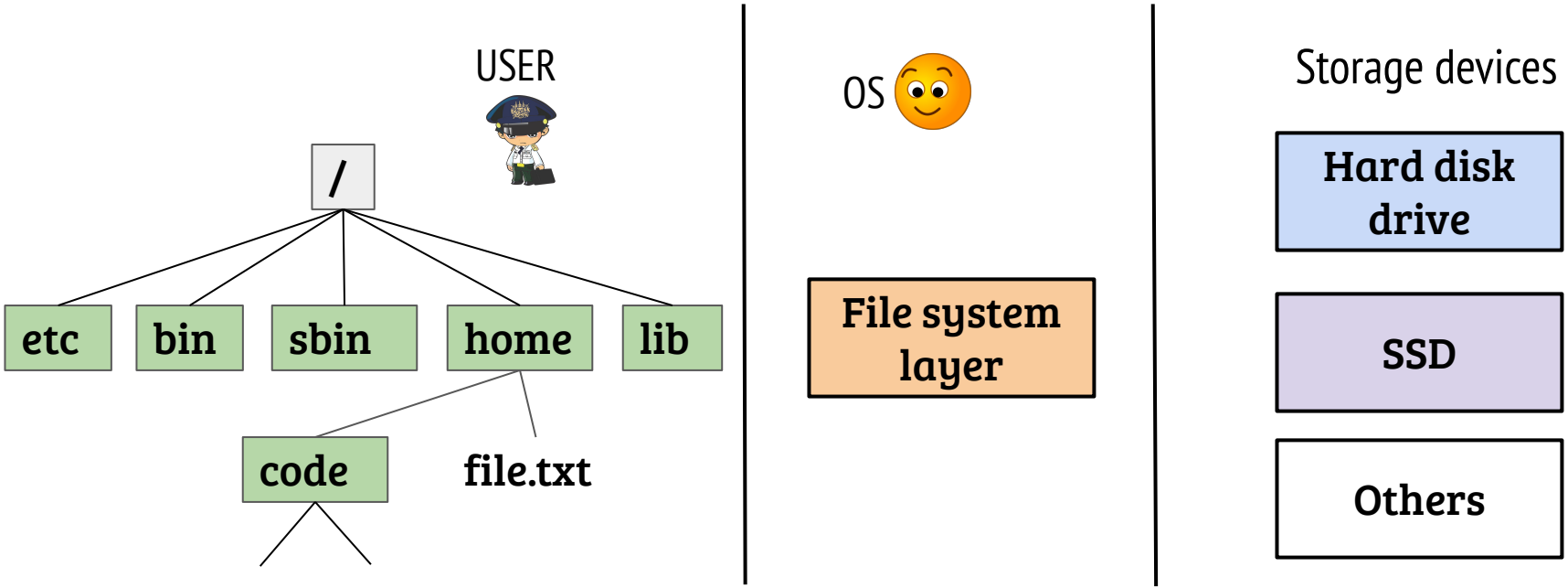
CS330: Operating Systems

Filesystem: caching and consistency

Recap: file system



- File system is an important OS subsystem
  - Provides abstractions like files and directories
  - Hides the complexity of underlying storage devices

Recap: file system organization

- File systems maintain several meta-data structures like super blocks, inodes, directory entries to provide a file system abstractions like files, directories
- How to search/lookup files/directories in a given path?
- Read the content of the root inode and search the next level dir using the name and find out its inode number
- Read the inode to check permissions and repeat the process
- Inode contains the index structures to deduce the disk block address given an logical offset

}

File system and caching

- Accessing data and metadata from disk impacts performance
- Many file operations require multiple block access

File system and caching

- Accessing data and metadata from disk impacts performance
- Many file operations require multiple block access
- Examples:
  - Opening a file

```
fd = open("/home/user/test.c", O_RDWR);
```

File system and caching

- Accessing data and metadata from disk impacts performance
- Many file operations require multiple block access
- Examples:
  - Opening a file

```
fd = open("/home/user/test.c", O_RDWR);
```

  - Normal shell operations

```
/home/user$ ls
```

File system and caching

- Accessing data and metadata from disk impacts performance
- Many file operations require multiple block access
- Executables, configuration files, library etc. are accessed frequently
- Many directories containing executables, configuration files are also accessed very frequently. Metadata blocks storing inodes, indirect block pointers are also accessed frequently

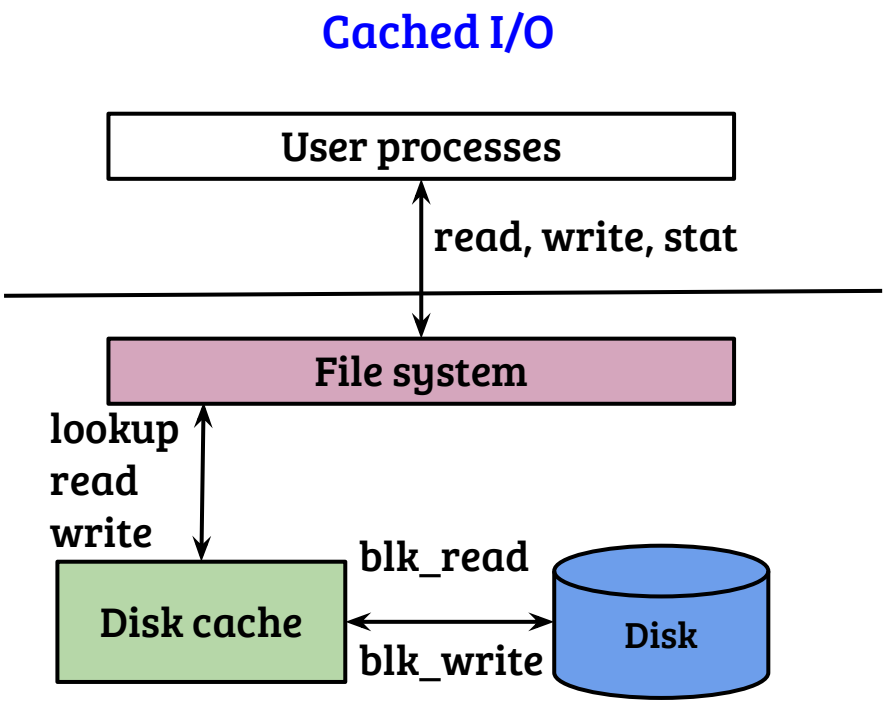
/home/user$ ls

File system and caching

- Accessing data and metadata from disk impacts performance
- Can we store frequently accessed disk data in memory?
  - What is the storage and lookup mechanism? Are the data and metadata caching mechanisms same?
  - Are there any complications because of caching?
  - How the cache managed? What should be the eviction policy?

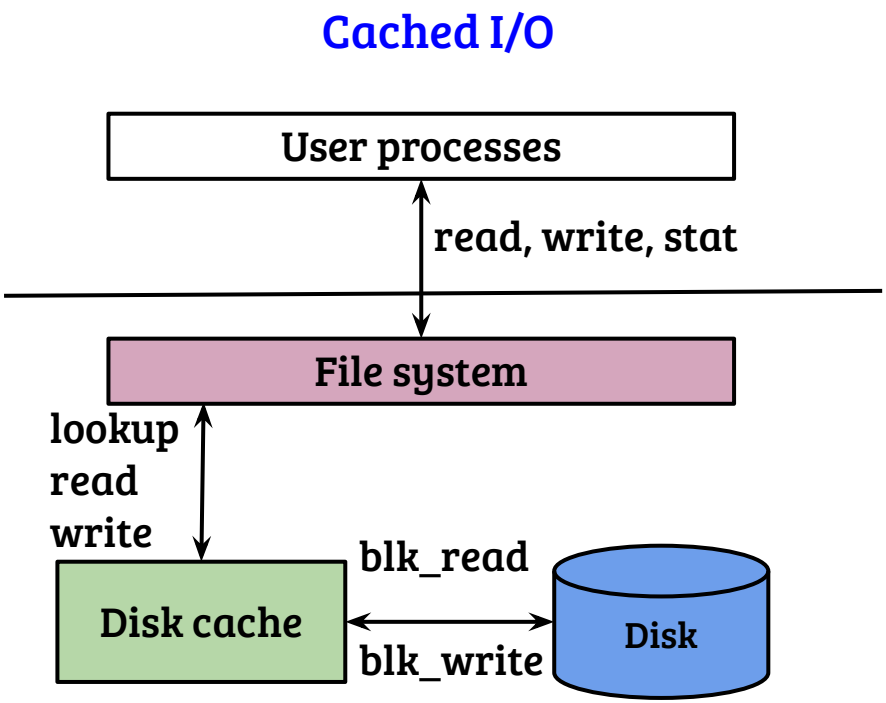
/home/user$ ls

Block layer caching



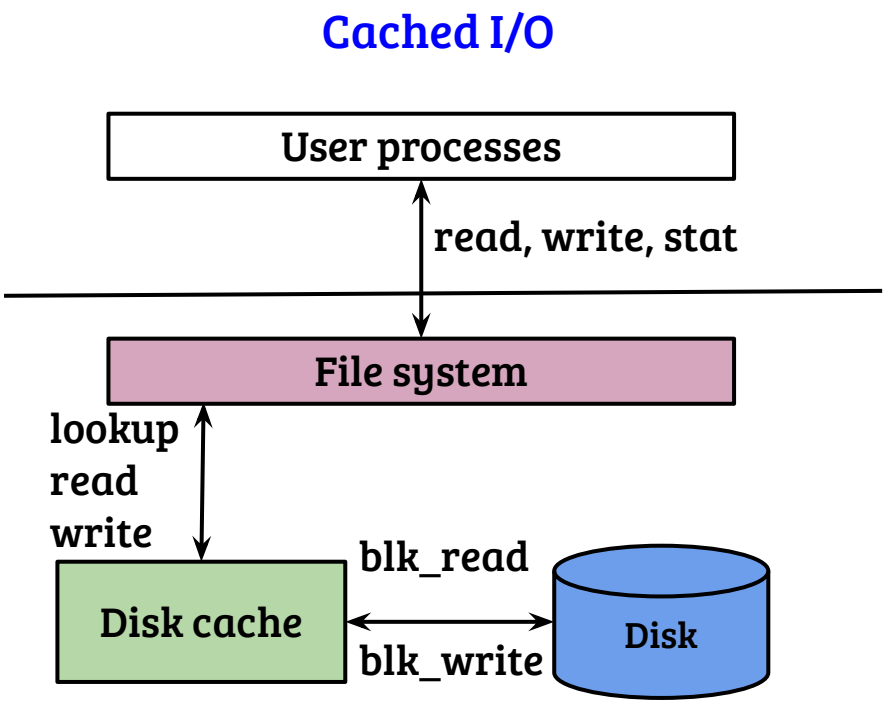
- Lookup memory cache using the block number as the key
- How does the scheme work for data and metadata?

Block layer caching



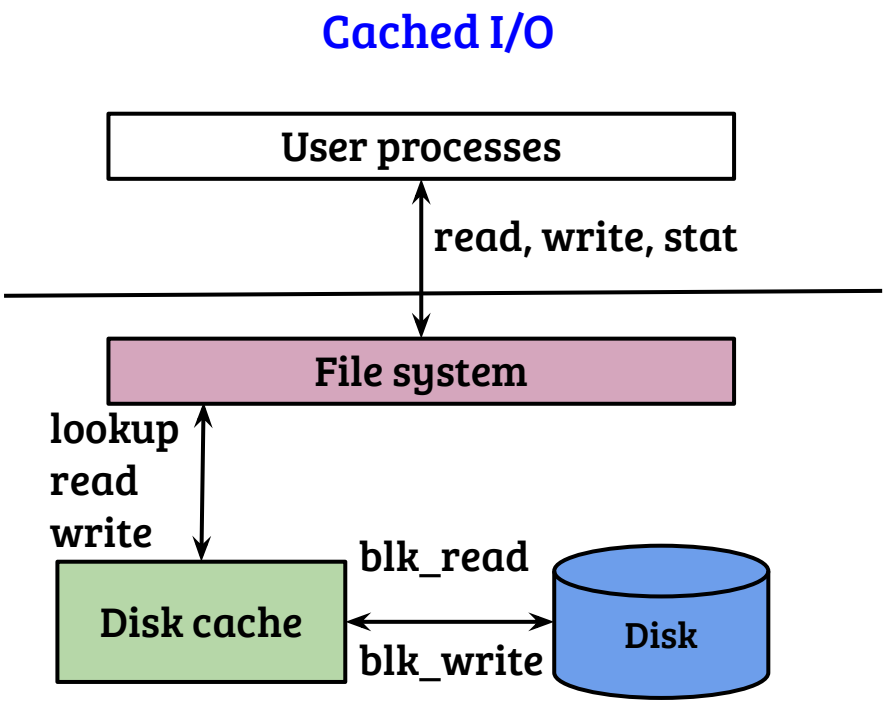
- Lookup memory cache using the block number as the key
- How does the scheme work for data and metadata?
- For data caching, file offset to block address mapping is required before using the cache

Block layer caching



- Lookup memory cache using the block number as the key
- How does the scheme work for data and metadata?
- For data caching, file offset to block address mapping is required before using the cache
- Works fine for metadata as they are addressed using block numbers

File layer caching (Linux page cache)



- Store and lookup memory cache using {inode number, file offset} as the key
- For data, index translation is not required for file access
- Metadata may not have a file association, should be handled differently (using a special inode may be!)

File system and caching

- Accessing data and metadata from disk impacts performance
- Can we store frequently accessed disk data in memory?
  - What is the storage and lookup mechanism? Are the data and metadata caching mechanisms same?
  - File layer caching is desirable as it avoids index accesses on hit, special mechanism required for metadata.
  - Are there any complications because of caching?
  - How the cache managed? What should be the eviction policy?

Caching and consistency

- Caching may result in inconsistency, but what type of consistency?

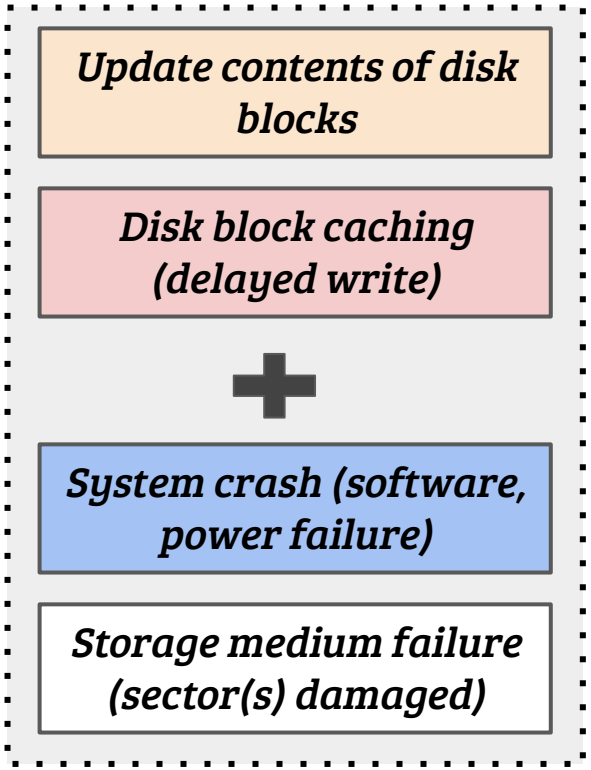
Caching and consistency

- Caching may result in inconsistency, but what type of consistency?
- System call level guarantees
  - Example-1: If a write( ) system call is successful, data must be written
  - Example-2: If a file creation is successful then, file is created.
  - Difficult to achieve with asynchronous I/O

Caching and consistency

- Caching may result in inconsistency, but what type of consistency?
- System call level guarantees
  - Example-1: If a write( ) system call is successful, data must be written
  - Example-2: If a file creation is successful then, file is created.
  - Difficult to achieve with asynchronous I/O
- Consistency w.r.t. file system invariants
  - Example-1: If a block is pointed to by an inode data pointers then, corresponding block bitmap must be set
  - Example-2: Directory entry contains an inode, inode must be valid
  - Possible, require special techniques

File system inconsistency: root causes



- No consistency issues if user operation translates to read-only operations on the disk blocks

- Always keep in mind: device level atomicity guarantees