CS301
Embedded System and Microcomputer Principle

Lecture 12: Embedded Storage Management

2023 Fall

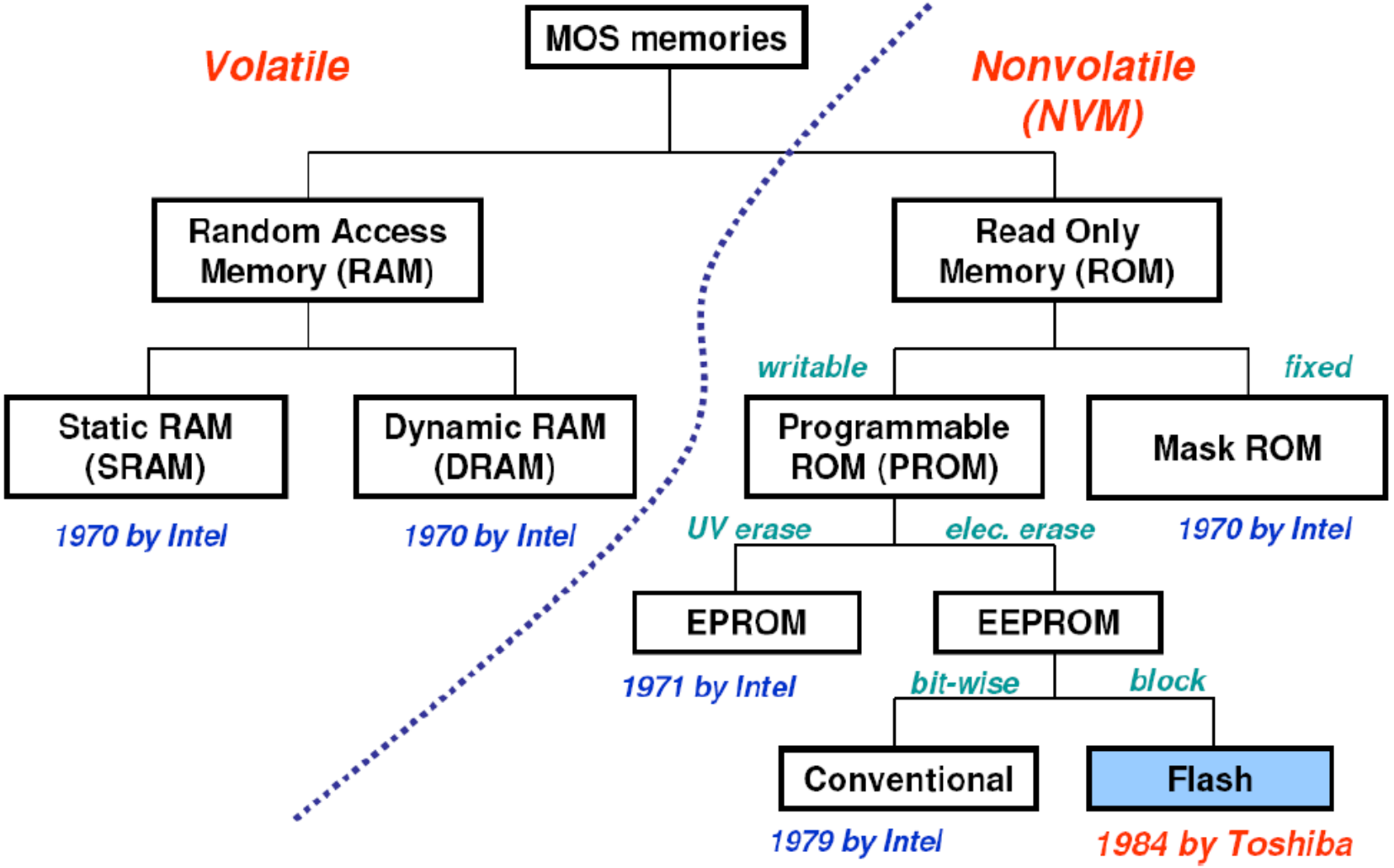
Outline

- **Massive Storage**
- SD Card
- File System

Volatile & Non-Volatile Memory

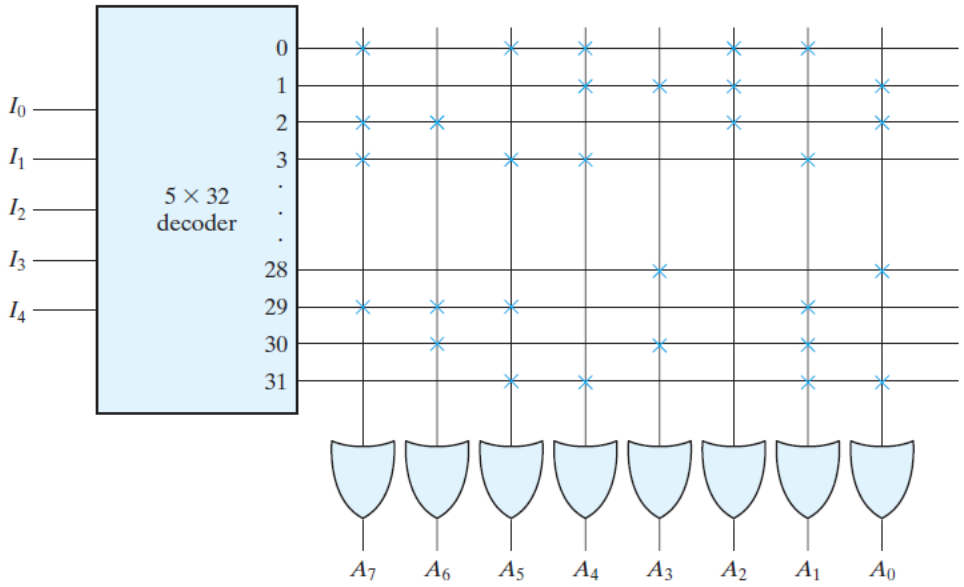
- Volatile memory: temporary storage that loses its content when power is turned off
  - SRAM (Static Random Access Memory)
    - Low density, high power, expensive, fast
    - Content will last until lose power
    - Often used for caches
  - DRAM (Dynamic Random Access Memory)
    - High density, low power, cheap, slow
    - Need to be refreshed regularly
    - Used for main memory
- Non-volatile memory: retains stored information even when power is disconnected
  - ROM (Read-Only Memory):
    - Contains permanent, pre-programmed data.
    - Retains information across power cycles.
  - EPROM/Flash (Erasable Programmable ROM)
    - Rewritable, non-volatile storage used for firmware, configuration, and data storage.

Memory


MOS memories
Volatile
Nonvolatile (NVM)
Random Access Memory (RAM)
Read Only Memory (ROM)
Static RAM (SRAM)
Dynamic RAM (DRAM)
writable
fixed
Programmable ROM (PROM)
Mask ROM
1970 by Intel
1970 by Intel
UV erase
elec. erase
1970 by Intel
EPROM
EEPROM
1971 by Intel
bit-wise
block
Conventional
Flash
1979 by Intel
1984 by Toshiba

Mask ROM

- The “simplest” memory technology
- Presence/absence of diode at each cell denote value
- Pattern of diodes(fuse) defined by mask used in fab process
- Contents are fixed when chip is made; cannot be changed
- Good for applications where
  - Upgrading contents not an issue
  - e.g. boot ROM
- Exercise:
  - What are the contents:
    - When I = 00011?
    - When I = 11111?

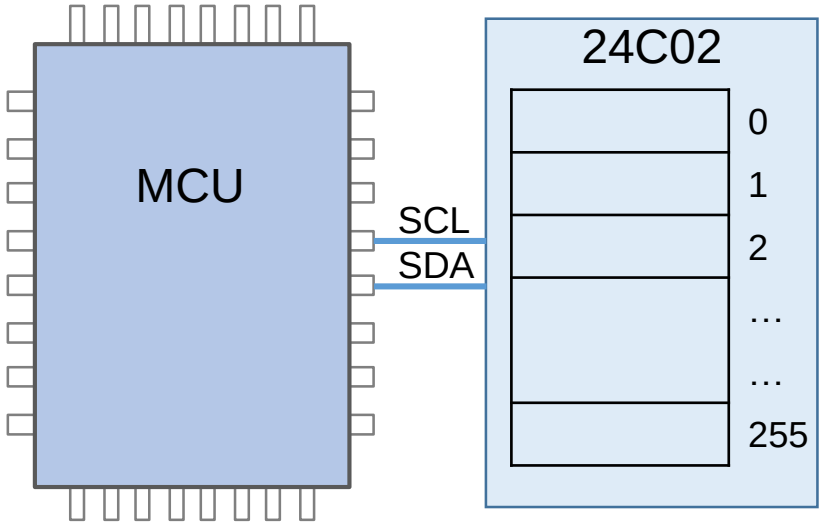


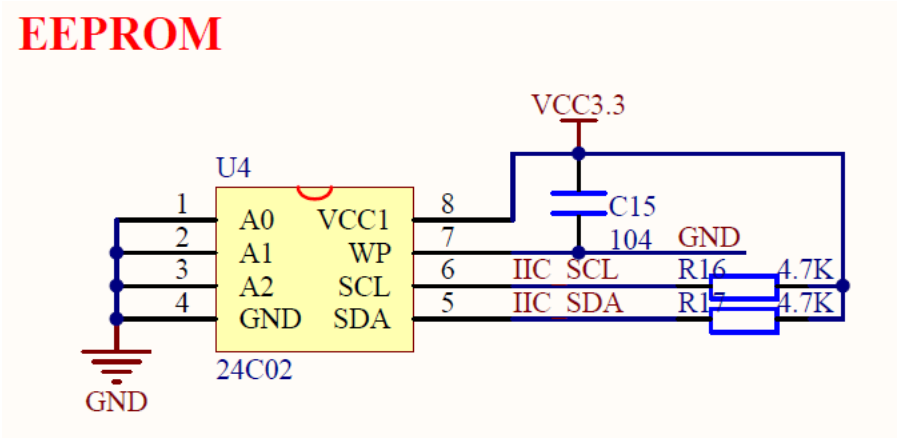
EPROM

- Erasable & Programmable ROM
  - Erasing means changing form 0 → 1
    - Uses UV light (not electrically!)
  - Writing means changing from 1 → 0
- Erase unit is the whole device
- Retains data for 10-20 years
- Not used much these days
- Costly because
  - Use of quartz window (UV transparent)
  - Use of ceramic package

EEPROM

- Electrically Erasable & Programmable ROM
- Typical EEPROM: 24C02
  - 256 bytes
  - Organized and accessed in bytes
  - Random access
  - No need of erase before rewriting
- STM32 AT24C02 EEPROM (256 x 8 (2K))



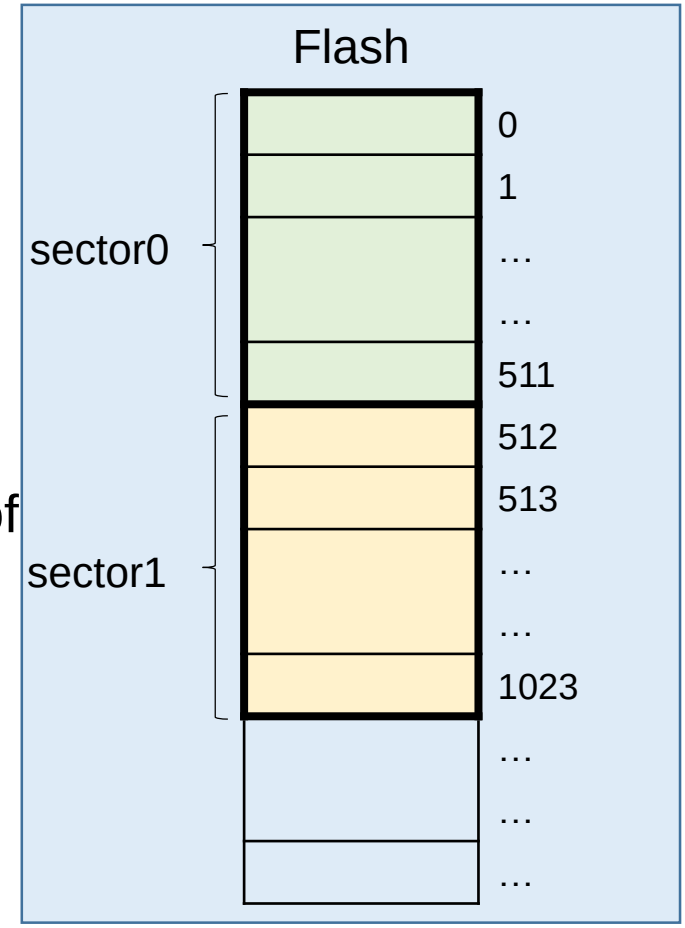


EEPROM Storage Example

Address	Data
0	Volume
1	Brightness
2	Channel
…	…
255	…

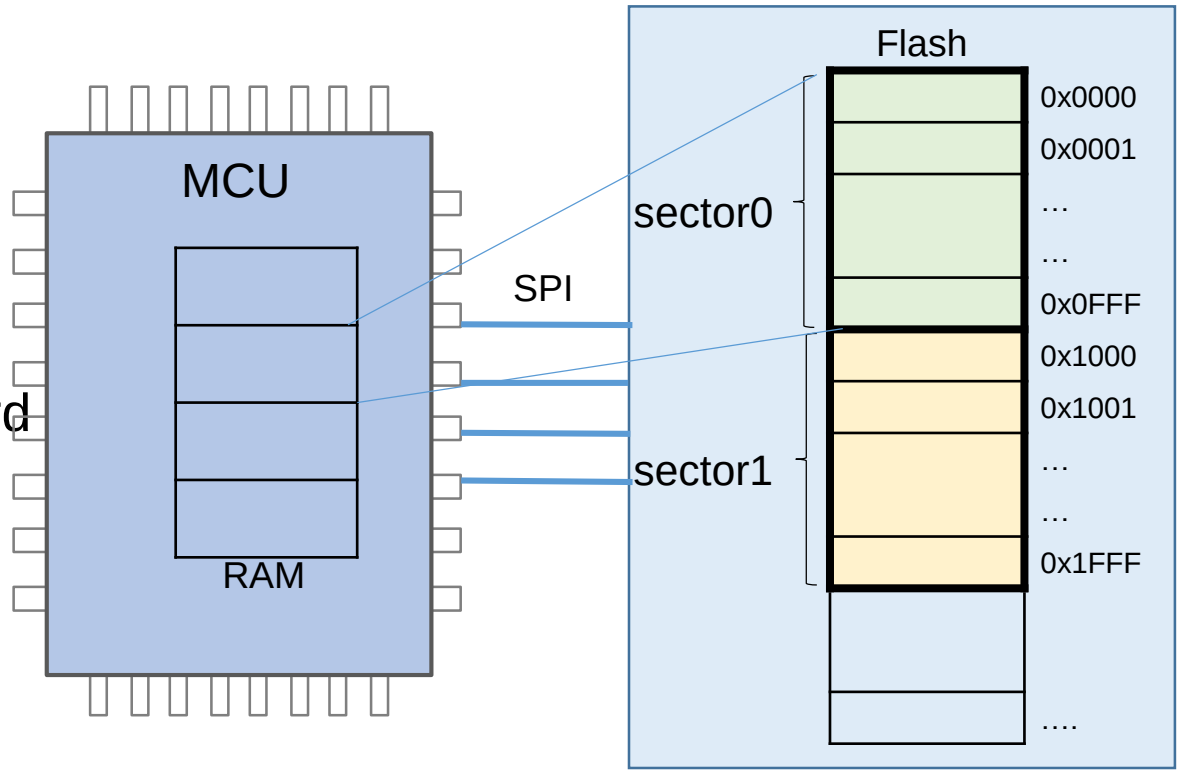
Flash Memory

- Electrically erasable (like EEPROM, unlike EPROM)
- Used in many reprogrammable systems these days
- Sectors(扇区):
  - refers to a fixed-size, contiguous block of storage
  - typically contains a specific number of bytes, e.g. 512
  - represents the smallest addressable unit for reading and writing data.
  - Smallest erasable unit



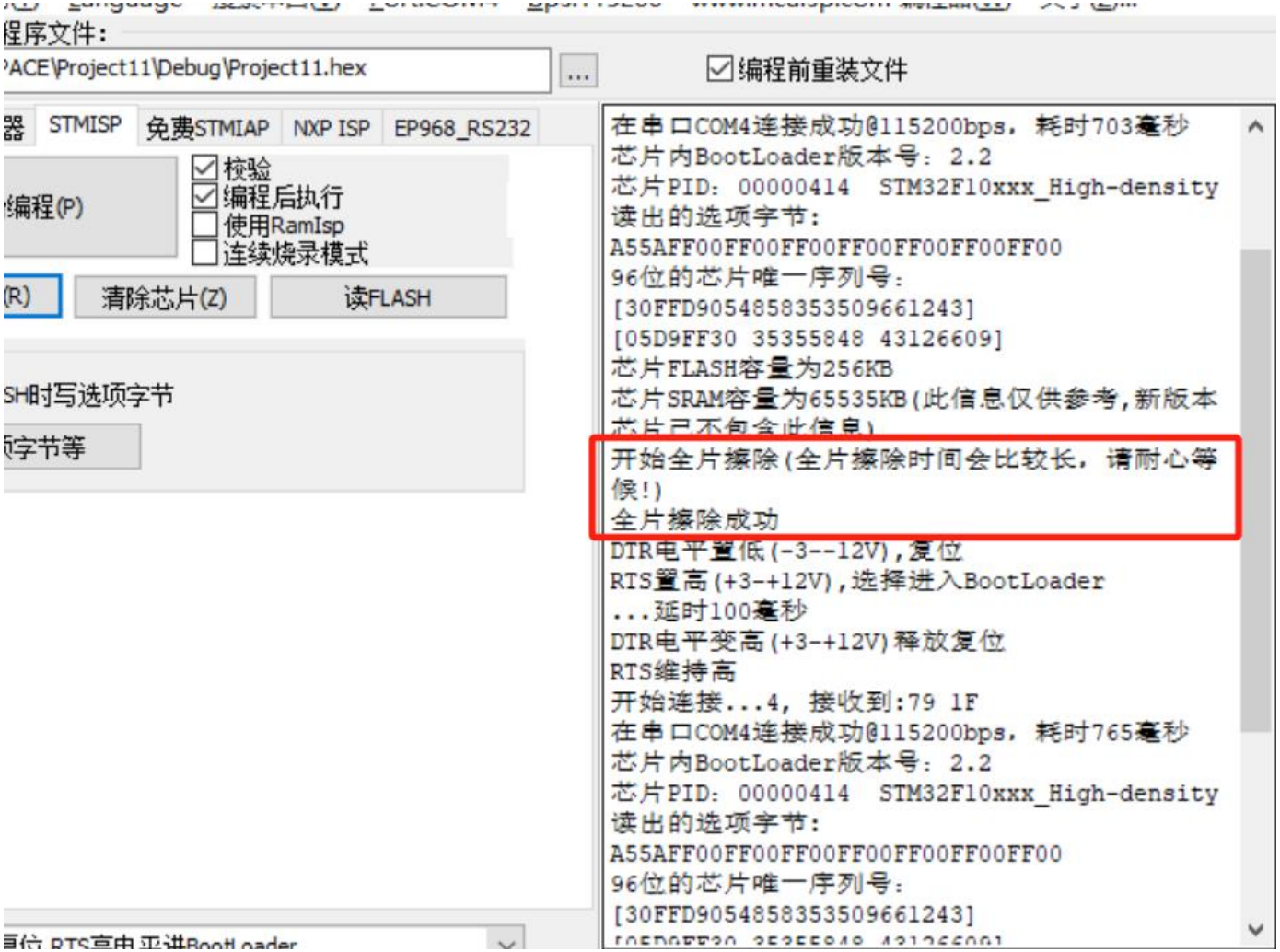
Flash Memory

- Read: are like standard RAM
- Write: must explicitly erase entire sector before writing
  - Erase sets entire sector contents to ‘1’
  - RAM can help transferring data without interference of neighbor contents
    - Fetch
    - Modify portion
    - Write
- Applications
  - Flash
  - Secure Digital Card
  - USB Flash Drive
  - Solid State Drive
  - etc



STM32 Flash Programming

- Erase before writing



Outline

- Massive Storage
- **SD Card**
- File System

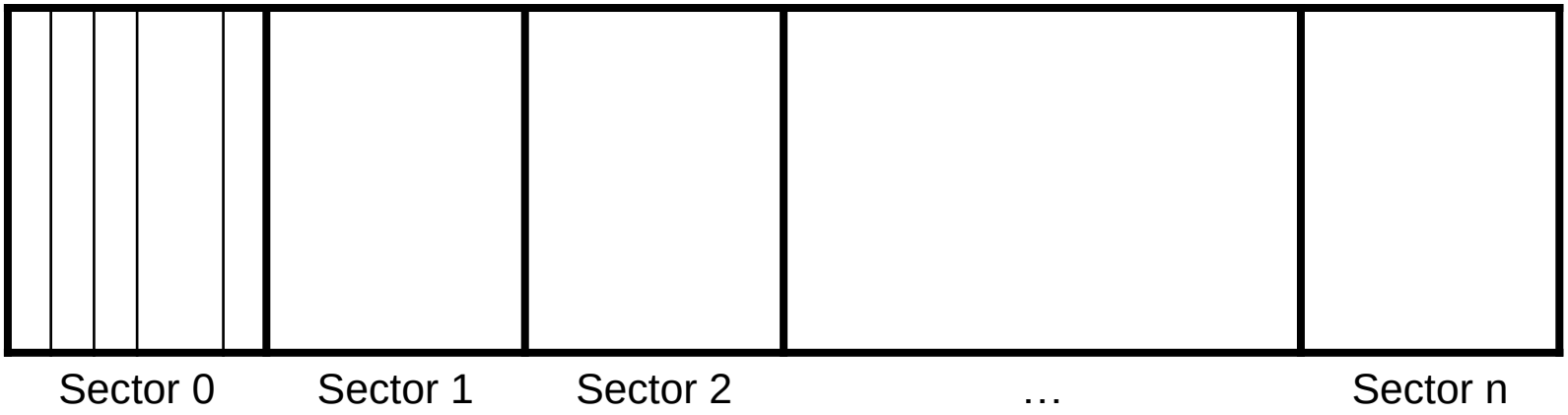
SD Card

- SD Card: Secure Digital Card, Small, portable, non-volatile memory card for data storage.
- Underlying Technology:
  - SD cards primarily use Flash memory for data storage
  - Data is written and erased at the sector level.

SD Card Operation

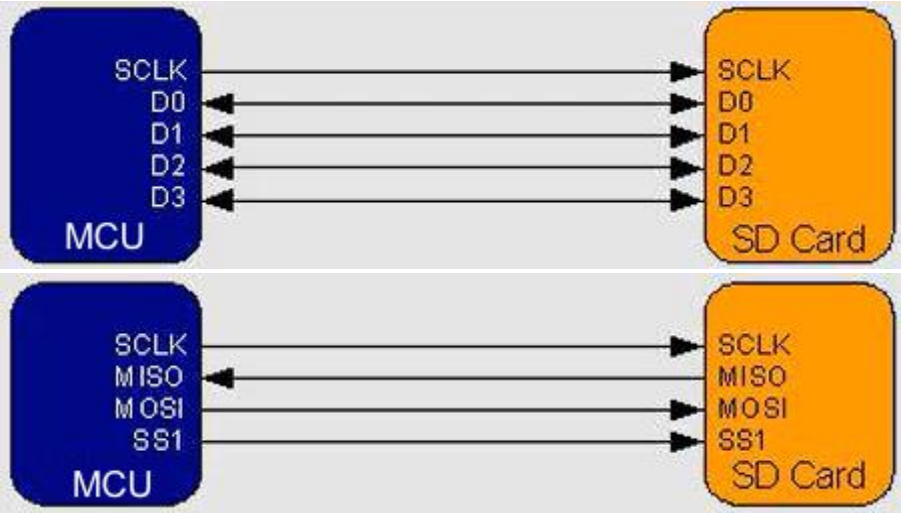
- Read a sector
- Write a sector
- Get SD Card status
- Initialize SD Card

SD Card Driver



SD Card Interface

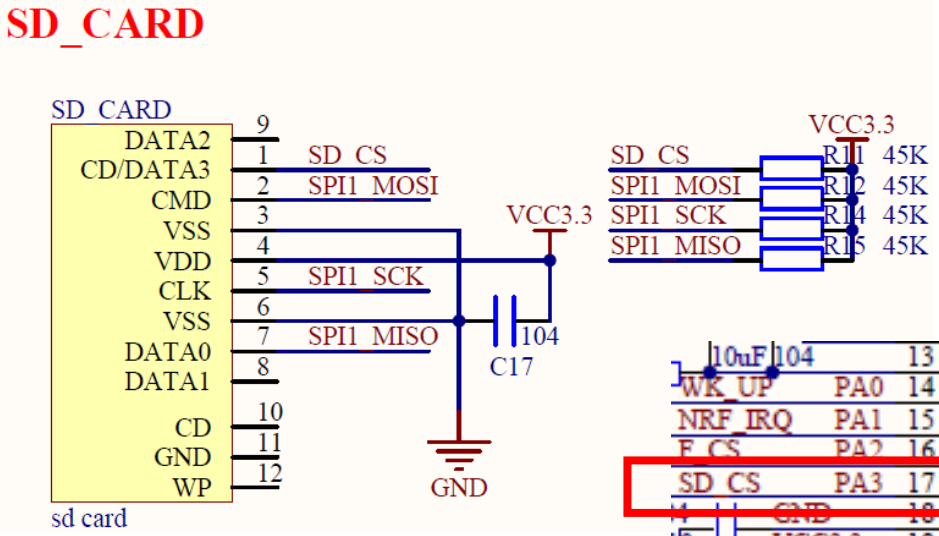
- SD Mode
  - Clock
  - 4 Data Lines
- SPI Mode
  - Clock
  - Card Select
  - 2 Data Lines

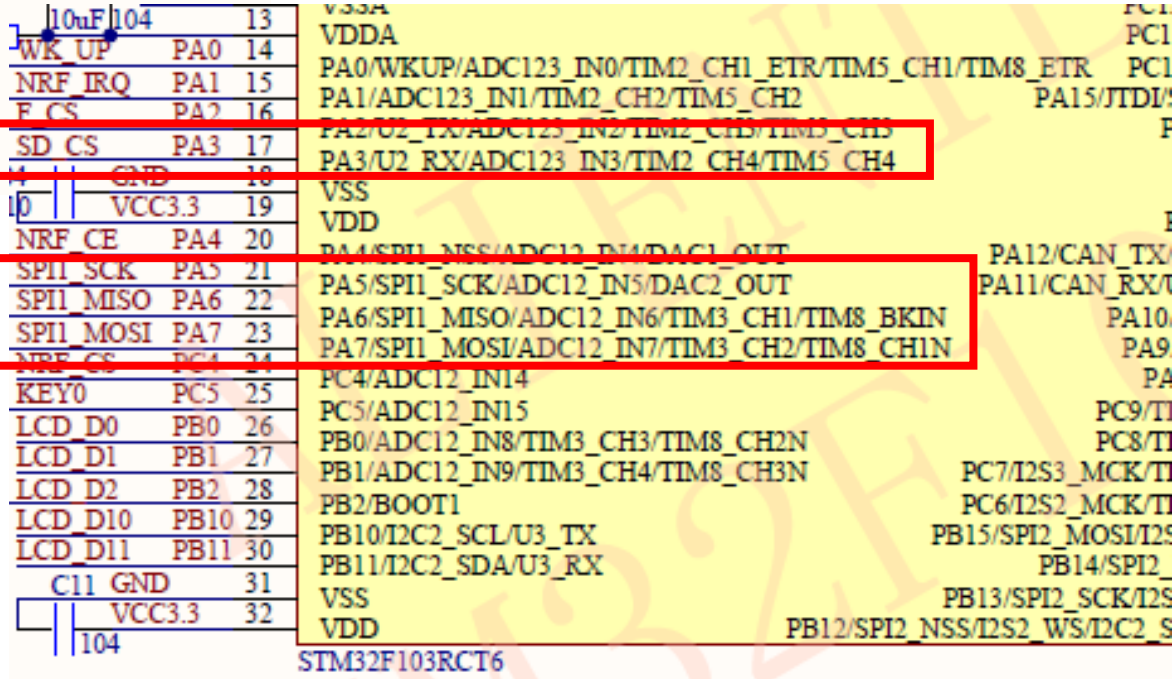


Pin	SD	SPI
1	DAT3	CS
2	CMD	DI(MOSI)
3	Vss	Vss
4	Vcc	Vcc
5	CLK	CLK
6	Vss2	Vss2
7	DAT0	DO(MOSO)
8	DAT1	Reserved
9	DAT2	Reserved

STM32 SD Card

- Alientek Mini board only supports SPI Mode, CS pin is PA3





SD Commands

C.1 SD Mode Command List

Table C- 1 and Table C- 2 show the commands that are supported by SD memory and SDIO devices in both SPI and SD modes. If a command is not identified as either mandatory or optional, then it is not supported by that device.

Supported Commands	Abbreviation	SDMEM System	SDIO System	Comments
CMD0	GO_IDLE_STATE	Mandatory	Mandatory	Used to change from SD to SPI mode
CMD2	ALL_SEND_CID	Mandatory		CID not supported by SDIO
CMD3	SEND_RELATIVE_ADDR	Mandatory	Mandatory	
CMD4	SET_DSR	Optional		DSR not supported by SDIO
CMD5	IO_SEND_OP_COND		Mandatory	
CMD6	SWITCH_FUNC	Mandatory[1]		Added in Part 1 v1.10
CMD7	SELECT/DESELECT_CARD	Mandatory	Mandatory	
CMD8	SEND_IF_COND	Optional	Optional	SDHC or SDXC
CMD9	SEND_			
CMD10	SEND_			
CMD11	VOLTA(			
CMD12	STOP_			
CMD13	SEND_			
CMD15	GO_IN/			
CMD16	SET_BL			

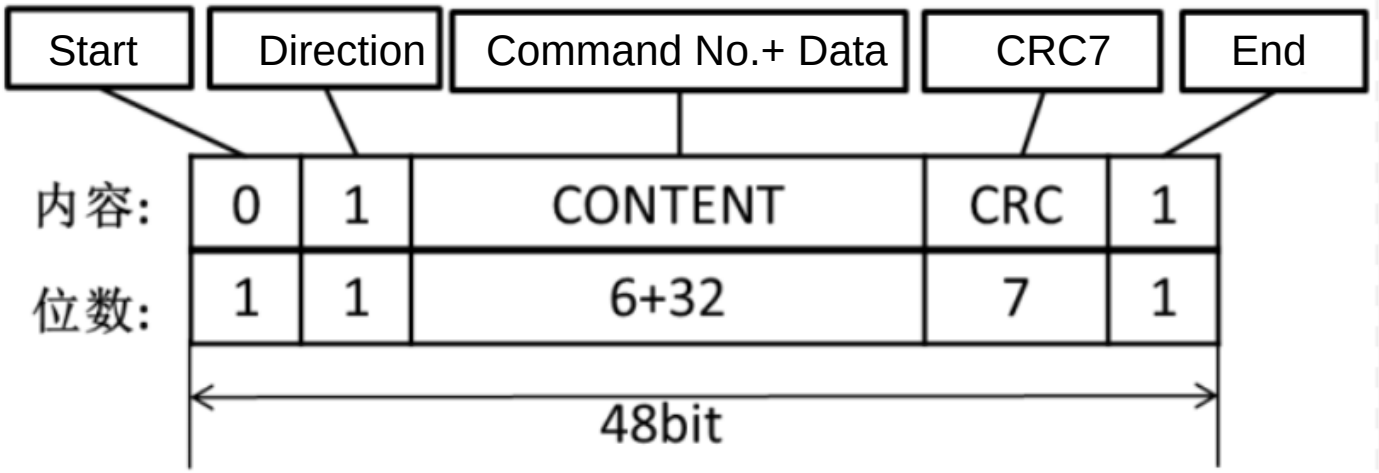
C.2 SPI Mode Command List

Supported Commands	Abbreviation	SDMEM System	SDIO System	Comments
CMD0	GO_IDLE_STATE	Mandatory	Mandatory	Used to change from SD to SPI mode
CMD1	SEND_OP_COND	Mandatory		
CMD5	IO_SEND_OP_COND		Mandatory	
CMD6	SWITCH_FUNC	Mandatory[1]		Added in Part 1 v1.10
CMD9	SEND_CSD	Mandatory		CSD not supported by SDIO
CMD10	SEND_CID	Mandatory		CID not supported by SDIO
CMD12	STOP_TRANSMISSION	Mandatory		
CMD13	SEND_STATUS	Mandatory		Card Status includes only SDMEM information.
CMD16	SET_BLOCKLEN	Mandatory		
CMD17	READ_SINGLE_BLOCK	Mandatory		.
CMD18	READ_MULTIPLE_BLOCK	Mandatory		
CMD24	WRITE_BLOCK	Mandatory		
CMD25	WRITE_MULTIPLE_BLOCK	Mandatory		

SD Specifications

SD Commands

- Each Command has 48-bits



Outline

- Massive Storage
- SD Card
- **File System**

File System

- File System: a program used to manage and organize file data on a disk, facilitating operations such as searching, modifying, editing, etc.
- Typical File Systems
  - FAT
    - FAT12, FAT16, FAT32, ExFAT
  - NTFS
  - ext2, ext3, ext4
  - CDFS
    - CD-R, CD-RW

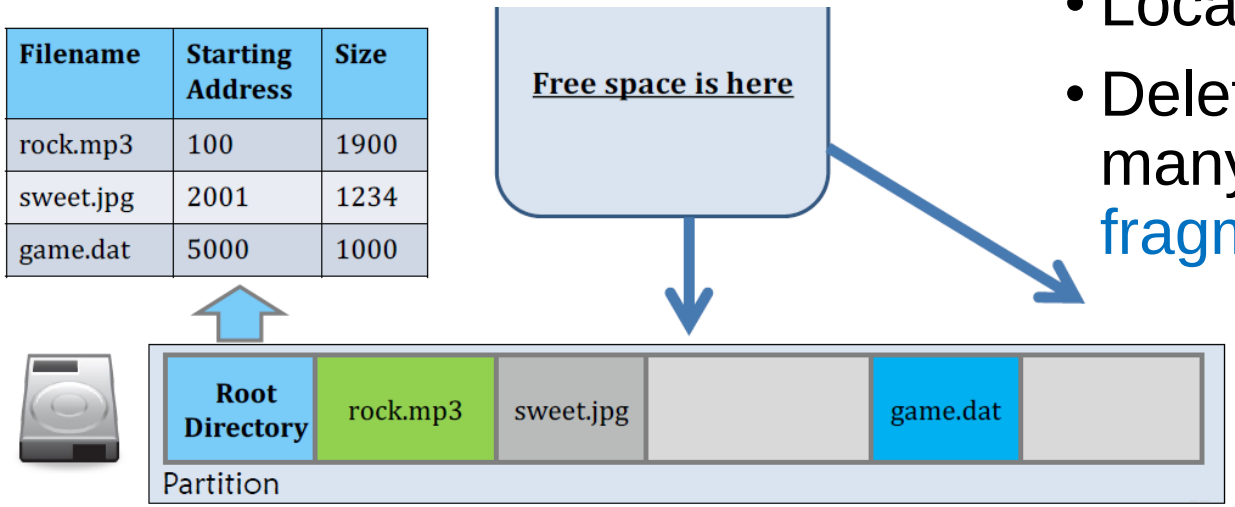
SD card file systems

	Storage	File System
SD	128MB~2GB	FAT16
SDHC	4GB~32GB	FAT32
SDXC	64GB~2TB	exFAT

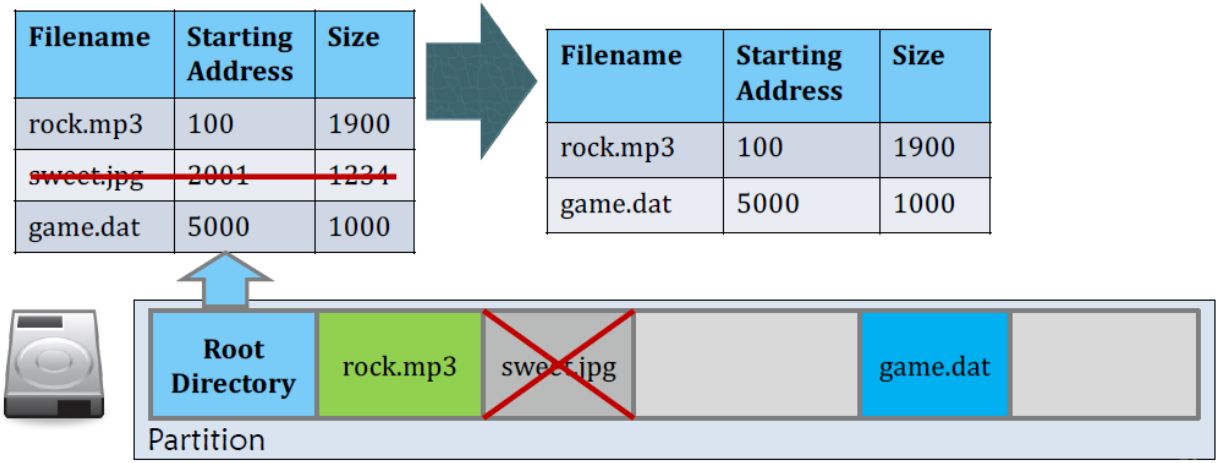
File System Structure

- I/O transfers between memory and disk are performed in units of blocks
  - one block is one or more sectors
  - one sector is usually 512 bytes
- Two design problems in FS
  - interface to user programs
  - interface to physical storage (disk)

FAT Motivation

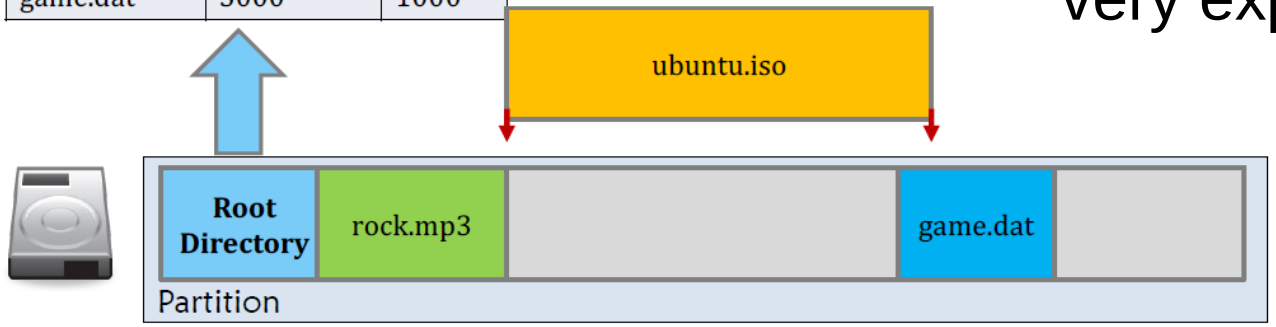


- Locate files is easy
- Deleting file will cause many external fragmentation



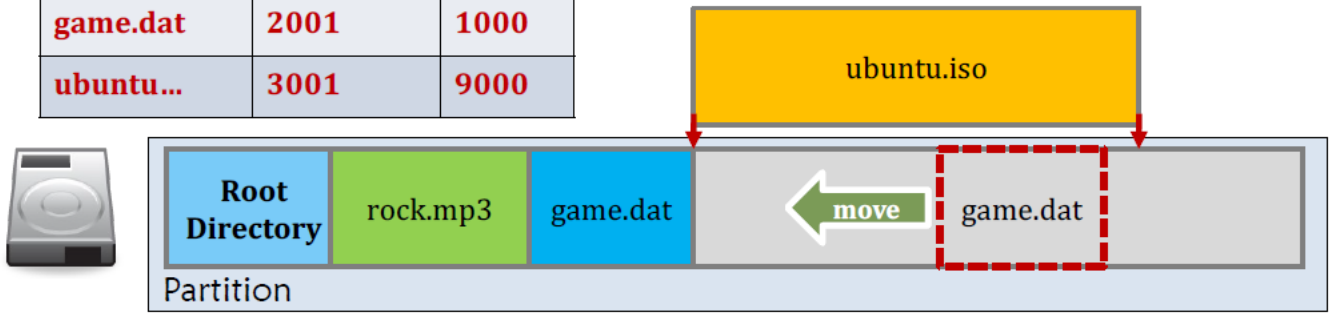
FAT Motivation

Filename	Starting Address	Size
rock.mp3	100	1900
game.dat	5000	1000



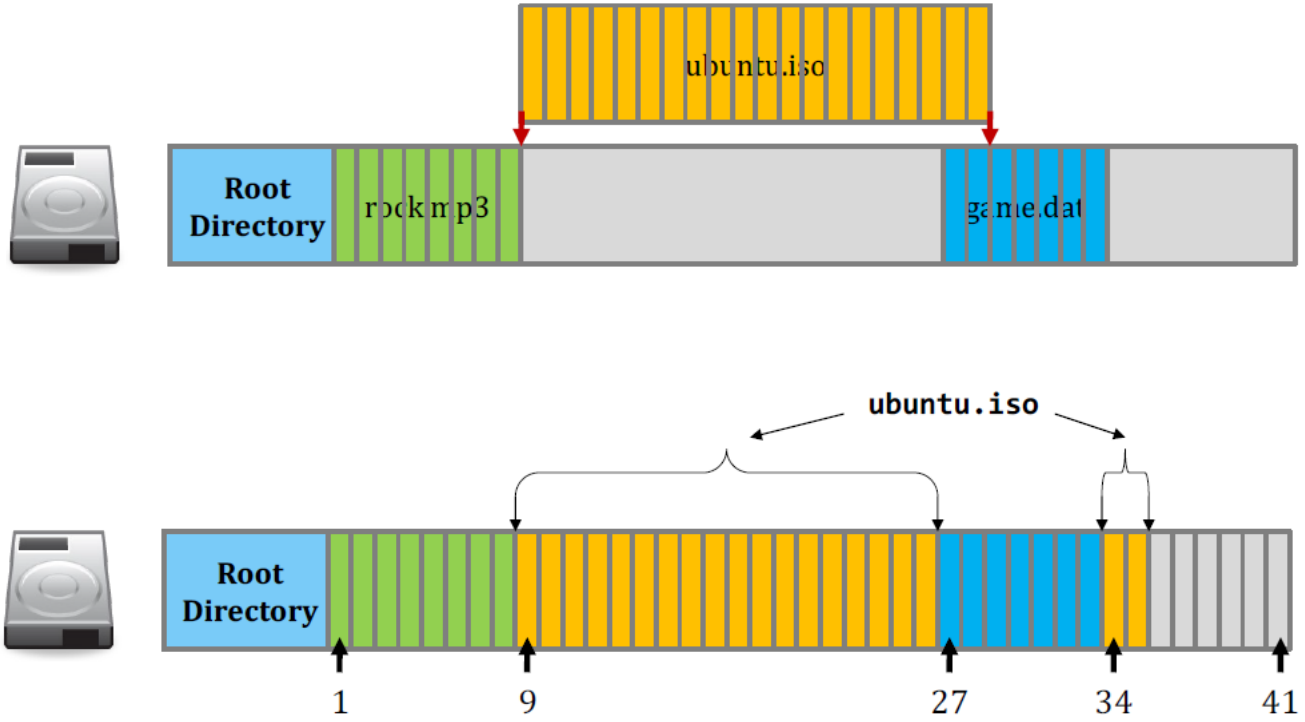
- Defragmentation process may help, but very expensive

Filename	Starting Address	Size
rock.mp3	100	1900
game.dat	2001	1000
ubuntu...	3001	9000



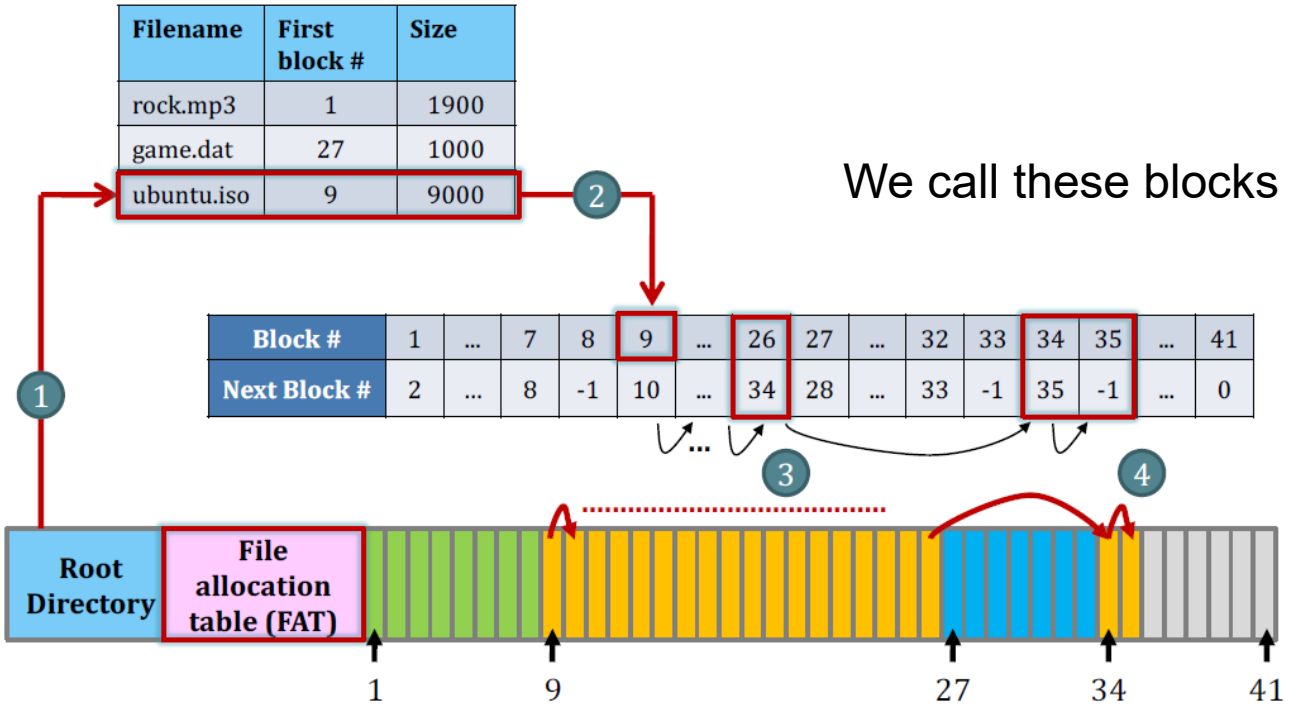
FAT Motivation

- Chop the storage device and data into equal sized blocks.
- Fill the empty space in a block-by-block manner



File Allocation Table (FAT)

- Task: read “ubuntu.iso” sequentially
  - Step 1. Read the root directory and retrieve the first block number.
  - Step 2. Read the FAT to determine the location of next block.
  - Step 3. After reading the 2nd block, the process continues. Note that the blocks may not be contiguously allocated.
  - Step 4. The process stops until the FAT says the next block # is -1.



We call these blocks as “Clusters”

FAT

- Cluster(簇)
  - 1 cluster = 2^n sectors
  - E.g. FAT12: 12-bit cluster address
  - Can point up to 2^{12} = 4096 clusters

Example:
Cluster size: 32KB
Cluster address width = 28bits
File system size = $(32 * 2^{10}) * 2^{28}$
$= 2^5 * 2^{10} * 2^{28} = 2^{43}$ ($8TB = 2^{(40+3)}$)

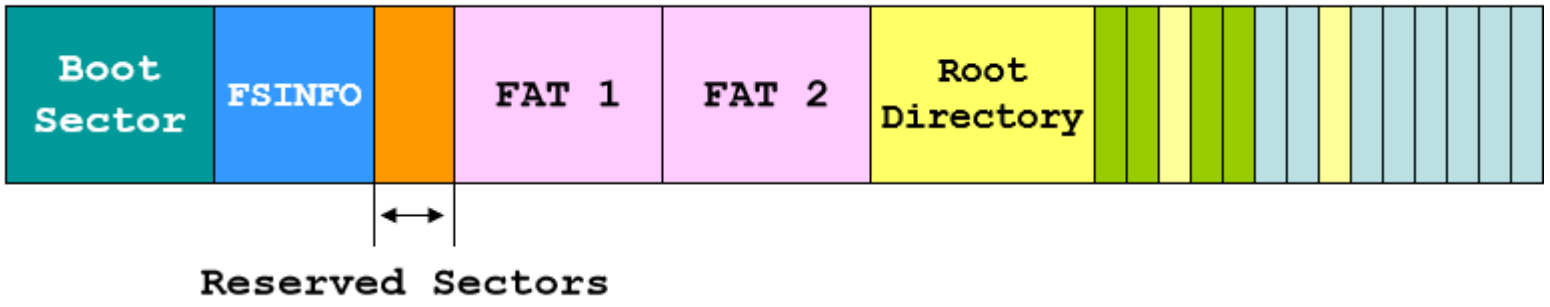
FS Size = number of Clusters × Cluster size
$= 2^{\text{cluster address width}}$ × Cluster size

	FAT12	FAT16	FAT32
Cluster address width	12 bits	16 bits	28 bits 4 bits reserved
Number of Clusters	2^{12} (4K)	2^{16} (64K)	2^{28} (256M)

FAT Layout

	Propose	Size
Boot sector	FS-specific parameters	1 sector, 512 bytes
FSINFO	Free-space management	1 sector, 512 bytes
More reserved sectors	Optional	Variable, can be changed during formatting
FAT (2 pieces)	1 copy as backup	Variable, depends on disk size and cluster size.
Root directory	Start of the directory tree.	At least one cluster, depend on the number of directory entries.

Reserved sectors



File Read

Task: read "C:\windows\gamedata.dat" sequentially.

FAT1

0	...
1	...
...	...
32	33
33	EOF
34	0
35	0

Filename	Attributes	Cluster #
gamedata.dat	.....	32

Damaged	= 0x0ffffff7
EOF	>= 0x0ffffff8
Unallocated	= 0x0

Step 1. Read the content from Cluster #32.
Note. The **file size** may also help determining if the last cluster is reached.

Step 2. Look for the next cluster and it is Cluster #33.

File Read

Task: read "C:\windows\gamedata.dat" sequentially.

0	...
1	...
...	...
32	33
33	EOF
34	0
35	0

Filename	Attributes	Cluster #
gamedata.dat	.....	32

Step 3. Since the FAT has marked "EOF", we have reached the last cluster of that file.

Note. The file size help determining **how many bytes to read** from the last cluster.

File Write

Task: append data to "C:\windows\gamedata.dat".

0	...
1	...
...	...
32	33
33	EOF
34	0
35	0

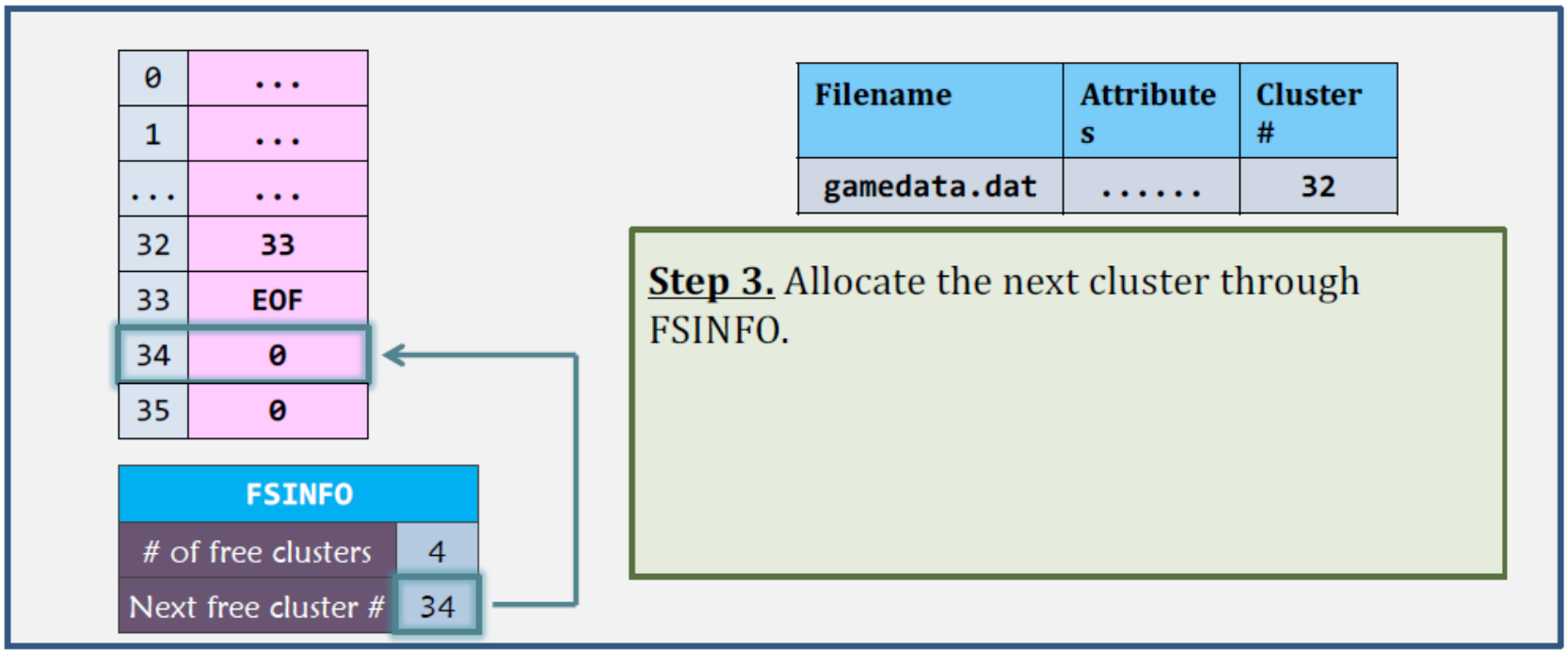
Filename	Attributes	Cluster #
gamedata.dat	.....	32

Step 1. Locate the last cluster.

Step 2. Start writing to the non-full cluster.

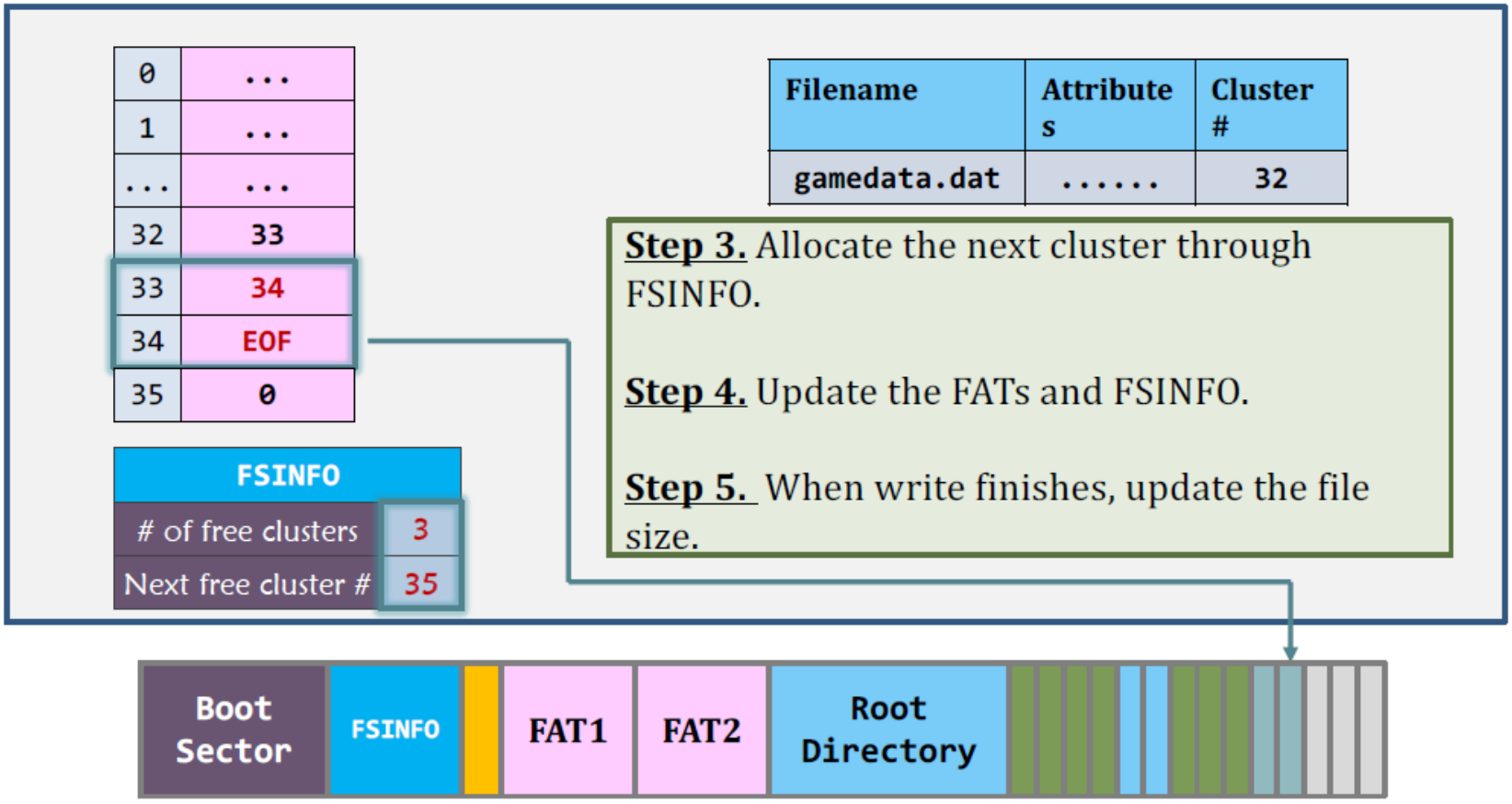
File Write

Task: append data to "C:\windows\gamedata.dat".



File Write

Task: append data to "C:\windows\gamedata.dat".



File Delete

Task: delete "C:\windows\gamedata.dat".

0	...
1	...
...	...
32	33
33	34
34	EOF
35	0

0	...
1	...
...	...
32	0
33	0
34	0
35	0

Filename	Attributes	Cluster #
gamedata.dat	.....	32

Step 1. De-allocate all the blocks involved. Update FSINFO and FATs.

FSINFO	
# of free clusters	3
Next free cluster #	35

FSINFO	
# of free clusters	6
Next free cluster #	32

File Delete

Task: delete "C:\windows\gamedata.dat".

Directory "windows"		
Filename	Attributes	Cluster #
.	.....	?
..	.....	?
_amedata.dat	.....	32
notepad.exe	.....	456

Step 2. Change the first byte of the directory entry to _ (0xE5)

That's the end of deletion!

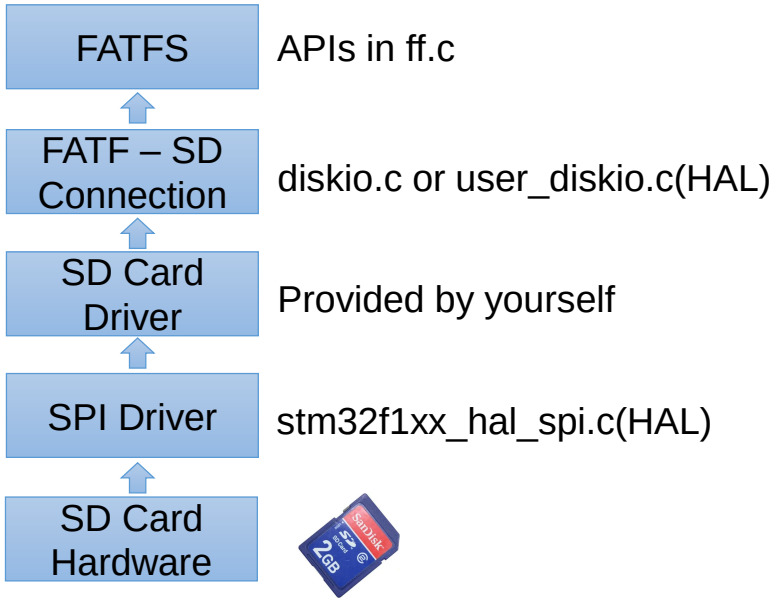
File Recovery

- “Deleted data” persists until the de-allocated clusters are reused.
- If you really care about the deleted file, then…
  - PULL THE POWER PLUG AT ONCE!
    - Pulling the power plug stops the target clusters from being over-written.

File size is within one block (cluster)	Because the first cluster address in the direct is still readable, the recovery is having a very high successful rate.
File size spans more than 1 Block	Because of the next-available search, clusters of a file are likely to be contiguous allocated. This provides a hint in looking for deleted blocks.

File System Implementation

- FATFS
  - Generic FAT/exFAT filesystem module for small embedded systems
  - Platform Independent. Easy to port.
- API
  - f_mount - Register/Unregister the work area of the volume
  - f_open - Open/Create a file
  - f_close - Close an open file
  - f_read - Read data from the file
  - f_write - Write data to the file
  - …
  - …



File System Implementation

- The storage device control module is storage dependent (e.g. SD Card Driver), it needs to be **provided by implementer**
  - disk_status - Get SD card status
  - disk_initialize - Initialize SD card
  - disk_read - Read a sector
  - disk_write - Write a sector

SD Card Driver