Mass storage and file system

Date @November 18, 2024

A file system is a way to organize, store, retrieve and manage data on storage devices. A file system provides:

- file naming and metadata: handling file names and storing associated metadata like size, type and creation date
- storage management: allocating and tracking space on the storage medium
- data organization: arranging files based on a logical structure
- access control: managing permissions for reading, writing and executing files.

It can be very simple or super complex depending on the requirements of the OS.

History

File systems have been dominated by a single technology named magnetic disk hard drives, whose key features are:

- block access: the sector is the smallest data unit
- seek time (a few ms): is the time required for the disk drive's read/write head to move to the track where the desired data is locate
- rotational latency (a few ms): is the time it takes for the desired sector of the disk to rotate under the read/write head
- transfer time (hundreds of us): is the time taken to transfer the data from the disk to the computer once the read/write head is positioned correctly
- controller overhead (hundreds of us): is the time taken by the disk controller to process the data request

If you want to move from a sector to another you have to move mechanical parts, leading to a slow technologies. In order to minimize this latency it should move as little as possibile.

Disk layout

The boot block indicates which operating systems is installed on the disk and it is read by the bootloader to find the OS image to boot. It is put at the beginning because it is a kind of agreement so you know that at the beginning of the disk you find the raw image of your bootloader and you just need to read it sector by sector.

The super block defines a file system, specifying its size, the size of the file descriptor area, the start of the list of free blocks, the location of the FCB of the root directory and other meta-data such as permissions and times.

The FCB contains file metadata that tells the name of the file, permissions, dates, owner, file size and location of file contents.

The remaining part is actually where you store data.

Block allocation

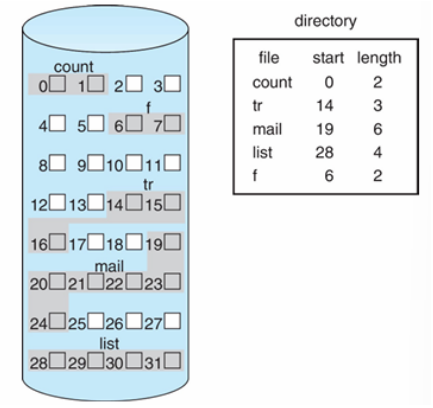
The block allocation defines how blocks are allocated to files:

- contiguous allocation
- linked allocation
- indexed allocation

Contiguous block allocation

This is the simplest allocation method. Each file is stored in contiguous blocks on the disk and the FCB contains the start block and the length for each file.

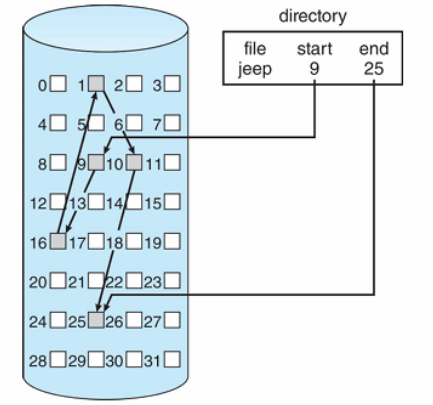
In magnetic disk is very efficient to minimize the movement of the head. It provides an efficient read/seek. The problem is that when creating a file, we don't know how many blocks may be required, and this can lead to disk fragmentation.



Linked block alllocation

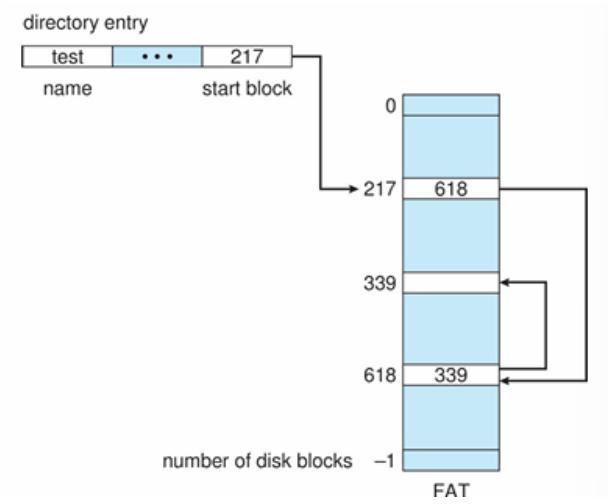
This approach is quite used. Here files can be stored on non-contiguous blocks on disk and the FCB contains the start (and optionally the end) block for each file and each block of a file points to the next block in the file.

In this way there's less fragmentation and a flexible file allocation. But sequential read requires disk seek to jump to the next block and random read will be very inefficient and there would be an O(n) seek operation.



File allocation table (FAT)

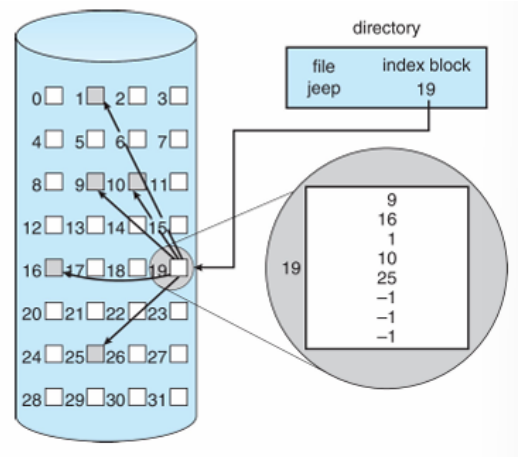
A FAT is like a linked list but faster on disk and cacheable and is a table associated to the beginning of each volume, indexed by block number. It is easy to implement, new block allocation is simple and it is very sensitive to power loss, for instance if power is lost when updating the FAT entry or the linked list of blocks.



Indexed block allocation

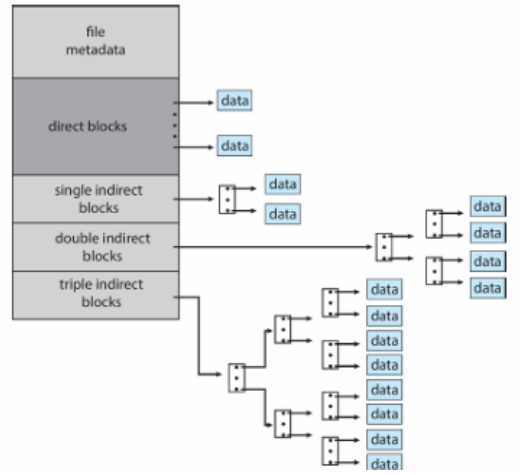
In this approach each file can be stored on non-contiguous blocks on the disk. The FCB maintains an array of pointers to data blocks. The order of the data blocks is fundamental and the cells that do not point to any block are filled with -1.

In this way random access becomes as easy as sequential access.



UNIX file system (UFS)

The UFS is a family of file systems supported by many unix and unix-like operating systems. It uses a hierarchical index allocation mechanism based on inodes. The inode (index node) is a data structure that describes a file system object such as a file or a directory. Each inode stores the attributes and disk block locations of the object's data.



Free space management

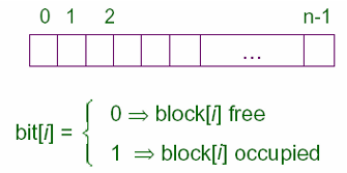
When a file is deleted we need to keep track of free blocks. File systems use two approaches:

- bit vector (or BitMap)
- linked list

Bit vector

The bit vector is an array of bits with a number of entries equal to the number of blocks is kept in memory.

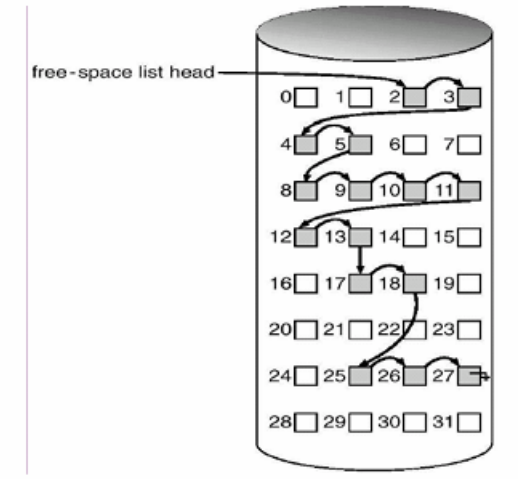
This could be very efficient with hardware support but the bitMap size grows as disk size grows. It can be inefficient if the entire bitMap can't be loaded into memory.



Linked list

In this approach there is a list of tree block whose head points to the first free block and each tree block points to the next in the list.

In this case there is no need to keep global table but we have to access each block in the disk one by one to find more than one free block.



Flash memory

Flash memory is a non-volatile memory, which means that it preserves state without any power, it has a solid state, meaning that there are no moving parts larger than electrons and it is fast, compared to disk.

Types

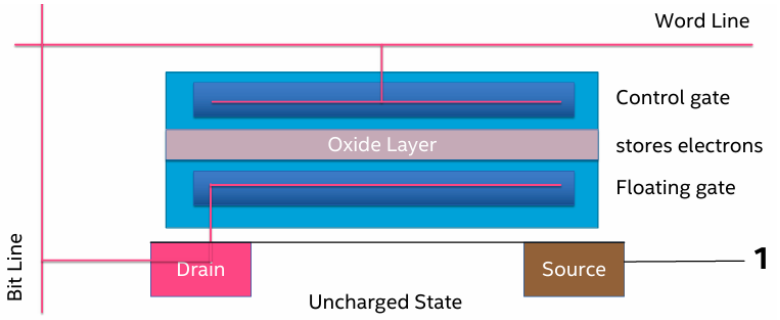
- NOR flash: Used to store firmware images on embedded devices. NOR flash chips connect to the processor through address and data lines like normal RAM, allowing code on NOR flash to be executed in place
- NAND flash: Used for large-scale, dense, and cost-effective storage in solid-state mass storage devices like USB drives and Disk-on-Modules (DOMs). NAND flash chips interface through I/O and control lines, and code stored on NAND flash must be copied to RAM before execution

NAND flash memory

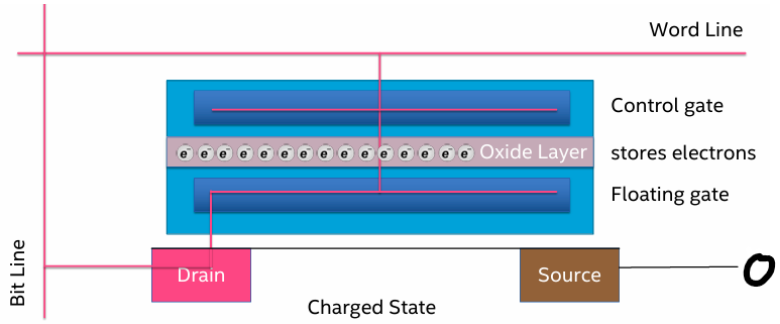
The idea of all flash memory is that we can store information in a transistor, which is a tri-pole, meaning you have three inputs and you can control, using inputs, if there is a

current flows from the source to the drain.

In a flash NAND memory we have a transistor that basically has a oxide layer that completely disconnects the control gate and the floating gate. Usually there is no current that flows and this is associated to a bit equal to 1.



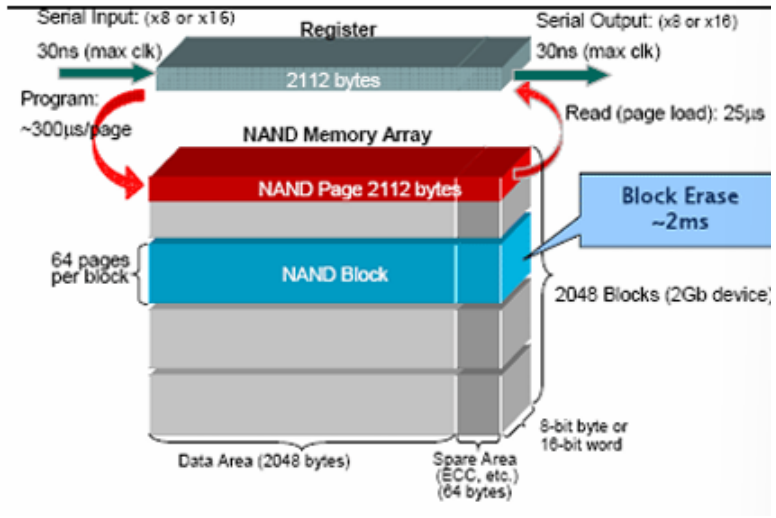
This technology has the possibility of injecting electrons in the oxide layer that are not able to escape, so you can program this by injecting electrons which are trapped creating a connection between source and drain in a permanent way. In normal transistors when you remove current from the world line the information disappears, while in floating gates the information remains.



Challenges of flash memory

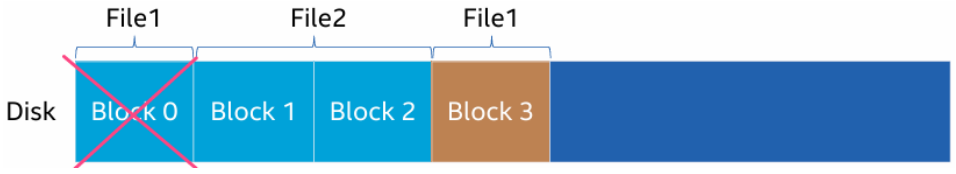
Flash memories are initialized with 1s. Writing 1→0 is expensive and writing (erasing) 0→1 is super expensive because we need to apply electric field to release charge and we can only erase a full block at a time. Furthermore, reading disturbs nearby cells, so we cannot read same cell too many times.

Flash memory is organized in pages and blocks inside a page. There is also a small space called spare area in which the error correction code is stored.

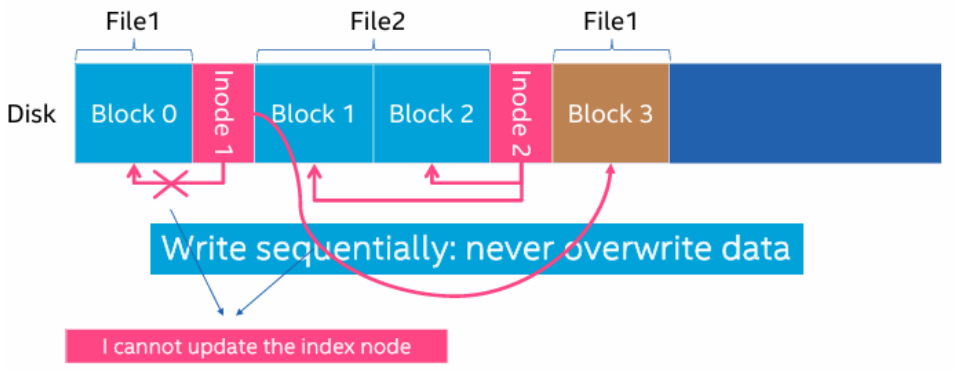


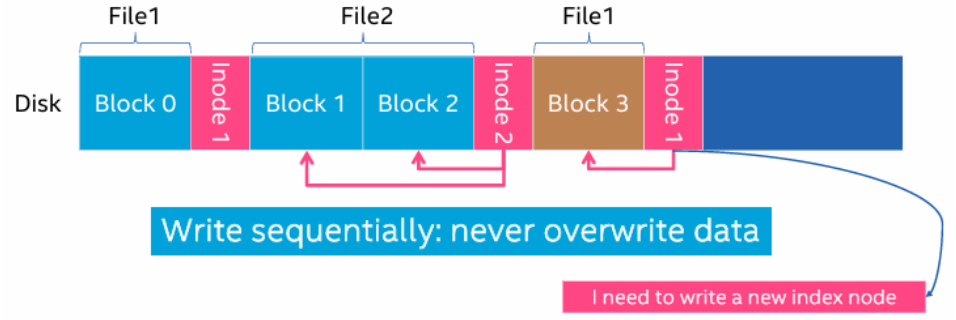
Log-structured file system

The basic concept is to never overwrite data, so to save data in a different location each time, sequentially.

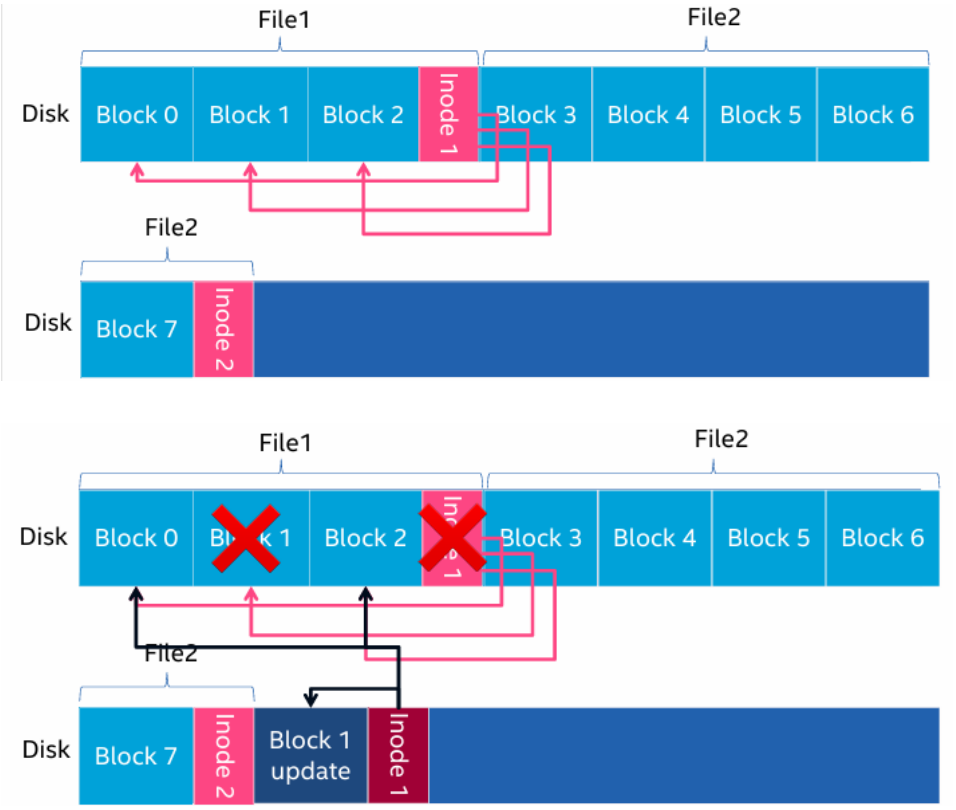


In order to find the correct version of the file we can use and index node to locate files.

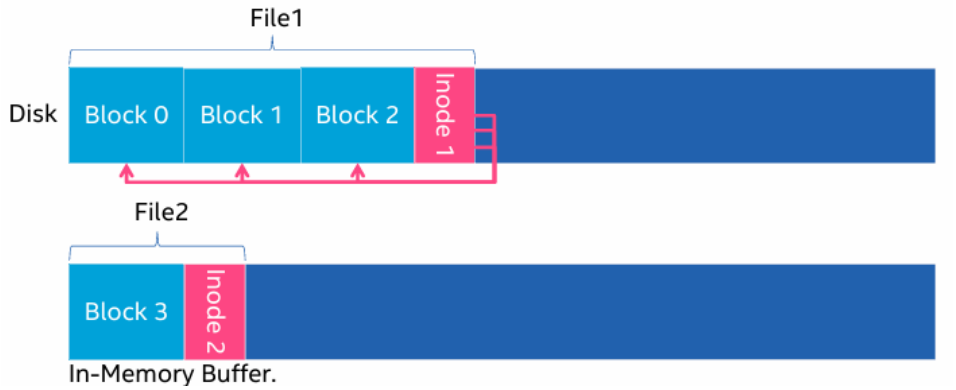




If we have two files, composed of many blocks, what can we do is:



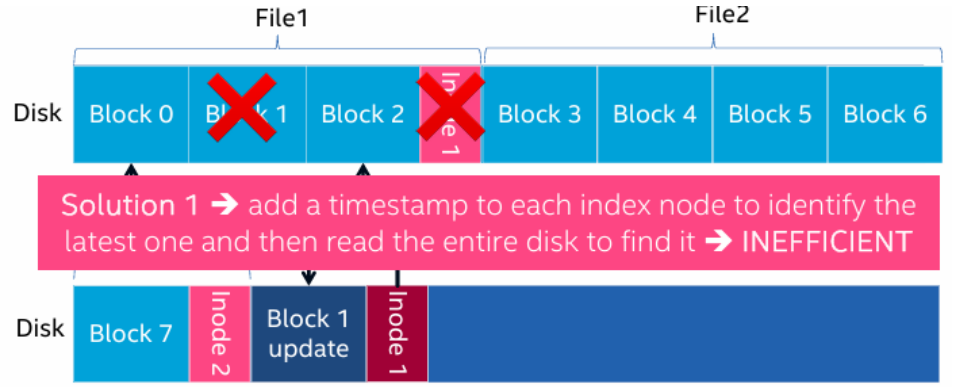
When is it time to write?



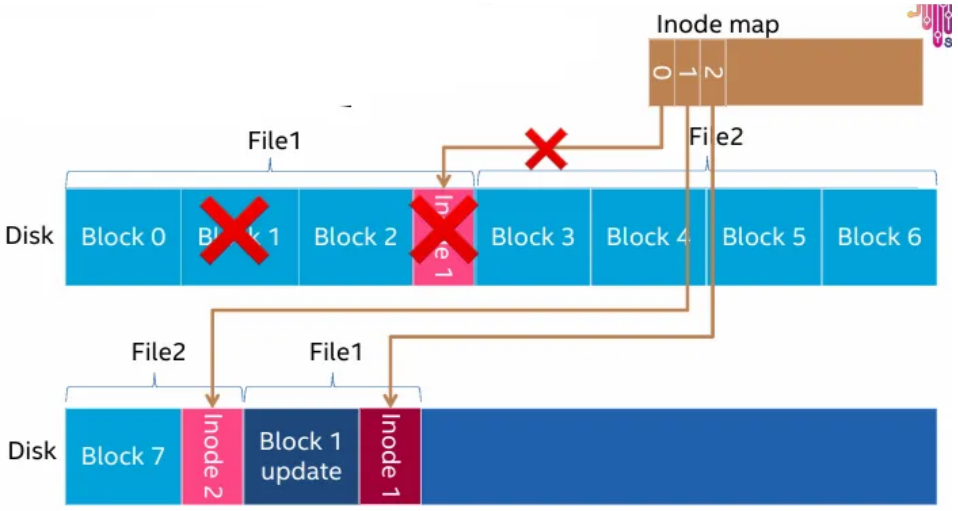
Using in-memory buffer can be a problem because we may lose data in the case of a power off.

If we have more version of the inode of the same file, how do I find a file?

1. You can add a timestamp to each index node to identify the latest one and then read the entire disk to find it. This solution is inefficient if you have a big flash memory.

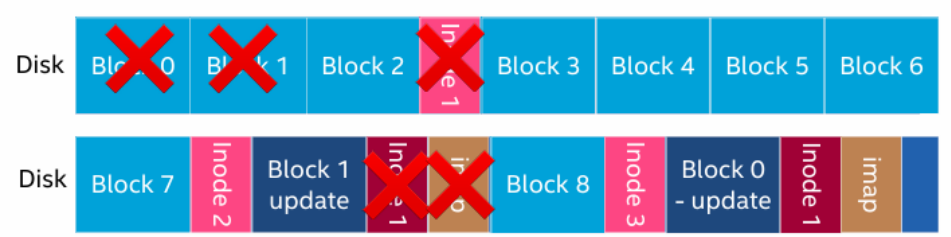


2. You can create a file map (imap), that contains pointers to the inodes. If we have to change something we can set a new entry in the imap.



In order to make the imap persistent, it should be written on disk at the end of each write, when necessary. It is small (4 bytes * #inodes) and sequential writes are cheap.

When the disk is full we can use the garbage collection, which starts from the imap to find all reachable inodes and from there all reachable blocks. Everything else is garbage. Whatever you can reach from the imap is still in use, the remainings elements are garbage.



Nowadays, in the log-structured file system there is no need for sequential writes, you just need to find unused blocks. You can do 1→0 rewrites, maintaining a bitmap of used blocks at fixed places. It has lots of complexities, because of bits wear out, read disruption, etc.

Embedded file systems

- LittleFS: little fail-safe filesystem designed for microcontrollers
- Yaffs (Yet Another Flash File System): an open-source file system specifically designed to be fast, robust and suitable for embedded use with NAND and NOR flash

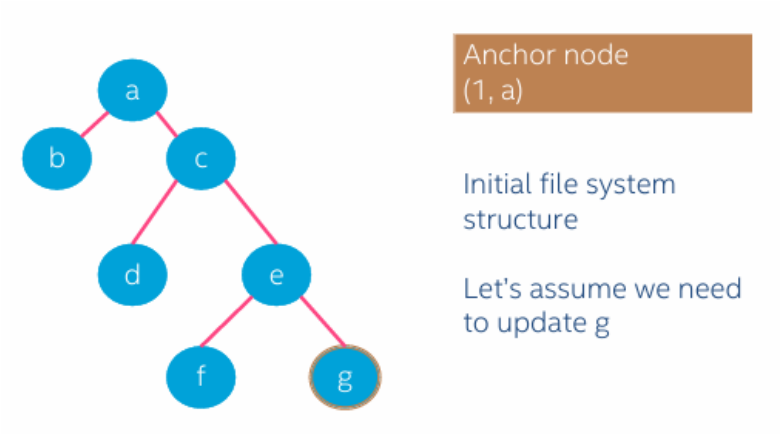
Journaling file system

In embedded systems power failures can be recurring events. File systems not completely updated can be corrupted due to power file. The solution is to adopt the journaling.

The journal keeps track of the changes that will be made to the file system in a special area of the disk named journal before committing them to the main file system. The journal is usually a circular log in a dedicated area of the file system. The file system records the changes it will make ahead of time, and in the event of a system crash of power failure, recovery involves reading the journal from the file system and replaying changes from this journal until the file system is consistent again. In this type of file system changes are atomic, as they are either succeed or are not replayed at all.

Example: LOGFS

LOGFS is designed for raw flash devices (NAND, NOR), and it is based on a tree. Update is done out-of-place, so the new value of updated data is stored separately from the old value. The root node is locatable in O(1) time since an anchor node is used. This is used to find a tree's root node in flash. A flash block stores a list of (version, offset) tuples, so the highest version is the current root node of the flash. When a file system is mounted the anchor node is scanned looking for the root node. A write operation is completed by updating the anchor node.



If any failure takes place before updating anchor mode, changes are discarded. If any failure takes place during anchor node update, changes are discarded. If failure takes place after updating anchor node, at the next system boot the file system is consistent.