CSE150
Operating Systems
Lecture 21

File Systems

Building a File System (Review)

- File System: Layer of OS that transforms block interface of disks (or other block devices) into Files, Directories, etc.

- File System Components
  – Disk Management: organizing disk blocks into files
  – Naming: Interface to find files by name, not by blocks
  – Protection: Layers to keep data secure
  – Reliability/Durability: Keeping of files durable despite crashes, media failures, attacks, etc.

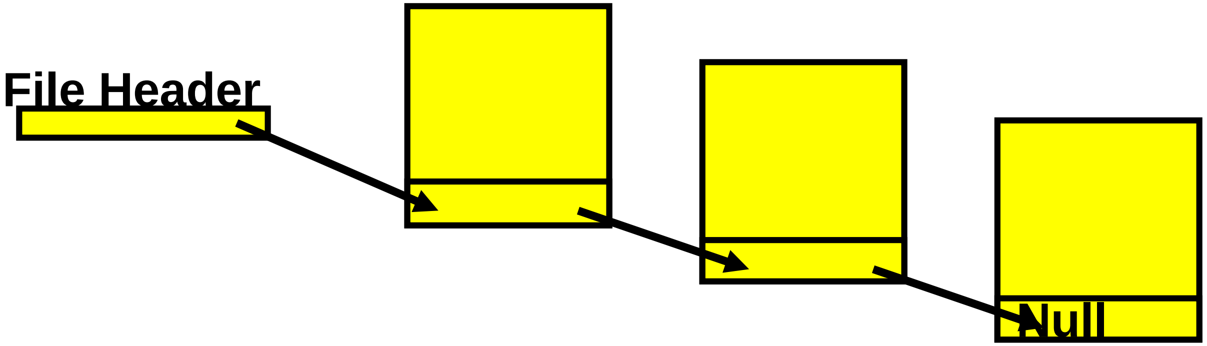
- File System Goals
  – Maximize sequential performance
  – Efficient random access to file
  – Easy management of files (growth, truncation, etc.)

How to organize files on disk

- First Technique: Continuous Allocation
  - Use continuous range of blocks in logical block space
    - » Analogous to base+bounds in virtual memory
    - » User says in advance how big file will be (disadvantage)
  - Search bit-map for space using best fit/first fit
    - » What if not enough contiguous space for new file?
  - File Header Contains:
    - » First block/LBA in file
    - » File size (# of blocks)
  - Pros: Fast Sequential Access, Easy Random access
  - Cons: External Fragmentation/Hard to grow files
    - » Free holes get smaller and smaller
    - » Could compact space, but that would be *really* expensive

Linked List Allocation

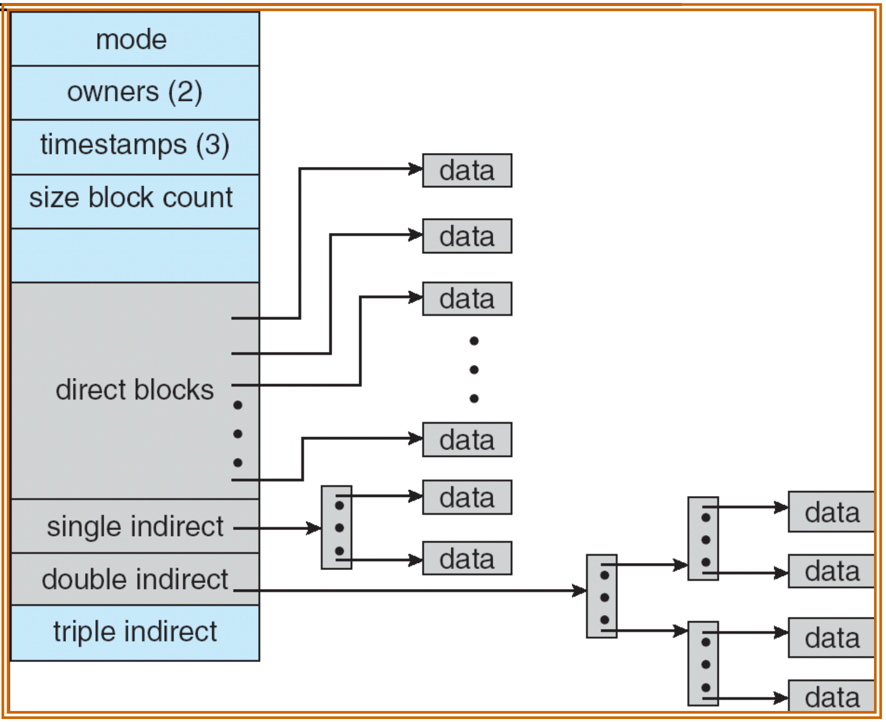
- Second Technique: Linked List Approach
  - Each block, pointer to next on disk



  - Pros: Can grow files dynamically, Free list same as file
  - Cons: Bad Sequential Access (seek between each block), Unreliable (lose block, lose rest of file)
  - Serious Con: Bad random access!!!!
  - Technique originally from Alto (First PC, built at Xerox)
    - » No attempt to allocate contiguous blocks

Multilevel Indexed Files (UNIX 4.1)

- Multilevel Indexed Files: (from UNIX 4.1 BSD)
  - Key idea: efficient for small files, but still allow big files



- File hdr contains 13 pointers
  - Fixed size table, pointers not all equivalent
  - This header is called an “inode” in UNIX
- File Header format:
  - First 10 pointers are to data blocks
  - Ptr 11 points to “(singly) indirect block” containing 256 block ptrs
  - Pointer 12 points to “doubly indirect block” containing 256 indirect block ptrs for total of 64K blocks
  - Pointer 13 points to a triply indirect block (16M blocks)

Today

- Naming and Directories
- File Caching, Durability

How do we actually access files?

- All information about a file contained in its file header
  - UNIX calls this an "inode"
    - » Inodes are global resources identified by index ("inumber")
  - Once you load the header structure, all blocks of file are locatable

- Question: how does the user ask for a particular file?
  - One option: user specifies an inode by a number (index).
    - » Imagine: open("14553344")
  - Better option: specify by textual name
    - » Have to map name→inumber
  - Another option: Icon
    - » This is how Apple made its money. Graphical user interfaces. Point to a file and click

Naming

- Naming (name resolution): process by which a system translates from user-visible names to system resources

- In the case of files, need to translate from strings (textual names) or icons to inumbers/inodes

Directories

- Directory: a relation used for naming
  - Just a table of (file name, inumber) pairs
- How are directories constructed?
  - Directories often stored in files
    - » Reuse of existing mechanism
    - » Directory named by inode/inumber like other files
  - Needs to be quickly searchable
    - » Options: Simple list or Hash table
    - » Can be cached into memory in easier form to search
- How are directories modified?
  - System calls for manipulation: `mkdir`, `rmdir`
  - Ties to file creation/destruction
    - » On creating a file by name, new inode grabbed and associated with new file in particular directory
  - Then, direct read/write of special file.

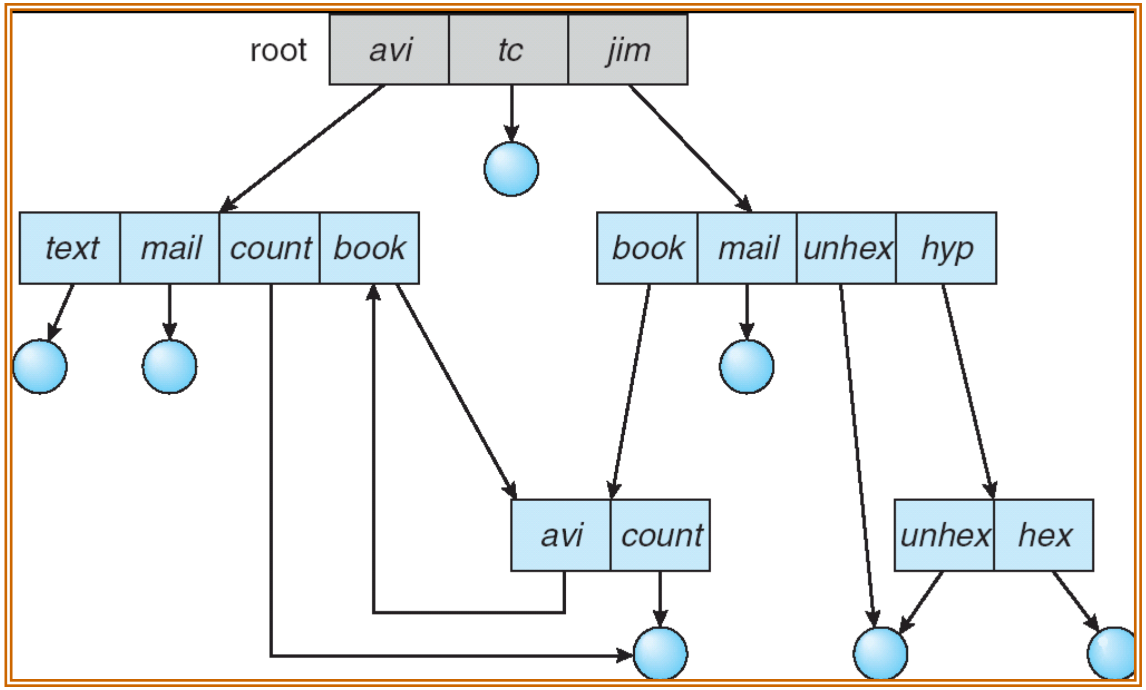
Directory Organization

- Directories organized into a hierarchical structure
  – Permits much easier organization of data structures

- Entries in directory can be either files or directories

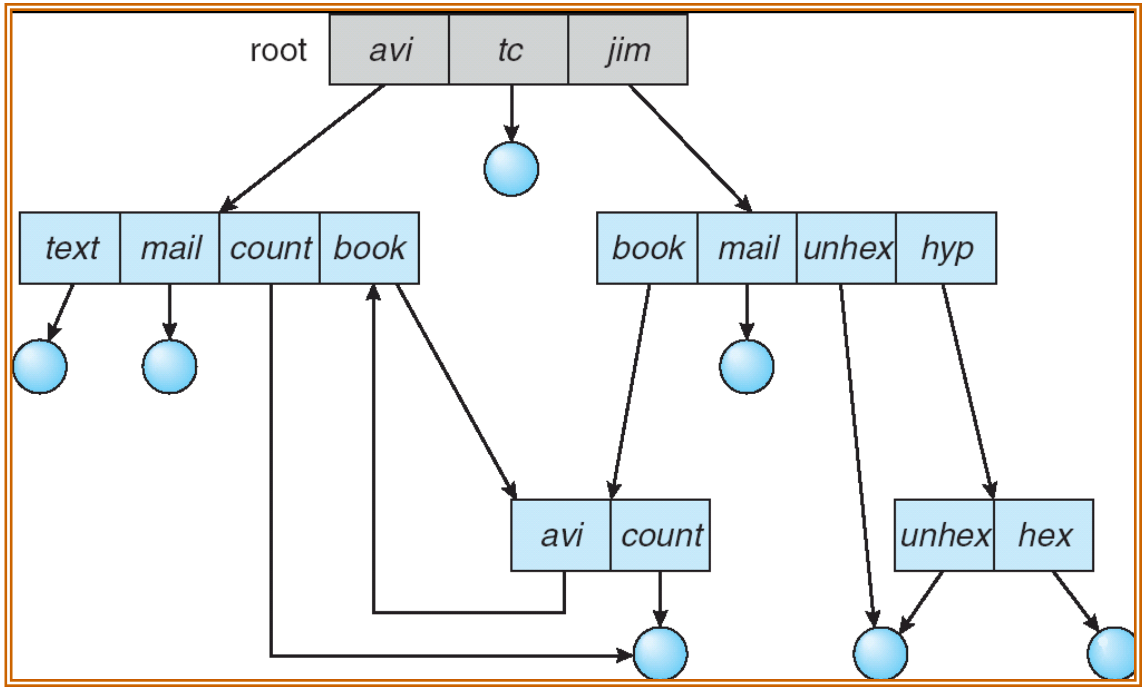
- Files named by ordered set (e.g., `/programs/p/list`)

Directory Structure



- Not really a hierarchy!
  - Many systems allow directory structure to be organized as an acyclic graph or even a (potentially) cyclic graph
  - Hard Links: different names for the same file
    - » Multiple directory entries point at the same file
  - Soft Links: "shortcut/symlink" pointers to other files
    - » Implemented by storing the logical name of actual file

Directory Structure



- Name Resolution: The process of converting a logical name into a physical resource (like a file)
  - Traverse succession of directories until reach target file

Where are inodes stored?

- In early UNIX and DOS/Windows' FAT file system, headers stored in special array in outermost cylinders
  - Header not stored anywhere near the data blocks. To read a small file, seek to get header, seek back to data.
  - Fixed size, set when disk is formatted. At formatting time, a fixed number of inodes were created (They were each given a unique number, called an "inumber")

Where are inodes stored?

- Later versions of UNIX moved the header information to be closer to the data blocks
  – Often, the inode for a file stored in the same "cylinder group" as the parent directory of the file (makes an `ls` of that directory run fast).
  – Pros:
    » For small directories, can fit all data, file headers, etc. in same cylinder $\Rightarrow$ no seeks!
    » File headers much smaller than whole block (a few hundred bytes), so multiple headers fetched from disk at same time
    » Reliability: whatever happens to the disk, you can find many of the files (even if directories disconnected)
  – Part of the Fast File System (FFS)
    » General optimization to avoid seeks

Where are inodes stored?

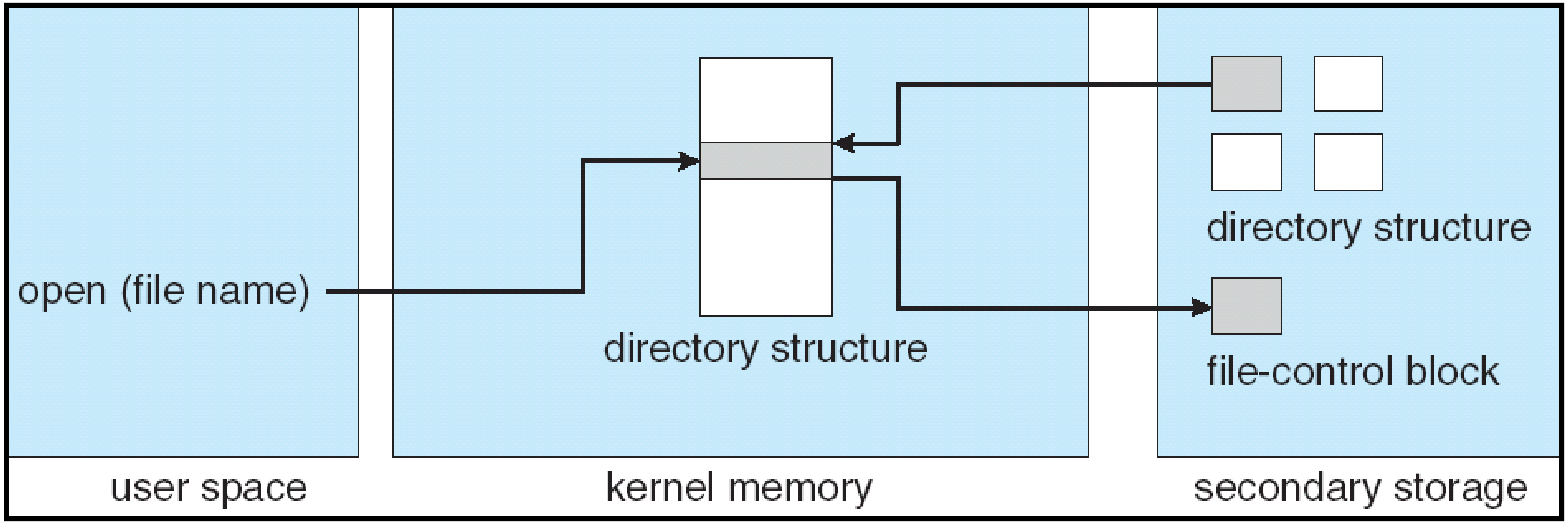
- How many disk accesses to resolve "`./my/book/count`"?
  - Read in file header for root (fixed spot on disk)
  - Read in first data block for root
    - » Table of (filename, inumber/index) pairs. Search linearly – ok since directories typically very small
  - Read in file header for "my"
  - Read in first data block for "my"; search for "book"
  - Read in file header for "book"
  - Read in first data block for "book"; search for "count"
  - Read in file header for "count"

- Current working directory: Per-address-space pointer to a directory (inode) used for resolving file names
  - Allows user to specify relative filename instead of absolute path (say CWD="`/my/book`" can resolve "count")

File System Caching

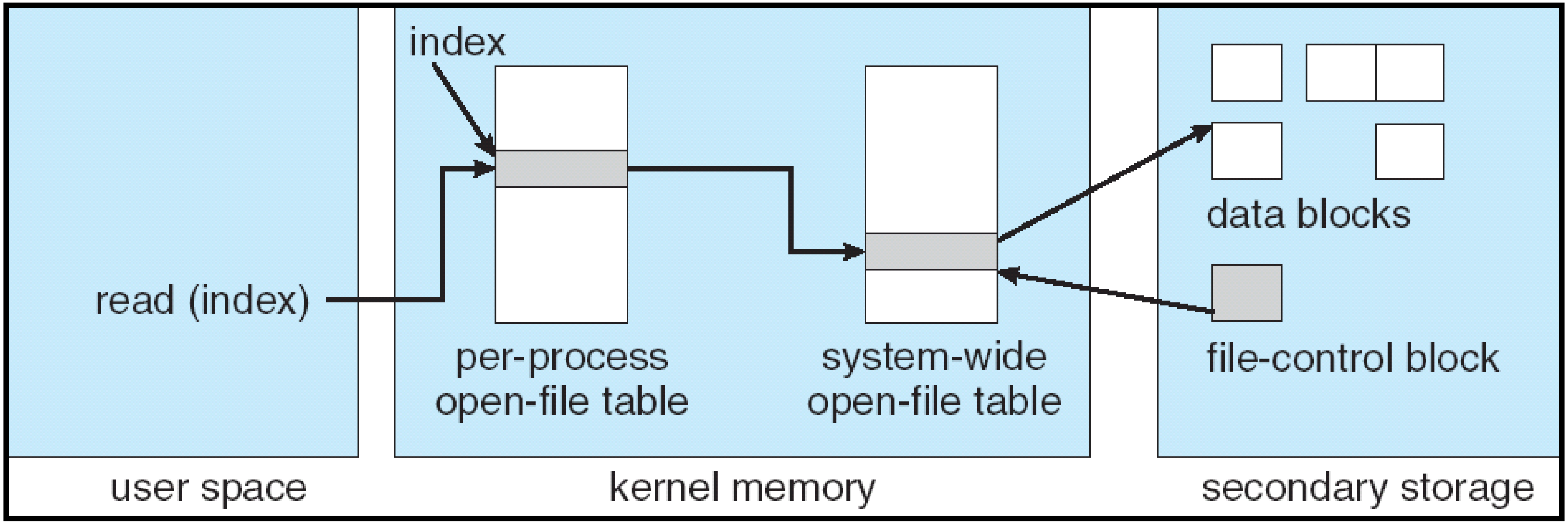
- Key Idea: Exploit locality by caching data in memory
  - Name translations: Mapping from paths→inodes
  - Disk blocks: Mapping from block address→disk content
- Buffer Cache: Memory used to cache kernel resources, including disk blocks and name translations
  - Can contain “dirty” blocks (blocks not yet on disk)

File System Caching



- Open system call:
  - Resolves file name, finds file control block (inode)
  - Makes entries in per-process and system-wide tables
  - Returns index (called "file handle") in open-file table

File System Caching



- Read/write system calls:
  – Use file handle to locate inode
  – Perform appropriate reads or writes

File System Caching

- Key Idea: Exploit locality by caching data in memory
  - Name translations: Mapping from paths→inodes
  - Disk blocks: Mapping from block address→disk content
- Buffer Cache: Memory used to cache kernel resources, including disk blocks and name translations
  - Can contain “dirty” blocks (blocks not yet on disk)
- Replacement policy? LRU
  - Can afford overhead of timestamps for each disk block
  - Advantages:
    - » Works very well for name translation
    - » Works well in general as long as memory is big enough to accommodate a host’s working set of files.
  - Disadvantages:
    - » Fails when some application scans through file system, thereby flushing the cache with data used only once
    - » Example: `find . –exec grep foo {} \;`

File System Caching (con't)

- Cache Size: How much memory should the OS allocate to the buffer cache vs virtual memory?
  - Too much memory to the file system cache $\Rightarrow$ won't be able to run many applications at once
  - Too little memory to file system cache $\Rightarrow$ many applications may run slowly (disk caching not effective)
  - Solution: adjust boundary dynamically so that the disk access rates for paging and file access are balanced
- Read Ahead Prefetching: fetch sequential blocks early
  - Key Idea: exploit fact that most common file access is sequential by prefetching subsequent disk blocks ahead of current read request (if they are not already in memory)
  - How much to prefetch?
    - Too many imposes delays on requests by other applications
    - Too few causes many seeks (and rotational delays) among concurrent file requests

Important “ilities”

- Availability: the probability that the system can accept and process requests
  - Often measured in “nines” of probability. So, a 99.9% probability is considered “3-nines of availability”
  - Key idea here is independence of failures
- Durability: the ability of a system to recover data despite faults
  - This idea is fault tolerance applied to data
  - Doesn’t necessarily imply availability: information on pyramids was very durable, but could not be accessed until discovery of Rosetta Stone
- Reliability: the ability of a system or component to perform its required functions under stated conditions for a specified period of time (IEEE definition)
  - Usually stronger than simply availability: means that the system is not only “up”, but also working correctly
  - Includes availability, security, fault tolerance/durability
  - Must make sure data survives system crashes, disk crashes, other problems

Journaled File System

- Better reliability through use of log
  - All changes are treated as *transactions*.
    - » A transaction either happens *completely* or *not at all*
  - A transaction is *committed* once it is written to the log
    - » Data forced to disk for reliability
- Journaled system:
  - Log used to asynchronously update filesystem
    - » Log entries removed after used
  - After crash:
    - » Remaining transactions in the log performed (“Redo”)
  - Ext3 (Linux), XFS (Unix), etc.
- Although File system may not be updated immediately, data preserved in the log

Other ways to make file system durable?

- Make sure writes survive in short term
  - Either abandon delayed writes or
  - use special, battery-backed RAM (called non-volatile RAM or NVRAM) for dirty blocks in buffer cache.
- Make sure that data survives in long term
  - Need to replicate! More than one copy of data!
  - Important element: independence of failure
    - » Could put copies on one disk, but if disk head fails…
    - » Could put copies on different disks, but if server fails…
    - » Could put copies on different servers, but if building is struck by lightning….
    - » Could put copies on servers in different continents…
- RAID: Redundant Arrays of Inexpensive Disks
  - Data stored on multiple disks (redundancy)
  - Either in software or hardware
    - » In hardware case, done by disk controller; file system may not even know that there is more than one disk in use
- Disk blocks contain Reed-Solomon error correcting codes (ECC) to deal with small defects in disk drive
  - Can allow recovery of data from small media defects

Summary

- Naming: act of translating from user-visible names to actual system resources
  – Directories used for naming for local file systems

- Important system properties
  – Availability: how often is the resource available?
  – Durability: how well is data preserved against faults?
  – Reliability: how often is resource performing correctly?

- Use of Log to improve Reliability
  – Journaled file systems such as ext3

Next two lectures

- Chapter 17 – Distributed systems and networking.