MTD

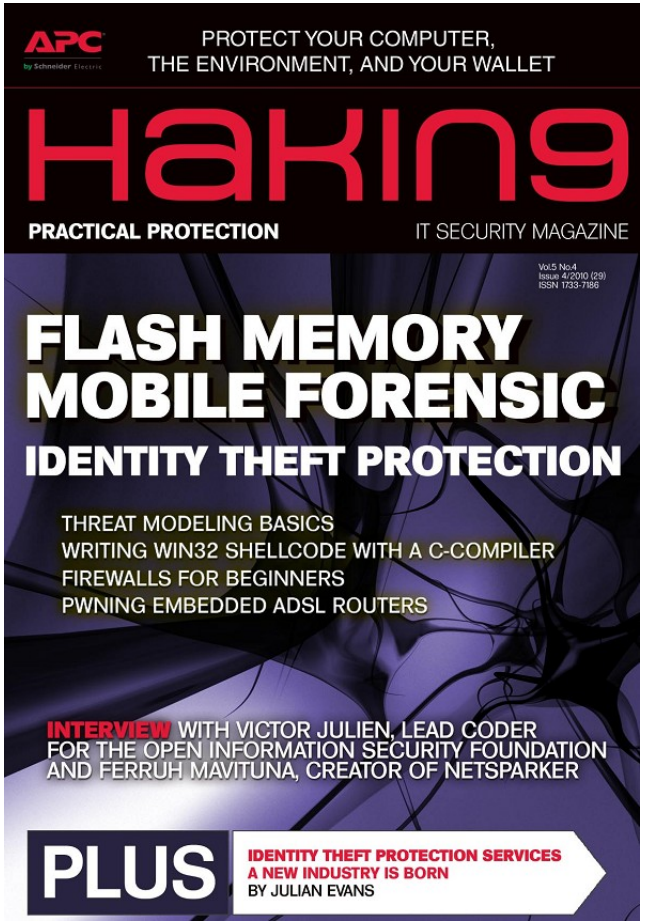
Memory Technology Devices

http://www.linux-mtd.infradead.org/

Mobile phones

Memory technologies

MMC, eMMC, SD & UFS

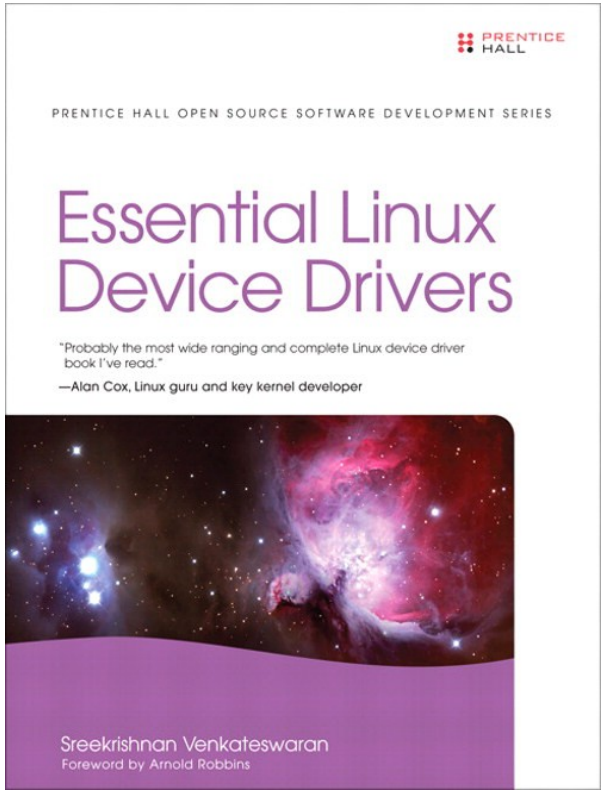


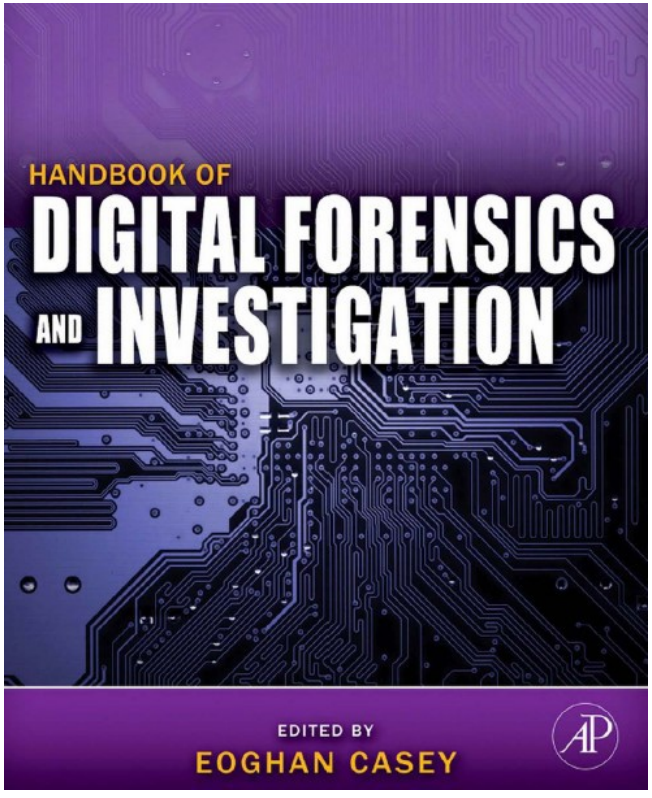
Good reads

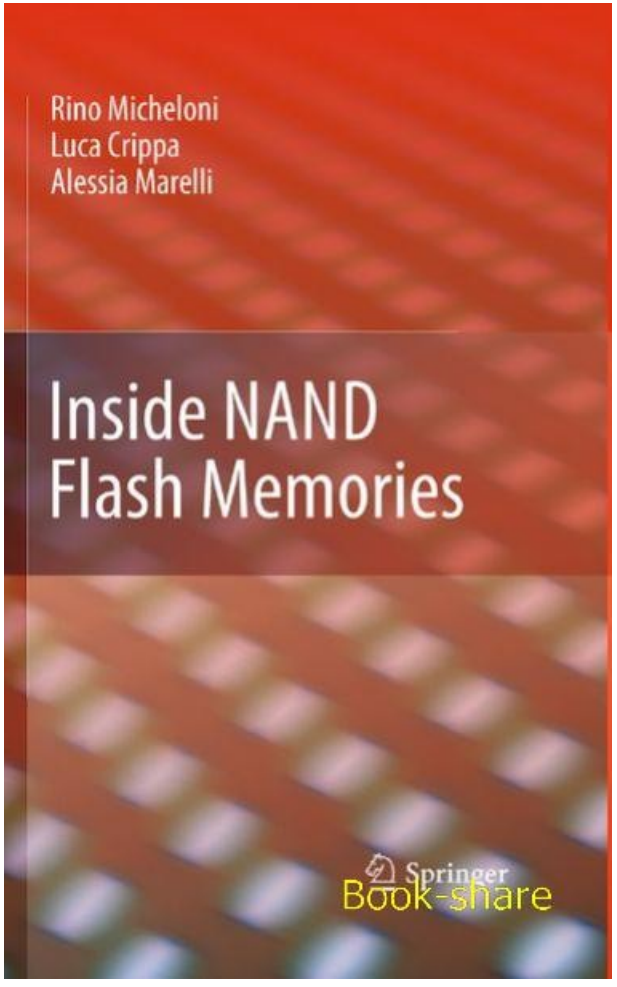
- The ”Flash memory mobile forensic” article must be read!
- http://www.informit.com/store/product.aspx?isbn=0132396556

Online Sample Chapter 17
[server]\embedded_forensics\
docs\common_documents

Ch 8







Memory types 1

- Read only Memory (ROM)
  - A nonvolatile memory in which data are deposited during the production process and thereafter can only be read. ROM is used to store software and other static data
- Programmable ROM (PROM)
  - PROM is nonvolatile memory in which data are deposited not during the production process but at a later programming stage and thereafter only can be read
  - Erasable PROM (EPROM)
    - Erased as a whole with UV light
  - Electrically erasable PROM (EEPROM)
    - Induvidual bytes can be erased
- Flash
  - Flash is an EEPROM in which data can only be erased in blocks (up to one million times for the most modern types)

Memory types 2

- Random Access Memory (RAM)
  - RAM is volatile memory that can be written to as well as read. RAM can be divided into Dynamic RAM (DRAM needs to be refreshed periodically) and Static RAM (SRAM does not need to be refreshed).
  - In embedded systems, RAM is used as working memory and in combination with a battery for nonvolatile storage of dynamic data.
- Ferro electric (FeRAM) and Magnetoresistive (MRAM) RAM
  - FeRAM and MRAM are new, **nonvolatile (NV)** memories with the read and write characteristics of DRAM.
  - For not only embedded systems, FeRAM and especially MRAM has the potential in the future to replace all other types of memory as universal memory.
  - http://en.wikipedia.org/wiki/Random-access_memory

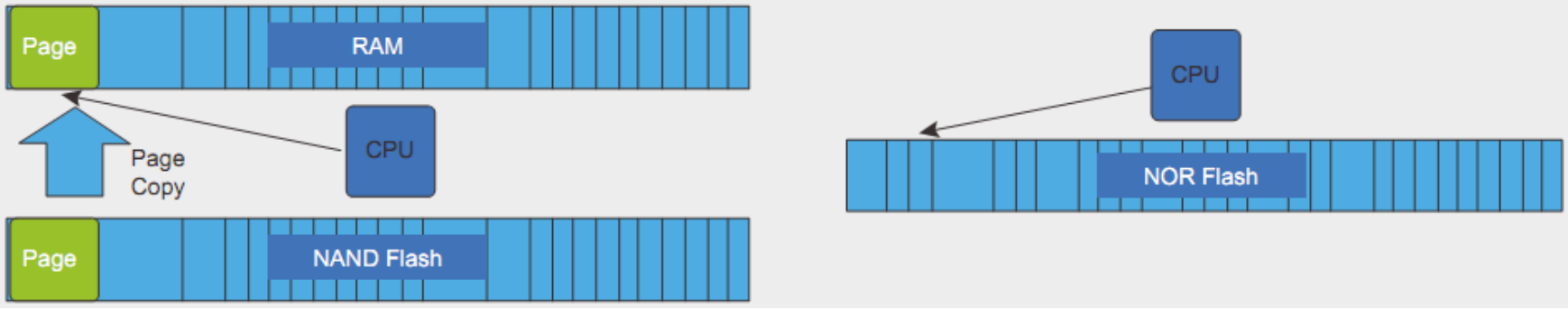
Table 8.1 Main Characteristics of Different Memory Types

Characteristics	ROM	PROM	EPROM	EEPROM	FLASH	DRAM	SRAM	FeRAM
Density	6	1.5	4	1.5	6	4	1	5
Retention time	∞	∞	10 years	10 years	10 years	0	0	10 years
Rewritable	n/a	1	100	10^6	10^6	∞	∞	10^{16}
Writing speed	n/a	--	-	-	+	++	++	++
Reading speed	+	++	+	+	+	+	++	+

From: Handbook of Digital Forensics and Investigation, ch8

Flash memory 1

- Flash memory chips generally come in two flavors: NOR and NAND
- NOR is the variety used to store firmware images on embedded devices, whereas NAND is used for large, dense, cheap, but imperfect storage as required by solid-state mass storage media such as USB key drives
- NOR flash chips are connected to the processor via **address and data lines** like normal RAM and therefore is read fast, but NAND flash chips are interfaced using **I/O and control** lines
- Code resident on NOR flash can be eXecuted In Place (XIP), but code stored on NAND flash has to be copied to RAM before execution



Flash memory 2

- In older cell phones NOR memory is used as the code storage media since it suffers from low write and erase speed
  - Things such as the OS, default applications etc.
- While the cheaper, higher capacity NAND is used for storing user data such as pictures, videos and the like
  - These days however most - if not all storage is NAND

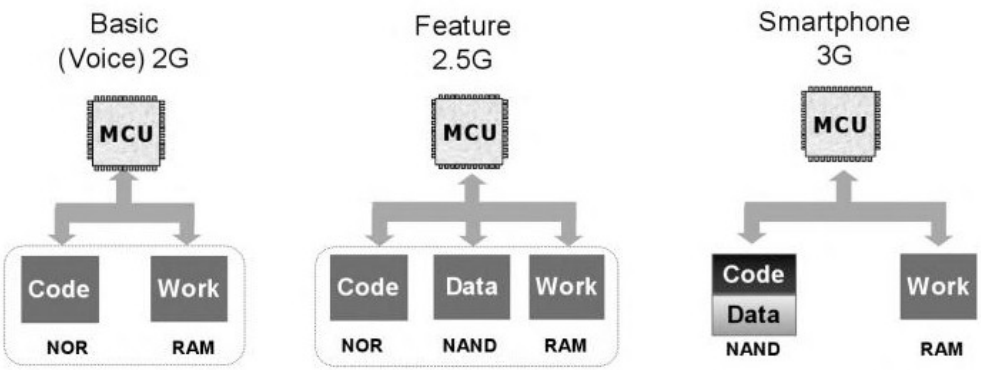


Fig. 1.13. Memory system architectures in mobile phones

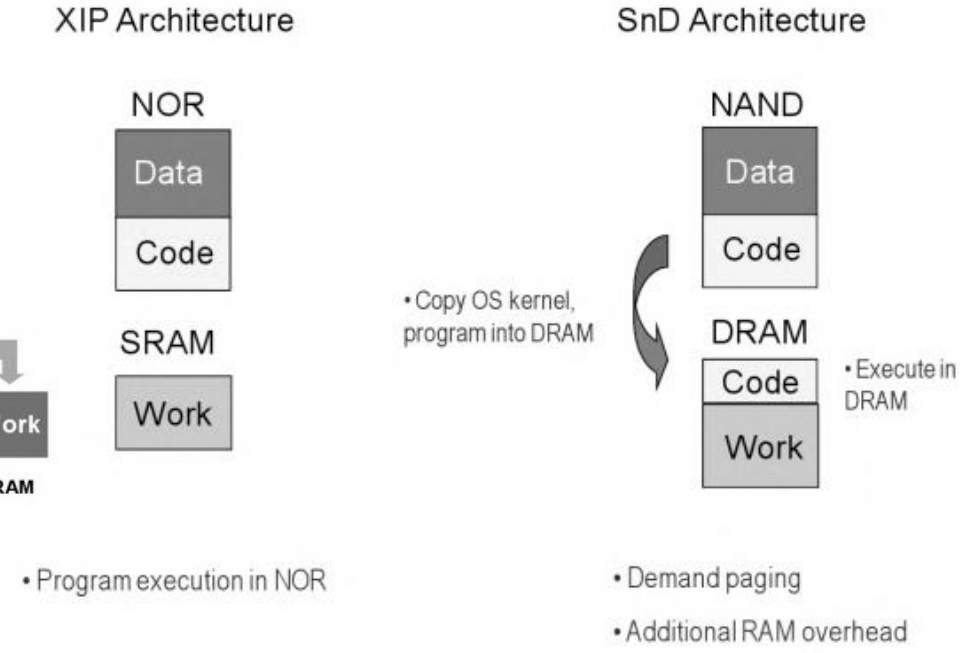


Fig. 1.14. Store and download versus XIP architecture

Flash memory 3

- Flash (NAND) memory also requires that there is some sort of flash memory management in place
- Flash memory is often represented at the system level as an ATA disk with a FAT (or other) file system
  - In this case a NFTL (NAND Flash Translation Layer) driver is needed - which either is embedded in **hardware** (managed nand) or with external logic by the **OS** or by an **external controller** (raw nand)
- Other file systems specially suited for flash
  - JFFS* (Journaling Flash File System)
  - YAFFS* (Yet another Flash File System)
  - LogFS
  - Nokia PM records?
  - Sony Ericsson GDFS (Global Data File System)
  - Samsung RFS (Robust File System)
  - UBIFS (Unsorted Block Image File System), Nokia N900
  - F2FS (Flash-Friendly File System)
    - http://en.wikipedia.org/wiki/F2FS

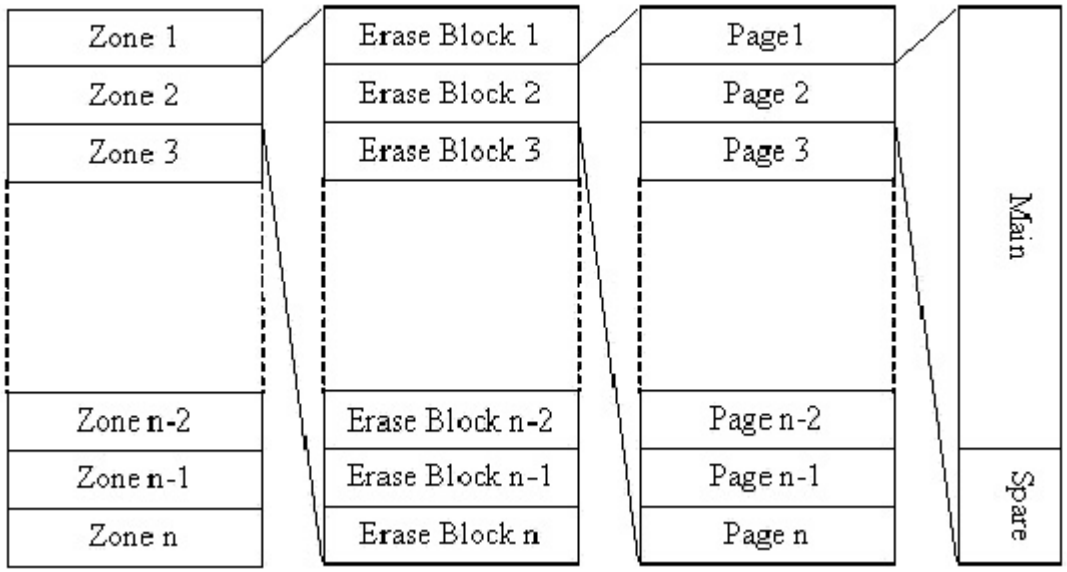
NAND flash 1

- Flash memory can be written to byte for byte (a page is however programmed), but it has to be erased in blocks at a time before it can be re-written
  - Erasing results in a memory block that is filled completely with 1's (FF)
- Erase blocks are divided into pages, for example 64 per erase block
- A page is usually a multiple of 512 bytes in size, to emulate 512 byte sector size commonly found in file systems on magnetic media
- Additionally, a page has a number of so called 'spare area' bytes, generally used for storing meta data

Some flash disk drivers use the concept of zones or LUNs (Logical Unit). A zone is a group of blocks, usually 256 to 1024.

Contrary to blocks and pages, a zone is just a logical concept, there is no physical representation.

From:
SSDDFJ_V1_1_Breeuwsma_et_al.pdf



NAND flash 2

- The spare area can contain information on the status of the block or the page. For instance when a block turns bad (worn out), it will be marked here
- The spare area can also contain ECC data and information necessary for the physical (flash) to logical (fs) address mapping
- With the ECC data an error of one bit can be corrected, after which the block will be marked bad
- NAND usually has bad blocks already when leaving the factory
- Because of these constraints, flash memory is best used with device drivers and filesystems that are tailored to suit them
- On Linux, such specially designed drivers and filesystems are provided by the MTD (Memory Technology Device) subsystem.

EXAMPLE SPARE AREA SIZES FOR DIFFERENT PAGE SIZES (IN BYTES)

Page Size	Spare area size	Total page size	Block size
256	8	264	8448
512	16	528	16896
2048	64	2112	135168

From SSDDFJ_V1_1_Breeuwsma_et_al.pdf

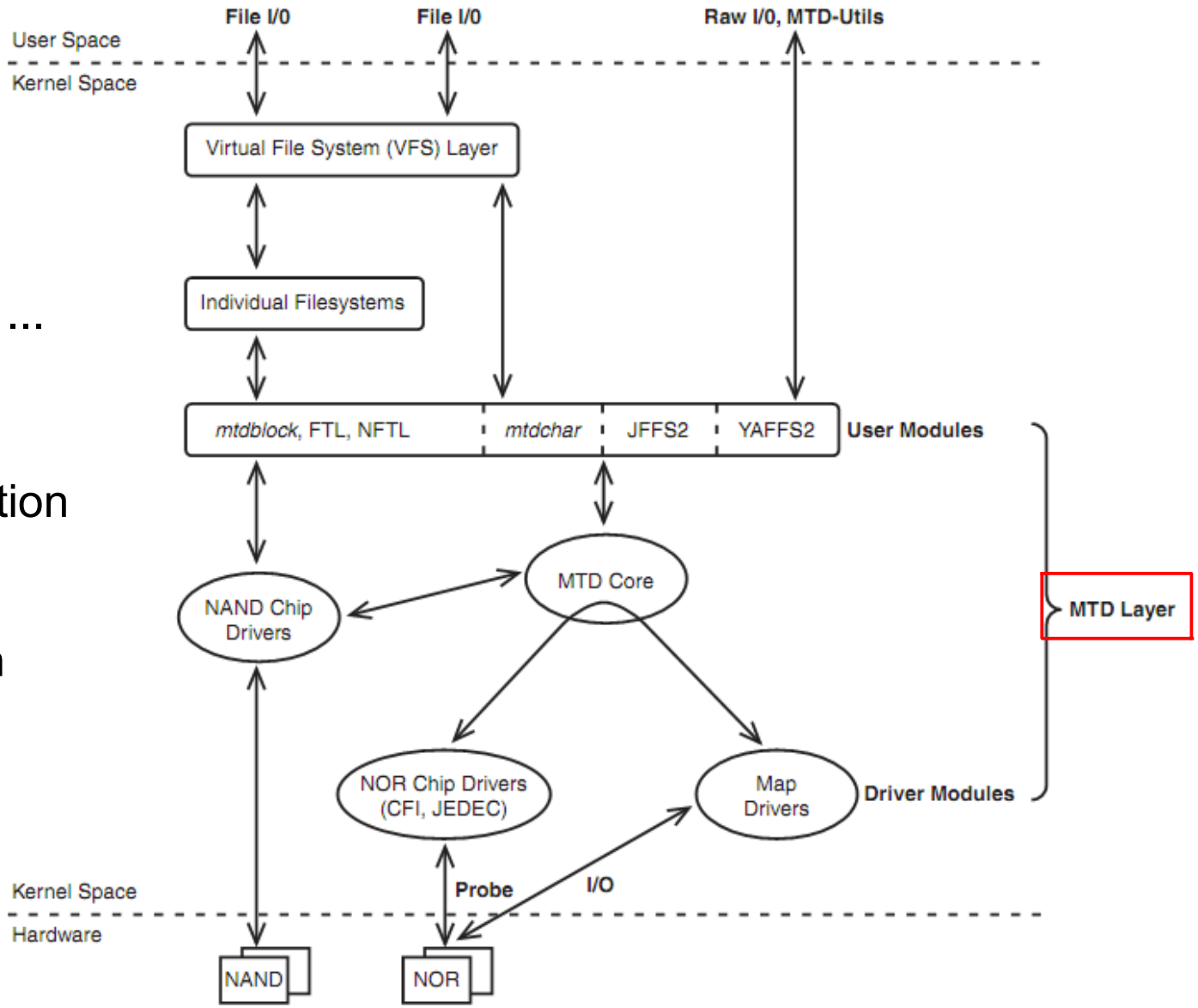
Linux-MTD Subsystem 1

- Linux has penetrated the embedded space and manifested itself in many electronic devices (even medical-grade)
- The kernel's MTD subsystem shown on next slide provides support for flash and similar non-volatile solid-state storage. It consists of the following:
  - The **MTD core**, which is an infrastructure consisting of library routines and data structures used by the rest of the MTD subsystem
  - **Map drivers** that decide what the processor ought to do when it receives requests for accessing the flash
  - **NOR Chip drivers** that know about commands required to talk to NOR flash chips
  - **NAND Chip drivers** that implement low-level support for NAND flash controllers
  - **User Modules**, the layer that interacts with user-space programs
  - Individual device drivers for some special flash chips

Linux-MTD Subsystem 2

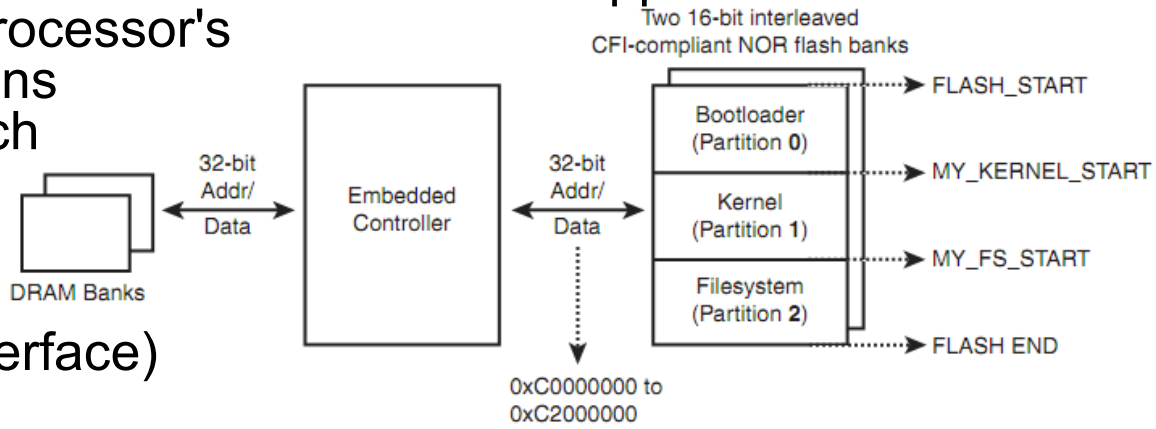
NFS, ext*, VFAT, ...

FTL (Flash Translation Layer)
and
NFTL (NAND Flash Translation Layer)



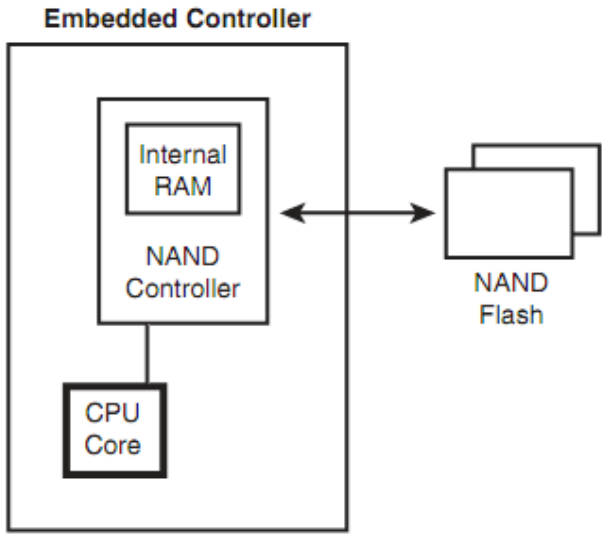
Map Drivers

- To MTD-enable your device, your first task is to tell MTD how to access the flash device. For this, you have to map your flash memory range for CPU access and provide methods to operate on the flash
- The next task is to inform MTD about the different storage partitions residing on your flash. Unlike hard disks on PC-compatible systems, flash-based storage does not contain a standard partition table on the media. Because of this, disk-partitioning tools such as fdisk and cfdisk2 cannot be used to partition flash devices. **Instead, partitioning information has to be implemented as part of kernel code**
- These tasks are accomplished with the help of an MTD map driver
- Example: The flash has a size of 32MB and is mapped to 0xC0000000 in the processor's address space. It contains three partitions, one each for the bootloader, the kernel, and the root filesystem
- CFI (Common Flash Interface)



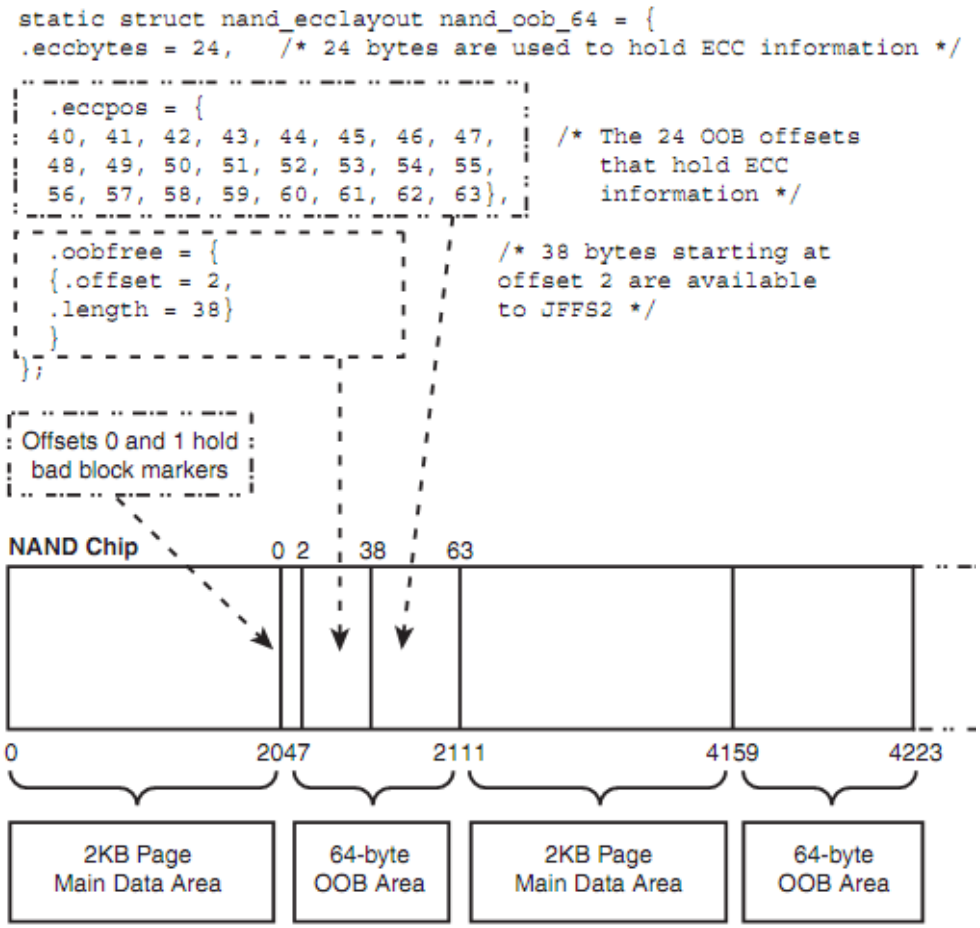
NAND Chip Drivers

- External NAND technology emulate standard storage interfaces such as SCSI or IDE over NAND flash - but On-board NAND flash chips need special drivers
- To read data from NAND flash, the CPU issues an appropriate read command to the NAND controller
- The controller transfers data from the requested flash location to an internal RAM memory, also part of the controller
- The data transfer is done in units of the flash chip's page size (for example, 2KB). In general, the denser the flash chip, the larger is its page size
- Note that the page size is different from the flash chip's block size, which is the minimum erasable flash memory unit (for example, 128 KB)
- After the transfer operation completes, the CPU reads the requested NAND contents from the internal RAM
- Writes to NAND flash are done similarly, except that the controller transfers data from the internal RAM to flash



NAND spare area

- There is another characteristic that goes hand in hand with NAND memory
- NAND flash chips, unlike NOR chips, are not faultless. It's normal to have some problem bits and bad blocks scattered across NAND flash regions
- To handle this, NAND devices associate a spare area with each flash page (for example, 64 bytes of spare area for each 2KB data page)
- The spare area contains Out-Of-Band (OOB) information to help perform bad block management and error correction
- The OOB area includes Error Correcting Codes (ECCs) to implement error correction and detection
- ECC algorithms correct single-bit errors and detect multibit errors



User Modules

- After you have added a map driver and chosen the right chip driver, you're all set to let higher layers use the flash
  - User-space applications that perform file I/O need to view the flash device as if it were a disk, whereas programs that desire to accomplish raw I/O access the flash as if it were a character device
  - The MTD layer that achieves these and more is called User Modules
- Block Device Emulation (disk)
  - The MTD subsystem provides a block driver called mtdblock that emulates a hard disk over flash memory. You can put any filesystem, say EXT2, over the emulated flash disk
  - Device nodes created by mtdblock are named /dev/mtdblock/X, where X is the partition number
  - The File Translation Layer (FTL) and the NAND File Translation Layer (NFTL) perform a transformation called wear leveling. Flash memory sectors can withstand only a finite number of erase operations (in the order of 100,000). Wear leveling prolongs flash life by distributing memory usage across the chip
  - Both FTL and NFTL provide device interfaces similar to mtdblock over which you can put normal filesystems. The corresponding device nodes are named /dev/nftl/X, where X is the partition number. Certain algorithms used in these modules are patented

User Modules and MTD-Utils

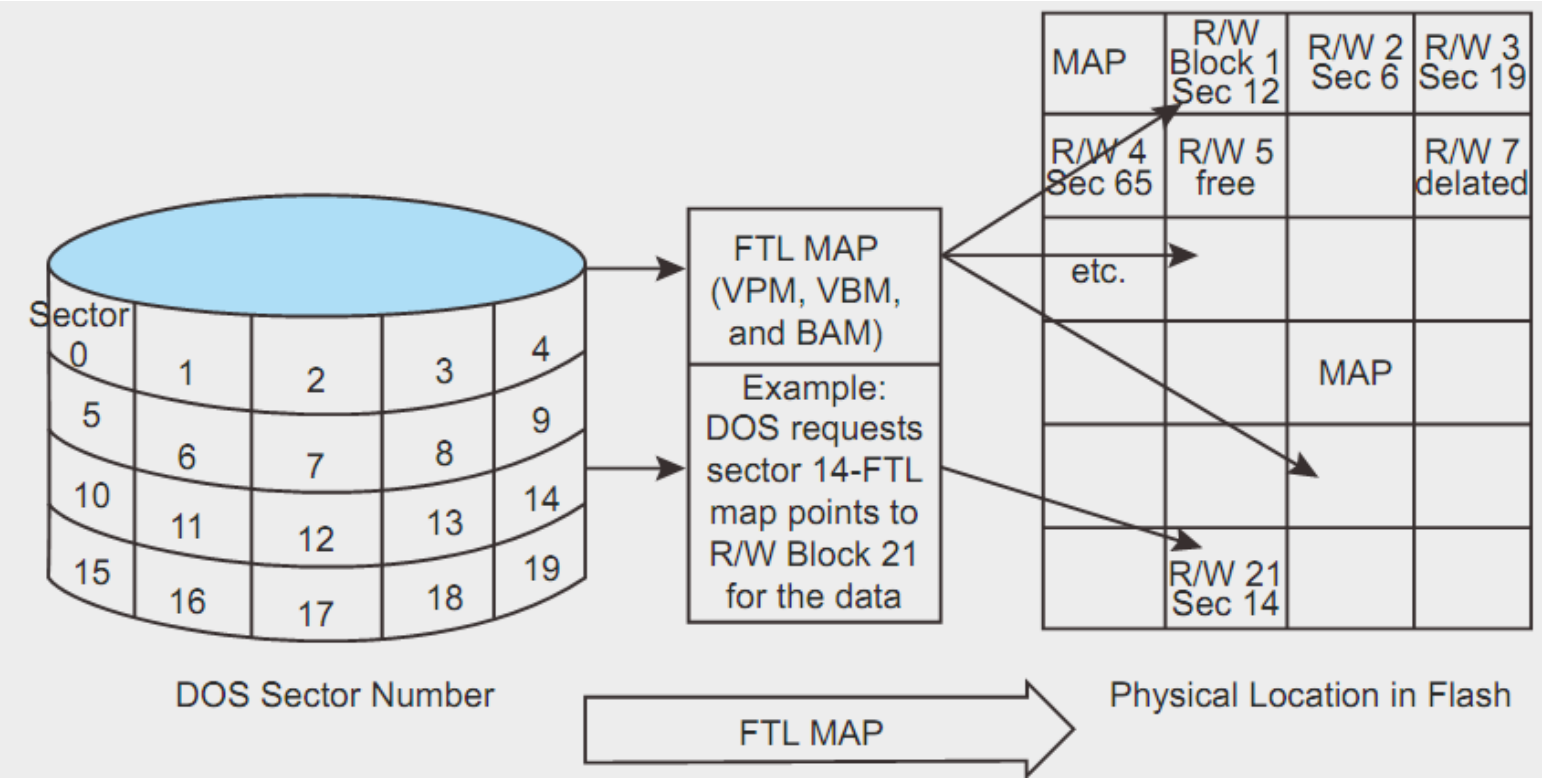
- Char Device Emulation (raw)
  - The mtdchar driver presents a linear view of the underlying flash device, rather than the block-oriented view required by filesystems. Device nodes created by mtdchar are named /dev/mtd/X, where X is the partition number
  - Example use of a raw mtdchar partition is the **bootloader** partition, hold POST (Power On Self Test) error logs and store system information as Vital Product Data (VPD)
- The MTD-utils package, contains several (20+) useful tools that work on top of MTD-enabled flash memory. Examples of included utilities are flash_info, nanddump, jffs2dump etc.
- Because NAND chips may contain bad blocks, use ECC-aware programs such as nandwrite and nanddump to copy raw data, instead of general-purpose utilities, such as dd
- To obtain the erase size (block size) of your flash partitions

```
$ cat /proc/mtd
dev:    size    erasesize   name
mtd0: 05180000  00020000  "system"
mtd1: 04000000  00020000  "userdata"
mtd2: 04000000  00020000  "cache"
```

0x20000 = 128kB

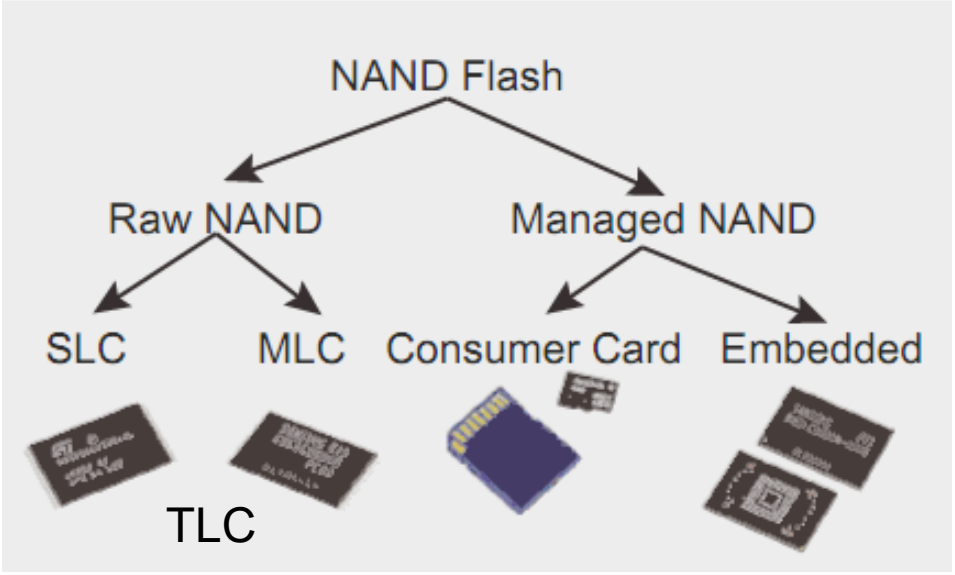
FTL sector relocation MAP

- VPM (Virtual Page Map)
- VBM (Virtual Block Map)
- BAM (Block Allocation Map)



NAND flash types

- When the FTL logic and relative functions are embedded in the NAND as one package, the flash is categorized as **managed** NAND
- When FTL is under care of host filesystem (the logic is external to the NAND) then the flash is said to be **raw** NAND
- Raw NAND contains just the flash memory array and a Program/Erase/Read (P/E/R) controller
- For forensic analysis, it is fundamental considering difference between raw and managed NAND, with particular regard to effects of reclaim and bad block management



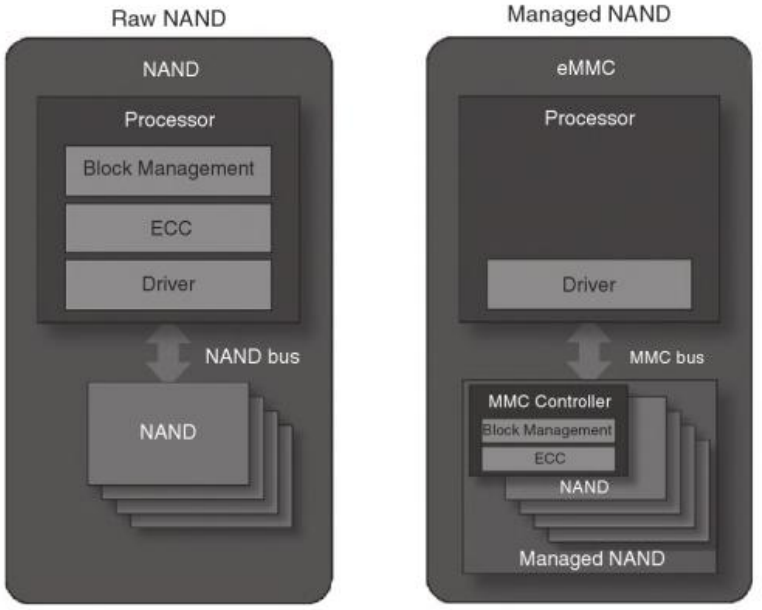


Fig. 1.15. Managed NAND versus raw NAND (Source: Micron Technology)

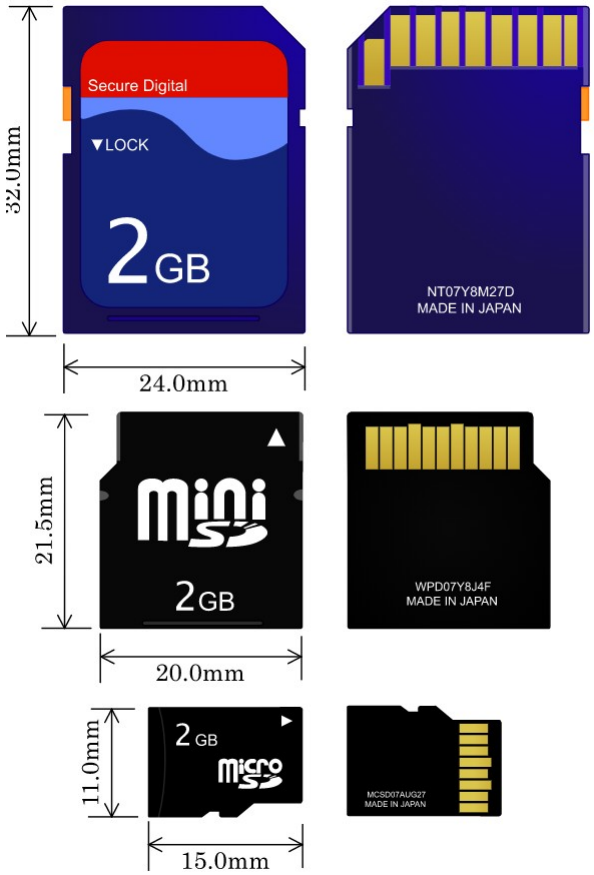
MMC, eMMC, SD and UFS (1)

- MultiMediaCard (MMC) is a memory standard for portable devices
  - SD (Secure Digtal) cards are an improvement to MMC
- Most if not all devices since 2012/2013 use (embedded) eMMC
- The eMMC architecture puts the MMC components (flash memory plus controller) into a small ball grid array (BGA) IC package for use in circuit boards as an embedded non-volatile memory system
- Usually all device partitions reside on the same shared eMMC
  - Some storage areas use the same partition as for example /data and /storage/emulated but with different mount points in the file system
  - Storing large amounts of data in one place may therefore affect the total storage capability
- Latest eMMC standard is v5.1 (2015), transfer rate 400 MB/s
- From 2016 the Universal Flash Storage (UFS) standard have taken off which is ment to replace SD cards and eMMCs
  - Unlike eMMC, Universal Flash Storage is based on the SCSI architectural model and supports SCSI Tagged Command Queuing
  - The Linux kernel supports UFS

https://en.wikipedia.org/wiki/MultiMediaCard

MMC, eMMC, SD and UFS (2)

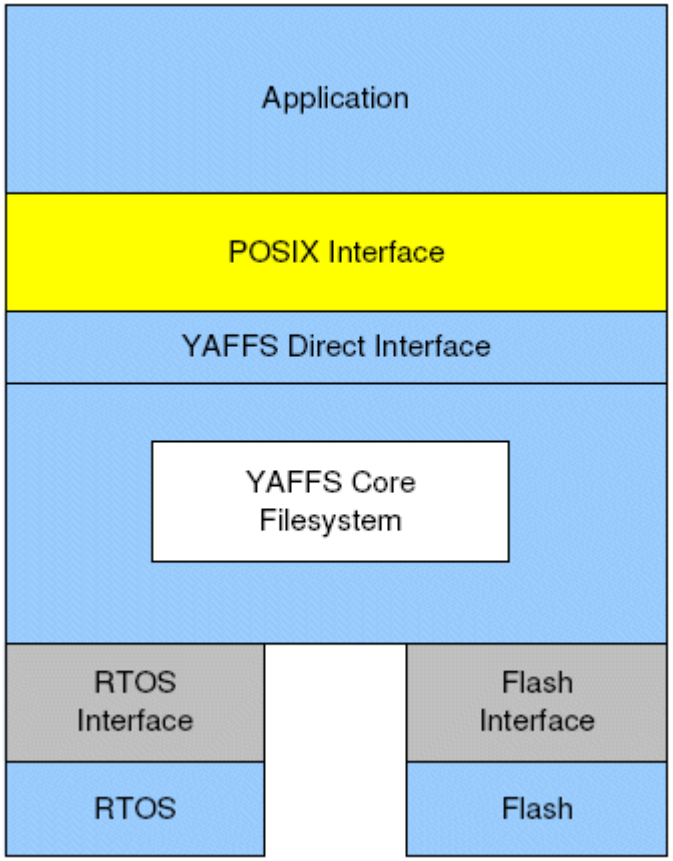
- Form factor and family
  - SD, miniSD and microSD
  - SDSC, SDHC, SDXC
  - SDXC uses exFAT
- Ultra High Speed (UHS) bus
  - Is available on some SDHC and SDXC
  - UHS-I, UHS-II and UHS-III (higher is better)
- Speed Class rating
  - Class 2 – Class 10
  - UHS (U1 – U3)
  - Video speed class (V6 - V90)
- Application Performance Class
  - A1 - A2



https://en.wikipedia.org/wiki/Secure_Digital

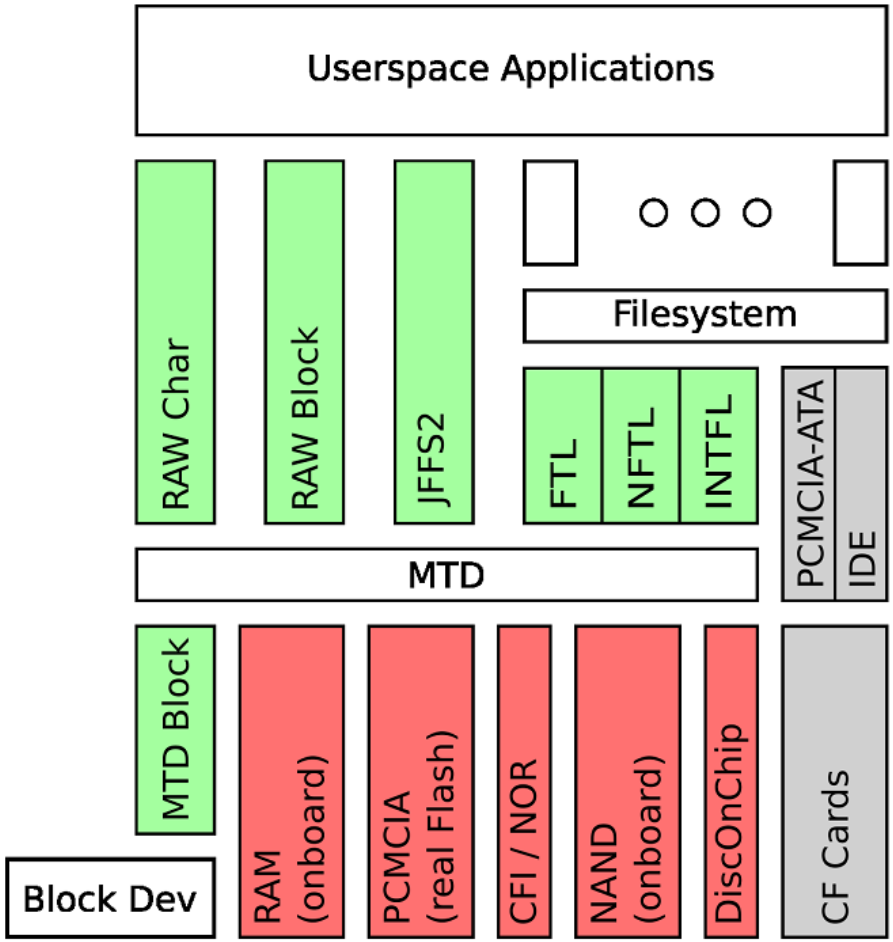
YAFFS2 (Yet Another Flash File System)

- YAFFS is a filesystem designed specifically for the characteristics of NAND flash. Its well-proven primary features are:
- Fast - typically much faster than other alternatives
- Easily ported (currently ported to GNU/Linux, WinCE, eCOS, pSOS, VxWorks, and various bare-metal systems)
- Log structured, providing wear-levelling and making it very robust
- Supports many flash geometries including 2K-Byte and 512-Byte page NAND flash chips
- Very fast mount - almost immediate startup
- Typically uses less RAM than comparable File Systems
- http://www.yaffs.net

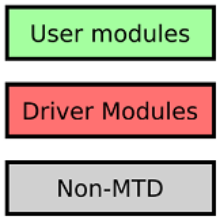


YAFFS2 (Yet Another Flash File System)

- MTD (Memory Technology Device)
- The Linux MTD System And Flash File systems paper

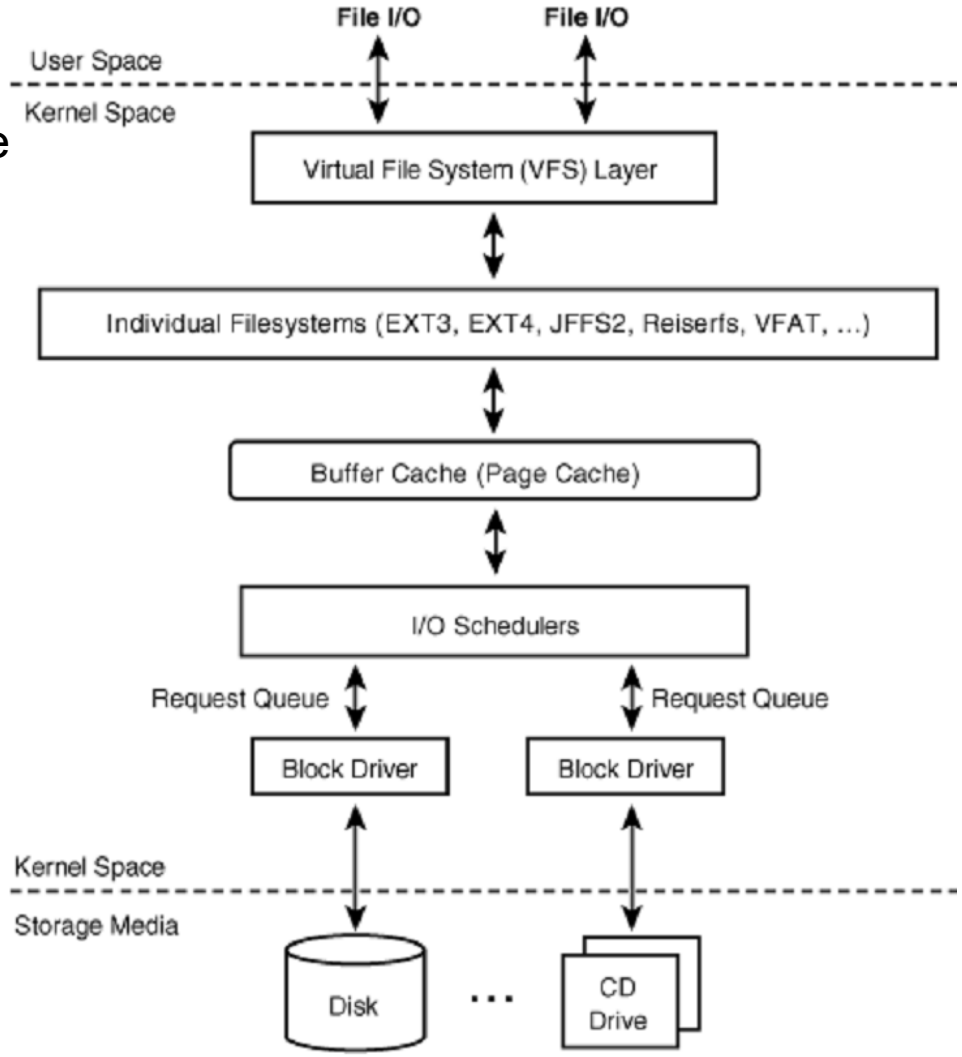


The structure of MTD in Linux



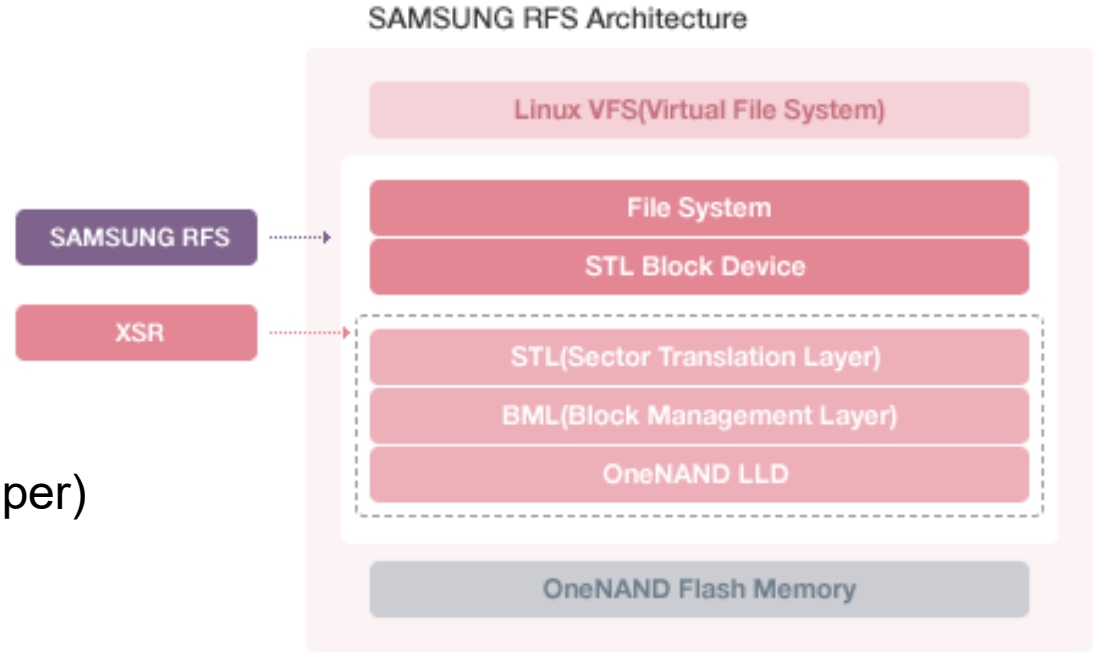
Virtual File System (VFS) Layer

- The block I/O layer was considerably overhauled between the 2.4 and 2.6 kernel releases. The motivation for the redesign was that the block layer, more than other kernel subsystems, has the potential to impact overall system performance.
- The storage media contains files residing in a filesystem, such as EXT3/4 or Reiserfs. User applications invoke I/O system calls to access these files. The resulting filesystem operations pass through the generic Virtual File System (VFS) layer before entering the individual filesystem driver.
- The buffer cache speeds up filesystem access to block devices by caching disk blocks. If a block is found in the buffer cache, the time required to access the disk to read the block is saved.



Samsung RFS and OneNAND

- Samsung RFS (Robust File System) = FAT16/32 + journal
  - Used in Android phones (at start) as well as in dumb phones
  - http://movitool.ntd.homelinux.org/trac/movitool/wiki/RFS
- SAMSUNG OneNAND takes both advantages from high-speed data read function of NOR flash and the advanced data storage function of NAND flash
  - NAND Core + NOR Interface Logic + SRAM Buffer

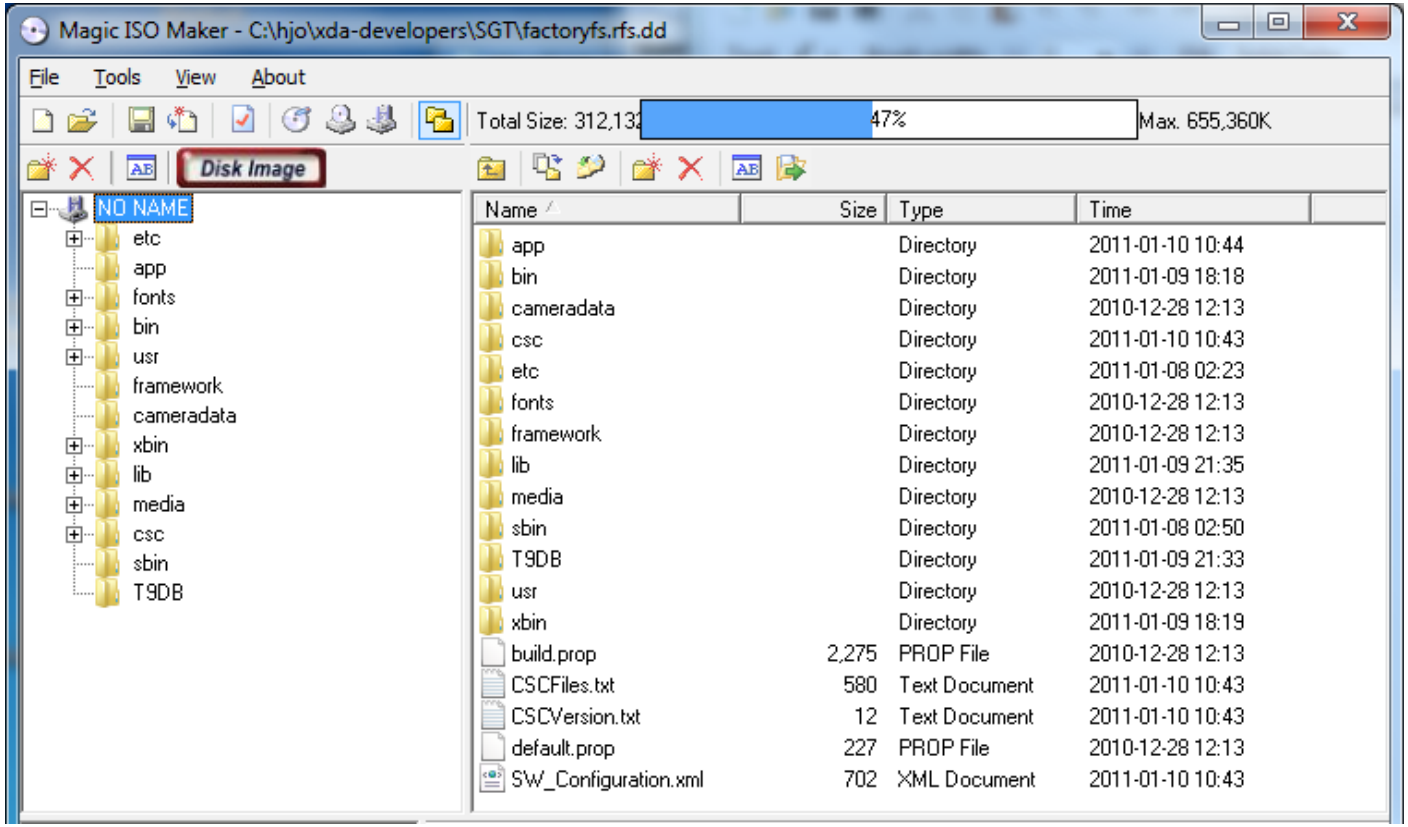


Note the STL and BML in the XSR (eXtended Sector Remapper)

Samsung RFS

- RFS dd dump of a partition from a Samsung Galaxy Tab
  - Mounted with Magic ISO
  - Dump is possible to mount under Linux as well

mount -o loop factory.rfs /some_dir



Rfs and Ext – df and mount

Other commands to inspect the block device

cat /proc/partitions
cat /proc/mtd

rfs

df
/system: 326336K total, 317068K used, 9268K available (block size 4096)
/data: 1992608K total, 412176K used, 1580432K available (block size 16384)
mount
/dev/block/stl9 /system rfs ro,relatime,vfat,log_off,check=no,gid/uid/rwx,iocharset=utf8 0 0
/dev/block/mmcblk0p2 /data rfs
rw,nosuid,nodev,relatime,vfat,llw,check=no,gid/uid/rwx,iocharset=utf8 0 0

After conversion to ext4

df
/dev/block/stl9 323028 275524 47504 85% /system
/dev/block/mmcblk0p2 1963408 335408 1628000 17% /data
mount
/dev/block/stl9 on /system type ext4 (ro,relatime,barrier=1,data=ordered)
/dev/block/mmcblk0p2 on /data type ext4
(rw,nosuid,nodev,noatime,barrier=1,data=ordered,noauto_da_alloc)