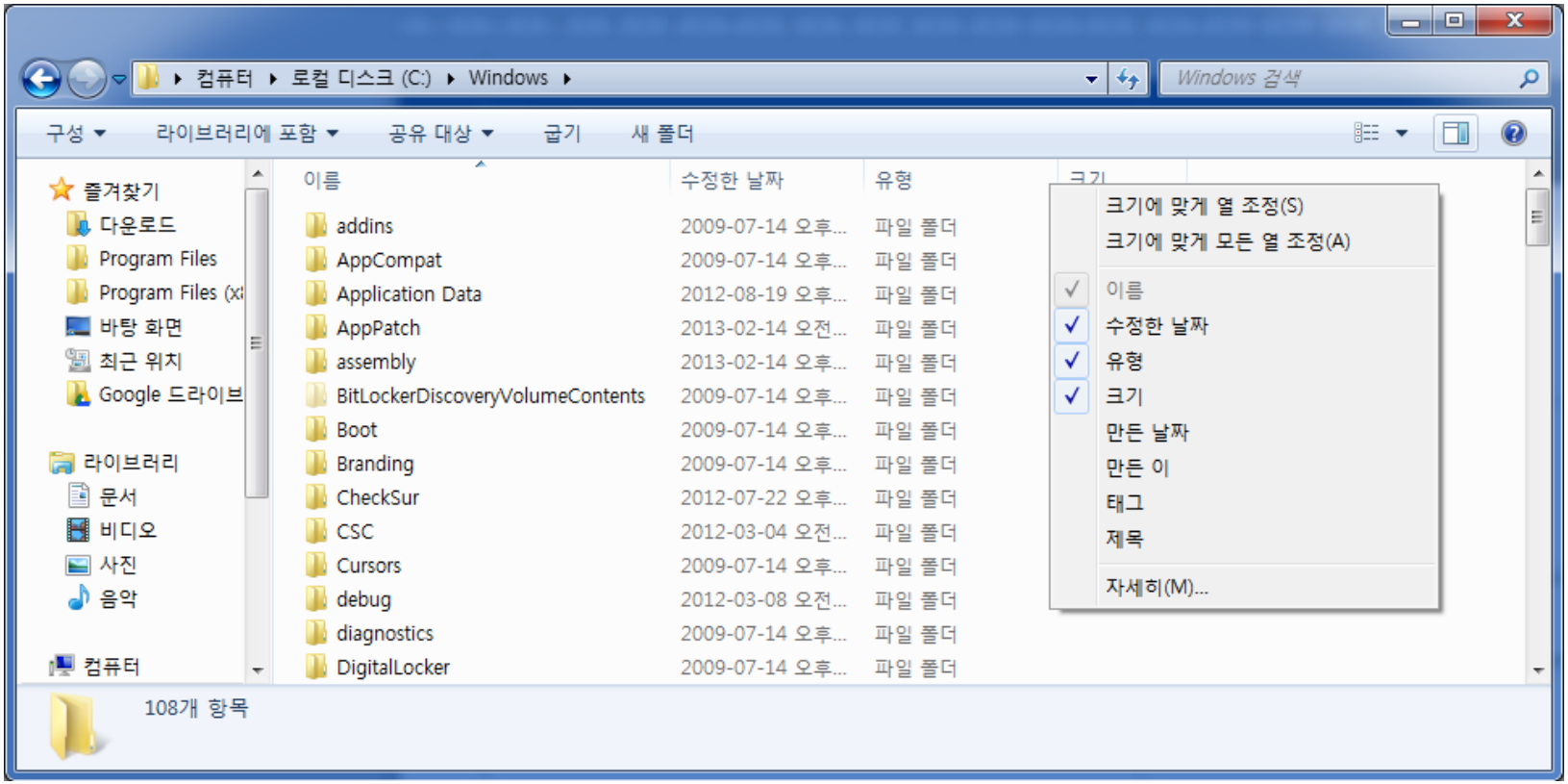
저장매체	운영체제	파일시스템
디스크 장치	**Windows**	FAT(FAT12/16/32, exFAT), NTFS
	Linux	ext2/3/4
	Unix-like	UFS
	OS-2	HPFS
	Mac OS	HFS, HFS+
	Solaris	ZFS
	AIS	JFS
	IRIX	XFS
	HP-UX	ODS-5, VxFS
광학장치		ISO 9660, UDF

추상적 구조

- 거의 모든 파일시스템은 메타 영역과 데이터 영역으로 구분
- 메타 영역 – 파일 이름, 속성, 크기, 시간 정보 등
- 데이터 영역 – 파일의 실제 데이터

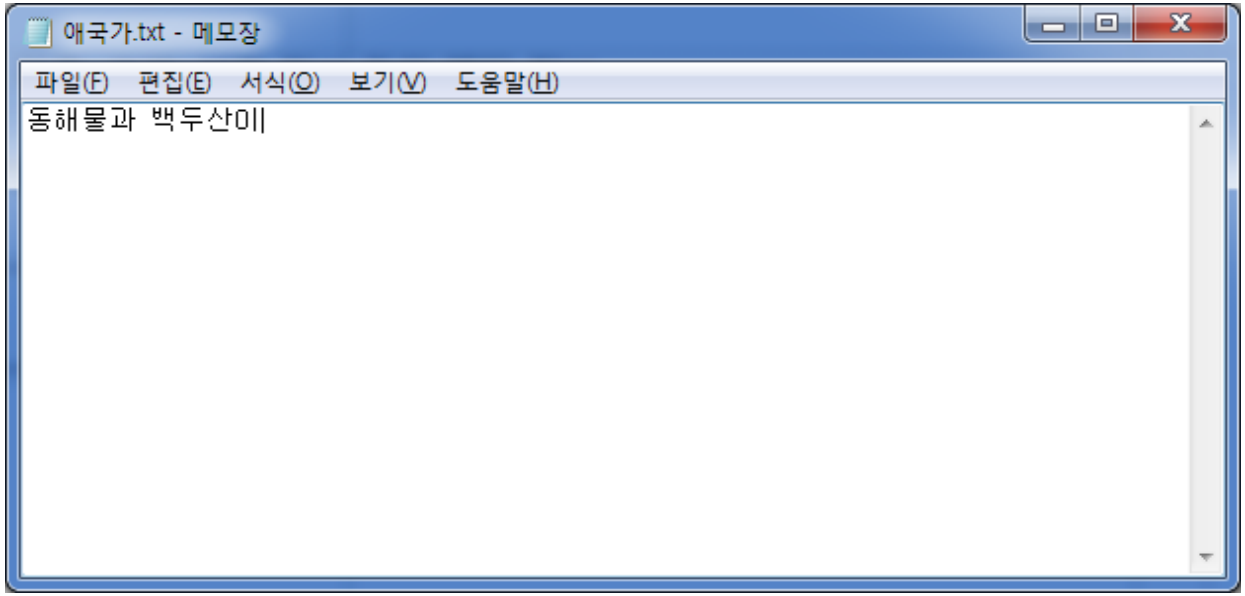
메타 영역

- 파일을 열거나 실행하기 전까지는 메타 정보만을 이용해 탐색
- 파일을 실행하면 메타 정보를 기반으로 실제 데이터 접근



데이터 영역

- 파일의 실제 데이터가 저장된 영역
- 메모장에 "동해물과 백두산이"라고 입력 후 "C:₩애국가.txt" 저장
- **메타 영역** – 파일 이름, 위치, 크기, 유형 등
- **데이터 영역** – "동해물과 백두산이"



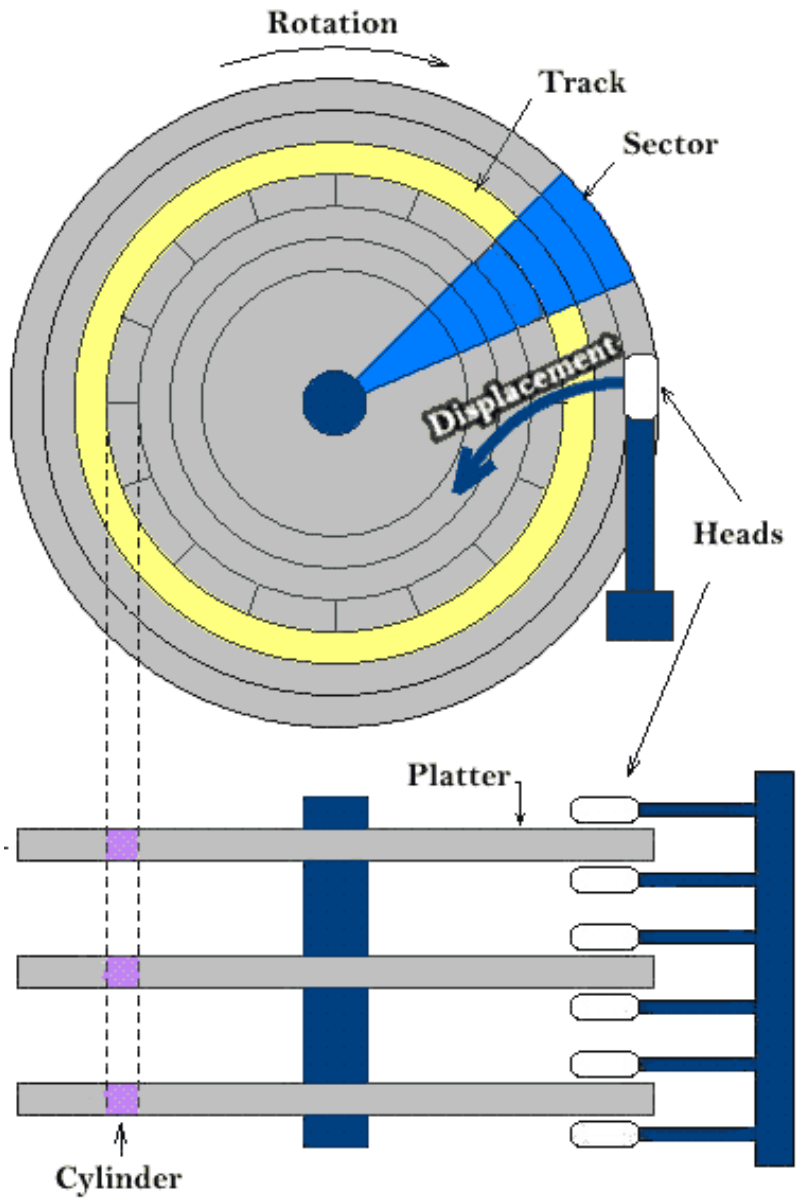
섹터 주소 지정 방식

- 저장장치는 섹터라는 최소한의 데이터 입/출력 단위 사용
- 섹터는 보통 512 바이트 (물리적으로는 ??)
- 최근 섹터 크기가 4,096 바이트(Advanced Format)인 제품 출시
- 특정 파일에 접근하고자 할 경우, 파일이 위치한 섹터 주소로 접근해야 함

- **섹터 주소 지정 방식**
  - **CHS** (Cylinder-Head-Sector)
  - **LBA** (Logical Block Addressing)

섹터 주소 지정 방식

- CHS 주소 지정 방식



섹터 주소 지정 방식

- **CHS 주소 지정 방식**
  - 실린더(Cylinder), 헤드(Head), 섹터(Sector)의 물리적인 구조를 기반으로 주소 지정
  - CHS(21, 3, 20) 주소에서 파일 읽기
    - ✓ 3번째 헤드를 21번째 실린더의 20번째 섹터로 이동한 후 정해진 크기만큼 읽기
  - **용량 제약 발생**

	할당비트 (실린더 수)	할당비트 (헤드 수)	할당비트 (섹터 수)	표현 가능 최대 용량
IDE/ATA 표준	16 (65,536)	4 (16)	8 (256)	128 GB
BIOS INT 13h	10 (1,024)	8 (256)	6 (63)	7.88GB
최소 가능 비트	10 (1,024)	4 (16)	6 (63)	**504 MB**

    - ✓ 2^{10} (1,024) X 2^{4} (16) X 2^{6}-1 (63) X 512 = 528,482,304 **(504 MB)**
    - ✓ 실린더, 헤드는 0부터 시작, 섹터는 1부터 시작

섹터 주소 지정 방식

- **CHS 주소 지정 방식**
  - BIOS보다 ATA 표준이 더 많은 수의 비트를 할당
  - BIOS를 통해 전달되는 비트를 변환하여 지정함으로써 용량 증가
  - Large Mode 또는 Extended CHS (ECHS)
  - 예) 웨스턴 디지털 (WD, Western Digital) 社의 Caviar AC33100

	실린더 수	헤드 수	섹터 수	표현 용량
IDE/ATA 표준	65,536	16	256	128 GB
Hard Disk Logical Geometry	6,136	16	63	2.95 GB
BIOS Translation Factor	Divide by 8	Multiply by 8	-	-
BIOS Translated Geometry	767	128	63	2.95 GB
BIOS INT 13h	1,024	256	63	7.88 GB

섹터 주소 지정 방식

- **LBA 주소 지정 방식**
  - 저장매체 용량 증가에 따라 CHS를 대체하기 위한 방식
  - CHS, LBA 모두 ATA-1 명세에 정의되었으나 직관적인 이유로 CHS가 먼저 사용
  - 물리적인 구조와 상관없이 모든 섹터를 선형적으로 배열 ➔ 논리적인 주소
  - 섹터 주소는 0부터 시작
  - 논리 주소와 물리 주소의 맵핑은 저장장치 컨트롤러가 담당
  - LBA 등장으로 CHS는 ATA-6 명세부터 사라짐

섹터 주소 지정 방식

- **LBA 주소 지정 방식**
  - **CHS ➔ LBA 변환**

```
LBA = ( (CYLINDER * heads per cylinder + HEAD ) * sectors per track ) + SECTOR -1
```

  - **LBA ➔ CHS 변환**

```
CYLINDER = LBA / ( heads per cylinder * sectors per track )
HEAD = ( LBA / sectors per track ) % heads per cylinder
SECTOR = ( LBA % sector per track ) + 1
```

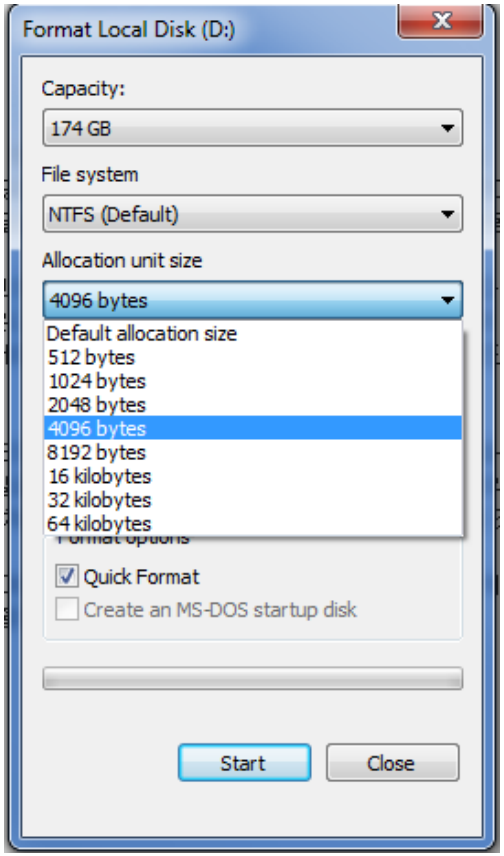
  - **ZBR** (Zone Bit Recording) 환경도 고려

섹터 주소 지정 방식

- **LBA 주소 지정 방식**
  - LBA는 초기 ATA 표준에서 28비트(2^{28})를 할당하여 주소 지정 (4비트는 다른 용도)
  - 2^{28} (268,435,456 섹터) X 512 (섹터 크기) = **128 GB**
  - ATA-6 표준에서 용량 제약의 문제로 48비트 LBA로 확장
  - 2^{48} (281,474,976,710,656) X 512 = 134,217,728 GB = **128 PB**
  - 이 용량이 제약이 될 수 있을까?

클러스터와 블록

- 데이터 관리와 CPU 성능 효율을 위해 클러스터 또는 블록을 사용
- 4MB 데이터를 쓰기 위해 4K라면 **1,024번**, 512바이트라면 **8,192번**



FAT32

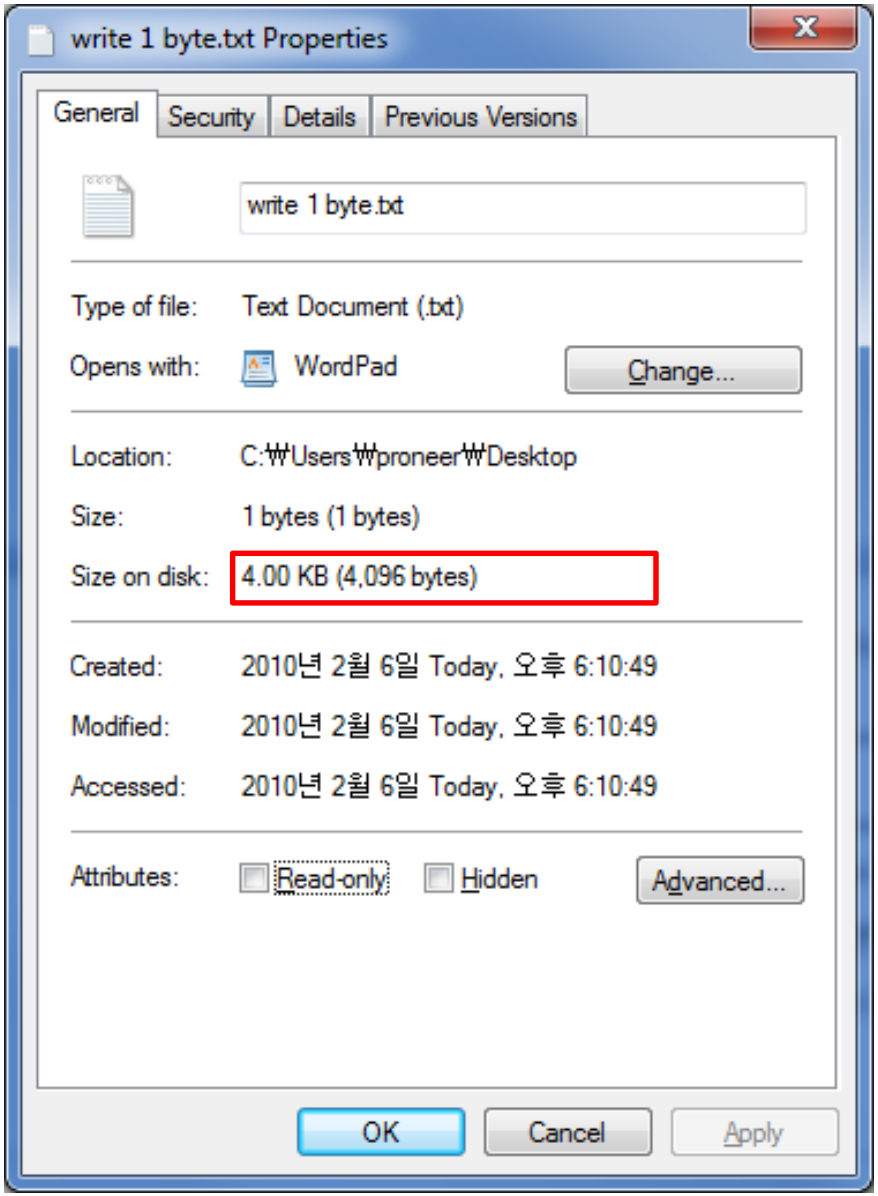
볼륨 크기	클러스터 크기
32MB – 8GB	4KB
8GB – 16GB	8KB
16GB – 32GB	16KB
32GB -	32KB

NTFS

볼륨 크기	클러스터 크기
7MB – 512MB	512Byte
513MB – 1GB	1KB
1GB – 2GB	2KB
2GB -	4KB

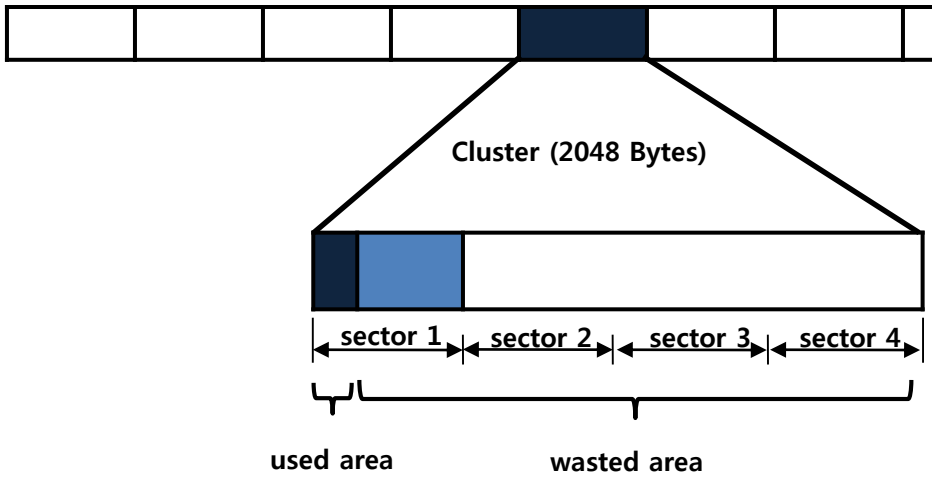
클러스터와 블록

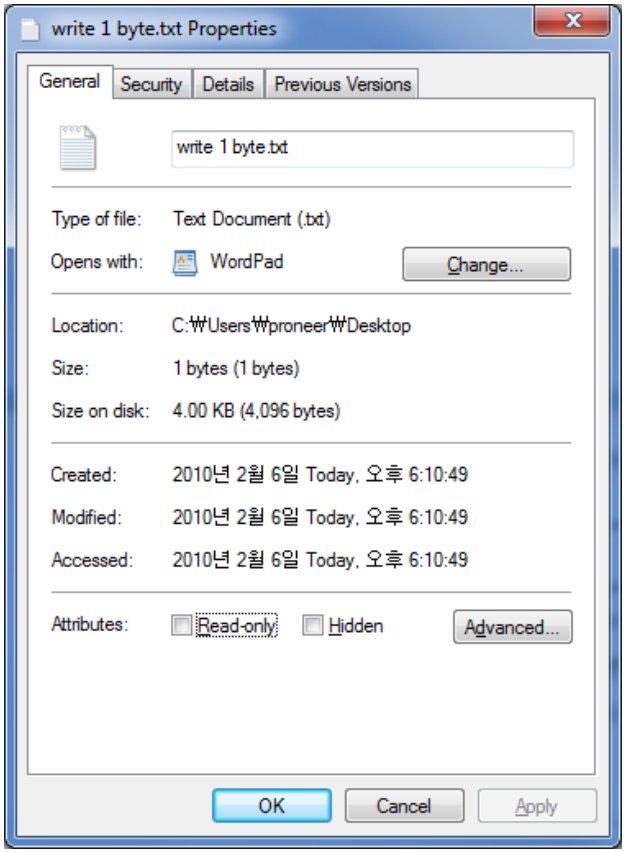
- 클러스터 크기 확인



클러스터와 블록

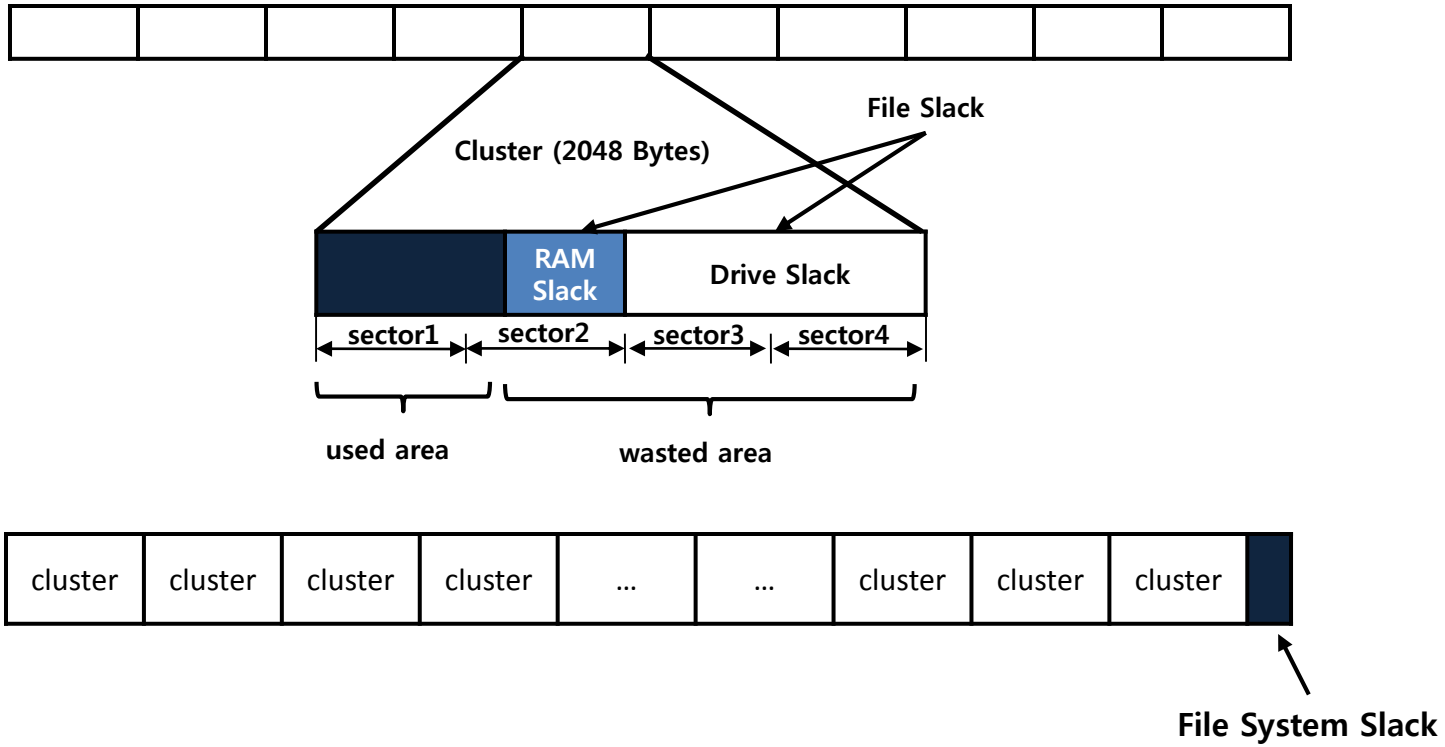
- I/O 효율 vs. 낭비되는 공간





슬랙 공간

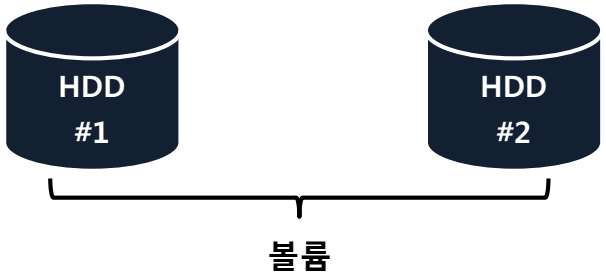
- 물리적 구조와 논리적 구조의 차이로 발생하는 낭비되는 공간



파티션과 볼륨

- **파티션**
  - 물리적으로 연속된 섹터들의 집합

- **볼륨**
  - 논리적으로 연속된 섹터들의 집합



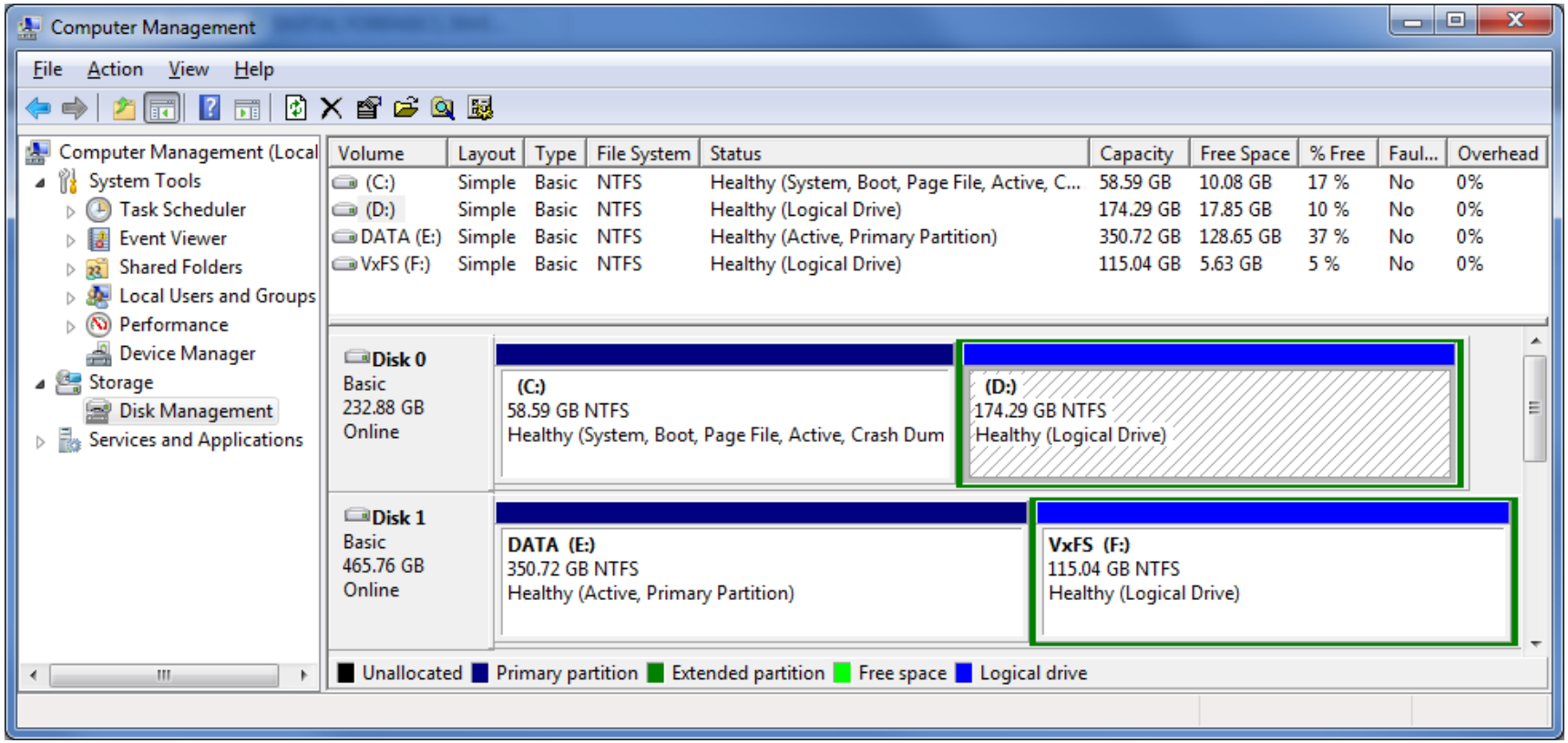
  - **파티션 ⊂ 볼륨**

파티션과 볼륨

- **파티션 사용의 이점**
  - 시스템 파티션과 구분하여 데이터 저장이나 백업용으로 사용
  - 하나의 시스템에 다양한 운영체제 설치 (멀티 부팅)
  - NTFS의 경우 MFT 크기의 감소로 성능 향상 (경우에 따라 다름)
  - 파일 탐색 시, 헤드 움직임 감소로 성능 향상 ➔ 탐색 시간 향상

- **동적 볼륨 사용의 이점 (LVM, Dynamic Disk…)**
  - 동적으로 사용량 증가 (저장매체 추가/제거 가능)
  - 대용량 저장소 사용

파티션과 볼륨


Computer Management
File Action View Help
Computer Management (Local
System Tools
Task Scheduler
Event Viewer
Shared Folders
Local Users and Groups
Performance
Device Manager
Storage
Disk Management
Services and Applications
Volume Layout Type File System Status Capacity Free Space % Free Faul... Overhead
(C:) Simple Basic NTFS Healthy (System, Boot, Page File, Active, C... 58.59 GB 10.08 GB 17 % No 0%
(D:) Simple Basic NTFS Healthy (Logical Drive) 174.29 GB 17.85 GB 10 % No 0%
DATA (E:) Simple Basic NTFS Healthy (Active, Primary Partition) 350.72 GB 128.65 GB 37 % No 0%
VxFS (F:) Simple Basic NTFS Healthy (Logical Drive) 115.04 GB 5.63 GB 5 % No 0%
Disk 0
Basic
232.88 GB
Online
(C:)
58.59 GB NTFS
Healthy (System, Boot, Page File, Active, Crash Dum
(D:)
174.29 GB NTFS
Healthy (Logical Drive)
Disk 1
Basic
465.76 GB
Online
DATA (E:)
350.72 GB NTFS
Healthy (Active, Primary Partition)
VxFS (F:)
115.04 GB NTFS
Healthy (Logical Drive)
Unallocated Primary partition Extended partition Free space Logical drive

Quiz!!

Quiz!!

MBR과 GPT

- 윈도우 7에서 MBR Slack의 크기는?
- MBR 부트 코드의 크기와 역할은?
- MBR 파티션 테이블에 저장할 수 있는 주 파티션 개수는?
- MBR 시그니처 값은 ?
- VBR의 크기는?
- VBR의 역할은?
- NTFS의 파티션 타입 코드값은?
- GPT를 사용하는 이유는?
- GPT에서 생성 가능한 주 파티션 개수는?

Quiz!!

FilE SYsTeM

- 파일시스템을 메타 영역과 데이터 영역으로 구분하는 이유는?
- 1섹터의 크기는?
- LBA 주소 지정 방식을 사용하는 이유는?
- 파일시스템에서 클러스터나 블록 단위를 사용하는 이유는?
- 클러스터 4K, 파일 크기 2K 일 때, 램슬랙과 드라이브 슬랙 크기는?
- 램슬랙의 특징은?
- 드라이브 슬랙의 특징은?
- 파티션과 볼륨의 차이는?
- 윈도우가 기본 인식(마운트)할 수 있는 파일시스템은?

질문 및 답변