UNIVERSITY OF ILLINOIS
AT URBANA-CHAMPAIGN

CS411 - Distributed Hash Tables

illinois.edu

Announcements

- HW4 released today
  - Due on last day of class
  - We're dropping your lowest HW/MP score
- Project Stage 5 due next week
- Final Exam
  - Friday 05/03 at 8-11am
  - Multiple rooms. Go to the right one!

Review

- What is “blind SQL injection”?
- Why do we “sanitize” user input?
- Why do we “hash” passwords?
- Why do we “salt” a hash?
- How can we mitigate SQL injection?

Peer-to-Peer Network

- Nodes are:
  - autonomous: nodes join and leave at will
  - loosely coupled: communicate over the internet
  - equal: no central control node
  - willing to share resources

Peer-to-Peer Search

- We want to be able to search for items on a P2P system
- Examples:
  - distributed BitTorrent search
  - FreeNet/GnuNet
  - CloudSNAP, Space Monkey
  - distributed relational database
  - NoSQL

NoSQL

- Throw away:
  - relational model
  - schemas/tables
  - transactions, integrity, etc. etc.
- Data stored as keys and objects
- Examples: CouchDB, MongoDB, Cassandra, BigTable, Dynamo, Voldemort

NoSQL

- Throw away:
  - relational model
  - schemas/tables
  - transactions, integrity, etc. etc.
- Data stored as keys and objects
- Examples: Facebook, Amazon, LinkedIn, Google, Oracle, TOR

Example

```
{“id”: “034e5fc4097b5f243266cbb918678c59640b501a”,
   ”name” : ”value”}
{”id”: ”a1e756f676bcac92ed5127215dfb6c638bd3b596”,
   “x”: 1}
{”id”: ”941dac88b5360264af152db2b02c654d5f5bc87f”,
   “x”: 1, “y”: ”happy”}
{”id”: ” 0f6879a27088581129ad2da99437ae93aa6ee570”,
   “x”: 1, “y”: ”tree”}
{”id”: ” a64ab7b189cdddfd691c4044f75e8e73b494ab1a”,
   “x”: 1, “y”: 12901}
```

Distributed Hash Table

- Search for items using a hash value
  - Key-value pairs
- Key-value table can't belong to one node
  - violates P2P principles
  - could be too big for one node
- Values must be distributed through the network in a balanced way

Hash Function

- Same input always produces same output
- Given the output, can't infer the input
- Output is evenly distributed across codomain (range, image)

Distributed Hash Table

- Choose a hash function with m bit output
  - MD5 m=128 bits
  - SHA-1 m=160 bits
  - Example m=6
    - Output from 0 to 63

Side note

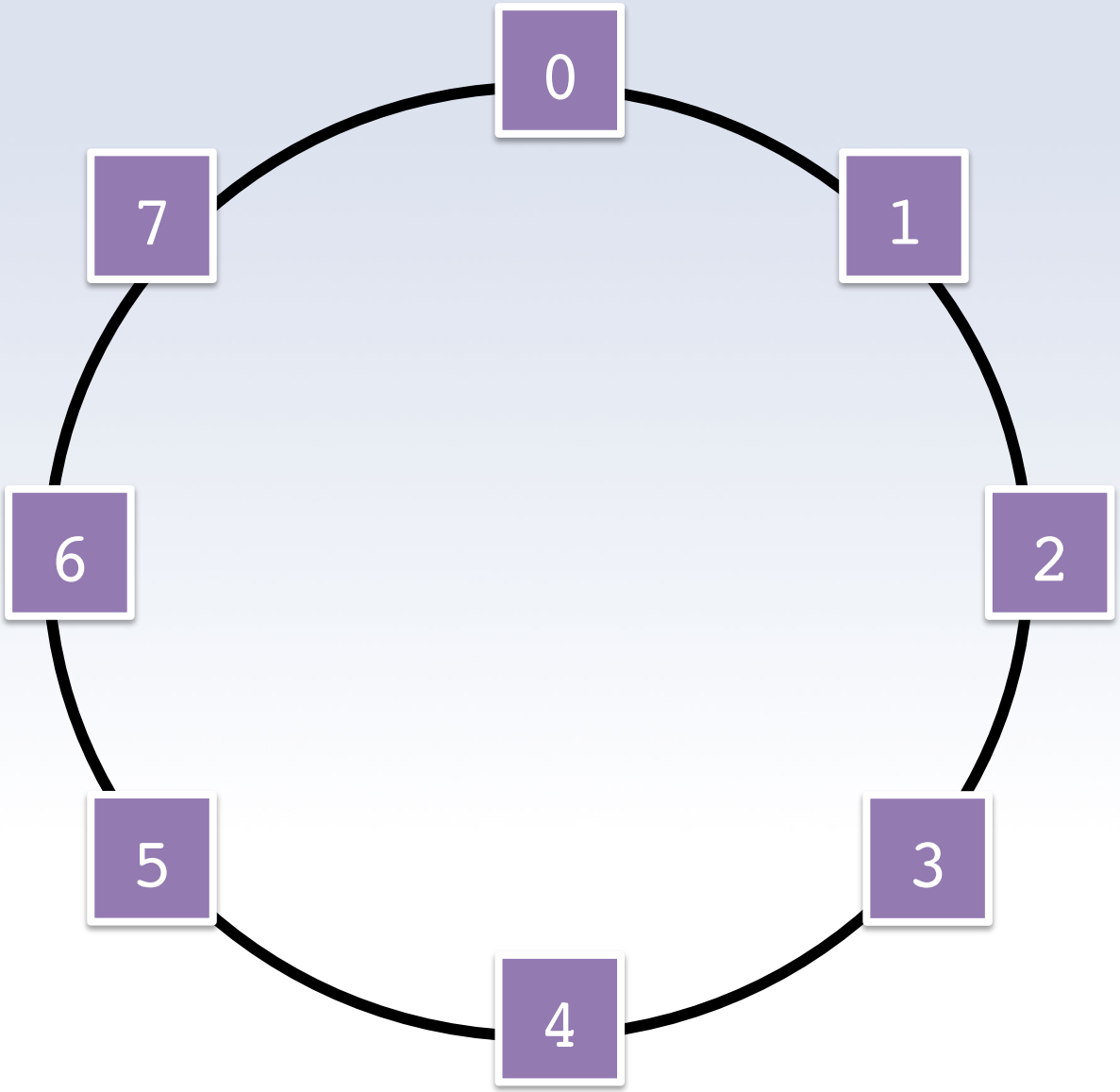
2^{128}=340 undecillion, 282 decillion, 366 nonillion, 920 octillion, 938 septillion, 463 sextillion, 463 quintillion, 374 quadrillion, 607 trillion, 431 billion, 768 million, 211 thousand and 456 (thanks Wolfram Alpha!)

2^{160}=more than one value for every grain of sand on every beach on Earth

Distributed Hash Table

- Imagine output space (range, image) of hash as a circle
  - Like a clock, starts at 0 and proceeds clockwise to 2^m-1

Circle (m=3)

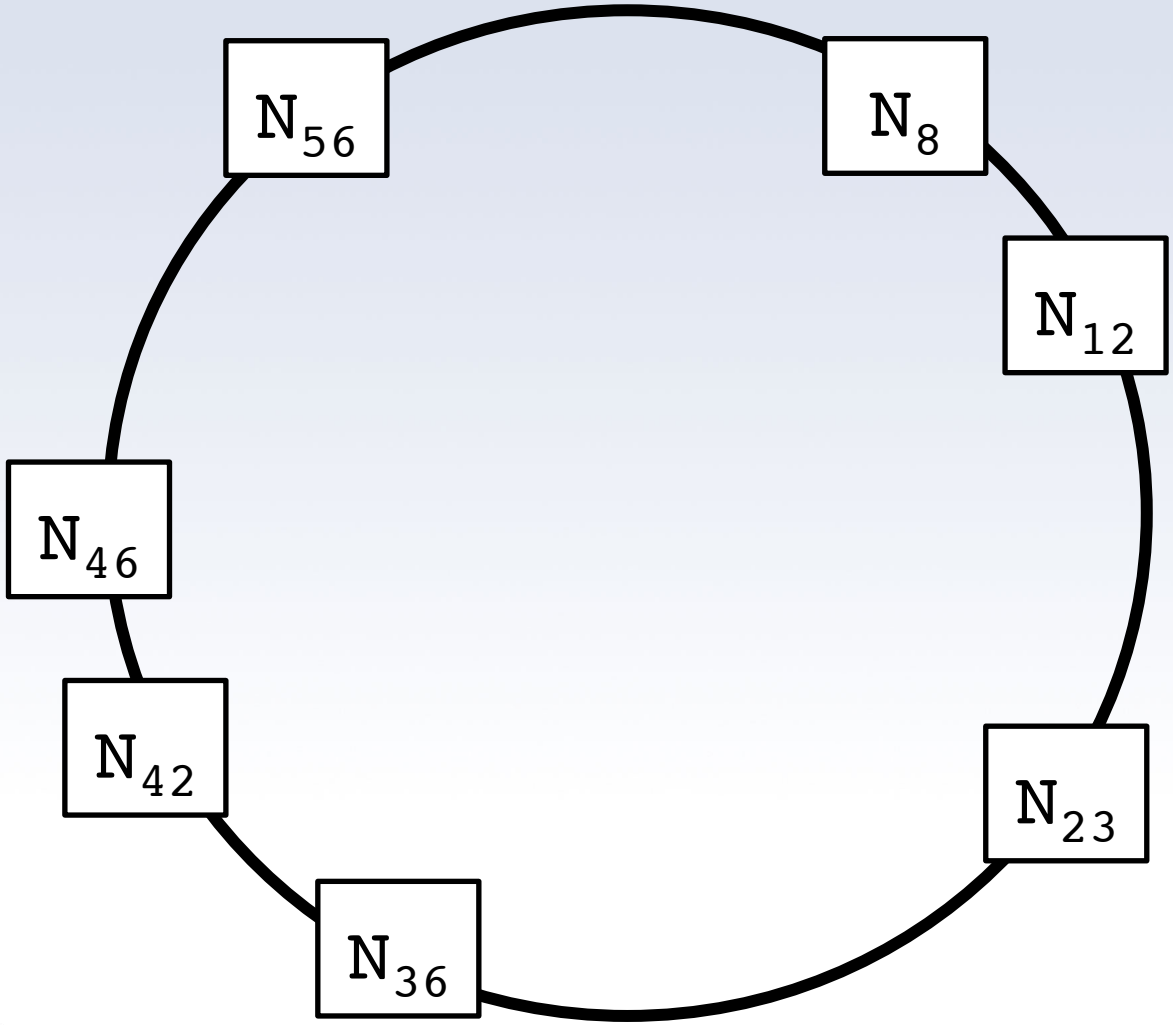


illinois.edu

Chord Circle

- Nodes are distributed on this hash circle:
  - Hash unique identifier (e.g. IP address, username, etc.)
  - Track predecessor and successor
    - predecessor - first node with a lower hash
    - successor - first node with a higher hash
  - Predecessor and successors “wrap around” the origin (just like a clock)

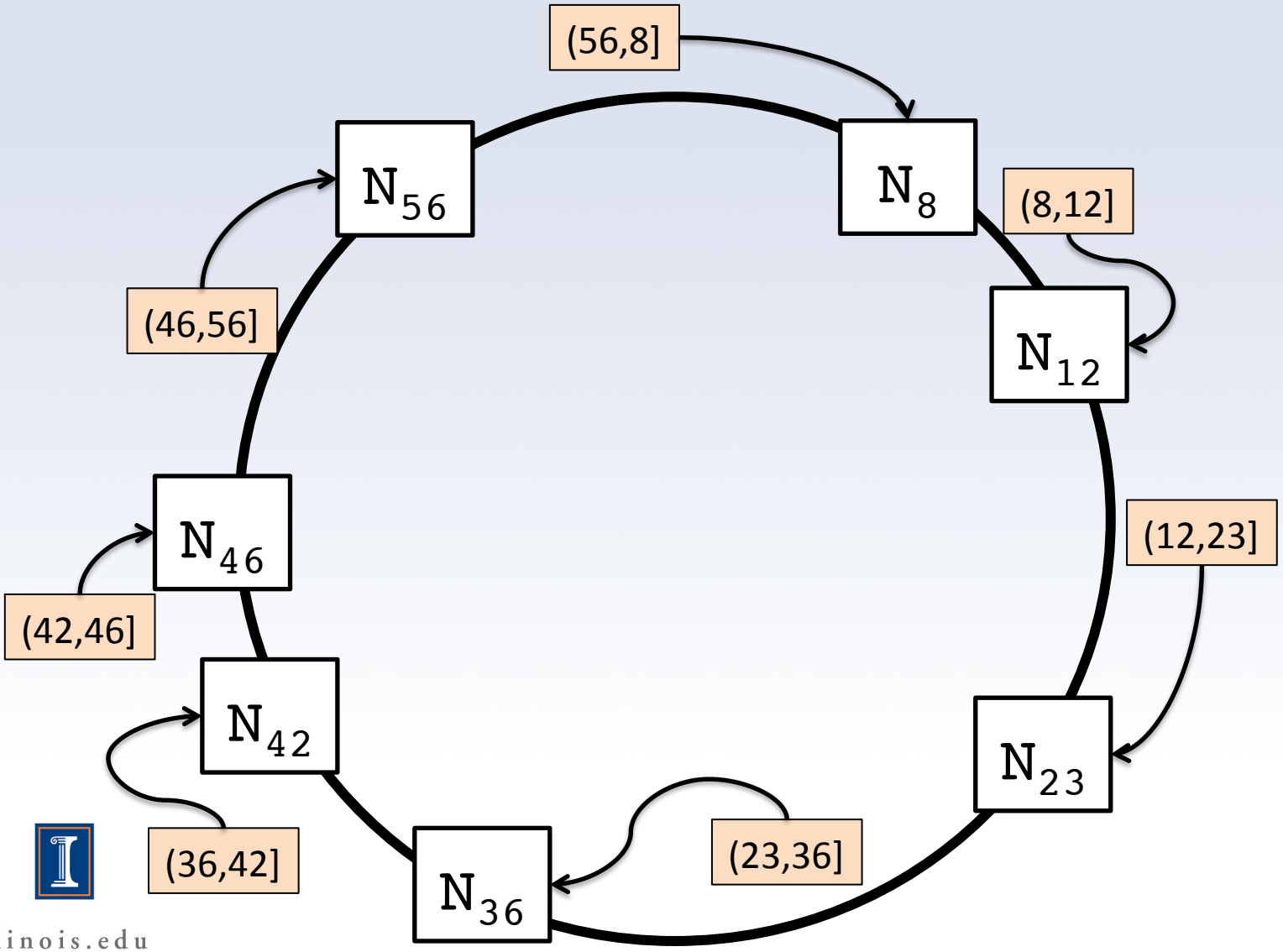
Chord Circle (m=6)



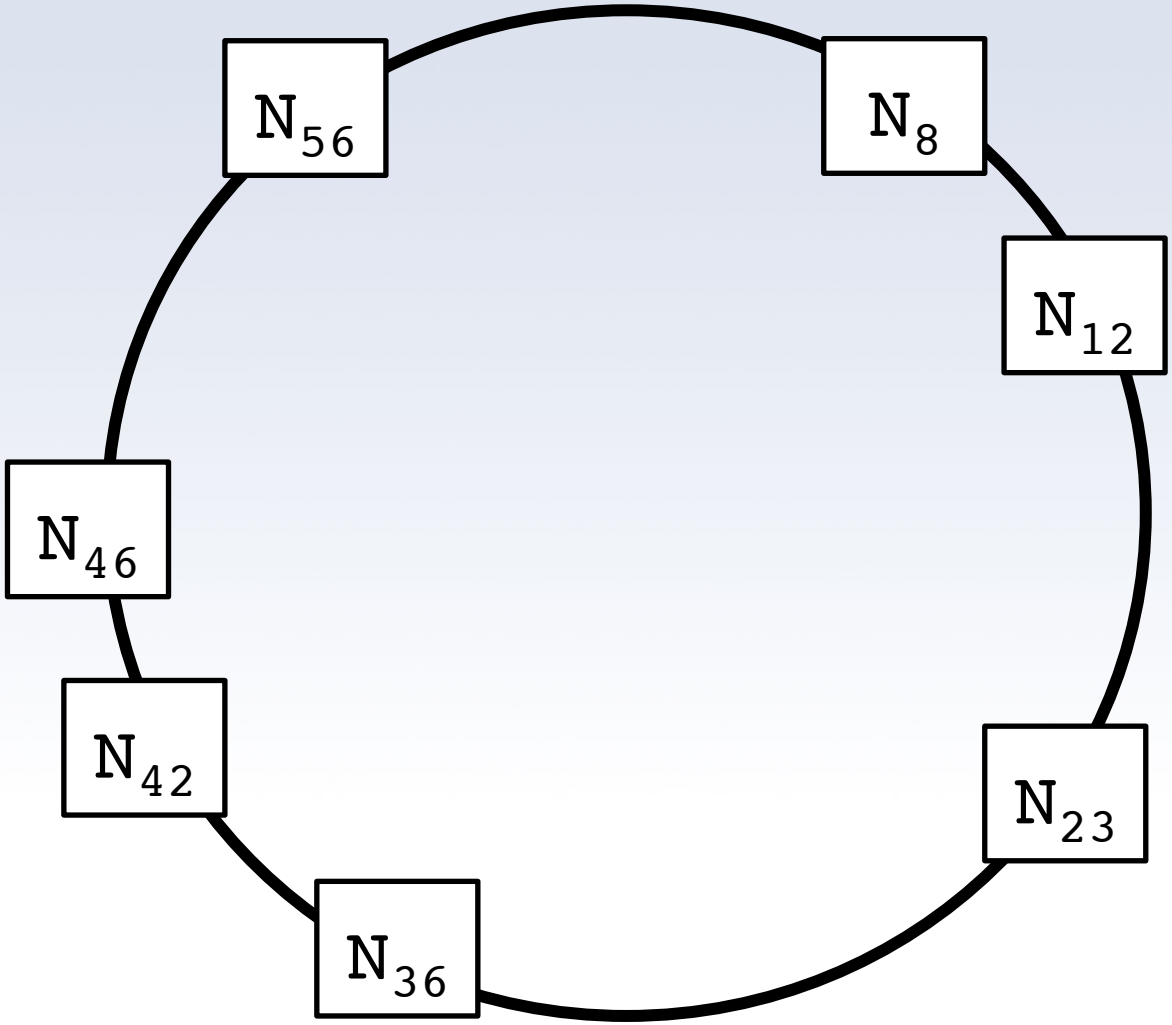
Chord Circle

- Nodes organized
- What about data? <K,V>
  - Use the same hash function!
  - Find smallest node where h(K)≤h(N)
  - Store <K,V> there

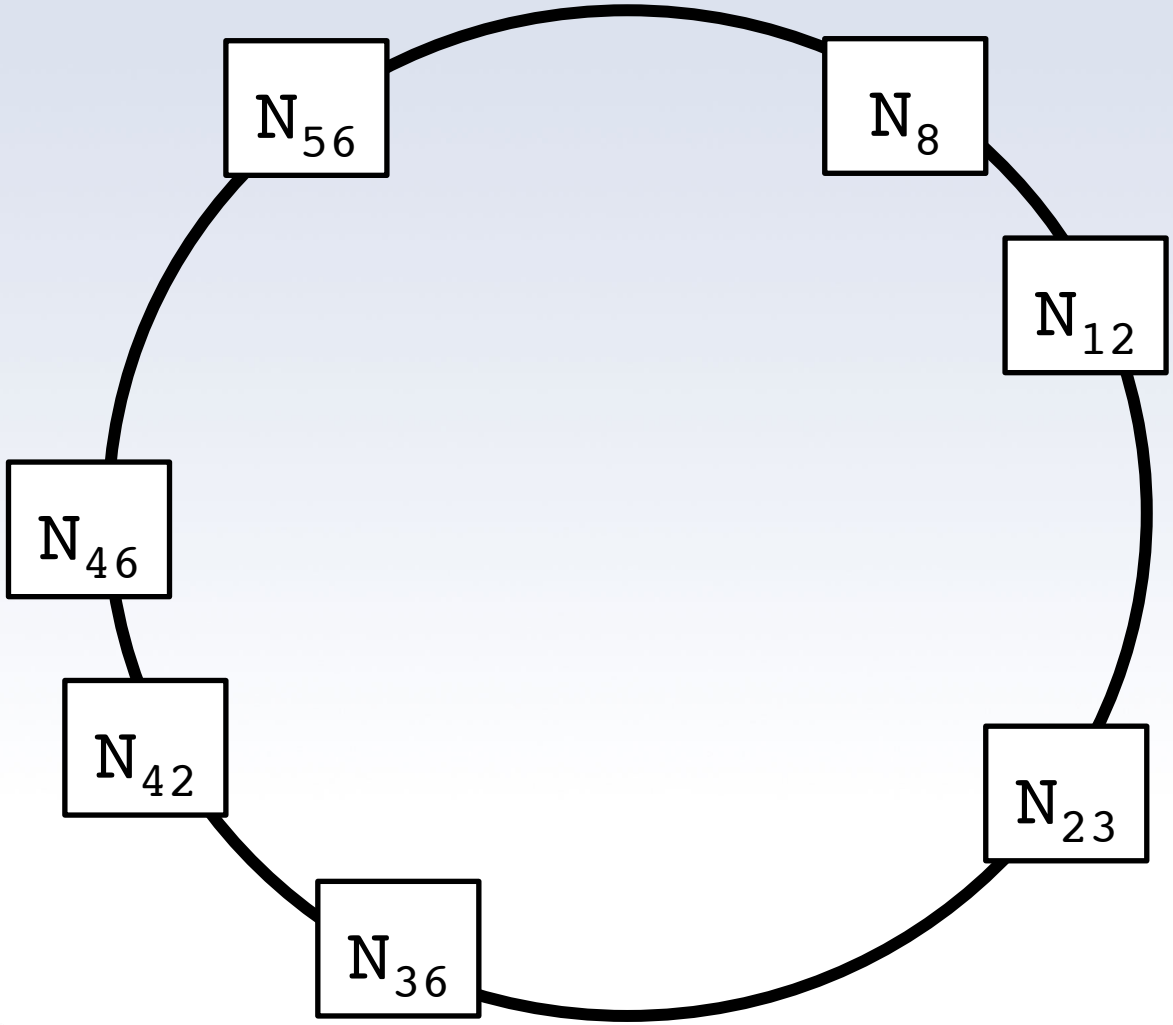
Example



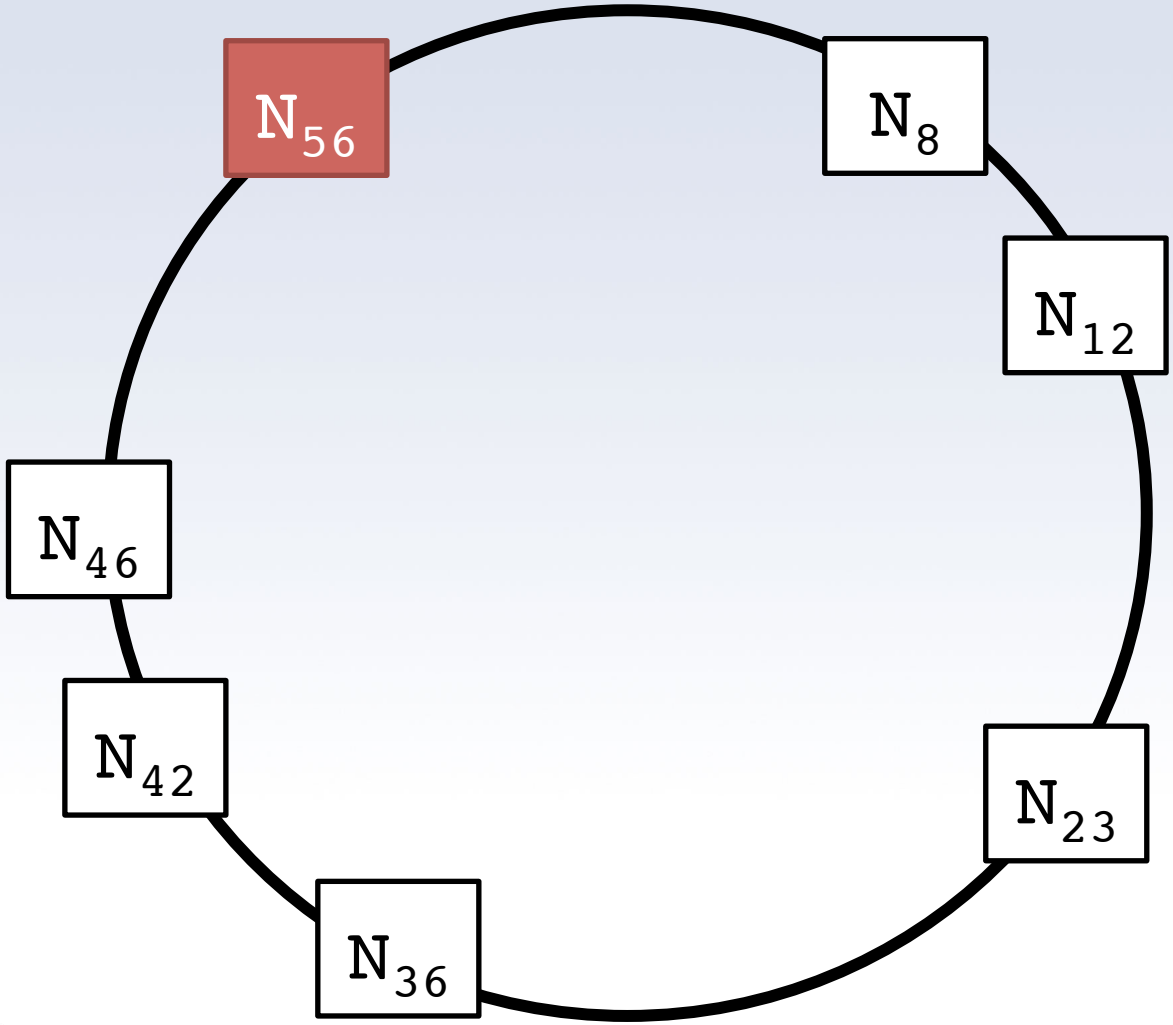
<“Electric Feel, MGMT”,electricfeel.mp3>



<“Electric Feel, MGMT”,electricfeel.mp3>
h(“Electric Feel”)=54



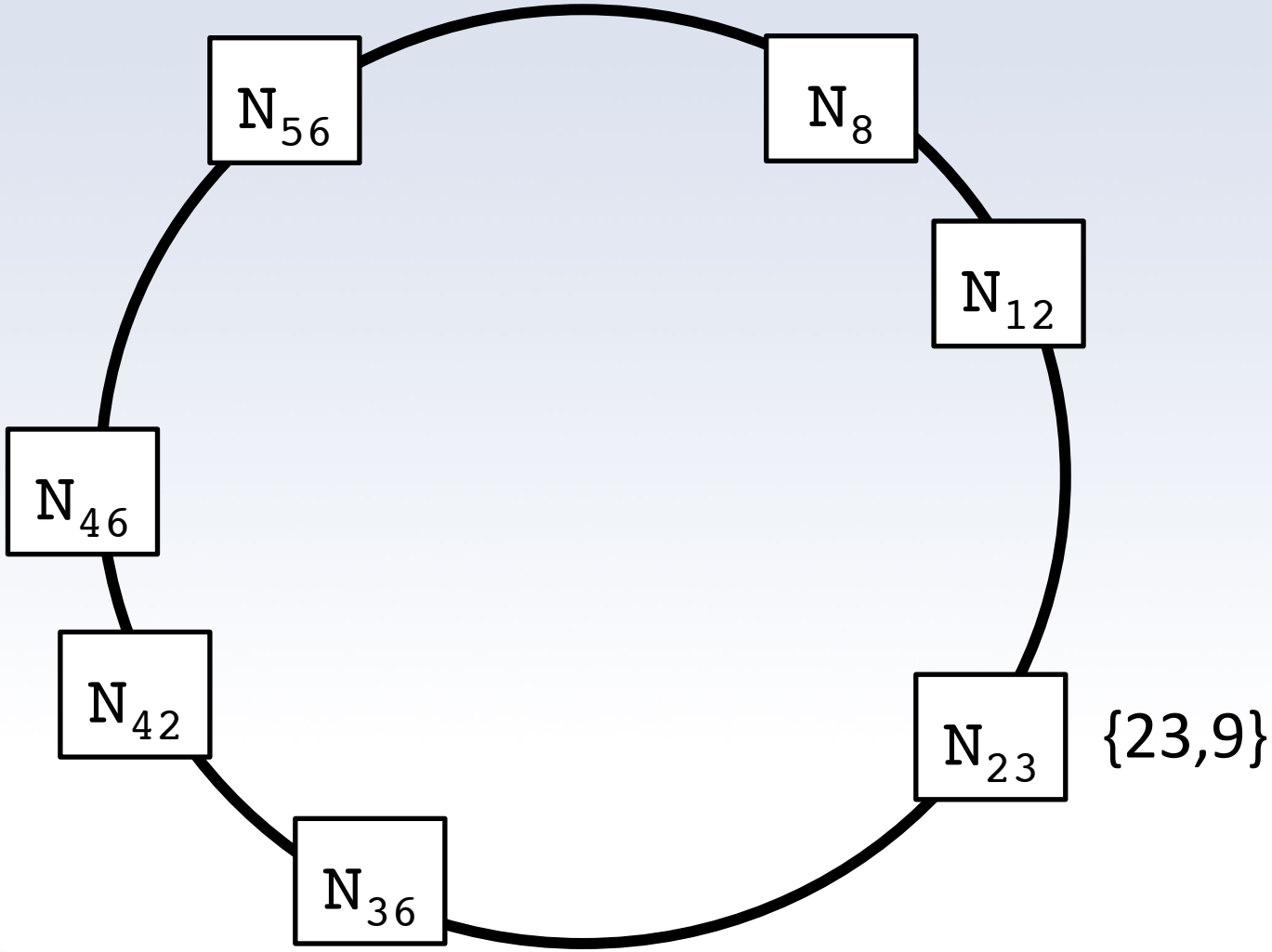
<“Electric Feel, MGMT”,electricfeel.mp3>
h(“Electric Feel”)=54



illinois.edu

Naïve Search

- Send a message around the circle until we find the right node
  - Send out our hash and hash of the search key
    - e.g. {23,9} “node 23 is looking for file with hash 9”
  - Correct node directly sends back result



{23,9}

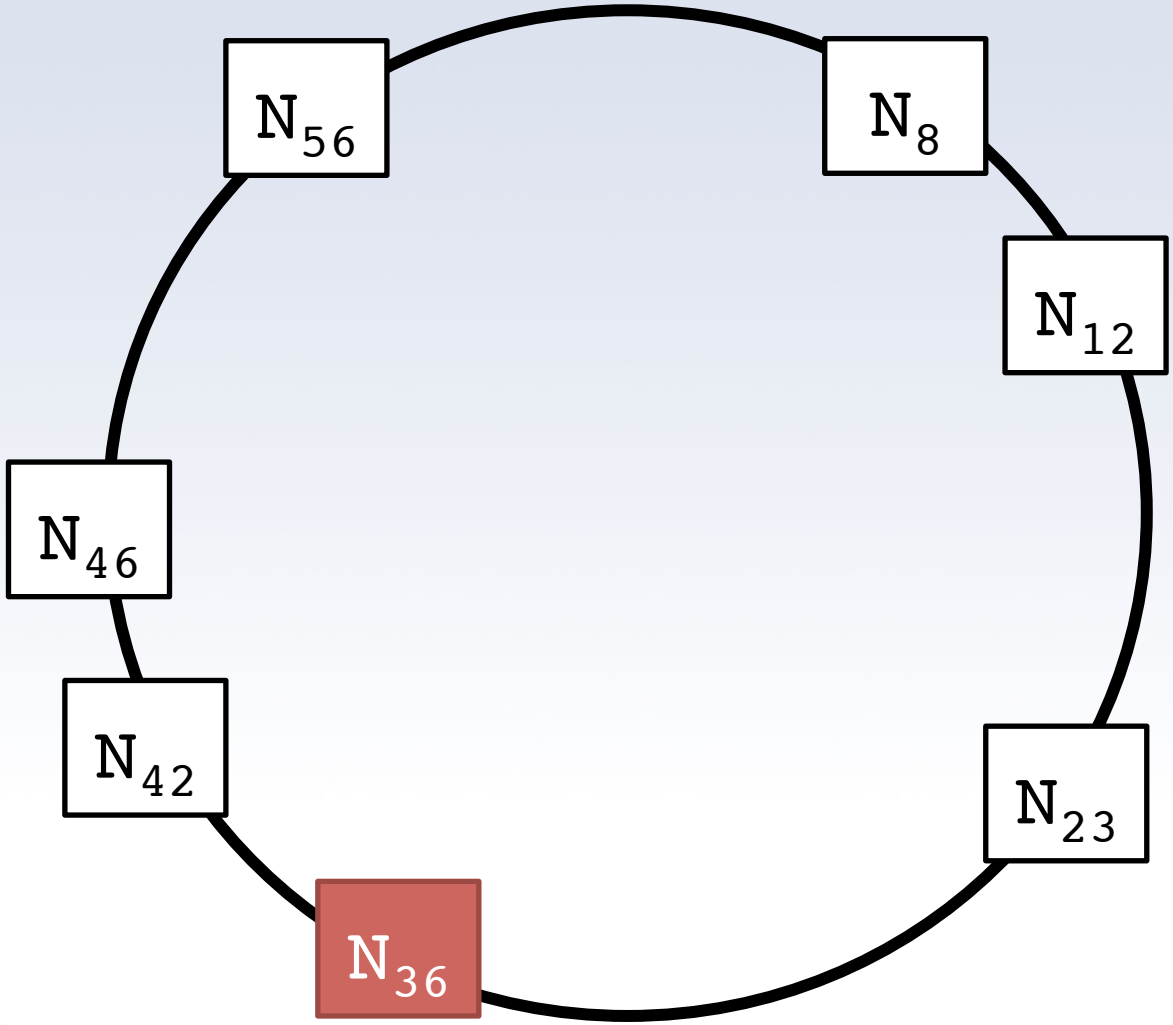
Naïve Search

- Too expensive
  - Have to send, on average O(Nodes) messages
  - Too slow!
- Keep track of all other nodes?
  - Have to store O(Nodes) messages
  - Too big!

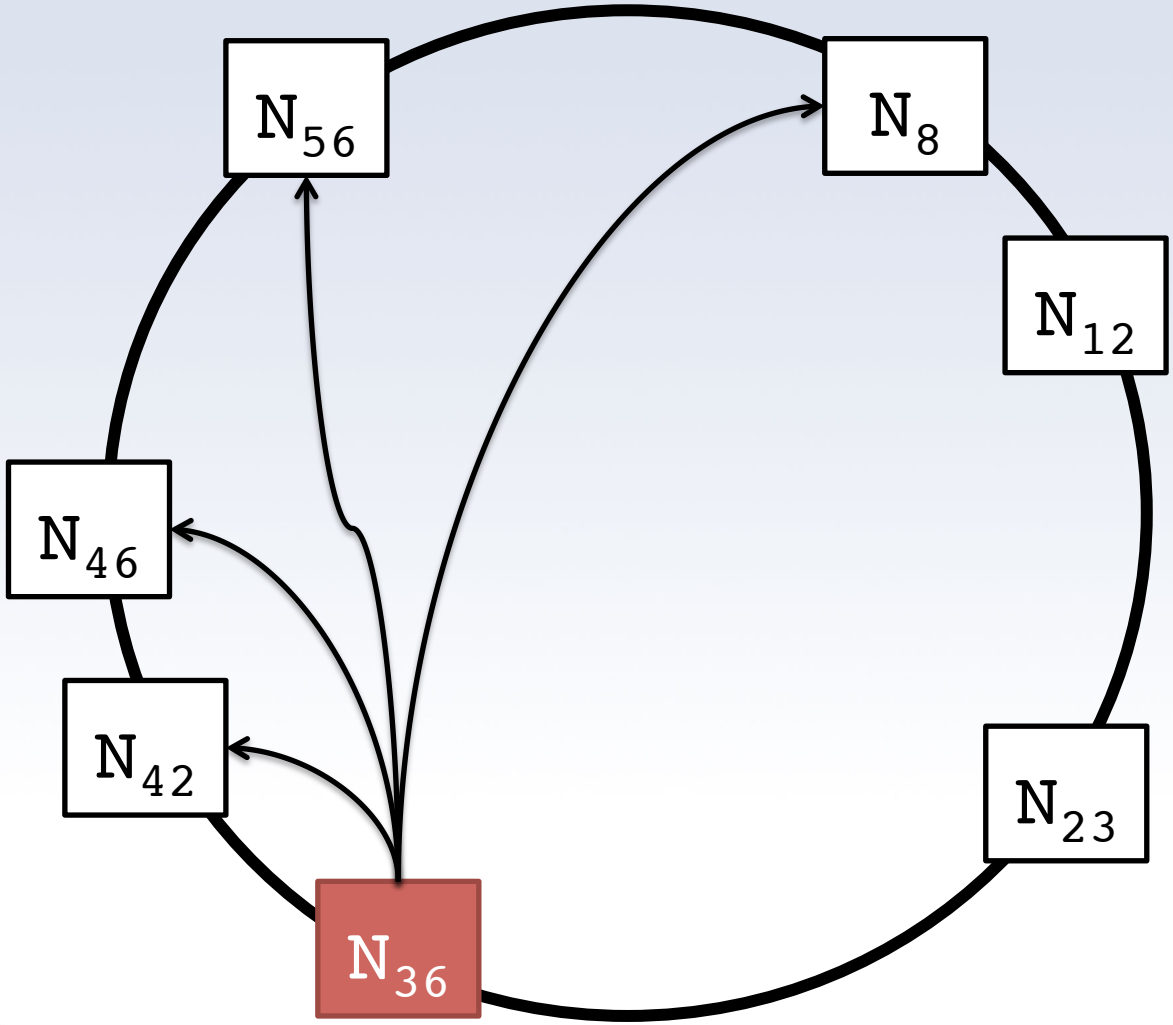
Finger Table

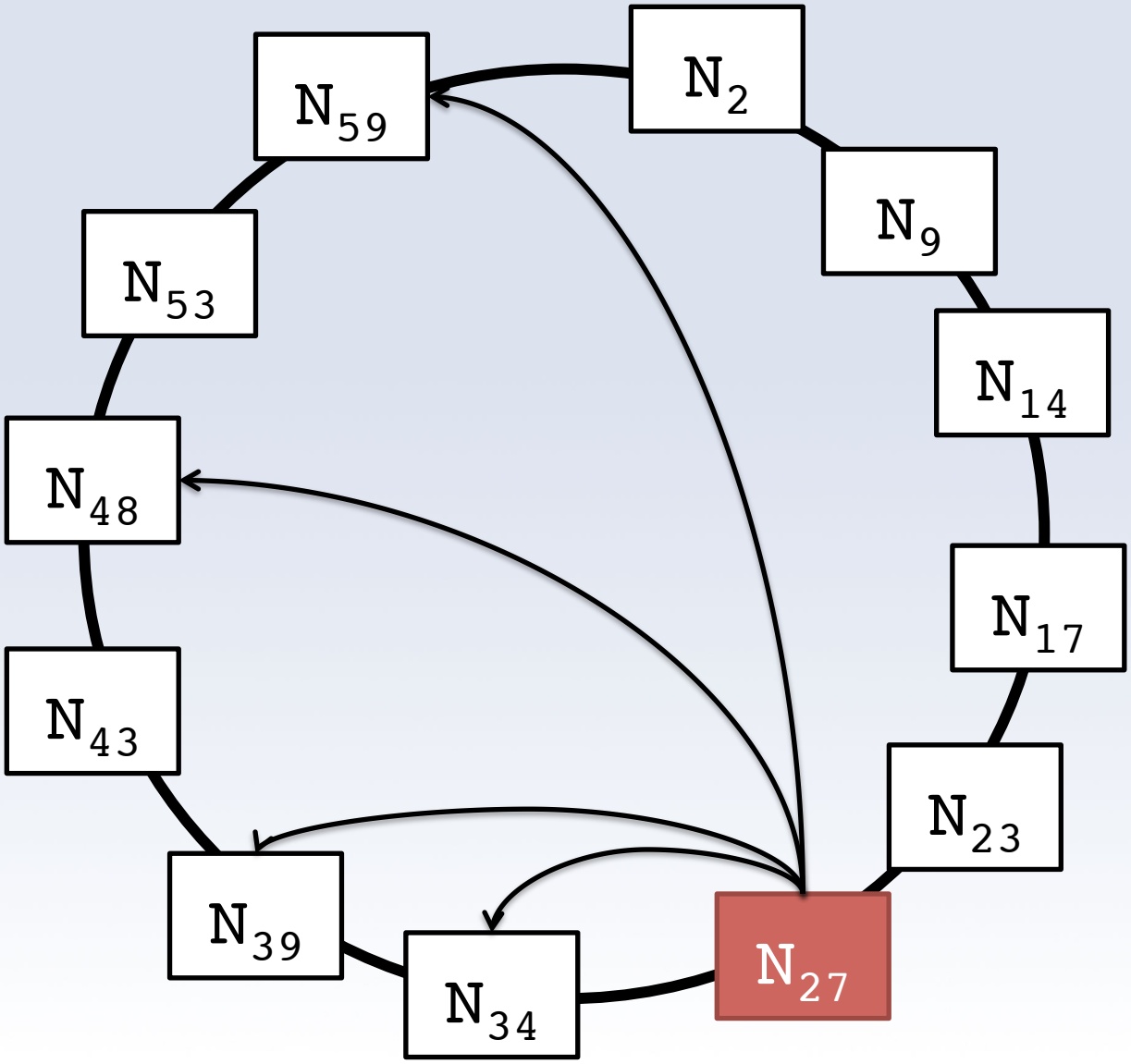
- Track all nodes at power of 2 distances
  - $1,2,4,8,16,\ldots,2^{m-1}$
  - Distance in hash codomain (range, image)
- Basically, look up value that distance away
  - Find h(N)+distance mod 2^m

Distance	1	2	4	8	16	32
Node	42	42	42	46	56	8



Distance	1	2	4	8	16	32
Node	42	42	42	46	56	8





Distance	1	2	4	8	16	32
Node	34	34	34	39	48	59

Finger Table

- How does this help?
  - Route our searches like a binary search
  - First node sends it halfway there
  - Next node sends it halfway again
  - Repeat until we find the right node

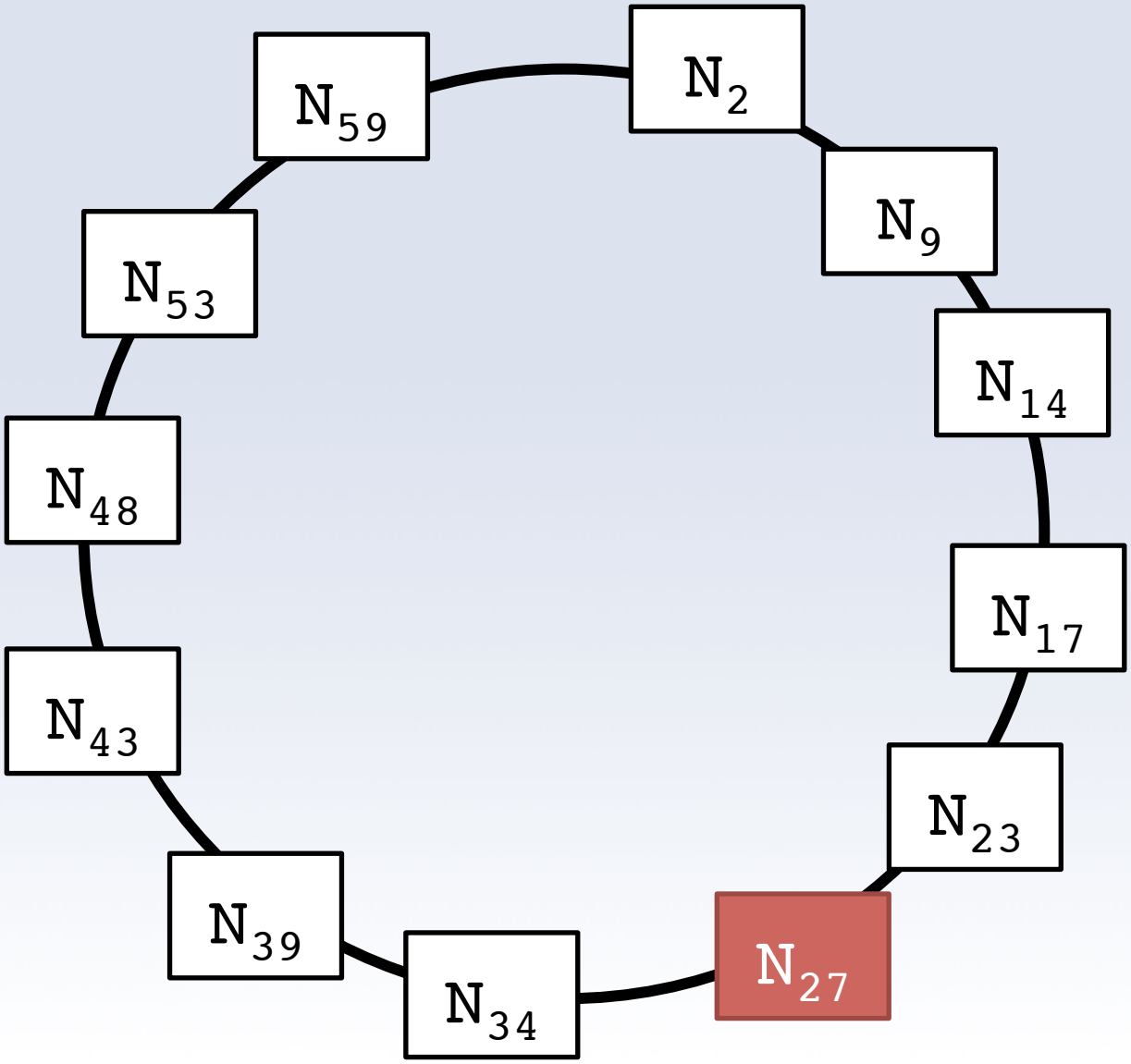
Finger Table Search

- Terminology
  - N_r is requesting node (initiated search)
  - N_c is current node (initially N_c=N_r)
  - x is the hash of search key (h(K)=x)

Finger Table Search

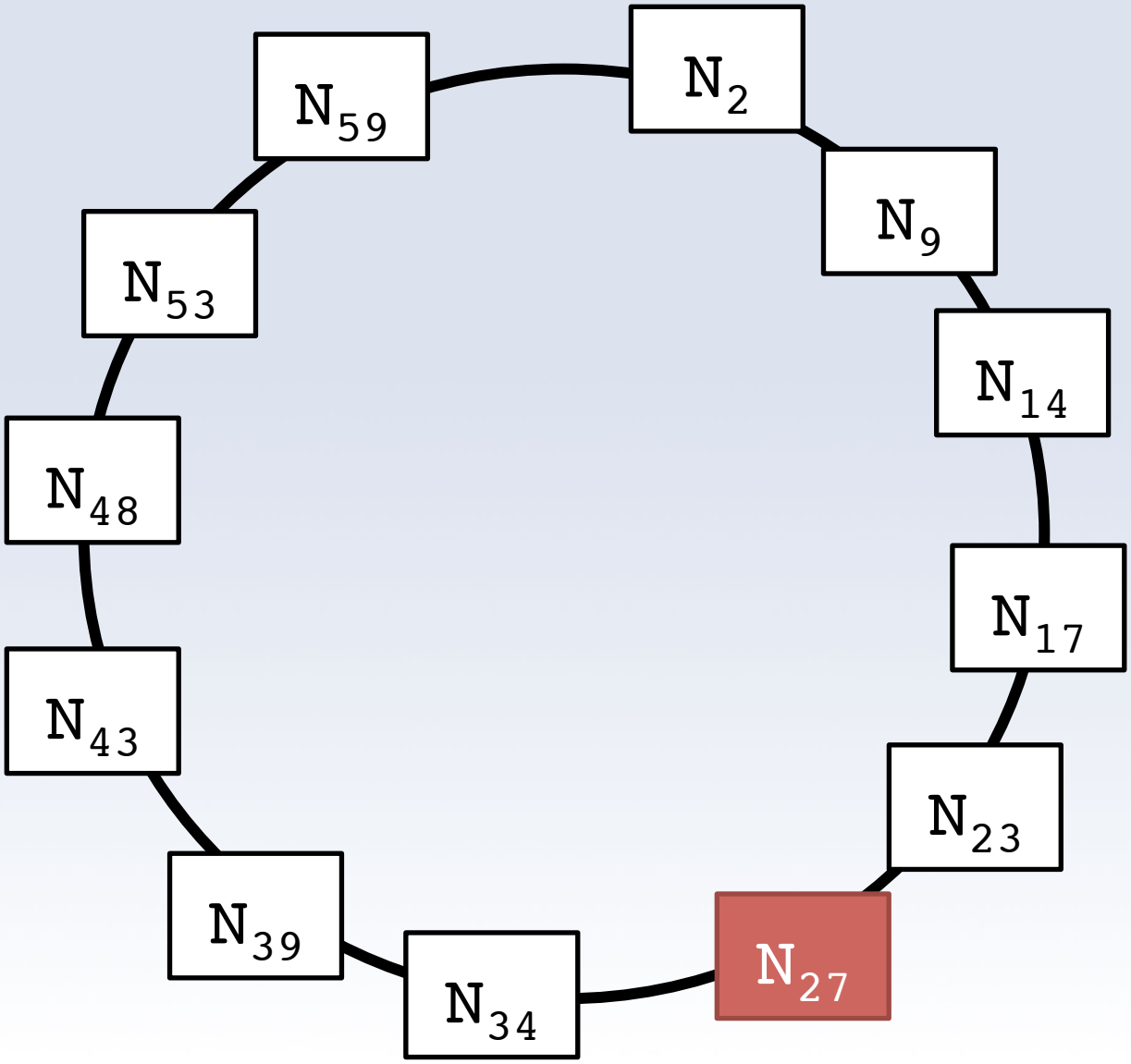
1. N_c checks if its successor N_s has search key ($c<x\leq s$)
   - If so, N_s looks up x and returns result to N_r.
   - Search is done.
2. N_c searches its finger table for highest numbered node N_h less than x.
   - Tell N_h to search for x on behalf of N_r
   - Now $N_c=N_h$ and search continues

N_{27} searching for h(x)=19



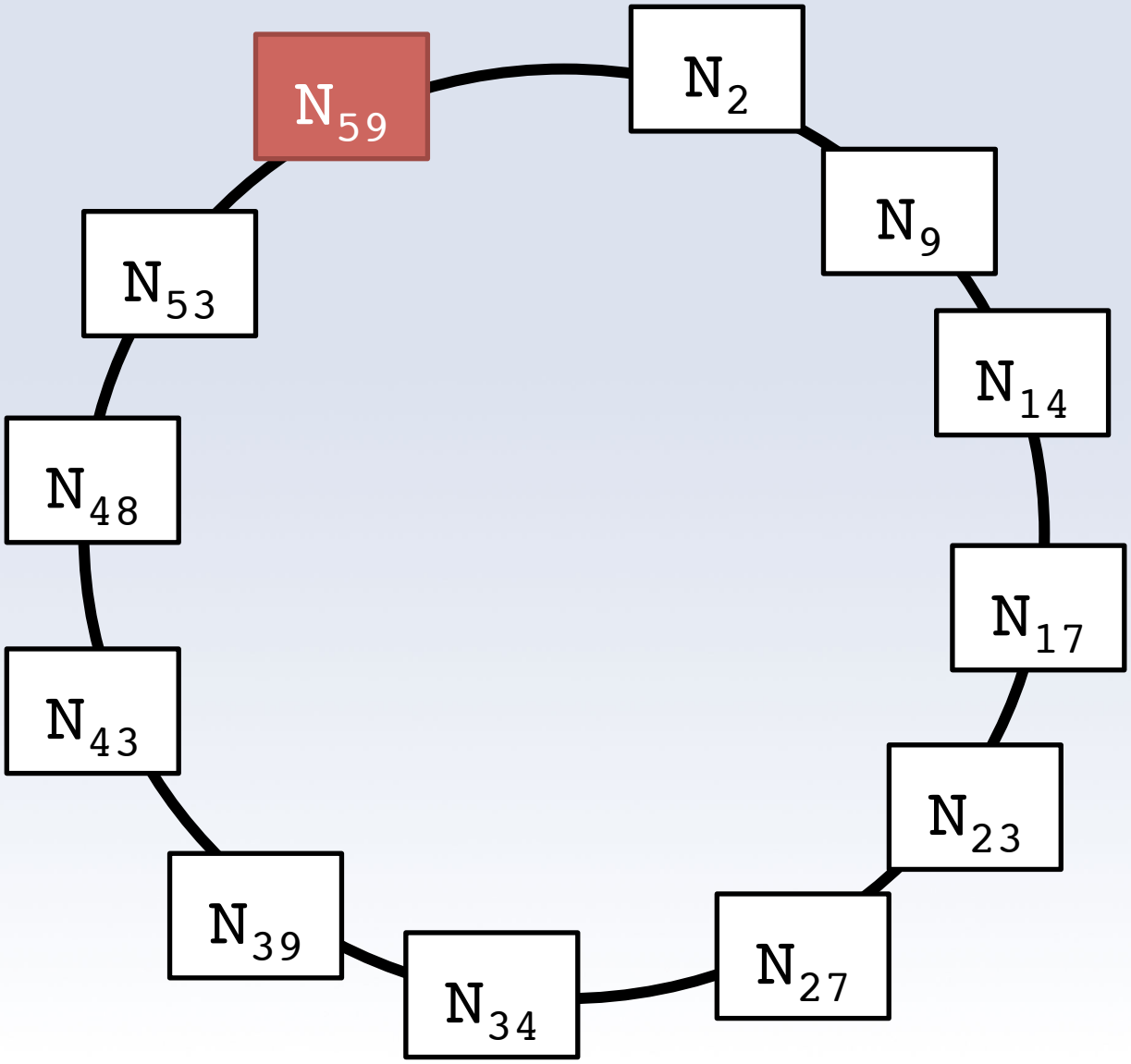
Distance	1	2	4	8	16	32
Node	34	34	34	39	48	59

N_{27} searching for h(x)=19



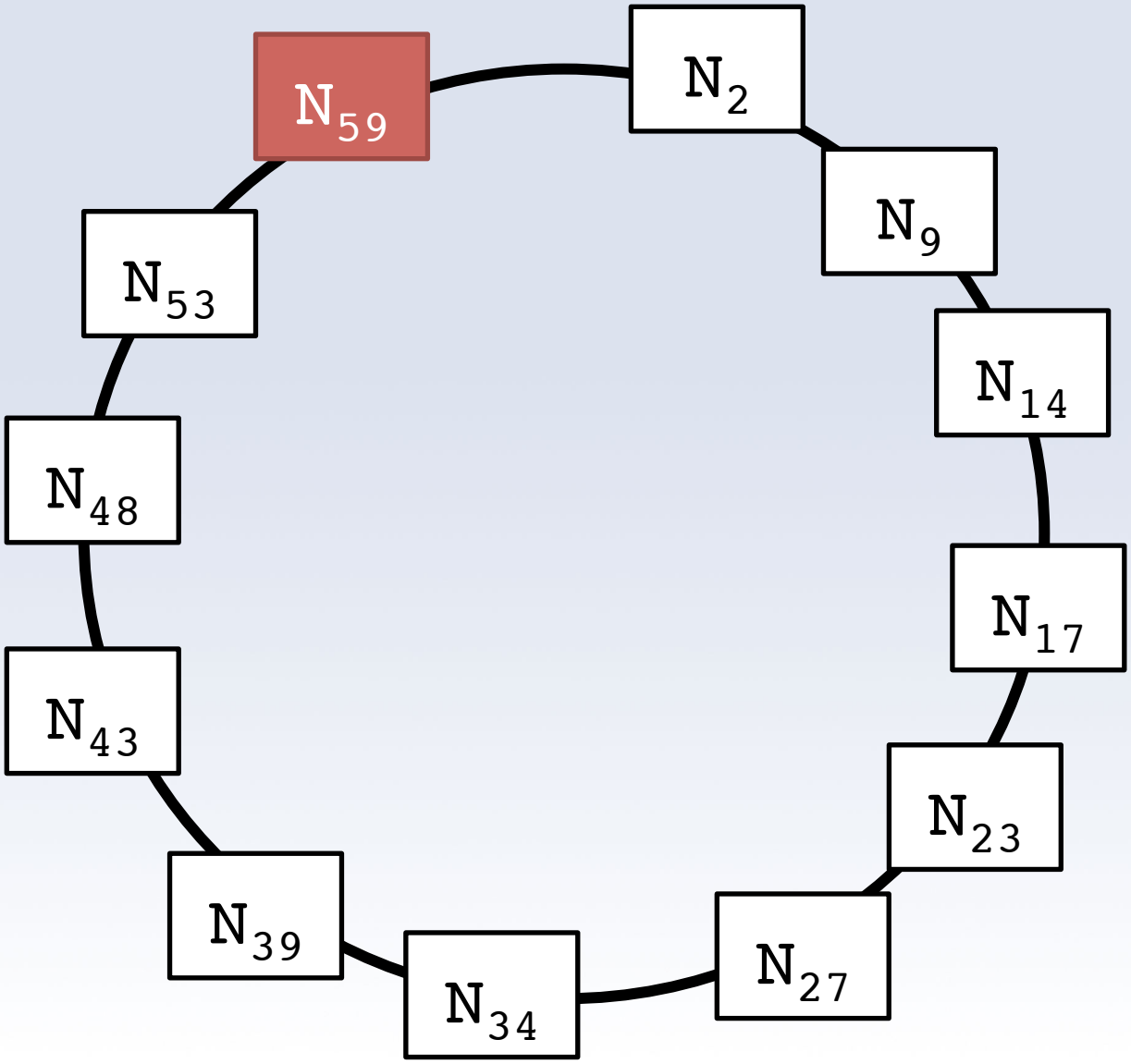
Distance	1	2	4	8	16	32
Node	34	34	34	39	48	59

N_{27} searching for h(x)=19



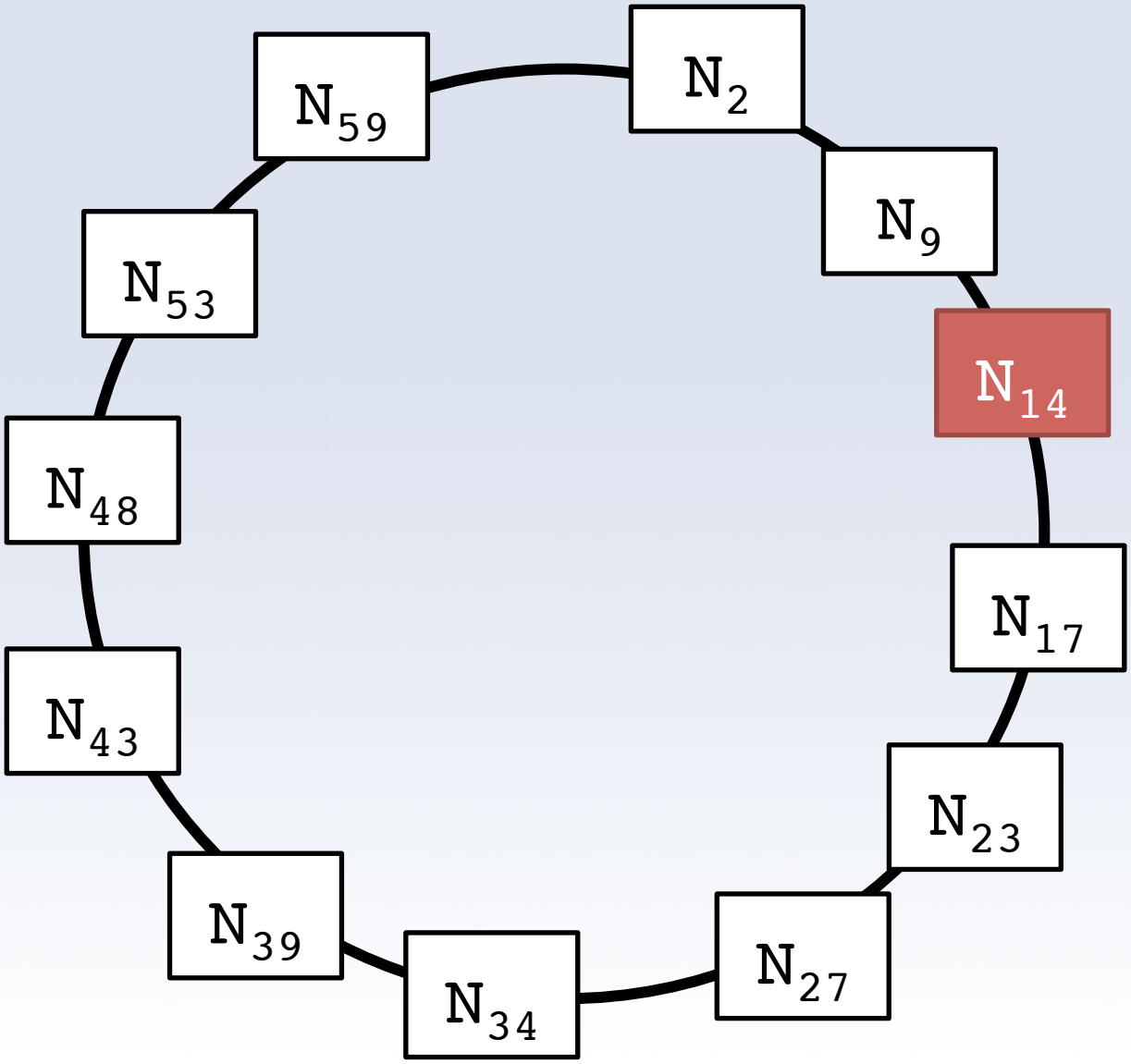
Distance	1	2	4	8	16	32
Node	2	2	2	9	14	27

N_{27} searching for h(x)=19



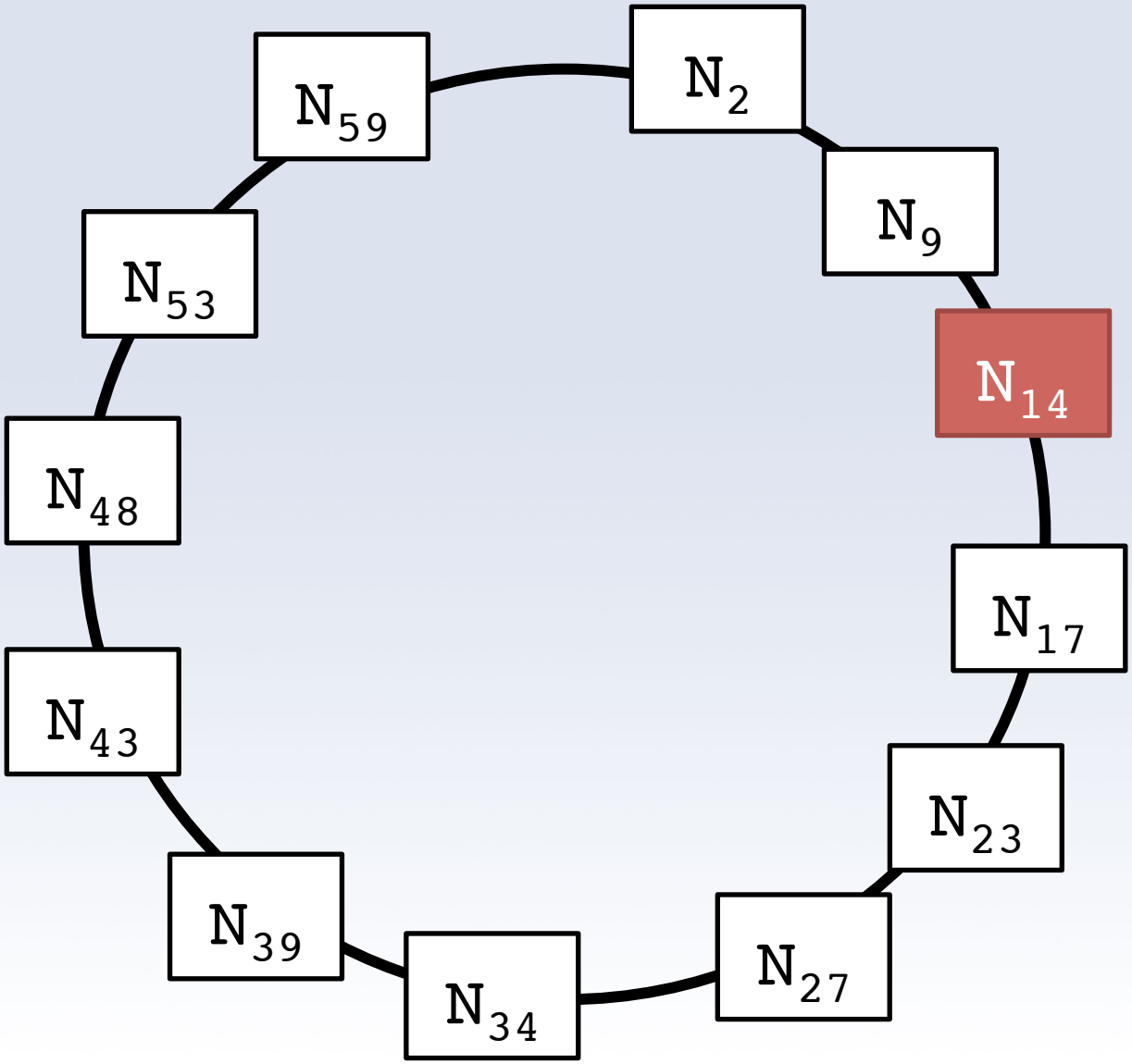
Distance	1	2	4	8	16	32
Node	2	2	2	9	14	27

N_{27} searching for h(x)=19



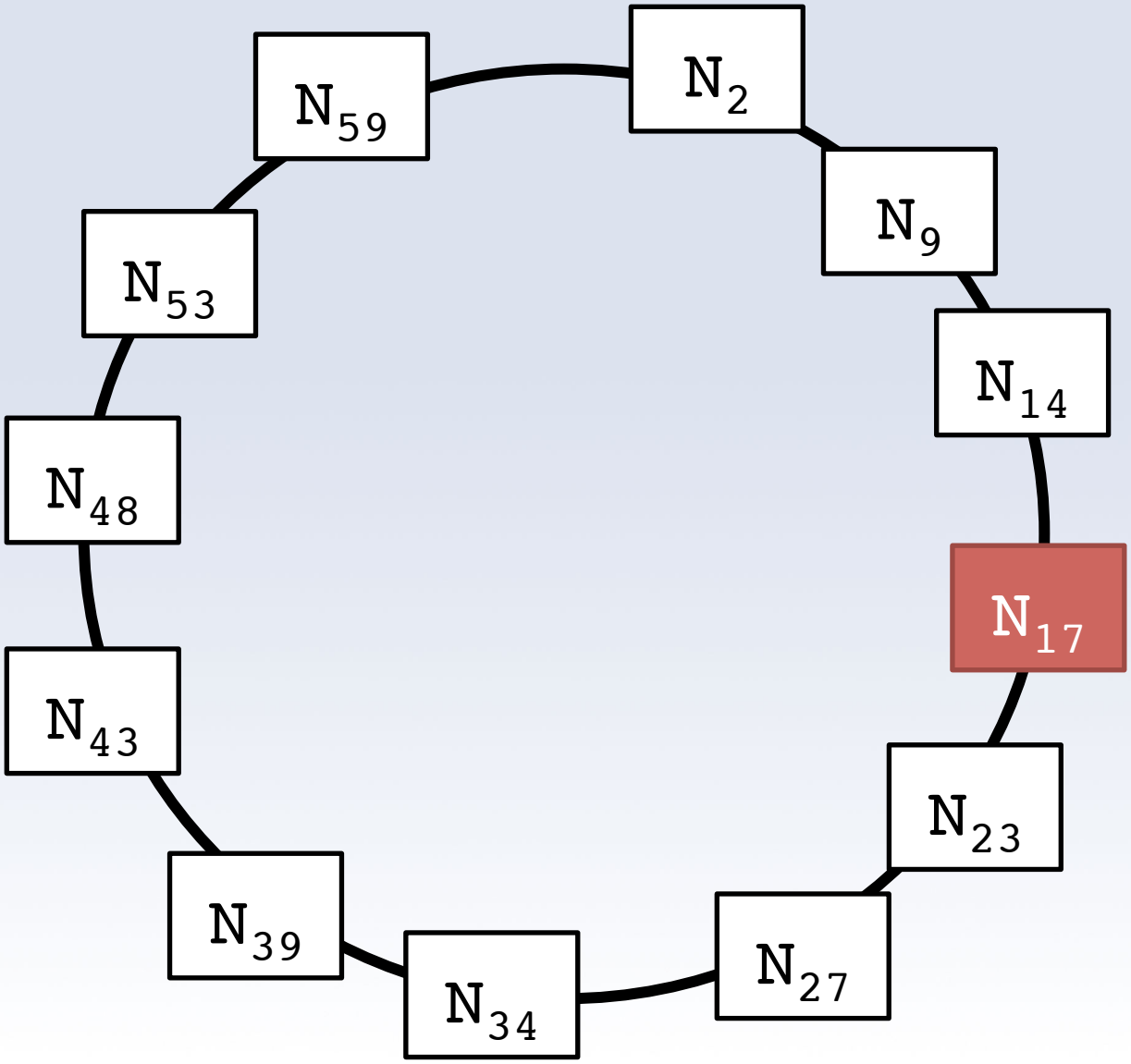
Distance	1	2	4	8	16	32
Node	17	17	23	23	34	48

N_{27} searching for h(x)=19



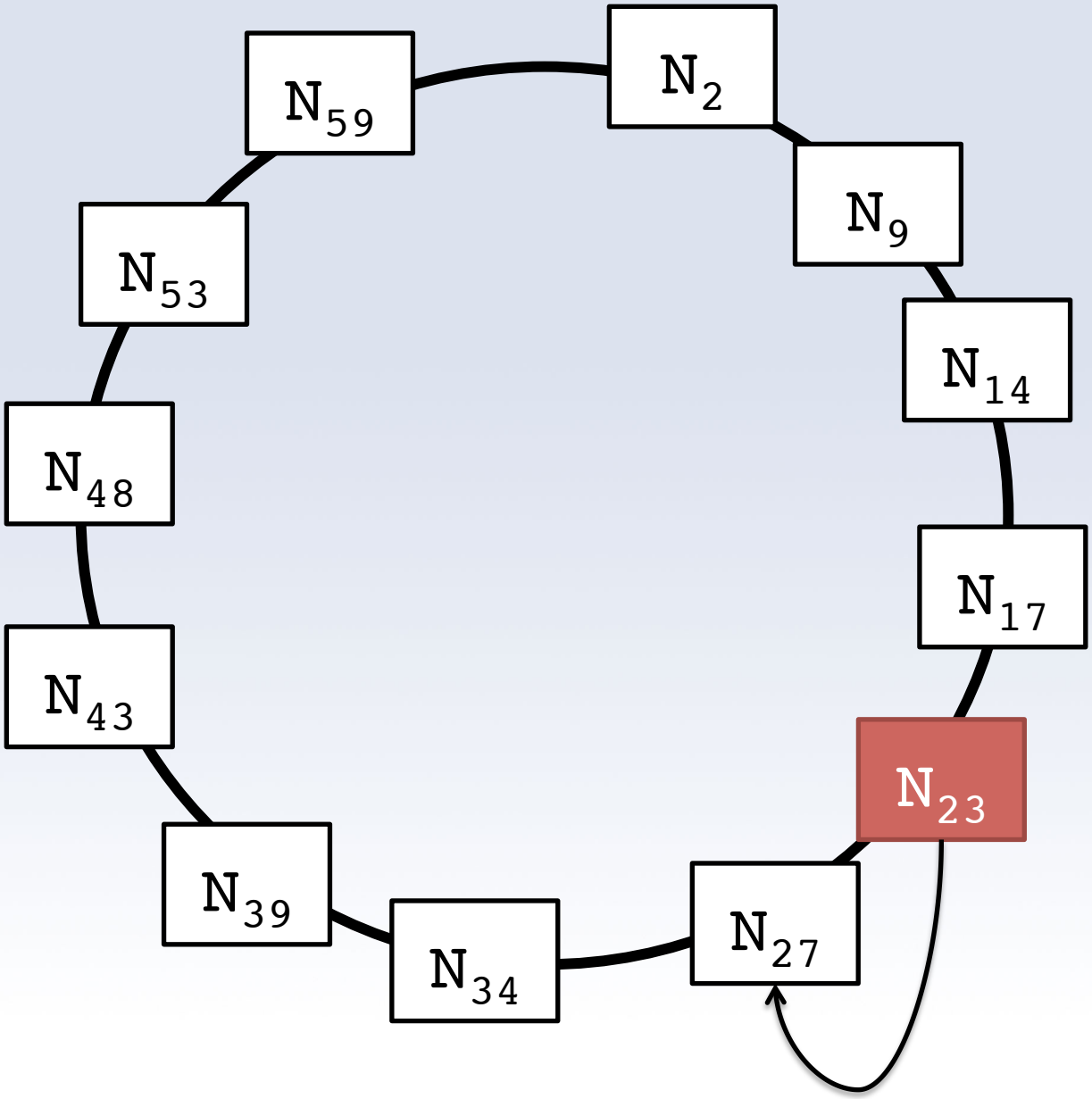
Distance	1	2	4	8	16	32
Node	17	17	23	23	34	48

N_{27} searching for h(x)=19



Distance	1	2	4	8	16	32
Node	17	17	23	23	34	48

N_{27} searching for h(x)=19



illinois.edu

Finger Table Search

- Number of entries in table: m-1
- Number of messages sent for search:
  O(log(nodes))

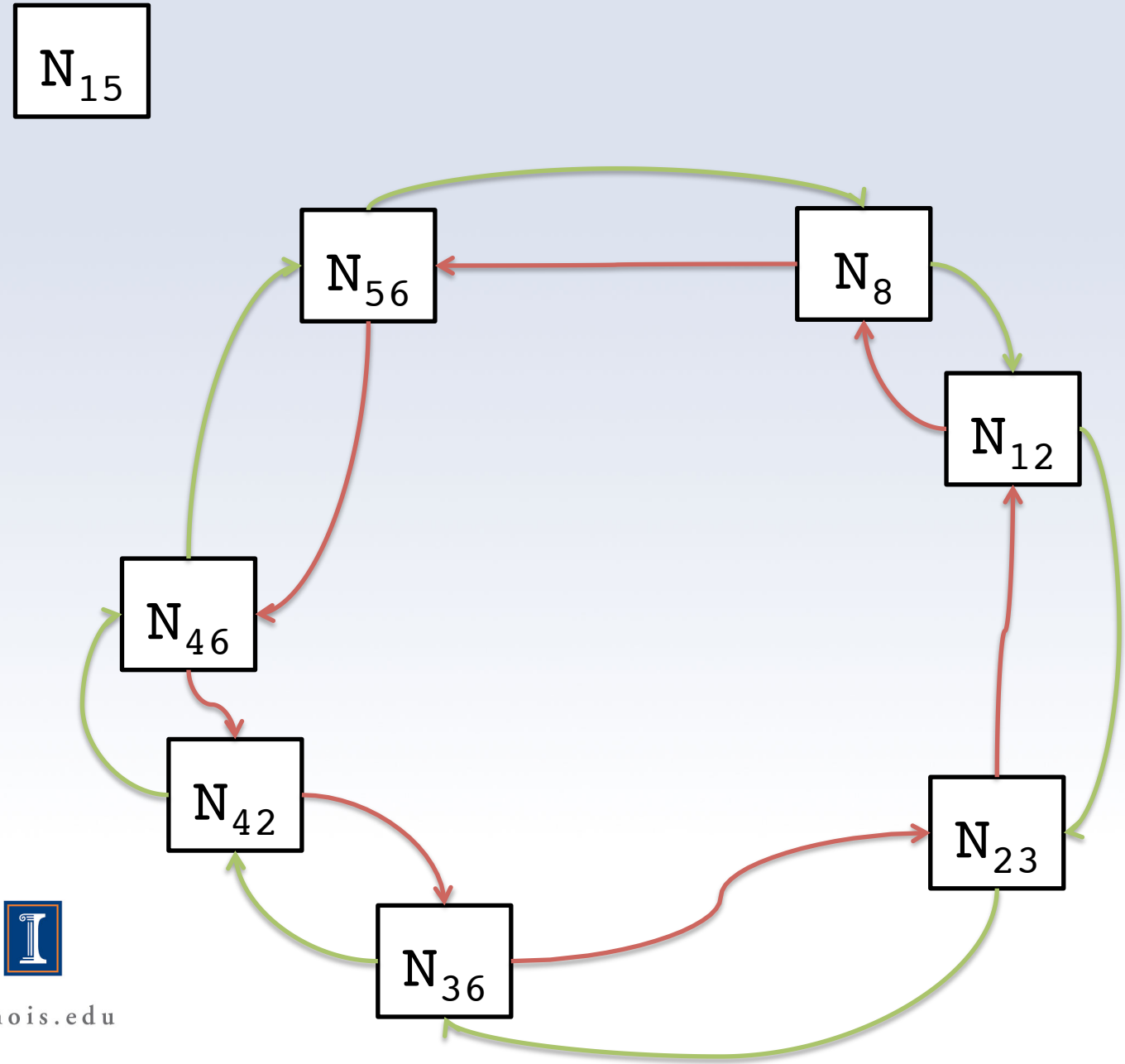
So…

- Given a chord-circle
  - We can search for a key
  - We know where to store keys and values
- But wasn't this supposed to be an ad-hoc peer-to-peer network?
- How do we add a node?

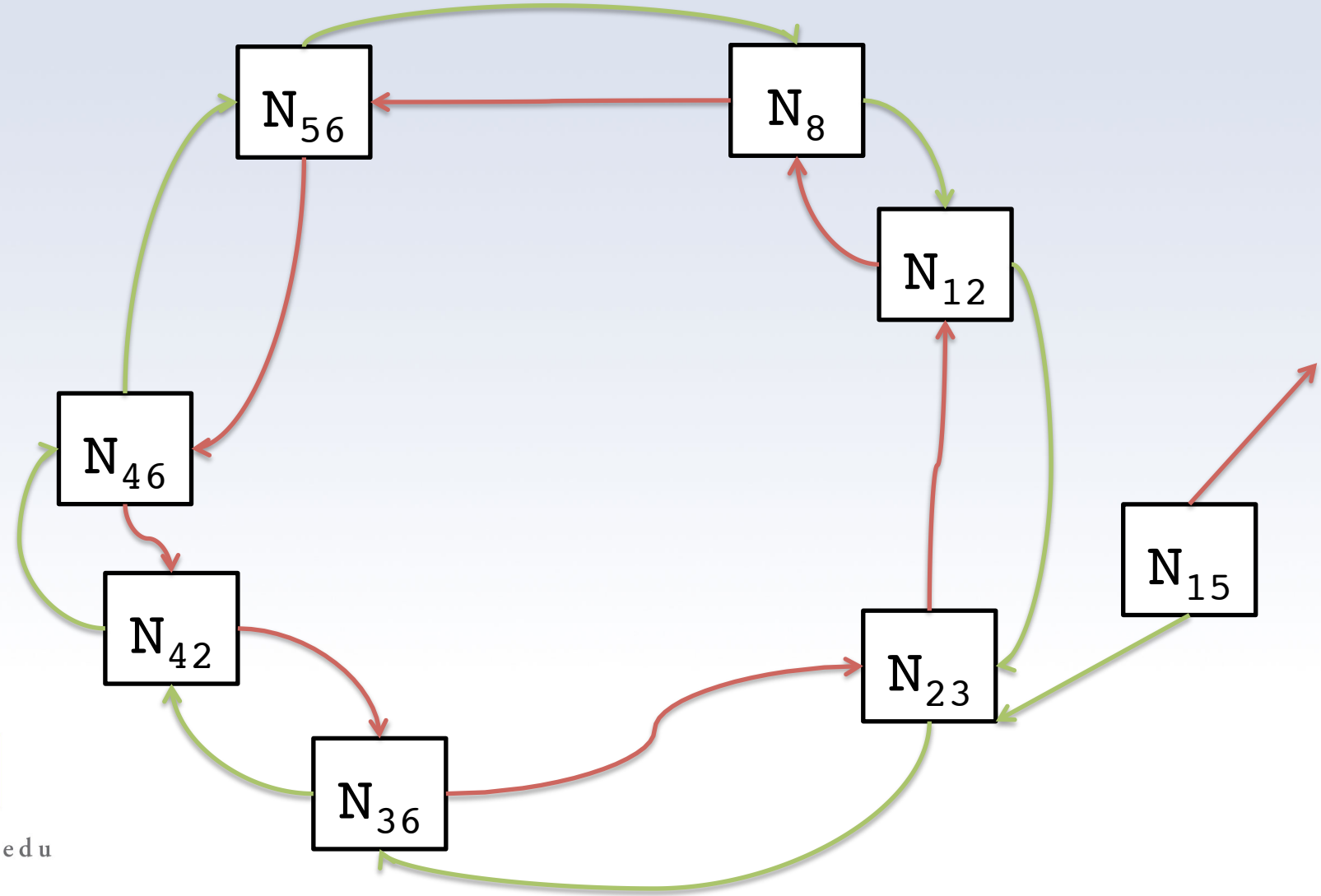
Adding a Node

1. Search for node's key value
2. Set successor to result
3. Leave predecessor as null

That's really it! We'll let the stabilization algorithm clean up our mess.



illinois.edu



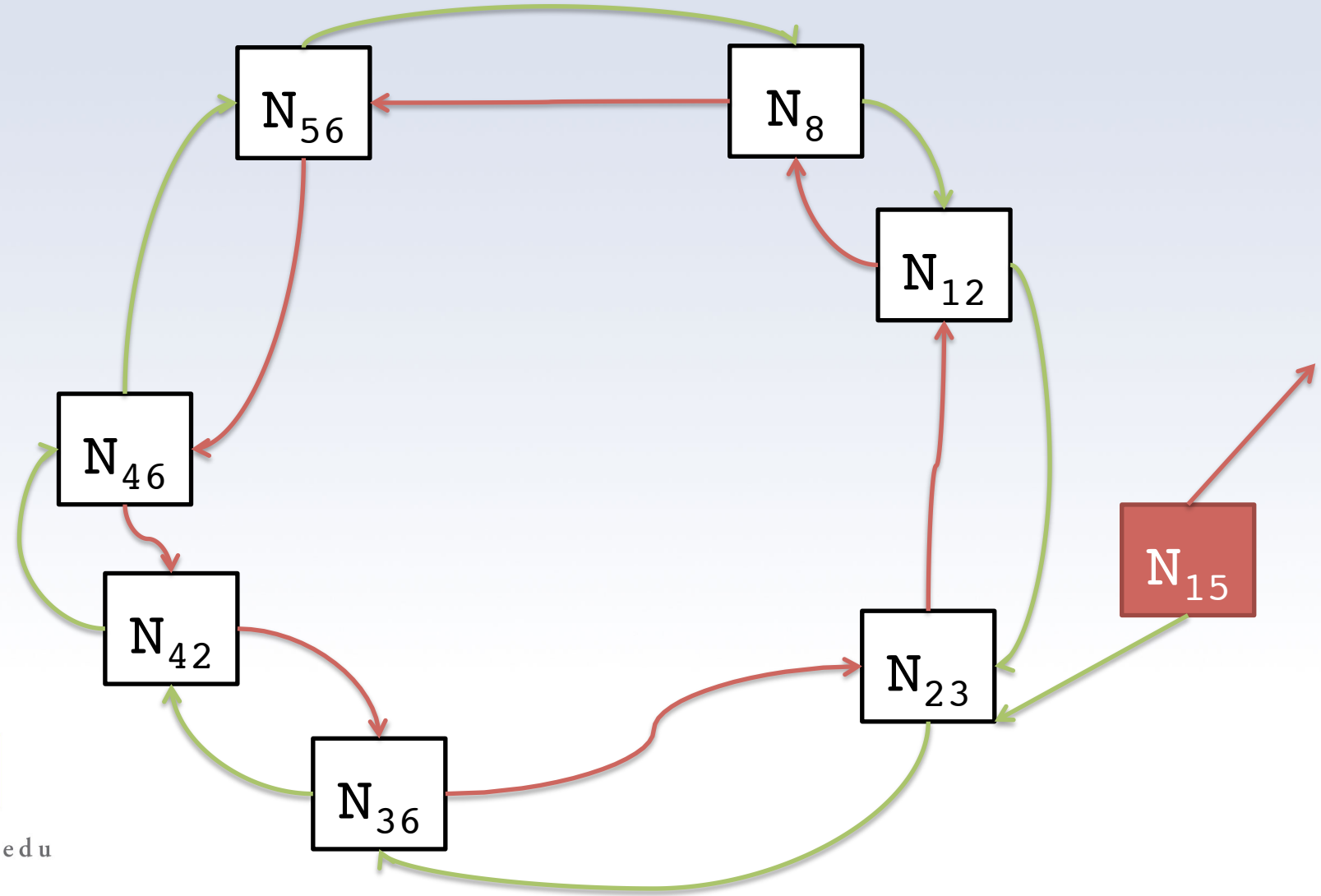
illinois.edu

Stabilization

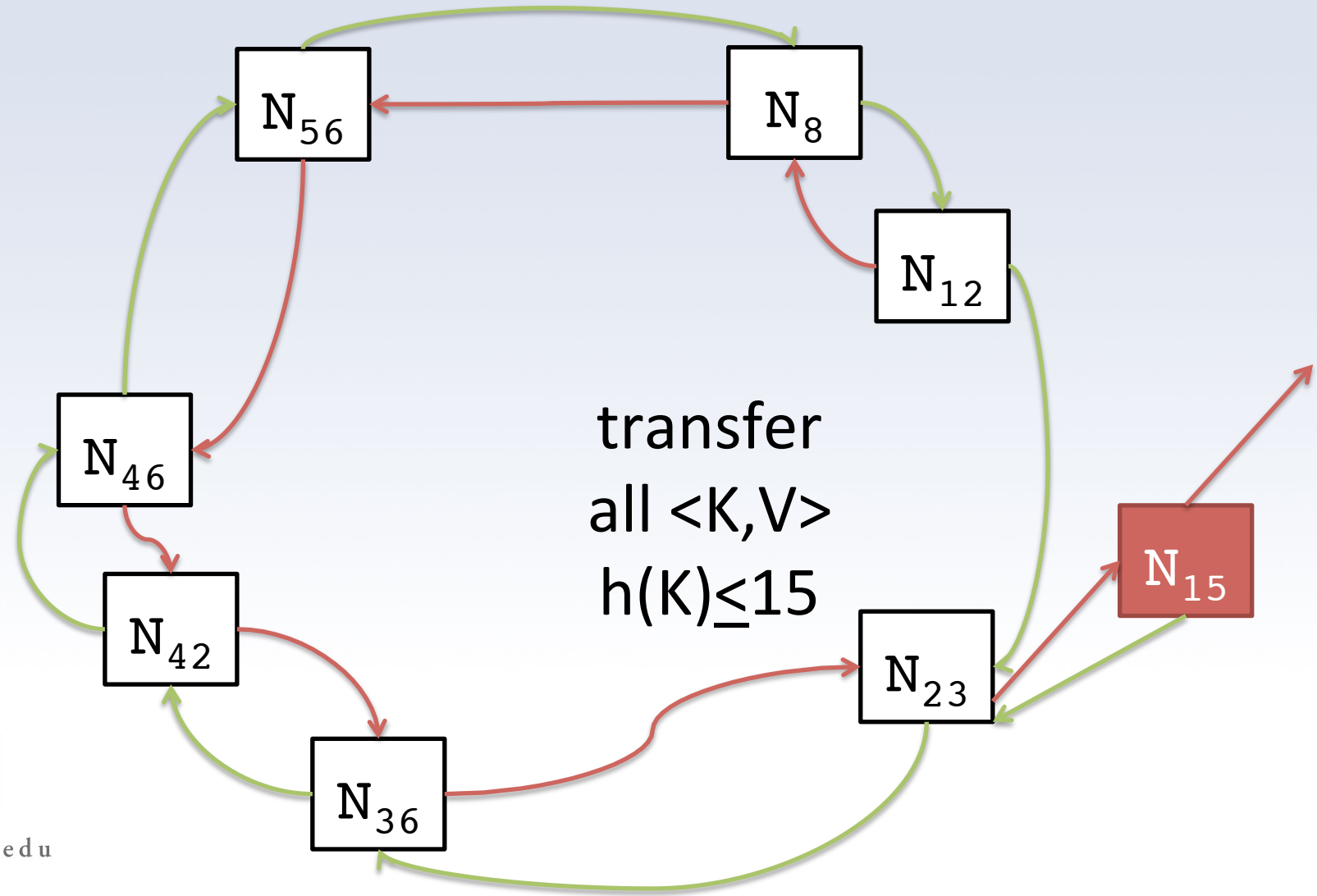
- Each node periodically updates successor and predecessor pointers
- Asynchronously restores/updates structure of chord circle

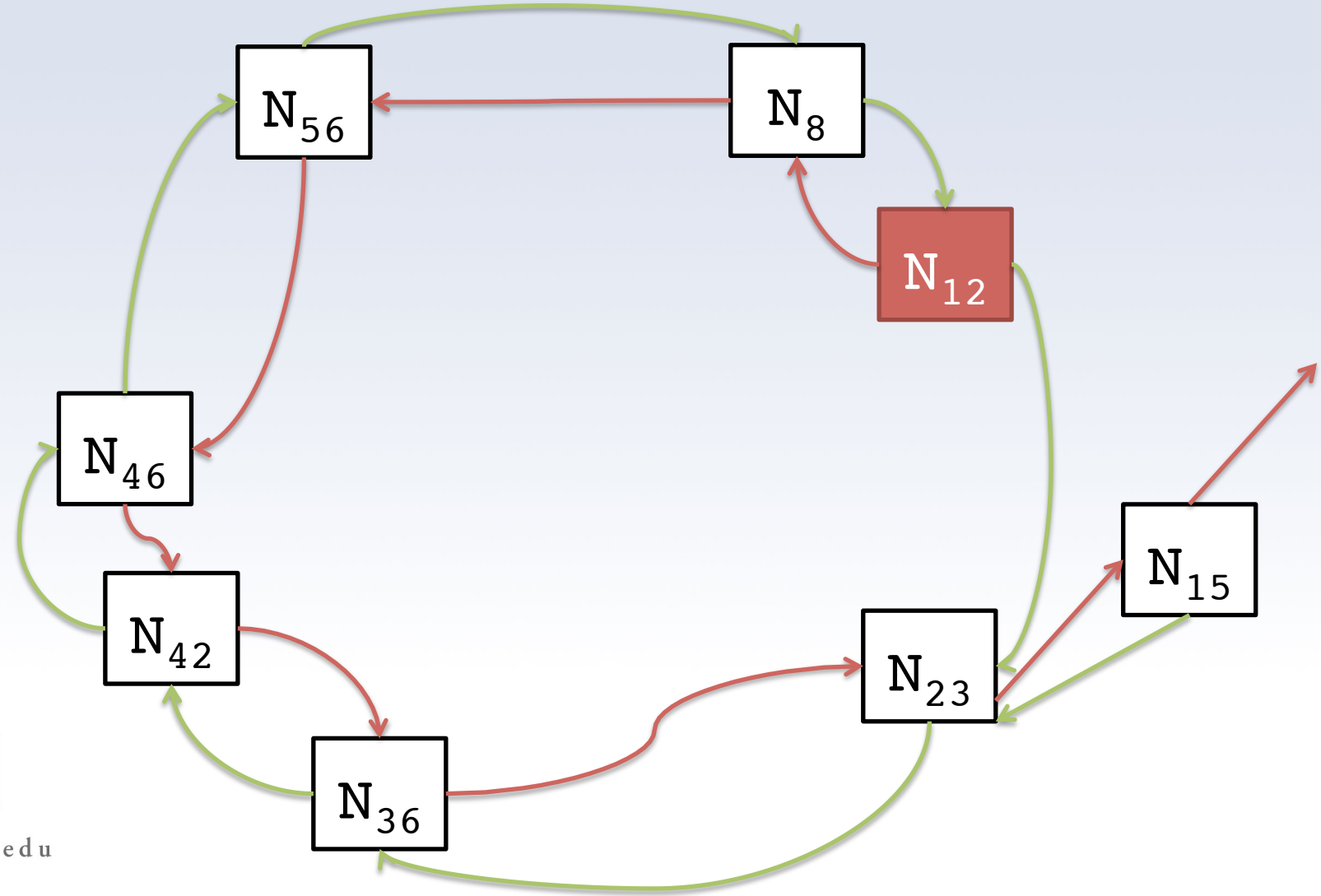
Stabilization

1. N asks S (its successor) for P (successor's predecessor).
   - If P=N, done.
2. If N<P<S, N records P as its successor.
3. If predecessor of S is nil, or S<N<P, S sets N as its predecessor.
4. S shares data with N.

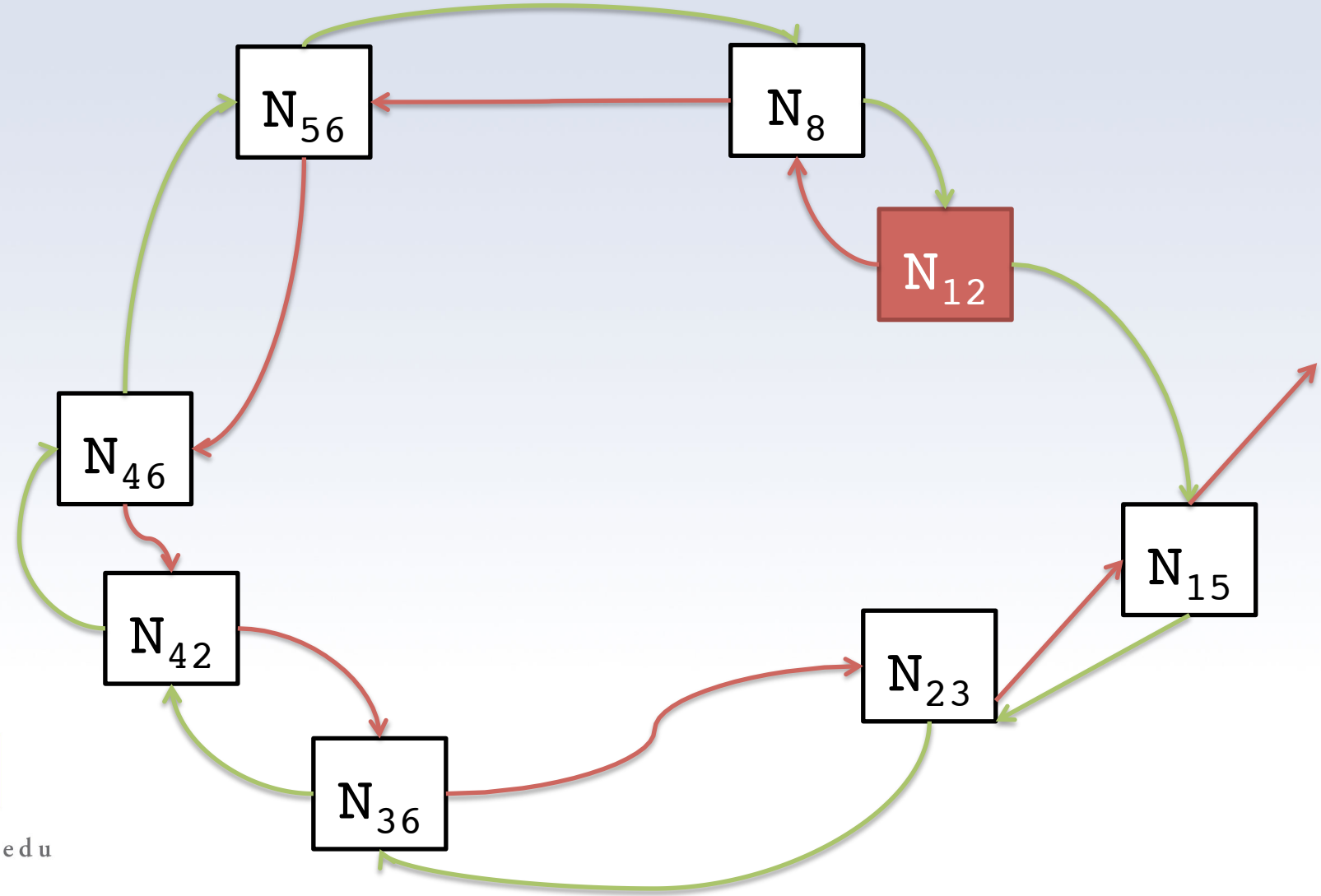


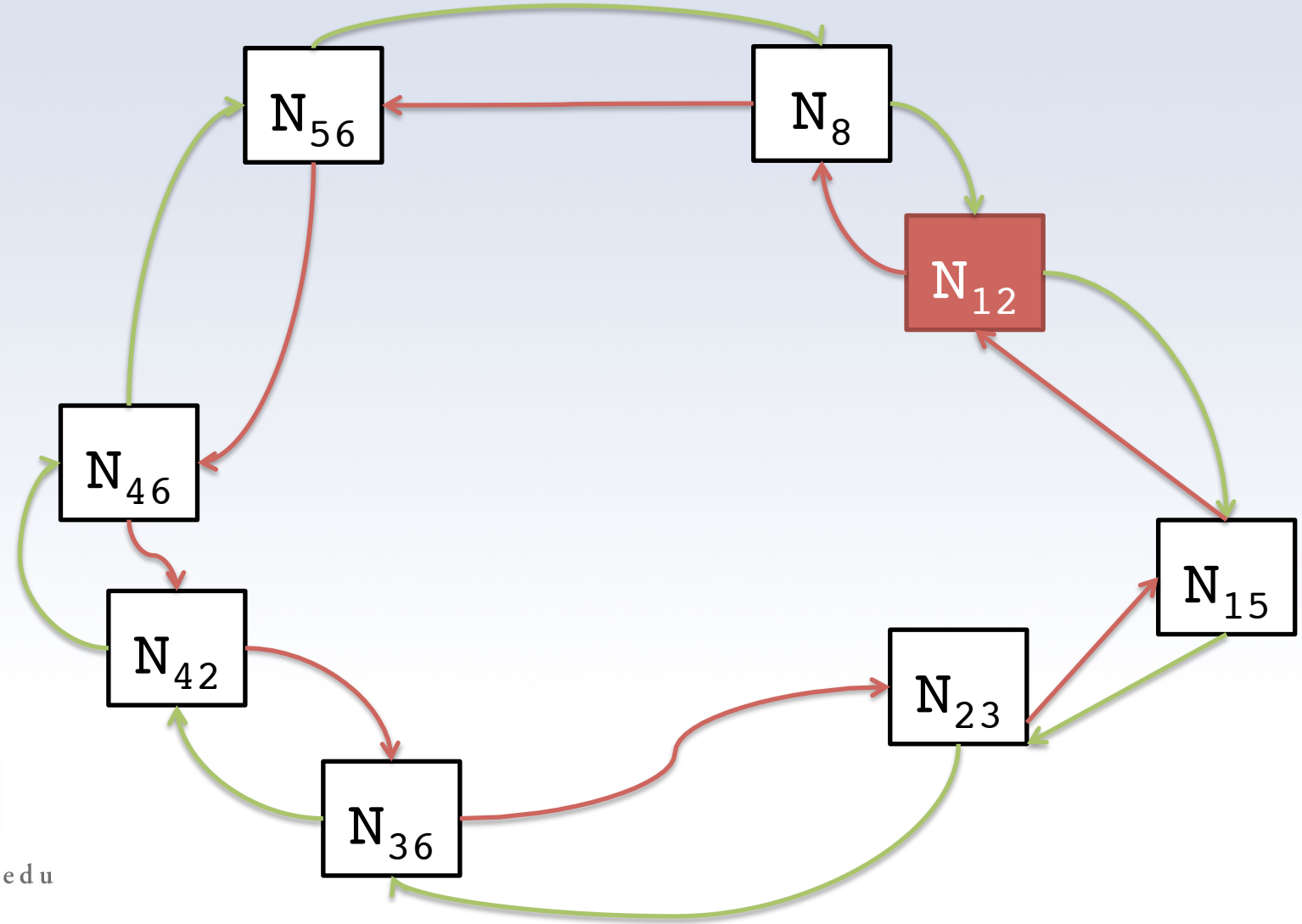
illinois.edu

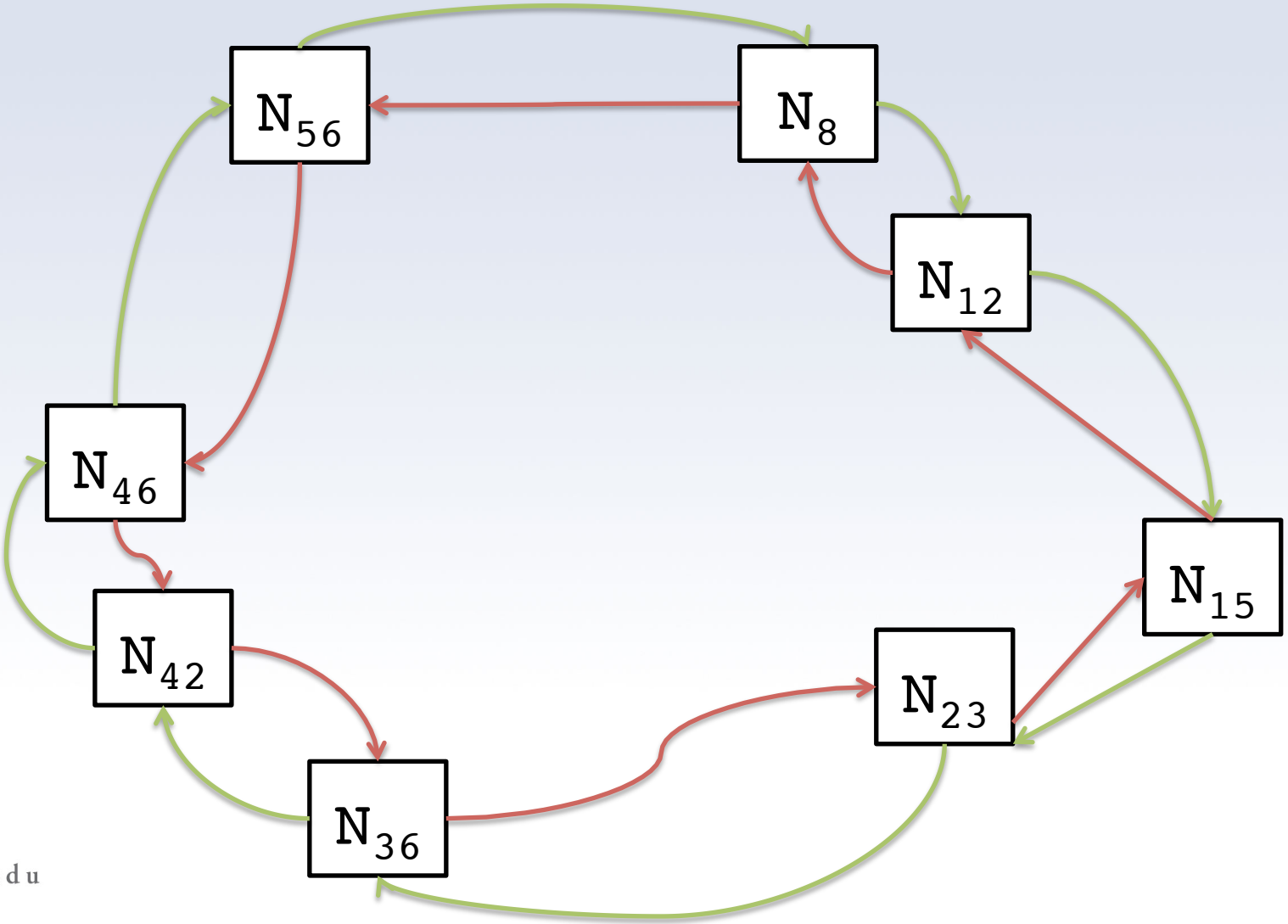

N56
N8
N12
N46
N42
N36
N23
N15
transfer
all <K,V>
h(K)≤15



illinois.edu


N56
N8
N12
N46
N15
N42
N23
N36


N56
N8
N12
N46
N15
N42
N23
N36

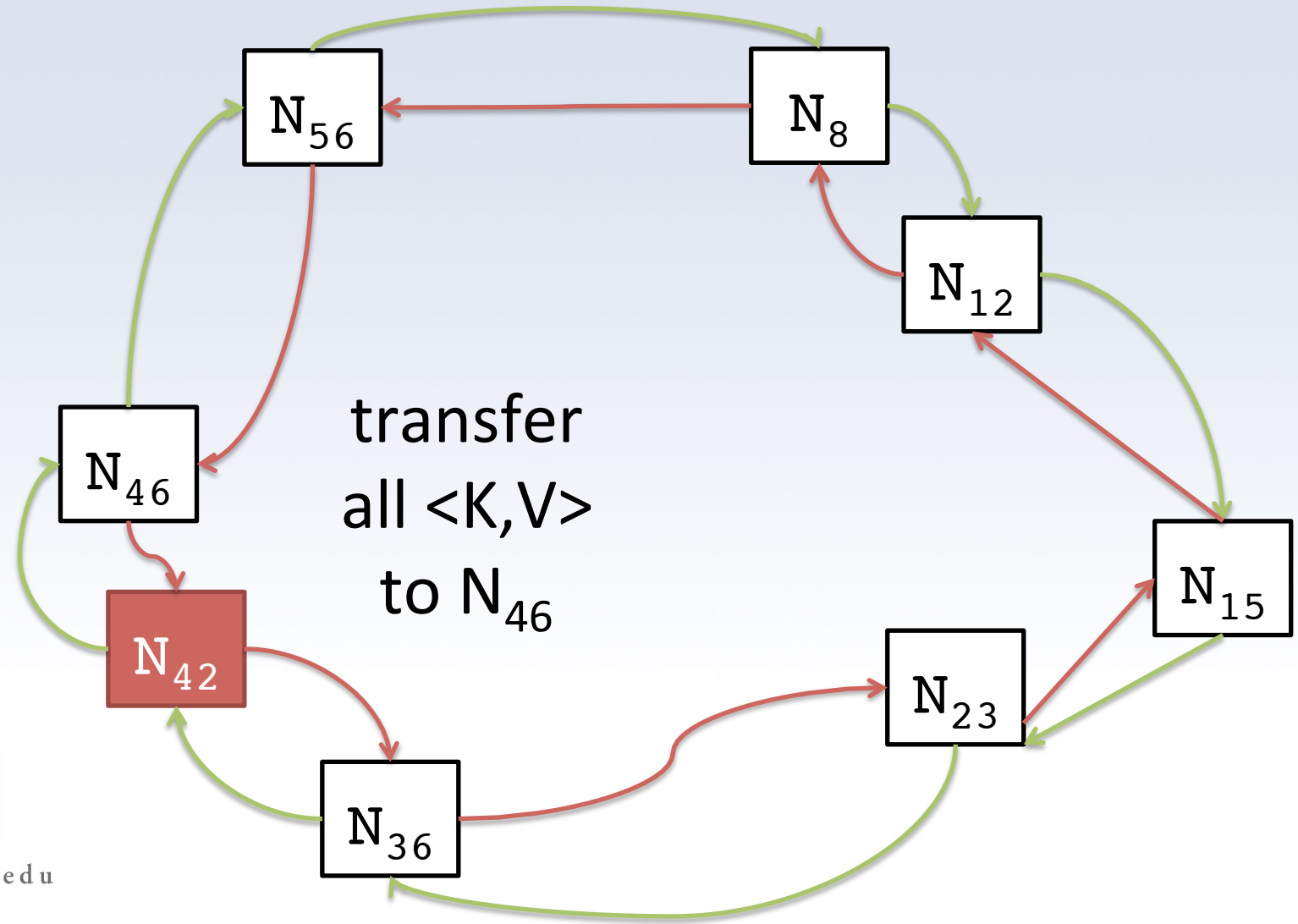

N56
N8
N12
N46
N15
N42
N23
N36

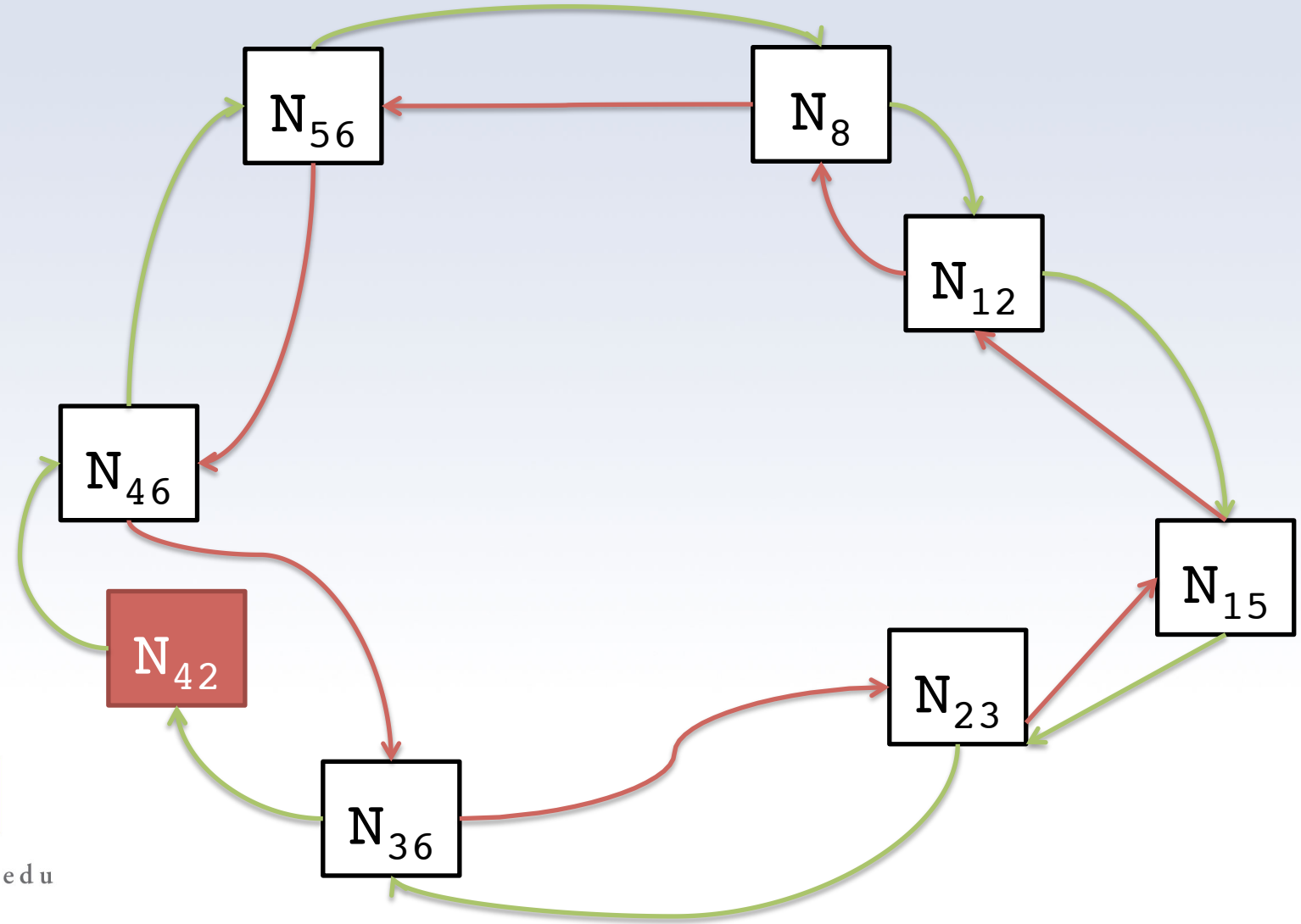
Stabilization

- Should also periodically update finger table
- Look up values and update based on results
- Entry for 1 is successor, so we always have a place to start

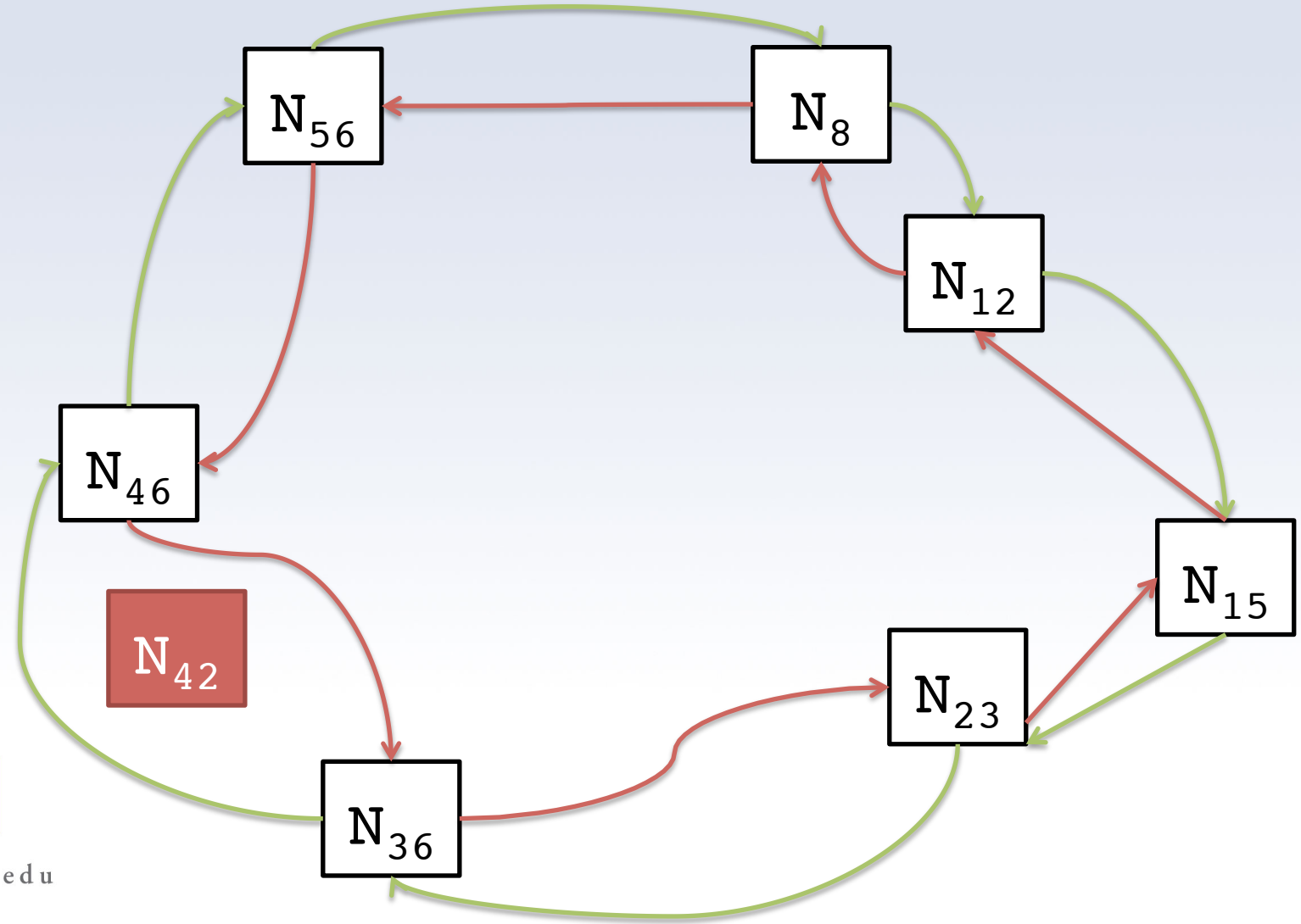
Leaving the network

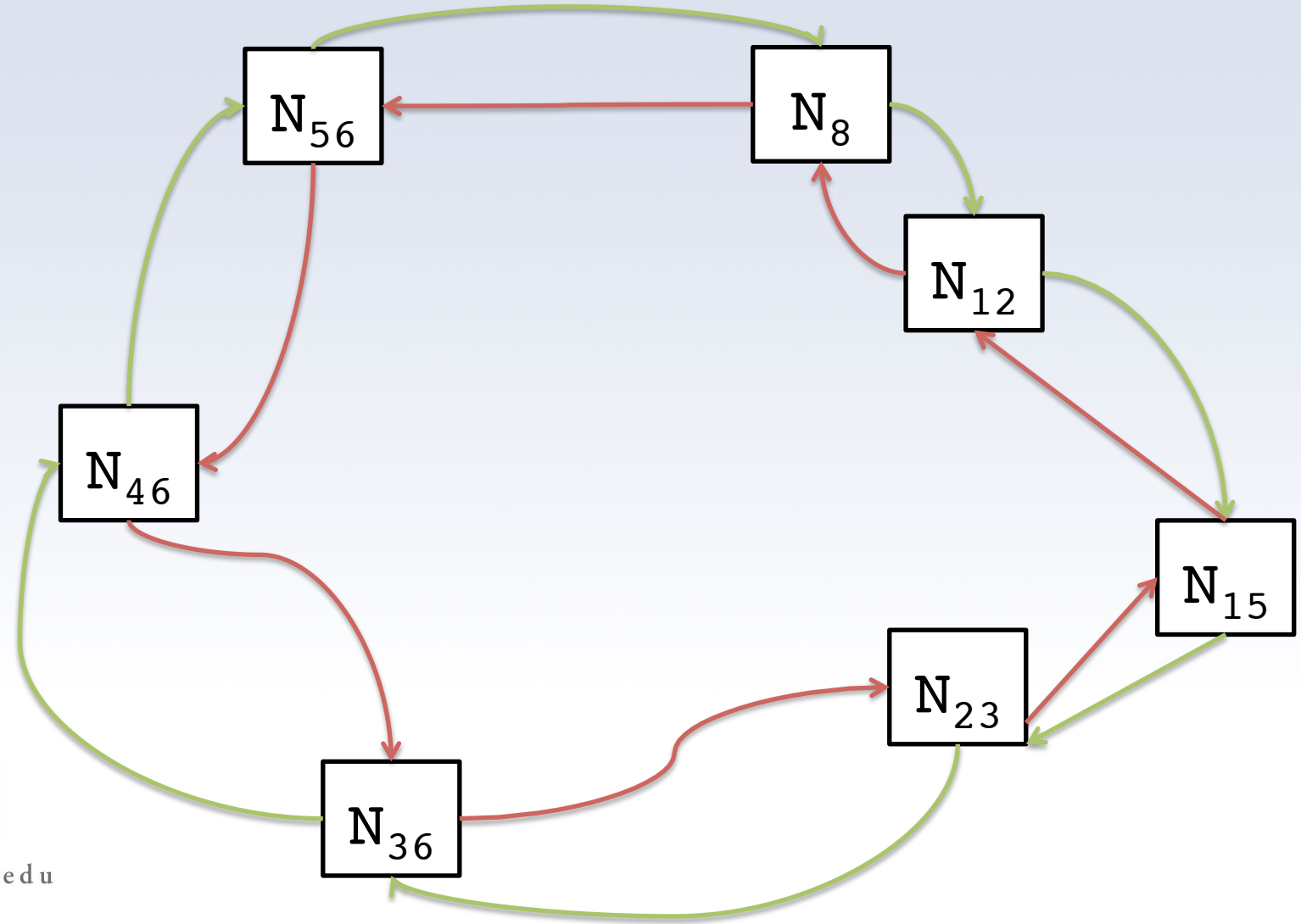
- Notify predecessor and successor
- Transfer data to successor


N56
N8
N12
N46
transfer
all <K,V>
to N46
N15
N42
N23
N36



illinois.edu


N56
N8
N12
N46
N15
N42
N23
N36


N56
N8
N12
N46
N15
N23
N36

Durability

- If a node fails, we lose
  - Our circle structure
  - Some of our <K,V> pairs
- Add redundancy
  - Successor's successor pointers
  - Predecessor's predecessor pointer
  - Replicate <K,V> pairs of successor and predecessor

If this is interesting…

- NoSQL/MapReduce/DHT
  - CS425 Distributed Systems
- SQL Injection/Hashing/Encryption
  - Computer Security/Applied Cryptography courses

Next time…

- We'll talk about some interesting data mining topics