Distributed Hash Tables and Chord

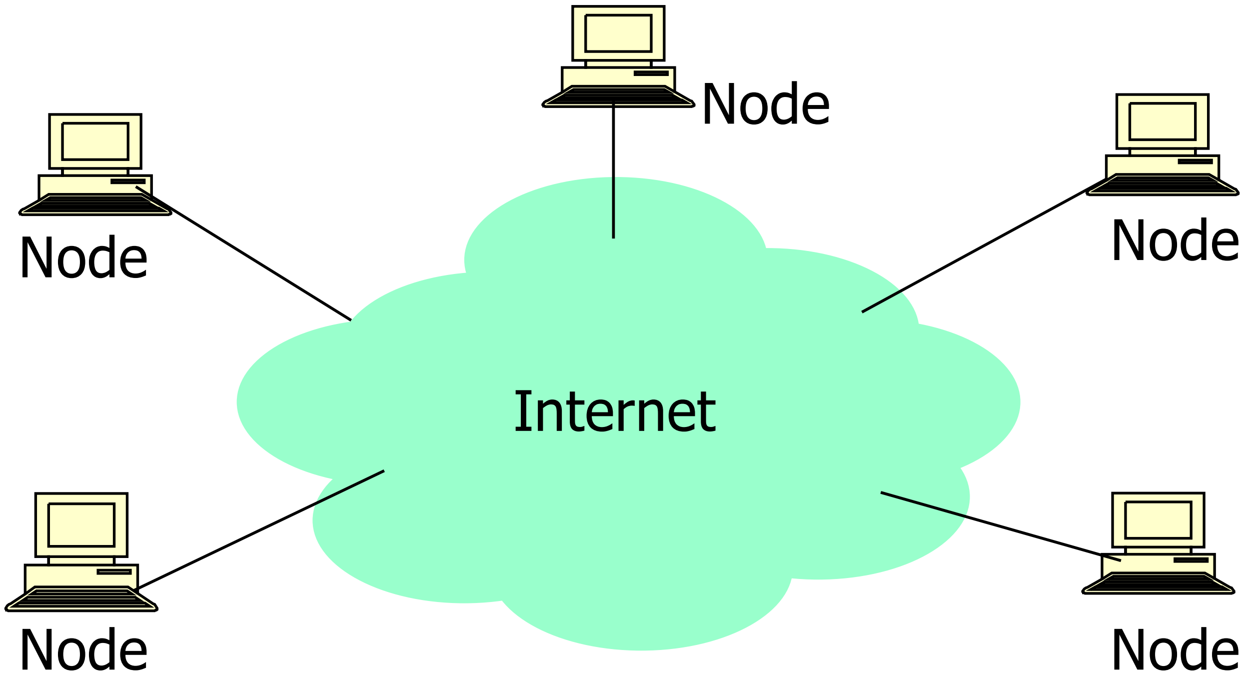
Hari Balakrishnan

6.829 Fall 2018

October 30, 2018

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

What is a P2P system?



- A distributed system architecture in which:
  - There's no centralized control
  - Nodes are symmetric in function
- Large number of (unreliable) nodes

What can P2P teach us about *infrastructure* design?

- Resistant to DoS and failures
  - Safety in numbers, no single point of failure
- Self-assembling
  - Nodes insert themselves into structure
  - No manual configuration or oversight
- Flexible: nodes can be
  - Widely distributed or colocated
  - Powerful hosts or low-end PCs
- Each peer brings a little bit to the dance
  - Aggregate is equivalent to a big distributed server farm behind a fat network pipe

General Abstraction?

- Big challenge for P2P: finding content
  - Many machines, must find one that holds data
  - Not too hard to find "hay", but what about "needles"?
- Essential task: lookup(key)
  - Given key, find host that has data ("value") corresponding to that key
- Higher-level interface: put(key,val)/get(key)
  - Easy to layer on top of lookup()
  - Allows application to ignore details of storage
  - Good for some apps, not for others

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Data-centric network abstraction

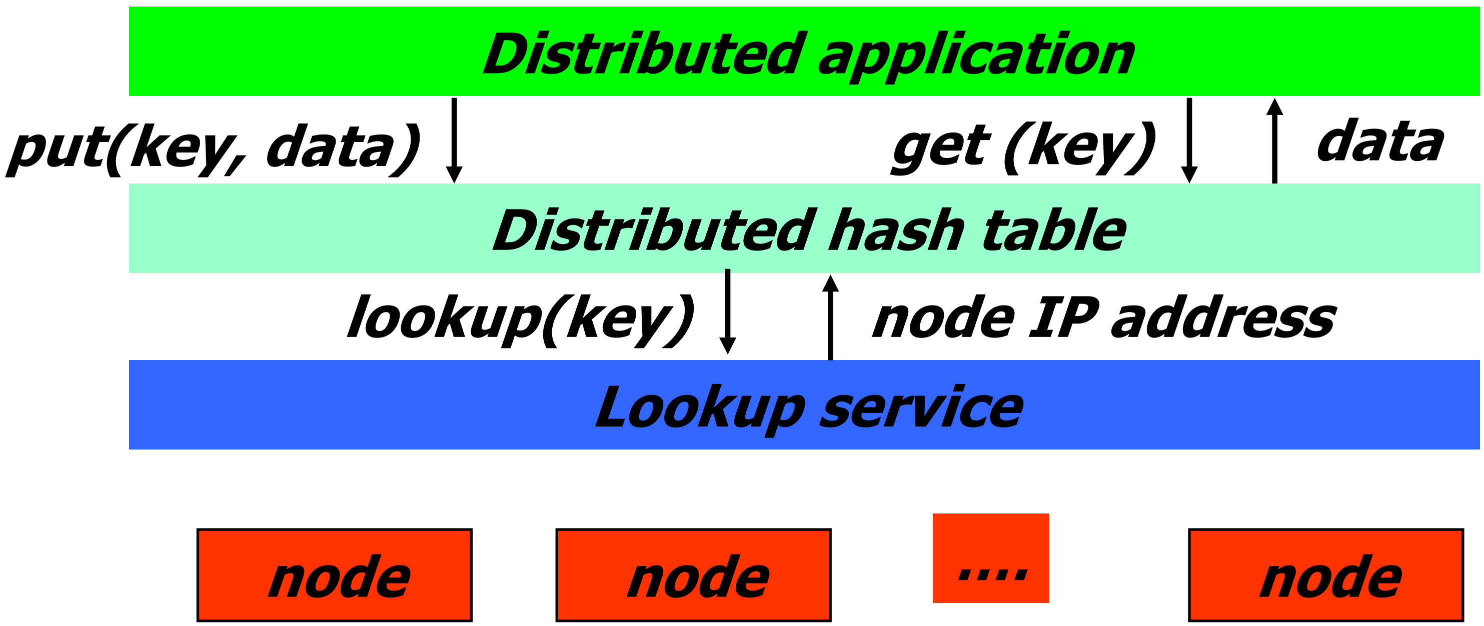
- TCP provides a “conversation” abstraction

  socket = connect (IP address, port);

  send(data on socket); /* goes to IP addr / TCP port */
- A DHT provides a “data-centric” abstraction as an overlay over the Internet
  - A key is a semantic-free identifier for data
  - E.g., key = hash(filename)

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

DHT layering



- Application may be distributed over many nodes
- DHT distributes the key-value data store over many nodes
- Many applications can use the same DHT infrastructure

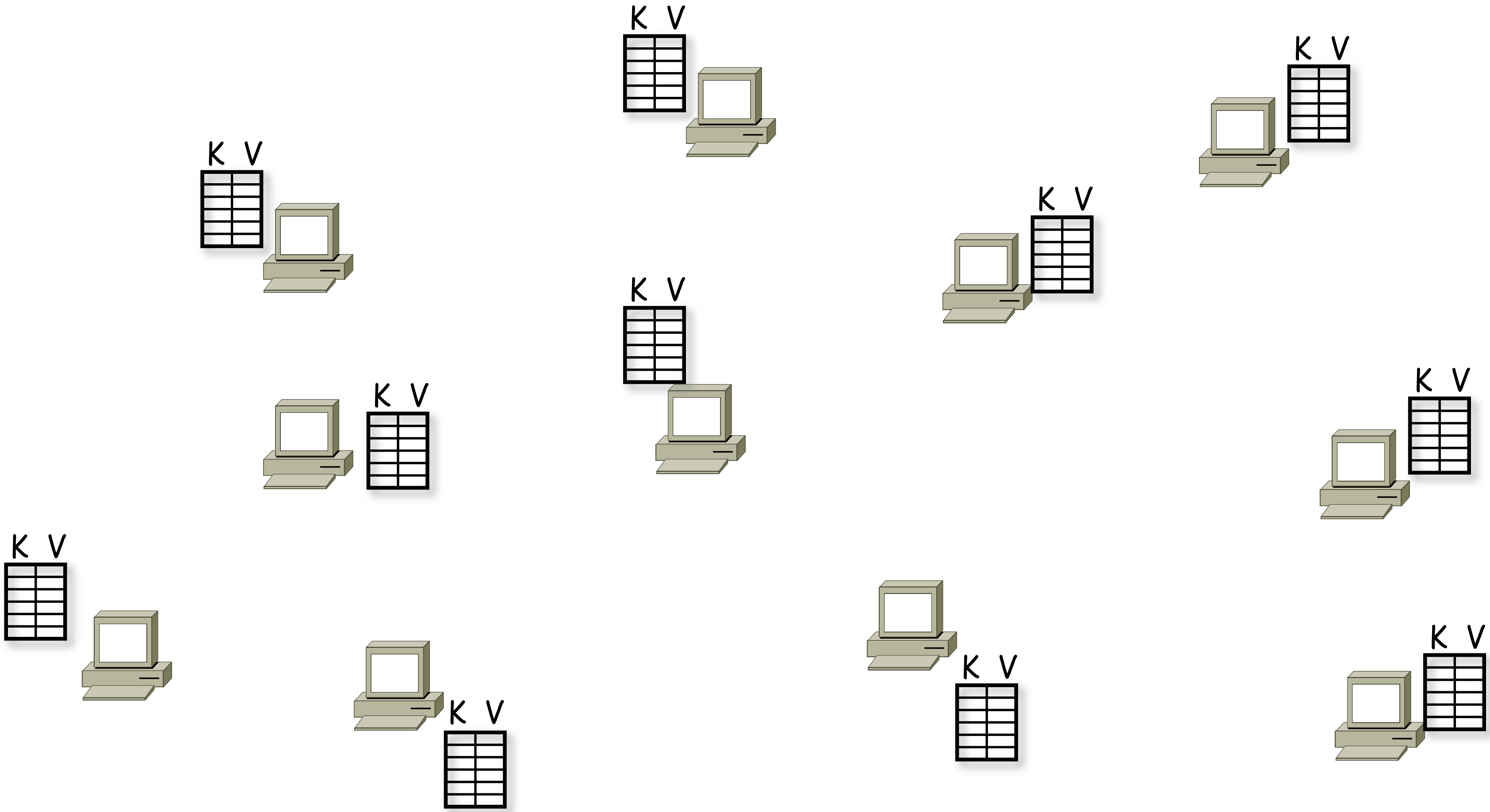
Virtues of DHT Interface

- Simple and useful
- put/get API supports wide range of apps
  - No structure/meaning imposed on keys
  - Scalable, flat name space
  - Location-independent names → easy to replicate and move keys (content)
- Key/value pairs are persistent and global
  - Can store other keys (or other names or IP addresses) in DHT values
  - And thus build complex data structures

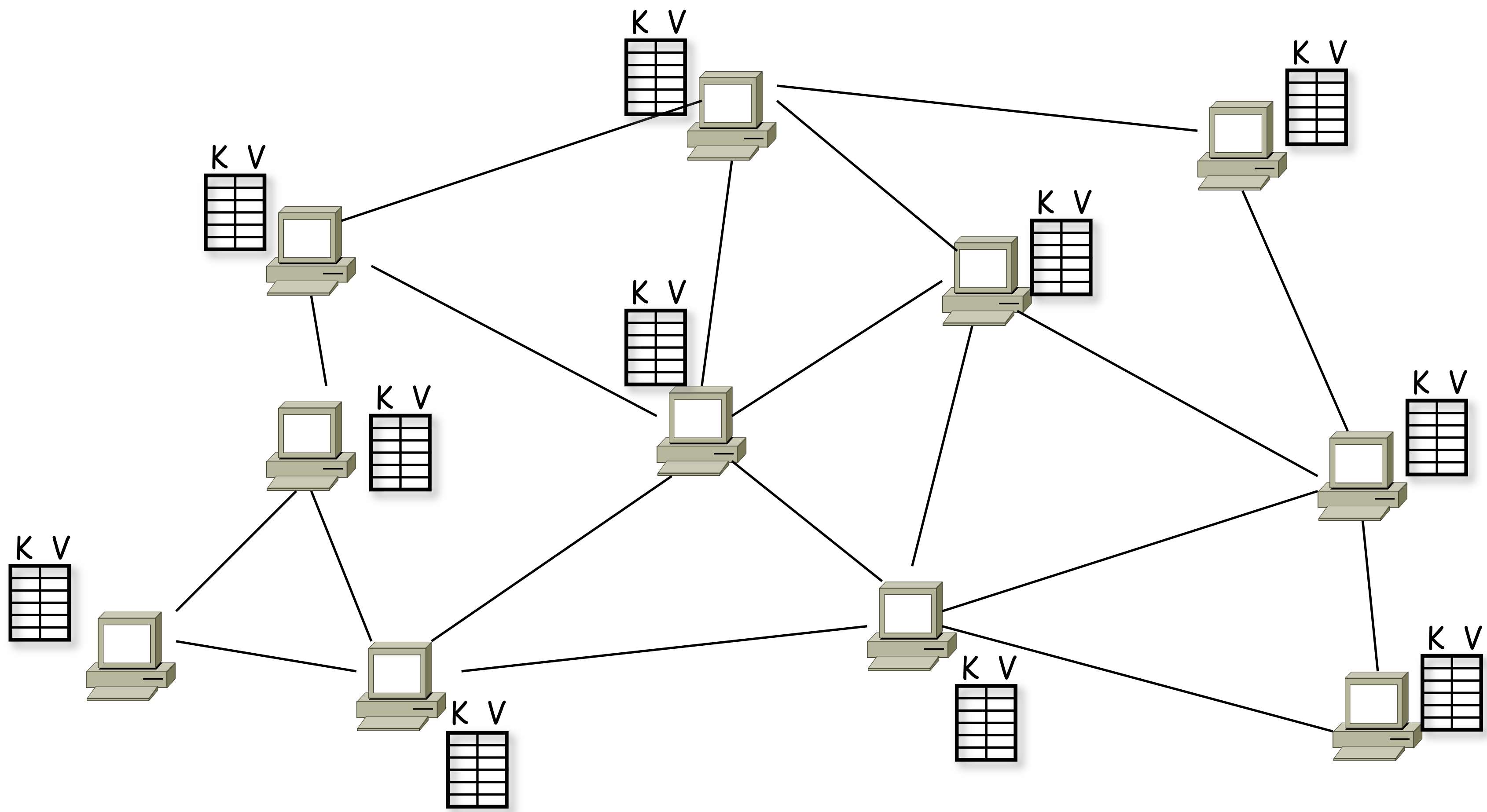
Some DHT applications

- Storage systems
  - Persistent backup store ("P2P backup")
  - Read/Write file systems
  - Cooperative source code repository
- Content distribution
  - "Grassroots" Web replication & content distribution
  - Robust netnews (Usenet)
  - Resilient Web links, untangling the Web from DNS
  - Web archiver with timeline
- Communication
  - Handling mobility, multicast, indirection
  - Email spam control
  - Better firewalls and coping with NATs
  - Various naming systems
- Distributed database query processing; event notification

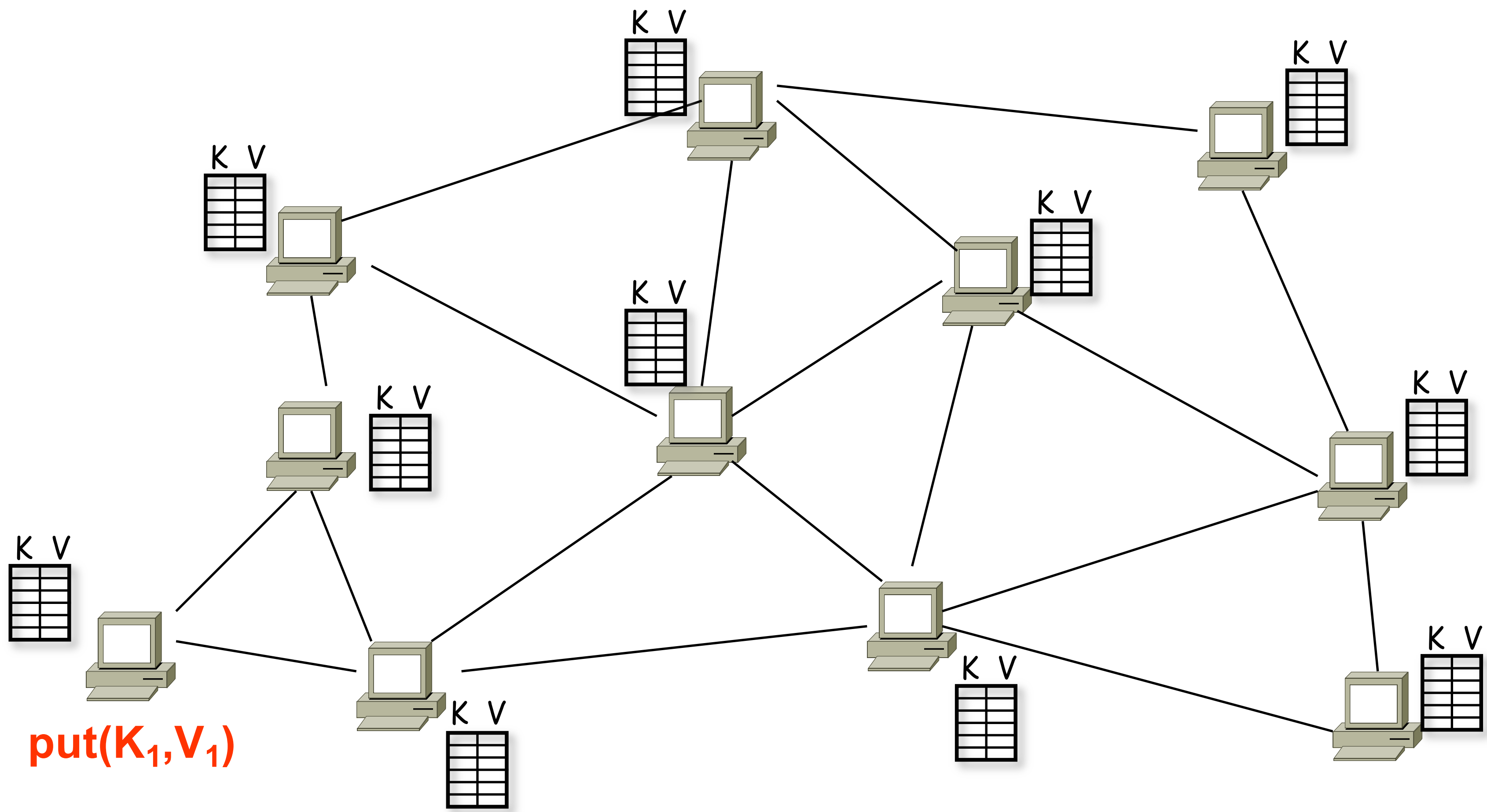
A DHT in Operation: Peers



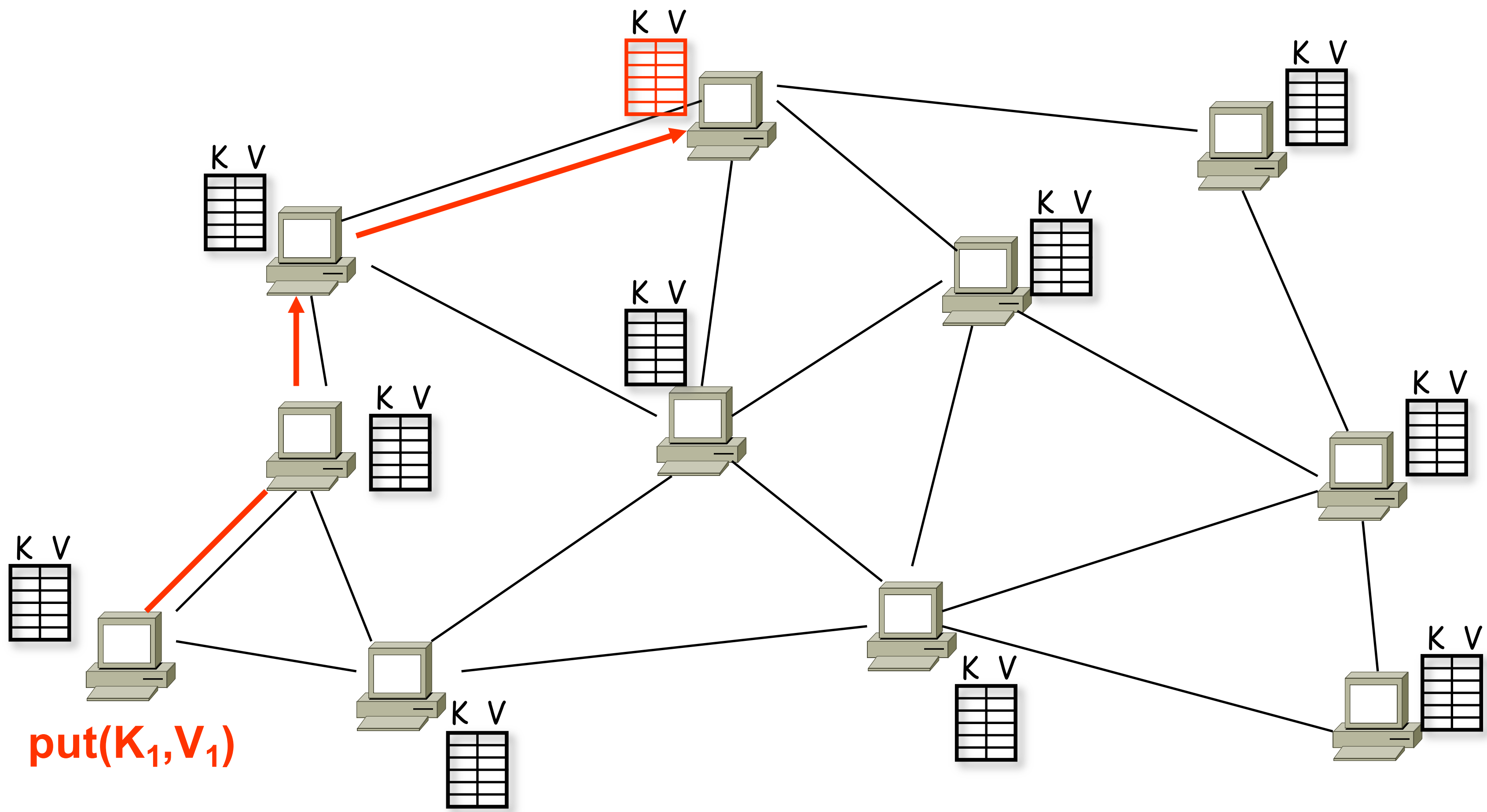
A DHT in Operation: Overlay



A DHT in Operation: put()

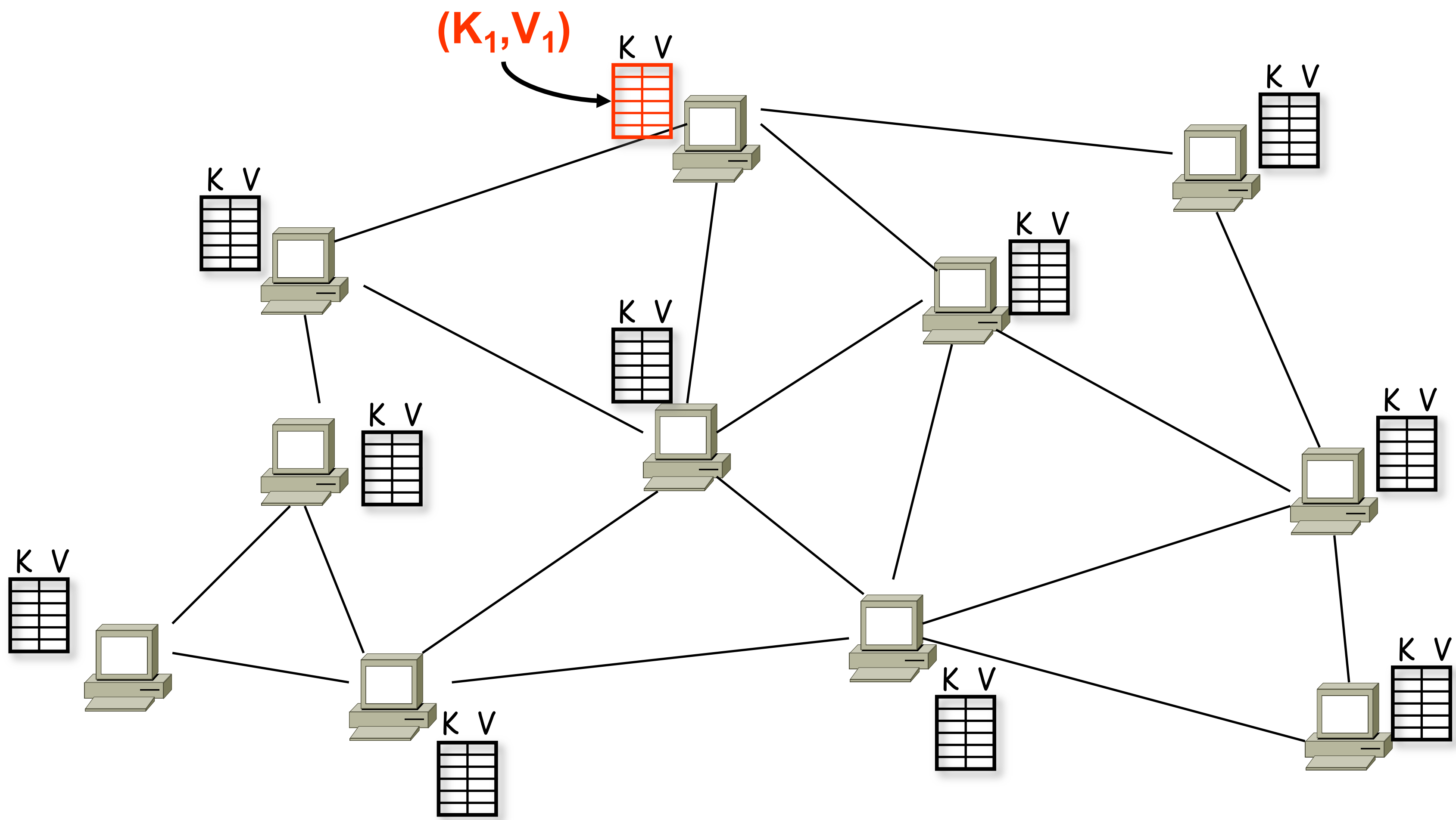


A DHT in Operation: put()

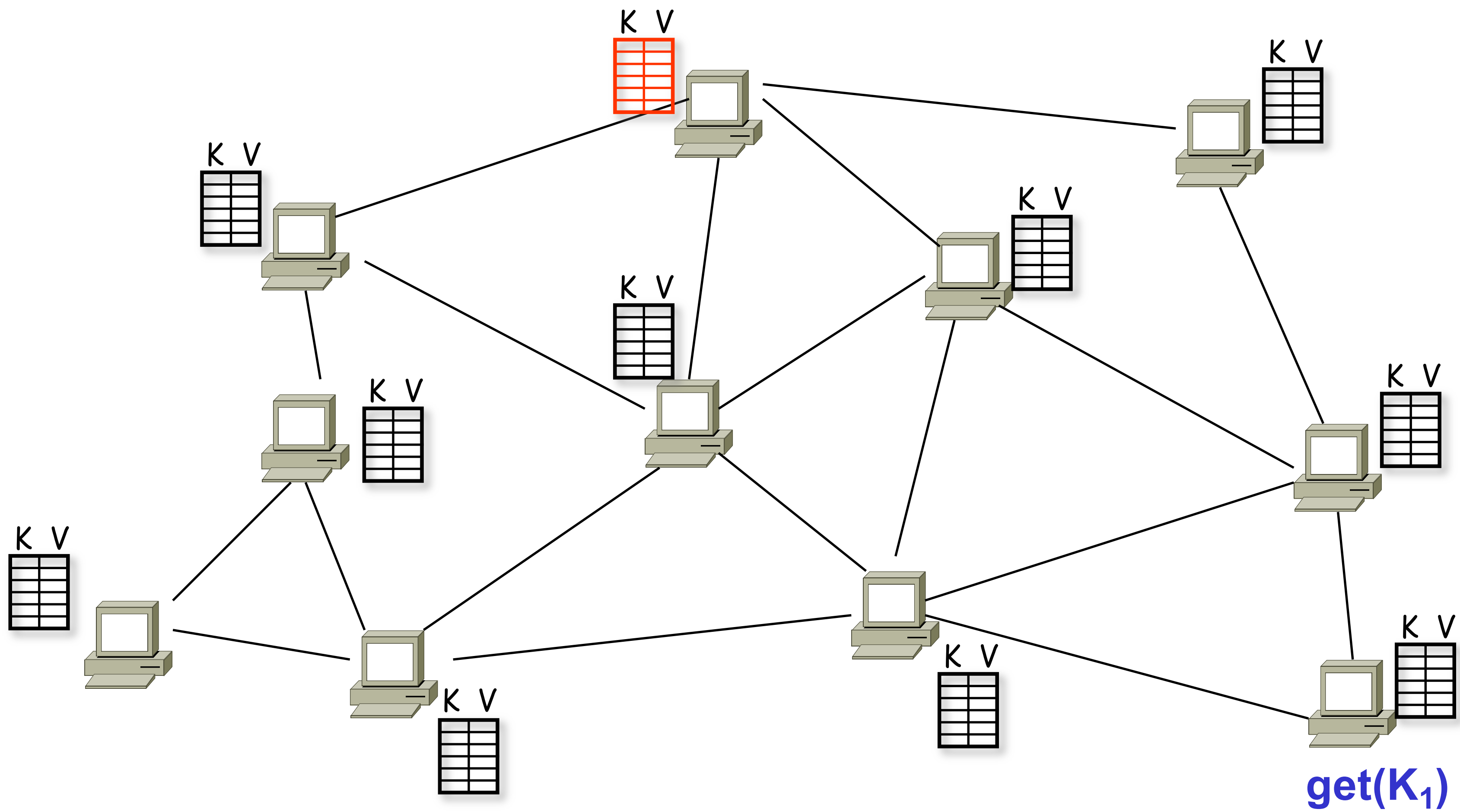


MASSACHUSETTS INSTITUTE OF TECHNOLOGY

A DHT in Operation: put()

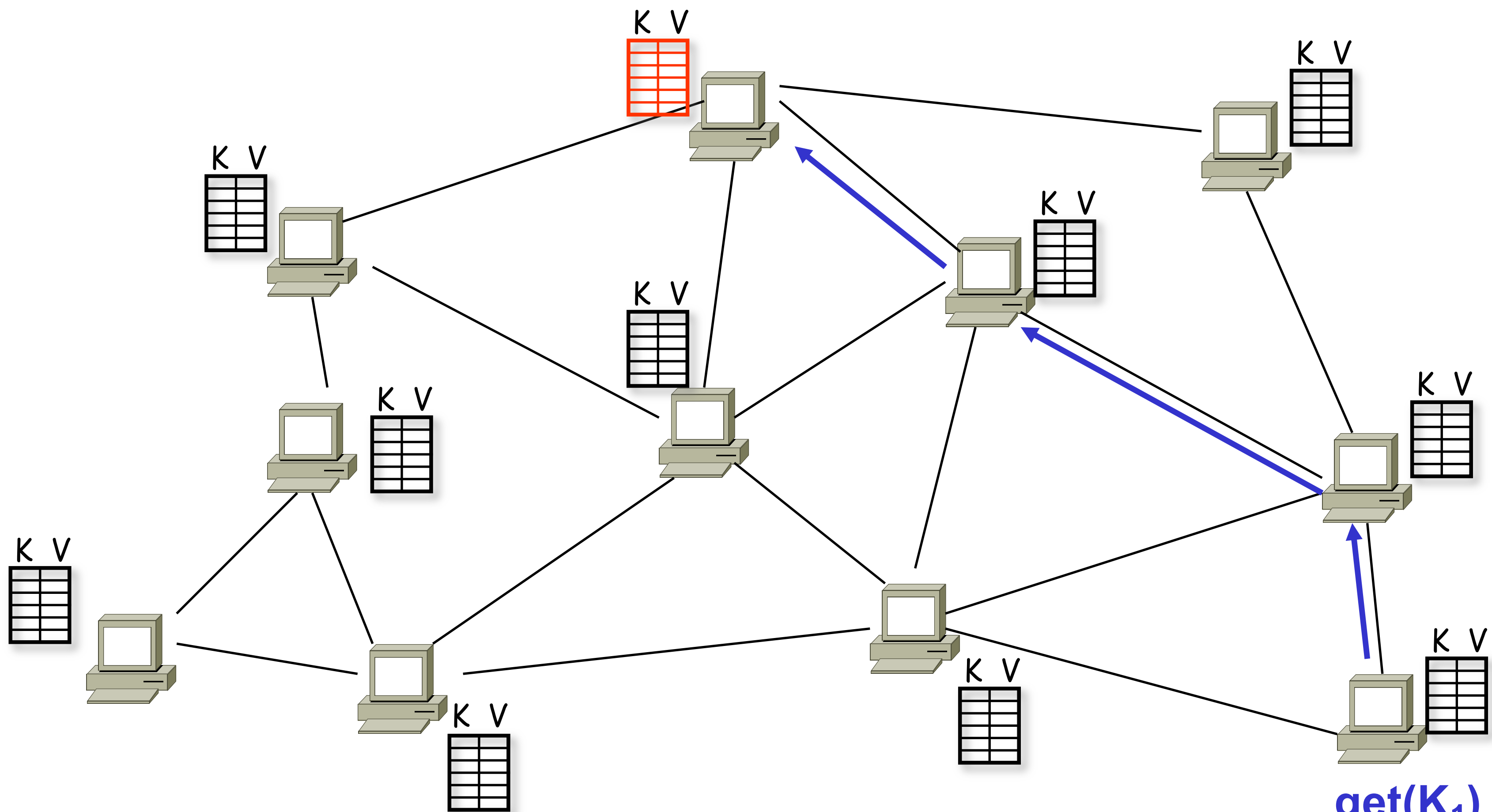


A DHT in Operation: get()



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

A DHT in Operation: get()



Designing a good lookup algorithm

- Map every conceivable key identifier to some machine in the network
  - Store key-value on that machine
  - Update mapping/storage as items and machines come and go
- Note: User does not choose key location
  - Not really restrictive: key in DHT can be a pointer

Requirements

- Load balance
  - Want responsibility for keys spread "evenly" among nodes
- Low maintenance overhead
  - As nodes come and go
- Efficient lookup of key to machine
  - Fast response
  - Little computation/bandwidth (no flooding queries)
- Fault tolerance to sudden node failures

Consequences

- As nodes come and go, costs too much bandwidth to notify everyone immediately
- So, nodes only aware of some subset of DHT: their **neighbors**
- In particular, home node for key might not be a neighbor
- So, must find right node through a sequence of **routing hops**, asking neighbors about their neighbors…

Maintenance

- As nodes come and go, maintain set of neighbors for each machine
  - Keep neighbor sets small for reduced overhead
  - Low degree
- Maintain routing tables to traverse neighbor graph
  - Keep number of hops small for fast resolution
  - Low diameter

Degree-Diameter Tradeoff

- Suppose machine degree d
  - Each neighbor knows d nodes, giving d^2 at distance 2
  - Up to distance h, can reach $1+d^2+d^3\ldots+d^h \sim d^h$
- If n nodes, need $d^h > n$ to reach all nodes
  - Therefore, $h > \log_d n$
- Consequences:
  - For h = 2 (two-hop lookup), need d > $\sqrt{n}$
  - With degree $d = 2$, get $h=\log_2 n$

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Tradeoffs

- With larger degree, we can hope to achieve
  - Smaller diameter
  - Better fault tolerance
- But higher degree implies
  - More neighbor-table state per node
  - Higher maintenance overhead to keep neighbor tables up to date

Routing

- Low diameter is good, but not enough
- Item may be close: But how to find it?
- Need routing rules:
  - Way to assign each item to specific machine
  - Way to find that node by traversing (few) routing hops

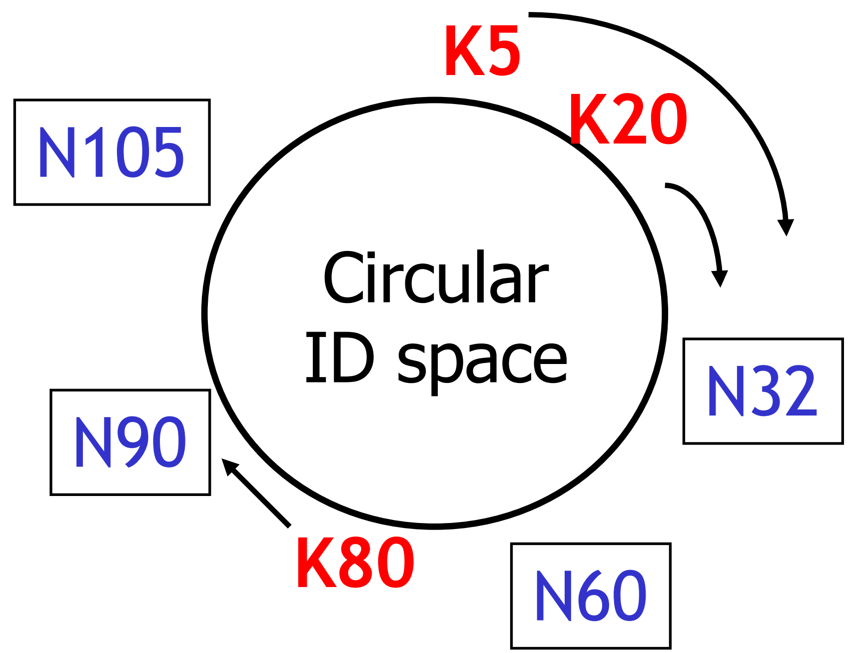
Routing by Imaginary Namespace Geography

- Common principle in all DHT designs
- Map all (conceivable) keys into some abstract geographic space
- Place machines in same space
- Assignment: key goes to “closest” node
- Routing: guarantee that any node that is not the destination has some neighbor “closer” to the destination
  - Route by repeatedly getting closer to destination

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

The Chord algorithm

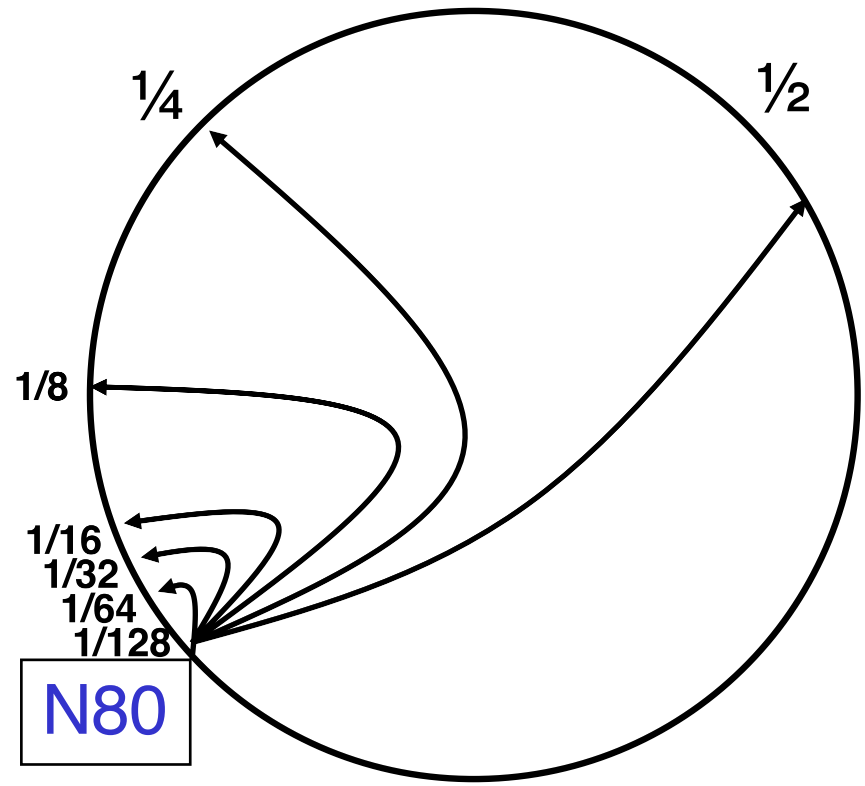
- Each node has 160-bit ID
- ID space is circular
- Data keys are also IDs
- A key is stored on the next higher node
- Good load balance
- *Consistent hashing*
- Easy to find keys slowly by following chain of successors



(N90 is responsible for keys K61 through K90)

Fast routing with a small routing table

- Each node's routing table lists nodes:
  - ½ way around circle
  - ¼ way around circle
  - …
  - next around circle
- The table is small:
  - At most log N entries

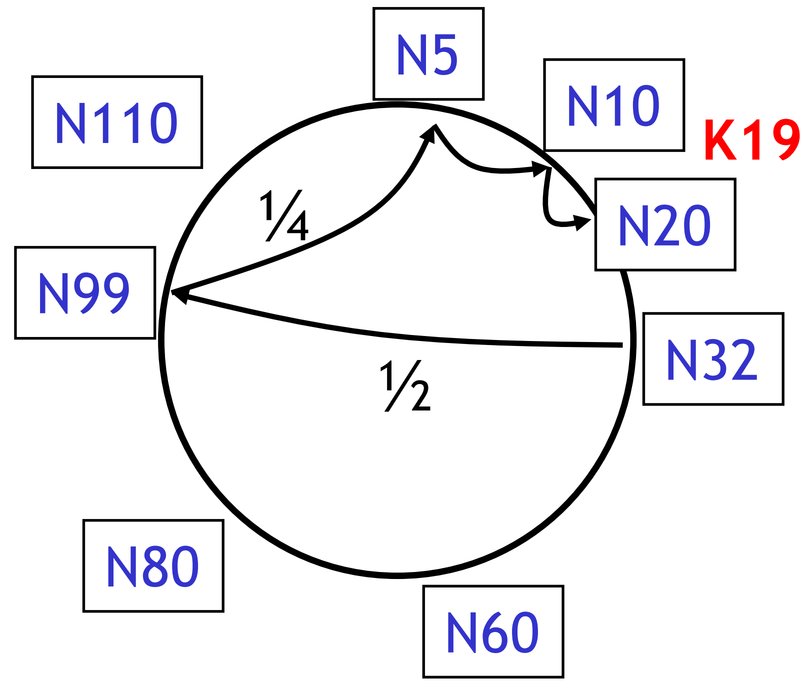


MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Chord lookups take O(log N) hops

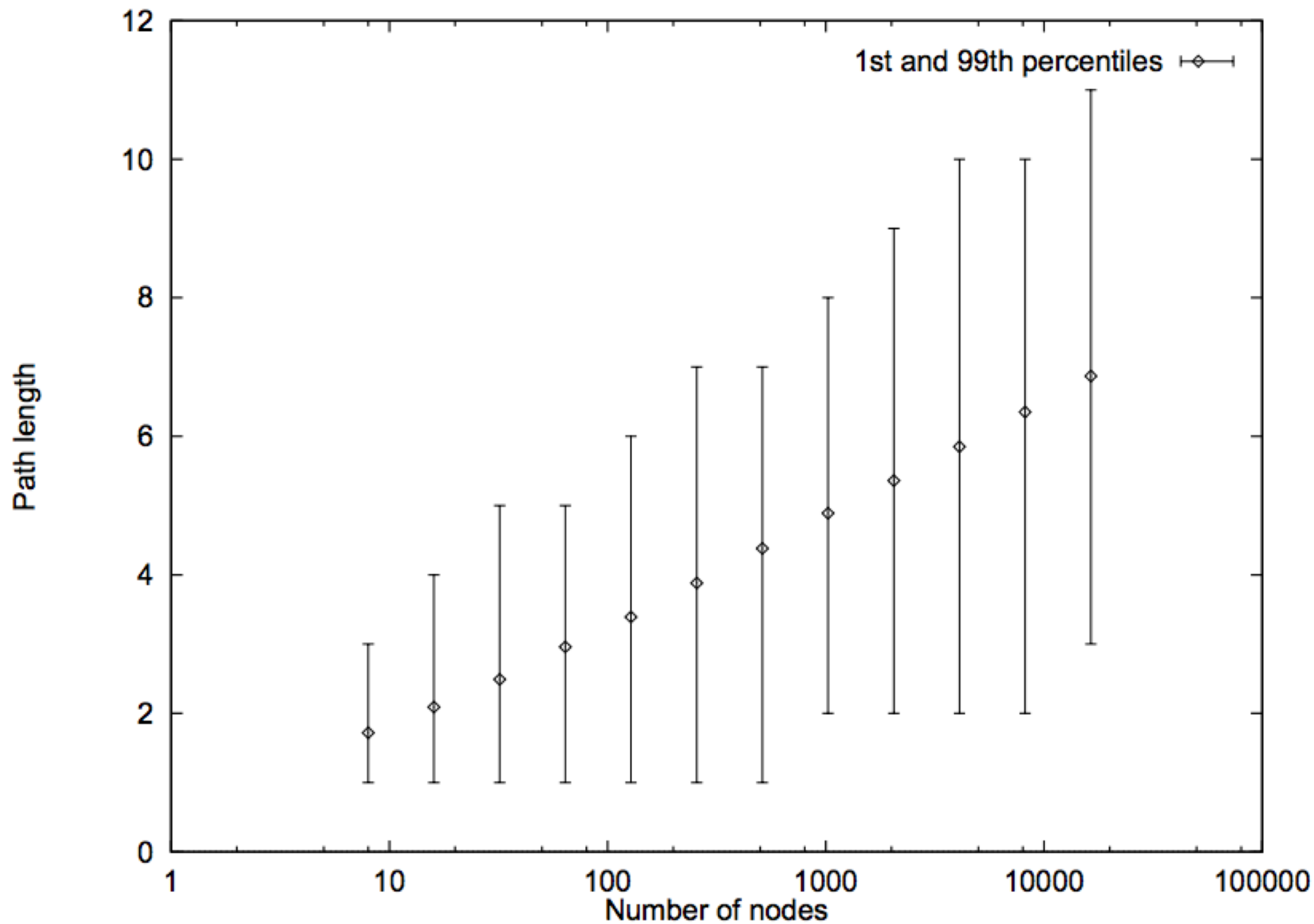
- Every step reduces the remaining distance to the destination by at least a factor of 2

- Lookups are fast:
  - At most O(log *N*) steps
  - Can be made even faster in practice



Node N32 looks up key K19

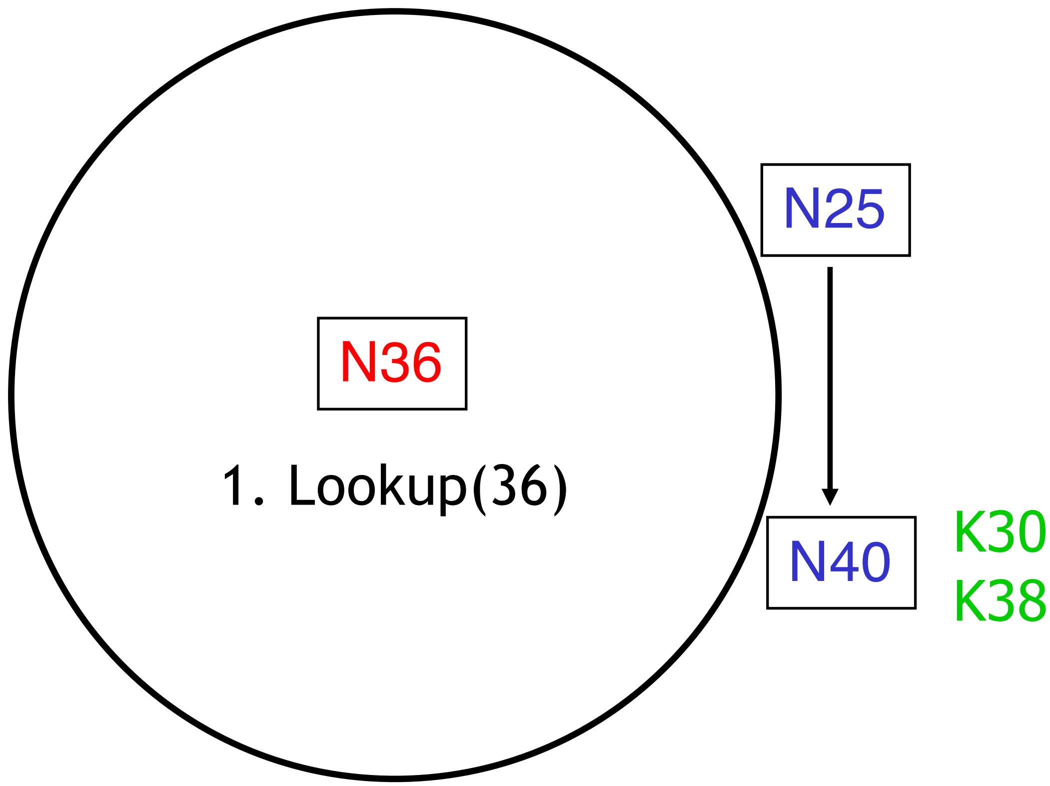
Lookups: ½ log N steps



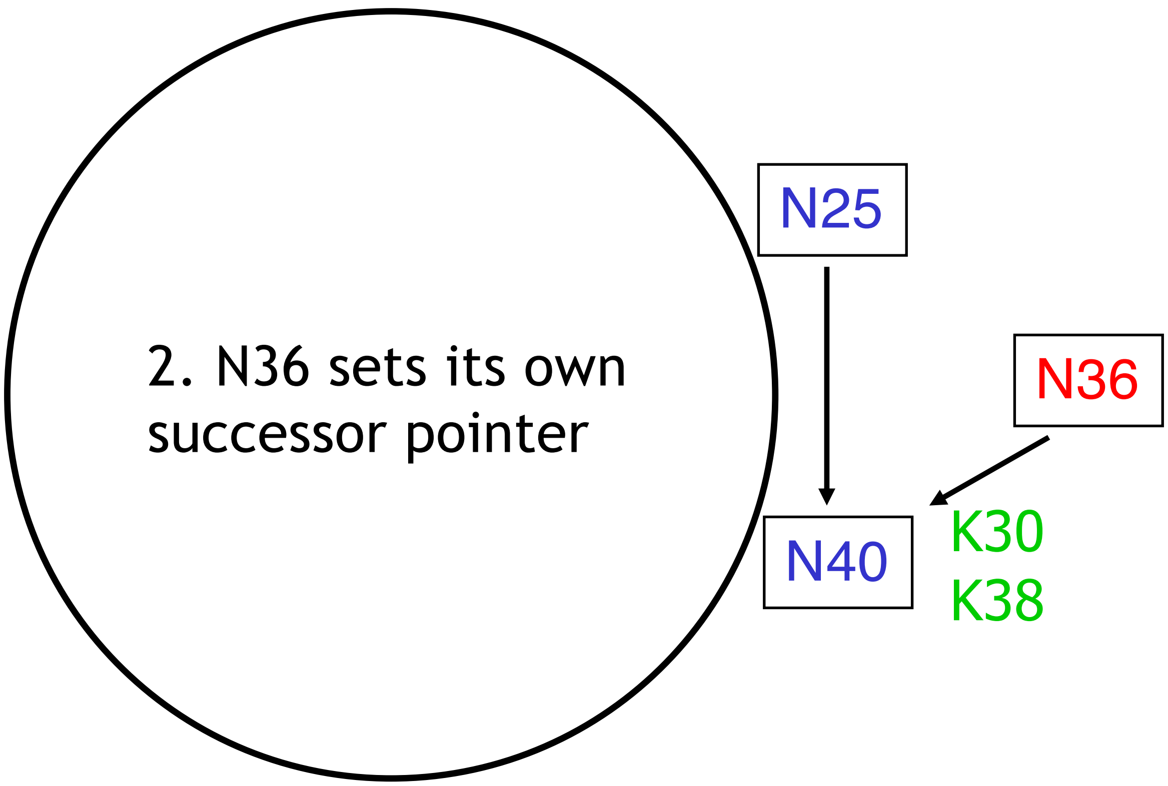
(a)

Why ½?

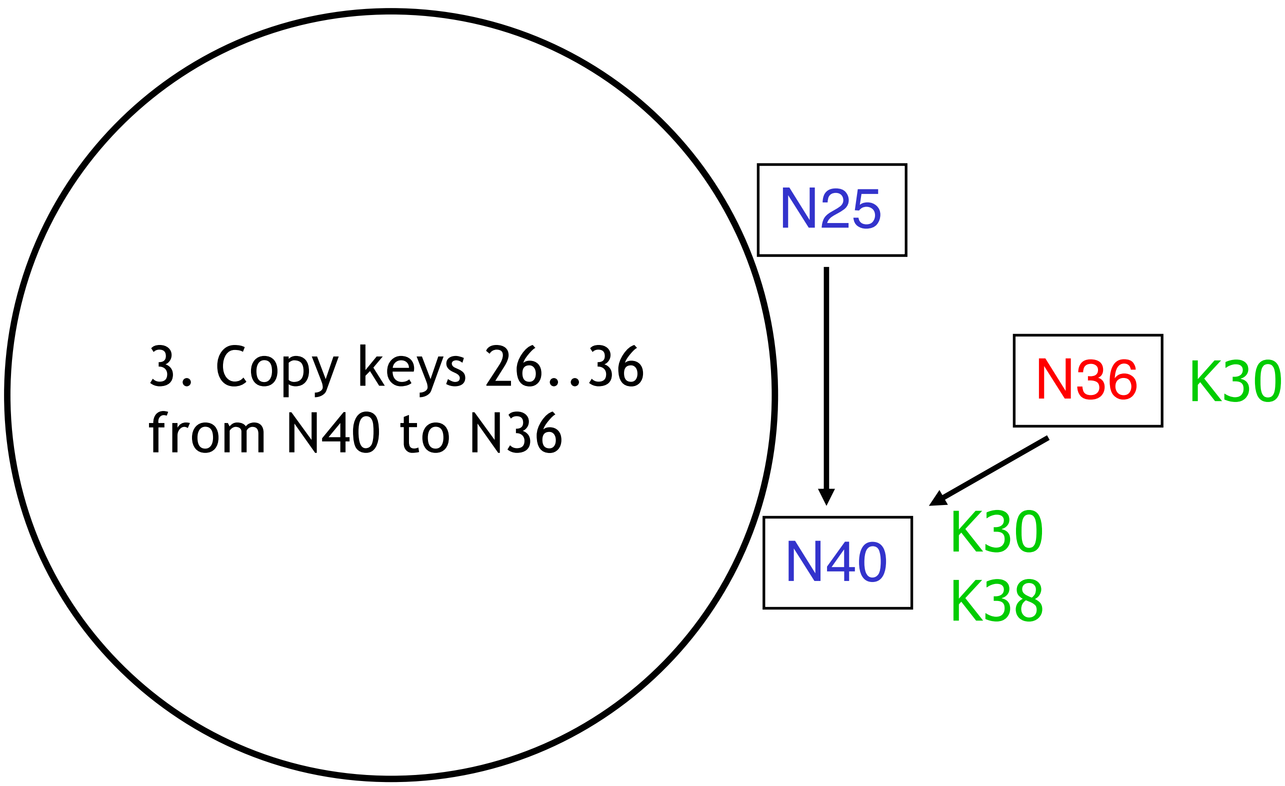
Joining: linked list insert



Join (2)

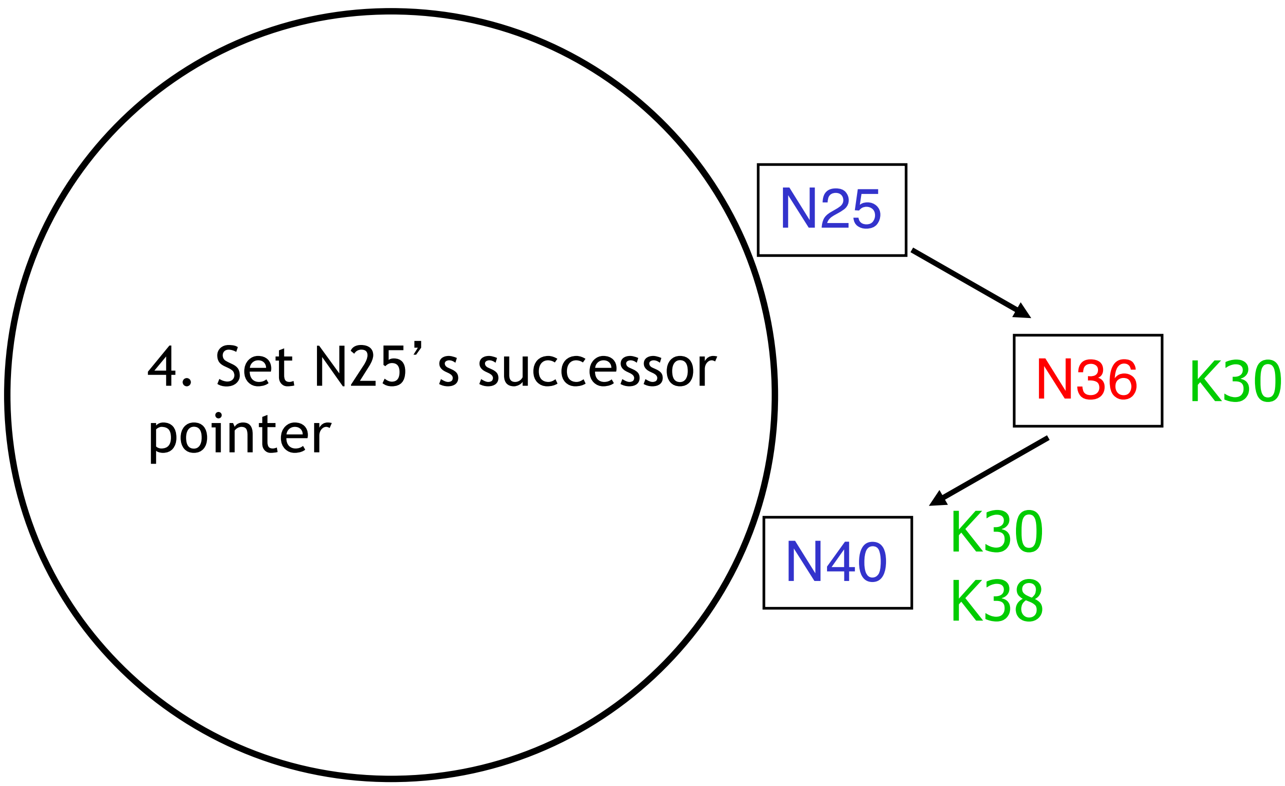


Join (3)



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Join (4)
[Done later, in stabilization]



Update other routing entries in the background
Correct successors produce correct lookups

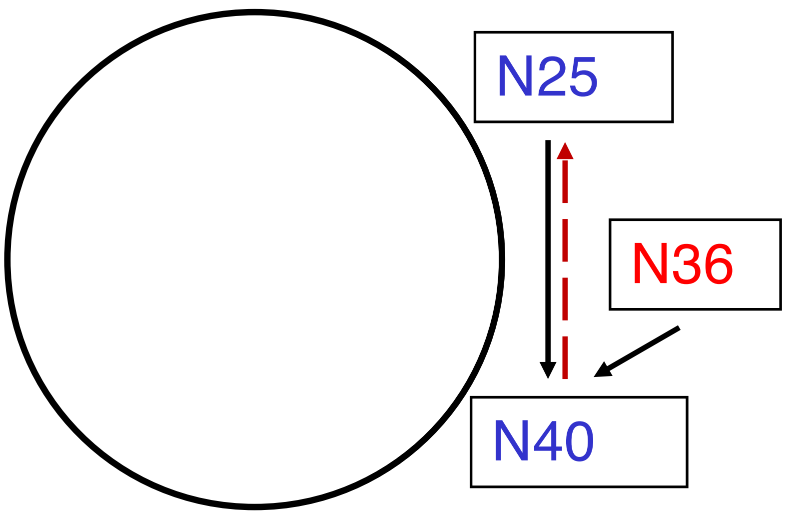
Join and stabilization

// join a Chord ring containing node n'.

n.**join**(n')

    predecessor = **nil**;

    successor = n'.*find_successor*(n);



// called periodically. verifies n's immediate

// successor, and tells the successor about n.

n.**stabilize**()

    x = *successor.predecessor*;

    if ($x \in (n, successor)$)

        successor = x;

    successor.notify(n);

// n' thinks it might be our predecessor.

n.**notify**(n')

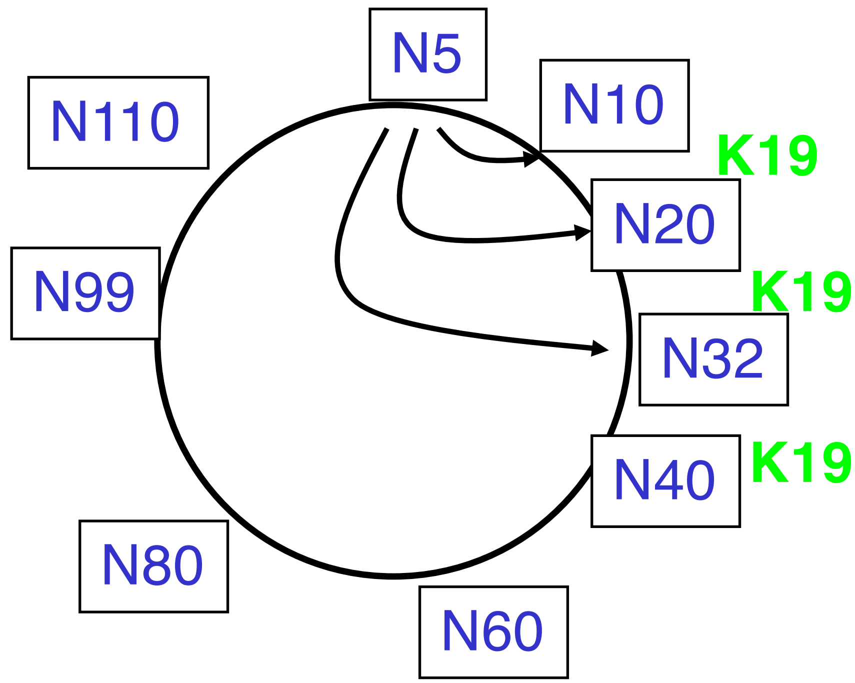
    if (*predecessor* **is nil or** $n' \in (predecessor, n)$)

        predecessor = n';

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

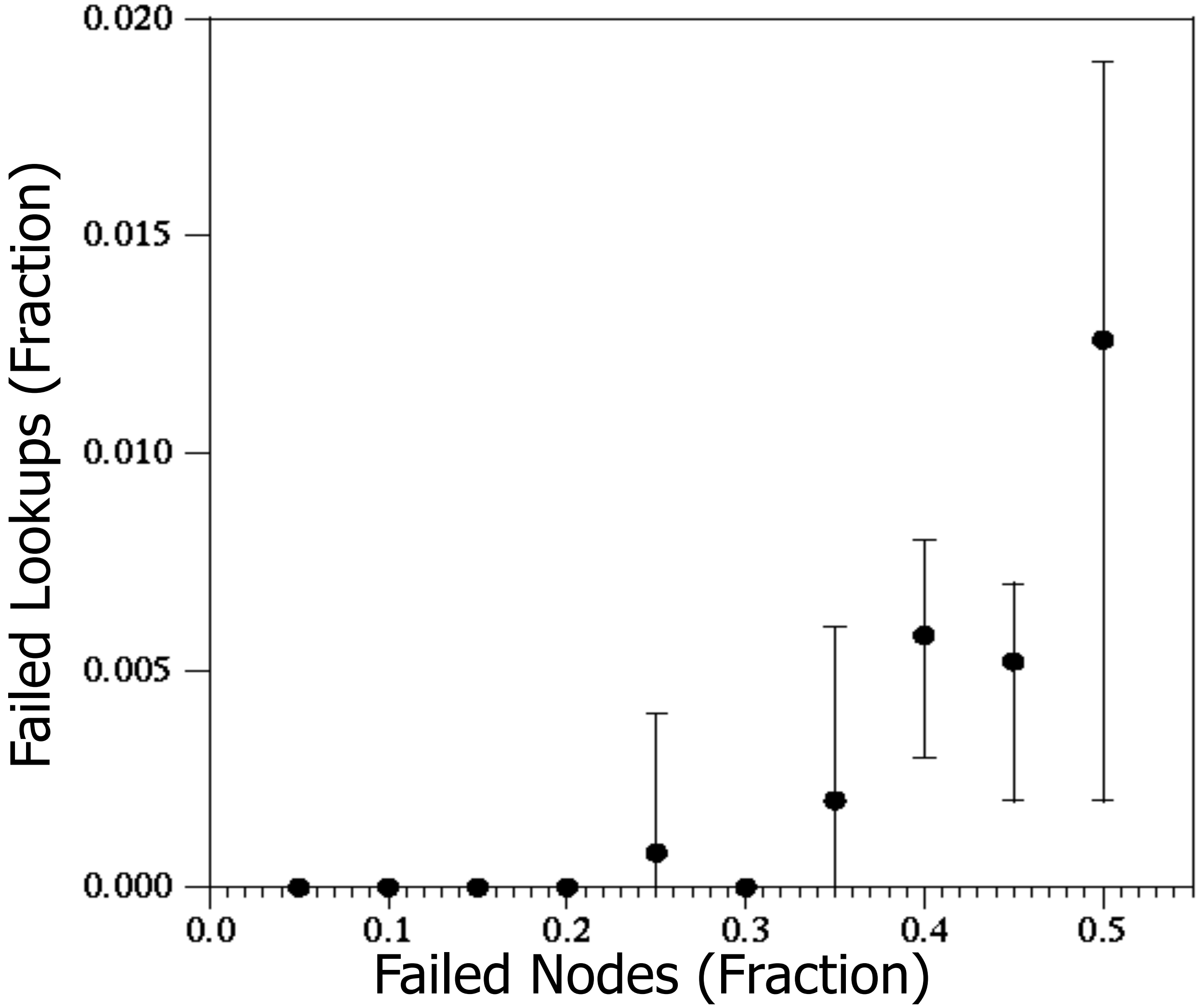
Fault-tolerance with successor lists

- When node n fails, each node whose finger tables include *n* must find *n*'s successor
- For correctness, however, need correct successor
- Successor list: each node knows about next r nodes on circle
- Each key is stored by the r nodes after "owner" on the circle
- If $r = O(\log N)$, lookups are fast *even when* P(node failure) $= 0.5$



MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Redundancy Provides Failure Resilience



- 1000 DHT nodes
- Average of 5 runs
- 6 replicas for each key (less than log N)

- Kill fraction of nodes
- Then measure how many lookups fail
- All replicas must be killed for lookup to fail
- Lookups still return fast!

When 50% of nodes fail, only 1.2% of lookups fail!

MASSACHUSETTS INSTITUTE OF TECHNOLOGY