Name and object lookup

CSE 561, Winter 2021

Ratul Mahajan

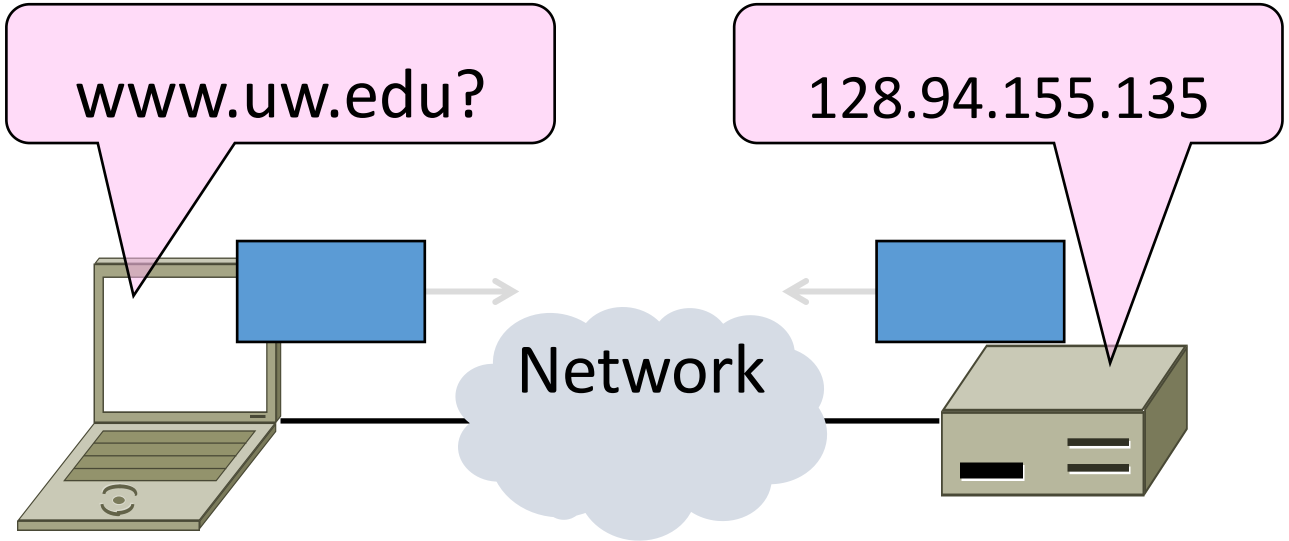
What we read

Two completely different approaches to looking up things

1. DNS
2. DHTs (Chord)

DNS

Maps human readable names to IP addresses (and more)



DNS

Goals

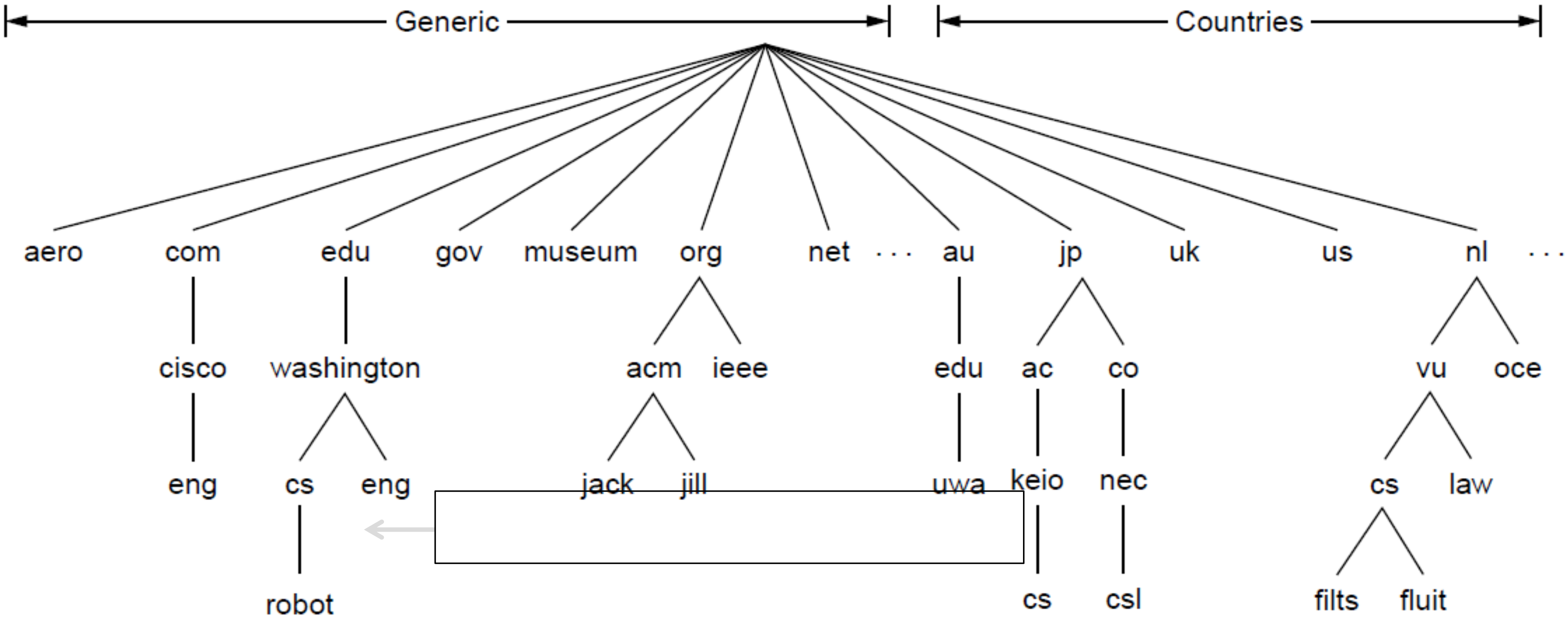
- Easy to manage with multiple parties
- Efficient (good performance, few resources)

Approach

- Distributed directory, hierarchical namespace
- Automated protocol to tie pieces together

DNS Namespace

Hierarchical, starting from "." (dot, typically omitted)



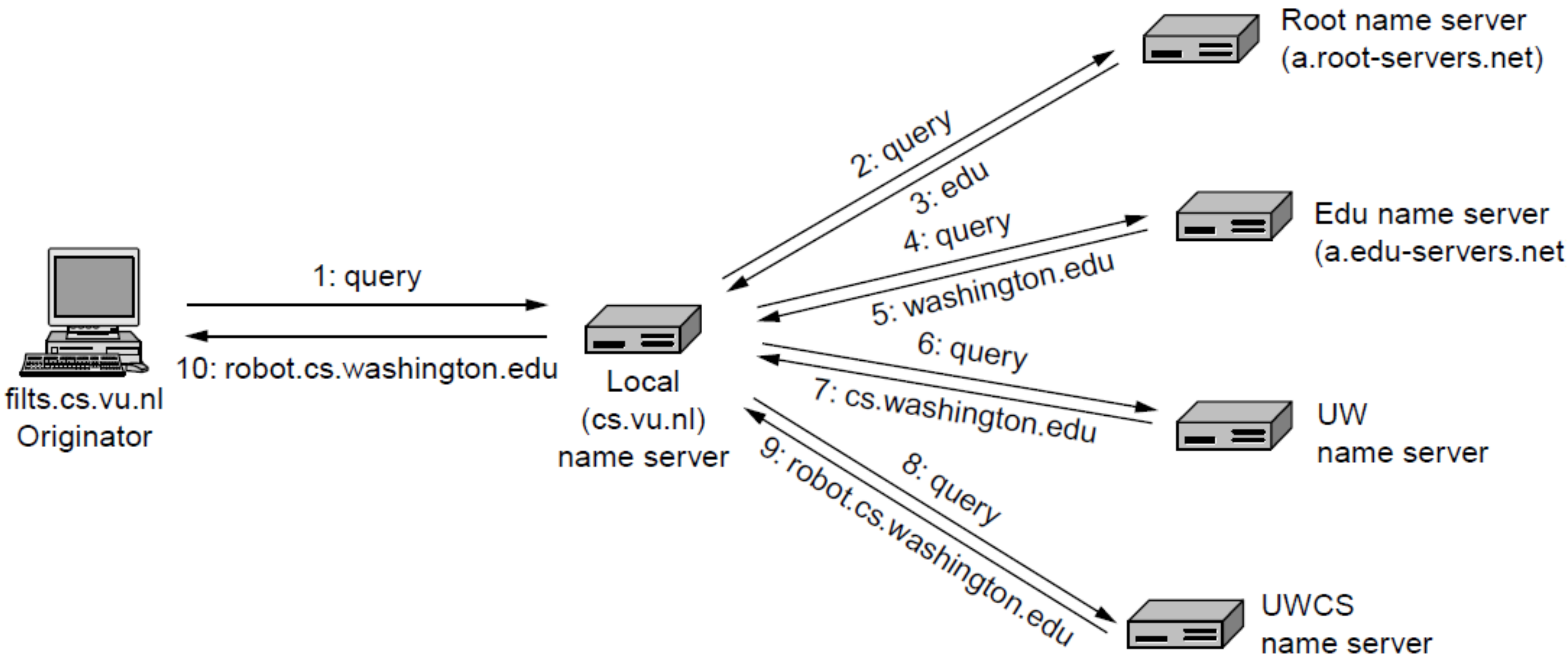
DNS Resolution

DNS protocol lets a host resolve any host name (domain) to IP address

If unknown, can start with the root nameserver and work down zones

DNS Resolution (2)

- flits.cs.vu.nl resolves robot.cs.washington.edu



Iterative vs. Recursive Queries

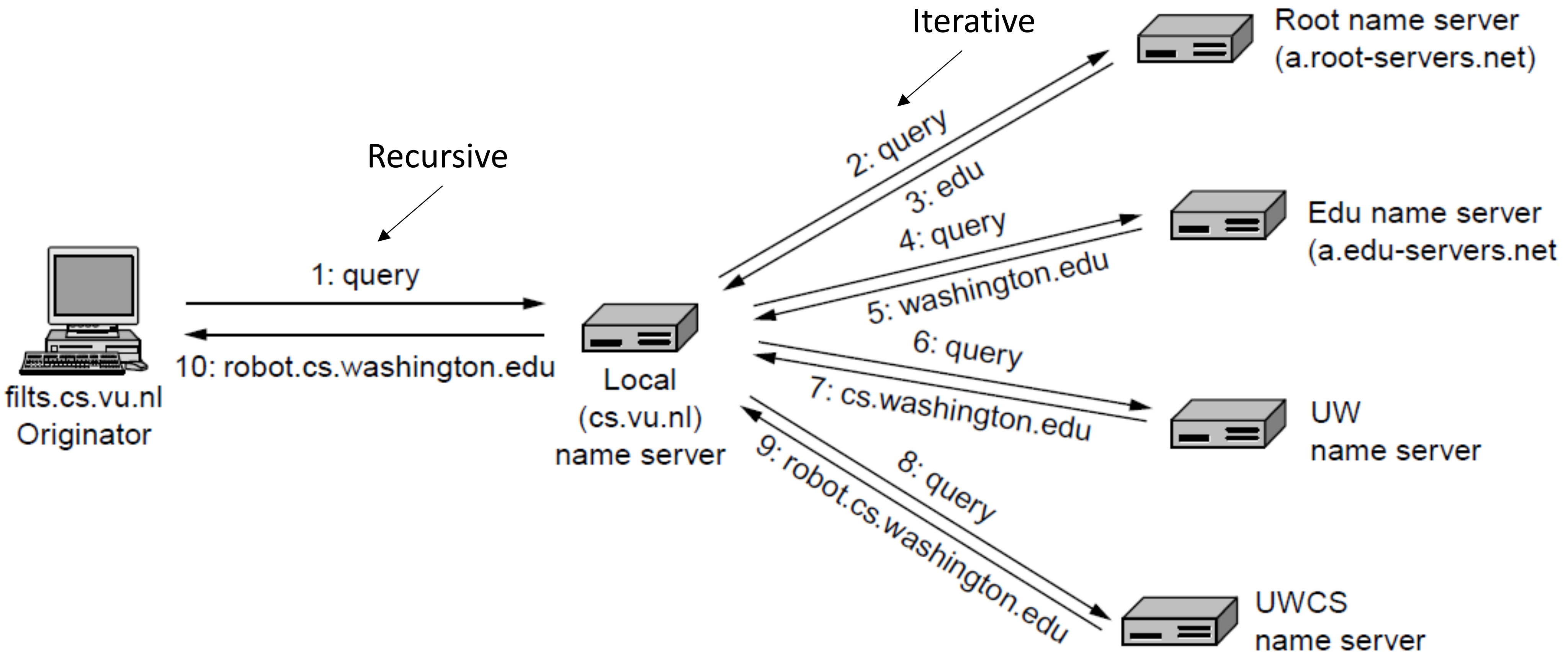
Recursive query

- Nameserver resolves and returns final answer
- E.g., flits → local nameserver

Iterative (Authoritative) query

- Nameserver returns answer or who to contact for answer
- E.g., local nameserver → all others

Iterative vs. Recursive Queries (2)



Iterative vs. Recursive Queries (3)

Recursive query

- Servers can offload client burden
- Servers can cache results for a pool of clients

Iterative query

- Server can “file and forget”
- Easy to build high load servers

Root Nameservers

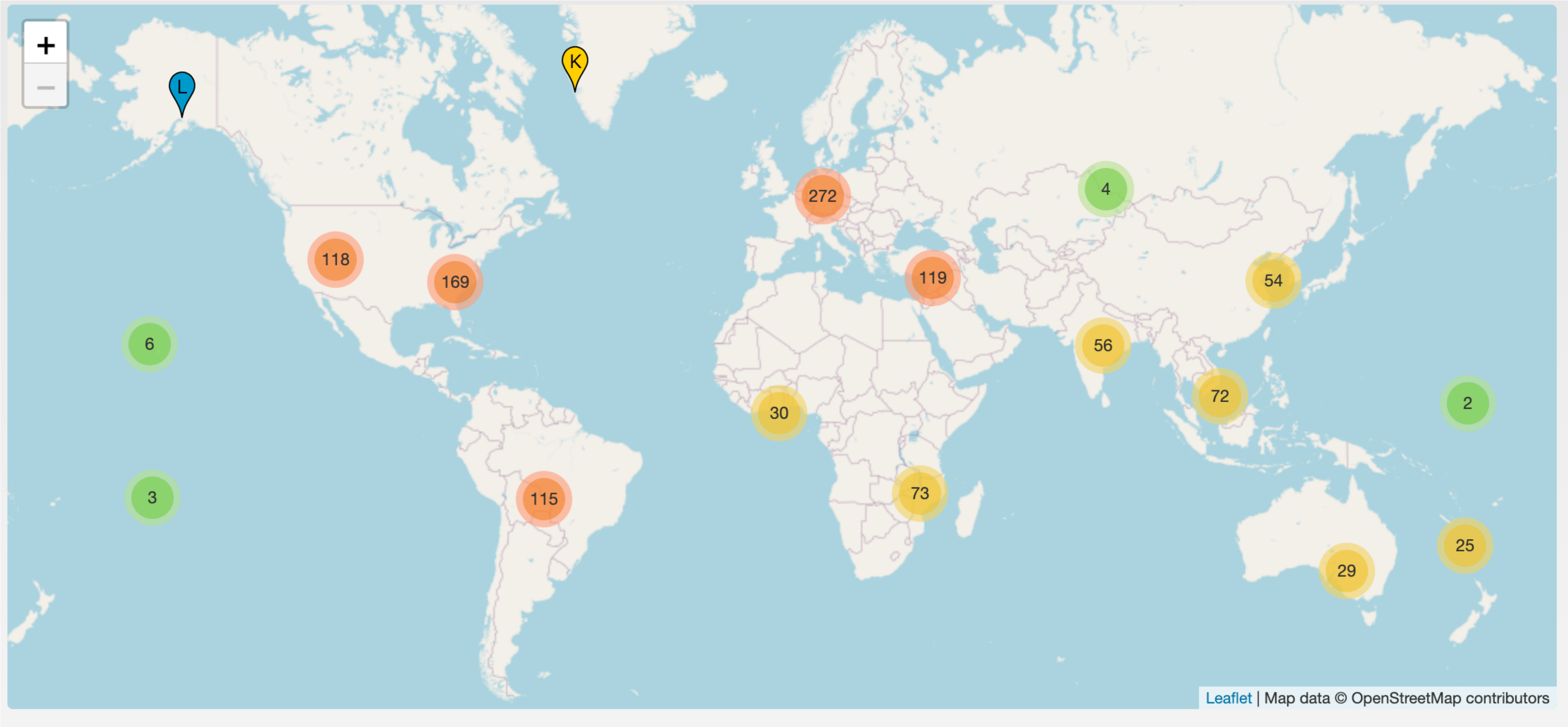
Root (dot) is served by 13 server names

- a.root-servers.net to m.root-servers.net
- All nameservers need root IP addresses
- Handled via configuration file (named.ca)

There are >250 distributed server instances

- Highly reachable, reliable service
- Most servers are reached by IP anycast
  - Multiple locations advertise same IP! Client go to the closest one.

Root Server Deployment [root-servers.org]



Local Nameservers

Often run by IT (enterprise, ISP)

- But may be your host or AP
- Or alternatives e.g., Google public DNS (8.8.8.8) Cloudflare’s public DNS (1.1.1.1)

Clients need to be able to contact local nameservers

- Configured via DHCP or statically

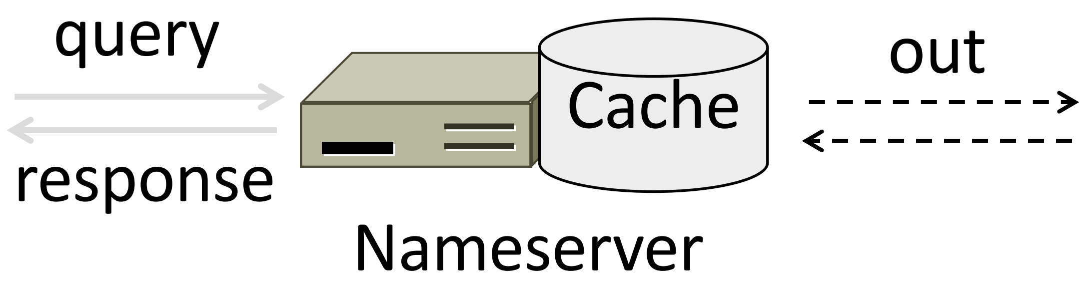
Caching

Resolution latency needs to be low

- Can take a while to trace from . (dot)

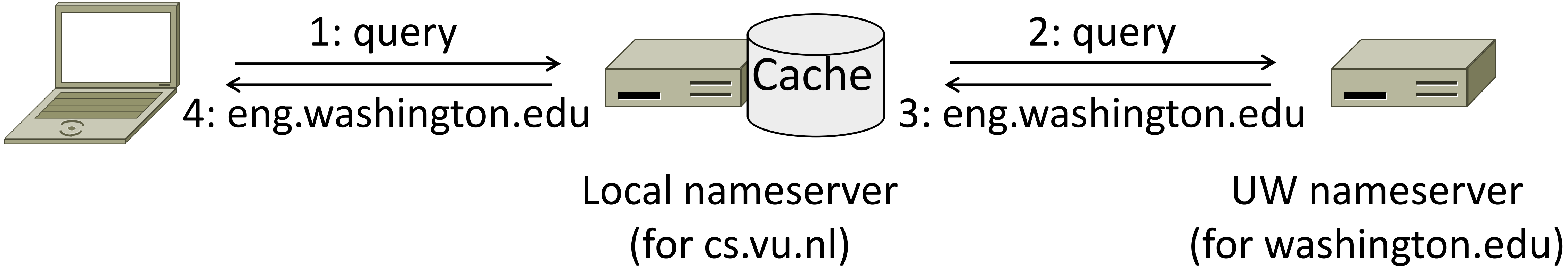
Cache query/responses to answer future queries

- Including partial (iterative) answers
- Responses carry a TTL for caching



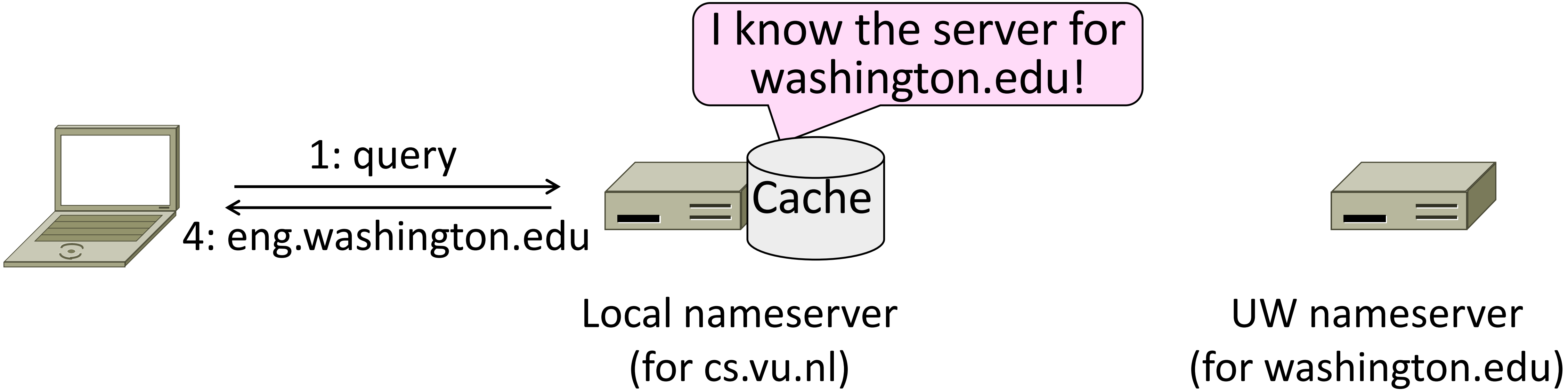
Caching (2)

flits.cs.vu.nl looks up and stores eng.washington.edu



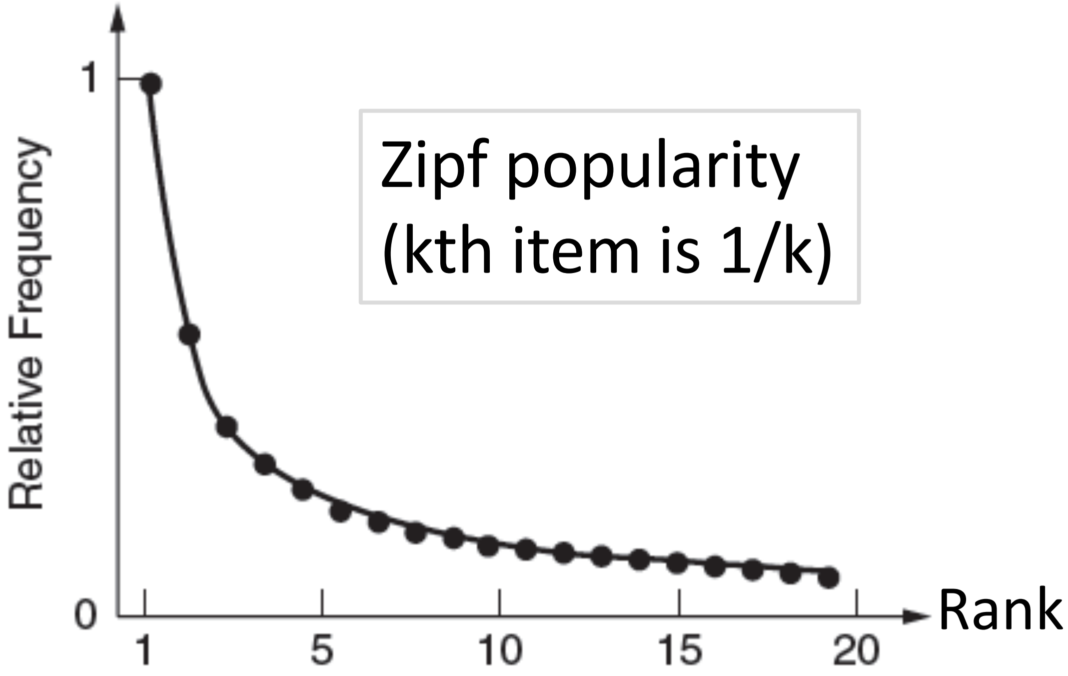
Caching (3)

flits.cs.vu.nl now directly resolves eng.washington.edu



Why caching works: Zipf's law

Few popular items, many unpopular ones



George Zipf
(1902-1950)

Source: Wikipedia

DNS answers need not be fixed

Give different answers to different clients and at different times

- Based on (an estimate of) client location
- Based on Web server load

Forms the basis of CDNs

- Direct clients to the nearest lightly-loaded server

Caching interferes with dynamic answers – use low TTL

Methods for distributed lookups

Hierarchical directories (e.g., DNS)

- Efficient but vulnerable to failures and attacks

Flooding

- Robust but not scalable

Distributed hash tables

- Robust and scalable but less efficient than hierarchical directories

History of DHTs

(Illegal) file sharing started it all

- Napster was a directory-based system (easy to takedown)
- Gnutella was flooding-based
- Popularity of Gnutella but its simplistic design inspired many researchers
  - Chord, CAN, Pastry, Tapestry were submitted to SIGCOMM the same year

File sharing turned out to be a fad but the core technology has become an important substrate for many distributed applications

What is a DHT?

Classic hash table

- Put(key, value)
- Get(key) → value

DHTs offer the same interface to applications but under the hood

- Lookup(key) → Address of node that owns the key
- Put(key, value) := Put(Lookup(key), key, value)
- Get(key) := Get(Lookup(key), key)

DHT overview

Goal: Implement lookup over possibly **millions** of **unreliable** nodes

- Global information is almost impossible
  - State maintained should grow slowly with the number of nodes
- Nodes can come and go (churn)
  - No node should be critical to the service

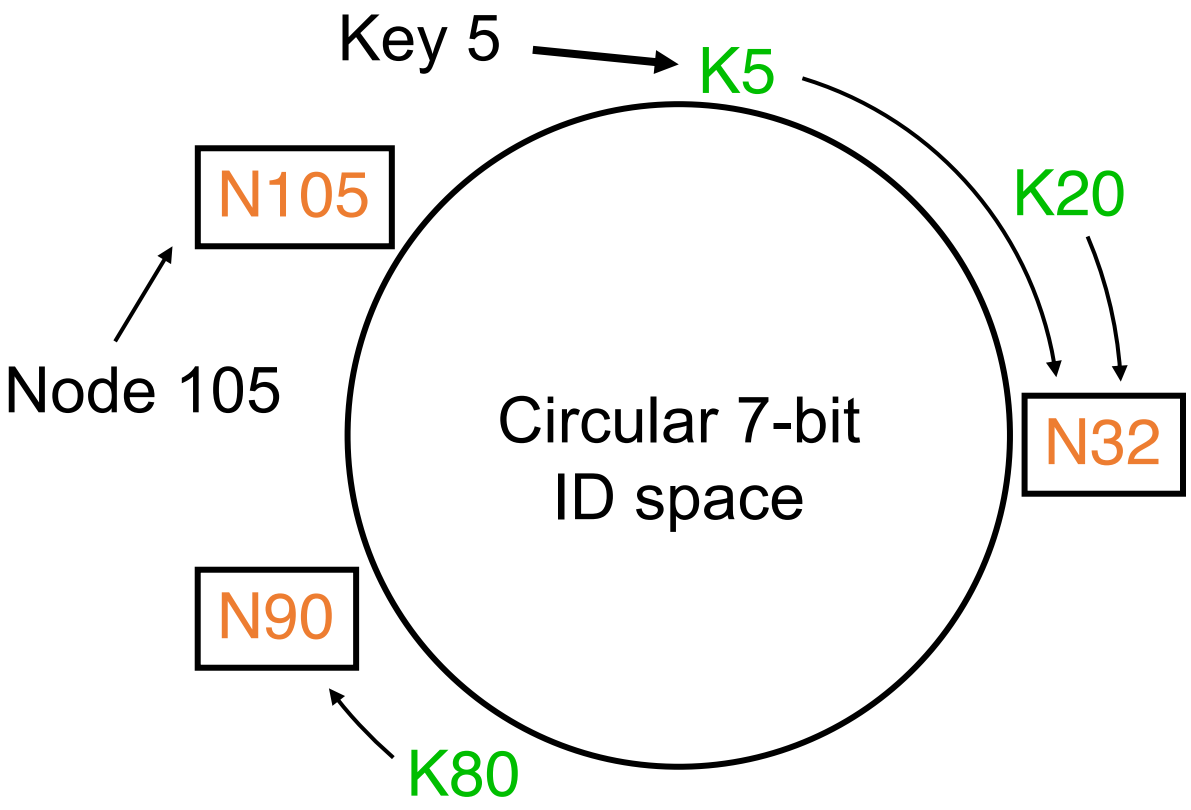
Approach: Different DHTs differ in details but there is a theme

- Map nodes to key space of objects
- Nodes own keys in the neighborhood
- Maintain pointers to other nodes to help route queries

Chord

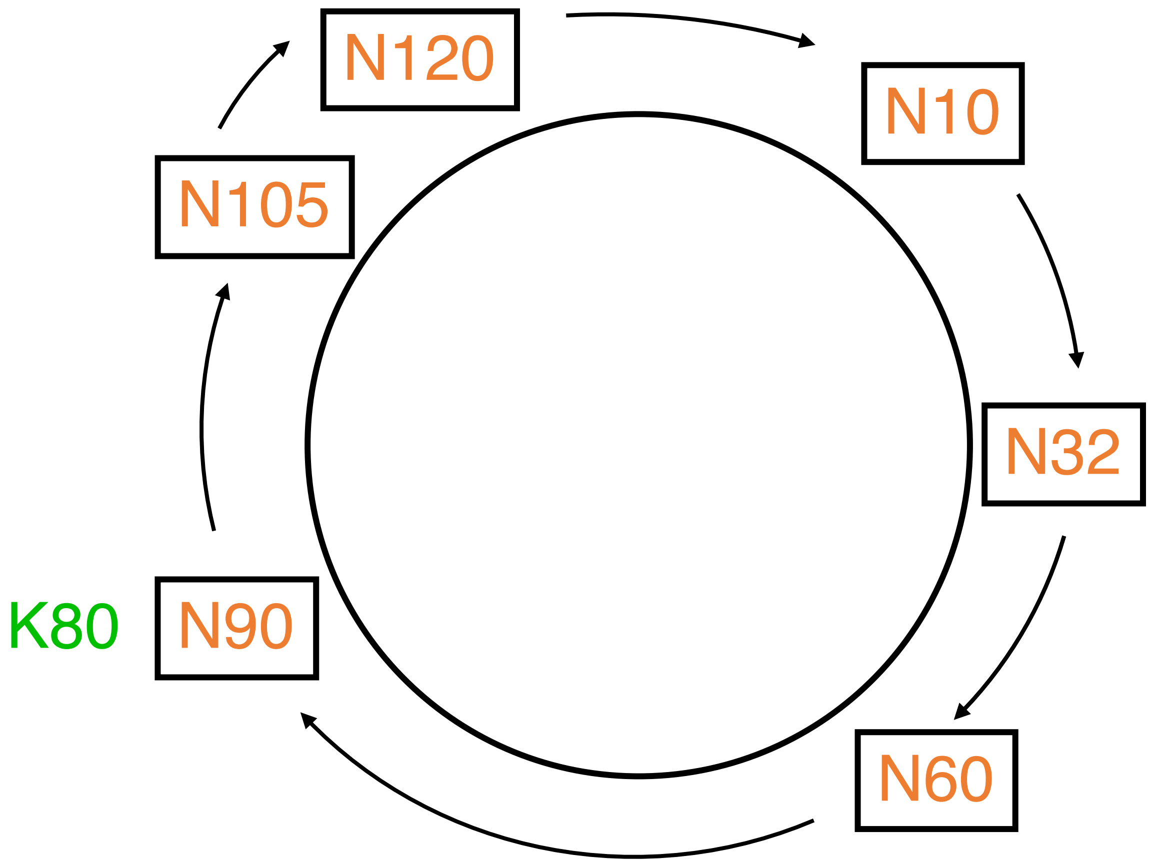
[Some slides from Kyle Jamieson]

Consistent hashing [Karger '97]

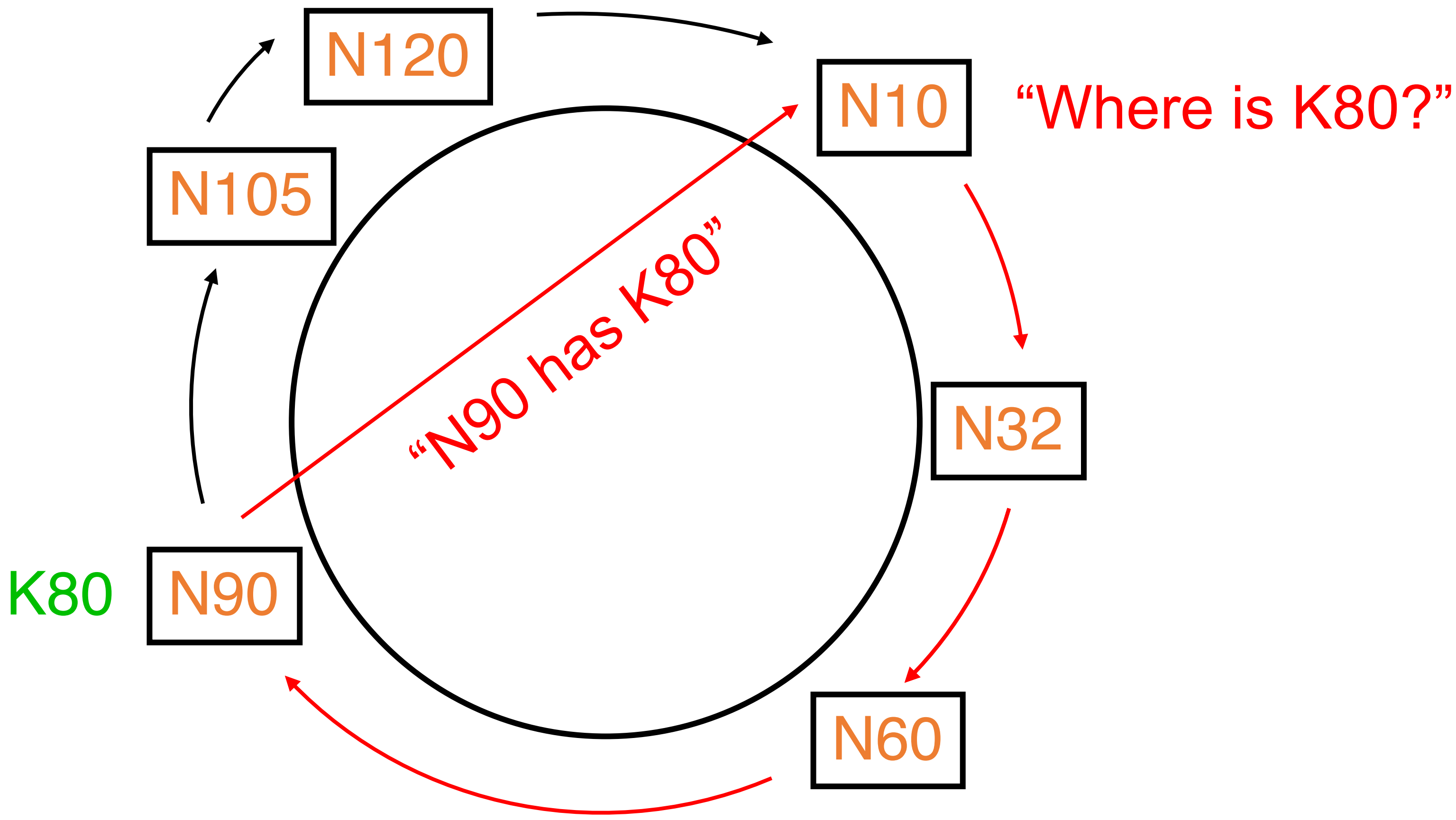


Key is stored at its successor: node with next-higher ID

Chord: Successor pointers



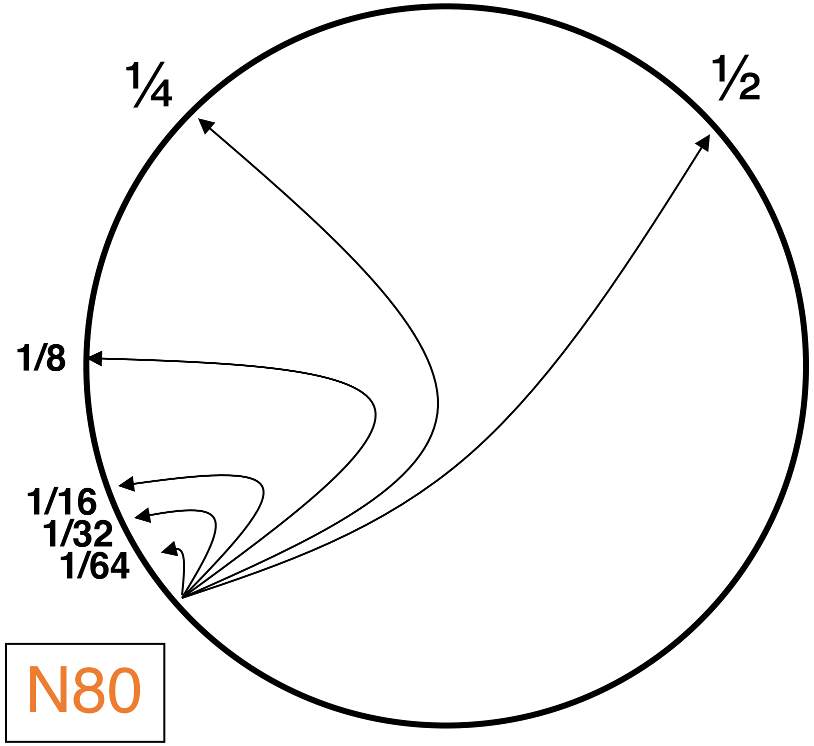
Basic lookup



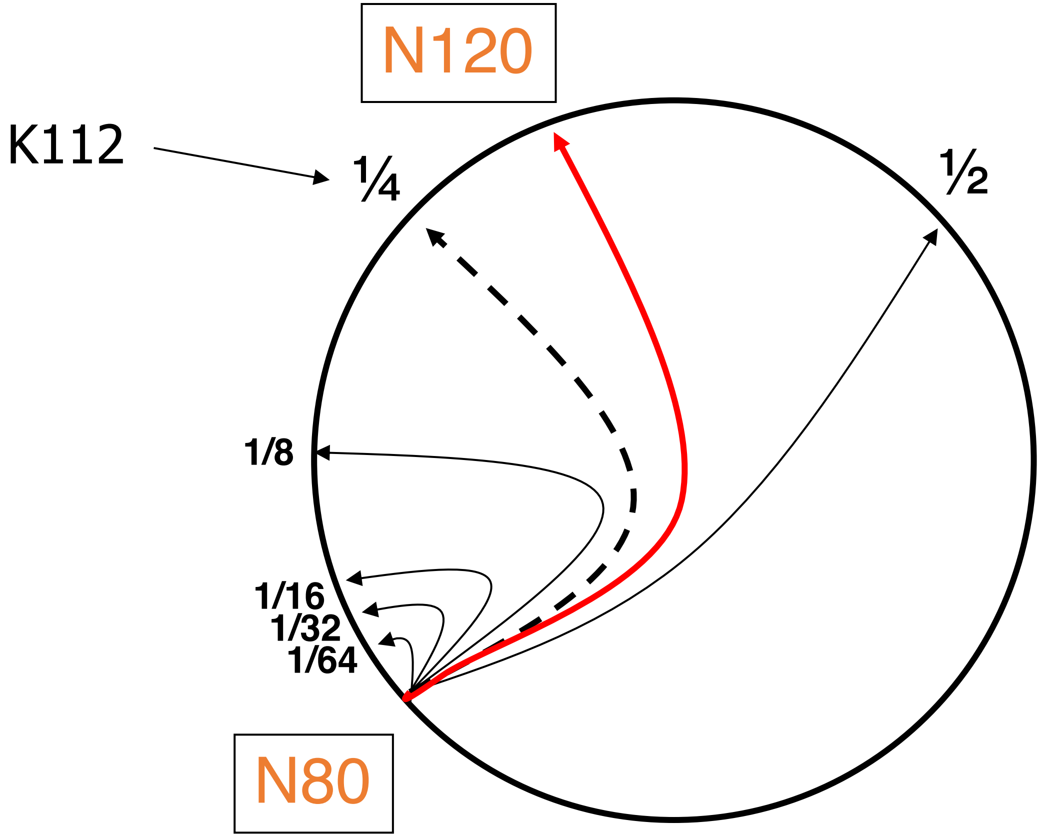
Improving performance

- **Problem:** Forwarding through successor is slow
- Data structure is a linked list: O(n)
- **Idea:** Can we make it more like a binary search?
  - Need to be able to halve distance at each step

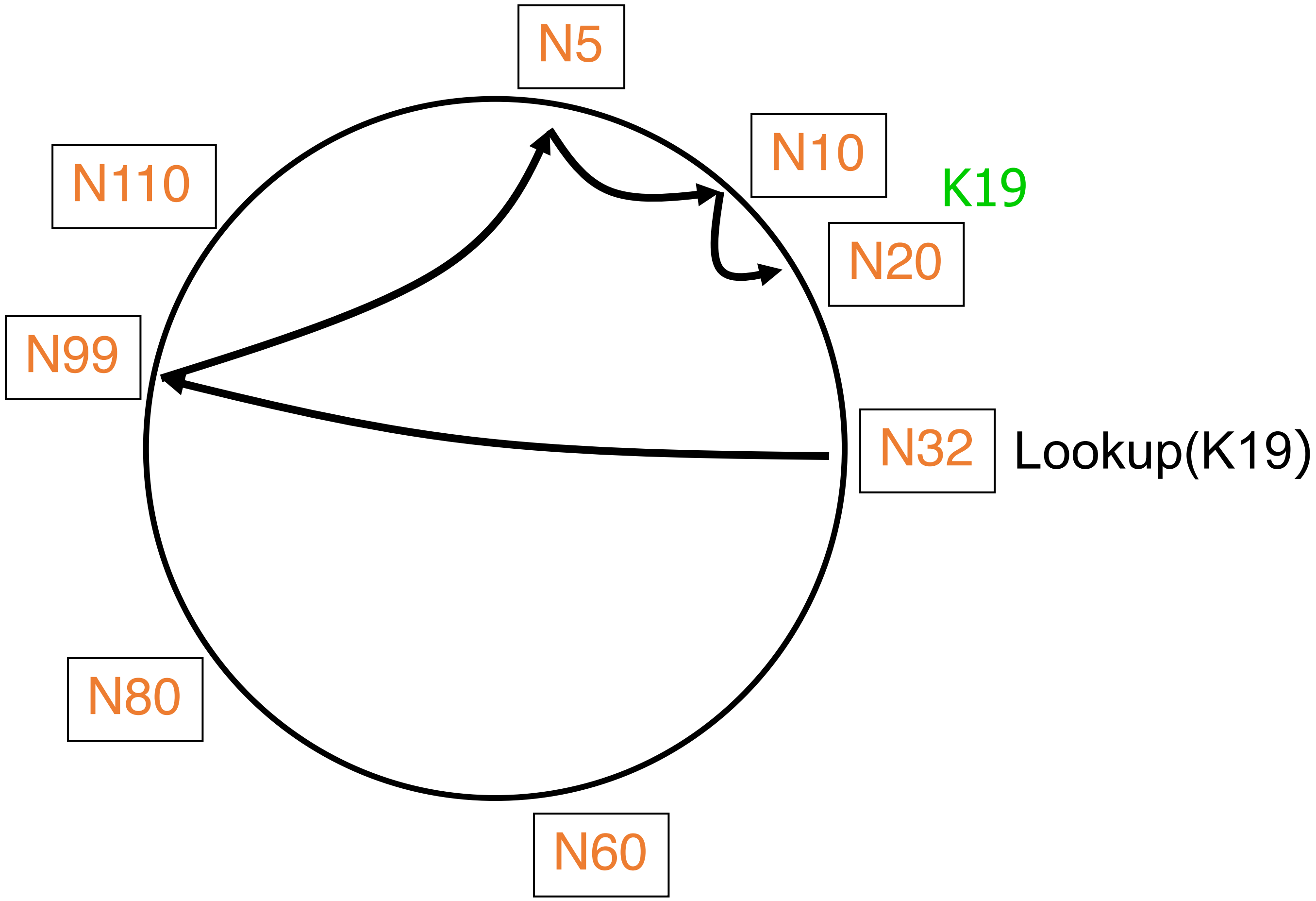
“Finger table” allows log N-time lookups



Finger i Points to Successor of $n+2^i$



Lookups Take O(log *N*) Hops

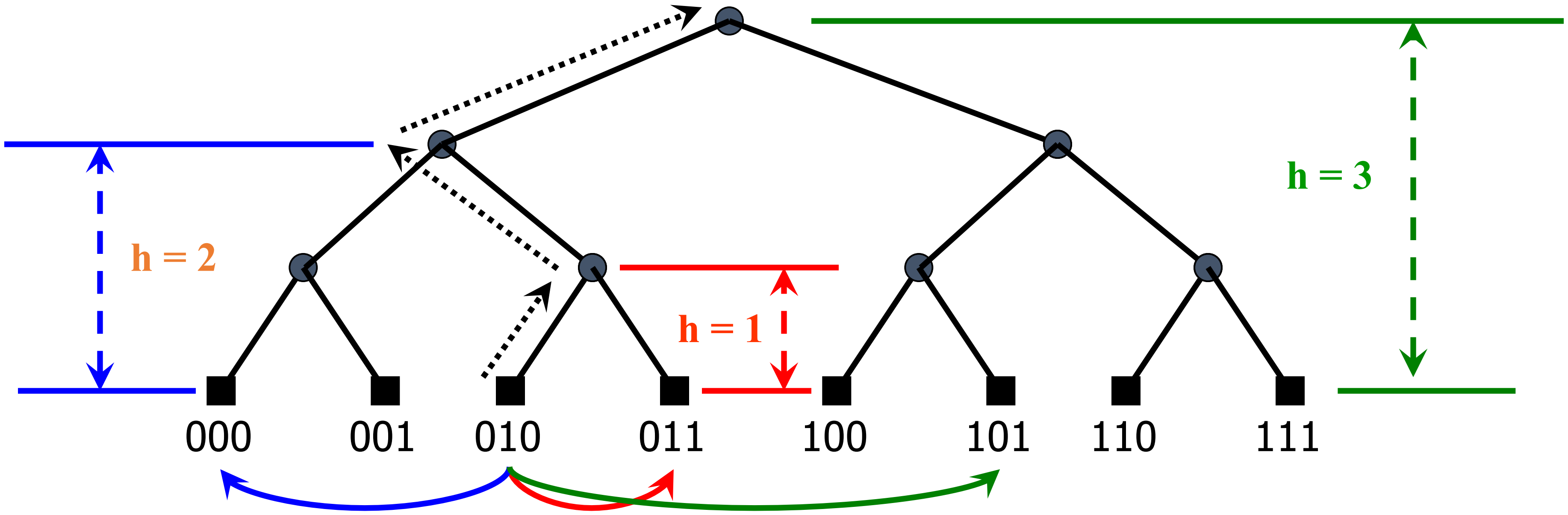


Implication of finger tables

- A **binary lookup tree** rooted at every node
  - Threaded through other nodes' finger tables

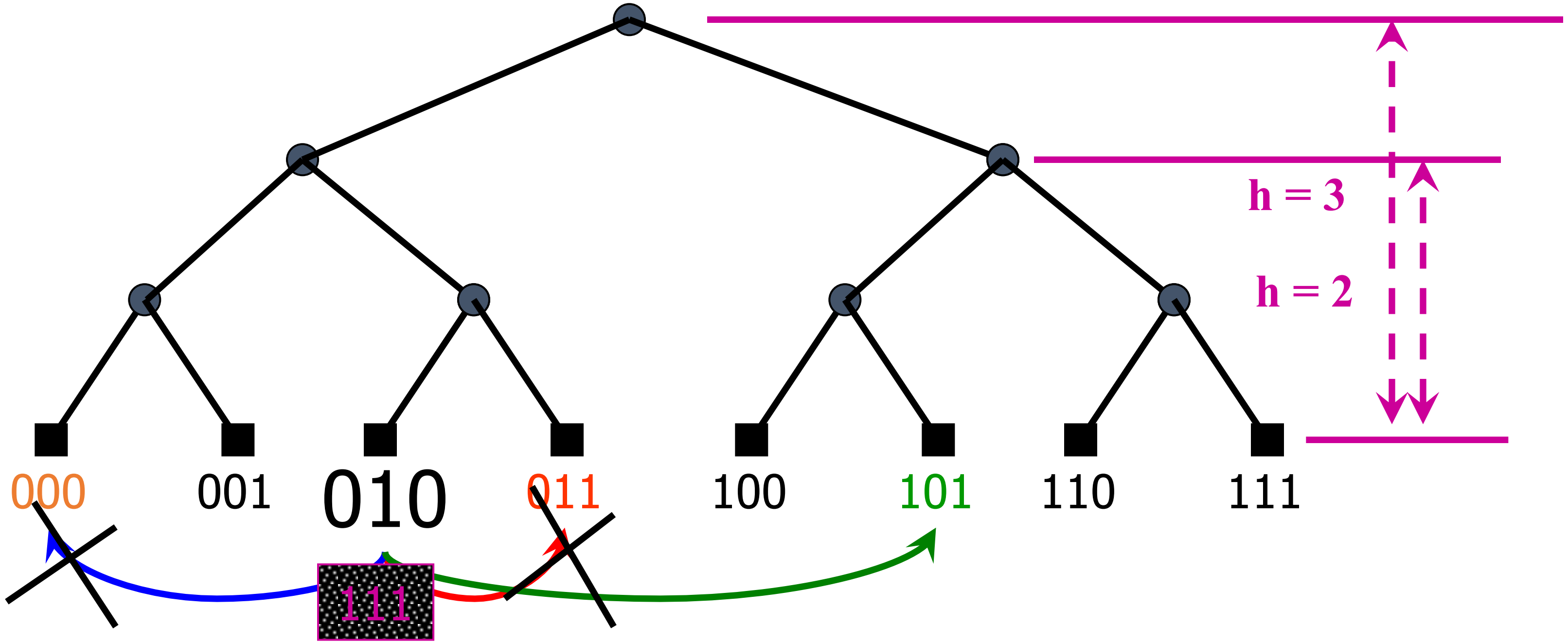
- This is **better** than simply arranging the nodes in a single tree
  - Every node acts as a root
    - So there's **no root hotspot**
    - **No single point** of failure
    - But a **lot more state** in total

Pastry DHT: Network organization



- Nodes are leaves in a tree
- logN neighbors in sub-trees of varying heights

Pastry DHT routing



- Route to the sub-tree with the destination

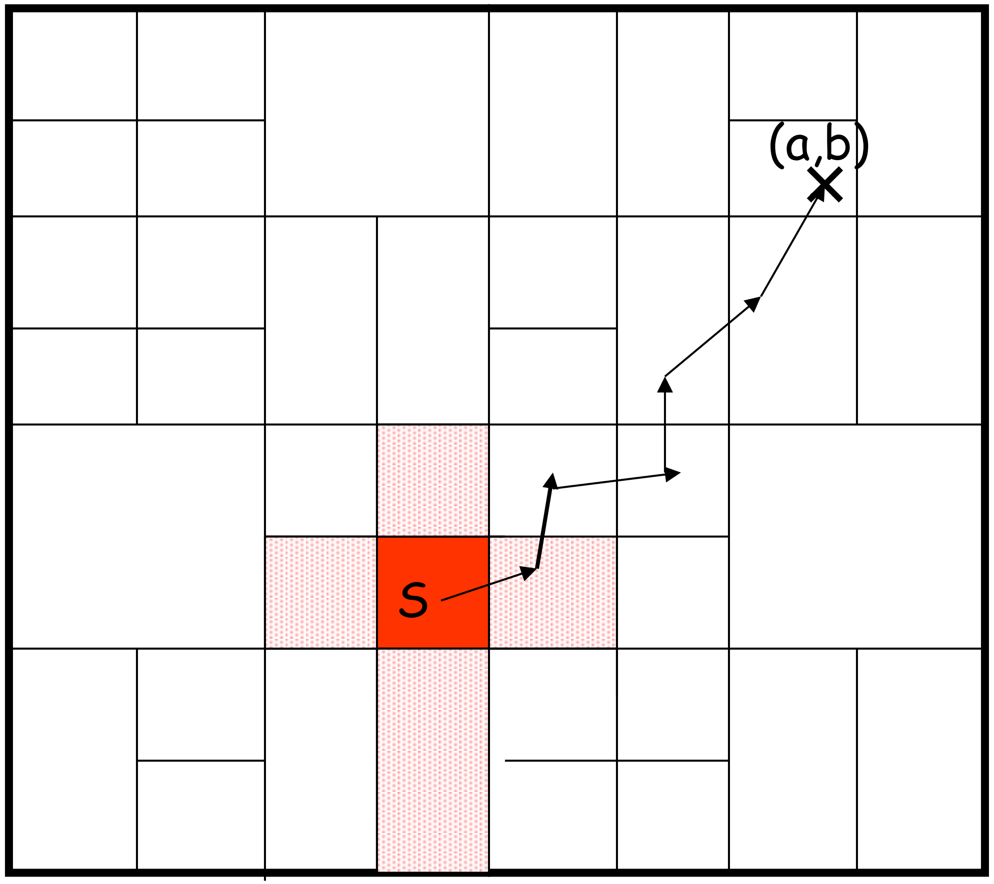
Content-addressable network (CAN) DHT

Embed nodes in a d-dimensional torus

Nodea own keys in their “zone”

Nodes have pointers to their neighbors in each dimension

Route to closest neighbor to the key



DNS vs DHTs

	DNS	DHTs
Node organization	Hierarchy	Flat meshes
Dynamic node membership	Not supported	Supported
Pointers to other nodes	Namespace-dependent (.com will have a LOT; .cs.washington.edu will have a handful)	DHT-design dependent

DNS vs DHTs

Which one is more load-balanced?

Answer: DHTs

- DNS – more load toward the top of the hierarchy
- DHTs – all nodes are equal (assuming keys are evenly distributed)

DNS vs DHTs

Which one is more scalable (amount of state)?

Answer: DHTs

- Because load is more evenly distributed

DNS vs DHTs

Which one is faster?

Answer: DNS

- Typically, 5 queries (depth)
- DHT: log(N) = 16 for N = 100K
  - Each hop could take you half way around the world

Locality-aware DHTs

Prefer neighbors that are proximate

Designs give you flexibility in picking neighbors

- Chord – fingers should point to a node in a key range

Should we build DNS using DHTs?

How about controlling your own availability and load?

- washington.edu depends only on its parents and itself
  - no dependence on cousins, siblings, or children
  - no impact if others go down
- can provision its own resources
  - Control its own cost and service availability
- its load depends only on its zone and children
  - Isolated from others

Next class

Distributed routing – finding paths to destinations

- Distance vector
- Link state
- Path vector