

# ON CONSISTENCY OF DATA IN STRUCTURED OVERLAY NETWORKS

Presented by: Tallat M. Shafaat (KTH)  
Authors: Tallat M. Shafaat, Monika Moser,  
Ali Ghodsi, Thorsten Schütt,  
Seif Haridi, Alexander Reinefeld



# Overview

- Introduction
- Motivation
- Background
- Problem Statement
- How does the problem occur?
- How often does the problem occur?
- Techniques to reduce the effect
- Conclusion

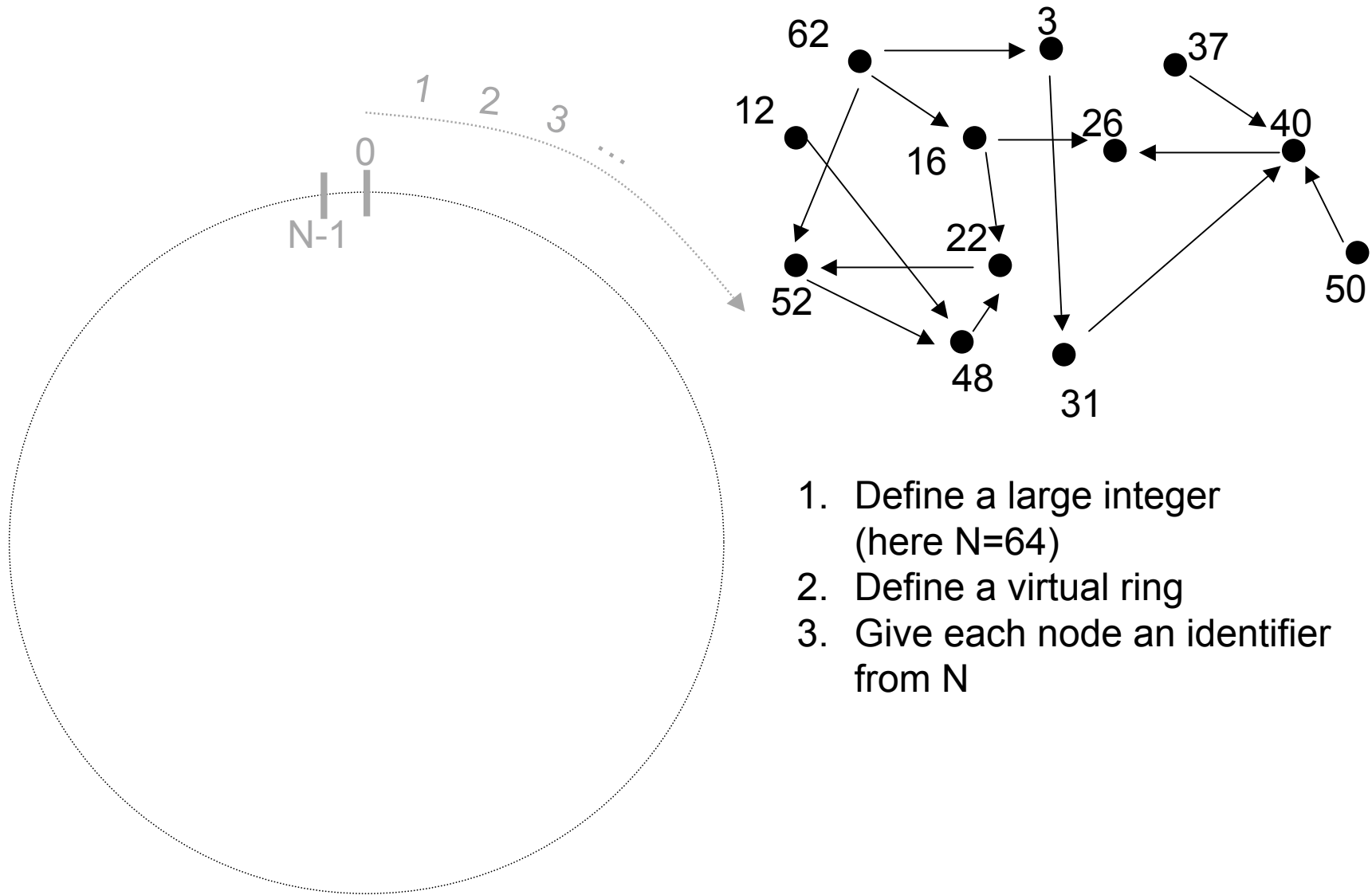
## Introduction

- Structured Overlay Networks (SONs) form a major class of peer-to-peer systems
- SONs provide lookup services for large-scale systems e.g. [SMKKB02, RD01]
- A lookup maps a **key** to a **node** in the system
- The node can then be used for **data storage** [G06, D05] or processing

## Motivation

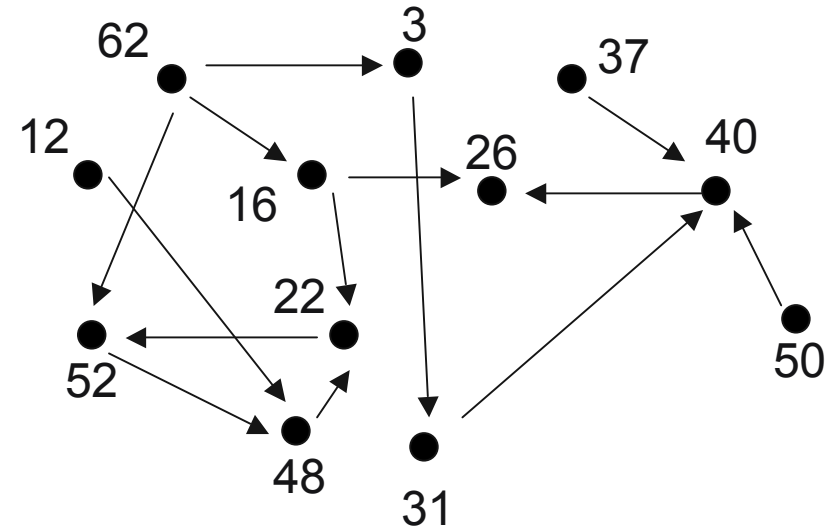
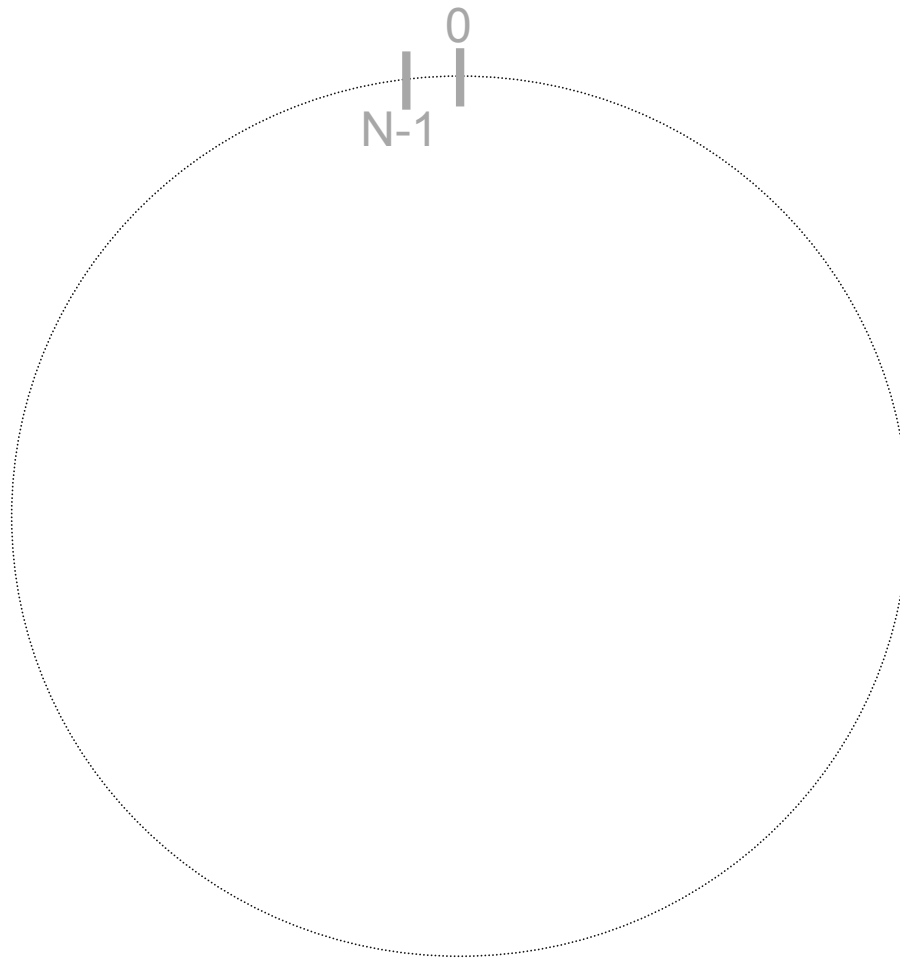
- SONs typically guarantee eventual consistency or are “best effort”
- Distributed data applications require stronger consistency guarantees e.g. DFS, DDb
- We investigate techniques to build a system on SONs that provides
  - **probabilistic consistency guarantees**
  - **consistency guarantees with high probability**

# Background



1. Define a large integer (here  $N=64$ )
2. Define a virtual ring
3. Give each node an identifier from  $N$

# Background: A ring-based overlay

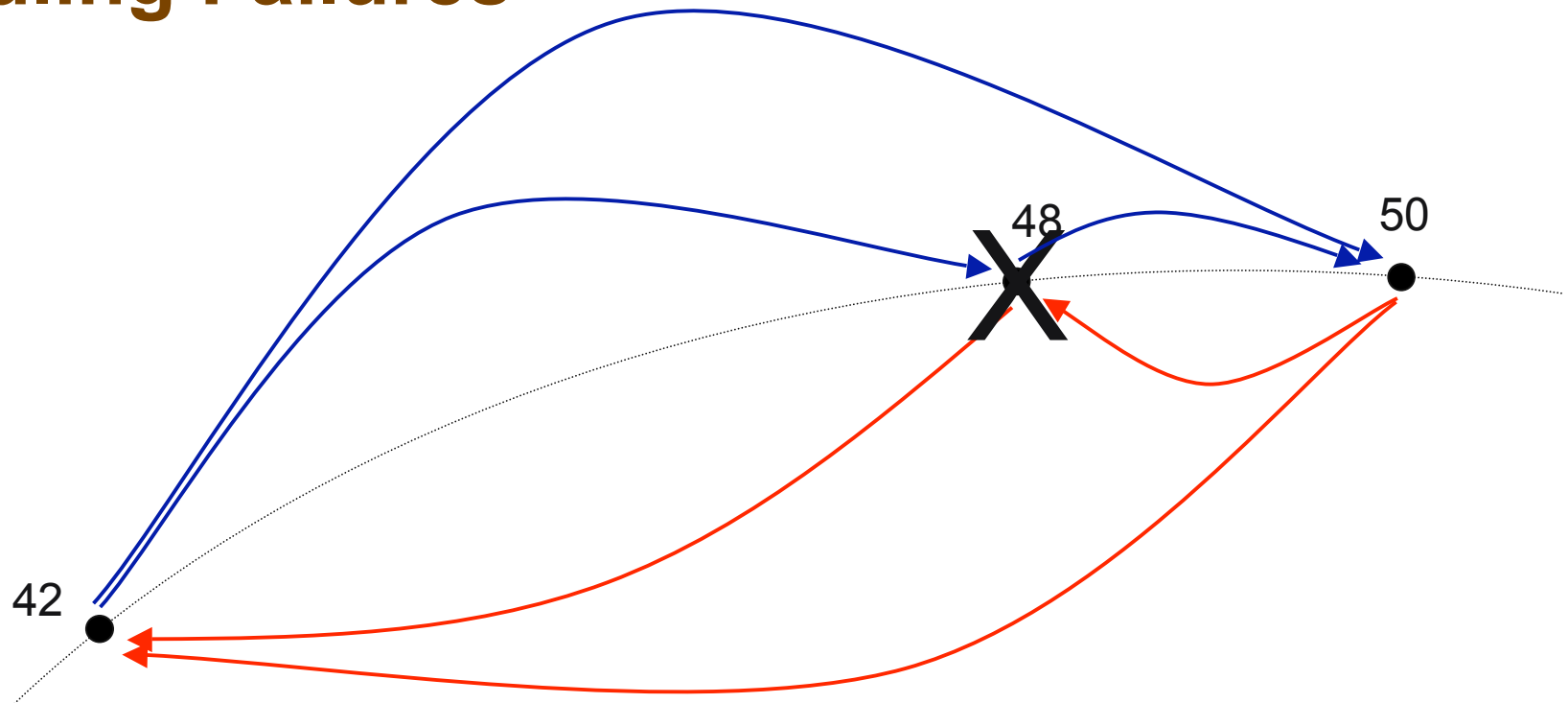


1. Define a large integer (here  $N=64$ )
2. Define a virtual ring
3. Give each node an identifier from  $N$
4. Place nodes on the ring
5. Create links





# Handling Failures



- **48** fails
- **42** detects **48** has failed. It updates its **successor pointer**
- **50** updates its **predecessor pointer**

—▶ **successor pointer**  
—▶ **predecessor pointer**

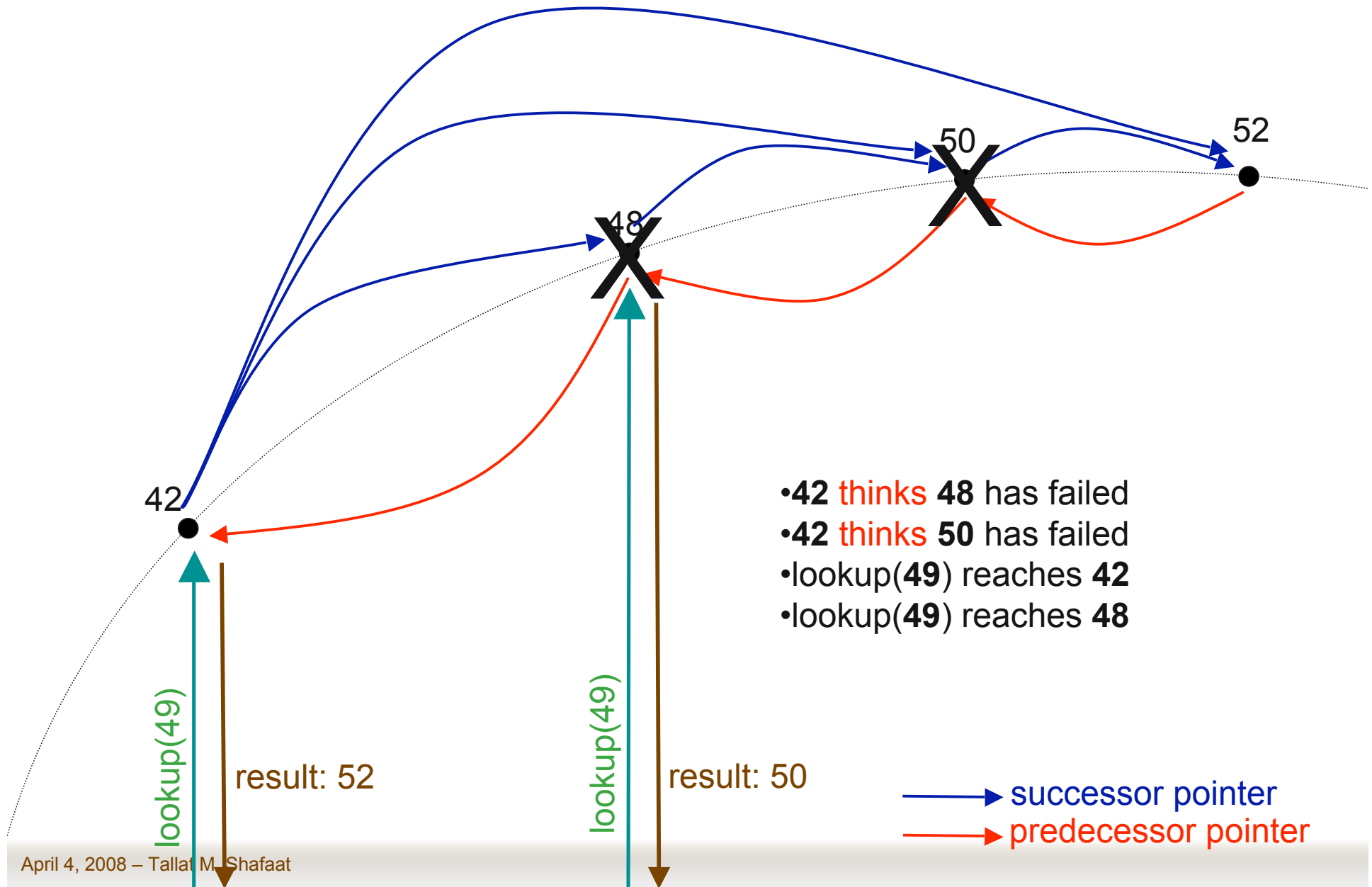
# Overview

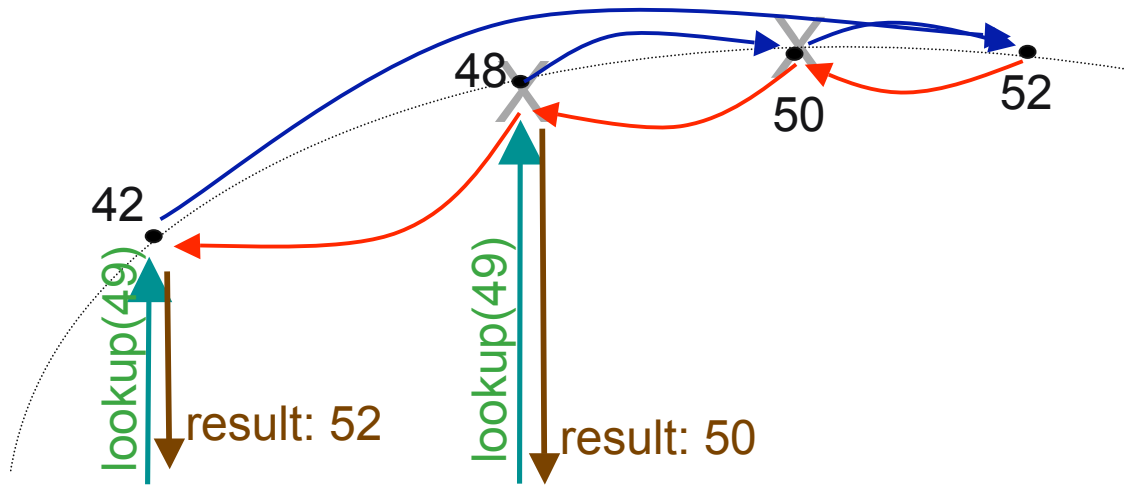
- ✓ Introduction
- ✓ Motivation
- ✓ Background
- **Problem Statement**
  - How does the problem occur?
  - How often does the problem occur?
  - Techniques to reduce the effect
  - Conclusion

## What can go wrong?

- Lookup consistency
  - **A state of the system is consistent if, in that state, lookups made for the same key from different nodes, return the same node**
  - **Otherwise, it is 'inconsistent'**
  - **How can this happen?**

# Incorrect Failure detection



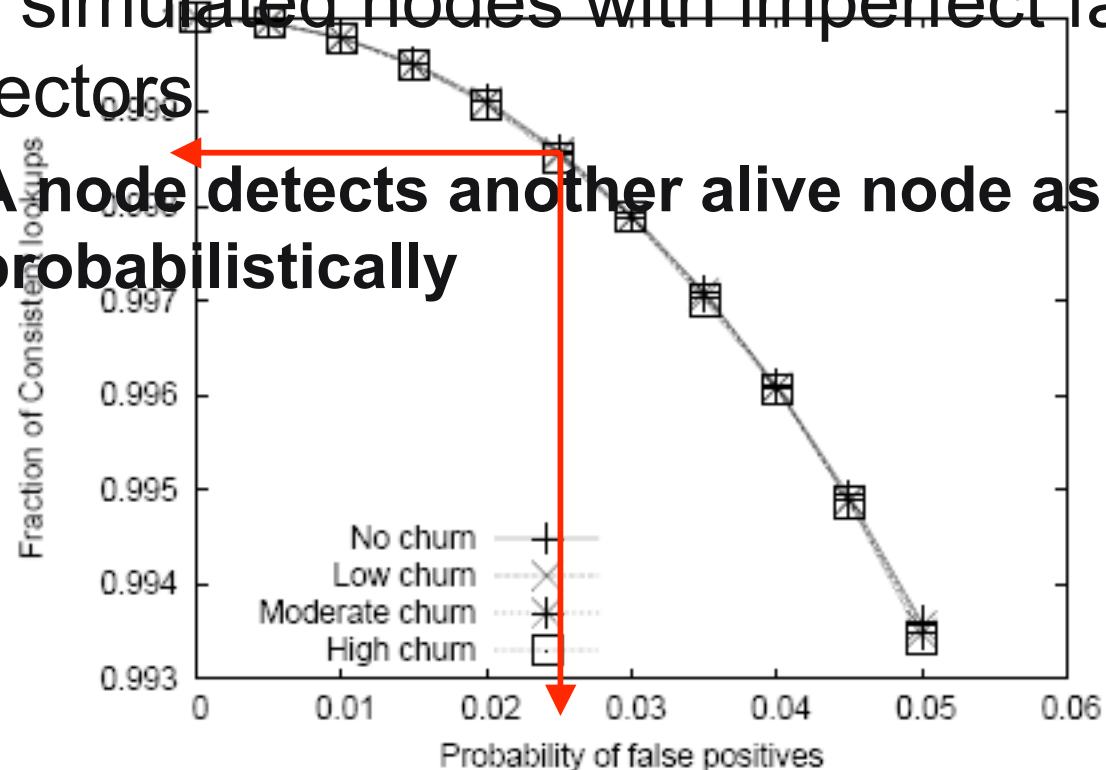


## The problem

- An update/write for identifier 49 can thus be stored at any of:
  - node 50
  - node 52
- Multiple reads for identifier 49 can return different values from:
  - node 50
  - node 52

## How often does this occur?

- We simulated nodes with imperfect failure detectors
  - A node detects another alive node as dead probabilistically



[ZGSK05]

# Overview

- ✓ Introduction
- ✓ Motivation
- ✓ Background
- ✓ Problem Statement
- ✓ How does the problem occur?
- ✓ How often does the problem occur?
- Techniques to reduce the effect
- Conclusion

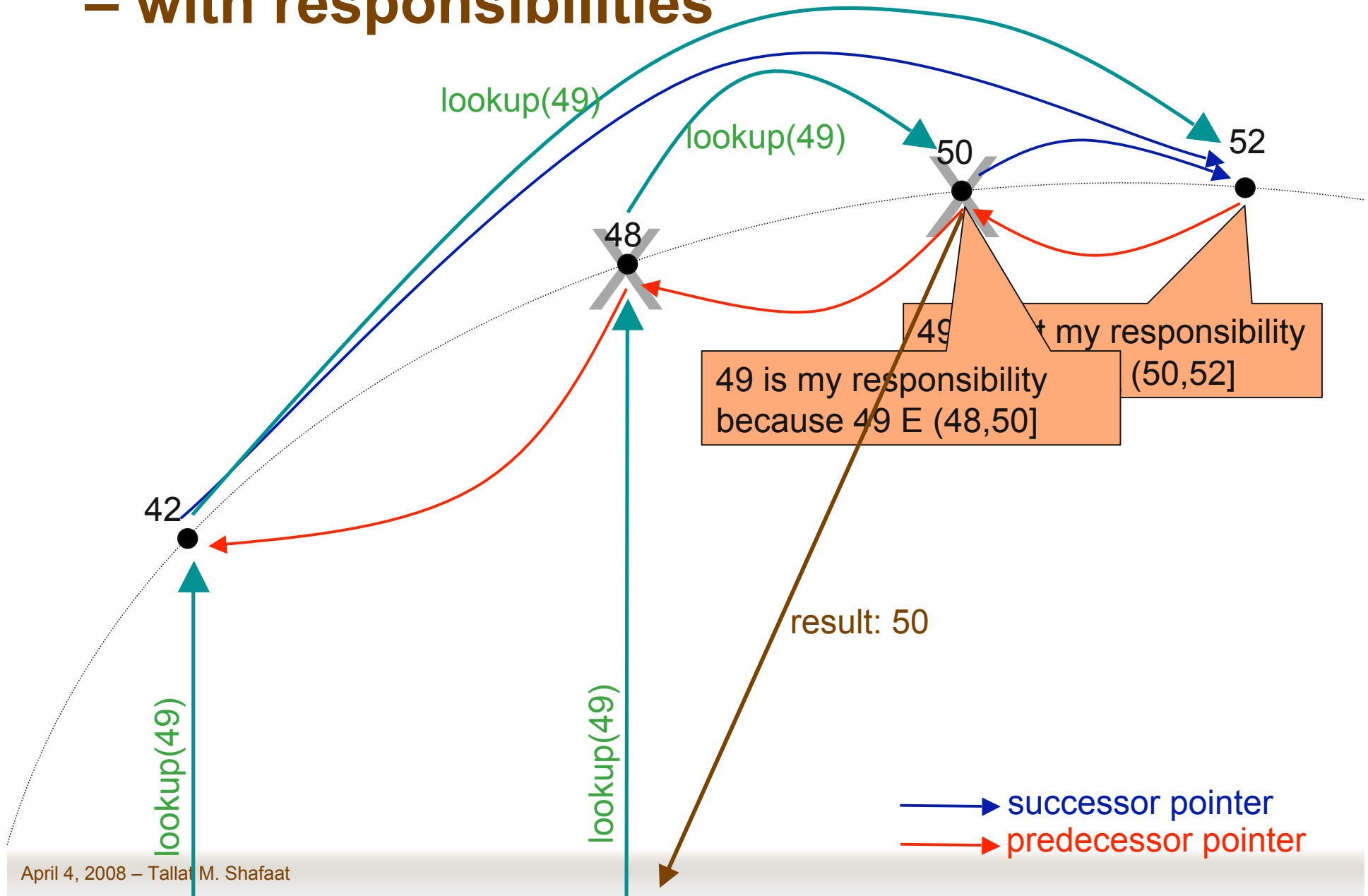
## How to decrease the effect?

- We use two methods to decrease the effect of lookup inconsistency
  - **Technique 1: Local responsibility**
  - **Technique 2: Using quorum-based algorithms**

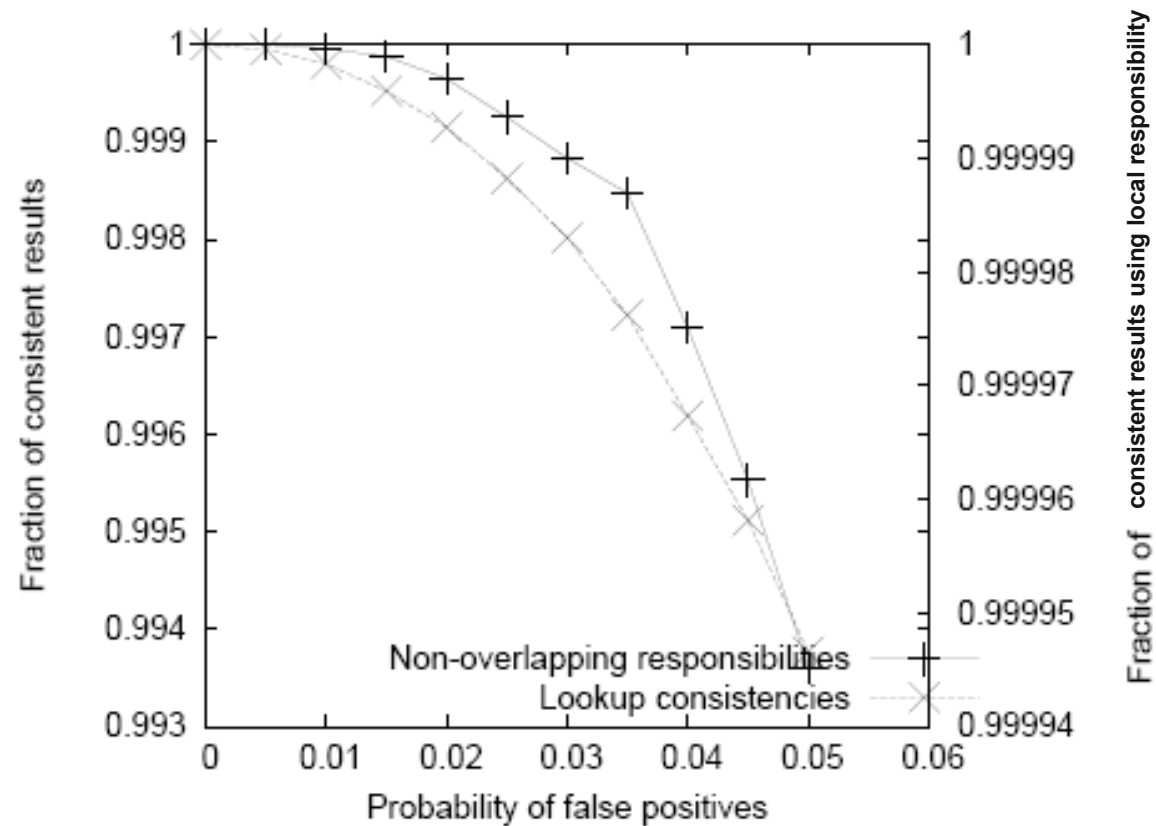
## Tech. 1: Local Responsibility

- A node is said to be **locally responsible** for a certain key, if the key is in the range between its **predecessor** and **itself**
- Lookup returns from the locally responsible node
  - **Before returning result of a lookup, the node checks if it is locally responsible for the key**

# Incorrect Failure detection – with responsibilities



# How much decrease?



# Overview

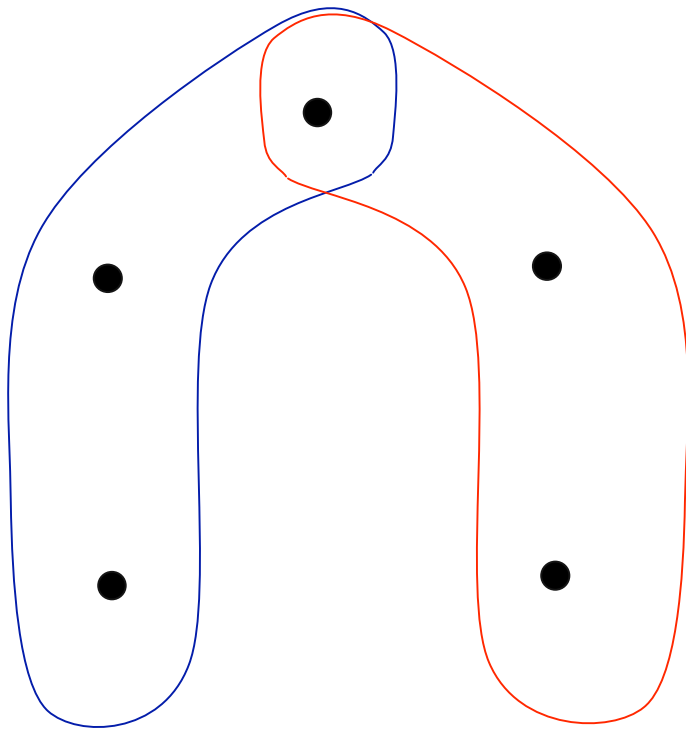
- ✓ Introduction
- ✓ Motivation
- ✓ Background
- ✓ Problem Statement
- ✓ How does the problem occur?
- ✓ How often does the problem occur?
- ✓ Techniques to reduce the effect
  - **Local responsibilities**
  - **Quorum-based Algorithms**
- Conclusion

## Tech. 2: Quorum-based algorithms

- To prevent loss of data, items are replicated in SONs
- Replication schemes in SONs:
  - **Symmetric replication [GAH05]**
    - Use a globally known function to determine a set of keys under which the data is stored
  - **Replication on successor list [SMKKB02]**
- Using replicas, the problem of lookup inconsistency is distributed

## Quorum based algorithms

- We work on majority based quorum techniques (MBQTs) as it is most widely used and most robust to failures
- A MBQT requires to read and write to a majority
- It relies on the fact that quorums always intersect



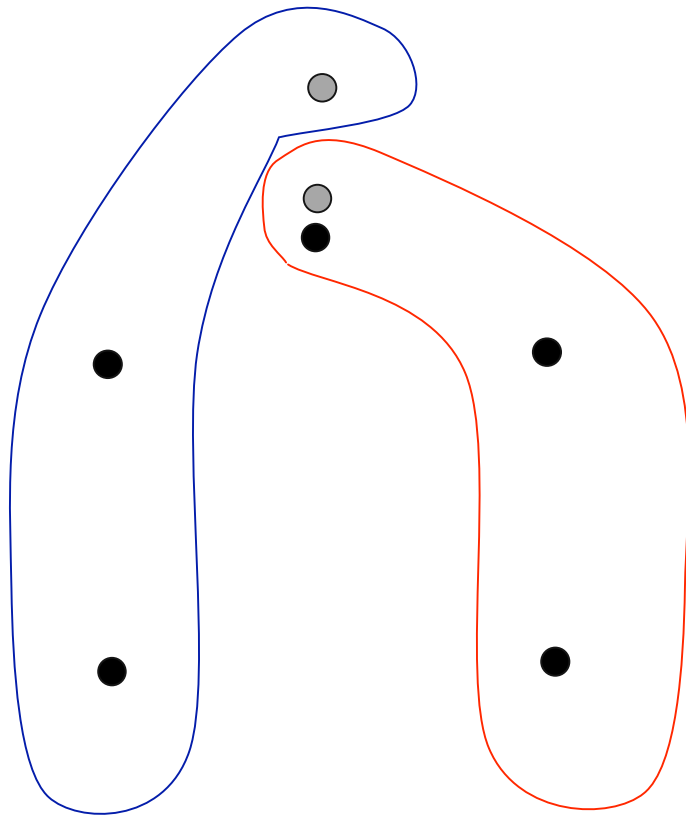
## Majority-based quorums

- Consider a system with 5 replicas
- **Update** requires a majority
- **Read** requires a majority
- Majorities always overlap, thus:
  - read will get latest value
  - multiple reads will always get same value

## MBQT in SONs

- Since number of replicas are constant, any quorums created from majority will never be disjoint
- In SONs, due to lookup inconsistency, the number of replicas is not constant
- Thus, two quorums can be created that don't intersect

## MBQTs in SONs



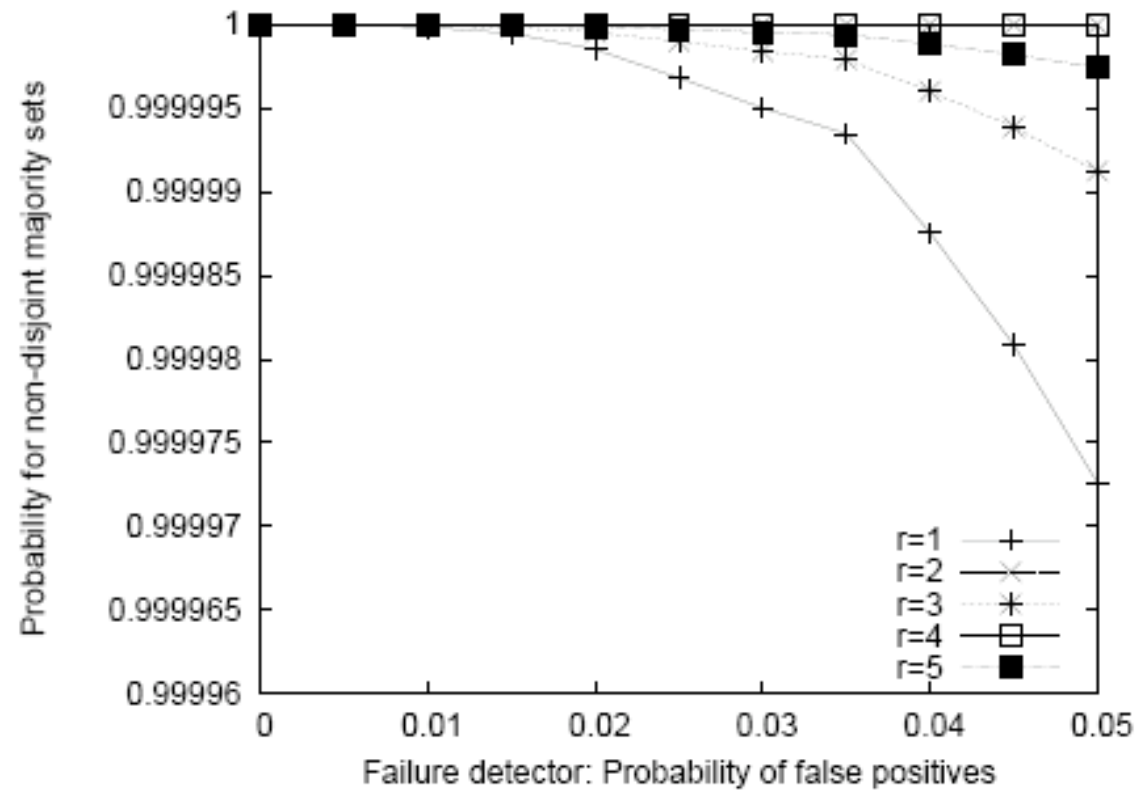
- Consider a system with 5 replicas
- Each replica is found by making a lookup to the key of the data item
- Due to lookup inconsistency, replica 1 appears as two nodes
- **Update** requires a majority
- **Read** requires a majority
- Majorities don't intersect, thus:
  - read does not get latest value
  - read on **blue majority** will return different result than a read on **red majority**

How does this technique help then?

Previously, if there was a lookup inconsistent, it will generate inconsistent data

Now, even with lookup inconsistency, multiple quorums exist that intersect

# Quorum based algorithms



## Conclusion

- Design and effectiveness of failure detectors is crucial
- Using quorum based techniques and local responsibilities, we can make a system that is consistent with high probability
- Inaccuracy of failure detectors is the main contributor to lookup inconsistency, not churn
- Using even number of replicas increases consistency compared to odd number of replicas

# References

- [GOH] A. Ghodsi, O. Alima and S. Haridi [Symmetric Replication for Structured Peer-to-Peer Systems](#), DBISP2P2005, Norway
- [SMKKB02] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek and Hari Balakrishnan [Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications](#), ACM SIGCOMM 2001, San Deigo, CA
- [RD01] A. Rowstron and P. Druschel [Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems](#), IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Germany, 2001.
- [G06] A. Ghodsi [Distributed k-ary System: Algorithms for Distributed Hash Tables](#), Doctoral Dissertation, KTH—Royal Institute of Technology, 2006
- [D05] F. Dabek [A Distributed Hash Table](#), Doctoral Dissertation, MIT — Massachusetts Institute of Technology, 2005
- [ZGSK05] S.Q. Zhuang, D. Geels, I. Stoica, R.H. Katz [On Failure Detection Algorithms in Overlay Networks](#), INFOCOM'05, Miami, 2005

# Questions

- Comments / Suggestions / Questions
  - **tallat (a) kth.se**
- Thank you!