The Renaissance of Decentralized Systems

Peter Druschel

Scientific Director

Max Planck Institute for Software Systems

previously

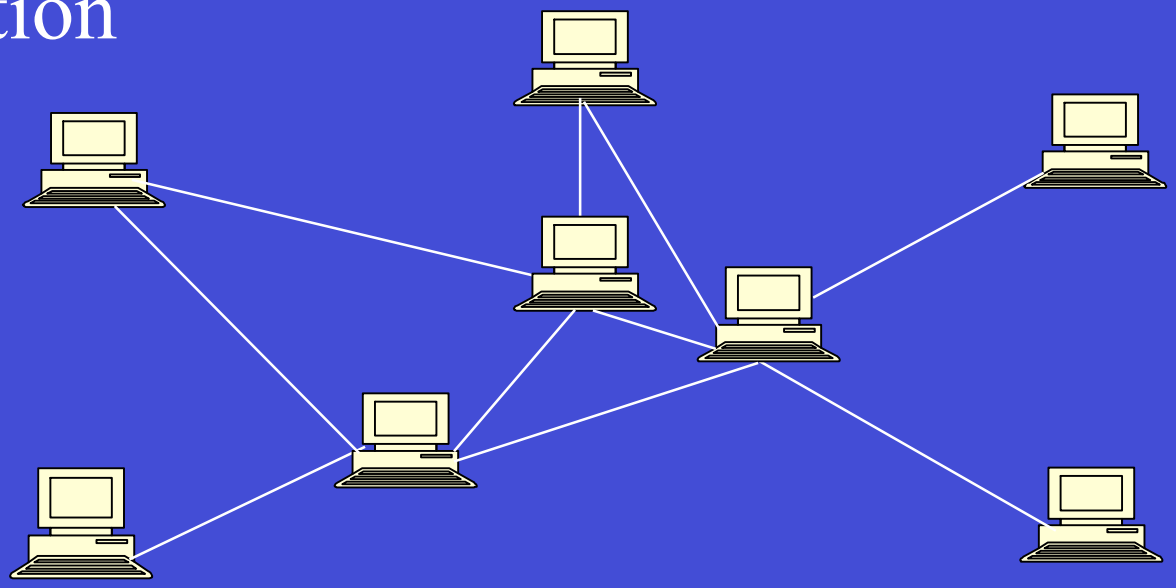
Professor of Computer Science and of Electrical and Computer Engineering

Rice University

Decentralized systems: Characteristics

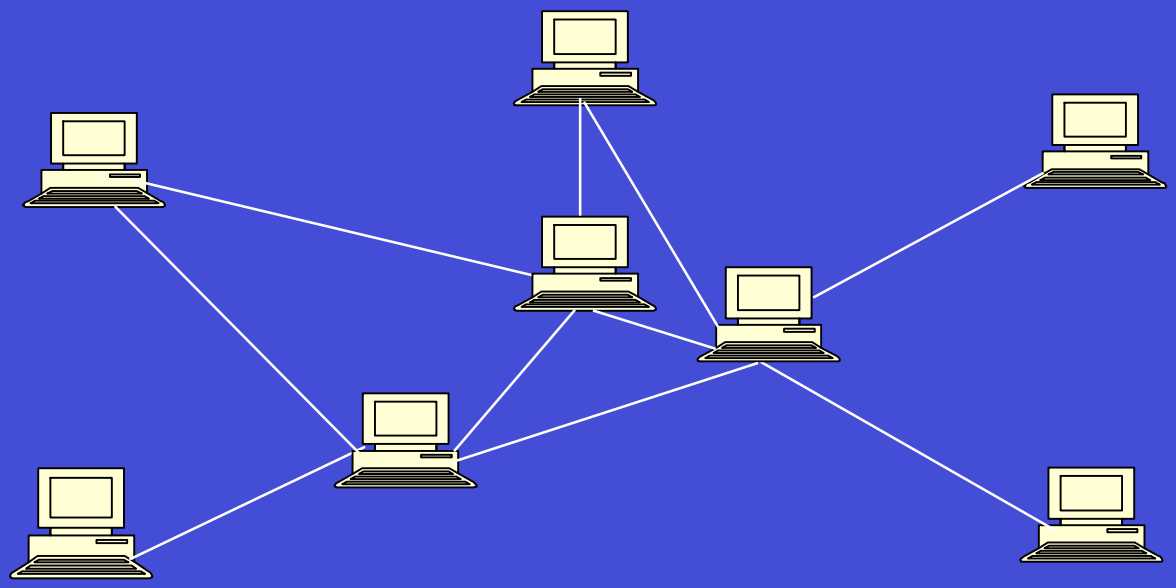
Distributed computer system with

- Symmetric components
- Decentralized control and state
- Self-organization



Decentralized systems: Promise

- “Organic” growth: Incremental deployment/operating costs
- Infrastructure independence
- Scalability
- Resilience



Decentralized systems: Examples

Cooperative systems:

- content sharing, distribution (Kazaa, BitTorrent)
- telephony, gaming, scientific computing

Ad hoc networks/systems:

- disaster relief, emergency response, military
- computing for the economically disadvantaged
- actor-/sensornetworks

Scalable, resilient network services:

- naming (SFR,CoDoNS), information management (PIER, Astrolabe, SDIMS)

Why decentralized now?

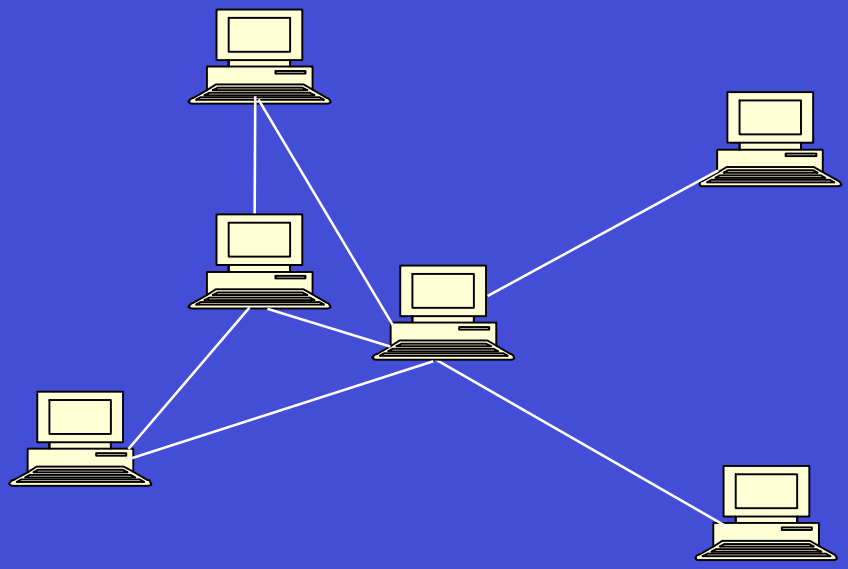
Demand pulls

- Cooperative computing
- Highly resilient services
- Ad hoc networks
- Large-scale distributed systems are difficult

Technology pushes

- Bigger, faster, and better: every PC can be a server
- Wireless devices
- Scalable lookup algorithms are available
- Trustworthy systems from untrusted components

Decentralized Systems Architecture



Applications

Decentralized application support (Overlay)

Routing (IP, DSR, AODV)

Decentralized application support: Research challenges

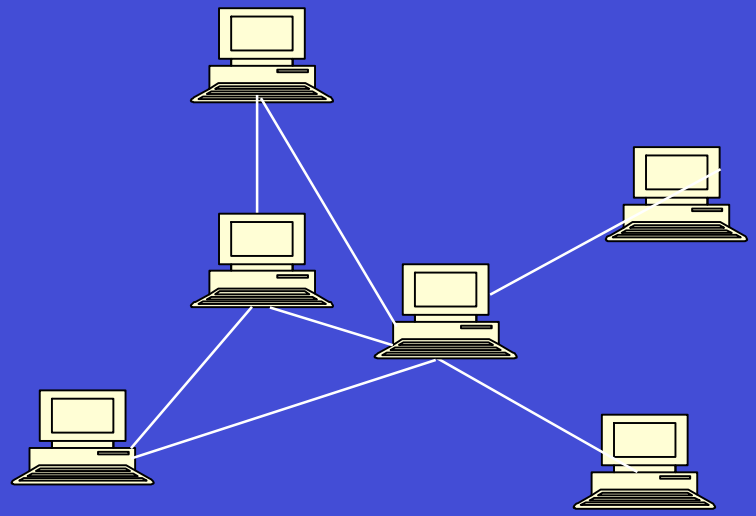
1. Scalable object lookup
2. Network-awareness for performance
3. Security
4. Persistent data
5. Consistency
6. Load balancing
7. Heterogeneity
8. Coping with dynamic membership/mobility
9. Coping with firewalls, NATs, intranets
10. Programming abstraction
11. Anonymity/anti-censorship

Outline

- Decentralized systems
- Challenges
  - Object location
  - Overlay route efficiency
- Key abstractions:
  - Distributed Hash Tables (DHT)
  - Group communication (Scribe)
- More challenges
  - Malicious participants
  - Correlated failures
- Putting it all together: ePOST

Challenge: Object location

- Resources partitioned among participating nodes
- Mapping from objects to nodes is dynamic



Point-to-point connectivity insufficient

- don’t know who to talk to
- don’t know where to store objects
- want to address *resources* (*objects)*, not *nodes*!

Object location: Approaches

Unstructured overlays

- Resources placed arbitrarily
- Arbitrary queries flooded through network
- Cannot achieve both scalability and recall

Structured overlays

- Objects uniquely mapped to a (set of) live node
- The mapping is maintained despite churn
- Reliable and efficient object lookup function
- Limited to identity lookup

Structured overlay networks

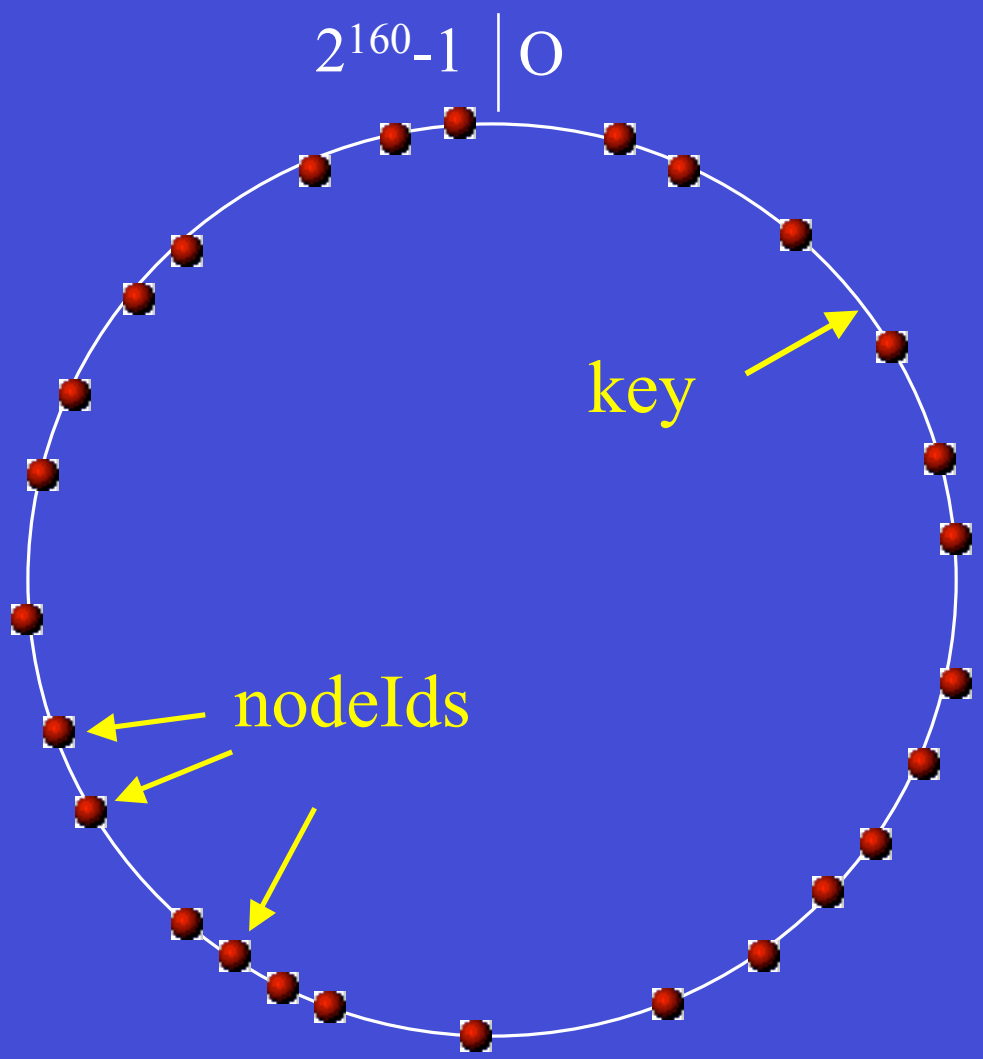
Primitive:

route(M, X): route message *M* to live node that is currently responsible for *X*

Example: **Pastry**

(others include Chord, CAN, Tapestry,Bamboo, Kademlia, SkipNet, Kelips, Accordeon, etc.)

Pastry: Identifier space



Consistent hashing
[*Karger et al. '97*]

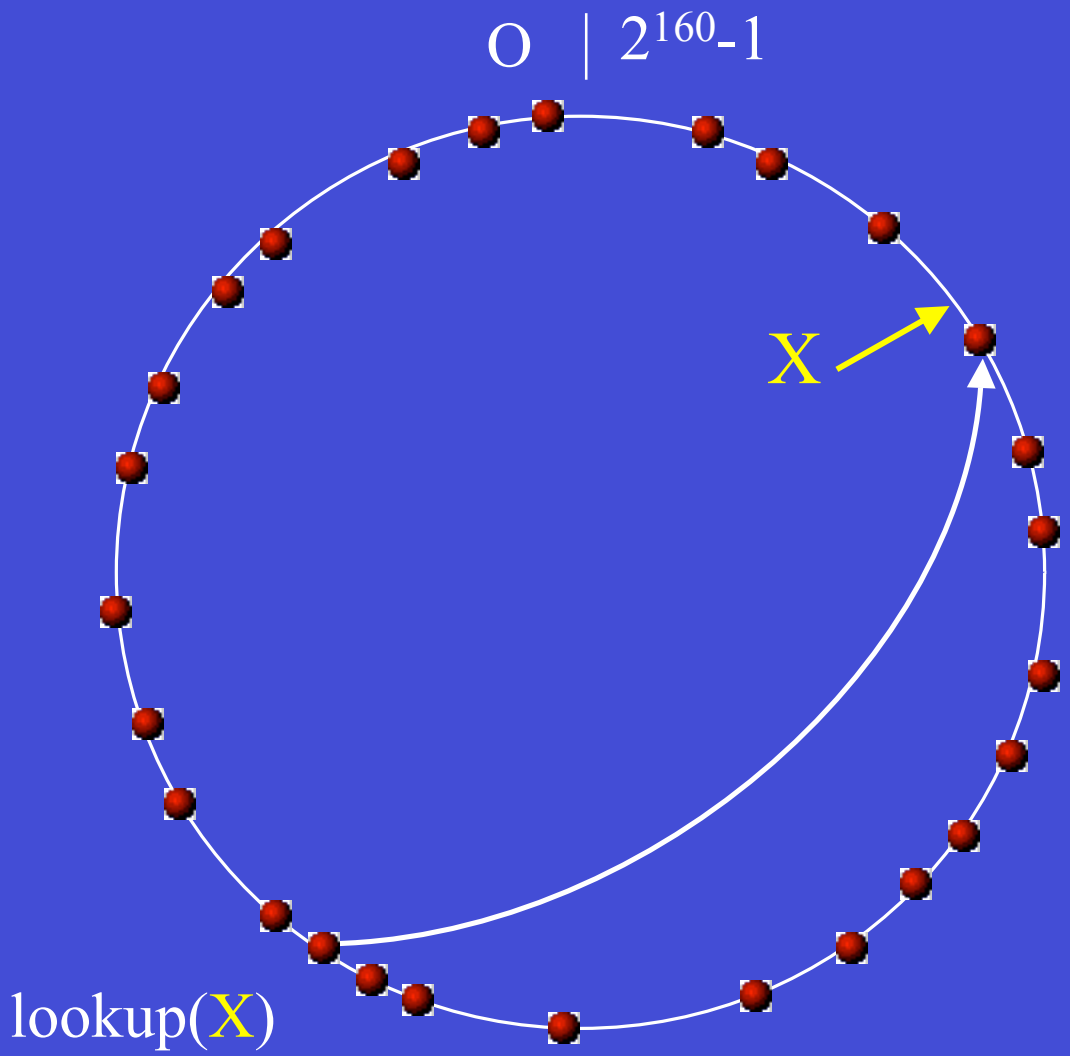
160 bit circular id space

nodeIds (uniform random)

keys (uniform random)

Each key is mapped to the live node with "closest" nodeId

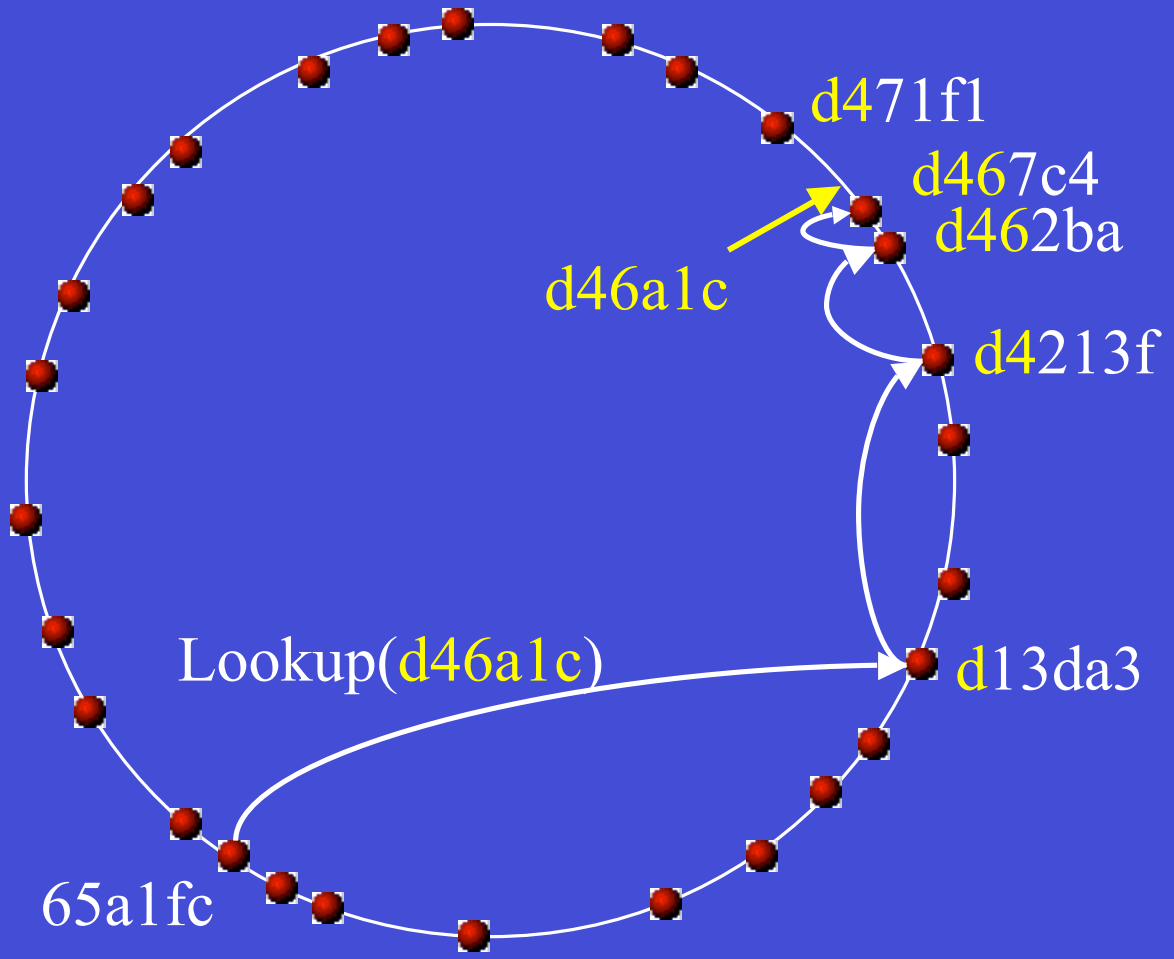
Pastry: Lookup



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

Problem:
complete routing table not scalable

Pastry: Prefix-based routing



Properties

- $\log_{16}$ N steps
- O(*log N*) state

Pastry: Routing table (# 65a1fc*x*)

Row 0	0 x	1 x	2 x	3 x	4 x	5 x		7 x	8 x	9 x	a x	b x	c x	d x	e x	f x
Row 1	6 0 x	6 1 x	6 2 x	6 3 x	6 4 x		6 6 x	6 7 x	6 8 x	6 9 x	6 a x	6 b x	6 c x	6 d x	6 e x	6 f x
Row 2	6 5 0 x	6 5 1 x	6 5 2 x	6 5 3 x	6 5 4 x	6 5 5 x	6 5 6 x	6 5 7 x	6 5 8 x	6 5 9 x		6 5 b x	6 5 c x	6 5 d x	6 5 e x	6 5 f x
Row 3	6 5 a 0 x		6 5 a 2 x	6 5 a 3 x	6 5 a 4 x	6 5 a 5 x	6 5 a 6 x	6 5 a 7 x	6 5 a 8 x	6 5 a 9 x	6 5 a a x	6 5 a b x	6 5 a c x	6 5 a d x	6 5 a e x	6 5 a f x

$log_{16} N$ rows

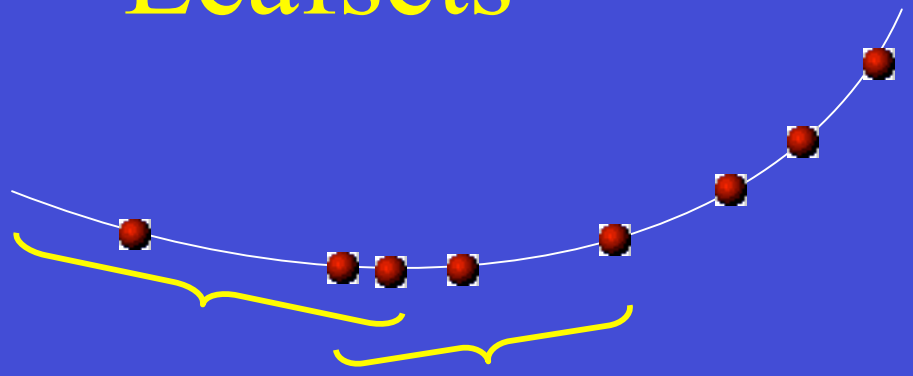
Pastry: Prefix-based routing

Similar to Plaxton Trees [Plaxton et al. '97]

But: Added

- *Leafsets* for consistency, robustness, security
- *Self-organization* (dynamic joins, fault tolerance)
- *Proximity neighbor selection* for efficiency
- *Secure routing* to defend against malicious nodes

Leafsets



Stabilization protocol ensures eventual consistency

- aids routing consistency
- enables secure routing
- localizes fault detection within set
- enables application-specific local coordination

(e.g., object replica management)

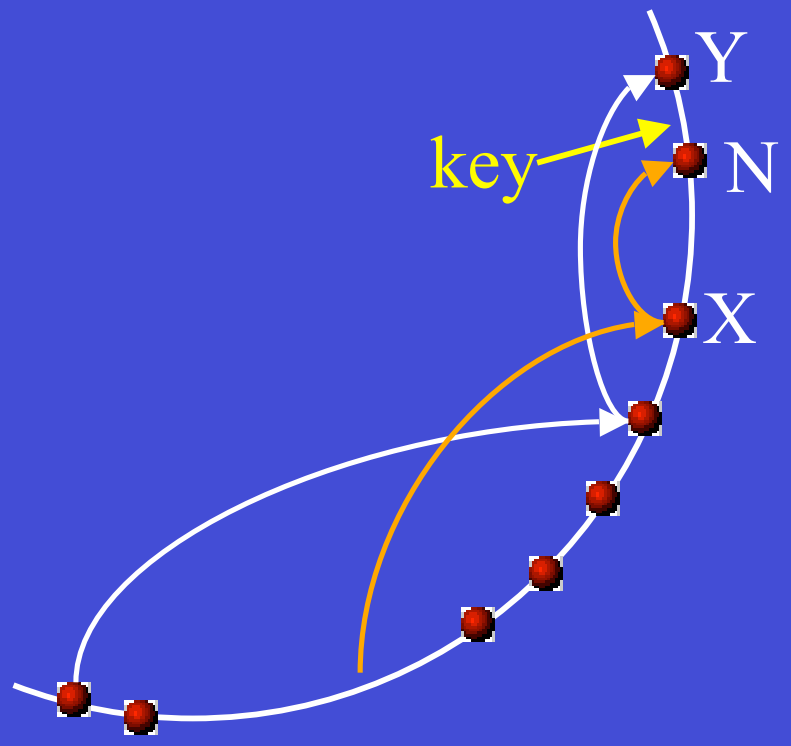
Challenge: Inconsistent routing

Routing consistency:

"At any point in time, at most one overlay node accepts messages with a given key"

Necessary for consistency of mutable data

New node N has informed X, but not yet Y of its arrival

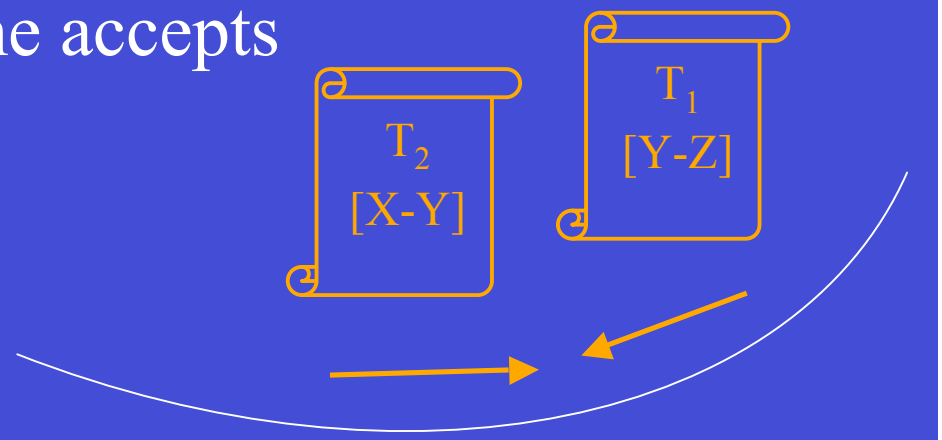


Ensuring routing consistency

- To accept a message with key *k*, a node requires a lease from its neighbors, for an interval $X<k<Z$
- Lease can be issued if grantor has a lease and previous lease has expired

Ensures:

- properly formed ring (eventually)
- at most one node at a time accepts messages with key k
- => routing consistency

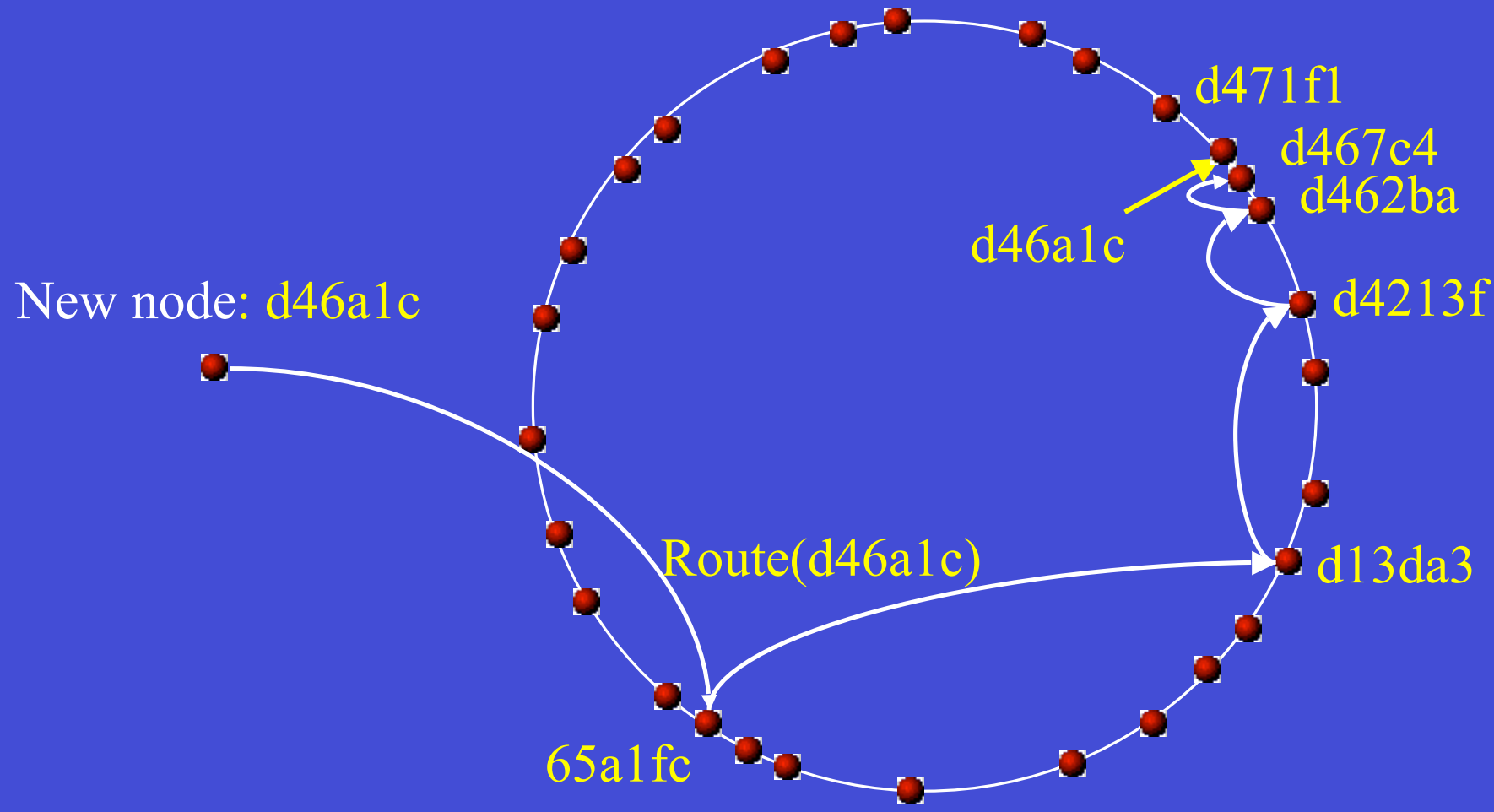


Challenge: Self-organization

Initializing and maintaining node state (overlay construction and maintenance)

- Node addition
- Node departure (failure)

Pastry: Node join

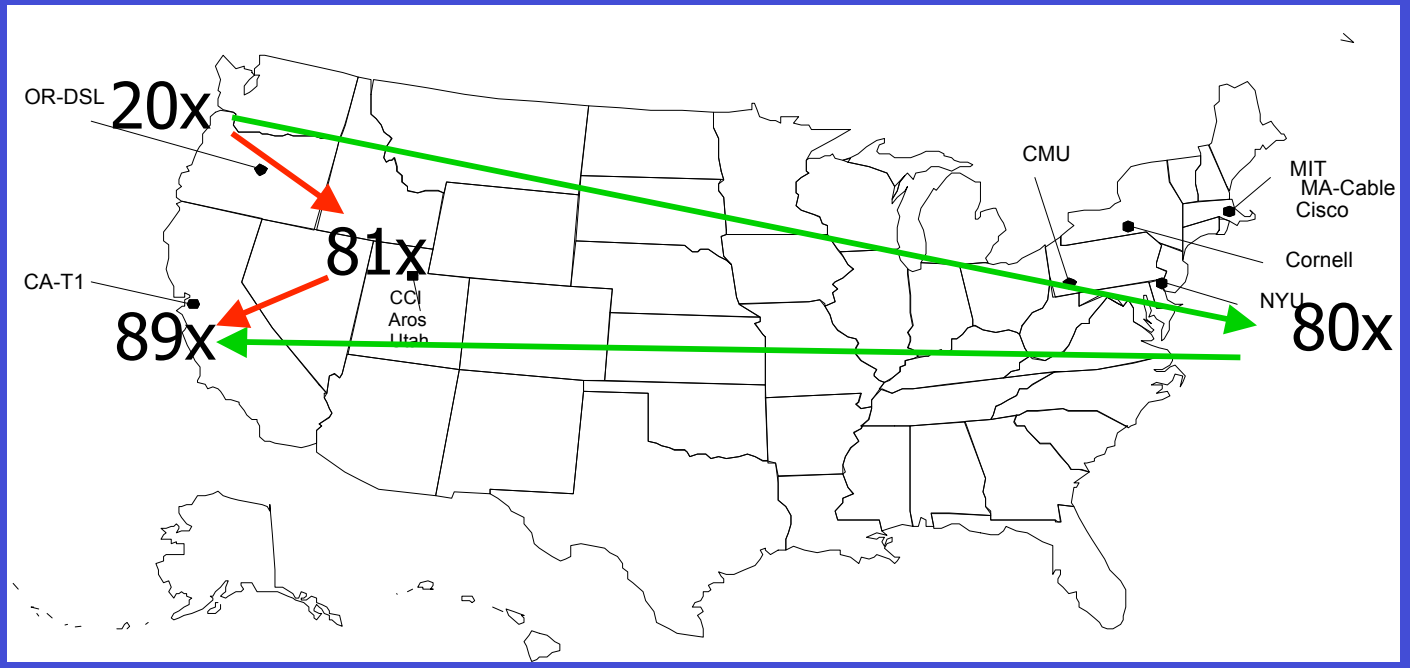


Pastry: Node departure (failure)

Leaf set members exchange keep-alive messages (failure detection, leaf set stabilization)

- **Leaf set repair (eager):** request set from farthest live node in set
- **Routing table repair (lazy):** get table from peers in the same row, then higher rows

Challenge: Overlay route efficiency



- Nodes *close* in id space, but *far away* in Internet
- **Goal:** choose routing table entries that yield few hops *and* low latency

Solution: Proximity neighbor selection (PNS)

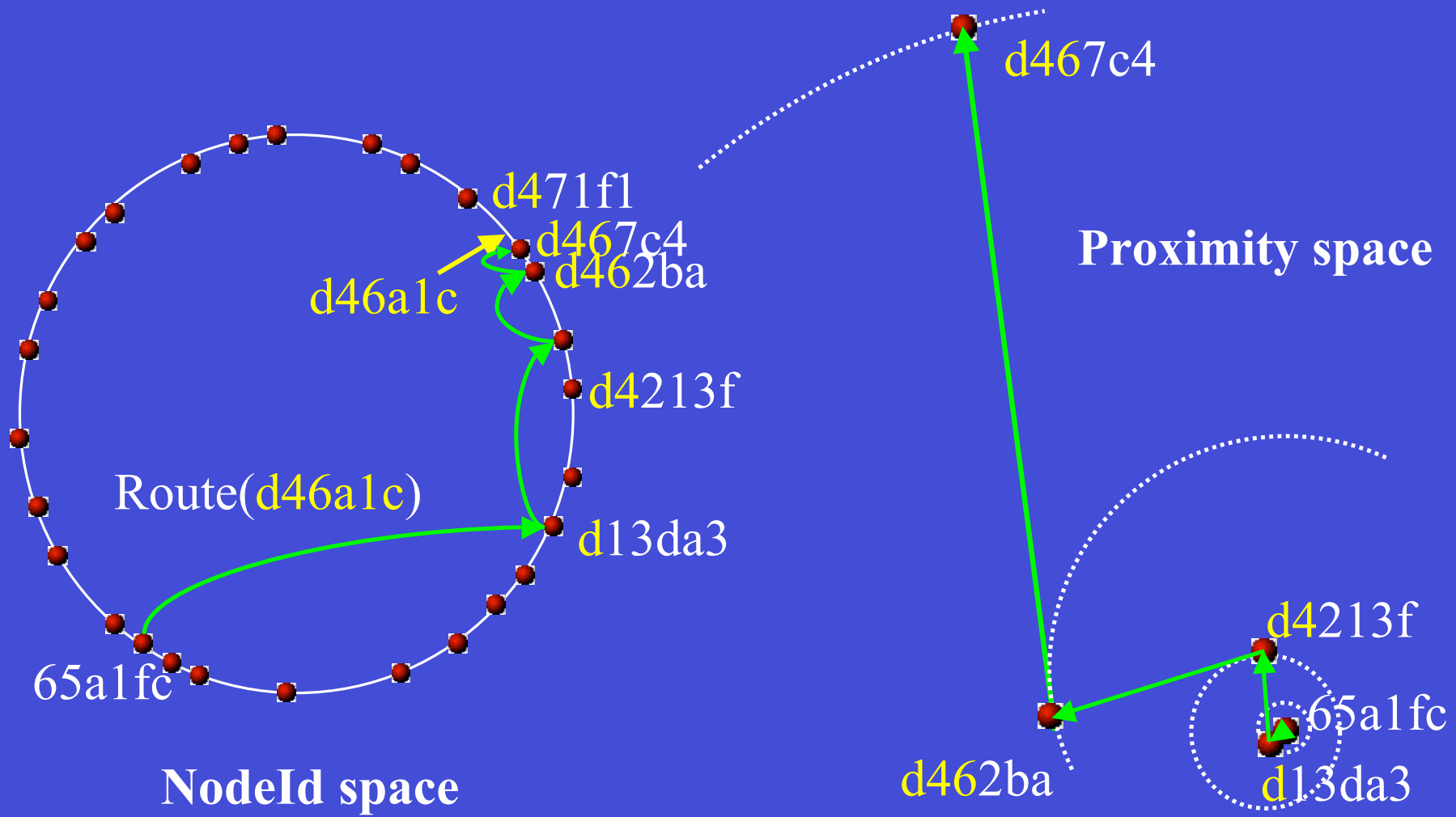
Assumptions:

- scalar proximity metric (e.g., RTT)
- a node can probe distance to any other node

Proximity invariant:

Each routing table entry refers to a node close to the local node (in the network), among all nodes with the appropriate nodeId prefix.

PNS: Routes in proximity space



PNS Properties

1) **Low-delay routes:** *Average delay penalty, relative to IP, is a small constant (1.3 - 2.2)*

2) **Route convergence:** *Routes of messages sent by nearby nodes with same keys converge at a node near the source nodes*

Outline

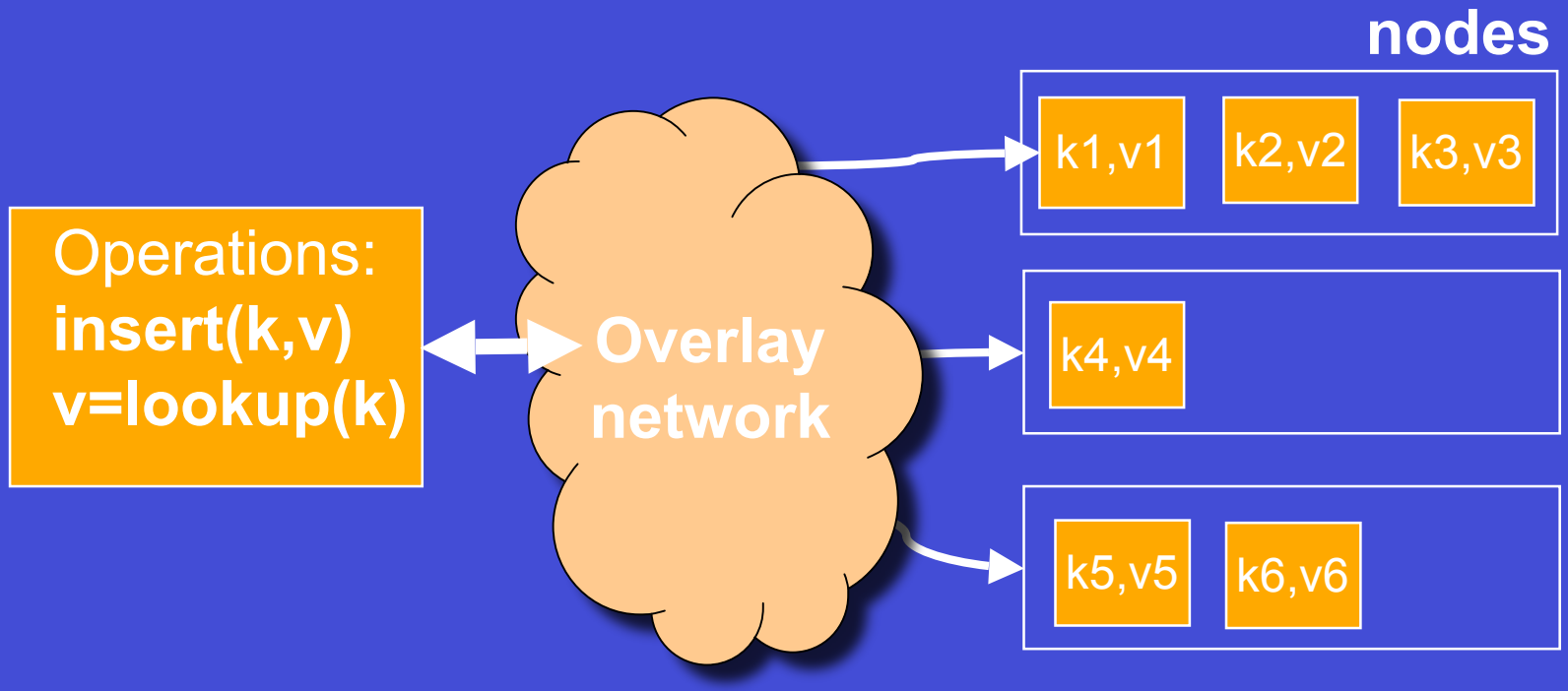
- Decentralized systems
- Challenges
  - Object location
  - Overlay route efficiency
- Key abstractions:
  - Distributed Hash Tables (DHT)
  - Group communication (Scribe)
- More challenges
  - Malicious participants
  - Correlated failures
- Putting it all together: ePOST

Sharing state: Distributed hash tables (DHT)

- Layered on top of a structured overlay
- Scalability, Robustness
- Persistence storage
- High availability

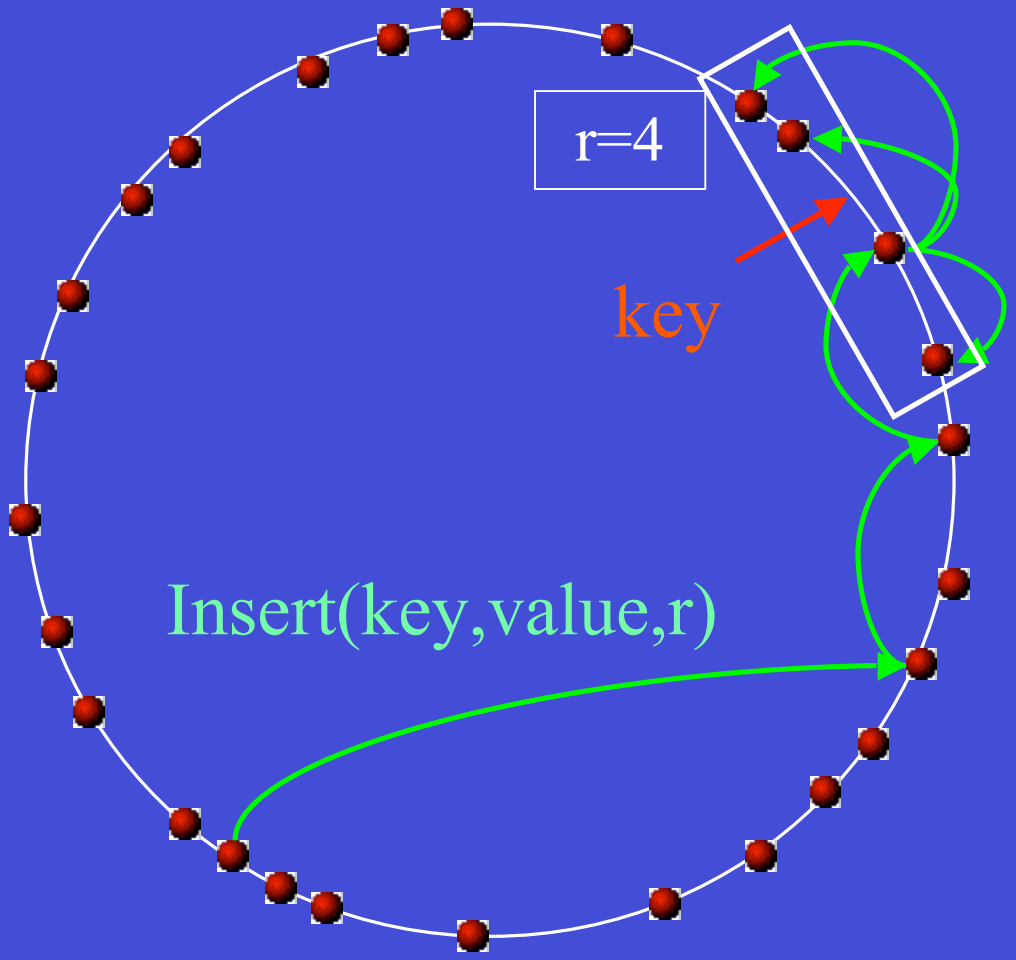
- Example: PAST [SOSP'01]

Distributed hash table (DHT)



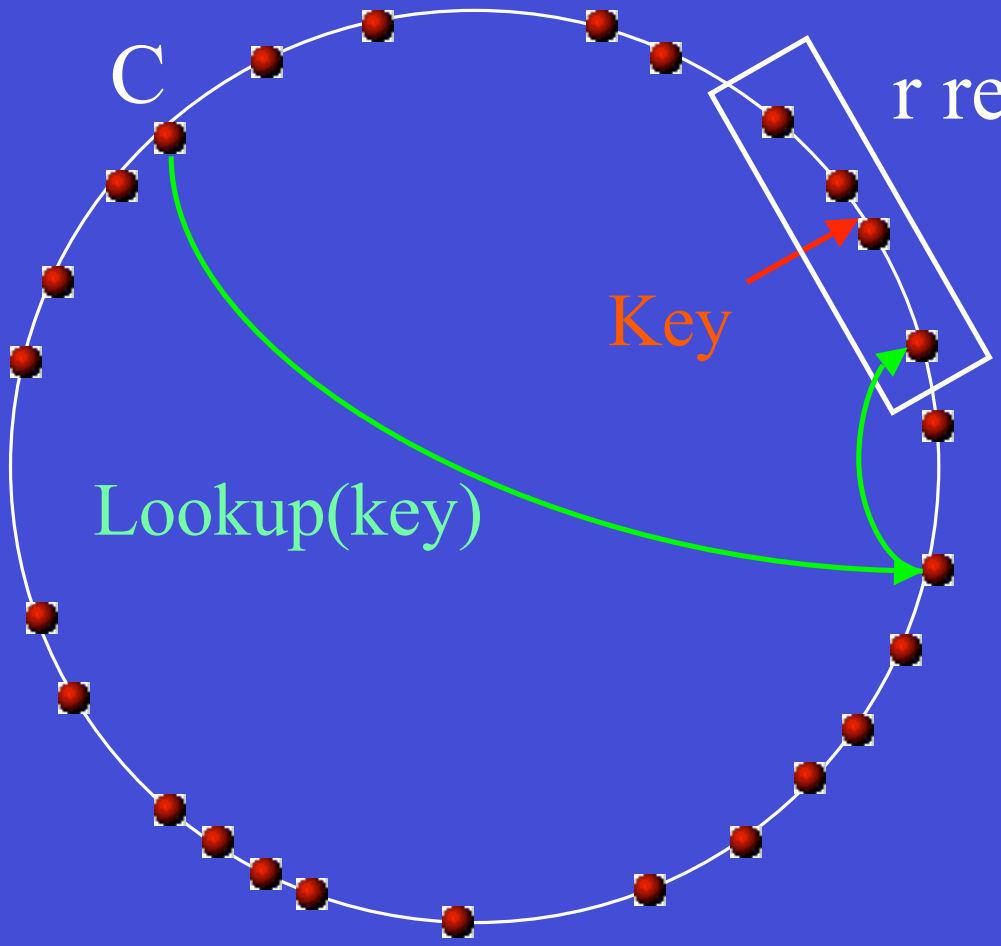
- Structured overlay maps keys to nodes
- Decentralized and self-organizing
- Scalable, robust

DHT: Insertion and replication



Storage Invariant: Tuple replicas are stored on *r* nodes with *nodeIds* closest to *key*

DHT: Lookup



Object located in $\log_{16}$ N steps (expected)

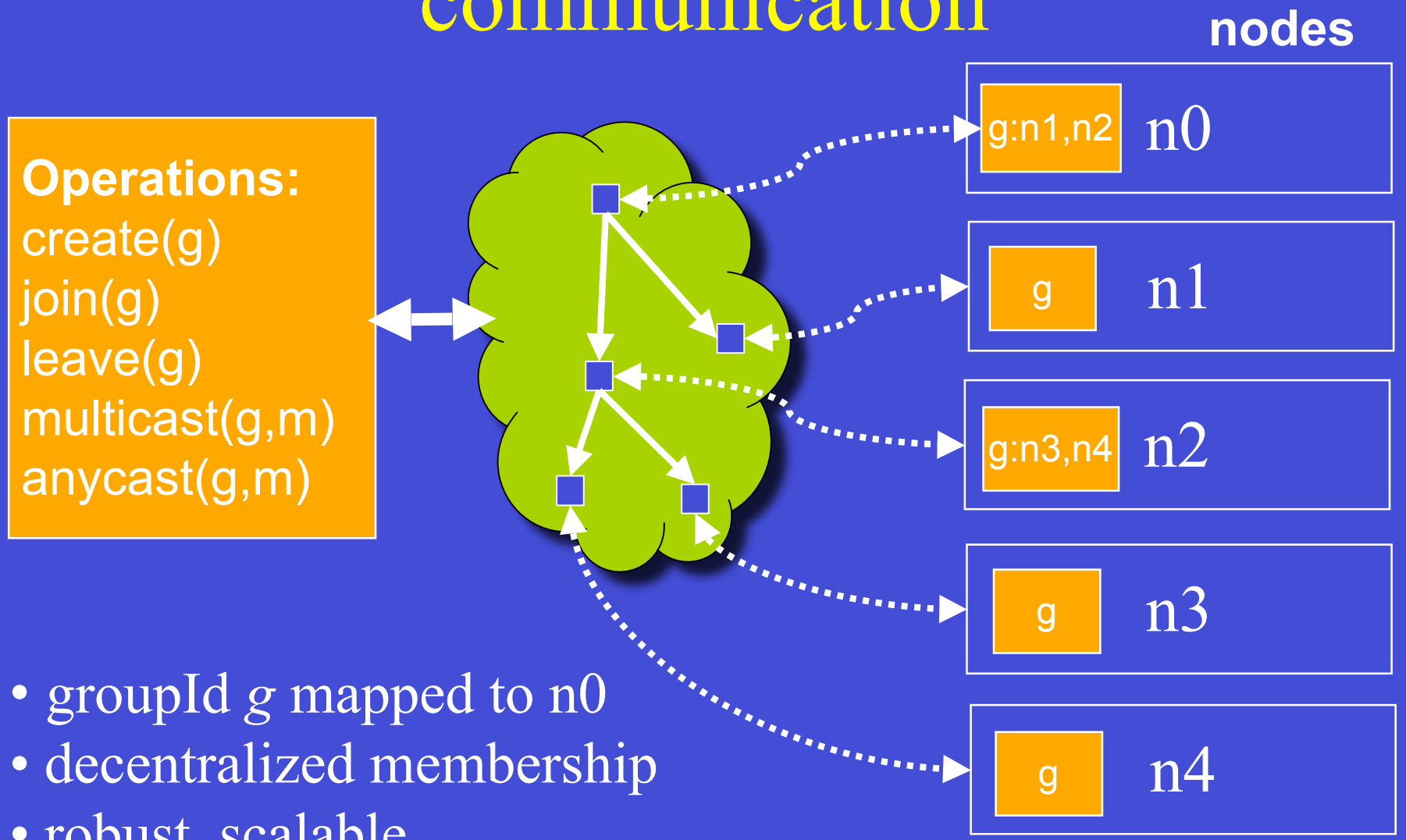
usually locates replica nearest client C

Coordination: Cooperative group communication

- Scribe: Tree-based group management
- Multicast, anycast, manycast primitives
- Scalable: large numbers of groups, members, wide range of members/group, dynamic membership

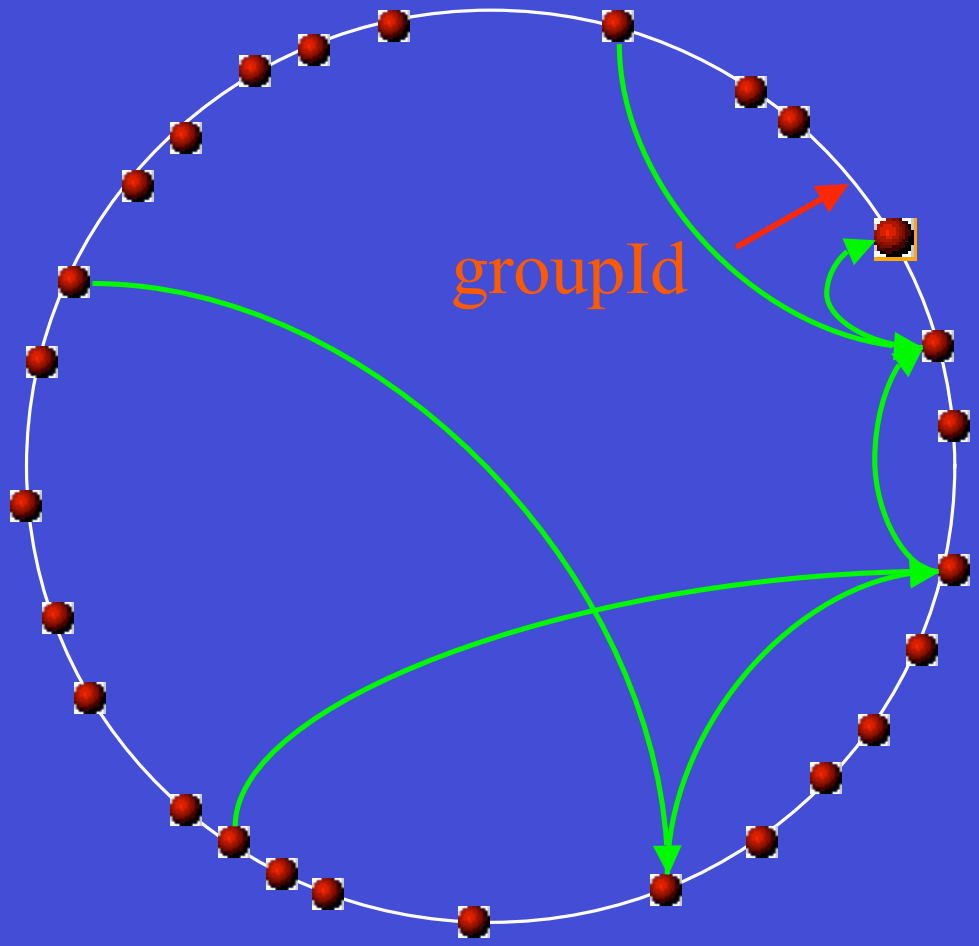
[IEEE JSAC ’02]

Cooperative group communication

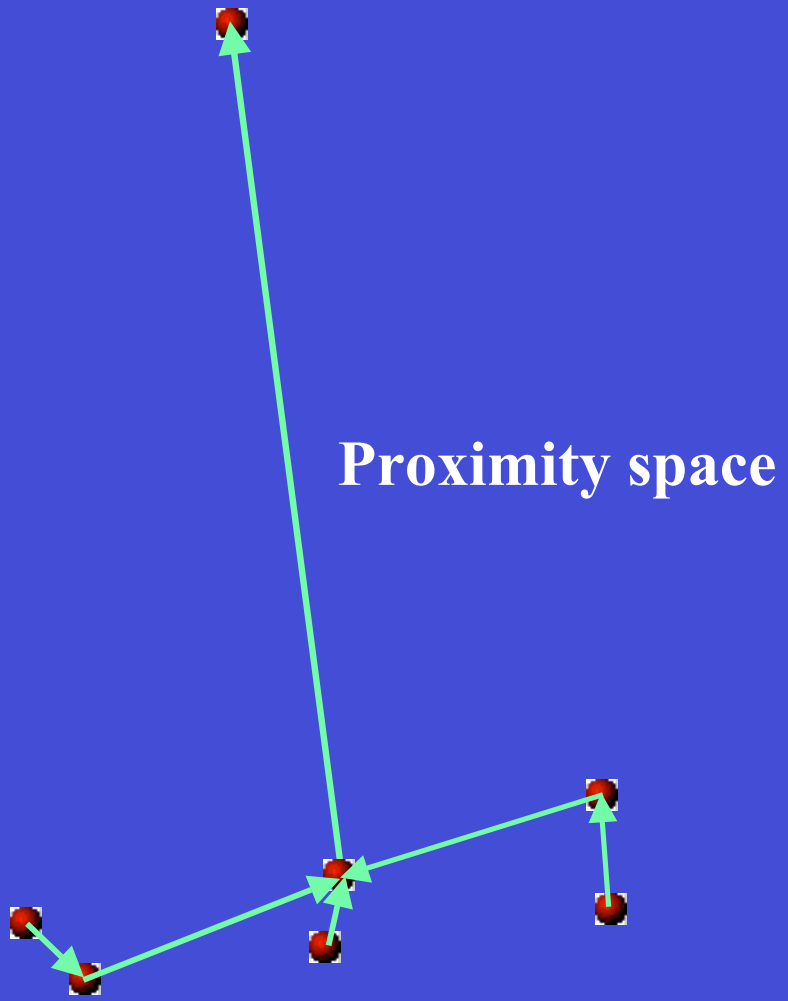


- groupId *g* mapped to n0
- decentralized membership
- robust, scalable

Scribe



Join(*groupId*)



Outline

- Decentralized systems
- Challenges
  - Object location
  - Overlay route efficiency
- Key abstractions:
  - Distributed Hash Tables (DHT)
  - Group communication (Scribe)
- More challenges
  - Malicious participants
  - Correlated failures
- Putting it all together: ePOST

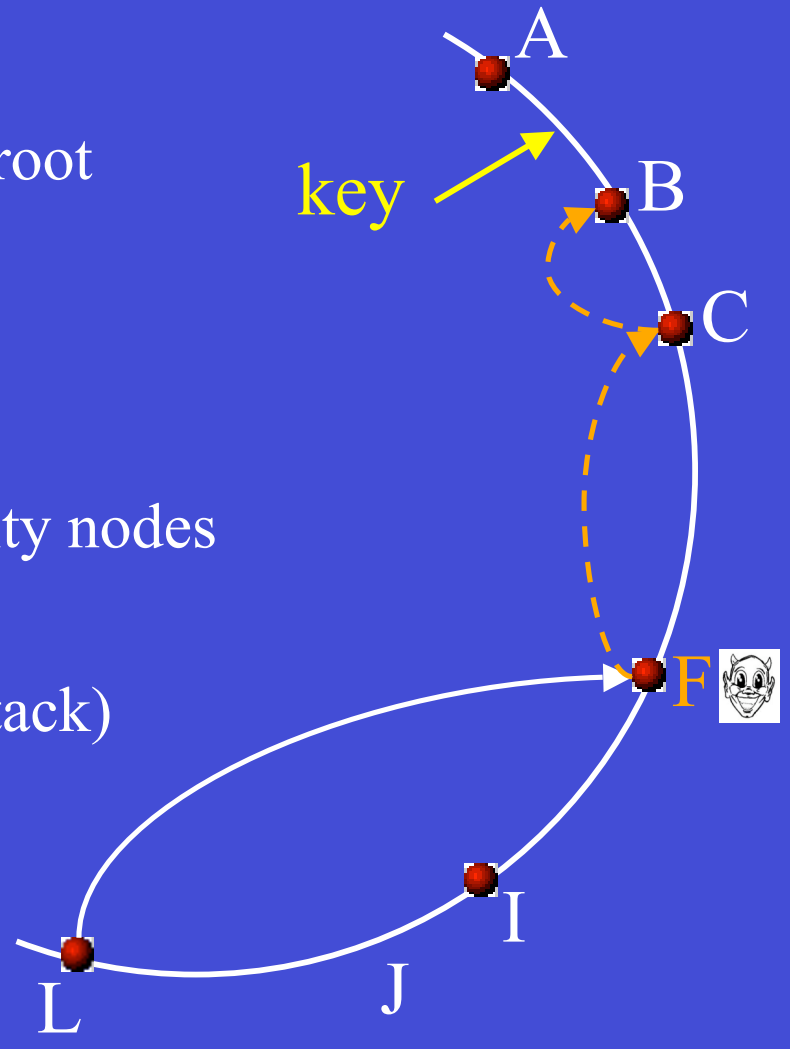
Challenge: Malicious participants

Prevent messages from reaching root

- drop or corrupt
- bias routing tables

Cause objects to be placed on faulty nodes

- choose nodeId values
- use many identities (Sybil attack)
- impersonate root



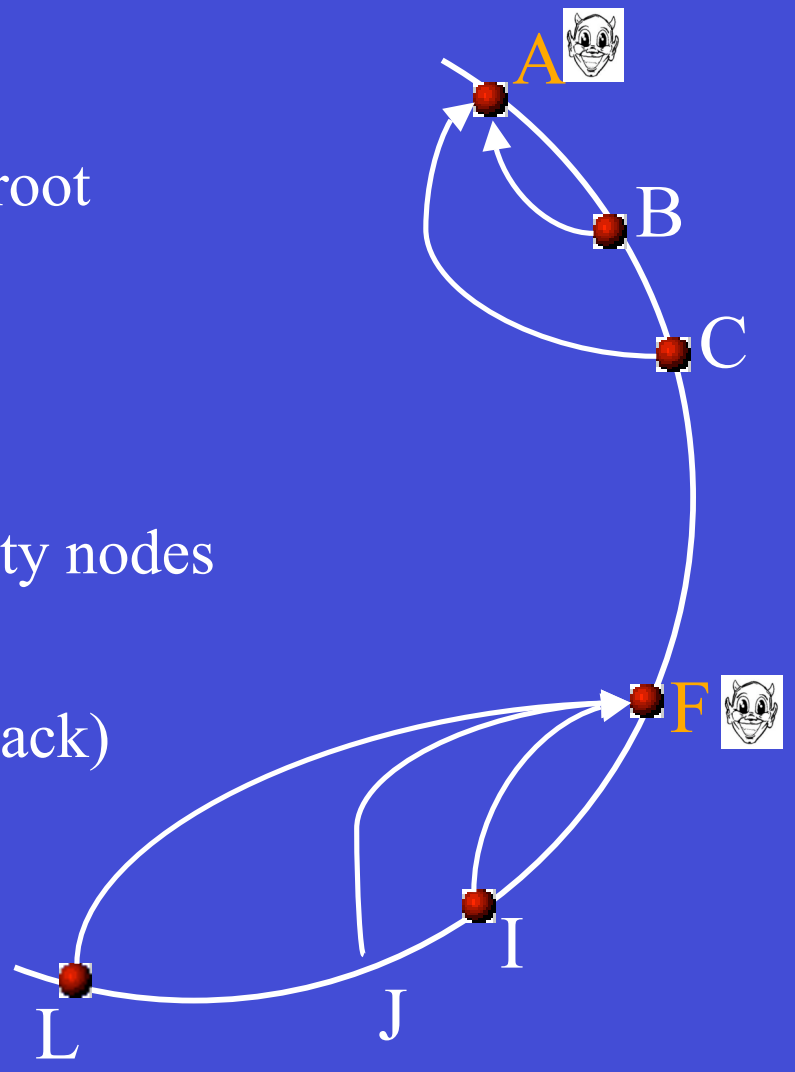
Challenge: Malicious participants

Prevent messages from reaching root

- drop or corrupt
- bias routing tables

Cause objects to be placed on faulty nodes

- choose nodeId values
- use many identities (Sybil attack)
- impersonate root



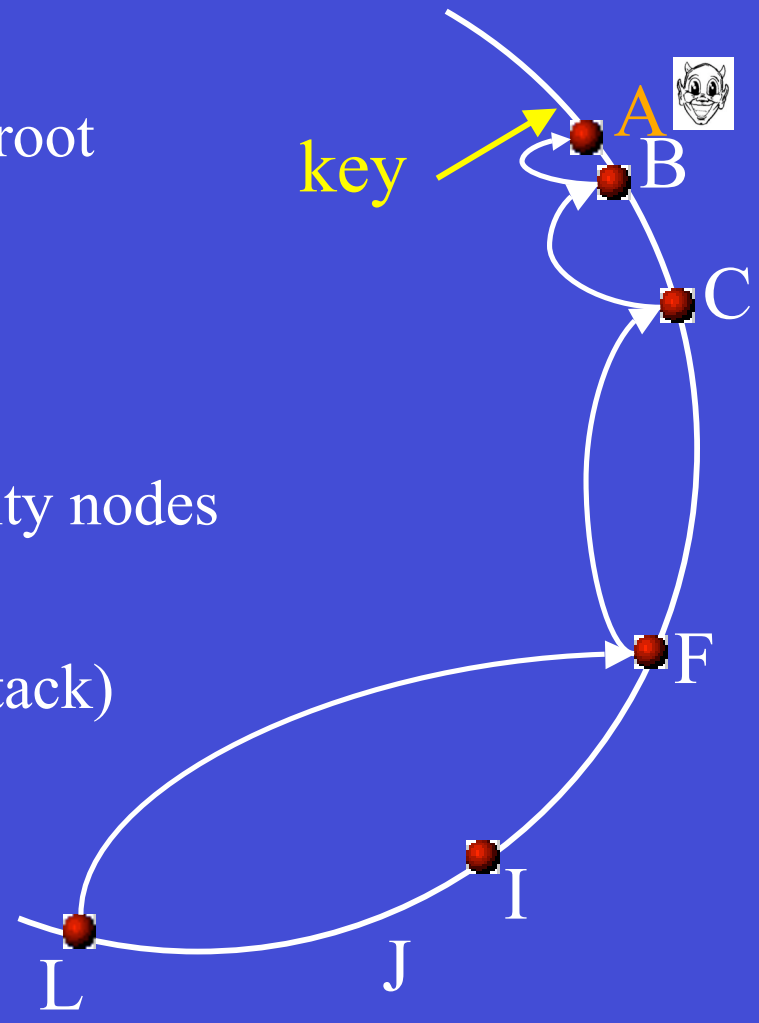
Challenge: Malicious participants

Prevent messages from reaching root

- drop or corrupt
- bias routing tables

Cause objects to be placed on faulty nodes

- choose nodeId values
- use many identities (Sybil attack)
- impersonate root



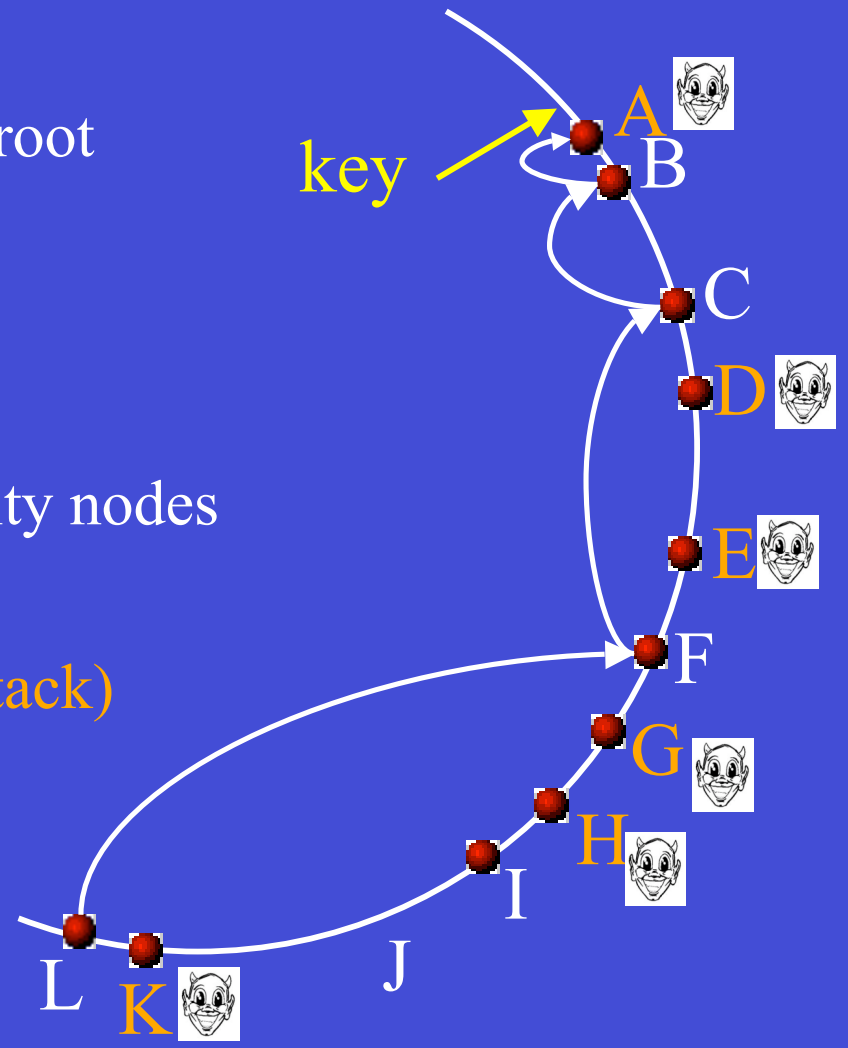
Challenge: Malicious participants

Prevent messages from reaching root

- drop or corrupt
- bias routing tables

Cause objects to be placed on faulty nodes

- choose nodeId values
- use many identities (Sybil attack)
- impersonate root



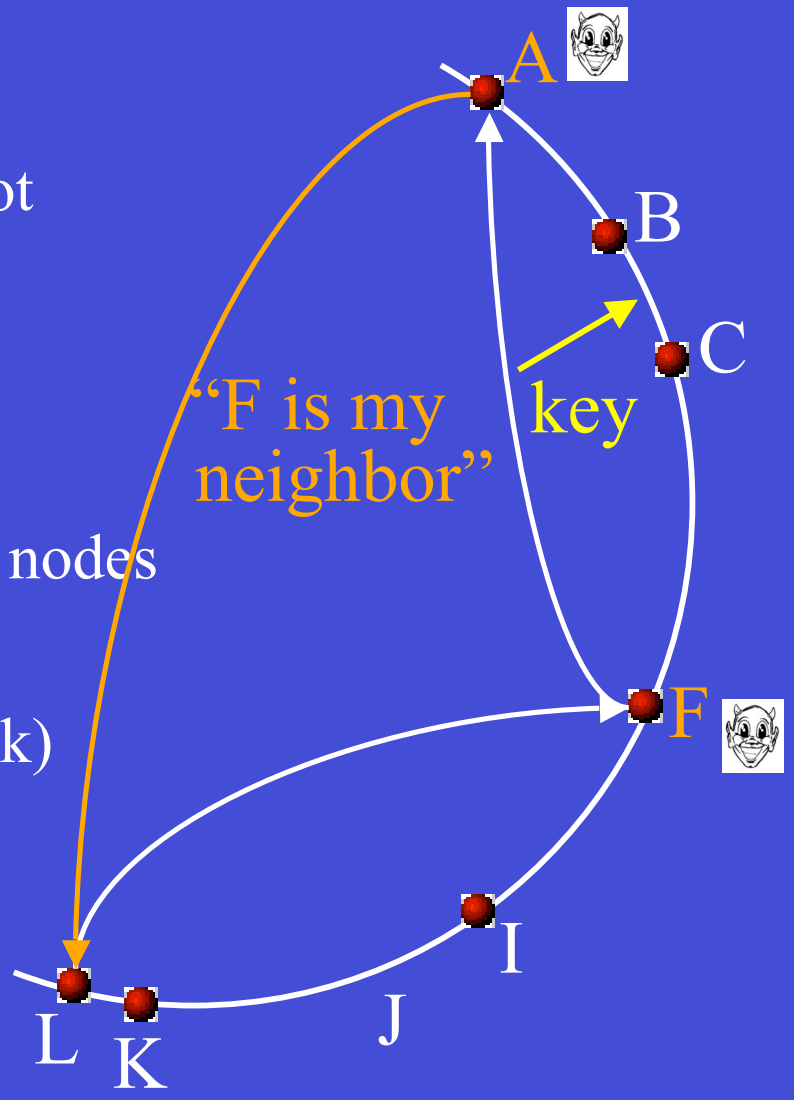
Challenge: Malicious participants

Prevent messages from reaching root

- drop or corrupt
- bias routing tables

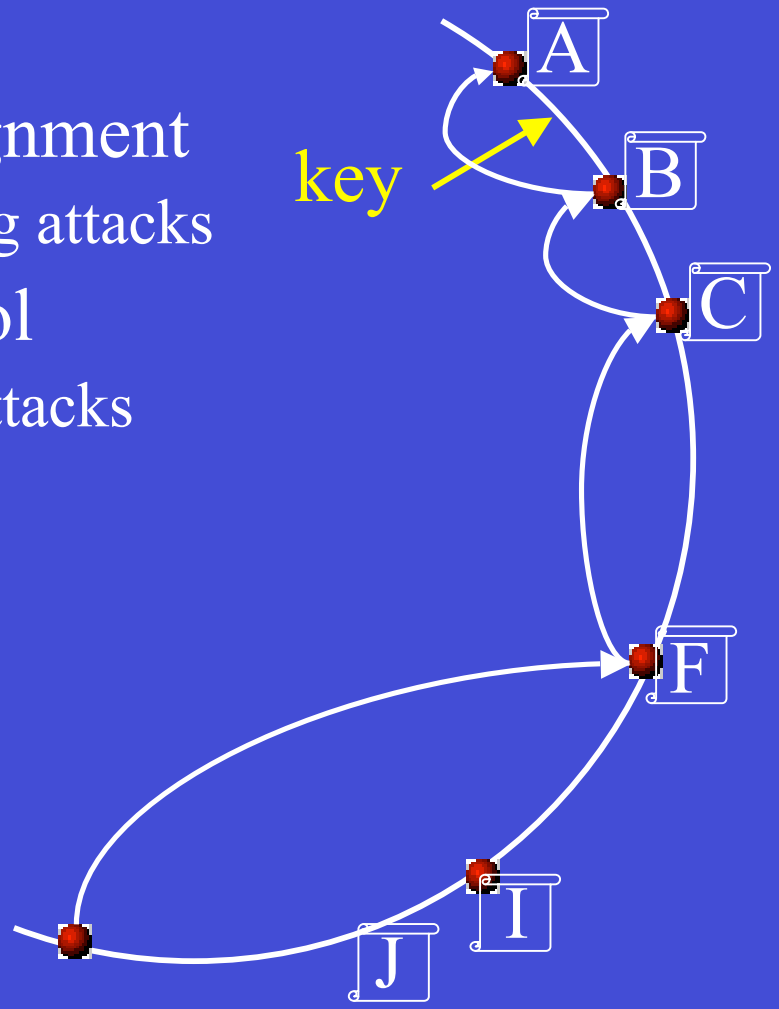
Cause objects to be placed on faulty nodes

- choose nodeId values
- use many identities (Sybil attack)
- impersonate root



Securing routing [OSDI'02]

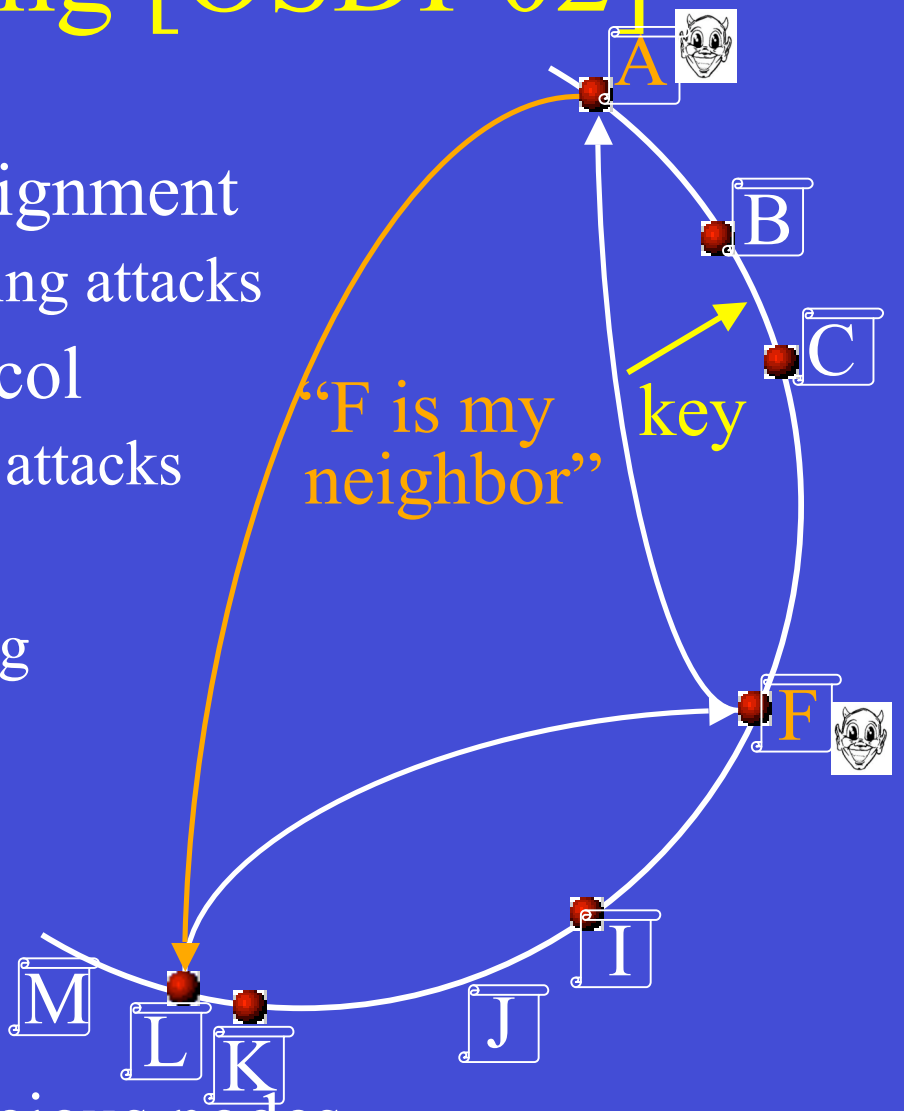
- Secure node identifier assignment
  - thwarts Sybil and id choosing attacks
- Secure membership protocol
  - Prevents routing table bias attacks
- Secure routing primitive
  - Prevents root impersonating



Can tolerate up to 25% malicious nodes

Securing routing [OSDI'02]

- Secure node identifier assignment
  - thwarts Sybil and id choosing attacks
- Secure membership protocol
  - Prevents routing table bias attacks
- Secure routing primitive
  - Prevents root impersonating



Can tolerate up to 25% malicious nodes

Other attacks

- Freeloading
- Censorship
- Denial-of-service
- Etc.

- Many defenses required in addition to secure routing

Ongoing work: PeerReview

Idea: Behavioral audit

- Auditor node *A* checks if it agrees with past actions of an auditee node *B*
- In case of disagreement, *A* broadcasts an *accusation* of *B*
- Interested third nodes verify evidence, take punitive action against auditor or auditee

PeerReview

How does it work?

1. Nodes implement identical *deterministic state machine*
2. Nodes keep a *secure log* of state transitions
3. Msg sender and receiver ensure mutual *commitment* to the communication event
4. Auditors *replay* portions of the auditee's log to ensure behavioral conformance
5. Auditors post *accusations* and cease to cooperate with deviant nodes

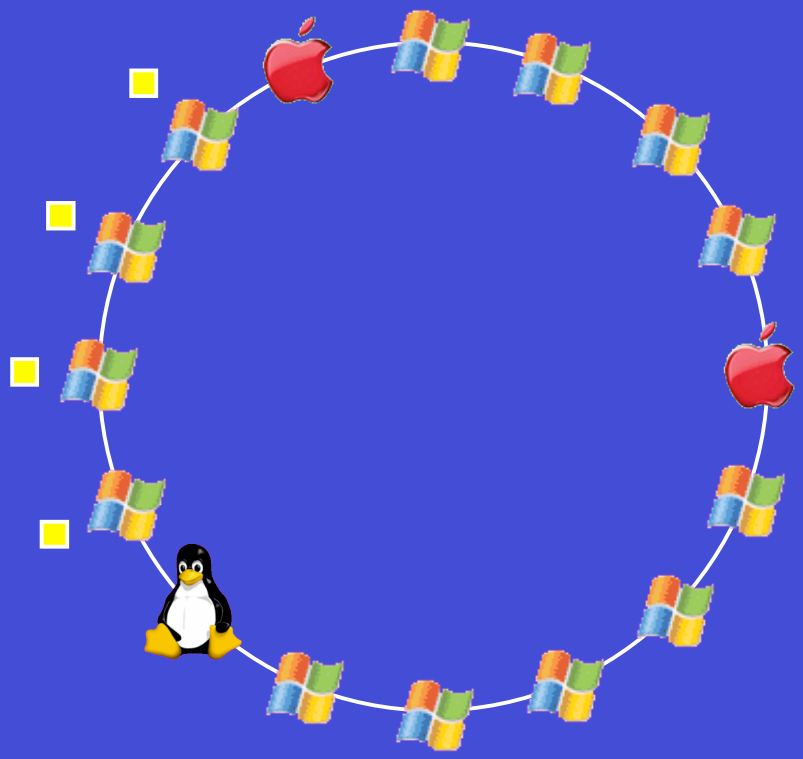
PeerReview: Summary

Eventually detects any attack that is based on changing a node's state machine

Costs:

- Need to isolate non-determinism, log all actions
- Logs compress well, auditing efficient
- Need to sign messages

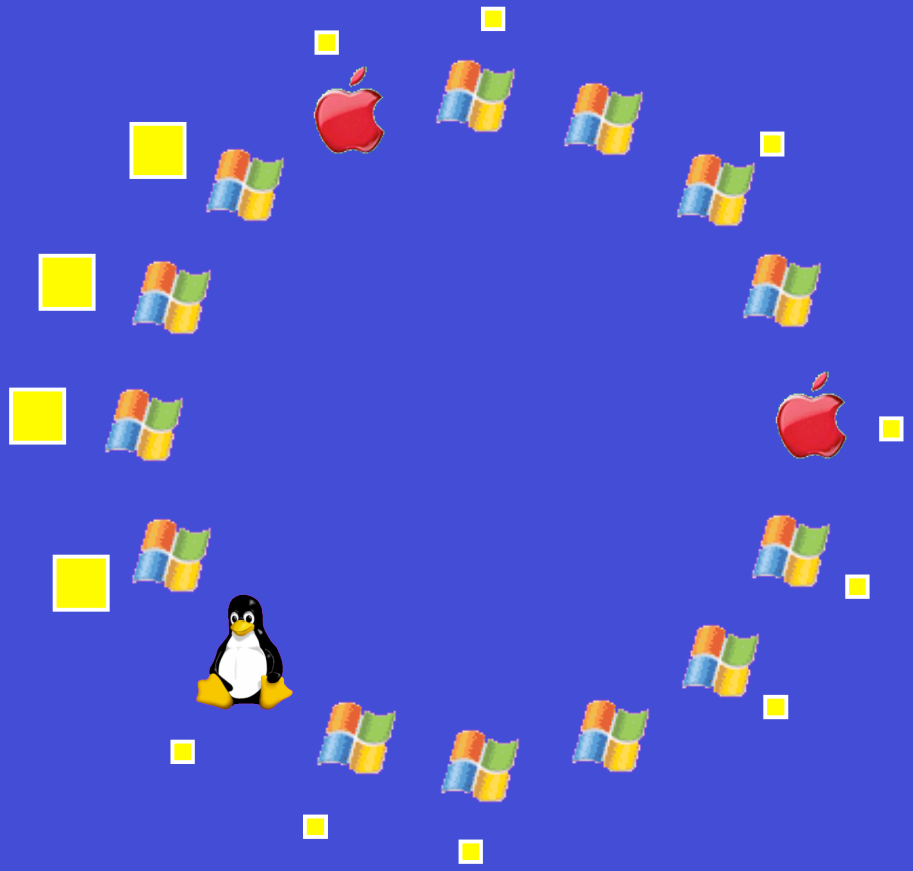
Challenge: correlated node failures



- Common assumption:
  High node diversity
  ⇒ Failure independence
- Worms can cause large-scale, correlated Byzantine failures
- Reactive systems are too slow to prevent data loss

Solution: Glacier (NSDI'05)

Create massive redundancy to ensure that data survives correlated failures with high probability



- Assumption: Magnitude of the failure can be bounded by fraction f_{max}
- Challenges:
  - Minimize storage and bandwidth requirements
  - Withstand attacks, Byzantine failures

Glacier: Summary

- Large-scale correlated failures are a realistic threat to distributed storage systems
- Glacier provides strong durability guarantees with minimal failure assumptions
- Transforms abundant client disk space into durable storage
- Bandwidth cost is low
- Details in [Haeberlen et al., NSDI'05]

Outline

- Decentralized systems
- Challenges
  - Object location
  - Overlay route efficiency
- Key abstractions:
  - Distributed Hash Tables (DHT)
  - Group communication (Scribe)
- More challenges
  - Malicious participants
  - Correlated failures
- Putting it all together: ePOST

Putting it all together

- Is it possible to support decentralized application with high reliability demands?
- Applications that users rely on for their daily work?
- Can an email service be built in this way?
- Tolerating byzantine failures, worm attacks?

Why Email?

Demanding user expectations:

- Security: privacy, authenticity of email messages
- Reliability: email service reliably delivers messages
- Durability: stored email is durable
- Availability: email services is always available

Premise: If email services can be provided using a decentralized overlay, other applications with similar demands are possible

ePOST: General design

- Completely decentralized, cooperative
- No ‘email servers’
- All messages transmitted and stored securely
- Standard mail clients (IMAP/POP)
- Interoperability via SMTP
- Assume nodes can fail arbitrarily
- Users only trust their local node

ePOST: Status

ePOST is deployed and in production use

- Members of group at Rice University and MPI-SWS have relied on ePOST for well over a year
- Project has driven much of the work I discussed
- Trace collection, experience with operation
- Goal: understand admin issues, reduce to hardware replacement
- http://www.epostmail.org

Other current work

- FeedTree: Cooperative dissemination of web micronews (http://www.feedtree.net)
- Cooperative Internet TV
- Decentralized application support for mobile ad hoc networks

Some related work

- Structured overlays: Chord, CAN, Kademlia, SkipNet, etc.
- Theory: Plaxton Trees, Small World Networks
- Reputation systems
- Trusted computing
- Shared storage [CFS, OceanStore, Ivy]
- Content distribution [BitTorrent, CoolStreaming, ESM]
- Querying and indexing [PIER, Astrolabe, SDIMS]
- Collaboration applications [UsenetDHT,Overseer]
- Backup store [HiveNet, Pastiche]
- Multicast/Anycast/Mobility [i3]
- Overlay QoS [OverQos]
- Naming systems [INS, SFR]

Credits: Group members

- Marcel Dischinger
- Andreas Haeberlen
- Jeff Hoye
- Petr Kouznetsov
- Alan Mislove
- Animesh Nandi
- Ansley Post
- Dan Sandler
- Atul Singh
- Jim Stewart

Credits: Collaborators

- Hui Zhang, *CMU*
- Anne-Marie Kermarrec, *INRIA Rennes*
- Petros Maniatis, Timothy Roscoe, *Intel Research Berkeley*
- Krishna Gummadi, *MPI for Software Systems*
- Miguel Castro, Antony Rowstron, *Microsoft Research, Cambridge*
- Frans Kaashoek, Robert Morris, *MIT*
- Y. Charlie Hu, *Purdue University*
- Dave Johnson, Eugene Ng, Rolf Riedi, Dan Wallach, *Rice University*
- John Kubiatowicz, Ion Stoica, *UC Berkeley*

Credits: Funding

- Max Planck Society
- National Science Foundation
- Texas Advanced Technology Program
- Intel Research
- Microsoft Research

MPI for Software Systems

- Founded in November 2004
- Two locations in Kaiserslautern und Saarbrücken
- Budget: ~ €10Mio/year (without outside funding)
- 5 Direktors, 12 tenure-track positions, ~80 doctoral/post-doc positions, plus administrative and technical support staff
- ***Mission:*** Basic research at an internationally leading level in software systems

Conclusions

- Decentralization is a powerful tool for the construction of scalable, resilient, cooperative, ad hoc services
- We're beginning to understand the pieces
- Many challenges remain
  - Trust, Security, Digital rights management
  - Consistency and Partitions
  - Administering decentralized systems

For more information

- http://www.mpi-sws.org