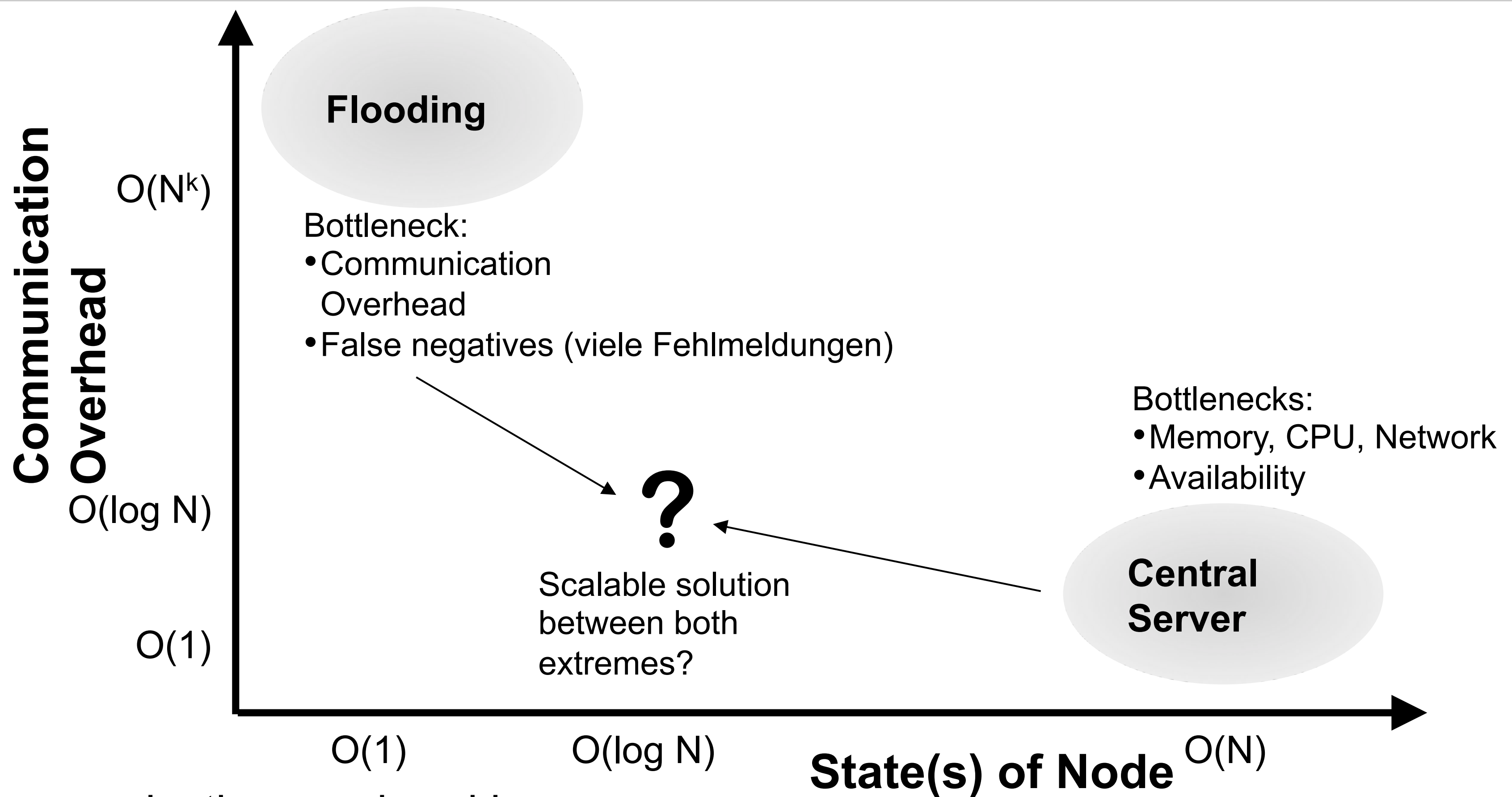
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Winter semester 2014
Jun.-Prof. Dr.-Ing. Kalman Graffi
Heinrich Heine University Düsseldorf

Motivation Distributed Indexing



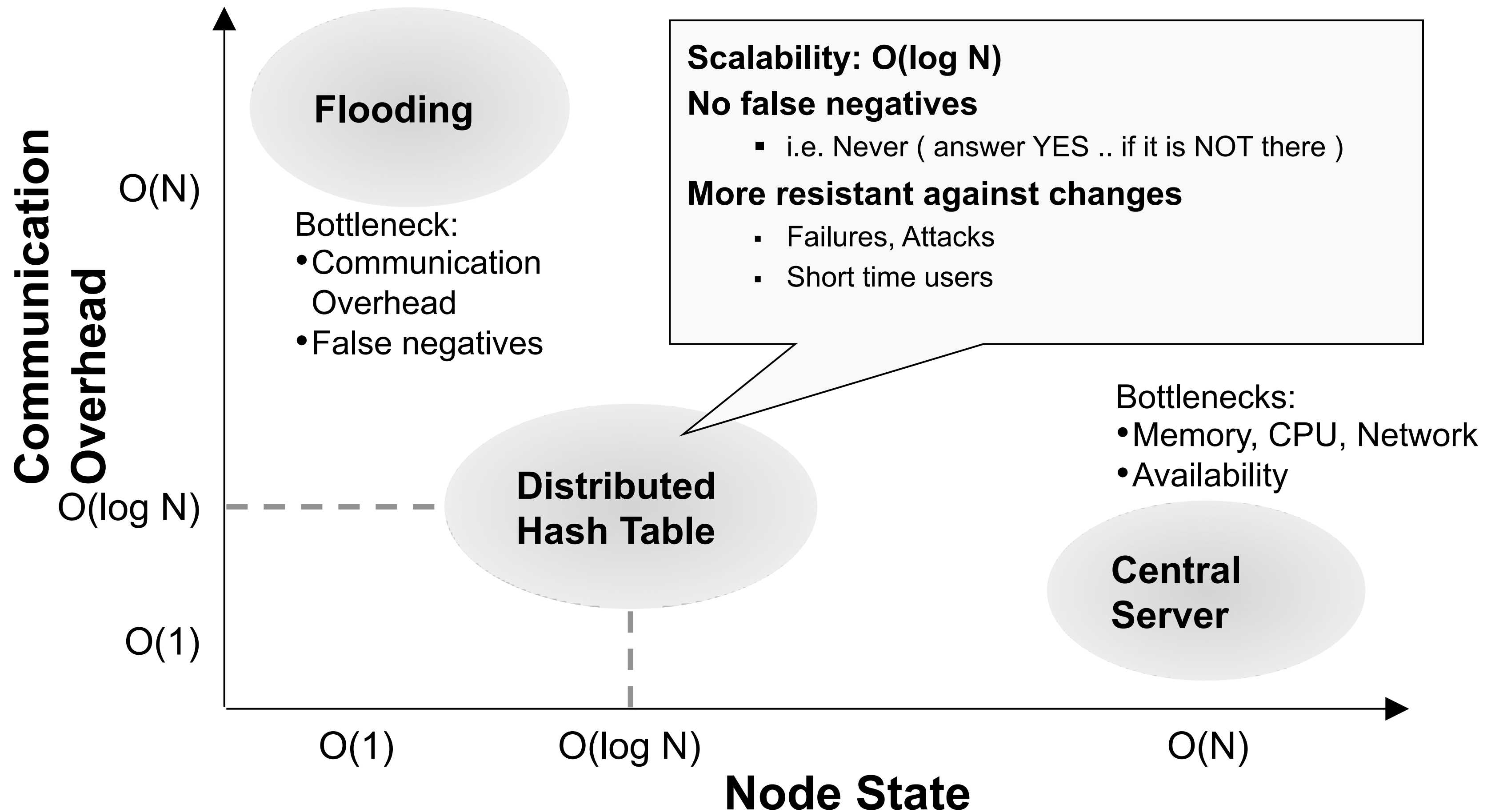
Communication overhead i.e.

- no. of hops vs.
- State(s) of node
  - (i.e. amount of routing entries stored in node, e.g. server)

The content of this slide has been adapted from "Peer-to-Peer Systems and Applications", ed. by Steinmetz, Wehrle

Motivation Distributed Indexing

Communication overhead vs. node state



The content of this slide has been adapted from “Peer-to-Peer Systems and Applications”, ed. by Steinmetz, Wehrle

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Distributed Hash Tables

Structured P2P Overlays

- Chord
- CAN
- Pastry

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Structured Homogenous P2P Overlay Networks
– DHT: Steps of Operation

This slide set is based on the lecture "Communication Networks 2" of Prof. Dr.-Ing. Ralf Steinmetz at TU Darmstadt

Distributed Hash Table: Steps of Operation

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Sequence of operations

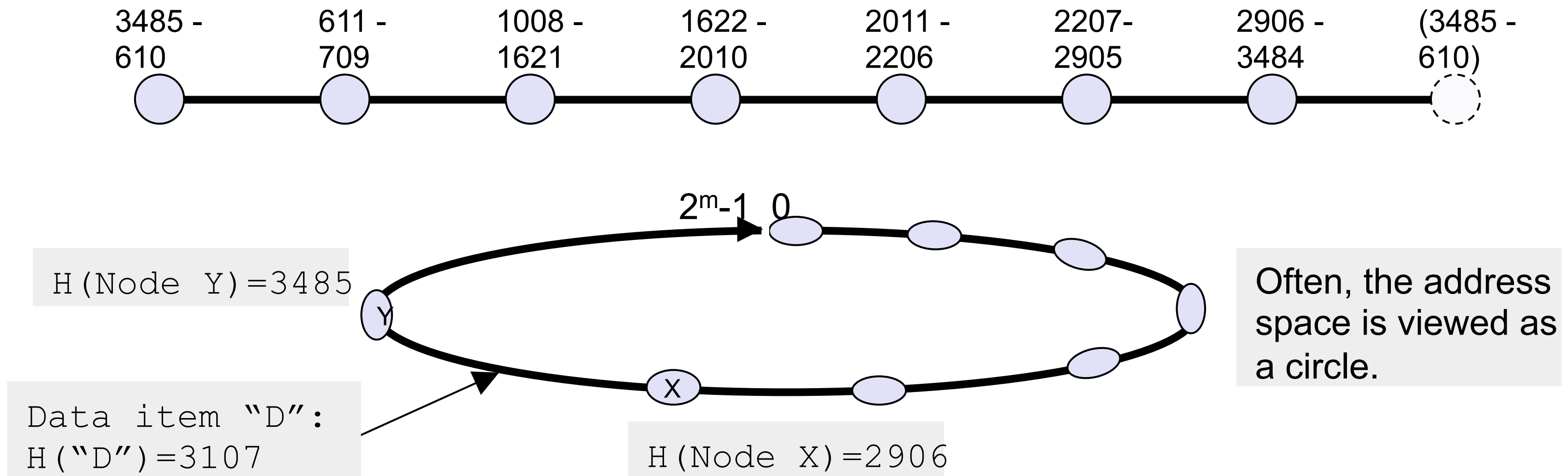
(at beginning) Mapping of
nodes and data → same address space

- Peers and content are addressed using flat identifiers (IDs)
- Common address space for
  - data and nodes
- Nodes are responsible for data in certain parts of the address space
- Association of data to nodes may change since nodes may disappear

(later) Storing / Looking up data in the DHT

- “Look-up” for data = routing to the responsible node
  - Responsible node not necessarily known in advance
  - Deterministic statement about availability of data

Step 1: Addressing in Distributed Hash Tables

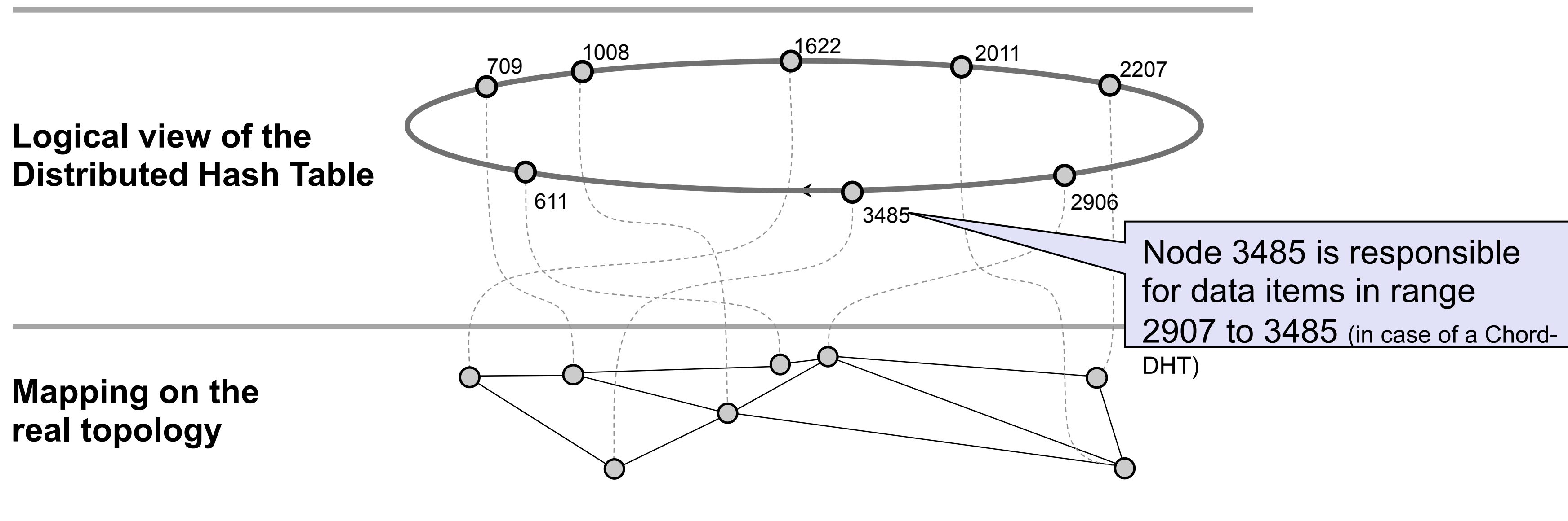


Mapping of content/nodes into linear space

- Usually: 0, …, 2^(m-1) >> number of objects to be stored
- Mapping of data and nodes → onto same address space (e.g. 0 to 2^(m-1))
  - with hash function
  - e.g., Hash(string) mod 2^m:
    - H(„my data") → 2313
- Association of parts of address space to DHT nodes

Step 2: Association of Address Space with Nodes

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF



Arrangement of the range of values

- Each node is responsible for part of the value range
  - Often with redundancy (overlapping of parts)
  - Continuous adaptation
- Real (underlay) and logical (overlay) topology are (mostly) uncorrelated

Step 3: Locating a Data Item

Locating the data

- content-based routing

Goal: Small and scalable effort

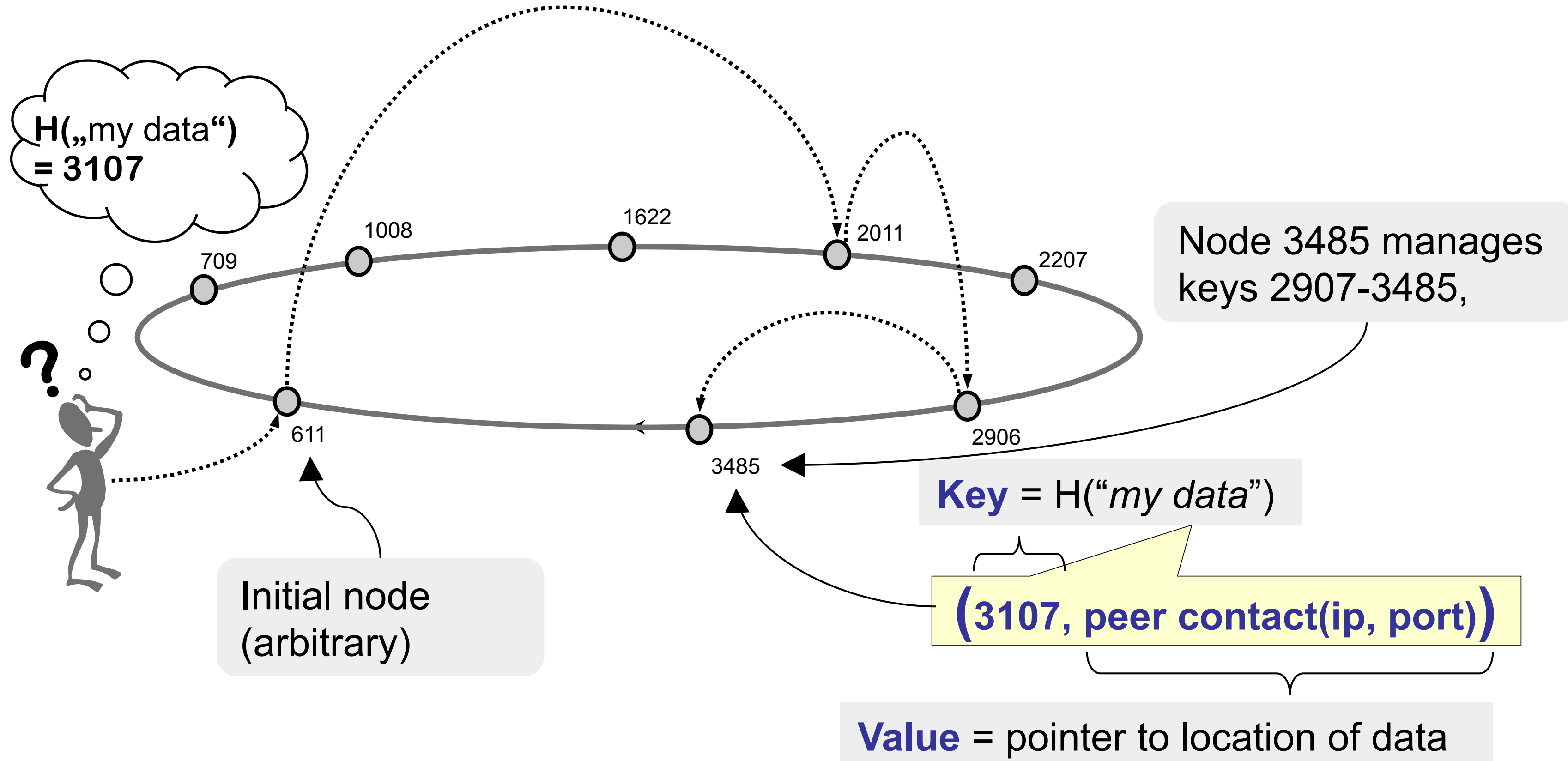
- O(1) with centralized hash table
  - But: Management of a centralized hash table too costly (server)

- Minimum overhead with distributed hash tables
  - O(log N):
    - DHT hops to locate object
  - O(log N):
    - number of keys and routing information per node
      - (N = no. of nodes)

Step 4: Routing to a Data Item

Routing to a key/value-pair

- Start lookup at arbitrary node of DHT
- Routing to requested data item (key)

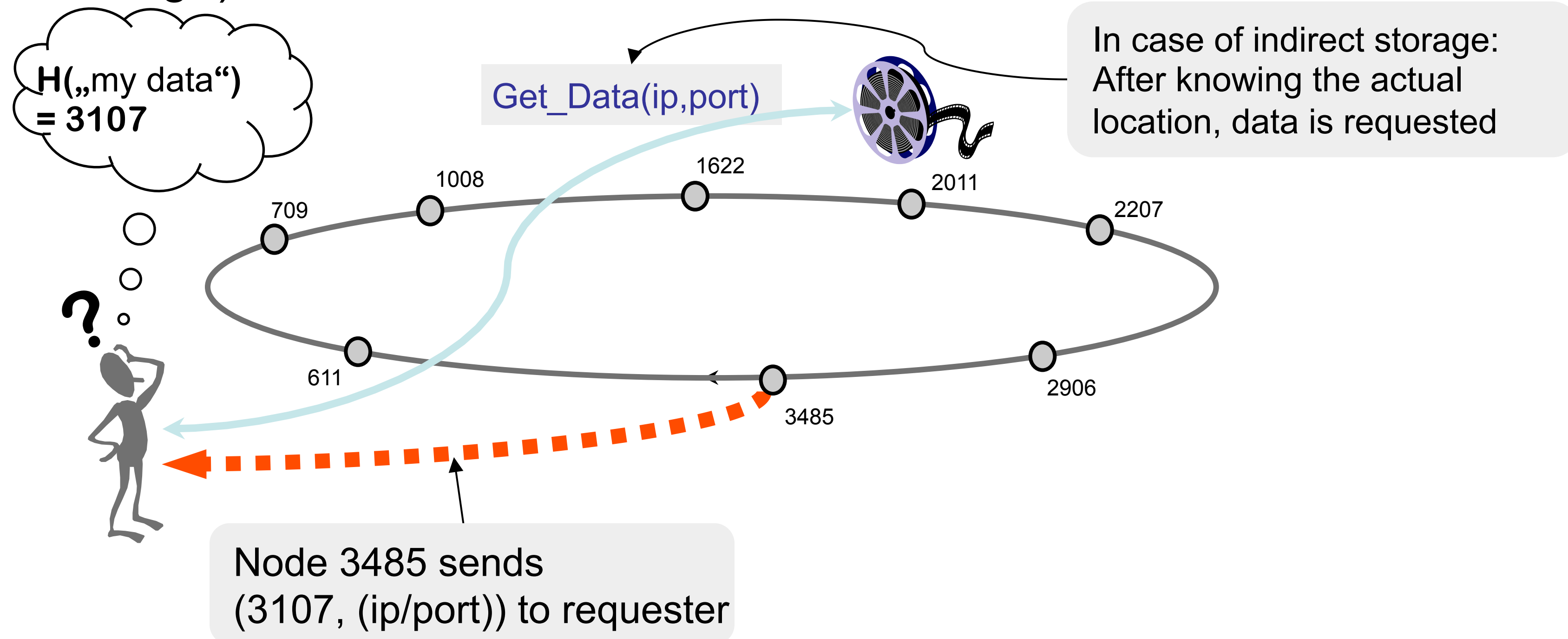


Step 5: Data Retrieval – Usage of located Resource

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Accessing the content

- Key/value-pair is delivered to requester
- Requester analyzes key/Value-tuple (and downloads data from actual location – in case of indirect storage)

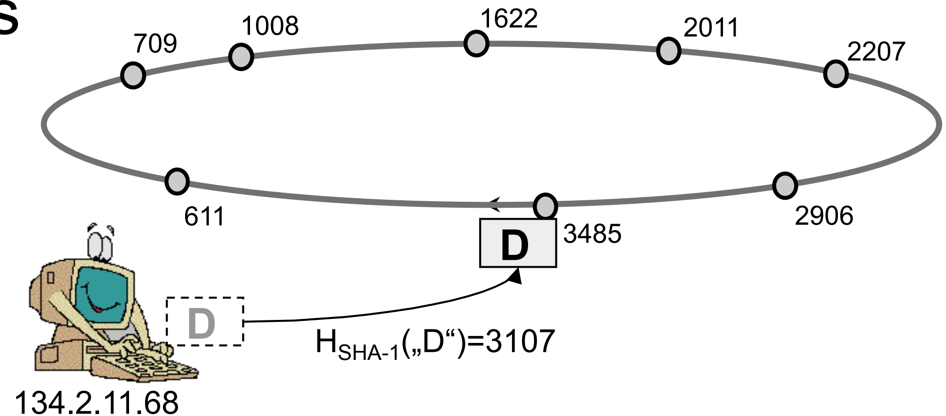


(Step 6) Where is the Data located?

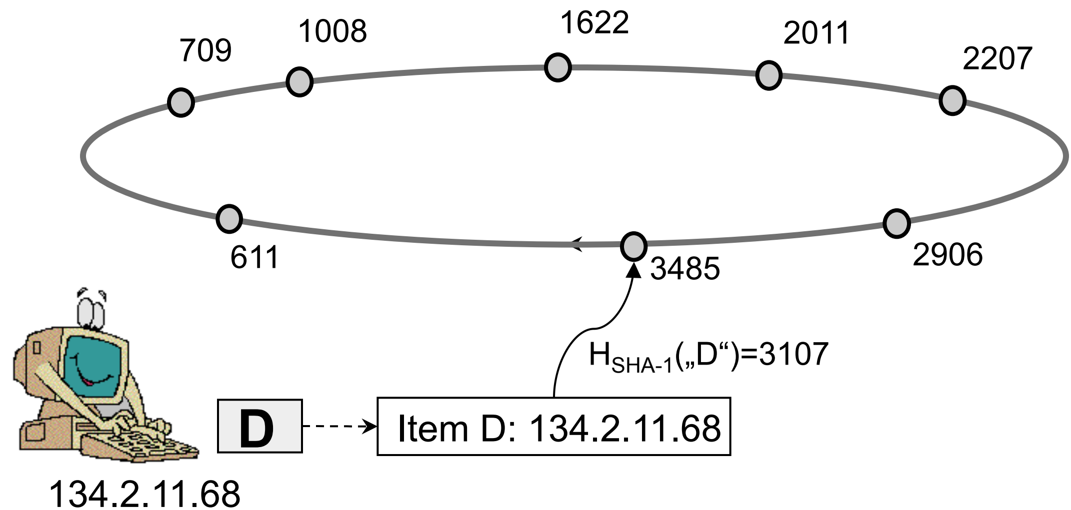
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Association of Data with IDs

- Direct Storage



- Indirect Storage



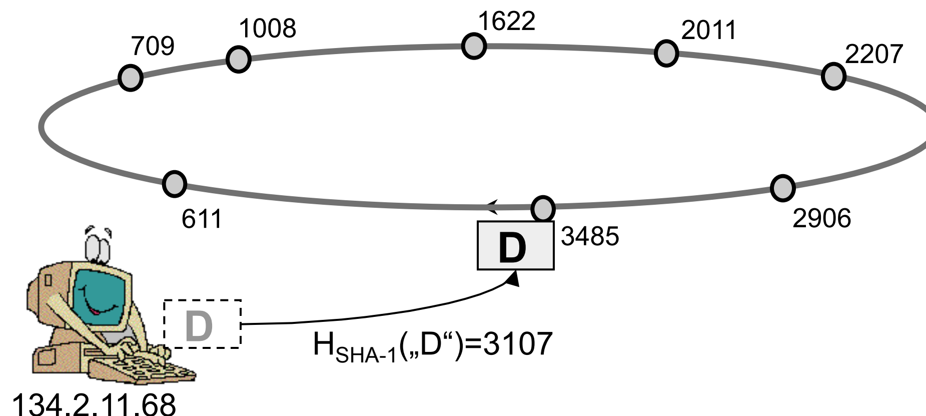
Association of Data with IDs – Direct Storage

How is content stored at the nodes?

- Example:
  H(“my data”) = 3107 is mapped onto DHT address space

Direct storage

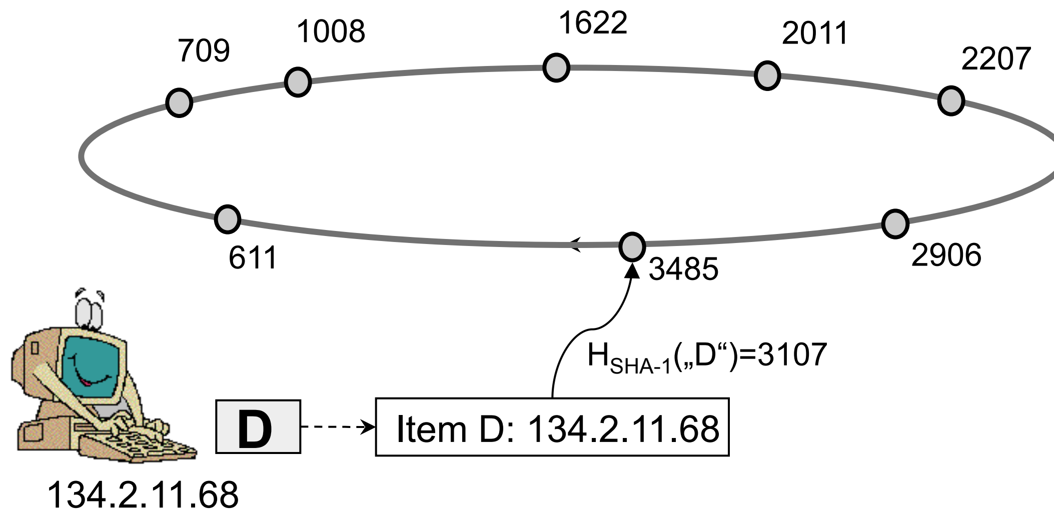
- Content is stored at responsible node for H(“my data”)
- Content is transferred at publication
  → INFLEXIBLE for large amount of content
  only OK if small amount (<1KB)
  → Not recommended for most applications



Association of Data with IDs – Indirect Storage

Indirect storage

- Nodes in a DHT store tuples like (key,value)
  - Key = Hash(„my data”) → 2313
  - Value is often real storage address of content: (IP, Port) = (134.2.11.140, 4711)
- MORE FLEXIBLE,
  - but one step more to reach content

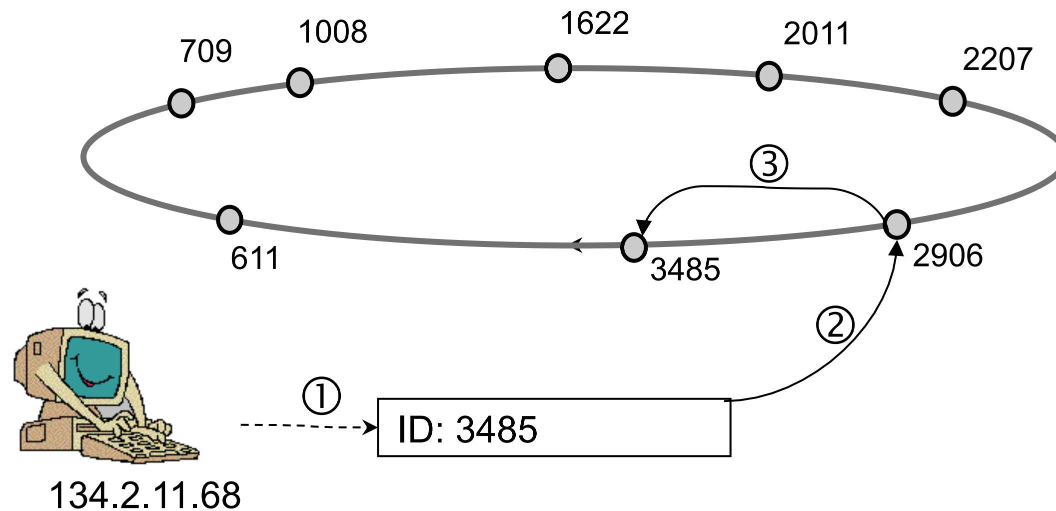


Distributed Hash Table: Insert and Delete a Node

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Join of a new node

- 1. Calculation of node ID
- 2. New node contacts DHT via arbitrary node
- 3. Assignment of a particular hash range
- 4. Copying of key/value-pairs of hash range (usually with redundancy)
- 5. Binding into routing environment



Node Failure and Node Departure

Failure of a node

- Use of redundant key/value pairs (if a node fails)
- Use of redundant / alternative routing paths
- Key-value usually still retrievable if at least one copy remains

Departure of a node

- Partitioning of hash range to neighbor nodes
- Copying of key/value pairs to corresponding nodes
- Unbinding from routing environment

Stabilize Function

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Routing table maintained

- Connectivity to specific nodes is important
- Node failures: no time to inform contacts

Stabilize topology

- Overlay nodes typically check periodically connectivity
  - Heartbeat / alive message
- If contact lost:
  - Find new suitable contanct
- If new (closer) contact know
  - Establish new connection

Some overlay exist, that do not actively stabilize

Summary and Conclusion of DHTs

Properties:

- Hash buckets distributed over nodes
- Nodes form an overlay network
  - Route messages in overlay to find responsible node
- Routing and ID labeling scheme in the overlay network is the difference between different DHTs
- DHT behavior and usage:
  - Node knows “object” name and wants to find it
- Unique and known object names assumed
  - Node routes a message in overlay to the responsible node
  - Responsible node replies with “object”
- Semantics of “object” are application defined

Summary and Conclusion of DHTs: Core Components

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Hash table

- Uniform distribution
- Shifted view for each node
  - (adding a node-related offset)

Mapping function

- Node IDs and item keys share the same key-space
- Rules for associating keys to particular nodes

Routing tables

- Per-node routing tables that refer to other nodes
- Rules for updating tables as nodes join and leave/fail

Routing algorithms (operations on keys):

- XOR-based (e.g. Kademlia)
- Shift operations (e.g. D2B)
- Distance-based (e.g. Chord)
- Prefix-based (e.g. Pastry)

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Structured Homogenous P2P Overlay Networks – Chord

This slide set is based on the lecture "Communication Networks 2" of Prof. Dr.-Ing. Ralf Steinmetz at TU Darmstadt

Chord: An Efficient Lookup Network

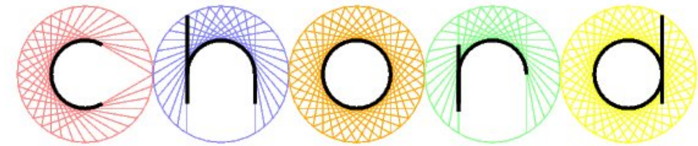
Chord uses SHA-1 hash function

- Results in a 160-bit object/node identifier
- Same hash function for objects and nodes

Node ID hashed from e.g., IP address

Object ID hashed from object name

- Object names assumed to be known



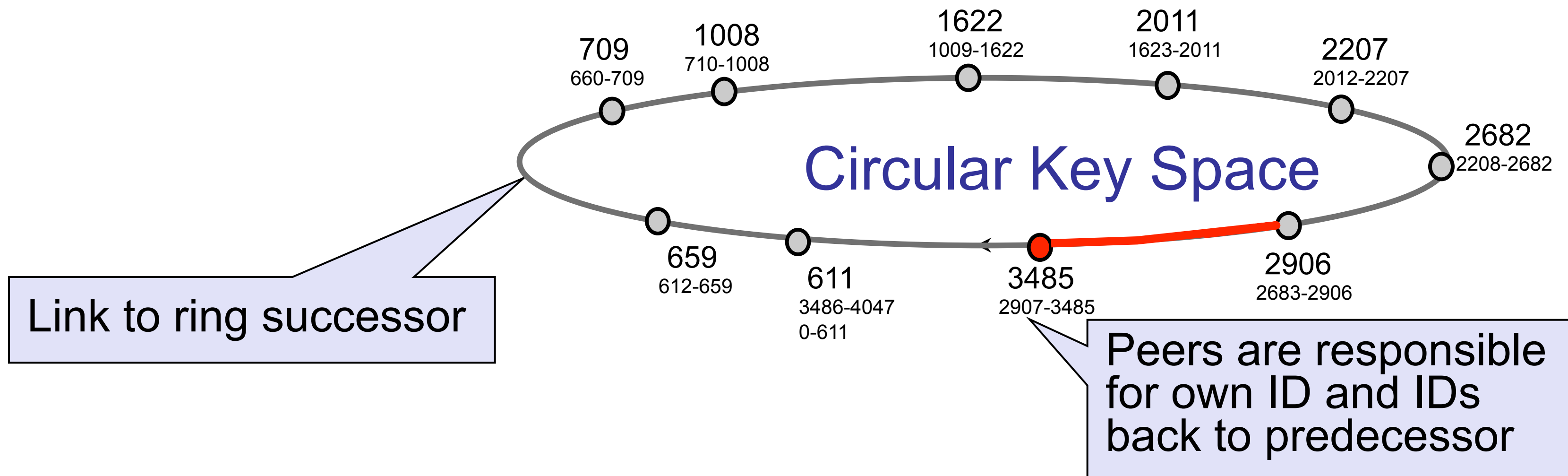
Chord is organized in a ring which wraps around

- Nodes keep track of predecessor and successor
  - System invariant for valid network operation
- Node responsible for
  - objects between its predecessor and itself
- Fingers used to enable efficient content addressing
  - O(log(N)) fingers lead to lookup operation of O(log(N)) length

Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications (2001) by Ion Stoica, et.al.

Chord: Network Topology

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF



Basic ring topology

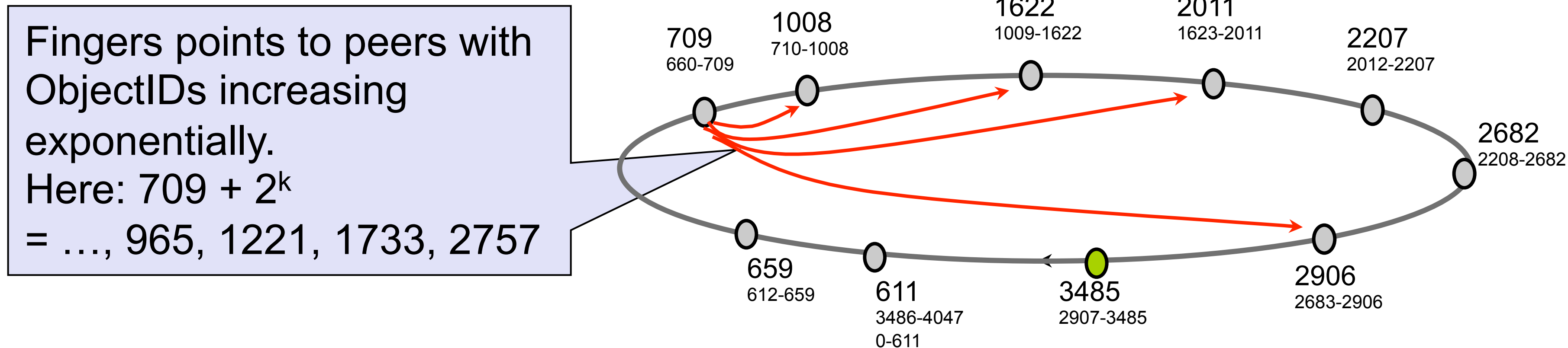
- Successor/ Predecessor

Uses SHA-1to map

- IP address/object name onto
- 160 Bit ID

Chord: Network Topology

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF



Enhanced topology

- kth finger of Peer n is shortcut pointing to peers being responsible for Object ID (n + 2^k)
- k ranges from 0 to log(N)
- O(log(N)) fingers lead to lookup operation of O(log(N))

Example Peer ID 7, ID range: 0 - 127

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

17 peers in the network:

- 3, 7, 10, 19, 21, 31, 36, 37, 51, 60, 65, 78, 82, 90, 93, 101, 105

log_2(128) = 7 → 7 fingers

- Calculate finger IDs, find corresponding peers

Fingers of node 10

Finger No.	Calculation	Finger ID	Real Peer
0	10+2^0	11	19
1	10+2^1	12	19
2	10+2^2	14	19
3	10+2^3	18	19
4	10+2^4	26	31
5	10+2^5	42	51
6	10+2^6	74	78

Responsibility range

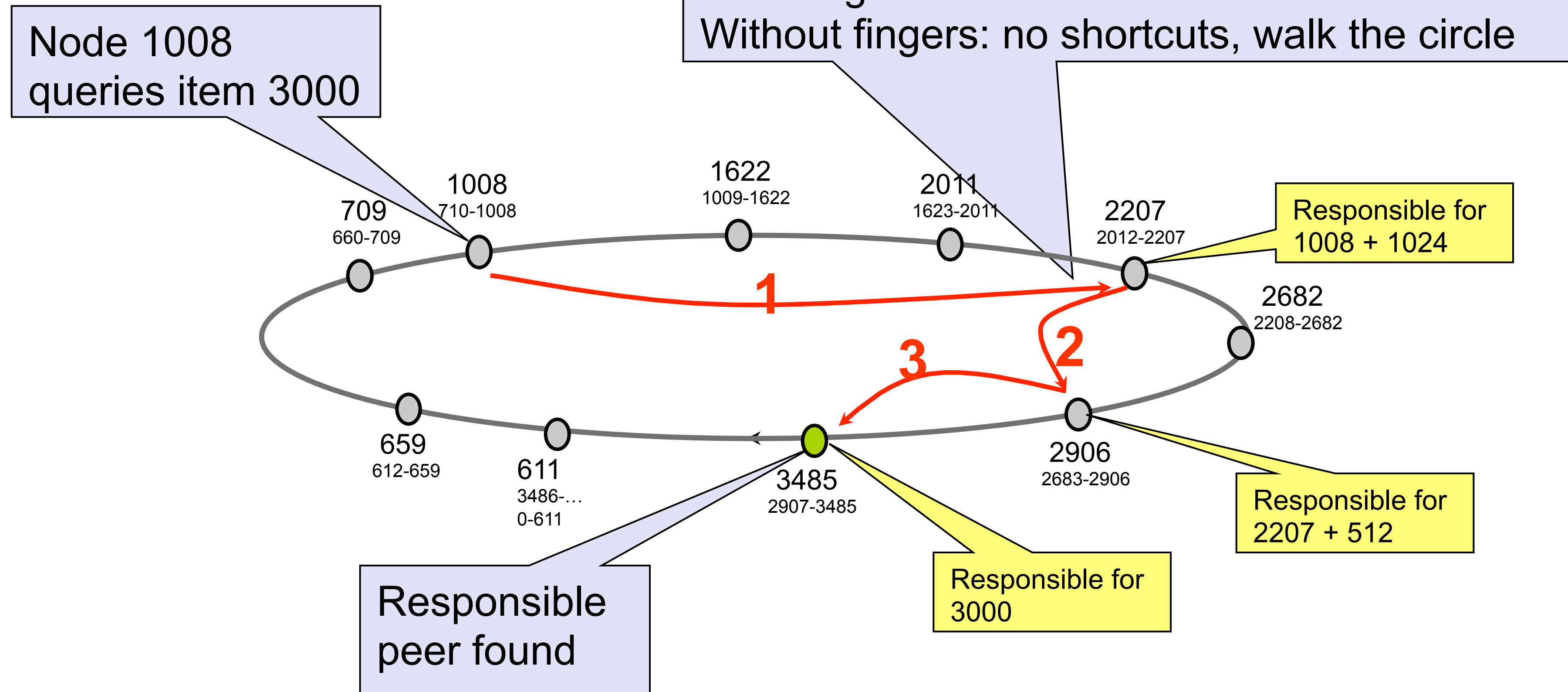
Peer ID	From	To
3	106	3
7	4	7
10	8	10
19	11	19
21	20	21
31	22	31
36	32	36
37	37	37
51	38	51
60	52	60
65	61	65
78	66	78
82	79	82
90	83	90
93	91	93
101	94	101
105	102	105

Chord: Addressing Content

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

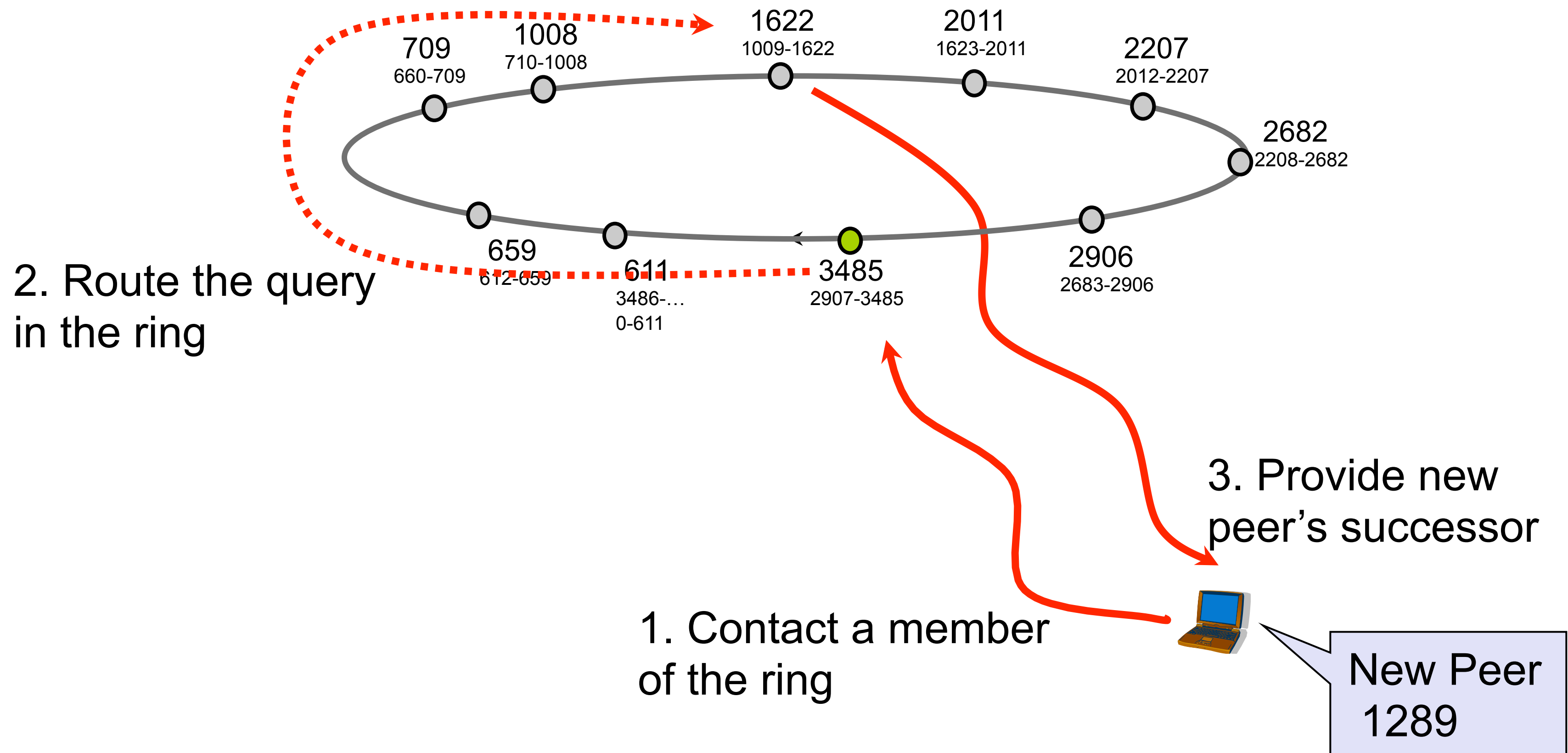
Query

- Contains the hash value of the queried content
- On each step the distance from the destination is halved (remember fingers)

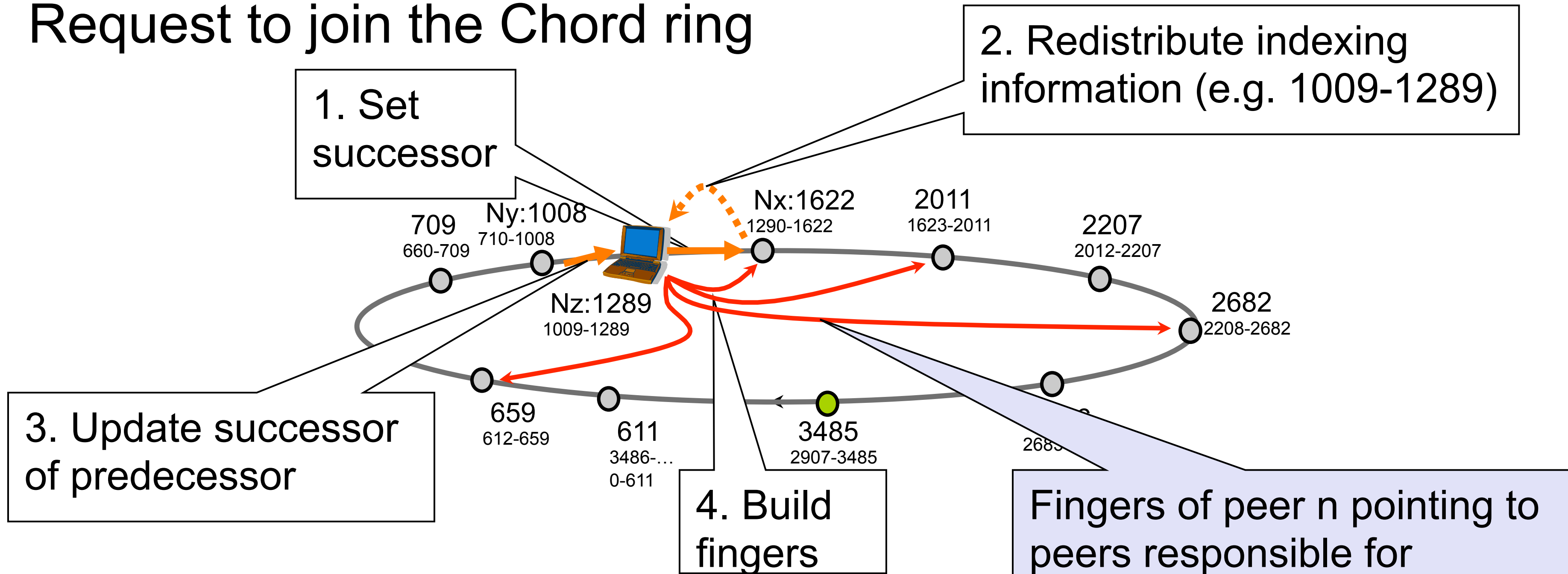


Chord: Join Procedure (1)

Request to join the Chord ring



Chord: Join Procedure (2)



1. & 2. Notify Successor

Actions:

N_Z: Set Successor (N_X)
N_Z: Notify N_X
N_X: Set Predecessor
N_X: Copy items (index) to N_Z

3. & 4. Stabilize

Actions:

N_Y: Ask Predecessor of N_X
N_Y: Set Successor (N_Z)
N_Y: Notify N_Z
N_Z: Set Predecessor (N_Y)
N_X: Clear moved items
All: Fix Fingers

Chord – Evaluation

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Advantages

- Lookups will find the target (if it exists)
- Good scalability of O(log n)
- Very popular, shows main idea of DHTs
- Often used as basis for research extensions

Drawbacks

- Heterogeneity not supported
  - all nodes are treated equally, although some are strong/weak
- Maintaining unidirectional links (fingers) is only beneficial for one party
  - Traffic costs not optimally used
- Only one-directional routing: not optimally efficient

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Structured Homogenous P2P Overlay Networks – CAN: Content Addressable Network

This slide set is based on the lecture "Communication Networks 2" of Prof. Dr.-Ing. Ralf Steinmetz at TU Darmstadt

Content Addressable Network (CAN)

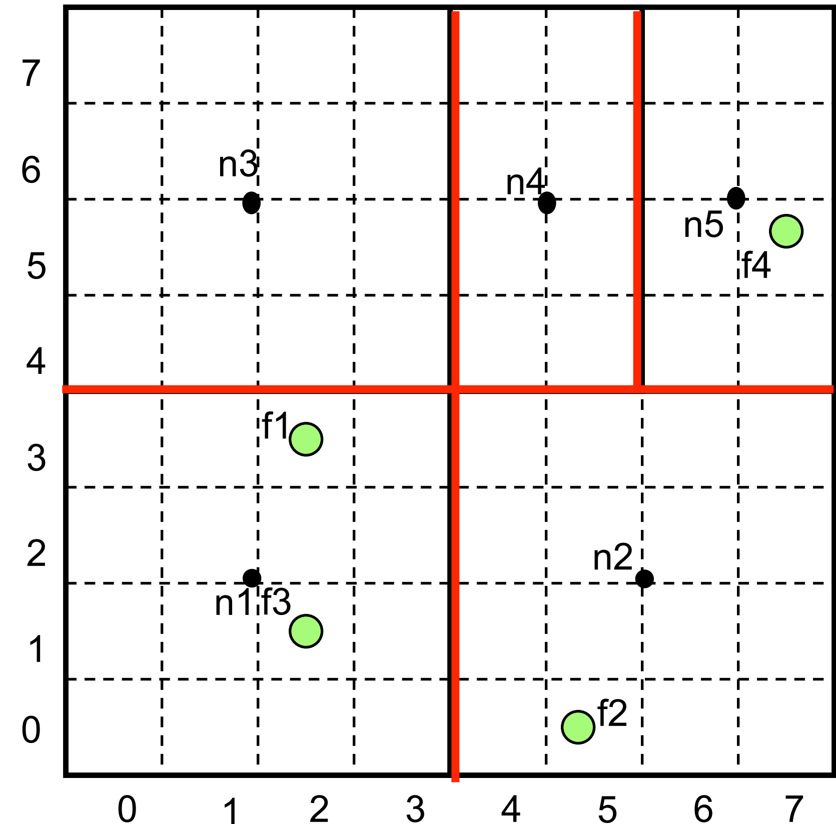
HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

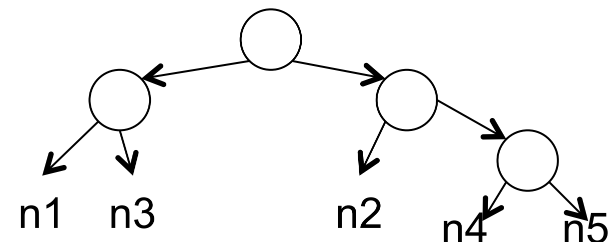
Architecture:
A hash-table in a d-dimensional Cartesian coordinate space, over a D-torus

- Cyclical d-dimensional space
- d hash-functions, 1 per coordinate
  - PeerID(p) = (h1(p),h2(p),... hd(p))
    - May shift dynamically
  - ObjID(obj) = (h1(obj),h2(obj),..., hd(obj))

CAN nodes

- Each node is responsible for a distinct rectangular zone of the space
- Manages all the files that hash into its zone
- Nodes cover together the entire space

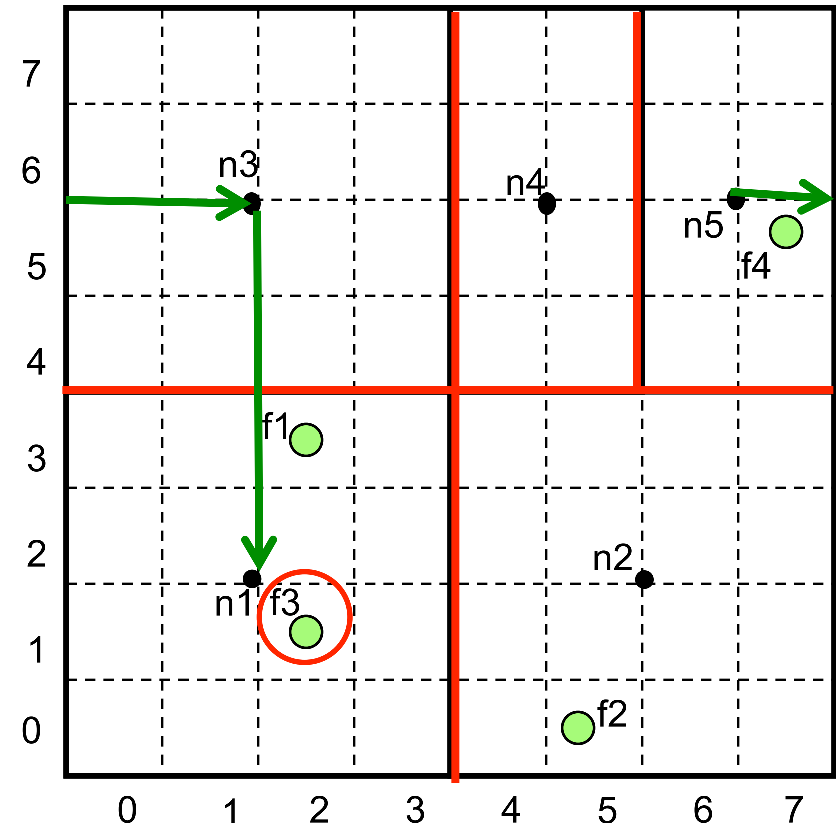




CAN: Routing

2 CAN nodes are neighbors if

- their zones overlap along d-1 dimensions and
- abut (“angrenzen”) along one dimension
- every node knows
  - the IP addresses of its neighbors
  - the coordinates of neighboring zones
- Nodes can communicate only with their neighbors



Properties

- Routing table size O(d)
- Guarantees that a file is found in at most … steps, where n is the total number of nodes

Lookup:

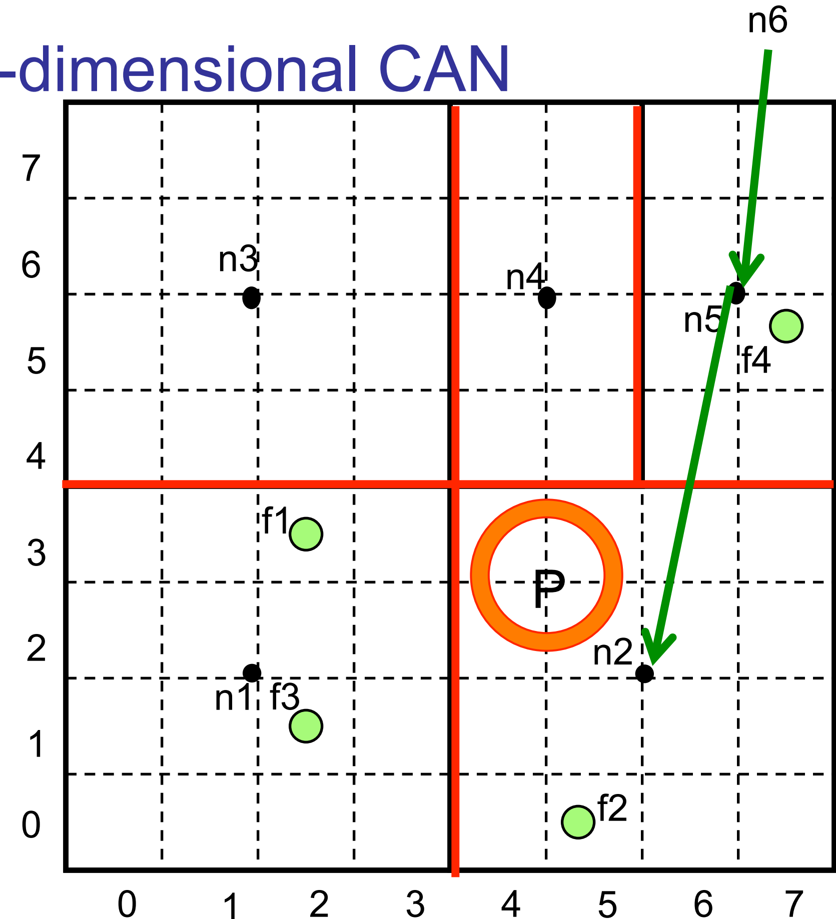
State per node: 2d

CAN: Routing

New node has to acquire a zone to be responsible for

Steps:

- Node chooses randomly a point P in the space
- Route to node responsible for P
  - Bootstrap node helps joining
- Zone of responsible node is split deterministically
  - Clear rules
  - With d=2: first x, then y axis



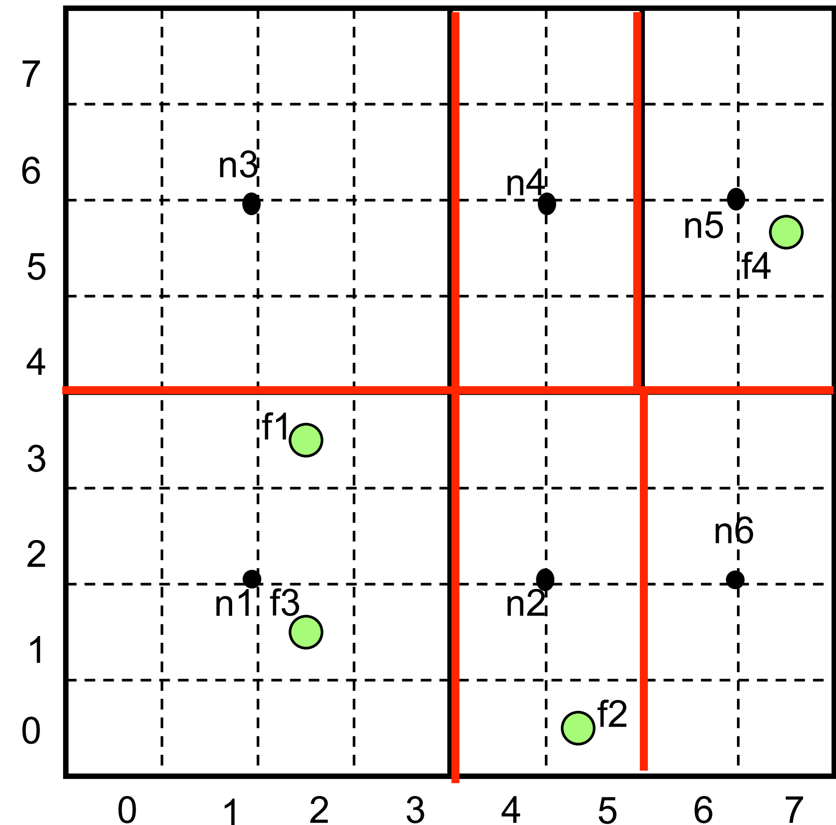
CAN: Joining of a new Node

Steps continued:

- Contacted node splits own responsibility space
- Note: P not of relevance
- Transfer to new node:
  - content corresponding to new space
  - Information on neighbors
- Inform neighbors about new node

Volume Balancing

- Bootstrap node forwards join request to largest neighbor
- Stops: local optimum

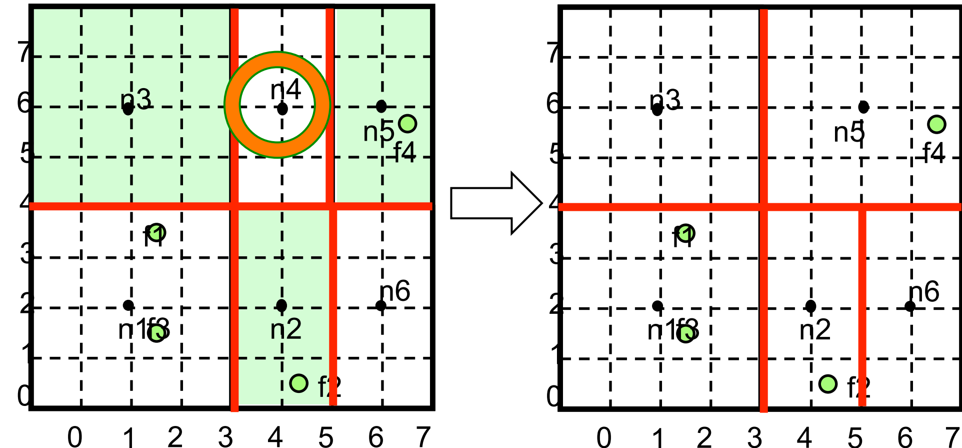


CAN: Leaving of a Node

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

When a node leaves

- Ideally notifies neighbors
- In bad case: failure



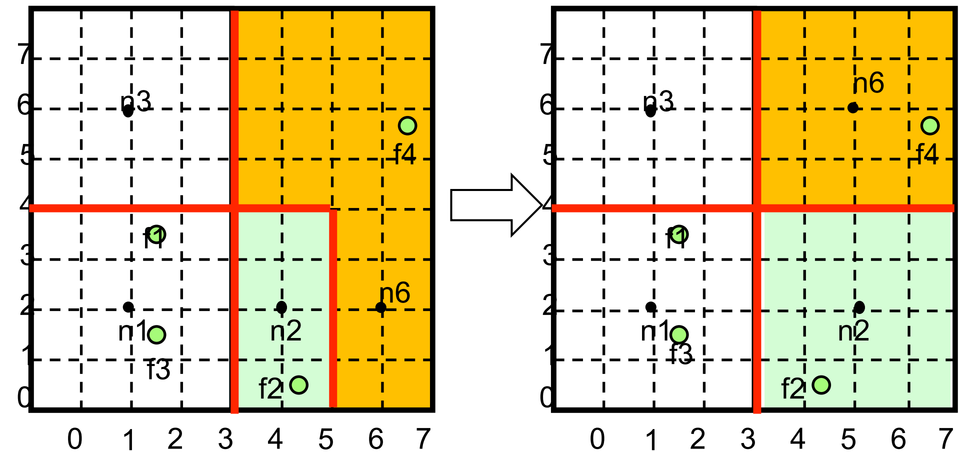
Takeover protocol

- Noticed by neighbors
- Neighbors can send takeover messages to lost node‘s neighbors after timeout
- Small zone → smaller timeout
- Smallest neighbor takes over

CAN: Defragmentation

Defragmentation

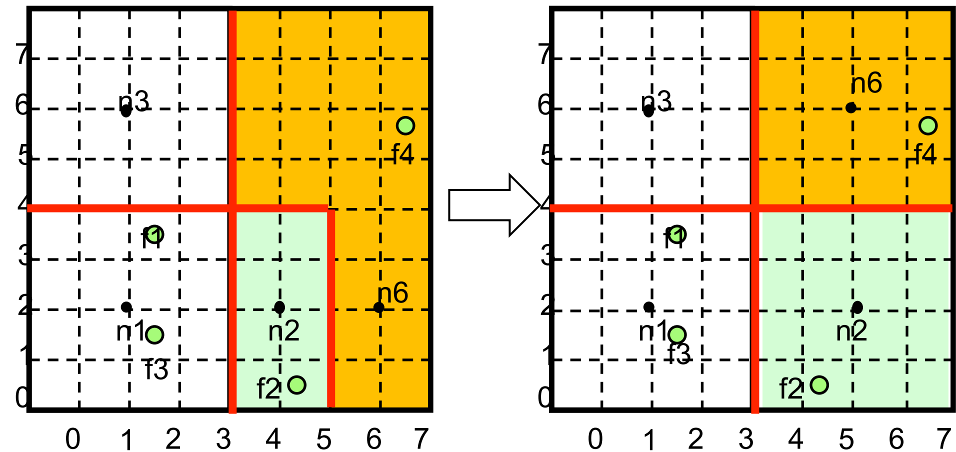
- Can be started by peers with several zones
- Means: handing over smallest zone of the peer



CAN: Defragmentation

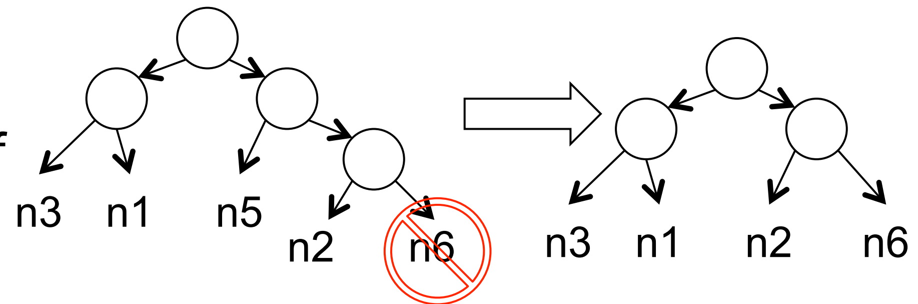
Defragmentation

- Can be started by peers with several zones
- Means: handing over smallest zone of the peer



Easy case:

- „Neighbor" of lost zone in tree is leaf
- Hand over lost zones to leaf
- Single leaf is „pulled" up in tree

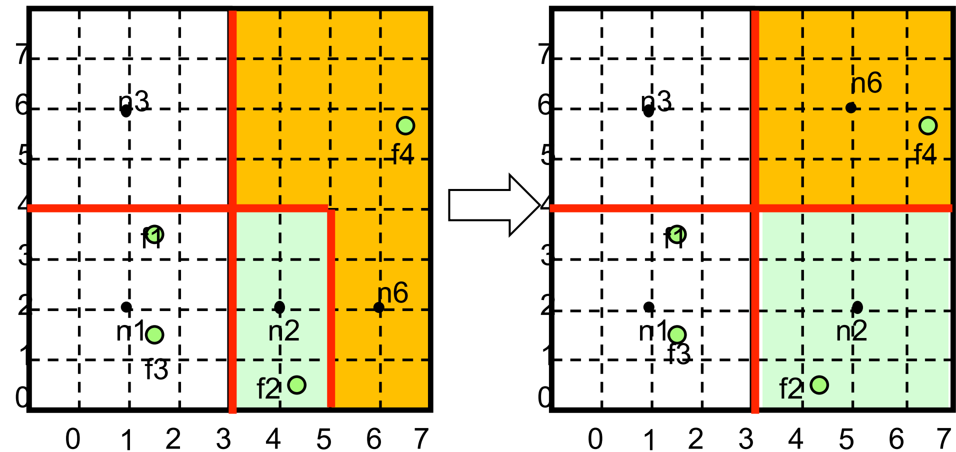


CAN: Defragmentation

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

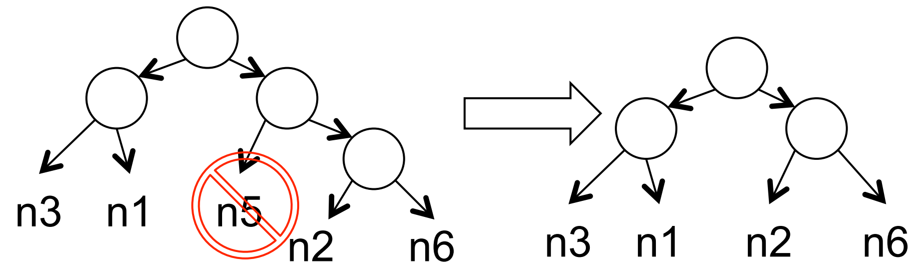
Defragmentation

- Can be started by peers with several zones
- Means: handing over smallest zone of the peer



Complex case:

- „Neighbor“ of lost zone in tree is inner node
- Find inner node x in sub-tree with two directly neighboring leafs A and B
- Merge zones of A and B → A
- B is free, takes over lost zone



CAN – Evaluation

Advantages

- Lookups will find the target (if it exists)
- Good scalability (small node state)
- Fixed routing table size
- Proximity-based routing included
- Often used as basis for multi-dimensional search

Drawbacks

- Lookup performance is not good O (d*n^(1/d))
- No relevance in practice

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Peer-to-Peer Systems

Structured Homogenous P2P Overlay Networks
– Pastry and Key-based Routing

This slide set is based on the lecture "Communication Networks 2" of Prof. Dr.-Ing. Ralf Steinmetz at TU Darmstadt

Pastry – Introduction

New approach: Emphasize locality

- If several equal nodes are available,
  prefer the node that is physically closest

Two metrics

- ID distance
- Physical distance (“Proximity”)

128-bit-IDs, arranged in a circle

- Generated by hashing e.g. the node’s IP address or public key

Variable b defines the size of the routing steps = 2^b; usual value = 4

- $b=4$ => 4 bits of the ID represented by a character 0-9,A-F
- Routing resolves with each hop at least one character more
- Tradeoff between routing table size and maximum number of hops

Lookup hops scale with $O(\log_{2^b} N)$

Delivery is guaranteed unless $L/2$ adjacent nodes fail concurrently (with usually $L = 16$ or 32)

Pastry – Routing procedure

In each routing step:

- Prefix-based forwarding:
- A node forwards a message to another node
  - whose ID shares with the target key a prefix
  - that is at least one digit (= b bits) longer
  - than the prefix that is shared with the current node's ID

If no such node is found:

- Numerical distance based forwarding:
- the message is forwarded to a node
  - with the same shared prefix length which is numerically closer

State Information in Pastry

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

ID Space: [0 , 2^128[

- Randomly assigned while joining
- Base b (often 4, here b=2)
  - b bits to encode a "character"

Routing table

- Used for prefix-based routing
- Typical size:
  - log_(2^b) (N) rows
  - 2^b – 1 entries per row
- Row nr. i contains only nodeIDs sharing a prefix of length i with current node

Leaf set

- |L| closest node IDs
- Typical size: L = 2^b or 2x2^b

Leaf Set

greater	10233121	10233122	10233131	10233133
smaller	10233033	10233021	10233001	10233000

Routing table

	0	1	2	3
0	-0-2212102	1	-2-2301203	-3-1203203
1	0	1-1-301233	1-2-230203	1-3-021022
2	10-0-31203	10-1-32102	2	10-3-23302
3	102-0-0230	102-1-1302	102-2-2302	3
4	1023-0-322	1023-1-000	1023-2-121	3
5	10233-0-01	1	10233-2-32	
6	0		102331-2-0	
7			2	

Routing state of node 10233102, base 2

Pastry Routing Table

Routing table

- Each node has a neighbor map with multiple levels
  - Each level represents a matching prefix up to digit position in ID
  - A given level has number of entries equal to the base of ID
  - i-th entry in j-th level is closest node which starts prefix(N,j)+"i"
  - Example: 7th entry of 2th level for node 123456… is the closest node with ID beginning with 127…

Routing in Pastry

Routing

- Always route to node closer to target
  - At n-th hop:
    - Prefix with first n digits already matched
    - look at n+1-th level in neighbor map / routing table
  - → "always" one digit more
- Not all nodes and links are shown

Object responsibility

- Node responsible for objects which have same/closest ID
  - Unlikely to find such node for every object
  - Node responsible also for "nearby" objects
- → Responsibility area

Routing Protocol

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Message for key K arrives at node X

- Let X= 10233102, b=2 → 4 characters

1. Check if K in scope of Leaf Set
   - E.g. K = 10233030
   - Direct forwarding to 10233033
2. If not (1) use Routing Table
   - Let l:= prefix length of K and X
   - E.g. K = 10320102, l=2
   - Check level 3, prefix 103 → 10-3-23302
3. If not (1) and no routing table entry
   - E.g. K = 10233300
   - Pick closest peer from routing table: 10233-2-32, as closer than 10233102
4. If X is closest to K than any node in Leaf Set (and Routing Table)
   - X is responsible for K, routing ends

Leaf Set

greater	10233121	10233122	10233131	10233133
smaller	10233033	10233021	10233001	10233000

Routing table

	0	1	2	3
0	-0-2212102	1	-2-2301203	-3-1203203
1	0	1-1-301233	1-2-230203	1-3-021022
2	10-0-31203	10-1-32102	2	10-3-23302
3	102-0-0230	102-1-1302	102-2-2302	3
4	1023-0-322	1023-1-000	1023-2-121	3
5	10233-0-01	1	10233-2-32	
6	0		102331-2-0	
7			2	

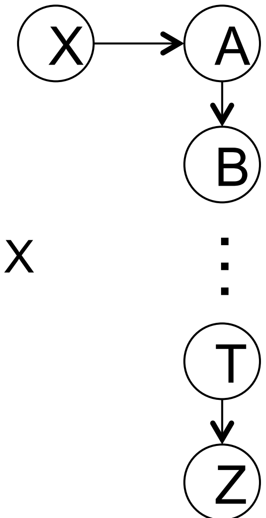
Routing state of node 10233102, base 2

Pastry – Node join

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

New node X wants to join

- A is assumed to be physically close to X
- Z is assumed to be responsible for the key "X"



Join protocol

- X asks existing node A to route JOIN message to key X
- JOIN message will be routed to node Z which is closest to key X
- A, Z and all nodes on the route send their state tables to X

X uses following sets as basis for its routing sets

- Z's leaf set

The n-th row of the routing table is copied from the n-th node encountered during the JOIN message routing process

- n = 0: A's row 0
- n = 1: B's row 1…

Finally, X sends a copy of its state tables to all nodes contained in them so that those nodes can update their state tables

Pastry – Node departure

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Failed node was in leaf set

- Ask the outermost node on the side of the failed node for its leaf set
- Insert appropriate new node into leaf set after validating that it is alive

Failed node was in routing table

- Ask another node in the same row for a contact at the position of the failed node
- If no alive node is in the failed node's row, use the next row

Pastry – Evaluation

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Advantages

- Lookups will find the target (if it exists)
- Good scalability of O(log n)
- Fixed routing table size (for specific b and ID length)
- A lot of follow-up work bases on Pastry

Drawbacks

- Two metrics used during lookup
  - 1. Phase: Great hops toward target according to shared prefix
  - 2. Phase: Numerical distance used as metric
  - Both metrics can differ in result (cp. 09999 and 10000 – no common prefix, but numerically close)
  - Replication based on numerical metric

Robustness against Failures

HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

Coping with departures and failures

- Nodes leave unexpectedly (fail)
- For detection:
  - Periodic checks of table entries
  - Keep-alive messages
- If node does not answer: failed
  - Failure in Leaf Set:
    - Update entry with leaf set of furthest node
  - Failure in Routing Table:
    - Ask nodes in same row as failed node
    - If all in row failed: as nodes in higher row

FreePastry

Prototypical implementation of Pastry

- Current version 2.1: released on 13.3.2009
- Java based, Sun JDK version 1.5.0
- NodeID: 160 bits, 20 byte: 10 hexadecimal number

Replication (PAST)

- Applications like DHTs use replication to ensure that stored data survives node failure.
- To replicate a newly received key (K) r times
  the application calls replicaSet (k,r)
  and sends a copy of the key to each returned node
- If the implementation is not able to return r suitable neighbors, then the application itself is responsible for determining replica locations

More details:

- FreePastry documentation
- Replication: PAST

Properties of Pastry / FreePastry

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Advantages

Well documented, clear APIs

- Modular, extendable software
- Large user base, still maintained

Basic functionality

- Routing, DHT (key-value mapping)
- Distributed storage

Disadvantages

No support for heterogeneity

- All nodes are treated equally
- Strong, long-living peers should do more

No further built-in security mechanisms

- Besides resistance against DoS
- Sensitive to malicious nodes
- → However, some changes in LifeSocial

Limited API

- "Only DHT"
- Also requires sufficient replication, additional services

Key-based Routing Interface

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

Towards a Common API for Structured Peer-to-Peer Overlays

- Dabek, Zhao, Druschel (Pastry), Kubiatowicz, Stoica (Chord)
- Allows to exchange the used DHT!!!

Notation: → read only , ←→ read and write

Routing API

void route(key →K, msg →M, nodehandle →hint)

- K or hint may be NULL

void forward(key ←→K, msg ←→M, nodehandle ←→nextHop)

- Upcall at receiving node, that may read all parameters

deliver(key →K, msg →M)

- Delivers the message to the receiving application

KBR: Routing state access

HEINRICH HEINE UNIVERSITÄT DÜSSELDORF

nodehandle[] local lookup(key →K, int →num, boolean →safe)

- Returns a list of possible (num) hops for routing to (key K)

nodehandle [] neighborSet (int → num)

- Returns unordered list of (num) peers in the neighborhood list

nodehandle [] replicaSet (key → k, int → max rank)

- Returns an ordered set of peers of magnitude (max rank) on which replicas of the object with key k can be stored
- The nodes which become roots for the key k when the local node fails

update(nodehandle →n, bool →joined)

- Upcall: informs that node n has either joined or left the local neighbor set

boolean range (nodehandle →N, rank →r, key ←→lkey, key ←→rkey)

- Provides information about ranges of keys for which the node N is responsible
- Returns false if the range could not be determined, true otherwise
- Can only be used for nodes in neighbor set
- lkey and rkey are modified by the method → inclusive range