CDNs and Peer-to-Peer

EECS 489 Computer Networks

http://www.eecs.umich.edu/~zmao/eecs489

Z. Morley Mao

Tuesday Nov 9, 2004

Acknowledgement: Some slides taken from Kurose&Ross and Katz&Stoica

This Lecture

- This will be a “why” lecture, not a “how to” one
- Emphasis is on why these developments are important, and where the fit into the broader picture
- TAs will fill in the technical details

Outline

- Motivation: information sharing
  - what's the role of peer-to-peer (P2P)?
- Centralized P2P networks
  - Napster
- Decentralized but unstructured P2P networks
  - Gnutella
- Decentralized but structured P2P networks
  - Distributed Hash Tables
- Implications for the Internet (speculative)

Information Sharing in the Internet

- The Internet contains a vast collection of information (documents, web pages, media, etc.)
- One goal of the Internet is to make it easy to share this information
- There are many different ways this can be done...

In the beginning...

- ...there was FTP
- People put files on a server and allowed anonymous FTP
  - does anyone here remember anonymous FTP?
- Only people who were explicitly told about the file would know to retrieve it
- But it was a painful, command-line interface

The Early Web

- The early web was essentially a GUI for anon ftp
  - URLs were easily distributed pointers to files
  - Browsers allowed one to easily retrieve files
- Web pages could contain pointers to other files
  - not all downloads were result of being explicitly told
- But information sharing was still mostly explicitly arranged
  - someone sent you a URL
  - and you bookmarked it

The Current Web

- Search engines changed the web
  - long before your time....
- Now one can proactively find the desired information, not just wait for someone to tell you about it
- In the process, it became less important who was hosting the information (because they don't need to tell you)
  - the nature of the content is all that matters now

Two Transitions

- From push to pull:
  - old: people would tell others about information (push)
  - new: people can find information via google (pull)
- From hosts to servers
  - anonymous ftp could run on anyone's desktop
  - then migrated to specialized servers
  - the web almost exclusively uses servers
  - popular sites have to use big server farms
- What about "pull" with "hosts"?
  - that's peer-to-peer networking!

Why Is Pull/Host Relevant?

- There are many pieces of content that:
  - are already widely replicated on many machines
  - people want, but don't know where it is
- Setting up a web site for all such content would:
  - attract huge amount of traffic
  - require sizable investment in server farm and bandwidth
- If we could harness the hosts that already have the content, we wouldn't need a server farm!
- But how do users know which host to contact?

Peer-to-Peer (P2P) Networking

- Aims to use the bandwidth and storage of the many hosts
  - sum of access line speeds and disk space
- But to use this collection of machines effectively requires coordination on a massive scale
  - key challenge: who has the content you are looking for?
- Moreover, the hosts are very flaky
  - behind slow links
  - often connected only a few minutes
  - so system must be very robust

Napster

- Centralized search engine:
  - all hosts with songs register them with central site
  - users do keyword search on site to find desired song
  - site then lists the hosts that have the song
  - user then downloads content

- What makes this work?
  - central site only has to handle searches: little bandwidth
  - vast collection of hosts can supply huge aggregate bandwidth
  - system is self-scaling: more users means more resources

What Happened to Napster?

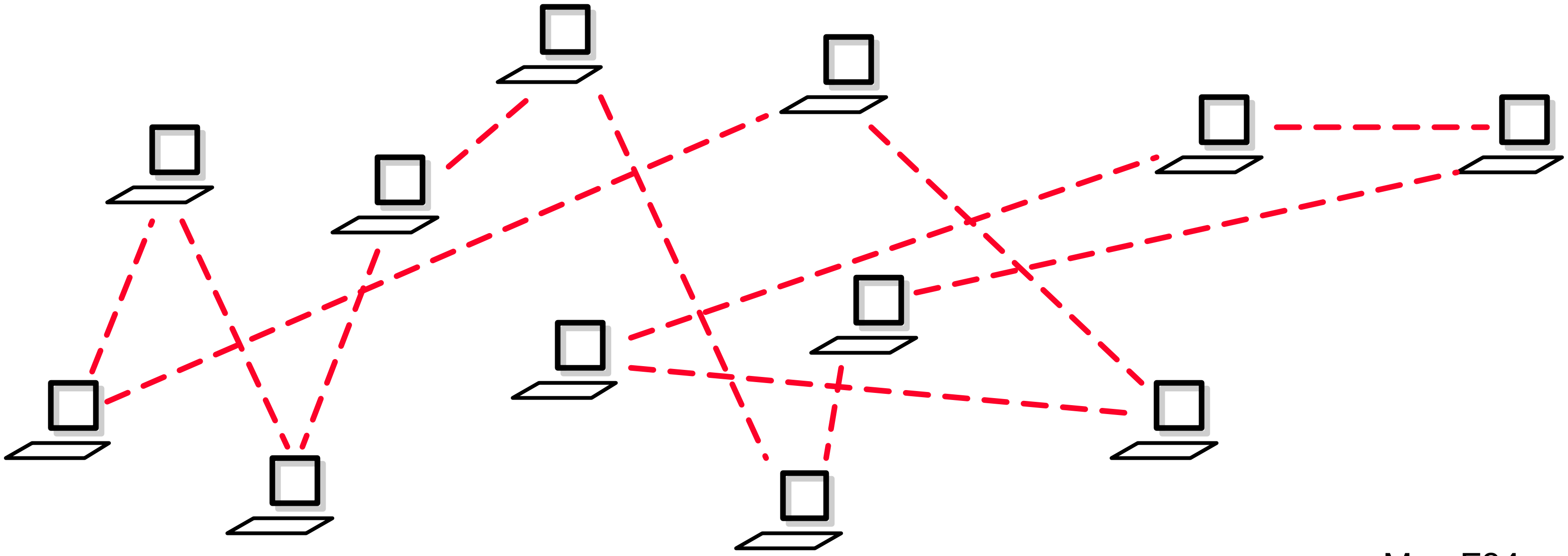
- Fastest growing Internet application ever
  - P2P traffic became, and remains, one of the biggest sources of traffic on the Internet!
- But legal issues shut site down
- Centralized system was vulnerable to legal attacks, and system couldn't function without central site
- Can one still do "pull" without central site?
  - that's the hard question in peer-to-peer networking!

Gnutella

- An example of an unstructured, decentralized P2P system
- Context:
  - many hosts join a system
  - each offers to share its own content
  - in return, each can make queries for others content
- Goal:
  - enable users to find desired content on other hosts

“Basic” Gnutella

- Step one: form an overlay network
  - each host, when it joins, “connects” to several existing Gnutella members
  - an “overlay” link is merely the fact that the nodes know each other’s IP address, and thus can send each other packets



“Unstructured” Overlay

Gnutella is unstructured in two senses:

- Links between nodes are essentially random
- The content of each node is random (at least from the perspective of Gnutella)

Implications:

- Can’t route on Gnutella
- Wouldn’t know where to route even if could

Querying in Gnutella

- Queries are typically keyword searches
- Each query is flooded within some scope
  - TTL is used to limit scope of flood
  - flooding means you don't need any routing infrastructure
- All responses to queries are forwarded back along path query came from
  - path marked with breadcrumbs
  - gives a degree of privacy to requester

Gnutella Performance

- Tradeoff:
  - if TTL is small, then searches won't find desired content
  - if TTL is large, network will get overloaded
- Either Gnutella overloads network, or doesn't provide good search results

Gnutella Enhancements

- Supernodes:
  - normal nodes attach to supernodes, who search for them
  - only flood among well-connected supernodes

- Random-walk rather than flooding
  - provides correct TTL automatically

- Proactive replication
  - replicate content that is frequently queried, to make it easier to find

In Reality

- Gnutella++ works well enough
  - KaZaA, etc.
- Why?
  - enhancements (supernodes)
  - query distribution
- Most downloads are for widely-replicated content
  - Gnutella is good at finding the “hay”
  - But how would you find “needles”?

Finding Objects by Name

- Assume you know the “name” of an object
  - song title, file name, etc.
- Assume that there is one copy of this object in the system
- Is there a way to store this object so that anyone can find it merely by knowing its name?
- Sound familiar? Hash tables

Distributed Hash Tables (DHTs)

- Hash Table
  - data structure that maps “keys” to “values”
  - essential building block in software systems

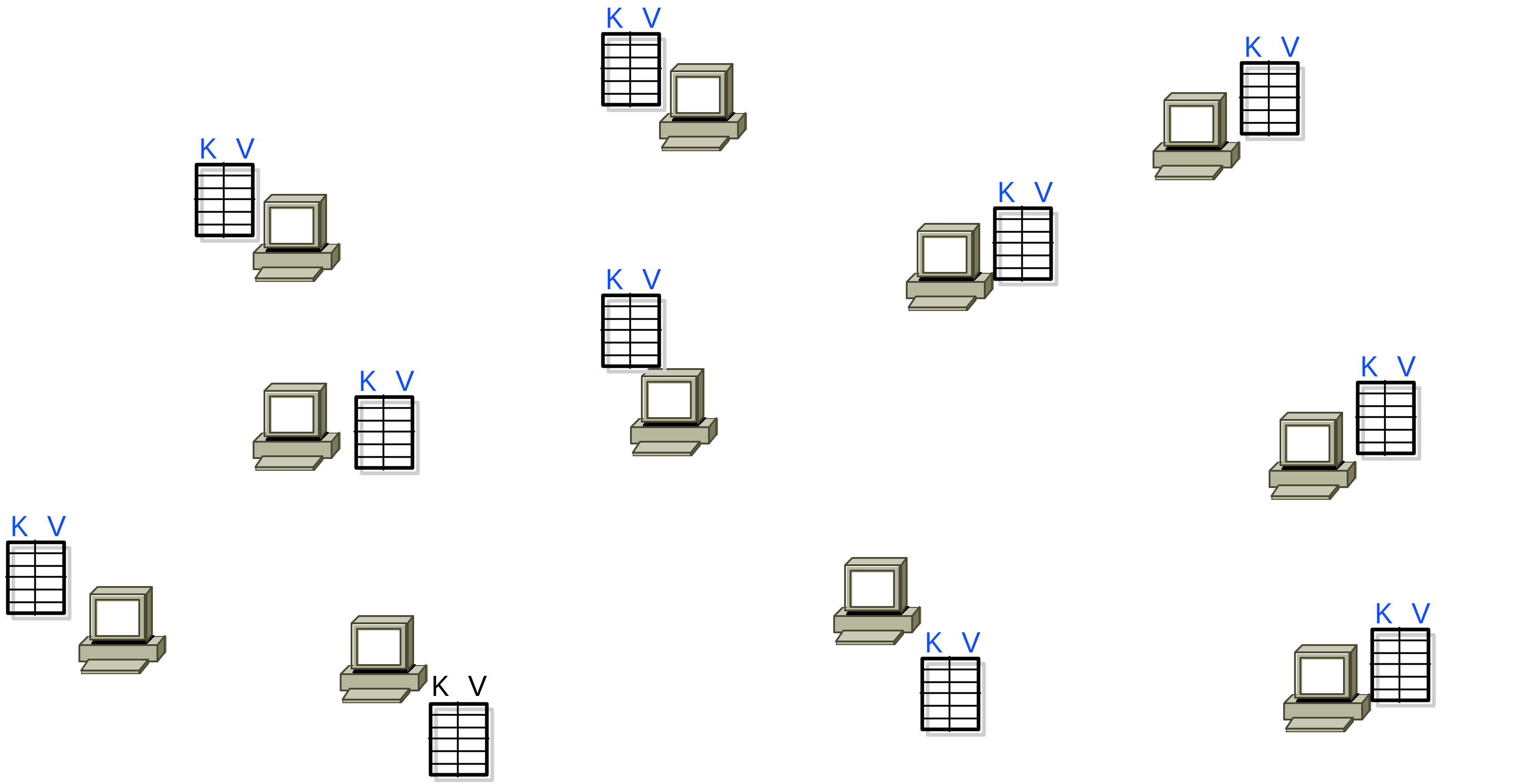
- Distributed Hash Table (DHT)
  - similar, but spread across the Internet

- Interface
  - insert(key, value)
  - lookup(key)

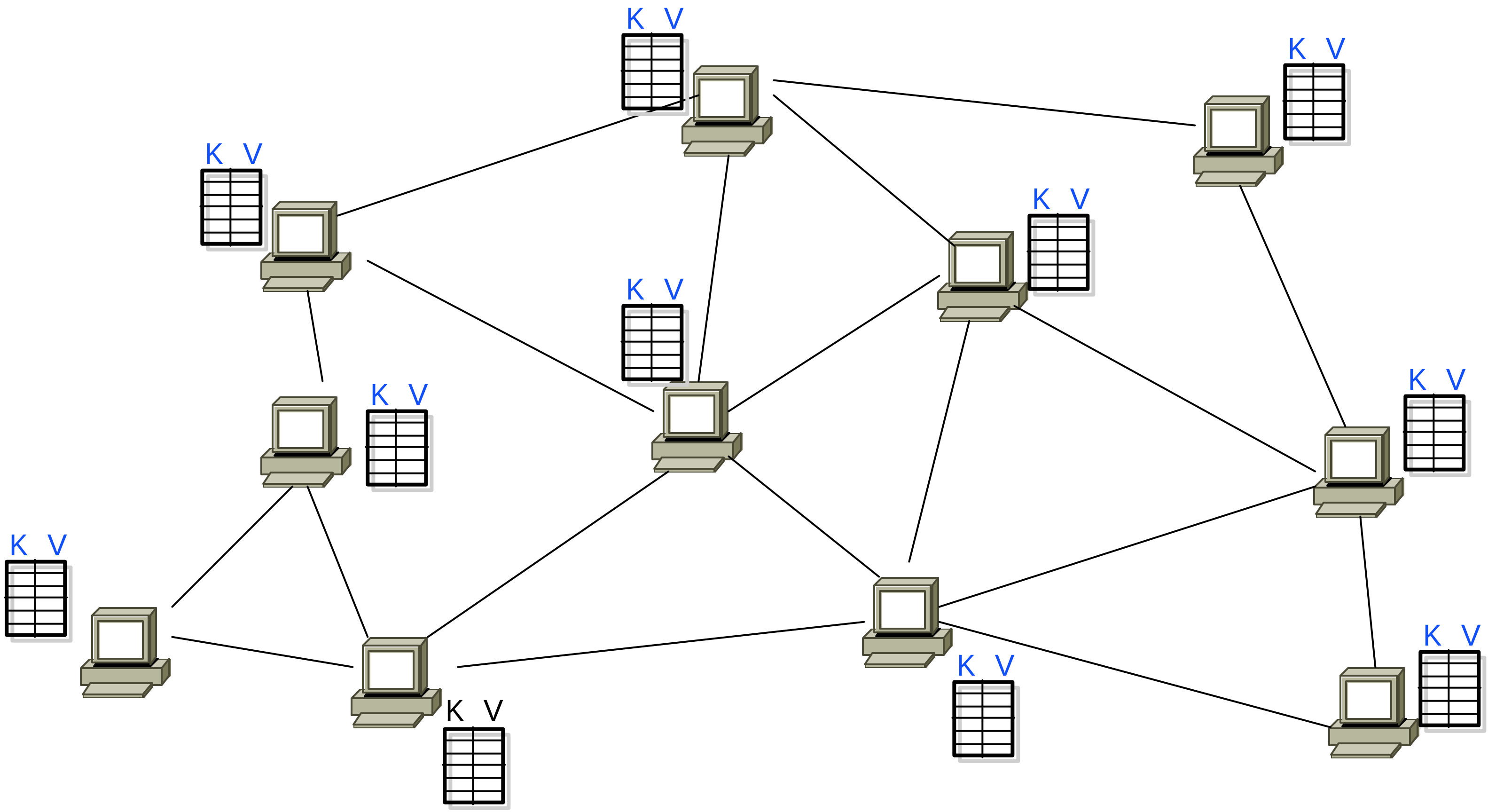
Usage

- key = hash(name)
  - hash function is a deterministic function that is quasi-random
  - gives uniform distribution of keys
- Store by key
- Retrieve by key

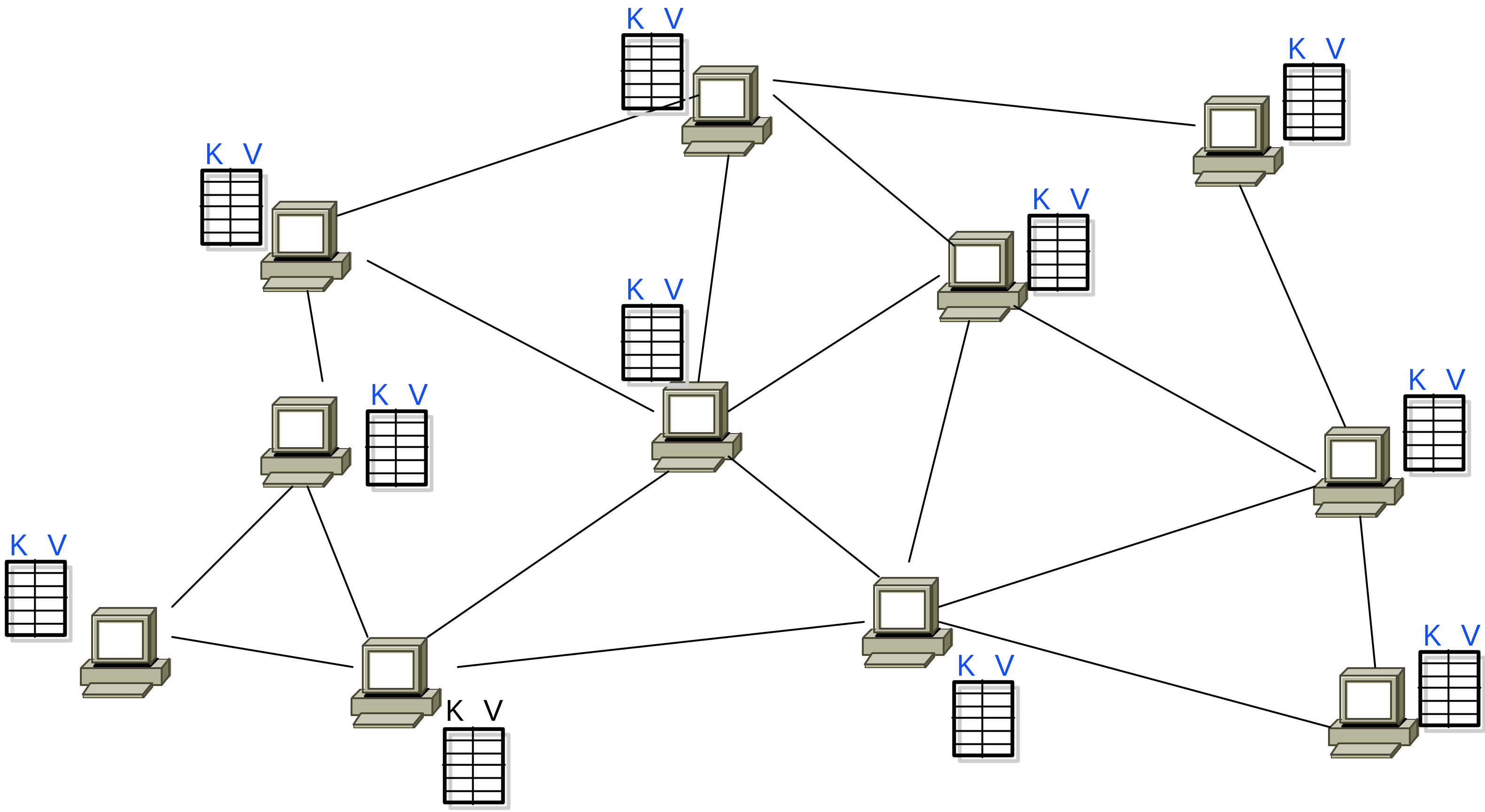
DHT: basic idea



DHT: basic idea

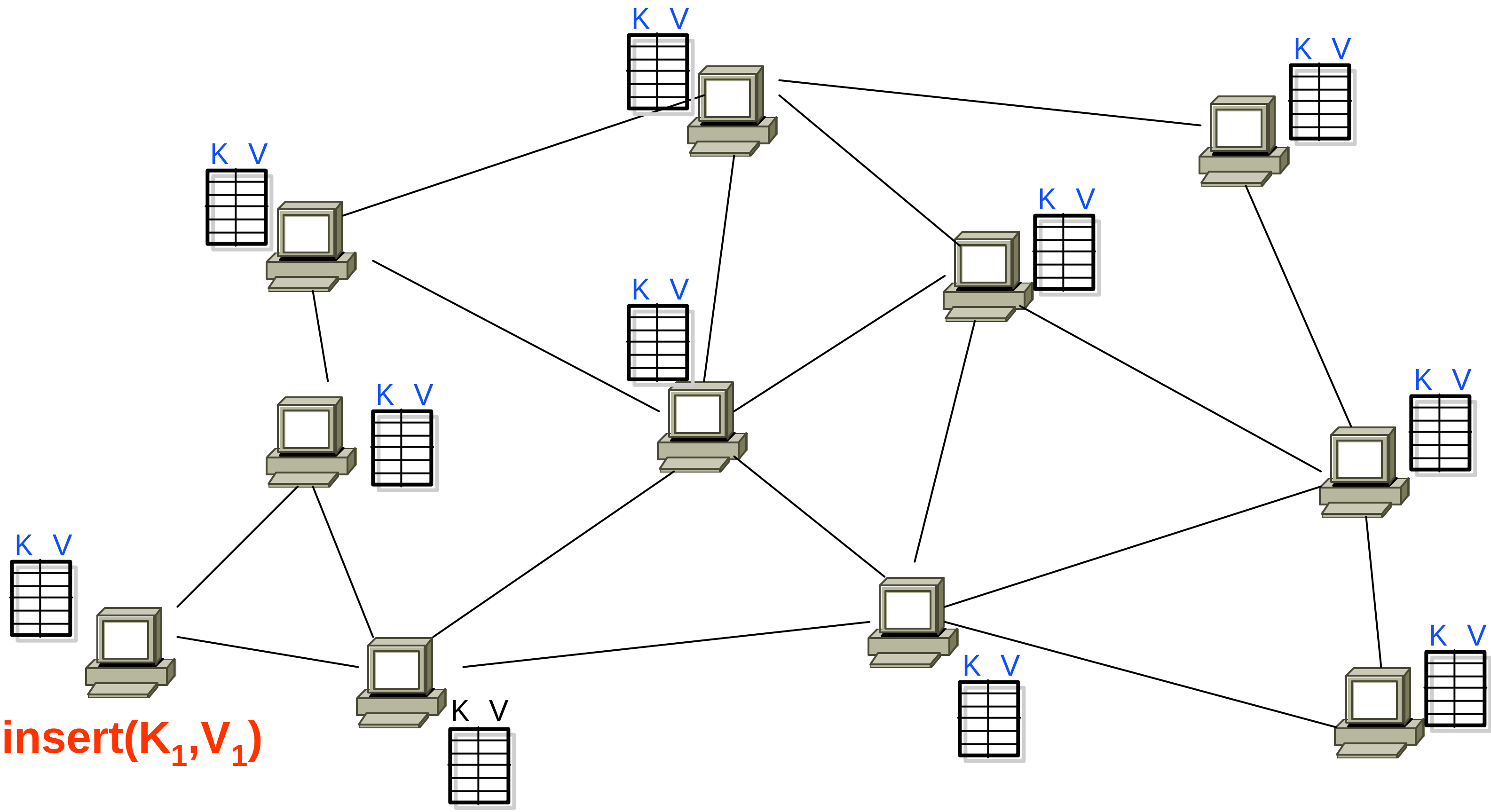


DHT: basic idea



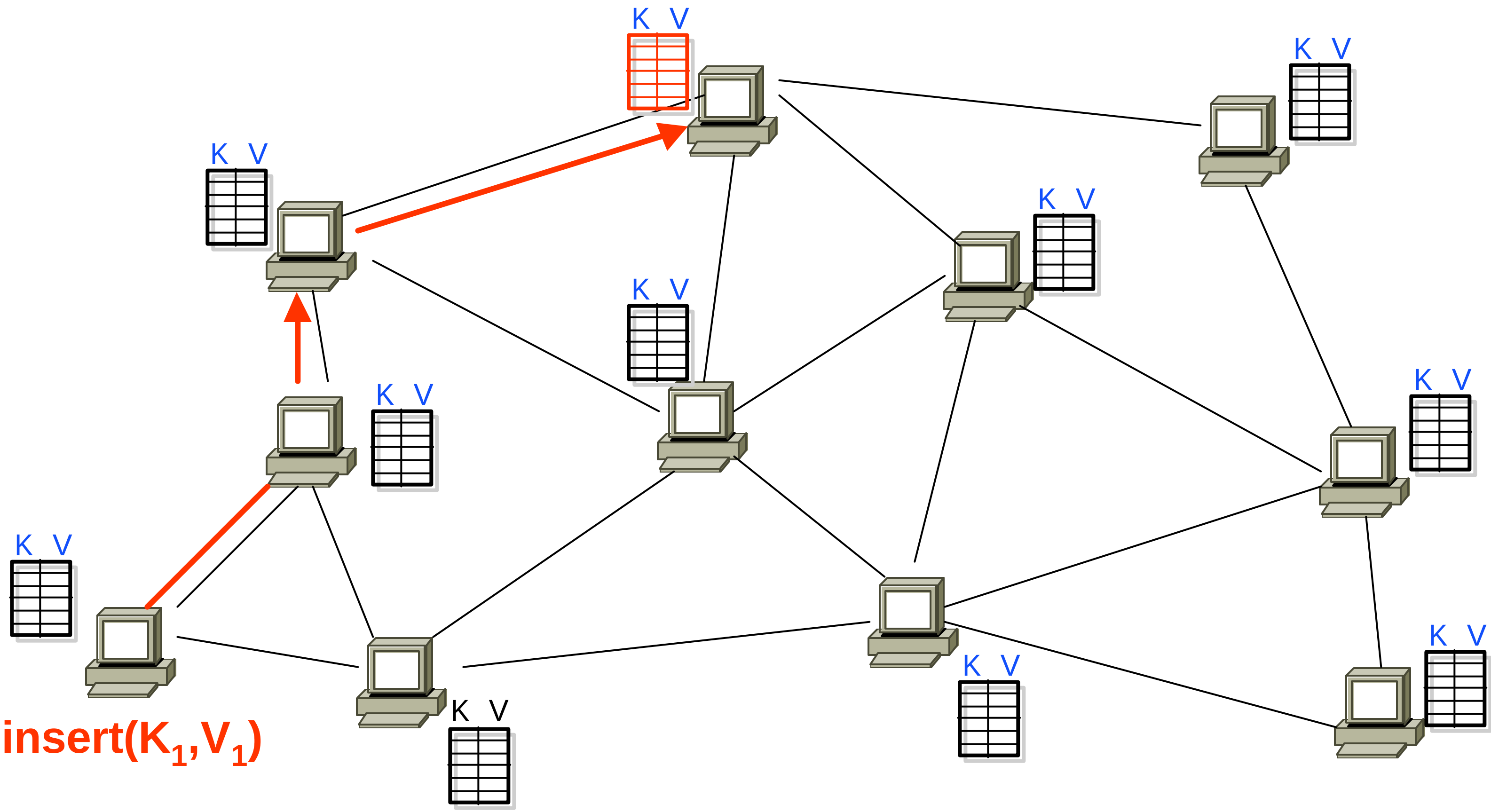
Operation: take *key* as input; route messages to node holding *key*

DHT: basic idea



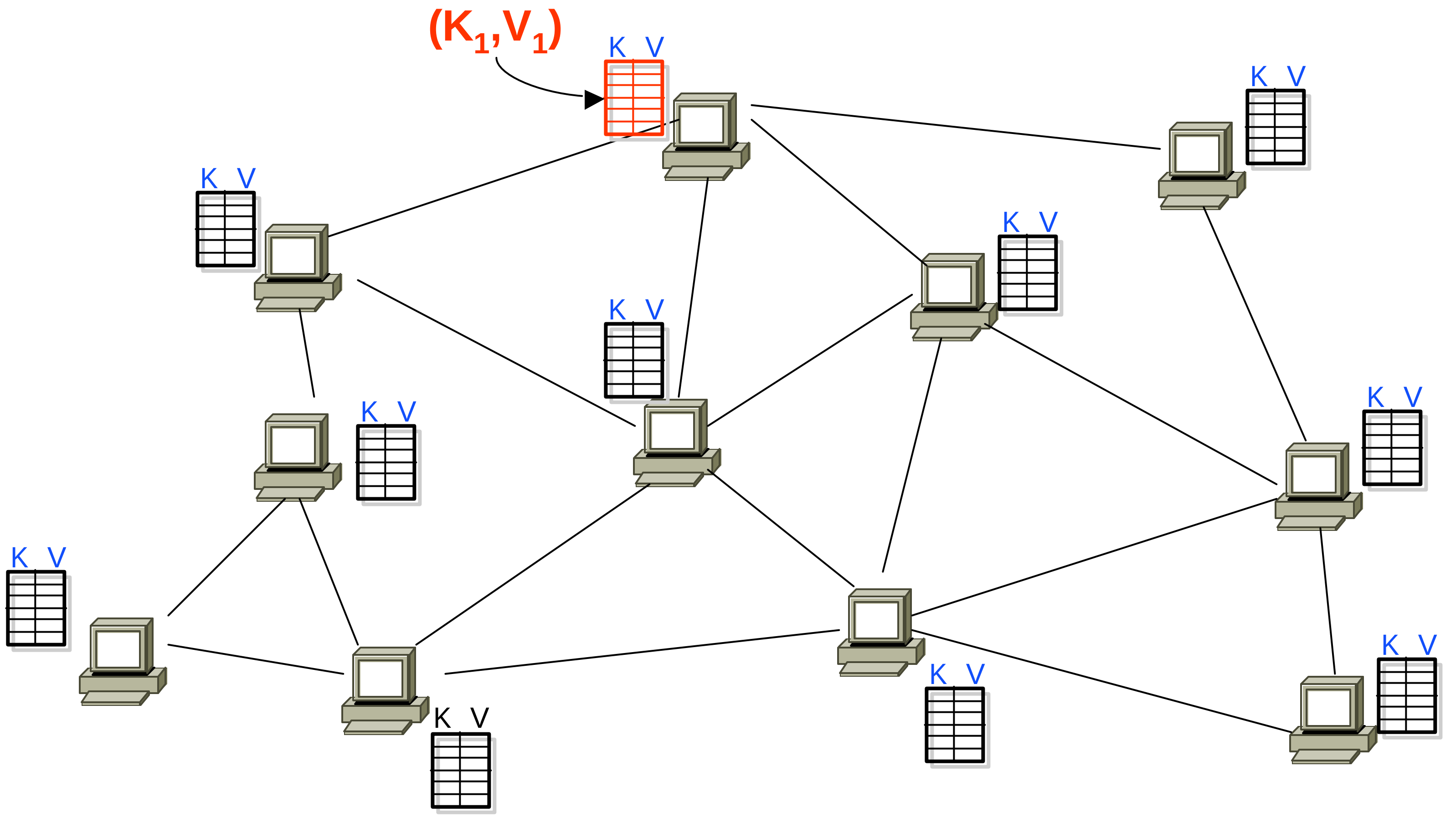
Operation: take *key* as input; route messages to node holding *key*

DHT: basic idea



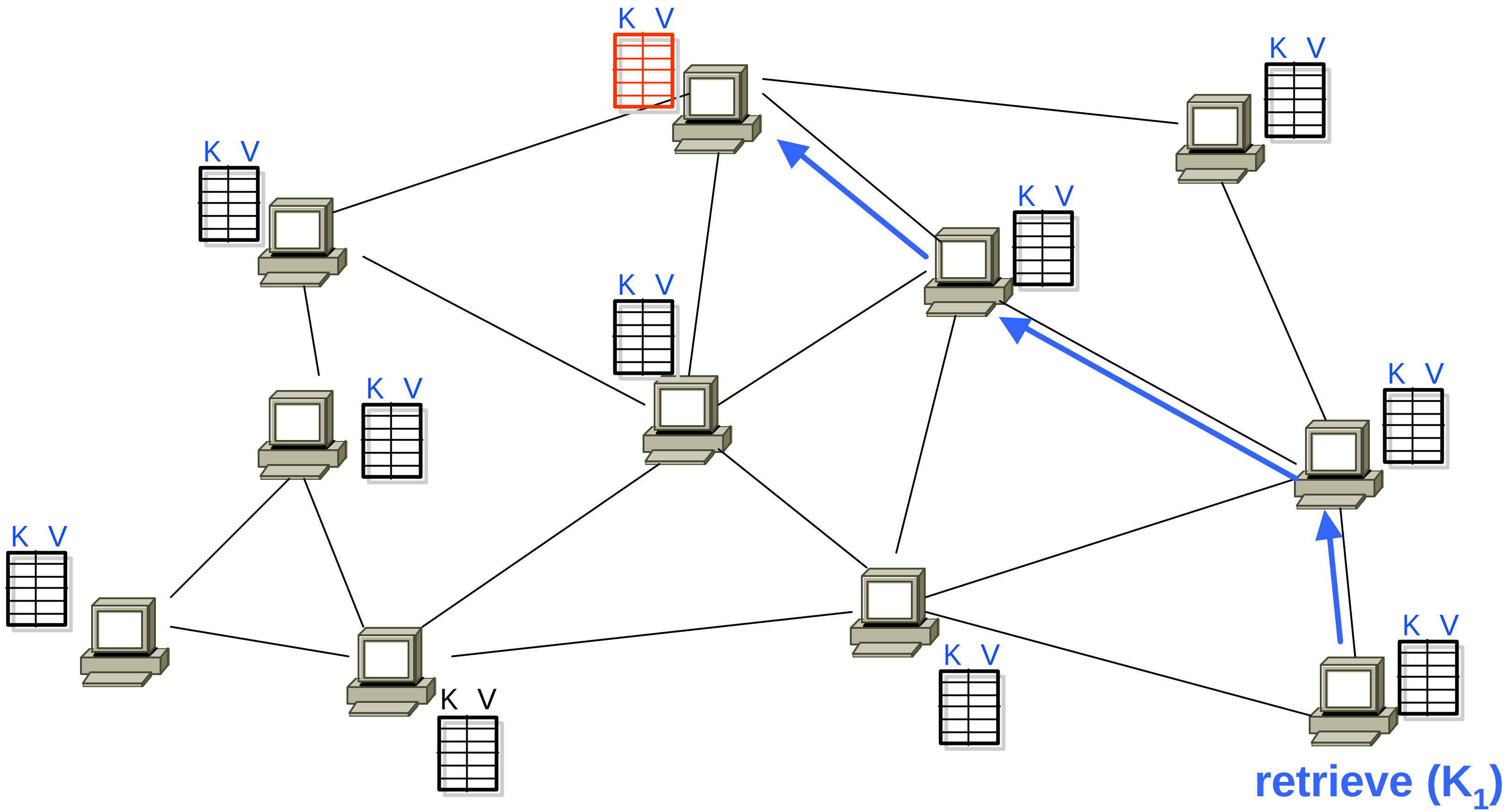
Operation: take *key* as input; route messages to node holding *key*

DHT: basic idea



Operation: take *key* as input; route messages to node holding *key*

DHT: basic idea



Operation: take *key* as input; route messages to node holding *key*

DHT Designs

- There are many DHT designs
  - invented in 2000, so they are quite new
- I will present CAN, readings present others
  - details will be gone over by your TAs
- But don’t worry about the details, focus on the general idea
- In what follows, id or identifier is a key

General Approach to DHT Routing

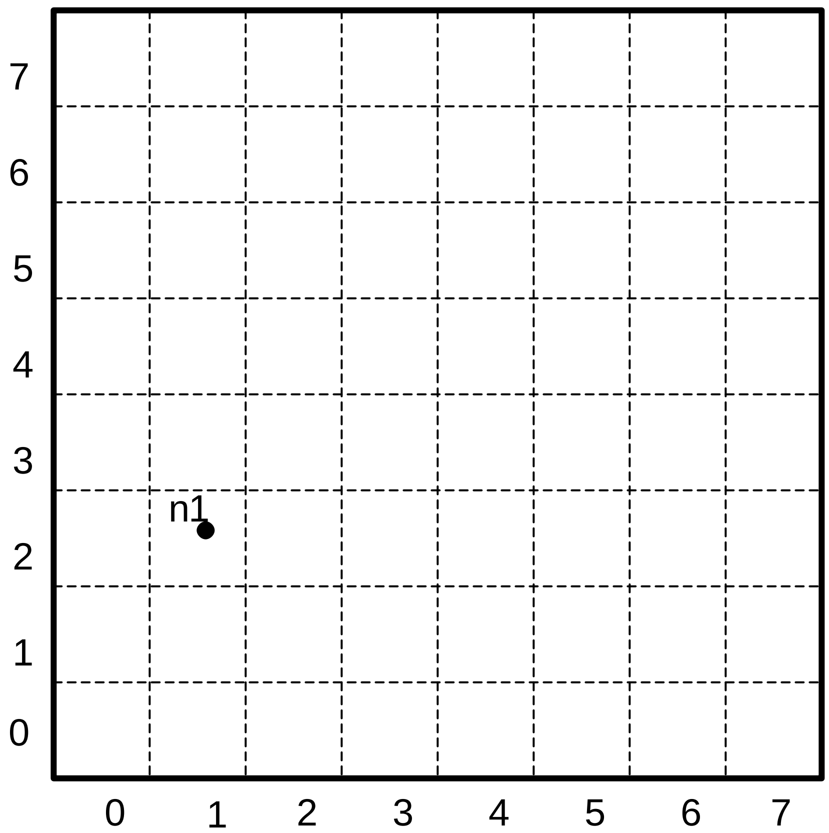
- Pick an identifier space
  - ring, tree, hypercube, d-dimensional torus, etc.
- Assign node ids randomly in space
  - choose a "structured" set of neighbors
- Assign objects ids (keys) randomly via hash function in space
  - Assign an object to node that is "closest" to it
- When routing to an id, pick neighbor which is closest to id
  - if neighbor set is wisely chosen, routing will be efficient

Content Addressable Network (CAN)

- Associate to each node and item a unique *id* in an *d*-dimensional space
- Properties
  - Routing table size O(*d*)
  - Guarantees that a file is found in at most $d*n^{1/d}$ steps, where *n* is the total number of nodes

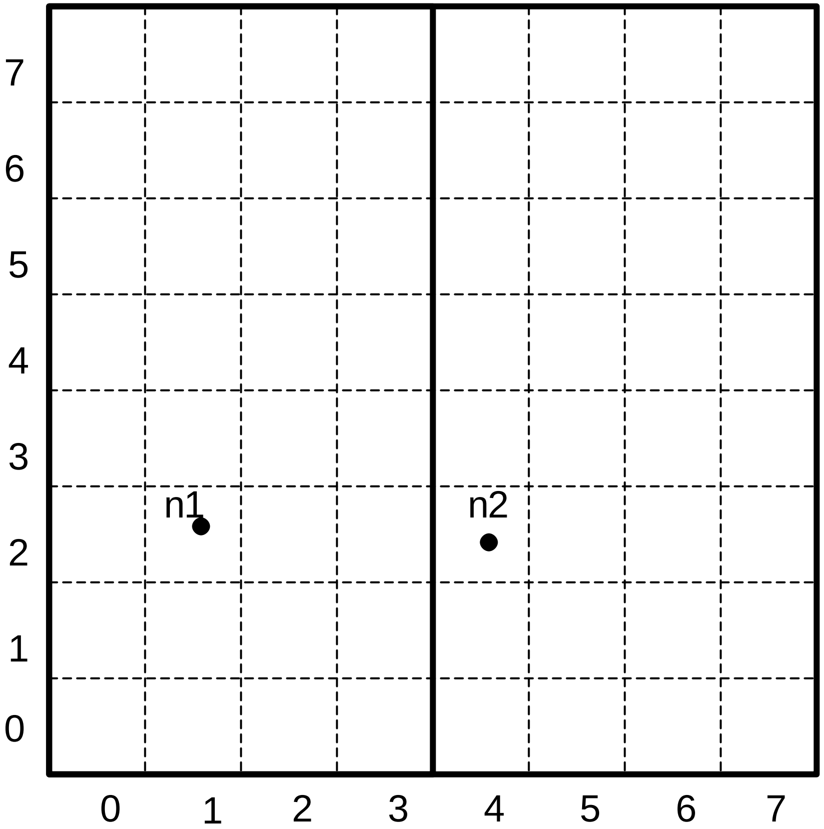
CAN Example: Two Dimensional Space

- Space divided between nodes
- All nodes cover the entire space
- Each node covers either a square or a rectangular area of ratios 1:2 or 2:1
- Example:
  - Assume space size (8 x 8)
  - Node n1:(1, 2) first node that joins → cover the entire space



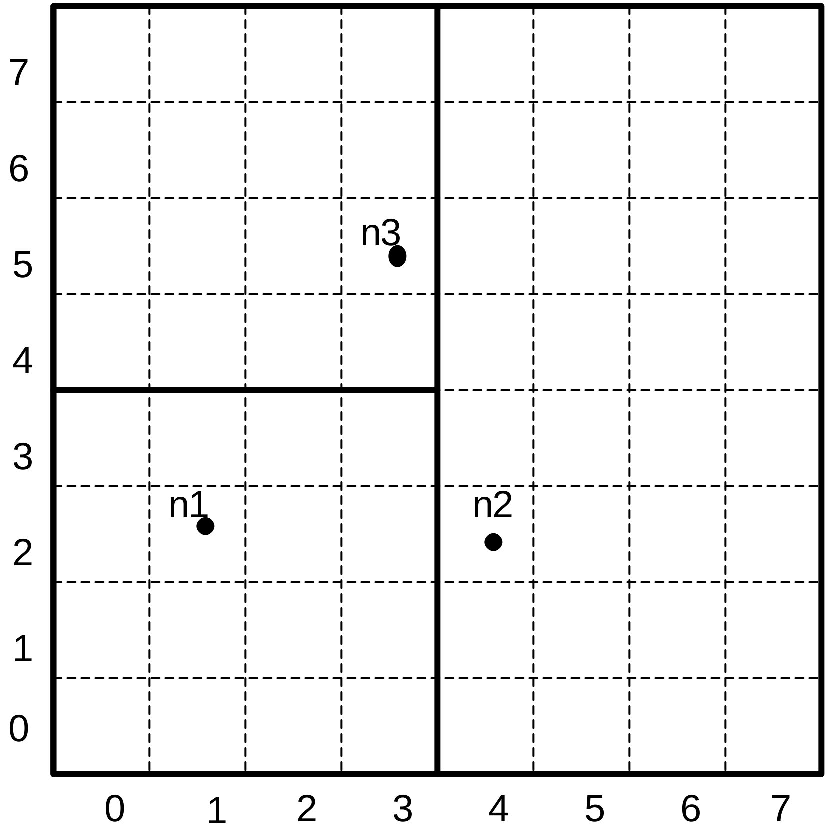
CAN Example: Two Dimensional Space

- Node n2:(4, 2) joins → space is divided between n1 and n2



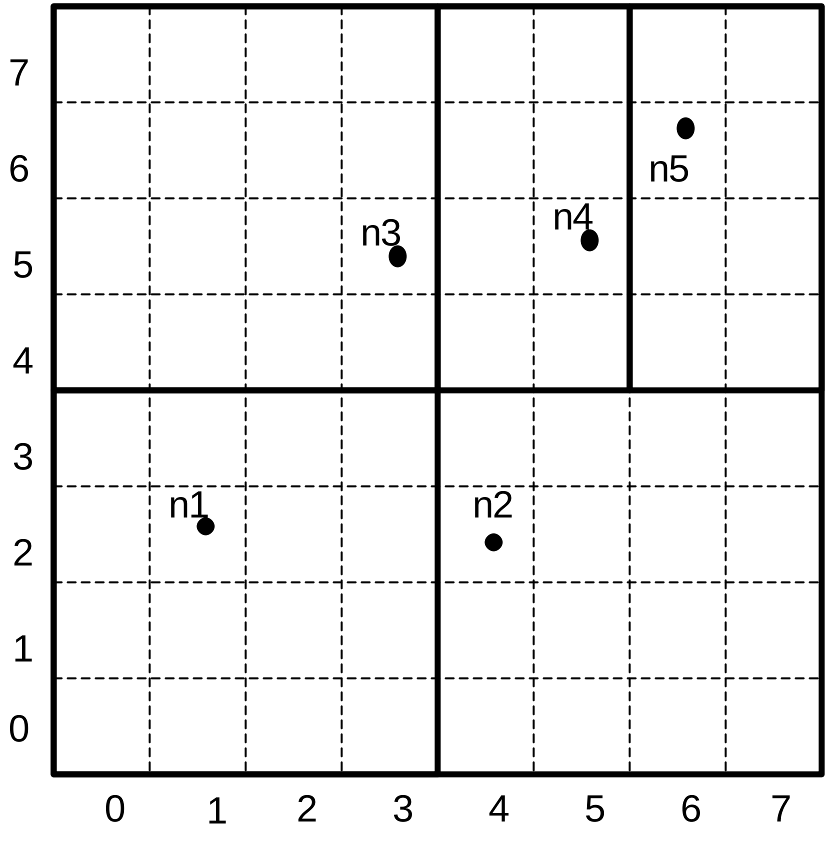
CAN Example: Two Dimensional Space

- Node n2:(4, 2) joins → space is divided between n1 and n2



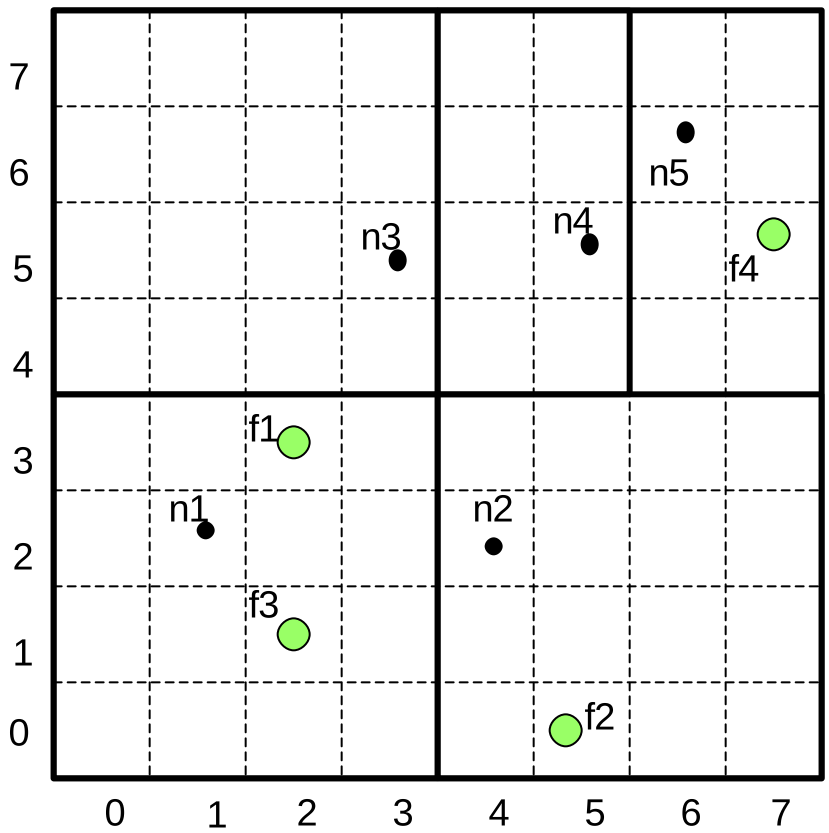
CAN Example: Two Dimensional Space

- Nodes n4:(5, 5) and n5:(6,6) join



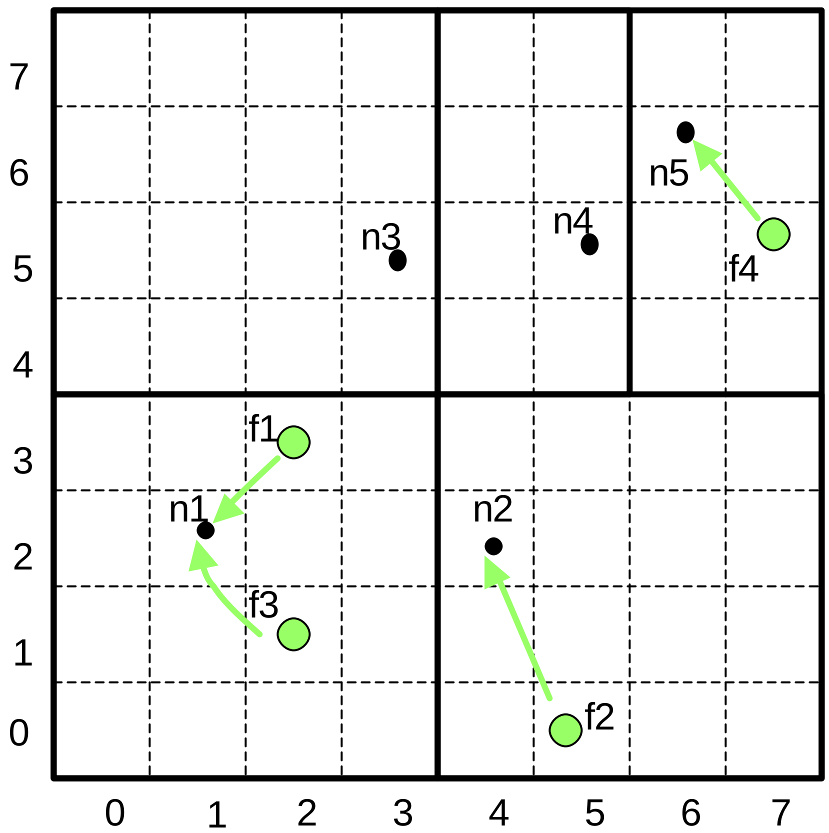
CAN Example: Two Dimensional Space

- Nodes: n1:(1, 2); n2:(4,2); n3:(3, 5); n4:(5,5);n5:(6,6)
- Items: f1:(2,3); f2:(5,1); f3:(2,1); f4:(7,5);



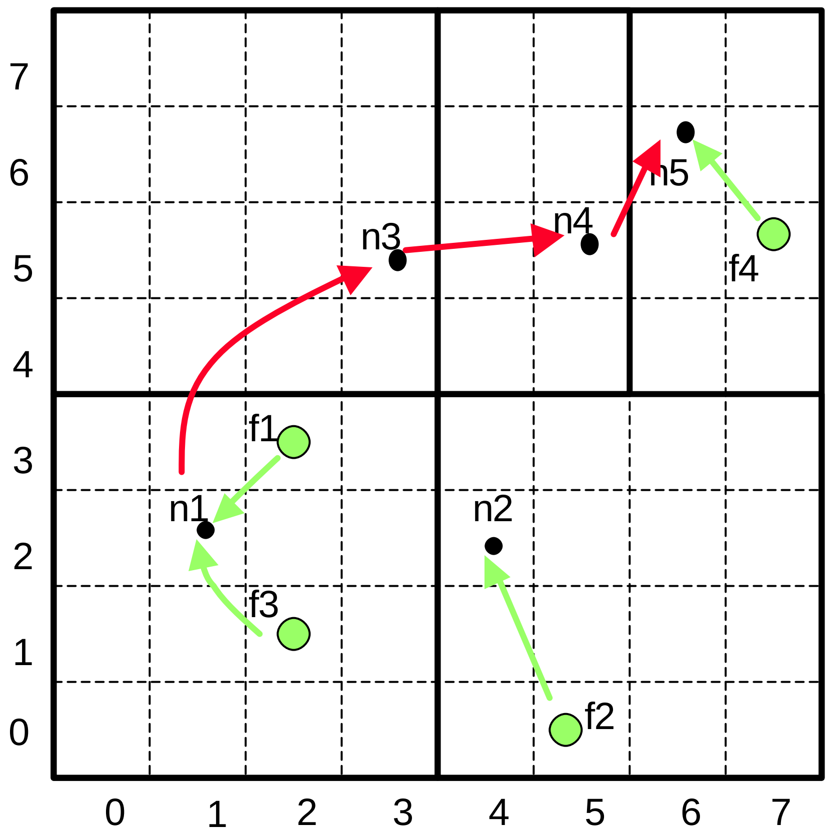
CAN Example: Two Dimensional Space

- Each item is stored by the node who owns its mapping in the space



CAN: Query Example

- Each node knows its neighbors in the *d*-space
- Forward query to the neighbor that is closest to the query *id*
- Example: assume n1 queries f4

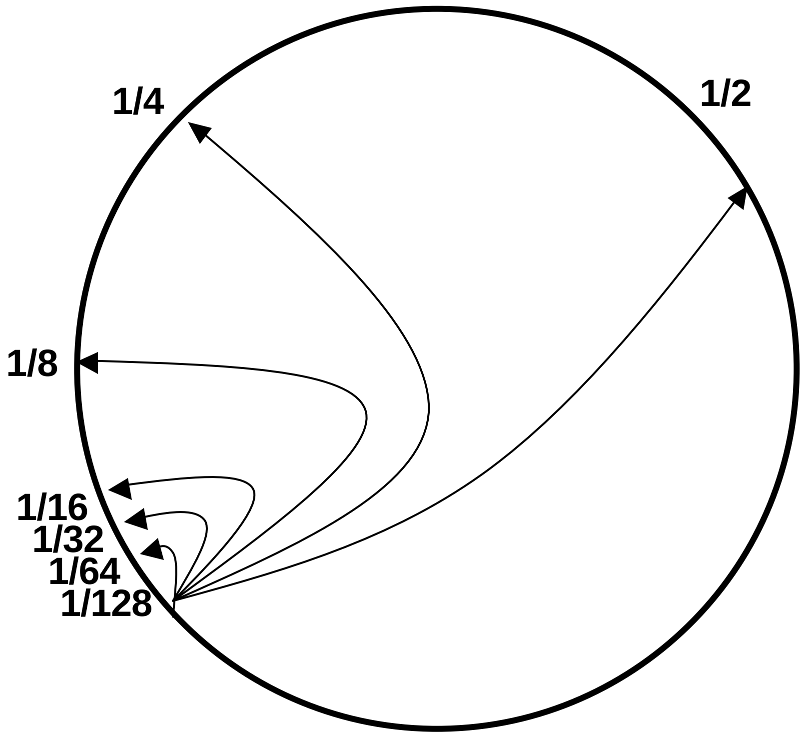


Many Other DHT Designs

- Chord:
  - id space is circle
  - routing table includes predecessor node and nodes 2^{-i} away
  - routing always halves distance

- Pastry and Tapestry
  - id space is tree
  - routing table includes neighboring subtree of varying heights
  - routing always fixes at least one bit on each step

Chord Routing Table



Performance

- Routing in the overlay network can be more expensive than in the underlying network
- Because usually there is no correlation between node ids and their locality; a query can repeatedly jump from Europe to North America, though both the initiator and the node that store the item are in Europe!
- Solution: make neighbor relationships depend on link latency
  - Can achieve “stretch” of ~1.3

Other Issues

- Data replication
- Security
- Resilience to failures, node churn
- Monitoring
- .....

General DHT Properties

- Fully decentralized: all nodes equivalent
- Self-organizing: no need to explicitly arrange routing, algorithm does it automatically
- Robust: can tolerate node failures
- Scalable: can grow to immense sizes
- Flat namespace: does not impose semantics
  - as opposed to DNS

Structured vs Unstructured

- Unstructured:
  - can tolerate churn
  - can find hay
  - can do searches easily
- Structured:
  - designed for needles
  - have trouble with keyword searches
  - have some trouble with extreme churn
  - have different sharing model

Other Design Options

- Centralized?
  - single point-of-failure
  - requires infrastructure to scale (business model)

- Hierarchical?
  - requires given hierarchical organization
  - static hierarchy of servers: not robust or flexible
  - dynamic hierarchy of servers: essentially a DHT

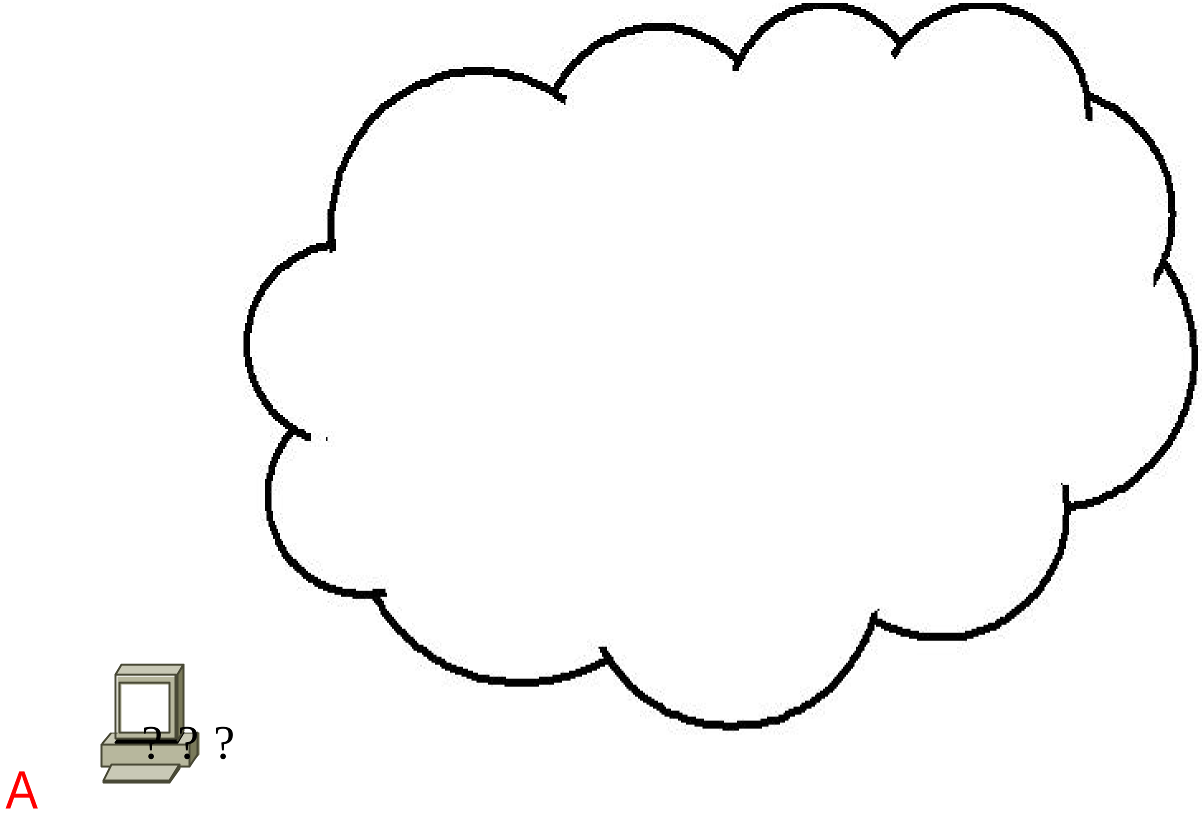
Are DHTs Just for File Sharing?

- Think of DHTs as a new DNS
  - mapping names to identifiers
  - identifiers are persistent and general
- A web based with persistent pointers, not ephemeral URLs
- Overlay networks based on persistent keys, not changeable IP addresses
  - send to identifier, translated into current IP address

More Generally

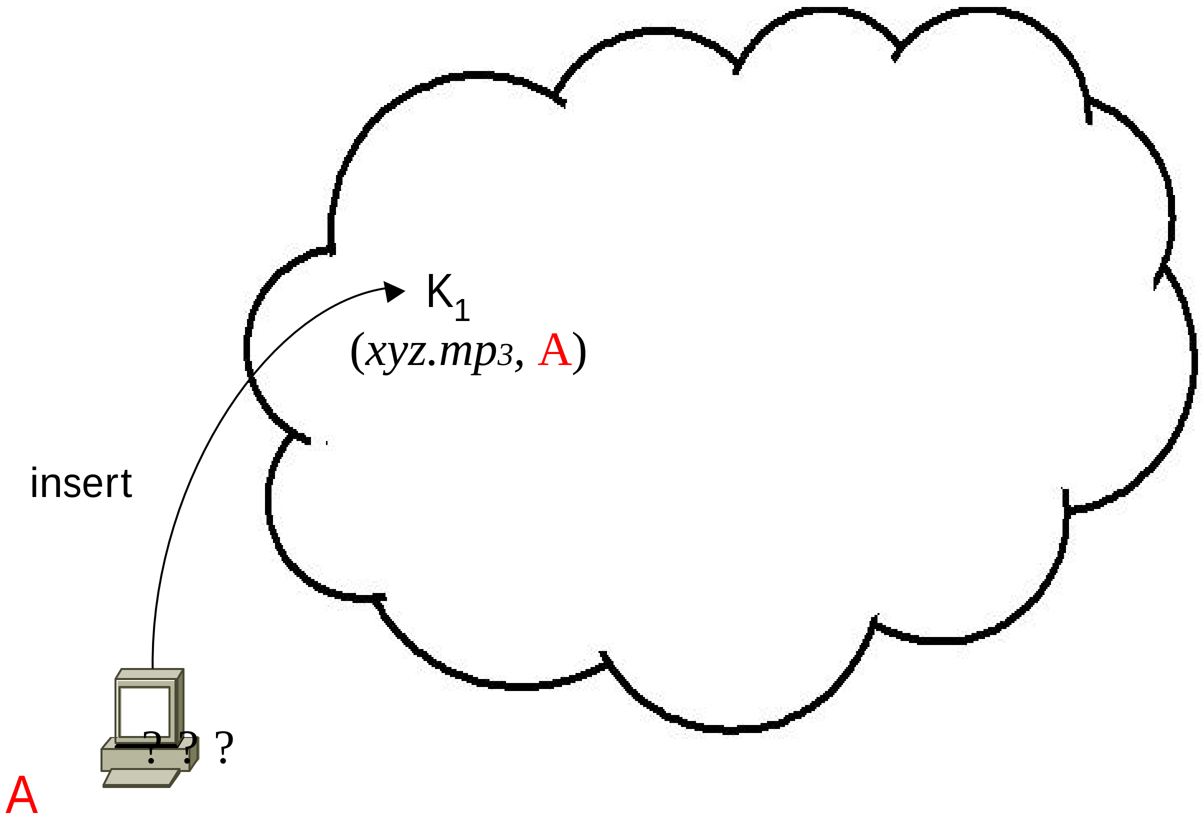
- Hash tables are useful data structures for many programs
- Distributed hash tables should be generally useful data structures for distributed programs
- Examples: file systems, event notification, application-layer multicast, mail systems, ...

Indexing



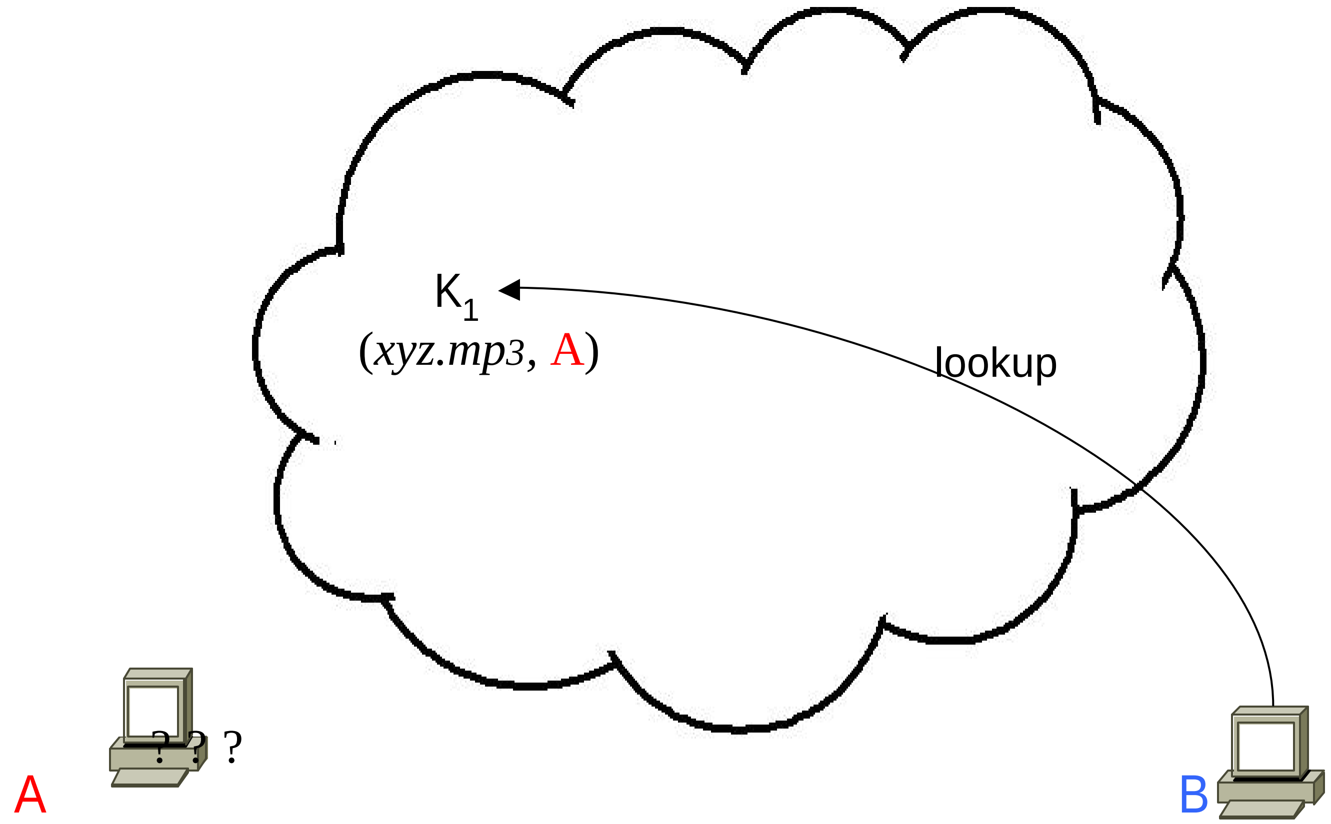
HASH (*xyz.mp3*) = K_1

Indexing



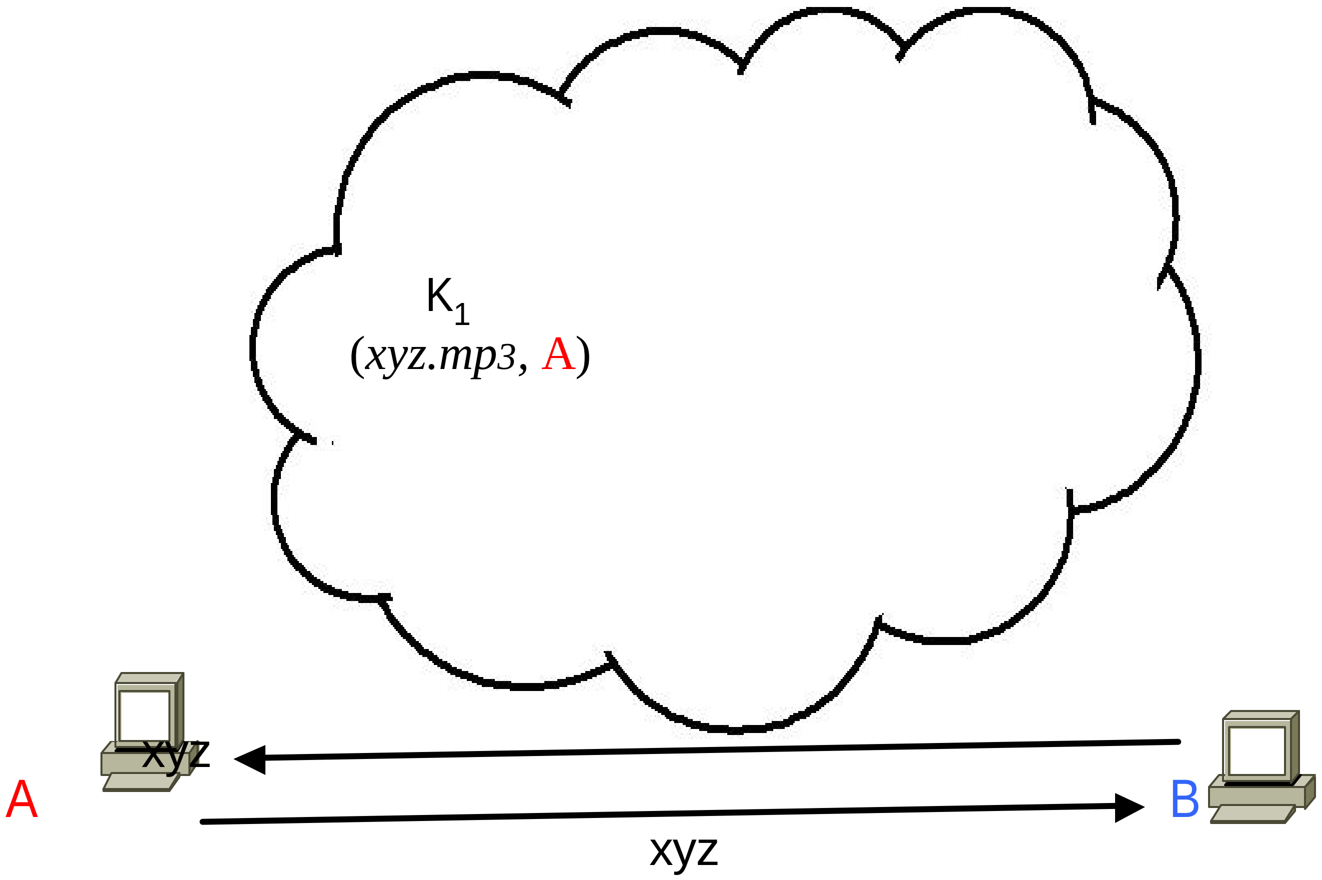
HASH (*xyz.mp3*) = K_1

Indexing

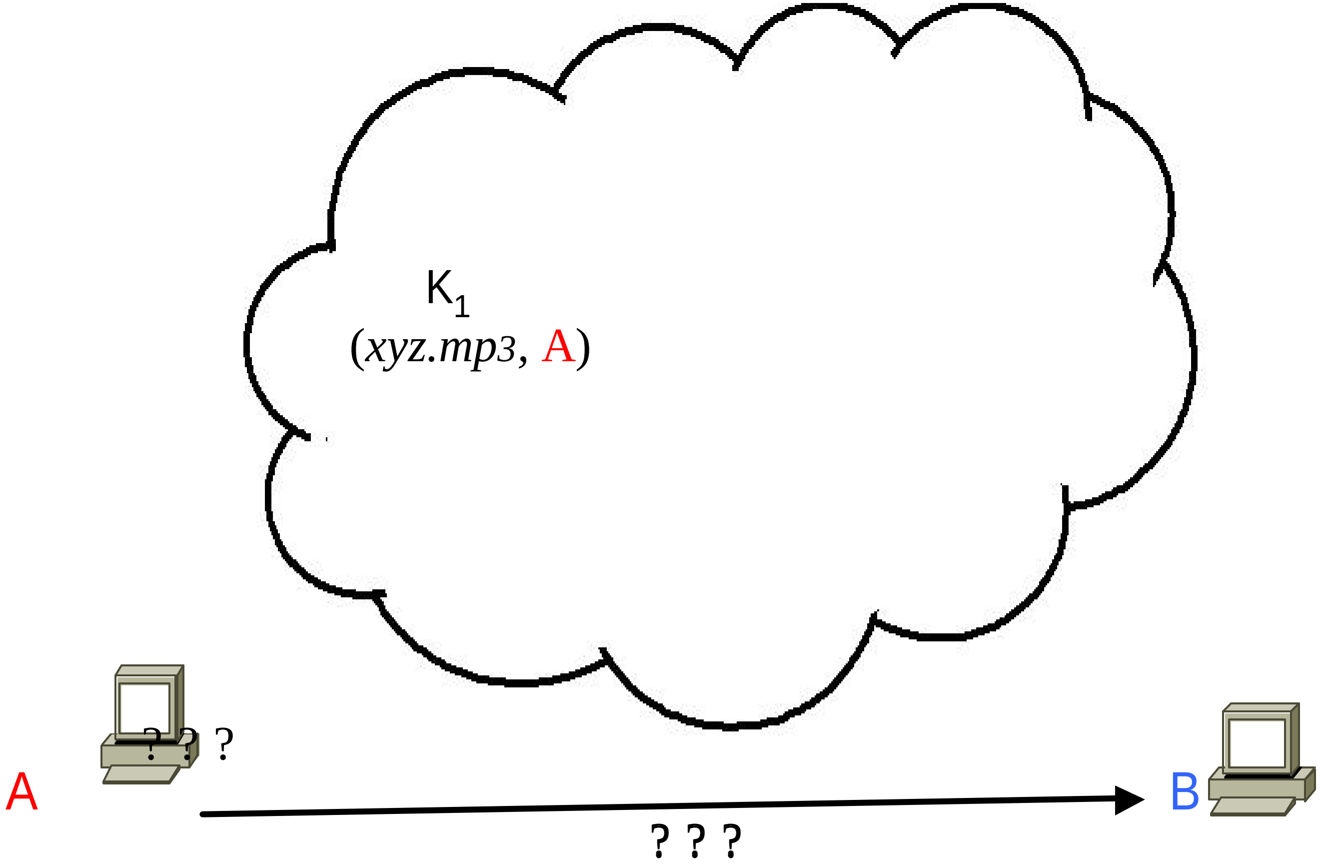


HASH (*xyz.mp3*) = K_1

Indexing

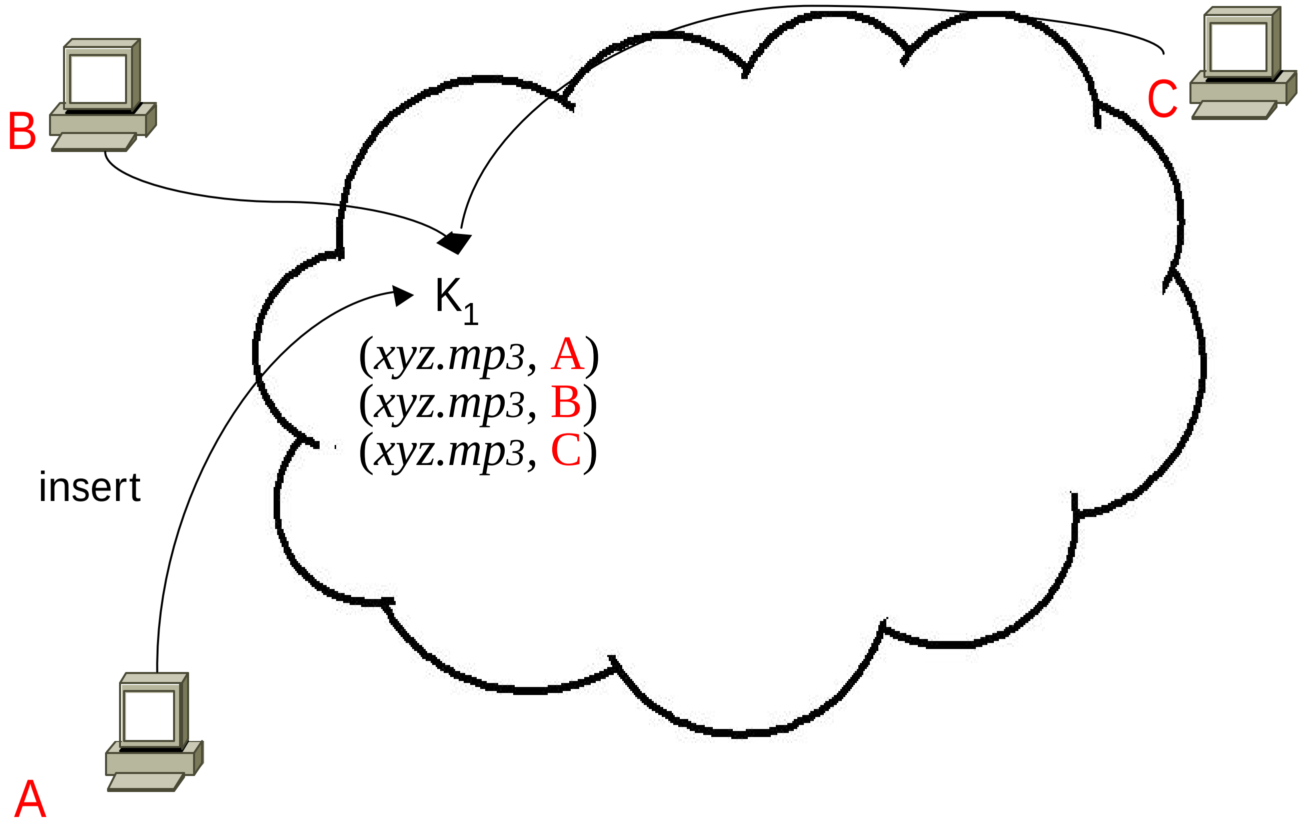


Indexing

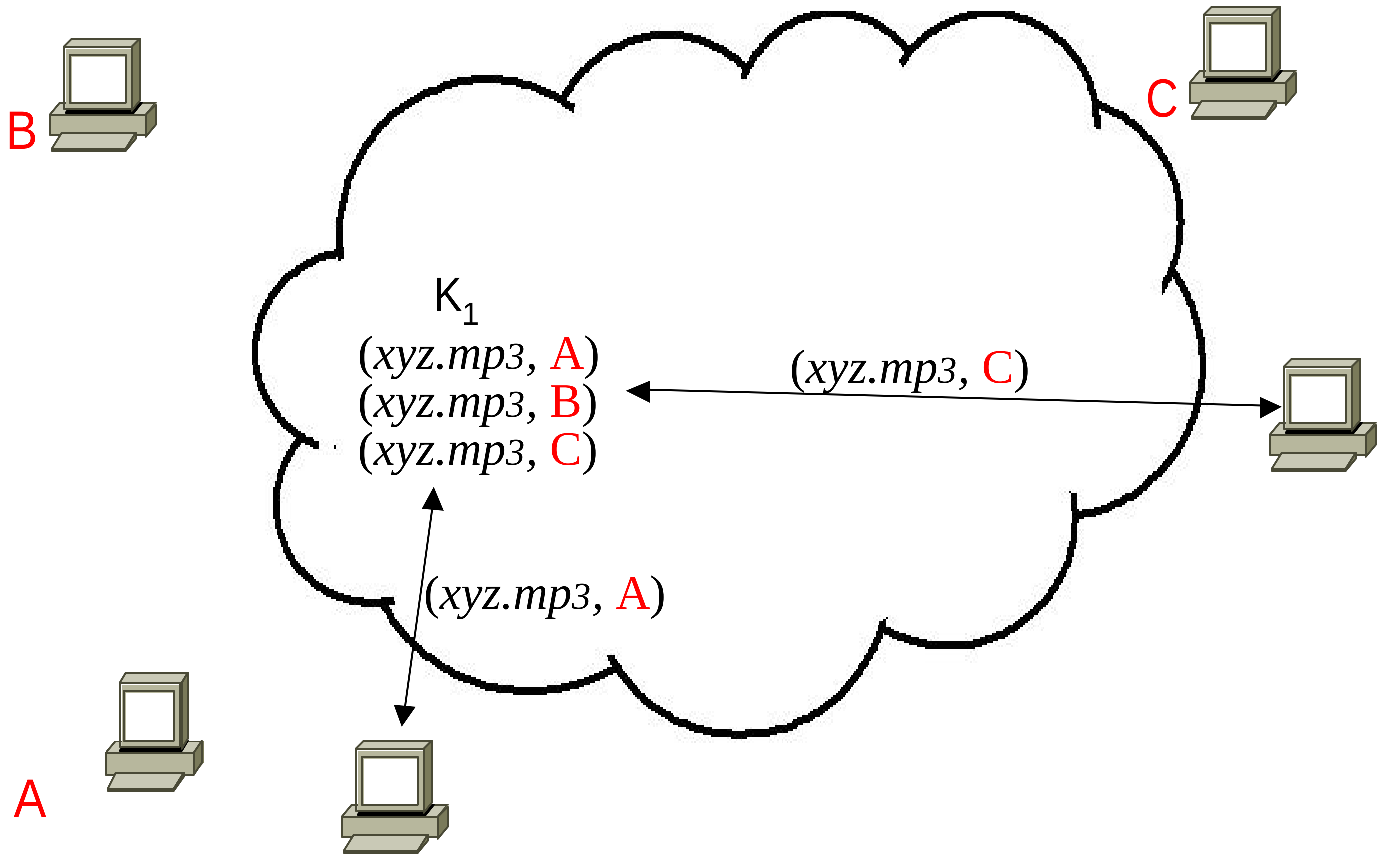


content could as easily have been a web page, disk block, data object, DNS name, ...

Anycast Communication



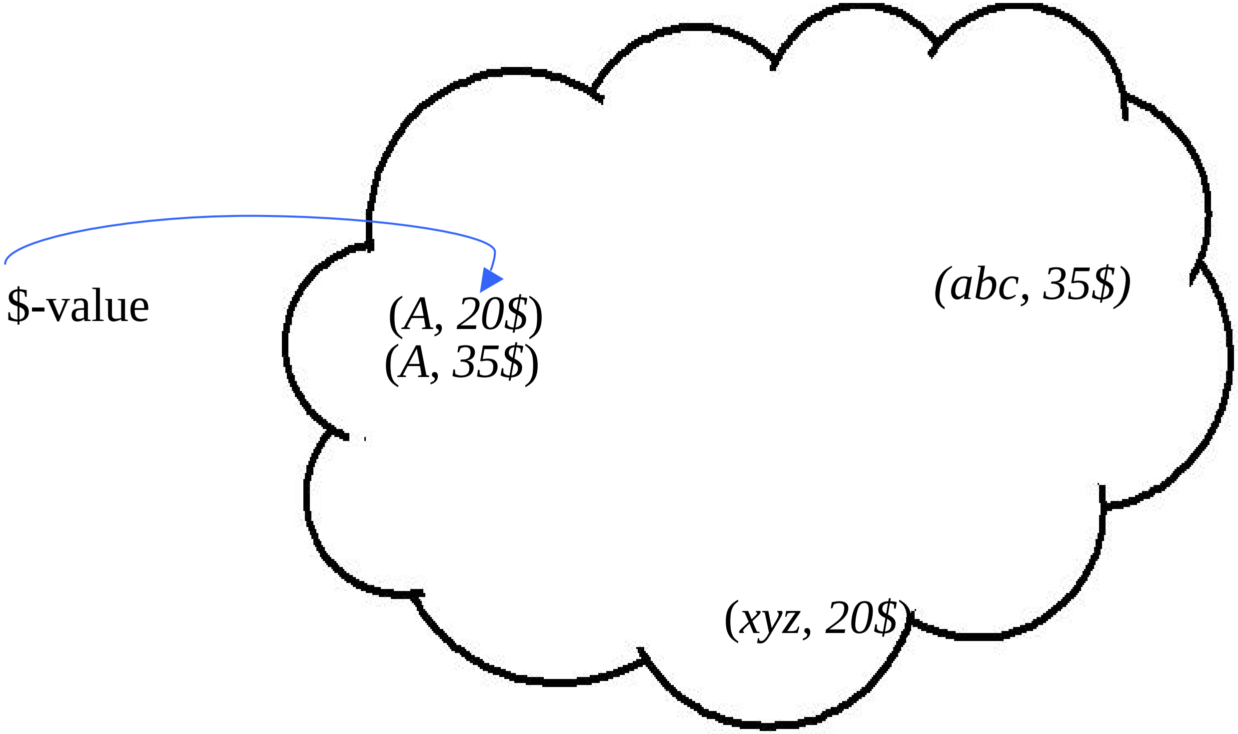
Anycast Communication



"anycast" lookup; based on a number of metrics

Database Join

Join on $-value

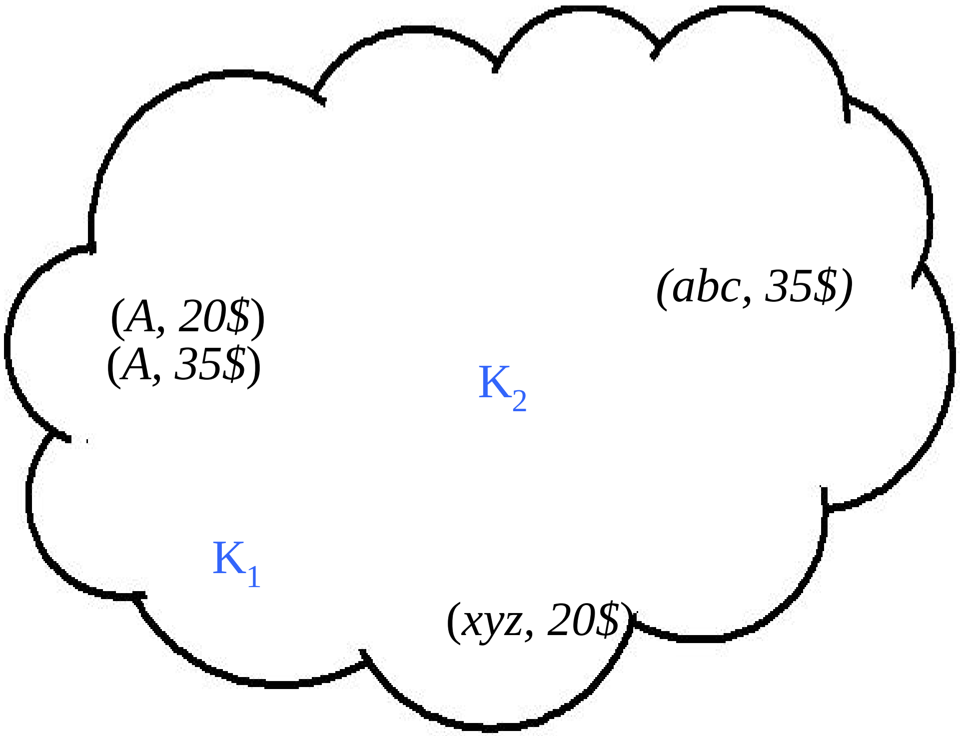


Database Join

Join on $-value

HASH ($20\$$) = K_1

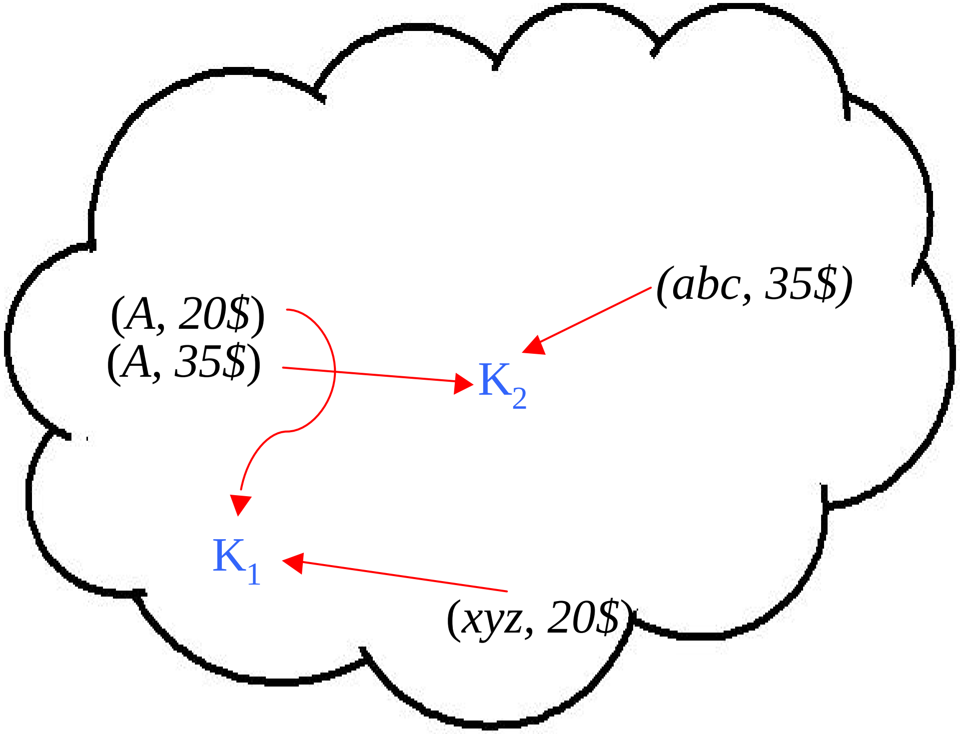
HASH ($35\$$) = K_2



Database Join

Join on $-value

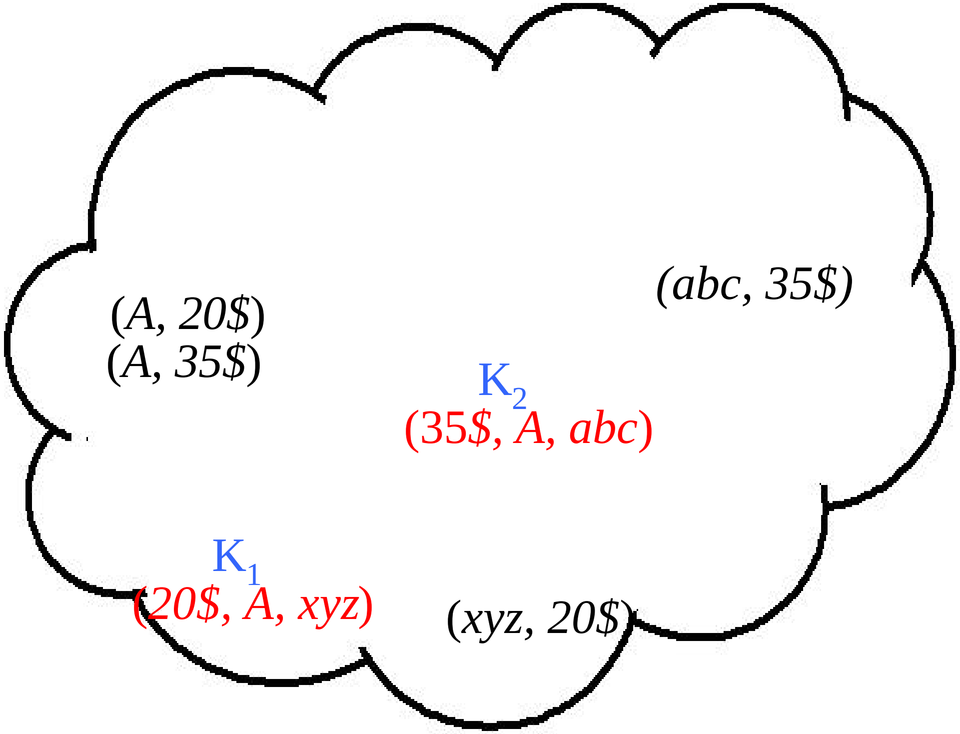
HASH (*20$*) = K_1
HASH (*35$*) = K_2



Database Join

Join on $-value

HASH (*20$*) = K_1
HASH (*35$*) = K_2

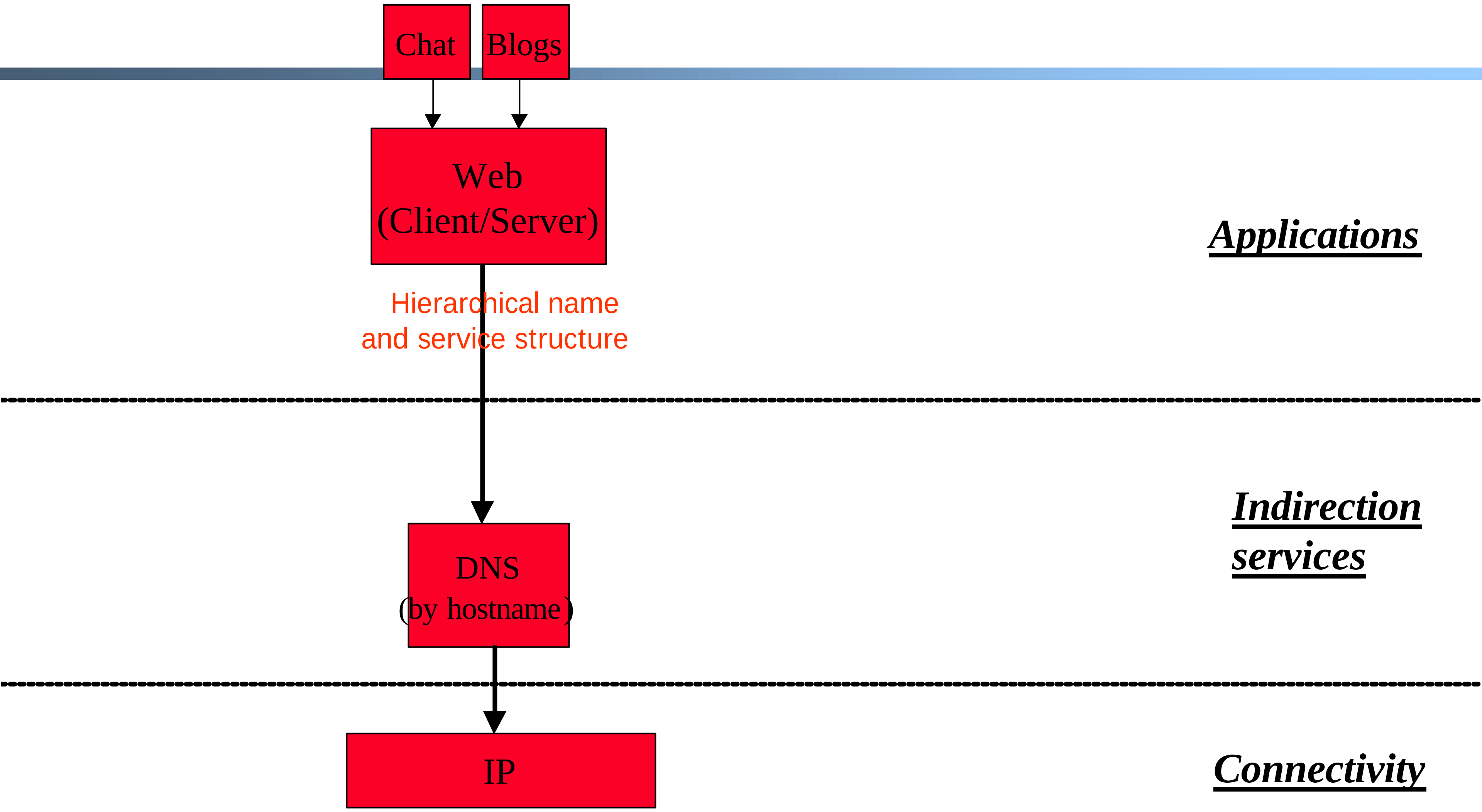


Massively parallel, distributed join on Internet scales!

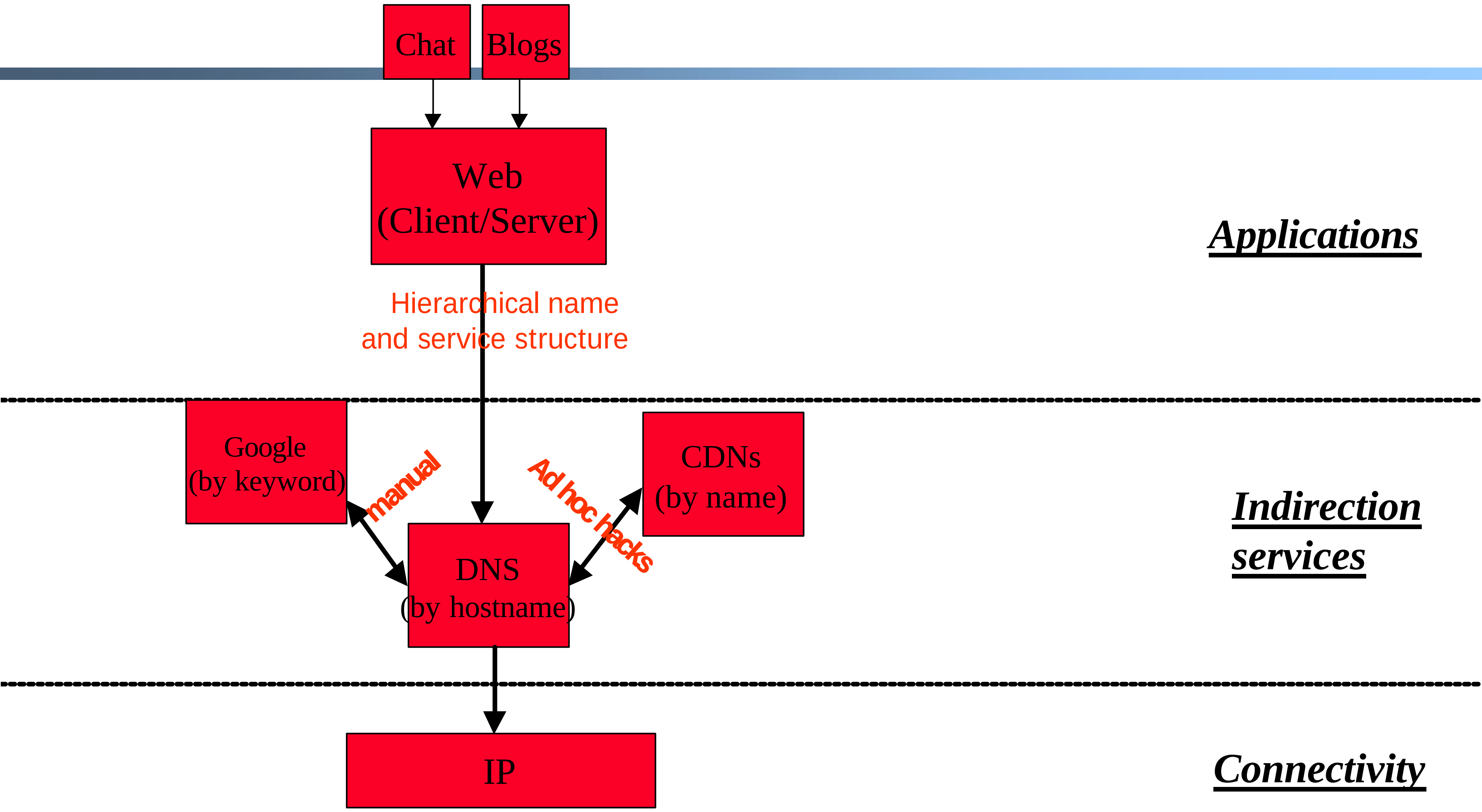
DHTs: Key Insight

- Many uses for DHTs
  - Indexing
  - Multicast, anycast
  - Database joins, sort, range search
  - Service composition
  - Event notification
  - …
- DHT namespace essentially provides a level of indirection
  - *"Any computer systems problem can be solved by adding a level of indirection"*
- How is indirection done today?

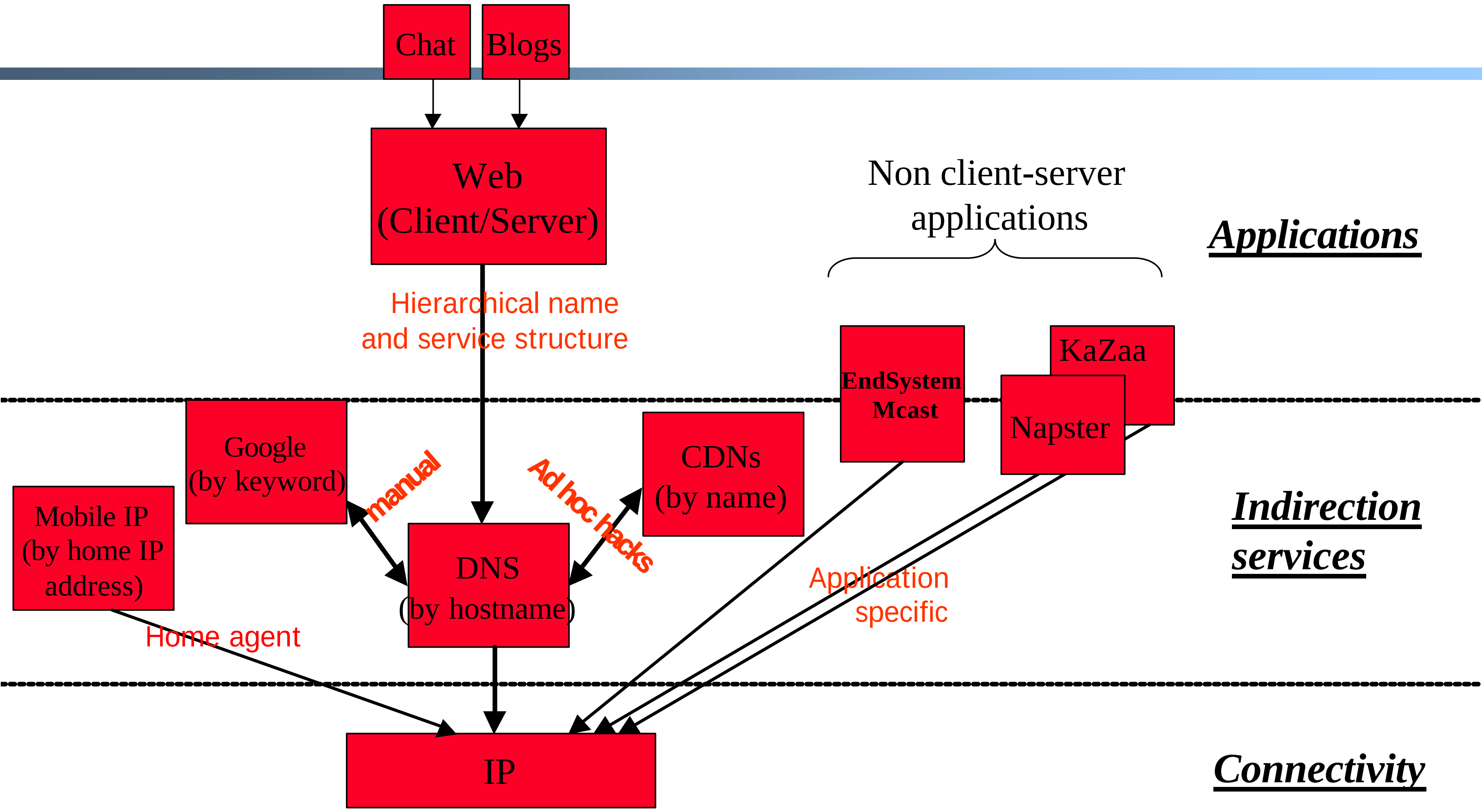
Indirection today



Indirection today



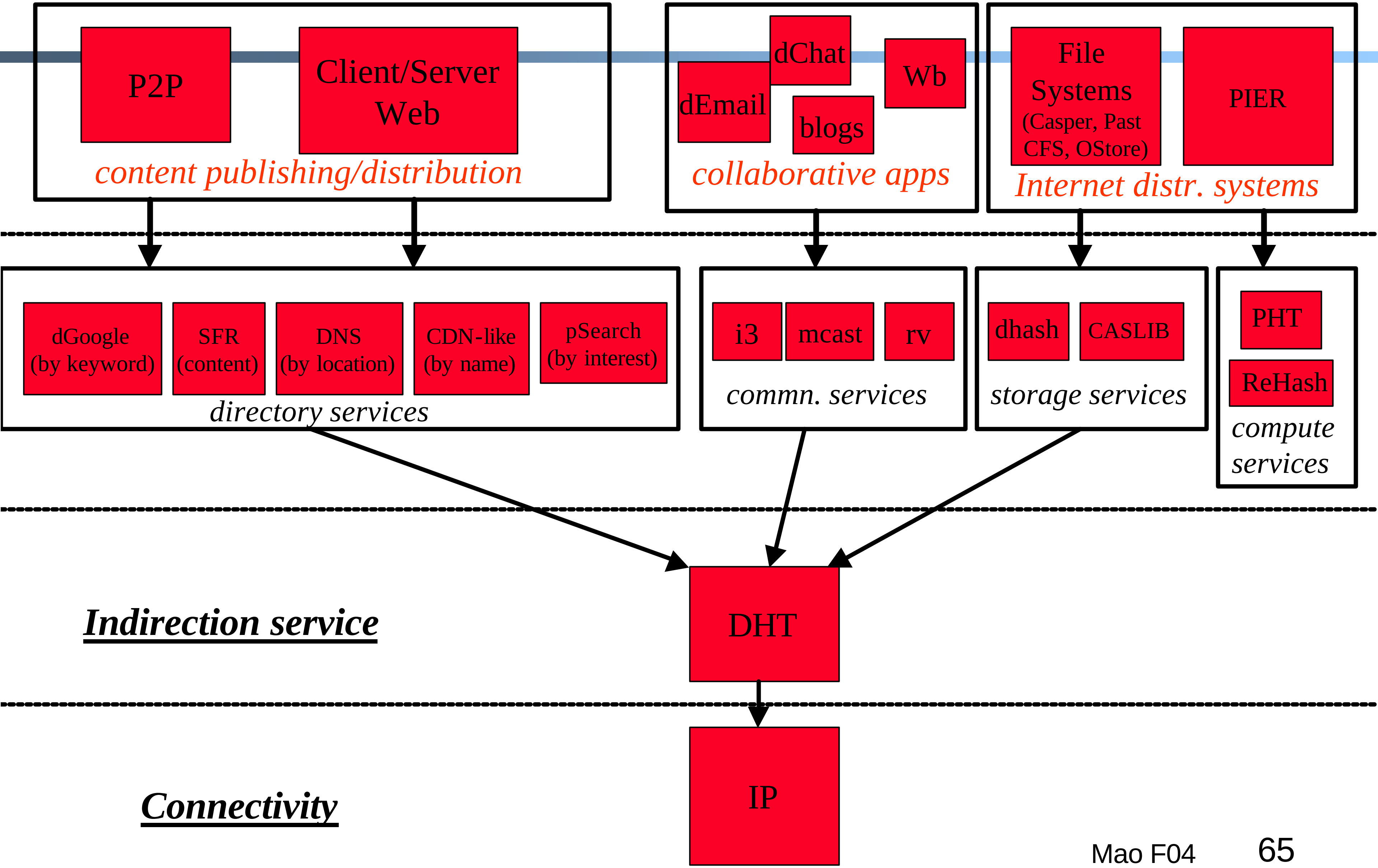
Indirection today



Indirection in Today's Internet

- No explicit interface that applications can build on
  - besides DNS
- Two options
  - Retrofit over the DNS through a variety of creative hacks
  - Customized solution designed/implemented anew for each application

A DHT-enabled Internet



Another Pipe-Dream?

- Will DHTs go the way of QoS, Multicast, etc.?
- Perhaps, but DHTs don't need the cooperation of ISPs, so the barriers to adoption are lower

What You Need to Know

- Napster
- Gnutella
- DHT: basic ideas