EEL-4736/5737 Principles of Computer System Design

Lecture Slides 5

Textbook Chapter 2

Case Study: UNIX File System Layering and Naming

Introduction

- File system:
  - Builds upon the file abstraction
  - Creates a system to organize and control access to files
  - Hierarchical organization, with support for links
- Objects: files and directories
  - Files hold data
  - Directories hold other objects
- Meta-data
  - Permissions (r/w/x), owner, times

Application Programming Interface

- System calls exposed by O/S
- fd = OPEN(name, flag, mode)
  - Flag: e.g. CREATE
  - Mode: r/w/x
  - Pointer/cursor=0 (beginning of file)
  - Return descriptor (or error code)
- READ/WRITE(fd, buffer, n)
- SEEK(fd,offset,whence)
  - Set cursor to offset relative to begin/end/current
- CLOSE(fd)

Application Programming Interface

- MKDIR(name), RMDIR(name)
  - Create/remove directory
- CHDIR(name)
  - Change working directory
- CHROOT(name)
  - Change default root directory
- MOUNT(name, device)
  - Associate file system on device with name
- STAT(name)
  - Return meta-data

Naming layers

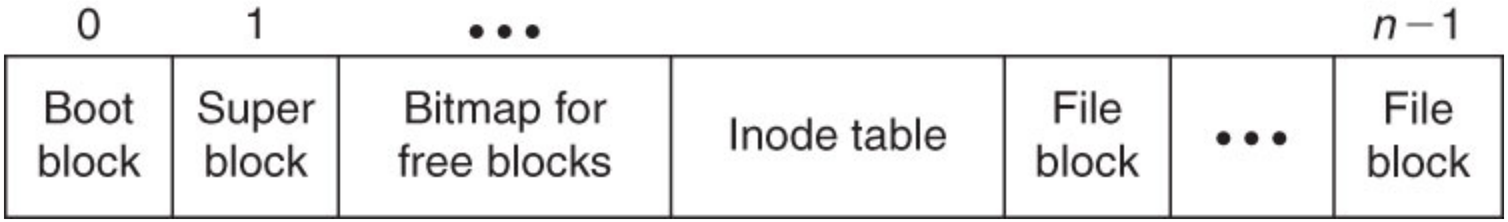
Layer	Purpose	
Symbolic link	Integrate multiple file systems	User-oriented names
Absolute path name layer	Provide a root for naming hierarchies	
Path name layer	Organize files into naming hierarchies	
File name layer	Human-oriented names	User/machine Interface
Inode number layer	Machine-oriented names	Machine-oriented names
File layer	Organize blocks into files	
Block layer	Identify disk blocks	

The Block Layer

- Sector – minimum cell of information addressable in a disk
- Block – small number of disk sectors
  - E.g. 8 Kbytes
- Names are block numbers of a disk device
  - E.g. offset from block 0
  - Note: device may have its own layer mapping of block to physical location
    - E.g. hard disks: bad sectors

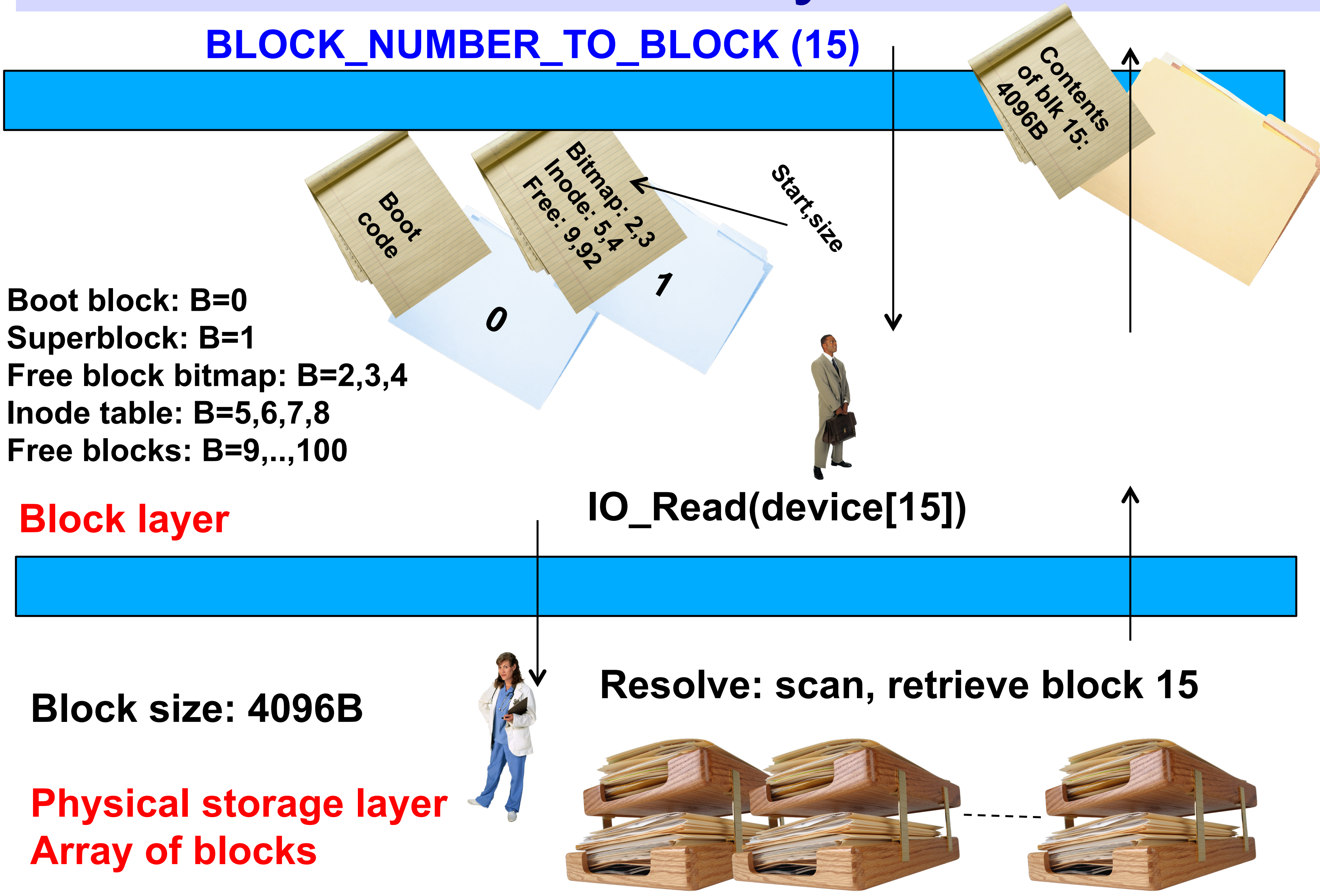
The block layer

- How to discover which blocks are in use?
  - – Super-block – well-known name (1)
    - Size of file system, bitmap, inode table
  - – Bitmap of free blocks
    - Bit i determines if block i is free or not
    - Well-known name – e.g. after the super-block
    - Boot block also a well-known name (0)



0	1	• • •				*n* − 1
Boot block	Super block	Bitmap for free blocks	Inode table	File block	• • •	File block

The Block Layer



BLOCK_NUMBER_TO_BLOCK (15)

Boot block: B=0
Superblock: B=1
Free block bitmap: B=2,3,4
Inode table: B=5,6,7,8
Free blocks: B=9,...,100

Block layer

IO_Read(device[15])

Block size: 4096B

Resolve: scan, retrieve block 15

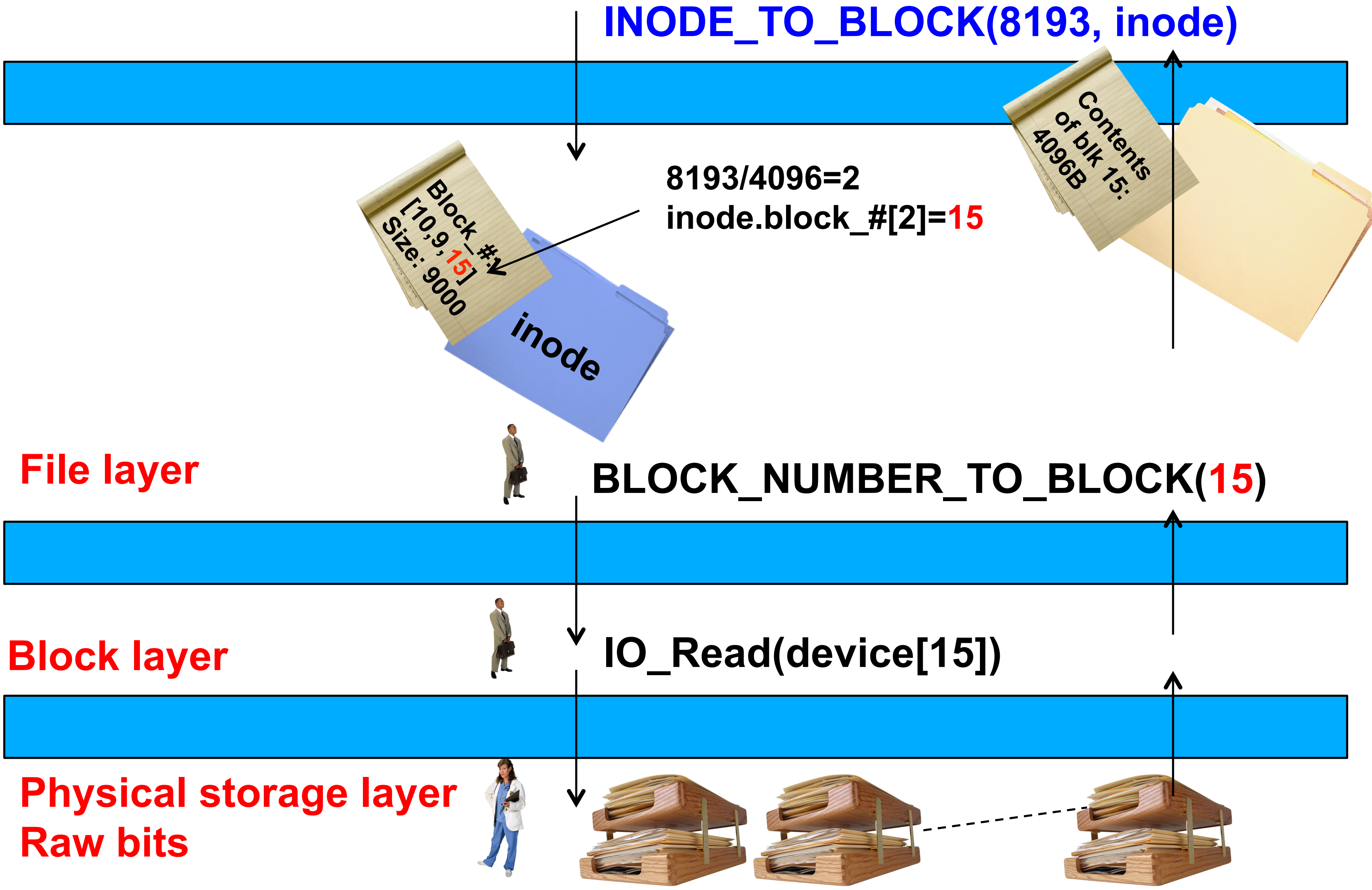
Physical storage layer
Array of blocks

The File Layer

- Objects have 1+ blocks and can grow or shrink over time
- Next naming layer – files
  - File: linear array, arbitrary length
- Which blocks belong to file?
  - Information in index node (*inode)* object

```
structure inode
    integer block_numbers[N] // which blocks in file
    integer size             // file size
```

The File Layer



INODE_TO_BLOCK(8193, inode)

8193/4096=2
inode.block_#[2]=15

File layer

BLOCK_NUMBER_TO_BLOCK(15)

Block layer

IO_Read(device[15])

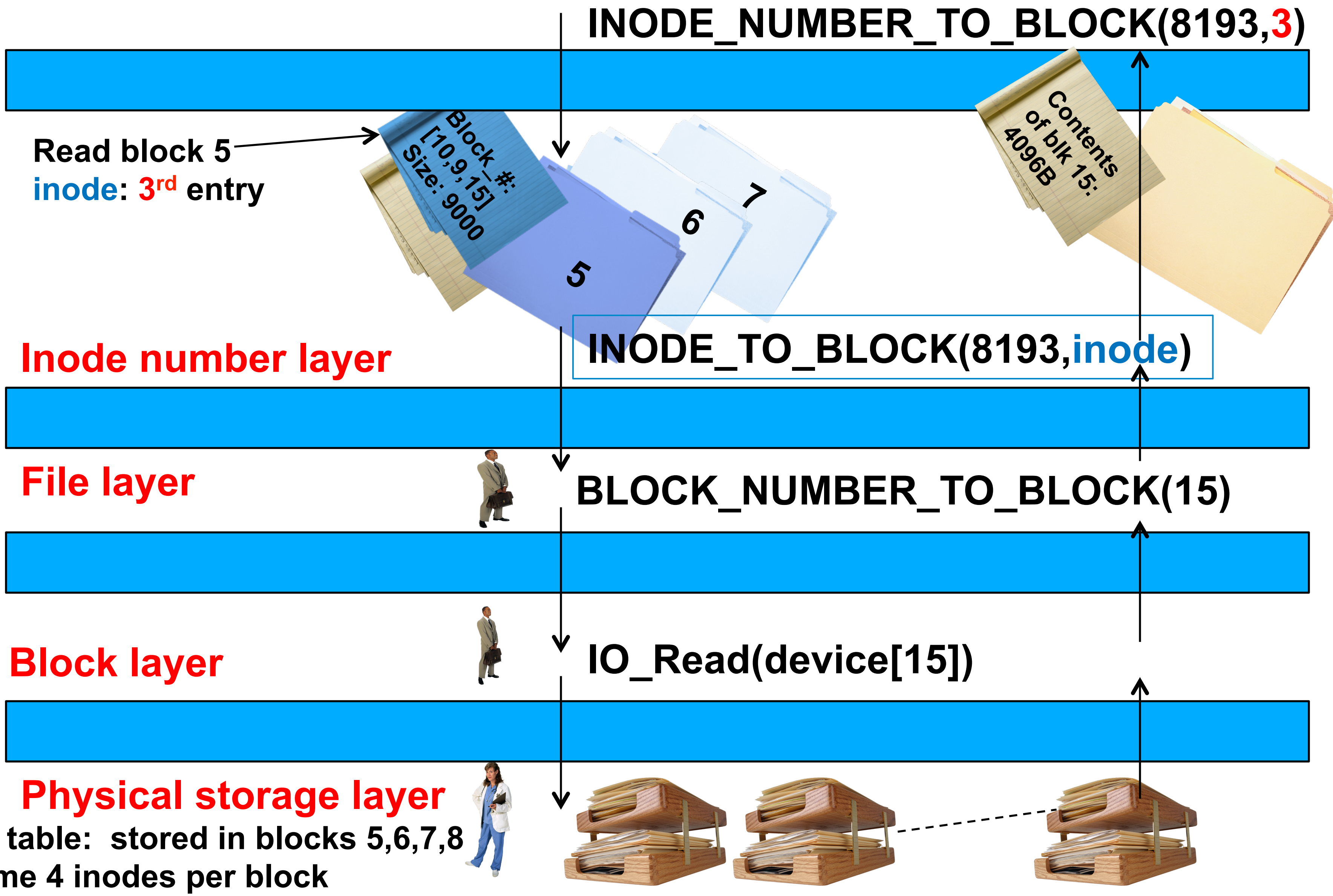
Physical storage layer
Raw bits

The Inode Number Layer

- Instead of passing inodes themselves, inodes are named
  - Table indexed by inode number
  - Putting it all together – returning the block which contains byte *offset* of file with *inode_number*

```
procedure INODE_NUMBER_TO_BLOCK (int offset, int
    inode_number) returns block
    inode instance i <- INODE_NUMBER_TO_INODE
    (inode_number)
    o <- offset / BLOCKSIZE
    b <- INDEX_TO_BLOCK_NUMBER (i, o)
    return BLOCK_NUMBER_TO_BLOCK (b)
```

The inode Number Layer



INODE_NUMBER_TO_BLOCK(8193,3)

Read block 5
inode: 3rd entry

Inode number layer

INODE_TO_BLOCK(8193,inode)

File layer

BLOCK_NUMBER_TO_BLOCK(15)

Block layer

IO_Read(device[15])

Physical storage layer

inode table: stored in blocks 5,6,7,8
Assume 4 inodes per block

Resolving lower layers

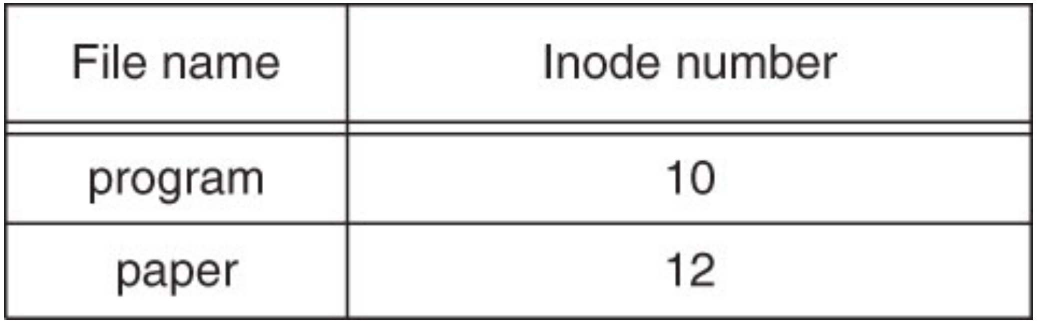
```
procedure INODE_NUMBER_TO_INODE
   (integer inode_number) returns inode
   return inode_table[inode_number]

procedure INDEX_TO_BLOCK_NUMBER
   (inode instance i, integer index) returns
   integer
   return i.block_numbers[index]

procedure BLOCK_NUMBER_TO_BLOCK
   (integer b) returns block
   return device[b]
```

The File Name Layer

- inode numbers clearly not convenient to users
  - Another problem: specify a fixed location
- Naming layer introduced to *hide the meta-data of file management*
- Directory
  - A *context* containing set of *bindings* between *character-string names* and *inode numbers*
  - Represented, itself, as a file
    - Extend inode to also have a “type”: regular file, or directory, (device, …)

Directory example

- Each file name associated with an inode number
- Directory – context; represented as file



File name	Inode number
program	10
paper	12

A directory with two files

Directory has inode (e.g. inode 4)

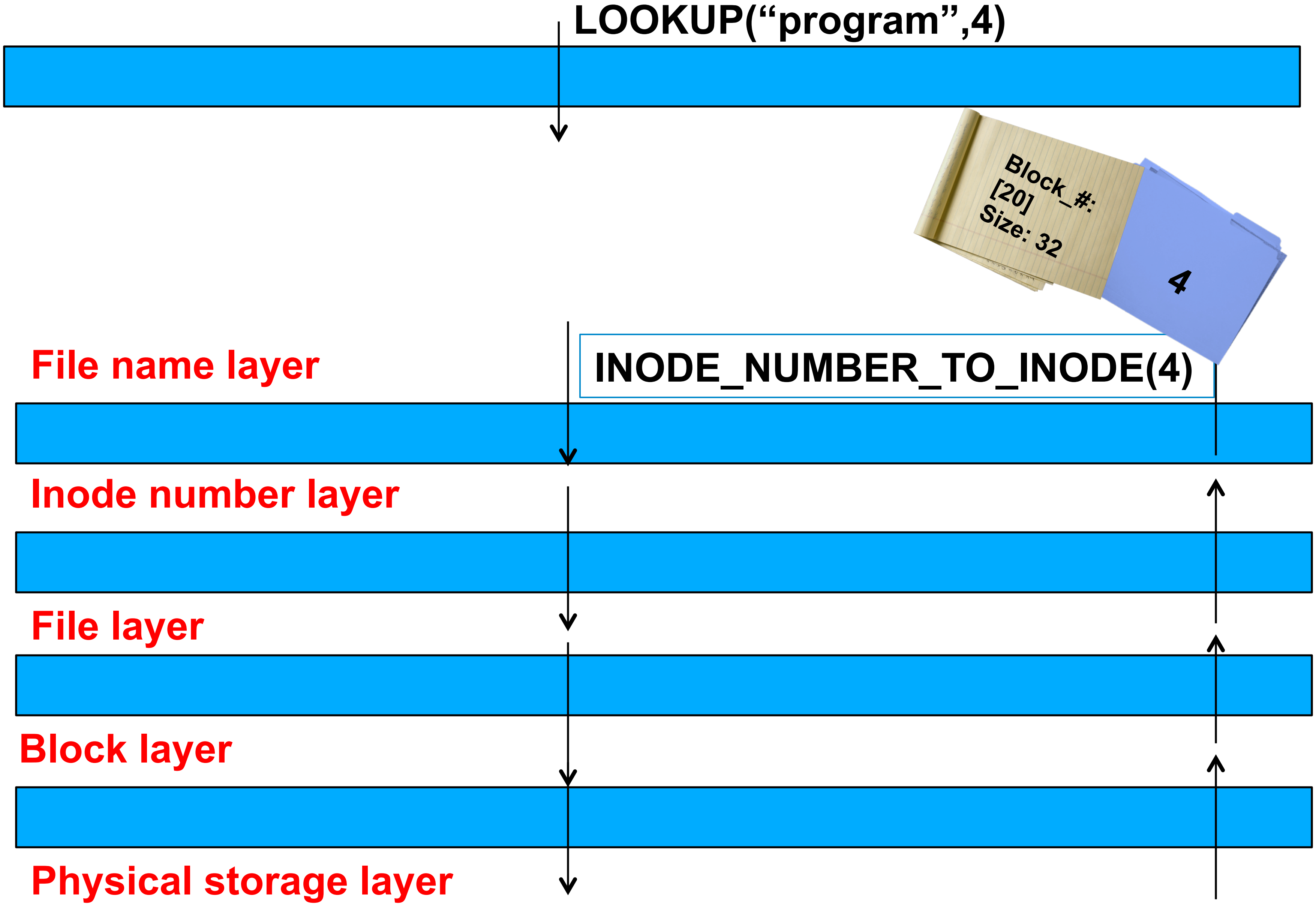
Blocks in inode list store table with context instead of file data

E.g. block #20:

program□□□□□□□10paper□□□□□□□□□12

14B **2B** **14B** **2B**

The File Name Layer



LOOKUP("program",4)

File name layer

INODE_NUMBER_TO_INODE(4)

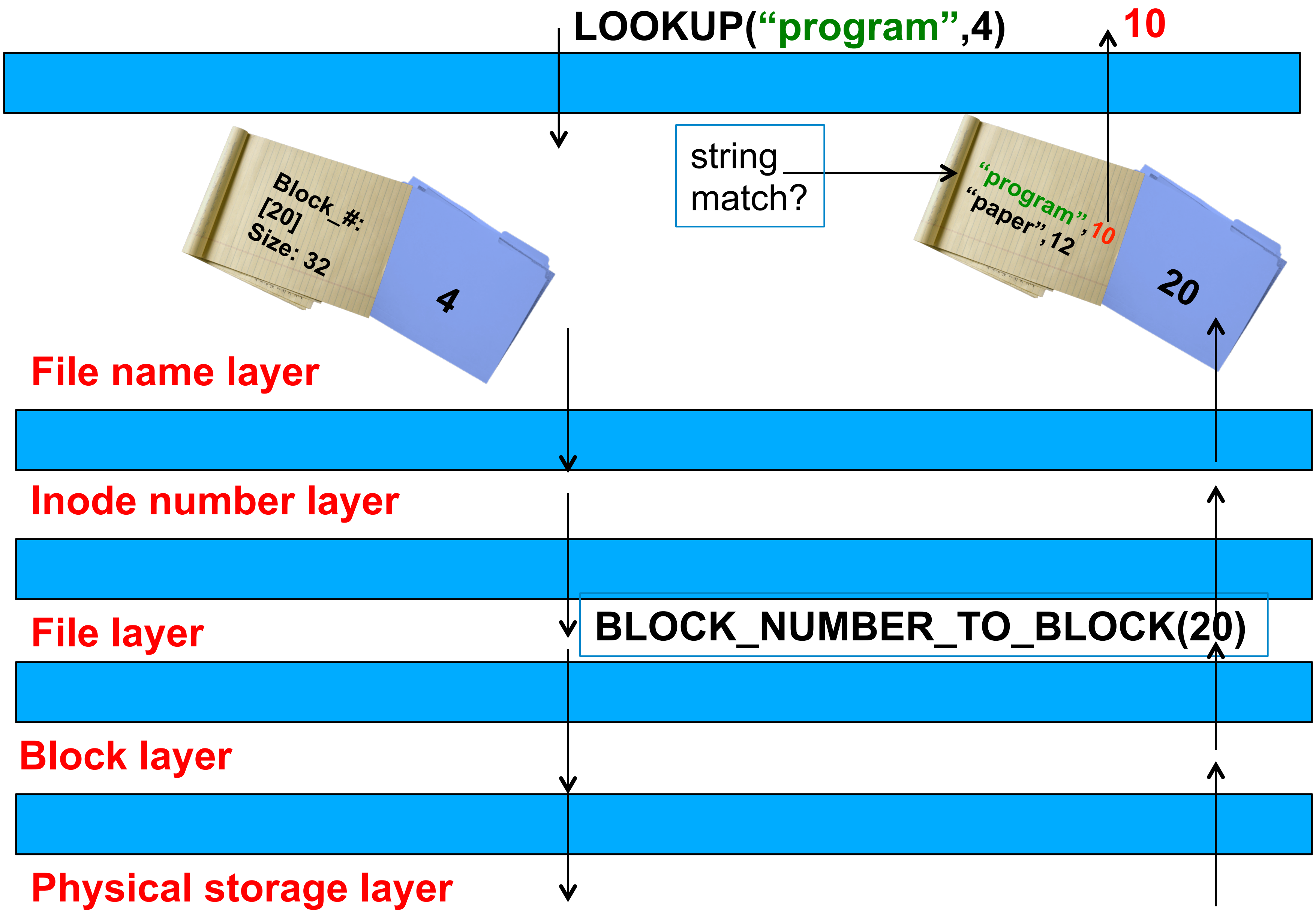
Inode number layer

File layer

Block layer

Physical storage layer

The File Name Layer



LOOKUP("program",4)

10

string match?

File name layer

Inode number layer

File layer

BLOCK_NUMBER_TO_BLOCK(20)

Block layer

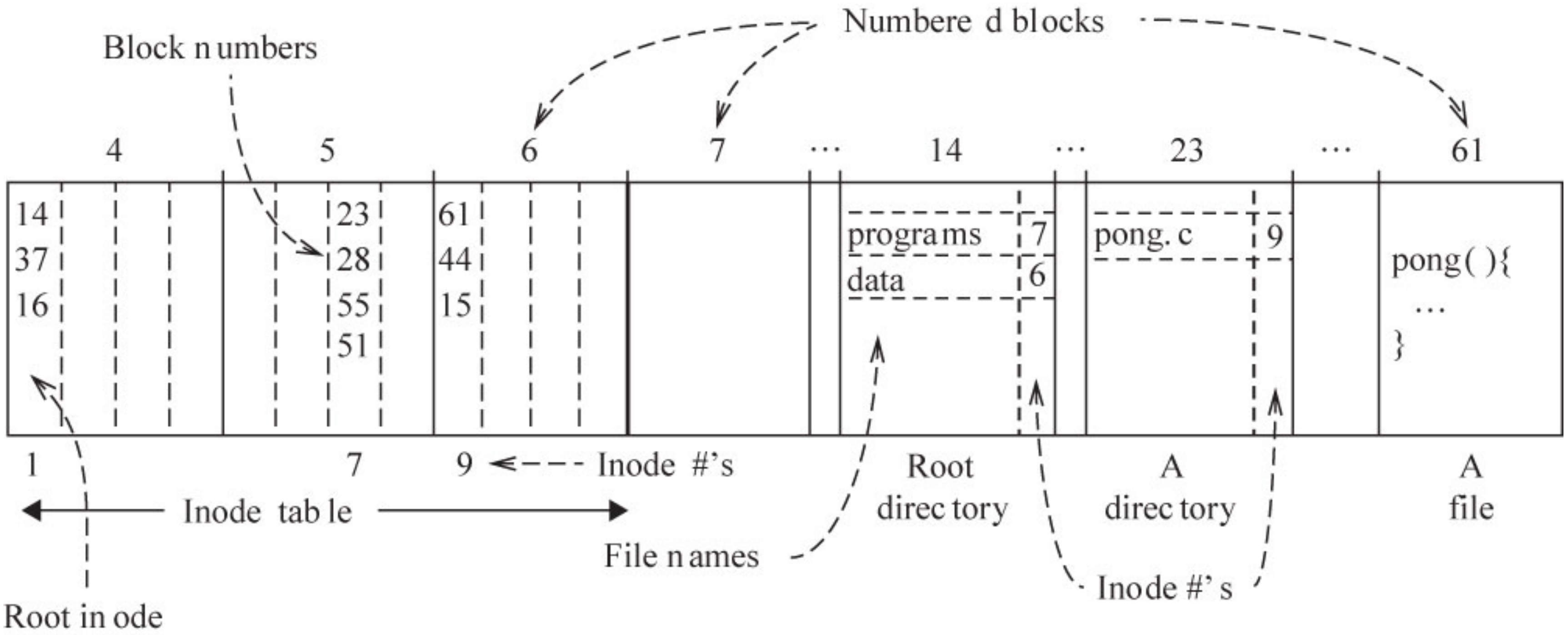
Physical storage layer

Lookup

- Resolve *filename* to *inode* in the context of a *directory*

```
procedure LOOKUP (string filename, int dir) returns integer

    block instance b
    inode instance i <- INODE_NUMBER_TO_INODE(dir)
    if (i.type != DIR) then return FAILURE
    for offset from 0 to i.size-1 do
        b <- INODE_NUMBER_TO_BLOCK (offset, dir)
        if STRING_MATCH(filename, b) then
                return INODE_NUMBER(filename,b)
        offset = offset + BLOCKSIZE
    return FAILURE
```

Example


Block numbers
Numbere d blocks
4
5
6
7
...
14
...
23
...
61
14
37
16
23
28
55
51
61
44
15
programs
7
data
6
pong. c
9
pong( ){
...
}
1
7
9
Inode #'s
Root
directory
A
directory
A
file
Inode table
File names
Inode #' s
Root in ode

The Path Name Layer

- Single directory - difficult to organize files
  - Path names and recursive lookup

```
procedure PATH_TO_INODE_NUMBER (string path, int dir) returns integer

    if (PLAIN_NAME(path)) return NAME_TO_INODE_NUMBER(path,dir)
            // plain name – no “/” separator
            // NAME_TO_INODE_NUMBER is a LOOKUP
    else
            dir <- LOOKUP(FIRST(path), dir)  // peel off and lookup first name
            path <- REST(path)  // pick rest of path name
             return PATH_TO_INODE_NUMBER(path,dir)  // recursion
```

The Path Name Layer

- Path names and recursive lookup
  - Result of a LOOKUP(name,dir) is an inode
  - Structured naming scheme provides ability to specify a path to name; recursive lookup
    - PATH_TO_INODE_NUMBER(“dir1/dir2/file”,inode_root) unfolds as:
    - LOOKUP (“dir1”, inode_root) -> inode_dir1
      - FIRST(“dir1/dir2/file”) -> “dir1”; REST -> “dir2/file”
      - PATH_TO_INODE_NUMBER(“dir2/file”,inode_dir1)
        - LOOKUP (“dir2”, inode_dir1) -> inode_dir2
        - PATH_TO_INODE_NUMBER(“file”,inode_dir2)
        - LOOKUP (“file”, inode_dir2) -> inode_file

Links

- Allow creation of *synonyms*
  - Bind a *name* in different *context* to the *same inode*
- E.g. creating a link to existing file in “projects”
  - LINK (“Mail/inbox/new-assignment”, “assignment”)
  - “assignment” is a synonym
    - Assume in context “Mail/inbox”, {“new-assignment”,481}
    - Now, in context “projects”, bind to same inode:
      - {“assignment”, 481)
  - Increment number of links for the inode 481
    - Extend inode data structure to add: **integer** *refcnt*
  - UNLINKing a file removes a binding
    - If the last binding (refcnt=0), mark inode and blocks as free

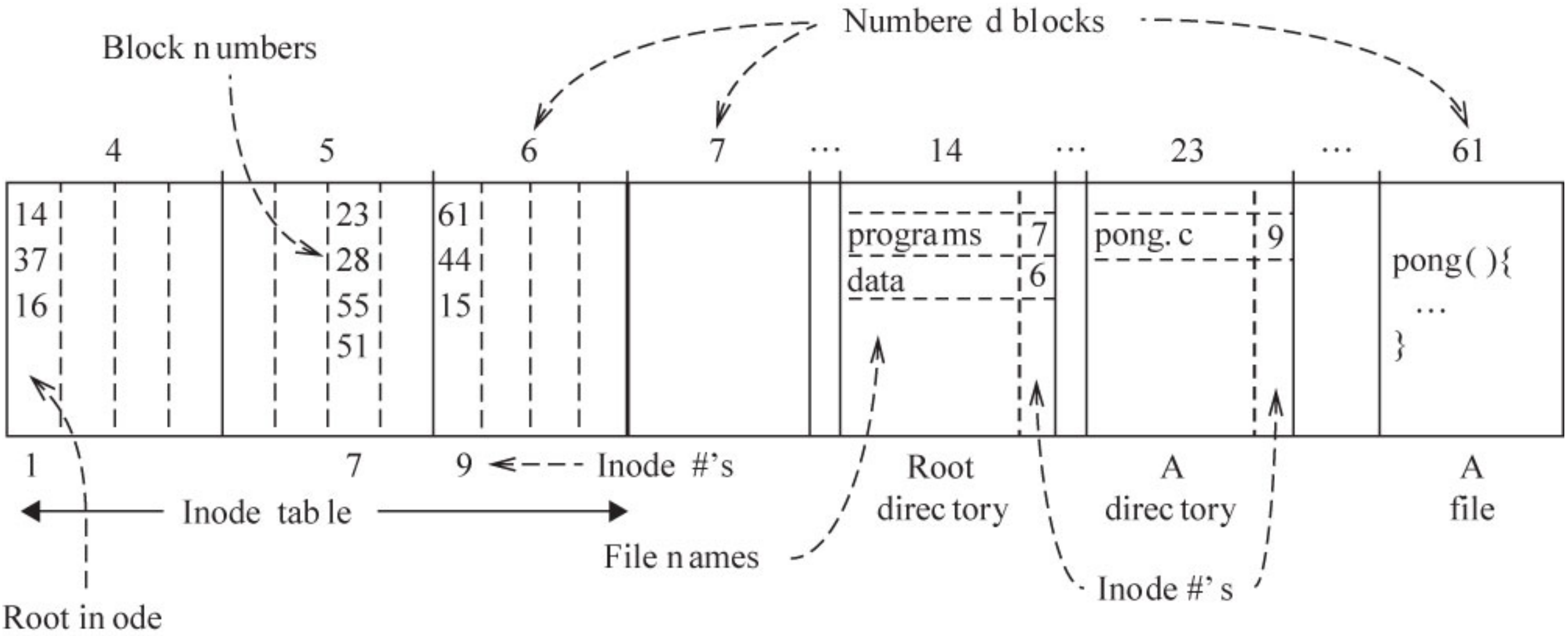
Renaming

- Can be achieved by adding a link, then removing a link:
  - LINK (from_name,to_name)
  - UNLINK (from_name)
- inode does not change; link count goes to 2, then back to 1

Absolute Path Name Layer

- Users often want both private and shared files
  - E.g. shared executable files/libraries with all; private documents; selectively share files with a group
- Need to provide a universal context
  - Root directory
  - Default: *well-known name*: “/”, inode 1
    - CHROOT: can specify different root inode
    - E.g. to constrain a program to a directory

Example


Block numbers
Numbere d blocks
4
5
6
7
…
14
…
23
…
61
14
37
16
23
28
55
51
61
44
15
programs
7
data
6
pong. c
9
pong( ){
…
}
1
7
9
Inode #'s
Inode tab le
Root
direc tory
A
direc tory
A
file
File n ames
Inode #' s
Root in ode

Symbolic Link Layer

- Allow users to create indirect names
- "Hard" links create additional bindings name -> inode
- Symbolic links create name -> name bindings
  - E.g. to name files on other devices
    - Can't easily guarantee unique inode namespace across multiple devices, e.g. USB drive
- Inode block_numbers[] contain characters of a file name rather than block numbers for inodes of type symbolic link
  - "assignments", "/other/disk/assignment3"

OPEN

- Open existing/create file; create entry in file table; return fd

```
procedure OPEN(string filename, int flags, mode)
    inode_number <- PATH_TO_INODE_NUMBER(filename,wd)
    if (inode_number = FAILURE) and flags=O_CREATE then
        inode_number <- CREATE (filename, mode)
    if (inode_number = FAILURE) then return FAILURE
    inode <- INODE_NUMBER_TO_INODE (inode_number)
    if (PERMITTED(mode,flags)) then
        file_index <- INSERT (file_table, inode_number)
        fd <- FIND_UNUSED_ENTRY(fd_table)
        fd_table[fd] <- file_index
        return fd
    else return FAILURE
```

READ

- Read “n” bytes of data from an open file

```
procedure READ(fd, char array reference buf, int n)
    file_index <- fd_table[fd]
    cursor <- file_table[file_index].cursor
    inode <- INODE_NUMBER_TO_INODE (file_table[file_index].inode_number)
    m <- MINIMUM(inode.size-cursor, n)
    atime of inode <- NOW()
    if (m=0) then return END_OF_FILE
    for (i from 0 to m-1) do {
        b <- INODE_NUMBER_TO_BLOCK(i+cursor, inode_number)
        COPY(b, buf, MINIMUM(m-i, BLOCKSIZE))
        i <- i + MINIMUM(m-i, BLOCKSIZE)
    file_table[file_index].cursor <- cursor+m
    return m
```

Mounting a file system

- Example: MOUNT(“/dev/fd1”, “/flash”)
  - “grafts” directory tree of file system in physical device /dev/fd1 onto /flash
    - This association is kept in volatile memory; does not persist after shutdown/reboot
    - Recorded in in-memory inode for “flash” that a file system has been mounted on it
      - Virtual file system layer – more on Chapter 4
    - LOOKUP under “/flash” uses the root file system of the mounted device

Shells – Implied Contexts

- A part of the UNIX system is the “shell”
  - A command interpreter
  - Where you type commands such as ls, cd, cat, echo, …
  - Commands can reference files
    - E.g. “ls” is an executable file
  - How does the shell resolve names?
    - Absolute path references
      - /bin/ls
    - Search paths
      - Environment variable determines the search path – e.g. PATH = /bin:/usr/bin:/home/usr/bin

Reading

- We will now skip to chapter 4
- Start reading 4.1, 4.2