Peer-to-Peer Systems and Distributed Hash Tables

COS 418/518: Distributed Systems
Lecture 9 & 10

Mike Freedman, Wyatt Lloyd

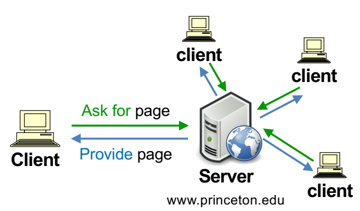
1

Today

1. Peer-to-Peer Systems
2. Distributed Hash Tables (DHT)
3. The Chord Lookup Service

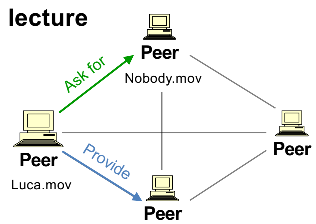
2

Distributed Application Architecture



Client-Server

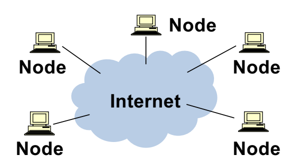
This lecture



Peer-to-Peer

3

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



- A **distributed** system architecture:
  - **No centralized control**
  - Nodes are **roughly symmetric** in function
- **Large** number of **unreliable** nodes

4

P2P adoption

Successful adoption in **some niche areas**

1. Client-to-client (legal, illegal) **file sharing**
   1. Napster (1990s), Gnutella, BitTorrent, etc.

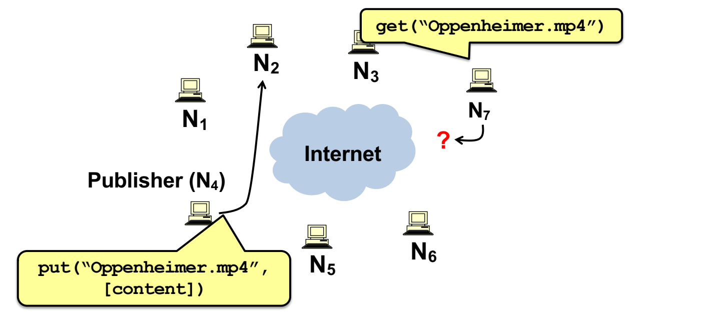
2. **Digital currency:** no natural single owner (Bitcoin)

3. **Voice/video telephony:** user to user anyway (Skype in old days)
   – Issues: Privacy and control

5

5

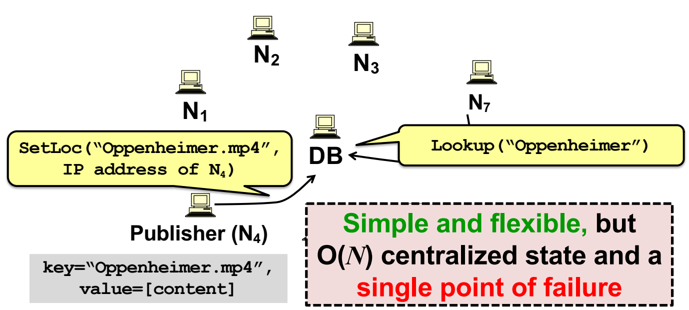
The lookup problem: locate the data



6

6

Centralized lookup (Napster)

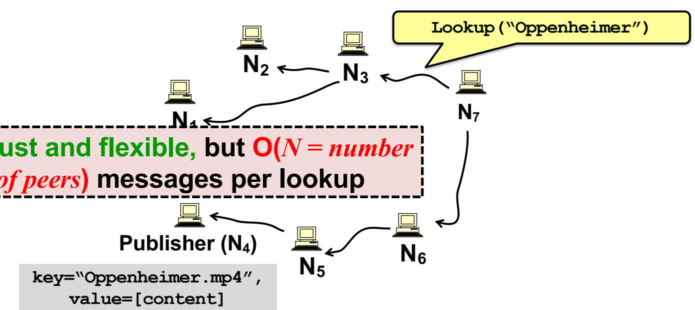


Simple and flexible, but $O(N)$ centralized state and a **single point of failure**

7

7

Flooded queries (original Gnutella)

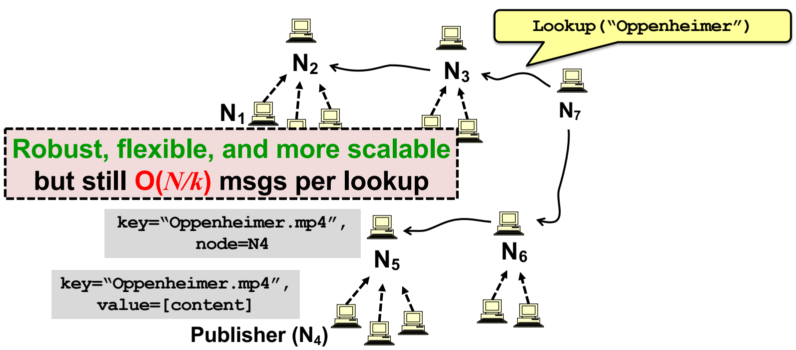


Robust and flexible, but $O(N = \textit{number of peers})$ messages per lookup

8

8

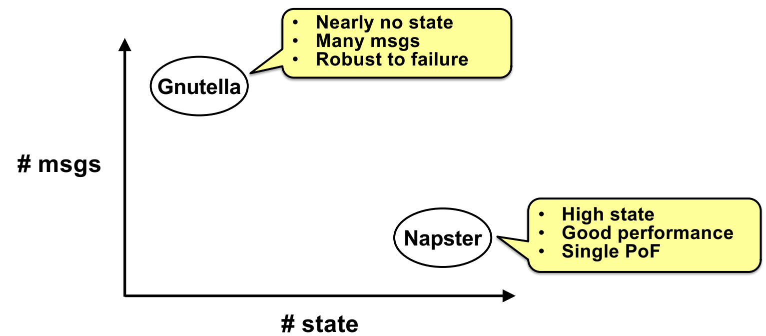
Flooded queries pt 2 (Gnutella w/ SuperPeers)



9

9

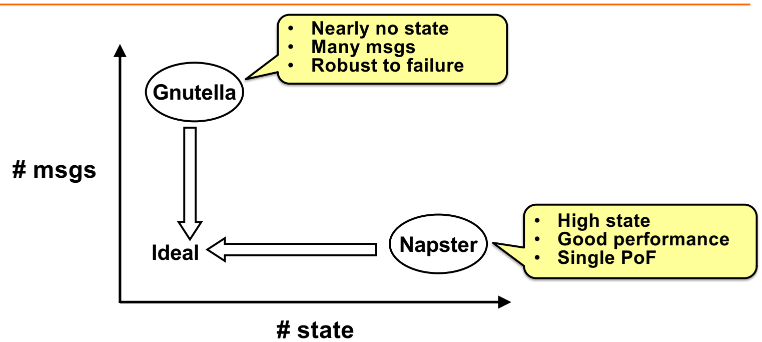
Tradeoffs in distributed systems



10

10

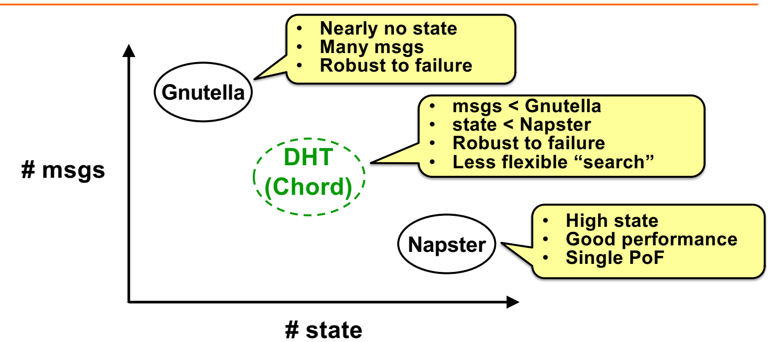
Tradeoffs in distributed systems



11

11

Tradeoffs in distributed systems



12

12

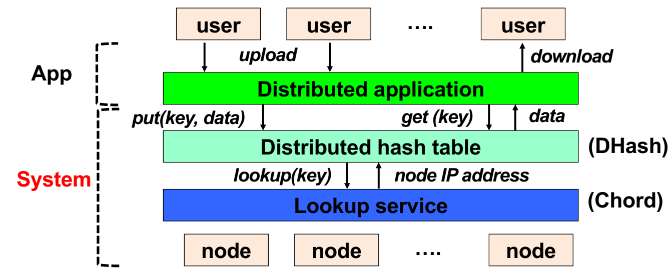
What is a DHT (and why)?

- Distributed Hash Table: an abstraction of hash table in a distributed setting

```
key = hash(data_one)
lookup(key) → IP addr   (Chord lookup service)
send-RPC(IP address, put, key, data_two)
send-RPC(IP address, get, key) → data_two
```

- **Partitioning data** in **large-scale distributed systems**
  - Tuples in a global database engine
  - Data blocks in a global file system
  - Files in a P2P file-sharing system

Cooperative storage with a DHT



DHT is expected to be

- Decentralized: no central authority
- Scalable: low network traffic overhead
- Efficient: find items quickly (latency)
- Dynamic: nodes fail, new nodes join

Today

1. Peer-to-Peer Systems
2. Distributed Hash Tables (DHT)
3. **The Chord Lookup Service**

Chord identifiers

- **Hashed values (integers) using the same hash function**
  - **Key identifier** = *SHA-1(key) mod 2^{160}*
  - **Node identifier** = *SHA-1(IP address) mod 2^{160}*

- **What is “SHA-1”?**
  - SHA-1 is a cryptographic hash function that maps input to 160-bit output hash
  - Some properties:
    1. Output hashes looks randomly distributed across output space
    2. Given *hash1*, hard to find *input1* where *SHA1(input1) = hash1*
    3. Given *input1* and *hash1*, hard to find *input2* where *SHA1(input2) = hash1*
    4. Hard to find *input1* and *input2* where *SHA1(input1) = SHA1(input2)*

17

17

Chord identifiers

- **Hashed values (integers) using the same hash function**
  - **Key identifier** = *SHA-1(key) mod 2^{160}*
  - **Node identifier** = *SHA-1(IP address) mod 2^{160}*

- **How does Chord partition data?**
  - i.e., map key IDs to node IDs

- **Why hash key and address?**
  - Uniformly distributed in the ID space
  - Hashed key → load balancing;   hashed address → independent failure

18

18

Alternative: mod (n) hashing

- **System of n nodes: 1…n**
  - Node that owns key is assigned via *hash(key) mod n*
  - Good load balancing
- **What if a node fails?**
  - Instead of n nodes, now *n -1* nodes
  - Mapping of all keys change, as now *hash(key) mod (n-1)*

• **N = 5**	• **N = 4**
– 12594 mod 5 = 4	– 12594 mod 4 = **2**
– 28527 mod 5 = 2	– 28527 mod 4 = **3**
– 816 mod 5 = 1	– 816 mod 4 = **0**
– 716565 mod 5 = 0	– 716565 mod 4 = **1**

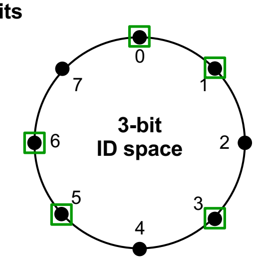
19

19

Consistent hashing [Karger ‘97] Data partitioning

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● **Identifiers/key space**
□ **Node**



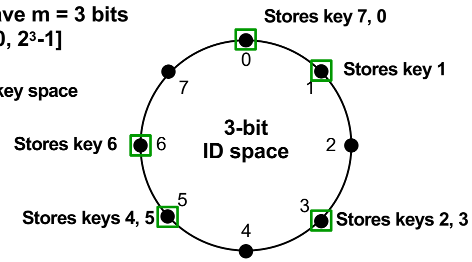
20

20

Consistent hashing [Karger '97]
Data partitioning

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node

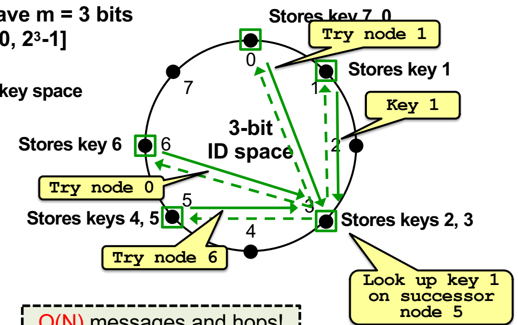


Key is stored at its **successor:** node with next-higher ID

Consistent hashing [Karger '97]
Strawman lookup vis successors

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node

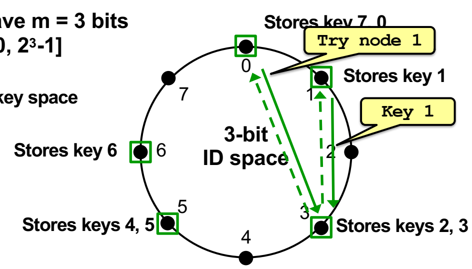


O(N) messages and hops!

Consistent hashing [Karger '97]
Observation about last hop

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node



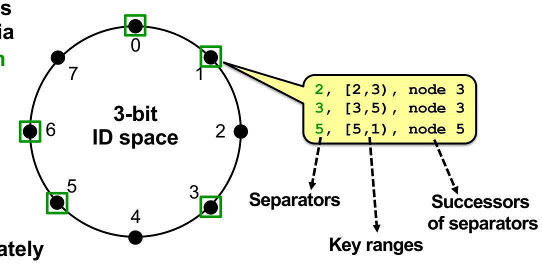
Try to find key's predecessor node as fast as possible.
This precedessor will know key's successor (owner).

Chord – finger tables for *find_predecessor*

Each node keeps m states
Key space → m ranges via
($N+2^{k-1}$) mod 2^m, 1<=k<=m

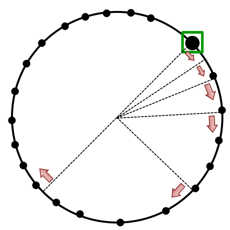
Example for node N = 1:
- 1 + 1 mod 8 => 2
- 1 + 2 mod 8 => 3
- 1 + 4 mod 8 => 5

N 2^{k-1}



"Finger" is node immediately succeeding separator

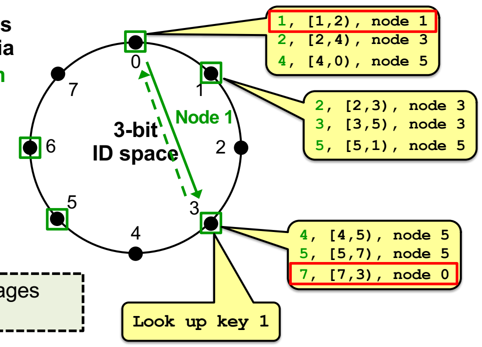
Chord – finger tables for *find_predecessor*



25

Chord – finger tables for *find_predecessor*

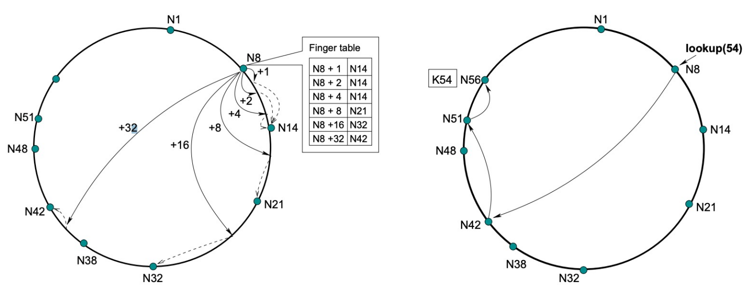
Each node keeps m states
Key space → m ranges via
(N+2^{k-1}) mod 2^m, 1<=k<=m



O(logN) messages
and hops!

26

Chord – finger tables



From Chord ToN paper

27

Implication of finger tables

- A **binary lookup tree** rooted at every node
  - Threaded through other nodes' finger tables
- Better than arranging nodes in a single tree
  - Every node acts as a root
    - So there's **no root hotspot**
    - **No single point** of failure
    - But a **lot more state** in total: N nodes each have O(log N)

28

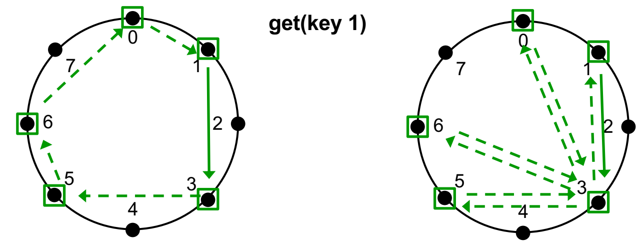
Chord lookup algorithm properties

- **Interface:** lookup(key) → IP address
- **Efficient:** O(log N) messages per lookup
  - N is the total number of nodes (peers)
- **Scalable:** O(log N) state per node
- **Robust:** survives massive failures

29

29

Chord – Recursive vs. Iterative Lookup



Recursive Lookup

Iterative Lookup

30

30

System Dynamics

- Handling node joins
- Handling node failures
  - Rebuilding lookup structures
  - Ensure data durability

31

31

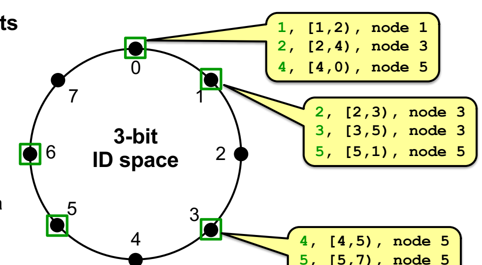
Chord – finger tables

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node

Each node keeps m states
Key space → m ranges via
($N+2^{k-1}$) mod 2^m, 1<=k<=m

"Finger" is node immediately succeeding separator



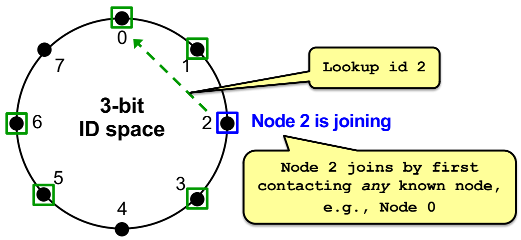
32

32

Chord – node joining

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

- Identifiers/key space
- Node



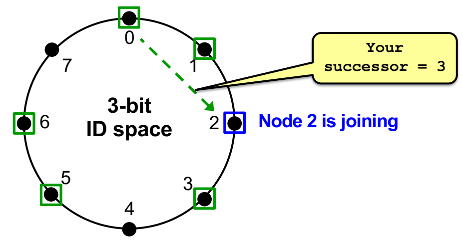
33

33

Chord – node joining

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

- Identifiers/key space
- Node



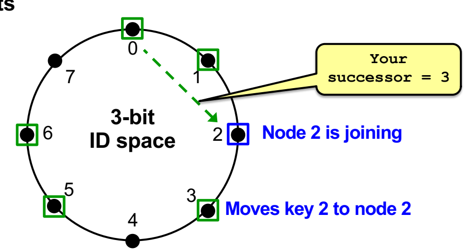
34

34

Chord – node joining

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

- Identifiers/key space
- Node



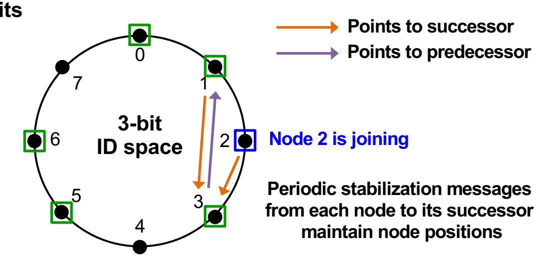
35

35

Chord – node joining

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

- Identifiers/key space
- Node



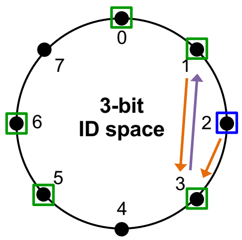
36

36

Chord – node joining

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node



→ Points to successor
→ Points to predecessor

Node 2 is joining

```
STABILIZE() [N.successor = M]
   N->M: "What is your predecessor?"
   M->N: "x is my predecessor"
   if x between (N,M), N.successor = x
   N->N.successor: NOTIFY()
NOTIFY()
   N->N.successor: "I think you are my successor"
M: upon receiving NOTIFY from N:
   If (N between (M.predecessor, M))
      M.predecessor = N
```

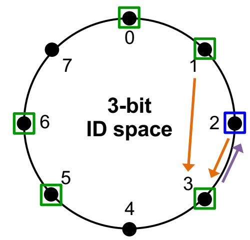
Pseudocode from Rodrigo Fonseca's lecture notes

37

Chord – node joining

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node



→ Points to successor
→ Points to predecessor

Node 2 is joining

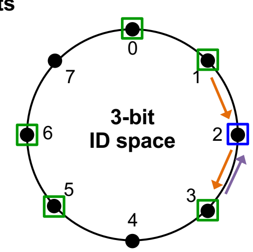
```
STABILIZE() [N.successor = M]
   N->M: "What is your predecessor?"
   M->N: "x is my predecessor"
   if x between (N,M), N.successor = x
   N->N.successor: NOTIFY()
NOTIFY()
   N->N.successor: "I think you are my successor"
M: upon receiving NOTIFY from N:
   If (N between (M.predecessor, M))
      M.predecessor = N
```


38

Chord – node joining

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node



→ Points to successor
→ Points to predecessor

Node 2 is joining

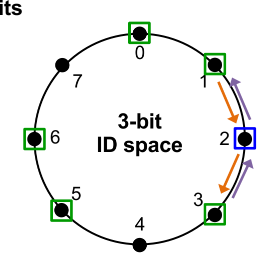
```
STABILIZE() [N.successor = M]
   N->M: "What is your predecessor?"
   M->N: "x is my predecessor"
   if x between (N,M), N.successor = x
   N->N.successor: NOTIFY()
NOTIFY()
   N->N.successor: "I think you are my successor"
M: upon receiving NOTIFY from N:
   If (N between (M.predecessor, M))
      M.predecessor = N
```


39

Chord – node joining

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node



→ Points to successor
→ Points to predecessor

Node 2 is joining

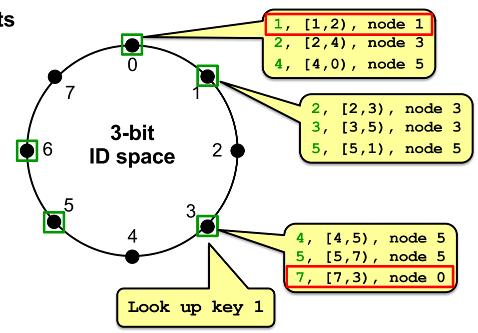
```
STABILIZE() [N.successor = M]
   N->M: "What is your predecessor?"
   M->N: "x is my predecessor"
   if x between (N,M), N.successor = x
   N->N.successor: NOTIFY()
NOTIFY()
   N->N.successor: "I think you are my successor"
M: upon receiving NOTIFY from N:
   If (N between (M.predecessor, M))
      M.predecessor = N
```


40

Chord – failures and successor list

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node



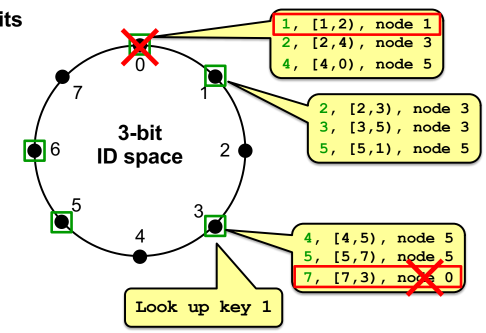
41

41

Chord – failures and successor list

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node



42

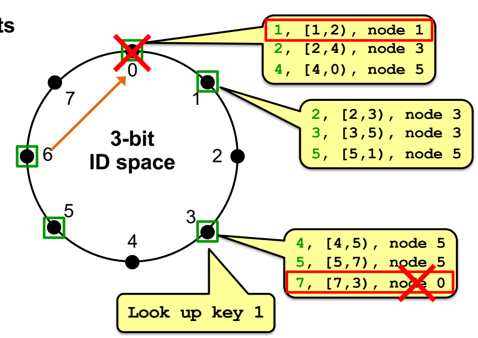
42

Chord – failures and successor list

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node

→ Points to successor



43

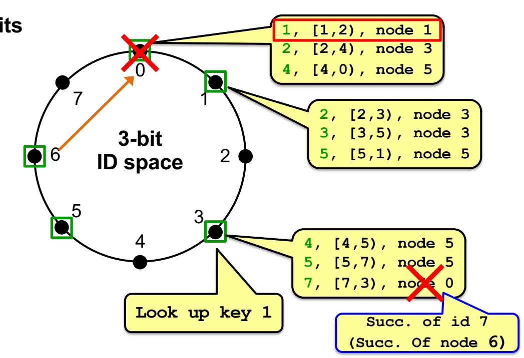
43

Chord – failures and successor list

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node

→ Points to successor



44

44

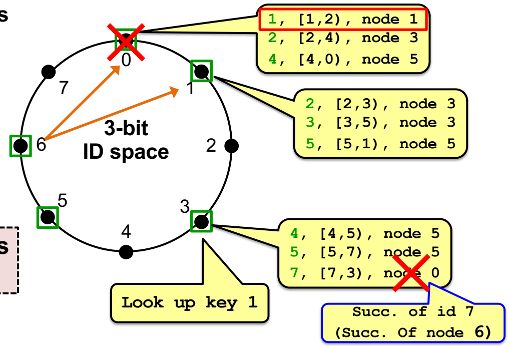
Chord – failures and successor list

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node

→ Points to successor

r-nearest successors
(r = logN)



45

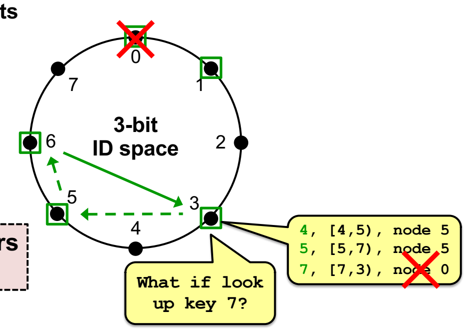
45

Chord – failures and successor list

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node

r-nearest successors
(r = logN)



46

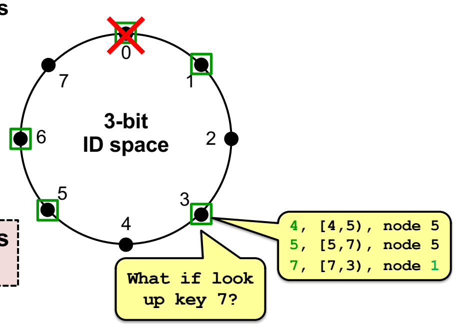
46

Chord – failures and successor list

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node

r-nearest successors
(r = logN)



47

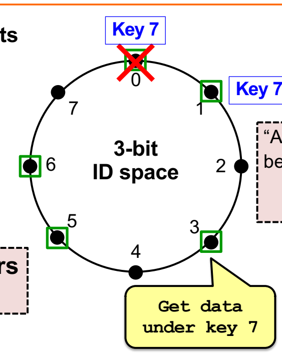
47

DHash replicates data blocks at *r* successors

Identifiers have m = 3 bits
Key space: [0, 2^3-1]

● Identifiers/key space
□ Node

r-nearest successors
(r = logN)



"Adjacent" nodes in the ring may be far away in the network
→ Independent failures

48

48

Today

1. Peer-to-Peer Systems
2. Distributed Hash Tables
3. The Chord Lookup Service
4. **Concluding thoughts on DHT, P2P**

49

49

Why don't all services use P2P?

- **High latency and limited bandwidth** between peers (vs. intra/inter-datacenter, client-server model)
  - 1 M nodes = 20 hops; 50 ms / hop gives 1 sec lookup latency (assuming no failures / slow connections…)
- User computers are **less reliable** than managed servers
- **Lack of trust** in peers' correct behavior
  - Securing DHT routing hard, unsolved in practice

50

50

DHTs in retrospective

- Seem promising for finding data in large P2P systems
- Decentralization seems good for load, fault tolerance
- **But:** the **security problems** are difficult
- **But:** **churn** is a problem, particularly if log(n) is big
- DHTs have not had the hoped-for impact

51

51

What DHTs got right

- **Consistent hashing**
  - Elegant way to divide a workload across machines
  - Very useful in clusters: actively used today in Amazon Dynamo and other systems
- **Replication** for high availability, efficient recovery
- **Incremental scalability**
  - Peers join with capacity, CPU, network, etc.
- **Self-management:** minimal configuration

52

52