Ciências ULisboa

# Programação em Sistemas Distribuídos

## MEI-MI-MSI

## 2018/19

3. Models of Distributed Computing

Prof. António Casimiro

# Distributed Computing Models

## Main models, neither exhaustive nor air-tight

- Client-Server (RPC, RMI, WWW) (Cliente-Servidor)
- Distributed Objects (Objectos Distribuídos)
- Distributed Shared Memory (DSM, Tuples) (Memória Partilhada Distribuída)
- Distributed Atomic Transactions (Transacções Atómicas Distribuídas)
- Message-oriented (Message Queue, Publish/Subscribe) (Orientado para mensagens, Fila de Mensagens, Editor/Assinante)
- **Stream (Corrente)**
- Group-Oriented (Orientado para Grupos)
- Peer-to-peer (Inter-pares)

# Stream

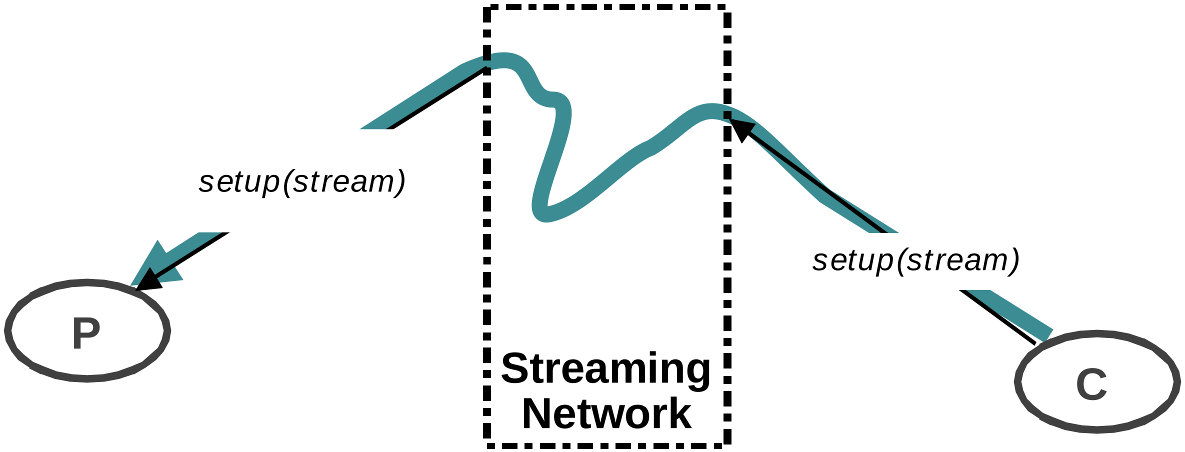
# Stream models
## Applications

- **Web-based multimedia**:
  - Best-effort quality of service for access to streams of audio and video data published via Web
  - Multimedia data retrieved from large online storage systems
  - Applications such as YouTube, Netflix and BBC iPlayer
- **Audio/Video conferencing**:
  - Full-duplex end-to-end audio/video conferencing
  - Applications such as Skype (Microsoft), Hangouts (Google), iChat (Apple)
- **Live streaming**:
  - Requires dedicated network bandwidth for good quality and dedicated video server
  - Dedicated or web-based clients

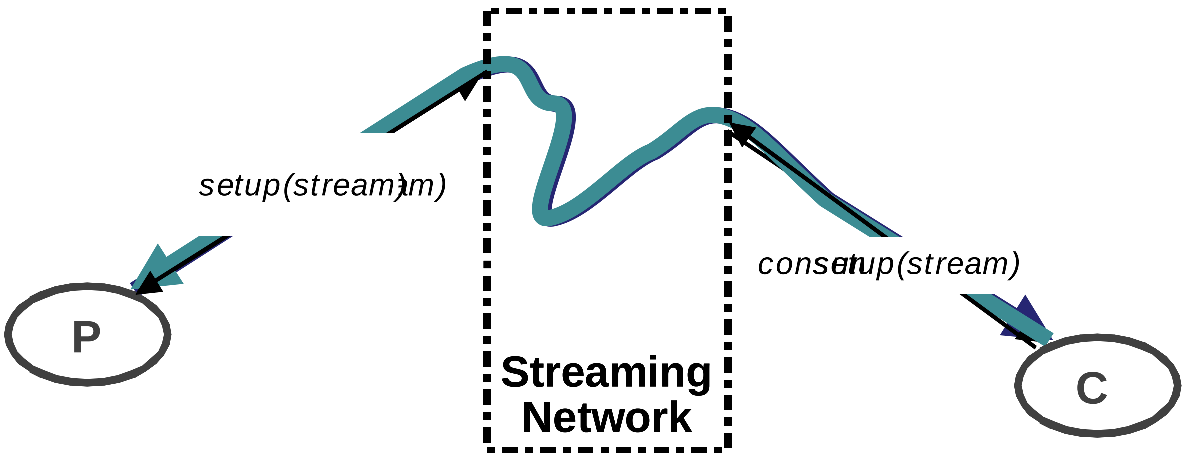
# Stream models

- Can be seen as **special case of channel-based publish/subscribe**
- Space decoupling
  - Source (producer) and sink (consumer) do not have to know each other
- Time coupling
  - Producer and consumer **must be simultaneously active**
  - Stream is a flow of information obeying **real-time constraints** from the end-user viewpoint
  - In content distribution networks (CDN), producer holds repository
- Synchronization coupling
  - Producers and consumers are synchronized to enforce the stream abstraction
- Stream delivery:
  - Stream is divided in ordered chunks, which are transmitted through the network
  - R/T message dissemination protocols enforcing message delivery deadlines make sure that each chunk arrives in time to be collated at the right place in the timeline
  - Client side buffering helps mitigating network latency jitter

# Streaming
## Information flow

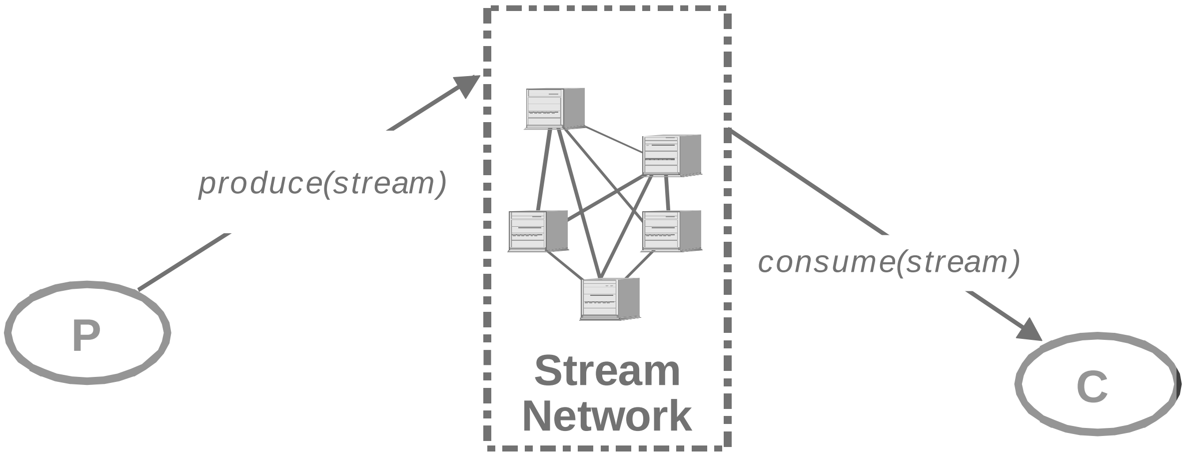


# Streaming
## Information flow



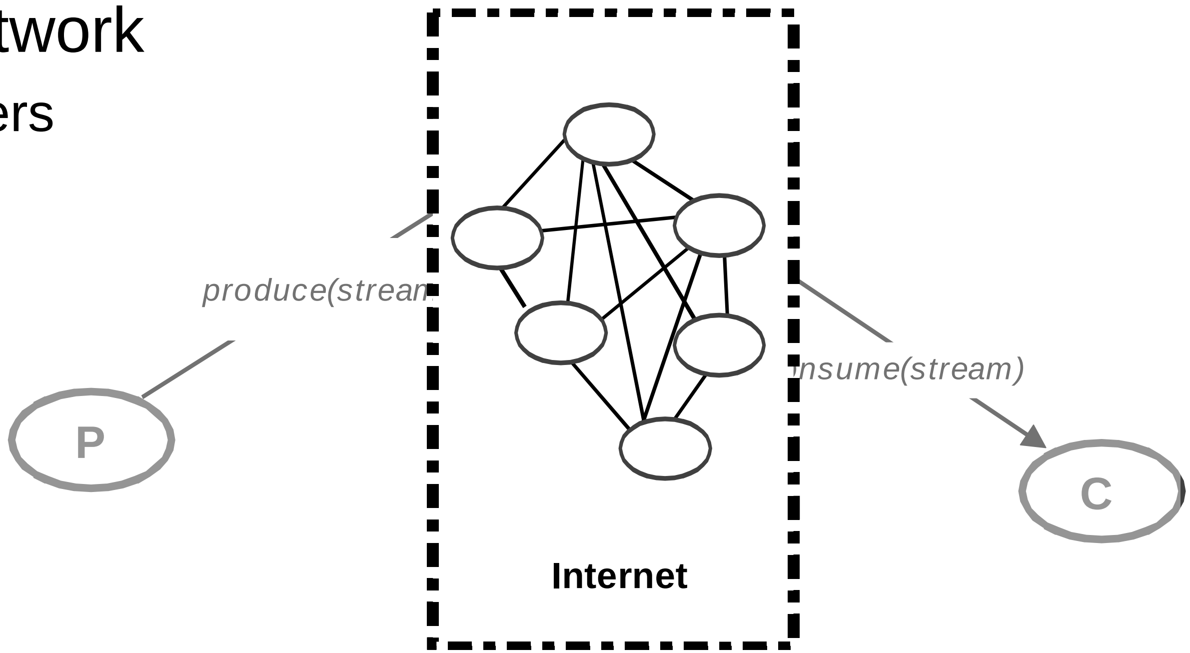
# Streaming Network
## Architectural alternatives

- Baseline Internet
  - R/T protocols (RTP)



# Streaming Network
## Architectural alternatives

- Baseline Internet
  - R/T protocols (RTP)
- R/T Overlay Network
  - streaming brokers



# Distributed Computing Models

## Main models, neither exhaustive nor air-tight

- Client-Server (RPC, RMI, WWW) (Cliente-Servidor)
- Distributed Objects (Objectos Distribuídos)
- Distributed Shared Memory (DSM, Tuples)  (Memória Partilhada Distribuída)
- Distributed Atomic Transactions (Transacções Atómicas Distribuídas)
- Message-oriented (Message Queue, Publish/Subscribe) (Orientado para mensagens, Fila de Mensagens, Editor/Assinante)
- Stream (Corrente)
- **Group-Oriented  (Orientado para Grupos)**
- Peer-to-peer (Inter-pares)

# Group-oriented

# Group-based communication

Recall:

- RPC is point-to-point and asymmetric
  - How to handle multi-participant interactions?
    - E.g. fragmented and/or replicated database
  - How to handle (symmetric) conversations?
    - E.g. email-group, video conversations, collaborative on-line file editing
- RPC is a remote operation, in closed-circuit, blocking, for reliability
  - How to achieve reliability in open-circuit?
  - E.g. with non-blocking but reliable operations

- **Group-based communication has interesting properties to solve the problems mentioned above**

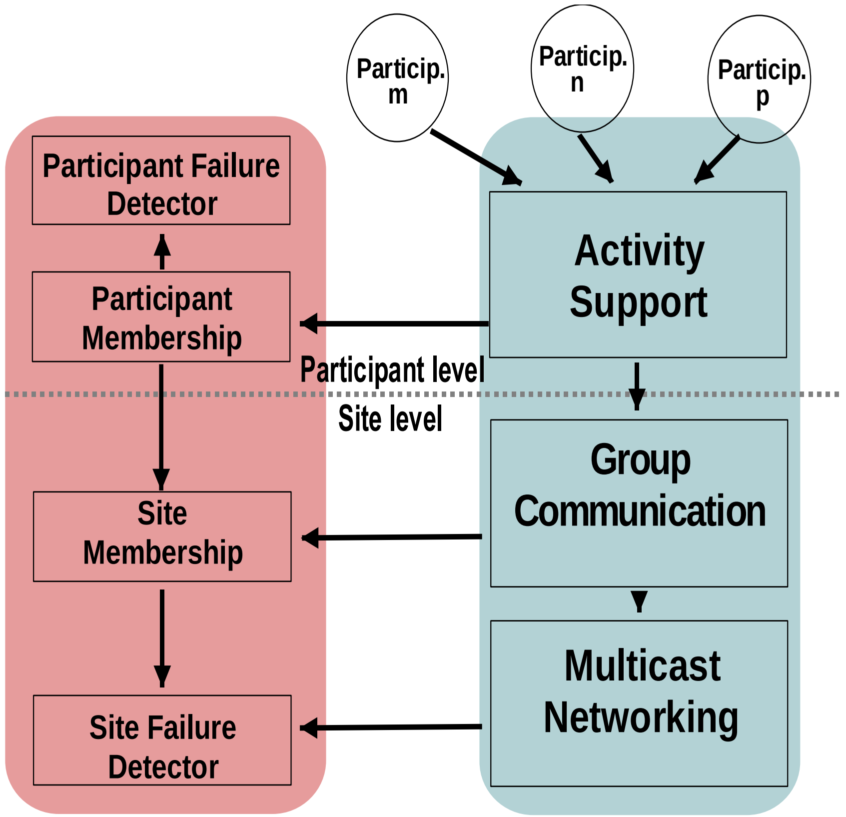
# Group-oriented systems
## Characteristics

- **Group**: set of entities that may receive messages
  - E.g. group of machines, group of processes
- **Group address**: the group is addressed collectively
  - E.g. list of addresses, multicast address, logical address
- **Group membership**: management of joining and leaving actions and information about membership
  - E.g. who is in the group (view), synchronization of join requests, failure detection (forced leave)
- **Reliable group communication**: usually with high QoS semantics
  - E.g. reliable delivery; ordered delivery; causal delivery

# Group-Oriented Systems

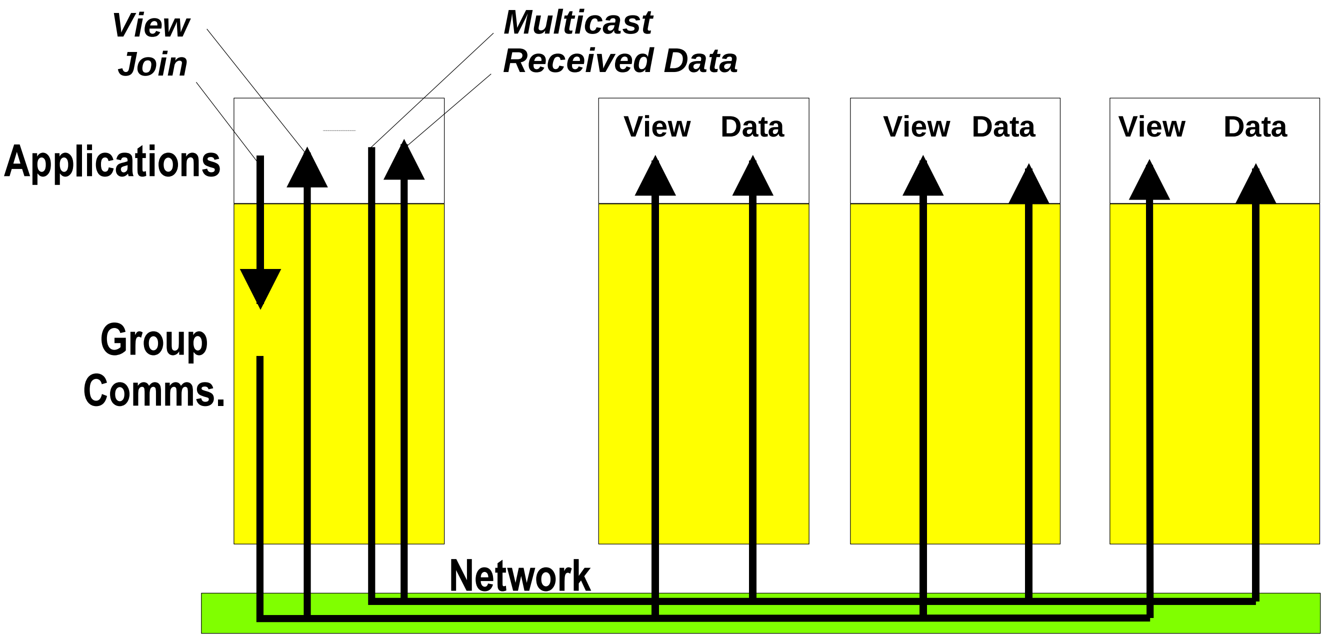
## Architecture - site vs. participant modules

- **Activity Support**
  - Provides support for distributed paradigms (e.g. key or replication management)
- **Participant Failure Detector**
  - Marks as unreachable the failed participants
  - Or participants at a site detected unreachable by Site Membership Service
- **Participant Membership**
  - Provides membership information at the participant level
- **Site Membership Service**
  - Information about sites participating in group communication
- **Group Communication**
  - Provides reliability and ordering guarantees to messages exchanged among sites
  - Uses Site Membership to obtain the list of active sites
- **Site Failure Detector**
  - Detects failure of other sites
- **Multicast Networking**
  - Supports unreliable message multicast services
  - Uses Site failure detection to update addressing



# Groups in action

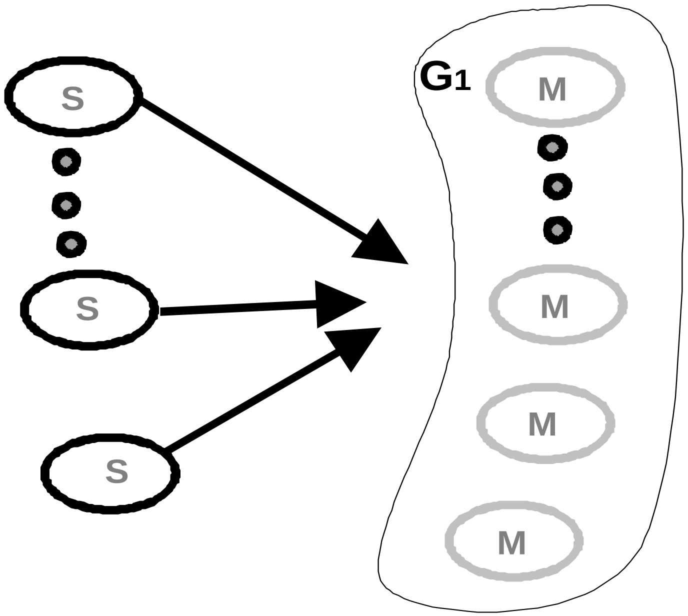
- Participant becomes member of a group in response to **join** request
- Participant receives the membership and **view** of the group
- Membership information is also **updated** at all the other members
- As member, participant can **send** to and **receive data** from the group

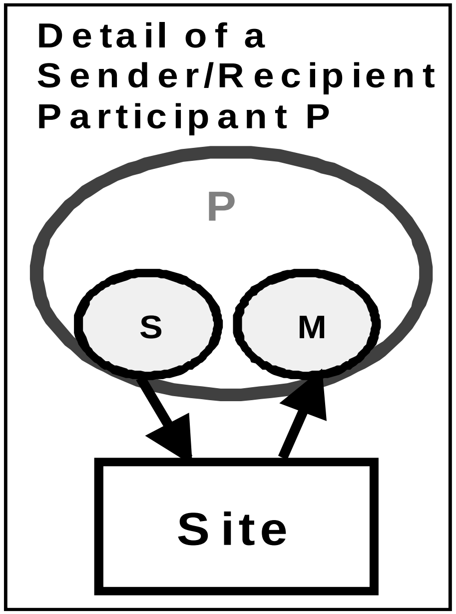


# Modelling group access
## Unidirectional interaction

- Unidirectional communication
  - Open-circuit
- Full-duplex group access
  - Circuit closed in the process

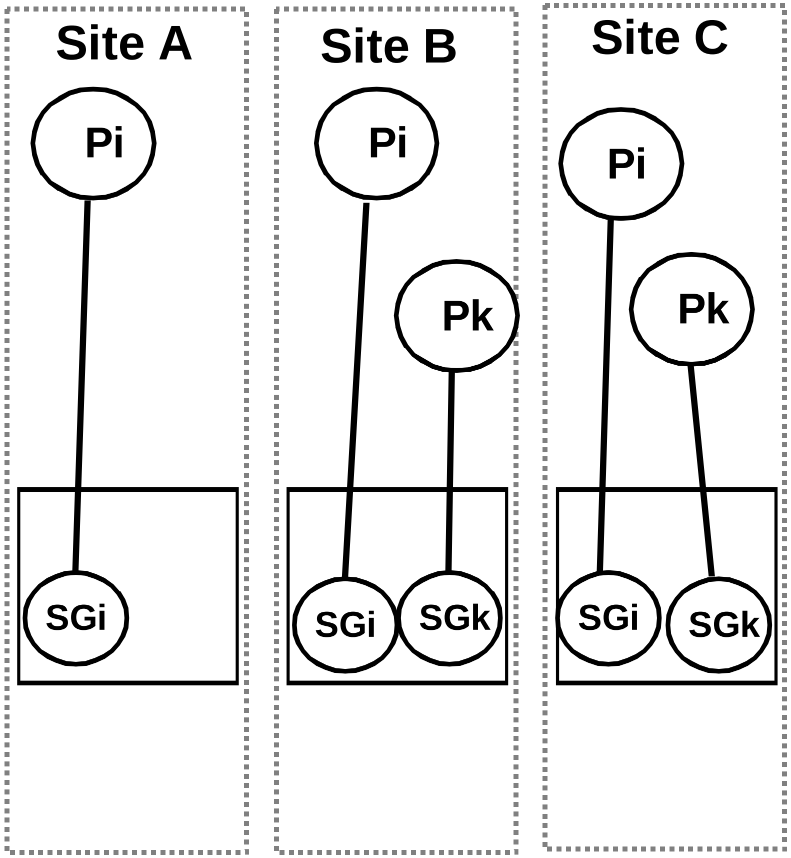
**Useful to distinguish *sender* and *member*; *members* receive messages**



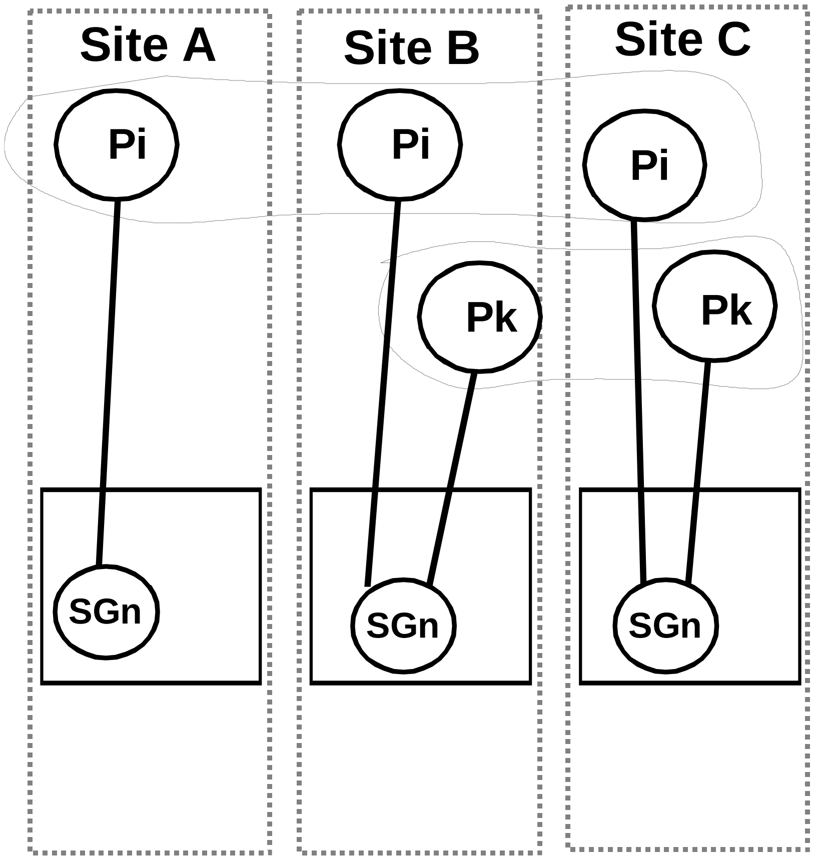
# Group access methods
## Normal

- Participant-level groups Pi and Pk, each supported by site-level groups SGi and SGk



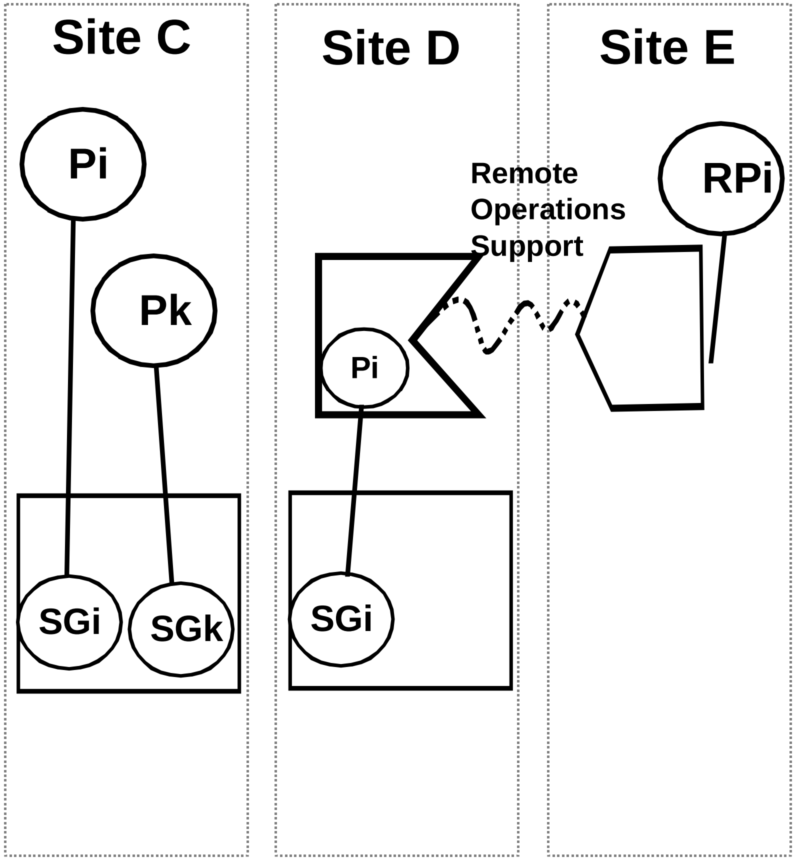
# Group access methods
## Lightweight

- Lightweight groups do resource sharing when several application-level groups have similar membership
- Pi and Pk are now supported by same site-group SGn



# Group access methods
## Remote

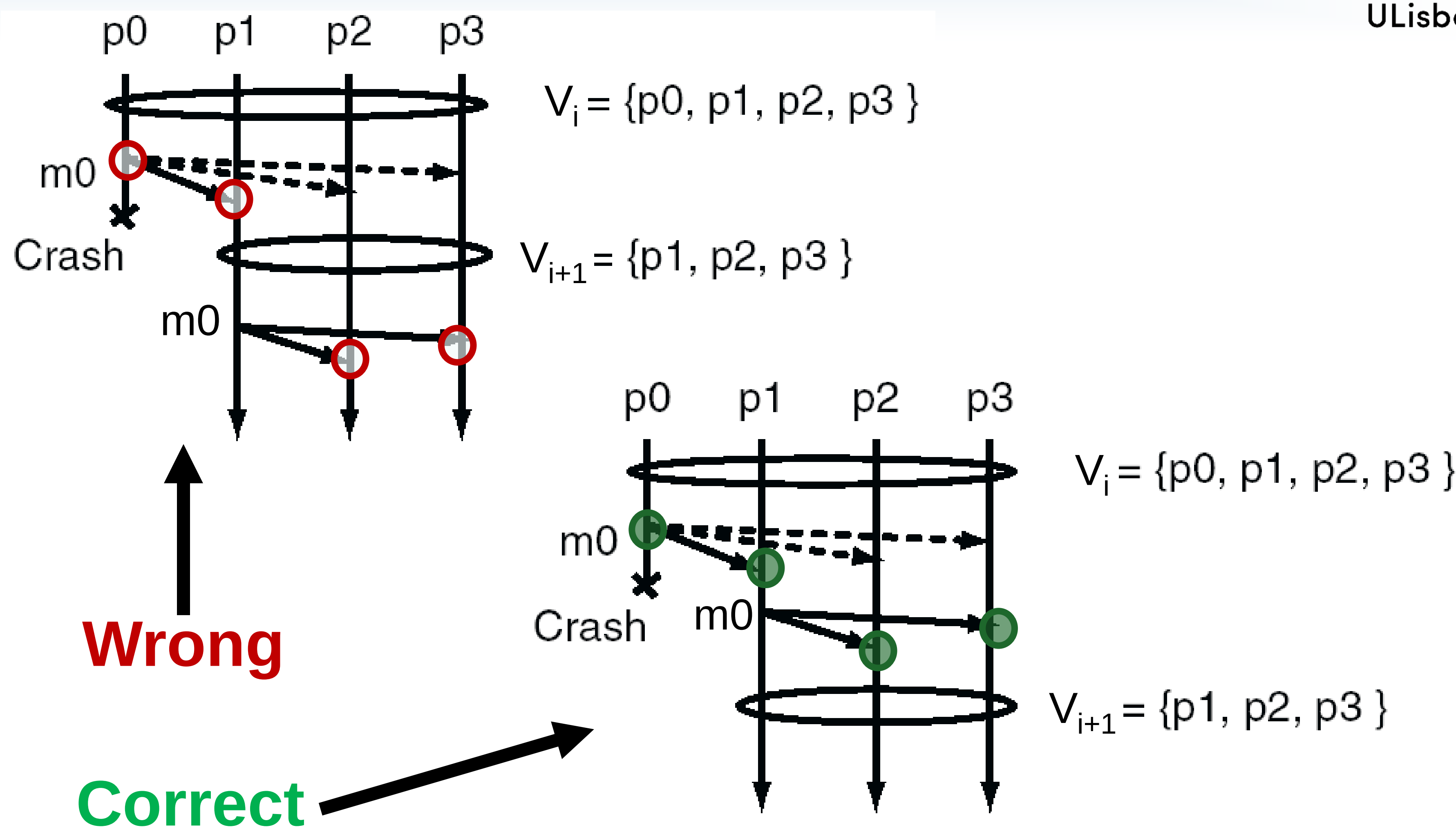
- Senders can be:
  - Groups members
  - Processes outside a group
- **Closed group**
  - Only allows group members to send to the group
- **Open group**
  - Allows non-members to send to the group, called detached senders
- Detached senders must interact with some proxy (group contact): a member or a sender to a group
- Remote sender RPi communicates with group Pi through proxy at Site D



# Group management

- Keep track of nodes or participants
- Dynamic group join/leave
- Failure detection
- Application support:
  - Short addresses (e.g. order number within group)
  - Join/leave/failure notification
  - Consistent group view: the same for all participants
  - View synchrony: total order between view changes and sets of delivered messages

# View synchrony

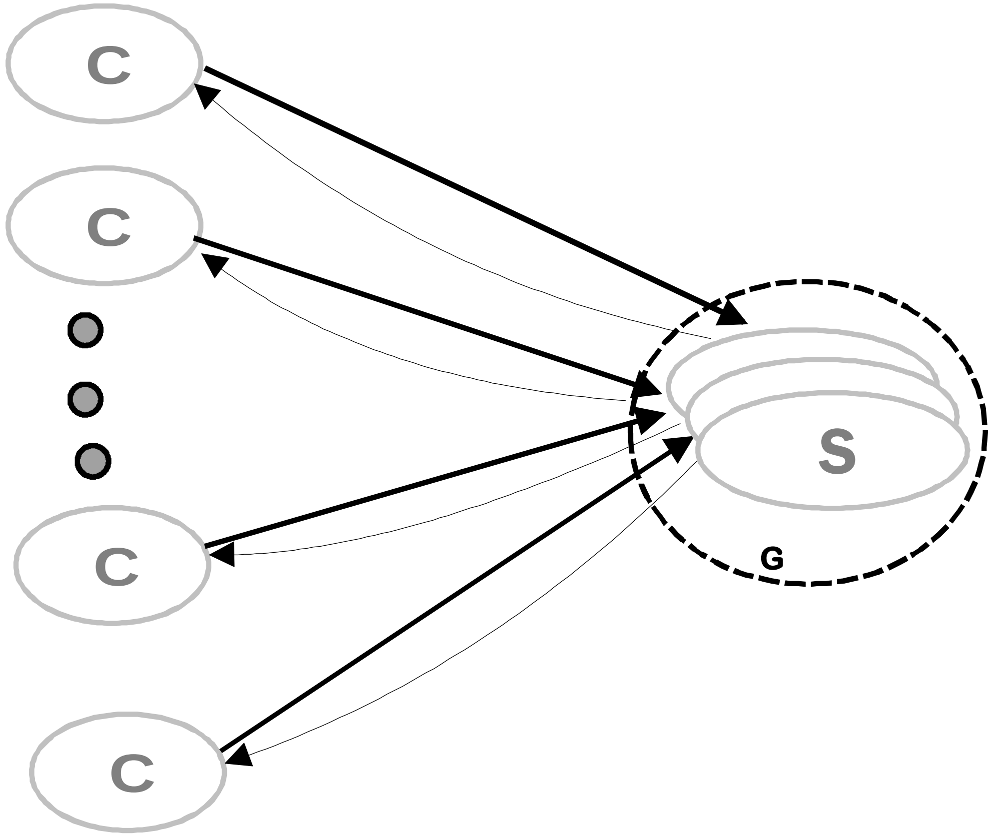


# Interaction models with groups

- Client-server
- Conversations
- Dissemination

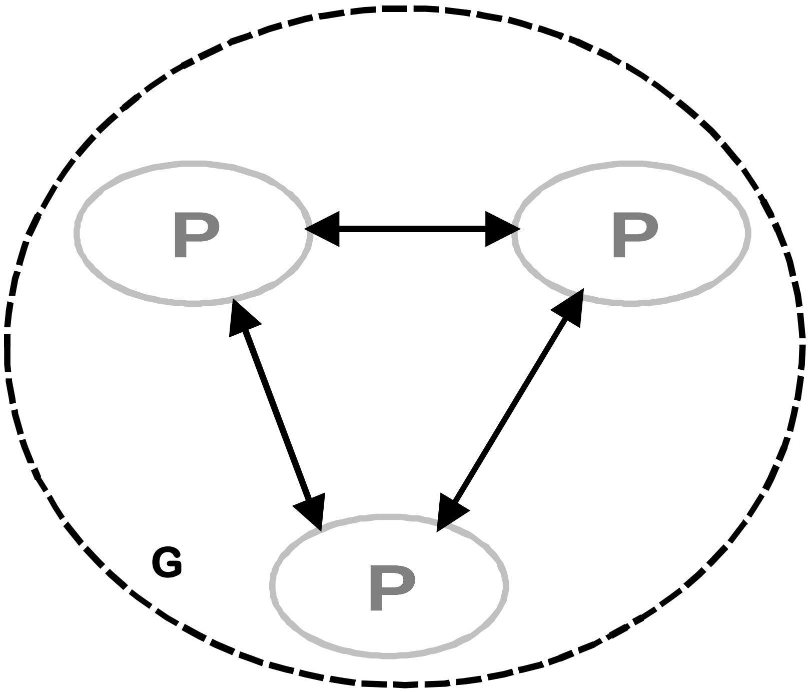
# Interaction models with groups
## Client-server

- Objectives:
  - Fault tolerance
  - Load balancing
- Characteristics:
  - Clients see **one** logical server, referred by the group address
  - Clients don't know location and number of physical servers
  - Consolidation in origin or destination



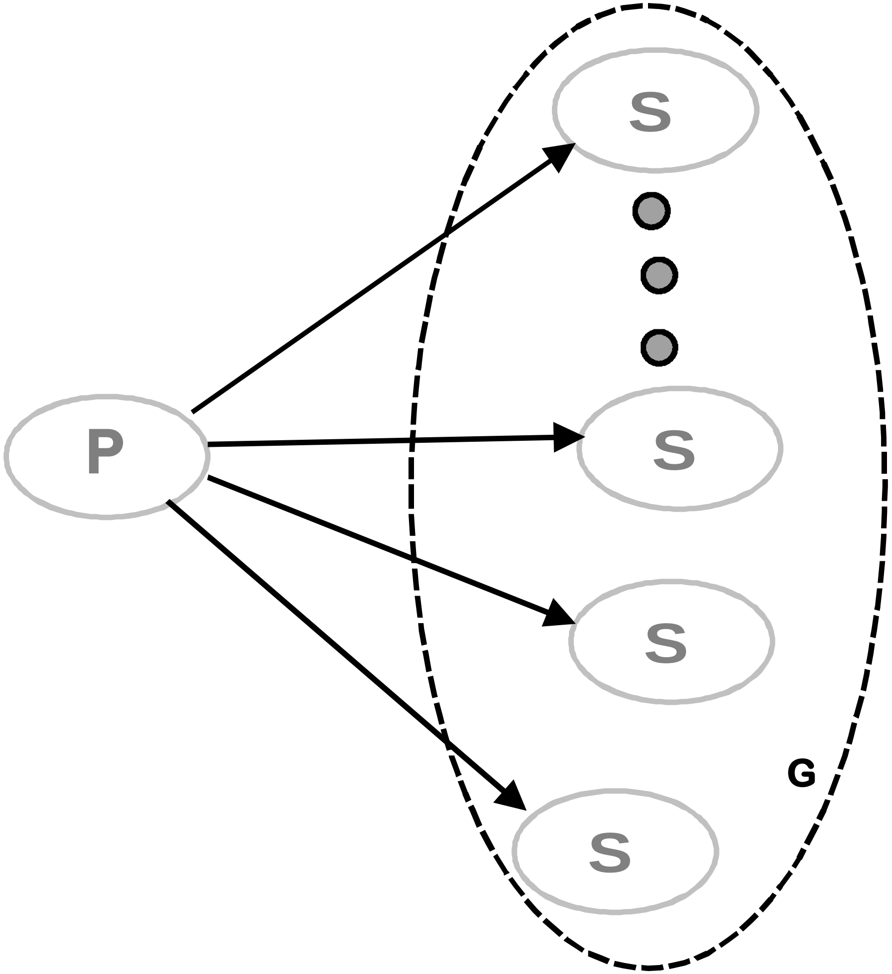
# Interaction models with groups
## Conversations

- Objectives:
  - Symmetric and peer-to-peer activities
- Characteristics:
  - Members with the same role
  - All send and all receive
  - Small groups achieve better QoS



# Interaction models with groups
## Dissemination

- Characteristics:
  - Producer sends messages to group of consumers
  - Producer sees a logical group of consumers
  - Producer does not need to know number and location of consumers (which might grow)
    - E.g. Multimedia transmission over the Internet, Event publishing



# Distributed Computing Models

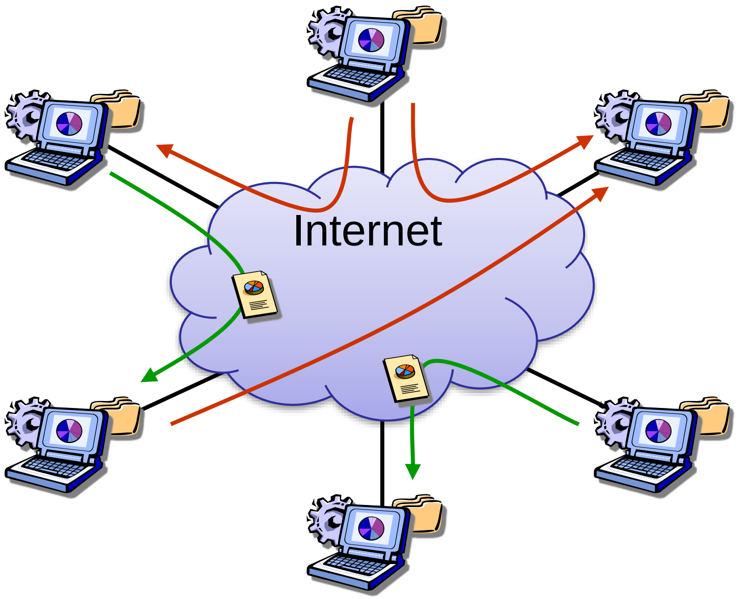
## Main models, neither exhaustive nor air-tight

- Client-Server (RPC, RMI, WWW) (Cliente-Servidor)
- Distributed Objects (Objectos Distribuídos)
- Distributed Shared Memory (DSM, Tuples)  (Memória Partilhada Distribuída)
- Distributed Atomic Transactions (Transacções Atómicas Distribuídas)
- Message-oriented (Message Queue, Publish/Subscribe) (Orientado para mensagens, Fila de Mensagens, Editor/Assinante)
- Stream (Corrente)
- Group-Oriented  (Orientado para Grupos)
- **Peer-to-peer (Inter-pares)**

# Distributed
# Peer-to-Peer (P2P)

# Main characteristics of P2P systems

- Distributed
- Decentralized
- Self-organizing
- Dynamic
- Resilient
- Anonymous
- Load balanced
- Symmetric roles
- Resource-sharing



# Distributed Peer-to-Peer Model

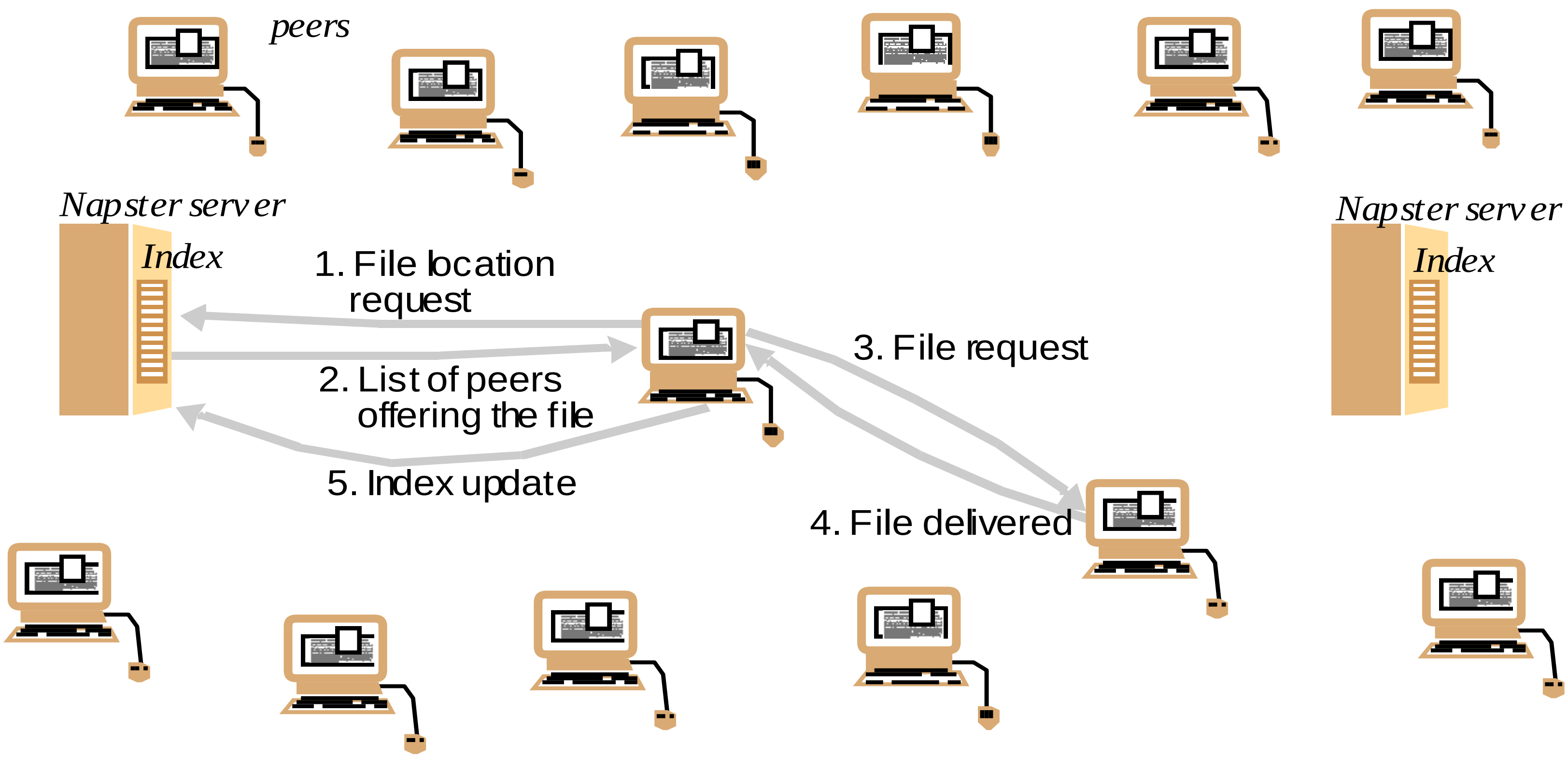
- Computing with an unreliable and heterogeneous large collection of nodes, with unusual properties:
  - Are **symmetric** in role, unlike Client-Server model
  - Contribute themselves resources: bandwidth, content, storage, memory, CPU (like clients can also be servers and even routers)
- No centralized control, **self-organizing**
  - Nodes are autonomous from central servers or administration
  - Communicate and collaborate directly with each other (not through well-known servers or official routes)
  - Dynamic environment, **frequent join and leave (churn)**
- Inherently resilient
  - No central single point of failure
  - Geographic separation
  - Replication for fault-tolerance easily included and adapted
- Nodes may have widely varying capabilities
  - Storing, content delivery, computing, etc.

# Canonical P2P application examples
## A snapshot

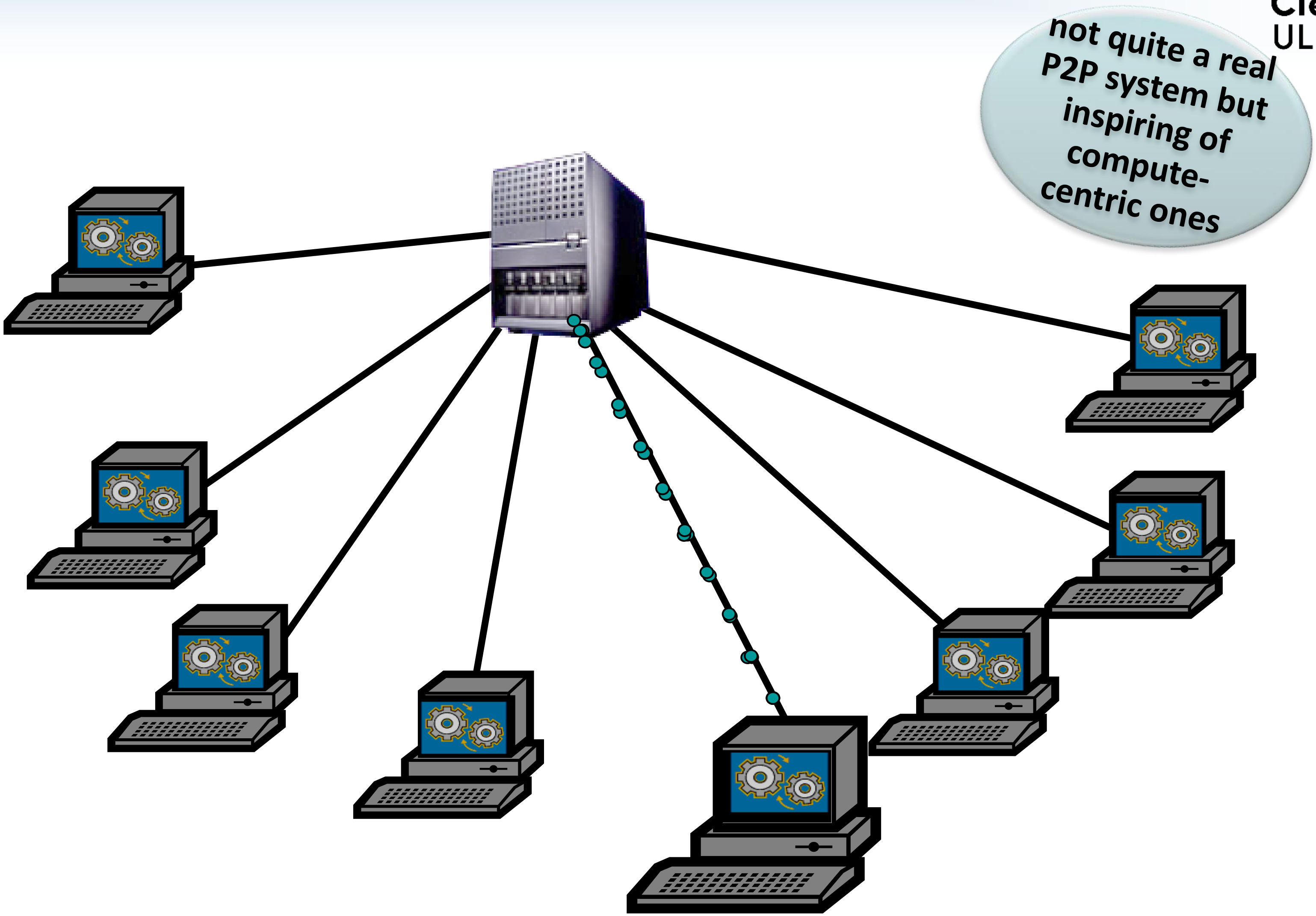
- Napster
- SETI@HOME
- Video Streaming
- Gnutella

# Napster
## Peer-to-peer file sharing

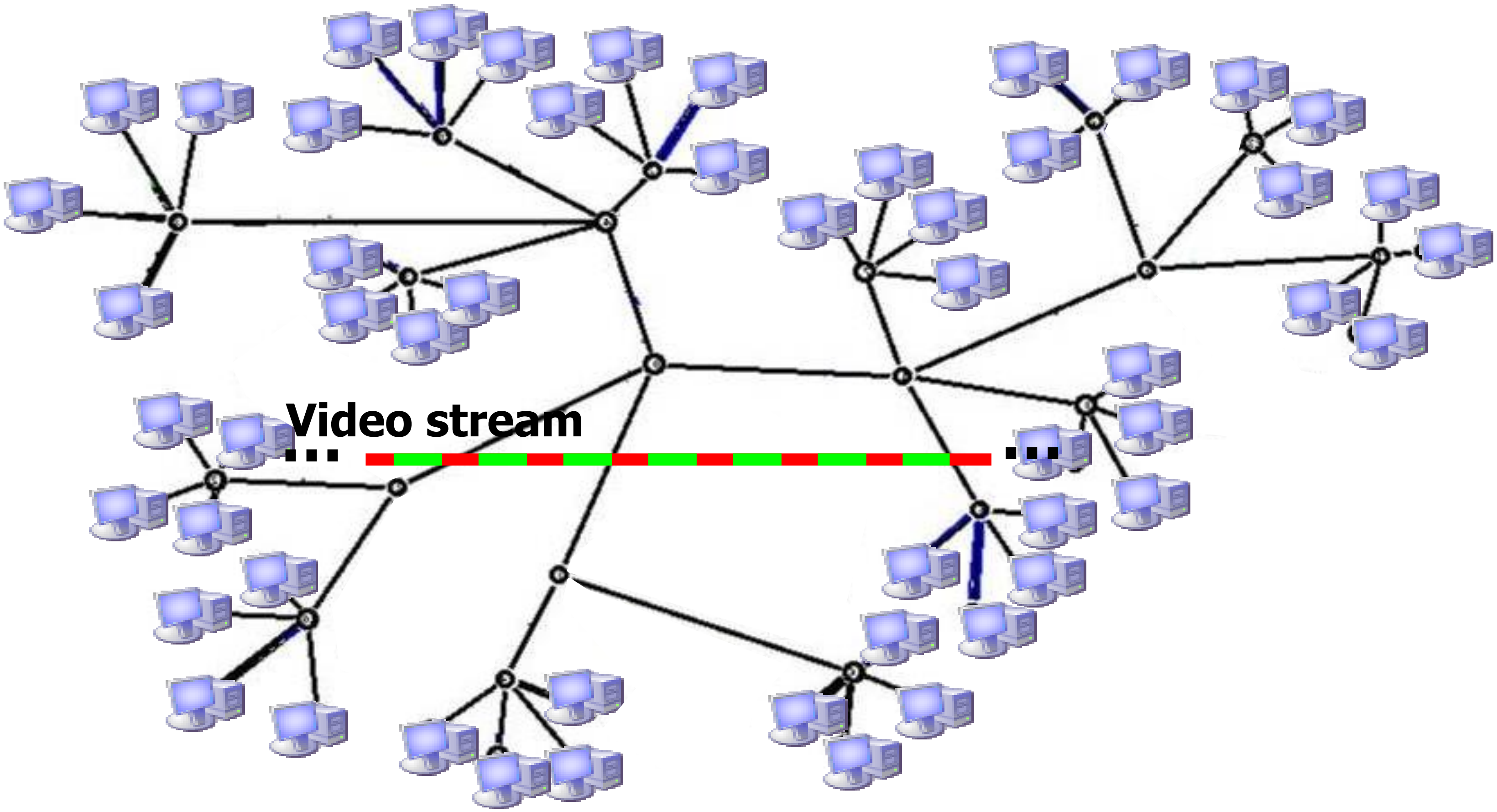
- Centralized, replicated index
- Decentralized edges



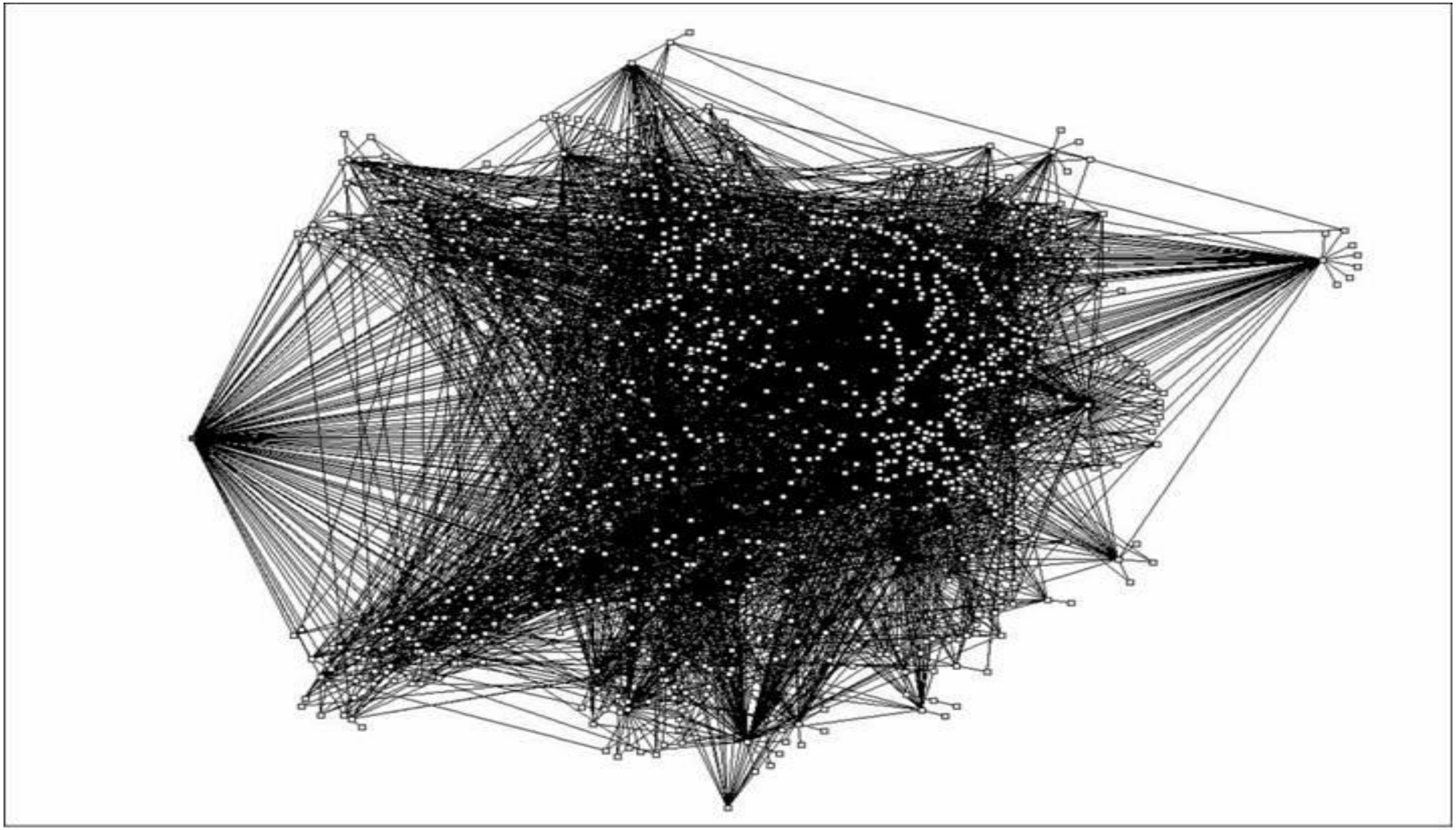
# SETI@Home - Example



# Peer-to-Peer Video Streaming



# Topology of a Gnutella Network

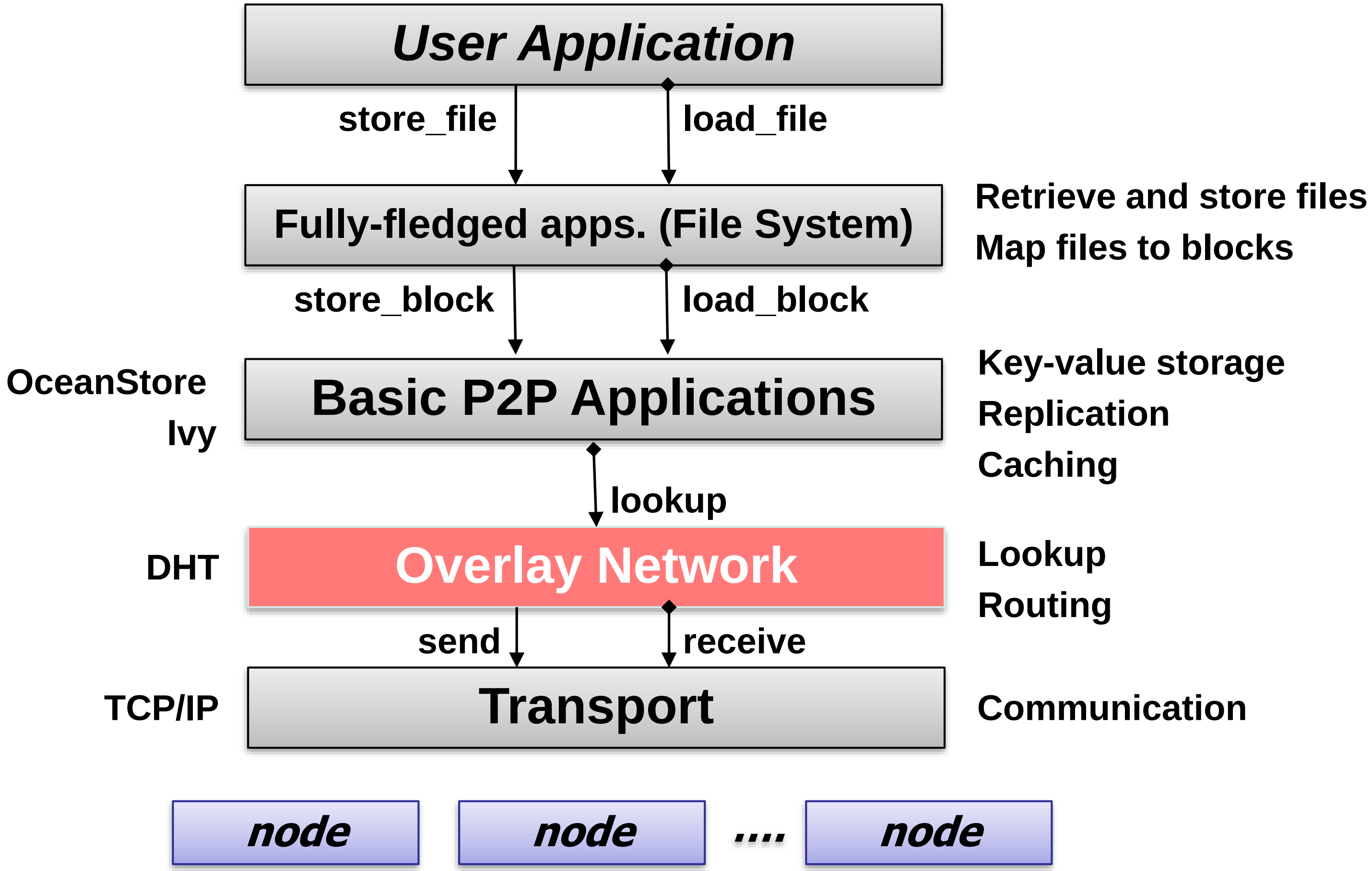


Source: Mihajlo A. Jovanovic, Fred S. Annexstein, and Kenneth A. Berman, Laboratory of Networks and Applied Graph Theory, University of Cincinnati.

# Peer-to-Peer System components

- Peer
  - An end-system, computer, end-user, application, etc.
- Overlay Network or Routing Overlay
  - Abstract (application-level) network maintained by nodes of a P2P system, keeping track of neighbours and/or a routing table
- P2P middleware
  - Intermediate layer for application-independent management of distributed P2P resources
- Leecher
  - A peer that uses and contributes resources (e.g. data)
- Seed
  - A peer that only contributes resources (e.g. data)

# P2P system architecture

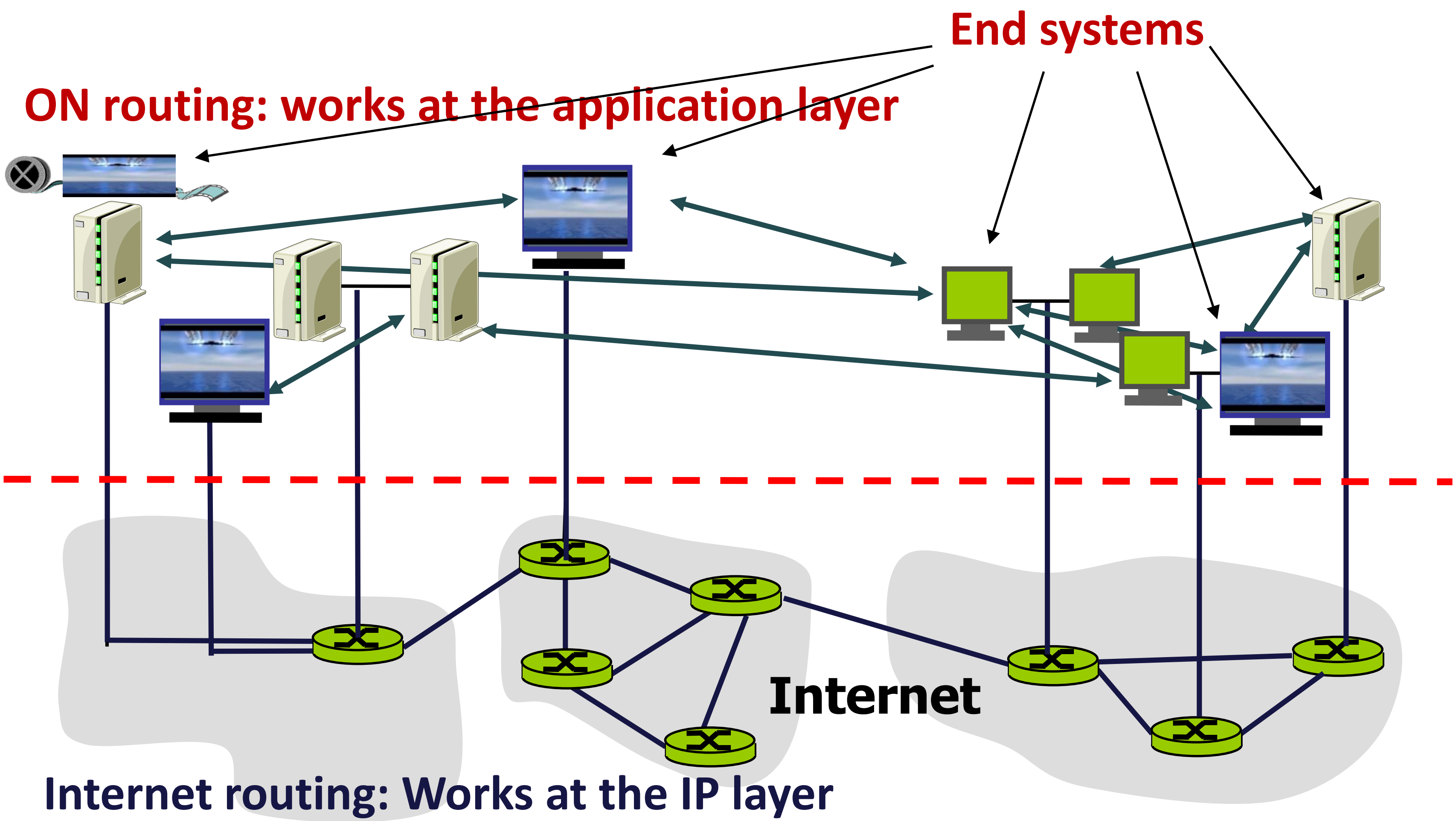


# P2P systems vs. Overlay Networks

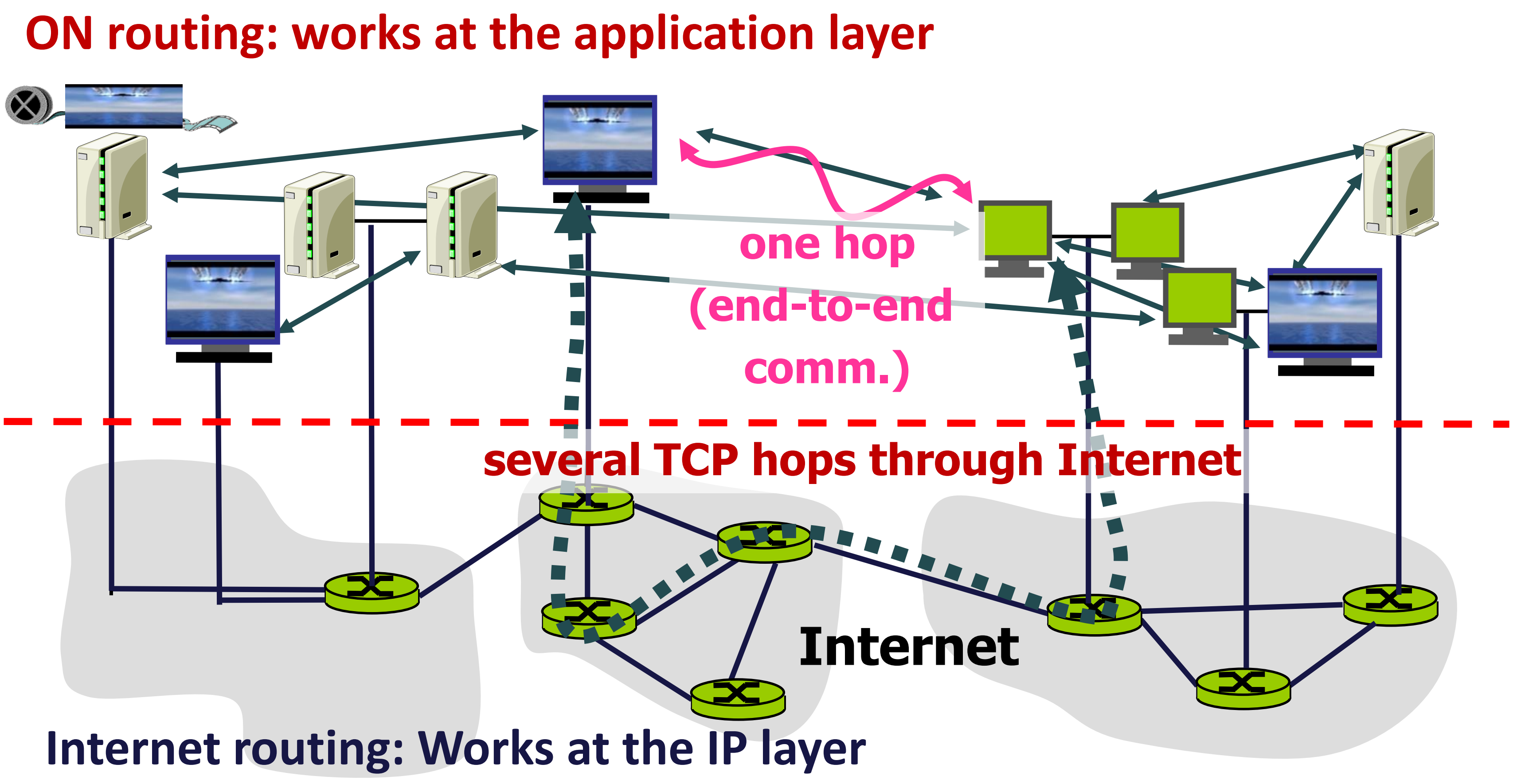
- A P2P system always resorts to an overlay network
- But an overlay network is not always a P2P system
- In P2P, an overlay network is formed by end-systems at application layer
- Overlay paths may be longer than IP paths (route "stretch")

**Routing stretch: the ratio between the ON distance (nr of ON hops) and the direct IP distance (nr of IP hops) of an ON route between x and y**

# P2P systems vs. Overlay Networks



# P2P systems vs. Overlay Networks



# Classification of P2P systems

- **Hybrid (also called centralized)** – Preserves some of the traditional C/S architecture. A central server links between clients, stores indices tables, etc
  - Napster
- **Unstructured** – no control over topology and resource (e.g. file) placement
  - Gnutella, Morpheus, Kazaa, etc
- **Structured** – topology is tightly controlled and placement of files is not random
  - Chord, CAN, Kademlia, PAST, Pastry, Tapestry, Tornado, BitTorrent, etc

# Unstructured vs Structured P2P

| | Structured | Unstructured |
|---|---|---|
| PRO : | Guaranteed to locate objects (assuming they exist);<br>Can offer time and complexity bounds on this operation;<br>Relatively low message overhead | Self-organizing and naturally resilient to node failure |
| CON : | Need to set up and maintain often complex overlay structures, which can be difficult; and costly to advertise/ discover resources;<br>Gets worse in highly dynamic environments (churn and load) | Only offers probabilistic guarantees on locating objects;<br>Excessive messaging overhead can affect scalability |