DHT Routing Geometries and Chord

DIETRICH GEISLER

Initial Attempts

- Freenet, Gnutella, BitTorrent, Napster
- File-sharing services that take advantage of networks
- Two major approaches for file delivery system
  - Central indexing which searches for data – vulnerable to failure
  - Querying machines for file – inefficient and potentially incorrect

Problem

How do we store and access distributed data?

Peer-to-Peer Systems

- Distributed system for managing data
- No central controller
- No notion of hierarchy among member nodes

Distributed Hash Tables

- Structure of a hash table – (key, value) pairs
- Member nodes contain local keys and values
- Nodes have information for retrieving ‘neighbor’ nodes in the form of keys
- Only a subset of all references are stored on a given node

Introduction to Chord

- Chord consists of a DHT protocol and program
- Nodes are placed on a ring structure
- Each node acts independently

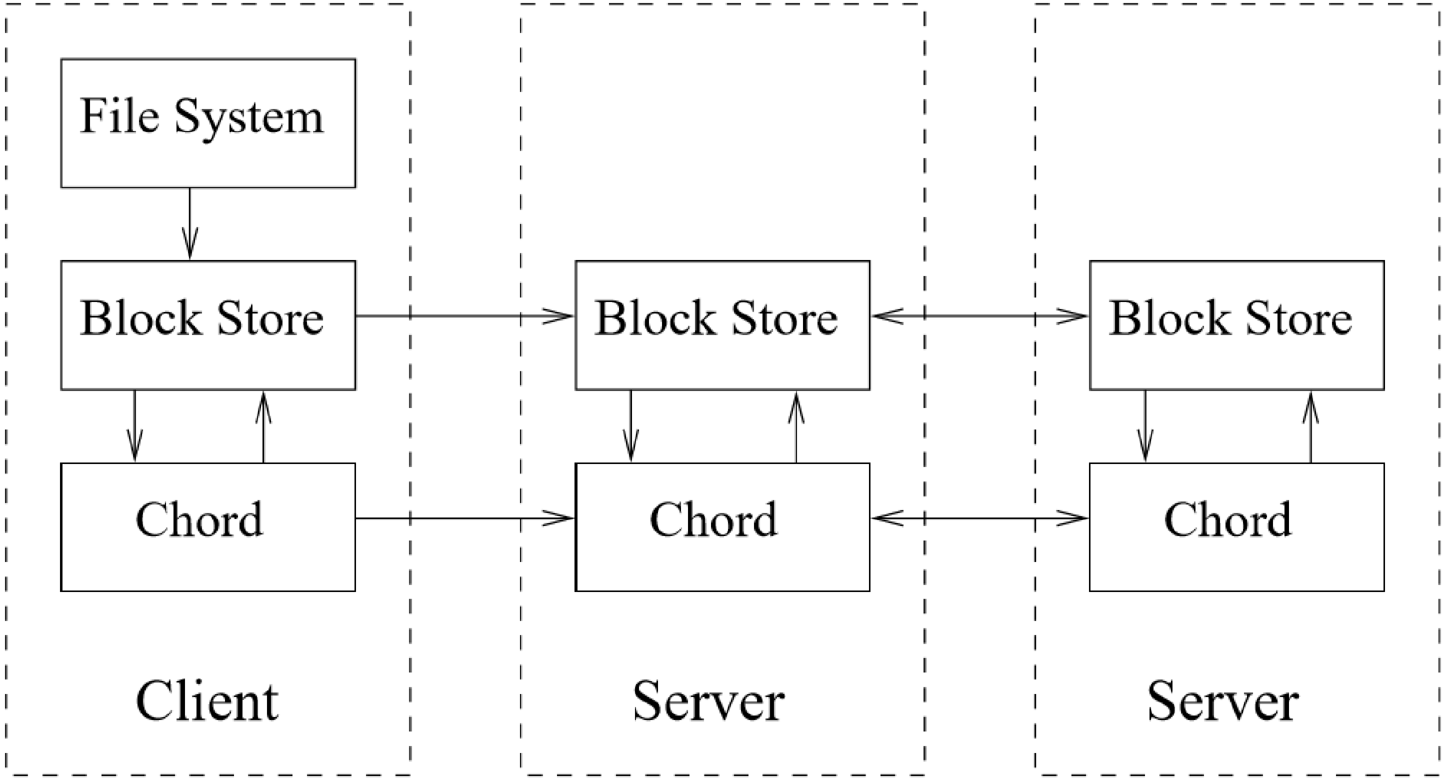
Chord Goals

- Load Balancing
- Decentralization
- Scalability
- Flexible Naming
- Availability

Outline of Chord

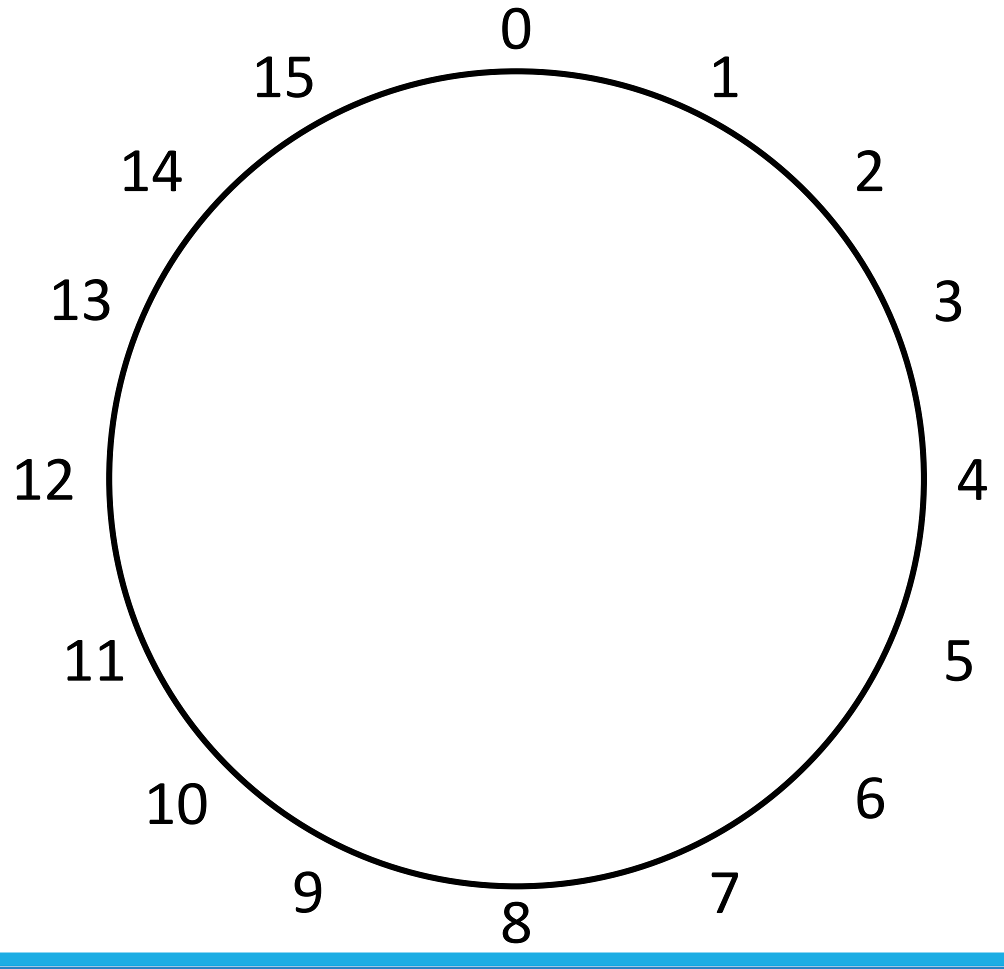
- Key assignment
- Linking nodes
  - Finger tables
- Adding/Removing nodes
- Optimizations
  - Concurrency
  - Load Balancing

Chord Structure



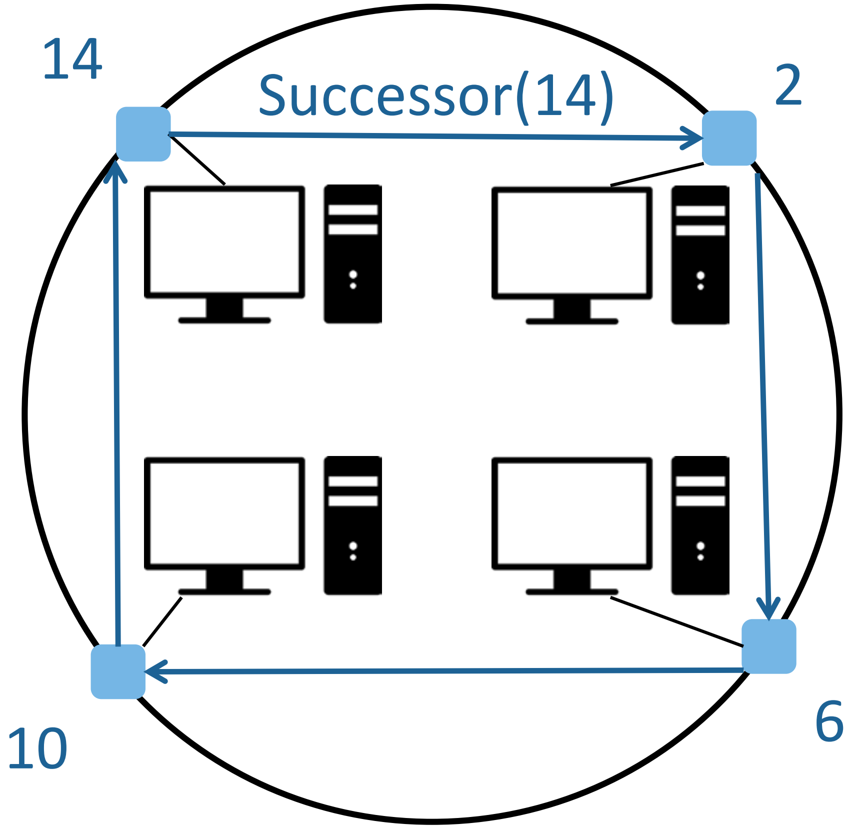
From **Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications**

Ring of Nodes



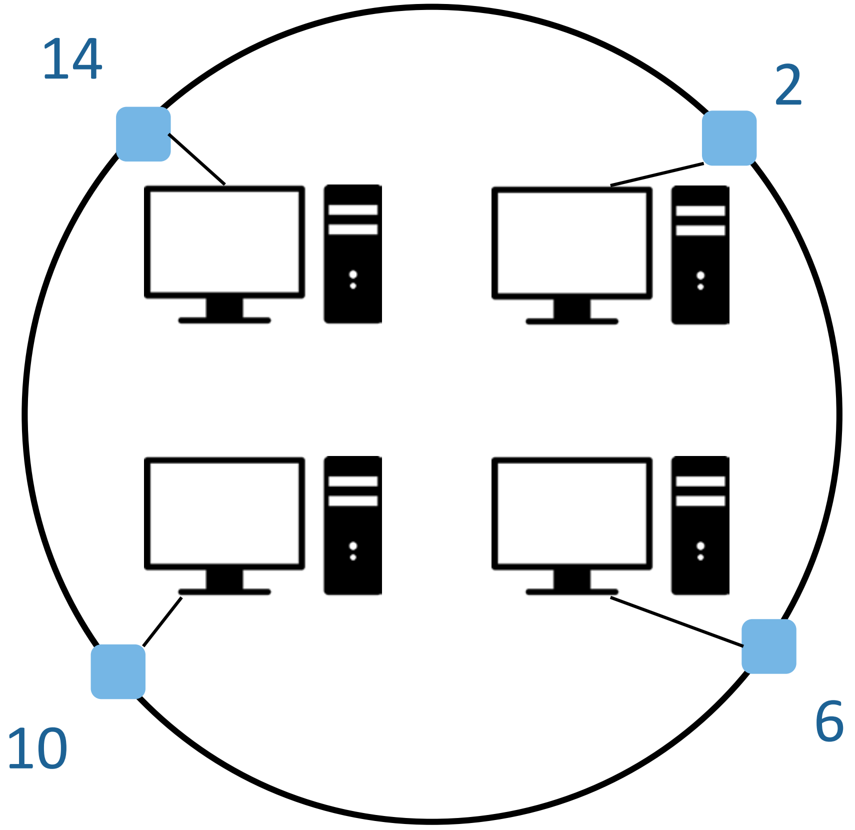
$m = 4, 2^m = 16$

Key Assignment in Chord



$m = 4$, $2^m = 16$

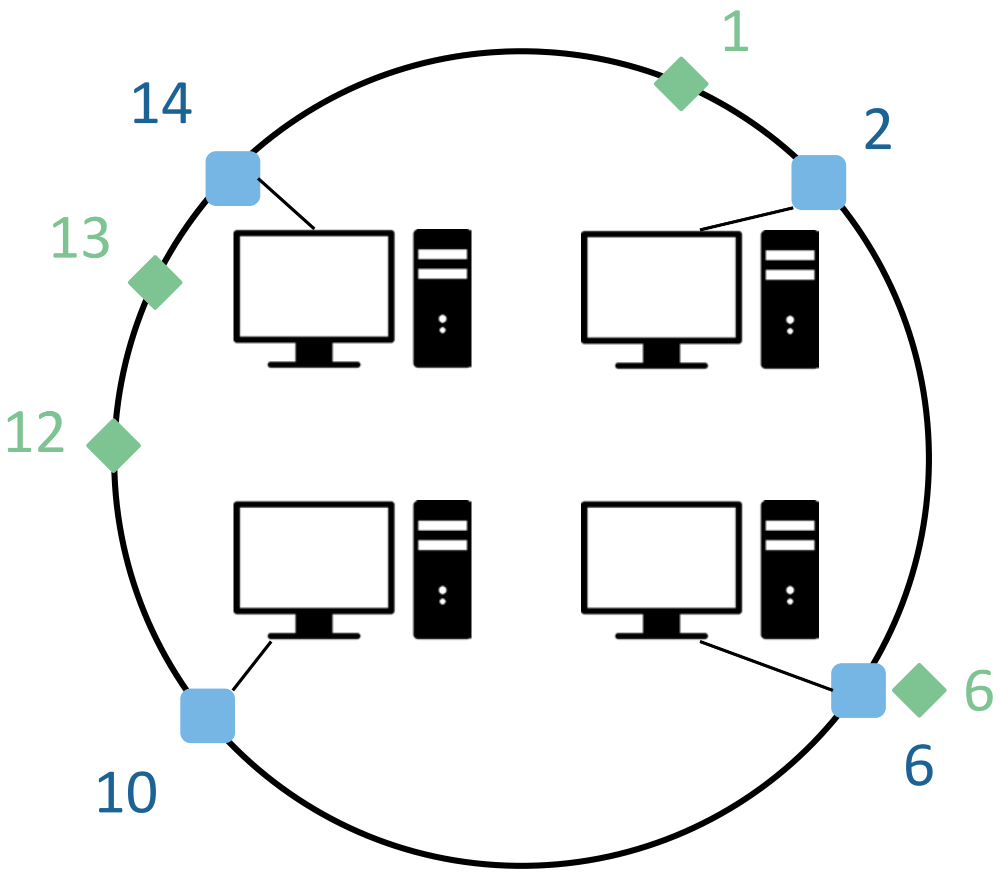
Key Assignment in Chord



$m = 4$, $2^m = 16$

Key Assignment in Chord

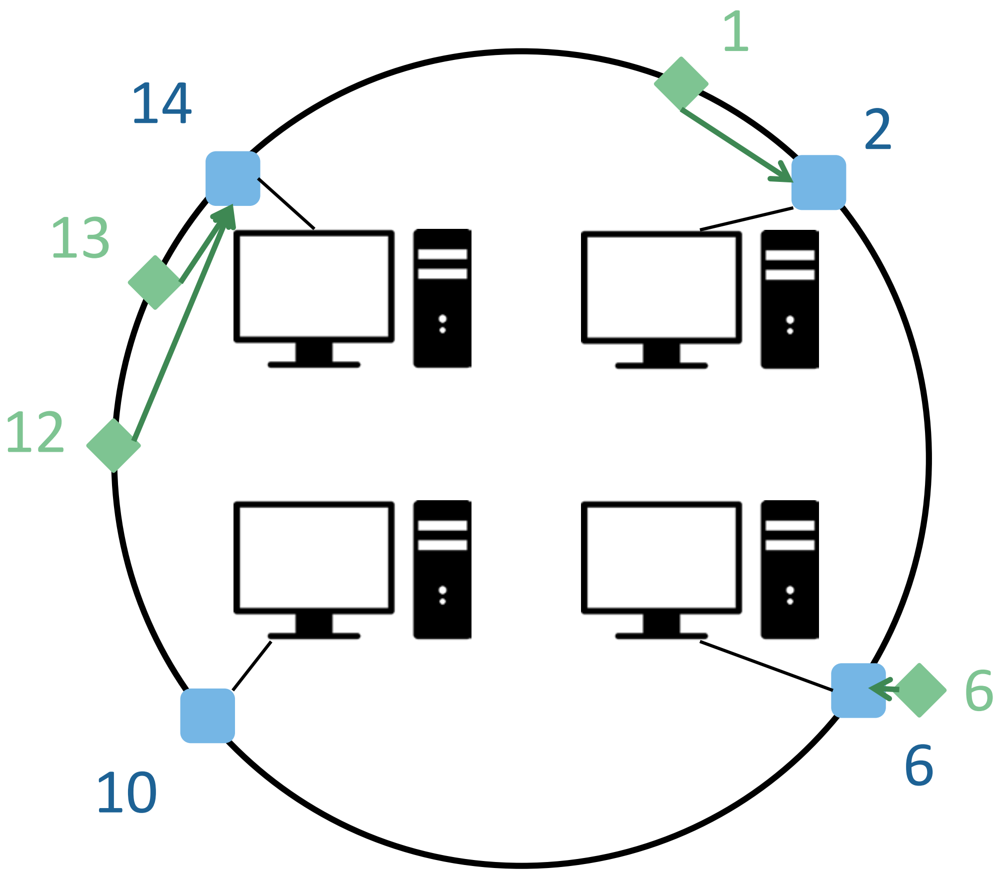
1	v_0
6	v_1
12	v_2
13	v_3



$m = 4, 2^m = 16$

Key Assignment in Chord

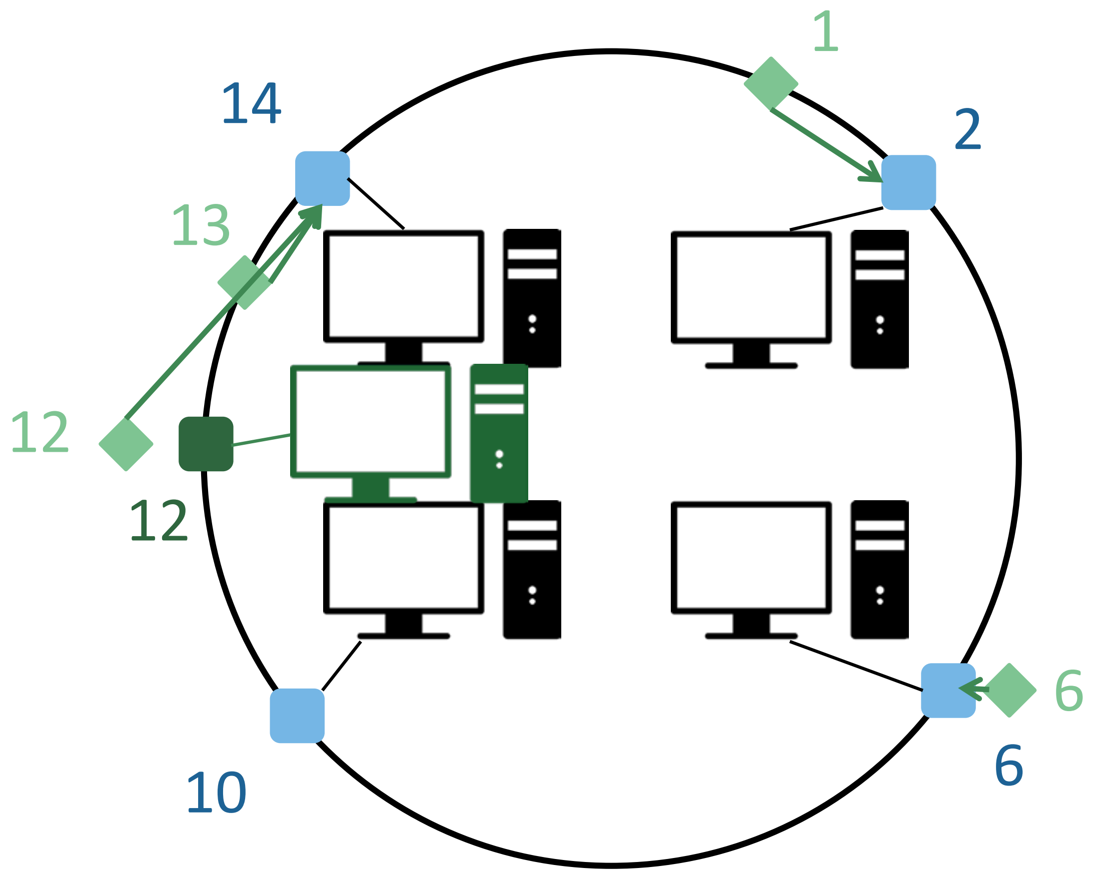
1	v_0
6	v_1
12	v_2
13	v_3



m = 4, 2^m = 16

Key Assignment in Chord

1	v_0
6	v_1
12	v_2
13	v_3

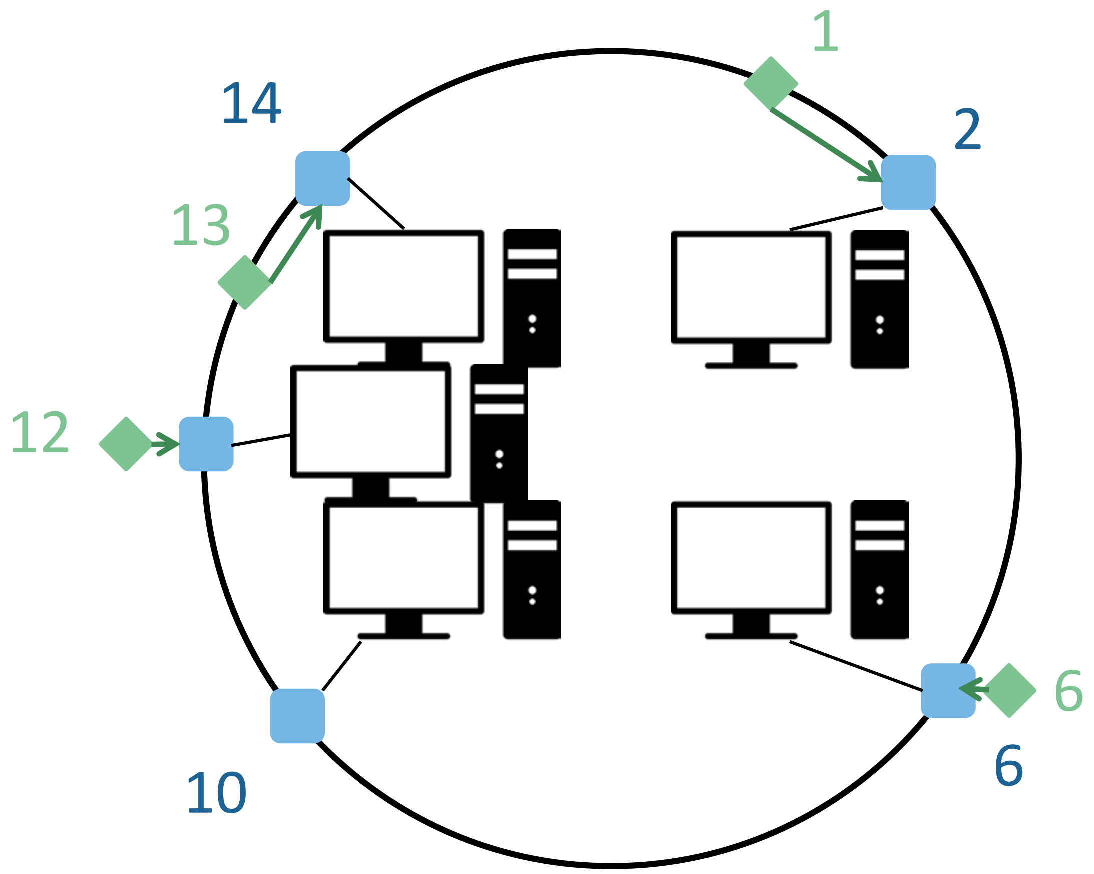


$m = 4$, $2^m = 16$

Which keys do we need to update?

Key Assignment in Chord

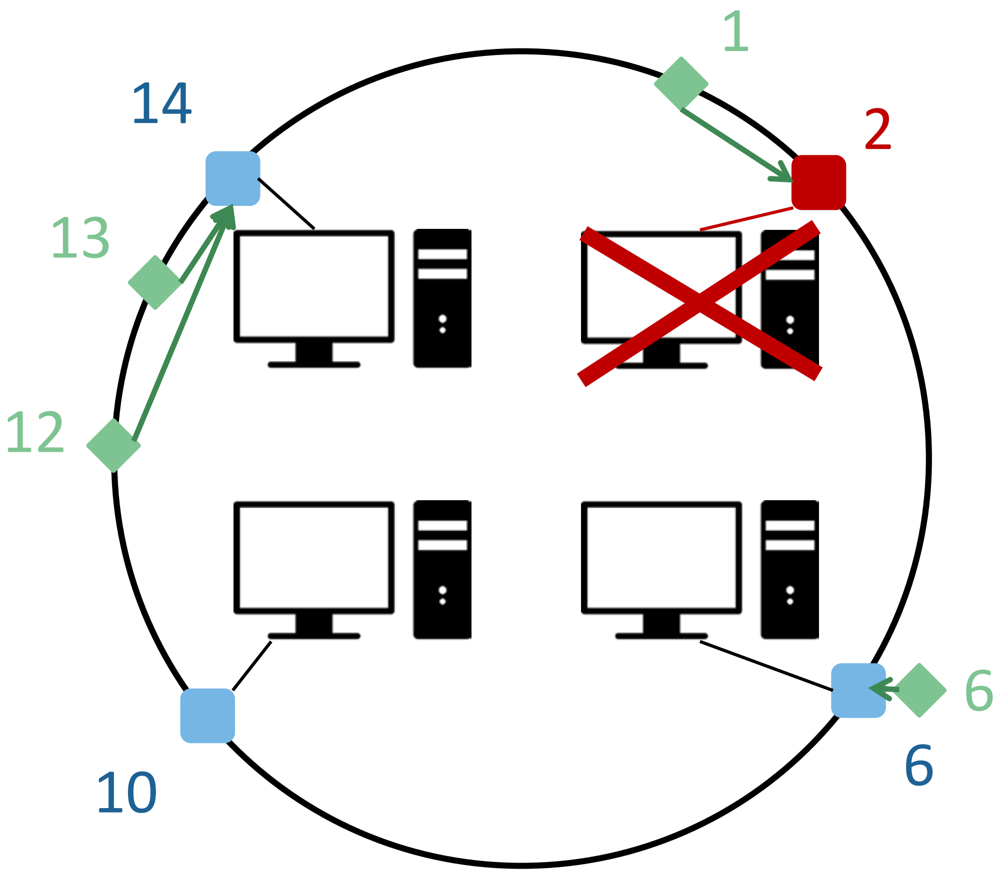
1	v_0
6	v_1
12	v_2
13	v_3



$m = 4$, $2^m = 16$

Key Assignment in Chord

1	v_0
6	v_1
12	v_2
13	v_3

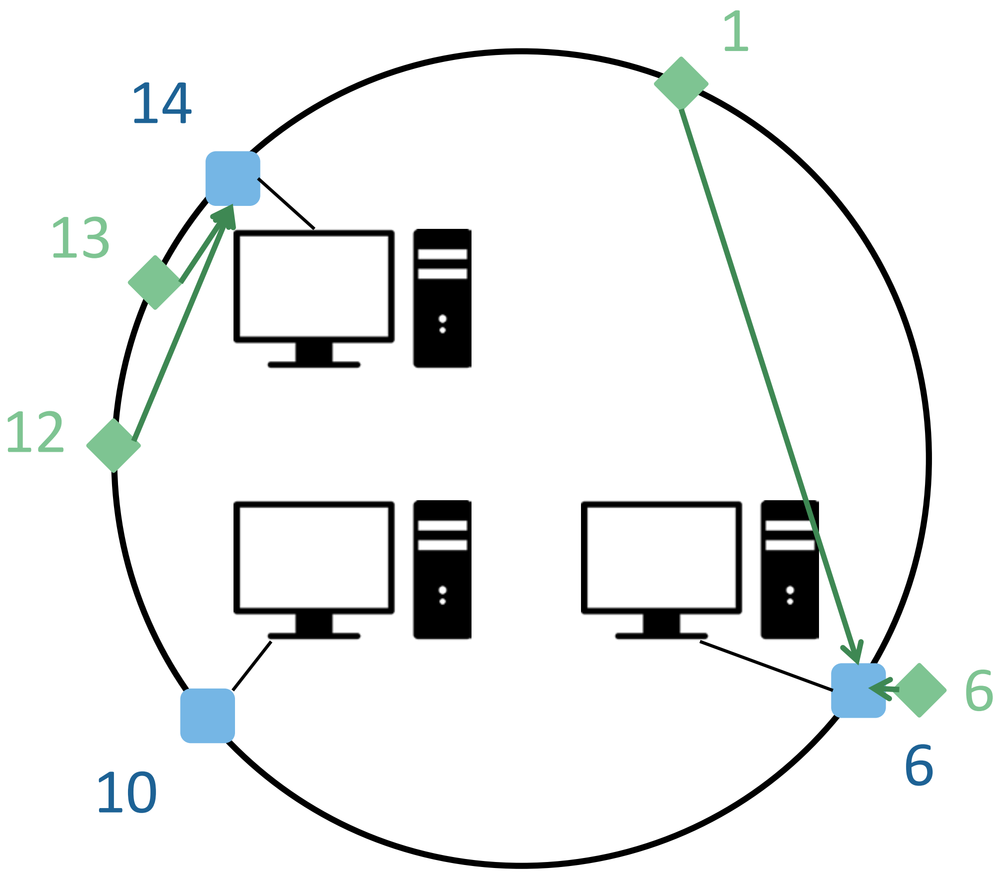


$m = 4, 2^m = 16$

Which keys do we need to update?

Key Assignment in Chord

1	v_0
6	v_1
12	v_2
13	v_3



$m = 4, 2^m = 16$

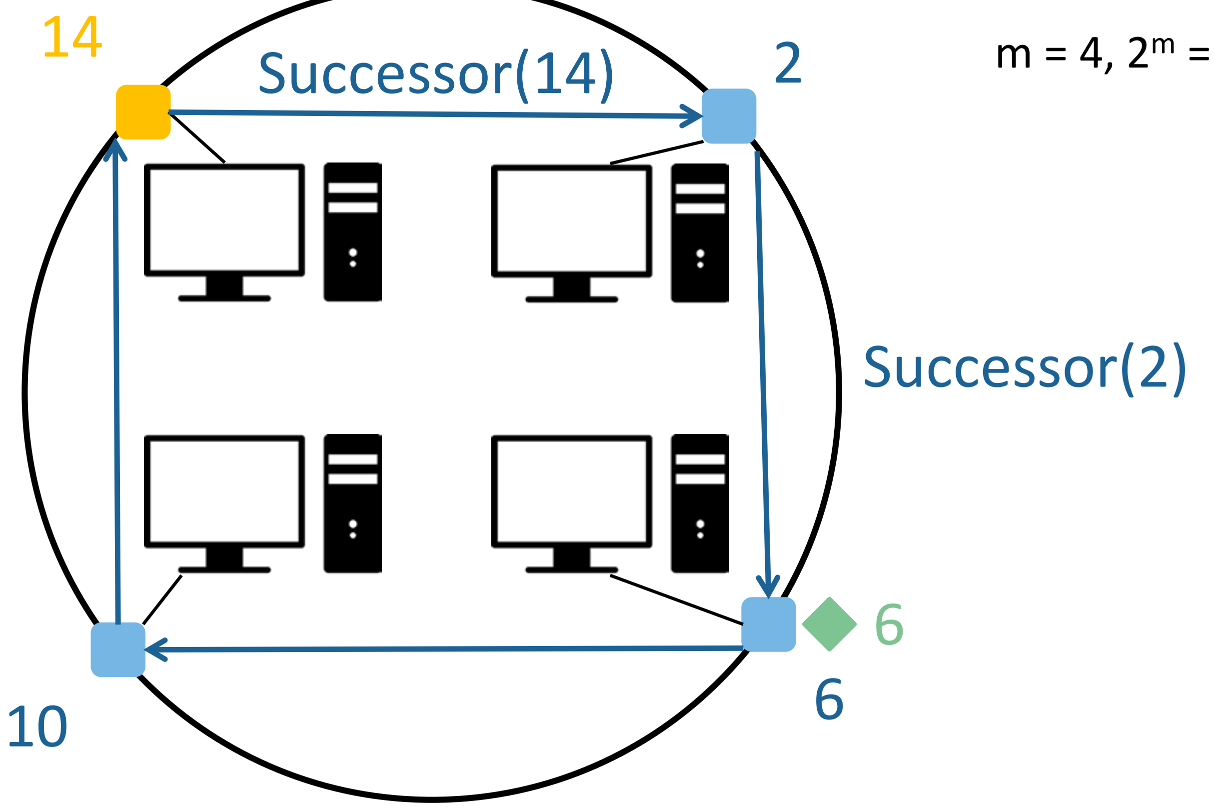
Properties of Consistent Hashing

- For any set of N nodes and K keys, with high probability,

- Each node is responsible for at most $(1 + \varepsilon)K=N$ keys
  - $\varepsilon = O(\log N)$

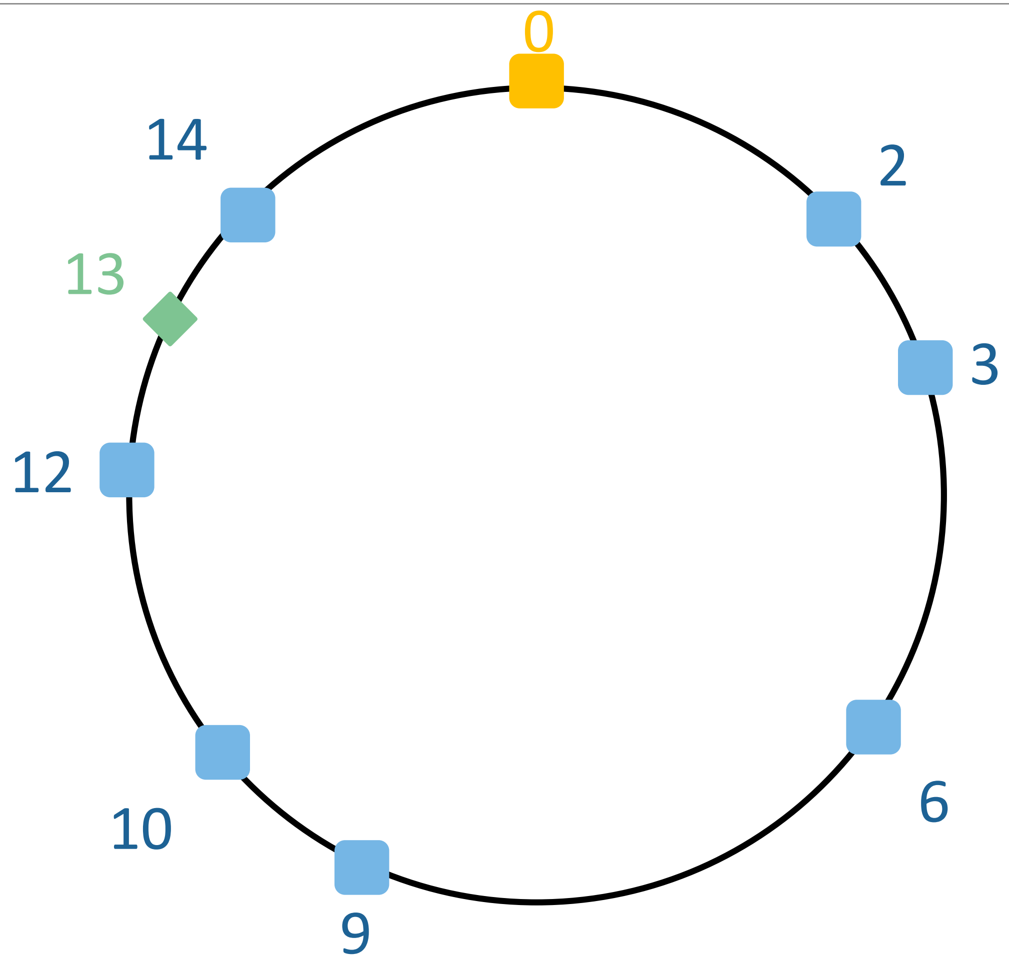
- When an $(N +1)$st node joins or leaves the network, responsibility for $O(K=N)$ keys changes hands (and only to or from the joining or leaving node)

Naïve Linking



$m = 4$, $2^m = 16$

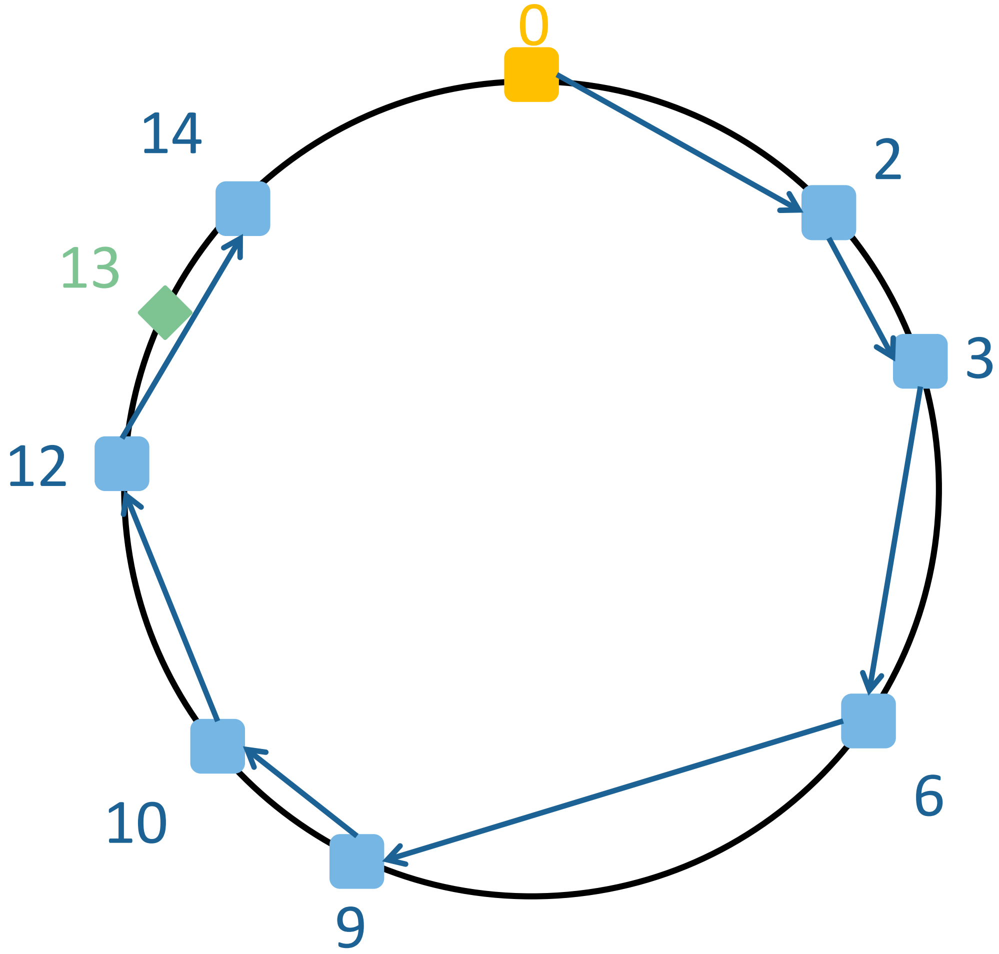
Naïve Linking



$m = 4$, $2^m = 16$

How do we get the item with identifier 13 from node 0 using only successor nodes?

Naïve Linking



$m = 4,\ 2^m = 16$

Finger Tables

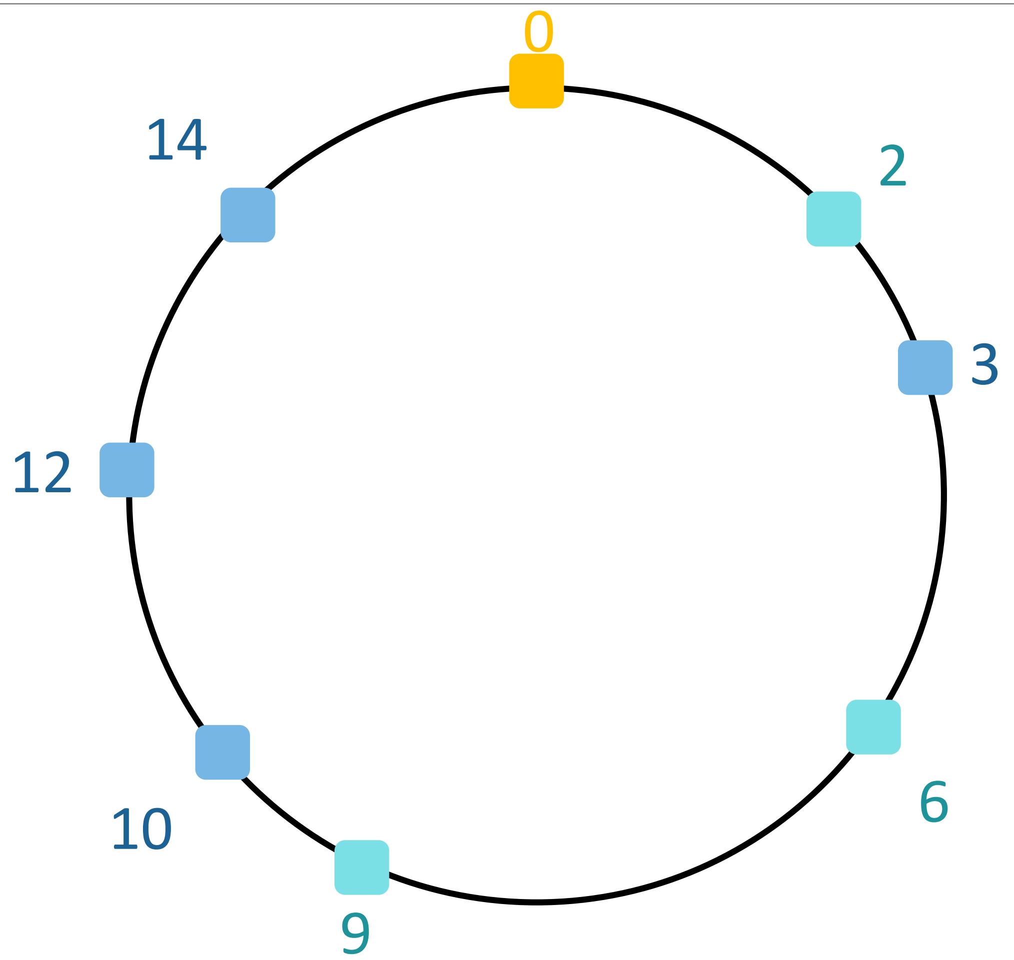
- Maintain a list of nodes at intervals of the ring
- Provide good coverage while minimizing space
- Tables where *i*th entry of the node *n* is the successor of

$$(n + 2^{i-1}) \bmod 2^m$$

Naïve Linking

Finger Table for node 0

i	ident.	node
1	1	2
2	2	2
3	4	6
4	8	9

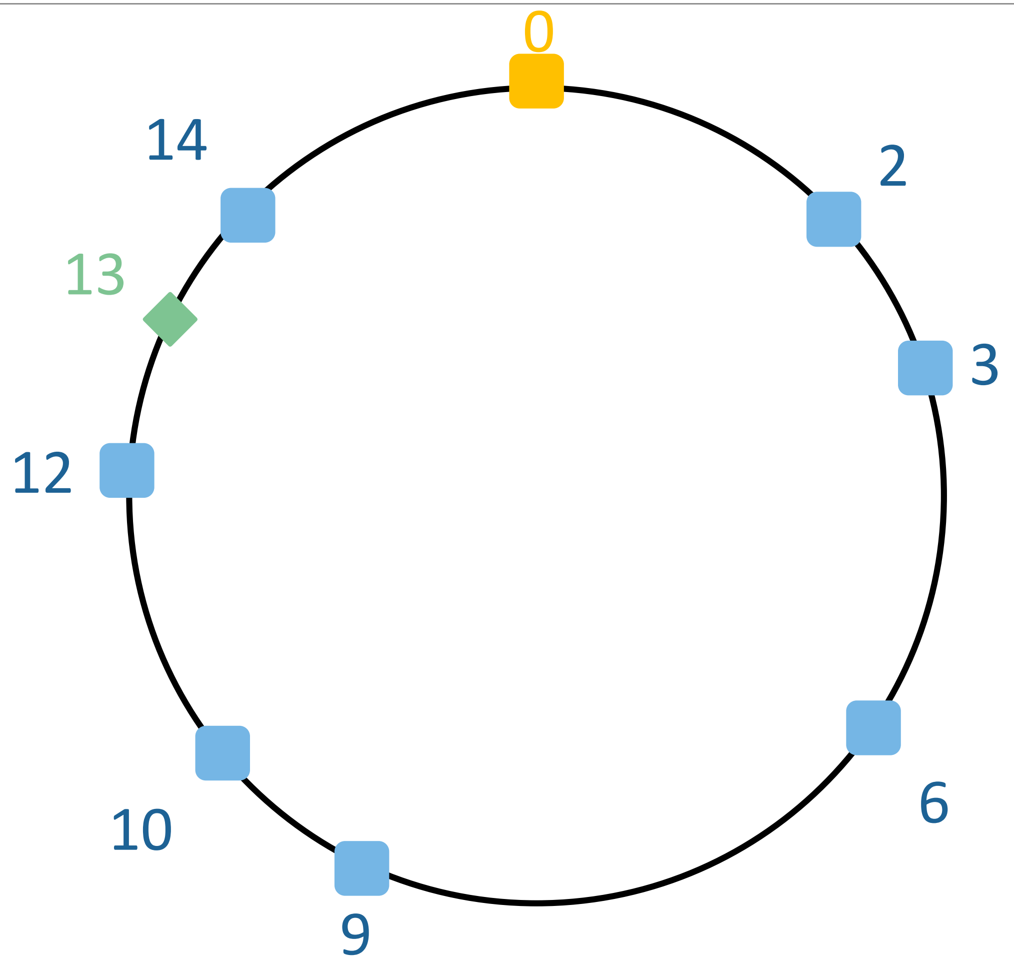


$m = 4$, $2^m = 16$

Naïve Linking

Finger Table for node 0

i	ident.	node
1	1	2
2	2	2
3	4	6
4	8	9



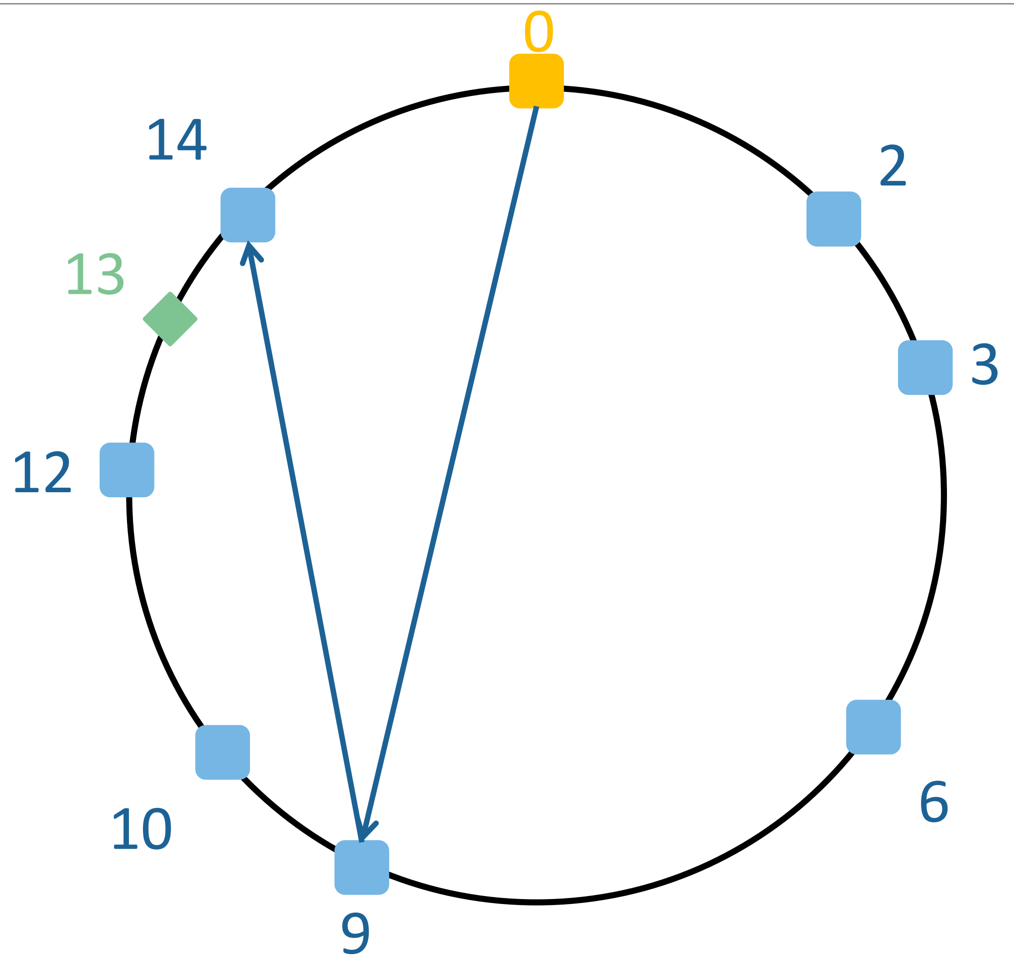
$m = 4$, $2^m = 16$

How do we get the item with identifier 13 from node 0? (We don't have a notion of predecessor yet!)

Naïve Linking

Finger Table for node 0

i	ident.	node
1	1	2
2	2	2
3	4	6
4	8	9



$m = 4$, $2^m = 16$

Node Joins

- Preserve ability to locate a given key
  - Each node’s successor must be correctly maintained
  - For every $k \in Keys$, *successor(k)* is responsible for *k*
- To help with this, each node will maintain a predecessor node in addition to the nodes stored in the finger table

Node Join Requirements

- Initialize the predecessor and fingers of node *n*
- Update the fingers and predecessors of existing nodes to reflect the addition of *n*
- Notify the higher layer software so that it can transfer state (e.g. values) associated with keys that node n is now responsible for.

Node Join Requirements

- Initialize the predecessor and fingers of node *n*

$$O(\log N)$$

- Update the fingers and predecessors of existing nodes to reflect the addition of *n*

$$O(\log N)$$

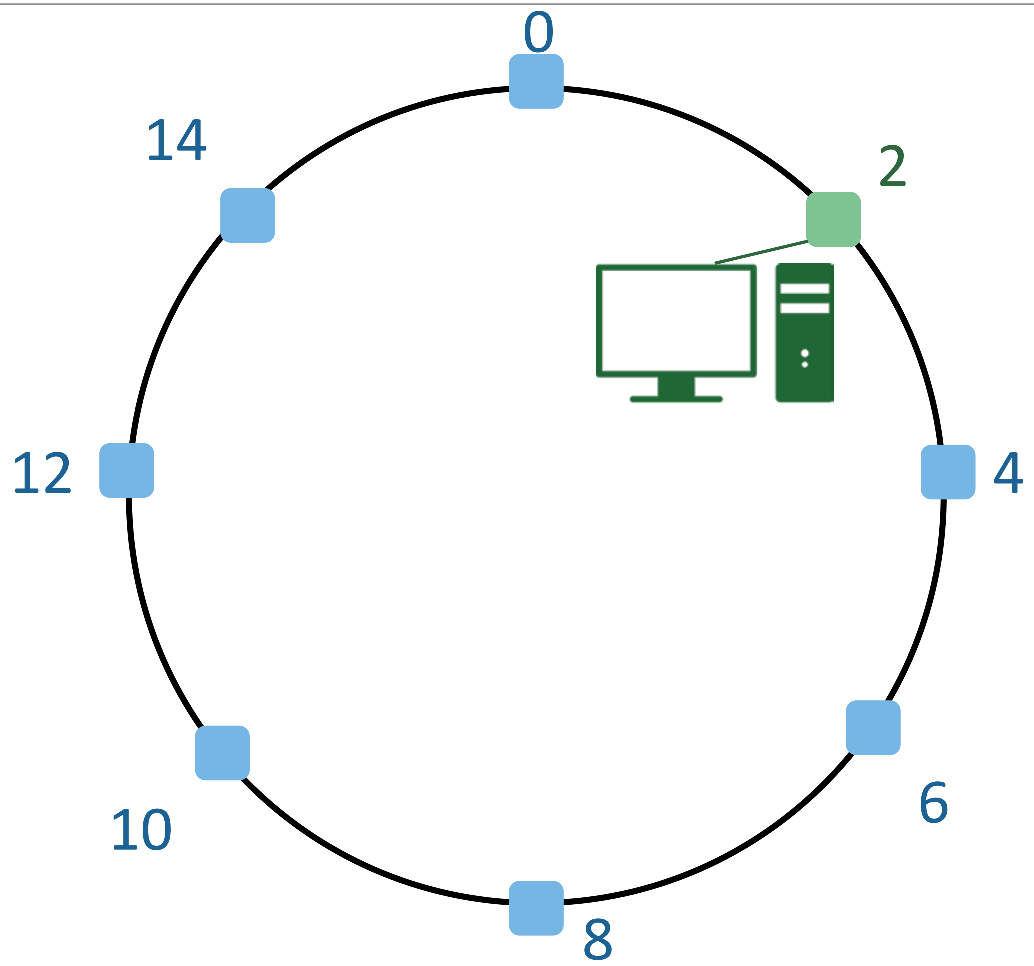
- Notify the higher layer software so that it can transfer state (e.g. values) associated with keys that node n is now responsible for.

$$O(1)$$

Node Join

Finger Table for node 0

i	ident.	node
1	1	4
2	2	4
3	4	4
4	8	8



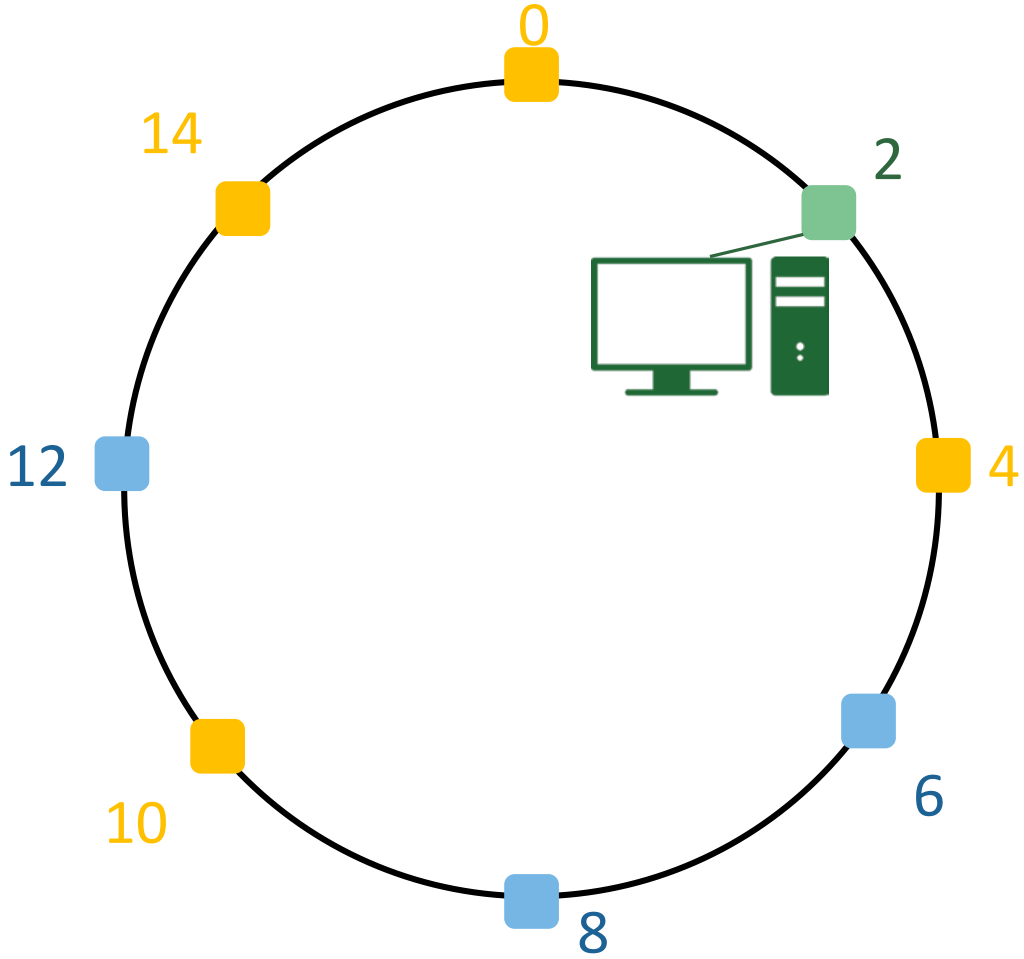
$m = 4$, $2^m = 16$

Which nodes have to update their tables?

Node Join

Finger Table for node 0

i	ident.	node
1	1	2
2	2	2
3	4	4
4	8	8

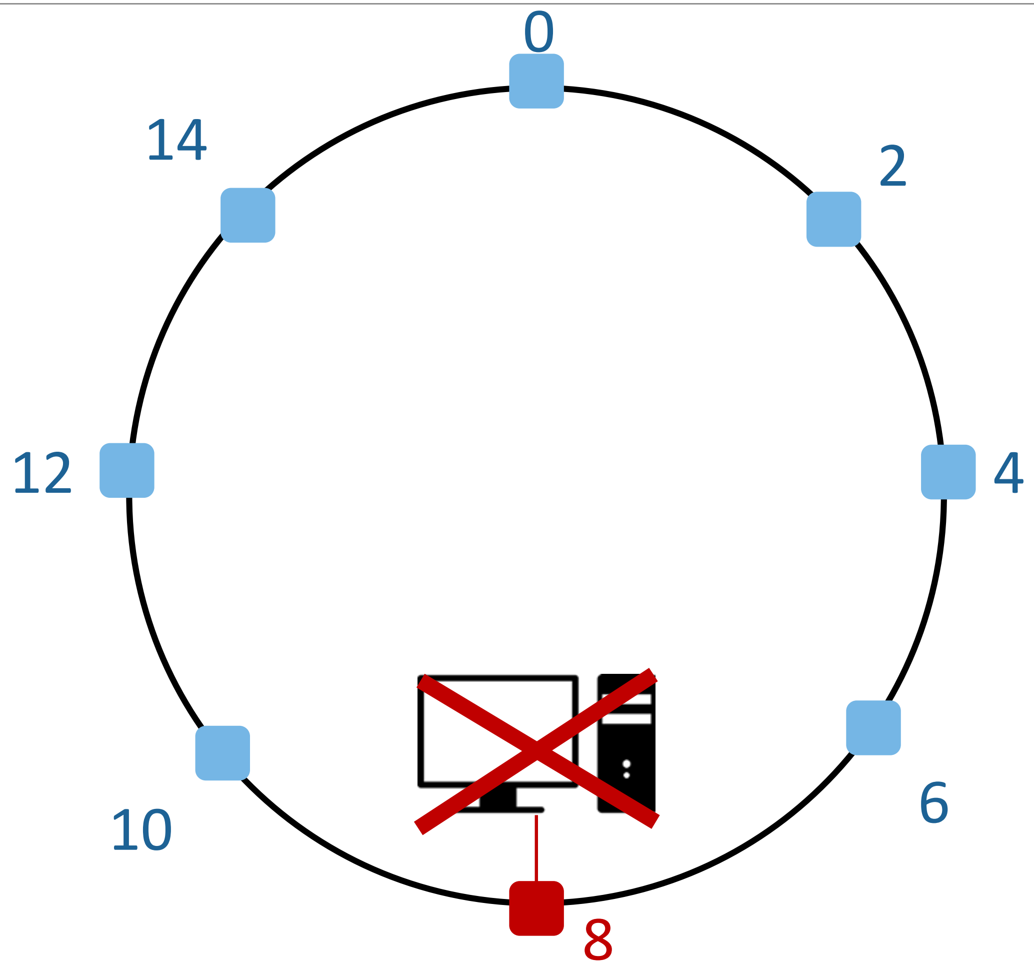


$m = 4, 2^m = 16$

Node Join

Finger Table for node 0

i	ident.	node
1	1	2
2	2	2
3	4	4
4	8	8



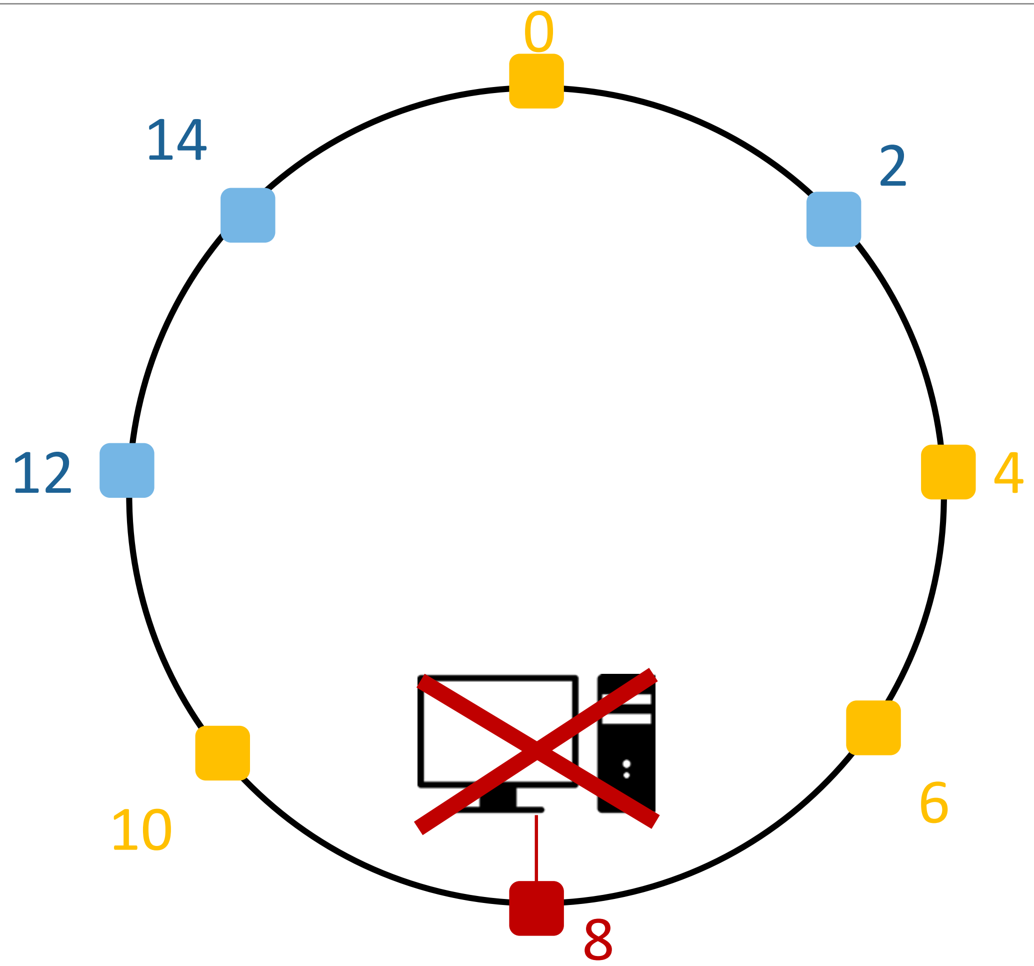
$m = 4$, $2^m = 16$

Which nodes have to update their tables?

Node Join

Finger Table for node 0

i	ident.	node
1	1	2
2	2	2
3	4	4
4	8	10



$m = 4$, $2^m = 16$

What can go wrong?

- We have a protocol that can reach any node from a given node in *O(log N)* steps with high probability
- Nodes can join and leave while maintaining our tables
- So what problems do we still need to address?

Concurrency and Failures

- It is necessary to handle concurrent joins and failures
- It cannot be assumed the operations described will be able to handle these conditions
- In particular, we want to maintain finger tables to be as accurate as possible to minimize searching cost

Stabilization

- A node informs neighboring nodes of its existence
- Iterative runs of stabilize will eventually result in a system with fully correct tables
- If *N* nodes are added to a system with *N* initial nodes concurrently, lookups will take *O(log N)* time with high probability

Fault Tolerance

- To handle faults, each node must maintain an additional list of replacement successors of size *r*

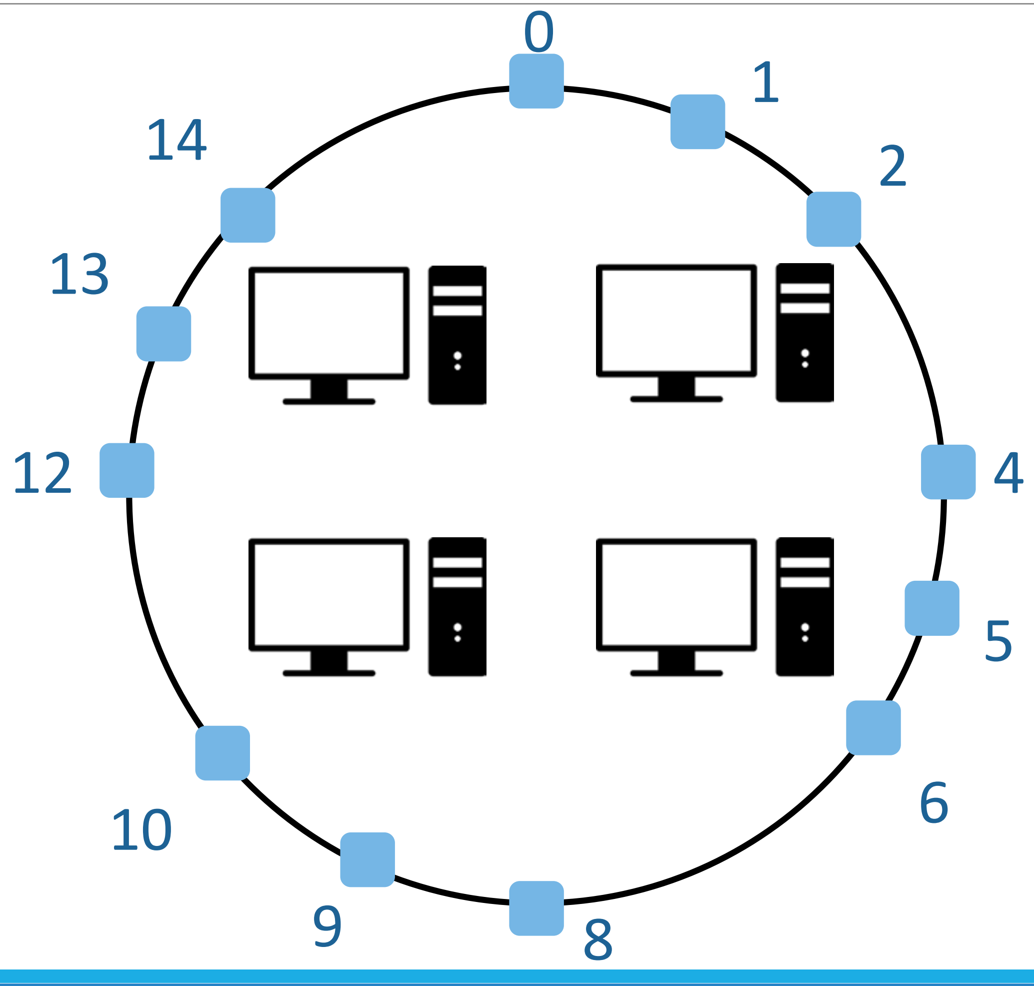
- As *r* increases, fault tolerance clearly increases, but memory costs increase linearly

- In particular, $r = O(\log N)$ allows for recovery with high probability if each node fails with probability ½

Virtual Nodes

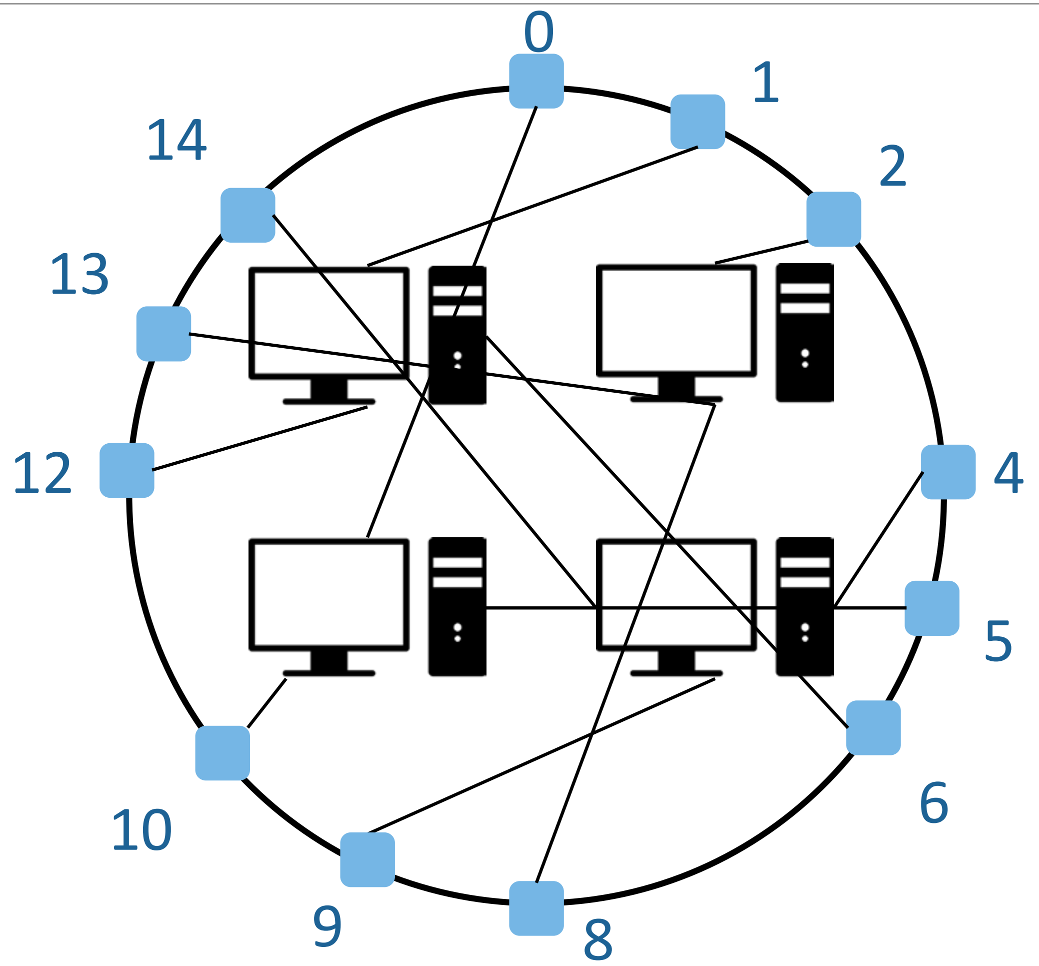
- The load balancing of this system can be improved
- How about we allow each member to house multiple ‘nodes’?

Virtual Nodes



$m = 4$, $2^m = 16$

Virtual Nodes

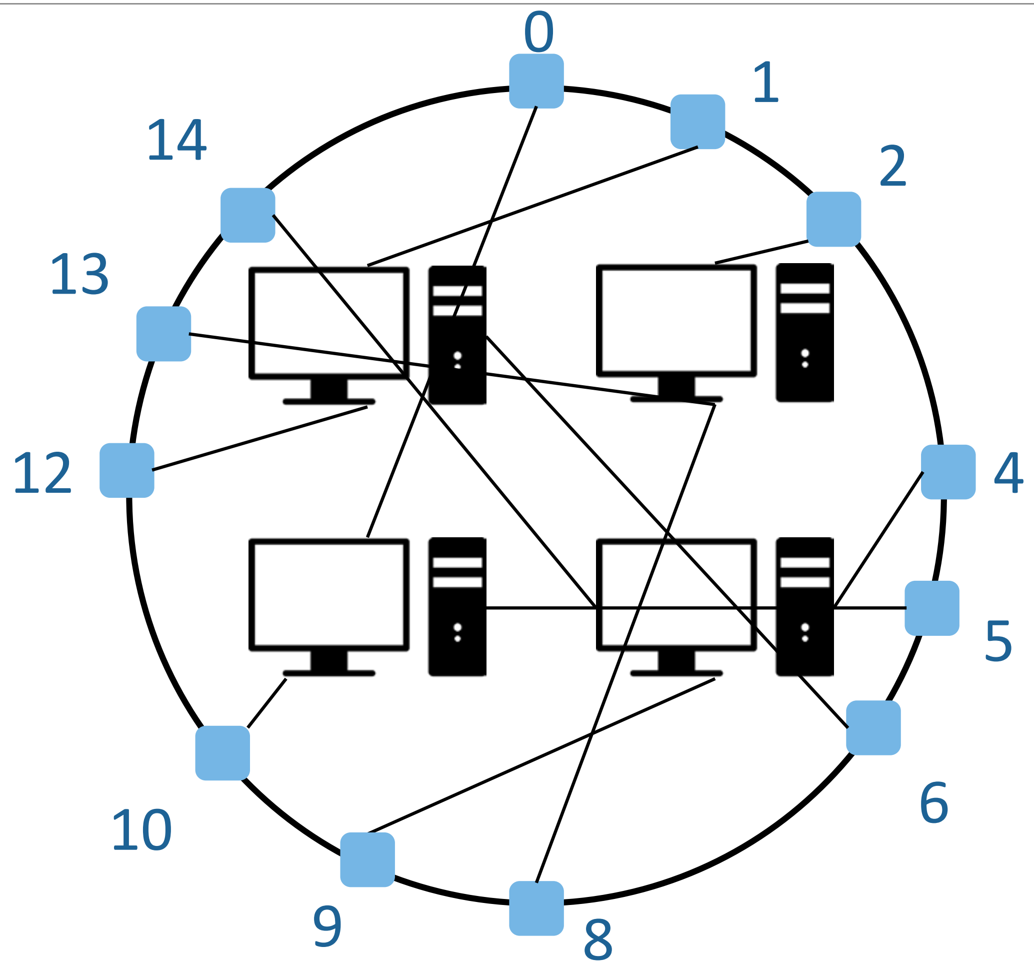


$m = 4,\ 2^m = 16$

Virtual Nodes

Finger Table for node 0

i	ident.	node
1	1	1
2	2	2
3	4	4
4	8	8



$m = 4, 2^m = 16$

Virtual Nodes

- Virtual nodes allow for improved load balancing
- The probability that a given node contains no keys is

$$(1-1/N)^N \approx e^{-1} \approx 0.368$$

- With $M < N$ virtual nodes, this probability is reduced to

$$1-(1-(1-1/N)^N)^M \approx 1-(1- 0.368)^M \approx 1-(0.632)^M$$

Results

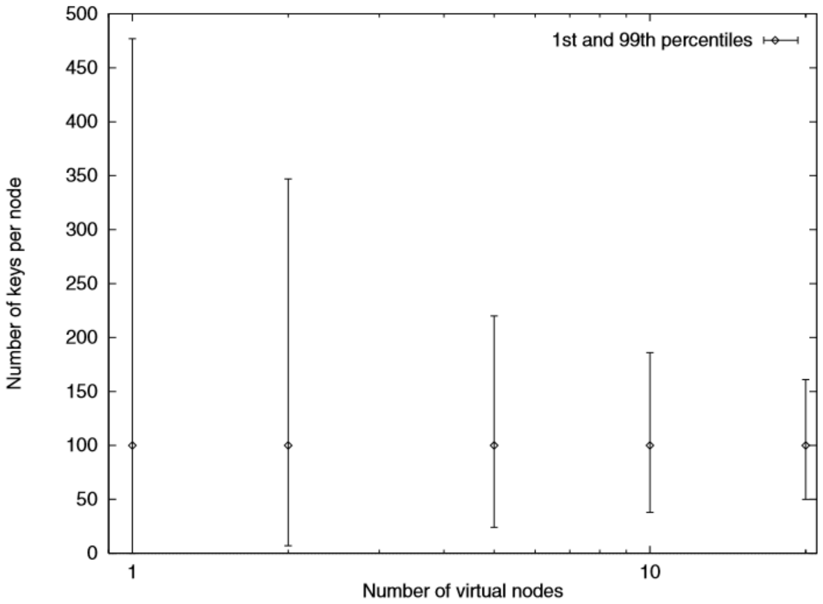
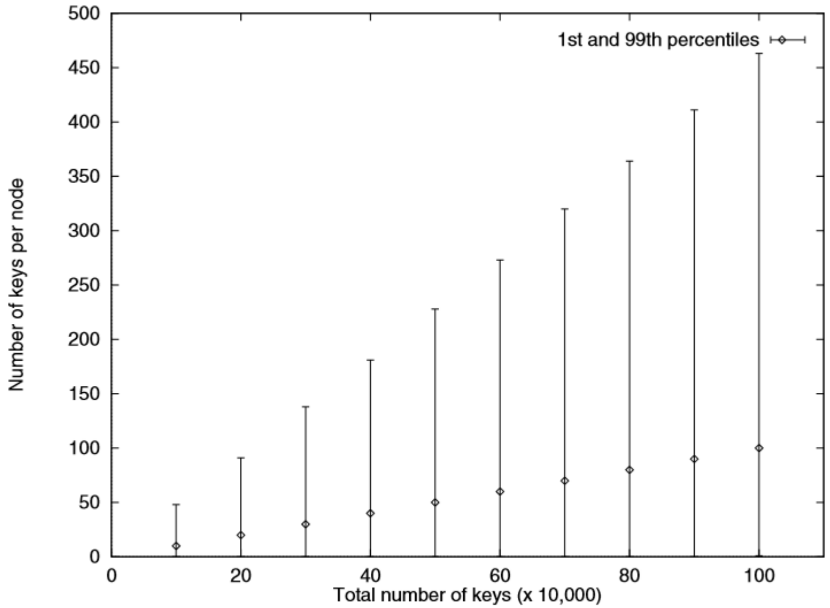
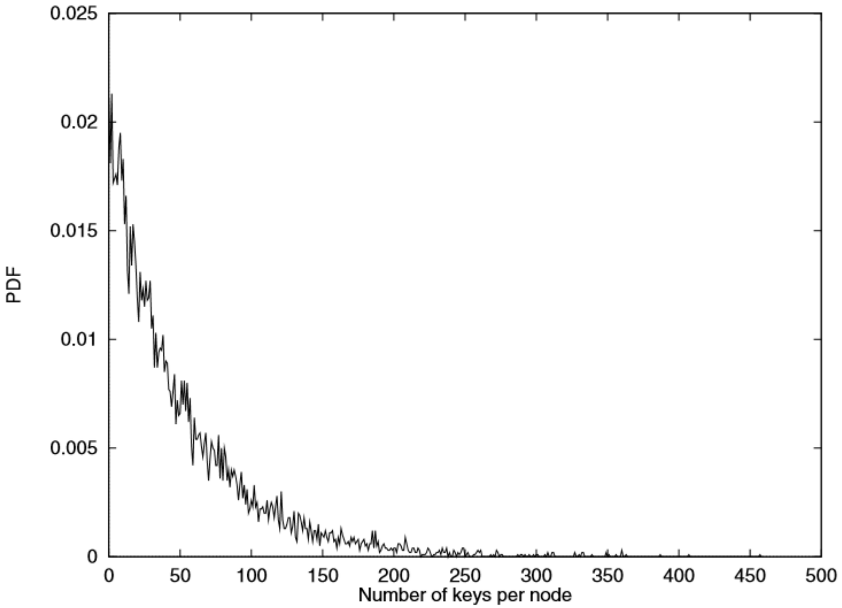


Figure 9: The 1st and the 99th percentiles of the number of keys per node as a function of virtual nodes mapped to a real node. The network has 10^4 real nodes and stores 10^6 keys.

Results



(a)



(b)

Figure 8: (a) The mean and 1st and 99th percentiles of the number of keys stored per node in a 10^4 node network. (b) The probability density function (PDF) of the number of keys per node. The total number of keys is 5×10^5.

Results

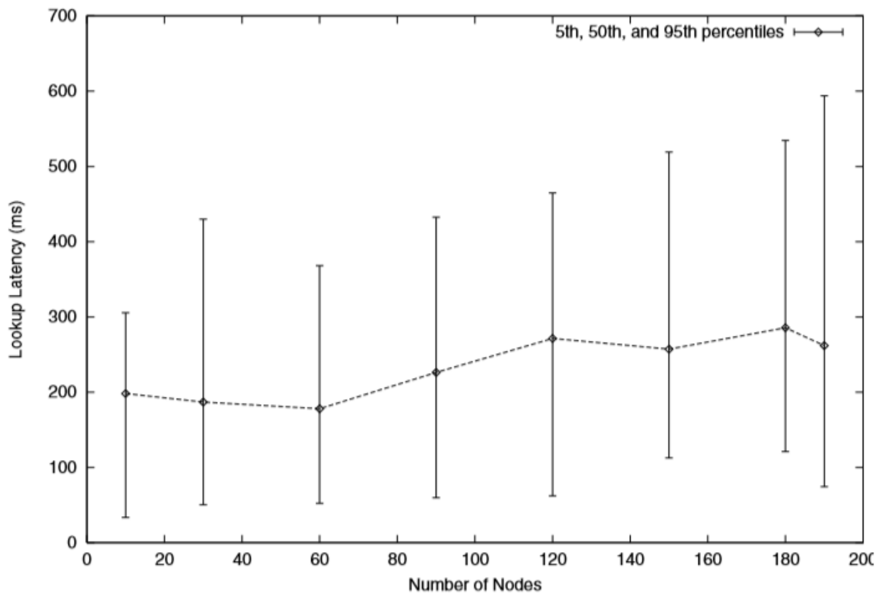


Figure 13: Lookup latency on the Internet prototype, as a function of the total number of nodes. Each of the ten physical sites runs multiple independent copies of the Chord node software.

Chord Goals

- How well did Chord achieve its goals?
  - Load Balancing
  - Decentralization
  - Scalability
  - Flexible Naming
  - Availability
- What are the problems with this protocol?

Distributed Hash Tables (Recall)

- Structure of a hash table – (key, value) pairs
- Member nodes contain local keys and values
- Nodes have information for retrieving ‘neighbor’ nodes in the form of keys
- Only a subset of all references are stored on a given node

Geometry (and why it matters)

- How can we compare a variety of DHT algorithms?

Geometry (and why it matters)

- How can we compare a variety of DHT algorithms?
- The structure of how they connect and search for nodes within the system seems like a good place to start
- The geometry of a DHT refers loosely to the connections between nodes in the DHT structure

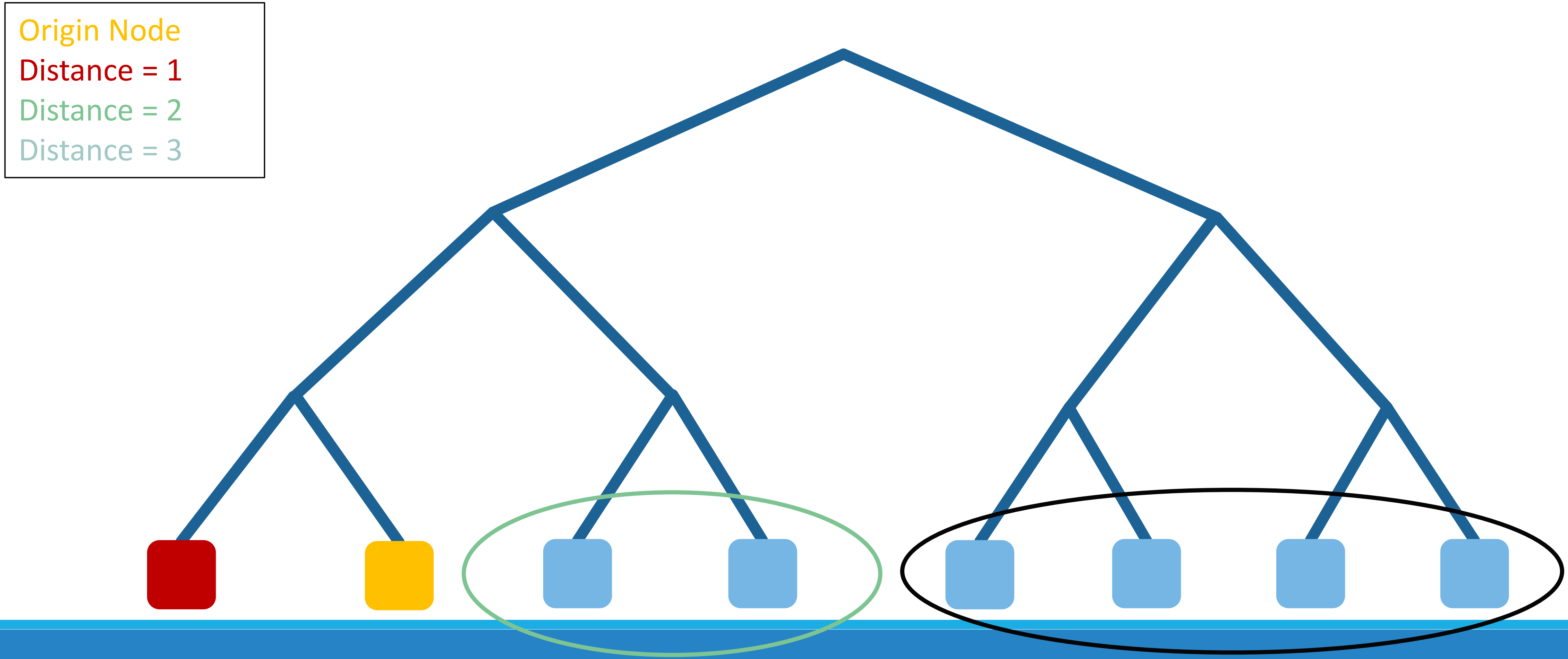
Outline of DHT Geometries

- Notable Geometries
- Theoretical Speed and Resilience Comparison
- Experimental Results

Notable Geometries

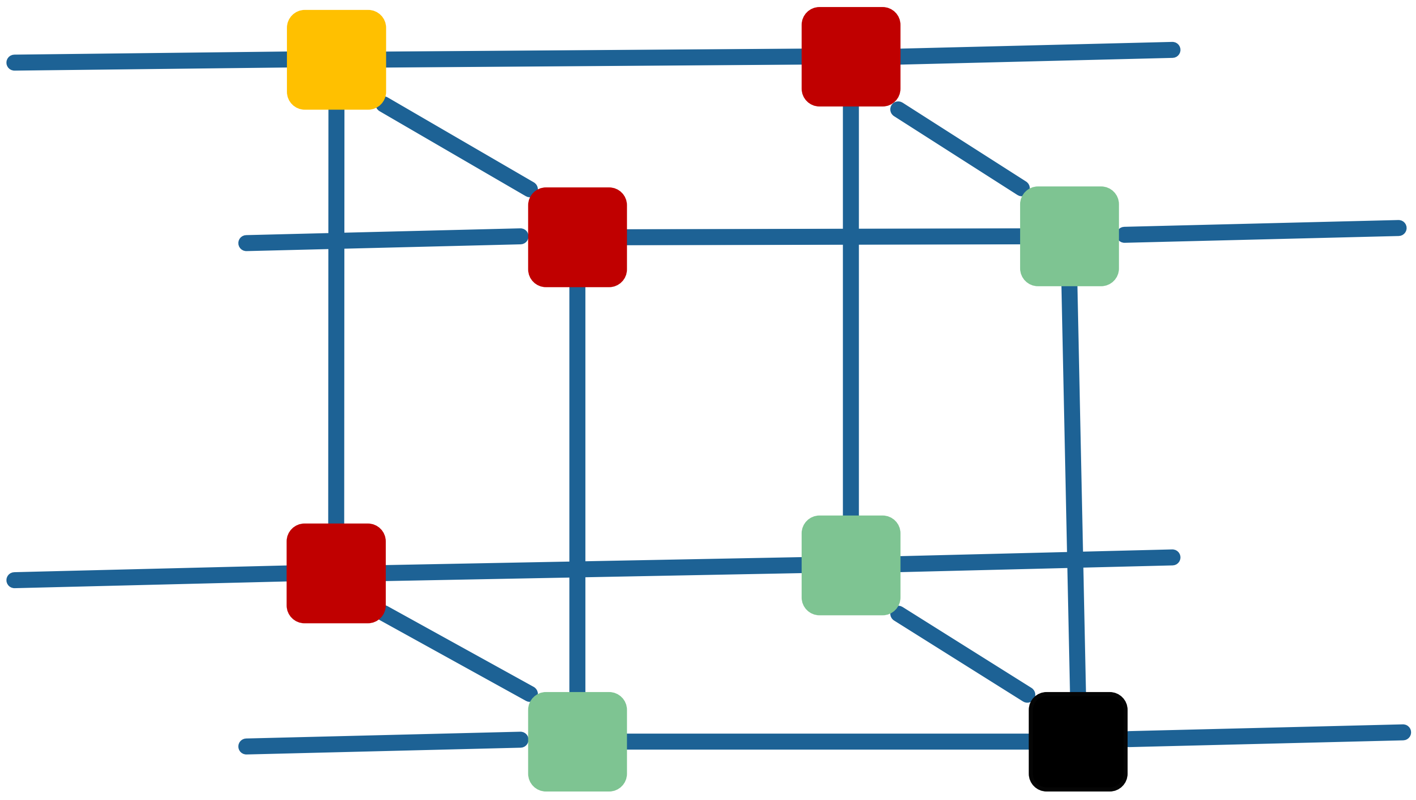
- Tree
- Hypercube
- Butterfly
- Ring (Chord!)
- XOR
- Hybrid

Tree Geometry



Hypercube Geometry

Origin Node
Distance = 1
Distance = 2
Distance = 3

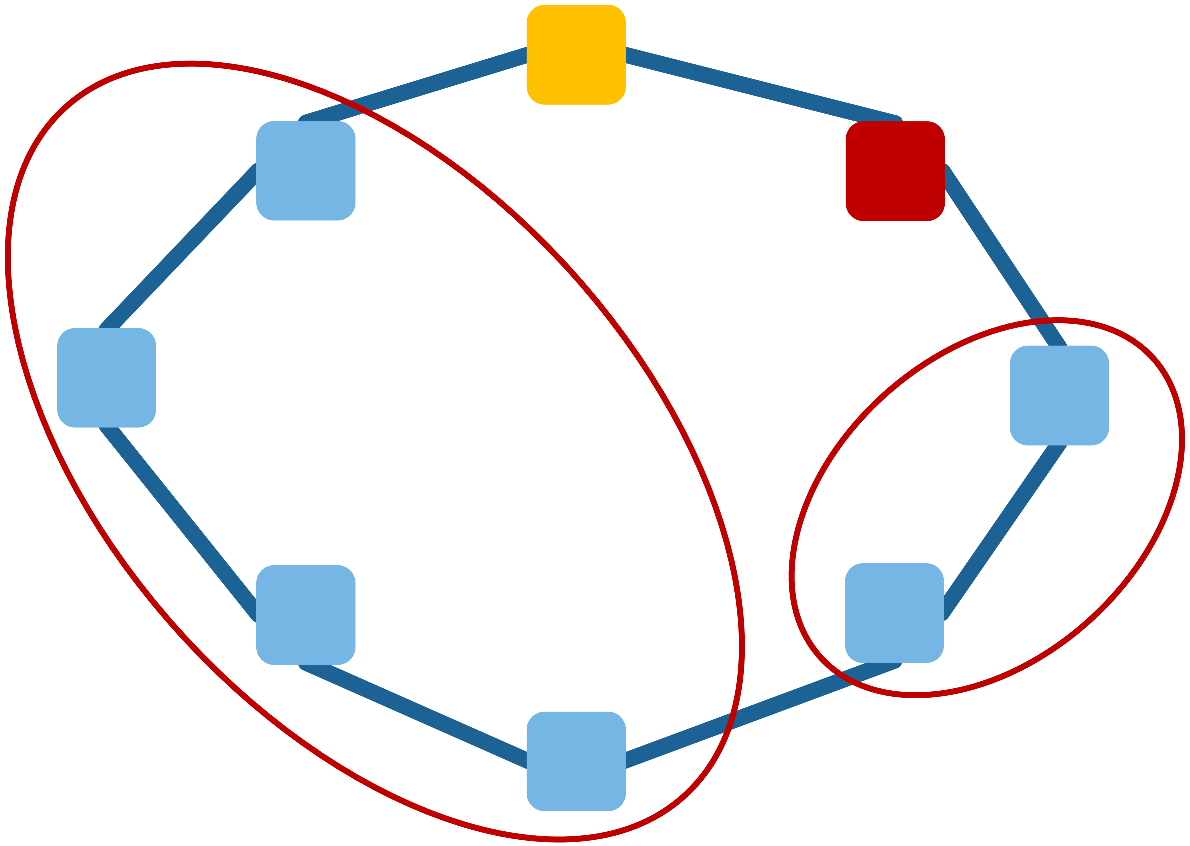


Butterfly Geometry

- Stages for each bit of the identifier
- *O(log n)* jumps for each stage
- Fast, but very little flexibility

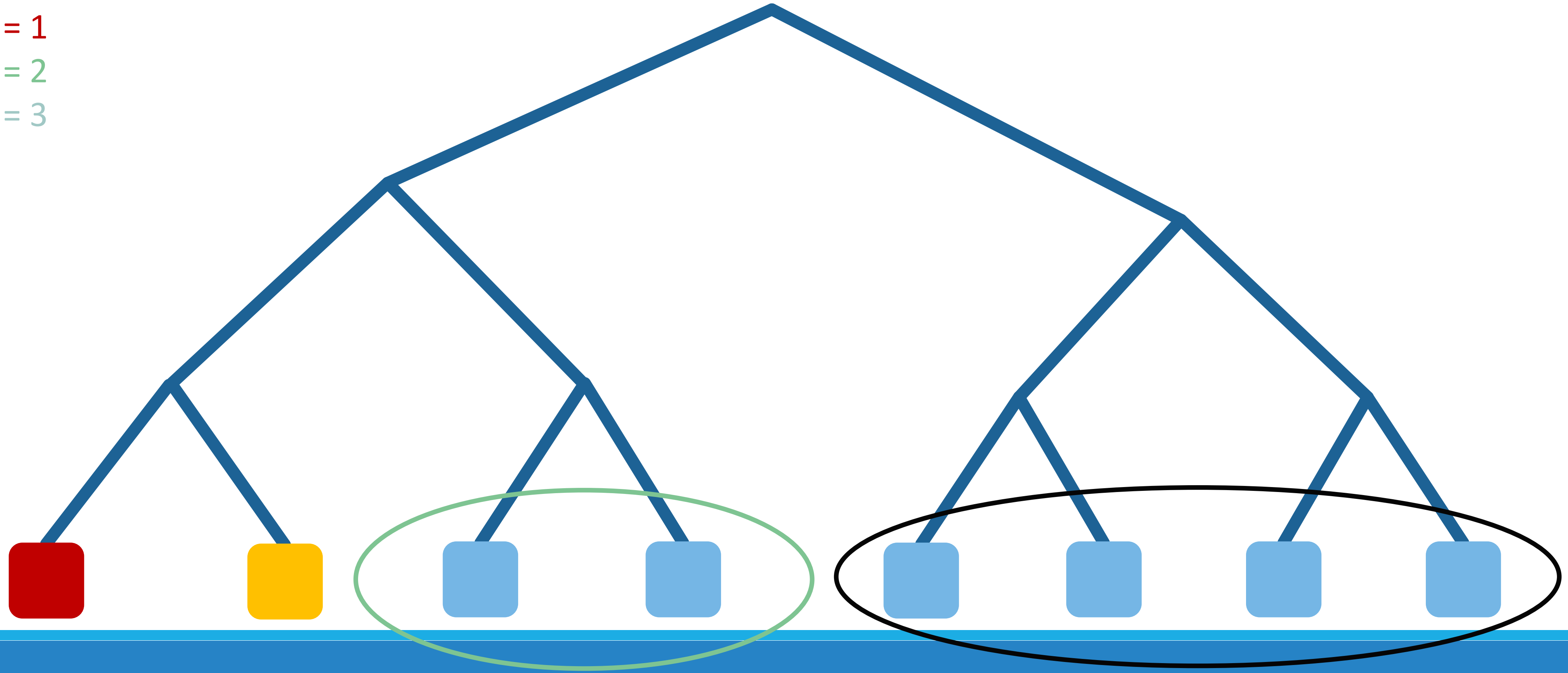
Ring Geometry

Origin Node
Distance = 1



XOR Geometry (Tree, but fault-tolerant)

Origin Node
Distance = 1
Distance = 2
Distance = 3



Hybrid Geometry

- Allows combining the strengths of different geometries
- Can be implemented by computing the distance between two nodes multiple ways
- More space intensive than a single geometry and search complexity can be subtle

Geometry Summary

	Tree	Hyper Cube	Butter fly	Ring	XOR
Neighbor selection (Speed)	✓		✓	✓	✓
Route selection (Resilience)		✓		✓	~

Results

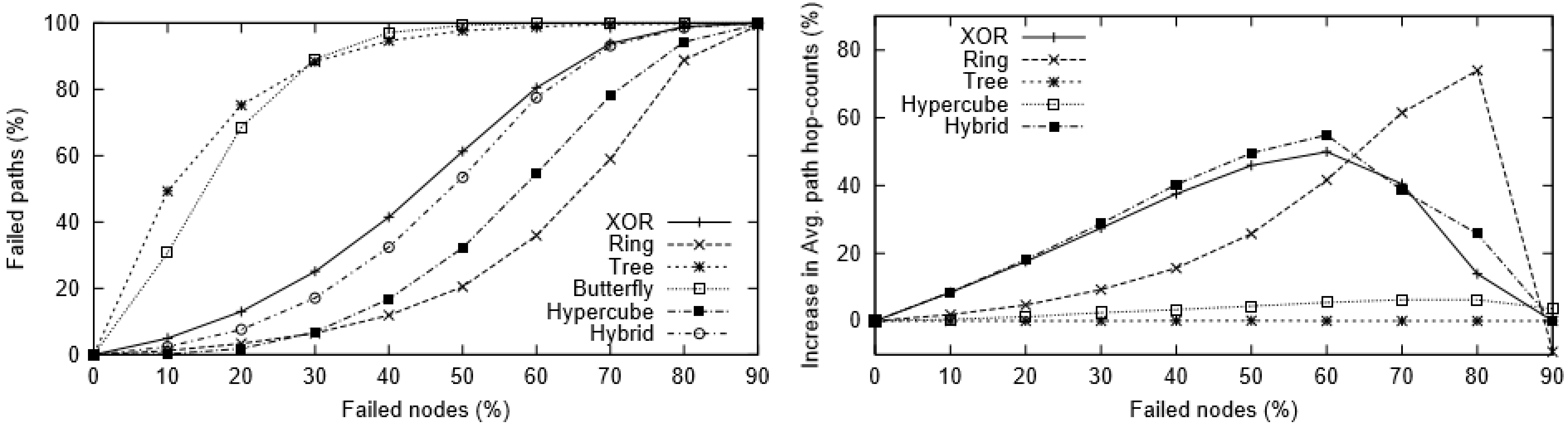


Figure 1: Left: Percentage of failed paths for varying percentages of node failures across different routing geometries. Right: Percent increase in average path hop-counts of successful paths for varying percentages of node failures across different routing geometries. The Butterfly is left off of this graph because so few routes are usable, and those that are sometimes take *shorter* paths than the original ones, resulting in a negative path stretch.

Results

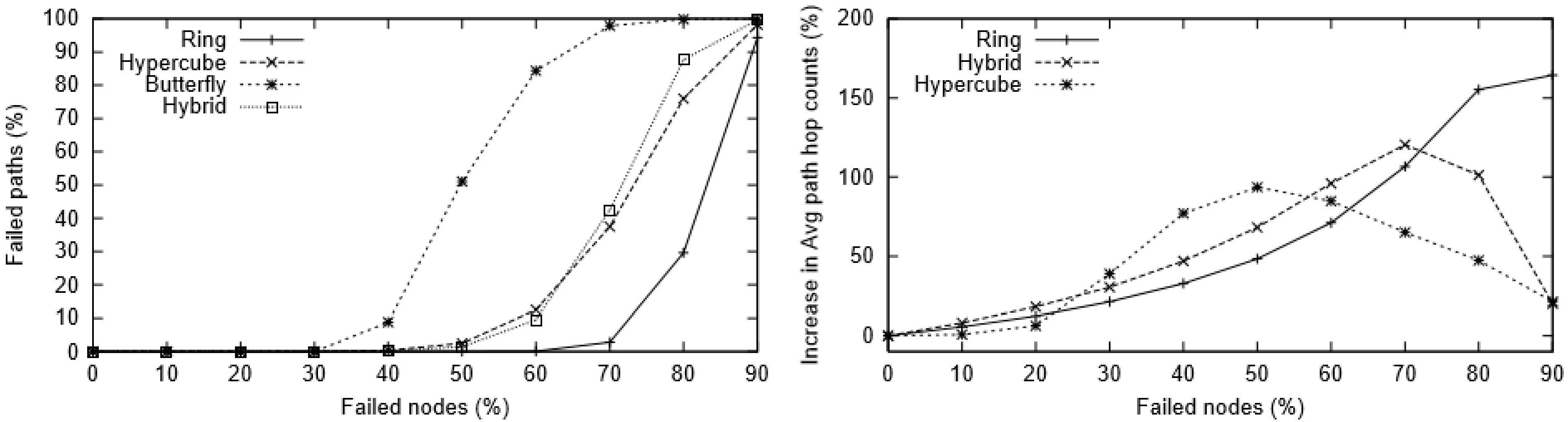


Figure 2: Left: Percentage of failed paths for varying percentages of node failures across different routing geometries. Right: Percent increase in average path hop-counts of successful paths for varying percentages of node failures across different routing geometries. Butterfly is left off of this graph because its path increase is so much higher than the others, reaching 700%, that it would distort the y-axis. All algorithms use 16 sequential neighbors.

Fault-Tolerance

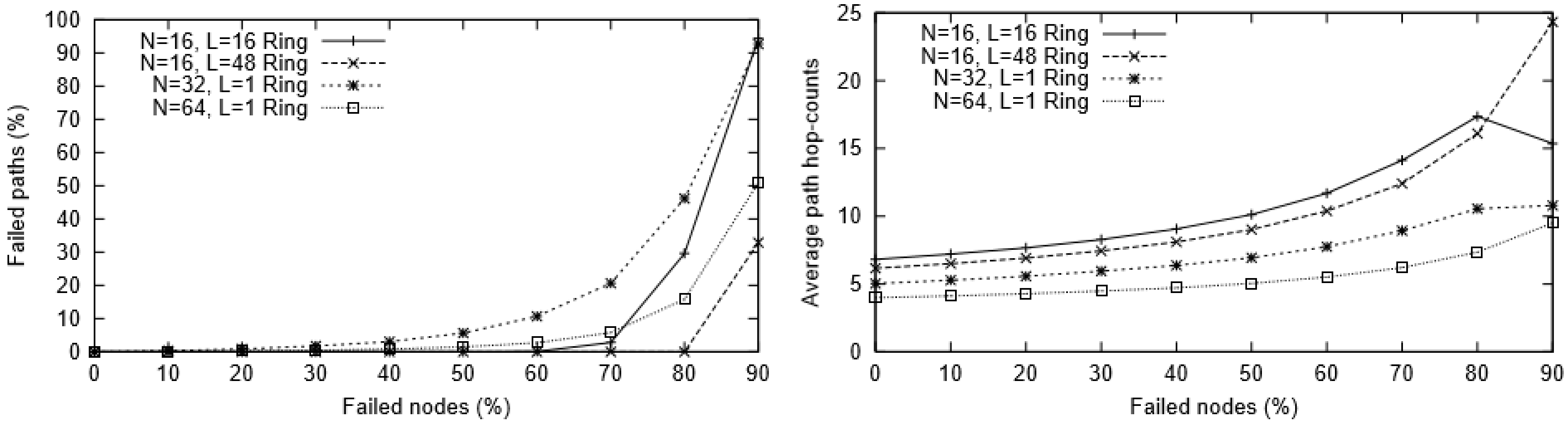
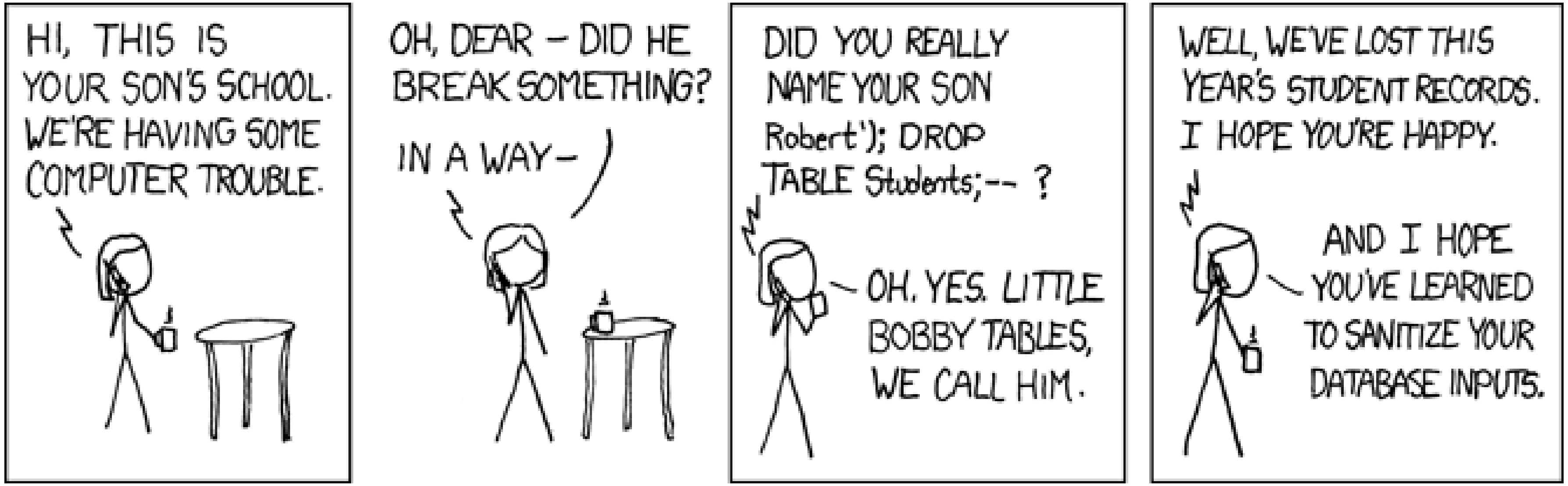


Figure 3: Left: Percentage of failed paths for varying percentages of node failures for a Ring geometry for varying numbers of neighbors (N) and sequential neighbors (L). Right: Average path hop-counts of the successful paths for varying percentages of node failures.

NOTE

At this point, I think I'll be out of time; I can add slides on PNS/PRS, Convergence, or the Discussion section if you think they're relevant

Questions



https://xkcd.com/327/