ECE454 Notes

June 15, 2019

1 Introduction

1.1 Distributed System

- **Distributed System**: A collection of autonomous computing elements that appears to its users as a single coherent system.

1.2 Motivations for Distributed Systems

1. Resource Sharing
2. Simplify Processes by Integrating Multiple Systems
3. Limitations in Centralized Systems: Weak/Unreliable
4. Distributed/Mobile Users

1.3 Goals for Distributed Systems

1. Resource Sharing
   - CPUs, Data, Peripherals, Storage.
2. Transparency
   - Access, Location, Migration, Relocation, Replication, Concurrency, Failure.
3. Open
   - Interoperability, Composability, Extensibility.
4. Scalable
   - Size, Geography, Administration.

1.4 Types of Distributed Systems

- Web Services
- High Performance Computing, Cluster Computing, Cloud Computing, Grid Computing
- Transaction Processing
- Enterprise Application Integration
- Internet of Things, Sensor Networks

1.5 Middleware

- **Middleware**: A layer of software that separates applications from the underlying platforms.
  - Supports Heterogeneous Computers/Networks.
  - *e.g.*: Communication, Transactions, Service Composition, Reliability.
  - **Single-System View**

1.6 Scaling Techniques

1. *Hiding Communication Latencies*: At Server vs. At Client?
2. *Partitioning*
3. *Replication*

1.7 Fallacies of Networked and Distributed Computing

1. Network is reliable.
2. Network is secure.
3. Network is homogeneous.
4. Topology is static.
5. Latency is zero.
6. Bandwidth is infinite.
7. Transport cost is zero.
8. There is only one administrator.

1.8 Shared Memory vs. Message Passing

- **Shared Memory**:
  - Less Scalable
  - Faster
  - CPU-Intensive Problems
  - Parallel Computing
- **Message Passing**:
  - More Scalable
  - Slower
  - Resource Sharing / Coordination Problems
  - Distributed Computing
- Apache Hadoop is an example of a hybrid computing framework that uses message passing at a broad-view and shared memory at a detailed-view.

1.9 Cloud and Grid Computing

- **IaaS**: Infrastructure as a Service
  - VM Computation, Block File Storage
- **PaaS**: Platform as a Service
  - Software Frameworks, Databases

- **SaaS**: Software as a Service
    - Web Services, Business Apps

1.10 Transaction Processing Systems

- **Transaction Processing Monitor**: Coordinates Distributed Transactions

2 Architectures

2.1 Definitions

- **Component**: A modular unit with well-defined interfaces.
- **Connector**: A mechanism that mediates communication, coordination, or cooperation among components.
- **Software Architecture**: Organization of software components.
- **System Architecture**: Instantiation of software architecture in which software components are placed on real machines.
- **Autonomic System**: Adapts to its environment by monitoring its own behavior and reacting accordingly.

2.2 Architectural Styles

- Layered
    - *Note: Assignment Topic*
- Object-Based
- Data-Centered
- Event-Based

2.3 Layered Architecture

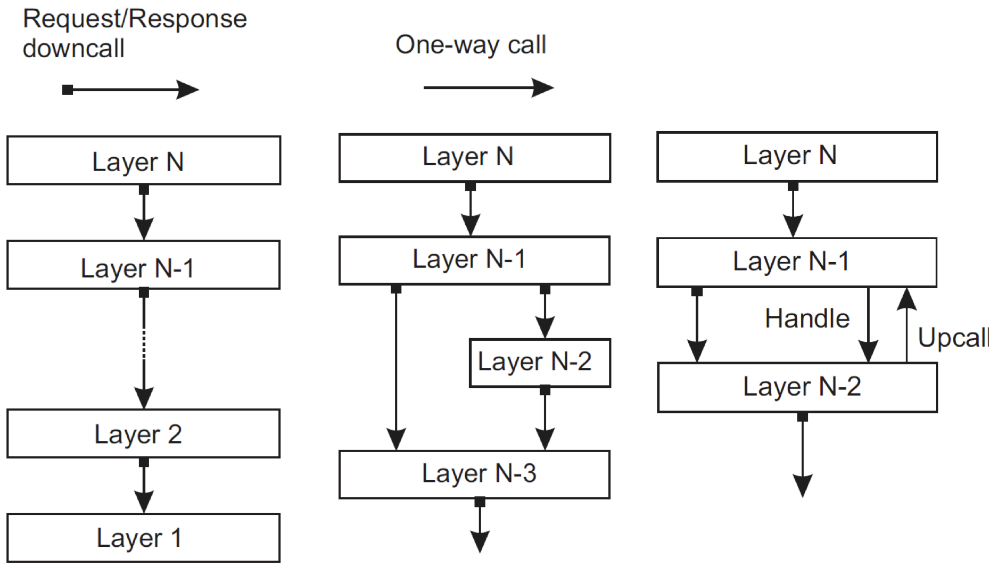
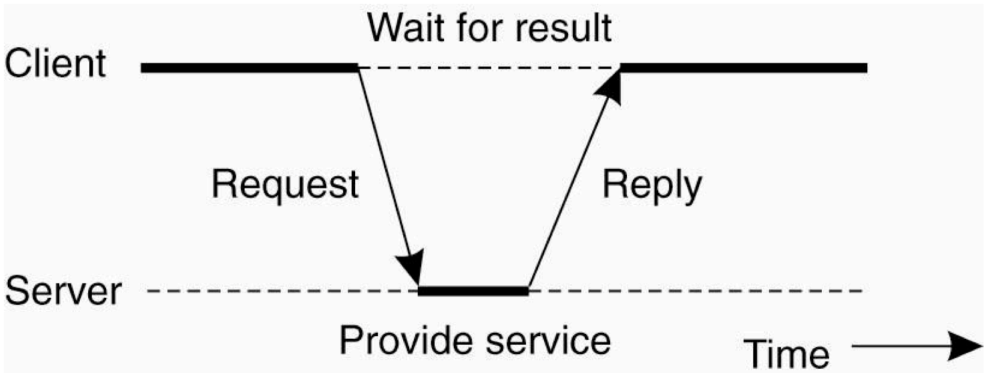


Figure 2.1: (a) Pure layered organization. (b) Mixed layered organization. (c) Layered organization with upcalls (adopted from [Krakowiak, 2009]). 5

Layers

- **Examples**:
  - Database Server, Application Server, Client
  - SSH Server, SSH Client
- Requests Flow Down Stack
- Responses Flow Up Stack
- *Handle-Upcall*: Async Notification
  - Subscribe with Handle
  - Publish with Upcall

2.4 Client-Server Interactions



Client-Server Interactions

- *Bolded Lines* = Busy
- *Dashed Lines* = Idle
- **Client**: Initiates with a Request
- **Server**: Follows with a Response
- **Total Round-Trip Time**: $(N-1) \times t_{\text{Request-Response}}$
  - Layering can reduce the amount of processing time per layer, but the additional communication overhead between the layers introduces diminishing returns.
- An intermediate layer can be both a client and a server to the others.

2.5 Multi-Tiered Architecture

- Logical Software Layers $\mapsto$ Physical Tiers
  - *Trade-Offs*: Ease of Maintenance vs. Reliability

2.6 Horizontal vs. Vertical Distribution

- **Vertical Distribution**: When the logical layers of a system are organized as separate physical tiers.
  - *Performance*: High.
  - *Scalability*: Low.
  - *Dependability*: Low-Medium.
- **Horizontal Distribution**: When one logical layer is split across multiple machines - **sharding**.
  - *Performance*: Low.
  - *Scalability*: High.
  - *Dependability*: Medium-High.

2.7 Object-Based Architecture

- In an object-based architecture, components communicate using remote object references and method calls.

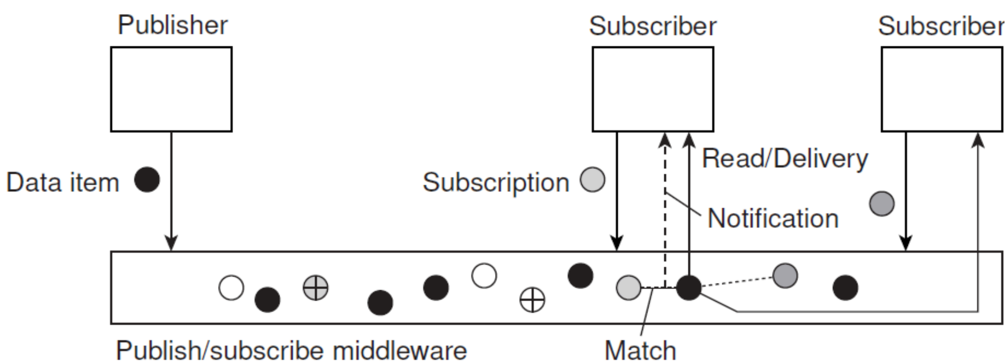
2.8 Problems with Object-Based Architecture

- Complex Communication Interfaces
- Complex Communication Costs
- Not Scalable
- Not Language Agnostic

2.9 Data-Centered Architecture

- In a data-centered architecture, components communicate by accessing a shared data repository.

2.10 Event-Based Architecture



Publish/Subscribe Middleware

- In an event-based architecture, components communicate by propagating events using a publish/subscribe system.

2.11 Handling Asynchronous Delivery Failure

- *At-Least Once Delivery*: Do Retransmit
- *At-Most Once Delivery*: Do Not Retransmit
- *Exactly Once Delivery*: Unknown/Unachievable

2.12 Peer-to-Peer Systems

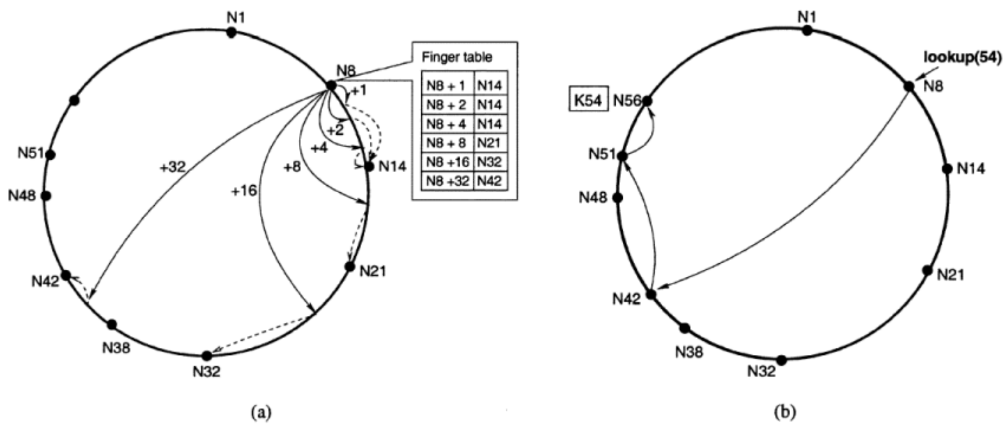


Fig. 4. (a) Finger table entries for node 8. (b) Path of a query for key 54 starting at node 8, using the algorithm in Fig. 5.

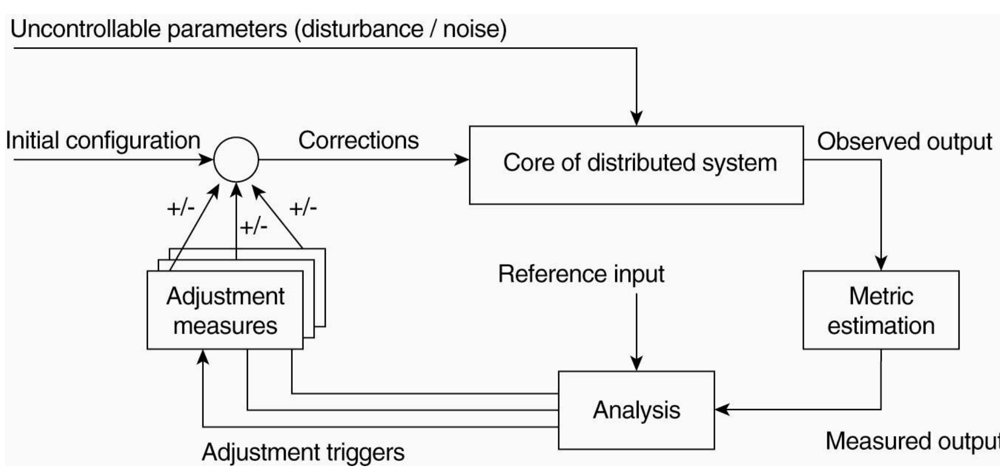
Chord's Finger Table

- In a peer-to-peer system, decentralized processes are organized in an overlay network that defines a set of communication channels.
- In a peer-to-peer, distributed hash table, a keyspace is represented by a consistent hash ring on top of which nodes partition ranges amongst themselves.
- The mappings of partition ranges to nodes are maintained by a finger table which can be queried in a logarithm process.

2.13 Hybrid Architectures

- BitTorrent is an example of a hybrid architecture combining a client-server architecture and a peer-to-peer architecture.

2.14 Self-Management



Self-Management Systems

- In self-management, systems use a feedback control loop that monitors system behaviors and adjusts system operations.
- **Assignment Note**: Useful for Unknown Assignment

3 Processes

3.1 IPC

- **Inter-Process Communication (IPC)**: Expensive b/c Context Switching

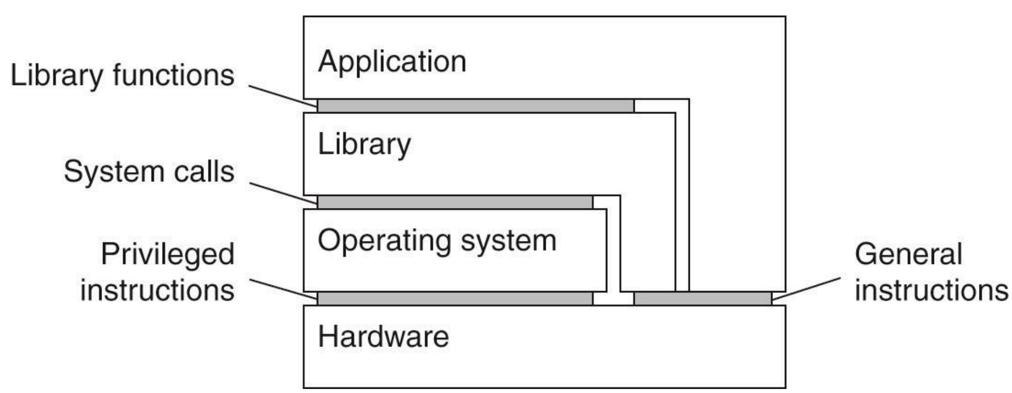
3.2 Threads

- Typically, an operating system kernel support multi-threading through **lightweight processes (LWP)**.
- **Assignment Note**: Do Not Spawn Too Many Threads

3.3 Multi-Threaded Servers

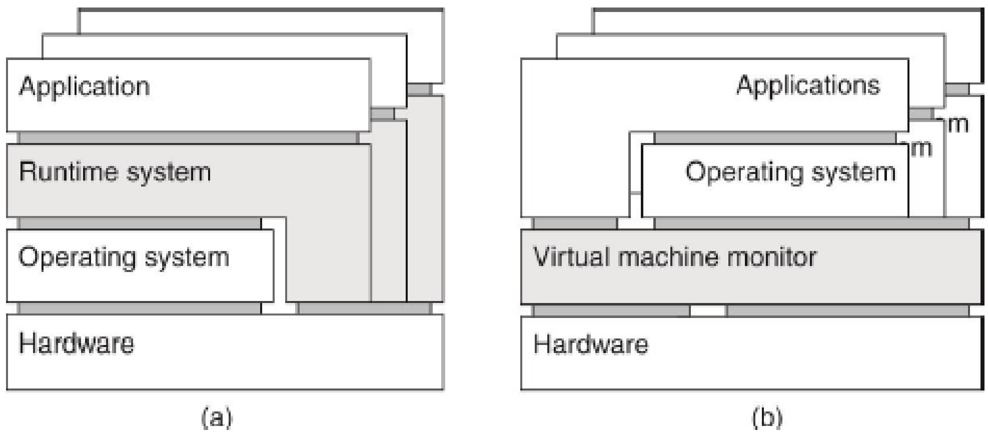
- **Dispatcher/Worker Design**: A dispatcher thread receives requests from the network and feeds them to a pool of worker threads.
- **Assignment Note**: Useful for Assignment 1 & Partition into Sequential Work and Parallel Work

3.4 Hardware and Software Interfaces



Hardware and Software Interfaces

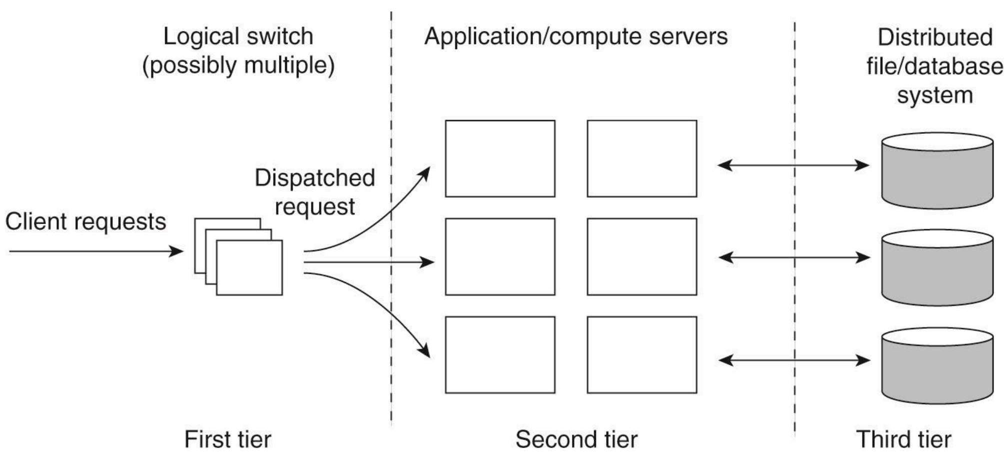
3.5 Virtualization



VMs

- **Advantage**:
  - Portability
  - Live Migration of VMs
  - Replication for Availability/Fault Tolerance
- **Disadvantage**:
  - Performance

3.6 Server Clusters

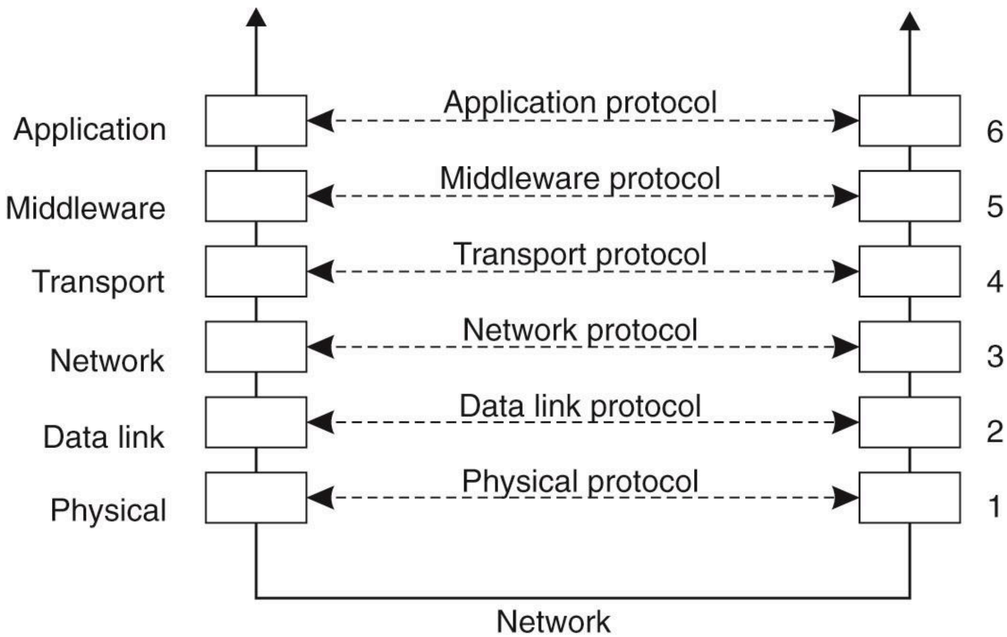


Three Physical Tier

- **Assignment Note**: Useful for Assignment 2

4 Communication

4.1 Layered Network Model

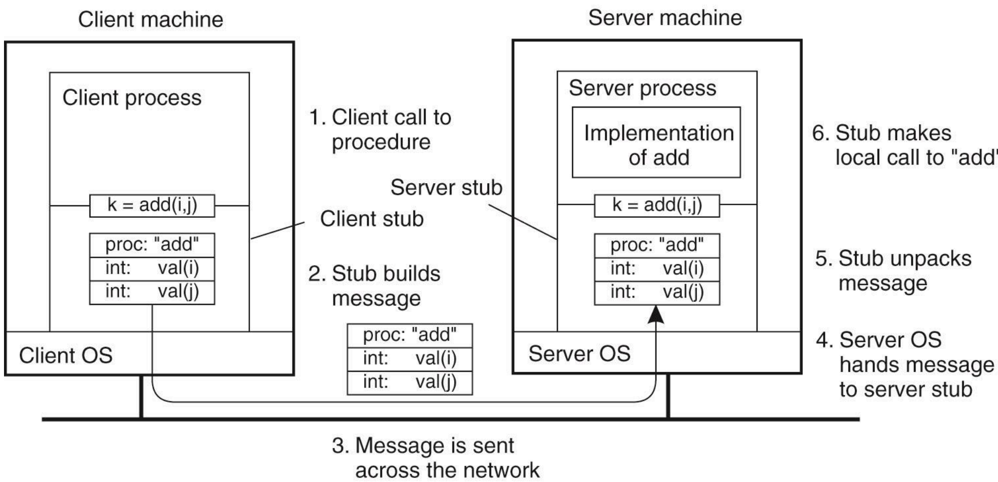


Layered Network Model

4.2 Remote Procedure Calls

- **Remote Procedure Calls**: A transient communication abstraction implemented using a client-server protocol.
- **Client Stub**: Translate a RPC on the client.
- **Server Stub**: Translate a RPC on the server.

4.3 Steps of a RPC



Steps of a RPC

1. The client process invokes the client stub using an ordinary procedure call.
2. The client stub builds a message and passes it to the client's OS.
3. The client's OS sends the message to the server's OS.
4. The server's OS delivers the message to the server stub.
5. The server stub unpacks the parameters and invokes the appropriate service handler in the server process.
6. The **service handler** does the work and returns the result to the server stub.
7. The server stub packs the result into a message and passes it to the server's OS.
8. The server's OS sends the message to the client's OS.
9. The client's OS delivers the message to the client stub.
10. The client stub unpacks the result and returns it to the client process.

- **Parameter Marshalling**: Packing Parameter $\rightarrow$ Message
  - Processor Architectures, Network Protocols, and VMs $\Longrightarrow$ **Little-Endian** vs. **Big-Endian**
- **Number of System Calls**: 4
  1. Client Process $\rightarrow$ Client OS Socket
  2. Server OS Socket $\rightarrow$ Server Process
  3. Server Process $\rightarrow$ Server OS Socket
  4. Client OS Socket $\rightarrow$ Client Process

4.4 Defining RPC Interfaces

- **Interface Definition Language (IDL)**: Specify RPC Signatures $\rightarrow$ Client/Server Stubs
  - High-Level Format
  - Parameter Ordering
  - Byte Sizes

4.5 Synchronous vs. Asynchronous RPCs

- **Synchronous RPC**: The client blocks to wait for the return value.
- **Asynchronous RPC**: The client blocks to wait for the server acknowledgement of the receipt of the request.
- **One-Way RPC**: The client does not block to wait.

4.6 Message Queuing Model

Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block
Notify	Install a handler to be called when a message is put into the specified queue

Message Queue Interface

- **Message Queue**: Alternative to RPCs
- **Persistent Communication**: Loose Coupling between Client/Server
  - *Advantage*: Resilient to Client/Server Hardware Failure
  - *Disadvantage*: Guaranteed Delivery = Impossible
- **Message-Oriented Middleware (MOM)**: Asynchronous Message Passing

4.7 Process Coupling

- **Referential Coupling**: When one process explicitly references another.
  - *Positive Example*: RPC client connects to server using an IP address and a port number
  - *Negative Example*: Publisher inserts a news item into a pub-sub system without knowing which subscriber will read it.
- **Temporal Coupling**: Communicating processes must both be up and running.
  - *Positive Example*: A client cannot execute a RPC if the server is down.
  - *Negative Example*: A producer appends a job to a message queue today, and a consumer extracts the job tomorrow.

4.8 RPC vs. MOM

4.9 RPC

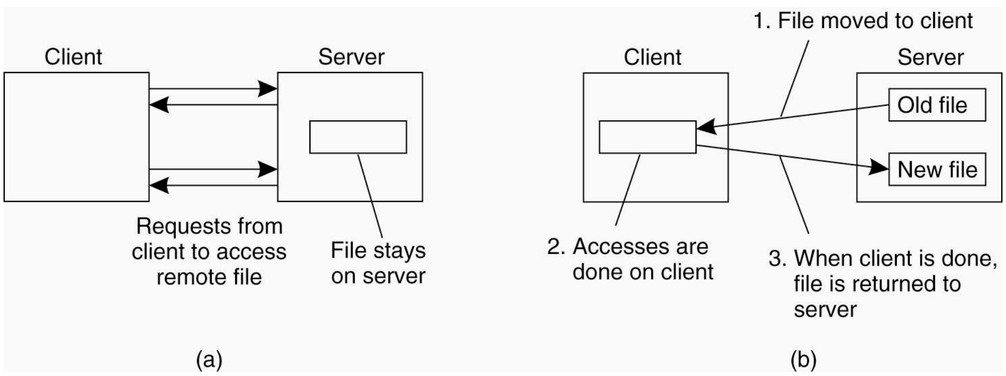
- Used mostly for two-way communication, particularly where the client requires immediate response from the server.
- The middleware is linked into the client and the server processes.
- Tighter coupling means that server failure can prevent client from making progress.

4.10 MOM

- Used mostly for one-way communication where one party does not require an immediate response from another.
- The middleware is a separate component between the sender/publisher/producer and the receiver/subscriber/consumer.
- Looser coupling isolates one process from another which contributes to flexibility and scalability.

5 Distributed File Systems

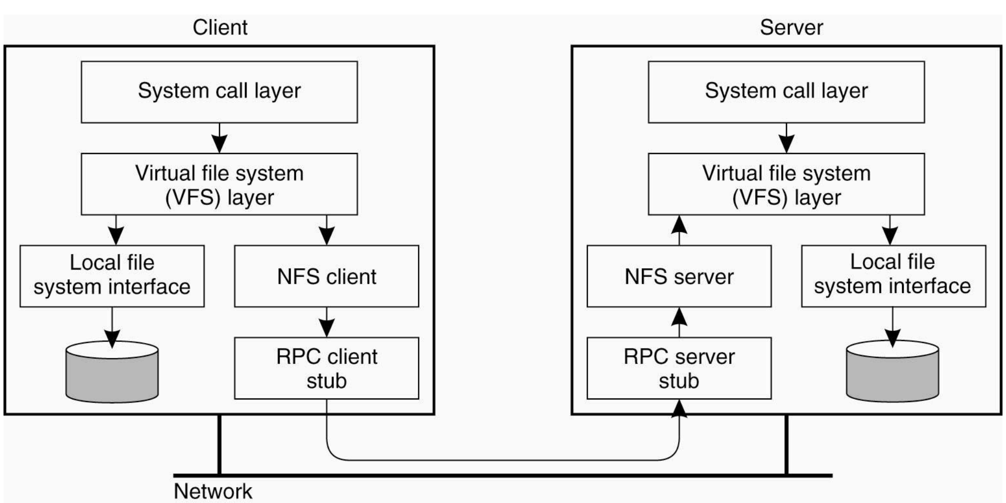
5.1 Accessing Remote Files



DFS Models

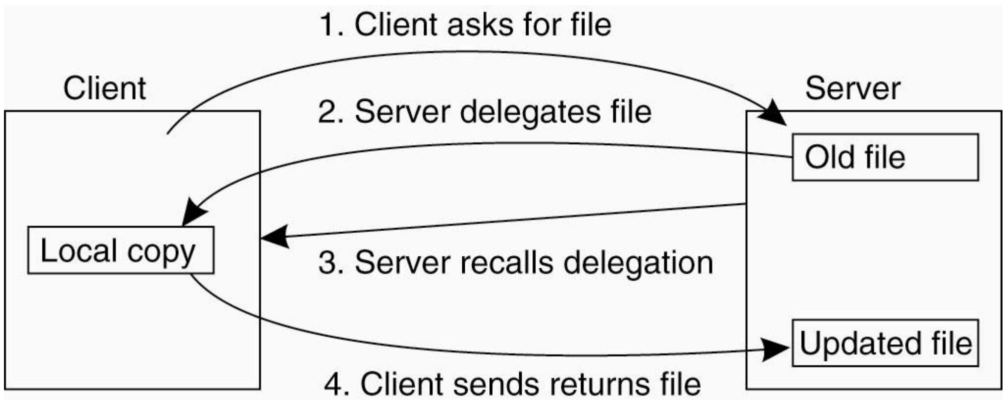
- **Remote Access Model**
- **Upload/Download Model**

5.2 Network File System (NFS)



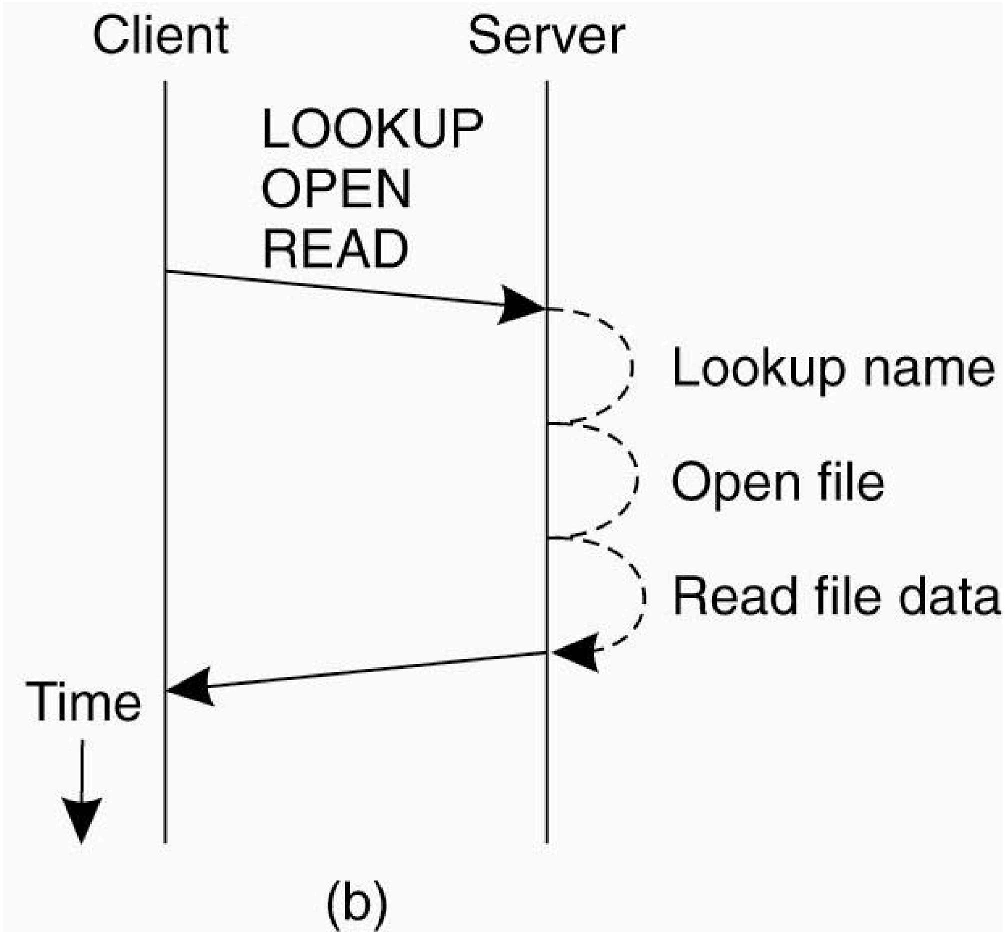
Overview of NFS

- *Supports Client-Side Caching*
  - Modifications are flushed to the server when the client closes the file.
  - Consistency is implementation dependent.



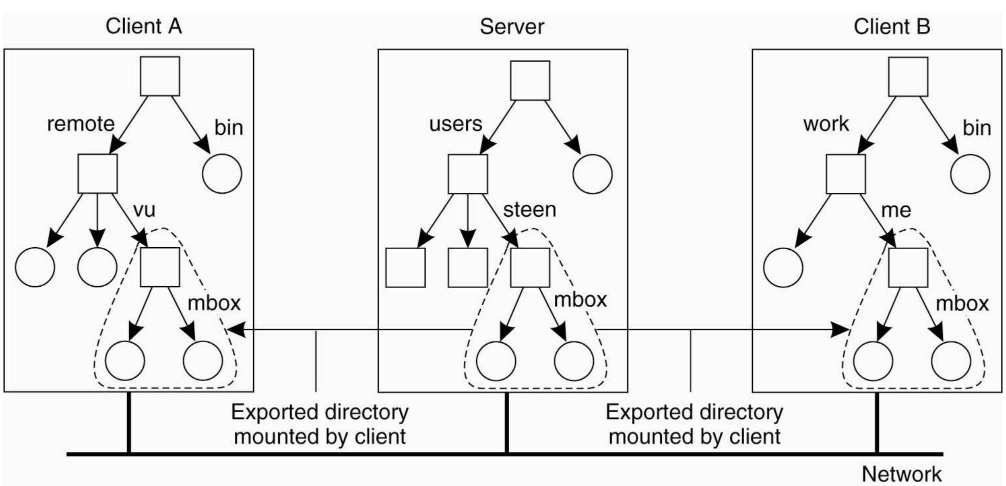
Authority Delegation

- *Supports Authority Delegation*
  - A server can delegate authority to a client and recall it through a callback mechanism.



Compound Procedure

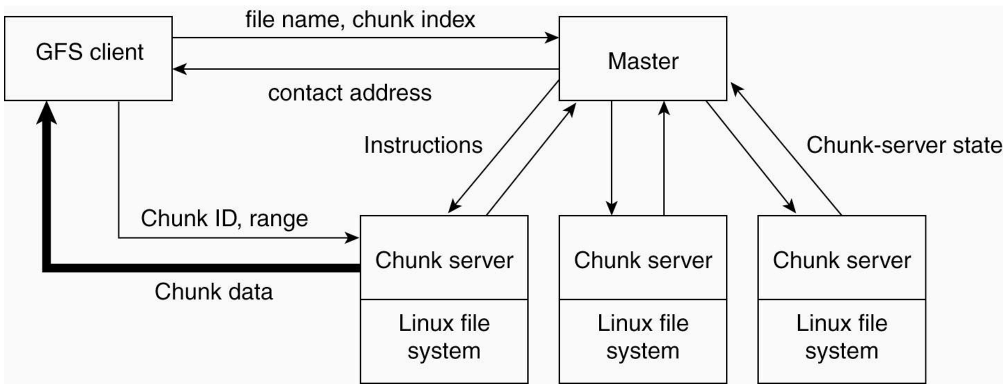
- *Supports Compound Procedures*
  - Multiple Round Trips to Single Round Trip



Partial Exports

- *Supports Partial Exports*

5.3 Google File System (GFS)



Google File System

- **GFS**: A distributed file system that stripes files across inexpensive commodity servers without RAID.
  - *Layered Above Linux File System*
  - *Fault Tolerance Through Software*
- **GFS Master**: *Stores Metadata About Files/Chunks*
  - *Metadata Cache in Main Memory*
  - *Updated Log in Local Storage*
  - *Periodically Polls Client Servers for Consistency*

5.4 Reading a File

1. A client sends the file name and chunk index to the master.
2. The master responds with a contact address.
3. The client then pulls data directly from a chunk server, bypassing the master.

5.5 Updating a File

1. The client pushes its updates to the nearest chunk server holding the data.
2. The nearest chunk server pushes the update to the next closest chunk server holding the data, and so on.
3. When all replicas have received the data, the primary chunk server assigns a sequence number to the update operation and passes it on to the secondary chunk servers.
4. The primary replica informs the client that the update is complete.

5.6 File Sharing Semantics

Method	Comment
UNIX semantics	Every operation on a file is instantly visible to all processes
Session semantics	No changes are visible to other processes until the file is closed
Immutable files	No updates are possible; simplifies sharing and replication
Transactions	All changes occur atomically

File Sharing Semantics