Unit 2: Architecture (4 Hrs.)

Background

2.1 Architectural Structure

2.2 Middleware organization

2.3 System Architecture

2.4 Example Architecture

Background

- Distributed systems are often complex pieces of software of which the components are dispersed across multiple machines.
- To master their complexity, it is crucial that these systems are properly organized.
- The software architectures tell us how the various software components are to be organized and how they should interact.

2.1 ARCHITECTURAL STYLE

1. Layered architectures
2. Object-based architectures
3. Data-centered architectures
4. Event-based architectures

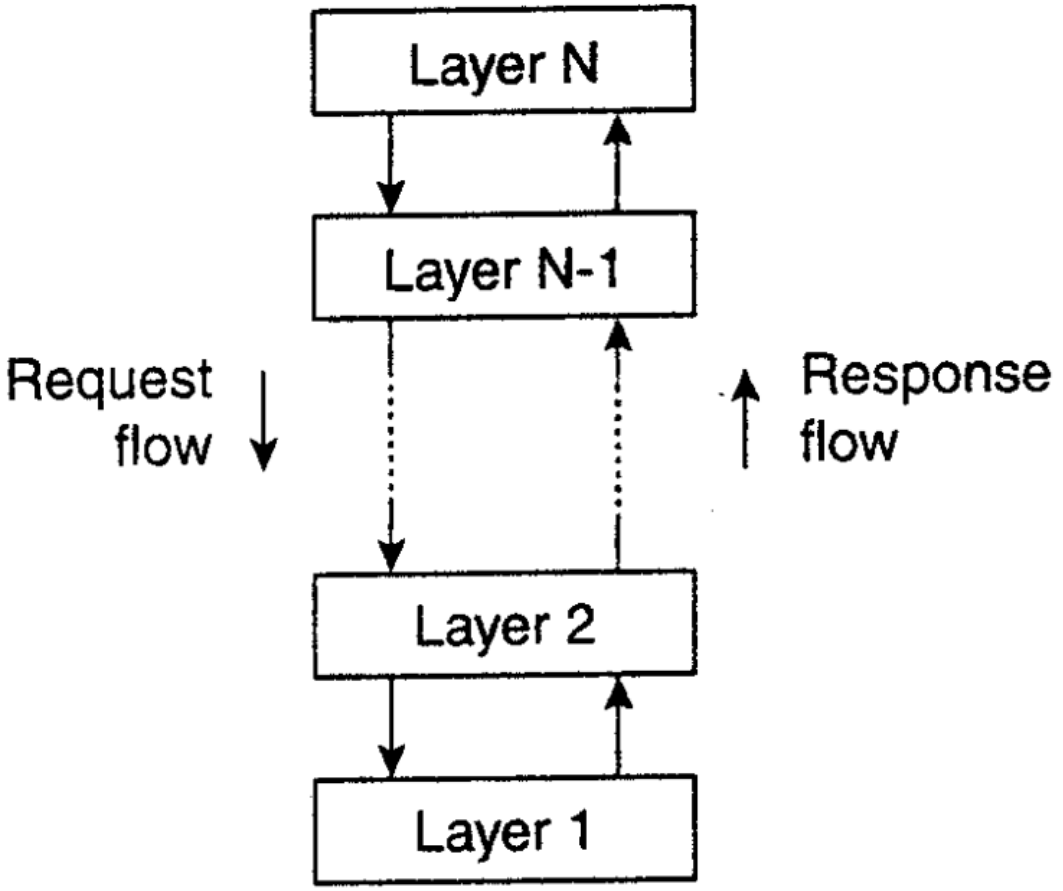
Layered architectures

- **Basic Idea:**

  --components are organized in a layered fashion where a component at layer L; is allowed to call components at Li, but not the other way around, as shown in Fig.

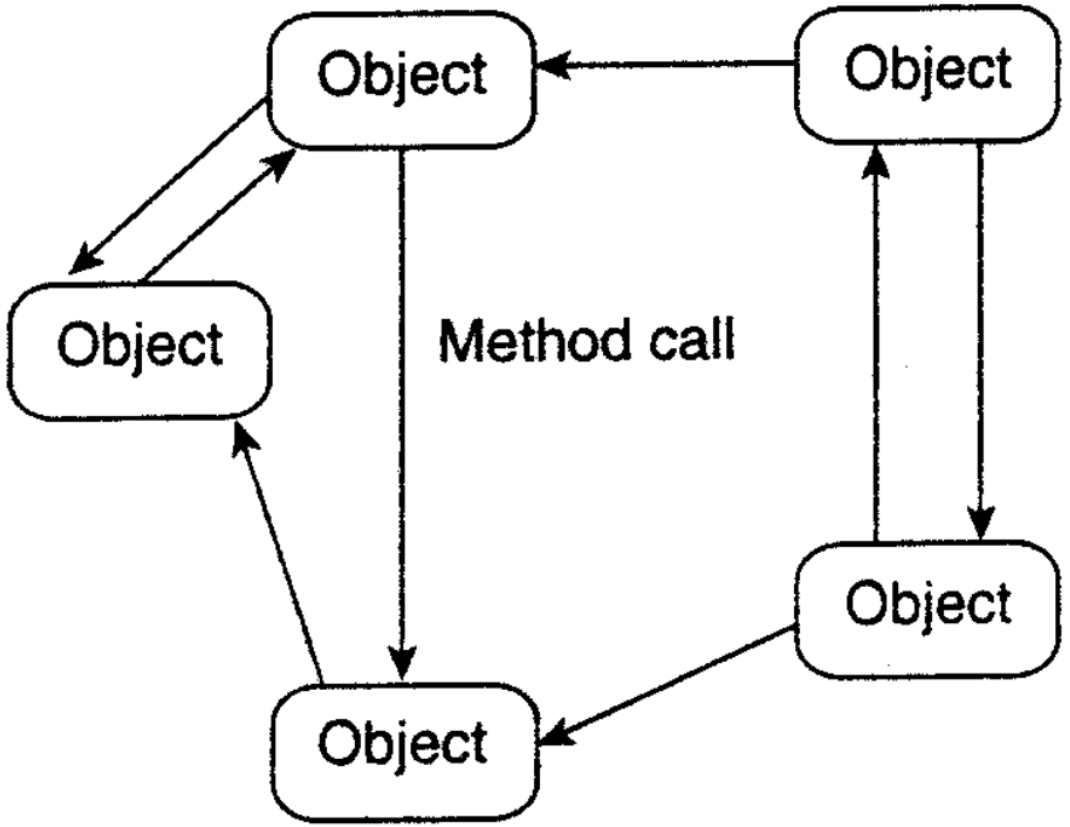
- Widely adopted by the networking community.

- Key observation is that control generally flows from layer to layer: requests go down the hierarchy whereas the results flow upward.



Object-based architectures

- This software architecture matches the client-server system architecture.
- Each object corresponds to what we have defined as a component, and these components are connected through a (remote) procedure call mechanism.



Data-centered architectures

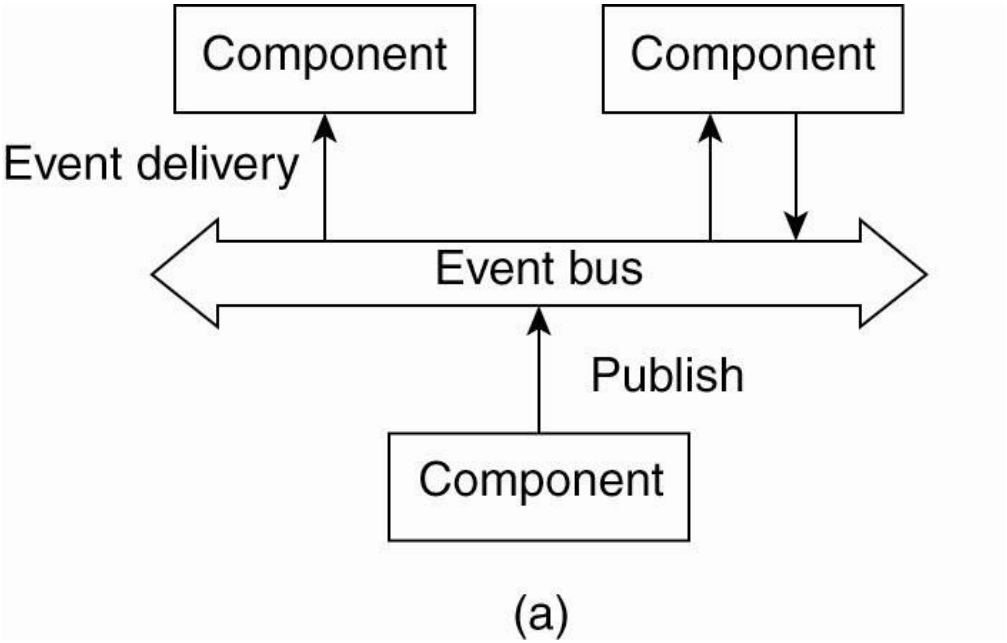
- Basic Idea:

--processes communicate through a common (passive or active) repository.

- As important as the layered and object based architectures. Examples:
  - Network applications: communicate through shared distributed file systems
  - Web applications: processes communicate through the use of shared Web-based data services

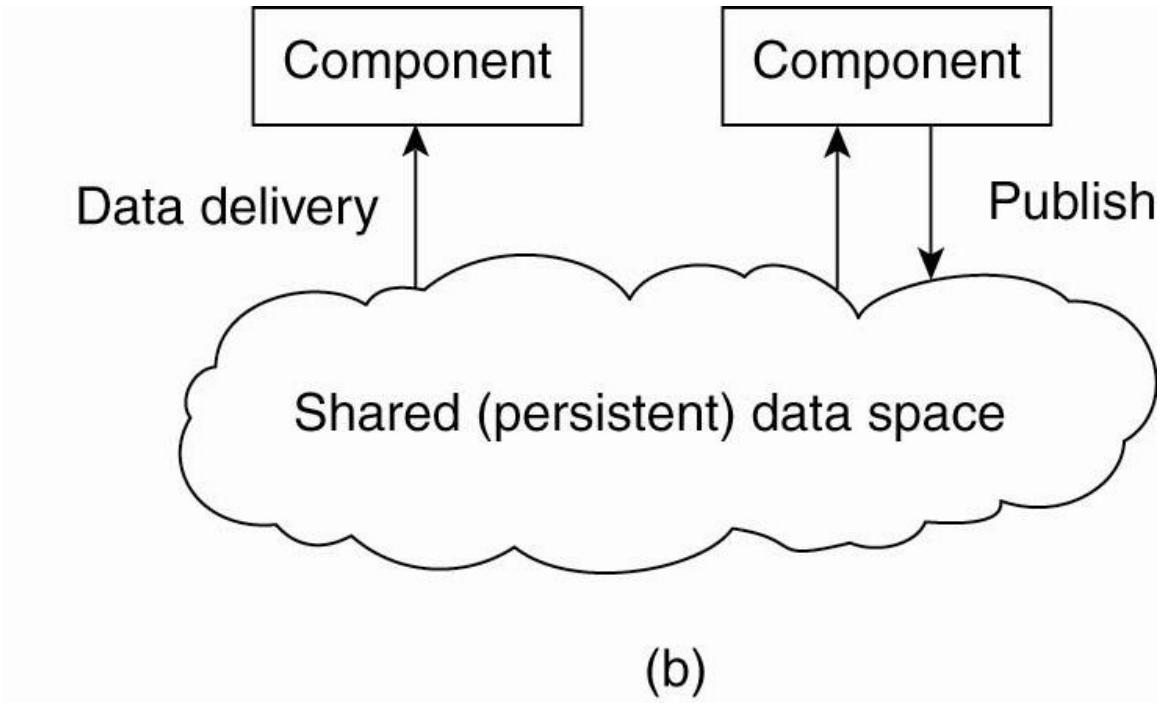
Event-based Architectural Style

- Processes essentially communicate through the propagation of events
  - Optionally also carry data
- Publish/subscribe systems
  - Only subscribed processes will receive the published events
  - **Referentially decoupled**: Processes are loosely coupled



Shared Data-space Architectural Style

- Combine event-based architectures and data-centered architectures
  - Processes are decoupled in time



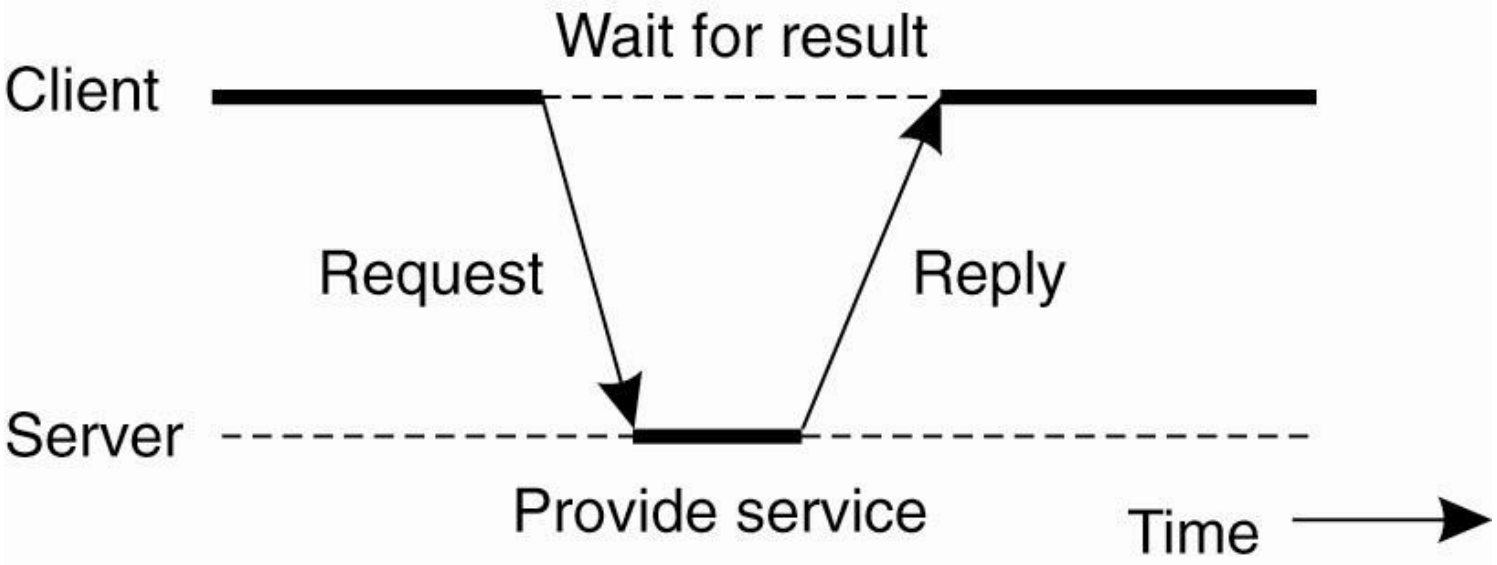
2.2 SYSTEM ARCHITECTURES

How many distributed systems are actually organized by considering where software components are placed.

- Centralized Architectures
- Decentralized Architectures
- Hybrid Architectures

Centralized Architectures

- Client-server model:
  - Processes are divided into two (possibly overlapping) groups
  - **Server**: a process implementing a specific service
    for example, a file system service or a database service.
  - **Client**: a process sending a request to a server and subsequently waiting for the server's reply



Communication between Clients and Servers

- Connectionless protocol
  - Efficient, but unreliable
    - Good for LANs

In these cases, when a client requests a service, it simply packages a message for the server, identifying the service it wants, along with the necessary input data. The message is then sent to the server. The latter, in turn, will always wait for an incoming request, subsequently process it, and package the results in a reply message that is then sent to the client.

- Connection-oriented protocol
  - Inefficient, but reliable
    - Good for WANs
- For example, virtually all Internet application protocols are based on reliable TCP/IP connections. In this case, whenever a client requests a service, it first sets up a connection to the server before sending the request. The server generally uses that same connection to send the reply message, after which the connection is torn down. The trouble is that setting up and tearing down a connection is relatively costly, especially when the request and reply messages are small.

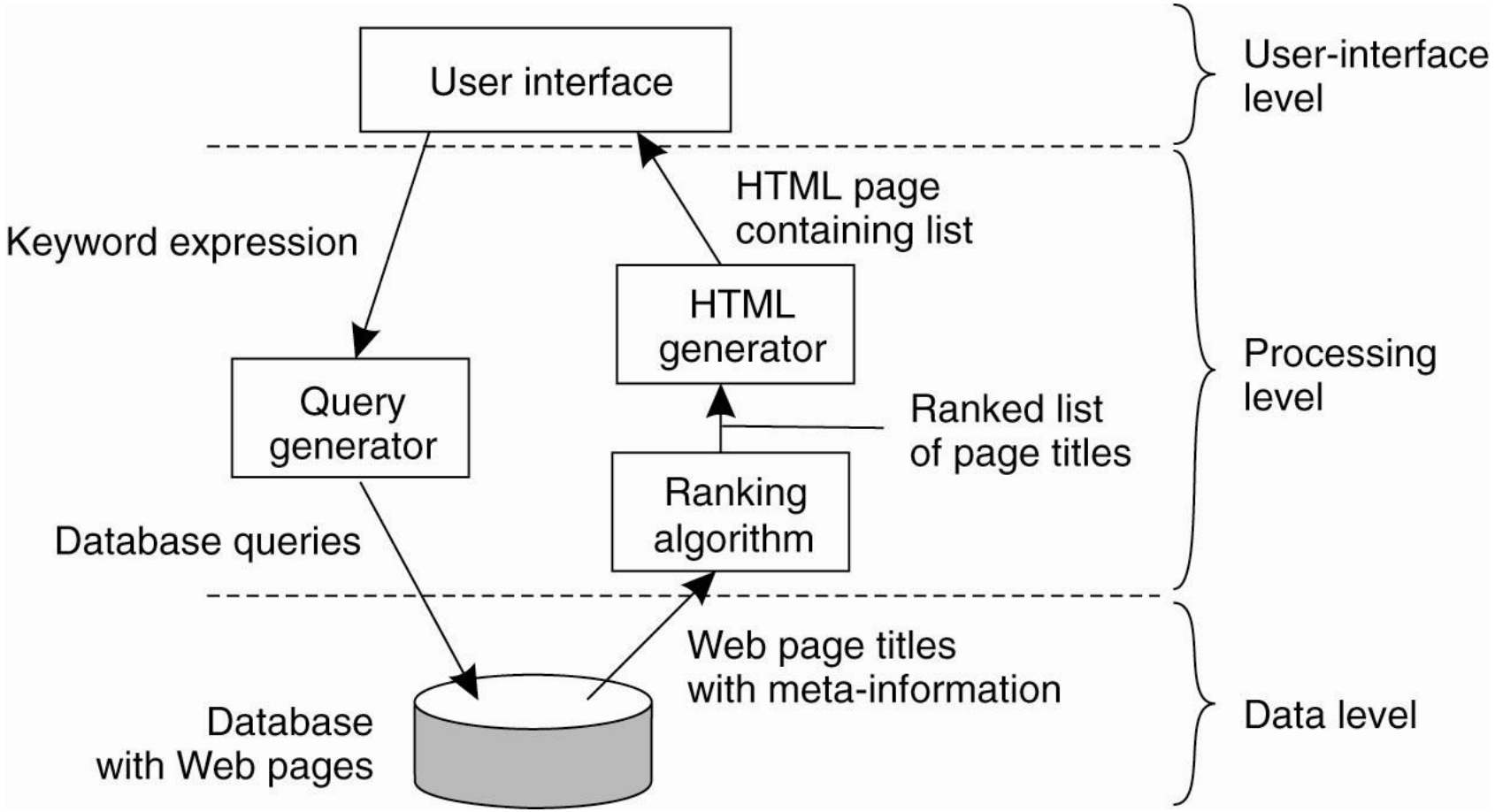
Application Layering

- Traditional three-layered view:
  - User-interface layer
    - Contains units for an application's user interface
  - Processing layer
    - Contains the functions of an application, i.e. without specific data
  - Data layer
    - Contains the data that a client wants to manipulate through the application components

- Observation:
  - This layering is found in many distributed information systems, using traditional database technology and accompanying applications.

Internet Search Engine

- The core : information retrieval part



More Examples

- A Stock Brokerage System
  - User Interface
  - Process Level
    - Analysis of financial data requires sophisticated methods and techniques from statistics and artificial intelligence
  - Data Level
    - Financial database
- Word Processor

Data Level

- Persistency of data
- Keeping data consistent across different applications
- Database
  - Relational database
  - Object-oriented database

Assignment-II (January 23rd)

- Difference between connectionless and connection oriented Protocol. Also focus on their comparative advantages and disadvantages.
- What is the main importance of separating the data level and application layer in Multi-tier Client Server Model. Illustrate.

Multi-Tiered Architecture

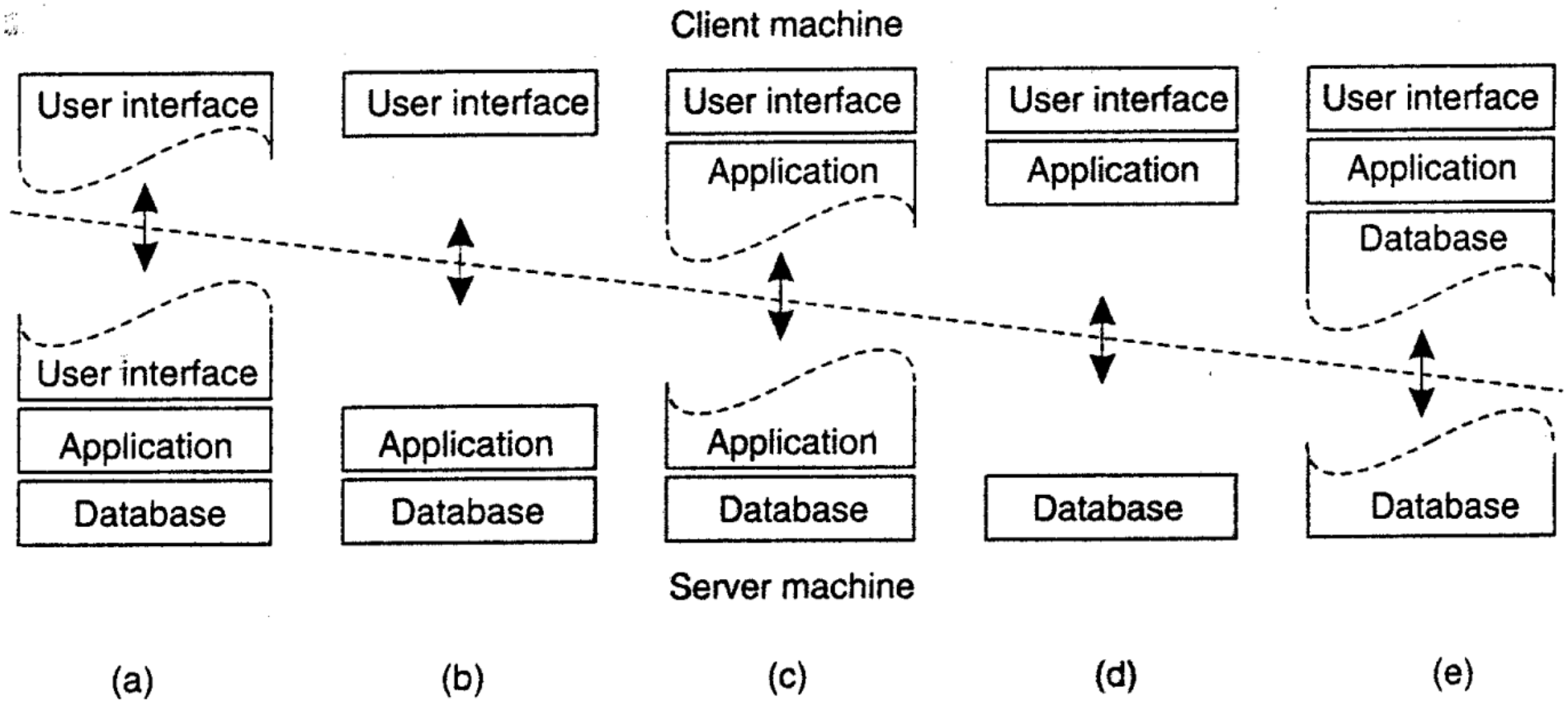
- The distinction into three logical levels as discussed so far, suggests a number of possibilities for physically distributing a client-server application across several machines.
- The simplest organization is to have only two types of machines:

  1. A client machine containing only the programs implementing (part of) the user-interface level

  2. A server machine containing the rest, that is the programs implementing the processing and data level

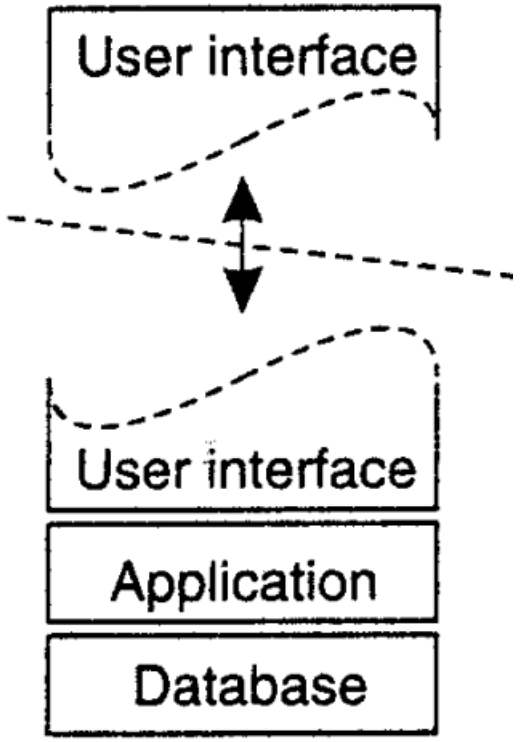
- In this organization everything is handled by the server while the client is essentially no more than a dumb terminal, possibly with a pretty graphical interface.

Alternative Client Server Organization



Client Server organization-I

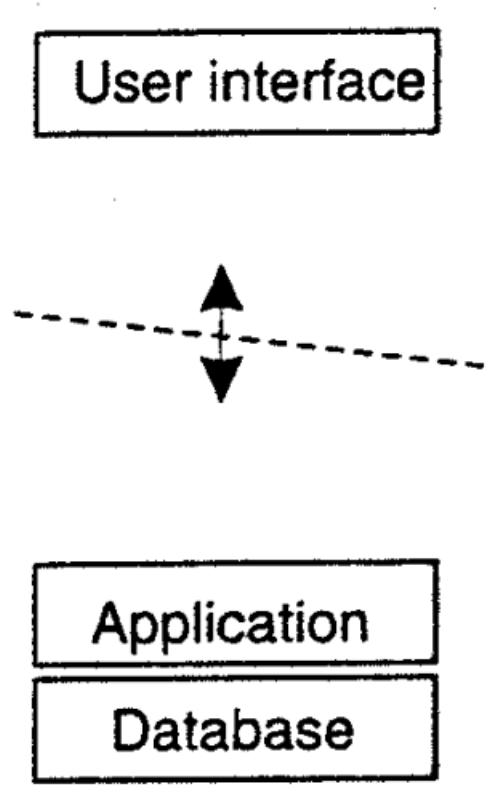
- One possible organization is to have only the terminal-dependent part of the user interface on the client machine and give the applications remote control over the presentation of their data.



(a)

Client Server organization-II

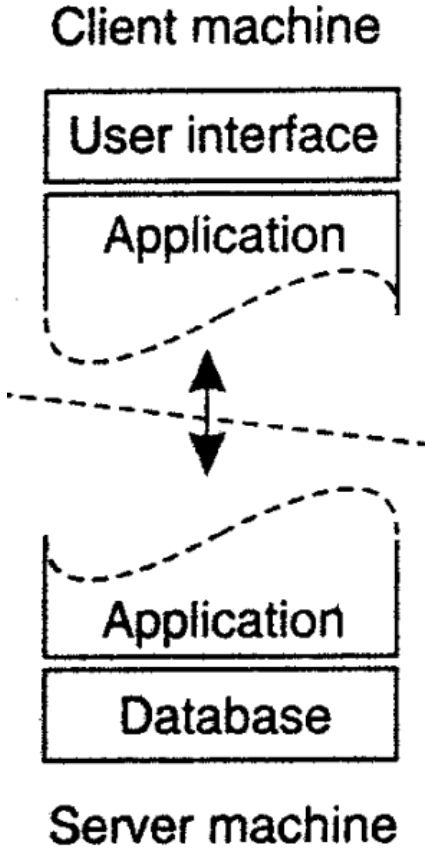
- An alternative is to place the entire user-interface software on the client side.
- In such cases, we essentially divide the application into a graphical front end, which communicates with the rest of the application (residing at the server) through an application-specific protocol.
- In this model, the front end (the client software) does no processing other than necessary for presenting the application's interface.



(b)

Client Server organization-III

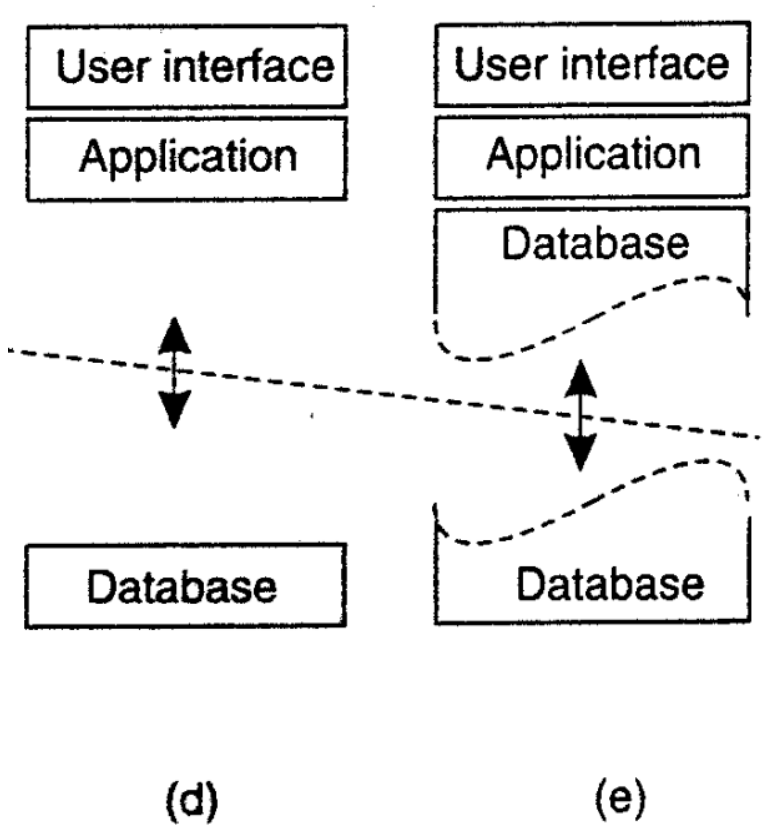
- Move part of the application to the front end.
- EG1. the application makes use of a form that needs to be filled in entirely before it can be processed. The front end can then check the correctness and consistency of the form, and where necessary interact with the user.
- EG2. a word processor in which the basic editing functions execute on the client side where they operate on locally cached, or in-memory data. but where the advanced support tools such as checking the spelling and grammar execute on the server side



(c)

Popular Client Server organization

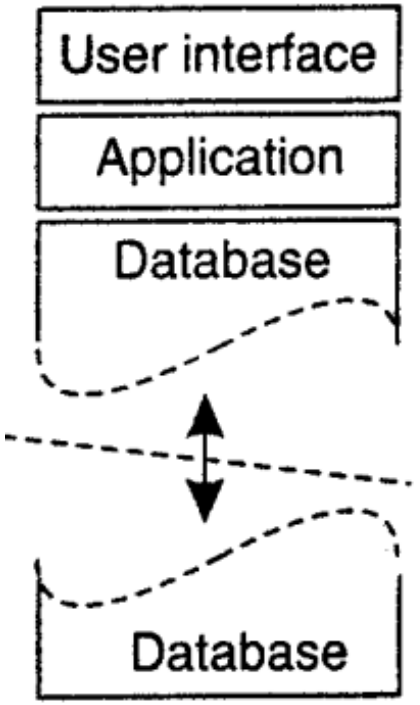
- used where the client machine is a PC or workstation, connected through a network to a distributed file system or database.
- Essentially, most of the application is running on the client machine, but all operations on files or database entries go to the server.



Example

- Many banking applications run on an end-user's machine where the user prepares transactions and such.
- Once finished, the application contacts the database on the bank's server and uploads the transactions for further processing.

- represents the situation where the client's local disk contains part of the data.
- For example, when browsing the Web, a client can gradually build a huge cache on local disk of most recent inspected Web pages.



(e)

Points to be Noted

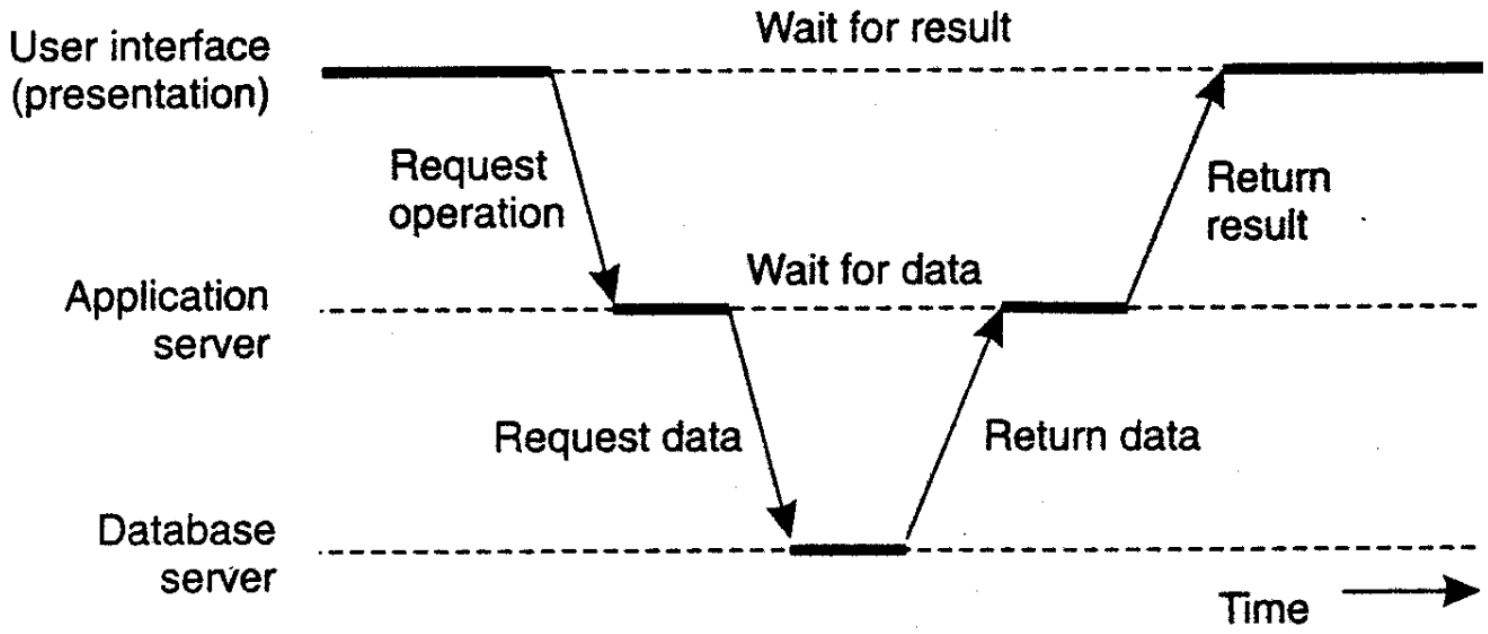
- There been a strong trend to move away from the configurations shown in Fig (d) and Fig (e) in those case that client software is placed at end-user machines.
- In these cases, most of the processing and data storage is handled at the server side.
- The reason for this is simple: although client machines do a lot, they are also more problematic to manage.
- Having more functionality on the client machine makes client-side software more prone to errors and more dependent on the client's underlying platform (i.e., operating system and resources).
- From a system's management perspective, having what are called fat clients is not optimal.
- Instead the thin clients as represented by the organizations shown in Fig (a)-(c) are much easier, perhaps at the cost of less sophisticated user interfaces and client-perceived performance.

Contd…
Point to be noted

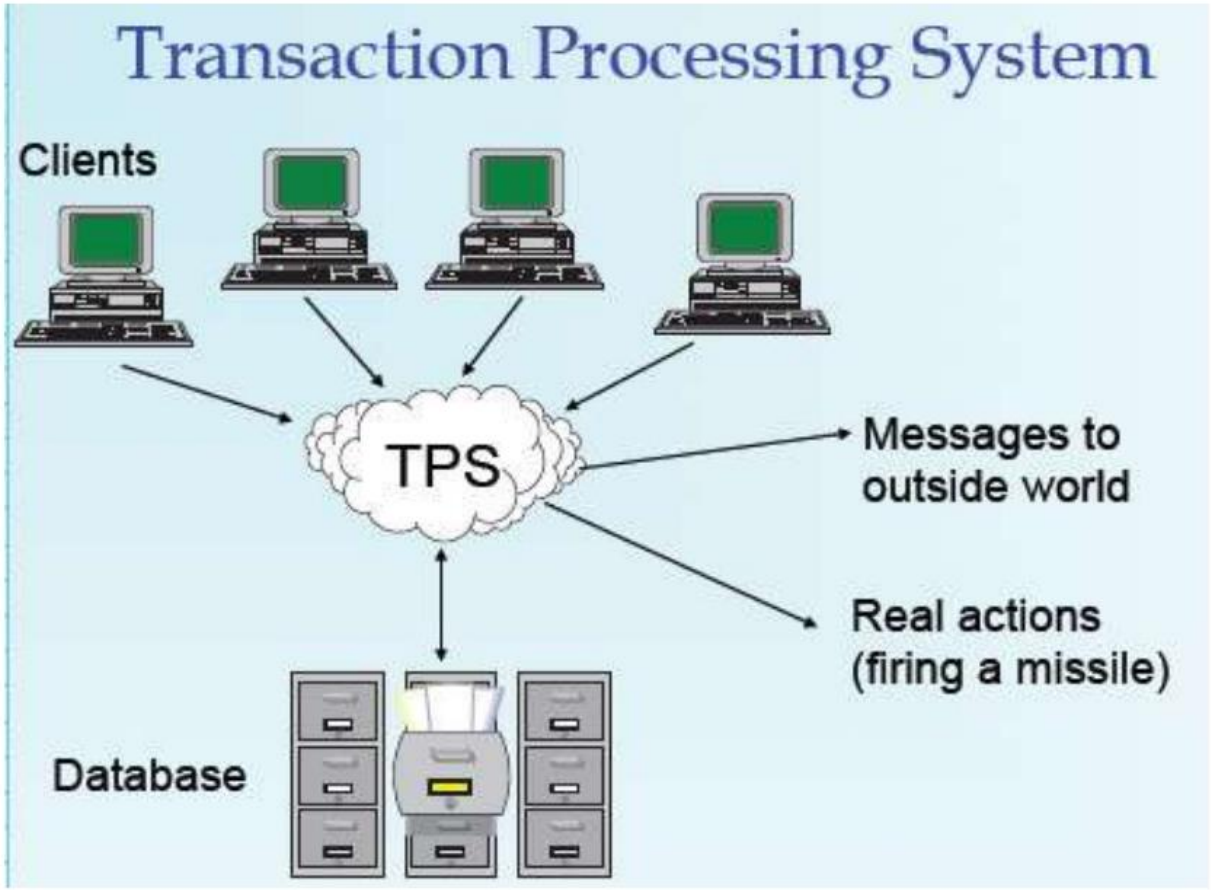
- Note that this trend does not imply that we no longer need distributed systems.
- On the contrary, what we are seeing is that server-side solutions are becoming increasingly more distributed as a single server is being replaced by multiple servers running on different machines.
- In particular, when distinguishing only client and server machines as we have done so far, we miss the point that a server may sometimes need to act as a client, leading to a (physically) three-tiered architecture.

An example of a server acting as client

- In this architecture, programs that form part of the processing level reside on a separate server, but may additionally be partly distributed across the client and server machines.
- Eg. Transaction Processing : a separate process, called the transaction processing monitor, coordinates all transactions across possibly different data servers



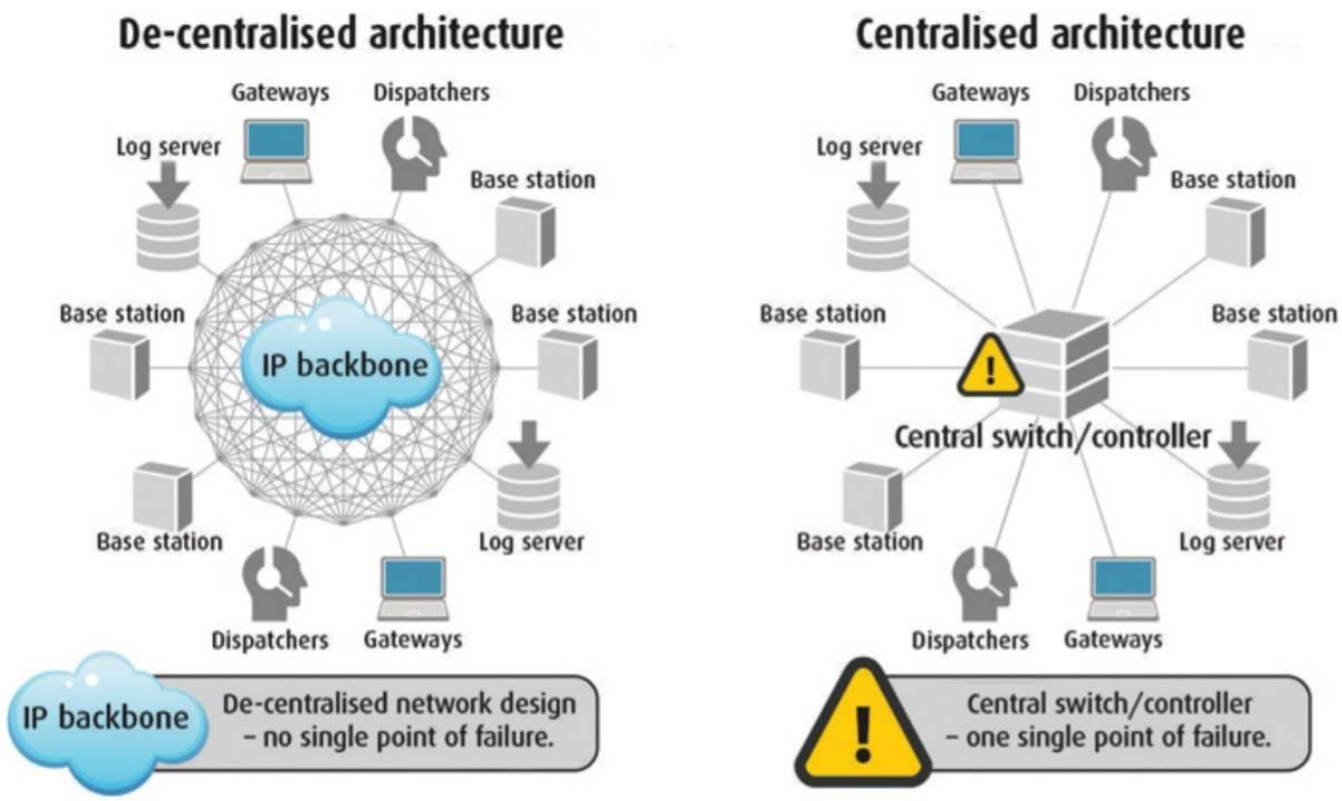
Transaction processing system



Example

- Another, but very different example where we often see a three-tiered architecture is in the organization of Web sites.
- In this case, a Web server acts as an entry point to a site, passing requests to an application server where the actual processing takes place.
- This application server, in turn, interacts with a database server.

Decentralized Architectures



- Multitiered client-server architectures → Vertical Distribution
- Vertical distribution is only one way of organizing client-server applications.
- Modern architectures →Horizontal Distribution → AKA peer-to-peer systems

P2P System

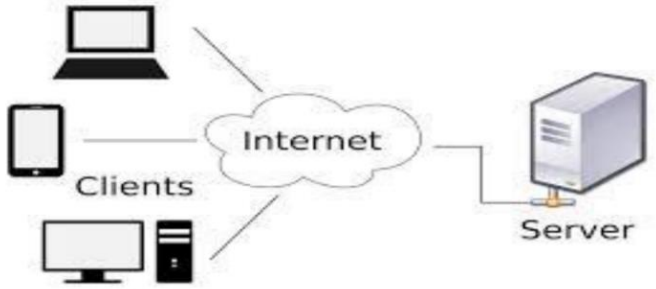
- Peer – to – Peer Computing model is based on how we (human) communicate in real world.
- If we need something then we communicate directly to other corresponding peers (may be friends) who may in turn refer us to their corresponding peers for working towards completion of the request.
- Thus there is a direct access between the peers without any third party intervention.

Examples

- Easy file sharing
- Efficient instant messaging
- Smooth voice communication
- Secure search and communication network
- High performance computing

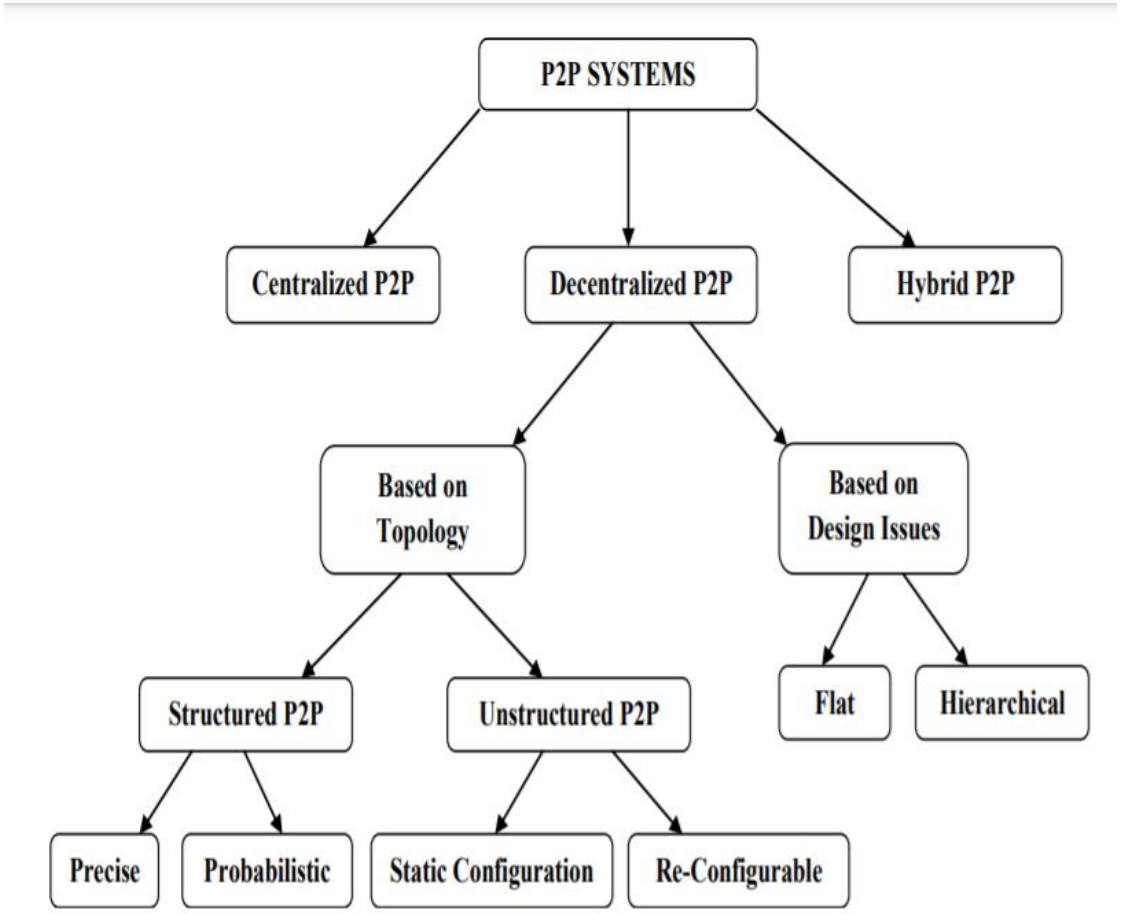
Evolution of P2P from Client-Server Model

- Many of the internet applications are using Client-Server model Example: WWW, email etc.
- In such model there will be a centralized server shared by many clients.
- The clients query the request to server and get services.
- It will be facilitated if the server is available and capable of serving all the requests from distinct clients at a particular moment.



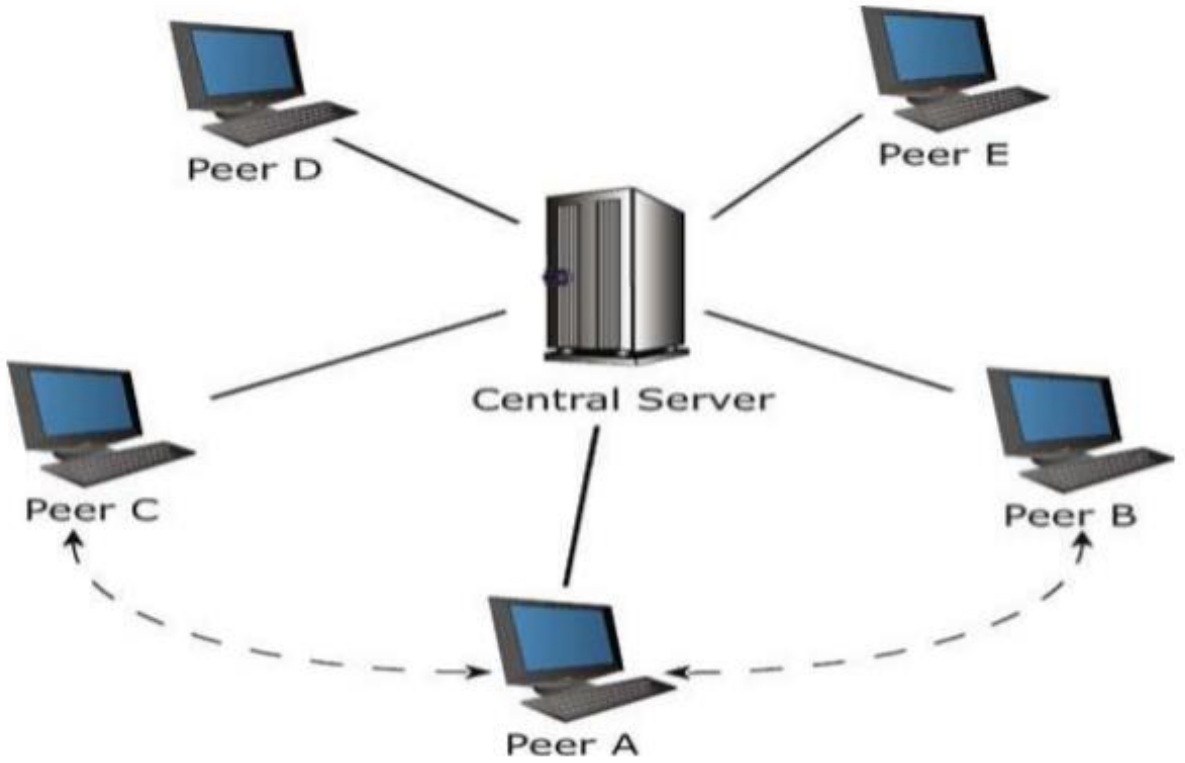
- There may be a chance of getting performance issues because of the unavailability of resources (e.g.: memory, bandwidth, processing speed) at the server system when too many requests arise.

Taxonomy of P2P System

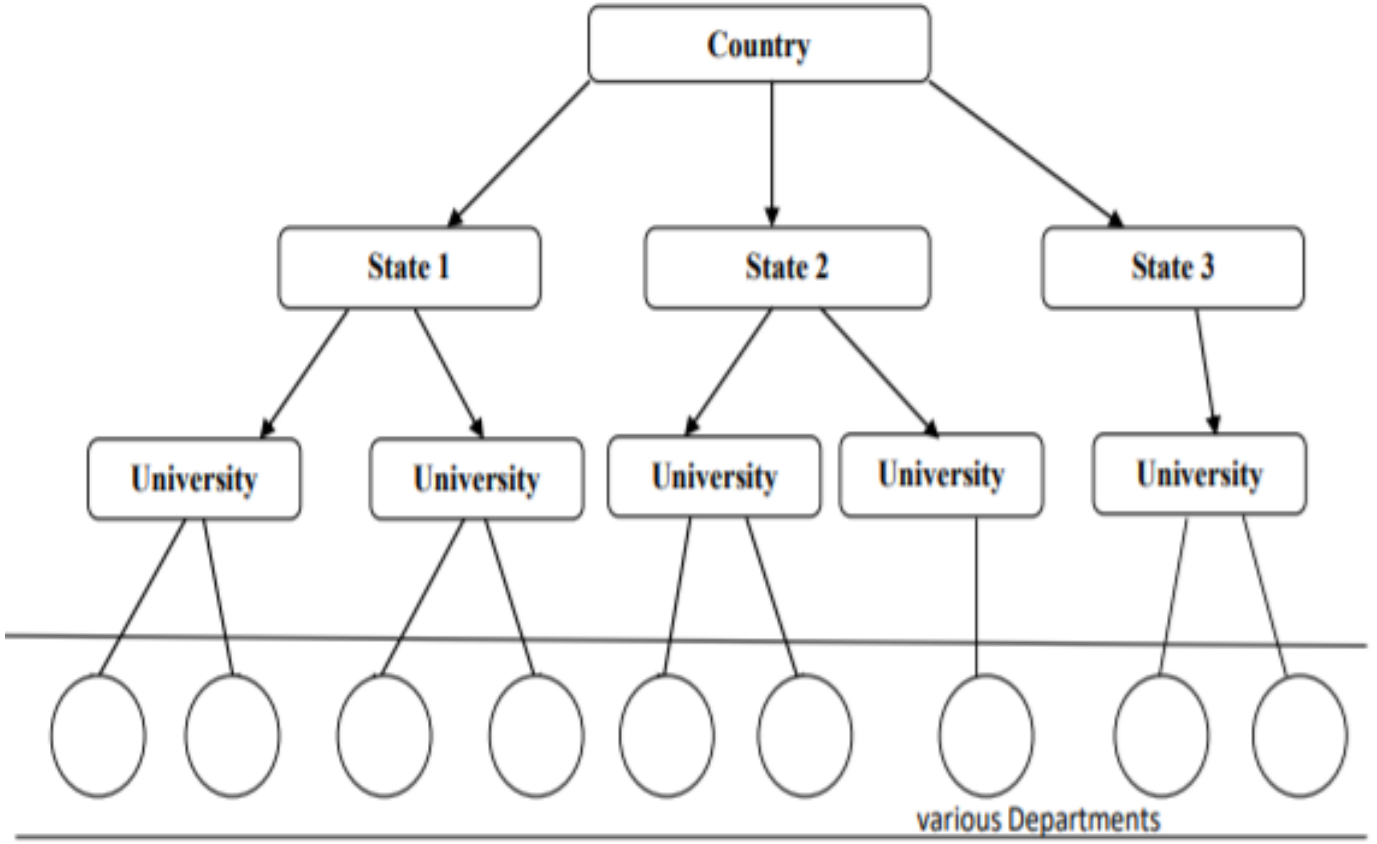


Centralized P2P System

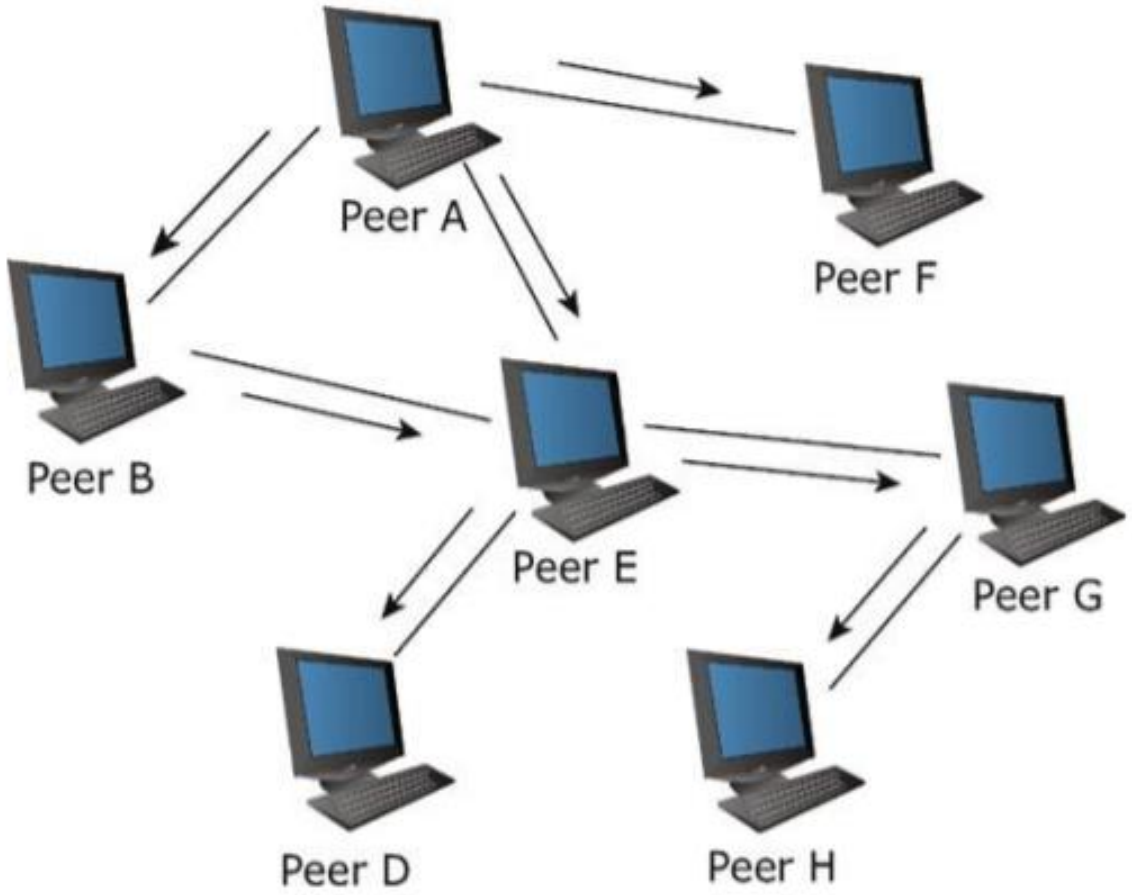
Sharing of MP3 music files



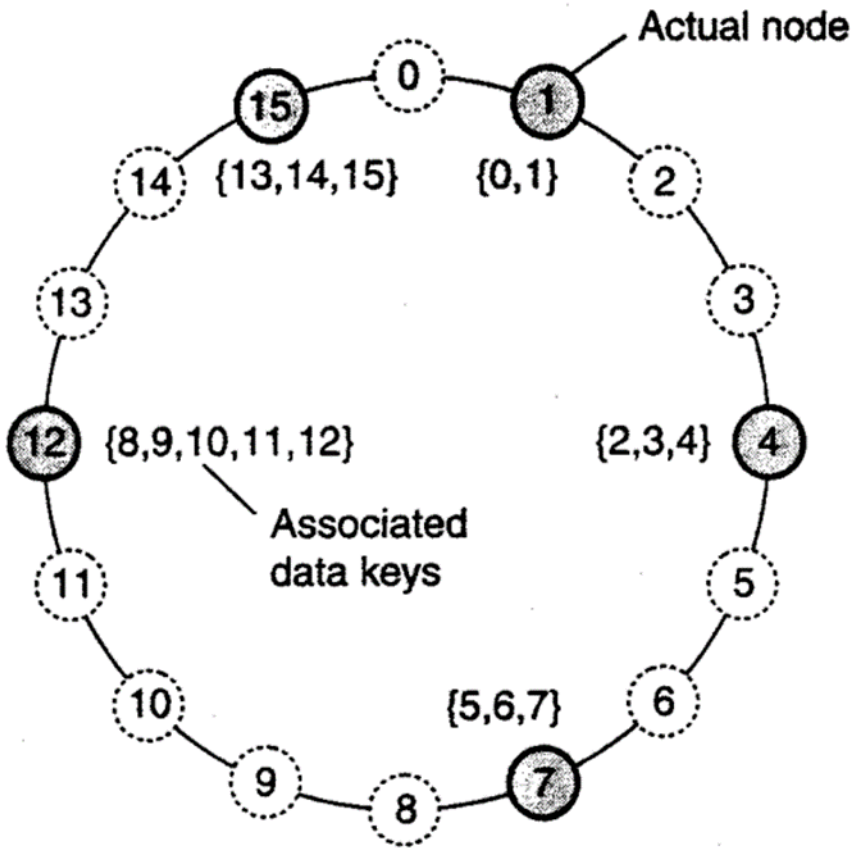
Decentralized P2P Svstem



Unstructured P2P System



Structured P2P System



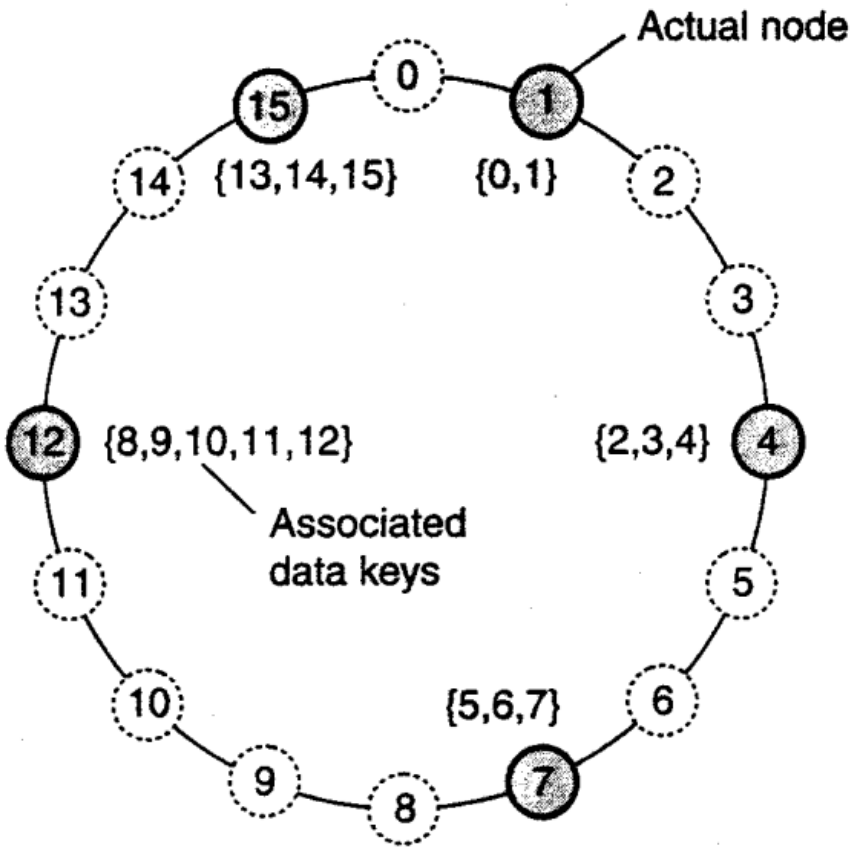
peer-to-peer systems

- From a high-level perspective, the processes that constitute a peer-to-peer system are all equal.
- This means that the functions that need to be carried out are represented by every process that constitutes the distributed system.
- As a consequence, much of the interaction between processes is symmetric: each process will act as a client and a server at the same time (which is also referred to as acting as a servent).
- Types:
  - Structured Peer-to-Peer Architectures
  - Unstructured Peer-to-Peer Architectures

Structured Peer-to-Peer Architectures

- structured → the system already has a predefined structure that other nodes will follow.
- Every structured network inherently suffers from poor scalability, due to the need for structure maintenance.
- In general, the nodes in a structured overlay network are formed in a logical ring, with nodes being connected to the this ring.
- In this ring, certain nodes are responsible for certain services.

- A common approach that can be used to tackle the coordination between nodes, is to use distributed hash tables (DHTs).
- A traditional hash function converts a unique key into a hash value, that will represent an object in the network.
- The hash function value is used to insert an object in the hash table and to retrieve it.



Unstructured P2P Systems

- There is no specific structure in these systems, hence the name "unstructured networks".
- Due to this reason, the scalability of the unstructured p2p systems is very high.
- These systems rely on randomized algorithms for constructing an overlay network.
- As in structured p2p systems, there is no specific path for a certain node. It's generally random, where every unstructured system tried to maintain a random path. Due to this reason, the search of a certain file or node is never guaranteed in unstructured systems.

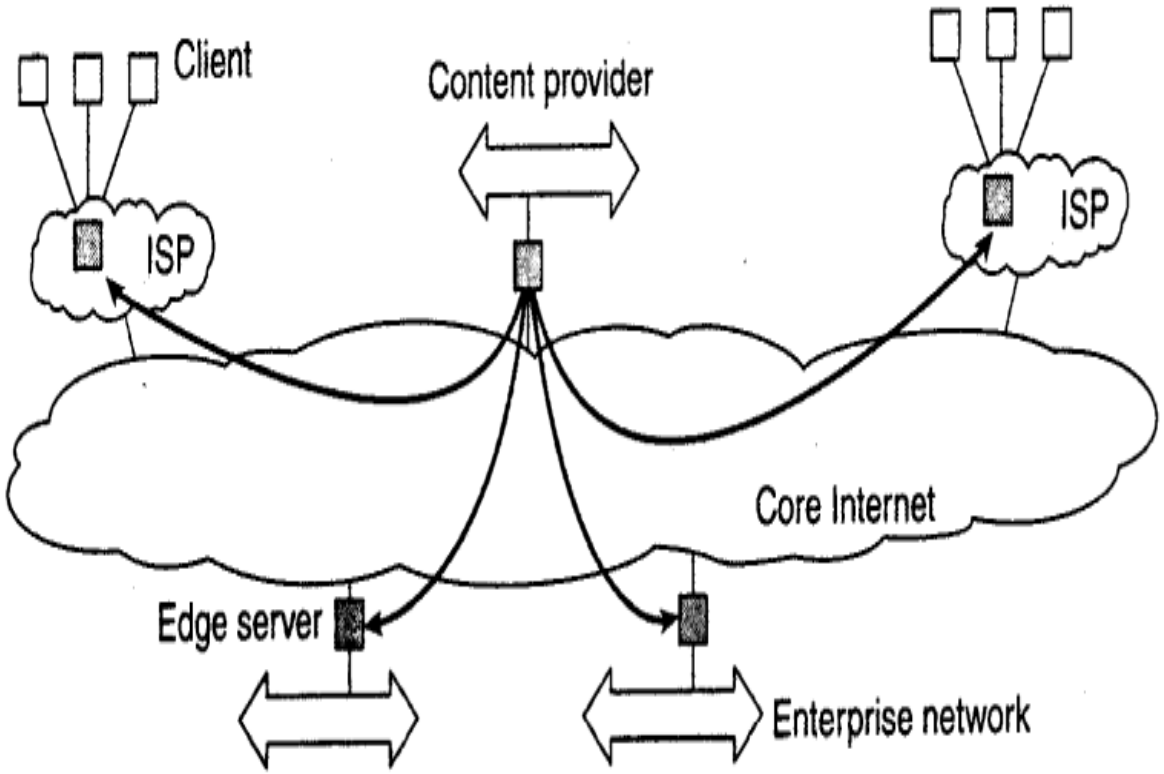
- The basic principle is that each node is required to randomly select another node, and contact it.
    - Let each peer maintain a partial view of the network, consisting of n other nodes
    - Each node P periodically selects a node Q from its partial view
    - P and Q exchange information and exchange members from their respective partial views

Hybrid Architectures

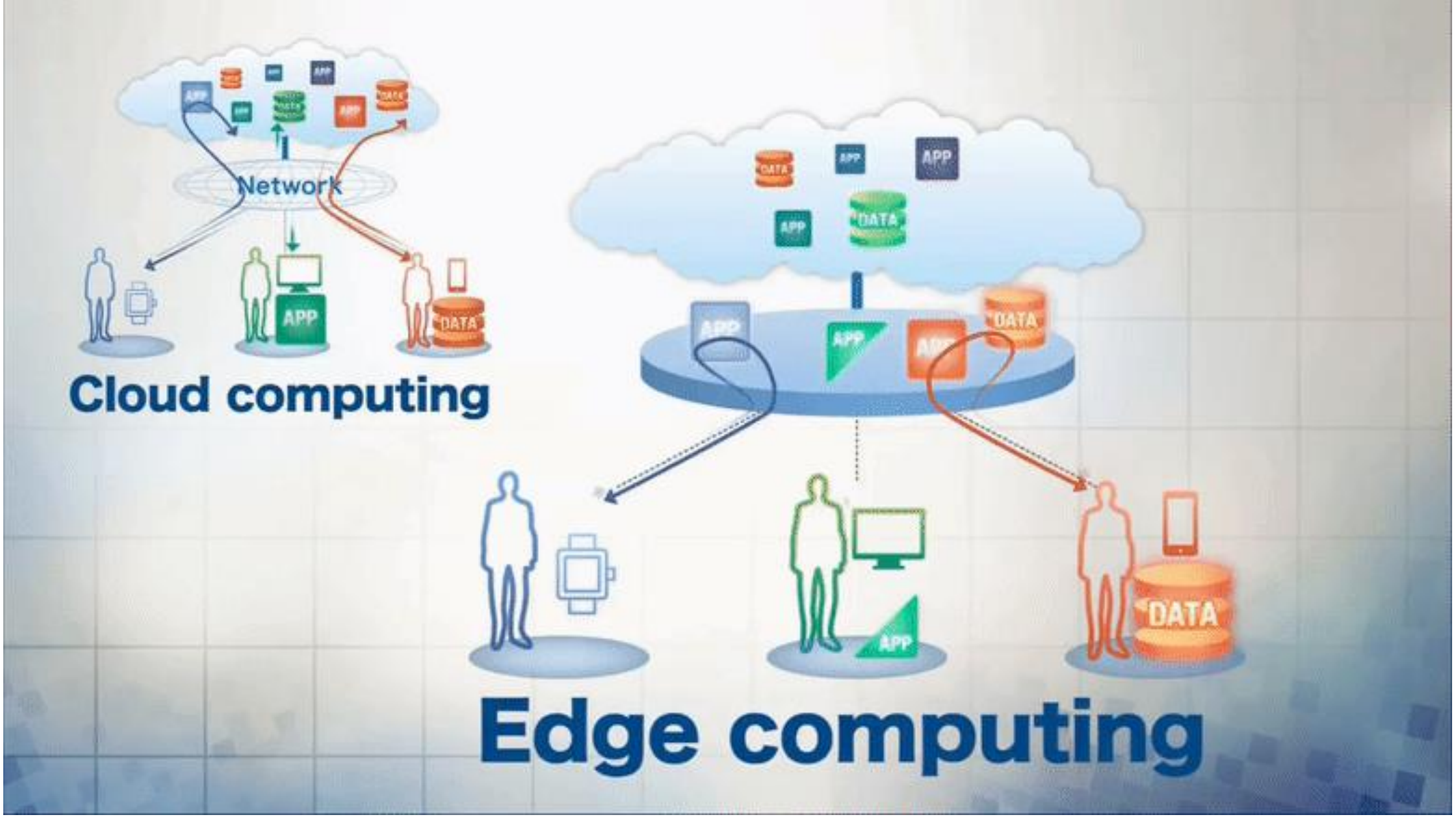
- In this section we take a look at some specific classes of distributed systems in which client-server solutions are combined with decentralized architectures
- Some of these architectures are as follows:
  - Edge-Server Systems
  - Collaborative Distributed Systems

Edge-Server Systems

- An important class of distributed systems that is organized according to a hybrid architecture is formed by edge-server systems.
- These systems are deployed on the Internet where servers are placed "at the edge" of the network. This edge is formed by the boundary between enterprise networks and the actual Internet (ISP).
- EG. Likewise, where end users at home connect to the Internet through their ISP, the ISP can be considered as residing at the edge of the Internet.



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


Network
APP
DATA
Cloud computing
APP
DATA
APP
APP
APP
DATA
APP
DATA
Edge computing

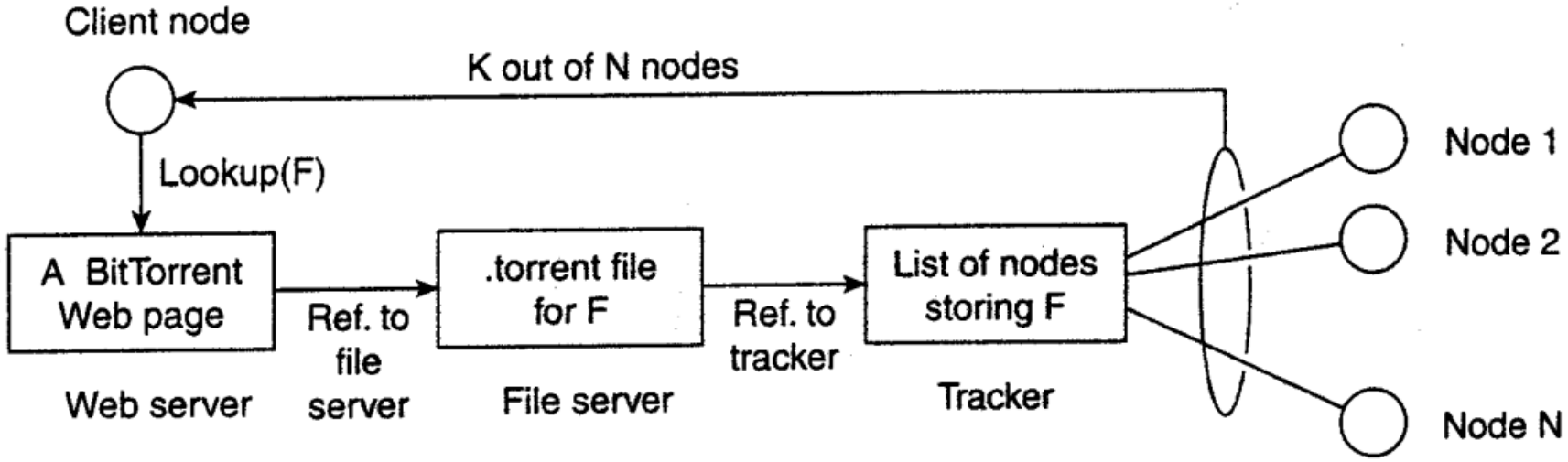
- End users, or clients in general, connect to the Internet by means of an edge server.
- The edge server's main purpose is to serve content, possibly after applying filtering and transcoding functions.
- More interesting is the fact that a collection of edge servers can be used to optimize content and application distribution.
- The basic model is that for a specific organization, one edge server acts as an origin server from which all content originates.
- That server can use other edge servers for replicating Web pages and such.

Collaborative Distributed Systems

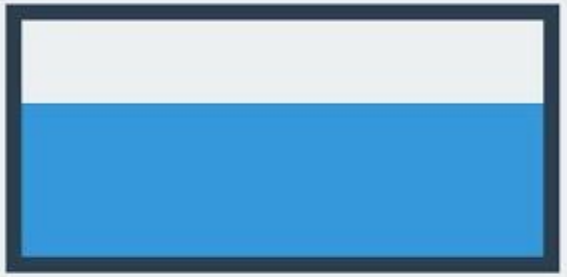
- Hybrid structures are especially deployed in collaborative distributed systems.
- The main issue in many of these systems to first get started, for which often a traditional client-server scheme is deployed.
- Once a node has joined the system, it can use a fully decentralized scheme for collaboration

BitTorrent file-sharing system

- BitTorrent is a peer-to-peer file downloading system.
- The basic idea is that when an end user is looking for a file, he downloads chunks of the file from other users until the downloaded chunks can be assembled together yielding the complete file.



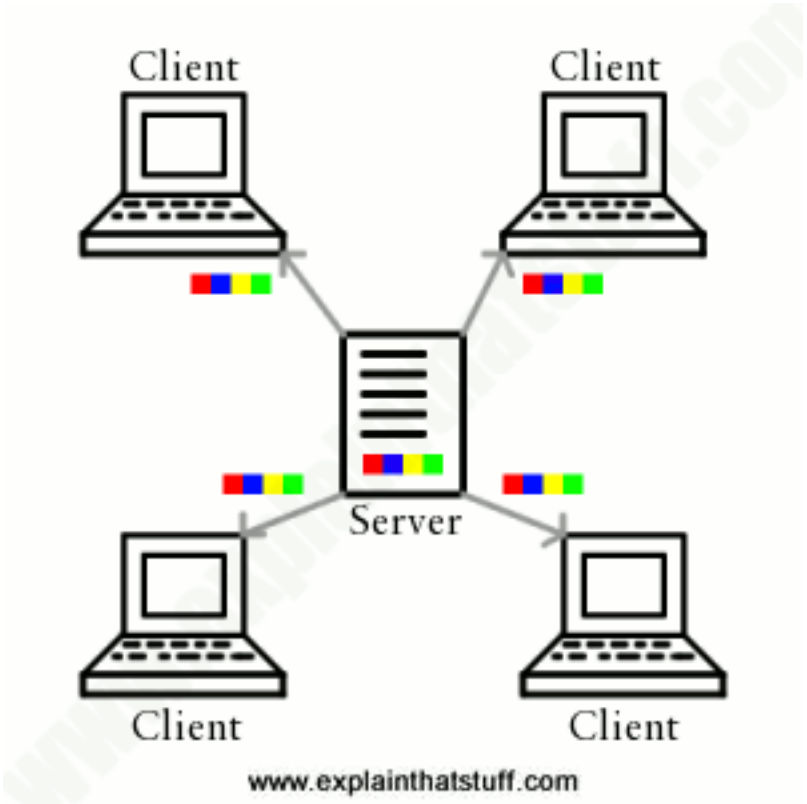
DIRECT DOWNLOAD SERVER

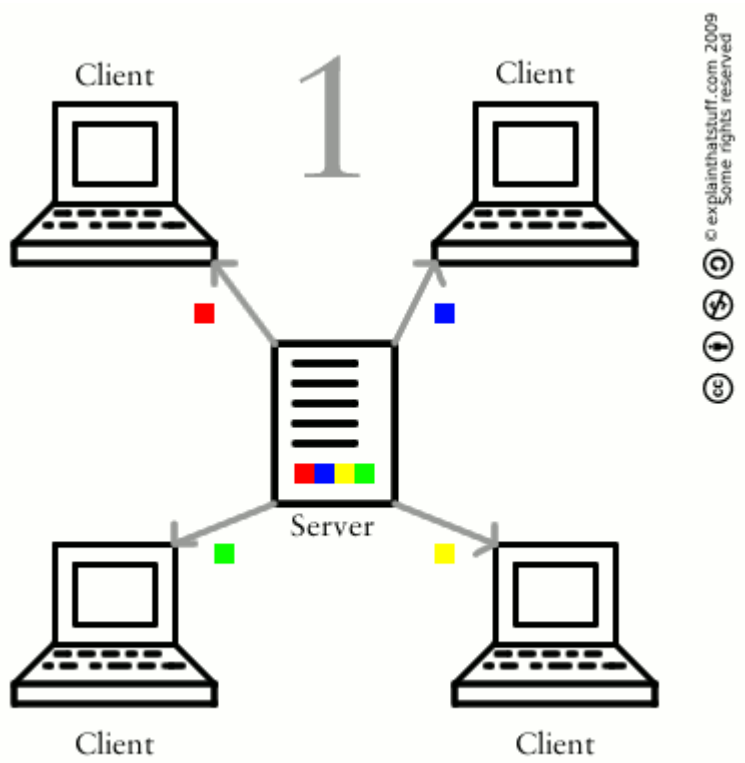


USERS

BITTORRENT PEER-TO-PEER

USERS


Client
Client
Server
Client
Client
www.explainthatstuff.com


Client
Client
Server
Client
Client
© explainthatstuff.com 2009
Some rights reserved