CLD
From P2P to Key-Value Stores

Davide Frey
ASAP Team, INRIA Rennes

Gossip in Key Value Stores

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Gossip (Wikipedia)

Gossip consists of casual or idle talk of any sort, sometimes (but not always) slanderous and/or devoted to discussing others.

While gossip forms one of the oldest and (still) the most common means of spreading an

reputation for the intro o the

information thus transr

Reliable way of spreading information

Epidemic (Wikipedia)

In epidemiology, an **epidemic** is a disease that appears as new cases in a given human population, during a given period, at a rate that substantially exceeds what is "expected".

Non-biological usage:

The term is often use

widespread and growi

Efficient way of spreading something

Gossip/epidemics in distributed computing

Replace

- people by computers (nodes or peers),
- words by data

We retain

- Gossip: peerwise exchange of information
- Epidemic: wide and exponential spread

Refer to gossip in the following

Why Gossip

Scenario:

- Very Large scale Systems
- Lots of data
- Continuous Changes

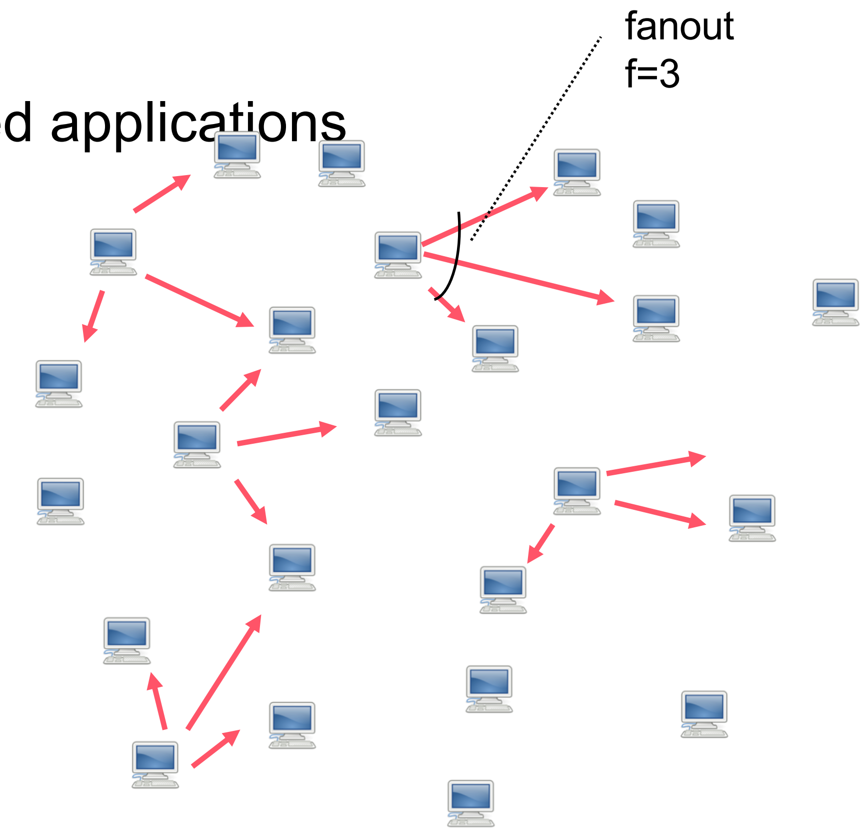
Gossip:

- Peer to peer communication: no unique point of failure
- Eventual convergence
- Probabilistic nature

Gossip / Epidemic Protocols

Fundamental tool for decentralized applications

- Completely decentralized
- Periodic pairwise exchanges
- Some form of randomness



Applications of Gossip

Consistency Management
[Demers &al, PODC

Epidemic disseminatio
Bimodal Multicast [Birman&al, ACM
[Kermarrec&al, IEEETPDS
Lpbcast [Eugster&al DSN01, ACM
Stream[Patel & al, NCA 20

Content-based search
Vicinity[Voulgaris & Steen,Euro-Par 05]
VoroNet [Beaumont & al, IPDPS 07]
RayNet[Beaumont & al, OPODIS 07]

Aggregation
[Jelasity&al, ACM TOCS 05]
Astolabe [Birman & al, 2003]

Slicing
[Jelasity, Kermarrec, P2P06]
[Fernandez & al, ICDCS07]

Overlay maintenance
Lpbcast [Eugster & al,ACM TOCS 03]
Cyclon[Voulgaris& al, 2005]
Newscats[Jelasity & al, 2003]

Publish-subscribe
Sub-2-Sub [Voulageris & al, IPTPS06]
Tera[Bald

Clustering
Vicinity, Jstream, Tman, Gossple

Streaming
BAR Gossip [Li & al, OSDI06]
Middleware 2009]

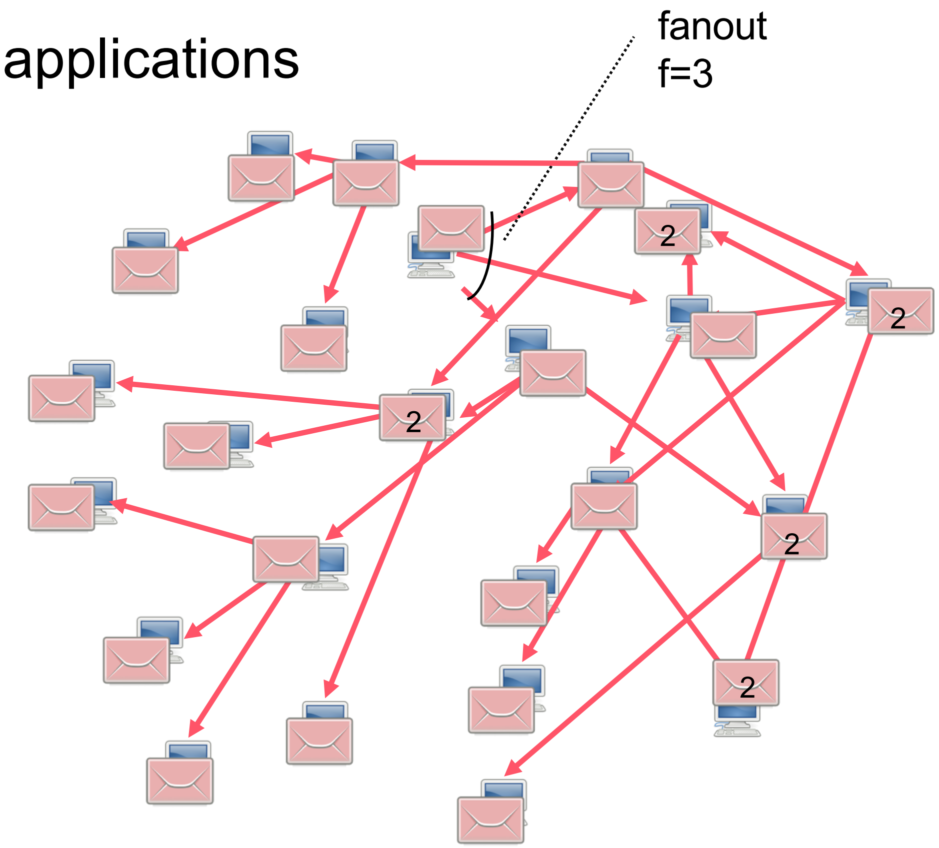
Secure Sampling
Brahms [Bortnikov & al, 08]

Recommendation
Gossple[Bertier & al, Middleware 2010]
WhatsUp[Boutet & al, IPDPS 2013]

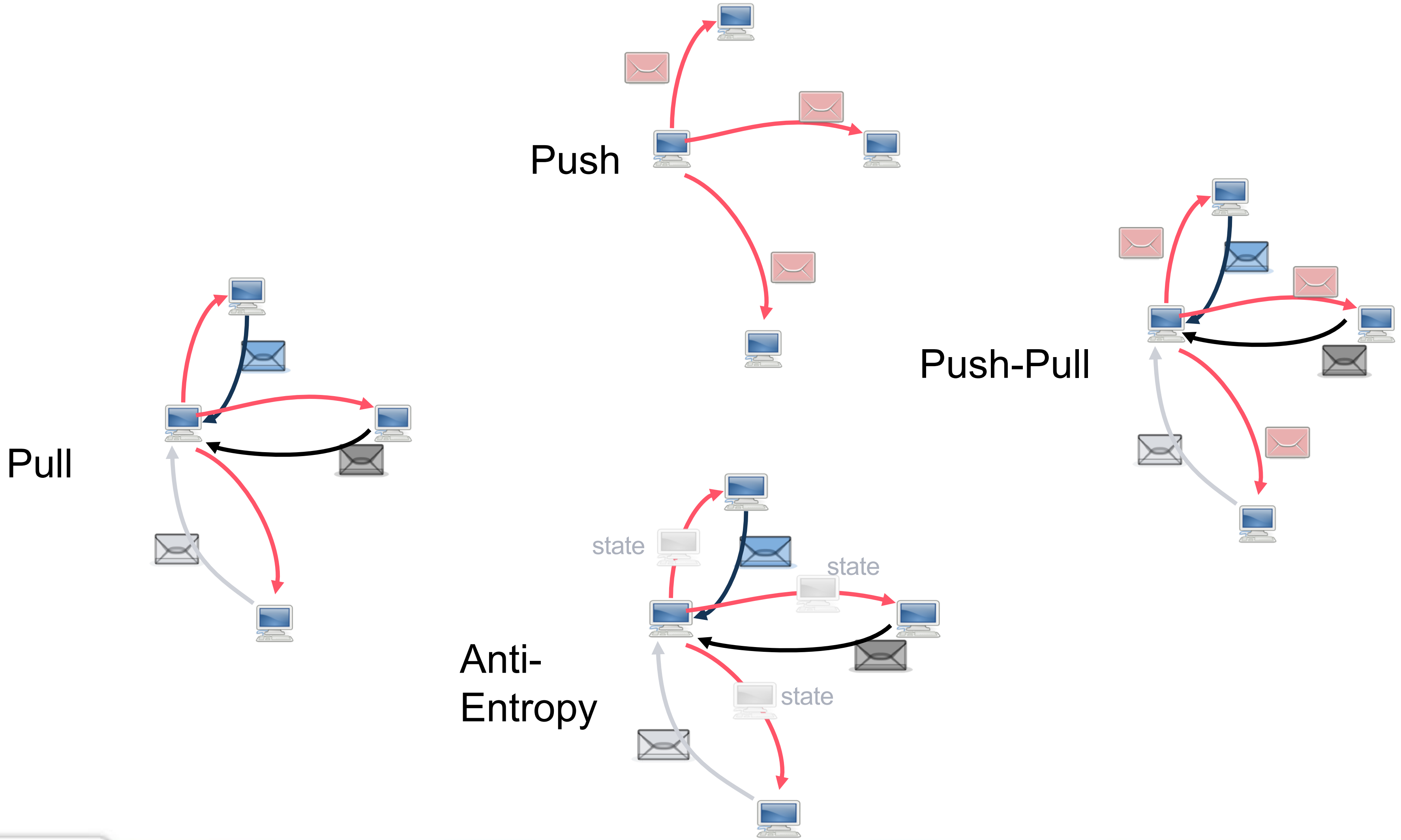
Data Dissemination

Fundamental tool for decentralized applications

- Data Dissemination



Gossip Variants

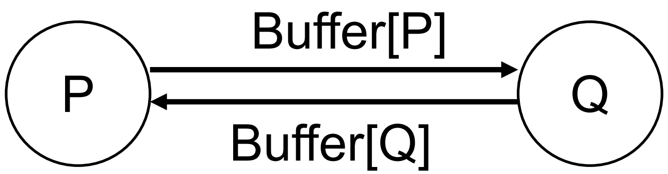


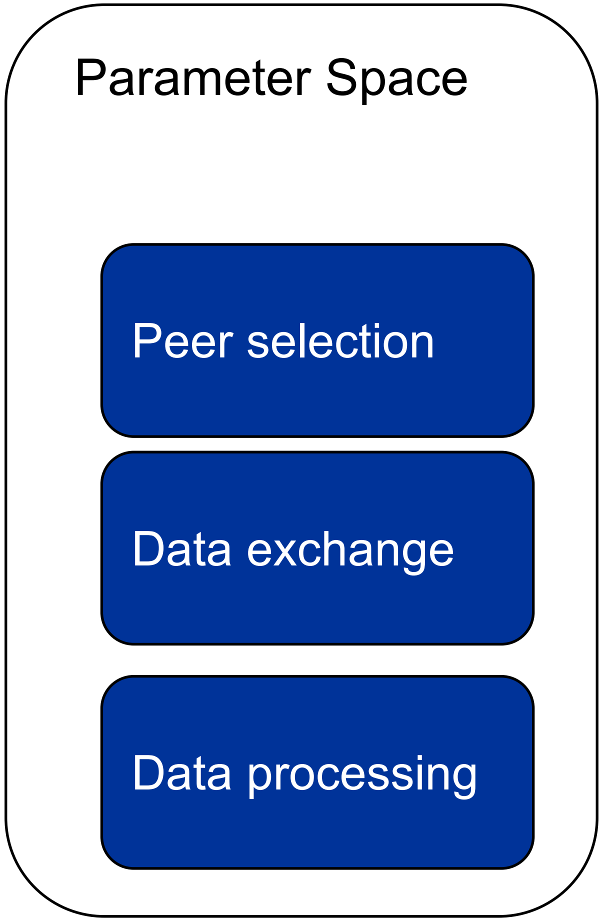
Generic Gossip Protocol

Each node maintains a set of neighbours (c entries)

Periodic peerwise exchange of information

Each process runs an active and passive threads





Generic Gossip Protocol

Active Cycle

Periodically

- Select a/some peer(s) p
- Select some data D
- Send D to p

Peer selection

Data exchange

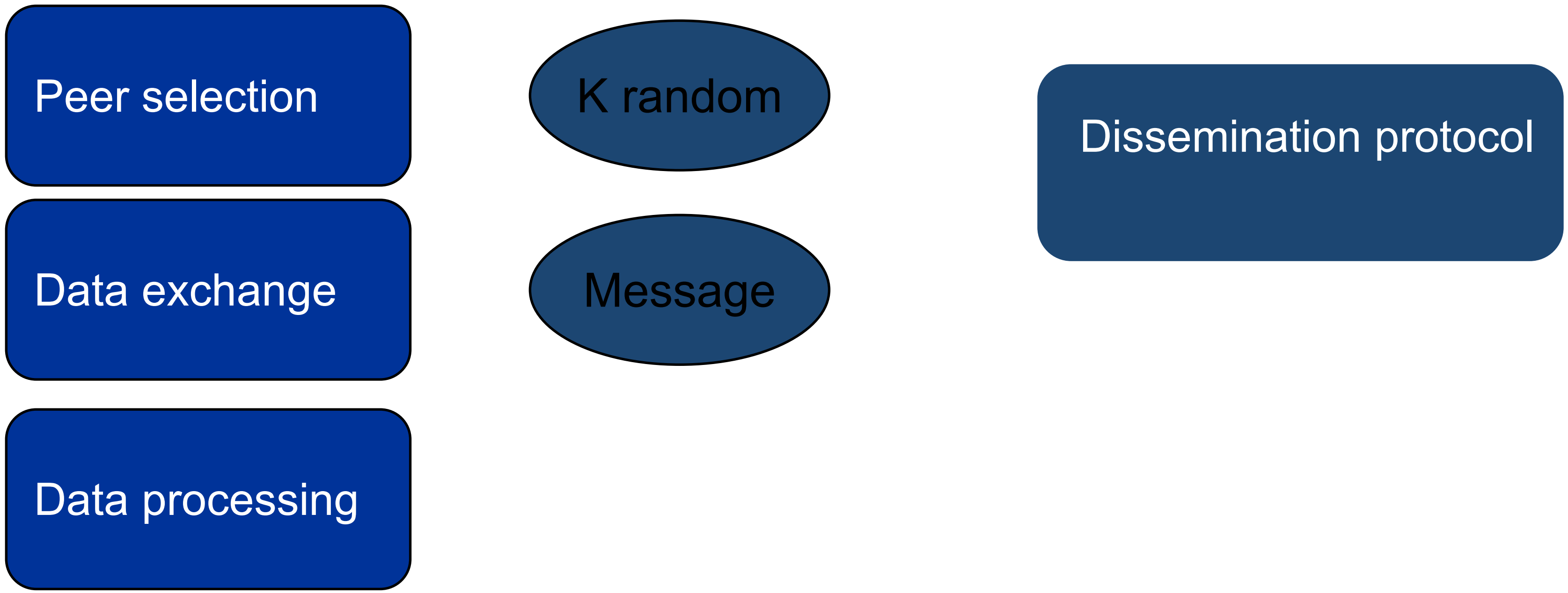
Passive Cycle

Upon message M from p

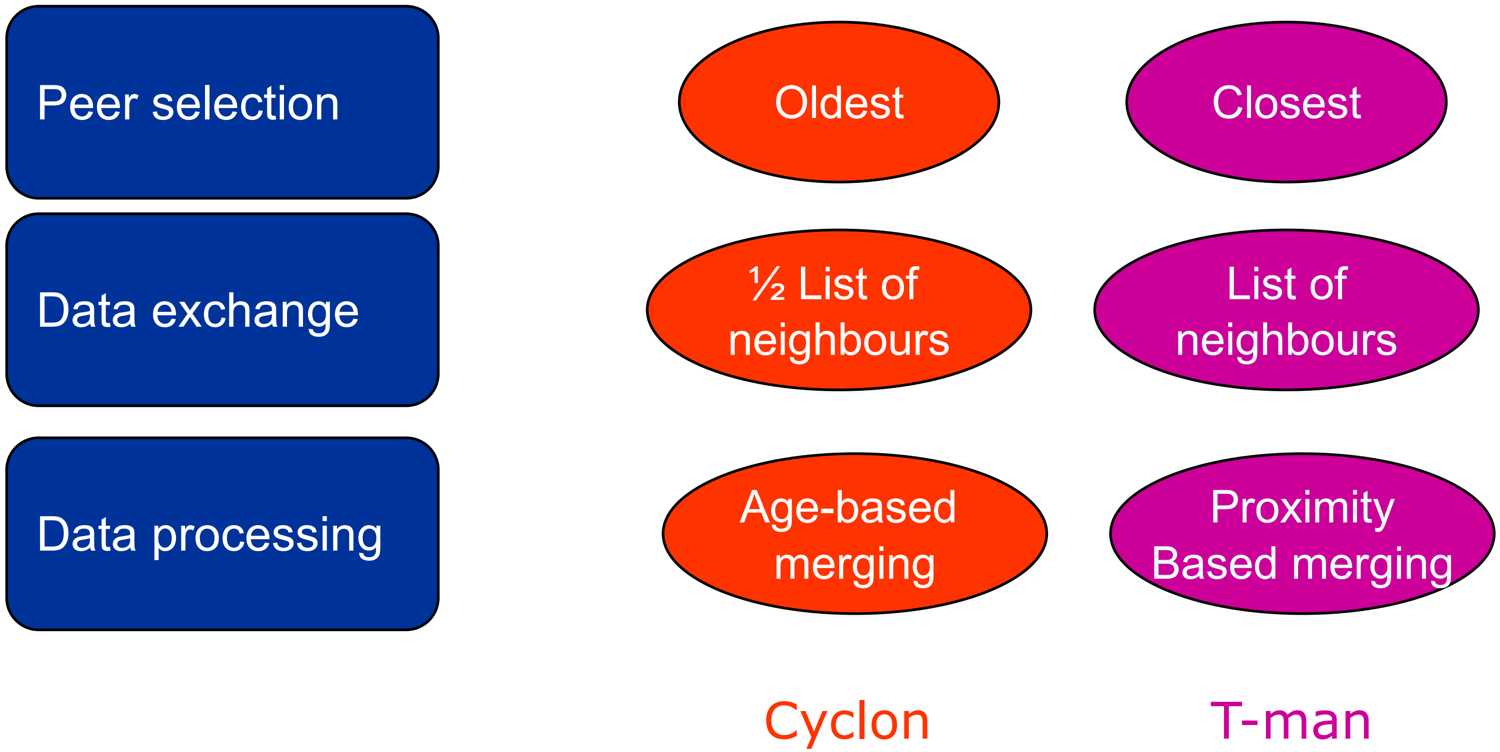
- Incorporate M into own state
- If (M not a response)
  - Select some data D
  - Send D to p

Data processing

Dissemination



Overlay maintenance



Decentralized computations

	Aggregation	System size estimation	Slicing
Peer selection	Random	Random	Random
Data exchange	value	value	Attribute value Random value
Data processing	Aggregation Average	Aggregation	Attribute/random matching

Epidemic-based dissemination

Goal:

Broadcast reliably to a large number of peers

System model:

- *n* processes
- Each process forwards the message once to *f* (fanout) neighbors, picked up uniformly at random.
- Alternatively f times to 1 neighbour.

Success metrics:

- Proportion of infected processes

  $Y_r = Z_r / n$

  Z_r is the number of infected processes prior to round r
- Probability of atomic "infection"

  $P(Z_r = n)$

Proportion of infected processes

Large system of size n

Probabibility that the epidemic catches (1-p_{ext})

Proportion of processes eventually contaminated

$\pi = 1 - e^{-\pi f}$ where f is the fanout

Independent of n, a fixed average of descendants will lead to the same proportion of infected processes

Probability of atomic infection

Erdos/Renyi examine final system state, the system is represented as a graph where each node is a process, there is an edge from n_1 to n_2 if n_1 is infected and chooses n_2 .

An epidemic starting at n_0 is successful if there is a path from n_0 to all members. If the fanout is $\log(n) + c$, the probability that a random graph is connected is

$$p(connect) = e^{-e^{-c}}$$

Other measures

Latency of infection

[Bollobas, Random Graphs, Cambridge University Press, 2001]

Logarithmic number of rounds

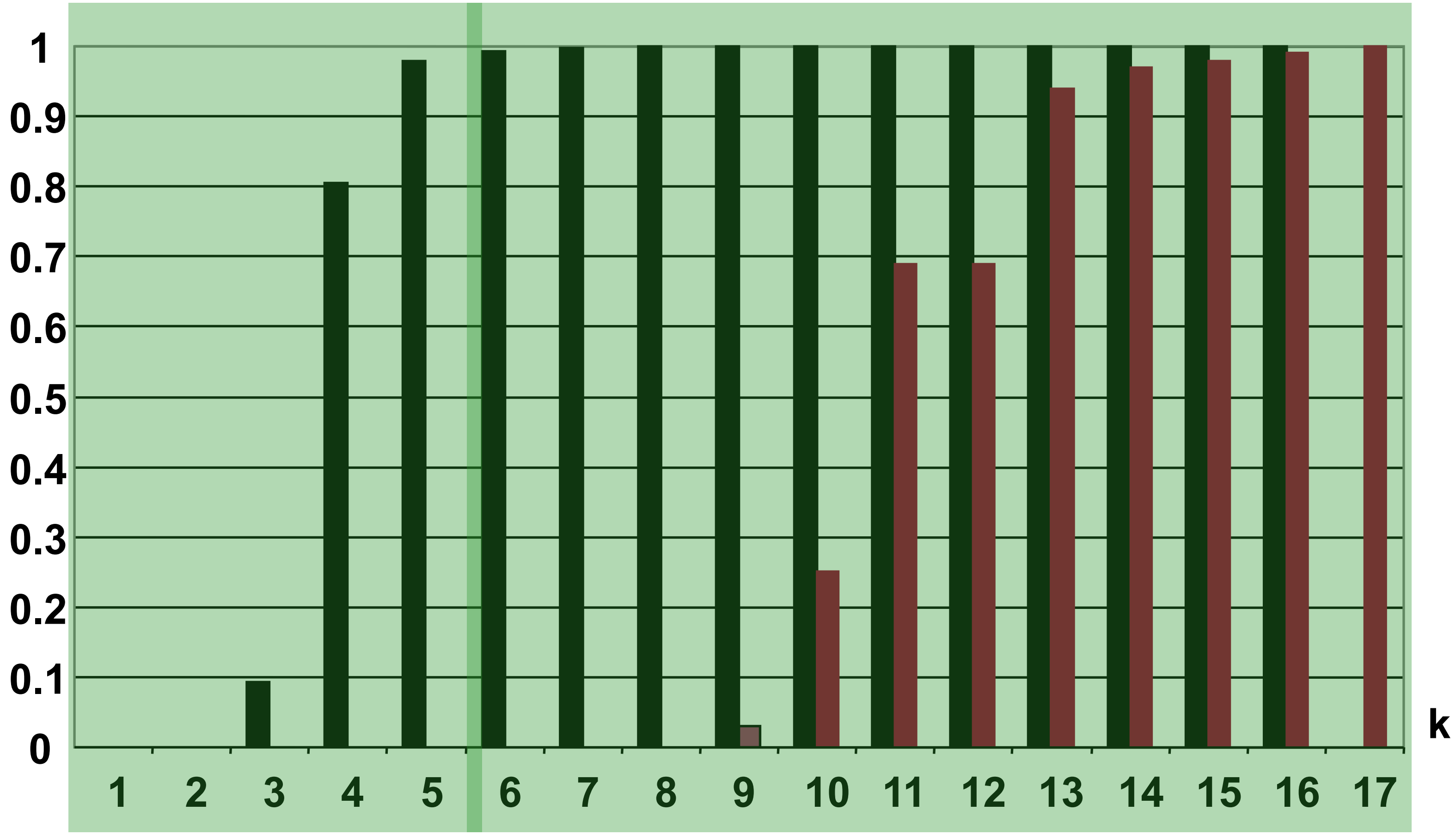
$$R = \frac{\log(n)}{\log(\log(n))} + O(1)$$

Resilience to failure

[KMG, IEEE Tpds 14(3), Probabilistic reliable dissemination in Large-scale systems, 2003]

$$k = (n / n')[\log(n') + c + O(1)]$$

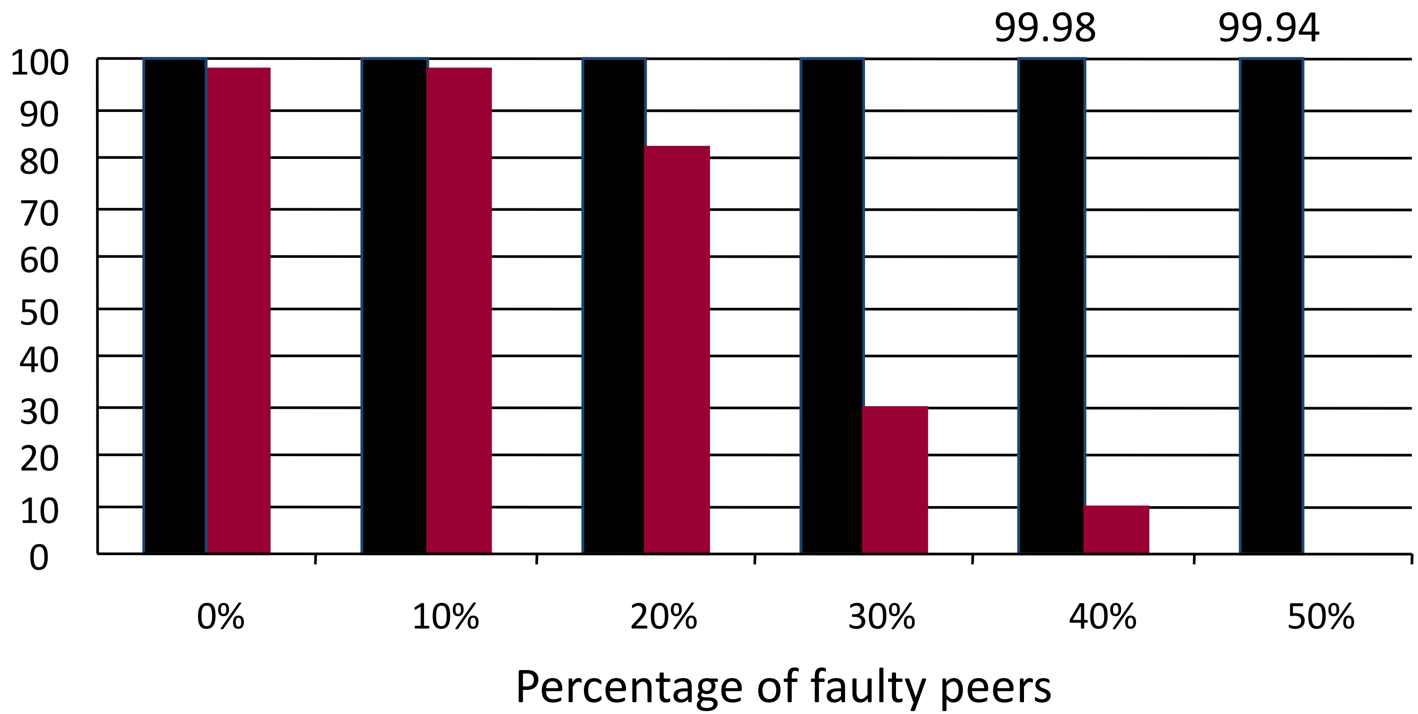
Performance (100,000 peers)



Proportion of “atomic”broadcast

Proportion of connected peers in non “atomic” broadcast

Failure resilience (100,000 peers)



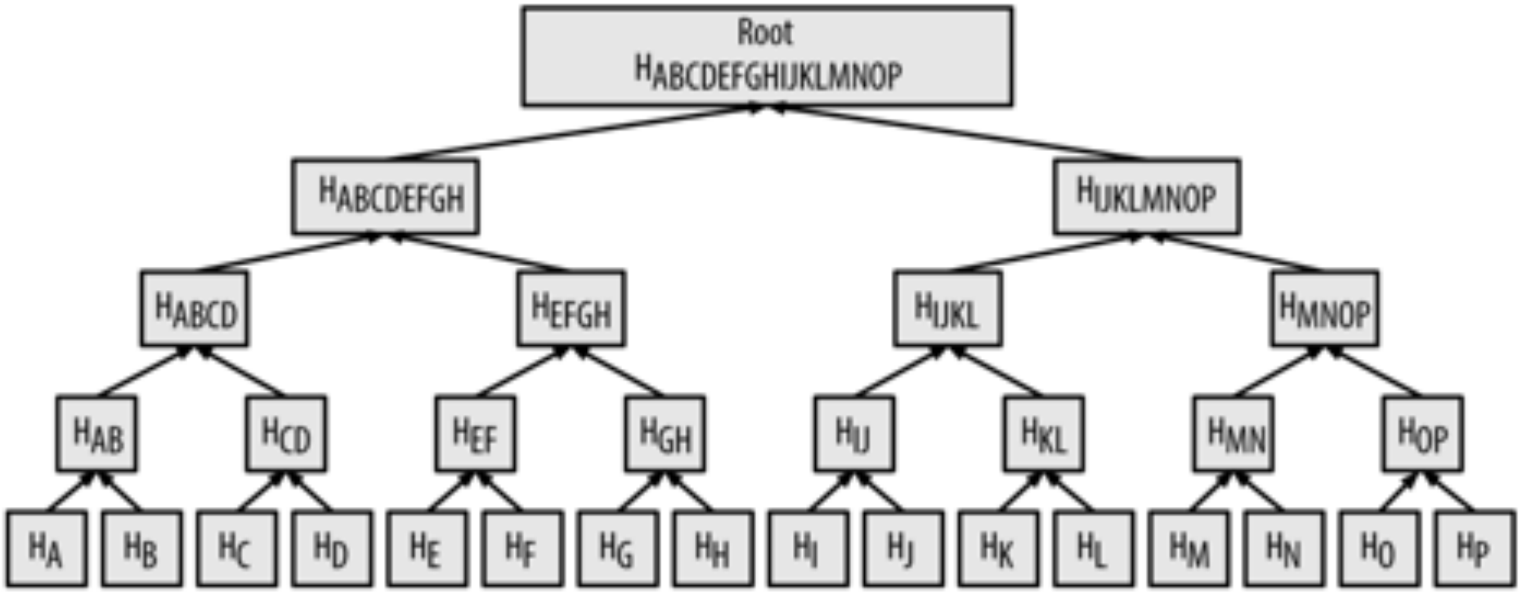
Proportion of "atomic" broadcast

Proportion of connected peers in non "atomic" broadcast

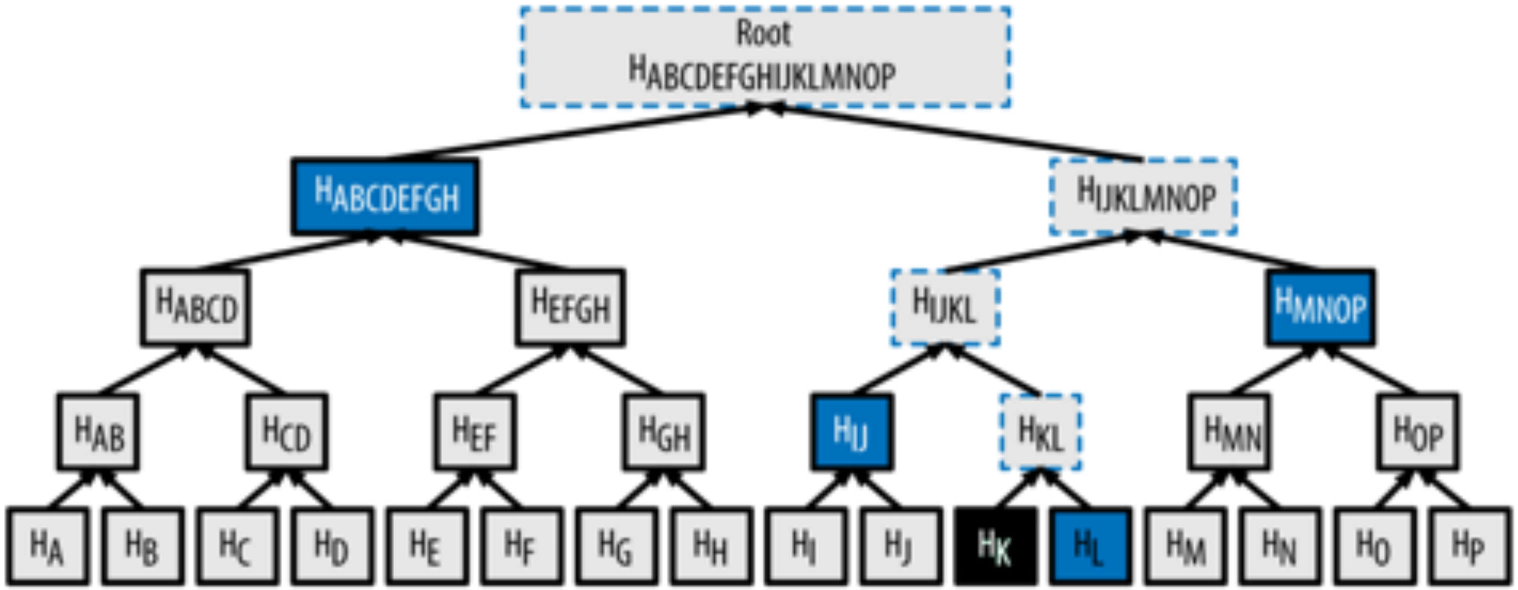
Gossip in Key Value Stores

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Merkle Tree of Transactions

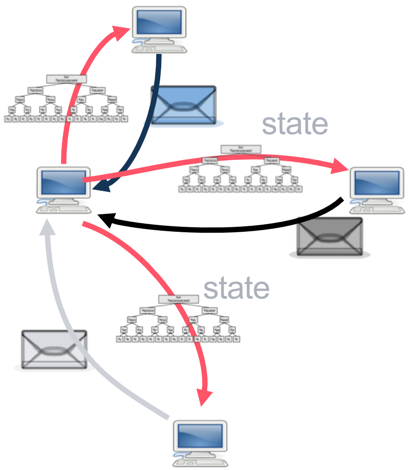


Merkle Tree of Transactions



Anti Entropy with Merkle Trees

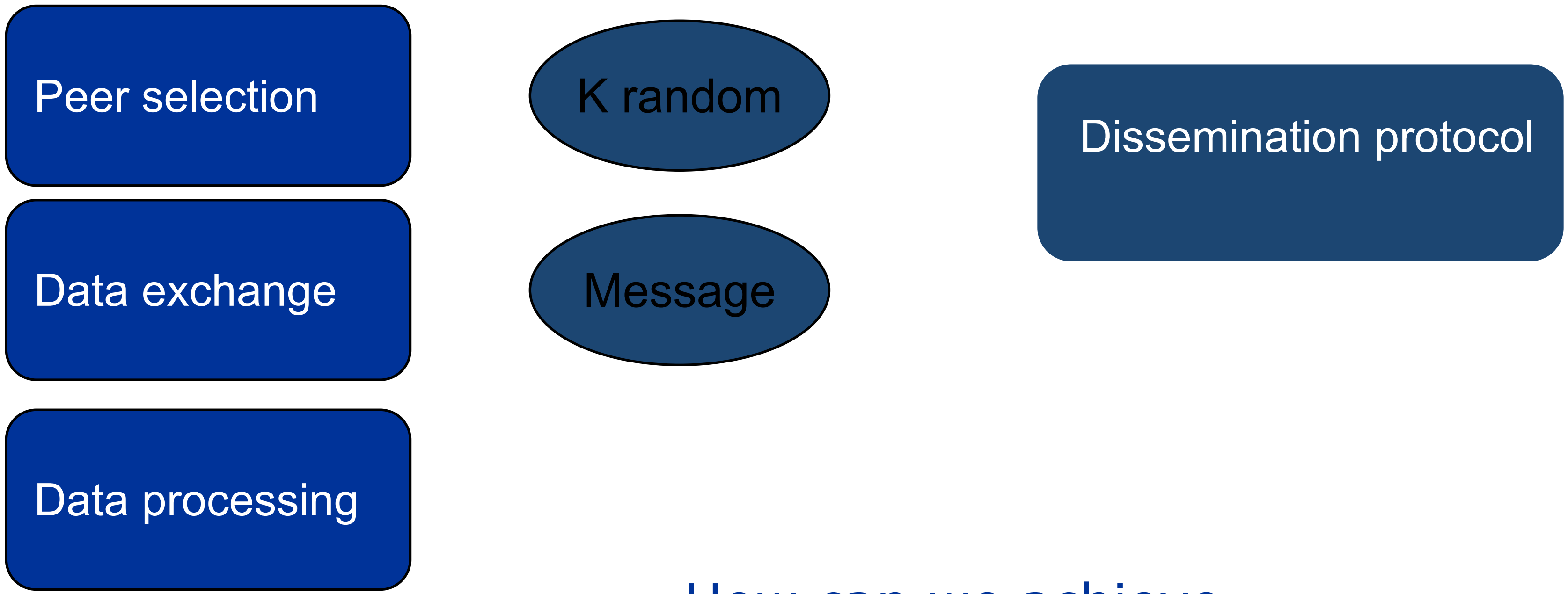
Anti-
Entropy



Gossip in Key Value Stores

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

Dissemination relies on Random Sampling



How can we achieve Random sampling?

Today

- Gossip Basics
- Overlay Maintenance
  - Random peer sampling
  - Clustering

Gossip Overlays: Random Peer Sampling

Goal:

- Provide each peer with a continuously changing random sample of the network.

Effect:

- Overlay consists of a continuously changing random-like graph

The Peer Sampling Service

Creates unstructured overlay network topologies

Interface

- *Init()*: service initialization
- *GetPeer():* returns a peer address, ideally drawn uniformly at random

The Peer Sampling service

System Model

- System of *n* peers
- Peers join and leave (and fail) the system dynamically and are identified uniquely (IP @)
- Epidemic interaction model:
  Peers exchange some membership information periodically to update their own

Data Structures

- Each peer maintains a view (membership table) of *c* entries
  - Network @ (IP@)
  - Timestamp (freshness of the descriptor)

Protocol

Active Cycle
Periodically

Peer selection

P <- selectPeer()

myDescriptor <- (my@, now)
buffer <- merge (view,
{myDescriptor})

send buffer to p

Data exchange
(View Propagation)

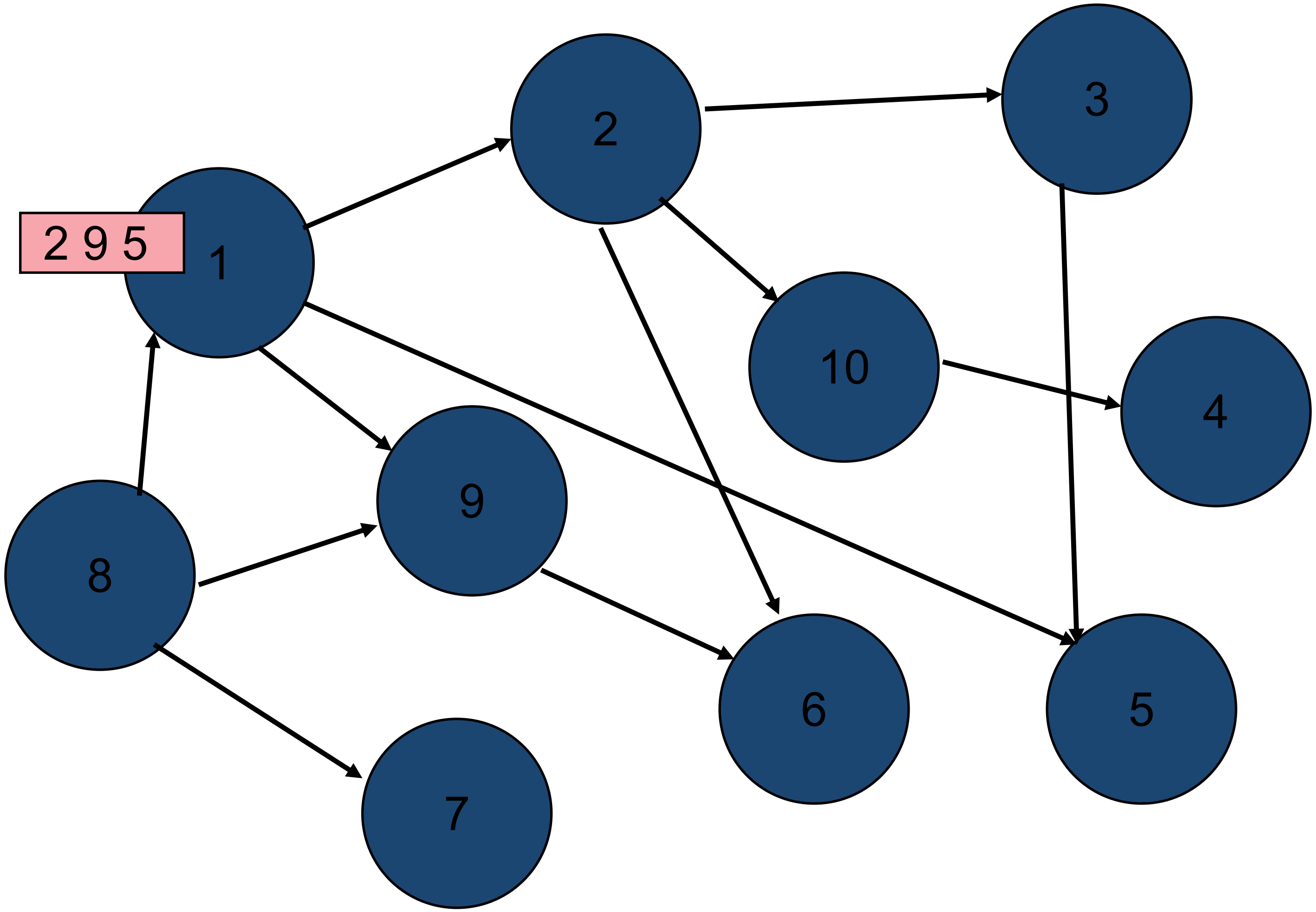
Passive Cycle
When message received from p

buffer <- merge(view_p, view)
View <-selectView(buffer)

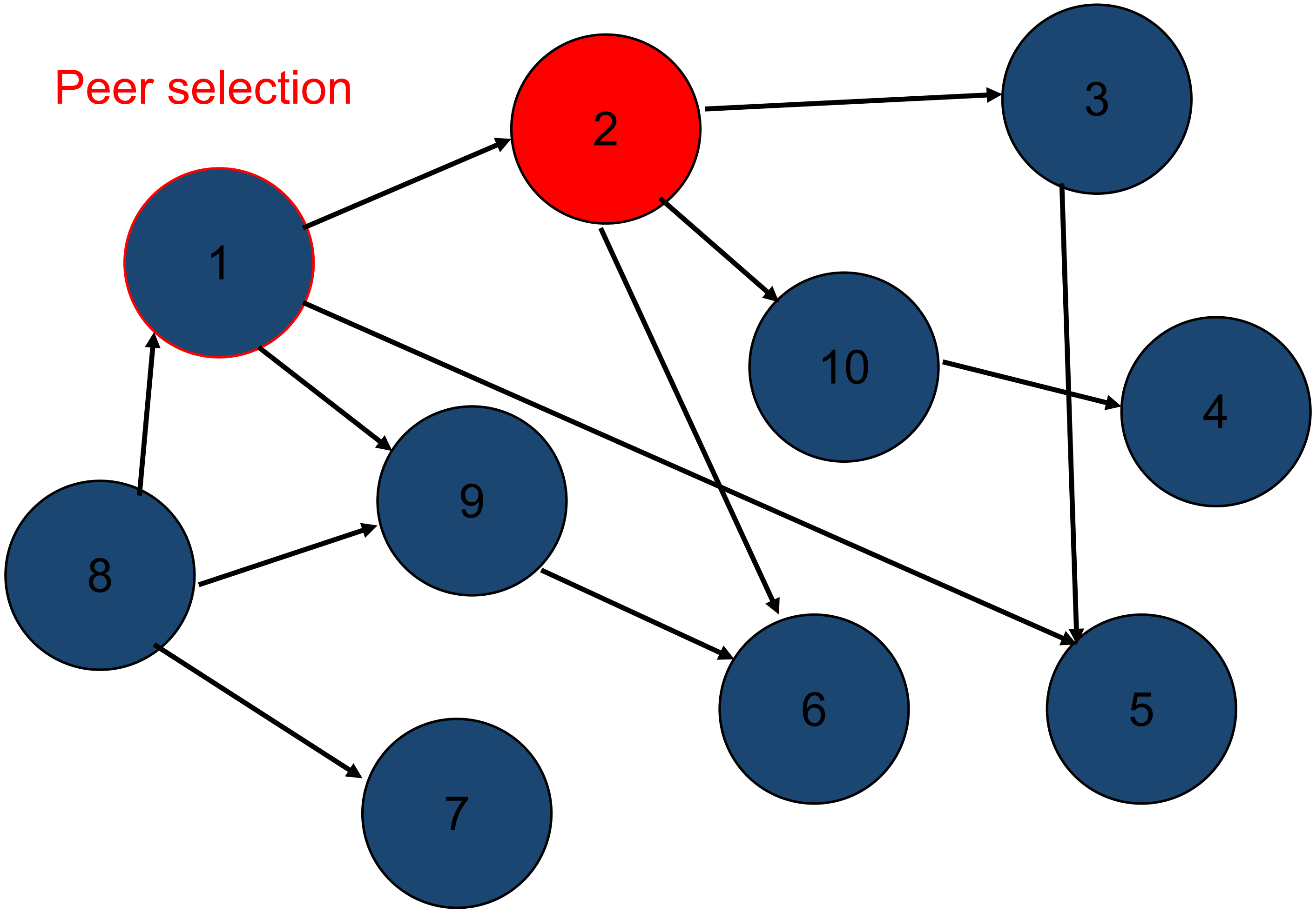
if pull **and** not receiving response **then**
myDescriptor <-(my@, now)
buffer <-merge(view,{myDescriptor})
send buffer to p

Data processing
(View Selection)

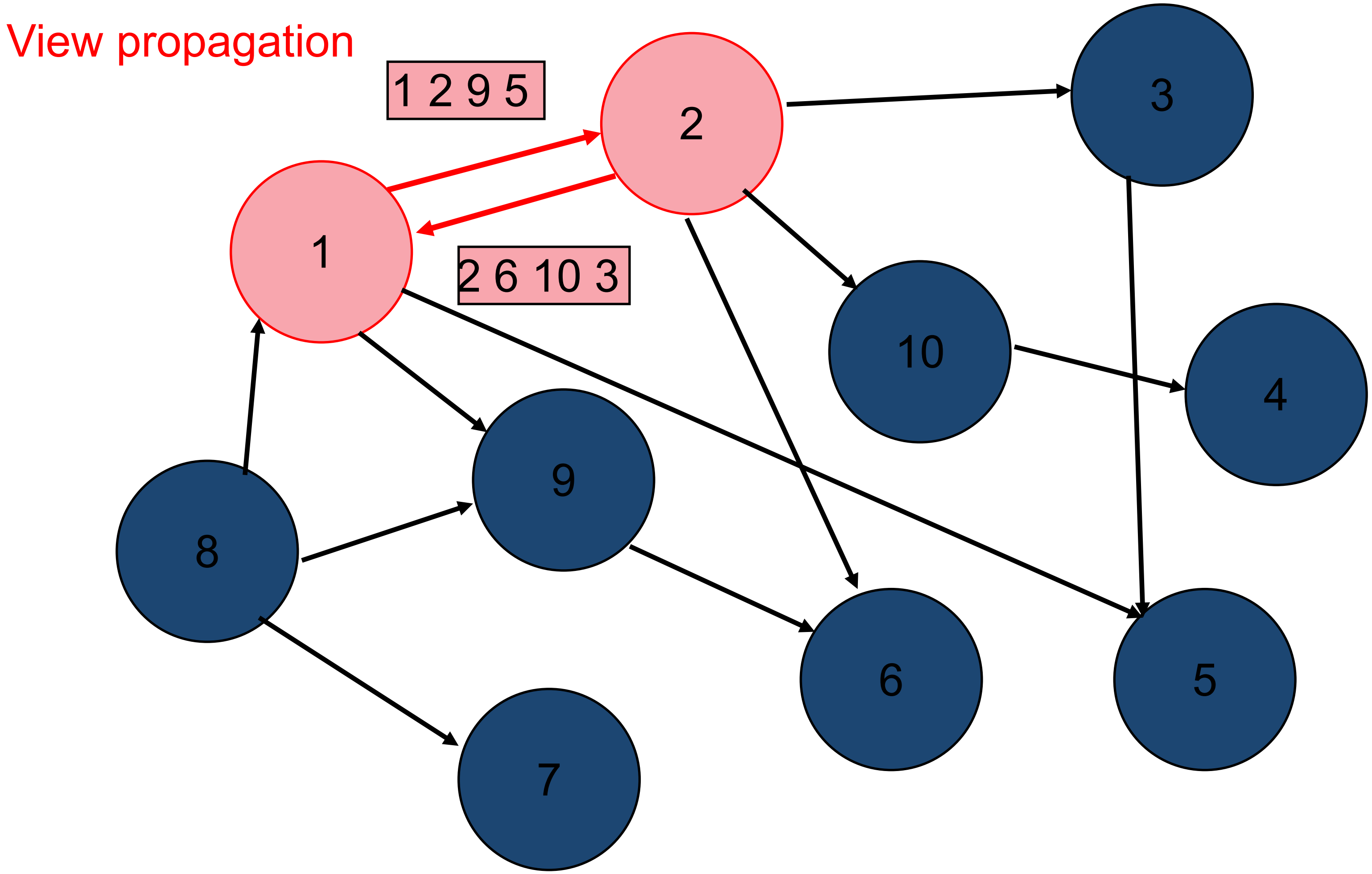
Generic protocol



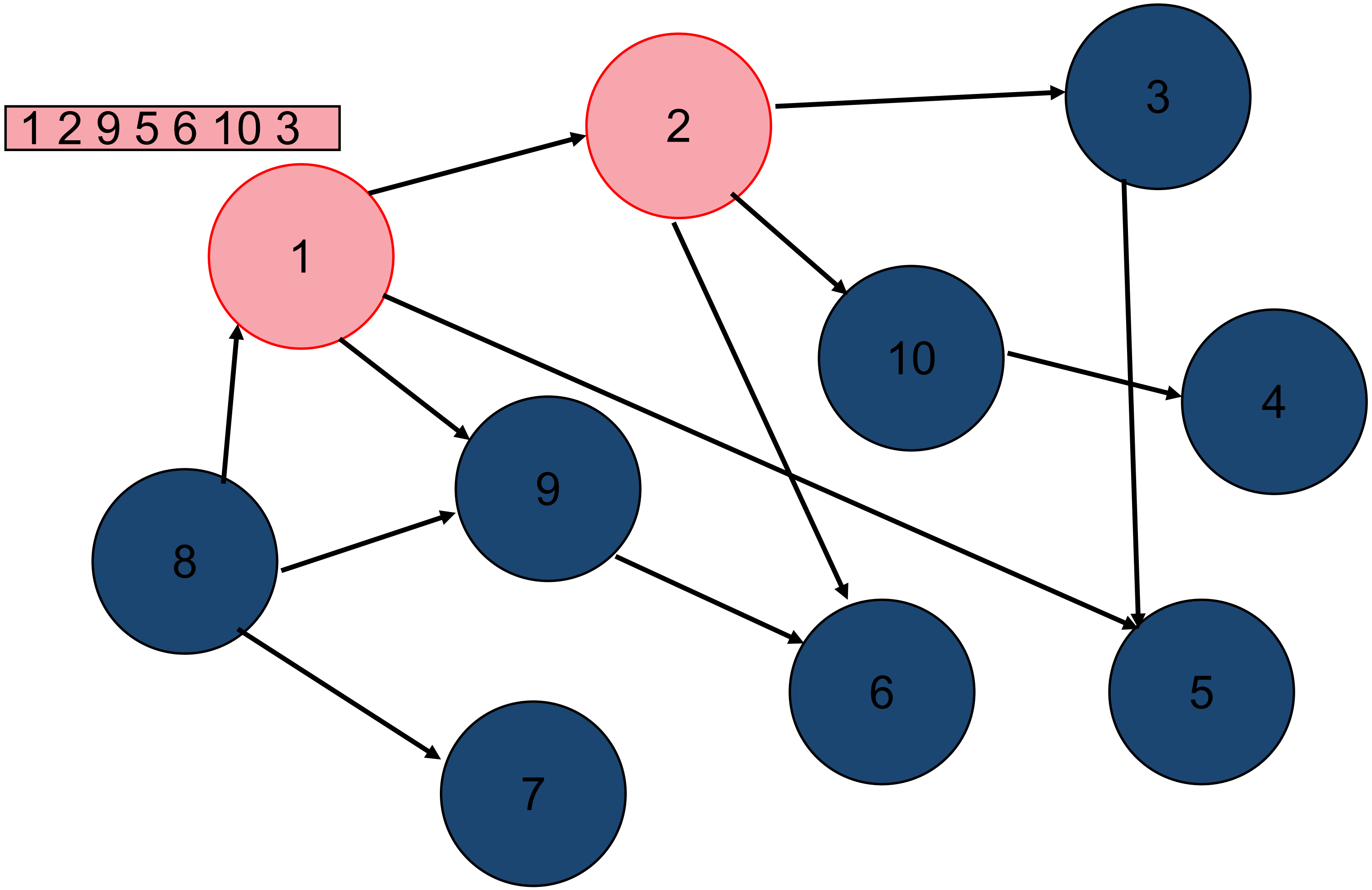
Generic protocol



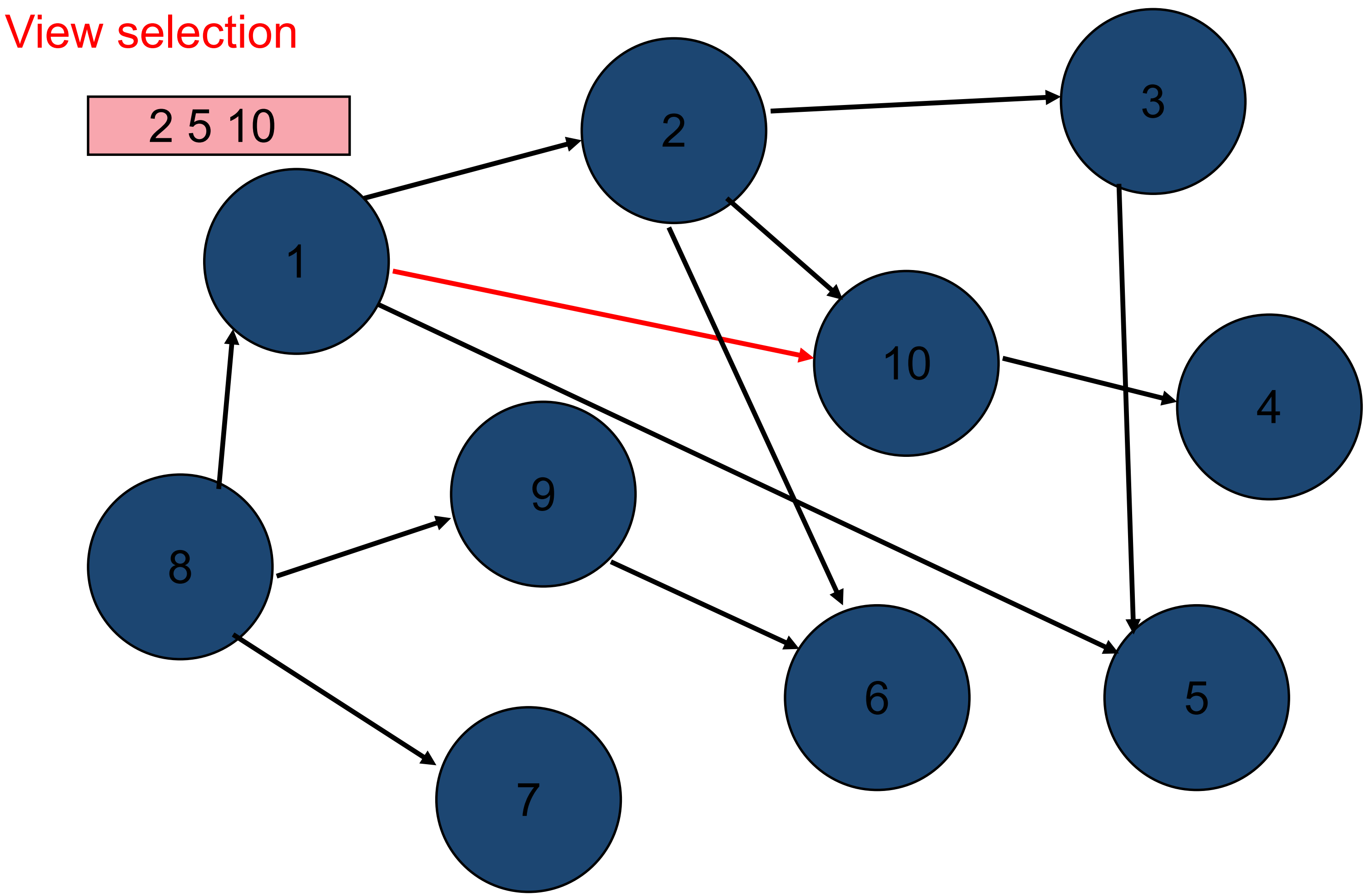
Generic protocol



Generic protocol



Generic protocol



Protocol

Active Cycle
Periodically

Peer selection

P <- selectPeer()

myDescriptor <- (my@, now)
    buffer <- merge (view, {myDescriptor})

send buffer to p

Data exchange
(View Propagation)

Passive Cycle
When message received from p

buffer <- merge(view_p, view)
View <-selectView(buffer)

if pull **and** not receiving response **then**
    myDescriptor <-(my@, now)
    buffer <-merge(view,{myDescriptor})
    send buffer to p

Data processing
(View Selection)

Design space

- **Peer selection**

Periodically each peer initiates communication with another peer

- **Data exchange (View propagation)**

How peers exchange their membership information?
What do they exchange?

- **Data processing (View selection)**: Select (c, buffer)

c: size of the resulting view
Buffer: information exchanged

Design space: peer selection

Three Strategies

Rand: pick a peer uniformly at random

Head: pick the "youngest" peer

Tail: pick the "oldest" peer

Note that *head* leads to correlated views.

Design space: data exchange

Buffer (h)

initialized with the descriptor of the gossiper

contains *c/2* elements

ignore *h* "oldest"

Two Strategies

Push: buffer sent

Push/Pull: buffers sent both ways

(Pull: left out, the gossiper cannot inject information about itself, harms connectivity)

Design space: Data processing

Select(*c*,*h*,*s*,*buffer*)

1. Buffer appended to view
2. Keep the freshest entry for each node
3. *h* oldest items removed
4. *s* first items removed (the one sent over)
5. Random nodes removed

Merge strategies

Blind (*h=0,s=0)*: select a random subset

Healer (*h=c/2*): select the "freshest" entries

Shuffler (*h=0, s=c/2*): minimize loss

c: size of the resulting view

H: self-healing parameter

S: shuffle

Buffer: information exchanged

Design space summary

Peer selection

rand	Select a peer at random from the view
tail	Select the node with the highest hop count

Head leads to correlated views

View propagation

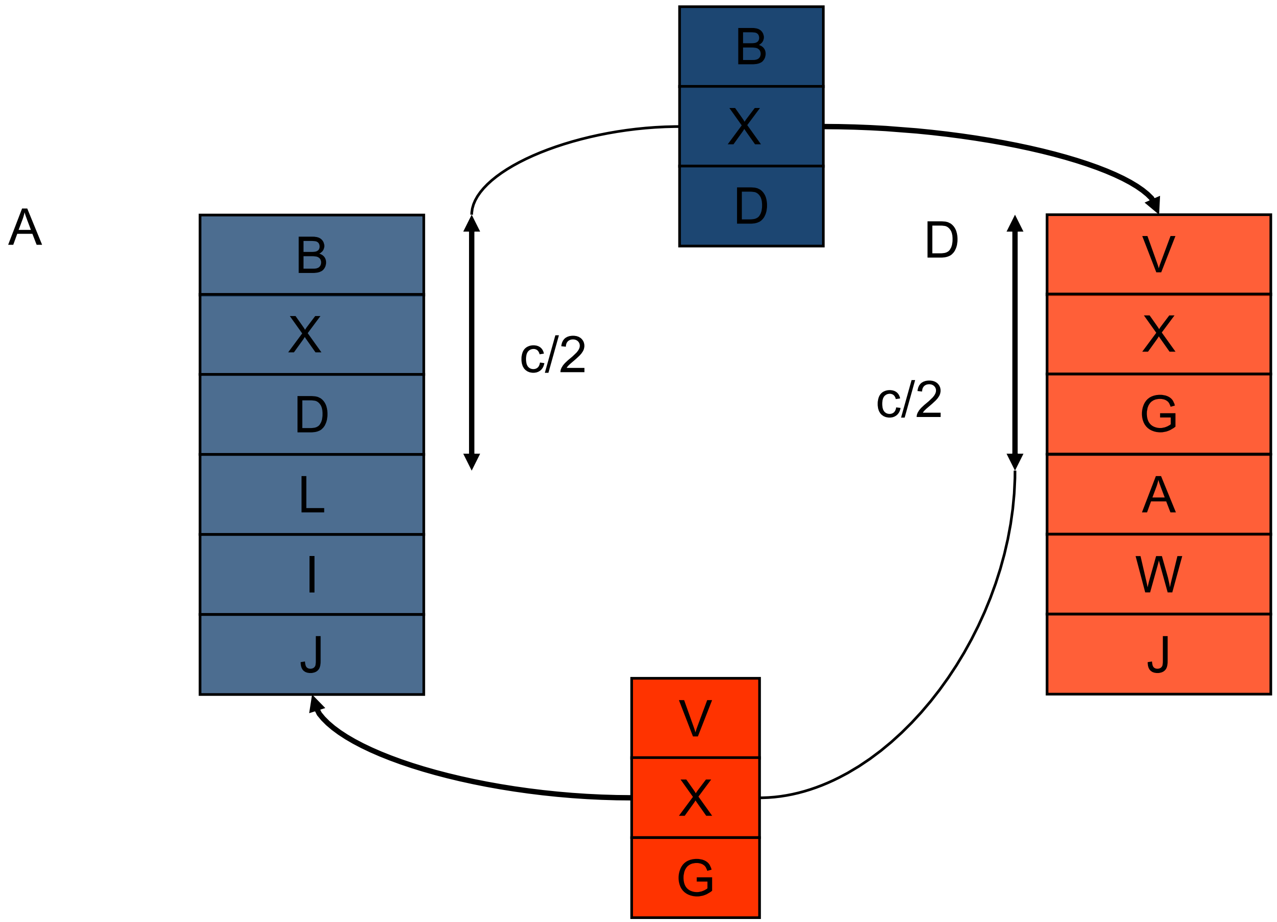
push	The node sends its buffer to the selected peer
pushpull	The node and the selected peer exchange information

Pull: risk of partition (a node has no possibility to inject information about itself)

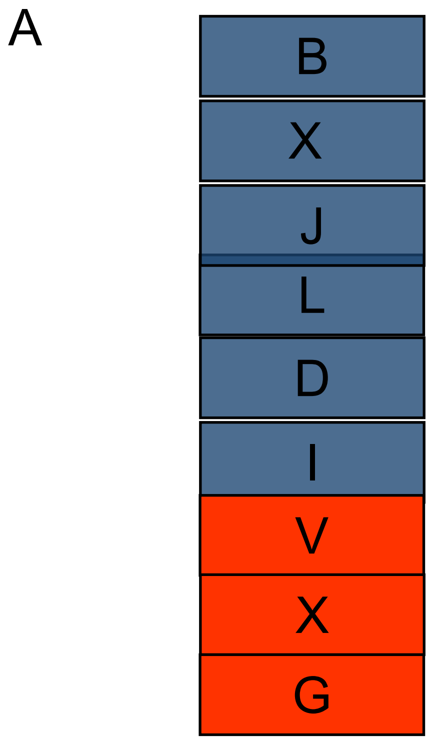
View selection

blind	H = 0, S = 0	Blind selection of a random subset
healer	H = c/2	Select the freshest entries
shuffler	H = 0, S = c/2	Minimize loss of information

Example



Example



1. Buffer appended to view
2. Keep the freshest entry for each node
3. h (=1) oldest items removed
4. s (=1) first items removed (the one sent over)
5. Random nodes removed

Some systems

Lpbcast [Eugster & al, DSN 2001,ACM TOCS 2003]
Peer selection: random
View propagation: push
View selection: random

Newscast [Jelasity & van Steen, 2002]
Peer selection: head
View propagation: pushpull
View selection: head

Cyclon [Voulgaris & al JNSM 2005]
Peer selection: random
View propagation: pushpull
View selection: Shuffle

Experimental Study

- Relationship « who knows who »
  - Highly dynamic
  - Capture quickly changes in the overlay networks
- Protocol Variants
  - Healer (h=c/2, s=0)
  - Shuffler (h=0, s=c/2)
- Scenarios
  - lattice
  - random
  - growing networks
- Metrics
  - Degree distribution
  - Average path length
  - Clustering coefficient

Degree distribution

Out degree = c (30) in 10.000 node system

Distribution of in-degree

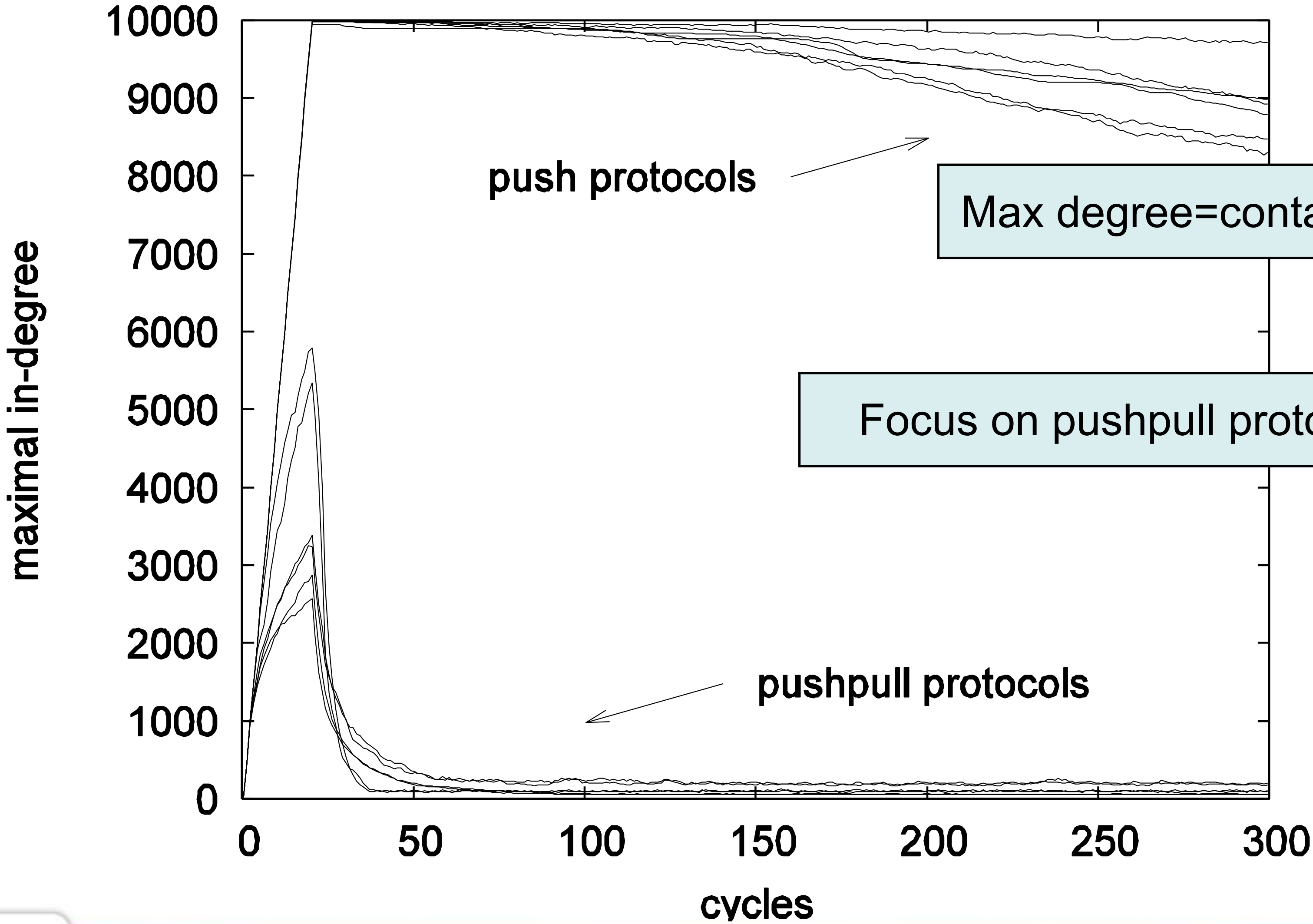
Detect hotspot and bottleneck

Load balancing properties

Convergence

Self-organization ability irrespective of the initial topology

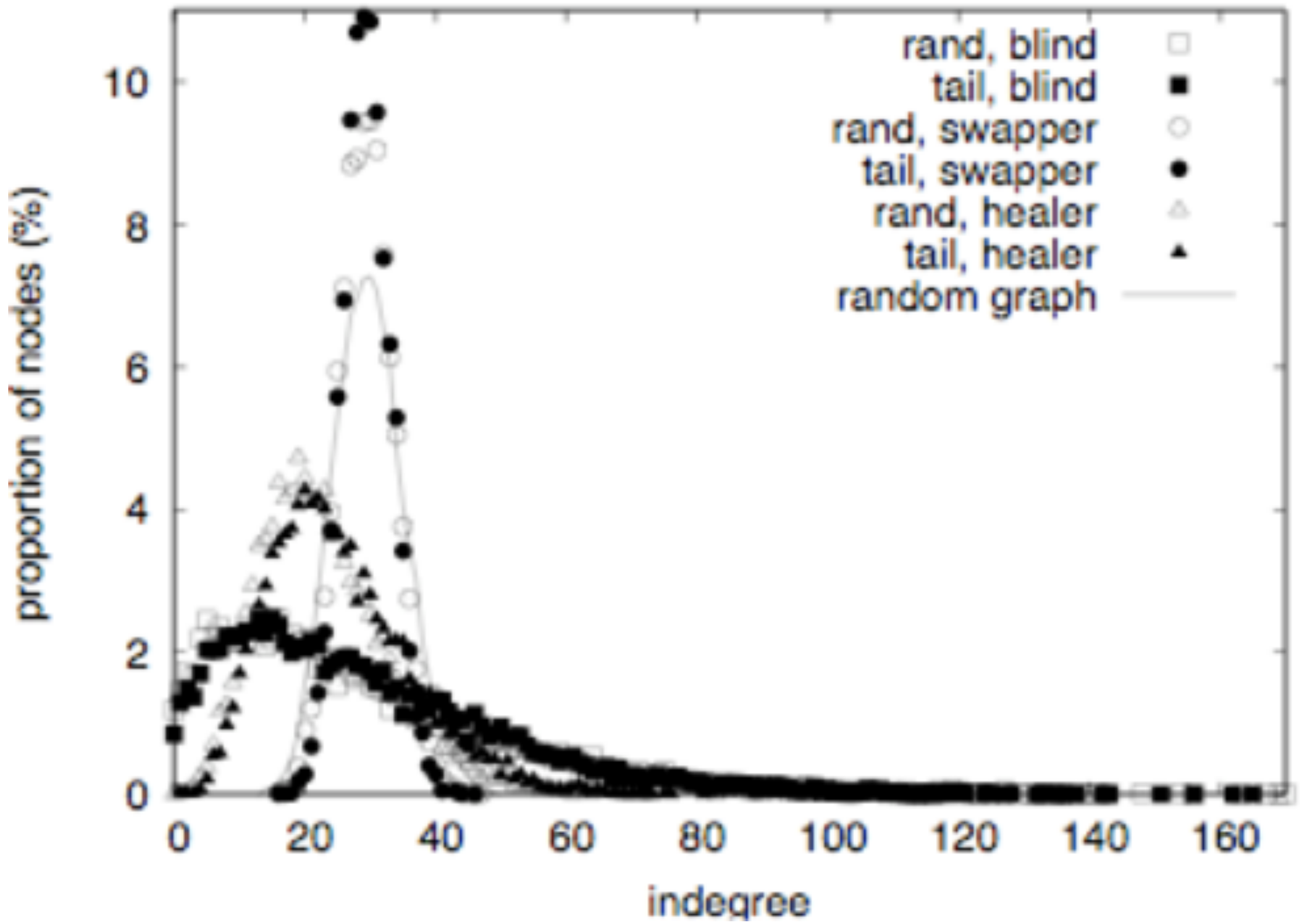
Degree distribution growing scenario



Max degree=contact node

Focus on pushpull protocols

Degree distribution



Degree distribution

Convergence

- Even in growing scenario
- Shuffler and healer result in lower standard deviation for opposite reasons

Shuffler

- Controlled degree distribution
- New links to a node are created only when the node itself injects its own fresh node descriptor during communication.

Healer

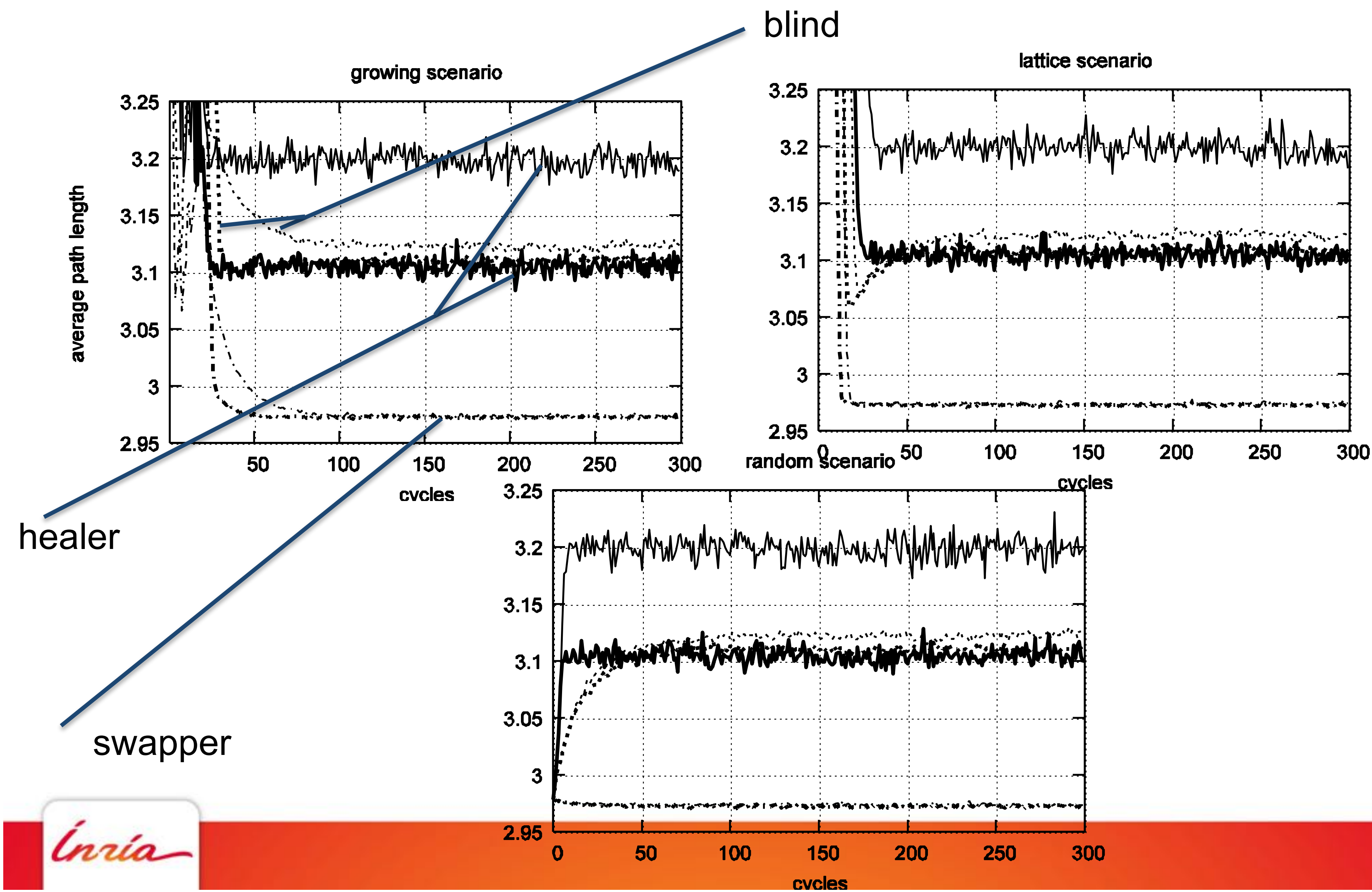
- Short life time of links
- When a node injects a new descriptor about itself, this descriptor is copied to other nodes for a few cycles.
- Later all copies are removed because they are pushed out by new links injected in the meantime

Average path length

Shortest path length between a and b

- minimal number of edges required to traverse in the graph to reach b from a
- Defines a lower bound on the time and costs of reaching a peer.
- Short average path length essential for scalability

Average path length



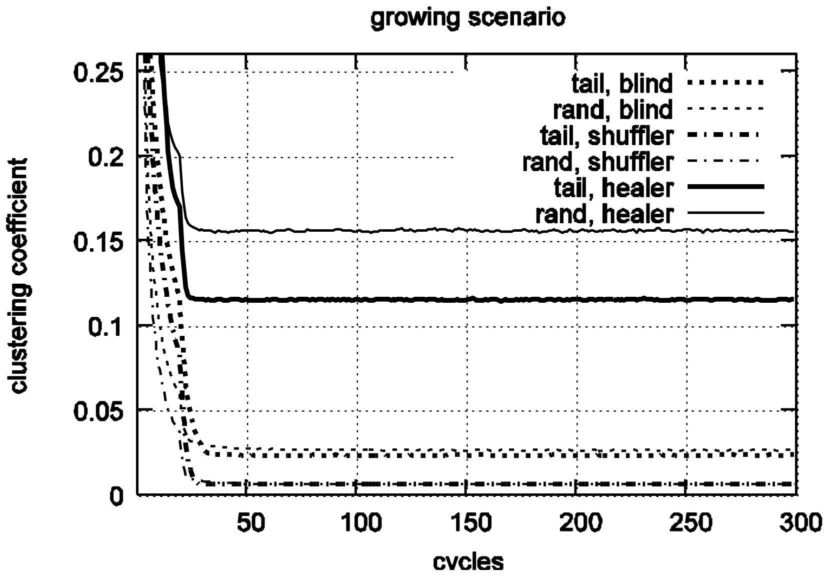
Clustering coefficent

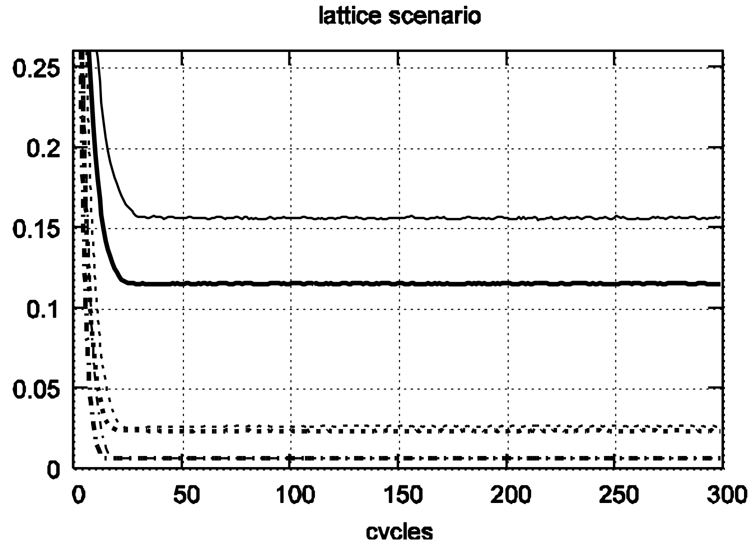
Indicates to what extent neighbours of neighbours are neighbours

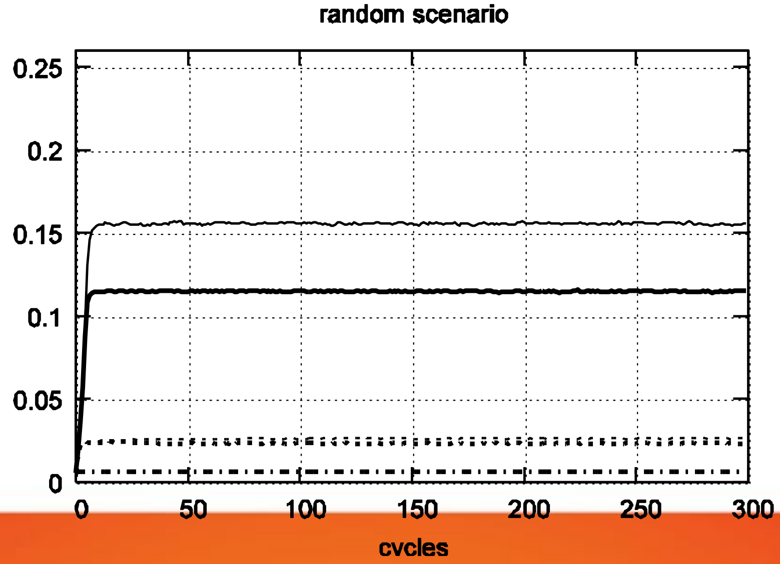
(1 for complete graph)

Important factor for information dissemination and partitioning risks

Clustering coefficient





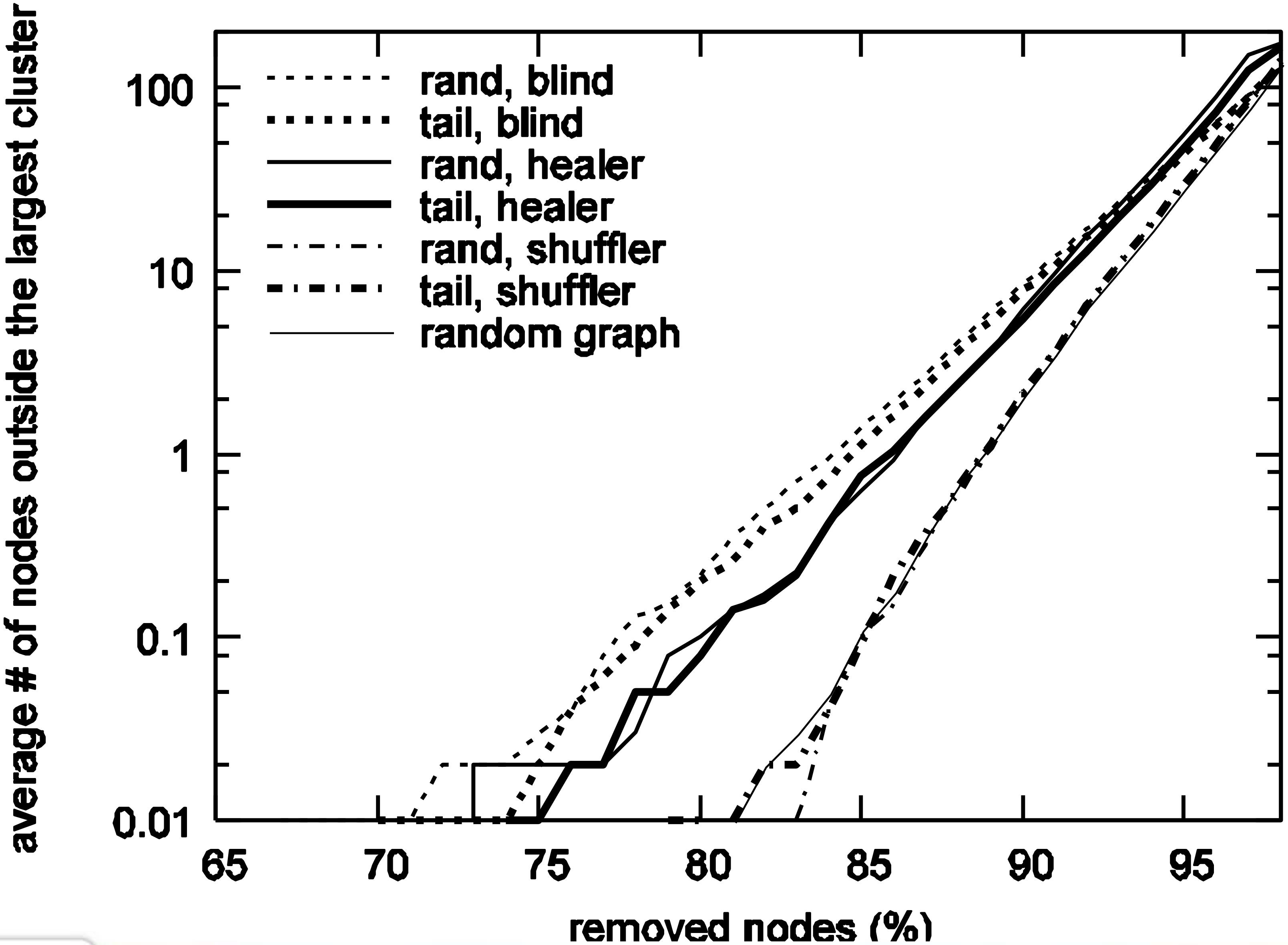


Clustering coefficient

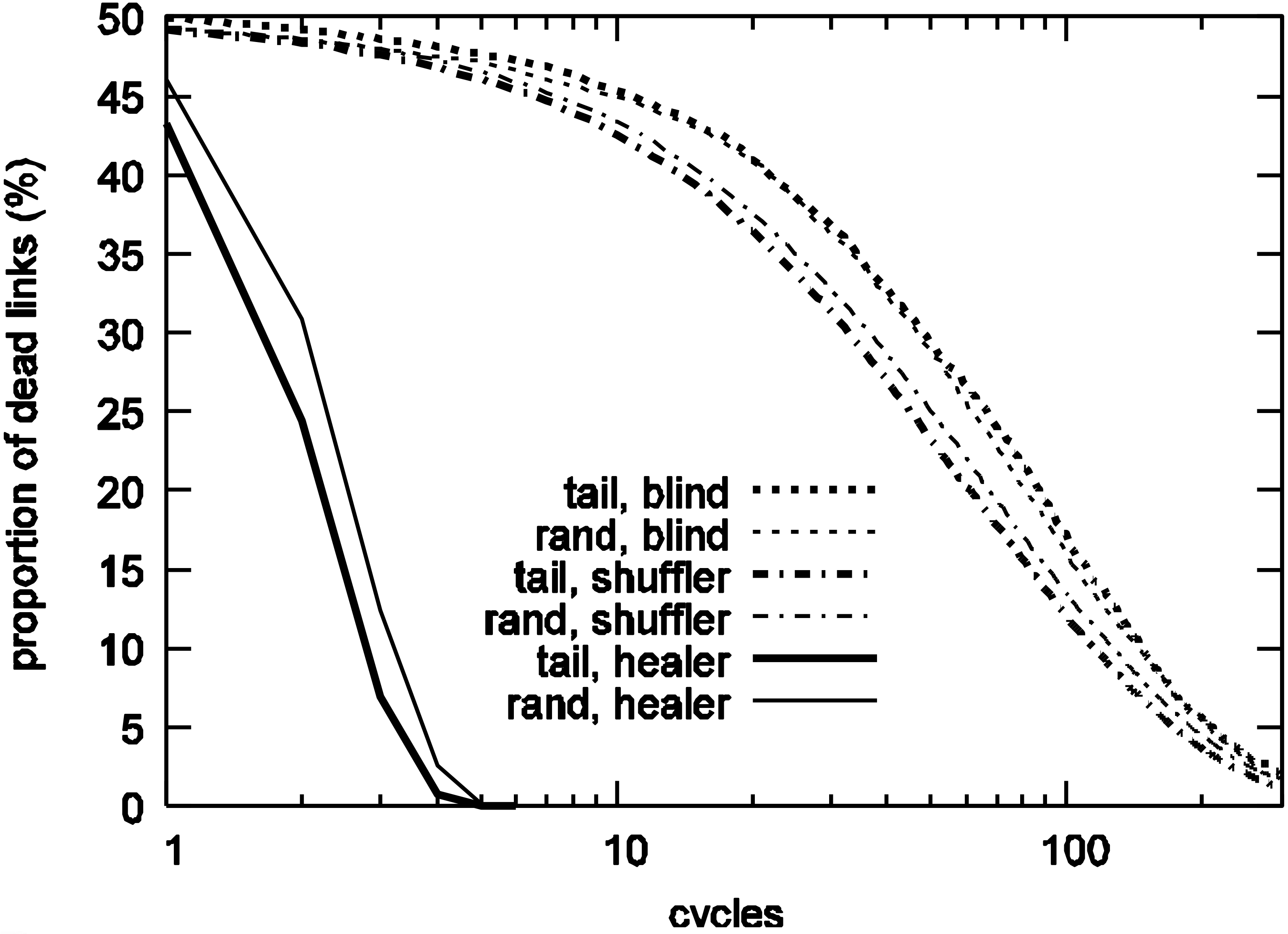
Results

- clustering coefficient also converges
- controlled mainly by H.
  - Large value of H result in significant clustering, where the deviation from the random graph is large.
    - large part of the views of any two communicating nodes overlap right after communication (freshest entries).
  - Large values of S, clustering is close to random

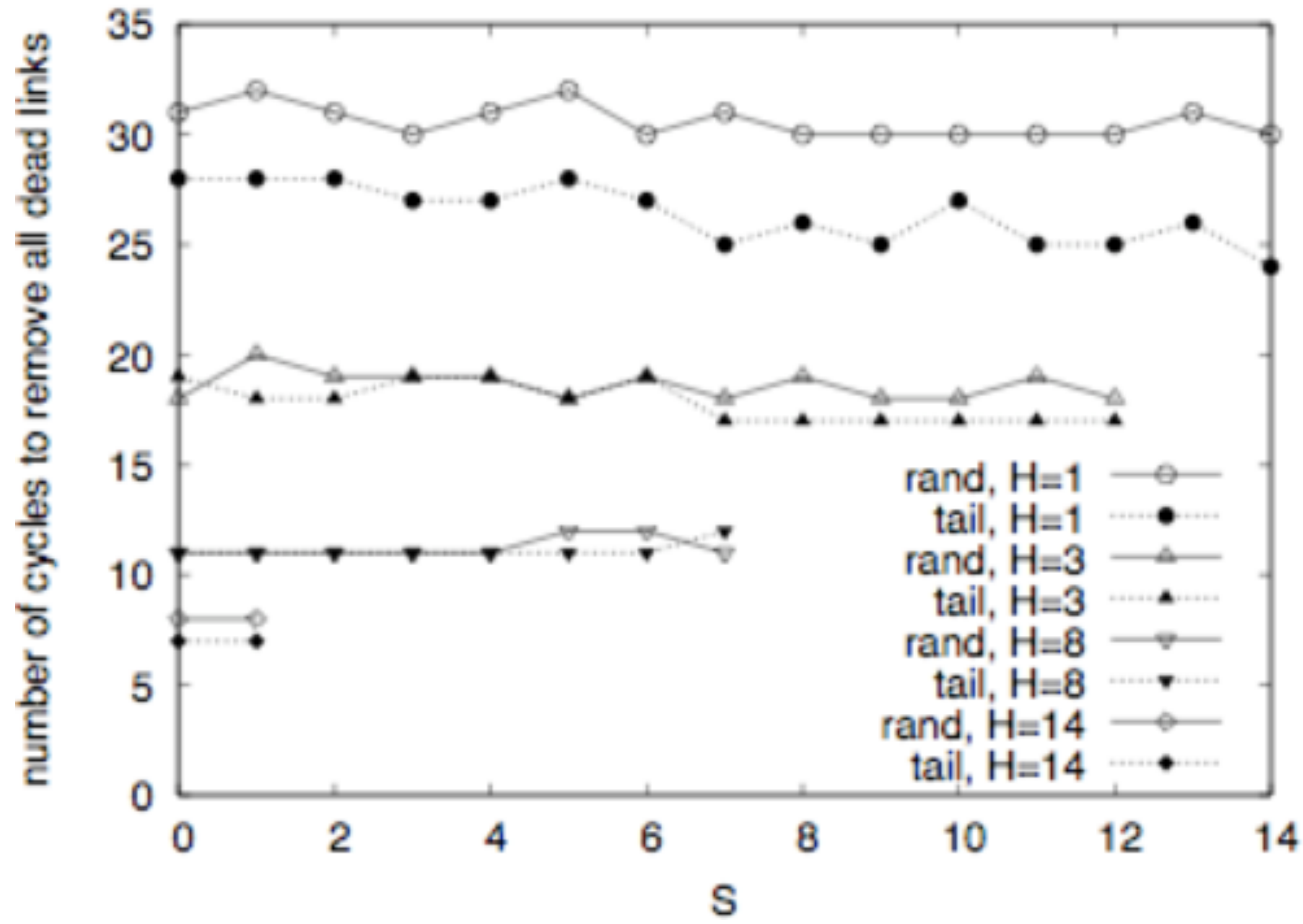
Catastrophic failures



Self-healing with 50% failures



Self-healing with 50% failures



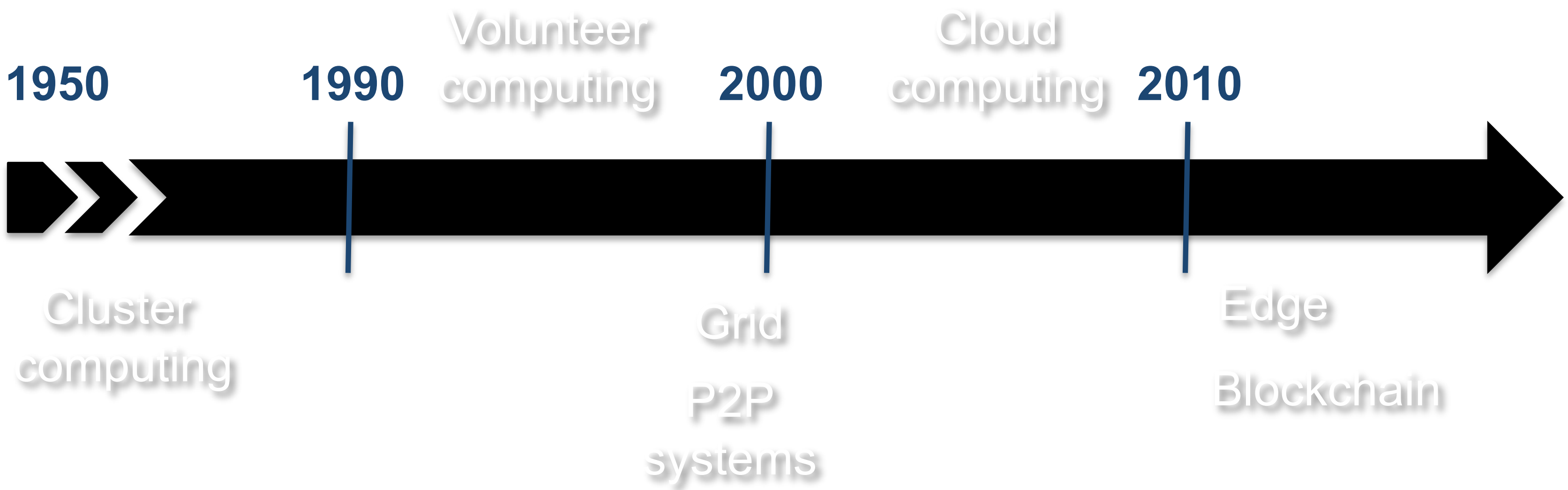
Dynamo's Ancestors

DISTRIBUTED HASH TABLES

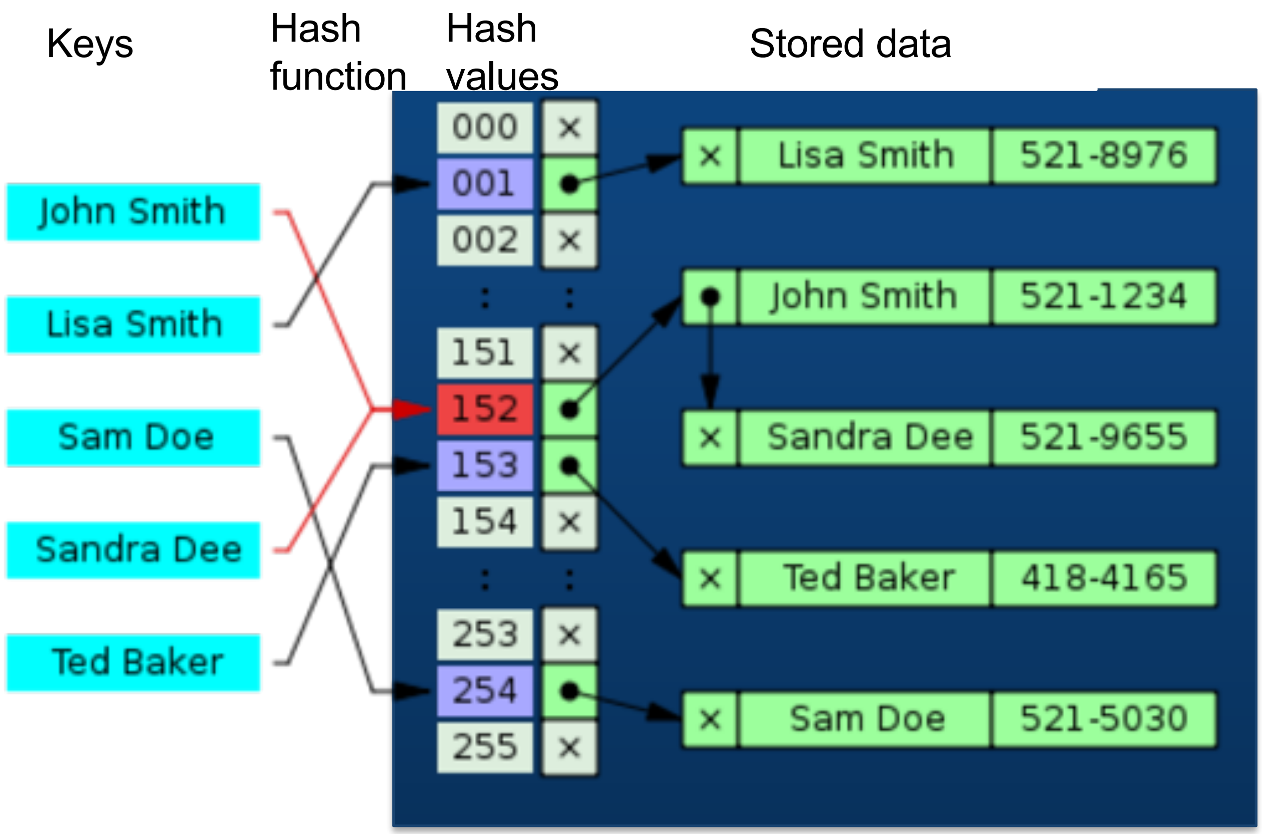
DHT technology in Key Value Stores

Problem	Technique	Advantage
Partitioning	Consistent Hashing	Incremental Scalability
High Availability for writes	Vector clocks with reconciliation during reads	Version size is decoupled from update rates.
Handling temporary failures	Sloppy Quorum and hinted handoff	Provides high availability and durability guarantee when some of the replicas are not available.
Recovering from permanent failures	Anti-entropy using Merkle trees	Synchronizes divergent replicas in the background.
Membership and failure detection	Gossip-based membership protocol and failure detection.	Preserves symmetry and avoids having a centralized registry for storing membership and node liveness information.

A (rough) timeline

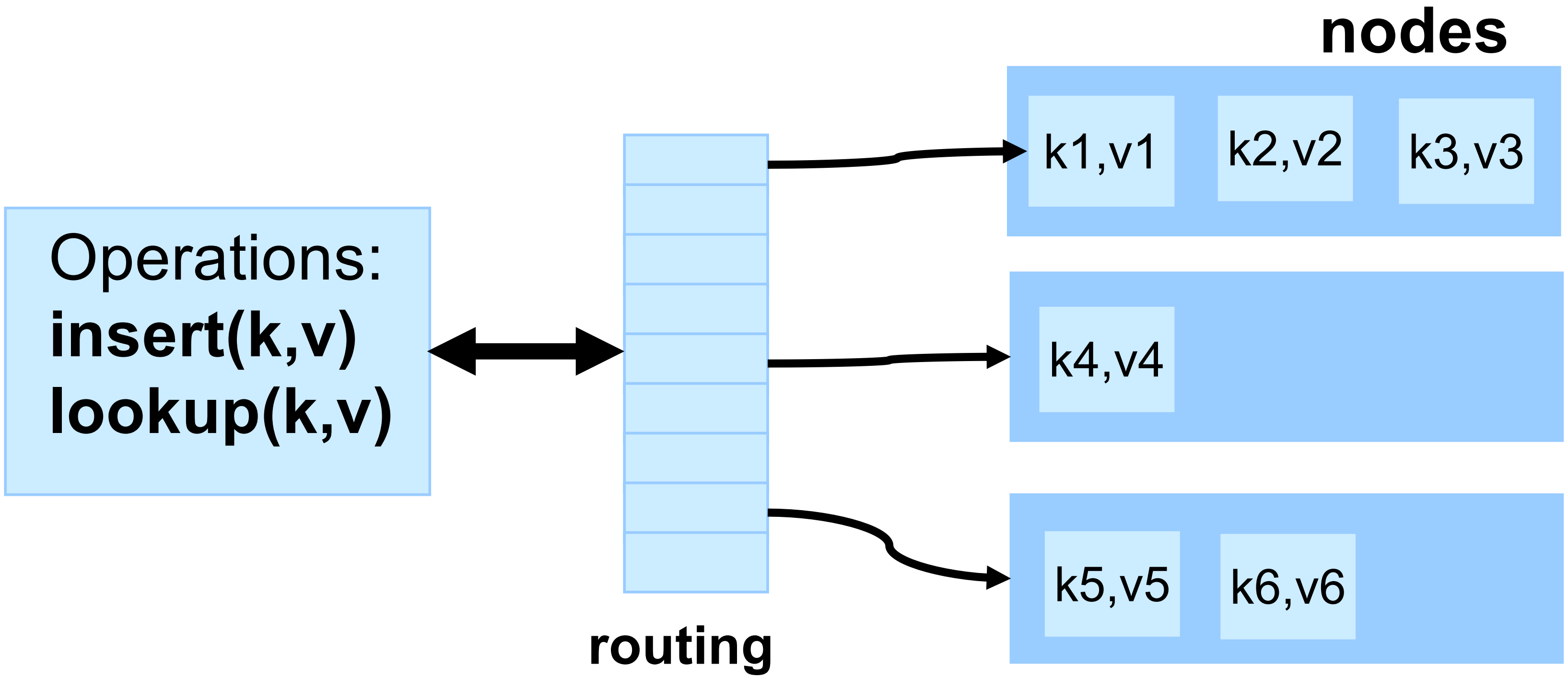


Hash Table



Efficient information lookup

Distributed Hash Table (DHT)

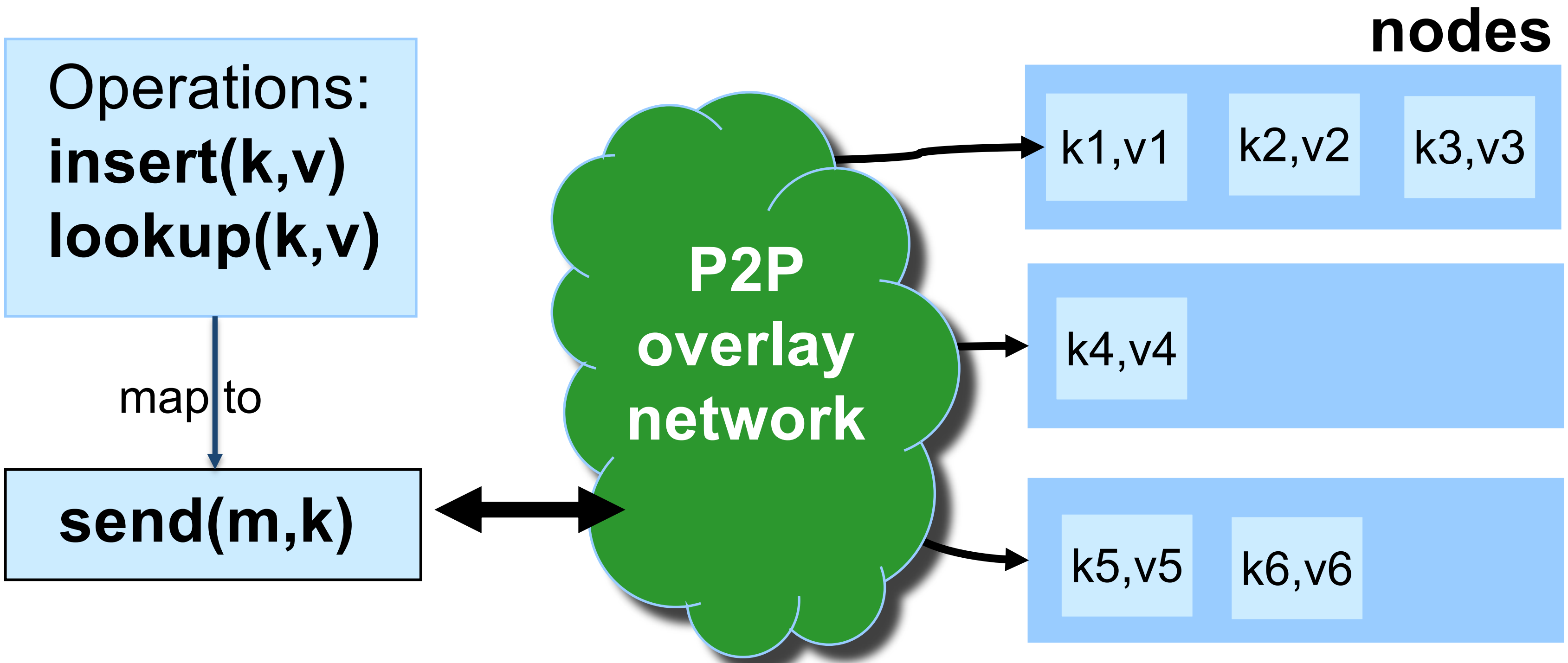


Store <key,value> pairs

Efficient access to a value given a key

Must route hash keys to nodes.

Distributed Hash Table

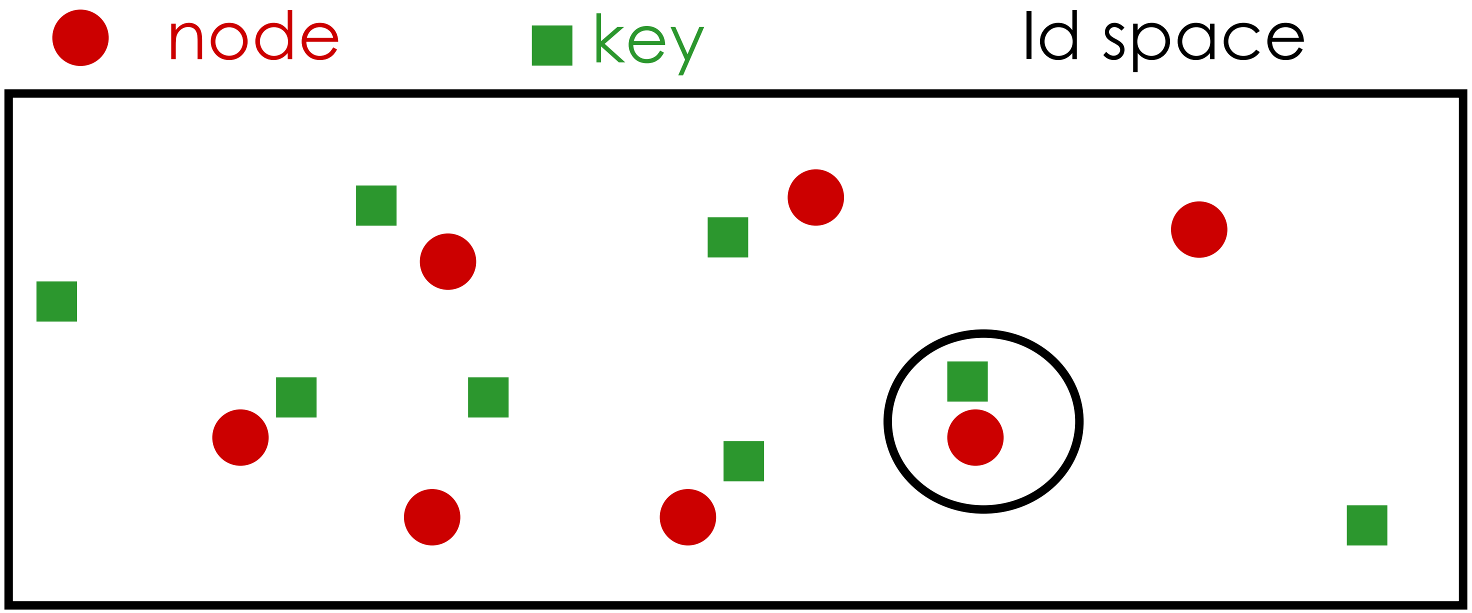


Insert and Lookup send messages keys

P2P Overlay defines mapping between keys and physical nodes

Decentralized routing implements this mapping

Pastry (MSR/RICE)



NodeId = 128 bits

Nodes and key place in a linear space (ring)

Mapping : a key is associated to the node with the numerically closest nodeId to the key

Pastry (MSR/Rice)

Naming space :

- Ring of 128 bit integers
- *nodeIds* chosen at random
- Identifiers are a set of digits in base 16

Key/node mapping

- key associated with the node with the numerically closest node id

Routing table:

- Matrix of 128/4 lines et 16 columns
- routeTable(i,j):

nodeId matching the current node identifier up to level I
with the next digit is j

Leaf set

- 8 or 16 closest numerical neighbors in the naming space

Proximity Metric

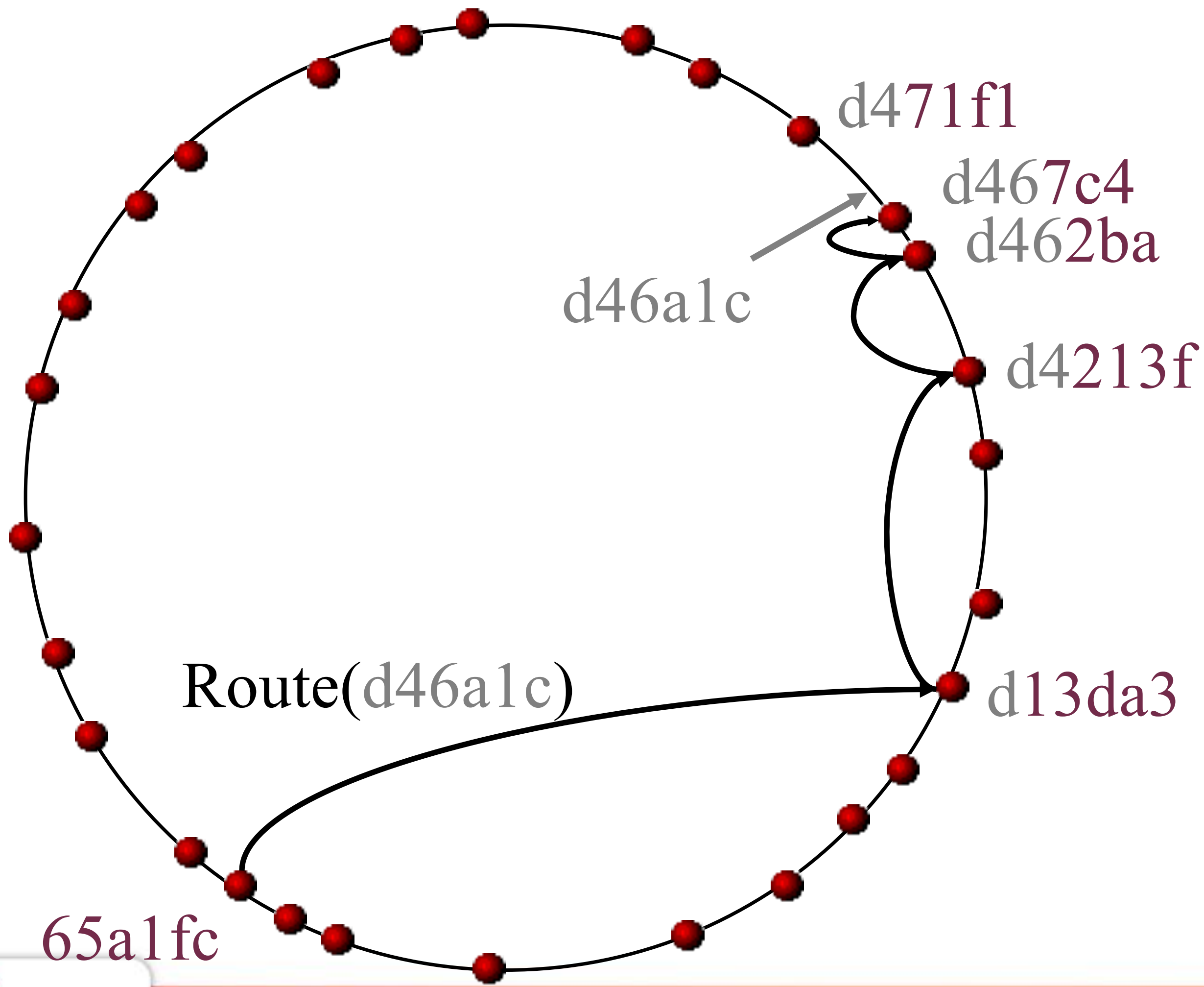
- *Bias selection of nodes*

Pastry: Routing table(#65a1fcx)

Line 0	*0x*	*1x*	*2x*	*3x*	*4x*	*5x*		*7x*	*8x*	*9x*	*ax*	*bx*	*cx*	*dx*	*ex*	*fx*
Line 1	*60x*	*61x*	*62x*	*63x*	*64x*		*66x*	*67x*	*68x*	*69x*	*6ax*	*6bx*	*6cx*	*6dx*	*6ex*	*6fx*
Line 2	*650x*	*651x*	*652x*	*653x*	*654x*	*655x*	*656x*	*657x*	*658x*	*659x*		*65bx*	*65cx*	*65dx*	*65ex*	*65fx*
Line 3	*65a0x*		*65a2x*	*65a3x*	*65a4x*	*65a5x*	*65a6x*	*65a7x*	*65a8x*	*65a9x*	*65aax*	*65abx*	*65acx*	*65adx*	*65aex*	*65afx*

$\log_{16} N$ liges

Pastry: Routing



Properties

$\log_{16} N$ hops

Size of the state maintained (routing table): O(*log N*)

Routing algorithm (on node A)

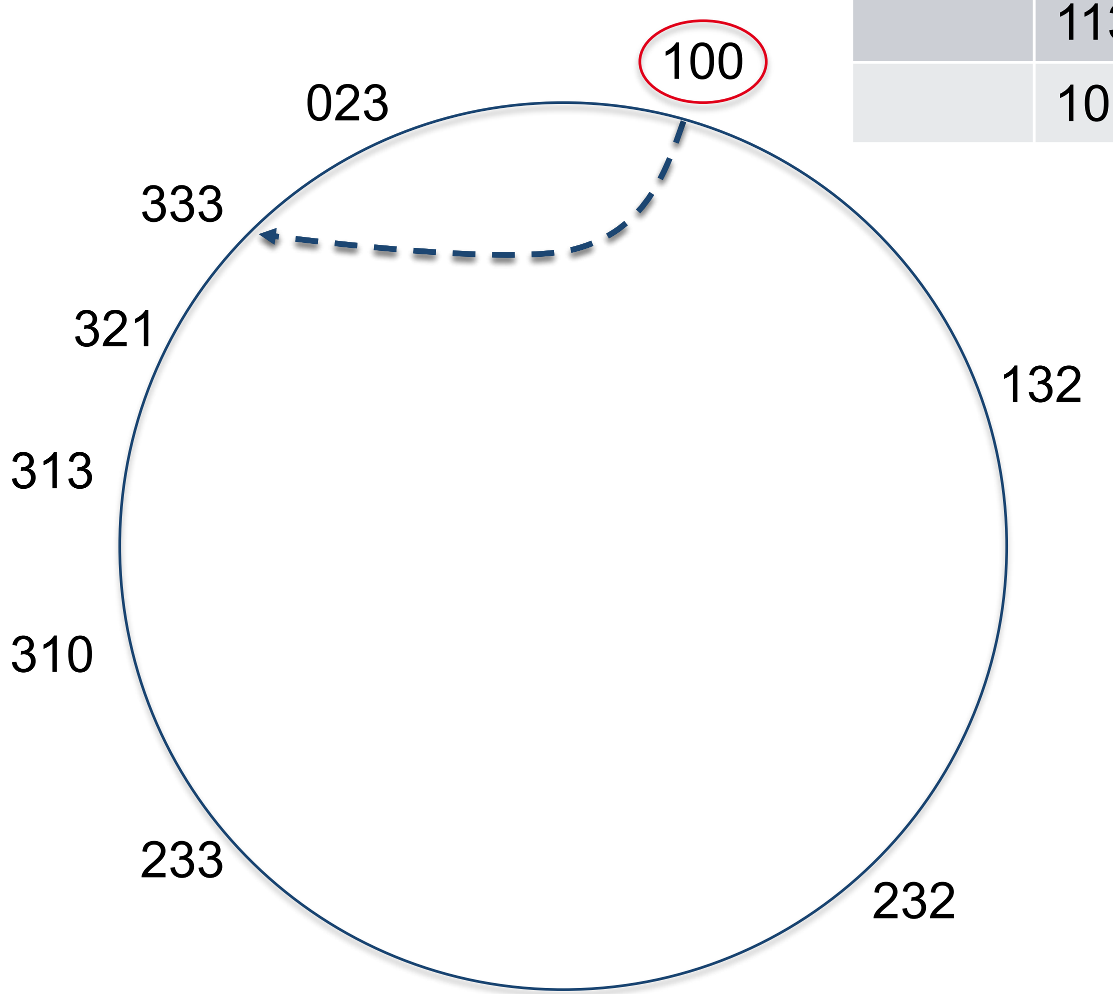
if $(L_{-\lfloor |L|/2 \rfloor} \leq D \leq L_{\lfloor |L|/2 \rfloor})$ {
    // D is within range of our *leaf set*
    forward to L_i, s.th. $|D - L_i|$ is minimal;
} else {
    // use the routing table
    Let $l = shl(D, A)$;
    if $(R_l^{D_l} \neq null)$ {
        forward to $R_l^{D_l}$;
    }
    else {
        // rare case
        forward to $T \in L \cup R \cup M$, s.th.
            $shl(T, D) \geq l$,
            $|T - D| < |A - D|$
    }
}

R_l^i : entry of the routing table R, $0 \leq i \leq 2^b$,
line l, $0 \leq l \leq \lfloor 128/b \rfloor$
L_i : ith closest nodeId in the leafset
D_l : value of the l digits of key D
$SHL(A, B)$: length of the shared prefix between A and B

Pastry Example

b=4

Route to 311

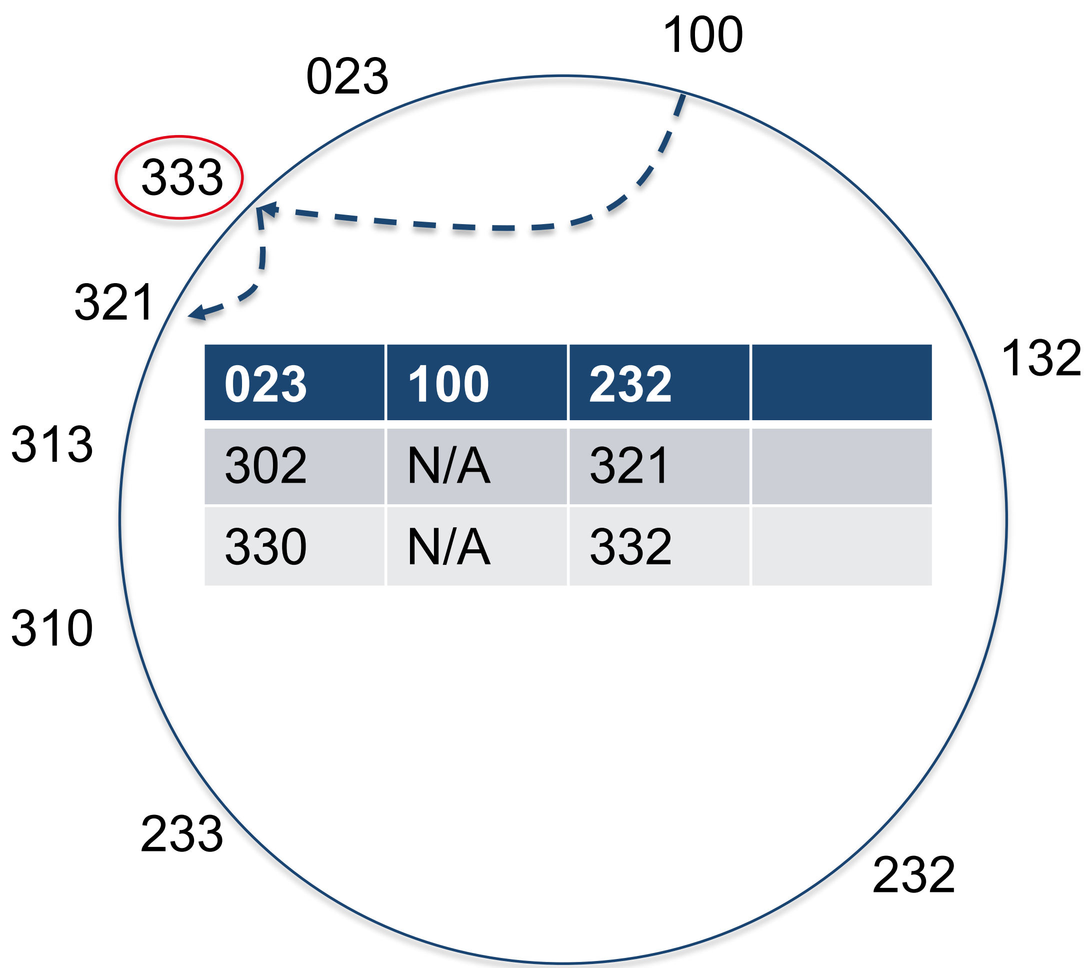


023		232	333
	113	122	132
	101	N/A	103

Pastry Example

b=4

Route to 311

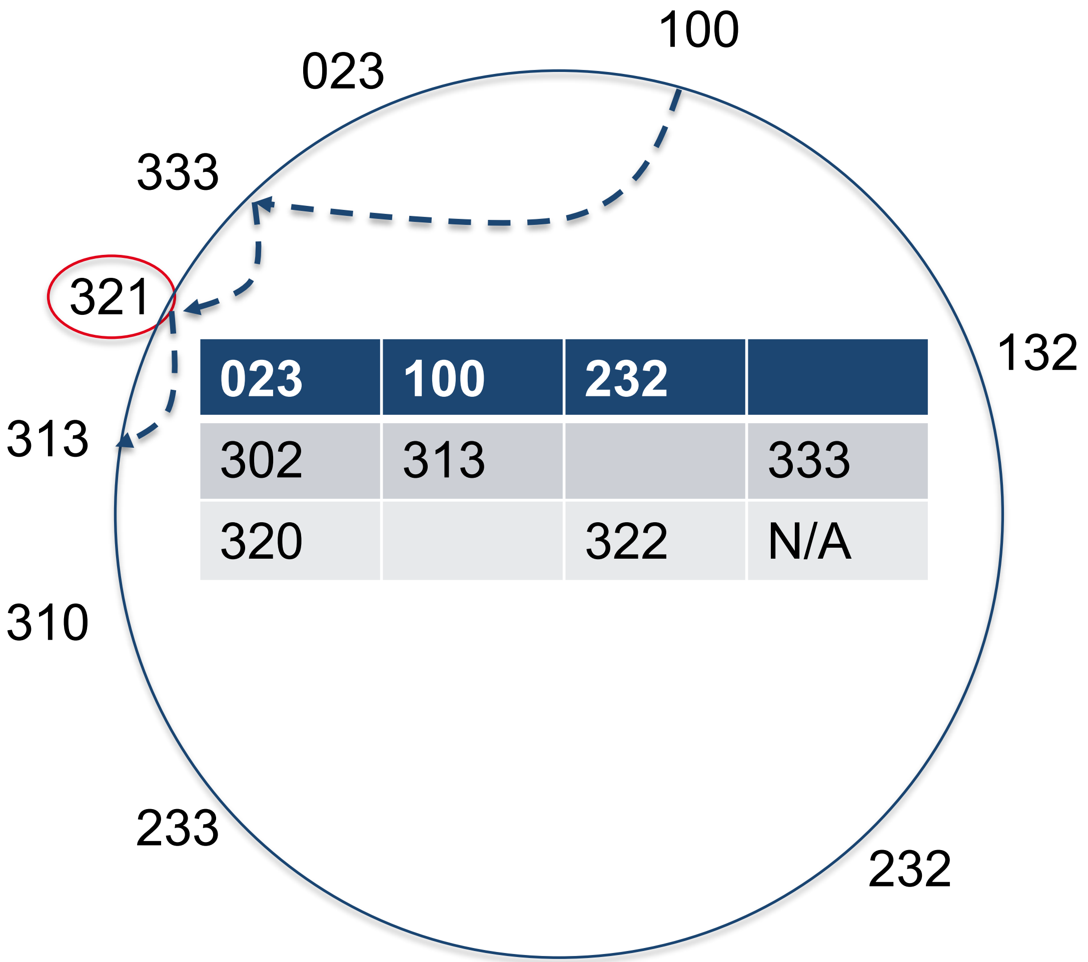


023	100	232	
302	N/A	321	
330	N/A	332	

Pastry Example

b=4

Route to 311

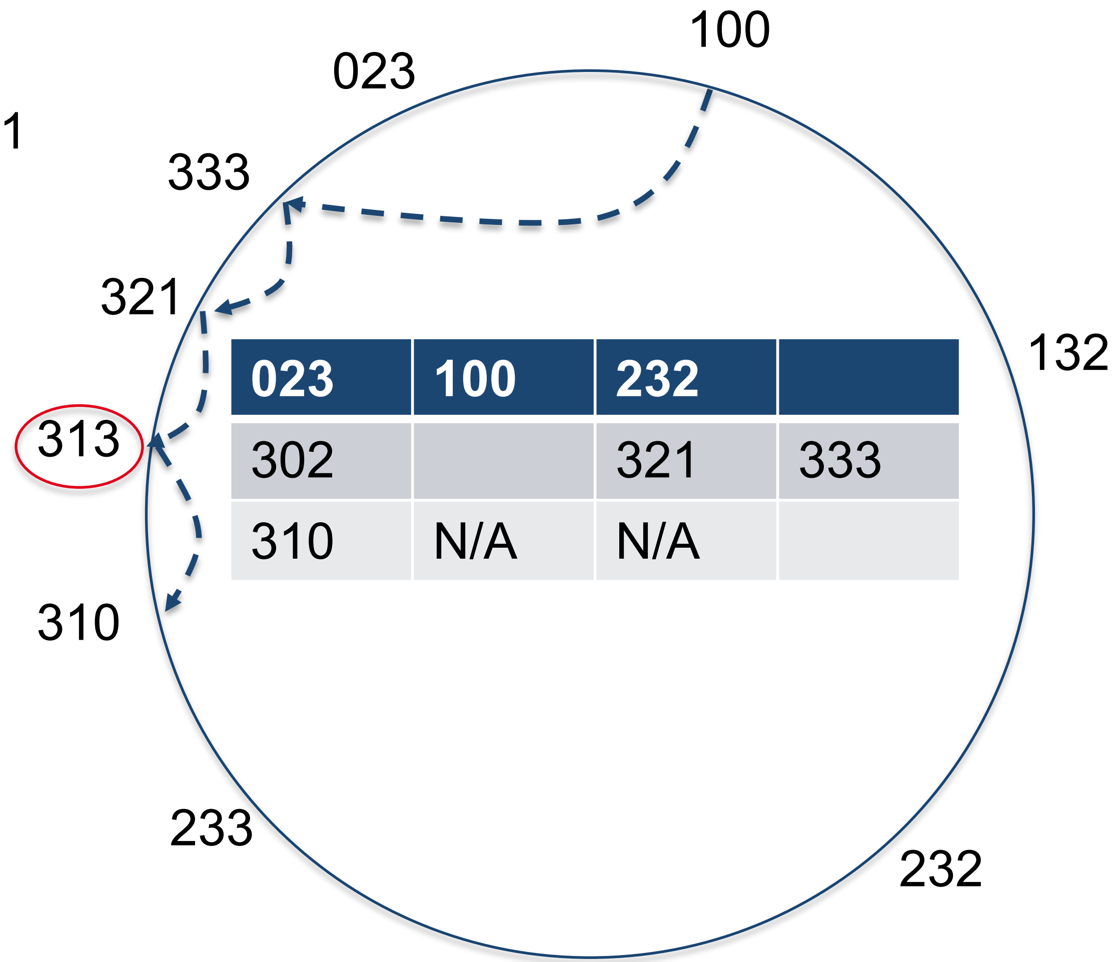


023	100	232	
302	313		333
320		322	N/A

Pastry Example

b=4

Route to 311

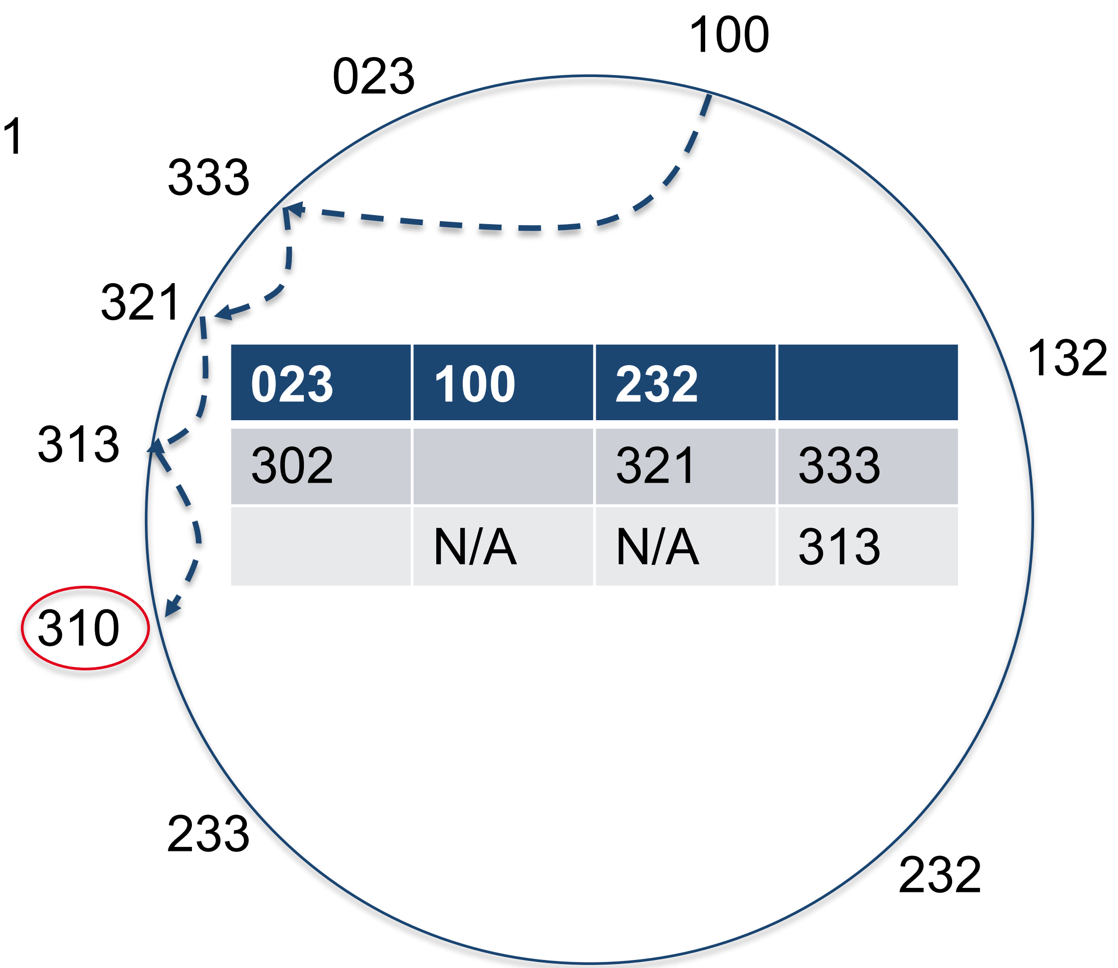


023	100	232	
302		321	333
310	N/A	N/A	

Pastry Example

b=4

Route to 311



023	100	232	
302		321	333
	N/A	N/A	313

Node departure

Explicit departure or failure

Replacement of a node

The leafset of the closest node in the leafset contains the closest

new node, not yet in the leafset

Update from the leafset information

Update the application

Failure detection

Detected when immediate neighbours in the name space (leafset) can no longer communicate

Detected when a contact fails during the routing

Routing uses an alternative route

Fixing the routing table of A

Repair

R_l^d : entry of the routing table of A to repair

A contacts another entry (at random) R_l^i from the same line so that $(i \neq d)$ and asks for entry R_l^d, otherwise another entry from R_{l+1}^i $(i \neq d)$ if no node in line l answers the request.

State maintenance

Leaf set

- is aggressively monitored and fixed

Routing table

- are lazily repaired

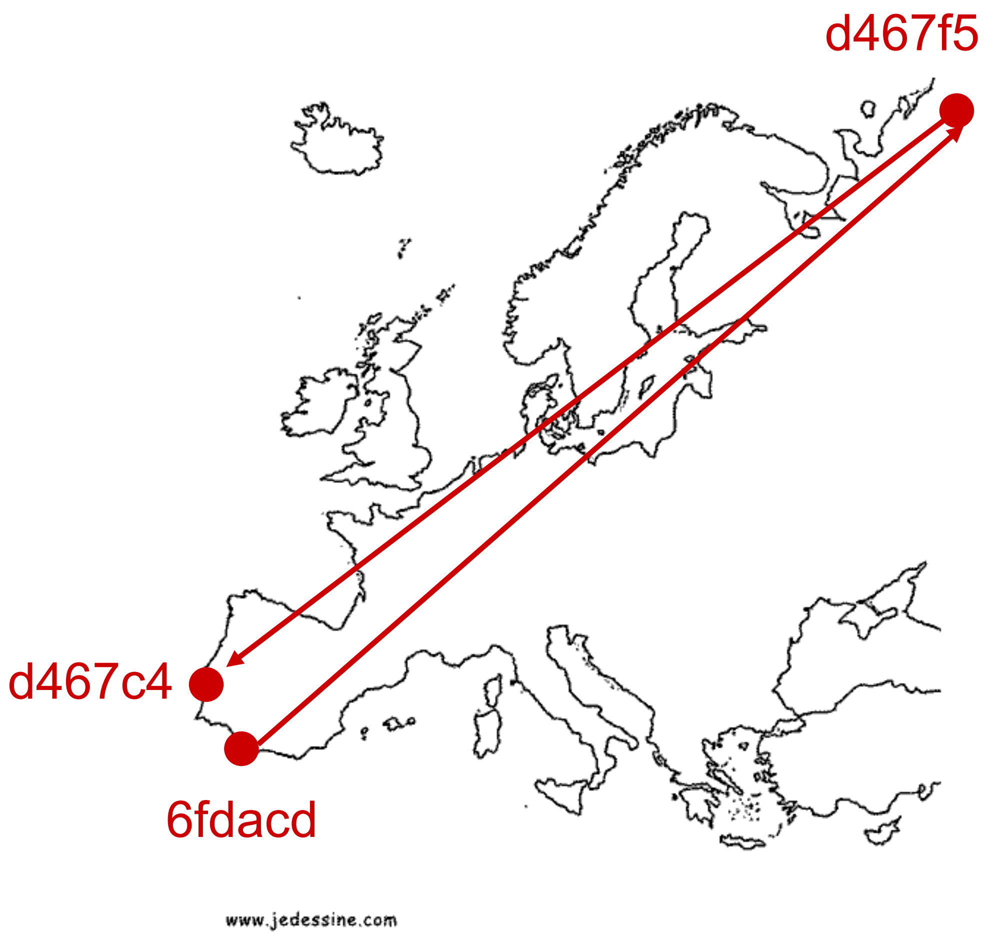
When a hole is detected during the routing

- Periodic gossip-based maintenance

Reducing latency

Random assignment of nodeId: Nodes numerically close are geographically (topologically) distant

Objective: fill the routing table with nodes so that routing hops are as short (latency wise) as possible



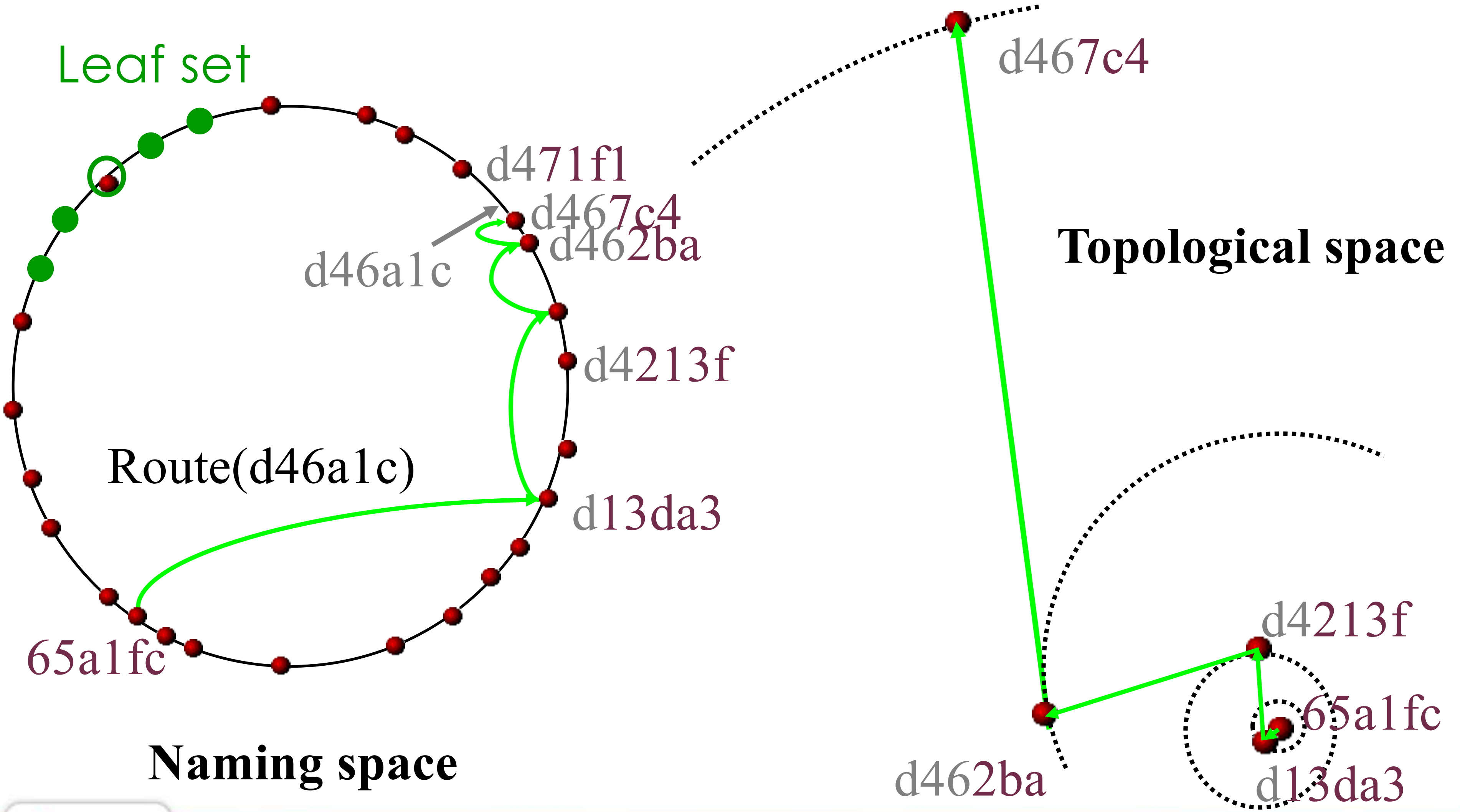
Topological Metric:

Exploiting locality in Pastry

Neighbour selected based of a network proximity metric:

- Closest topological node
- Satisfying the constraints of the routing table routeTable(i,j):
- *nodeId* corresponding to the current *nodeId* up to level i next digit = j
- nodes are close at the top level of the routing table
- Farther nodes at the bottom levels of the routing tables

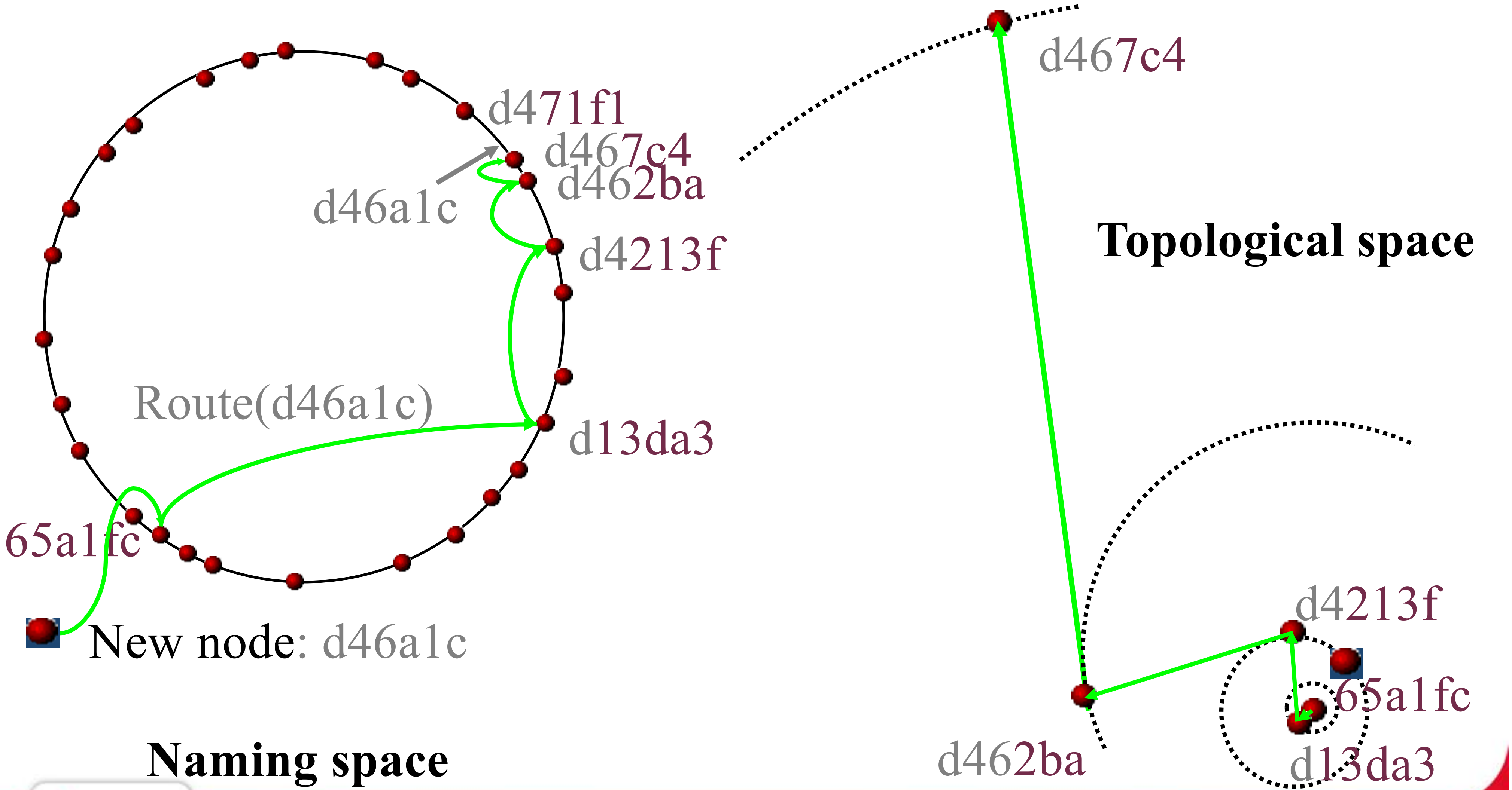
Proximity routing in Pastry



Locality

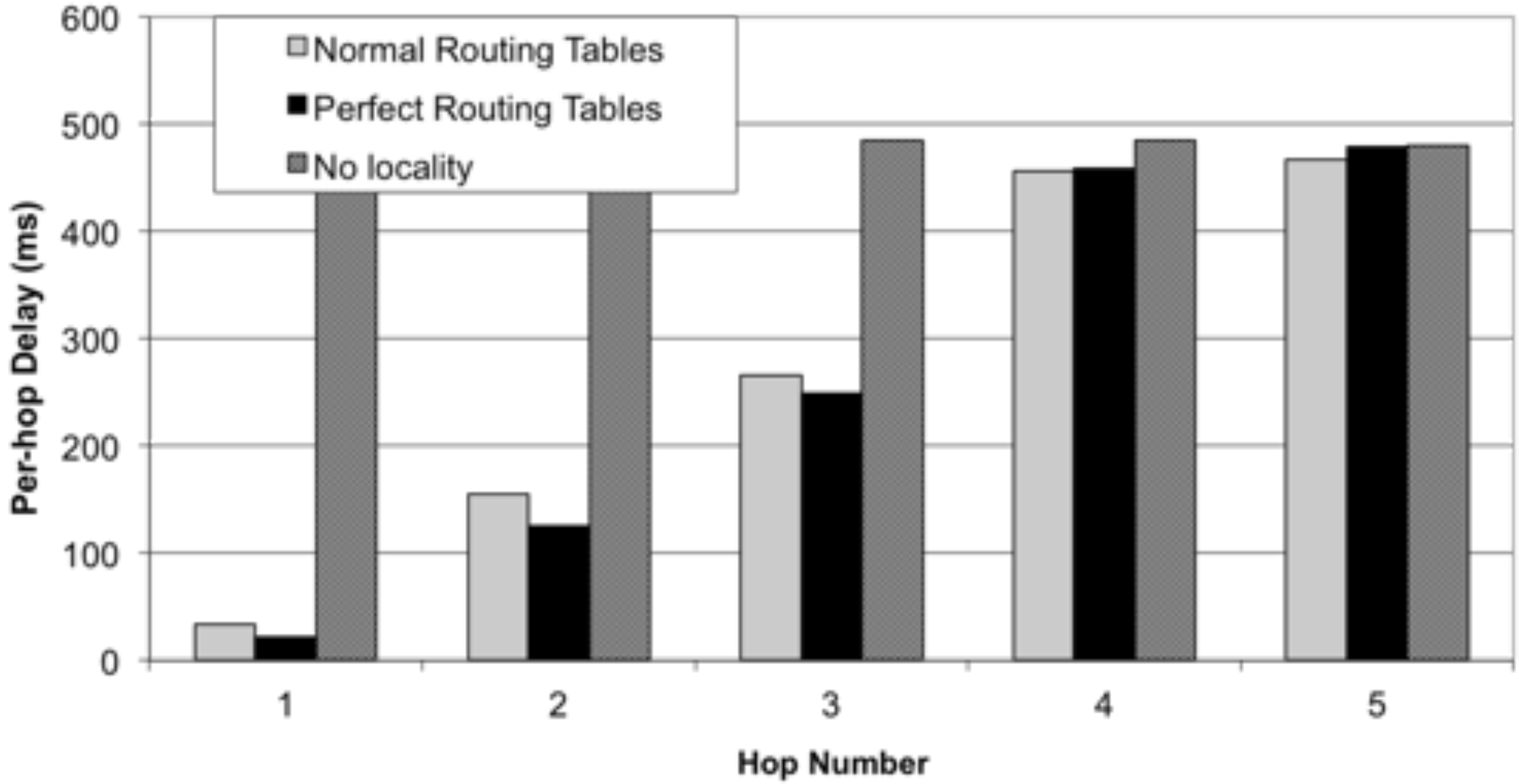
1. Joining node X routes asks A to route to X
- Path A,B,… -> Z
- Z numerically closest to X
- X initializes line i of its routing table with the contents of line i of the routing table of the *ith* node encountered on the path
2. Improving the quality of the routing table
- X asks to each node of its routing table its own routing state and compare distances
- Gossip-based update for each line (20mn)
    - Periodically, an entry is chosen at random in the routing table
    - Corresponding line of this entry sent
    - Evaluation of potential candidates
    - Replacement of better candidates
    - New nodes gradually integrated

Node insertion in Pastry



Performance

1.59 slower than IP on average



References

•Rowstron and P. Druschel, "**Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems**", *Middleware'2001*, Germany, November 2001.