1 FSCK and Journaling

Vocabularies

1. **Persistence**
    - Refers to object and process characteristics that continue to exist even after the process that created it ceases or the machine it is running on is powered off
2. **Inconsistent state**
    - Is a state where an operation from A to B is partially complete
3. **Crash Consistency Problem**
    - Is the problem of maintaining persistent data structure despite the presence of a power loss or system crash
4. **Page Cache**

Question What is the definition of page cache? It's definition is weakly defined on google :'(

- Is also called **Disk Cache**
- Is the main disk cache used by the Linux kernel when reading from or writing to disk

5. **Buffer Cache**
    - Is a cache to relieve processes from having to wait for relatively slow disks to retrieve or store data
6. **FSCK**
    - Is a tool for checking the consistency of a file system in Unix and Unix-like operating systems, such as Linux, macOS, and FreeBSD
7. **Physical Logging**
    - Is a type of journaling that logs an advance copy of every block that will later be written to the main file system
8. **Logical Logging**
    - Is a type of journaling that stores only changes to file metadata (i.e. update instructions) in the journal.
9. **Checkpointing**

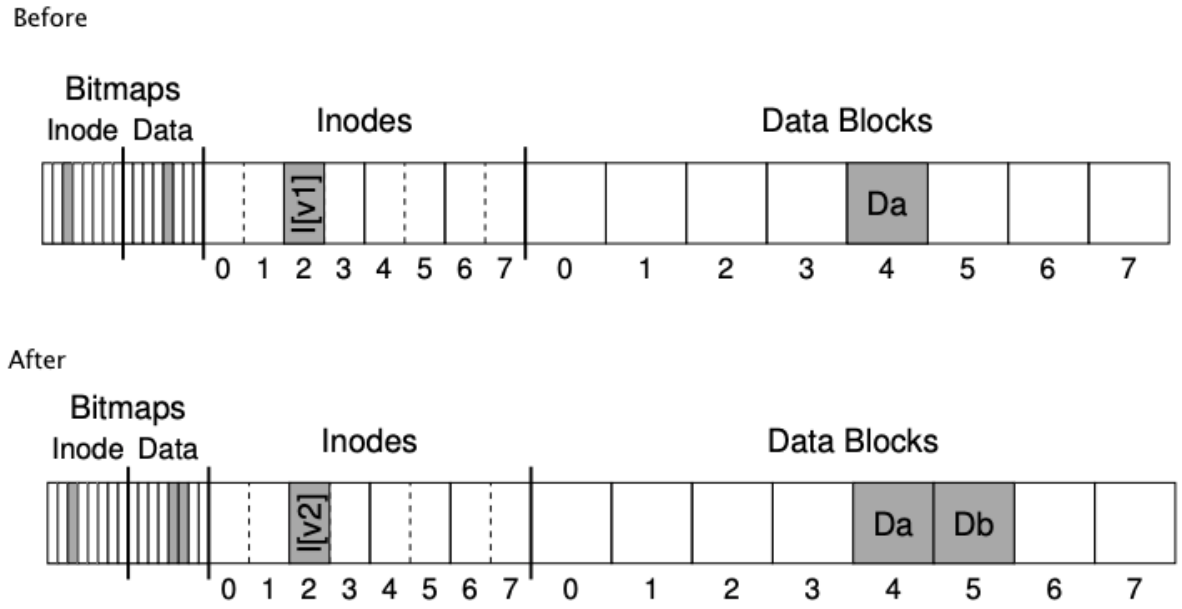
- Is the process of overwriting the old structures in the file system when its transaction is safely on disk

10. **Circular Log**
    - Is also called **circular buffer**
    - Is a data structure that uses a single, fixed-size buffer as if it were connected end-to-end
11. **Data Journaling**
    - Is a type of journaling that writes all data and metadata regarding transaction to a data structure called **journal** before making changes to the file system
12. **Metadata journaling (Ordered journaling)**
    - Is a simpler type of journaling that's nearly the same as data journaling, except user data is not written to the journal
13. **Copy on Write**
    - Is an optimization strategy that if multiple callers ask for resources which are initially indistinguishable, you can give them pointers to the same resource
    - This function is maintained until a caller tries to modify its "copy" of the resource, at which point a true private copy is created to prevent the changes becoming visible to everyone else

1.1 Detailed Example

- Desired: **atomic** updates. That is, on crash, the file on write is either in (state 1 - before the file got updated) or (state 2 - after the file got updated)
- Reality: This is not possible
- Is the reason why computers have 'Don't turn off computer' message

Crash Scenarios



1) Just the data block (Db) is written to disk

- No inode that points to it
- No bitmap that says the block is allocated
- It is as if the write never occured
- There is no problem here. All is well. (In file system's point of view)

2) Just the updated inode (I[v2]) is written to disk

- Inode points to the disk where `Db` is about to be written
- No bitmap that says the block is allocated
- No `Db` is written
- Garbage data will be read
- Also creates **File-system Inconsistency**
  - Caused by on-disk bitmap telling us Db 5 is not allocated, but inode saying it does

3) Just the updated bitmap (B[v2]) is written to disk

- Bitmap indicates tht block 5 is allocated

- No inode exists at block 5
- Creates **file-system inconsistency**
- Creates **space-leak** if left as is
  - block 5 can never be used by the file system

4) Inode (I[v2]) and bitmap (B[v2]) are written to disk, and not data

- File system metadata is completely consistent (in perspective of file system)
- Garbage data will be read

5) Inode (I[v2]) and data block (Db) are written, but not the bit map

- Creates **file-system inconsistency**
- Needs to be resolved before using file system again

6) Bitmap (B[v2]) and data block (Db) are written, but not the inode (I[v2])

- Creates **file-system inconsistency** between inode and data bitmap
- Creates **space-leak** if left as is
  - Inode block is lost for future use
- Creates **data-leak** if left as is
  - Data block is lost for future use

1.2 Solution #1: File System Checker

- Basic Idea: Let inconsistencies happen and fix them later (when rebooting)
- Is used by UNIX tool **fsck** ('file system checker')
- Summary of how it works
  - **Inode State**
    - Corruption in file is checked (e.g. does it have valid file type such as directory file, or links)
    - Solved by removing it, and updating the bitmap if inode cannot be fixed easily
  - **Inode links**
    - Number of references in each inode is checked

    - Check is done by reading the entire directory tree and building its own link count
    - Solved by fixing the count if there is mismatch, or by moving to `lost+found` directory if there is no directory refers to it
  - **Duplicates**
    - Duplicate pointers (i.e. two different inodes pointing to same block) is checked
    - Solved by either removing one of two inodes, or creating a copy for each
  - **Bad Blocks**
    - A pointer that points to something outside is partition is checked
    - Solved by removing the block
  - **Directory Checks**
    - Making sure that `.` and `..` are first entry is checked
    - Allocation of inodes referred to in a directory entry is checked
    - Making sure that no directory is linked more than once is checked

- Disadvantage
  - Way too slow. May take Hours.
  - Wasteful (Make mistake once, and check everything)
  - Doesn't solve all problems (e.g. inode with incorrect data blocks)

1.3 Solution #2: Journaling

- Is a popular solution to **crash-consistency problem**
- Many file systems use this idea (e.g. ext3, ext4, windows NTFS)
- Basic idea
  - before overwriting the structures in place, write down (in a well-known location) a little note of what you are about to do
  - If crash occurs, read note and try again

- Advantage
  - Greatly reduces amount of work required during recovery

Transaction Beginning (TxB)

- Where does computer read update instruction (journal ? journal superblock ?)?
- In data Journaling, where is comitted data generated and stored prior to putting it in file system?
- Includes information about current update
- Contains **Transaction Identifier** or TID

Transaction End (TxE)

- Is marker of the end of transaction
- Also contains **Transaction Identifier** or TID

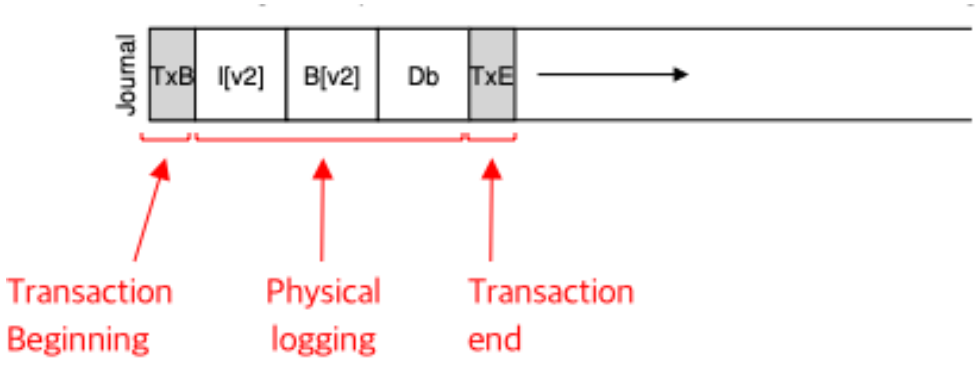
Checkpointing

- Act of overwriting of old structure in the file system between **transaction beginning** and **transaction end**

Journaling Superblock

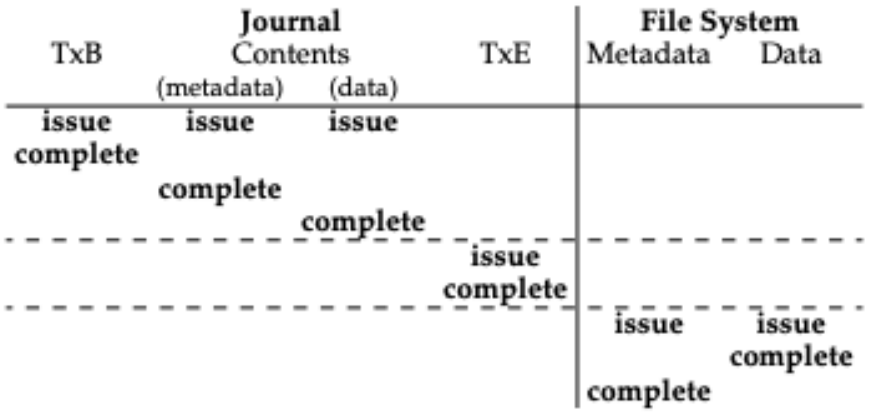
- Records information on which transactions have not yet been checkpointed
- Oldest and newest non-checkpointed transactions exist here
- Is different from file system superblock

Data Journaling

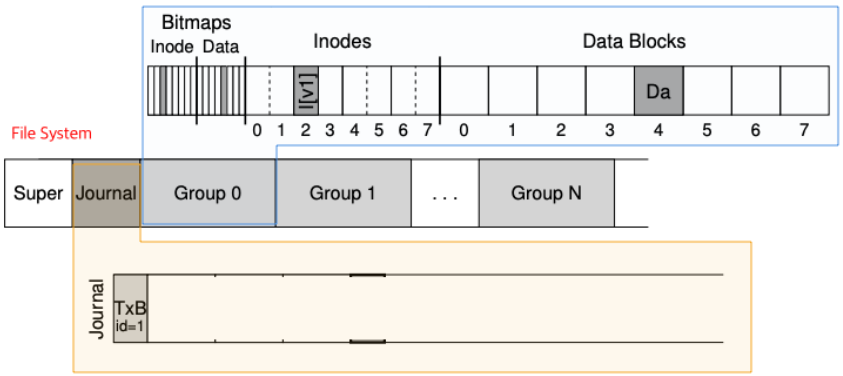


Important Is written to journal before putting onto file system!!!

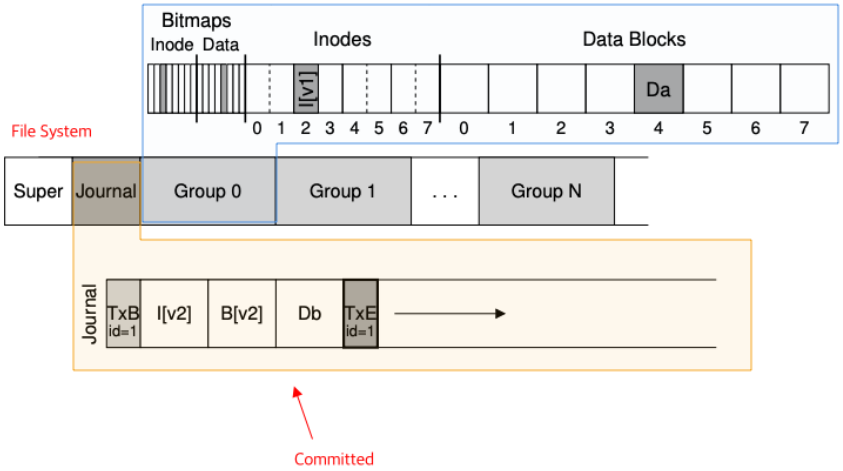
- Steps



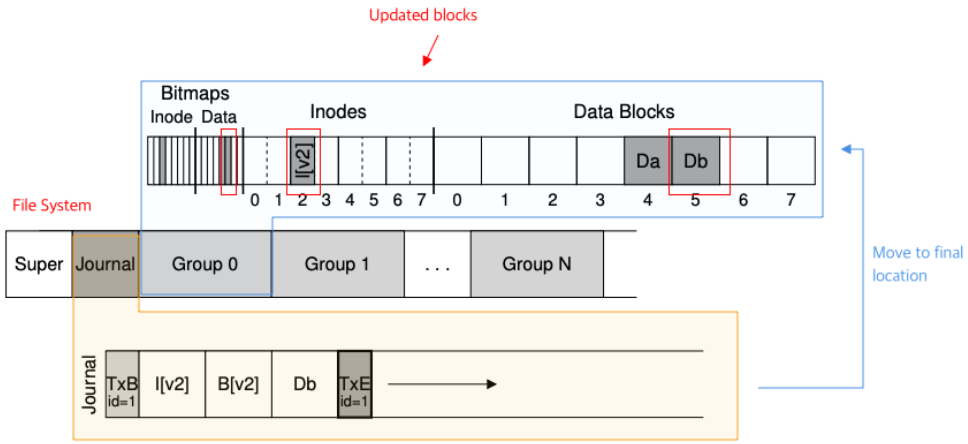
1. **Journal Write**: Write the contents of the transaction (including TxB, metadata and data) to log



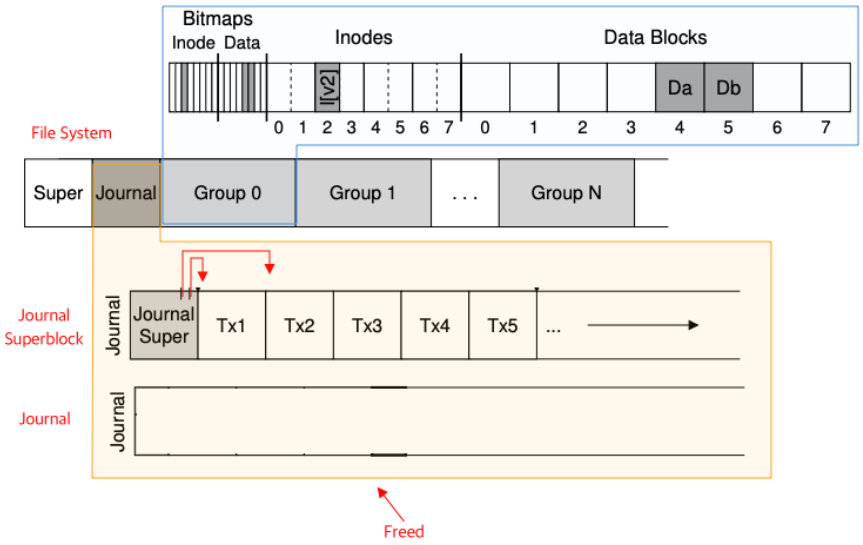
2. **Journal Commit:** Write the transaction commit block (containing TxE) to log; wait wait for write to complete
   - After this, transaction is **committed**



3. **Checkpoint:** Write the contents of the update (metadata and data) to their final on-disk location



4. **Free:** Mark the transaction free in the journal by updating the journal superblock



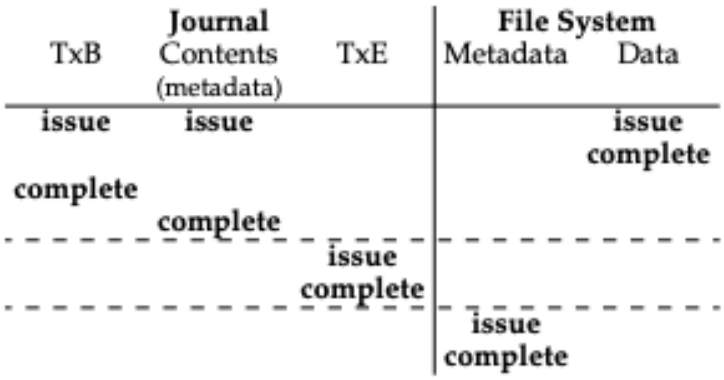
5. Repeat until done

- Disadvantage
  - Each data block is written twice
- Recovery Steps
  - Crash at step 1 → skip pending update
  - Crash during step 2 and 3 → replay the update
    - Happens during boot

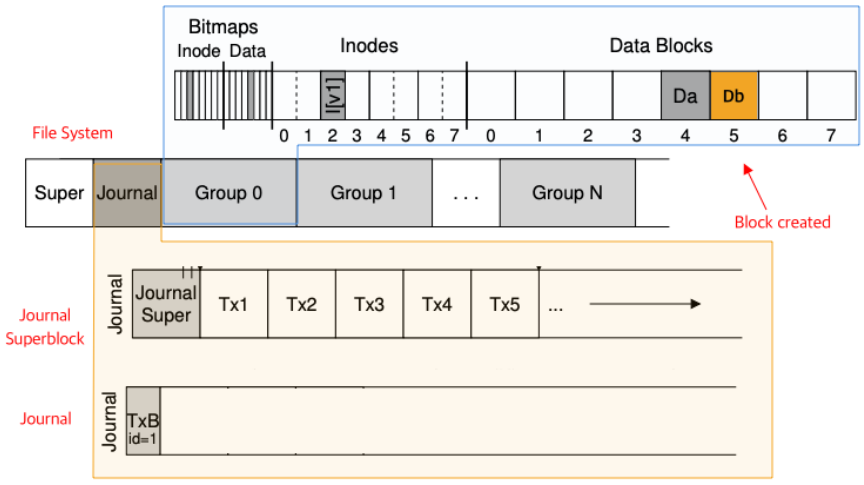
Metadata Journaling

- Goal: Reduce number of writes
- Data block is written to file system first
- Metadata (inode and bitmap information) are written to journal before checkpoint
- Is order dependent
  - e.g. I[v2] and B[v2] make to disk and data block does not
  - If data block is a garbage data, file-system will assume all is okay
  - Writing data block first guarentees that a pointer will never point to garbage

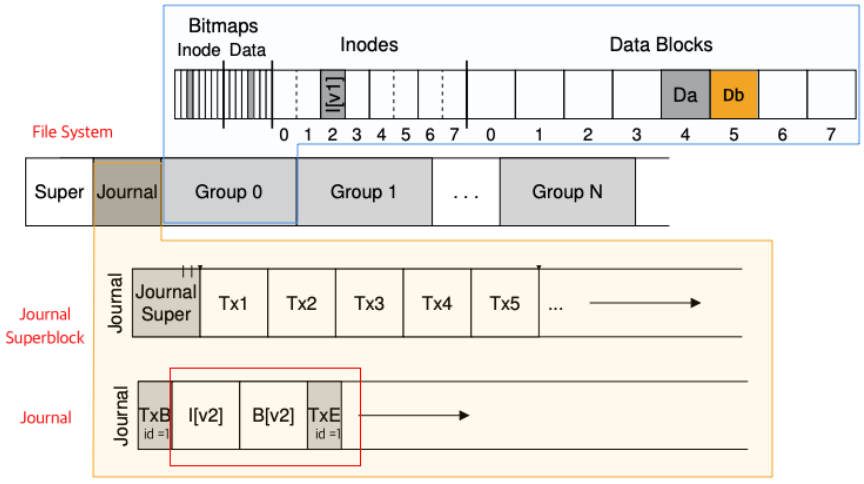
- Steps



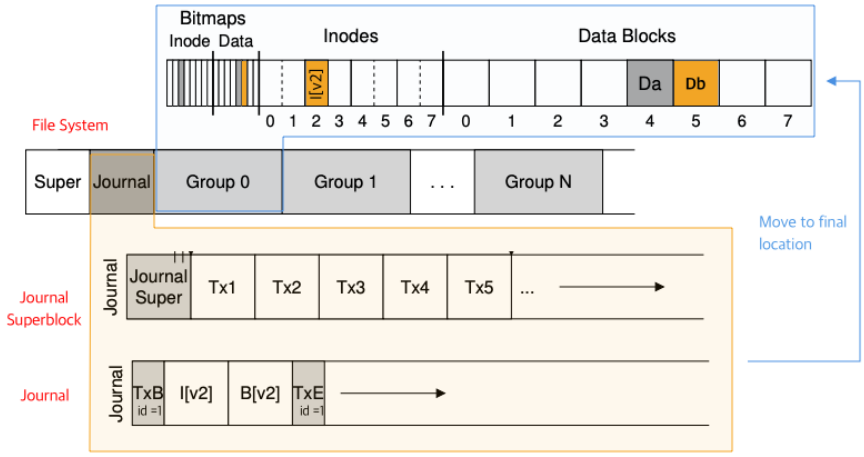
1. **Data Write:** Write data to final location; wait for completion



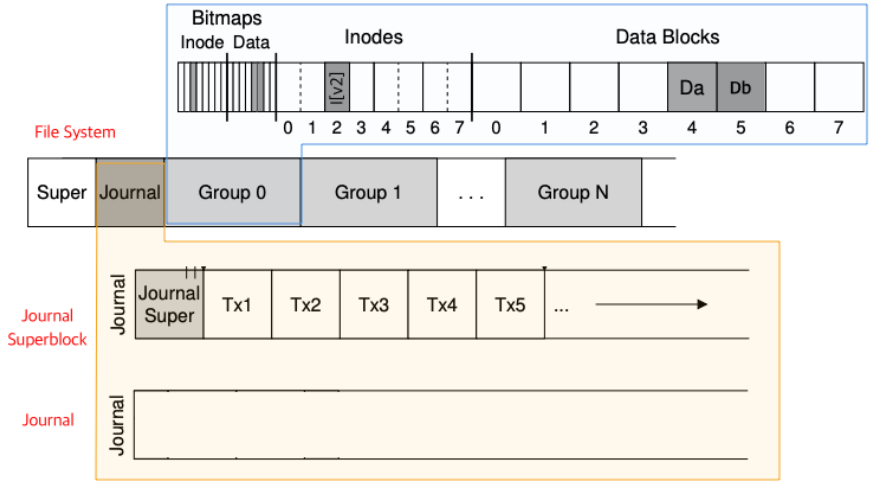
2. **Journal Metadata Write:** Write the begin block and metadata to the log; wait for writes to complete
3. **Journal Commit:** Write the transaction commit block (containing TxE) to the log; wait for the write to complete



4. **Checkpoint Metadata:** Write the contents of the metadata update to their final locations within the file system



5. **Free:** Mark the transaction free in journal superblock



- Block Reuse
  - Never reuse blocks until checkpointed out of the journal
- Advantage
  - Solves double write problem in **data journaling**

1.4 Solution #3: Other Approaches

- **Copy on Write**
  - Is developed based on **log-structured file system**