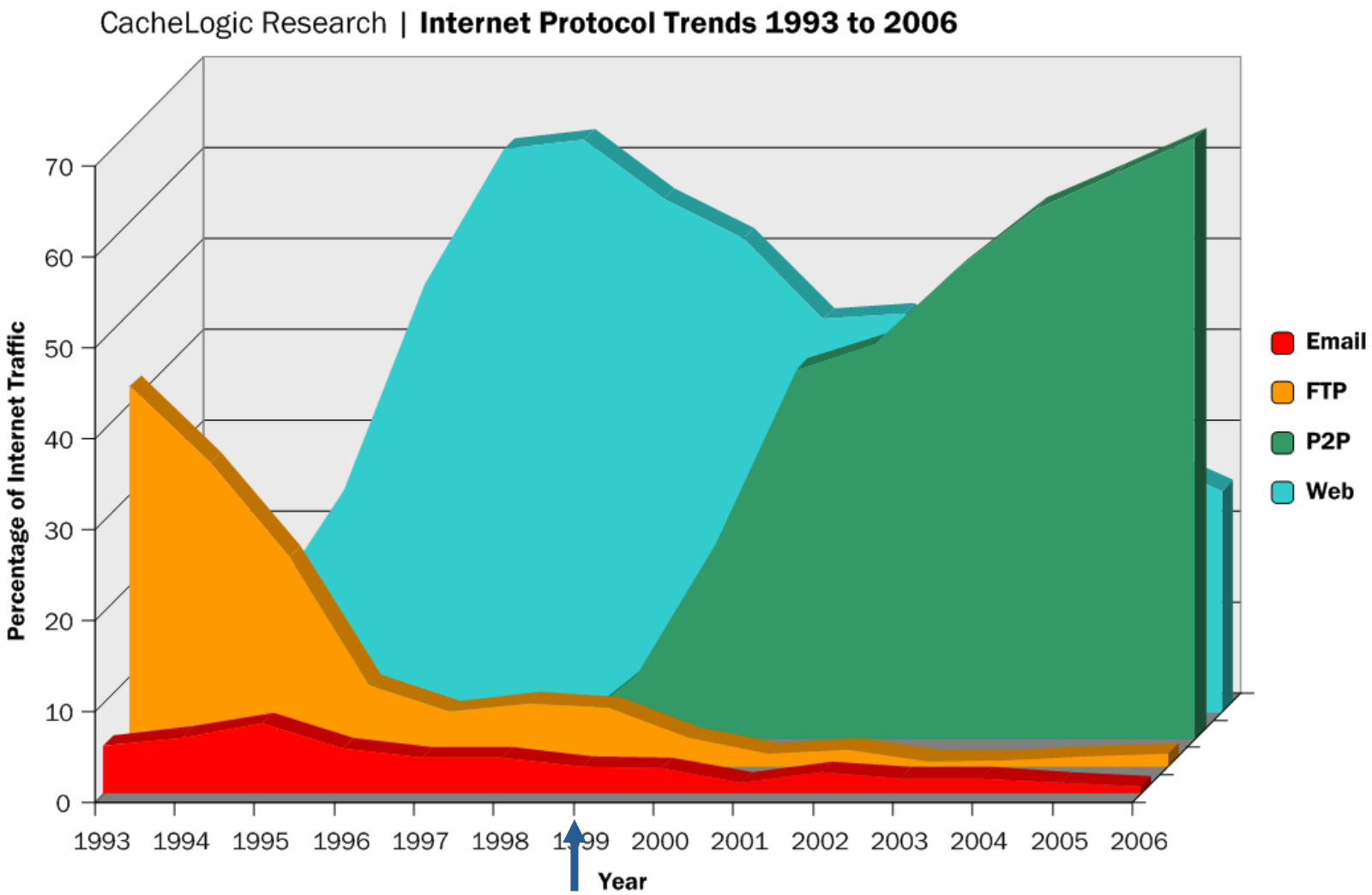
Overlays of Peer to Peer Systems

Franck Petit
Lip6, UPMC

Mainly based on materials by:
- Yang Guo and Christoph Neumann, Corporate Research, Thomson Inc.
- Jon Crowcroft, Univ. of Cambridge
- Jim Kurose, Brian Levine, Don Towsley, UMASS
- Matthew Allen, Univ. of California Santa Barbara
- Manan Rawal, Bhaskar Gupta
- Olivier Marin

- History, motivation and evolution
  - History: Napster and beyond
  - What is Peer-to-peer?
  - Why Peer-to-peer?
- Brief P2P technologies overview
  - Unstructured P2P-overlays
  - Structured P2P-overlays

History, motivation and evolution



P2P represented ~65% of Internet Traffic at end 2006

- 1999: *Napster*, first widely used P2P-application

1999: *Napster*, first widely used P2P application

The application:

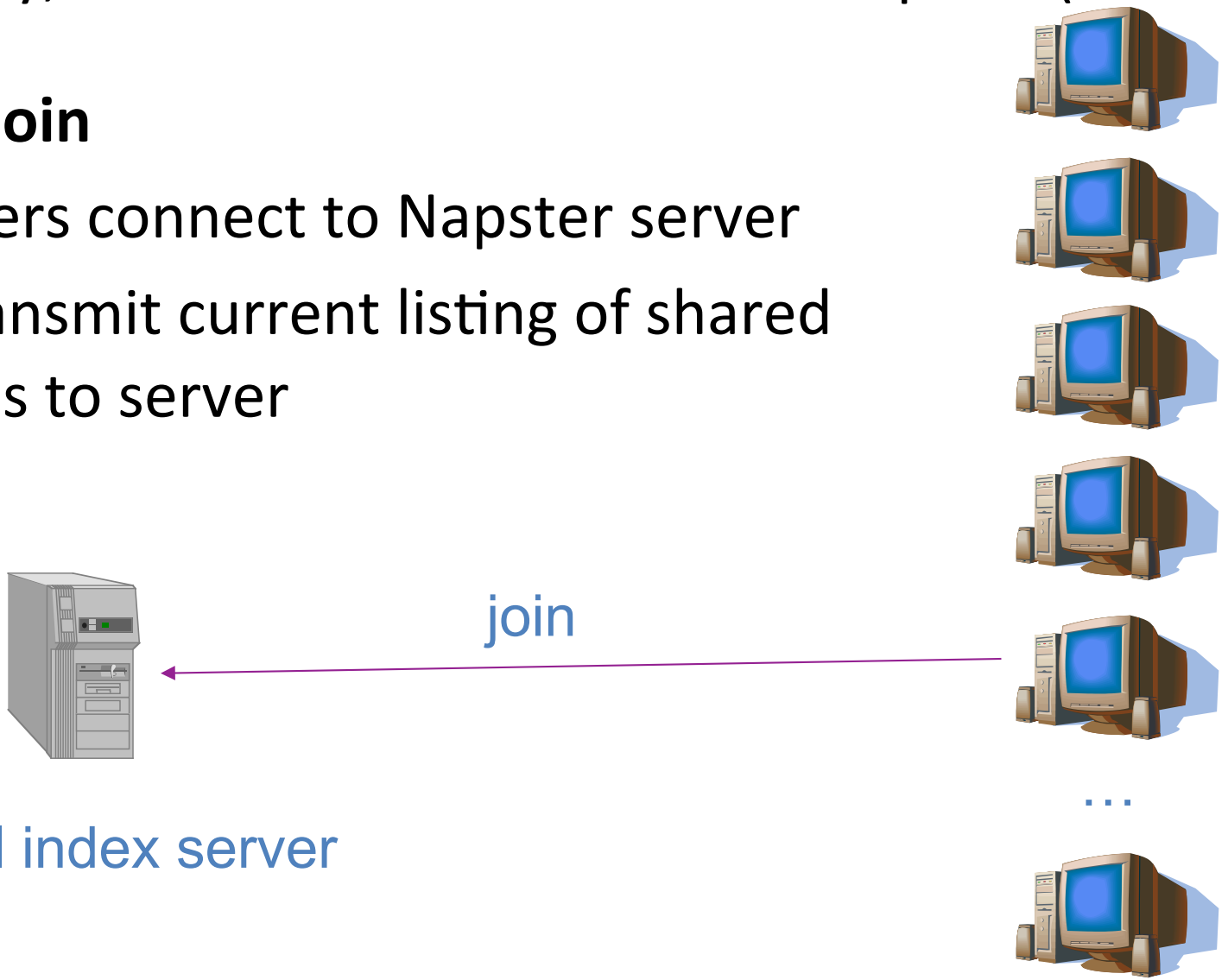
- A P2P application for the distribution of mp3 files
  - Each user can contribute its own content

How it works:

- Central index server
  - Maintains list of all active peers and their available content
- Distributed storage and download
  - Client nodes also act as file servers
  - All downloaded content is shared

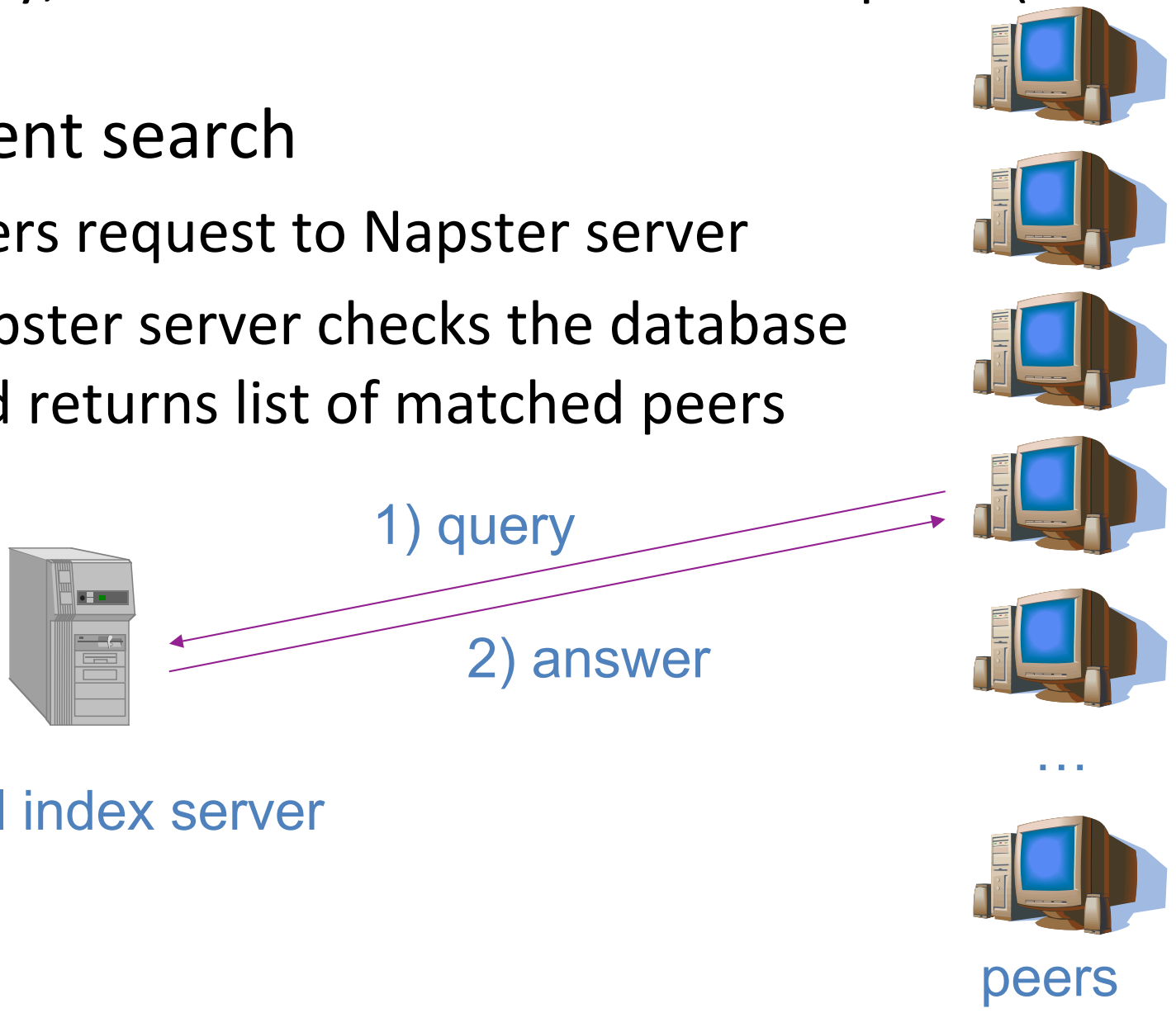
History, motivation and evolution - Napster (cont'd)

- **Initial join**
  - Peers connect to Napster server
  - Transmit current listing of shared files to server



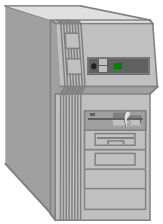
History, motivation and evolution - Napster (cont'd)

- Content search
  - Peers request to Napster server
  - Napster server checks the database and returns list of matched peers

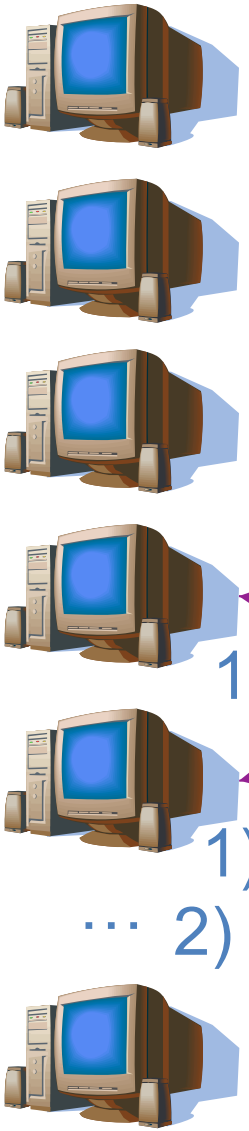


History, motivation and evolution - Napster (cont'd)

- File retrieval
  - The requesting peer contacts the peer having the file directly and downloads the it



Central index server



History, motivation and evolution - File Download

- Napster was the first simple but successful P2P-application. Many others followed…

P2P File Download Protocols:

- 1999: Napster (closed in 2001)
- 2000: Gnutella, eDonkey
- 2001: Kazaa (closed in 2012)
- 2002: eMule, BitTorrent

- History, motivation and evolution
  - History: Napster and beyond
  - What is Peer-to-peer?
  - Why Peer-to-peer?
- Brief P2P technologies overview
  - Unstructured P2P-overlays
  - Structured P2P-overlays

Definition of Peer-to-peer (or P2P)

- *DEFINITION 1* (early 2000s):
  A peer-to-peer (or P2P) computer network is a network that relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating it in a relatively small number of servers.
- *DEFINITION 2* (2009-2016):
  A peer-to-peer (or P2P) is any **distributed network architecture** composed of **participants** that make a portion of their resources (such as processing power, disk storage, or network bandwidth) directly available to other network participants, without the need for central coordination instances.

Taken from the wikipedia free encyclopedia - www.wikipedia.org

Definition of Peer-to-peer (or P2P)

- *DEFINITION 3* (2016-):
  Peer-to-peer (P2P) computing or networking is a distributed application architecture that **partitions tasks or workloads between peers**. Peers are equally privileged, equipotent participants in the application. They are said to form a peer-to-peer network of nodes.

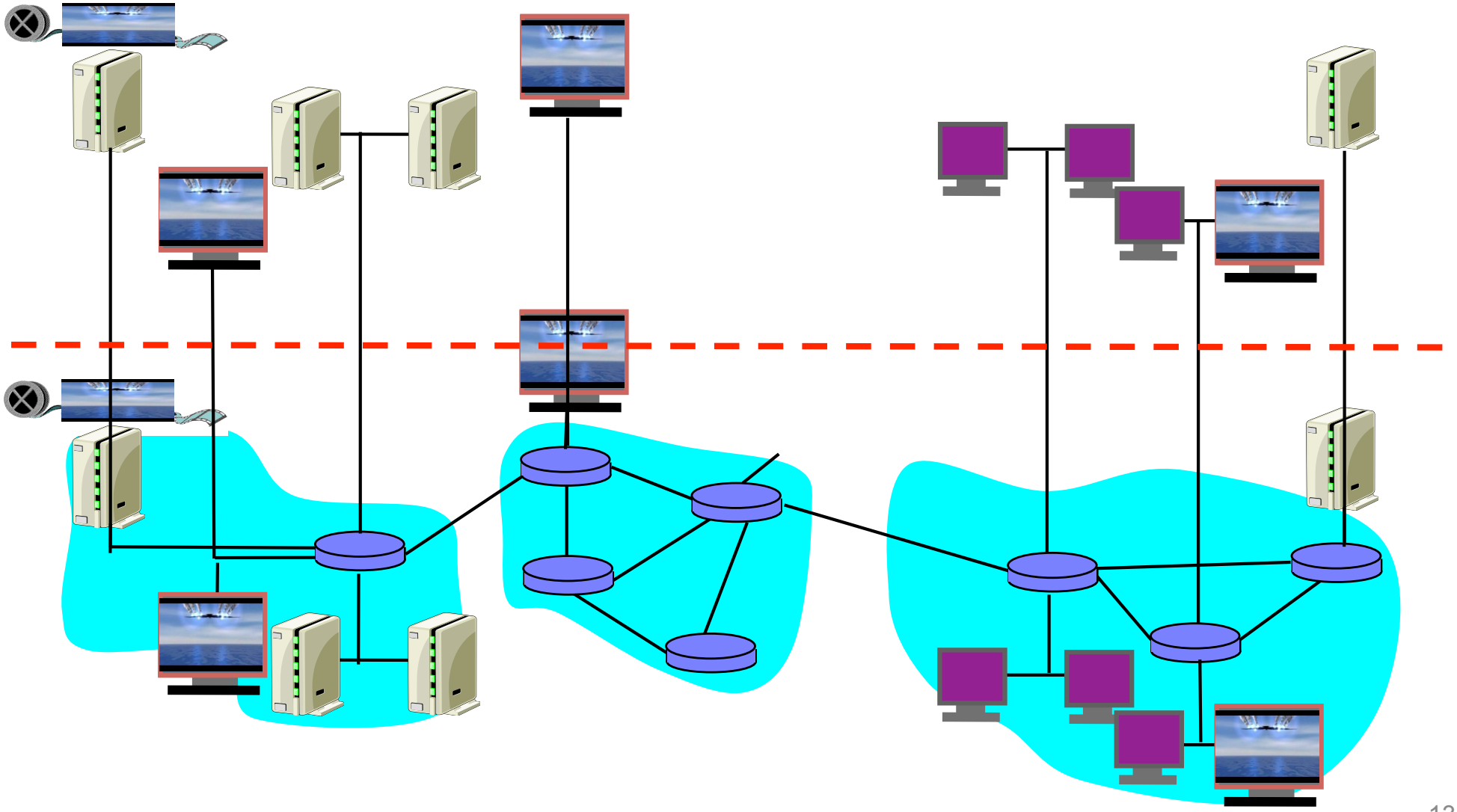
Taken from the wikipedia free encyclopedia - www.wikipedia.org

Peer to peer systems

- Assumed to be with **no distinguished role**
- So no single point of bottleneck or failure
- However, this means they need **distributed algorithms** for
  - Connection protocol
  - Service discovery (name, address, route, metric, etc)
  - Neighbour status tracking
  - Application layer routing (based possibly on content, interest, etc)
  - Resilience, handling link and node failures
  - etc

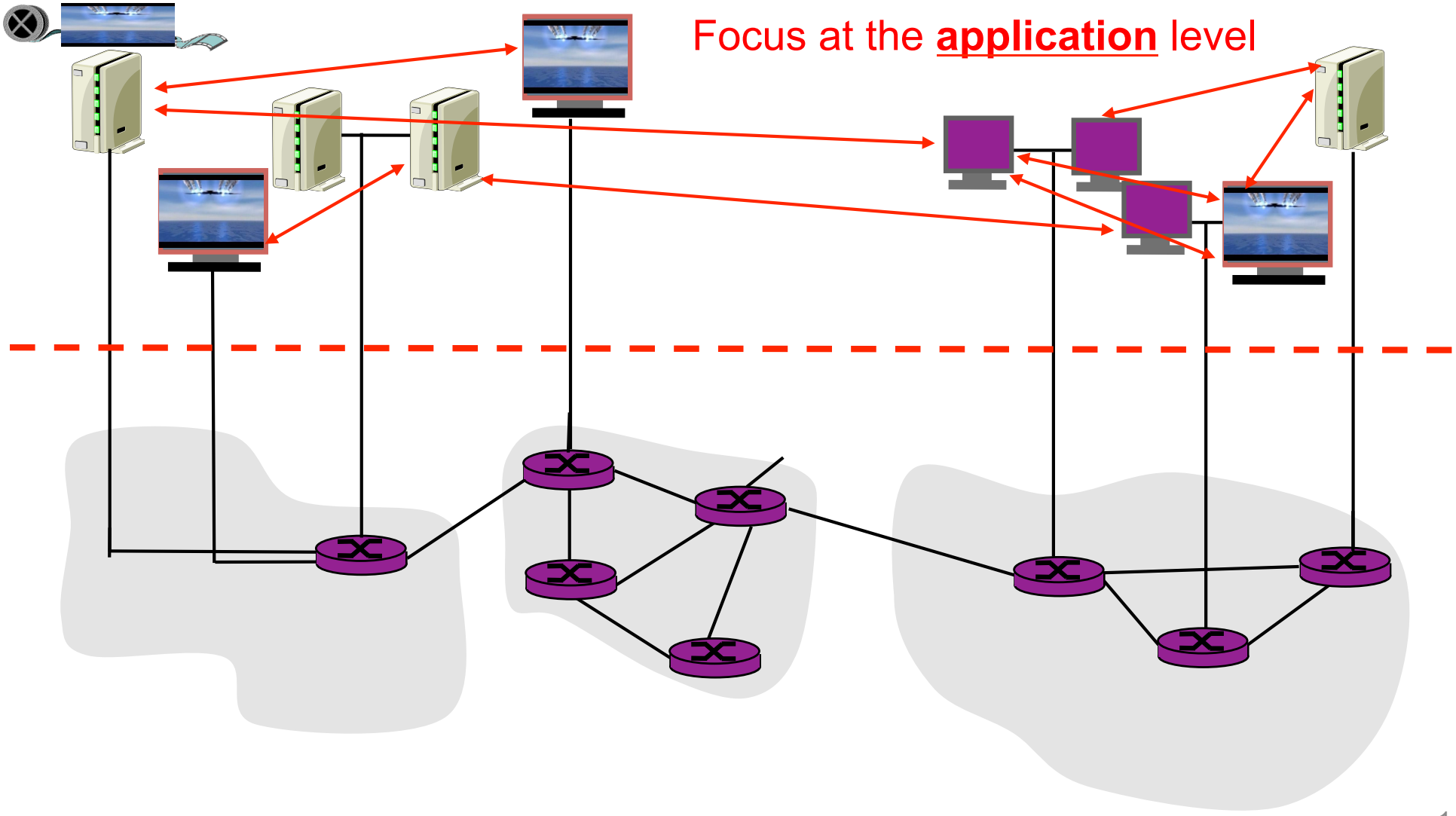
Overlays and peer 2 peer systems

- P2P systems rely on **overlays** structure

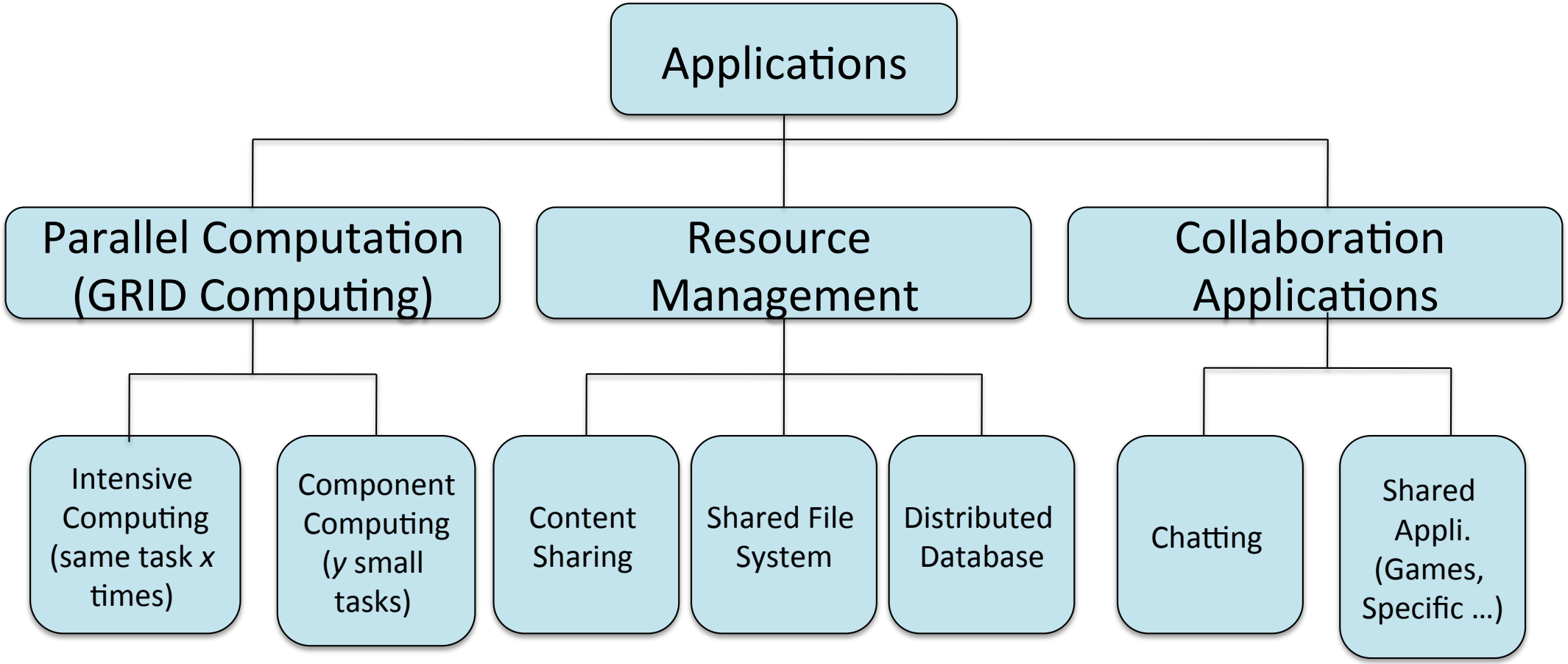


Overlays and peer 2 peer systems

- P2P systems rely on **overlays** structure



Taxonomy of Applications



Taxonomy of Applications

- P2P-File download
  - Napster, Gnutella, KaZaa, eDonkey, Bittorrent…
- P2P-Communication
  - VoIP, Skype, Messaging, …
- P2P-Video-on-Demand
- P2P-Computation
  - GRID, scientific computation (seti@home, XtreemOS, BOINC,…)
- P2P-Streaming
  - PPLive, End System Multicast (*ESM*),…
- P2P-Gaming
  - WOW, City of Heroes,…

History, motivation and evolution - Applications

- P2P is not restricted to file download!

P2P Protocols:

- 1999: Napster, End System Multicast (ESM)
- 2000: Gnutella, eDonkey
- 2001: Kazaa
- 2002: eMule, BitTorrent
- 2003: Skype
- 2004: PPLive
- 2006: TVKoo, TVAnts, PPStream, SopCast, Video-on-Demand, Gaming
- 2009: Bitcoin (Blockchain)

Application type:

File Download

Streaming

Telephony

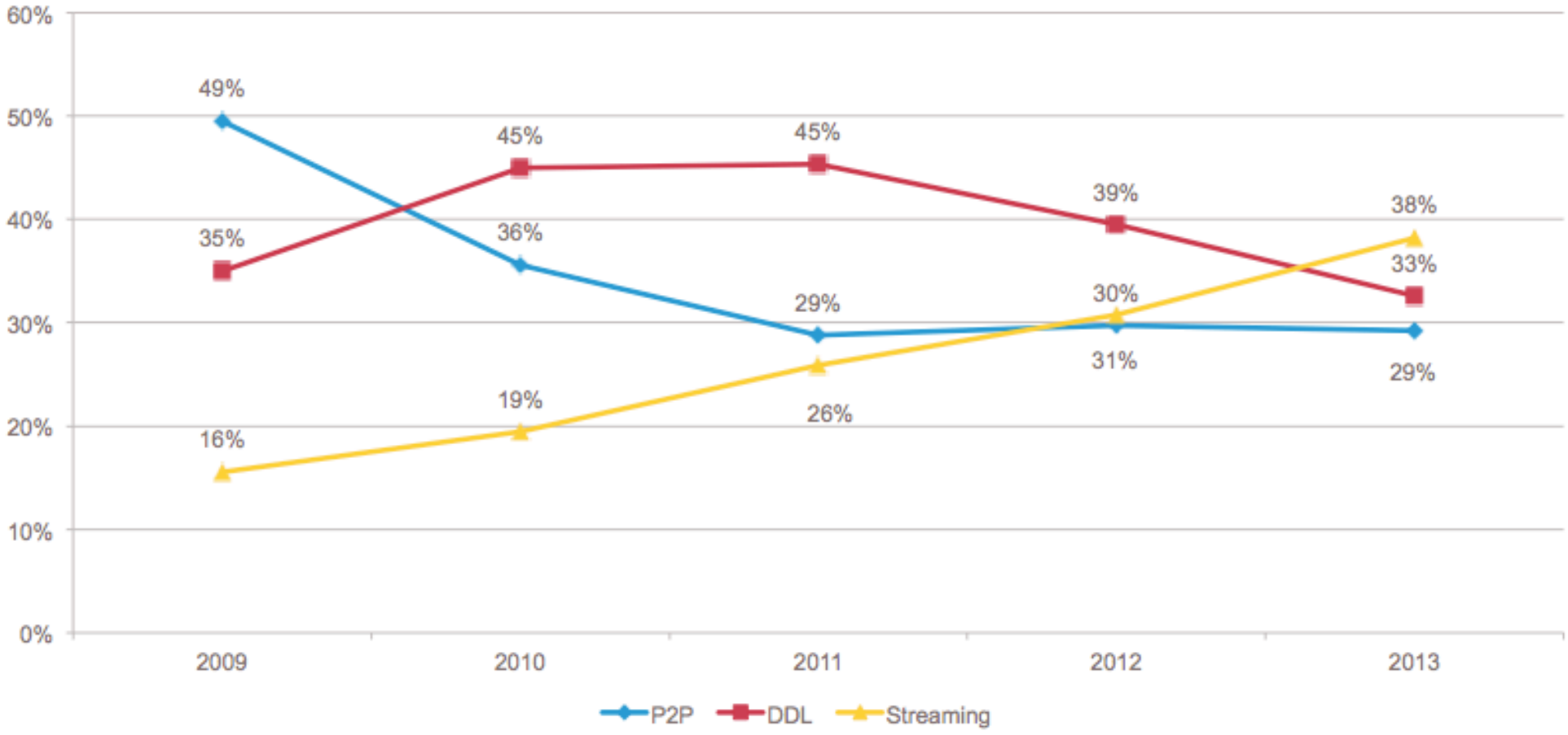
Video-on-Demand

Gaming

Virtual Money

Evolution

Évolution de la répartition des usages (nombre de pages vues) par protocole depuis 2009

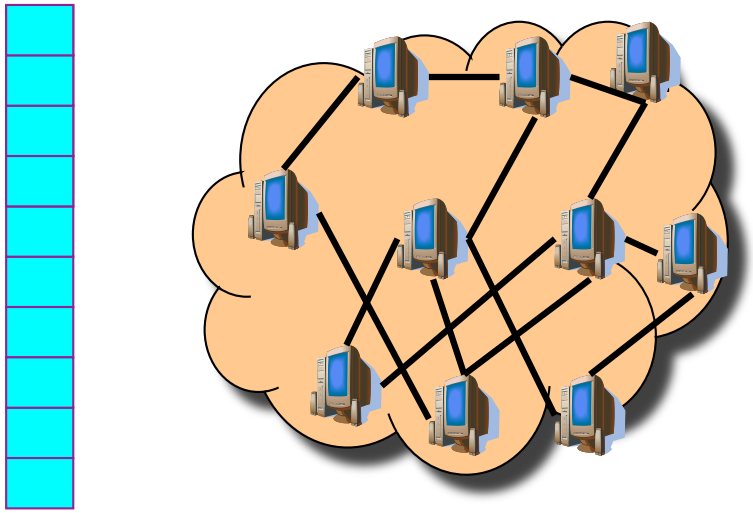


- History, motivation and evolution
  - History: Napster and beyond
  - What is Peer-to-peer?
  - Why Peer-to-peer?
- Brief P2P technologies overview
  - Unstructured P2P-overlays
  - Structured P2P-overlays

Why is P2P so successful?

- Scalable – It's all about sharing resources
  - No need to provision servers or bandwidth
  - Each user brings its own resource
  - *e.g.* resistant to flash crowds
    - flash crowd = a crowd of users all arriving at the same time

capacity



Resources could be:

- Files to share;
- Upload bandwidth;
- Disk storage;…

Why is P2P so successful? (cont'd)

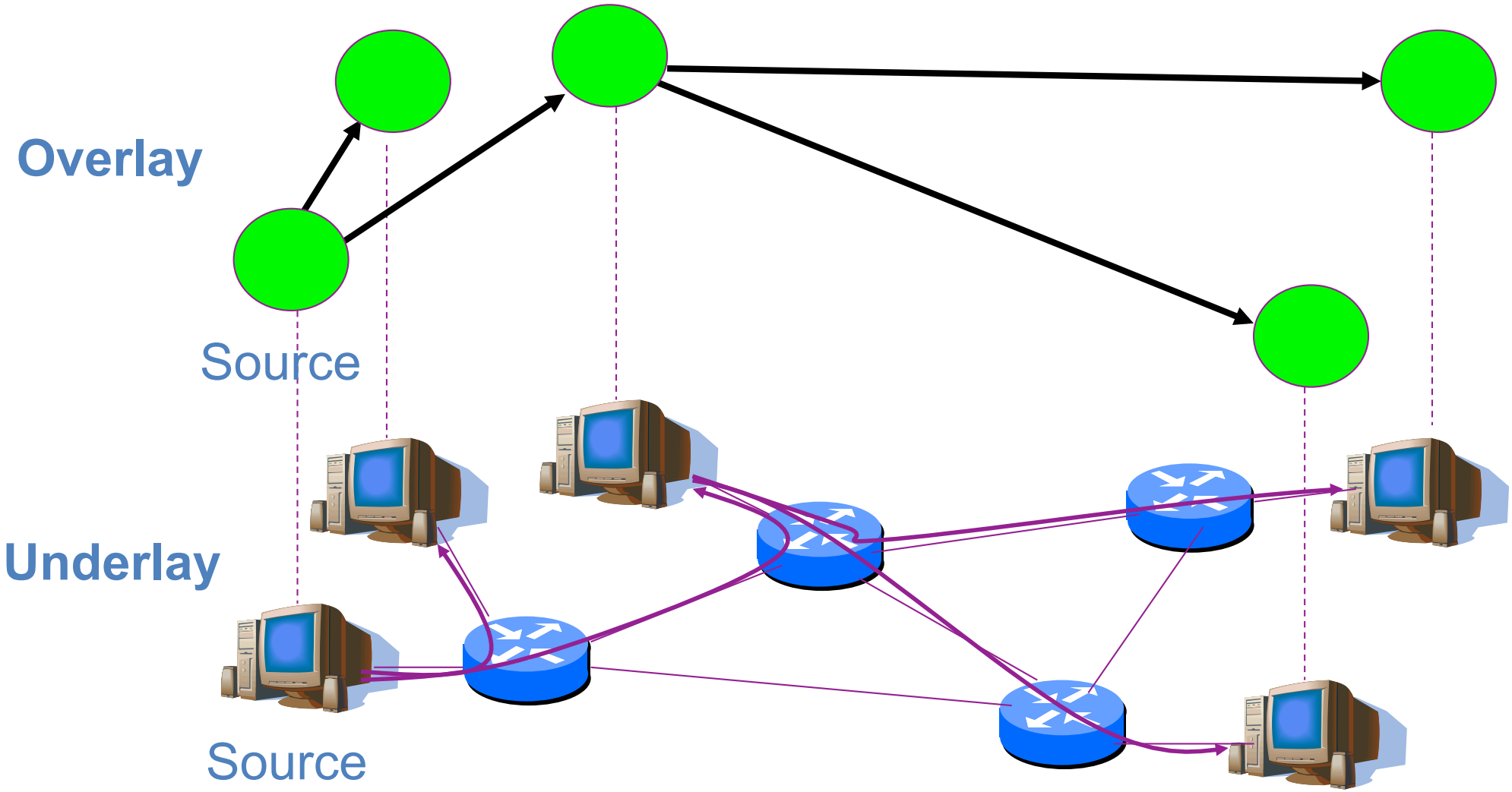
- Cheap - No infrastructure needed
- Everybody can bring its own content (at no cost)
  - Homemade content
  - Ethnic content
  - Illegal content
  - But also *legal* content
  - …
- High availability – Content accessible most of the time (replication)

- History, motivation and evolution
  - History: Napster and beyond
  - What is Peer-to-peer?
  - Why Peer-to-peer?
- Brief P2P technologies overview
  - Unstructured P2P-overlays
  - Structured P2P-overlays

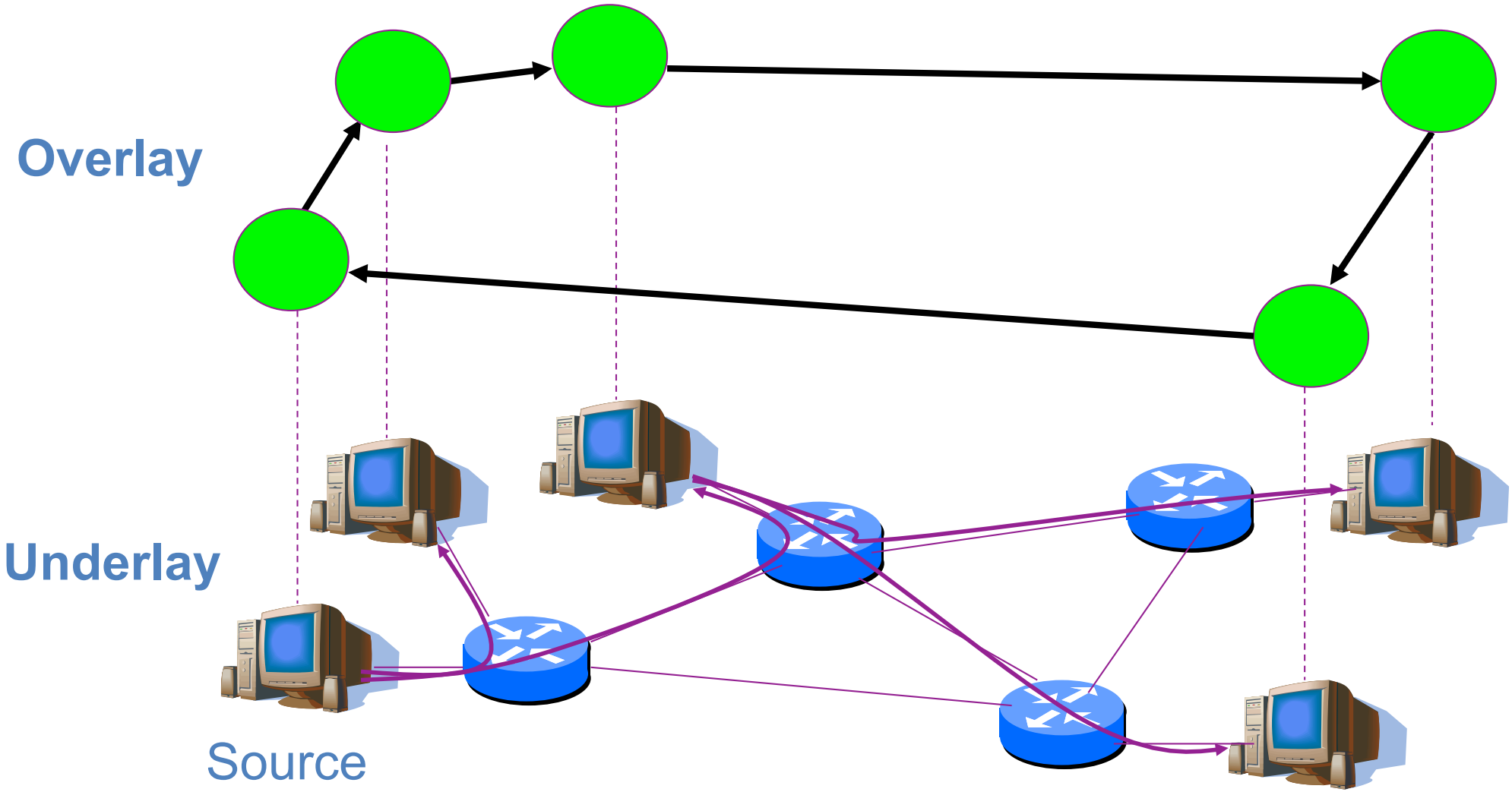
P2P-Overlay

- Build graph at application layer, and forward packet at the application layer
- It is a *virtual* graph
  - Underlying physical graph is transparent to the user
  - Edges are TCP connection or simply a entry of an neighboring node's IP address
- The graph has to be continuously maintained (e.g. check if nodes are still alive)

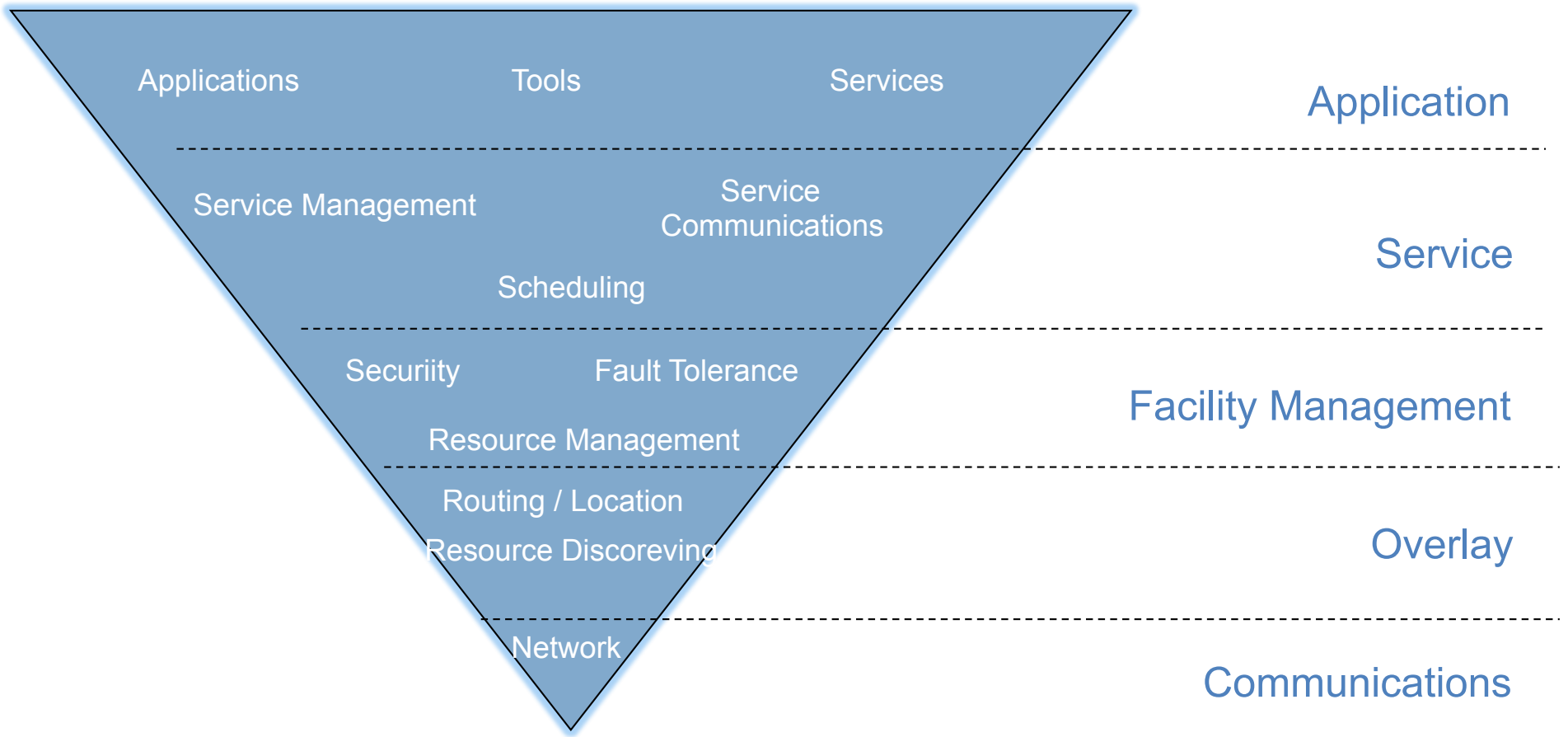
P2P-Overlay (cont'd)



P2P-Overlay (cont'd)



P2P-Overlay Abstration



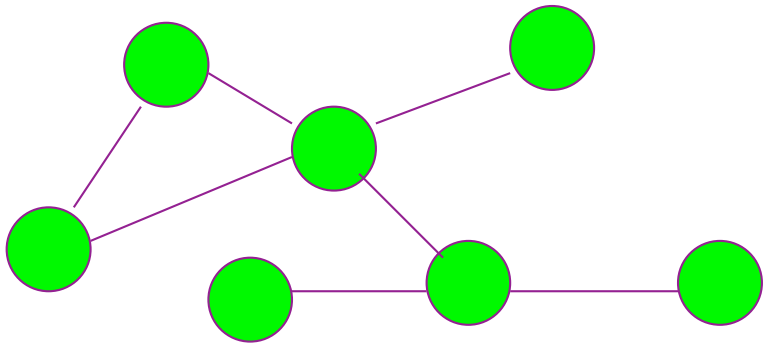
The P2P enabling technologies

- Unstructured P2P-overlays
  - Generally random overlay
  - Used for content download, telephony, streaming
- Structured P2P-overlays
  - Distributed Hash Tables (DHTs)
  - Used for node localization, content download, streaming

- History, motivation and evolution
  - History: Napster and beyond
  - What is Peer-to-peer?
  - Why Peer-to-peer?
- Brief P2P technologies overview
  - Unstructured P2P-overlays
  - Structured P2P-overlays

Unstructured P2P-overlays

- Unstructured P2P-overlays do not really care how the overlay is constructed
  - Peers are organized in a random graph topology
    - *e.g.*, new node randomly chooses three existing nodes as neighbors
    - Flat or hierarchical
  - Build your P2P-service based on this graph
- Several proposals
  - Gnutella
  - KaZaA/FastTrack
  - BitTorrent



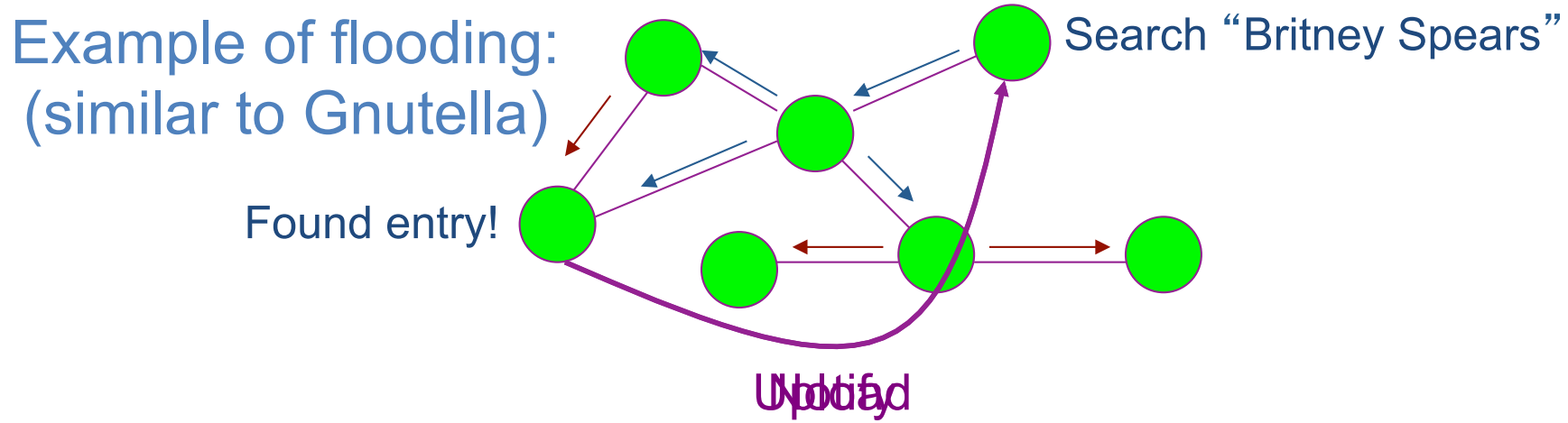
Unstructured P2P-overlays (cont'd)

- Unstructured P2P-overlays are just a framework, you can build many applications on top of it
- "Pure" P2P

- Unstructured P2P-overlays pros & cons
  - Pros
    - Very flexible: copes with dynamicity
    - Supports complex queries (conversely to structured overlays)
  - Cons
    - Content search is difficult: There is a tradeoff between generated traffic (overhead) and the horizon of the partial view

One Example of usage of unstructured overlays

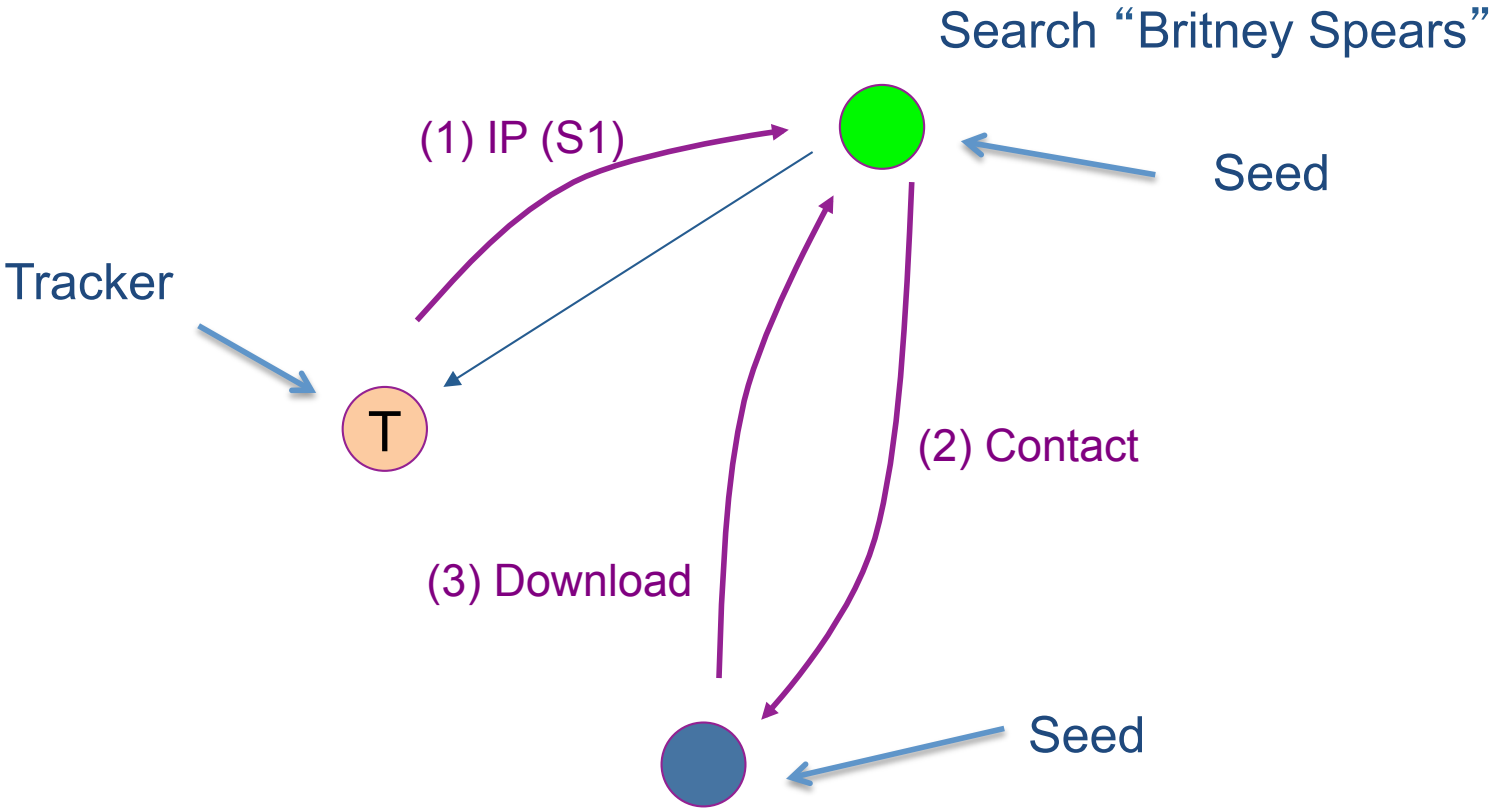
- Typical problem in unstructured overlays: How to do content search and query?
  - Flooding



  - Limited Scope, send only to a subset of your neighbors
  - Time-To-Live, limit the number of hops per messages

Another Example of usage of unstructured overlays

- Bittorent



- History, motivation and evolution
  - History: Napster and beyond
  - What is Peer-to-peer?
  - Why Peer-to-peer?
- Brief P2P technologies overview
  - Unstructured P2P-overlays
  - Structured P2P-overlays

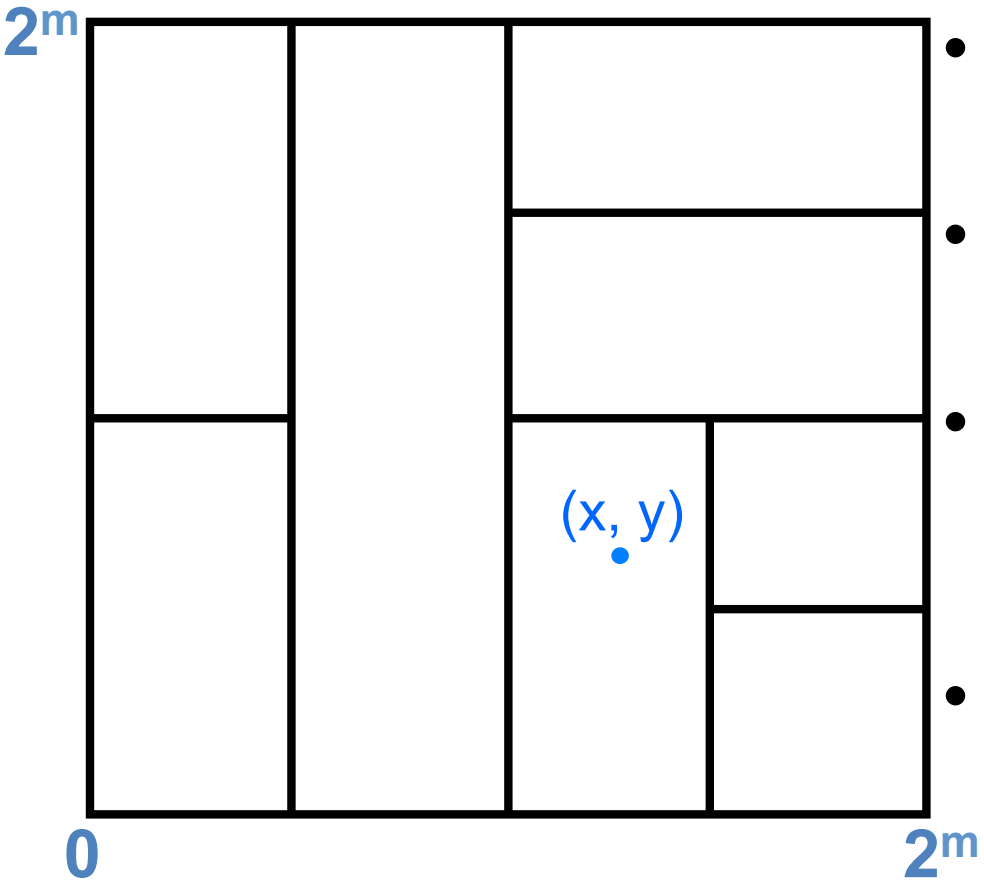
Structured P2P-overlays

- Motivation
  - Locate content efficiently
- Solution – DHT (Distributed Hash Table)
  - Particular nodes hold particular keys
  - Locate a key: *route search request to a particular node that holds the key*
  - Representative solutions
    - *CAN, Pastry/Tapestry, Chord, etc.*

Challenges to Structured P2P-overlays

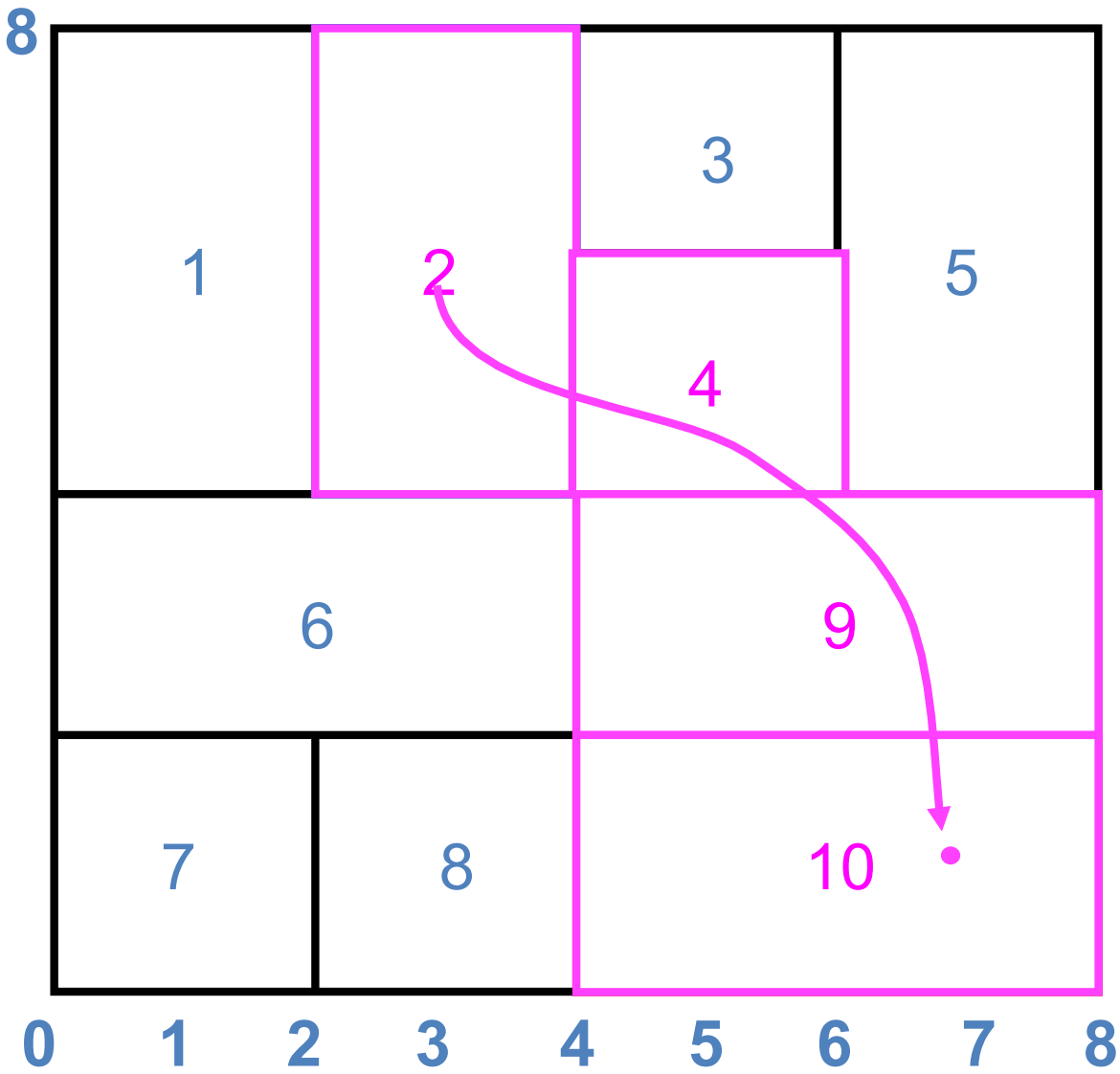
- Load balance
  - spreading keys evenly over the nodes.
- Decentralization
  - no node is more important than any other.
- Scalability
  - Lookup must be efficient even with large systems
- Peer dynamics
  - Nodes may come and go, may fail
  - Make sure that “the/a” node responsible for a key can always be found

Content-Addressable Network (CAN)



- Maps a 2-dimensional coordinate space
- Each node is responsible for a fraction of the space
- A hash pair is provided using two hash functions on the value to be stored
- Routing is done by forwarding to the neighbor that is closest in Cartesian distance

CAN example



Routing table for node 2

Node	Range
2	(2, 4) to (4, 8)
1	(0, 4) to (2, 8)
3	(4, 7) to (5, 8)
4	(4, 6) to (5, 7)
6	(0, 2) to (4, 4)

Entry hashed to (7, 1) is stored at node 10

Search path from 2 to (7, 1):

$2 \rightarrow 4 \rightarrow 9 \rightarrow 10$

CAN joins and departures

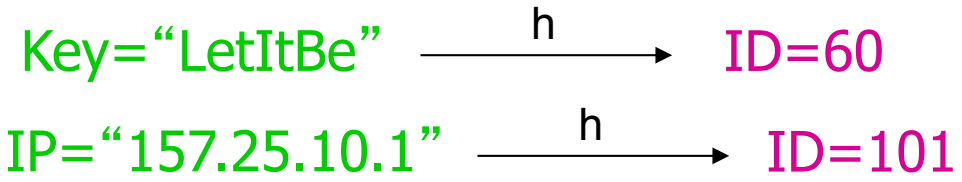
- Joining node picks a random pair (X, Y) and contacts node A
  - A sends a join message to B, the node responsible for the pair
  - B divides is its space with A and shares its routing information appropriately
  - A and B contact all its neighbors with updated info
- Departing node contacts its neighbors and finds one that can manage its space
  - The node sends updated routing info to its neighbors

CAN features

- Simple to understand and thus simple to add features to:
  - Overlapping regions for redundancy
  - D-dimensional space (not just 2)
  - Dynamic division of space for load balancing
- Hop #: $O(Dn^{1/D})$
- Routing Table: O(D)
- Possible overload

Tapestry/Pastry

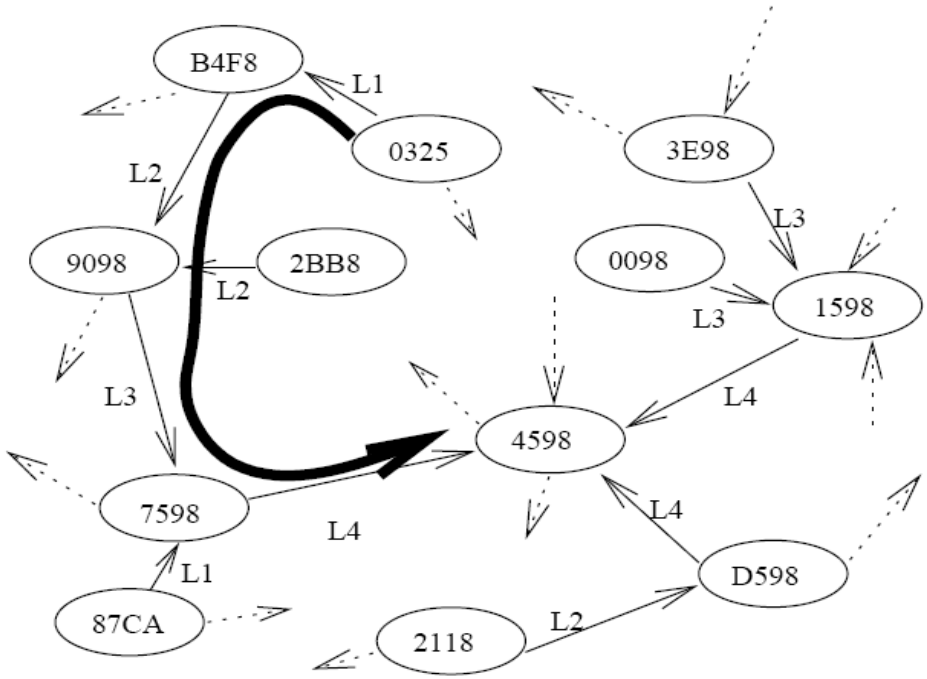
- Both based on **Plaxton Mesh**
  - Nodes – Act as routers, clients and servers simultaneously.
  - Objects – Stored on servers, searched by clients.
- Objects and nodes have **unique, location independent names**
  - Random fixed-length bit sequence in some base
  - Typically, hash of hostname or of object name
  - A hashing function *h* assigns each node and key a *m*-bit identifier



- What Plaxton does:
  - Given **message M** and **destination D**, a Plaxton mesh **routes M** to the node whose name is **numerically closest to D**

Plaxton Mesh Routing

- Routing done incrementally, digit by digit
  - In each step, message sent to node with address one digit closer to destination.
- Example - node 0325 sends M to node 4598
  - Possible route:
    - 0325
    - B4F**8**
    - 90**98**
    - 7**598**
    - **4598**



Plaxton Mesh
Routing Tables

- Each node has a neighbor map
  - Used to find a node whose address has one more digit in common with the destination
- Size of the map: $b * \log_b N$, where b is the base used for the address

Plaxton Mesh
Routing Tables

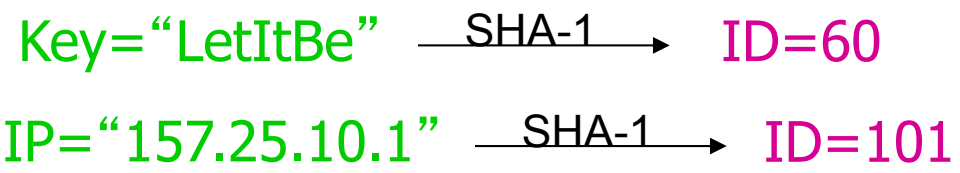
- x are wildcards. Can be filled with any address.
  - Each slot can have several addresses. Redundancy.
- Example: node 3642 receives message for node 2342
  - Common prefix: XX**42**
    - Look into second column.
  - Must send M to a node one digit "closer" to the destination.
    - Any host with an address like X342.

Node 3642

0642	x042	xx02	xxx0
1642	x142	xx12	xxx1
2642	x242	xx22	xxx2
3642	x342	xx32	xxx3
4642	x442	xx42	xxx4
5642	x542	xx52	xxx5
6642	x642	xx62	xxx6
7642	x742	xx72	xxx7

Chord Protocol

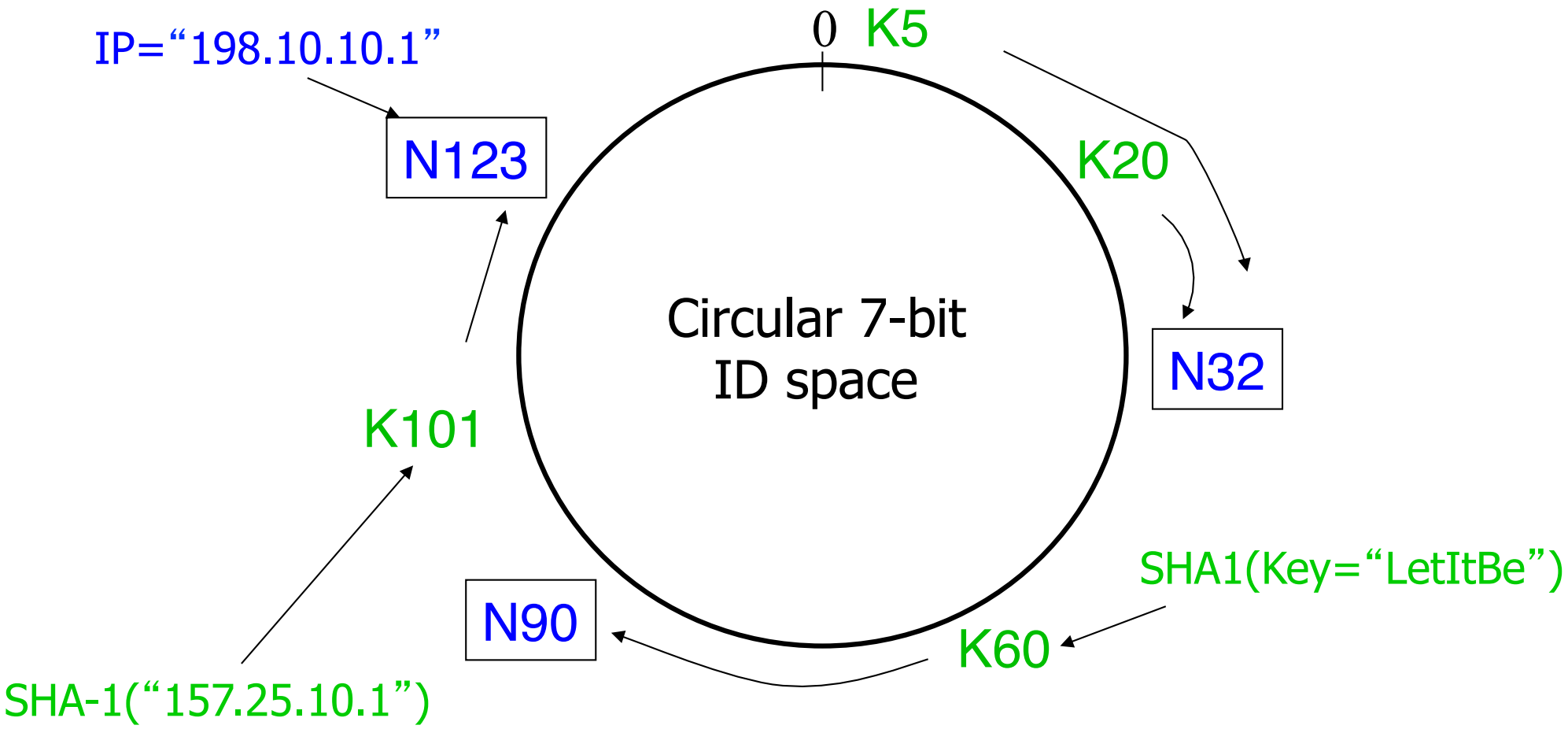
- A consistent hashing function (SHA-1) assigns each node and key a *m*-bit identifier



- Identifiers are ordered on a identifier circle modulo 2^m called a chord ring.
- *successor(k)* = first node whose identifier is >= identifier of k in identifier space.

Consistent Hashing (cont.)

- **A key is stored at its successor: node with next higher ID**

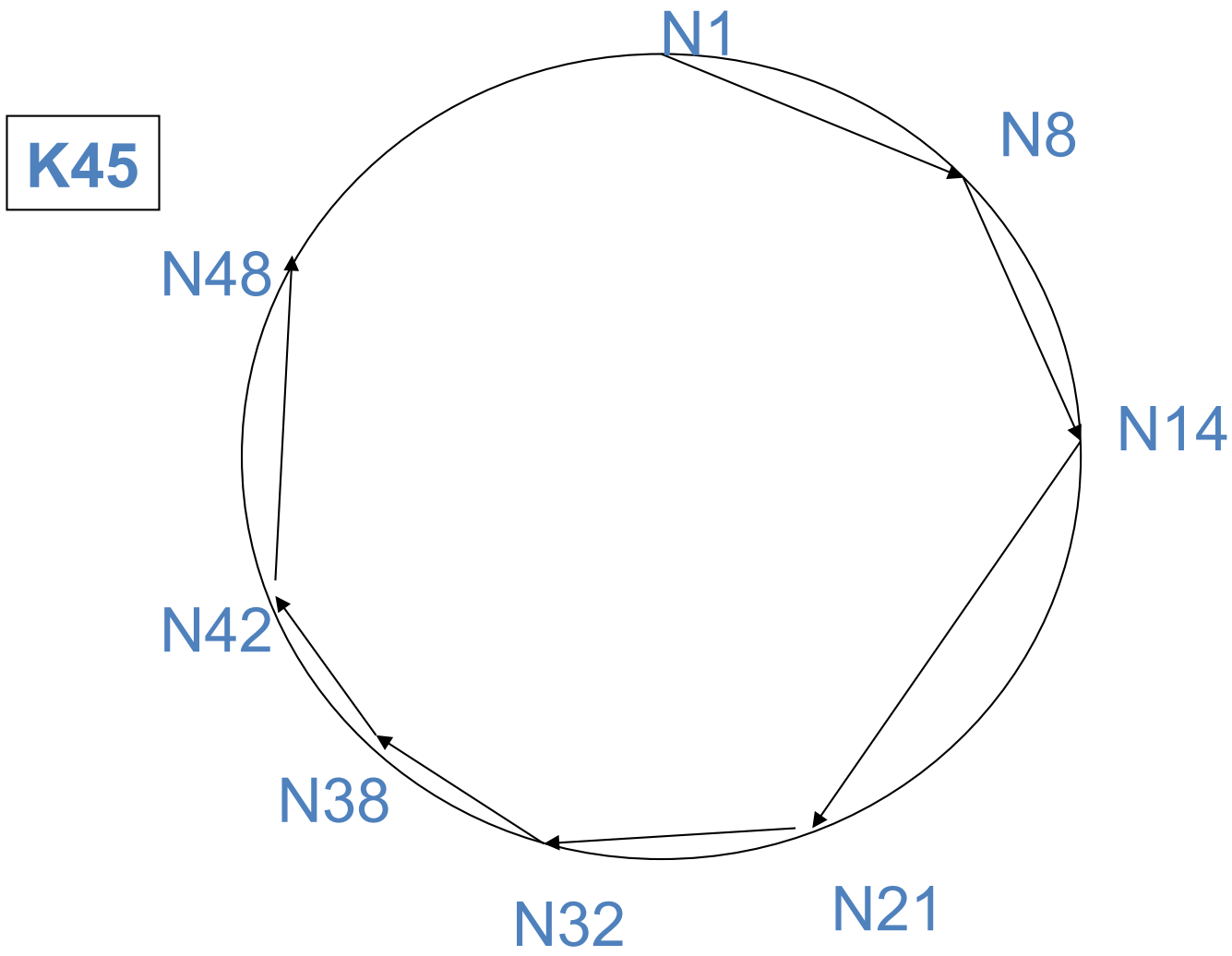


Theorem

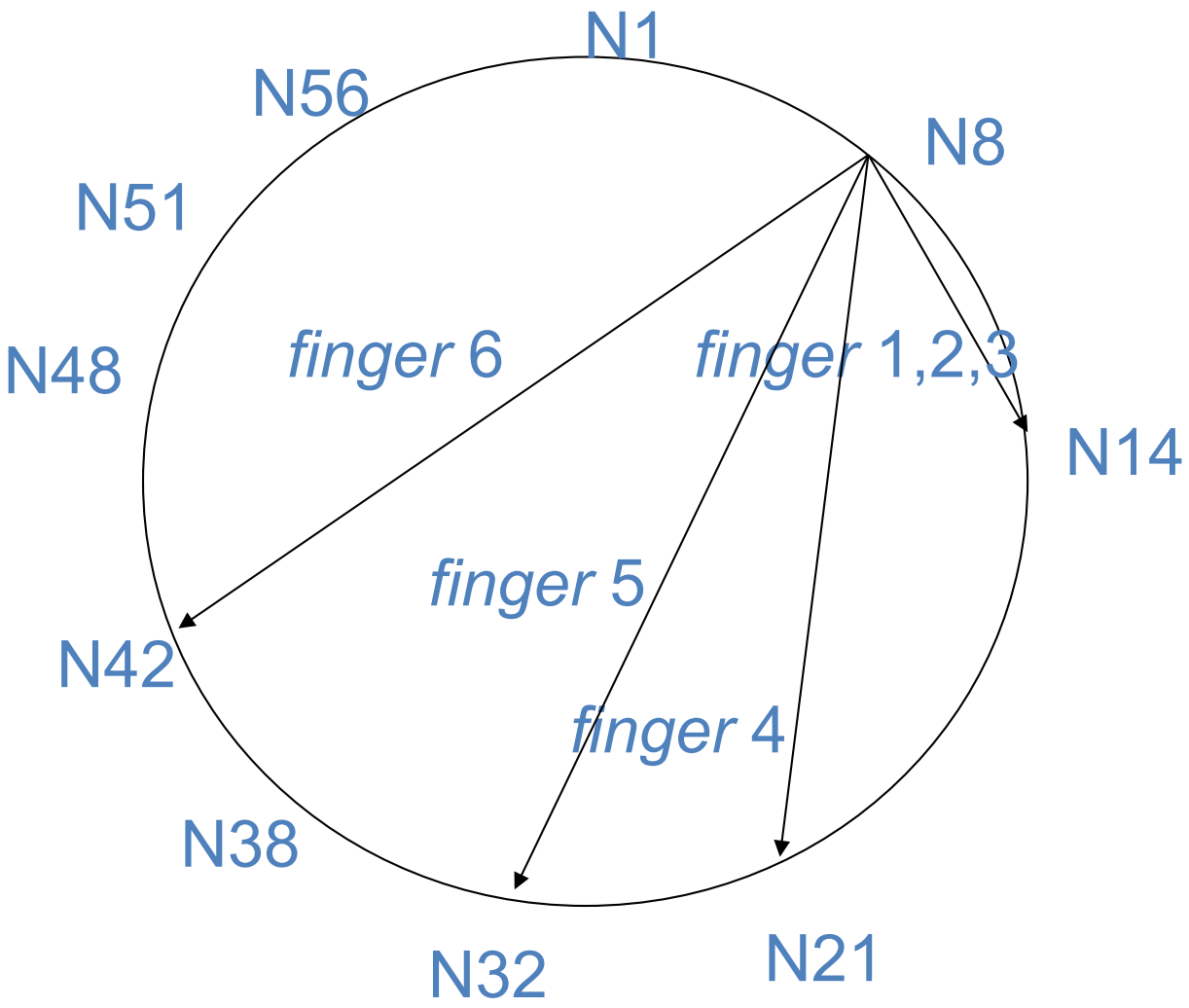
- For any set of N nodes and K keys, with *high probability*:
  1. Each node is responsible for at most (1+e)K/N keys.
  2. When an $(N+1)^{st}$ node joins or leaves the network, responsibility for O(K/N) keys changes hands.

  e = O(log N)

Simple Key Location Scheme



Scalable Lookup Scheme

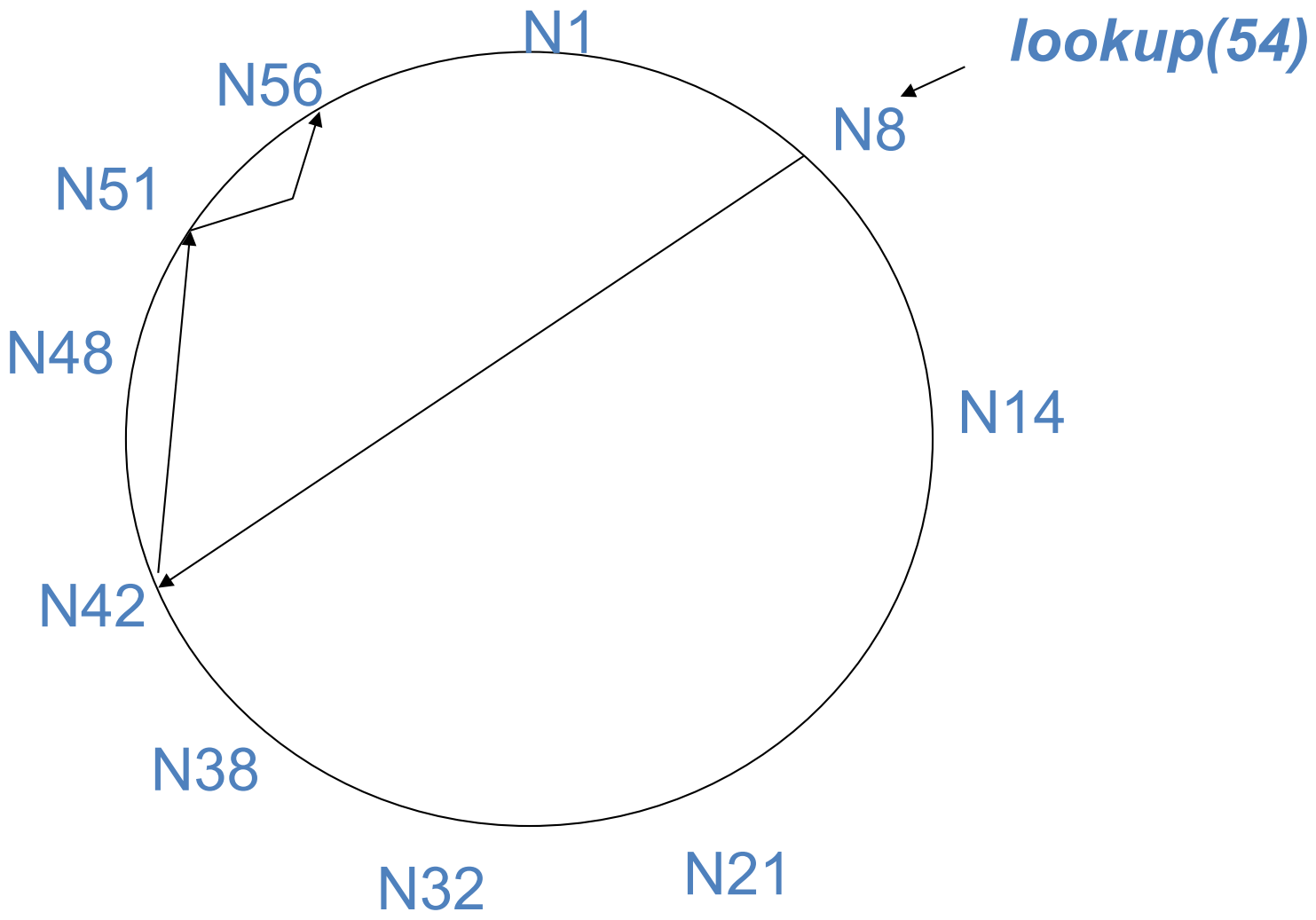


Finger Table for N8

N8+1	N14
N8+2	N14
N8+4	N14
N8+8	N21
N8+16	N32
N8+32	N42

finger [k] = first node that succeeds
$(n+2^{k-1}) \bmod 2^m$

Lookup Using Finger Table



Scalable Lookup Scheme

- Each node forwards query at least halfway along distance remaining to the target
- Theorem: With high probability, the number of nodes that must be contacted to find a successor in a N-node network is O(log N)

Chord joins and departures

- Joining node gets an ID N and contacts node A
  - A searches for N's predecessor B
  - N becomes B's successor
  - N contacts B and gets its successor pointer and finger table, and updates the finger table values
  - N contacts its successor and inherits keys
- Departing node contacts its predecessor, notifies it of its departure, and sends it the new successor pointer

Chord features

- Correct in the face of routing failures since can correctly route with successor pointers
- Replication can easily reduce the possibility of loosing data
- Caching can easily place data along a likely search path for future queries
- Simple to understand