Distributed Systems

Chapter 1

Agegnehu A.

Outline

- Definition of a Distributed System
- Goals of a Distributed System
  - (transparency, openness, availability, scalability)
- Types of Distributed Systems
  - (computing, information, embedded)

What Is A Distributed System?

- A collection of independent computers that appears to its users as a single coherent system.
- Features:
  - No shared memory – message-based communication
  - Each runs its own local OS
  - Heterogeneity
- 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
  - Provide uniform interface
- 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

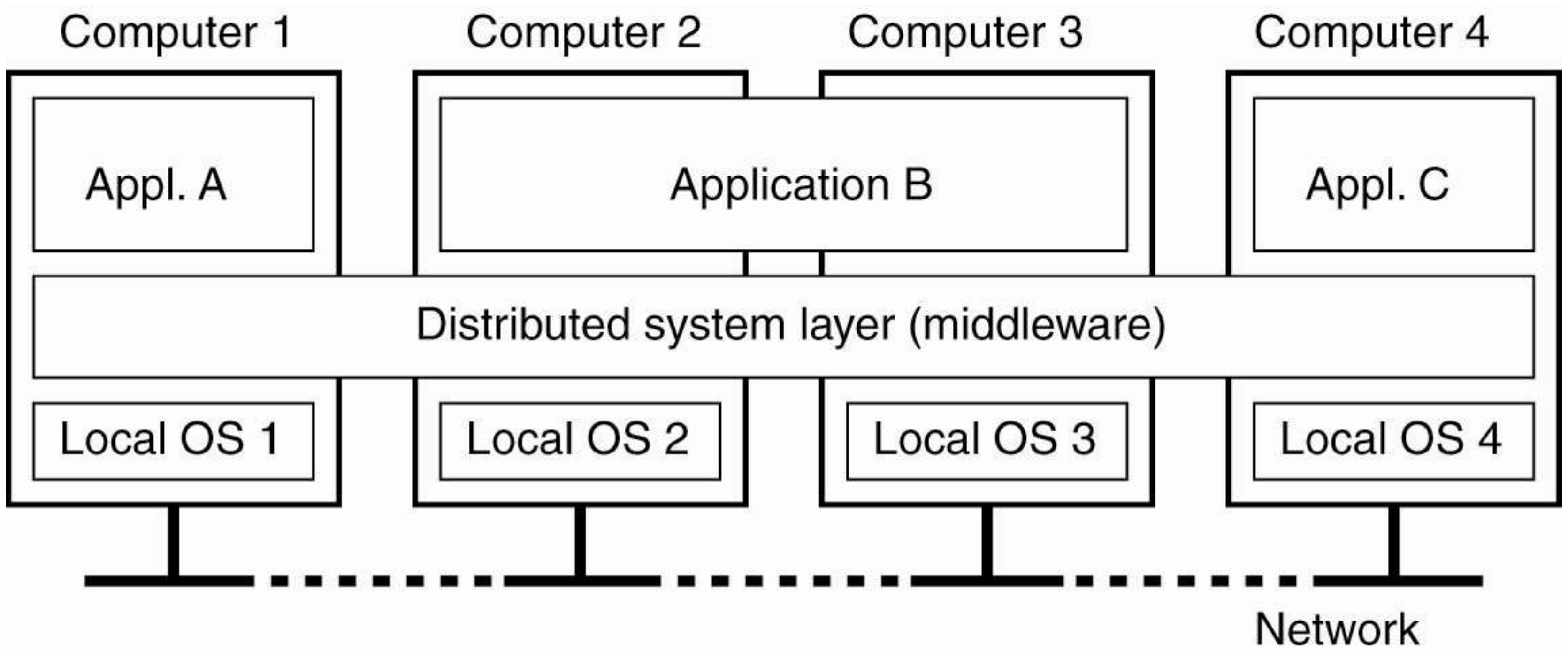


Figure 1-1. A distributed system organized as middleware. The middleware layer runs on all machines, and offers a uniform interface to the system

Role of Middleware (MW)

- In some early research systems: MW tried to provide the illusion that a collection of separate machines was a single computer.
- Today:
  - clustering software allows independent computers to work together closely
  - MW also supports seamless access to remote services, doesn’t try to look like a general-purpose OS

Middleware Examples

- CORBA (Common Object Request Broker Architecture)
- DCOM (Distributed Component Object Management) – being replaced by .net
- Sun's RPC (Remote Procedure Call)
- RMI (Remote Method Invocation)
- SOAP (Simple Object Access Protocol)

Middleware Examples…

- All of the previous examples support communication across a network:
- They provide protocols that allow a program running on one kind of computer, using one kind of operating system, to call a program running on another computer with a different operating system
  - The communicating programs must be running the *same* middleware.

Distributed System Goals

- Resource Accessibility
- Distribution Transparency
- Openness
- Scalability

Goal 1 – Resource Availability

- 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
- Performance enhancement – due to multiple processors; also due to ease of collaboration and info - exchange – access to remote services
  - Groupware: tools to support collaboration
- 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 during use

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

Goal 2: Degrees of Transparency

- Trade-off/balance: transparency versus other factors
  - Reduced performance: multiple attempts to contact a remote server can slow down the system – should you report failure and let user cancel request?
  - Convenience: direct the print request to my local printer, not one on the next floor
- Too much emphasis on transparency may prevent the user from understanding system behavior.

Goal 3 - Openness

- An **open distributed system** : the interfaces to the system are clearly specified and freely available.
  - Compare to network protocols
  - *Not* proprietary
- **Interface Definition/Description Languages (IDL):** used to describe the interfaces between software components, usually in a distributed system
  - Definitions are language & machine independent
  - 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.

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.
  - Multiple implementations of the same service may be provided, as long as the interface is maintained
- **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
- A scalable system still performs well as it scales up along any of the three dimensions.

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
  - Centralized algorithms: one site collects all information, processes it, distributes the results to all sites.
    - Complete knowledge: good
    - Time and network traffic: bad

Decentralized Algorithms

- No machine has complete information about the system state
- Machines make decisions based only on local information
- Failure of a single machine doesn't distract the algorithm
- There is no assumption that a global clock exists.

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
  - Filling in forms to access a DB: send a separate message for each field, or download form/code and submit finished version.
  - i.e., shorten the wait times

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

Scaling Techniques (2)

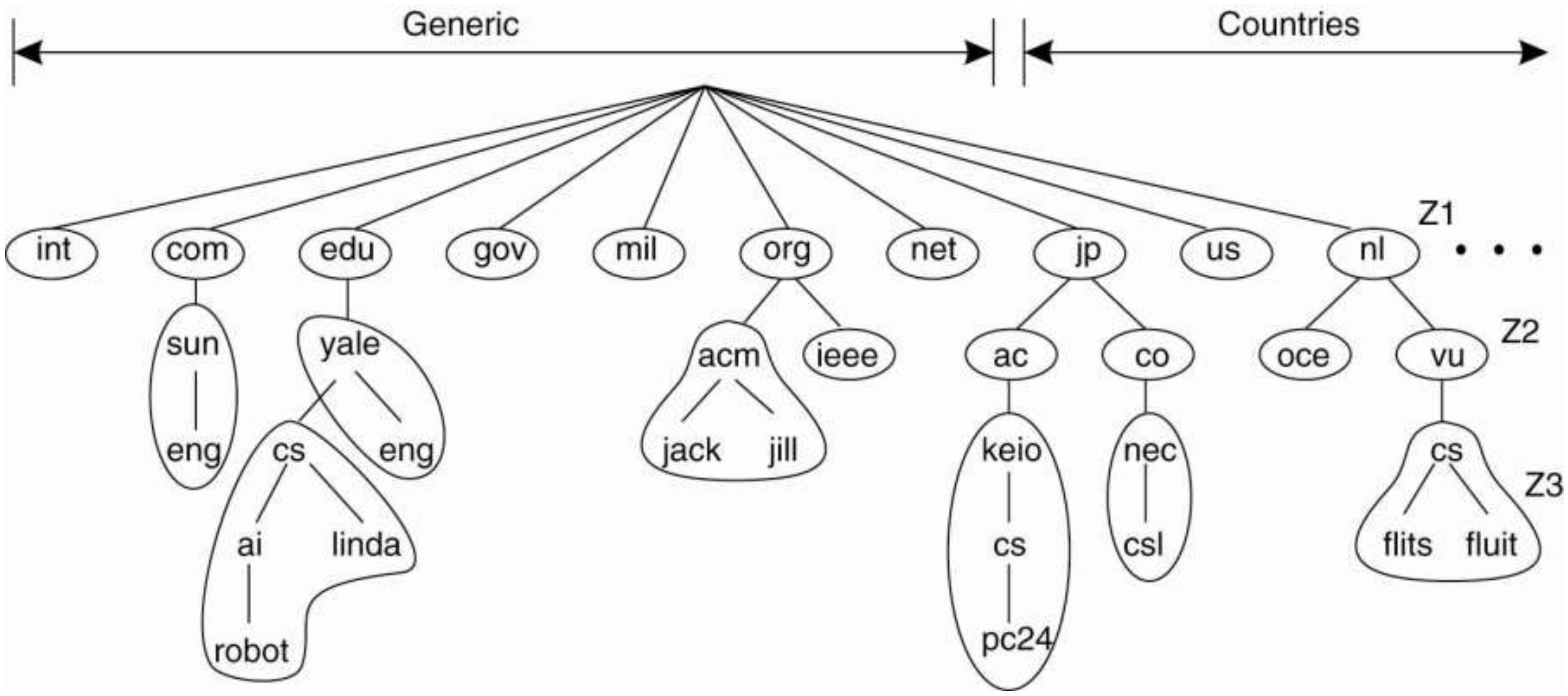


Figure 1-5. An example of dividing the DNS name space into zones.

Third Scaling Technique - 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 by improving proximity of resource

Caching

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

Summary Goals for Distribution systems

- Resource accessibility
  - For sharing and enhanced performance
- Distribution transparency
  - For easier use
- Openness
  - To support interoperability, portability, extensibility
- Scalability
  - With respect to size (number of users), geographic distribution, administrative domains

Issues/Pitfalls of Distribution

- Requirement for advanced software to realize the potential benefits.
- Security and privacy concerns regarding network communication.
- Replication of data and services provides fault tolerance and availability, but at a cost.
- Network reliability, security, heterogeneity, topology
- Latency and bandwidth
- Administrative domains

Types of Distributed Systems

- Distributed Computing Systems
  - Clusters
  - Grids
  - Clouds
- Distributed Information Systems
  - Transaction Processing Systems
  - Enterprise Application Integration
- Distributed Embedded Systems
  - Home systems
  - Health care systems
  - Sensor networks

Cluster Computing

- A collection of similar processors (PCs, workstations) running the same operating system, connected by a high-speed LAN.
- Parallel computing capabilities using inexpensive PC hardware
- Replace big parallel computers (MPPs)

Cluster Types & Uses

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

Clusters – Beowulf model

- Linux-based
- Master-slave paradigm
  - One processor is the master; allocates tasks to other processors, maintains batch queue of submitted jobs, handles interface to users
  - Master has libraries to handle message-based communication or other features (the middleware).

Cluster Computing Systems

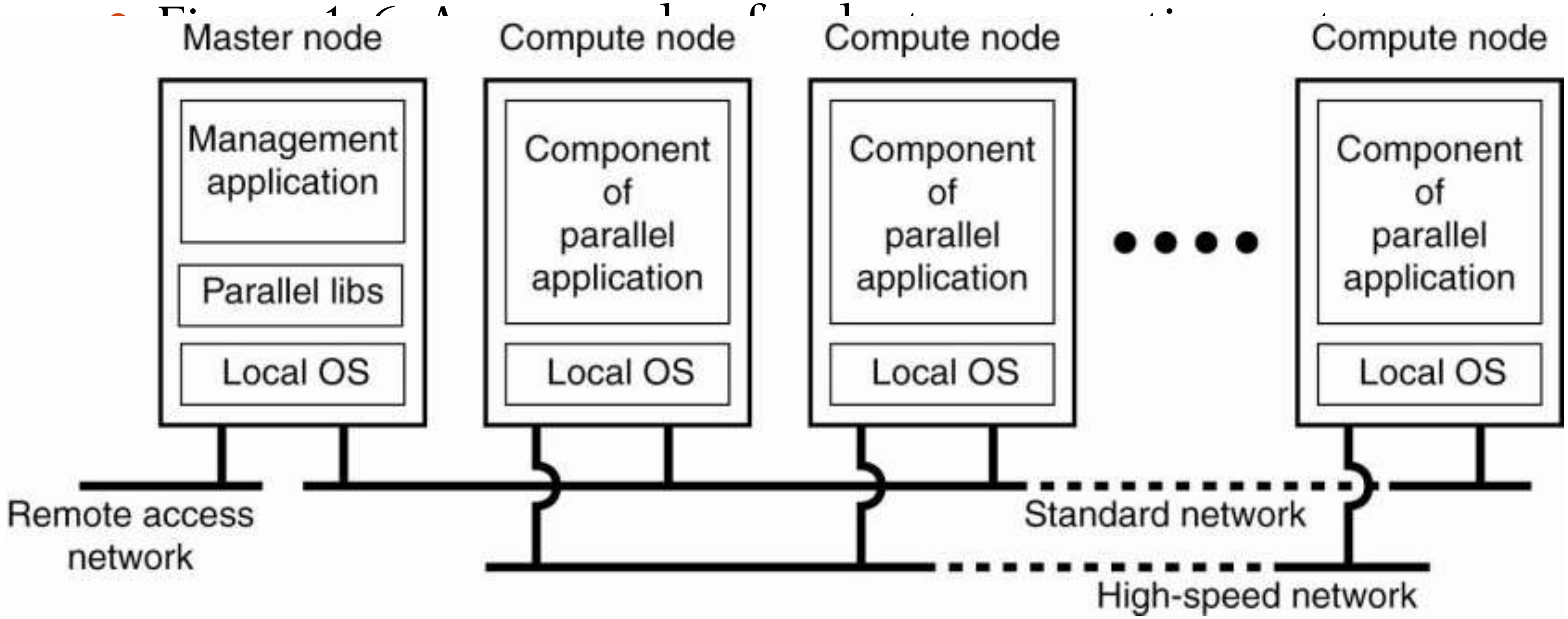


Figure 1-6. An example of a (Beowolf) cluster computing system

Clusters – MOSIX model

- Provides a symmetric, rather than hierarchical paradigm
  - High degree of distribution transparency (single system image)
  - Processes can migrate between nodes dynamically and preemptively (more about this later.) Migration is automatic
- Used to manage Linux clusters
- Provides resource discovery and workload distribution among clusters
- Clusters can be partitioned for use by an individual or a group
- Best for compute-intensive jobs

Grid Computing Systems

- Modeled loosely on the electrical grid.
- Highly heterogeneous with respect to hardware, software, networks, security policies, etc.
- Grids support **virtual organizations**: a collaboration of users who pool resources (servers, storage, databases) and share them
- Grid software is concerned with managing sharing across administrative domains.

Grids Computing

- Similar to clusters but processors are more loosely coupled, tend to be heterogeneous, and are not all in a central location.
- Can handle workloads similar to those on supercomputers, but grid computers connect over a network (Internet?) and supercomputers' CPUs connect to a high-speed internal bus/network
- Problems are broken up into parts and distributed across multiple computers in the grid – less communication betw parts than in clusters.

A Proposed 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; e.g., access a remote resource or transfer data between resources; and protocols to provide security
- **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
- The collective, connectivity and resource layers together form the middleware layer for a grid

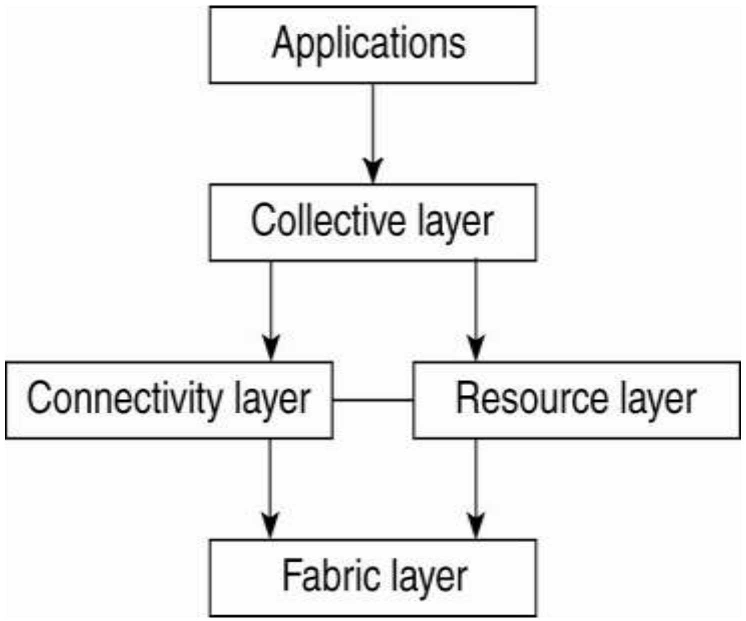


Figure 1-7. A layered architecture for grid computing systems

Cloud Computing

- Provides scalable services as a utility over the Internet.
- Often built on a computer grid
- Users buy services from the cloud
  - Grid users may develop and run their own software
- Cluster/grid/cloud distinctions distortion at the edges!

Types of Distributed Systems

1. Distributed Computing Systems
   - Clusters
   - Grids
   - Clouds
2. Distributed Information Systems
   - Transaction Processing Systems
   - Enterprise Application Integration
3. Distributed Embedded Systems
   - Home systems
   - Health care systems
   - Sensor networks

Lesson two

2.Distributed Information Systems

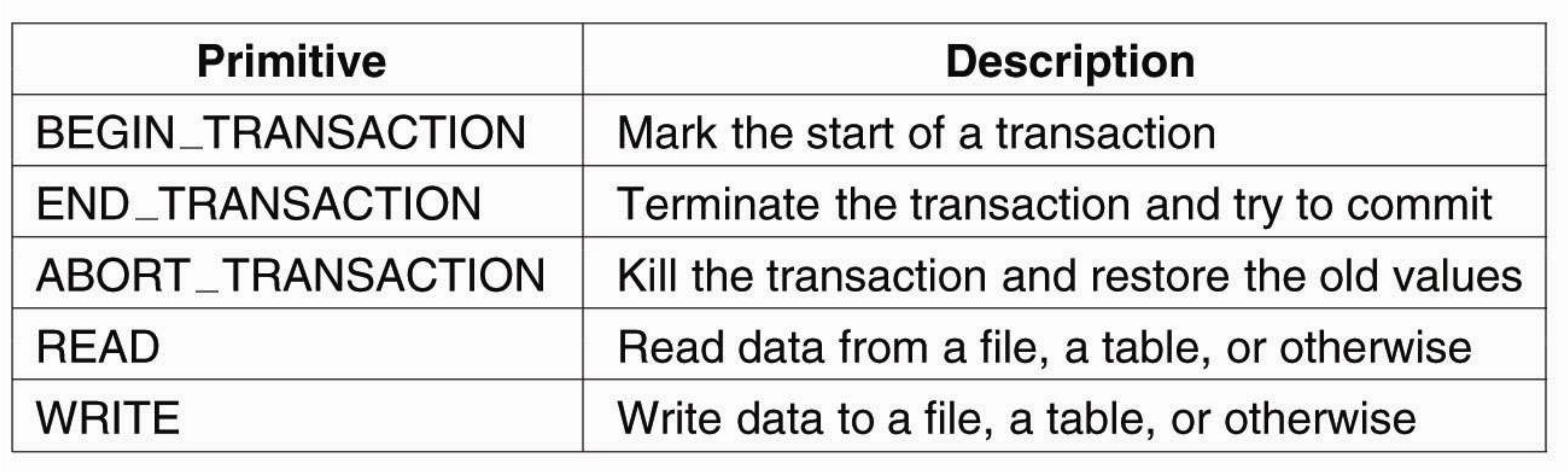
- Business-oriented
- Systems to make a number of separate network applications interoperable and build "enterprise-wide information systems".
- Two types :
  - Transaction processing systems
  - Enterprise application integration (EAI)

2.1.Transaction Processing Systems

- Provide a highly structured client-server approach for database applications
- Transactions are the communication model
- Its has the following properties:
  - **Atomic:** all or nothing
  - **Consistent:** invariants are preserved
  - **Durable:** committed operations can’t be uncompleted

Transaction Processing Systems

- Figure 1-8. Example primitives for transactions.



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

- Transaction processing may be centralized (traditional client/server system) or distributed.
- A distributed database is one in which the *data storage* is distributed – connected to separate processors.

2.2. Enterprise Application Integration

- Less structured than transaction-based systems
- EA components communicate directly
  - Enterprise applications are things like HR(human resource) data, inventory programs, …
  - May use different OSs, different DBs but need to interoperate sometimes.
- Communication mechanisms to support include CORBA, Remote Procedure Call (RPC) and Remote Method Invocation (RMI)

Enterprise Application Integration

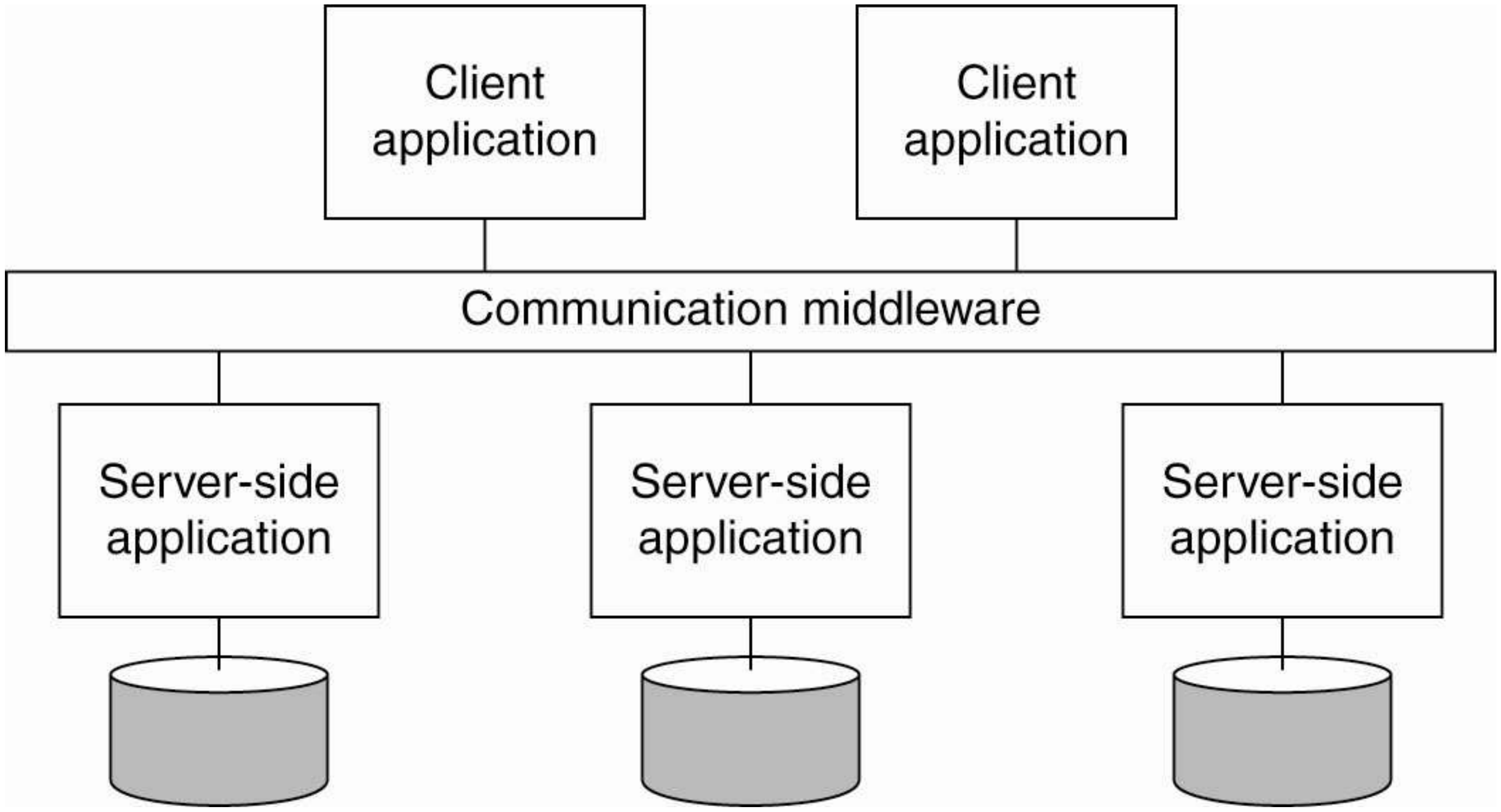


Figure 1-11. Middleware as a communication facilitator in enterprise application integration.

3. Distributed Embedded Systems

- This type of system is likely to incorporate small, battery-powered, mobile devices
  - Home systems
  - Electronic health care systems – patient monitoring
  - Sensor networks – data collection, surveillance

Home System

- Built around one or more PCs, but can also include other electronic devices:
  - Automatic control of lighting, alarm systems, etc.
  - Network enabled appliances
  - PDAs and smart phones, etc.

Electronic Health Care Systems

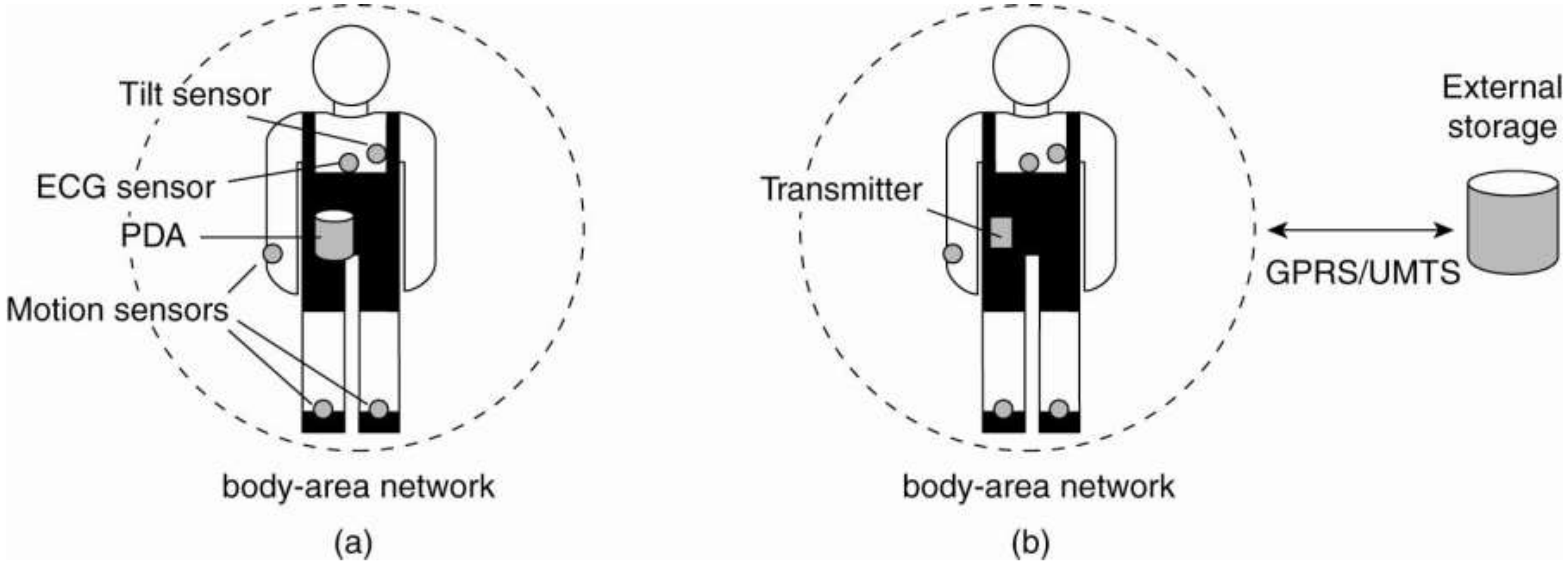


Figure 1-12. Monitoring a person in a pervasive electronic health care system

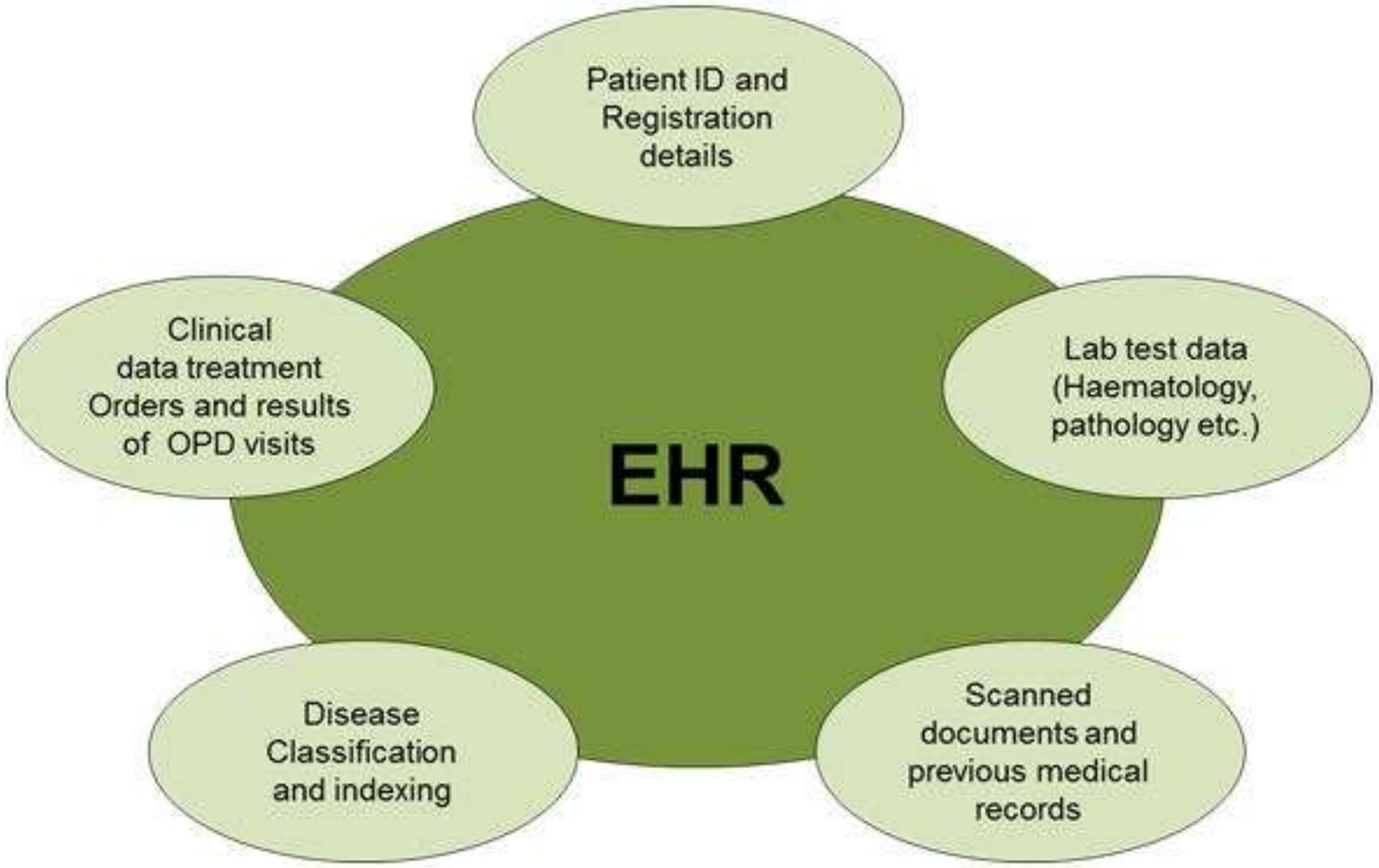
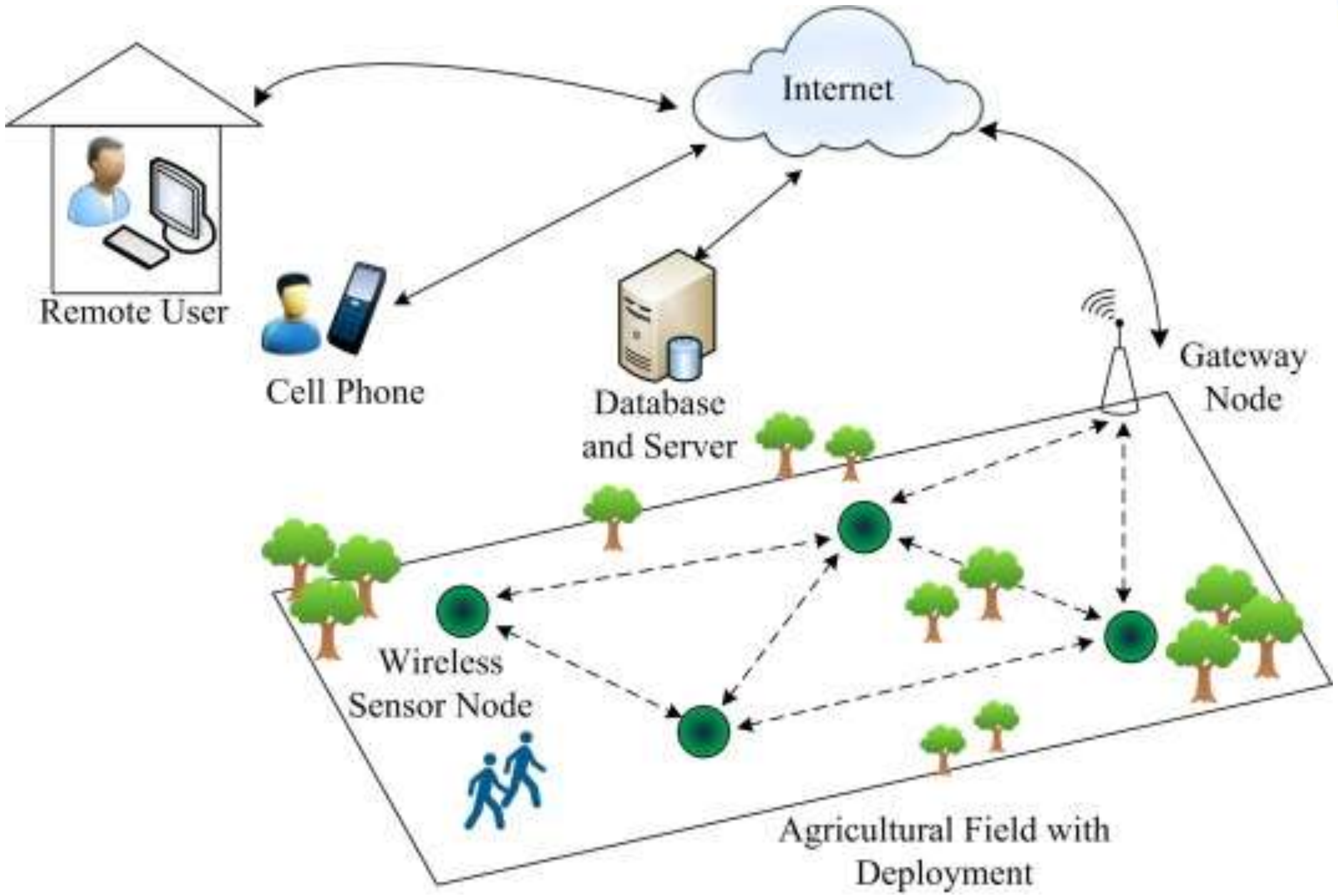

Patient ID and Registration details
Lab test data (Haematology, pathology etc.)
Scanned documents and previous medical records
Disease Classification and indexing
Clinical data treatment Orders and results of OPD visits
EHR

Figure 1; *Source: Electronic Health Records: Manual for Developing Countries by WHO.*

Sensor Networks

- A collection of geographically distributed nodes consisting of a comm. device, a power source, some kind of sensor, a small processor…
- **Purpose:** to collectively monitor sensory data (temperature, sound, moisture etc.,) and transmit the data to a base station
- "**smart environment**" – the nodes may do some basic processing of the data in addition to their communication responsibilities.


Internet
Remote User
Cell Phone
Database
and Server
Gateway
Node
Wireless
Sensor Node
Agricultural Field with
Deployment

Questions?

Distributed Systems

Chapter 2: Distributed Systems Architectures

Agegnehu A. (Weekend)

Inline

- Definition
- Architectural Styles
- System Architecture in distributed system
    - Central, Decentralize and Hybrid architecture
- System architecture (2 tier, 2tier)
- Over relay network
    - Peer to peer architecture, Client ser ver

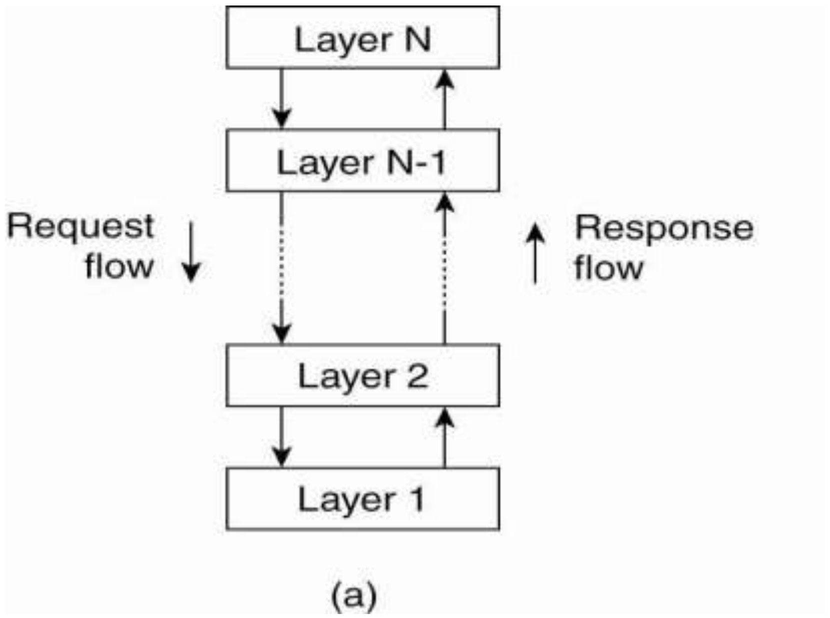
Definitions

- **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
  - The realization of an architecture may be
    - Centralized (most components located on a single machine),
    - Decentralized (most machines have approximately the same functionality),
    - Hybrid (some combination).

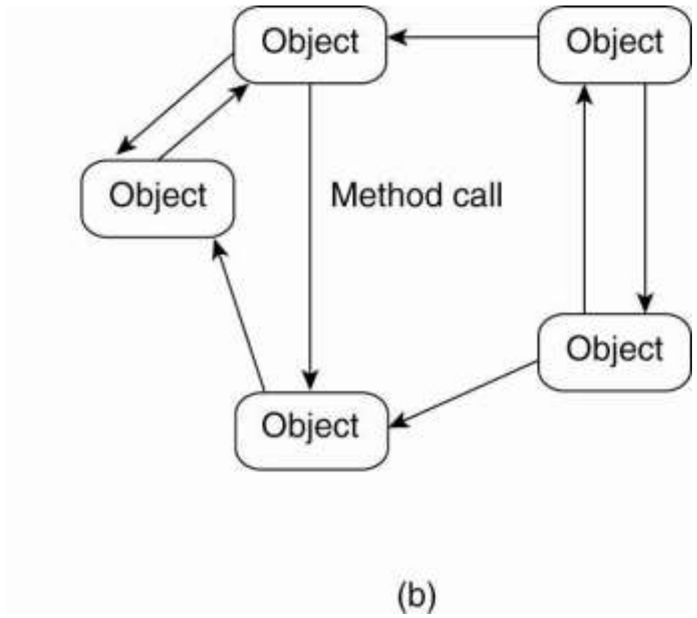
Architectural Styles

- 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 suitable for distributed systems:
  - Layered architectures*
  - Object-based architectures*
  - Data-centered architectures
  - Event-based architectures

Architectural Styles



layered architectural style



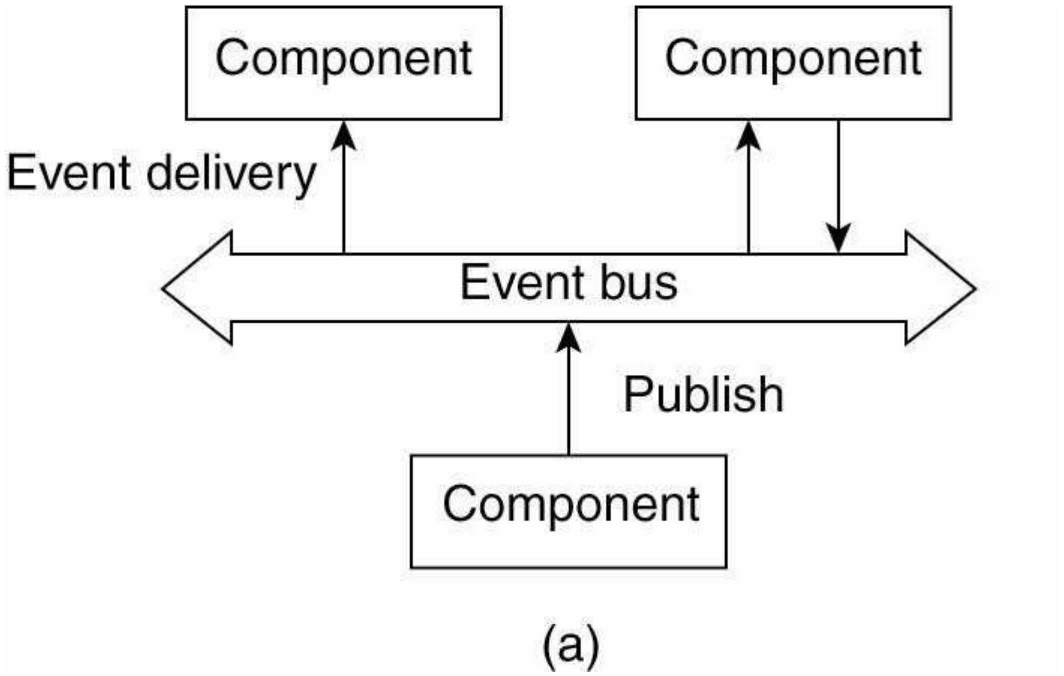
The object-based architectural style.

Object based is less structured
component = object
connector = RPC or RMI

Data-Centered Architectures

- Main purpose: data access and update
- Processes interact by reading and modifying data in some shared repository (active or passive)
  - Traditional data base (passive): responds to requests
  - Blackboard system (active): clients solve problems collaboratively; system updates clients when information changes.

Event-based architectural style



Event-based arch. supports several communication styles:

- Publish-subscribe
- Broadcast
- Point-to-point

E.g., register interest in market info; get email updates

Distribution Transparency

- An important characteristic of software architectures in distributed systems is that they are designed to support distribution transparency.
- Transparency involves trade-offs (balance achieved between two desirable but incompatible features)
- Different distributed applications require different solutions/architectures

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 Client Server and P2P
  - Edge-server systems
  - Collaborative distributed systems.

Traditional Client-Server(Centralized)

- Processes are divided into two groups (clients and servers).
- Synchronous communication: request-reply protocol
- In LANs, often implemented with a <u>connectionless</u> protocol (unreliable)
- In WANs, communication is typically <u>connection-oriented</u> TCP/IP (reliable)

Client Server Architectures

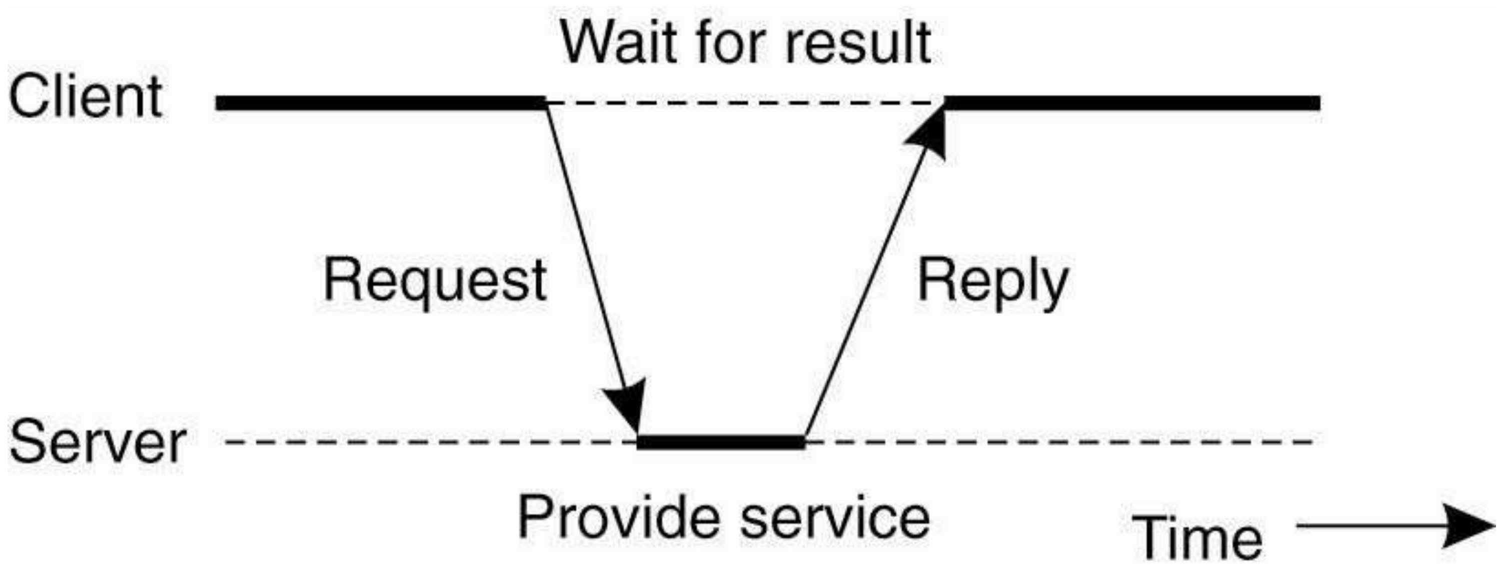


Figure 2-3. General interaction between a client and a ser ver.

Transmission Failures

- With connectionless transmissions, failure of any sort means no reply
- Possibilities:
  - Request message was lost
  - Reply message was lost
  - Server failed either before, during or after performing the service

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
  - File or database system

Example:

- **Web search engine**
  - **Interface:** type in a keyword string
  - **Processing level:** processes to generate DB queries
  - **Data level:** database of web pages
- **Desktop "office suites"**
  - **Interface:** access to various documents, data,
  - **Processing:** word processing, database queries, spreadsheets,…
  - **Data :** file systems and/or databases

Application Layering

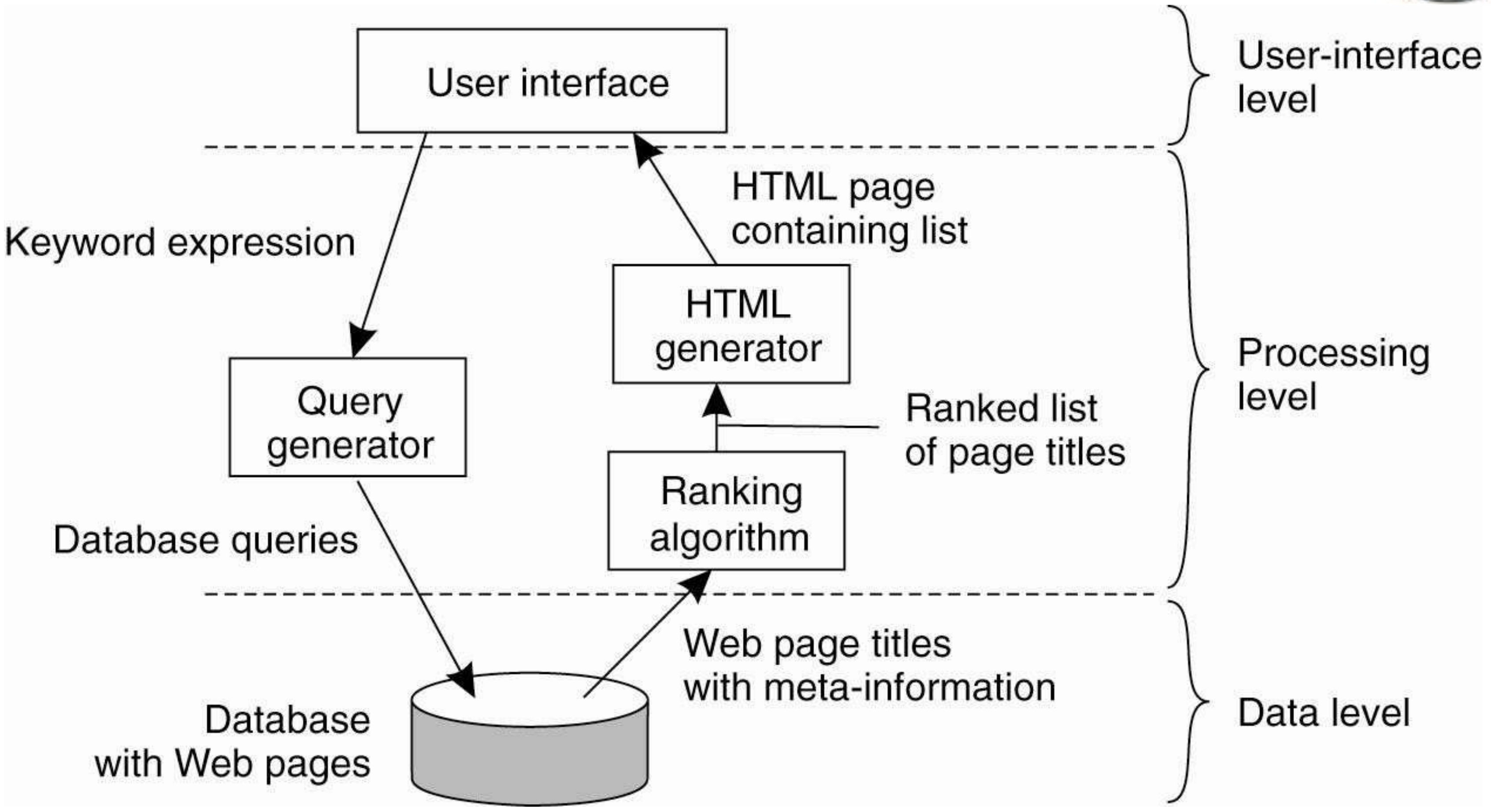


Figure 2-4. organization of an Internet search engine into three different layers.

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 Client Server Architectures

- Server provides processing and data management; client provides simple graphical display (**thin-client**)
  - Perceived performance loss at 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**)
  - Pro: reduces work load at server; more scalable
  - Con: harder to manage by system admin, less secure

Multi tiered Architectures

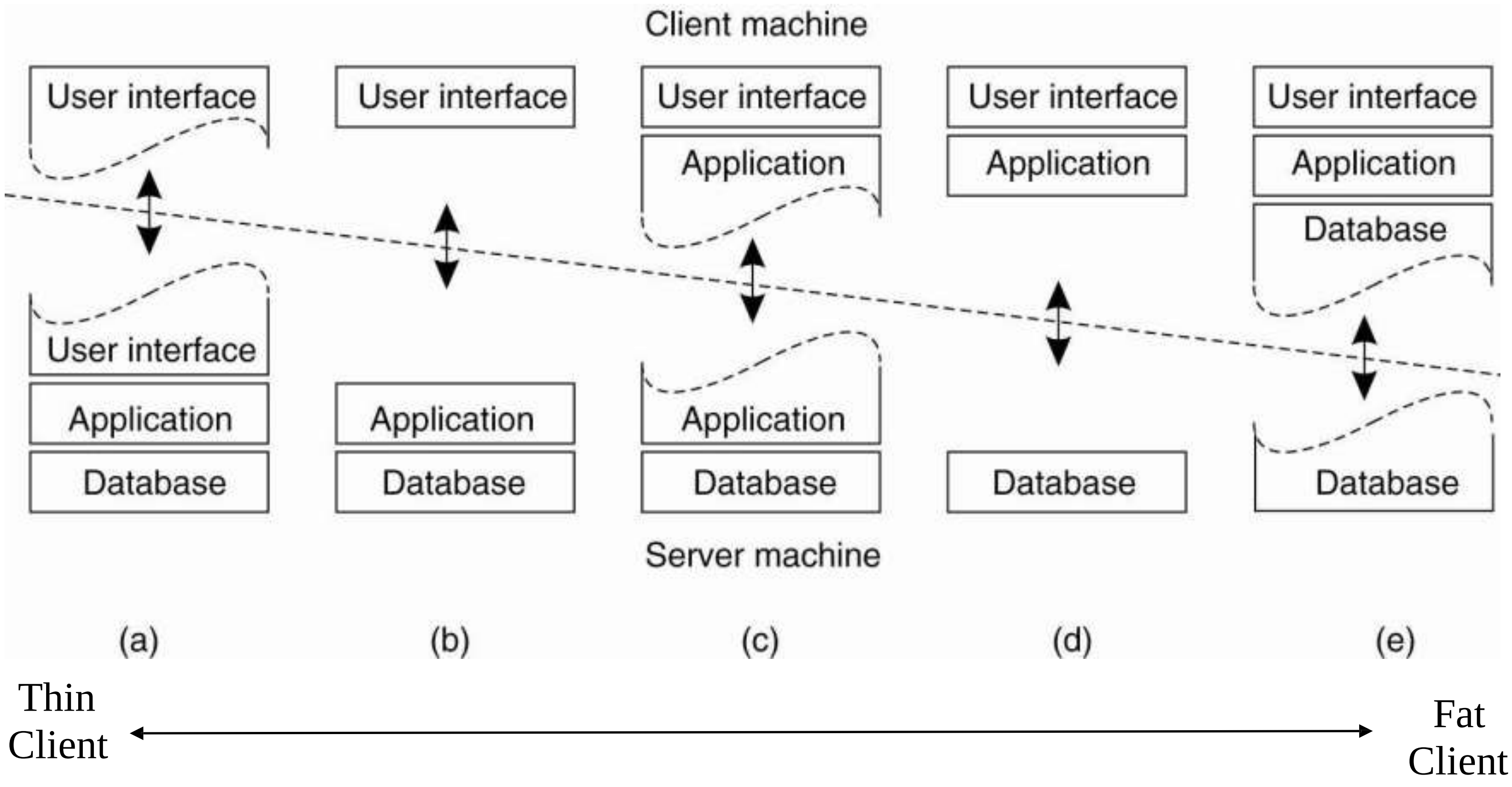


Figure 5. Alternative client-server organizations (a)–(e).

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.

Multi tiered Architectures (3 Tier Architecture)

- 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.

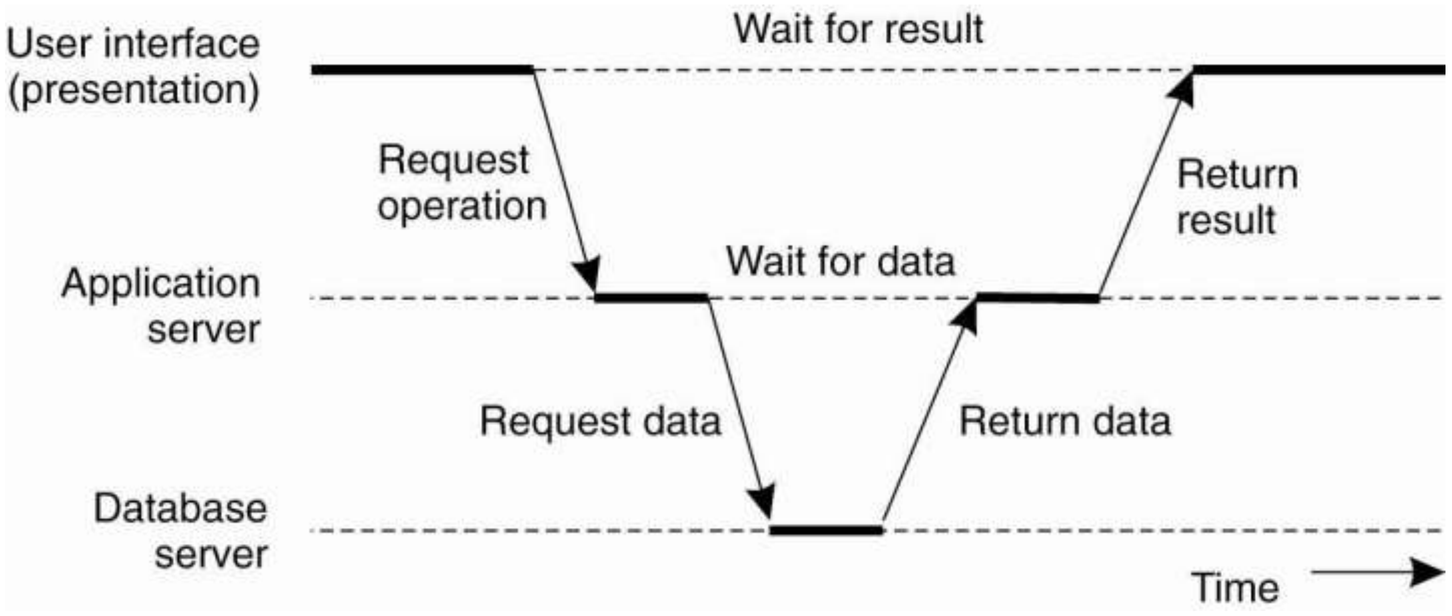


Figure 6. An example of a server acting as client.

Centralized vs. Decentralized Architectures

- **Vertical distribution**: traditional client-server architectures
  - 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 or manages part of the total system data.

  - Better load balancing, more resistant to denial-of-service attacks, harder to manage than C/S
  - Communication & control is not hierarchical; all about equal

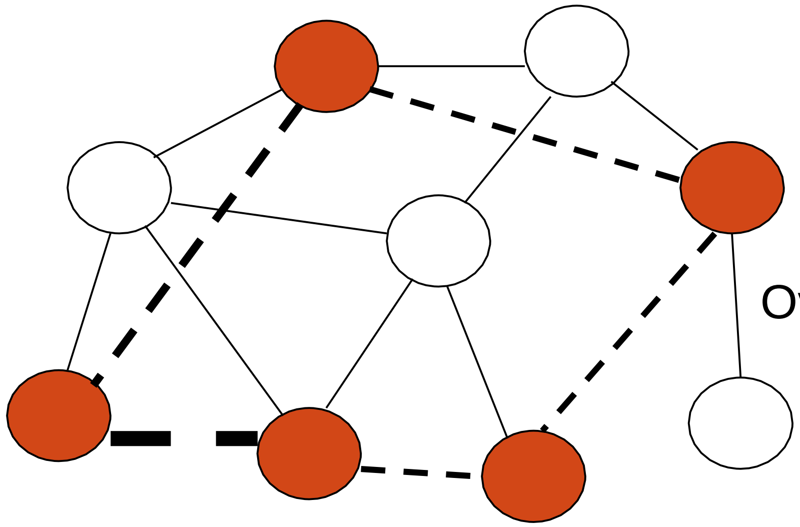
Peer-to-Peer

- Nodes act as both client and server
- Each node acts as a server for part of the total system data

- **Overlay/stored instruction/ 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



Overlay Network Example

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

Overlay Networks

- The overlay network may be
  - Structured (nodes and content are connected according to some design that simplifies later lookups) or
  - Unstructured (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
- Traditional hash functions convert a key to a hash value, which can be used as an index into a hash table.
  - **Keys are unique** – each represents an object to store in the table
  - The hash function value is used to insert an object in the hash table and to retrieve it.

Structured P2P Architectures

- 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 lookup, also based on hash function value, returns the network address of the node that stores the requested object.

Characteristics of DHT

- **Scalable** – to thousands, even millions of network nodes
- **Fault tolerant** – able to re-organize itself when nodes fail
- **Decentralized** – no central coordinator

Chord Routing Algorithm Structured P2P

- Nodes are logically arranged in a circle
- Nodes and data items have m-bit identifiers (keys) from a 2^m namespace.
    - *e.g.*, a node's key is a hash of its IP address and a file's key might be the hash of its name or of its content or other unique key.

Structured Peer-to-Peer Architectures

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` (mod 2^m)
- This node is the successor of `k`, or `succ(k)`
- Modular arithmetic is used

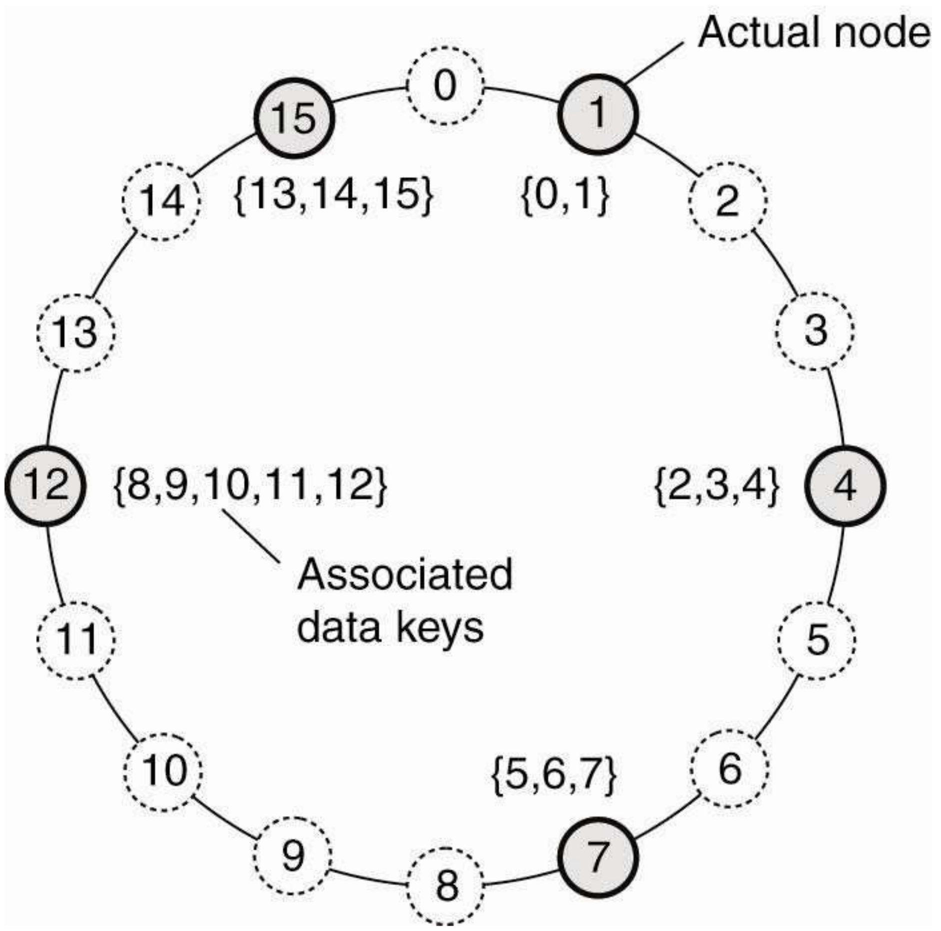


Figure 2.The mapping of data items onto nodes in Chord for m = 4, $2^{4 \text{ (hashvalues}} = 16$ (key nodes)

Unstructured P2P

- Unstructured P2P organizes the overlay network as a random graph.
- Each node knows about a subset of nodes, its “neighbors”.
- Data items are randomly mapped to some node in the system & lookup is random, unlike the structured lookup in Chord.

Locating a Data Object by Flooding

- Send a request to all known neighbors
- Works well in small to medium sized networks, doesn’t scale well
- “Time-to-live” counter can be used to control number of hops
- Example system: Freenet

Comparison

- **Structured networks** typically guarantee that if an object is in the network it will be located in a bounded amount of time – usually *O(log(N))*
- **Unstructured networks** offer no guarantees.
  - For example, some will only forward search requests a specific number of hops.
  - Random graph approach means there may be loops
  - Graph may become disconnected

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 Client Server system, then operate in decentralized manner.

Edge-Server Systems

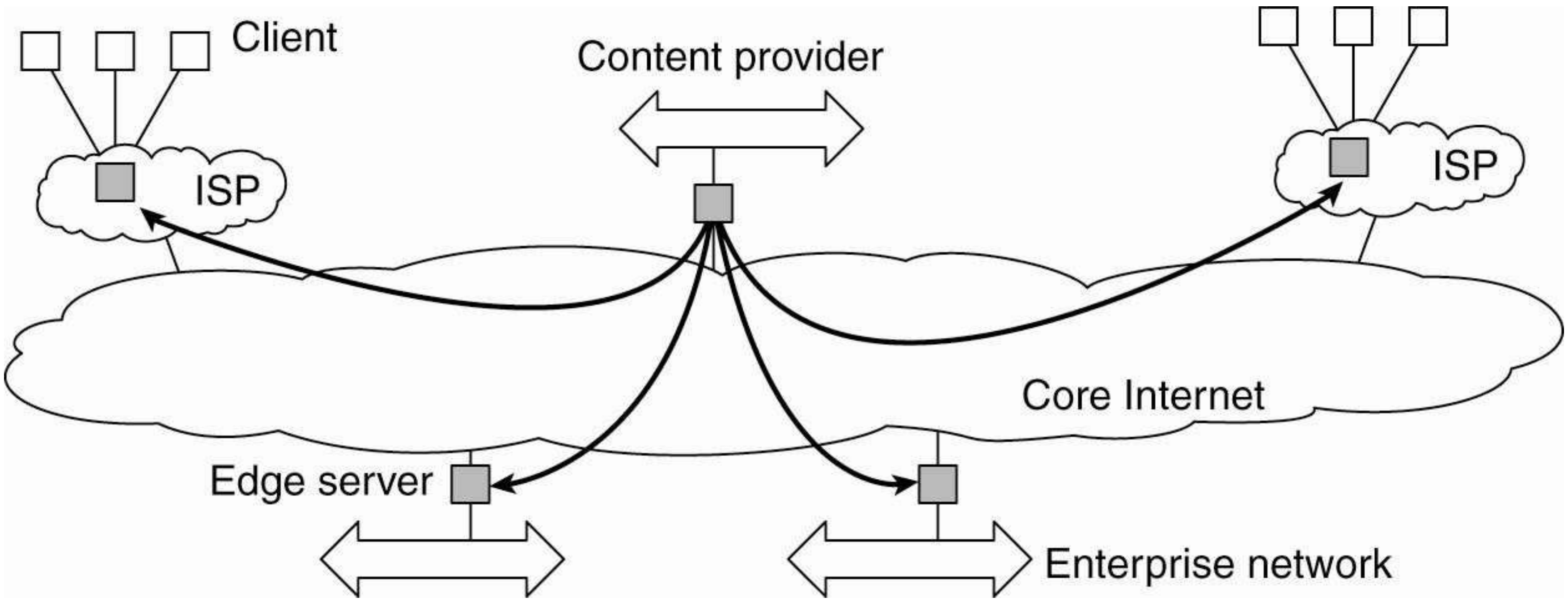


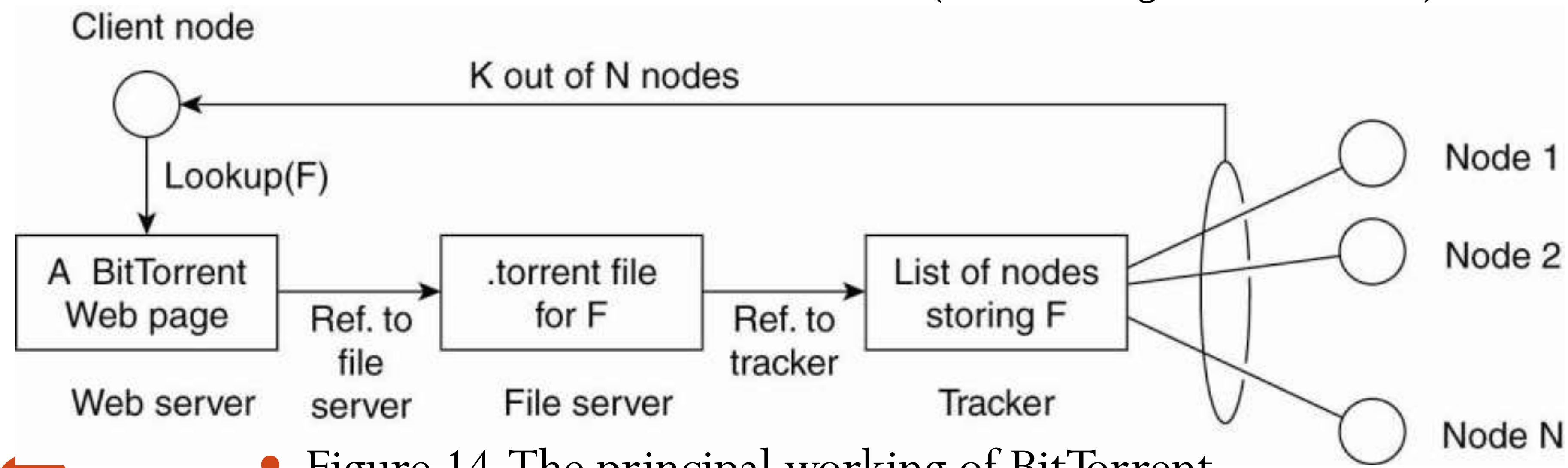
Figure 13.Viewing the Internet as consisting of a collection of edge servers.

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.

Trackers know which nodes are active (downloading chunks of a file)



- Figure 14.The principal working of BitTorrent

Freenet

- “Freenet is free software which lets you publish and obtain information on the Internet without fear of restriction.
- To achieve this freedom, the network is entirely decentralized and publishers and consumers of information are unidentified.
- Without anonymity/unidentified there can never be true freedom of speech, and without decentralization the network will be vulnerable to attack.”

P2P v Client/Server

- **P2P** computing allows end users to communicate without a dedicated server.
- There is less likelihood of performance bottlenecks since communication is more distributed.
  - Data distribution leads to workload distribution.

- Resource discovery is more difficult than in centralized client-server computing & look-up/retrieval is slower
- **P2P** can be more fault tolerant, more resistant to denial of service attacks because network content is distributed.
  - Individual hosts may be unreliable, but overall, the system should maintain a consistent level of service

Architecture versus Middleware

- Where does middleware fit into an architecture?
- **Middleware:** the software layer between user applications and distributed platforms.
- **Purpose:** to provide distribution transparency
  - Applications can access programs running on remote nodes without understanding the remote environment.
- Middleware may also have an architecture
  - e.g, CORBA has an object-oriented style.

- Use of a specific architectural style can make it easier to develop applications, but it may also lead to a less flexible system.
- **Possible solution:** develop middleware that can be customized as needed for different applications.

Thank you

Distributed Systems

Chapter 3: Process in Distributed System

Inline

- Thread
- Virtualization
- Client
- Server
- Code Migration

Overview of the process

- Process is a program that executes on the operating system's virtual processors.
- The management and scheduling of processes are perhaps the most important issues to deal with.
- Thread? is smallest unit of executable code that performs a particular task
- Multithreading techniques results in a high level of performance.
- Virtualization deals with extending/replacing an existing interface to mimic the behavior of another system
- In virtualization, the diversity of platforms and machines can be reduced by letting each application run on its own virtual machine.
- It support a replication of dynamic content . Besides, it allows to run concurrently with other applications, but highly independent of the underlying hardware and platforms, leading to a high degree of portability.
- Also virtualization helps in isolating failures caused by errors or security problems.
- Process migration or more specifically, code migration, can help in achieving scalability, but can also help to dynamically configure clients and servers.

Process Vs Threads

- Process is a collection of one or more threads and associated system resources. It Cannot exist by its own, it must be part of process,
- Unlike processes, thread does not have a separate allocation of memory, but shares memory with other threads created
- Thread is building block in distributed systems, lightweight process that can be managed by scheduler.
- To execute a program, an operating system creates a number of virtual processors, each one for running a different program.
- A process **table,** containing entries to store CPU register values, memory maps, open files, accounting information.
- Multiple processes may be concurrently sharing the same CPU and other hardware resources is made transparent this refers to concurrency transparency
- When a process is created, the operating system must create a complete independent address space.

Process Vs Threads con't

- Like a process, a thread executes its own piece of code, independently from other threads.
- But no attempt is made to achieve a high degree of concurrency transparency if this would result in performance degradation
- A thread system generally maintains only the minimum information to allow a CPU to be shared by several threads it refers to multithreading.
- Multithreading leads to high performance
- Multithreading is that it becomes possible to exploit parallelism when executing the program on a multiprocessor system.
- A collection of cooperating programs, each to be executed by a separate process. Mean that a process contain many thread.

Process Vs Threads con't

Process

- Program in execution
- Not light weight
- Isolated/no shared memory
- Not share data
- Long time to communication
- Independent each other

Thread

- Segmentation of process
- Light weight
- Share memory
- Share data each other
- Short time to communication
- Dependent each other

Threads in Distributed Systems

- Threads can provide a convenient means of allowing blocking system calls without blocking the entire process in which the thread is running.
- This property makes threads particularly attractive to use in distributed systems as it makes it much easier to express communication in the form of maintaining multiple logical connections at the same time.
- We illustrate this point by taking a closer look at multithreaded clients and servers, respectively

Multithreaded Clients

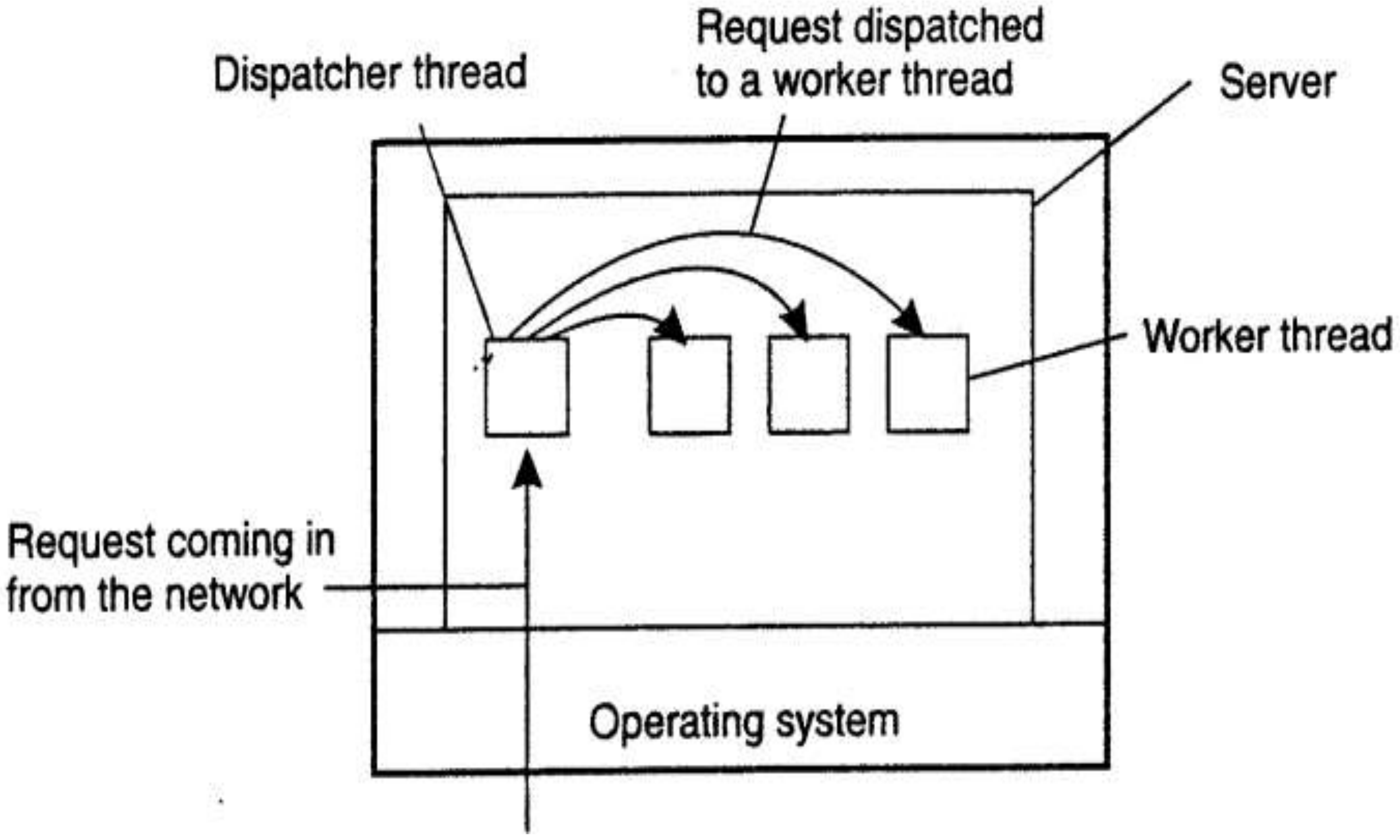
- To establish a high degree of distribution transparency, distributed systems that operate in wide-area networks may need to conceal long inter-process message propagation times.
- A Web browser often starts with fetching the HTML page and subsequently displays it.
- To hide communication latencies as much as possible, some browsers start displaying data while it is still coming in.
- Web browser is doing a number of tasks simultaneously.
- As soon as the main HTML file has been fetched, separate threads can be activated to take care of fetching the other parts.
- Each thread sets up a separate connection to the server and pulls in the data.

Multithreaded Sever

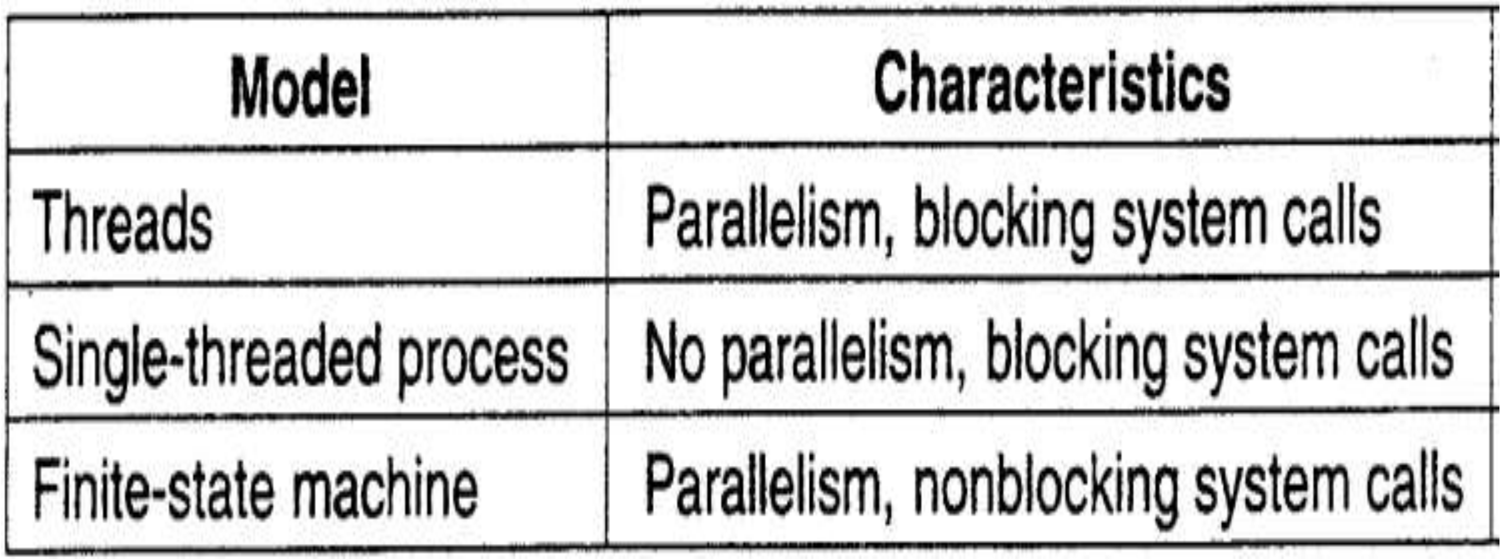
- The main use of multithreading in distributed systems is found at the server side.
- Also makes it much easier to develop servers that exploit parallelism to attain high performance, even on uni-processor systems.
- Multithreading for parallelism is even more useful.
- The file server normally waits for an incoming request for a file operation, subsequently carries out the request, and then sends back the reply.

- Here one thread, the **dispatcher,** reads incoming requests for a file operation.
- The requests are sent by clients to a well-known end point for this server.
- After examining the request, the server chooses an idle (i.e., blocked) **worker thread** and hands it the request.
- The worker proceeds by performing a blocking read on the *local* file system, which may cause the thread to be suspended until the data are fetched from disk.
- If the thread is suspended, another thread is selected to be executed.

A multithreaded server organized in a dispatcher/worker mode.


Dispatcher thread
Request dispatched to a worker thread
Server
Worker thread
Request coming in from the network
Operating system

- Three ways to construct a server.



Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls

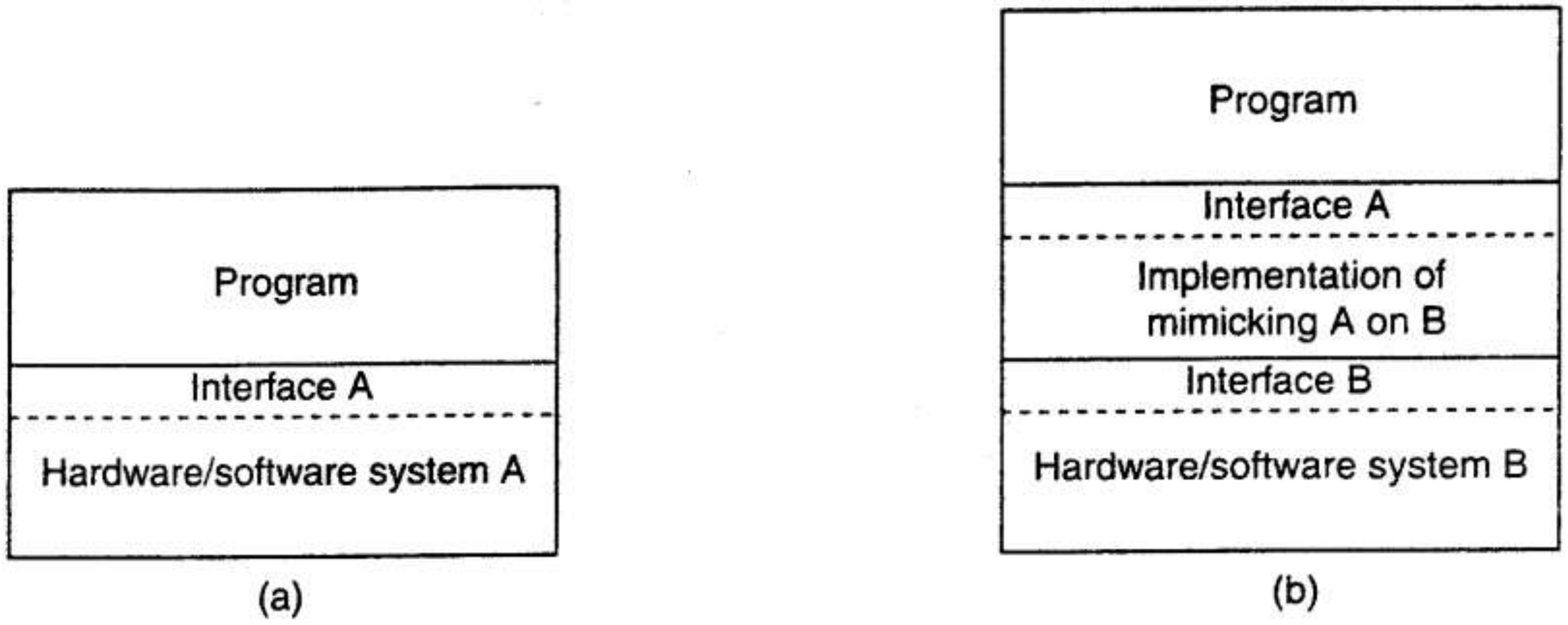
VIRTUALIZATION

- By rapidly switching between threads and processes, the illusion of parallelism is created (executed simultaneously).
- It is impossible in single processor
- This separation between having a single CPU and being able to pretend there are more can be extended to other resources as well, leading to what is known as resource virtualization.

- **The Role of Virtualization in Distributed Systems**

- Virtualization deals with extending or replacing an existing interface so as to mimic/replicate the effect/ the behaviour of another system.

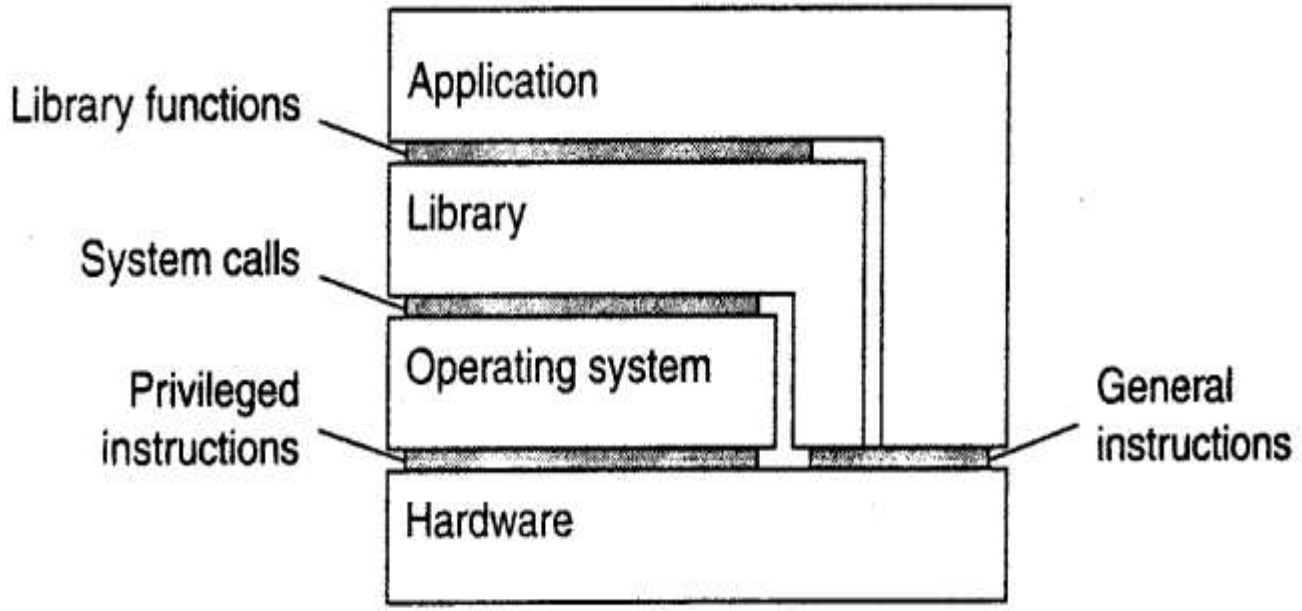
VIRTUALIZATION con’t


Program
Interface A
Hardware/software system A
(a)
Program
Interface A
Implementation of mimicking A on B
Interface B
Hardware/software system B
(b)

(a) General organization between a program, interface, and system.
(b) General organization of veritualizing system A on top of system B.

Architectures of Virtual Machines

- Computer systems generally offer four different types of interfaces, at four different levels
- The essence of virtualization is to mimic the behaviour of these interfaces

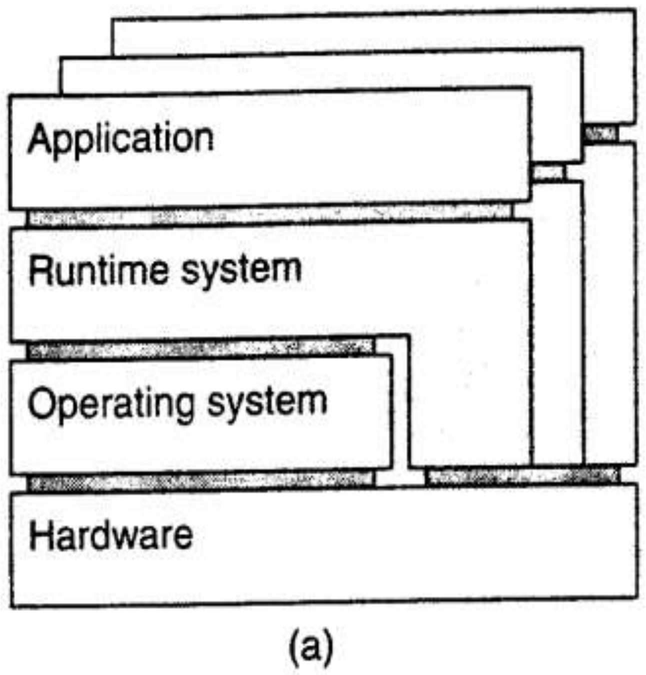


Architectures of Virtual Machines con’t

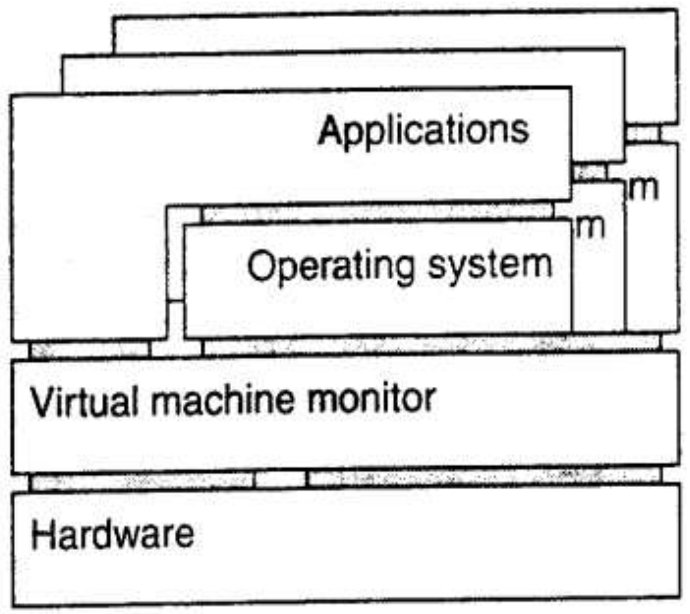
- Virtualization can take place in two different ways.

(a) A process virtual machine, with multiple instances of (application, runtime) combinations.

(b) A virtual machine monitor. with multiple instances of (applications, operating system) combinations



(a)



(b)

CLIENTS

- A major task of client machines is to provide the means for users to interact with remote servers.
- There are roughly two ways in which this interaction can be supported.
- First, for each remote service the client machine will have a separate counterpart that can contact the service over the network.

  A second solution is to provide direct access to remote services by only offering a convenient user interface.

Client-Side Software for Distribution Transparency

- Client software comprises more than just user interfaces.
- In many cases, parts of the processing and data level in a client-server application are executed on the client side as well.
- A special class is formed by embedded client software, such as for cash registers, barcode readers, TV set-top boxes, etc.
- In these cases, the user interface is a relatively small part of the client software, in contrast to the local processing and communication facilities.
- Client software comprises components for achieving distribution transparency.

    Access transparency

    Location, migration

    Relocation transparency

    Replication transparency

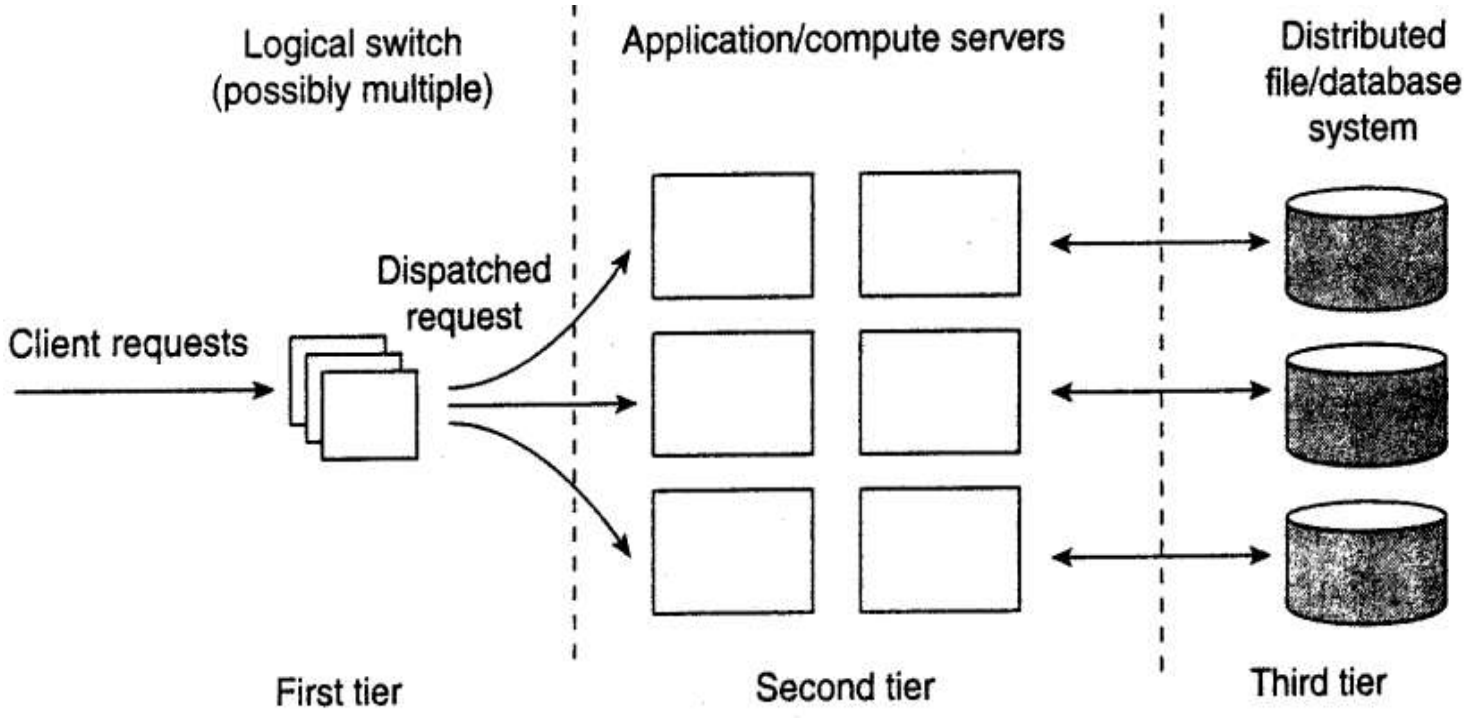
SERVERS

General Design Issues

- A server is a process implementing a specific service on behalf of a collection of clients.
- In essence, each server is organized in the same way: it waits for an incoming request from a client and subsequently ensures that the request is taken care of, after which it waits for the next incoming request.
- A multithreaded server is an example of a concurrent server.
- The place where clients contact a server in all cases, clients send requests to an end point, also called a port, at the machine where the server is running.

Server Clusters

- A server cluster is nothing else but a collection of machines connected through a network, where each machine runs one or more servers.
- Connected through a local-area network, often offering high bandwidth and low latency.
- Logically organized into three tiers
- The general organization of a three-tiered server cluster.


Logical switch (possibly multiple)
Application/compute servers
Distributed file/database system
Dispatched request
Client requests
First tier
Second tier
Third tier

CODE MIGRATION

- Situations in which passing programs, sometimes even while they are being executed, simplifies the design of a distributed system.
- Moving a running process to a different machine is a costly and intricate task,
- There had better be a good reason for doing so. That **reason** has always been performance.

Thank you

Questions?

Quiz

1. Discus the difference between thread and process in distributed system.
2. What are the types of transparency in distributed systems?, and write each transparency differences
3. How do you manage Virtualization in multithreading in distributed concepts?