Outline

- Definition of a Distributed System
- Goals of a Distributed System
- Types of Distributed Systems

  -Distributed Computing System.

  -Distributed Information System.
- Architecture:
  - Architectural Styles
  - System Architecture

What Is A Distributed System?

- A collection of independent computers that appears to its users as a single coherent system.
- Features:
  - Message-based communication
  - Each runs its own local OS
- Ideal: to present a single-system image:
  - The distributed system “looks like” a single computer rather than a collection of separate computers.

Distributed System Characteristics

- To present a single-system image:
  - Hide internal organization, communication details
- Easily expandable
  - Adding new computers is hidden from users
- Continuous availability
  - Failures in one component can be covered by other components
- Supported by middleware

Definition of a Distributed System

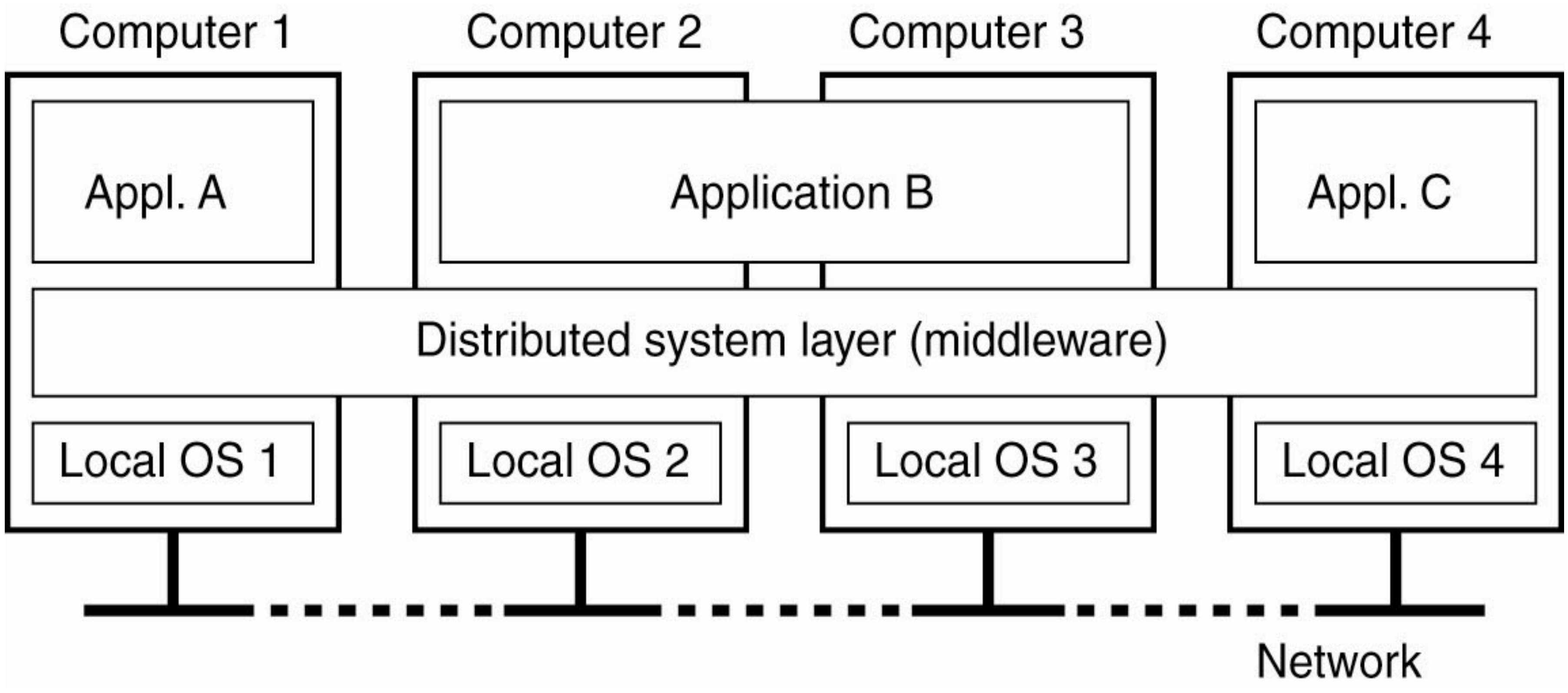


Figure 1-1. A distributed system organized as middleware.
The middleware layer runs on all machines, and offers a uniform interface to the system

Distributed System Goals

- Resource Accessibility
- Distribution Transparency
- Openness
- Scalability

Goal 1 – Resource Accessibility

- Support user access to remote resources (printers, data files, web pages, CPU cycles) and the fair sharing of the resources

- Economics of sharing expensive resources

- Example: Sharing of a printer in small office

- Resource sharing introduces security problems.

Goal 2 – Distribution Transparency

- Software hides some of the details of the distribution of system resources.
  - Makes the system more user friendly.
- A distributed system that appears to its users & applications to be a single computer system is said to be *transparent*.
  - Users & apps should be able to access remote resources in the same way they access local resources.
- Transparency has several dimensions.

Types of Transparency

Transparency	Description
Access	Hide differences in data representation & resource access (enables interoperability)
Location	Hide location of resource (can use resource without knowing its location)
Migration	Hide possibility that a system may change location of resource (no effect on access)
Replication	Hide the possibility that multiple copies of the resource exist (for reliability and/or availability)
Concurrency	Hide the possibility that the resource may be shared concurrently
Failure	Hide failure and recovery of the resource. How does one differentiate betw. slow and failed?
Relocation	Hide that resource may be moved <u>during use</u>

Figure 1-2. Different forms of transparency in a distributed system (ISO, 1995)

Goal 3 - Openness

- An **open distributed system** "…offers services according to standard rules that describe the syntax and semantics of those services."

- **Interface Definition/Description Languages (IDL):** used to describe the interfaces between software components, usually in a distributed system
  - Support communication between systems using different OS/programming languages; e.g. a C++ program running on Windows communicates with a Java program running on UNIX
  - Communication is usually RPC-based. concurrent

Open Systems Support …

- **Interoperability**: the ability of two different systems or applications to work together
  - A process that needs a service should be able to talk to any process that provides the service.
- **Portability**: an application designed to run on one distributed system can run on another system which implements the same interface.
- **Extensibility**: Easy to add new components, features.

Goal 4 - Scalability

- Dimensions that may scale:
  - With respect to size
  - With respect to geographical distribution
  - With respect to the number of administrative organizations spanned.

Size Scalability

- Scalability is negatively affected when the system is based on
  - Centralized server: one for all users
  - Centralized data: a single data base for all users
    - Complete knowledge: good
    - Time and network traffic: bad

Geographic Scalability

- Early distributed systems ran on LANs, relied on **synchronous communication(blocking client).**
  - May be too slow for wide-area networks
  - Wide-area communication is unreliable, point-to-point;
  - Unpredictable time delays may even affect correctness
- Centralized components + wide-area communication: waste of network bandwidth

Scalability - Administrative

- Different domains may have different policies about resource usage, management, security, etc.

- Trust often stops at administrative boundaries
  - Requires protection from malicious attacks

Scaling Techniques

- Scalability affects performance more than anything else.

- Three techniques to improve scalability:
  - Hiding communication latencies
  - Distribution
  - Replication

Hiding Communication Delays

- Structure applications to use **asynchronous communication** (no blocking for replies)
  - While waiting for one answer, do something else; *e.g.*, create one thread to wait for the reply and let other threads continue to process or schedule another task
- Download part of the computation to the requesting platform to speed up processing
  - i.e., shorten the wait times

Hiding Communication Delays

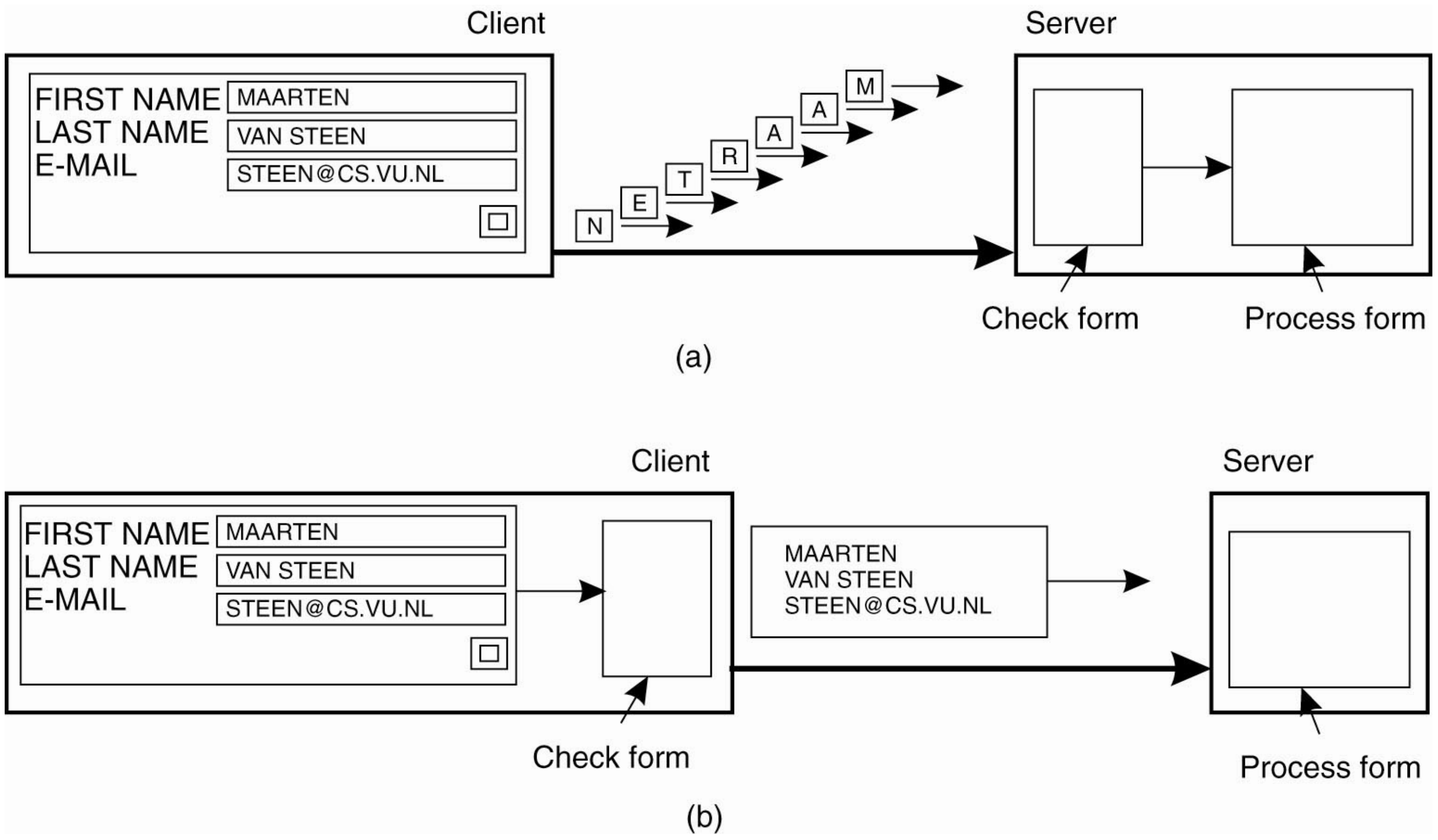


Figure 1-4. The difference between letting (a) a server or (b) a client check forms as they are being filled.

Distribution

- Instead of one centralized service, divide into parts and distribute geographically

- Each part handles one aspect of the job

  - Example: DNS namespace is organized as a tree of domains; each domain is divided into zones; names in each zone are handled by a different name server
  - WWW consists of many (millions?) of servers

Continue…

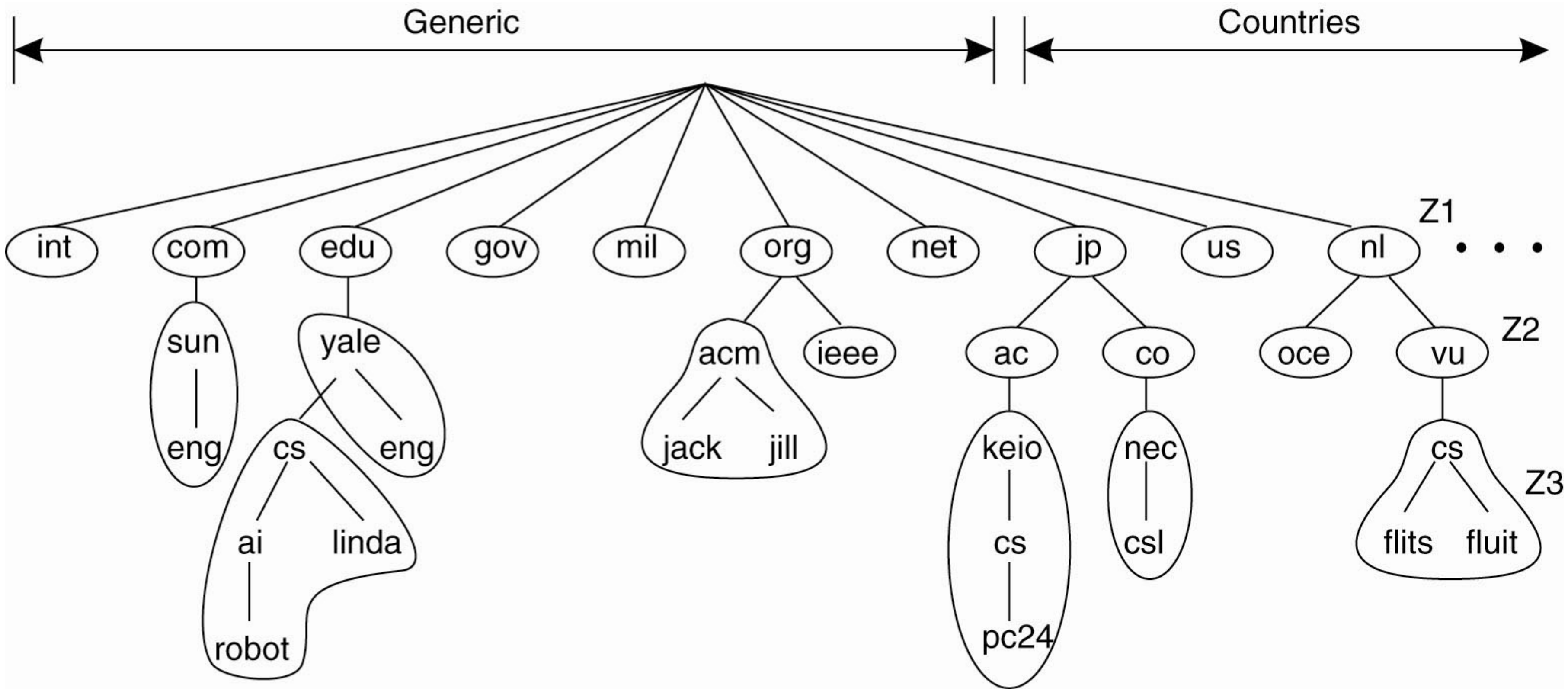


Figure 1-5. An example of dividing the DNS name space into zones.

Replication

- Replication: multiple identical copies of something
  - Replicated objects may also be distributed, but aren't necessarily.

- Replication
  - Increases availability
  - Improves performance through load balancing
  - May avoid latency.

Caching

- Caching is a form of replication
  - Normally creates a (temporary) replica of something closer to the user
- Replication is often more permanent
- User (client system) decides to cache, server system decides to replicate
- Both lead to **consistency** problems

Issues/Pitfalls of Distribution

Assumptions that everyone makes when developing distributed application for first time:

- The network is reliable.
- The network is secure.
- The network is homogeneous.
- The topology does not change.
- Latency is zero.
- Bandwidth is infinite.
- Transport cost is zero.
- There is one Administrator.

Types of Distributed Systems

- Distributed Computing Systems
  - Cluster Computing
  - Grid Computing
  - Cloud Computing
- Distributed Information Systems
  - Transaction Processing Systems
  - Enterprise Application Integration.
- Distributed Embedded Systems

Cluster Computing

- A collection of similar processors
- Parallel computing capabilities using inexpensive PC hardware

Cluster Types & Uses

- High Performance Clusters (HPC)
  - run large parallel programs
  - Scientific, military, engineering apps; weather modeling
- Load Balancing Clusters
  - Front end processor distributes incoming requests
- High Availability Clusters (HA)
  - Provide back up systems
  - May be fault tolerant.

Clusters

- Master-slave paradigm
  - One processor is the master; allocates tasks to other processors, handles interface to users
  - Master has libraries to handle message-based communication or other features.

Cluster Computing Systems

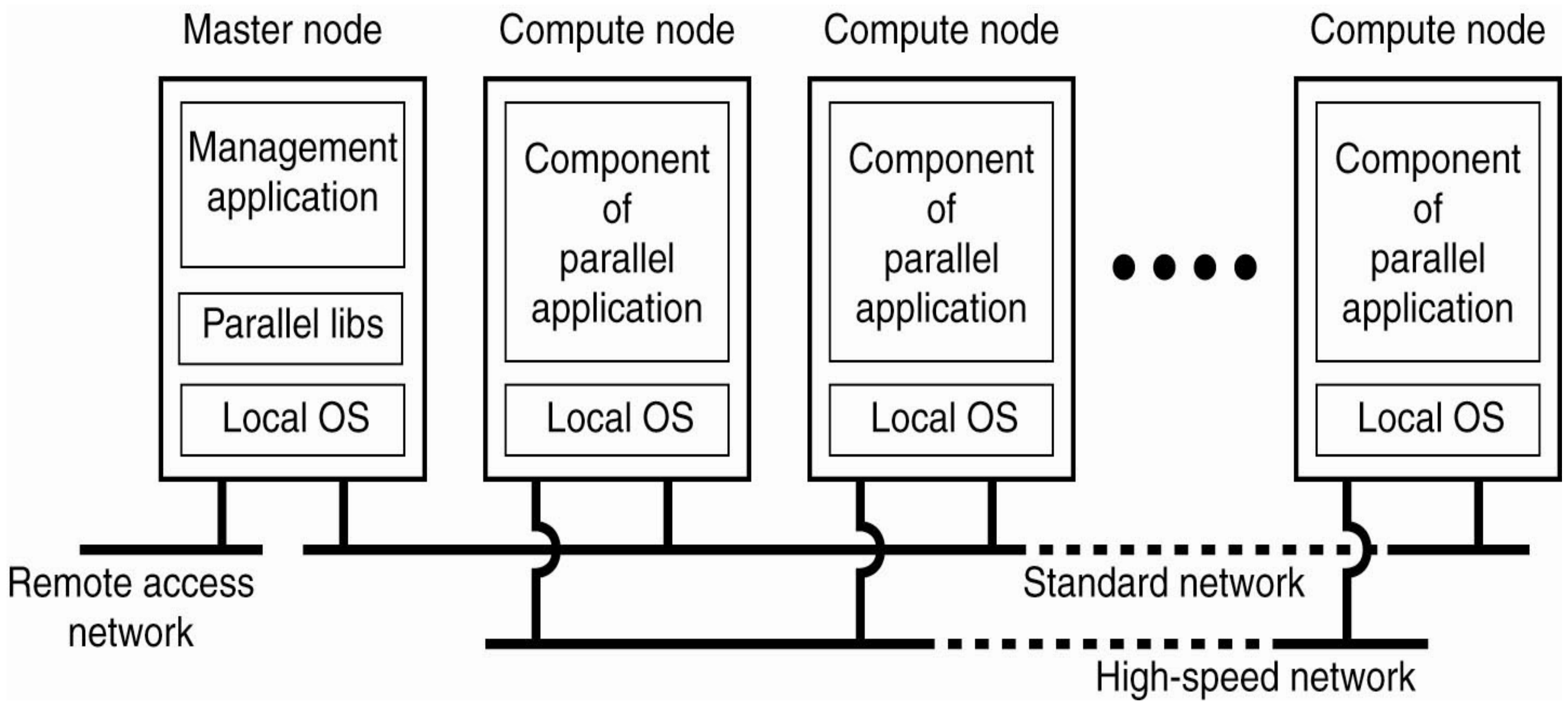


Figure 1-6. An example of a cluster computing system

Grid Computing Systems

- Highly heterogeneous
- Grids support **virtual organizations**

Architecture for Grid Systems*

- **Fabric layer**: interfaces to local resources at a specific site
- **Connectivity layer**: protocols to support usage of *multiple resources* for a single application;
- **Resource layer** manages a *single resource,* using functions supplied by the connectivity layer
- **Collective layer:** resource discovery, allocation, scheduling, etc.
- **Applications**: use the grid resources

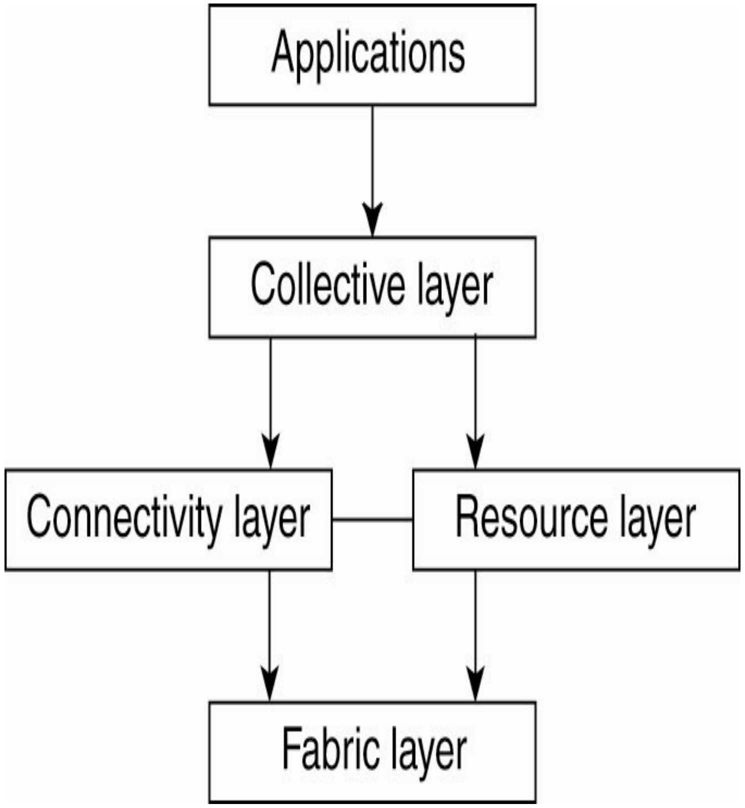


Figure 1-7. A layered architecture for grid computing systems

Types of Distributed Systems

- Distributed Computing Systems
  - Clusters
  - Grids
  - Clouds
- Distributed Information Systems

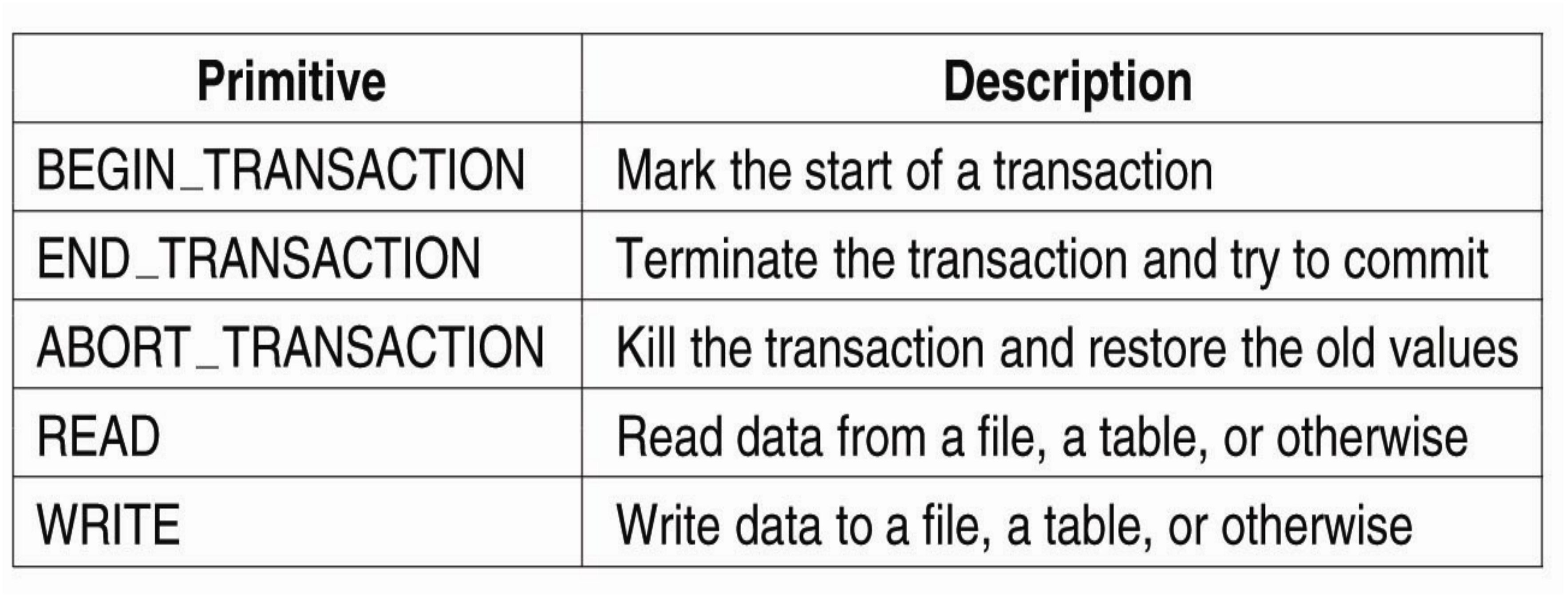
Distributed Information Systems

- Business-oriented.
- Systems to make a number of separate network applications interoperable .
- Two types discussed here:
  - Transaction processing systems
  - Enterprise application integration (EAI)

Transaction Processing Systems

- Provide a highly structured client-server approach for database applications
- Transactions

Transaction Processing Systems



Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

Figure 1-8. Example primitives for transactions

Transactions

- Obey the ACID properties:
  - Atomic: all or nothing
  - Consistent: invariants are preserved
  - Isolated : serializable
  - Durable: committed operations can’t be undone

Nested Transactions

- A nested transaction is a transaction within another transaction (a sub-transaction)
  - Example: a transaction may ask for two things (e.g., airline reservation info + hotel info)

- Primary transaction waits for the results.
  - While children are active parent may only abort, commit, or spawn other children

Transaction Processing Systems

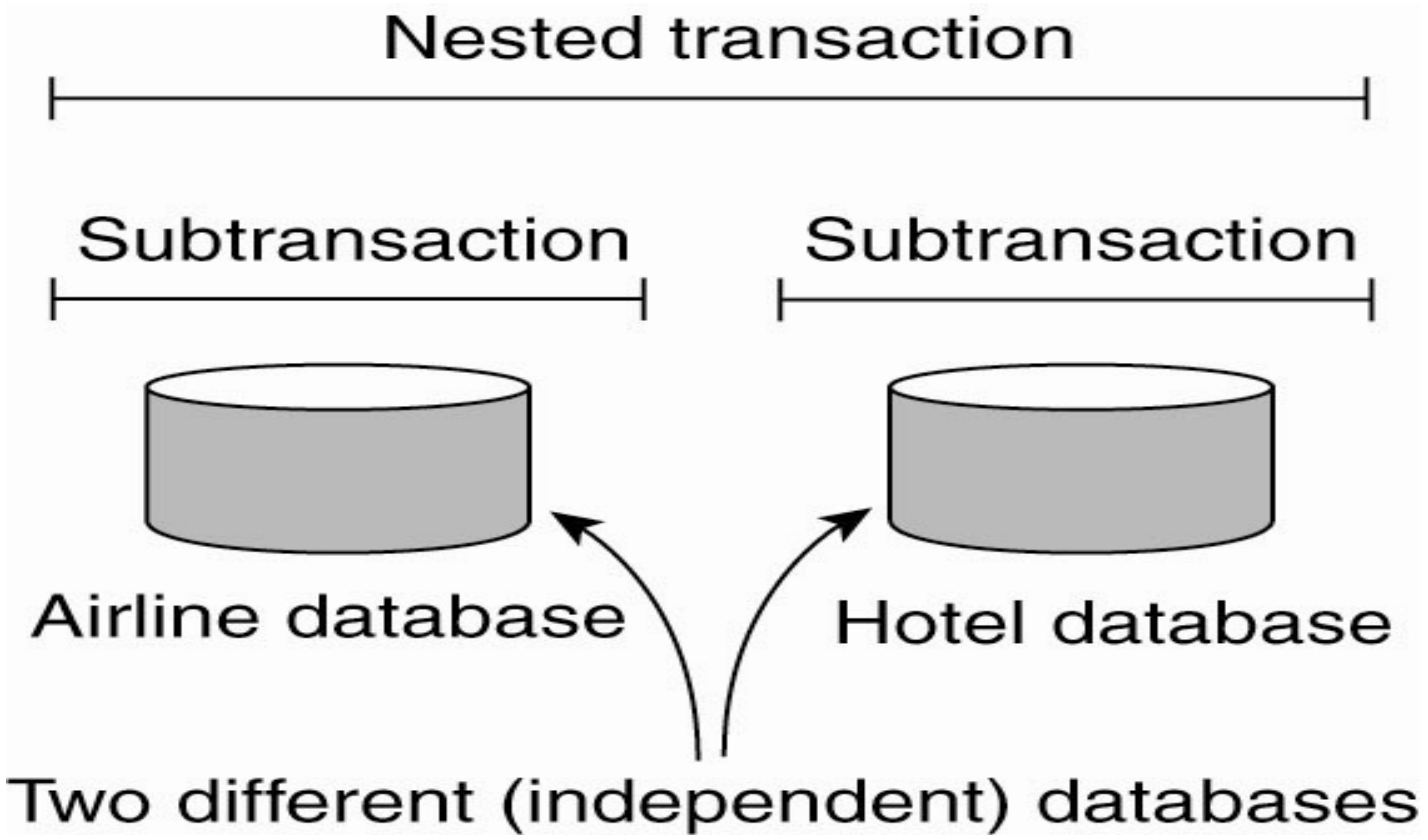


Figure 1-9. A nested transaction.

TP Monitor:

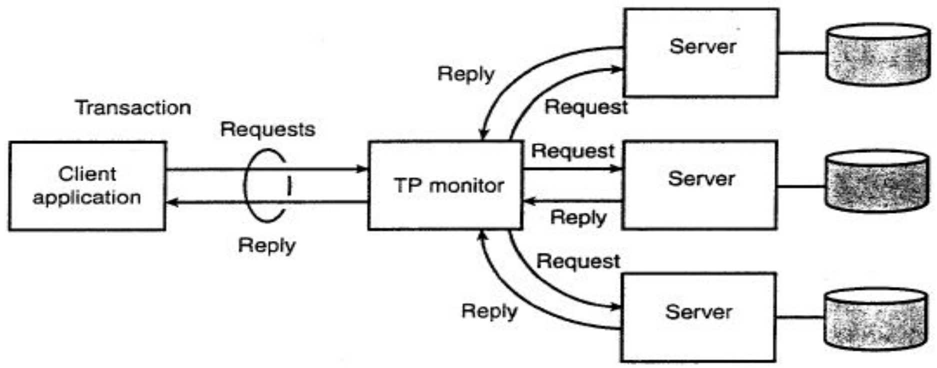


Figure 1-10. The role of a TP monitor in distributed systems.

Enterprise Application Integration

- Less structured than transaction-based systems
- EA components communicate directly
- May use different OSs, different DBs but need to interoperate sometimes.
- Communication mechanisms to support this include, Remote Procedure Call (RPC) and Remote Method Invocation (RMI)

Enterprise Application Integration

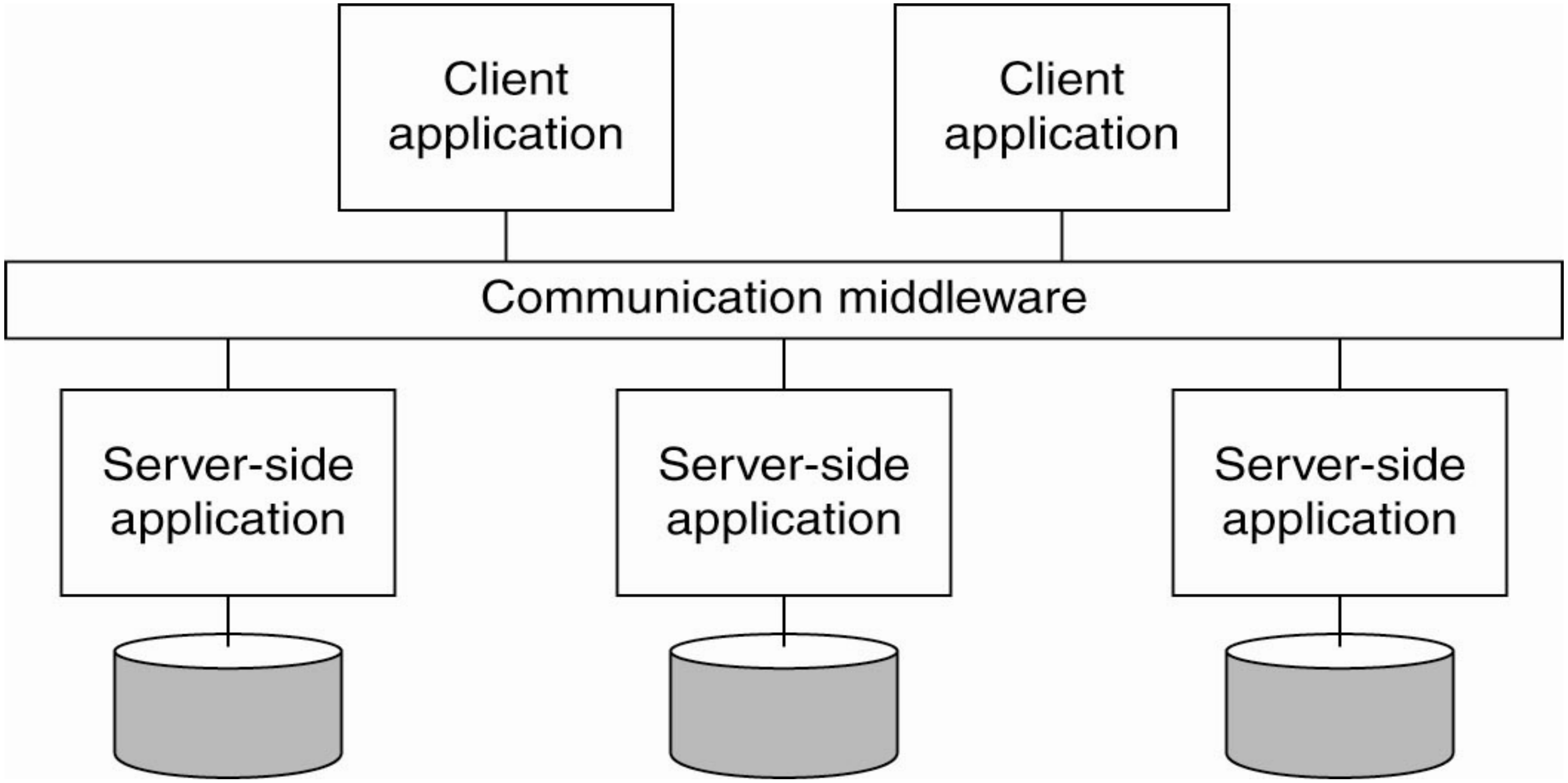


Figure 1-11. Middleware as a communication facilitator in enterprise application integration.

Architectures for Distributed Systems

Definitions

- **Software Architectures** – describe the organization and interaction of software components; focuses on logical organization of software (component interaction, etc.)

- **System Architectures** - describe the placement of software components on physical machines.

Architectural Styles

- An **architectural style** describes a particular way to configure a collection of components and connectors.
  - **Component -** a module with well-defined interfaces; reusable, replaceable.
  - **Connector** – communication link between modules.

- Architectures styles for distributed systems:
  - Layered architectures
  - Object-based architectures
  - Data-centered architectures
  - Event-based architectures

Architectural Styles

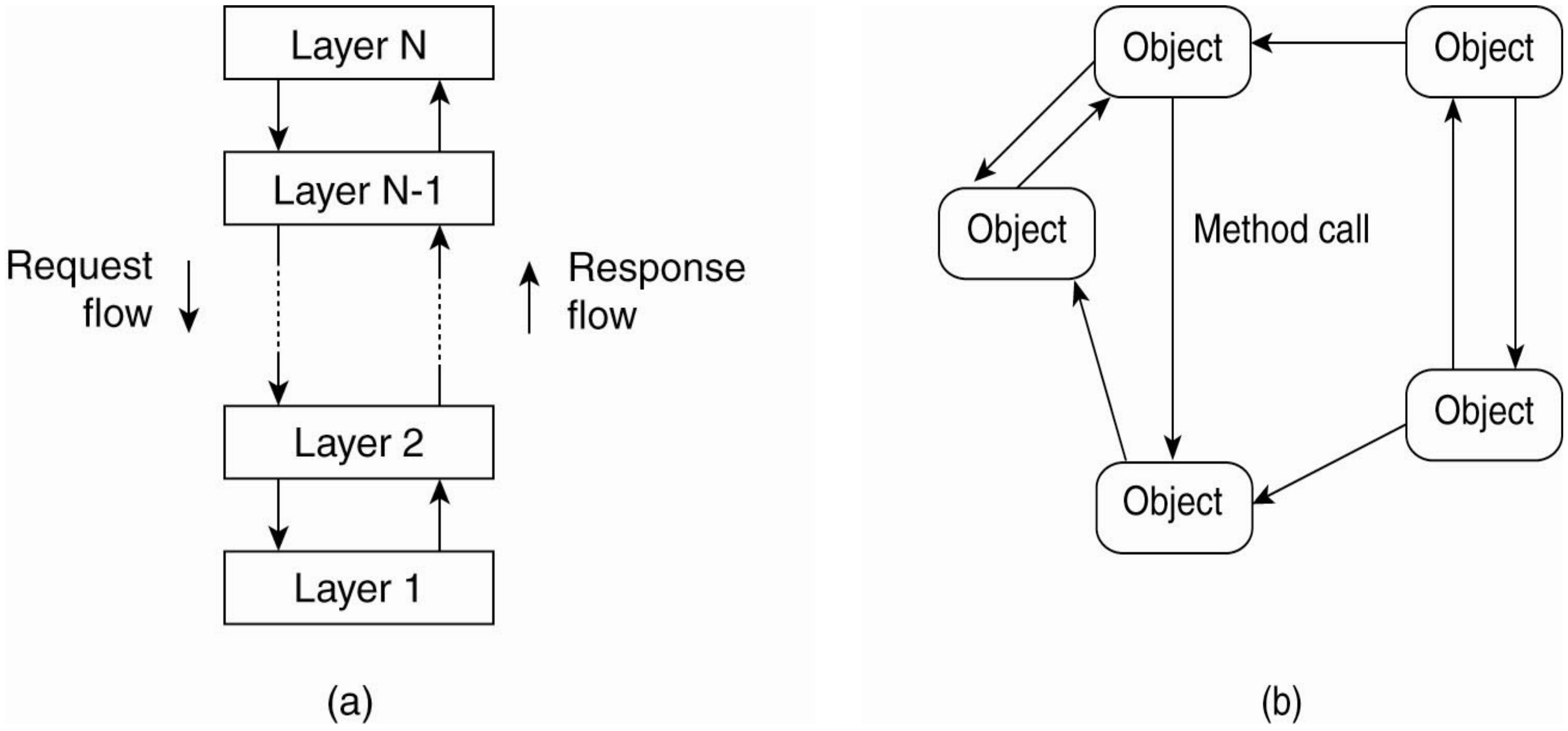
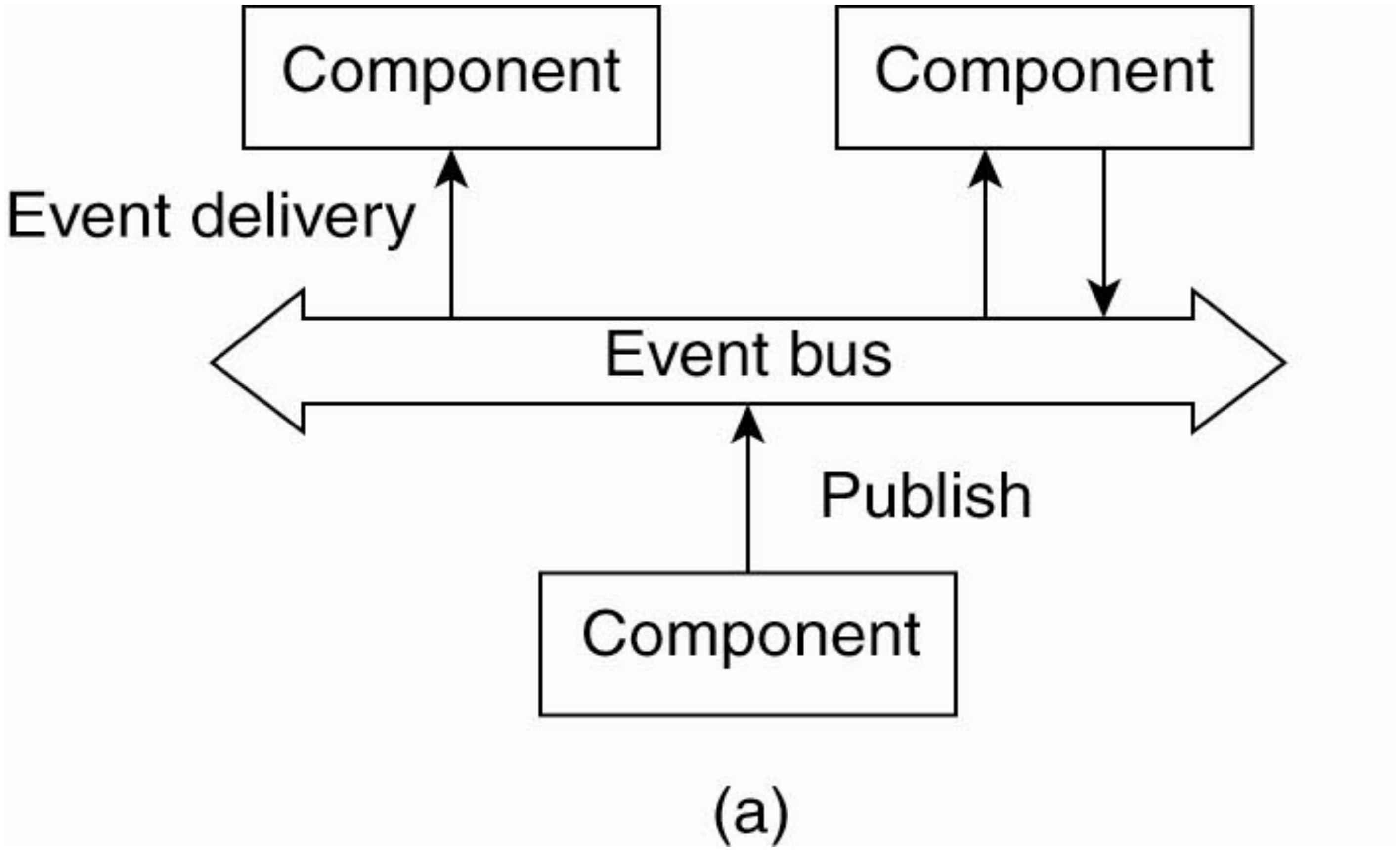


Figure. The (a) layered architectural style & (b) The object-based architectural style.

Data-Centered Architectures

- Main purpose: data access and update
- Processes interact by reading and modifying data in some shared repository (active or passive)
- Another example: web-based distributed systems where communication is through web services

Architectural Styles



- Figure. (a) The event-based architectural style

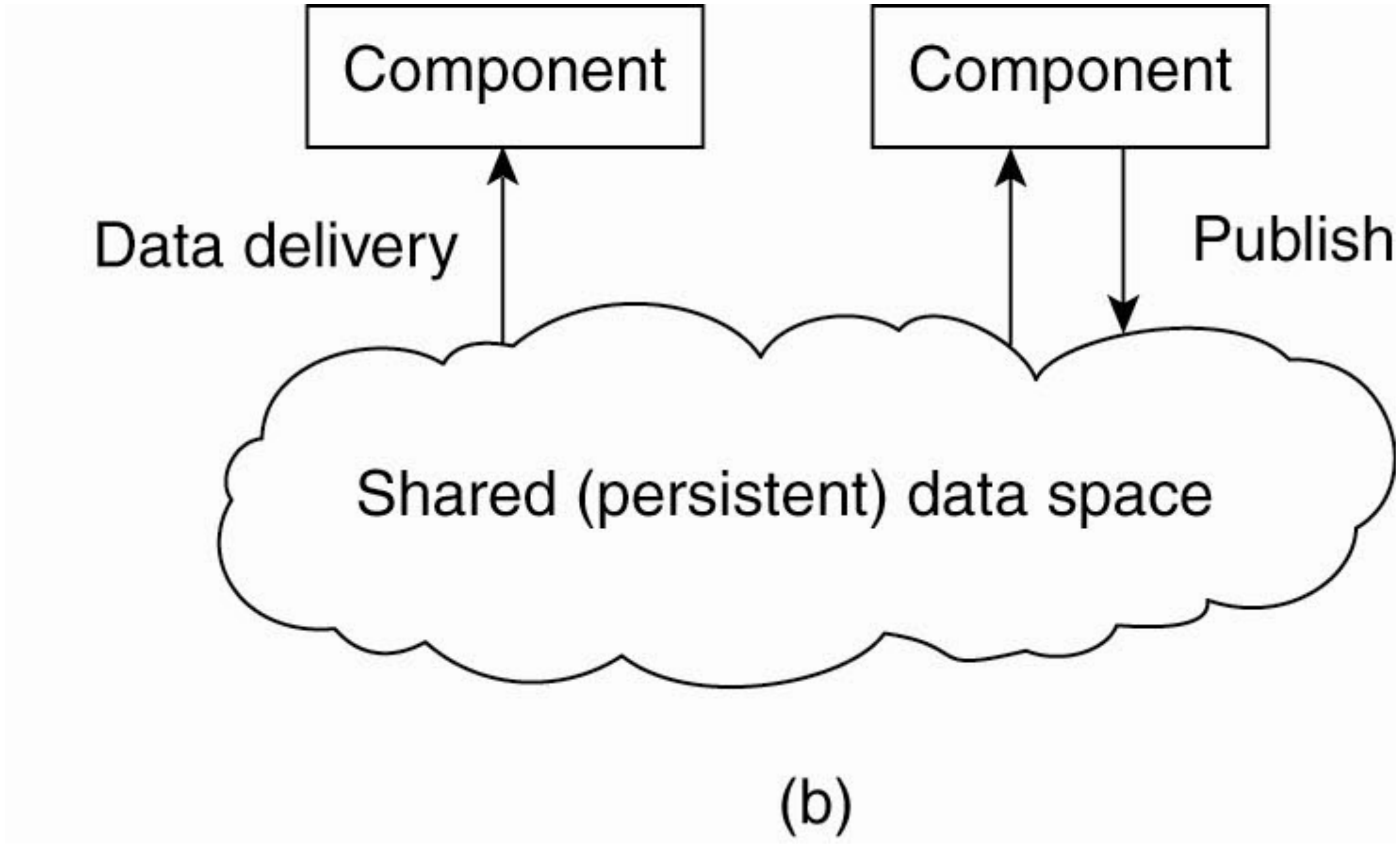


Figure. (b) The shared data-space architectural style.

System Architectures for Distributed Systems

- **Centralized**: traditional client-server structure
  - Vertical (or hierarchical) organization of communication and control paths (as in layered software architectures)
  - Logical separation of functions into client (requesting process) and server (responder)

- **Decentralized**: peer-to-peer
  - Horizontal rather than hierarchical comm. and control
  - Communication paths are less structured; symmetric functionality

- **Hybrid:** combine elements of C/S and P2P
  - Edge-server systems
  - Collaborative distributed systems.

Traditional Client-Server

- Processes are divided into two groups (clients and servers).
- Synchronous communication: request-reply protocol.
- connectionless protocol (unreliable)
- connection-oriented TCP/IP (reliable)

C/S Architectures

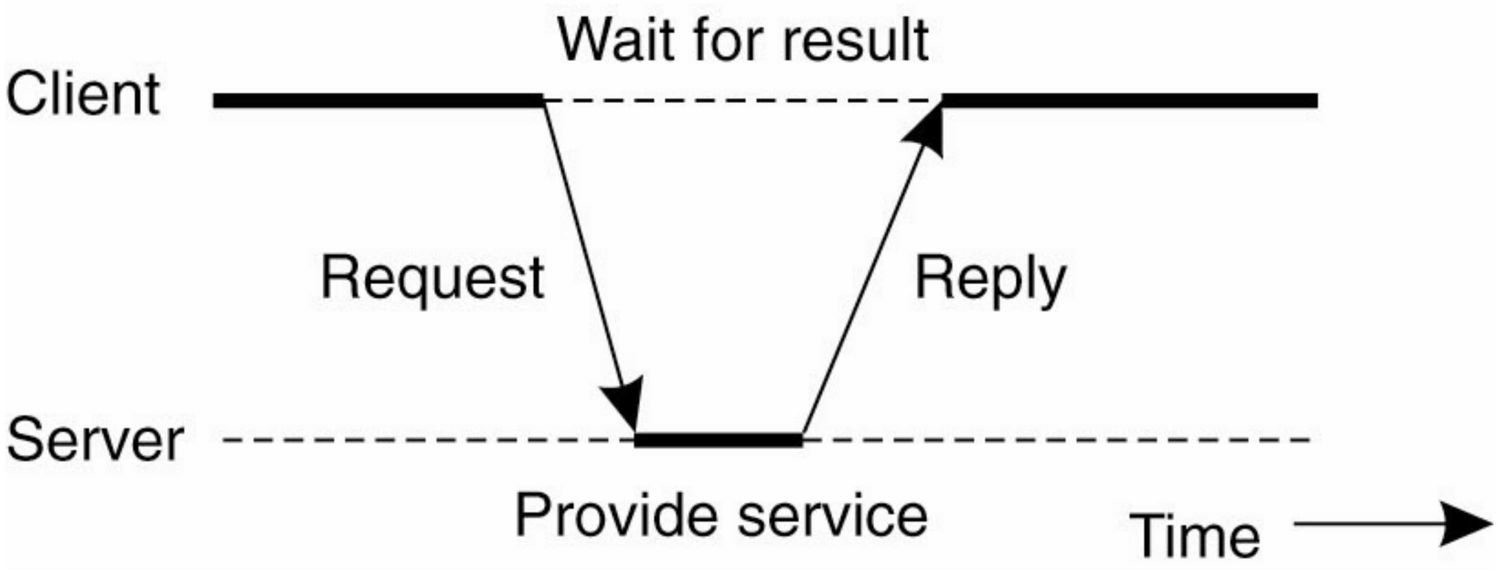


Figure. General interaction between a client and a server.

Transmission Failures

- no reply

- Possibilities:
  - Request message was lost
  - Reply message was lost
  - Server failed either before, during or after performing the service

- Can the client tell which of the above errors took place?

- re-transmission

Layered (software) Architecture for Client-Server Systems

- **User-interface level**: GUI's (usually) for interacting with end users
- **Processing level**: data processing applications – the core functionality
- **Data level**: interacts with data base or file system
  - Data usually is persistent; exists even if no client is accessing it
  - File or database system

Examples

- Web search engine
  - Interface: type in a keyword string
  - Processing level: processes to generate DB queries, rank replies, format response
  - Data level: database of web pages

- Desktop “office suites”
  - Interface: access to various documents, data,
  - Processing: word processing, database queries..
  - Data : file systems and/or databases

Application Layering

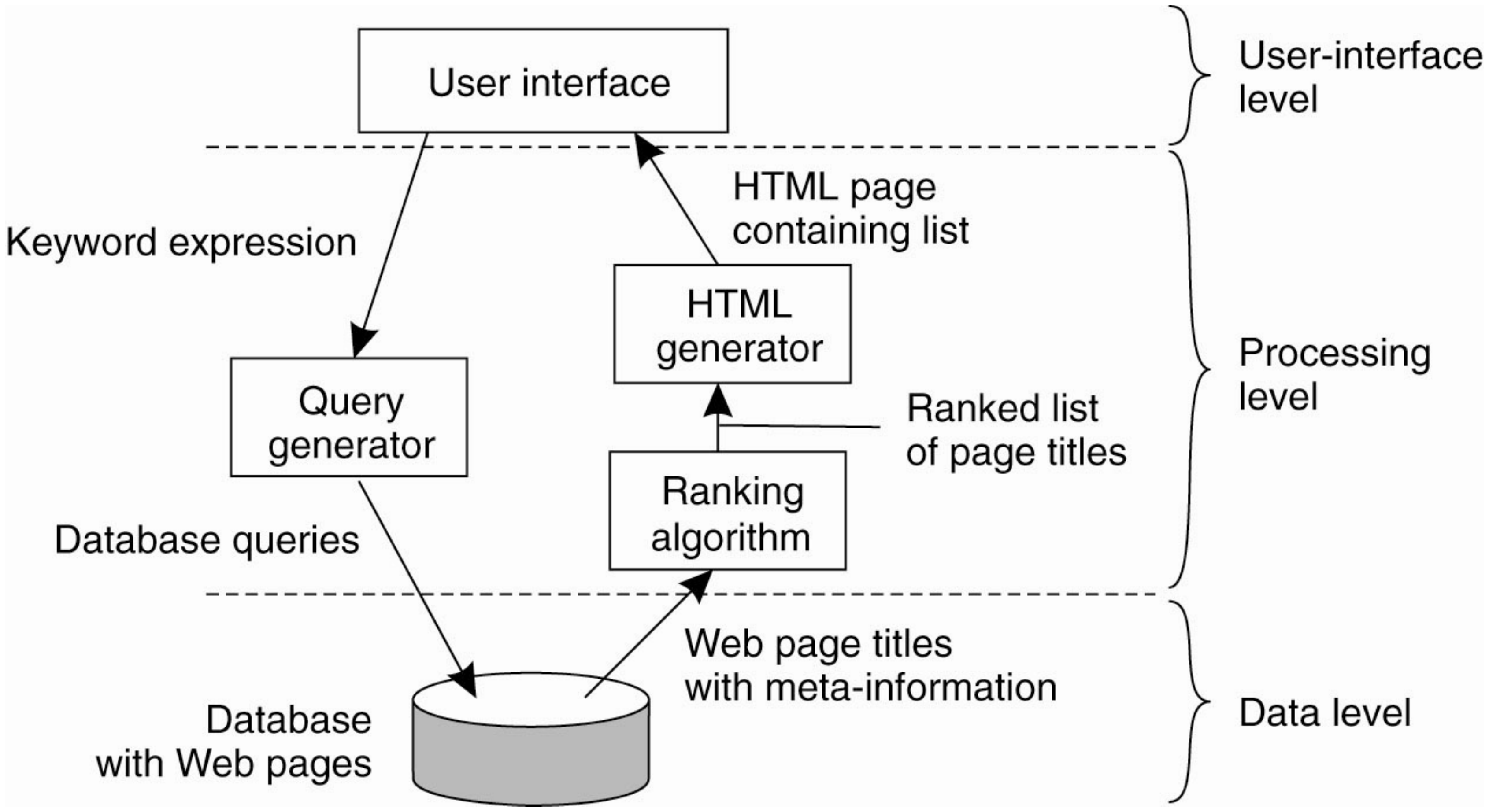


Figure. The simplified organization of an Internet search engine into three different layers.

System Architecture

- Mapping the software architecture to system hardware
  - Correspondence between logical software modules and actual computers

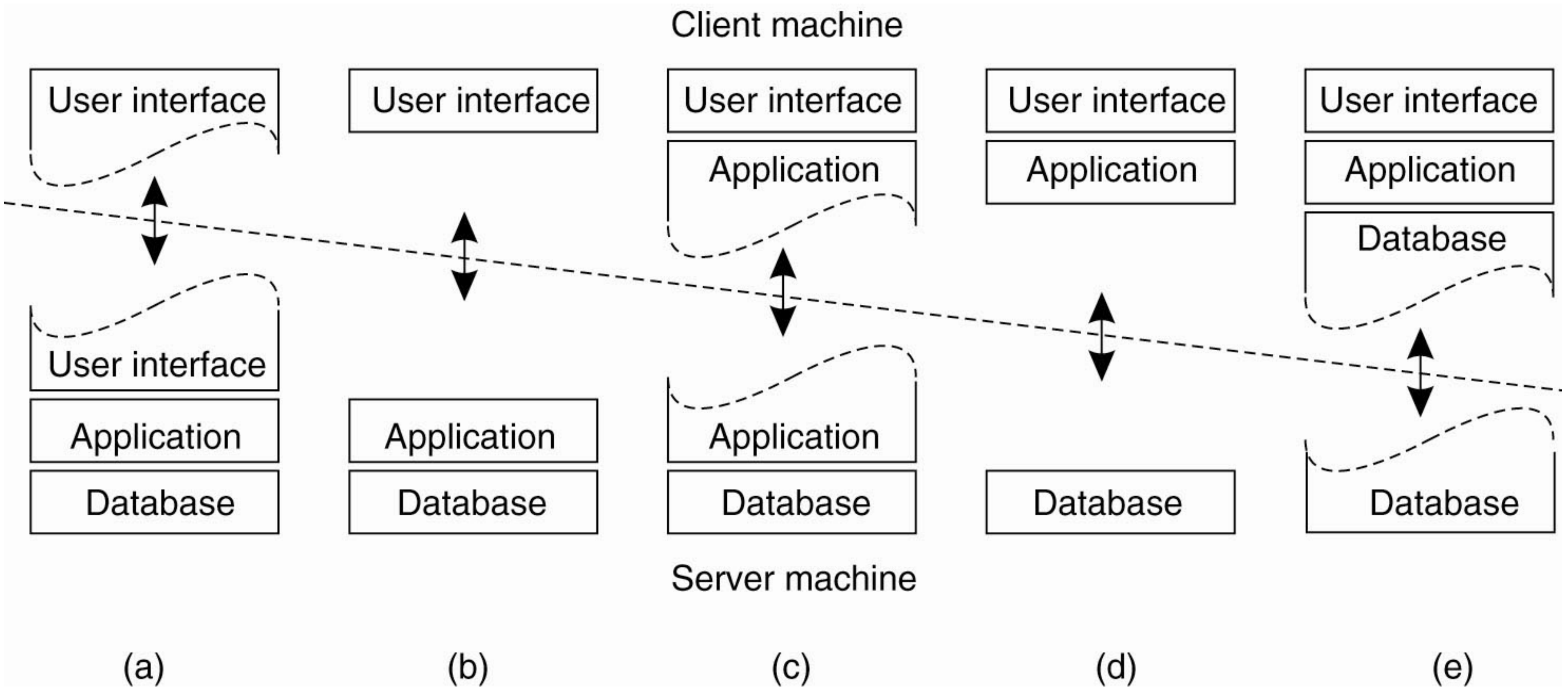
- Multi-tiered architectures
  - *Layer* and *tier* are roughly equivalent terms, but *layer* typically implies software and *tier* is more likely to refer to hardware.
  - Two-tier and three-tier are the most common

Two-tiered C/S Architectures

- Server provides processing and data management; client provides simple graphical display (**thin-client**)
  - Easier to manage, more reliable, client machines don't need to be so large and powerful

- At the other extreme, all application processing and some data resides at the client (**fat-client** approach)
  - Pro: reduces work load at server; more scalable
  - Con: harder to manage by system admin, less secure

Multitiered Architectures



Thin Client ⟷ **Fat Client**

Figure. Alternative client-server organizations (a)–(e).

Three-Tiered Architectures

- In some applications servers may also need to be clients, leading to a three level architecture

  - Distributed transaction processing
  - Web servers that interact with database servers

- Distribute functionality across three levels of machines instead of two.

Multitiered Architectures (3 Tier Architecture)

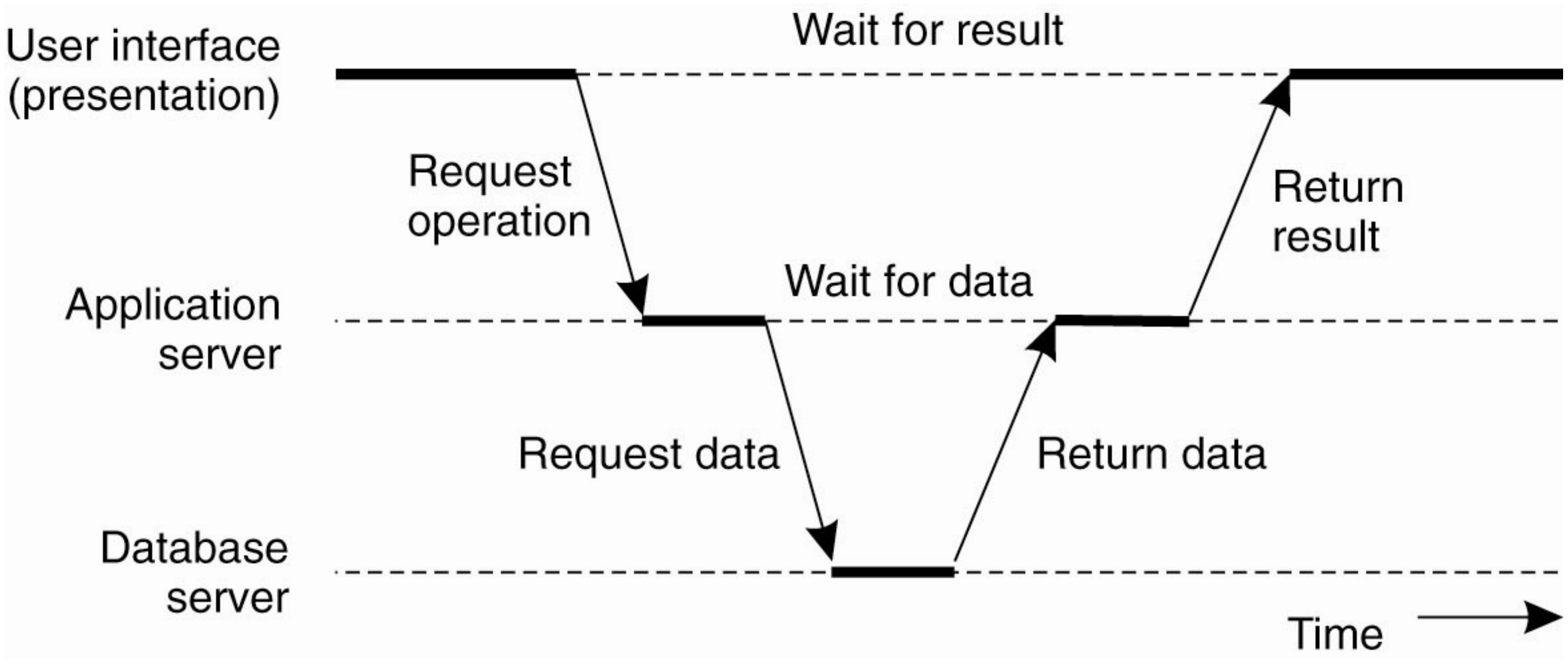


Figure. An example of a server acting as client.

Centralized vs Decentralized Architectures

- Traditional client-server architectures exhibit **vertical distribution**. Each level serves a different purpose in the system.
  - *Logically* different components reside on different nodes

- **Horizontal distribution** (P2P): each node has roughly the same processing capabilities and stores/manages part of the total system data.
  - Better load balancing, more resistant to denial-of-service attacks, harder to manage than C/S

Peer-to-Peer

- Nodes act as both client and server; interaction is symmetric

- Each node acts as a server for part of the total system data

- **Overlay networks** connect nodes in the P2P system
  - Nodes in the overlay use their own addressing system for storing and retrieving data in the system
  - Nodes can route requests to locations that may not be known by the requester.

Overlay Networks

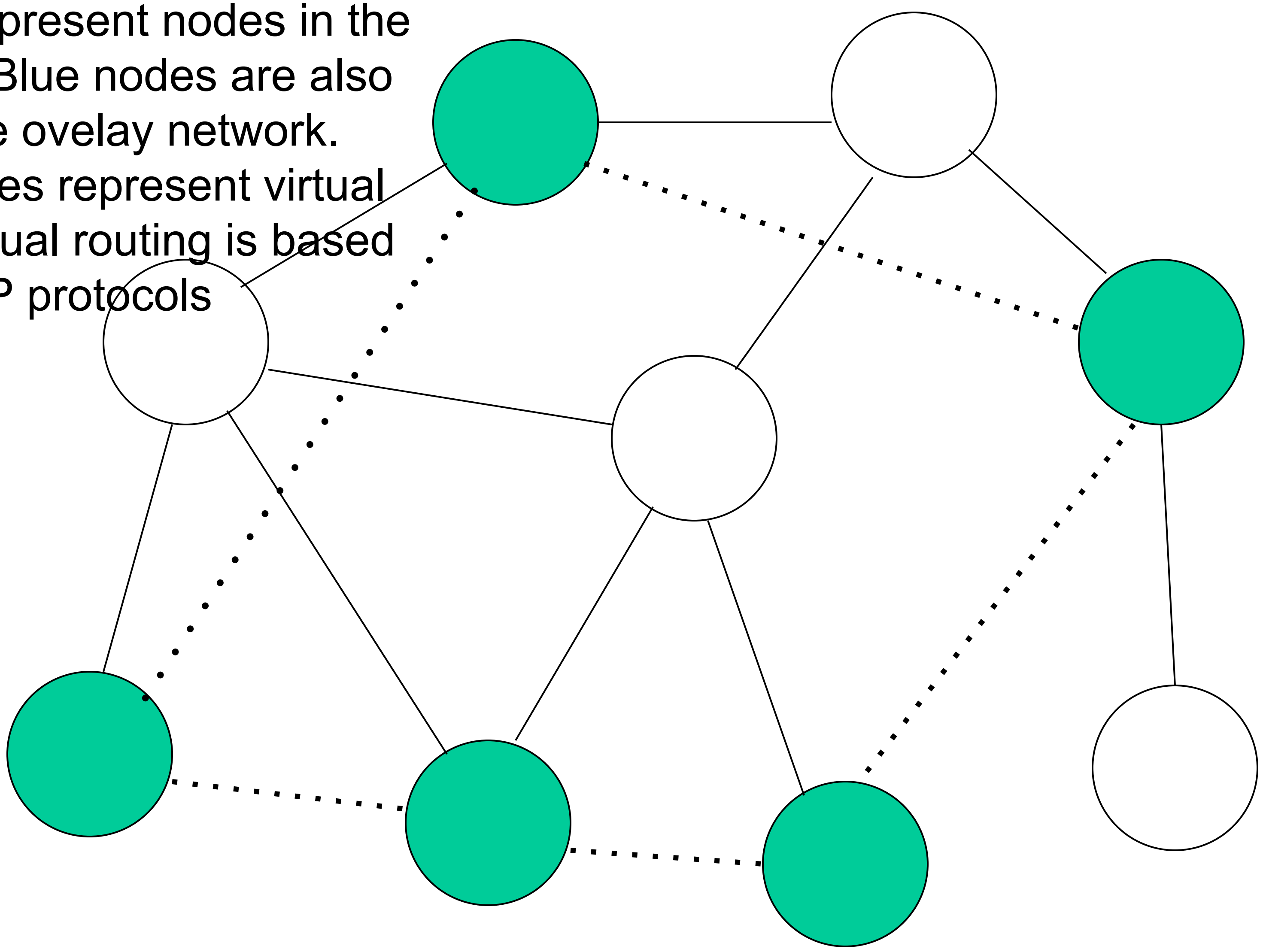
- Are logical or *virtual* networks, built on top of a physical network

- A link between two nodes in the overlay may consist of several physical links.

- Messages in the overlay are sent to logical addresses, not physical (IP) addresses

- Various approaches used to resolve logical addresses to physical.

Circles represent nodes in the network. Blue nodes are also part of the ovelay network. Dotted lines represent virtual links. Actual routing is based on TCP/IP protocols



Overlay Network Example

Overlay Networks

- Each node in a P2P system knows how to contact several other nodes.
- The overlay network may be <u>structured</u> (nodes and content are connected according to some design) or <u>unstructured</u> (content is assigned to nodes without regard to the network topology.)

Structured P2P Architectures

- A common approach is to use a **distributed hash table** (DHT) to organize the nodes.
- In a DHT, data objects and nodes are each assigned a key which hashes to a random number from a very large identifier space (to ensure uniqueness)
- A mapping function assigns objects to nodes, based on the hash function value.
-

Characteristics of DHT

- Scalable – to thousands, even millions of network nodes
  - Search time increases more slowly than size; usually $O(\log(N))$
- Fault tolerant – able to re-organize itself when nodes fail
- Decentralized – no central coordinator

Inserting Items in the DHT

- A data item with key value k is mapped to the node with the smallest identifier id such that id $\geq$ k
- This node is the successor of k, or succ(k)

Structured Peer-to-Peer Architectures

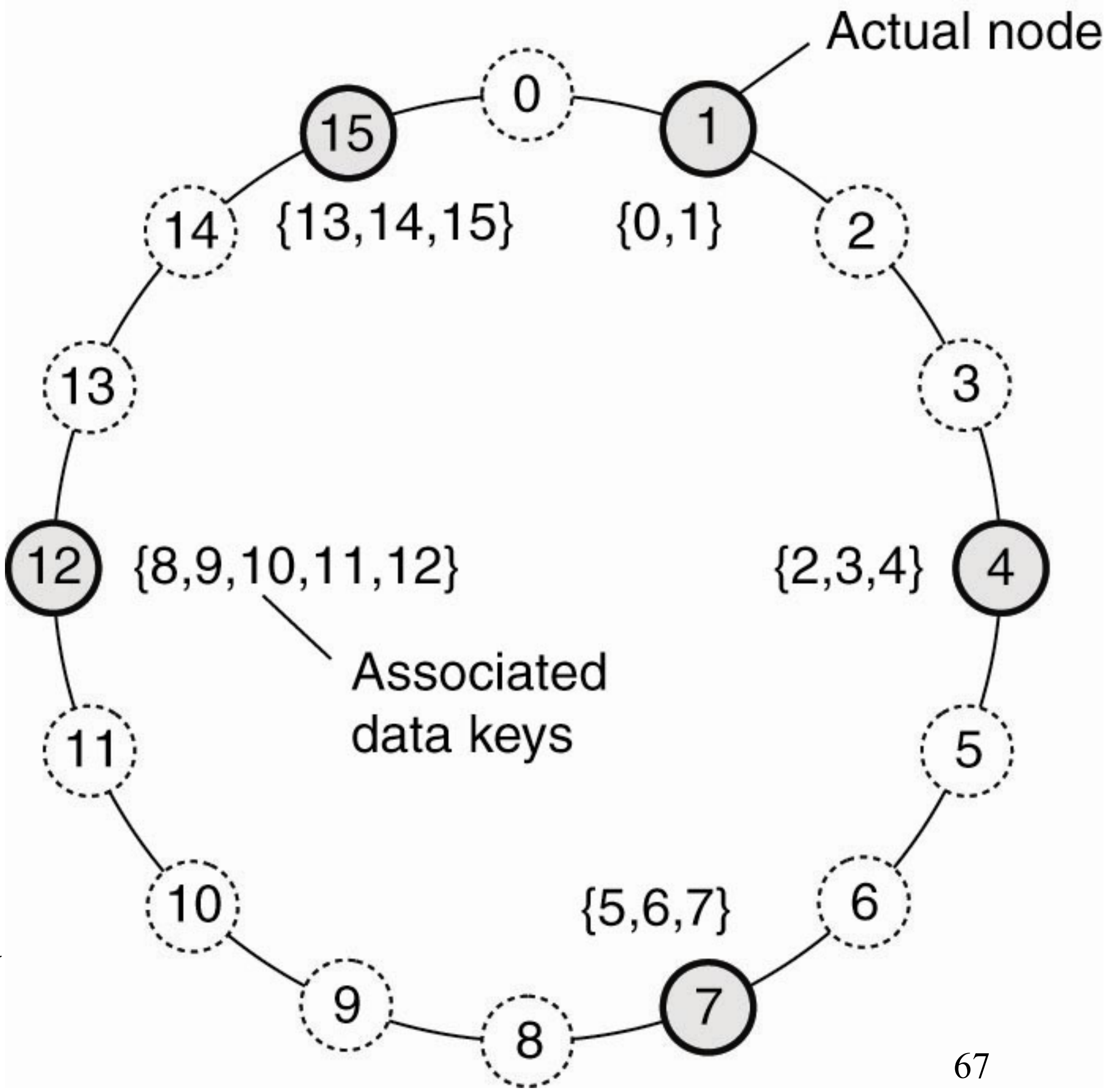


Figure. The mapping of data items onto nodes in Chord for m = 4

Hybrid Architectures

- Combine client-server and P2P architectures
  - Edge-server systems; e.g. ISPs, which act as servers to their clients, but cooperate with other edge servers to host shared content
  - Collaborative distributed systems; e.g., BitTorrent, which supports parallel downloading and uploading of chunks of a file. First, interact with C/S system, then operate in decentralized manner.

Edge-Server Systems

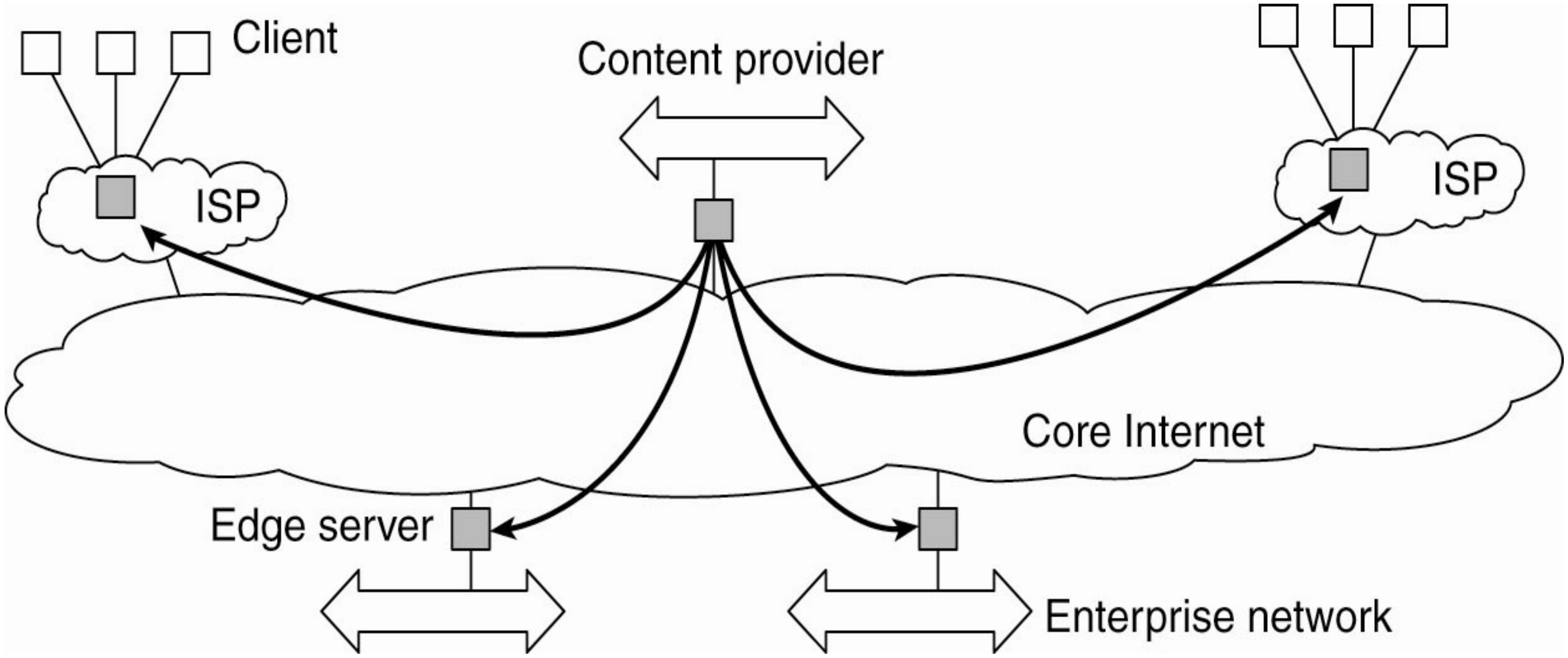


Figure. Viewing the Internet as consisting of a collection of edge servers.

Collaborative Distributed Systems BitTorrent

- Clients contact a global directory (Web server) to locate a *.torrent* file with the information needed to locate a **tracker**; a server that can supply a list of active nodes that have chunks of the desired file.

- Using information from the tracker, clients can download the file in chunks from multiple sites in the network. Clients must also provide file chunks to other users.

Collaborative Distributed Systems

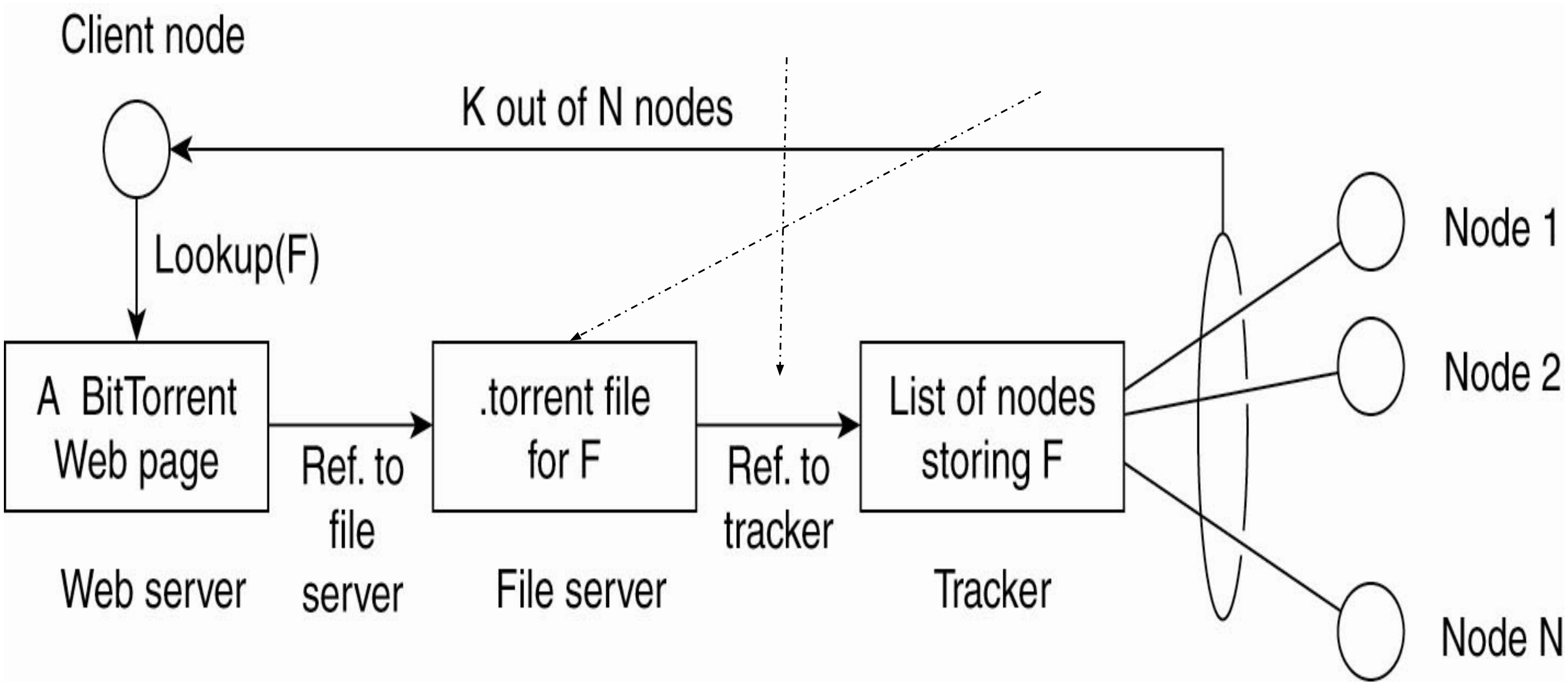


Figure 2-14. The principal working of BitTorrent.

BitTorrent - Justification

- Designed to force users of file-sharing systems to participate in sharing.
  - When a user downloads your file, he becomes in turn a server who can upload the file to other requesters.
  - Share the load .