

# inode-based File System

Yubin Xia

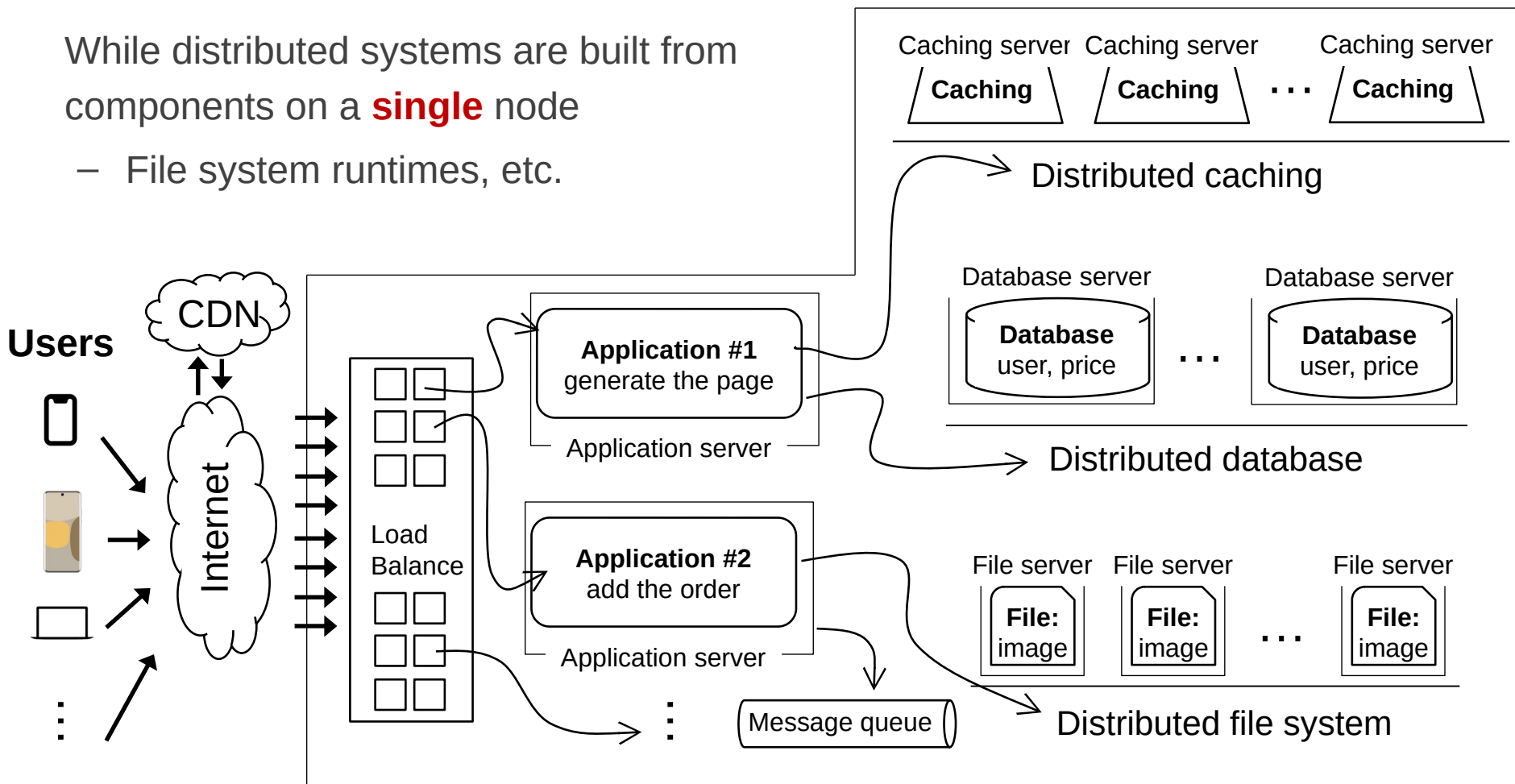
IPADS, Shanghai Jiao Tong University

<https://www.sjtu.edu.cn>

# Large-scale websites are built from distributed systems

While distributed systems are built from components on a **single** node

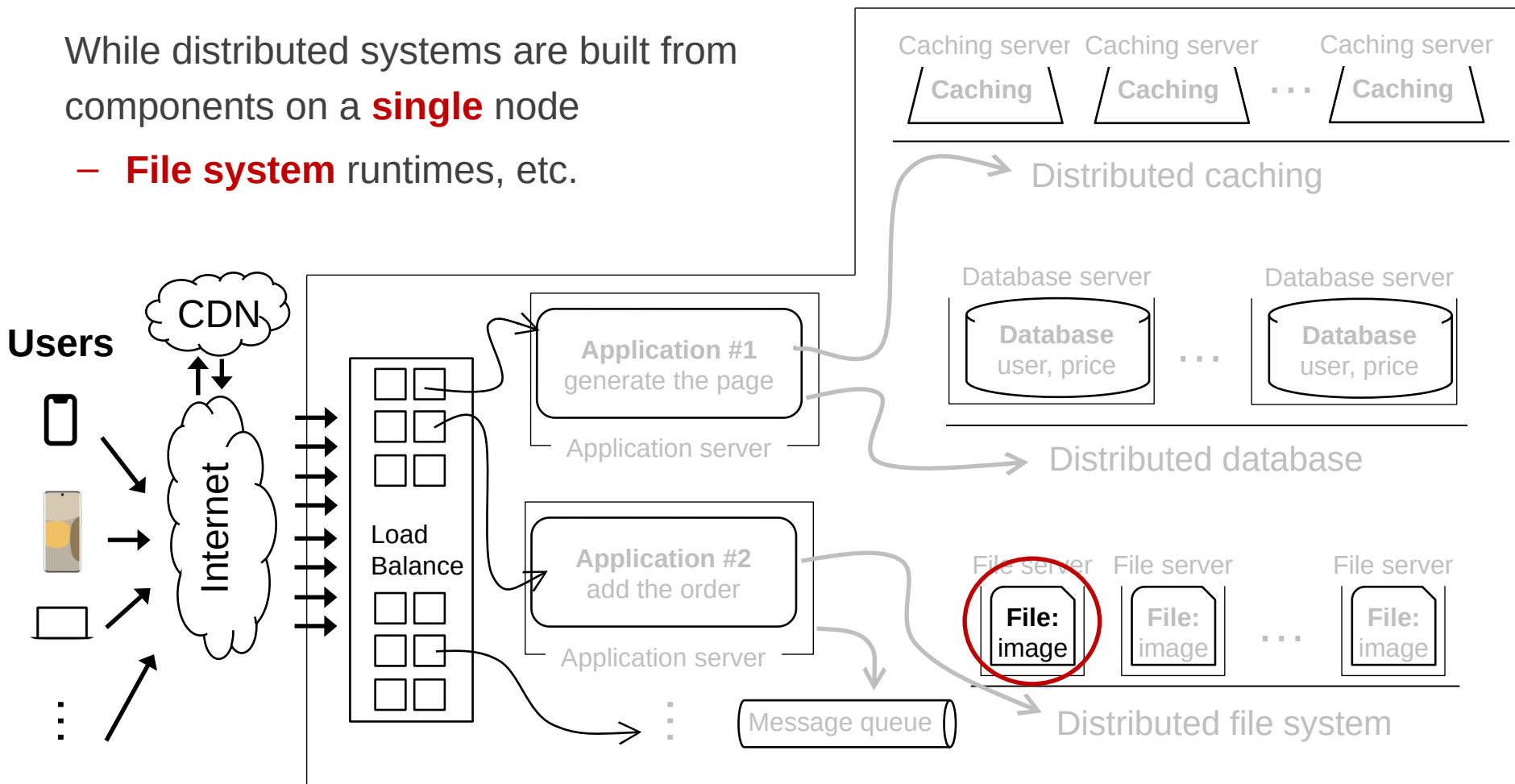
- File system runtimes, etc.



# Large-scale websites are built from distributed systems

While distributed systems are built from components on a **single** node

- **File system** runtimes, etc.





## **iNode-based File System**

# What is a file?

Storage as files



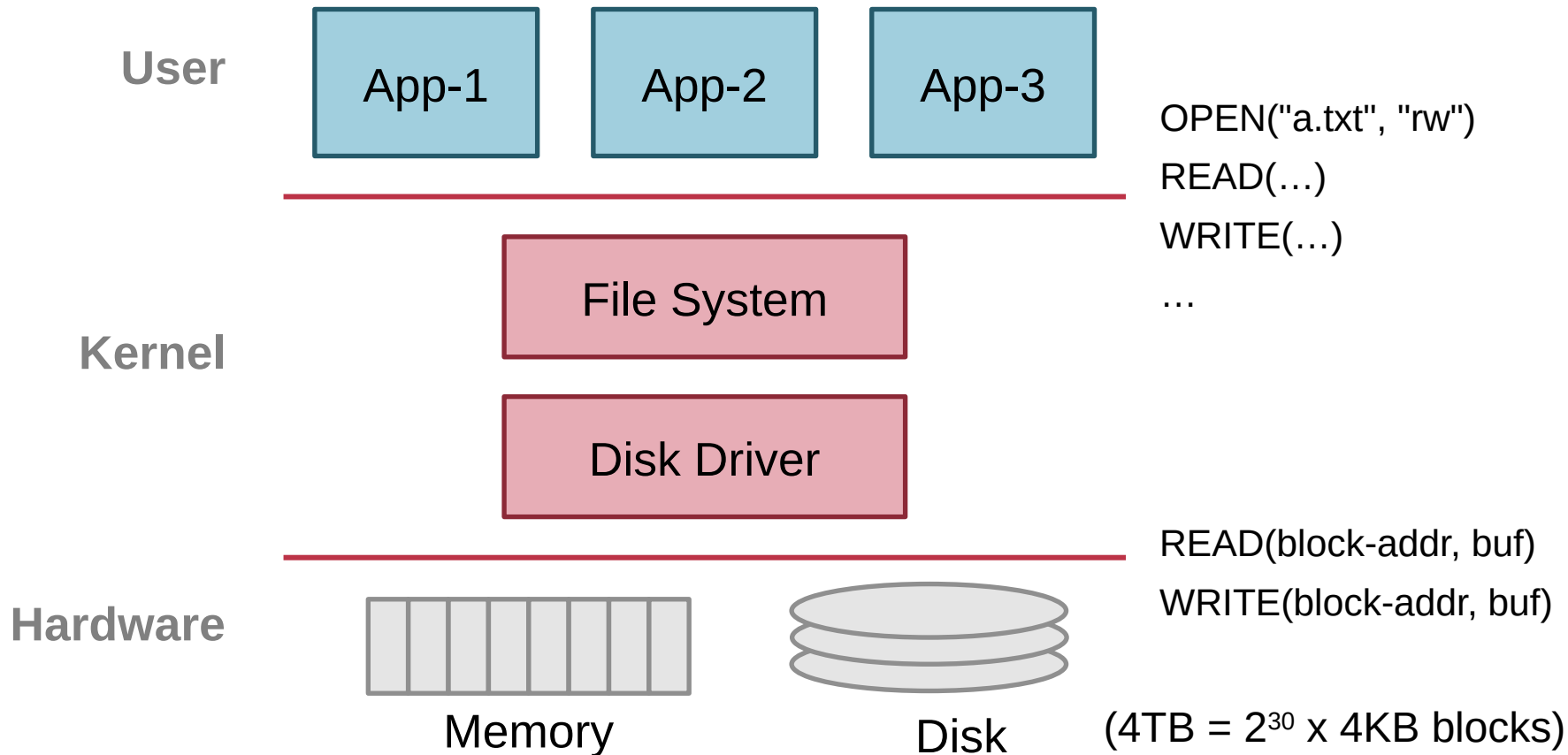
A file has **two key properties:**

- It is **durable** & **has a name**
- It is a high-level version of the memory abstraction

System layer implements files using **modules from hardware layer**

- **Divide-and-conquer** strategy
- Makes use of several hidden layers of machine-oriented names (addresses), one on another, to implement files
- Maps user-friendly names to these files

# The Big Picture



## Abstraction: API of UNIX File System

OPEN, READ, WRITE, APPEND, SEEK, CLOSE

FSYNC

STAT, CHMOD, CHOWN

RENAME, LINK, UNLINK, SYMLINK

MKDIR, CHDIR, CHROOT

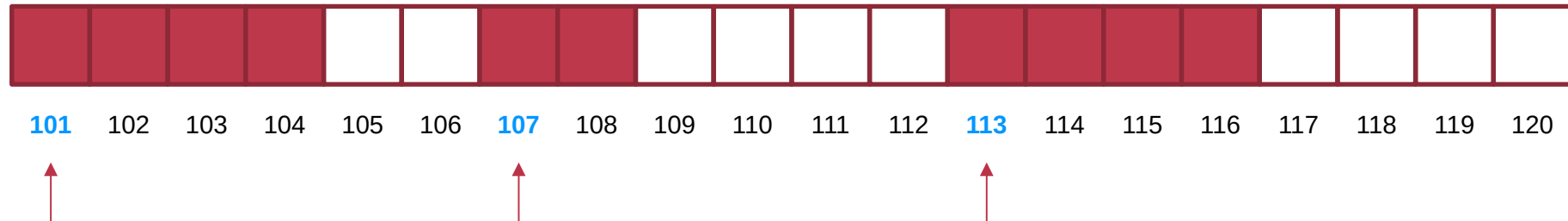
MOUNT, UNMOUNT

... .

# A Naive File System

Each file occupies one continuous range of blocks

- Use **block index** as file name, e.g., 107, 113
- Every file write will either **append** or **reallocate**



What are the **problems**?





**inode: 7 software layers**

# The Naming Layers of the UNIX FS (version 6)

Layer	Purpose	
Symbolic link layer	Integrate multiple file systems with symbolic links.	↑
Absolute path name layer	Provide a root for the naming hierarchies.	user-oriented names
Path name layer	Organize files into naming hierarchies.	↓
File name layer	Provide human-oriented names for files.	machine-user interface
Inode number layer	Provide machine-oriented names for files.	↑
File layer	Organize blocks into files.	machine-oriented names
Block layer	Identify disk blocks.	↓


# L1: Block Layer

Block num	Disk Block
--------------	---------------

**Mapping:** block number -> block data

```
procedure BLOCK_NUMBER_TO_BLOCK(integer b) -> block  
  return devices[b]
```

How to know the **size** of block?

- How to know which block is free?
- These **metadata** will also be stored on the same disk
- Super block! 

# Super Block

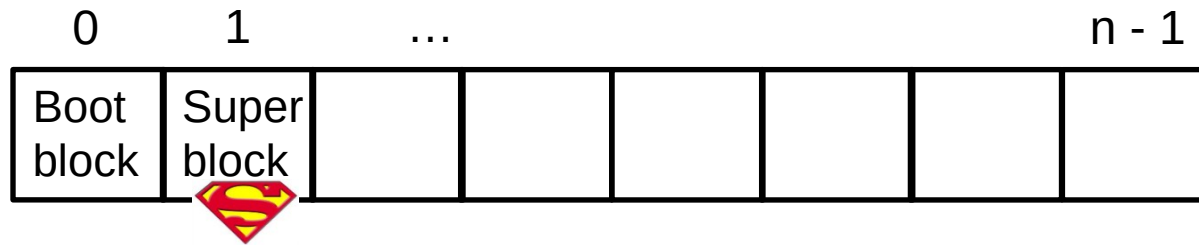
Block num	Disk Block
-----------	------------

One superblock per file system

- Kernel reads superblock when mount the FS

Superblock **contains**:

- Size of the blocks
- Number of free blocks
- A list of free blocks
- Other metadata of the file system (including inode info)



# L1: Block Layer

Block num	Disk Block
--------------	---------------

## Block size: a trade-off

- Neither too small or too big

## Question

- What will happen if the block size is too small? What if too big?
- How to efficiently track free blocks?

# L1: Block Layer

Block num	Disk Block
--------------	---------------

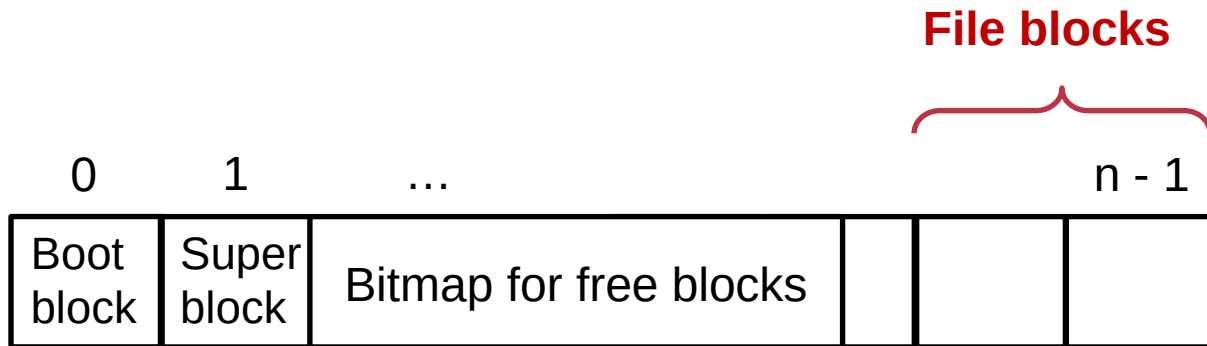
## Block size: a trade-off

- Neither too small or too big

## Question

- What will happen if the block size is too small? What if too big?
- How to efficiently track free blocks?

## Use a bitmap



## L2: File Layer

File (inode)	Block num	Disk Block
-----------------	--------------	---------------

### File requirements

- Store items that are larger than one block
- May grow or shrink over time
- A file is a linear array of bytes of arbitrary length
- Record which blocks belong to each file

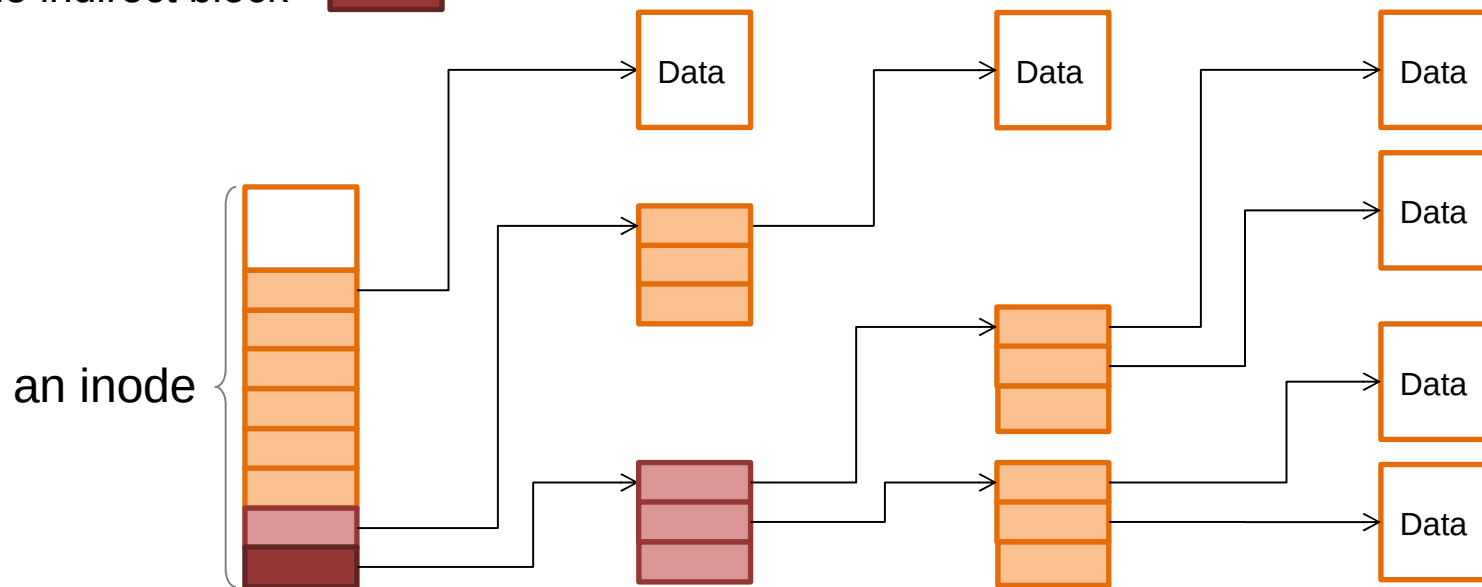
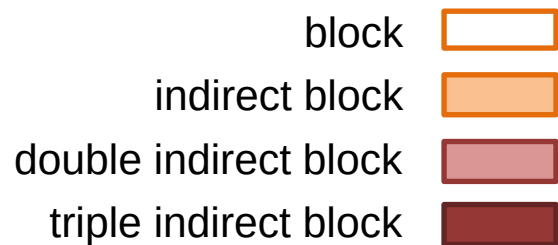
### inode (index node)

- A container for **metadata** about the file

```
struct inode
    integer block_nums[N]
    integer size
```

# inode for Larger Files

File (inode)	Block num	Disk Block
--------------	-----------	------------





## L2: File Layer

File (inode)	Block num	Disk Block
-----------------	--------------	---------------

Given an inode, can map a block index number (of a file) to a block number (of a disk)

- Index number: e.g., the 3rd block of a file is number 78

**procedure** INODE\_TO\_BLOCK(**integer** offset, **inode** i) -> **block**

*o* <- offset / BLOCKSIZE

*b* – INDEX\_TO\_BLOCK\_NUMBER(*i*, *o*)

**return** BLOCK\_NUMBER\_TO\_BLOCK(*b*)



**procedure** INDEX\_TO\_BLOCK\_NUMBER(**inode** i, **integer** index) -> **integer**

**return** *i.block\_nums[index]*

## L3: inode Number Layer

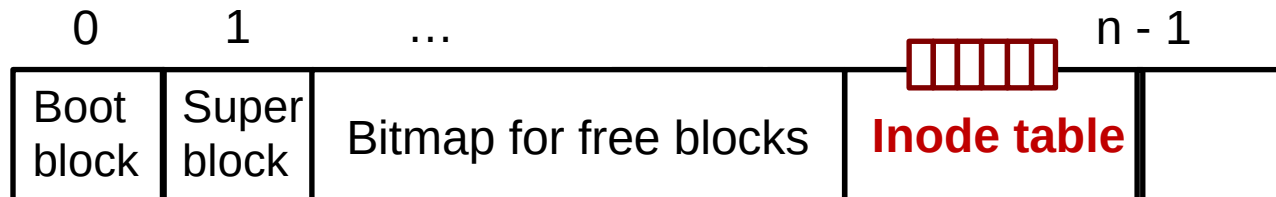
Inode num	File (inode)	Block num	Disk Block
-----------	--------------	-----------	------------

Mapping: **inode number** -> **inode**

```
procedure INODE_NUMBER_TO_INODE(integer num) -> inode  
  return inode_table[num]
```

**inode table**: at a fixed location on storage

- inode number is the index of inode table
- Track which inode number are in use, e.g. free list, a field in inode



## Put Layers so far Together

Inode num	File (inode)	Block num	Disk Block
--------------	-----------------	--------------	---------------

```
procedure INODE_NUMBER_TO_BLOCK(integer offset,  
integer inode_number) -> block  
  inode i = INODE_NUMBER_TO_INODE(inode_number)  
  o <- offset / BLOCKSIZE  
  b <- INDEX_TO_BLOCK_NUMBER(i, o)  
  return BLOCK_NUMBER_TO_BLOCK(b)
```

inode number is sufficient to operate a file. However,

- inode numbers are convenient names only for computer
- inode numbers change on different storage device

A file needs a more **user-friendly name**!

## L4: File Name Layer

File name	Inode num	File (inode)	Block num	Disk Block
-----------	-----------	--------------	-----------	------------

### File name

- Hide metadata of file management
- Files and I/O devices

### Mapping

- Mapping table is saved in directory
- Default context: **current working directory**
  - Context reference is an inode number
  - The current working directory **is also a file**

```
struct inode
    integer block_nums[N]
    integer size
    integer type
```

Overview of inode content

File name	inode num
helloworld.txt	12
cse2021.md	73

```
procedure name_to_inode(string filename, integer dir)-> integer
    return LOOKUP(dir, filename)
```

- Max length of a file name is **14 bytes** in UNIX version 6 (what does it mean?)

# LOOKUP in a Directory

File name	Inode num	File (inode)	Block num	Disk Block
-----------	-----------	--------------	-----------	------------

```
procedure LOOKUP(string filename, integer dir)-> integer
```

```
  block b
```

```
  inode i = INODE_NUMBER_TO_INODE(dir)
```

```
  if i.type != DIRECTORY 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
```

Name comparing method: **STRING\_MATCH**

– LOOKUP("cse2021", dir) will return 73

Next problem:

– What if there are too many files?

Overview of inode content

File name	inode num
helloworld.txt	12
cse2021.md	73

## L5: Path Name Layer

Path name	File name	Inode num	File (inode)	Block num	Disk Block
-----------	-----------	-----------	--------------	-----------	------------

Hierarchy of directories and files

- Structured naming: E.g. "projects/paper"

```
procedure PATH_TO_INODE_NUMBER(string path, integer dir)-> integer  
  if PLAIN_NAME(path)return NAME_TO_INODE_NUMBER(path,dir)  
else  
  dir <- LOOKUP(FIRST(path), dir)  
  path <- REST(path)  
  return PATH_TO_INODE_NUMBER(path,dir)
```

**Context:** the working directory **dir**

# Links

Path name	File name	Inode num	File (inode)	Block num	Disk Block
-----------	-----------	-----------	--------------	-----------	------------

**LINK:** create shortcut for long names

- LINK("Mail/inbox/new-assignment", "assignment")
- Turns strict hierarchy into a directed graph
  - Users cannot create links to directories -> acyclic graph
- Different filenames, same inode number

## UNLINK

- Remove the binding of filename to inode number
- If UNLINK last binding, put inode/blocks to a free-list
  - A reference counter is needed

# Links

Path name	File name	Inode num	File (inode)	Block num	Disk Block
-----------	-----------	-----------	--------------	-----------	------------

## Reference count

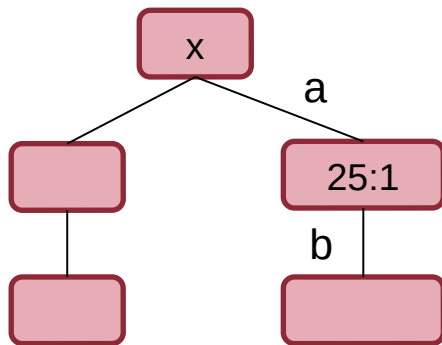
- An inode can bind multiple file names
- +1 when **LINK**, -1 when **UNLINK**
- A file will be deleted when reference count is 0
- **No cycle allowed**
  - Except for '.' and '..'
  - Naming current and parent directory with no need to know their names

```
struct inode
    integer block_nums[N]
    integer size
    integer type
    integer refcnt
```

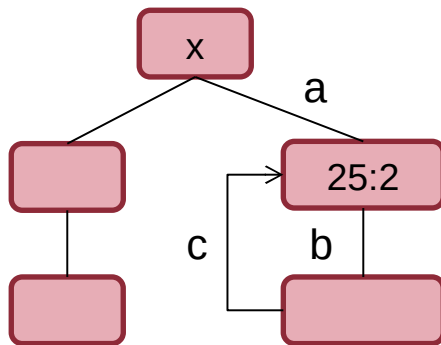


# No Cycle for LINK

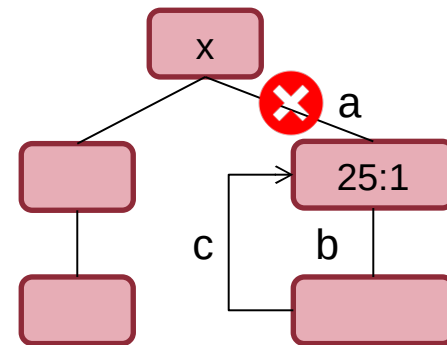
Path name	File name	Inode num	File (inode)	Block num	Disk Block
-----------	-----------	-----------	--------------	-----------	------------



- /a/b is a directory
- The refcnt of a is 1
- a's inode num is 25



- LINK ("/a/b/c", "a")
- Cause a cycle!
- Refcnt of a is 2



- UNLINK ("/a")
- Refcnt of a is 1, so the inode 25 is not deleted
- Now inode 25 is disconnected from graph
- No one can get it!

**Think:** Why deleting a directory requires to delete all the files in it first?

## Renaming - 1

Path name	File name	Inode num	File (inode)	Block num	Disk Block
-----------	-----------	-----------	--------------	-----------	------------

- 1 UNLINK(to\_name)
- 2 LINK(from\_name, to\_name)
- 3 UNLINK(from\_name)

**Text edit** usually save editing file in a **temp** file

- Edit in `.a.txt.swp`, then rename `.a.txt.swp` to `a.txt`

What if the computer **fails between 1 & 2**?

- The file `to_name` will be lost, which will surprise the user
- Need **atomic** action (in later lectures)

## Renaming - 2

Path name	File name	Inode num	File (inode)	Block num	Disk Block
-----------	-----------	-----------	--------------	-----------	------------

1 LINK(from\_name, to\_name)

2 UNLINK(from\_name)

Weaker specification without atomic actions

- 1. Changes the inode number in for to\_name to the inode number of from\_name
- 2. Removes the directory entry for from\_name

If fails between 1 & 2

- Must increase reference count of **from\_name**'s inode on recovery

If **to\_name** already exists

- It will always exist even if the machine fails between 1 & 2

## L6: Absolute Path Name Layer

Absolute path	File name	Inode num	File (inode)	Block num	Disk Block
Path name					

### HOME directory

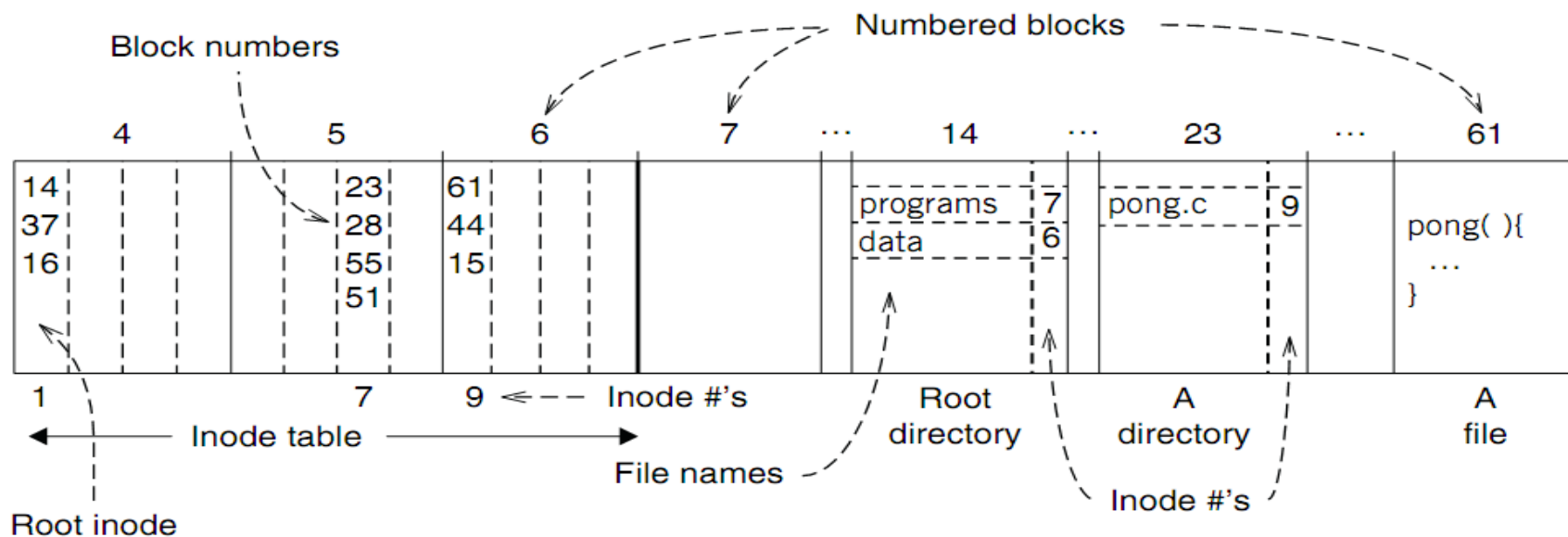
- Every user has a default working directory (HOME) after login
- Problem: no sharing of files between users

### Introducing the **root** directory

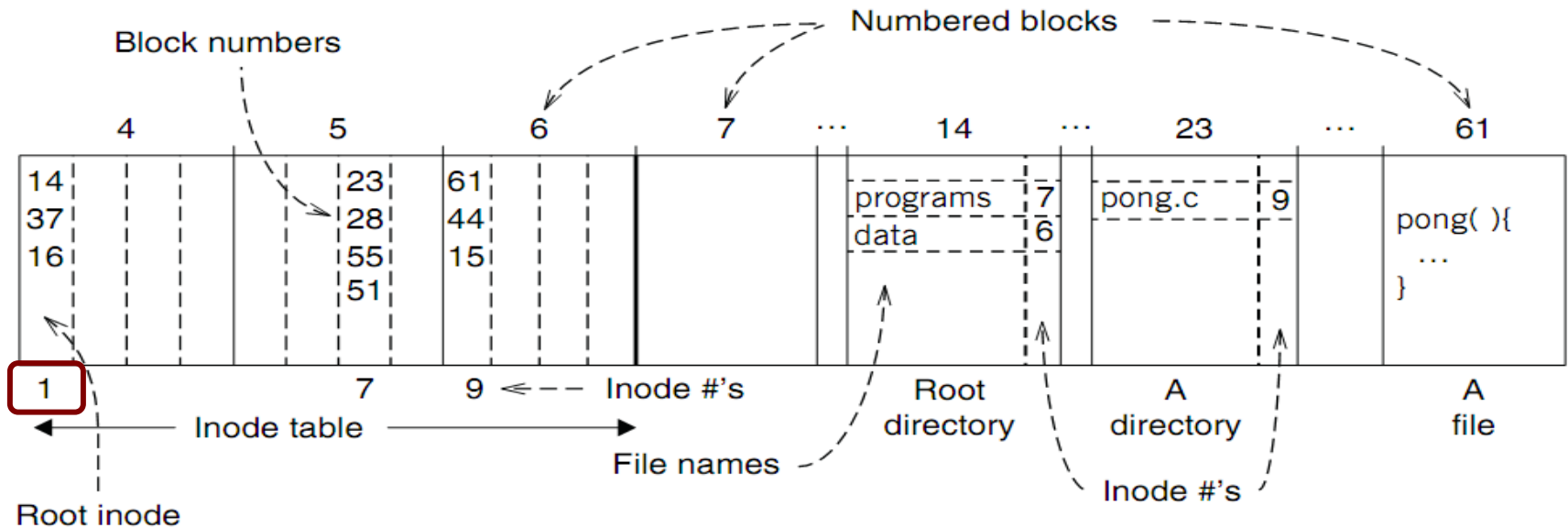
- A universal context for all users
- A well-known name: '/'
- Both './' and '../' are linked to '/'

```
procedure GENERATEPATH_TO_INODE_NUMBER(string path) -> integer  
  if path[0] = "/" return PATH_TO_INODE_NUMBER(path, 1)  
  else return PATH_TO_INODE_NUMBER(path, wd)
```

## An Example: Find Blocks of "/programs/pong.c"

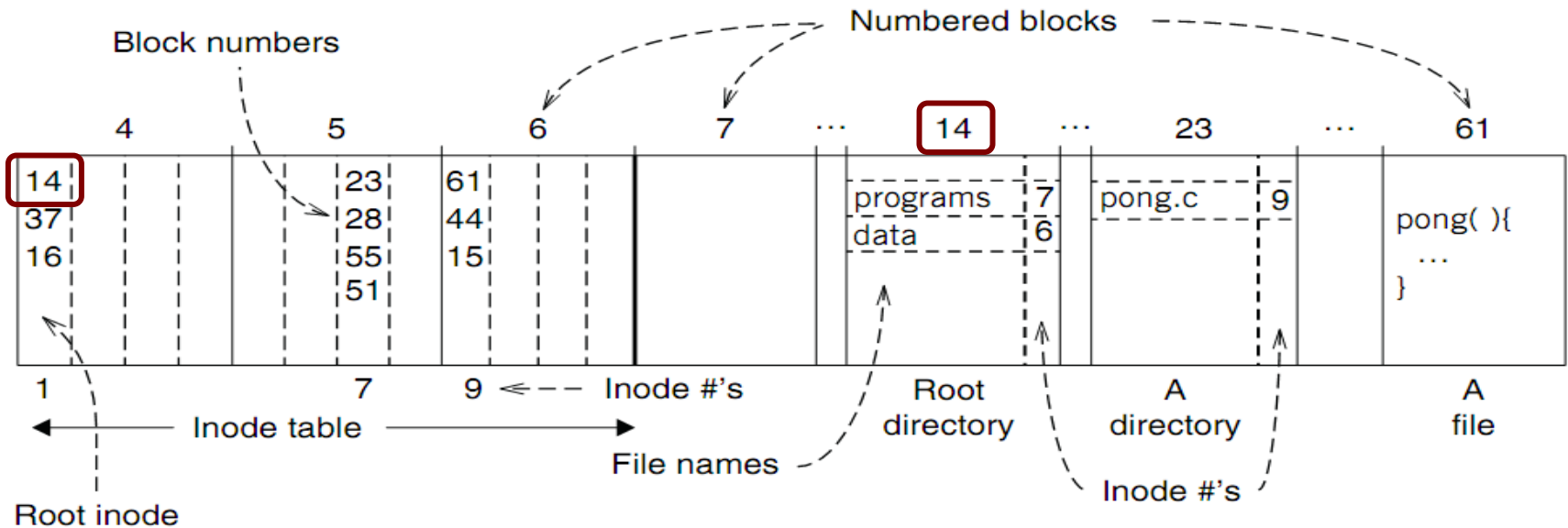


## An Example: Find Blocks of "/programs/pong.c"



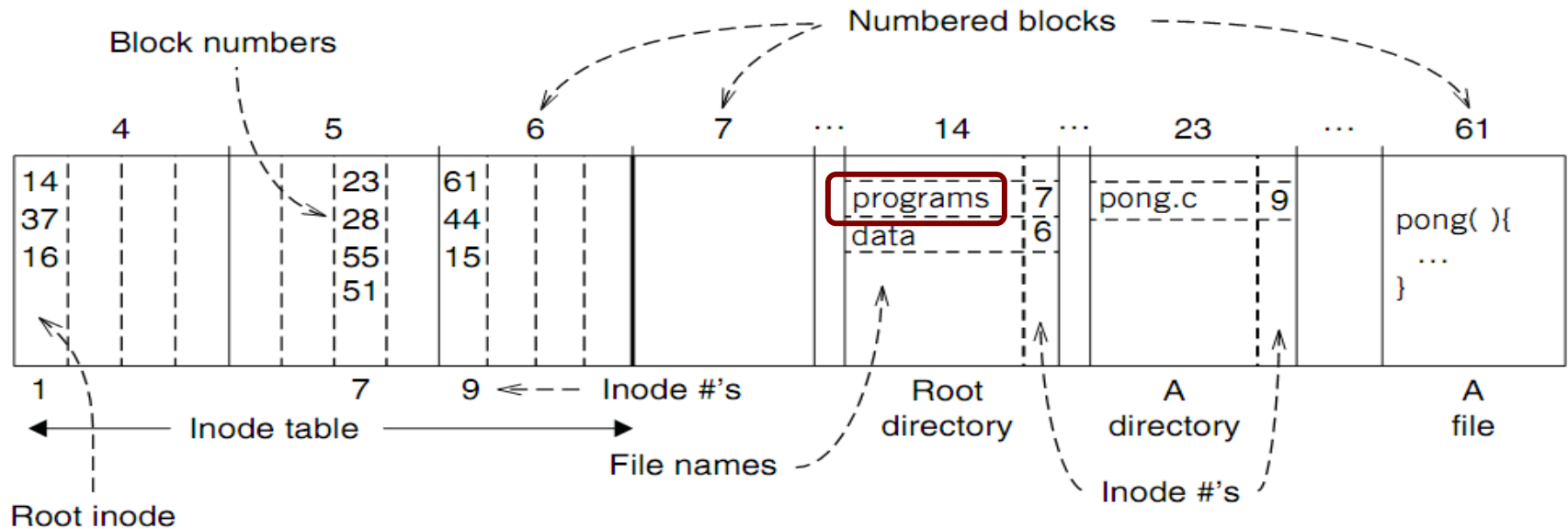
'/' root directory: inode is 1

## An Example: Find Blocks of "/programs/pong.c"



Find the first directory in '/' by block number

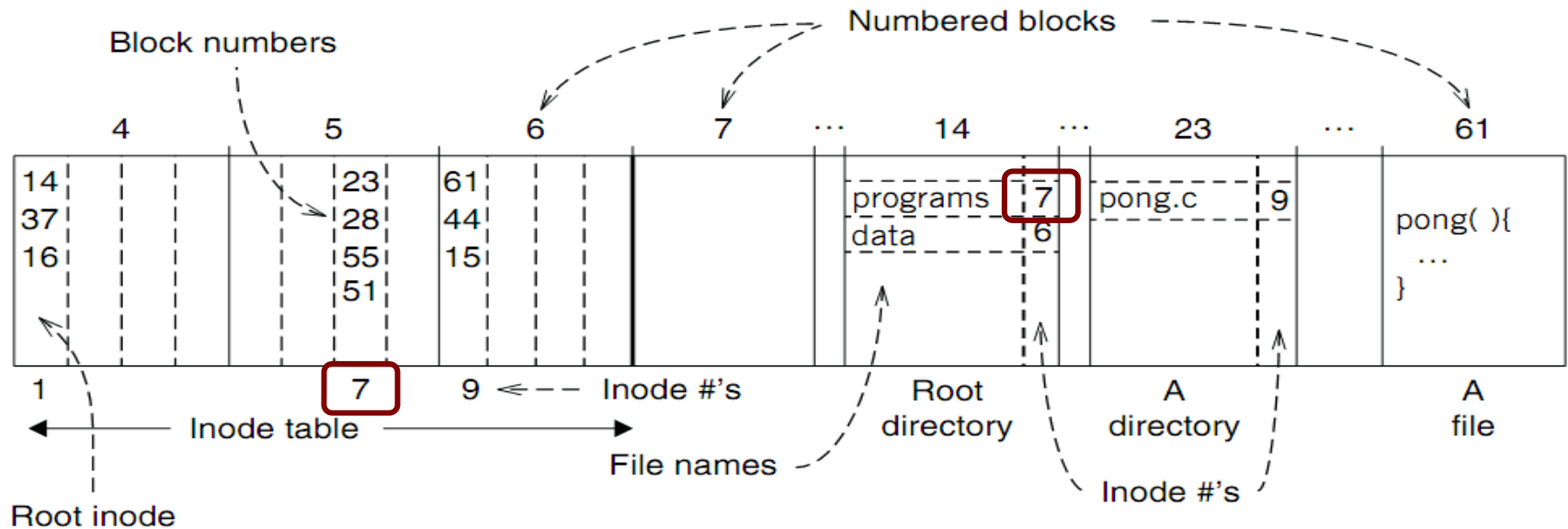
## An Example: Find Blocks of "/programs/pong.c"



Find '/programs' by comparing name

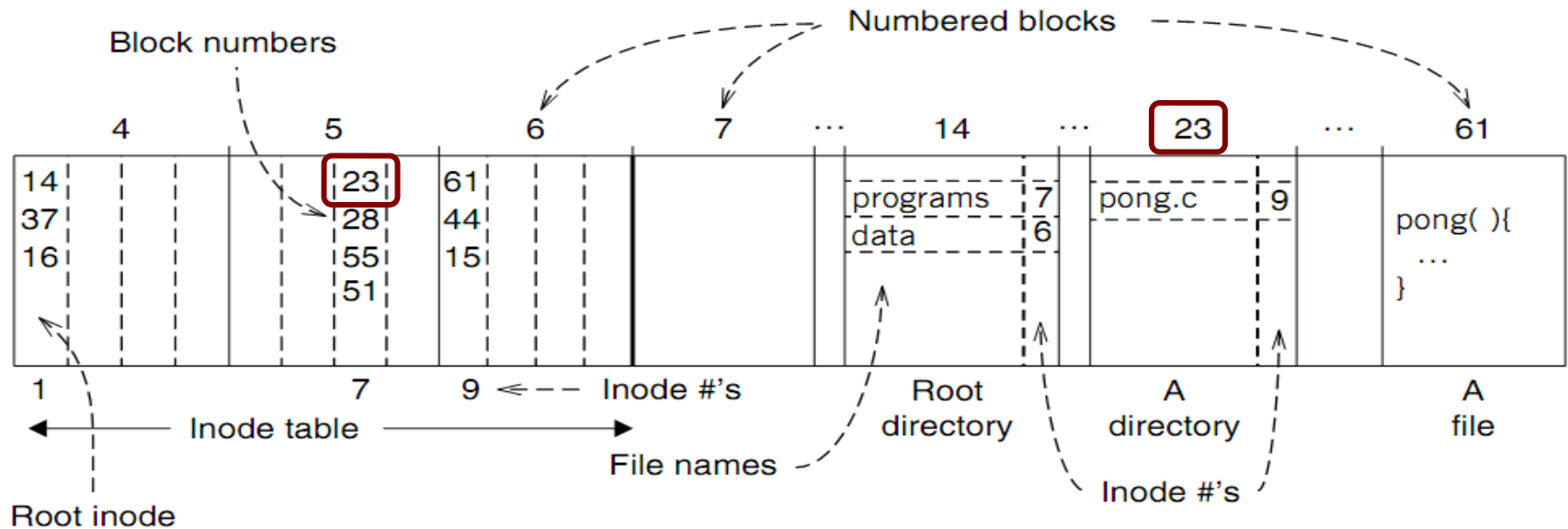


## An Example: Find Blocks of "/programs/pong.c"



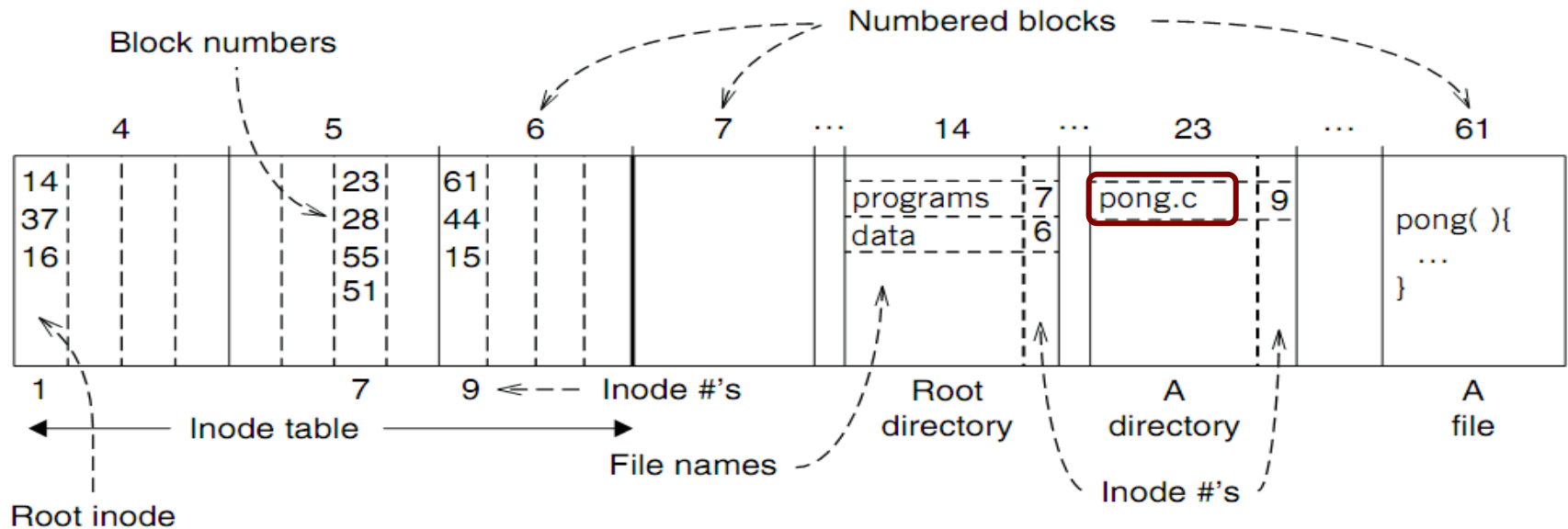
Find `/programs` inode by its inode number 7

## An Example: Find Blocks of "/programs/pong.c"



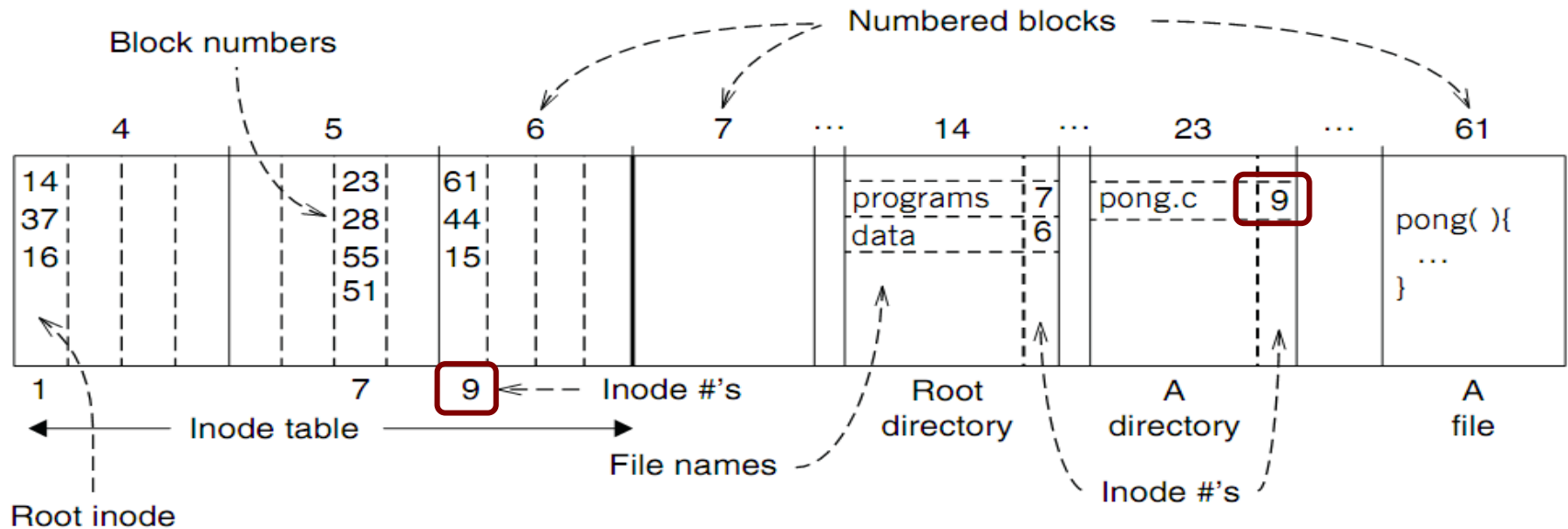
Find the first file in '/programs/'

## An Example: Find Blocks of "/programs/pong.c"



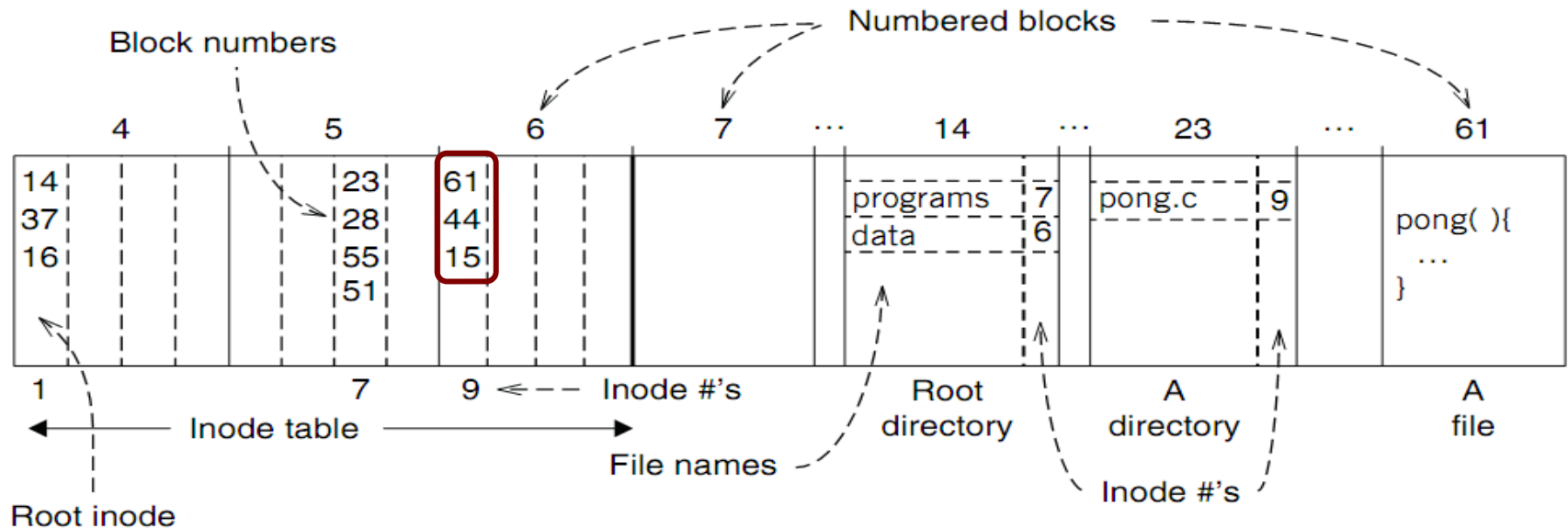
Find `/programs/pong.c` by comparing its name

## An Example: Find Blocks of "/programs/pong.c"



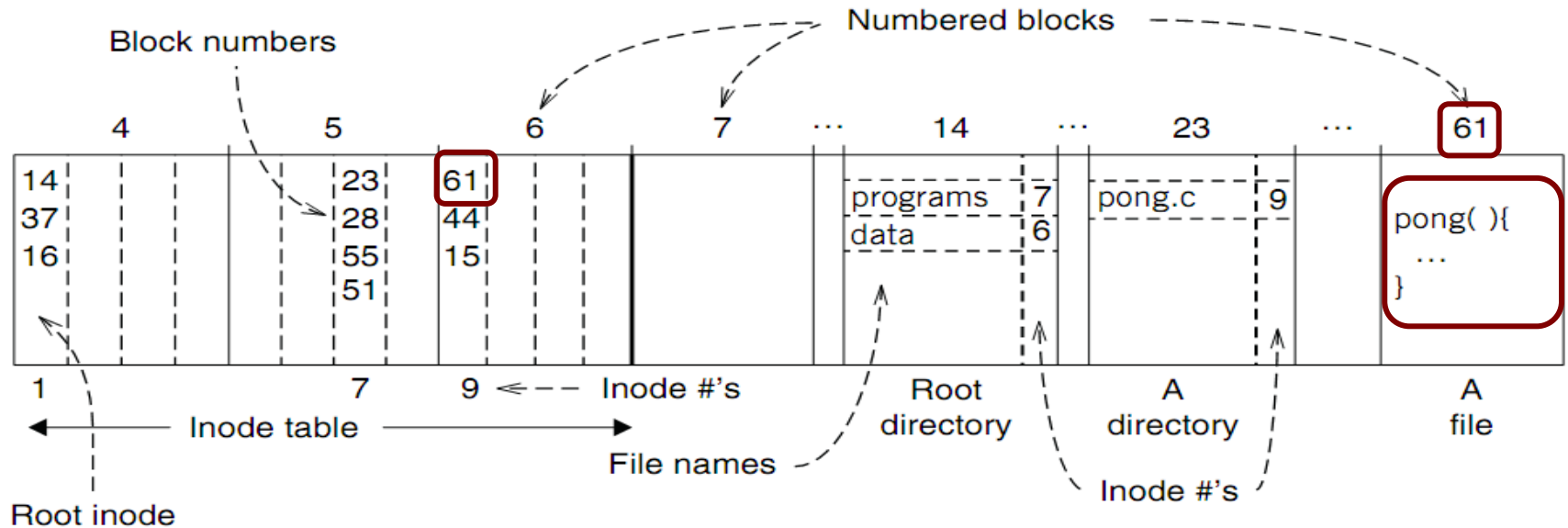
Find inode of '/programs/pong.c' by the inode number 9

## An Example: Find Blocks of "/programs/pong.c"



Find block number of '/programs/pong.c'

## An Example: Find Blocks of "/programs/pong.c"



Find data of block 61 by its block number

- And data of block 44 & 15

# Directly Dump a Directory

```
$ ls -ai temp
```

```
7536909 . 7530417 .. 7536939 a 7536940 b 7536941 c 7536942 d
```

```
$ echo "obase=16;7536909;7530417;7536939;7536940;7536941;7536942" | bc
73010D 72E7B1 73012B 73012C 73012D 73012E
```

```
$ sudo /sbin/debugfs /dev/sda1
```

```
debugfs 1.43.4 (31-Jan-2017)
```

```
debugfs: dump temp temp.out
```

```
debugfs: quit
```

```
$ xxd temp.out
```

```
00000000: 0d01 7300 0c00 0102 2e00 0000 b1e7 7200 ..s.....r.
00000100: 0c00 0202 2e2e 0000 2b01 7300 0c00 0101 .....+.s....
00000200: 6100 0000 2c01 7300 0c00 0101 6200 0000 a...,.s....b...
00000300: 2d01 7300 0c00 0101 6300 0000 2e01 7300 -.s.....c....s.
00000400: c40f 0101 6400 0000 0000 0000 0000 0000 ....d.....
00000500: ...
```

# Directly Dump a Directory

```
struct ext4_dir_entry {  
    uint32_t  inode_number;  
    uint16_t  dir_entry_length;  
    uint8_t   file_name_length;  
    uint8_t   file_type;  
    char      name[EXT4_NAME_LEN];  
}
```

## File Type

0x0: Unknown  
0x1: Regular file  
0x2: Directory  
0x3: Character device file  
0x4: Block device file  
0x5: FIFO  
0x6: Socket  
0x7: Symbolic link

0d01	7300	0c00	0102	2e00	0000
b1e7	7200	0c00	0202	2e2e	0000
2b01	7300	0c00	0101	6100	0000
2c01	7300	0c00	0101	6200	0000
2d01	7300	0c00	0101	6300	0000
2e01	7300	c40f	0101	6400	0000

0d01	7300	0c00	0102	2e00	0000
------	------	------	------	------	------



0d01 7300: inode number  
0c00: entry length is 12 bytes  
01: file name length is 1 byte  
01: file type is regular file  
2e00 0000: file name (2e -> ".")



## L7: Symbolic Link Layer

Symbolic link					
Absolute path	File name	Inode num	File (inode)	Block num	Disk Block
Path name					

Name files **on other disks**

- Inode is different on other disks
- Supports to attach new disks to the name space

### Two options

- Make inodes unique across all disks (not good)
- Create synonyms for the files on the other disks

### Introducing layer 7: **soft link (symbolic link)**

- SYMLINK
- Add another type of inode

## Directly Dump a Symbolic Link

```
$ ln -s "/tmp/abc" s-link
```

```
$ ls -l s-link
```

```
7536945 lrwxrwxrwx 1 xiayubin 8 Sep 20 08:01 s-link -> /tmp/abc
```

```
$ readlink s-link
```

```
/tmp/abc
```

What does "8" means **File size**

```
$ cat s-link
```

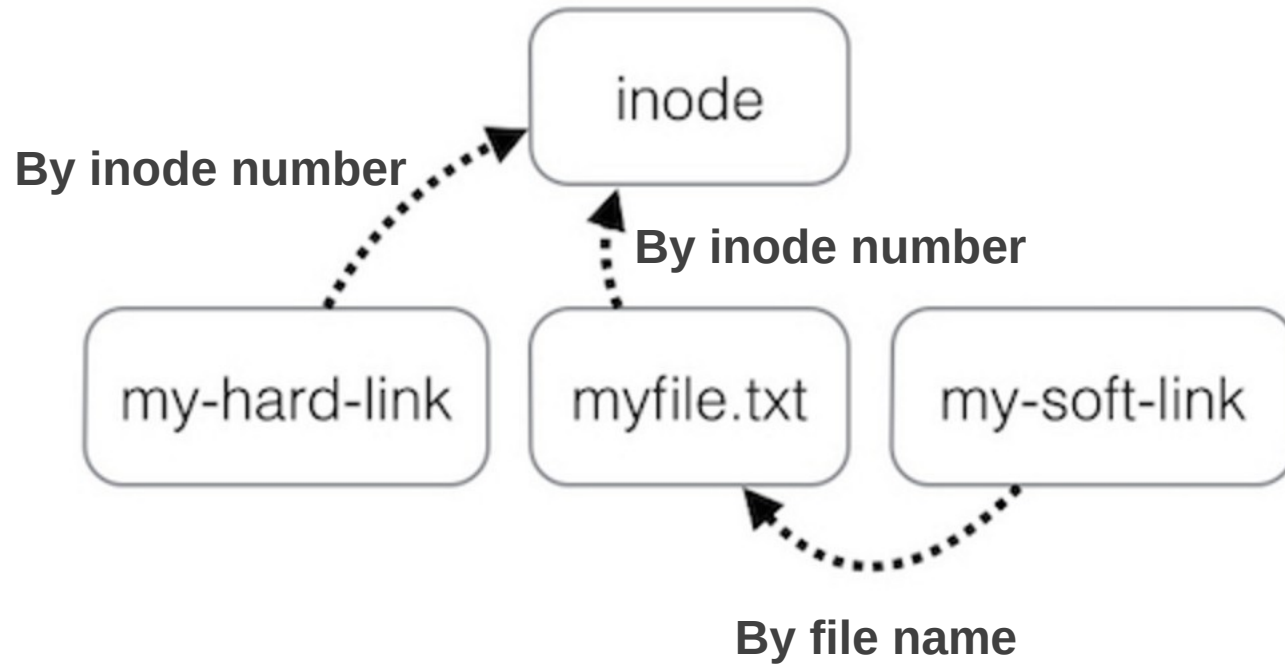
```
cat: slink: No such file or directory
```

```
$ echo "hello, world" > /tmp/abc
```

```
$ cat s-link
```

```
hello, world
```

## Two Types of Links (Synonyms)



## Sidebar: Notice the Context Change

Another **interesting behavior** of soft link

- There is a directory: `"/Scholarly/programs/www"`
- The root directory contains a soft link
  - `"/CSE-web" -> "/Scholarly/programs/www"`
- Run following commands
  - `cd CSE-web`
  - `cd ..`
- What is the current directory? Why?
  - `".."` is resolved in a new default context: by bash, not file system

## Sidebar: Notice the Context Change

The bash tries to be **"human-friendly"**

- When you `cd /into/a/symlink/`, the shell remembers the old location (in **\$OLDPWD**) and will use that directory when you `cd ..` under the assumption that you want to return to the directory you were just in

If you want to use the real `..`, then you must also use **"`cd -P ..`"**

*The -P option says to use the physical directory structure instead of following symbolic links (see also the -P option to the set built-in command); the -L option forces symbolic links to be followed.*

```
$ cd
$ cd a/b/symlink
$ cd -P ..
$ pwd -P
/home/sarnold
$
```

# Summary of File System's 7 Layers

## File name is **not** part of a file

- Name is **not** a part of an inode
- Name is data of a directory, and metadata of a file system
- One inode can have several names (hard links)

## Hard links are equal

- If a file has two names, both are links (instead of "a link and a name")

## Directory size is small

- Only mapping from name to inode number
- The term "folder" is somewhat misleading