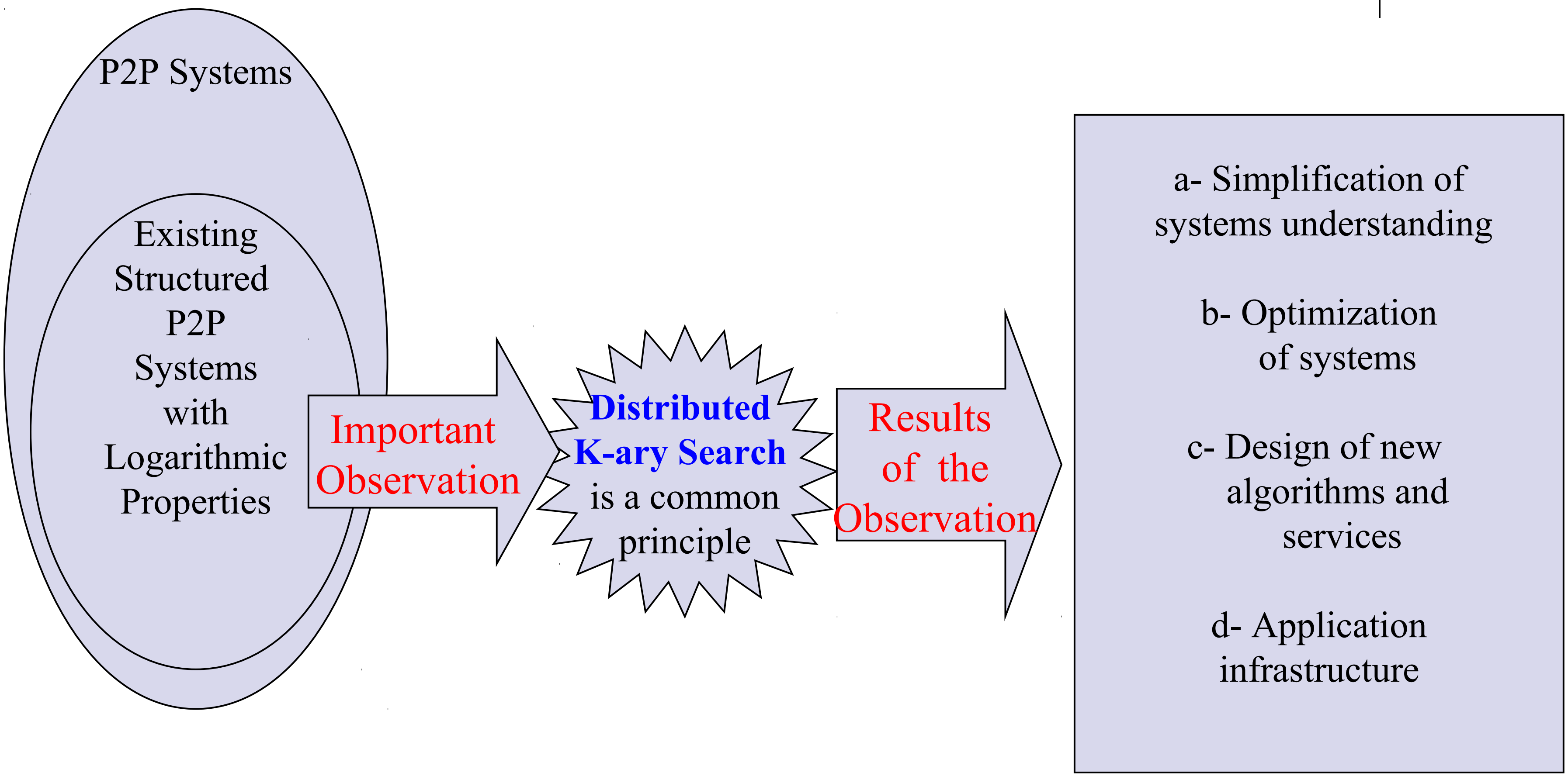
Structured Peer-to-Peer Systems for Distributed Applications

THE ADVENTURES OF SELFMAN

www.ist-selfman.org

Seif Haridi (SICS/KTH)
Sameh El-Ansary (SICS)
Ali Ghodsi (KTH)
Luc Onana Alima (SICS/KTH)
Per Brand (SICS)
Thorsten Schütt (ZIB)
Alexander Reinefeld (ZIB)
Peter Van Roy (UCL)
Boris Mejias (UCL)

The Talk in One Slide



Outline

- Overview of P2P systems
  - Three generations of P2P
- Distributed Hash Tables (DHTs)
- DKS: A Realistic DHT
- Broadcast service in DKS
- DHT as application infrastructure
  - Scalaris and Beernet libraries
  - DeTransDraw application on gPhone
  - Distributed Wikipedia application

Overview of P2P systems

What is Peer-To-Peer Computing? (1/3)

- **A. Oram ("Peer-to-Peer: Harnessing the Power of Disruptive Technologies")**
  P2P is a class of applications that:
  - Takes advantage of resources – (storage, CPU, etc,..) – available at the edges of the Internet.
  - Because accessing these decentralized resources means operating in an environment of unstable connectivity and unpredictable IP addresses, P2P nodes must operate outside the DNS system and have significant or total autonomy from central servers.

What is Peer-To-Peer Computing? (2/3)

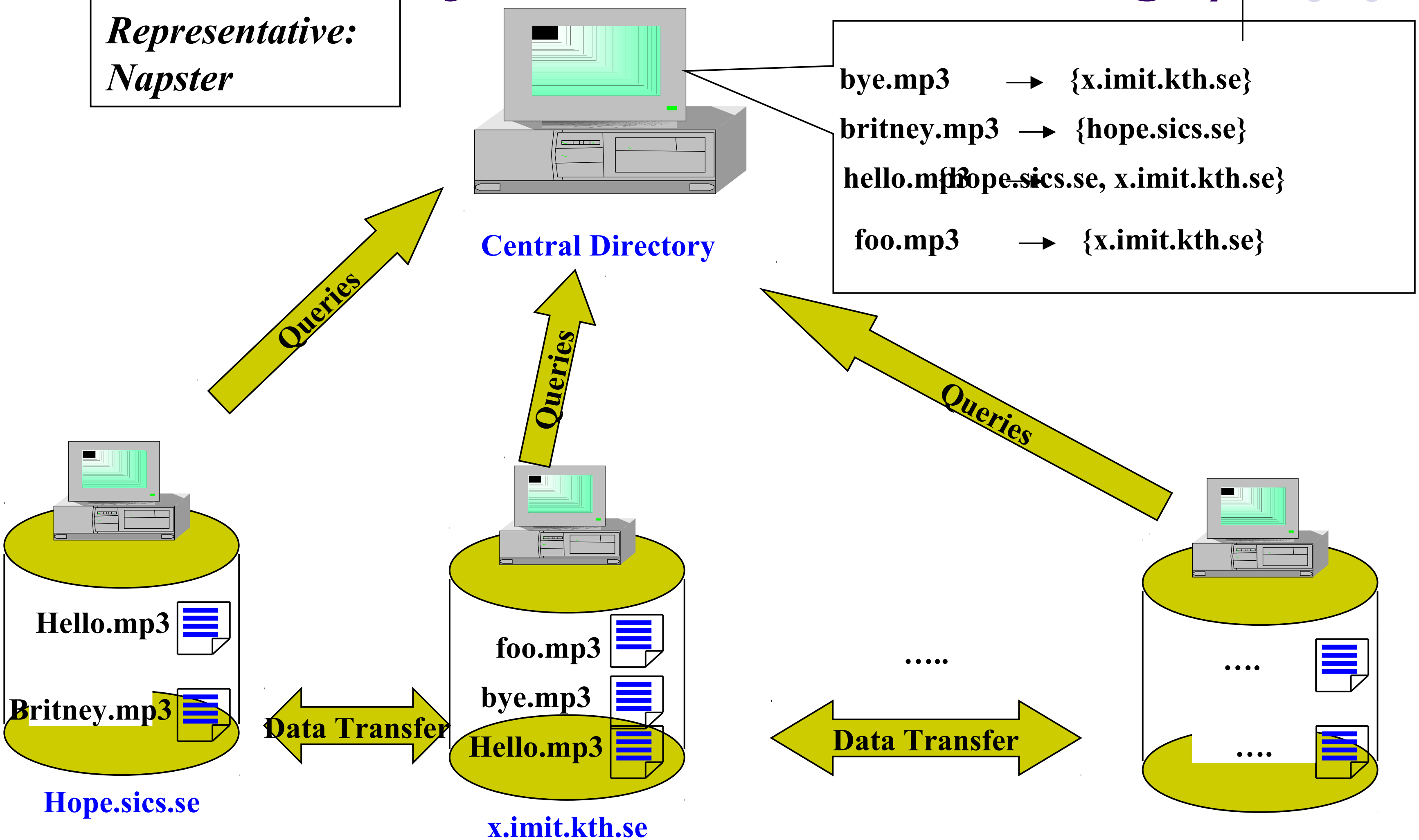
- **P2P Working Group (A Standardization Effort)** P2P computing is:
  - The sharing of computer resources and services by direct exchange between systems.
  - Peer-to-peer computing takes advantage of existing computing power and networking connectivity, allowing economical clients to leverage their collective power to benefit the entire enterprise.

What is Peer-To-Peer Computing? (3/3)

- **Our view**
  P2P computing is distributed computing with the following desirable properties:
  - Resource sharing
  - Dual client/server role (same role for all nodes)
  - Decentralization/autonomy
  - Scalability
  - Robustness/self-organization

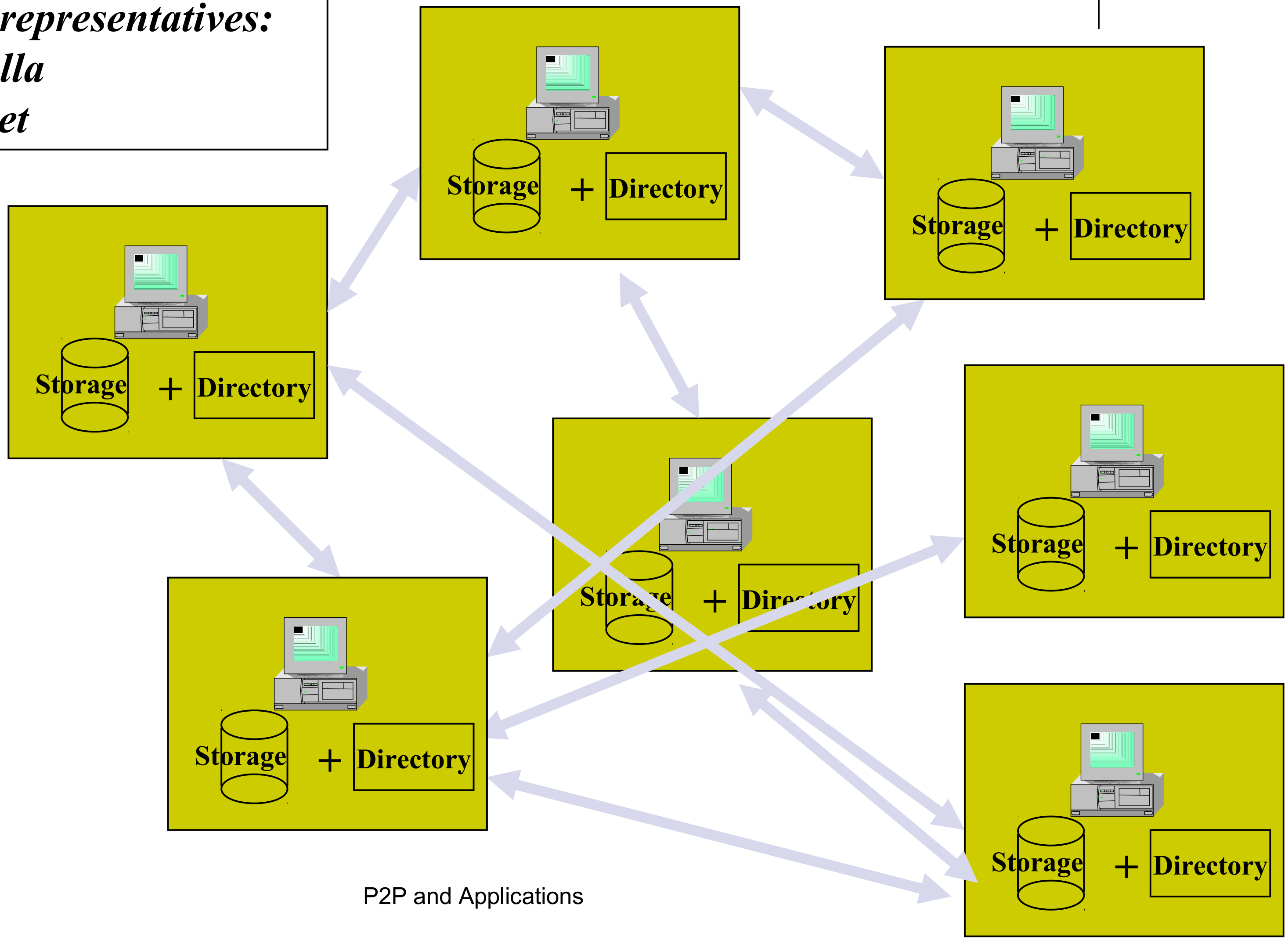
Evolution of P2P: 1st Generation (Central Directory + Distributed Storage)

Representative: Napster



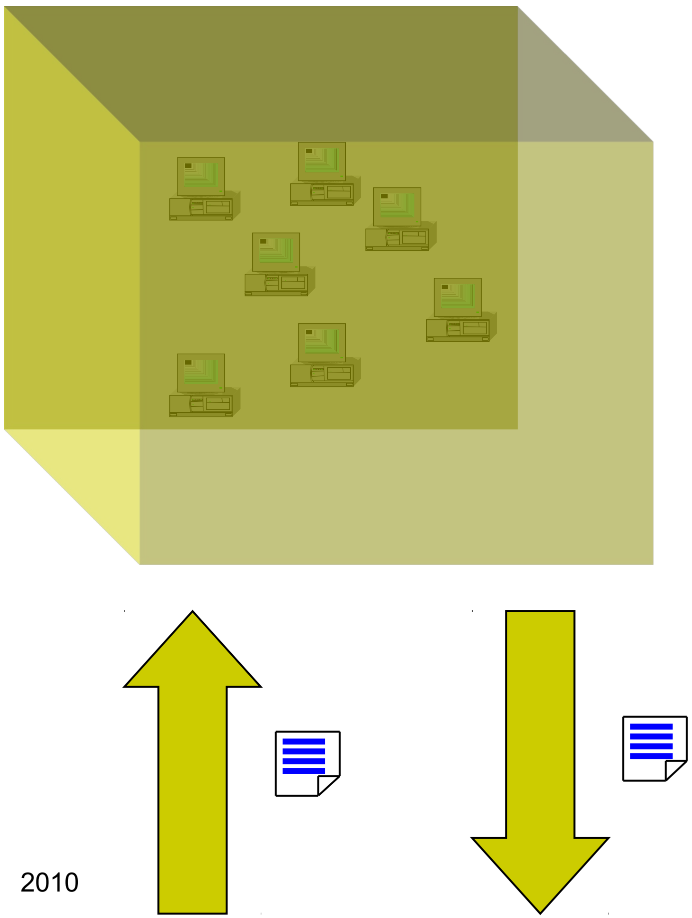
Evolution of P2P: 2nd Generation (Random Overlay Networks)

Some representatives:
Gnutella
Freenet



Evolution of P2P - 3rd Generation (Structured Overlay Networks / DHTs) (1/2)

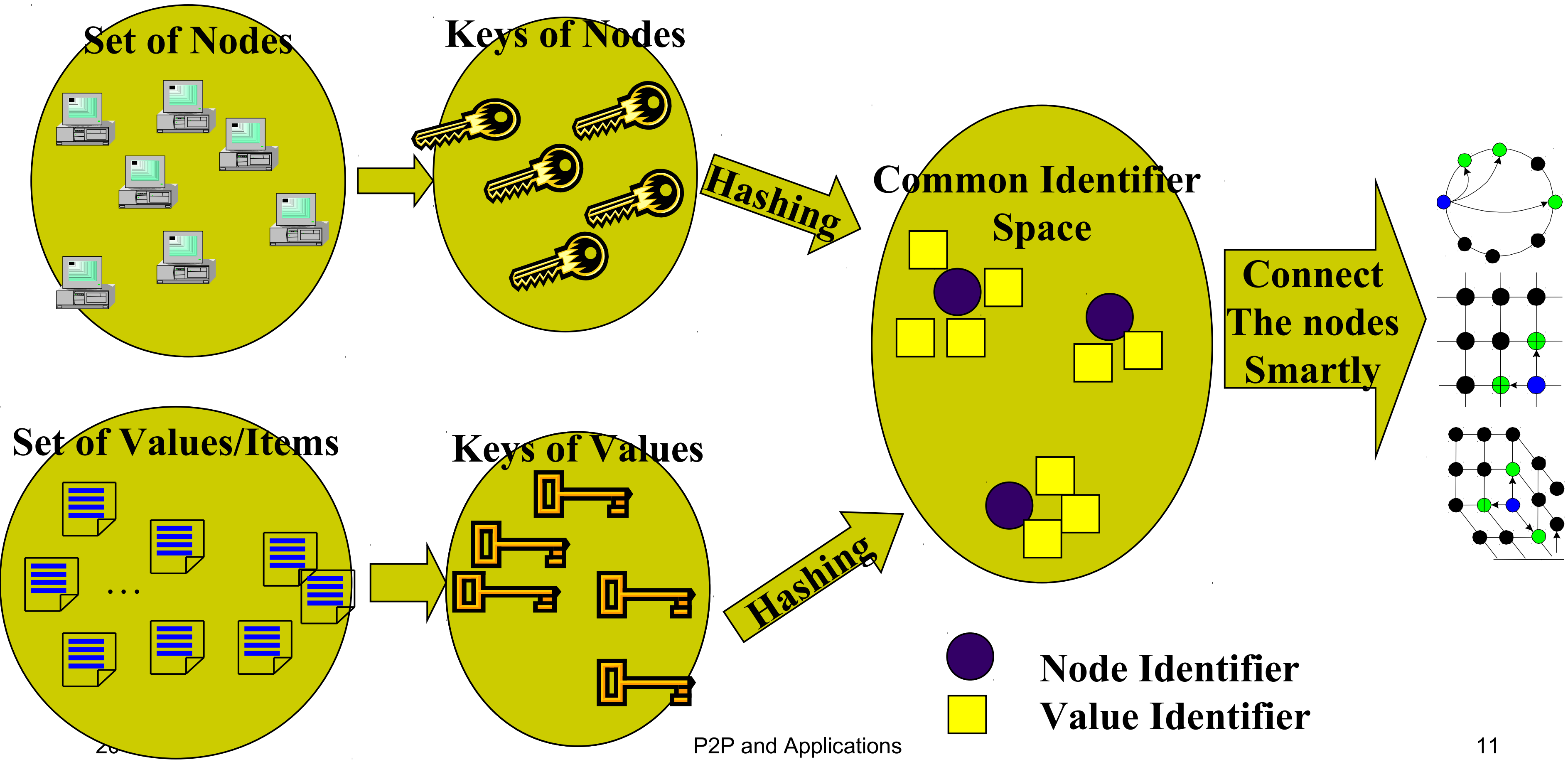
The Distributed Hash Table Abstraction



- put(key,value), get(key) interface
- The neighbors of a node are well-defined and not randomly chosen
- A value inserted from any node, will be stored at a certain well-defined node
- How do we do this?

Evolution of P2P - 3rd Generation (Structured Overlay Networks / DHTs) (2/2)

Main representatives:
Chord, Pastry, Tapestry, CAN,
Kademlia, P-Grid, Viceroy

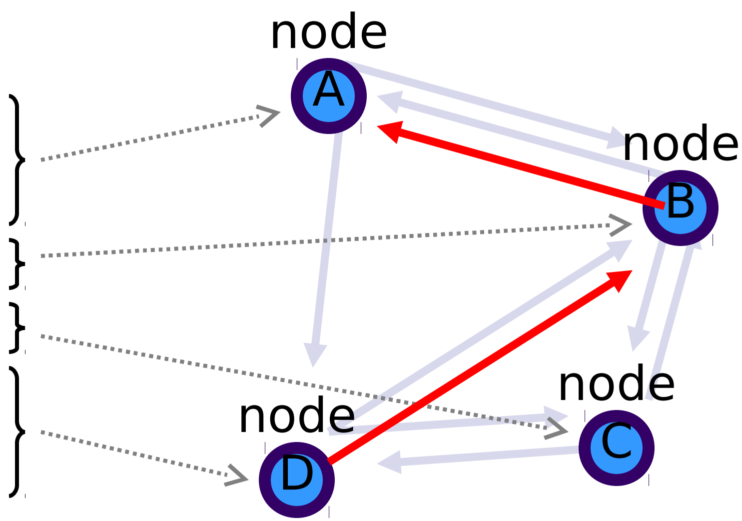


Distributed Hash Tables

The Principle of Distributed Hash Tables

- A dynamic distribution of a *hash table* onto a set of cooperating nodes

Key	Value
1	Algorithms
9	Routing
11	DS
12	Peer-to-Peer
21	Networks
22	Grids

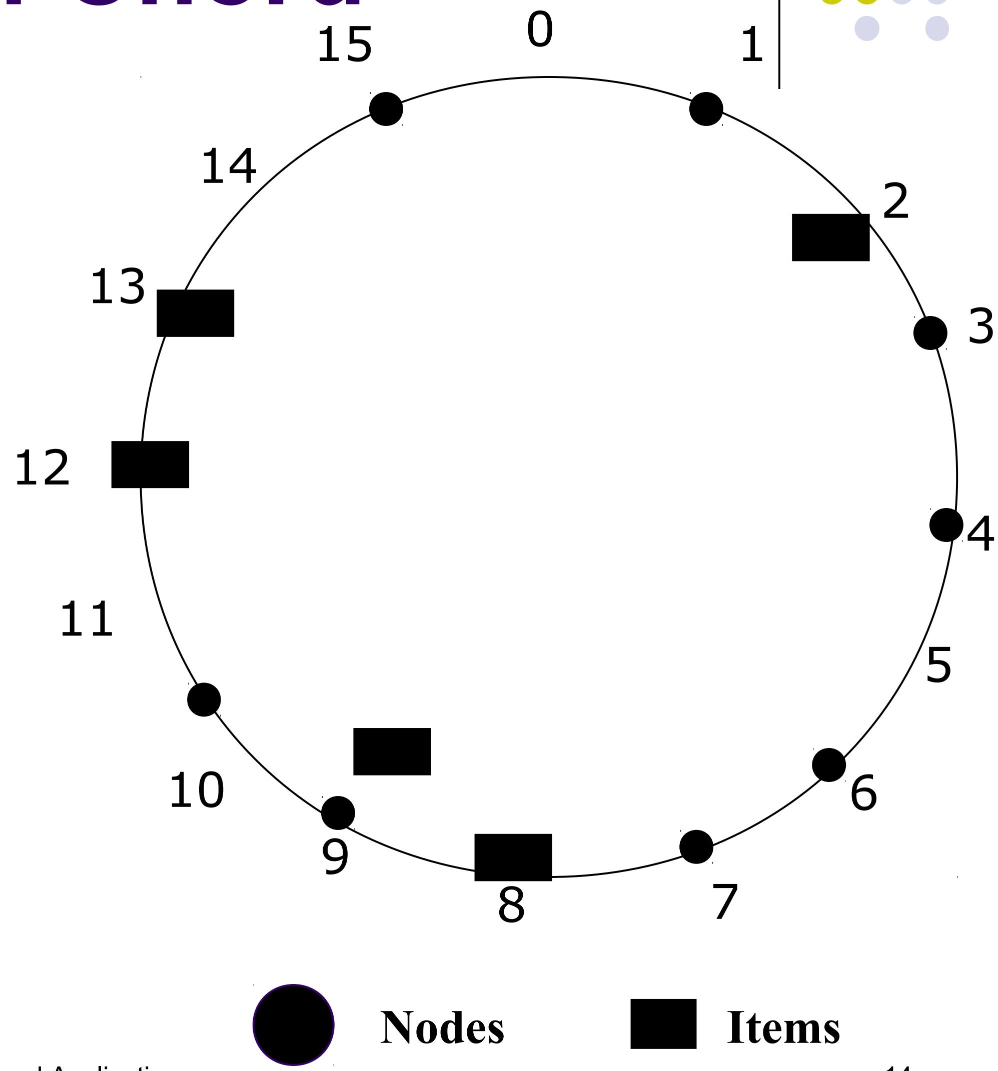


- Basic service: *lookup operation*
  - Key resolution from any node
- Each node has a *routing table*
  - Pointers to some other nodes
  - Typically, a constant or a logarithmic number of pointers

→Node *D* : *lookup(9)*

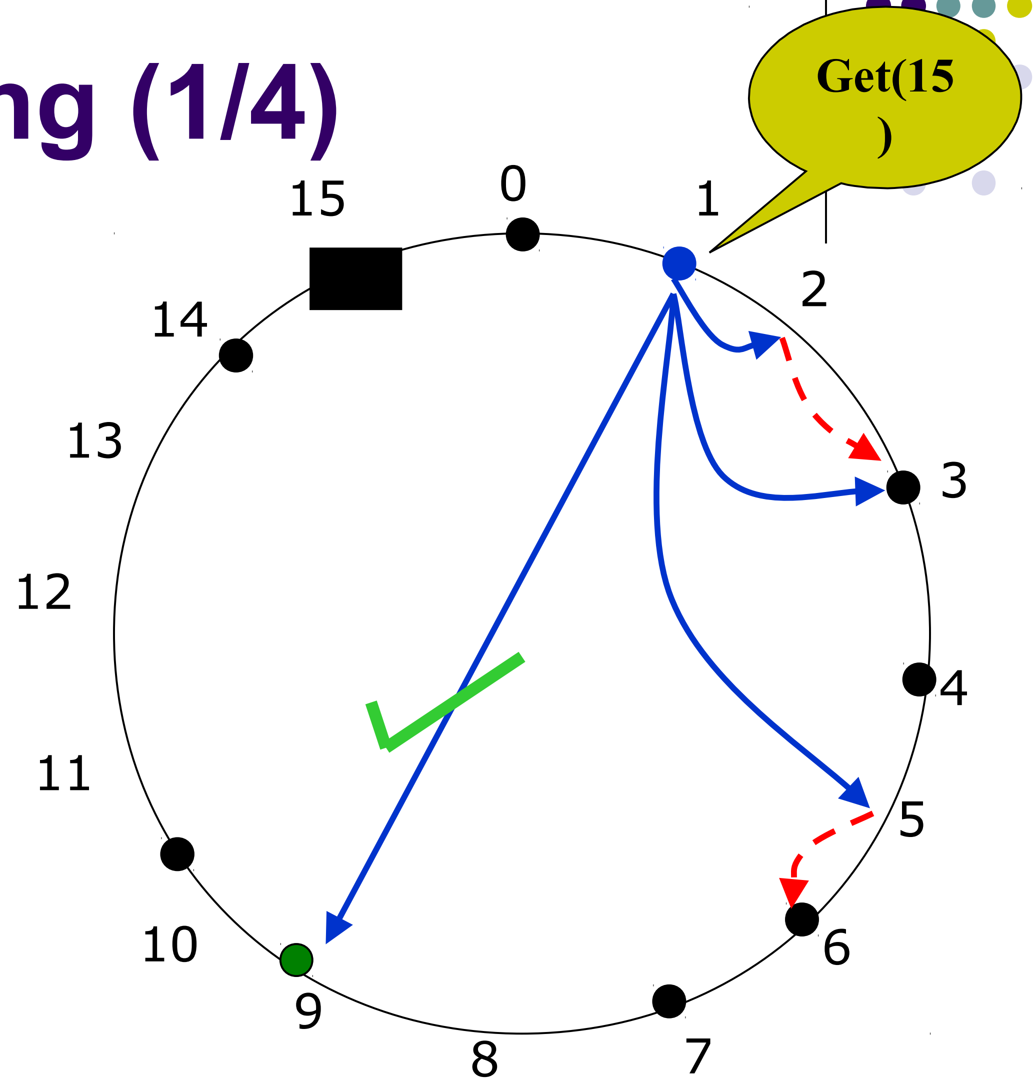
A DHT Example: Chord

- Ids of nodes and items are arranged in a circular space.
- An item id is assigned to the first node id that follows it on the circle.
- The node at or following an id on the space (circle) is called the successor
- Not all possible ids are actually used (sparse set of ids, e.g., 2^{128})!



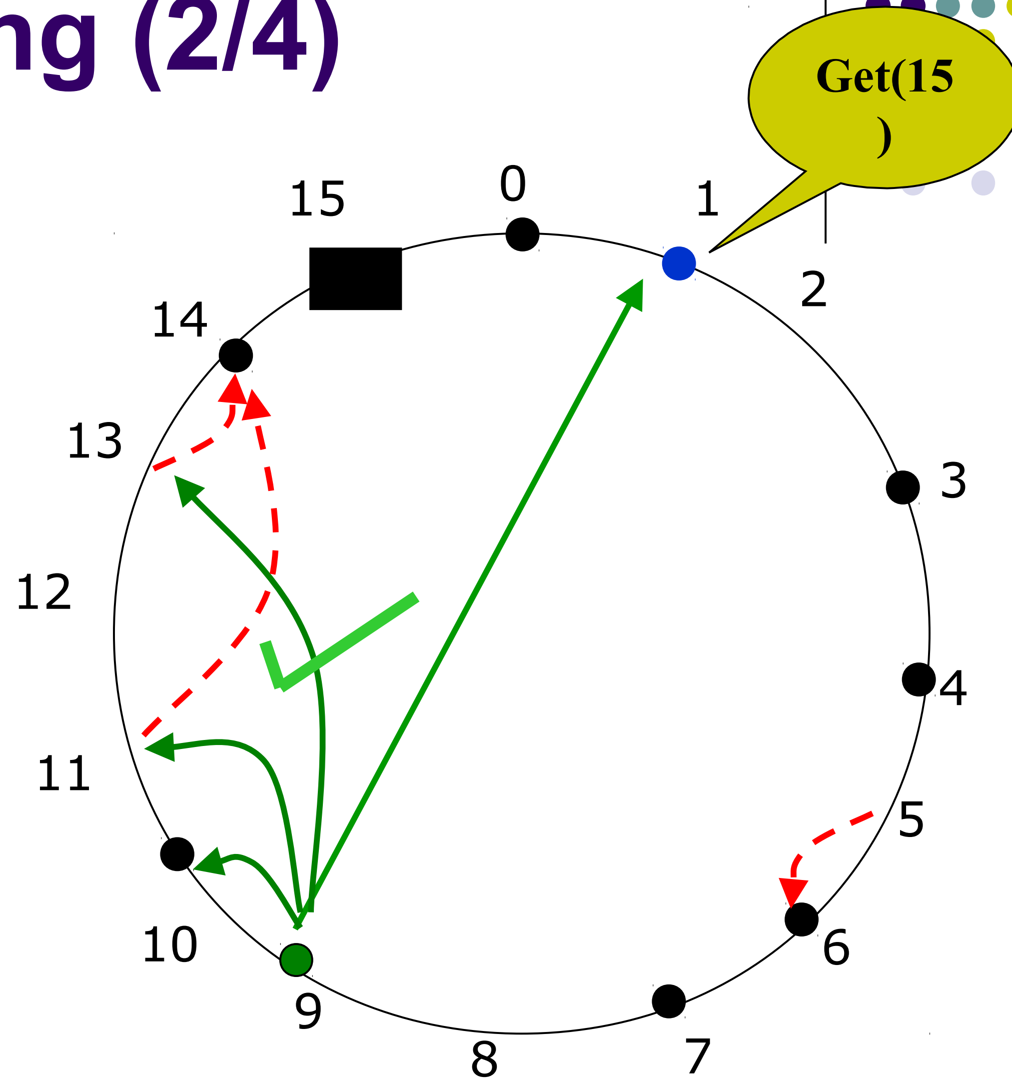
Chord – Routing (1/4)

- Routing table size: *M*, where $N = 2^M$
- Every node n knows successor $(n + 2^{i-1})$, for $i = 1..M$
- Routing entries = $log_2(N)$
- $log_2(N)$ hops from any node to any other node



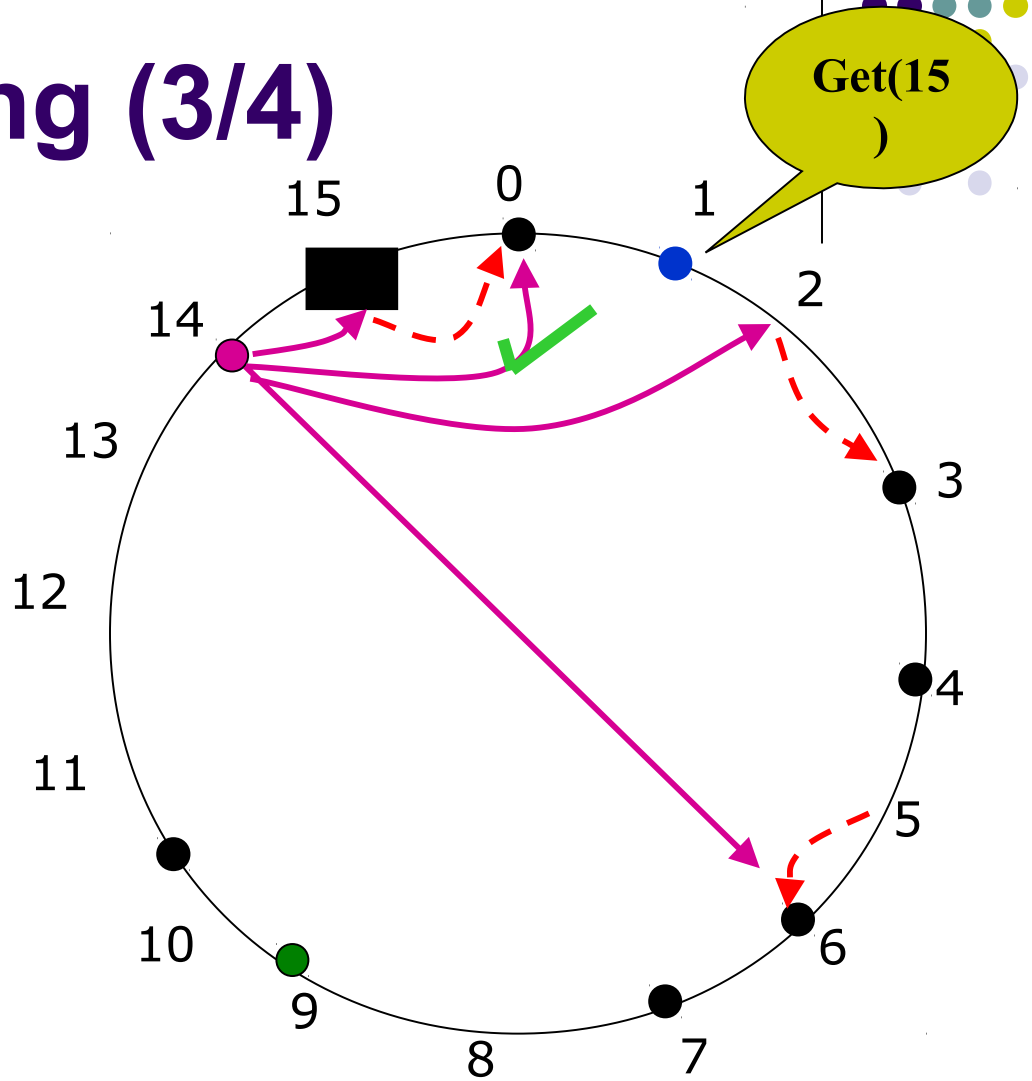
Chord – Routing (2/4)

- Routing table size: M, where $N = 2^M$
- Every node n knows successor $(n + 2^{i-1})$, for $i = 1..M$
- Routing entries = $\log_2(N)$
- $\log_2(N)$ hops from any node to any other node



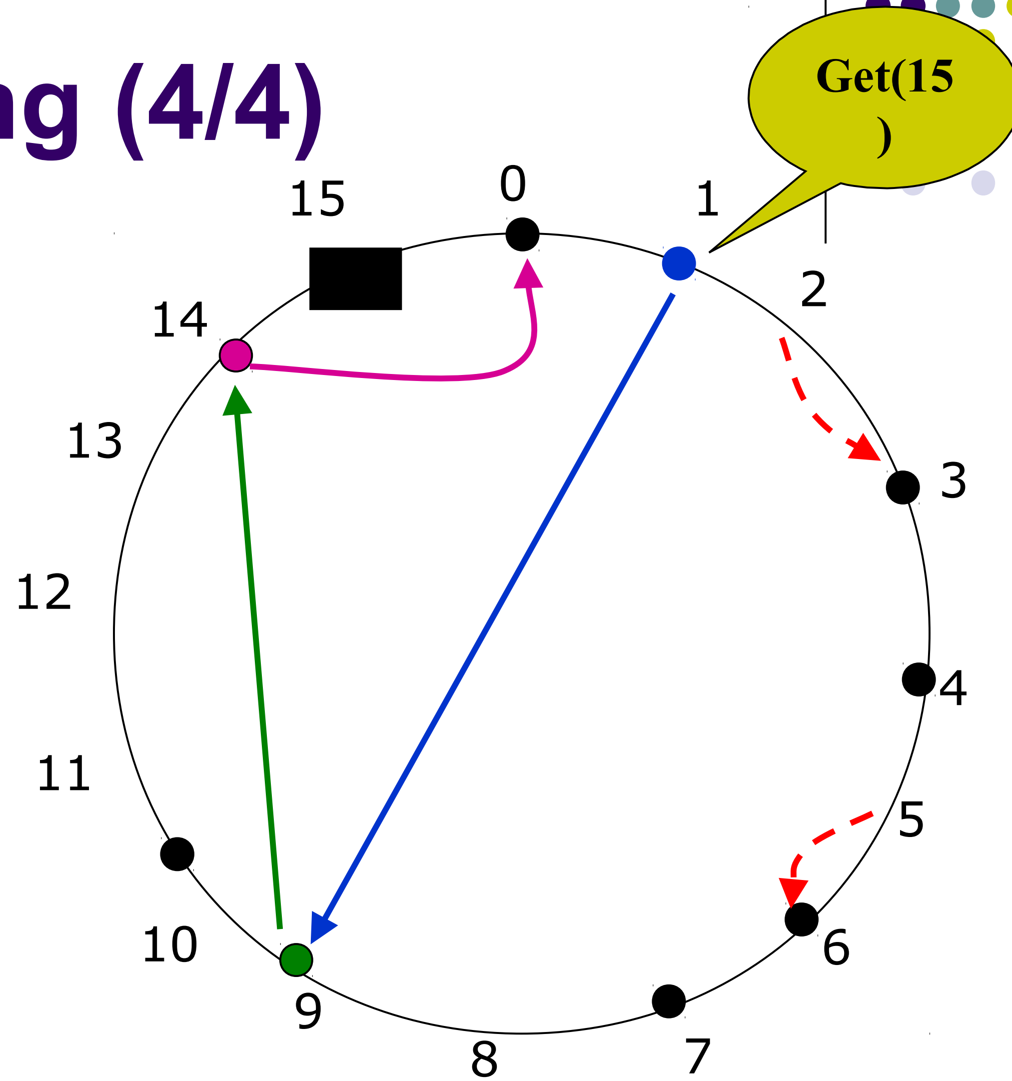
Chord – Routing (3/4)

- Routing table size: *M*, where $N = 2^M$
- Every node n knows successor($n + 2^{i-1}$) , for $i = 1..M$
- Routing entries = $log_2(N)$
- $log_2(N)$ hops from any node to any other node



Chord – Routing (4/4)

- From node 1, only 3 hops to node 0 where item 15 is stored
- For 16 nodes, the maximum is $\log_2(16)$ = 4 hops between any two nodes



Taxonomy of P2P Systems

- P2P Systems
  - Unstructured
    - Hybrid Decentralized (*Napster*)
    - Fully Decentralized (*Gnutella*)
    - Partially Decentralized (*Kazaa*)
  - Structured (*Chord*, *CAN*, *Tapestry*, *Pastry*)

Comparison of P2P Systems

	Napster	Gnutella	Pastry	Tapestry	Chord	CAN
Scalable	No	No	Yes	Yes	Yes	Yes
Guarantees	High	Low	High	High	High	High
Locality	Default	Default	Yes	Yes	No	No
Replication	Ad-hoc	Ad-hoc	Well-placed	Well-placed	Well-placed	Well-placed
Routing Table	$O(N)$	Varies	$O(\log(N))$	$O(\log(N))$	$O(\log(N))$	$2d$
Item Read	$O(1)$	$O(N)$	$O(\log(N))$	$O(\log(N))$	$O(\log(N))$	$O(N^{\frac{1}{d}})$
Item Insert	$O(1)$	$O(1)$	$O(\log(N))$	$O(\log(N))$	$O(\log(N))$	$O(N^{\frac{1}{d}})$
Item Delete	$O(1)$	$O(1)$	$O(\log(N))$	$O(\log^2(N))$	$O(\log(N))$	$O(N^{\frac{1}{d}})$
Node Insert	$O(1)$	$O(N)$	$O(\log(N))$	$O(\log(N))$	$O(\log(N))$	$O(N^{\frac{1}{d}})$
Node Delete	$O(1)$	$O(1)$	$O(\log(N))$	$O(\log(N))$	$O(\log(N))$	$O(1)$

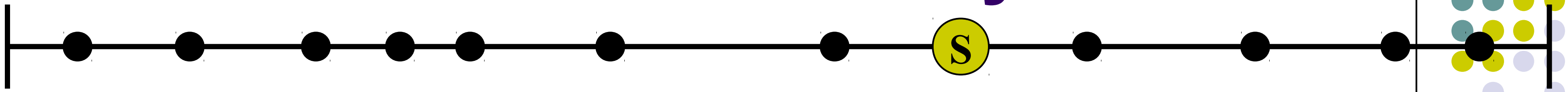
Research Issues in DHTs

- Lack of a Common Framework
- Absence of Locality
- Cost of Maintaining the Structure
- Complex Queries
- Heterogeneity
- Group Communication/Higher Level Services
- Cloud/Grid Integration

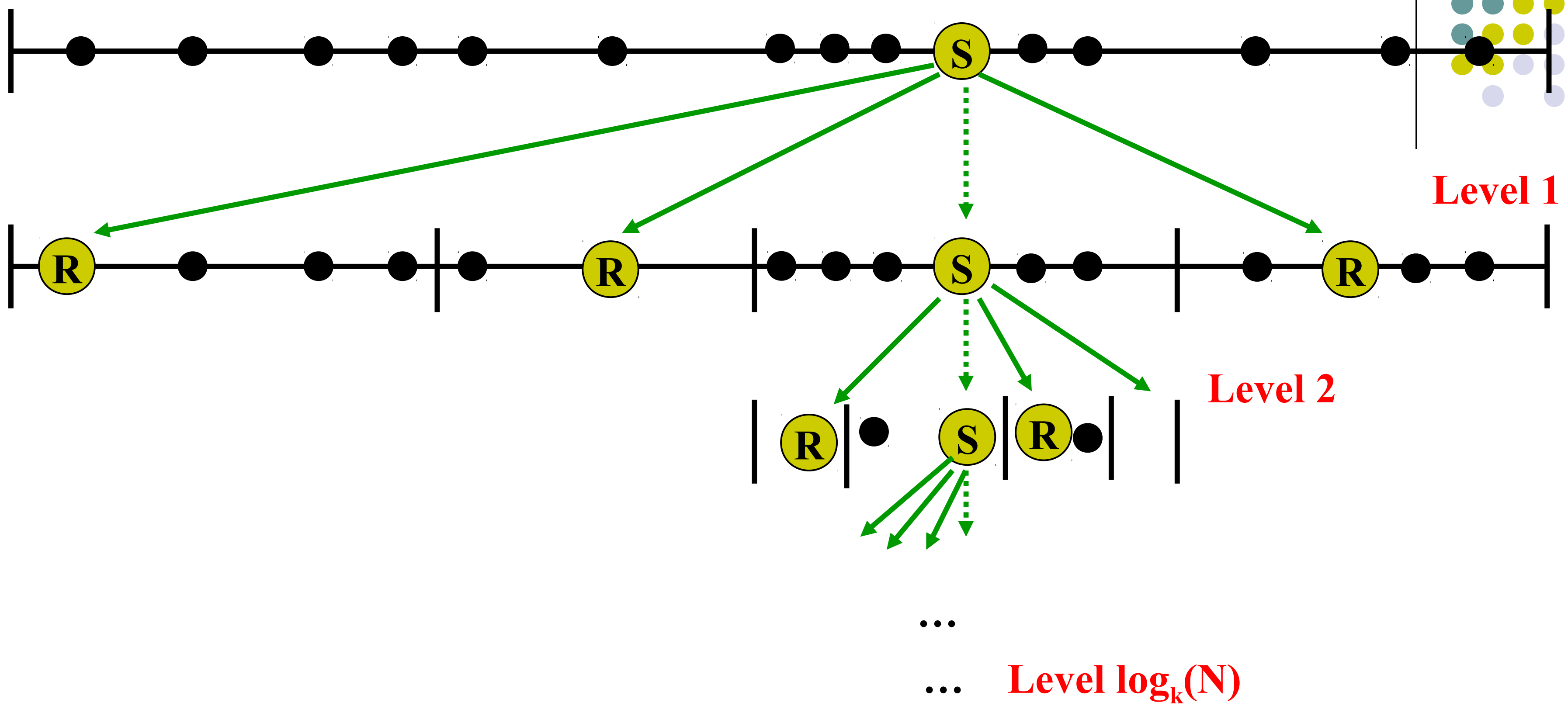
Framework

- **A Framework for Peer-To-Peer Lookup Services Based On k-ary Search**
- *Aspects*: **Understanding, Optimization**

DHTs as Distributed k-ary Search



DHTs as Distributed k-ary Search



S The lookup origin

● A node

|—| Interval in id space

R The **responsible** of an interval :

- *Successor* (Chord)
- *Next matching prefix* + proximity metric (Pastry, Tapestry)
- Next matching prefix + XOR metric (Kademlia)

→ Network Hop

⇢ Virtual Hop

The Space-Performance Trade-off

- We have ***N*** nodes.
- A node keeps info about a subset of peers
- Lookup length vs. routing table size trade-off
- Extremes:
  - Keep info about all peers
  - Keep info about one peer

Relating N, H and R

In general, for N nodes, the maximum lookup path length H and the number of routing entries R are as follows:

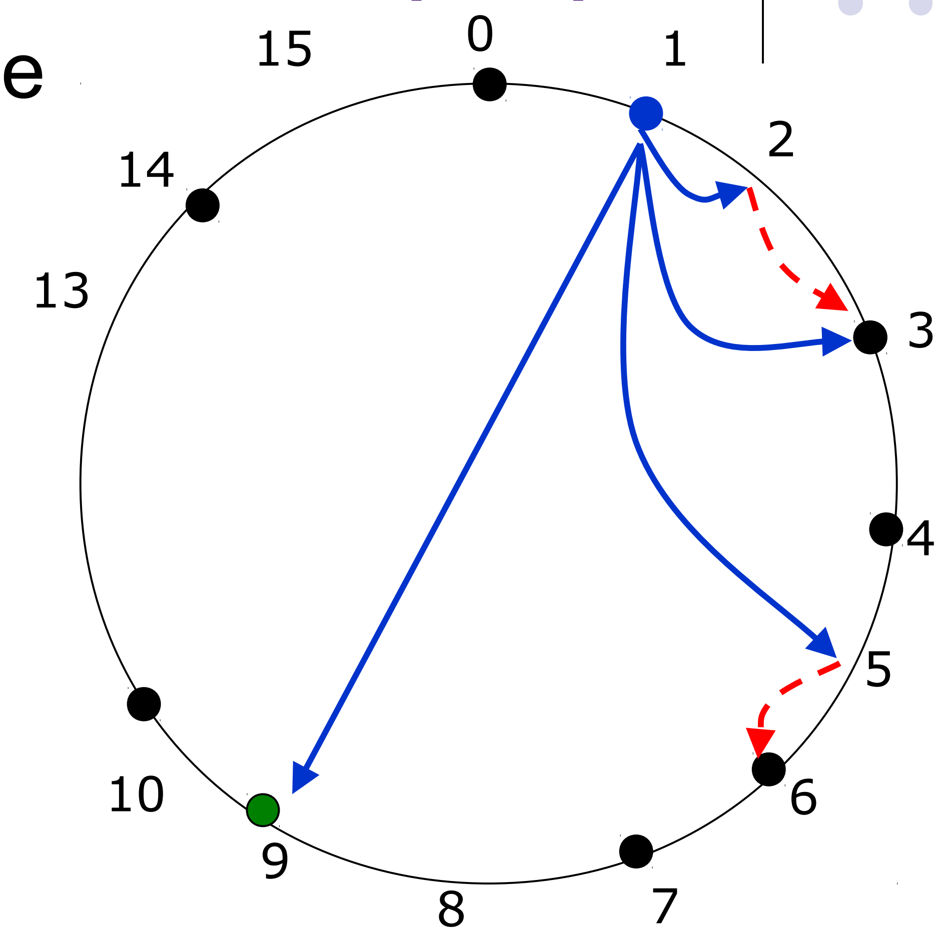
$H = \log_k(N)$ (Number of levels in the tree)

$R = (k-1) * \log_k(N)$ (k-1 neighbors per level)

$$N = (R/H + 1)^H$$

Chord as binary search (1/2)

- Chord is a special case of our view with with *k=2*, i.e., binary search
- *H= log₂(N)*
- *R= log₂(N)*



Chord as binary search (2/2)

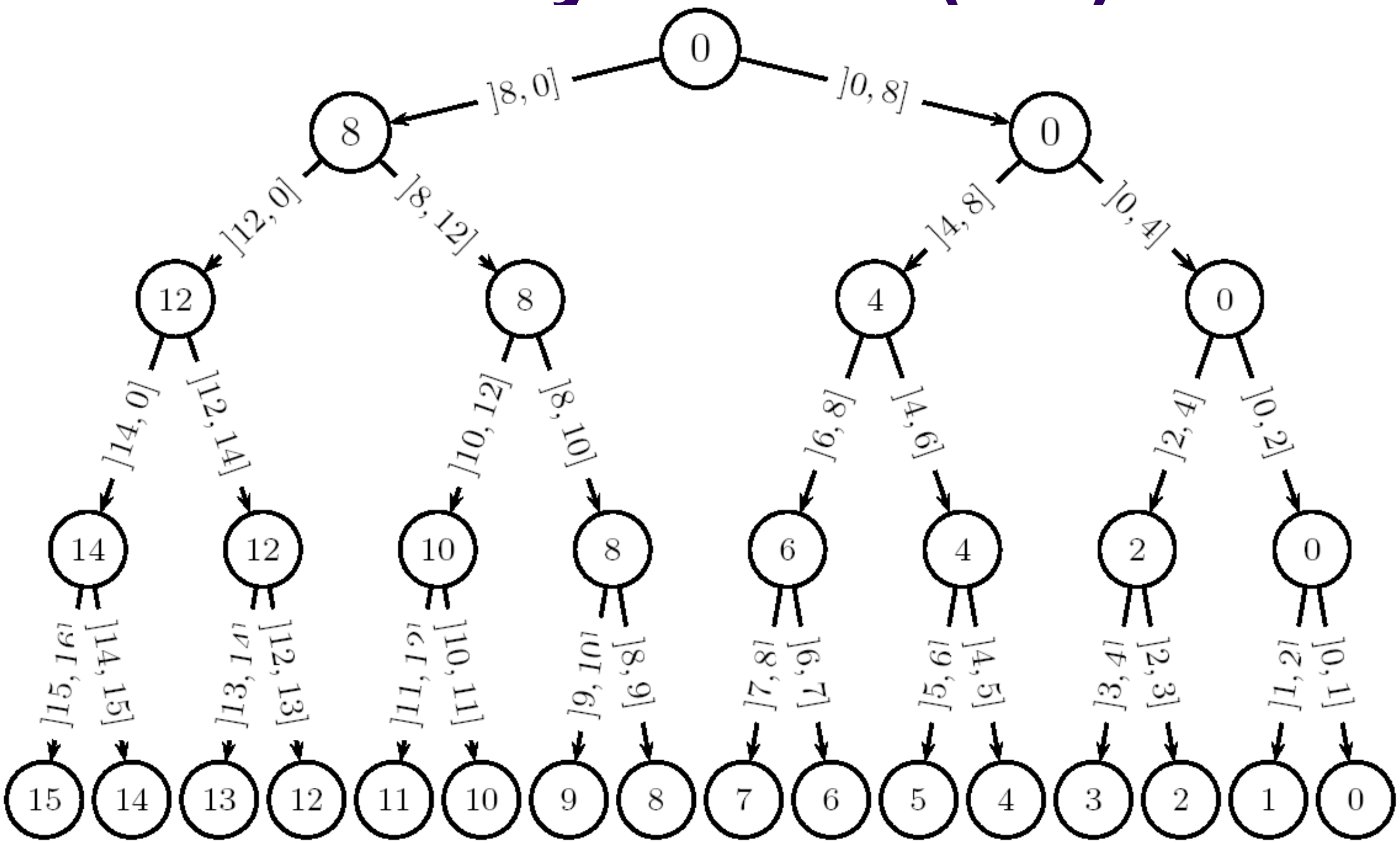


Figure 2: Decision tree for a query originating at node 0 in a 16-node network applying binary search

Generalizing Chord

Suggestion: Increase the search arity k by following the guidelines of our view and put enough info for k-ary search

$$H = \log_k(N)$$

$$R = (k-1)\log_k(N)$$

Why does routing table size matter?

- Not because of storage capacity
- First reason: because of the number of hops
- Second reason: because of the effort needed to correct an inconsistent routing table after the network changes

DKS: A Realistic DHT

DKS

- A P2P system that:
  - Realizes the DKS principle
  - Offers strong guarantees because of the local atomic actions
  - Introduces novel technique that avoids unnecessary bandwidth consumption
- Relevance to research issues in state-of-the-art P2P systems:
  - Common framework
  - Cost of maintaining the structure

DKS(N,k,f)

- *Title*: **DKS(N,k,f): Family of Low Communication, Scalable and Fault-Tolerant Infrastructures for P2P Applications**
- *Authors*: Luc Onana Alima, Sameh El-Ansary, Per Brand, and Seif Haridi.
- *Place*: In The 3rd International Workshop on Global and Peer-To-Peer Computing on Large-scale Distributed Systems - CCGRID2003, Tokyo, Japan, May 2003.
- *Aspects*: **Understanding, Design**

Design principles in DKS(N,k,f)

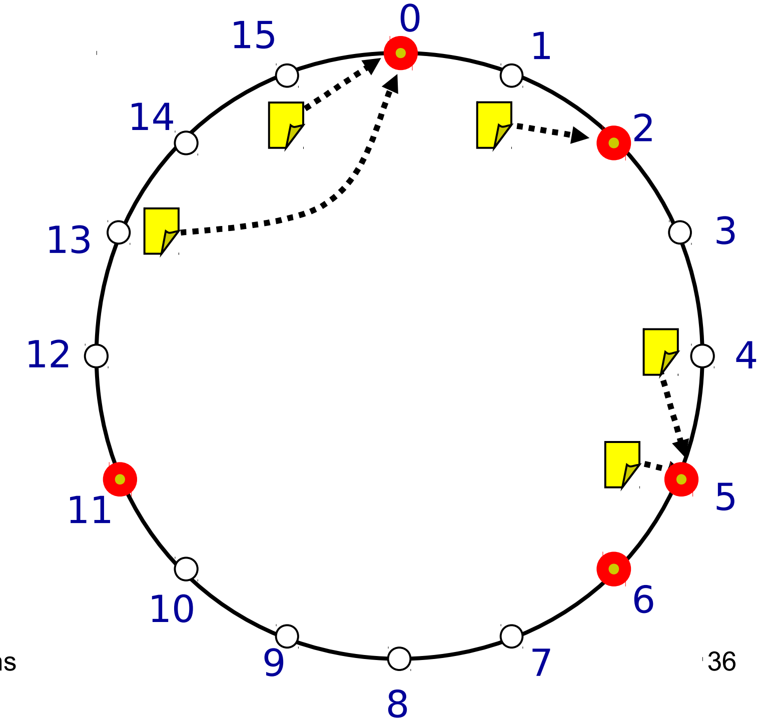
- Distributed K-ary Search (DKS) principle
- Local atomic action for joins and leaves
- Correction-on-use technique
- (Replication for fault tolerance) – *not discussed*

Design Principles in DKS

- Tunability
  - Routing table size vs. lookup length
  - Fault-tolerance degree
- Local atomic join and leave
  - Strong guarantees
- Correction-on-use
  - No unnecessary bandwidth consumption

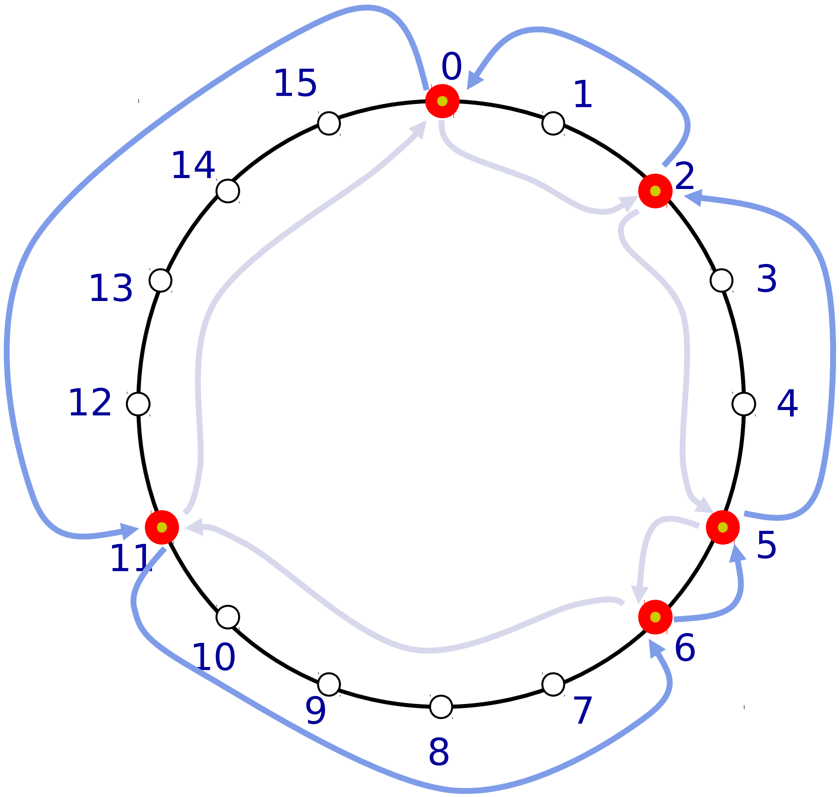
DKS overlay illustrated (1/3)

- An *identifier space* of size $N=k^L$ is used
  - A logical ring of N positions
- A hash function, H, maps:
  - Nodes onto the logical ring
  - Items onto the logical ring
- An item (*key,value*) is stored at *successor* of *H(key)*, first node that follows *H(key)* on the ring in the clockwise direction



DKS overlay illustrated (2/3)

- Basic interconnection
  - Bidirectional linked list of nodes
  - Each node points to its
    - Predecessor
    - Successor
  - Resolving key
    - *O(N) hops* in an *N*-node system



Design principle 1: Distributed K-ary Search (DKS) principle

- The DKS is designed from the beginning based on the distributed k-ary search principle
- The system uses the successor of an identifier in a circular space for assigning responsibilities

DKS overlay illustrated (3/3)

- Enhanced Interconnection
  - Speeding up key resolution: *$\log_k(N)$ hops*
  - At each node, a RT of $\log_k(N)$ levels
  - Each level of RT has *k* intervals
  - For level *l* and interval *i*
    - *(RT(l))(i)* = address of the first node that follows the start of the interval *i*

      (responsible node)

Notation

Ιδεντιφιερ σπαχε I

$$I = \{0,1,\ldots,N-1\}$$

$N = k^L$ (σιζε οφ τηε σπαχε)

α $\oplus$ β $\equiv (a+b)$ μοδ N

Λεϖελσ ανδ ςιεωσ

νυμβερ οφ λεϖελσ λογ$_k N \equiv L$

ατ λεϖελ 1, $V^1 = I_0^1 \cup I_1^1 \cup \ldots \cup I_{k-1}^1$

$$I_i^1 = [x_i^1, x_{i+1}^1[, \quad x_i^1 = n \oplus i\frac{N}{k}$$

Levels and views

Λεϖελσανδ ςιεωσ

number of levels $\log_{\kappa} N \equiv \Lambda$

at level 1, $\varsigma^{1} = I_0^1 \cup I_1^1 \cup \ldots \cup I_{\kappa-1}^1$

$$I_\iota^1 = [\,\xi_\iota^1, \xi_{\iota+1}^1[, \quad \xi_\iota^1 = \nu \oplus \iota \frac{N}{\kappa} \text{ for } 0 \le \iota \le \kappa - 1$$

at level $2 \le \lambda \le \Lambda, \varsigma^{\lambda} = I_0^\lambda \cup I_1^\lambda \cup \ldots \cup I_{\kappa-1}^\lambda$

$$I_0^\lambda = [\,\xi_0^\lambda, \xi_1^\lambda[, \ldots, I_{\kappa-1}^\lambda = [\,\xi_{\kappa-1}^\lambda, \xi_0^{\lambda-1}[$$

$$I_\iota^\lambda = [\,\xi_\iota^\lambda, \xi_{\iota+1}^\lambda[, \quad \boxed{\xi_\iota^\lambda(\nu) = \nu \oplus \iota \frac{N}{\kappa^\lambda}} \text{ for } 0 \le \iota \le \kappa - 1$$

Responsibility

At each level V^l there is a node $R(I_i^l)$ that is responsible for each interval I_i^l

let x be a node, $S(x)$ is the first node encountered in $[x, x[$

For an arbitrary node n and an arbitrary level l, the responsible for I_i^l is $S(x_i^l)$

Each node n is itself responsible for I_1^l for any l

Observe that $S(x_i^l)$ *is a function of* n

can be computed by any node

DKS Overlay illustrated-4

- Example, k=4, N=16 (4^2)
  - *At each node an RT of two levels*
  - *In each level, 4 intervals*
- Let us focus on node *1*

- **Level 1, 4 intervals**

  I_0 = [1,5[: (*RT*(1))(0) = 1

  I_1 = [5,9[: (*RT*(1))(1) = 6

  I_2 = [9,13[: (*RT*(1))(2) = 10

  I_3 = [13,1[: (*RT*(1))(3) = 15

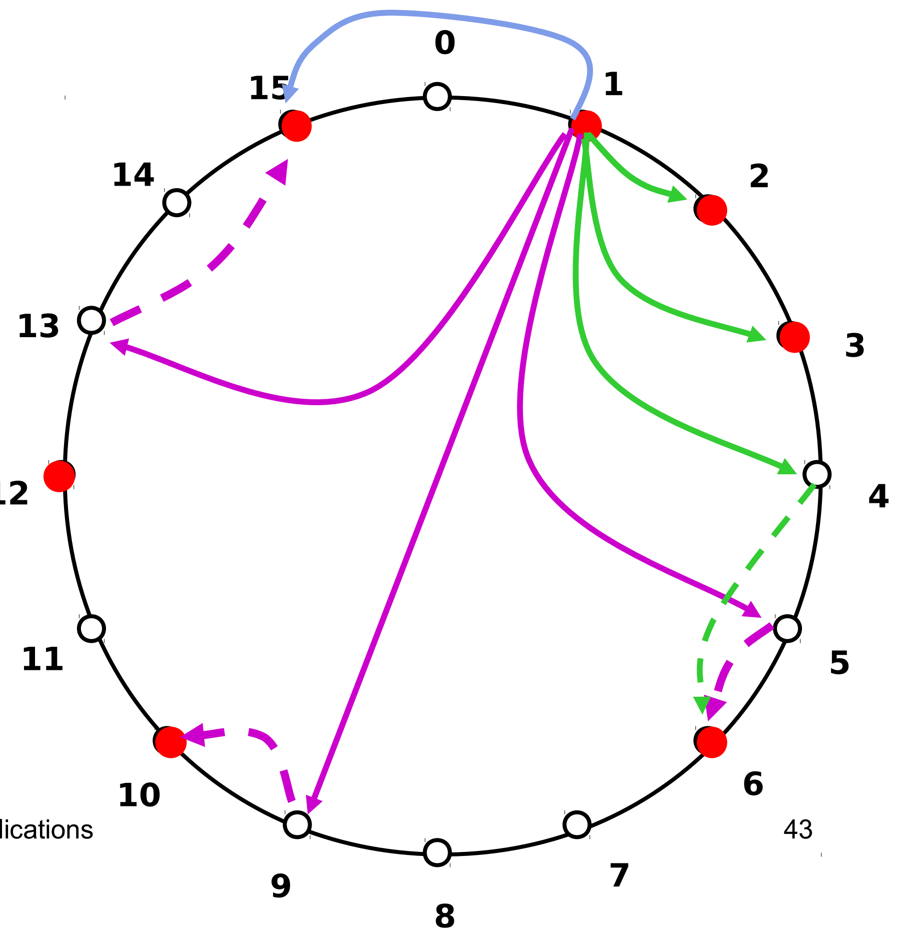
- **level 2, 4 intervals**

  I_0 = [1,2[: (*RT*(2))(0) = 1

  I_1 = [2,3[: (*RT*(2))(1) = 2

  I_2 = [3,4[: (*RT*(2))(2) = 3

  I_3 = [4,5[: (*RT*(2))(3) = 6

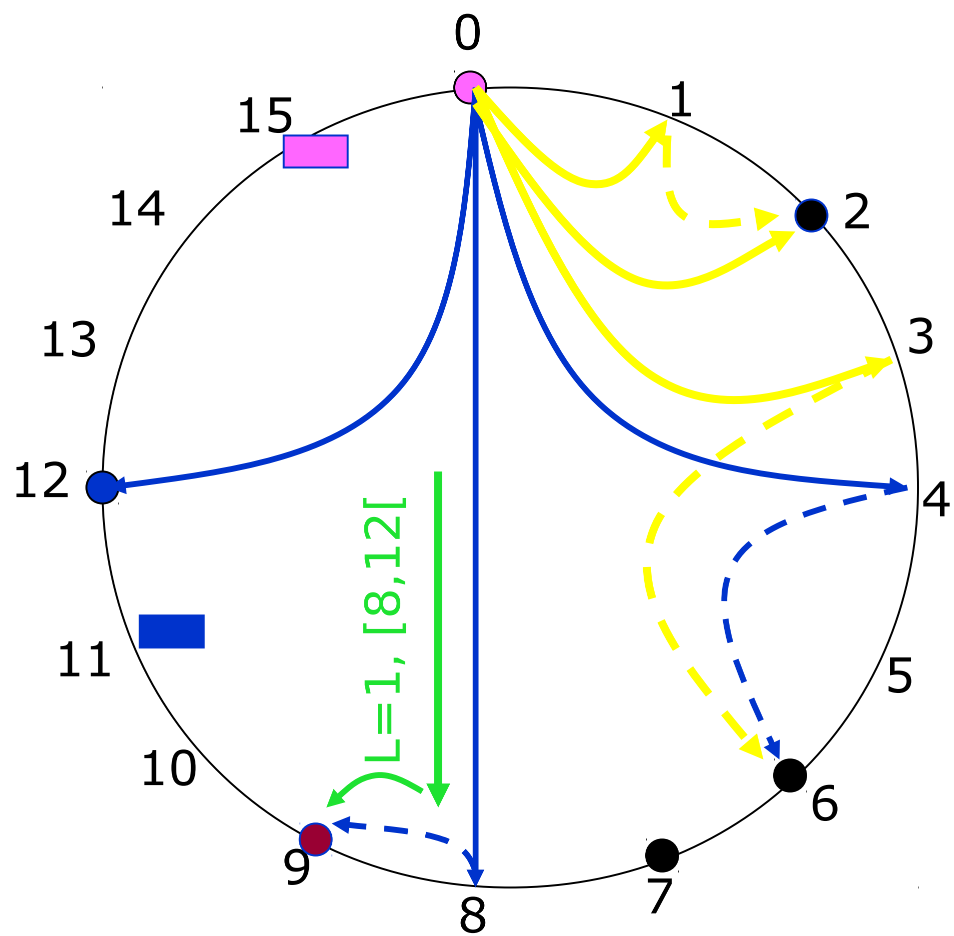


Lookup in a DKS(N,k,f) network (basic idea)

- A predecessor pointer is added at each node
- Interval routing
  - If key between my predecessor and me, done
  - Otherwise, systematic forwarding level by level

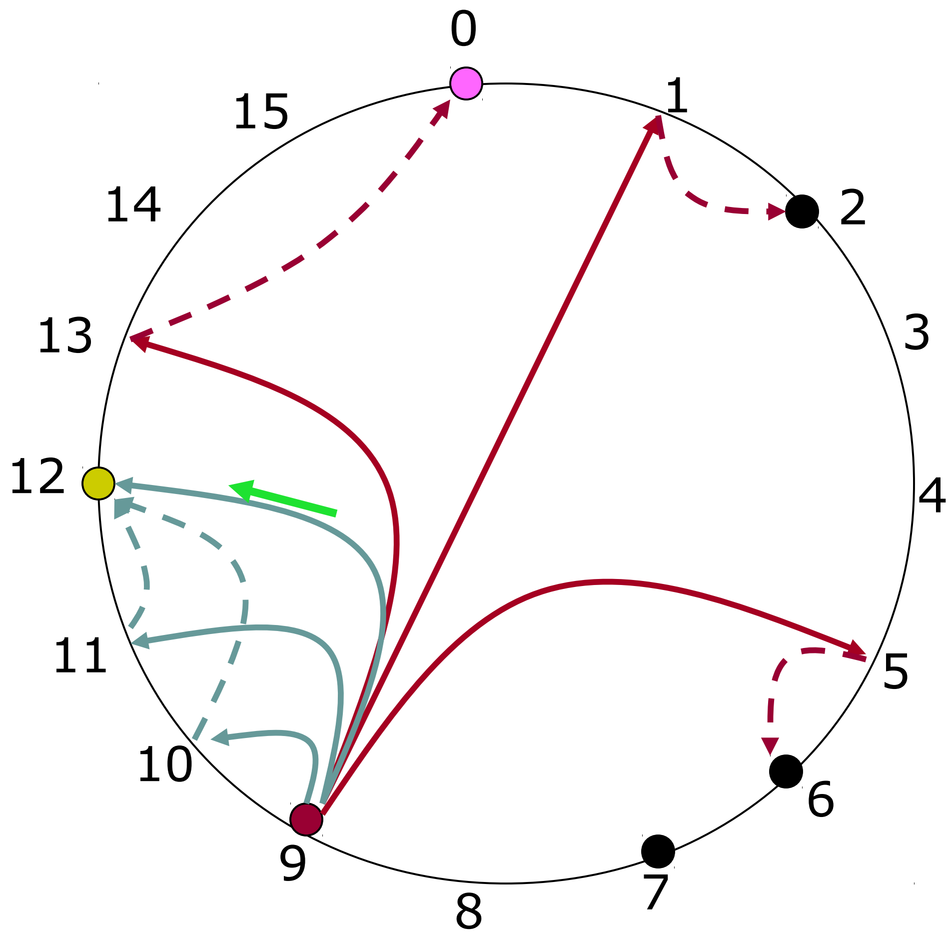
Lookup in a DKS(N,k,f) network illustrated (1/2)

- **A lookup request for 11 from node 0**
- **Node 0 sends a request to 9**
  - **Piggybacking of sender's current position on its "tree"**



Lookup in a DKS(N,k,f) network illustrated (2/2)

- **A lookup request for 11 from node 0**
- **Node 9 behaves similarly**
  - **Uses its level 2 for forwarding**
- **Request resolved in two hops**



L=2, [11,12[

Design principle 2: Local atomic action for guarantees

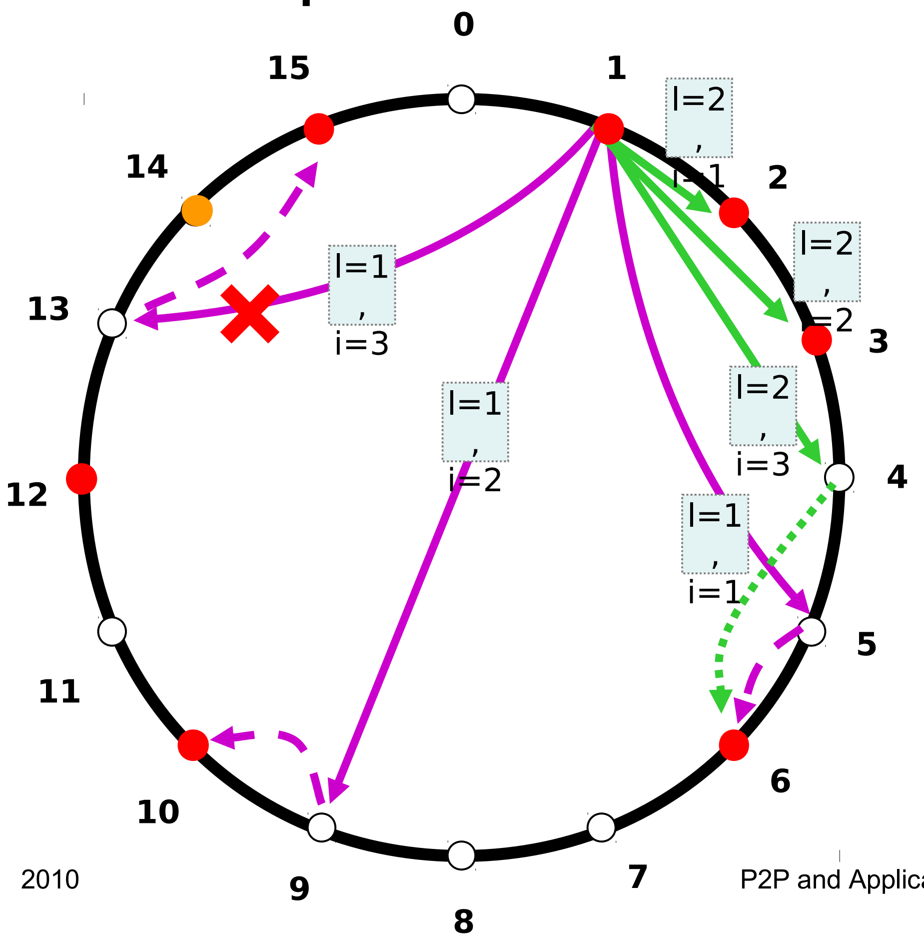
- To ensure that any key-value pair previously inserted is found despite concurrent joins and leaves
  - We use **local atomic operation** for
    - Node join
    - Node leave
- Stabilization-based systems do not ensure this

DKS(N,k,f) network construction

- A joining node is **atomically** inserted by its current successor on the virtual space
- The atomic insertion involves only three nodes in fault-free scenarios
- The new node receives **approximate** routing information from its current successor
- Concurrent joins on the same segment are **serialized** by mean of local atomic action

DKS routing table maintenance

- Node *14* joins the system
  - Example: node *1* in DKS(N=*16*, k=*4*, *f*)



- Node *1*'s pointer on level=*1*, interval=*3* becomes invalid
- Will be corrected by *Correction-on-use*

Design principle 3: Correction-on-use

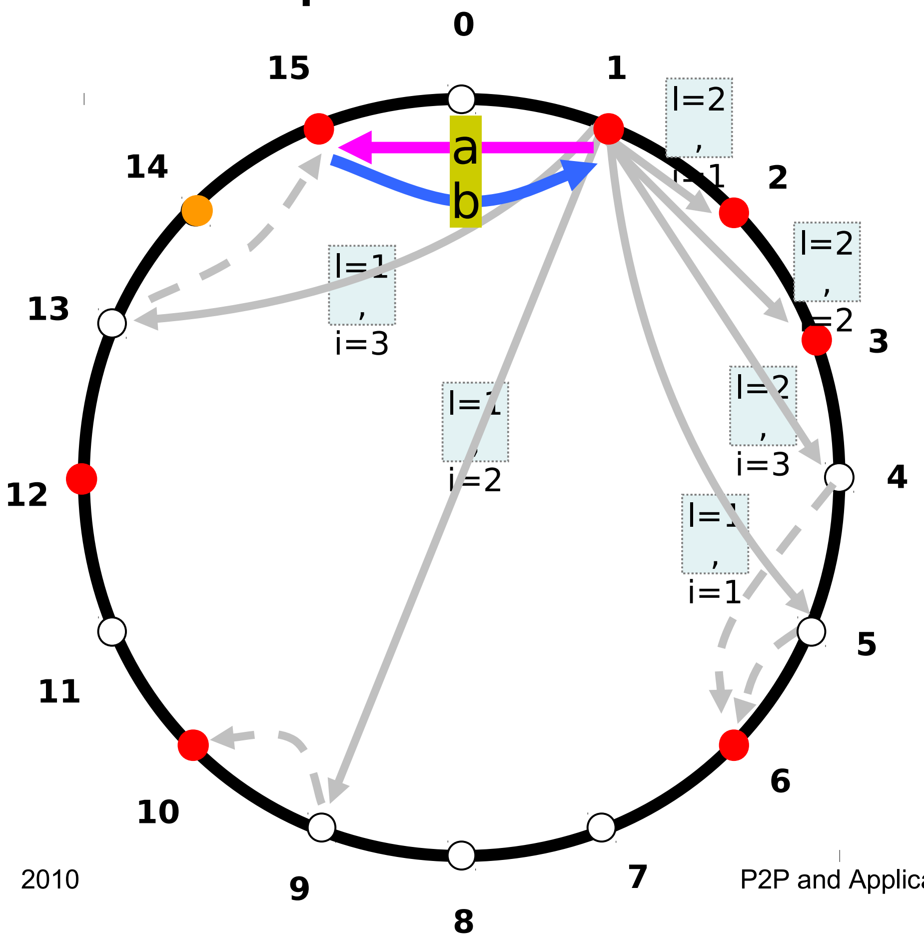
- A node always talks to a responsible node
  - Knowledge of responsible may be erroneous
- “*If you tell me from where (**in your “tree”,**) you are contacting me, then I can tell you whether you know the correct responsible*”
  - Help others to correct themselves
- “*If I heard from you, I learn about your existence*”
  - Help to correct myself

Correction on use

- Look-up or insert messages from node *n* to node *n'*
- Add the following to the message
  - *i* (interval) and *l* (level)
- Node *n'* can compute : $x_i^l(n)$
- Node *n'* maintains a list of predecessors *BL*

DKS *correction-on-use*

- **Node 1: *lookup(key:13)***
  - Example: node *1* in DKS(*N=16*, *k=4*, *f*)



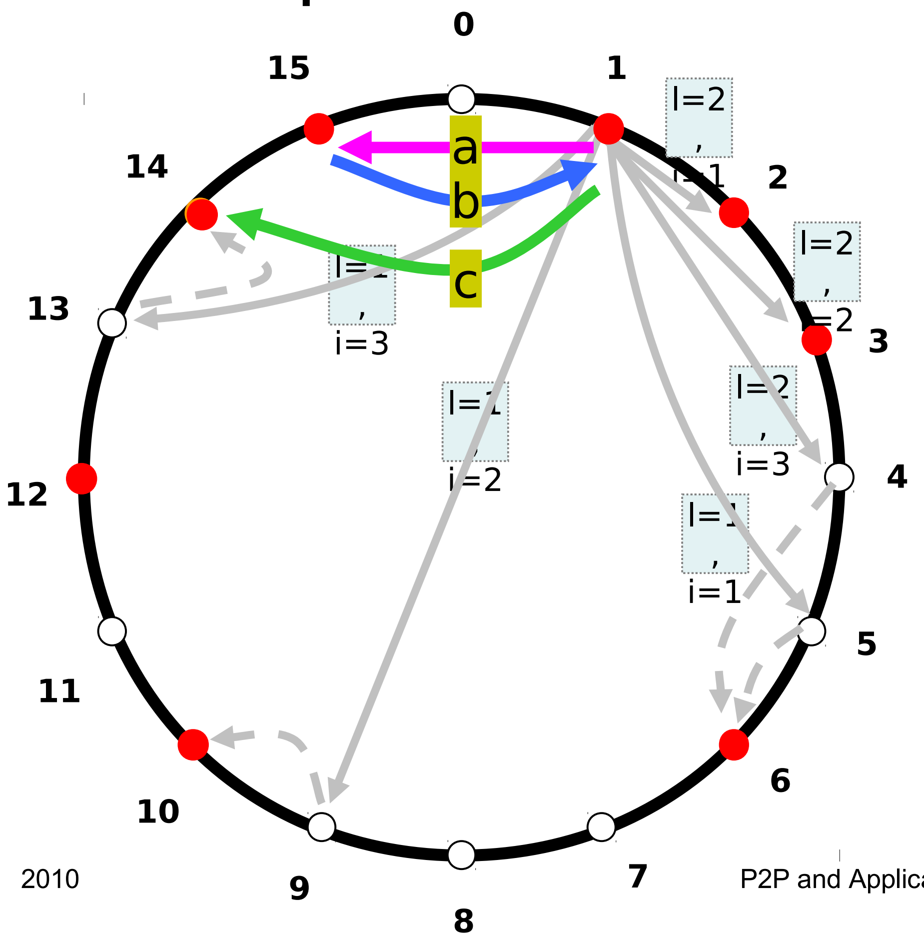
- **Node *1*'s uses its pointer on level=*1* interval=*3***

a) Lookup(Key:13, *Level:1, Interval: 3*)

b) BadPointer(Key:13,Candidate:14)

DKS *correction-on-use*

- **Node 1: *lookup(key:13)***
  - Example: node *1* in DKS(*N*=*16*, *k*=*4*, *f*)



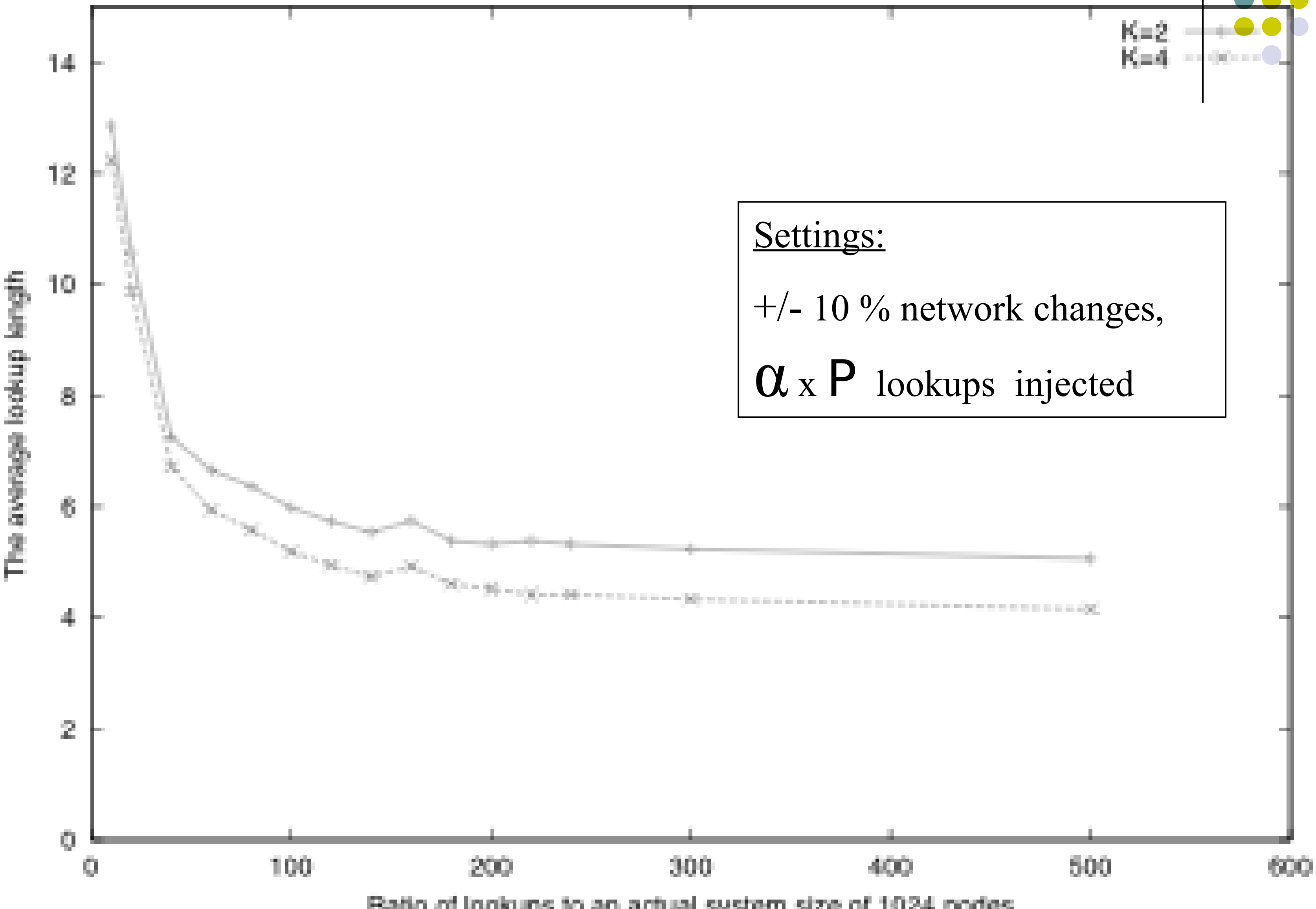
- **Node *1*'s uses its pointer on level=*1* interval=*3***

a) **Lookup(Key:13, Level:1, Interval: 3)**

b) **BadPointer(Key:13,Candidate:14)**

c) **Lookup(Key:13, Level:1, Interval: 3)**

Correction-on-use works given enough traffic



Settings:

+/- 10 % network changes,

α x P lookups injected

Broadcast in DKS

Efficient Broadcast

- *Title*: **Efficient Broadcast in Structure P2P Systems**
- *Authors*: Sameh El-Ansary, Luc Onana Alima, Per Brand, and Seif Haridi.
- *Place*: In The 2nd International Workshop on Peer-to-Peer Systems (IPTPS ’03), February 2003.
- *Related aspects*: Design

Motivation: Why broadcast is needed for DHTs?

- In general, support for **global dissemination/collection** of info in DHTs.
- In particular, the ability to perform **arbitrary queries** in DHTs.

The Broadcast Problem in DHTs

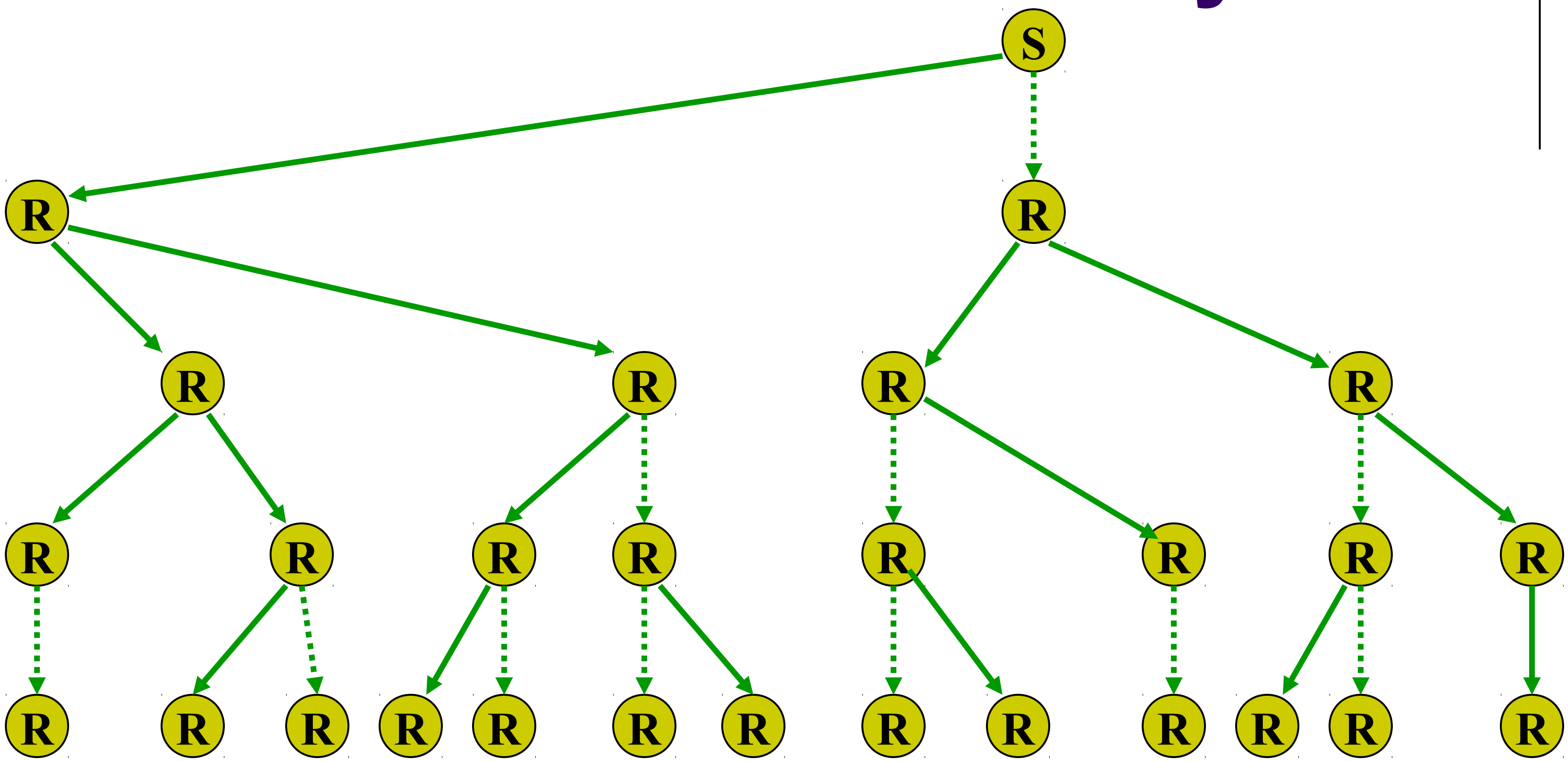
Problem: *Given an overlay network constructed by a P2P DHT system, find an efficient algorithm for broadcasting messages. The algorithm should not depend on global knowledge of membership and should be of equal cost for any member in the system.*

The Efficient Broadcast Solution

Construct a spanning tree derived from the decision tree of the distributed k-ary search after removal of the virtual hops.

This is equivalent to a Best-Effort Broadcast: it relies on the correctness of the initial node.

DHTs as Distributed k-ary Search



S The lookup origin	● A node	Interval in id space

R The **responsible** of an interval :
- *Successor* (Chord)
- *Next matching prefix* **node** in (Pastry, Tapestry)

Network Hop

Virtual Hop

Other Solutions for Broadcast

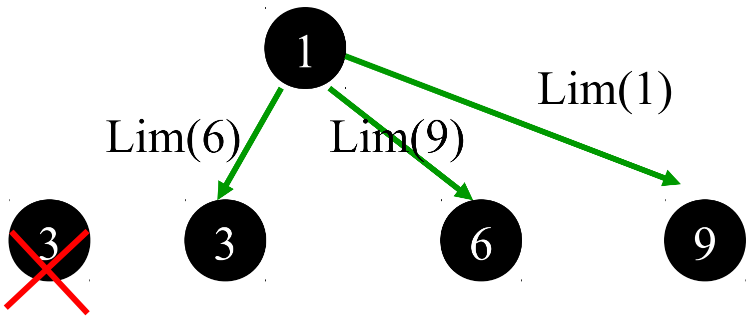
- Gnutella-like Flooding in DHT
  - (Pro) Known diameter→ Correct TTL → High Guarantees
  - (Con) The traffic is high with redundant messages
- Traversing the ring in Chord or Pastry
  - (Pro) No redundant messages
  - (Con) Sequential execution time
  - (Con) Highly sensitive to failure

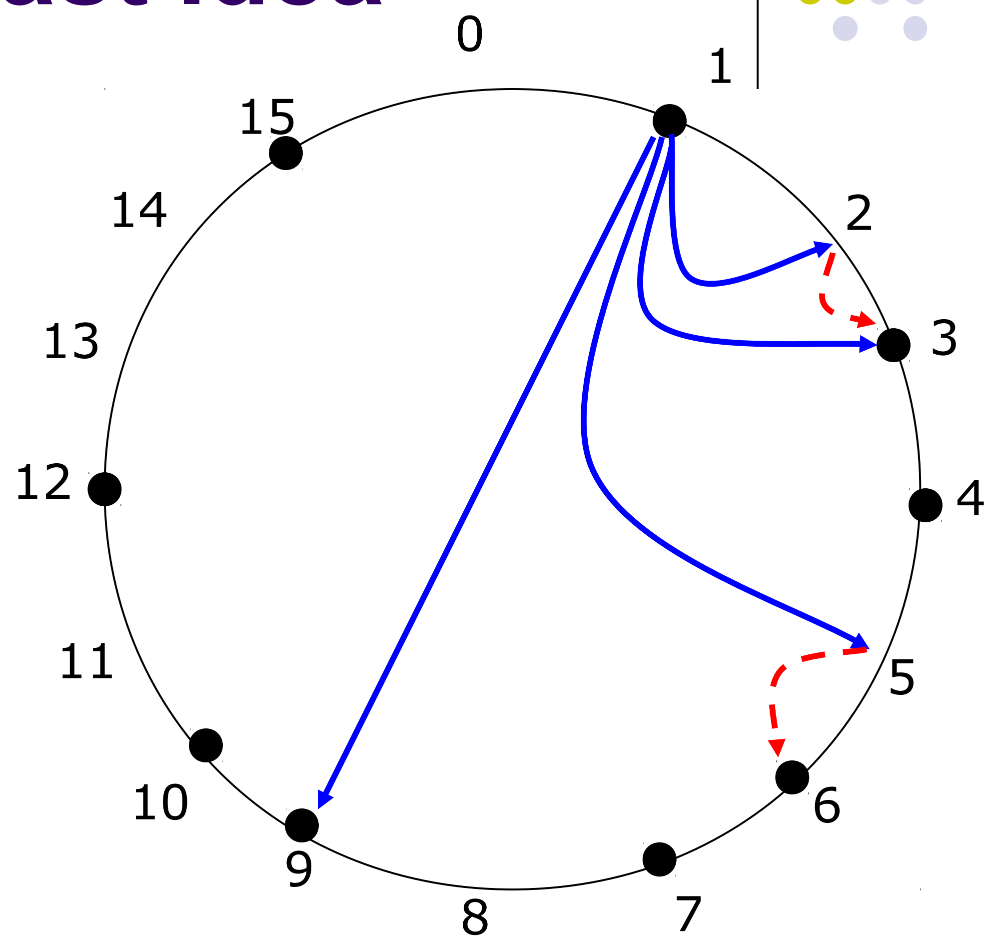
Efficient Broadcast Algorithm Invariants

- Any node sends to distinct routing entries.
- Any sender informs a receiver about a forwarding limit, that should not be crossed by the receiver or the neighbors of the receiver.

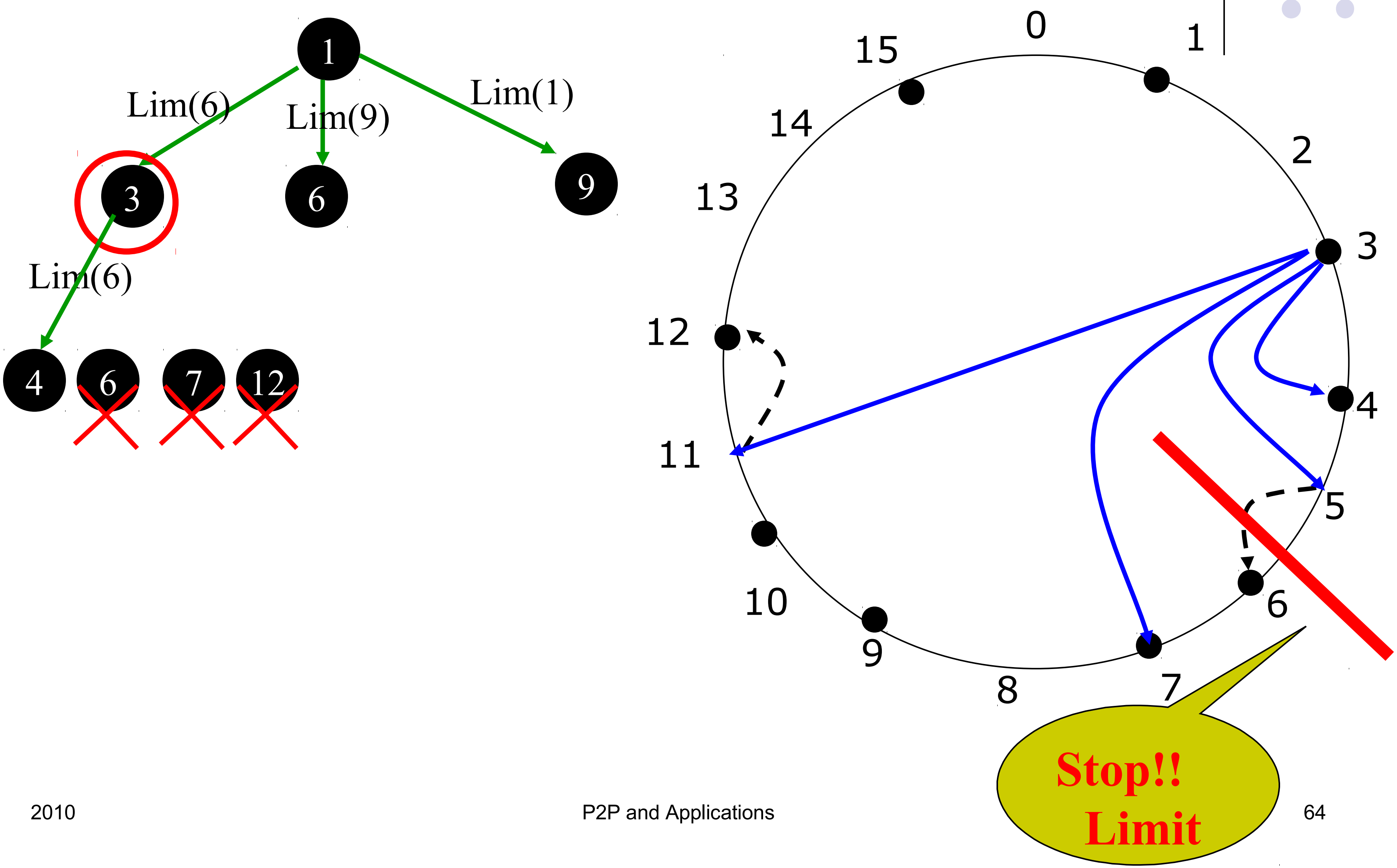
Forwarding within disjoint intervals where every node receives a message exactly once.

Efficient Broadcast Idea

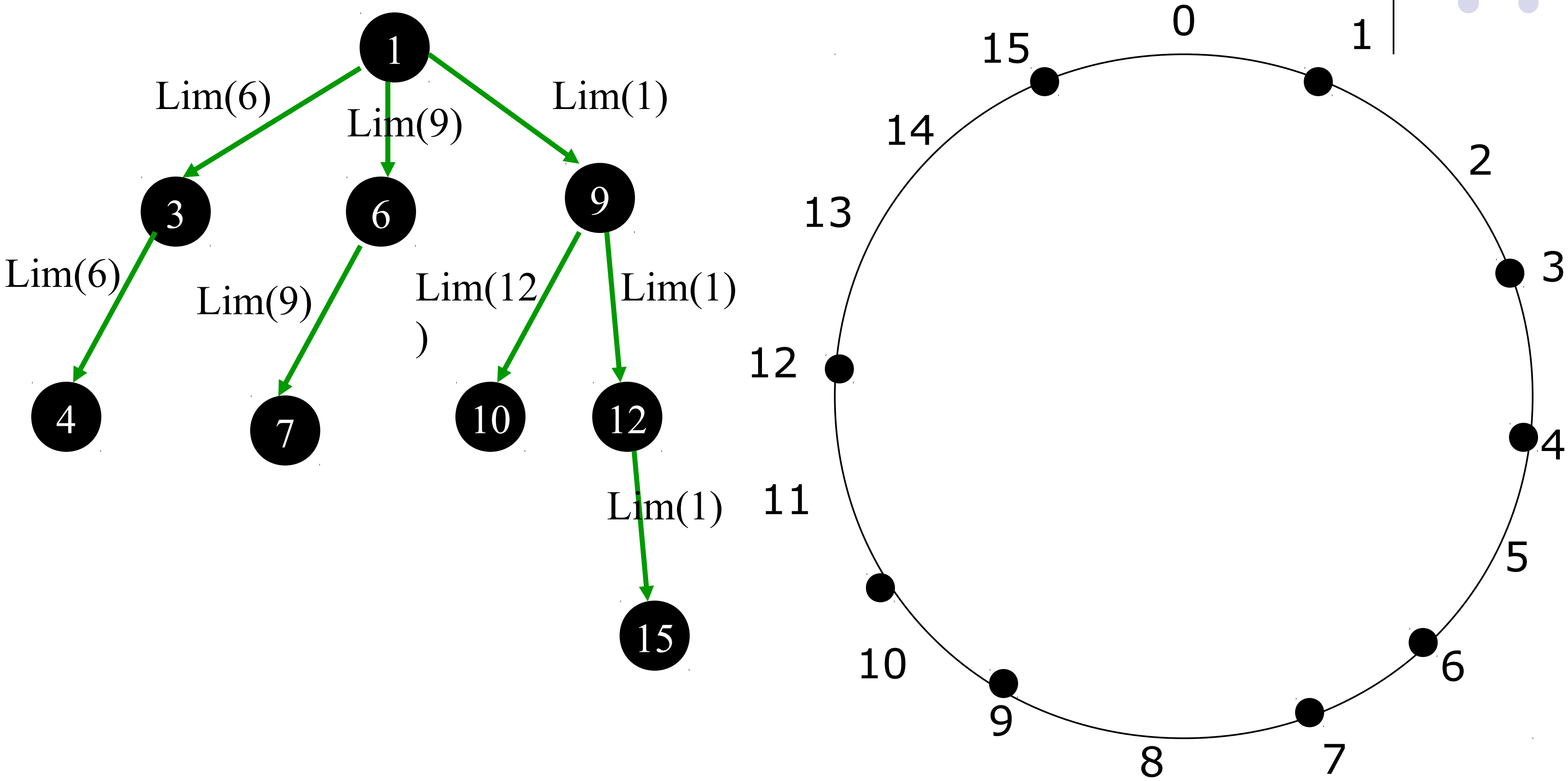




Efficient Broadcast Idea



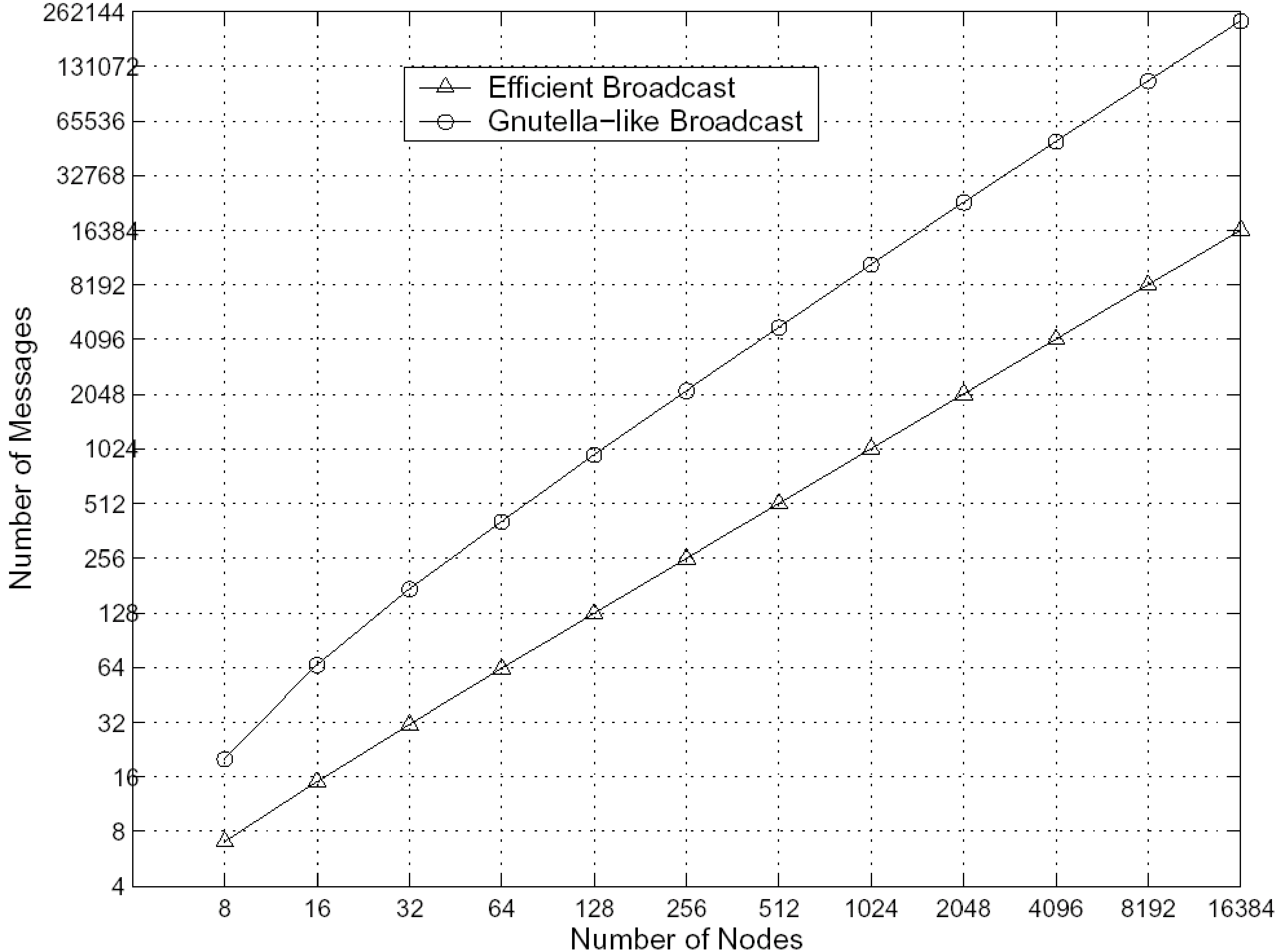
Efficient Broadcast Idea



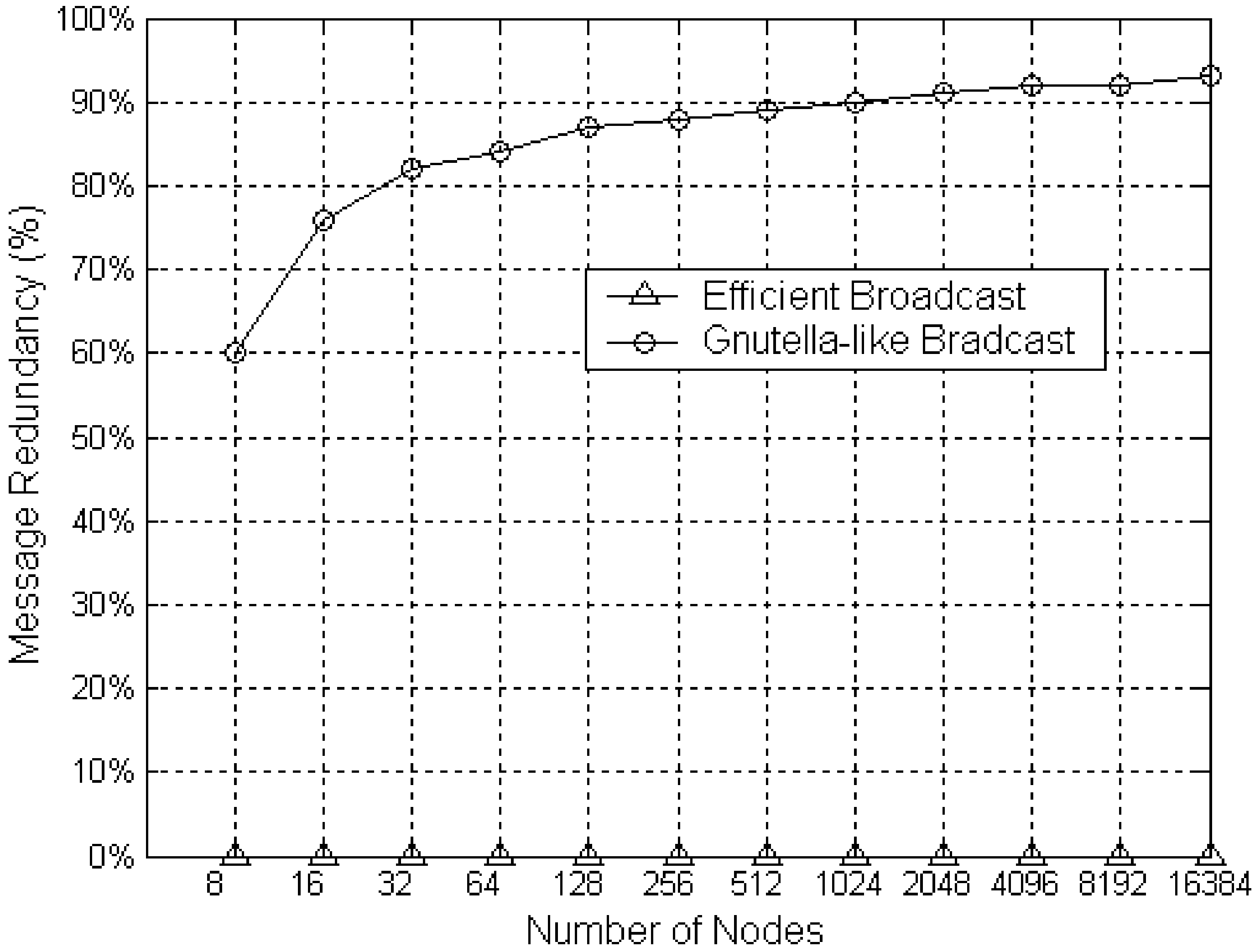
Cost Versus Guarantees

- Q: Is N-1 messages tolerable for any application?
- A1: Broadcast is a costly basic service, if necessary, broadcast wisely.
- A2: If less guarantees are desirable, prune or traverse the spanning tree differently.

Simulation Results (1/2)



Simulation Results (2/2)



Broadcast Contributions

- Presents an optimal algorithm for broadcasting in DHTs
- Relevance to research issues in state-of-the-art P2P systems:
  - Group communication
  - Complex queries

Conclusion

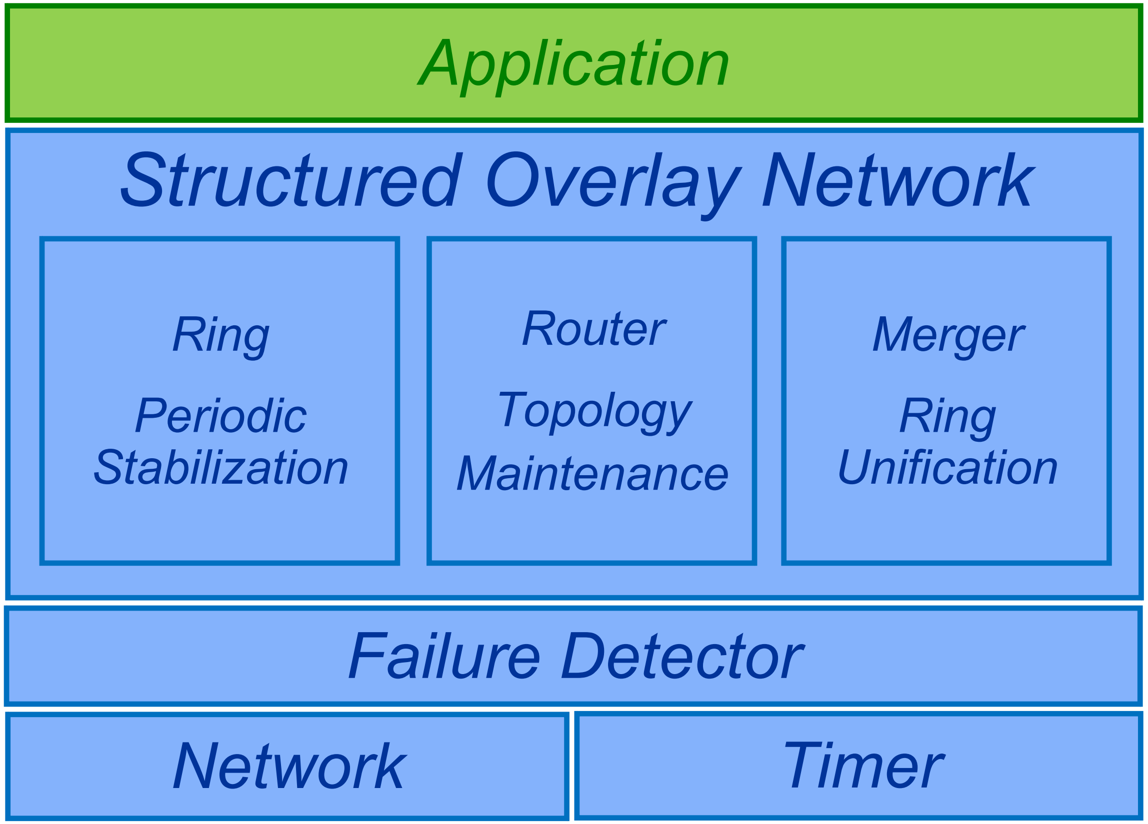
- By using the distributed k-ary search framework for the understanding, optimization and design of existing structured P2P systems with logarithmic performance properties, we were able to provide solutions to current research issues in state-of-the-art systems namely:
  - Lack of a common framework
  - Group communication
  - Complex queries
  - Cost of maintaining the structure

DHT as Application Infrastructure

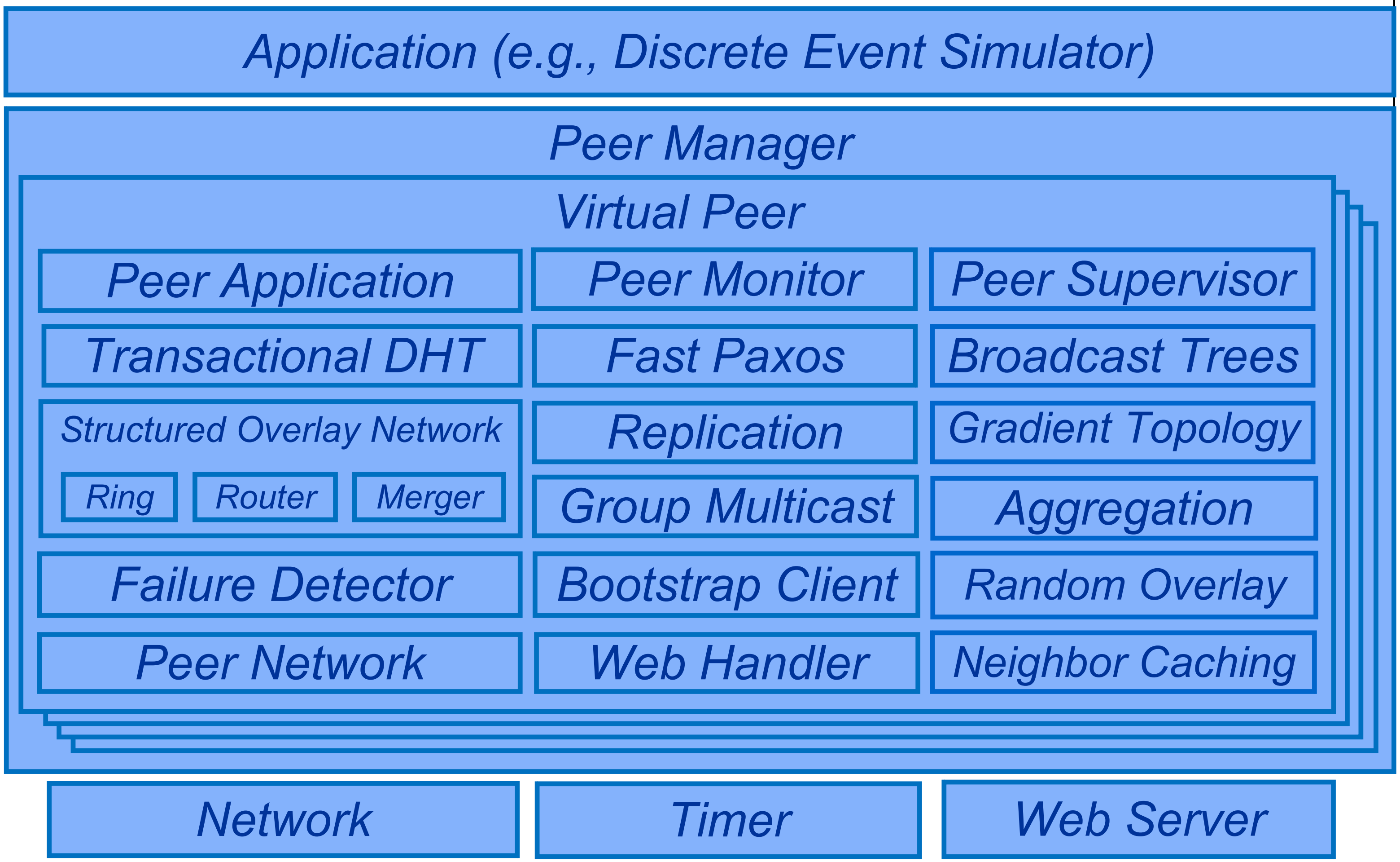
DHT as Application Infrastructure

- A DHT can be used as an infrastructure for building large-scale distributed applications
  - Basic services: storage, replication, transactions, broadcast, …
  - It is easy to build a scalable decentralized application on top
  - In the SELFMAN project we built a Distributed Wikipedia
- We start with a component model
  - Services and applications are concurrent components
  - We create a layered structure (like the project for SINF2345!)
  - Let us show how it is done by building our DHT from reusable components
- We built several applications using this architecture
  - Collaborative drawing (DeTransDraw), Distributed Wikipedia

- Now let us look inside our DHT's implementation
- We want to build the DHT from components



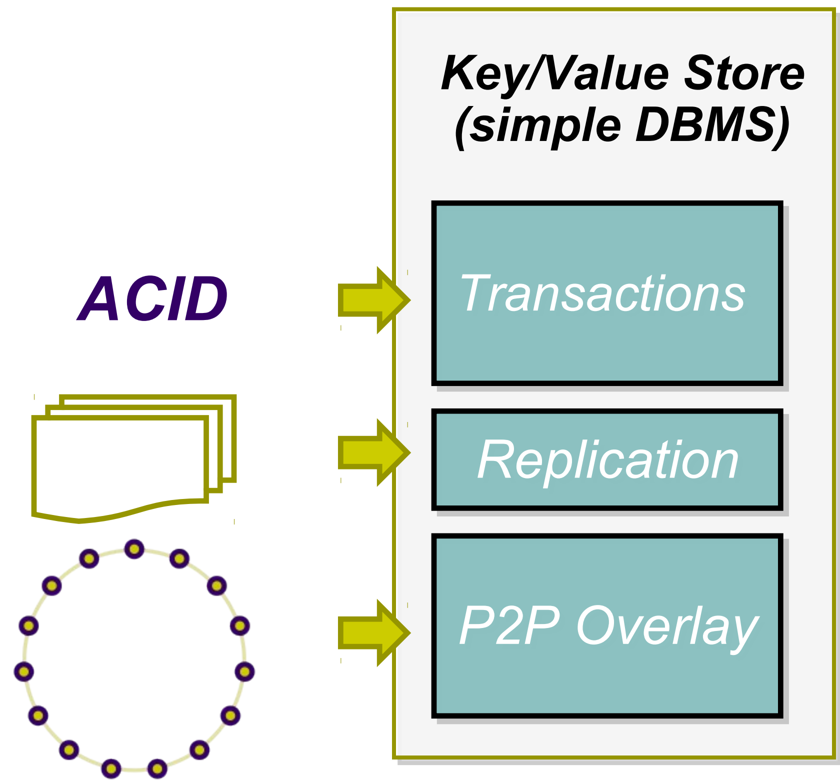
- DHT = Structured overlay network
- Reconstruct DHT with building blocks



- In the SELFMAN project we built a realistic application architecture: Scalaris and Beernet (www.ist-selfman.org)

Scalaris and Beernet

- Scalaris and Beernet are key-value stores developed in the SELFMAN project (www.ist-selfman.org) (*)
  - They provide transactions and strong consistency on top of loosely coupled peers using the Paxos uniform consensus algorithm for atomic commit
  - They are scalable to hundreds of nodes; with ten nodes they have similar performance as MySQL servers
  - Scalaris won first prize in the IEEE Scalable Computing Challenge 2008
- They are an example of a scalable application infrastructure



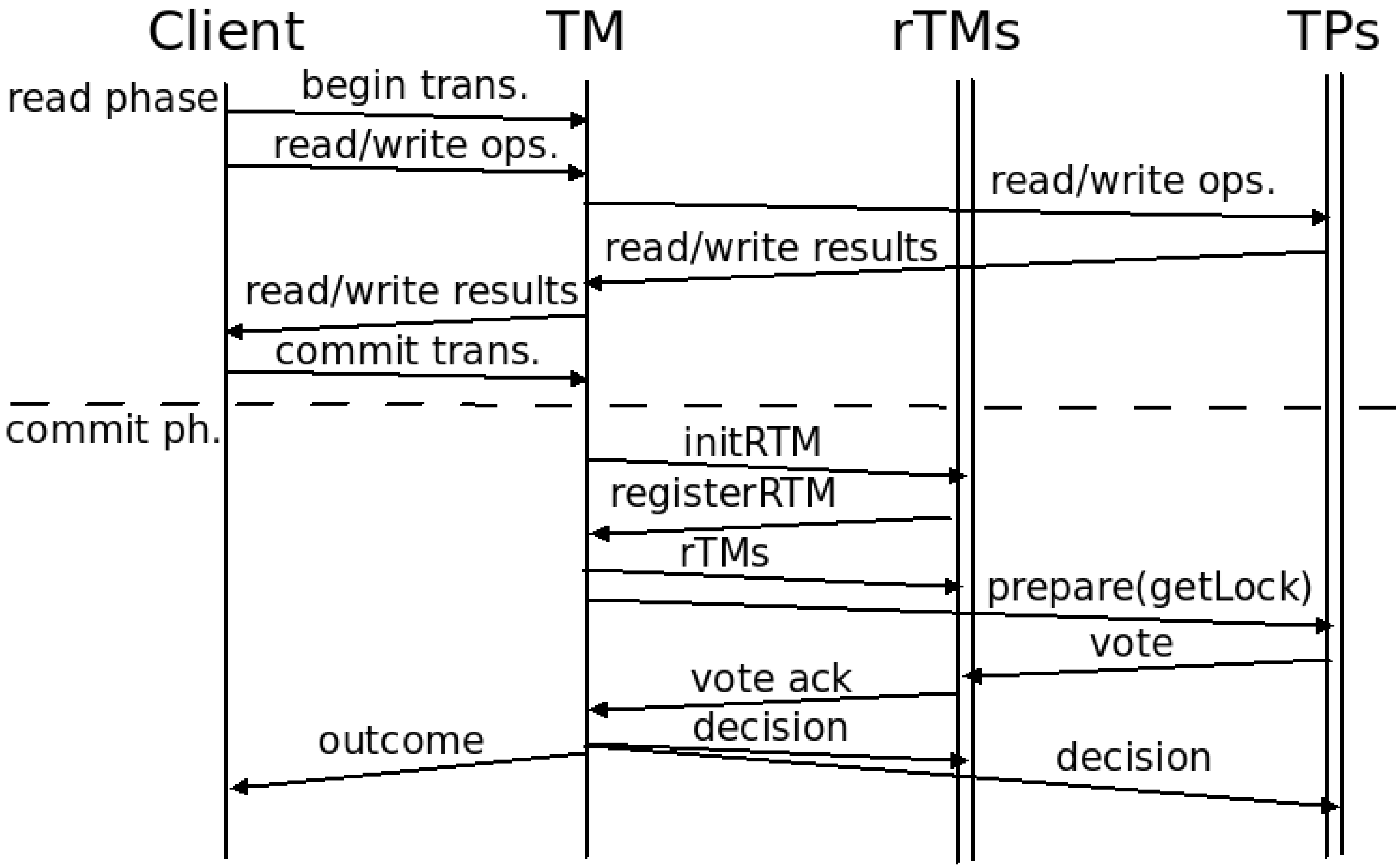
(*) *SELFMAN was coordinated by P. Van Roy*

DeTransDraw on Beernet for gPhone

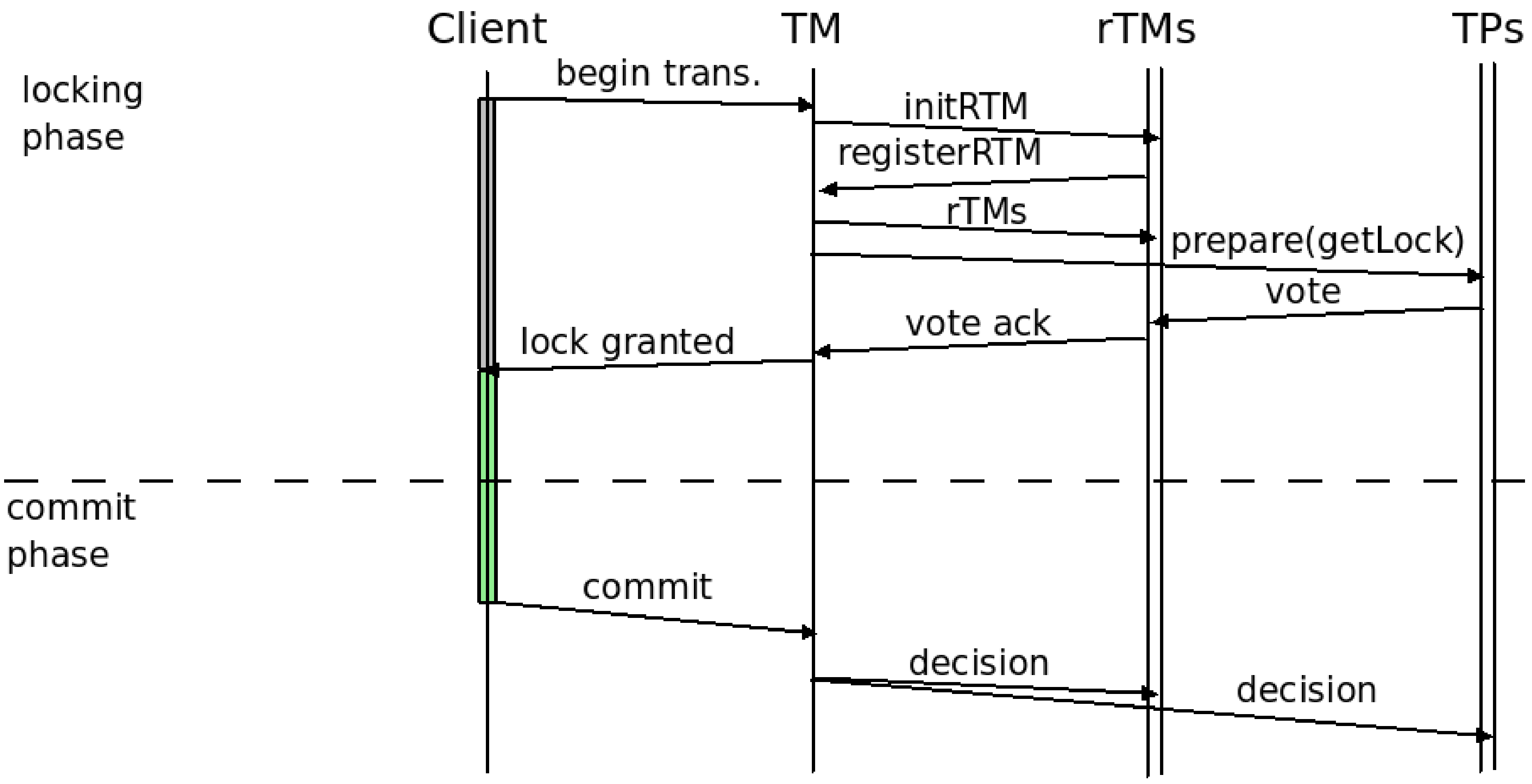
DeTransDraw

- DeTransDraw is a collaborative drawing application
  - Each user sees exactly the same drawing space
  - Users update the drawing space using transactions
  - For quick response time, the transaction is initiated concurrently with the display update
- Prototype application implemented on top of Beernet
  - Beernet implemented in Mozart, ported to gPhone with Android operating system

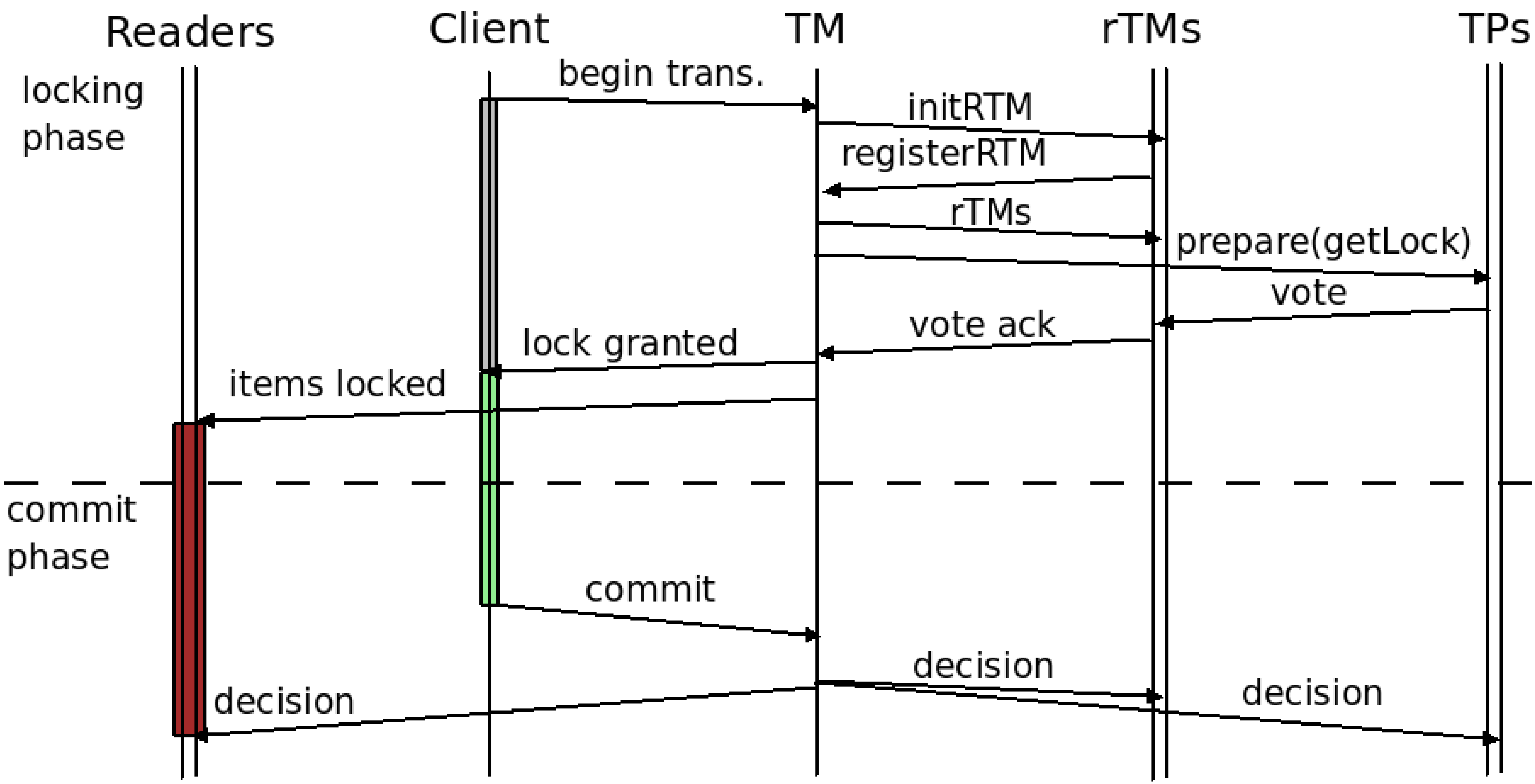
Paxos consensus protocol



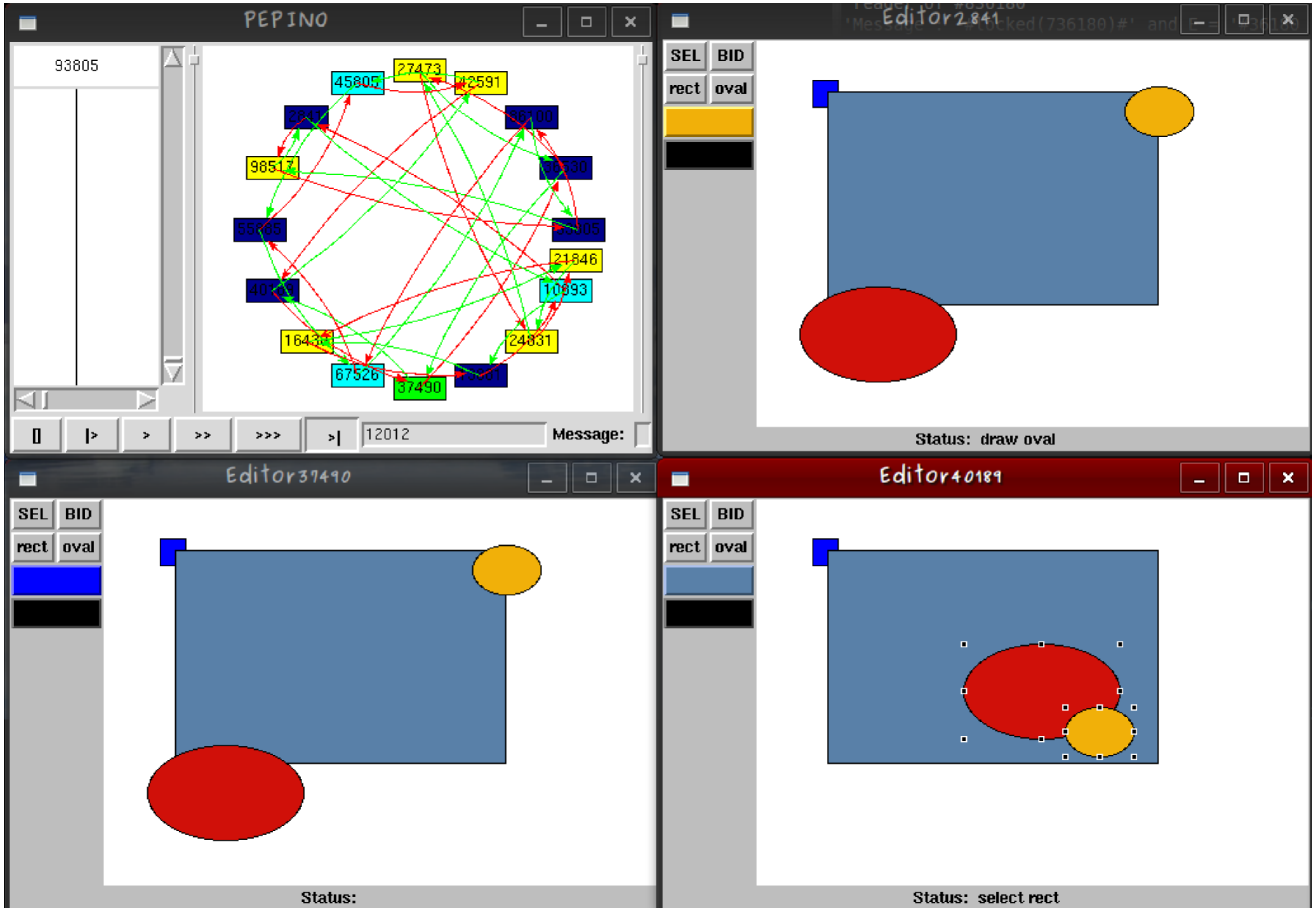
Eager Locking for Paxos



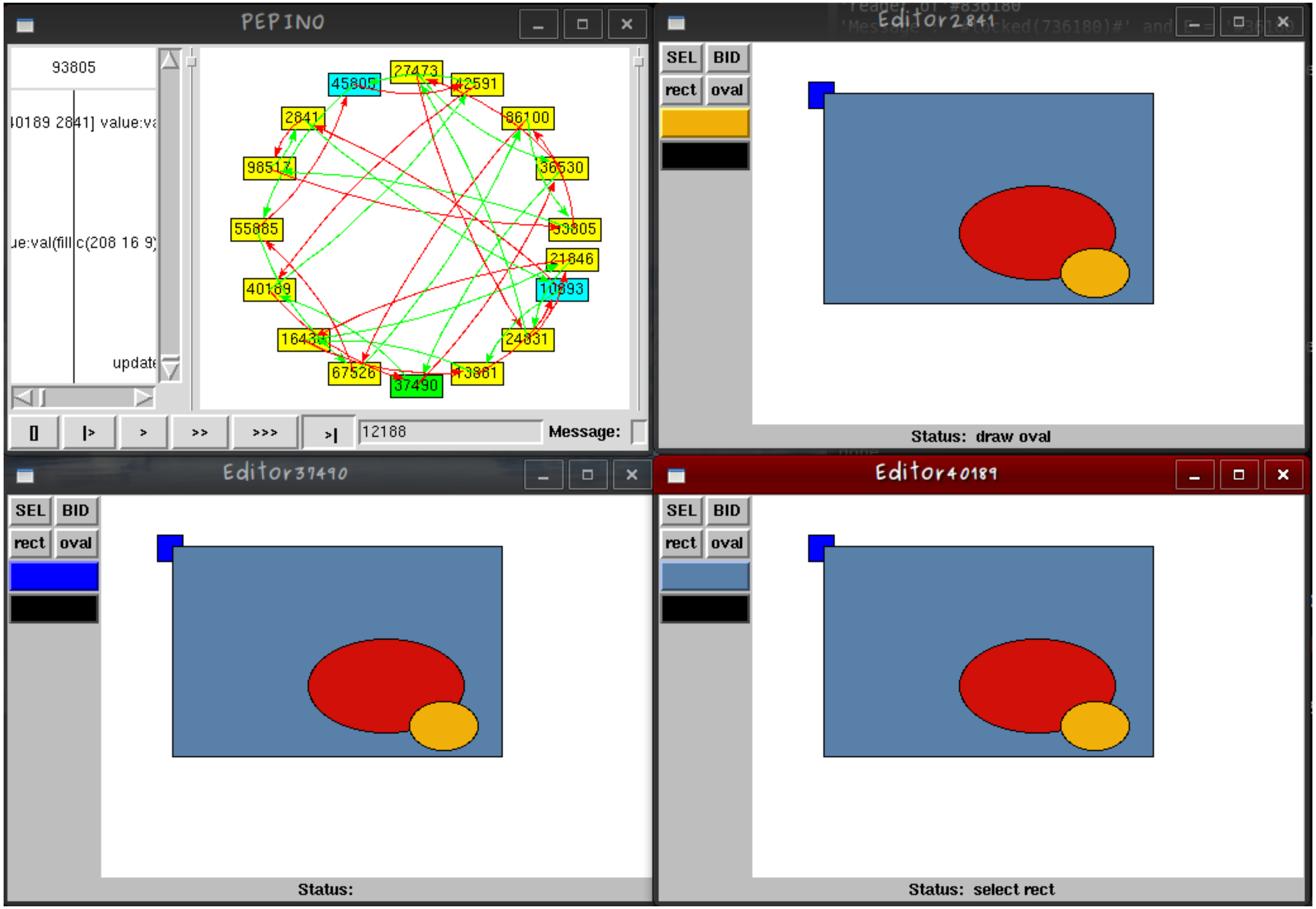
Notification Layer



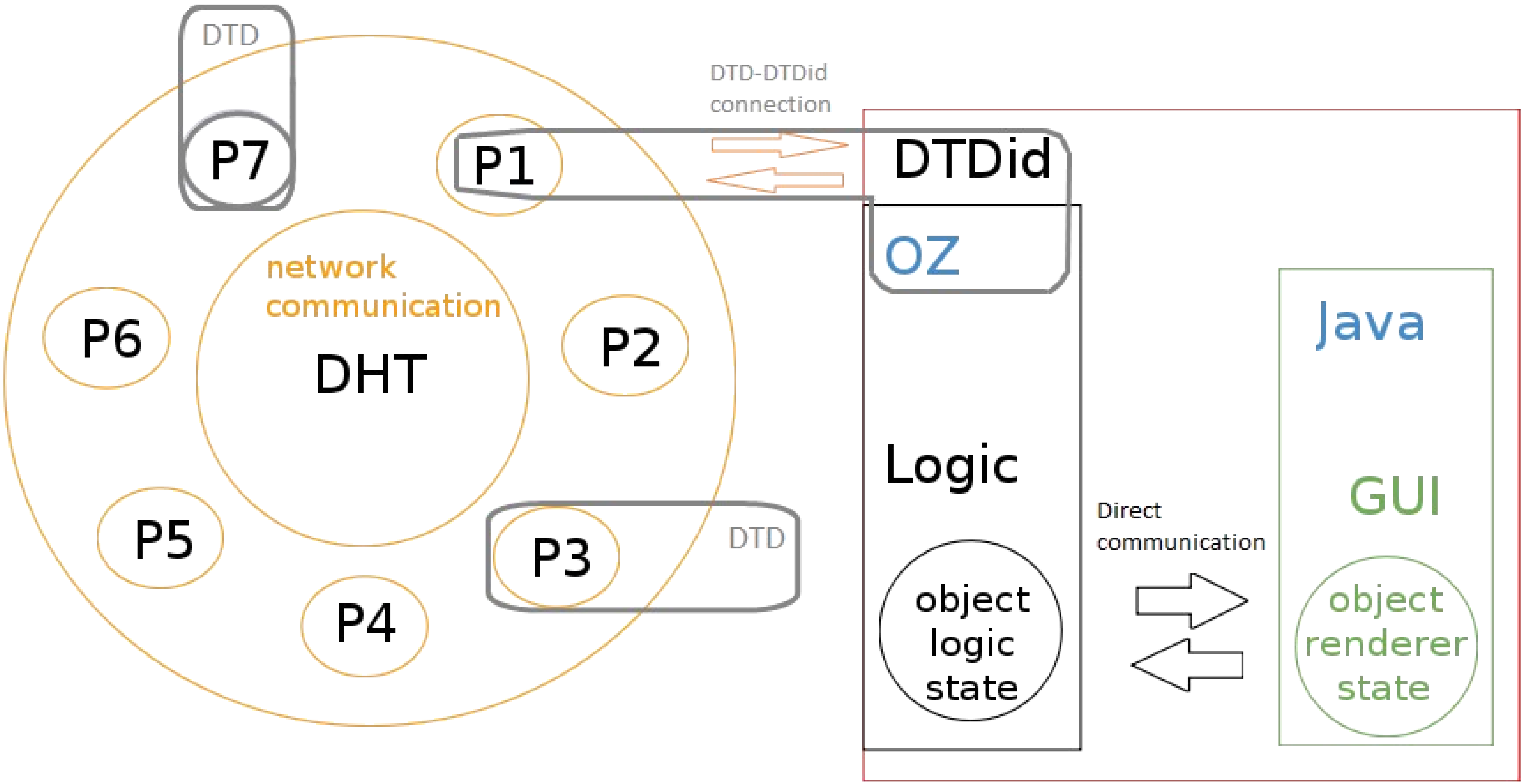
DeTransDraw – Getting Locks



DeTransDraw – Propagating Update



DTD and DTDid architecture



Distributed Wikipedia on Scalaris

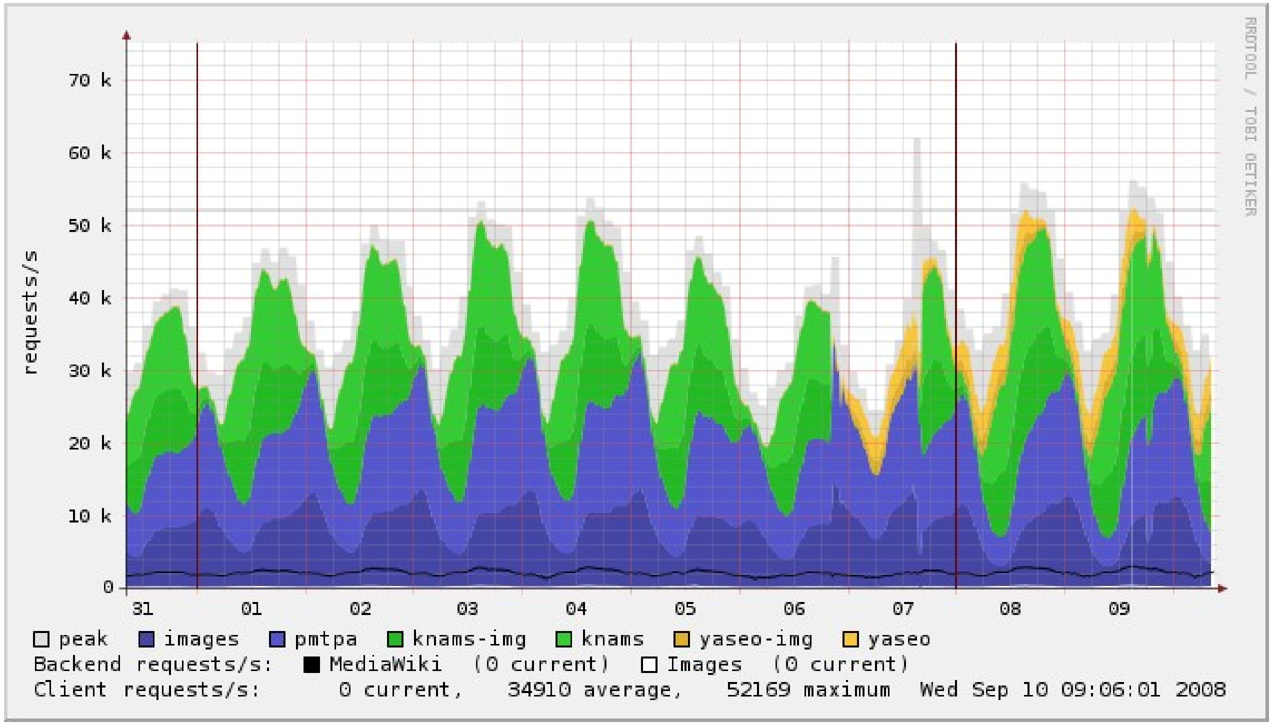
Wikipedia

Top 10 Web sites

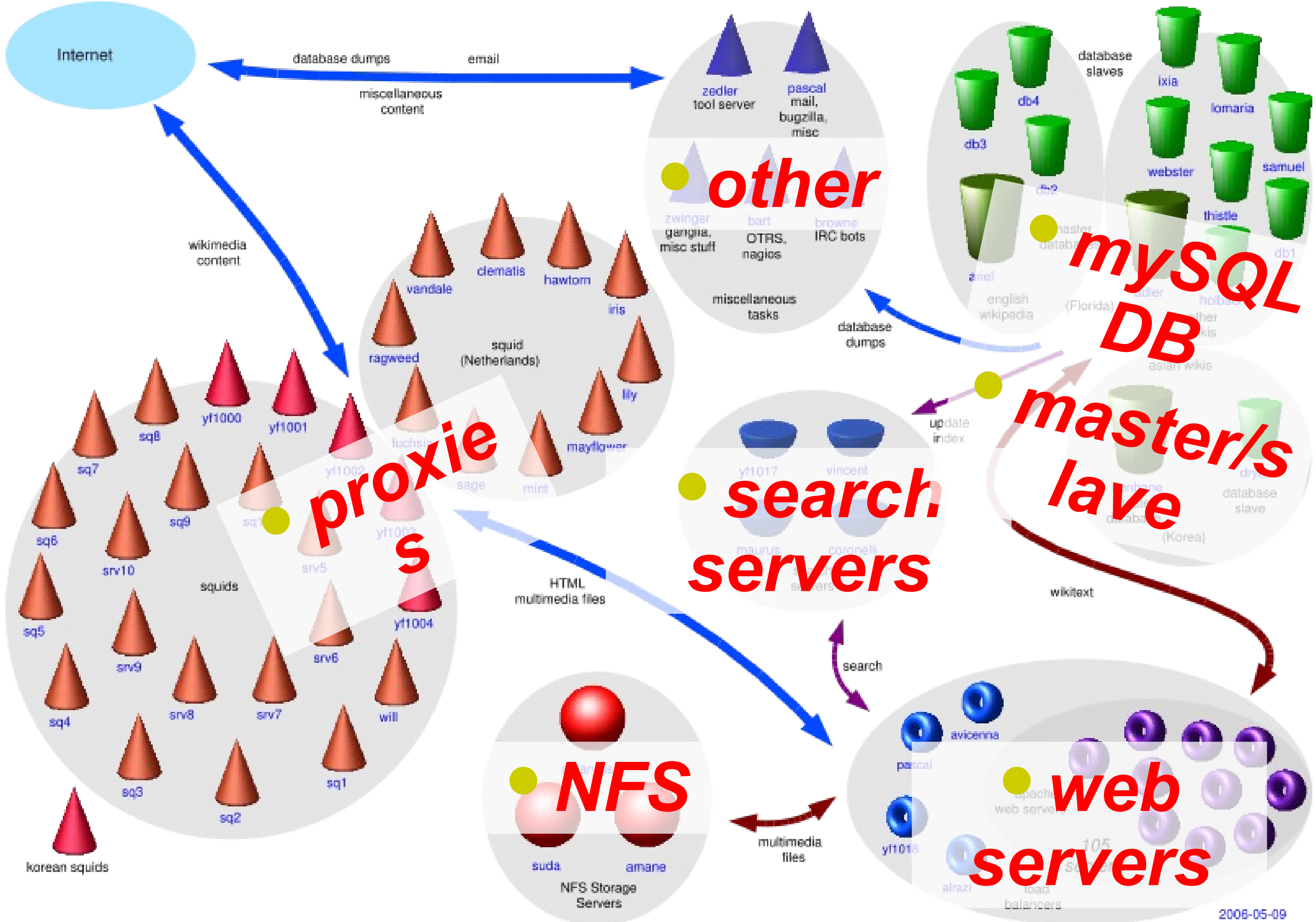
1. Yahoo!
2. Google
3. YouTube
4. Windows Live
5. Facebook
6. MSN
7. Myspace
8. **Wikipedia**
9. Blogger.com
10. Yahoo! カテゴリ

50.000 requests/sec

- 95% are answered by squid proxies
- **only 2,000 req./sec hit the backend**



The Wikipedia System Architecture



Data Model

Wikipedia
- SQL DB

Scalaris
- Key-Value Store

CREATE TABLE /$wgDBprefix*/page (*
page_id int unsigned NOT
NULL auto_increment,
page_namespace int NOT NULL,
...

Map Relations to Key-Value Pairs
- (Title, List of Versions)
- (CategoryName, List of Titles)
- (Title, List of Titles) //Backlinks

Data Model (Simple Query Layer)

```
void updatePage(string title, int oldVersion, string newText)
{
    //new transaction
    Transaction t = new Transaction();
    //read old version
    Page p = t.read(title);
    //check for concurrent update
    if(p.currentVersion != oldVersion)
        t.abort();
    else{
        //write new text
        t.write(p.add(newText));
        //update categories
        foreach(Category c in p)
            t.write(t.read(c.name).add(title));
        //commit
        t.commit();
    }
}
```

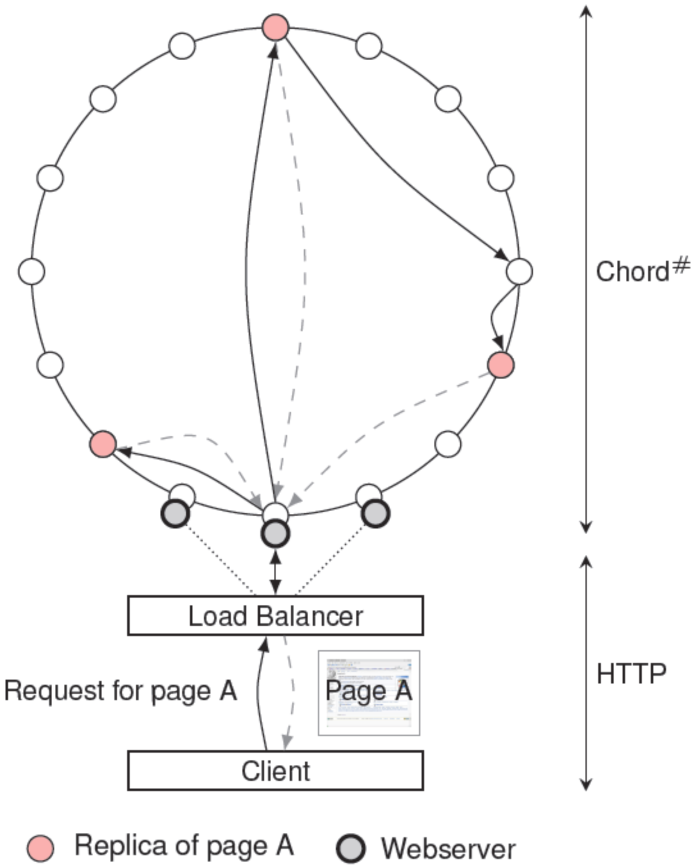
Self-* Architecture

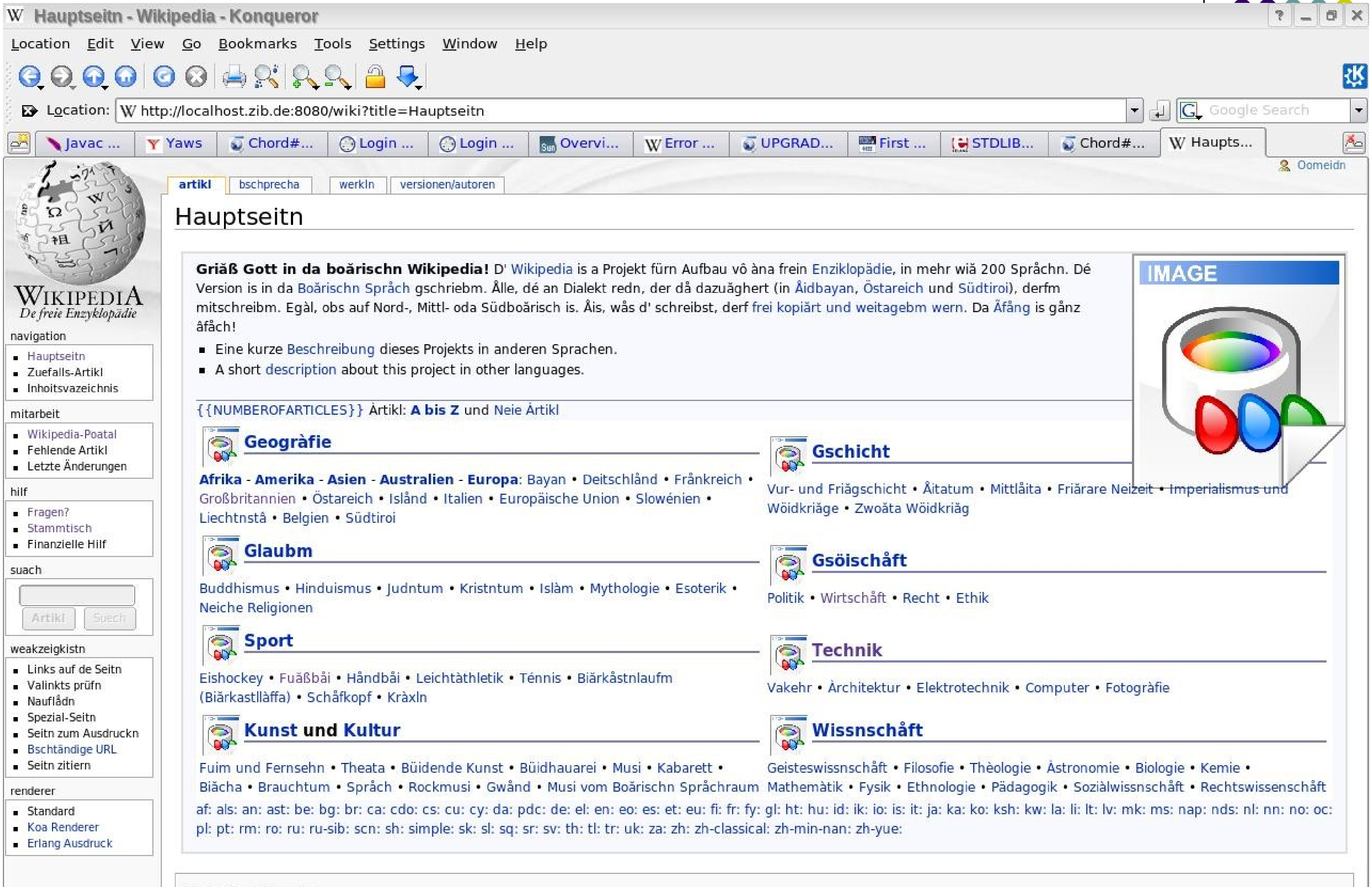
Database:

- Chord#
- Mapping
  - Wiki -> Key-Value Store

Renderer:

- Java
  - Tomcat
  - Plog4u
- Jinterface
  - Interface to Erlang




Hauptseitn - Wikipedia - Konqueror
Location Edit View Go Bookmarks Tools Settings Window Help
Location: http://localhost.zib.de:8080/wiki?title=Hauptseitn
Google Search
Javac ... Yaws Chord#... Login ... Login ... Overvi... Error ... UPGRAD... First ... STDLIB... Chord#... Haupts...
Oomeidn
artikl bschprecha werkln versionen/autoren
Hauptseitn
Griaß Gott in da boarischn Wikipedia! D' Wikipedia is a Projekt fürn Aufbau vô àna frein Enziklopädie, in mehr wiä 200 Språchn. Dé Version is in da Boarischn Språch gschriebm. Ålle, dé an Dialekt redn, der då dazuaghert (in Åidbayan, Östareich und Südtiroi), derfm mitschreibm. Egàl, obs auf Nord-, Mittl- oda Südboarisch is. Åis, wås d' schreibst, derf frei kopiärt und weitagebm wern. Da Åfång is gånz åfåch!
Eine kurze Beschreibung dieses Projekts in anderen Sprachen.
A short description about this project in other languages.
{{NUMBEROFARTICLES}} Artikl: A bis Z und Neie Artikl
IMAGE
Geogràfie
Afrika - Amerika - Asien - Australien - Europa: Bayan • Deitschlånd • Frånkreich • Großbritannien • Östareich • Islånd • Italien • Europäische Union • Slowénien • Liechtnstâ • Belgien • Südtiroi
Gschicht
Vur- und Friägschicht • Åitatum • Mittlåita • Friärare Neizeit • Imperialismus und Wöidkriäge • Zwoàta Wöidkriäg
Glaubm
Buddhismus • Hinduismus • Judntum • Kristntum • Islàm • Mythologie • Esoterik • Neiche Religionen
Gsöischåft
Politik • Wirtschåft • Recht • Ethik
Sport
Eishockey • Fuäßbåi • Håndbåi • Leichtàthletik • Ténnis • Biärkåstnlaufm (Biärkastllåffa) • Schåfkopf • Kràxln
Technik
Vakehr • Àrchitektur • Elektrotechnik • Computer • Fotogràfie
Kunst und Kultur
Fuim und Fernsehn • Theata • Büidende Kunst • Büidhauarei • Musi • Kabarett • Biächa • Brauchtum • Språch • Rockmusi • Gwånd • Musi vom Boarischn Språchraum
Wissnschåft
Geisteswissnschåft • Filosofie • Thèologie • Åstronomie • Biologie • Kemie • Mathemàtik • Fysik • Ethnologie • Pädagogik • Soziàlwissnschåft • Rechtswissenschåft
af: als: an: ast: be: bg: br: ca: cdo: cs: cu: cy: da: pdc: de: el: en: eo: es: et: eu: fi: fr: fy: gl: ht: hu: id: ik: io: is: it: ja: ka: ko: ksh: kw: la: li: lt: lv: mk: ms: nap: nds: nl: nn: no: oc: pl: pt: rm: ro: ru: ru-sib: scn: sh: simple: sk: sl: sq: sr: sv: th: tl: tr: uk: za: zh: zh-classical: zh-min-nan: zh-yue:
Kategorie: Wikipedia
WIKIPEDIA
De freie Enzyklopädie
navigation
Hauptseitn
Zuefalls-Artikl
Inhoitsvazeichnis
mitarbeit
Wikipedia-Poatal
Fehlende Artikl
Letzte Änderungen
hilf
Fragen?
Stammtisch
Finanzielle Hilf
suach
Artikl Suech
weakzeigkistn
Links auf de Seitn
Valinkts prüfn
Naufladn
Spezial-Seitn
Seitn zum Ausdruckn
Bschtändige URL
Seitn zitiern
renderer
Standard
Koa Renderer
Erlang Ausdruck

IEEE Scale Challenge 2008

Live Demos

- Bavarian
- Simple English

- Browsing
- Editing with full History
- Category-Pages

- Deployments
  - Planet-Lab
    - 20 nodes
  - Cluster
    - 320 nodes in Berlin


TCSC
IEEE
IEEE Technical Committee on Scalable Computing
A Transactional Scalable Distributed Data Store: Wikipedia on a DHT
T. Schuett, M. Moser, S. Plantikow, F. Schintke and A. Reinefeld
TCSC SCALE Award
of the first IEEE International Scalable Computing Challenge (SCALE-08) presented at the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid-08) in Lyon, France, May 19-22, 2008.
The award is sponsored by the IEEE Technical Committee on Scalable Computing (TCSC) and endorsed by the IEEE Technical Committee on Parallel Processing (TCPP).
TCSC Awards Coordinators
Manish Parashar and Rajkumar Buyya
TCSC Chair
Laurence T. Yang
IEEE
IEEE COMPUTER SOCIETY

Conclusions

- DHT is a practical base to build scalable decentralized applications
  - DHT provides efficient communication and storage
  - Important services, including broadcast and transactions can be built on top of a DHT
  - We built two implementations, Scalaris and Beernet
    - Both are available as open-source systems
- We built several applications in the SELFMAN project
  - DeTransDraw with Beernet on gPhone
  - Distributed Wikipedia with Scalaris (won a prize!)

Stub Slides

Notation

Ιδεντιφιερσπαχε I

$$I = \{0, 1, \ldots, N-1\}$$

$N = k^L$ (σιζεοφ τηεσπαχε)

$\alpha \oplus \beta \equiv (a+b)$ μοδ N

Λεϖελσανδ ςιεωσ

νυμβεροφ λεϖελσ λογ$_k$ $N \equiv L$

ατλεϖελ1, $V^1 = I_0^1 \cup I_1^1 \cup \ldots \cup I_{k-1}^1$

$$I_i^1 = [x_i^1, x_{i+1}^1[, \quad x_i^1 = n \oplus i \frac{N}{k}$$

Levels and views

Λεϖελανδ ςιεωσ

number of levels $\log_{\kappa} N \equiv \Lambda$

at level 1, $\varsigma^{1} = I_0^1 \cup I_1^1 \cup \ldots \cup I_{\kappa-1}^1$

$$I_\iota^1 = [\,\xi_\iota^1, \xi_{\iota+1}^1[, \quad \xi_\iota^1 = \nu \oplus \iota \frac{N}{\kappa} \text{ for } 0 \le \iota \le \kappa - 1$$

at level $2 \le \lambda \le \Lambda$, $\varsigma^{\lambda} = I_0^\lambda \cup I_1^\lambda \cup \ldots \cup I_{\kappa-1}^\lambda$

$$I_0^\lambda = [\,\xi_0^\lambda, \xi_1^\lambda[, \ldots, I_{\kappa-1}^\lambda = [\,\xi_{\kappa-1}^\lambda, \xi_0^{\lambda-1}[$$

$$I_\iota^\lambda = [\,\xi_\iota^\lambda, \xi_{\iota+1}^\lambda[, \quad \xi_\iota^\lambda(\nu) = \nu \oplus \iota \frac{N}{\kappa^\lambda} \text{ for } 0 \le \iota \le \kappa - 1$$

Responsibility

At each level V^l there is a node $R(I_i^l)$ that is responsible for each interval I_i^l

let x be a node, $S(x)$ is the first node encountered in $[x, x[$

For an arbitrary node n and an arbitrary level l, the responsible for I_i^l is $S(x_i^l)$

Each node n is itself responsible for I_i^l for any l

$Observe\ \ that\ S(x_i^l)\ is\ a\ \ function\ \ of\ \ n$

$can\ be\ computed\ by\ any\ node$

Routing table

At each level ς^{λ} there is a node $P(I_{\iota}^{\lambda})$ that is responsible for

$PT_{v} : \Lambda \rightarrow \mathrm{K} \rightarrow \mathrm{I}$

$+$ a predecessor pointer $\Pi(v)$

Λ : set of levels

$\mathrm{K} : \{1,.., \kappa\}$

I : set of nodes

Node insertion I

$p : \text{πρεδεχεσσορποιντερ}$

$Insert : \text{νοδεστοβε ινσερτε}$

$n_j : \text{νοδε τοβε ινσερτεδ}$

$\textbf{Εμπτψ Νετωορκ}(n_j)$

$\text{φορεαχη}\, 1 \le l \le L, i \in \text{Κ δο}$

$$RT(l)(i) := n_j$$

$p := n_j$

$Insert := nil$

Node insertion II

ΝονΕμπτψ Νετωορκωιτηασινγλενοδε(n)

Node n computes the RT of n_j :

for $l \in L$ do

$$RT_{n_j}(l)(0) := n_j$$

for $l \in L, i \in \mathrm{K} / \{0\}$ do

if $x_i^l(n_j) \in [n_j, n[$ then

$$RT_{n_j}(l)(i) := n$$

else $RT_{n_j}(l)(i) := n_j$

$$p_{n_j} := n$$

n adapts its RT_n in a similar way

Node insertion III

ΝονΕμπτψ Νετωορκωιτημυλτιπλενοδεσ

Νοδε n χομπυτεστηεΡΤ οφ n_j :

φορ $l \in L$ δο $RT_{n_j}(l)(0) := n_j$

φορ $l \in L, i \in \mathrm{K}/\{0\}$ δο

ιφ $x_i^l(n_j) \in [n_j, n[$ τηεν $RT_{n_j}(l)(i) := n$

ελσειφ $x_i^l(n_j) \in [p, n_j[$ τηεν $RT_{n_j}(l)(i) := n_j$

ελσειφ $x_i^l(n_j) \in [n, p[$ τηεν $RT_{n_j}(l)(i) := RT_n(l)(i)$ $(approximation)$

$p_{n_j} := p_n$

n αδαπτσιτσ RT_n ιν ασιμιλαρ ωαψ

$p_n := n_j$

Node insertion IV

- Node insertion is an atomic operation
- Coordinated and serialized by *n*
- *p* is informed of n_j
- Other insertion requests to *n* wait
- *n* is the coordinator of 2PC
- Clients p and n_j

Correction on use

- Look-up or insert messages from node *n* to node *n'*
- Add the following to the message
  - *i* (interval) and *l* (level)
- Node *n'* can compute : $x_i^l(n)$
- Node *n'* maintains a list of predecessors *BL*