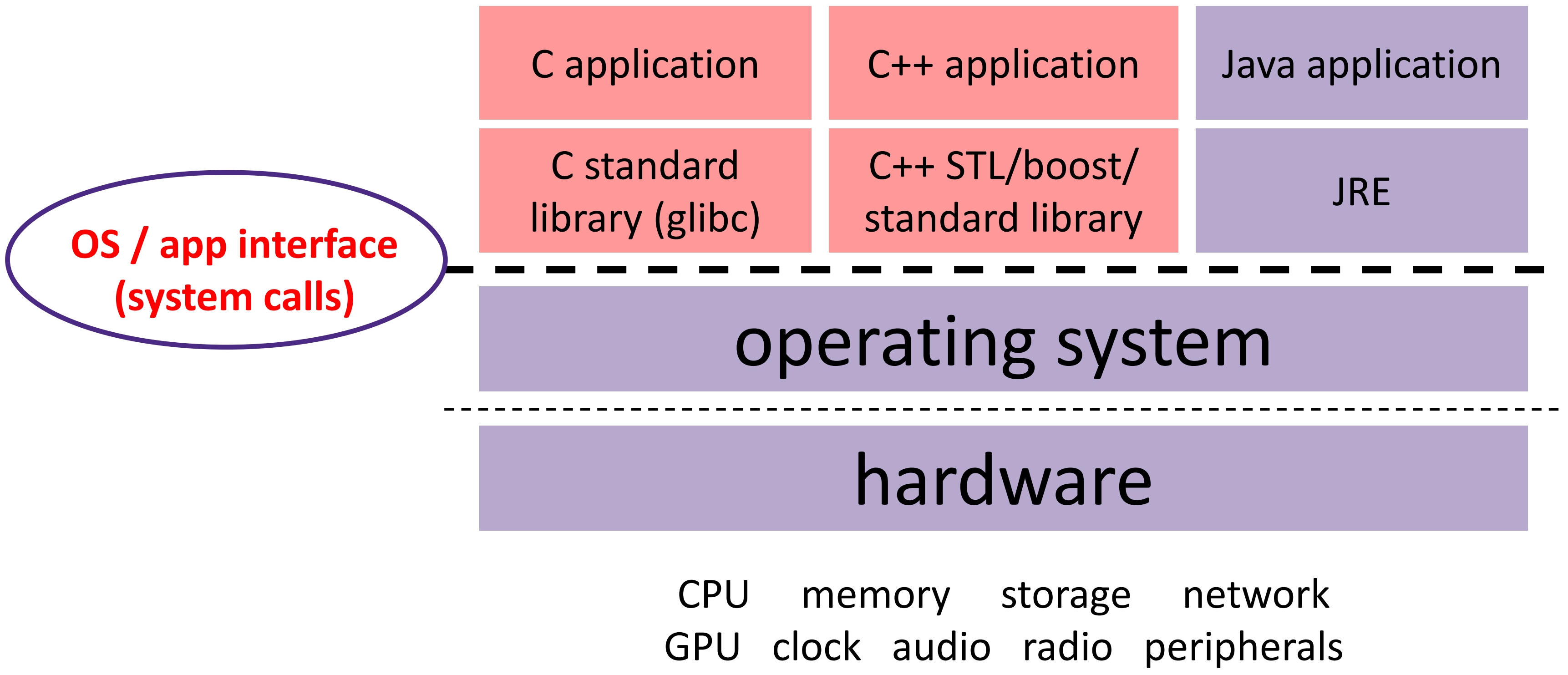
CS/SE 3377

FILE API

Sridhar Alagar

System Calls


C application
C++ application
Java application
C standard library (glibc)
C++ STL/boost/ standard library
JRE
OS / app interface (system calls)
operating system
hardware
CPU memory storage network
GPU clock audio radio peripherals

From C to POSIX

- Most Linux versions support a common set of lower-level file access APIs: (conforming to POSIX – Portable Operating System Interface)
  - `open()`, `read()`, `write()`, `close()`, `lseek()`
    - Lower-level
    - Also, less convenient

open()/close()

- To open a file:
  - Pass in the filename and access mode
  - Get back a "file descriptor"
    - is just an `int;`
    - Lowest numbered file descriptor available
    - Defaults: **0** is `stdin`, **1** is `stdout`, **2** is `stderr`

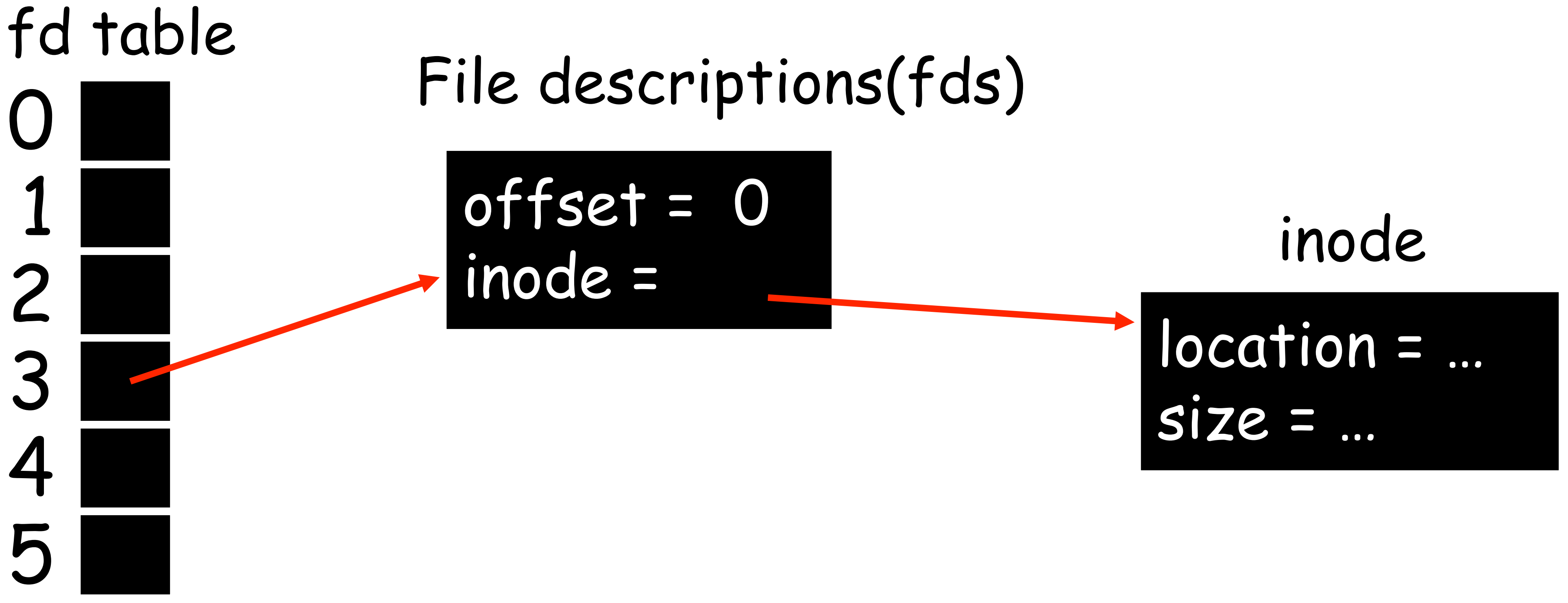
```
#include <fcntl.h>     // for open()
#include <unistd.h>    // for close()
  ...
  int fd = open("foo.txt", O_RDONLY);
  if (fd == -1) {
    perror("open failed");
    exit(EXIT_FAILURE);
  }
  ...
  close(fd);
```

Reading from a file

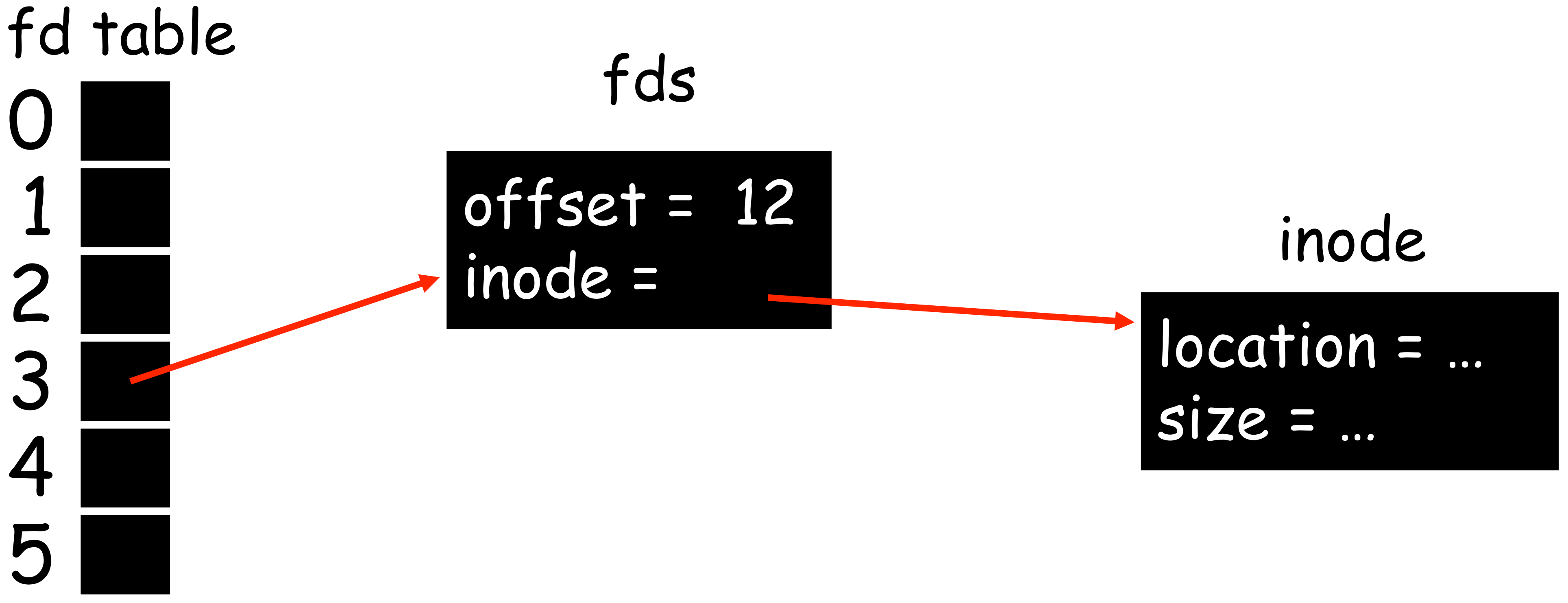
- `ssize_t read(int fd, void* buf, size_t count);`
  - Returns the number of bytes read
    - Might be fewer bytes than you requested (**!!!**)
    - Returns **0** if you're already at the end-of-file
    - Returns **-1** on error (and sets `errno`)
  - There are some surprising error modes (check `errno`)
    - `EBADF`: bad file descriptor
    - `EFAULT`: output buffer is not a valid address
    - `EINTR`: read was interrupted, please try again (ARGH!!!!)
    - And many others…

Sample Code

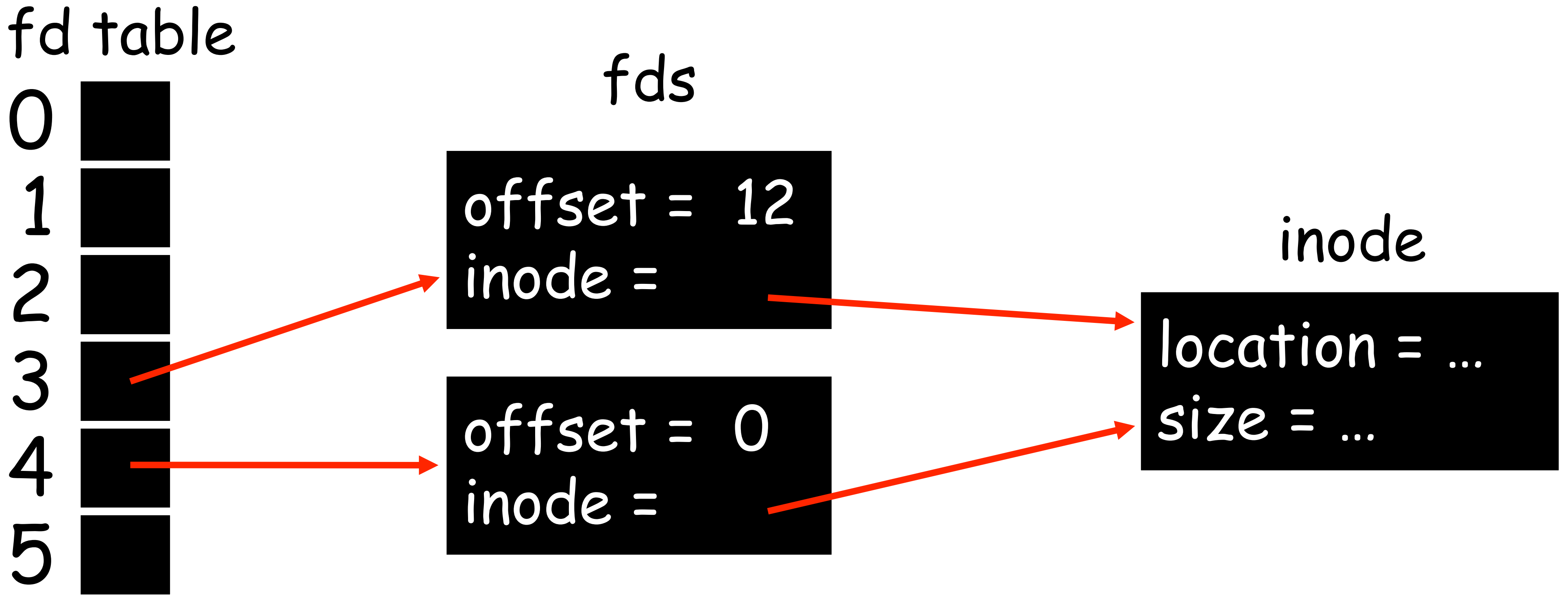
```
int fd1 = open(“file.txt”); // returns ?
read(fd1, buf, 12);
int fd2 = open(“file.txt”); // returns ?
int fd3 = dup(fd2);         // returns ?
Close (fd2);
```



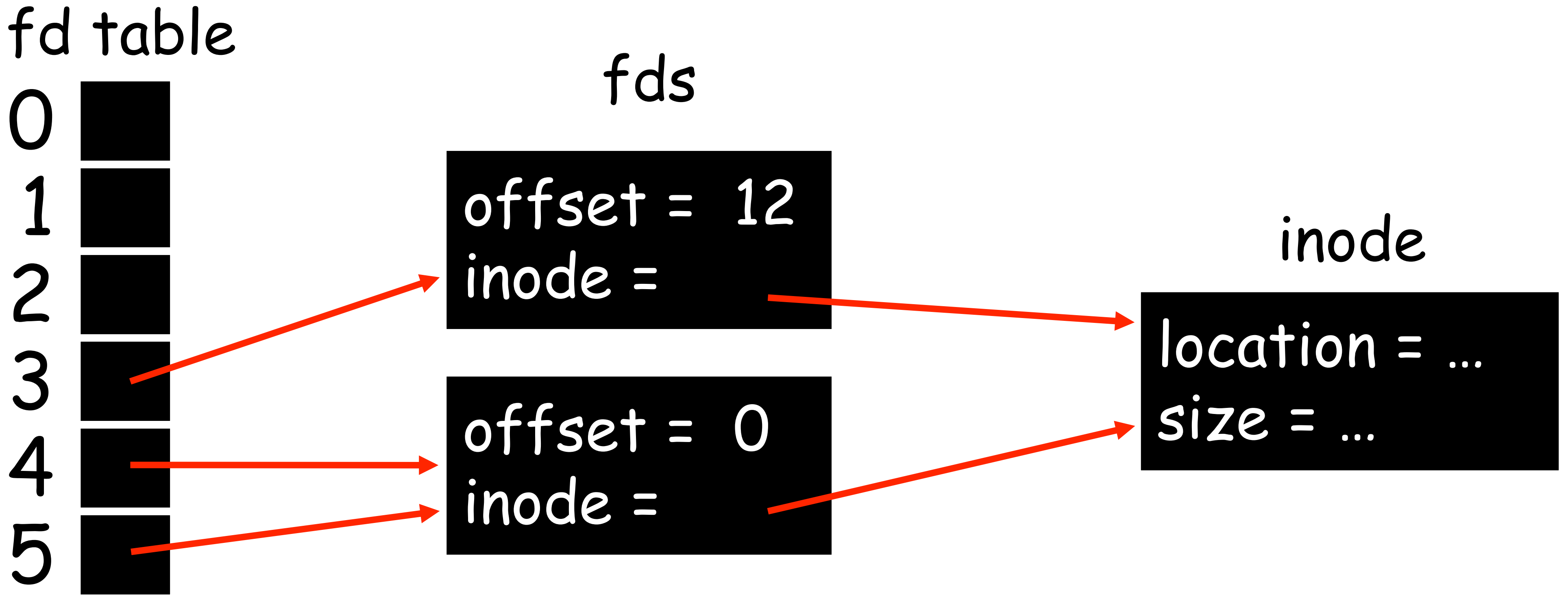
```
int fd1 = open("file.txt"); // returns 3
```



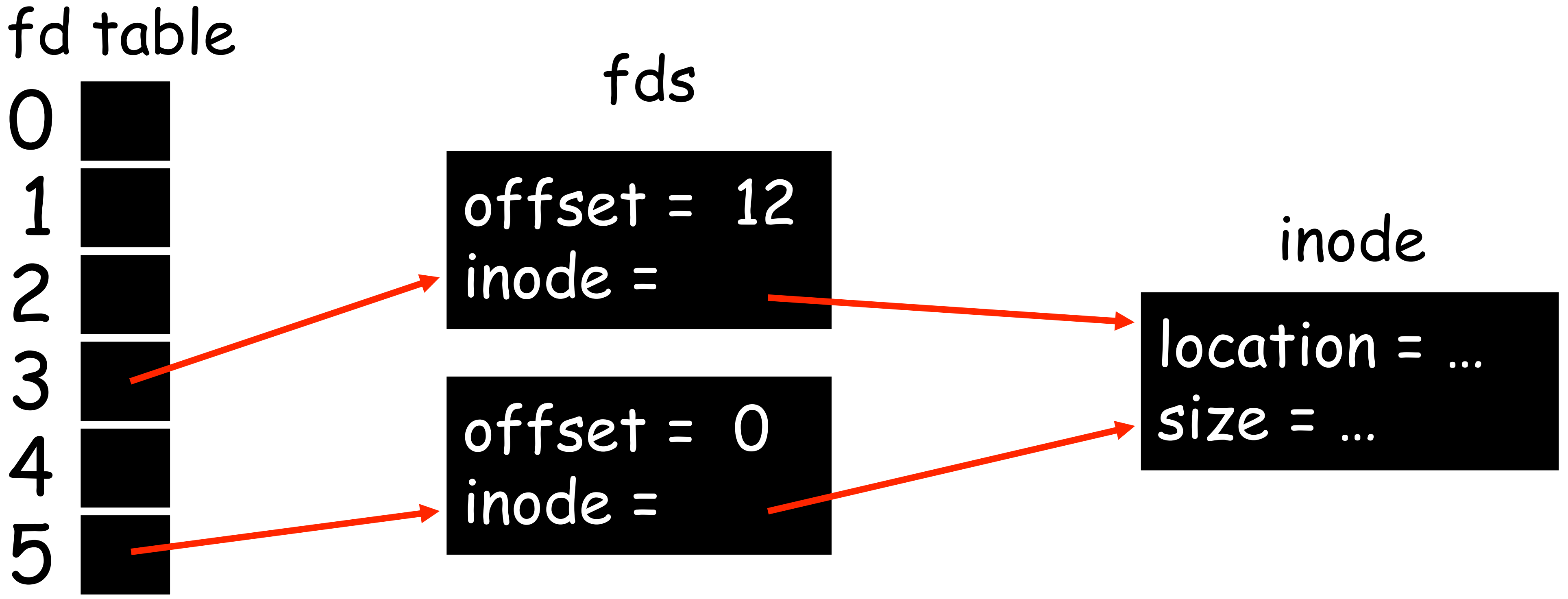
```
int fd1 = open("file.txt"); // returns 3
read(fd1, buf, 12);
```



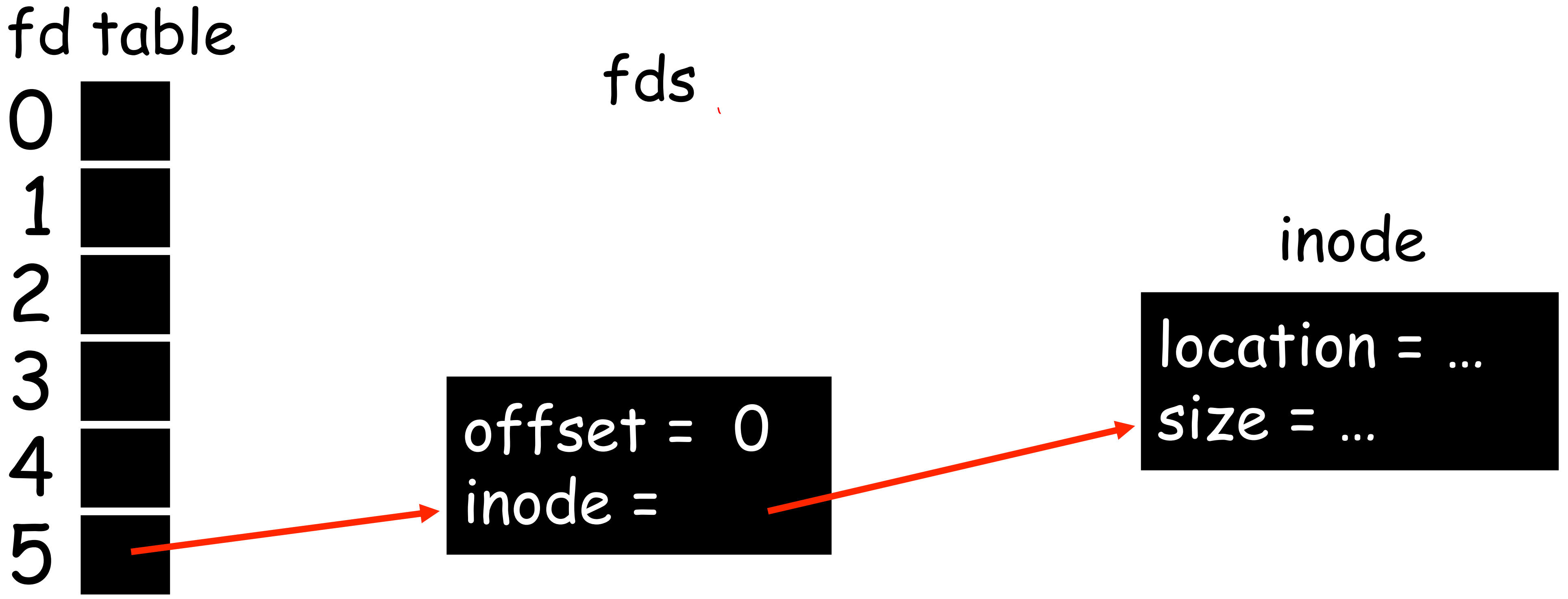
```
fd1 = open("file.txt"); // returns 3
read(fd1, buf, 12);
fd2 = open("file.txt"); // returns 4
```



```
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);
```

Other Low-Level Functions

- Read man pages to learn about:
  - `write()` – write data
    - `#include <unistd.h>`
  - `lseek()` – reposition offset
    - `#include <unistd.h>`
  - `fsync()` – flush data to the underlying device
    - `#include <unistd.h>`
  - `opendir()`, `readdir()`, `closedir()` – deal with directory listings
    - `#include <dirent.h>`

- A useful shortcut sheet (from CMU):
  http://www.cs.cmu.edu/~guna/15-123S11/Lectures/Lecture24.pdf

One way to read() n bytes

```
int fd = open(filename, O_RDONLY);
char* buf = ...;  // buffer of appropriate size
int bytes_left = n;
int result;

while (bytes_left > 0) {
  result = read(fd, buf + (n - bytes_left), bytes_left);
  if (result == -1) {
    if (errno != EINTR) {
      // a real error happened, so return an error result
    }
    // EINTR happened, so do nothing and try again
    continue;
  } else if (result == 0) {
    // EOF reached, so stop reading
    break;
  }
  bytes_left -= result;
}

close(fd);
```

Disclaimer

Some of the materials in this lecture slides are from

- the lecture slides of CS333 Univ of Washington
- the materials prepared by Prof. Arpaci, and Prof. Youjip