Filesystems and FAT

CS 4411

Spring 2020

Announcements

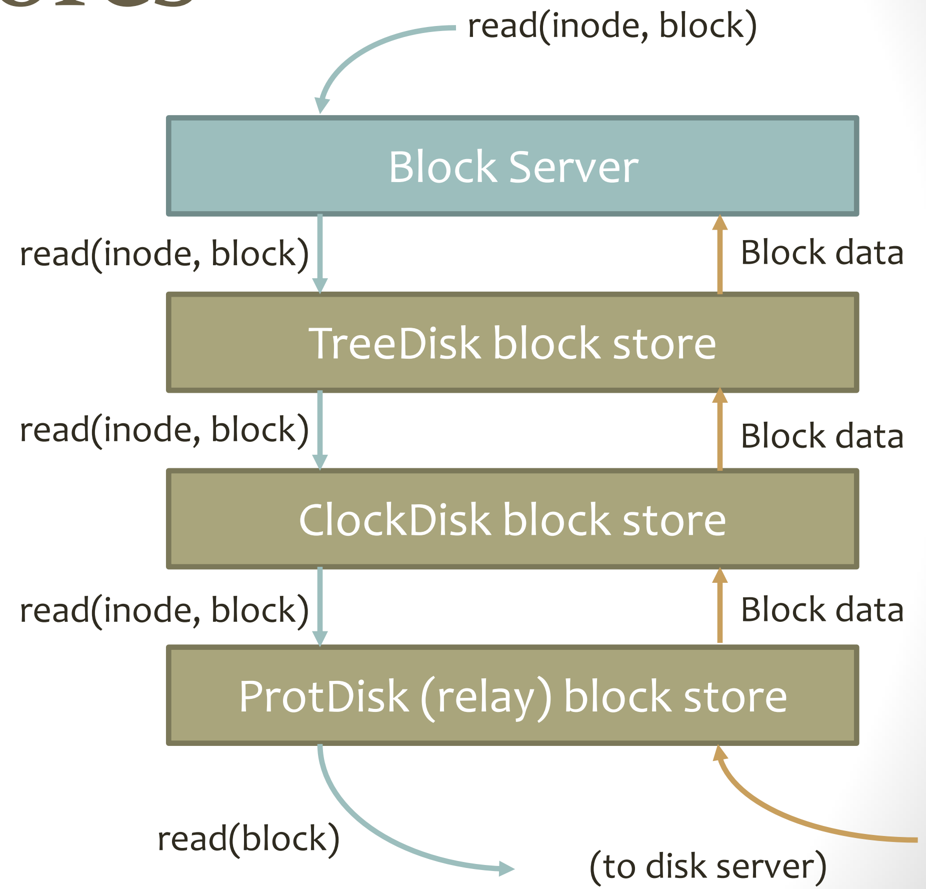
- Major EGOS update
- Careful when using Pull Requests to get updates
- Project 4 Eliminated

Outline for Today

- EGOS File/Storage System
- FAT Filesystem Design
- Project 5 Concepts

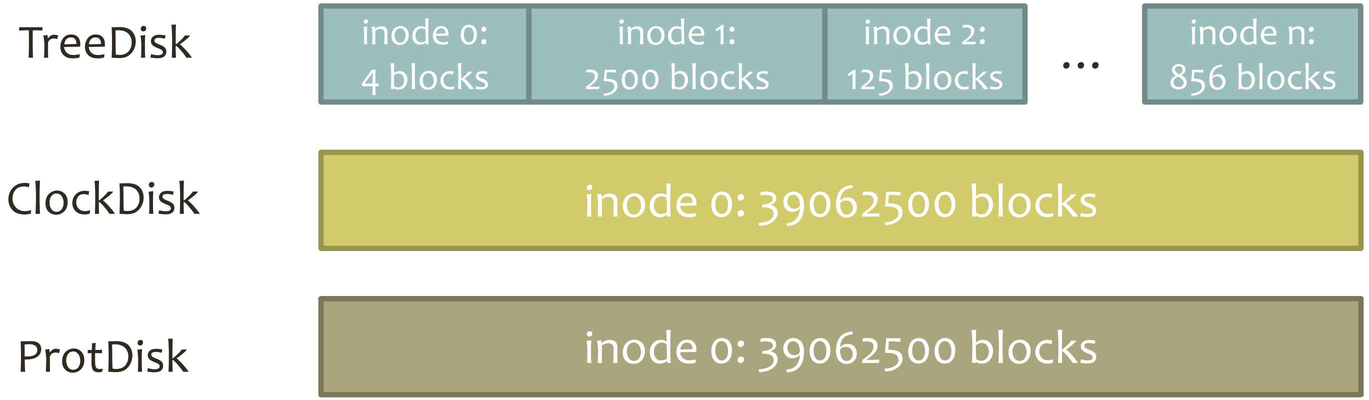
Layered Block Stores

- Within the block server, a **stack** of block stores
- Each block store has the same interface
- Block server sends requests to top of stack
- Each block store knows the block store below it



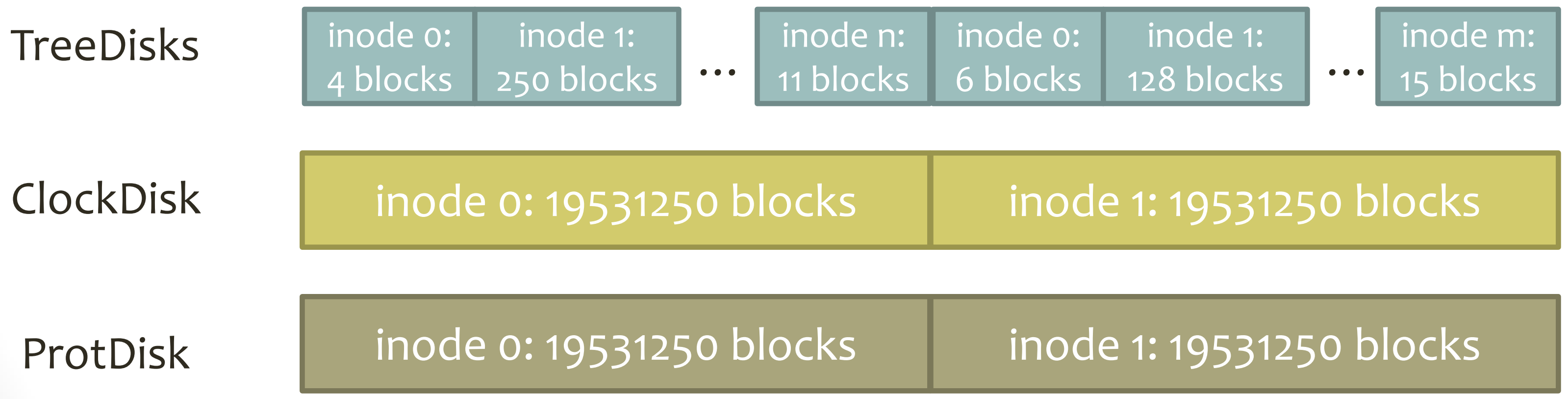
“Inodes” and Virtualization

- Each block store uses inode numbers to group blocks
- Blocks within an inode: a *virtual* block storage device
- TreeDisk partitions a large block store into many VBSs



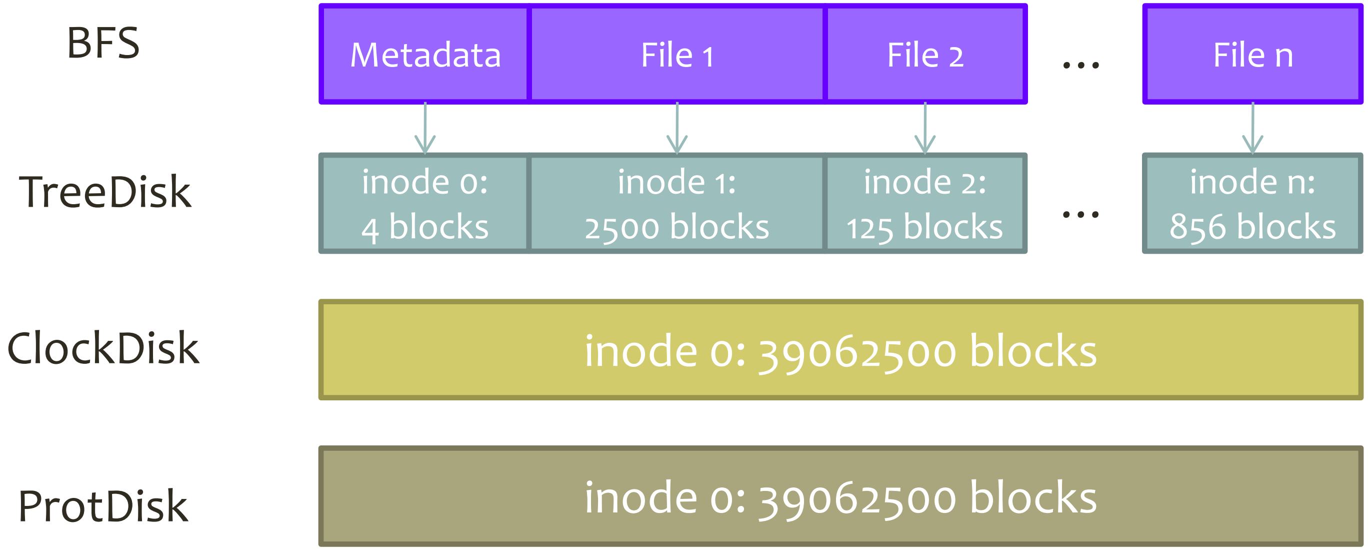
The Magic Number 0

- Really, TreeDisk splits a single *inode* into many inodes
- Need not be inode 0 – disk could use inodes as partitions
- Inode for layer below is (now) a parameter to TreeDisk



Adding the “Filesystem” layer

- Block File Server uses one VBS to store each file
- Metadata: Permissions, which inodes are free

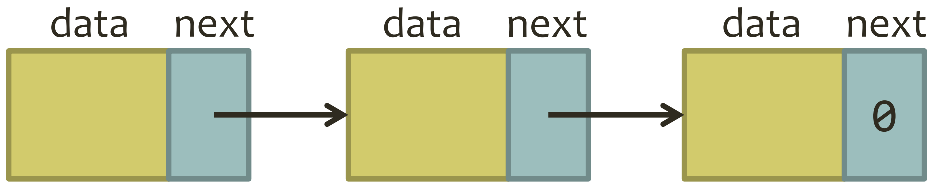


Outline

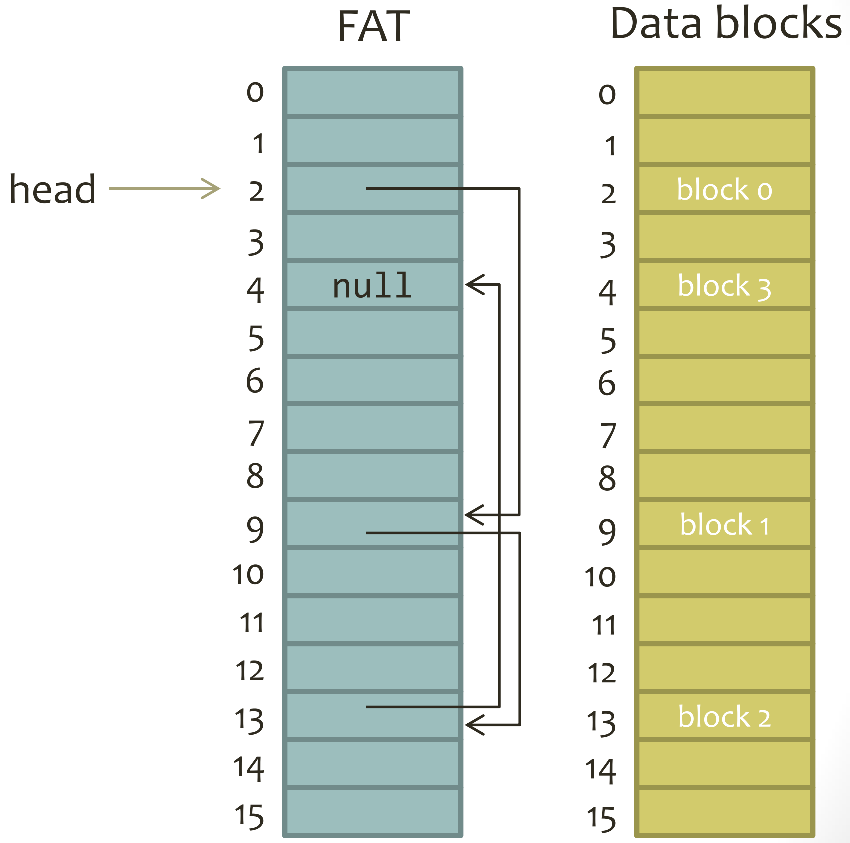
- EGOS File/Storage System
- **FAT Filesystem Design**
- Project 5 Concepts

FAT File System

- Basic idea: Each file is a linked list of blocks

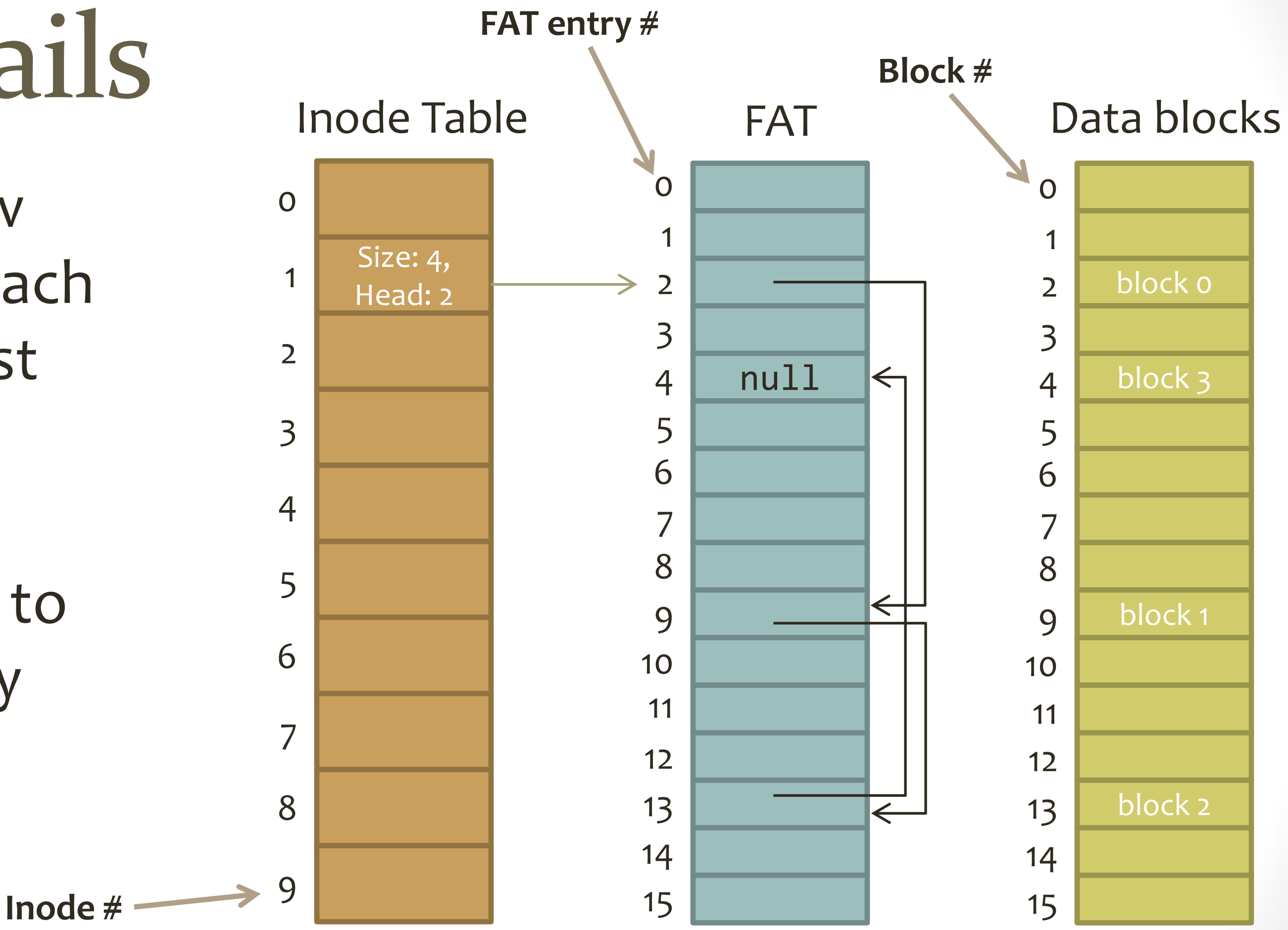


- Instead of storing a “next” pointer in each block, use a parallel table of next pointers



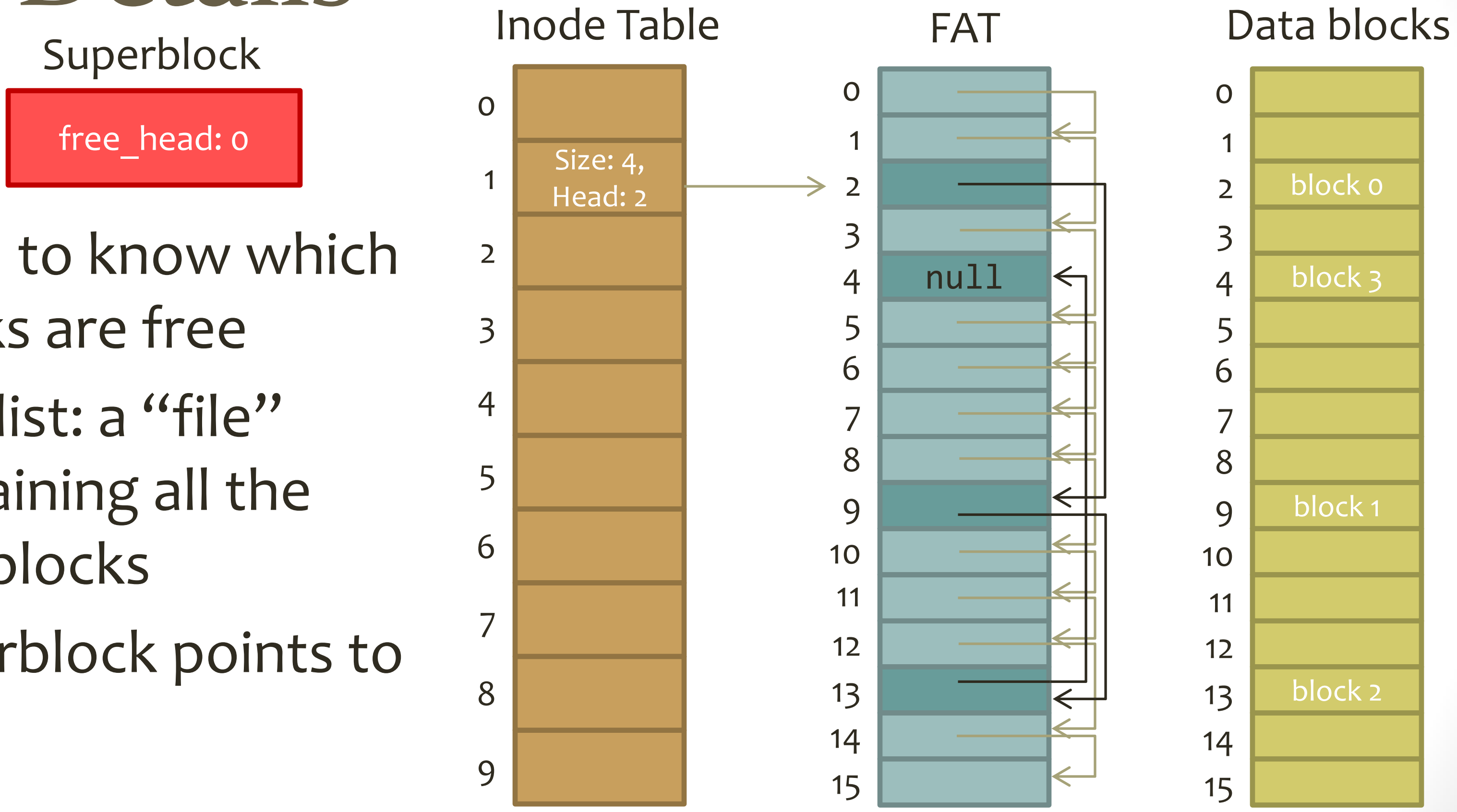
FAT Details

- Need to know the head of each file's linked list
- Inode table: Indexed by inode, points to first FAT entry



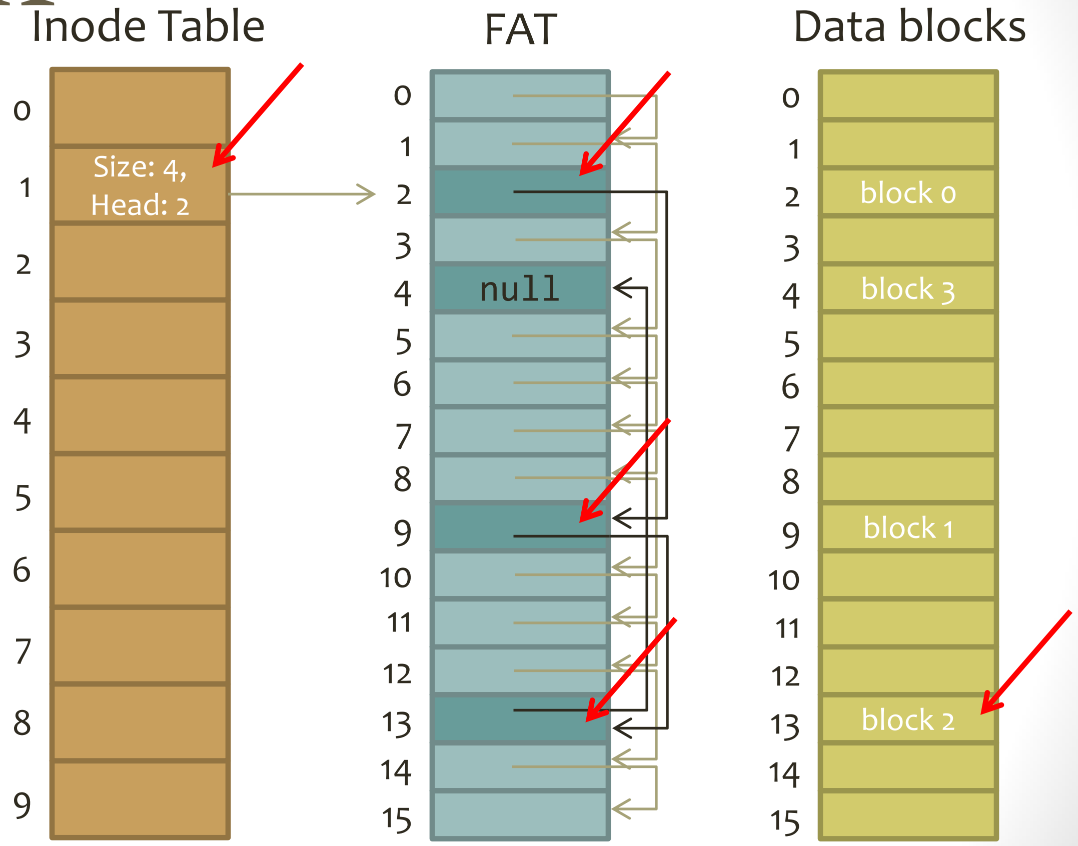
FAT Details

- Need to know which blocks are free
- Free list: a "file" containing all the free blocks
- Superblock points to head



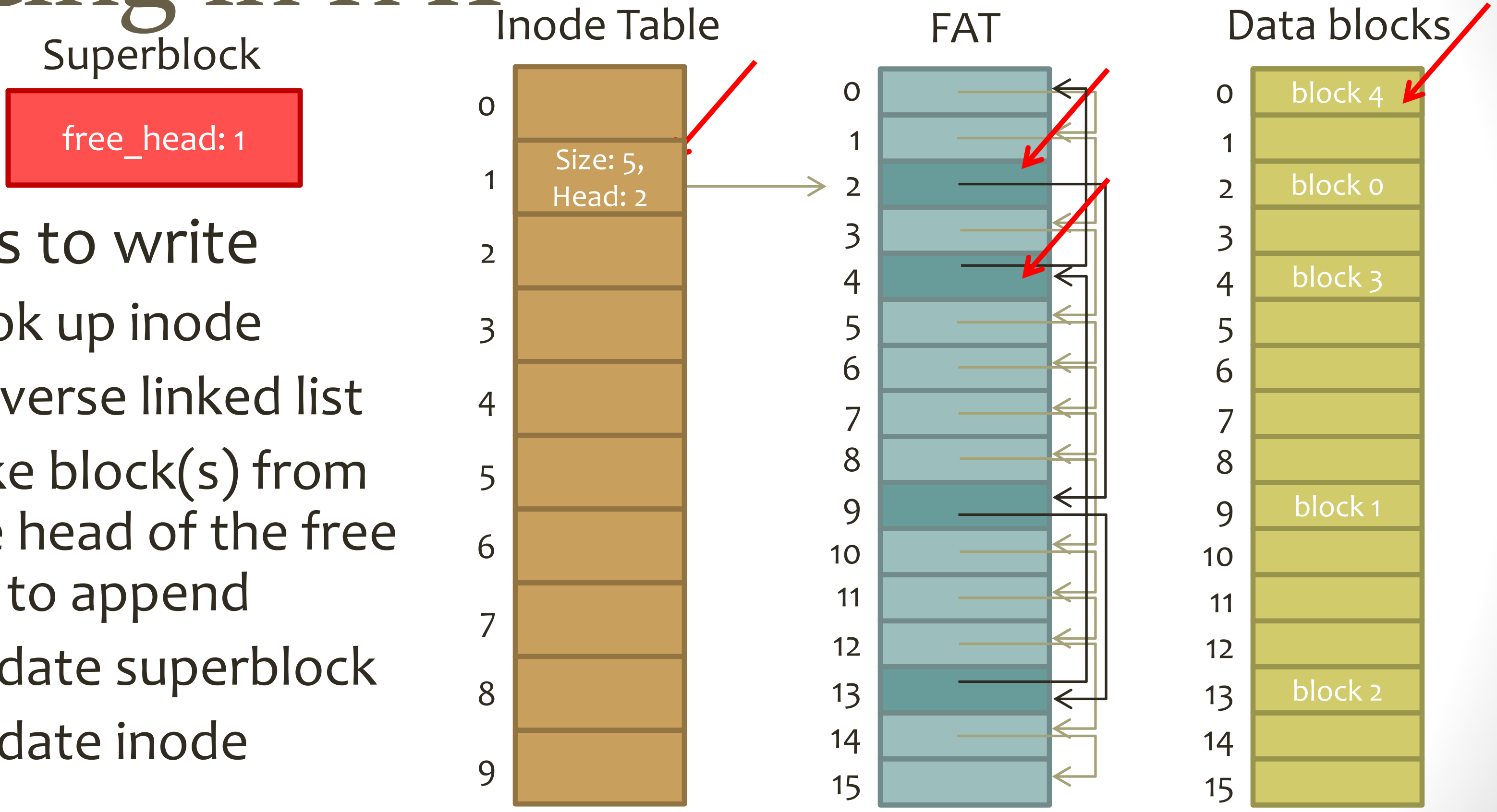
Reading in FAT

- Steps to read:
  - Look up inode
  - Traverse linked list in FAT
  - Read from corresponding data block
- Example: Read block 2 of inode 1



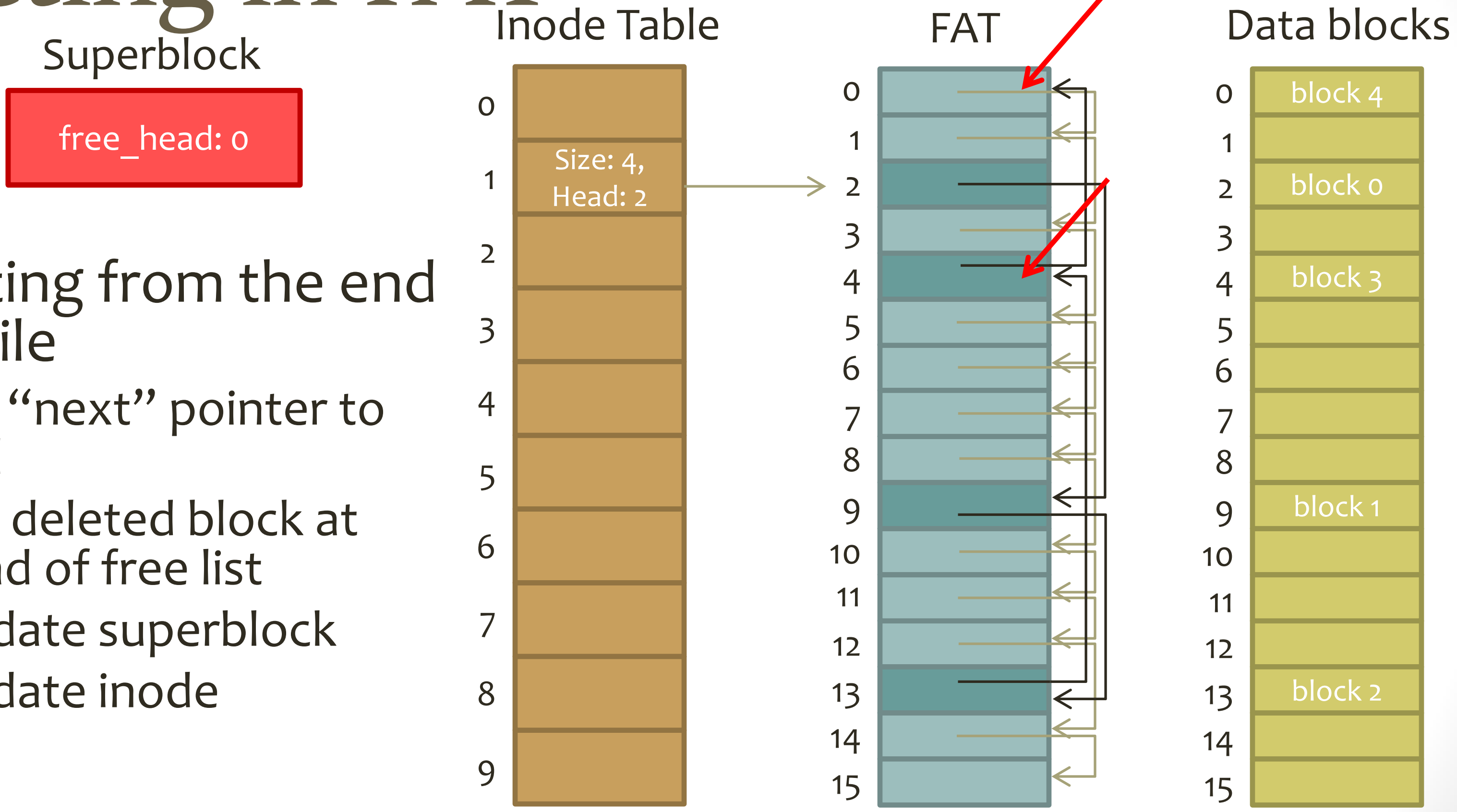
Writing in FAT

- Steps to write
  - Look up inode
  - Traverse linked list
  - Take block(s) from the head of the free list to append
  - Update superblock
  - Update inode



Deleting in FAT

- Deleting from the end of a file
  - Set "next" pointer to null
  - Put deleted block at head of free list
  - Update superblock
  - Update inode

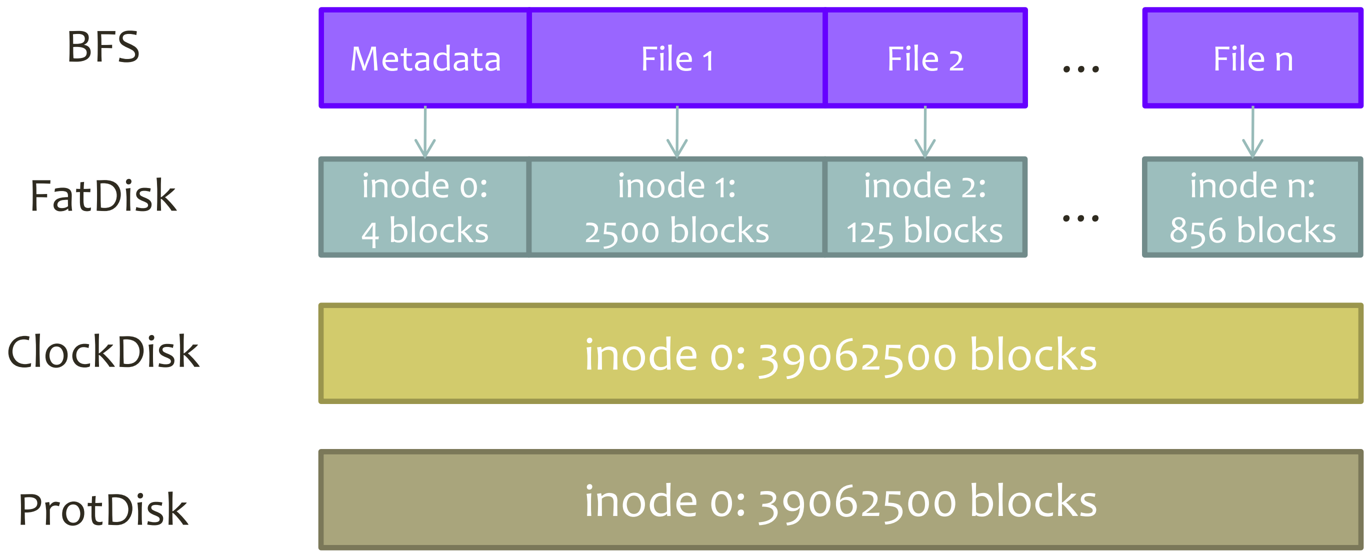


Outline

- EGOS File/Storage System
- FAT Filesystem Design
- **Project 5 Concepts**

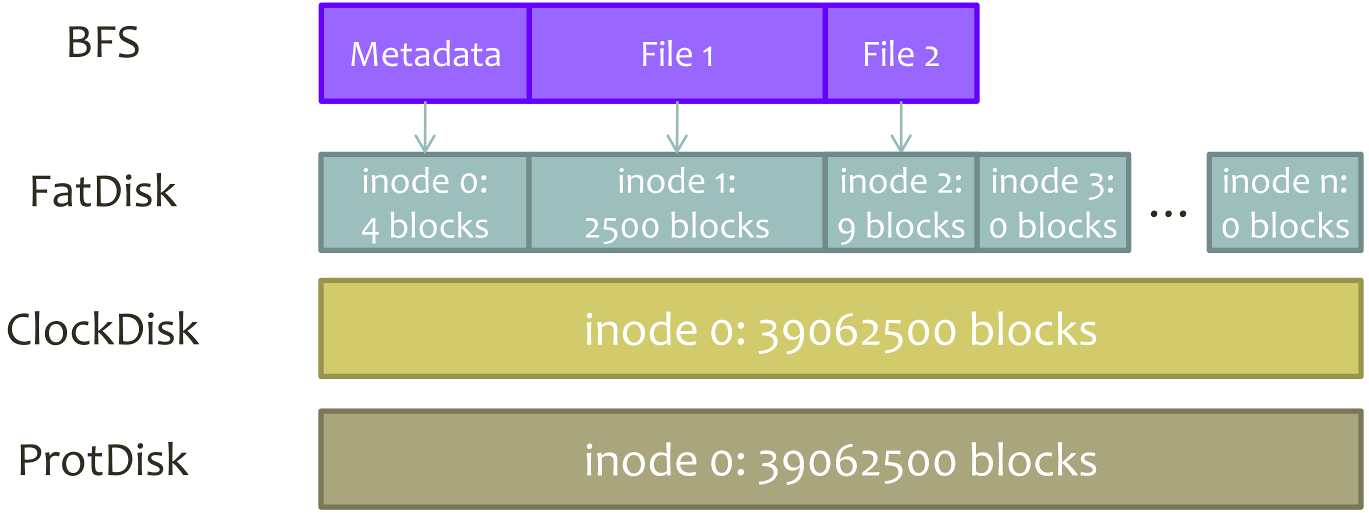
FAT as an EGOS Block Store

- Instead of "files," FatDisk has "virtual block stores"
- Replaces TreeDisk as layer between BFS and physical disk



EGOS Filesystem Concepts

- FatDisk has a fixed number of inodes/VBSs, set at creation
- All inodes "exist" with size 0 at first: BFS keeps track of which are being used by files, assigns new files to unused inodes



Functions to Implement

```
int fatdisk_create(block_if below, unsigned int
below_ino, unsigned int ninodes);
```

- Creates a new FAT filesystem consisting of `ninodes` VBSs on inode `below_ino` of block store `below`
- Expect this to be called before `fatdisk_init(below, below_ino)` with the same `below_ino`
- This may be called on a block store that already contains a FAT filesystem! (It happens during bootup.) If so, do nothing.

Functions to Implement

```
int fatdisk_read(block_if this_bs, unsigned int
ino, block_no offset, block_t *block);
int fatdisk_write(block_if this_bs, unsigned int
ino, block_no offset, block_t *block);
```

- Read or write a single block at index `offset` within inode# `ino`
- Write to an offset larger than the inode's size implies expanding it with more blocks

```
void fatdisk_free_file(struct fatdisk_snapshot
*snapshot, struct fatdisk_state *fs);
```

- Deletes an entire inode (indicated by `snapshot`)