Persistence: Files

Sridhar Alagar

File

- Sequence of persistent bytes that can be read/written
- Logical storage unit with contiguous logical address space
- A **file system** organizes and stores file
- A file has many attributes

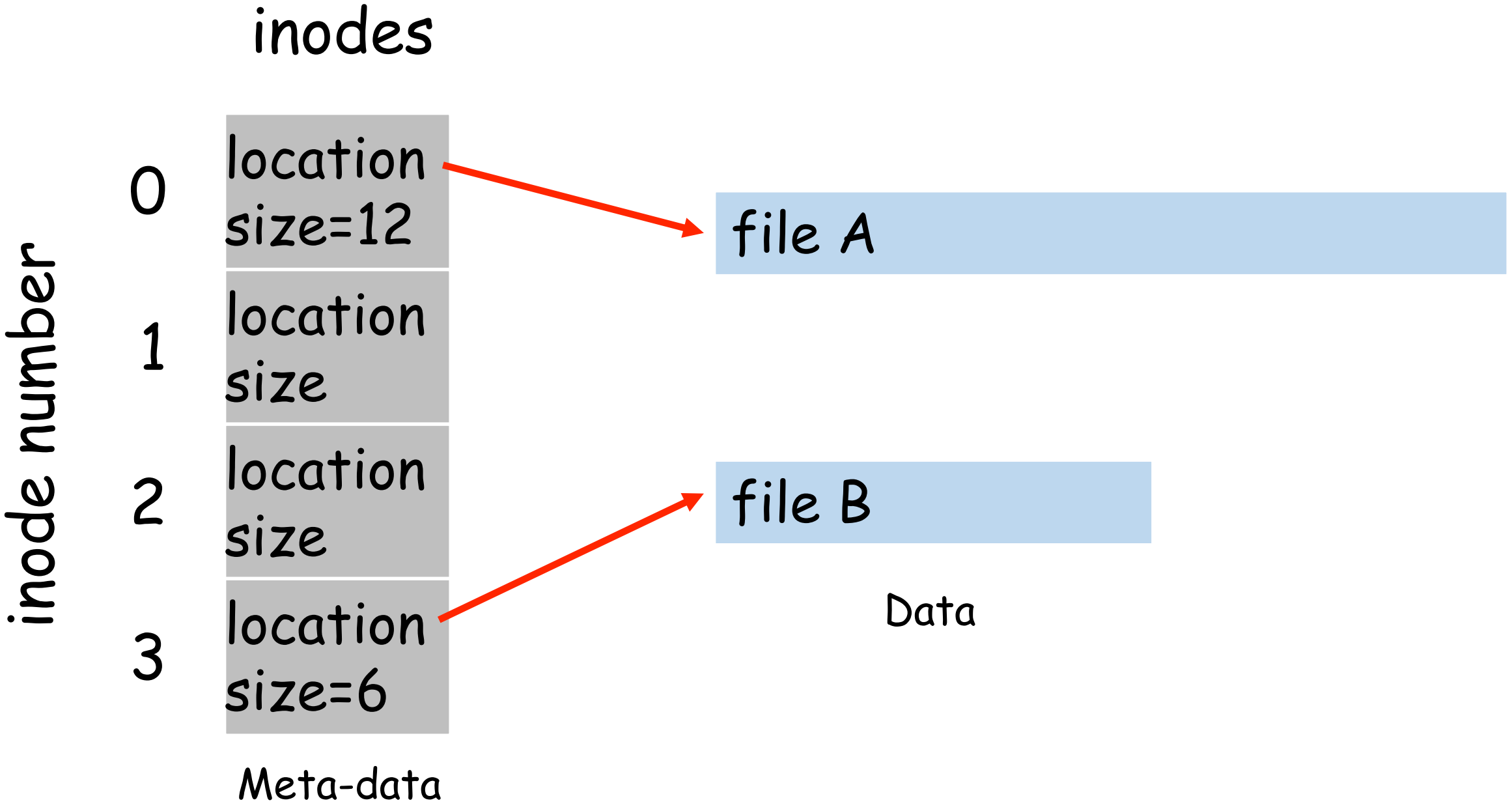
File attributes (meta data)

- Name – only information kept in human-readable form
- Identifier – unique number identifies file within file system
- Location – pointer to file location on device
- Size – current file size
- Protection – controls who can do reading, writing, executing
- Time, date, and user identification – data for protection, security, and usage monitoring

Meta data should be persistent

Inode

- Inode is an **on-disk** data structure that contains a file's attributes
- Every file has a unique inode
- Inodes are kept in inode table
- Inode number is the file identification number within the file system


inodes
inode number
0
1
2
3
location
size=12
location
size
location
size
location
size=6
file A
file B
Data
Meta-data

How to access a file?

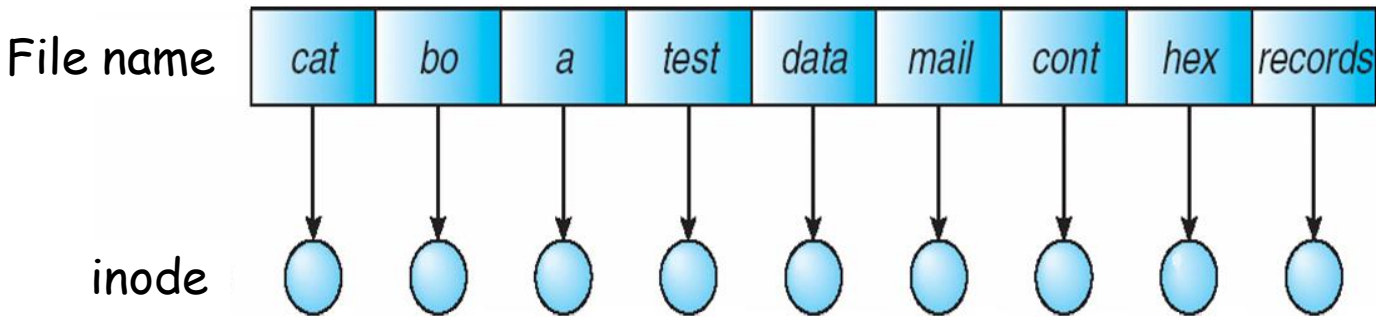
- Find its inode
- Need mapping between file name and file's inode number
- Where will this mapping be stored?

Directory

Try ls -i

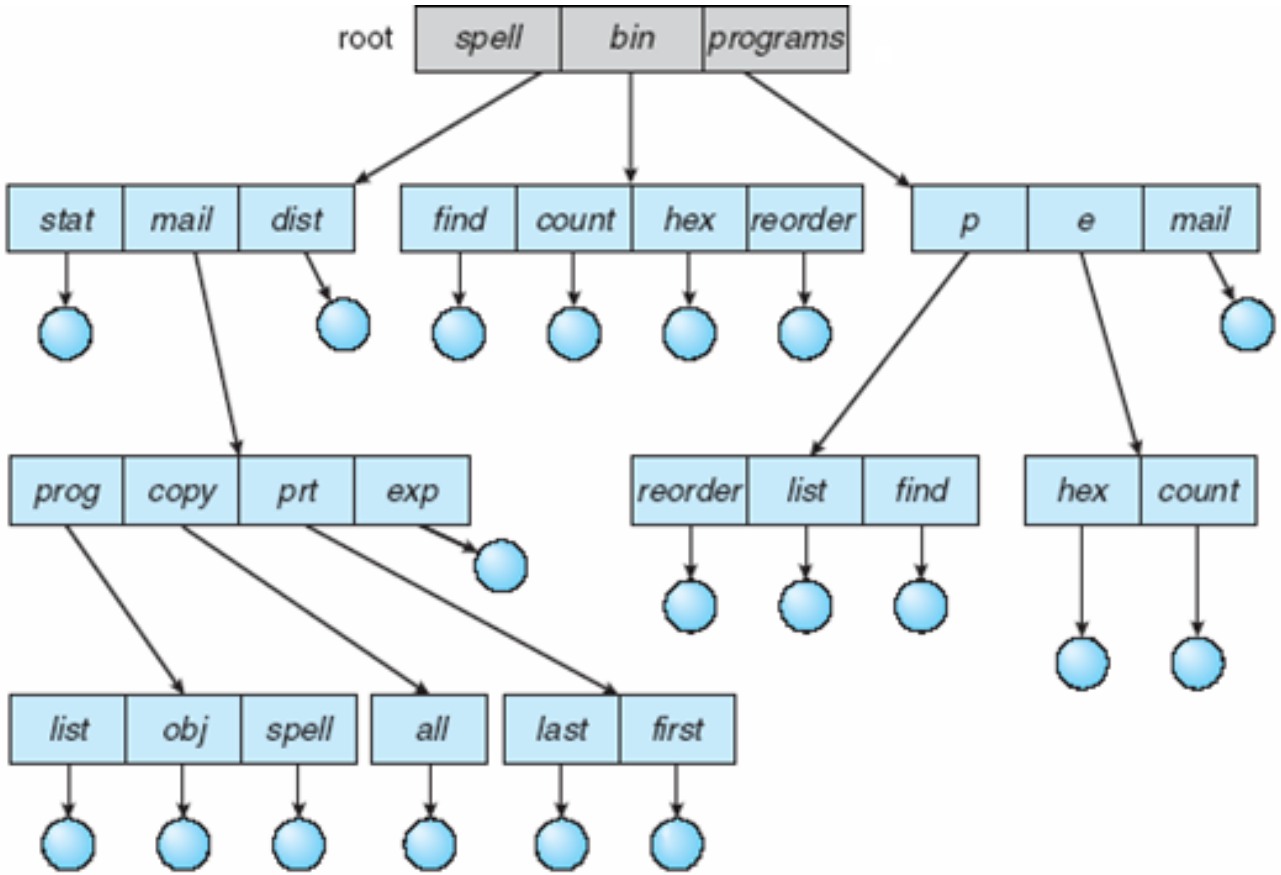
Directory Structure

- Single level



- Naming problem
- Grouping problem

Tree Structured Directories



- File name is specified with the path

  /bin/count

  /spell/mail/prog/list

- File (path) name is unique

Where is a directory stored?

- In the persistent store as a file
- Every entry in the directory is a file or a (sub)directory

File API (Attempt 1)

```
read(char* pathname, void *buf, int offset, size_t nbyte)
write(char* pathname, void *buf, int offset, size_t nbyte)
```

- Disadvantages?
  - Need to find the file every time read/write performed
    - traverse the directory tree, get the inode and then the file
  - User need to remember the current offset
- Solution
  - Open it once and maintain the state
    - in a file descriptor table

File Descriptor Table (per process)

- Is an array of file descriptors, maintained in proc structure

- xv6 definition of file descriptor, and proc structure:

```
struct file { // file descriptor
  ...
  struct inode *ip;
  uint off;
};
// Per-process state
struct proc {
  ...
  struct file *ofile[NOFILE];  // Open files
  ...
}
```

Open once

fd = open("file.txt", flags)

- Searches the directory once
- Fetch the inode and keep it in memory
- Find a free entry in file table ofile[]
- Update ofile[fd] to point to inode
- Set the offset value in ofile[fd]
- Subsequent file operations use the fd

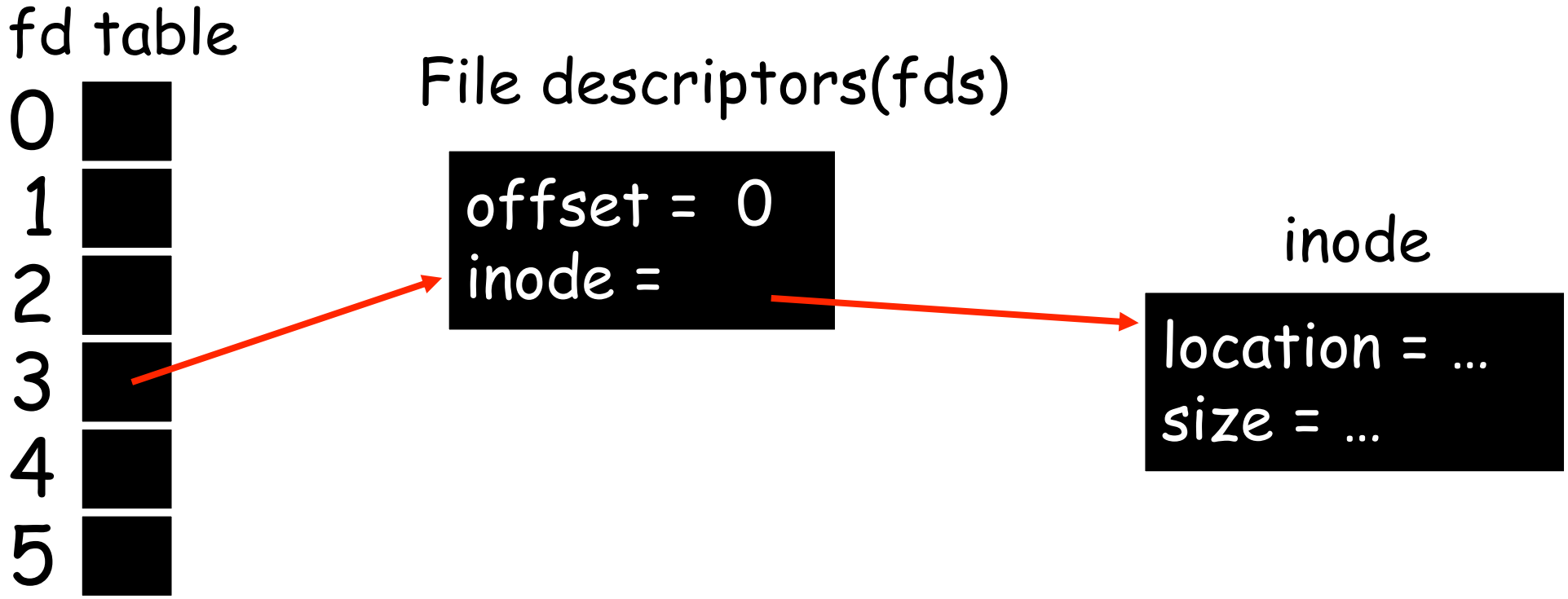
File API

```
int open(char* path_name, int flags)
int read(int fd, void *buf, size_t nbyte)
int write(int fd, void *buf, size_t nbyte)
close(int fd);
```

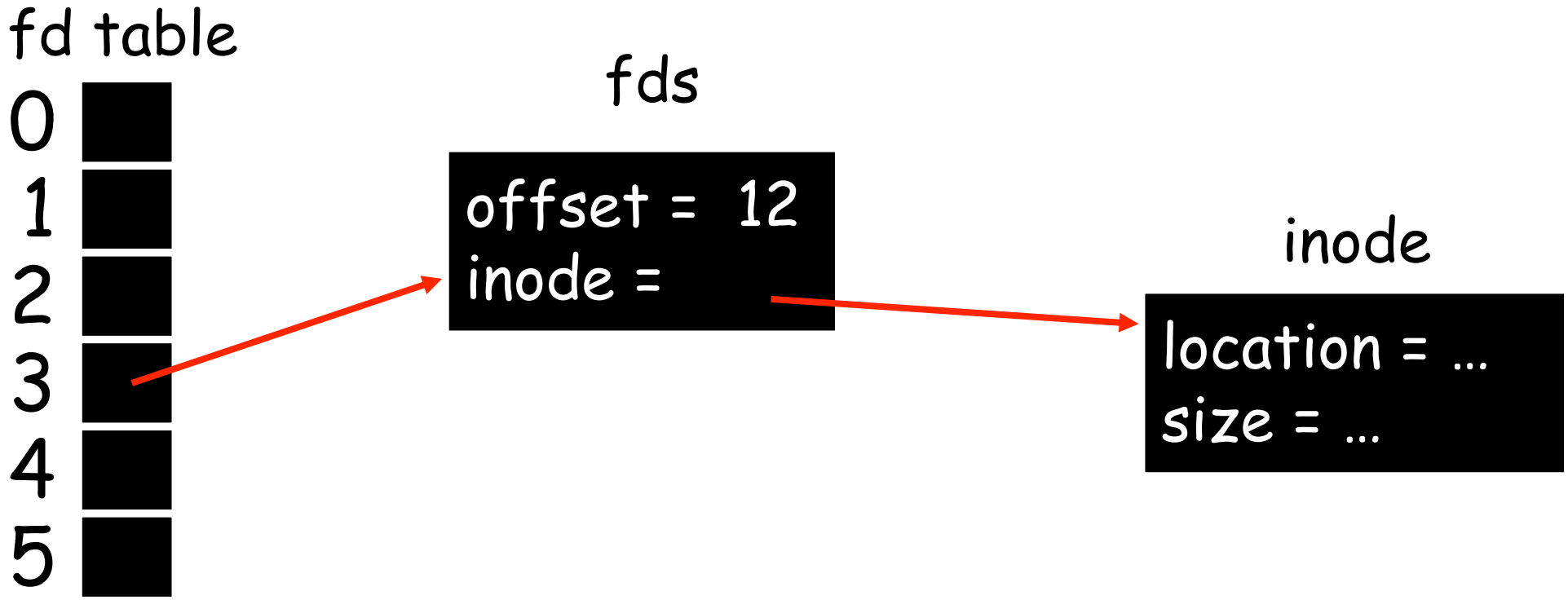
- fd is the file descriptor returned by open()
- No need to search for the file
- Read/write performed starting from the current offset

Sample Code

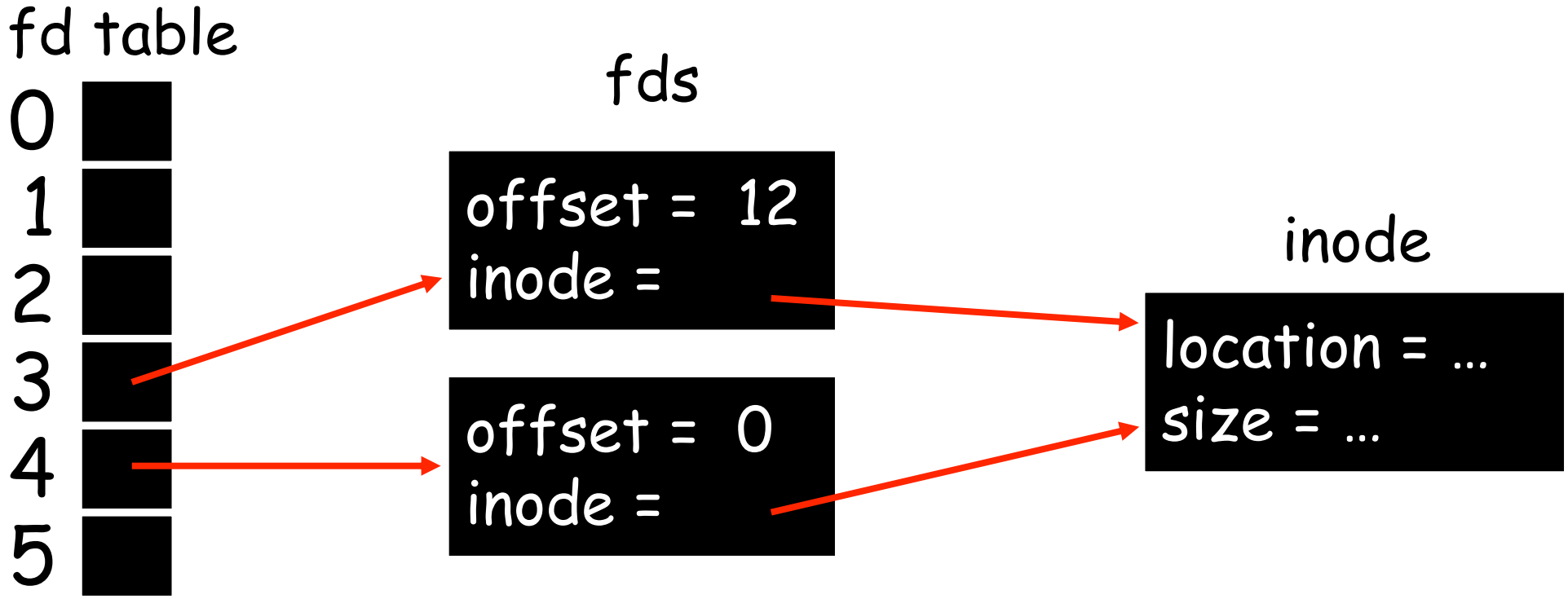
```
int fd1 = open("file.txt"); // returns 3
read(fd1, buf, 12);
int fd2 = open("file.txt"); // returns 4
int fd3 = dup(fd2);         // returns 5
Close (fd2);
```



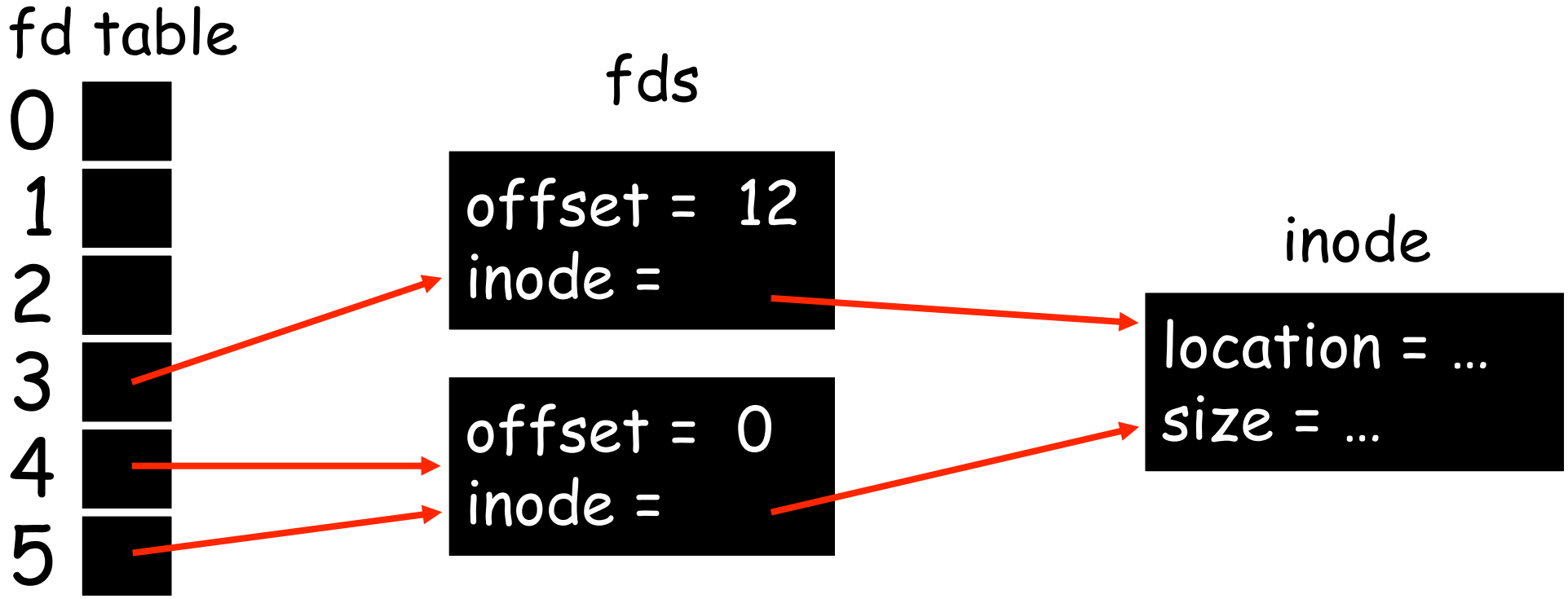

```
int fd1 = open("file.txt"); // returns 3
```



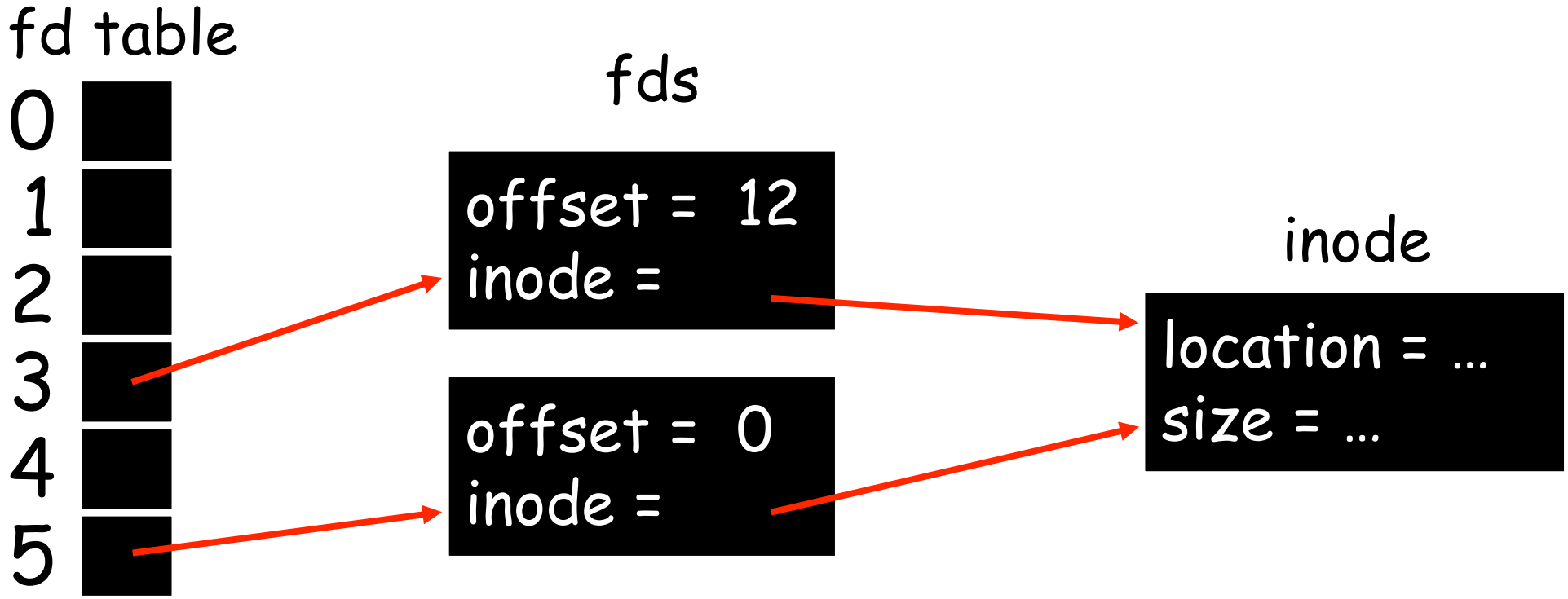

```
fd1 = open("file.txt"); // returns 3
read(fd1, buf, 12);
```



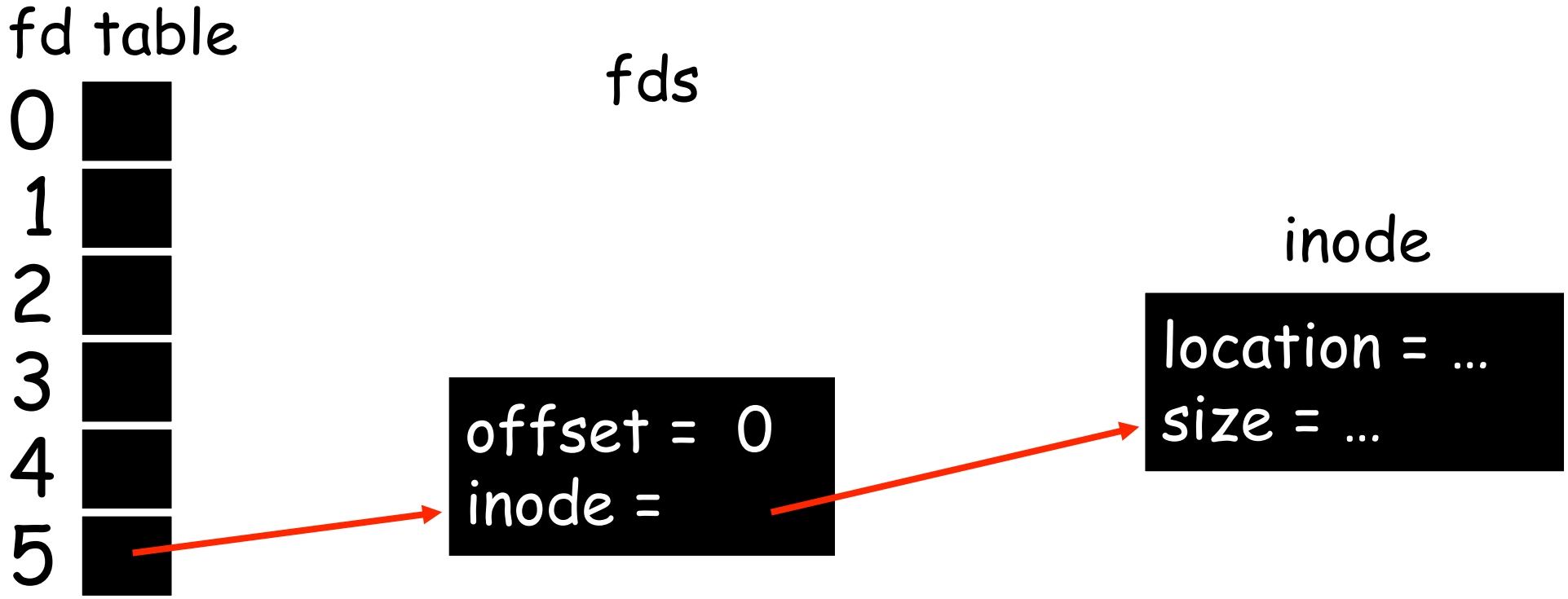

```
fd1 = open("file.txt"); // returns 3
read(fd1, buf, 12);
fd2 = open("file.txt"); // returns 4
```




```
fd1 = open("file.txt"); // returns 3
read(fd1, buf, 12);
fd2 = open("file.txt"); // returns 4
fd3 = dup(fd2);         // returns 5
```




```
fd1 = open("file.txt"); // returns 3
read(fd1, buf, 12);
fd2 = open("file.txt"); // returns 4
fd3 = dup(fd2);         // returns 5
close(fd2);
```



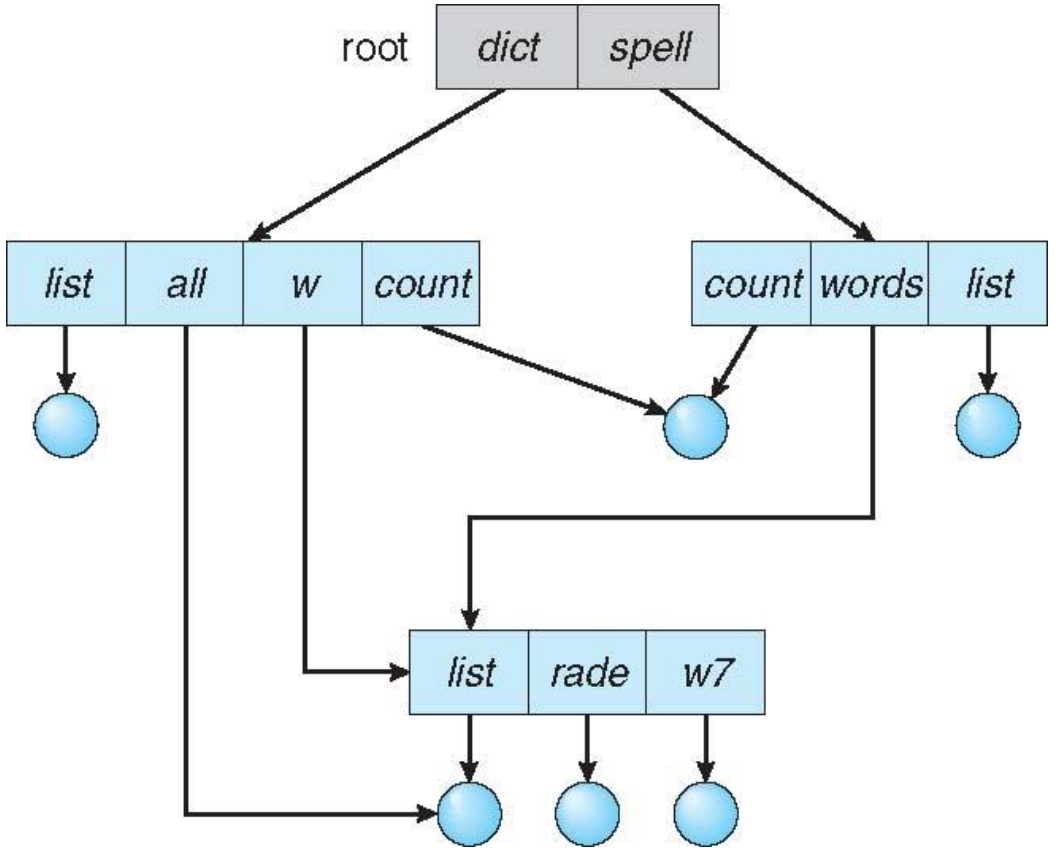

```
fd1 = open("file.txt"); // returns 3
read(fd1, buf, 12);
fd2 = open("file.txt"); // returns 4
fd3 = dup(fd2);         // returns 5
close(fd2);
close(fd1);
```

Links to Files

```
link(char* oldpath, char* newpath)
unlink(char* pathname)
```

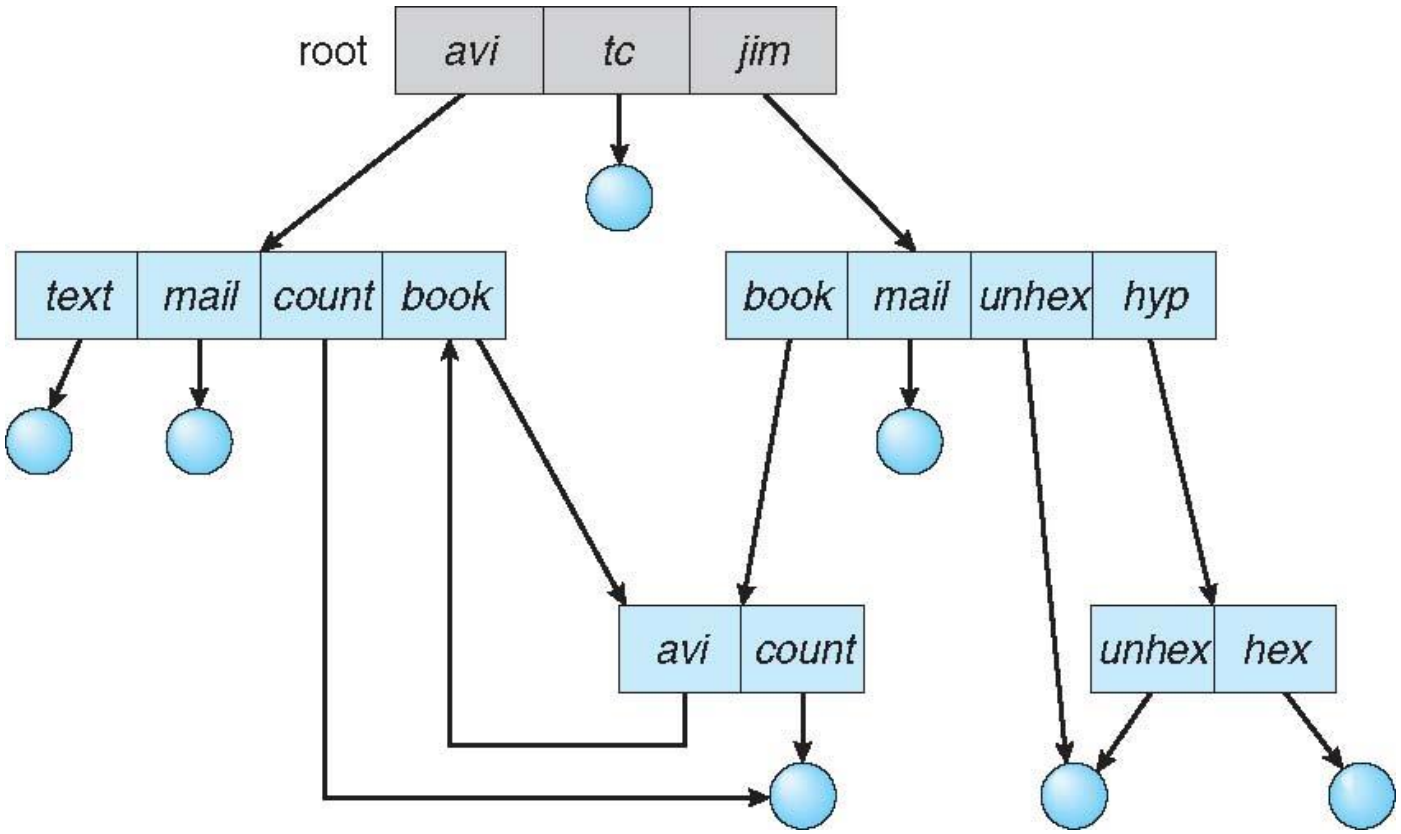
- link() creates a new (hard) link to existing file
  - inode not created
  - reference count increases
  - cannot link directories
- Unlink() removes link
  - If it is the last link to the file, deletes the file
- What is the directory structure now?

Acyclic Graph Directories



What is the directory structure when a symbolic link to a parent directory is created?

Directory structure – Graph (with cycles)



Files - More APIs

lseek(fd, offset, whence)

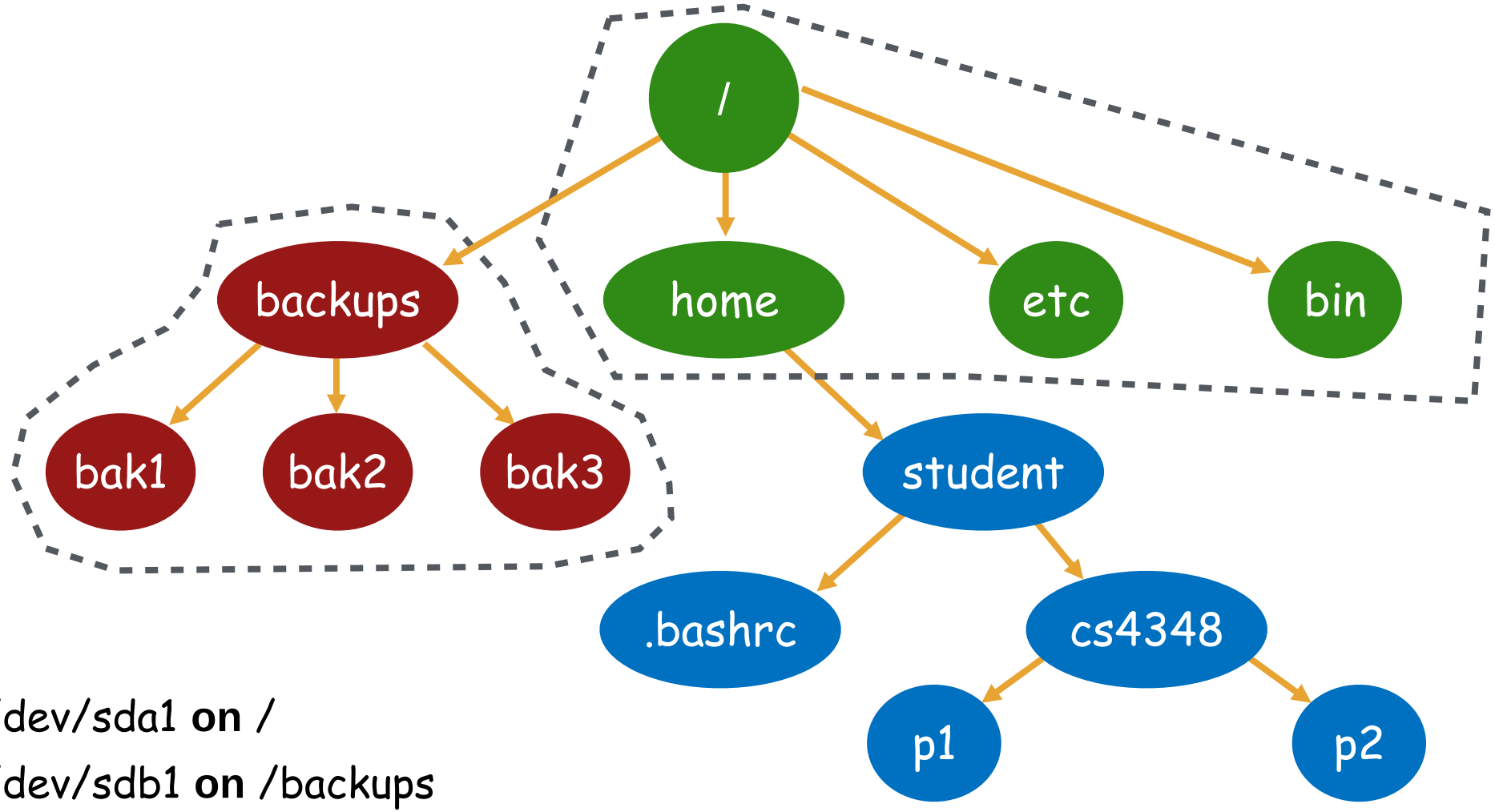
rename(oldname, newname)

stat(pathname, buf)

Fsync()

File System Tools

- mkfs: creates a new file system in a storage device
- mount: mounts another device with a file system onto an existing directory structure



- /dev/sda1 **on** /
- /dev/sdb1 **on** /backups
- hopper **on** /home