Slides from: Andra C. Arpaci-Dusseau
Remzi H. Arpaci-Dusseau

PERSISTENCE: FILE SYSTEM API

Questions answered in this lecture:

How to **name** files?

- What are **inode numbers**?
- How to **lookup** a file based on pathname?
- What is a **file descriptor**?

What is the difference between **hard and soft links**?

How can **special requirements** be communicated to file system (fsync)?

What is a File?

Array of persistent bytes that can be read/written

File system consists of many files

- Refers to collection of files
- Also refers to part of OS that manages those files

Files need names to access correct one

File Names

Three types of names

- Unique id: inode numbers
- Path
- File descriptor

Inode Number

Each file has exactly one **inode number**

Inodes are unique (at a given time) within file system

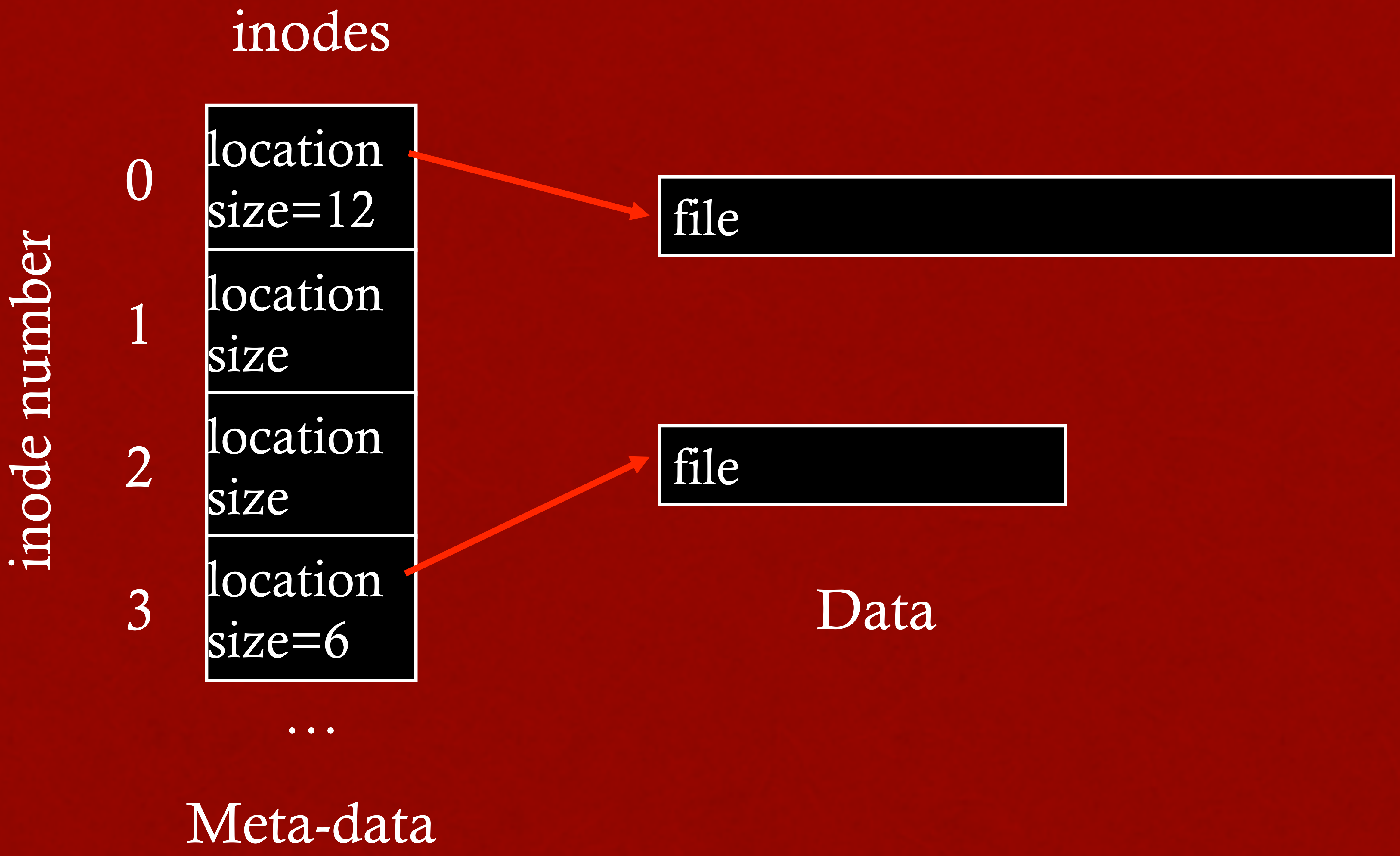
Different file system may use the same number, numbers may be recycled after deletes

See inodes via “ls –i”; see them increment…

WHAT DOES "I" STAND FOR?

"In truth, I don't know either. It was just a term that we started to use. 'Index' is my best guess, because of the slightly unusual file system structure that stored the access information of files as a flat array on the disk…"

~ Dennis Ritchie



inodes

inode number

0 location size=12

1 location size

2 location size

3 location size=6

…

Meta-data

file

file

Data

FILE API (ATTEMPT 1)

```
read(int inode, void *buf, size_t nbyte)

write(int inode, void *buf, size_t nbyte)

seek(int inode, off_t offset)
```

seek does not cause disk seek until read/write

Disadvantages?

- names hard to remember
- no organization or meaning to inode numbers
- semantics of offset across multiple processes?

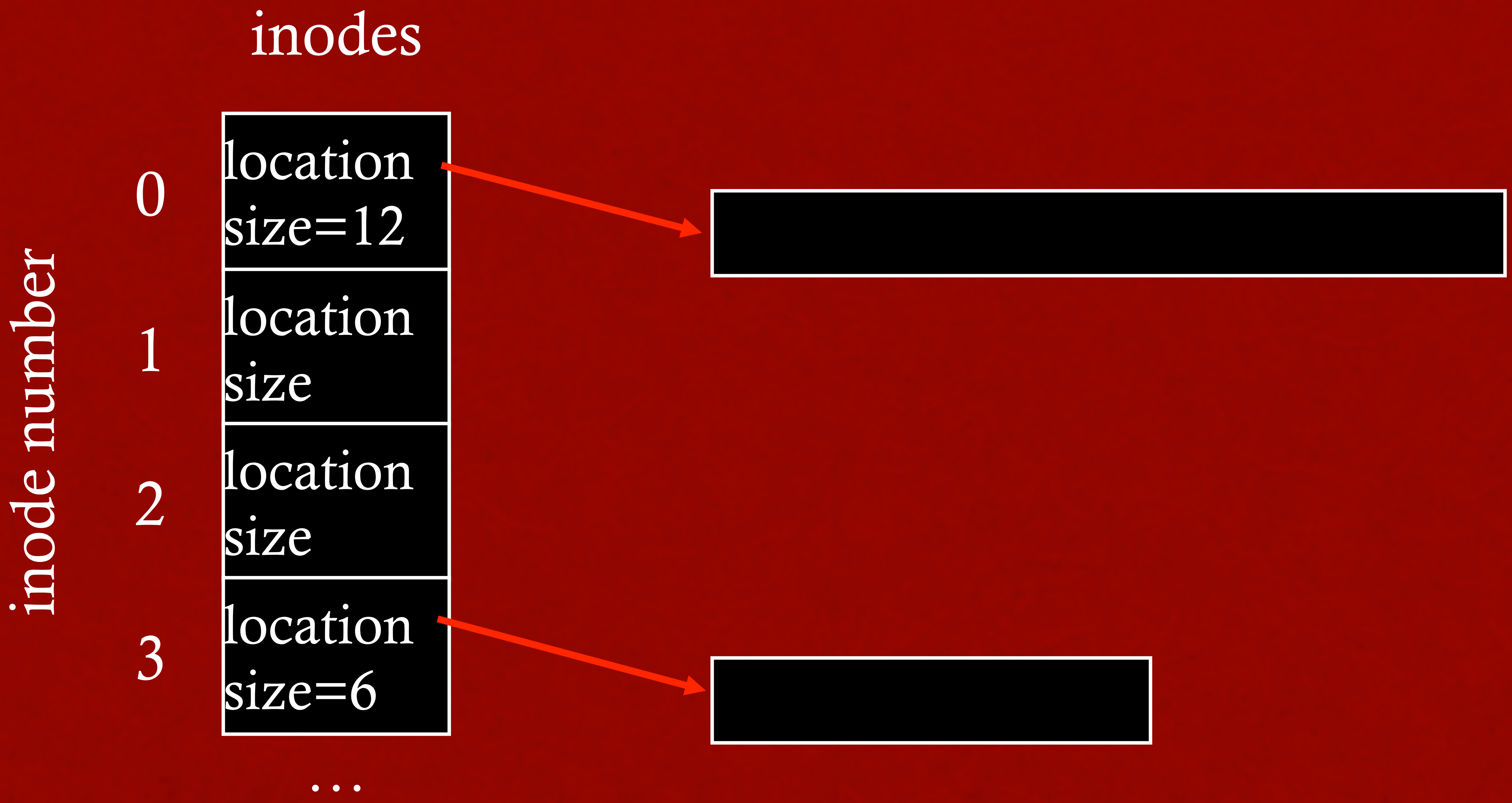
PATHS

String names are friendlier than number names

File system still interacts with inode numbers

Store *path-to-inode* mappings in predetermined “root” file (typically inode 2)

- Directory!



inodes

inode number

0 location
size=12

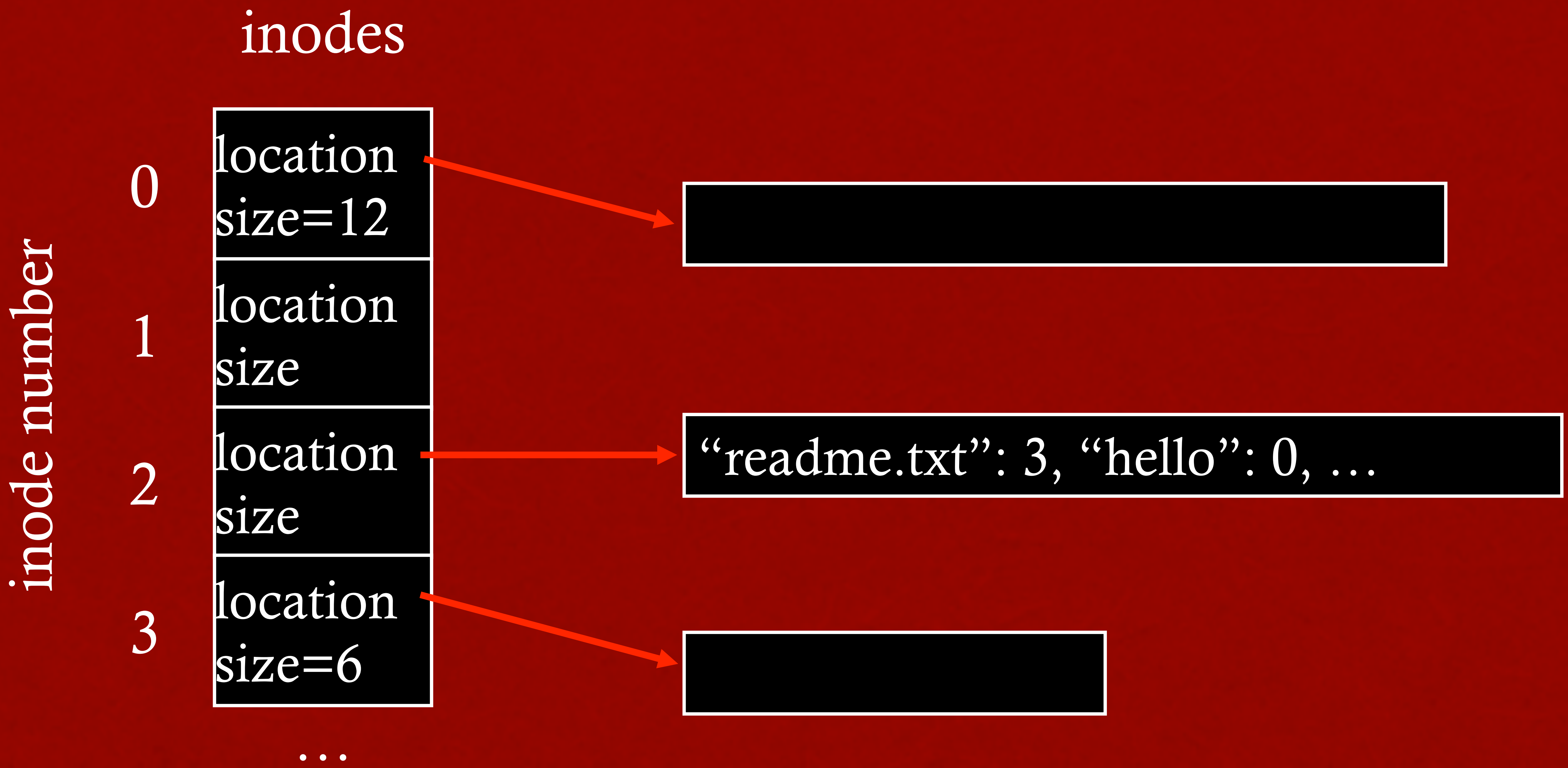
1 location
size

2 location
size

3 location
size=6

...

inodes



inode number

0 location size=12

1 location size

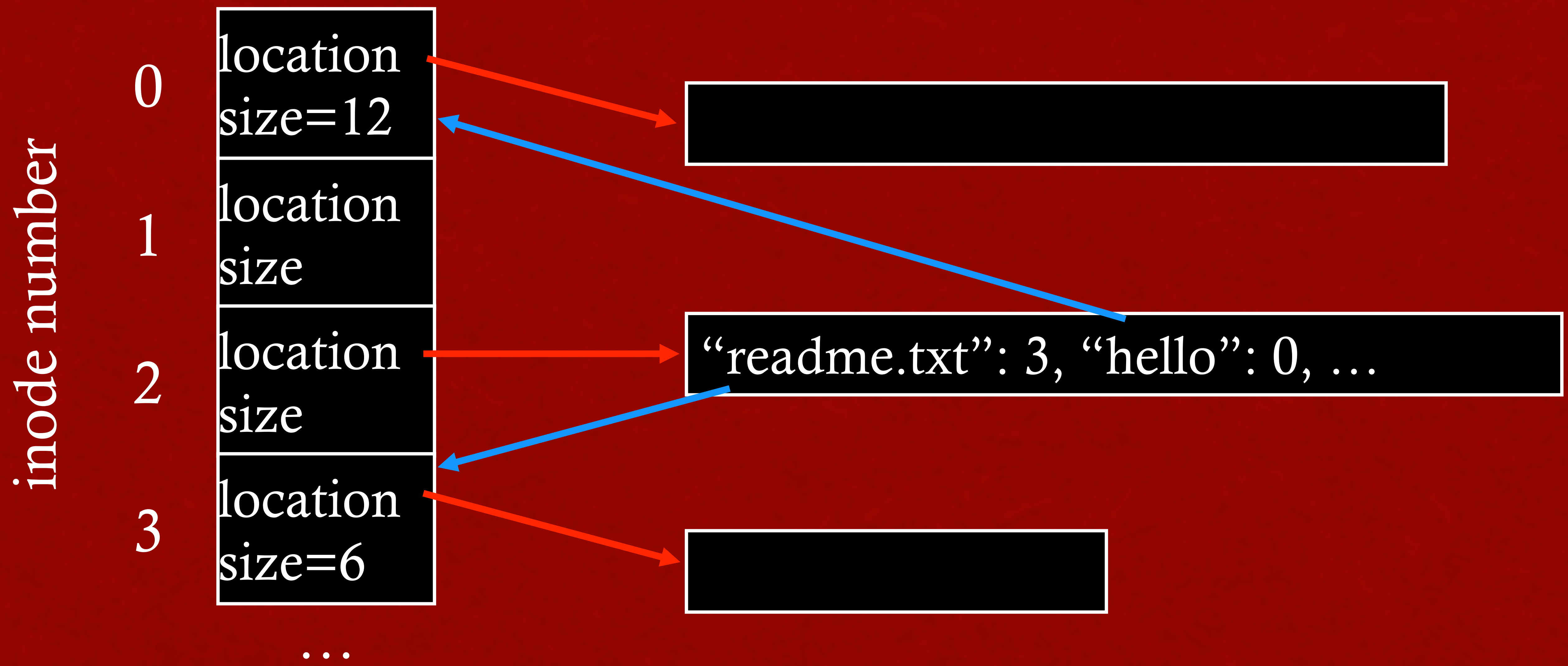
2 location size

3 location size=6

…

“readme.txt”: 3, “hello”: 0, …

inodes



inode number

0 location size=12

1 location size

2 location size

3 location size=6

…

“readme.txt”: 3, “hello”: 0, …

PATHS

Generalize!

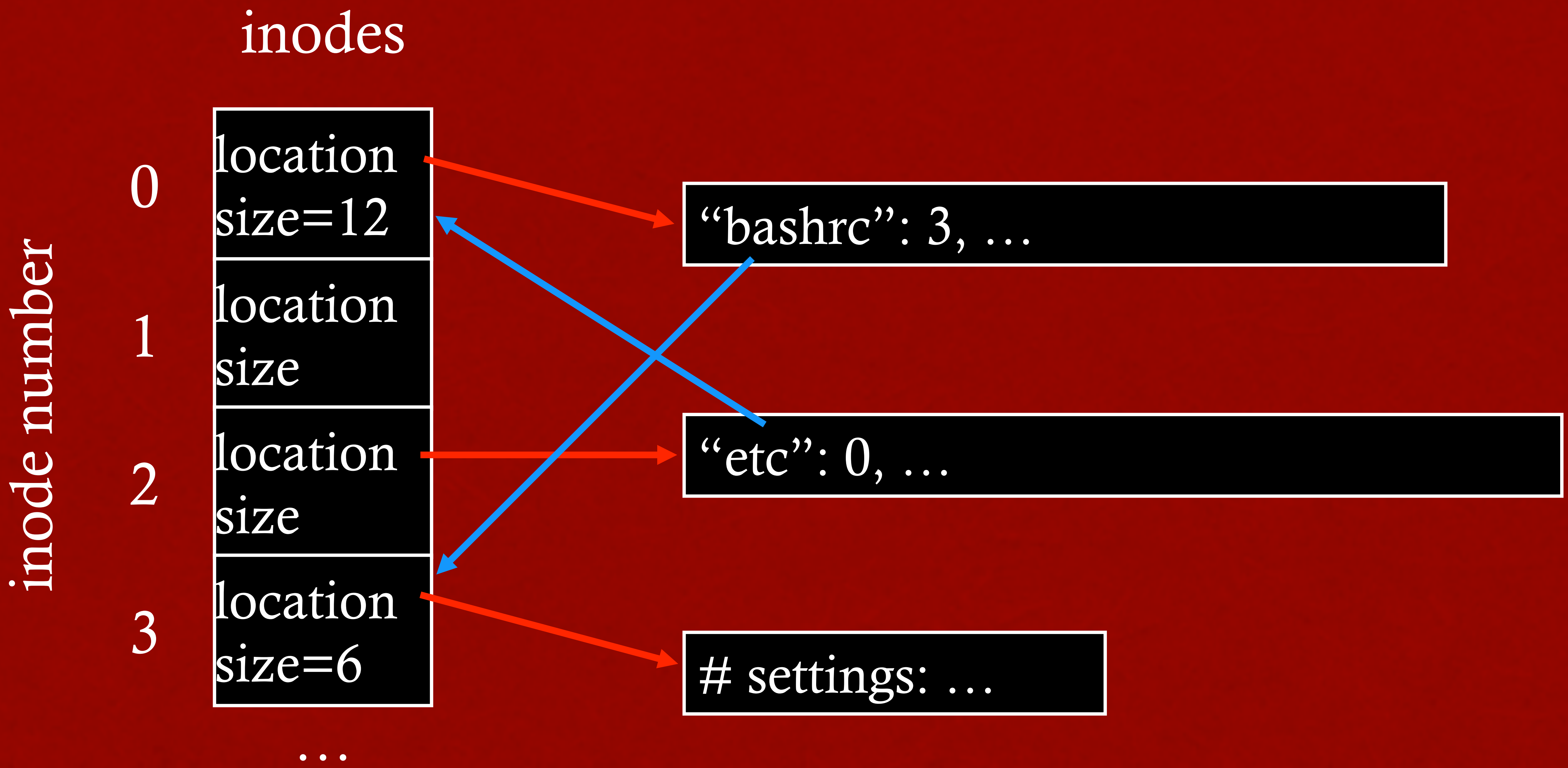
Directory Tree instead of single root directory

Only **file name** needs to be unique

/usr/dusseau/file.txt

/tmp/file.txt

Store file-to-inode mapping for each directory



inodes

inode number

0 location size=12

1 location size

2 location size

3 location size=6

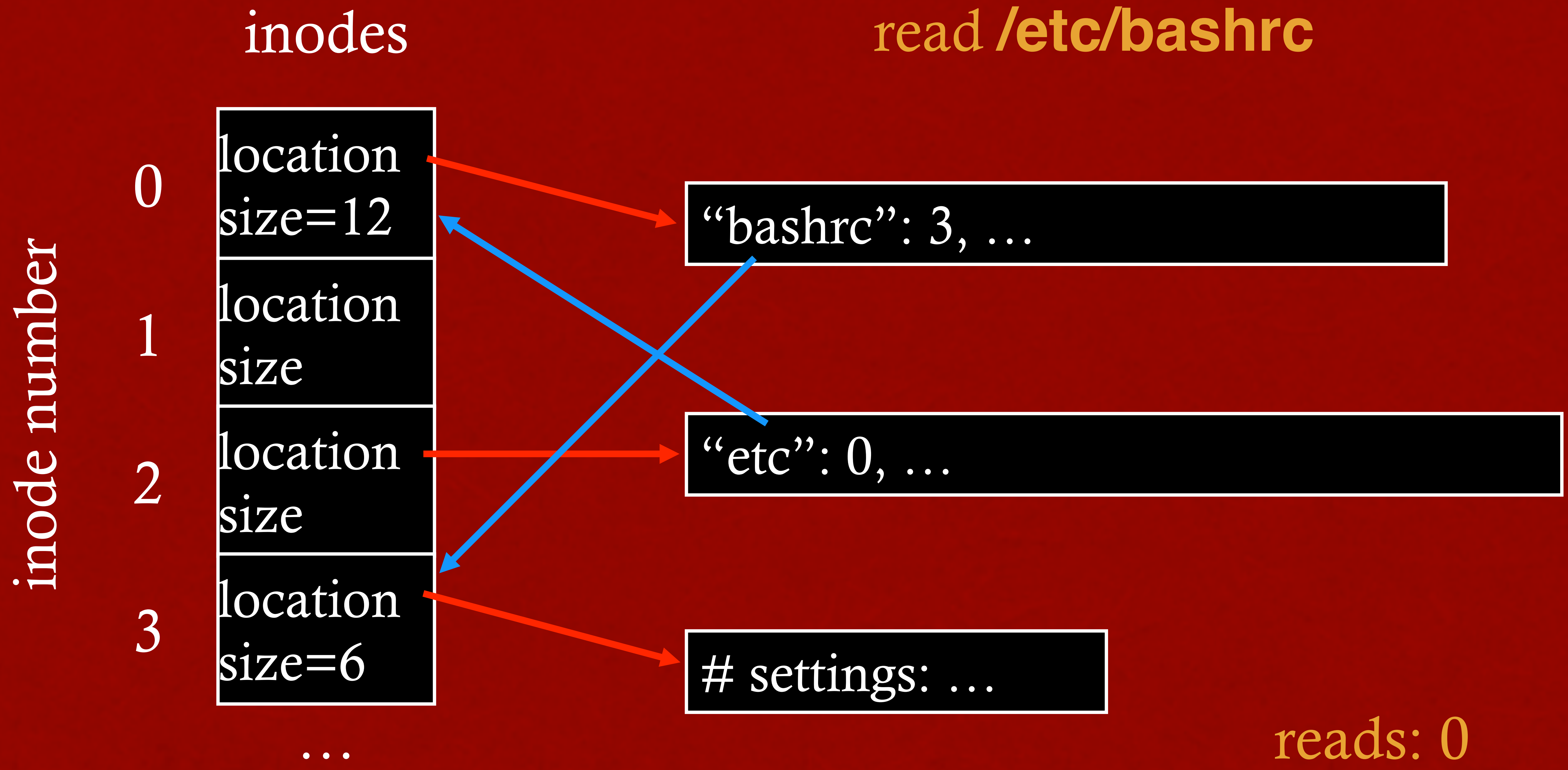
…

"bashrc": 3, …

"etc": 0, …

# settings: …

read **/etc/bashrc**



inodes

inode number

0
location
size=12

1
location
size

2
location
size

3
location
size=6

…

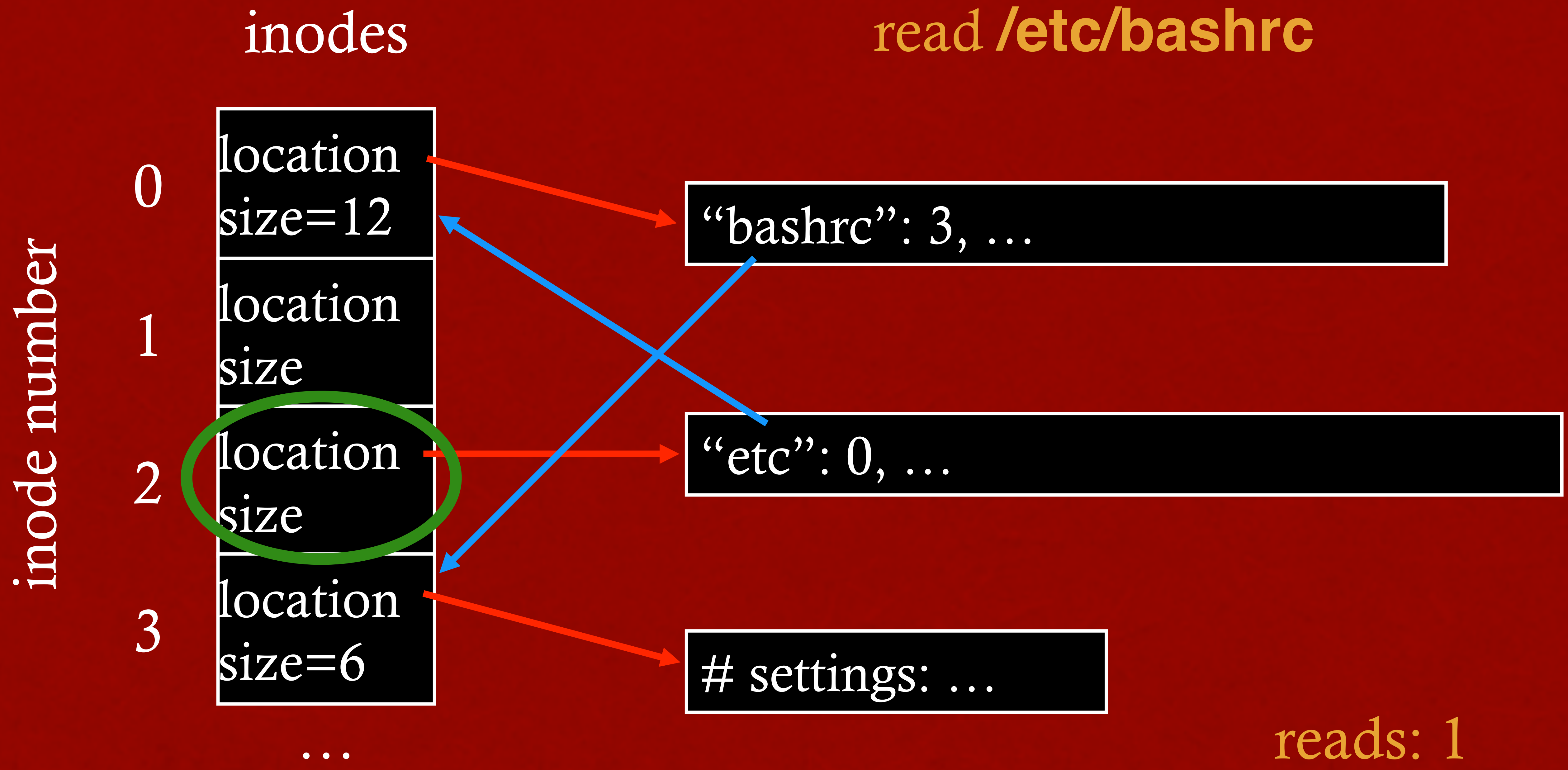
“bashrc”: 3, …

“etc”: 0, …

# settings: …

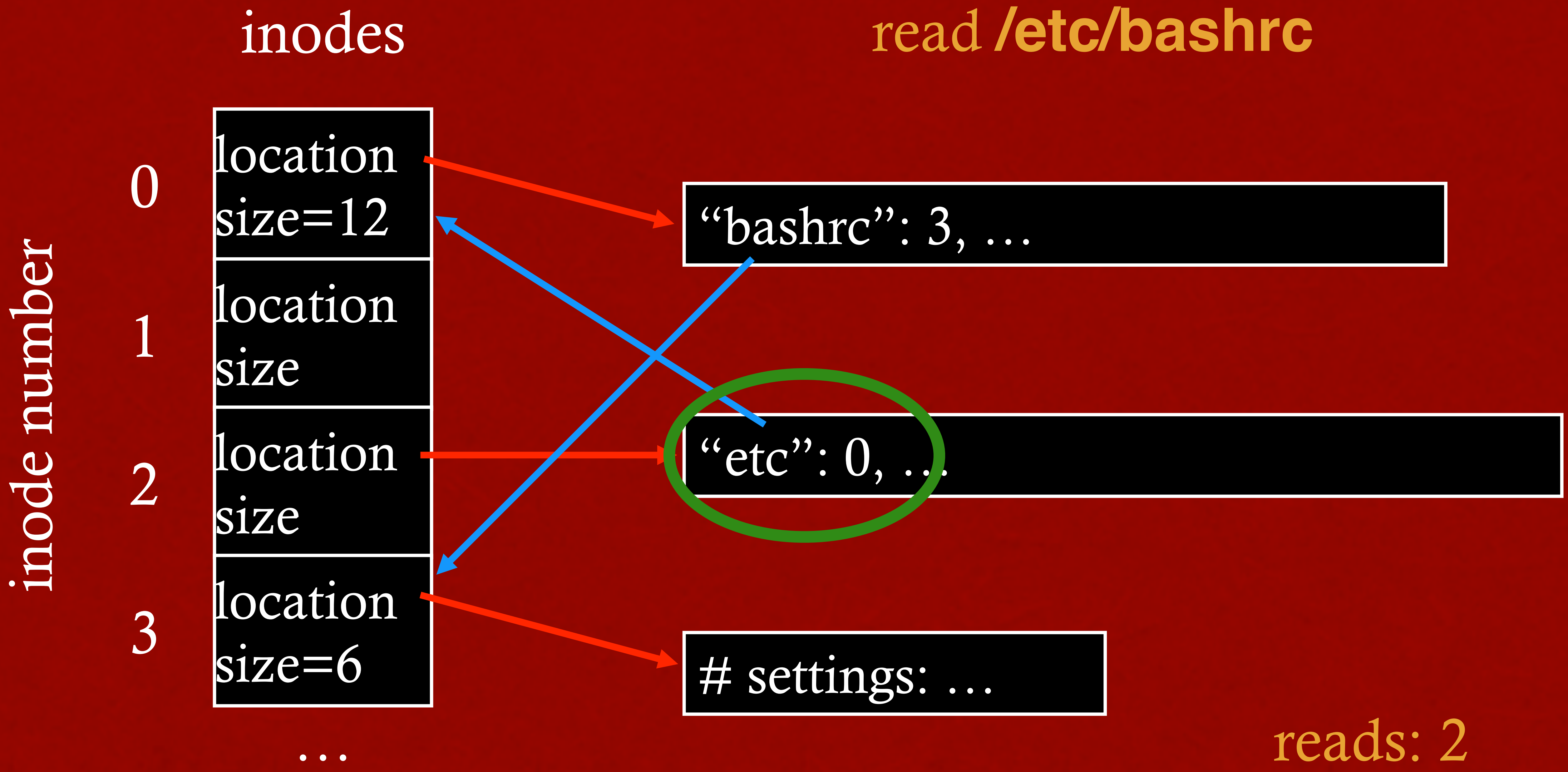
reads: 0

read **/etc/bashrc**



reads: 1

read **/etc/bashrc**



inodes

inode number

0 location size=12

1 location size

2 location size

3 location size=6

…

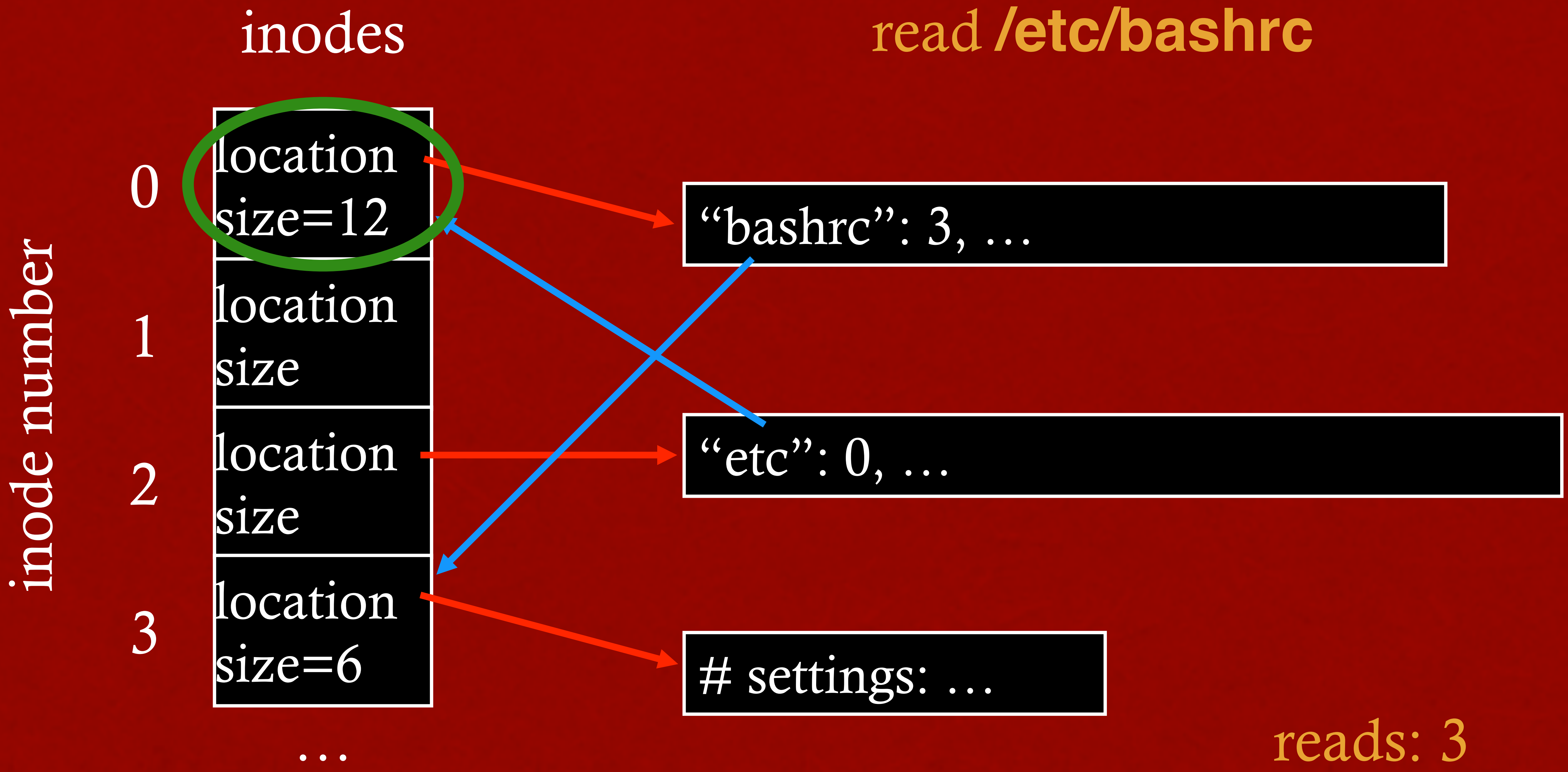
“bashrc”: 3, …

“etc”: 0, …

# settings: …

reads: 2

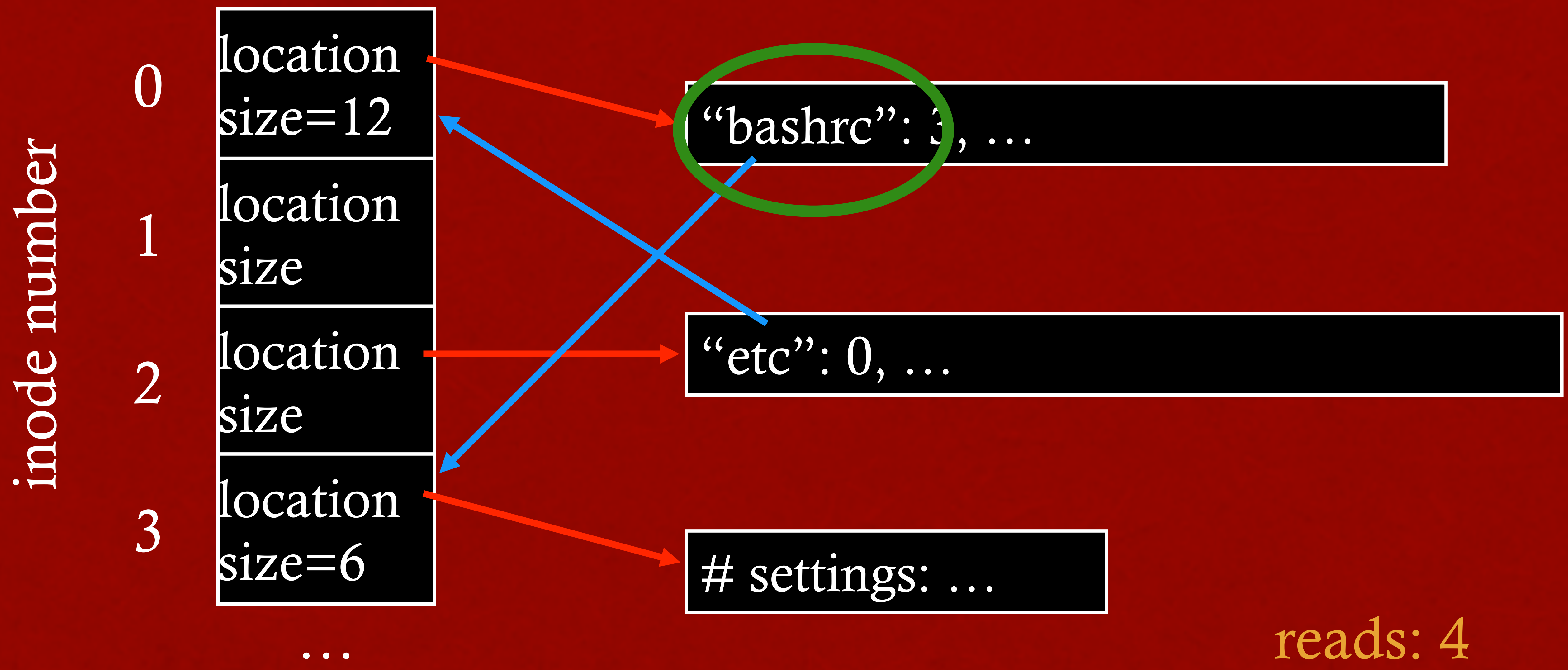
read **/etc/bashrc**



reads: 3

read **/etc/bashrc**

inodes



inode number

0 location size=12

1 location size

2 location size

3 location size=6

…

“bashrc”: 3, …

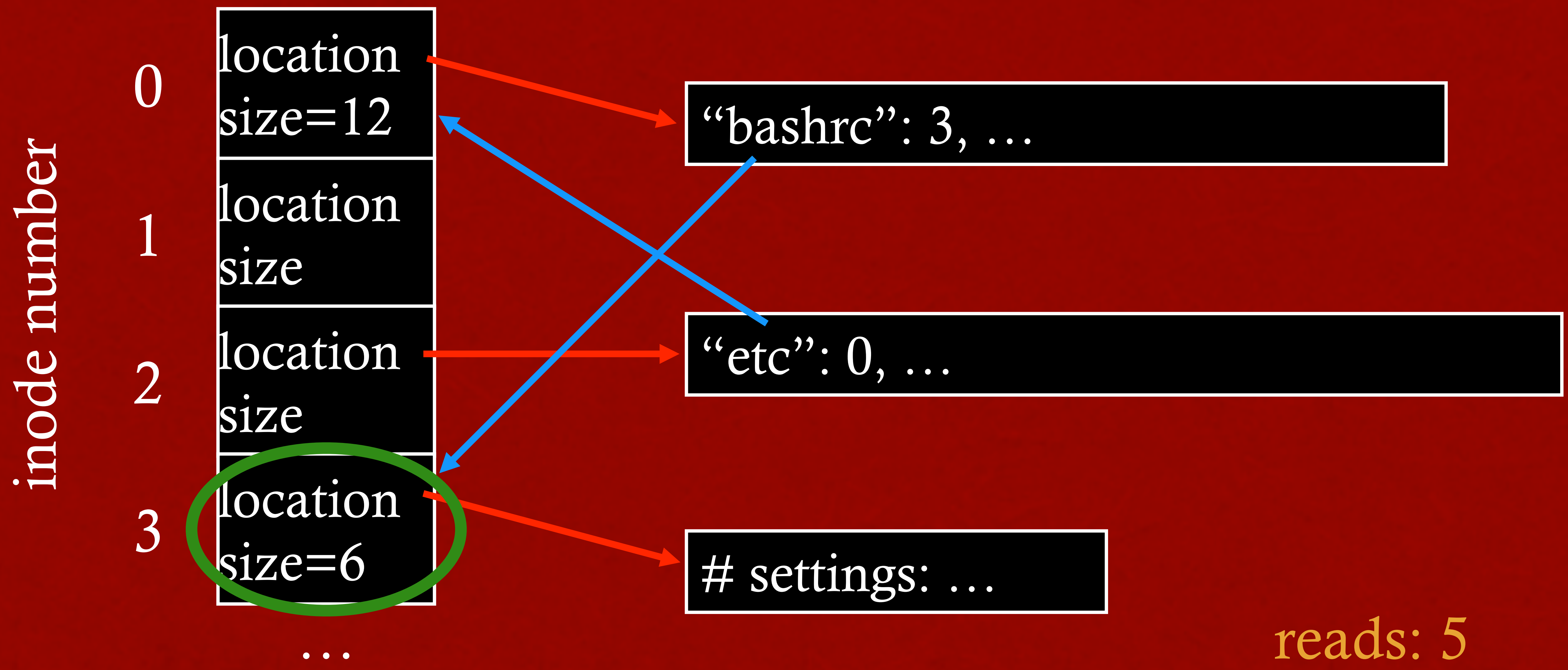
“etc”: 0, …

# settings: …

reads: 4

read **/etc/bashrc**

inodes



inode number

0	location size=12
1	location size
2	location size
3	location size=6

…

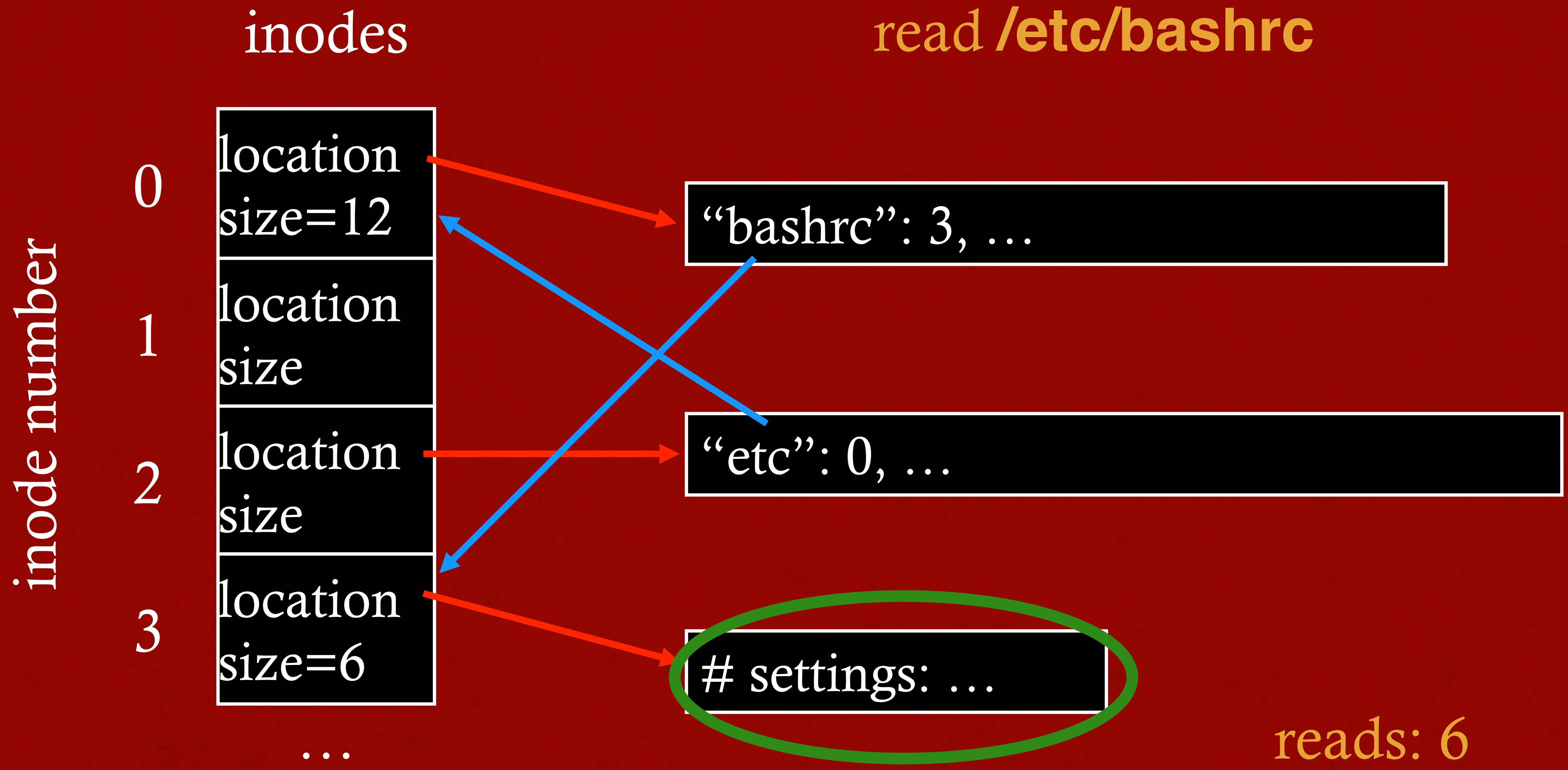
“bashrc”: 3, …

“etc”: 0, …

# settings: …

reads: 5

read **/etc/bashrc**



reads: 6

Reads for getting final inode called "traversal"

Read root dir (inode and data);
read etc dir (inode and data);
read bashrc file (indode and data)

Directory Calls

mkdir: create new directory

readdir: read/parse directory entries

Why no writedir?

Special Directory Entries

```
$ ls -la

total 728

drwxr-xr-x   34 trh   staff     1156 Oct 19 11:41 .

drwxr-xr-x+ 59 trh   staff     2006 Oct  8 15:49 ..

-rw-r--r--@  1 trh   staff     6148 Oct 19 11:42 .DS_Store

-rw-r--r--   1 trh   staff      553 Oct  2 14:29 asdf.txt

-rw-r--r--   1 trh   staff      553 Oct  2 14:05 asdf.txt~

drwxr-xr-x   4 trh   staff      136 Jun 18 15:37 backup

…
```

cd /; ls -lia

FILE API (ATTEMPT 2)

```
pread(char *path, void *buf,
      off_t offset, size_t nbyte)


pwrite(char *path, void *buf,
       off_t offset, size_t nbyte)
```

Disadvantages?

Expensive traversal!

Goal: traverse once

File Names

Three types of names:

- inode
- path
- file descriptor

File Descriptor (FD)

Idea:

- Do expensive traversal once (open file)
  store inode in descriptor object (kept in memory).
- Do reads/writes via descriptor, which tracks offset

Each process:

- File-descriptor table contains pointers to open file descriptors

Integers used for file I/O are indexes into this table

- stdin: 0, stdout: 1, stderr: 2

FD TABLE (XV6)

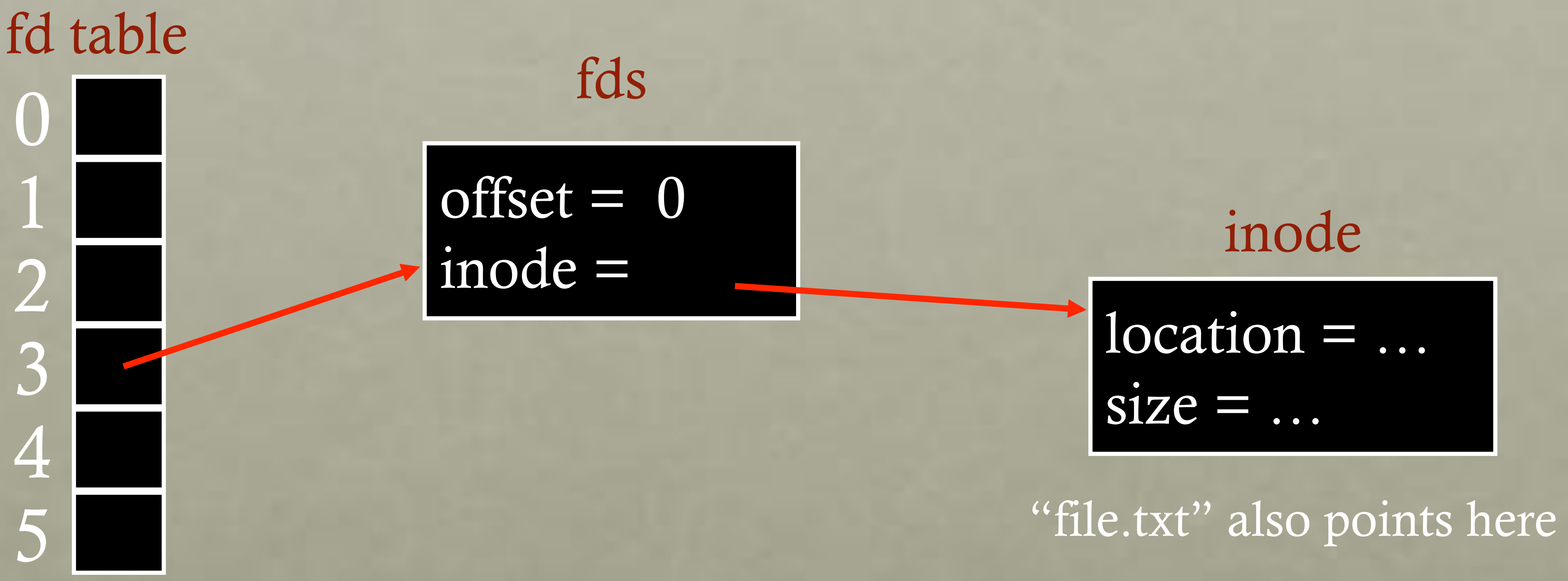
```
struct file {
  ...
  struct inode *ip;
  uint off;
};

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

Code Snippet

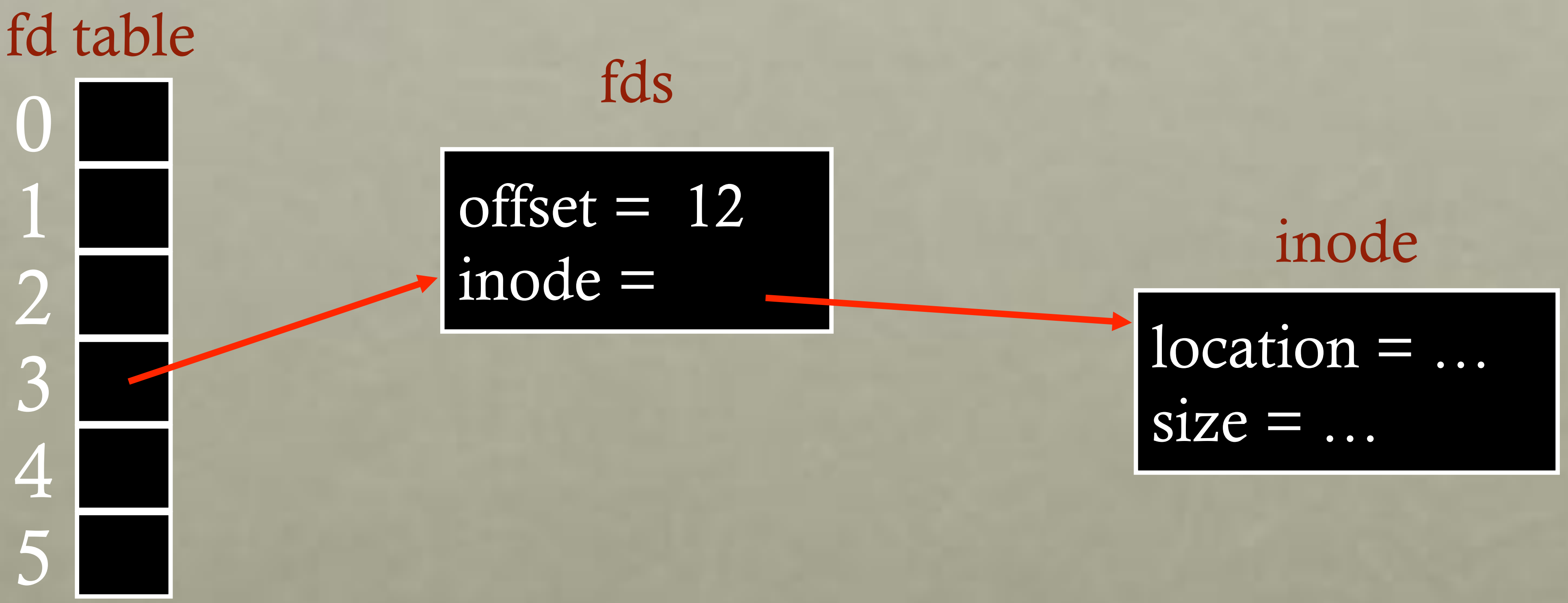
```
int fd1 = open(“file.txt”); // returns 3
read(fd1, buf, 12);
int fd2 = open(“file.txt”); // returns 4
int fd3 = dup(fd2);         // returns 5
```

CODE SNIPPET



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

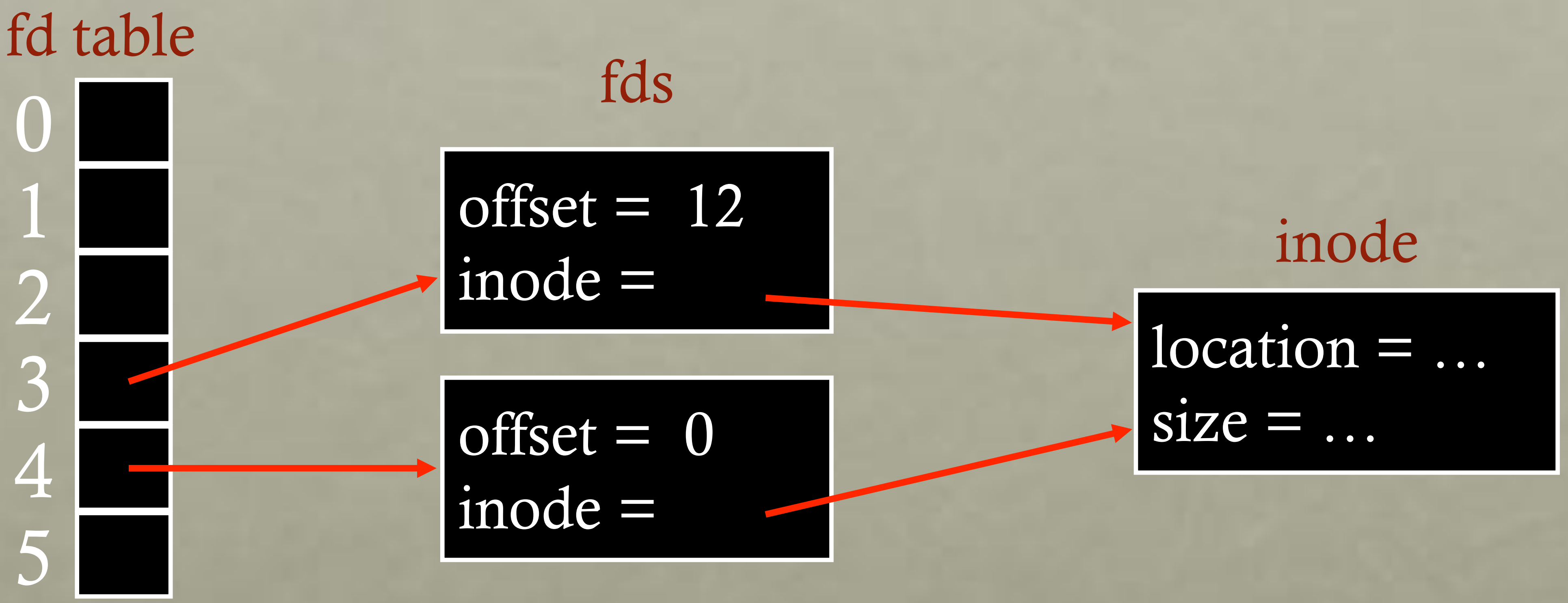
CODE SNIPPET



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

read(fd1, buf, 12);
```

Code Snippet

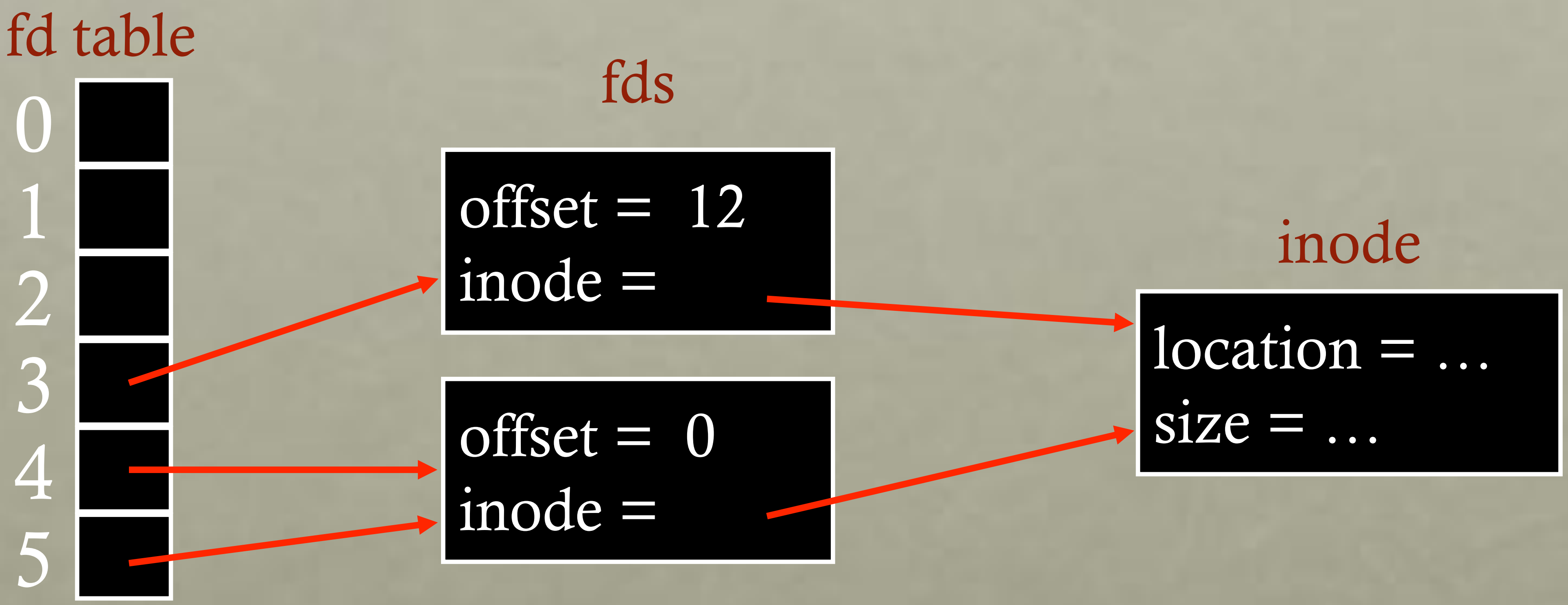


```
int fd1 = open(“file.txt”); // returns 3

read(fd1, buf, 12);

int fd2 = open(“file.txt”); // returns 4
```

CODE SNIPPET



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

FILE API (ATTEMPT 3)

```
int fd = open(char *path, int flag, mode_t mode)

read(int fd, void *buf, size_t nbyte)

write(int fd, void *buf, size_t nbyte)

close(int fd)
```

advantages:

- string names
- hierarchical
- traverse once
- different offsets precisely defined

Deleting Files

There is no system call for deleting files!

Inode (and associated file) is **garbage collected** when there are no references (from paths or fds)

Paths are deleted when: `unlink()` is called

FDs are deleted when: `close()` or process quits

Network File System Designers

A process can open a file, then remove the directory entry for the file so that it has no name anywhere in the file system, and still read and write the file. This is a disgusting bit of UNIX trivia and at first we were just not going to support it, but it turns out that all of the programs we didn't want to have to fix (csh, sendmail, etc.) use this for temporary files.

~ Sandberg *et al.*

Links: Demonstrate

- Show hard links: Both path names use same inode number
  - File does not disappear until all removed; cannot link directories
  - `Echo “Beginning…” > file1`
  - `“ln file1 link”`
  - `“cat link”`
  - `“ls -li” to see reference count`
  - `Echo “More info…” >> file1`
  - `“mv file1 file2”`
  - `“rm file2” decreases reference count`
- Soft or symbolic links: Point to second path name; can softlink to dirs
  - **`“ln –s oldfile softlink”`**
  - Confusing behavior: “file does not exist”!
  - Confusing behavior: “cd linked_dir; cd ..; in different parent!

Many File Systems

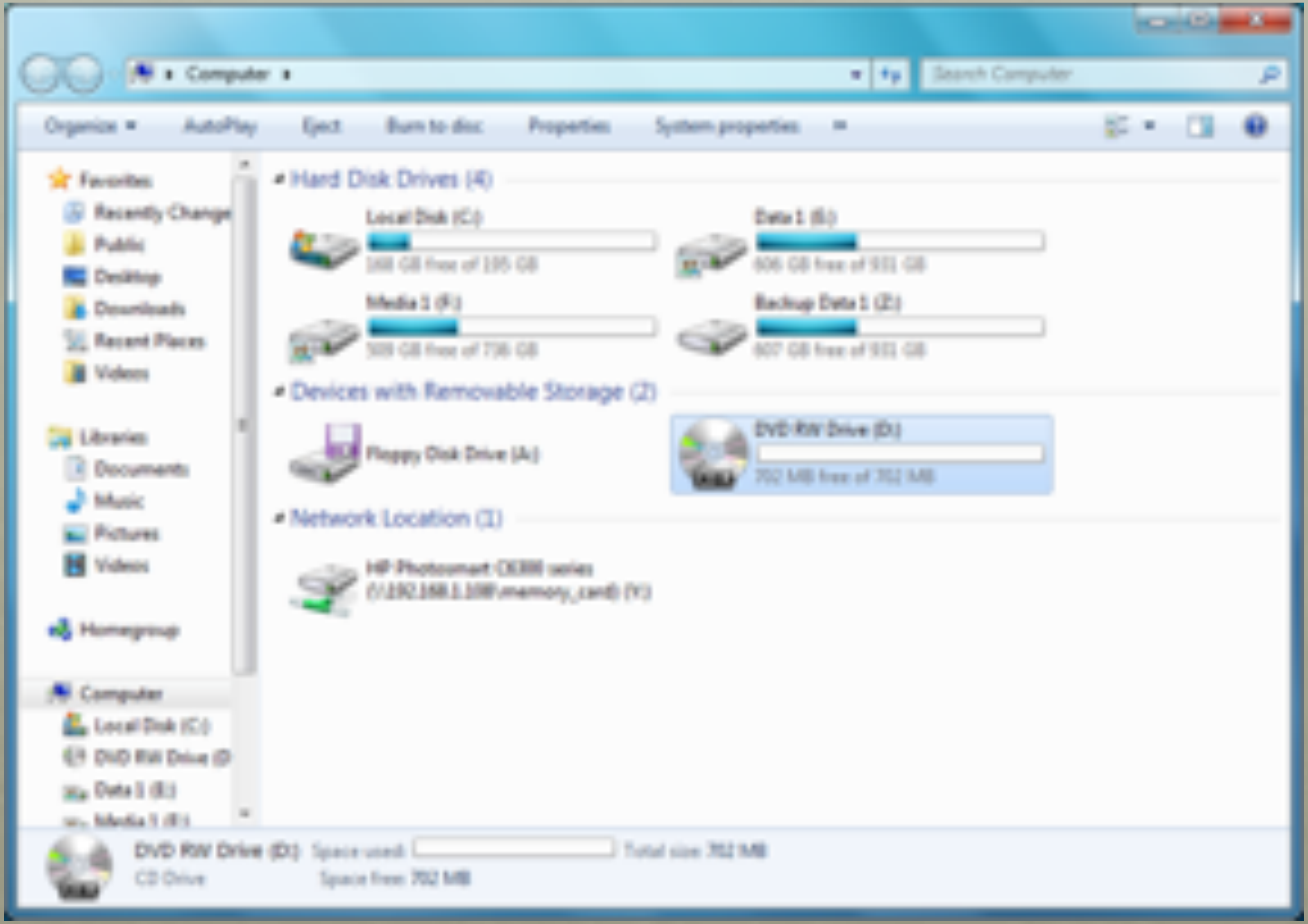
Users often want to use many file systems

For example:

- main disk
- backup disk
- AFS
- thumb drives

What is the most elegant way to support this?

Many File Systems: Approach 1



- http://www.ofzenandcomputing.com/burn-files-cd-dvd-windows7/

Many File Systems: Approach 2

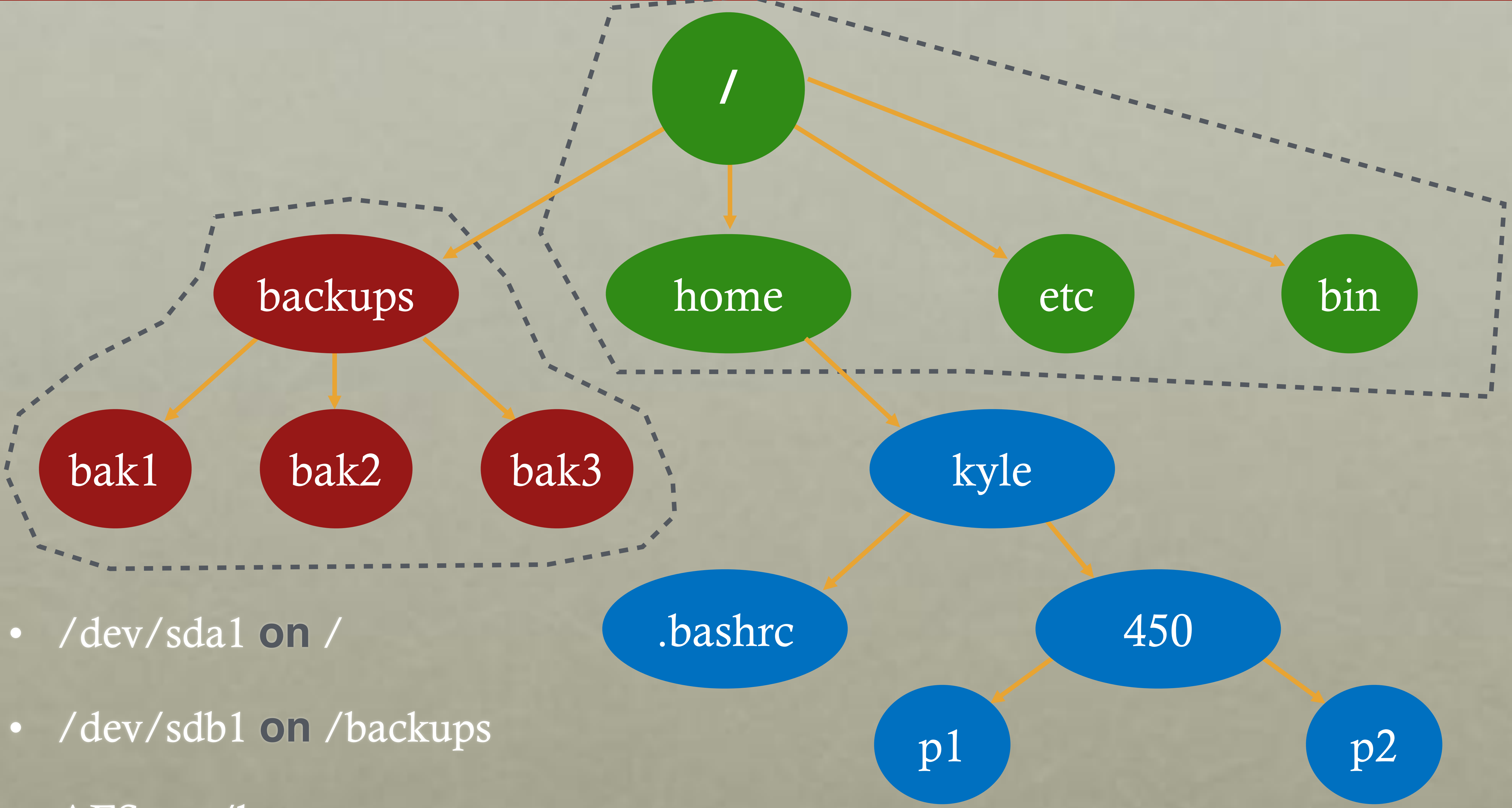
Idea: stitch all the file systems together into a super file system!

```
sh> mount

/dev/sda1 on / type ext4 (rw)

/dev/sdb1 on /backups type ext4 (rw)

AFS on /home type afs (rw)
```



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

Communicating Requirements: fsync

File system keeps newly written data in memory for awhile

Write buffering improves performance (why?)

But what if system **crashes** before buffers are flushed?

If application cares:

`fsync(int fd)` forces buffers to flush to disk, and (usually) tells disk to flush its write cache too

Makes data **durable**

RENAME

rename(char *old, char *new):

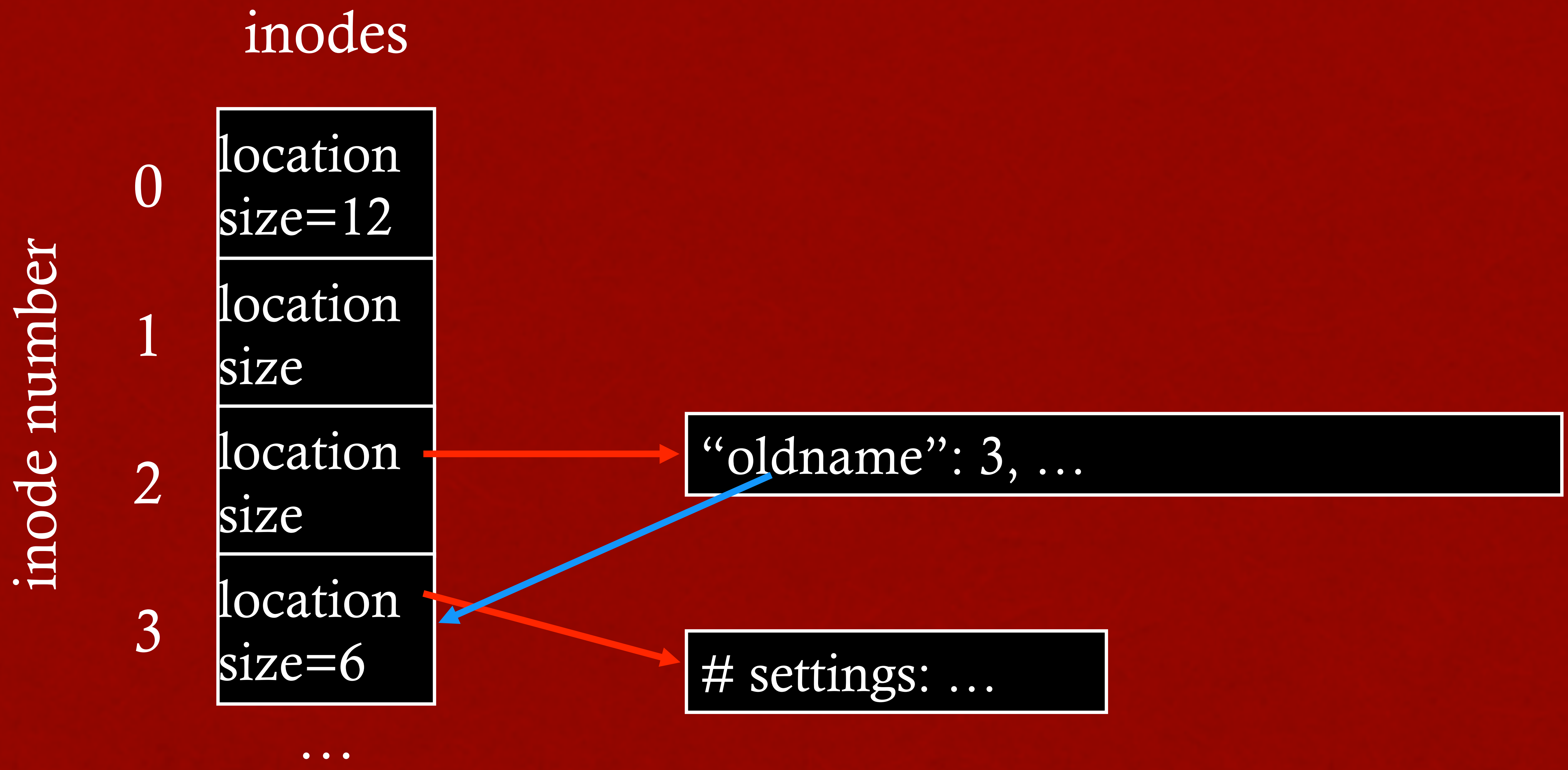
- deletes an old link to a file

- creates a new link to a file

Just changes name of file, does not move data

Even when renaming to new directory (unless…?)

What can go wrong if system crashes at wrong time?



inodes

inode number

0
location
size=12

1
location
size

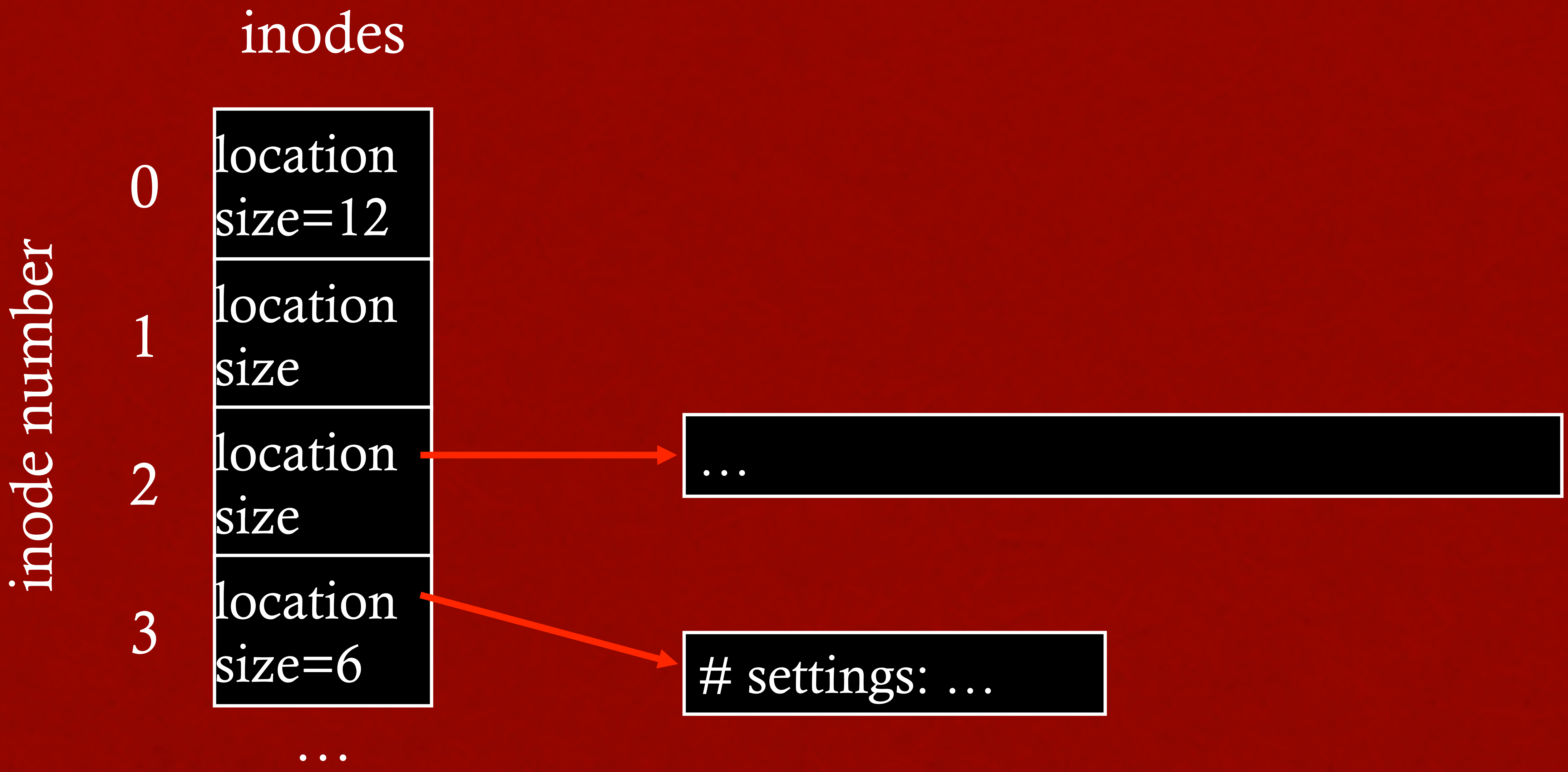
2
location
size

3
location
size=6

…

“oldname”: 3, …

# settings: …

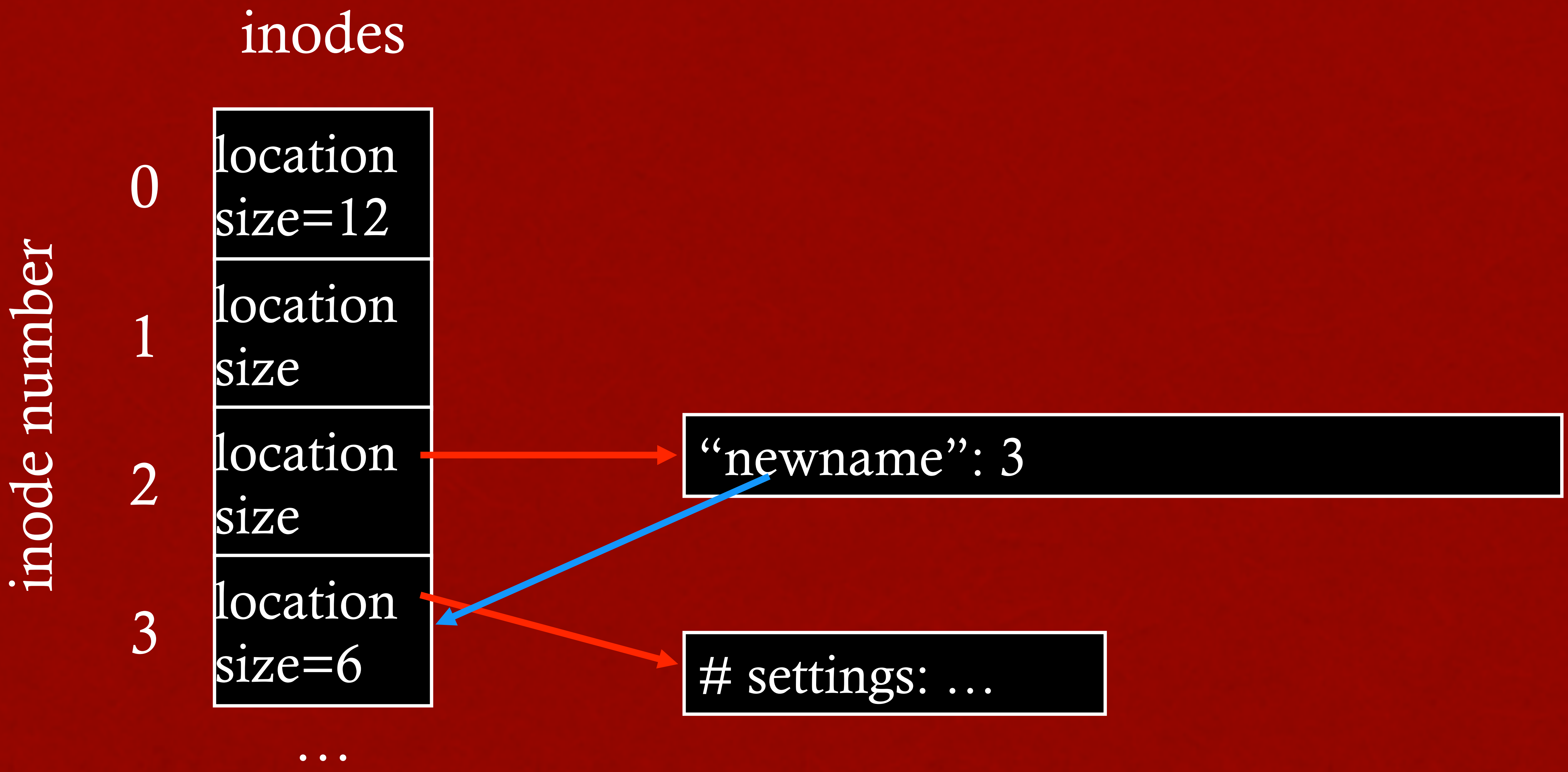


inodes

inode number	inodes
0	location size=12
1	location size
2	location size
3	location size=6
	…

…

# settings: …



inodes

inode number

0	location size=12
1	location size
2	location size
3	location size=6
…	

"newname": 3

# settings: …

RENAME

rename(char *old, char *new):

- deletes an old link to a file

- creates a new link to a file

What if we crash?

FS does extra work to guarantee atomicity; return to this issue later…

Atomic File Update

Say application wants to update `file.txt` atomically

- If crash, should see only old contents or only new contents

1. write new data to `file.txt.tmp` file
2. `fsync file.txt.tmp`
3. `rename file.txt.tmp` over `file.txt`, replacing it

SUMMARY

Using multiple types of name provides

- convenience

- efficiency

Mount and link features provide flexibility.

Special calls (fsync, rename) let developers communicate special requirements to file system