CS 43: Computer Networks

10: DHTs and CDNs

October 8, 2020

SWARTHMORE COLLEGE

Slides Courtesy: Kurose & Ross, K. Webb, D. Choffnes

Where we are

Application: (So far: HTTP, Email, DNS)
Today: P2P systems, Overlay Networks

Transport: end-to-end connections, reliability

Network: routing

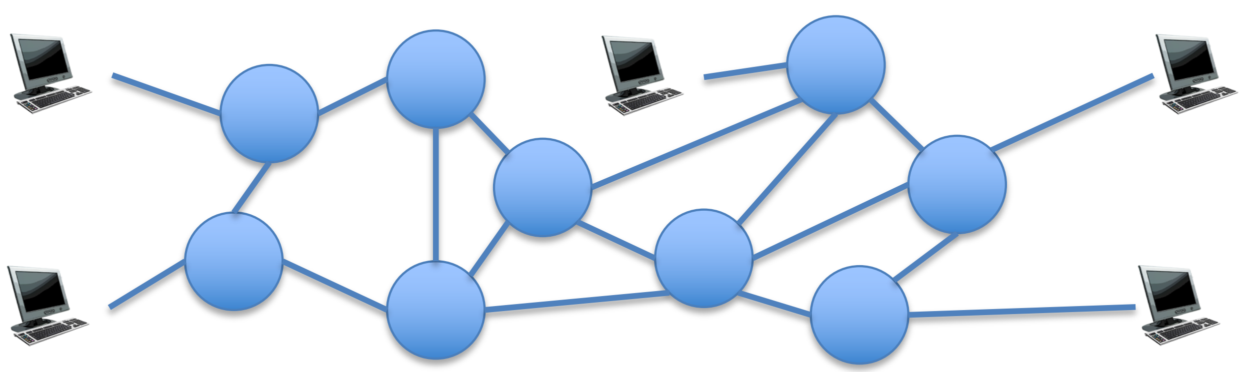
Link (data-link): framing, error detection

Physical: 1's and 0's/bits across a medium
(copper, the air, fiber)

Overlay Network (P2P)

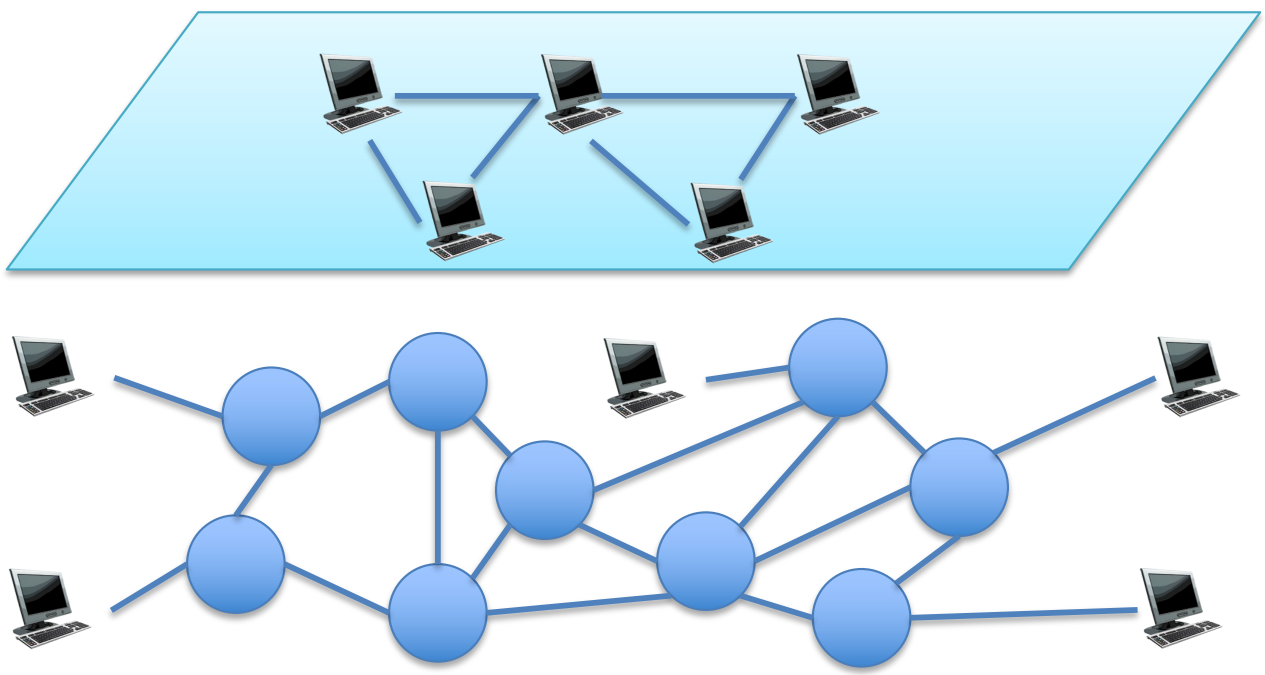
- A network made up of "virtual" or logical links

- Virtual links map to one or more physical links



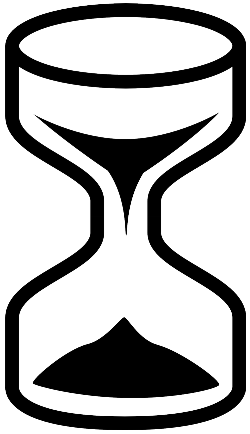
Overlay Network (P2P)

- A network made up of “virtual” or logical links
- Virtual links map to one or more physical links



In our P2P examples with no central server, what's the best mechanism to find content?

A. Flooding each node and querying
B. Maintaining an entire list at each node
C. Some other system that scales

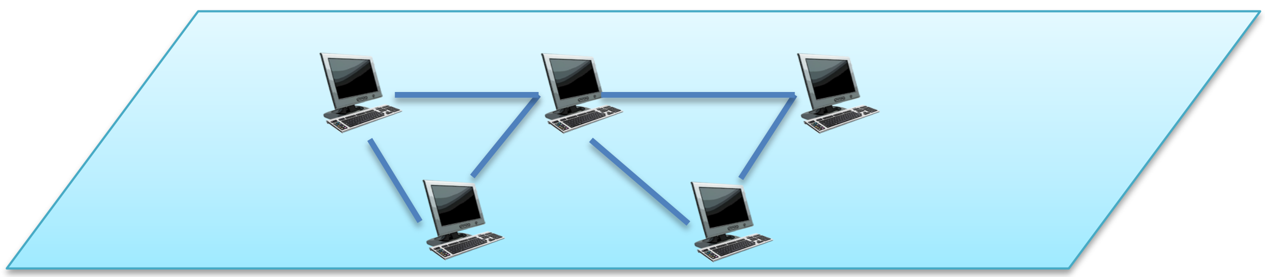


In our P2P examples with no central server, how would we maintain a mapping of content to nodes?

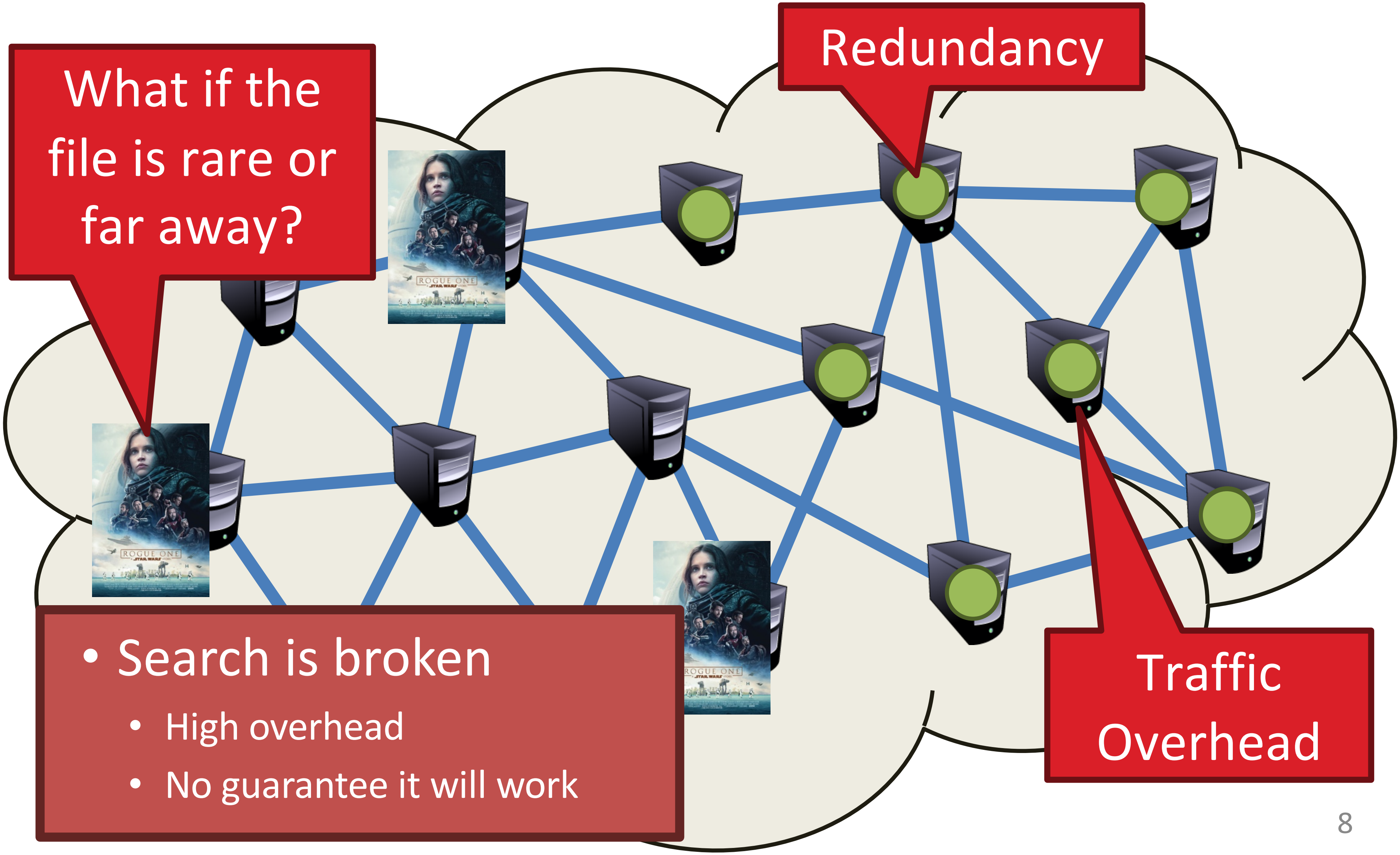
A. Flooding each node and querying
B. Maintaining an entire list at each node
C. Some other system that scales - a Distributed Hash Table.

Unstructured Overlay Networks

- Overlay links form random graphs
- No defined structure
- Examples: Gnutella: links are peer relationships



Unstructured Overlay Issues



- Search is broken
  - High overhead
  - No guarantee it will work

Structured Overlay Networks (I.e. getting rid of that bit-torrent 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 a hash value
- if keys are integers, with n nodes in the network
  - id = key % n
  - 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)

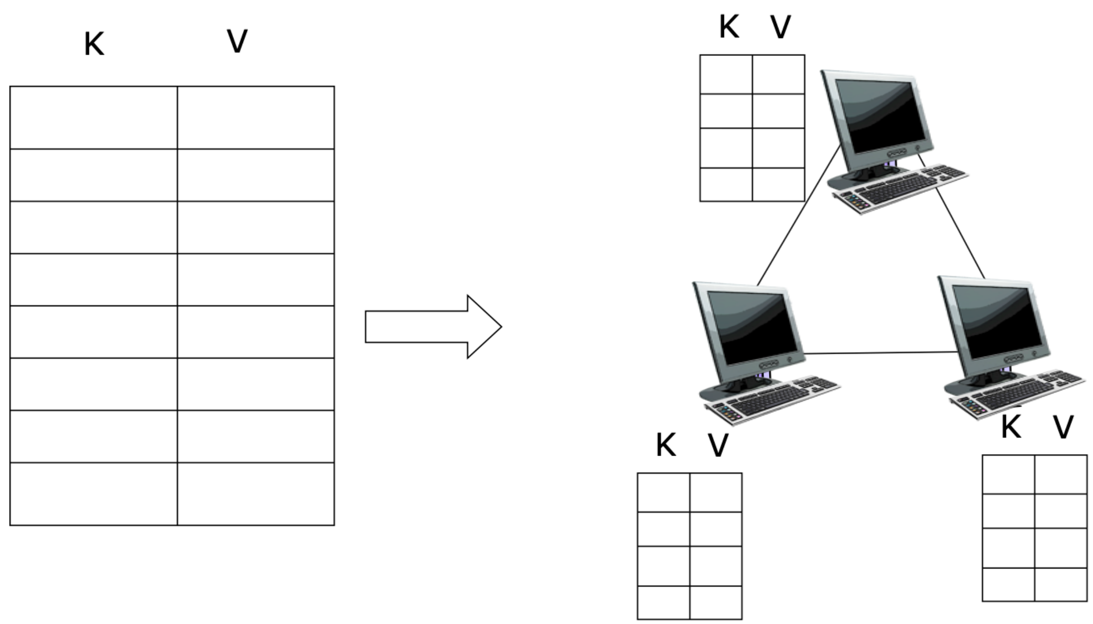
Distributed Hash Table (DHT)

- DHT: a *distributed P2P database*
  - Data items stored by a network of peers
- DHT abstraction:
  - Input: key
  - Output: node that stores the content
- 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

DHT Goals

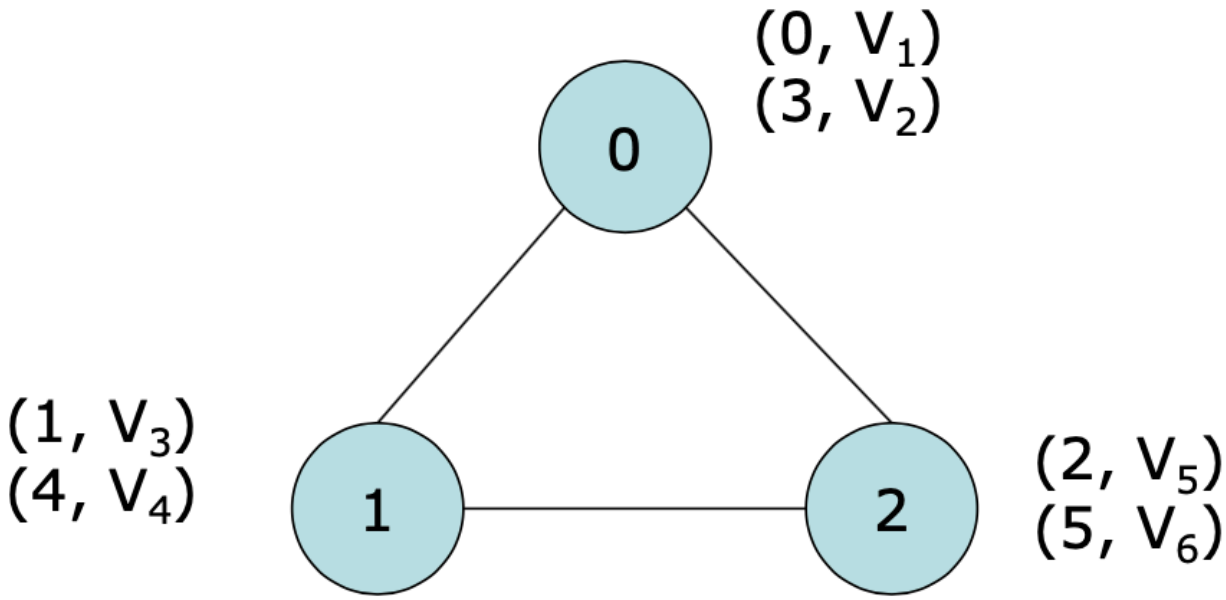
- Scalability: each node does not keep much state
- Performance: small look up latency
- Load balancing: no node is overloaded with a large amount of state
- Dynamic re-configuration: when nodes join and leave the amount of state moved amongst nodes is minimal
- Distributed: no node is more important than others

Distributed Hash Table



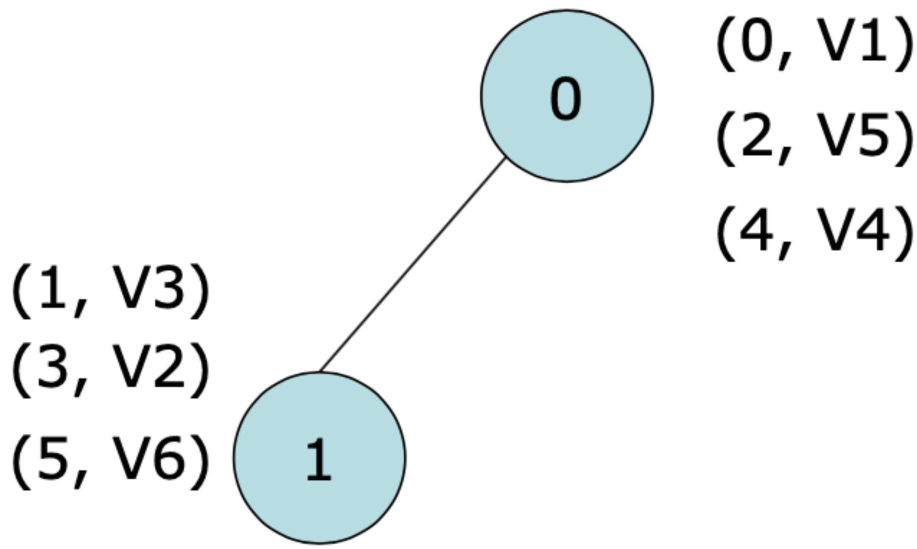
- Used in the real world
  - BitTorrent tracker implementation
  - Content distribution networks
  - Many other distributed systems including botnets

DHT: Strawman approach



- Suppose all the keys are integers
- The number of nodes in the network is n
  - id = key % n

DHT: Strawman approach:



- Node 2 dies
- A large number of data items need to be rehashed
  - id = key % n

DHT: Consistent Hashing

- Consistent hashing:
  - hash node -> identifier space
  - hash key -> identifier space
- Node is responsible for a range of keys
  - Multiple key-value pairs assigned to each node
- A key is stored at a node whose identifier is closest to the key in the identifier space
- All DHTs implement consistent hashing
- They differ in the underlying "geometry"

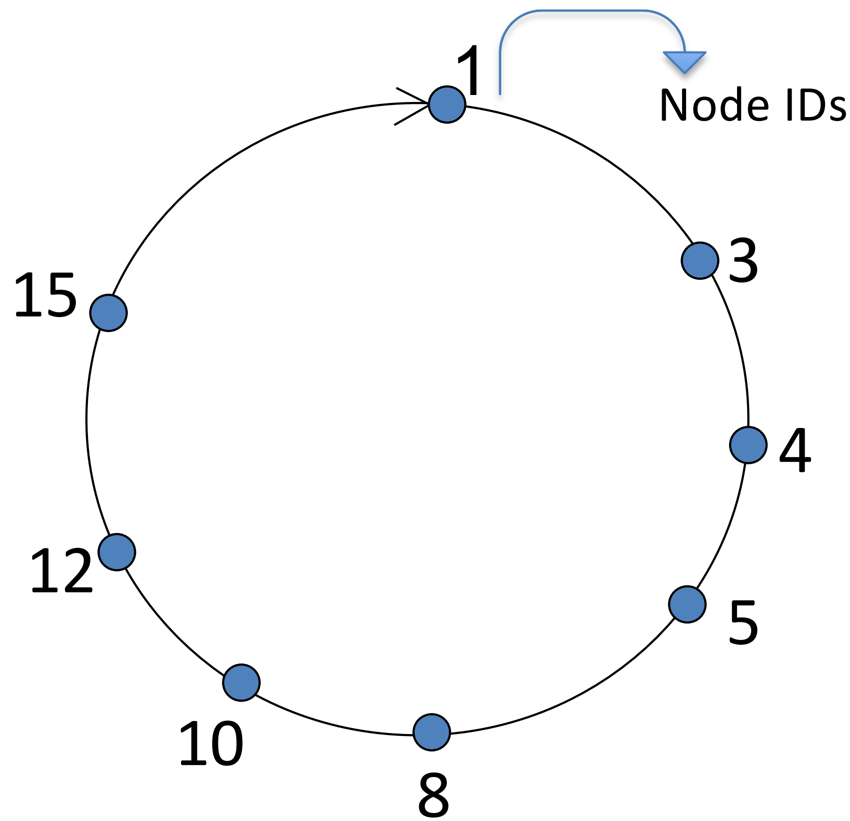
Challenges

- How do we assign (key, value) pairs to nodes?
- How do we find them again quickly?
- What happens if nodes join/leave?

Circular DHT Overlay

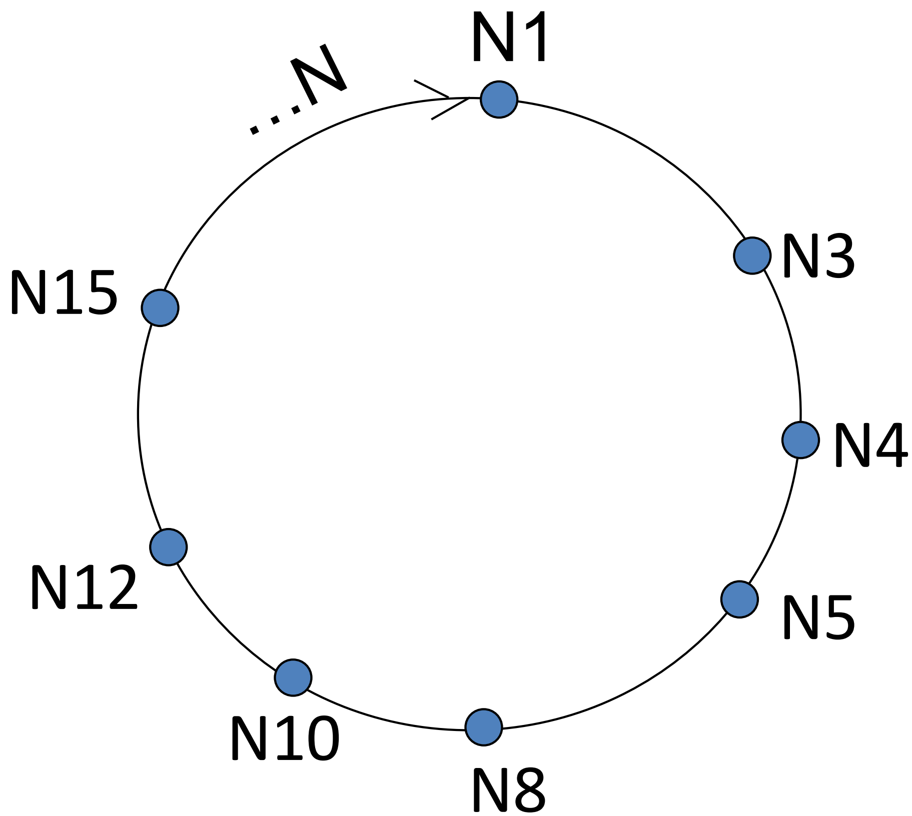
- Hash both node ID and key into an m-bit one-dimension circular identifier space
- Example: 4-bit identifier space [0 – 15]
  – Convert each content key to an integer [0-15] via hash.
  – Convert each node ID to an integer [0 – 15] via hash.
  – The key is stored at its successor: node with next highest integer

Circular DHT Overlay



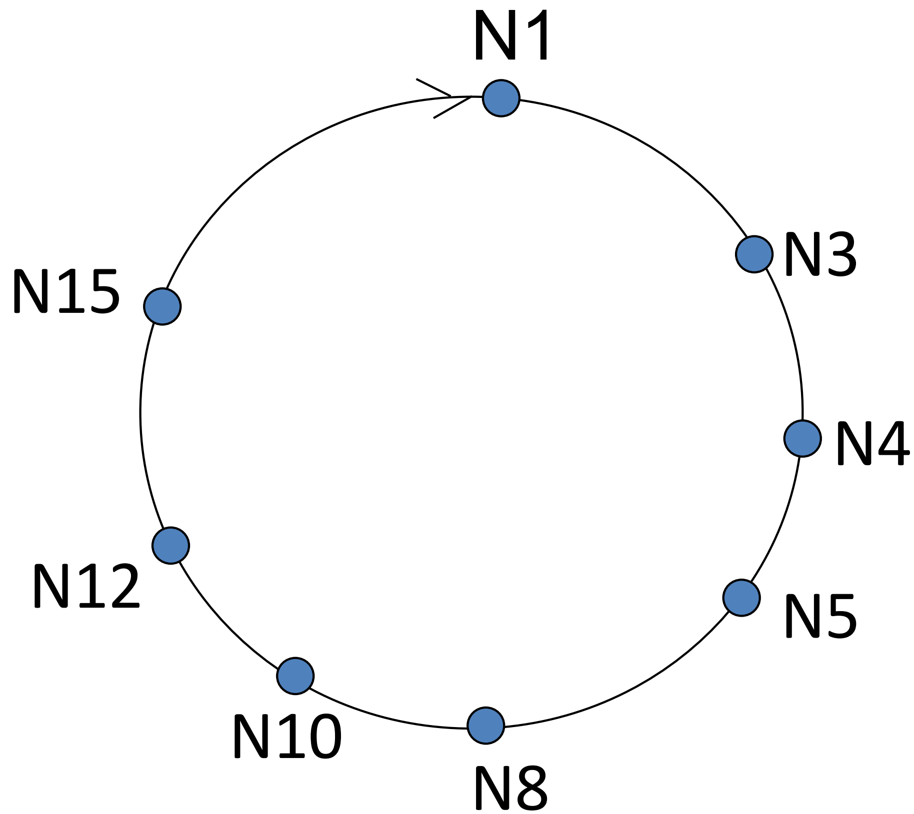
- Simplest form: each node *only* aware of immediate successor and predecessor.

Circular DHT Overlay



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

Circular DHT Overlay



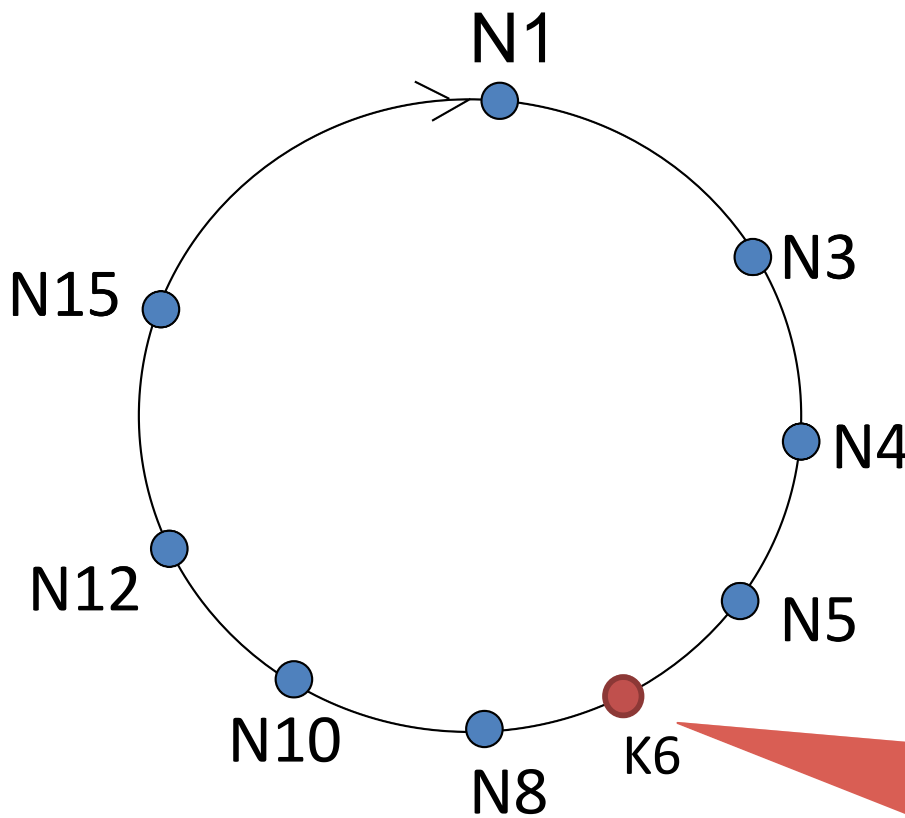
Hash both node id and key onto one- dimension circular identifier space

Each node is assigned an integer ID from the range $[0, 2^n - 1]$

Each key is hashed to an integer ID in the same range $[0, 2^n - 1]$

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

Circular DHT Overlay

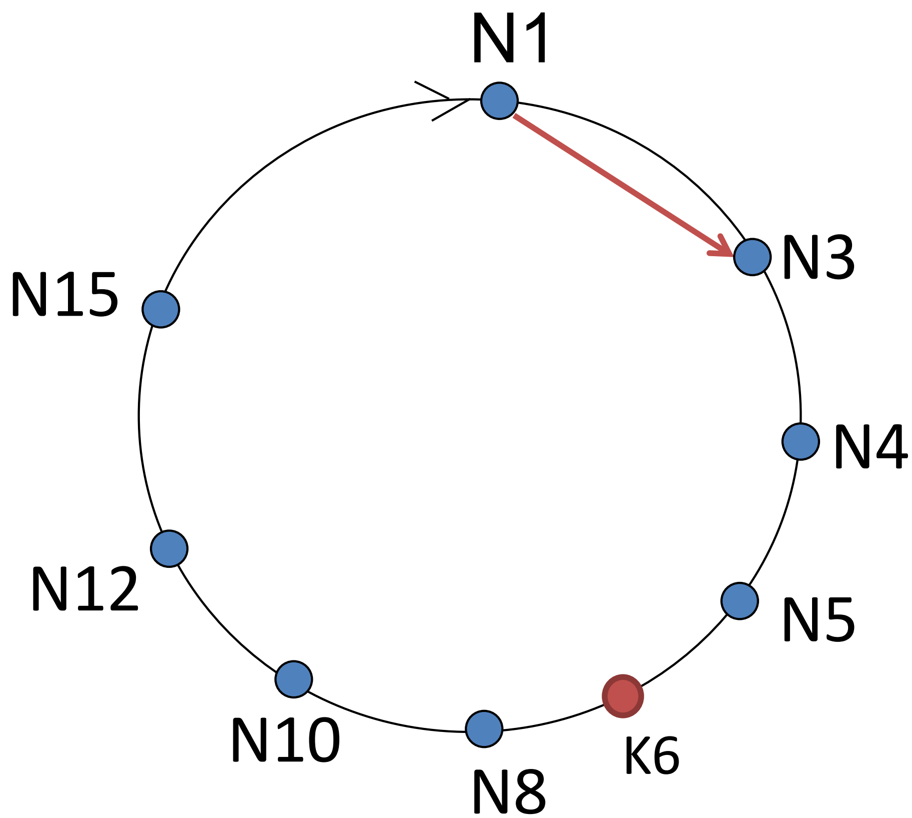


Hash both node id and key onto one- dimension circular identifier space

The key is stored at its successor: node with next highest integer

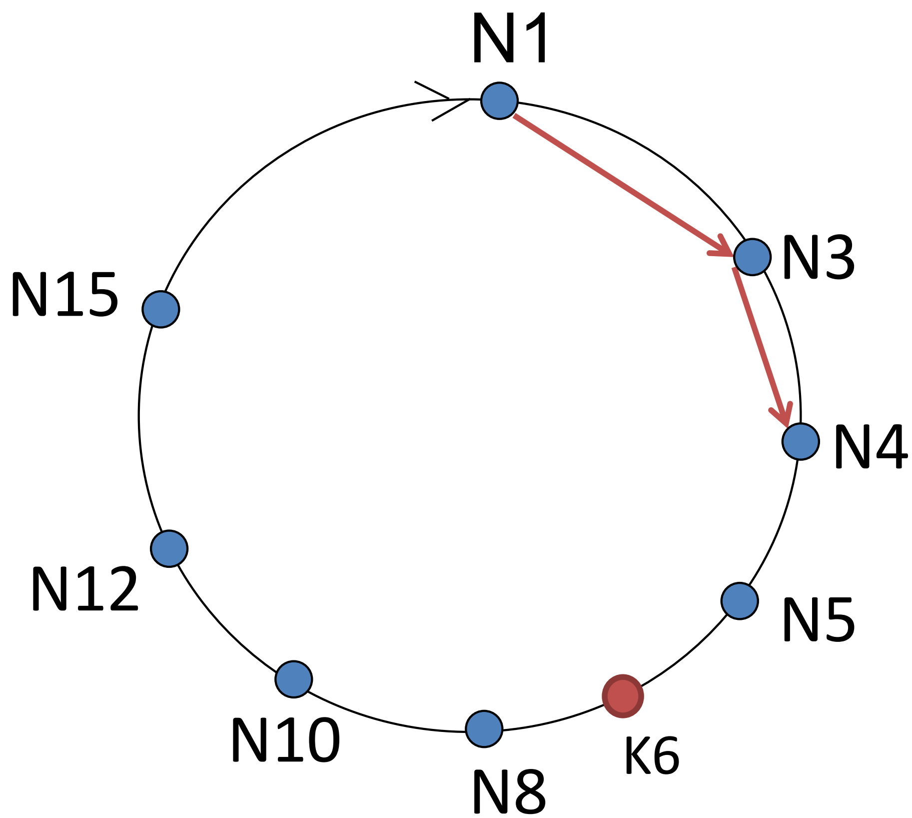
- Example: Node 1 wants key “Led Zeppelin IV”
  - Hash the key “Led Zeppelin IV” = K6

Circular DHT Overlay



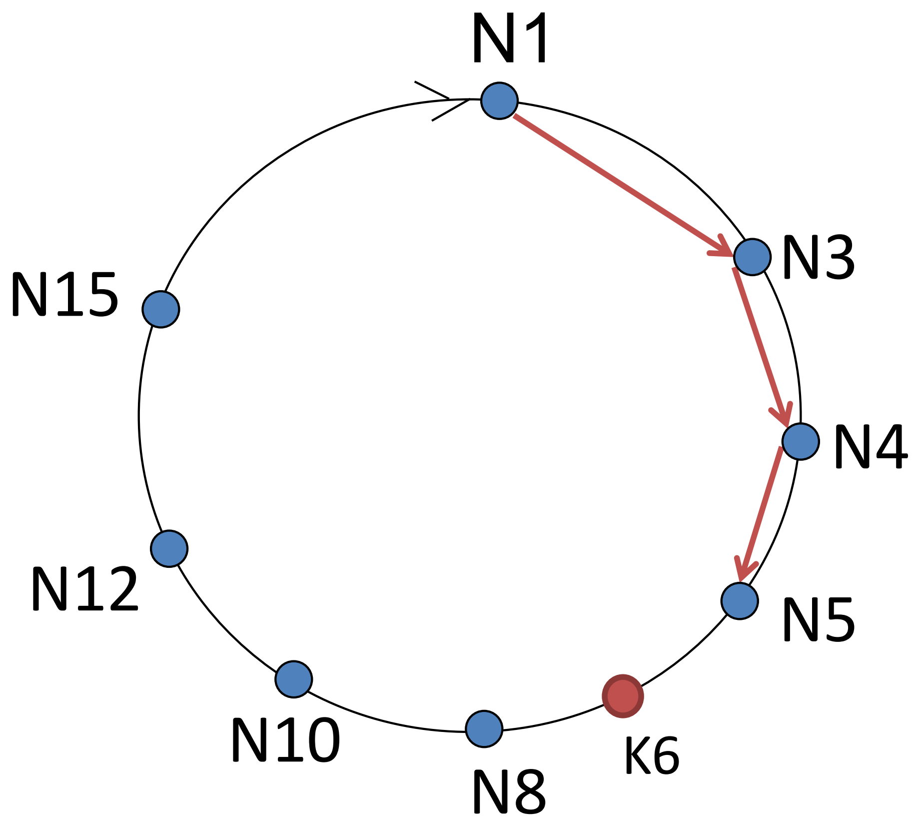
- Example: Node 1 wants key “Led Zeppelin IV”
  - Hash the key “Led Zeppelin IV” = K6

Circular DHT Overlay



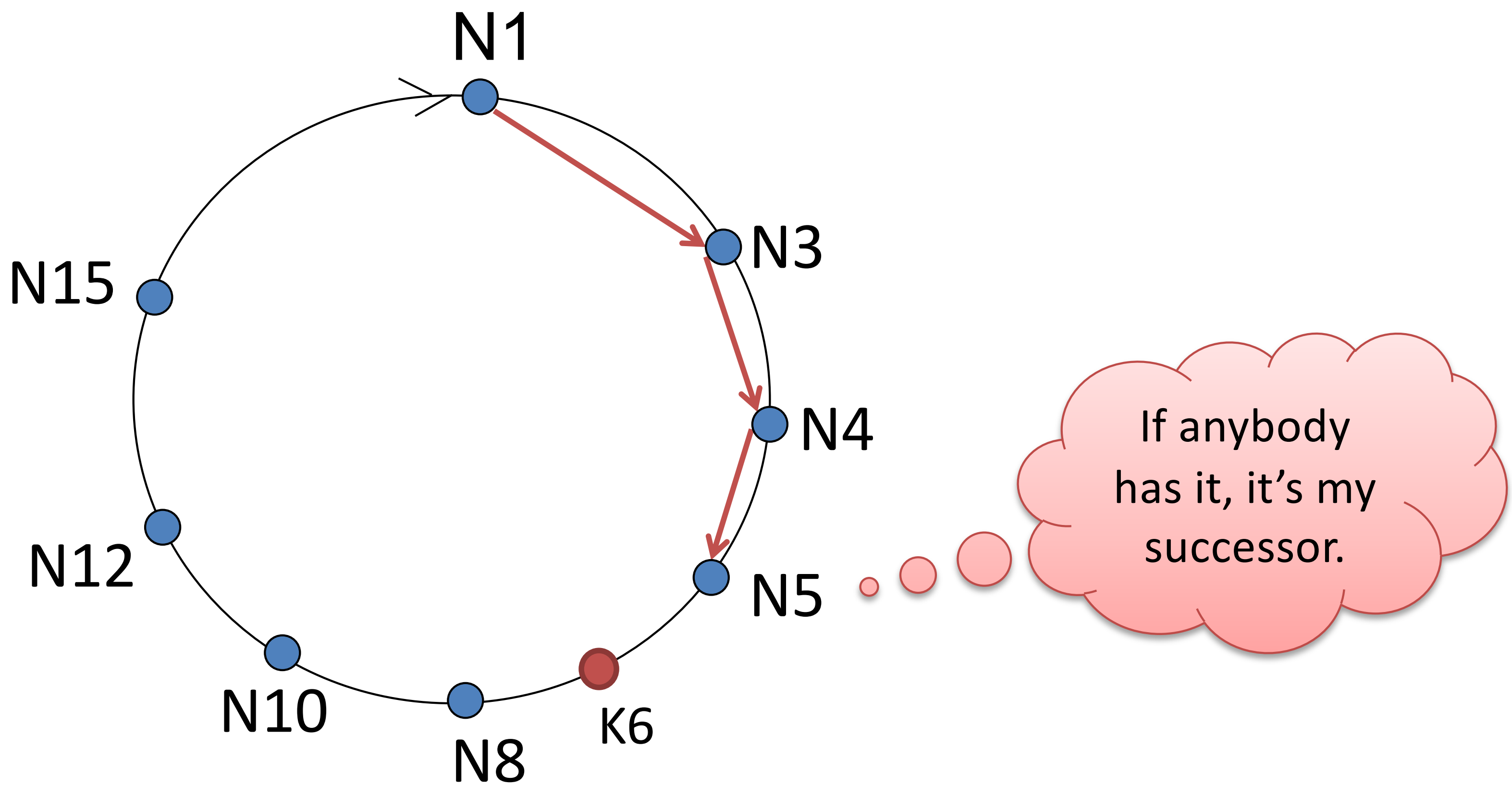
- Example: Node 1 wants key "Led Zeppelin IV"
  - Hash the key "Led Zeppelin IV" = K6

Circular DHT Overlay



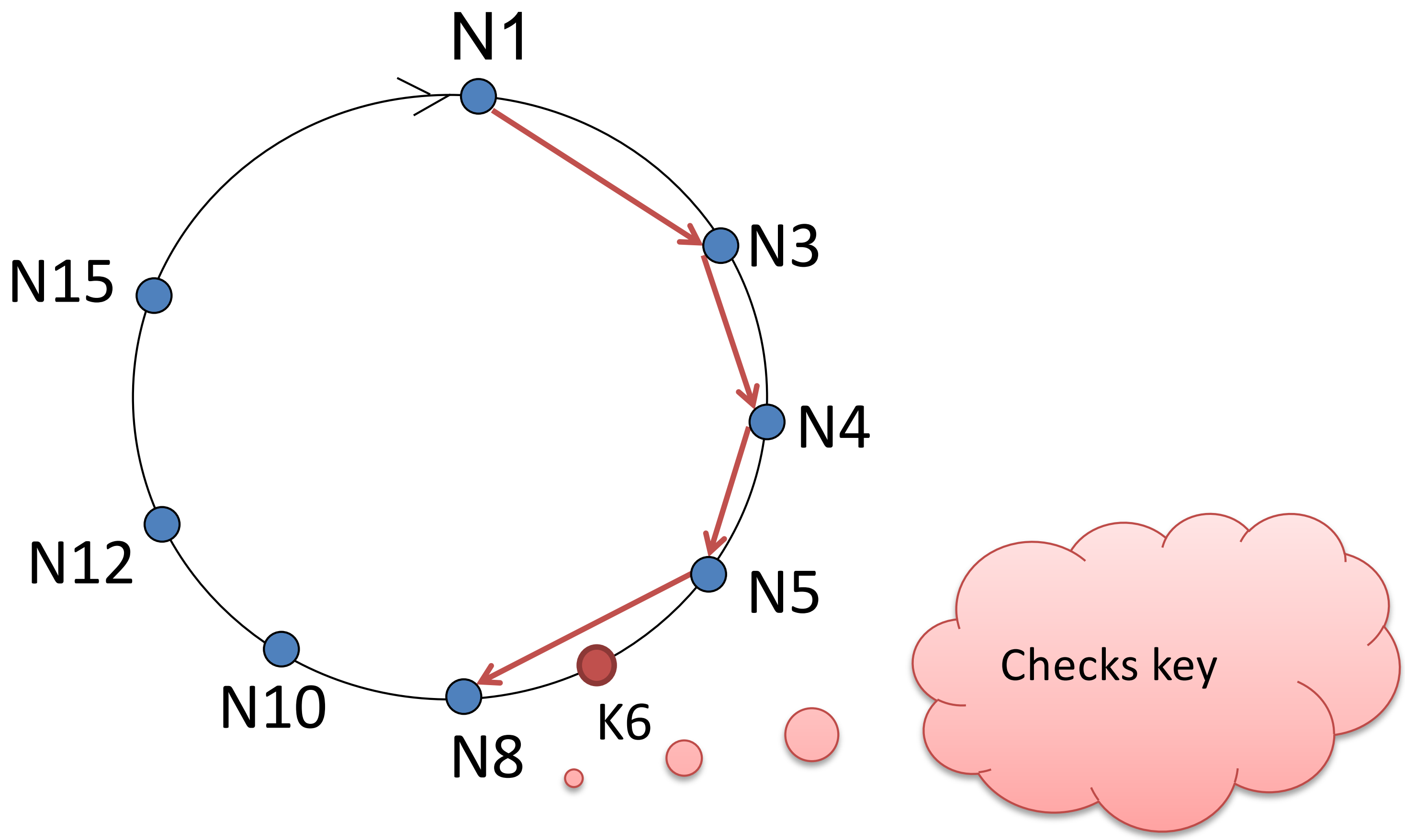
- Example: Node 1 wants key “Led Zeppelin IV”
  - Hash the key “Led Zeppelin IV” = K6

Circular DHT Overlay



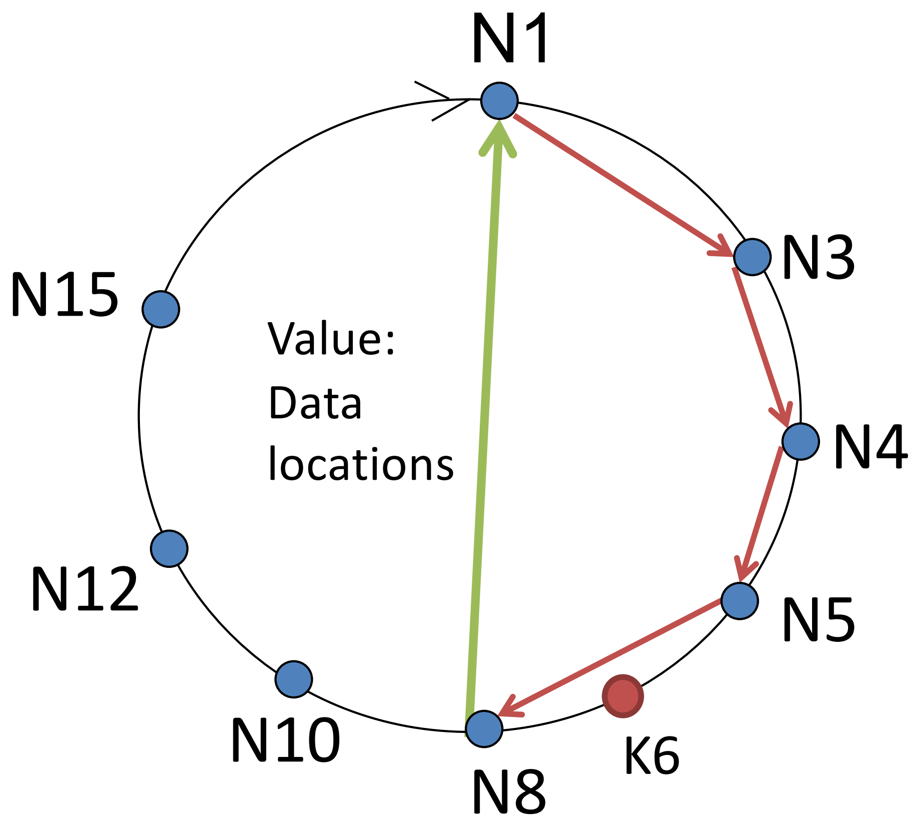
- Example: Node 1 wants key "Led Zeppelin IV"
  - Hash the key "Led Zeppelin IV" = K6

Circular DHT Overlay



- Example: Node 1 wants key "Led Zeppelin IV"
  - Hash the key "Led Zeppelin IV" = K6

Circular DHT Overlay



- Example: Node 1 wants key "Led Zeppelin IV"
  - Hash the key "Led Zeppelin IV" = K6

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

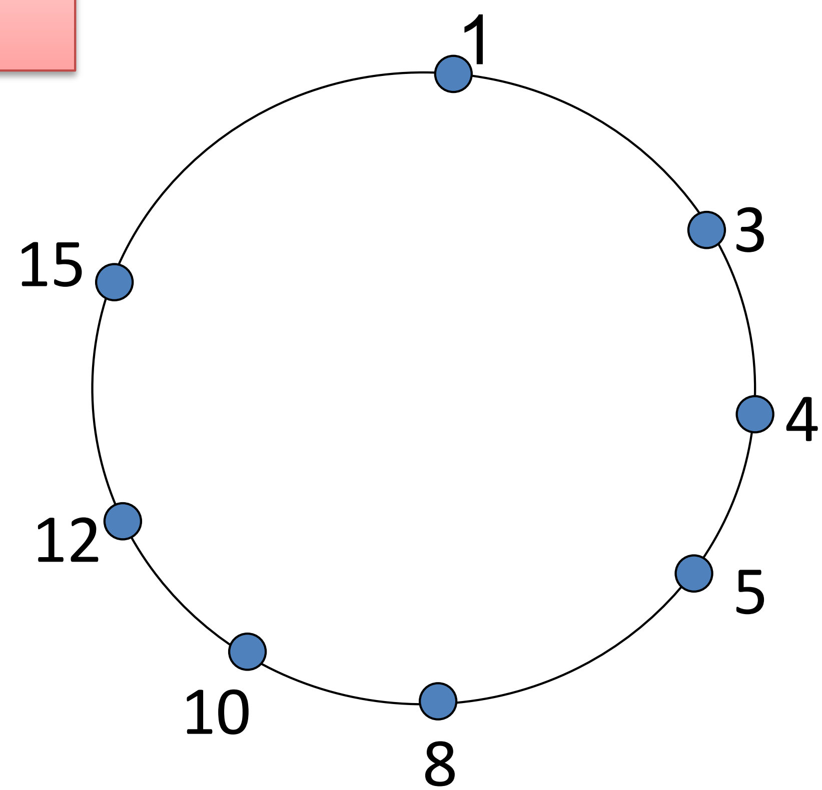
Can we do better?
How?

A. O(log n)

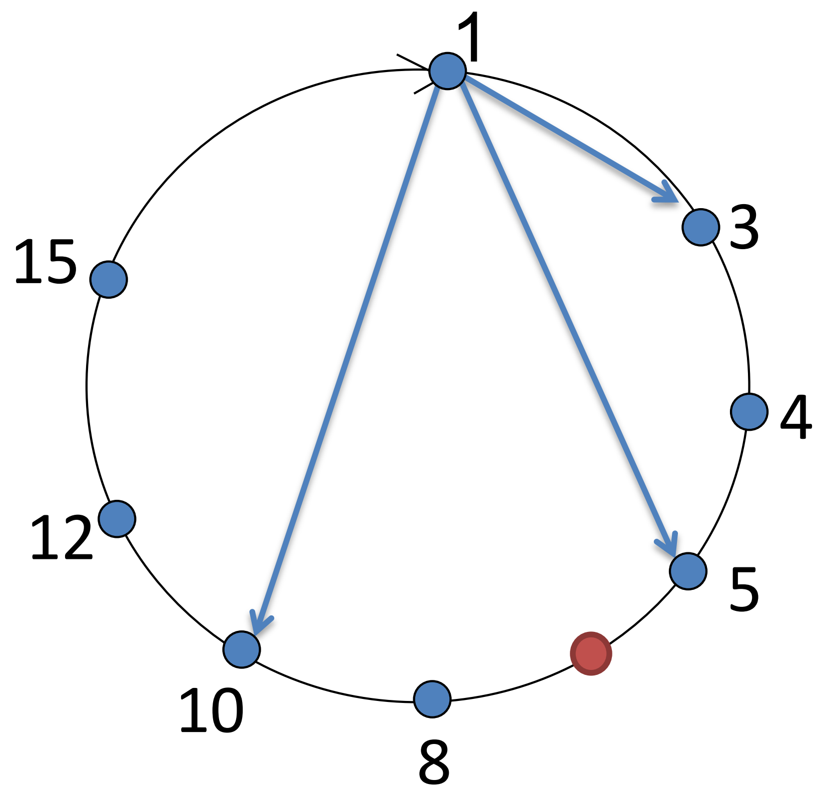
B. O(n)

C. $O(n^2)$

D. $O(2^n)$

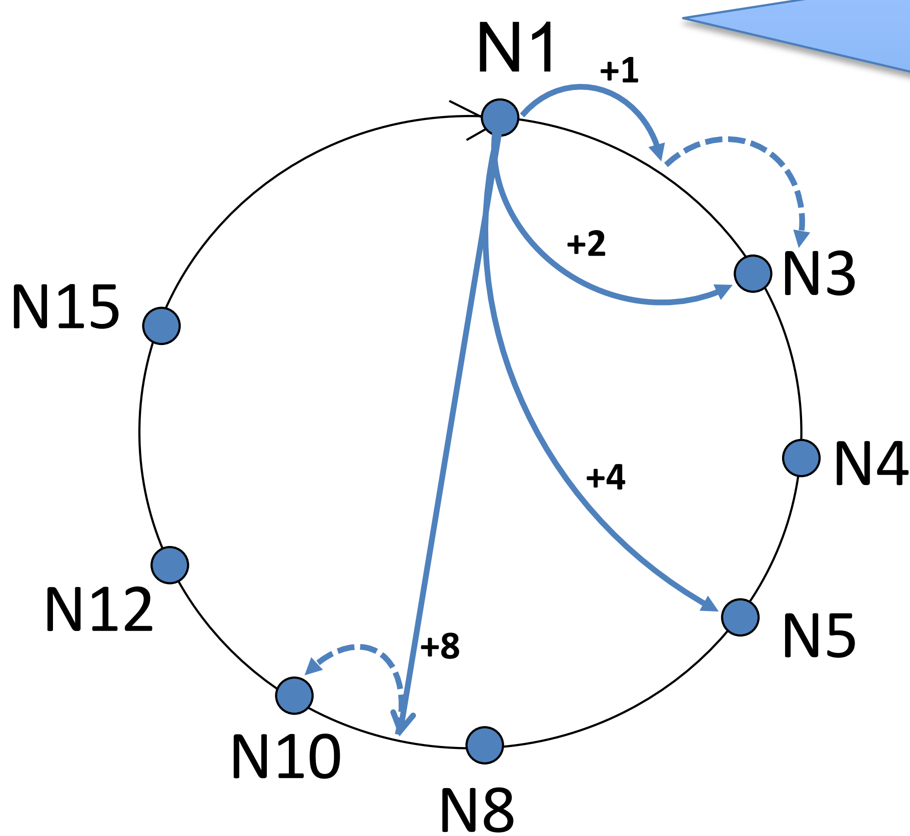


Reducing Message Count



- Store successors that are 2^0, 2^1, 2^2, 2^3, …, 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.

Each node maintains a finger table to log(N) other nodes

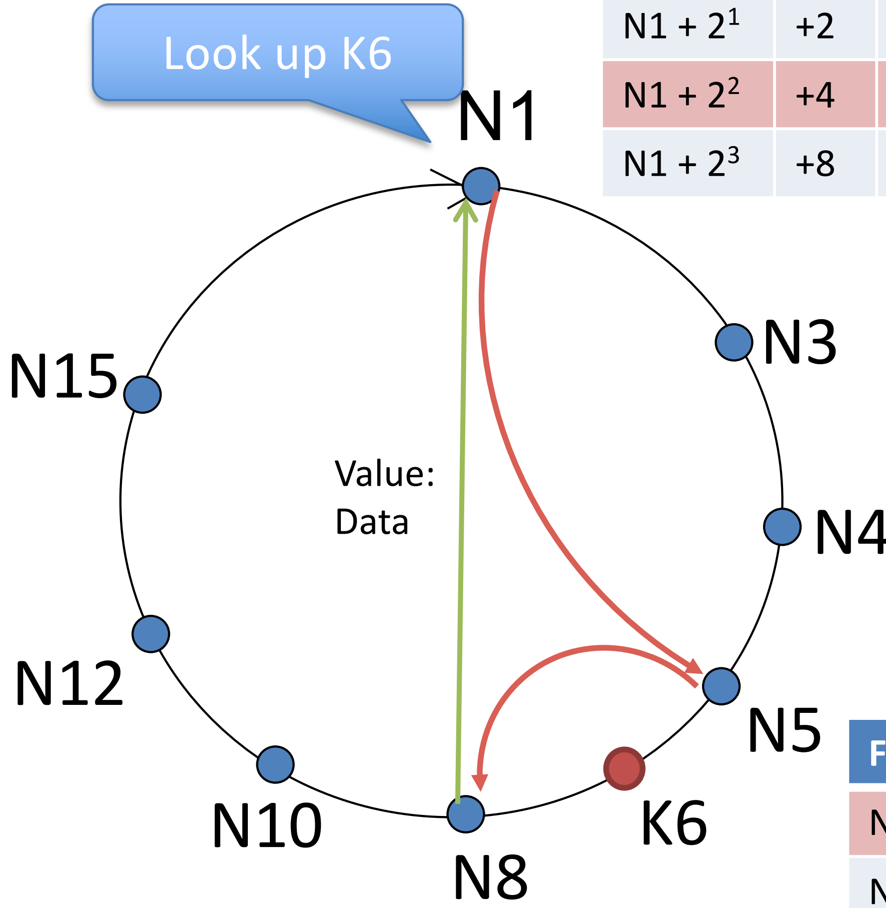


Finger Table Node 1		
N1 + 2^0	+1	N3
N1 + 2^1	+2	N3
N1 + 2^2	+4	N5
N1 + 2^3	+8	N10

Each node i in [1, n] knows of its successor and the nodes responsible for ID: ($i+2^k$) up to n/2

- $n/2 = 16/2 = 8 = 2^k \Rightarrow k = 3$
- $0 \leq k \leq 3$, in this example

Search with finger tables



Finger Table Node 1		
N1 + 2^0	+1	N3
N1 + 2^1	+2	N3
N1 + 2^2	+4	N5
N1 + 2^3	+8	N10

N5 is the closest predecessor node to K6

N8 is my immediate neighbor

Finger Table Node 5		
N5 + 2^0	+1	N8
N5 + 2^1	+2	N8
N5 + 2^2	+4	N10
N5 + 2^3	+8	N15

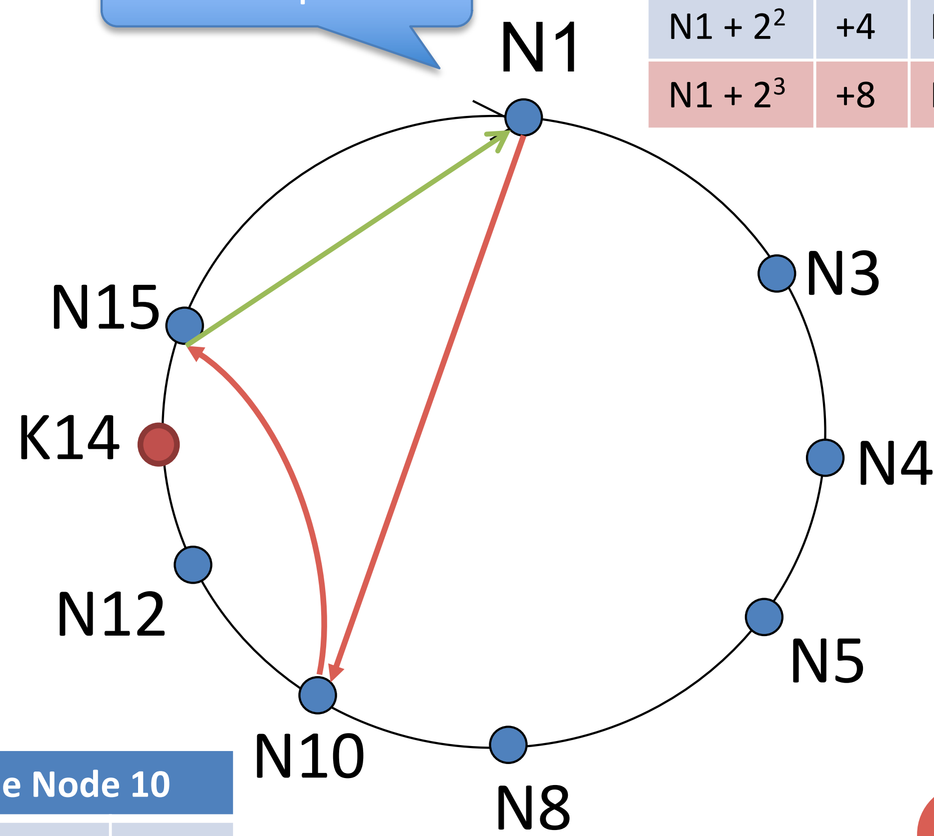
Lookup K6 from N1 = N1 -> N5 -> N8.

Search with finger tables

Look up K14

Finger Table Node 1		
N1 + 2^0	+1	N3
N1 + 2^1	+2	N3
N1 + 2^2	+4	N5
N1 + 2^3	+8	N10

N10 is the closest predecessor node to K14



Finger Table Node 10		
N10 + 2^0	+1	N12
N10 + 2^1	+2	N12
N10 + 2^2	+4	N15
N10 + 2^3	+8	N1

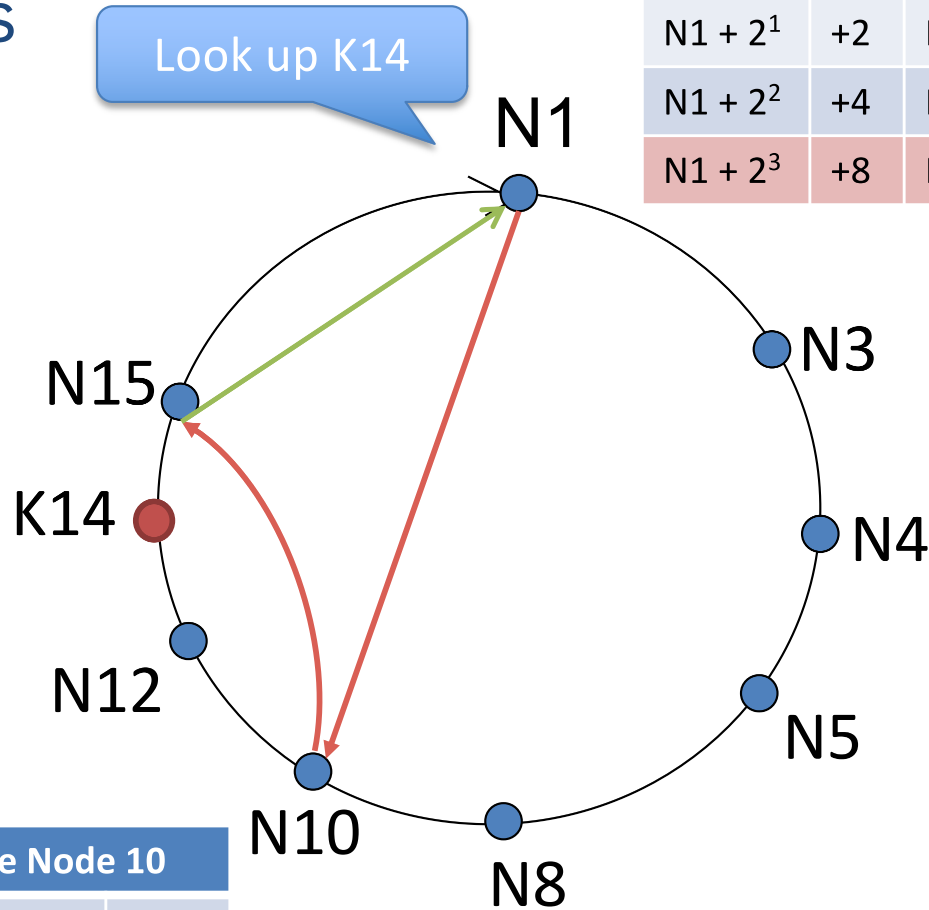
N15 is the next node that should hold K14

The key is stored at (the key's) successor: node with next highest integer

Lookup K14 from N1 = N1 -> N10 -> N15

Search with finger tables

Finger Table Node 1		
N1 + 2^0	+1	N3
N1 + 2^1	+2	N3
N1 + 2^2	+4	N5
N1 + 2^3	+8	N10



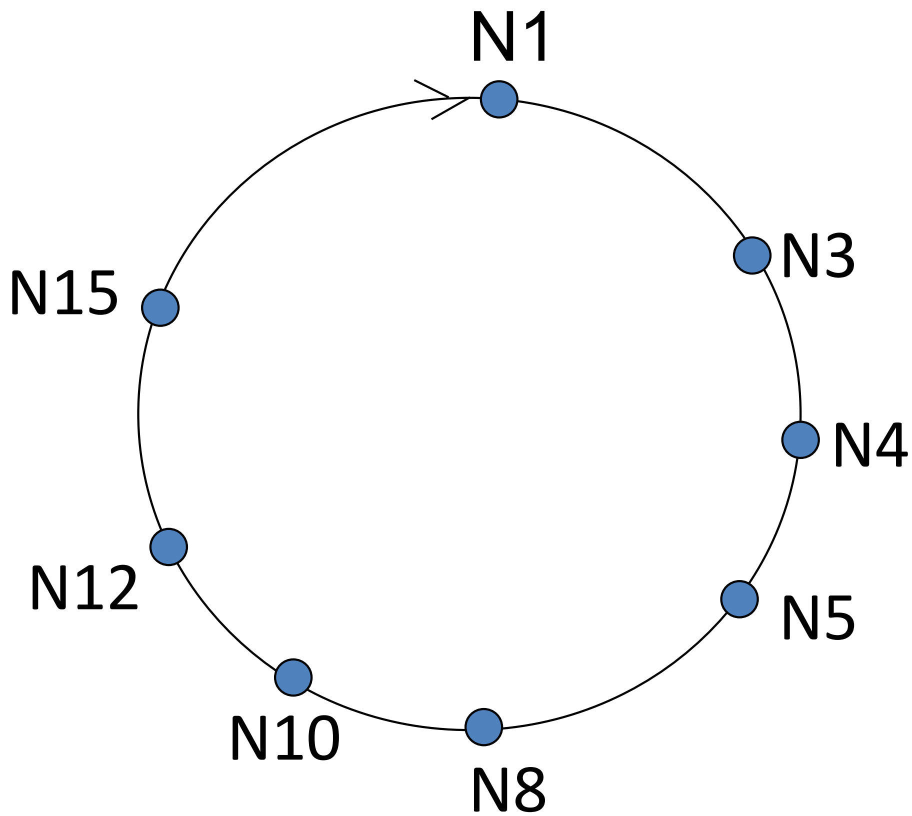
What is the size of the routing table at each node?

- n
- log n
- n^2

Finger Table Node 10		
N10 + 2^0	+1	N12
N10 + 2^1	+2	N12
N10 + 2^2	+4	N15
N10 + 2^3	+8	N1

Lookup K14 from N1 = N1 -> N10 -> N15

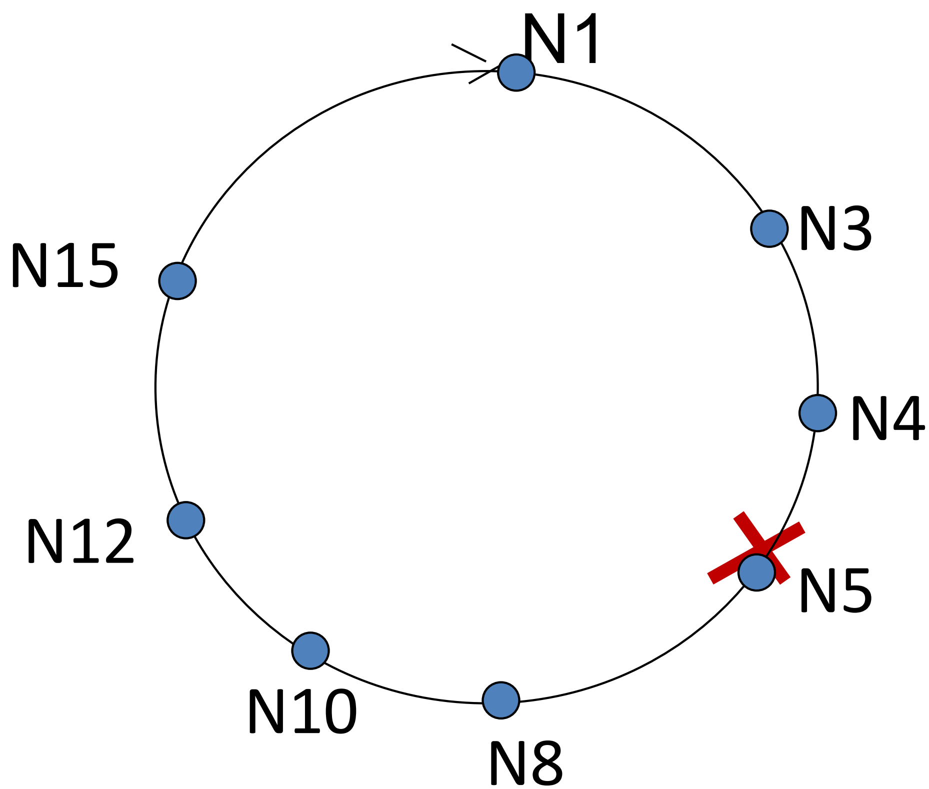
Peer/Node churn



Handling node churn:

- nodes may come and go (churn)
- each node knows address of two of its successors
- each node periodically pings its two successors to check aliveness
- if immediate successor leaves, choose next successor as new immediate successor

Peer churn

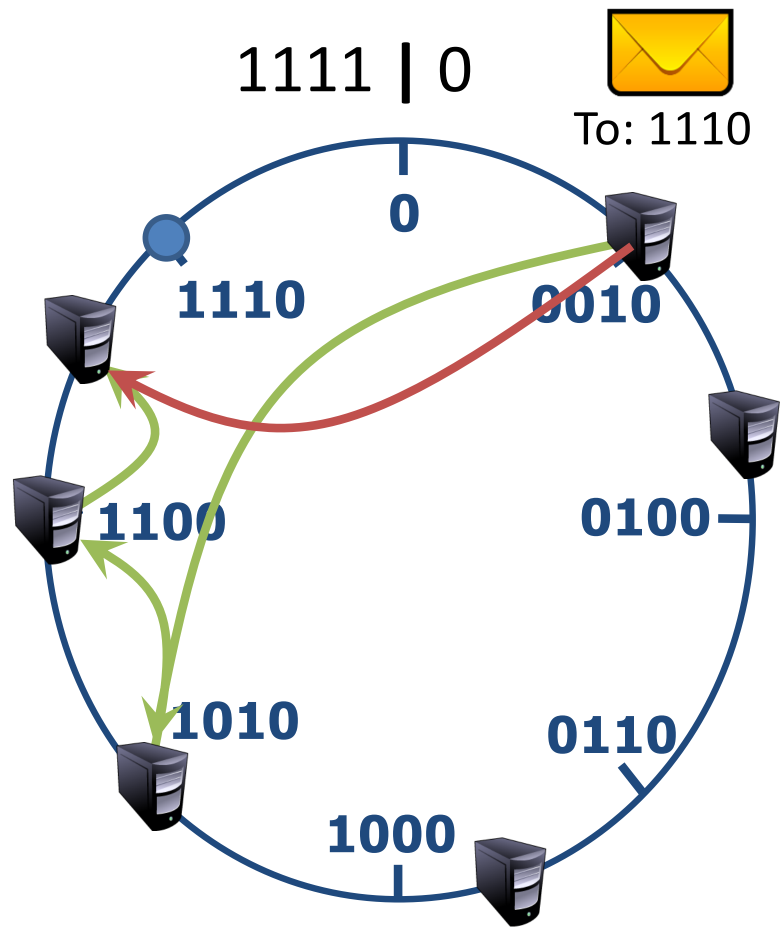


Example: node 5 abruptly leaves

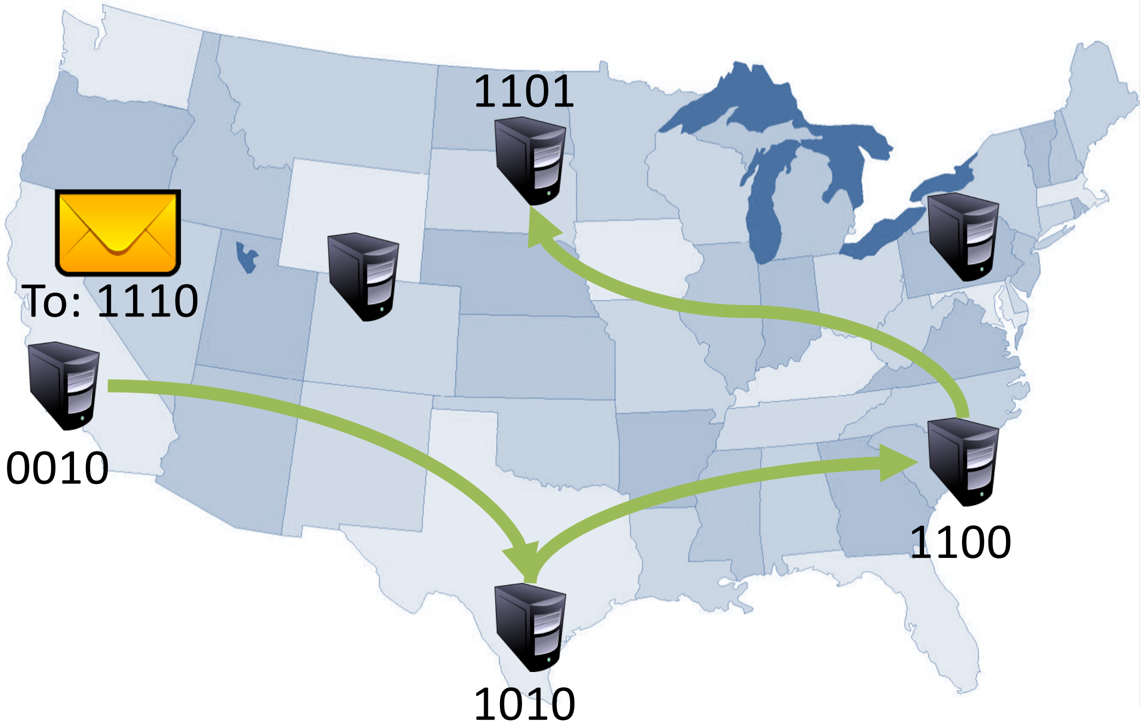
- Node 4 detects peer 5 departure;
- makes 8 its immediate successor;
- asks 8 who its immediate successor is;
- makes 8's immediate successor its second successor.

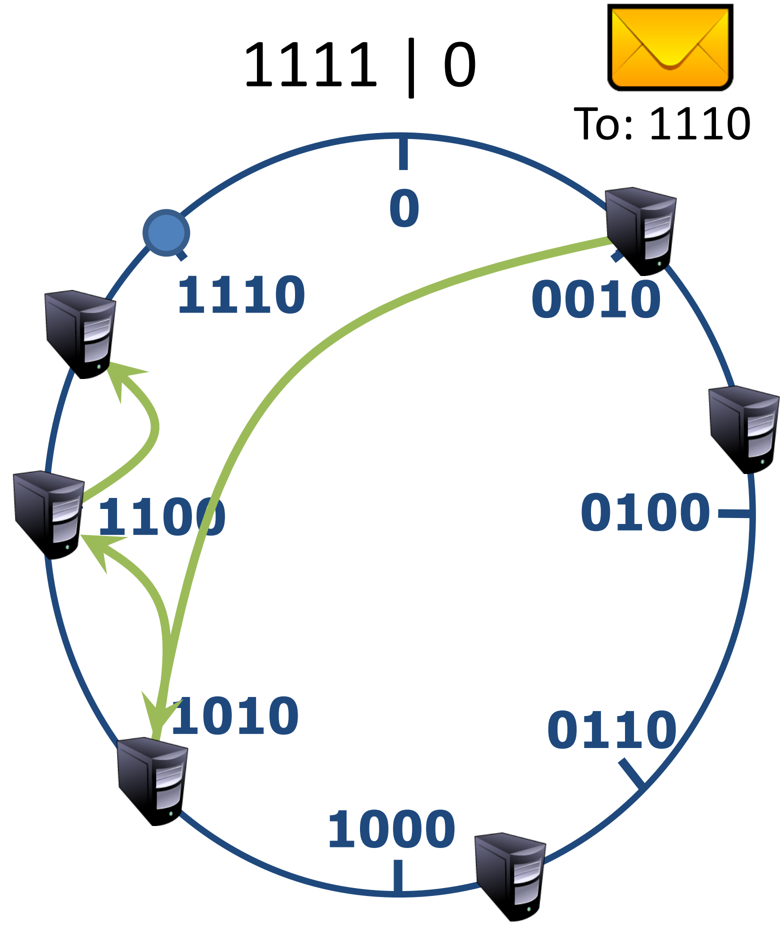
Tapestry/Pastry

- Node IDs are numbers in a ring
  - 128-bit circular ID space
- Node IDs chosen at random
- Messages for key *X* is routed to live node with longest prefix match to *X*
  - Incremental prefix routing
  - 1110: 1XXX→11XX→111X→1110



Physical and Virtual Routing





Summary of DHT Overlays

- A namespace
  - For most, this is a linear range from 0 to 2^{160}
- A mapping from key to node
  - Chord: keys between node X and its predecessor belong to X
  - Tapestry/Pastry: keys belong to node w/ closest identifier
  - 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)

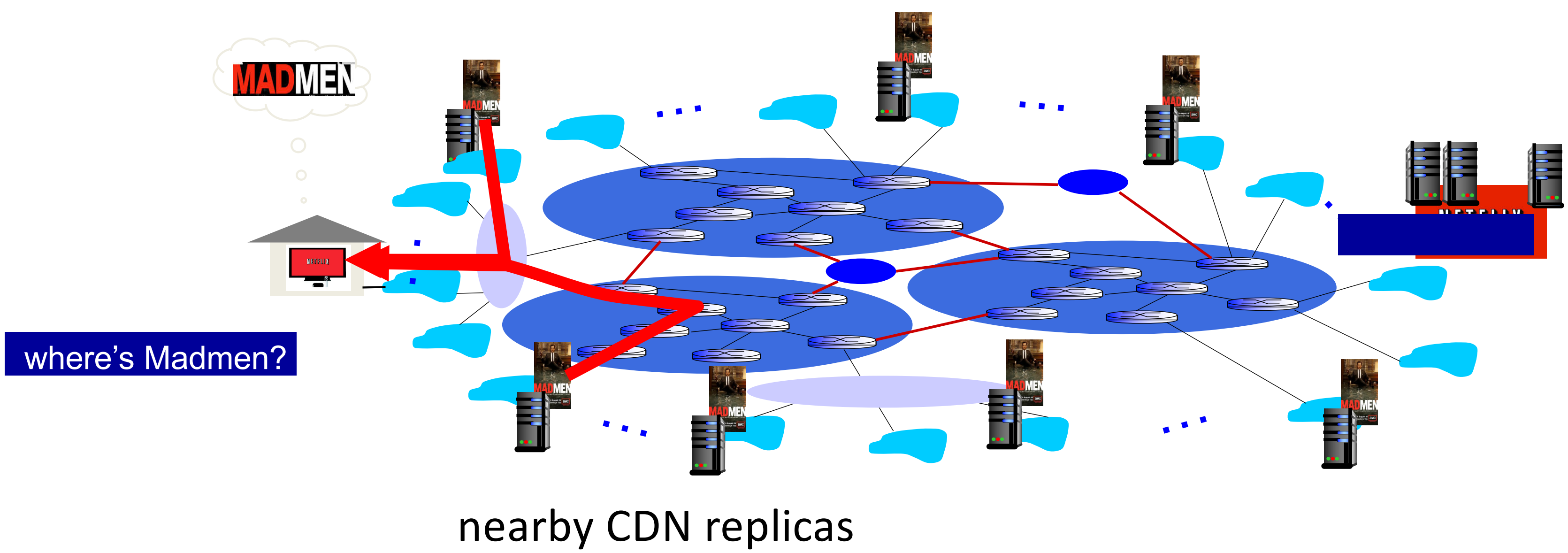
What is a Content Distribution Network?

An overlay network, that is geo-distributed and stores cached content “close” to users.

At least 70% of the world’s bits are delivered by a CDN!

Content distribution networks (CDNs)

- CDN: stores copies of content (e.g. MADMEN) at CDN nodes

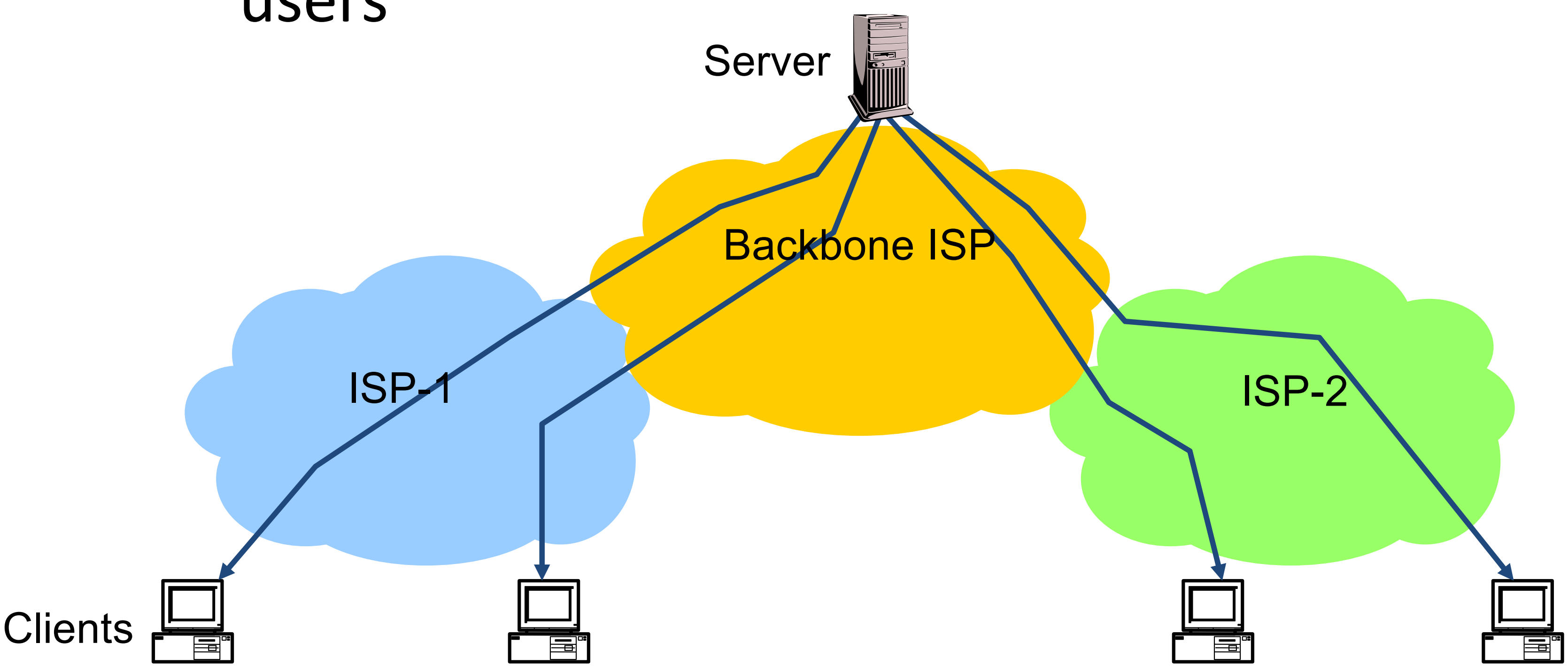


Examples of CDNs

- Akamai
  - 147K+ servers, 1200+ networks, 650+ cities, 92 countries
- Limelight
  - Well provisioned delivery centers, interconnected via a private fiber-optic connected to 700+ access networks
- Edgecast
  - 30+ PoPs, 5 continents, 2000+ direct connections
- Others
  - Google, Facebook, AWS, AT&T, Level3

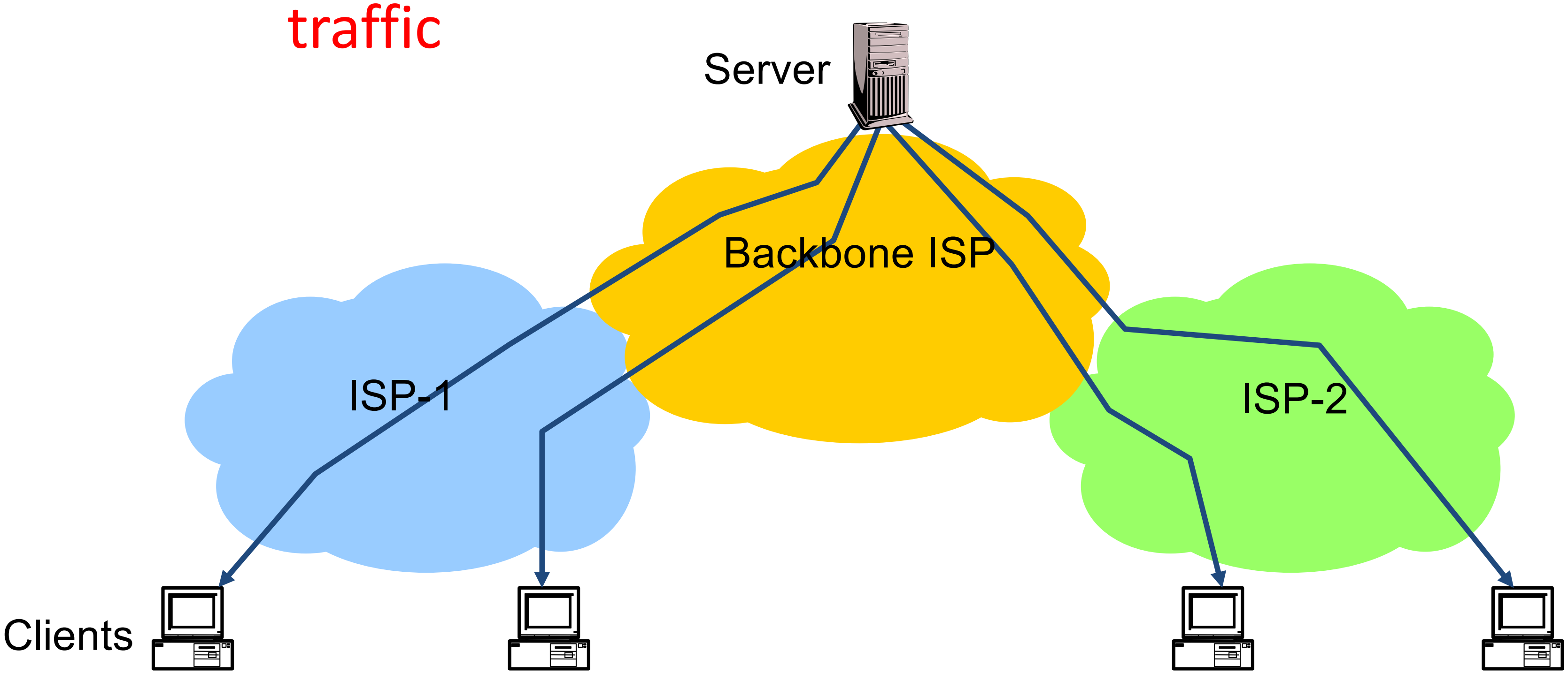
CDN caching

- Locality of reference:
  - Users tend to request the same object in succession
  - Some objects are popular: requested by many users



Where to cache content?

A. At the client (browser) – avoid extra network transfers
B. At the server (distributed server load) – reduce load
C. At the Service Providers (ISPs) – reduce external traffic



Key Components of a CDN

- Distributed servers
  - Usually located inside of other ISPs
  - Often located in IXPs
- High-speed network connecting them
- Clients (eyeballs)
  - Can be located anywhere in the world
  - They want fast web performance
- Glue
  - Something that binds clients to “nearby” replica servers

High-Performance Content Distribution

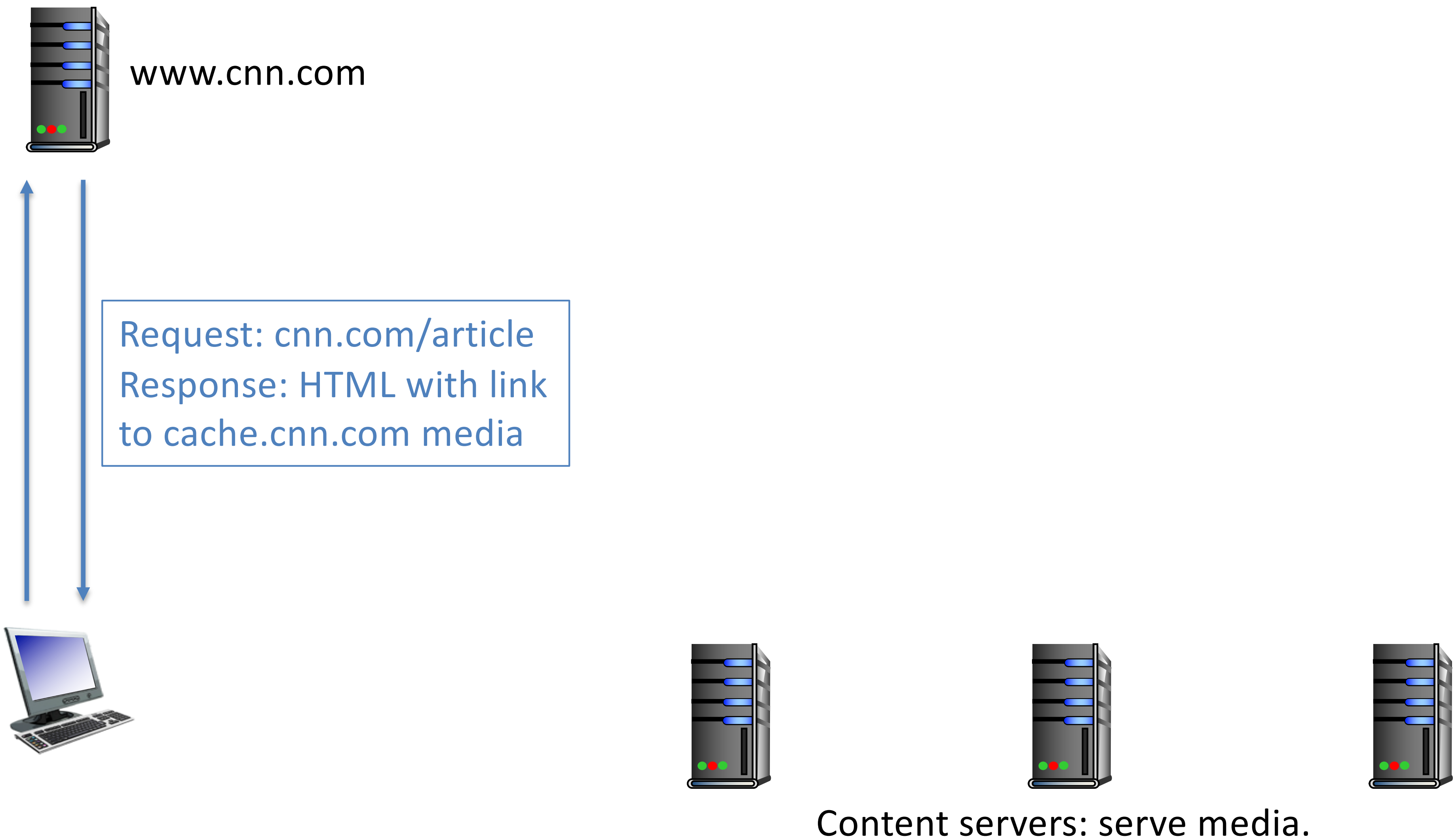
- Major challenges:
  - How do we direct the user to a nearby replica instead of the centralized source?
  - How do we determine which replica is the best to send them to?
  - Ensure that replicas are always available?

Challenge 1: Finding the CDN

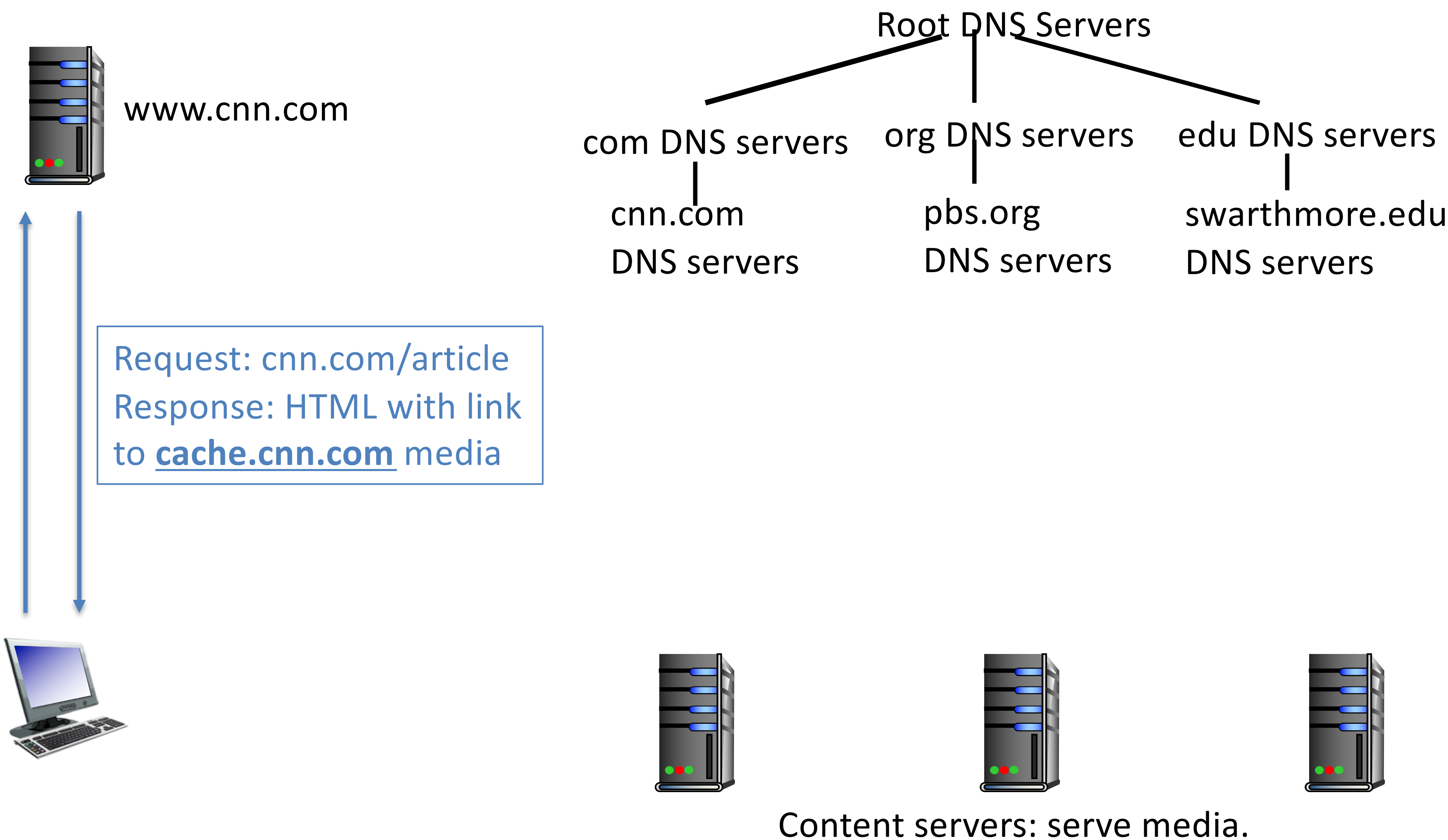
- Three main options:
  - Application redirect (e.g., HTTP)
  - "Anycast" routing
  - DNS resolution (most popular in practice)

- Example: CNN + Akamai

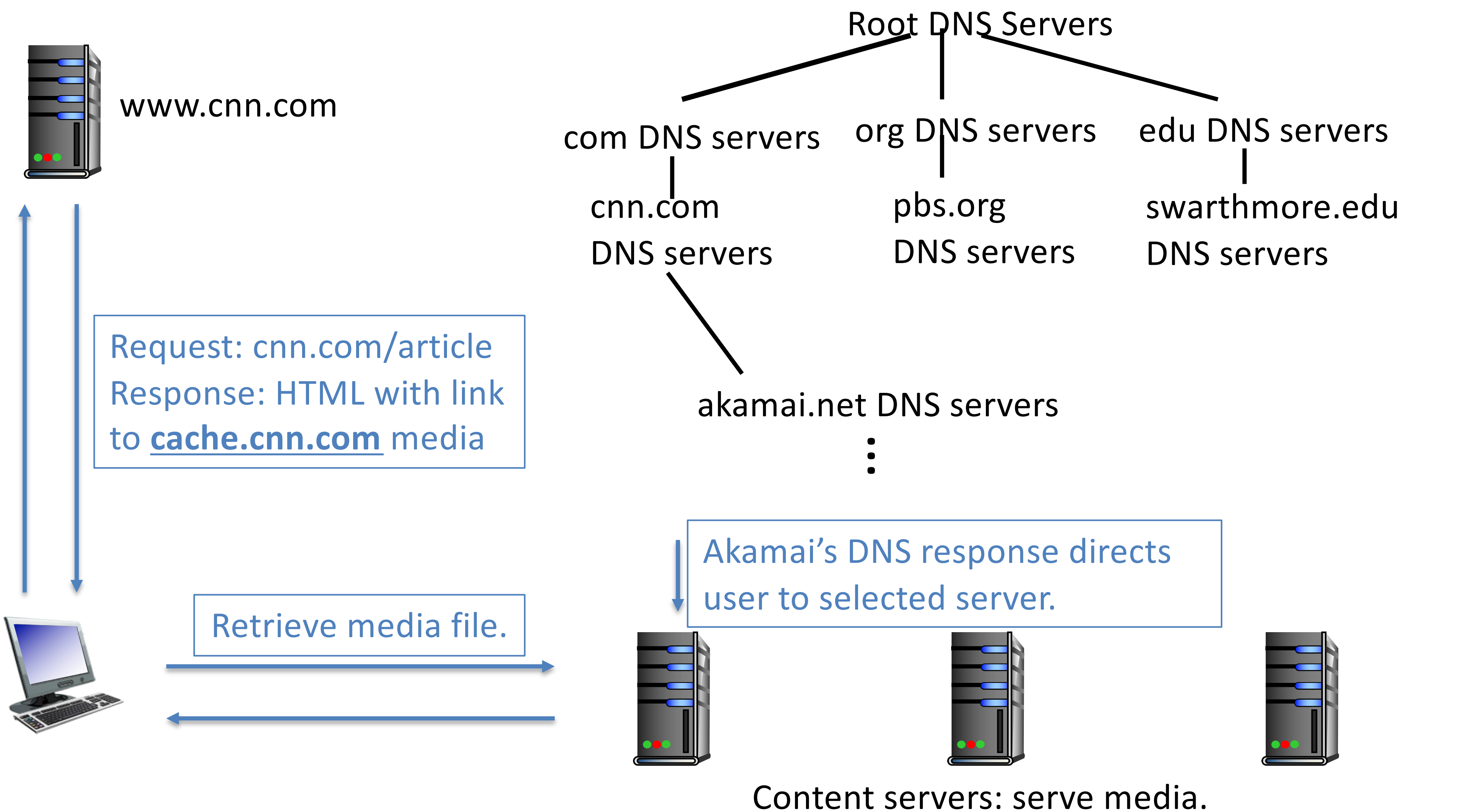
CNN + Akamai


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

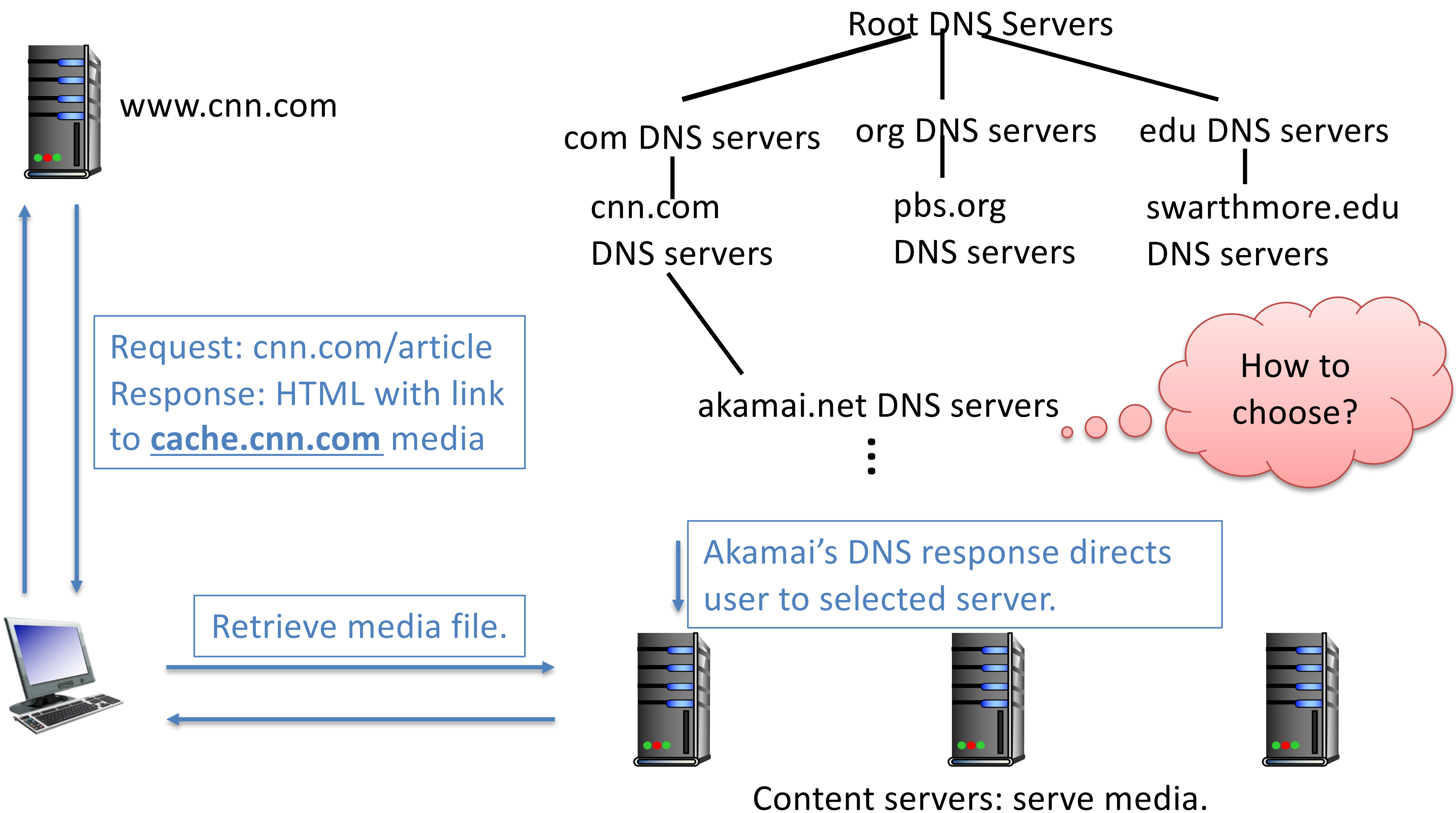
CNN + Akamai


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

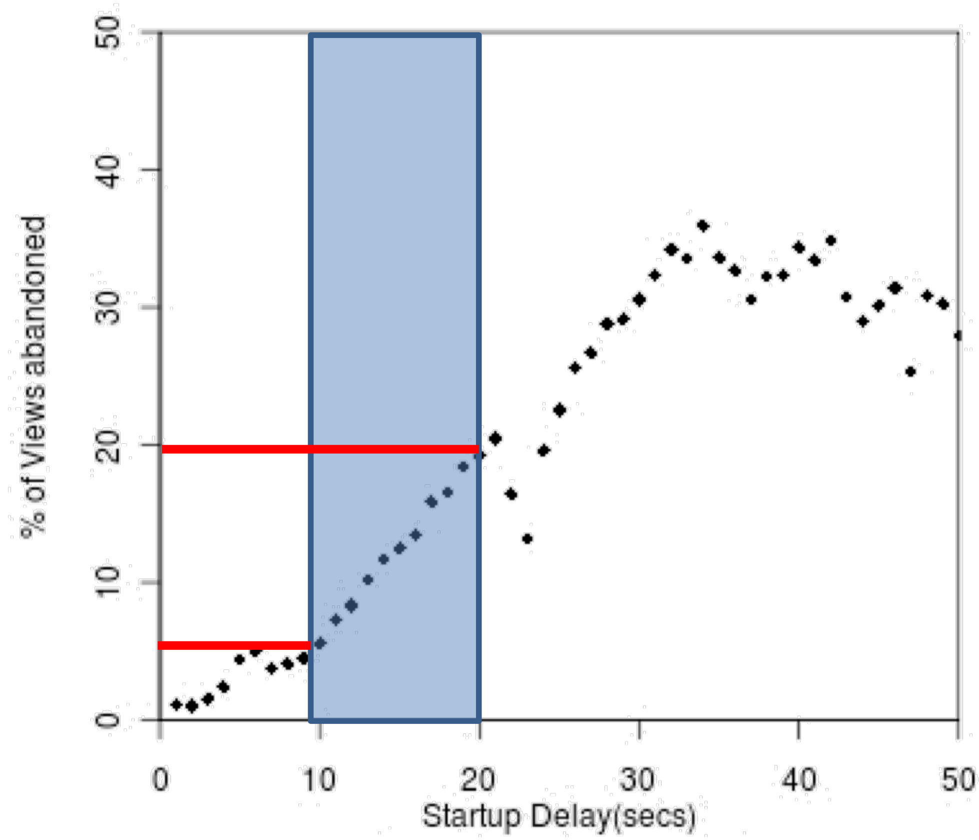
Content in today's Internet

- Most flows are HTTP
  - Web is at least 52% of traffic
  - Median object size is 2.7K, average is 85K (as of 2007)
- Is the Internet designed for this common case?
  - Why?

Why speed matters

- Impact on user experience
  - Users navigating away from pages
  - Video startup delay

- 4x increase in abandonment with 10s increase in delay



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