UNIX System Programming

Directories and File System

- Objectives
  - look at how to program with directories
  - briefly describe the UNIX file system

Overview

1. Directory Implementation
2. Subdirectory Creation
3. “.” and “..”
4. mkdir()
5. rmdir()
6. Reading Directories
7. chdir()
8. getcwd()
9. Links

1. Directory Implementation

- A UNIX directory is a *file*:
  - it has an owner, group owner, size, access permissions, etc.
  - many file operations can be used on directories

- Differences:
  - modern UNIXs have special directory operations
    e.g. **`opendir()`**, **`readdir()`**

Directory Structure

- A directory ‘file’ is a sequence of lines; each line holds an *i-node number* and a file name.

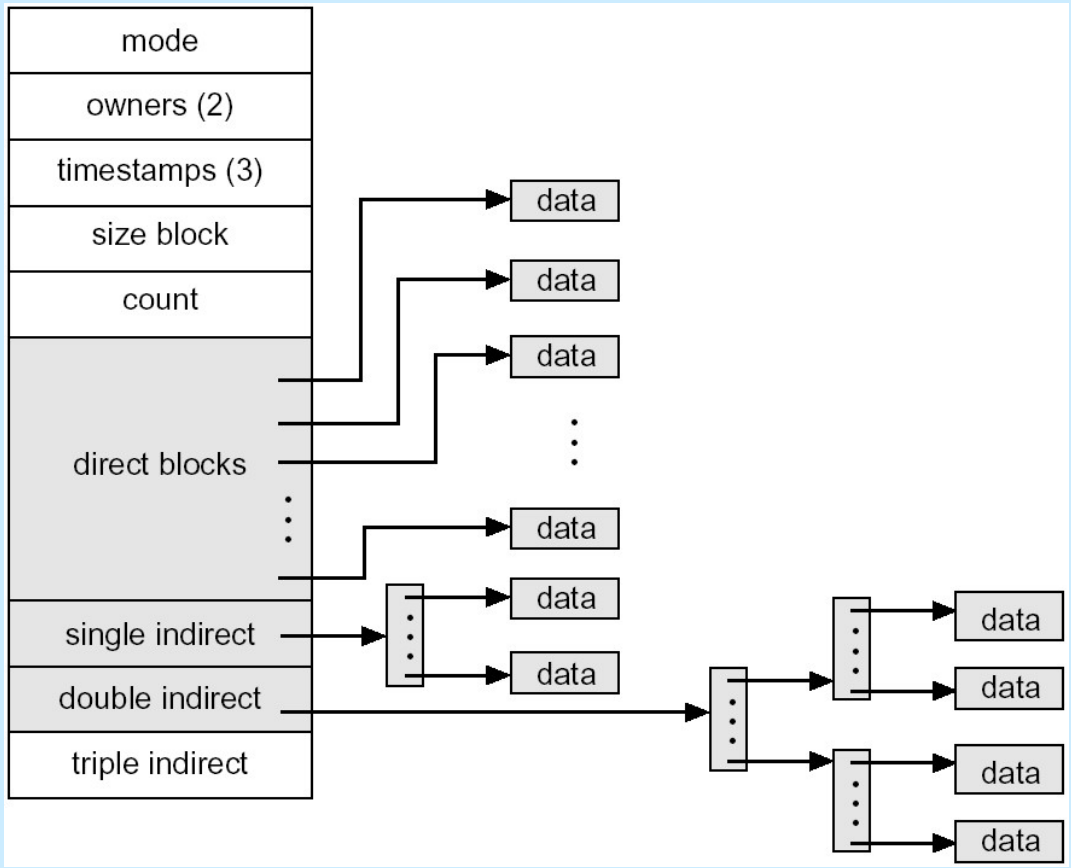
- The data is stored as binary, so we cannot simply use **`cat`** to view it

keeps track of what 'blocks' store your file's data

I-node:

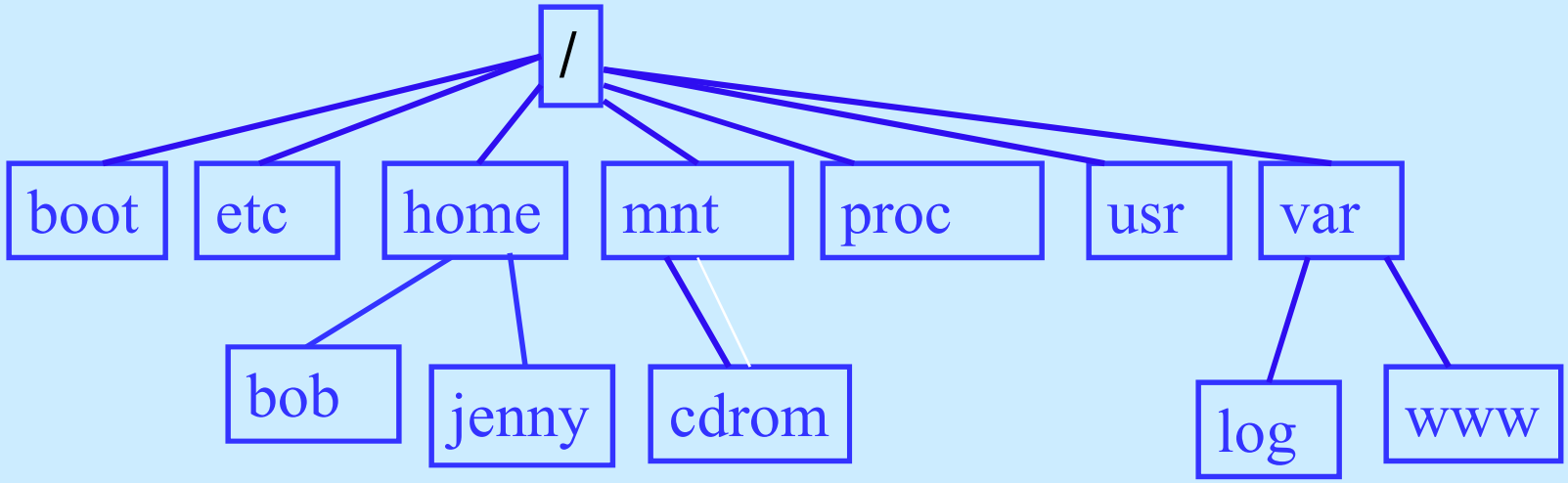
- The administrative information about a file is kept in a structure known as an *inode*.
  - Inodes in a file system, in general, are structured as an array known as an *inode table*.
- An inode number, which is an index to the inode table, *uniquely identifies* a file in a file system.

i-node and Data Blocks



UNIX directory hierarchy

- UNIX does not have the concept of separate drives as exist in the Windows world.
- All devices, when mounted, are accessed through a directory structure which starts at the / (root directory).



loki.trentu.ca

/ is the top of the tree

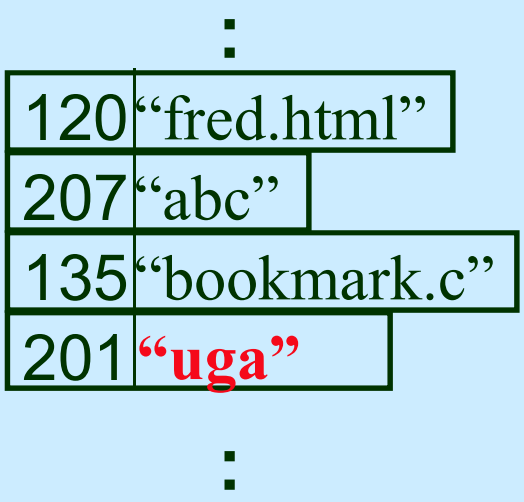
```
[jacques@loki ~]$ ls /
boot  etc   lib    lost+found  mnt  proc  run   srv  tmp  var
bin   dev   home   lib64  media       opt   root  sbin  sys  usr

[jacques@loki ~]$ df -h
```

df - disk subsystem

```
Filesystem      Size  Used Avail Use% Mounted on
/dev/sdb1        15G  6.7G  7.2G  49% /
devtmpfs        902M     0  902M   0% /dev
tmpfs           916M   84K  916M   1% /dev/shm
tmpfs           916M   97M  819M  11% /run
tmpfs           916M     0  916M   0% /sys/fs/cgroup
/dev/sda1        19G  301M   17G   2% /home
tmpfs           184M   16K  184M   1% /run/user/42
tmpfs           184M     0  184M   0% /run/user/1000
```

Directory Creation

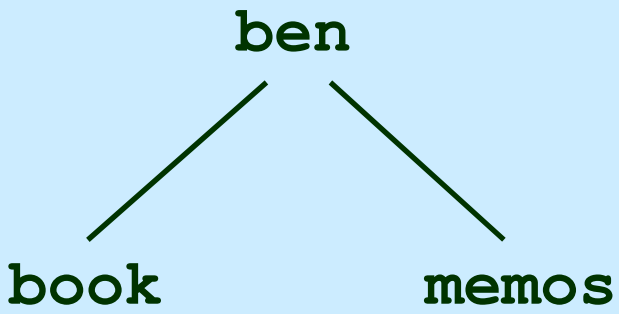
- **"`mkdir uga`"** causes:
  - the creation of a `uga` directory file and an i-node for it
  - an i-node number and name are added to the parent directory file



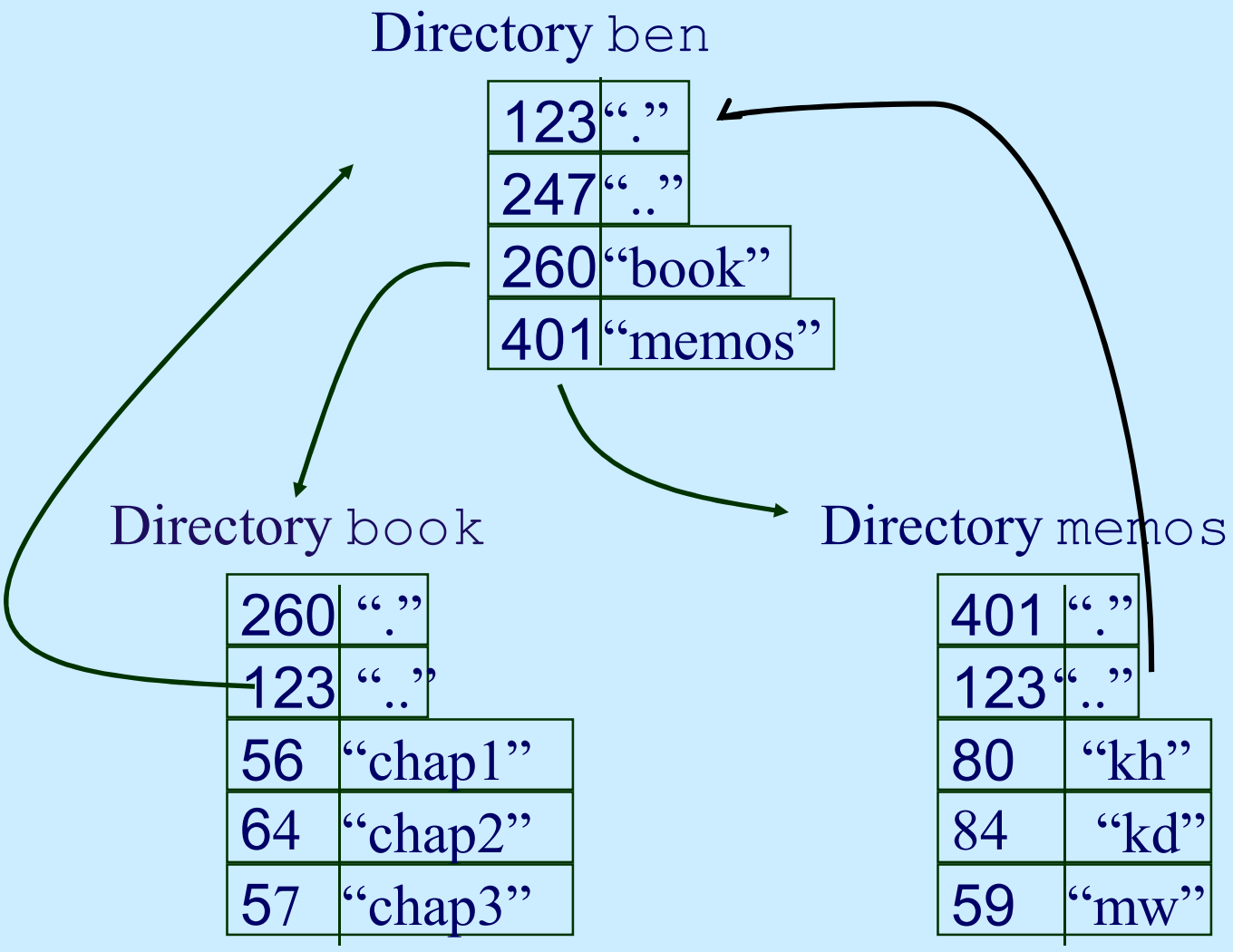
“.” and “..”

- In UNIX, the single period “.” represents the current directory.
- The double period “..” represents the parent directory.
- “.” and “..” are stored as ordinary file names with i-node numbers pointing to the correct directory files.

- Example:



In more detail:



mkdir()

```
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>

int mkdir(char *pathname, mode_t mode);
```

- Creates a new directory with the specified **mode**: return 0 if ok, -1 on error
- "." and ".." entries are added automatically
- **mode** must include execute permissions so the user(s) can use **cd**.

e.g. 0755

read write execute
r w x
1 1 1 (binary)
= 7 decomal

rmdir()

- ```
  #include <unistd.h>
  int rmdir(char *pathname);
  ```

- Delete an empty directory;
  return 0 if ok, -1 on error.

- Will delay until other processes have stopped using the directory.
  ```

Reading Directories

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(char *pathname);

struct dirent *readdir(DIR *dp);

int closedir(DIR *dp);
```

returns a pointer if ok, `NULL` on error

returns a pointer if ok, `NULL` at end or on error

dirent and DIR

- **struct dirent**

```
{
long d_ino;                    /* i-node number */
char d_name[NAME_MAX+1]; /* fname */
off_t d_off;                   /* offset to next rec */
unsigned short d_reclen; /* record length */
}
```

- DIR is a directory stream (similar to FILE)
  - when a directory is first opened, the stream points to the first entry in the directory

Example: listdir.c

List the contents of the current directory.

```
#include <stdio.h>
#include <dirent.h>

int main()
      {
      DIR *dp;
      struct dirent *dir;

      if( (dp = opendir(".")) == NULL )
                {
                fprintf( stderr, "Cannot open dir\n" );
                exit(1);
                }

      /* read entries */
      while( (dir = readdir(dp)) != NULL )
                {
                /* ignore empty records */
                if( dir->d_ino != 0 )
                          printf( "%s\n", dir->d_name );
                }
      closedir( dp );
      return 0;
      } /* end main */
```

if i node is not zero then print out the file name

chdir()

```
#include <unistd.h>
int chdir( char *pathname );
int fchdir( int fd );
```

- Change the current working directory (*cwd*) of the calling process; return 0 if ok, -1 on error.

Example: cd to /tmp

- Part of **to_tmp.c**:

```
 :
if( chdir("/tmp" ) < 0 )
        printf( "chdir error\n") ;
else
        printf( "In /tmp\n" );
```

Directory Change is Local

- The directory change is limited to within the program.

- e.g.

```
$ pwd
/usr/lib
$ to_tmp          /* from last slide */
In /tmp
$ pwd
/usr/lib
```

getcwd()

- ```
  #include <unistd.h>
  char *getcwd(char *buf, int size);
  ```

- Store the *cwd* of the calling process in `buf`; return `buf` if ok, `NULL` on error.

- `buf` must be big enough for the pathname string (`size` specifies the length of `buf`).
  ```

Example

```
#include <stdio.h>
#include <unistd.h>
#include <dirent.h>  /* for NAME_MAX */

int main()
     {
     char name[NAME_MAX+1];

     if( getcwd( name, NAME_MAX+1 ) == NULL )
             printf( "getcwd error\n" );
     else
             printf( "cwd = %s\n", name ):
       :
```

2. Links

2.1. What is a Link?

- A link is a pointer to a file.
- Useful for sharing files:
  - a file can be shared by giving each person their own link (pointer) to it.

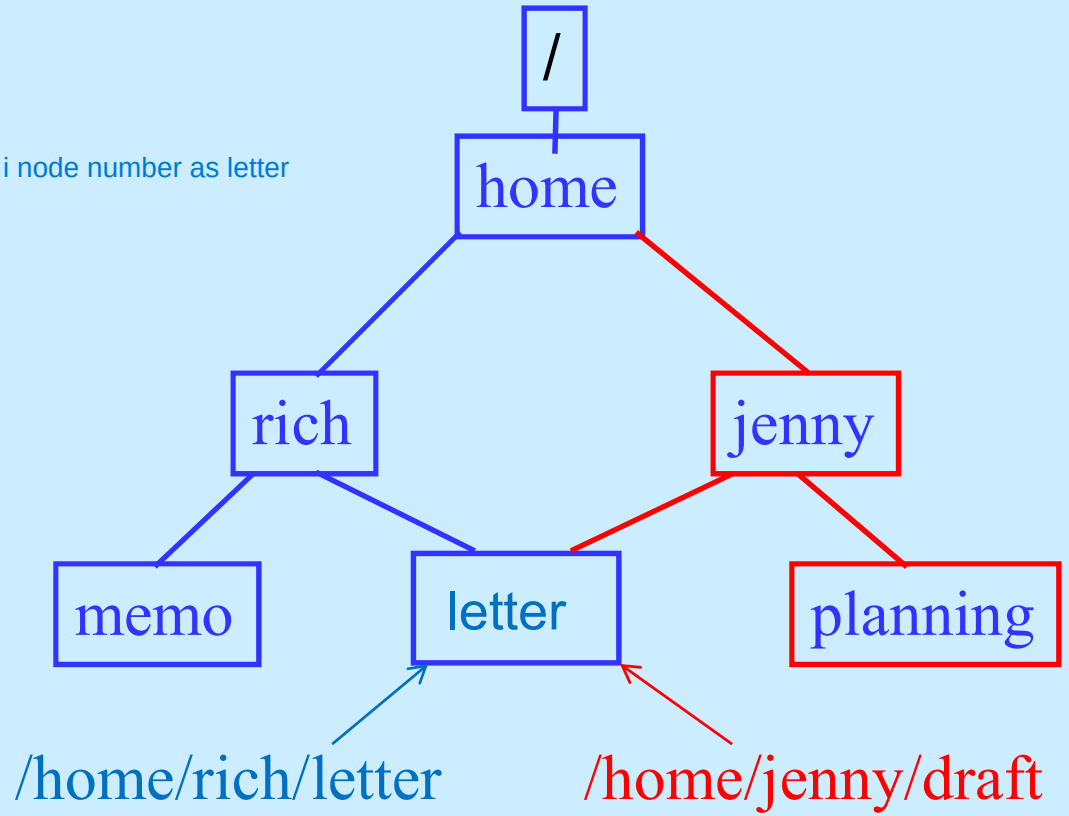
2.2. Creating a Link

syntax: ln existing-file new-pointer

- Jenny types:

```
ln draft /home/rich/letter
```

draft will have same i node number as letter



- Changes to a file affects every link:

```
$ cat file_a
This is file A.
$ ln file_a file_b
$ cat file_b
This is file A.

$ vi file_b
  :

$ cat file_b
This is file B after the change.
$ cat file_a
This is file B after the change.
```

2.3. Seeing Links

- Compare status information:

```
$ ls -l file_a file_b file_c file_d

-rw-r--r-- 2 dkl 33 May 24 10:52 file_a
-rw-r--r-- 2 dkl 33 May 24 10:52 file_b
-rw-r--r-- 1 dkl 16 May 24 10:55 file_c
-rw-r--r-- 1 dkl 33 May 24 10:57 file_d
```

- Look at inode number:

```
$ ls -i file_a file_b file_c file_d
```

-i to print i node numbers

```
3534 file_a   3534 file_b
5800 file_c   7328 file_d
```

2.4. Removing a Link

- Deleting a link does not remove the file.
- Only when the file *and* every link is gone will the file be removed.

2.5. Symbolic Links

- The links described so far are often called *hard links*
  - a hard link is a pointer to a file which must be on the *same* file system

on separate physical hard drive

- A *symbolic link* is an *indirect pointer* to a file
  - it stores the pathname of the pointed-to file
  - it can link across file systems

Lets say we have a system with a number of different hard drives.

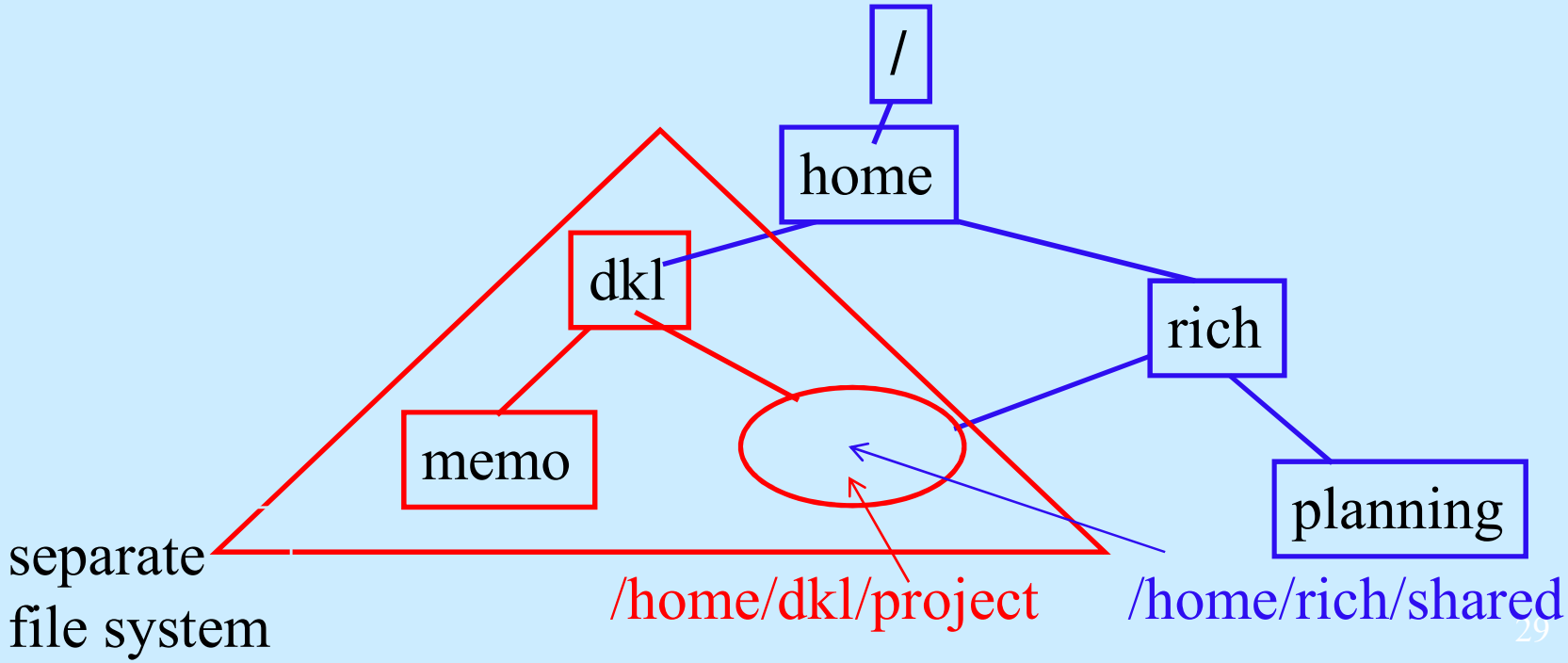
The **dkl** group has their own drive and have mounted off of the /home directory.

when Richard wants to access the file he has to type in /home/dkl/project every time.

He can create a link with a name in his own directory that he can access without having to specify a whole path.

- Richard types:

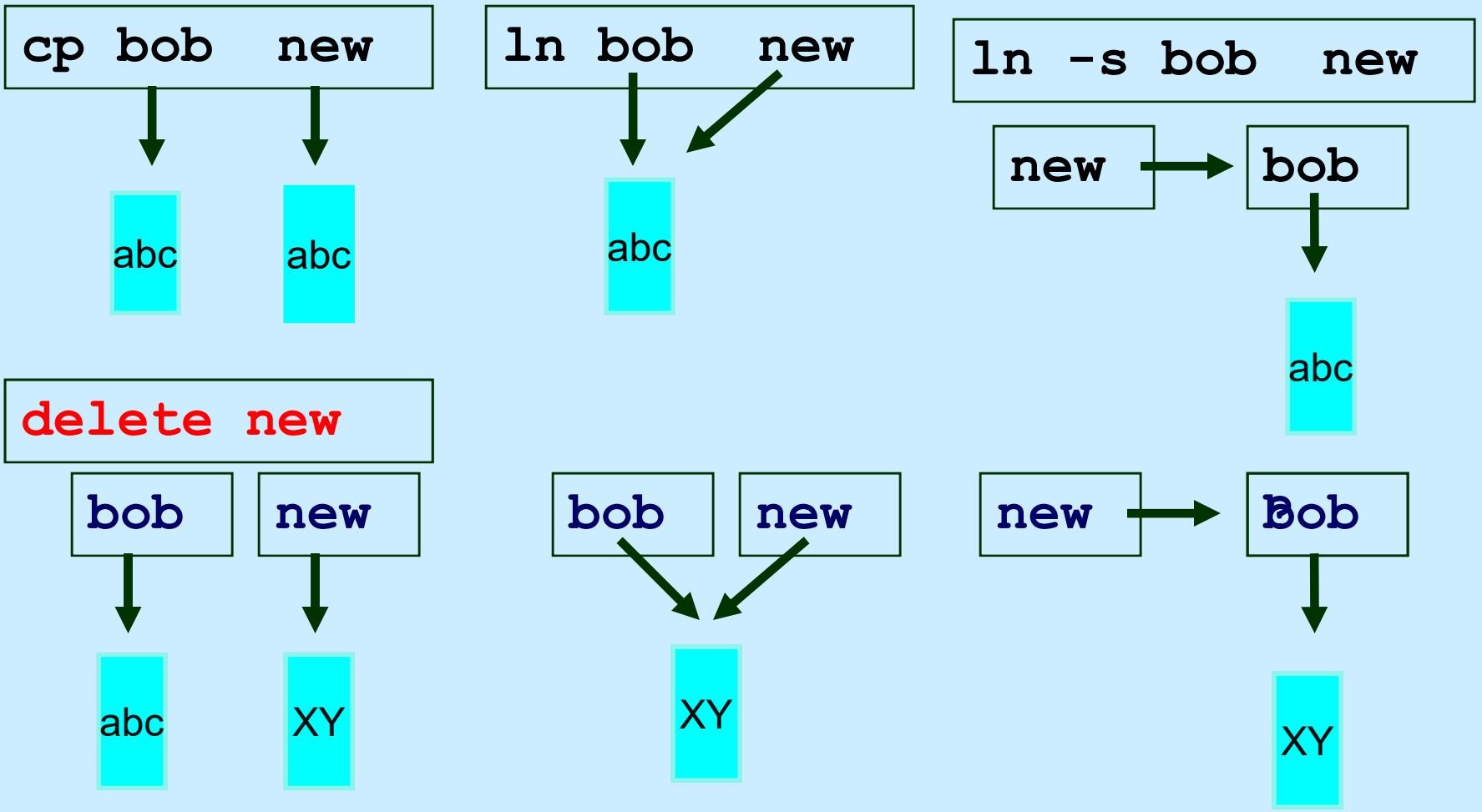
```
ln -s shared  /home/dkl/project
```



- Symbolic links are listed differently:

```
$ ln -s pics /home/mh/img
$ ls -lF pics /home/mh/img
drw-r--r-- 1 dkl staff 981 May 24 10:55 pics
lrwxrwxrxw 1 dkl staff   4 May 24 10:57/home/mh/img --> pics
```

2.6 Link Creation, Update & Removal

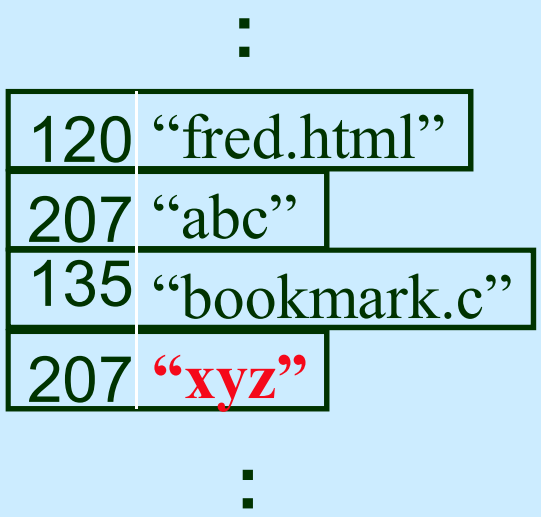


2.7 `link()` and `unlink()`

```
#include <unistd.h>
int link( const char *oldpath, const char *newpath );
```

- Meaning of:

```
link( "abc", "xyz" )
```



- `unlink()` clears the directory record
  - usually means that the i-node number is set to 0

- The i-node is only deleted when the last link to it is removed; the data block for the file is also deleted (reclaimed) & no process have the file opened

Example: unlink

```
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>

int main(void)
   {
   if( open( "tempfile", O_RDWR ) < 0 )
        {
        perror( "open error‟ );
        exit( 1 );
        }
   if( unlink( "tempfile‟ ) < 0 )
        {
        perror( "unlink error‟ );
        exit( 1 );
        }
   printf( "file unlinked\n‟ );
   exit(0);
   }
```

symlink()

```
#include <unistd.h>
int symlink(const char *oldpath, const char *newpath);
```

- Creates a symbolic link named *newpath* which contains the string *oldpath*.
- Symbolic links are interpreted at run-time.
- Dangling link – may point to an non-existing file.
- If *newpath* exists it will not be overwritten.

readlink()

```
#include <unistd.h>
int readlink( const char *path, char *buf, size_t bufsiz);
```

- Read value of a **symbolic link** (does *not* follow the link).
  - Places the contents of the symbolic link *path* in the buffer *buf*, which has size *bufsiz*.
  - Does not append a NULL character to *buf*.

- Return value
  - The count of characters placed in the buffer if it succeeds.
    - -1 if an error occurs.