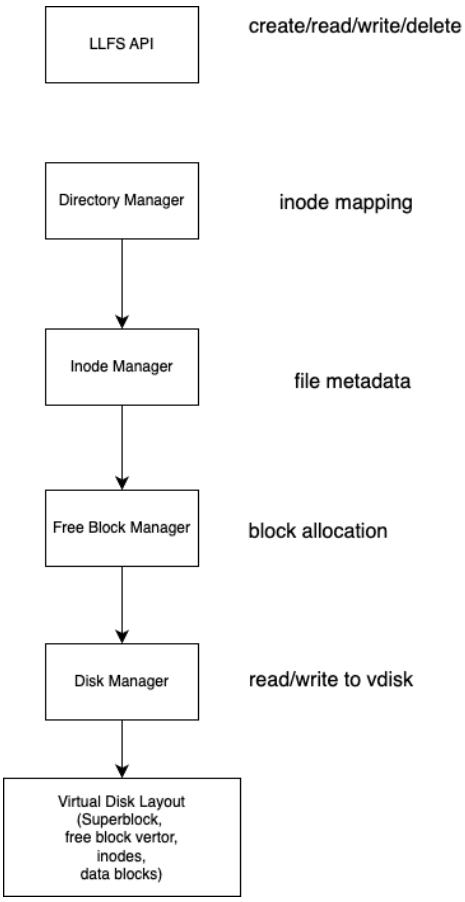

1. **Disk Initialization**:

- Create a virtual disk file (e.g., `vdisk`) to simulate the disk.
- Divide the disk into blocks (each 512 bytes).
- Reserve special blocks (superblock, free block vector).

Input: Disk file path, size, block size.

Process:

- Format disk: Initialize the superblock, free block vector, and root directory.
- Create a clean disk layout.

Output: Disk ready for file system operations.

2. **Superblock Management**:

- Contains metadata about the file system (e.g., magic number, block count, inode count).

Free Block Management:

- Use a bitmap in a specific block to track free and allocated blocks.

Inode Management:

- Each file or directory is associated with an **inode**.
- Inodes store file metadata (size, type, block pointers).

Directory Structure:

- Directories map file names to inode IDs.
- A hierarchical tree-like structure is maintained.

File Operations:

- **Create**: Allocate an inode, update directory entries, and write metadata.
- **Read/Write**: Map logical file offsets to disk blocks and perform I/O.
- **Delete**: Remove inode pointers, mark blocks as free, and clean up directories.

File Creation

- **Input**: File name and path.
- **Process**:
    1. Allocate an inode for the file.
    2. Update the directory structure to include the file name and inode ID.
    3. Mark blocks as allocated for future file writes.
- **Output**: File created in the file system.

3. File Write

- **Input**: File path, data to write.
- **Process**:
    1. Locate the file's inode via the directory structure.
    2. Write data to free blocks sequentially.
    3. Update the inode with block pointers and file size.
- **Output**: Data written to the file.

4. File Read

- **Input**: File path, read offset, size.
- **Process**:
    1. Locate the file's inode via the directory structure.
    2. Map the logical offset to disk blocks.
    3. Read data from the blocks.
- **Output**: File data.

5. File Deletion

- **Input**: File path.
- **Process**:
    1. Locate the file's inode via the directory structure.
    2. Mark data blocks as free in the free block vector.
    3. Remove the inode and directory entry.
- **Output**: File removed.

Crash Recovery:

- Use logs and checkpointing to ensure consistency after a crash.
- **Input**: Disk state after a crash.
- **Process**:
    - Use logs to identify uncommitted transactions.
    - Restore consistency by replaying or undoing operations.
- **Output**: Restored file system state.

DiskManager: Simulate a disk using a file (`vdisk`). Provide block-level read and write operations.	

Public	void formatDisk();
	void writeBlock(size_t blockNumber, const std::vector<char>& data);
	std::vector<char> readBlock(size_t blockNumber);

private	std::fstream diskFile; size_t blockSize; size_t totalBlocks;

struct Inode {

uint32_t fileSize;

uint32_t flags; // Type of file (file/directory)

uint16_t directBlocks[10];

uint16_t singleIndirectBlock;

uint16_t doubleIndirectBlock; };

InodeManager: Manage inodes and metadata. Map file names to disk blocks.	

public	void initializeInodes(size_t totalInodes);
	int allocateInode();
	void freeInode(int inodeId);
	Inode getInode(int inodeId);
	void updateInode(int inodeId, const Inode& inode);

private	std::vector<bool> inodeBitmap; std::vector<Inode> inodeTable;

FreeBlockManager: Track allocated and free blocks using a bitmap.	

public	int allocateBlock();
	void freeBlock(int blockNumber);
	bool isBlockFree(int blockNumber);

private	std::vector<bool> freeBlockBitmap;

struct DirectoryEntry {

uint8_t inodeId;

char fileName[31]; };

DirectoryManager: Maintain directory structure and entries.Map directory names to inodes.	

public	void createRootDirectory();
	void addEntry(const std::string& path, const std::string& fileName, uint8_t inodeId);
	DirectoryEntry getEntry(const std::string& path, const std::string& fileName);
	void removeEntry(const std::string& path, const std::string& fileName);

private	std::unordered_map<std::string, std::vector<DirectoryEntry>> directoryTable;

LLFS: Provide an interface for file system operations (create, read, write, delete).	

Public	void formatFileSystem();
	void createFile(const std::string& path);
	void writeFile(const std::string& path, const std::vector<char>& data);
	std::vector<char> readFile(const std::string& path);
	void deleteFile(const std::string& path);
	void createDirectory(const std::string& path);
	void deleteDirectory(const std::string& path);

Private	DiskManager diskManager; InodeManager inodeManager; FreeBlockManager freeBlockManager; DirectoryManager directoryManager;

Tests:

Validate block-level operations, file creation, reading, writing, deletion, and crash recovery.