

More On Naming

➡ (Almost) everything has a path name

- files
- directories
- devices (*known as **special files***)
 - keyboards, displays, disks, etc.

➡ Uniformity

```
// opening a normal file
```

```
int file = open("/home/bc/data", O_RDWR);
```

```
// opening a device (one's terminal or window)
```

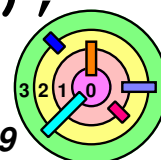
```
int device = open("/dev/tty", O_RDWR);
```

```
int bytes = read(file, buffer, sizeof(buffer));  
write(device, buffer, bytes);
```

— in warmup1, we saw that you can open a directory

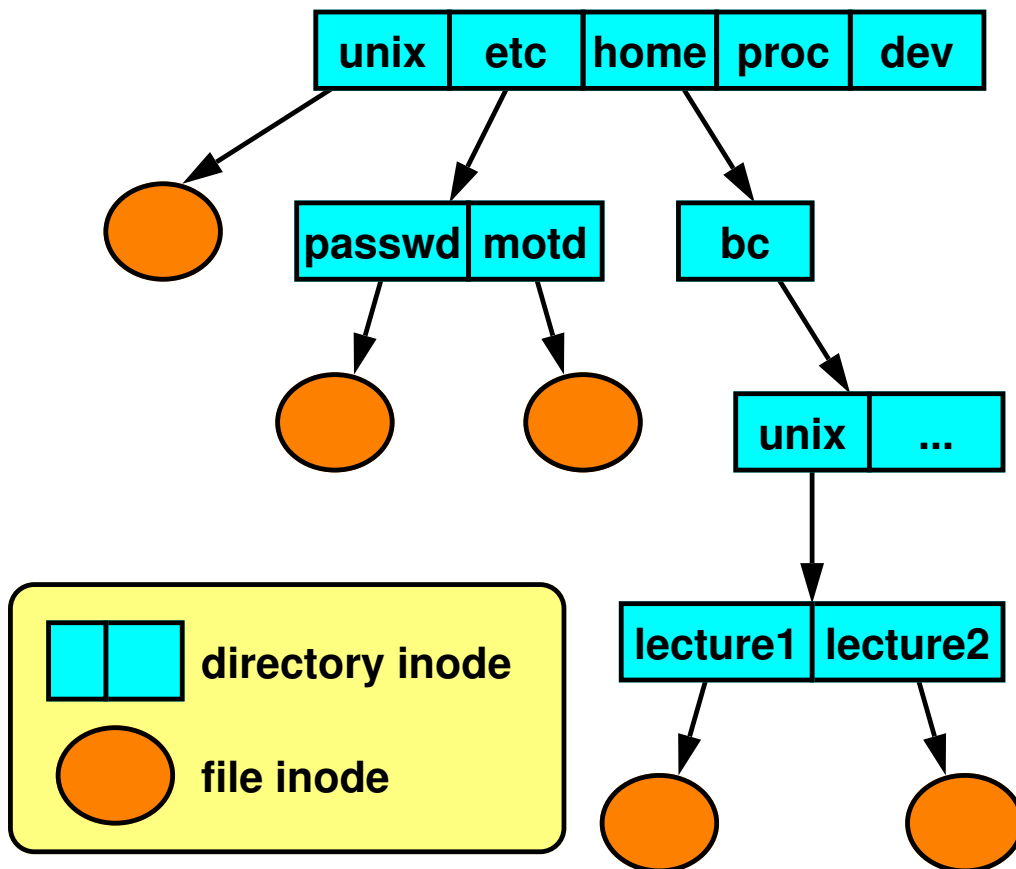
```
read(open("/etc", O_RDONLY), buf, sizeof(buf));
```

- although *reading* from a *directory* is *not standardized*

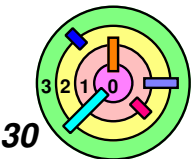


Directories

- ➡ A **directory** is a **file**
- interprets differently by the OS as containing **references** to other files/directories
 - a file is represented as an **index node** (or **inode**) in the file system

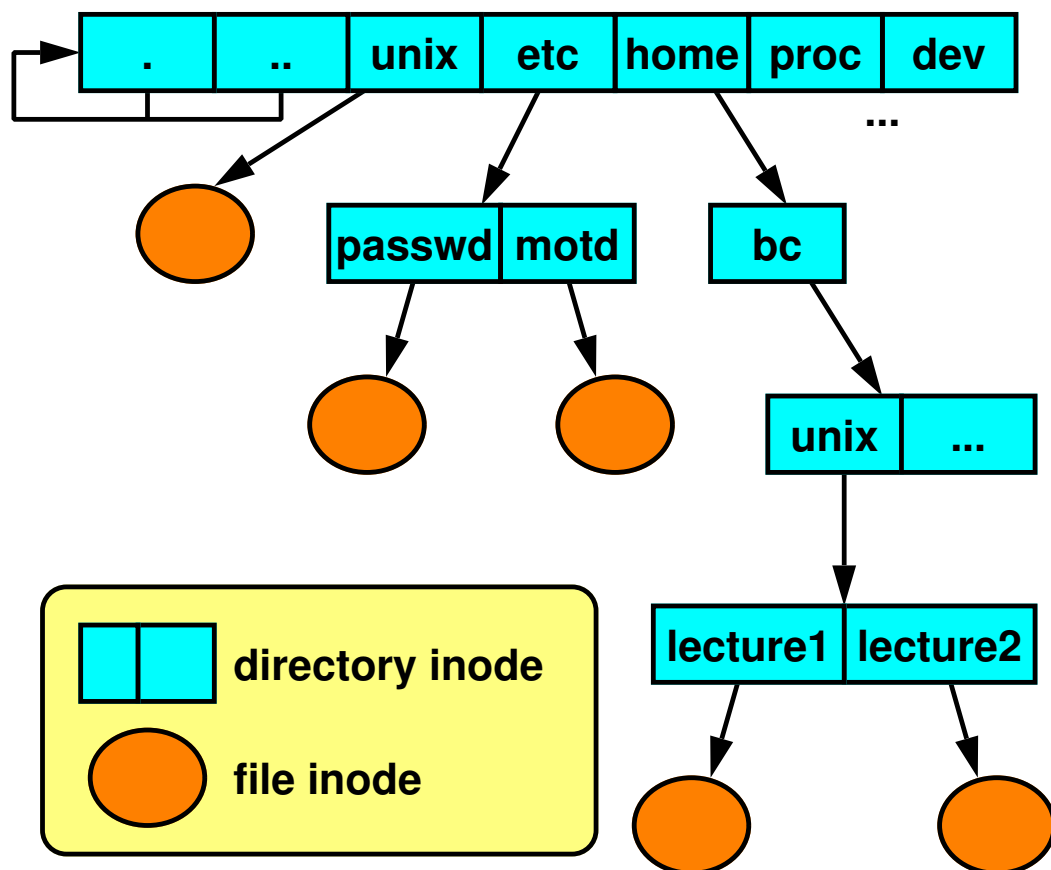


- ➡ A **directory** **maps** a **file name** to an **inode number**
- maps a string to an integer
 - done inside Virtual File System in `weenix`
- ➡ An **inode** maps an **inode number** to **disk address**
- done inside Actual File System in `weenix`



Directory Representation

- ➡ A root directory entry example
- parent inode number = its own inode number



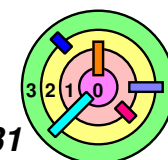
Component Name	Inode number
----------------	--------------

directory entry

.	1
..	1
unix	117
etc	4
home	18
proc	36
dev	93

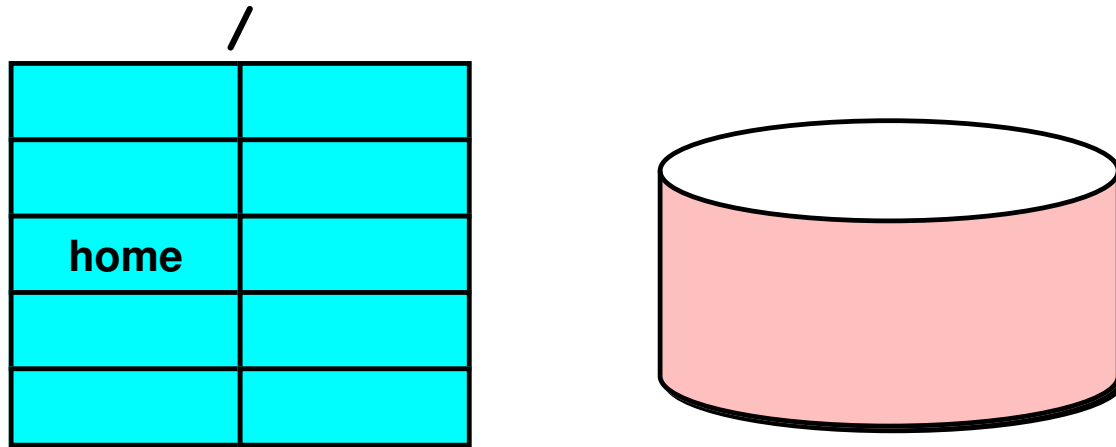
↑ this is what a S5FS (and RAMFS) directory looks like in weenix

- ➡ Tree structured hierarchy



Look Up Inode Number Of A Path

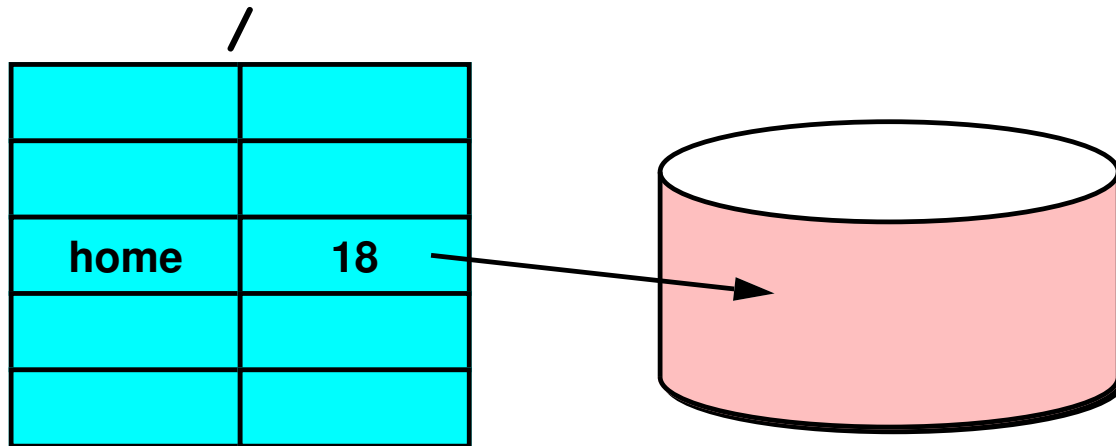
- ➡ Ex: how do figure out the inode number of `"/home/bc/foo.c"`?
➡ this process is called *pathname resolution*



- ➡ AFS creates an *inode pointer* to represent the directory file

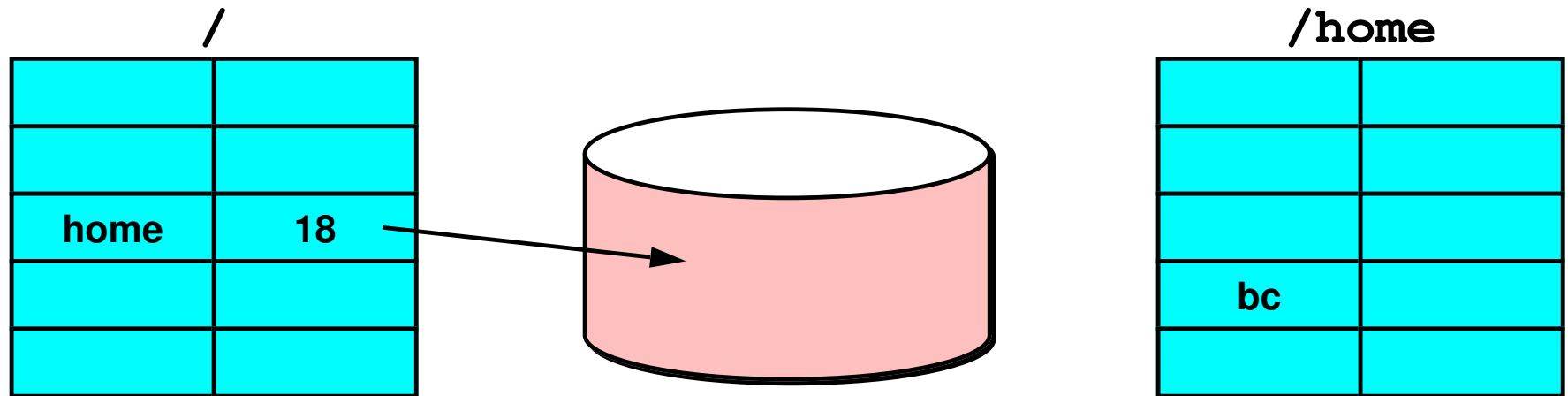
Look Up Inode Number Of A Path

- ➡ Ex: how do figure out the inode number of `"/home/bc/foo.c"`?
— this process is called *pathname resolution*



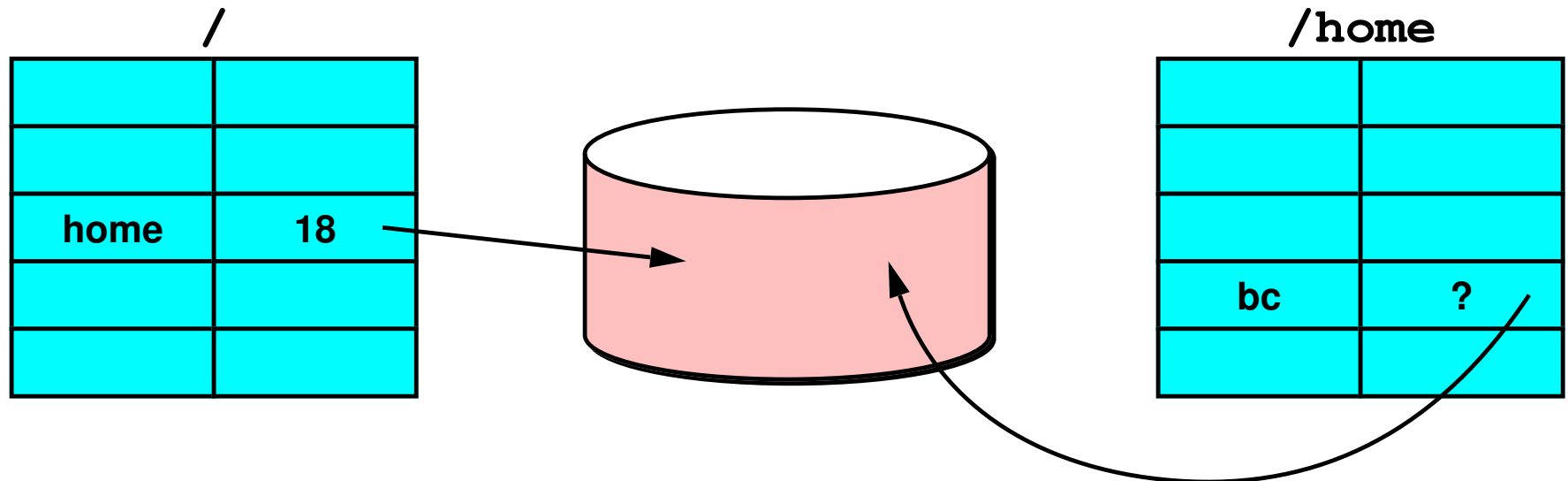
Look Up Inode Number Of A Path

- ➡ Ex: how do figure out the inode number of `"/home/bc/foo.c"`?
- ▬ this process is called *pathname resolution*



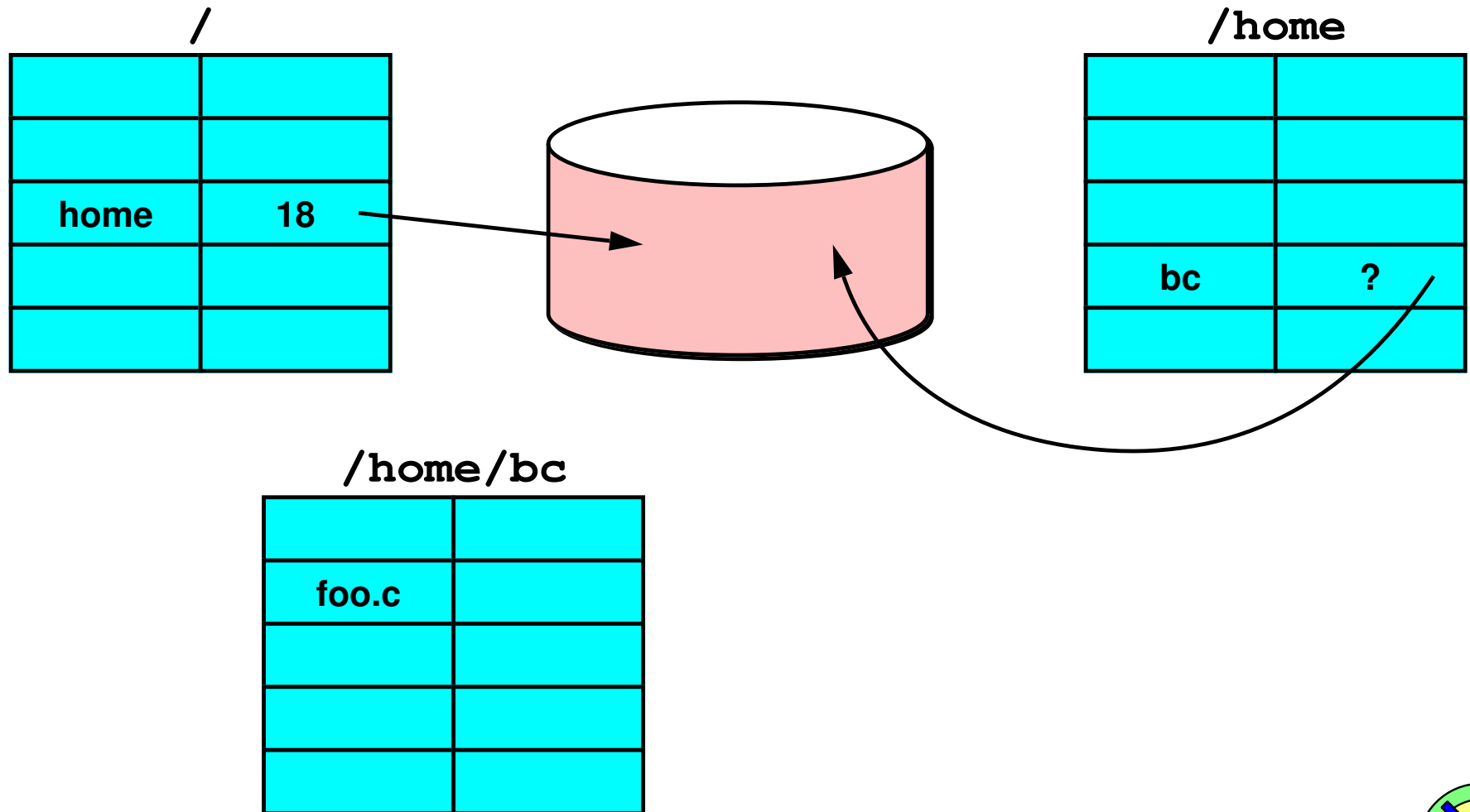
Look Up Inode Number Of A Path

- ➡ Ex: how do figure out the inode number of `"/home/bc/foo.c"`?
- ▬ this process is called *pathname resolution*



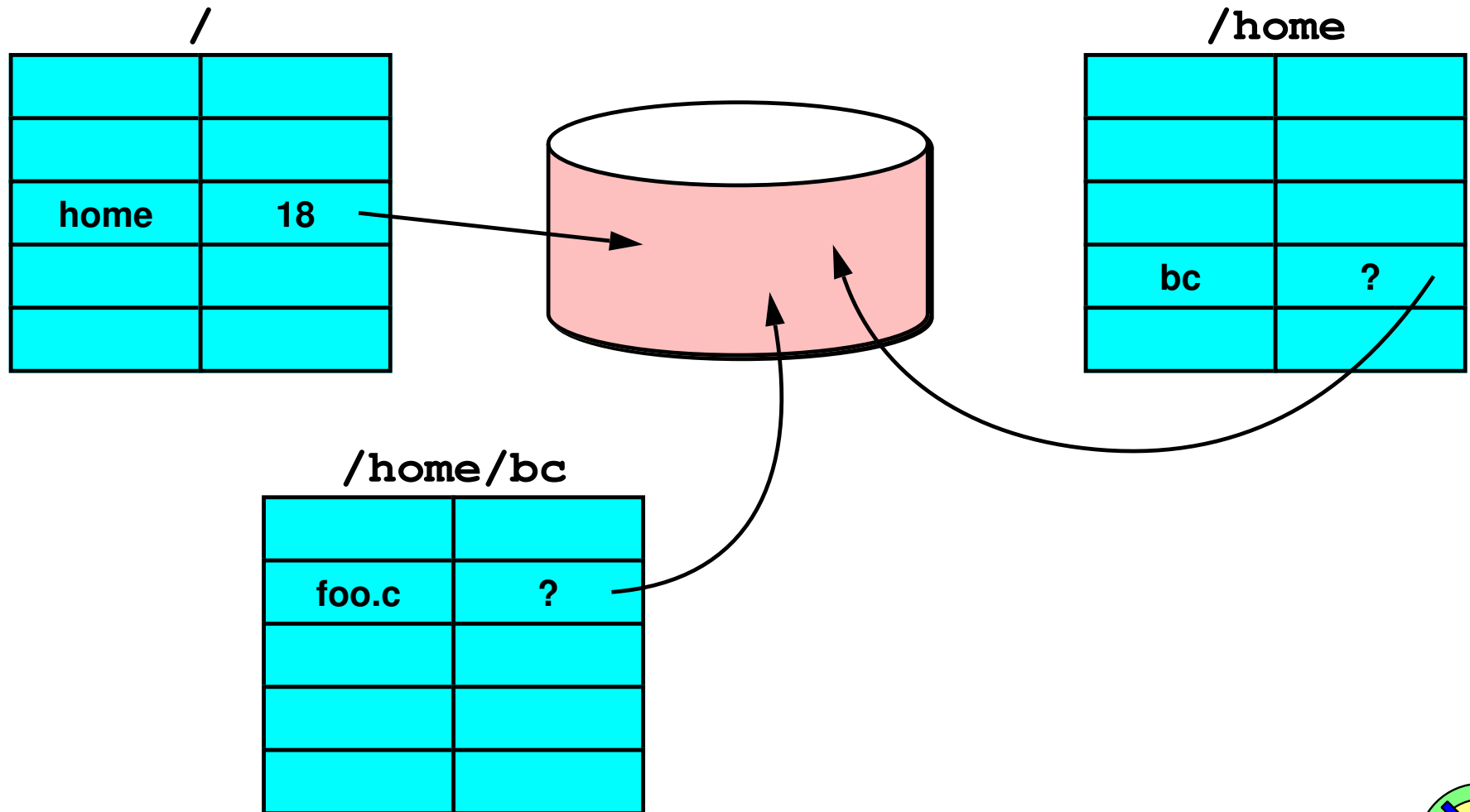
Look Up Inode Number Of A Path

- ➡ Ex: how do figure out the inode number of `"/home/bc/foo.c"`?
- ▬ this process is called *pathname resolution*



Look Up Inode Number Of A Path

- ➡ Ex: how do figure out the inode number of `"/home/bc/foo.c"`?
- ▬ this process is called *pathname resolution*



Directory Hierarchy

➡ Unix and many other OSes allow limited deviation from trees

— **hard links**

- reference to a file in one directory that also appears in another
- can use the `link()` system call or the "`ln`" shell command to **add** a hard link to a **file** (but not a directory)

— **soft links** or **symbolic links**

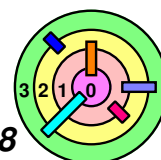
- a special kind of file containing the **name** of another file or directory
- can use the `symlink()` system call or the "`ln -s`" shell command to add a symbolic link to any type of file

➡ Why `link()` cannot be used on a directory?

— to avoid cycles

— Unix directory hierarchy can be viewed as a **directed acyclic graph** (DAG)

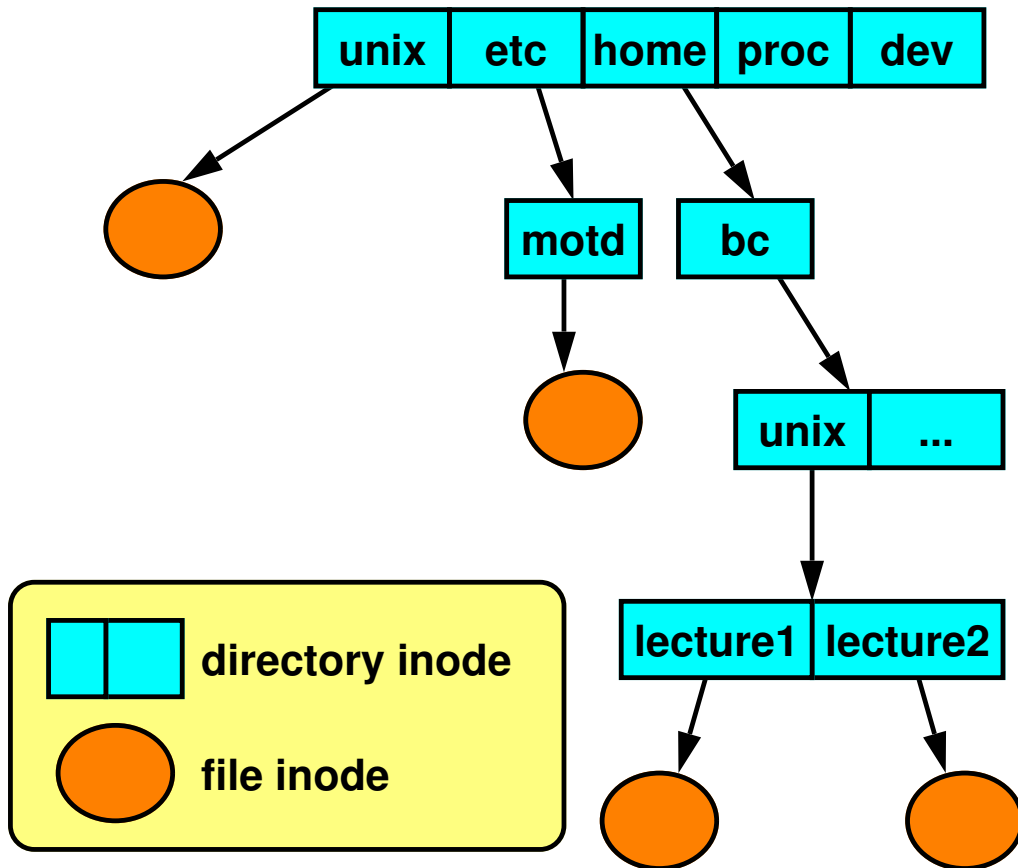
- straight-forward algorithm to traverse directory hierarchy efficiently



Hard Links

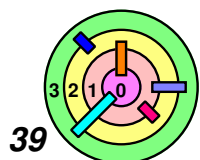
```
% ln /unix /etc/image
```

— create "image" in "/etc" to link to "/unix"



.	1
..	1
unix	117
etc	4
home	18
proc	36
dev	93

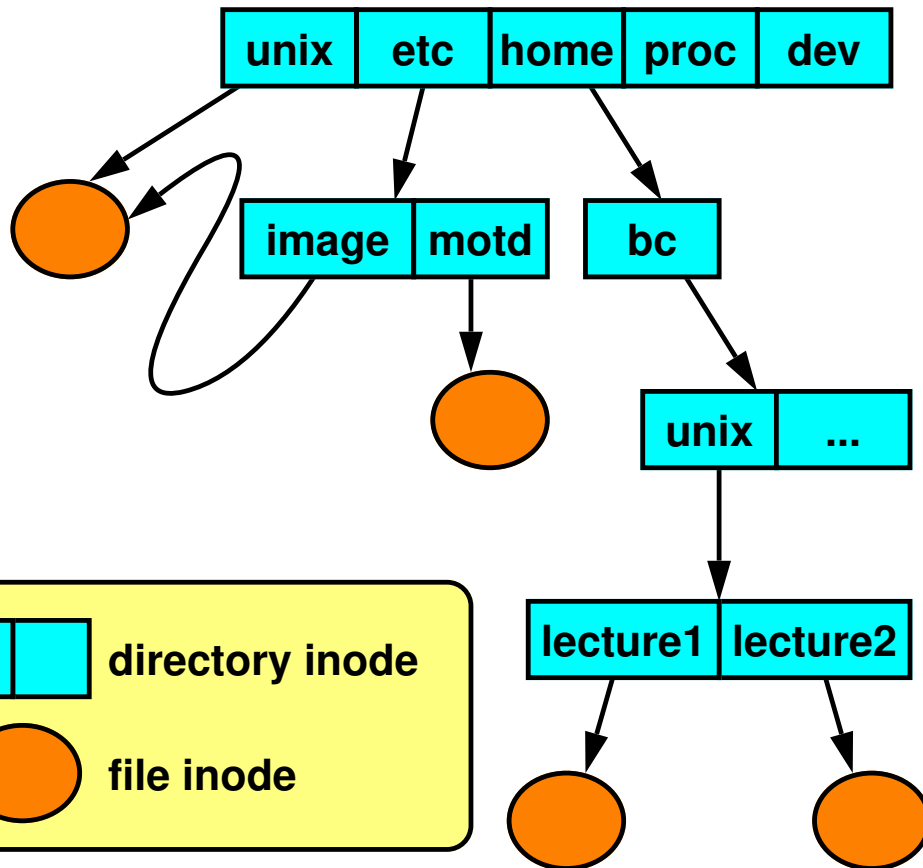
.	4
..	1
motd	33



Hard Links

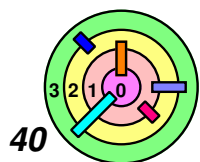
```
% ln /unix /etc/image
```

— create "image" in "/etc" to link to "/unix"



.	1
..	1
unix	117
etc	4
home	18
proc	36
dev	93

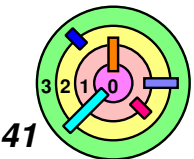
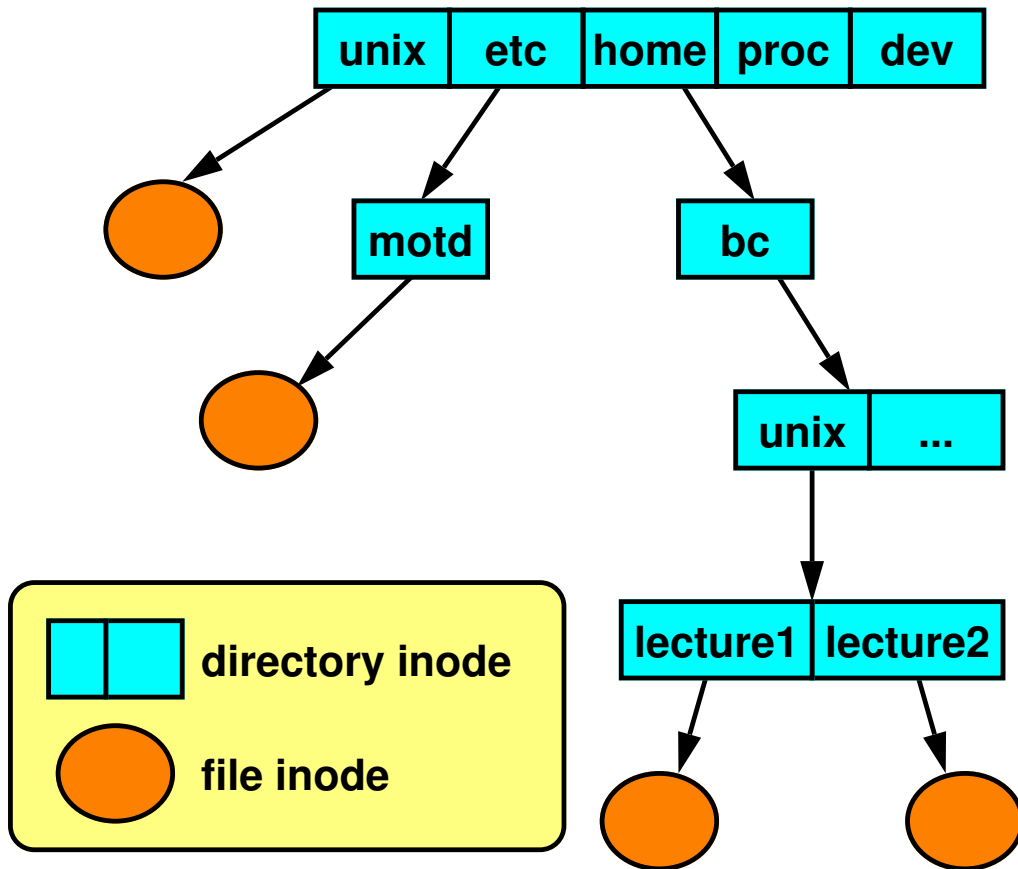
.	4
..	1
motd	33
image	117



Soft Links

```
% ln -s /unix /home/bc/mylink
```

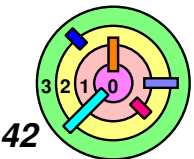
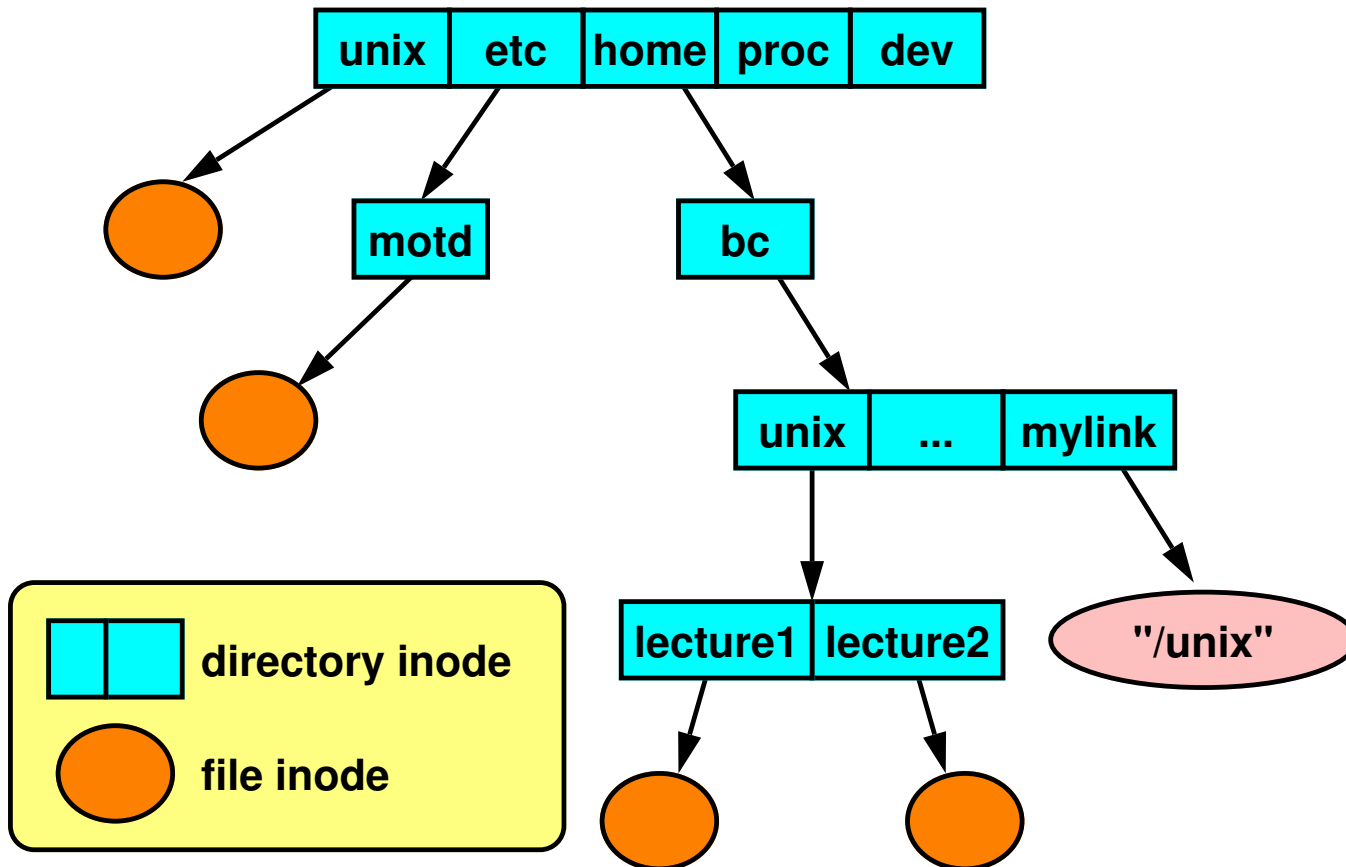
➡ create "mylink" in "/home/bc" to *soft-link* to "/unix"



Soft Links

```
% ln -s /unix /home/bc/mylink
```

➡ create "mylink" in "/home/bc" to *soft-link* to "/unix"

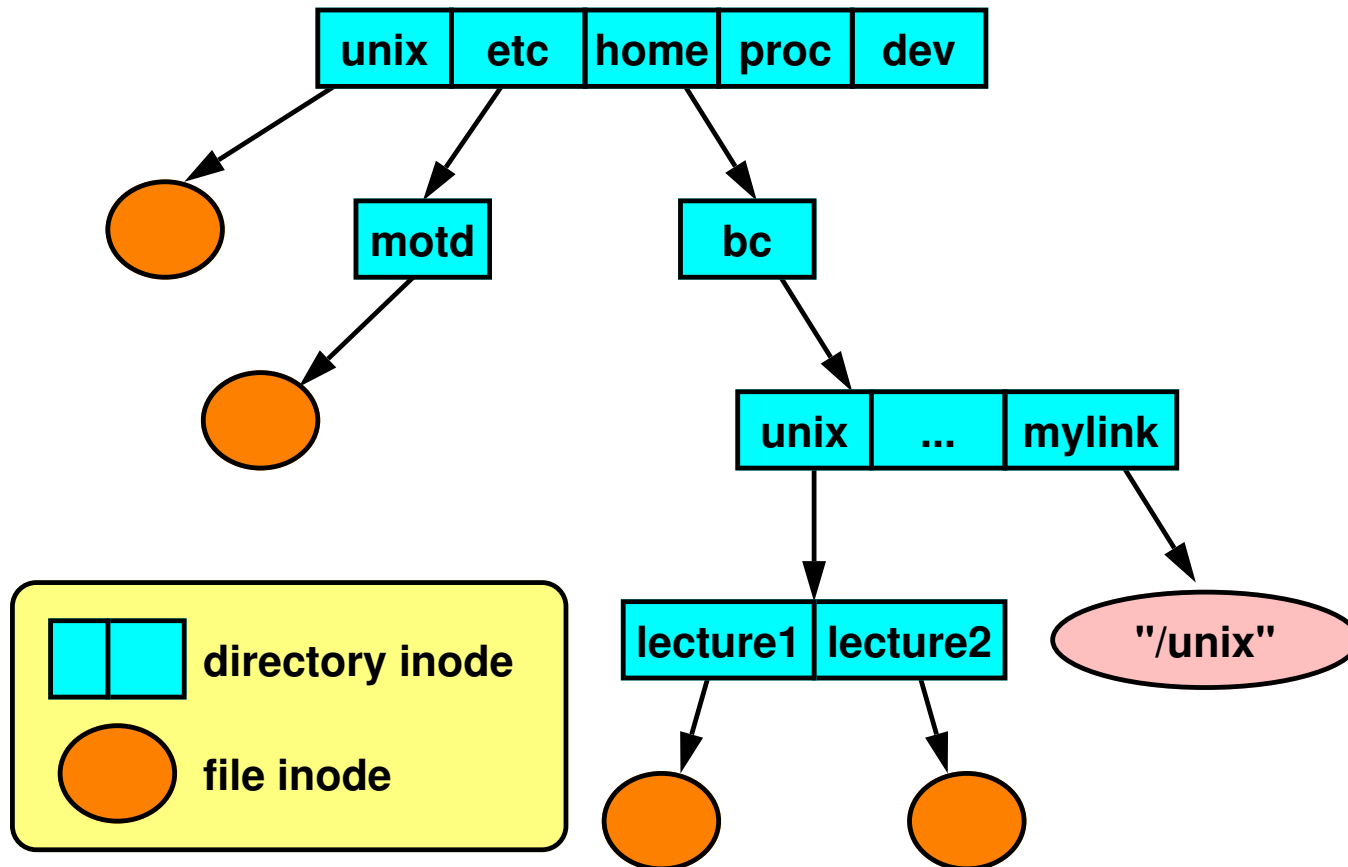


Soft Links

```
% ln -s /unix /home/bc/mylink
```

```
% ln -s /home/bc /etc/bc
```

— create "bc" in "/etc" to *soft-link* to "/home/bc"

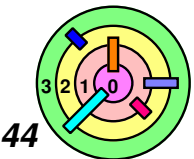
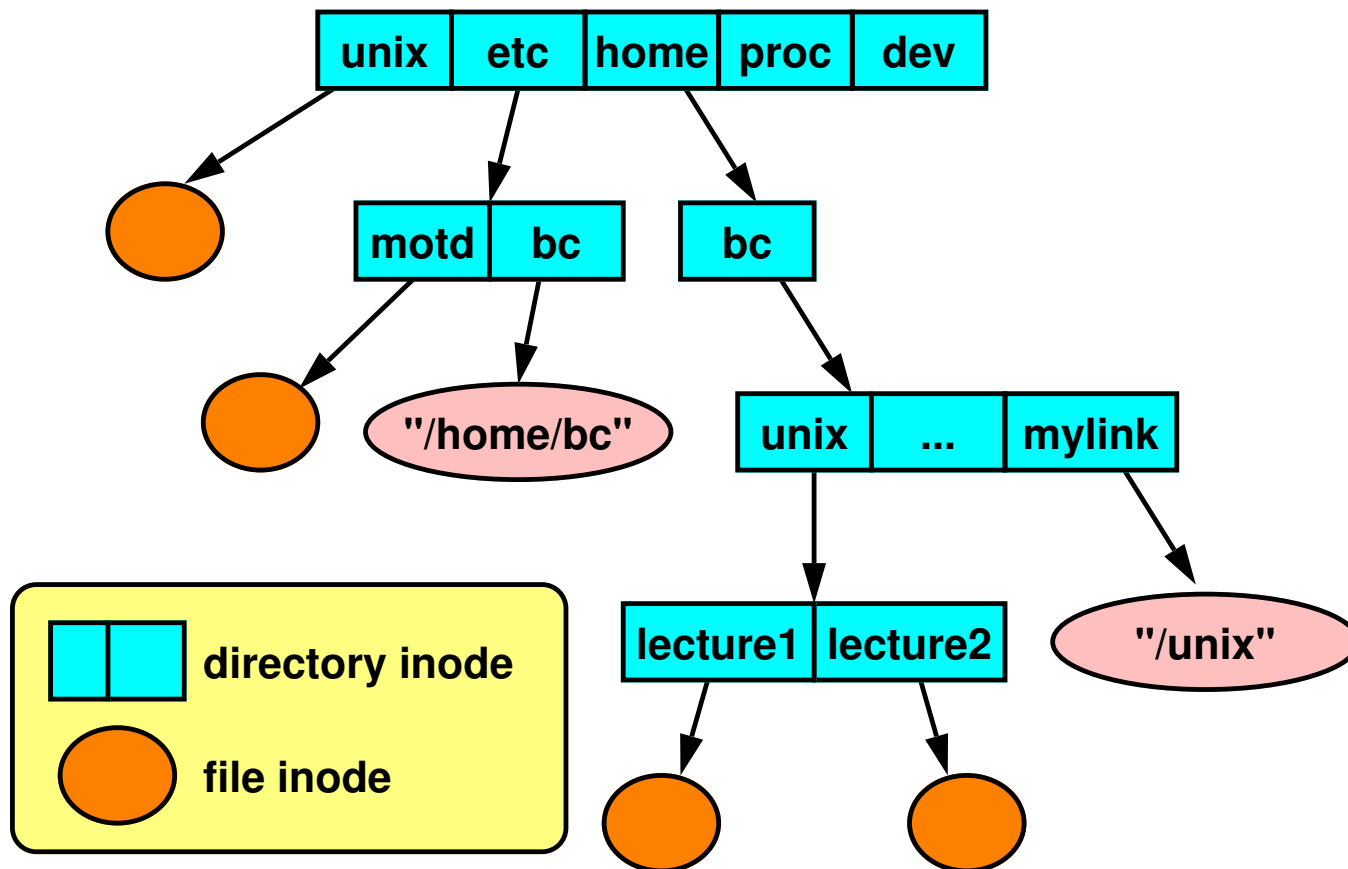


Soft Links

```
% ln -s /unix /home/bc/mylink
```

```
% ln -s /home/bc /etc/bc
```

— create "bc" in "/etc" to *soft-link* to "/home/bc"



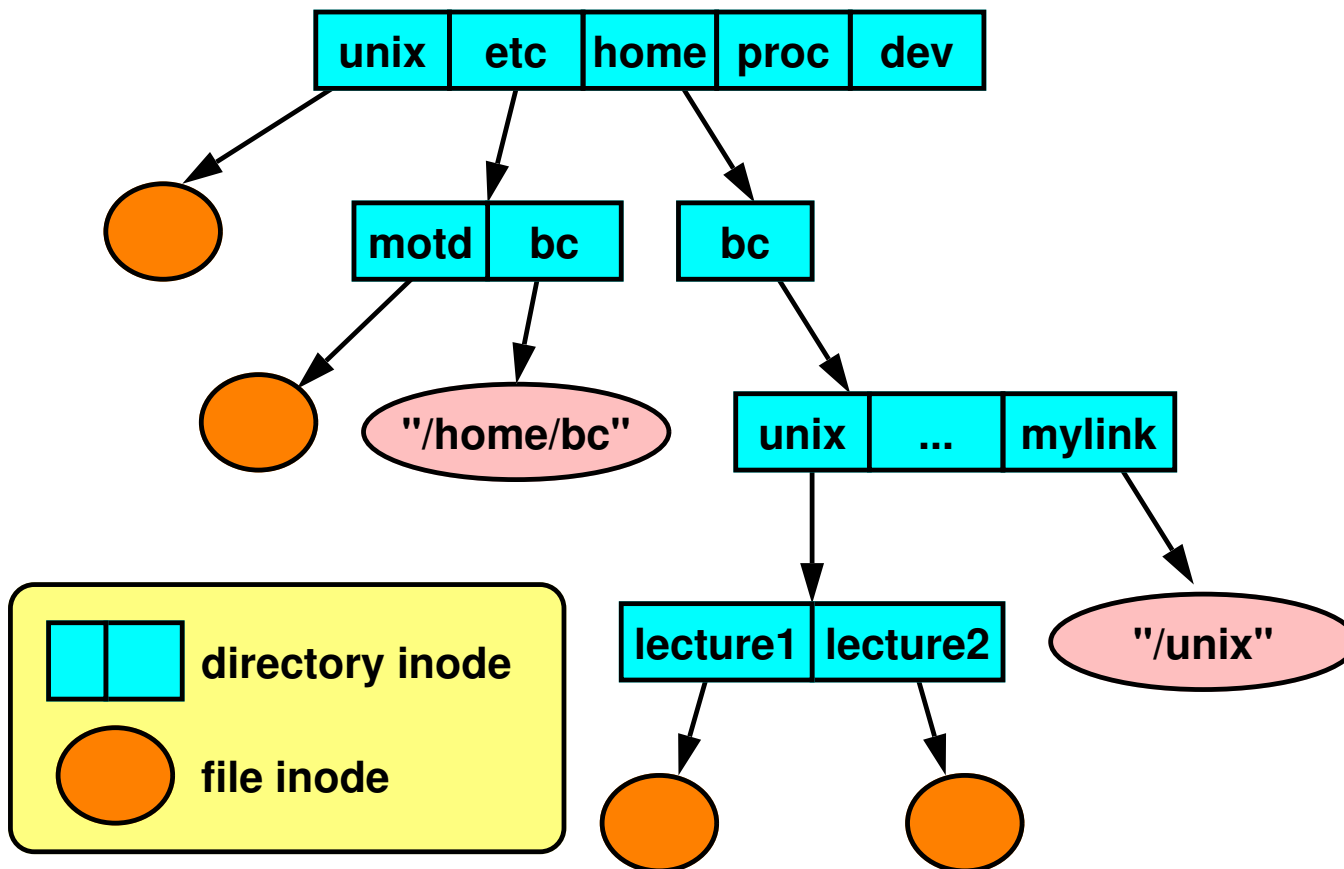
Soft Links

% `ls -l /etc/bc/unix/lecture1`

▢ same as "`ls -l /home/bc/unix/lecture1`", or is it?

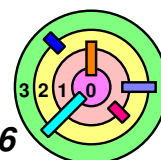
○ yes for the "root" account, may be no for the "bc" account

◆ see "access protection" later



Working Directory

- ➡ When you type "`ls`" in a Terminal, what directory content are you listing?
 - how does the shell know what directory content to list?
- ➡ **Working Directory:** maintained *in kernel for each process*
 - paths not starting from `"/` start with the working directory
 - get by using the `getcwd()` system call
 - set by using the `chdir()` system call
 - that's what the "`cd`" shell command does (clearly, "`cd`" cannot be a program)
 - displayed (via shell) using "`pwd`"



Kernel 2

- ➡ Now you have everything you need to complete kernel 2
 - ➡ if you are not familiar with a function, look at Linux man page
 - keep in mind that weenix is not Linux
 - your goal is to pass all the tests in the grading guidelines
- ➡ New things in kernel 2
 - ➡ *reference counting* (not as easy as it sounds)
 - if you have a reference counting bug, will get a kernel panic
 - ➡ *C++ polymorphism implemented in C*
 - the VFS layer is AFS-independent
 - can the VFS layer read data from disk?
 - ◆ no way
 - ◆ it needs to ask AFS to do it because only AFS knows what data structure is used on disk
 - ◆ need to invoke AFS in a FS-independent way
 - read the `ramfs` code
 - ➡ read *kernel 2 FAQ*

