Peer-to-Peer Systems and Applications

Lecture 2: Distributed Hash Tables (1)

Chapter 7 and 8:

Part III: Structured
Peer-to-Peer Systems

*Original slides for this lecture provided by K. Wehrle, S. Götz, S. Rieche (University of Tübingen)

0. Lecture Overview

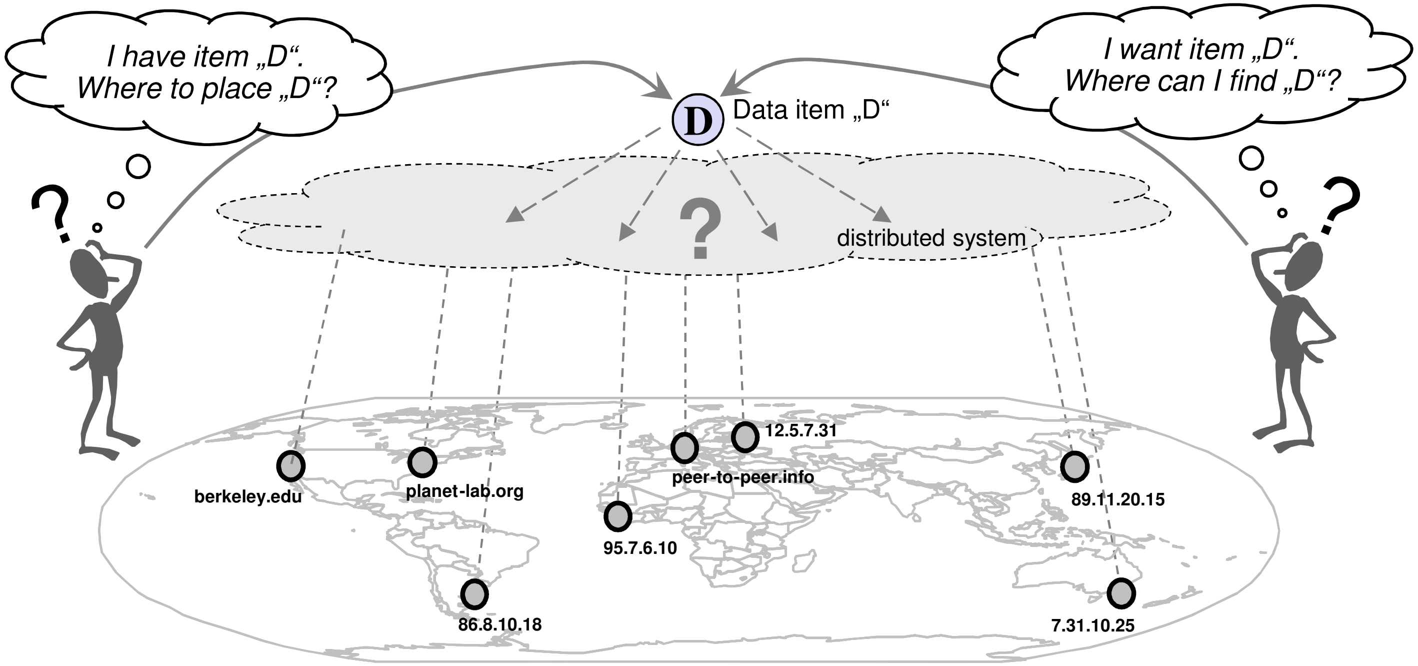
1. Distributed Management and Retrieval of Data
    1. Comparison of strategies for data retrieval
    2. Central server
    3. Flooding search
    4. Distributed indexing
    5. Comparison of lookup concepts
2. Fundamentals of Distributed Hash Tables
    1. Distributed management of data
    2. Addressing in Distributed Hash Tables
    3. Routing
    4. Data Storage
3. DHT Mechanisms
    1. Node Arrival
    2. Node Failure / Departure
4. DHT Interfaces
    1. Comparison: DHT vs. DNS
    2. Summary: Properties of DHTs
5. Selected DHT Algorithm: Pastry
    1. Identifier Space
    2. Routing Information
    3. Routing Procedure
    4. Node Addition and Failure
    5. Common API
    6. FreePastry Demo

1. Distributed Management and Retrieval of Data

Comparison of Strategies:
Central Server, Flooding, Distributed Indexing

1. Distributed Management and Retrieval of Data

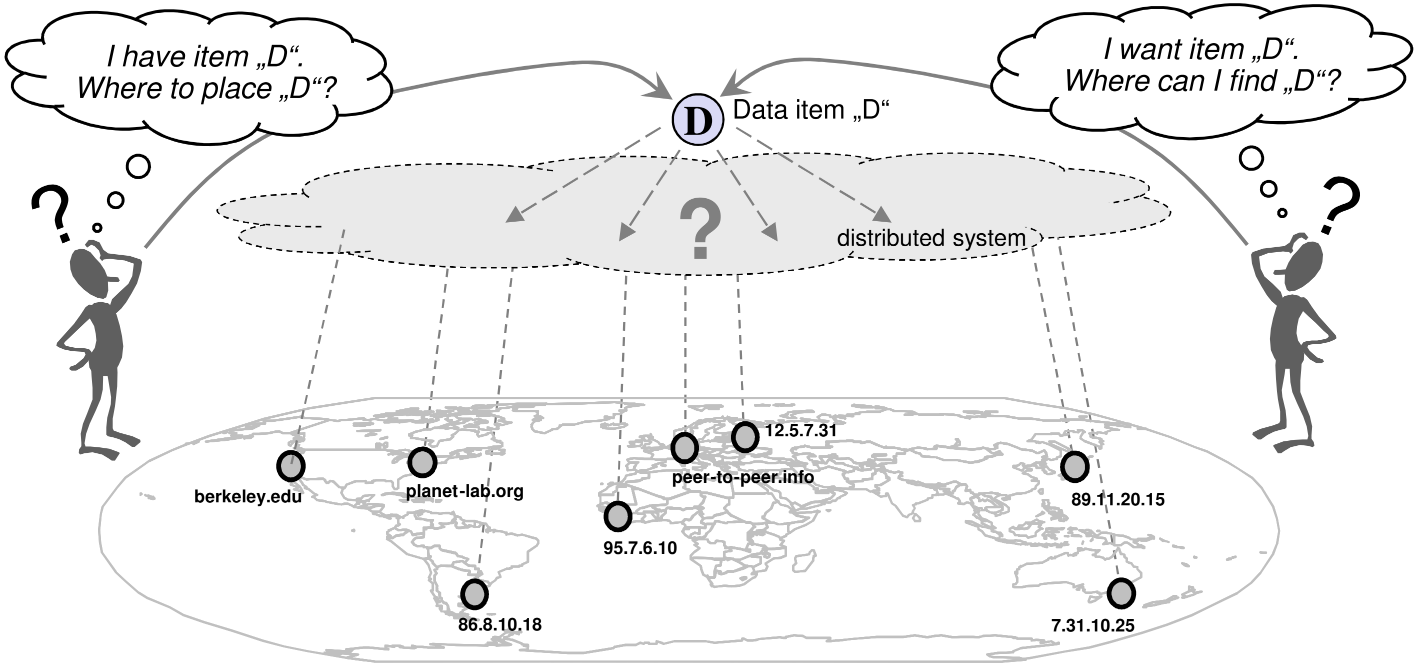
TECHNISCHE UNIVERSITÄT DARMSTADT



- Essential challenge in (most) Peer-to-Peer systems?
  - Location of a data item among systems distributed
    - Where shall the item be stored by the provider?
    - How does a requester find the actual location of an item?
  - Scalability: keep the complexity for communication and storage scalable
  - Robustness and resilience in case of faults and frequent changes

1.1. Comparison of Strategies for Data Retrieval

TECHNISCHE UNIVERSITÄT DARMSTADT

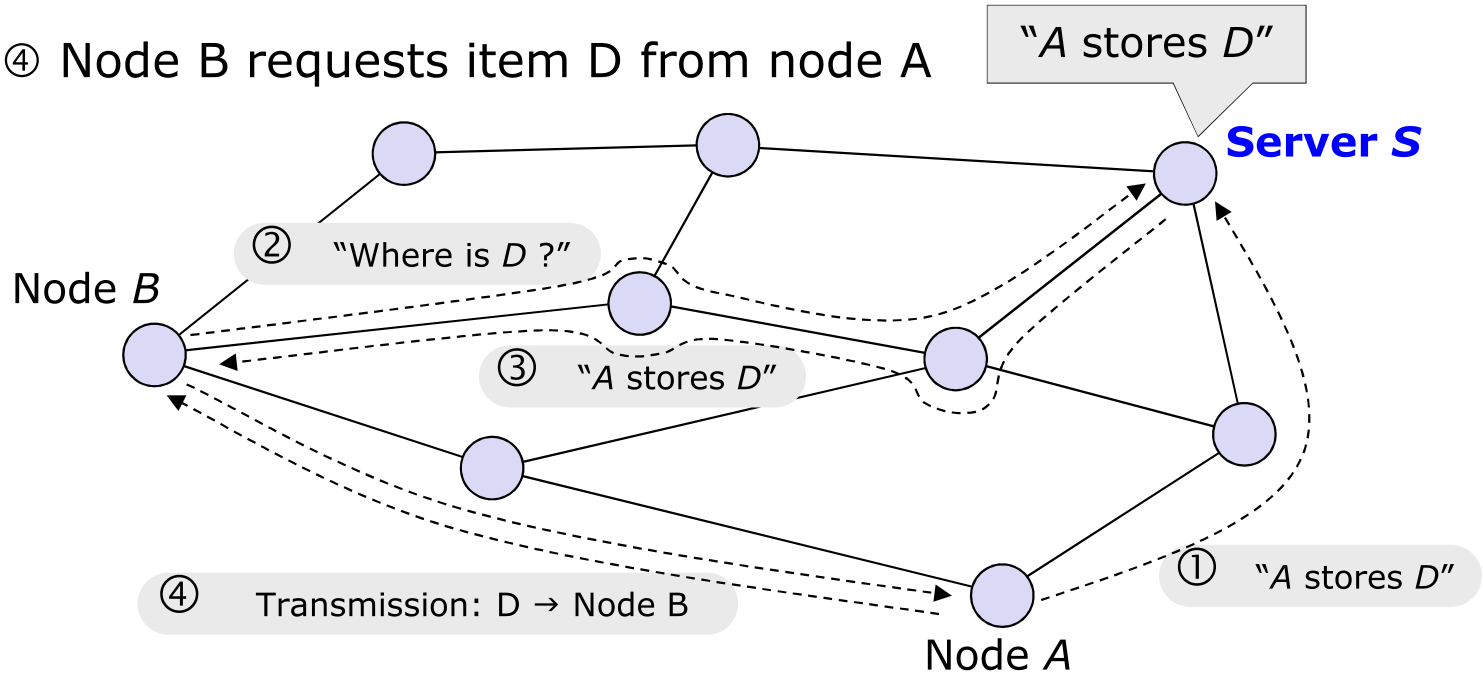


- Strategies to store and retrieve data items in distributed systems
  - Central server
  - Flooding search
  - Distributed indexing

1.2. Approach I: Central Server (1)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Simple strategy: Central Server
  - Server stores information about locations
    - ① Node A (provider) tells server that it stores item D
    - ② Node B (requester) asks server S for the location of D
    - ③ Server S tells B that node A stores item D
    - ④ Node B requests item D from node A



1.2. Approach I: Central Server (2)

- Advantages
  - Search complexity of O(1) – *"just ask the server"*
  - Complex and fuzzy queries are possible
  - Simple and fast
- Problems
  - No Scalability
    - O(N) node state in server
    - O(N) network and system load of server
  - Single point of failure or attack (also for law suites ;-)
  - Non-linear increasing implementation and maintenance cost
    (in particular for achieving high availability and scalability)
  - Central server not suitable for systems with massive numbers of users
- But overall, …
  - Best principle for small and simple applications!

1.3. Approach II: Flooding Search (1)

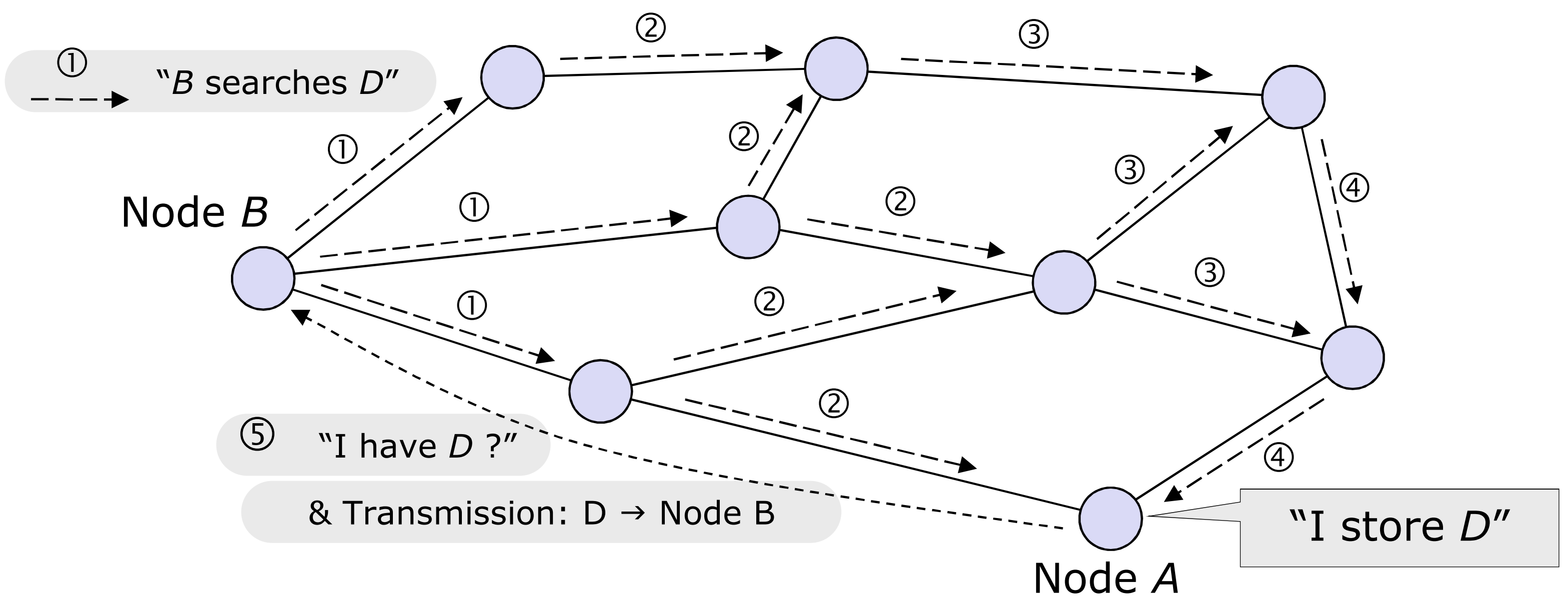
TECHNISCHE UNIVERSITÄT DARMSTADT

- Fully Distributed Approach
  - Central systems are vulnerable and do not scale
  - Unstructured Peer-to-Peer systems follow opposite approach
  - No information on location of a content
  - Content is only stored in the node providing it

- Retrieval of data
  - No routing information for content
  - Necessity to ask as much systems as possible / necessary
  - Approaches
    - Flooding: high traffic load on network, does not scale
    - Highest degree search: quick search through large areas – large number of messages needed for unique identification

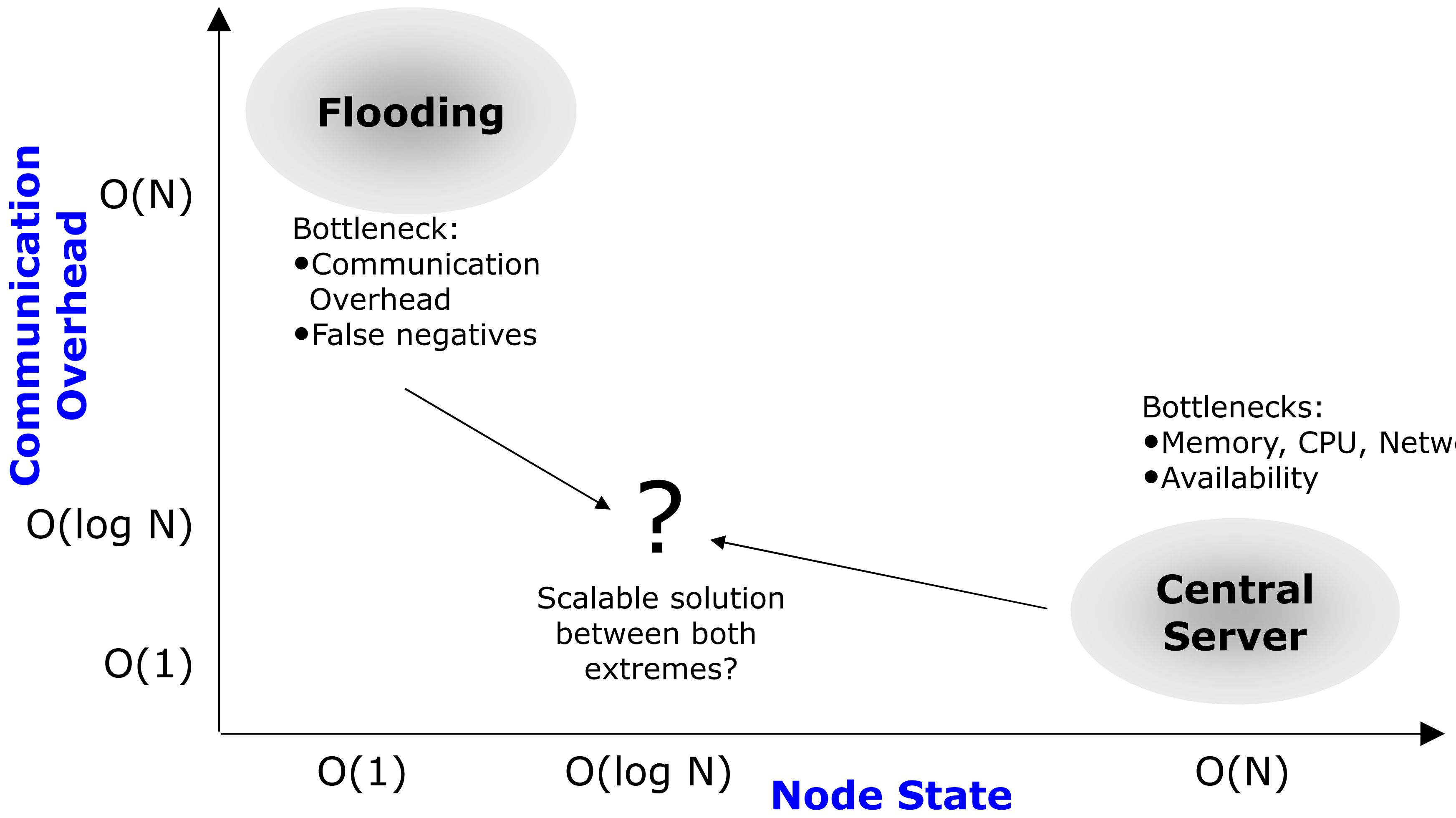
1.3. Approach II: Flooding Search (2)

- Fully Decentralized Approach: Flooding Search
  - No information about location of data in the intermediate systems
  - Necessity for broad search
    - ① Node B (requester) asks neighboring nodes for item D
    - ②-④ Nodes forward request to further nodes (breadth-first search / flooding)
    - ② Node A (provider of item D) sends D to requesting node B



1.4. Motivation Distributed Indexing (1)

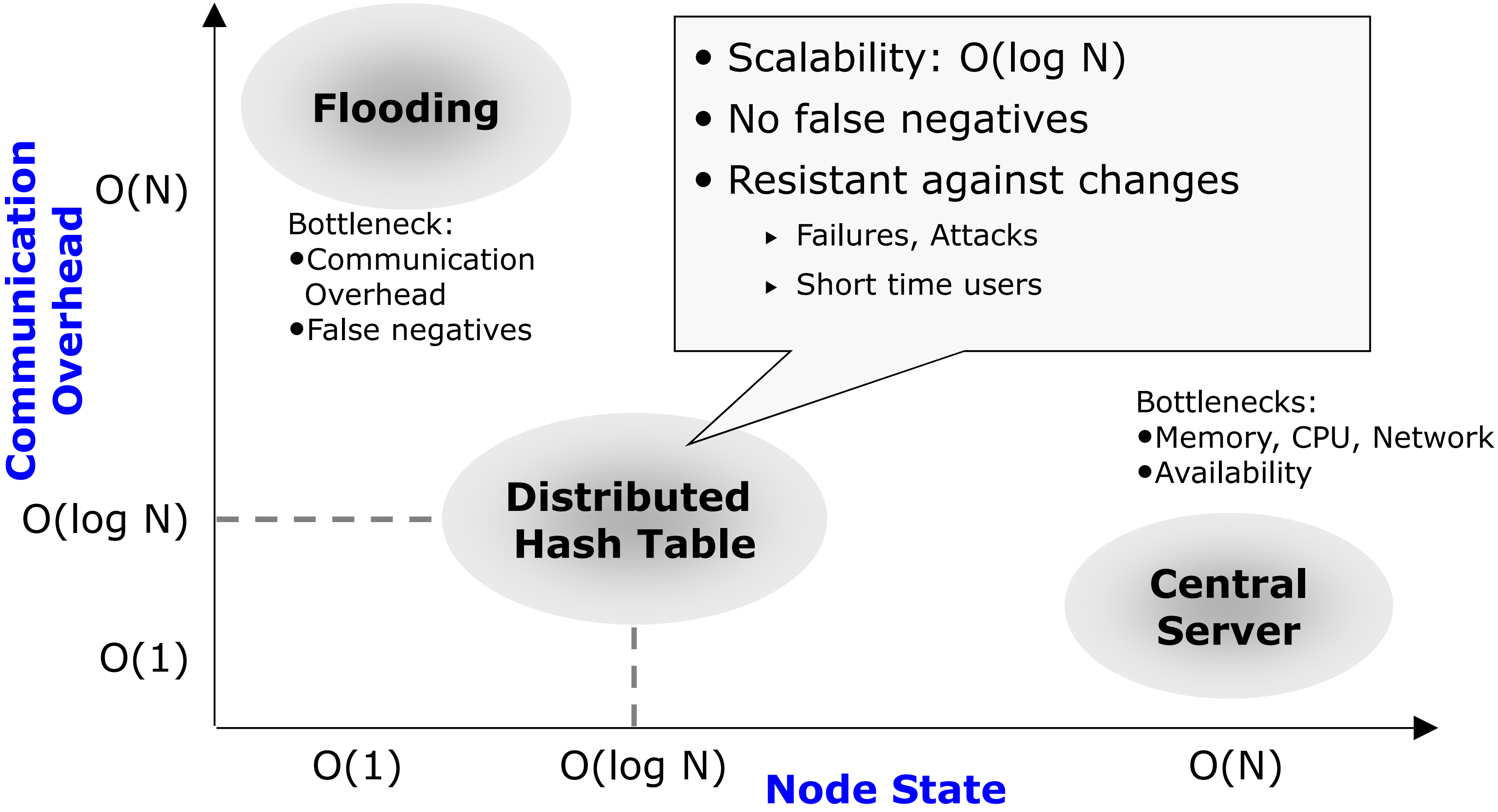
- Communication overhead vs. node state



1.4. Motivation Distributed Indexing (2)

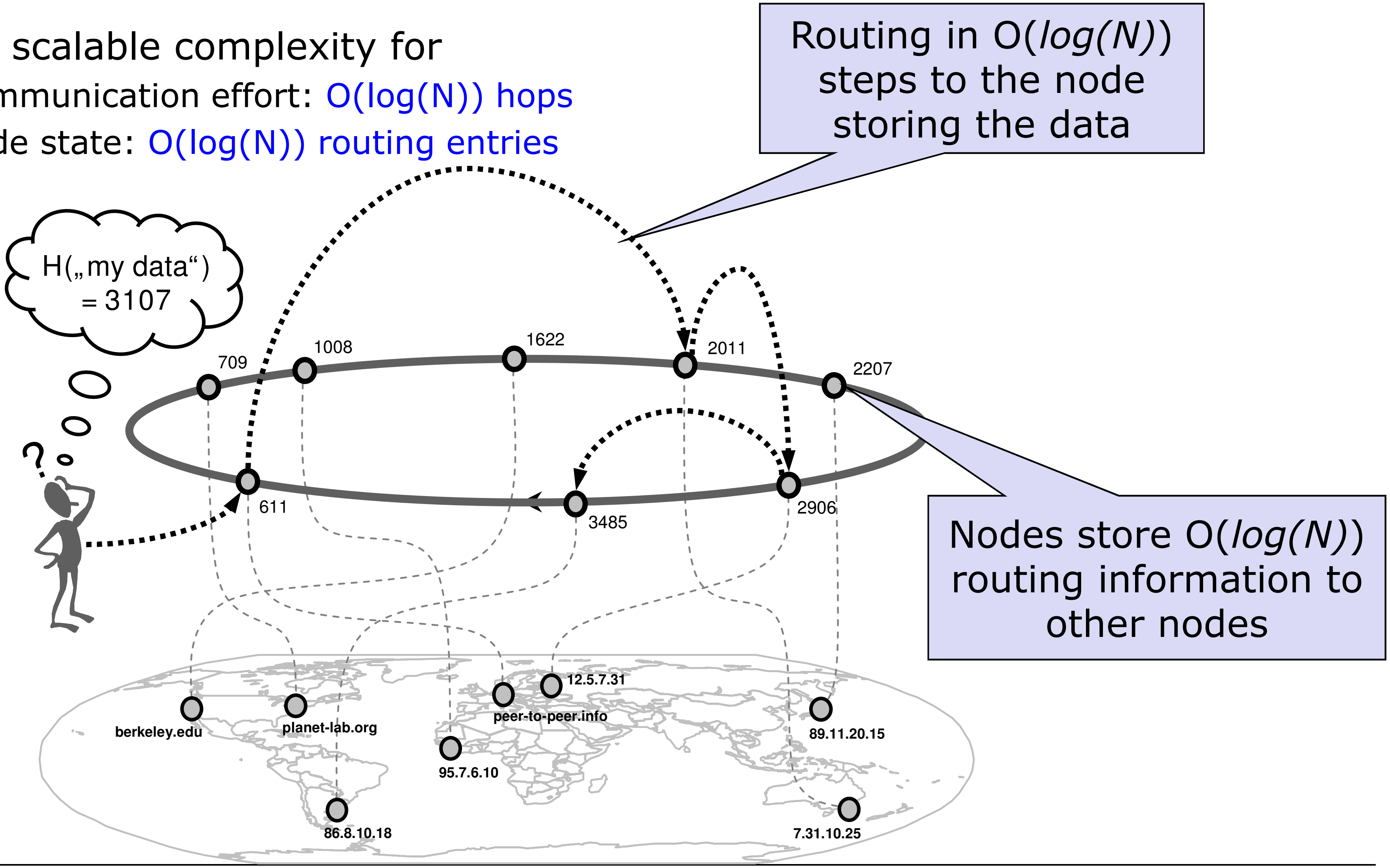
TECHNISCHE UNIVERSITÄT DARMSTADT

- Communication overhead vs. node state



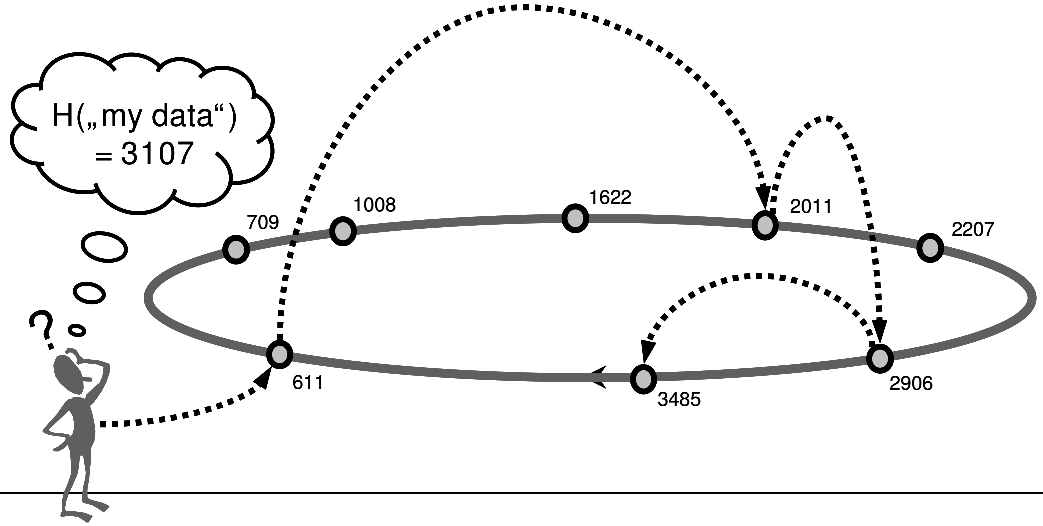
1.4. Distributed Indexing (1)

- Goal is scalable complexity for
  - Communication effort: O(log(N)) hops
  - Node state: O(log(N)) routing entries



1.4. Distributed Indexing (2)

- Approach of distributed indexing schemes
  - Data and nodes are mapped into same address space
  - Intermediate nodes maintain routing information to target nodes
    - Efficient forwarding to „destination“ (content – not location)
    - Definitive statement of existence of content
- Problems
  - Maintenance of routing information required
  - Fuzzy queries not primarily supported (e.g, wildcard searches)



1.5. Comparison of Lookup Concepts

System	Per Node State	Communi-cation Overhead	Fuzzy Queries	No false negatives	Robust-ness
Central Server	O(N)	O(1)	✓	✓	✗
Flooding Search	O(1)	O(N²)	✓	✗	✓
Distributed Hash Tables	O(log N)	O(log N)	✗	✓	✓

2. Fundamentals of Distributed Hash Tables

Distributed Data Management
Addressing, Routing, Data Storage

2. Fundamentals of Distributed Hash Tables

- Challenges for designing DHTs
  - Desired Characteristics
    - Flexibility
    - Reliability
    - Scalability
  - Equal distribution of content among nodes
    - Crucial for efficient lookup of content
  - Permanent adaptation to faults, joins, exits of nodes
    - Assignment of responsibilities to new nodes
    - Re-assignment and re-distribution of responsibilities in case of node failure or departure

2.1. Distributed Management of Data

Sequence of operations

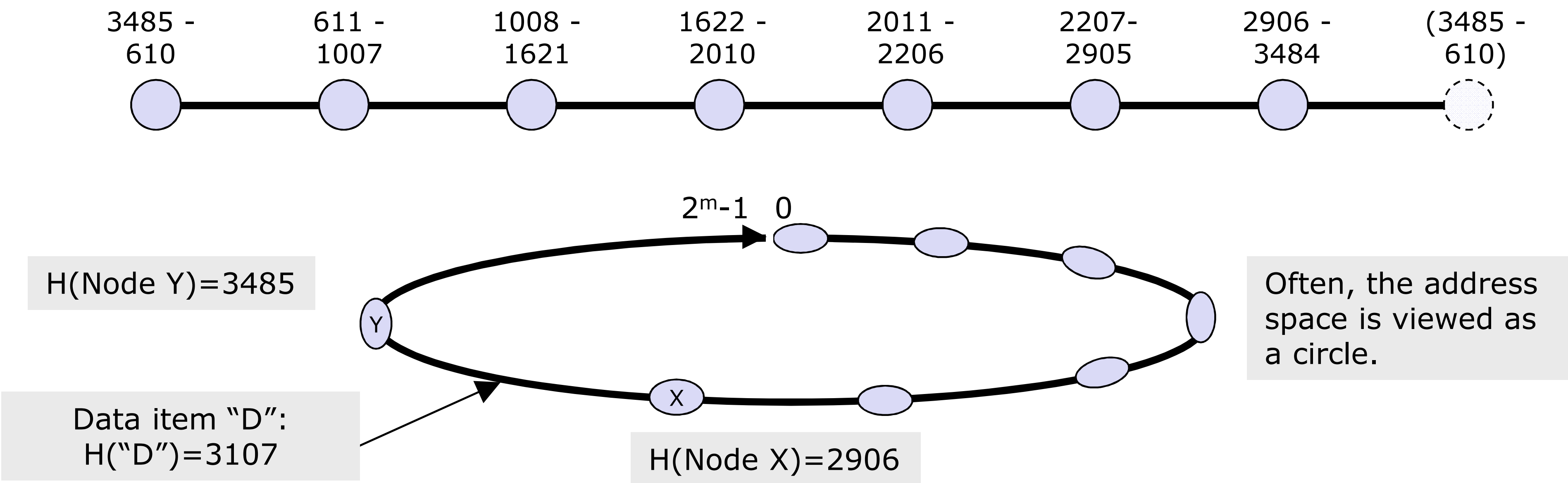
1. Mapping of nodes and data into same address space
   - Peers and content are addressed using flat identifiers (IDs)
   - Common address space for data and nodes
   - Nodes are responsible for data in certain parts of the address space
   - Association of data to nodes may change since nodes may disappear

2. Storing / Looking up data in the DHT
   - Search for data = routing to the responsible node
     - Responsible node not necessarily known in advance
     - Deterministic statement about availability of data

2.2. Addressing in Distributed Hash Tables

TECHNISCHE UNIVERSITÄT DARMSTADT

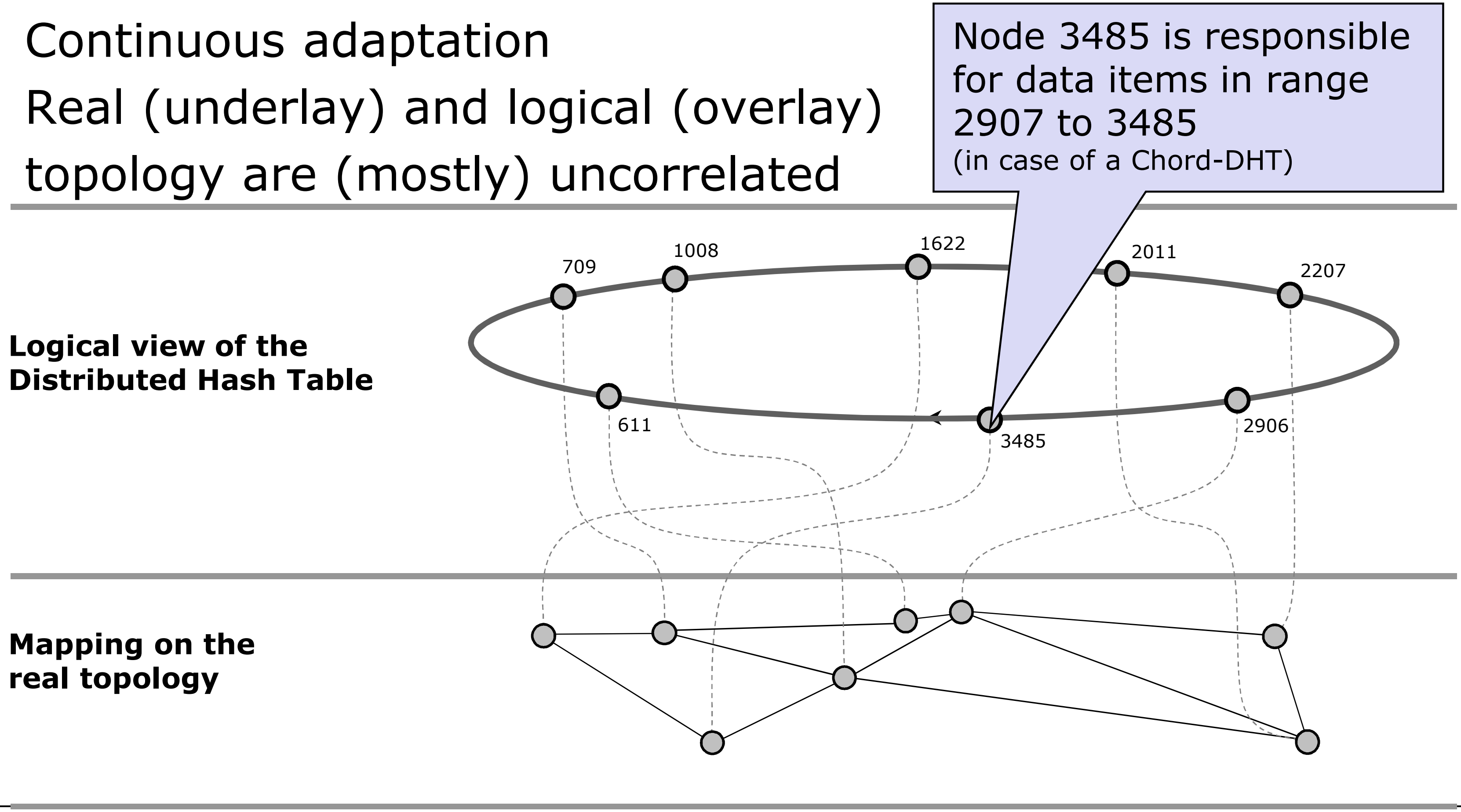
- Step 1: Mapping of content/nodes into linear space
  - Usually: 0, …, 2^m-1 >> number of objects to be stored
  - Mapping of data and nodes into an address space (with hash function)
    - E.g., Hash(*String*) mod 2^m: H(„*my data*") → 2313
  - Association of parts of address space to DHT nodes



2.2. Association of Address Space with Nodes

TECHNISCHE UNIVERSITÄT DARMSTADT

- 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



2.3. Step 2: Routing to a Data Item (1)

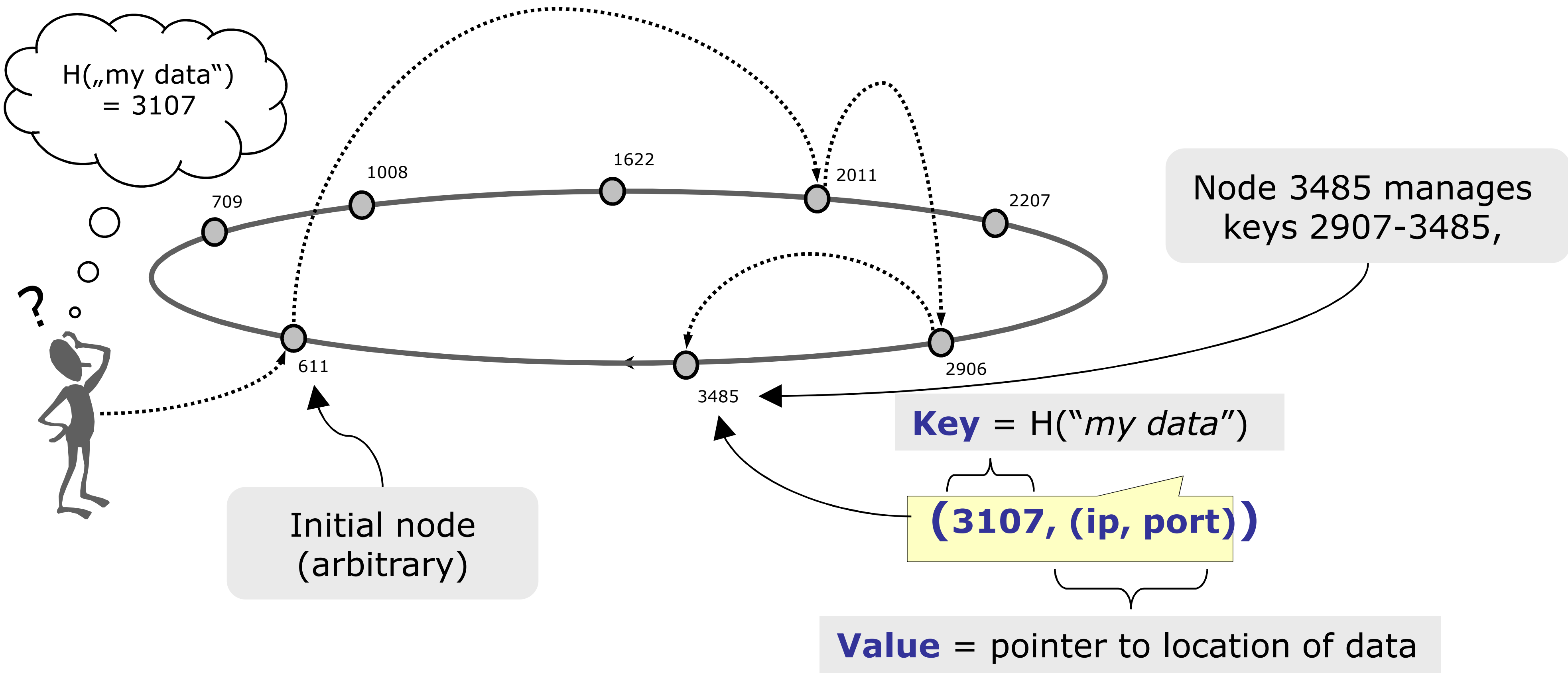
TECHNISCHE UNIVERSITÄT DARMSTADT

- Step 2:
  Locating the data (content-based routing)

- Goal: Small and scalable effort
  - O(1) with centralized hash table
    - But:
      Management of a centralized hash table is very 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 = # nodes)

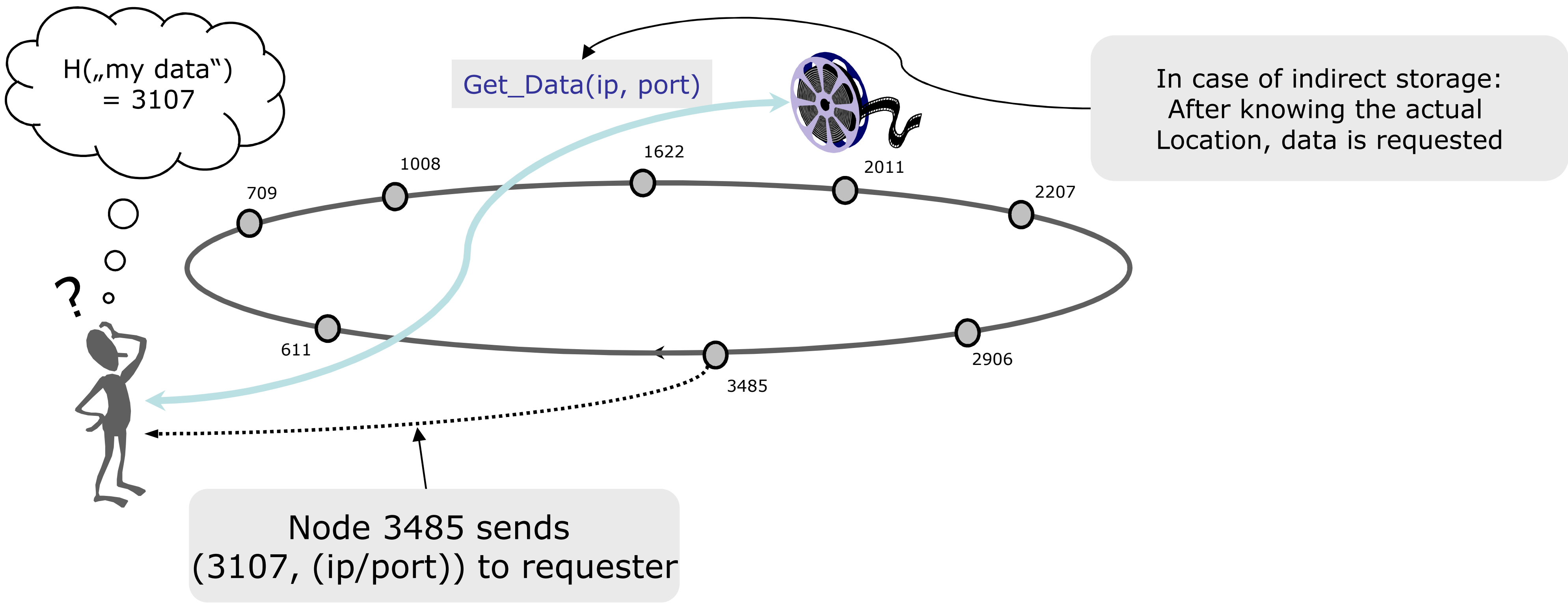
2.3. Step 2: Routing to a Data Item (2)

- Routing to a K/V-pair
  - Start lookup at arbitrary node of DHT
  - Routing to requested data item (key)



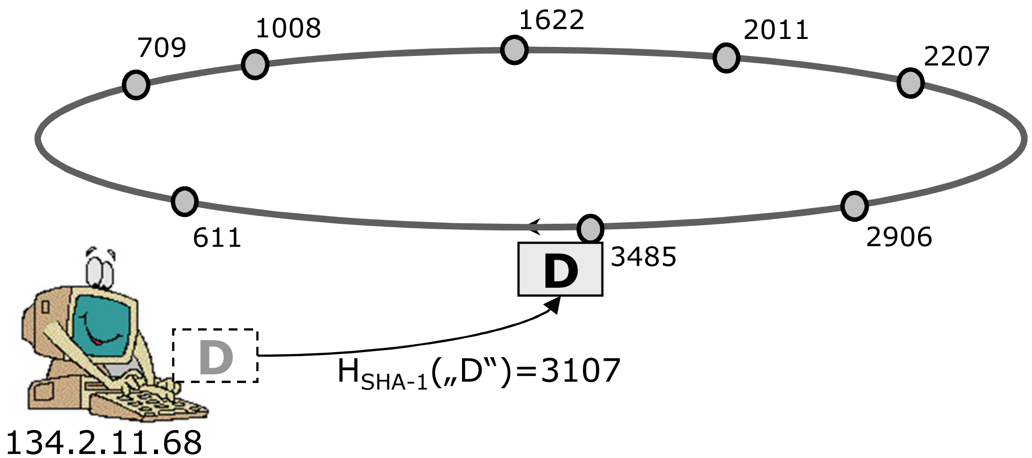
2.3. Step 2: Routing to a Data Item (3)

- Getting the content
  - K/V-pair is delivered to requester
  - Requester analyzes K/V-tuple
    (and downloads data from actual location – in case of indirect storage)



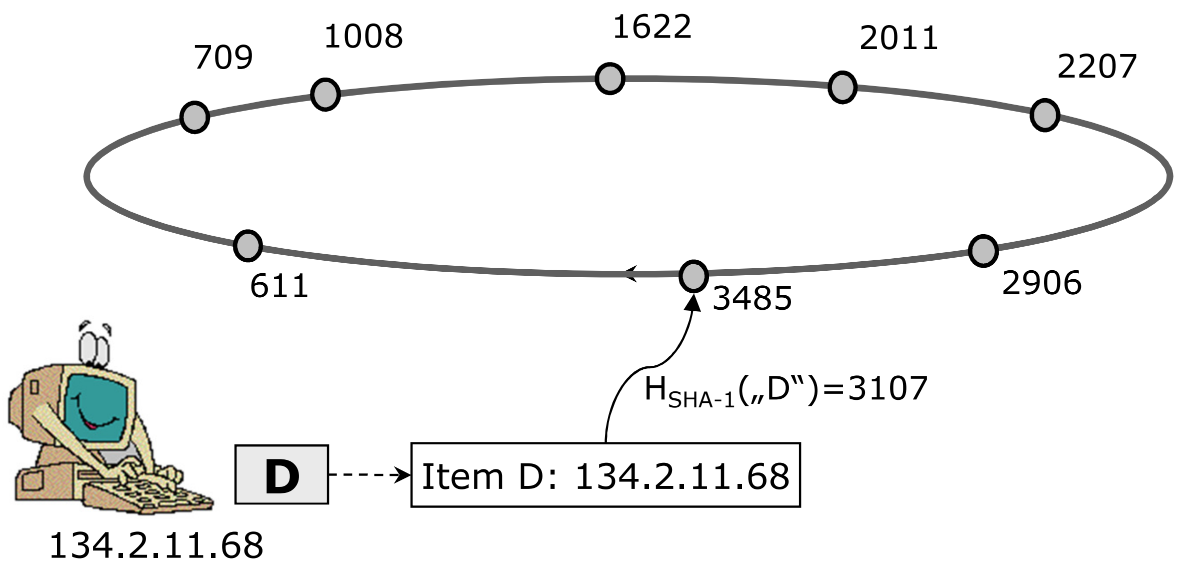
2.4. Association of Data with IDs – Direct Storage

- How is content stored on the nodes?
  - Example:
    H("*my data*") = 3107 is mapped into DHT address space
- Direct storage
  - Content is stored in responsible node for H("*my data*")
    → Inflexible for large content – o.k., if small amount data (<1KB)



2.4. 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

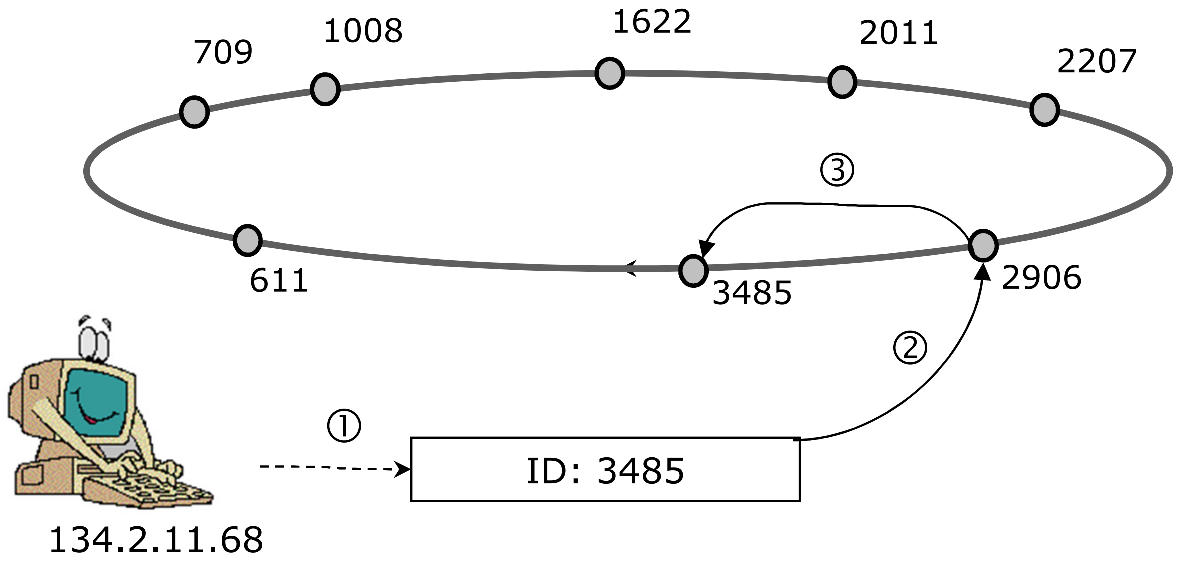


3. DHT Mechanisms

Node Arrival
Node Failure / Departure

3.1. Node Arrival

- Joining 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 K/V-pairs of hash range (usually with redundancy)
  5. Binding into routing environment



3.2. Node Failure / Departure

TECHNISCHE UNIVERSITÄT DARMSTADT

- Failure of a node
  - Use of redundant K/V 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 K/V pairs to corresponding nodes
  - Unbinding from routing environment

4. DHT Interfaces

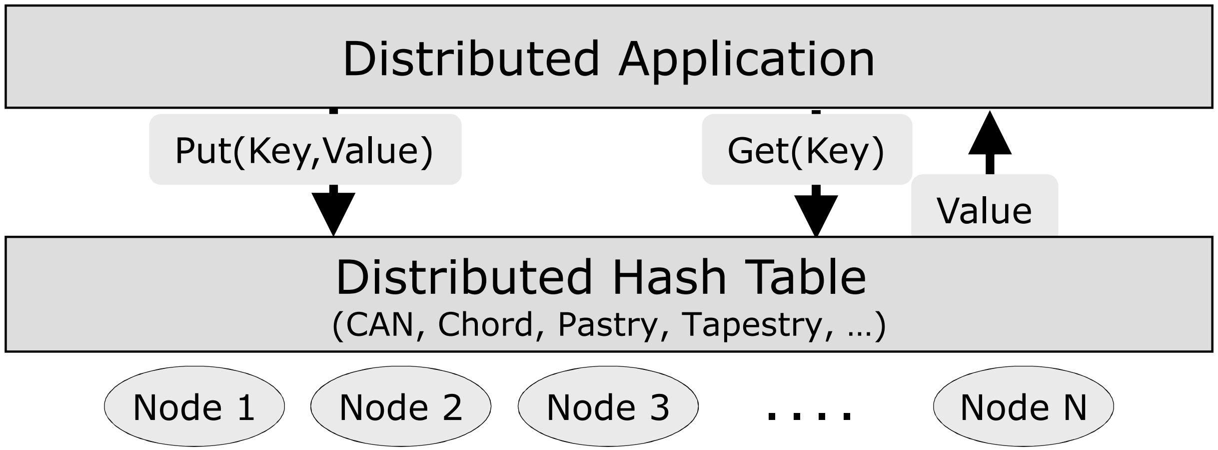
Comparison: DHT vs. DNS

Summary: Properties of DHTs

4. DHT Interfaces

TECHNISCHE UNIVERSITÄT DARMSTADT

- Generic interface of distributed hash tables
  - Provisioning of information
    - Publish(key,value)
  - Requesting of information (search for content)
    - Lookup(key)
  - Reply
    - value
- DHT approaches are interchangeable (with respect to interface)



4.1. Comparison: DHT vs. DNS (1)

- Comparison DHT vs. DNS
  - Traditional name services follow fixed mapping
    - DNS maps a logical node name to an IP address
  - DHTs offer flat / generic mapping of addresses
    - Not bound to particular applications or services
    - „*value*“ in *(key, value)* may be
      - an address
      - a document
      - or other data …

4.1. Comparison: DHT vs. DNS (2)

TECHNISCHE UNIVERSITÄT DARMSTADT

Domain Name System

- Mapping:
  Symbolic name → IP address
- Is built on a hierarchical structure with root servers
- Names refer to administrative domains
- Specialized to search for computer names and services

Distributed Hash Table

- Mapping: key → value can easily realize DNS
- Does not need a special server
- Does not require special name space
- Can find data that are independently located of computers

4.2. Summary: Properties of DHTs

- Use of routing information for efficient search for content
- Keys are evenly distributed across nodes of DHT
  - No bottlenecks
  - A continuous increase in number of stored keys is admissible
  - Failure of nodes can be tolerated
  - Survival of attacks possible
- Self-organizing system
- Simple and efficient realization
- Supporting a wide spectrum of applications
  - Flat (hash) key without semantic meaning
  - Value depends on application

Next …

- Specific examples of Distributed Hash Tables
  - Pastry
    Microsoft Research, Rice University
  - Chord
    UC Berkeley, MIT
  - Tapestry
    UC Berkeley
  - CAN
    UC Berkeley, ICSI
  - P-Grid
    EPFL Lausanne

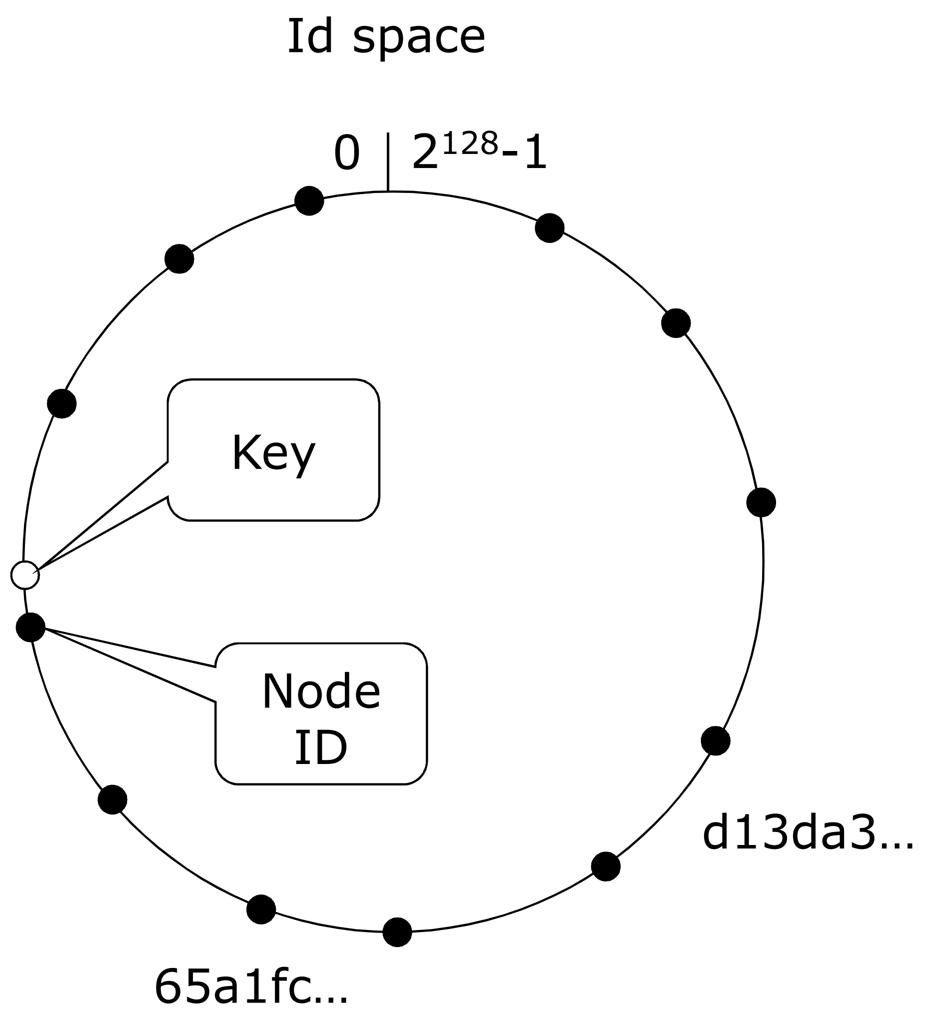
- … and there are plenty of others: Kademlia, Symphony, Viceroy, …

5. Selected DHT Algorithm: Pastry

Identifier Space, Routing Information, Routing Procedure, **Node Addition and Failure,** Common API, FreePastry Demo

5.1. Identifier Space

- Each node and data item has unique l-bit identifier
  - l typically 128
  - Nodes: Node ID
  - Data items: Key
  - Can be calculated from IP address or public key, and data item using secure hash function
- Keys are located on the node whose node ID is numerically closest to the key
- Pastry identifiers are strings of digits to the base 2^b
  - b typically 4
  - E.g. 65a1fc...



5.2. Routing Information

- Leaf Set L
  - Nodes close in the ID space
  - |L|/2 numerically closest larger nodeIds, and |L|/2 numerically closest smaller nodeIds
  - |L| typically 16
- Routing Table R
  - l/b rows with 2^b - 1 entries
  - Row n: Nodes that share an n-digit prefix, but whose n+1th digit is different
  - On average only $\log_{2^b}(N)$ rows are populated
- Neighborhood Set M
  - Nodes close in network locality

NodeId 10233102

Leaf set

SMALLER		LARGER	
10233033	10233021	10233120	10233122
10233001	10233000	10233230	10233232

Routing table

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

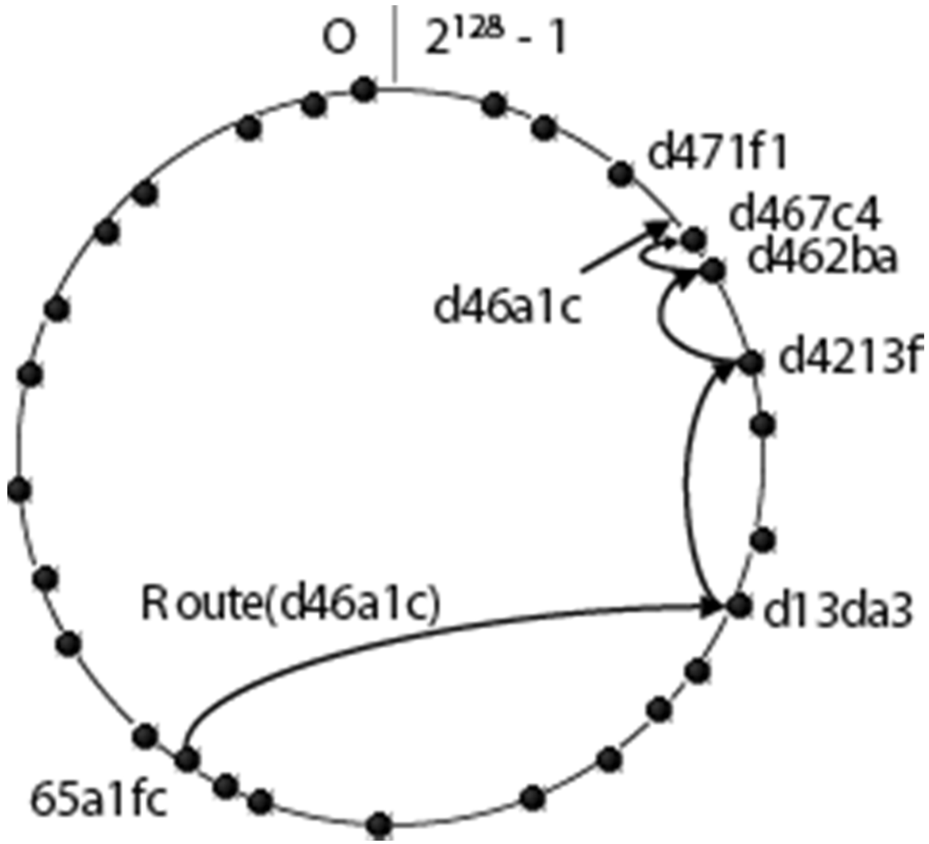
Neighborhood set

13021022	10200230	11301233	31301233
02212102	22301203	31203203	33213321

Pastry node with nodeId 10233102
b = 2, *l* = 16, |L| = 8

5.3. Routing Procedure

- Step 1
  - Forward message to a node who shares with the key a *prefix* that is at least *one digit (b bits) longer* than the prefix that the key shares with the current node
  - If no such nodes exists, forward message to a node who is numerically closer to the key
- Step 2
  - Forward message to a node in the leafset who is numerically closest to the key



Routing of a message from node 65a1fc with key d46a1c

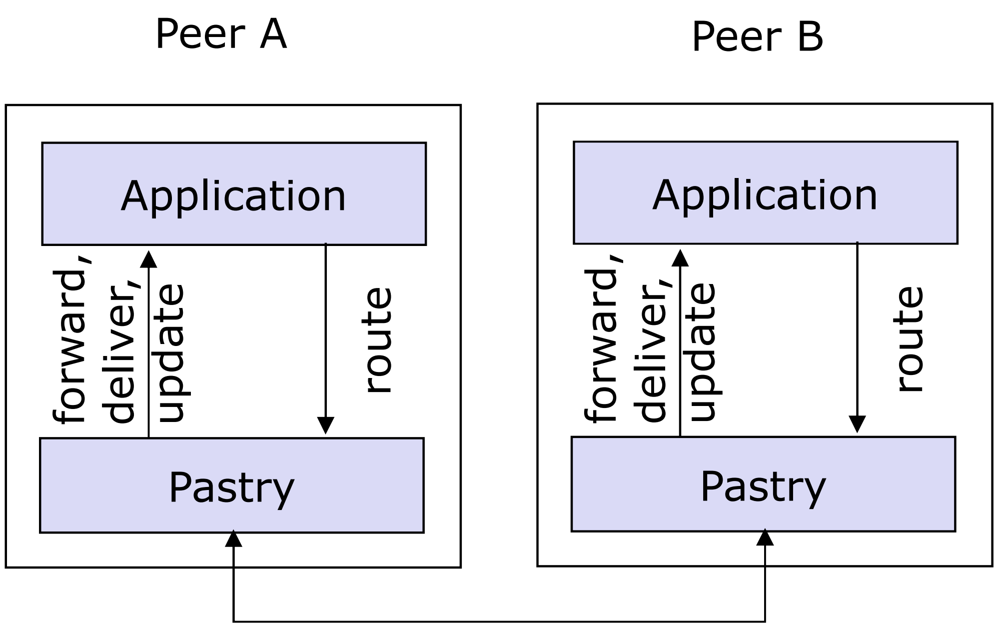
5.4. Node Addition and Failure

TECHNISCHE UNIVERSITÄT DARMSTADT

- Node state (routing information) has to be maintained efficiently
- Node Arrivals
  - New node with nodeId X asks nearby node A (bootstrap node) to route a special message to key X
  - Message is routed to node Z, X obtains leaf set from Z and i-th row of routing table from i-th node along the route from A to Z
- Node Failures
  - Leaf set nodes periodically exchange keepalive messages
  - If a node does not respond for a time T, it is declared dead
  - Members of the leaf set are notified and update their leaf set

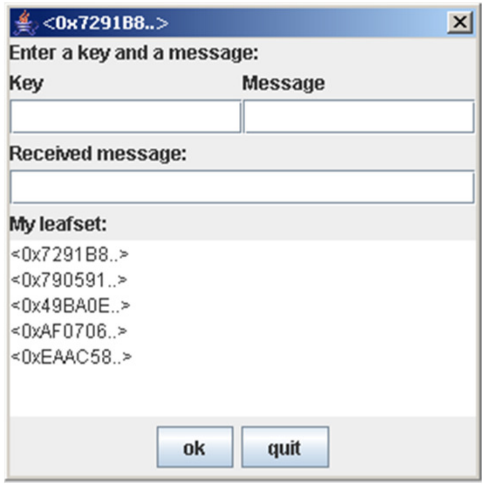
5.5. Common API for Structured P2P Overlays

- Standardized interface between applications and overlays
  - Implemented by Pastry, others implement it too (Chord, Tapestry, CAN)
- forward(RouteMessage)
  - Called just before a message is forwarded
  - Message could be changed or dropped
- deliver(Id, Message)
  - Called when a message is received
- update(NodeHandle, boolean)
  - Called when a node‘s leafset changes (node joined or left)
- route(Id, Message, NodeHandle)
  - Send a message to a node numerically closest to an Id (key)
  - NodeHandle can serve as a hint

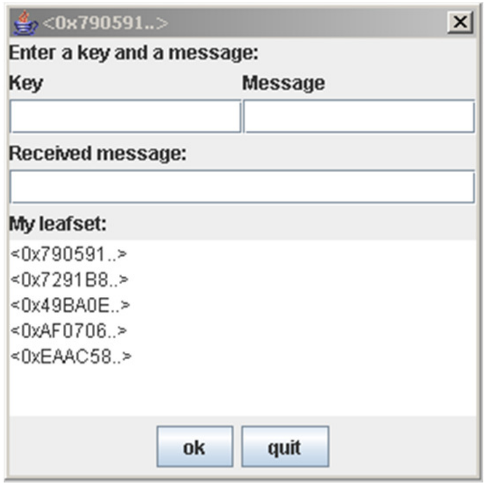


5.6. FreePastry Demo

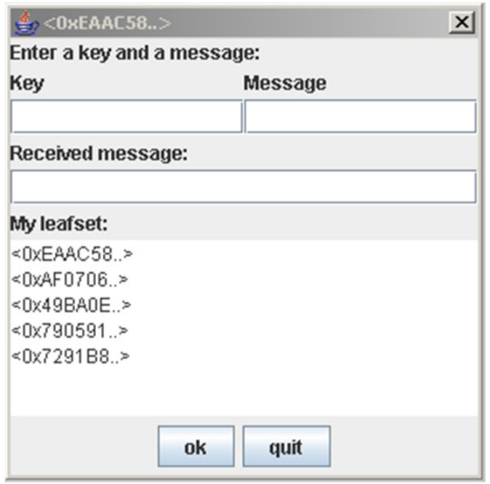
<0x7291B8..>



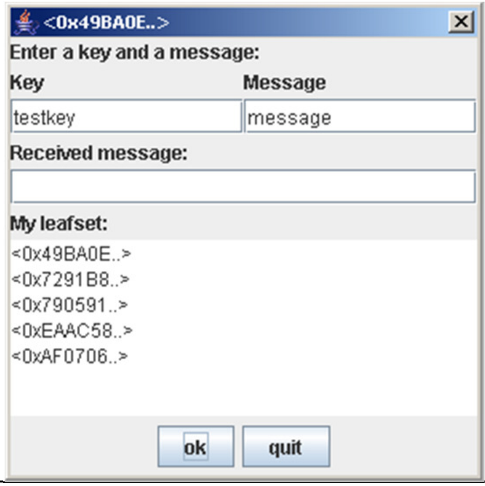
<0x790591..>



<0xEAAC58..>



<0x49BA0E..>



Key: testkey
Message: message

Key maps to
Id <B6B809..>

<0xAF0706..>