Peer-to-Peer Systems

Agenda

- Peer-to-peer networking
  - Definition
  - Use cases

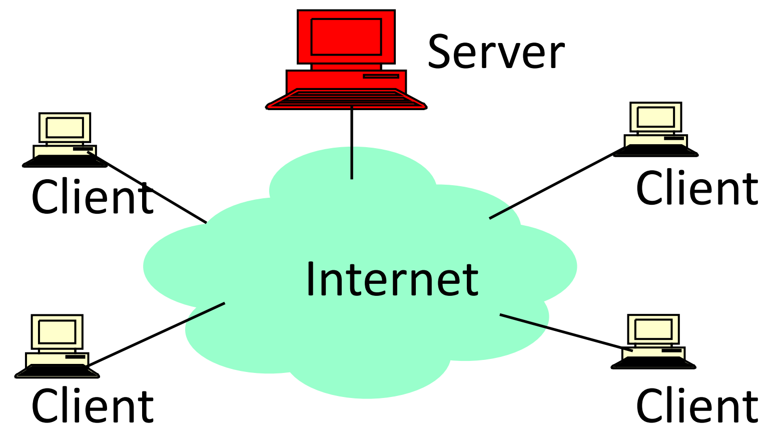
- Peer-to-peer overlays
  - Unstructured systems
  - Distributed hash table abstraction
  - Structured systems realizing DHTs

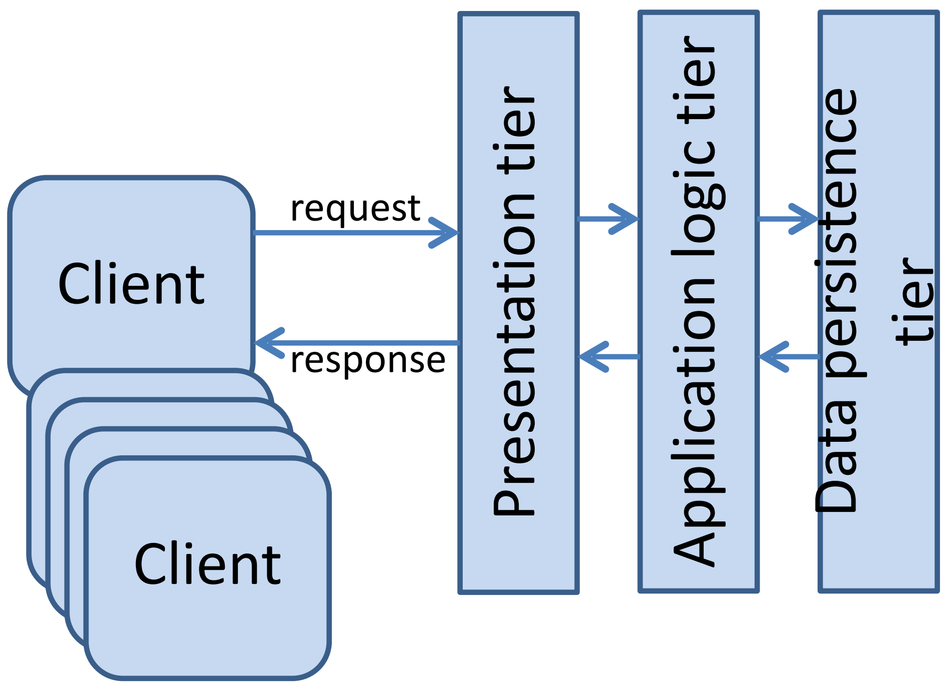
Peer-to-Peer Systems

Introduction

Client-Server Architecture

- Well known, powerful, reliable server as data source
- Clients request data from server
- Very successful model
  - WWW (HTTP), FTP, Web services, etc.
- N-tier architecture





Client-Server Limitations

- Scalability is expensive (vertical vs. horizontal)
- Presents a single point of failure
- Requires administration
- Unused resources at network edge

- P2P systems try to address these limitations and leverage (otherwise) unused resources

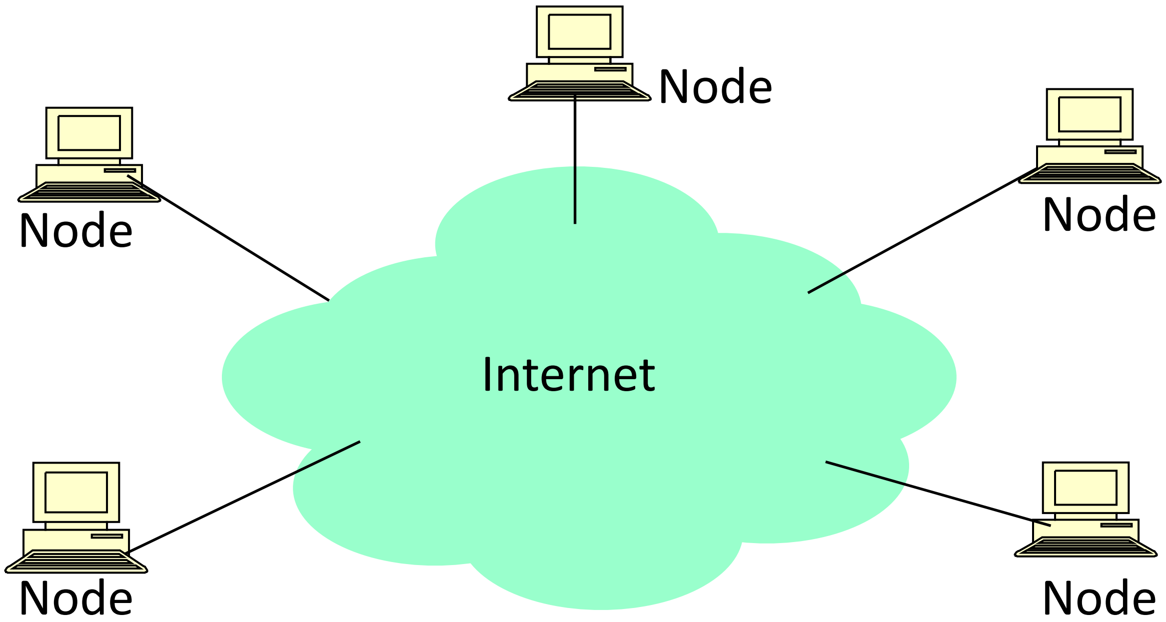
Peer-to-Peer

Compute, storage, network

- P2P computing is the sharing of **compute resources** and **services** by **direct** exchange between peers (a.k.a. nodes)
- These resources and services include the **exchange of data, processing cycles, cache storage**, **disk storage,** and **bandwidth**
- P2P computing takes advantage of existing computing power, computer storage and networking connectivity, allowing users to leverage their **collective power to the ' benefit' of all**

* From (accessed June, 2004) http://www-sop.inria.fr/mistral/personnel/Robin.Groenevelt/ Publications/Peer-to-Peer_Introduction_Feb.ppt

What is a P2P system?

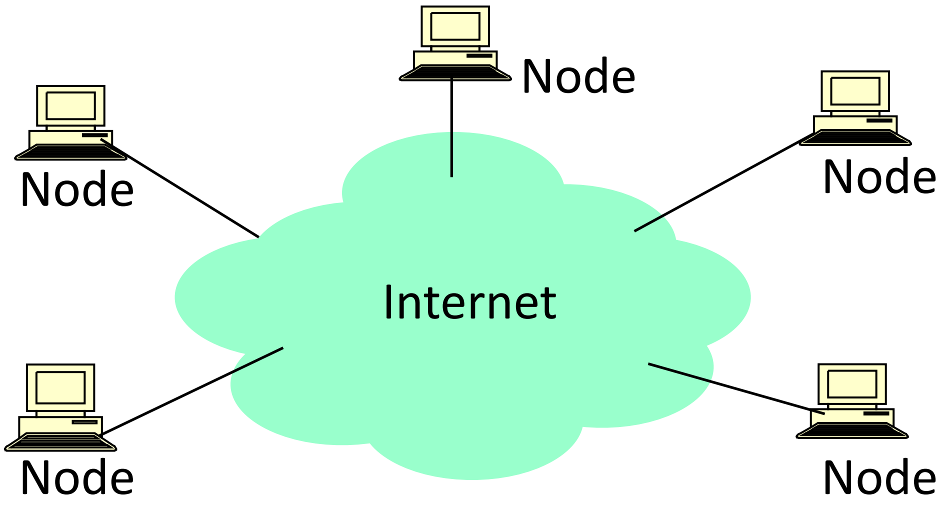


- A distributed system architecture
  - No centralized control
  - Nodes are symmetric in function
- Larger number of unreliable nodes
- Enabled by technology improvements

* Content from http://project-iris.net/dht-toronto-03.ppt

P2P Architecture

- All nodes are both *clients* and *servers*
  - Provide and consume
  - Any node can initiate a connection



- No centralized data source
  - *"The ultimate form of democracy on the Internet"*
  - *"The ultimate threat to copyright protection on the Internet"*

- In practice, hybrid models are popular
  - Combination of client-server & peer-to-peer
  - Skype (early days, now unknown)
  - Spotify (peer-assisted)
  - BitTorrent (trackers)

* Content from http://project-iris.net/dht-toronto-03.ppt

P2P Benefits I

- Efficient use of resources
  - Unused bandwidth, storage, processing power at the edge of network

- Scalability
  - Consumers of resources also donate resources
  - Aggregate resources grow naturally with utilization
    - Organic scaling (more users, more resources)
    - "Infrastructure-less" scaling

- **Caveat**: It is not a one size fits all
  - Large companies are not switching in droves to P2P

P2P Benefits II

- Reliability (in aggregate)
  - Many replicas, redundancy
  - Geographic distribution
  - No single point of failure and control

- Ease of administration
  - Nodes self-organize
  - No need to deploy servers to satisfy demand
  - Built-in fault-tolerance, replication, and load balancing

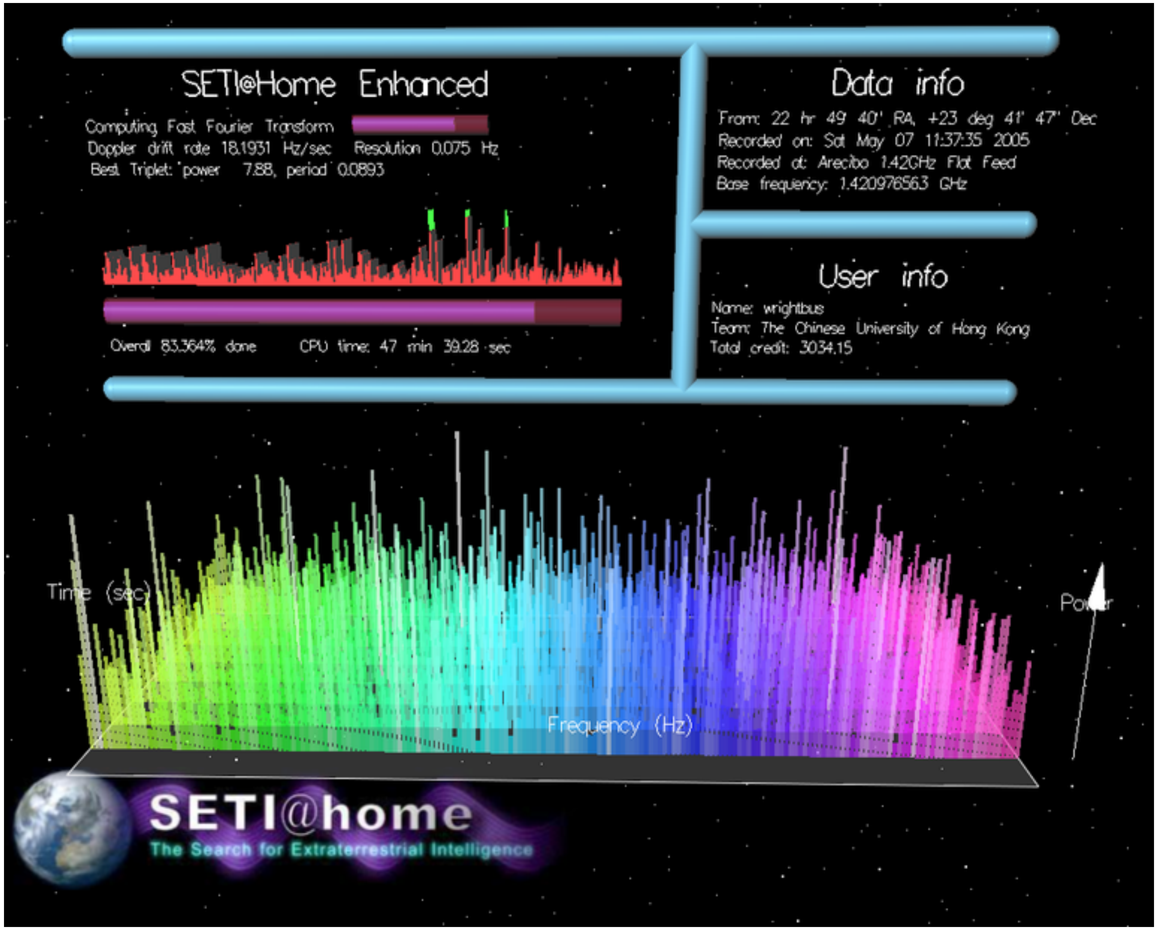
Use Cases: Large-Scale Systems

- Some applications require immense resources
  - CPU: Scientific data analysis (*@home)
  - Bandwidth: Streaming, file sharing
  - Storage: Decentralized data, file sharing

- Thousands or even millions of nodes
  - How to efficiently manage such a large network?

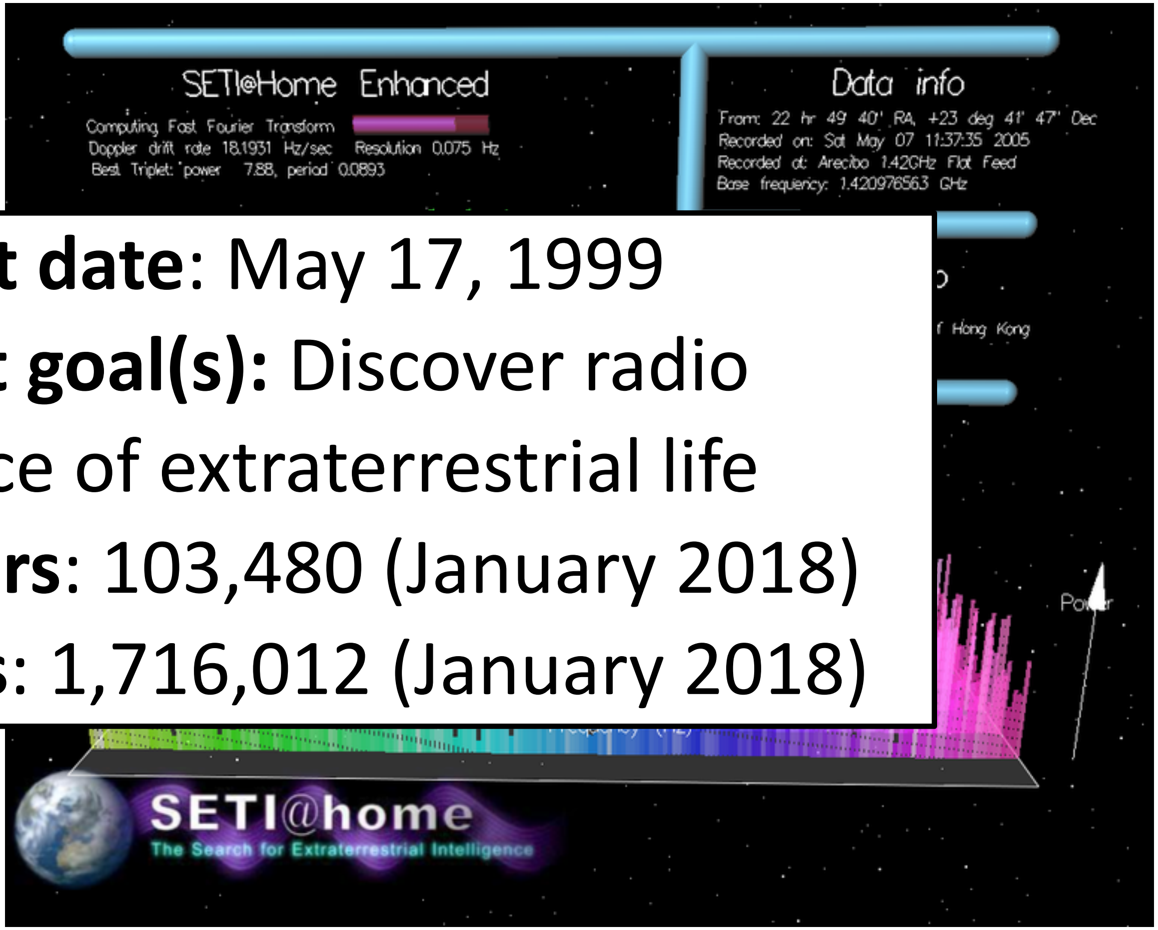
SETI@home – started in 1999

- 5.2 million participants worldwide
- On September 26, 2001, had performed a total of 10^{21} flops
- 35 GB/data per day, 140K work units
- 30 hours/WU
- 4.2M hours of computation/day
- Centralized database



SETI@home – started in 1999

- 5.2 million participants worldwide
- On Se 2001, perfo 10^{21} f
- 35 GE 140K
- 30 ho
- 4.2M hours of computation/day
- Centralized database



Start date: May 17, 1999

Project goal(s): Discover radio evidence of extraterrestrial life

Active users: 103,480 (January 2018)

Total users: 1,716,012 (January 2018)

2015 NA Traffic

Traffic share in North America during peak hours

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	28.56%	Netflix	37.05%	Netflix	34.70%
2	Netflix	6.78%	YouTube	17.85%	YouTube	16.88%
3	HTTP	5.93%	HTTP	6.06%	HTTP	6.05%
4	Google Cloud	5.30%	Amazon Video	3.11%	BitTorrent	4.35%
5	YouTube	5.21%	iTunes	2.79%	Amazon Video	2.94%
6	SSL - OTHER	5.10%	BitTorrent	2.67%	iTunes	2.62%
7	iCloud	3.08%	Hulu	2.58%	Facebook	2.51%
8	FaceTime	2.55%	Facebook	2.53%	Hulu	2.48%
9	Facebook	2.25%	MPEG - OTHER	2.30%	MPEG	2.16%
10	Dropbox	1.18%	SSL - OTHER	1.73%	SSL - OTHER	1.99%
		65.95%		78.69%		76.68%

sandvine

https://www.sandvine.com

2015 NA Traffic

Still #1 in upstream!

Traffic share in North America during peak hours

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	28.56%	Netflix	37.05%	Netflix	34.70%
2	Netflix	6.78%	YouTube	17.85%	YouTube	16.88%
3	HTTP	5.93%	HTTP	6.06%	HTTP	6.05%
4	Google Cloud	5.30%	Amazon Video	3.11%	BitTorrent	4.35%
5	YouTube	5.21%	iTunes	2.79%	Amazon Video	2.94%
6	SSL - OTHER	5.10%	BitTorrent	2.67%	iTunes	2.62%
7	iCloud	3.08%	Hulu	2.58%	Facebook	2.51%
8	FaceTime	2.55%	Facebook	2.53%	Hulu	2.48%
9	Facebook	2.25%	MPEG - OTHER	2.30%	MPEG	2.16%
10	Dropbox	1.18%	SSL - OTHER	1.73%	SSL - OTHER	1.99%
		65.95%		78.69%		76.68%

sandvine

https://www.sandvine.com

2015 NA Traffic

Still #1 in upstream!

Traffic share in North America during peak hours

But streaming is larger overall…

	Upstream		Downstream		Aggregate	
Rank	Application	Share	Application	Share	Application	Share
1	BitTorrent	28.56%	Netflix	37.05%	Netflix	34.70%
2	Netflix	6.78%	YouTube	17.85%	YouTube	16.88%
3	HTTP	5.93%	HTTP	6.06%	HTTP	6.05%
4	Google Cloud	5.30%	Amazon Video	3.11%	BitTorrent	4.35%
5	YouTube	5.21%	iTunes	2.79%	Amazon Video	2.94%
6	SSL - OTHER	5.10%	BitTorrent	2.67%	iTunes	2.62%
7	iCloud	3.08%	Hulu	2.58%	Facebook	2.51%
8	FaceTime	2.55%	Facebook	2.53%	Hulu	2.48%
9	Facebook	2.25%	MPEG - OTHER	2.30%	MPEG	2.16%
10	Dropbox	1.18%	SSL - OTHER	1.73%	SSL - OTHER	1.99%
		65.95%		78.69%		76.68%

sandvine

https://www.sandvine.com

P2P File Sharing Systems

- Large-scale sharing of files
  - User *A* makes files (music, video, etc.) on their computer available to others
  - User *B* connects to "network," searches for files and downloads files *directly* from User *A*

- P2P networks
  - Peers are connected to each other to form an **overlay network**
  - Peers communicate using links established in overlay

- Fallen out of favor (*RIP 1999-2015*)
  - Issues of copyright infringement
  - Cloud infrastructures has taken over (controlled resources)
  - Harder to exploit mobile and connected devices
  - Streaming makes file sharing obsolete (cf. P2P Streaming)

Types of P2P Systems

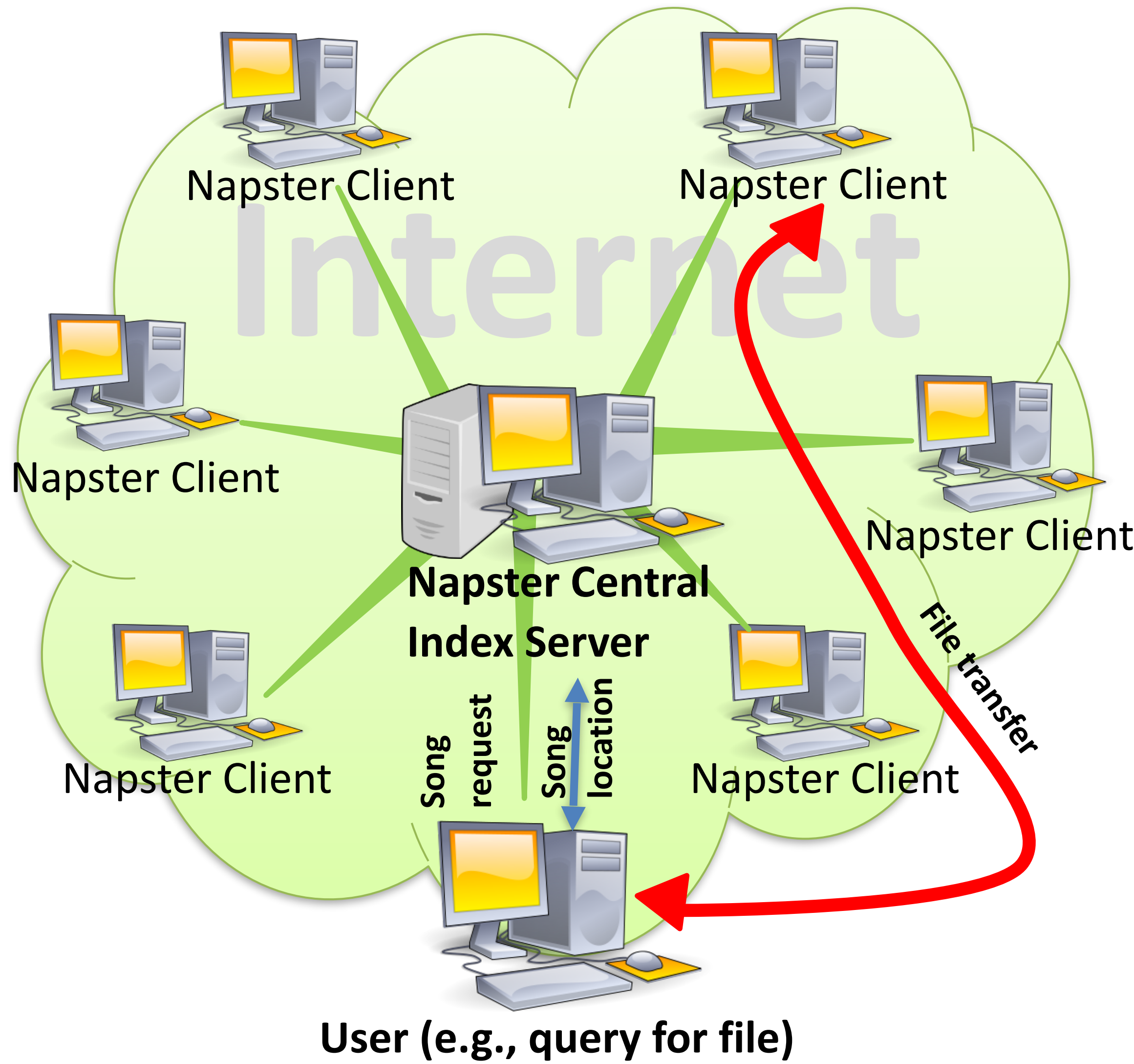
- Unstructured networks
  - No obvious structure in overlay topology
  - Peers simply connect to anyone in existing network
- First generation:
  - Centralized: **Napster**
  - Pure: **Gnutella**, Freenet
- Second generation:
  - Dynamic “supernodes”
  - Hybrid: **Skype**, **Spotify**, FastTrack, eDonkey, **BitTorrent**
- Structured networks
  - Topology of overlay is controlled: peers’ connections are fixed
  - Based on the distributed hash table abstraction (**DHT**)

Peer-to-Peer Systems

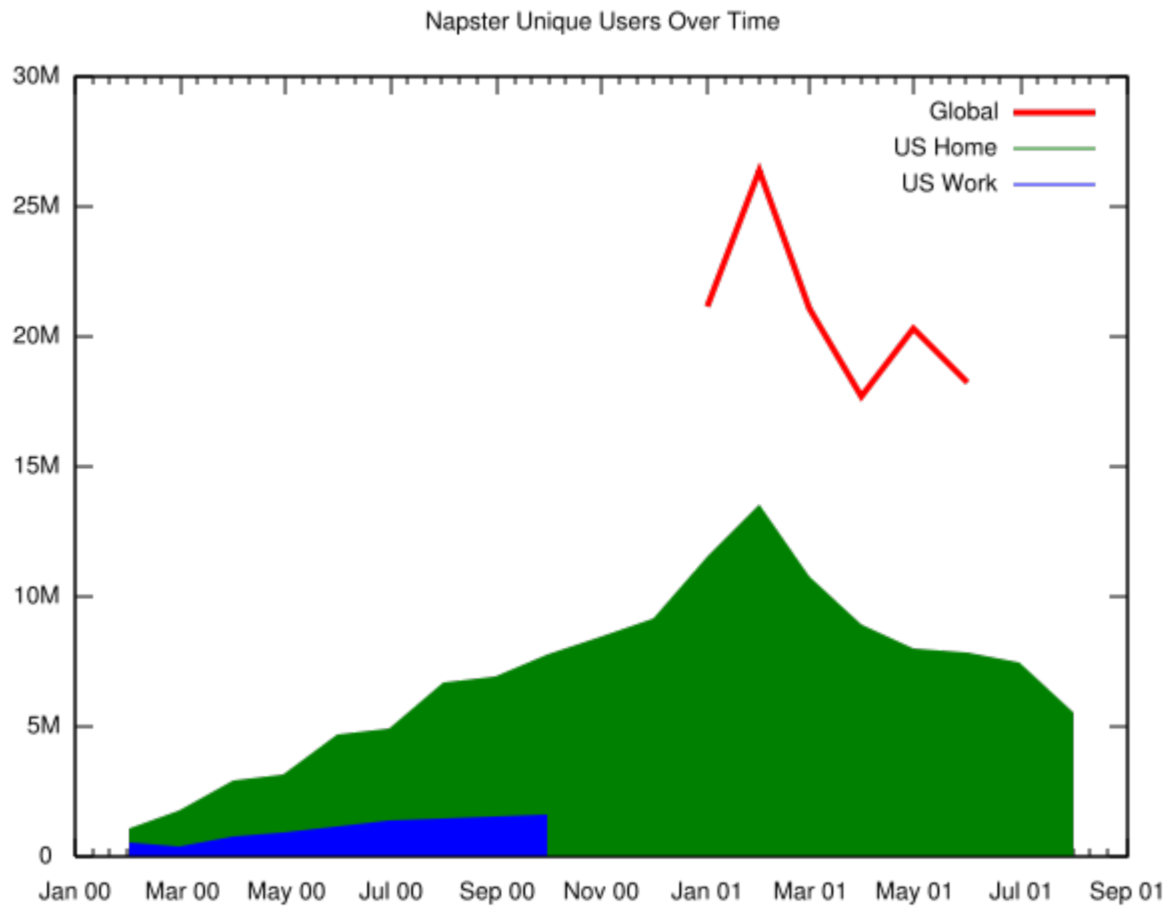
Unstructured peer-to-peer systems

Centralized: Napster “June 1999 – July 2001”

- Centralized search indexes music files
  - Perfect knowledge
  - Bottleneck
- Users query server
  - Keyword search (artist, song, album, bit rate, etc.)
- Napster server replies with IP address of users with matching files
- Querying users connect **directly** to remote node for file to download



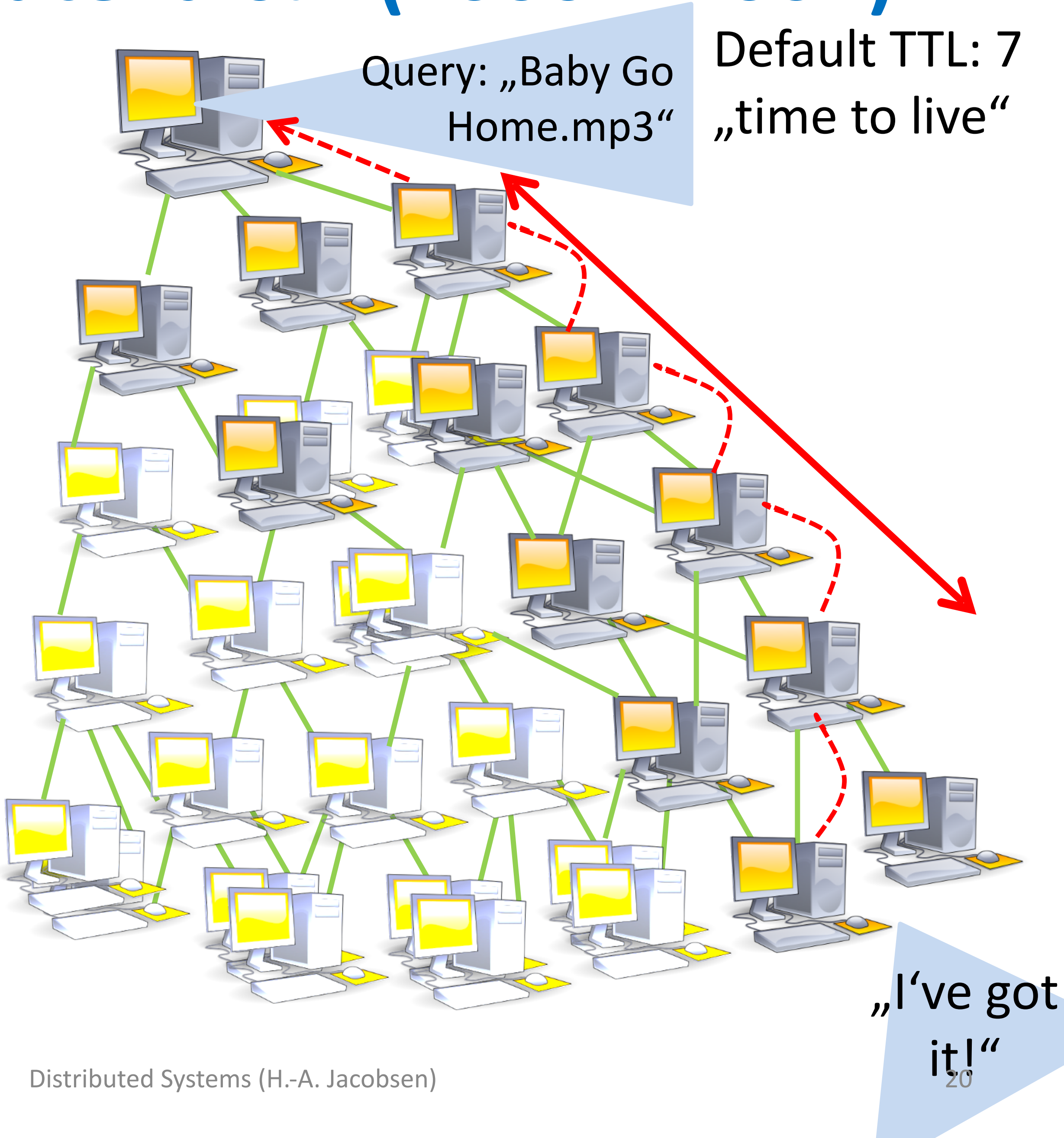
The Rise and Fall of Napster



RIAA shut down Napster easily because it used a central server!

Pure P2P: Gnutella 0.4 (2000 – 2001)

- Share any type of files
- Decentralized search
  - **Imperfect** content availability
- Client connect to (on average) 3 peers
- **Flood a QUERY** to connected peers
- Flooding propagates in network up to TTL
- Users with matching files **reply with QUERYHIT**
- Flooding wastes **bandwidth**: Later versions used more sophisticated search



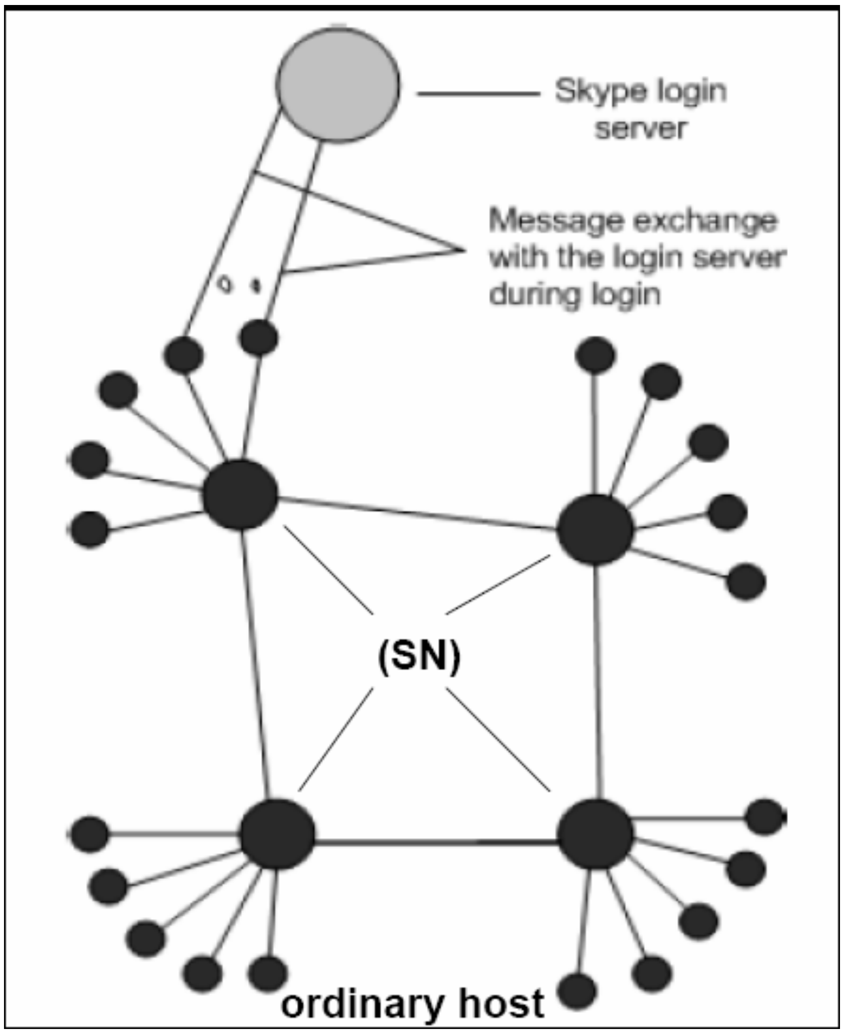
Hybrid P2P

- Both previous approaches have advantages and disadvantages
  – Centralized: single point of failure, easy to control, but perfect content availability
  – Pure: decentralized (resistant), costly and unreliable search
- Hybrid P2P combines both approaches
  – Hierarchy of peers
  – Superpeers with more capacity, discovered **dynamically**
  – Normal peers (leaf nodes) are users
- Superpeer responsibilities
  – Participates in search protocol, indexes and caches data
  – Improves content availability
  – Reduces message load

Skype Network

Around 2004

- **Super Nodes**: Any node with a public IP address having sufficient CPU, memory and network bandwidth is candidate to become a superpeer
- **Ordinary Host**: Host needs to connect to superpeer and must register itself with the Skype login server
- Login server and PSTN gateway (not shown) **are centralized components**



Responsibilities of Superpeers

In Skype Around 2004

- Indexes user directory
  - Distributed among superpeers
  - Communication among superpeers for lookup

- Communication relay
  - NAT traversal

- Phased out by Microsoft in 2011 (speculation)
  - Replaced with private servers
  - “[T]hat is in part why Skype has switched to server-based "dedicated supernodes"... nodes that **we control**, can handle orders of magnitudes **more clients per host**, are in **protected data centers** and up all the time, and **running code that is less complex** than the entire client code base. ”

Skype Impact

- Skype has shown, at least has suggested,:
  - Signaling, unique property of traditional phone system, is accomplished effortlessly with self-organizing P2P networks
  - P2P overlay networks can scale up to handle large-scale connection-oriented real-time services such as video and voice
- AT&T: *"The end of landlines …"*

Peer-to-Peer Systems

DHT - Distributed Hash Table

Structured Peer-to-Peer Systems

- Third generation P2P overlay networks
- Self-organizing, load balanced, fault-tolerant
- "Fast" and efficient **lookup guarantees**, e.g.,
  - $O(\log(n))$ lookups
  - O(1): centralized, $O(n)$: pure, $O(\#S)$: hybrid
- Based on a hash table interface (cf. KV-Store)
  - Put(Key, Data) and Get(Key)
  - Coined term **distributed hash table** (DHT)
- Systems: Chord, CAN, Pastry, etc.

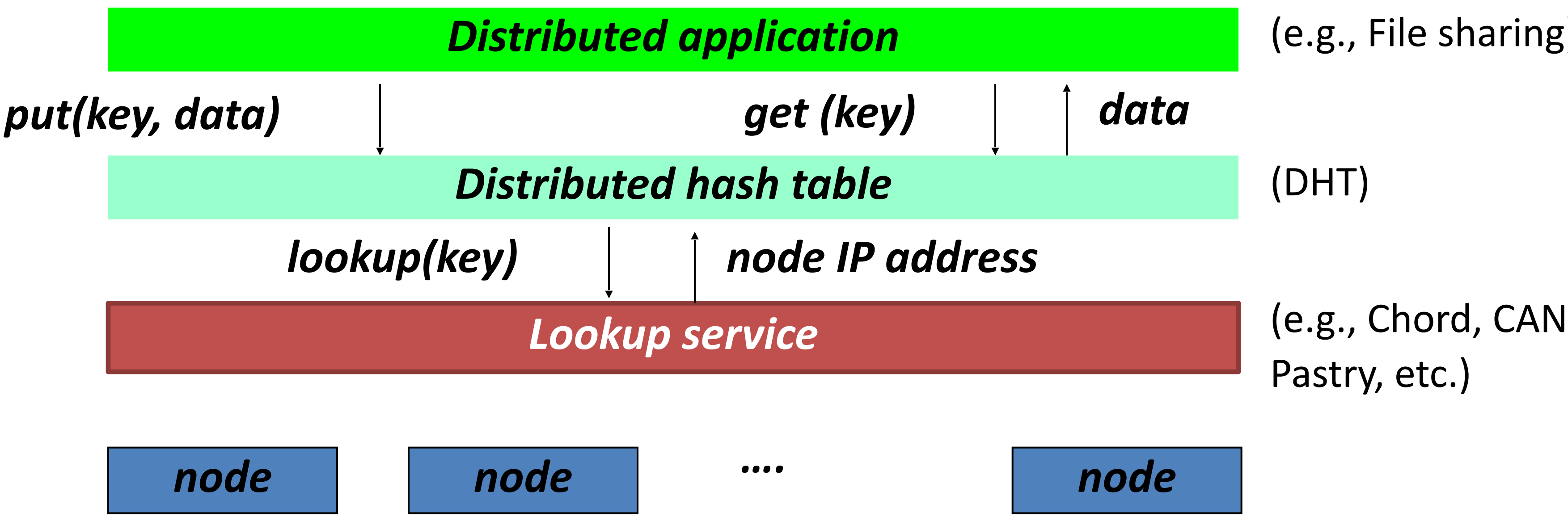
Distributed Hash Tables (DHT)

- Distributed version of a hash table data structure

- Store and retrieve (key, value)-pairs
  - **Key** is like a filename, hash of name, hash of content (since name could change)
  - **Value** is file content
    - Often just a reference to a node with the content

- Keys are hashed and mapped to a set of distributed nodes
  - Realization via consistent hashing *et al*.
  - System change should impact few nodes

DHT Abstraction



- Application distributed over many nodes
- DHT distributes data storage over many nodes

DHT Interface

- Put(key, value) and get(key) → value
  - Simple interface!
- API supports a wide range of applications
  - DHT imposes neither structure nor meaning on keys/values
- Key-value pairs are persisted and globally available
  - Good availability, content stored at edge
  - Store keys in other DHT values
  - Thus, build complex data structures

DHT as Infrastructure or Service

- Many applications can share single DHT service
- Eases deployment of new applications
- Pools resources from many participants (P2P…)
- Essentially, a middleware service, a piece of distributed systems infrastructure

DHT-based Projects

- File sharing [CFS, OceanStore, PAST, Ivy, …]
- Web cache [Squirrel, ..]
- Archival/Backup store [HiveNet, Mojo, Pastiche]
- Censor-resistant stores [Eternity,..]
- DB query and indexing [PIER, …]
- Event notification (Publish/Subscribe) [Scribe, ToPSS]
- Naming systems [ChordDNS, Twine, ..]
- Communication primitives [I3, …]
- Key-value stores [Cassandra*, Dynamo*, …]

Common denominator:

- **Data is location-independent**
- **All leverage DHT abstraction**

* In as far as they use consistent hashing among nodes

Peer-to-Peer Systems

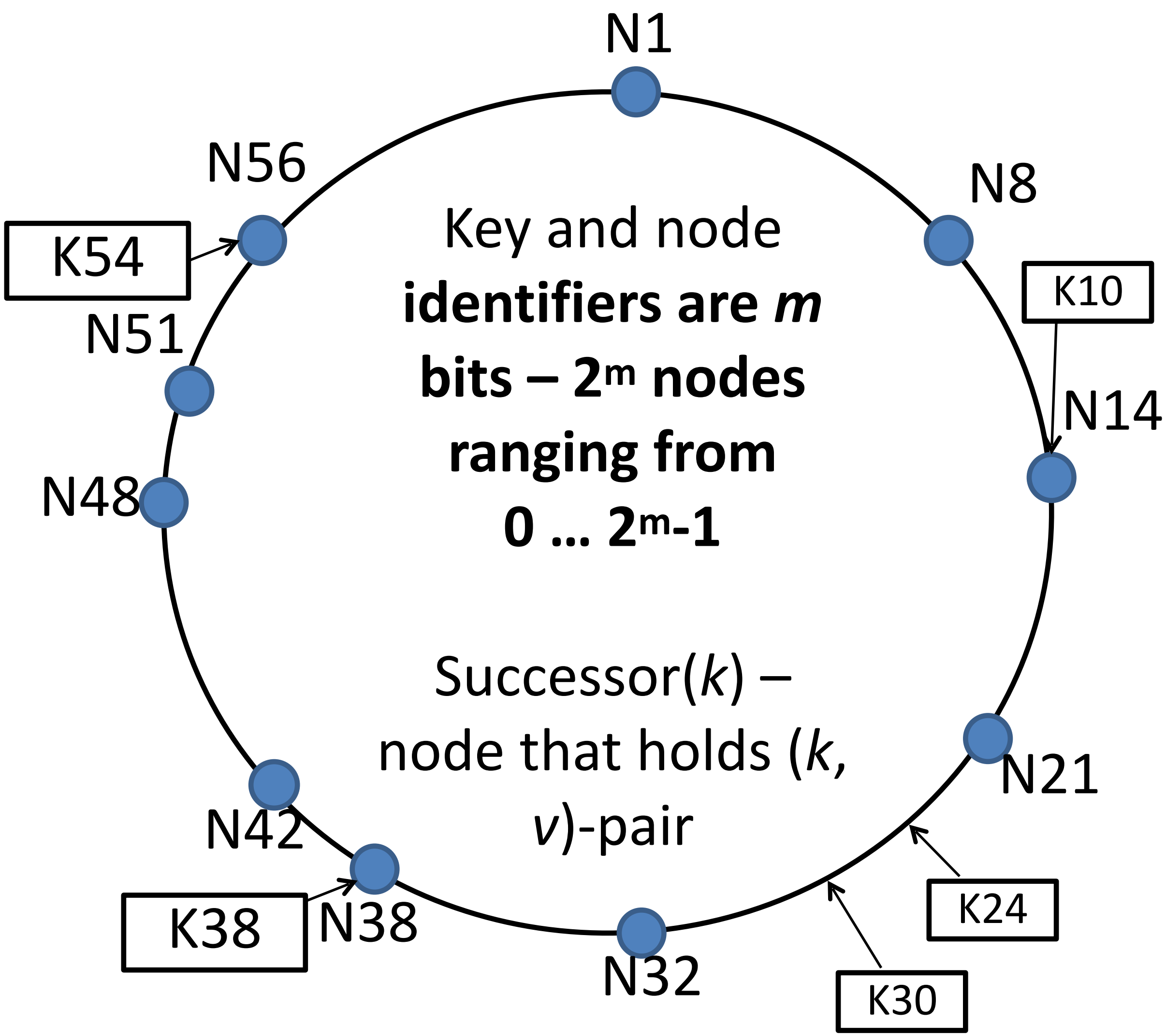
Chord DHT

DHT Requirements

- Keys mapped evenly to all nodes in the network (**load balancing**)
- Node arrivals & departures only affect a few nodes (**low maintenance**)
- Each node maintains information about only a few other nodes (**low maintenance**)
- Messages can be routed to a node efficiently (**fast lookup**)

Chord Identifier Circle

- **Nodes** organized in an identifier circle based on **node identifiers**
- **Keys** assigned to their successor node in the identifier circle
- **Hash function** ensures even distribution of nodes and keys on the identifier circle
- Cf. **consistent hashing**
- With N nodes and K keys each **node is responsible for** roughly K/N keys



* All Chord figures from "Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications", Ion Stoica et al., IEEE/ ACM Transactions on Networking, Feb. 2003.

Node Joins & Leaves

- Nodes may disappear from the network (e.g., failure, departure)
- Each node records a whole **segment of the circle** adjacent to it, i.e., ***r* nodes preceding** and **following** it
- With high probability a node is able to correctly locate its successor or predecessor (even under high node churn)

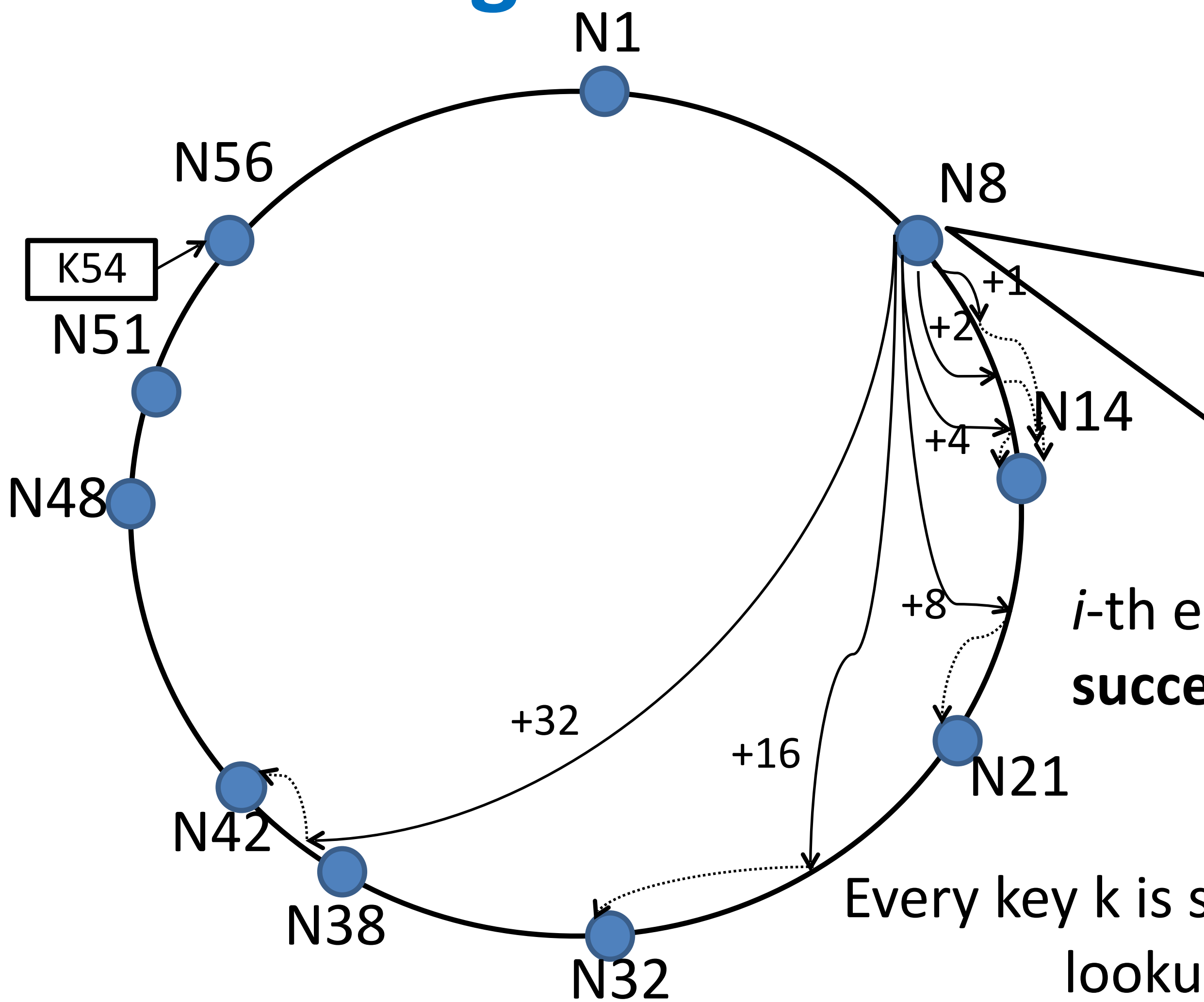
- When a new node joins or leaves the network, responsibility for O(K/N) keys changes hands

Searching in Chord

- With knowledge of a single successor, a linear search through network could locate key (naïve search)
- Any given message may potentially have to be relayed through most of the network, i.e., cost is O(*n*)

- Faster search method requires each node to keep a "***finger table***" (FT) containing up to ***m*** entries
  - ***i***-th entry in FT of node ***n*** contains the address of **successor((n + 2^{i-1}) mod 2^m)**
  - number of nodes that must be contacted to find a successor in an ***n***-node network is **O(log n)**

Chord Finger Table



Finger table

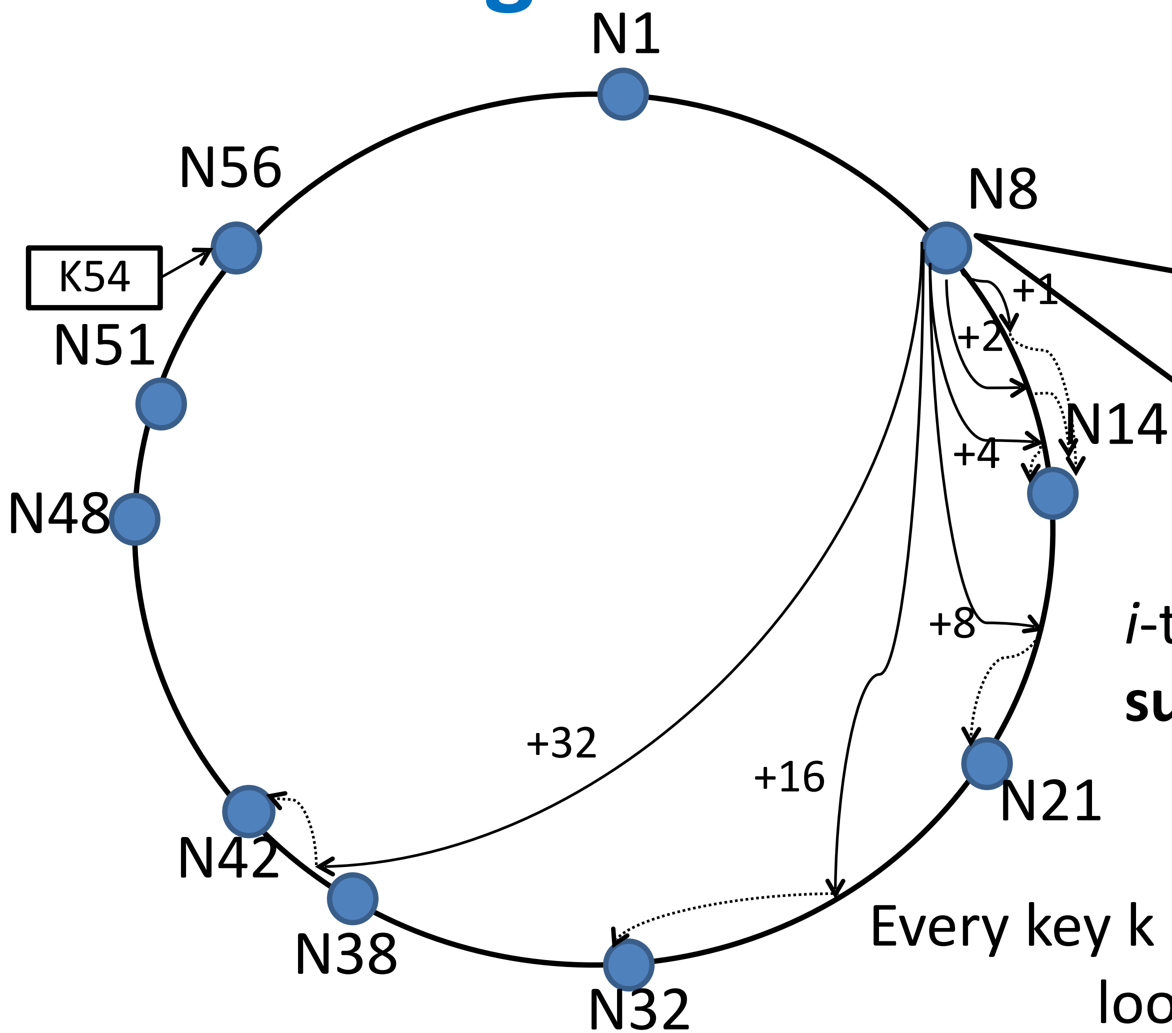
N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 + 16	N32
N8 + 32	N42

i-th entry is address of
successor(($n + 2^{i-1}$) mod 2^m)

Every key k is stored at its successor – lookup a key via successor(*k*)

successor(…) is the node on the circle associated with the input argument – whether it is a *key* or a *node identifier*

Chord Finger Table



Finger table

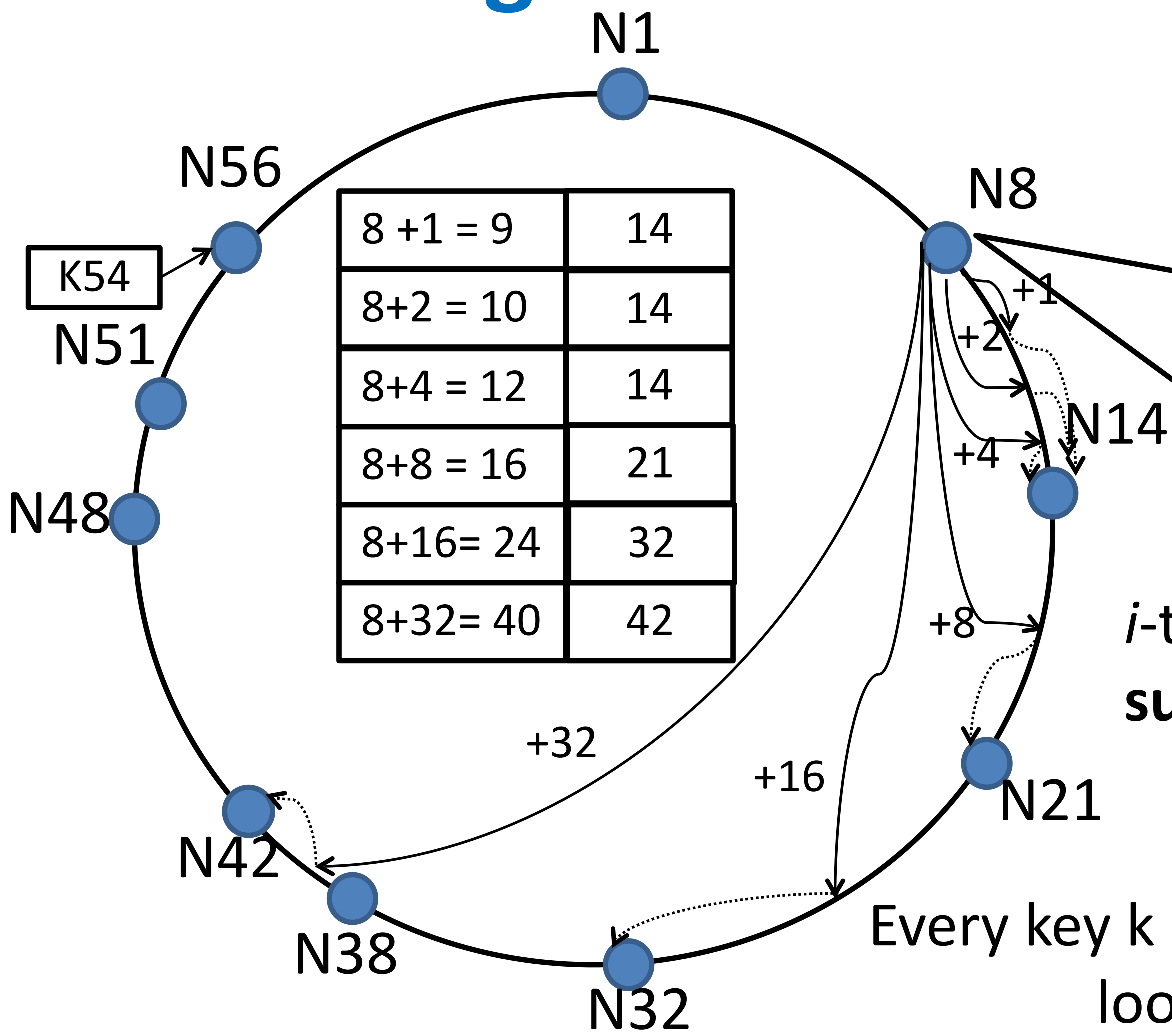
N8 + 1	N14
N8 + 2	N14
N8 + 4	N14
N8 + 8	N21
N8 + 16	N32
N8 + 32	N42

i-th entry is address of **successor((n + 2^{i-1}) mod 2^m)**

Every key k is stored at its successor – lookup a key via successor(*k*)

successor(…) is the node on the circle associated with the input argument – whether it is a *key* or a *node identifier*

Chord Finger Table



8 +1 = 9	14
8+2 = 10	14
8+4 = 12	14
8+8 = 16	21
8+16= 24	32
8+32= 40	42

Finger table

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

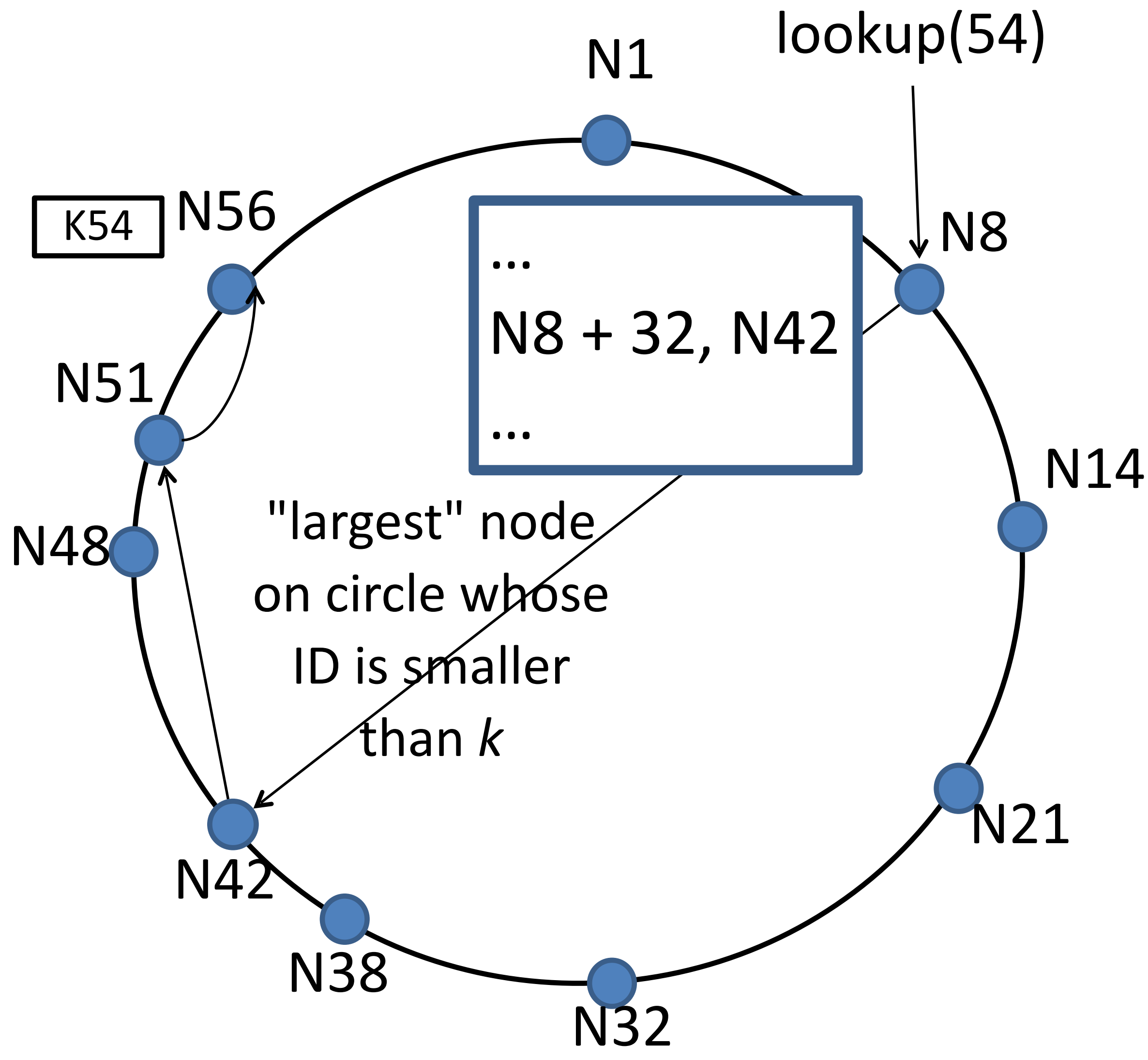
i-th entry is address of **successor(($n + 2^{i-1}$) mod 2^m)**

Every key k is stored at its successor – lookup a key via successor(*k*)

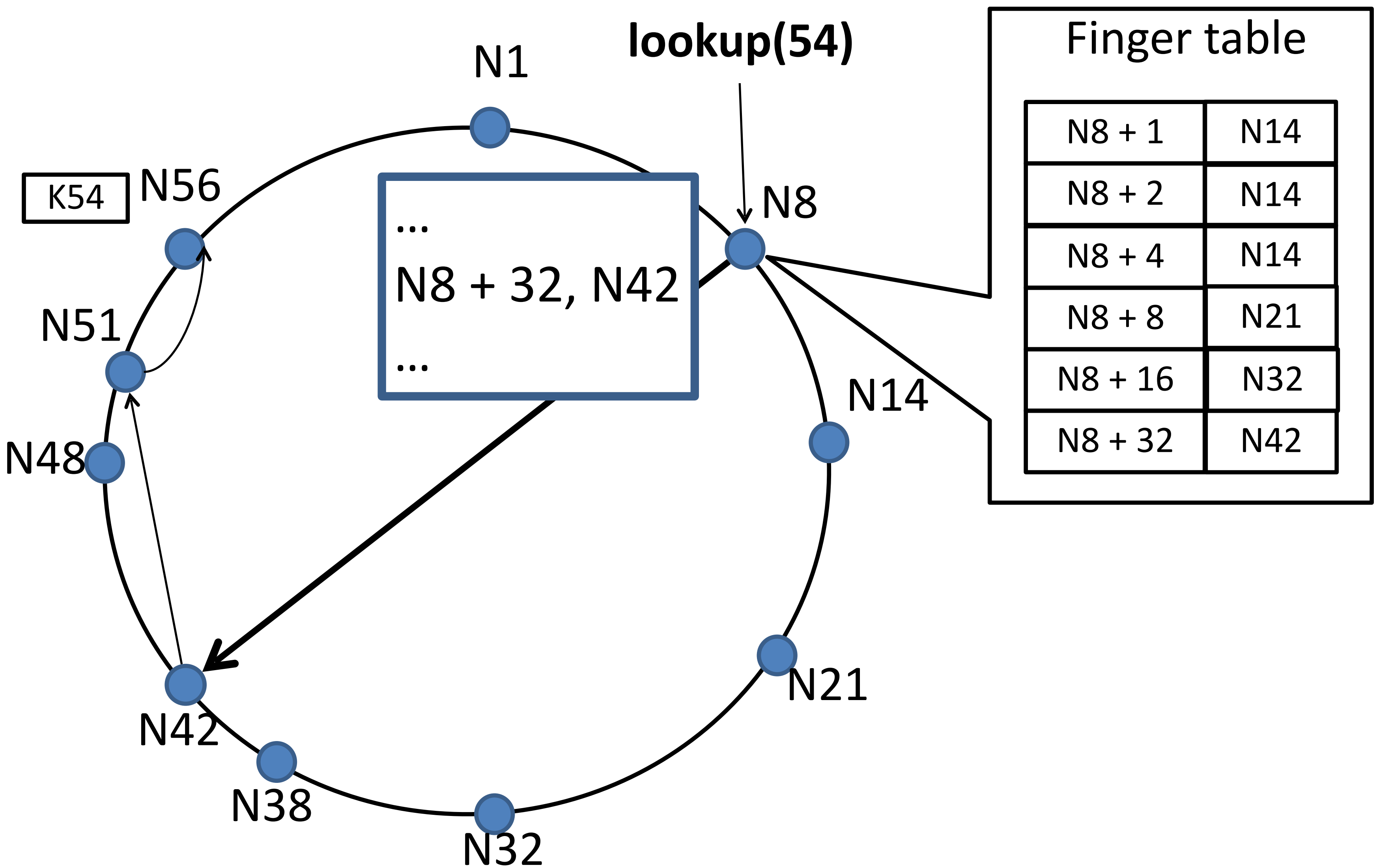
successor(…) is the node on the circle associated with the input argument – whether it is a *key* or a *node identifier*

Chord Key Location

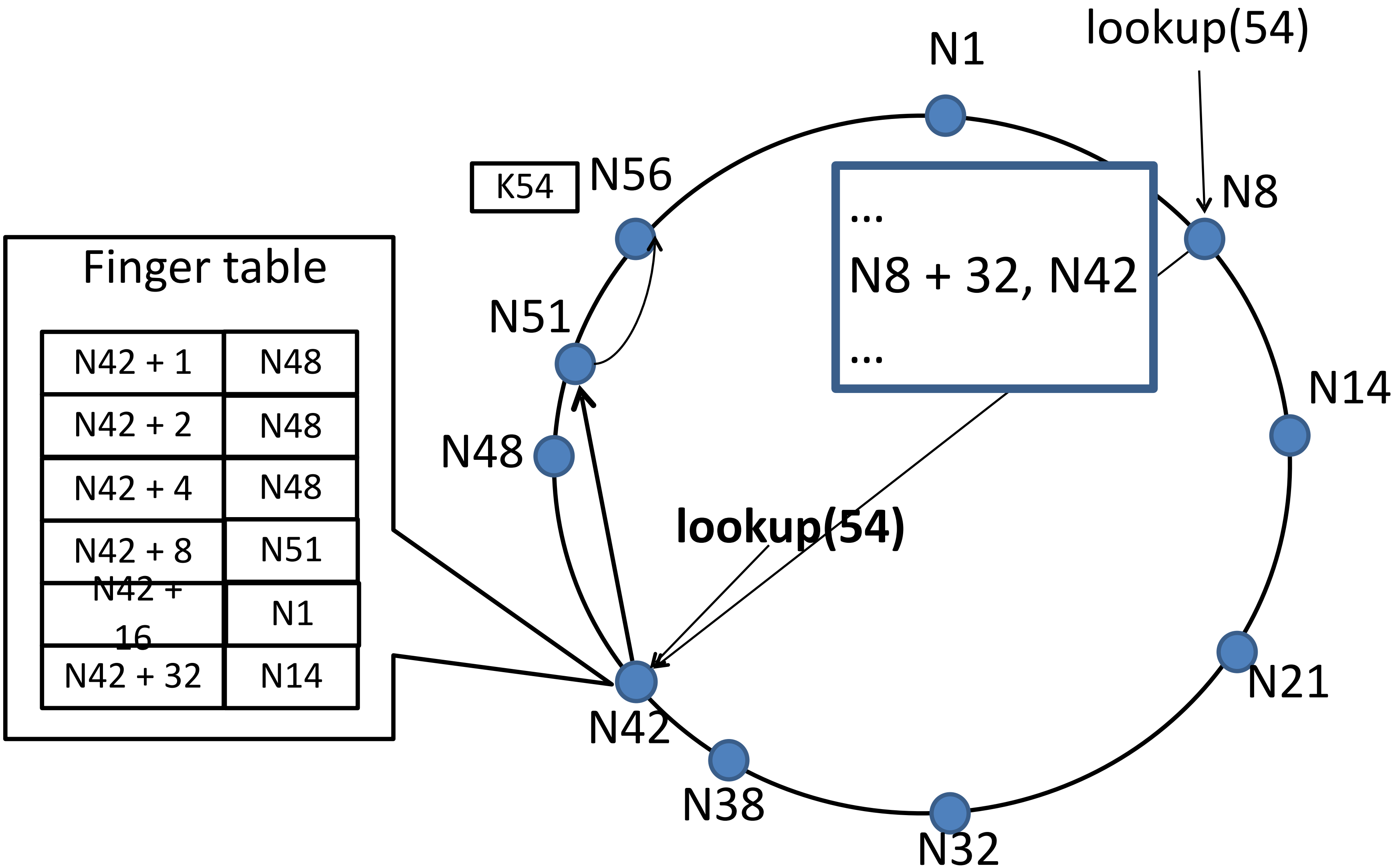
- Lookup in finger table the farthest node that precedes key – closest successor of key in FT
- Query homes in on target in O(log *n*) hops
- Each hop at least halves distance to destination



Lookup Example



Lookup Example (cont.)



Lookup Latencies

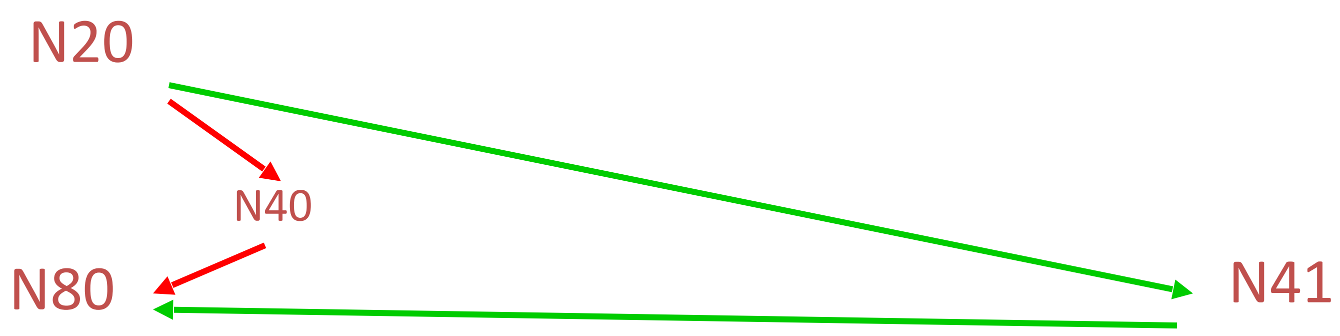
- While O(*log n*) is better than O(*n*), it can still take considerable amount of time to find the target

- For example, log (*1,000,000*) hops which may be distributed anywhere

- Results in potentially high response latencies

Network locality

- Nodes close on ring can be far away in network



* Figure from http://project-iris.net/dht-toronto-03.ppt

Peer-to-Peer Systems

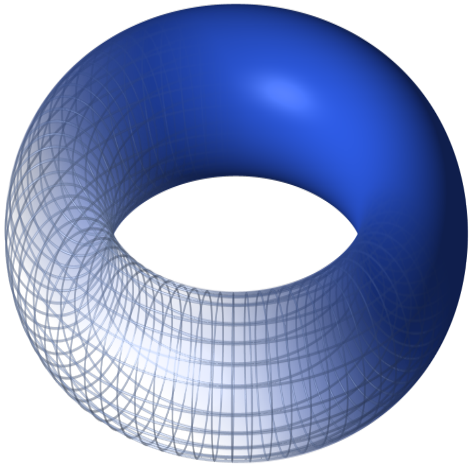
CAN Pastry DHT

CAN: Content Addressable Network

- Design is based on virtual multi-dimensional Cartesian coordinate space to organize overlay
- Nodes are mapped into space (co-ordinates at edges wrap around)
- Address space is independent of physical location and physical connectivity of nodes
- Points in the space are identified with coordinates
- General model is an *n*-dimensional torus that uses dimensions for routing

CAN: Content Addressable Network

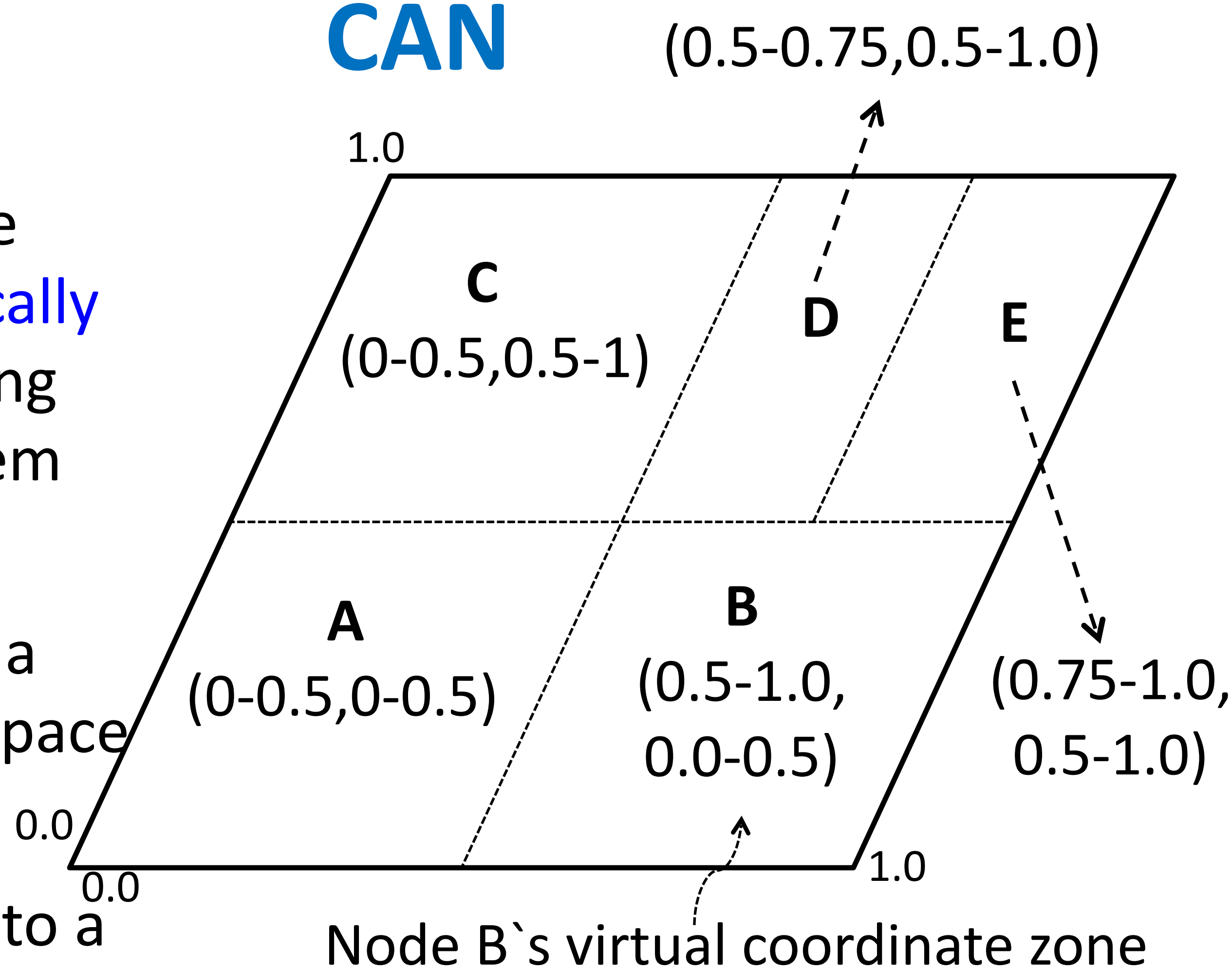
- Design is based on virtual multi-dimensional Cartesian coordinate space to organize overlay
- Nodes are mapped into space (co-ordinates at edges wrap around)
- Address space is independent of physical location and physical connectivity of nodes
- Points in the space are identified with coordinates
- General model is an *n*-dimensional torus that uses dimensions for routing



Example 2-d space with 5 nodes

CAN

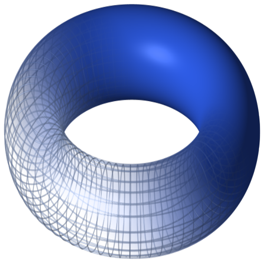
- Entire coordinate space is dynamically partitioned among all nodes in system
- Each node owns a distinct zone in space
- Each key hashes to a point in space

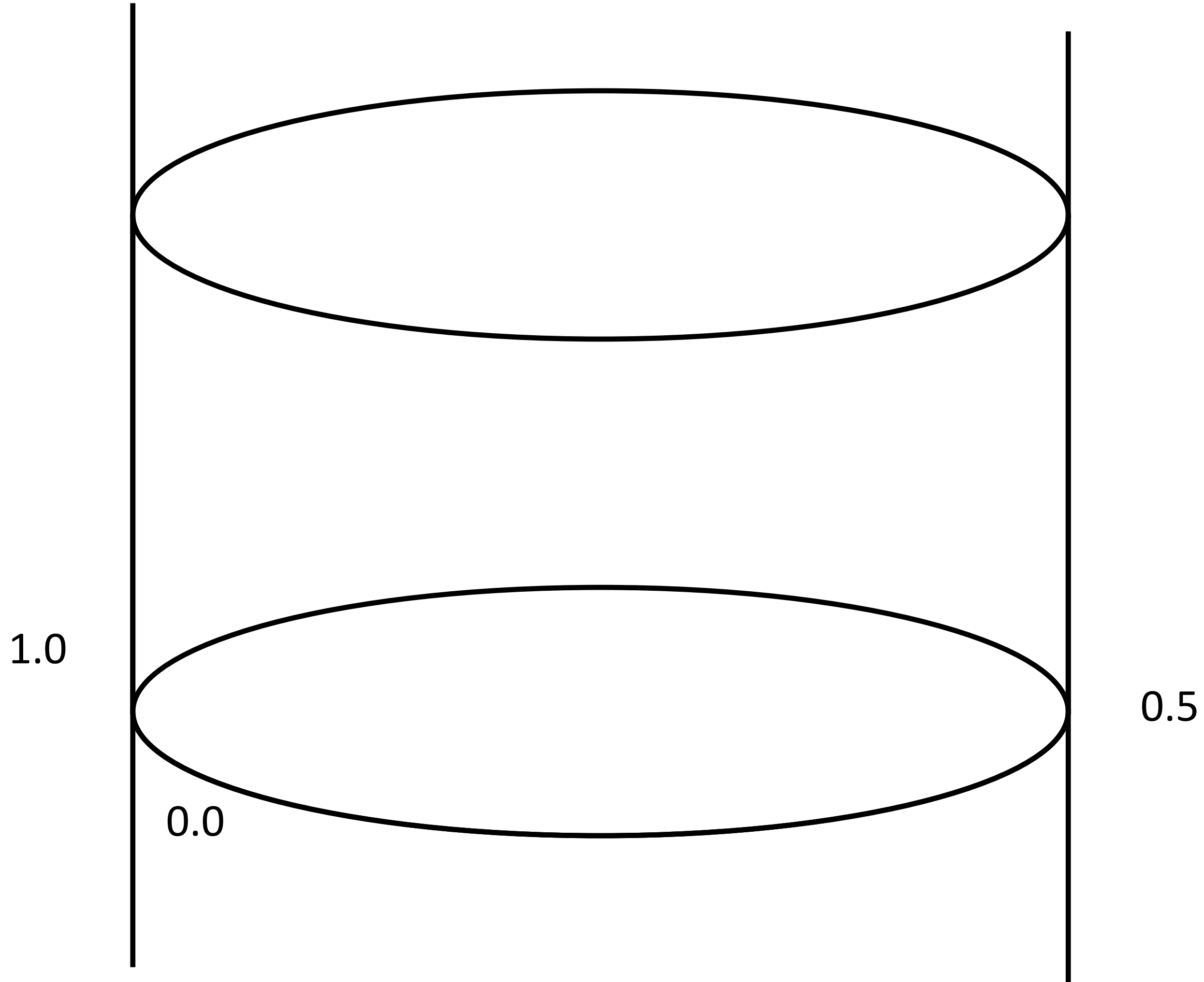


* All CAN figures from "A Scalable Content-Addressable Network", S. Ratnasamy et al., In Proceedings of ACM SIGCOMM 2001.

Coordinates Wrap Around

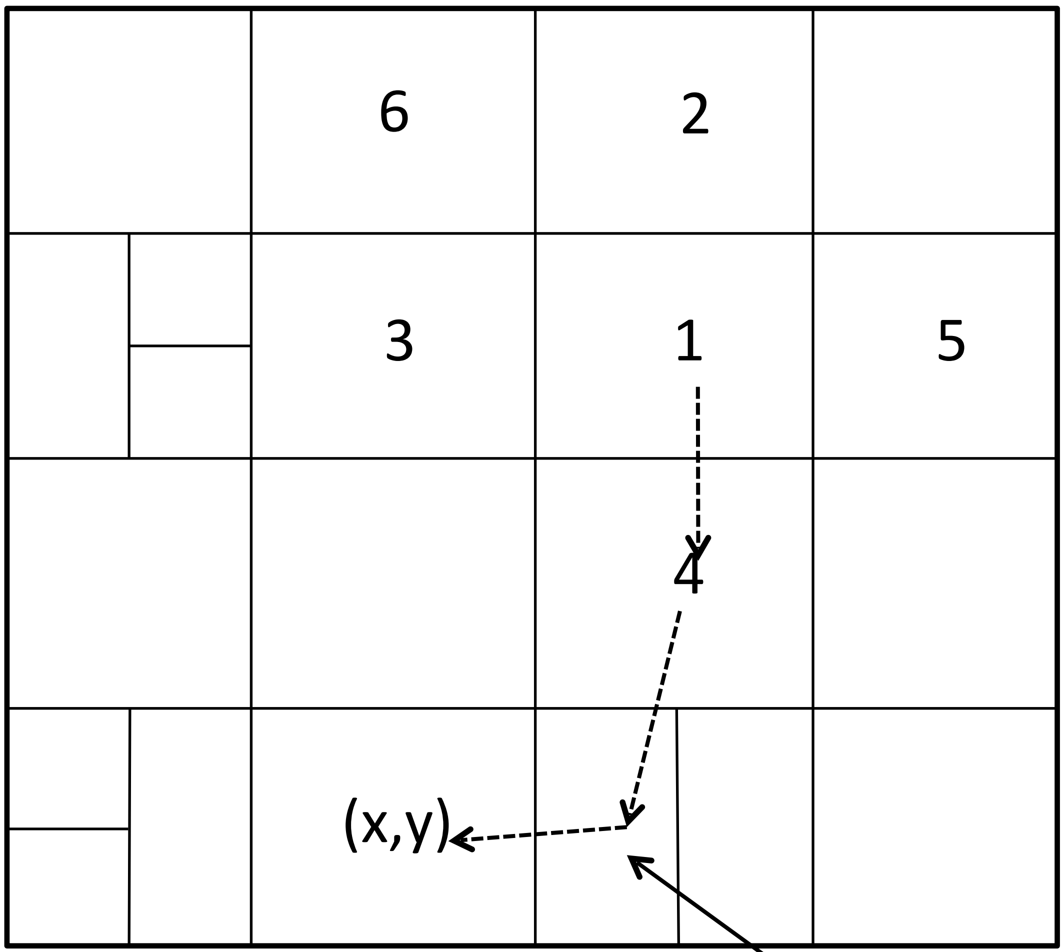
In all dimensions (only x-axis shown)





CAN Routing

- Put(key, data), get(key)
- Greedily forward message to neighbor closest to destination in Cartesian coordinate space
- Nodes maintain a routing table that holds IP address and zone of its neighbours



Sample routing path from node 1 to point (x,y)

1`s coordinate neighbor set = {2,3,4,5}

CAN Routing

- Many possible routing paths exist between two points in space
- If a neighbour on a path crashes, simply pick the next best available (node) path

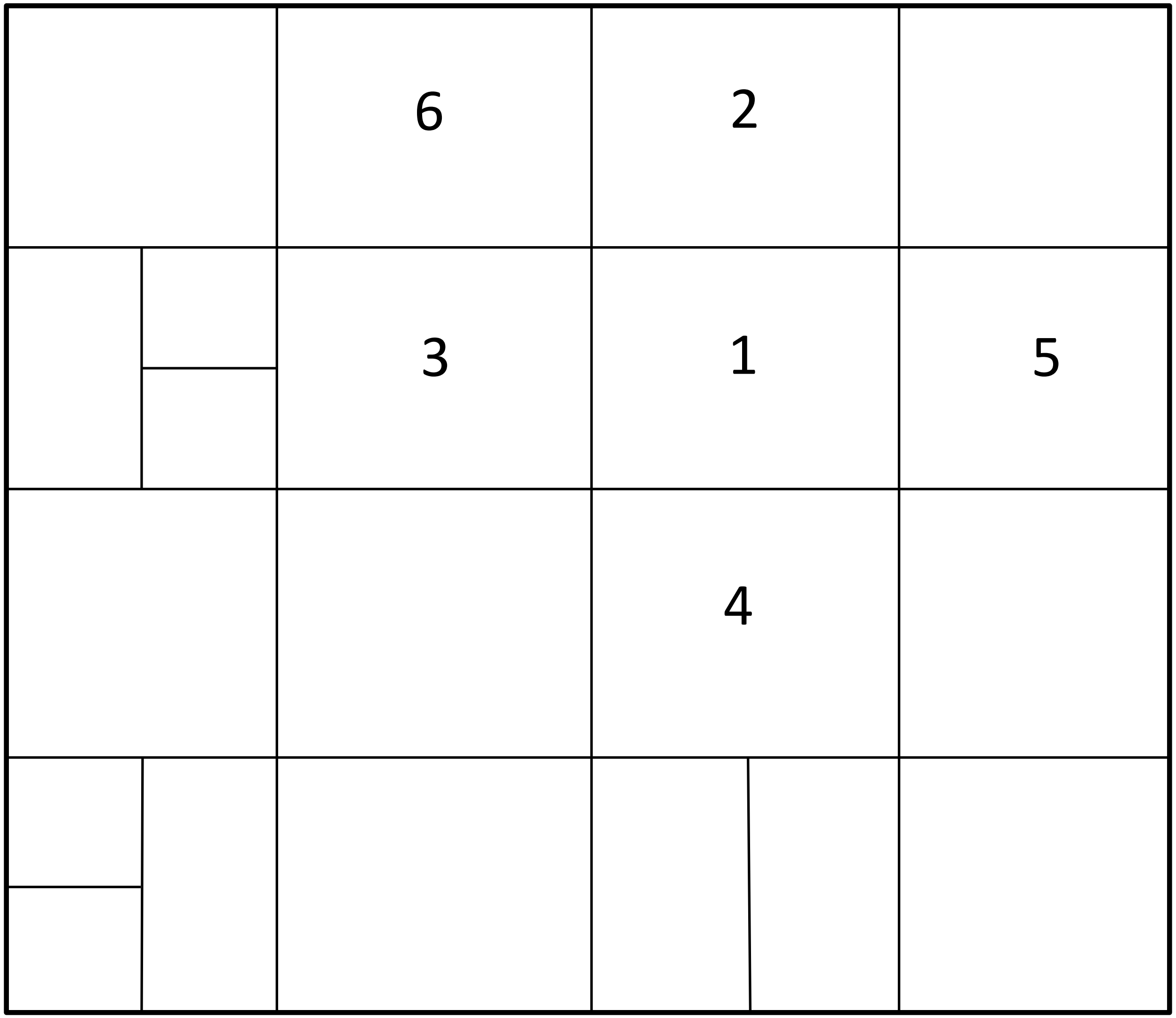
Average Path Length

- ***d*-dimensional space**, partitioned into ***n* equal-sized zones**, **average routing path length** is: $d/4 * n^{1/d}$
- Each node maintains 2*d* neighbours
- Grow number of nodes, without affecting per node state
- Grow number of nodes, increases path length by $O(n^{1/d})$
- 2-dimensional space: $1/2 * n^{1/2}$ (average routing path)
- 3-dimensional space: $3/4 * n^{1/3}$ (average routing path)

Node Joining a CAN

- Find a node already in overlay network
- Identify a zone that can be split
  - Pick random point
  - Route join request to node managing the point’s zone
  - Initiate split of zone at that node
- Update routing tables of nodes neighbouring newly split zone
- If refused, try with a new random point

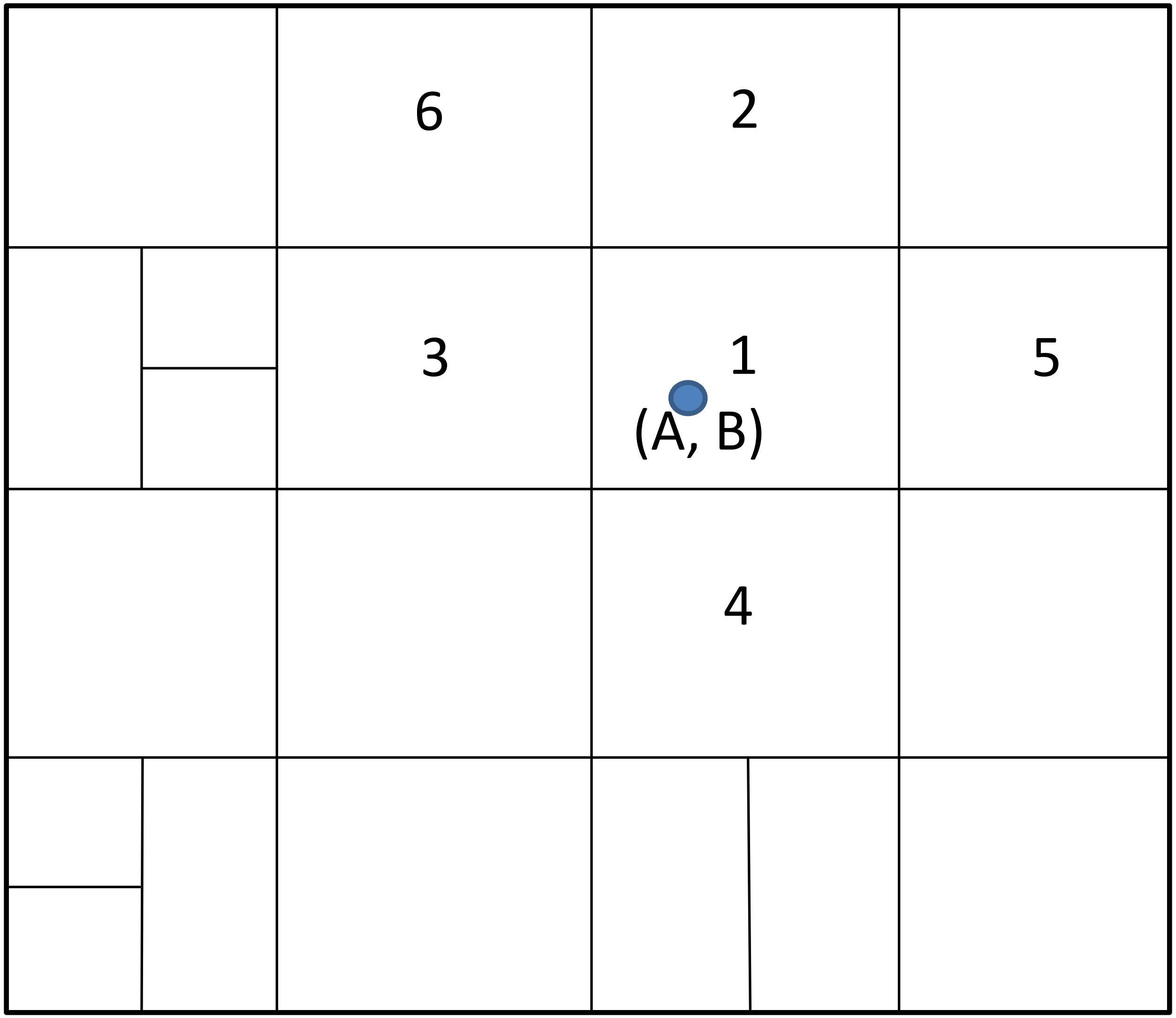
Zone Splitting Upon Node Joining



1`s coordinate neighbor set = {2,3,4,5}

Join request Node 7

Zone Splitting Upon Node Joining

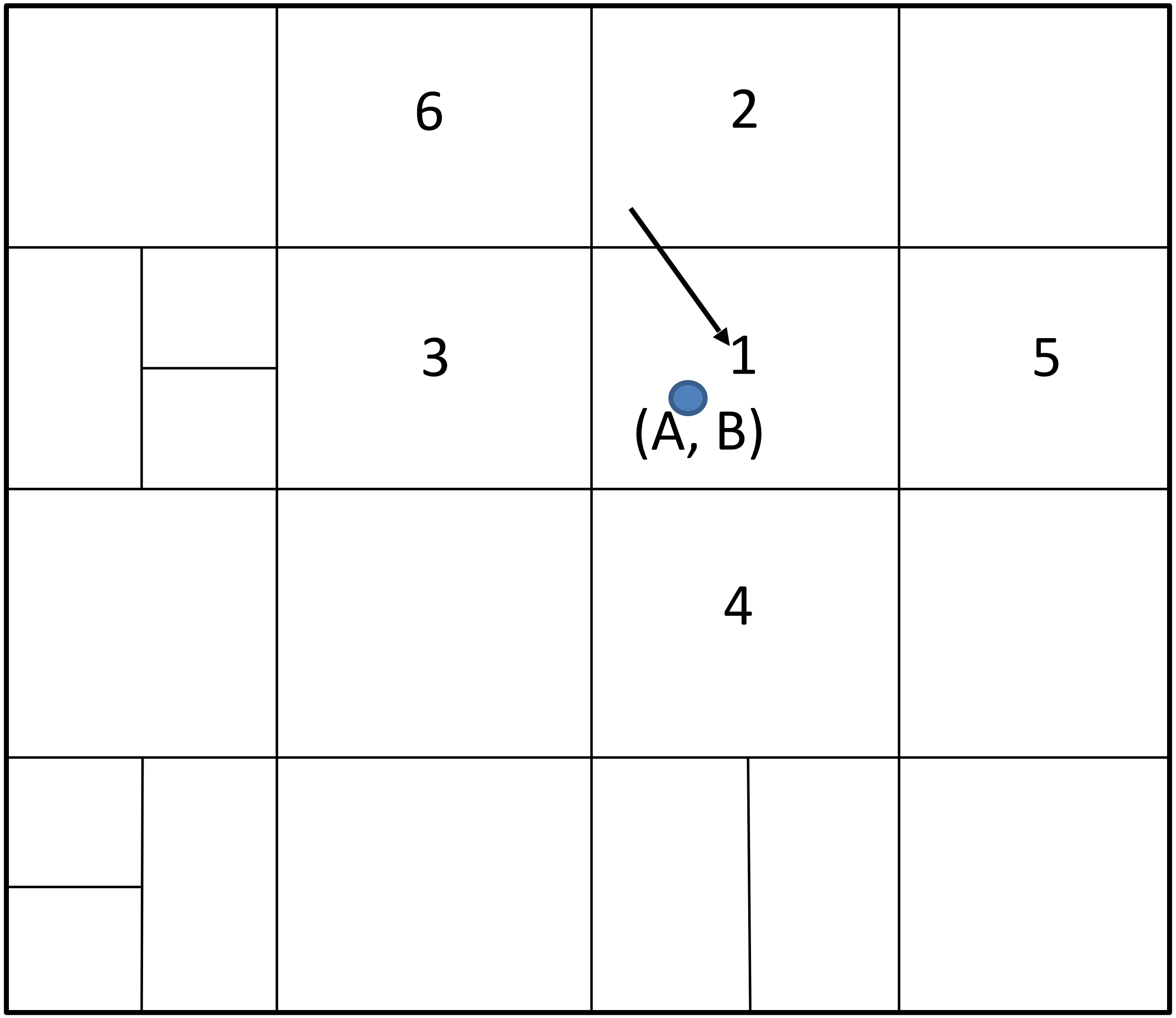


1`s coordinate neighbor set = {2,3,4,5}

Join request Node 7

- Pick random point (A, B)

Zone Splitting Upon Node Joining

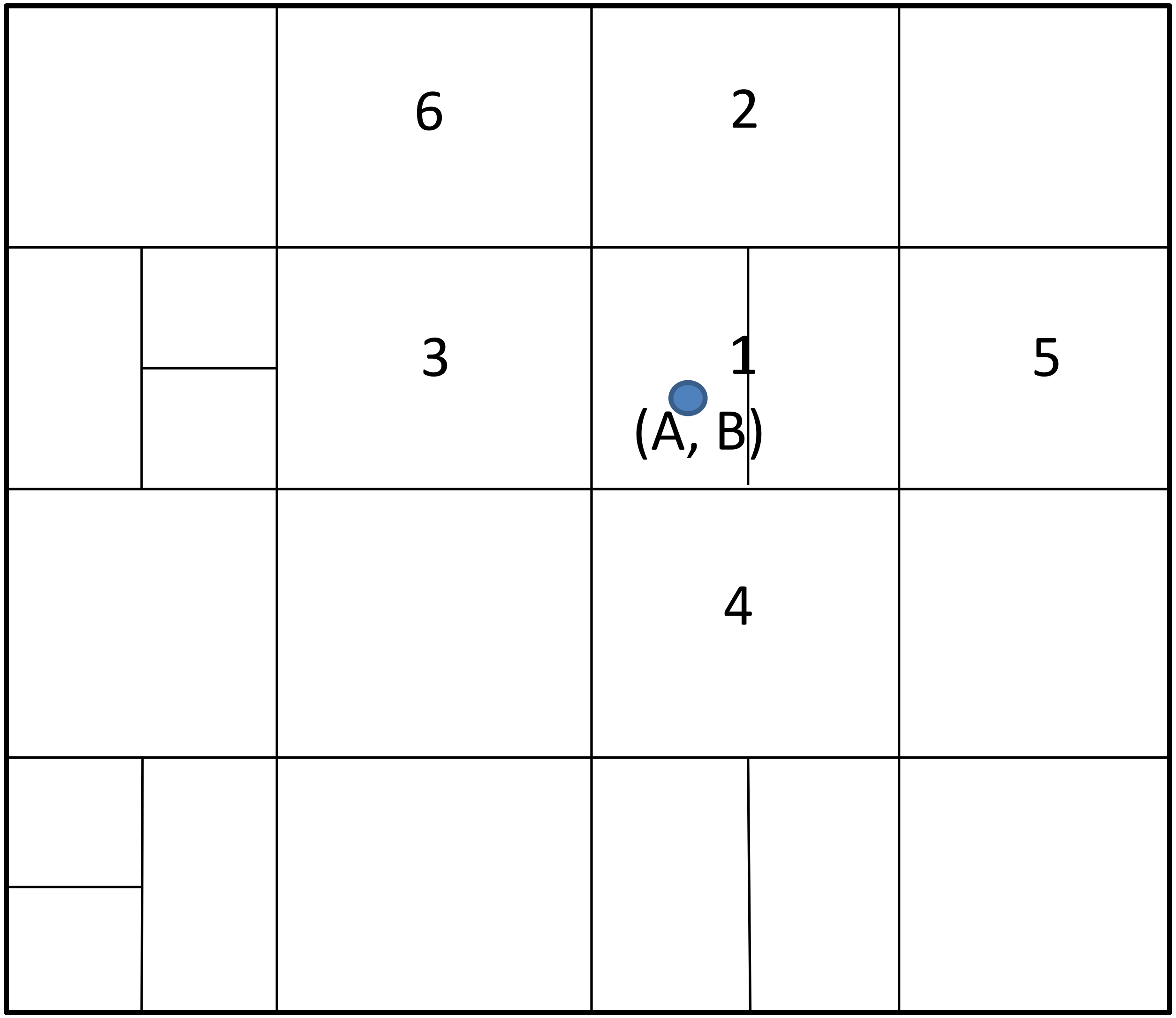


1`s coordinate neighbor set = {2,3,4,5}

Join request Node 7

- Pick random point (A, B)
- Route join request of 7 to 1

Zone Splitting Upon Node Joining

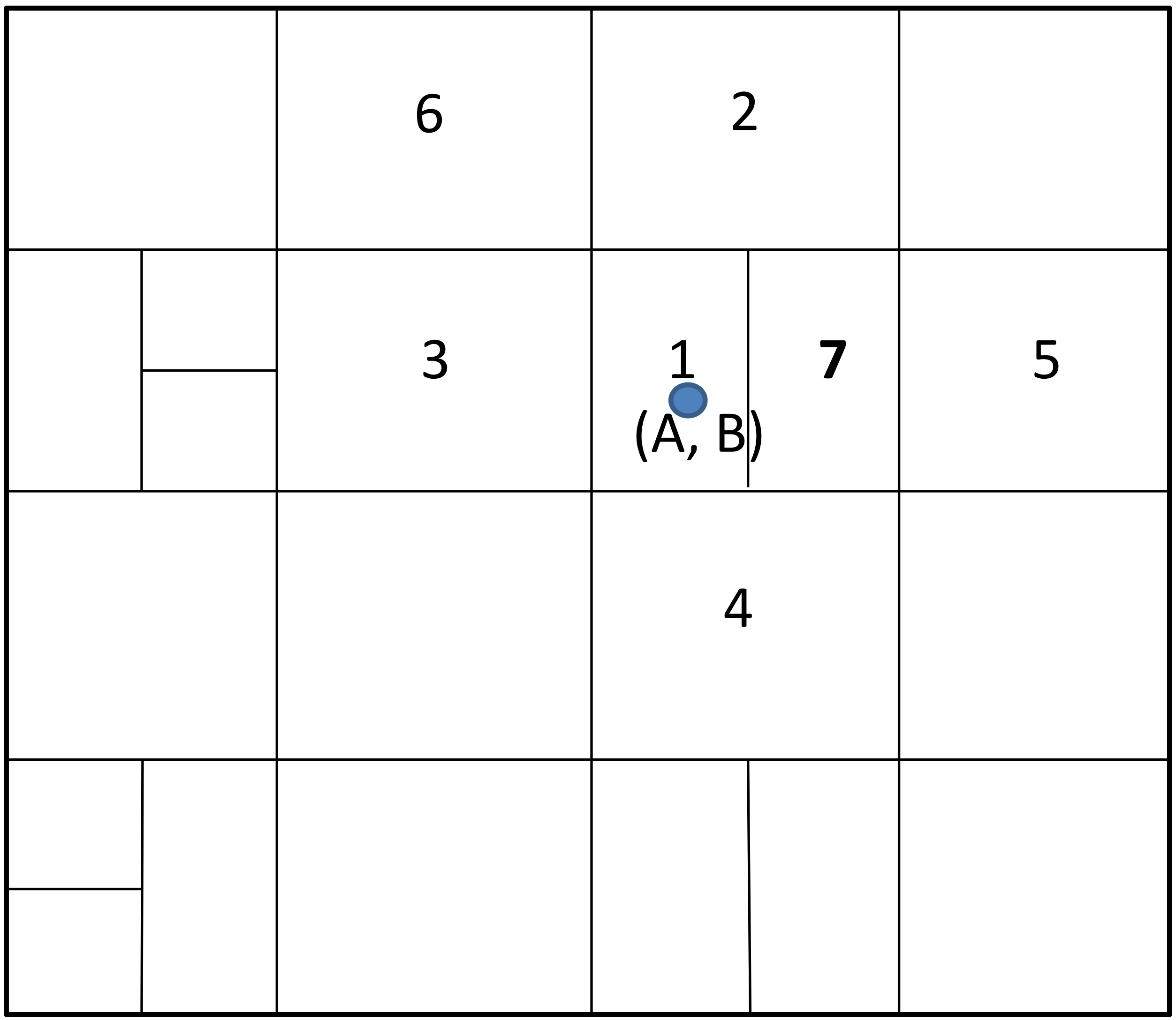


1`s coordinate neighbor set = {2,3,4,5}

Join request Node 7

- Pick random point (A, B)
- Route join request of 7 to 1
- Initiate zone split

Zone Splitting Upon Node Joining

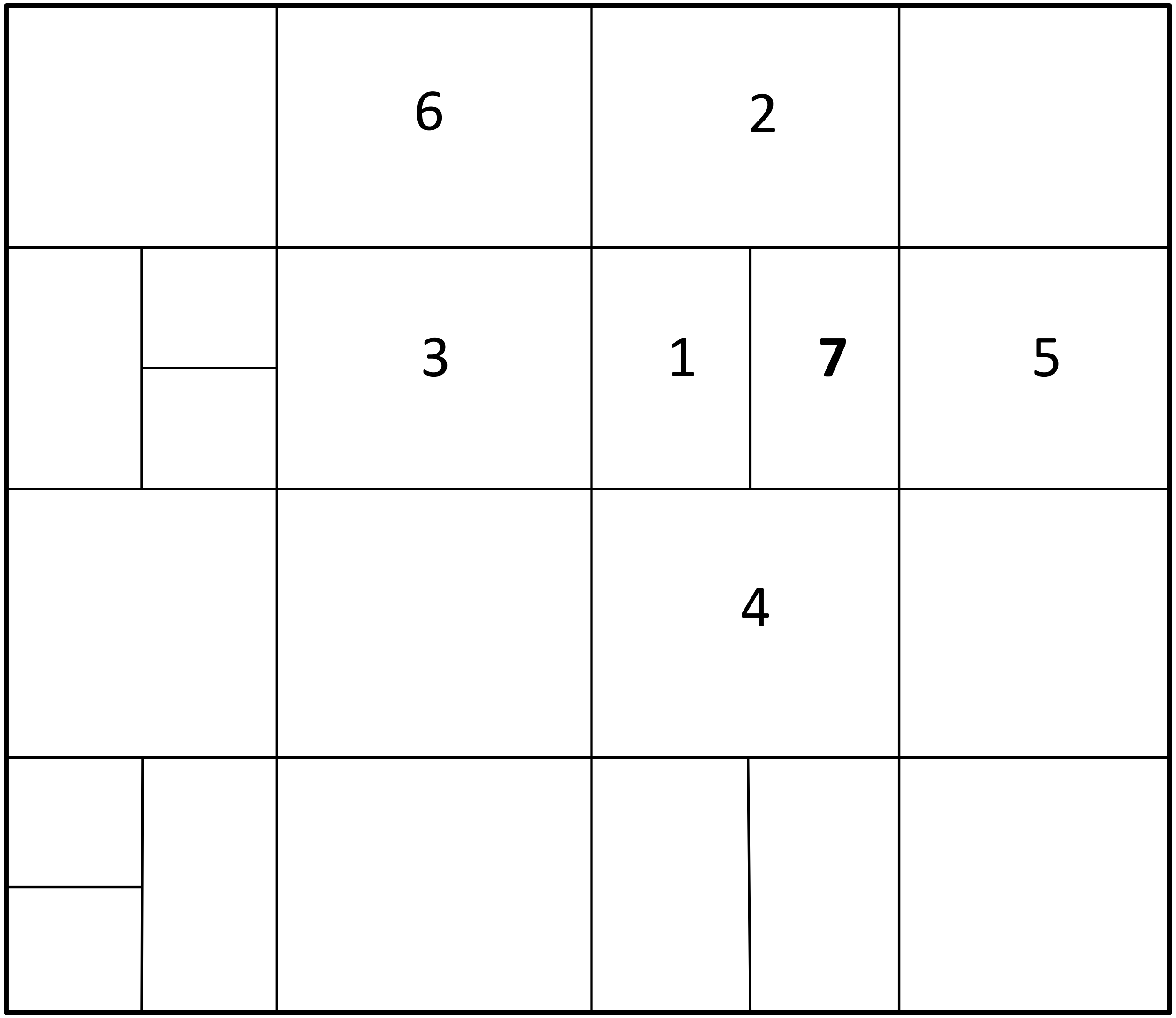


1`s coordinate neighbor set = {2,3,4,5}

Join request Node 7

- Pick random point (A, B)
- Route join request of 7 to 1
- Initiate zone split

Zone Splitting Upon Node Joining



1`s coordinate neighbor set = {2,3,4,7}

7`s coordinate neighbor set = {1,2,4,5}

Update routing tables of nodes, transfer state, i.e., (k, v)-pairs (not shown)

Node Join Properties

- Only O(*d*) nodes are effected when a node joins/leaves CAN (a node has 2d neighbours)
- Independent of *n*, number of nodes in CAN

Pastry (2001)

- Ring-based partitioning like Chord
- Each peer discovers and exchanges state information: List of leaf nodes, neighborhood list, routing table
- **Leaf node list** are *L/2* closest peers by Node ID in each direction around the circle
  - Lookups first search the leaf node list
- **Neighborhood list** are *M* closest peers in terms of routing metric (e.g. ping delay)
  - Good candidates for routing table

Routing Table

- 6 digits, base 4: table of 6x4 entries
- Row *i* contains nodes which share *i-1-th* long prefix
- Populate cells with neighbors if possible
- Column indicates *i-th* digit
- Lookup in RT finds a node with a longer prefix

Node 103220	0	1	2	3
1	031120	**1**03220	201303	312201
2	**10**3220	110003	120132	132012
3	100221	101203	102303	**103**220
4		103112	**1032**20	103302
5		103210	**10322**0	
6	**103220**			

Example: lookup(102332) -> 102303

DHT Routing Summary

- Chord
  - Finger table routing
    - Each hop at least halves distance (in identifier circle) to destination
- Pastry
  - Proximity-based Routing
- CAN
  - Neighbour routing
    - Forward to neighbor closest (in Cartesian coordinate space) to destination

Conclusions on P2P

- Hugely popular area of research 2000-2010
- Large-scale companies (Amazon & Google et al.) prefer self-managed cloud infrastructures
- P2P principles, techniques and abstractions are used by large-scale systems (e.g., **DHTs**)
- Active applications: **BitTorrent, Bitcoin** *et al.*
- Peer-assisted, hybrid systems were popular: **Skype**, **Spotify**, etc.