Distributed OS Concepts

Adapted from

Operating System Concepts *tenth edition*

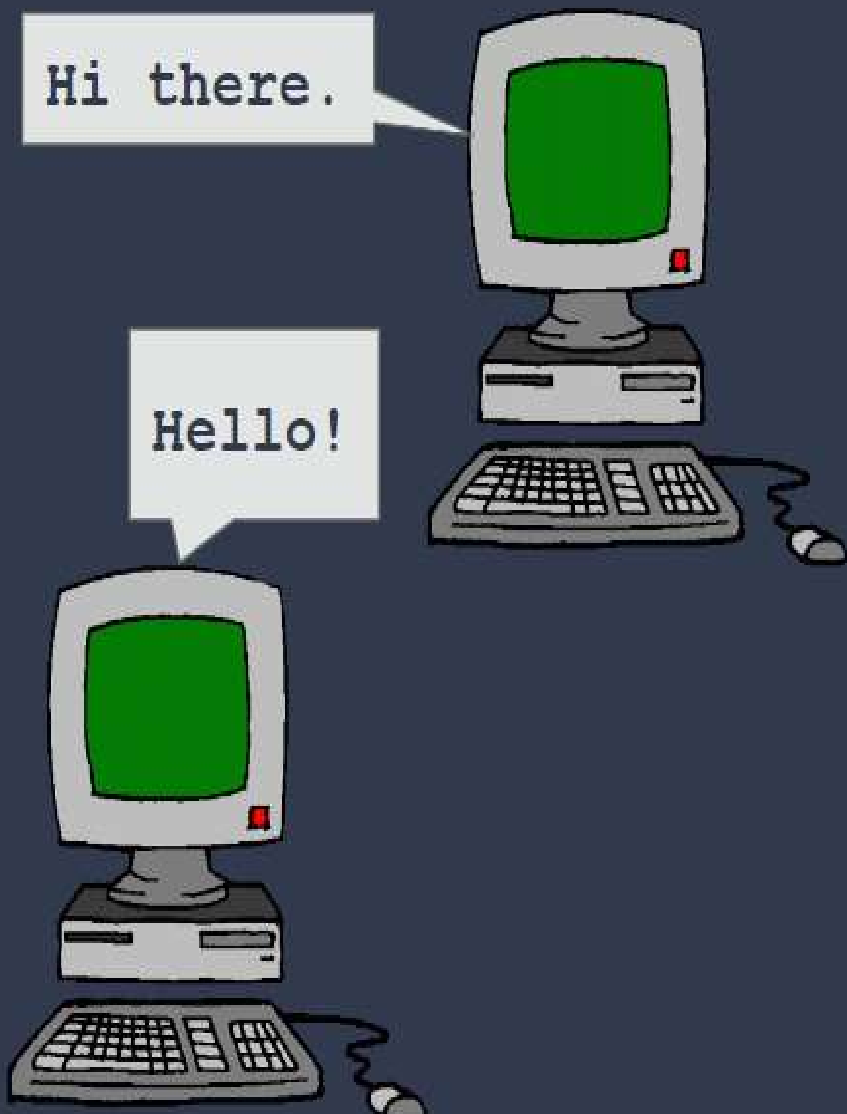
Abraham Silberschatz
Peter Baer Galvin
Greg Gagne

Operating System

- What is a distributed system?

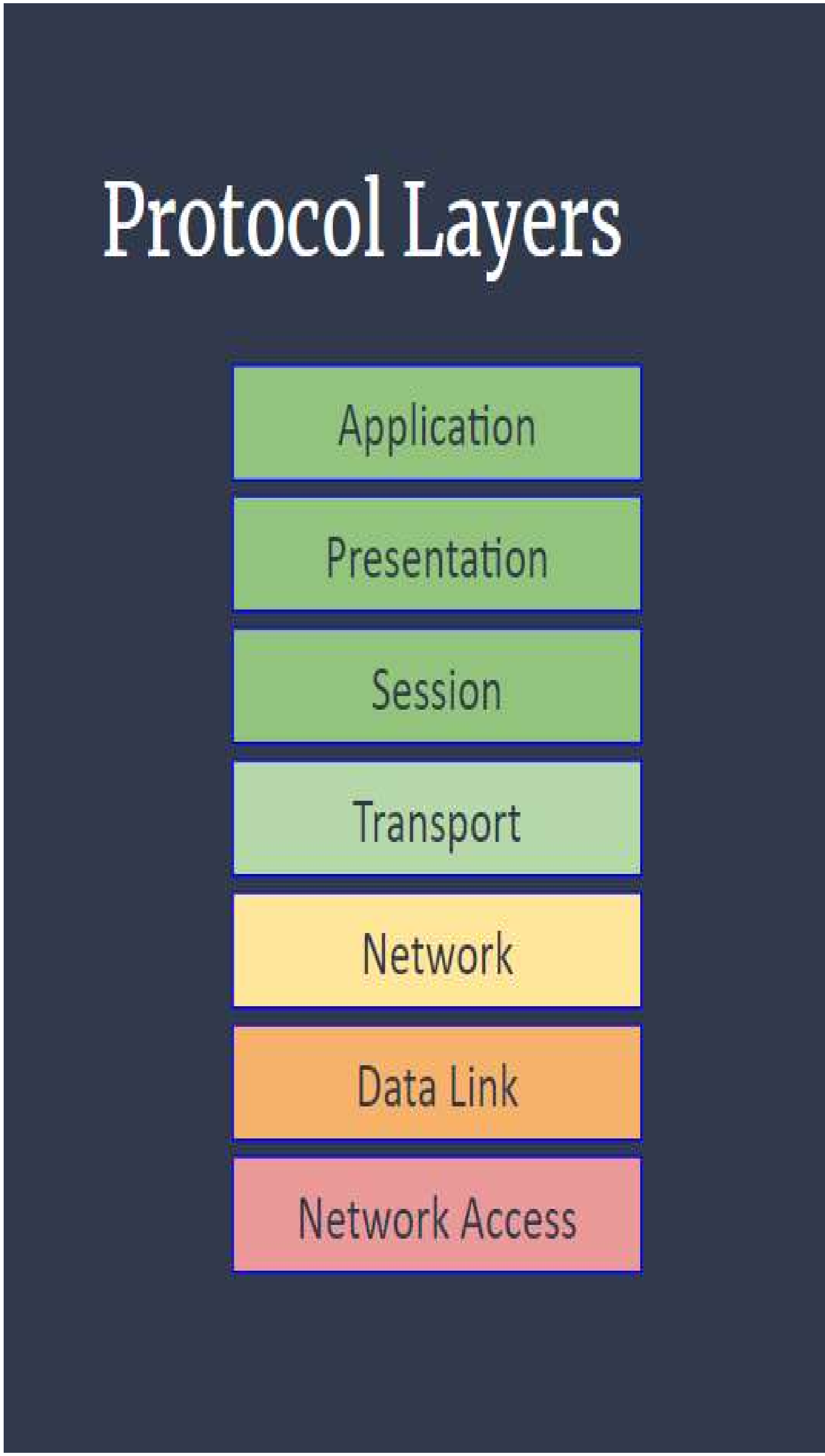
- Why would you want one?

Networking

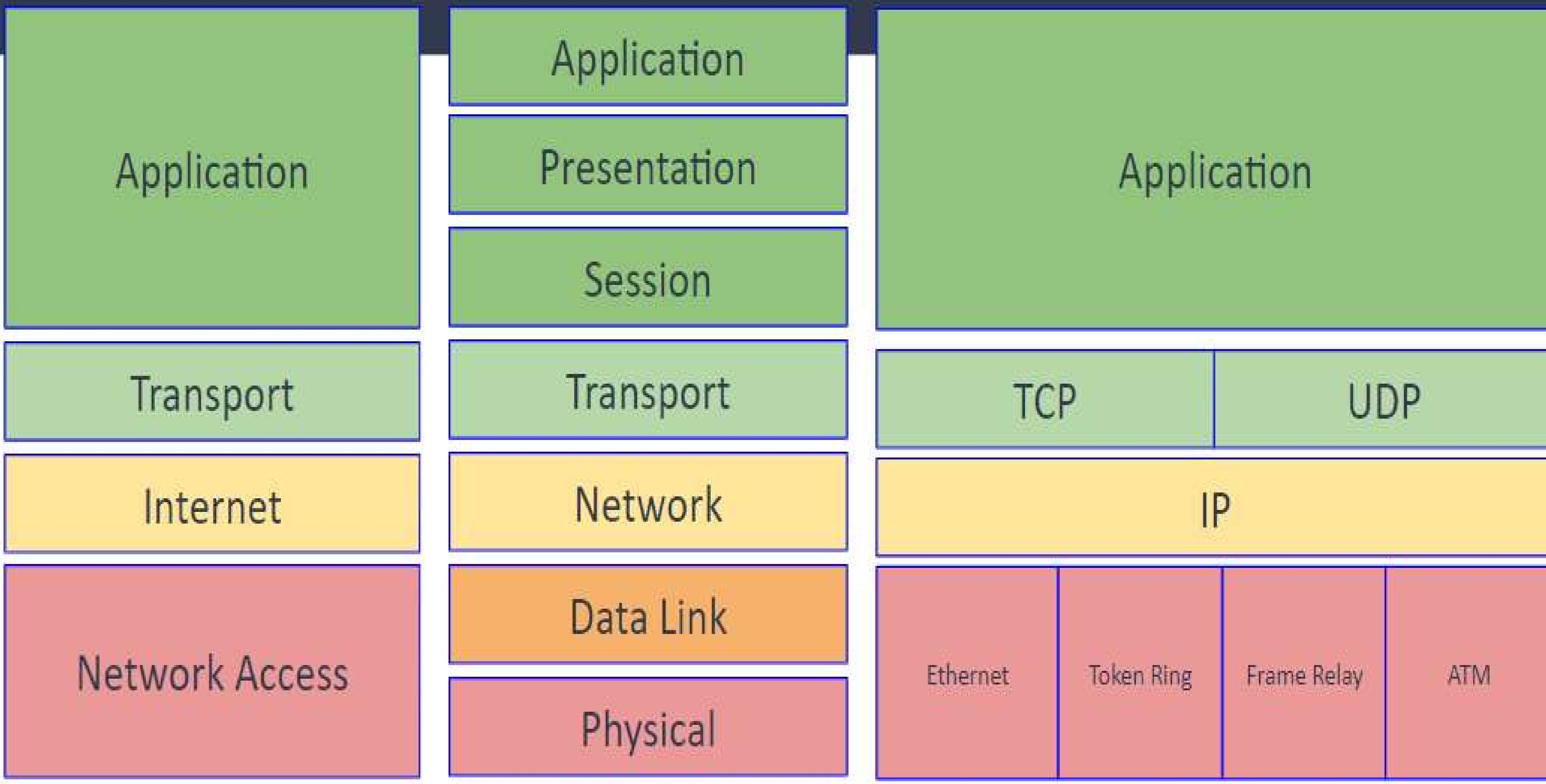


- A ***network*** comprises the hardware and software that provides two or more computers with the ability to ***communicate*** with each other
- Provided that one computer is listening, another computer can establish a ***connection***
- Both computers can then ***send*** and ***receive*** data over the connection
  - The data may be characters or binary data (bytes)

- Protocol ***layering*** is a common technique to simplify network designs by dividing them into functional ***layers***
  - For example, it is common to separate the functions of data ***delivery*** and connection ***management*** into separate layers
  - Each layer has its own ***protocol(s)***
- Each layer performs a specific ***purpose***, and doesn't need to know about the ***other layers*** or how to perform their purposes
- There are ***two*** major layered protocol designs in use today.
  - The DoD 4-Layer model
  - The OSI 7-Layer model



Network Models at a Glance

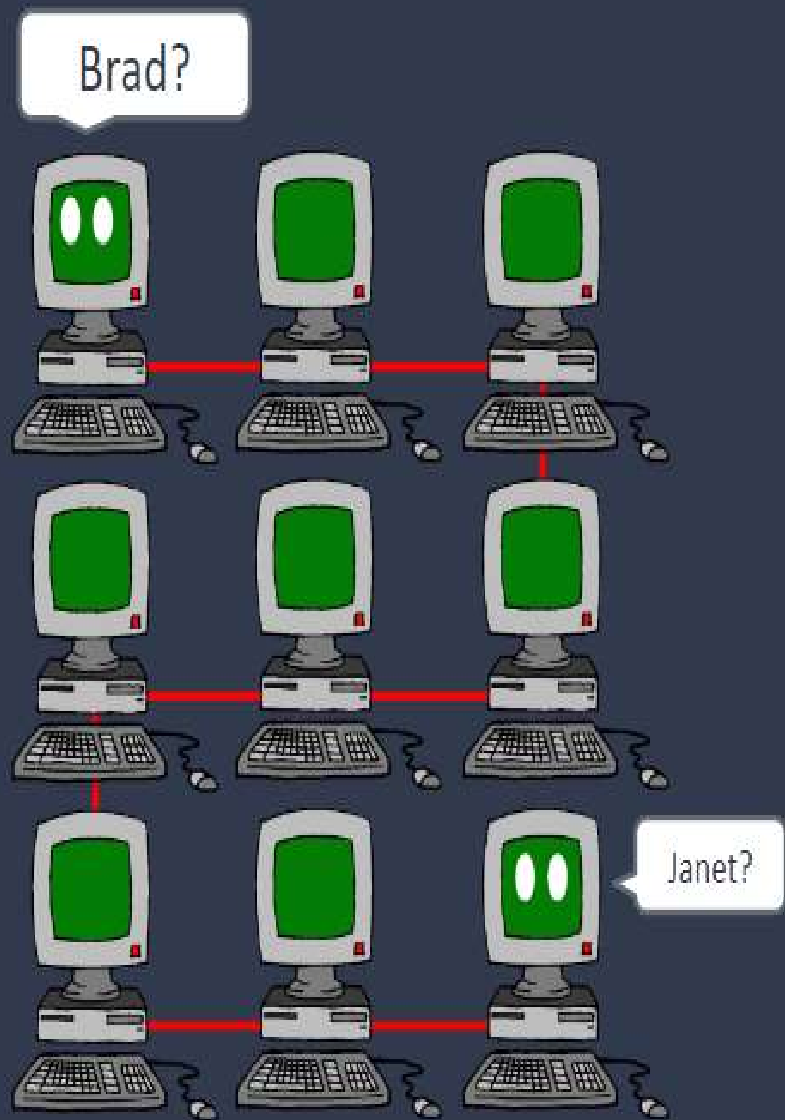


The Department of Defense (DoD) 4-Layer model. Also called the TCP/IP Model

The Open Systems Interconnection (OSI) 7-Layer Model.

TCP/IP is the main communications protocol used by the internet. It runs in the Transport and Network layers.

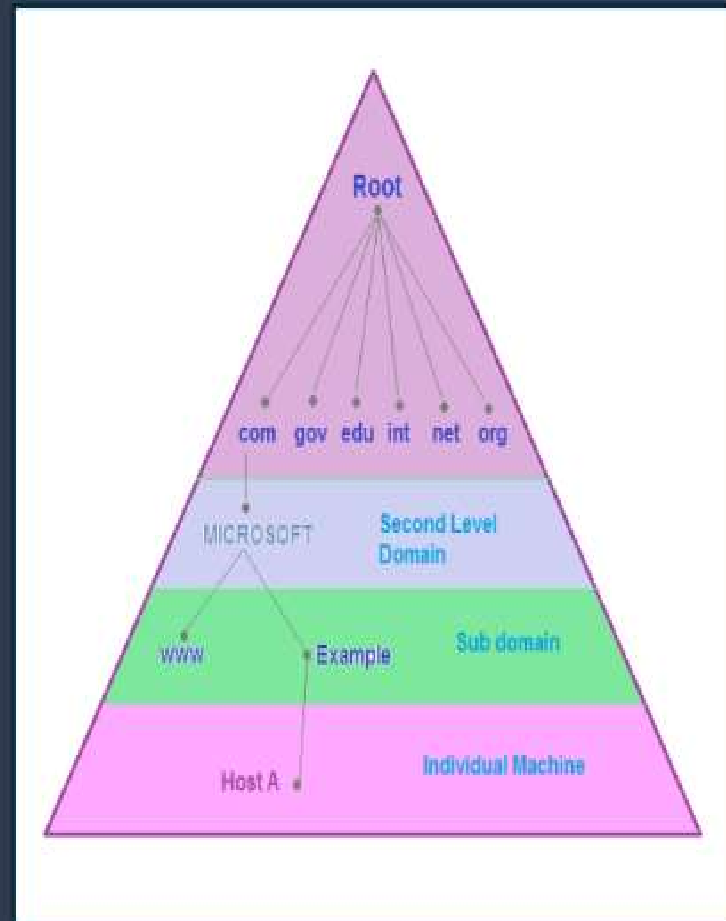
TCP/IP



- ***TCP/IP*** is the most widely used communications protocol on the internet
- It is actually two ***protocols*** working together
  - The ***Internet Protocol*** (IP) runs in the ***network*** layer and handles routing and relaying packets of information
  - The ***Transmission Control Protocol*** (TCP) establishes connections between two computers and helps to ensure that packets are ***delivered*** in order, reliably, and without corruption
- Most of what we will be talking about today will be how to send and receive data between ***two processes*** (on the same or different computers) using TCP/IP

Hostnames & DNS

- The IP address is ultimately used to send and receive data to and from a ***computer***
- But it is often the case that a computer is identified by its human readable ***hostname*** rather than its machine address
  - This is particularly the case when the network uses the dynamic host control protocol (***DHCP***), and a computer's address may change at any time
- A ***domain name server*** (DNS) on the network provides a service that ***maps*** a hostname to real address for the computer
- A hostname is used to ***look up*** the address before trying to communicate



DNSs on the internet are arranged into a hierarchy, with the *root* servers at the top and individual machines at the bottom.

Network Operating System

- Remote Login
  - `ssh nitron.se.rit.edu`
- Remote File Transfer
  - Each machine maintains its own files that can be transferred to other machines
  - ftp nitron.se.rit.edu
  - sftp nitron.se.rit.edu
- Cloud Storage
  - Examples: Dropbox, Google Drive
  - Requires the user to interact with files through a different paradigm than OS managed files

Distributed Operating System

- Distributed operating systems are designed to allow remote work to look the same as local work
- Data Migration
  - Non-local information is copied (in at least part) to the local system automatically
- Computation Migration
  - Executes functions remotely
  - Often done to be closer to resources needed
  - Could use RPC (remote procedure calls) or similar technology
    - RMI – Java
    - CORBA
    - Older systems that is less commonly used now
  - Could send messages to other machine which starts a new process
    - Web services
    - More common now

Process Migration

- Rather than offloading a single function to another machine, it is possible to run an entire process on one, or more, networked machines in a distributed operating system

- Reasons for this include
  - Load balancing: Keep nodes in a system evenly loaded
  - Computation speedup: Concurrency across nodes
  - Hardware preference: Specialized hardware on other nodes
  - Software preference: Specialized software on other nodes
  - Data access: Large amounts of data stored on another node, it can be cheaper to run the process co-located with the data

Robustness

- A distributed system has more points of failure than a single machine
- It is important if one component fails, it does not bring down the entire system, i.e. it is *Fault Tolerant*
- Fault tolerance can take many forms
- Failure Detection
  - Can I still talk to other components
- Reconfiguration
  - Once detected, make accommodations to prevent trying to use unavailable nodes
- Recover from Failure
  - Once the failed node is fixed, it needs to be added back into the system so it can be used again

Transparency

- Ideally, a distributed system should look the same as a conventional system
- Users should be able to access remote resources the same as local ones
- The users environment should be the same regardless of where they access the system
  - Home screen
  - Bookmarks
  - Available apps
  - Etc.

Salability

- Scalability involves increasing resources and the workload increases
- In a conventional system, this might be done by adding more resource
  - This is a manual physical process
  - Eventually you run into physical limitations
- In a networked distributed system, ideally it would be a simple task of adding new machines to the network
- While there is decreased efficiency due to networking delays, the ability to silently add more resources allows a system to scale much more uniformly

Question

- Based on the topics we discussed, is your operating system a distributed system?

- If yes, why?

- If no, does it have aspects of a distributed system and what are they?

Distributed File Systems

- Distributed file systems are a popular use of distributed computing.
- A file system provides file services to clients, which are ultimately maintained by some server
  - Same machine for local systems
  - One or more remote machines for distributed systems
- In a distributed system, files may reside across many machines
- The major performance aspect of a DFS is the amount of time to access storage
- Two standard models are used to accomplish this:
  - Client-Server Model
  - Cluster-Based DFS Model

The Client-Sever DFS Model

- Server stores both files and metadata of attached storage
- Clients request access to files using a well-known protocol
- Network File System (NFS):
  - Focus is simple and fast crash recovery
  - Server is stateless
  - Same operation can be issues repeatedly, with gives the same result (idempotent)
- Andrew File System (OpenAFS)
  - Focus on scalability
  - Requested files are stashed locally on the client
  - Updated to the server when they are closed
  - Much less communication than NFS
- Susceptible to single point of failure (Server)
  - Can be reduced/eliminated via computer clustering

The Cluster-Based DFS Model

- Cluster-based DFS is designed to increase fault-tolerance and scalability
- Google File System (GFS) is one example
  - Information is stored in redundant chunks across multiple servers
  - Metadata server lets client know where the chunks for the requested files are located
    - After that point, the client is responsible for collecting the needed information on the local system
  - Influenced by four main observations
    - Hardware failures are common and should be expected
    - Files stored on the system may be very large
    - Most files append rather than overwrite data
    - Redesigning the file system API increases flexibility
  - Requires applications to use the specified API

DFS Naming and Transparency

- Naming is the mapping between logical and physical objects
- In DFS systems, naming may include mapping to different machines or even redundant copies across multiple systems
- Naming Structures
  - Location Transparency: The name of the file does not reveal the physical storage
  - Location Independence: The filename need not change when the physical storage changes
- Naming Schemes
  - Unique identifier (URL is an example)
  - Attach remote directories to local (NFS)
  - Single global name structure for the entire system (OpenAFS)

Remote File Access

- On common way to transfer remote files is through a remote-service mechanism
  - Uses an RPC paradigm
- To ensure reasonable performance some type of caching scheme is needed
- In local systems cache is used to reduce disk I/O
- In distributed systems, it is used to reduce disk I/O and network traffic
- There are several aspects we need to consider
  - Caching Scheme
  - Cache Location
  - Cache Update Policy
  - Consistency

Basic Caching Scheme

- Simple concept, if the data is not already stored locally, then copy it from the server into a cache
- Access to data is always through the cached version
- When something changes the server needs to be updated
  - Cache-Consistency Problem
- Data can be cached in portions (blocks) of a file or the entire file
  - When using blocks, extra data is often collected to reduce subsequent server requests
- Block size and cache size are related
  - Larger blocks reduce the need for subsequent reads
  - However, fewer blocks increases the likelihood of a cache miss
  - Thus larger blocks benefit from a larger cache

Cache Location

- Where should the cached memory be stored?
- Disk
  - Increased reliability
  - Crash recovery may not require communication with the server
  - Slower
- Main Memory
  - Workstation can be diskless
  - Increased performance
  - Volatile memory is decreasing in cost relative to disk cost
  - Server caches will be in memory, thus allowing local and server to use the same mechanism
- NFS uses memory caching only
- OpenAFS uses both memory and disk

Cache Update Policy

- When to update the server copy can greatly impact system performance
- Write-through policy
  - Write as soon as a change happens
  - High reliability, low write performance
- Delayed-write policy (write-back caching)
  - Make changes locally (in the cache)
  - Occasionally write the cached changes
    - When the element is about to be cleared from the cache
    - At some interval (NFS)
- Write-on-close policy
  - Write when the local file is closed
  - Greatly increased performance for files that are kept open for a while
  - Used by OpenAFS

Consistency

- Client machines must determine if the local copy is still up to date with the server
- If not, a new copy must be added to the cache
- There are two common approaches to this determination:
- Client-initiated approach – the client checks validity
  - Every access
  - First access only
  - Some interval
- Server-initiated approach – the server tracks which clients have cached a file
  - If two, or more clients, cache the same location in conflicting modes the server disables caching
  - This results in a remote-service mode of operation
- DFS systems can greatly increase the complexity of maintaining consistency with the addition of meta-server and redundant chunks.