DTU Compute
Department of Applied Mathematics and Computer Science

Introduction to P2P Computing

Nicola Dragoni
Embedded Systems Engineering
DTU Compute

1. Introduction
   A. Peer-to-Peer vs. Client/Server
   B. Overlay Networks
2. Common Topologies
3. Data location
4. Gnutella Protocol

From the First Lecture (Architectural Models)...

- The architecture of a system is its structure in terms of separately specified components and their interrelationships

- 4 **fundamental building blocks** (and 4 key questions):
  - **Communicating entities**: *what are the entities that are communicating in the distributed system?*
  - **Communication paradigms**: *how do these entities communicate, or, more specifically, what communication paradigm is used?*
  - **Roles and responsibilities**: *what (potentially changing) roles and responsibilities do these entities have in the overall architecture?*
  - **Placement**: *how are these entities mapped on to the physical distributed infrastructure (i.e., what is their placement)?*

To Avoid Any Misunderstanding…

P2P is more than just
Pirate-to-Pirate
file-sharing!
&
distributing
illegal copies

Introduction

- Peer-to-peer (P2P) systems have become extremely popular and contribute to vast amounts of Internet traffic

- P2P basic definition:
  *A P2P system is a distributed collection of **peer** nodes, that act both as servers and as clients*
  - provide services to other peers
  - consume services from other peers

- Very different from the client-server model!!

It's a Broad Area…

- P2P file sharing
  - Gnutella
  - eMule
  - BitTorrent
- P2P communication
  - Instant messaging
  - Voice-over-IP: Skype
- DHTs & their apps
  - Chord, CAN, Kademlia, …
- P2P wireless
  - Ad-hoc networking
- P2P computation
  - Seti@home

P2P History: 1969 - 1990

- The **origins**:
  - In the beginning, all nodes in Arpanet/Internet were peers
  - Each node was capable of:

✓ Performing routing (locate machines)

✓ Accepting ftp connections (file sharing)

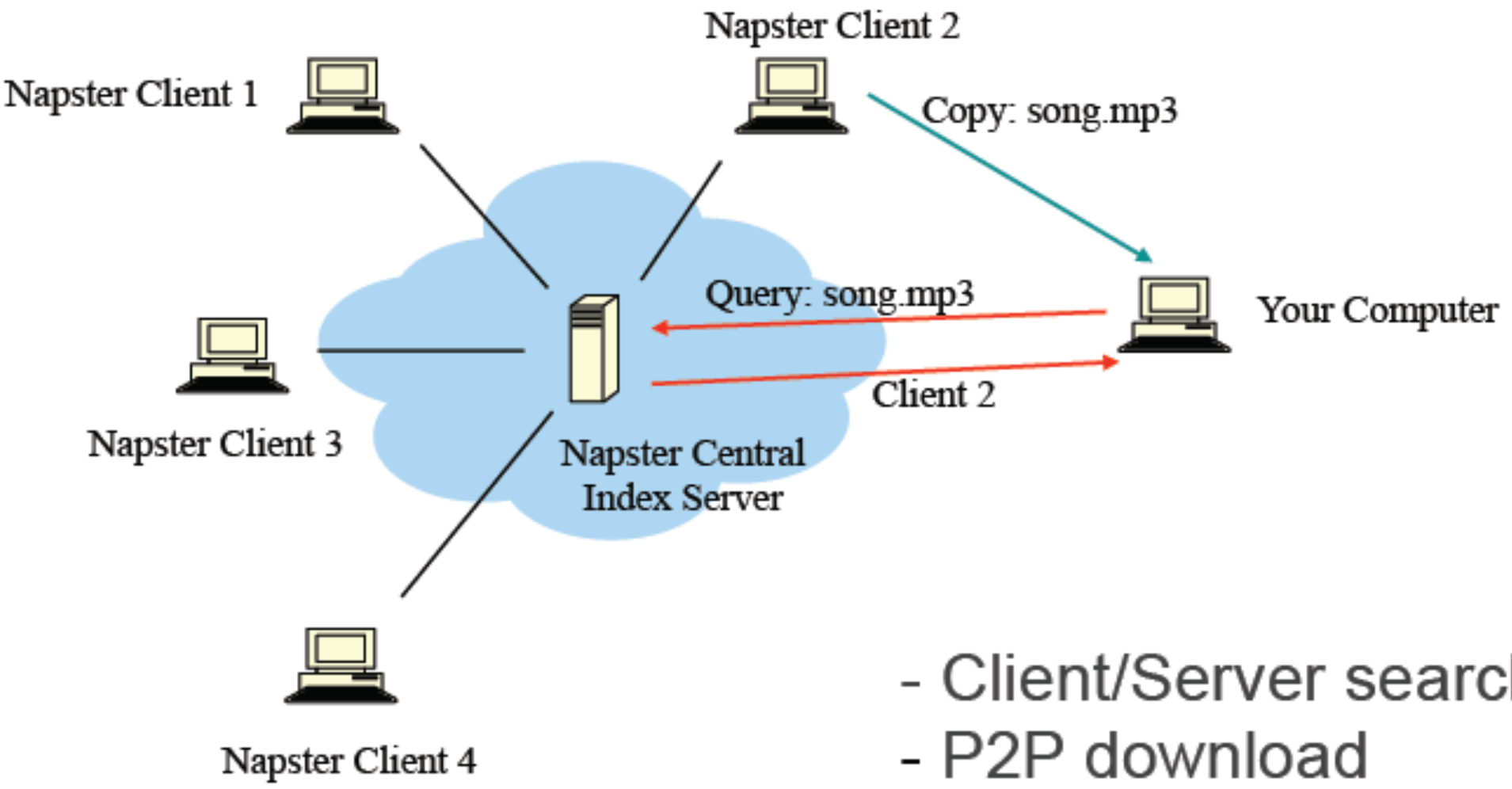
✓ Accepting telnet connections (distribution computation)

P2P History: 1999 - Today

- The advent of **Napster**:
  - Jan 1999: the first version of Napster was released by Shawn Fanning, ***student*** at the Northeastern University
  - July 1999: Napster Inc. founded
  - Feb 2001: Napster closed down

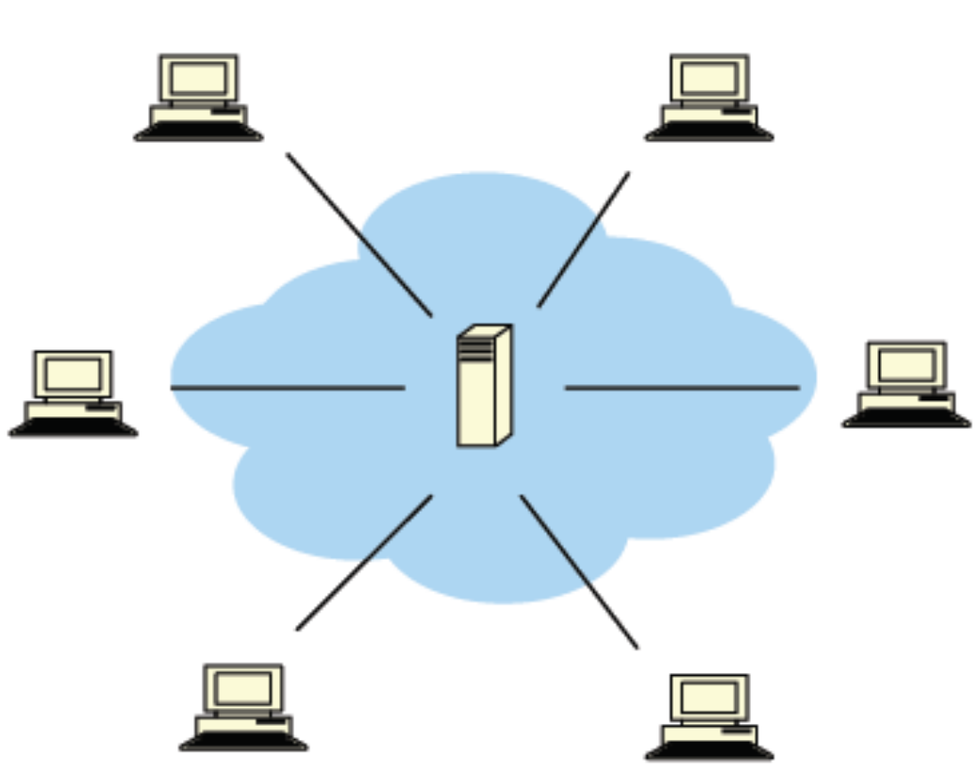
- After Napster:
  - Gnutella, KaZaa, BitTorrent, …
  - Skype
  - Content creation in Wikipedia
  - Open-source software development
  - Crowd-sourcing
  - …

Napster

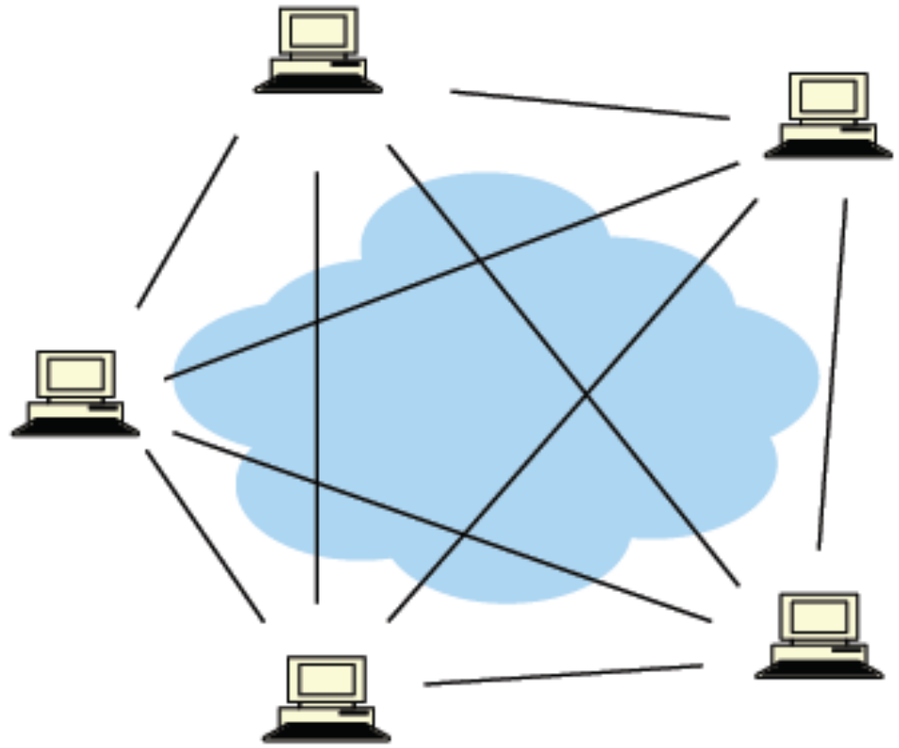


- Client/Server search
- P2P download
- Napster is not “pure P2P”

Client/Server vs. Peer-to-Peer

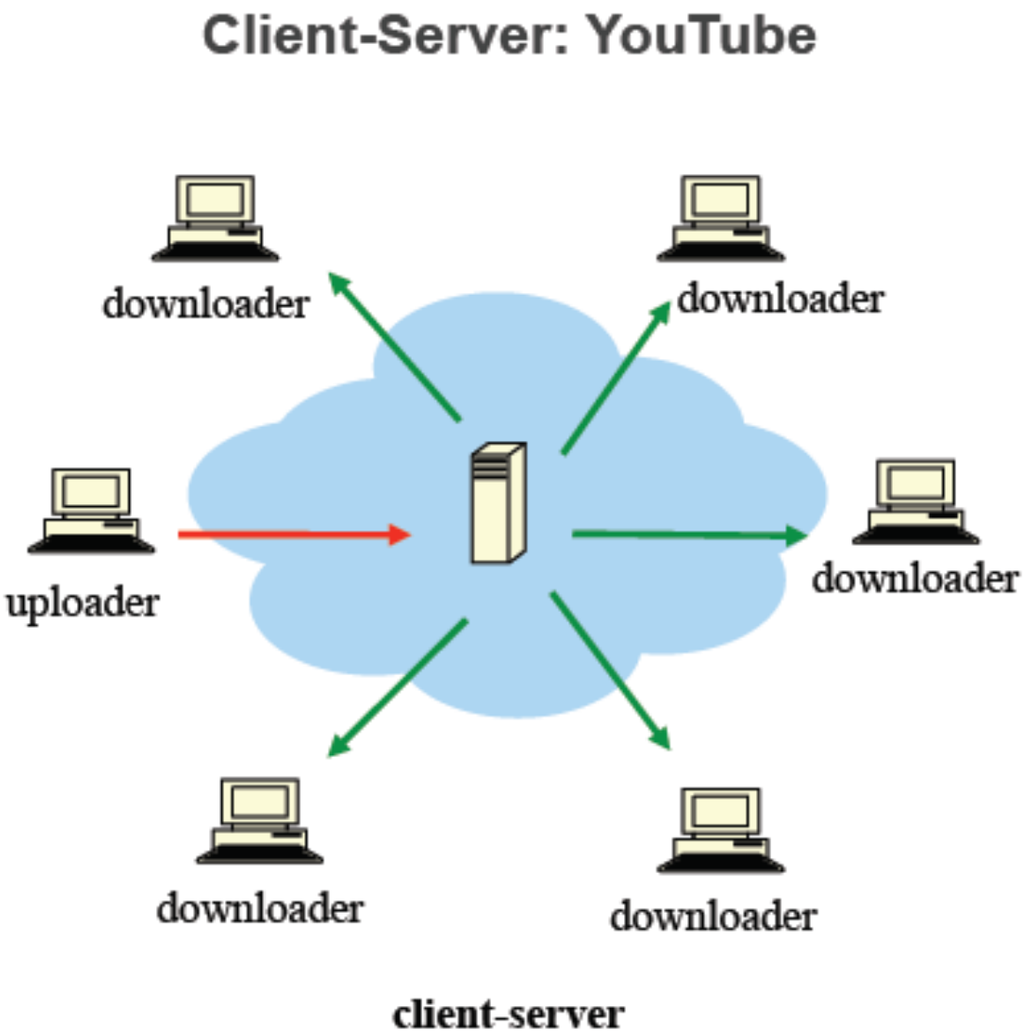


- Servers well connected to the "core" of the Internet
- Servers carry out critical tasks
- Clients only talk to servers



- Only nodes located at the "periphery of the Internet"
- Tasks distributed across all nodes
- Clients talk to other clients

Example – Video Sharing (YouTube vs BitTorrent)



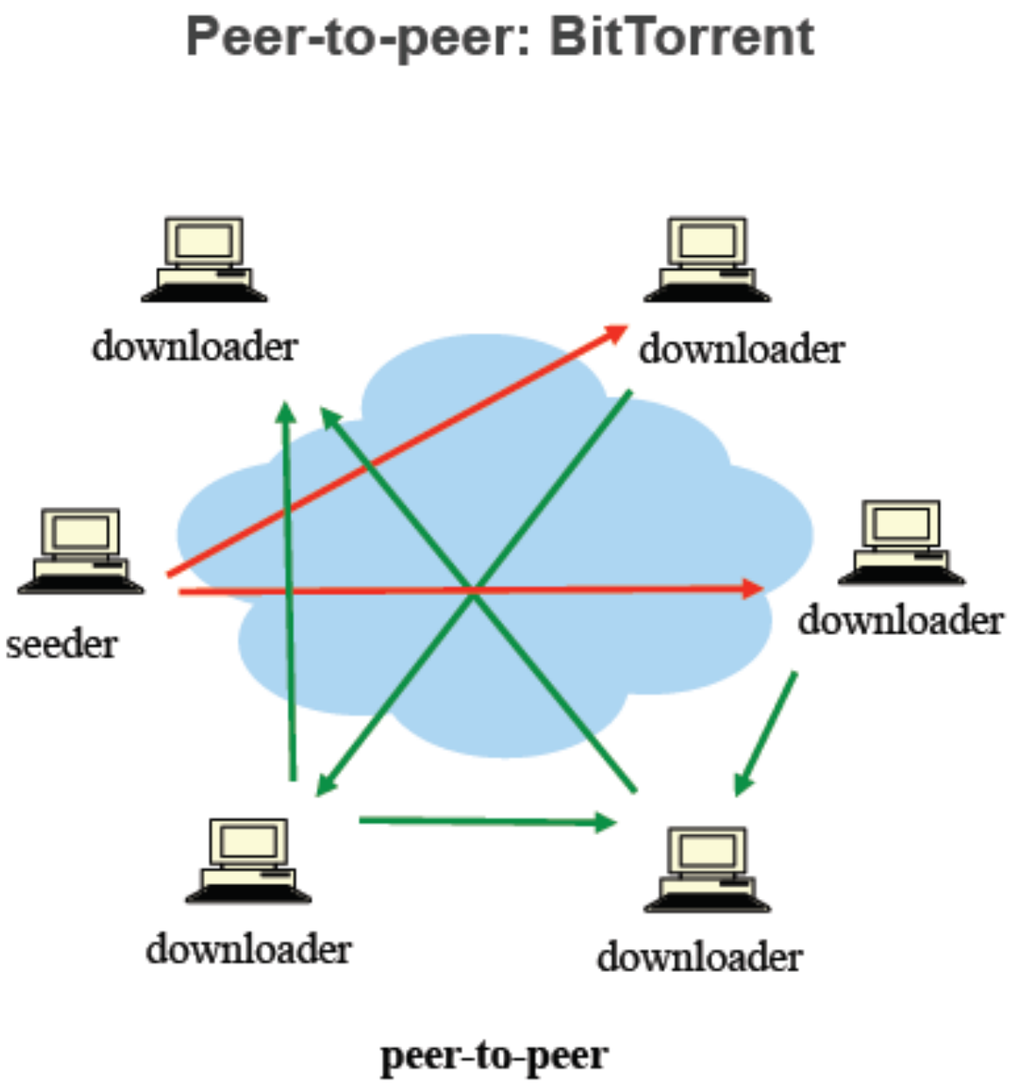
Advantages

- Client can disconnect after upload
- Uploader needs little bandwidth
- Other users can find the file easily (just use search on server webpage)

Disadvantages

- Server may not accept file or remove it later (according to content policy)
- Whole system depends on the server (what if shut down like Napster?)
- Server storage and bandwidth are expensive!

Example – Video Sharing (YouTube vs BitTorrent)



Advantages

- Does not depend on a central server
- Bandwidth shared across nodes (downloaders also act as uploaders)
- High scalability, low cost

Disadvantages

- Seeder must remain on-line to guarantee file availability
- Content is more difficult to find (downloaders must find .torrent file)
- Freeloaders cheat in order to download without uploading

P2P vs Client-Server

Client-server	Peer-to-peer
Asymmetric: client and servers carry out different tasks	**Symmetric**: No distinction between node; they are *peers*
Global knowledge: servers have a global view of the network	**Local knowledge**: nodes only know a small set of other nodes
Centralization: communications and management are centralized	**Decentralization**: no global knowledge, only local interactions
Single point of failure: a server failure brings down the system	**Robustness**: several nodes may fail with little or no impact
Limited scalability: servers easily overloaded	**High scalability**: high aggregate capacity, load distribution
Expensive: server storage and bandwidth capacity is not cheap	**Low-cost**: storage and bandwidth are contributed by users

P2P Environment

- Dynamic
  - Nodes may disconnect temporarily
  - New nodes are continuously joining the system, while others leave permanently
- Security
  - P2P clients runs on machines under the total control of their owners
  - Malicious users may try to bring down the system
- Selfishness
  - Users may run hacked clients in order to avoid contributing resources

Why P2P?

- Decentralisation enables deployment of applications that are:
  - Highly available
  - Fault-tolerant
  - Self-organizing
  - Scalable
  - Difficult or impossible to shut down
- This results in a “democratisation” of the Internet

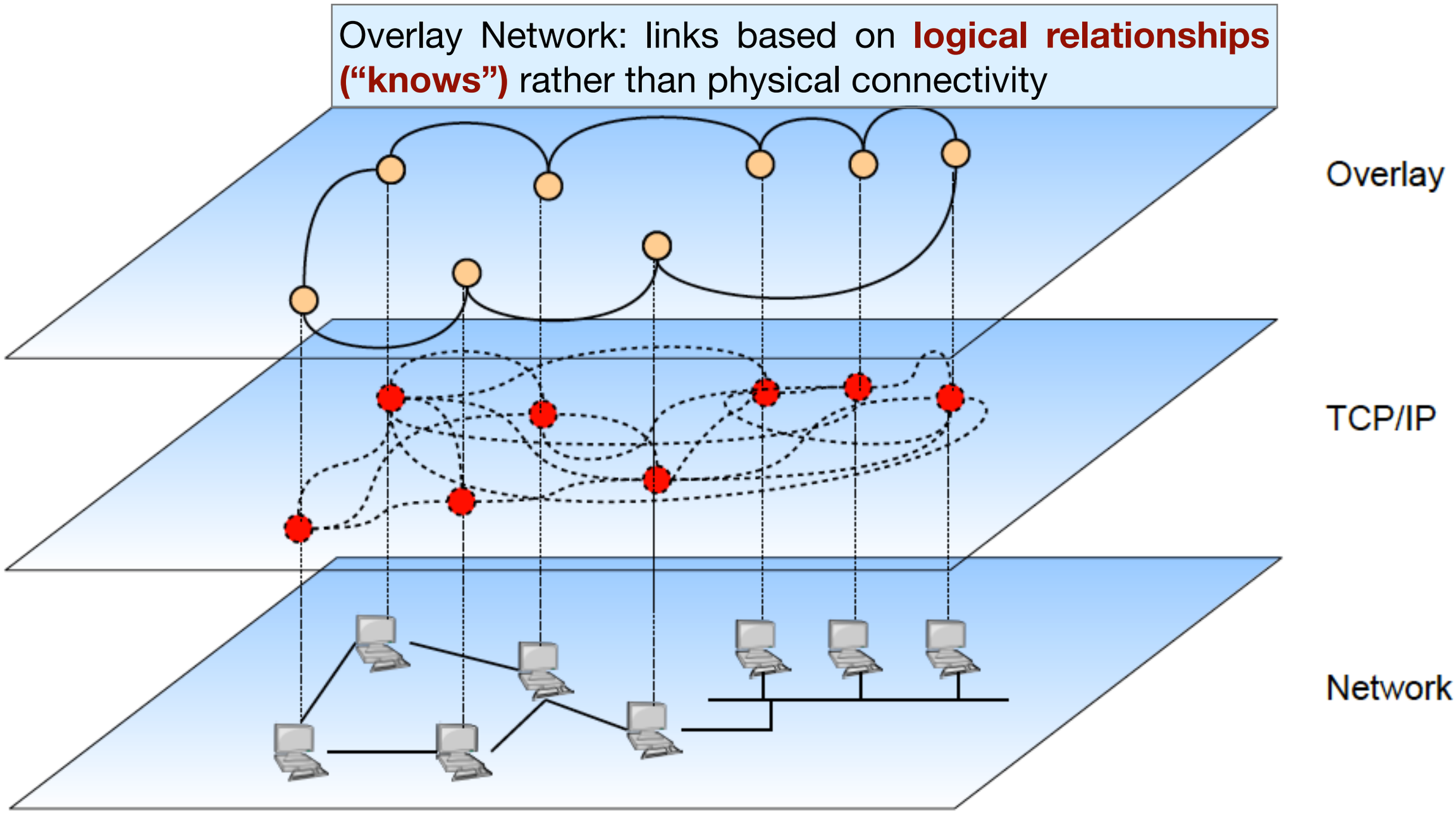
P2P and Overlay Networks

- Peer-to-Peer networks are usually "**overlays**"

- **Logical structures** built on top of a physical routed communication infrastructure (IP) that creates the allusion of a completely-connected graph

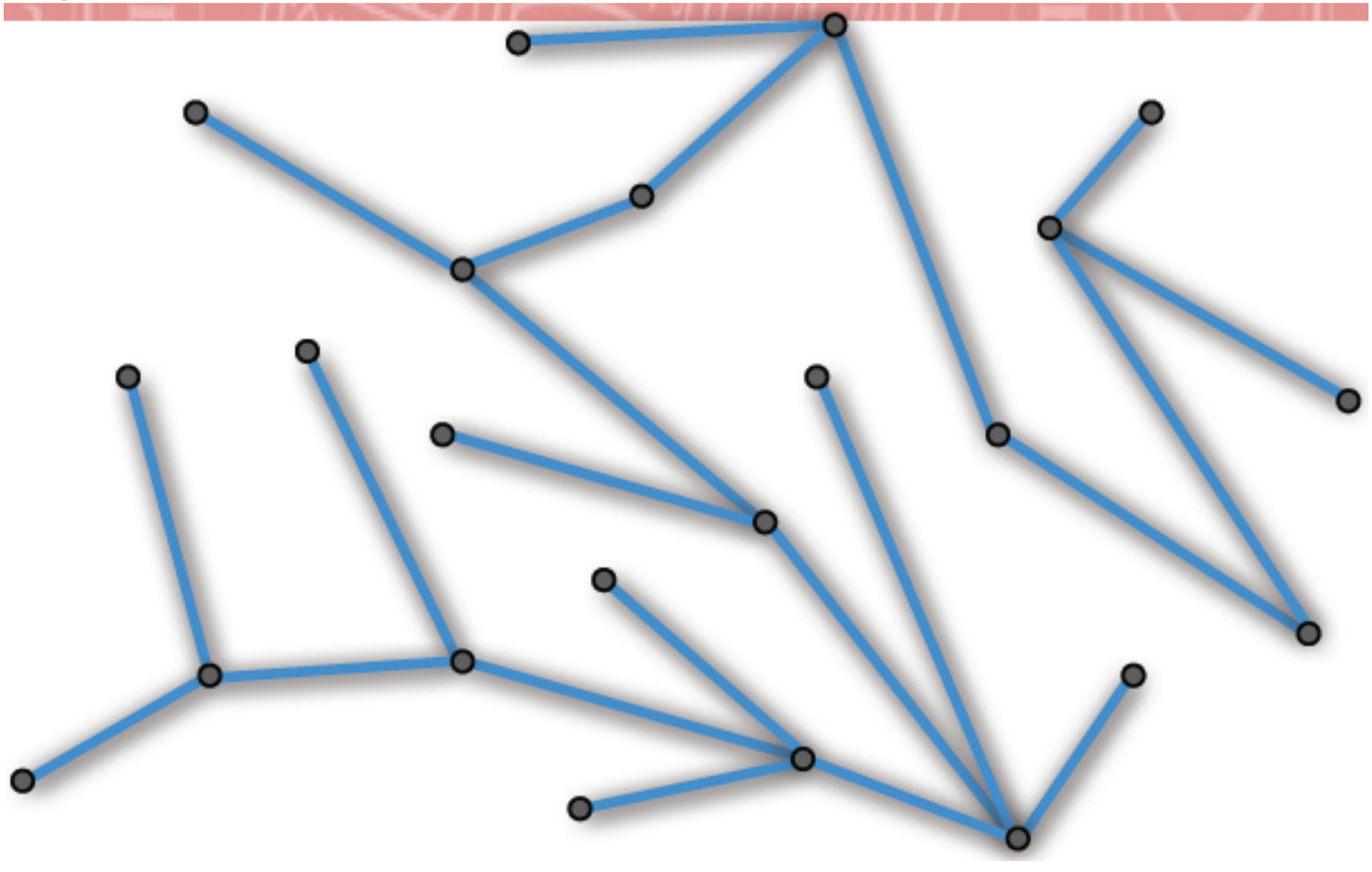
An **overlay network** is a virtual network of nodes and logical links that is built on top of an existing network with the purpose to implement a network service that is not available in the existing network

Overlay Networks



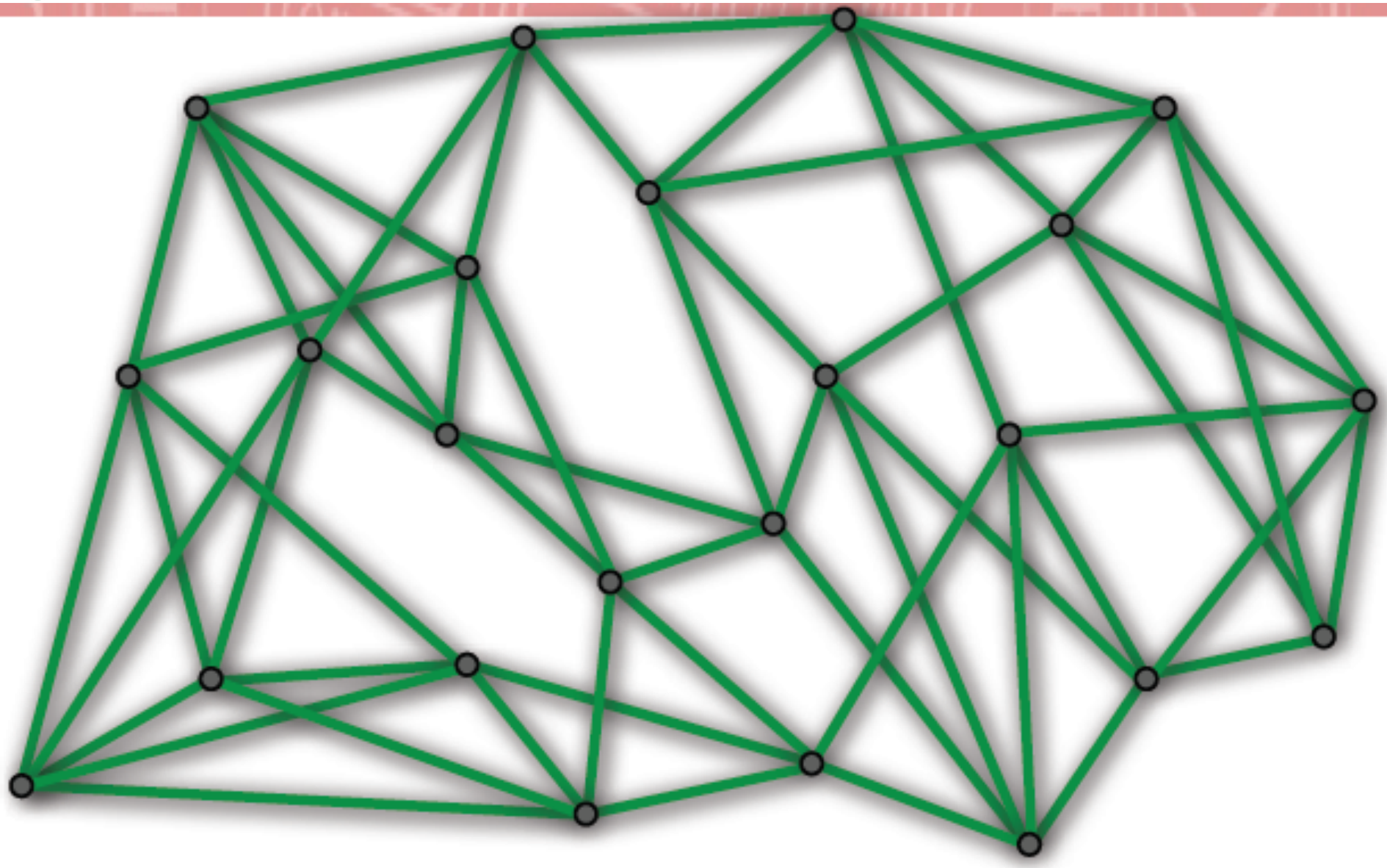
Overlay Networks

Physical network: “who has a communication link to whom”



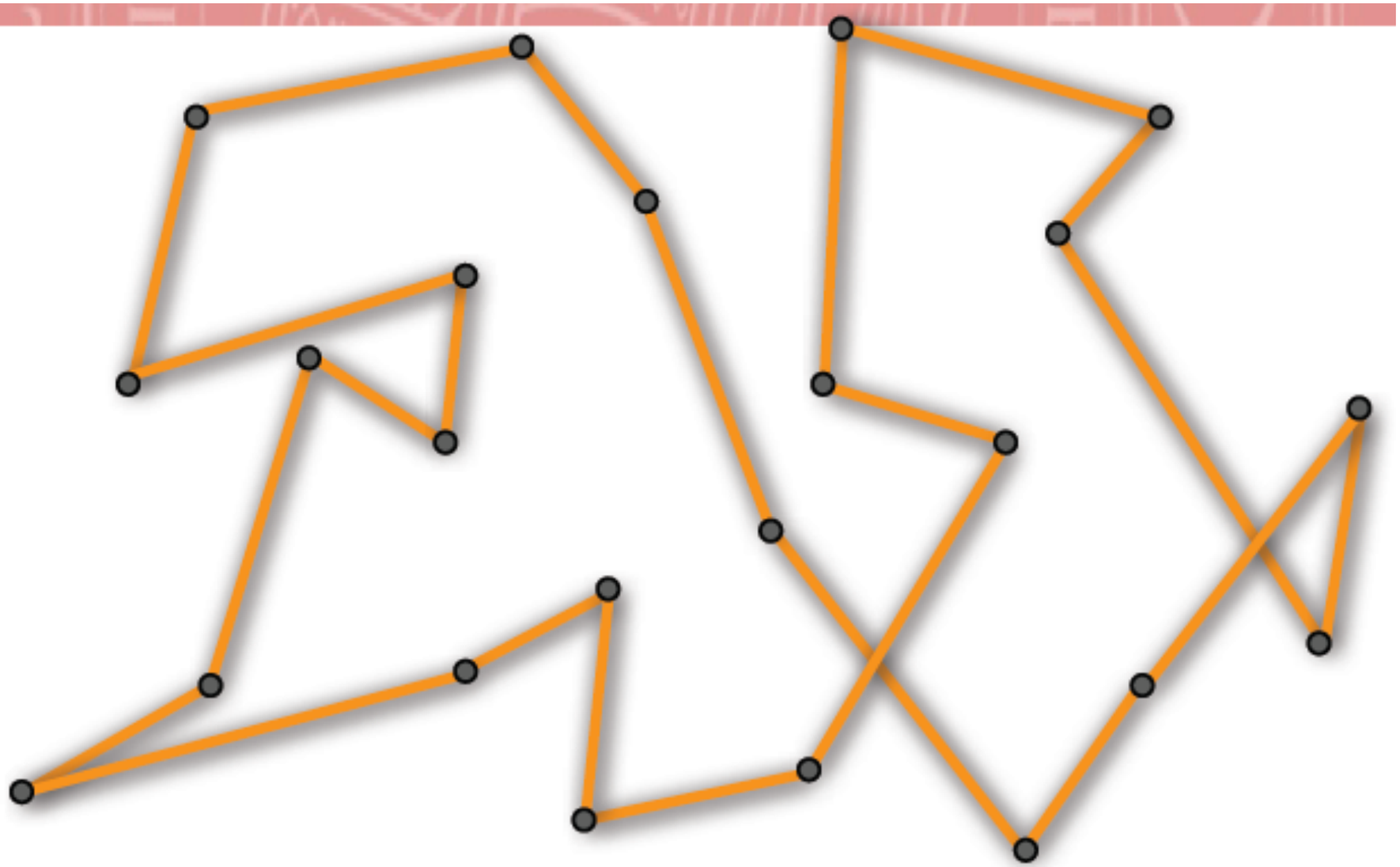
Overlay Networks

Logical network: “who can communicate with whom”



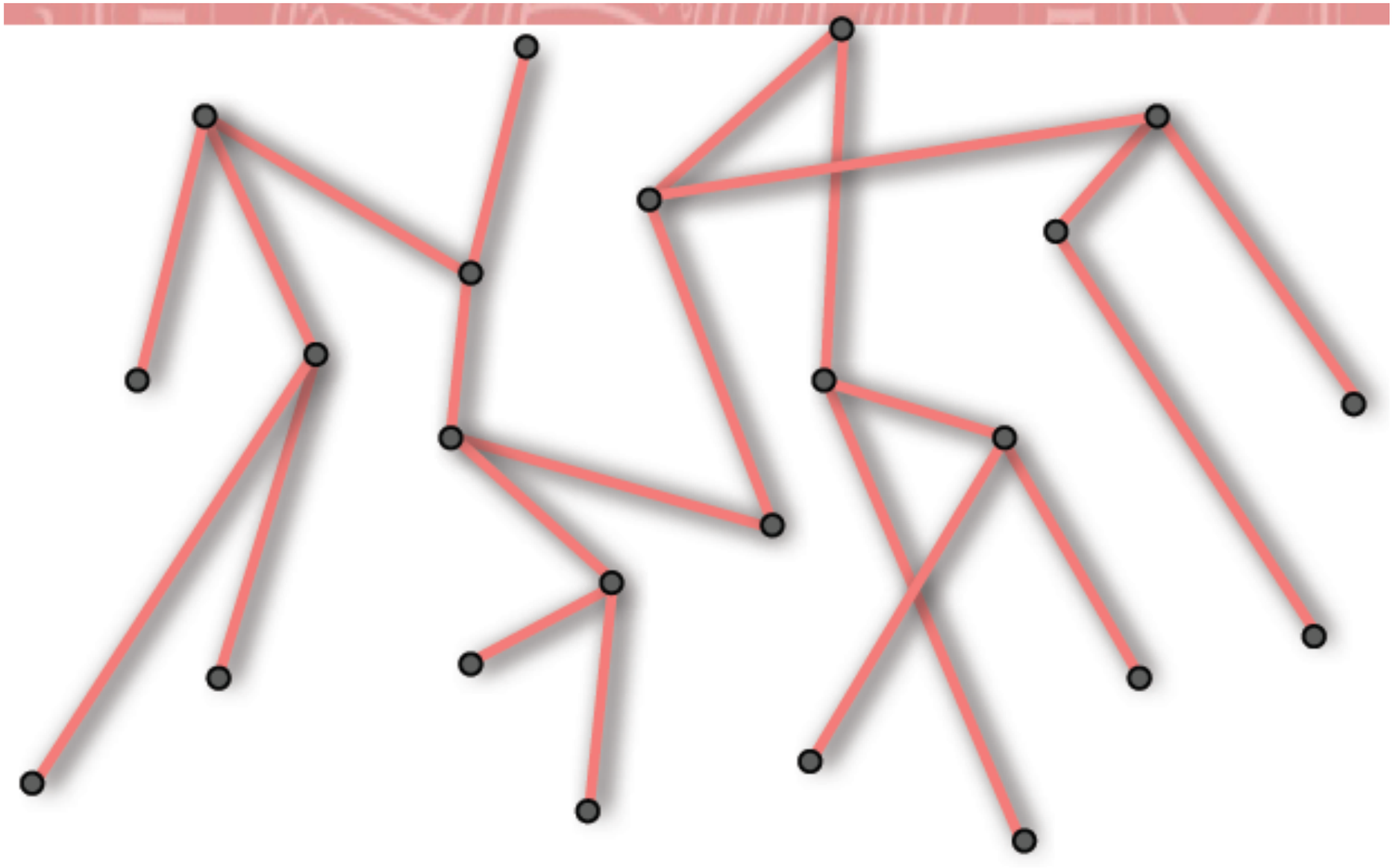
Overlay Networks

Overlay network (ring): “who knows whom”



Overlay Network

Overlay network (tree): “who knows whom”

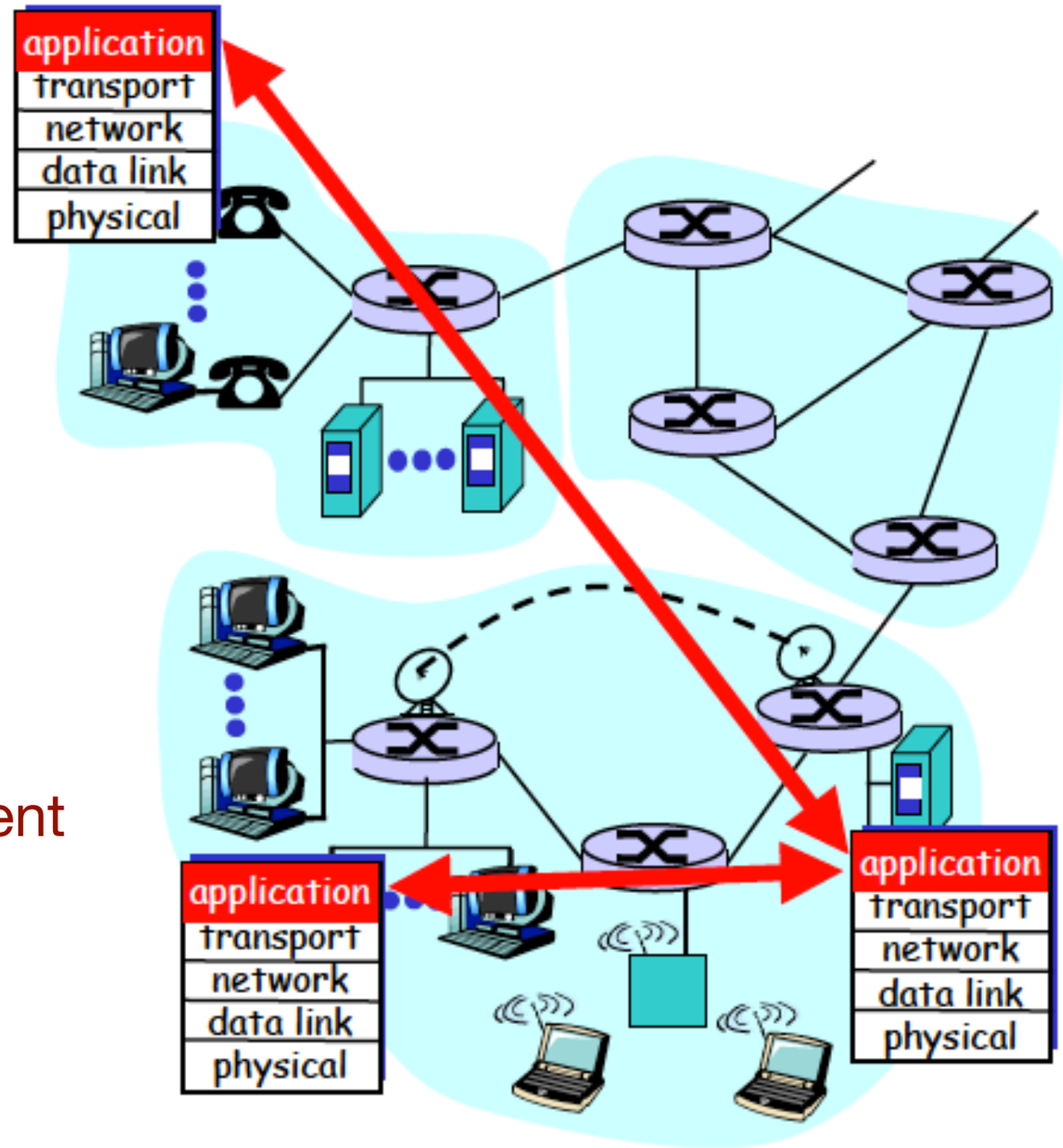


Overlay Networks

- Virtual edge
  - TCP connection
  - or simply a pointer to an IP address
- Overlay maintenance
  - Periodically ping to make sure neighbour is still alive
  - Or verify liveness while messaging
  - If neighbour goes down, may want to establish new edge

Overlay Networks

- Tremendous design flexibility
  - Topology
  - Message types
  - Protocols
  - Messaging over TCP or UDP
- Underlying physical net is transparent to developer

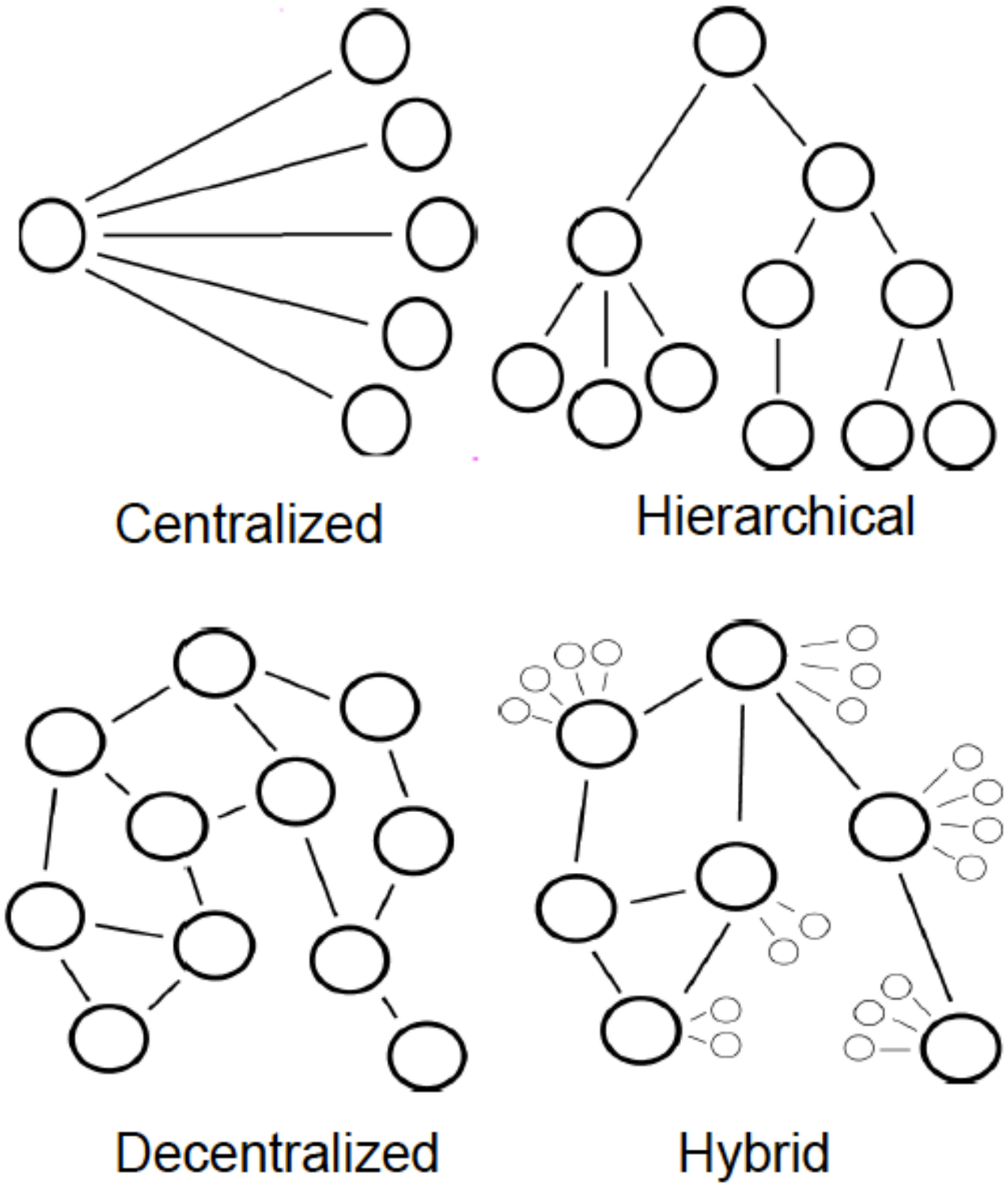


P2P Problems

- Overlay construction and maintenance
  - e.g., random, two-level, ring, etc.
- Data location
  - locate a given data object among a large number of nodes
- Data dissemination
  - propagate data in an efficient and robust manner
- Per-node state
  - keep the amount of state per node small
- Tolerance to churn (dynamic system)
  - maintain system invariants (e.g., topology, data location, data availability) despite node arrivals and departures

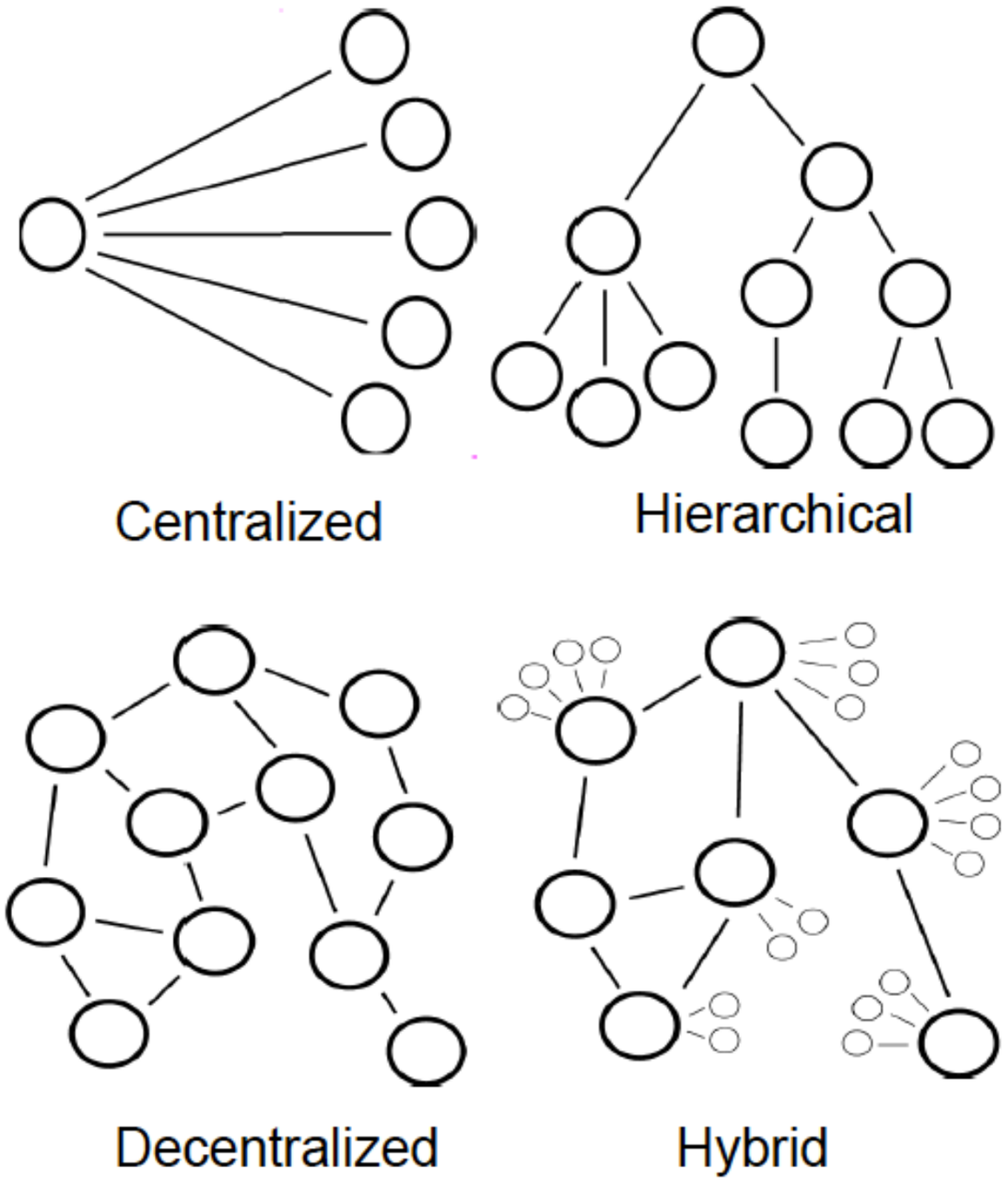
P2P Topologies

Overlay Topologies



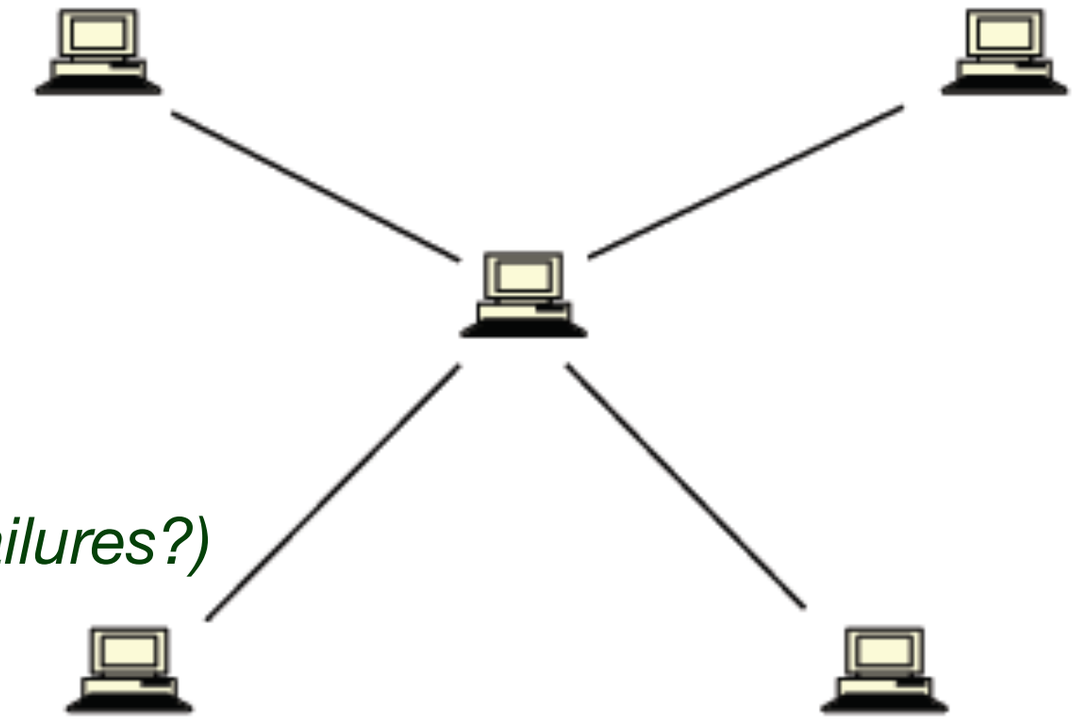
Evaluating Topologies

- Manageability
  - *How hard is it to keep working?*
- Information coherence
  - *How reliable is info?*
- Extensibility
  - *How easy is it to grow?*
- Fault tolerance
  - *How well can it handle failures?*
- Censorship
  - *How hard is it to shut down?*



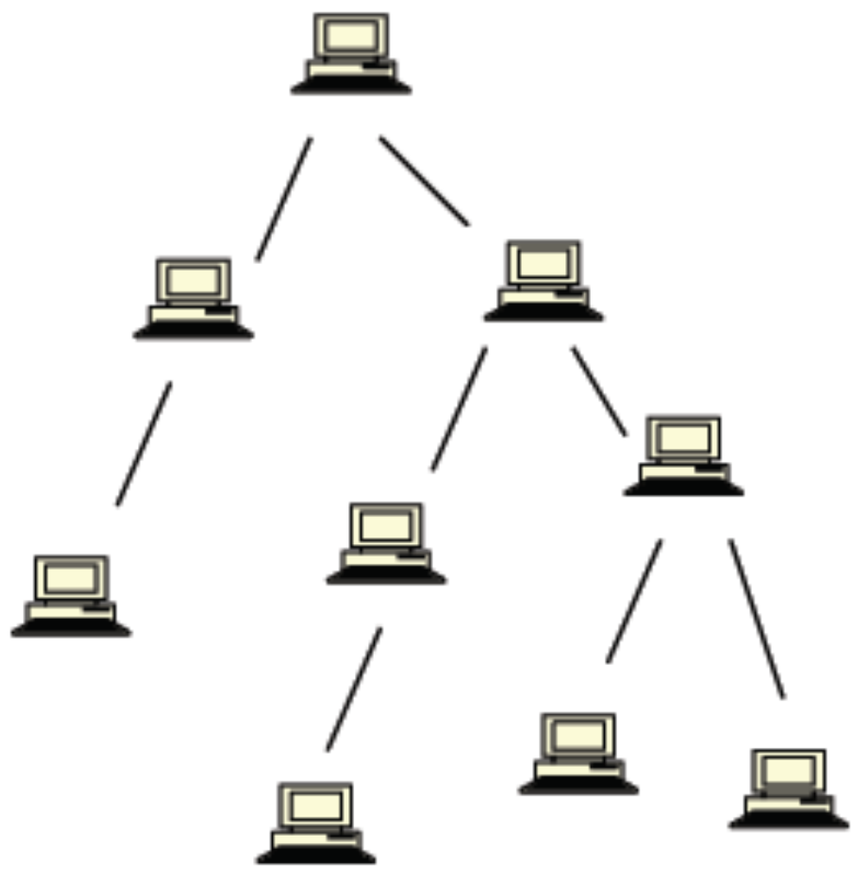
Evaluating Topologies: Centralized

- Manageable *(how hard is it to keep working?)*
  - System is all in one place
- Coherent *(how reliable is info?)*
  - Information is centralized
- Extensible *(how easy is it to grow?)*
  - No
- Fault tolerance *(how well can it handle failures?)*
  - Single point of failure
- Censorship *(how hard is it to shut down?)*
  - Easy to shut down



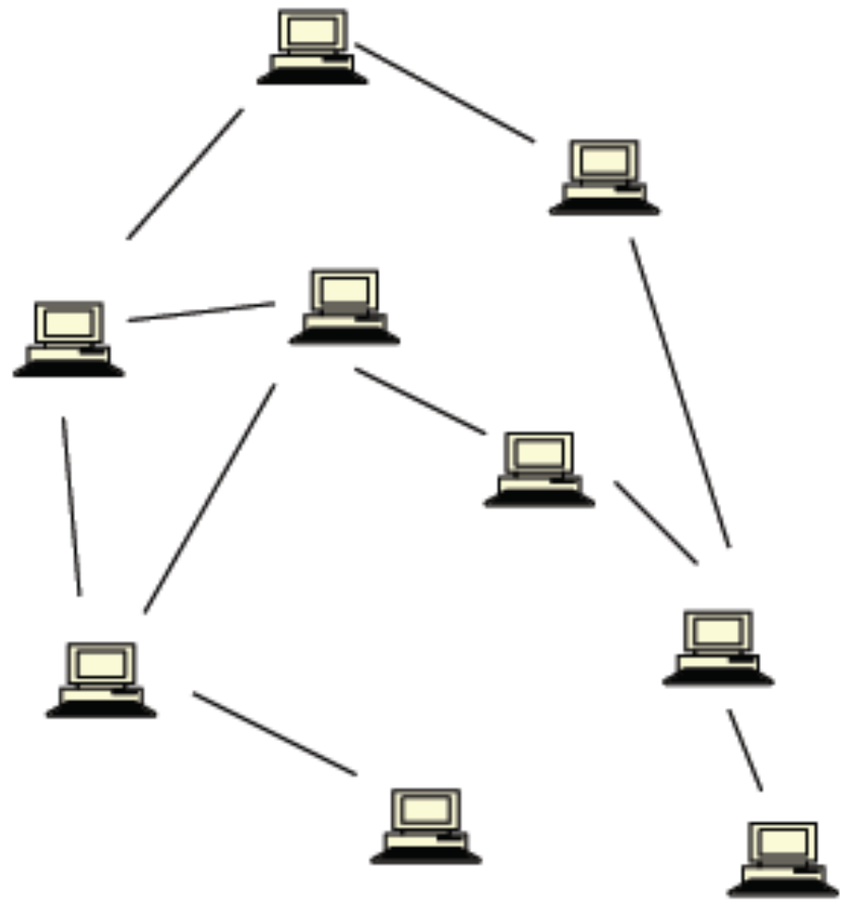
Evaluating Topologies: Hierarchical

- Manageable *(how hard is it to keep working?)*
  - Chain of authority
- Coherent *(how reliable is info?)*
  - Cache consistency
- Extensible *(how easy is it to grow?)*
  - Add more leaves, rebalance
- Fault tolerance *(how well can it handle failures?)*
  - Root is vulnerable
- Censorship *(how hard is it to shut down?)*
  - Just shut down the root



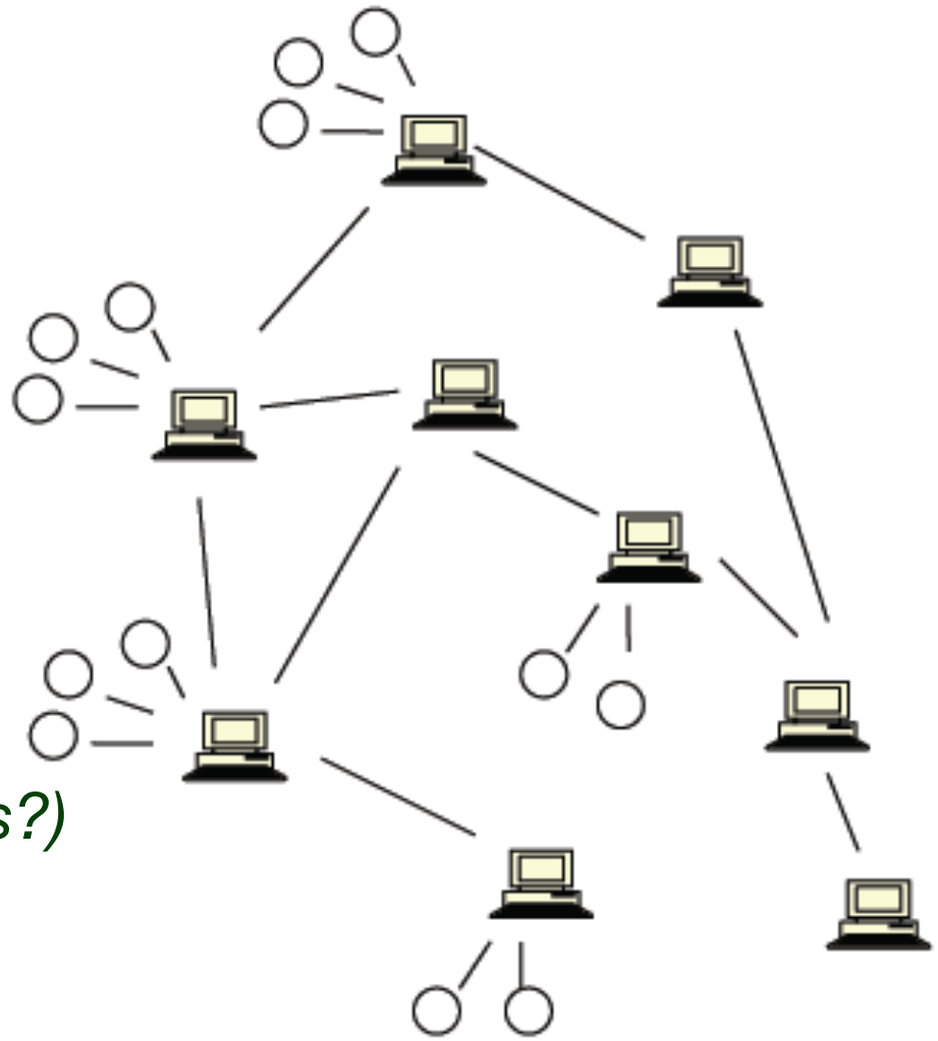
Evaluating Topologies: Decentralized

- Manageable *(how hard is it to keep working?)*
  - Difficult, many owners
- Coherent *(how reliable is info?)*
  - Difficult, unreliable peers
- Extensible *(how easy is it to grow?)*
  - Anyone can join in
- Fault tolerance *(how well can it handle failures?)*
  - Redundancy
- Censorship *(how hard is it to shut down?)*
  - Difficult to shut down



Evaluating Topologies: Centralized + Decentralized

- Manageable *(how hard is it to keep working?)*
  - Same as decentralized
- Coherent *(how reliable is info?)*
  - Better than decentralized
- Extensible *(how easy is it to grow?)*
  - Anyone can join in
- Fault tolerance *(how well can it handle failures?)*
  - Redundancy
- Censorship *(how hard is it to shut down?)*
  - Difficult to shut down



Searching VS Addressing

- Two basic ways to find objects:
  - **Search** for them
  - **Address** them using their **unique name**
- Difference between searching and addressing is **fundamental**
  - Determines how network is constructed
  - Determines how objects are placed
  - Determines efficiency of object location

Searching VS Addressing

- **"Addressing" networks**: find objects by *addressing them with their unique name* (cf. URLs in Web)
- **"Searching" networks**: find objects by *searching with keywords that match objects's description* (cf. Google)

Addressing

- Pros:
  - Each object uniquely identifiable
  - Object location can be made efficient
- Cons:
  - Need to know unique name
  - Need to maintain structure required by addresses

Searching

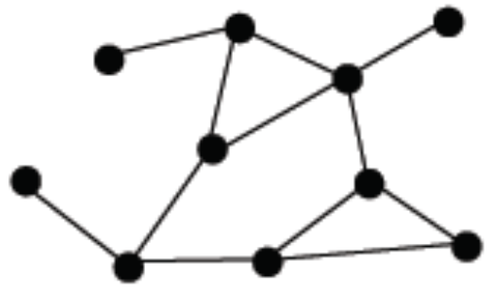
- Pros:
  - No need to know unique names
    - More user friendly
- Cons:
  - Hard to make efficient
    - Can solve with money, see Google
  - Need to compare actual objects to know if they are same

Unstructured VS Structured P2P Networks

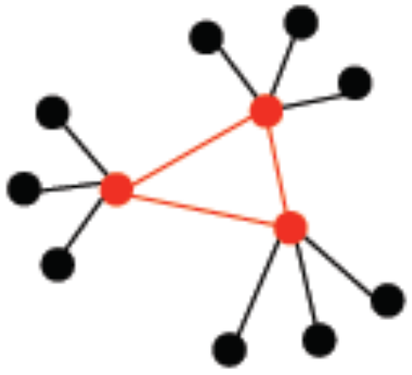
- **Unstructured networks**
  - Based on **searching**
  - Unstructured does **NOT** mean complete lack of structure
  - *Network has structure, but peers are free to join anywhere and objects can be stored anywhere*
- **Structured networks**
  - Based on **addressing**
  - *Network structure determines where peers belong in the network and where objects are stored*

Some Common Topologies

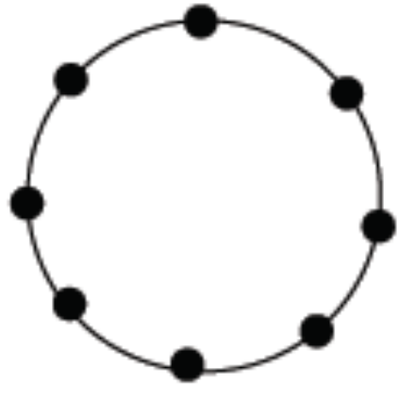
- Flat unstructured: a node can connect to any other node
  - only constraint: maximum degree d_{max}
  - fast join procedure
  - good for data dissemination, bad for location



- Two-level unstructured: nodes connect to a superpeer
  - superpeer form a small overlay
  - used for indexing and forwarding
  - high load on superpeer



- Flat structured: constraints based on node ids
  - allows for efficient data location
  - constraints require long join and leave procedures



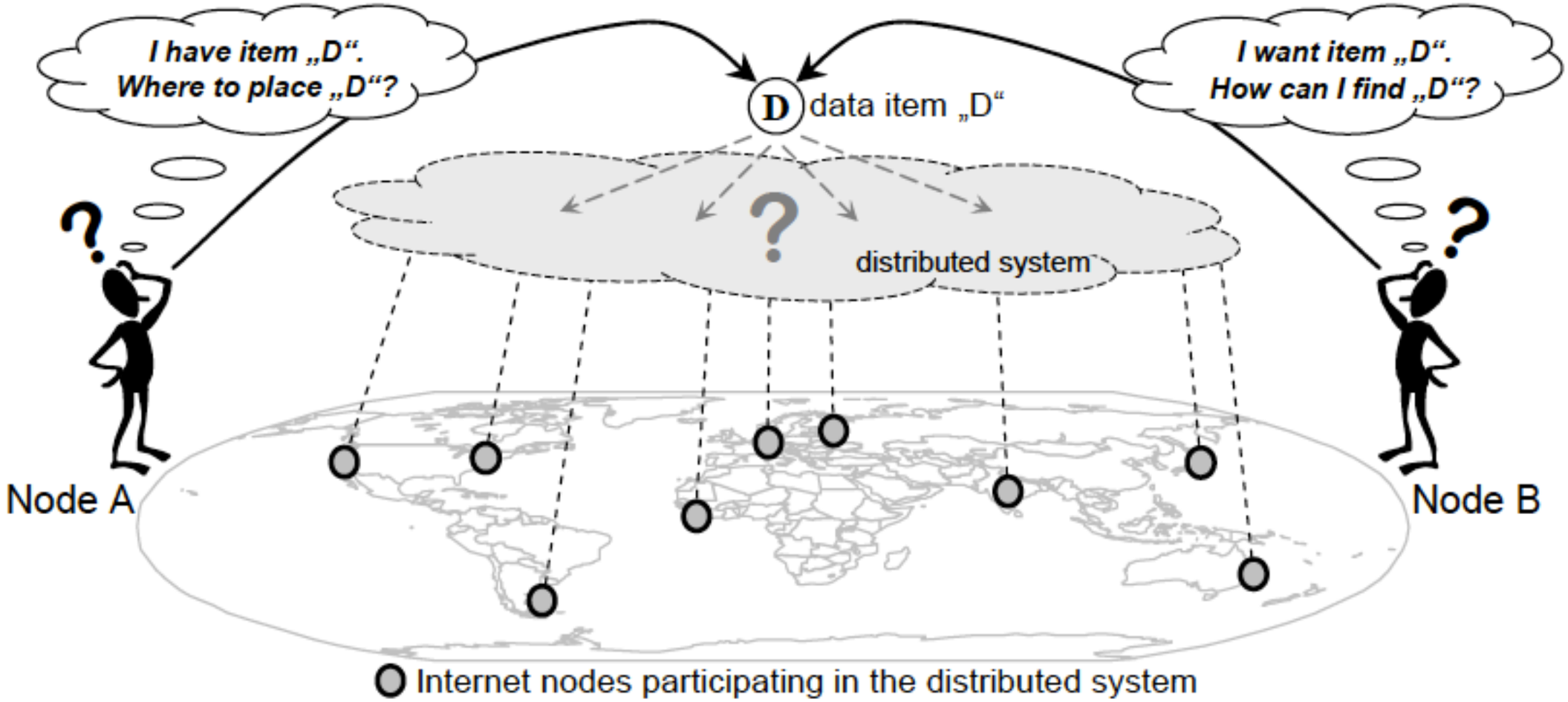
Data Location (Lookup)

1. Introduction
   A. Peer-to-Peer vs. Client/Server
   B. Overlay Networks
2. Common Topologies
3. Data location
4. Gnutella Protocol

Lookup Problem

- Node A wants to store a data item D
- Node B wants to retrieve D without prior knowledge of D's current location

How should the distributed system, especially data placement and retrieval, be organized (in particular, with regard to scalability and efficiency)?

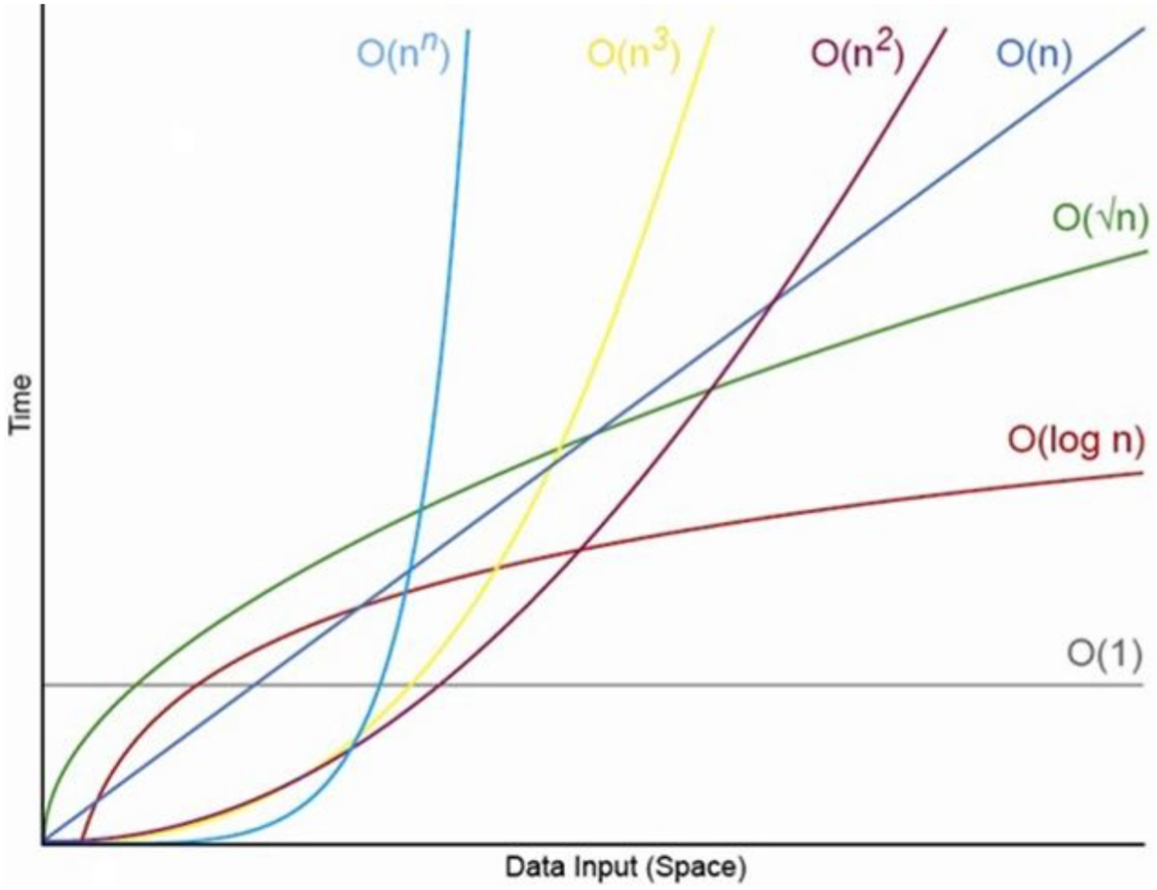


Strategies to Store and Retrieve Data

- Central servers
- Flooding
- Distributed indexing (Distributed Hash Tables)
- Superpeers
- Loosely structured overlays

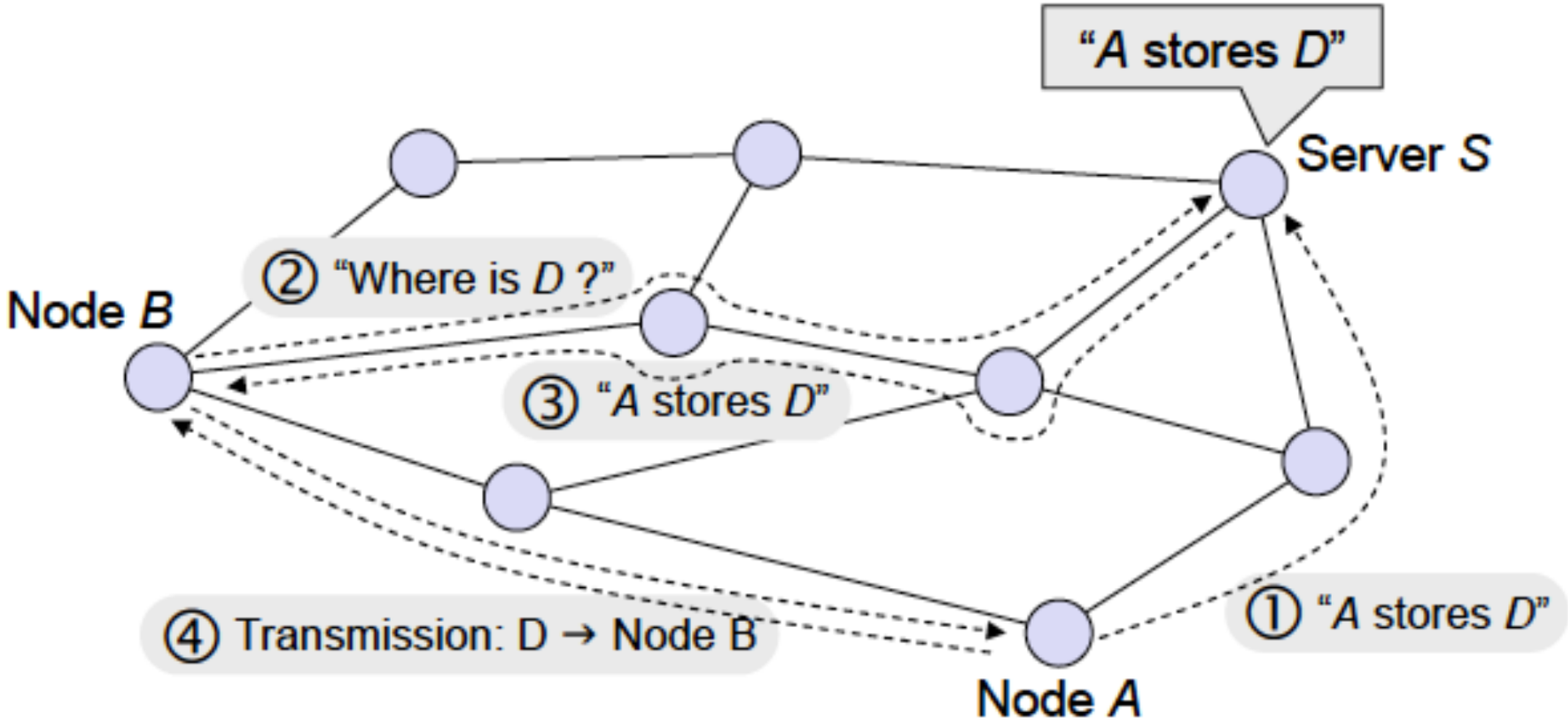
Big O Notation

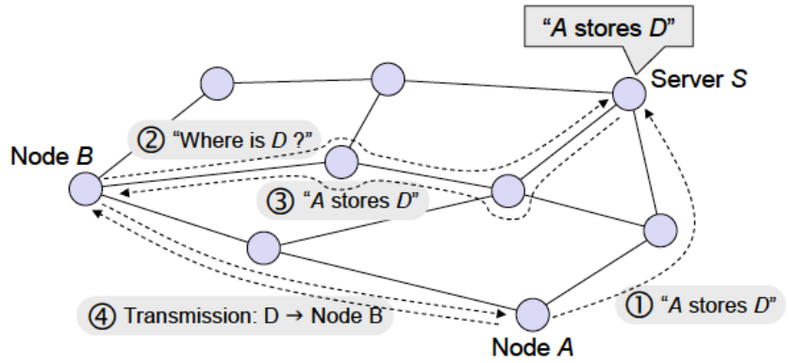
- Big O notation is widely used by computer scientists to concisely describe the behavior of algorithms
- Specifically describes the worst-case scenario, and can be used to describe the execution time required or the space used by an algorithm
- Common types of orders
  - O(1) – constant
  - O(log n) – logarithmic
  - O(n) – linear
  - $O(n^2)$ – quadratic



Central Server

(1) Node A publishes its content on the central server S
(2) Some node B requests the actual location of a data item D from the central server S
(3) If existing, S replies with the actual location of D
(4) The requesting node B transmits the content directly from node A





Central Server: Pros and Cons

- Approach of first generation Peer-to-Peer systems, such as Napster
- **Advantages**
  - search complexity of O(1) – the requester just has to know the central server
  - fuzzy and complex queries possible, since the server has a global overview of all available content
- **Disadvantages**
  - The central server is a critical element concerning scalability and availability
  - Since all location information is stored on a single machine, the complexity in terms of memory consumption is O(N), with N representing the number of items available in the distributed system
  - The server also represents a single point of failure and attack

Strategies to Store and Retrieve Data

- Central servers
- Flooding
- Distributed indexing (Distributed Hash Tables)
- Superpeers
- Loosely structured overlays

Flooding Search

- Approach of the so-called second generation of Peer-to-Peer systems [first Gnutella protocol]

- Key idea: no explicit information about the location of data items in other nodes, other than the nodes actually storing the content

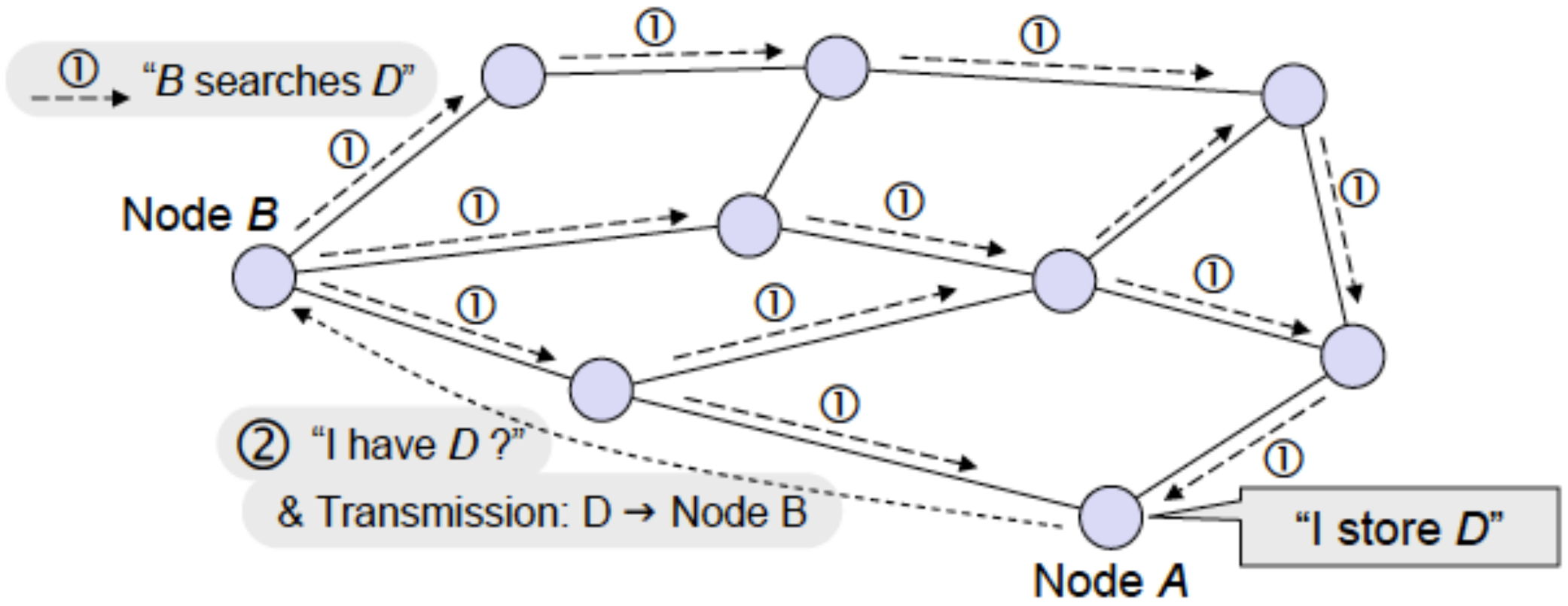
  - No additional information concerning where to find a specific item in the distributed system

  - Thus, to retrieve an item D the only chance is to ask (broadcast) as much participating nodes as necessary, whether or not they presently have item D

  - If a node receives a query, it floods this message to other nodes until a certain hop count (Time to Live – TTL) is exceeded

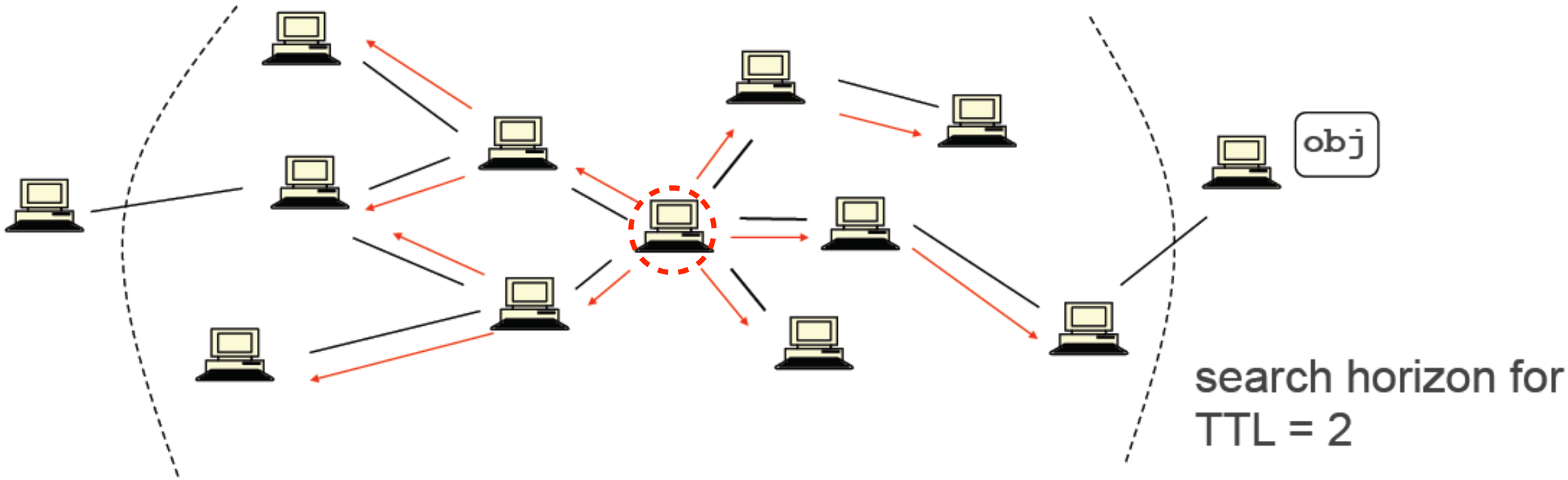
Flooding Search - Idea

- No routing information is maintained in intermediate nodes
  (1) Node B sends a request for item D to its “neighbours”, who forward the request to further nodes in a recursive manner (flooding/breadth-first search)
  (2) Nodes storing D send an answer to B; D is then transmitted directly from the answering node(s)



[Flooding] Search Horizon

- Search results are not guaranteed: flooding stopped by TTL, which produces **search horizon**



Objects that lie outside of the horizon are not found

Flooding: Pros and Cons

- **Advantages**:
  - ✓simplicity
  - ✓no topology constraints
  - ✓storage cost is O(1) because data is only stored in the nodes actually providing the data – whereby multiple sources are possible – and no information for a faster retrieval of data items is kept in intermediate nodes
- **Disadvantages**:
  - ✓broadcast mechanism that does not scale well
  - ✓high network overhead (huge traffic generated by each search request)
  - ✓complexity of looking up and retrieving a data item is $O(N^2)$
  - ✓search results are not guaranteed: flooding stopped by **T**ime-**T**o-**L**ive
  - ✓only applicable to small number of nodes

Why System Design is Important…

After the central server of Napster was shut down in July 2001 due to a court decision,
an enormous number of Napster users migrated to the Gnutella network within a few days

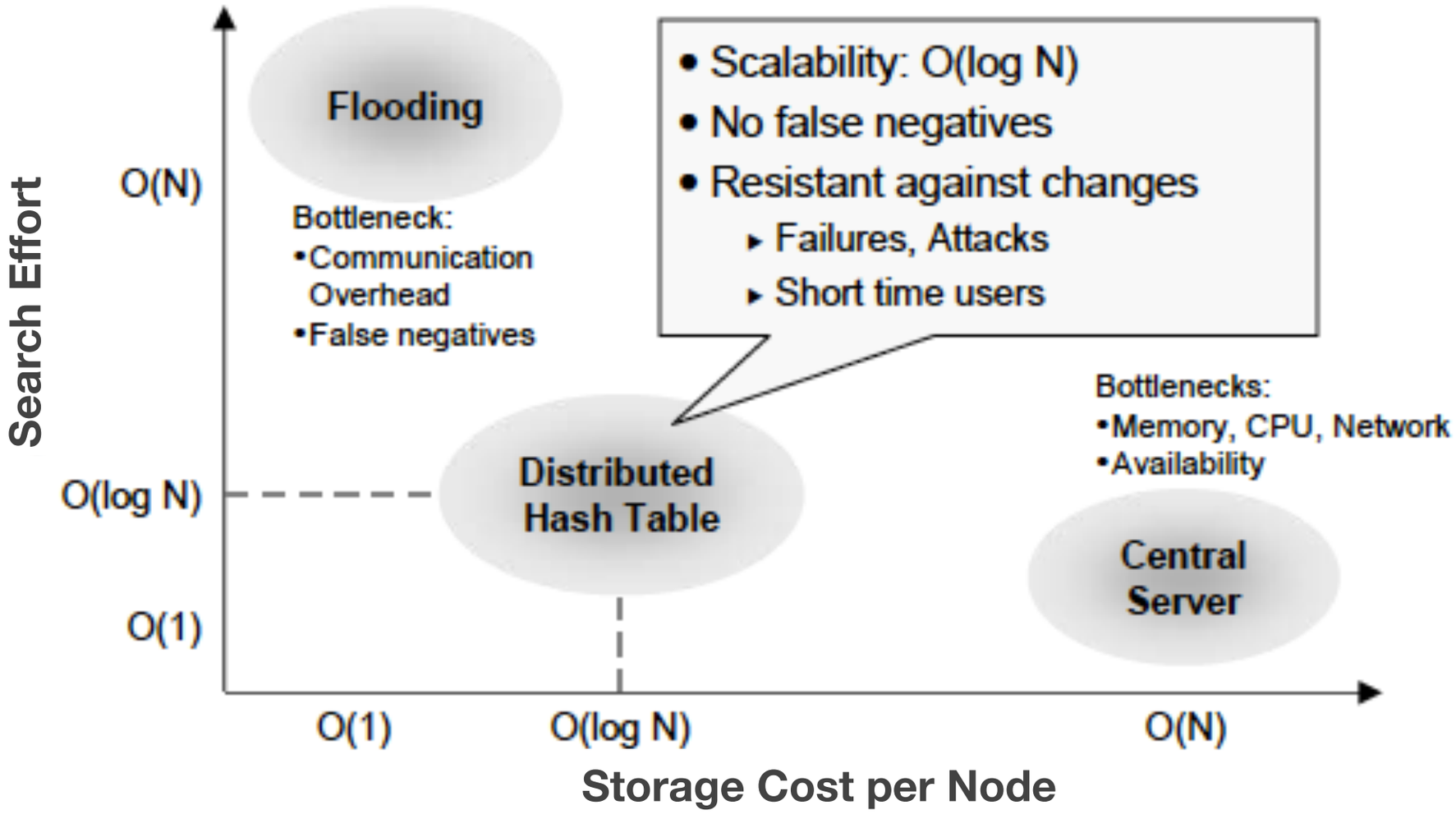
—> under this heavy network load the system collapsed

Strategies to Store and Retrieve Data

- Central servers
- Flooding
- Distributed indexing (Distributed Hash Tables)
- Superpeers
- Loosely structured overlays

Distributed Indexing – Distributed Hash Tables

- Both central servers and flooding-based searching exhibit crucial bottlenecks that contradict the targeted scalability and efficiency of P2P systems
- **Desired scalability**: search and storage complexity **O(log n)**, even if the system grows by some orders of magnitude

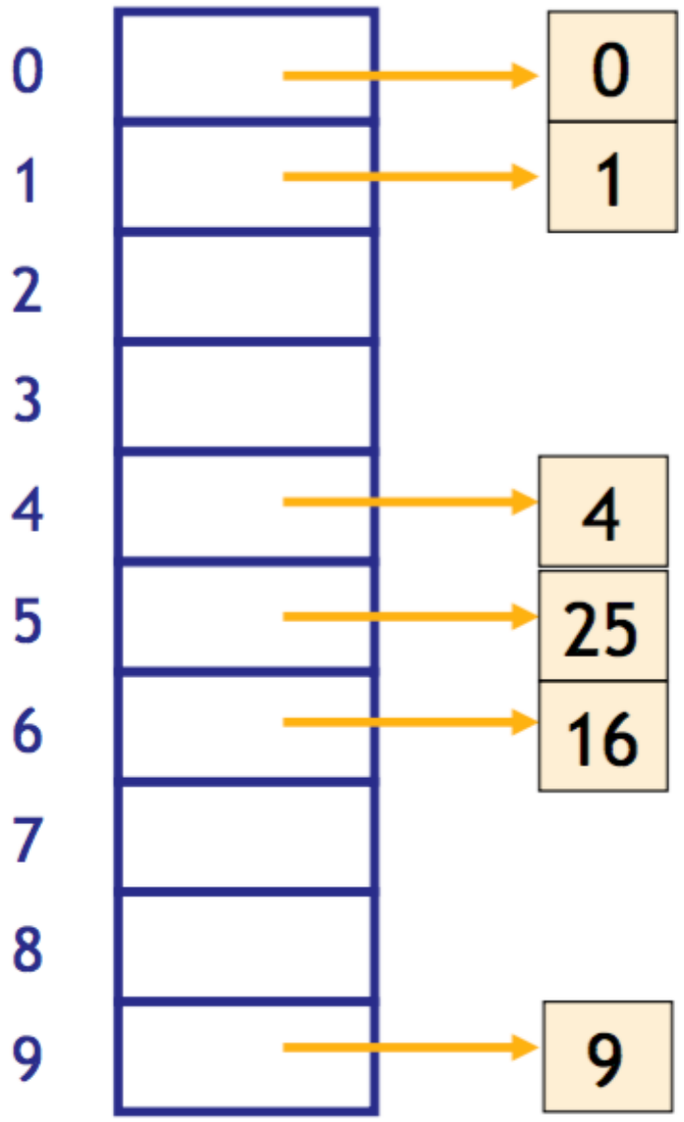


Recall: Hash Tables

- Hash tables are a well-known data structure
- Hash tables allow insertions, deletions, and finds in ***constant*** (average) time
- Hash table is a **fixed-size array**
  - Elements of array also called *hash buckets*
- **Hash function maps keys to elements in the array**
- Properties of **good** hash functions:
  - Fast to compute
  - Good distribution of keys into hash table
  - Example: SHA-1 algorithm

Hash Tables: Example

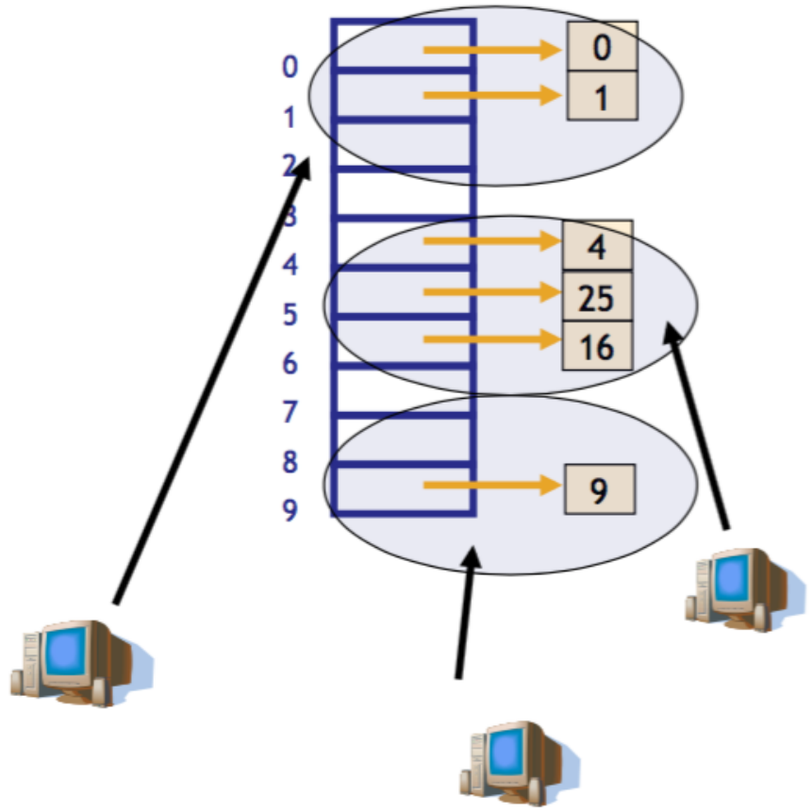
- **Hash function maps keys to elements in the array**

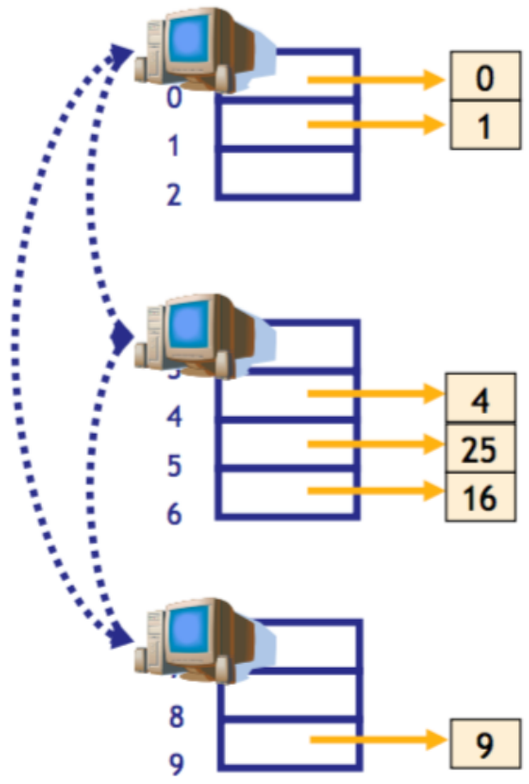


- Hash function: hash(k) = k mod 10
- Insert keys 0, 1, 4, 9, 16, and 25
- Easy to find if a given key is present in the table

Distributed Hash Table: Idea

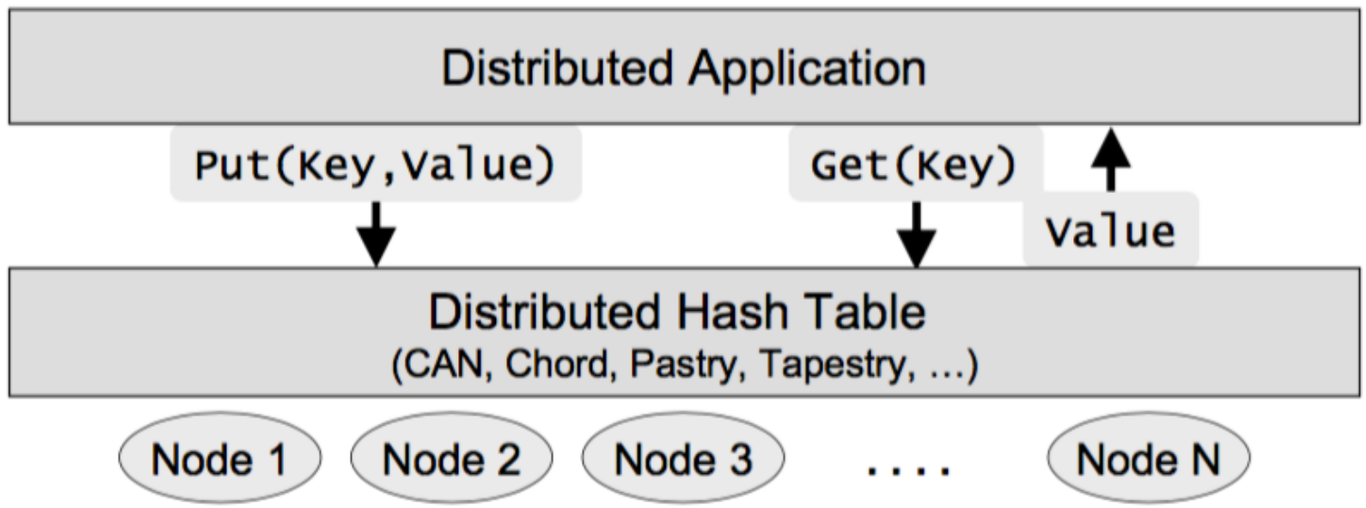
- Hash tables are fast for lookups (O(1))
- **Idea**: ***distribute hash buckets to nodes***
- Nodes form an *overlay network*
  - Route messages in overlay to find responsible node
  - Routing scheme in the overlay network is the difference between different DHTs
- Result is **Distributed Hash Table (DHT)**





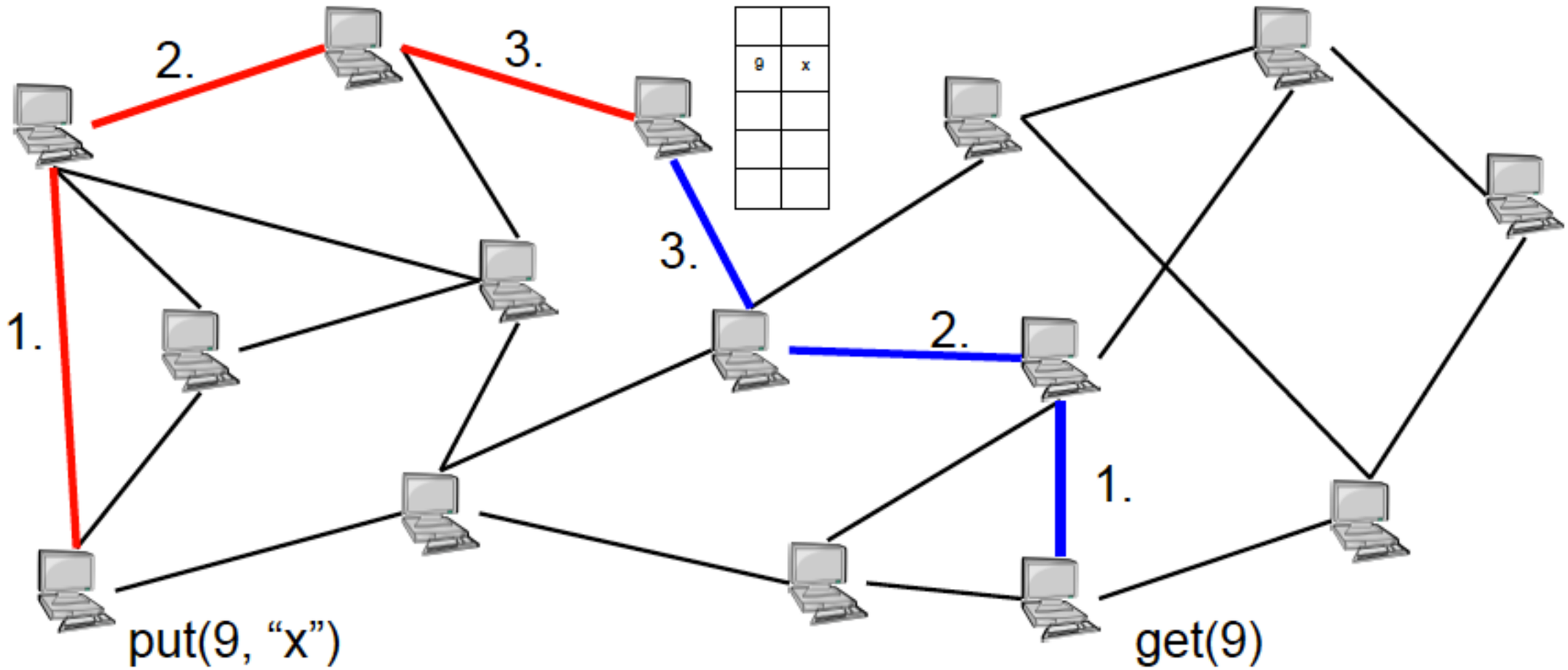
Distributed Indexing – Distributed Hash Tables

- A P2P algorithm that offers an associative **Map** interface:
  - *put(Key k; Value v)*: associate a value/item *v* to the key *k*
  - *Value get(Key k)*: returns the value associated to key *k*



- Distributed Hash Tables: map keys to nodes
- Organization:
  - Each node is responsible for a portion of the key space
  - Messages are routed between nodes to reach responsible nodes
  - Replication used to tolerate failures

Route Puts/Gets Through the Overlay



DHT Implementations

- The founders (2001):
  - Chord, CAN, Pastry, Tapestry
- The ones which are actually used:
  - Kademlia and its derivatives (up to 4M nodes!)
    - ✓BitTorrent, Kad (eMule), The Storm Botnet
  - Cassandra DHT
    - ✓Part of Apache Cassandra
    - ✓Initially developed at Facebook
- The ones which are actually used, but we don't know much about:
  - Microsoft DHT based on Pastry
  - Amazon's Dynamo key-value store

Step 1: From Keys and Nodes to IDs

- Keys and nodes are represented by **identifiers** taken from the *same* **ID space**
  - Key identifiers: computed through an hash function (e.g., SHA-1)

    e.g., *ID(k) = SHA1(k)*
  - Node identifiers: randomly assigned or computed through an hash function

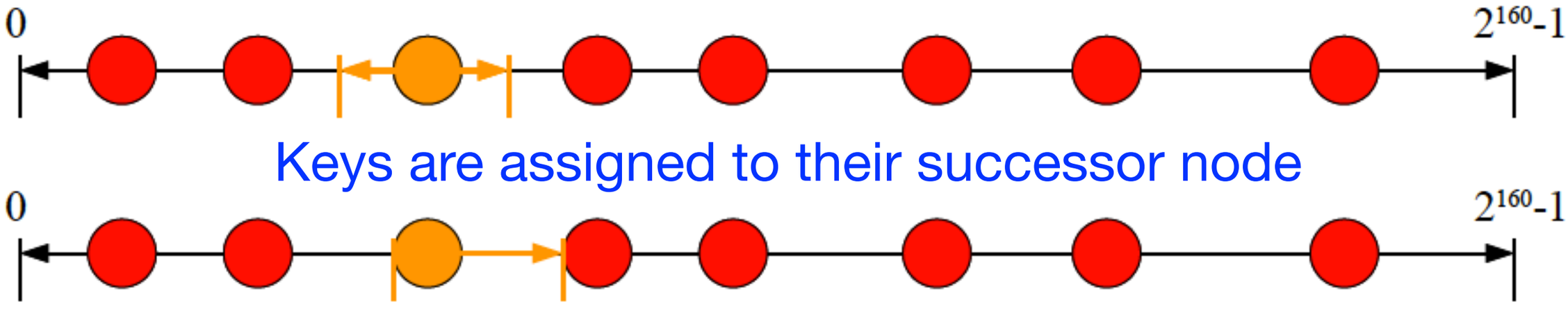
    e.g., *ID(n) = SHA1*(IP address of *n*)
- Why?
  - Very low probability that two nodes have exactly the same ID
  - Nodes and keys are mapped in the *same space*

Step 2: Partition the ID Space

- Each node in the DHT stores some *k, v* pairs.
- Partition the ID space in zones, depending on the node IDs:
  - a pair *(k, v)* is stored at the node *n* such that (examples):

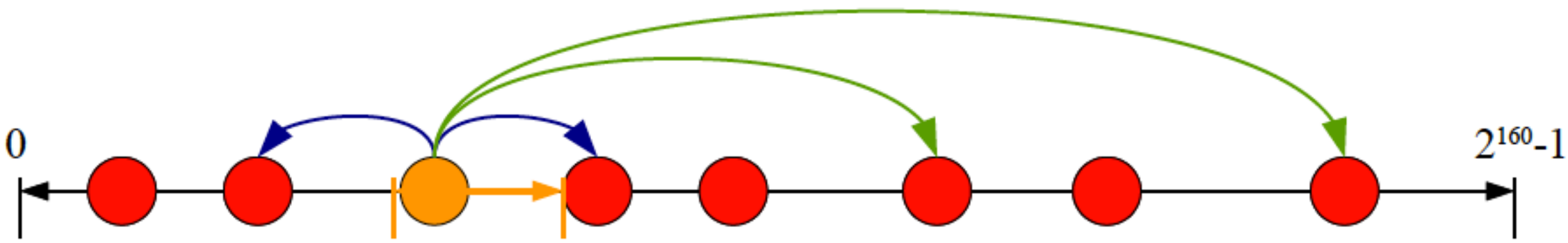
✓its identifier *ID(n)* is the closest to *ID(k)*;

✓its identifier *ID(n)* is the largest node id smaller than *ID(k)*



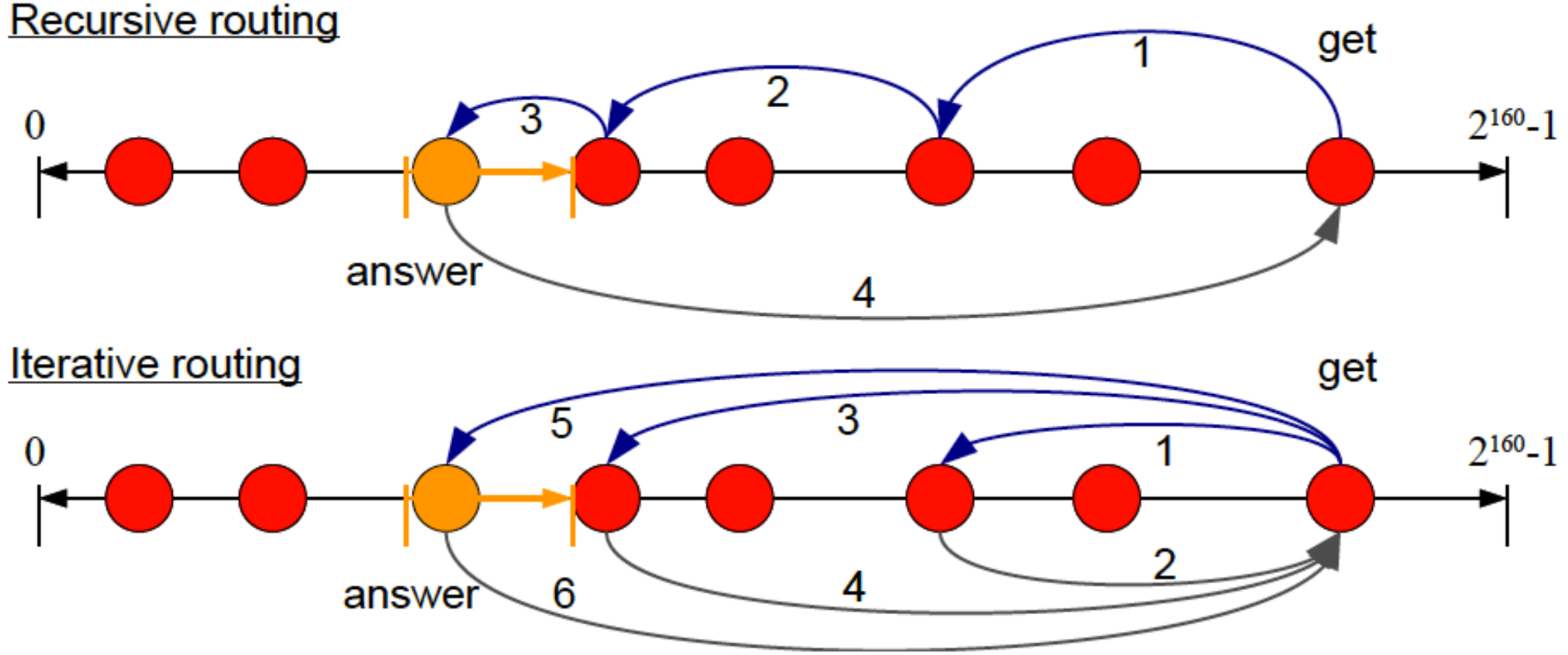
Step 3: Build Overlay Network

- Each DHT node manages a **O(log n)** references to other nodes, where **n** is the number of nodes in the system
- Each node has two sets of neighbors:
  - Immediate neighbors in the key space (leafs)
    - ✓Guarantee correctness, avoid partitions
    - ✓But with only them, linear routing time
  - Long-range neighbours
    - ✓Allow sub-linear routing
    - ✓But with only them, connectivity problems



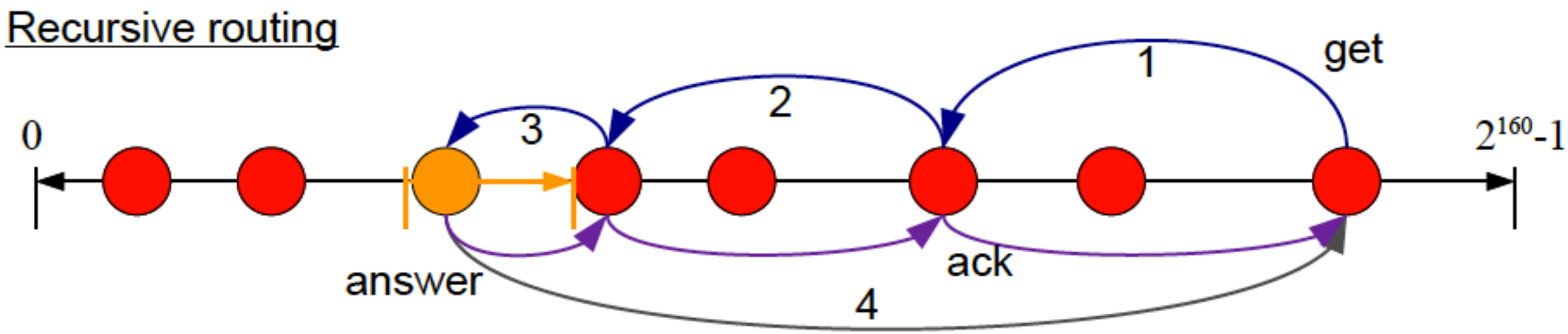
Step 4: Route Puts/Gets Through the Overlay

- Recursive routing: the initiator starts the process, contacted nodes forward the message
- Iterative routing: the initiator personally contact the nodes at each routing step



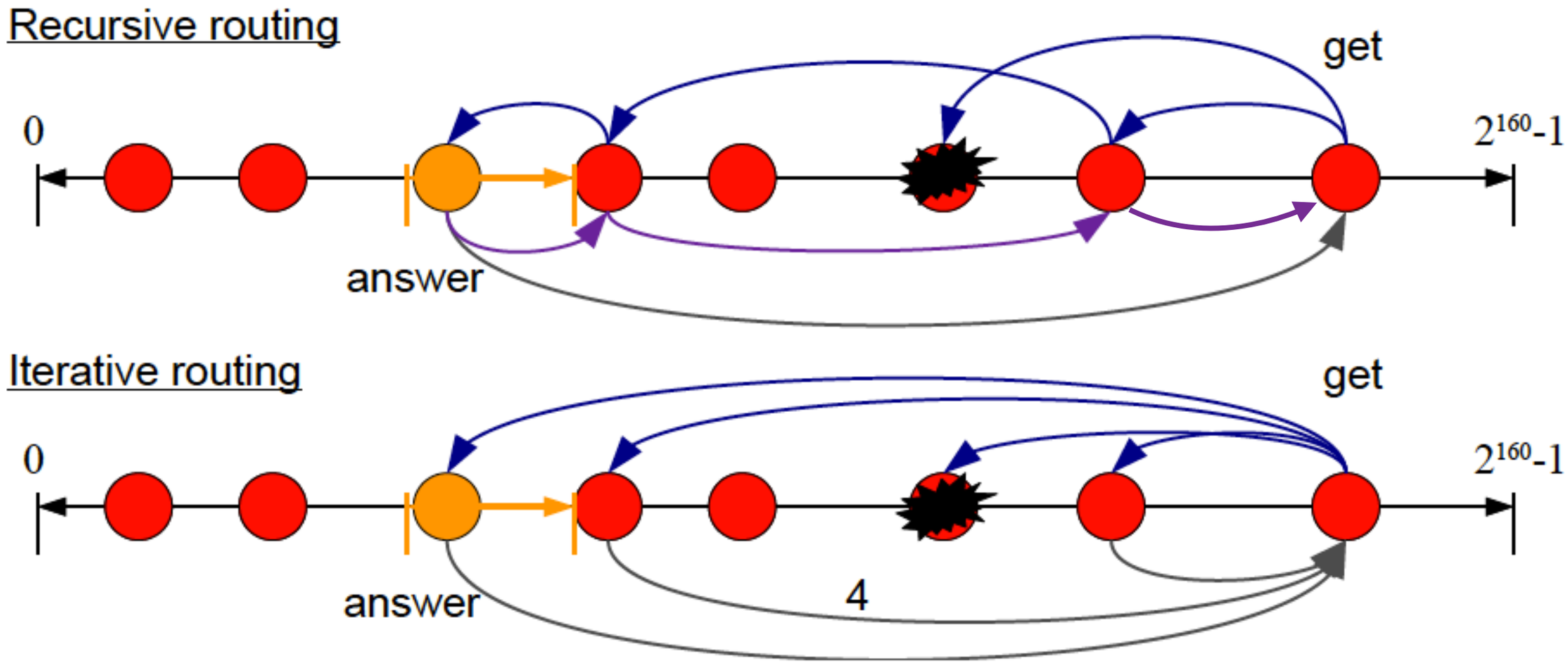
Routing Around Failures (1)

- Under churn, neighbors may have failed
- To detect failures, acknowledge each hop (recursive routing)



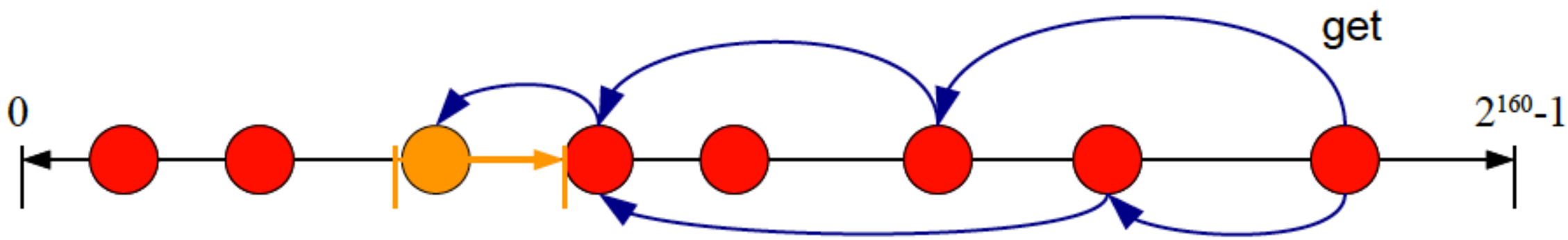
Routing Around Failures (2)

- If we don't receive ack or response, resend through a different neighbor



Routing Around Failures (3)

- Must compute timeouts carefully
  - If too long, increase put/get latency
  - If too short, get message explosion
- Parallel sending could be a design decision (see Kademlia)

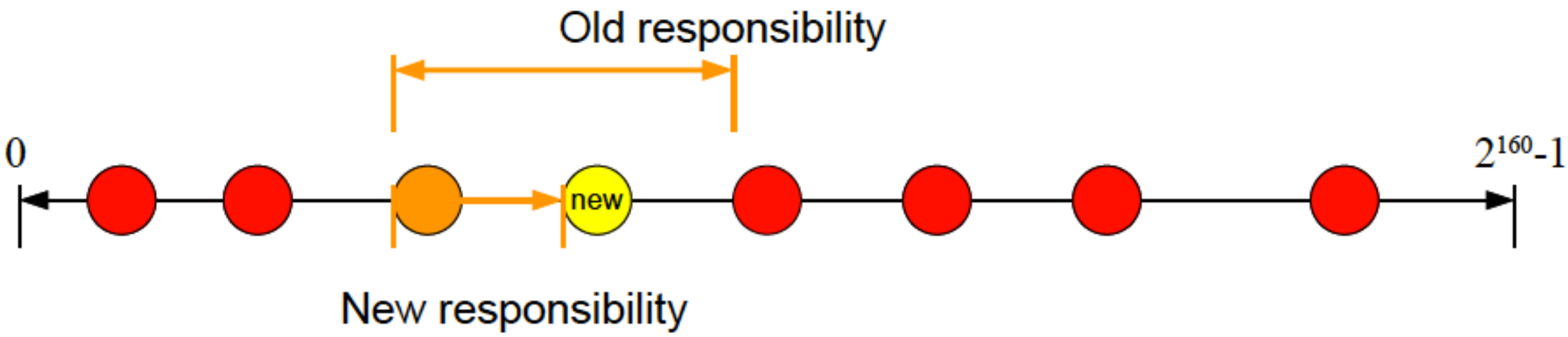


Computing Good Timeouts

- Use TCP-style timers
  - Keep past history of latencies
  - Use this to compute timeouts for new requests
- Works fine for recursive lookups
  - Only talk to neighbors, so history small, current
- In iterative lookups, source directs entire lookup
  - Must potentially have good timeout for any node

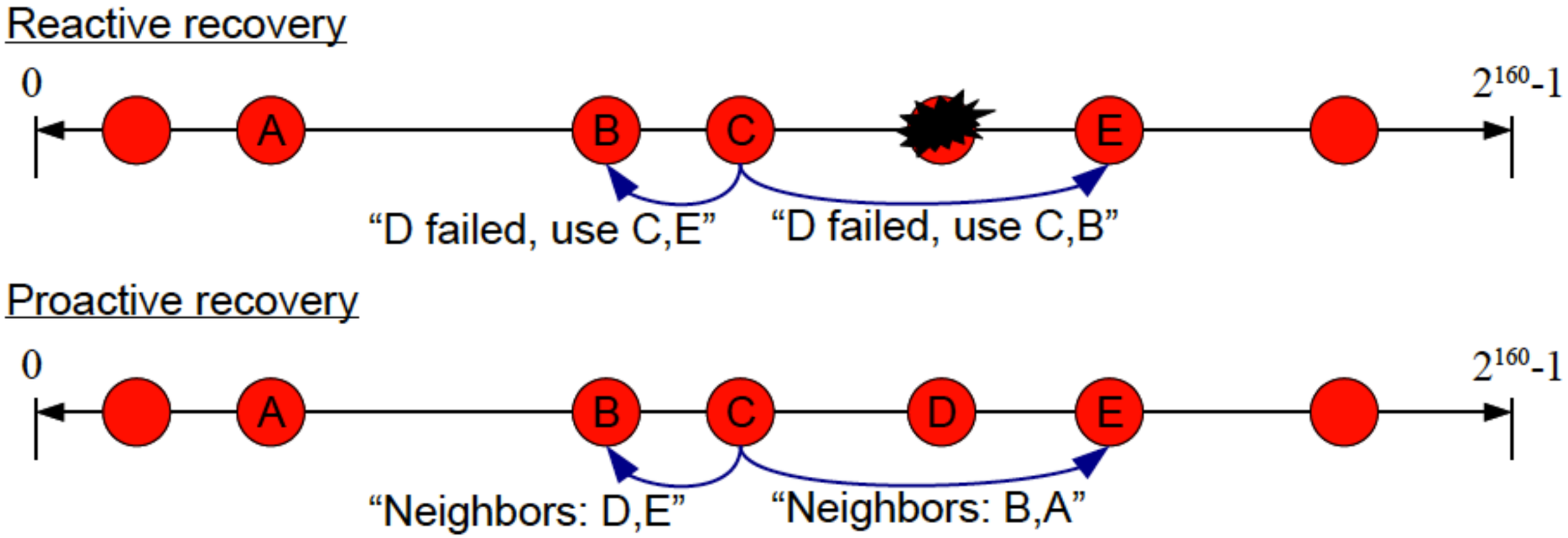
Recovering from Failures

- Can't route around failures forever
  - Will eventually run out of neighbors
- Must also find new nodes as they join
  - Especially important if they're our immediate predecessors or successors



Recovering from Failures

- Reactive recovery
  - When a node stops sending acknowledgments, notify other neighbors of potential replacements
- Proactive recovery
  - Periodically, each node sends its neighbor list to each of its neighbors



DHT: Pros and Cons

- **Advantages**:
  - completely decentralized (no need for superpeers)
  - routing algorithm achieves low hop count (O(log *n*))
  - storage cost per node: O(log *n*)
  - if a data item is stored in the system, the DHT guarantees that the data is found
- **Disadvantages**:
  - objects are tracked by unreliable nodes (which may disconnect)
  - keyword-based searches are more difficult to implement than with superpeers (because objects are located by their *objectid*)
  - the overlay must be structured according to a given topology in order to achieve a low hop count
  - routing tables must be updated every time a node joins or leaves the overlay

Comparison of Basic Lookup Concepts

System	Per Node State	Communication Overhead	Fuzzy Queries	Robust-ness
Central Server	$O(N)$	$O(1)$	✓	×
Flooding Search	$O(1)$	$\geq O(N^2)$	✓	✓
Distributed Hash Table	$O(\log N)$	$O(\log N)$	×	✓

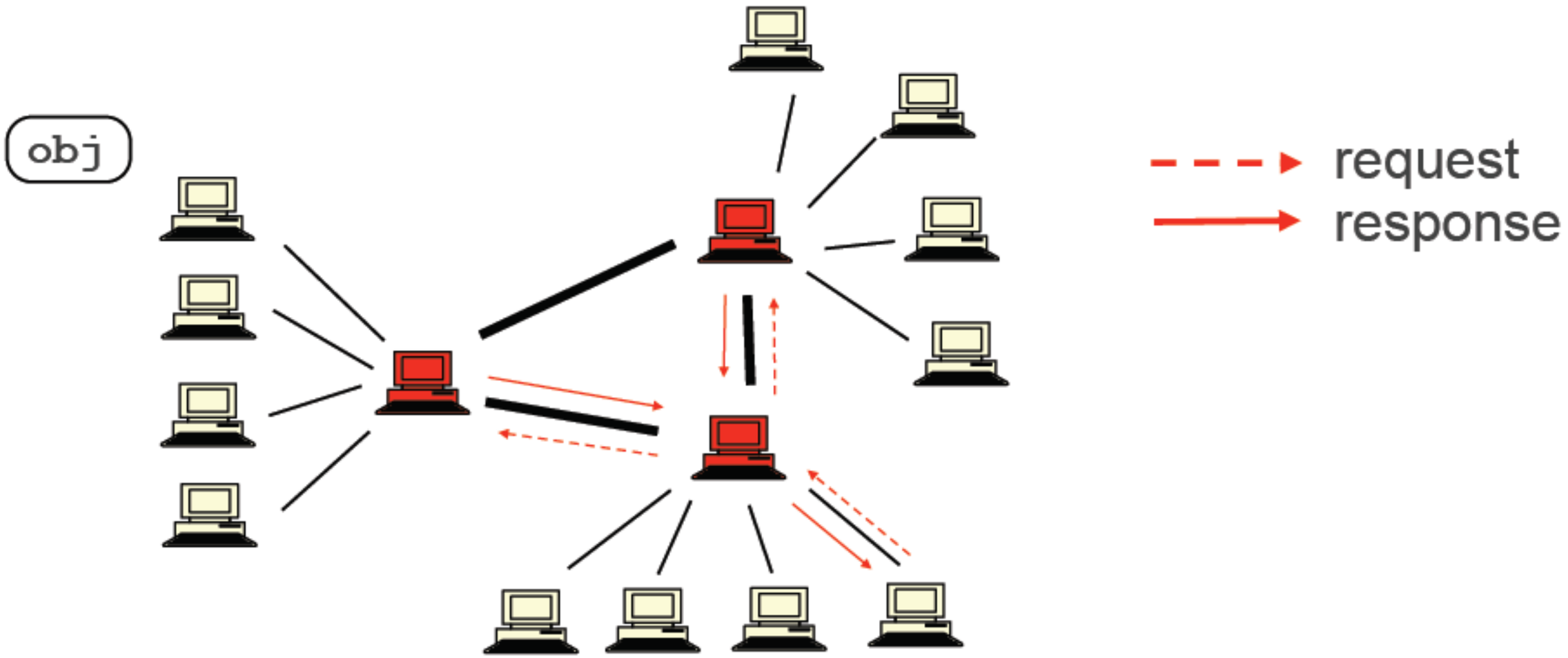
Strategies to Store and Retrieve Data

- Central servers
- Flooding
- Distributed indexing (Distributed Hash Tables)
- Superpeers
- Loosely structured overlays

Superpeers

- **Two-level overlay**: use superpeers to track the locations of an object [Gnutella 2, BitTorrent]
  - Each node connects to a superpeer and advertises the list of objects it stores
  - Search requests are sent to the supernode, which forwards them to other super nodes
- **Advantages**: highly scalable
- **Disadvantages**:
  - ✓ superpeers must be reliable, powerful and well connected to the Internet (expensive)
  - ✓ superpeers must maintain large state
  - ✓ the system relies on a small number of superpeers

Superpeers Example



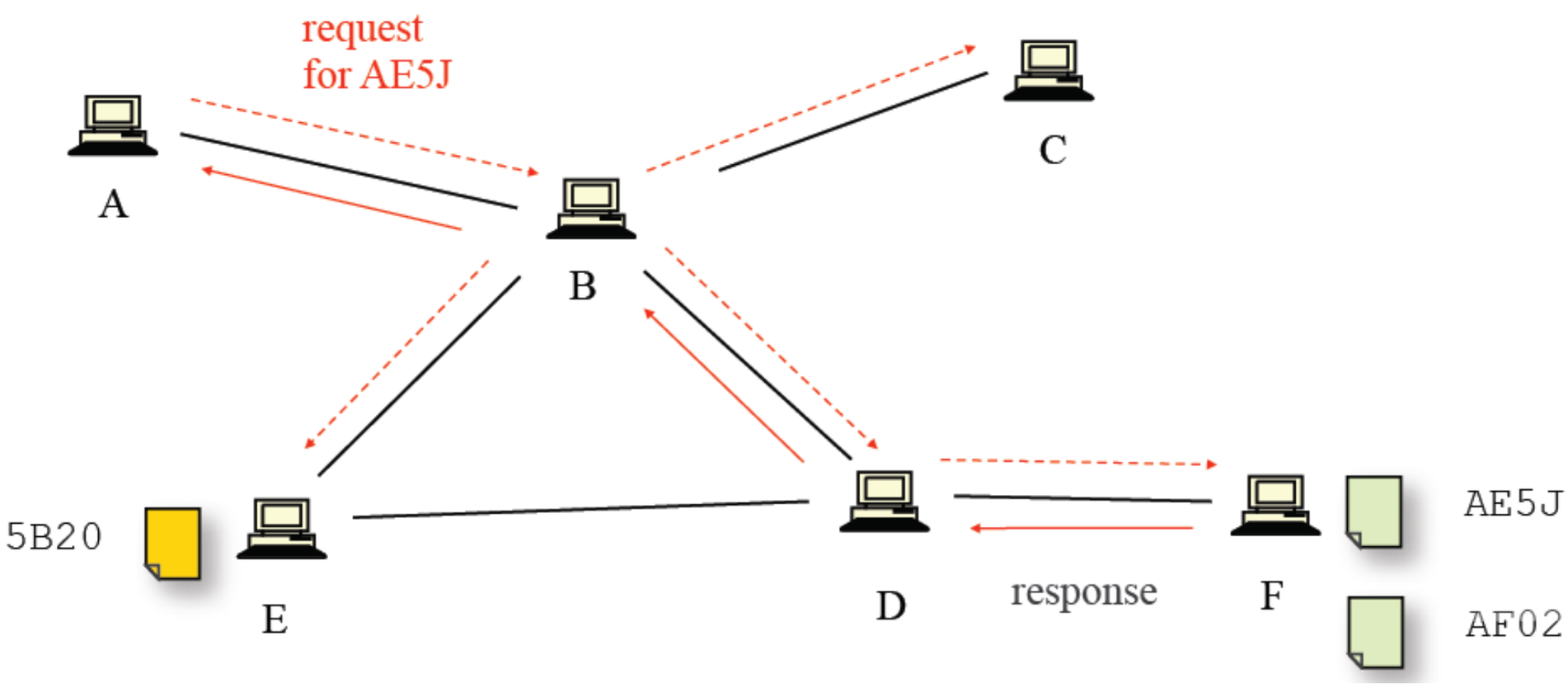
- A two-level overlay is a **partially centralized** system
- In some systems, superpeers may be disconnected (e.g., BitTorrent)

Strategies to Store and Retrieve Data

- Central servers
- Flooding
- Distributed indexing (Distributed Hash Tables)
- Superpeers
- Loosely structured overlays

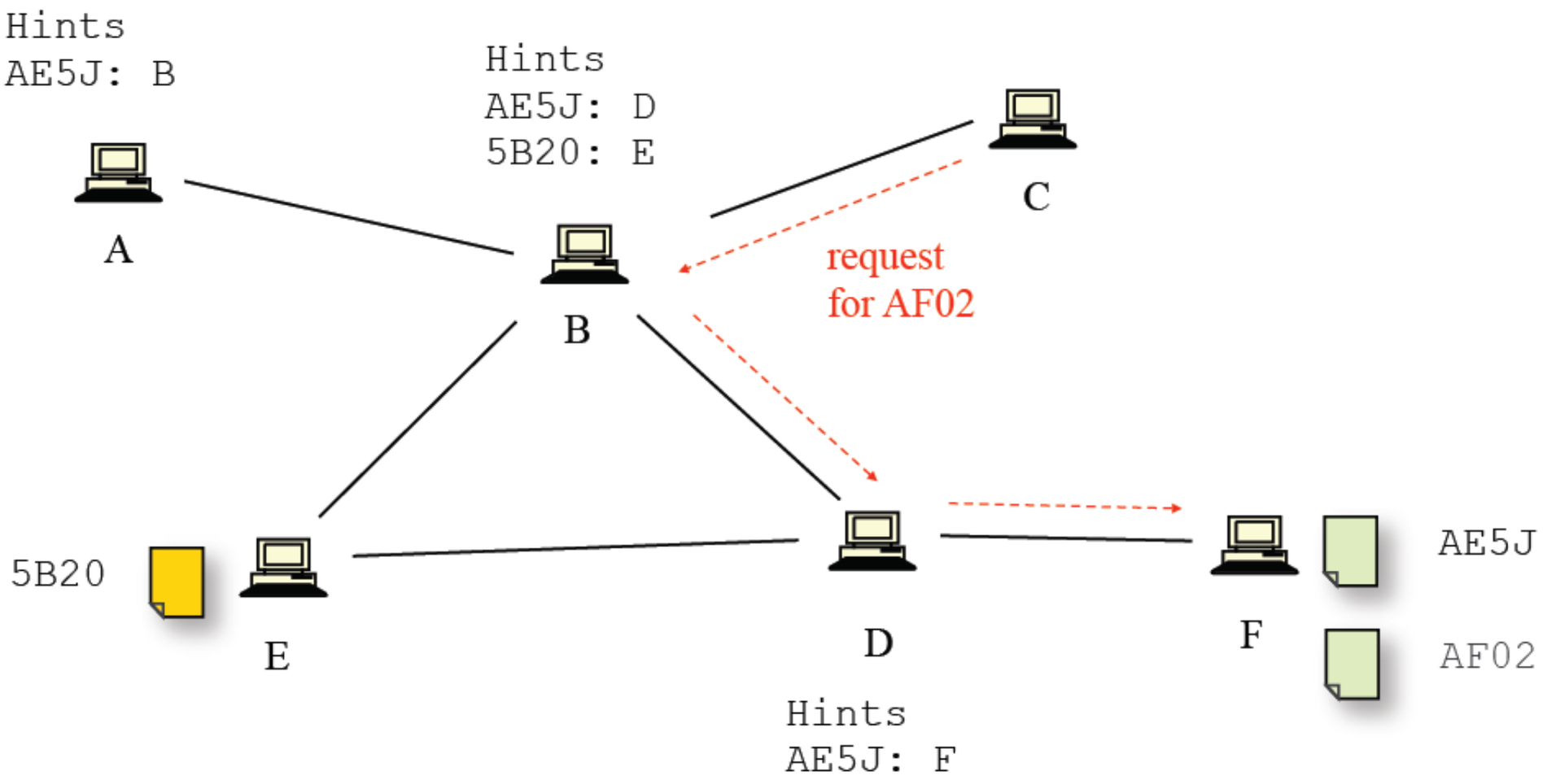
Loosely Structured Overlays

- Loosely structured networks: use ***hints*** for the location of objects [Freenet]
  - Nodes locate objects by sending search requests containing the *objectId*
  - Requests are propagated using a technique similar to flooding
  - *Objects with similar identifiers are grouped on the same nodes*



Loosely Structured Overlays (cont.)

- A search response leaves ***routing hints*** on the path back to the source
- ***Hints*** are used when propagating future requests for *similar object ids*



Loosely Structured Overlays: Pros and Cons

- **Advantages**:
  - no topology constraints, flat architecture
  - searches are more efficient than with plain flooding
- **Disadvantages**:
  - does not support keyword-based searches
  - search requests have a TTL
  - do not guarantee a low number of hops, nor that the object will be found

Data Location - Classification

- Classification of some (well known) P2P middleware according to structure and decentralisation

Degree of Decentralization	Degree of Structure: **low**	**loose**	**high**
partial	eMule Gnutella 2 BitTorrent	-	-
total	Gnutella	Freenet	Chord Pastry

Gnutella Protocol

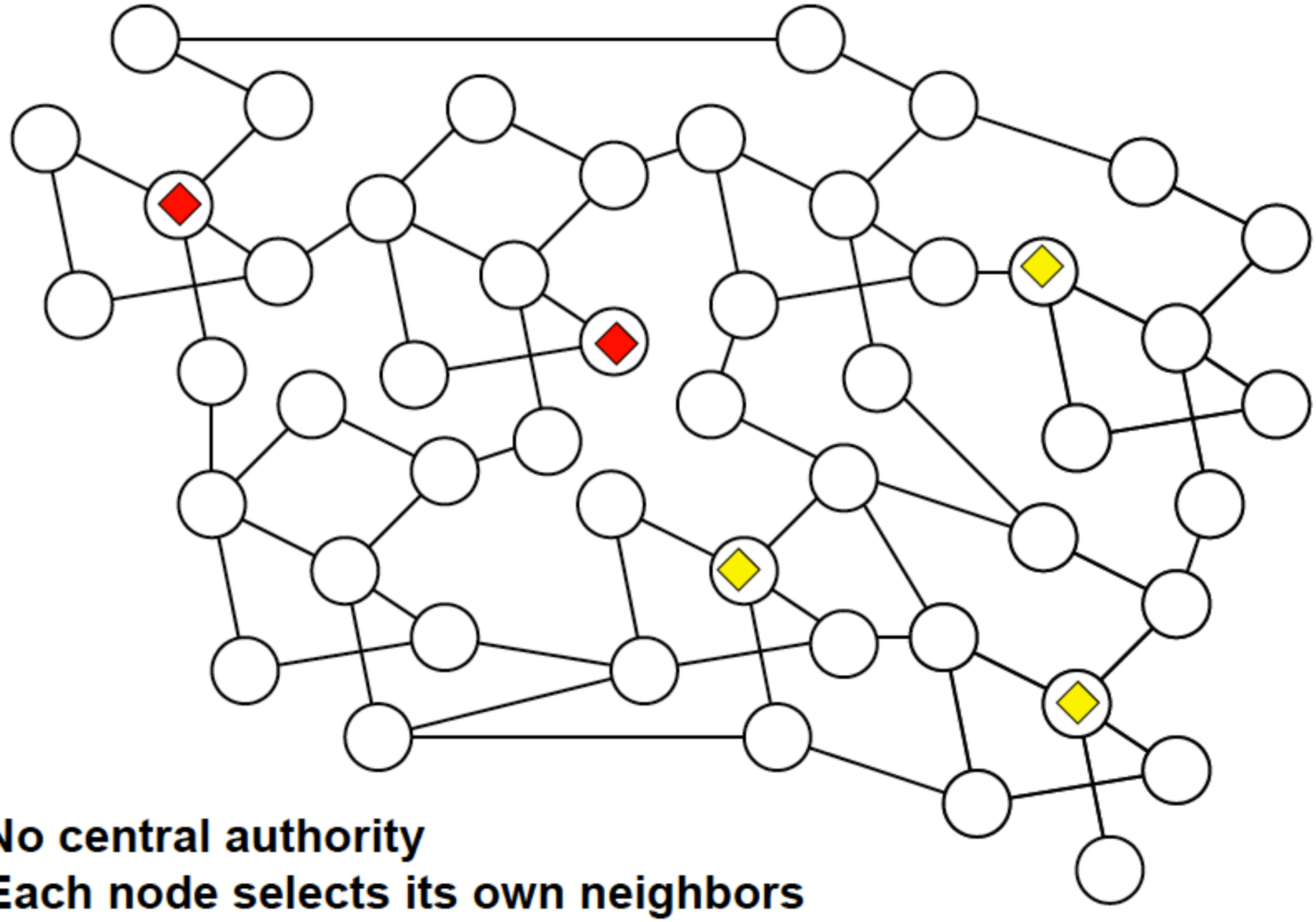
Gnutella: Brief History

- Nullsoft (a subsidiary of AOL) released Gnutella on March 14th, 2000, announcing it on Slashdot
- AOL removed Gnutella from Nullsoft servers on March 15th, 2000
- After a few days, the Gnutella protocol was reverse-engineered
- Napster was shutdown in early 2001, spurring the popularity of Gnutella
- On October 2010, LimeWire (a popular client) was shutdown by court's order

Gnutella

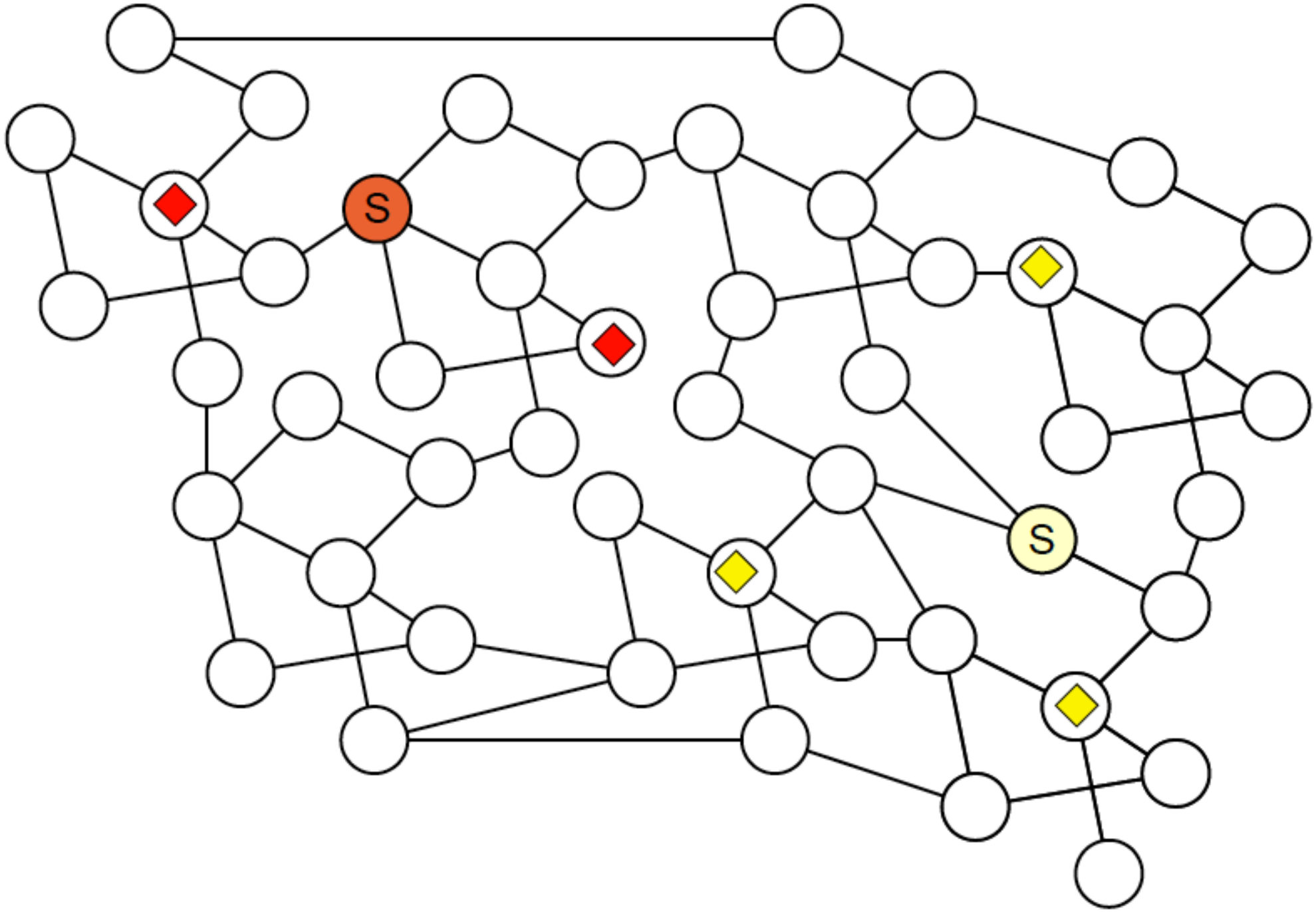
- Gnutella is a protocol for peer-to-peer search, consisting of:
  - A set of message formats
    - ✓5 basic message types
  - A set of rules governing the exchange of messages
    - ✓Broadcast
    - ✓Back-propagate
    - ✓Handshaking
  - An hostcache for node bootstrap

Gnutella Topology: Unstructured

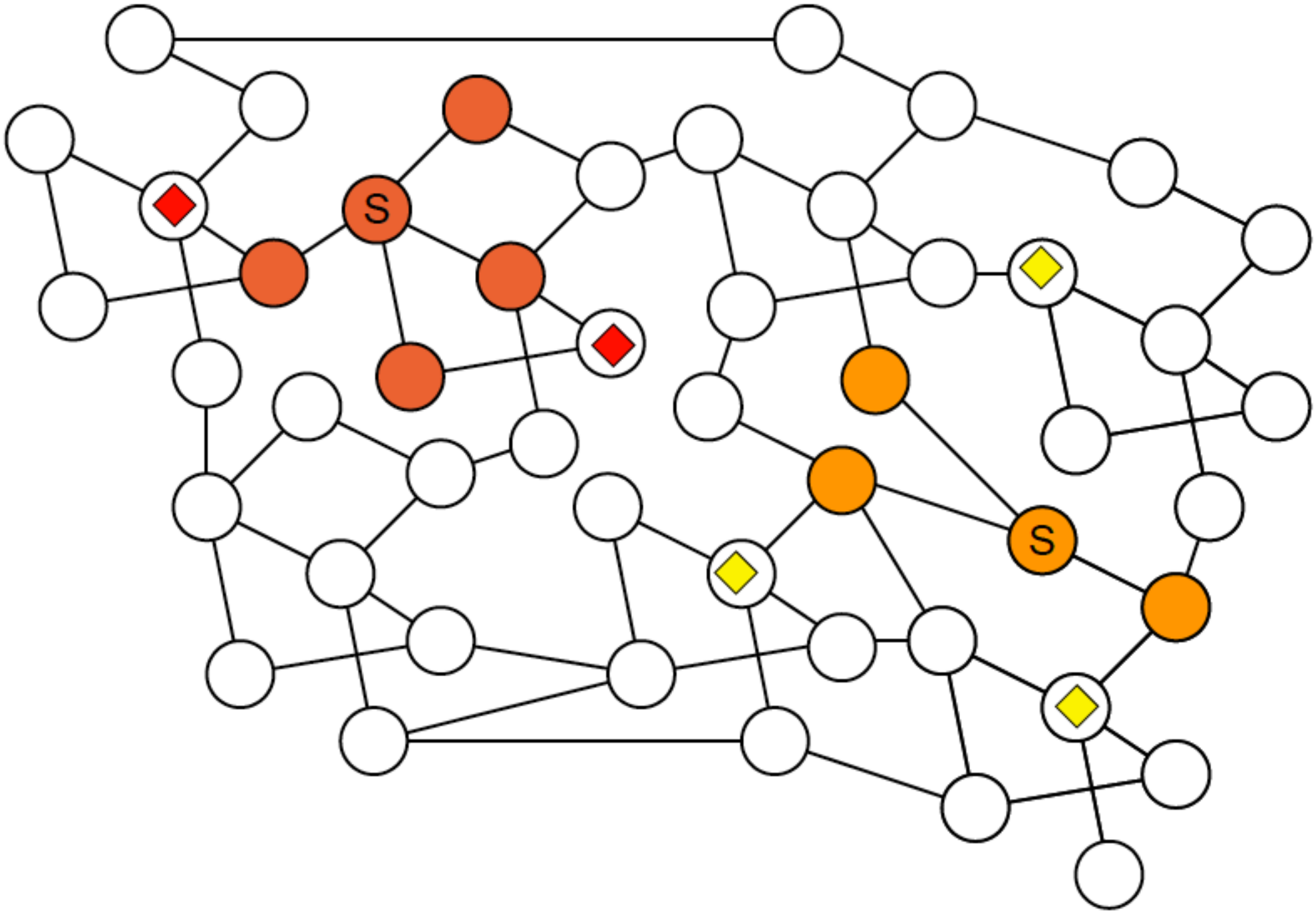


No central authority
Each node selects its own neighbors

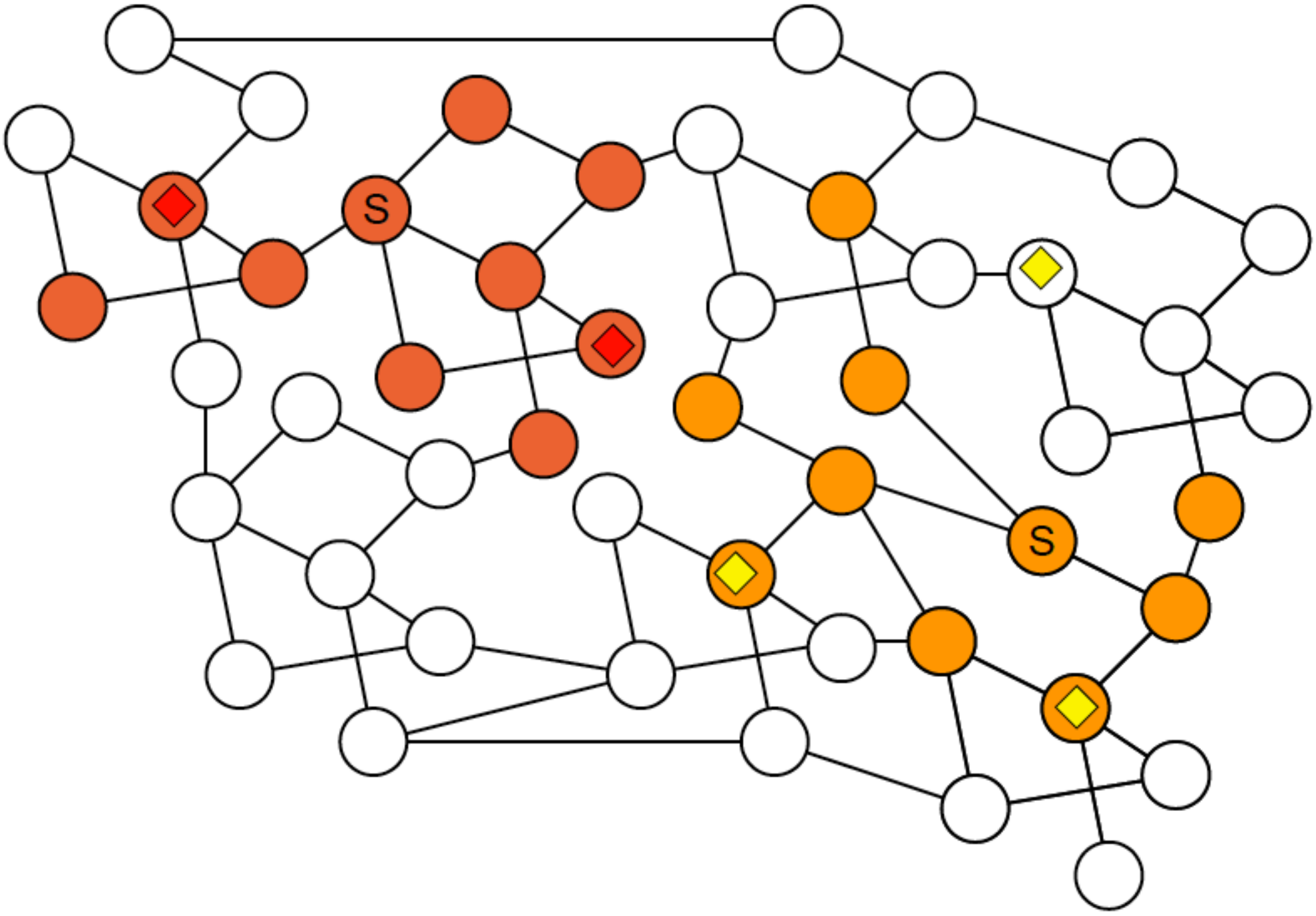
Gnutella Routing



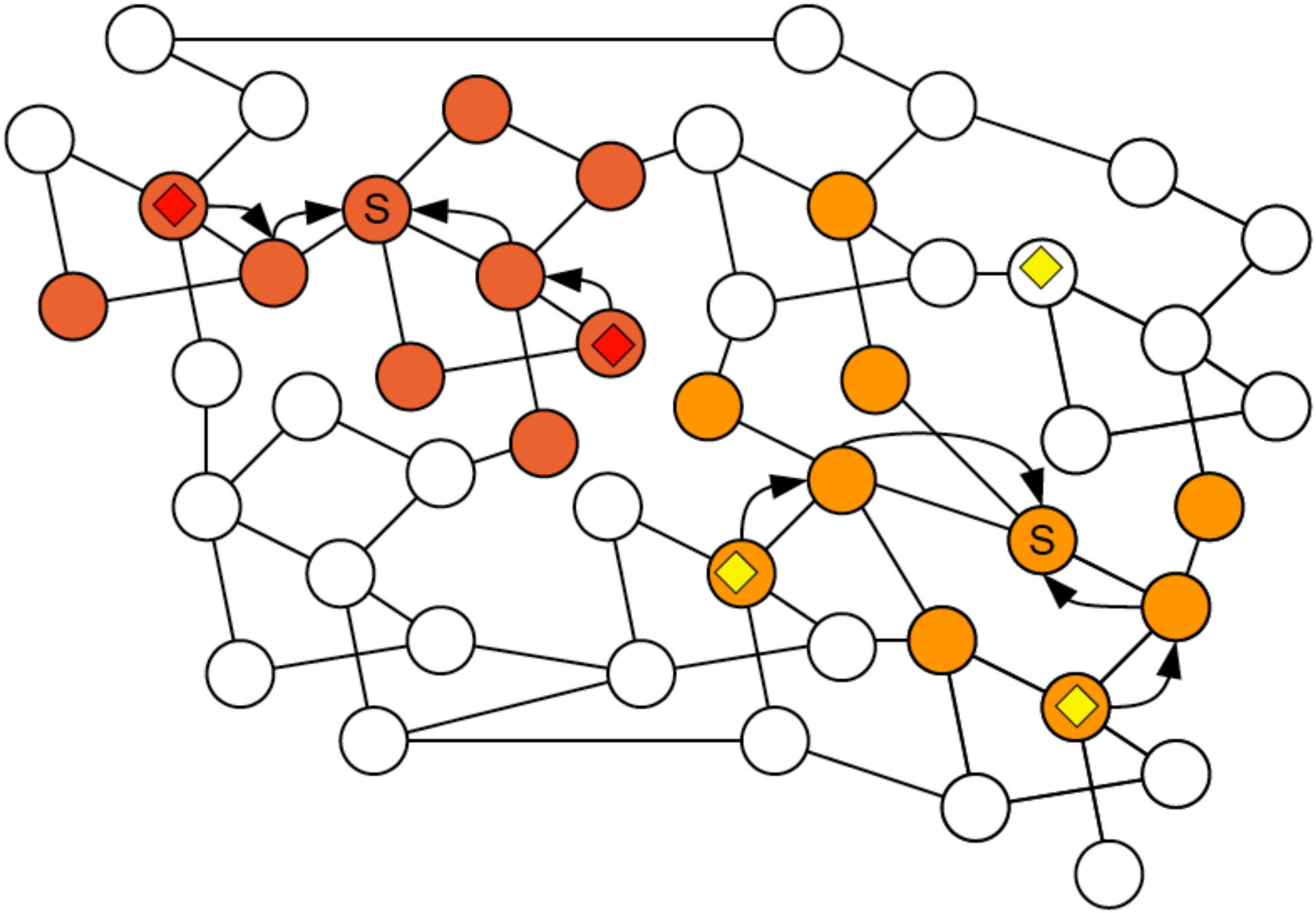
Gnutella Routing



Gnutella Routing



Gnutella Routing



Gnutella Messages

- Each message is composed of:
  - A 16-byte ID field uniquely identifying the message
    - ✓randomly generated
    - ✓not related to the address of the requester (anonymity)
    - ✓used to detect duplicates and route back-propagate messages
  - A message type field
    - ✓PING, PONG
    - ✓QUERY, QUERYHIT
    - ✓PUSH(for rewalls)
  - A Time-To-Live (TTL) Field
  - Payload length

Gnutella Messages

- PING (broadcast)
  - Used to maintain information about the nodes currently in the network
  - Originally, a “who's there" flooding message
  - A peer receiving a ping is expected to respond with a pong message

- PONG (back-propagate)
  - A pong message has the same ID of the corresponding ping message
  - Contains:

    ✓address of connected Gnutella peer

    ✓total size and total number of files shared by this peer

Gnutella Messages

- QUERY (broadcast)
  - The primary mechanism for searching the distributed network
  - Contains the query string
  - A servent is expected to respond with a QUERYHIT message if a match is found against its local data set

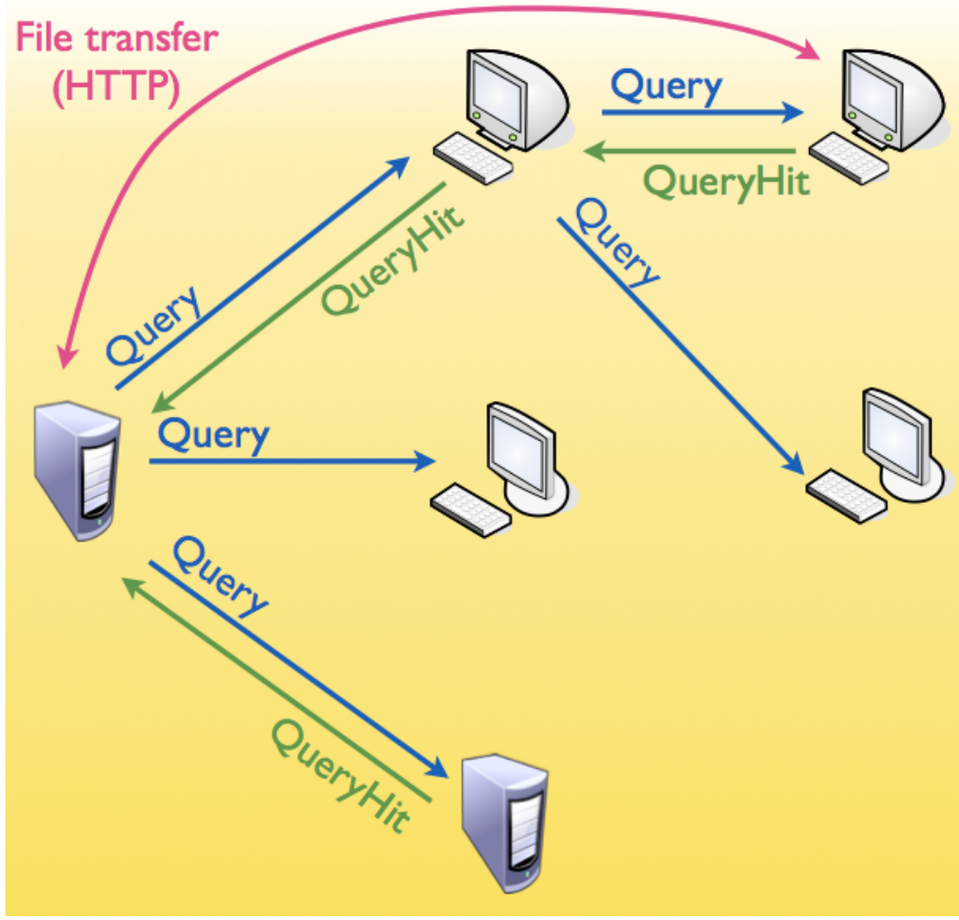
- QUERYHIT (back-propagate)
  - The response to a query
  - Has the same ID of the corresponding query message
  - Contains enough info to acquire the data matching the corresponding query

    ✓IP Address + port number

    ✓List of file names

Flooding Search - Step by Step…

- Peers send msgs to neighbouring in the overlay network over pre-existing TCP connections
- The neighbours forward the **Query** msg to all of their neighbours, recursively
- When a peer receives a **Query** msg, it checks to see whether the keyword matches any of the files it is making available for sharing
- Once a match is found, it sends back a **QueryHit** msg, containing the name and size of the file
- The **QueryHit** msg follows the reverse path as the **Query** msg, using pre-existing TCP connections
- Multiple **QueryHit** messages may be received, in which case the user decides which file to download
- The Gnutella process then sets up a direct TCP connection with the desired user and sends a **HTTPGET** message that includes the specific file name
- The file is sent with a **HTTP** response message
- Once the entire file is received, the direct TCP connection is terminated



Beyond the Original Gnutella

- Several problems in Gnutella 0.4 (the original one):
  - PING-PONG traffic

    ✓More than 50% of the traffic generated by Gnutella 0.4 is PING-PONG related
  - Scalability

    ✓Each query generates a huge amount of traffic

    - e.g. *TTL = 6; d = 10 ==> 10^6 messages*

    ✓Potentially, each query is received multiple times from all neighbors

Gnutella Conclusions

- Gnutella 0.6:
  - Superpeer-based organisation
  - Ping/pong caching
  - Query routing
- Summary:
  - A milestone in P2P computing

    ✓Gnutella proved that *full decentralization is possible*
  - But:
    - Gnutella is a patchwork of hacks
    - The ping-pong mechanism, even with caching, is just plain inefficient