CS 43: Computer Networks
BitTorrent & Content Distribution

Kevin Webb
Swarthmore College
September 28, 2017

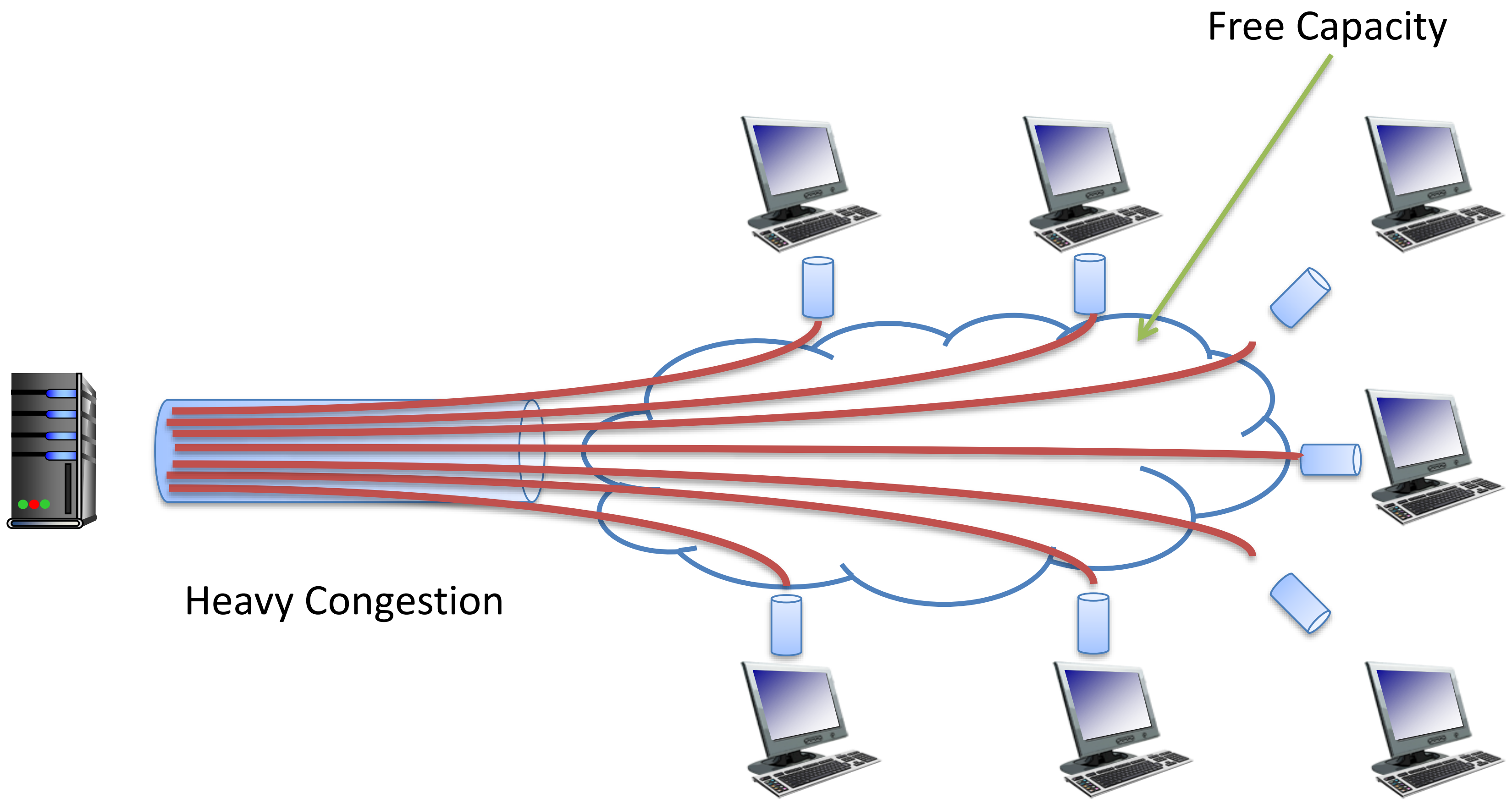
Agenda

- BitTorrent
  - Cooperative file transfers
- Briefly: Distributed Hash Tables
  - Finding things without central authority
- Content distribution networks (CDNs)
  - Add hosts to network to exploit locality
- Video streaming (DASH)

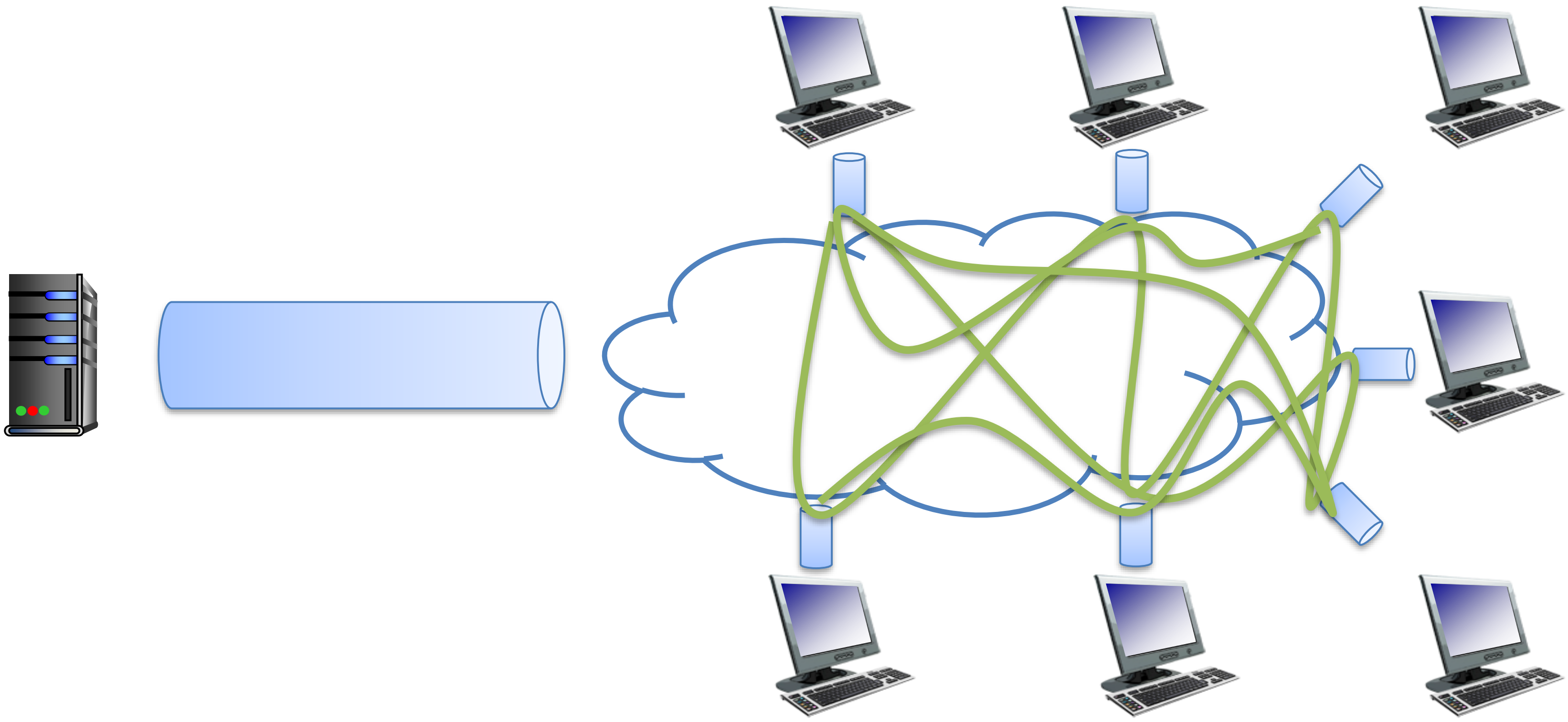
File Transfer Problem

- You want to distribute a file to a large number of people as quickly as possible.

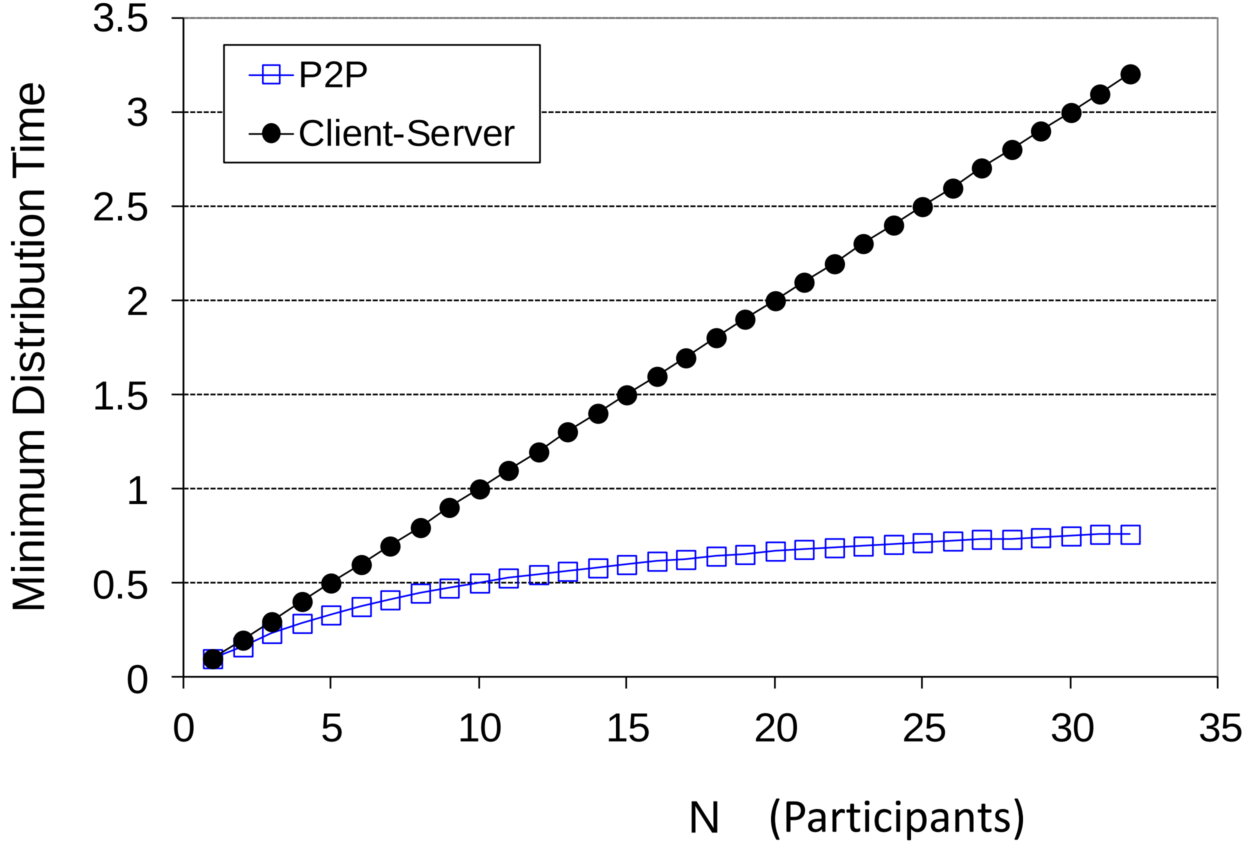
Traditional Client/Server



P2P Solution



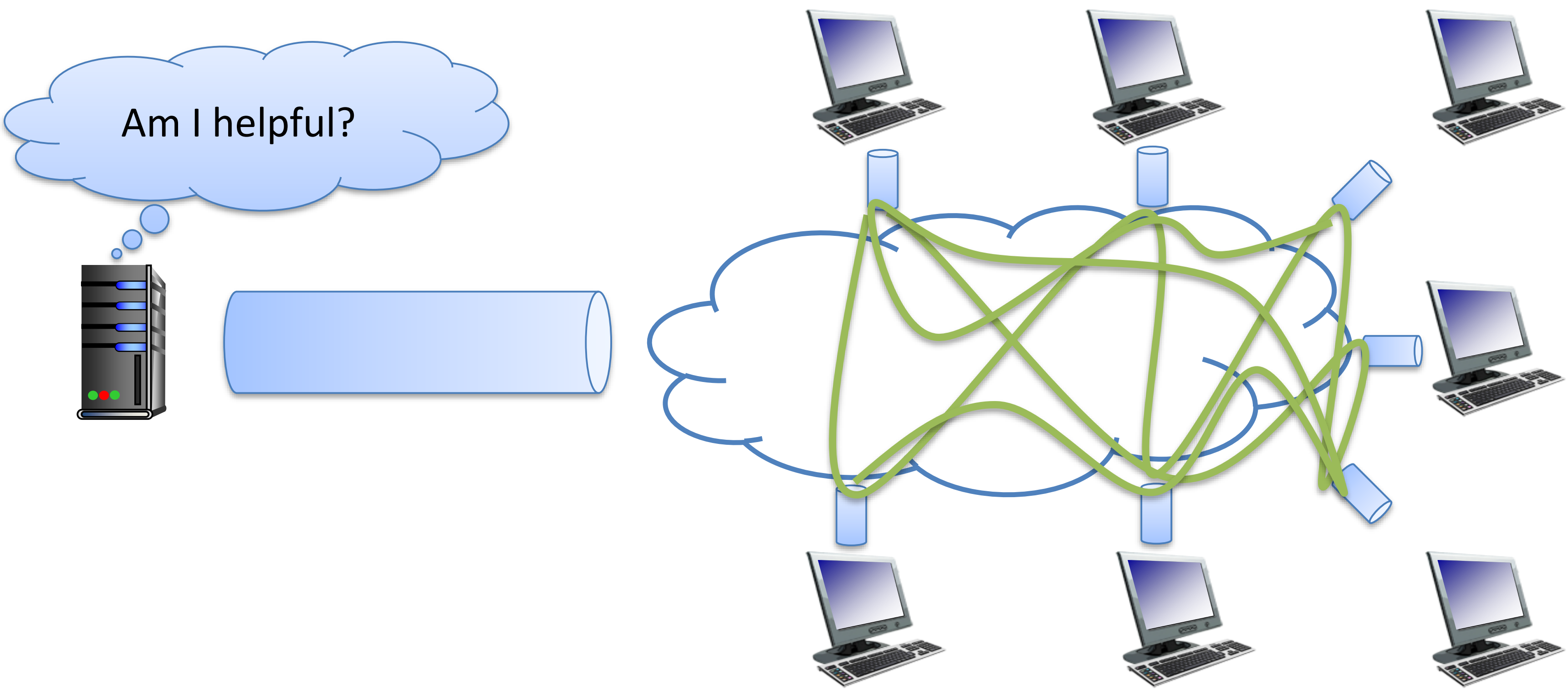
Client-server vs. P2P: example



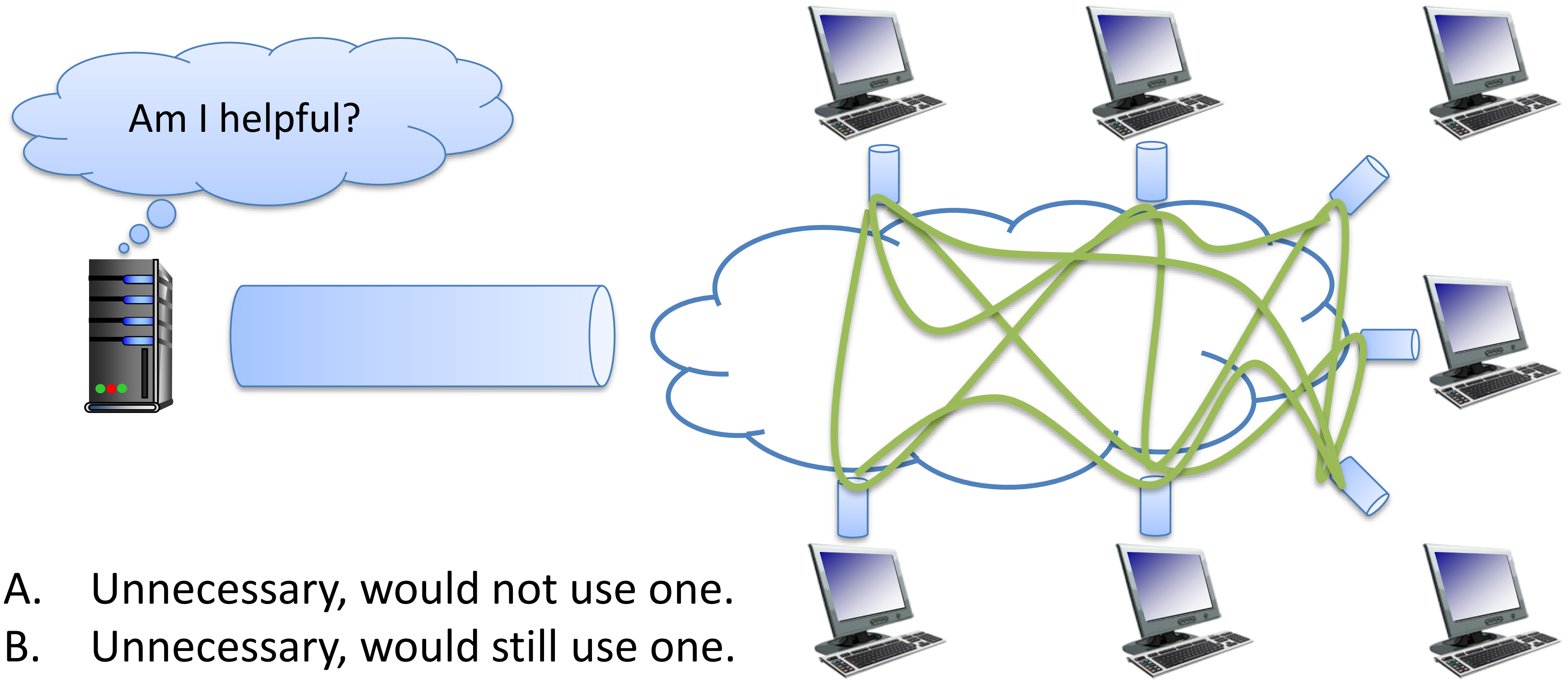
Let F = file size, client UL rate = *u*, server rate = u_s, d = client DL rate

Assumptions: F/u = 1 hour, $u_s = 10u$, $d_{min} \geq u_s$

P2P Solution



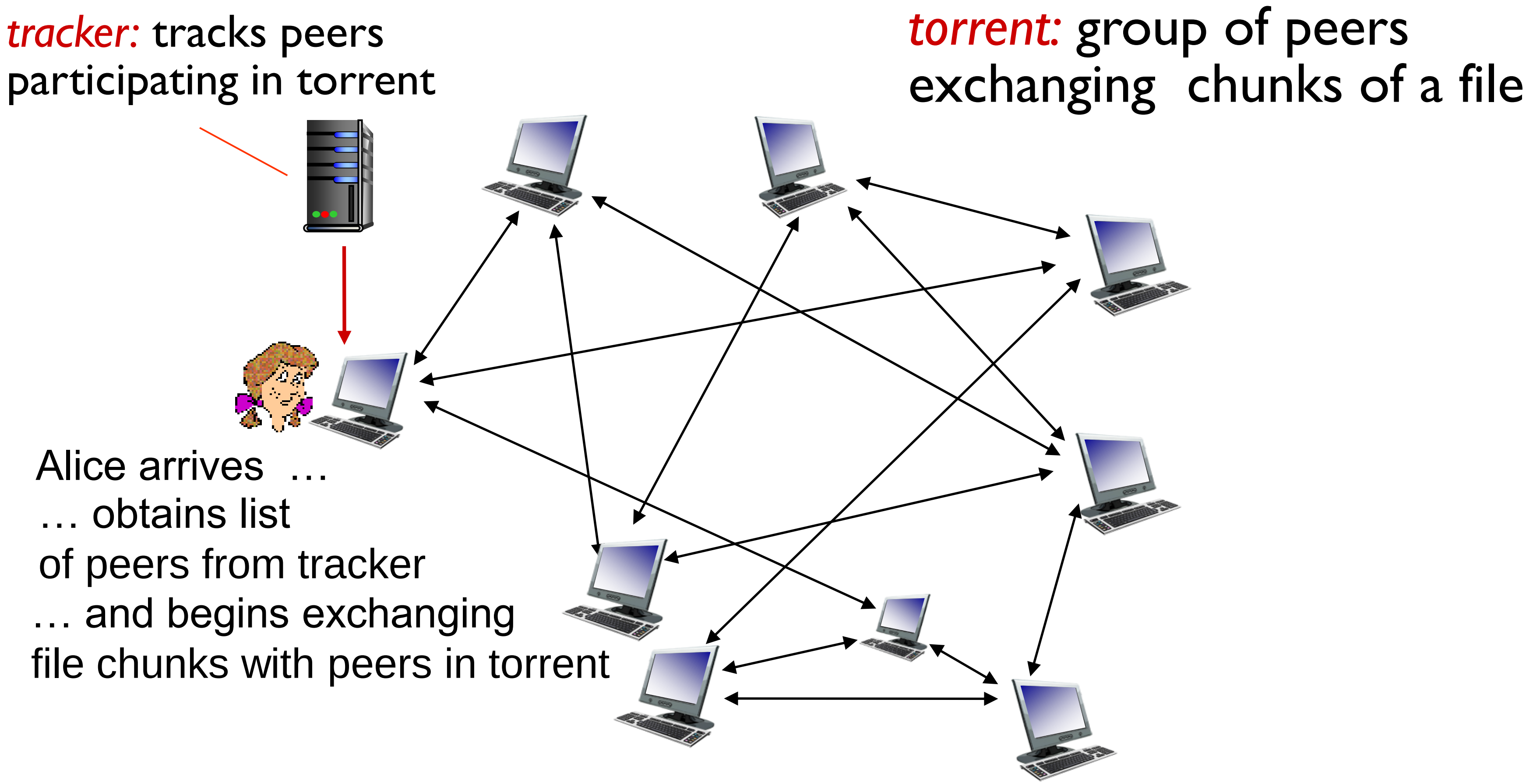
Do we need a centralized server at all? Would you use one for something?



A. Unnecessary, would not use one.
B. Unnecessary, would still use one.
C. Necessary, would have to use it.
D. Something else.

P2P file distribution: BitTorrent

- File divided into chunks (commonly 256 KB)
- Peers in torrent send/receive file chunks



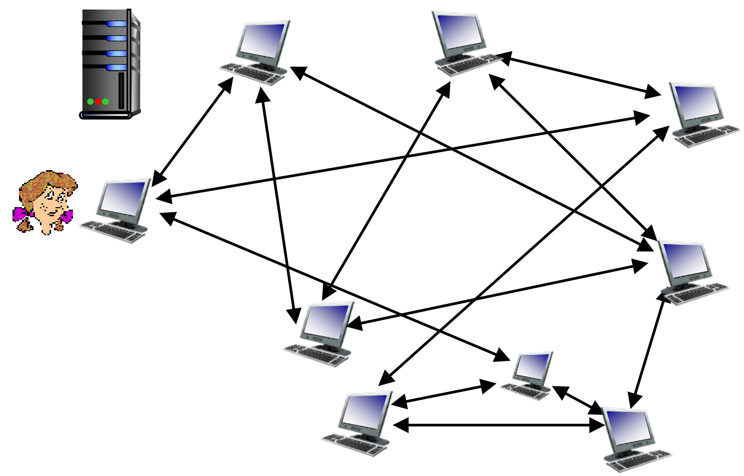
.torrent files

- Contains address of tracker for the file
  - Where can I find other peers?

- Contain a list of file chunks and their cryptographic hashes
  - This ensures pieces are not modified

P2P file distribution: BitTorrent

- Peer joining torrent:
  - has no chunks, but will accumulate them over time from other peers
  - registers with tracker to get list of peers, connects to subset of peers ("neighbors")

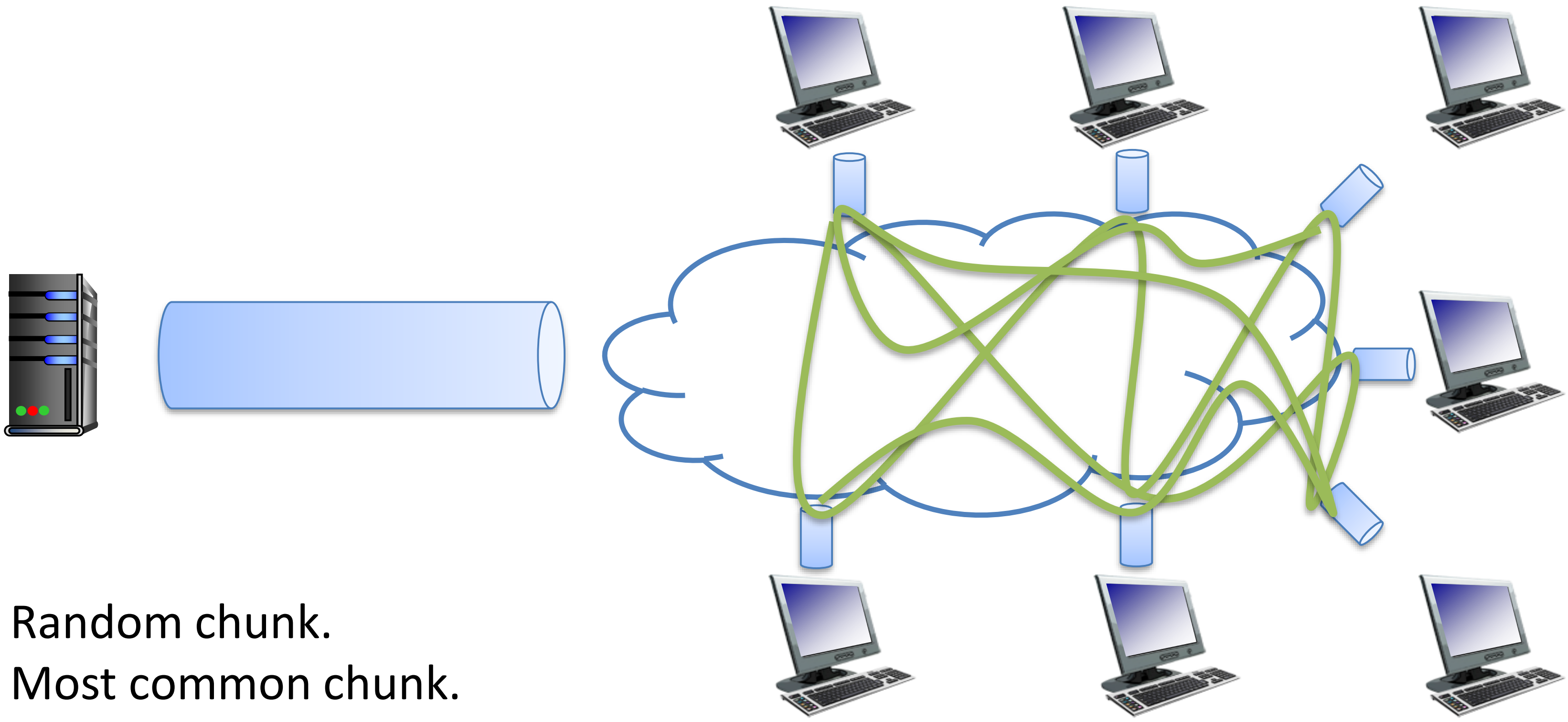


- While downloading, peer uploads chunks to other peers
- Peer may change peers with whom it exchanges chunks
- *Churn:* peers may come and go
- Once peer has entire file, it may (selfishly) leave or (altruistically) remain in torrent

Requesting Chunks

- At any given time, peers have different subsets of file chunks.
- Periodically, each asks peers for list of chunks that they have.

If you’re trying to receive a file, which chunk should you request next?



A. Random chunk.
B. Most common chunk.
C. Least common chunk.
D. Some other chunk.
E. It doesn’t matter.

Requesting Chunks

- At any given time, peers have different subsets of file chunks.
- Periodically, each asks peers for list of chunks that they have.
- In BitTorrent: Peers request rarest chunks first.

Sending Chunks

- A node sends chunks to those four peers currently sending it chunks *at highest rate*
  - other peers are choked (do not receive chunks)
  - re-evaluate top 4 every ~10 secs

- Every 30 seconds: randomly select another peer, start sending chunks
  - "optimistically unchoke" this peer
  - newly chosen peer may join top 4

Academic Interest in BitTorrent

- BitTorrent was enormously successful
  - Large user base
  - Lots of aggregate traffic
  - Invented relatively recently

- Academic Projects
  - Modifications to improve performance
  - Modeling peer communications (auctions)
  - Gaming the system (BitTyrant)

Getting rid of that server…

- Distribute the tracker information using a Distributed Hash Table (DHT)

- A DHT is a lookup structure.
  - Maps keys to an arbitrary value.
  - Works a lot like, well…a hash table.

Recall: Hash Function

- Mapping of any data to an integer
  - E.g., md5sum, sha1, etc.
  - md5: 04c3416cadd85971a129dd1de86cee49
- With a good (cryptographic) hash function:
  - Hash values *very* likely to be unique
  - Near-impossible to find collisions (hashes spread out)

Recall: Hash table

- N buckets
- Key-value pair is assigned bucket i
  - i = HASH(key)%N
- Easy to look up value based on key
- Multiple key-value pairs assigned to each bucket

Distributed Hash Table (DHT)

- DHT: a *distributed P2P database*

- Distribute the (k, v) pairs across the peers
  - key: ss number; value: human name
  - key: file name; value: BT tracker peer(s)

- Same interface as standard HT: (key, value) pairs
  - get(key) – send key to DHT, get back value
  - put(key, value) – modify stored value at the given key

Challenges

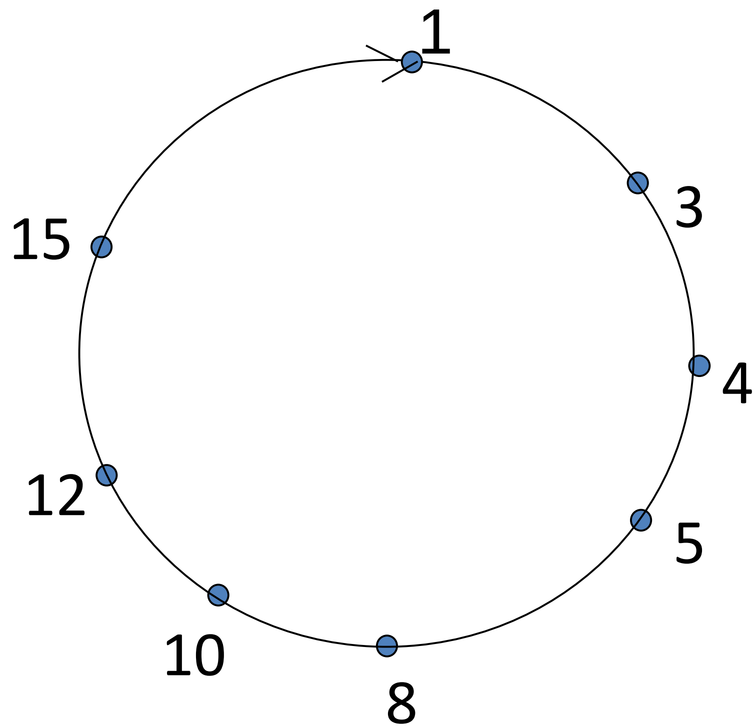
- How do we assign (key, value) pairs to nodes?

- How do we find them again quickly?

- What happens if nodes join/leave?

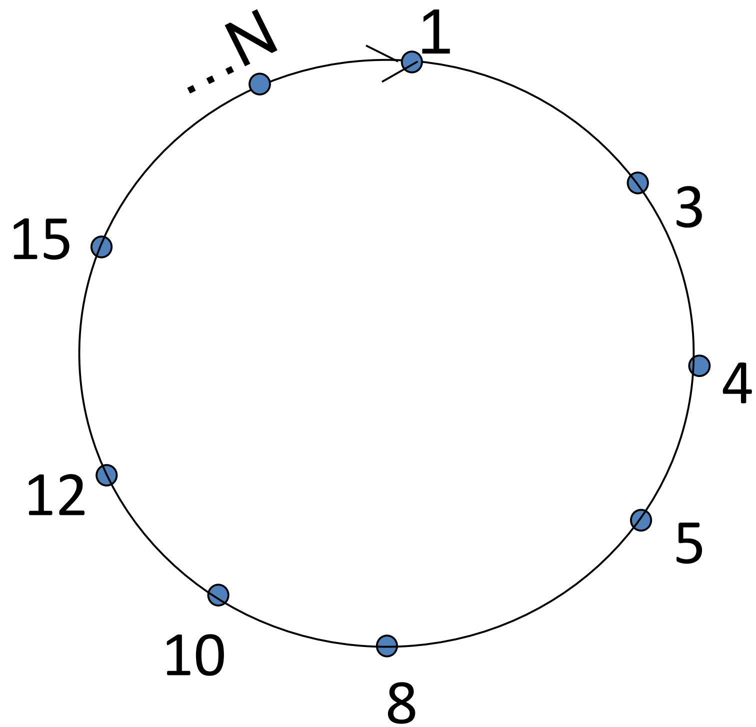
- Basic idea:
  - Convert each key to an integer via hash
  - Assign integer to each peer via hash
  - Store (key, value) pair at the peer closest to the key

Circular DHT Overlay



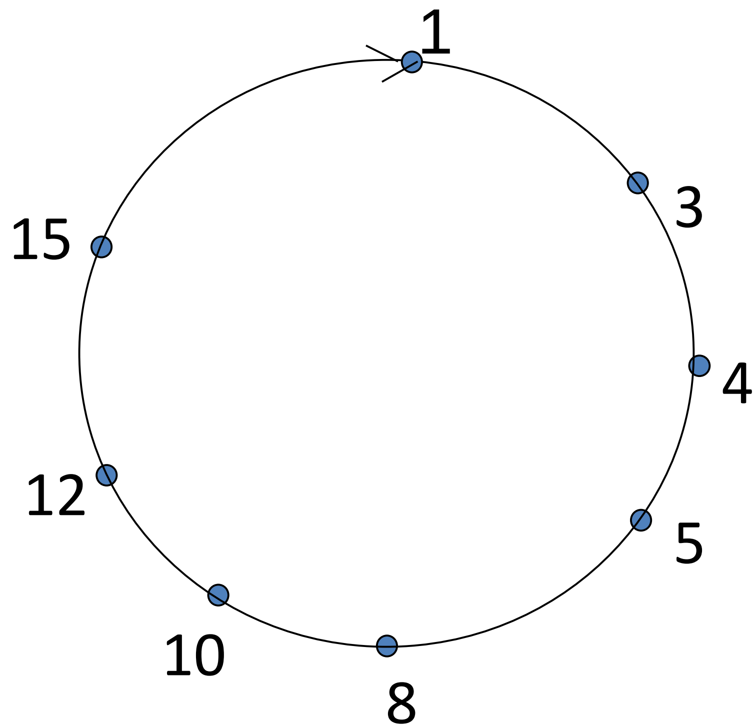
- Simplest form: each peer *only* aware of immediate successor and predecessor.

Circular DHT Overlay



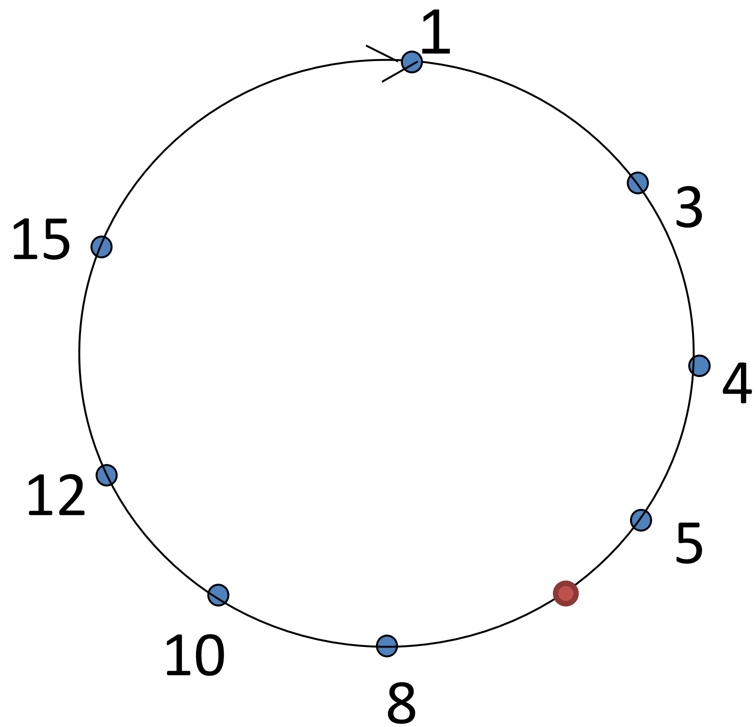
- Simplest form: each peer *only* aware of immediate successor and predecessor.

Circular DHT Overlay



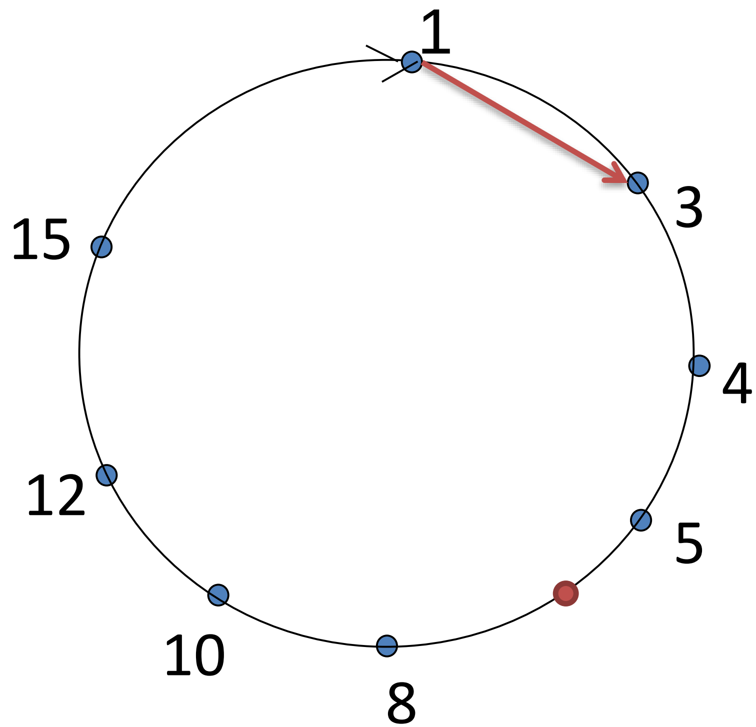
- Example: Node 1 wants key "Led Zeppelin IV"
  - Hash the key

Circular DHT Overlay



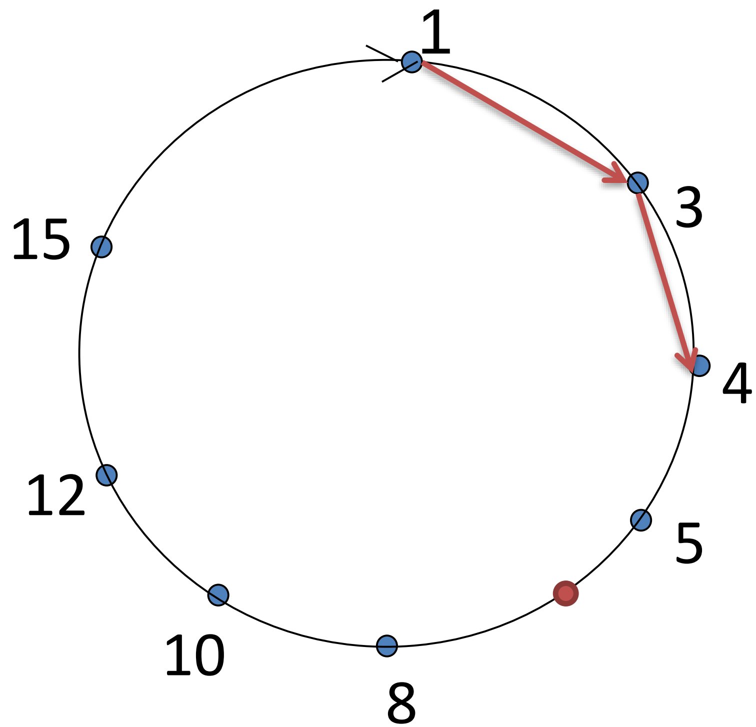
- Example: Node 1 wants key "Led Zeppelin IV"
  - Hash the key (suppose it gives us 6)

Circular DHT Overlay



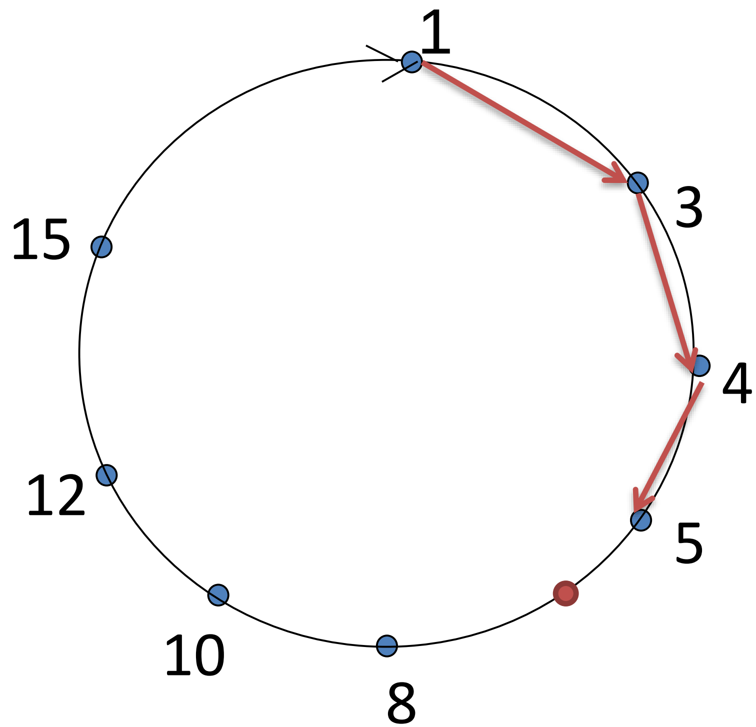
- Example: Node 1 wants key "Led Zeppelin IV"
  - Hash the key (suppose it gives us 6)

Circular DHT Overlay



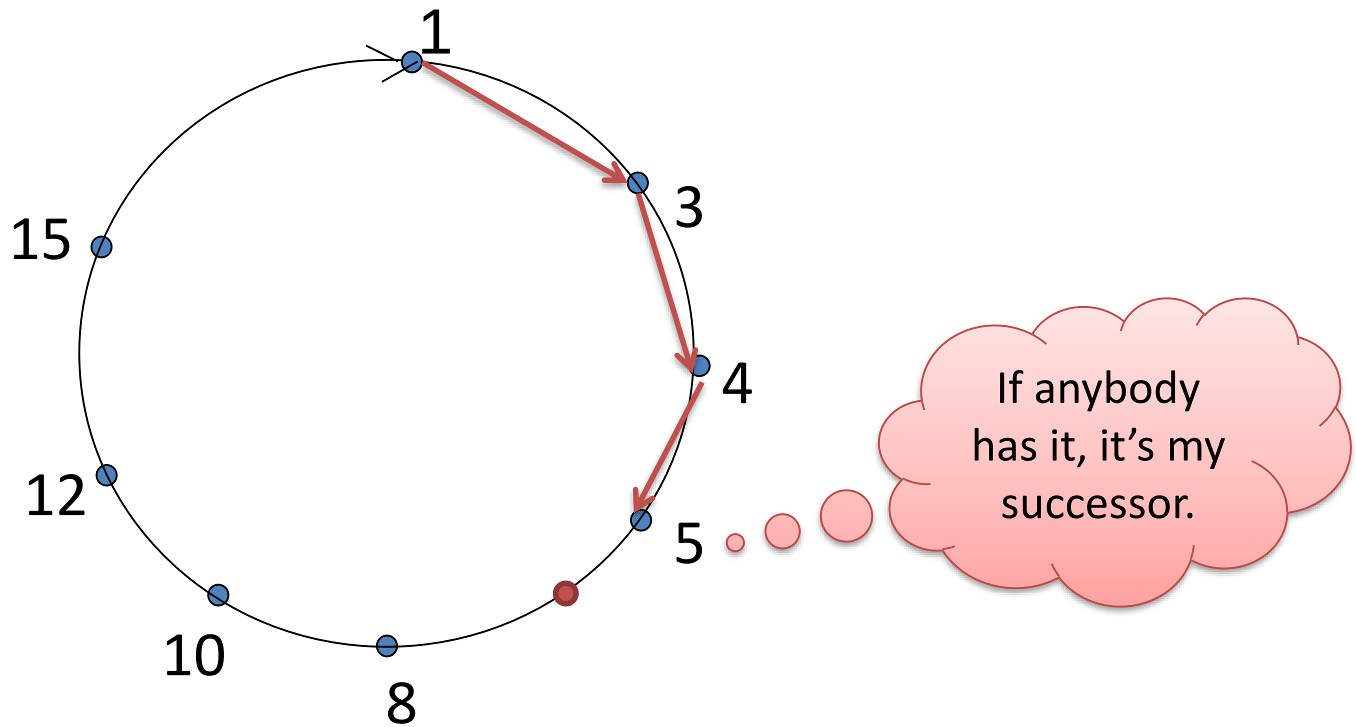
- Example: Node 1 wants key "Led Zeppelin IV"
  - Hash the key (suppose it gives us 6)

Circular DHT Overlay



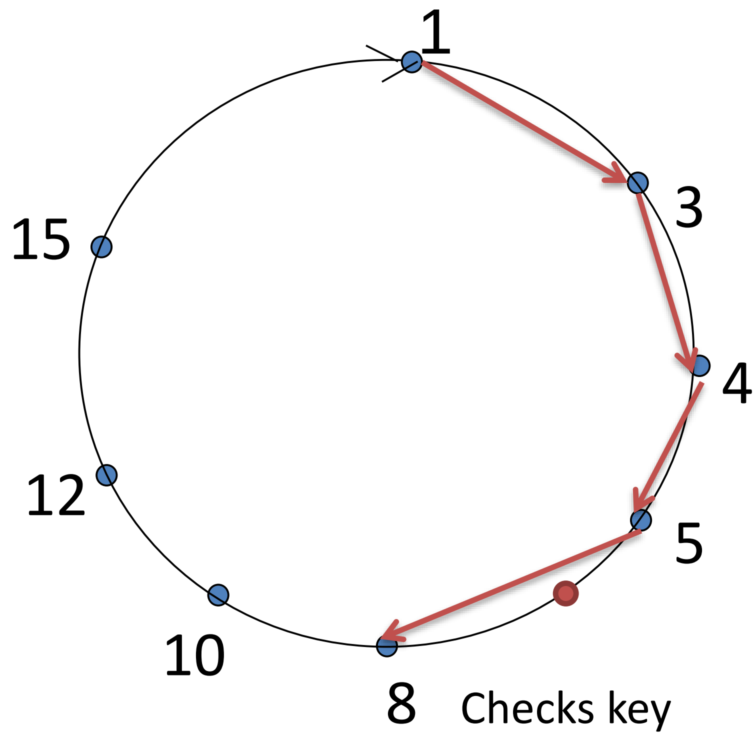
- Example: Node 1 wants key "Led Zeppelin IV"
  - Hash the key (suppose it gives us 6)

Circular DHT Overlay



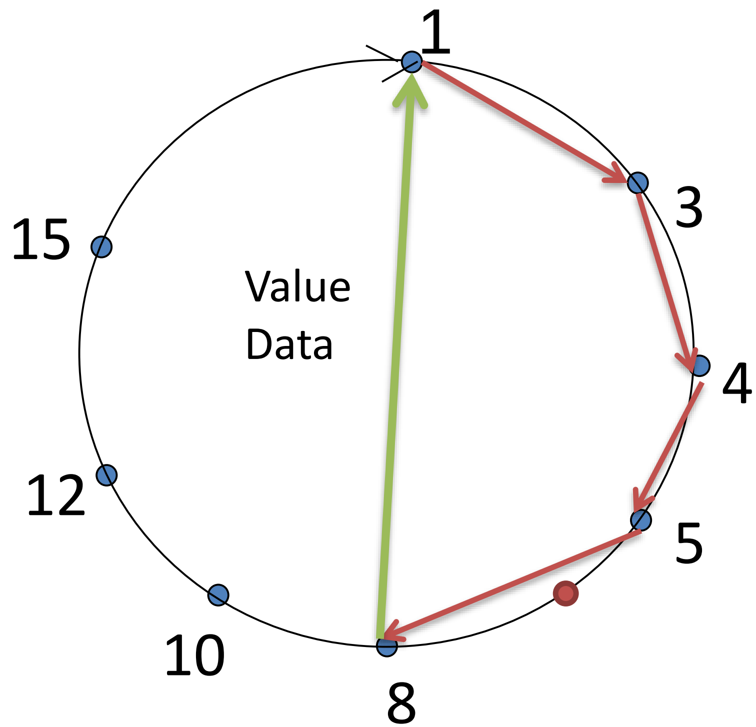
- Example: Node 1 wants key "Led Zeppelin IV"
  - Hash the key (suppose it gives us 6)

Circular DHT Overlay



- Example: Node 1 wants key "Led Zeppelin IV"
  - Hash the key (suppose it gives us 6)

Circular DHT Overlay



- Example: Node 1 wants key “Led Zeppelin IV”
  - Hash the key (suppose it gives us 6)

Given N nodes, what is the complexity (number of messages) of finding a value when each peer knows its successor?

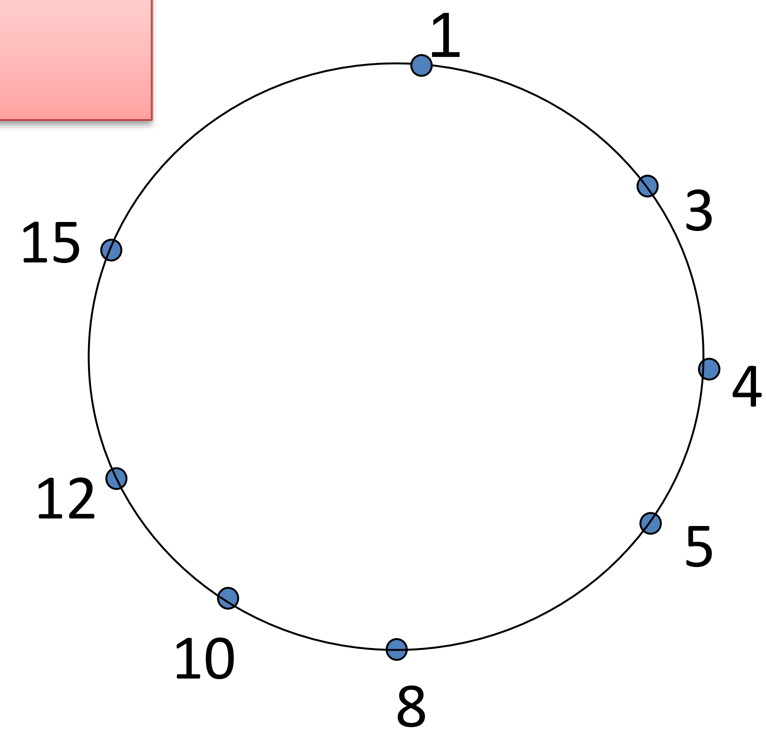
Can we do better?
How?

A. O(log n)

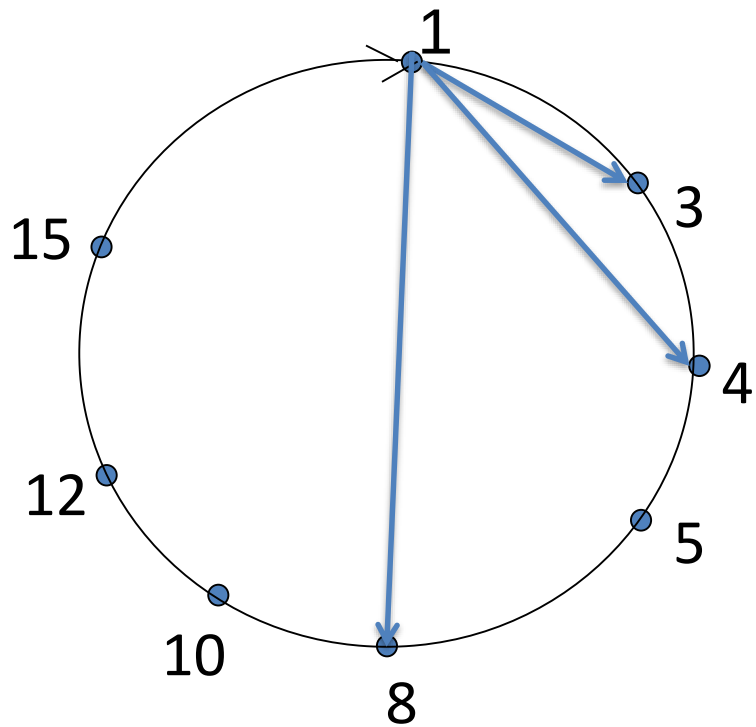
B. O(n)

C. $O(n^2)$

D. $O(2^n)$



Reducing Message Count



- Store successors that are 1, 2, 4, 8, …, N/2 away.
- Can jump up to half way across the ring at once.
- Cut the search space in half - lookups take O(log N) messages.

More DHT Info

- How do nodes join/leave?
- How does cryptographic hashing work?
- How much state does each node store?

More DHT Info

- How do nodes join/leave?
- How does cryptographic hashing work?
- How much state does each node store?

- Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications
- Dynamo: Amazon’s Highly Available Key-value Store

High-Performance Content Distribution

- Problem:
  You have a service that supplies lots of data.
  You want good performance for all users!

(often “lots of data” means media files)

High-Performance Content Distribution

- CDNs applied to all sorts of traffic.
  - You pay for service (e.g., Akamai), they'll host your content very "close" to many users.

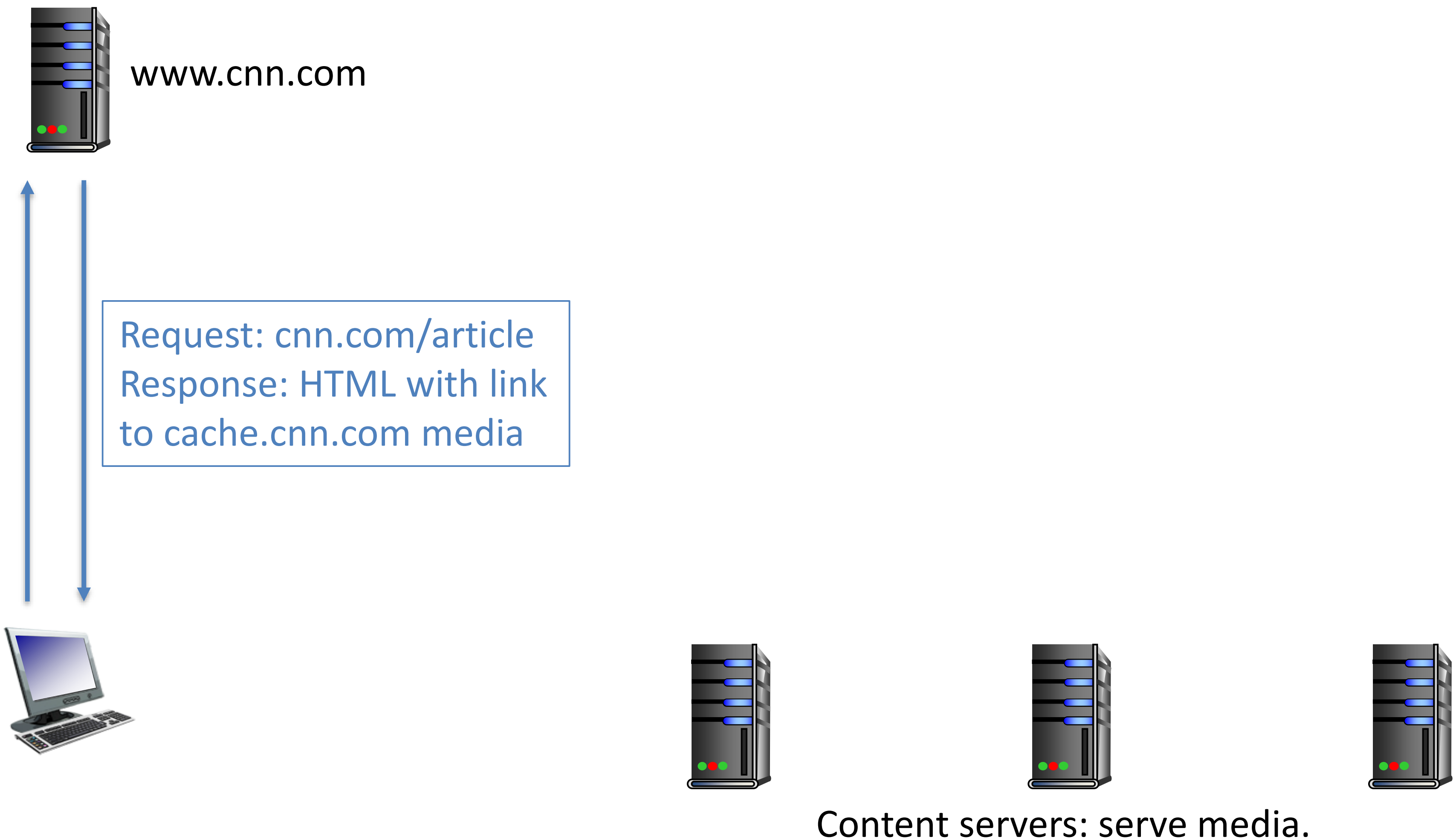
- Major challenges:
  - How do we direct the user to a nearby replica instead of the centralized source?
  - How do we determine which replica is the best to send them to?

Finding the CDN

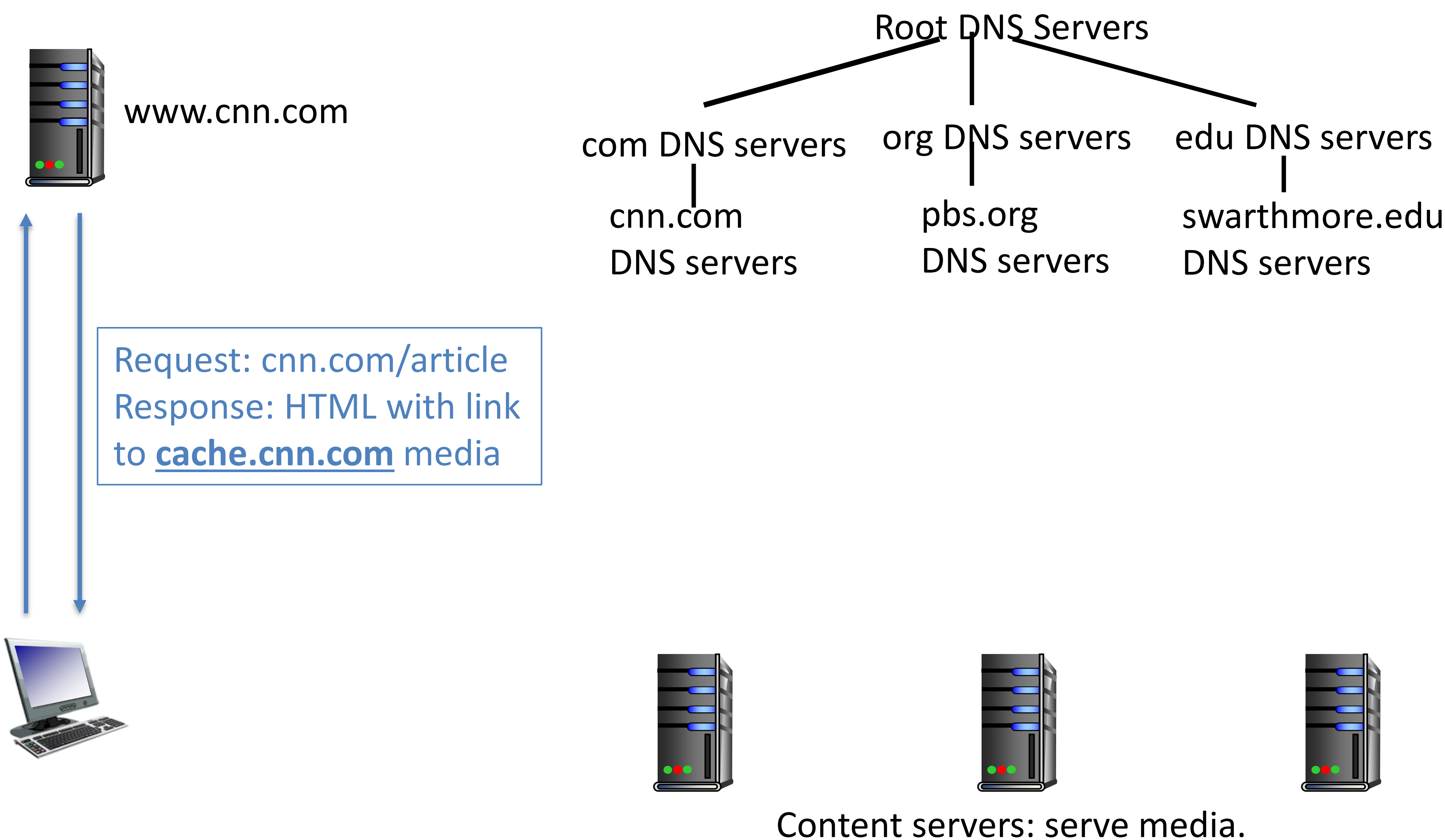
- Three main options:
  - Application redirect (e.g., HTTP)
  - “Anycast” routing
  - DNS resolution (most popular in practice)

- Example: CNN + Akamai

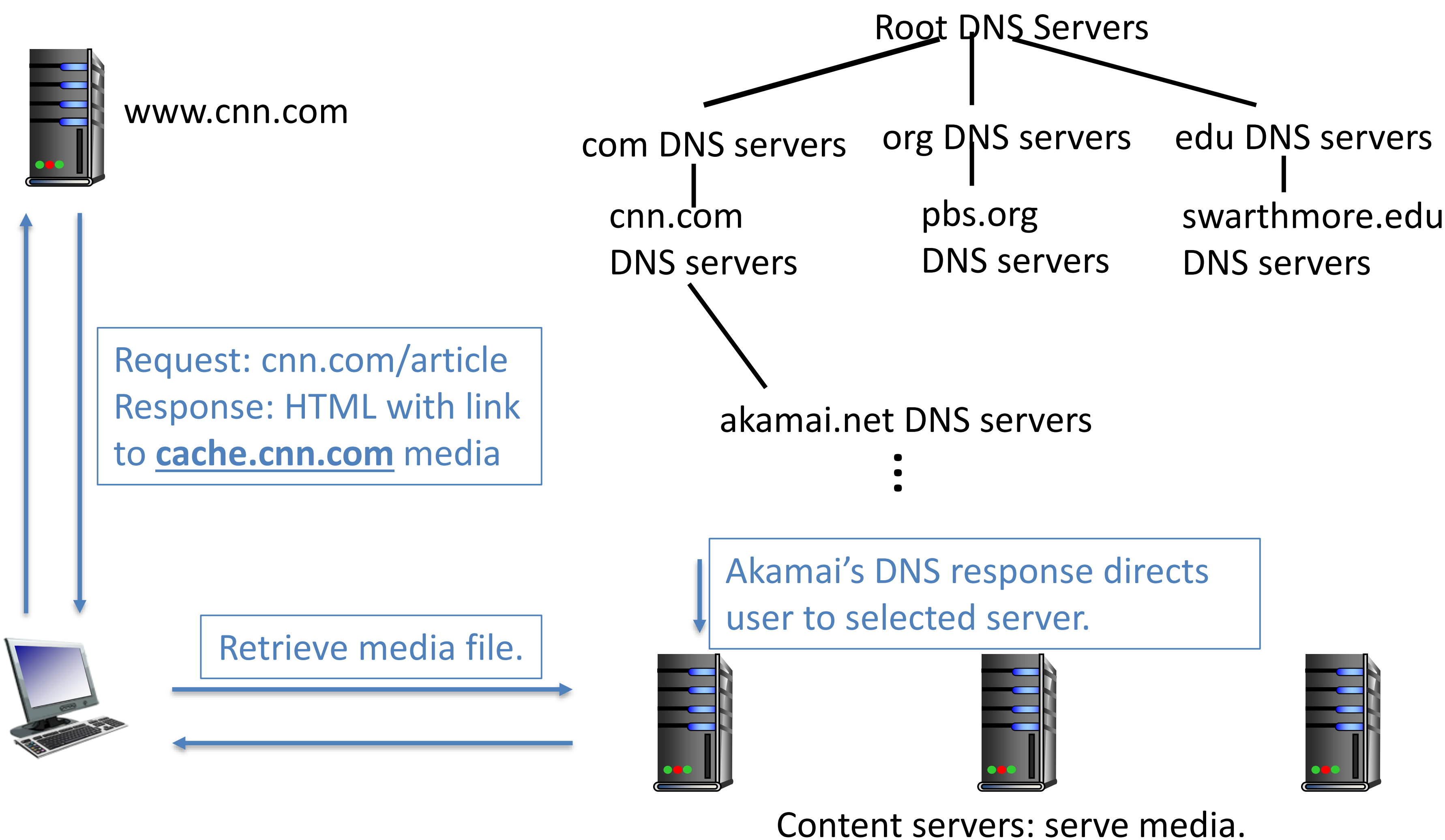
CNN + Akamai


www.cnn.com
Request: cnn.com/article
Response: HTML with link
to cache.cnn.com media
Content servers: serve media.

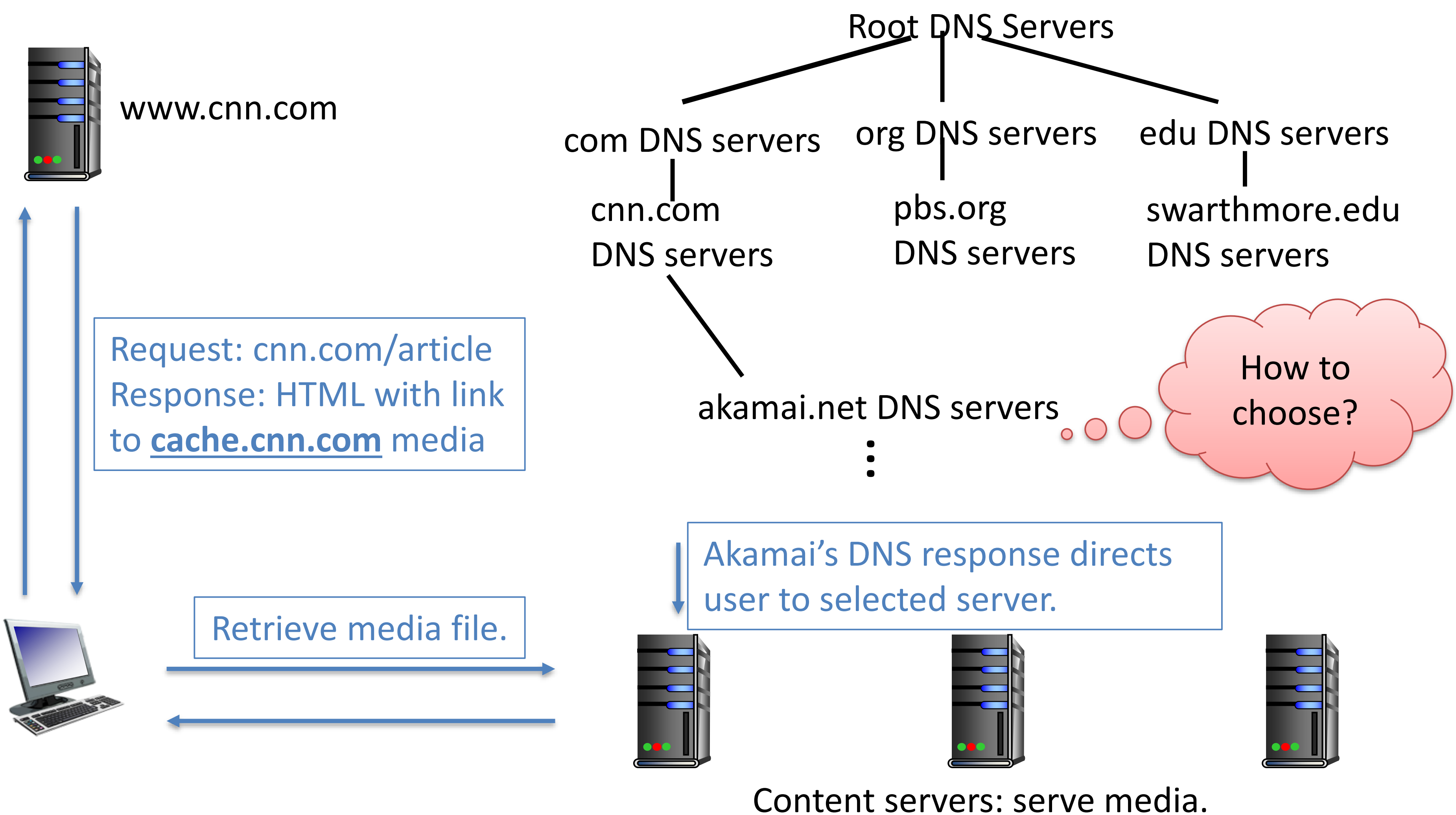
CNN + Akamai


Root DNS Servers
www.cnn.com
com DNS servers
org DNS servers
edu DNS servers
cnn.com
DNS servers
pbs.org
DNS servers
swarthmore.edu
DNS servers
Request: cnn.com/article
Response: HTML with link
to cache.cnn.com media
Content servers: serve media.

CNN + Akamai


Root DNS Servers
com DNS servers
org DNS servers
edu DNS servers
cnn.com DNS servers
pbs.org DNS servers
swarthmore.edu DNS servers
akamai.net DNS servers
www.cnn.com
Request: cnn.com/article
Response: HTML with link to cache.cnn.com media
Retrieve media file.
Akamai's DNS response directs user to selected server.
Content servers: serve media.

CNN + Akamai


Root DNS Servers
www.cnn.com
com DNS servers
org DNS servers
edu DNS servers
cnn.com
DNS servers
pbs.org
DNS servers
swarthmore.edu
DNS servers
Request: cnn.com/article
Response: HTML with link
to cache.cnn.com media
akamai.net DNS servers
How to
choose?
Akamai's DNS response directs
user to selected server.
Retrieve media file.
Content servers: serve media.

Which metric is most important when choosing a server? (CDN or otherwise)

A. RTT latency

B. Data transfer rate / throughput

C. Hardware ownership

D. Geographic location

E. Some other metic(s)   (such as?)

This is the CDN operator's secret sauce!

Streaming Media

- Straightforward approach: simple GET

- Challenges:
  - Dynamic network characteristics
  - Varying user device capabilities
  - User mobility

Dynamic Adaptive Streaming over HTTP (DASH)

- Encode several versions of the same media file
  - low / medium / high / ultra quality

- Break each file into chunks

- Create a “manifest” to map file versions to chunks / video time offset

Dynamic Adaptive Streaming over HTTP (DASH)

- Client requests manifest file, chooses version
- Requests new chunks as it plays existing ones
- Can switch between versions at any time!

Summary

- Peer-to-peer architectures for:
  - High performance: BitTorrent
  - Decentralized lookup: DHTs
- CDNs: locating “good” replica for media server
- DASH: streaming despite dynamic conditions