Distributed SYSTEM MODEL

Topics

- Introduction
- Architectural Models
- Fundamental Models

Introduction

- An architectural model of a distributed system is concerned with the placement of its parts and the relationships between them.

- Examples include:
  - Client-Server model
  - Peer-to-Peer model

Introduction

- Variations of client-sever model can be formed by:
  - The partition of data or replication at cooperative servers
  - The caching of data by proxy servers and clients
  - The use of mobile code and mobile agents
  - The requirement to add or remove mobile devices in a convenient manner

Introduction

- Fundamental Models deal with a more formal description of the properties that are common in all of the architectural models.

- Some of these properties in distributed systems are:
  - There is no global time in a distributed system.
  - All communication between processes is achieved by means of messages.

Introduction

- Message communication in distributed systems has the following properties:
    - Delay
    - Failure
    - Security attacks

Introduction

- Message communication issues are addressed by three models:
  - Interaction Model
    - It deals with performance and with the difficulty of setting of time limits in a distributed system.
  - Failure Model
    - It attempts to give a precise specification of the faults that can be exhibited by processes and communication channels.
  - Security Model
    - It discusses possible threats to processes and communication channels.

Architectural Models-Intro

- The architecture of a system is its structure in terms of separately specified components.

  - The overall goal is to ensure that the structure will meet present and likely future demands on it.

  - Major concerns are to make the system:
    - Reliable
    - Manageable
    - Adaptable
    - Cost-effective

Architectural Models-Intro

- An architectural Model of a distributed system first simplifies and abstracts the functions of the individual components of a distributed system.

- An initial simplification is achieved by classifying processes as:
  - Server processes
  - Client processes
  - Peer processes
    - Cooperate and communicate in a symmetric manner to perform a task.

Software Layers

- Software architecture referred to:
  - The structure of software as layers or modules in a single computer.
  - The services offered and requested between processes located in the same or different computers.

- Software architecture is breaking up the complexity of systems by designing them through layers and services.
  - Layer: a group of related functional components.
  - Service: functionality provided to the next layer.

(Figure 1)

Software Layers

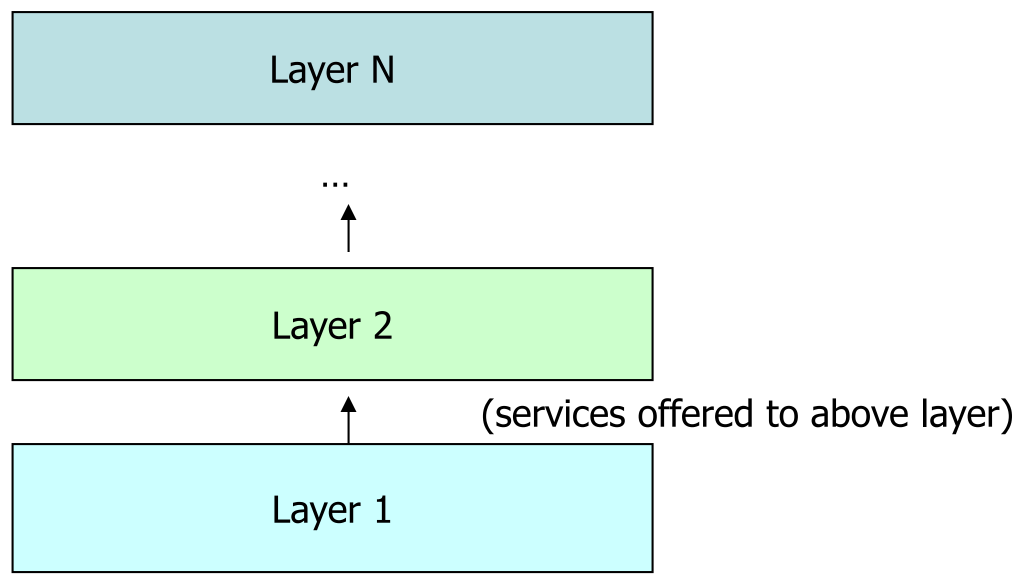


Figure 1. Software layers

Software Layers

- Platform
  - The lowest-level hardware and software layers are often referred to as a platform for distributed systems and applications.
    - ❖ These low-level layers provide services to the layers above them, which are implemented independently in each computer.
    - ❖ These low-level layers bring the system's programming interface up to a level that facilitates communication and coordination between processes.

(Figure 2)

Software Layers

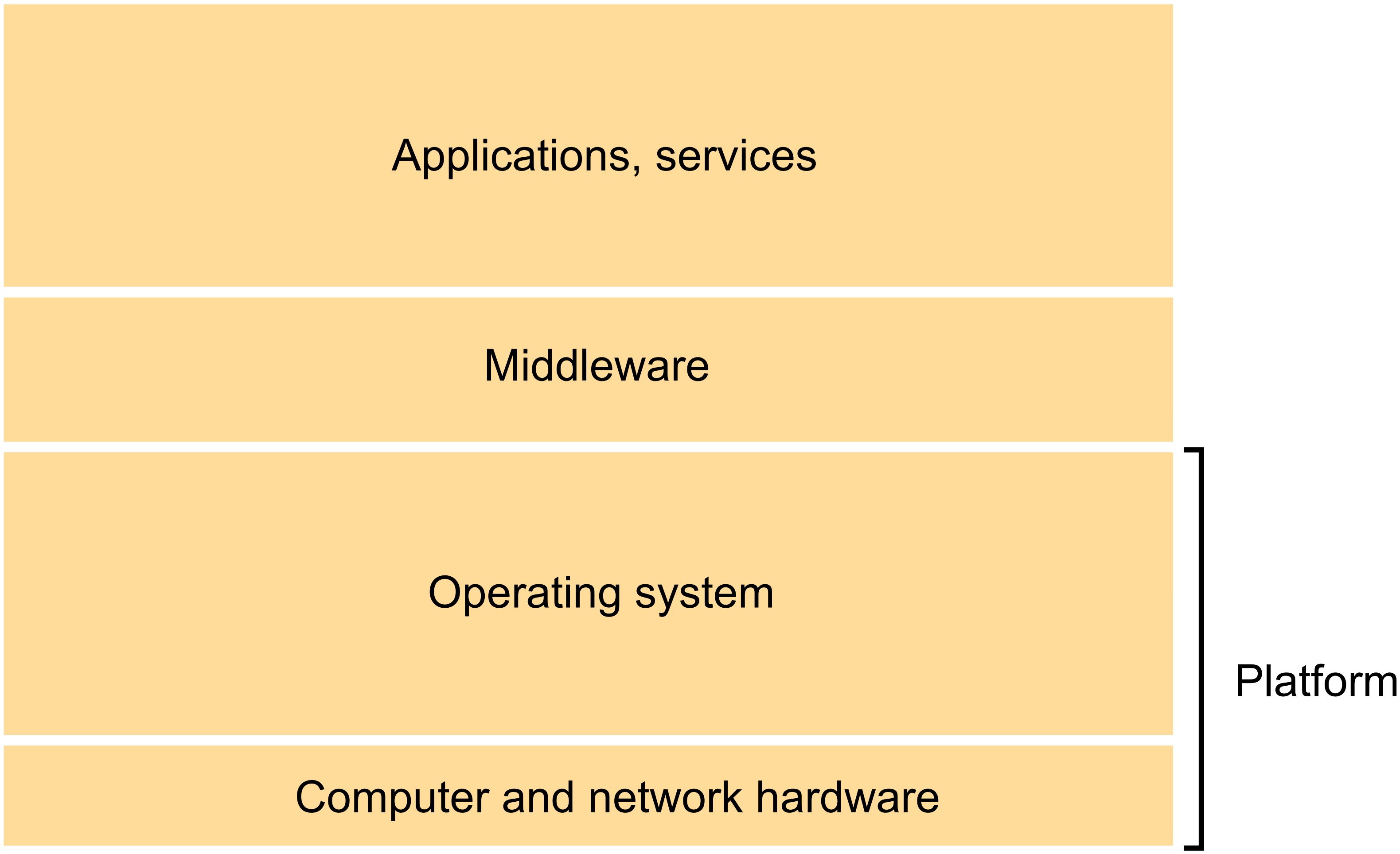


Figure 2. Software and hardware service layers in distributed systems

Software Layers

- Middleware
  - A layer of software whose purpose is
    - to mask heterogeneity presented in distributed systems.
    - To provide a convenient programming model to application developers.
  - Major Examples of middleware are:
    - Sun RPC (Remote Procedure Calls)
    - OMG CORBA (Common Object Request Broker Architecture)
    - Microsoft D-COM (Distributed Component Object Model)
    - Sun Java RMI

Variants of Client Sever Model

- The problem of client-server model is placing a service in a server at a single address that does not scale well beyond the capacity of computer host and bandwidth of network connections.

- To address this problem, several variations of client-server model have been proposed.
- Services provided by multiple servers
  - Services may be implemented as several server processes in separate host computers interacting as necessary to provide a service to client processes.

  - E.g. cluster that can be used for search engines.

(Figure 6)

Variants of Client Sever Model

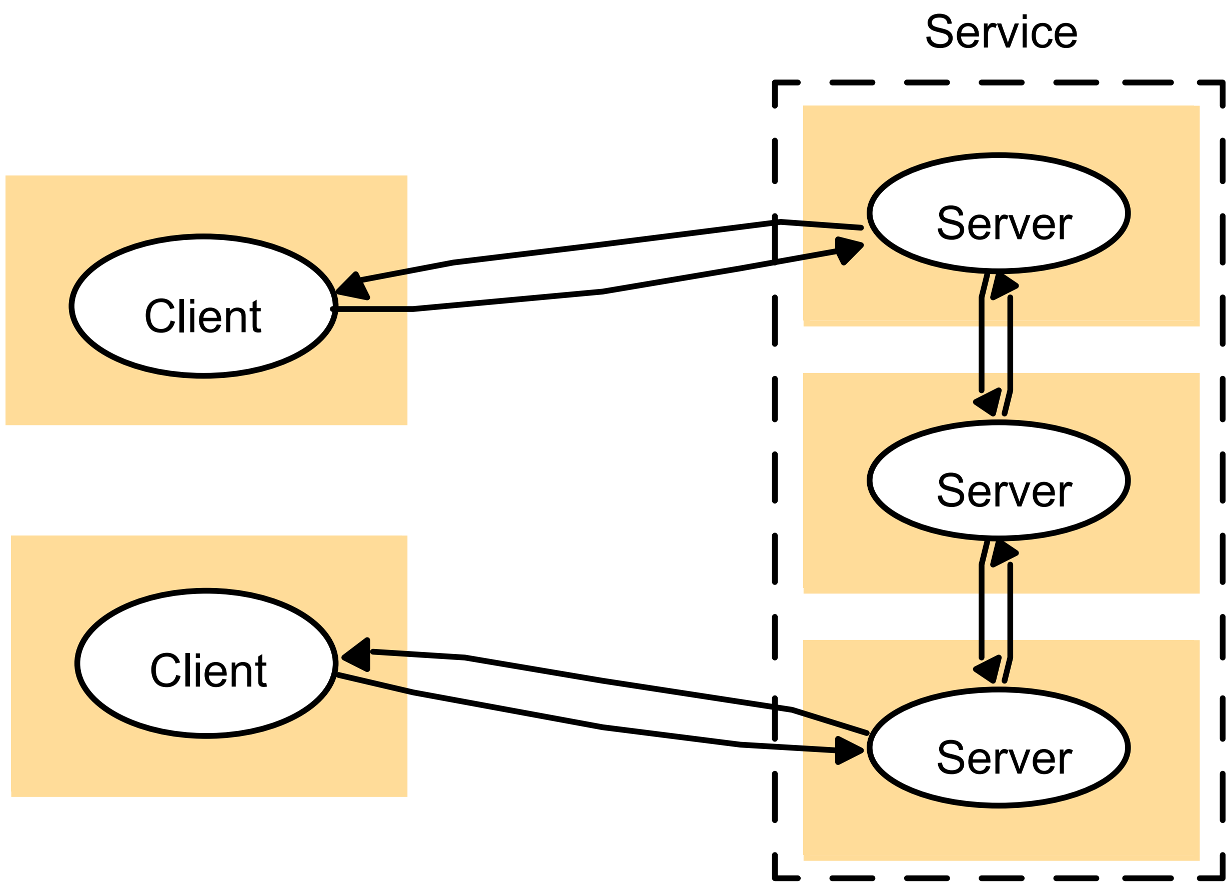


Figure 6. A service provided by multiple servers.

Variants of Client Sever Model

- Proxy servers and caches
  - A cache is a store of recently used data objects.
  - When a new object is received at a computer it is added to the cache store, replacing some existing objects if necessary.
  - When an object is needed by a client process the caching service first checks the cache and supplies the object from there if an up-to-date copy is available.
  - If not, an up-to-date copy is fetched.

Variants of Client Sever Model

- Caches may be collected with each client or they may be located in a proxy server that can be shared by several clients.

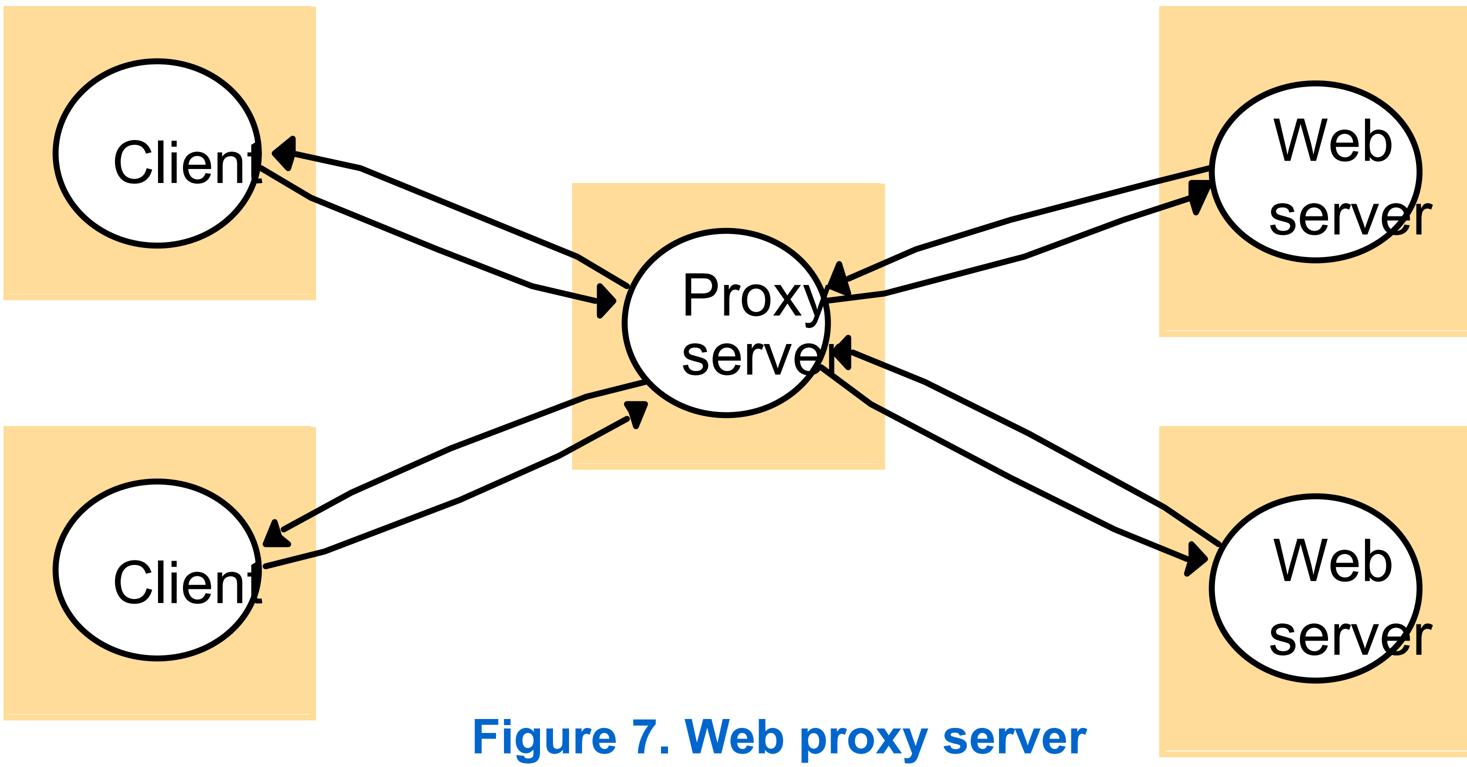


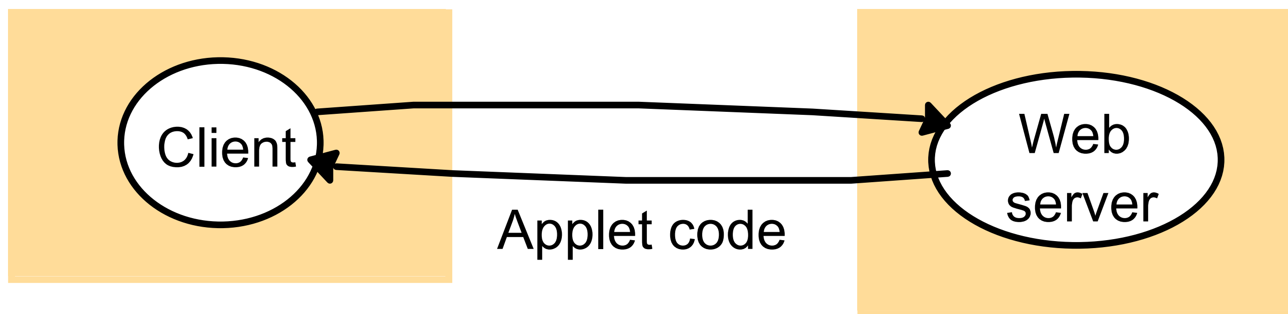
Figure 7. Web proxy server

Variants of Client Sever Model

- Mobile code
  - Applets are a well-known and widely used example of mobile code.
  - Applets downloaded to clients give good interactive response
  - Mobile codes such as Applets are a potential security threat to the local resources in the destination computer.

Variants of Client Sever Model

a) client request results in the downloading of applet code



b) client interacts with the applet

Figure 8. Web applets

Variants of Client Sever Model

- Mobile agents
  - A running program (code and data) that travels from one computer to another in a network carrying out of a task, usually on behalf of some other process.
  - Examples of the tasks that can be done by mobile agents are:
    - To collect information.
    - To install and maintain software maintained on the computers within an organization.
    - To compare the prices of products from a number of vendors.

Variants of Client Sever Model

- Mobile agents are a potential security threat to the resources in computers that they visit.

- The environment receiving a mobile agent should decide on which of the local resources to be allowed to use.

- Mobile agents themselves can be vulnerable
    - They may not be able to complete their task if they are refused access to the information they need.

Variants of Client Sever Model

- Mobile devices and spontaneous interoperation
  - Mobile devices are hardware computing components that move between physical locations and thus networks, carrying software component with them.
  - Many of these devices are capable of wireless networking ranges of hundreds of meters such as WiFi (IEEE 802.11), or about 10 meters such as Bluetooth.

Variants of Client Sever Model

- Network computers
  - It downloads its operating system and any application software needed by the user from a remote file server.
  - Applications are run locally but the files are managed by a remote file server.
  - Network applications such as a Web browser can also be run.

Variants of Client Sever Model

- Thin clients
  - It is a software layer that supports an user interface on a computer that is local to the user while executing application programs on a remote computer.
  - This architecture has the same low management and hardware costs as the network computer scheme.
  - Instead of downloading the code of applications into the user’s computer, it runs them on a compute server.

Variants of Client Sever Model

- Compute server is a powerful computer that has the capacity to run large numbers of application simultaneously.

- The compute server will be a multiprocessor or cluster computer running a multiprocessor version of an operation system such as UNIX or Windows.

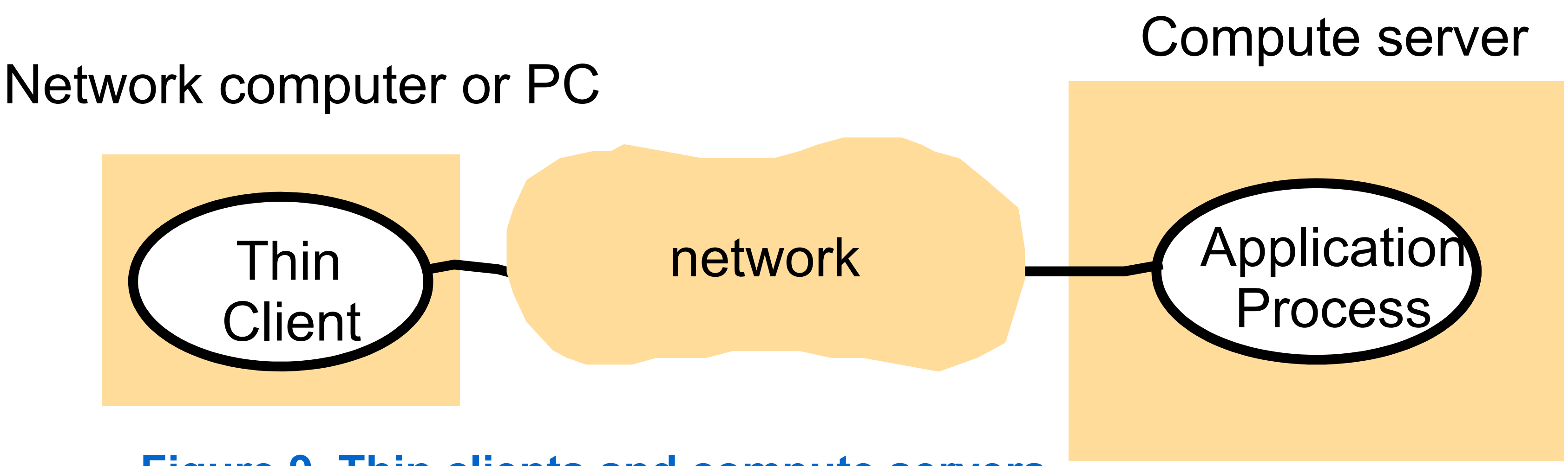
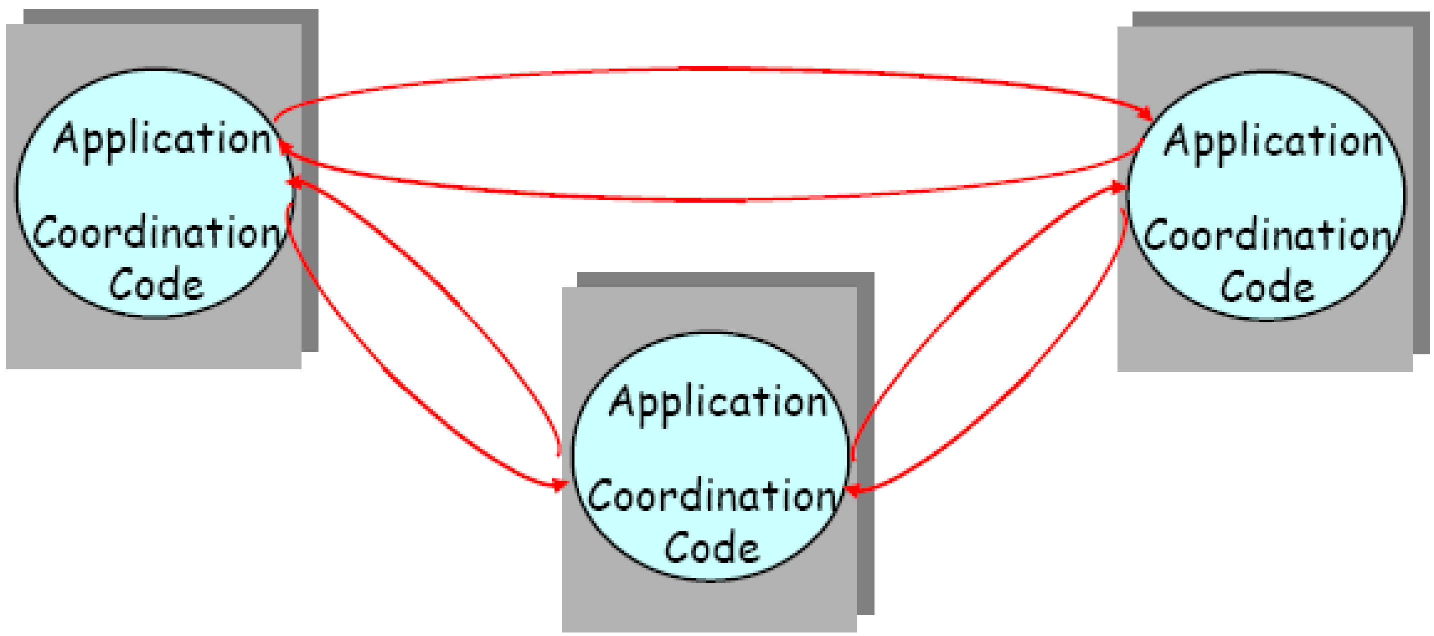


Figure 9. Thin clients and compute servers

Peer-to-Peer Computing

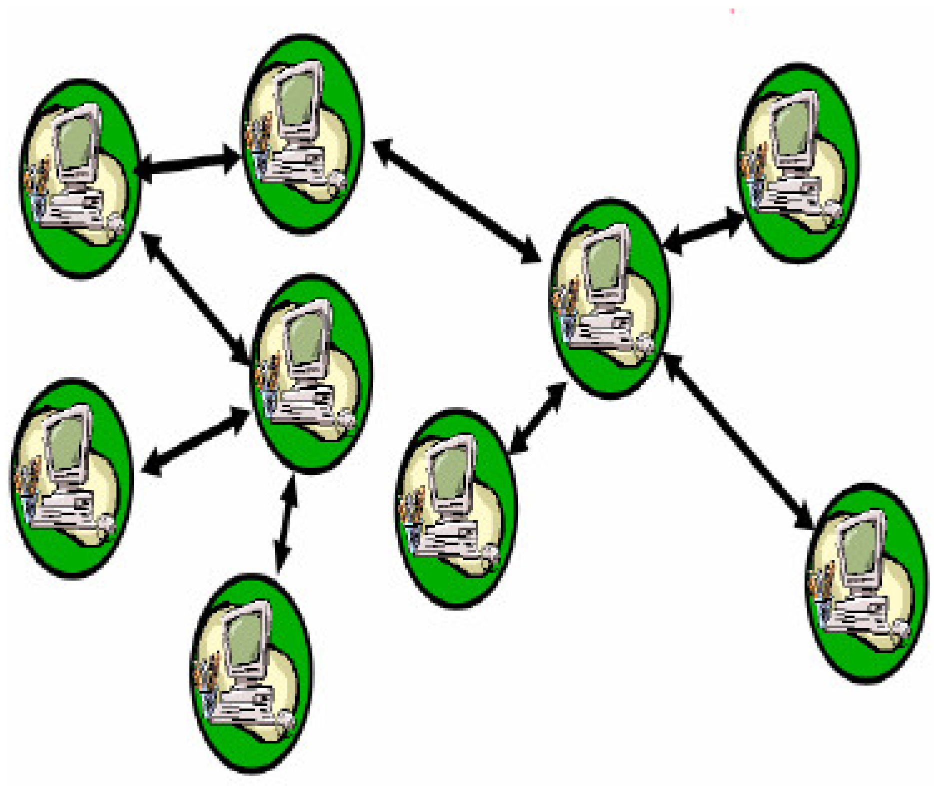
The Peer-to-Peer Model

- Applications based on peer processes
  - Not Client-Server
  - processes that have largely identical functionality



Definitions

- Everything except the client/server model
- Network of nodes with equivalent capabilities/responsibilities (symmetrical)
- Nodes are both Servers and clients called "Servents"
- Direct exchange of information between hosts at the edge of the Internet

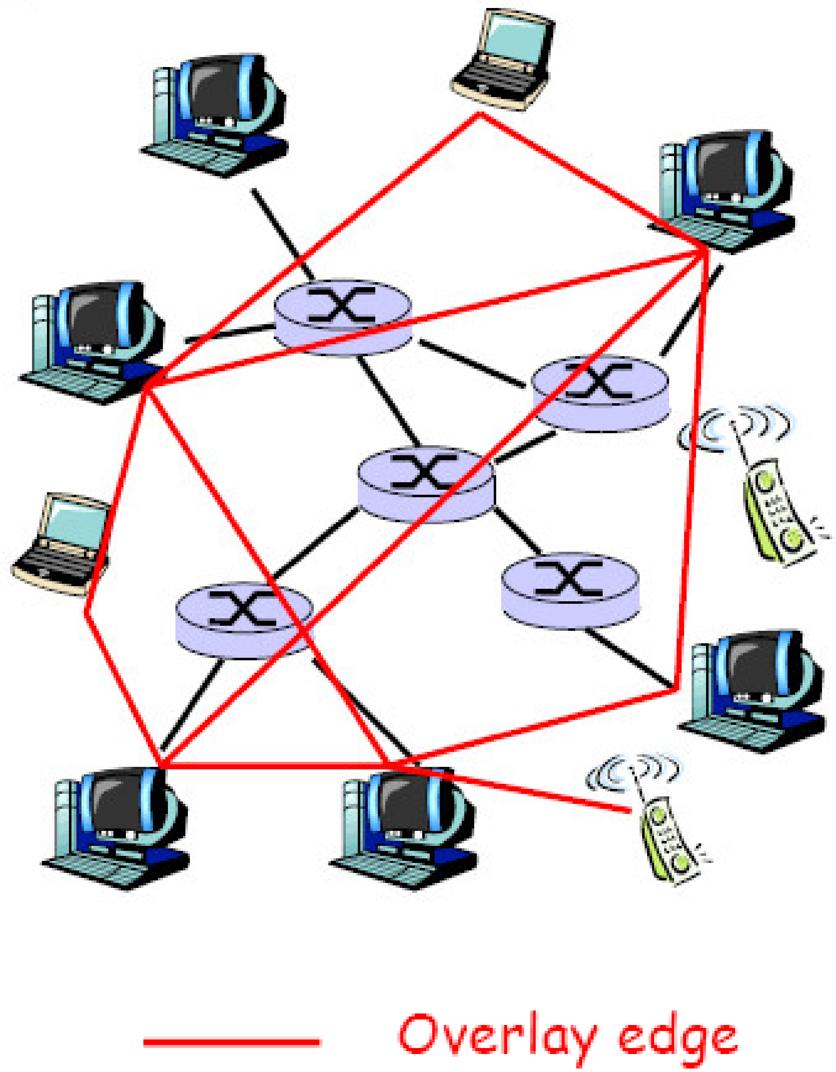


Definitions (cont.)

- A transient network that allows a group of computer users to connect with each other and collaborate by sharing resources (CPU, storage, content).
- The connected peers construct a virtual overlay network on top of the underlying network infrastructure
- Examples of overlays:
  - BGP routers and their peering relationships
  - Content distribution networks (CDNs)
  - And P2P apps !

Overlay Networks

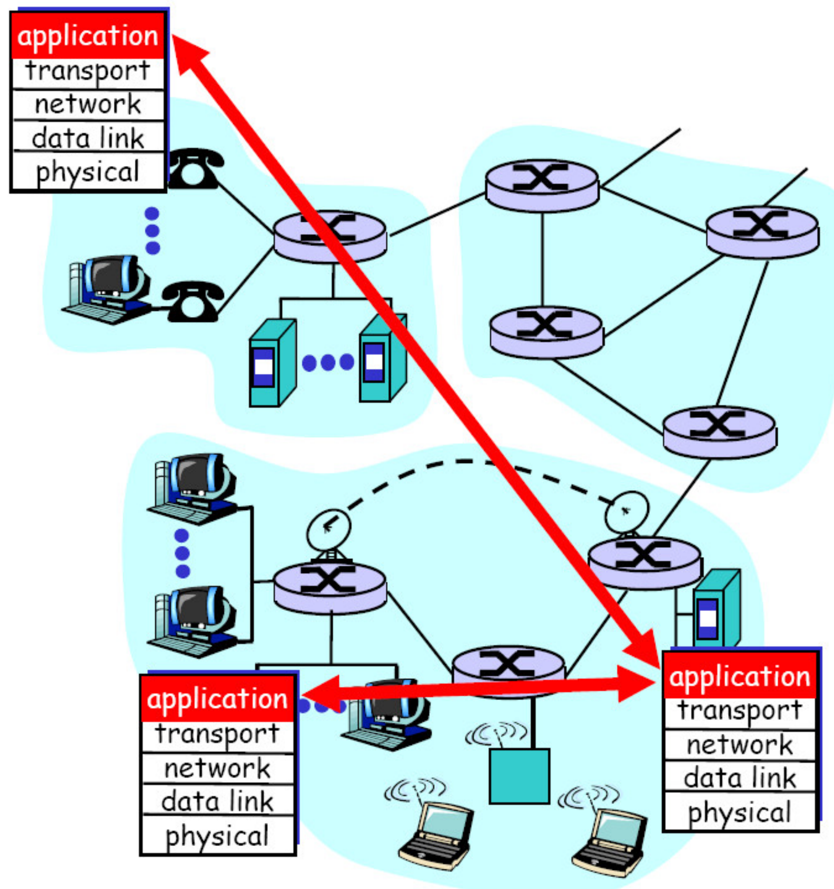
- An overlay network is a set of logical connections between end hosts
- Proximity not necessarily taken into account



Overlays: All in the application layer

- Design flexibility
  - Topology
  - Protocol
  - Messaging over TCP, UDP, ICMP

Underlying physical net is transparent to developer



P2P Goals

- Cost reduction through cost sharing
  - Client/Server: Server bears most of the cost
  - P2P: Cost spread over all the peers (+Napster, ++Gnutella,…)
- Interoperability
  - for the aggregation of diverse resources (storage, CPU, …)
- Increased autonomy
  - independence from servers, hence providers (e.g., A way around censorship, licensing restrictions, etc.)

Goals (cont.)

- Anypmity/privacy
  - Difficult to ensure with a central server
  - Required by users who do not want a server/provider to know their involvement in the system

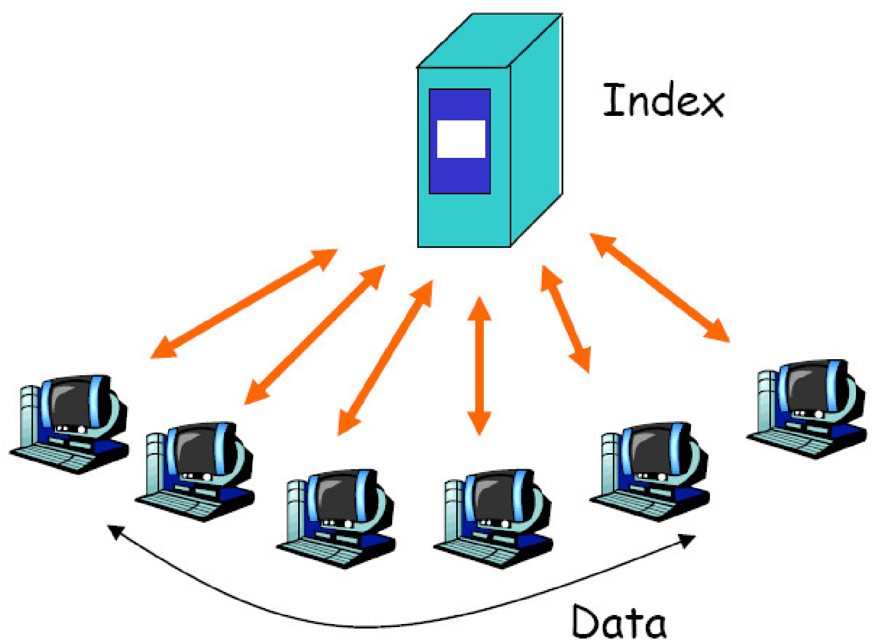
- Dynamism and Ad hoc communications
  - Resources (e.g., compute nodes) enter and leave the system continuously
  - P2P systems typically do not rely on an established infrastructure
  - they build their own, e.g. logical overlay in CAN

P2P Classification

- Degree of P2P decentralization
  - Hybrid decentralized P2P
  - Purely decentralized P2P
  - Partially centralized P2P
- Degree of P2P structure
  - Structured P2P
  - Loosely structured P2P
  - Unstructured P2P

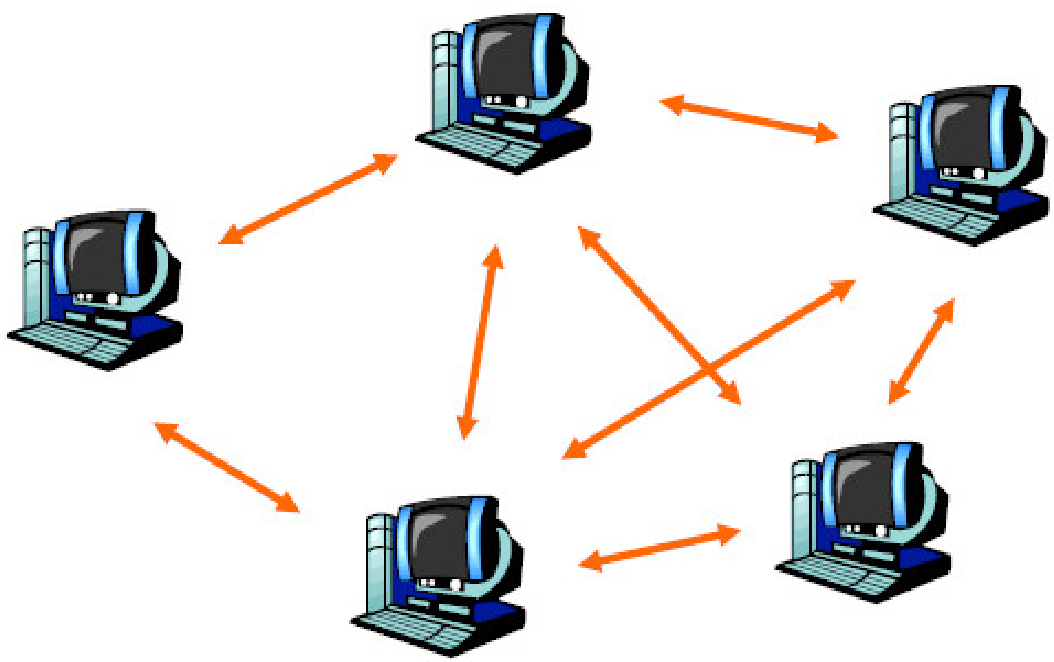
Hybrid decentralized P2P

- Central server facilitates the interaction b/w peers.
- Central server performs the lookups and identifies the nodes of the network.
- example: Napster
- (-) Single point of failure, scalability?, …



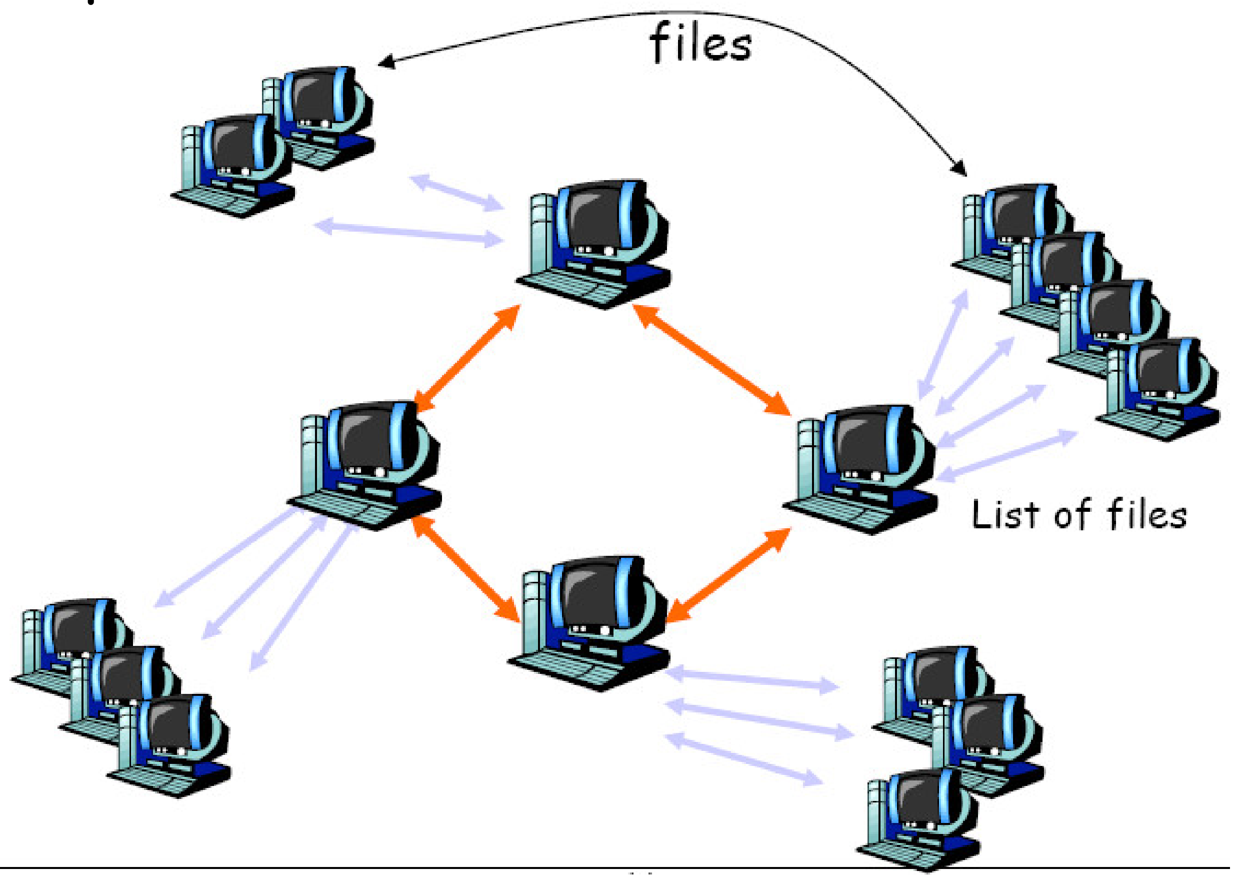
Purely decentralized P2P

- network nodes perform the same tasks (Servents)
- no central coordination activity
- examples: original Gnutella, Freenet
- (-) data consistency?, Manageability?, Security?, Comm. overhead



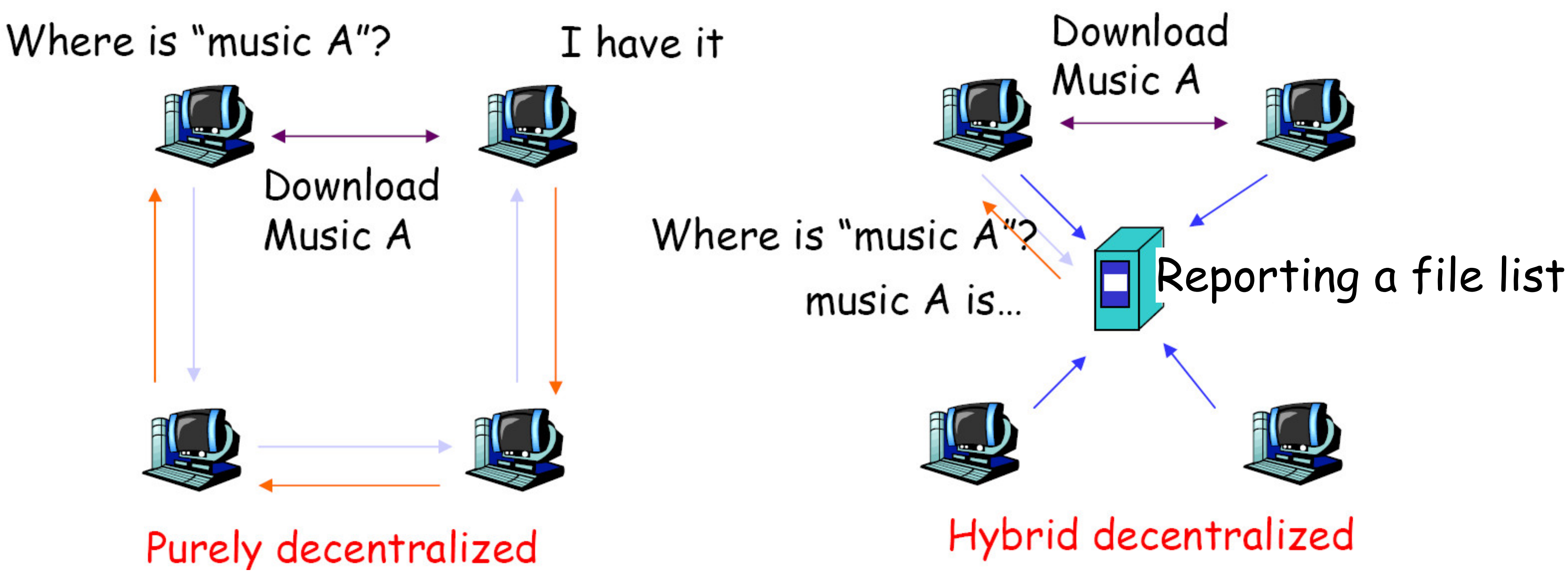
Partially centralized P2P

- some of the nodes assume a more important role
- Supernodes act as local central indexes
- examples: Kazaa, recent Gnutella



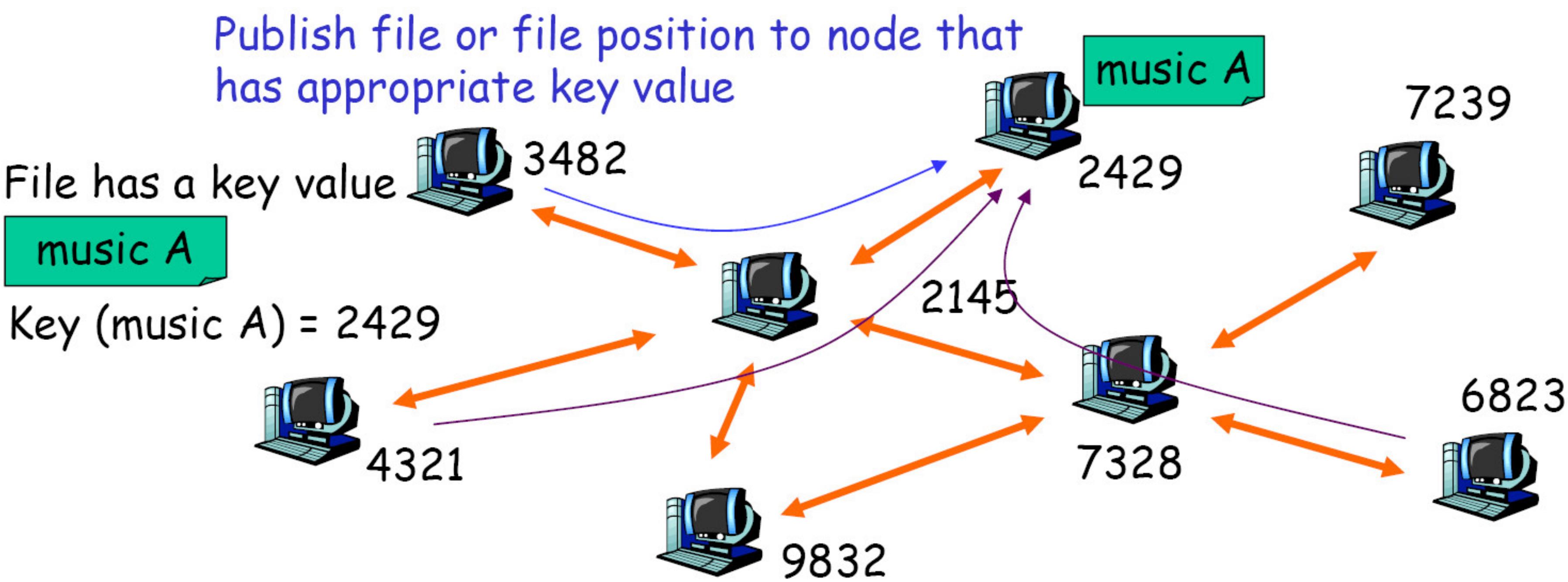
Unstructured P2P

- data is distributed randomly over the peers and broadcasting mechanisms are used for searching.
- examples: Napster, Gnutella, KaZaa



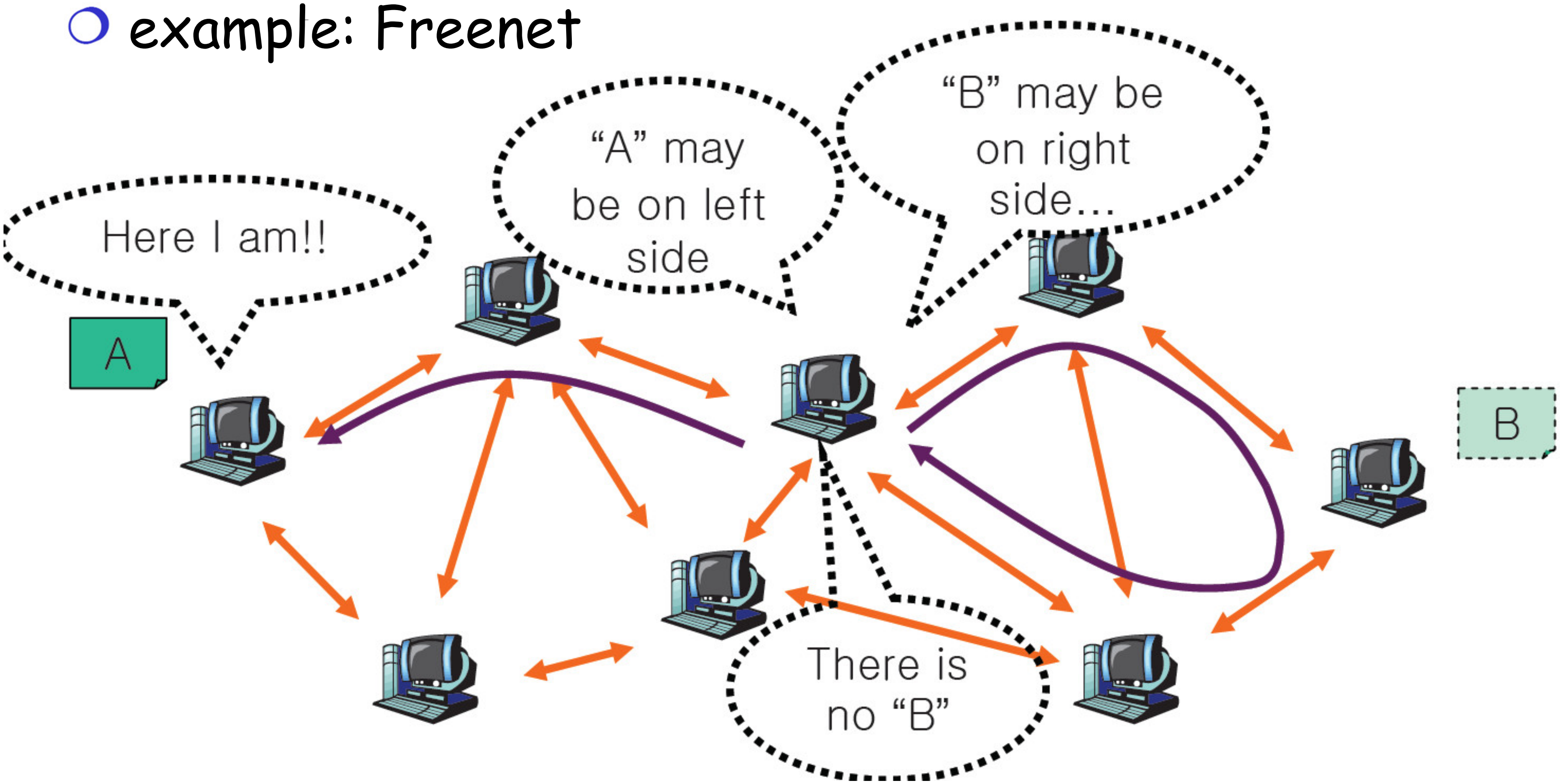
Structured P2P

- Network topology is tightly controlled and files are placed at precisely specified locations.
- Provide a mapping between the file identifier and location
- Examples: Chord, Tapestry, Pastry, etc.



Loosely Structured P2P

- Between structured and unstructured
- File locations are affected by routing hints, but they are not completely specified.
- example: Freenet



P2P Applications

- File Sharing
- Communication
- Collaboration
- Computation
- Databases
- Others

P2P File Sharing (cont.)

- Examples of P2P file sharing applications:
  - Napster
    - disruptive; proof of concept
  - Gnutella
    - open source
  - KaZaA
    - at some point, more KaZaAtraffic than Web traffic!
  - eDonkey
    - popular in Europe
  - BitTorrent:
    - 53% of all P2P traffic in June 2004 was BitTorrent traffic
  - and many others…

P2P Communication

- Instant Messaging (IM)
  - User A runs IM client on her PC
  - Intermittently connects to Internet; gets new IP address for each connection
  - Registers herself with “system”
  - Learns from “system”that user B in her “buddy list”is active
  - User A initiates direct TCP connection with User B: P2P
  - User A and User B chat.
  - Can also be voice, video and text.
- Audio-Video Conferencing
  - Example: Voice-over-IP (Skype)

P2P Databases

- Fragments large database over physically distributed nodes
- Overcomes limitations of distributed DBMS
  - Static topology
  - Heavy administration work
- Dissemination of data sources over the Internet
  - Each peer is a node with a database
  - Set of peers changes often (site availability, usage patterns)
- Examples:
  - AmbientDB (http://homepages.cwi.nl/~boncz/ambientdb.html)
  - XPeer: self-organizing XML DB

What is a DHT?

- Hash Table
  - data structure that maps “keys” to “values”
- Interface
  - put(key, value)
  - get(key)
- Distributed Hash Table (DHT)
  - similar, but spread across the Internet
  - challenge: locate content

What is a DHT? (cont.)

- Single-node hash table:

  Key = hash (data)

  put(key, value)

  get(key)->value

- Distributed Hash Table (DHT):

  Key = hash (data)

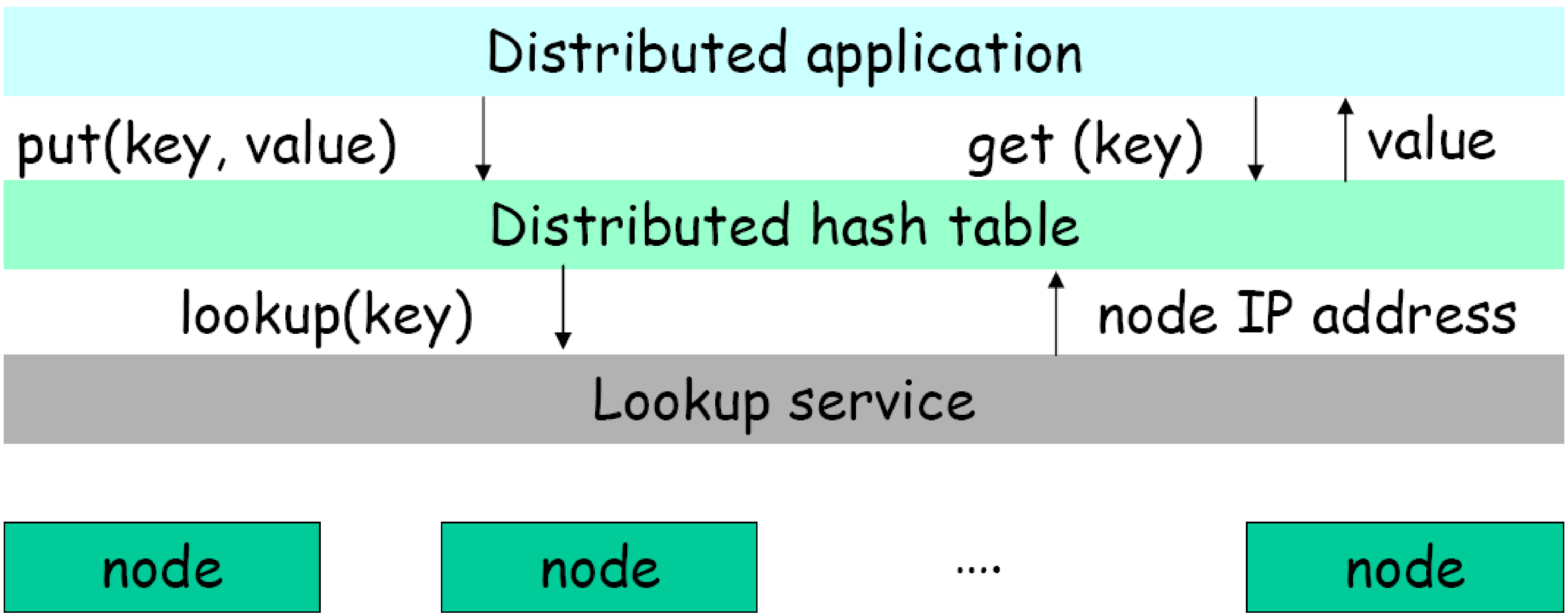
  Lookup (key) -> node-IP@

  Route (node-IP@, PUT, key, value)

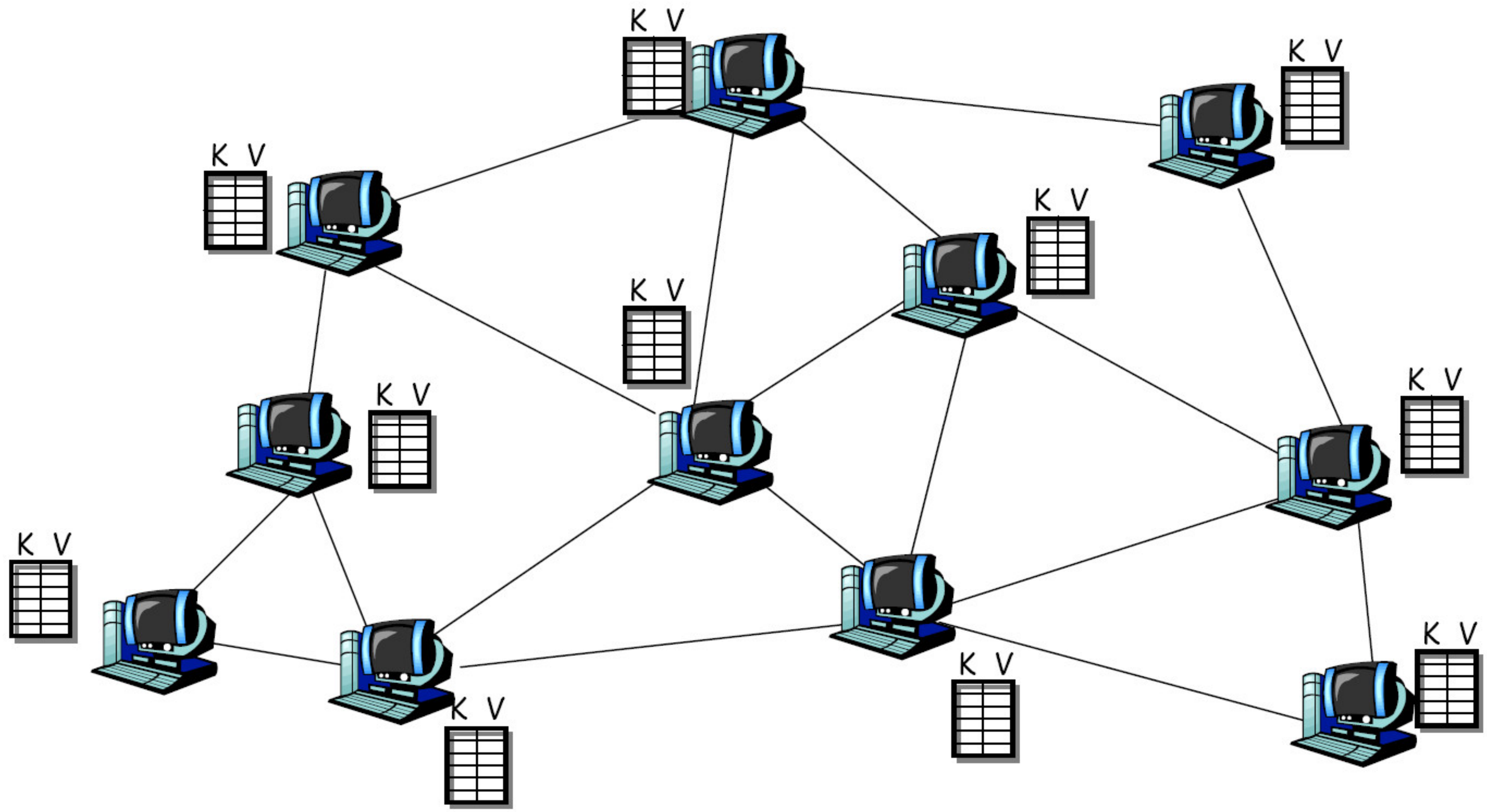
  Route (node-IP@, GET, key) -> value

- Idea:
  - Assign particular nodes to hold particular content (or reference to content)
  - Every node supports a routing function (given a key, route messages to node holding key)

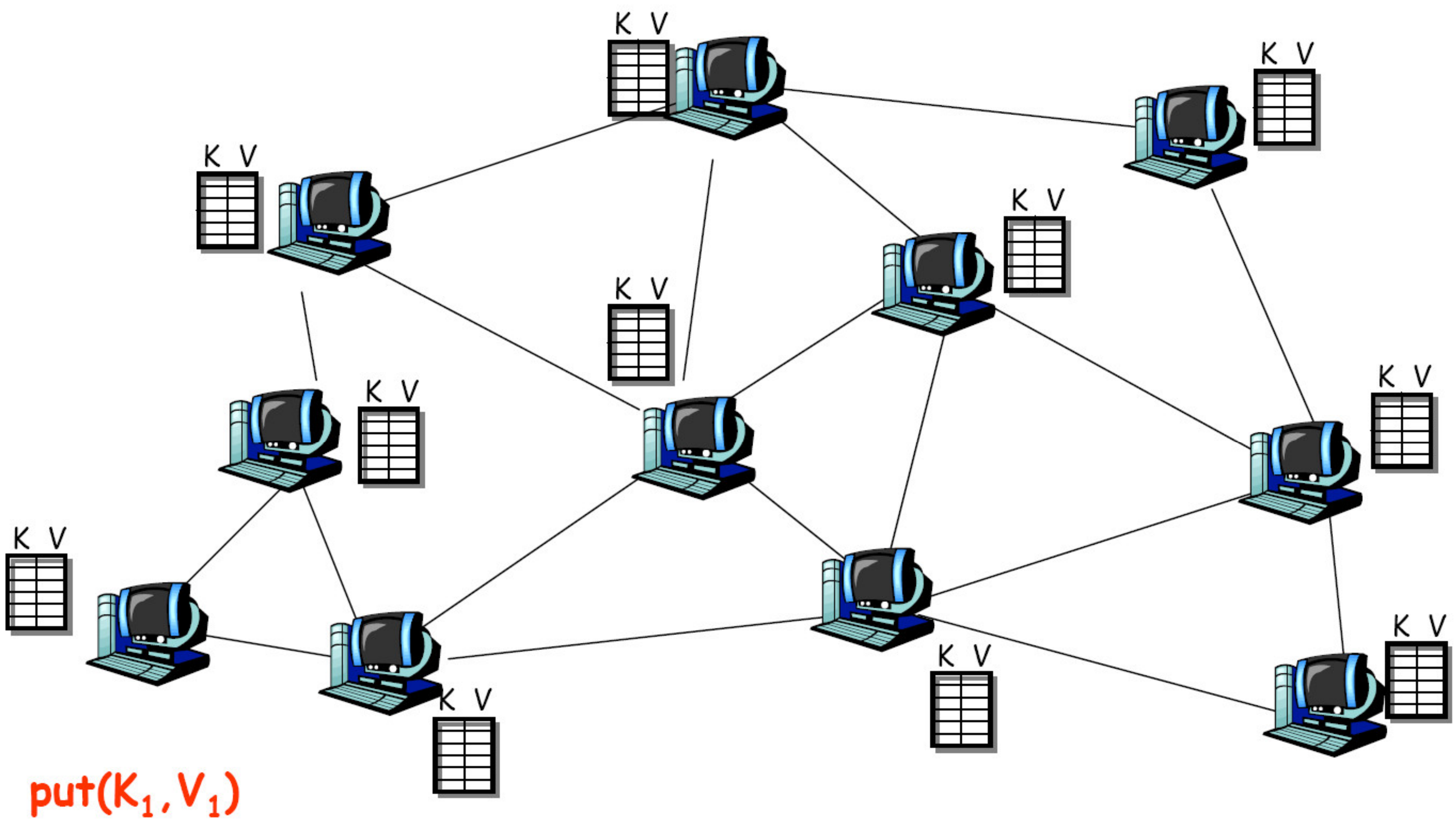
What is a DHT? (cont.)



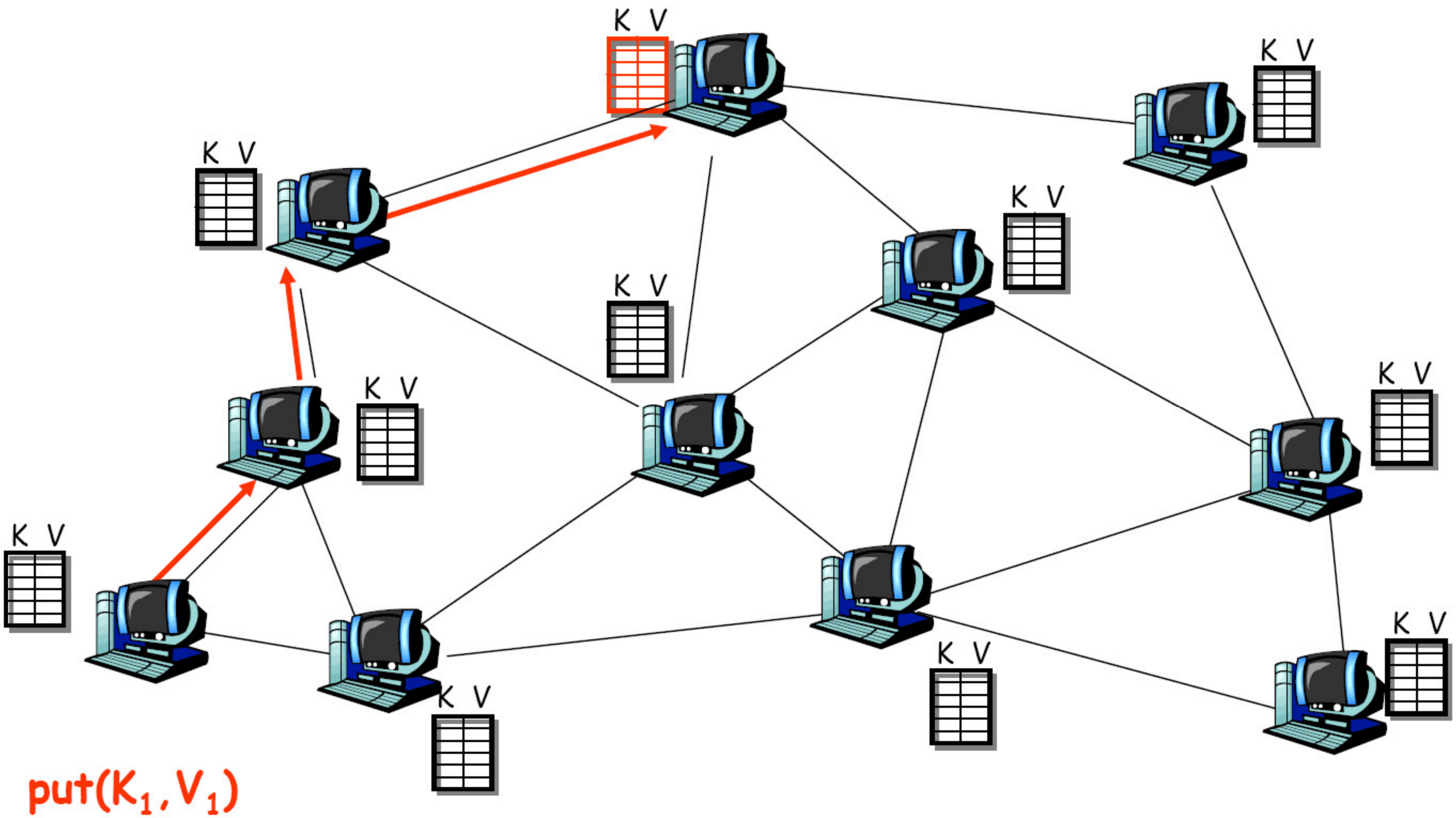
DHT in action



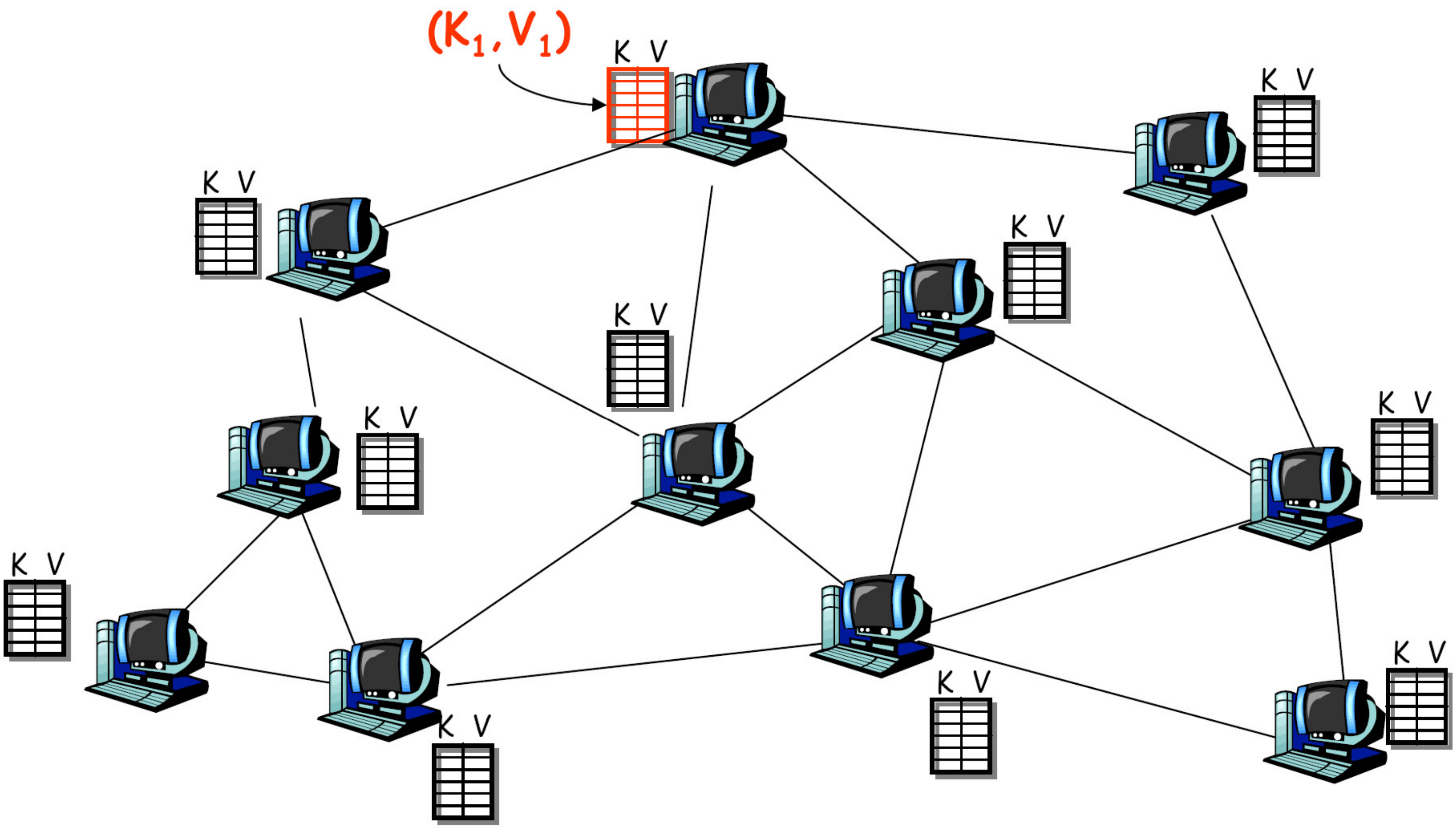
DHT in action: put()



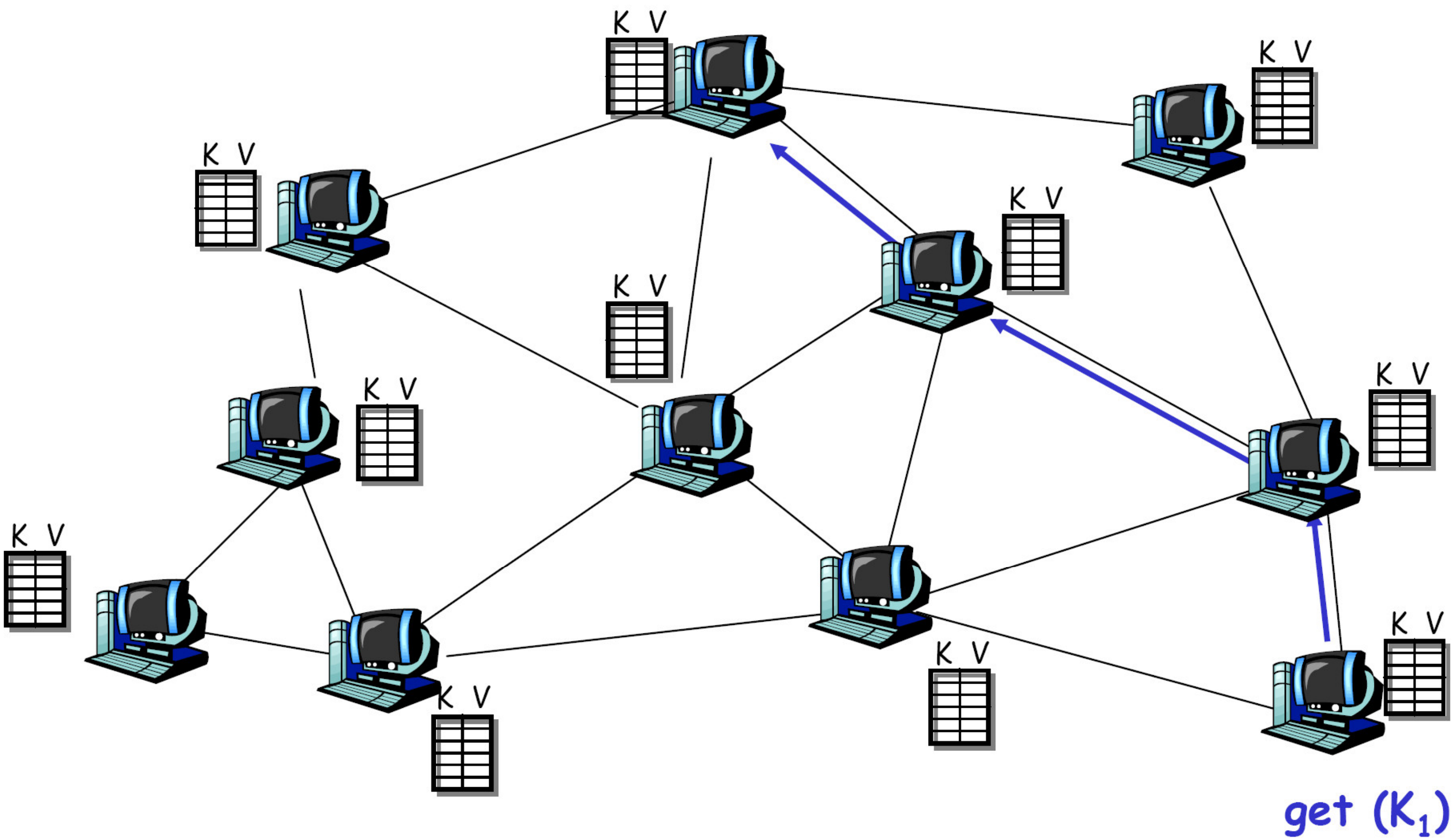
DHT in action: put()



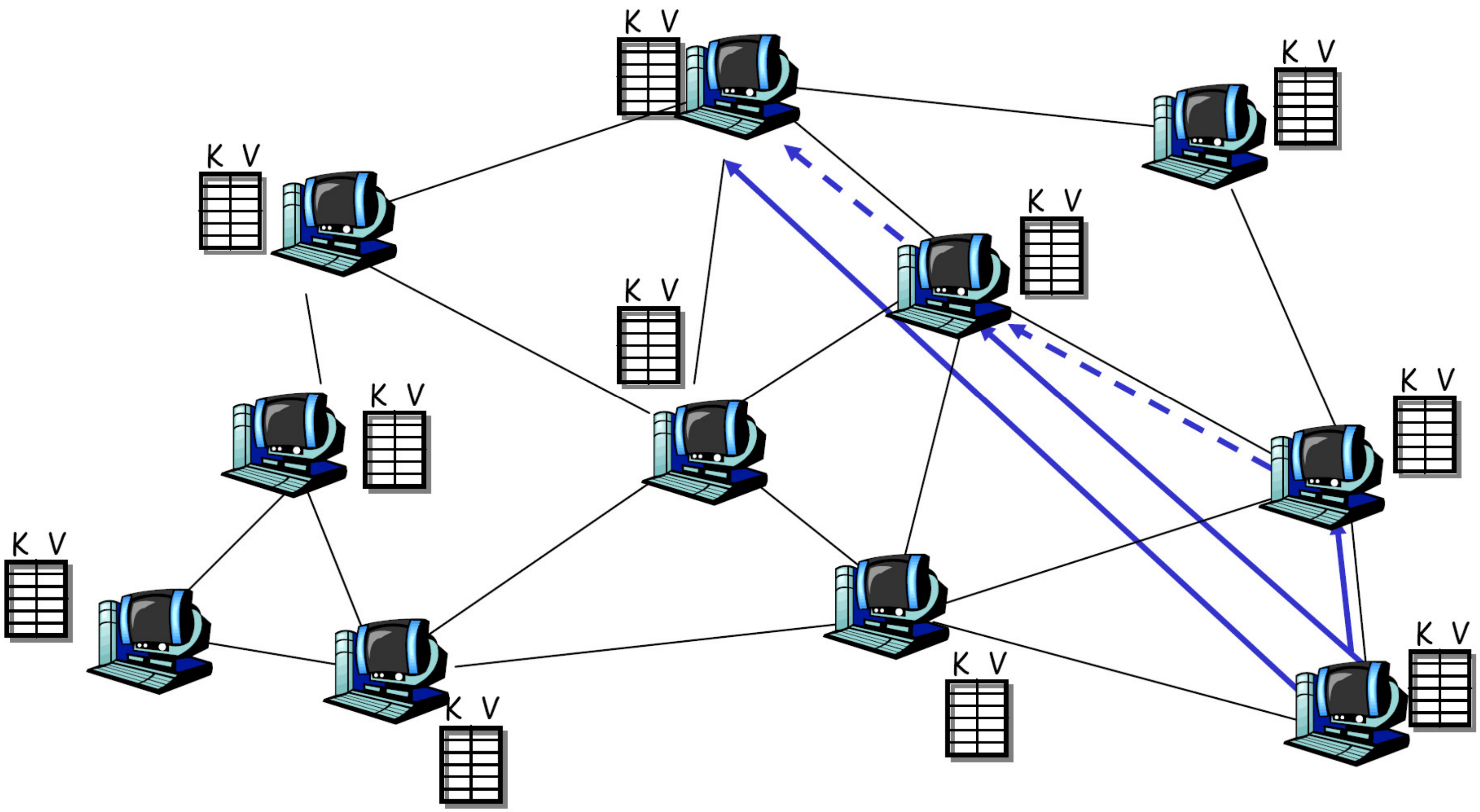
DHT in action: put()



DHT in action: get()



Iterative vs. Recursive Routing



Resource Management

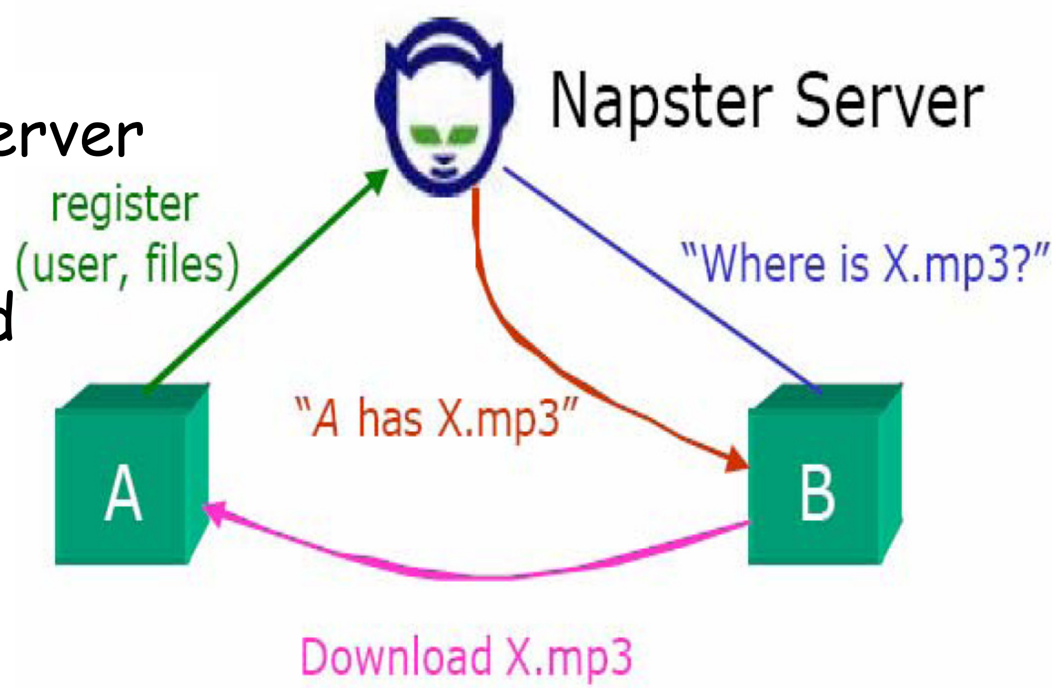
- Focus here is on p2p content distribution systems
- Main resources to be managed:
  - Content
  - Storage capacity
  - Bandwidth

Resource Management (cont.)

- Content management: deletion, update and versioning
  - Often not supported for security, robustness to attacks, lack of synchronization between peers
  - Update and deletion provided to publishers
  - Complex content history archival (OceanStore)

Napster

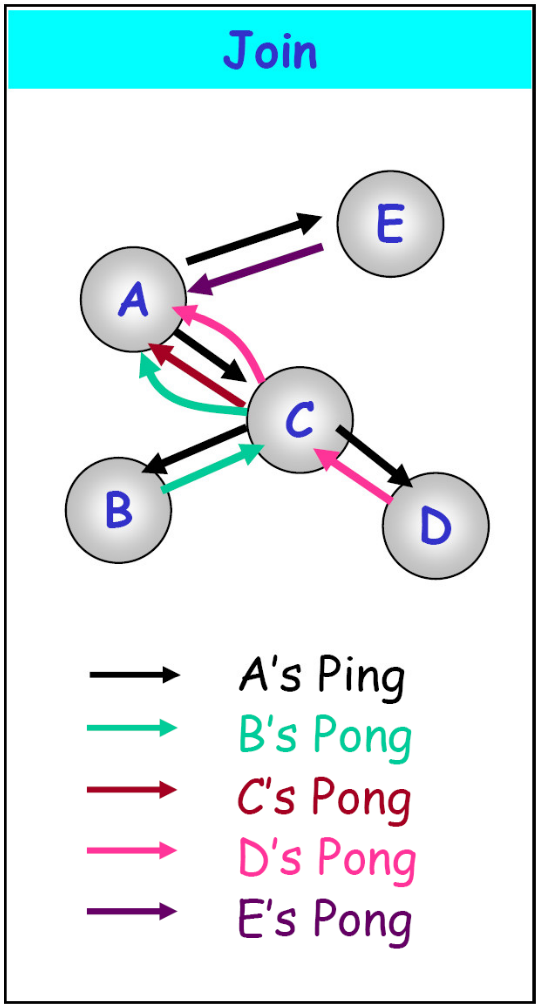
- Hybrid decentralized, instructure.
- Combination of client/server and P2P approaches
- A network of registered users running a client software, and a central directory server
- The server maintains 3 tables:
  - (File_Index, File_Metadata)
  - (User_ID, User_Info)
  - (User_ID, File_Index)

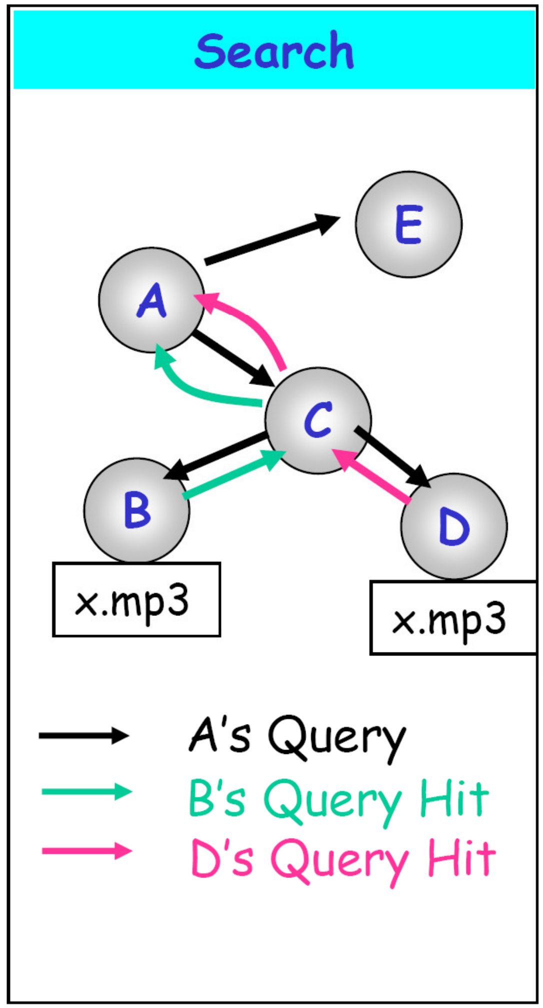


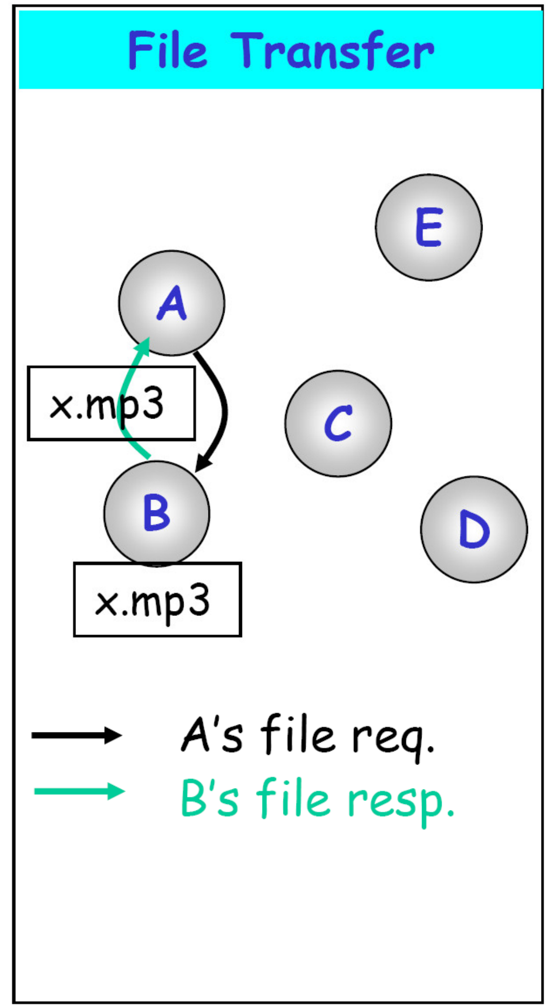
Gnutella

- Pure decentralized, unstructured
- Characteristic:
  - Few nodes with high connectivity.
  - Most nodes with sparse connectivity.
- Goal: distributed and anonymous file sharing
- Each application instance (node) :
  - stores/serves files
  - routes queries to its neighbors
  - responds to request queries

Gnutella (cont.)



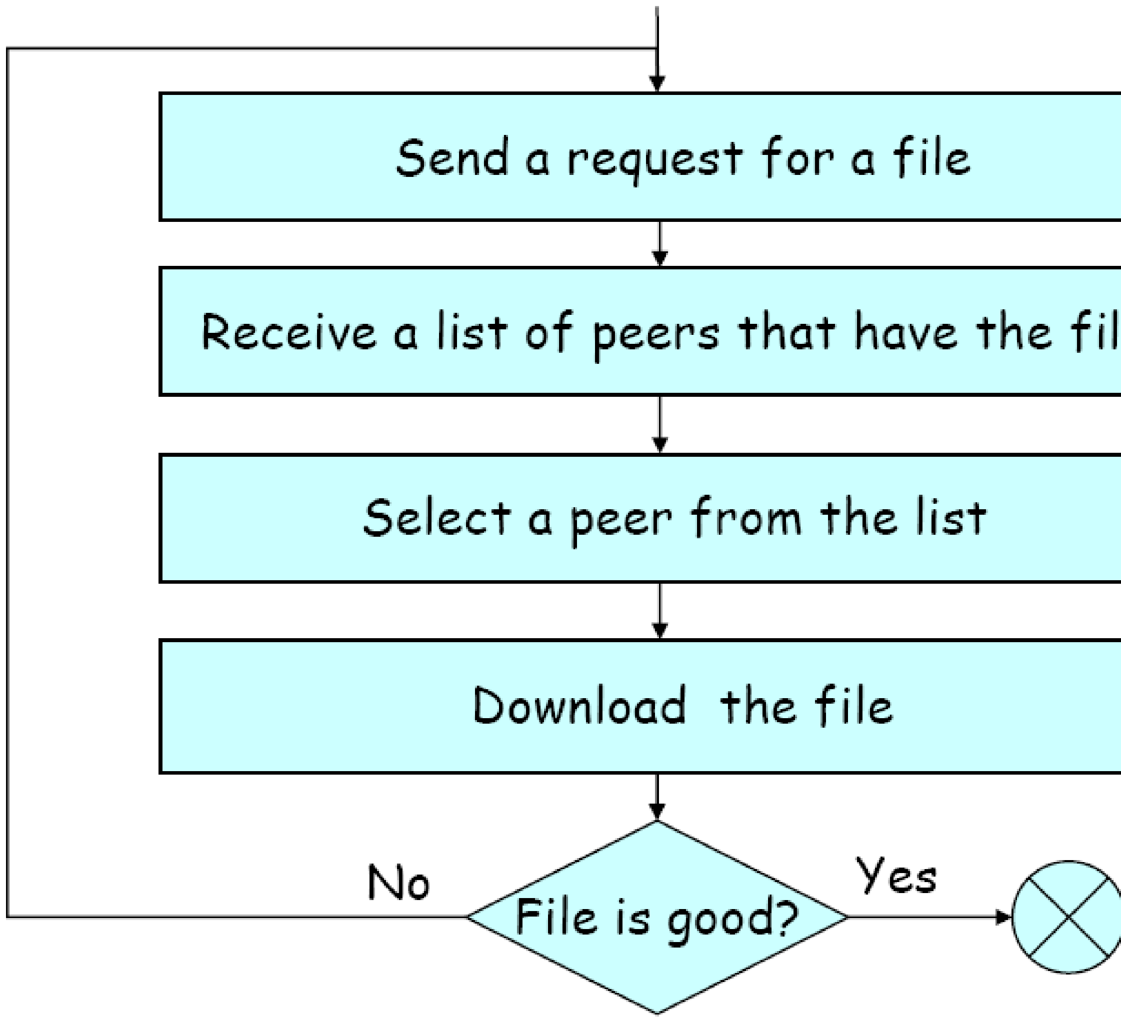




Gnutella (cont.)

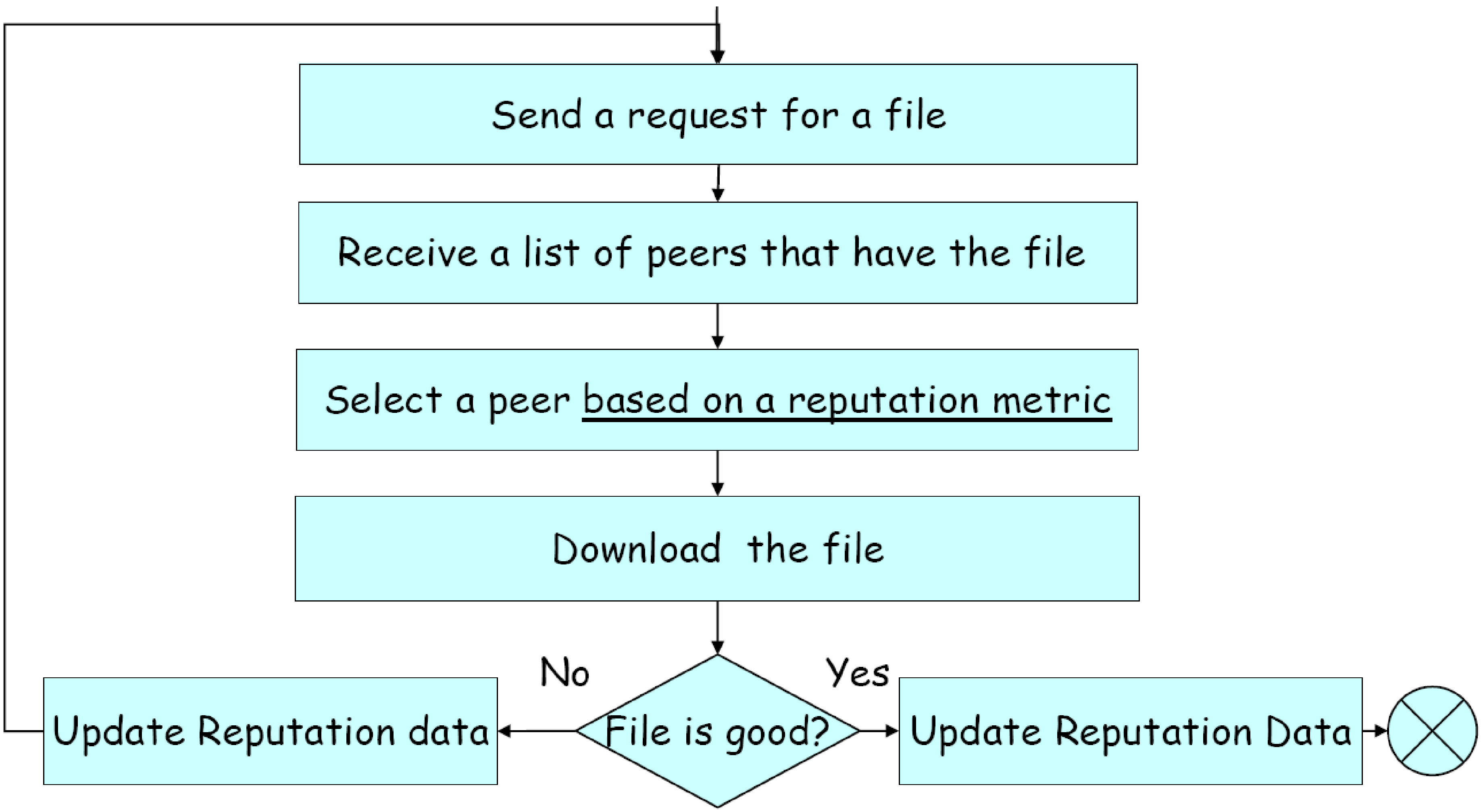
- Advantages:
  - Robustness to random node failure
  - Completeness (constrained by the TTL)
- Disadvantages:
  - Communication overhead
  - Network partition (controlled flooding)
  - Security

File Sharing in a P2P system



→ Need for a Reputation Management scheme

File Sharing in a Reputation-Based P2P system



Future Research Directions

- P2P research is an exciting area with many open problems and opportunities, including the design of:
  - New distributed object placement and query routing
  - New hash table data structures and algorithms
  - Efficient security and privacy
  - Semantic grouping of information in P2P networks
  - Incentive mechanisms and reputation systems
  - Convergence of Grid and P2P systems
  - Providing transactional and atomic guarantees on P2P

Fundamental Models

- Introduction
- Interaction Model
- Failure Model
- Security Model

Fundamental Models-Intro

- Fundamental Models are concerned with a more formal description of the properties that are common in all of the architectural models.

- All architectural models are composed of processes that communicate with each other by sending messages over a computer networks.

Fundamental Models-Intro

- Aspects of distributed systems that are discussed in fundamental models are:
  - Interaction model
    - Computation occurs within processes.
    - The processes interact by passing messages, resulting in:
      - Communication (information flow)
      - Coordination (synchronization and ordering of activities) between processes
    - Interaction model reflects the facts that communication takes place with delays.
  - Failure model
    - Failure model defines and classifies the faults.

Fundamental Models-Intro

- Security model
  - Security model defines and classifies the forms of attacks.
  - It provides a basis for analysis of threats to a system
  - It is used to design of systems that are able to resist threats.

Interaction Model

- Distributed systems are composed of many processes, interacting in the following ways:
  - Multiple server processes may cooperate with one another to provide a service
    - E.g. Domain Name Service
  - A set of peer processes may cooperate with one another to achieve a common goal
    - E.g. voice conferencing

Interaction Model

- Significant factors affecting interacting processes in a distributed system are:
  - Communication performance is often a limiting characteristic.
  - It is impossible to maintain a single global notion of time.

Interaction Model-Communication Channels

- Performance of communication channels
  - The communication channels in our model are realized in a variety of ways in distributed systems, for example
    - By an implementation of streams
    - By simple message passing over a computer network
  - Communication over a computer network has the performance characteristics such as:
    - Latency
      - The delay between the start of a message's transmission from one process to the beginning of its receipt by another.

Interaction Model-Communication Channels

- ❖ Bandwidth
  - The maximum amount of information that can be transmitted over a computer network in a given time.
  - Communication channels using the same network, have to share the available bandwidth.
- ❖ Jitter
  - The variation in the time taken to deliver a series of messages.
  - It is relevant to multimedia data.
    - ❑For example, if consecutive samples of audio data are played with differing time intervals, then the sound will be badly distorted.

Interaction Model-Computer Clock

- Computer clocks and timing events
  - Each computer in a distributed system has its own internal clock, which can be used by local processes to obtain the value of the current time.
  - Two processes running on different computers can associate timestamp with their events.
  - Even if two processes read their clock at the same time, their local clocks may supply different time.

Interaction Model-Computer Clock

- This is because computer clock drift from perfect time and their drift rates differ from one another.
- Clock drift rate refers to the relative amount that a computer clock differs from a perfect reference clock.
- Even if the clocks on all the computers in a distributed system are set to the same time initially, their clocks would eventually vary quite significantly unless corrections are applied.
- There are several techniques to correct time on computer clocks.
  - For example, computers may use radio signal receivers to get readings from GPS (Global Positioning System) with an accuracy about 1 microsecond.

Interaction Model-Variations

- Two variants of the interaction model
  - In a distributed system it is hard to set time limits on the time taken for process execution, message delivery or clock drift.
  - Two models of time assumption in distributed systems are:
    - Synchronous distributed systems
      - It has a strong assumption of time
      - The time to execute each step of a process has known lower and upper bounds.
      - Each message transmitted over a channel is received within a known bounded time.
      - Each process has a local clock whose drift rate from real time has a known bound.

Interaction Model

- Asynchronous distributed system
  - It has no assumption about time.
  - There is no bound on process execution speeds.
    - Each step may take an arbitrary long time.
  - There is no bound on message transmission delays.
    - A message may be received after an arbitrary long time.
  - There is no bound on clock drift rates.
    - The drift rate of a clock is arbitrary.

Interaction Model

- Event ordering
  - In many cases, we are interested in knowing whether an event (sending or receiving a message) at one process occurred before, after, or concurrently with another event at another process.
  - The execution of a system can be described in terms of events and their ordering despite the lack of accurate clocks.

Interaction Model

- ❖ For example, consider a mailing list with users X, Y, Z, and A.

1. User X sends a message with the subject Meeting.

2. Users Y and Z reply by sending a message with the subject RE: Meeting.

- In real time, X's message was sent first, Y reads it and replies; Z reads both X's message and Y's reply and then sends another reply, which references both X's and Y's messages.

- But due to the independent delays in message delivery, the messages may be delivered in the order is shown in figure 10.

- It shows user A might see the two messages in the wrong order.

(Figure 10)

Interaction Model

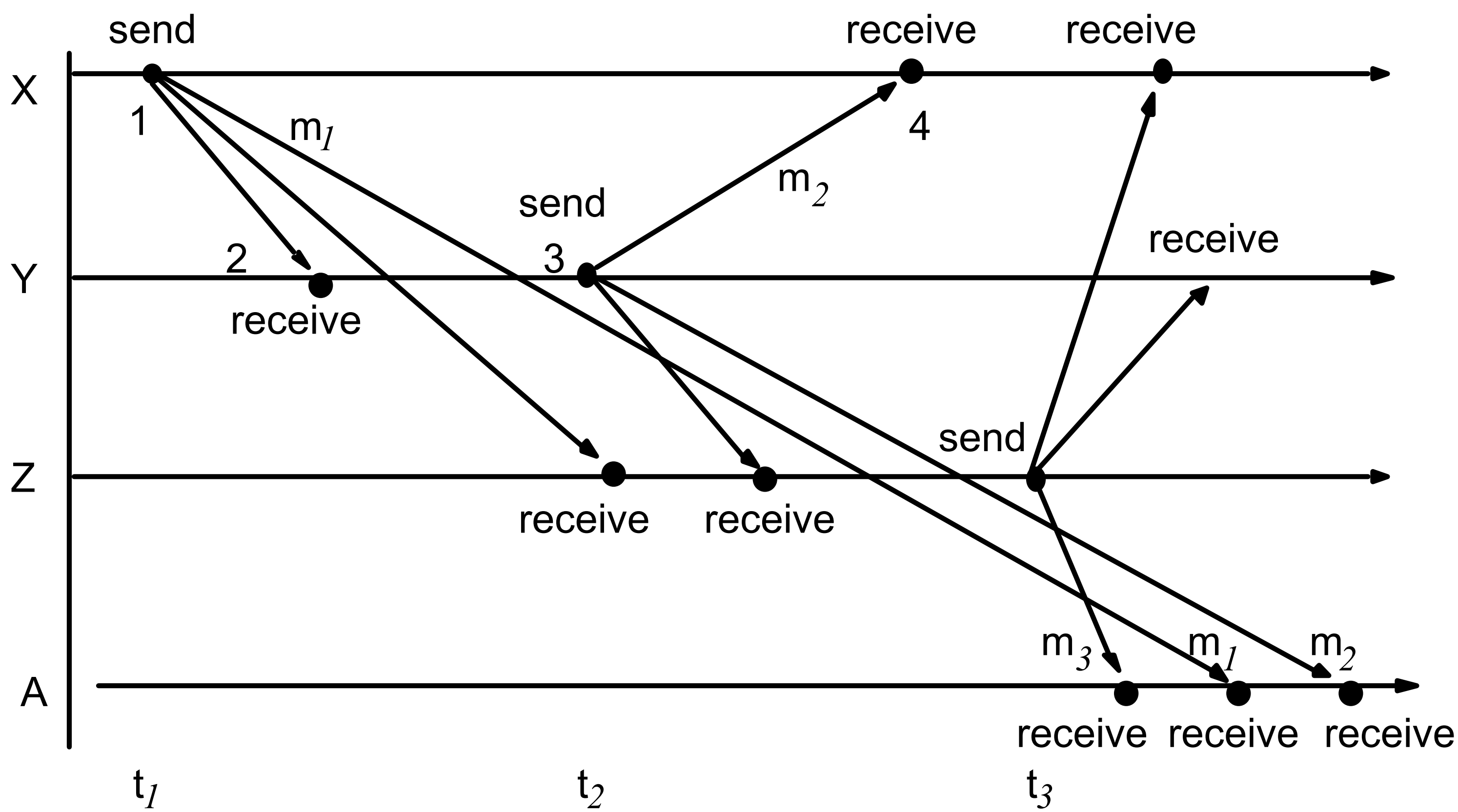


Figure 10. Real-time ordering of events.

Interaction Model

- Some users may view two messages in the wrong order, for example, user A might see
- *Item* is a sequence number that shows the order of receiving emails.

Item	*From*	*Subject*
23	Z	Re: Meeting
24	X	Meeting
26	Y	Re: Meeting

Failure Model

- In a distributed system both processes and communication channels may fail – That is, they may depart from what is considered to be correct or desirable behavior.

- Types of failures:
  - Omission Failures
  - Arbitrary Failures
  - Timing Failures

Failure Model

- Omission failure
  - Omission failures refer to cases when a process or communication channel fails to perform actions that it is supposed to do.
  - The chief omission failure of a process is to crash. In case of the crash, the process has halted and will not execute any further steps of its program.
  - Another type of omission failure is related to the communication which is called communication omission failure shown in Figure 11.

Failure Model

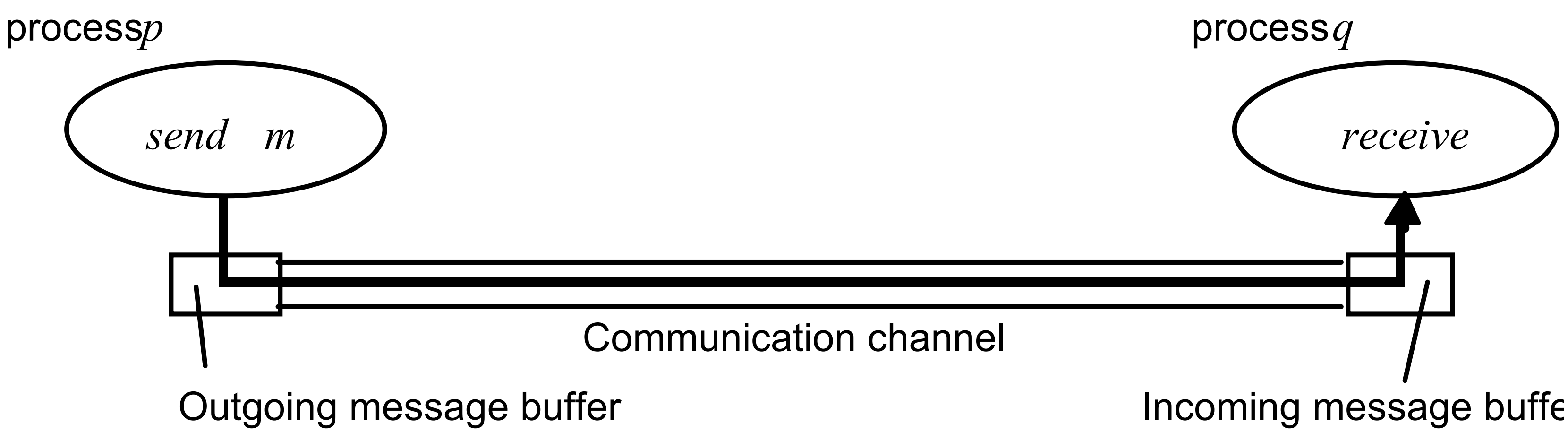


Figure 11. Processes and channels.

- The communication channel produces an omission failure if it does not transport a message from "p"s outgoing message buffer to "q"'s incoming message buffer.

- This is known as "dropping messages" and is generally caused by lack of buffer space at the receiver or at a gateway or by a network transmission error, detected by a checksum carried with the message data.

Failure Model

- Arbitrary failure
  - Arbitrary failure is used to describe the worst possible failure semantics, in which any type of error may occur.
    - E.g. a process may set a wrong values in its data items, or it may return a wrong value in response to an invocation.
  - Communication channel can suffer from arbitrary failures.
    - E.g. message contents may be corrupted or non-existent messages may be delivered or real messages may be delivered more than once.

Failure Model

- The omission failures are classified together with arbitrary failures shown below

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send,* but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (complex)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Failure Model

- Timing failure
  - Timing failures are applicable in synchronized distributed systems where time limits are set on process execution time, message delivery time and clock drift rate.

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Failure Model

- Masking failure
  - It is possible to construct reliable services from components that exhibit failure.
    - E.g. multiple servers that hold replicas of data can continue to provide a service when one of them crashes.
  - A service masks a failure, either by hiding it altogether or by converting it into a more acceptable type of failure.
    - E.g. checksums are used to mask corrupted messages- effectively converting an arbitrary failure into an omission failure.

Security Model

- The security of a distributed system can be achieved by securing the processes and the channels used in their interactions.

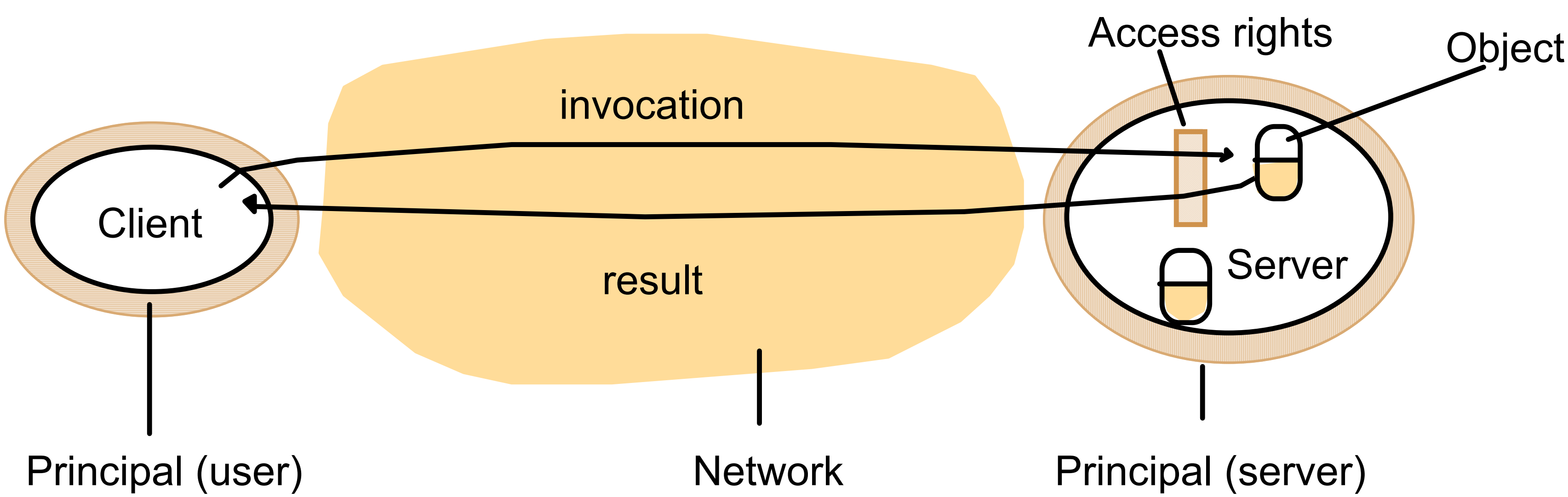
- Also, by protecting the objects that they encapsulate against unauthorized access.

Security Model

- Protecting Objects
  - Access rights
    - Access rights specify who is allowed to perform the operations on an object.
      - Who is allowed to read or write its state.
  - Principal
    - Principal is the authority associated with each invocation and each result.
    - A principal may be a user or a process.
    - The invocation comes from a user and the result from a server.

Security Model

- The sever is responsible for
  - Verifying the identity of the principal (user) behind each invocation.
  - Checking that they have sufficient access rights to perform the requested operation on the particular object invoked.
  - Rejecting those that do not.



Security Model

- Other possible threats from an enemy
  - Denial of service
    - This is a form of attack in which the enemy interferes with the activities of authorized users by making excessive and pointless invocations on services of message transmissions in a network.
    - It results in overloading of physical resources (network bandwidth, server processing capacity).

Security Model

- Mobile code
  - Mobile code is security problem for any process that receives and executes program code from elsewhere, such as the email attachment.
  - Such attachment may include a code that accesses or modifies resources that are available to the host process but not to the originator of the code.