IPADS INSTITUTE OF PARALLEL AND DISTRIBUTED SYSTEMS

上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

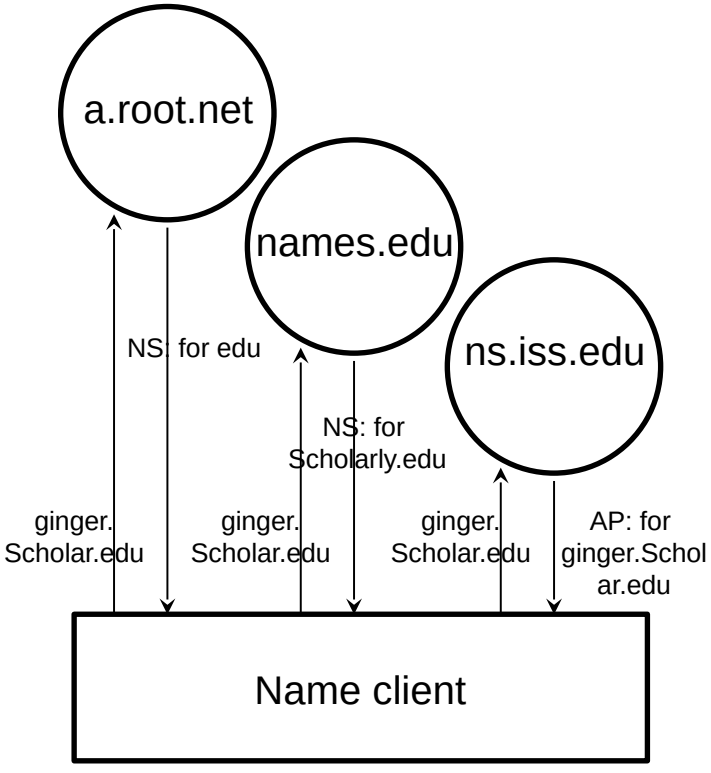
P2P Network

IPADS, Shanghai Jiao Tong University

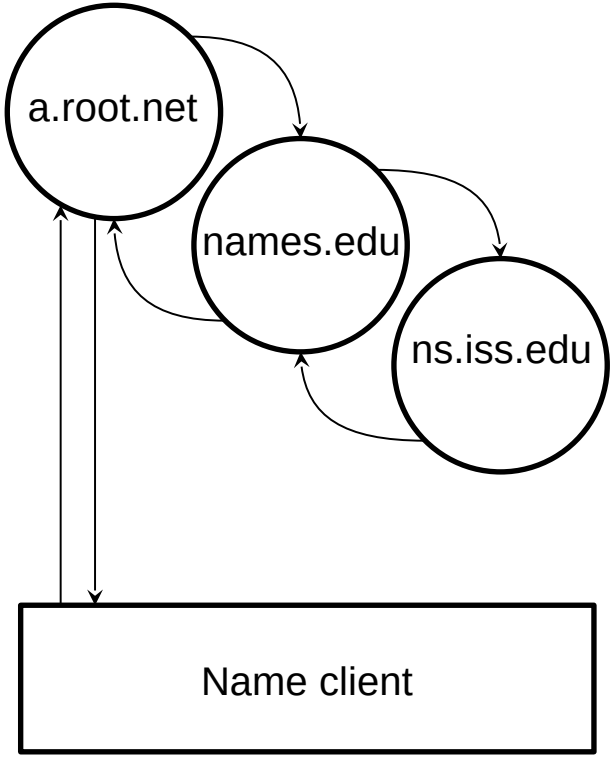
https://www.sjtu.edu.cn

Review: DNS Hierarchy (a partial view)

Review: DNS Request Process



Non-Recursion

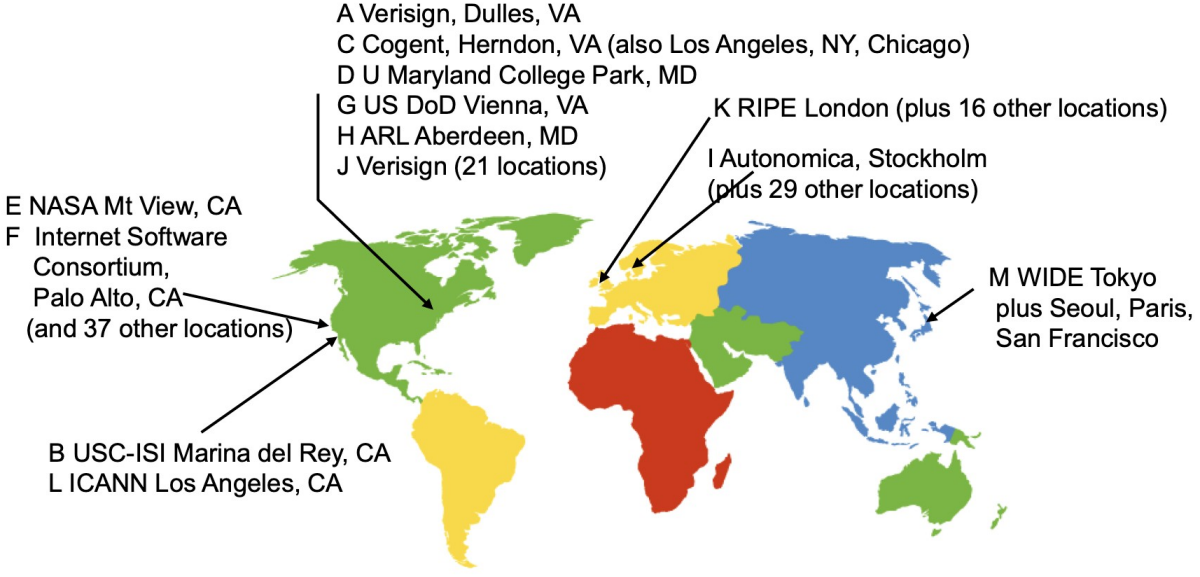


Recursion

Review: Other Features of DNS

At least two identical replica servers

- **80 replicas** of the root name server in 2008
- Replicas are placed separated around the world



Other Features of DNS

Organization's name server (e.g., SJTU)

- **Several replicas** in campus
  - To enable communications within the organization
- At least one **out of the campus**
  - To validate the address for outside world

Name Discovery in DNS (at the first place)

A client must discover the name of a nearby name server

- Name discovery broadcast to ISP at first time
- Ask network manager
- Ask by email, Google, etc.

Comparing Hostname & Filename

They are both for more user friendly

- File-name -> inode number
- Host-name -> IP address
- The file-name and host-name are **hierarchical**; inode num and IP addr. are **plane**

They are both not a part of the object

- File-name is not a part of a file (stored in directory)
- Host-name is not a part of a website (stored on name server)

Name and value binding

- File: 1-name -> N-values (no); N-name -> 1-value (yes)
- DNS: 1-name -> N-values (**yes**); N-name -> 1-value (yes)

Behind the DNS Design

Why was DNS designed in this way?

Benefits of Hierarchical Design

Hierarchies delegate responsibility

Each zone is only responsible for a small portion

Hierarchies also limit interaction between modules

- A type of **de-centralization**

Good Points on DNS Design

Global names (assuming same root servers)

- No need to specific a context
- DNS has no trouble generating unique names
- The name can also be user-friendly

Scalable in performance

- Simplicity: look-up is simple and can be done by a PC
- Caching: reduce number of total queries
- Delegation: many name severs handle lookups

Good Points on DNS Design

Scalable in management

- Each zone makes its own policy decision on binding
- Hierarchy is great here

Fault tolerant

- If one name server breaks, other will still work
- Duplicated name server for a same zone

Bad Points on DNS Design

Policy

- Who should control the root zone, .com zone, etc.? Governments?

Significant load on root servers

- Many DNS clients starts by talking to root server
- Many queries for non-existent names, becomes a DoS attack

Security

- How does a client know if the response is correct?
- How does VeriSign know "change Amazon.com IP" is legal?

Naming Scheme

Naming: the glue of modules

Naming in General

- ipads.se.sjtu.edu.cn – hostname
- steven@apple.com - email
- steven – username
- EAX - x86 processor register name
- main() - function name
- WebBrowser - class name
- /courses/cse/index.html - path name (fully-qualified)
- index.html - path name (relative)
- http://ipads.se.sjtu.edu.cn/courses/cse/index.html - URL
- 13918275839- Phone number
- 202.120.40.188 - IP Address

Naming a Disk

File name: /dev/sda1

- As a special type of inode: device inode
- 8,0 as (major, minor)

PCI address (name): 19:00.0

- SCSI storage controller: LSI Logic / Symbios Logic SAS1068E PCI-Express Fusion-MPT SAS (rev 08)

Naming for Modularity

Retrieval: e.g., using URL to get a web page

Sharing: e.g., passing an object reference to a function

- Save space as well: only sending the name, not the object

Hiding: e.g., using a file name without knowing file system

- Can support access control: use an object only if knowing its name
- E.g., Windows has many undocumented API

User-friendly identifiers: e.g., “homework.txt” instead of 0x051DE540

Indirection: e.g., OS can move the location of the file data without notifying the user

- Have you ever defragmented your hard driver?

Addresses as Names

Software uses these names in an obvious way

- E.g., memory addresses

Hardware modules connected to a bus

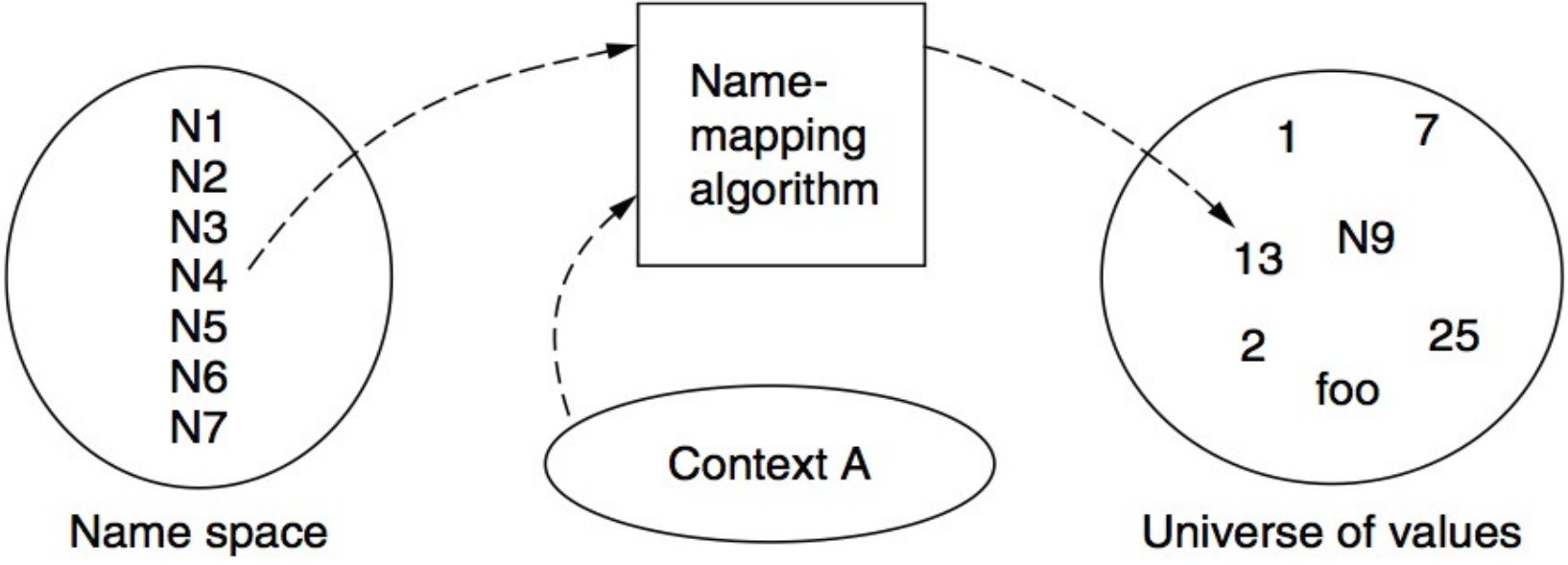
- Use bus addresses (a kind of name) for interconnection

Naming Schemes

A naming schemes contains three parts:

- 1. Set of all possible **names**
  - You cannot use ‘for’ as a variable in C
- 2. Set of all possible **values**
- 3. **Look-up algorithm** to translate a name into a value
  - or a set of values, or “none”

Naming Model



Naming Terminology

Binding – A mapping from a name to value

- Unbind is to delete the mapping
- A name that has a mapping is bound

A name mapping algorithm resolves a name

Naming Context

Type-1: context and name are separated

- E.g., inode number’s context is the file system

Type-2: context is part of the name

- E.g., xiayubin@sjtu.edu.cn

Name spaces with only one possible context are called universal name spaces

- Example: credit card number, UUID, email address

Determining Context - 1

Hard code it in the resolver

- Examples: Many universal name spaces work this way

Embedded in name itself

- cse@sjtu.edu.cn:
  - Name = “cse”
  - Context = “sjtu.edu.cn”
- /ipads.se.sjtu.edu.cn/courses/cse/README :
  - Name = “README”
  - Context = “/ipads.se.sjtu.edu.cn/courses/cse”

Determining Context - 2

Taken from environment (Dynamic)

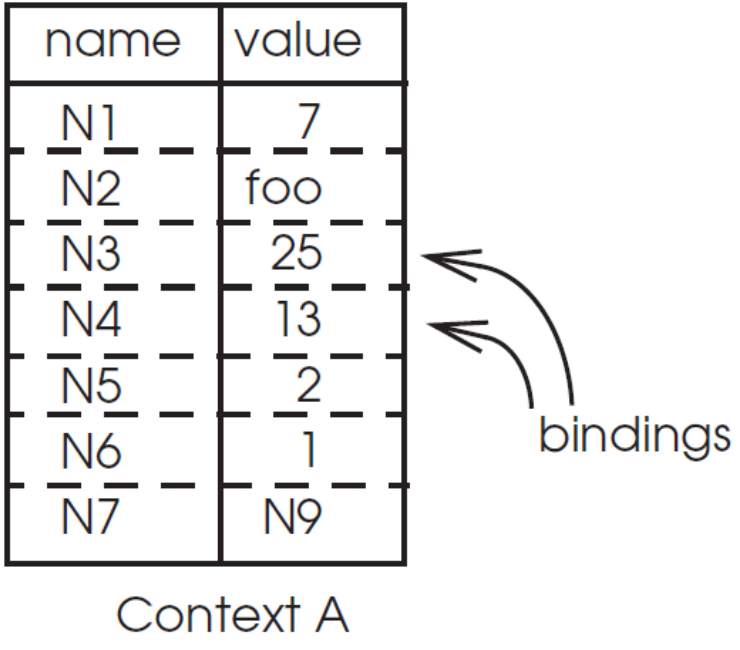
- Unix cmd: “**rm foo**”:
  - Name = “foo”, context is current dir
  - **Question**: how to find the binary of “rm” command?
- Read memory 0x7c911109:
  - Name = “0x7c911109”,
  - Context is thread’s address space

Many errors in systems due to using wrong context

Name Mapping Algorithms - 1

Table lookup

- Find name in a table
  - E.g., Phone book
- Context: which table?
  - Implicit VS. explicit
  - Default context



Name Mapping Algorithms - 2

Recursive lookup

- E.g., “/usr/bin/rm”
- First find “usr” in “/”, then find “bin” in “/usr”, then “rm”
- Each look-up process is the same

Multiple lookup

- **Recall**: how to find “rm” without absolute name?
- $PATH
  - E.g., “/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin”
- Look-up in a predefined list of context

Interpreter Naming API

value ← **RESOLVE(*name*, *context*)**

– Return the mapping of *name* in the *context*

status ← **BIND(*name*, *value*, *context*)**

– Establish a *name* to *value mapping* in the *context*

status ← **UNBIND(*name*, *context*)**

– Delete name from *context*

list ← **ENUMERATE(*context*)**

– Return a list of all bindings

result ← **COMPARE(*name1*, *name2*)**

– Check if *name1* and *name2* are equal

FAQ of Naming Scheme - 1

What is the syntax of names?

What are the possible value?

What context is used to resolve names?

Who specifies the context?

Is a particular name global (context-free) or local?

FAQ of Naming Scheme - 2

Does every name have a value?

- Or, can you have “dangling” names?

Can a single name have multiple values?

Does every value have a name?

- Or, can you name everything?

Can a single value have multiple names?

- Or, are there synonyms?

Can the value corresponding to a name change over time?

P2P Network

Downsides of Centralized Infrastructure

Centralized point of failure

High management costs

- If one org has to host millions of files, etc.

Not suitable for many scenarios

- E.g., cooperation between you and me

Lack ability to aggregate clients

P2P (Peer-to-peer): No central servers!

How to track nodes and objects in the system?

How do you find other nodes in the system?

How should data be split up between nodes?

How to prevent data from being lost?

- How to keep it available?

How to provide consistency?

How to provide security? anonymity?

BitTorrent

Usage Model: Cooperative

- User downloads file from someone using simple user interface
- While downloading, BitTorrent serves file also to others
- BitTorrent keeps running for a little while after download completes

3 Roles

- *Tracker*: What peer serves which parts of a file
- *Seeder*: Own the whole file
- *Peer*: Turn a seeder once has 100% of a file

BitTorrent

Publisher a .torrent file on a Web server (e.g., suprnova.org)

- URL of tracker, file name, length, SHA1s of data blocks (64-512Kbyte)

Tracker

- Organizes a swarm of peers (who has what block?)

Seeder posts the URL for .torrent with tracker

- Seeder must have complete copy of file

Peer asks tracker for list of peers to download from

- Tracker returns list with random selection of peers

Peers contact peers to learn what parts of the file they have etc.

- Download from other peers

A torrent file

```
{
   'announce': 'http://bttracker.debian.org:6969/announce',
   'info':
   {
      'name': 'debian-503-amd64-CD-1.iso',
      'piece length': 262144,
      'length': 678301696,
      'pieces':
          '841ae846bc5b6d7bd6e9aa3dd9e551559c82abc1...d14f1631d

           776008f83772ee170c4241161819 0a4'
   }
}
```

Which Piece to Download?

Different policies: Order of parts downloading

- Strict?
- Rarest first?
- Random?
- Parallel?

BitTorrent

- Random for the first one
- Rarest first for the rest
- Parallel for the last one

Drawback of BitTorrent

Rely on Tracker

- Tracker is central component
- Cannot scale to large number of torrents

Scalable Lookup: DHT

DHT: Distributed hash table

- Used to find the node responsible for a key quickly
- Distributed index: each node keeps a subset of the index

Typical DHT interface:

- put(key, value)
- get(key) -> value
- Loose guarantees about keeping data alive

P2P Implementation of DHT

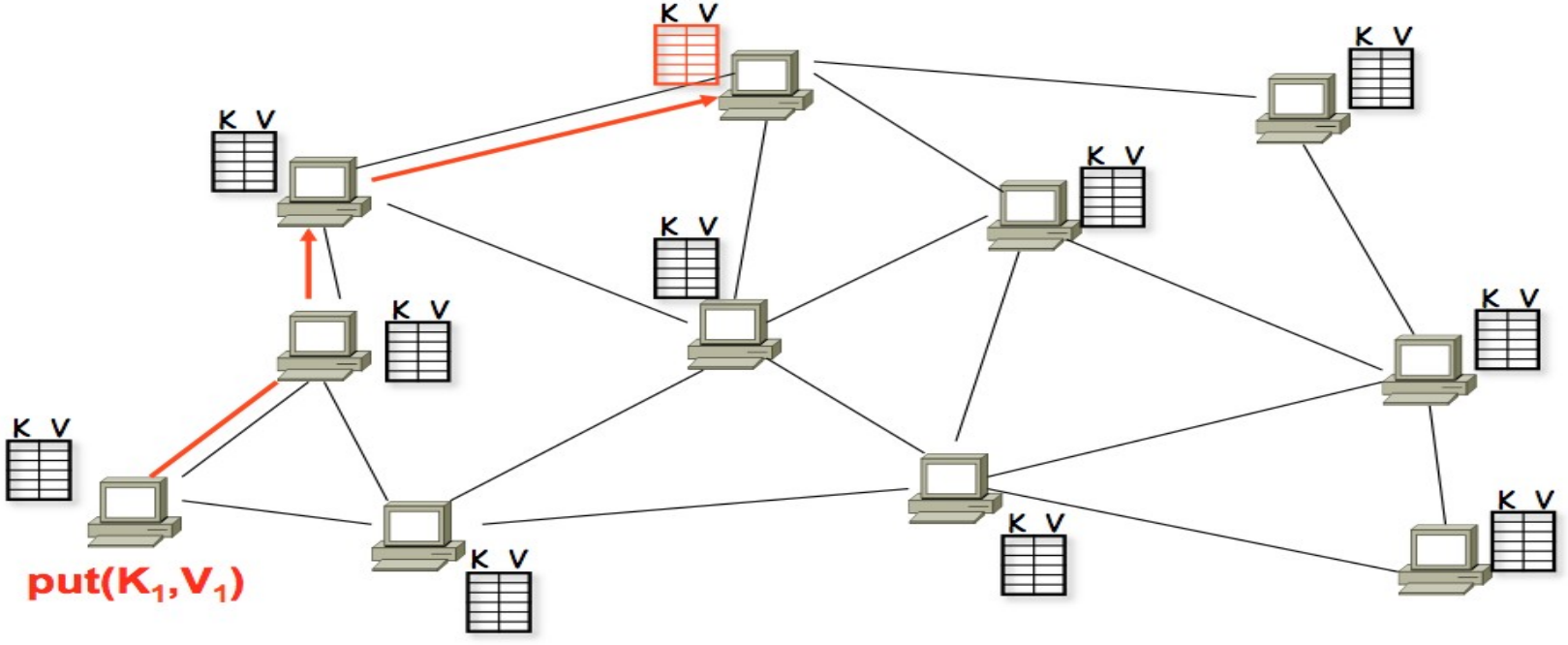
Overlay Network

- Partition hash table over n nodes
- Not every node knows about all other n nodes
- Route to find right hash table

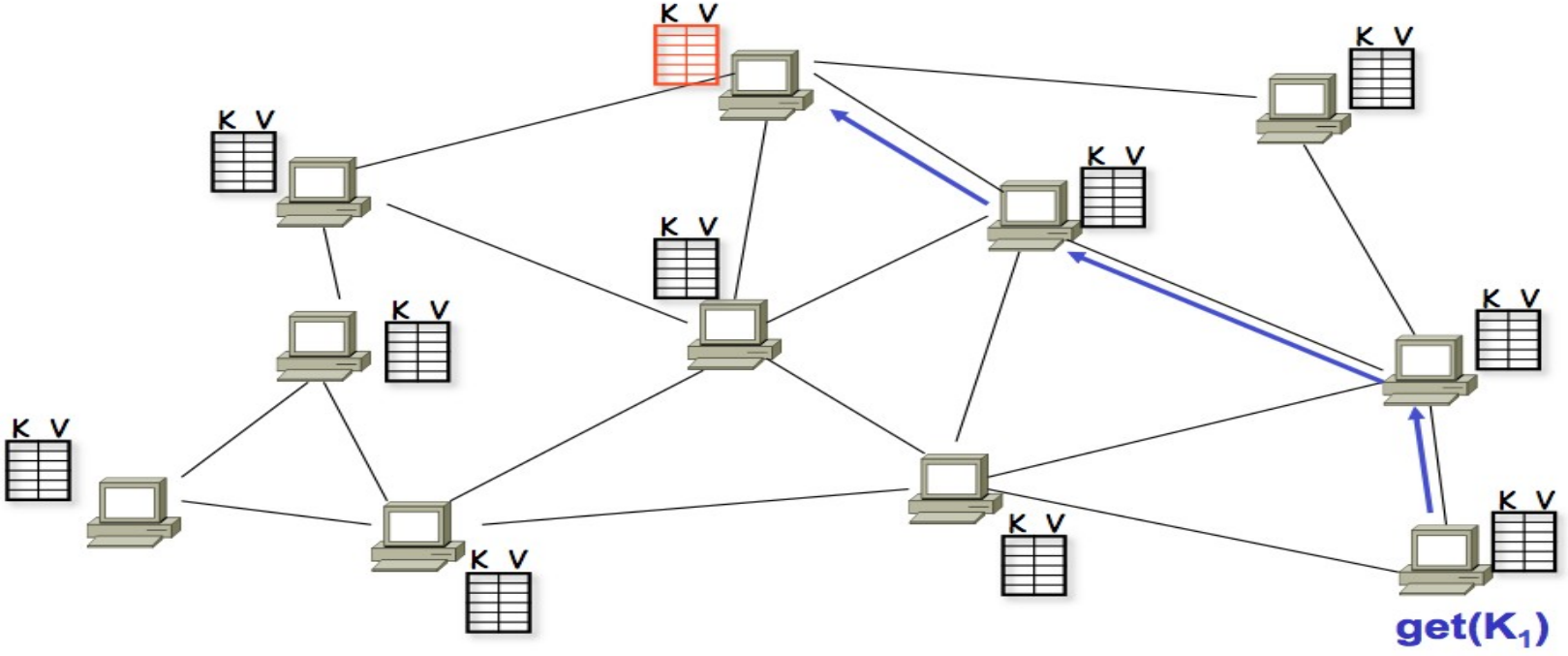
Goals

- log(n) hops
- Guarantees about load balance

A DHT in Operation: put()



A DHT in Operation: get()



Chord Properties

Efficient: O(*log(N)*) messages per lookup

- N is the total number of servers

Scalable: O(*log(N)*) state per node

Robust: survives massive failures

Chord IDs

Key identifier = SHA-1(key)

- SHA-1 is a hash function

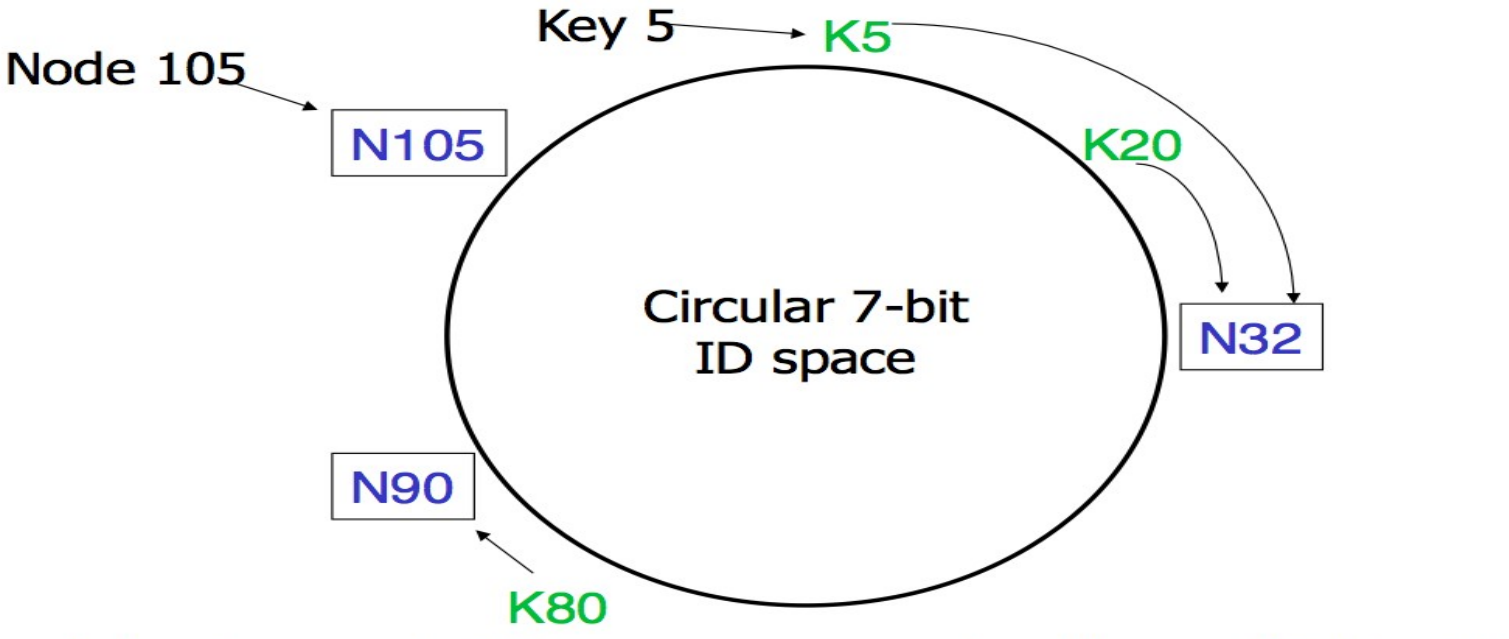
Node identifier = SHA-1(IP address)

Both are uniformly distributed

Both exist in the same ID space

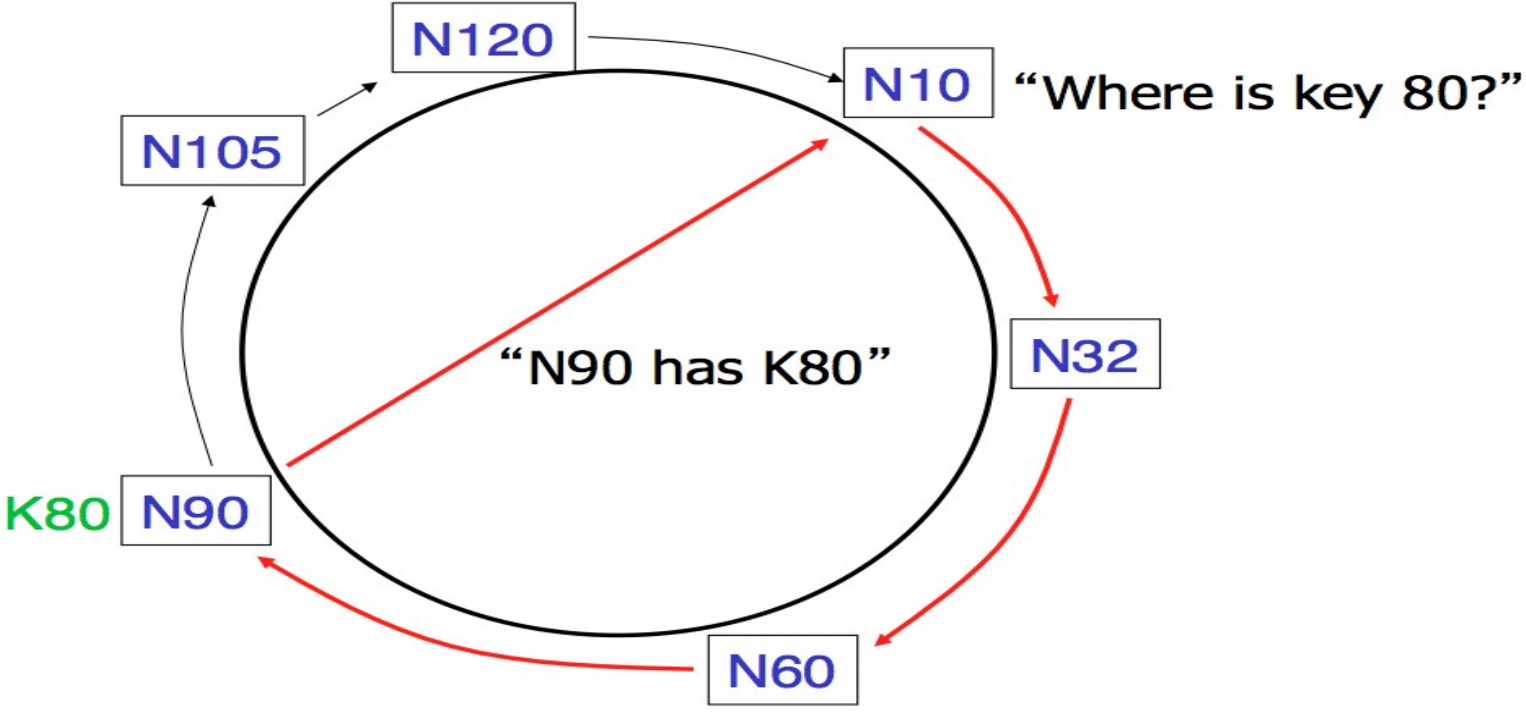
How to map key IDs to node IDs? by mod?

Consistent Hashing



A key is stored at its successor: node with next higher ID

Basic Lookup

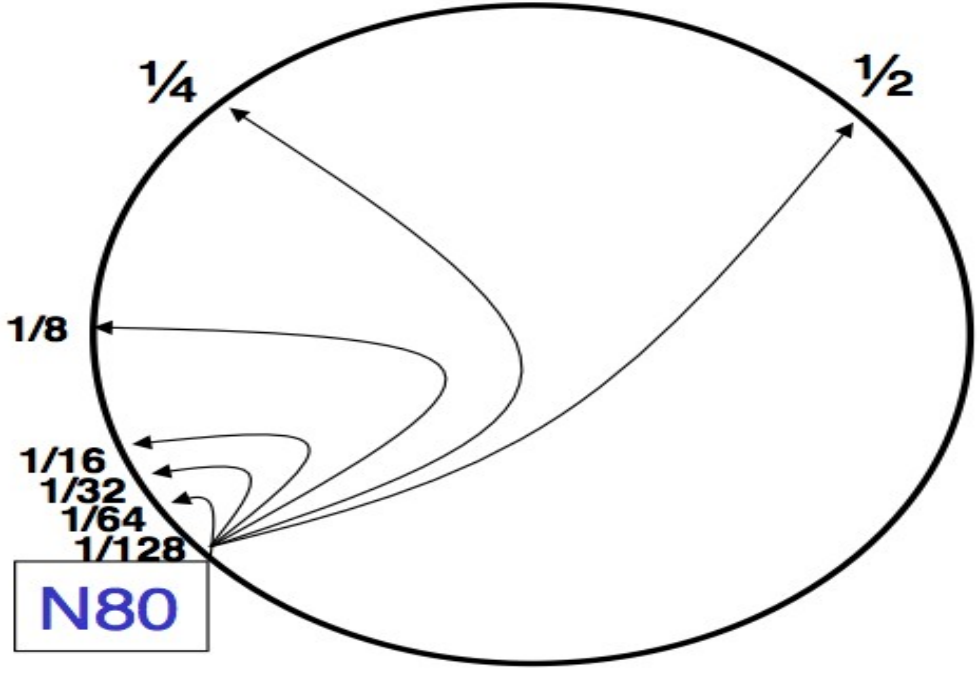


Simple Lookup Algorithm

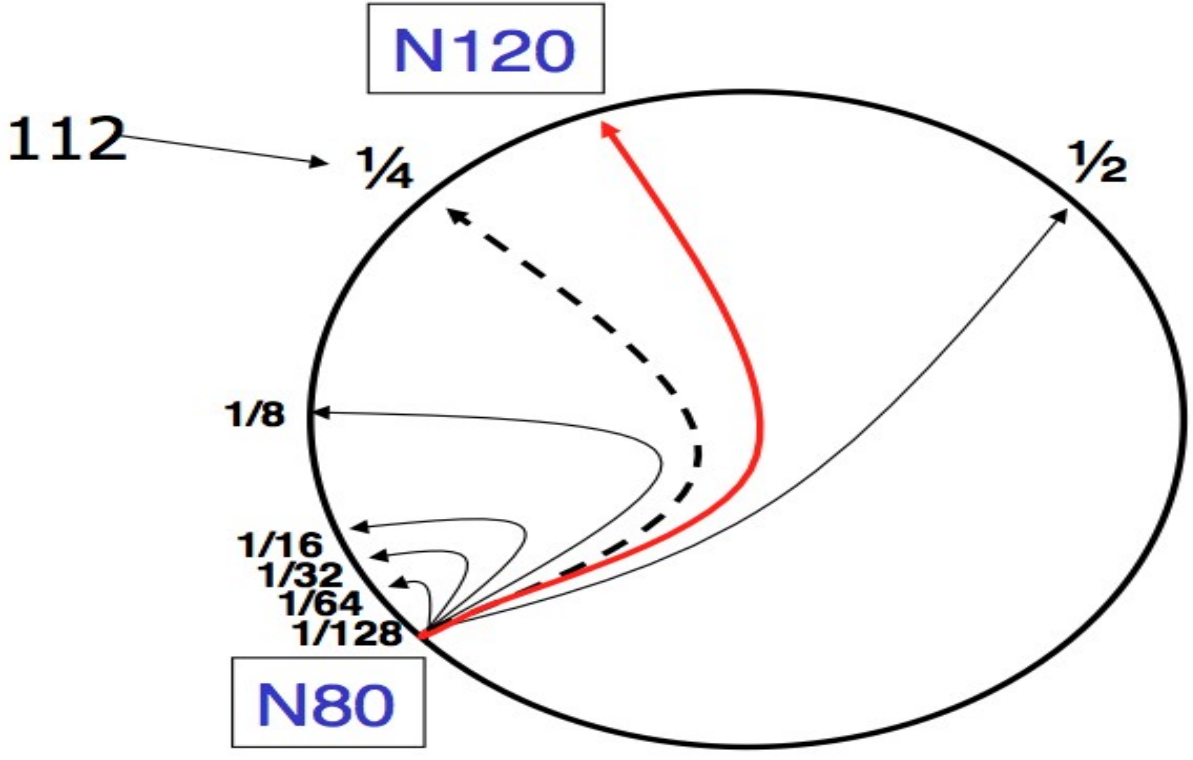
```
Lookup(my-id, key-id)
    n = my successor
    if my-id < n < key-id
        call Lookup(id) on node n    // next hop
    else
        return my successor              // done
```

Correctness depends only on successors

"Finger Table" allows log(N) lookups



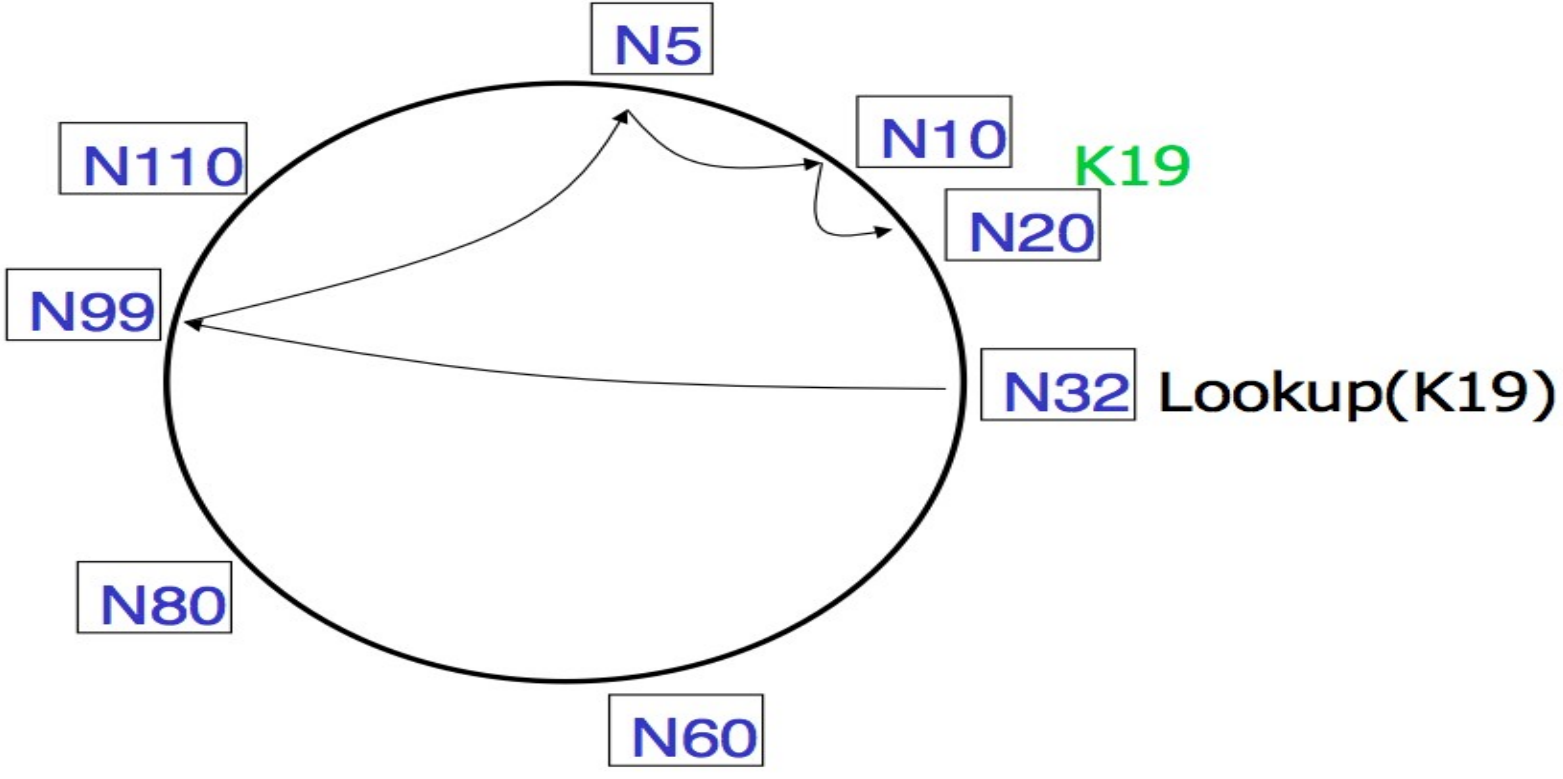
Finger i points to successor of $n+2^i$



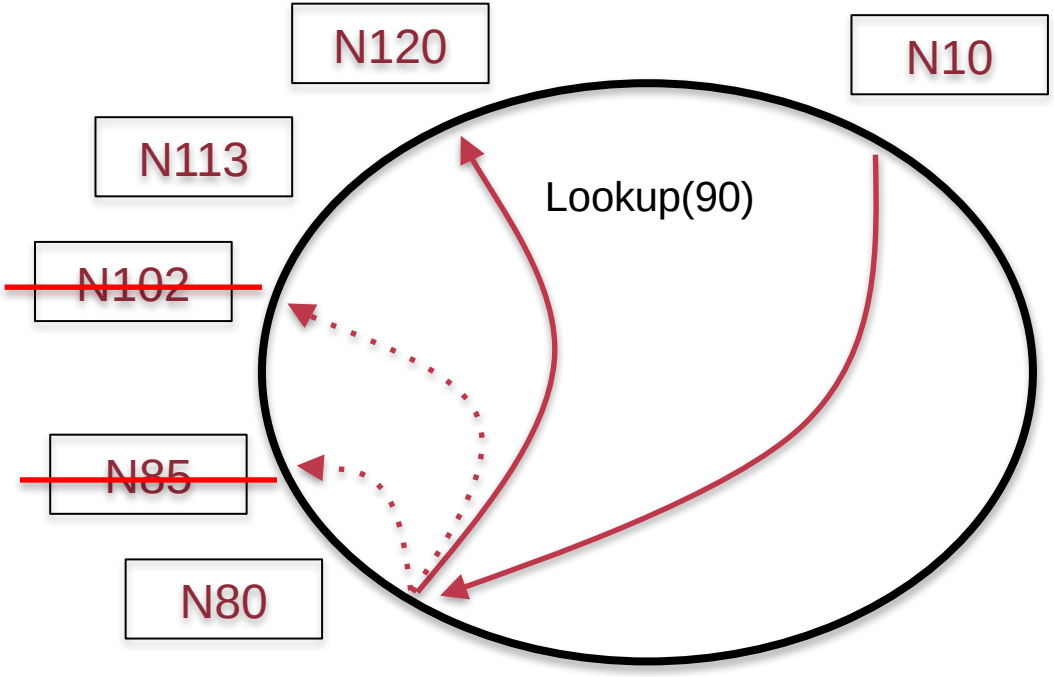
Lookup with fingers

```
Lookup(my-id, key-id)
    look in local finger table for
        highest node n s.t. my-id < n < key-id
    if n exists
        call Lookup(id) on node n    // next hop
    else
        return my successor              // done
```

Lookups take O(*log(N)*) hops



Failures might cause incorrect lookup



N80 doesn't know correct successor, so incorrect lookup

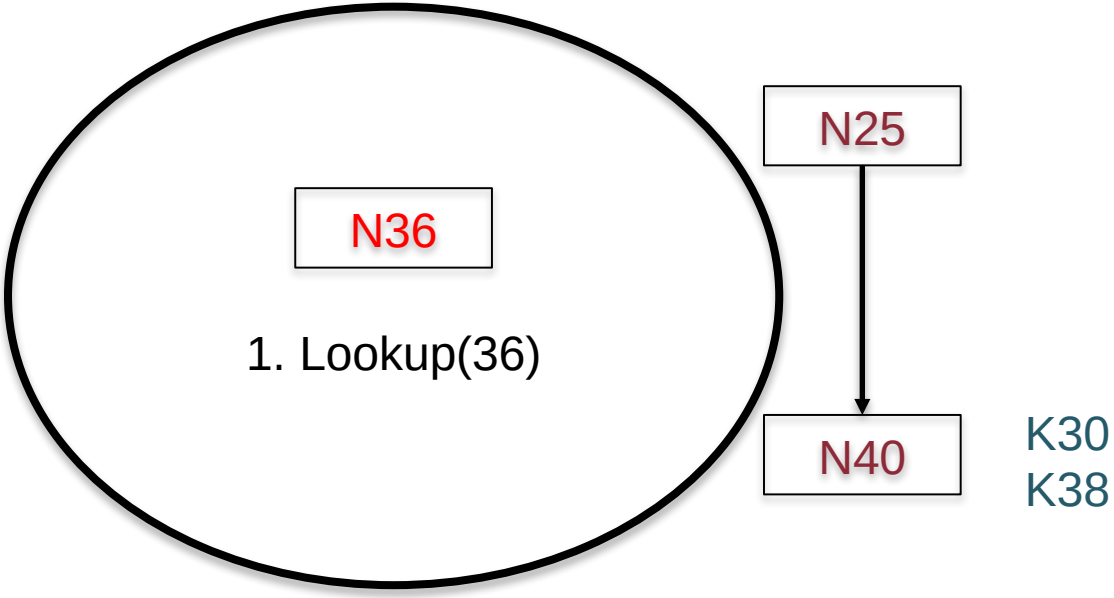
Solution: successor lists

Successor Lists

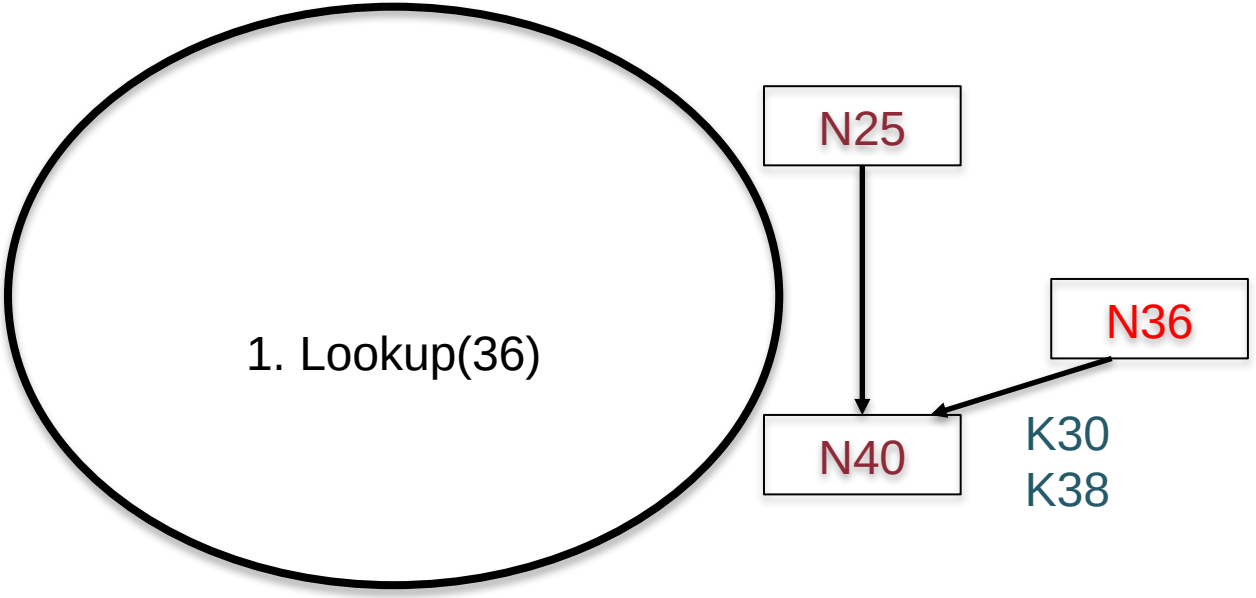
- Each node knows *r* immediate successors
- After failure, will know first live successor
- Correct successors guarantee correct lookups

Guarantee is with some probability

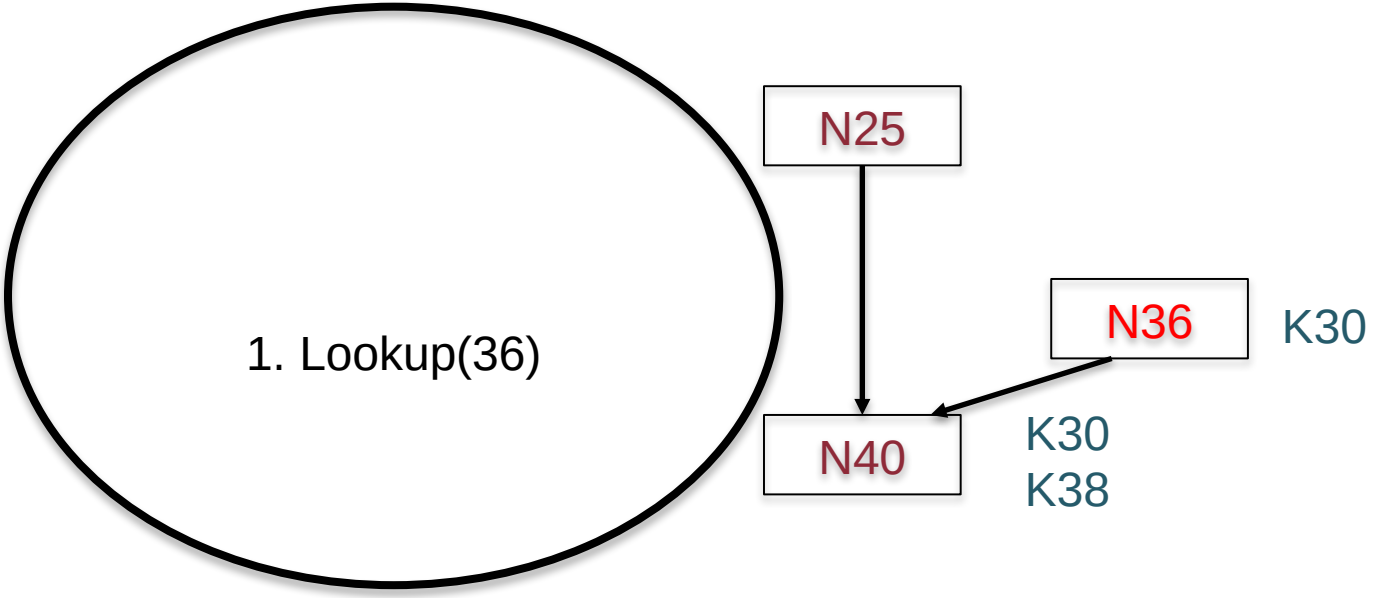
Join (1)



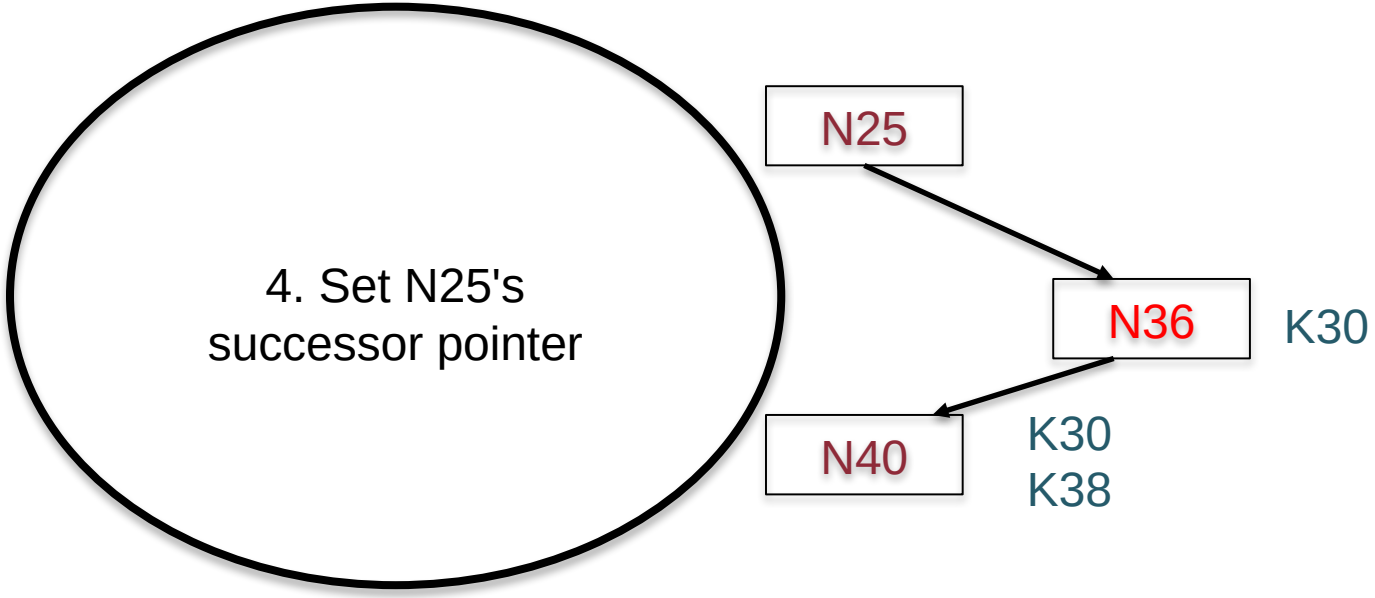
Join (2)



Join (3)



Join (4)



Update finger pointers in the background

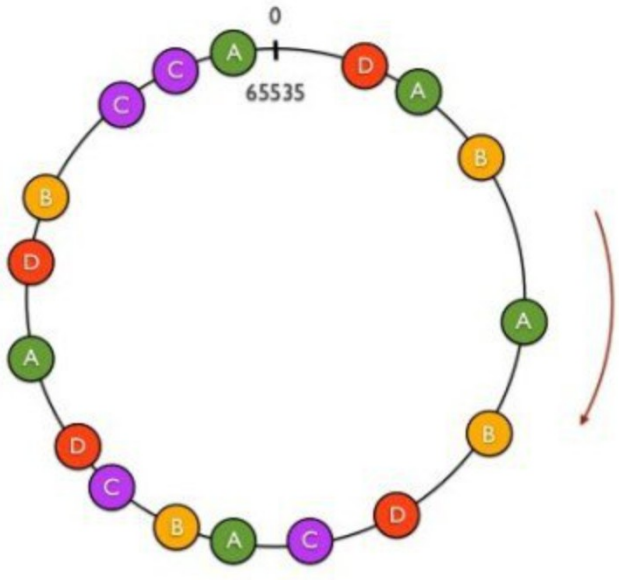
Virtual Nodes for Load Balance

Load balance could be a problem

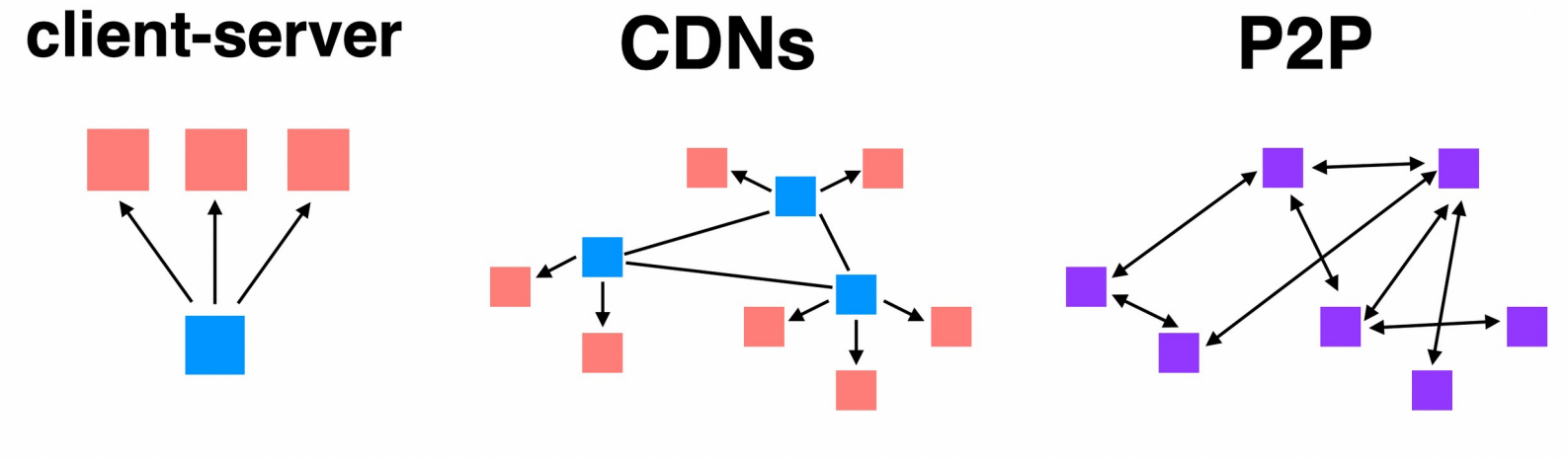
- Some nodes have most value

Solution: virtual nodes

- Virtual nodes are distributed across the ring
- One physical node can have multiple virtual nodes
- More virtual nodes, better load balance



Summary: Different File Sharing Techniques



BITCOIN & BLOCKCHAIN

Overview of BitCoin

BitCoin is a decentralized public-accessible ledger

- Using hashes as a pointers, which form a chain
- Tamper-proof (assuming 51% honest)

What is Crypto-Concurrency?

A ledger recording all the money

Why double-spending is possible?

- Y creates two transactions for same coin: Y->Z, Y->Q
  - both with hash(T7)
- Y shows different transactions to Z and Q
  - both transactions look good, including signatures and hash
  - now both Z and Q will give hamburgers to Y

How to avoid double-spending?

Phases in Blockchain

Proposal phase

- Who to propose the block?

Validation phase

- Who to validate the block?

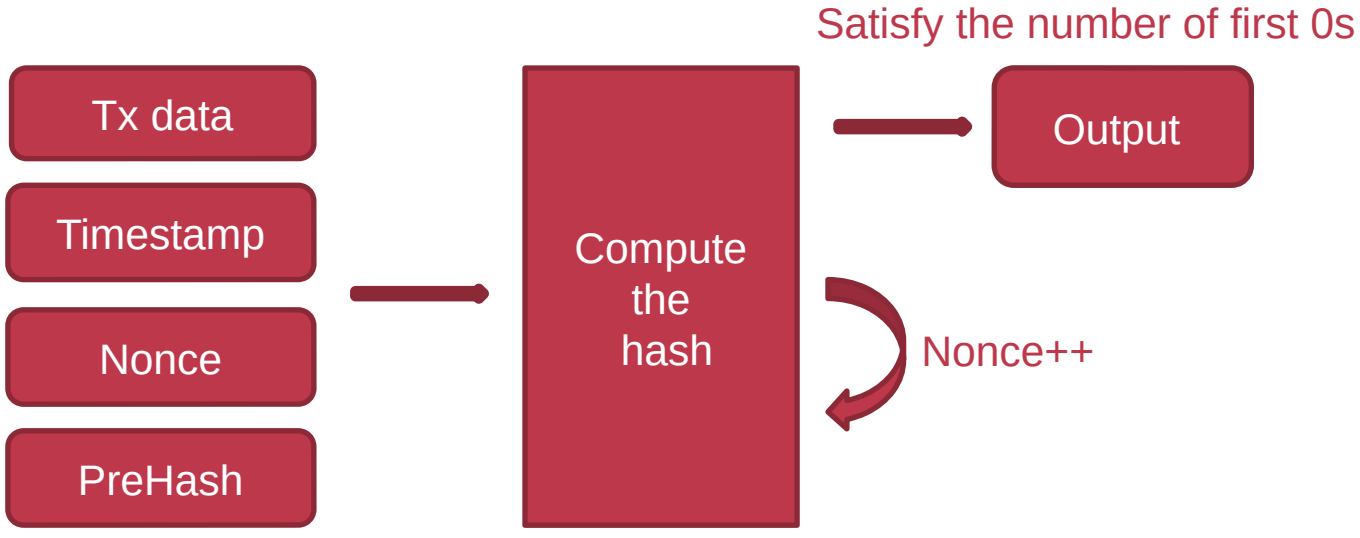
Usually all the participants in the network

- What if an attacker run multiple processes as fake nodes, in order to control 51% nodes?

Existing Solutions

Proof of Work (PoW)

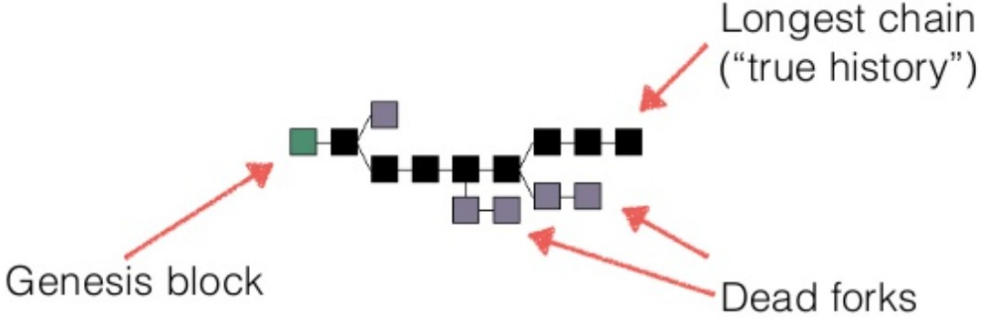
- Brute force to solve a random number (nonce)
- Inefficient, waste of resources



Trust the Majority

If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains

To modify a past block, an attacker would have to redo the proof-of-work of the block and all blocks after it and then catch up with and surpass the work of the honest nodes



Network Steps

1) New transactions are broadcast to all nodes.

2) Each node collects new transactions into a block.

3) Each node works on finding a difficult proof-of-work for its block.

4) When a node finds a proof-of-work, it broadcasts the block to all nodes.

5) Nodes accept the block only if all transactions in it are valid and not already spent.

6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Incentive

By convention, the first transaction in a block is a special transaction that starts a new coin owned by the creator of the block

The incentive can also be funded with transaction fees

The incentive may help encourage nodes to stay honest

- If a greedy attacker is able to assemble more CPU power than all the honest nodes, he ought to find it more profitable to play by the rules than to undermine the system and the validity of his own wealth

Questions

Can some one other than the owner spend a coin?

What if I lose my private key?

Is it possible to have inflation?

Is mining really decentralized?

Is it necessary to use PoW? It's a waste of power!

How can I get bitcoin?

Smart Contract

The nodes not only store transactions, but also code

... and also execute the code to gen new transaction!

The code is known as "smart contract"

- Execution can be triggered by events/invocations

Larger Storage

Store general data, instead of only transaction

Then a node needs to prove it does store the data

- Proof of retrievability, in challenge-response form

Examples: IPFS, STORJ

Permissioned Chain

Bitcoin is a permission-less chain

- Protect user's privacy by anonymity
- Also hard to track

Permissioned chain is not anonymous

- E.g., several companies build a chain used by themselves only
- It is just a multi-manager distributed database, which has nothing new

It's all about Trust

Why blockchain?

- We need a consensus for a ledger
- There is no third party to be trusted

If there is someone trusted by all, then blockchain is not needed

- Blockchain is slow and hard to use
- It is designed to be that way
  - Considering the storage and bandwidth of the ledger

Questions

Why BitCoin is so valuable?

- A bit-string itself does not worth anything
- It is the **consensus** that matters
  - A decentralized global consensus is hard to achieve
  - Before BitCoin, it is even thought as impossible
  - BitCoin solves this problem by incentive. Human is in the loop!

How to ban BitCoin?