Booting, Partitioning and File Systems

Nothing is what it seems

Karst Koymans, Jaap van Ginkel

Informatics Institute
University of Amsterdam
(version 1.20, 2012/10/01 13:37:34)

Tuesday, October 10, 2012

Table of Contents

Booting principles

PC (x86) bootstrap

Apple Mac (pre-Intel) bootstrap

Apple Intel and (future, non-BIOS) PC firmware

Hard disk geometry and addressing

"Secret" areas

Raw access

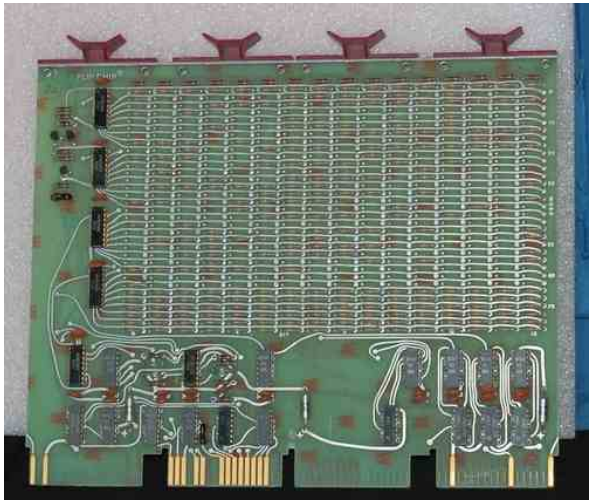
Part I

Booting and hard disks

Booting

- Short for bootstrapping
  - Building something out of nothing
    - Munch
  - Starting with
    - Minimal knowledge
    - Minimal structure

DEC Bootstrap diode board



Source: `http://decpicted.blogspot.nl/`

Booting progress

- Daisy chain of programs (more knowledge)
  - Boot ROM, firmware
  - Boot loaders (primary, secondary, tertiary, ...)
  - Boot managers
  - OS kernel
- Implementing access to data (more structure)
  - MBR, partition schemes
  - File systems, executable formats

Boot loader functionality

Definition (Boot Loader)

A boot loader (or bootstrap loader) is a simple program that has the task to load another, usually more complex, program and transfer control to it

- This is a staged, sequential, process:
  primary, secondary, tertiary ... bootstrap
- To start off this needs firmware (for instance BIOS) support
- The last step in the chain loads and executes
  the kernel of the desired operating system

Boot manager functionality

Definition (Boot Manager)

A boot manager is a program that offers choices to the user to bootstrap multiple operating systems and assists in the steps necessary to daisy chain the chosen environment

- Bootstrap managers can be complex and
  might need their own bootstrap loaders
- Bootstrap managers might have knowledge of the booting process
  of a certain kernel, but can also chainload a specific boot loader

Multiboot specification

- Boot sequence standard
  - Targeted to 32-bit and 64-bit
  - Targeted to x86 and PowerPC
- Well-defined interface between loader and OS
- Uses local storage and/or network
- Uses Multiboot header
  - Somewhere in the first 8KiB
  - Contains just enough information to load the OS without the need to know all executable formats

Intermezzo 1: $2^{10} \approx 10^3$

- But not precisely!!!
- $2^{10} = 1024$ and $10^3 = 1000$
- The difference is not very big...
  - ...but grows with each multiplication

Power of ten	Power of two	Increase in %
1000	1024	2.4
1000000	1048576	4.9
1000000000	1073741824	7.4
1000000000000	1099511627776	10.0

Intermezzo 2: SI prefixes

Prefix	Symbol	Value
kilo	k	10^3
mega	M	10^6
giga	G	10^9
tera	T	10^{12}
peta	P	10^{15}
...	...	...

Intermezzo 3: IEC binary prefixes

Prefix	Symbol	Value
kibi	Ki	2^{10}
mebi	Mi	2^{20}
gibi	Gi	2^{30}
tebi	Ti	2^{40}
pebi	Pi	2^{50}

- Examples
  - 1MiB (1 Mibibyte) is 1048576 bytes
  - 1Gb/s (1 Gigabit per second) is 1000000000 bits per second
- Also note the difference between B (byte) and b (bit)

Booting a BIOS-based PC

- Firmware is inside BIOS
  - Started from FFFF:FFF0
- Floppy boot sector
  - First sector is loaded at 0000:7C00
- Hard disks have no boot sector
  - MBR (Master Boot Record)
    - Contains boot code and partition information
  - First sector is also loaded at 0000:7C00
- CD-ROM booting is specified in El Torito, extending ISO 9660
- USB flash drives can be like floppy, hard disk or even CD-ROM

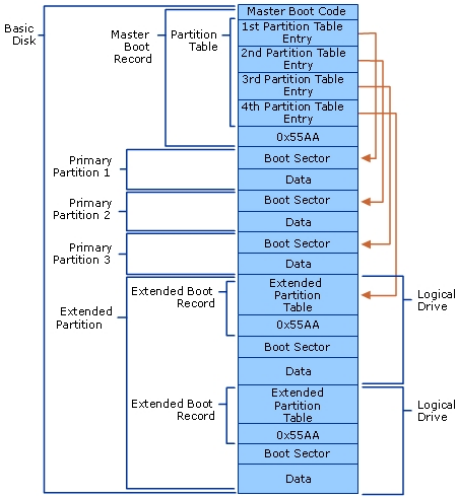
The Master Boot Record – MBR

- The Master Boot Record (MBR)
  - is the very first sector (LBA 0) on a disk
  - consists of
    - boot code
    - a partition table
    - a signature
  - is small (512 bytes) and the boot code within is just smart enough to parse the partition table

Contents of (primary and logical) partitions

- OS specific boot sector
- OS specific contents
  - File System
  - Kernel
- Normally independent of other partitions
  - Except for multiboot Windows setup...

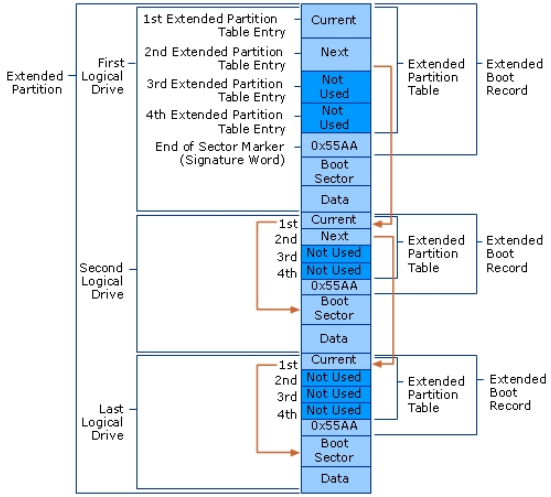
MBR – basic



Source: Microsoft Technet[1]

[1] The picture should use 0xAA55 in little endian order

MBR – extended



Source: Microsoft Technet

Standard Microsoft MBR

- Looks for the "active" partition and loads its boot sector
- Boot sector for Windows 95/98
  - loads IO.SYS, MSDOS.SYS
- Boot sector for Windows NT/2000/XP
  - loads ntldr, ntdetect.com
  - ntldr looks at other Microsoft partitions
    - recording results in boot.ini
- Boot sector for Windows Vista/7
  - loads BOOTMGR (Windows Boot Manager)
  - BOOTMGR accesses Boot Configuration Data (BCD)
  - BOOTMGR finally calls winload.exe to load ntoskrnl.exe

3rd party boot loaders and managers

- All can replace the standard MBR completely
  - Leave partitions alone
  - LILO (LInux LOader)
  - GRUB (GRand Unified Bootloader)
  - Boot Magic (commercial)
  - XOSL (eXtended OS Loader; graphical)
  - ...
- Are sometimes installed in the volume (partition) boot sector

Booting a (new world, pre-Intel) Macintosh

- Macs have "Open Firmware"
  - Sun started to use this in 1989 as OpenBoot on SparcStation 1
  - IEEE 1275 open standard
  - Based on Forth
  - Some implementations were open sourced in 2006
    - FirmWorks' Open Firmware
    - CodeGen's SmartFirmware
    - Sun's OpenBOOT

Why use Open Firmware?

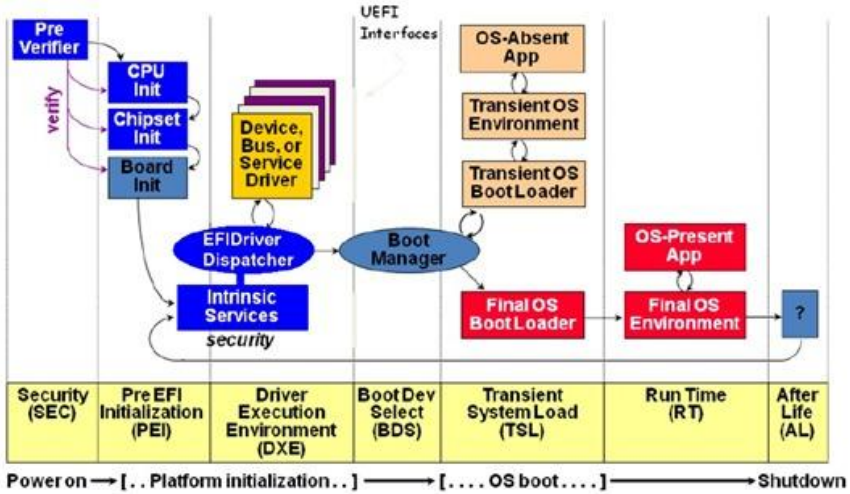
- Not tied to any particular
  - Processor family
  - Expansion bus
- Plug-in device drivers written in FCode
- Easy debugging
- Device-tree building and naming

Extensible Firmware Interface (EFI)

- Developed by Intel
- PC BIOS replacement
- Incorporates ACPI[2], SMBIOS[3]and boot loaders
- OS-independent EFI drivers
- Uses GPT (GUID Partition Table)

[2]Advanced Configuration and Power Interface
[3]System Management BIOS

UEFI secure boot



Source: `UEFIspecs`

Why are hard disks interesting?

- Most common storage device
- Central focus for (semi-)permanent data
  - Text files
  - Data files
  - Applications
  - Memory fragments (swap)
  - Temporary data
  - "Deleted" data

Hard Disk Structure

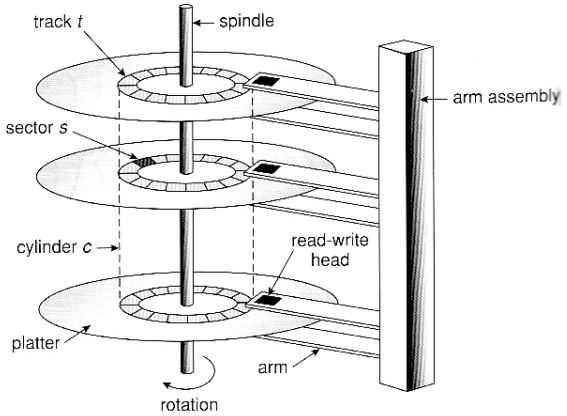


Figure 12.1 Moving-head disk mechanism.

Source: John Bell, University of Illinois, Chicago

Hard Disk Addressing – CHS

- Legacy format
  - Every sector is addressed by
    - A cylinder: $0 \leq C <$ number_of_cylinders (C_{max})
    - A head: $0 \leq H <$ heads_per_cylinder (H_{max})
    - A sector: $1 \leq S \leq$ sectors_per_track (S_{max})
  - Every sector contains 512 bytes (0.5 KiB)
- Modern format
  - Sectors are indexed as a (logical) array with a Logical Block Address (LBA)
    - $LBA = (C \times H_{max} + H) \times S_{max} + S - 1$

Hard Disk Addressing – complications

- In modern disks
  - There is no constant number of sectors per track (cylinder,head)
  - Bad blocks can be remapped to anywhere on the disk by firmware
  - Blocks can be interleaved for performance reasons
- CHS is just an illusion (legacy)
  - which is part of the original ATA specification
  - which is still used by classic BIOS "Int 13H"
  - and unfortunately is still part of many data structures

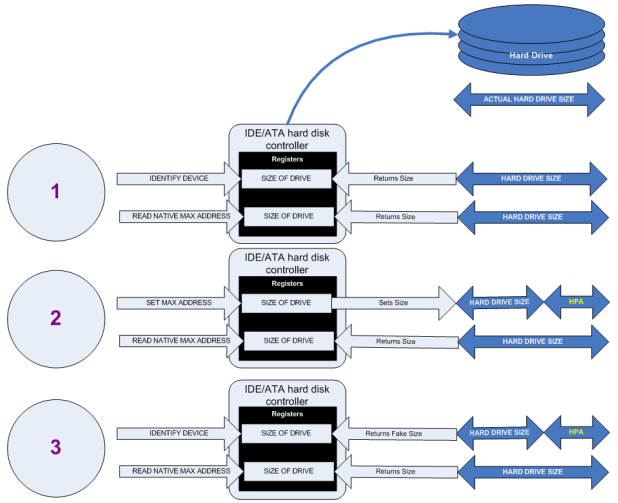
Hard Disk Addressing – interfering standards

- The ATA-1 specification used the following sizes
  - Cylinders 16 bits: $C_{max} = 65536$
  - Heads 4 bits: $H_{max} = 16$
  - Sectors 8 bits: $S_{max} = 255$ (no sector 0)
  - Maximum (total) capacity: $65536 \times 16 \times 255 = 267386880$ sectors
    (127.5 GiB, or almost 137 GB)
- Older BIOS's use the following sizes
  - Cylinders 10 bits: $C_{max} = 1024$
  - Heads 8 bits: $H_{max} = 256$
  - Sectors 6 bits: $S_{max} = 63$ (no sector 0)
  - Maximum (total) capacity: $1024 \times 256 \times 63 = 16515072$ sectors
    (7.875 GiB, or almost 8.5 GB)
- Worst case combination
  - Cylinders 10 bits, heads 4 bits, sectors 6 bits
  - Maximum (total) capacity: $1024 \times 16 \times 63 = 1032192$ sectors
    (504 MiB, or almost 529 MB)

ATA extensions

- ATA-1 introduces 28-bit logical addressing
- ATA-3 introduces security features (passwords)
- ATA-4 introduces the Host Protected Area
- ATA-6 introduces
  - 48-bit logical addressing
  - The Device Configuration Overlay
  - The removal of CHS-addressing

Host Protected Area – creation



Source:
`http://en.wikipedia.org/wiki/Host_Protected_Area`

Host Protected Area – HPA

A disk is not what it seems

- The HPA (Host protected Area) is a part at the (logical) end of the disk, which is not seen by the standard IDENTIFY_DEVICE command
- In order to see or change this extra space you need to
  - Issue the READ_NATIVE_MAX_ADDRESS command to find the real size of the disk
  - Issue the SET_MAX_ADDRESS command to enlarge the logical size
  - There are features to
    - Lock the status of the HPA until the next reset
    - Make changes to the HPA volatile

Device Configuration Overlay – DCO

It is not what it doesn't seem to be...

- Hard disk firmware has yet another surprise
  - The IDENTIFY_DEVICE command does not really identify the device
  - The DEVICE_CONFIGURATION_IDENTIFY command comes closer
    - It may tell you that the disk is even bigger than what READ_NATIVE_MAX_ADDRESS tells you!
- To create, change or remove a DCO one can use
  - The DEVICE_CONFIGURATION_SET command (create/change)
  - The DEVICE_CONFIGURATION_RESET command (remove)
- There is a feature to
  - Freeze the DCO configuration command until next power up

Sector extractor tools

dd command

```
dd if=<input file or device; default standard input>
   of=<output file or device; default standard output>
   bs=<blocksize; default 512 (bytes)>
   skip=<number of blocks to skip; default 0>
   count=<number of blocks to read; default infinite>
```

dd variants

- There are some variants of dd with special properties
  - Two similar GNU tools (unrelated)
    - `dd_rescue`: has better error handling
    - `ddrescue`: has better error handling
  - Three tools that are products of DoD's Cyber Crime Center
    - `dccidd`: calculates hashes on the fly
    - `dcfldd`: newer version of dccidd
    - `dc3dd`: calculates hashes on the fly

Hexdump tool

xxd command

```
xxd [<input file> [<output file>]]
    -b (use bits in stead of nibbles)
    -c <number of columns; default 16>
    -g <number of bytes to group; default 2>
    -r (revert: create binary from hexdump)
```

Part II

Partitioning

Table of Contents

Volumes and partitions

DOS partitions

Acquisition

Apple partitions, *BSD disklabels, Solaris slices

GPT disks

Terminology

- A volume is a sequence of sectors
  - which do not need to be consecutive in a physical sense
  - which has a logical consecutive numbering scheme
  - sometimes differentiating between physical and logical volumes
- A partition scheme is a subdivision of a volume
  - in multiple parts called partitions
  - which are consecutive within the volume
  - which can be considered volumes themselves and hence can be partitioned themselves

Partition properties

- Partitions may overlap
  - Most partitions are disjoint
  - If not, in most cases one is a subset of the other
- Partitions need not cover the complete volume
  - Sectors may belong to metadata like partition tables or disklabels
  - Sectors may belong to unallocated or unused space

Master Boot Record structure (1)

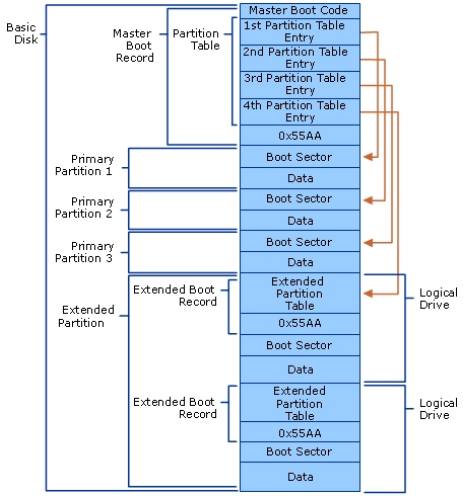
- Master boot code
  - is loaded at address 0000:7C00 by the firmware
- Partition table
  - Up to 4 primary partitions
  - Optionally 1 of these 4 is an extended partition

Master Boot Record structure (2)

MBR parts

Position	Hex	Content
0..445	0x0..0x1bd	Boot code
446..461	0x1be..0x1cd	Partition Table Entry #1
462..477	0x1ce..0x1dd	Partition Table Entry #2
478..493	0x1de..0x1ed	Partition Table Entry #3
494..509	0x1ee..0x1fd	Partition Table Entry #4
510..511	0x1fe..0x1ff	Signature (0xaa55; little endian)

MBR – basic



Source: Microsoft Technet[1]

[1] The picture should use 0xAA55 in little endian order

Primary partition table entry structure

Entry parts

Position	Content
0..0	Bootable flag
1..3	Starting CHS address[4]
4..4	Partition Type
5..7	Ending CHS address
8..11	Starting LBA address
12..15	Partition size (in sectors)

[4] The format is $H_{7-0}C_{9-8}S_{5-0}C_{7-0}$

Partition types

Common types

Type	Meaning
0x00	Empty (unused slot)
0x05	Extended (CHS format)
0x07	NTFS
0x0c	FAT32 (LBA format)
0x0f	Extended (LBA format)
0x82	Solaris x86 / Linux swap
0x83	Linux (generic)

Partition types – continued

Common types – continued

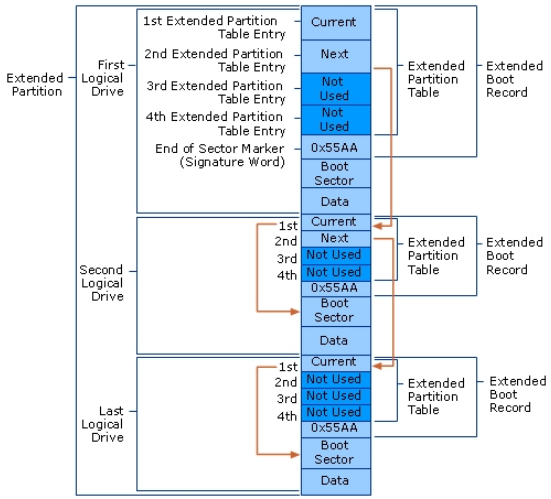
Type	Meaning
0xa5	FreeBSD
0xa6	OpenBSD
0xa8	Mac OS X
0xa9	NetBSD
0xab	Mac OS X boot
0xee	EFI (legacy protection)

See also `http://www.win.tue.nl/~aeb/partitions/`

Extended ("logical") partitions

- Extended partition table entry
  - At most 1
  - Replaces a primary partition table entry
  - An extended partition contains logical partitions described by Extended Boot Records

MBR – extended

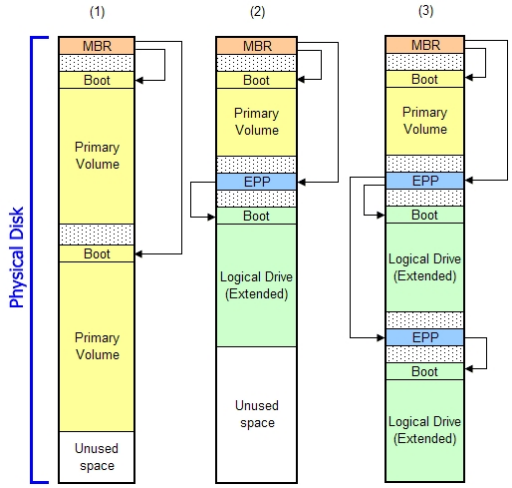


Source: Microsoft Technet

Extended Boot Record (EBR)

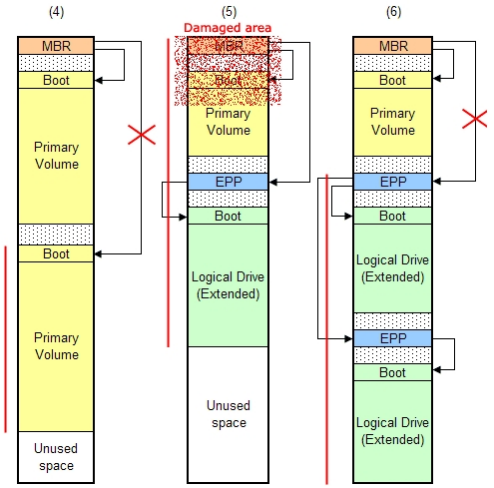
- One block of 512 bytes, just like the MBR
- No master boot code (unused space)
- "Partition table"
  - Information about start and length of logical partition
    - Start is always relative to the secondary extended partition
  - Pointer to the next EBR, also in the form of start and length
    - Start is always relative to the primary extended partition
- The start of the first secondary extended partition coincides with
  the start of the primary extended partition

Normal acquisition



Source: `http://www.z-a-recovery.com/`

Damaged acquisition



Source: `http://www.z-a-recovery.com/`

Data acquisition

- BIOS access or direct access?
  - Directly talking to the disk is best
  - BIOS may get the disk geometry wrong for some reason
- Dead or live acquisition?
  - Dead acquisition may use hardware of the suspect system (but not the OS)
  - Live acquisition must use hardware of the suspect system (and also the OS)
- Use write blockers!
  - Hardware (do they take care of HPA and DCO?)
  - Software (replacing BIOS "Int 13H" or inside OS drivers)

Apple Partition Scheme

- Block 0 is the Driver Descriptor Map
- Block 1-N (often N=63) is the Apple Partition Map
  - Each entry describes a partition of the disk
  - The partition map contains an entry for itself
  - All blocks except block 0 should belong to exactly 1 partition

BSD disklabels and boot code

- *BSD disklabels are inside primary or logical partitions, numbering from the start of this partition
  - Sector 0 is boot code
  - Sector 1 is the disklabel
  - Sector 2 often continues the boot code
  - From sector 16 onwards we typically see the file systems themselves

BSD disklabel structure

Interesting fields

Position	Meaning
0..3	Signature (0x82564557; little endian)
40..43	Size of a sector in bytes (512)
132..135	Signature again (0x82564557)
138..139	Number of partitions
148..163	BSD disklabel partition entry #1
⋮	⋮
388..403	BSD disklabel partition entry #16

BSD disklabel partition entry structure

Partition entry parts

Position	Content
0..3	Size of BSD partition (sectors)
4..7	Starting sector of BSD partition
12..12	Partition Type

- The starting sector is relative to the start of the whole disk!
- The remaining space is used for file system parameters like fragment size, fragments per block and cylinders per group

BSD partition types

Common types

Type	Meaning
0x00	Empty (unused slot)
0x01	Swap
0x07	4.2BSD fast file system (FFS)
0x08	MSDOS file system (FAT)
0x09	4.4BSD log file system (LFS)
0x0a	Used (unsupported)
0x0c	CDROM (ISO9660)

Differences between FreeBSD and other BSDs

- FreeBSD
  - refers in the disklabel only to partitions within the DOS partition it is part of
  - interprets the DOS partition entries itself and uses "disk slices" to refer to those DOS partitions
- OpenBSD and NetBSD
  - refer in the disklabel also to partitions outside the DOS partition it is part of
  - do not interpret or use the DOS partition entries themselves and integrate outside partitions within the total set of disklabels

Some more confusion with Solaris slices

- What others call a partition, Solaris often calls a "slice"
- Solaris 9 and later optionally use the EFI partition scheme
- Solaris 8 and earlier use a disklabel-like scheme
- SPARC systems put the disklabel right in front
  - Sector 0 is the disklabel
  - Sector 1-15 contains the boot code
- i386 systems follow the BSD disklabel convention
  - Sector 1 is the disklabel
  - Boot code is in a separate DOS partition (type 0x82)

Solaris x86 disklabel structure

Interesting fields

Position	Meaning
12..15	Signature (0x600ddeee; little endian)
28..29	Size of a sector in bytes (512)
30..31	Number of partitions
72..83	Solaris partition #1
⋮	⋮
252..263	Solaris partition #16
508..509	Signature (0xdabe; little endian)

Solaris x86 partition entry structure

Partition entry parts

Position	Content
0..1	Partition Type
4..7	Starting sector of Solaris partition
8..11	Size of Solaris partition (sectors)

- The starting sector is now again relative to the start of the DOS partition!

Solaris x86 partition types

Common types

Type	Meaning
0x00	Unassigned (unused slot)
0x01	/boot
0x02	/
0x03	swap
0x04	/usr
0x05	Entire DOS partition
0x07	/var
0x08	/home

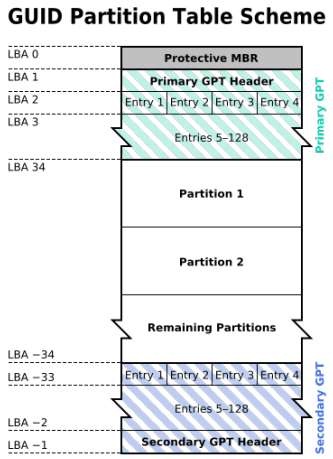
Solaris likes to give the mount point in stead of the filesystem type as other systems do

GUID Partition Table (GPT)

- Block 0 is the Legacy MBR or Protective MBR
- Block 1 is the (Primary) Partition Table Header
- Blocks 2-33[5] are the (Primary) Partition Entries (128 items)
- Primaries are repeated as secondaries at the end of the disk for redundancy purposes

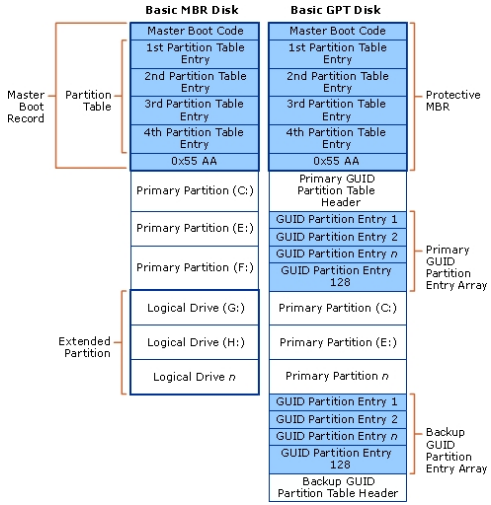
[5] These numbers and sizes are for the Windows environment. The EFI specification is more general.

GPT – Wikipedia



Source: Wikipedia

MBR versus GPT



Source: Microsoft Technet

Legacy (Protective) MBR

- A normal DOS partition table is used
- There is only one used entry in this table
- This entry has type 0xee, which means GPT partition
- The corresponding primary partition covers the remainder of the disk, starting from sector 1

Partition Table Header (GPT Header)

- Disk GUID
- Size and location of itself and secondary table
- Size and location of the partition area
- CRC32 checksum for itself and the partition table entries

Partition Table Header (GPT Header) – detail

GPT Header

Position	Meaning
0..7	Signature (0x4546492050415254; big endian)
24..31	LBA of this GPT header
32..39	LBA of backup GPT header
40..47	LBA of start of partition area
48..55	LBA of end of partition area
56..71	Disk GUID
72..79	LBA of start of partition table
80..83	Number of partition table entries

Partition Entry

- Partition Type GUID (16 bytes)
- Unique Partition GUID (16 bytes)
- Starting LBA (8 bytes)
- Ending LBA (8 bytes)
- Attribute Bits (8 bytes: 64 bits)
- Partition Name (72 bytes: 36 unicode chars (UTF-16))
- Total size of each entry is 128 bytes
- Each sector contains 4 of these entries
- Hence there is a maximum of 32*4=128 partitions