Chord and the DHT Routing Geometry

Yucheng Lu
Nov 19

Overview

History of peer-to-peer systems.

From unstructured networks to structured networks.

Routing geometry of structured networks.

Chord:

- The Base Protocol: Routing and node joins.
- Concurrency and Failures.

Conclusions

What is a peer-to-peer (P2P) system?

Peer-to-peer systems and applications are distributed systems without any centralized control or hierarchical organization, where the software running at each node is equivalent in functionality.

Napster: a P2P music file sharing platform

[1999.6] Shawn releases Napster online.

[1999.12] RIAA sues Napster.

[2001.3] US Federal Appeals Court: Users of Napster are violating copyright law!

[2001.9] Napster started running paid service.

[Now] Napster open source.

Shawn Fanning

How Napster works

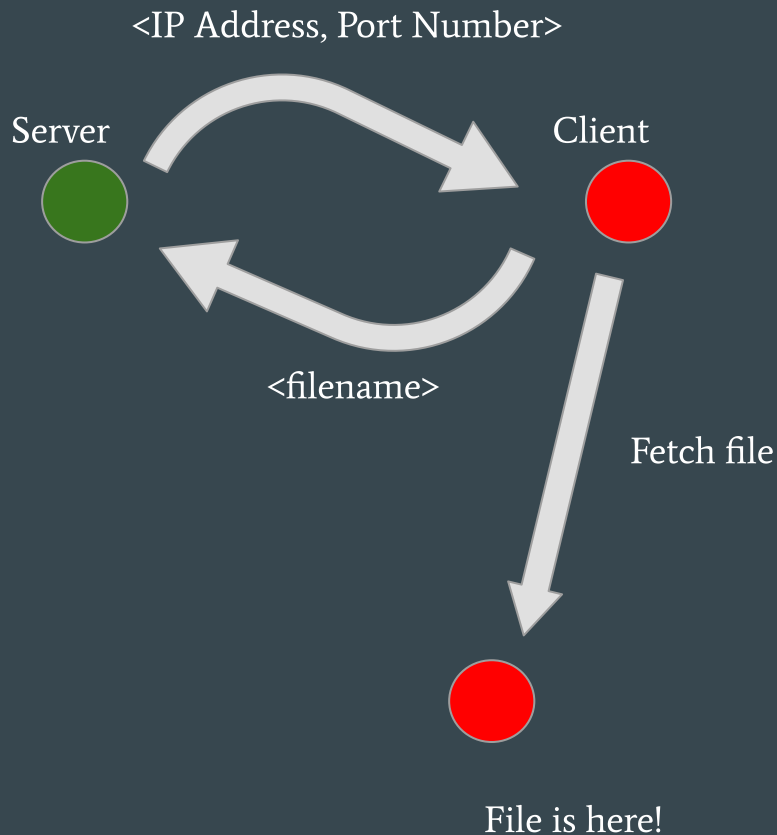
Client-Server structure.

Server stores no files but a list of <filename, IP Address, Port Number> tuples.

Client queries server <filename>.

Server returns a list of hosts to client: <IP Address, Port Number>.

Client pings each hosts and download file from best host.



After Napster

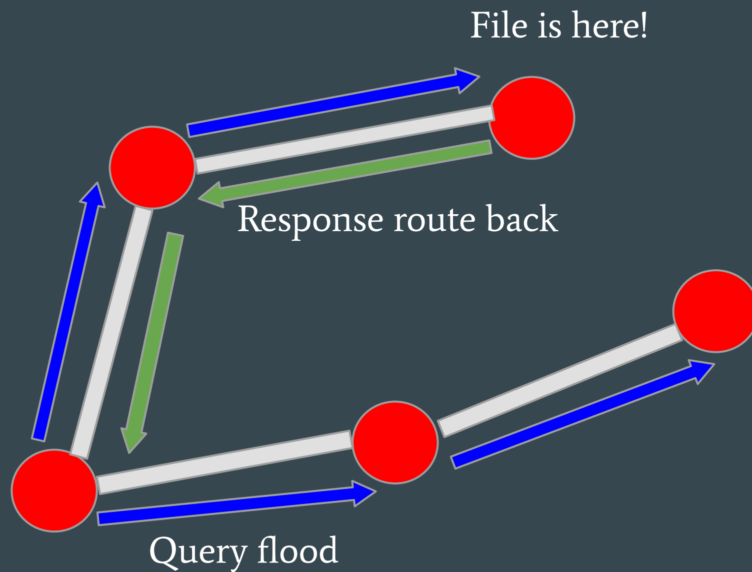
Later, more decentralized projects followed Napster's P2P file-sharing example, such as Gnutella, Freenet, BearShare and Soulseek.

Gnutella: Remove central server

No server! Nodes are connected via an overlay graph.

Queries flood out with TTL restricted.

Once find file, reverse path routed.



From unstructured networks to structured networks

<u>Unstructured networks</u>:

The overlay networks does not have fixed structure (Napster, Gnutella, Gossip). Perform keyword search.

Pros:

Easy to build.

Robust to “Churn” -- Large number of nodes leaving or joining.

Cons:

Flooding queries cause substantial overhead and not guaranteed response.

From unstructured networks to structured networks

Unstructured networks:

The overlay networks does not have fixed structure (Napster, Gnutella, Gossip). Perform keyword search.

Pros:

Easy to build.

Robust to "Churn" -- Large number of nodes leaving or joining.

Cons:

Flooding queries cause substantial overhead and not guaranteed response.

Structured networks:

The overlay is organized in to a specific topology. Perform exact Match.

Pros:

Efficient search files via queries.

Guaranteed response.

Cons:

Less robust to Churns.

Distributed Hash Table!

Distributed Hash Table (DHT)

Structure of DHT:

An abstract keyspace, such as the set of 160-bit strings.

A keyspace partitioning scheme splits ownership of this keyspace.

An overlay network connects the nodes, allowing them to find the owner of any given key in the keyspace.

Many Proposals for DHT

- Tapestry (UCB)
- Pastry (MSR, Rice)
- Chord (MIT, UCB)
- CAN (UCB, ICSI)
- Viceroy (Technion)
- Kademlia (NYU)
- Kelips (Cornell)

- Symphony (Stanford)
- Tangle (UCB)
- SkipNet (MSR,UW)
- Bamboo (UCB)
- Hieras (U.Cinn)
- Sprout (Stanford)
- Calot (Rochester)

- 1hop (MIT)
- conChord (MIT)
- Apocrypha (Stanford)
- LAND (Hebrew Univ.)
- ODRI (TexasA&M)

- Koorde (MIT)
- JXTA's (Sun)

(This slides is taken from CMU 15-744 2016 Fall Lecture Notes)

What makes a “good” DHT?

Flexibility: The algorithmic freedom left after the basic routing geometry has been chosen.

Flexibility in Neighbor Selection: freedom to choose neighbors based on other criteria in addition to identifiers.

Flexibility in Route Selection: When route is down, other options for the next hop.

Static Resilience: How well a DHT can route before routing state is restored due to node failures.

Comparison of DHTs

Geometry	Algorithm	Neighbor Selection	Route Selection (log(N) hops)	Route Selection (> log(N) hops)
Tree	Plaxton	$N^{log(N)/2}$	1	0
Hypercube	CAN	1	$c_1(log(N))$	0
Butterfly	Viceroy	1	1	0
Hybrid	Tapestry, Pastry	$N^{log(N)/2}$	1	$c_2(log(N))$
XOR	Kademila	$N^{log(N)/2}$	1	$c_2(log(N))$
Ring	Chord	$N^{log(N)/2}$	$c_1(log(N))$	$2c_2(log(N))$

Static Resilience: Tree ≒ Butterfly < XOR ≒ Hybrid < Hypercube < Ring

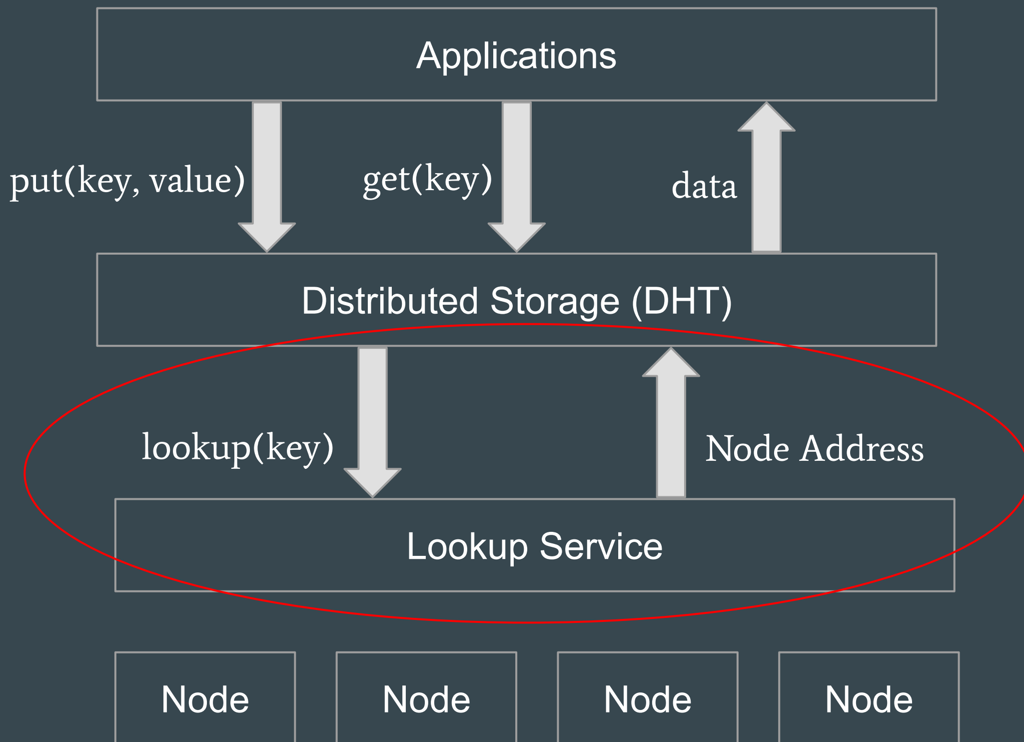
Now we've seen...

Some early P2P system

Some structured networks

Distributed Hash Table

How to find data in a structured network?



A fundamental Question: How to locate data?

Design an algorithm "**A**" such that:

<node_id> = A(<key>)

Given a key, maps the key to the node that stores the data item.

Two questions need to be addressed:

I. How to assign keys to nodes?
II. How to route a query based on the assignment?

A fundamental Question: How to locate data?

Design an algorithm "**A**" such that:

<node_id> = A(<key>)

Given a key, maps the key to the node that stores the data item.

Two questions need to be addressed:

I. How to assign keys to nodes?
II. How to route a query based on the assignment?

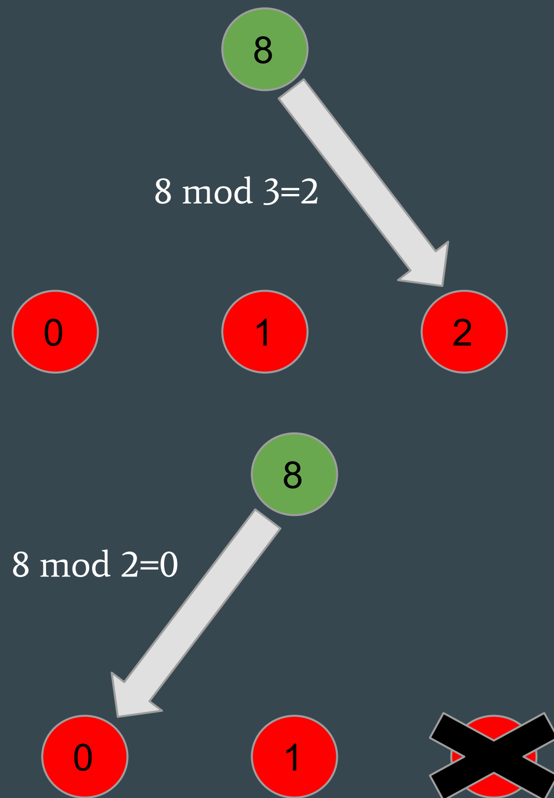
Basic Hashing?

Given an object, hash it to a bin from a set of N bins.

Objects are evenly distributed to the bins.

But...

If N changes, all objects need to be reassigned.

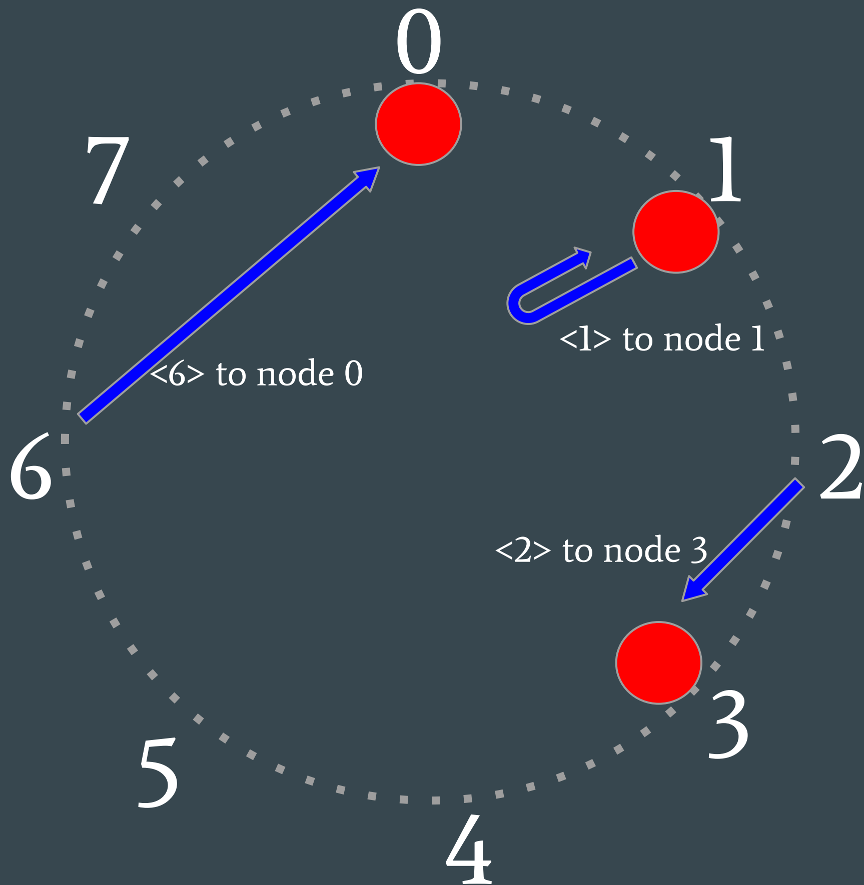


Consistent Hashing

Assign each node and key a m-bit identifier using a hash function. (e.g. SHA-1)

Assign keys to nodes:

I. Identifiers are ordered in an identifier circle modulo 2^m.
II. Key <k> is assigned to the first node whose identifier is equal to or follows <k> in the identifier space, called successor node.
III. Successor node of <k> is the first node clockwise from <k> in a ring case.



What's nice about consistent hashing?

Theorem 1. For any set of N nodes and K keys, with high probability:

1. Each node is responsible for at most $(1+\varepsilon)K/N$ keys.
2. 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.)

Does it fix trivial hashing?

When N changes, only $O(K/N)$ objects need to be changed.

A fundamental Question: How to locate data?

Design an algorithm "**A**" such that:

<node_id> = A(<key>)

Given a key, maps the key to the node that stores the data item.

Two questions need to be addressed:

I. How to assign keys to nodes?
II. How to route a query based on the assignment? (How to lookup?)

How did people solve it?

I. DNS: direct mapping, relies on a set of special servers.
II. Freenet: search for cached copies, does not guarantee retrieval existing files.
III. Ohaha: uses consistent hashing, does not guarantee retrieval existing files.
IV. Plaxton: a prefix-based routing protocol, complicated.

Can we do better?

Can we do better?
Chord!

Authors of Chord

Ion Stoica

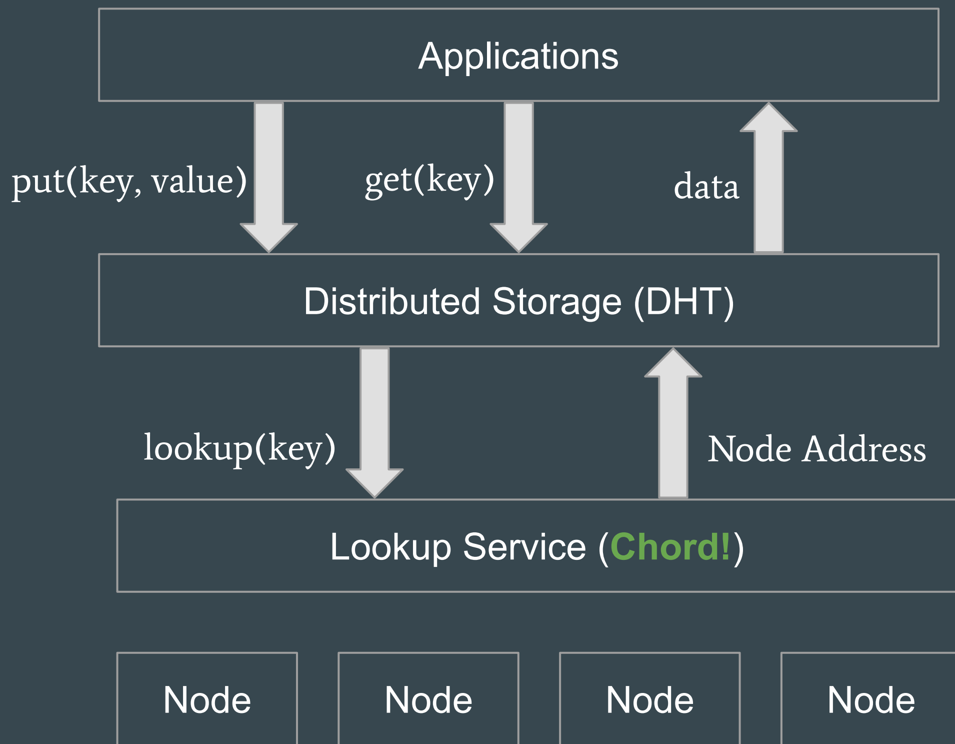
Robert Morris

David Karger

Frans Kaashoek

Hari Balakrishnan

What is Chord trying to solve?



Chord Properties

Load balance:

Evenly spread keys over nodes.

Decentralization:

Nodes are equally important.

Scalability:

Cost of lookup grows as the log of number of nodes.

Chord Properties

Availability:

Automatically updates lookup tables as nodes join, leave or fail. (make sure the node responsible for a key can always be found.)

Flexible naming:

No constraints on the structure of the keys.

Chord needs to address

How to find the locations of keys? (Routing)

Consistent Hashing? Not good enough due to large routing information.

How new nodes join the system? (Joins and Leaves)

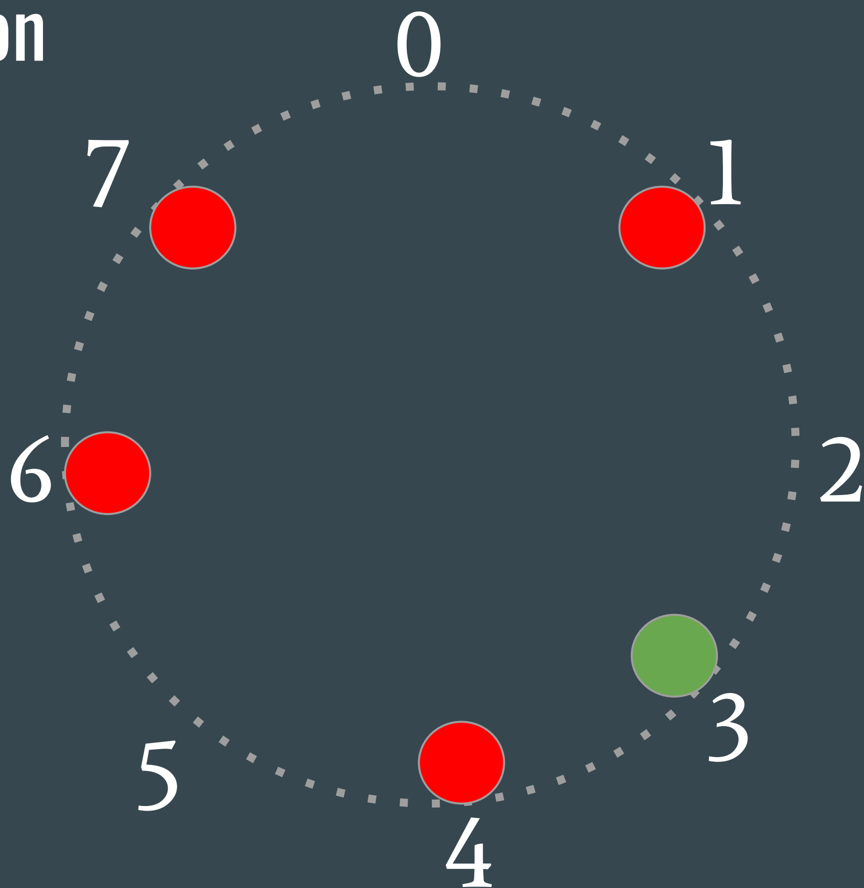
How to support concurrent operations? (Concurrency)

How to recover from the failure of existing nodes? (Failures)

Challenge 1: Routing Information

A trivial approach: Each node needs only be aware of its successor node on the circle.

Problem: It may require traversing all N nodes to find the target.



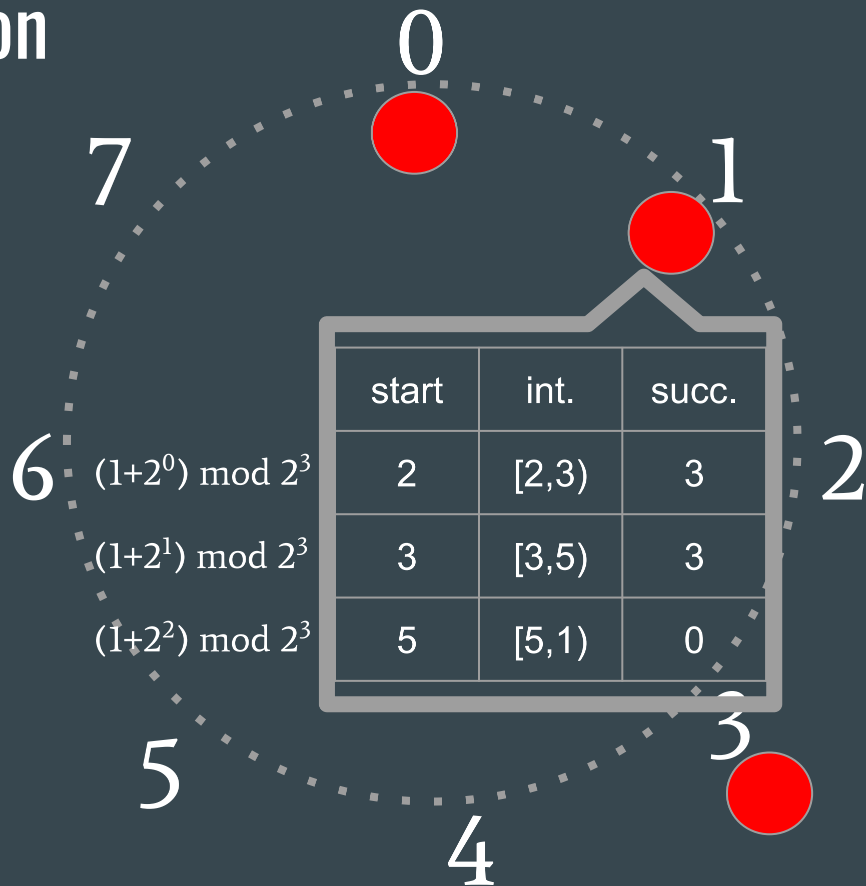
Challenge 1: Routing Information

Chord let nodes maintain additional information:

Finger table: Each node, n, maintains a routing table with at most m (length of identifiers) entries.

I-th finger: i-th entry in that table:

$$S = \text{successor}(n+2^{i-1})$$

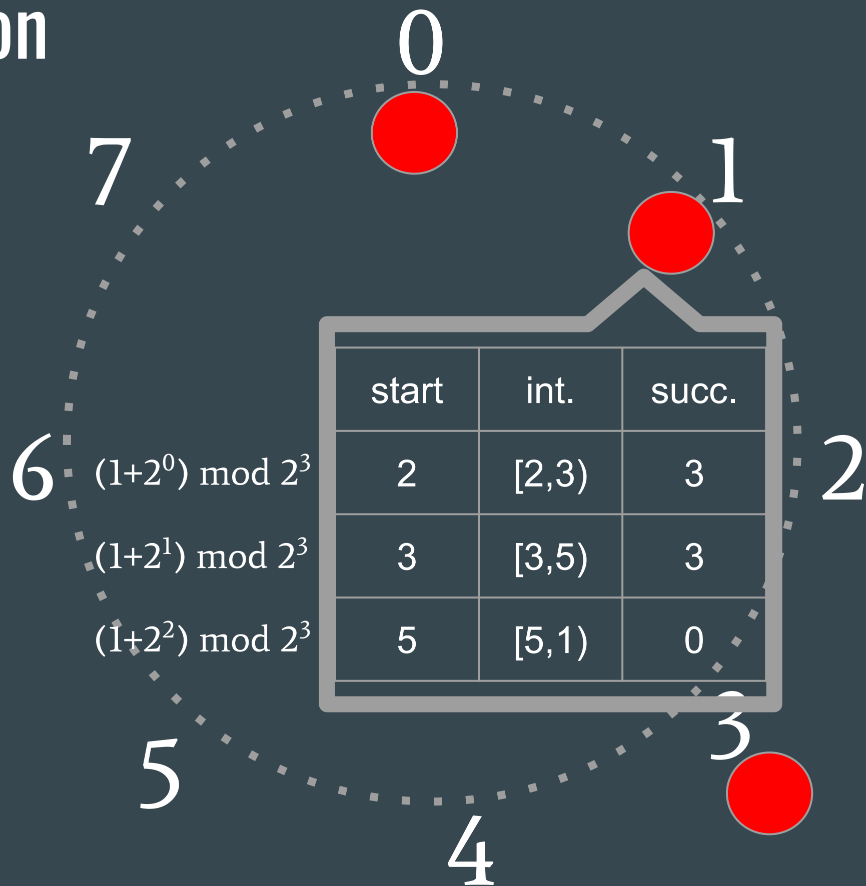


Challenge 1: Routing Information

Two important characteristics of finger tables:

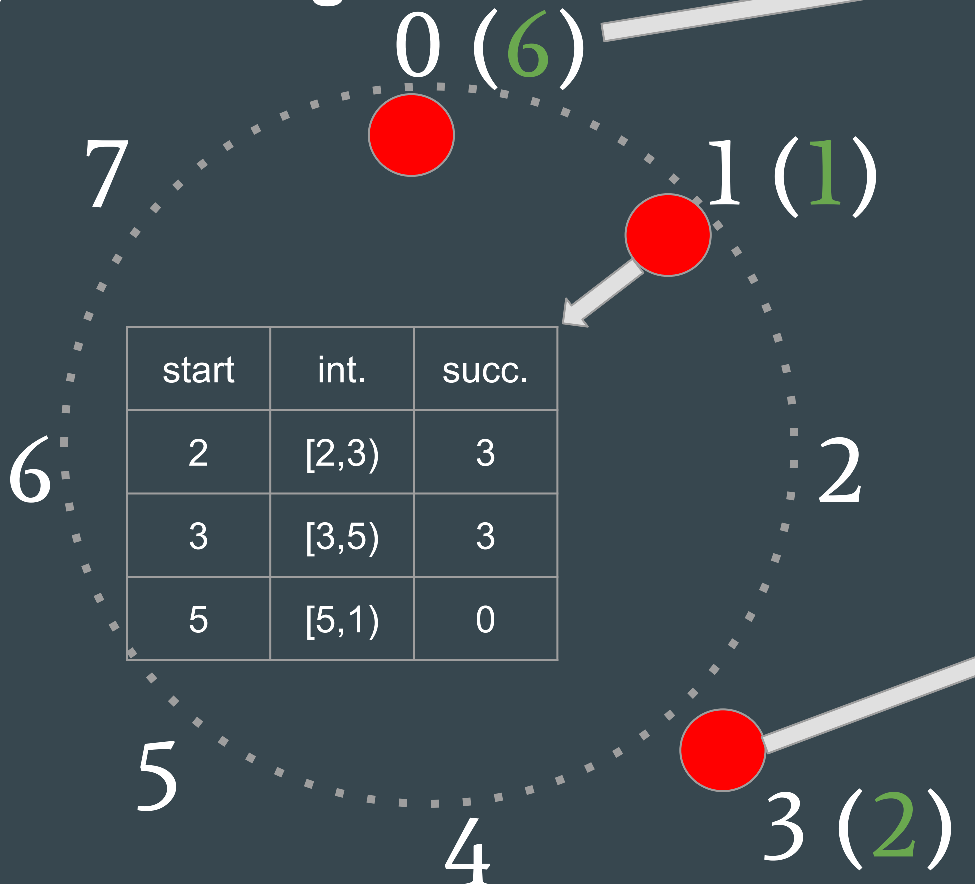
1. Each node stores information about only a small number of other nodes.
2. A node's finger table generally does not contain enough information to determine the successor of an arbitrary key.

How to solve 2? Recursively do that!



Challenge 1: Routing Information

Example: 3 wants to find location of key 1.



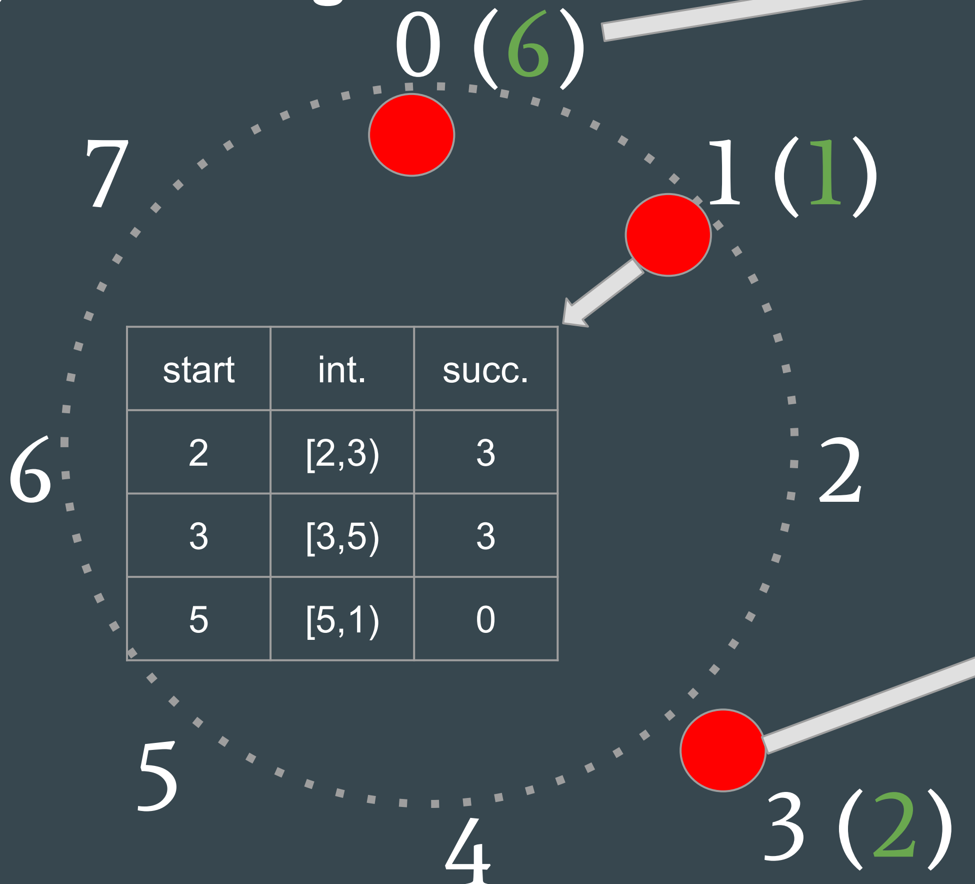
start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Challenge 1: Routing Information

Example: 3 wants to find location of key 1.



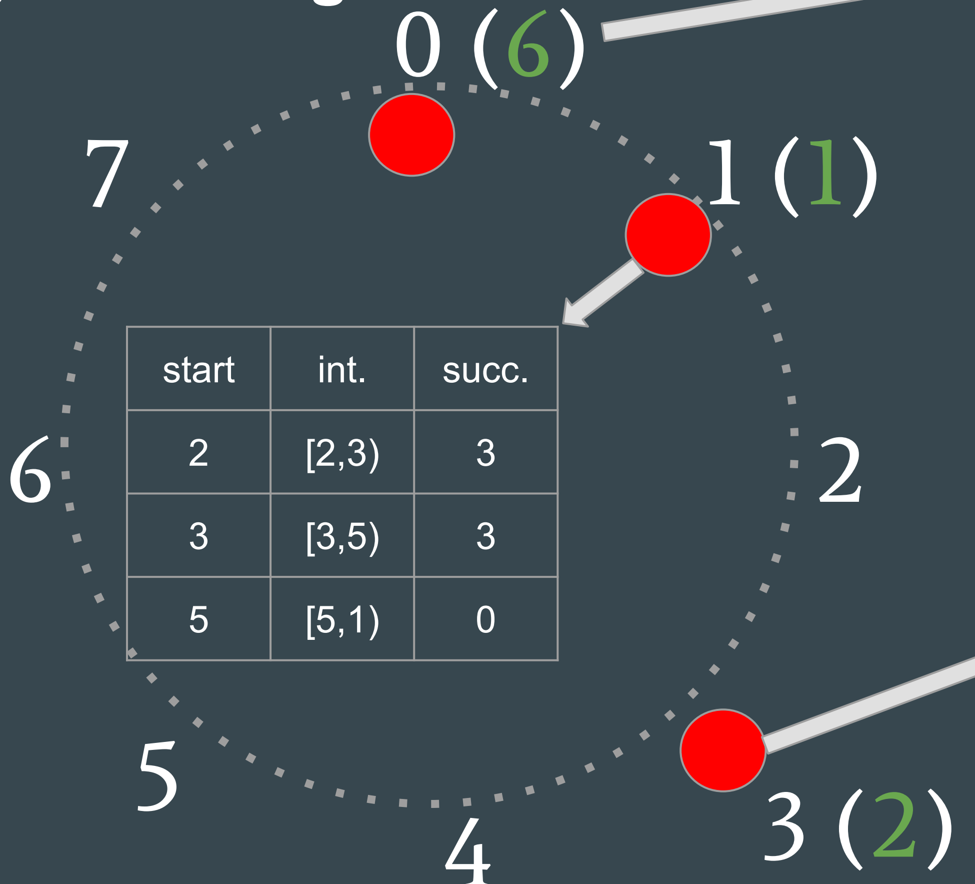
start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0

start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Challenge 1: Routing Information

Example: 3 wants to find location of key 1.



start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Challenge 1: Routing Information

Intuition behind finger table lookup:

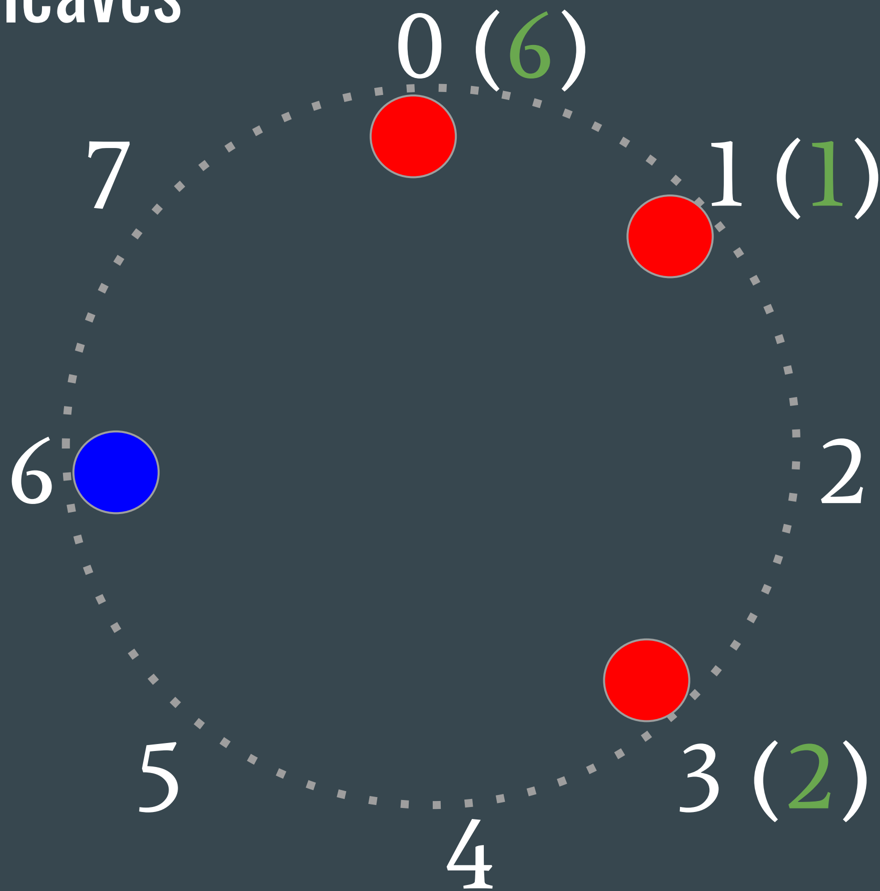
The finger pointers at repeatedly doubling distances around the circle cause each iteration to halve the distance to the target identifier.

Is it guaranteed theoretically?

Theorem 2. With high probability, the number of nodes that must be contacted to find a successor in an N-node network is $O(\log N)$.

Challenge 2: Node Joins and leaves

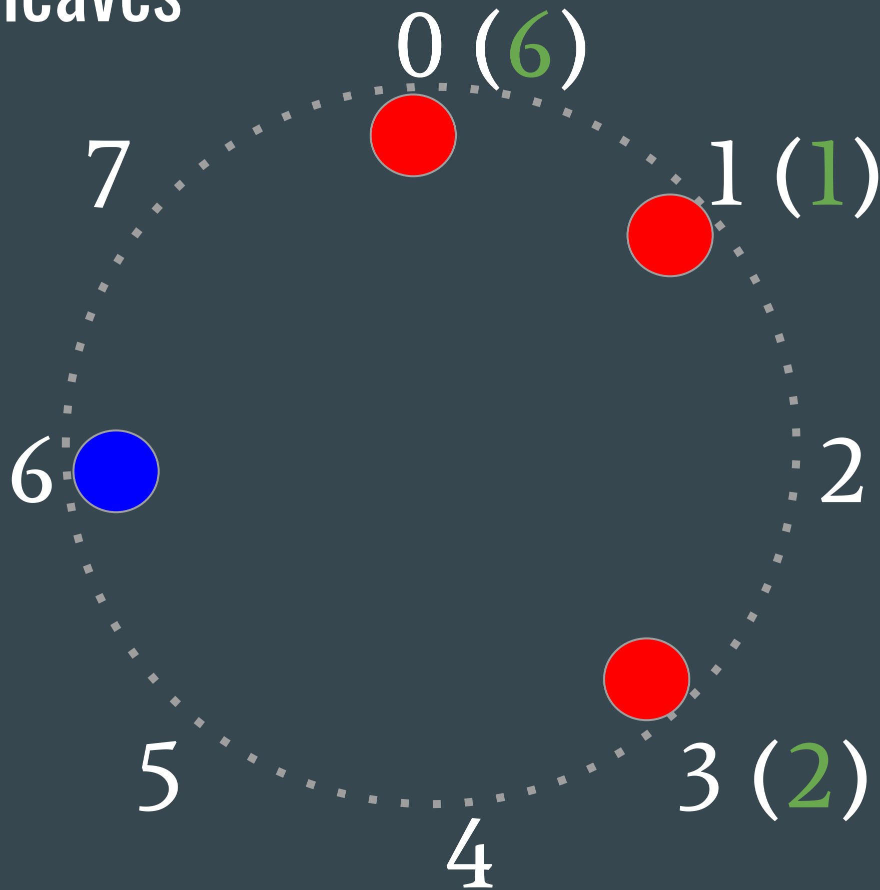
In a dynamic network, nodes can join (and leave) at any time. The main challenge in implementing these operations is preserving the ability to locate every key in the network.



Challenge 2: Node Joins and leaves

Two invariants need to be preserved:

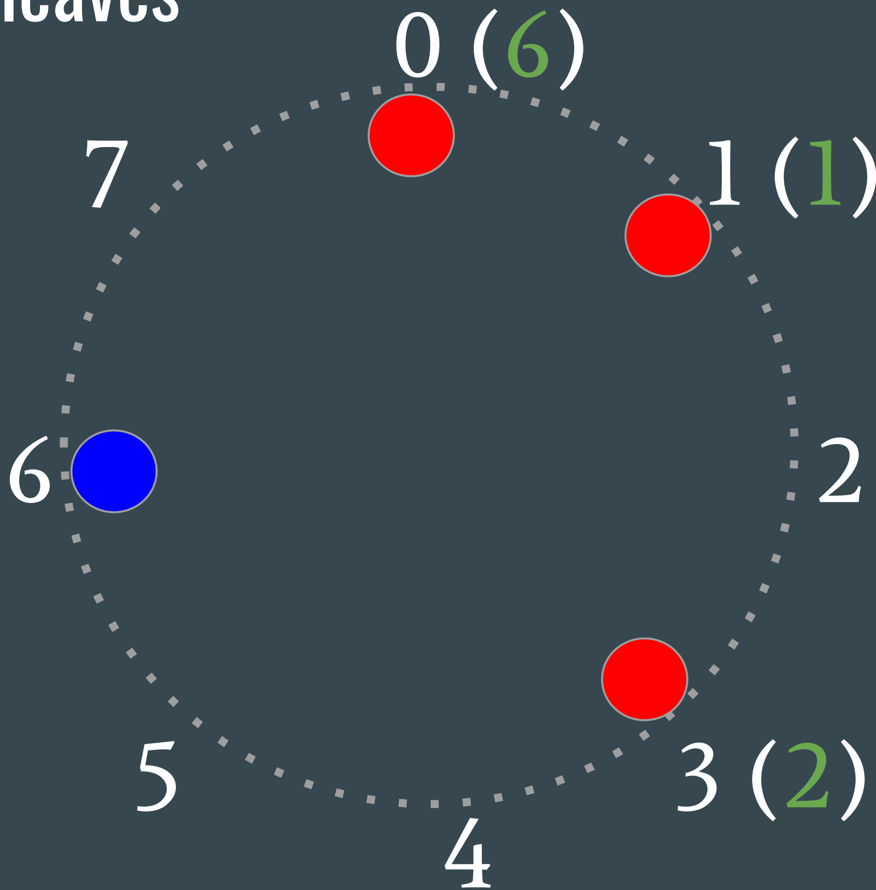
1. Each node's successor is correctly maintained.
2. For every key k, node successor(k) is responsible for k.



Challenge 2: Node Joins and leaves

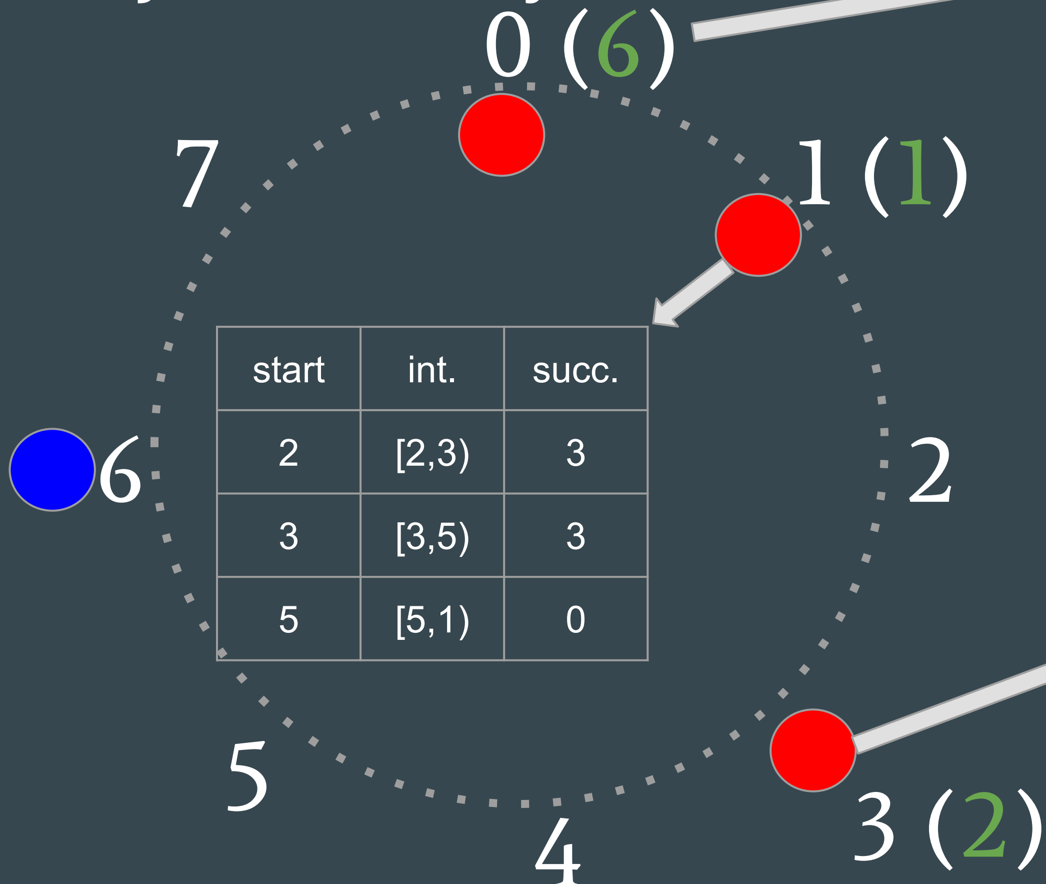
Chord needs to:

1. Initialize the predecessor and fingers of new node n.
2. Update the fingers and predecessors of existing nodes to reflect the addition of n.
3. Notify the higher layer software so that it can transfer state associated with keys that node n is now responsible for.



Case Study 1: one node joins

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0



start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

Case Study 1: one node joins

0 (6)

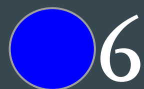
start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	0

start	int.	succ.
7	[7,0)	0
0	[0,2)	0
2	[2,6)	3

7

1 (1)

start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	0



6

2

start	int.	succ.
4	[4,5)	0
5	[5,7)	0
7	[7,3)	0

5

4

3 (2)

Case Study 1: one node joins

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	6

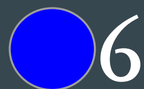
0 (6)

start	int.	succ.
7	[7,0)	0
0	[0,2)	0
2	[2,6)	3

7

1 (1)

start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	6



6

2

start	int.	succ.
4	[4,5)	6
5	[5,7)	6
7	[7,3)	0

5

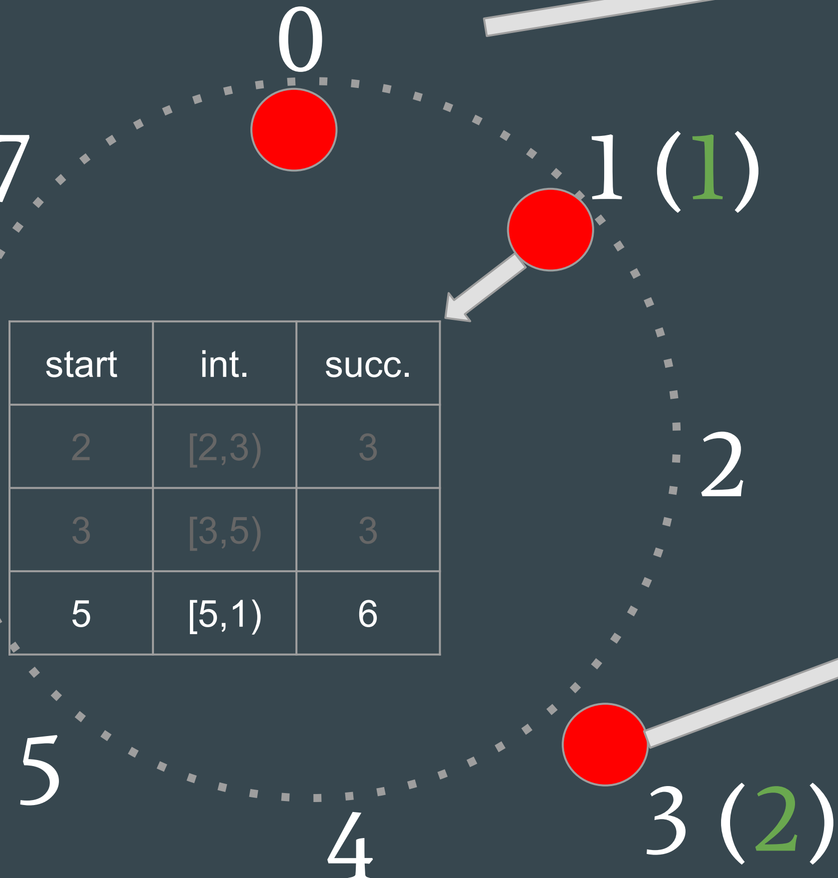
4

3 (2)

Case Study 1: one node joins

start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	6

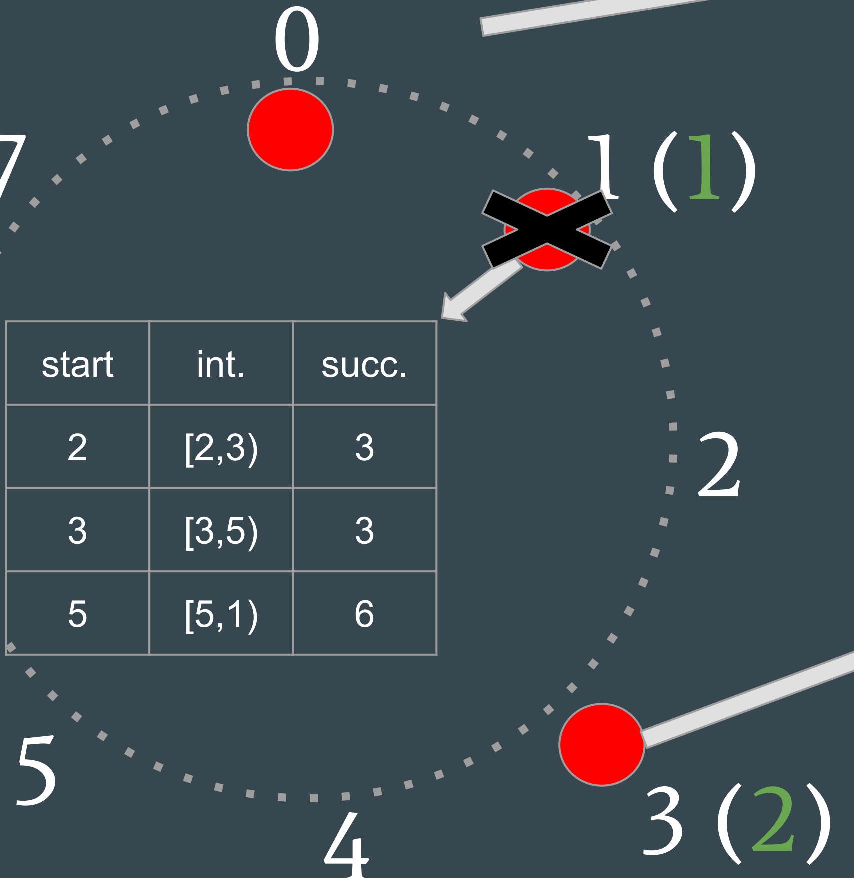
start	int.	succ.
7	[7,0)	0
0	[0,2)	0
2	[2,6)	3



start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	6

start	int.	succ.
4	[4,5)	6
5	[5,7)	6
7	[7,3)	0

Case Study 2: one node leaves



start	int.	succ.
7	[7,0)	0
0	[0,2)	0
2	[2,6)	3

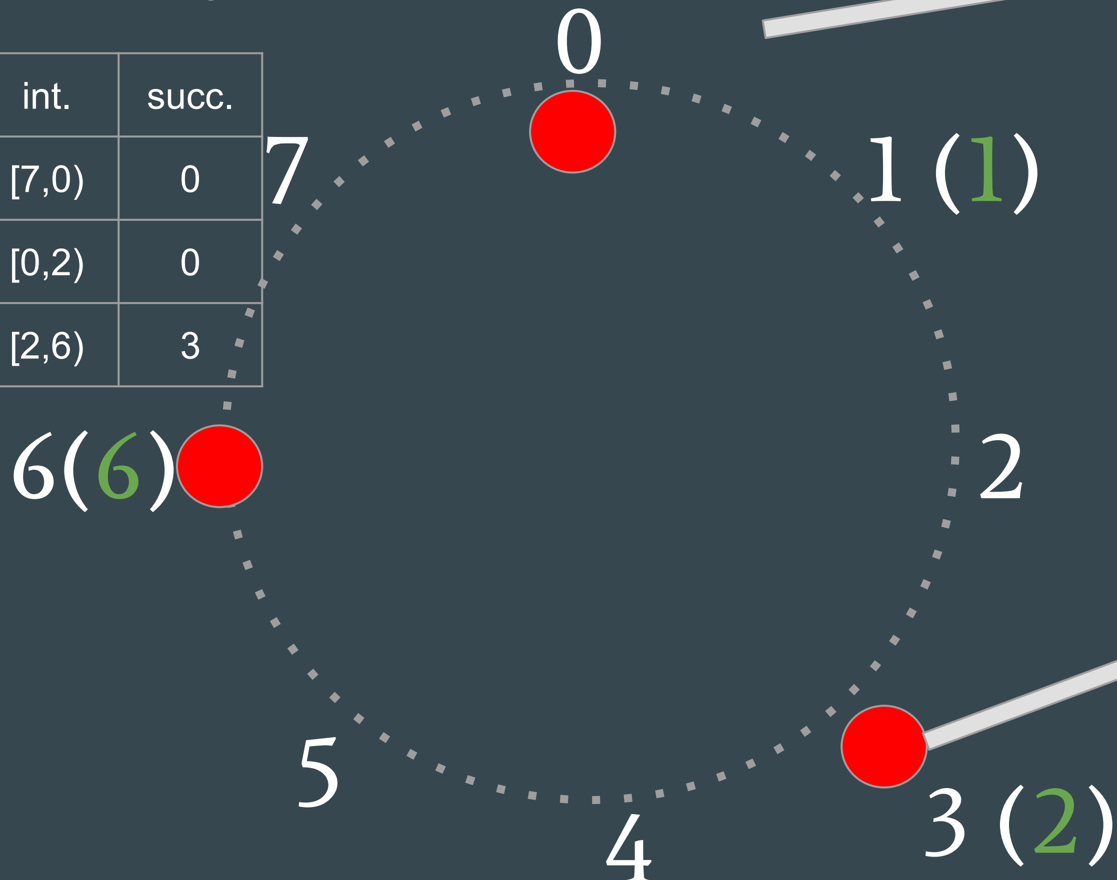
start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	6

start	int.	succ.
2	[2,3)	3
3	[3,5)	3
5	[5,1)	6

start	int.	succ.
4	[4,5)	6
5	[5,7)	6
7	[7,3)	0

Case Study 2: one node leaves

start	int.	succ.
7	[7,0)	0
0	[0,2)	0
2	[2,6)	3



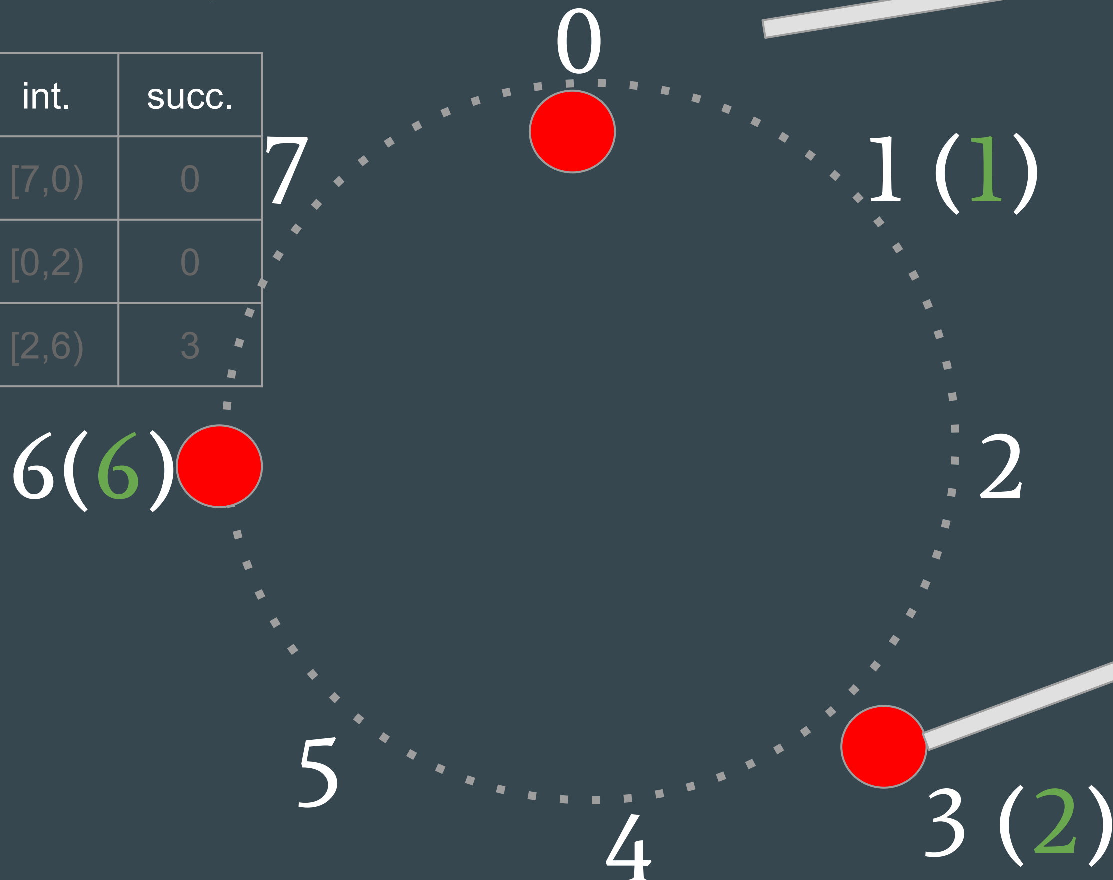
start	int.	succ.
1	[1,2)	1
2	[2,4)	3
4	[4,0)	6

start	int.	succ.
4	[4,5)	6
5	[5,7)	6
7	[7,3)	0

Case Study 2: one node leaves

start	int.	succ.
1	[1,2)	0
2	[2,4)	3
4	[4,0)	6

start	int.	succ.
7	[7,0)	0
0	[0,2)	0
2	[2,6)	3

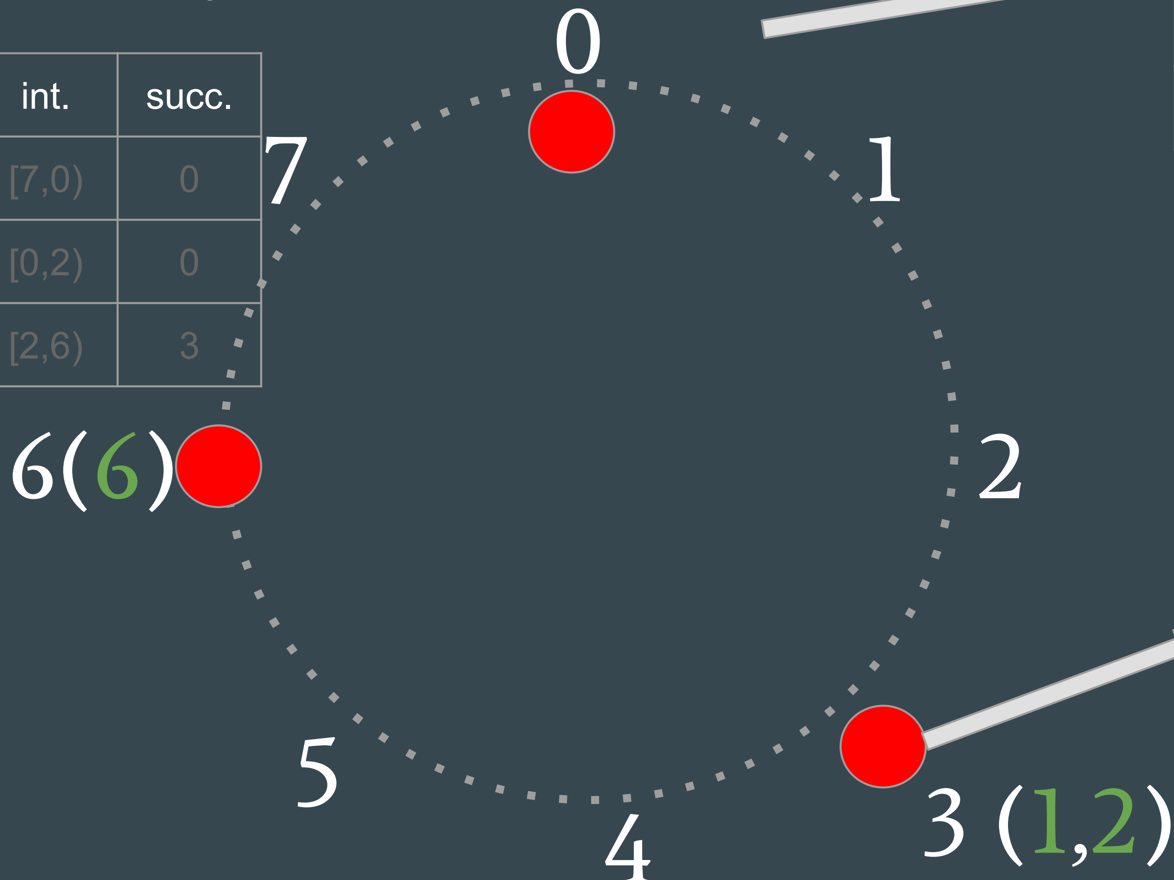


start	int.	succ.
4	[4,5)	6
5	[5,7)	6
7	[7,3)	0

Case Study 2: one node leaves

start	int.	succ.
1	[1,2)	0
2	[2,4)	3
4	[4,0)	6

start	int.	succ.
7	[7,0)	0
0	[0,2)	0
2	[2,6)	3



start	int.	succ.
4	[4,5)	6
5	[5,7)	6
7	[7,3)	0

Challenge 2: Node Joins and leaves

Will it cause communication overhead?

Theorem 3. With high probability, any node joining or leaving an N-node Chord network will use $O(\log^2(N))$ messages to re-establish the Chord routing invariants and finger tables.

Challenge 3: Concurrent Operations

What if there are concurrent joins?

A basic stabilization protocol **S**: Keep nodes' successor pointers up to date.

-- sufficient to guarantee correctness of lookups.

Challenge 3: Concurrent Operations

What if there are concurrent joins?

A basic stabilization protocol **S**: Keep nodes' successor pointers up to date.

-- sufficient to guarantee correctness of lookups.

But does it solve the problem?

-- What about the ongoing lookups?

Challenge 3: Concurrent Operations

A look up that occurs before stabilization has finished can exhibit the following three behaviors:

1. All the finger table entries involved in the lookup are current. (correct!)
2. Successor pointers are correct but fingers are not. (correct but slower...)
3. Incorrect successor pointers. (fail!)

What can we do about 3? Retrying after a pause.

Challenge 3: Concurrent Operations

What if there are concurrent joins?

A basic stabilization protocol **S**: Keep nodes' successor pointers up to date.

-- sufficient to guarantee correctness of lookups.

But does it solve the problem?

-- What about the ongoing lookups? (Problem solved!)

But how does **S** guarantee adding nodes preserves reachability of existing nodes?

Challenge 3: Concurrent Operations

Every node runs stabilize() periodically.

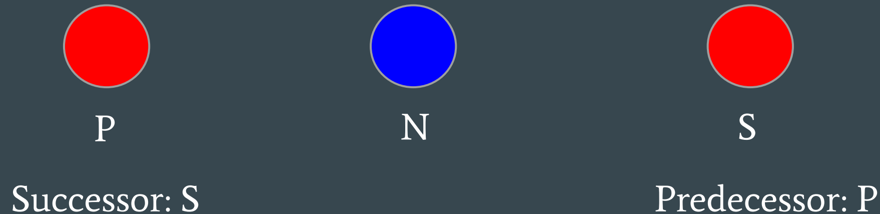
Correct the successor pointers:

(Connect to the predecessor) When n runs stabilize(), it asks n's successor for the successor's predecessor p, and decides whether p should be n's successor instead. This would be the case if node p recently joined the system.

(Connect to the successor) Notify node n's successor of n's existence, giving the successor the chance to change its predecessor to n. The successor does this only if it knows of no closer predecessor than n.

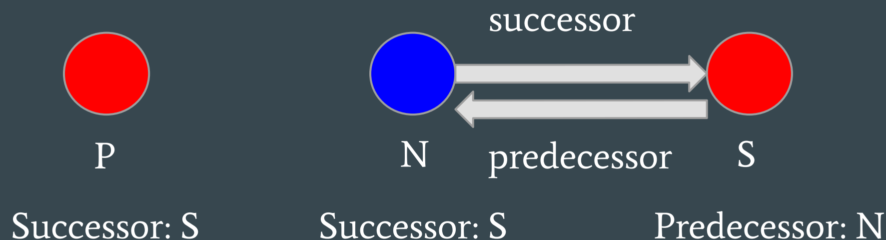
Challenge 3: Concurrent Operations

A concrete example: the blue node n joins between P and S.



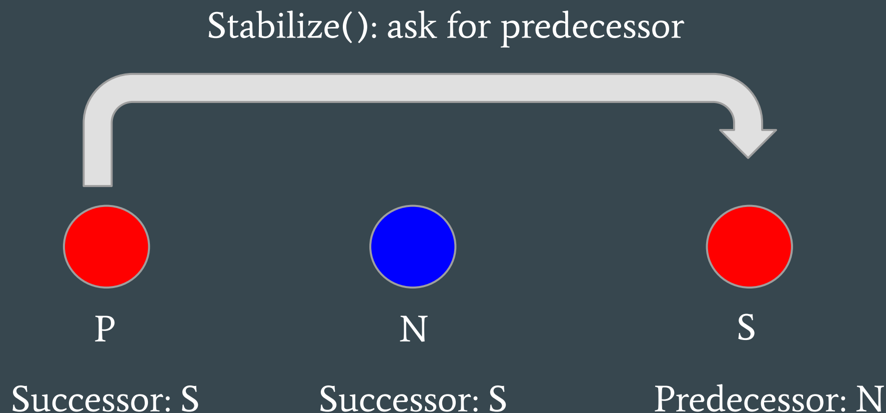
Challenge 3: Concurrent Operations

A concrete example: the blue node N joins between P and S.



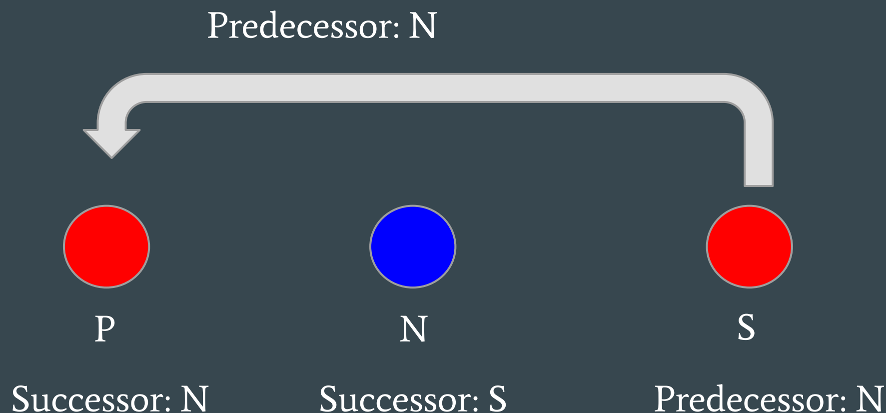
Challenge 3: Concurrent Operations

A concrete example: the blue node N joins between P and S.



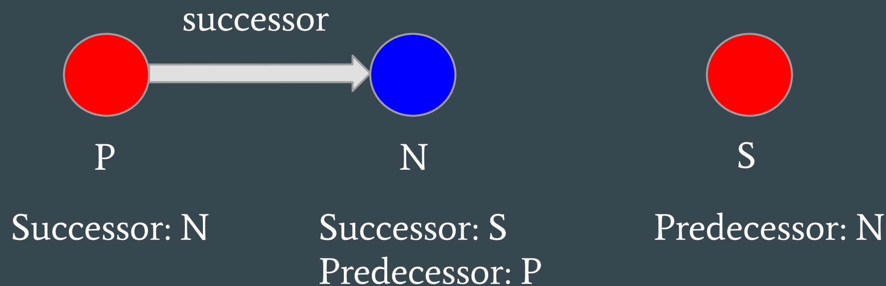
Challenge 3: Concurrent Operations

A concrete example: the blue node N joins between P and S.



Challenge 3: Concurrent Operations

A concrete example: the blue node N joins between P and S.



Challenge 3: Concurrent Operations

All the problems cause by concurrent joins are transient!

Assume any two nodes trying to communicate will eventually succeed, then

Theorem 4. Once a node can successfully resolve a given query, it will always be able to do so in the future.

Theorem 5. At some time after the last join all successor pointers will be correct.

Challenge 3: Concurrent Operations

The number of nodes between two old nodes is likely to be very small.

Theorem 6. If we take a stable network with N nodes, and another set of up to N nodes joins the network with no finger pointers (but with correct successor pointers), then lookups will still take O(log(N)) time with high probability.

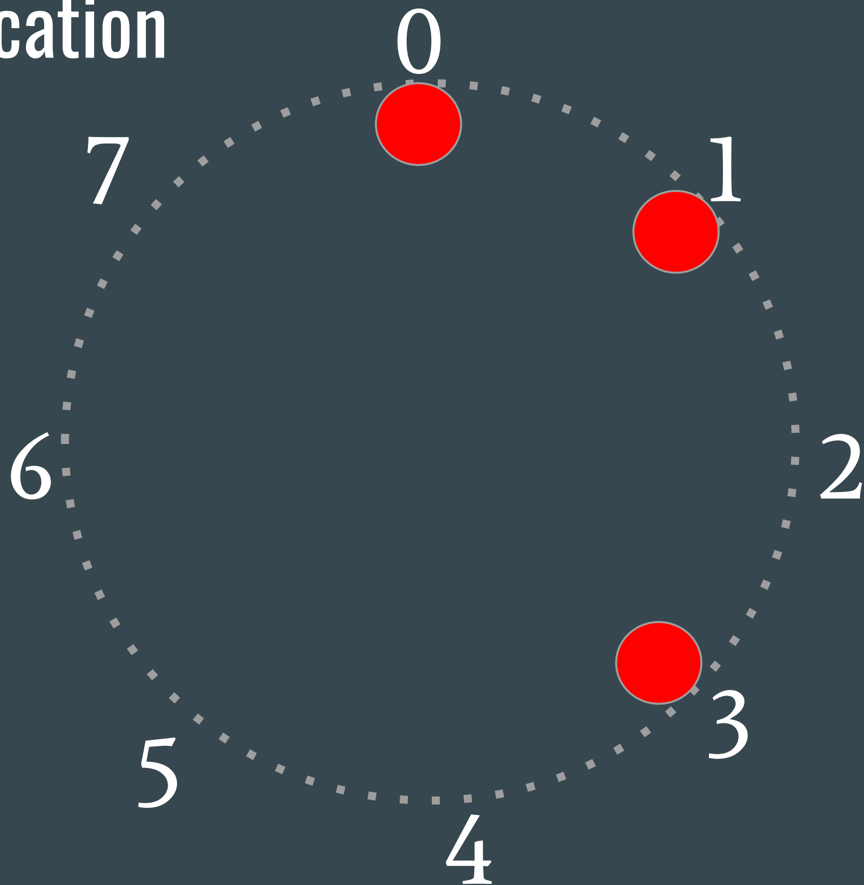
Challenge 4: Failures and Replication

When a node n fails, nodes whose finger tables include n must find n's successor.

The key step in failure recovery is maintaining correct successor pointers.

Chord maintains a "successor-list" of its r nearest successors on the Chord ring.

If node n notices that its successor has failed, it replaces it with the first live entry in its successor list.



Challenge 4: Failures and Replication

Successor-list allows lookups to succeed, and be efficient, even during stabilization:

Theorem 7. If we use a successor list of length r=O(log(N)) in a network that is initially stable, and then every node fails with probability ½, then with high probability find_successor returns the closest living successor to the query key.

Theorem 8. If we use a successor list of length r=O(log(N)) in a network that is stable, and then every node fails with probability ½, then the expected time to execute find_successor in the failed network is O(log(N)).

Challenge 4: Failures and Replication

Successor-list can also be used for replication!

Application on top of Chord might store replicas of data associated with a key on k nodes from n's successor list. When successor list changes, application can create new replicas.

Chord: a geometry perspective

A node can route to an arbitrary destination in O(log(N)) hops because each hop cuts the distance to the destination by half.

Chord: a geometry perspective

Why ring?

The static resilience of a geometry is largely determined by the amount of routing flexibility it offers. Thus, the ring which has the greatest routing flexibility has the highest resilience.

Acknowledgement

Thanks to Prof. Hakim Weatherspoon and Daniel Amir for constructive comments on this presentation!

Reference:

[1] Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications.

Ion Stoica, Roberts Morris, David Karger, Frans Kaashork, Hari Balakrishnan

[2] The Impact of DHT Routing Geometry on Resilience and Proximity.

K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, I. Stoica

[3] Lecture Notes, 15-744 CMU, fall 2016

[4] Kelips : Building an Efficient and Stable P2P DHT Through Increased Memory and Background Overhead.

Indranil Gupta, Ken Birman, Prakash Linga, Al Demers, Robbert van Renesse

Thank you!