IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering

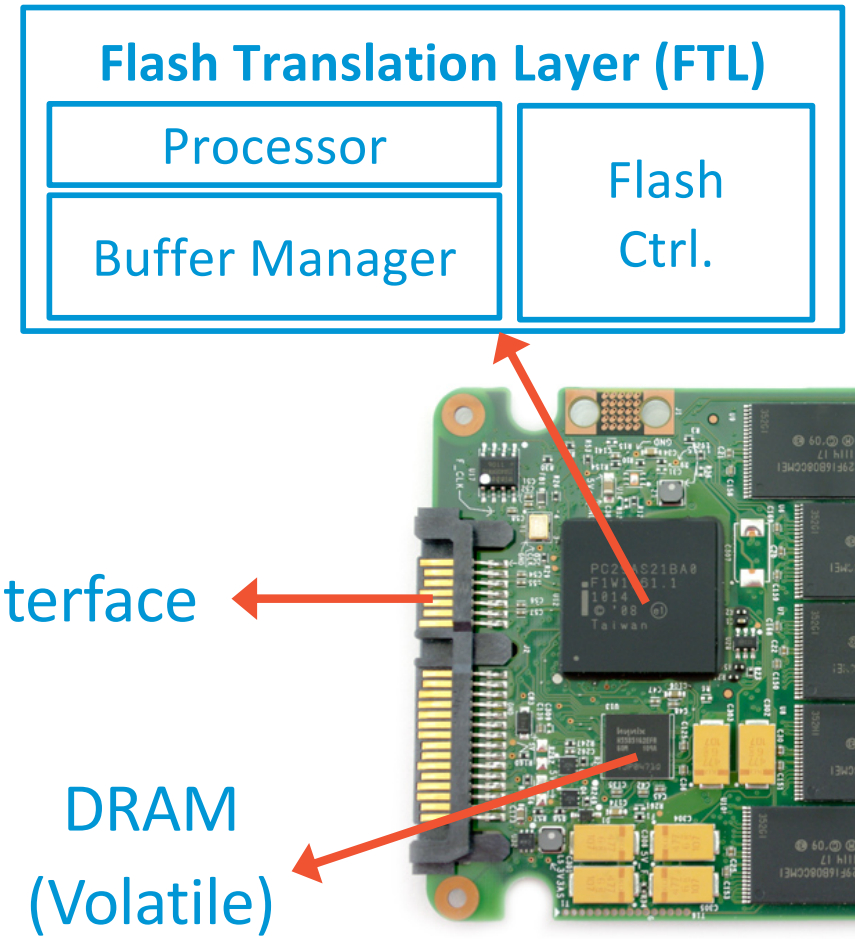
Lecture 30:
File & Directory

Agenda

- Recap
- File & Directory
  - File Operations
  - Directory operations
  - Hard link & Symbolic link

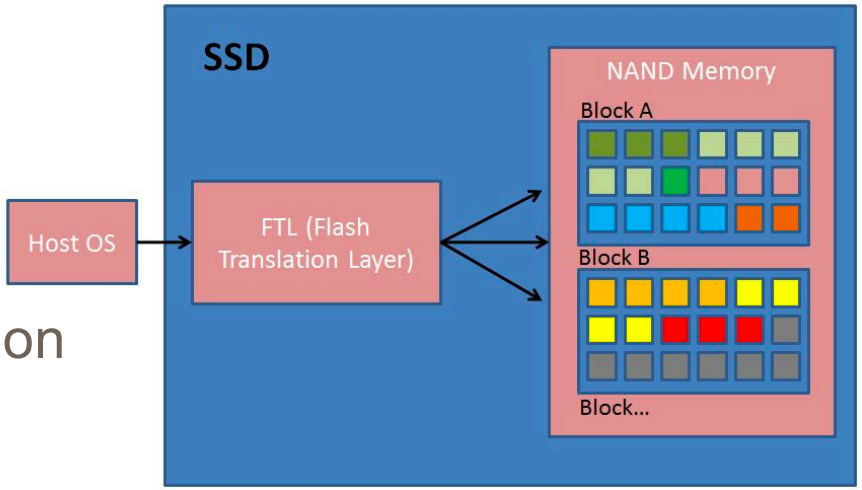
Recap

- SSD Internals



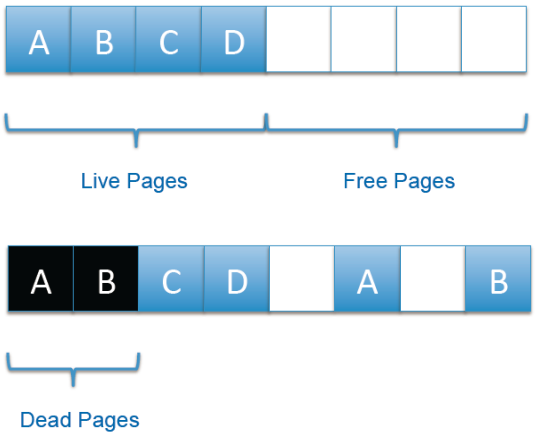
Recap

- Flash Memory
  - **SLC vs MLC**
    - MLC is used in consumer marker
  - **NOR** vs **NAND**
    - NAND is used in SSDs
  - **Block**
    - minimum unit of **erase** operation
    - contain multiple pages
  - **Page**
    - minimum unit of **program** operation
  - each cell can only stand a limited number of program/erasure cycles (**P/E cycles**)



Recap

- Flash Translation Layer (FTL)
  - Logical block mapping
    - maps logical addresses to physical addresses
    - maintains a mapping table
    - out-of-space update (append-only)
  - Garbage Collection
    - re-cycle invalid pages
    - source of I/O instability
  - Wear leveling
    - let the flash cells be erased/programmed about the same number of times



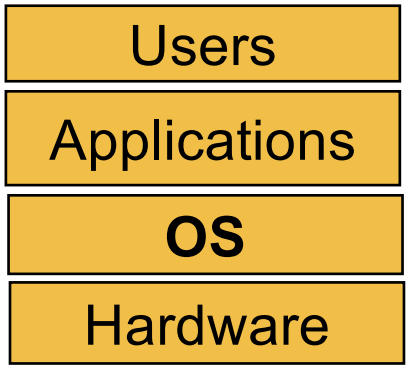
Agenda

- ~~Recap~~
- File & Directory
  - File Operations
  - Directory operations
  - Hard link & Symbolic link

Persistent Storage of Data

- Hardware devices
  - Hard disk drives (HDDs)
  - Flash-based solid state drives (SSDs)

- OS abstractions for storage
  - File
  - Directory



File

- Represent various data
  - naming: diff. requirements on diff. OSes
    - Windows: filename + extension
    - Most UNIX-like OSes do not rely on extension
      - just for human reading

Extension	Meaning
file.bak	Backup file
file.c	C source program
file.gif	Compuserve Graphical Interchange Format image
file.hlp	Help file
file.html	World Wide Web HyperText Markup Language document
file.jpg	Still picture encoded with the JPEG standard
file.mp3	Music encoded in MPEG layer 3 audio format
file.mpg	Movie encoded with the MPEG standard
file.o	Object file (compiler output, not yet linked)
file.pdf	Portable Document Format file
file.ps	PostScript file
file.tex	Input for the TEX formatting program
file.txt	General text file
file.zip	Compressed archive

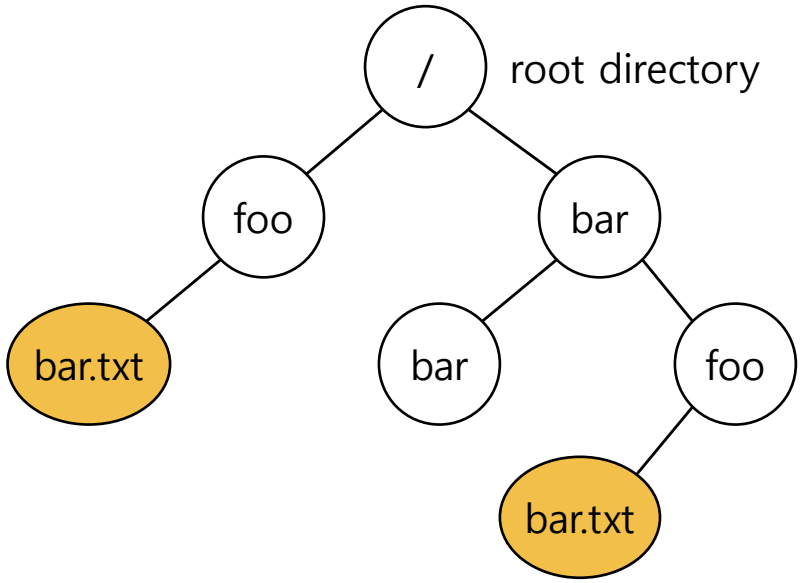
File

- A linear array of bytes
- A **file system** (FS) is responsible for managing and storing files persistently on disk
  - e.g., Ext4, FAT, NTFS
  - consist of a number of data structures
    - on disk & in memory
  - also include implementations of file operations
    - e.g., open/read/write/close/...
- Each file has a unique, low-level name called **inode number** in the file system
  - the user is not aware of this name

Directory

- Directory is similar to a file
  - also has a inode number in FS
  - contains a list of (user-readable name, low-level name) pairs.
  - Each entry in a directory refers to either *files* or other *directories*
- Example
  - A directory has an entry (“foo”, “10”)
    - A file “foo” with the low-level name “10”

Directory Tree (Directory Hierarchy)



Valid files (absolute pathname) :
/foo/bar.txt
/bar/foo/bar.txt

Valid directory :
/
/foo
/bar
/bar/bar
/bar/foo/

Sub-directories

File Operations

- Creating file
  - Use `open()` system call with `O_CREAT` flag.

```
int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);
```

    - `O_CREAT` : create file
    - `O_WRONLY` : only write to that file while opened.
    - `O_TRUNC` : make the file size zero (remove any existing content)
  - `open()` system call returns a **file descriptor**
    - *File descriptor* is an integer, and is used as a identifier to access the corresponding file

File Operations

- Reading and Writing Files
  - e.g., reading and writing '`foo`' file
    - redirect the output of `echo` to the file
    - `cat` : dump the content of the file to the screen

```
prompt> echo hello > foo
prompt> cat foo
hello
prompt>
```

File Operations

- Reading and Writing Files
  - e.g., reading and writing '`foo`' file (cont')
    - use `strace` to trace the system calls made by a program

```
prompt> strace cat foo
…
open("foo", O_RDONLY|O_LARGEFILE)  = 3
read(3, "hello\n", 4096)           = 6
write(1, "hello\n", 6)             = 6 // file descriptor 1: standard out
hello
read(3, "", 4096)                  = 0 // 0: no bytes left in the file
close(3)                           = 0
…
prompt>
```

File Operations

- Reading and Writing Files
  - e.g., reading and writing '`foo`' file (cont')
    - use `strace` to trace the system calls made by a program

      - `open(file descriptor, flags)`
        - Return file descriptor (3 in example)
        - File descriptor 0, 1, 2, is for standard input/ output/ error.
      - `read(file descriptor, buffer pointer, the size of the buffer)`
        - Return the number of bytes it reads
      - `write(`file descriptor, buffer pointer, the size of the buffer`)`
        - Return the number of bytes it writes

File Operations

- File offset
  - An open file has a **current offset**.
    - Determine **where** the next read or write will begin reading from or writing to within the file.
  - Update the current offset
    - **Implicitly**: A read or write of `N` bytes takes place, `N` is added to the current offset.
    - **Explicitly**: `lseek()`

File Operations

- File offset

```
off_t lseek(int fildes, off_t offset, int whence);
```

  - `fildes` : File descriptor
  - `offset` : Position the file offset to a particular location within the file
  - `whence` : Determine how the seek is performed
    - from the man page

```
If whence is SEEK_SET, the offset is set to offset bytes.
If whence is SEEK_CUR, the offset is set to its current
location plus offset bytes.
If whence is SEEK_END, the offset is set to the size of the
file plus offset bytes.
```

File Operations

- File sync
  - By default, FS will buffer writes in memory for some time
    - e.g., 30 seconds
    - Performance reasons
  - Write seems to complete quickly
    - the data will actually be flush from memory to the storage device at a later point in time
    - Data can be lost (e.g., power outages or machine crashes)
  - some applications require more than eventual guarantee.
    - e.g., DBMS requires durable transactions
      - force writes to disk from time to time

File Operations

- File sync

```
off_t fsync(int fd)
```

  - forces all dirty in-memory data to disk for the file referred to by the file descriptor
  - returns 0 on success
    - all of theses writes are complete
  - e.g.,

```
int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);
assert (fd > -1)
int rc = write(fd, buffer, size);
assert (rc == size);
rc = fsync(fd);
assert (rc == 0);
```

    - on some FS, need to fsync the directory containing the foo file

File Operations

- Rename file

```
rename(char* old, char *new)
```

  - Rename a file to different name
    - implemented as an **atomic call**
    - e.g., How to update a file atomically:

```
int fint fd = open("foo.txt.tmp", O_WRONLY|O_CREAT|O_TRUNC);
write(fd, buffer, size); // write out new version of file
fsync(fd);
close(fd);
rename("foo.txt.tmp", "foo.txt");
```

File Operations

- Getting Information About Files
  - `stat(), fstat()`: Show the file metadata
    - e.g., Size, Low-level name, Permission, …
    - `stat` structure:

```
struct stat {
    dev_t st_dev;         /* ID of device containing file */
    ino_t st_ino;         /* inode number */
    mode_t st_mode;       /* protection */
    nlink_t st_nlink;     /* number of hard links */
    uid_t st_uid;         /* user ID of owner */
    gid_t st_gid;         /* group ID of owner */
    dev_t st_rdev;        /* device ID (if special file) */
    off_t st_size;        /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for filesystem I/O */
    blkcnt_t st_blocks;   /* number of blocks allocated */
    time_t st_atime;      /* time of last access */
    time_t st_mtime;      /* time of last modification */
    time_t st_ctime;      /* time of last status change */
};
```

File Operations

- Getting Information About Files
  - `stat(), fstat()`: Show the file metadata
    - e.g., Size, Low-level name, Permission, …
    - command line tool `stat`:

```
prompt> echo hello > file
prompt> stat file

File: ‘file’
Size: 6 Blocks: 8 IO Block: 4096 regular file
Device: 811h/2065d Inode: 67158084 Links: 1
Access: (0640/-rw-r-----) Uid: (30686/ root) Gid: (30686/ remzi)
Access: 2011-05-03 15:50:20.157594748 -0500
Modify: 2011-05-03 15:50:20.157594748 -0500
Change: 2011-05-03 15:50:20.157594748 -0500
```

    - FS keeps this metadata information in a `inode` structure

File Operations

- Remove file
  - `rm` is Linux command to remove a file
    - `rm` call `unlink()` to remove a file.

```
prompt> strace rm foo
…
unlink("foo")                = 0      // return 0 upon success
…
prompt>
```

Directory Operations

- Make a directory
  - `mkdir()`

```
prompt> strace mkdir foo
…
mkdir("foo", 0777)                    = 0
prompt>
```

  - When a directory is created, it is empty
    - Empty directory have two default entries: . (itself), .. (parent)

```
prompt> ls -a
./       ../
prompt> ls -al
total 8
drwxr-x---  2 remzi remzi    6 Apr 30 16:17 ./
drwxr-x--- 26 remzi remzi 4096 Apr 30 16:17 ../
```

Hard Link

- **Link** a new file name to an old one

```
int link(const char *oldpath, const char *newpath);
```

  - Create another way to refer to *the same file*
  - The command-line link program : `ln`

```
prompt> echo hello > file
prompt> cat file
hello
prompt> ln file file2  // create a hard link, link file to file2
prompt> cat file2
hello
```

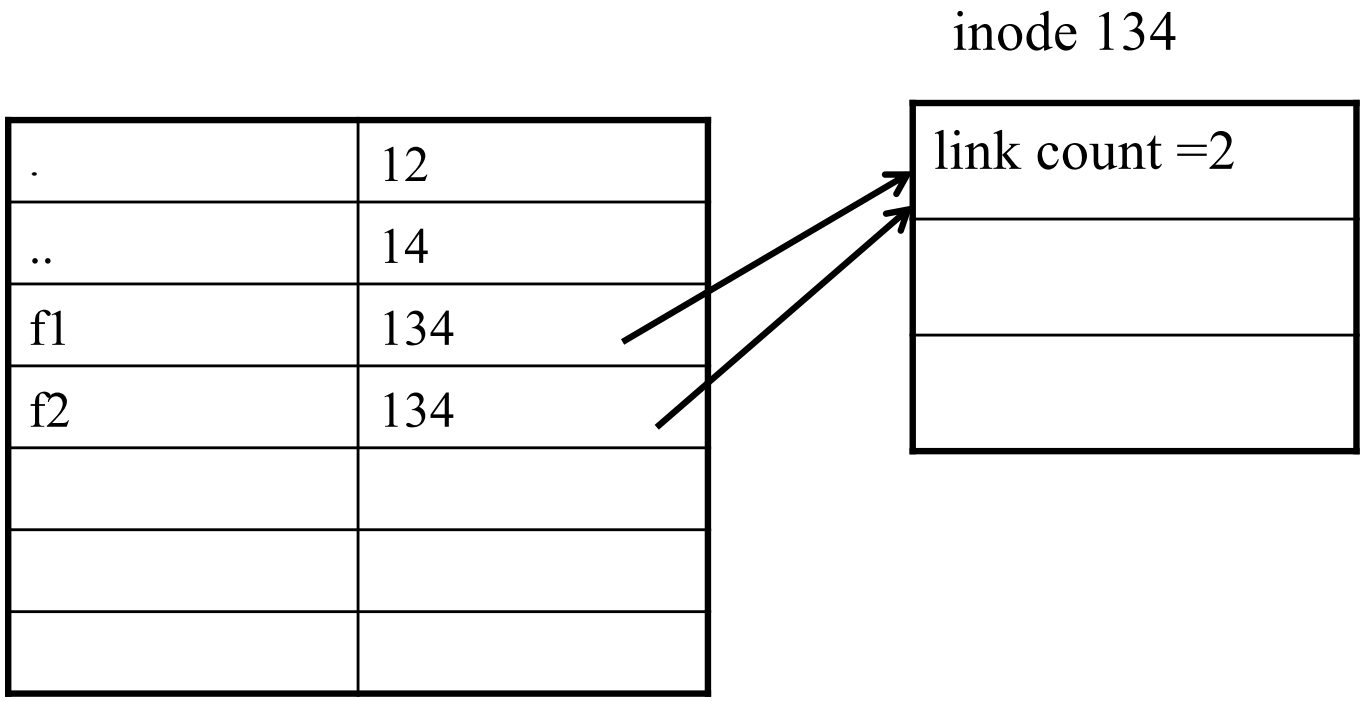
Hard Link

- **Link** a new file name to an old one
  - The way `link` works:
    - **Create** another name in the directory.
    - **Refer** it to the same inode number of the original file.
      - The file content is not copied in any way.
    - Then, we have two human names (`file` and `file2`) that are both referenced by the same inode number
      - same file content
      - ls –i: list inode number for each file

```
prompt> ls -i file file2
67158084 file  /* inode value is 67158084 */
67158084 file2 /* inode value is 67158084 */
prompt>
```

Hard Link

- Both files point to the same inode
  - e.g., `ln /home/mai/f1 /home/mai/f2`



Hard Link

- Thus, to remove a file, we call `unlink()`.

```
prompt> rm file
removed 'file'
prompt> cat file2          // Still access the file
hello
```

- ***reference count***
  - Track how many different file names have been linked to this inode.
  - When `unlink()` is called, the reference count decrements.
  - If the reference count reaches zero, the filesystem free the inode and related data blocks. → truly "delete" the file

Symbolic Link

- Symbolic link is more flexible than Hard link.
  - Hard Link cannot create to a directory.
  - Hard Link cannot create to a file to other partition.
    - Because inode numbers are only unique within a file system
  - Create a symbolic link: `ln -s`

```
prompt> echo hello > file
prompt> ln –s file file2   /* option –s : create a symbolic link, */
prompt> cat file2
hello
```


reference count

Track how many different file names have been linked to this

Symbolic Link

- Symbolic link is **a special type** of file

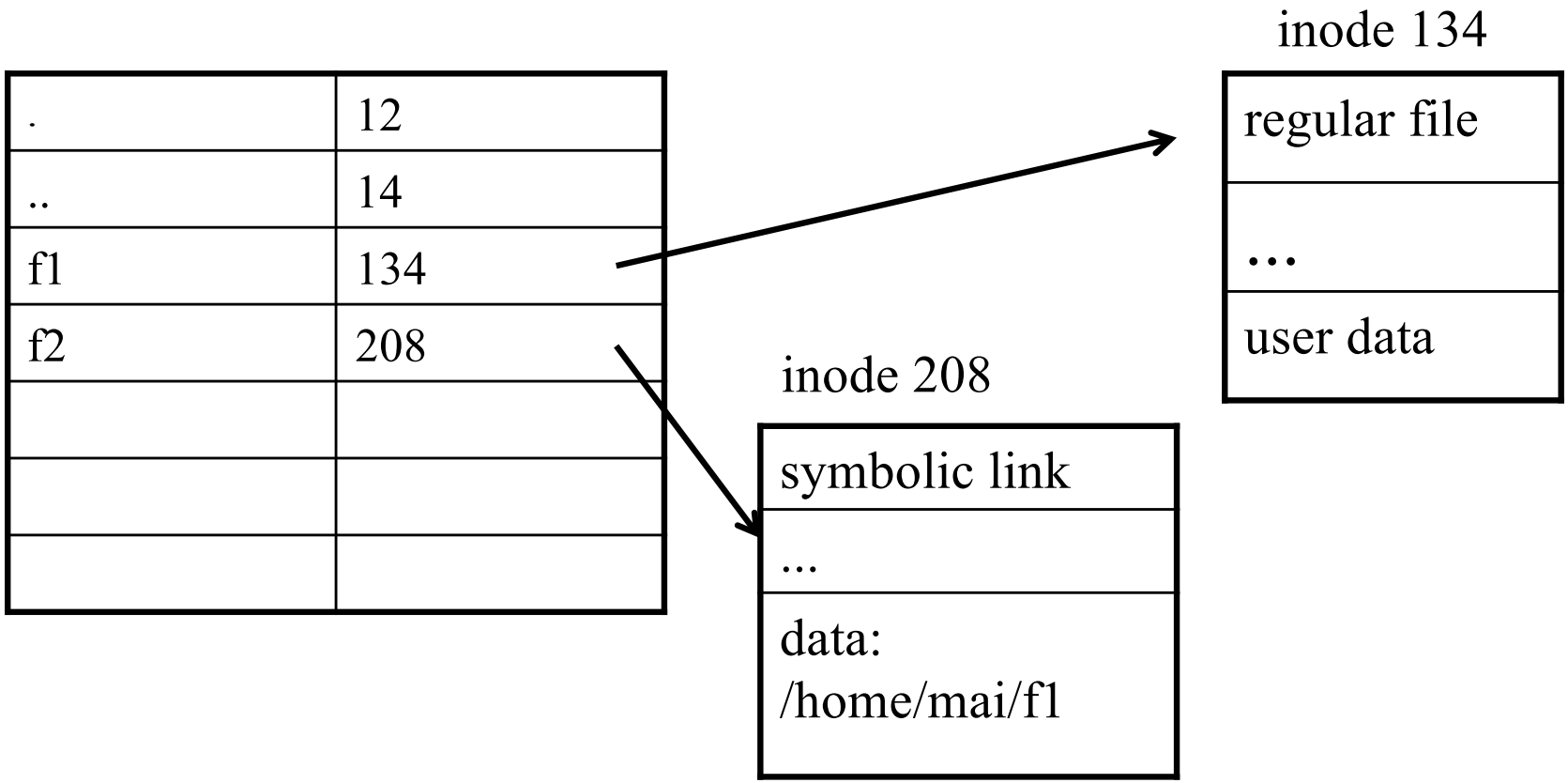
```
prompt> stat file
 ... regular file ...
prompt> stat file2
 ... symbolic link ...        // Actually a file it self of a different type
```

  - A symbolic link holds the pathname of the linked-to file as the data of the link file
    - a longer name may lead to large size

```
prompt> ls -al
drwxr-x---  2 remzi remzi   29 May 3 19:10 ./
drwxr-x--- 27 remzi remzi 4096 May 3 15:14 ../              // directory
-rw-r-----  1 remzi remzi    6 May 3 19:10 file             // regular file
lrwxrwxrwx  1 remzi remzi    4 May 3 19:10 file2 -> file    // symbolic link
```

Symbolic Link

- A symbolic link has its own inode number
  - e.g., `ln -s /home/mai/f1 /home/mai/f2`



Symbolic Link

- Dangling reference
  - When remove a original file, symbolic link points to nothing

```
prompt> echo hello > file
prompt> ln -s file file2
prompt> cat file2
hello
prompt> rm file            // remove the original file
prompt> cat file2
cat: file2: No such file or directory
```

Agenda

- ~~Recap~~
- ~~File & Directory~~
  - ~~File Operations~~
  - ~~Directory operations~~
  - ~~Hard link & Symbolic link~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.