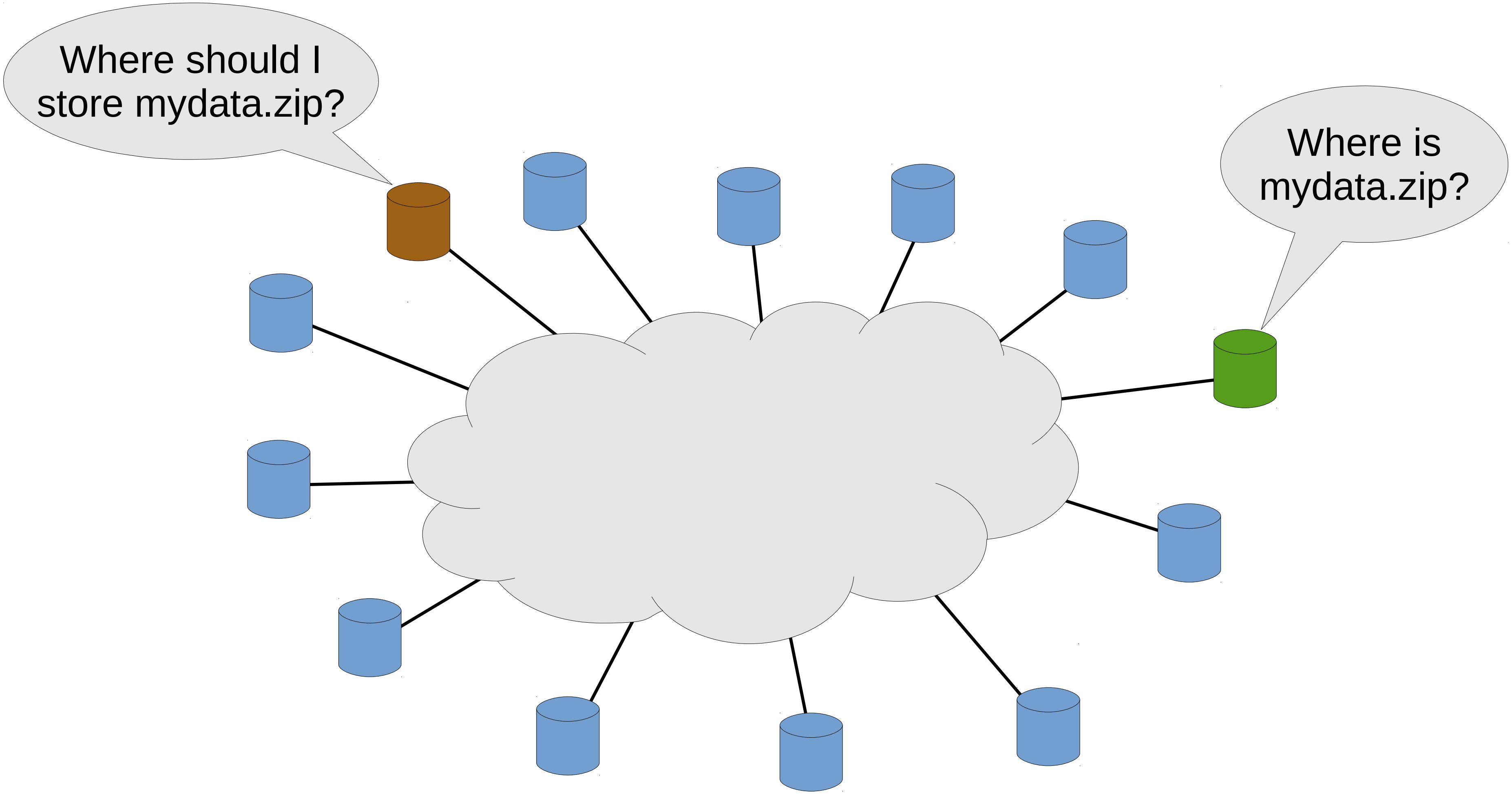
Distributed Hash Tables

Manfred Dellkrantz

Introduction to Cloud Computing, 2015-03-17

Department of Automatic Control, Lund University, Sweden

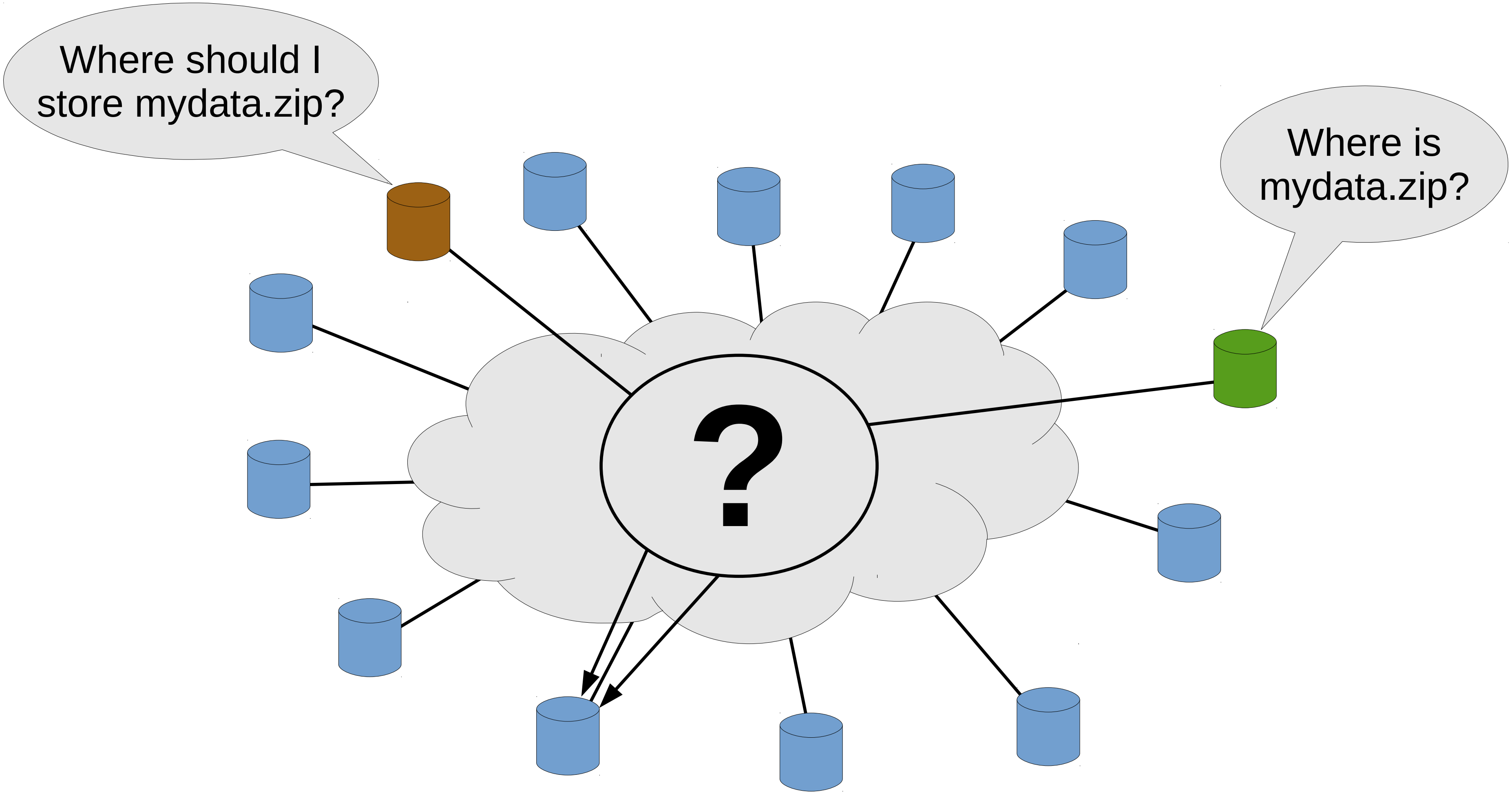
DHT – Finding data



Where should I store mydata.zip?

Where is mydata.zip?

DHT – Finding data



DHT – Finding data

When storing lots of data in a network of computers, how do we find a certain piece of data?

Applications:

- NoSQL databases (Amazon Dynamo, Cassandra)
- File sharing (Torrent Mainline DHT)
- Distributed file systems (GlusterFS)
- Content Distribution (Coral CDN)

References

[Stoica2001]	Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, Ion Stoica *et al*
[DeCandia2007]	Dynamo: Amazon’s Highly Available Key-value Store, Giuseppe DeCandia *et al*
[Maymounkov2002]	Kademlia: A Peer-to-Peer Information System Based on the XOR Metric, Petar Maymounkov *et al*
[Rowstron2001]	Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems, Antony Rowstron *et al*

Traditional Hash Table

- Use *N* buckets to store (*key*, *value*) items
- Store item in bucket number `id=hash(key)%N`
- If the item is in the table we know it is stored in bucket `id=hash(key)%N`
- Store and retrieve value in `O(1)` time

Modulo operator

Hash Table example

N = 5 buckets.

```
// Store data '0x45' in key 'mydata.zip'
store('mydata.zip', '0x45')
  hash('mydata.zip') = 73
  73 % 5 = 3
  table[3] = '0x45'

// Get data for key 'mydata.zip'
get('mydata.zip')
  hash('mydata.zip') = 73
  73 % 5 = 3
  return table[3]
```

Bucket	Data
0	
1	
2	
3	0x45
4	

Can this be directly distributed?

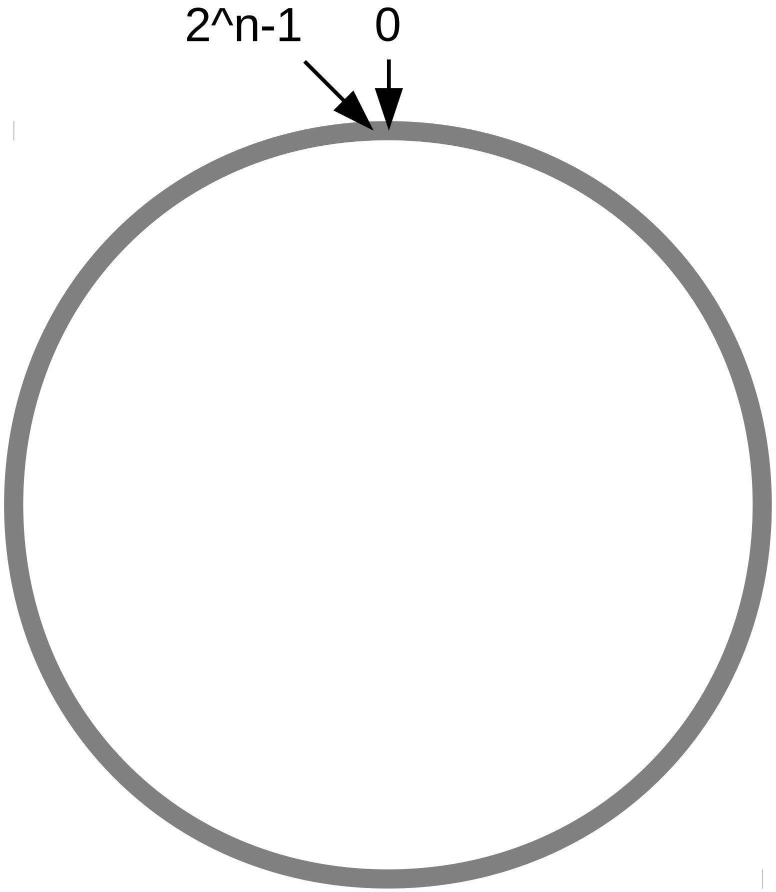
Distributing the Hash Table

- Use *N* networked computers (nodes)
- Store item in node number `id=hash(key)%N`

- *N* will change!
  - *Nodes go offline/crash or we need to increase capacity*
- Changing the number of buckets in a hash table will cause almost all items to move

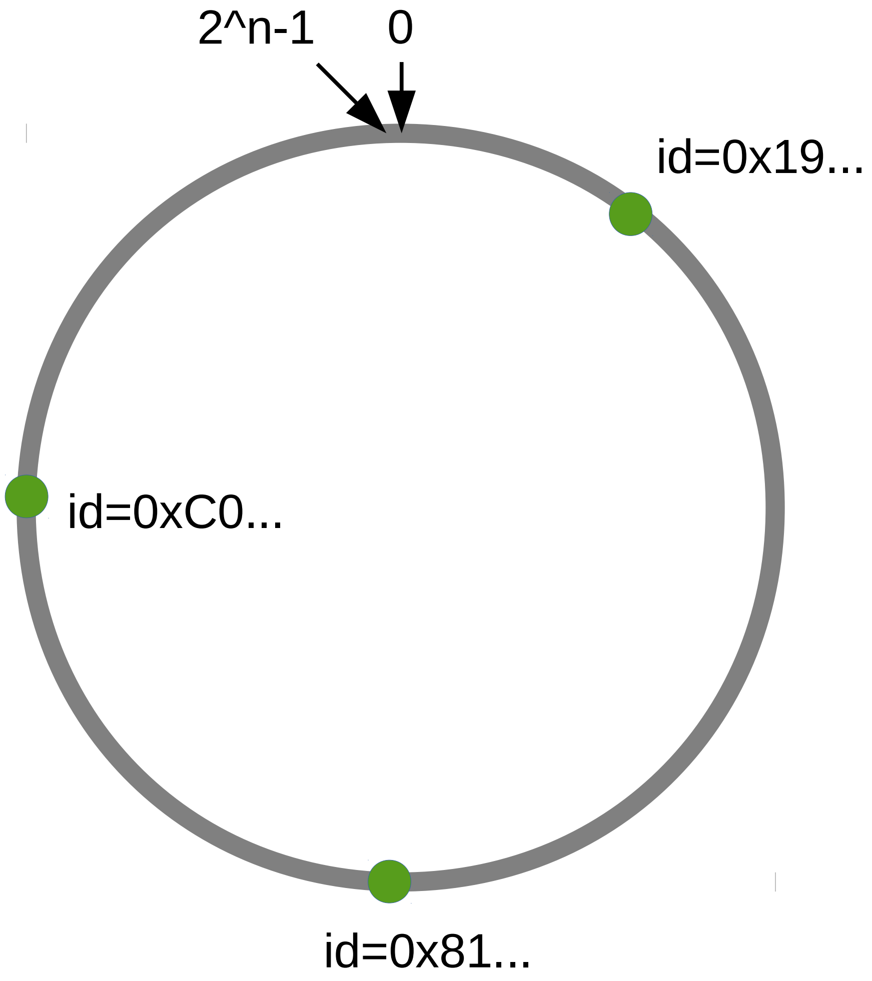
Consistent Hashing

- Map the *n*-bit hash to a ring
- Place all nodes at some ids on the ring
- Items are placed on the ring at its `hash(key)`
- An item is the responsibility of the node “nearest” to the item on the ring
- “Nearest” often means nearest clock-wise, including its own id



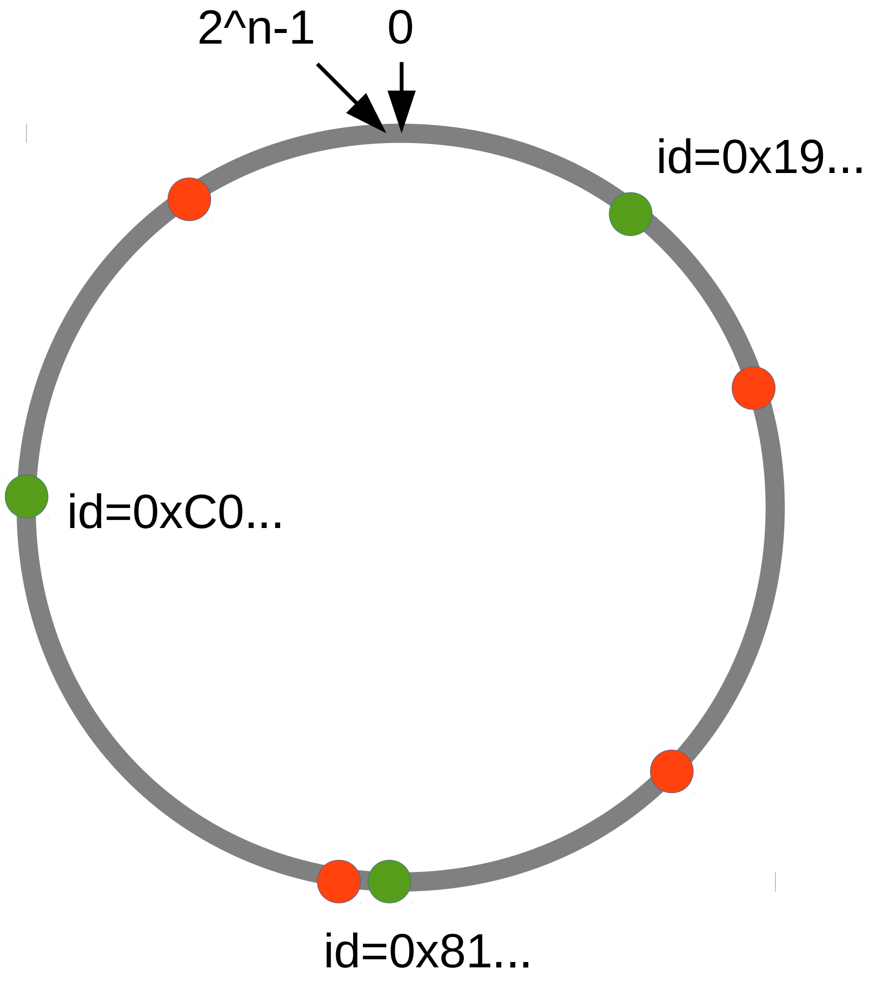
Consistent Hashing

- Map the *n*-bit hash to a ring
- Place all nodes at some ids on the ring
- Items are placed on the ring at its `hash(key)`
- An item is the responsibility of the node “nearest” to the item on the ring
- “Nearest” often means nearest clock-wise, including its own id



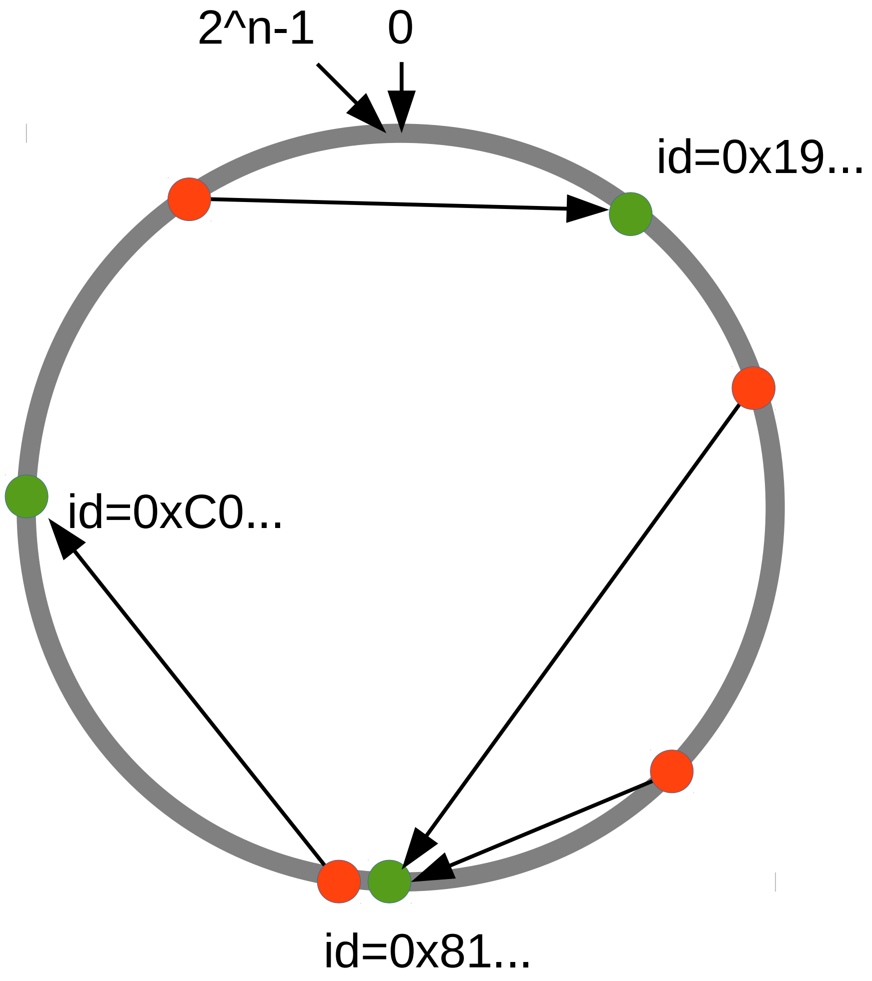
Consistent Hashing

- Map the *n*-bit hash to a ring
- Place all nodes at some ids on the ring
- Items are placed on the ring at its `hash(key)`
- An item is the responsibility of the node “nearest” to the item on the ring
- “Nearest” often means nearest clock-wise, including its own id



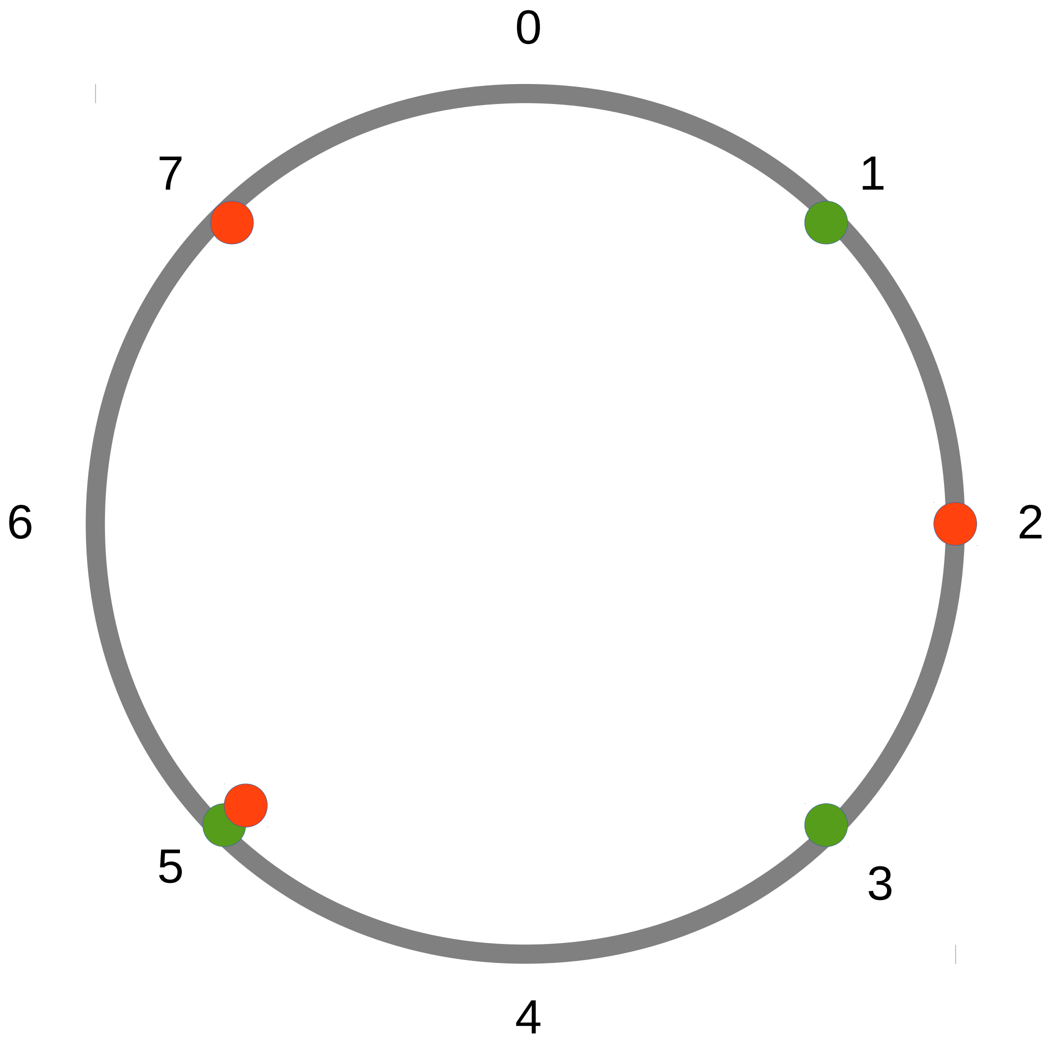
Consistent Hashing

- Map the *n*-bit hash to a ring
- Place all nodes at some ids on the ring
- Items are placed on the ring at its `hash(key)`
- An item is the responsibility of the node “nearest” to the item on the ring
- “Nearest” often means nearest clock-wise, including its own id



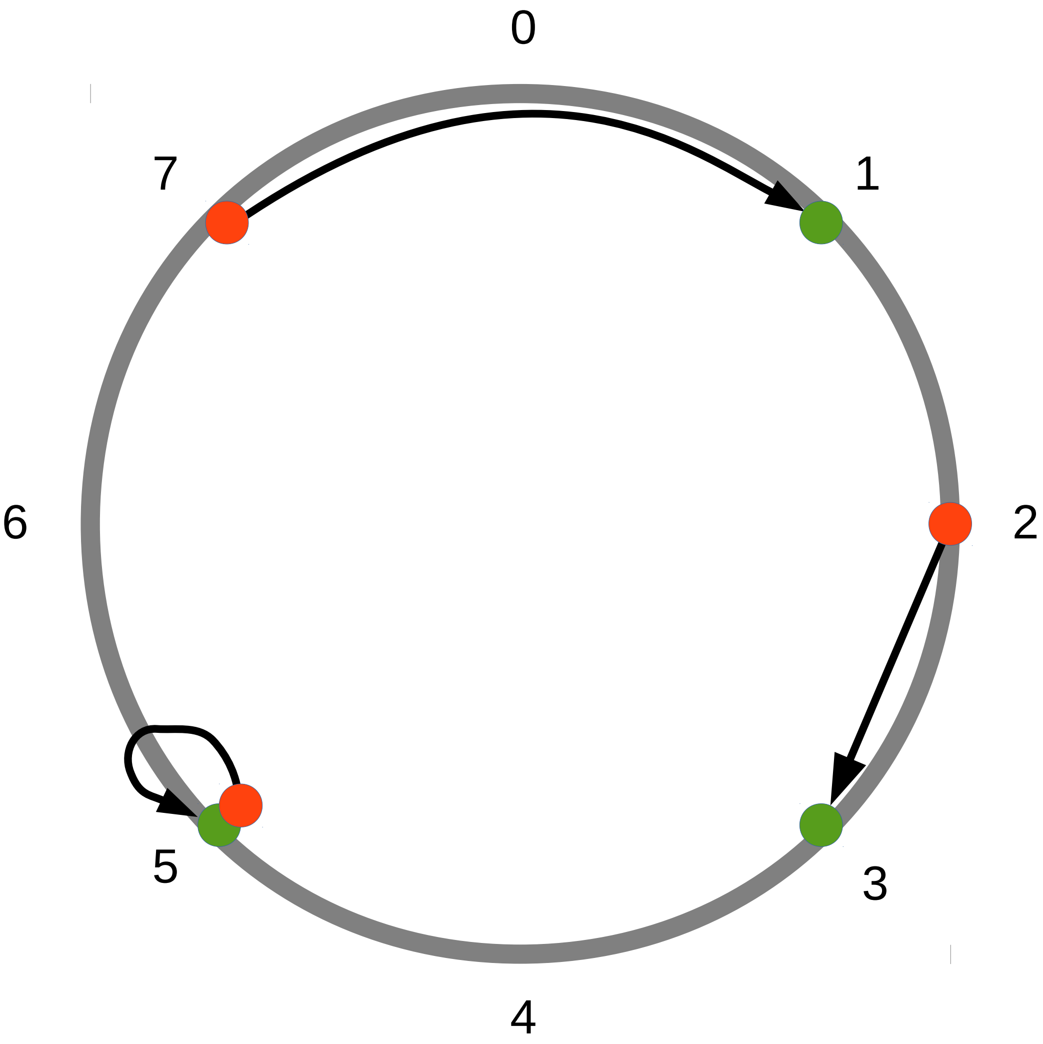
Consistent Hashing Example

- $n = 3$
- $2^3 = 8$ possible ids
- Three nodes with ids 1, 3, 5
- Three items with ids 2, 5, 7



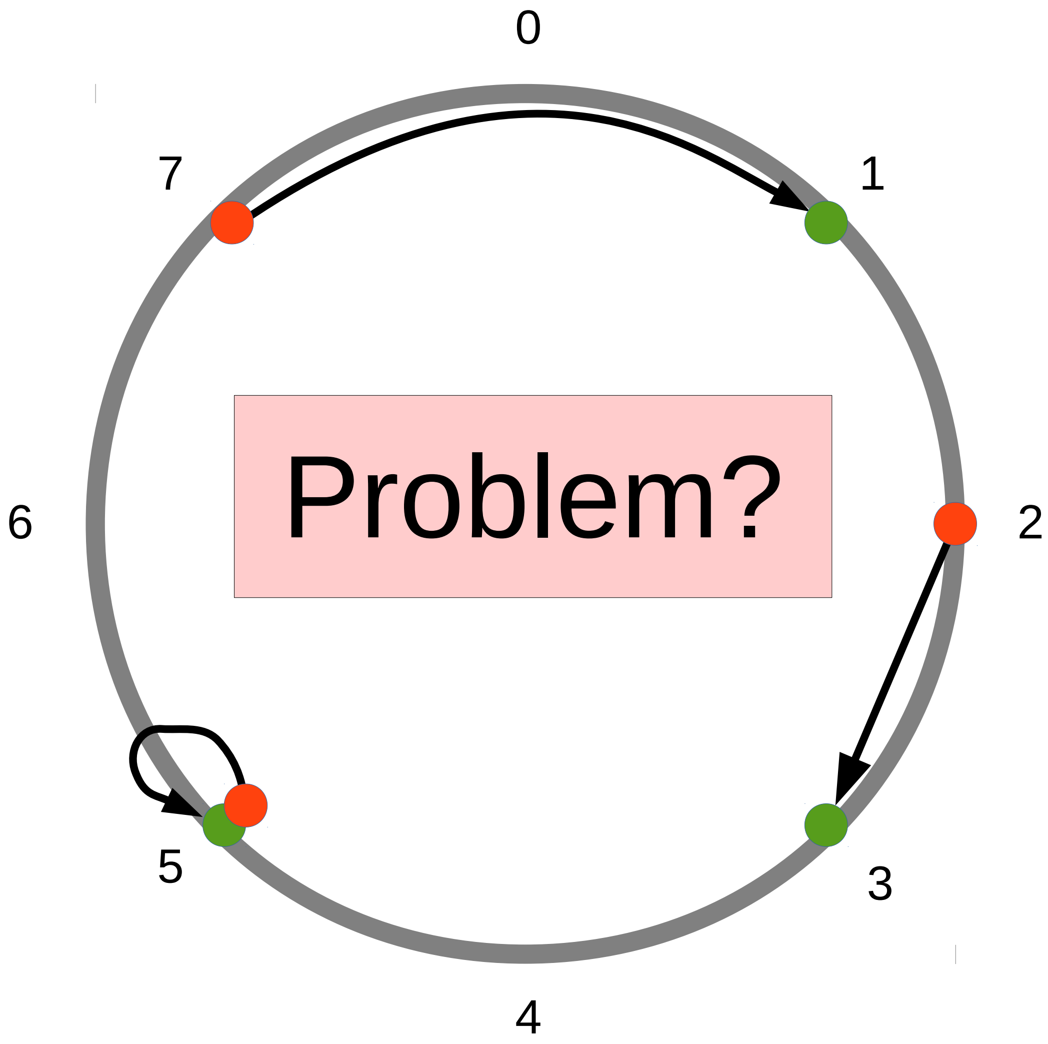
Consistent Hashing Example

- $n = 3$
- $2^3 = 8$ possible ids
- Three nodes with ids 1, 3, 5
- Three items with ids 2, 5, 7



Consistent Hashing Example

- $n = 3$
- $2^3 = 8$ possible ids
- Three nodes with ids 1, 3, 5
- Three items with ids 2, 5, 7

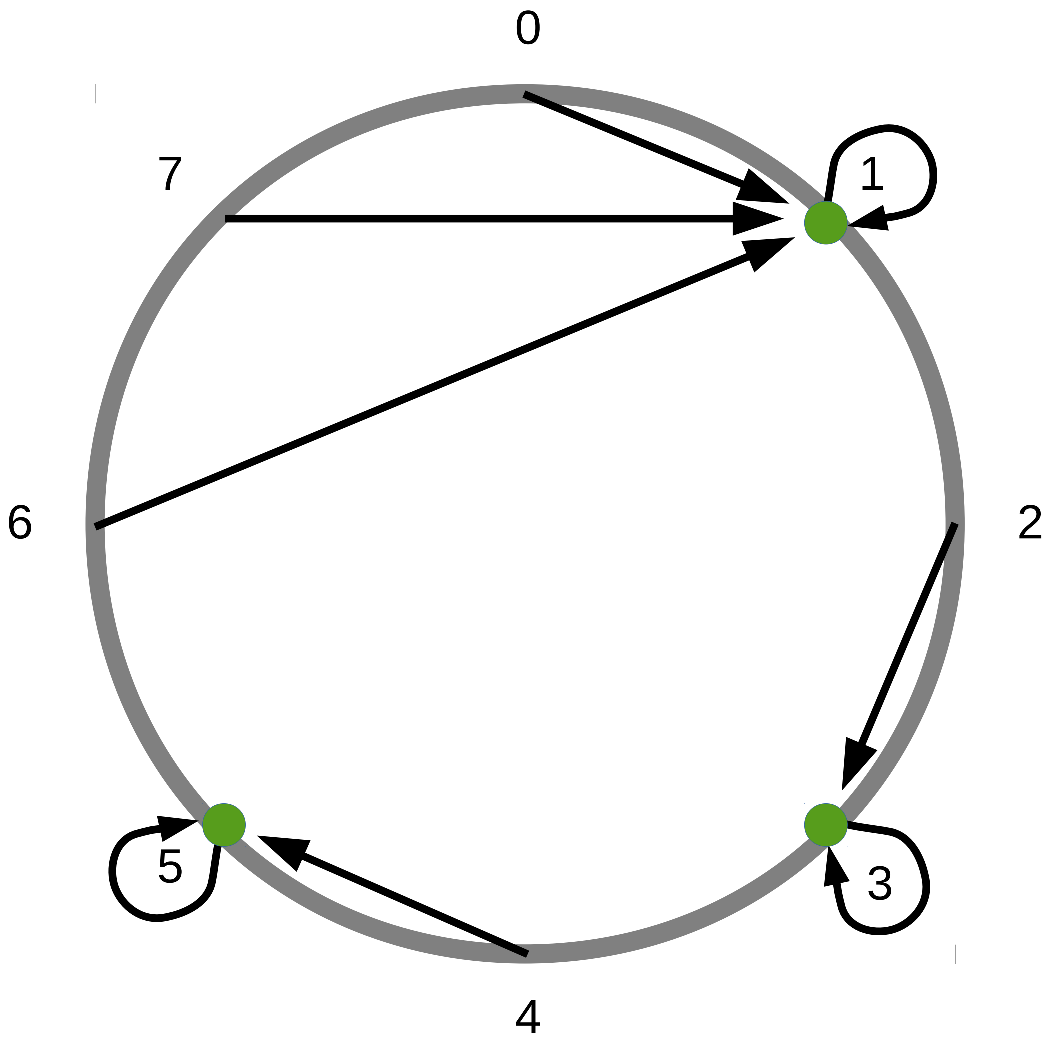


Virtual Nodes

Problem: Node 1 has double the responsibility compared to the other nodes!

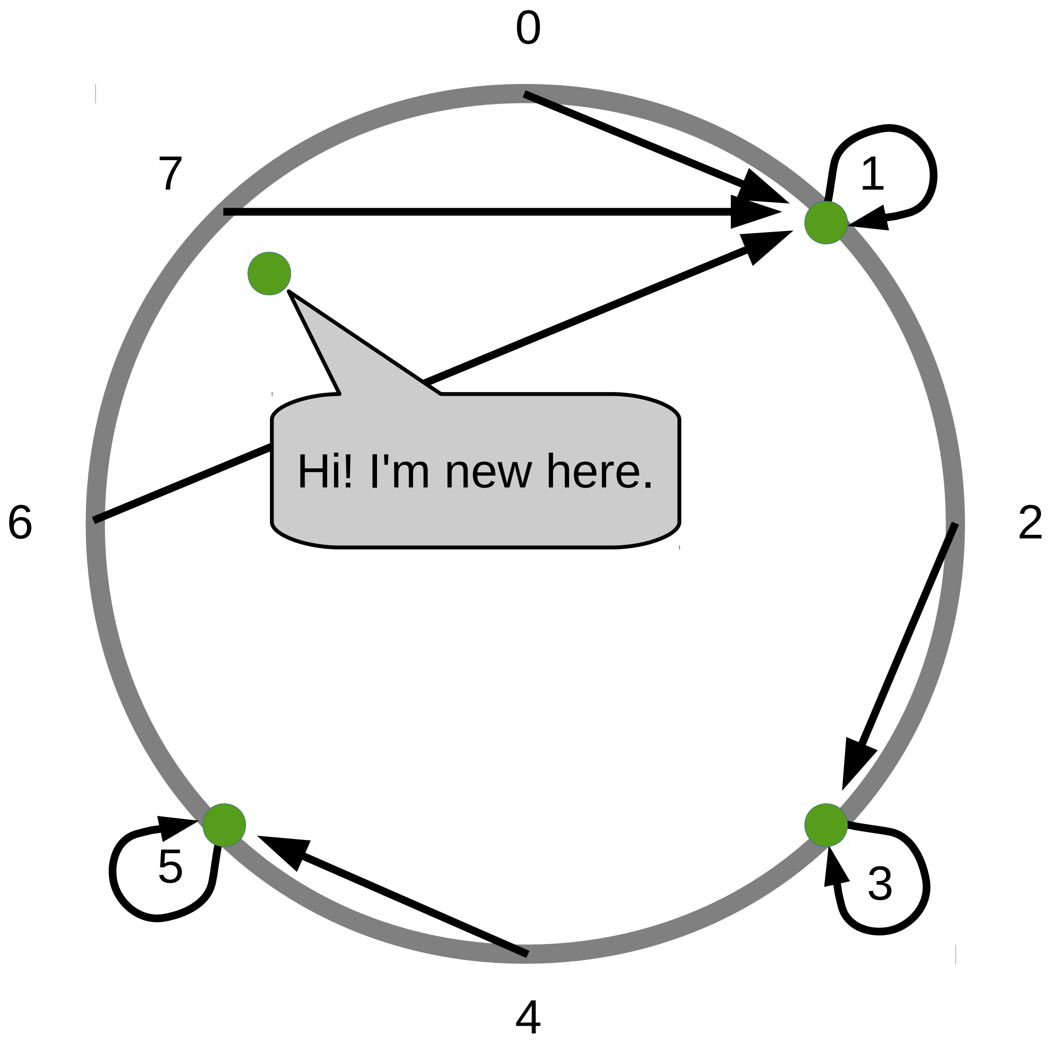
Solution: Each “physical” node has several virtual nodes spread out over the ring

For heterogeneous nodes the number of virtual nodes can be made proportional to the node's capacity.



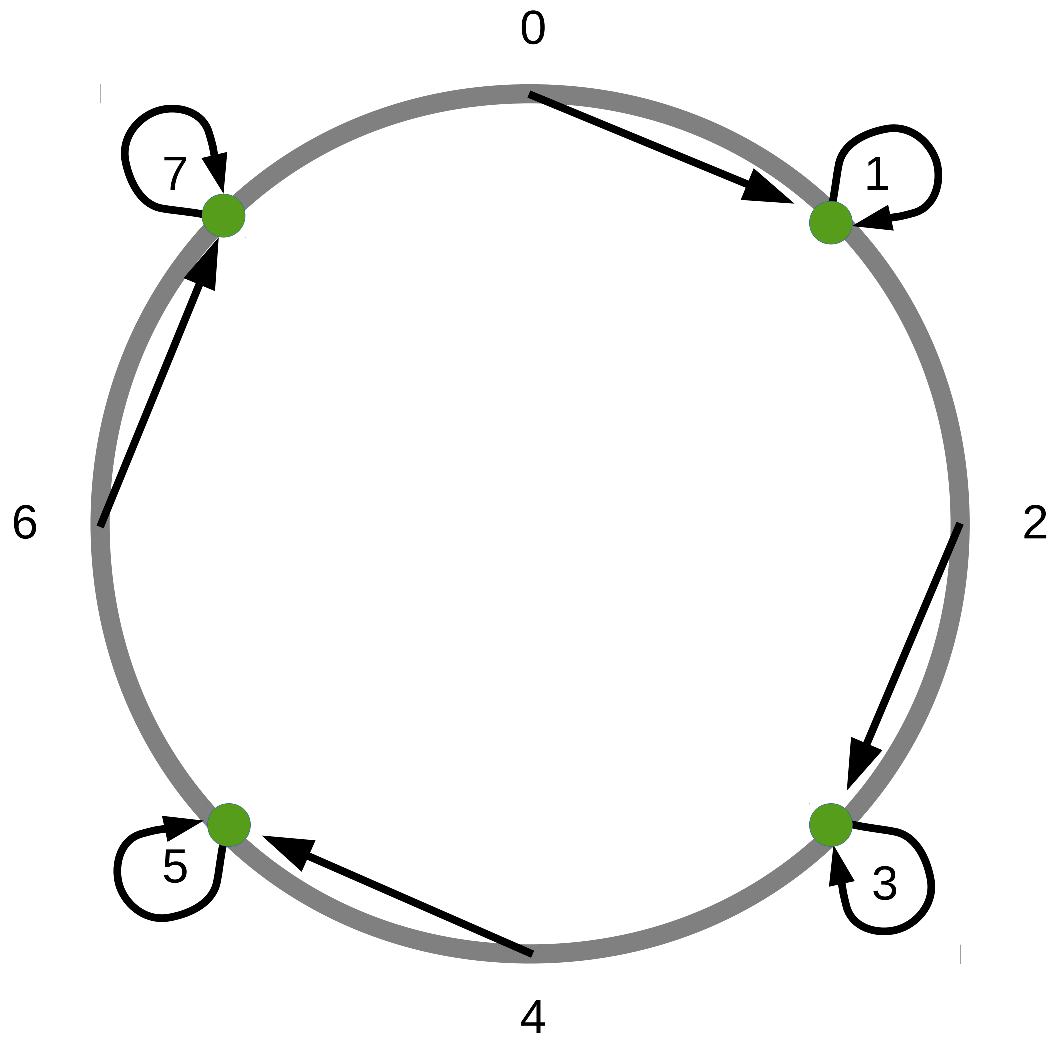
Adding nodes

Adding a new node affects only the node that had responsibility of the interval where the new node is added.



Adding nodes

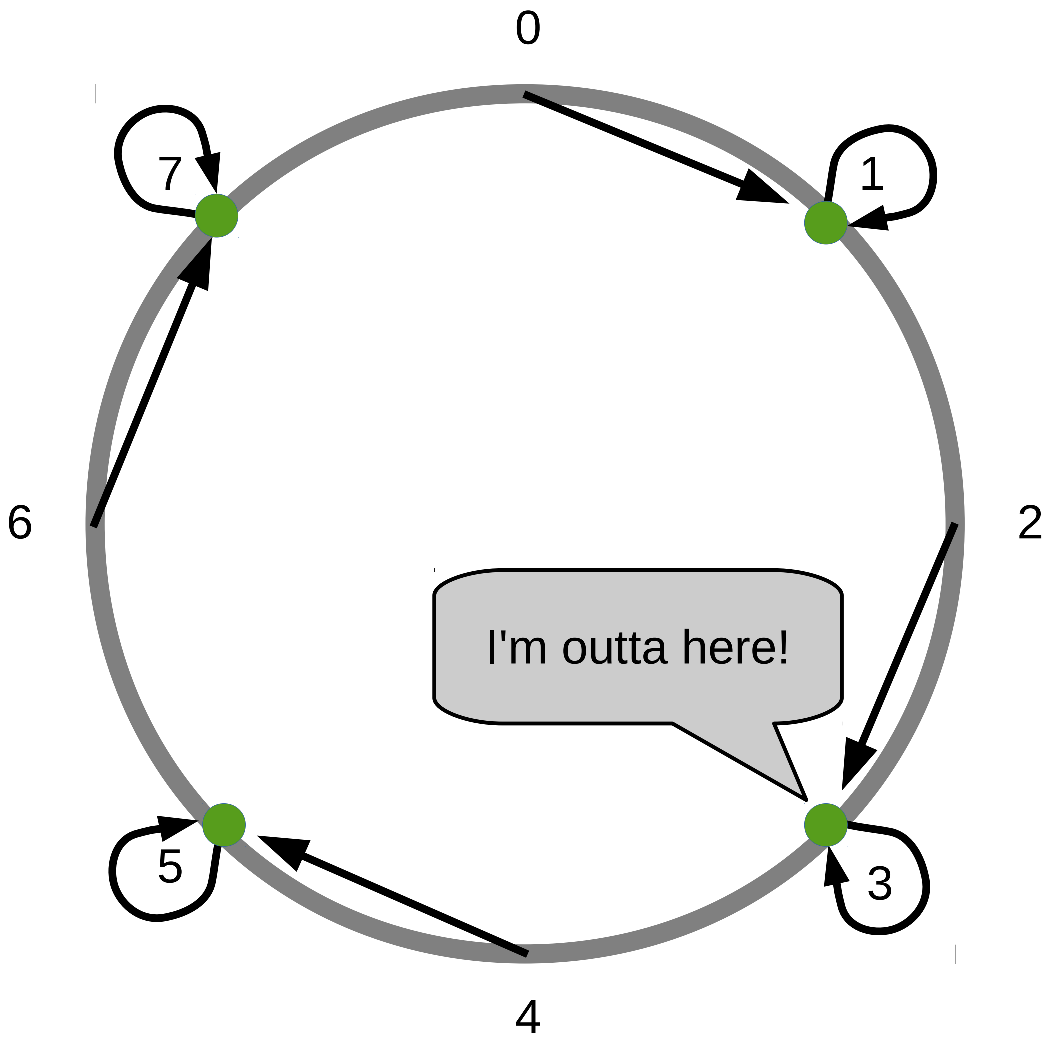
Adding a new node affects only the node that had responsibility of the interval where the new node is added.



Removing nodes

Removing a node affects only those items stored by the leaving node.

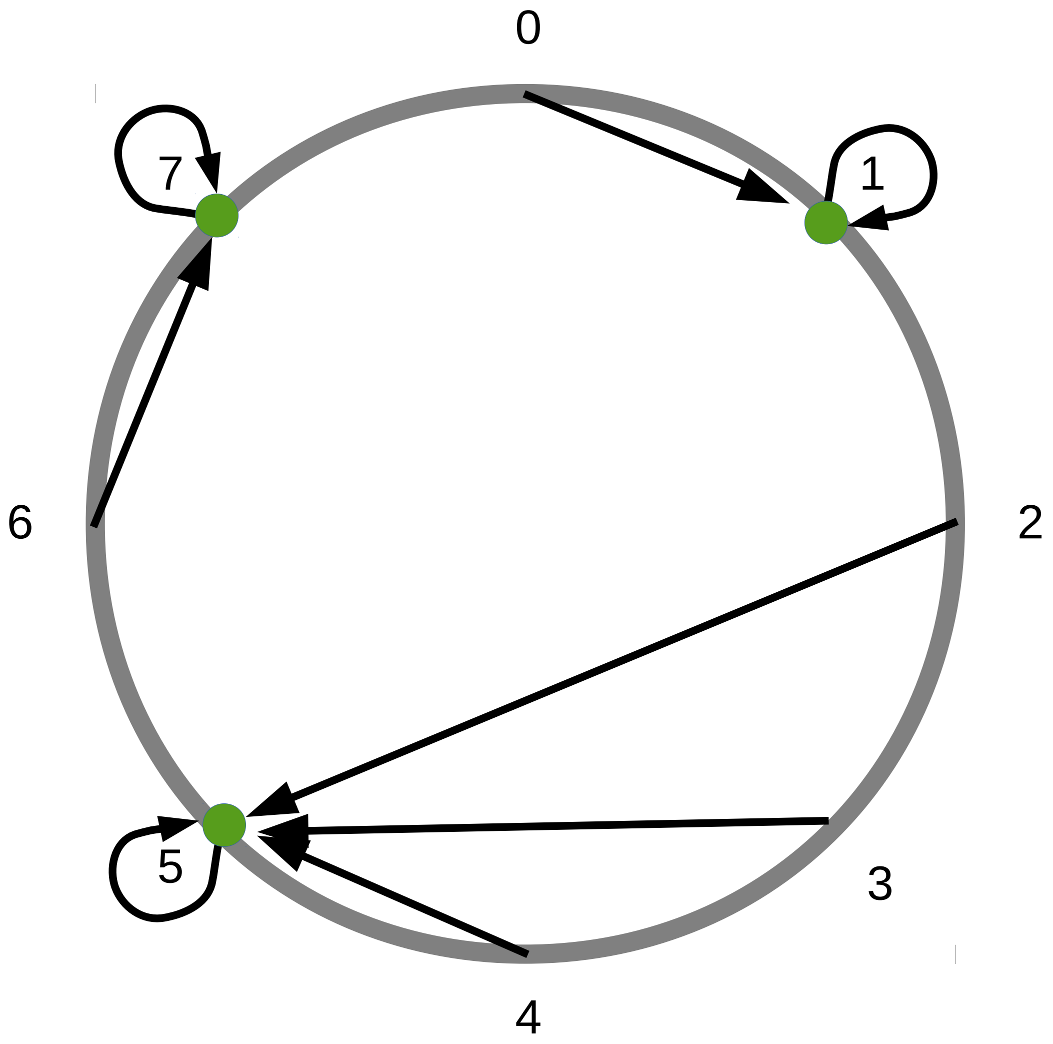
All other responsibilities are left as they were.



Removing nodes

Removing a node affects only those items stored by the leaving node.

All other responsibilities are left as they were.

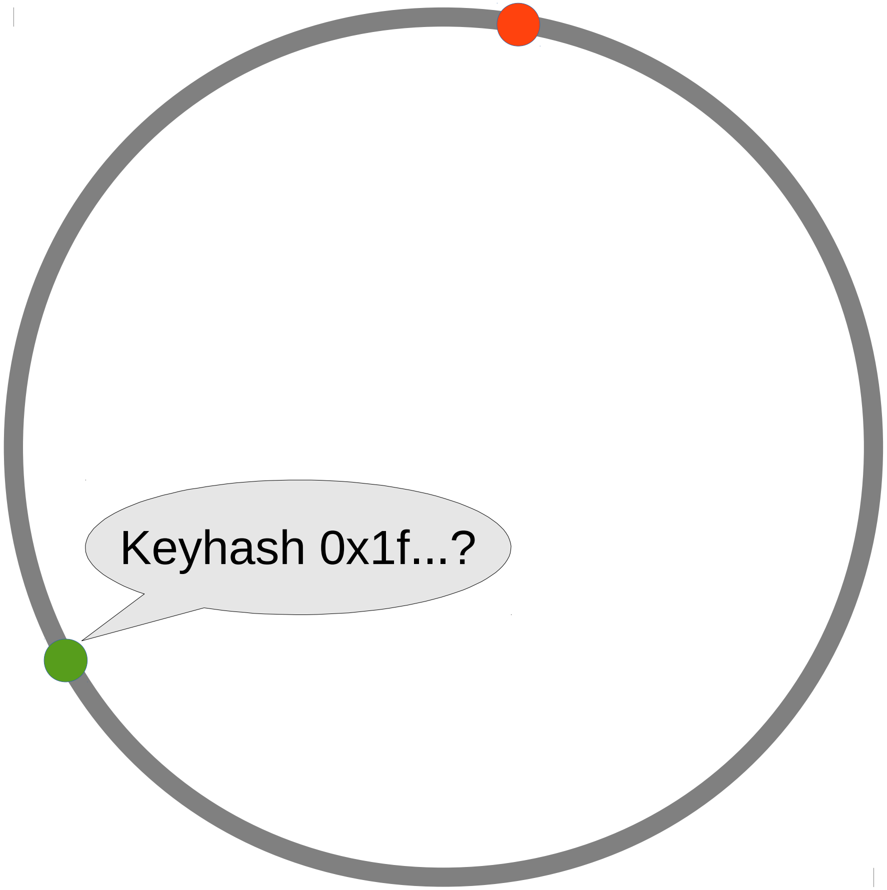


Routing

The problem now is to **locate** the **node** responsible **for a key**:

- using the **lowest number of messages**, but...
- keeping node **memory low**.

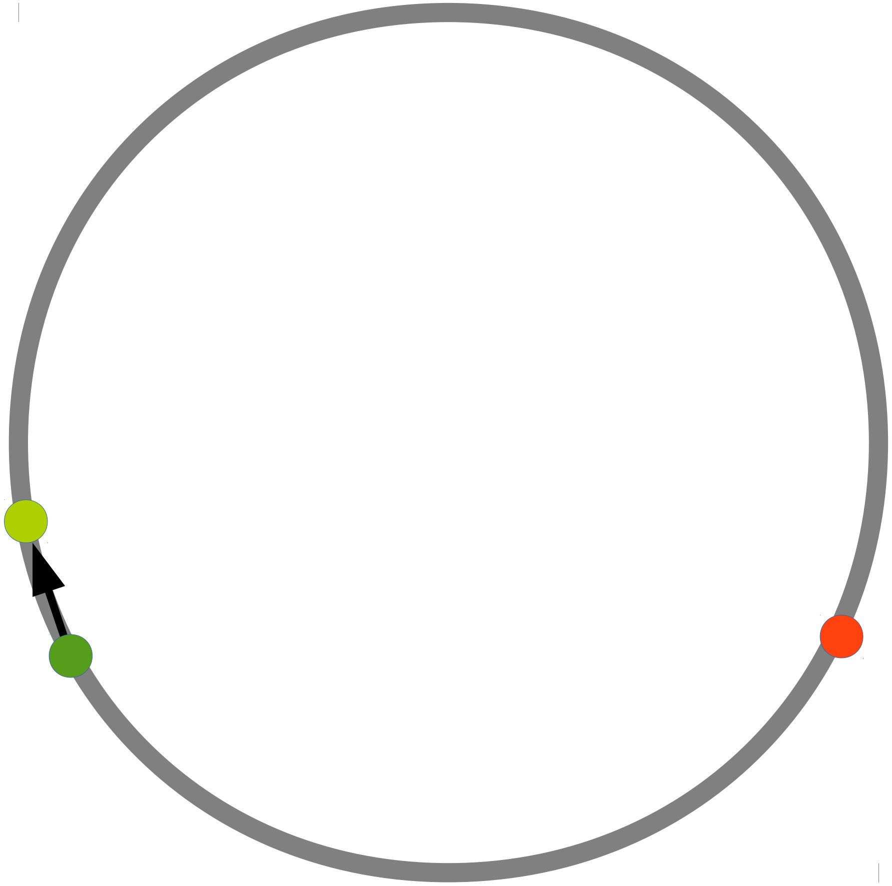
Can be done in a few different ways, depending on **assumptions** and **prioritization**.



Constant Memory Routing

Minimizes the amount of data stored in each node.

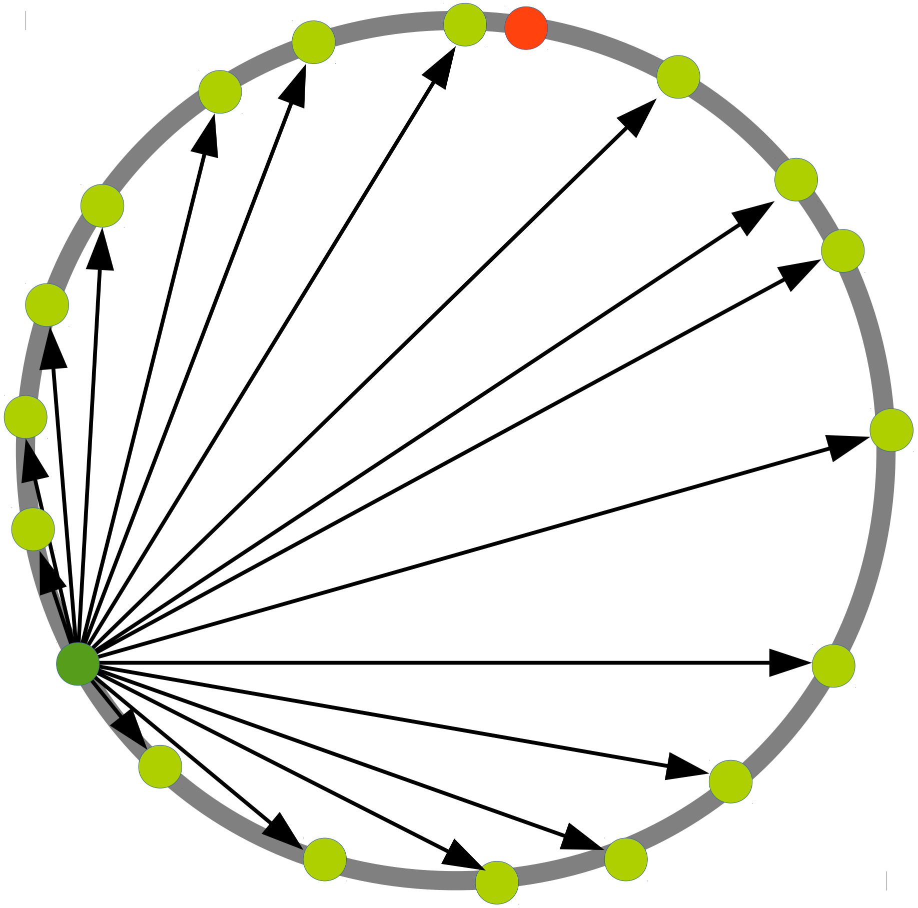
- Nodes know about the next node on the ring
- Step through the ring until we find the right node
- Requires `O(1)` memory in each node
- Requires `O(N)` messages to reach destination
- Not used anywhere to my knowledge, but stated as the worst-case for not yet initialized nodes in Chord [Stoica2001]
- Not feasible for large networks



Constant Time Routing

Minimizes the number of messages required.

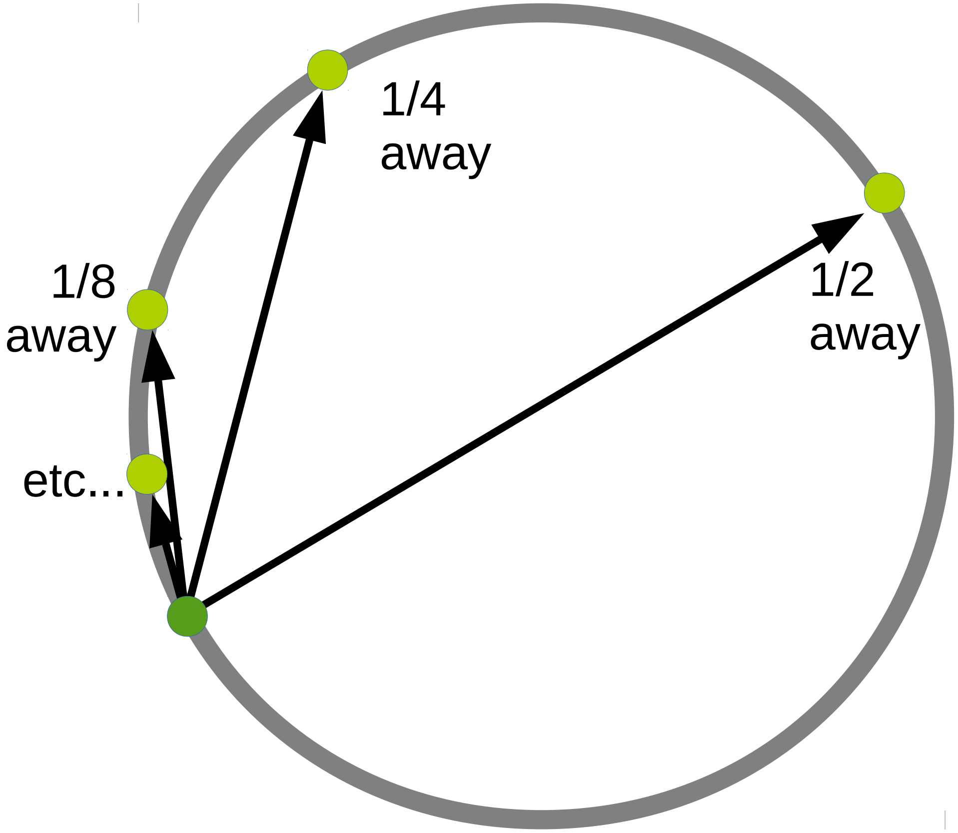
- All nodes have complete knowledge of all other nodes
- Requires `O(1)` messages to reach destination
- Requires `O(N)` memory in each node
- As seen in Amazon Dynamo
- Not feasible for extremely large networks



Logarithmic Routing

The academic solution

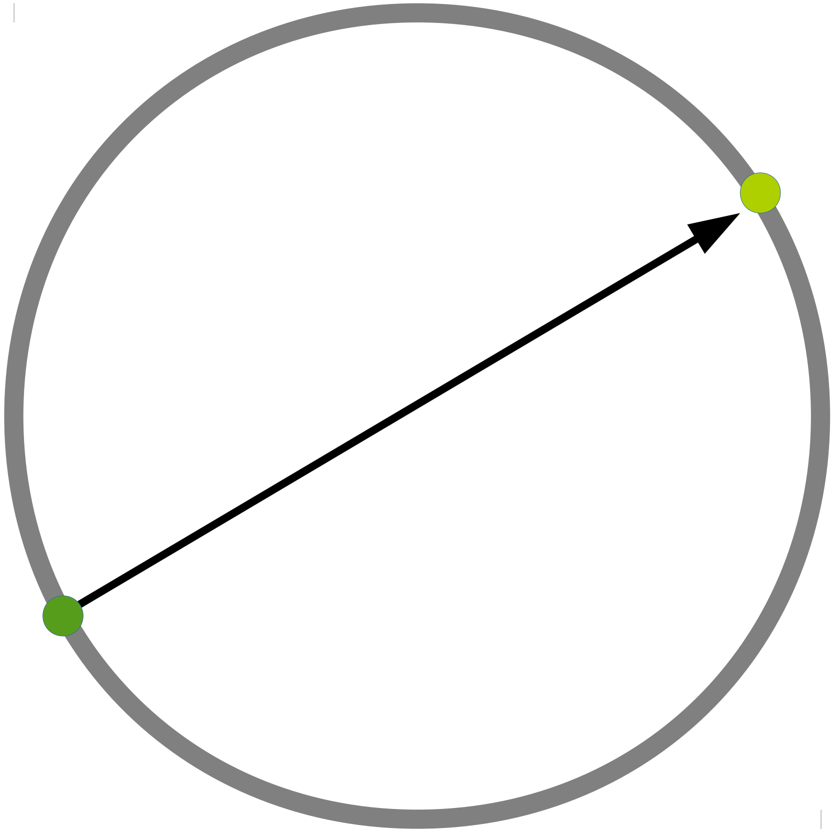
- Keep an updated smart routing-table for efficient node search
- Forwards request to best known node
- Requires `O(log N)` memory in each node
- Requires `O(log N)` messages
- As seen in **Chord** [Stoica2001] and slightly different versions in **Kademlia** [Maymounkov2002] and **Pastry** [Rowstron2001]



O(log N) – Memory Usage

#nodes, N	Routing table size
$2 = 2^1$	1
$4 = 2^2$	2
$8 = 2^3$	3
$16 = 2^4$	4

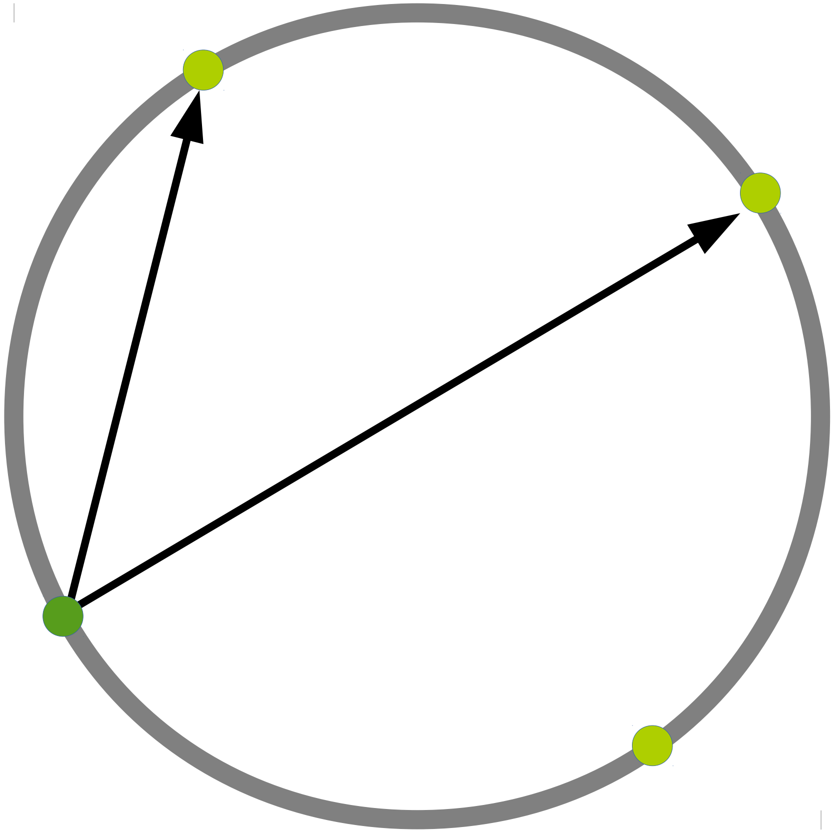
Routing table size = `O(log N)`



O(log N) – Memory Usage

#nodes, N	Routing table size
$2 = 2^1$	1
$4 = 2^2$	2
$8 = 2^3$	3
$16 = 2^4$	4

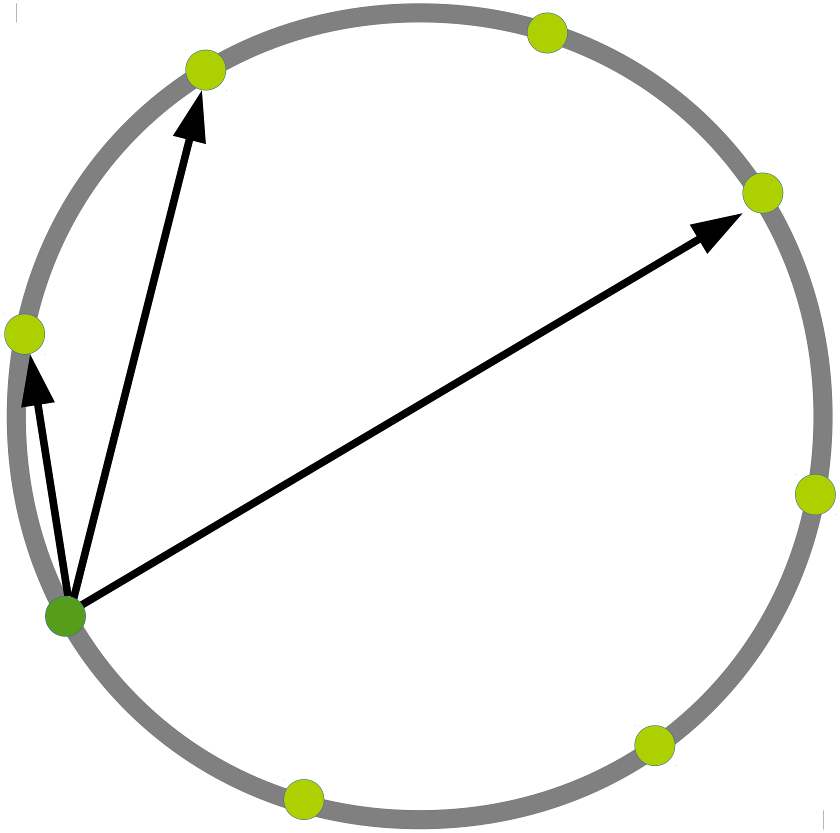
Routing table size = `O(log N)`



O(log N) – Memory Usage

#nodes, N	Routing table size
$2 = 2^1$	1
$4 = 2^2$	2
$8 = 2^3$	3
$16 = 2^4$	4

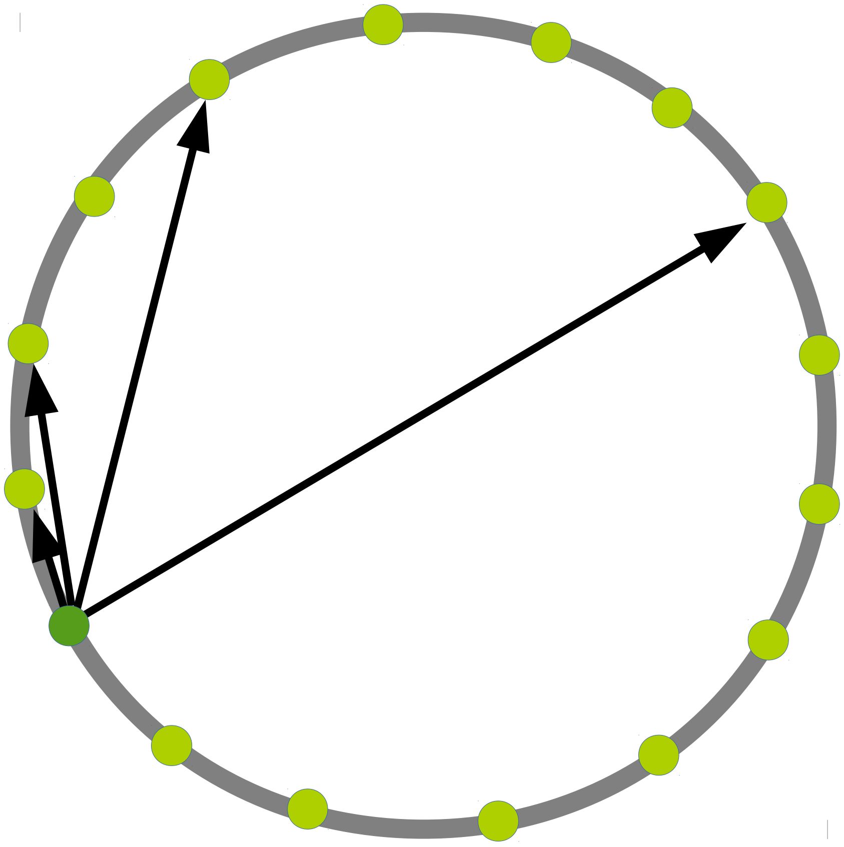
Routing table size = `O(log N)`



O(log N) – Memory Usage

#nodes, N	Routing table size
$2 = 2^1$	1
$4 = 2^2$	2
$8 = 2^3$	3
$16 = 2^4$	4

Routing table size = `O(log N)`

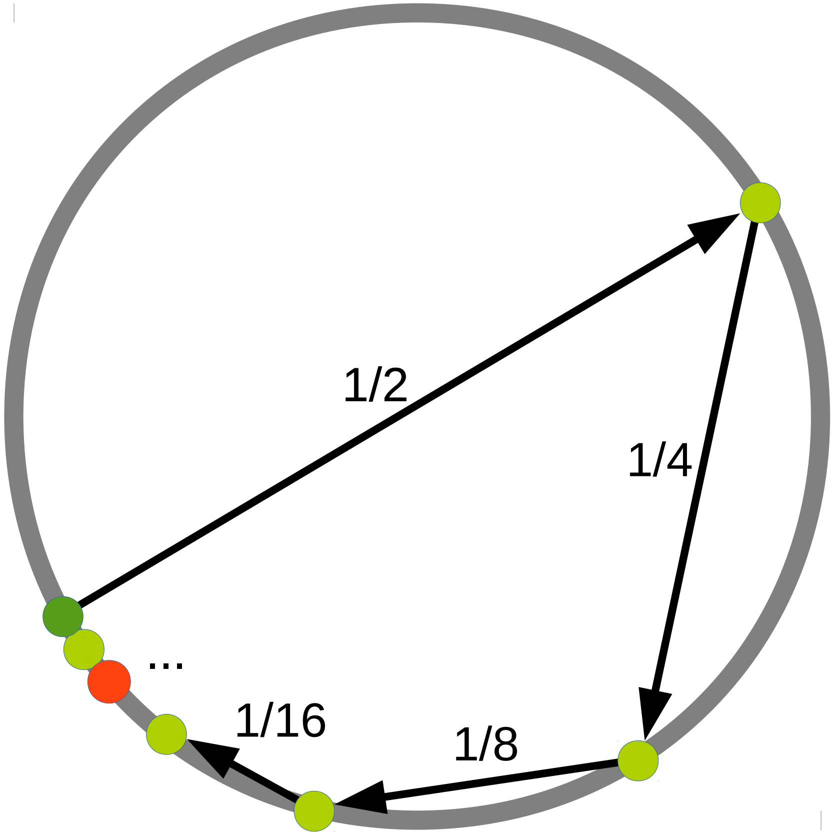


`O(log N)` – Hops

Worst case: Looking for data on a node infinitely close “behind” me.

Each step halves the distance left to the target.

Number of hops = `O(log N)`



Torrent

- File-sharing
- **Files** are split in **chunks**
- Torrent files tell users what chunks they need
- A **central tracker** tells users what user(s) has certain chunks
- The tracker is a **single point of failure**

Torrent DHT

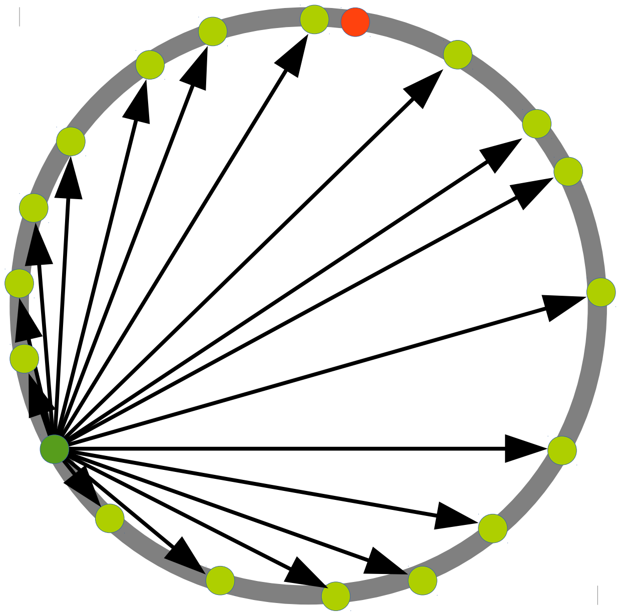
- Introduced in Azureus in 2005. “Mainline DHT” specified by BitTorrent in 2008.
- Each client is a DHT node
- Chunk and user info is inserted in the table
- Using a DHT the torrent protocol becomes tracker-free (no single point of failure!)
- 15-27 million nodes. (Too big for constant time routing?)
- Based on Kademlia published in [Maymounkov2002]

[Measuring Large-Scale Distributed Systems: Case of BitTorrent Mainline DHT, Wang *et al*, 2013]

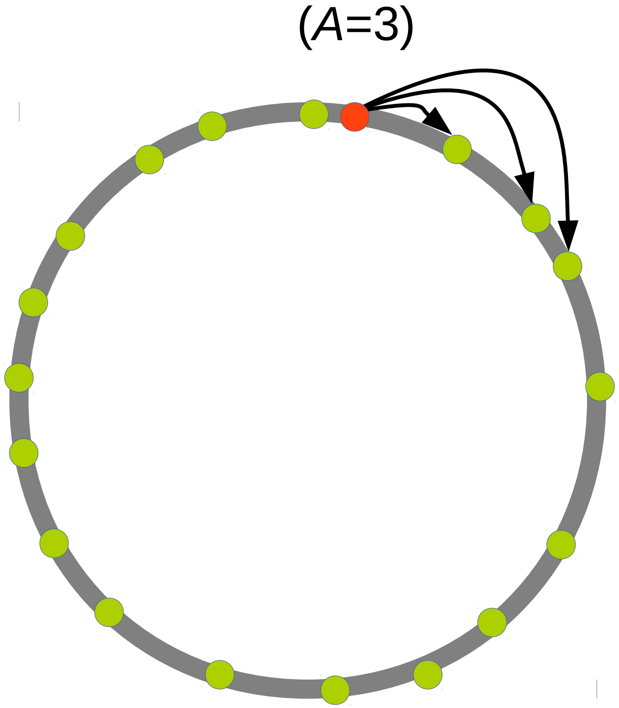
[http://www.bittorrent.org/beps/bep_0005.html]

Dynamo: Amazon's Key-value Store

- Several different internal uses at Amazon, mostly storing state for stateful services, for example the **shopping cart**
- Stores key-value items. Typical value size ~1 MB.
- All nodes have knowledge of all nodes. Storage `O(N)` in each node. Routing takes `O(1)` hops.



Dynamo: Amazon's Key-value Store



- Items replicated over the ***A* nearest nodes**.
- Unavailable nodes can cause diverging replicas. Solved by **versioning** the item updates. Dynamo is **always-writable**!
- Handles temporary failures with **hinted handoff**
- Uses **Merkle trees** to detect lost replicas (differences between nodes with overlapping responsibilities).

[DeCandia2007]

Dynamo: Amazon's Key-value Store

- Items replicated over the ***A* nearest nodes**.
- Unavailable nodes can cause diverging replicas. Solved by **versioning** the item updates. Dynamo is **always-writable**!
- Handles temporary failures with **hinted handoff**
- Uses **Merkle trees** to detect lost replicas (differences between nodes with overlapping responsibilities).

[DeCandia2007]

DHT Security

- Maliciously overwriting data
  - Hard to authenticate nodes in distributed system
- Disturb a node
  - Insert yourself right before a node
  - Change or destroy all the node's data as it is transferred to you
- Take over data
  - Place yourself near data, making you responsible for it
  - Change or destroy data

Amazon Dynamo assumes we are operating in a closed, friendly environment.

Some DHT networks require nodes to choose `nodeid=hash(IP_address)`.

Advantages

- Distributed storage
- Highly scalable (Chord requires routing table size 32 for *N*=2^32)
- Can be made robust against node failures
- Decentralized, no node is unique or irreplaceable
- Self-organizing
- Can take advantage of heterogeneous nodes through virtual nodes

Disadvantages

- Can not search, only look up (typical for hash tables)
- Security

The End