

# **Issues in Peer-to-Peer Computing**

**Barbara Liskov**

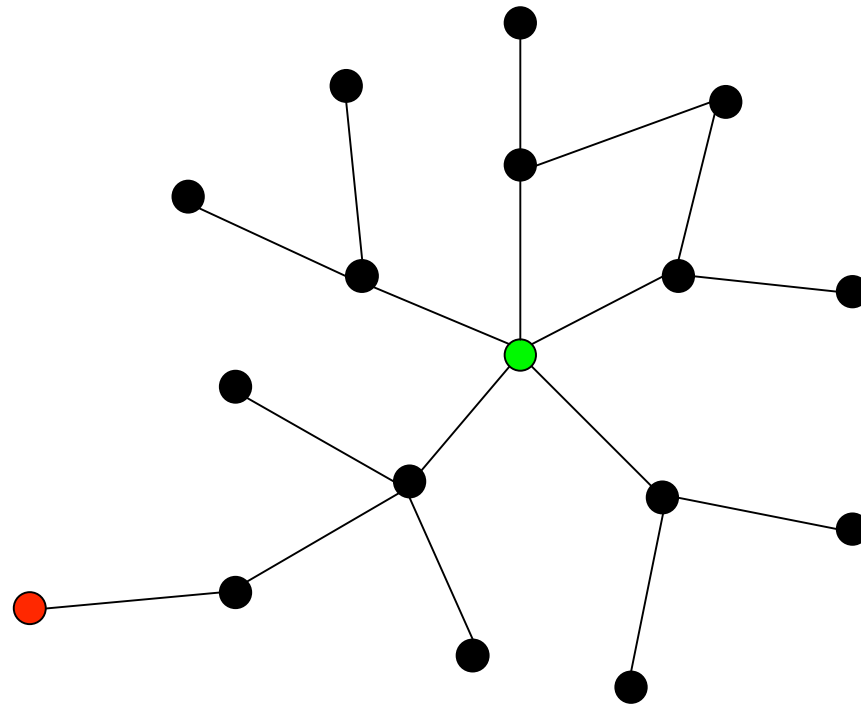
**MIT Laboratory for Computer Science**

**June 11, 2003**

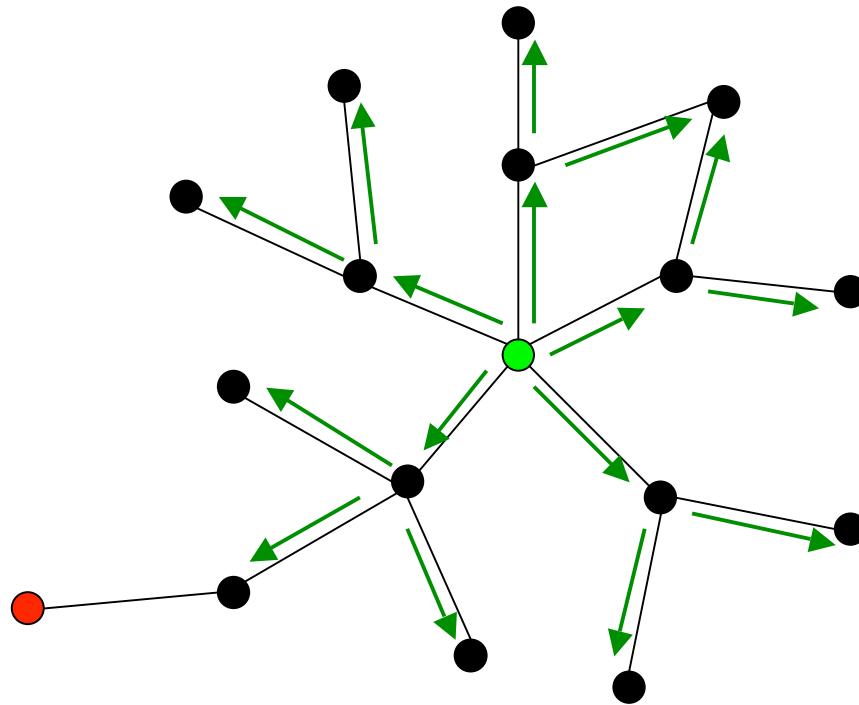
# P2P Systems

- **Started with file sharing apps**
  - **Gnutella, Napster, Kazaa, ...**
- **For sharing music and videos**
  
- **Every node is a “peer”**
  - **Can get data from others**
  - **Can provide data to others**

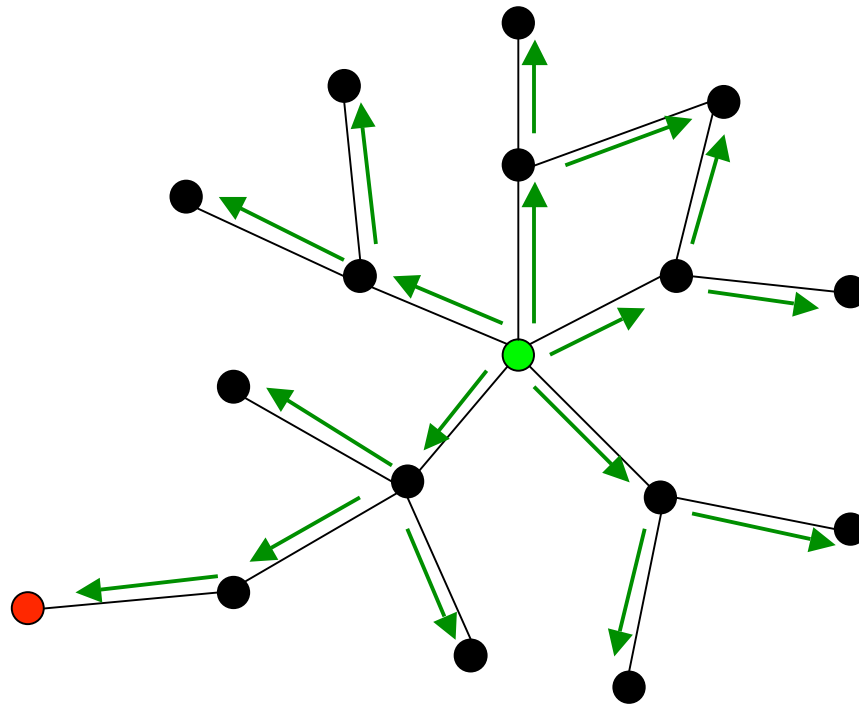
# How it works



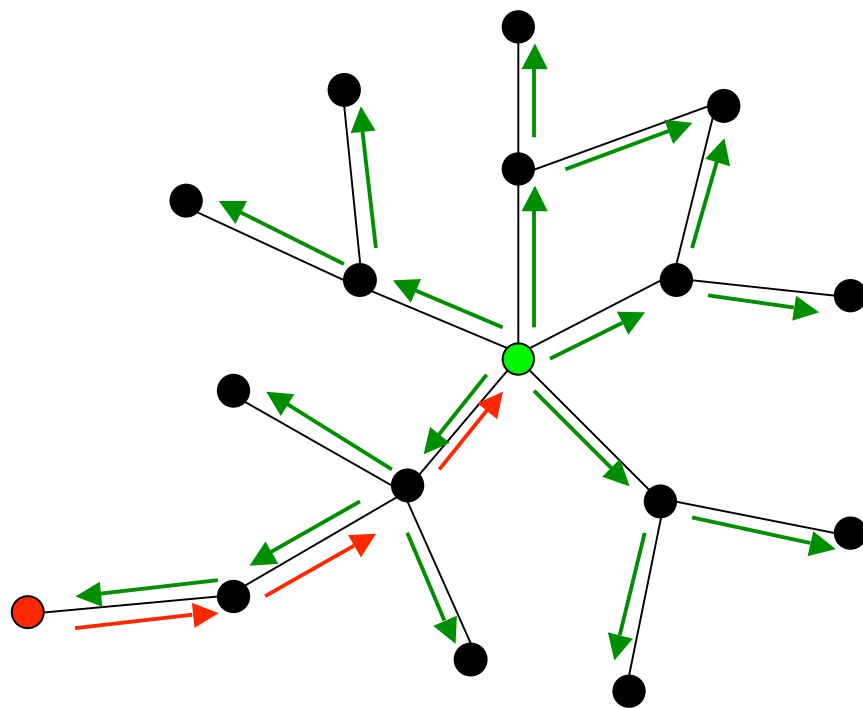
# How it works



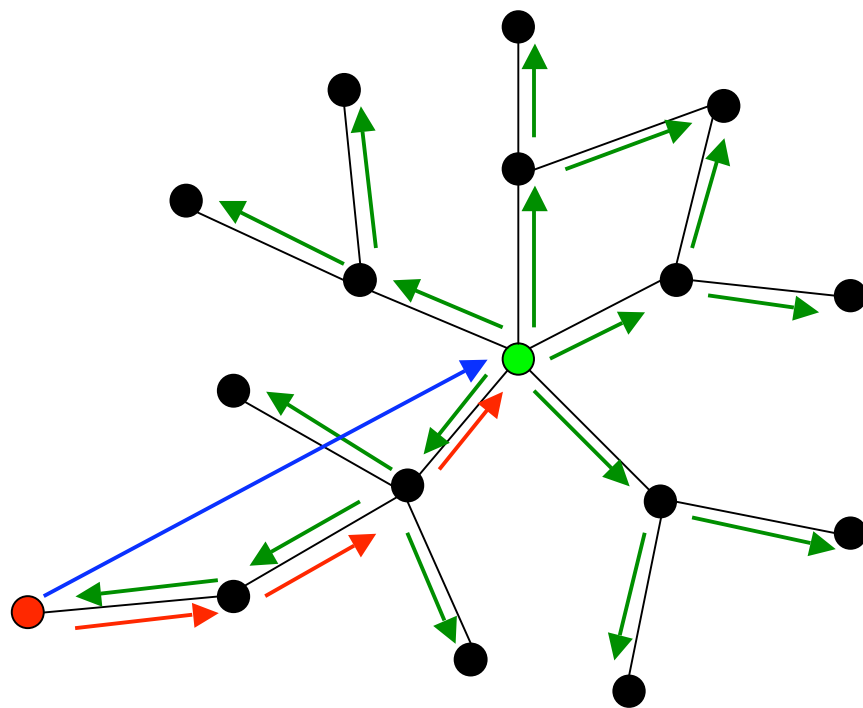
# How it works



# How it works



# How it works



# Virtues

- **Scalability**
  - **Size is potentially very large**
- **Ease of reconfiguration**
  - **System easily adjusts to joins and leaves**

# Limitations

- **Real state can be lost**
  - **P2P storage is just a publishing mechanism**
- **Search costs can be high**
  - **OK if shared files are large**
- **No retrieval guarantees**
  
- **Fine for intended use**

# Structured P2P

- **Let's retain the advantages**
- **But overcome the limitations**
  
- **Then we can expand the application domain**
- **And maybe application code will be simpler**

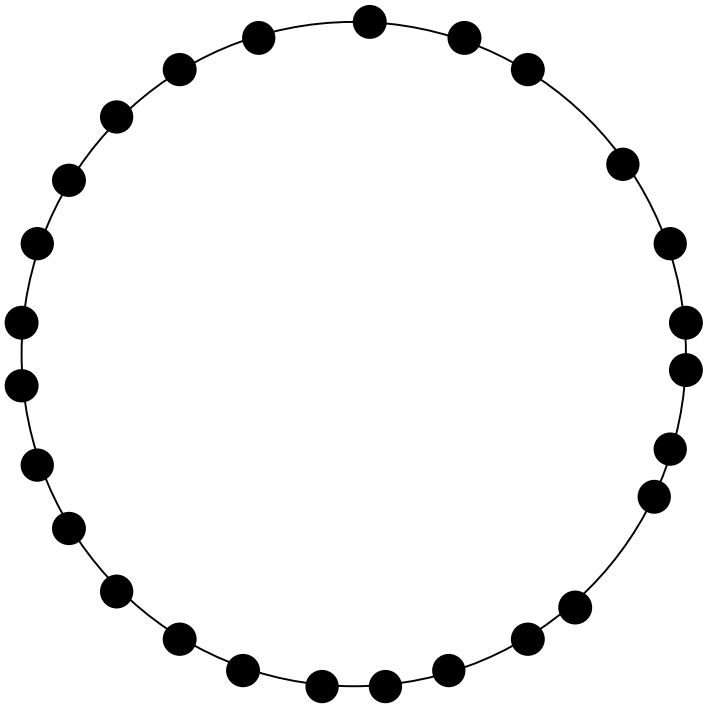
# Talk Outline

- **Existing (structured) p2p systems**
- **Faster routing**
- **BFT storage**
- **Conclusions**

# Structured P2P Systems

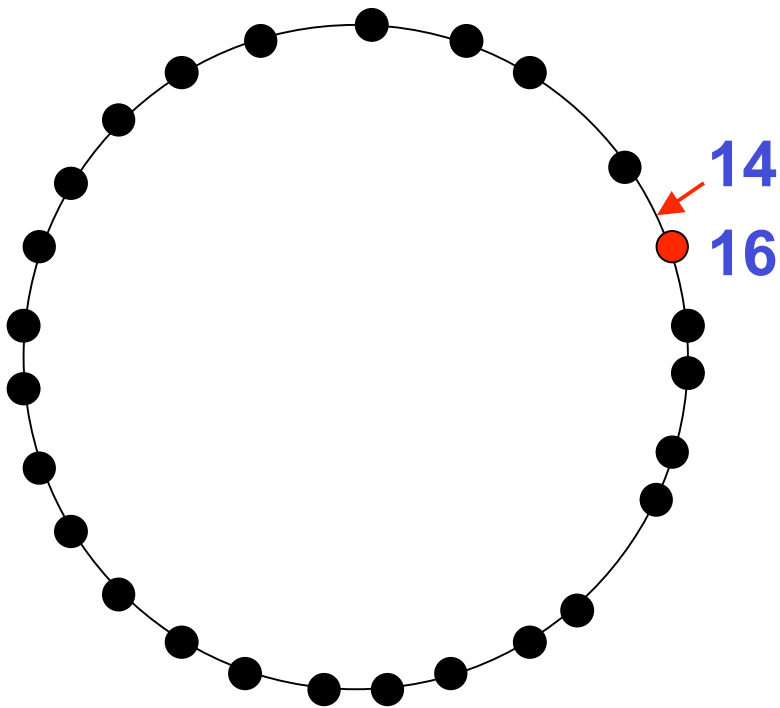
- **A number of contenders**
  - **Chord [SIGCOMM'01], Pastry [Middleware'01], Tapestry [SPAA'02], Kademlia [IPTPS'02]**
- **All are similar (but different!)**
- **Talk is based on Chord**

# P2P Ring



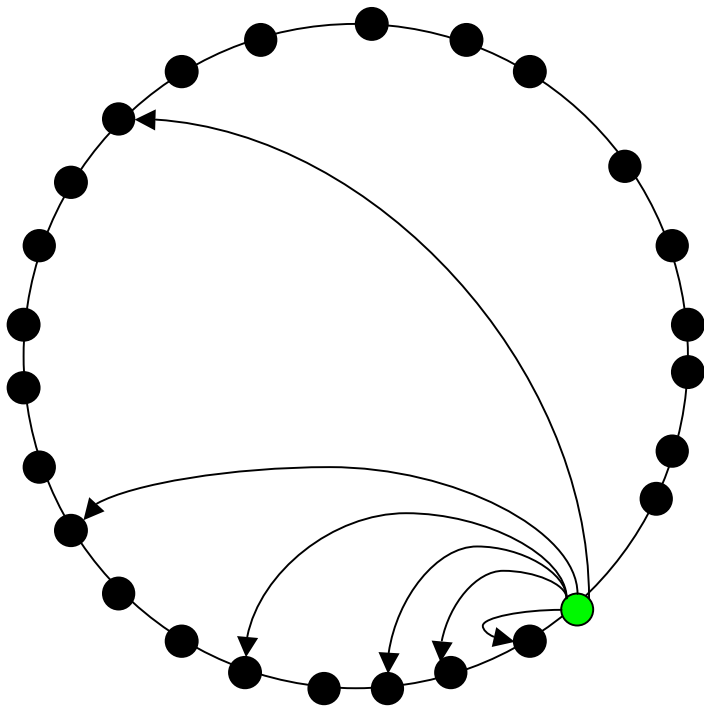
- **Nodes are arranged in a ring based on id**
- **Ids are assigned randomly**
- **Very large id space**

# P2P Storage



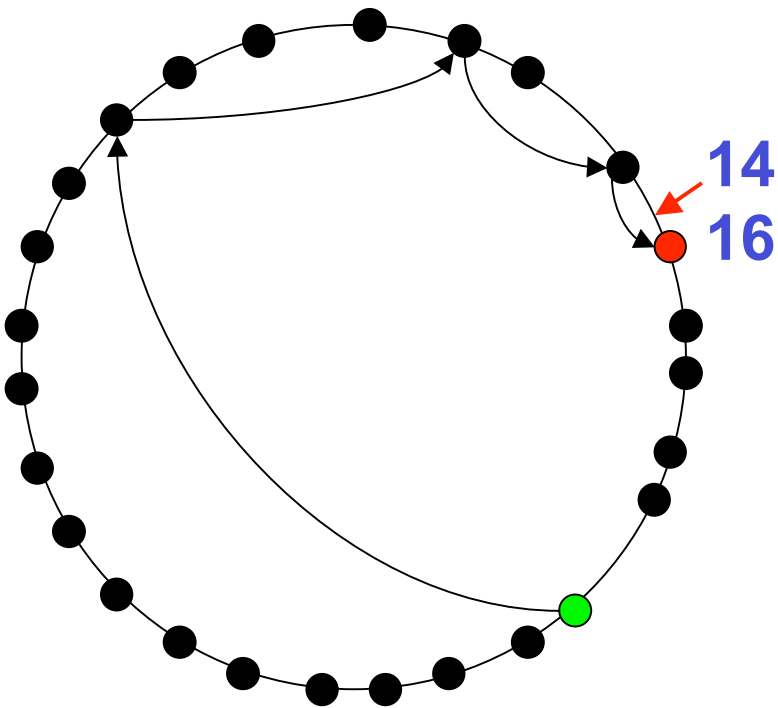
- Data items also have ids
- Every node is responsible for a subset of the data
- Assignment based on  $f(id)$ , e.g., id's successor
- Supports reconfiguration

# Routing State



- **Nodes have limited info**
- **Small routing state, changes slowly**
- **Each node knows its  $k$  successors**
- **Each node knows  $\log(N)$  fingers**

# Routing



- **Route via binary search**
- **Use fingers first**
- **Then successors**
  
- **Cost is order  $\log n$**

# Distributed Hash Tables

- A large shared memory implemented by p2p nodes
- Addresses are **logical**, not physical
- Implies applications can select them as desired
  
- Typically a hash of other information
- System looks them up

# Room for Improvement

- **Faster routing**
- **Reliable storage**

# Faster Routing

- **Routing is still costly (order  $\log n$ )**
- **Limits applications**
  
- **Small routing state is cheap to maintain**
- **BUT is big state expensive?**

# Our Thesis

**It is feasible to :**

- **Keep full routing state on every node**
- **Route in one hop**
- **Achieve high success rate**

**Thus we can :**

- **Enable a large class of applications for peer-to-peer systems**

# One-hop Routing

- **Goal: almost always get to node in one hop, e.g., 99% success rate**
- **Implies routing table is almost accurate**
- **Implies fast propagation of configuration events (adds and deletes)**
  
- **Joint work with Anjali Gupta and Rodrigo Rodrigues**

# Configuration Change Rate

- **Gnutella study [MMCN'02]**
  - **Average node session time is 2.9 hours**
- **Implies**
  - **20 events/sec in 100,000 node system**
  - **Assuming rate of change proportional to system size**
- **Worst-case assumption**

# Key Issue

- **Bandwidth consumption**
  - In sum
  - At individual nodes
- **Minimum bandwidth required**
  - Every node must learn of every event
  - E.g., 20 events/second in 100K system
  - Issue: what overhead

# Simple Approach 1

- **Just communicate with successors**
  - Each node probes its successor, e.g., once a second
  - Piggyback new info
  - Implies 20 events/message (in 100K system)
- **Minimal bandwidth use**
  - Don't send redundant information!
- **Much too slow**

# Simple Approach 2

- **Node discovering event tells everyone**
- **Huge load at that node**
- **Load on network too**
  - **E.g., 2M messages/second in 100K system**
- **Roughly doubles the bandwidth use**
  - **Because of header overhead**

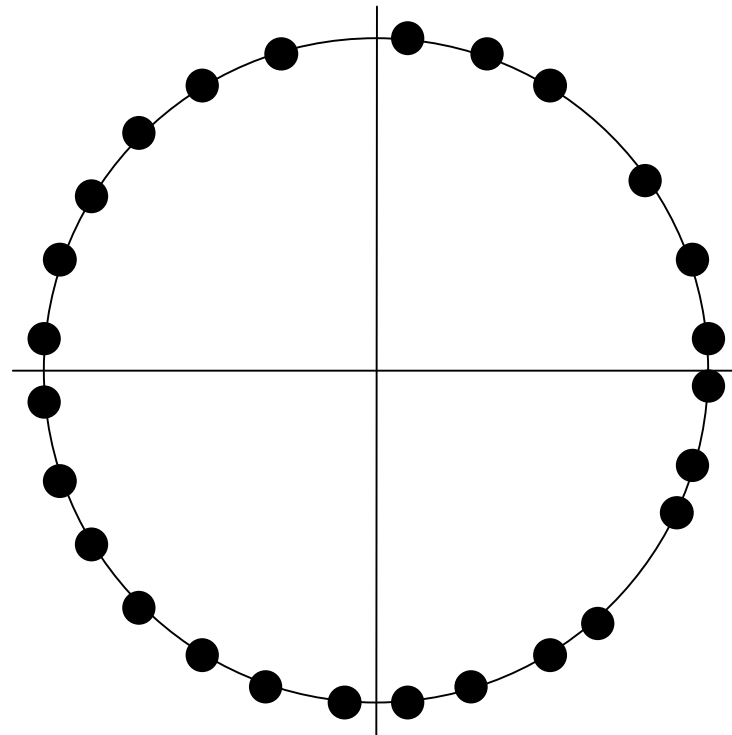
# Our Approach

- **Hierarchical**
- **A two level hierarchy**
  - Ring is divided into slices
  - Each slice is divided into units
- **Goal**
  - **Get close to minimum bandwidth**
  - **But also fast propagation**

# Time Constraint

- **Goal of 99% lookup success rate implies routing table must be 99% accurate**
- **The answer is 50 seconds!**
  - **given assumption about number of changes**
- **A worst-case analysis**

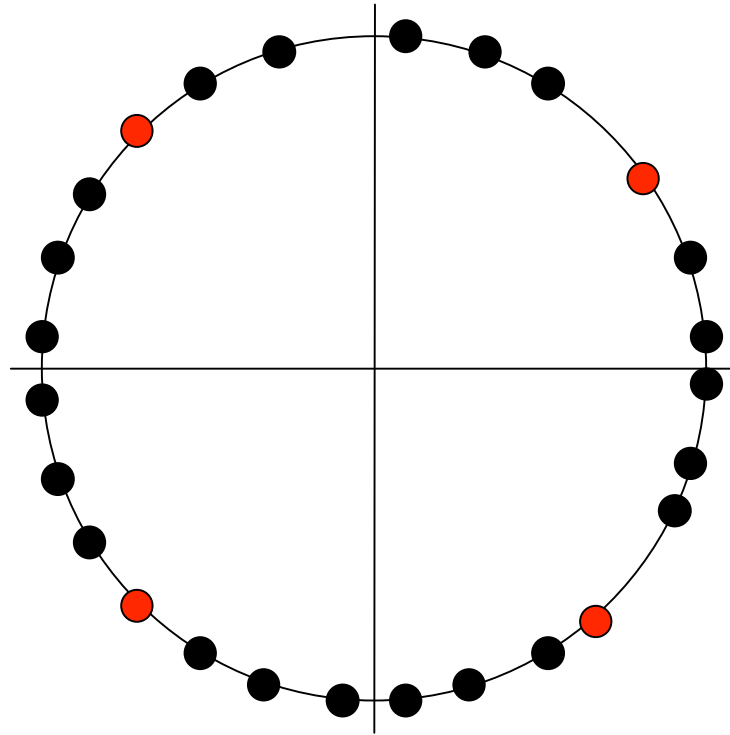
# Slices



**Ring is divided statically into  $k$  slices**

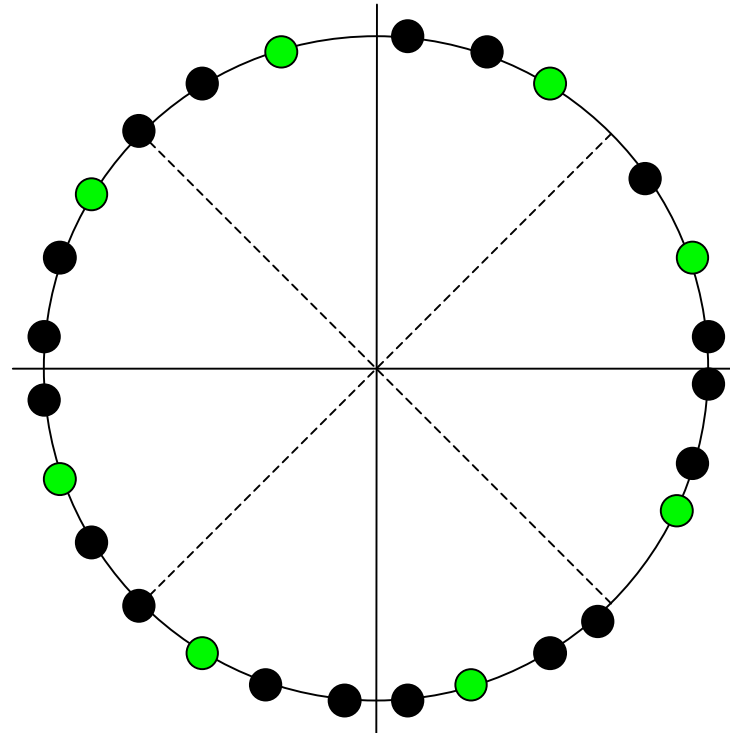
**$K \sim 500$  slices in a 100K system**

# Slice leader



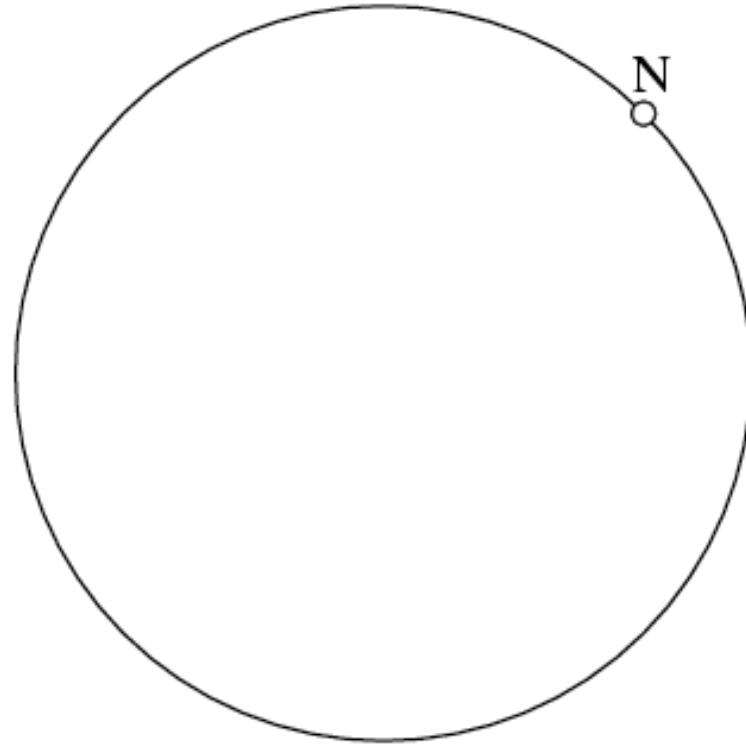
**Successor of midpoint of slice is slice leader**

# Unit leader



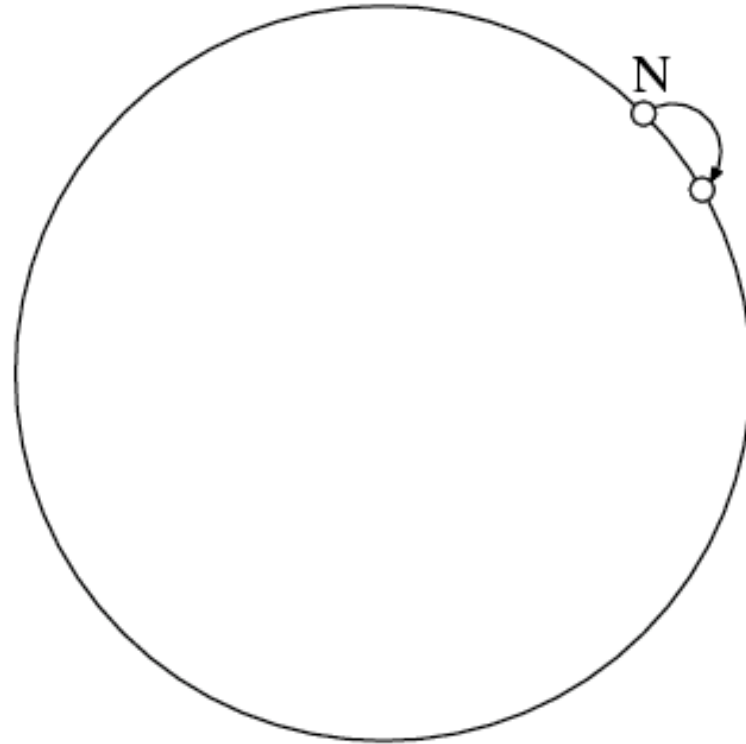
**Successor of midpoint of unit is unit leader**

# Propagating Information - I



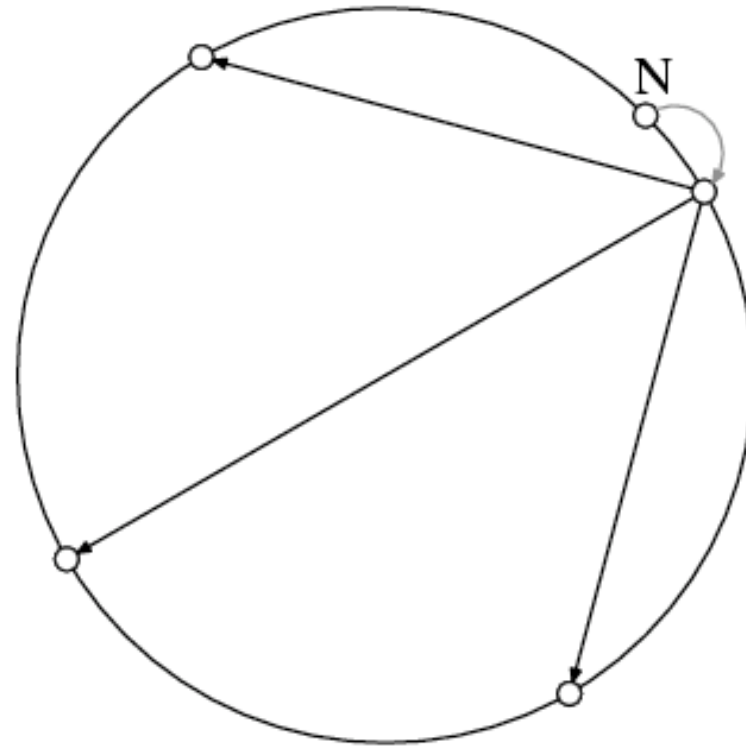
**Step 1: Event detected by node N**

# Propagating Information - II



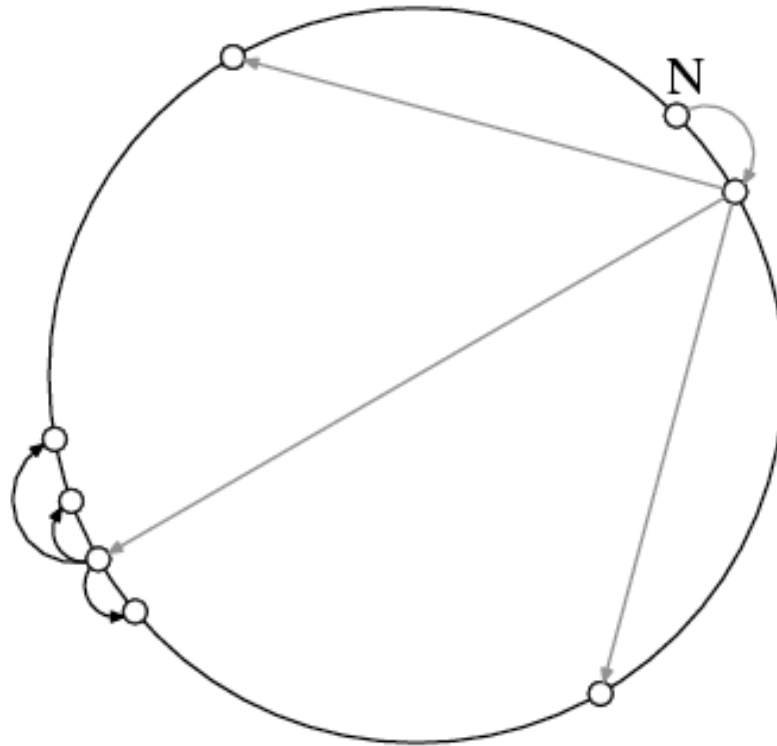
**Step 2: N notifies its slice leader**

# Propagating Information – III



**Step 3: N's slice leader collects events for some time, then notifies other slice leaders**

# Propagating Information - IV



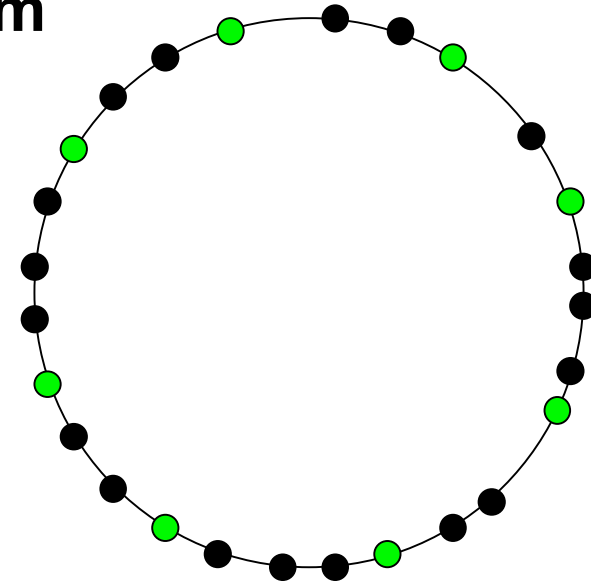
**Step 4: Slice leaders notify their unit leaders periodically; then use base communication**

# Speed of Propagation

- If (expected) unit size is 40, the time taken to reach farthest nodes is:
  - Node to Slice leader **4 s**
  - Slice leader to other slice leaders **25 s**
  - Slice leader to unit leaders **1 s**
  - Unit leader to edge of unit **20 s**
- Total time **50 seconds**

# Discussion

- **BW at slice leaders**
  - About 35 kbps in 1M system
  - **Solution: supernodes**



- **Ultimate limitations**
  - E.g., > 10M nodes?
  - Working on two-hop scheme

# Reliable Storage

- **Requires replication**
- **But what failure model?**
- **Failstop failures**
  - **Nodes fail by stopping, e.g., node crashes**
- **Byzantine failures**
  - **Arbitrary failures, e.g., nodes can lie**
  - **Happen because of malicious attacks and software errors**

# Rosebud

- **A BFT storage system**
- **Based on p2p**
  - **scalability, ease of reconfiguration, and load balancing**
- **Good performance (ideally, 1 phase)**
- **Handle reconfiguration**
- **Joint work with Rodrigo Rodrigues**

# BFT Replication

- **Requires  $3f+1$  replicas to survive  $f$  faults**
- **Works for fixed replica groups**

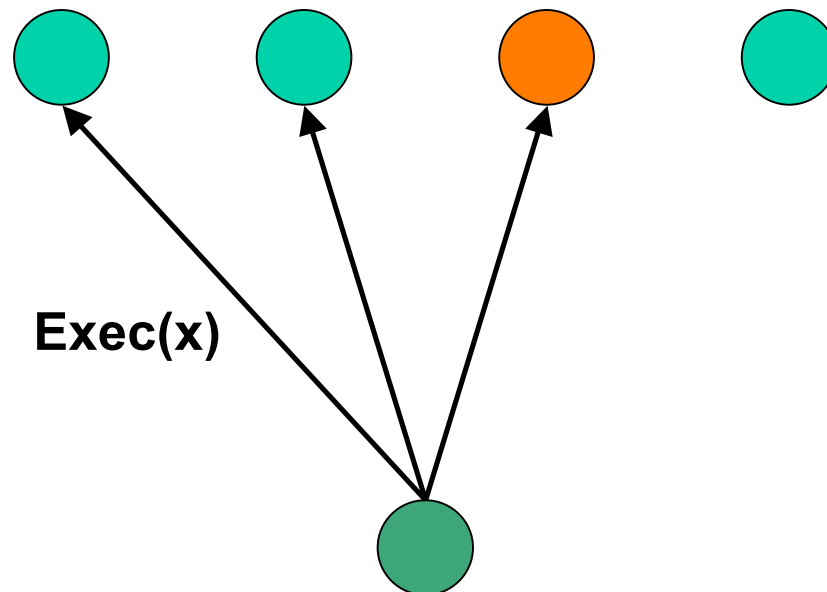
# Performing Operations

- Operation completes when  $2f+1$  valid responses



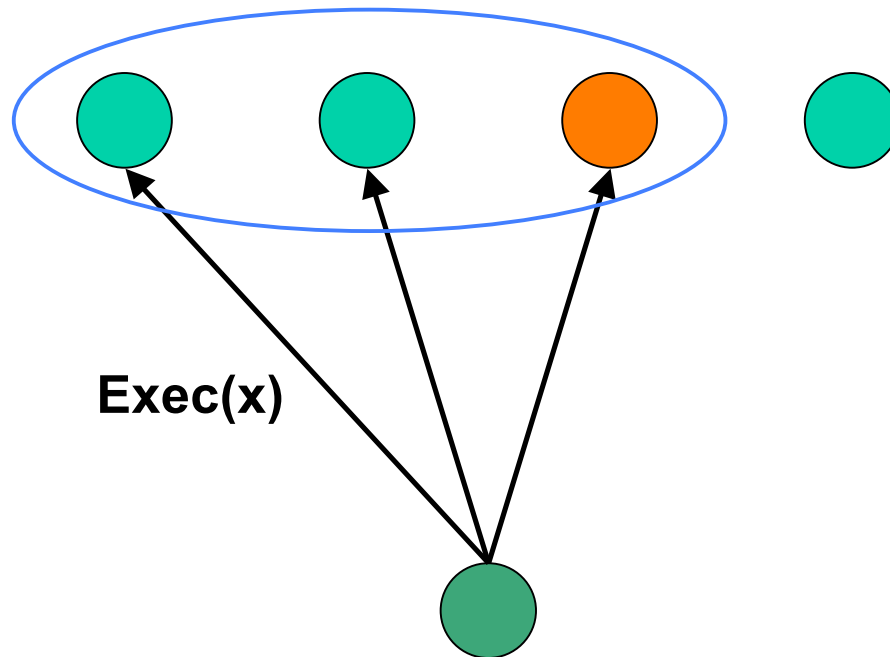
# Performing Operations

- Operation completes when  $2f+1$  valid responses



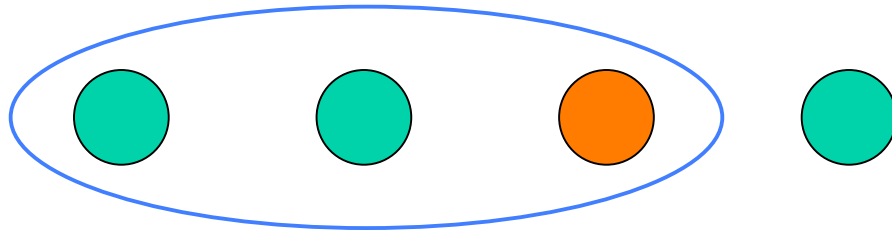
# Performing Operations

- Operation completes when  $2f+1$  valid responses



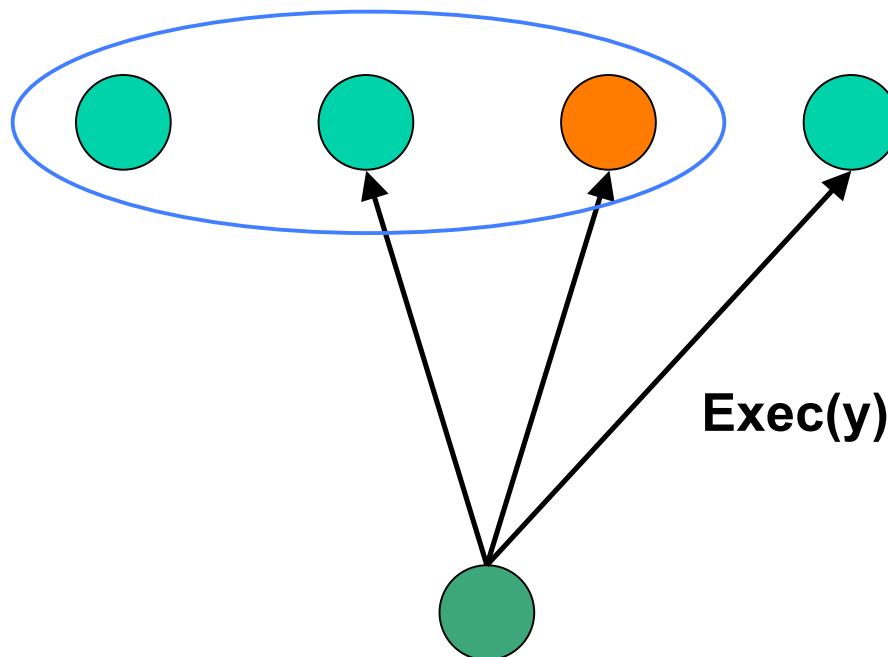
# Next operation

- **Wait for  $2f+1$  legal, matching responses**



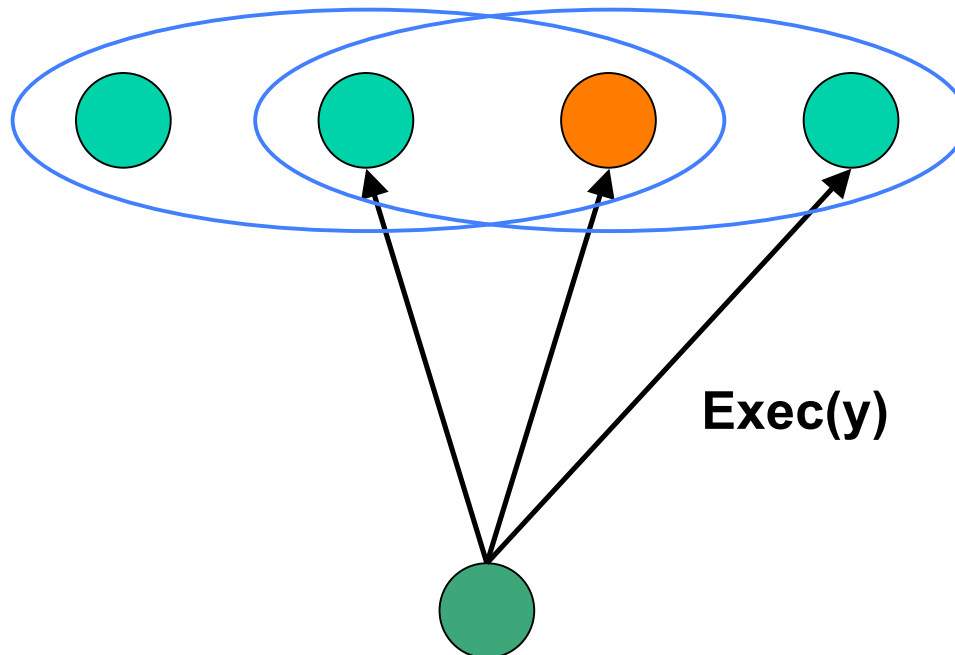
# Next operation

- **Wait for  $2f+1$  legal, matching responses**



# Next operation

- Wait for  $2f+1$  legal, matching responses



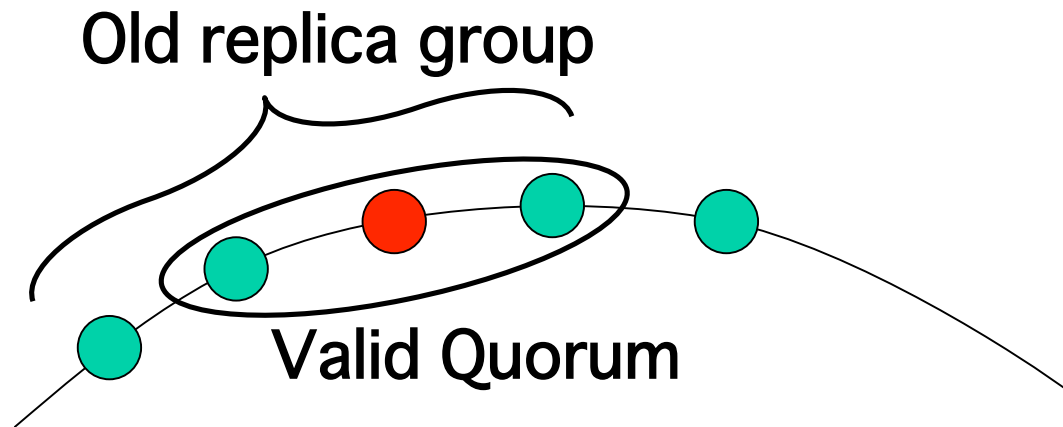
# BFT protocols

- **State-machine replication [OSDI'99]**
  - arbitrary operations, e.g., insert and lookup a public key
  - 3-phase protocols
- **Read/write replication [STC'97]**
  - Only supports read and write of data item
  - Client implements operations, so less safe
  - 1-phase protocols

# Choosing replicas

- **$3f+1$  successors per data item**
- **Replicas must fail independently**
  - **Ids selected randomly**
  - **Admission control (Sybil attack)**
- **Therefore, not random clients**
  - **E.g., PCs on employees' desks**

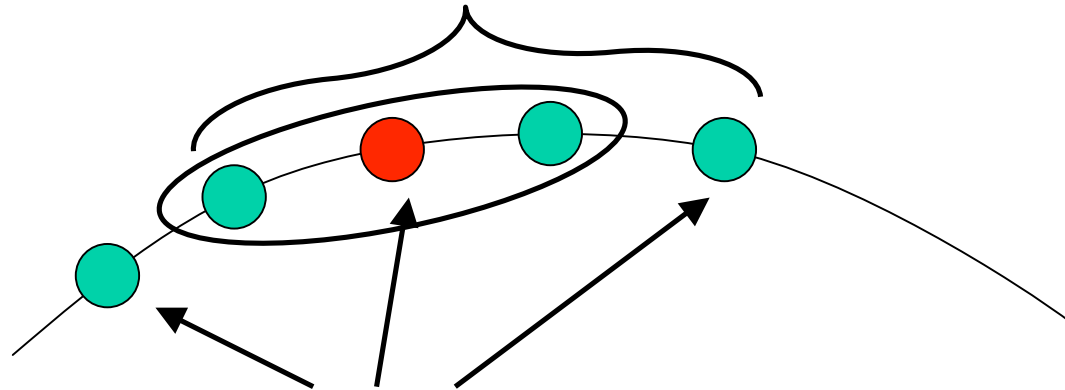
# Configuration Change Problem



- **Operation 1 executes prior to change**

# Configuration Change Problem

New Replica Group



Hybrid Quorum Does Not Satisfy Intersection Properties

- **Operation 2 executes incorrectly**

# **Our Solution**

- **Epochs**
- **The configuration service**

# Epochs: Periods of Stability

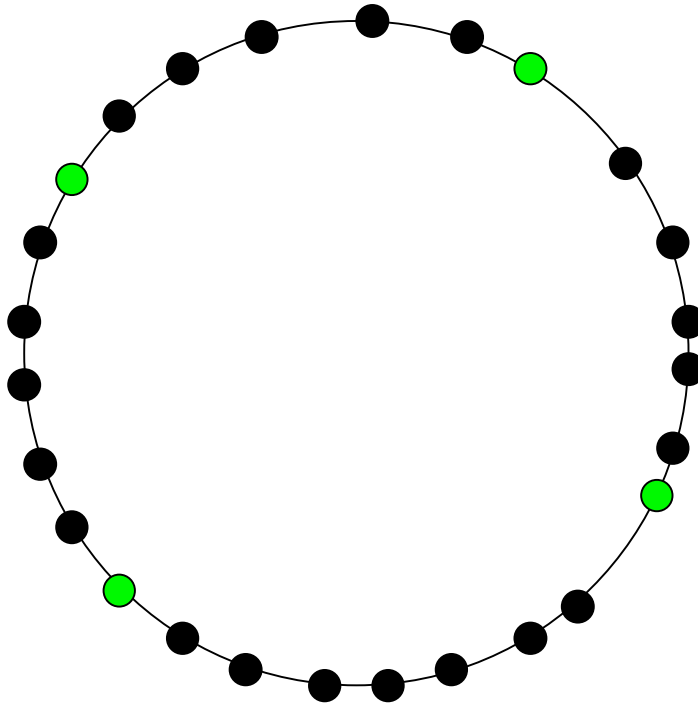
- **System moves through sequence of epochs**
  - **Configuration changes only at epoch boundary**
- **Epoch duration**
  - **Can be long, e.g., several hours**
  - **BUT, small enough that no more than  $f$  failures in any group of successors**

# Computing the Next Configuration

- Nodes need to agree on configuration
- Requires Byzantine agreement
- Too hard to do this with  $N$  nodes!
- Instead configuration service does this
  - CS runs at  $3F+1$  nodes

# System Organization

- **Configuration service is superimposed on nodes**



# Configuration Functions

- **Admission control**
  - Requires certificate signed by authorized party
  - New node has a public/private key pair
- **Probing**
- **Revocation**
- **Computing the next configuration**
- **Uses BFT state-machine protocols**

# Dissemination

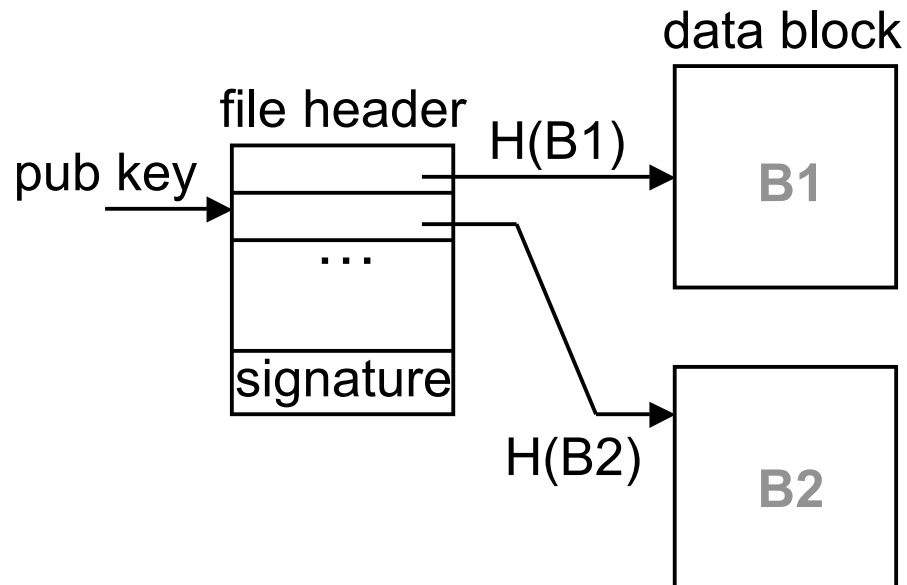
- **Epoch description contains**
  - Epoch number**
  - Start and end time**
  - Adds and Deletes**
  - Signature**
- **Propagated to all nodes**
  - **Over a tree plus gossip**
- **CS moves in each configuration**

# Storage

- Content-hashed objects
  - Id is hash of content
  - immutable
- Public-key objects
  - Mutable
  - Id is hash of public-key of allowed writer
  - Contains a version number
- In either case, self-verifying

# Example

- **File system (CFS) [SOSP'01]**
- **Data blocks are content-hashed**
- **File header is public-key, lists ids of blocks**



# Semantics

- **Public-key objects are atomic**
  - A serial order on reads and writes
  - If  $r1$  sees  $v1$ ,  $r2$  sees at least  $v1$
- **Content-hashed objects are weaker**
  - Matches their use
- **Implementation uses Byzantine quorums**

# Content-hashed objects

- **Write goes to  $3f+1$  successors**
  - **Writer waits for  $2f+1$  valid replies**
- **Reading from one replica is sufficient**
- **Note: one phase**

# Public-key Objects

- **Write goes to  $3f+1$  replicas**
  - **Writer waits for  $2f+1$  valid replies**
- **Read goes to  $3f+1$  replicas**
  - **Reader waits for  $2f+1$  valid replies**
  - **If  $2f+1$  are identical (same  $v\#$ , same value), done**
  - **Else write back the highest  $v\#$**

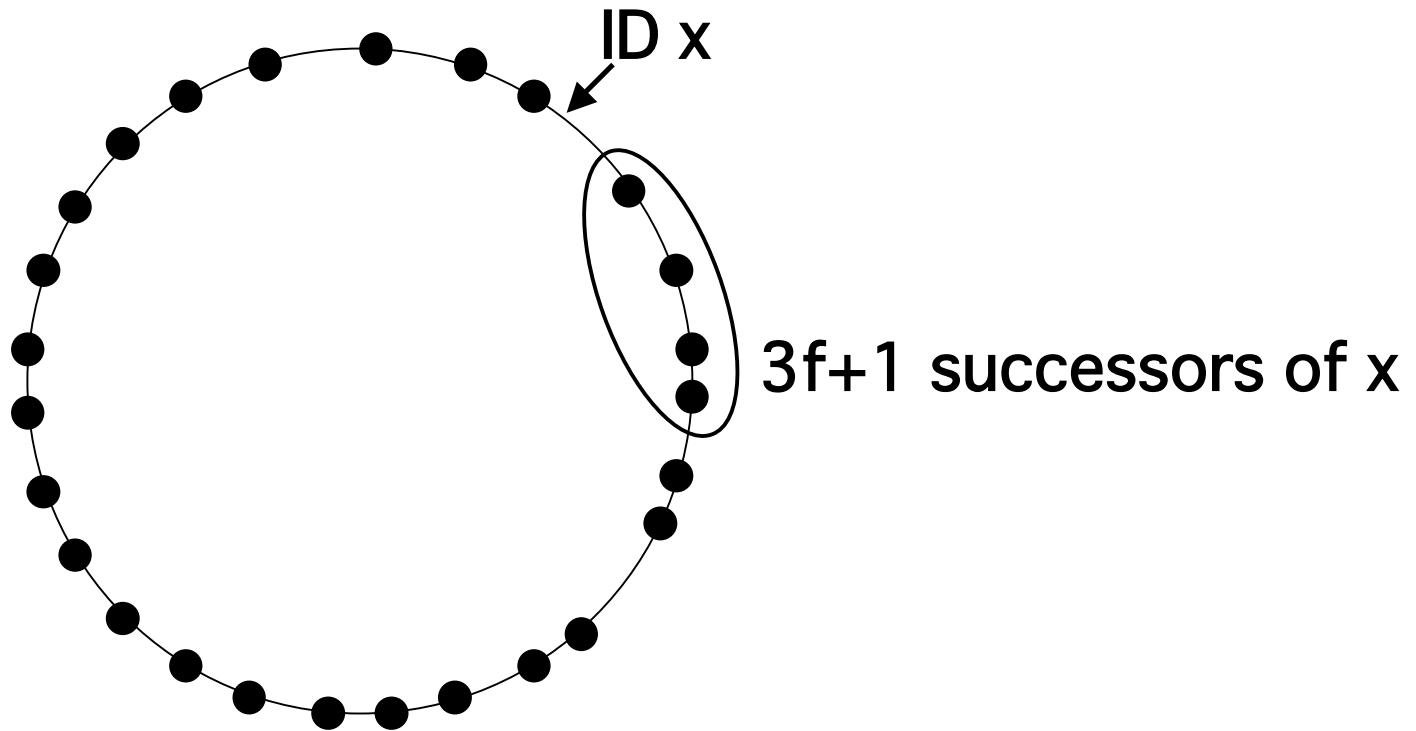
# Public-key objects

- **Normally one phase**
- **Read finishes write**
- **Handles**
  - **Concurrent writes**
  - **Faulty clients including malicious ones**
  - **E.g., client writes different values to different replicas**

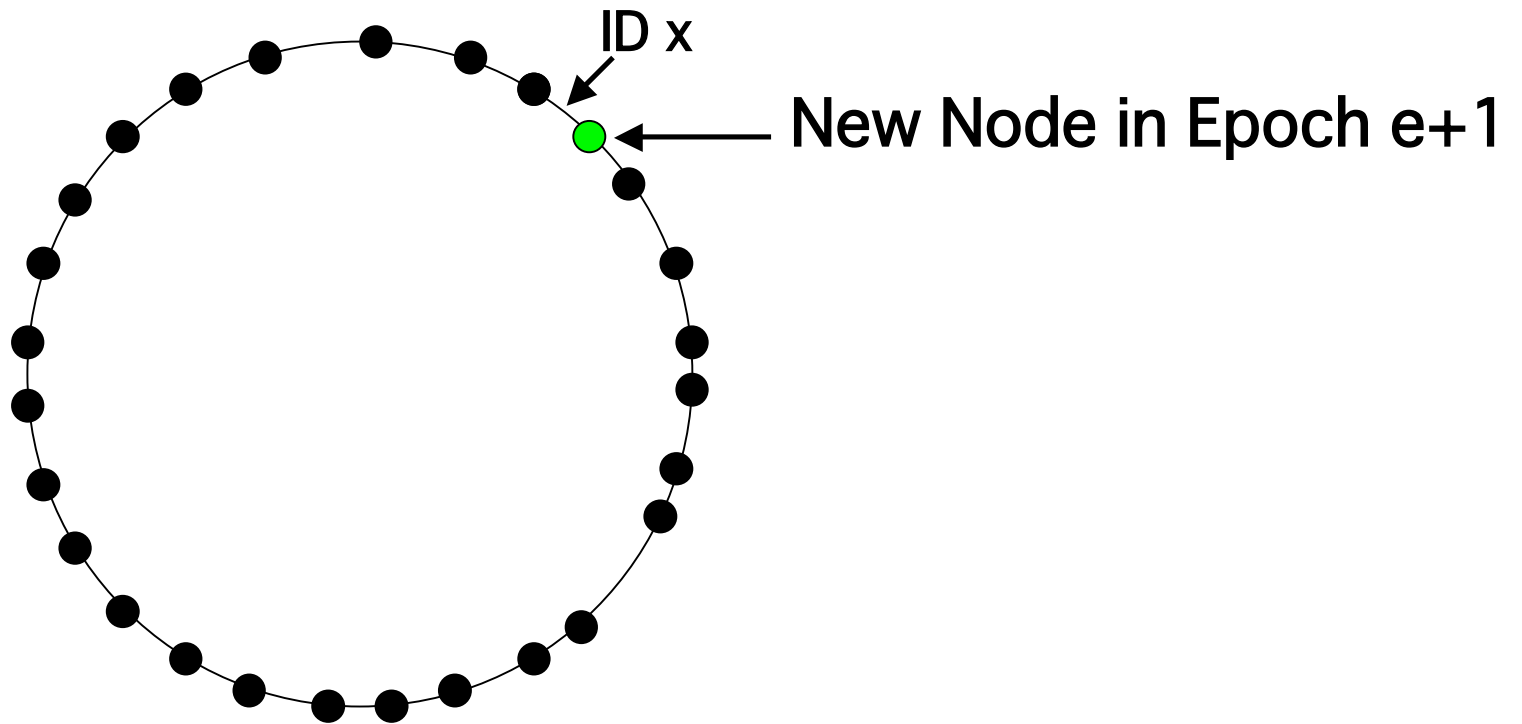
# Epoch changes

- **Each replica**
  - **Switches to new epoch when new configuration arrives**
  - **Only handles requests for objects it owns**
  - **Does state transfer**

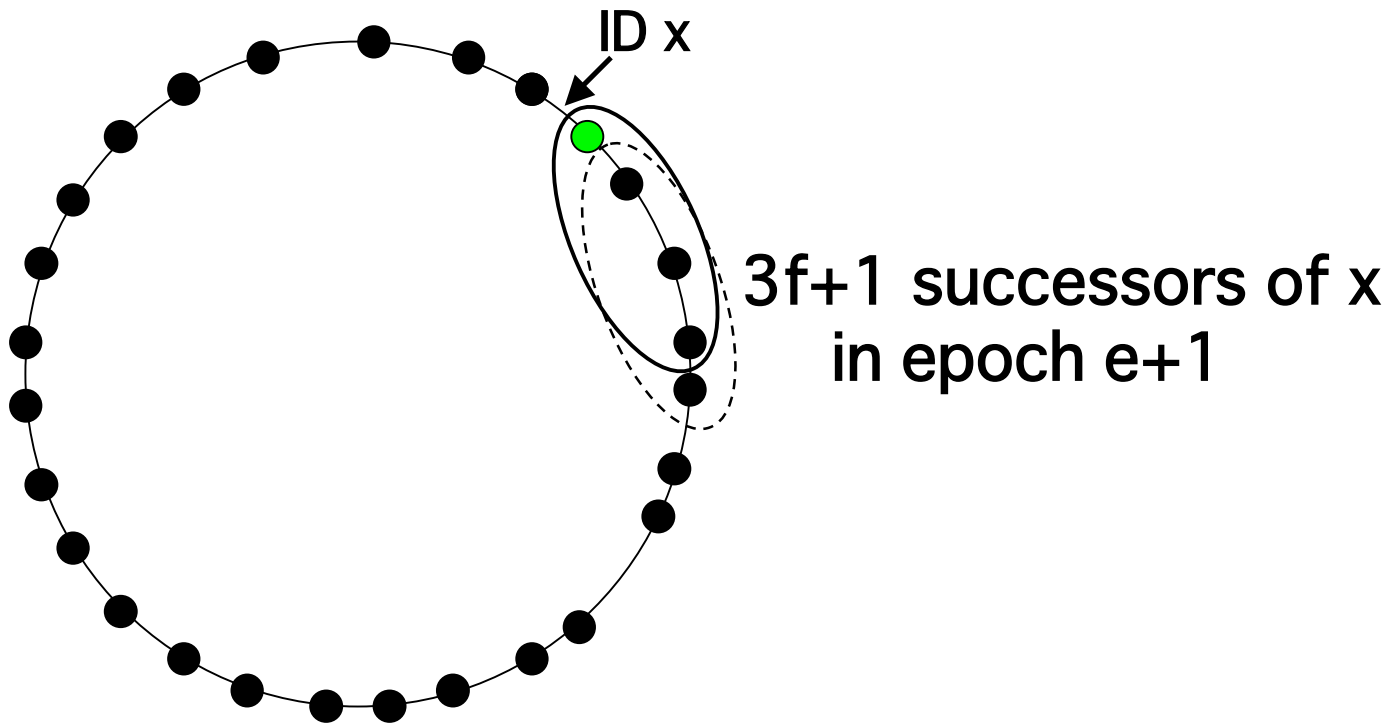
# Current Epoch



# Next Epoch



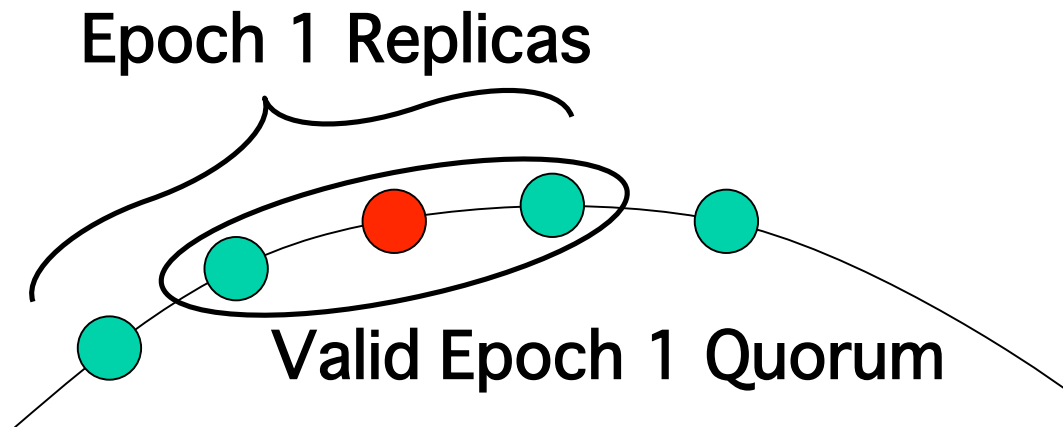
# Change in Ownership



# Epoch Constraint

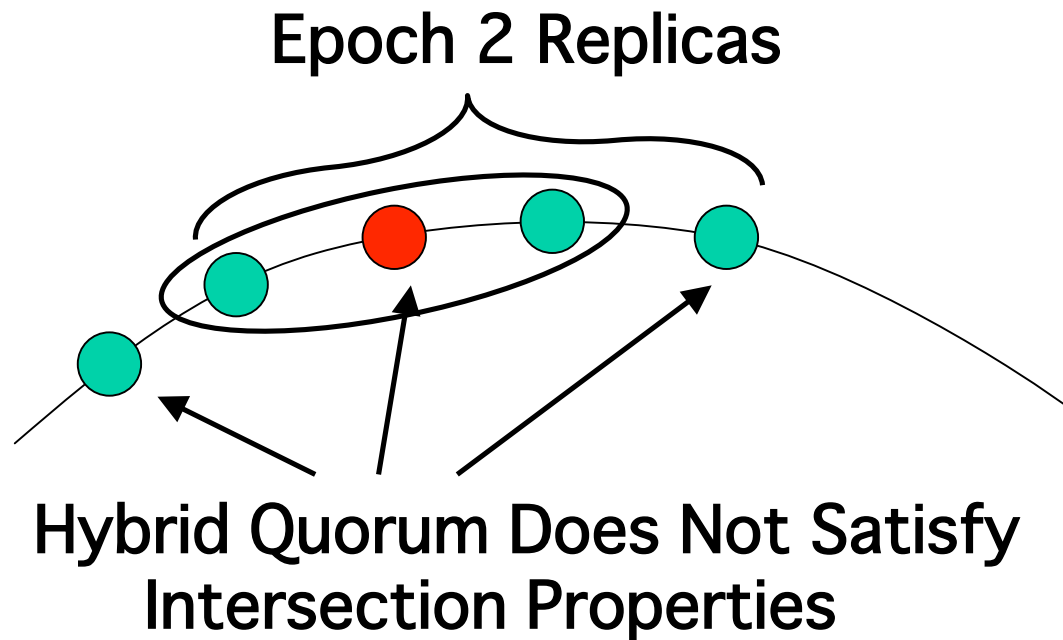
- **Client requests**
  - **Include epoch id**
  - **Rejected if wrong id**
  - **Client requires all responses from the same epoch**

# Epoch Changes



- **Operation 1 executes in epoch 1**

# Epoch Changes



- **Operation 2 must run in epoch 2**

# Conclusion

- **P2P is an interesting paradigm**
  - **Scalability**
  - **Easy reconfiguration**
  - **Potentially simpler application code**
- **But is it broadly useful?**
- **First make it work as well as possible**
  - **One-hop routing**
  - **Rosebud**

# **Programming Methodology Group**

- **Sameer Ajmani**
- **Anjali Gupta**
- **Roger Moh**
- **Steven Richman**
- **Rodrigo Rodrigues**
- **Liuba Shrira**
  
- **<http://www.pmg.lcs.mit.edu>**

# **Issues in Peer-to-Peer Computing**

**Barbara Liskov**

**MIT Laboratory for Computer Science**

**June 11, 2003**