

# Hash Tables: Distributed Hash Tables

Michael Levin

Higher School of Economics

Data Structures  
Data Structures and Algorithms

# Outline

① Online Storage Systems

② Distributed Hash Tables

- Have you ever wondered, how large files are sometimes uploaded instantly to Dropbox?

- Have you ever wondered, how large files are sometimes uploaded instantly to Dropbox?
- Want to know how Dropbox, Google Drive and Yandex Disk save petabytes of storage using the ideas from this module?

- Have you ever wondered, how large files are sometimes uploaded instantly to Dropbox?
- Want to know how Dropbox, Google Drive and Yandex Disk save petabytes of storage using the ideas from this module?
- Interested in distributed storage?

- Have you ever wondered, how large files are sometimes uploaded instantly to Dropbox?
- Want to know how Dropbox, Google Drive and Yandex Disk save petabytes of storage using the ideas from this module?
- Interested in distributed storage?
- This lesson

# Storage optimization



# Storage optimization



Cat.avi





# Storage optimization



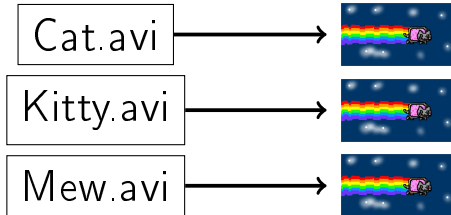
Cat.avi



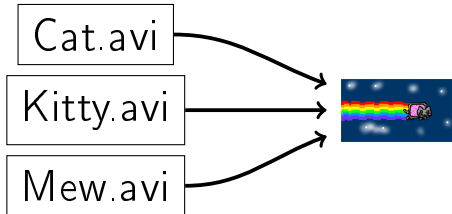
Kitty.avi



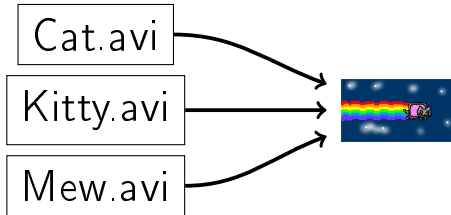
# Storage optimization



# Storage optimization

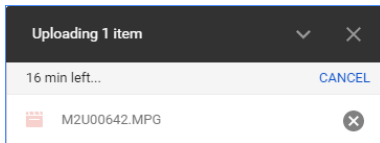


# Storage optimization



66% of storage space saved!

# New File Upload



- Need to determine whether there is already the same file in the system

# Naive Comparison

- Upload new file

# Naive Comparison

- Upload new file
- Go through all stored files

# Naive Comparison

- Upload new file
- Go through all stored files
- Compare each stored file with new file byte-by-byte



# Naive Comparison

- Upload new file
- Go through all stored files
- Compare each stored file with new file byte-by-byte
- If there's the same file, store a link to it instead of the new file

# Drawbacks of Naive Comparison

- Have to upload the file first anyway

# Drawbacks of Naive Comparison

- Have to upload the file first anyway
- $O(NS)$  to compare file of size  $S$  with  $N$  other files

# Drawbacks of Naive Comparison

- Have to upload the file first anyway
- $O(NS)$  to compare file of size  $S$  with  $N$  other files
- $N$  grows, so total running time of uploads grows as  $O(N^2)$

# Idea: Compare Hashes

- As in Rabin-Karp's algorithm, compare hashes of files first

# Idea: Compare Hashes

- As in Rabin-Karp's algorithm, compare hashes of files first
- If hashes are different, files are different

# Idea: Compare Hashes

- As in Rabin-Karp's algorithm, compare hashes of files first
- If hashes are different, files are different
- If there's a file with the same hash, upload and compare directly

# Drawbacks of Hash Comparison

- There can be collisions



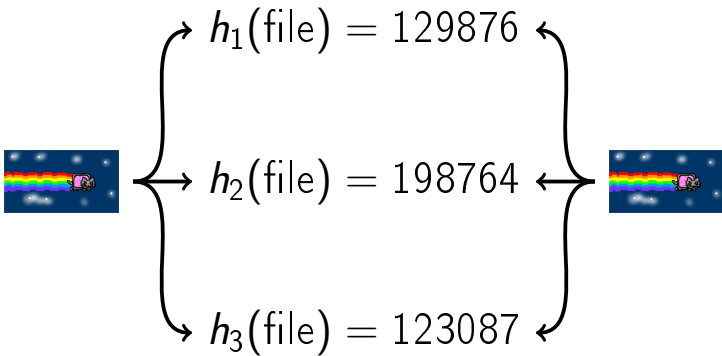
# Drawbacks of Hash Comparison

- There can be collisions
- Still have to upload the file to compare directly

# Drawbacks of Hash Comparison

- There can be collisions
- Still have to upload the file to compare directly
- Still have to compare with all  $N$  stored files

# Idea: Several Hashes



# Idea: Several Hashes

- Choose several different hash functions

# Idea: Several Hashes

- Choose several different hash functions
- Polynomial hashing with different  $p$  or  $x$

# Idea: Several Hashes

- Choose several different hash functions
- Polynomial hashing with different  $p$  or  $x$
- Compute all hashes for each file

# Idea: Several Hashes

- Choose several different hash functions
- Polynomial hashing with different  $p$  or  $x$
- Compute all hashes for each file
- If there's a file with all the same hashes, files are probably equal

# Idea: Several Hashes

- Choose several different hash functions
- Polynomial hashing with different  $p$  or  $x$
- Compute all hashes for each file
- If there's a file with all the same hashes, files are probably equal
- Don't upload the new file in this case!



# Idea: Several Hashes

- Choose several different hash functions
- Polynomial hashing with different  $p$  or  $x$
- Compute all hashes for each file
- If there's a file with all the same hashes, files are probably equal
- Don't upload the new file in this case!
- Compute hashes locally before upload

# Problem: Collisions

- Collisions can happen even for several hashes simultaneously

# Problem: Collisions

- Collisions can happen even for several hashes simultaneously
- There are algorithms to find collisions for known hash functions

# Problem: Collisions

- Collisions can happen even for several hashes simultaneously
- There are algorithms to find collisions for known hash functions
- However, even for one hash function collisions are extremely rare

# Problem: Collisions

- Collisions can happen even for several hashes simultaneously
- There are algorithms to find collisions for known hash functions
- However, even for one hash function collisions are extremely rare
- Using 3 or 5 hashes, you probably won't see a collision in a lifetime

# Problem: $O(N)$ Comparisons

- Still have to compare with  $N$  already stored files

# Idea: Precompute Hashes

- When a file is submitted for upload, hashes are computed anyway

# Idea: Precompute Hashes

- When a file is submitted for upload, hashes are computed anyway
- Store file addresses in a hash table



# Idea: Precompute Hashes

- When a file is submitted for upload, hashes are computed anyway
- Store file addresses in a hash table
- Also store all the hashes there

# Idea: Precompute Hashes

- When a file is submitted for upload, hashes are computed anyway
- Store file addresses in a hash table
- Also store all the hashes there
- Only need the hashes to search in the table

# Final Solution

- Choose 3 – 5 hash functions

# Final Solution

- Choose 3 – 5 hash functions
- Store file addresses and hashes in a hash table

# Final Solution

- Choose 3 – 5 hash functions
- Store file addresses and hashes in a hash table
- Compute the hashes of new file locally before upload

# Final Solution

- Choose 3 – 5 hash functions
- Store file addresses and hashes in a hash table
- Compute the hashes of new file locally before upload
- Search new file in the hash table

# Final Solution

- Choose 3 – 5 hash functions
- Store file addresses and hashes in a hash table
- Compute the hashes of new file locally before upload
- Search new file in the hash table
- Search is successful if all the hashes coincide

# Final Solution

- Choose 3 – 5 hash functions
- Store file addresses and hashes in a hash table
- Compute the hashes of new file locally before upload
- Search new file in the hash table
- Search is successful if all the hashes coincide
- Don't upload the file, store a link to the existing one



# More Problems

- Billions of files are uploaded daily

# More Problems

- Billions of files are uploaded daily
- Trillions stored already

# More Problems

- Billions of files are uploaded daily
- Trillions stored already
- Too big for a simple hash table

# More Problems

- Billions of files are uploaded daily
- Trillions stored already
- Too big for a simple hash table
- Millions of users upload simultaneously

# More Problems

- Billions of files are uploaded daily
- Trillions stored already
- Too big for a simple hash table
- Millions of users upload simultaneously
- Too many requests for a single table

# More Problems

- Billions of files are uploaded daily
- Trillions stored already
- Too big for a simple hash table
- Millions of users upload simultaneously
- Too many requests for a single table
- See the next lecture

# Outline

① Online Storage Systems

② Distributed Hash Tables

# Big Data

- Need to store trillions or more objects



# Big Data

- Need to store trillions or more objects
- File addresses, user profiles, e-mails

# Big Data

- Need to store trillions or more objects
- File addresses, user profiles, e-mails
- Need fast search/access

# Big Data

- Need to store trillions or more objects
- File addresses, user profiles, e-mails
- Need fast search/access
- Hash tables provide  $O(1)$  search/access on average, but for  $n = 10^{12}$ ,  $O(n + m)$  memory becomes too big

# Big Data

- Need to store trillions or more objects
- File addresses, user profiles, e-mails
- Need fast search/access
- Hash tables provide  $O(1)$  search/access on average, but for  $n = 10^{12}$ ,  $O(n + m)$  memory becomes too big
- Solution: distributed hash tables

# Distributed Hash Table

- Get 1000 computers

# Distributed Hash Table

- Get 1000 computers
- Create a hash table on each of them

# Distributed Hash Table

- Get 1000 computers
- Create a hash table on each of them
- Determine which computer “owns” object  $O$ : number  $h(O) \bmod 1000$

# Distributed Hash Table

- Get 1000 computers
- Create a hash table on each of them
- Determine which computer “owns” object  $O$ : number  $h(O) \bmod 1000$
- Send request to that computer, search/modify in the local hash table



# Problems

- Computers sometimes break

# Problems

- Computers sometimes break
- Computer breaks once in 2 years  $\Rightarrow$  one of 1000 computers breaks every day!

# Problems

- Computers sometimes break
- Computer breaks once in 2 years  $\Rightarrow$  one of 1000 computers breaks every day!
- Store several copies of the data

# Problems

- Computers sometimes break
- Computer breaks once in 2 years  $\Rightarrow$  one of 1000 computers breaks every day!
- Store several copies of the data
- Need to relocate the data from the broken computer

# Problems

- Computers sometimes break
- Computer breaks once in 2 years  $\Rightarrow$  one of 1000 computers breaks every day!
- Store several copies of the data
- Need to relocate the data from the broken computer
- Service grows, and new computers are added to the cluster

# Problems

- Computers sometimes break
- Computer breaks once in 2 years  $\Rightarrow$  one of 1000 computers breaks every day!
- Store several copies of the data
- Need to relocate the data from the broken computer
- Service grows, and new computers are added to the cluster
- $h(O) \bmod 1000$  no longer works

# Consistent Hashing

- Choose hash function  $h$  with cardinality  $m$  and put numbers from 0 to  $m - 1$  on a circle clockwise

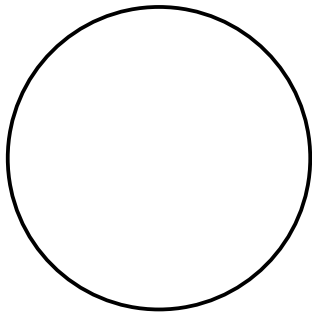
# Consistent Hashing

- Choose hash function  $h$  with cardinality  $m$  and put numbers from 0 to  $m - 1$  on a circle clockwise
- Each object  $O$  is then mapped to a point on the circle with number  $h(O)$

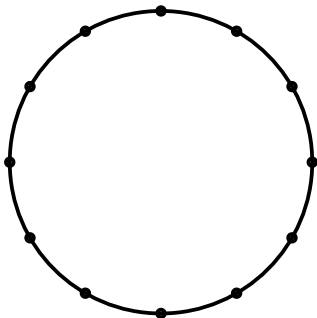


# Consistent Hashing

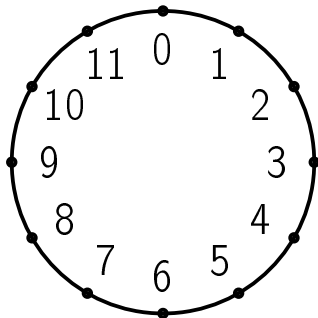
- Choose hash function  $h$  with cardinality  $m$  and put numbers from 0 to  $m - 1$  on a circle clockwise
- Each object  $O$  is then mapped to a point on the circle with number  $h(O)$
- Map computer IDs to the same circle:  $compID \rightarrow$  point number  $h(compID)$



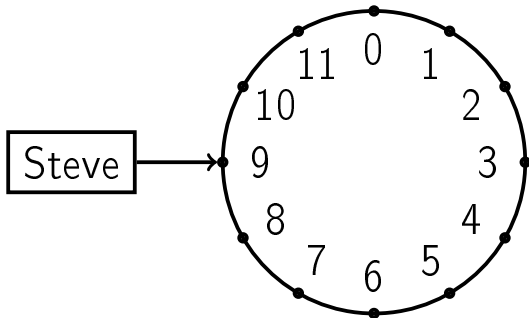
$$m = 12$$



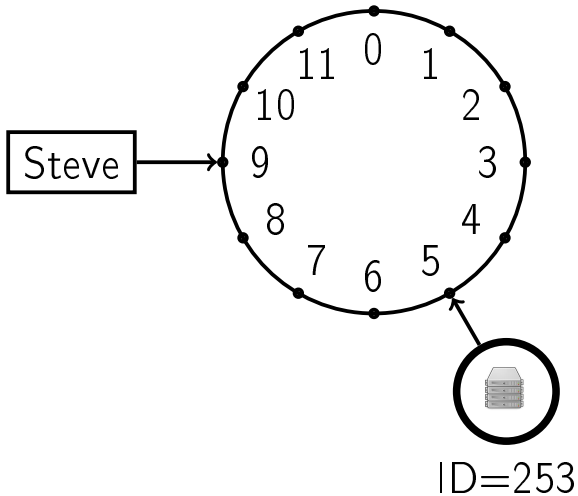
$$m = 12$$



$$m = 12$$



$$m = 12$$



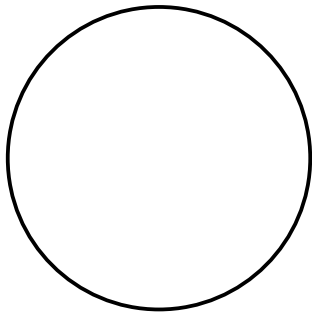
# Consistent Hashing

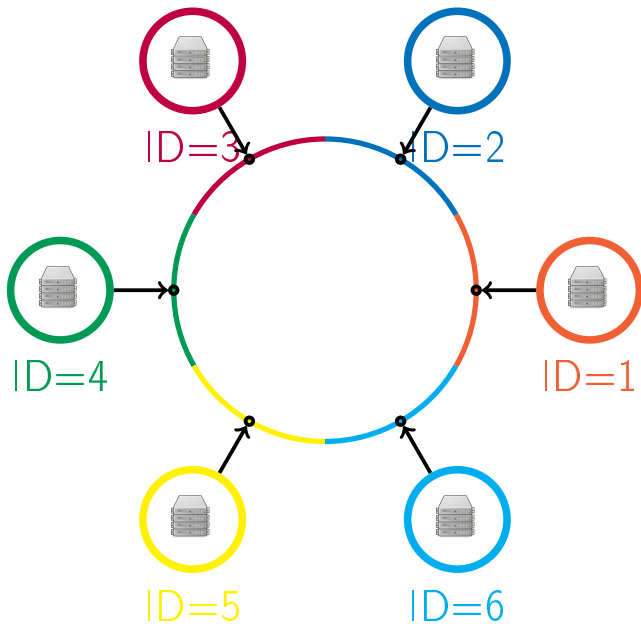
- Each object is stored on the “closest” computer

# Consistent Hashing

- Each object is stored on the “closest” computer
- Each computer stores all objects falling on some arc of the circle





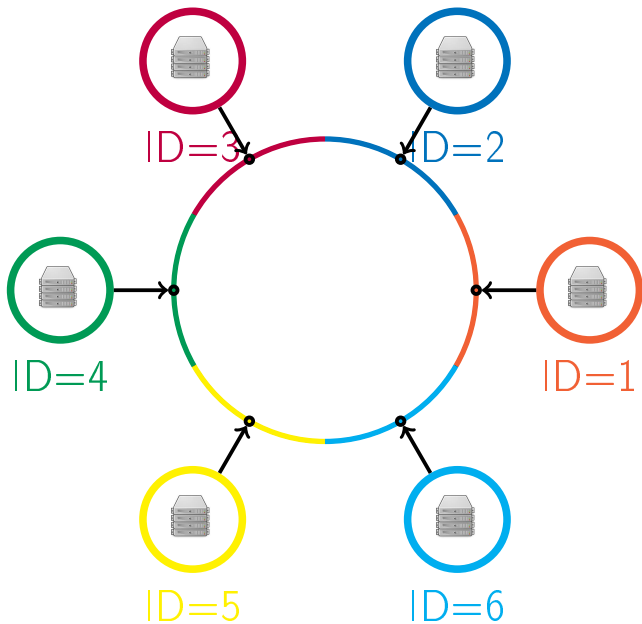


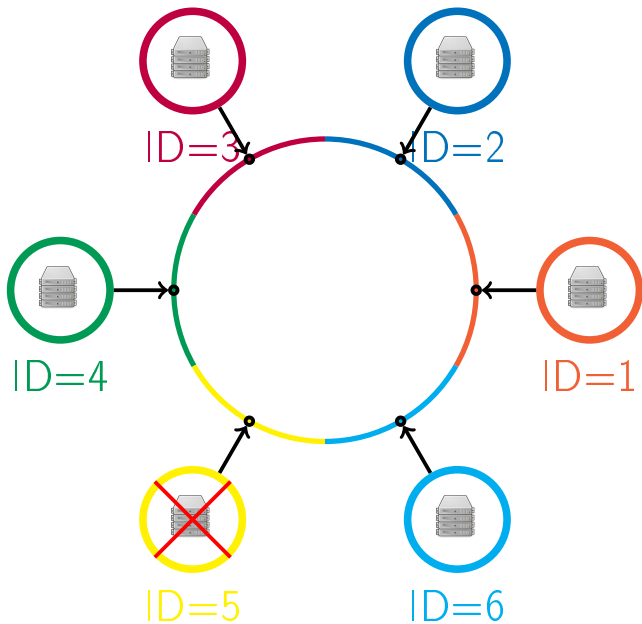
# Computers In and Out

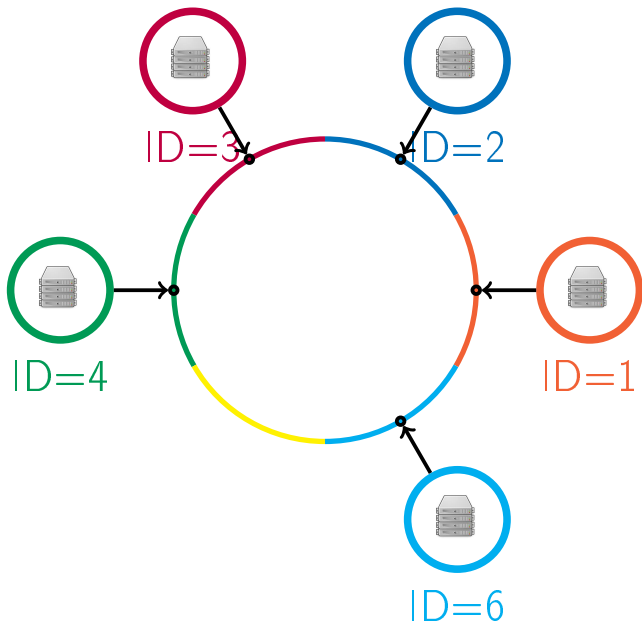
- When a computer goes off, “neighbors” take its data

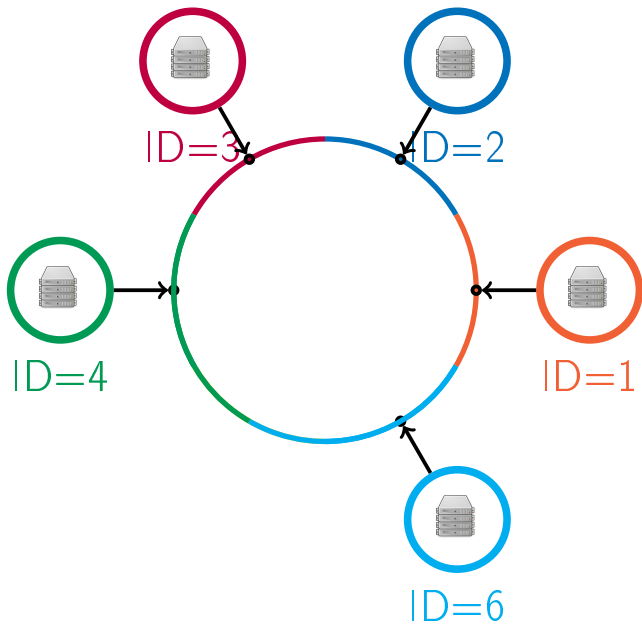
# Computers In and Out

- When a computer goes off, “neighbors” take its data
- When a new computer is added, it takes data from the “neighbors”











# Overlay Network

- Need to copy/relocate data

# Overlay Network

- Need to copy/relocate data
- How will a node know where to send the data?

# Overlay Network

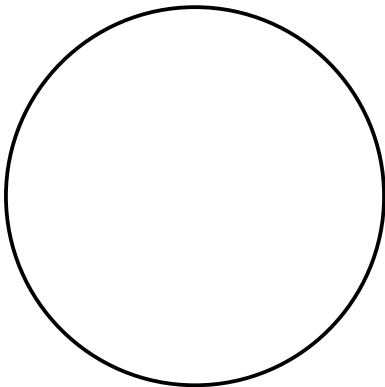
- Need to copy/relocate data
- How will a node know where to send the data?
- Each node will know a few neighbors

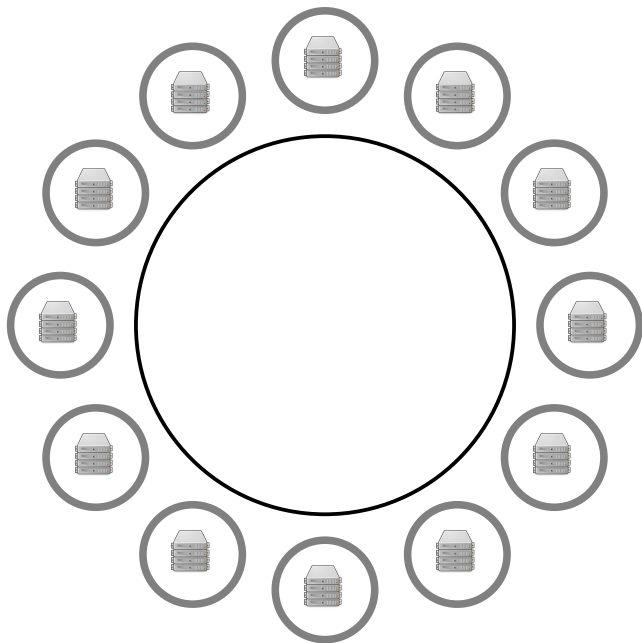
# Overlay Network

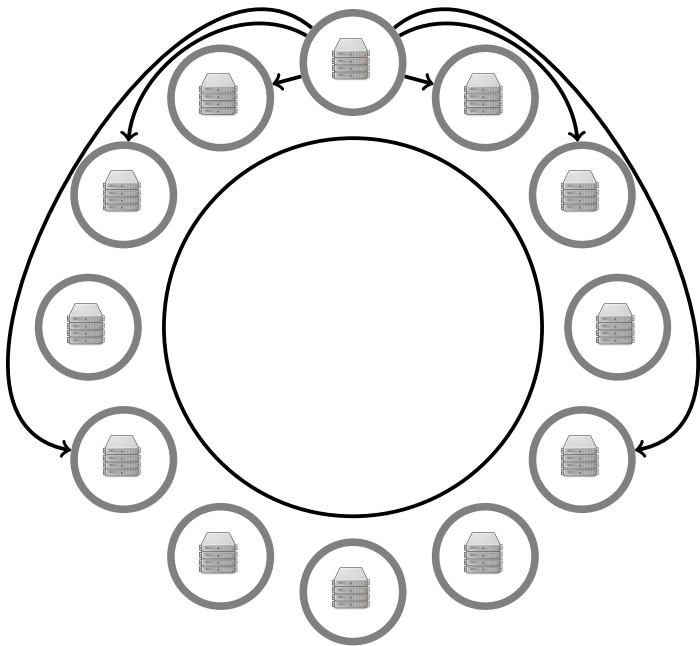
- Need to copy/relocate data
- How will a node know where to send the data?
- Each node will know a few neighbors
- For each key, each node will either store it or know some node “closer” to this key

# Overlay Network

- Need to copy/relocate data
- How will a node know where to send the data?
- Each node will know a few neighbors
- For each key, each node will either store it or know some node “closer” to this key
- E.g., each node knows neighbors,  $\pm 1, \pm 2, \pm 4, \pm 8, \dots$  —  $O(\log n)$  nodes, and can get/send any key in  $O(\log n)$









# Conclusion

- Distributed Hash Tables (DHT) store Big Data on many computers
- Consistent Hashing (CH) is one way to determine which computer stores which data
- CH uses mapping of keys and computer IDs on a circle
- Each computer stores a range of keys
- Overlay Network is used to route the data to/from the right computer