Application Layer
Part 6

Mark Allman
Case / ICSI

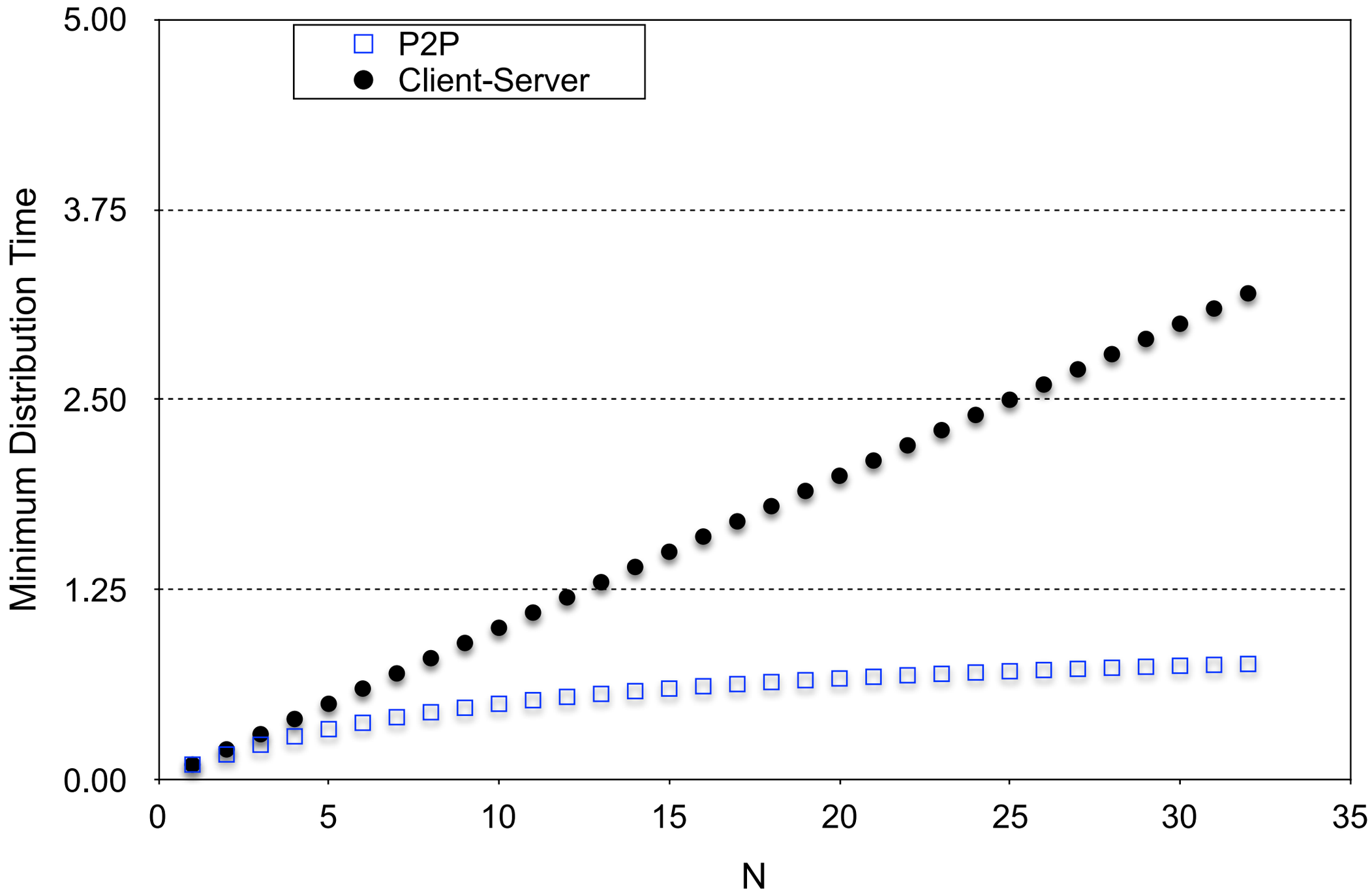
EECS 325/425
Fall 2018

"Now don't you call James Bond or Secret Agent Man,
'Cause they can't do it Like I Can …"

Many of these slides are more-or-less directly from the slide set developed by Jim Kurose and Keith Ross for their book “Computer Networking: A Top Down Approach, 5th edition”.

The slides have been lightly adapted for Mark Allman’s EECS 325/425 Computer Networks class at Case Western Reserve University.

Server-client vs. P2P: example



File distribution: BitTorrent

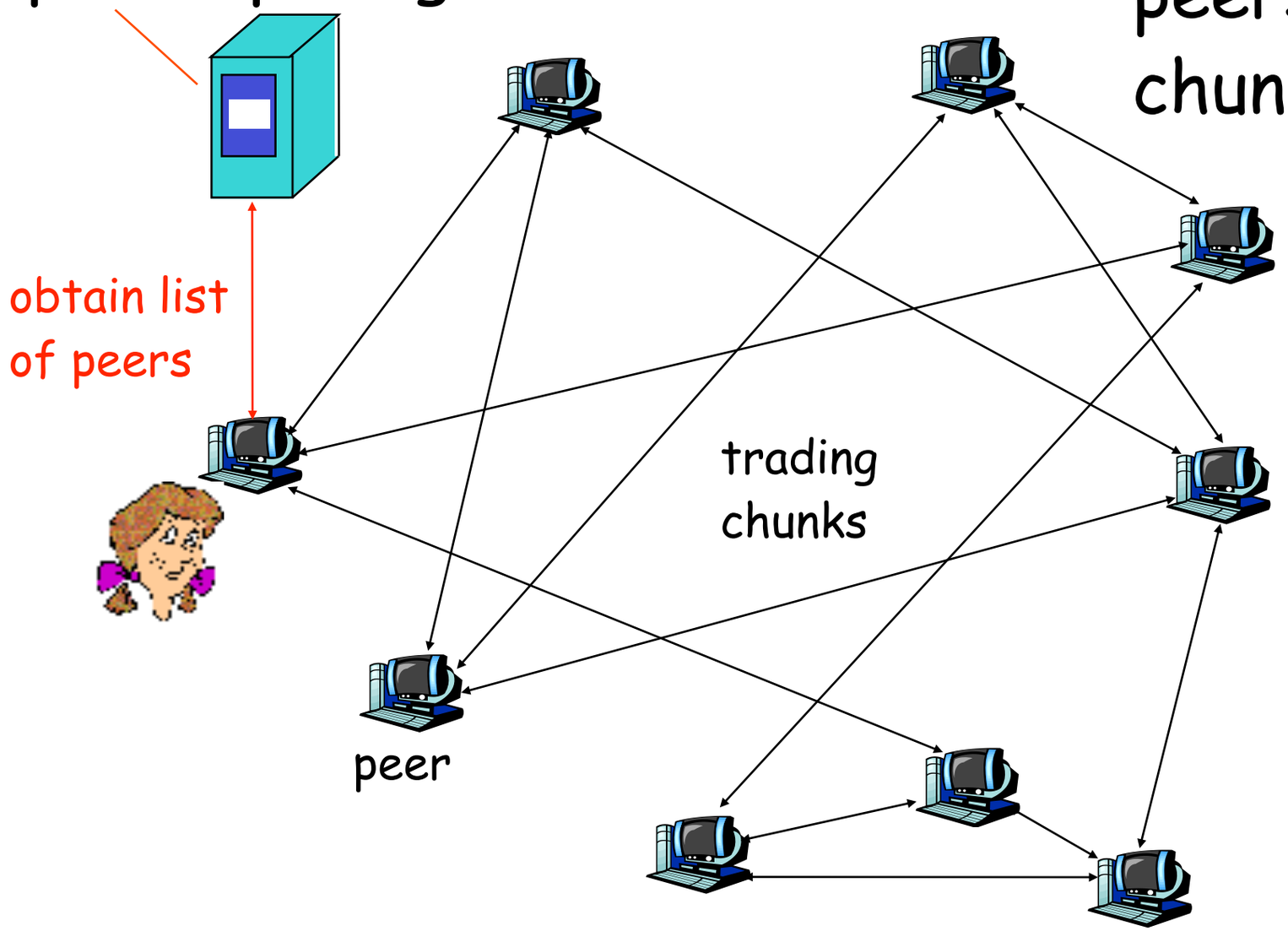
P2P file distribution

File distribution: BitTorrent

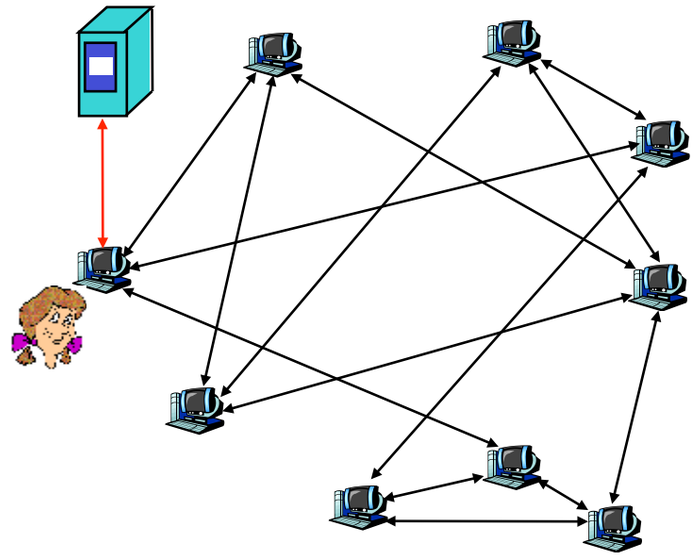
P2P file distribution

tracker: tracks peers participating in torrent

torrent: group of peers exchanging chunks of a file

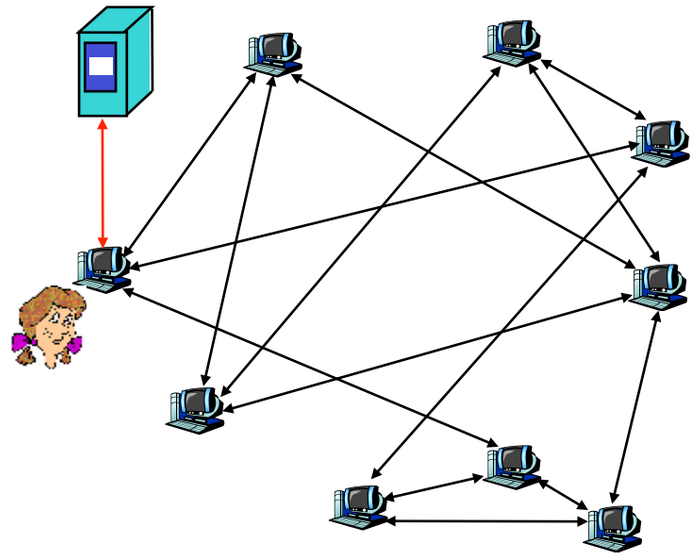


BitTorrent (1)

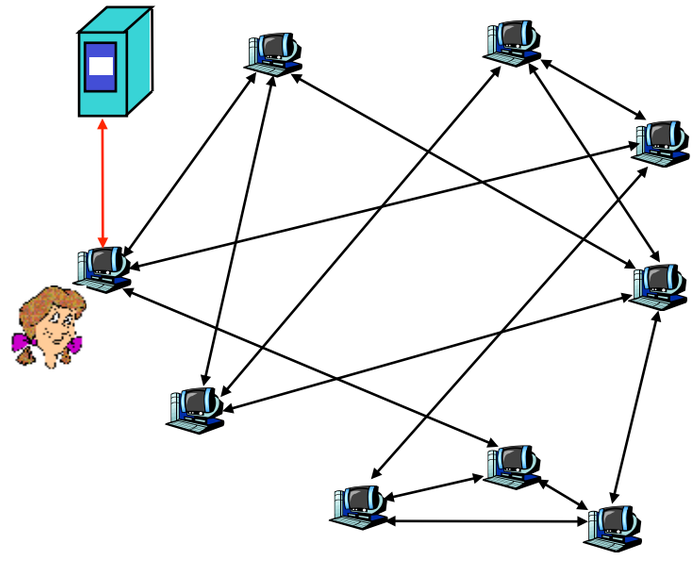


BitTorrent (1)

- file divided into 256KB chunks.

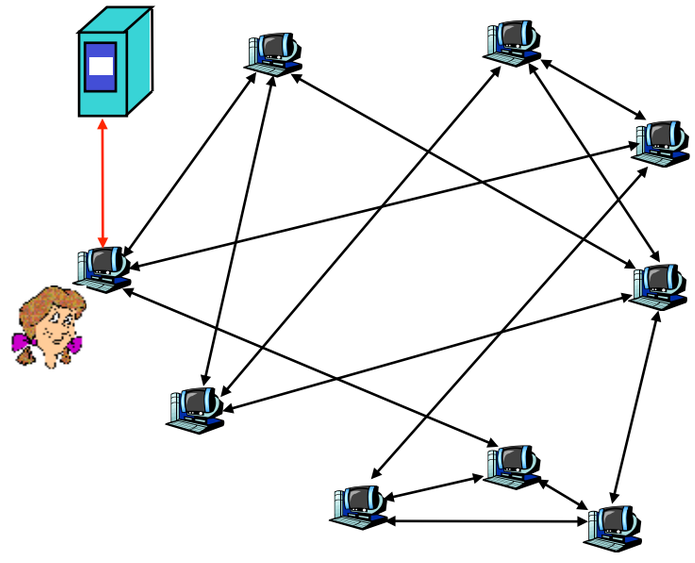


BitTorrent (1)



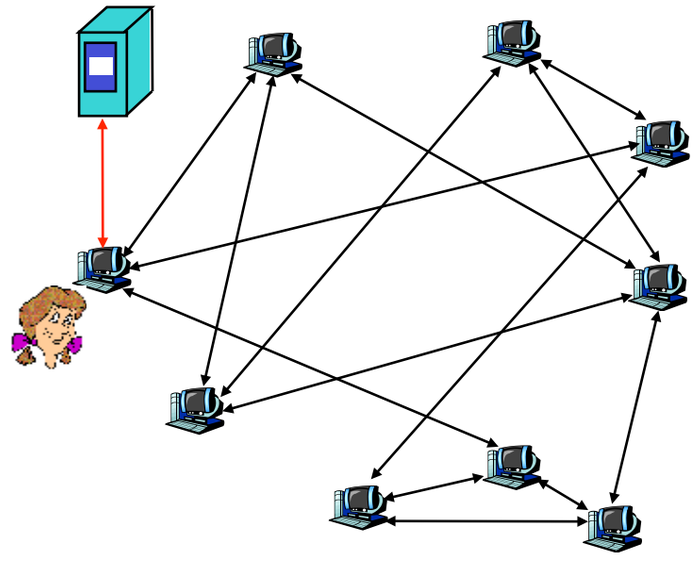
- file divided into 256KB chunks.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")

BitTorrent (1)



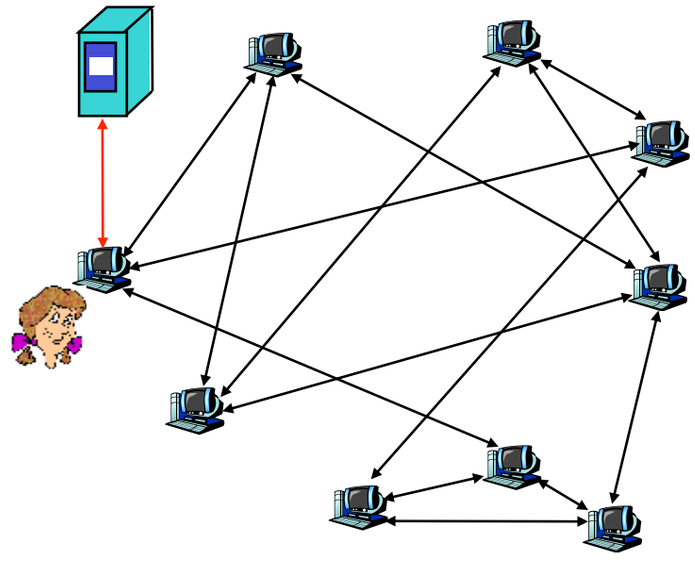
- file divided into 256KB chunks.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.

BitTorrent (1)



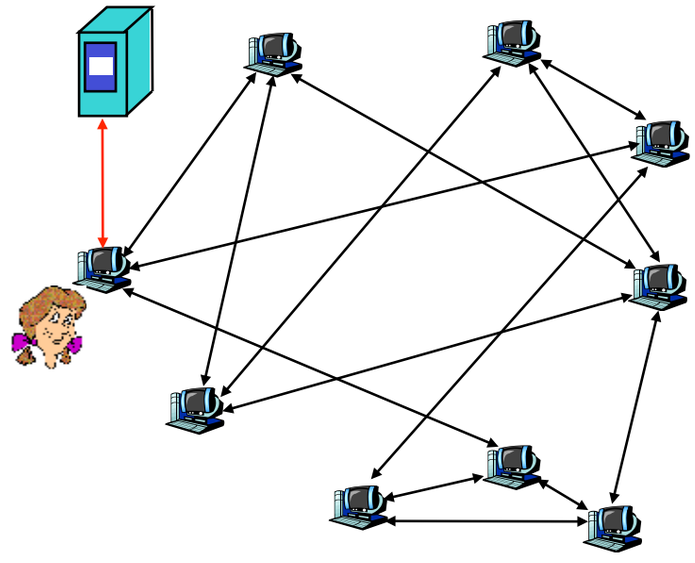
- file divided into 256KB chunks.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go

BitTorrent (1)



- file divided into 256KB chunks.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, … ?

BitTorrent (1)



- file divided into 256KB chunks.
- peer joining torrent:
  - has no chunks, but will accumulate them over time
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")
- while downloading, peer uploads chunks to other peers.
- peers may come and go
- once peer has entire file, it may (selfishly) leave or (altruistically) remain

BitTorrent (2)

BitTorrent (2)

Pulling Chunks

BitTorrent (2)

Pulling Chunks

- at any given time, different peers have different subsets of file chunks

BitTorrent (2)

Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.

BitTorrent (2)

Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

BitTorrent (2)

Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

BitTorrent (2)

Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

- Alice sends chunks to four neighbors currently sending her chunks at the highest rate
  - re-evaluate top 4 every 10 secs

BitTorrent (2)

Pulling Chunks

- at any given time, different peers have different subsets of file chunks
- periodically, a peer (Alice) asks each neighbor for list of chunks that they have.
- Alice sends requests for her missing chunks
  - rarest first

Sending Chunks: tit-for-tat

- Alice sends chunks to four neighbors currently sending her chunks at the highest rate
  - re-evaluate top 4 every 10 secs
- every 30 secs: randomly select another peer, starts sending chunks
  - newly chosen peer may join top 4
  - "optimistically unchoke"

BitTorrent (3)

- rough sketch

- why 30sec? why 256KB? why 4 neighbors?

Distributed Hash Table (DHT)

Distributed Hash Table (DHT)

- DHT: distributed (via P2P) data structure

Distributed Hash Table (DHT)

- DHT: distributed (via P2P) data structure

- data structure has (key, value) pairs;
  - key: ss number; value: human name

Distributed Hash Table (DHT)

- DHT: distributed (via P2P) data structure

- data structure has (key, value) pairs;
  - key: ss number; value: human name

- peers query data structure with key
  - value that match the key is returned

Distributed Hash Table (DHT)

- DHT: distributed (via P2P) data structure

- data structure has (key, value) pairs;
  - key: ss number; value: human name

- peers query data structure with key
  - value that match the key is returned

- peers can also insert (key, value) peers

DHT Identifiers

DHT Identifiers

- assign integer identifier to each peer in range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.

DHT Identifiers

- assign integer identifier to each peer in range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.
- require each key to be an integer in same range.

DHT Identifiers

- assign integer identifier to each peer in range $[0,2^n-1]$.
  - Each identifier can be represented by n bits.
- require each key to be an integer in same range.
- to get integer keys, hash original key.
  - e.g., key = h("Led Zeppelin IV")
  - this is why they call it a distributed "hash" table

How to assign keys to peers?

How to assign keys to peers?

- central issue:
  - assigning (key, value) pairs to peers.

How to assign keys to peers?

- central issue:
  - assigning (key, value) pairs to peers.
- rule: assign key to the peer that has the closest ID.

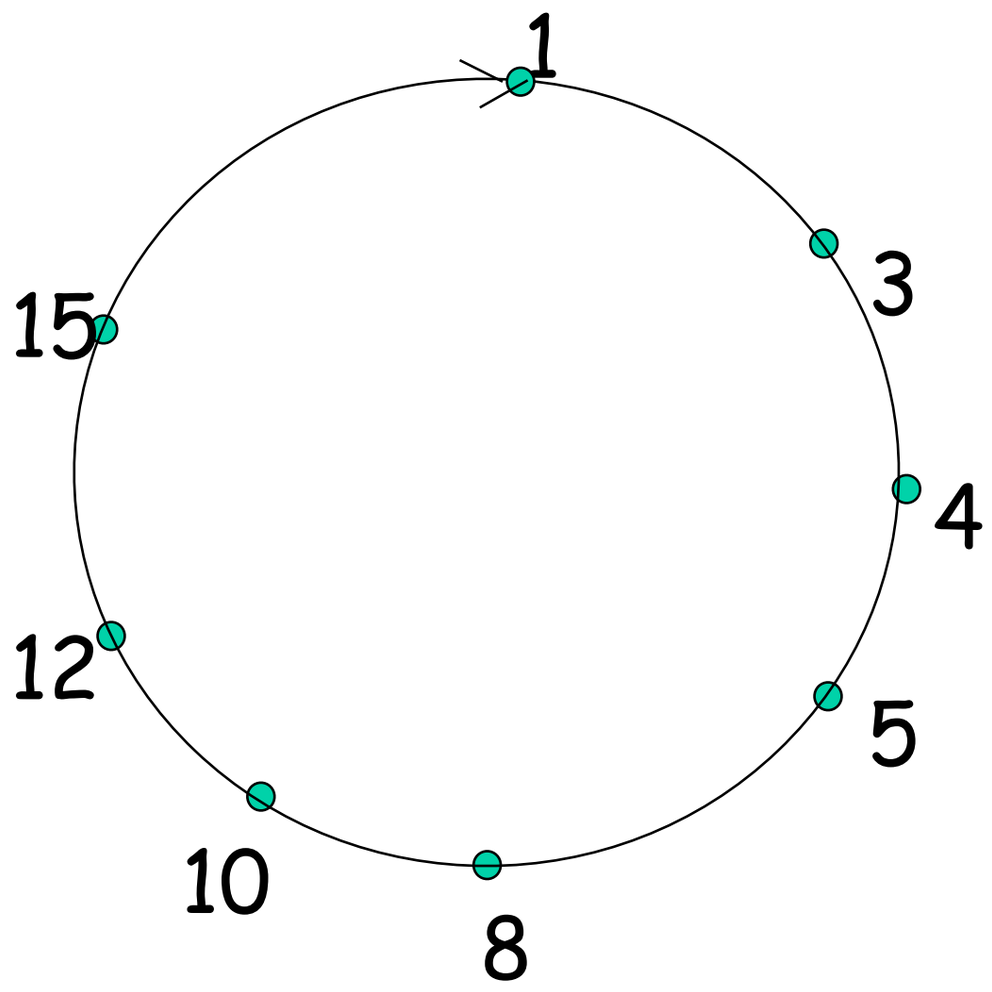
How to assign keys to peers?

- central issue:
  - assigning (key, value) pairs to peers.
- rule: assign key to the peer that has the closest ID.
- convention in lecture: closest is the immediate successor of the key.

How to assign keys to peers?

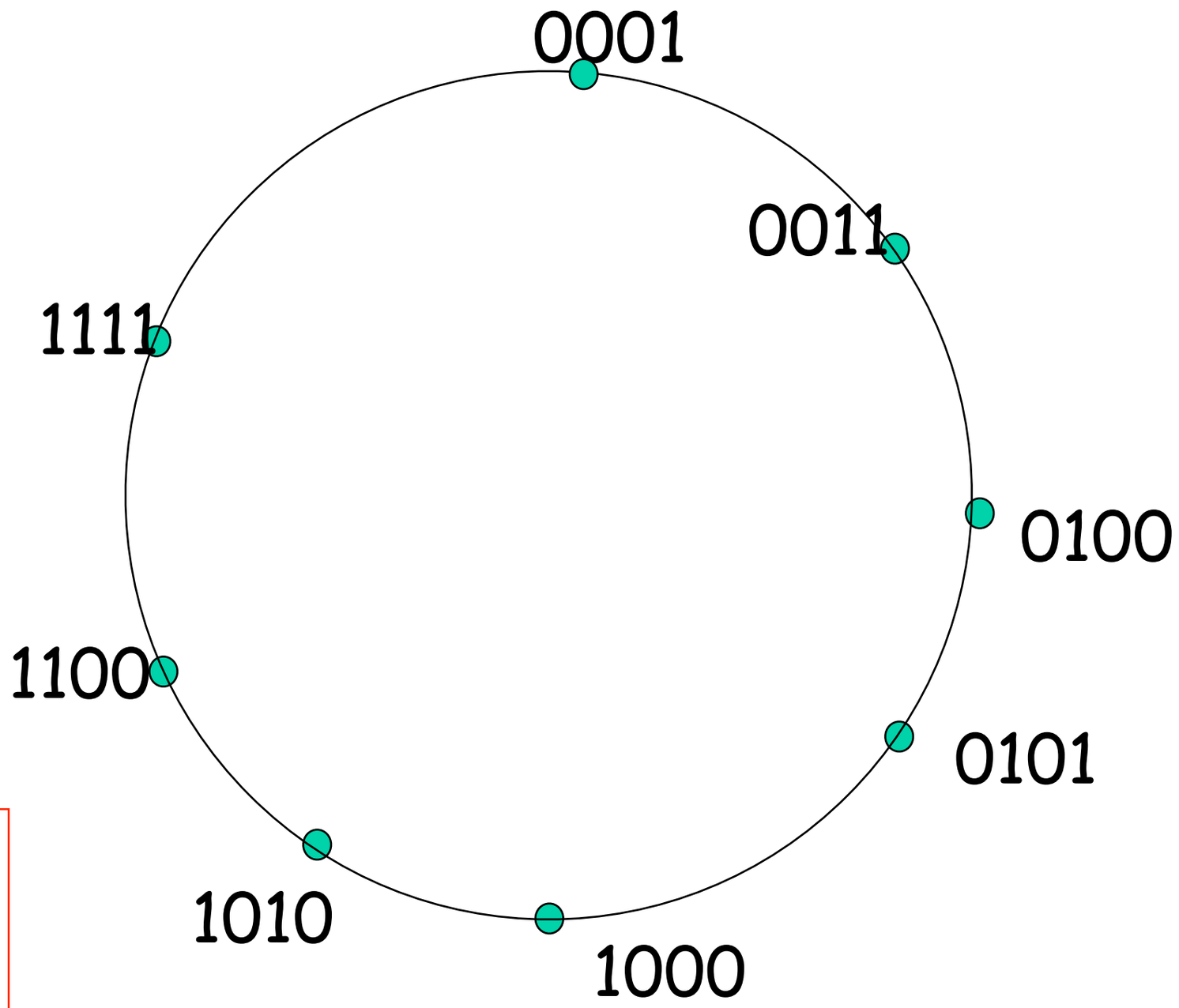
- central issue:
  - assigning (key, value) pairs to peers.
- rule: assign key to the peer that has the closest ID.
- convention in lecture: closest is the immediate successor of the key.
- e.g.,: n=4; peers: 1,3,4,5,8,10,12,14;
  - key = 13, then successor peer = 14
  - key = 15, then successor peer = 1

Circular DHT (1)



- each peer only aware of immediate successor and predecessor.
- "overlay network"

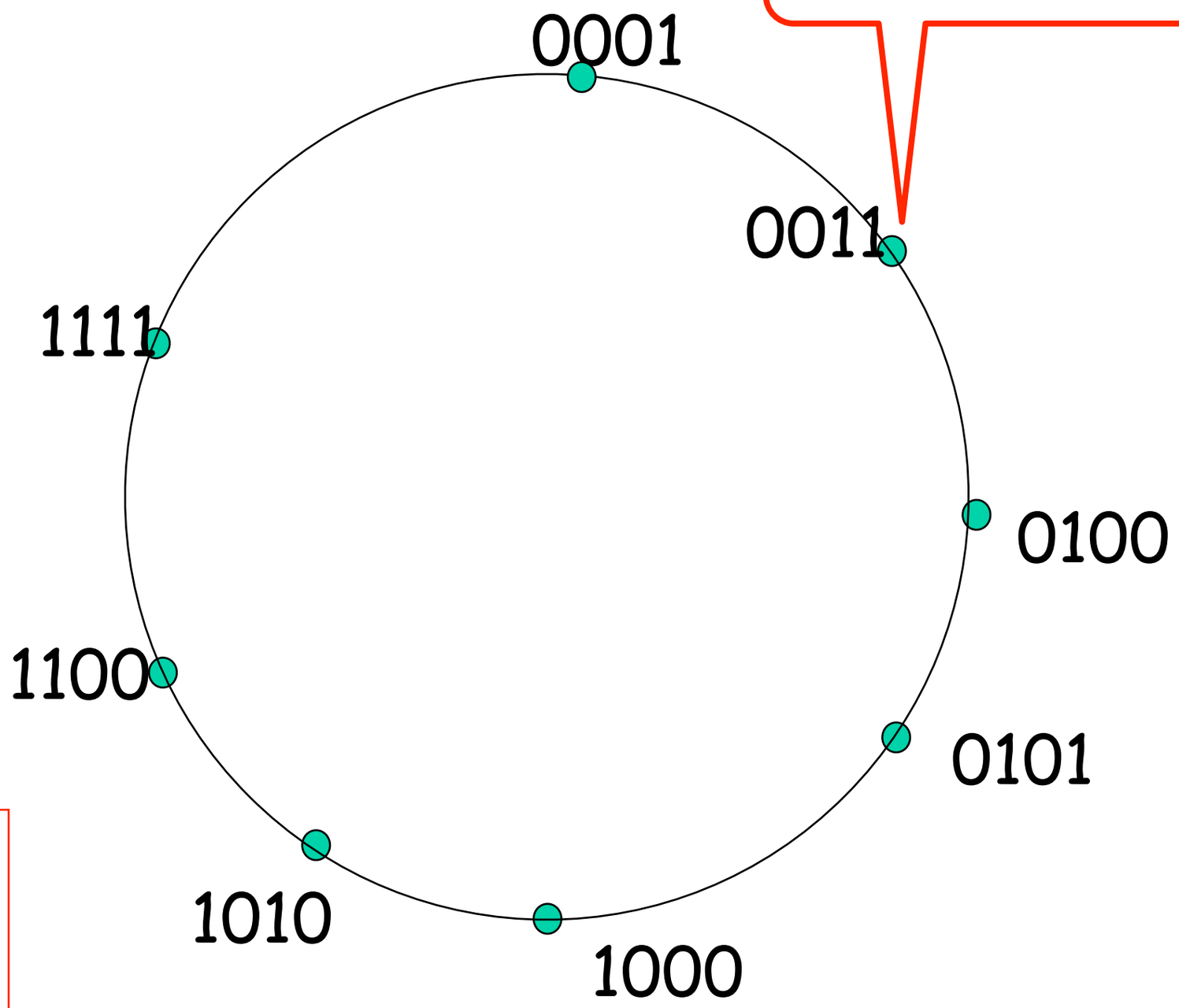
Circular DHT (2)



Define closest as closest successor

Circular DHT (2)

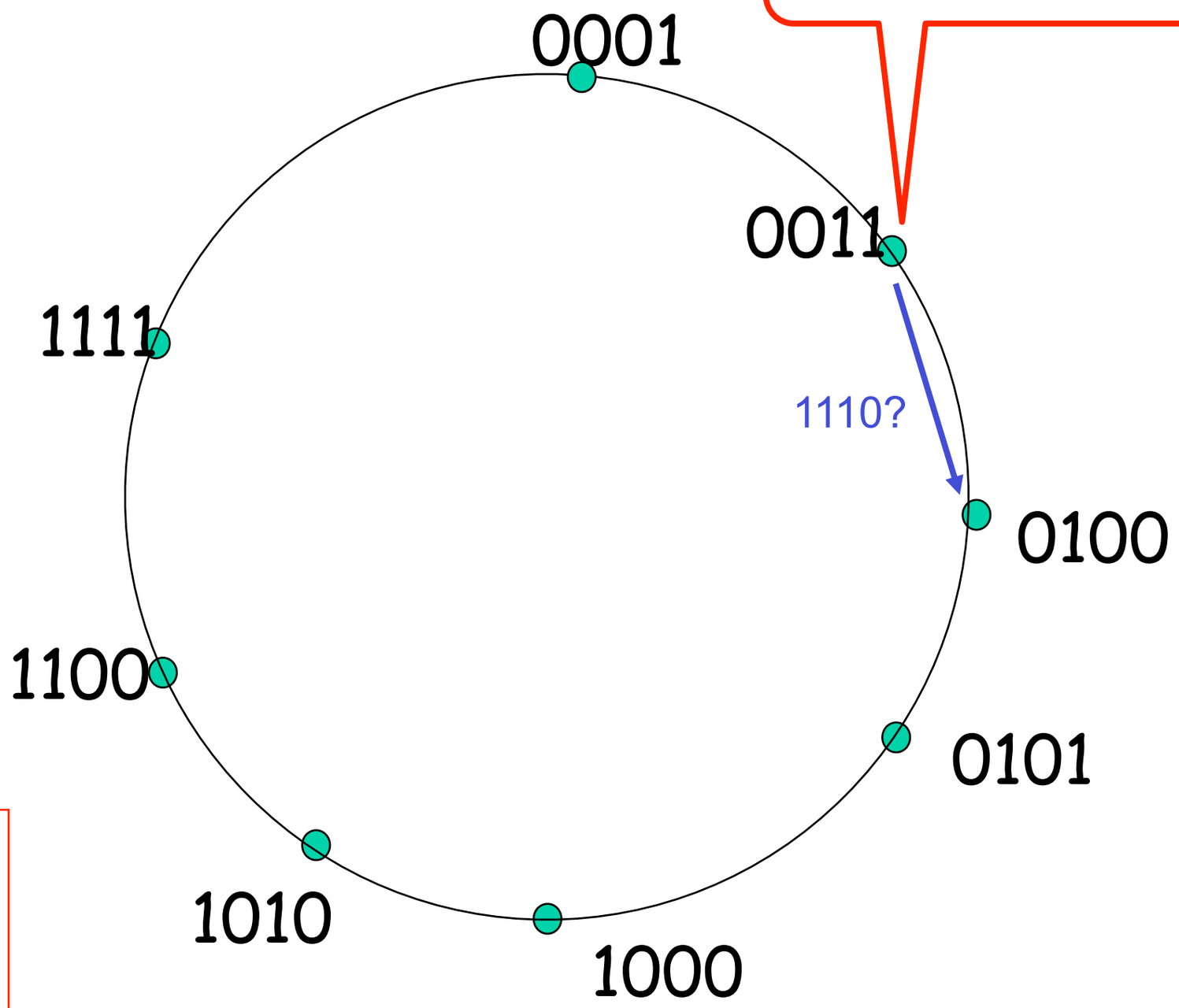
x = h("Led Zepplin IV")
x=1110
Who's resp. for 1110?



Define closest as closest successor

Circular DHT (2)

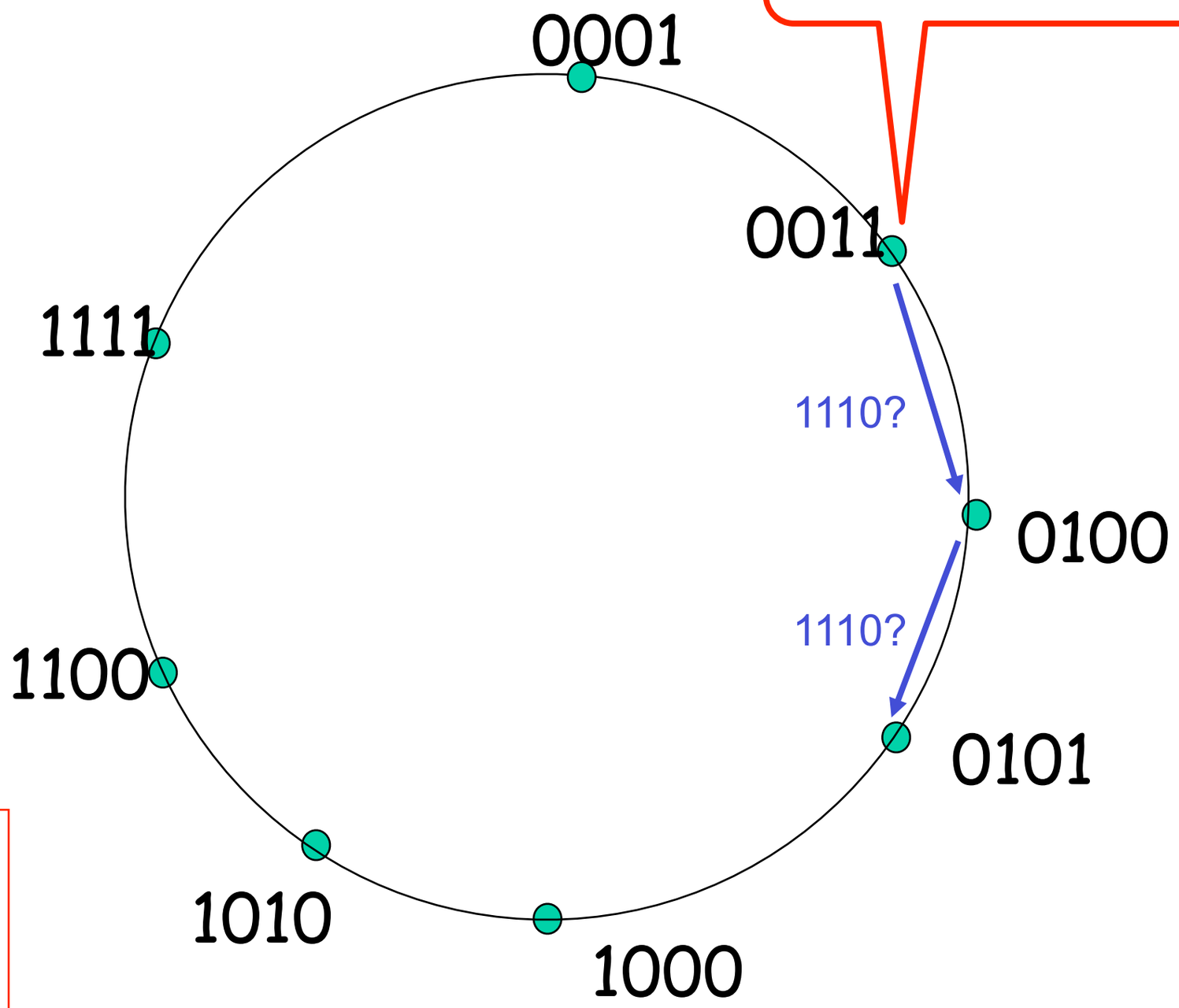
x = h("Led Zepplin IV")
x=1110
Who's resp. for 1110?



Define closest as closest successor

Circular DHT (2)

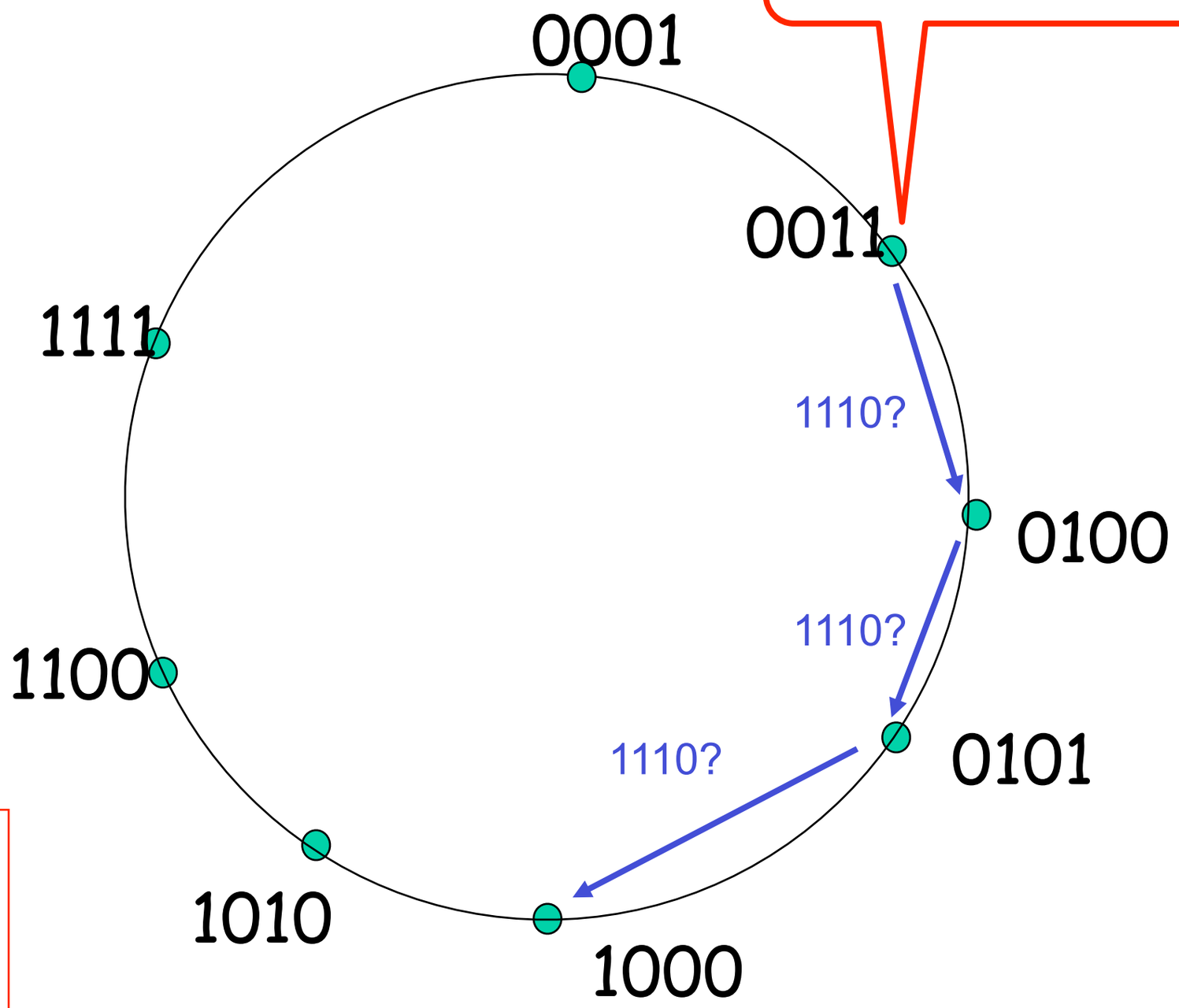
x = h("Led Zepplin IV")
x=1110
Who's resp. for 1110?



Define closest as closest successor

Circular DHT (2)

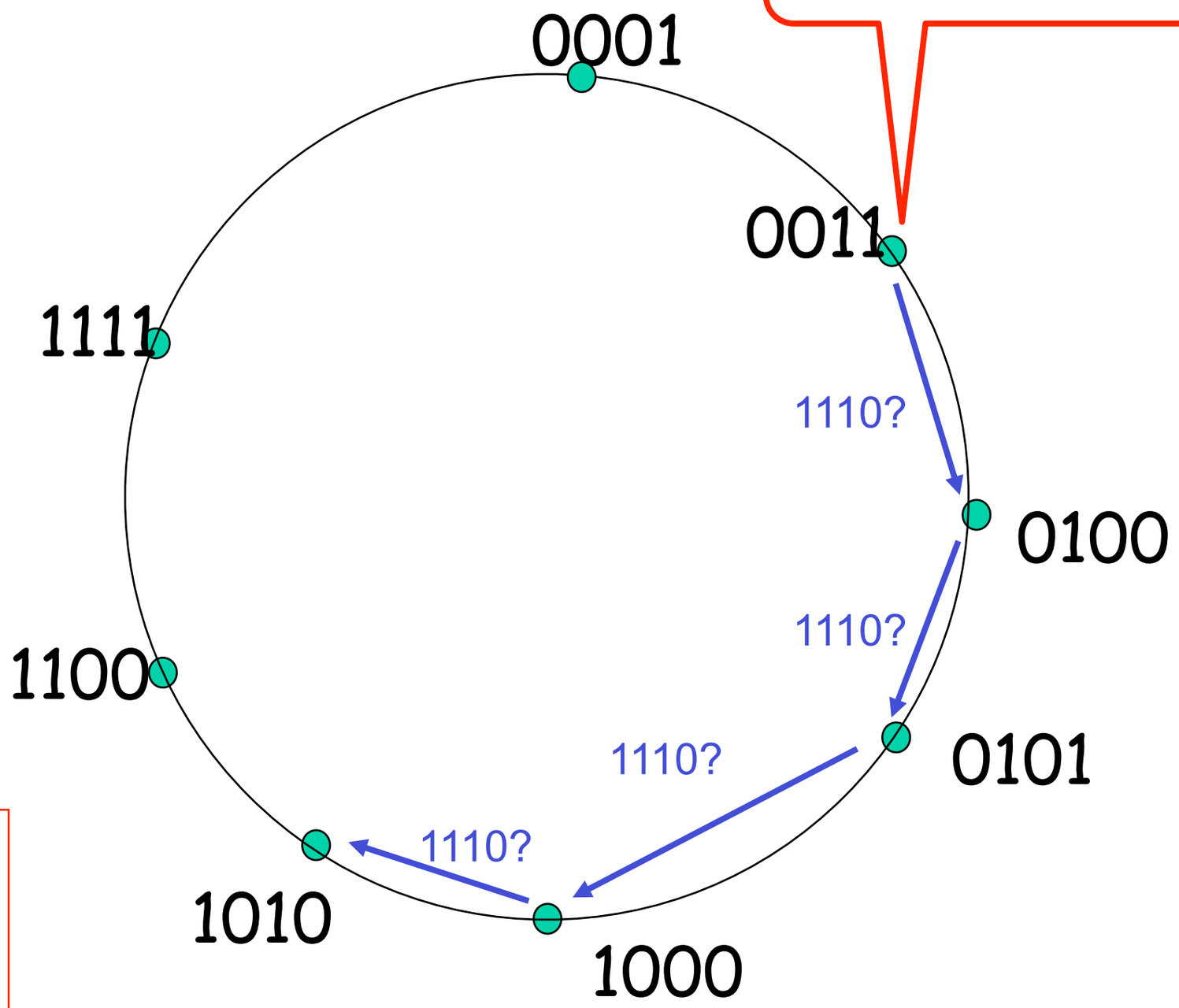
x = h("Led Zepplin IV")
x=1110
Who's resp. for 1110?



Define closest as closest successor

Circular DHT (2)

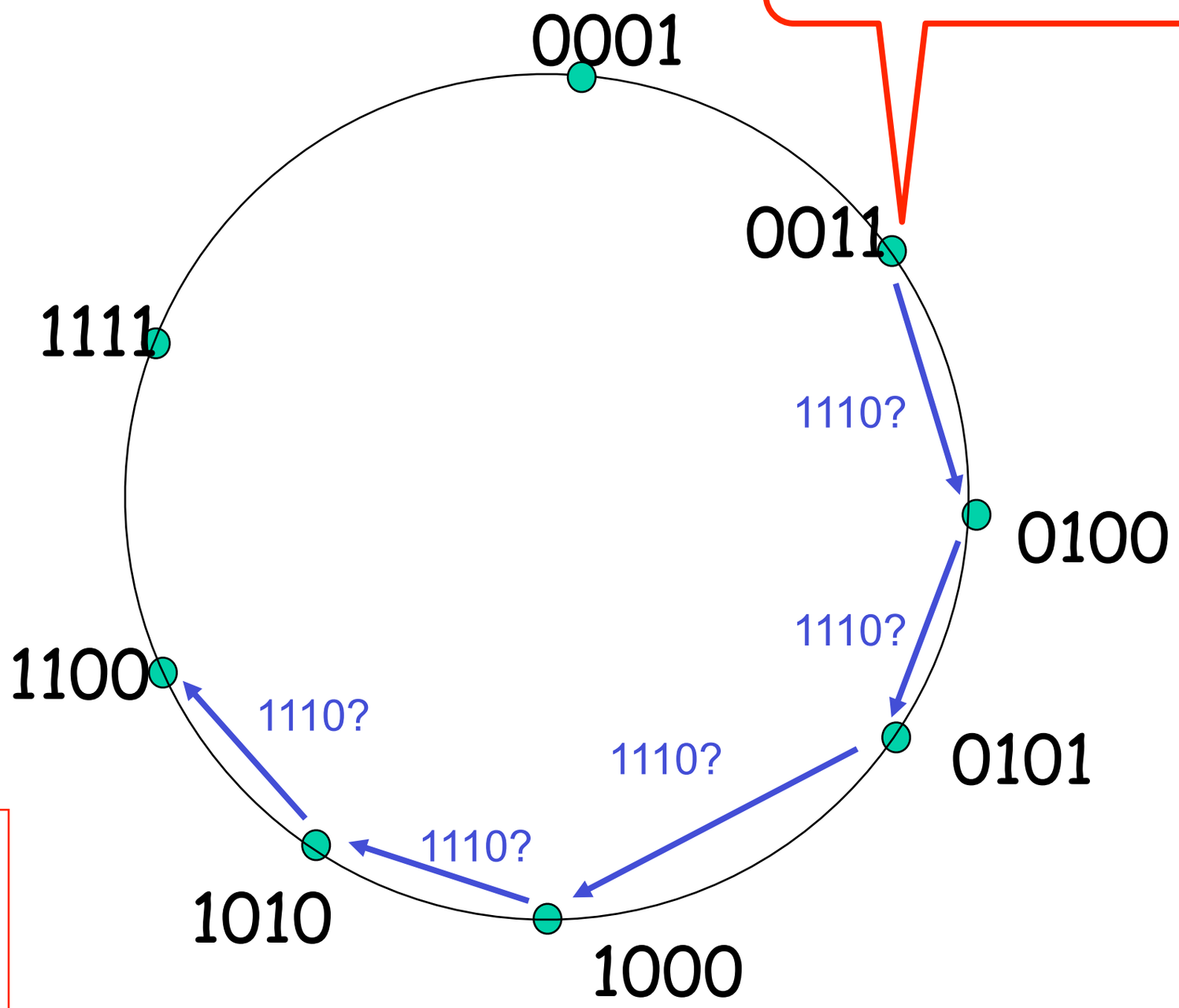
x = h("Led Zepplin IV")
x=1110
Who's resp. for 1110?



Define closest as closest successor

Circular DHT (2)

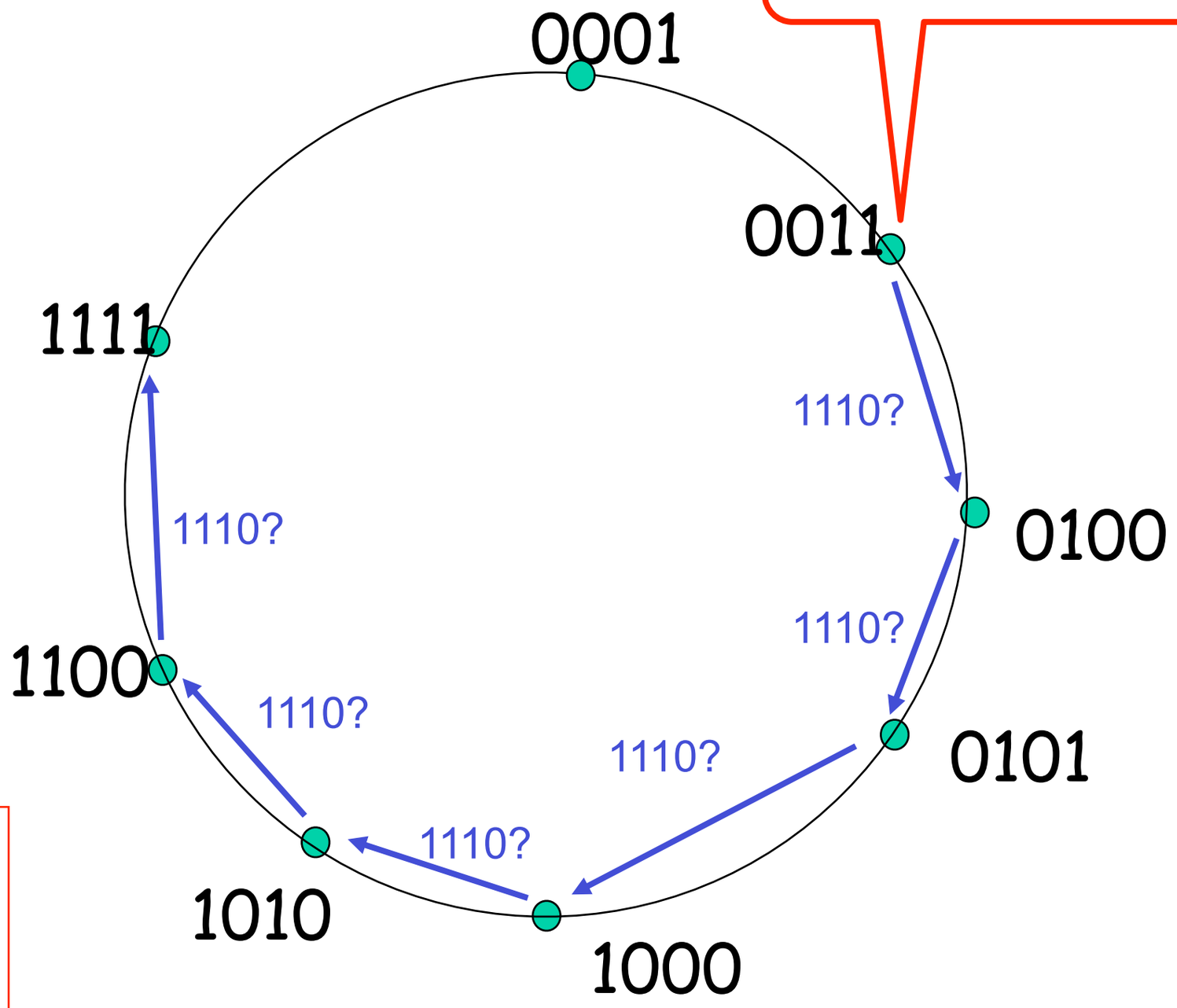
x = h("Led Zepplin IV")
x=1110
Who's resp. for 1110?



Define closest as closest successor

Circular DHT (2)

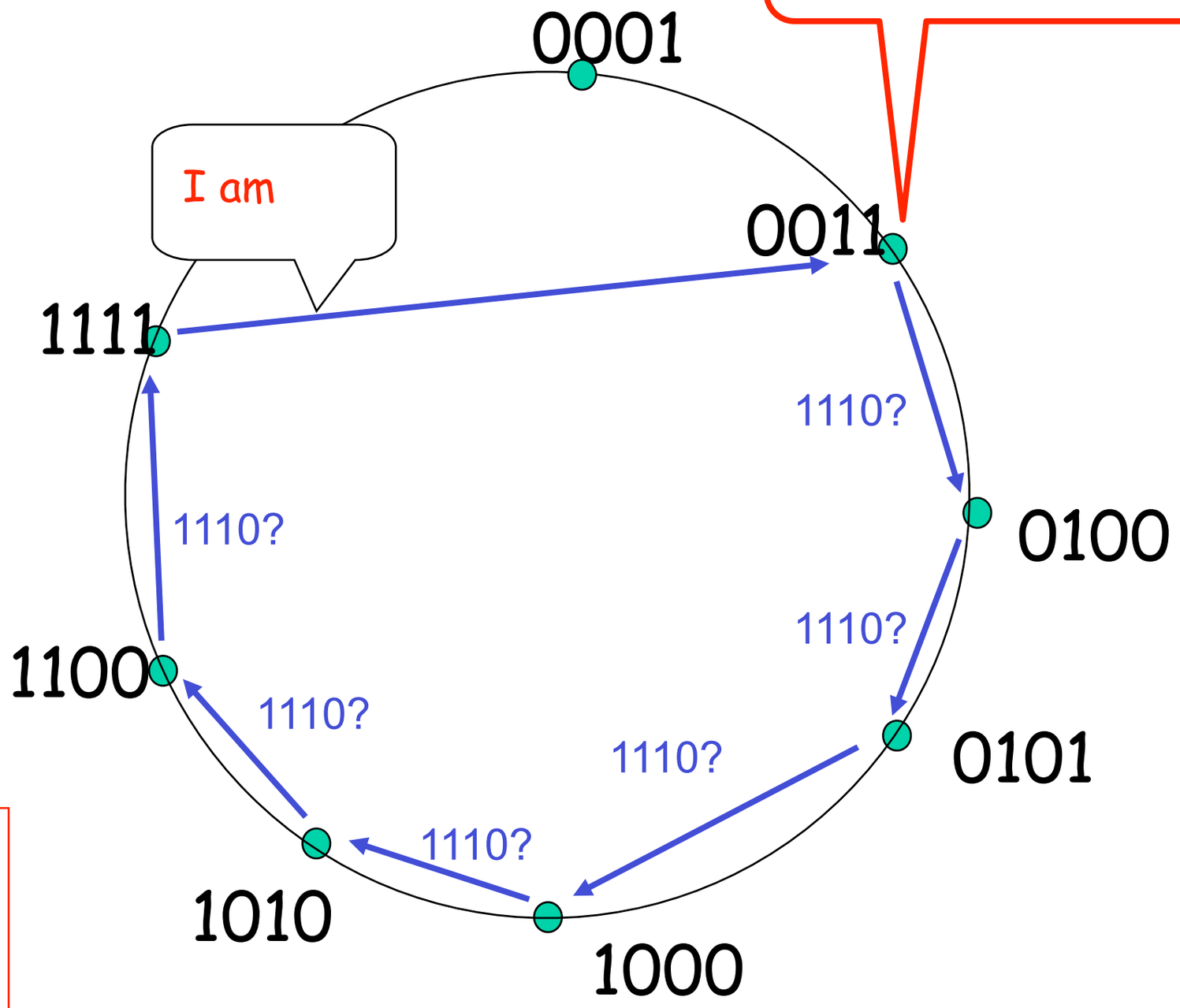
x = h("Led Zepplin IV")
x=1110
Who's resp. for 1110?



Define closest as closest successor

Circular DHT (2)

x = h("Led Zepplin IV")
x=1110
Who's resp. for 1110?

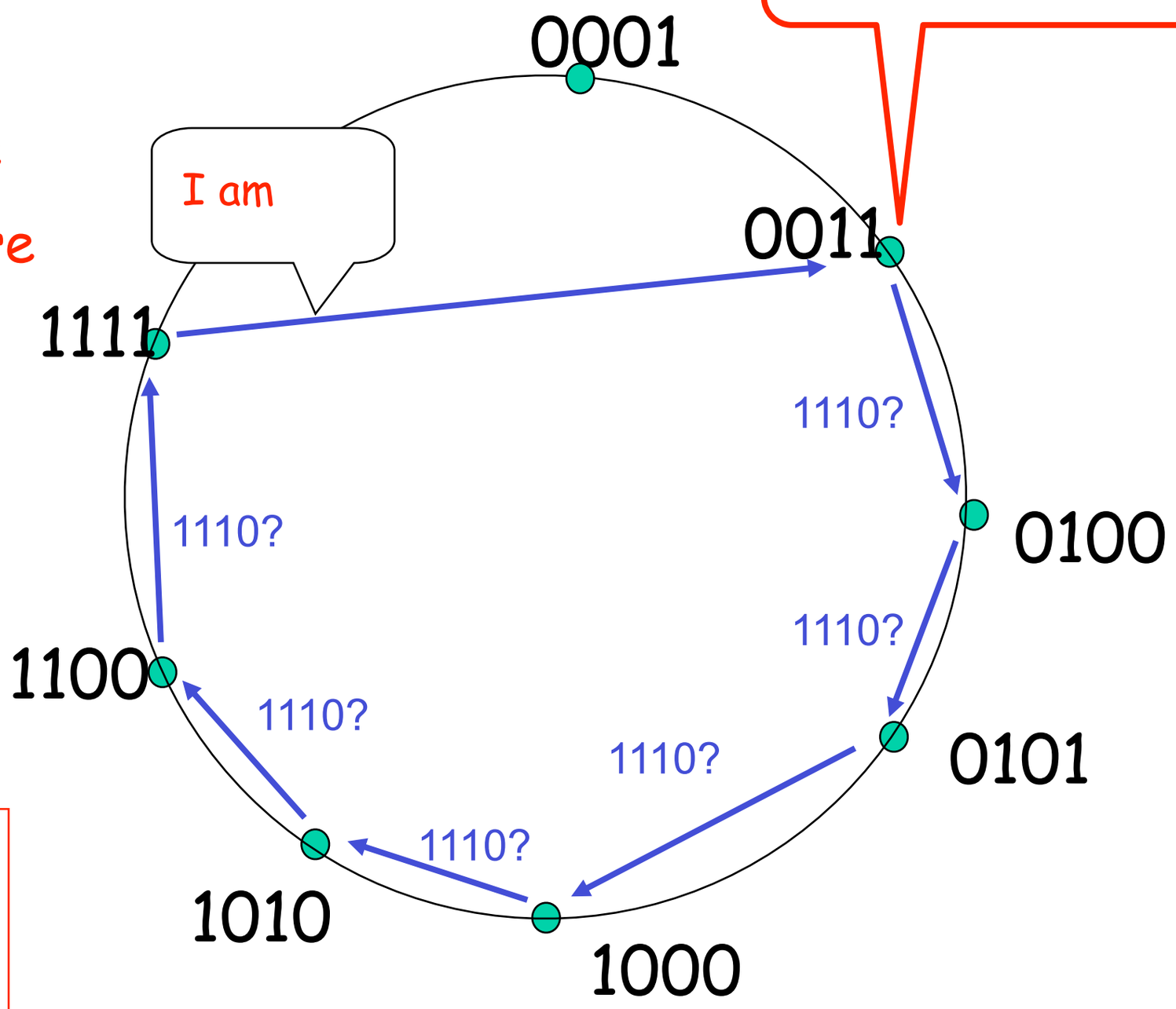


Define closest as closest successor

Circular DHT (2)

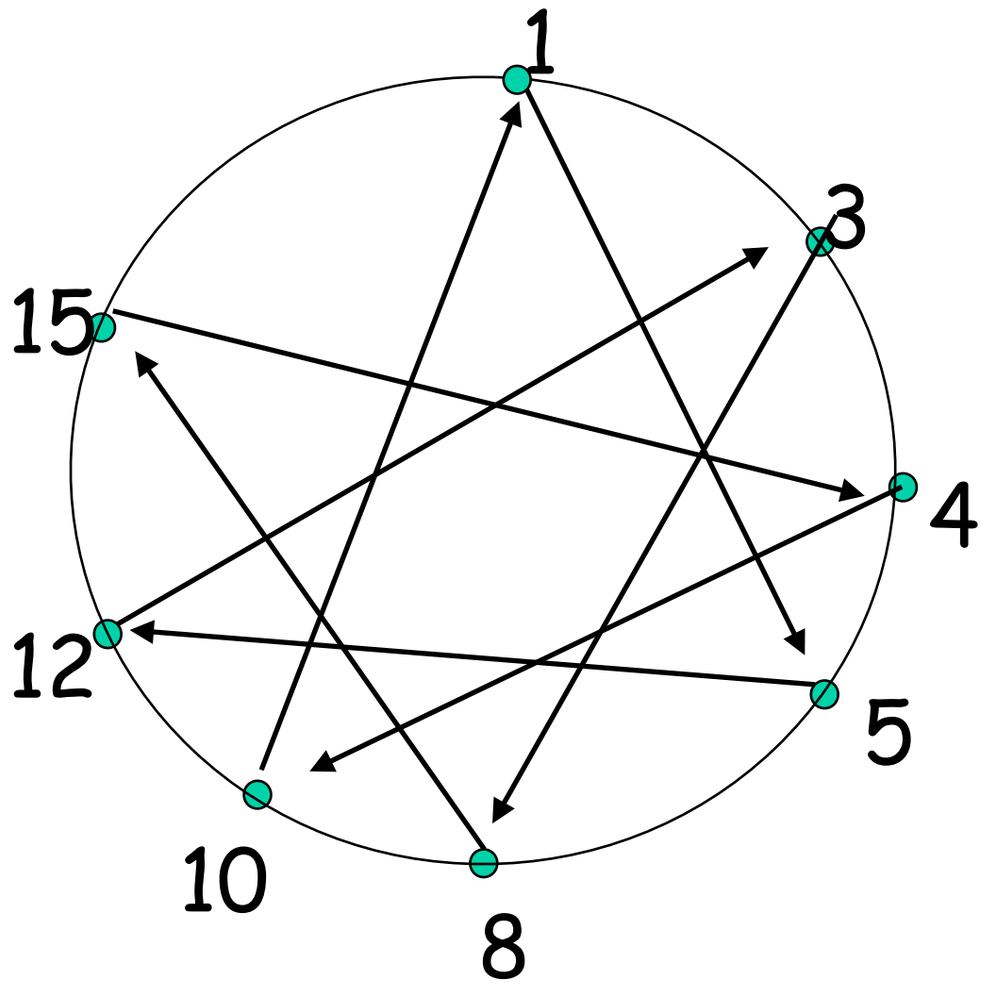
x = h("Led Zepplin IV")
x=1110
Who's resp. for 1110?

O(N) messages on avg to resolve query, when there are N peers

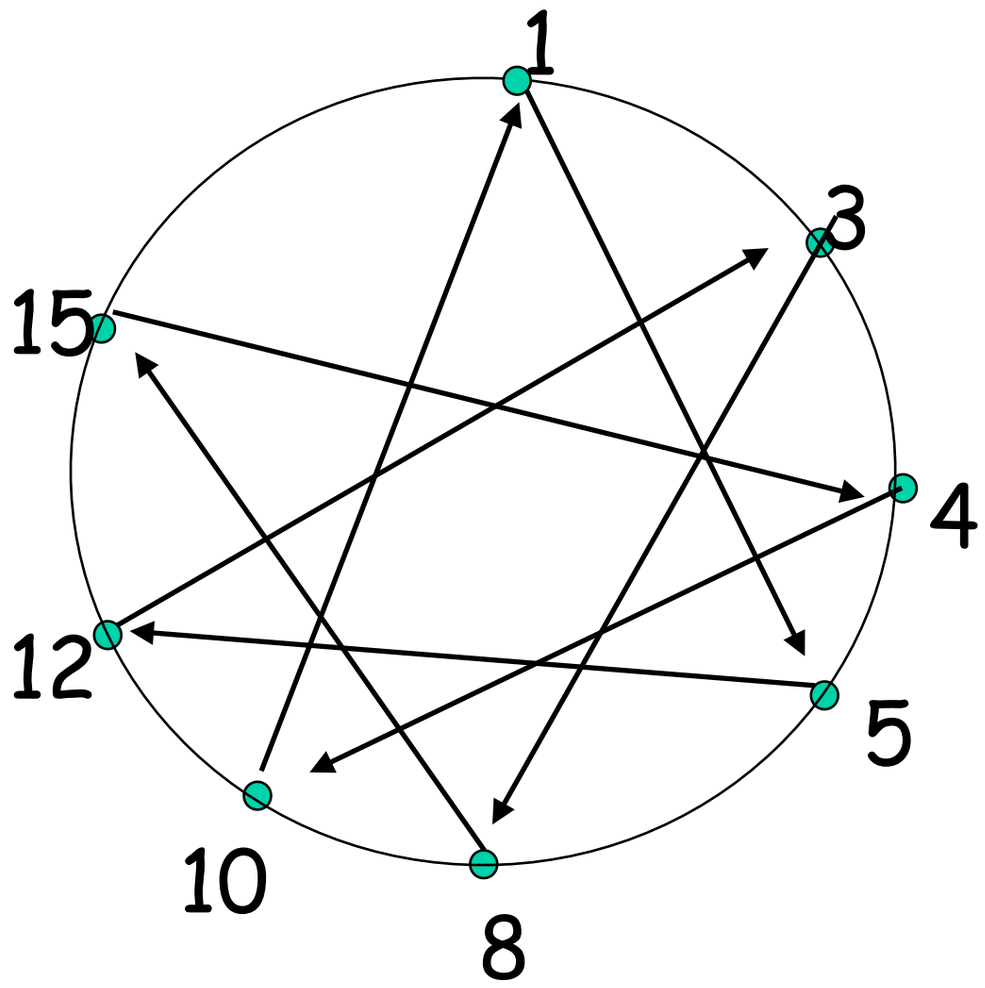


Define closest as closest successor

Circular DHT with Shortcuts

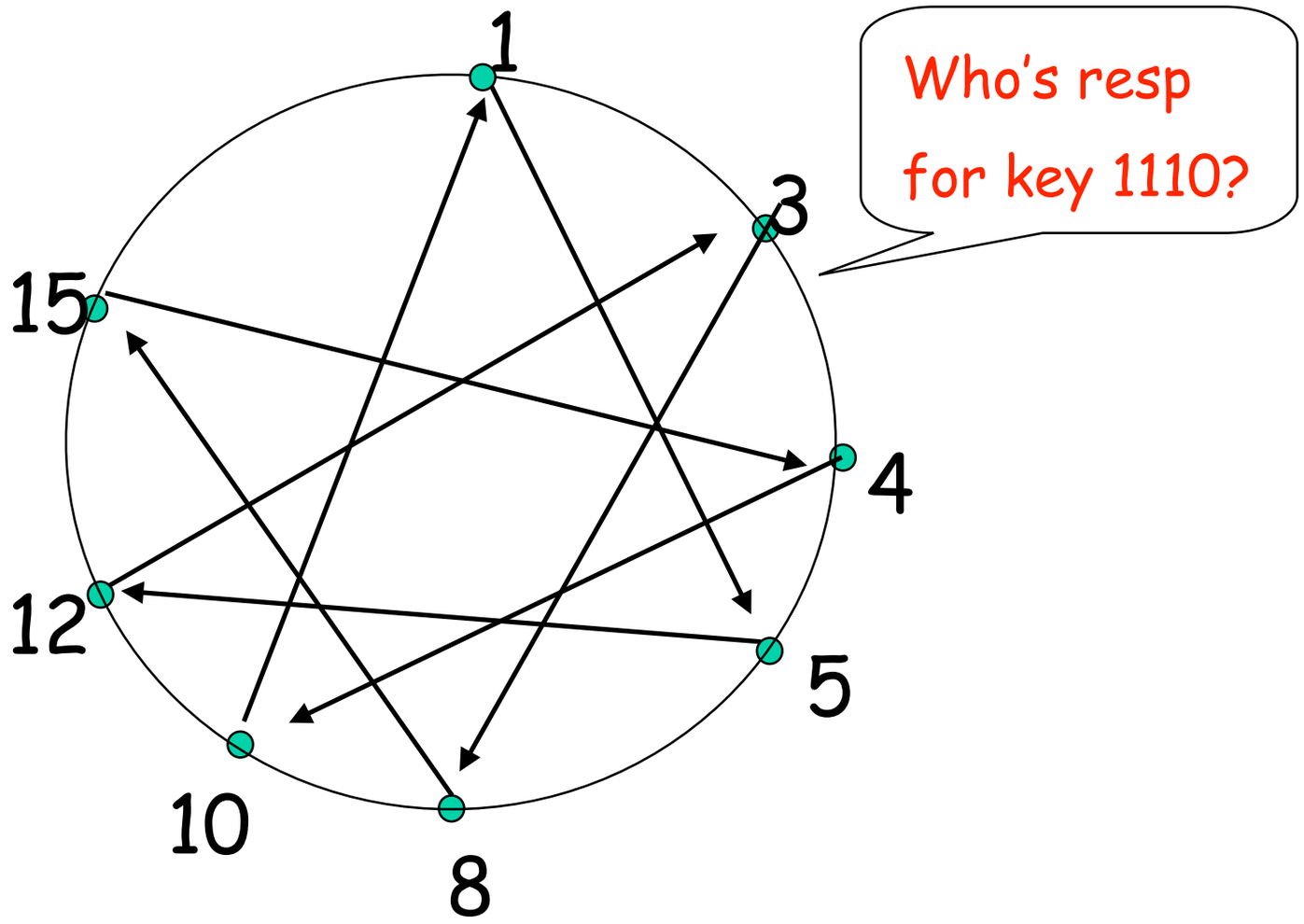


Circular DHT with Shortcuts



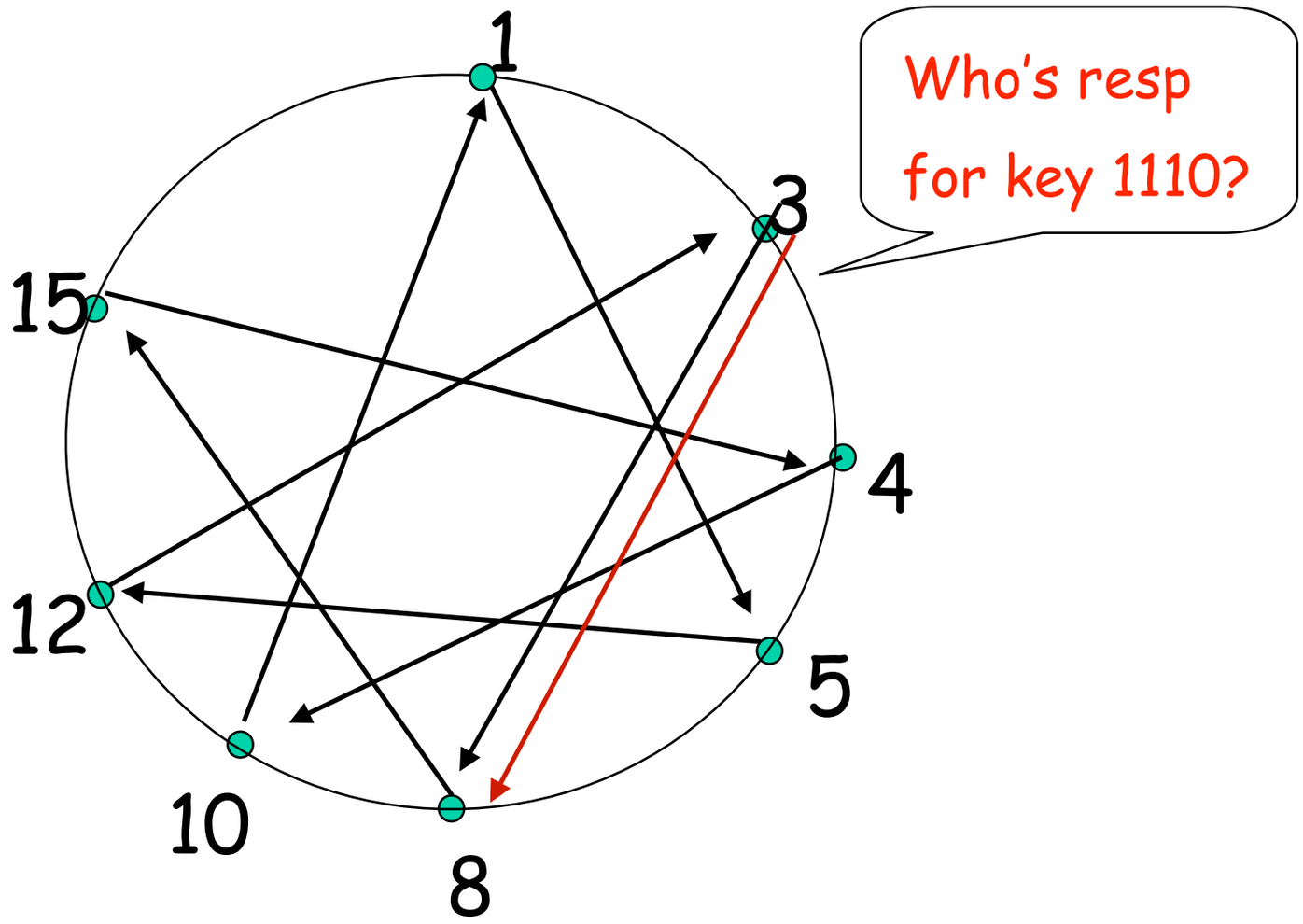
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

Circular DHT with Shortcuts



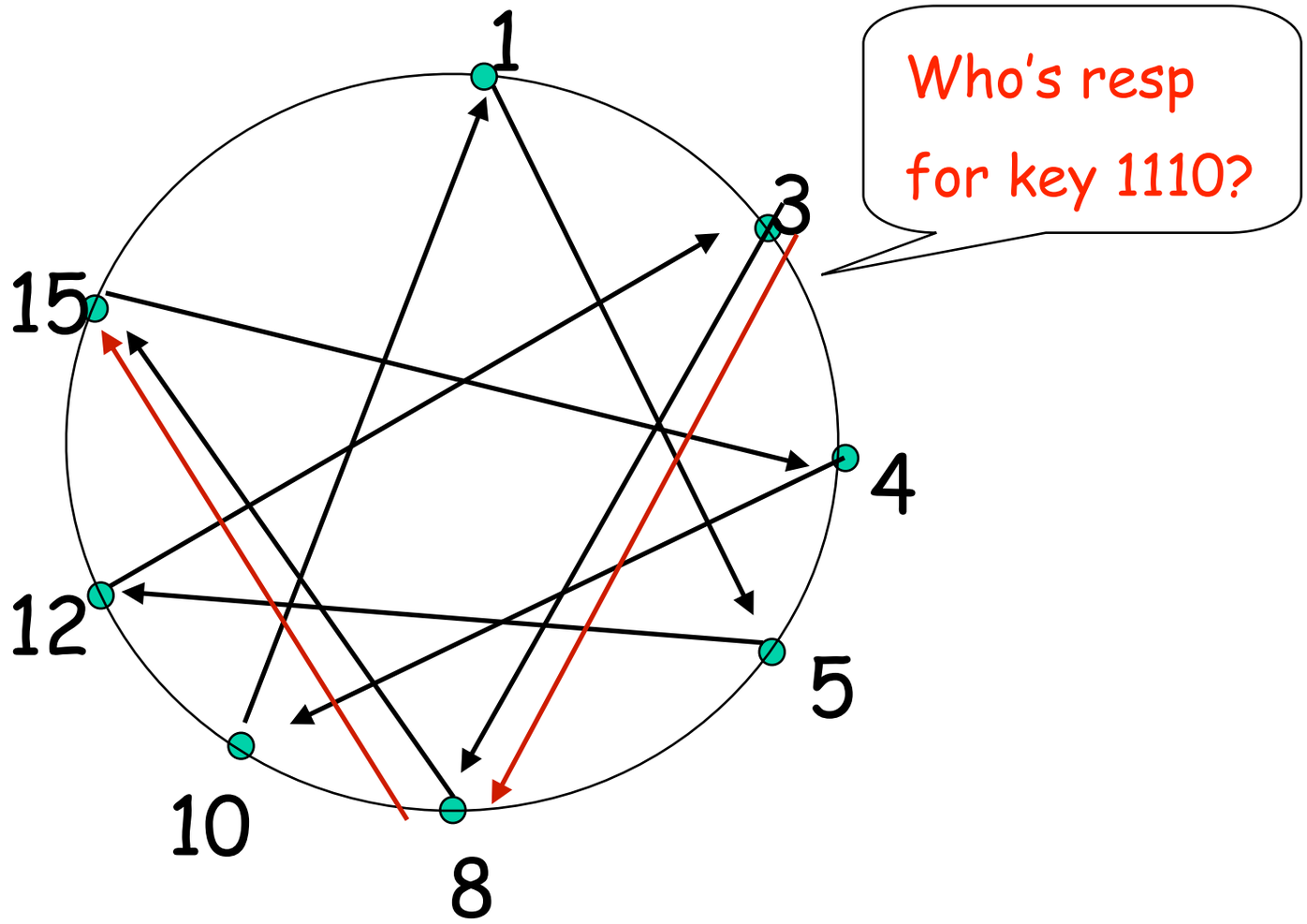
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

Circular DHT with Shortcuts



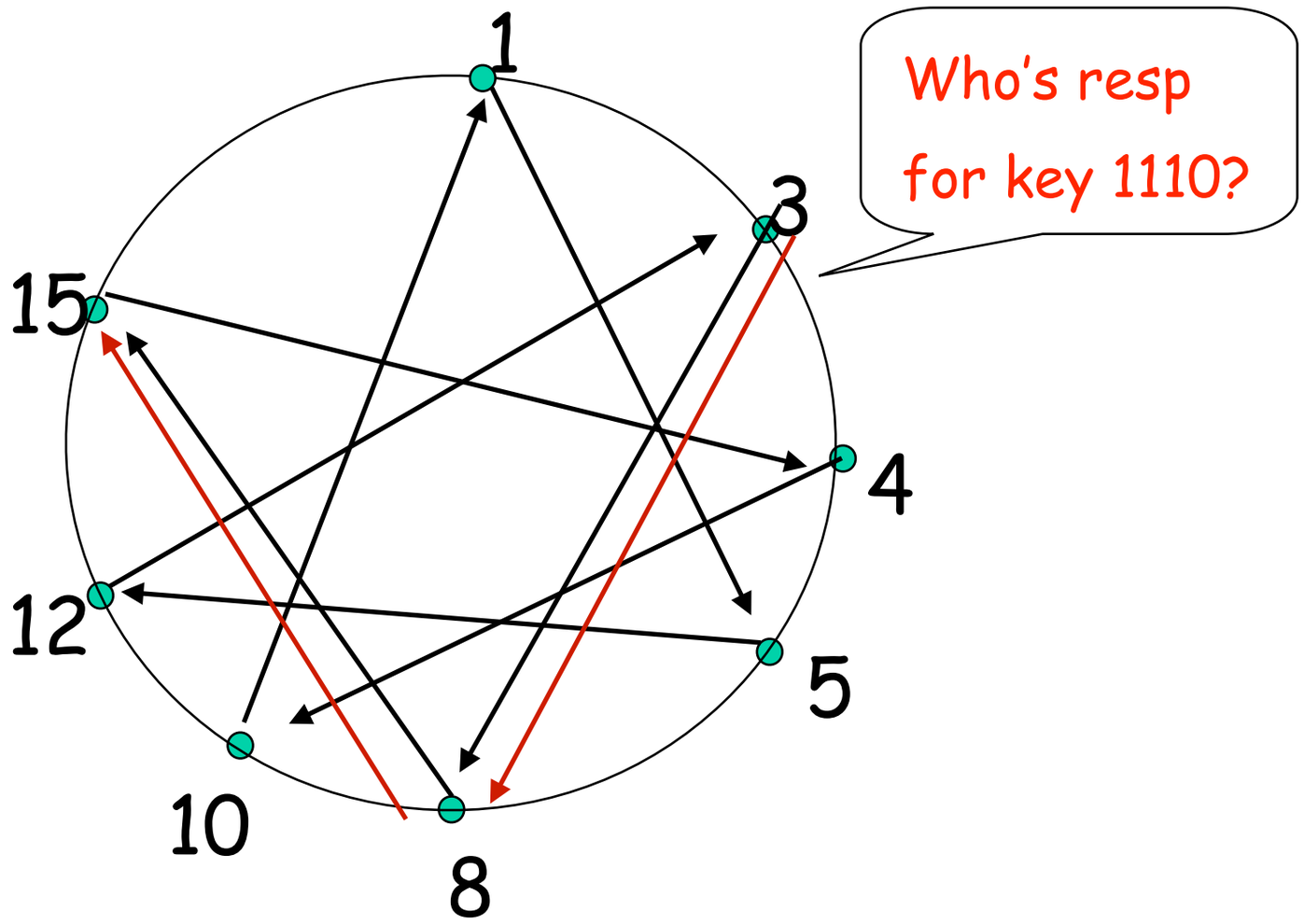
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

Circular DHT with Shortcuts



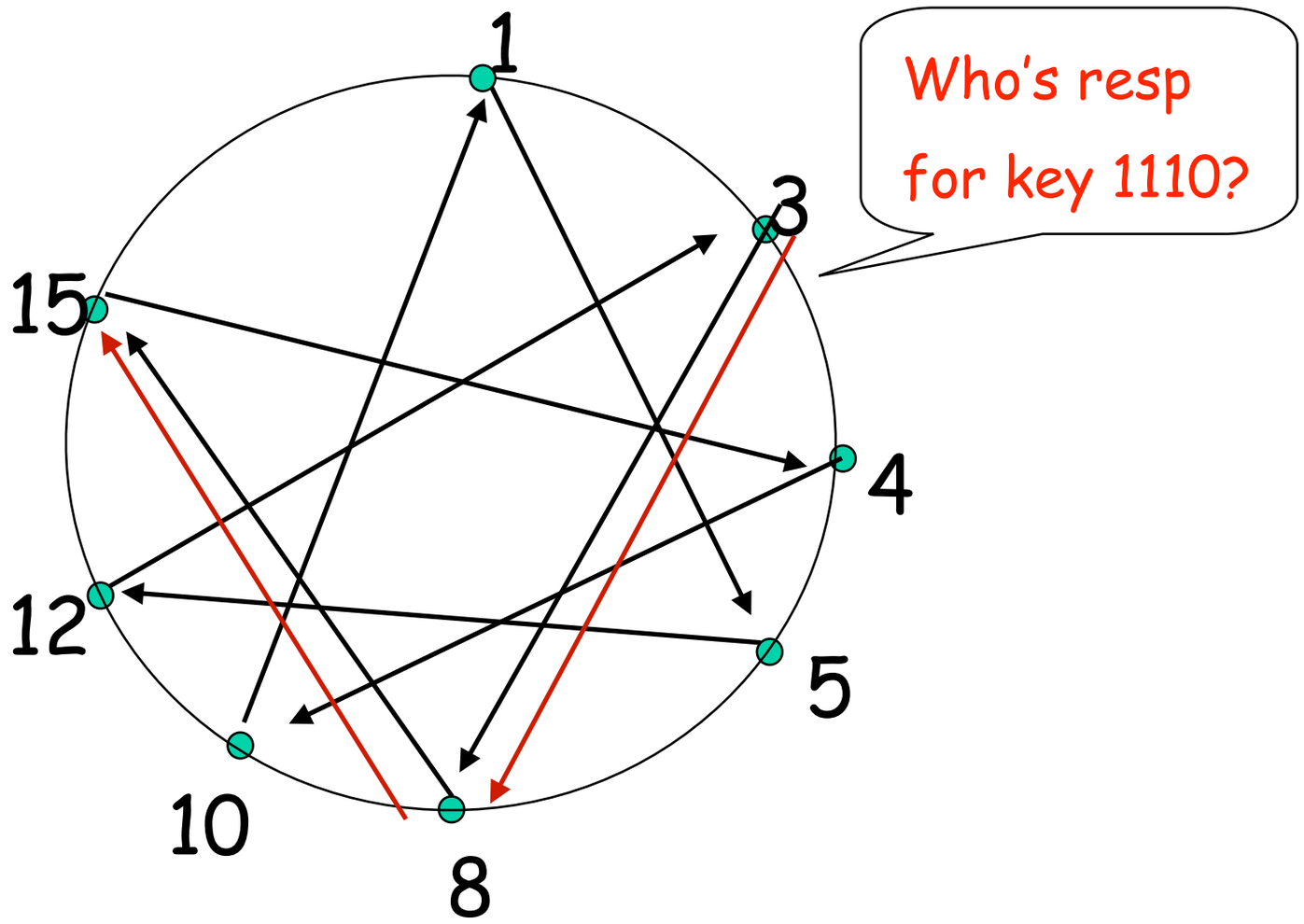
- each peer keeps track of IP addresses of predecessor, successor, short cuts.

Circular DHT with Shortcuts



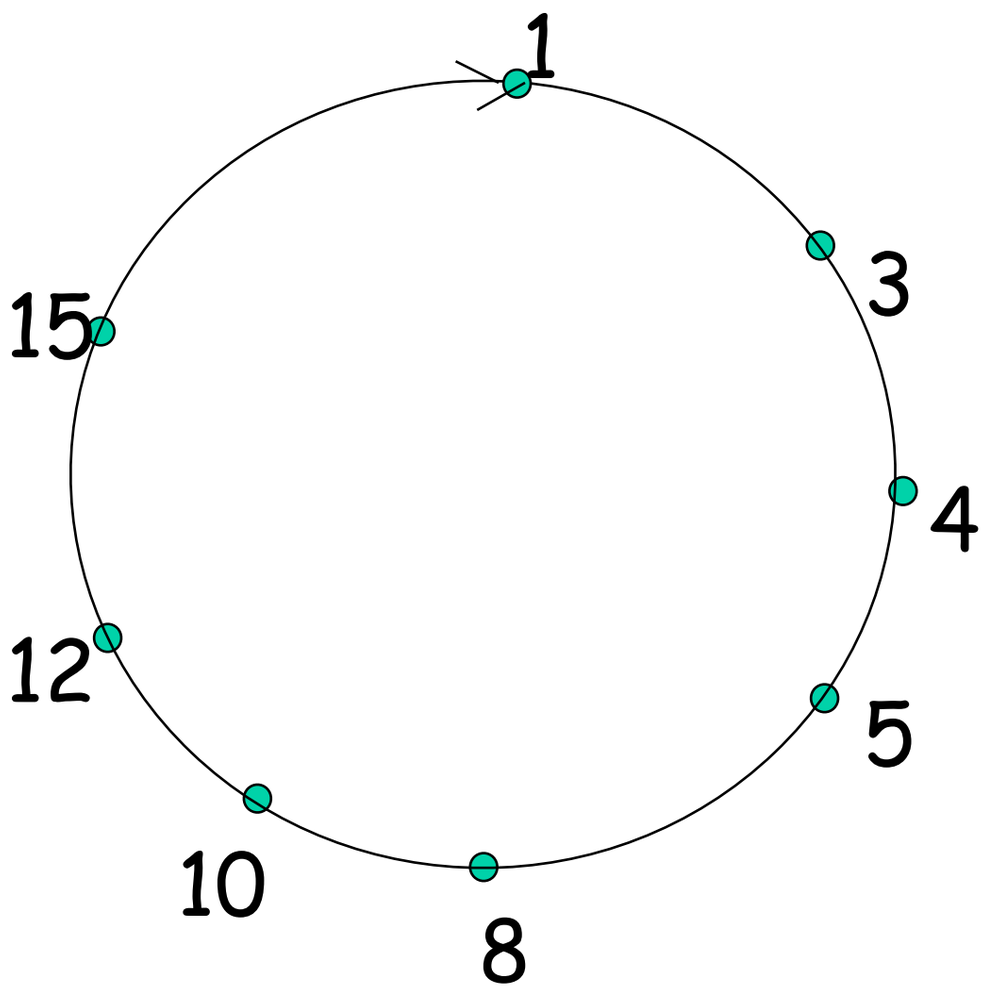
- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 2 messages.

Circular DHT with Shortcuts



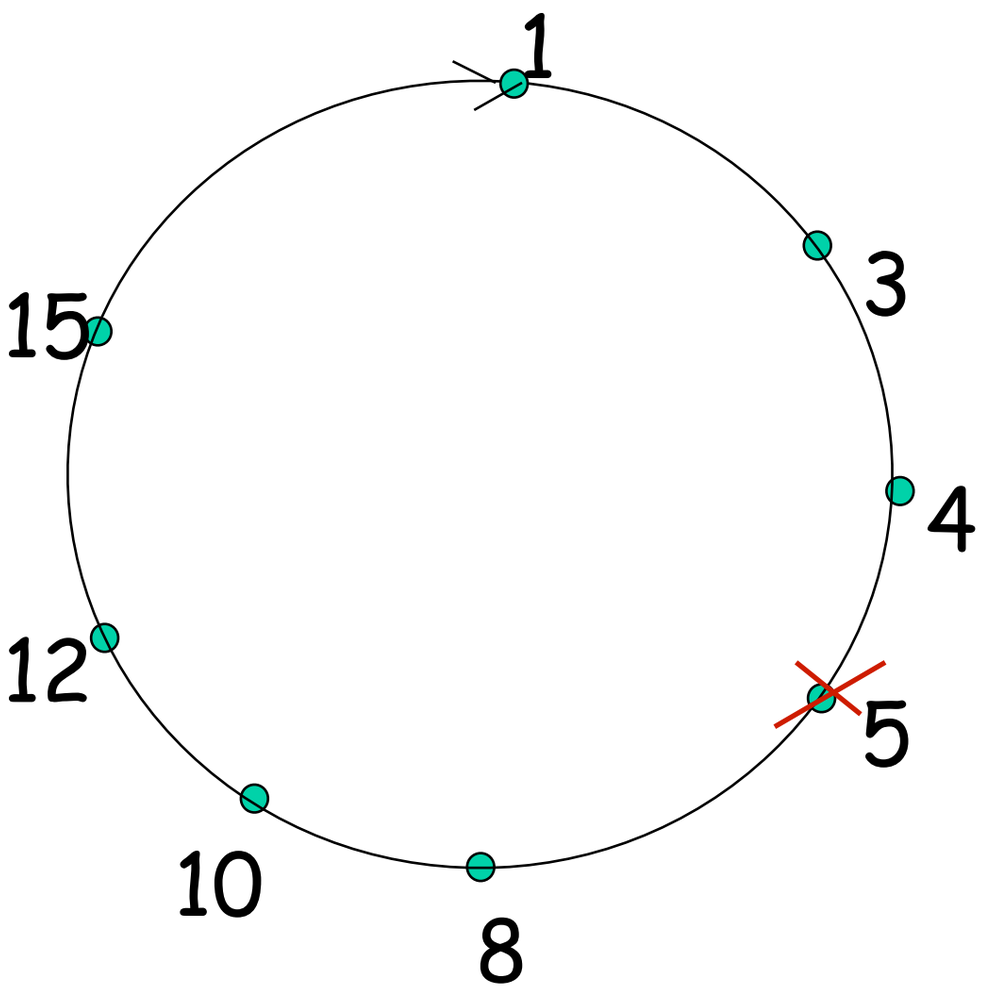
- each peer keeps track of IP addresses of predecessor, successor, short cuts.
- reduced from 6 to 2 messages.
- possible to design shortcuts so O(log N) neighbors, O(log N) messages in query

Peer Churn



- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.

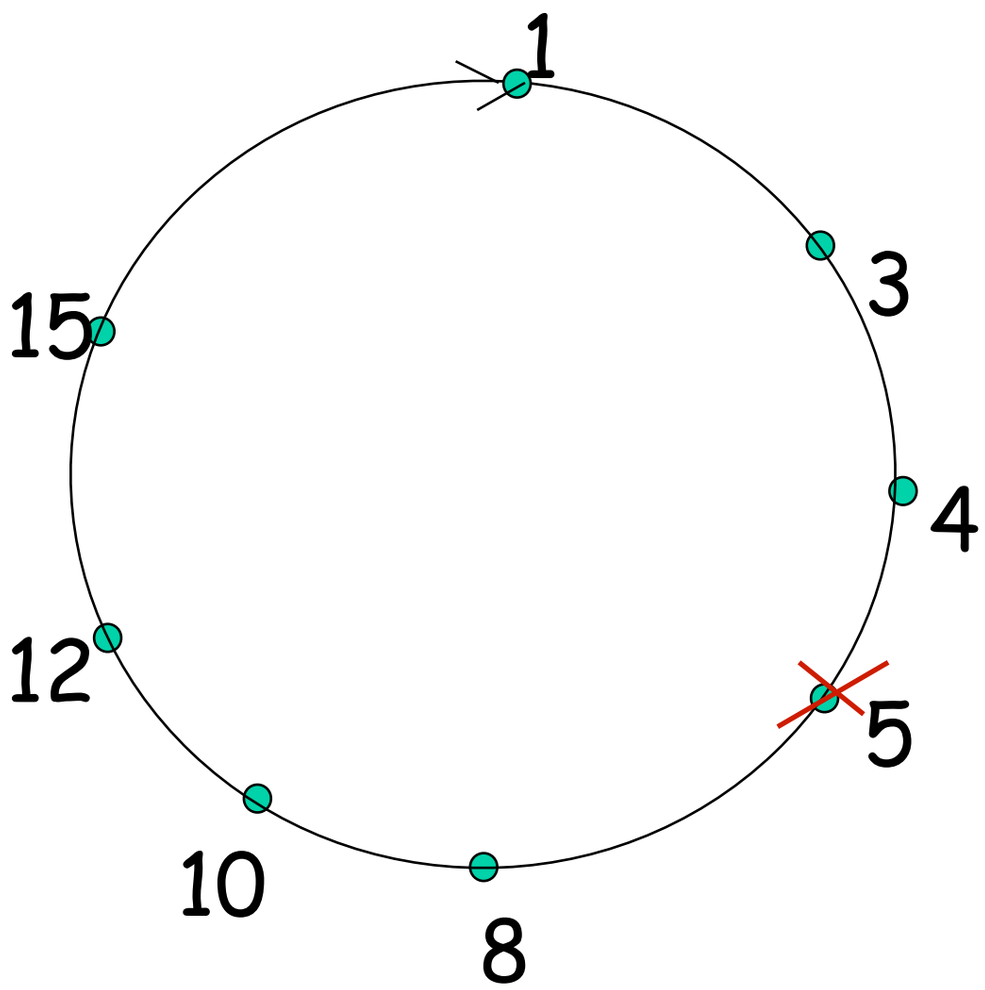
Peer Churn



- peer 5 abruptly leaves

- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.

Peer Churn



- To handle peer churn, require each peer to know the IP address of its two successors.
- Each peer periodically pings its two successors to see if they are still alive.

- peer 5 abruptly leaves
- Peer 4 detects; makes 8 its immediate successor; asks 8 who its immediate successor is; makes 8's immediate successor its second successor.

Distributed Data Structures

Distributed Data Structures

- DHTs are broadly applicable to many applications

Distributed Data Structures

- DHTs are broadly applicable to many applications
- What is another distributed data structure?

Distributed Data Structures

- DHTs are broadly applicable to many applications

- What is another distributed data structure?
  - blockchain
  - essentially a cryptographically secure ledger that can be broadly shared with integrity guarantees

P2P Case study: Skype

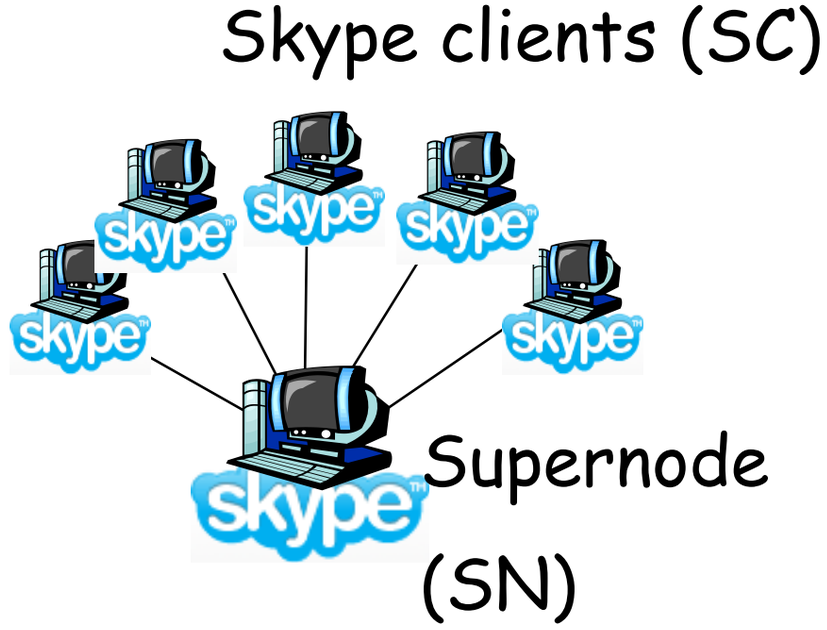
- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SuperNodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs

P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SuperNodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs

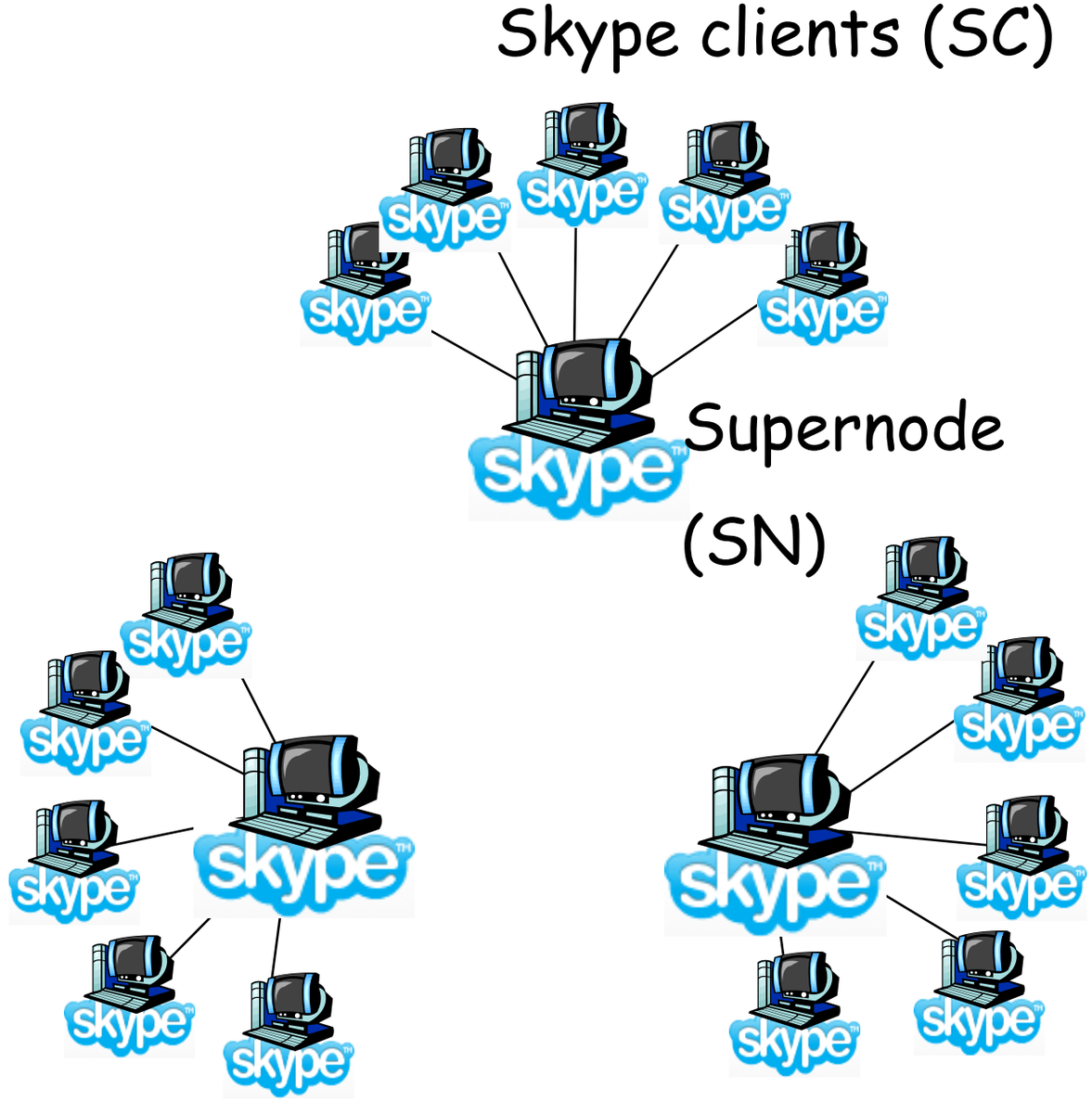
P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SuperNodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs



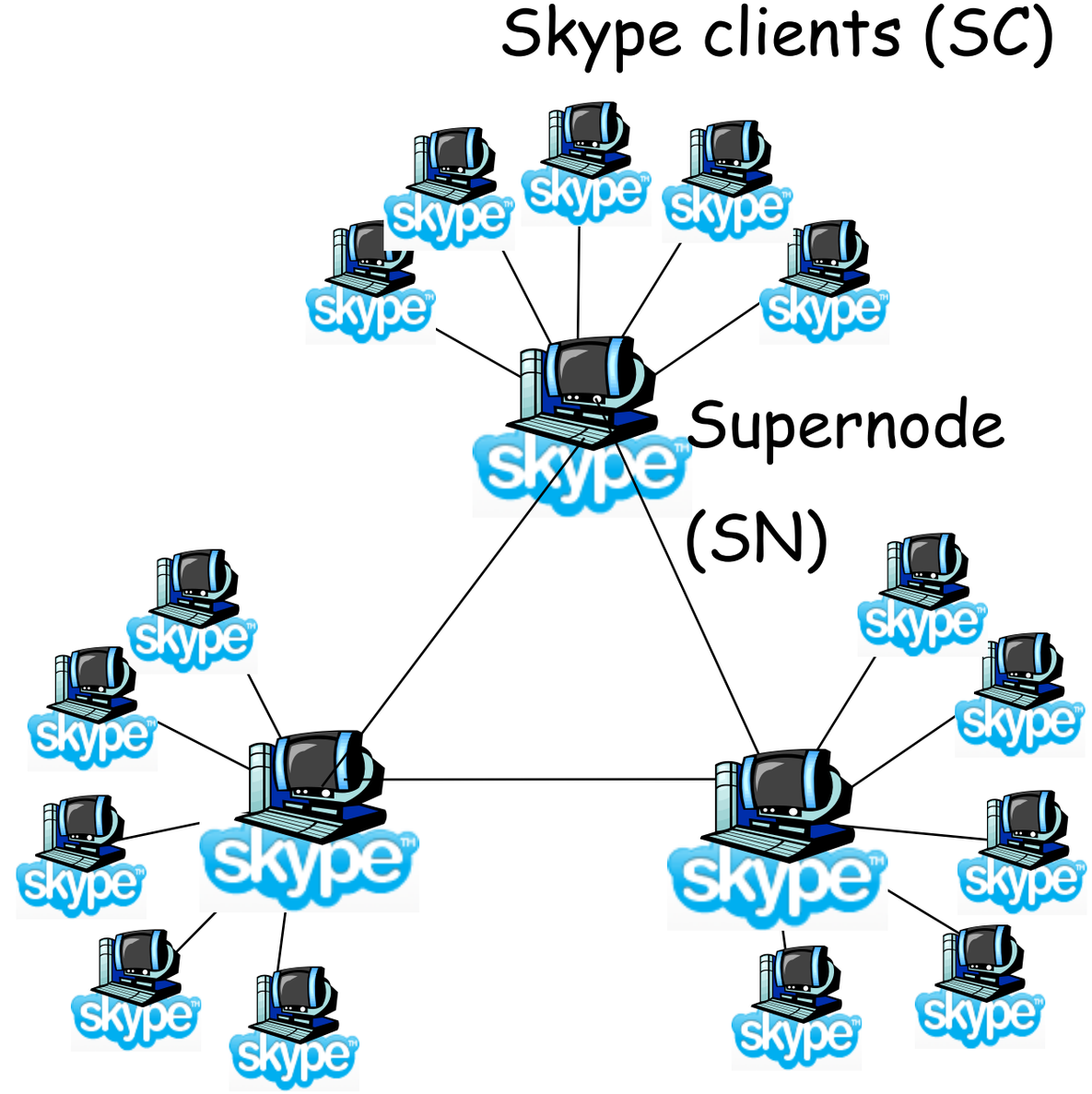
P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SuperNodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs



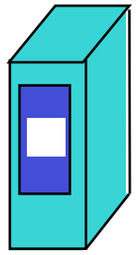
P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SuperNodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs



P2P Case study: Skype

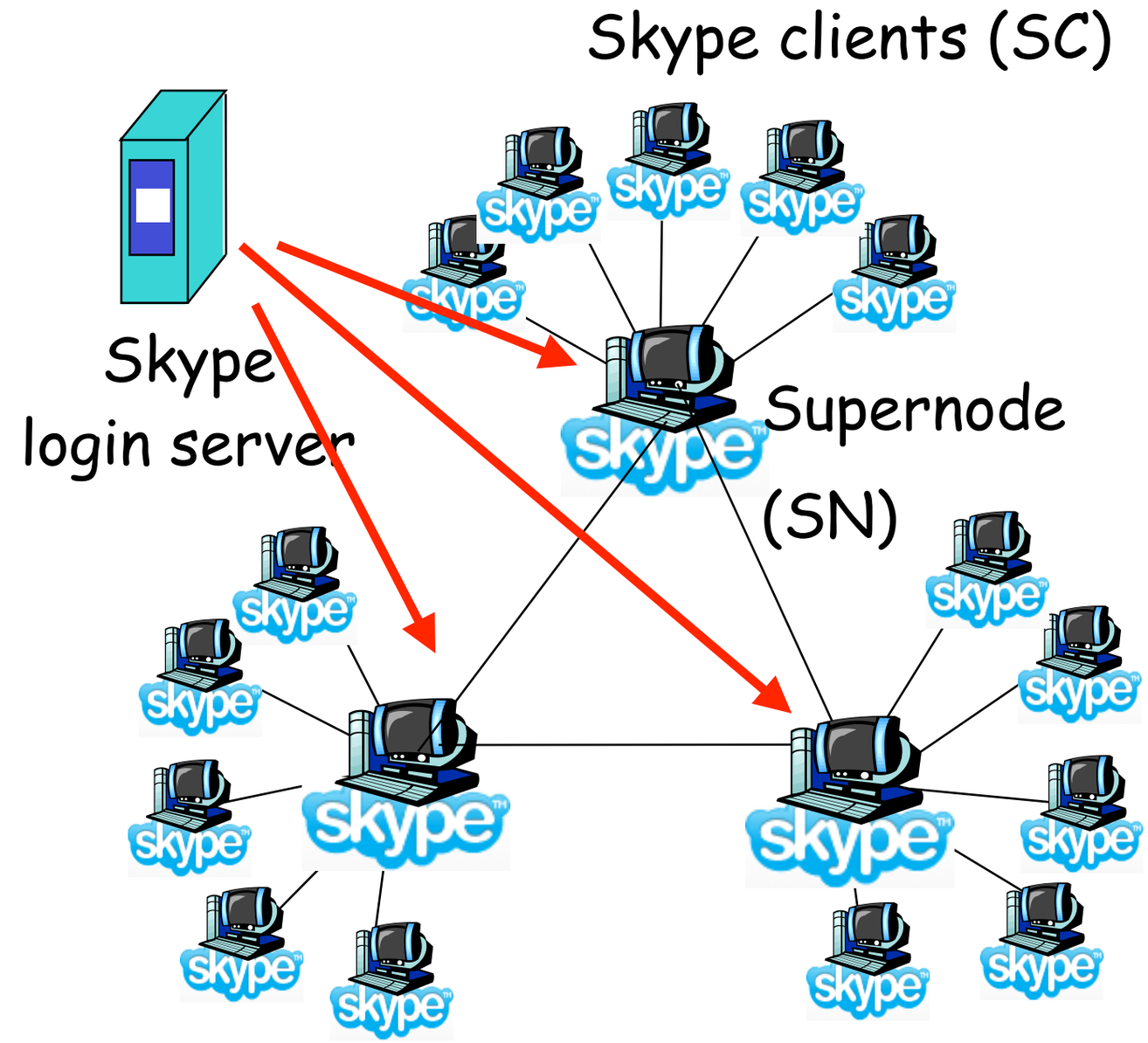
- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SuperNodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs



Skype login server

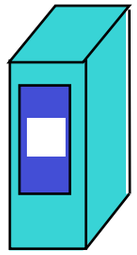
P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SuperNodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs



P2P Case study: Skype

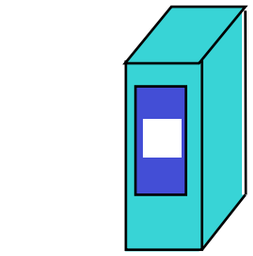
- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SuperNodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs



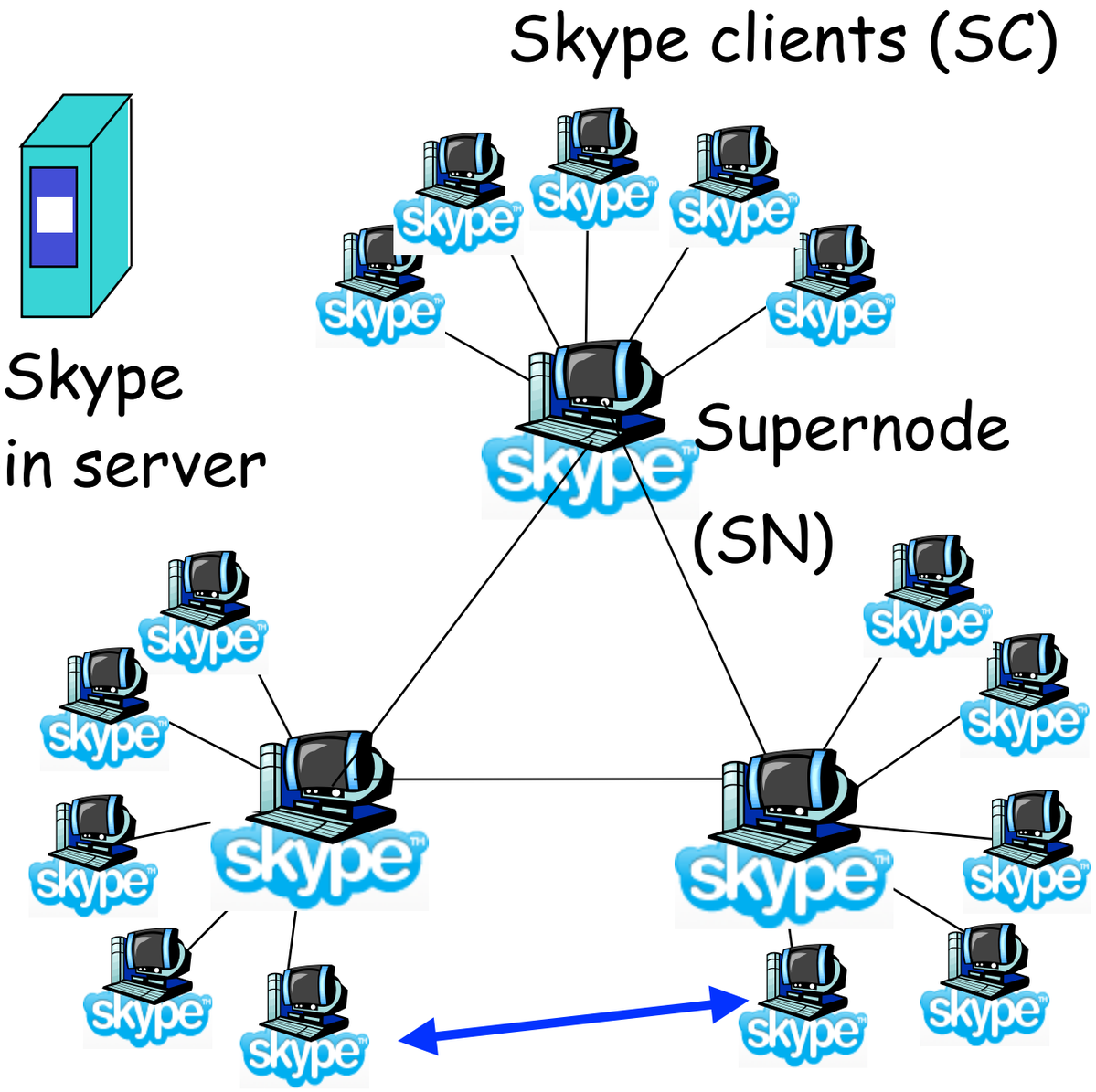
Skype login server

P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SuperNodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs

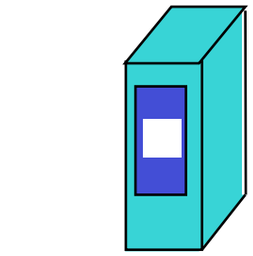


Skype login server



P2P Case study: Skype

- inherently P2P: pairs of users communicate.
- proprietary application-layer protocol (inferred via reverse engineering)
- hierarchical overlay with SuperNodes (SNs)
- Index maps usernames to IP addresses; distributed over SNs



Skype login server