Distributed System

Unit:2

Unit 2. Architecture **4 Hrs.**

2.1 Architectural Styles
2.2 Middleware organization
2.3 System Architecture
2.4 Example Architectures

Distribution System Architecture

- Distribution system architectures are bundled up with components and connectors.
- Components can be individual nodes or important components in the architecture whereas connectors are the ones that connect each of these components.

  - Component: A modular unit with well-defined interfaces; replaceable, reusable.

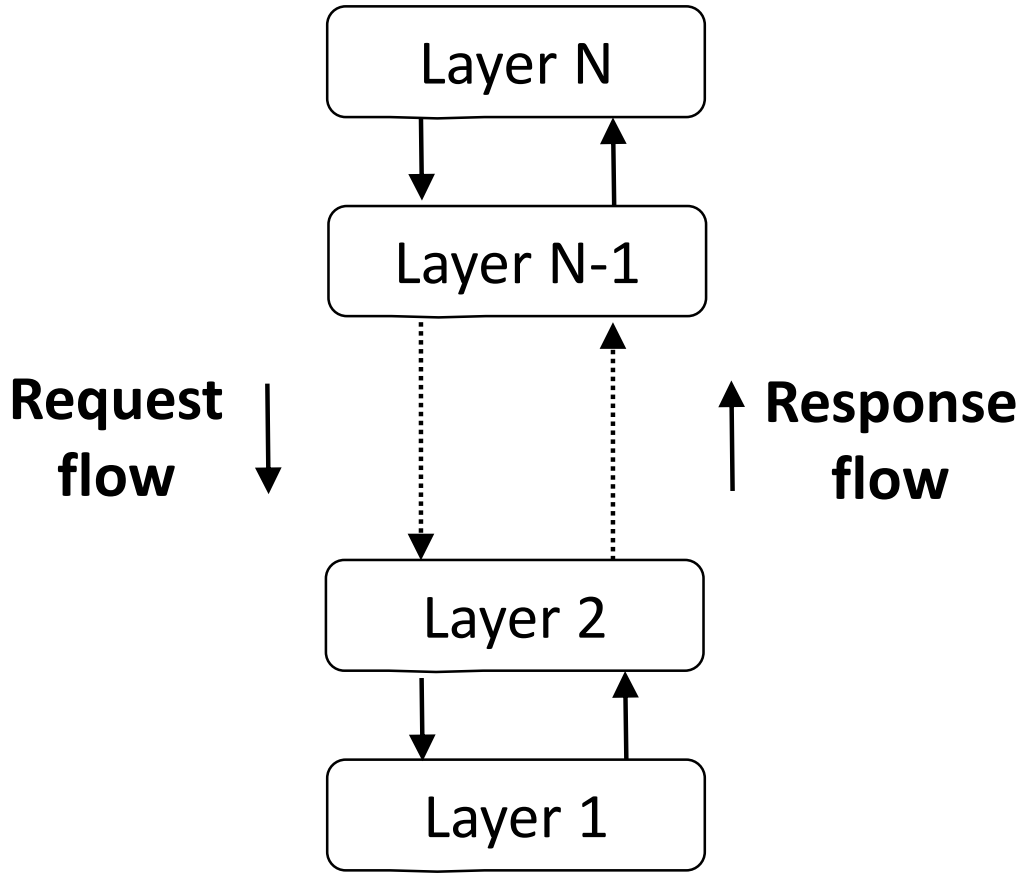
  - Connector: A communication link between modules which mediates coordination or cooperation among component.

Architectural Styles

- Distributed systems are often complex pieces of software of which the components are by definition dispersed across multiple machines.
- To master their complexity, it is crucial that these systems are properly organized.
- There are different ways on how to view the organization of a distributed system, but an obvious one is to make a distinction between the logical organization of the collection of software components and on the other hand the actual physical realization.
- Important styles of architecture for distributed systems:
  - Layered architectures
  - Object-based architectures
  - Data-centered architectures
  - Event-based architectures

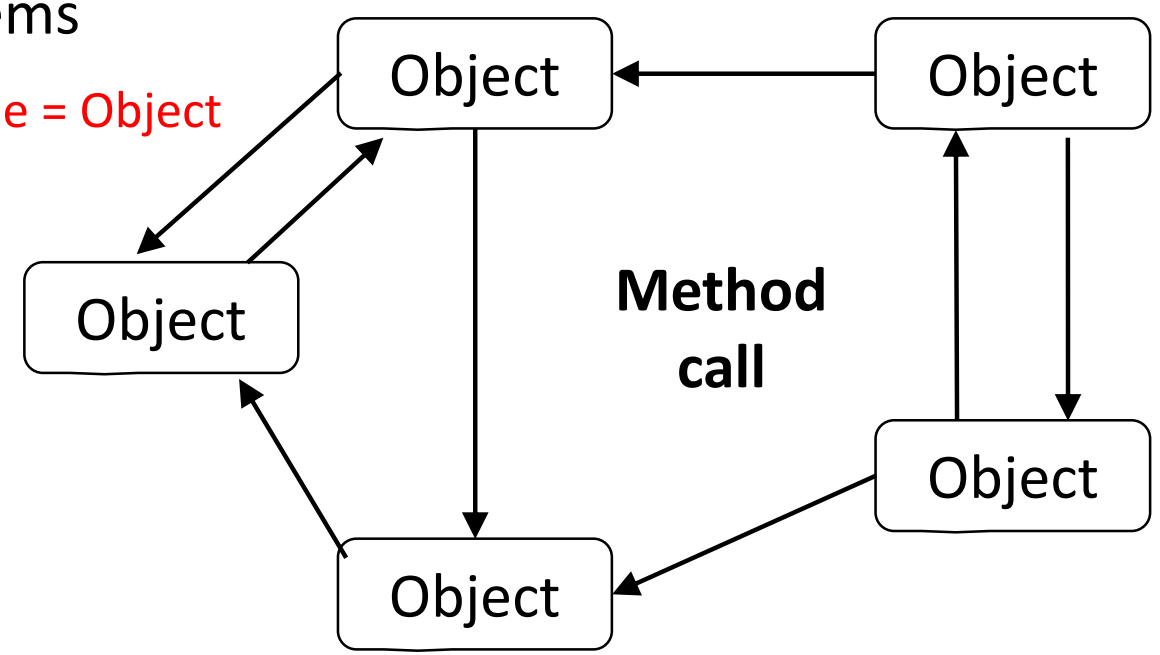
Layered architectures

- The Components are organized in a layered fashion where a component at layer Li is allowed to call components at the underlying layer Li-1, but not the other way around,
- This model has been widely adopted by the networking community
- An key observation is that control generally flows from layer to layer; requests go down the hierarchy whereas the results flow upward.



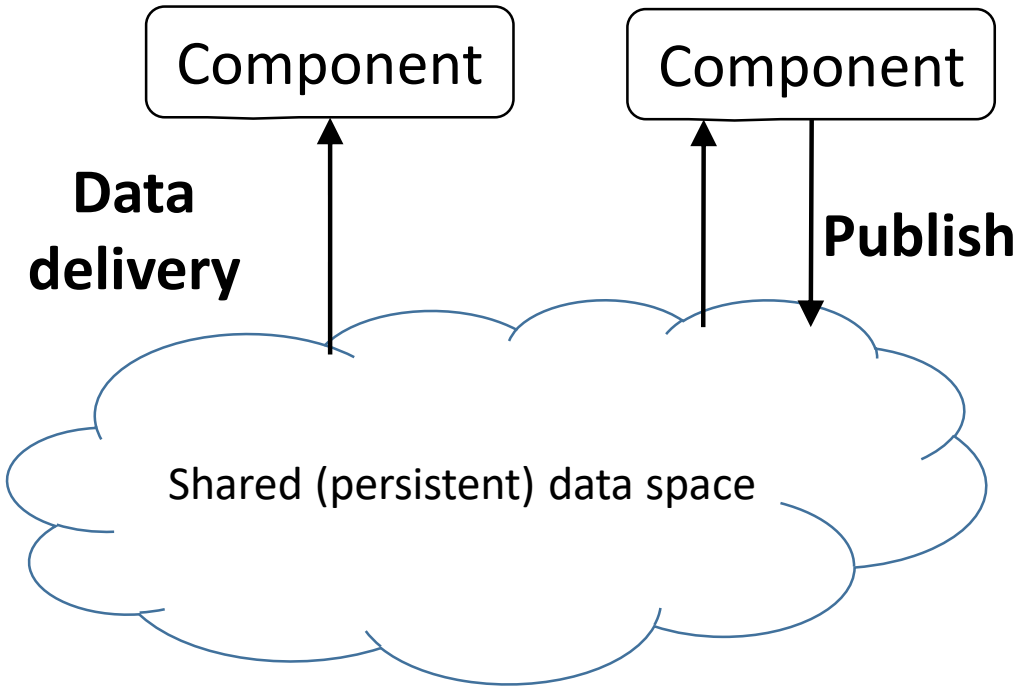
Object-based architectures

- Loosely coupled *(Own Private Memory)* arrangements of objects.
- Each object corresponds a component.
- Components are connected through a (remote) procedure call (RPC) mechanism.
- The layered and object-based architectures still form the most important styles for large software systems
- Component, Node = Object
- Connector = RPC



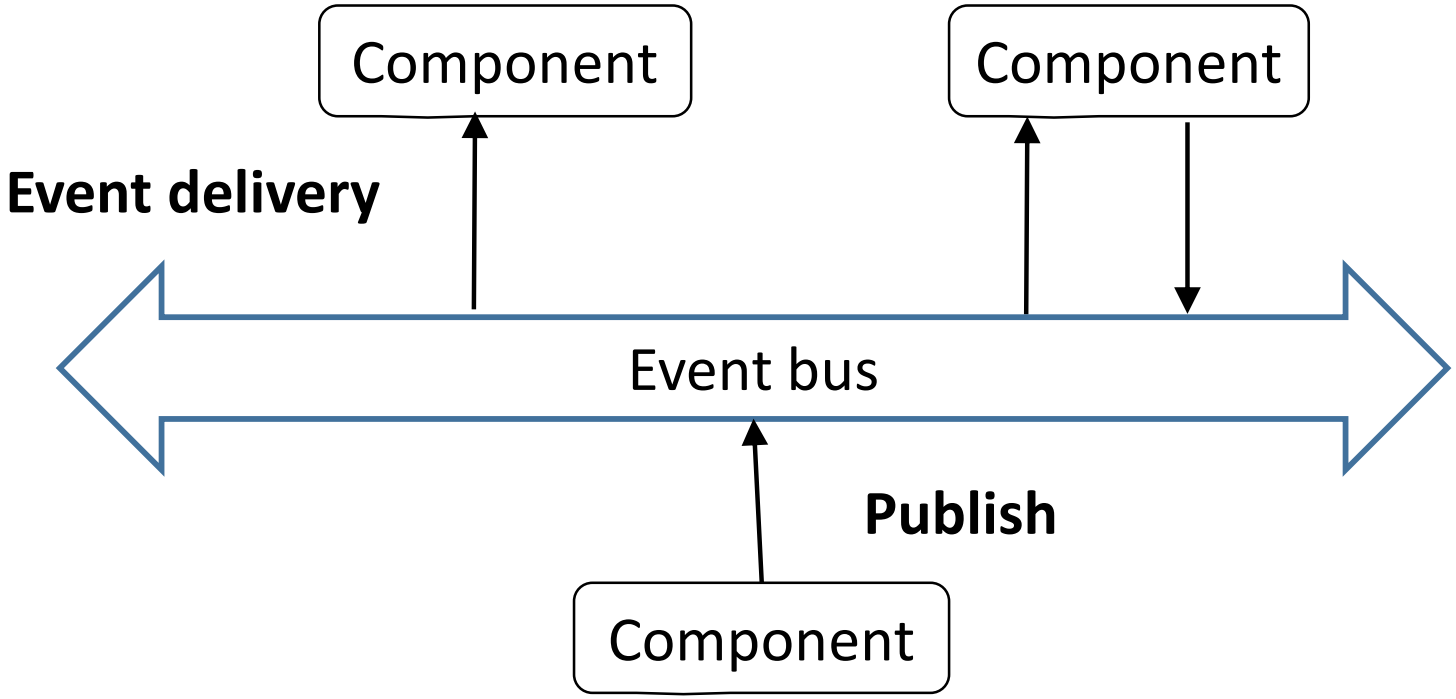
Data-centered architectures

- It evolve around the idea that processes communicate through a common (passive or active) repository.
- Service Provider and Service User
- It can be argued that for distributed systems these architectures are as important as the layered and object-based architectures
- For example,
  - A wealth of networked applications have been developed that rely on a shared distributed file system in which virtually all communication takes place through files.
  - Web-based distributed systems are largely data-centric: processes communicate through the use of shared Web-based data services.

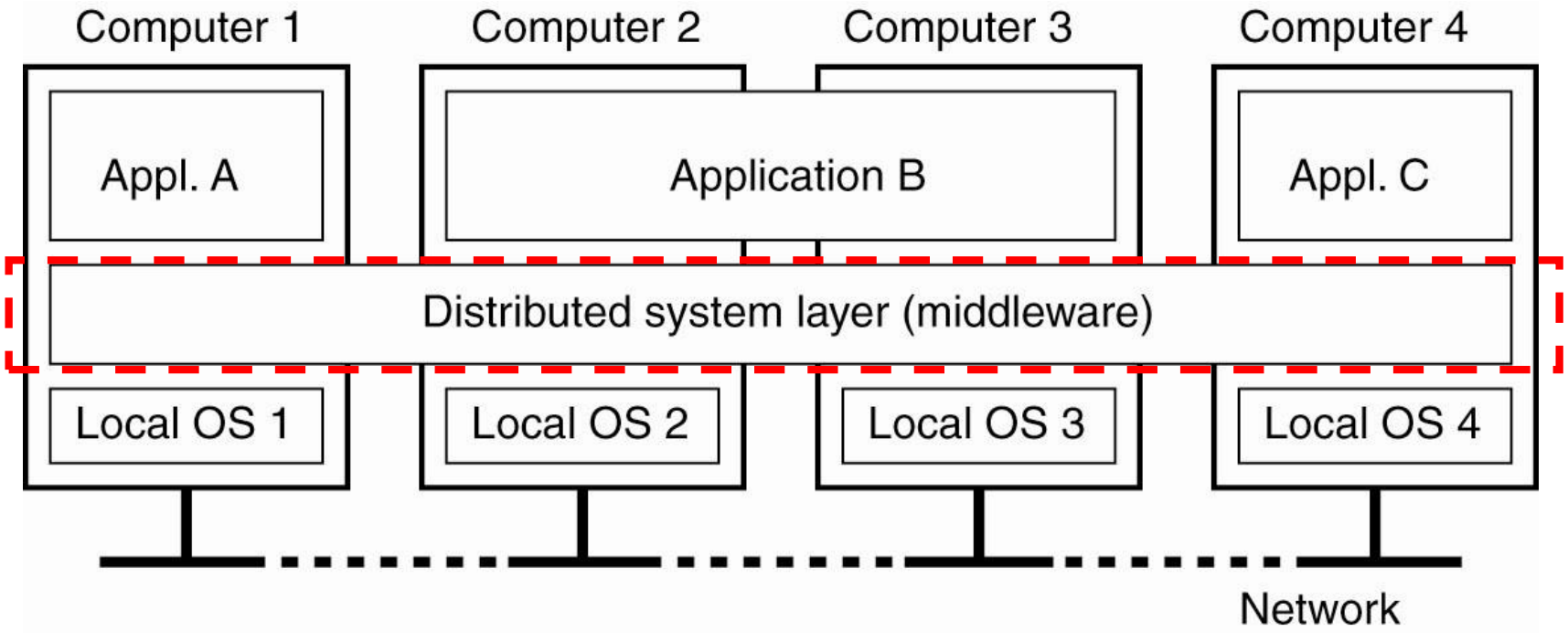


Event-based architectures

- Processes communicate through the propagation of events.
- event producers and event consumers
- For distributed systems, event propagation has generally been associated with what are known as publish/subscribe systems.
- **Advantage** - processes are loosely coupled. In principle, they need not explicitly refer to each other. This is also referred to as being decoupled in space, or referentially decoupled.



Middleware (MW)



- A distributed system organized as Middleware.
- The middleware layer runs on all machines, and offers a uniform interface to the system.
- Middleware is software which lies between an operating system and the applications running on it.

Middleware (MW)

- Software that manages and supports the different components of a distributed system. In essence, it sits in the middle of the system.
- Essentially functioning as hidden translation layer, middleware enables communication and data management for distributed applications.
- It enables multiple systems to communicate with each other across different platforms.
- Examples:
  - Transaction processing monitors
  - Data converters
  - Communication controllers

Role of Middleware (MW)

- **In some early systems:**
  - Middleware 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.
  - Middleware also supports seamless access to remote services, doesn't try to look like a general-purpose OS.

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

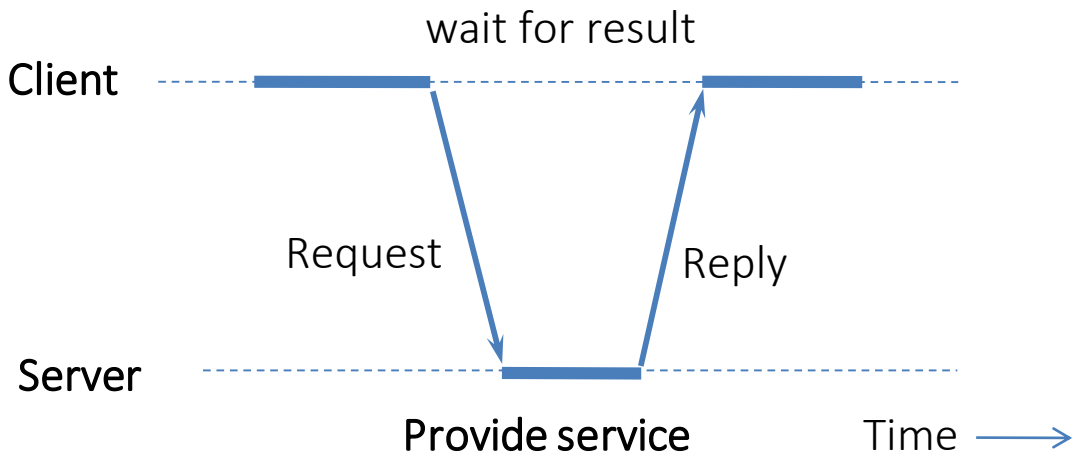
System Architecture

There are three views toward system architectures:

- Centralized Architectures
- Decentralized architectures
- Hybrid Architectures

Centralized Architectures

- Manage distributed system complexity – think in terms of clients that request services from servers.
- Processes are divided into two groups:
  1. A server is a process implementing a specific service, for example, a file system service or a database service.
  2. A client is a process that requests a service from a server by sending it a request and subsequently waiting for the server's reply.



Centralized Architectures

- **Communication** - implemented using a connectionless protocol when the network is reliable e.g. local-area networks.
  1. Client requests a service – packages and sends a message for the server, identifying the service it wants, along with the necessary input data.
  2. The Server will always wait for an incoming request, process it, and package the results in a reply message that is then sent to the client.

Connectionless protocol

- Describes communication between two network end points in which a message can be sent from one end point to another without prior arrangement.
- Device at one end of the communication transmits data to the other, without first ensuring that the recipient is available and ready to receive the data.
- The device sending a message sends it addressed to the intended recipient.

Application Layering

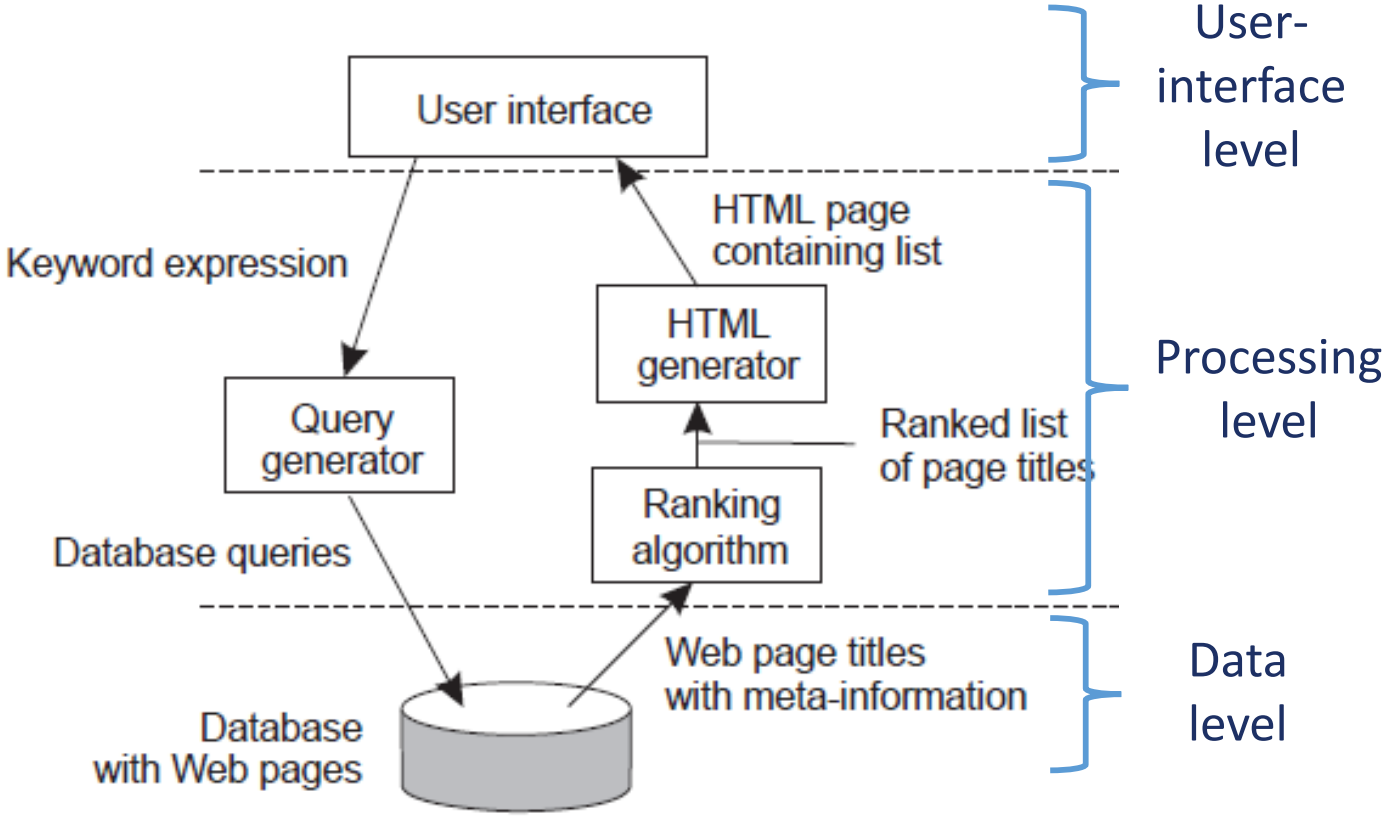
Traditional three-layered view:

1. **The user-interface level**
   - It contains units for an application's user interface
   - Clients typically implement the user-interface level
     - Simplest user-interface program - character-based screen
     - Simple GUI
2. **The processing level**
   - It contains the functions of an application, i.e. without specific data
   - Middle part of hierarchy -> logically placed at the processing level
3. **The data level**
   - It contains the data that a client wants to manipulate through the application components
   - manages the actual data that is being acted on

Internet search engine- An example of Application Layering

Traditional three-layered view:

- **User-interface level:** a user types in a string of keywords and is subsequently presented with a list of titles of Web pages.
- **Data Level:** huge database of Web pages that have been prefetched and indexed.
- **Processing level:** search engine that transforms the user's string of keywords into one or more database queries.
  - Ranks the results into a list
  - Transforms that list into a series of HTML pages

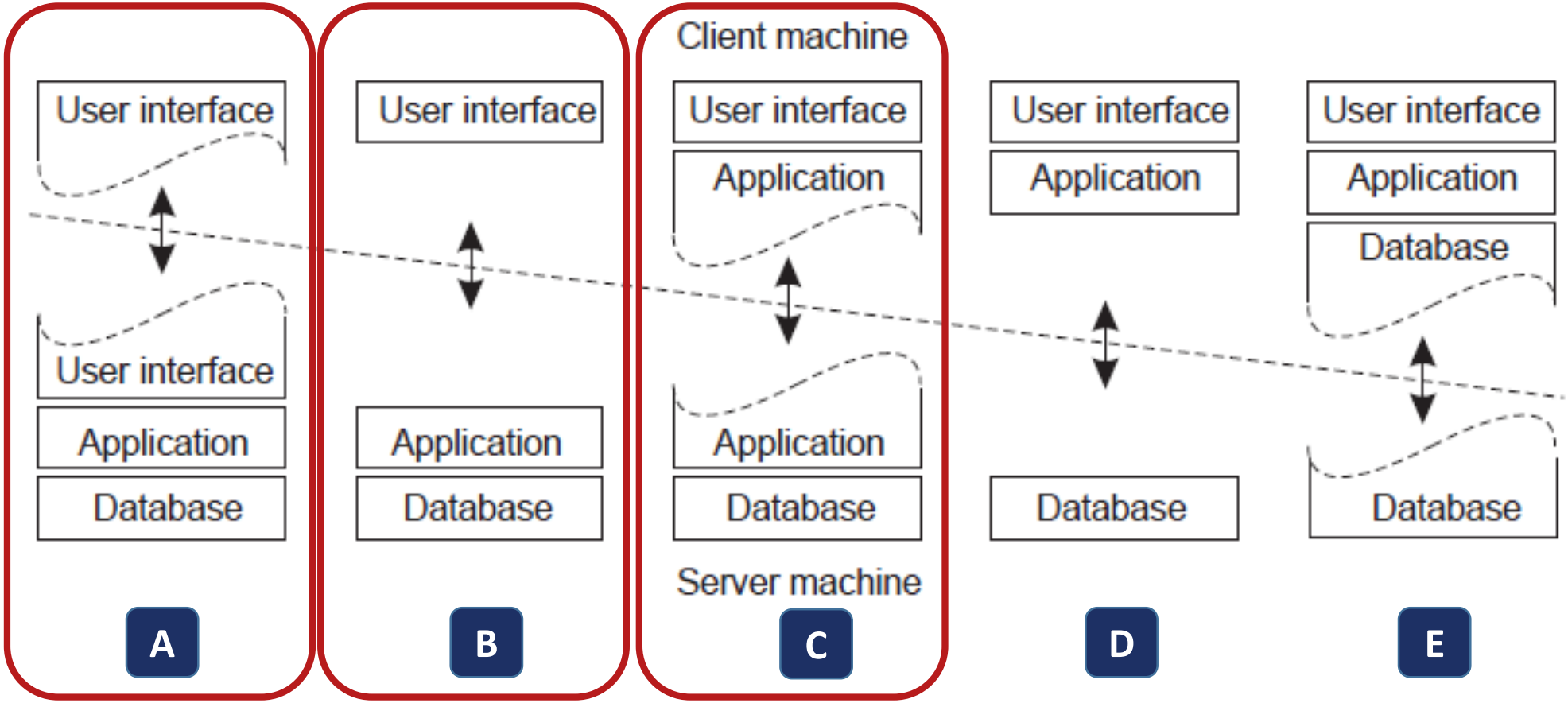


Multi-Tiered Architectures

- Simplest organization - 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

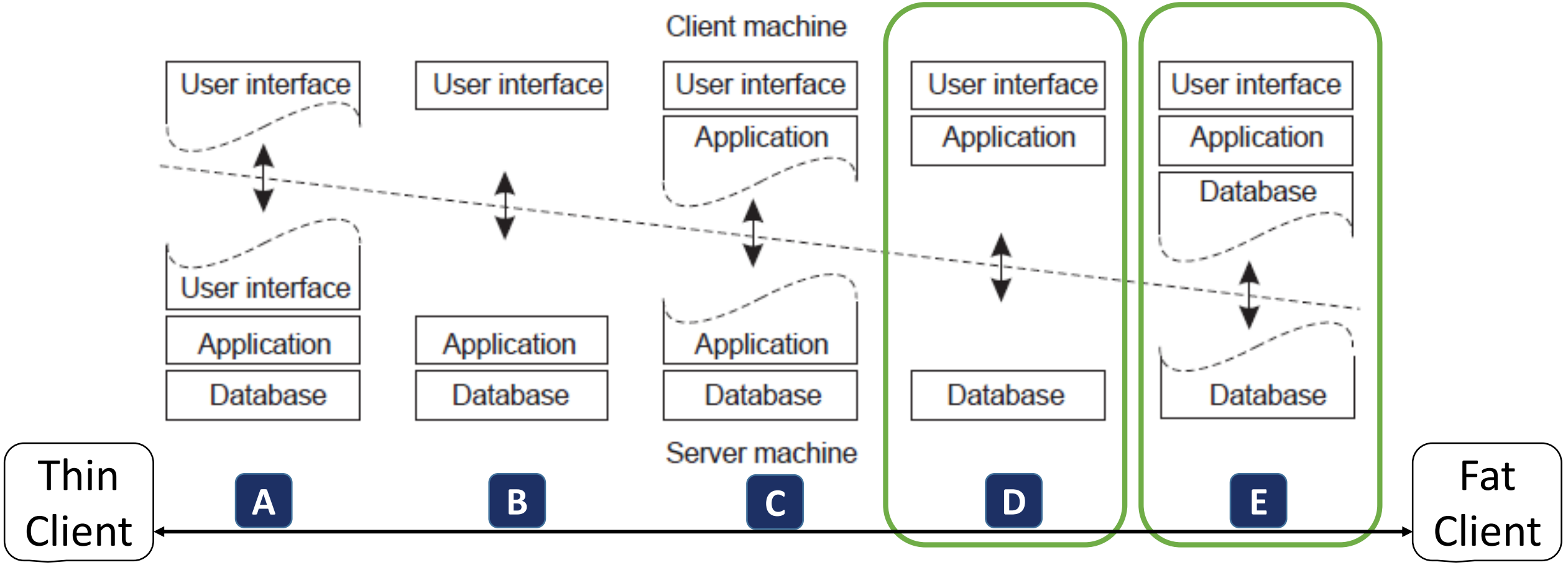
- Single-tiered: dumb terminal/mainframe configuration
- Two-tiered: client/single server configuration
- Three-tiered: each layer on separate machine

Two-tiered Architectures - Thin-client model and fat-client model



- Case-A: Only the terminal-dependent part of the user interface
- Case- B: Place the entire user-interface software on the client side
- Case- C: Move part of the application to the front end

Two-tiered Architectures - Thin-client model and fat-client model



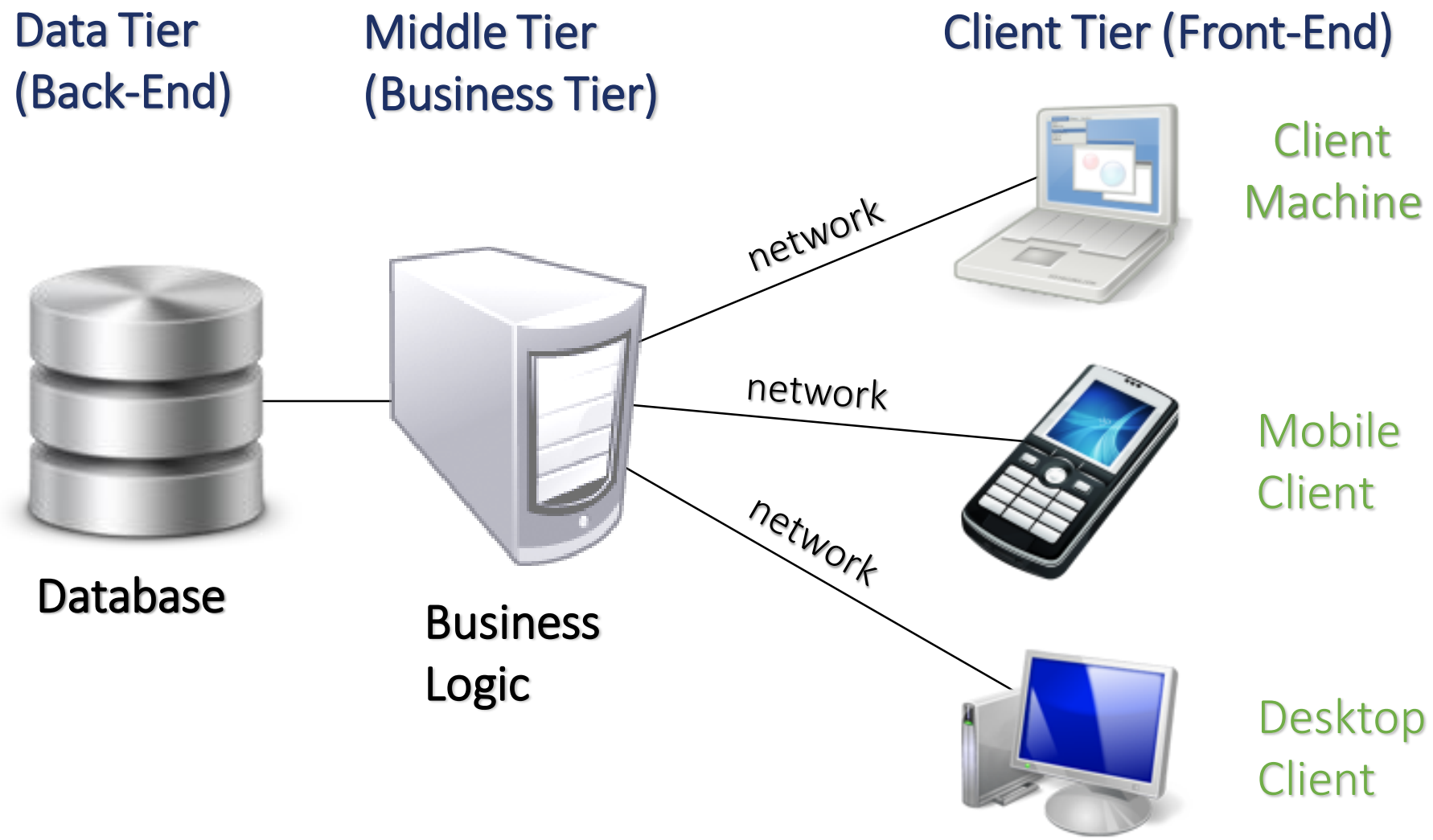
- Case-D: Used where the client machine is a PC or workstation, connected through a network to a distributed file system or database
- Case- E: Used where the client machine is a PC or workstation, connected through a network to a distributed file system or database

- **Case A**: 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.
- **Case B**: 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.
- **Case C**: Continuing along this line of reasoning, we may also move part of the application to the front end. An example where this makes sense is where 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.

- Another example of the organization is that of 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.
- **Case D**: In many client-server environments, the organizations shown in Case D & E are particularly popular. These organizations are 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. For 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.
- **Case E**: This 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.

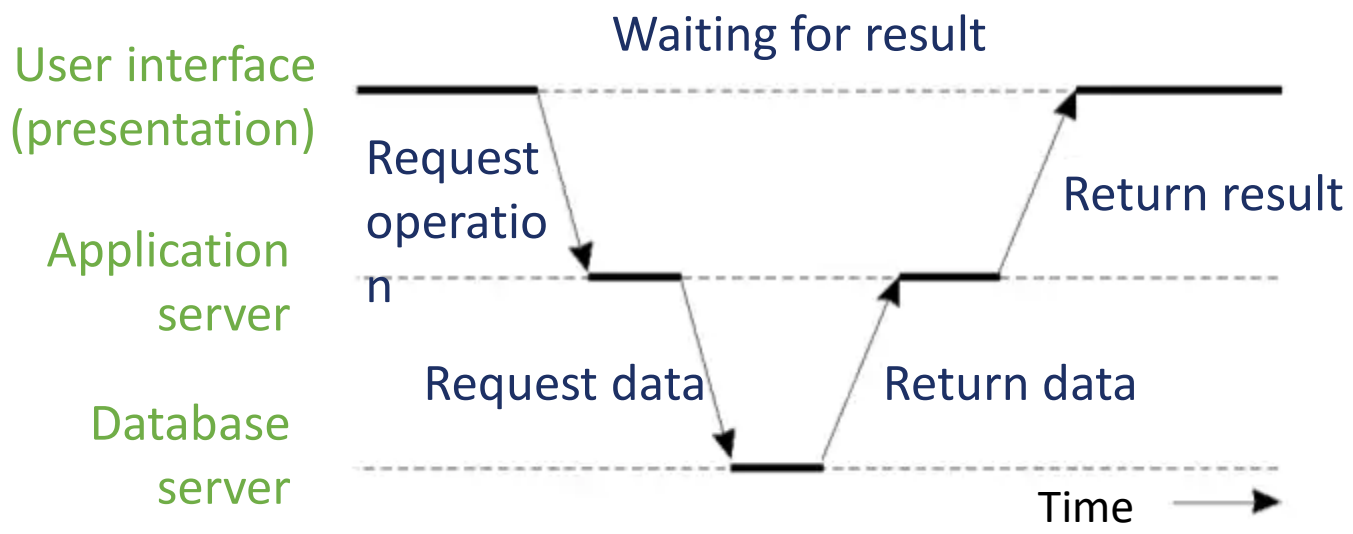
- We note that for a few years there has been a strong trend to move away from the configurations shown in Case D and Case E in those case that client software is placed at end-user machines. In these cases, most of the processing and data storage is handled at the server side. The reason for this is 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).

Multitiered Architectures (3-Tier Architecture)



Multitiered Architectures (3-Tier Architecture)

- The server tier in two-tiered architecture becomes more and more distributed
- Distributed transaction processing
  - A single server is no longer adequate for modern information systems
- This leads to three-tiered architecture
  - Server may acting as a client



An example of a server acting as client

An example of a server acting as client

- 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.
- Example: Three-tiered architecture - organization of Web sites.
  - Web server acts as an entry point to a site, passing requests to an application server where the actual processing takes place.
  - Application server interacts with a database server.

Decentralized Architectures

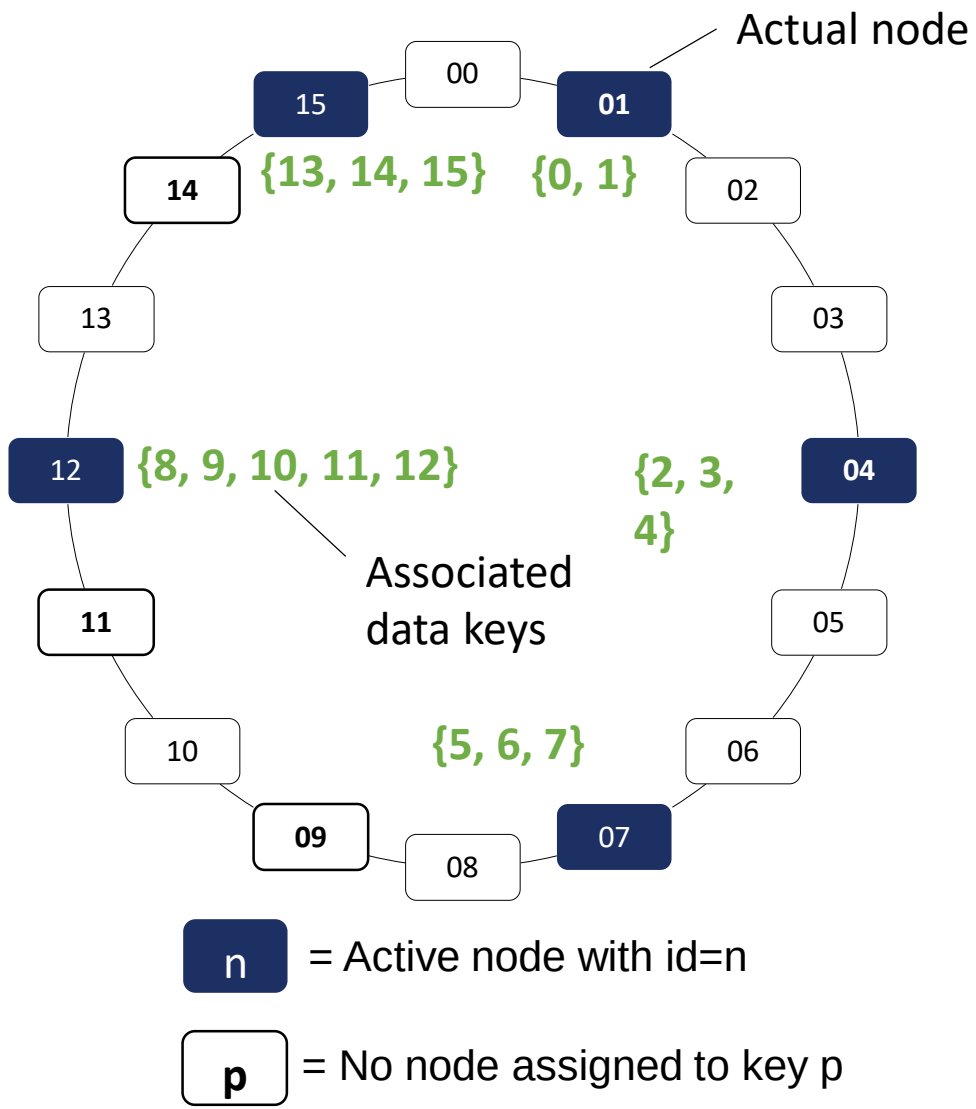
- Multi-tiered architectures can be considered as vertical distribution
  - Placing logically different components on different machines
- An alternative is horizontal distribution (peer-to-peer systems)
  - A collection of logically equivalent parts
  - Each part operates on its own share of the complete data set, balancing the load
- Peer-to-peer architectures - how to organize the processes in an overlay network in which the nodes are formed by the processes and the links represent the possible communication channels (which are usually realized as TCP connections).
- Decentralized Architectures Types
  1. Structured P2P
  2. Unstructured P2P
  3. Hybrid P2P

Structured P2P Architectures

- There are links between any two nodes that know each other
- Structured: the overlay network is constructed in a deterministic procedure
  - Most popular: distributed hash table (DHT)
- DHT-based system
  - Data items are assigned a random key from a large identifier space, such as a 128-bit or 160-bit identifier.
  - Nodes are assigned a random number from the same identifier space.
- Some well known structured P2P networks are Chord, Pastry, Tapestry, CAN, and Tulip.

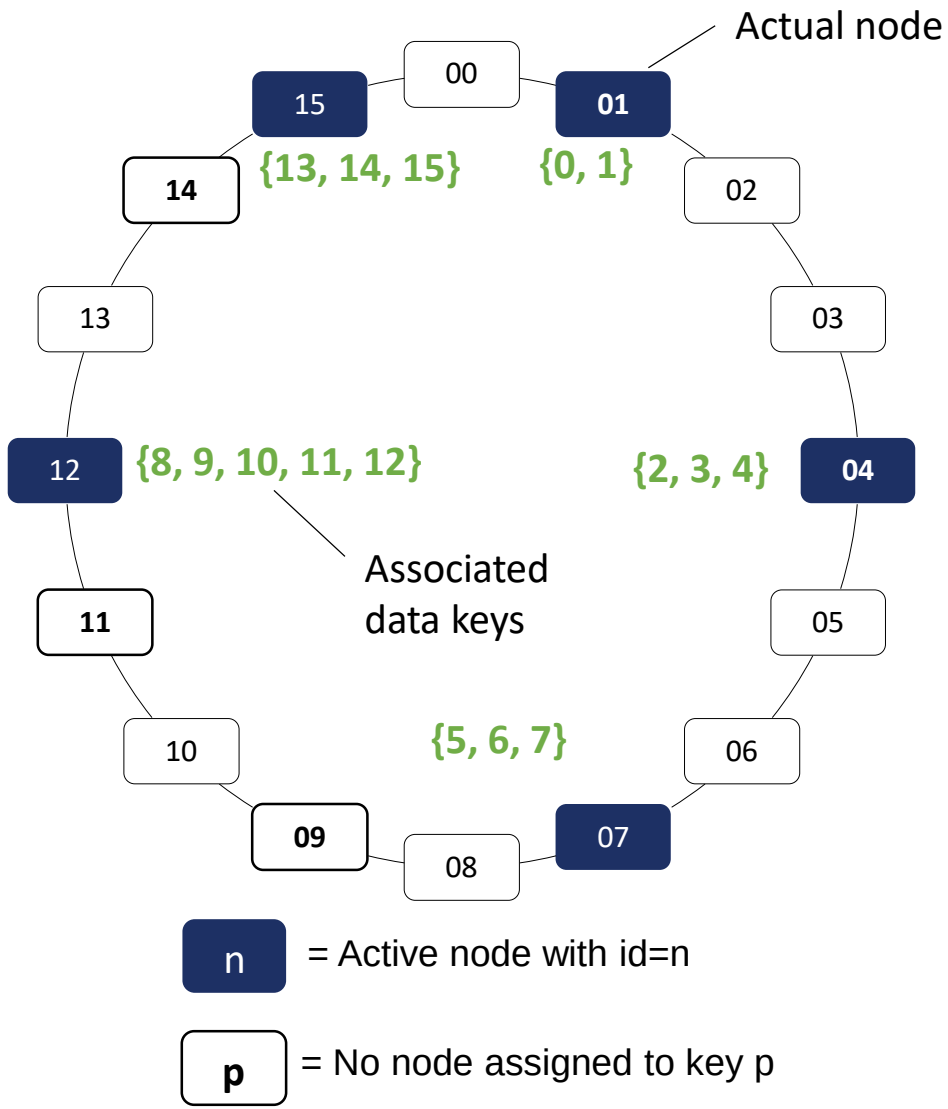
Chord

- Each node in the network knows the location of some fraction of other nodes.
  - If the desired key is stored at one of these nodes, ask for it directly
  - Otherwise, ask one of the nodes you know to look in *its* set of known nodes.
  - The request will propagate through the overlay network until the desired key is located
  - Lookup time is *O(log(N))*



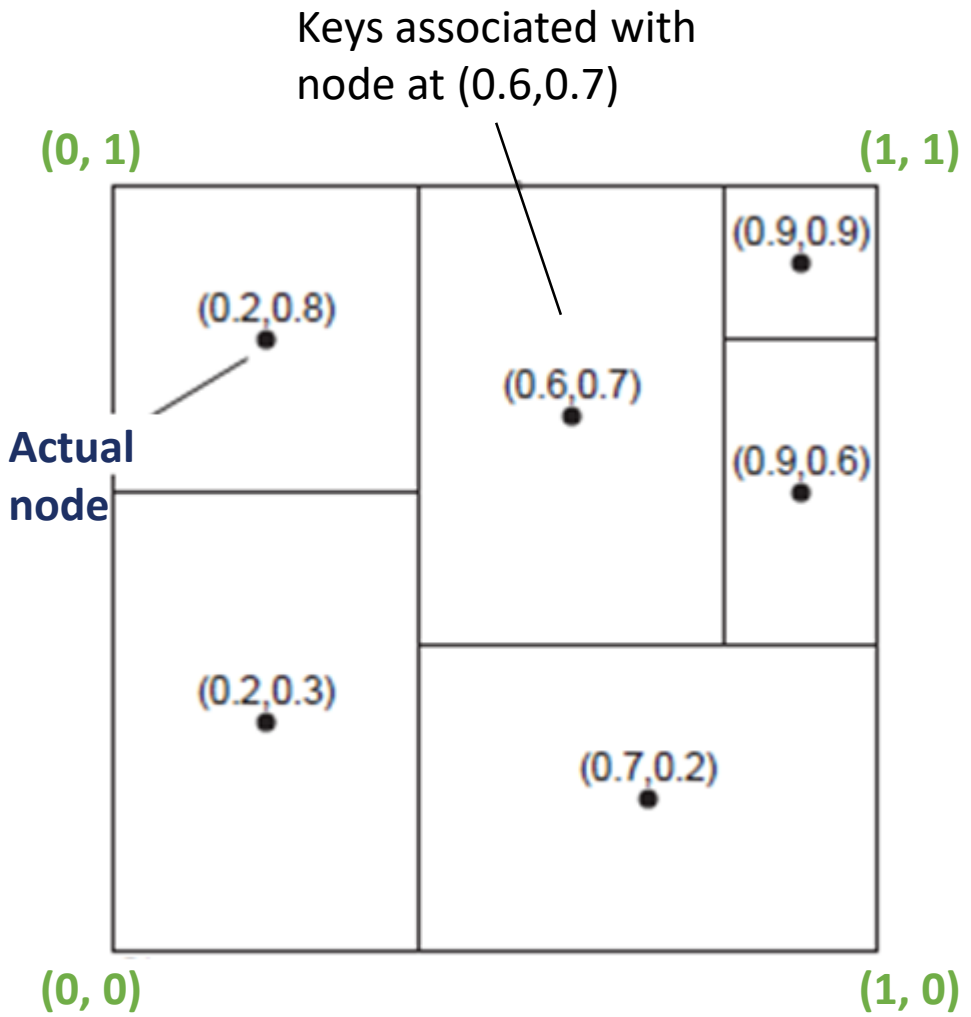
Chord

- **Join**
  - Generate the node's random identifier, id, using the distributed hash function
  - Use the lookup function to locate *succ(id)*
  - Contact *succ(id)* and its predecessor to insert self into ring.
  - Assume data items from *succ(id)*
- **Leave (normally)**
  - Notify predecessor & successor;
  - Shift data to *succ(id)*
- **Leave (due to failure)**
  - Periodically, nodes can run "self-healing" algorithms



Content Addressable Network (CAN)

- CAN deploys a d-dimensional Cartesian coordinate space, which is completely partitioned among all the nodes that participate in the system.
- Example: 2-dimensional case
  - Two-dimensional space [0,1]x[0,1] is divided among six nodes
  - Each node has an associated region
  - Every data item in CAN will be assigned a unique point in this space, after which it is also clear which node is responsible for that data



Content Addressable Network (CAN)

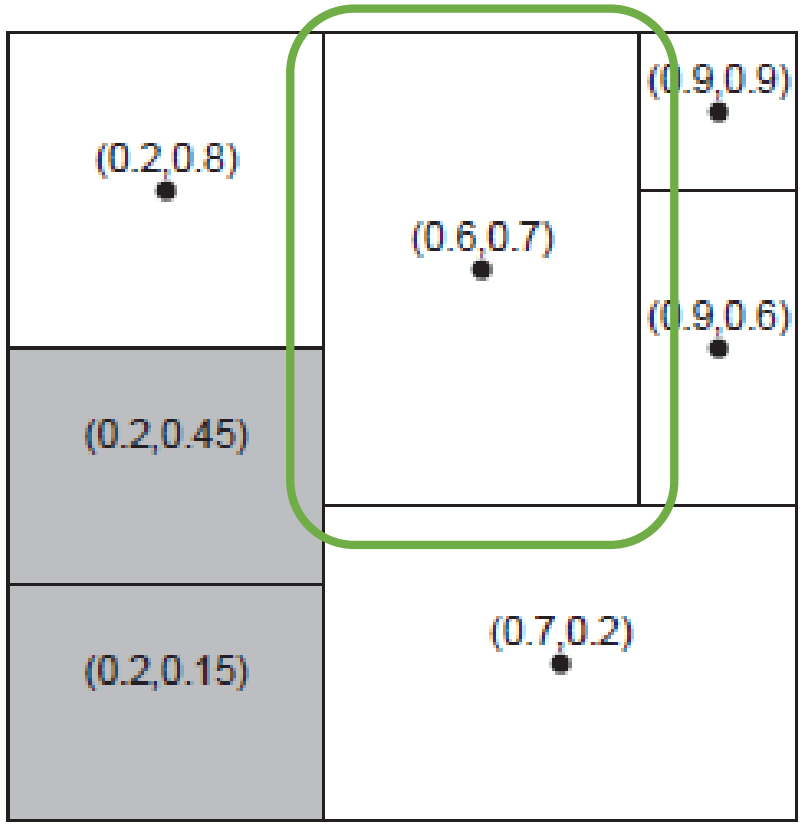
Joining CAN

- When a node P wants to join a CAN system, it picks an arbitrary point from the coordinate space and subsequently looks up the node Q in whose region that point falls.
- Node Q then splits its region into two halves and one half is assigned to the node P.
- Nodes keep track of their neighbors, that is, nodes responsible for adjacent region.
- When splitting a region, the joining node P can easily come to know who its new neighbors are by asking node Q.
- As in Chord, the data items for which node P is now responsible are transferred from node Q.

Content Addressable Network (CAN)

Leaving CAN

- Assume that the node with coordinate (0.6,0.7) leaves.
- Its region will be assigned to one of its neighbors, say the node at (0.9,0.9), but it is clear that simply merging it and obtaining a rectangle cannot be done.
- In this case, the node at (0.9,0.9) will simply take care of that region and inform the old neighbors of this fact



Overlay Networks

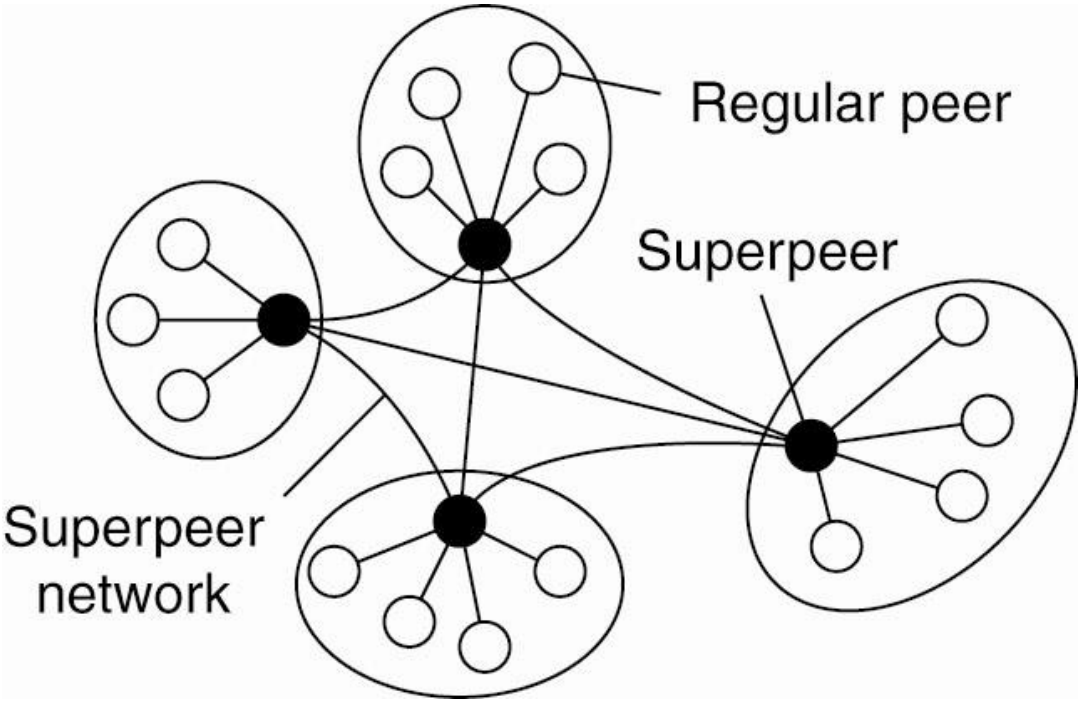
- Nodes act as both client and server; interaction is symmetric
- Each node acts as a server for part of the total system data
- Overlay networks connect nodes in the P2P system.
- An overlay network can be thought of as a computer network on top of another network. All nodes in an overlay network are connected with one another by means of logical or virtual links and each of these links correspond to a path in the underlying 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.

Unstructured P2P Architectures

- Largely relying on randomized algorithm to construct the overlay network
  - Each node has a list of neighbors, which is more or less
- Many systems try to construct an overlay network that resembles a random graph
  - Each node maintains a partial view, i.e., a set of live nodes randomly chosen from the current set of nodes constructed in a random way
- An unstructured P2P network is formed when the overlay links are established arbitrarily.
- Data items are randomly mapped to some node in the system & lookup is random, unlike the structured lookup in Chord.
- Such networks can be easily constructed as a new peer that wants to join the network can copy existing links of another node and then form its own links over time.
- In an unstructured P2P network, if a peer wants to find a desired piece of data in the network, the query has to be flooded through the network in order to find as many peers as possible that share the data..

Superpeers

- Used to address the following question
  - How to find data items in unstructured P2P systems
  - Flood the network with a search query?
- An alternative is using superpeers
  - Nodes such as those maintaining an index or acting as a broker are generally referred to as superpeers
  - They hold index of info. from its associated peers (i.e. selected representative of some of the peers)



A hierarchical organization of nodes into a superpeer network

Finding Data Items

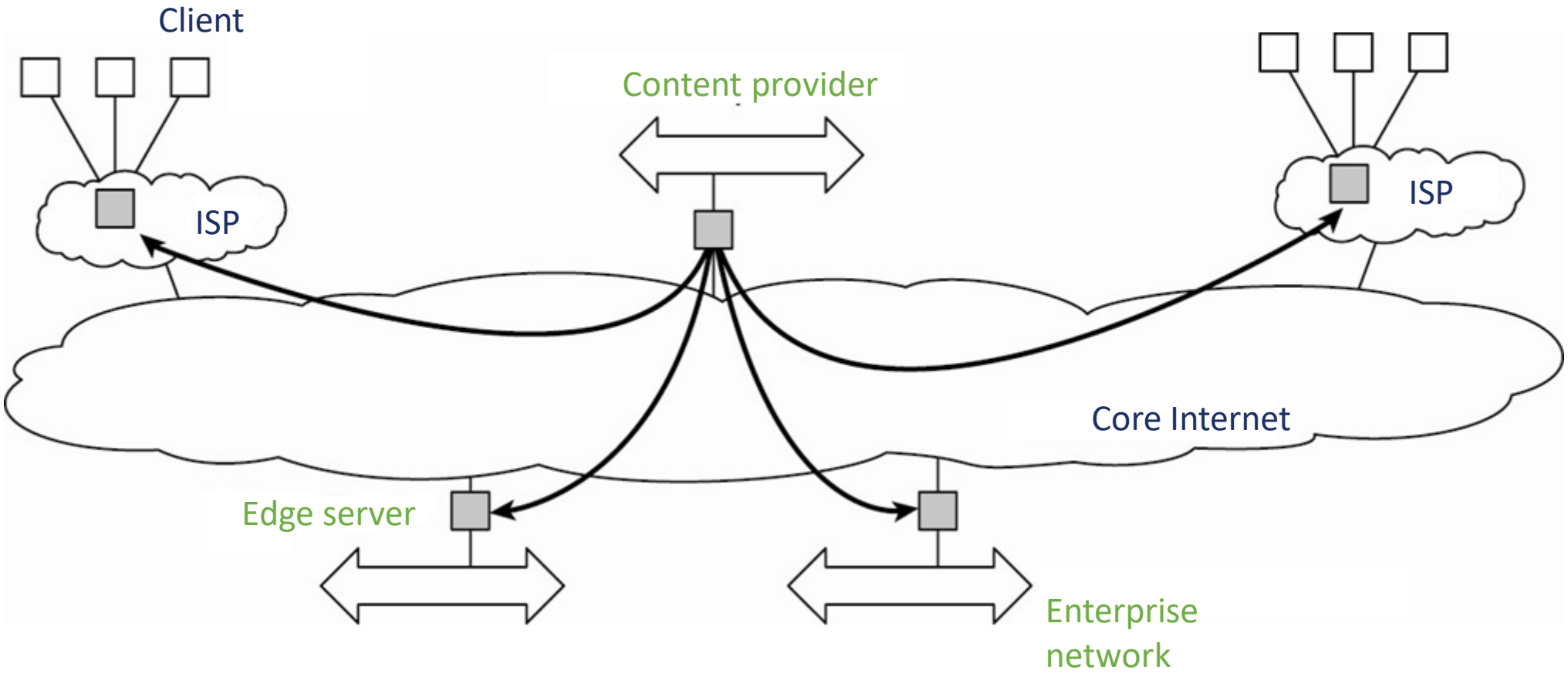
- This is quite challenging in unstructured P2P systems
  - Assume a data item is randomly placed
- Solution 1: Flood the network with a search query
- Solution 2: A randomized algorithm
  - Let us first assume that
    - Each node knows the IDs of k other randomly selected nodes
    - The ID of the hosting node is kept at m randomly picked nodes
  - The search is done as follows
    - Contact k direct neighbors for data items
    - Ask your neighbors to help if none of them knows
  - What is the probability of finding the answer directly?

Hybrid Architectures

- Many real distributed systems combine architectural features
  - E.g., the superpeer networks – combine client-server architecture (centralized) with peer-to-peer architecture (decentralized)
- Two examples of hybrid architectures
  - Edge-server systems
  - Collaborative distributed systems
  - Superpeer networks

Edge-Server Systems

- Deployed on the Internet where servers are “at the edge” of the network (i.e. first entry to network)
- Each client connects to the Internet by means of an edge server.



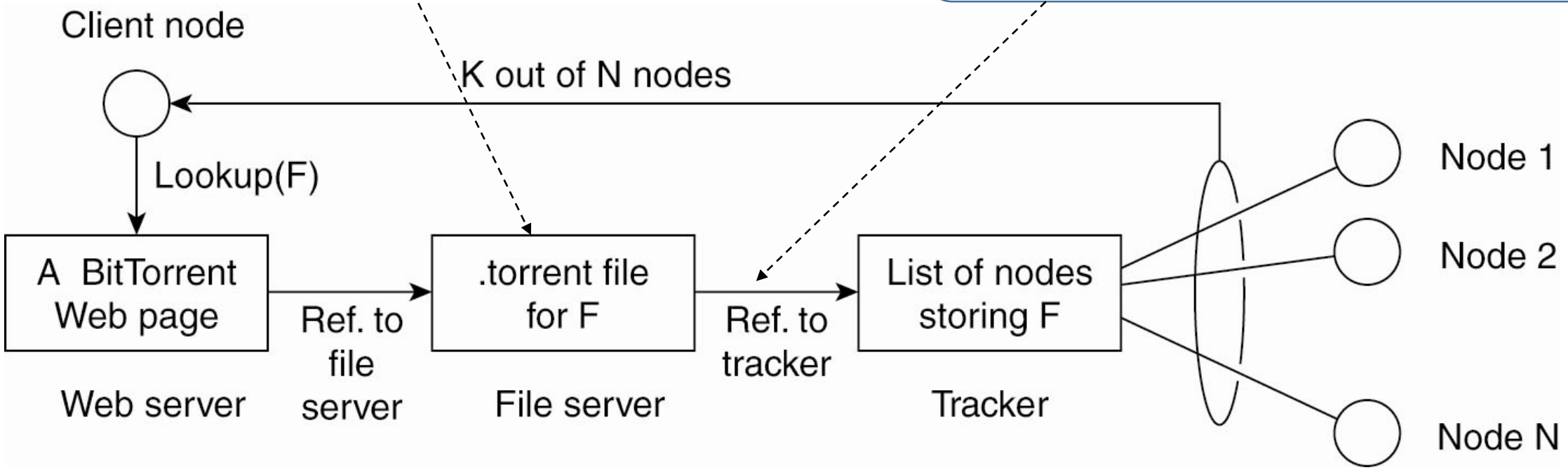
Collaborative Distributed Systems

- A hybrid distributed model that is based on mutual collaboration of various systems
  - Client-server scheme is deployed at the beginning
  - Fully decentralized scheme is used for collaboration after joining the system
- Examples of Collaborative Distributed System:
  - BitTorrent: is a P2P File downloading system. It allows download of various chunks of a file from other users until the entire file is downloaded
  - Globule: A Collaborative content distribution network. It allows replication of web pages by various web servers

BitTorrent

Information needed to download a specific file

Many trackers, one per file, tracker holds which node holds which chunk of the file

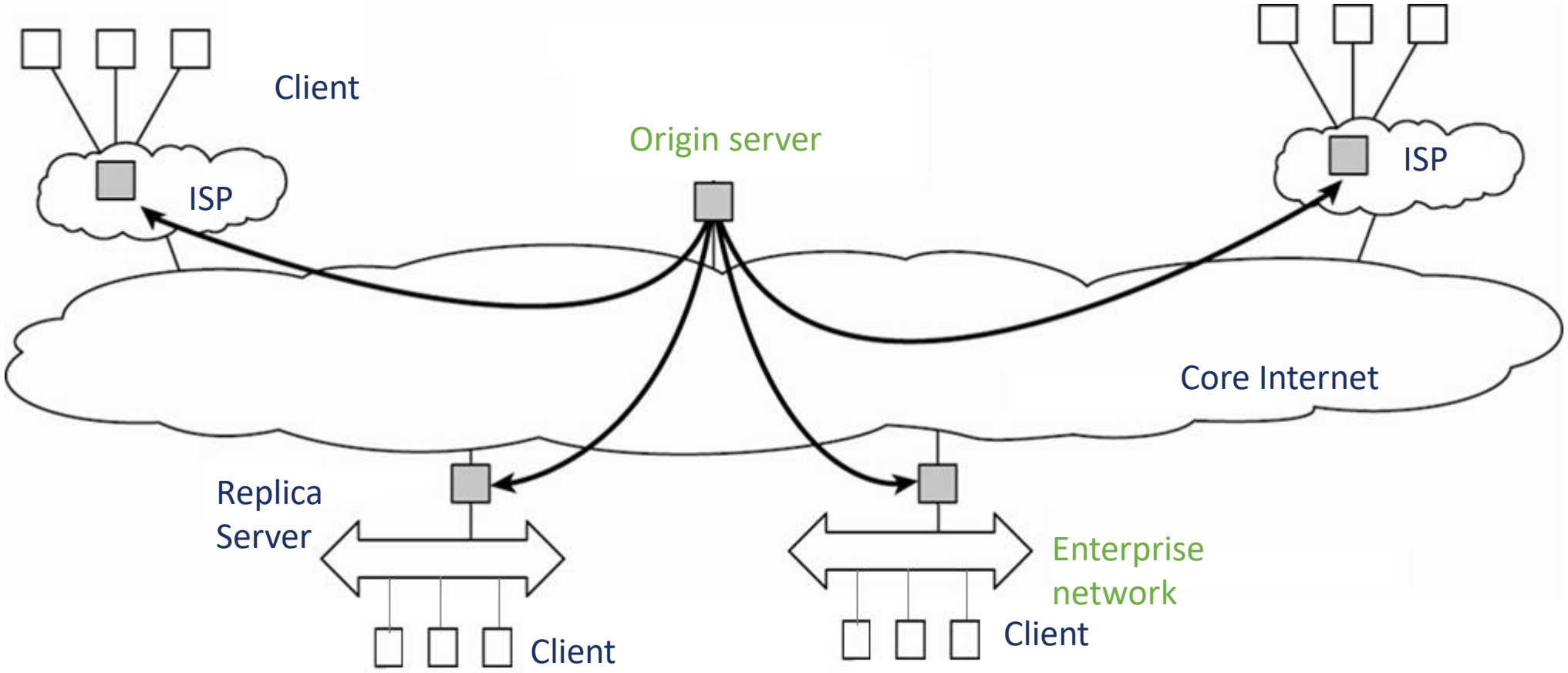


The principal working of BitTorrent (Pouwelse et al. 2004).

Globule

- Collaborative content distribution network:
  - Similar to edge-server systems
  - Enhanced web servers from various users that replicates web pages
- Components
  - A component that can redirect client requests to other servers.
  - A component for analyzing access patterns.
  - A component for managing the replication of Web pages.
- It Has a centralized component for registering the servers and make these servers known to others

Globule


Client
ISP
Origin server
ISP
Core Internet
Replica
Server
Enterprise
network
Client
Client

- To find more information on p2p architecture. Read this paper
- ***https://snap.stanford.edu/class/cs224w-readings/lua04p2p.pdf***