P2P Systems

Vania Marangozova-Martin

ids.forge.imag.fr

Peer to Peer in a Nutshell

- Distributed systems
  - response to the rapid growth of Internet
  - apparition of various/multiple hosts
  - that form a global set of computational, storage and data resources

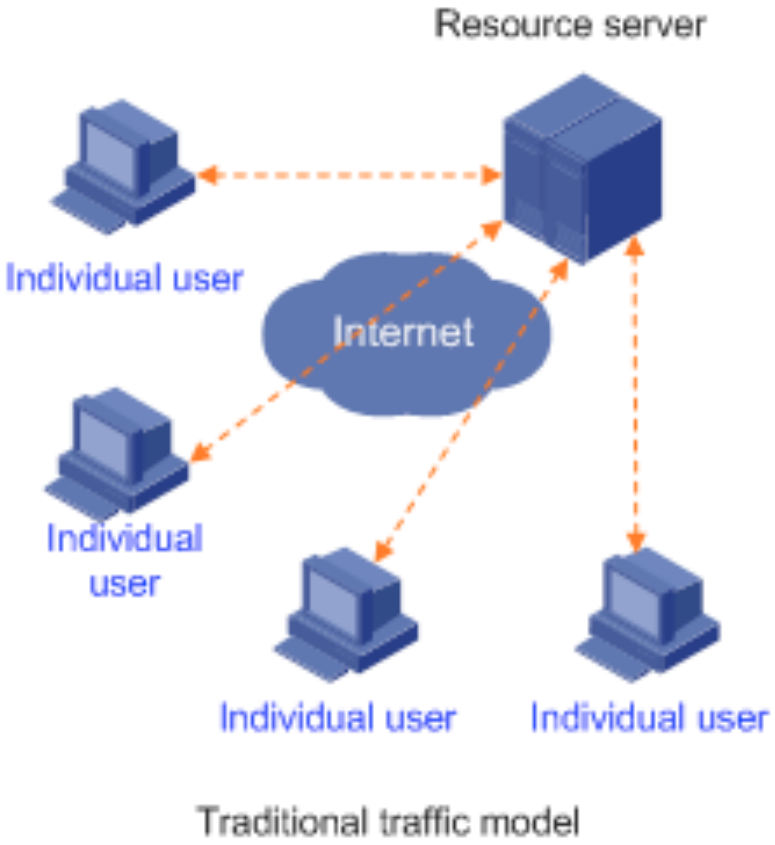
- Goal: provision of a uniform service
  - without managing servers and the corresponding infrastructure

General Motivations

- [Shirky 2000]
- Peer-to- peer applications are
  „applications that exploit resources available at the edges of the Internet – storage, cycles, content, human presence“

From C-S to P2P

- Connect the edges: connect users



Traditional traffic model

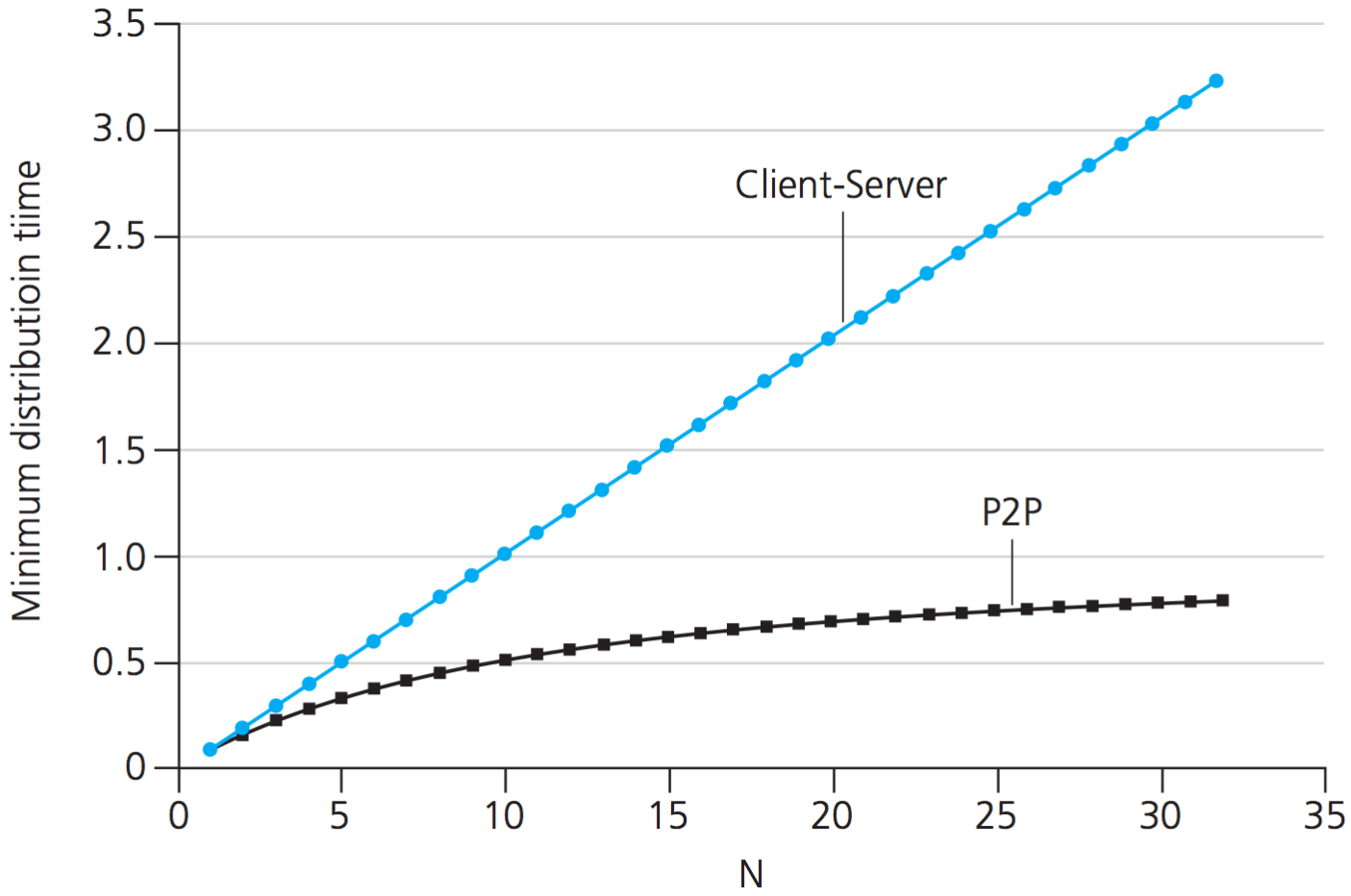
C-S Architecture : Discussion

- C-S paradigm gives access to a service
  - asymmetrical design
    - the server has a privileged function
  - centralized design
    - the server is in charge of the service
    - actually, if there are multiple servers (replicas), they would be peers…
  - few decisions are required about the placement of the resources or the management of server hardware resources
  - the scale of the service is limited by the server hardware capacity and network connectivity

Peer-to-peer Systems Characteristics

- Each user contributes resources to the system.
- All nodes have the same functional capabilities and responsibilities.
- Their correct operation does not depend on the existence of any centrally administered systems.
- They may offer limited degree of anonymity to the providers and users of resources.
- Use an algorithm for efficient placement of data across many hosts
  - the subsequent access balances the workload and ensures availability

C-S versus P2P Scalability



P2P and Resource Availability

- End users availability cannot be guaranteed
  - computers, devices, different organizations, etc. cannot be guaranteed to be connected or switched on forever
- Peer-to-peer services therefore cannot rely on guaranteed access to individual resources
  - need to manage the probability of « failure »

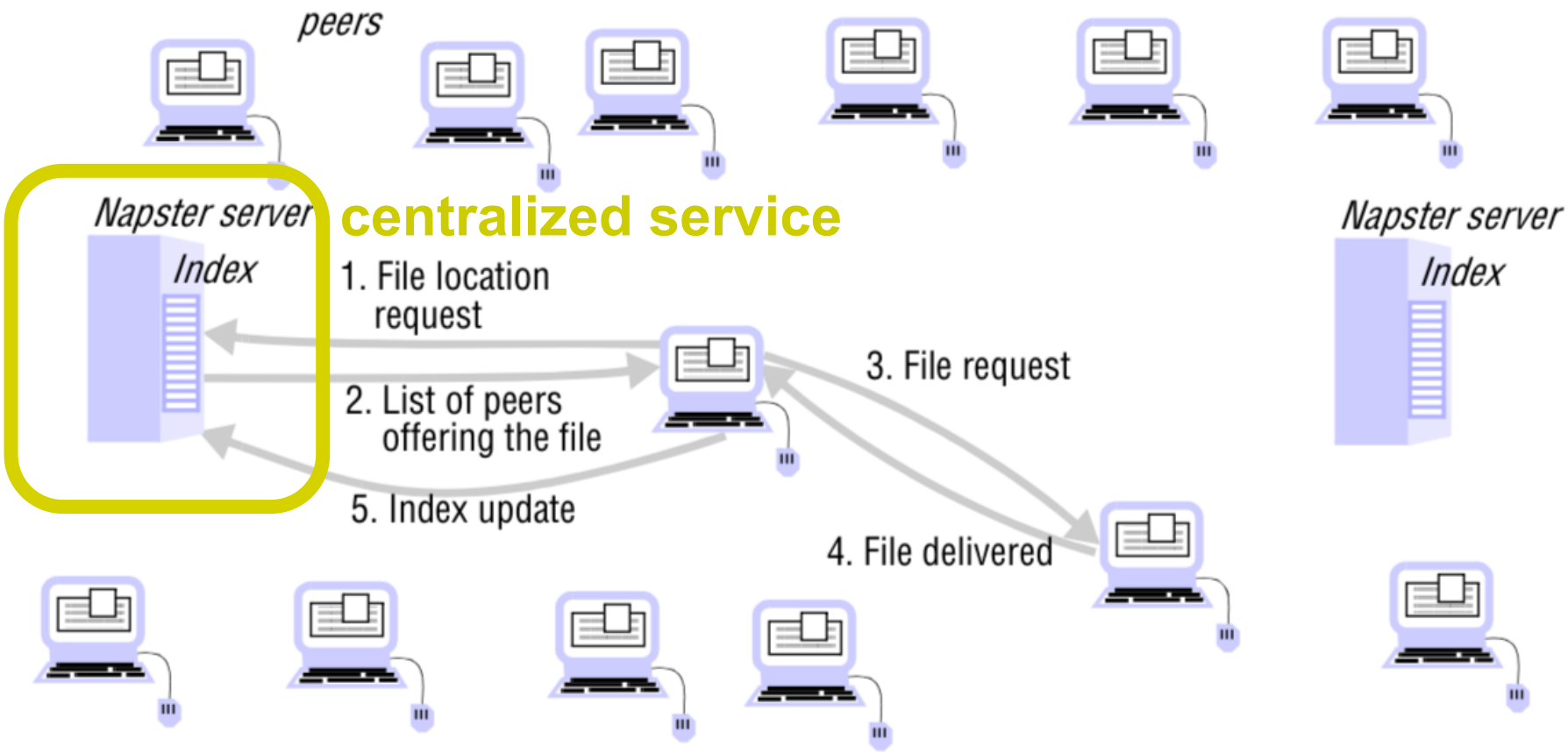
Three Generations of Peer-to-peer Systems

- 1) Napster 2001
- 2) File-sharing with greater scalability, anonymity and fault tolerance
  - Freenet, Gnutella, BitTorrent
- 3) Emergence of middleware layers for the application-independent management of distributed resources on a global scale
  - Pastry, CAN, Chord, …

1st Generation: Napster (2001)

- Download of digital music files
  - globally scalable information storage and retrieval service
  - Start 1999
  - At its peak, millions of users, exchanging music
  - Shutdown for copyright problems
- Users provide the files
- Napster provides the indexes (where to find the files)

Napster Function



The Question of Anonymity

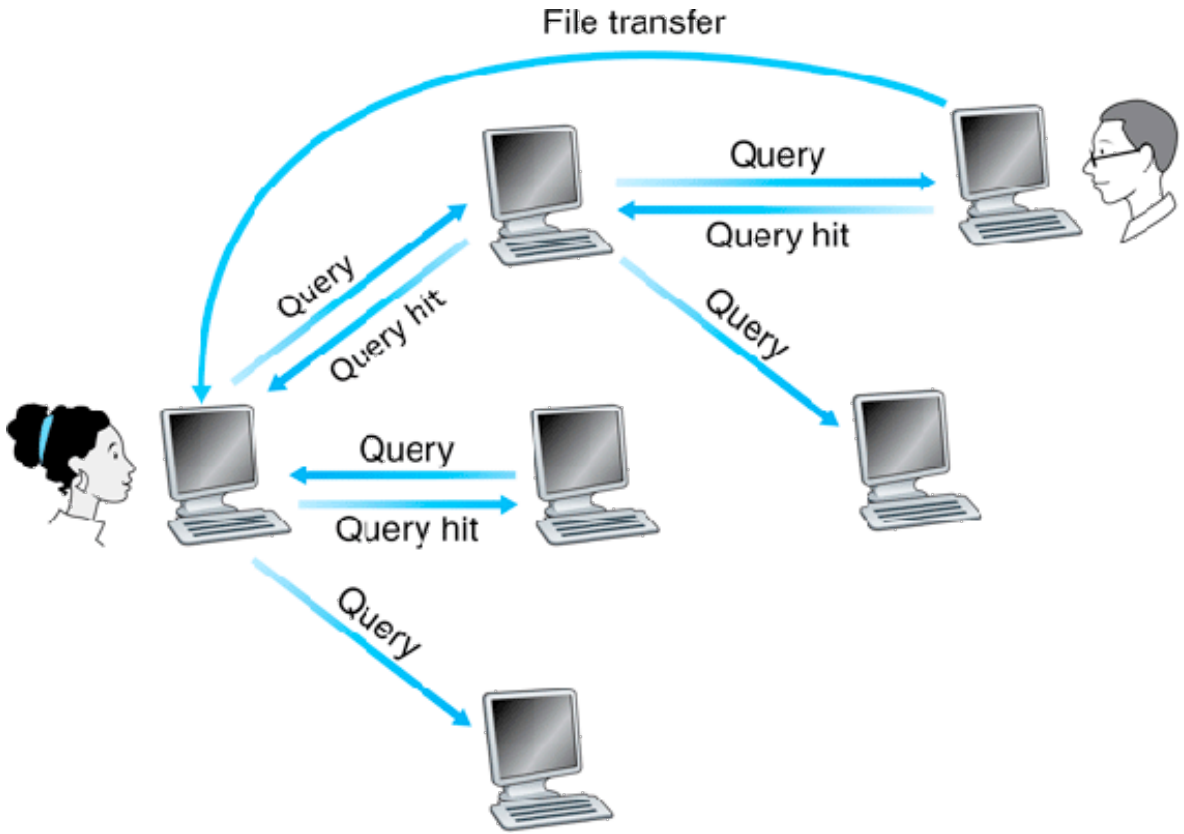
- How to provide anonymity for the receivers and the providers of shared data and other resources?
  - use complicated routing
  - the contents of a file may be spread over multiple nodes, spreading the responsibility for making them available
  - anonymous communication resisting analysis
    - augments the cost of sharing

Lessons learned from Napster

- Feasability of a useful large-scale service that depends almost wholly on data and computers owned by ordinary Internet users
- To avoid swamping the computing resources of individual users, Napster took account of network locality
  - the number of hops between the client and the server
- It worked well because of its specifics
  - audio files do not change (no need for updates)
  - no guarantees required for file availability

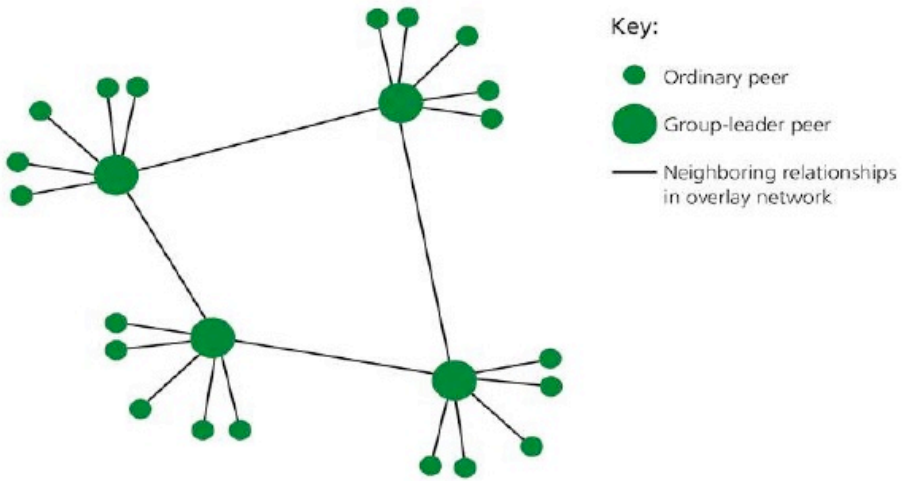
2nd Generation: Gnutella (2004)

- Fully decentralized approach to find files
- Query flooding
- Expensive in terms of number of messages
  - limit requests to the neighbourhood



KaZaA (2004)

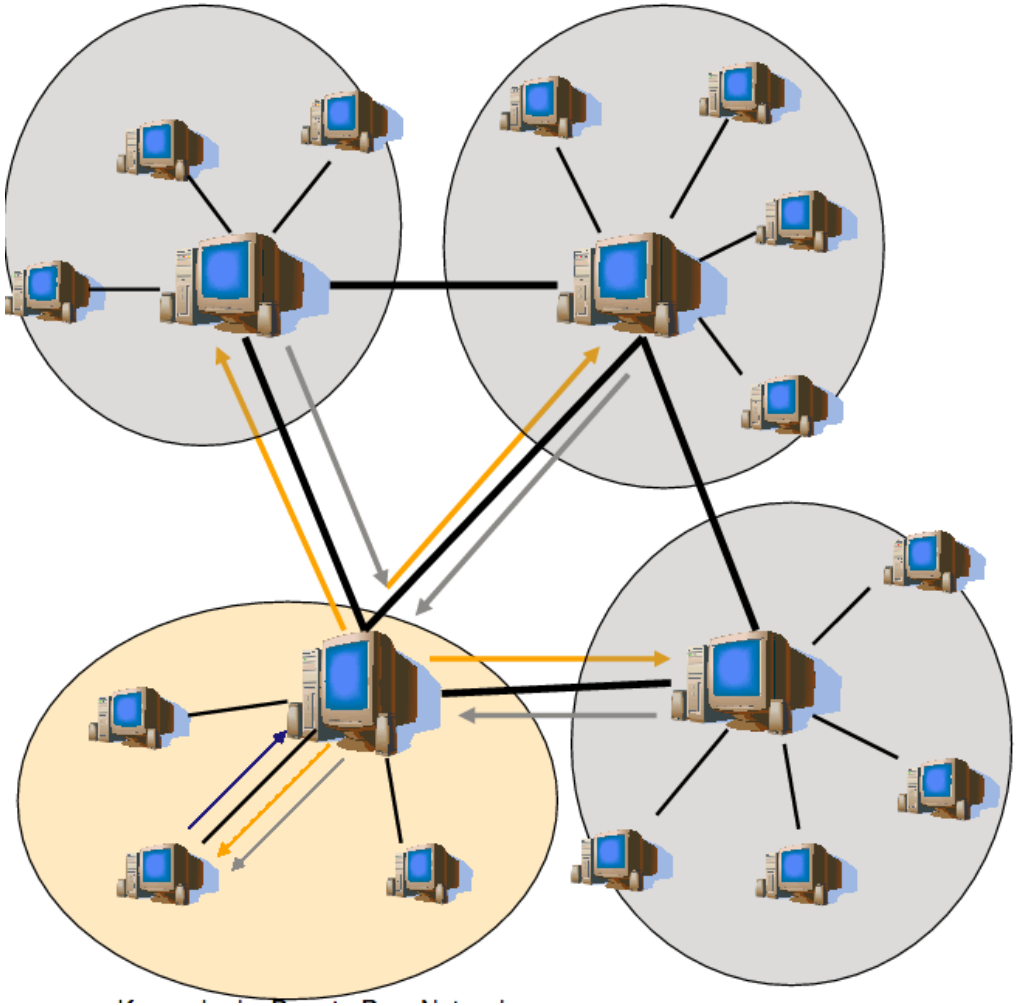
Hierarchical overlay network for P2P file sharing



- Ideas from both Napster & Gnutelle
- KaZaA based on a supernode-architecture
  - mini Napsters
- Many important lessons from KaZaA
  - Exploit heterogeneity of peers
  - Organize peers into a hierarchy

7/15/2010 15

KaZaA: Finding Stuff



1. Peer sends query to its own supernode
2. Supernode answers for all of its peers and forwards query to other supernodes
3. Other supernodes reply for all of their peers

KaZaA: Spyware

- KaZaA included spyware in their program
- Spyware does things like:
  - Send all DNS queries to a tracking server
  - Monitor visited websites
  - Additional popup windows on “partner” sites
- KaZaA originally denied existence of spyware

Napster vs. Gnutella vs. KaZaA

	Napster	Gnutella	KaZaA
Type of Network	Centralized	Distributed	Hybrid
Efficient Searching	+++	---	+
Resilience to Attacks	---	++?	+
Open Protocol	N	Y	N
Spyware-free	Y	Y	Y/N?
Popularity	+++	-	+++

For Your Long Term Memory

- Many different kinds of file sharing networks
- Three main architectural solutions for indexing
  - Centralized index
  - Distributed index
  - Hybrid index
- File sharing networks also called unstructured
  - Content can be placed anywhere in the network
- Contrast: Structured networks
  - Every file has a well-defined place
  - DHTs

BitTorrent (2011)

- BitTorrent is a new approach for sharing large files
- BitTorrent used widely also for legal content
  - For example, Linux distributions, software patches
  - Official movie distributions are also happening (WB)
- Goal of BitTorrent
  - Quickly replicate one file to a large number of clients
- BitTorrent more appropriately called peer-to-peer content distribution

BitTorrent (2011)

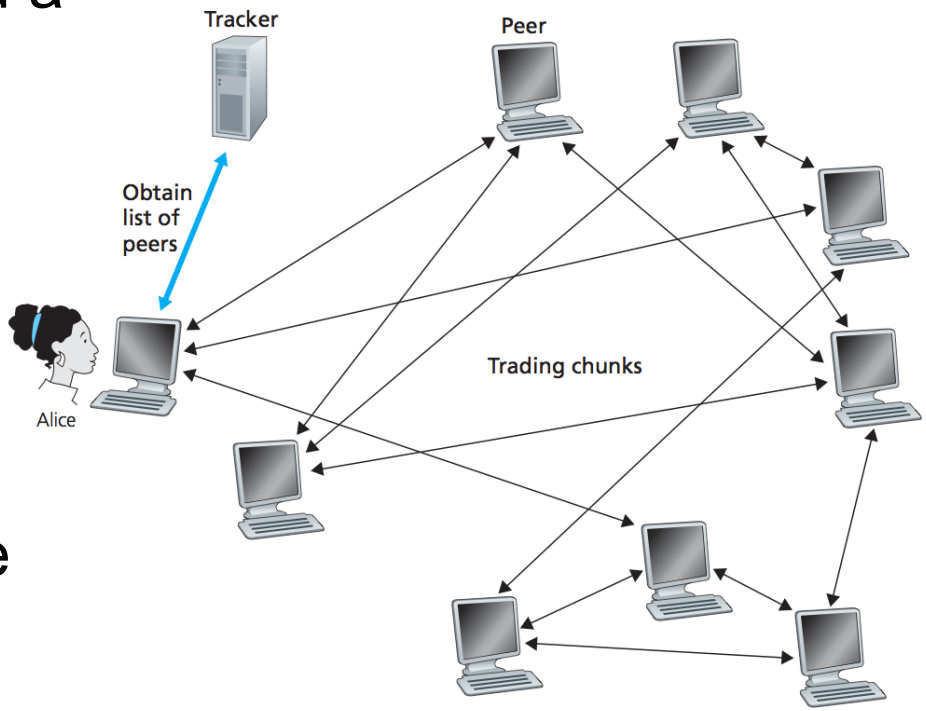
- BitTorrent builds a network for every file that is being distributed
  - A torrent = the collection of all peers participating in the distribution of a particular file
- Can send “link”to a friend
- “Link” always refers to the same file
- Downside of BitTorrent: No searching possible
  - Websites with “link collections” and search capabilities exist

BitTorrent (cont.)

- Peers in a torrent download equal-size chunks of the file from one another
- While a peer downloads chunks it also uploads chunks to other peers
- Once a peer has acquired the entire file, it may (selfishly) leave the torrent, or (altruistically) remain in the torrent

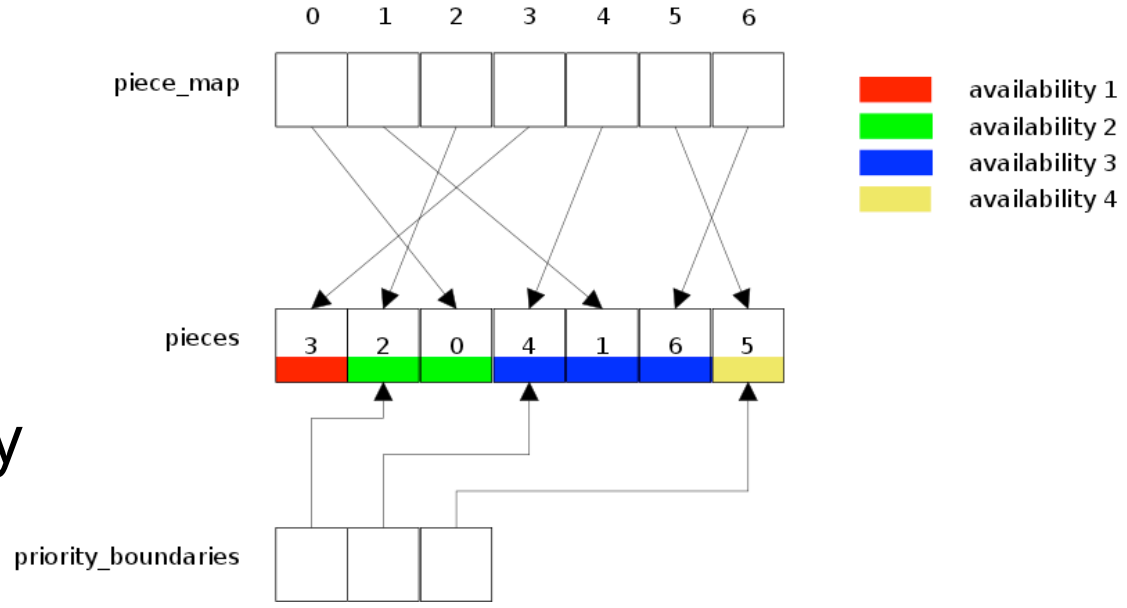
BitTorrent

- Each torrent has an infrastructure node called a tracker
- When a peer joins a torrent, it registers itself with the tracker and periodically informs the tracker that it is still in the torrent



Which request to handle first?

- Rarest first
  - peers know the chunks of their peers
  - ask first for the rarest
- Trading
  - give priority to the neighbors that are currently supplying at the highest rate



BitTorrent Evaluation

- Works quite well
  - Download a bit slow in the beginning, but speeds up considerably as peer gets more and more chunks
  - Efficient mechanism for distributing large files to a large number of clients
- Weaknesses
  - File needs to be quite large
    - 256 KB chunks
    - Rarest first needs large number of chunks
- Everyone must contribute
  - Low-bandwidth clients have a disadvantage?

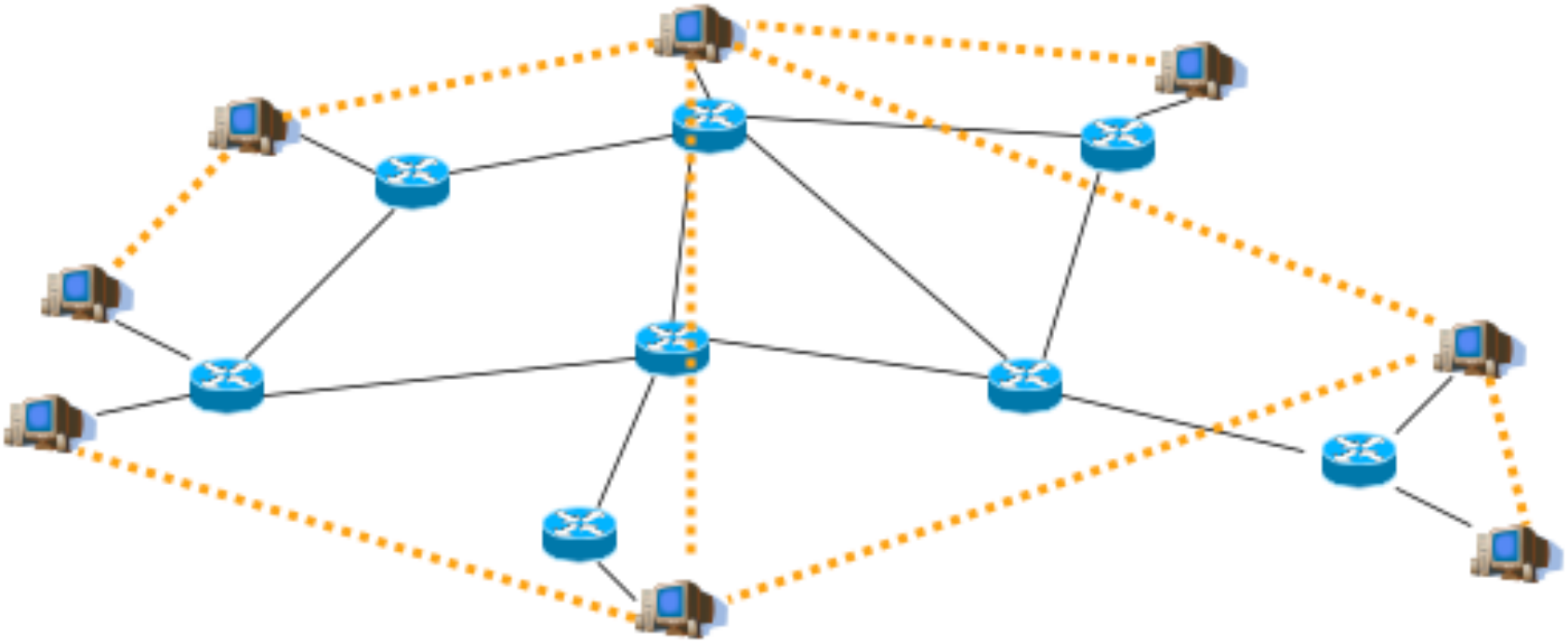
3d Generation: Distributed HashTables (DHT)

- Key issues (Functional)
  - provide a mechanism to enable clients to access data resources quickly and dependably wherever they are located throughout the network
  - ability to add new resources and to remove them at will
  - to add hosts to the service and remove them
- NFR
  - Global scalability: exploit the hardware resources of very large numbers of hosts connected to the Internet
  - Load balancing: balanced distribution of workload across resources
  - Optimization for local interactions between neighbouring peers

Routing Overlay

- A distributed algorithm for locating nodes and objects
  - Middleware that is responsible for routing requests form a client to the host that holds an object
  - the objects may be placed, relocated, replicated without the involvment of the client
  - overlay means that the routing mechanism is quite different from any other physical routing mechanisms such as IP routing

Overlay Example



Distinctions between IP and overlay routing for peer-to-peer applications

Criterium	IP	Routing overlay
Scale	IPv4 is limited to 232 addressable nodes. The IPv6 namespace is much more generous (2128), but addresses in both versions are hierarchically structured and much of the space is preallocated according to administrative requirements.	Peer-to-peer systems can address more objects. The GUID namespace is very large and flat (>2128), allowing it to be much more fully occupied.
Load balancing	Loads on routers are determined by network topology and associated traffic patterns.	Object locations can be randomized and hence traffic patterns are divorced from the network topology.

Distinctions between IP and overlay routing for peer-to-peer applications

Criterium	IP	Routing overlay
Network dynamics (addition/ deletion of objects/ nodes)	IP routing tables are updated asynchronously on a best-effort basis with time constants on the order of 1 hour.	Routing tables can be updated synchronously or asynchronously with fractions-of-a-second delays.
Fault tolerance	Redundancy is designed into the IP network by its managers, ensuring tolerance of a single router or network connectivity failure. n-fold replication is costly.	Routes and object references can be replicated n-fold, ensuring tolerance of n failures of nodes or connections.

Distinctions between IP and overlay routing for peer-to-peer applications

Criterium	IP	Routing overlay
Target identification	Each IP address maps to exactly one target node.	Messages can be routed to the nearest replica of a target object.
Security and anonymity	Addressing is only secure when all nodes are trusted. Anonymity for the owners of addresses is not achievable.	Security can be achieved even in environments with limited trust. A limited degree of anonymity can be provided.

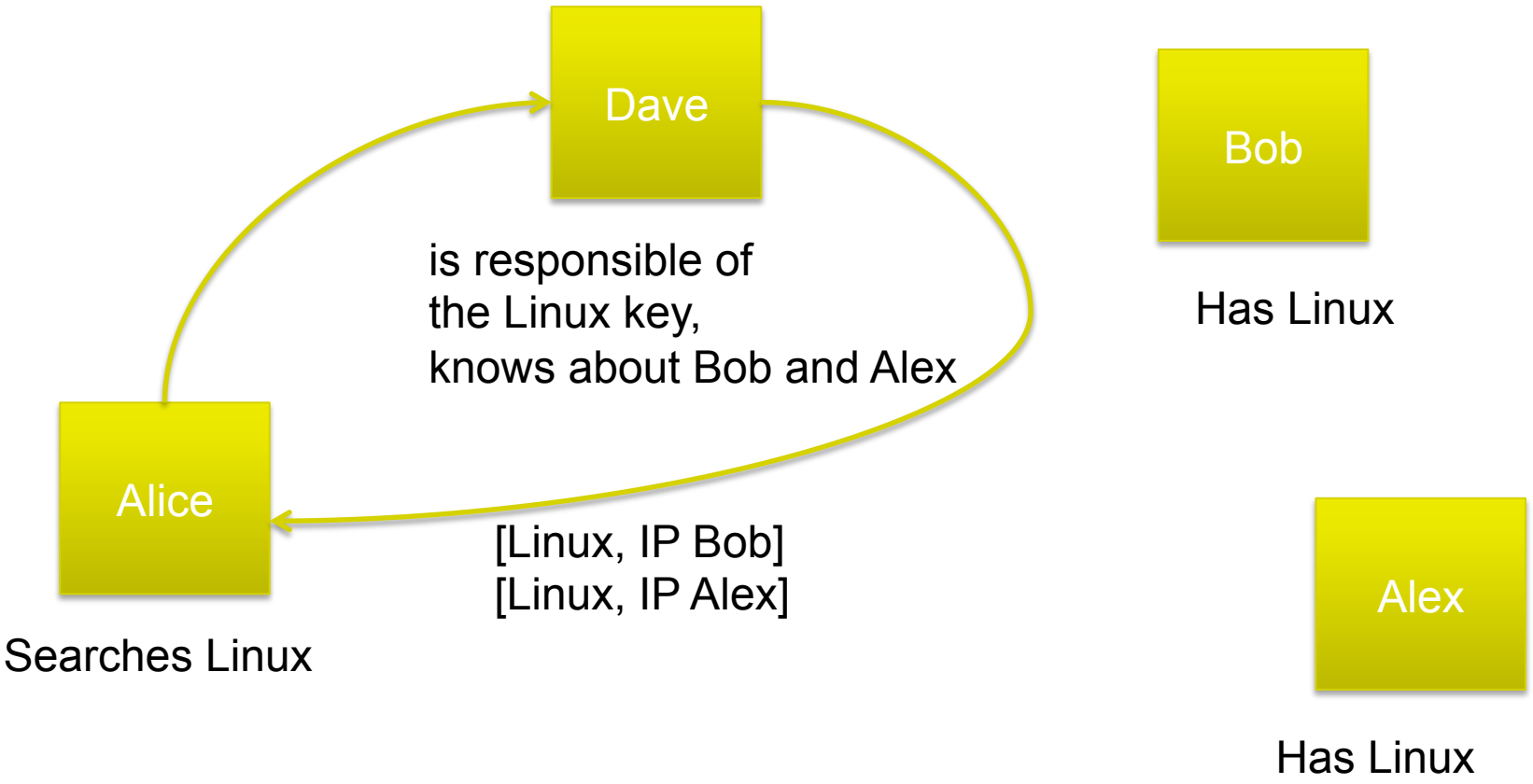
Managing objects in DHTs

- Functionality
  - Insert objects (store them, usually with replication)
  - Locate objects
  - Route requests to objects
- Use of GUID (Global Unique Identifiers)
  - Opaque i.e do not reveal the location
- Interface
  - `put(GUID, data)`
  - `remove (GUID)`
  - `value = get (GUID)`

GUID and Node Ids

- Computation of GUID
  - An object's GUID is computed from all or part of the state of the object using a function that delivers a value that is, with very high probability, unique – hash function
- Where do we put the object?
  - if we scatter randomly?
    - all peers must maintain the list of all other peers
    - does not use the network efficiently
    - does not scale
  - Assign each (key, value) pair to the peer whose identifier is the closest to the key

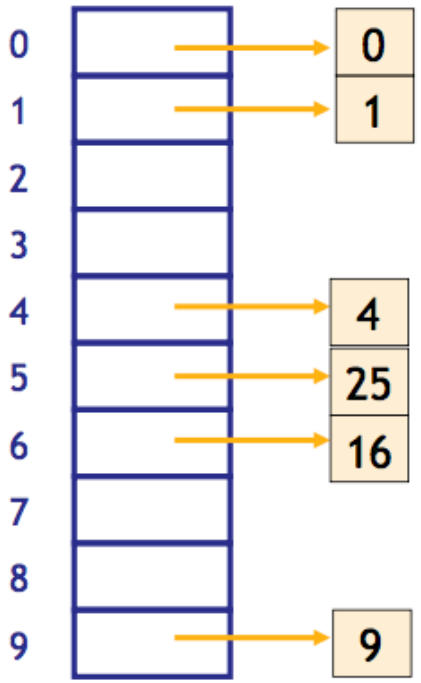
A comprehensive application example



Hash Tables Properties

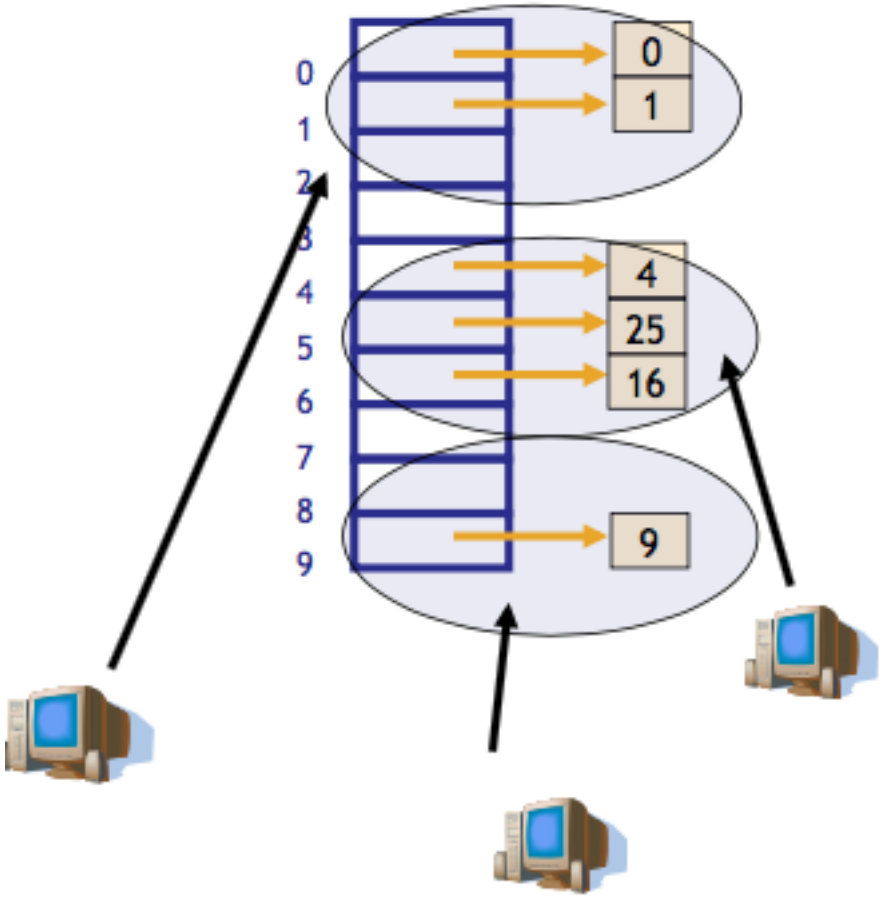
- Hash tables are a well-known data structure
- Hash tables allow insertions, deletions, and finds in constant (average) time
- HashTable is a fixed-size array
  - Elements of array also called hash buckets
  - Hash function maps keys to elements in the array
- Properties of good hash functions:
  -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

HashTable Example



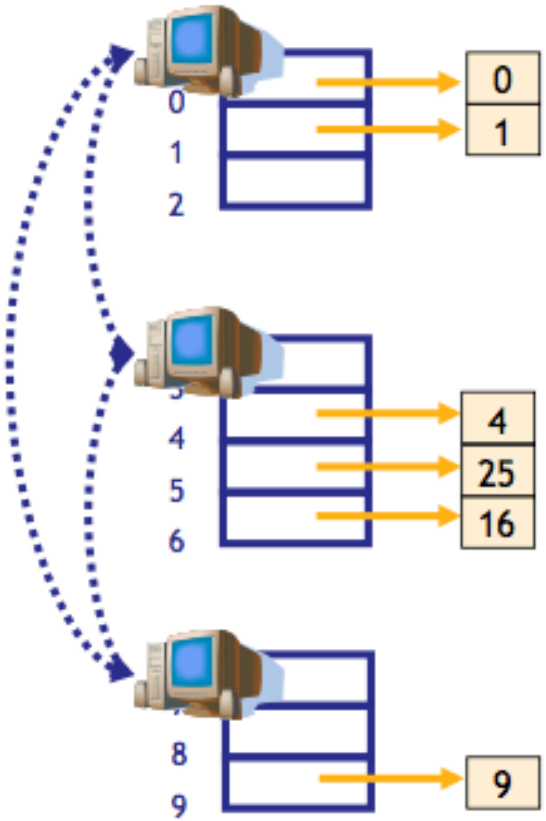
- Hash function: hash(x) = x mod 10
- Insert numbers 0, 1, 4, 9, 16, and 25
- Easy to find if a given key is present in the table

DHT: Idea



DHT: Principle

- In a DHT, each node is responsible for one or more hash buckets
  - As nodes join and leave, the responsibilities change
- Nodes communicate among themselves to find the responsible node
  - Scalable communications make DHTs efficient
- DHTs support all the normal hash table operations



Chord (2001)

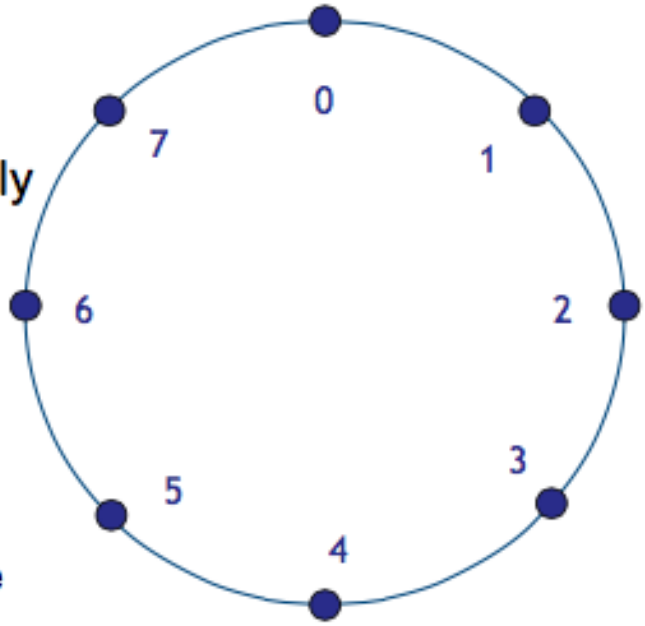
- Stoica *et al*., "Chord: a scalable peer-to-peer lookup protocol for Internet applications," in *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17-32, Feb 2003.
- Developed in MIT
- Paper has mathematical proofs of correctness and performance
- Projects around Chord
  - CFS stoarge system
  - Ivy storage system
  - ...

Chord Basics

- Chord uses SHA-1 hash function
  - Results in a 160-bit object/node identifier
  - Same hash function for objects and nodes
- Node ID hashed from IP address
- Object ID hashed from object name
  - Object names somehow assumed to be known by everyone
- SHA-1 gives a 160-bit identifier space
- Organized in a ring which wraps around
  - Nodes keep track of predecessor and successor
  - Node responsible for objects between its predecessor and itself
  - Overlay is often called “Chord ring” or “Chord circle”

Joining: Step-By-Step Example

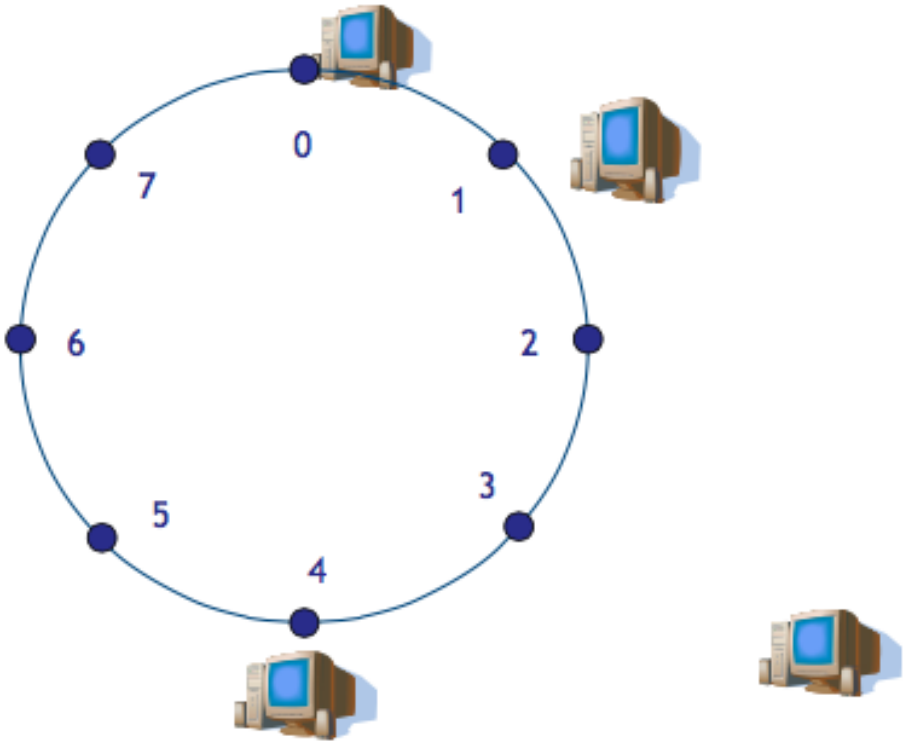
- Setup: Existing network with nodes on 0, 1 and 4
- Note: Protocol messages simply examples
- Many different ways to implement Chord
  - Here only conceptual example
  - Covers all important aspects



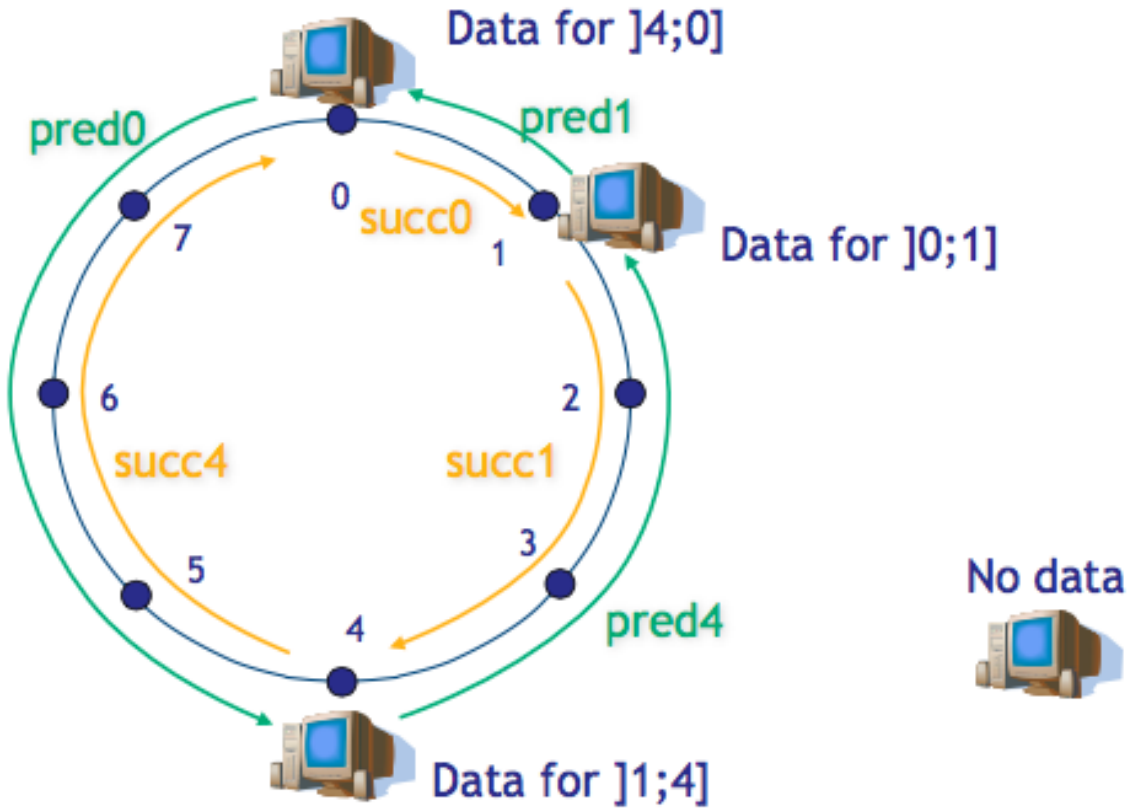
Joining: Step-By-Step Example: Start

- New node wants to join
- Hash of the new node: 6
- Known node in network: Node1

- Contact Node1
  - Include own hash

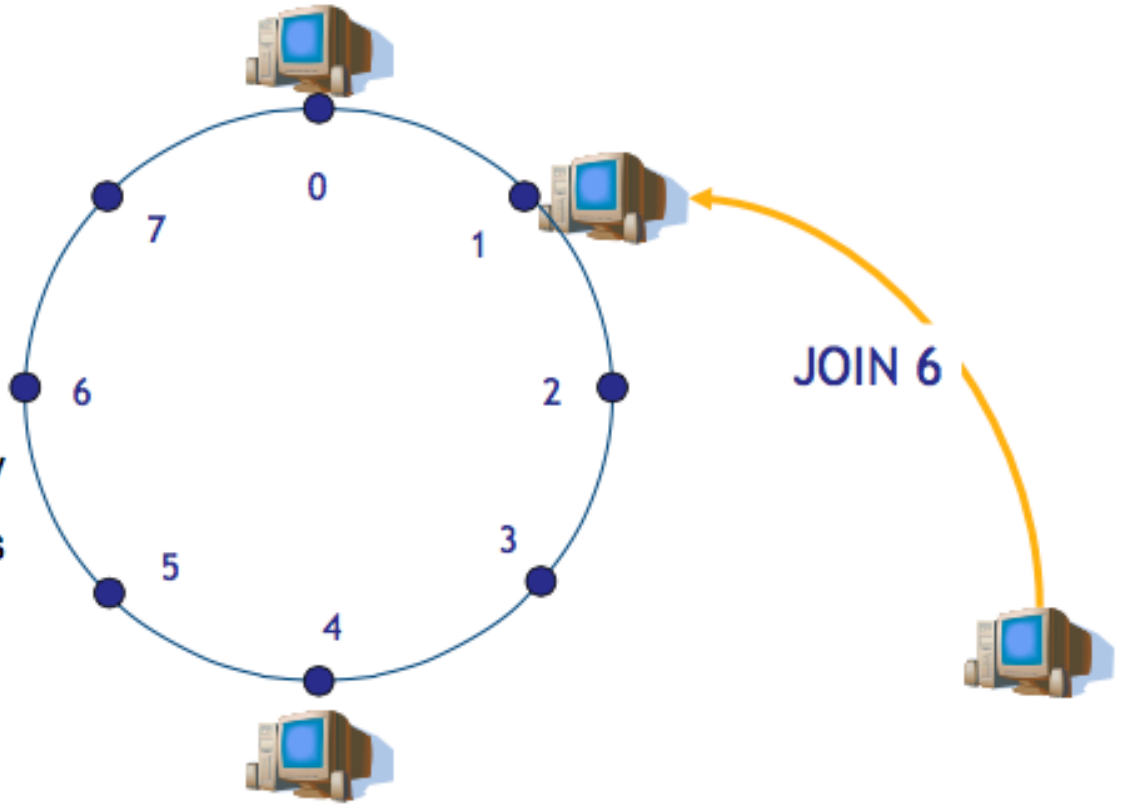


Joining: Step-By-Step Example: Situation Before Join

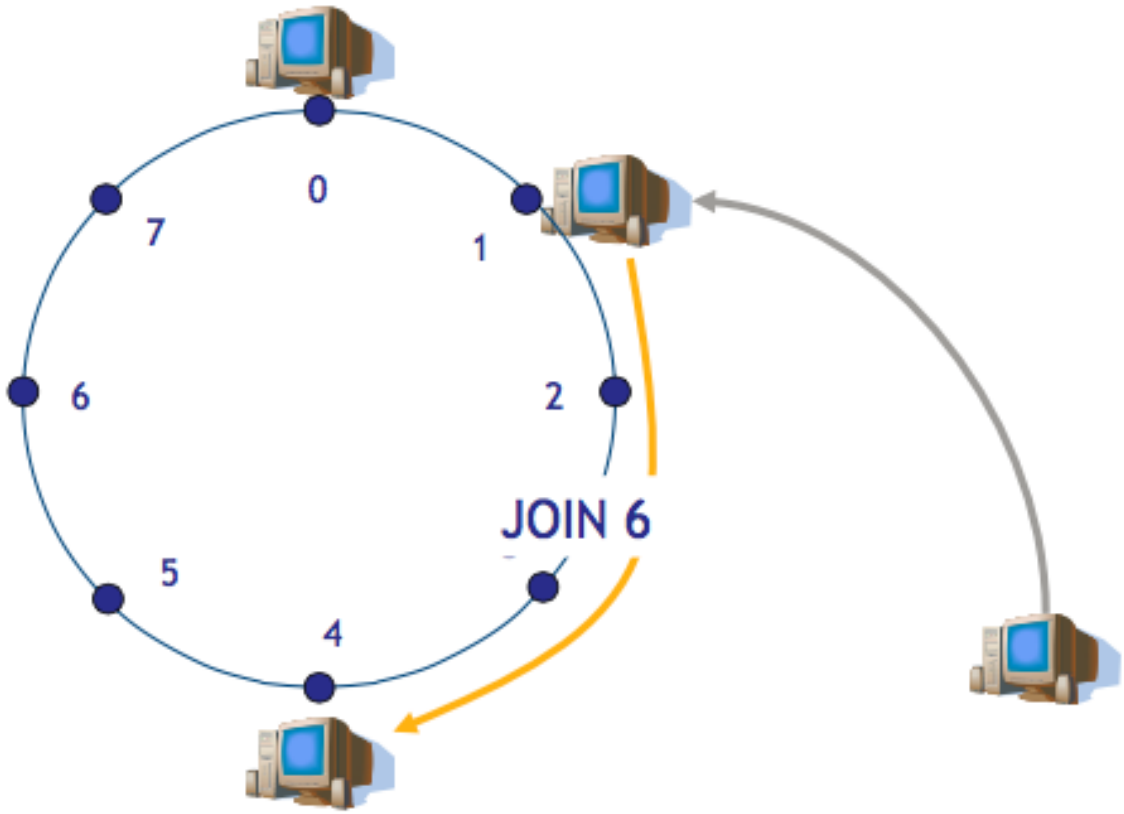


Joining: Step-By-Step Example: Contact known node

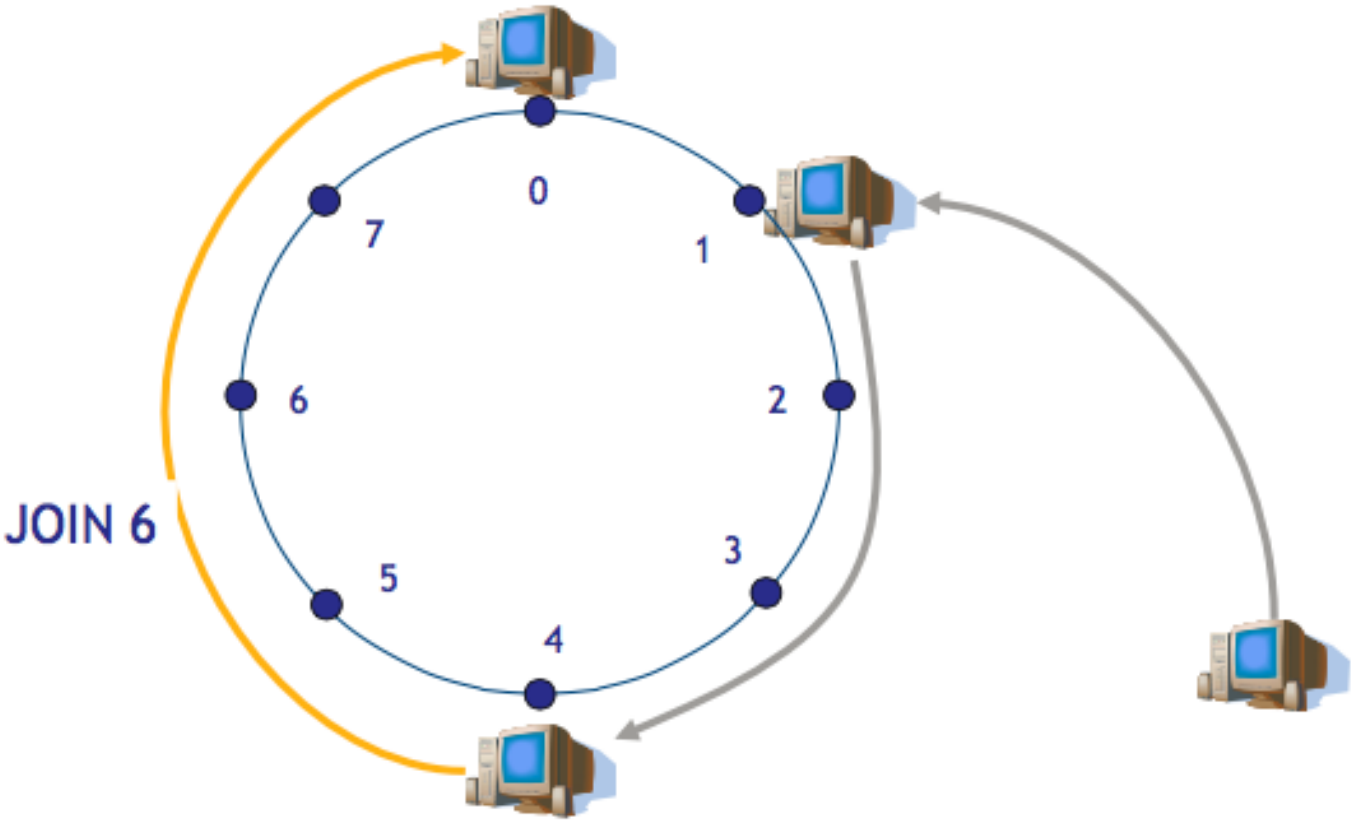
- Arrows indicate open connections
- Example assumes connections are kept open, i.e., messages processed recursively
- Iterative processing is also possible



Joining: Step-By-Step Example: Join gets routed along the network



Joining: Step-By-Step Example: Successor of New Node Found

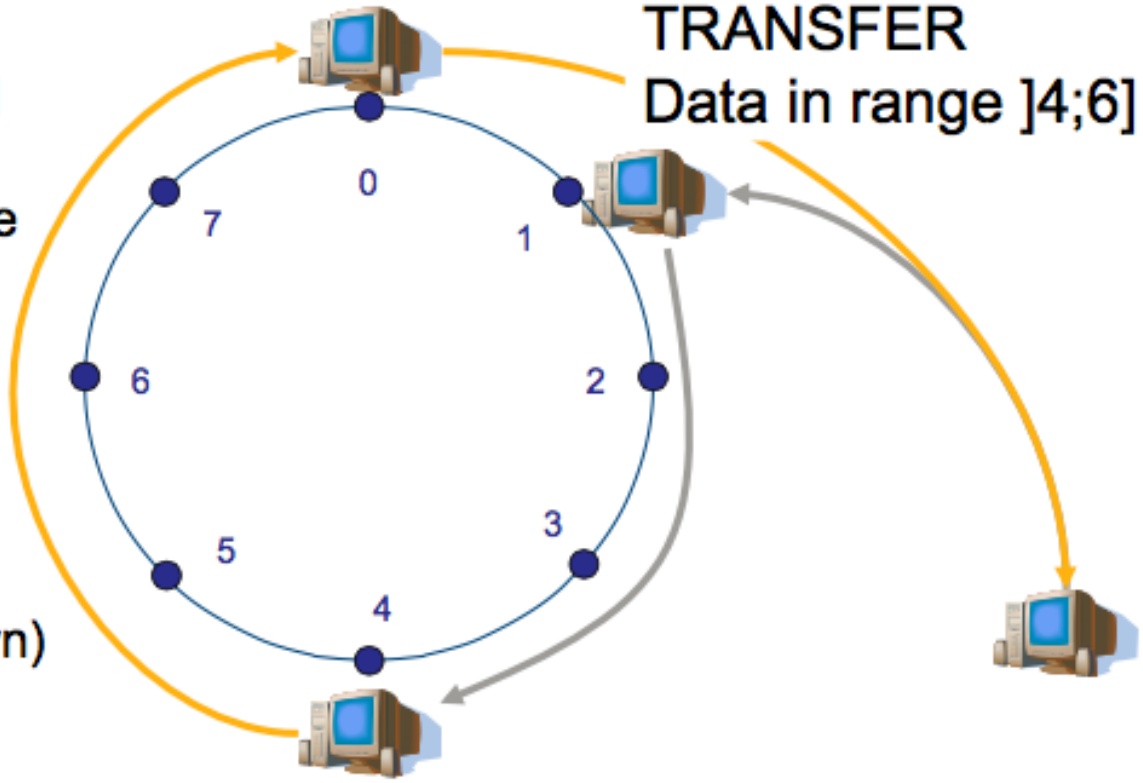


Joining: Step-By-Step Example: Joining Successful + Transfer

Joining is successful

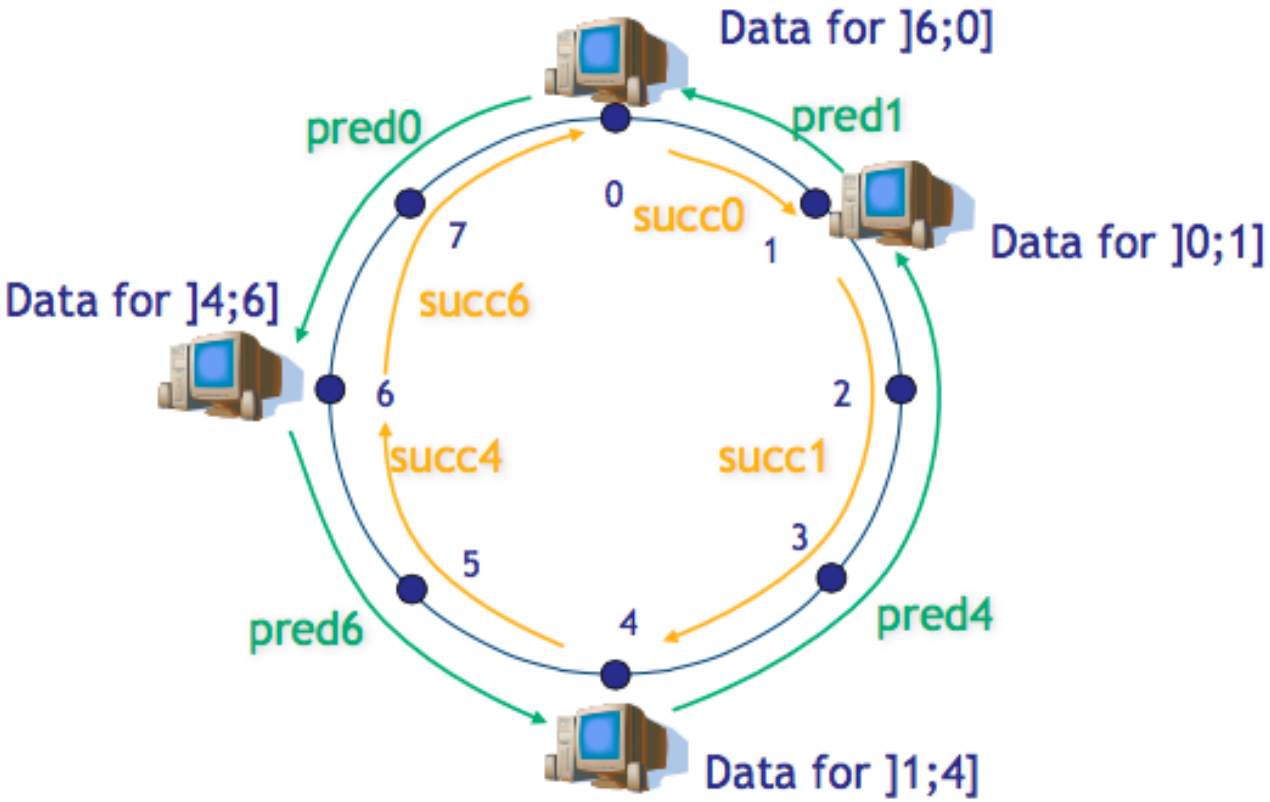
Old responsible node transfers data that should be in new node

New node informs Node4 about new successor (not shown)



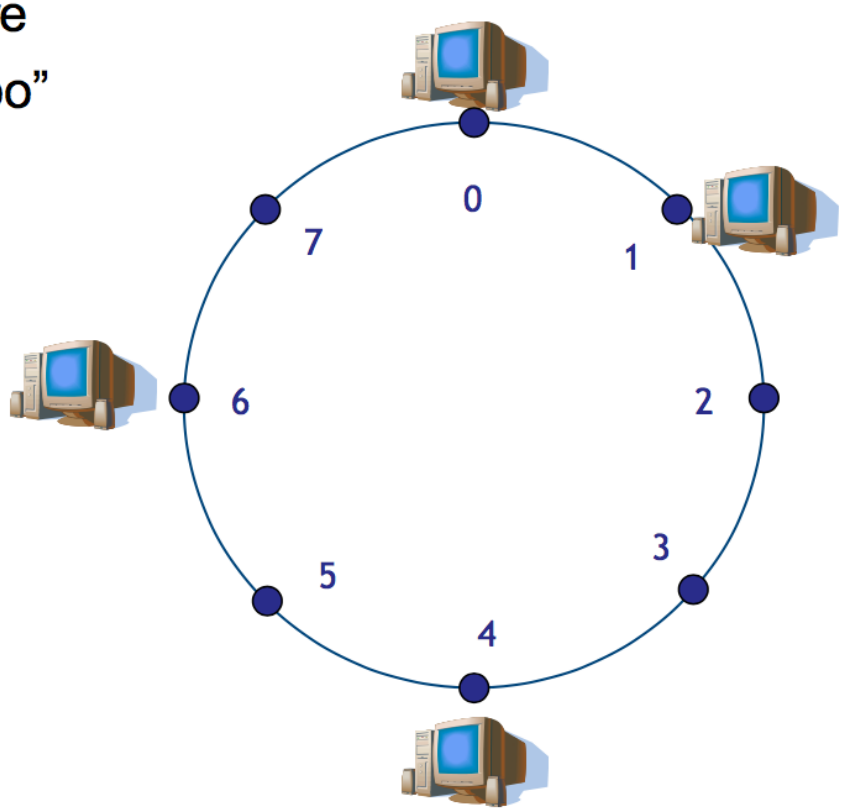
Note: Transferring can happen also later

Joining: Step-By-Step Example: All Is Done

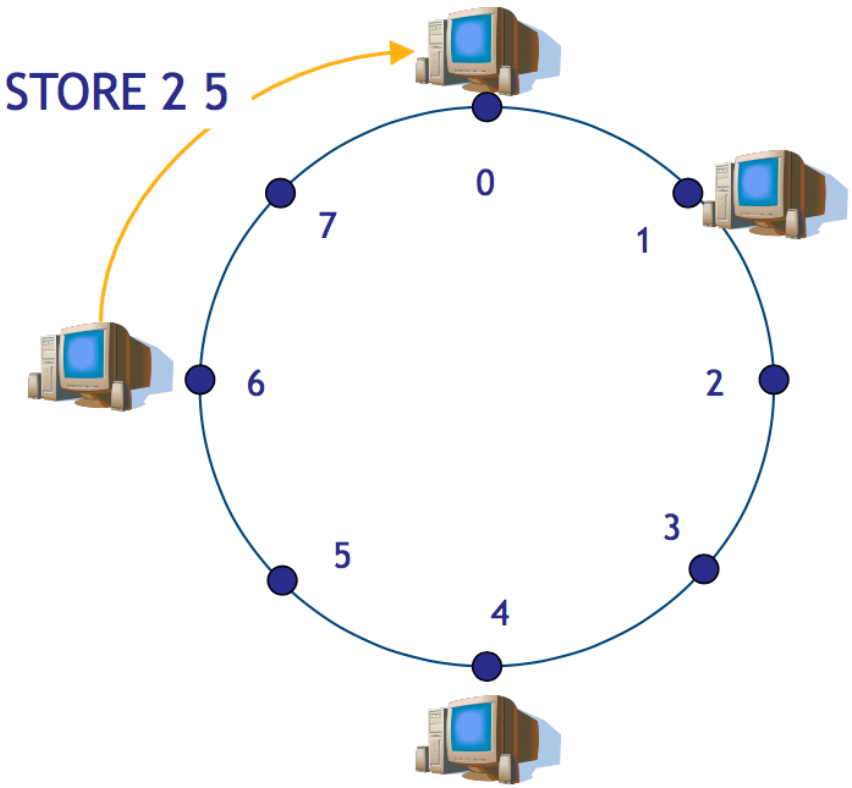


Storing a Value

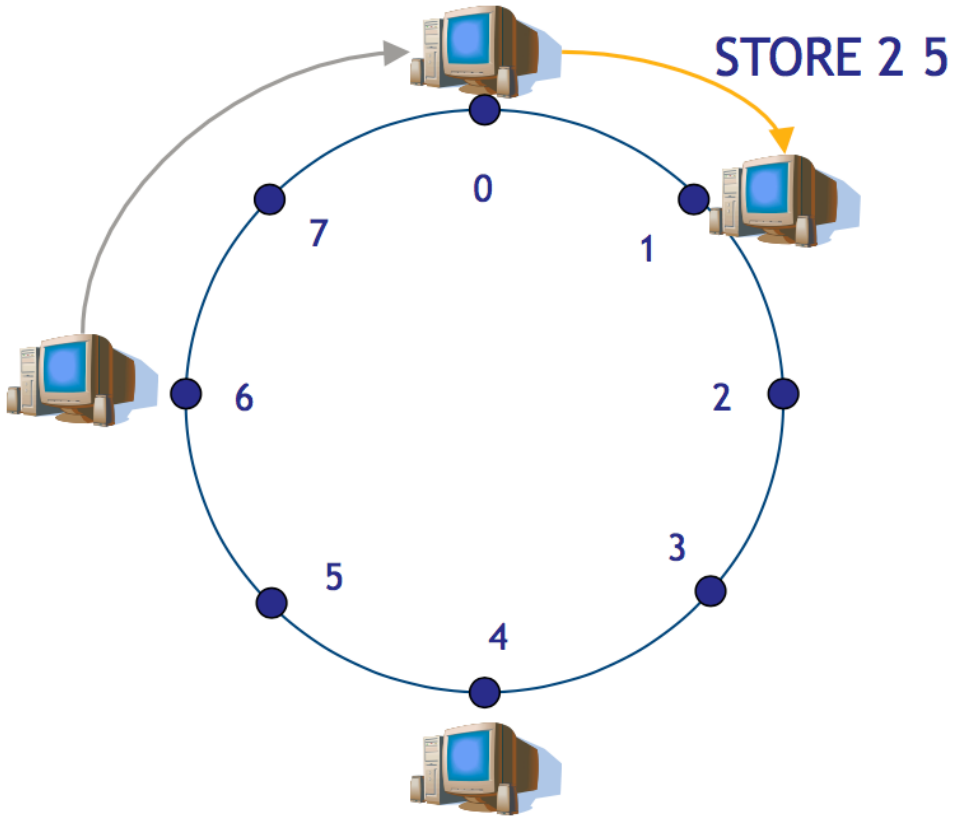
- Node 6 wants to store object with name “Foo” and value 5
- hash(Foo) = 2



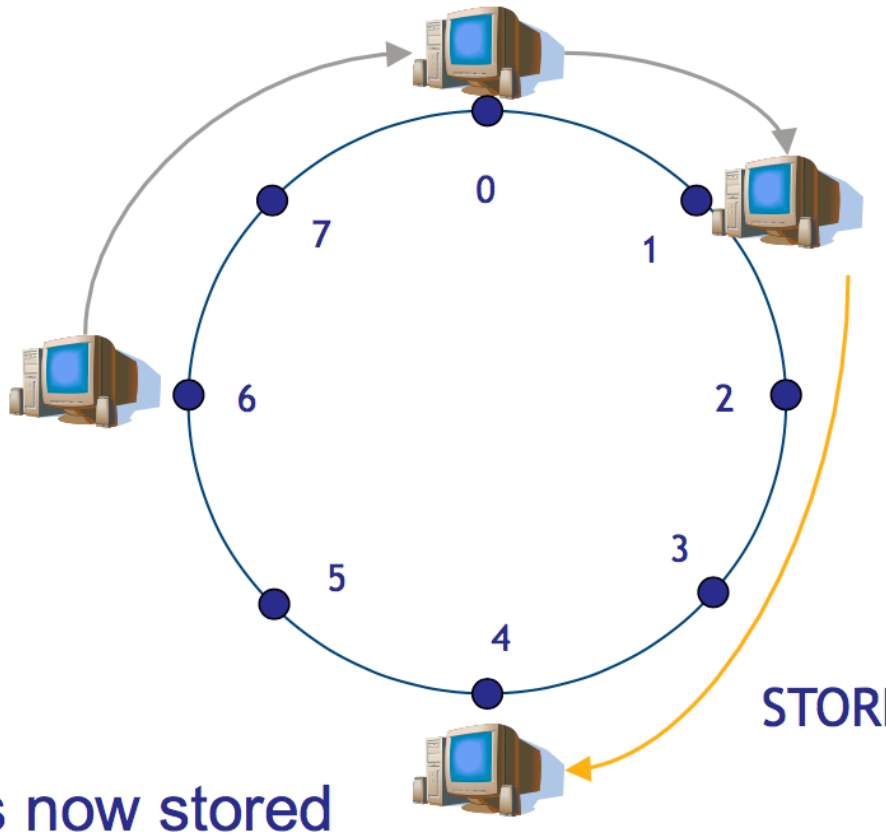
Storing a Value



Storing a Value



Storing a Value

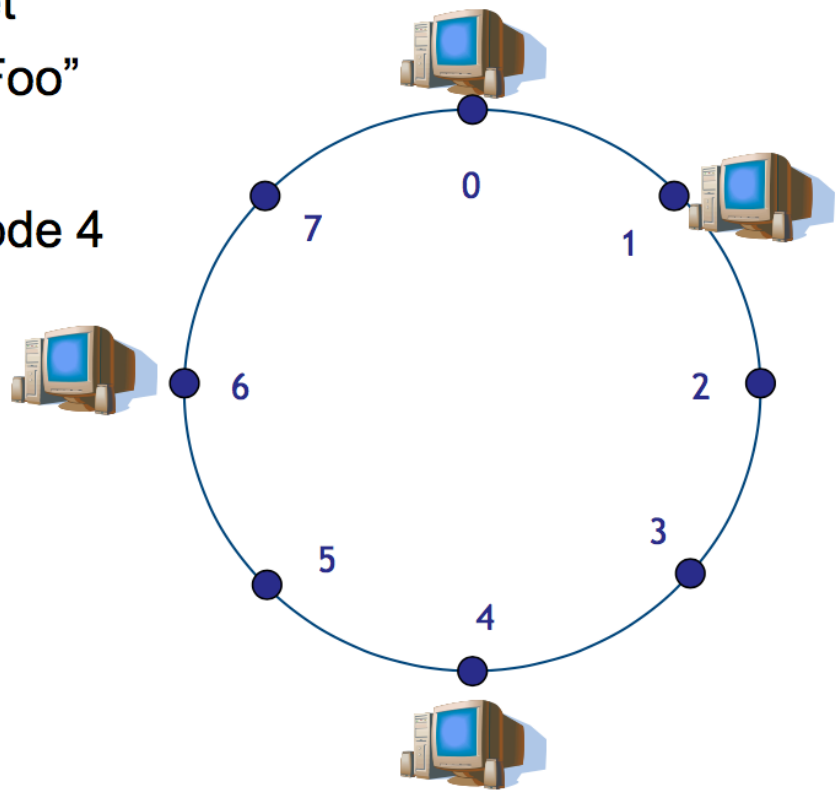


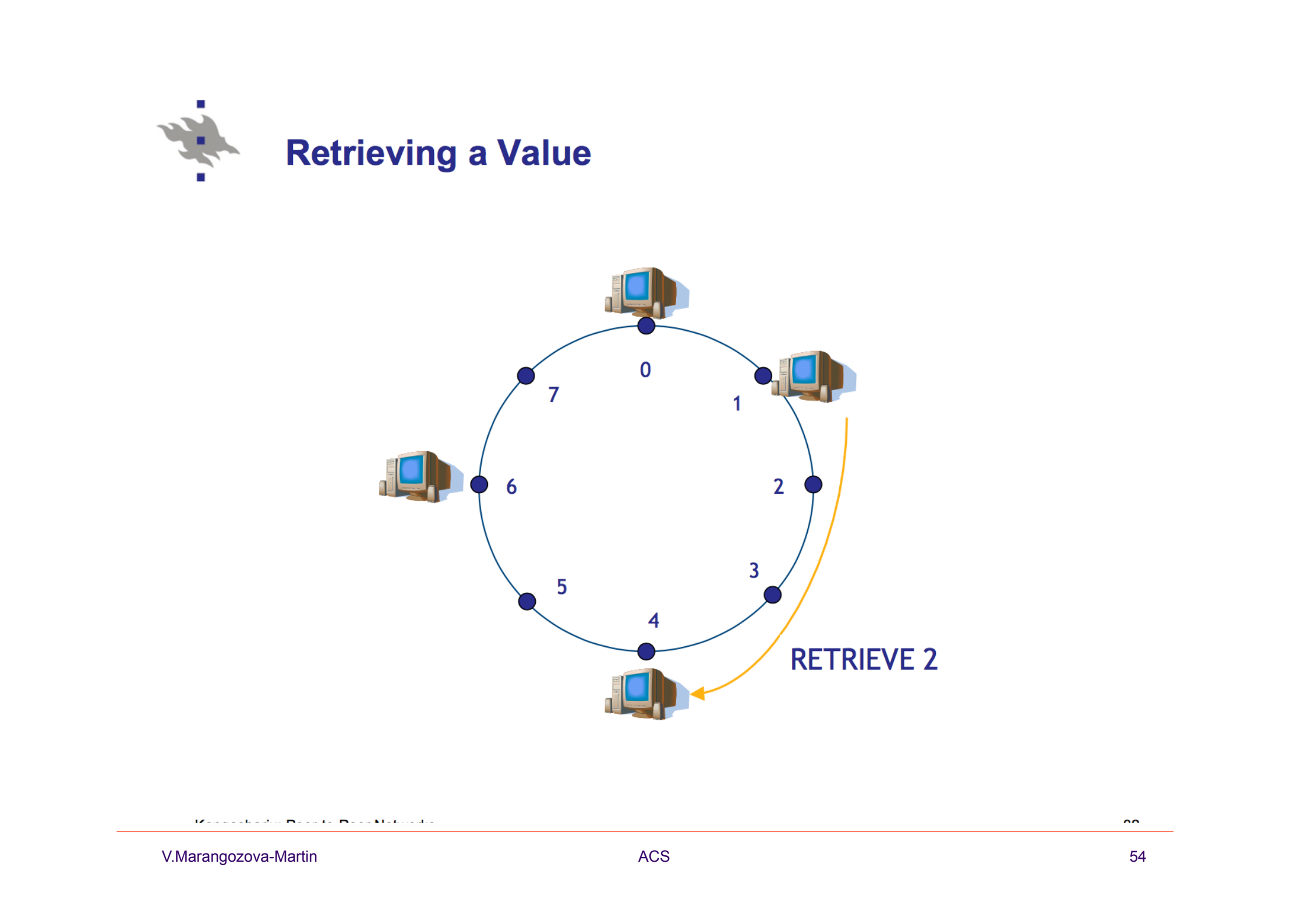
Value is now stored in node 4.

Retrieving a Value

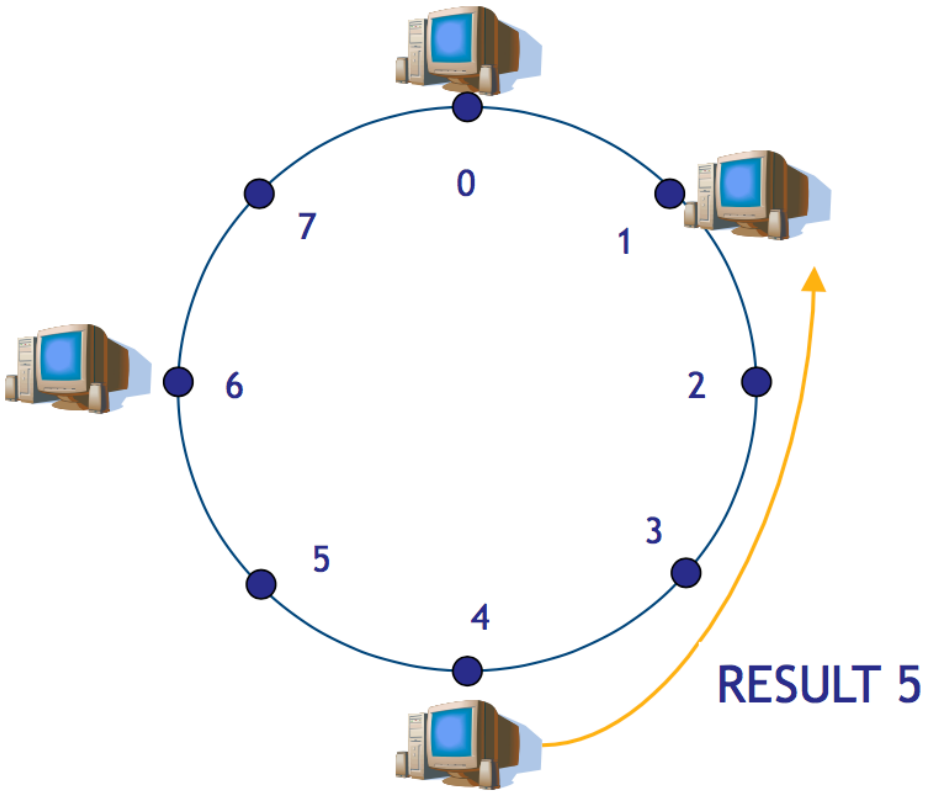
- Node 1 wants to get object with name “Foo”
- hash(Foo) = 2

→ Foo is stored on node 4




Retrieving a Value
0
1
2
3
4
5
6
7
RETRIEVE 2

Retrieving a Value



Chord: Scalable Routing

- Routing happens by passing message to successor
- What happens when there are 1 million nodes?
  - On average, need to route 1/2-way across the ring
  - In other words, 0.5 million hops! Complexity O(n)
- How to make routing scalable?
  - Answer: Finger tables
- Basic Chord keeps track of predecessor and successor
  - Finger tables keep track of more nodes
  - Allow for faster routing by jumping long way across the ring
  - Routing scales well, but need more state information
  - Finger tables not needed for correctness, only performance improvement

Chord Performance

- Search performance of “pure” Chord O(n)
  - Number of nodes is n
- With finger tables, need O(log n) hops to find the correct node
  - Fingers separated by at least 2i-1
  - With high probability, distance to target halves at each step
  - In beginning, distance is at most 2m
  - Hence, we need at most m hops
- For state information, “pure” Chord has only successor and
- predecessor, O(1) state
  - For finger tables, need m entries
  - Actually, only O(log n) are distinct
  - Proof is in the paper

DHT: Comparison

	Chord	CAN	Tapestry
Type of network	Ring	N-dimensional	Prefix routing
Routing	$O(\log n)$	$O(d \cdot n^{1/d})$	$O(\log_b N)$
State	$O(\log n)$	$O(d)$	$O(b \cdot \log_b N)$
Caching efficient	+	++	++
Robustness	-/+	+++	++
IP Topology-Aware	N	N/Y	Y
Used for other projects	+++	--	++

References

- This lecture is based on
  - George Coulouris, Jean Dollimore and Tim Kindberg, **Distributed Systems Concepts and Design**
  - Lectures of **Prof. Jussi Kangasharju,** http://www.cs.helsinki.fi/u/jakangas/