Peer-to-peer Systems

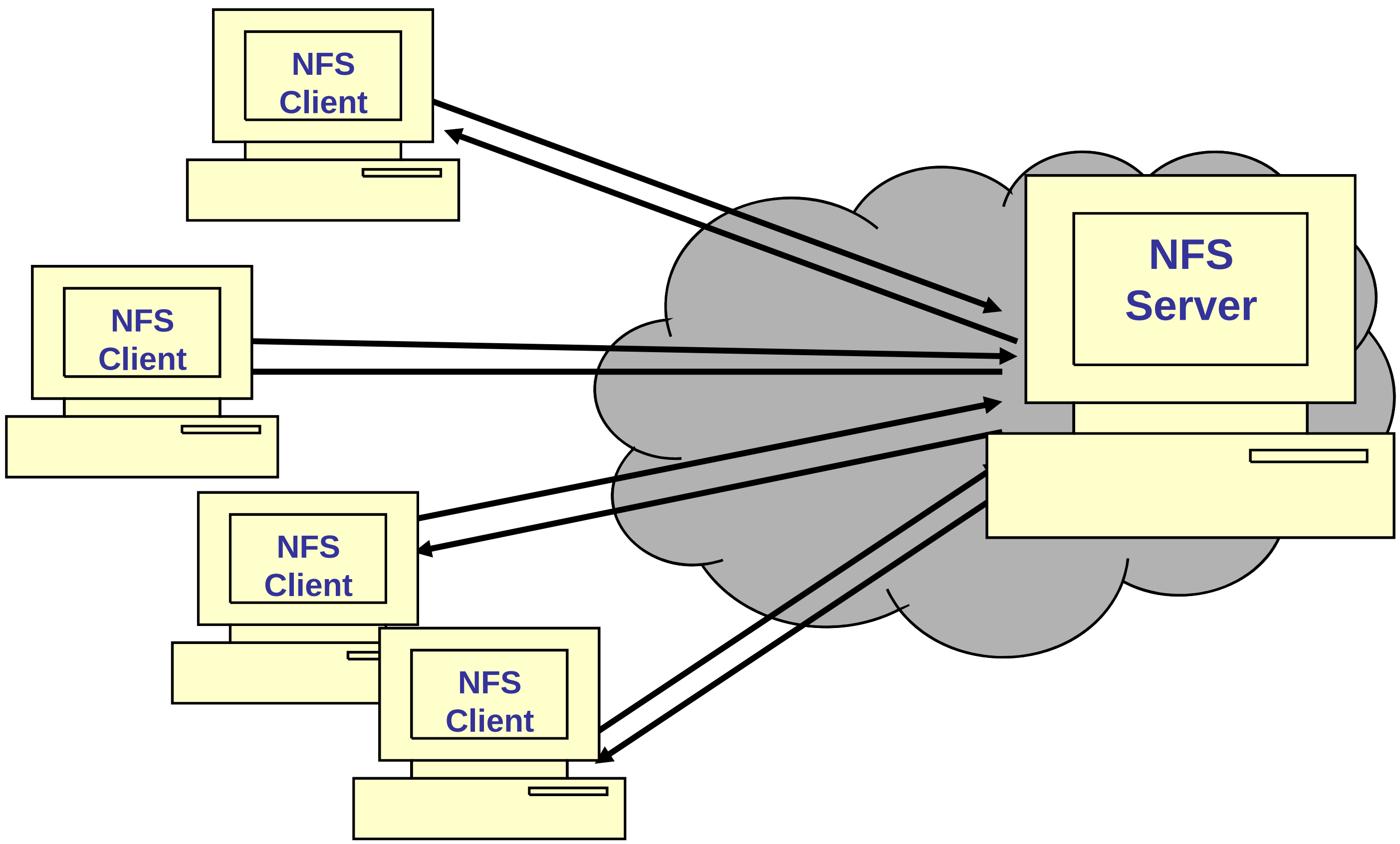
Brian Nielsen

bnielsen@cs.aau.dk

Client-Server

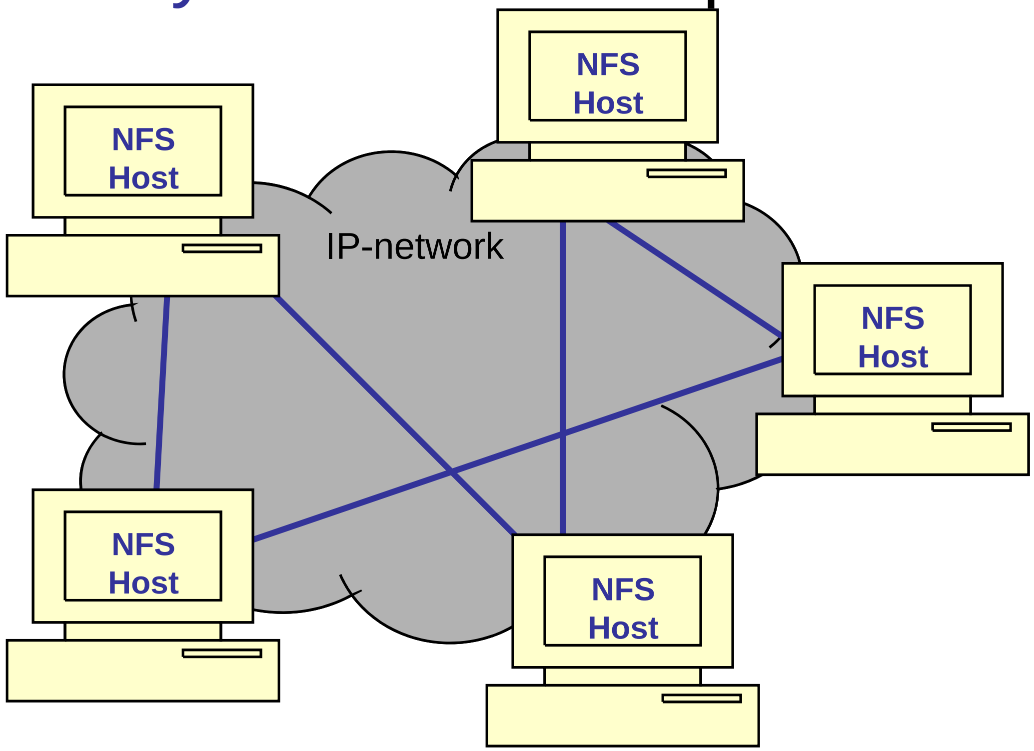
- Centralized
- Functional specialization
- Central administration

- Bottleneck
- Single point of failure



Peer2Peer

- Peer = “Equal Partner”
- Peers are “equal” computers located at the border of the network
- Logical “overlay network” on top of IP network



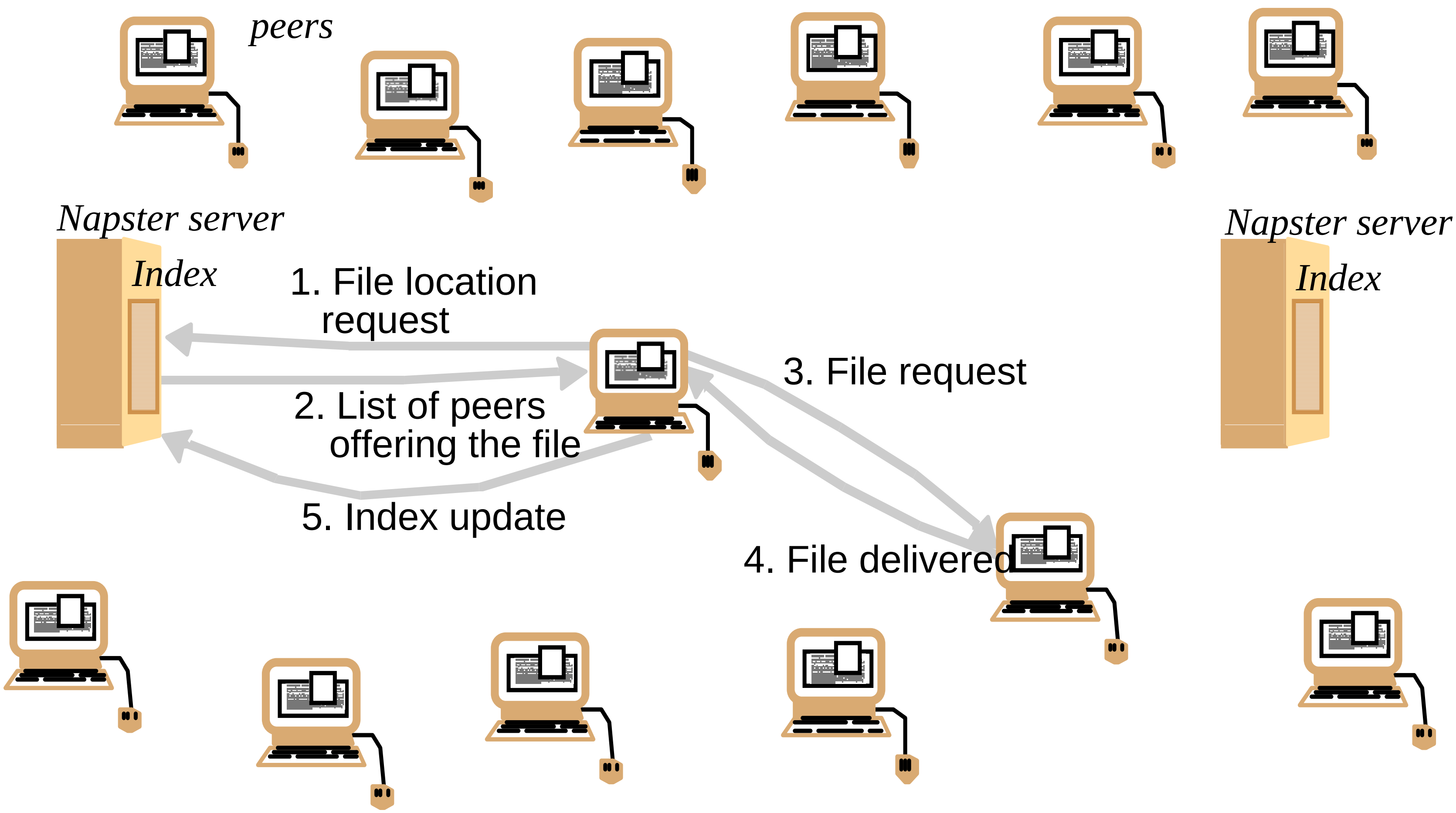
Aim of peer-to-peer systems

- Sharing of data and resources at very large scale
  - No centralized and separately managed servers and infrastructure
  - Share load by using computer resources (memory and CPU) contributed by "End-hosts" located at the edges of the internet"
- Security
- Anonymity

Background

- Pioneers:
  - Napster,
  - Gnutella, FreeNet
- Hot / new research topic:
  - Infrastructure
    - Pastry, Tapestry, Chord, Kademlia,..
  - Application:
    - **File sharing**: CFS, PAST [*SOSP'01*]
    - **Network storage**: FarSite [*Sigmetrics'00*], Oceanstore [*ASPLOS'00*], PAST [*SOSP'01*]
    - **Multicast**: Herald [*HotOS'01*], Bayeux [*NOSDAV'01*], CAN-multicast [*NGC'01*], SCRIBE [*NGC'01*]

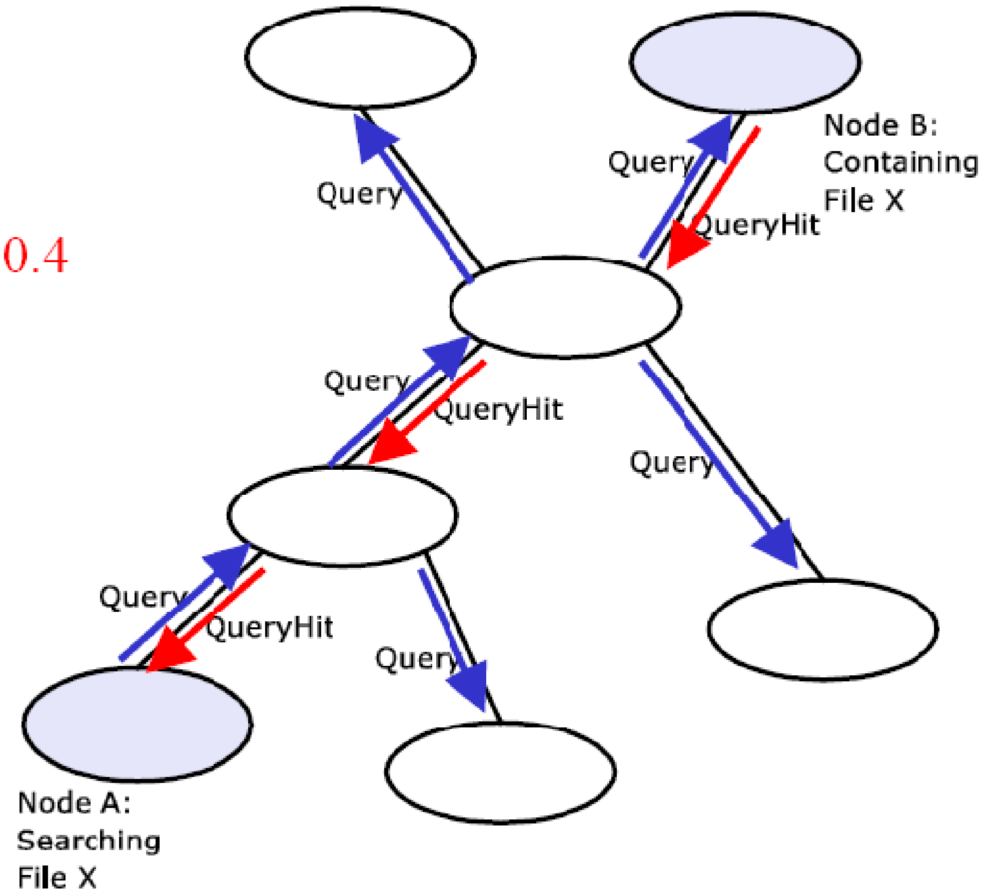
Napster: centralized, replicated index



Gnutella-Flooding

- *Unstructured* Overlay Network: Each node knows a set of other nodes
- Flood overlay network with query: Who has file X?

Gnutella 0.4



Characteristics

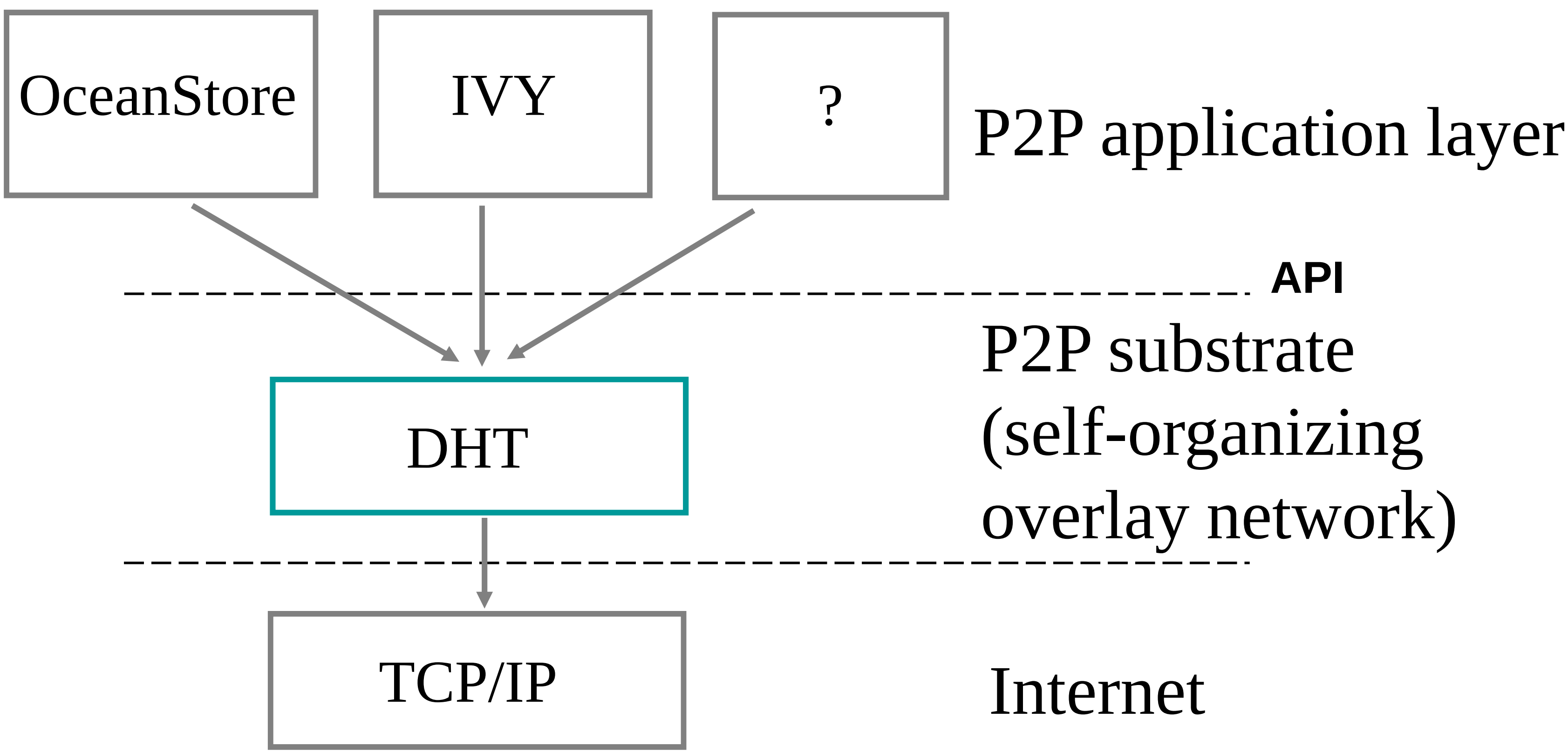
- Distributed
  - Participants distributed across the internet
  - All contributes with resources
- Decentralized control
  - no central decision point
  - no single point of failure
  - dynamic: unpredictable set of participants
- Self-organizing
  - No permanent infrastructure
  - No centralized administration
- Symmetric communication/roles
  - Same functional capabilities

Common issues

- Organize, maintain overlay network
  - node arrivals
  - node failures
- Resource allocation/load balancing
- Efficient Resource localization
- Locality (network proximity)

Idea: generic P2P middleware (aka “substrate”)

Architecture



Basic interface for distributed hash table (DHT)

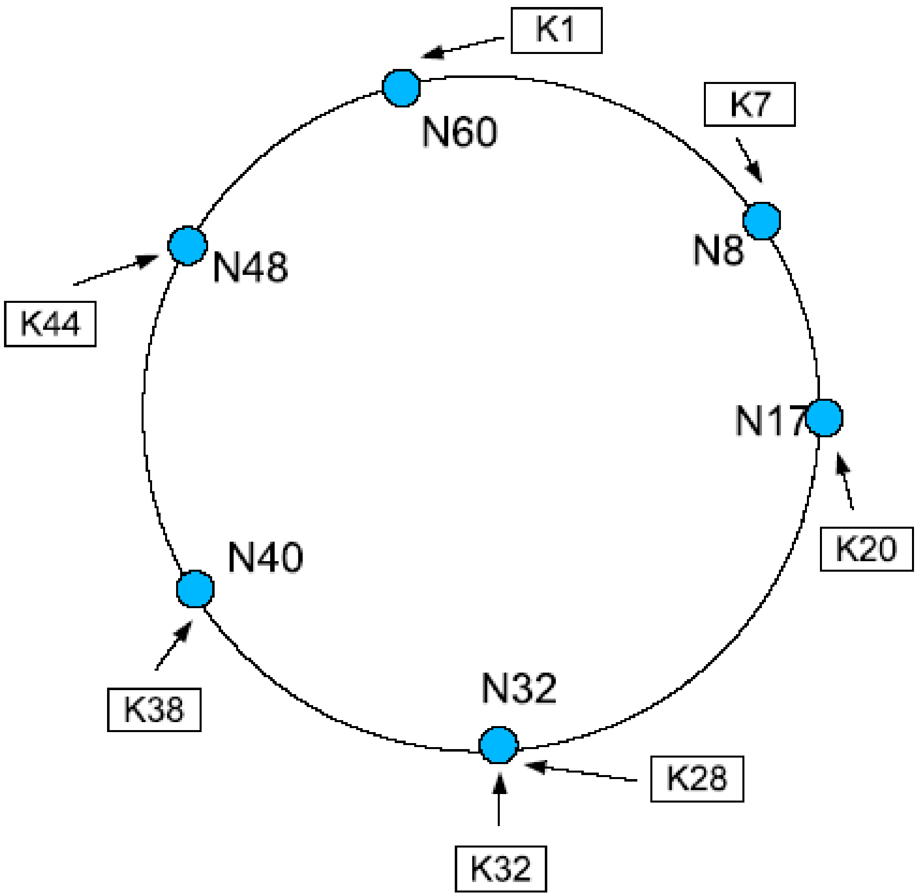
- Peer-to-peer object location and routing substrate
- Distributed Hash Table: maps object key to a live node

  put(GUID, data)
  The data is stored in replicas at all nodes responsible for the object identified by GUID.
  remove(GUID)
  Deletes all references to GUID and the associated data.
  value = get(GUID)
  The data associated with GUID is retrieved from one of the nodes responsible it.

- Pastry (developed at Microsoft Research Cambridge/Rice) is an example of such an infrastructure.

Example DHT

- Nodes are given a GUID **(Globally Unique ID)**
- Data values are identified by a "key" GUID
- Store (key, value) pairs
- Key computed as hash of value
  - Hash-key("Die Hard.mpg") = 28
- Store data at node whose id is numerically closest to key
- Each node receives at most K/N keys
- Keys are >= 128 bits
  - Hash-key (http://www. research.microsoft. com/~antr) = 4ff367a14b374e3dd99f (hex)



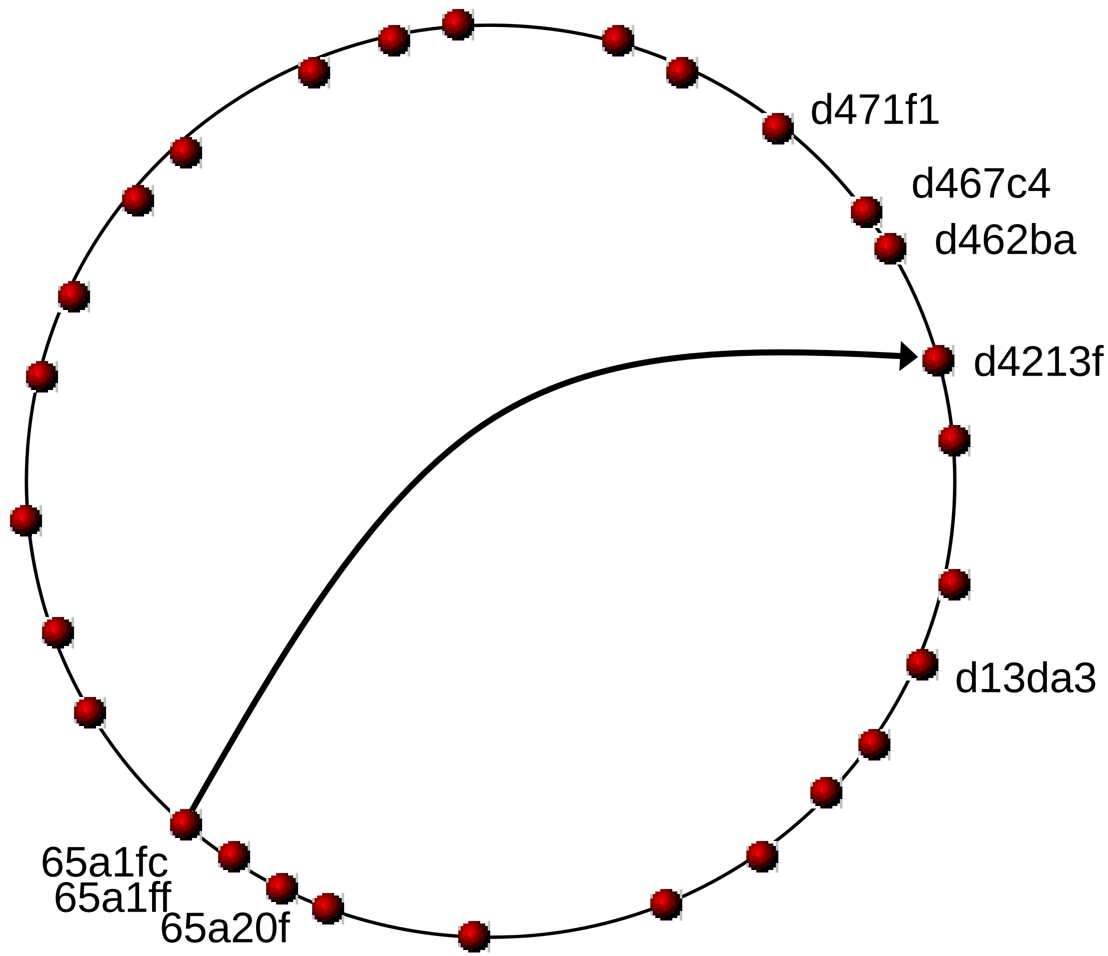
Secure Hash Function

- Aka. Secure digest,
- Given data item M, **h=H(M)**
- Properties
  1. Given M, h is easy to compute
  2. Given h, M is hard to compute
  3. Given M, it is hard to find M´ s.t H(M)=H(M')
- Uniqueness: For two items M, M' it is unlikely that H(M)=H(M')
- Tamperproof: Contents of M cannot be modified and produce same hash-key
- E.g MD5, SHA-1

Exhaustive Routing

Exhaustive Routing Table for node 65a1fc

Node ID	IP
65a1fc	self(127.0.0.1)
65a1ff	123.4.4.9
65a20f	47.122.99.7
…	…
d13da3	123.4.4.8
…	…
d4213f	10.10.34.56
…	…

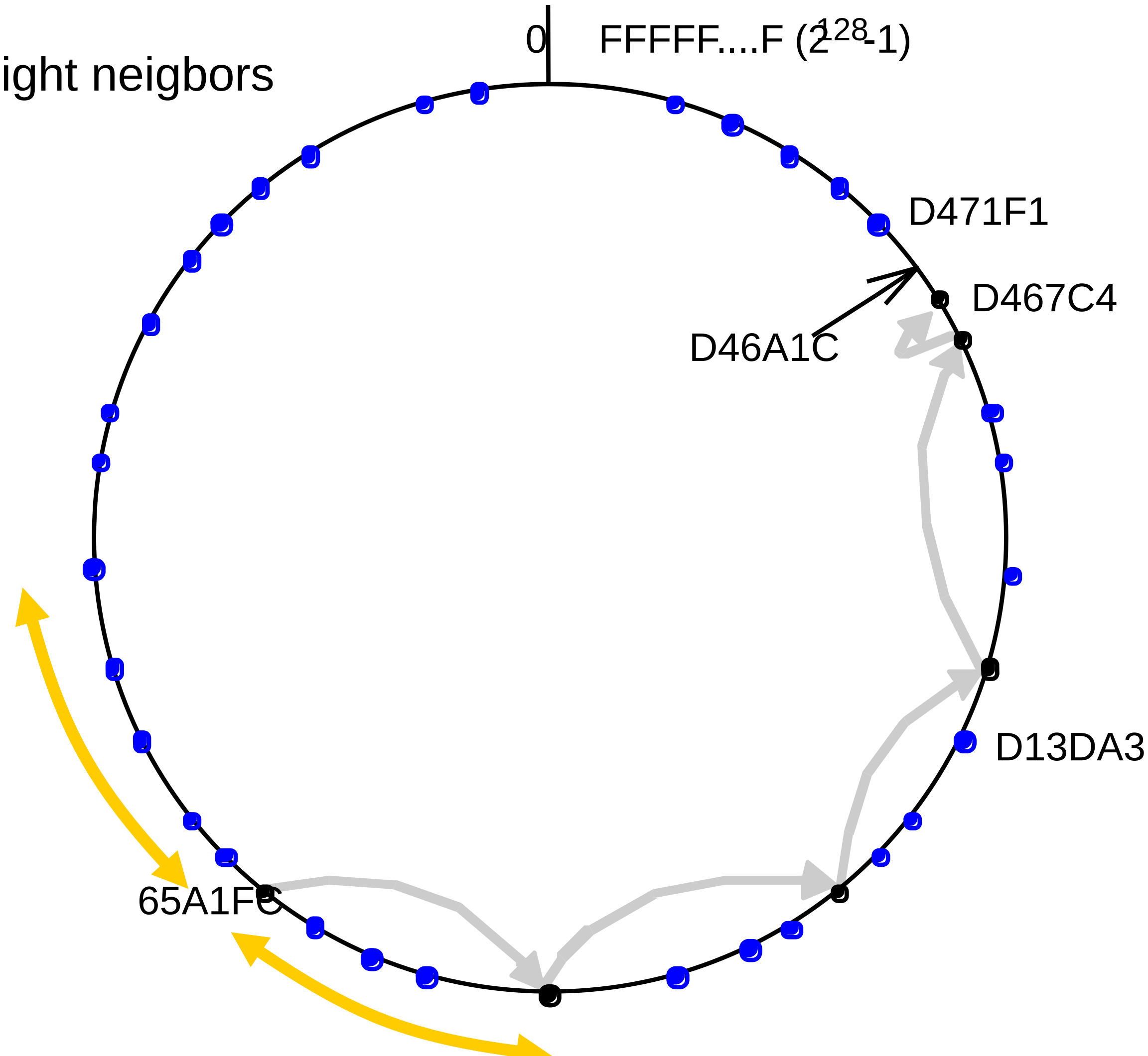


2 Million Nodes=2Million entries =2M*(128+32)/8 bytes= 40MB+data!!!!!!!!!

Impossible to collect and maintain

Circular routing

- Each node knows the *l* left and *l* right neigbors
- Leaf-set (size 2*l*):
- Route to node closest to target



Circular Routing Table for node 65A1FC

Node ID	IP
Left *l*	47.122.99.7
Left *l*-1	123.4.4.9
…	
Left 1	132.32.32.40
self	self(127.0.0.1)
Right1	123.4.4.8
…	…
Right *l*	10.10.34.56

2 Million nodes, *l* = 4 => 2M/(2*4) = 125000 hops!!!

Pastry

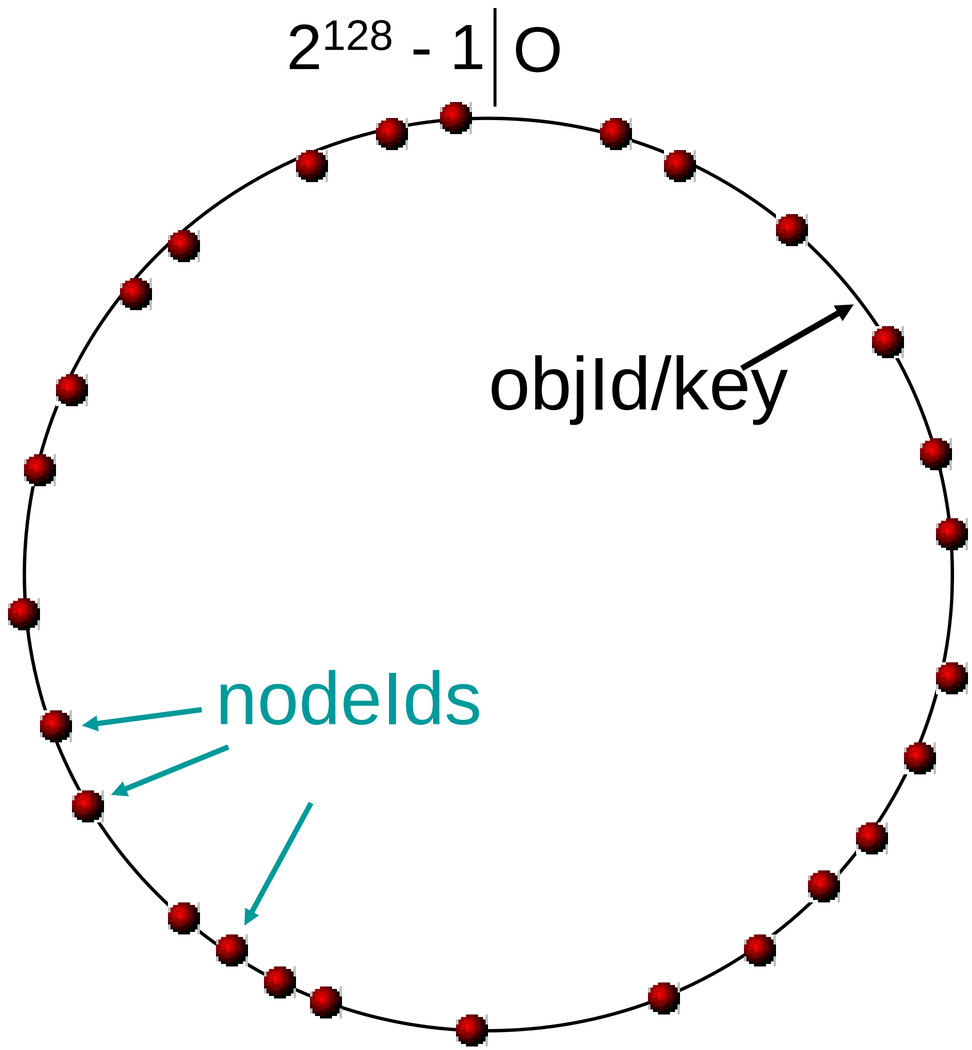
- Pastry (developed at Microsoft Research Cambridge/Rice) is an example of DHT

Generic p2p location and routing substrate (DHT)

- Self-organizing overlay network (join, departures, locality repair)
- Consistent hashing
- Lookup/insert object in < $log_{2^b} N$ routing steps (expected)
- *O(log N)* per-node state
- Network locality heuristics

"Scalable, fault resilient, self-organizing,
locality aware, secure" *(according to authors)*

Pastry: Object distribution



Consistent hashing

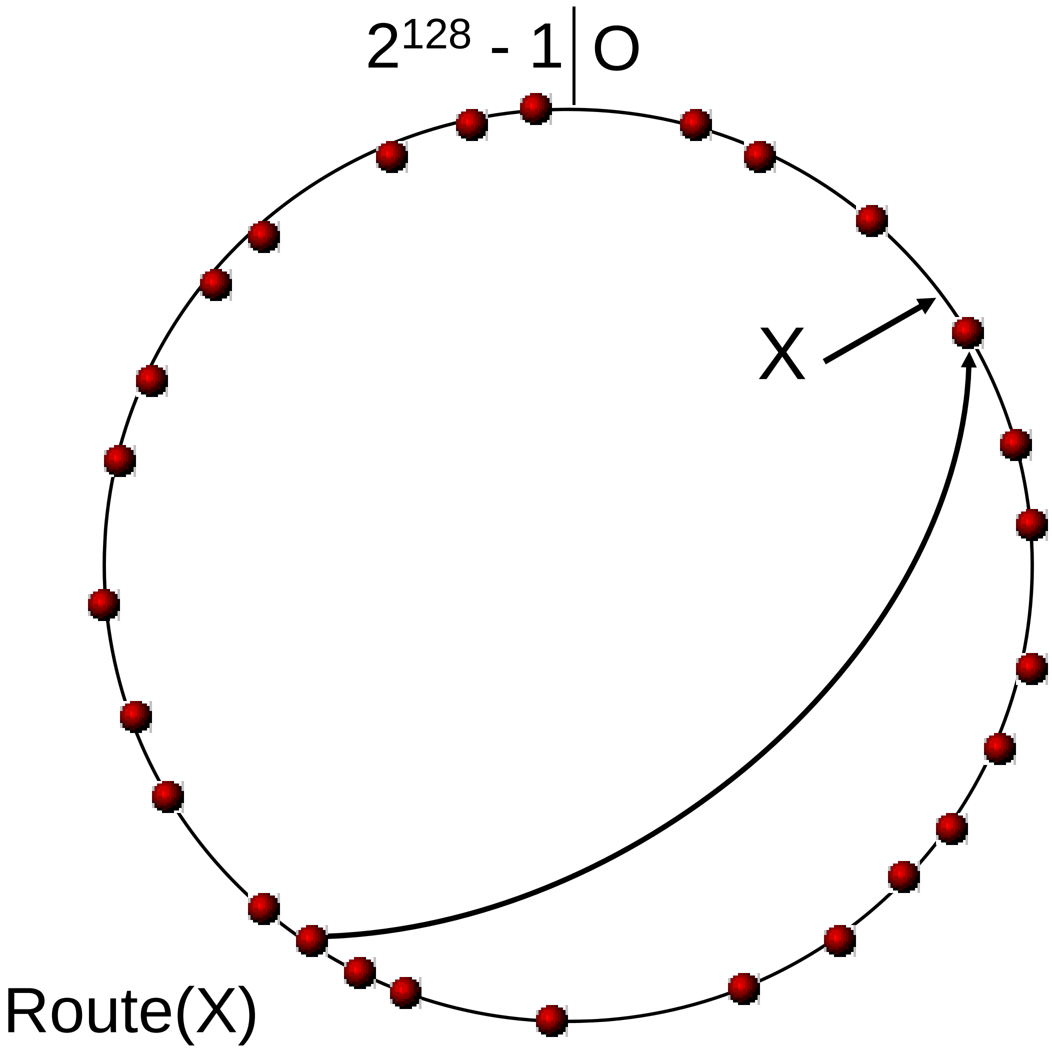
128 bit circular id space

nodeIds (uniform random)

objIds/keys (uniform random)

Invariant: node with numerically closest nodeId maintains object

Pastry: Object insertion/lookup



Msg with key *X* is routed to live node with nodeId closest to X

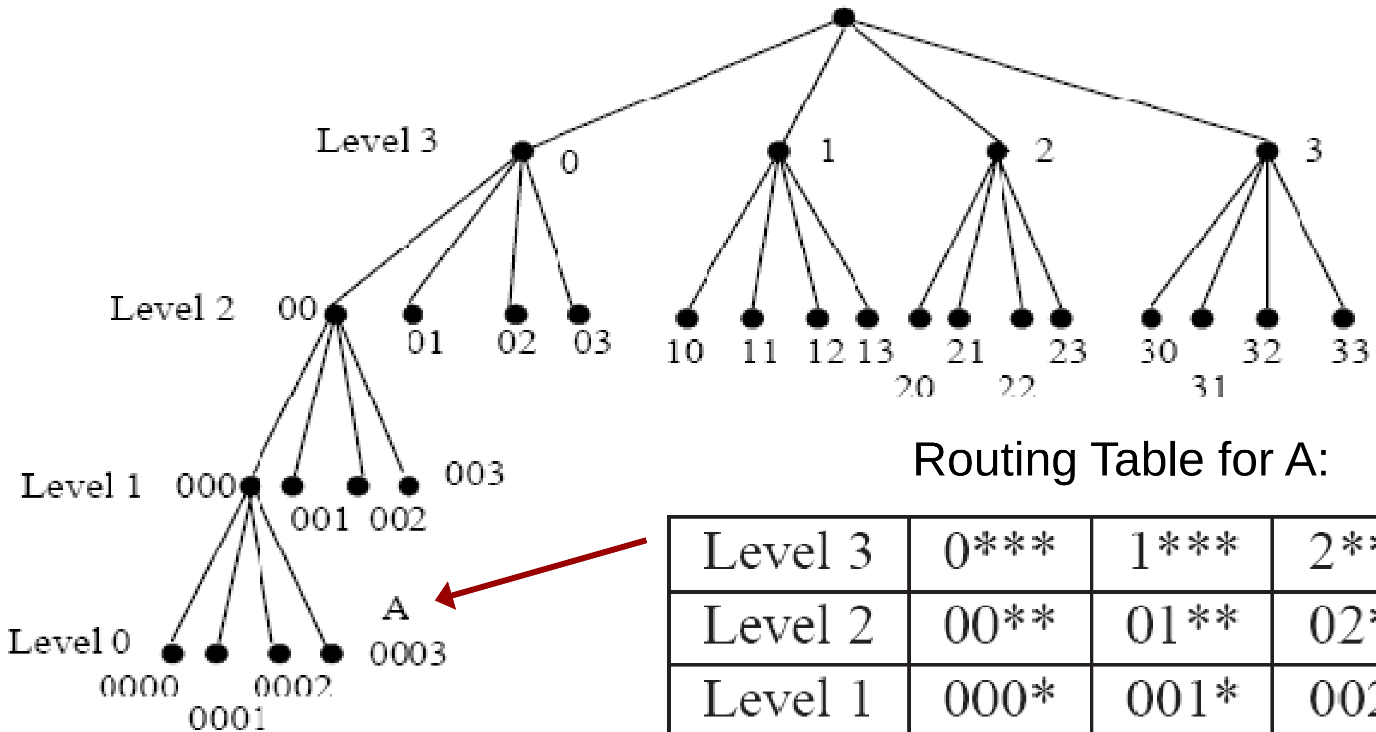
Problem:
complete routing table not feasible

Longest Common Prefix

- Two numerically close IDs are also close nodes in the overlay network
  - D471F1
  - D471F3
  - D47889
  - D99888
  - 999999
- The longer the ***common prefix*** the closer together (on logical ring)
- View address as hierarchy
- Cluster nodes with numerically ID close
- The closer $\Rightarrow$ more routing info $\Rightarrow$ denser routing table

Prefix Routing

Eg. Simple ID = 4 digit (range 0-3) string



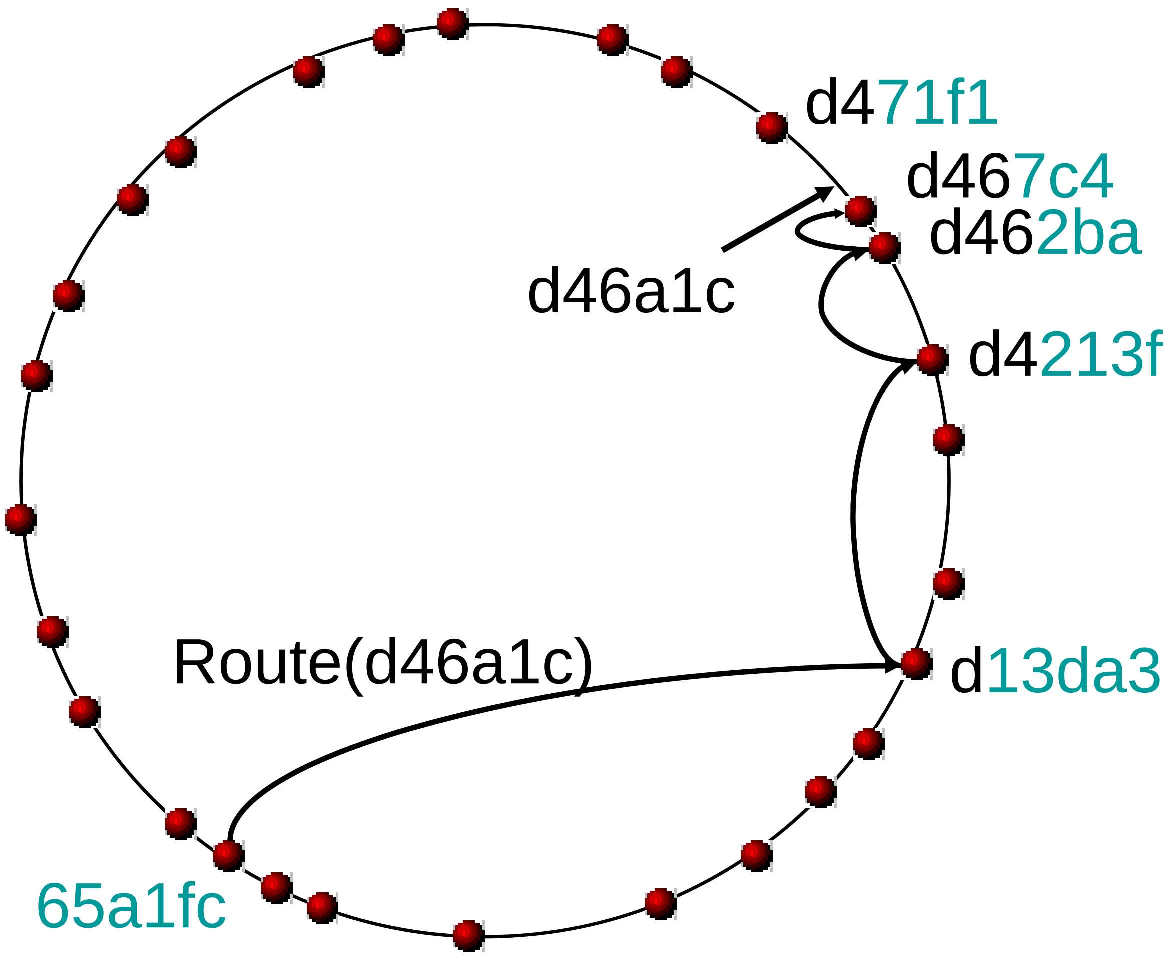
Routing Table for A:

Level 3	0***	1***	2***	3***
Level 2	00**	01**	02**	03**
Level 1	000*	001*	002*	003*
Level 0	0000	0001	0002	0003

NB: asociated IP not shown

***= ”don’t care” = select any (preferably close) node with matching prefix**

Pastry: Routing



Properties

- $\log_{2^b} N$ steps
- O(*log N*) state

Pastry routing table for node 65A1

NB: n = associated IP

p =	*GUID prefixes and corresponding nodehandles n*															
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	n	n	n	n	n	n		n	n	n	n	n	n	n	n	n
1	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6F	6E	6F
	n	n	n	n	n		n	n	n	n	n	n	n	n	n	n
2	650	651	652	653	654	655	656	657	658	659	65A	65B	65C	65D	65E	65F
	n	n	n	n	n	n	n	n	n	n		n	n	n	n	n
3	65A0	65A1	65A2	65A3	65A4	65A5	65A6	65A7	65A8	65A9	65AA	65AB	65AC	65AD	65AE	65AF
	n		n	n	n	n	n	n	n	n	n	n	n	n	n	n

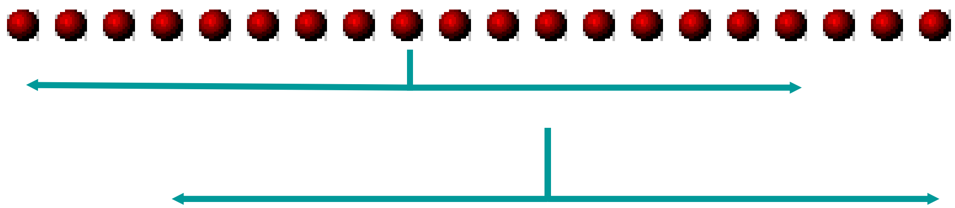
EX. Route

from: • **65A1** Common Prefix Length *p*=2

to: • **6544** Distinguising digit *i* = 4

Next-hop = *R[p,i]* = IP of 654* //index includes 0

Pastry: Leaf sets



Each node maintains IP addresses of the nodes with the L numerically closest larger and smaller nodeIds, respectively.

- routing efficiency/robustness
- fault detection (keep-alive)
- application-specific local coordination

Pastry: Routing procedure

If (destination is within range of our leaf set)
    forward to numerically closest member
else
    let *p* = length of shared prefix
    let *i* = value of *l*-th digit in *D*'s address
    if (*R[p,i]* exists)
        forward to *R[p,i]*
    else
        forward to a known node that
        (a) shares at least as long a prefix *p*
        (b) is numerically closer than this node

Pastry: Routing

Tradeoff

- O(*log N*) routing table size
  - $2^b * log_{2^b}N + 2l$
- O(*log N*) message forwarding steps
  - $\log_{2^b} N$

Pastry: Locality properties

- Overlay network not related to geography or network distance (IP hops, RTT,..)
- Risk very long/slow transmissions in overlay network
- Prefer to route via nodes in nearby in network distance

Proximity invariant:

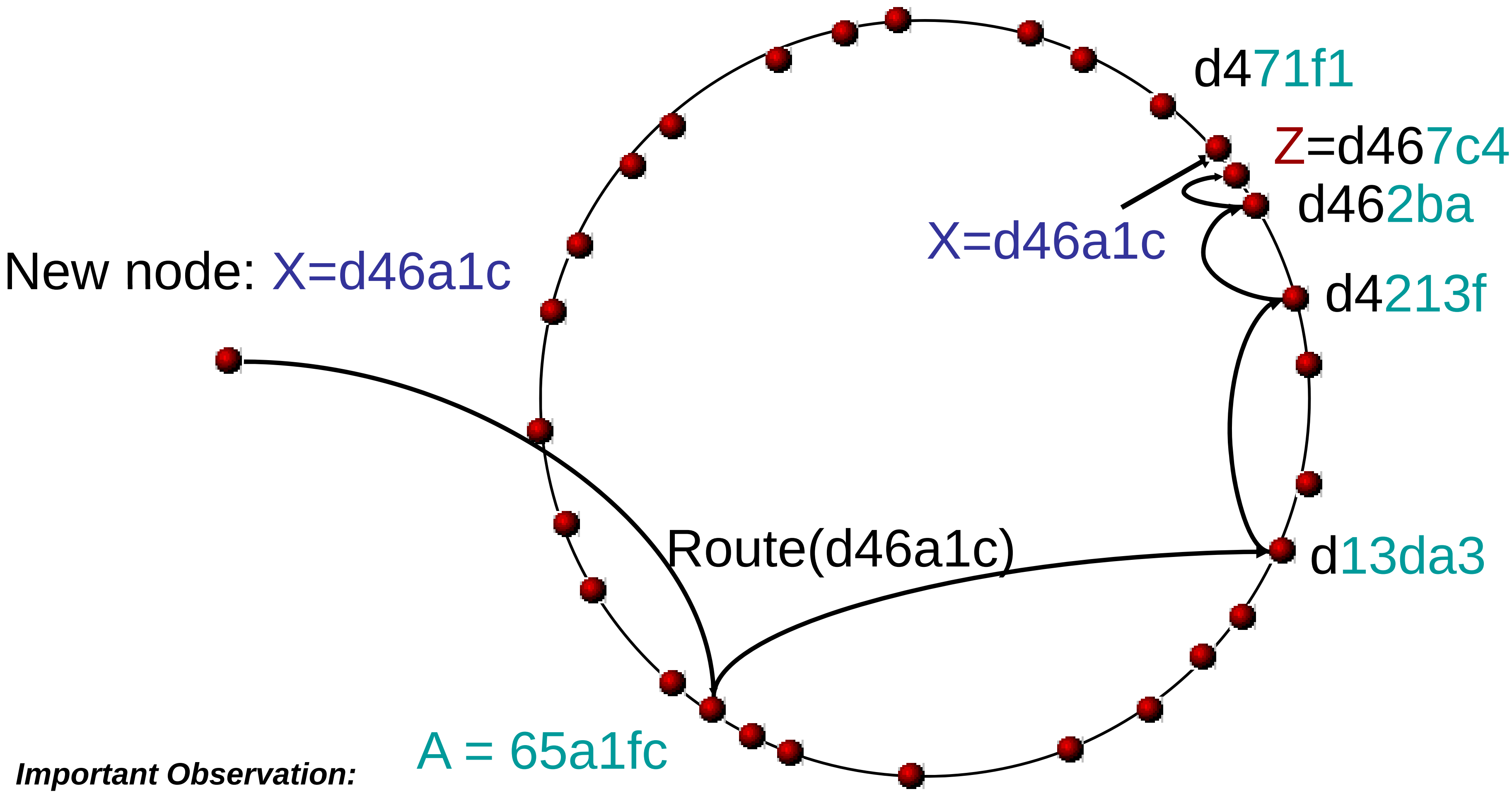
Each routing table entry refers to a node "nearby"
to the local node among
all nodes with the appropriate nodeId prefix.

Assumption: scalar proximity **metric**

- e.g. ping/RTT delay, # IP hops
- traceroute, subnet masks
- a node can probe distance to any other node
- (Incomplete DB on registration country of IP addresses)

- Maintain ***"Neighbor-set"*** of network distance nearby nodes

Pastry: Node addition



Important Observation:

- common prefix of X and intermediate node i increases by one at each hop
- Use i's routing column i as initial choice for X row i

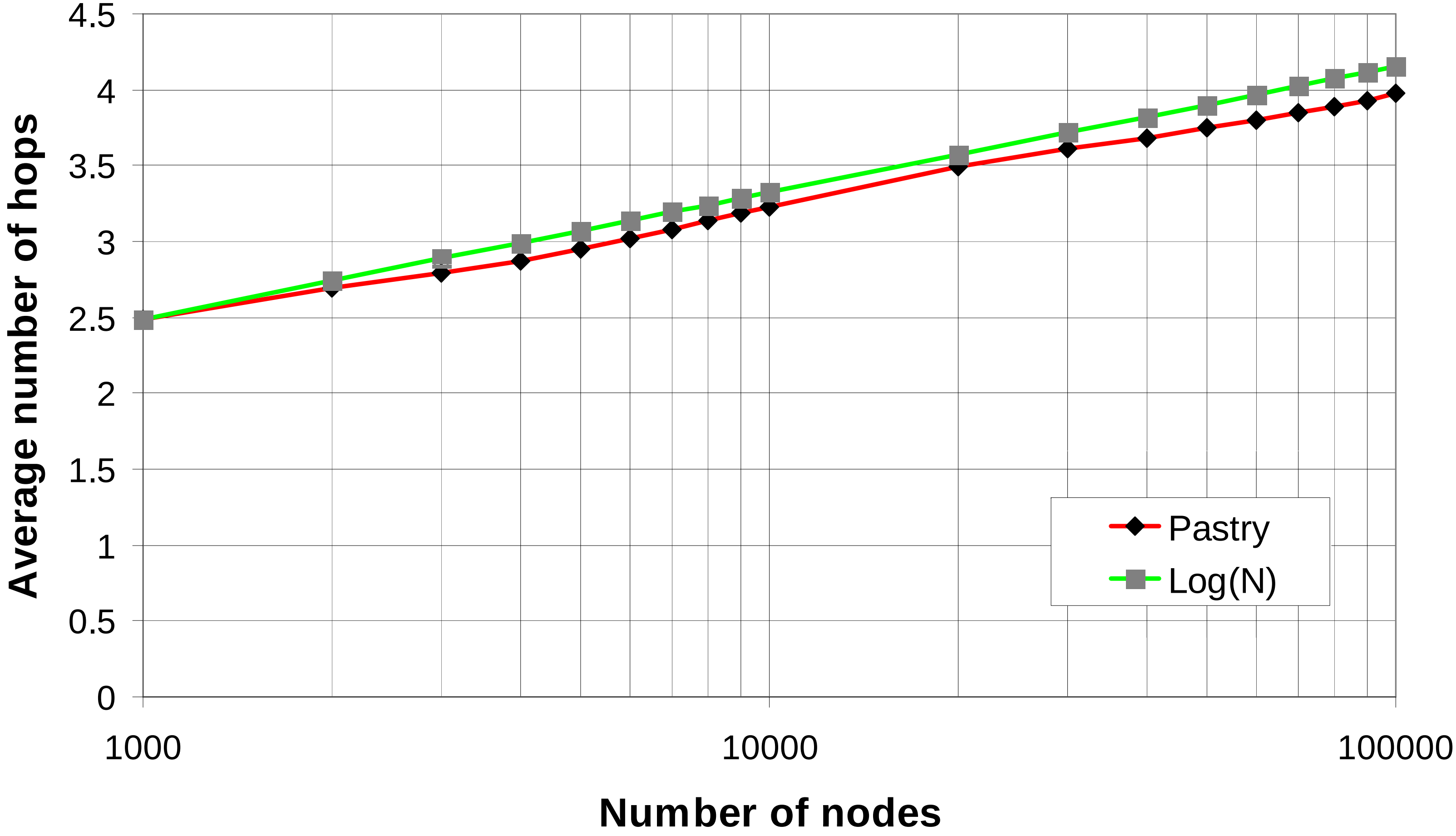
Pastry: Node addition

- New node **X** contacts “nearby” node **A**
- **A** routes “join” message to **X**, which arrives to **Z**, closest to **X**
- **X** obtains leaf set from **Z**, i’*th* row for routing table from i’*th* node from **A** to **Z**
- **X** informs any nodes that need to be aware of its arrival
  - **X** also improves its table locality by requesting neighborhood sets from all nodes **X** knows
  - In practice: optimistic approach

Node departure (failure)

- **Leaf set repair (eager – all the time):**
  - Send heart-beat messages to (left) leaf-set members
  - request set from furthest live node in set
- **Routing table repair (lazy – upon failure):**
  - get table from peers in the same row, if not found – from higher rows
- **Neighborhood set repair (eager)**

Pastry: Average # of hops



|L|=16, 100k random queries

END