CS 138: Distributed Computer Systems

Staff

- Faculty
  - Tom Doeppner
  - Rodrigo Fonseca
- Head TA
  - Jordan Hendricks
- Master's TAs
  - Junyang Chen
  - Hongkai Sun
  - Vivek Narayanan
- UTA
  - Jake Small

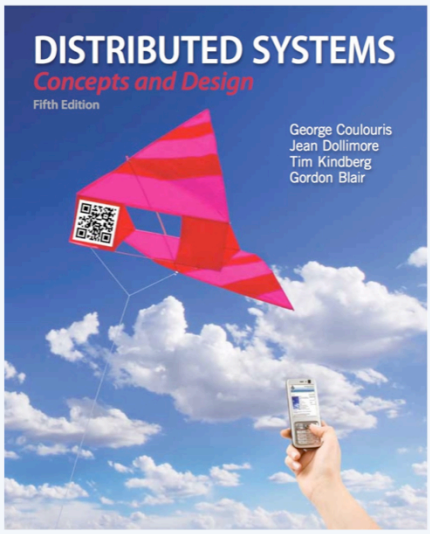
Workload

- Four programs (45%)
  - Chord (5%)
  - Tapestry (10%)
  - Raft (10%)
  - PuddleStore (20%)
- Four written homeworks (15%)
- One in-class midterm exam (15%)
- Final exam (25%)
- See http://www.cs.brown.edu/courses/csci1380/doc/syllabus.pdf

Skills Needed

- Ability to write and debug largish programs with threads
  - CS 32 or 33
- Ability to prove a theorem
  - there won't be many
  - CS 22 is helpful
- Willingness to learn a new programming language
  - Go

Textbook



Recommended, not required.

Facebook Database Replication

- Circa 2007, Facebook decided to add a second datacenter to its operations

https://www.facebook.com/notes/facebook-engineering/scaling-out/23844338919

Why?

- Major reason: latency
  - can't go faster than the speed of light yet
- Other reasons
  - scale: need to handle rapidly increasing loads
  - resiliency: what if an earthquake hits CA?
  - power: sometimes availability of power limits the size of a datacenter!

Caching objects

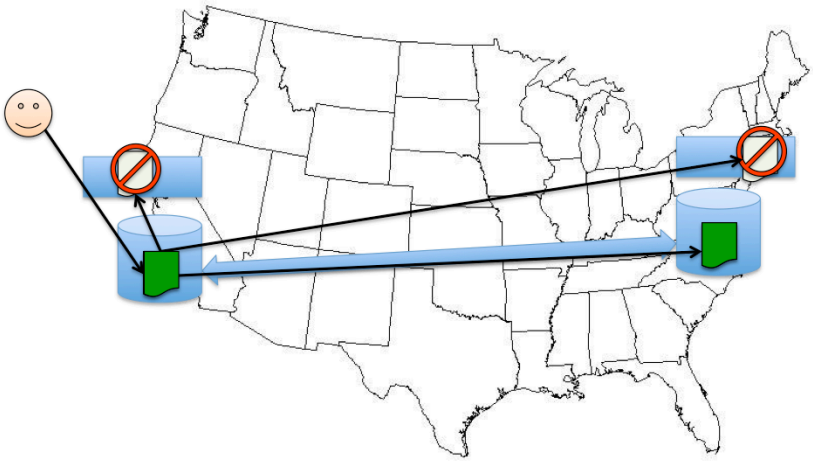
- Facebook handles reads via memcached

Caching objects

- Cache invalidated on a new write

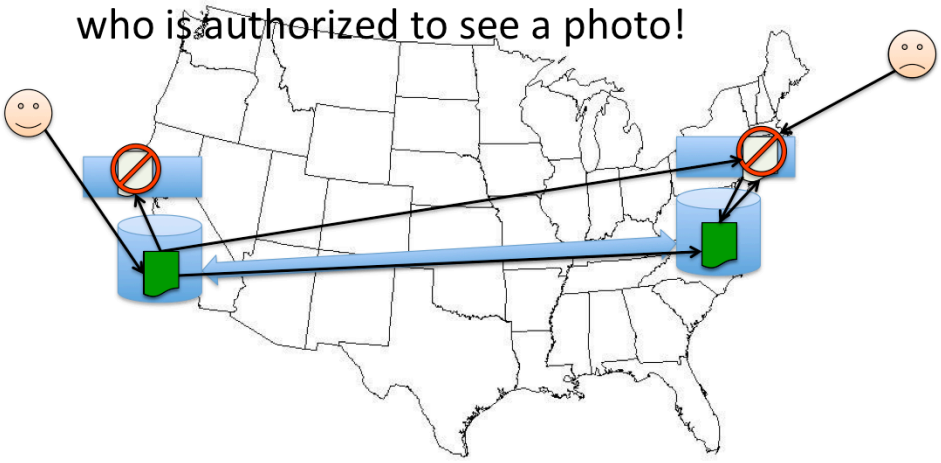
Adding a new Datacenter

- Initial design had a bug



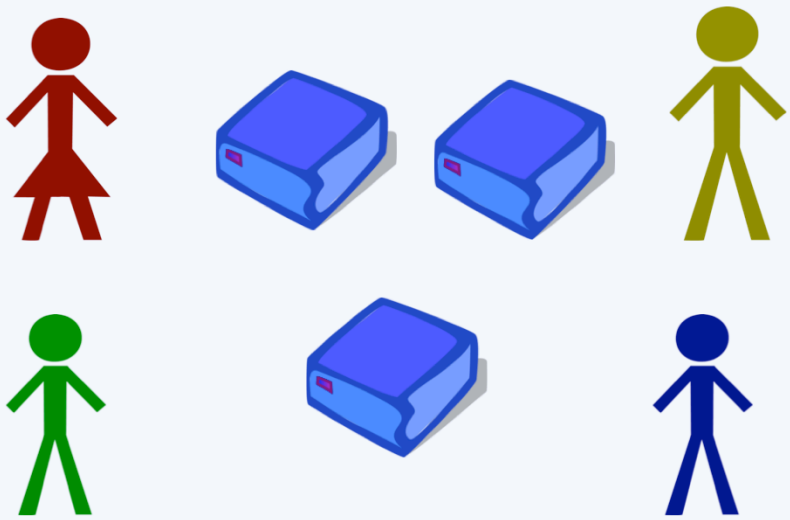
Adding a new Datacenter

- Stale data could be your relationship status, or who is authorized to see a photo!

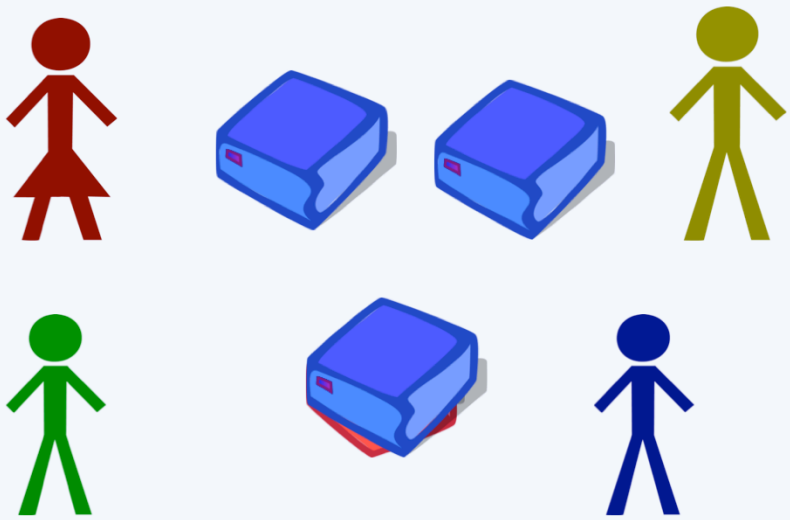


Grades Database

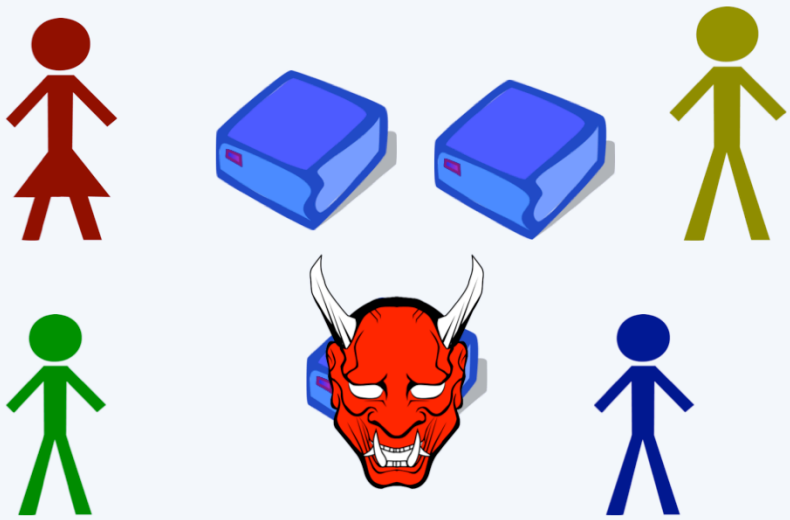
Distributed Grades Database



Failure



Byzantine Failure



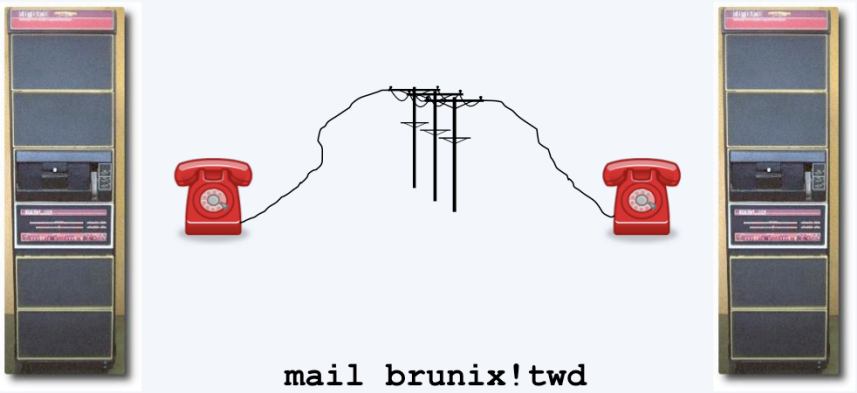
Application Examples

- Email
- DNS
- Content Distribution Networks

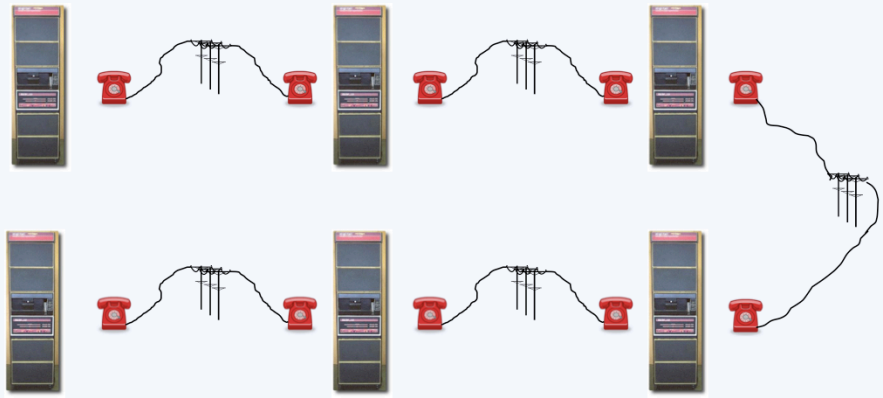
Email: Ancient History

```
mail twd
```

Enter UUCP: Distributed Email



But …



```
mail brunix!rayssd!necntc!husc6!seismo!rick
```

On My 1989 Business Card …

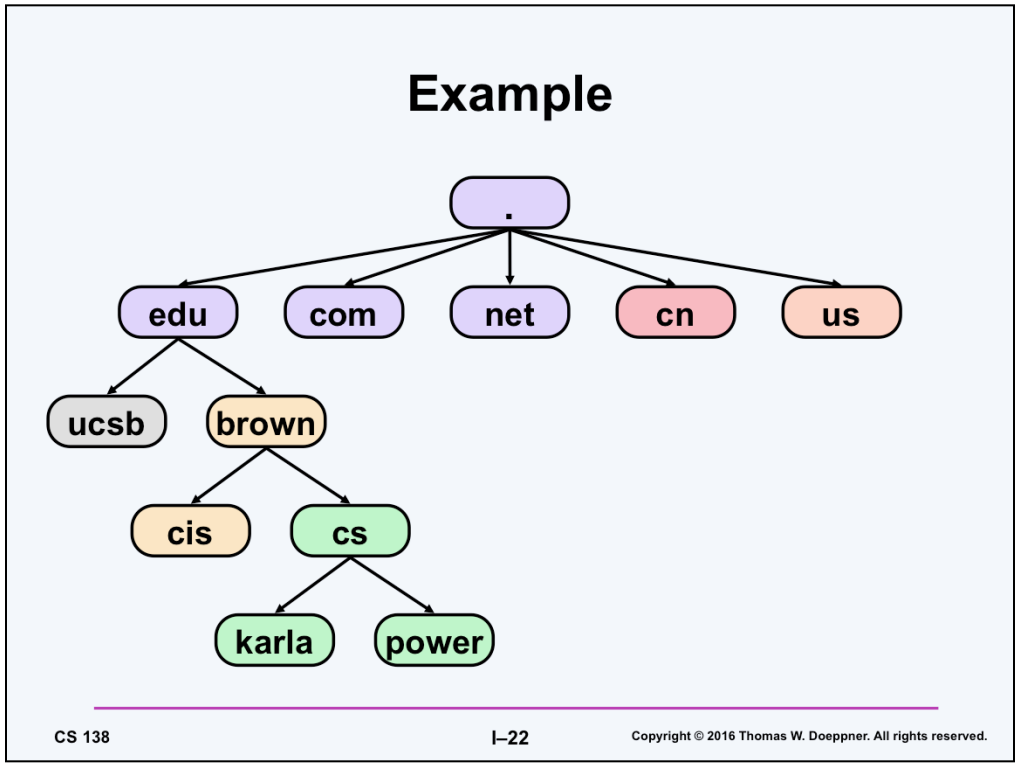
```
{decvax,ihnp4}!brunix!twd
twd@cs.brown.edu
twd@browncs
```


They were, in order, my uucp address, my internet address, and my bitnet address.

Domain Name System

- The naming system for the Internet
  - highly successful
  - widely distributed administration
  - good for long-lived, static information
  - not extensible
  - simple API

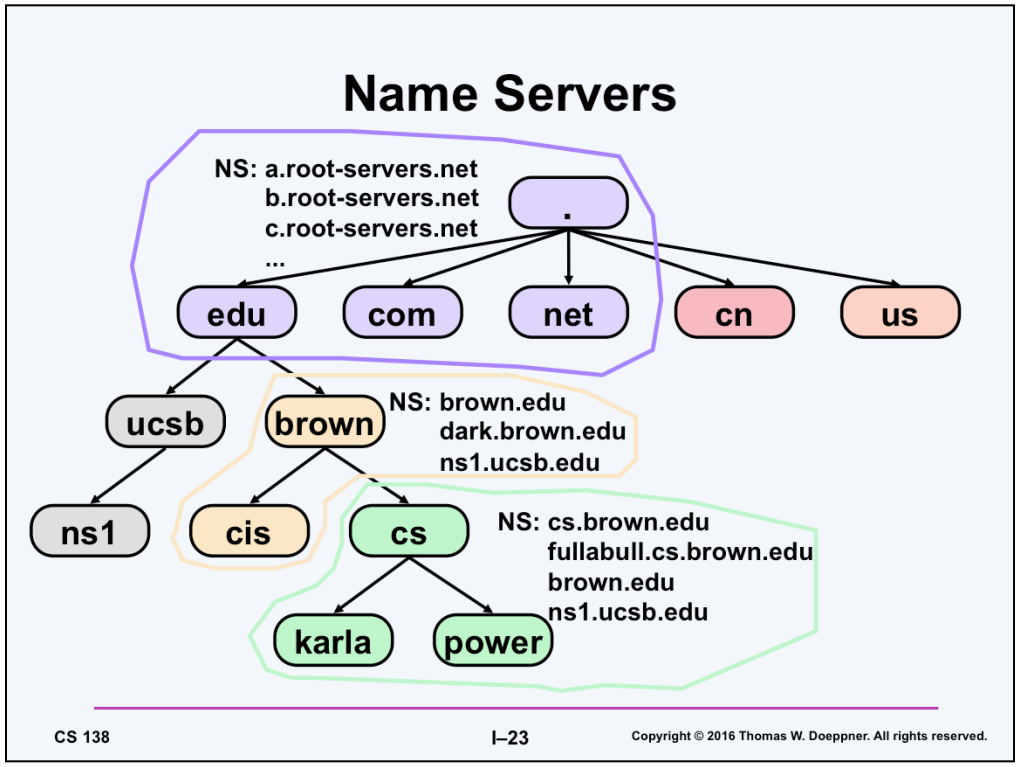
DNS is, by far, the most widely used and widely dispersed directory system in the world. To be this successful it must deal with most of the concerns mentioned on the previous slide. In particular, it must be highly available, meaning that the service must always appear to be "up," even if a number of components are down. Its naming facilities must allow for the addition of an unlimited number of new names. The Internet is much too large for there to be a single agency administrating DNS—its administration must be partitioned so that each company, university, department, etc. can administer its own portion of the DNS name space. Finally, it must be reasonably secure (though more work is required here).



The slide shows a very small portion of the DNS name space. Trees and subtrees are known as domains and subdomains. Thus the tree headed by the node labeled "." is known as the root domain. Beneath it are a number of subdomains, known as first-level domains. These are divided into "three-letter" domains, representing types of organizations, and "two-letter" domains, representing countries. Contrary to popular opinion, the three-letter domains are not restricted to US organizations. The management of the "com" domain has recently become extremely controversial.

The administration of the name space is split into "zones of authority," represented by the different colors of nodes (for those looking at this in black and white: the first zone is comprised of the nodes ".", "edu.", "com.", and "net.". Another zone is headed by "cn."; another is headed by "us.". Within the "edu." domain are the zones containing "brown.edu." and "cis.brown.edu." and the one containing "ucsb.edu." Finally, within the "brown.edu." domain is the zone containing "cs.brown.edu.", "karla.cs.brown.edu.", and "power.cs.brown.edu."). Each zone is separately administered. The administrators of a zone are responsible for making certain that the parent zone knows about them, as discussed in the next slide.

One minor syntactic issue is whether to include the "." representing the root node in DNS names. Strictly speaking, one should, but in practice, no one does. Thus one writes "cs.brown.edu" rather than "cs.brown.edu.".



Each zone must have one or more name servers providing the database containing the contents of the zone. The name servers for three of the zones are shown on the slide. To follow a path to *karla.cs.brown.edu* starting from the root, one would first contact a name server in the zone at the top of the root domain. This would refer to a name server at the top of the *brown.edu* domain, which would in turn refer to a name server at the top of the *cs.brown.edu* domain. This last name server would, presumably, know about *karla.cs.brown.edu*.

There is a requirement that there be at least two name servers for each domain, each an identical copy of the others. Preferably, at least one of the name servers should be geographically distant from the others, and certainly on a different power system. This is to increase the likelihood that at least one name server for a domain is up.

Replicating Name Servers

- One name server is the "primary"
- Others are "secondaries"
- Secondaries poll the primary for updates
  - information is tagged with a maximum lifetime (typically one week!)

One of the name servers is designated as the primary name server; the others are secondaries. Administrators make changes to the copy of the information in the primary; the secondaries periodically poll the primary to acquire such modifications. No attempt is made to keep the secondaries perfectly in sync with the primary. In fact, secondaries may continue to function even if they've been unable to contact the primary for up to a specified period of time, typically a week. For many databases, such a long period of no contact would be disastrous. But the sort of information kept in the DNS name space usually does not change very often; it is much better to obtain somewhat-out-of-date information than to obtain no information.

Lookups

- **Order of search**
  1. **contact name server in local domain**
  2. **contact root name server and proceed downwards**
- **Caching**
  - **results of recent queries are cached by name servers**
  - **local machine also caches recent lookups**
- **Recursive vs. iterative**
  - **recursive queries are handled completely by recipient**
  - **recipient sends referrals to sender of iterative queries**

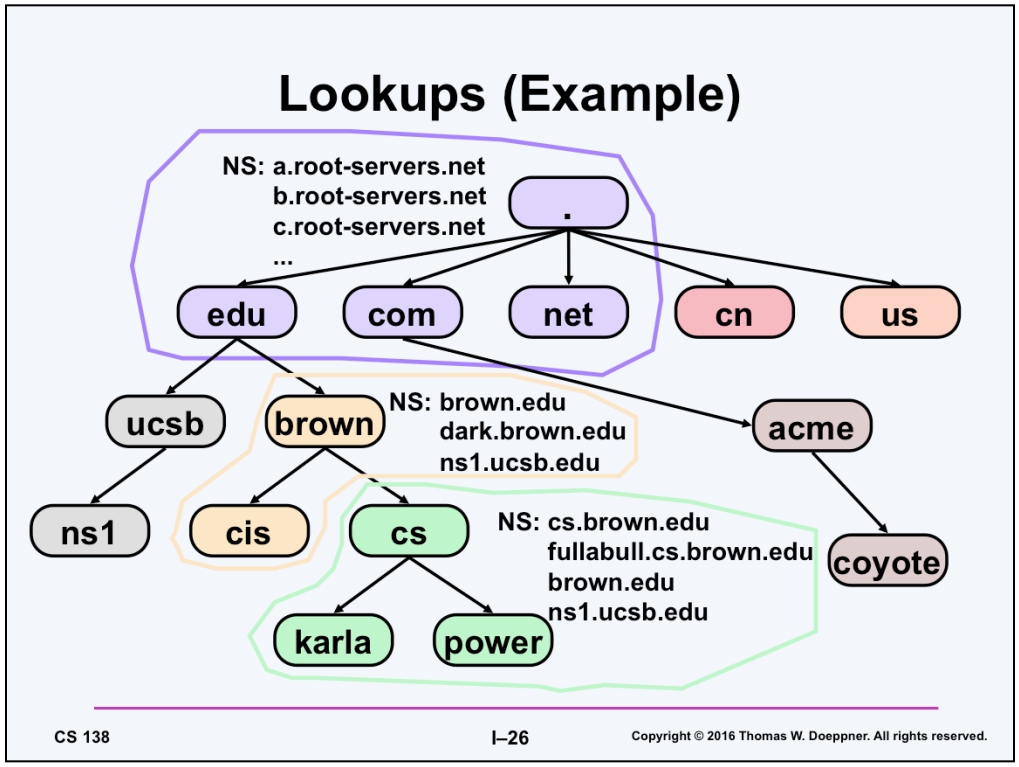
Unlike how things are done in most file systems, lookups in the DNS name space do not generally start with the root — if they did, the root name servers would be dramatically overloaded. Instead, lookups start with a name server in the local zone. Only if this name server does not have the information is a request sent to a root name server.

To further reduce the load on name servers, information obtained from them is cached, both on name servers and on client machines. In many environments, additional caching-only servers are employed which cache DNS information and make it available to a number of clients.

Caching is especially feasible with DNS since, as mentioned on the previous page, information tends to change infrequently. When a server provides information, it tags it with a TTL (time to live) indicating how long the information may reside in the cache. Such TTLs must be specified by the administrators setting up a server and are typically at least a day in length.

If a server responds with information from its database, the answer is said to be authoritative. However, it responds with information from its cache, the answer is unauthoritative (and is tagged as such).

Another issue is who does the work: if a client makes a recursive query, the lookup is handled completely by the first server contacted. If it doesn't have the requested information, it takes responsibility for contacting a root server and following the request down the tree. Another option is the iterative query, in which a contacted server responds either with the answer (authoritative or unauthoritative) or with a referral indicating which server to go to next. Whether to use recursive or iterative queries is negotiated between client and server. Root servers typically do not handle recursive queries (they're too busy). Lower-level servers often do.



Suppose that someone at coyote.acme.com does a lookup of the address of *karla.cs.brown.edu*. They will first check their local cache. If this doesn't have it, they contact the name server for their zone via a recursive query. This name server isn't authoritative for *cs.brown.edu* and also doesn't have the answer in its cache, so it contacts a root server, perhaps b.root-servers.net. This server returns a referral to the name servers for *brown.edu*. Of these servers, *acme.com*'s server chooses ns1.ucsb.edu. This server returns a referral to cs.brown.edu's server — *cs.brown.edu*. This server has the correct answer and returns *karla.cs.brown.edu*'s address.

Resource Records

- **Form logical contents of each node**
  - **a number of standard types, e.g.:**
    - **A: *address* of a machine**
    - **MX: *mail exchanger***
    - **SOA: *start of authority***
    - **PTR: *pointer***
    - **NS: *name server***
    - **CNAME: *canonical name***
  - **not easily extensible (everyone must agree to changes)**

Each node consists of a collection of information known as resource records. Each such record contains in a particular type of information—some of the more important types are shown on the slide. Though in principle resource records are extensible, in practice they are not, since adding a new type requires notification (and agreement) of the entire Internet.

Some of the standard record types are listed below:

A: *address* of a machine (router machines have a number of addresses)

MX: *mail exchanger*—address of machine that handles email

SOA: *start of authority*—defines beginning of zone of authority: indicates administrative boundary

PTR: *pointer*—points elsewhere in the name space

NS: *name server*—defines a name server for a domain

CNAME: *canonical name*—maps an alias or nickname to the real name

MX Example

- **Mail is sent to "twd@karla.cs.brown.edu"**
  - **mail-sending program queries DNS for an MX record in karla.cs.brown.edu**
  - **the following info is returned:**

```
karla.cs.brown.edu   preference = 10,
        mail exchanger = cs.brown.edu
cs.brown.edu  nameserver = cs.brown.edu
cs.brown.edu  internet address = 128.148.128.2
```

    - **mail is sent to cs.brown.edu**
    - **a name server for that domain can be found at cs.brown.edu**
    - **its internet address is 128.148.128.2**

In this example, someone is sending email to *twd@karla.cs.brown.edu*. There is an MX record for the node *karla.cs.brown.edu* that indicates that a "mail exchanger" can be found at *cs.brown.edu* (the preference value is a priority that's used if multiple mail exchangers are listed — preference is given to the one with the lowest preference value; if that one is not up, then the one with the next lowest preference value is used, etc.). As a convenience to the caller, along with the MX record is returned the identity of the name servers for the domain containing the mail exchanger and the address records for those name servers.

Note that including a machine name in one's email address is a poor idea. It works in this case, since the node *karla.cs.brown.edu* exists. However, it's likely that the lifetime of *karla.cs.brown.edu* is less than the lifetimes of both twd and *cs.brown.edu*: if karla ceases to exist, email to *twd@karla.cs.brown.edu* will fail.

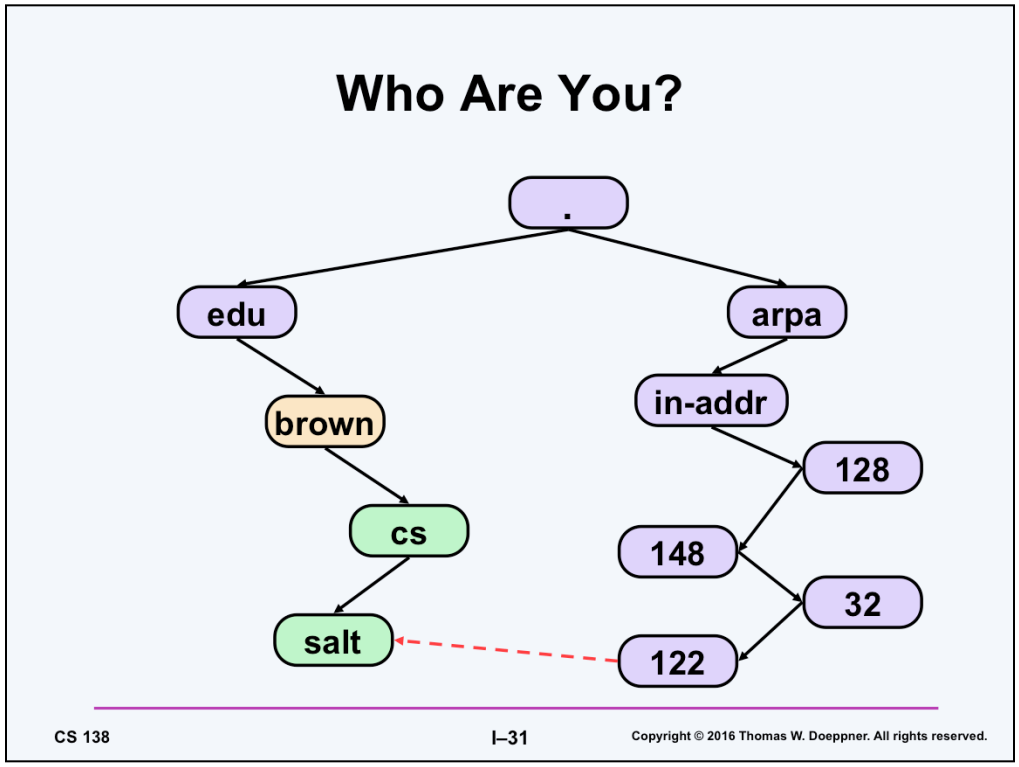
Administration

```
$ORIGIN .
$TTL 86400          ; 1 day
cs.brown.edu                IN SOA   ns.cs.brown.edu. root.cs.brown.edu. (
                                     5870       ; serial
                                     10800      ; refresh (3 hours)
                                     3600       ; retry (1 hour)
                                     604800     ; expire (1 week)
                                     86400      ; minimum (1 day)
                                     )
                            NS       dns.cs.brown.edu.
                            NS       ns1.ucsb.edu.
                            NS       knot.brown.edu.
                            A        128.148.32.110
                            MX       10 mx.cs.brown.edu.
                            AFSDB    1 radio.cs.brown.edu.
$ORIGIN cs.brown.edu.
0a                          CNAME    cslab0a
0b                          CNAME    cslab0b
cslab0a                     A        128.148.31.190
                            MX       10 mx
cslab0b                     A        128.148.33.106
                            MX       10 mx
mx                          A        128.148.32.120
```

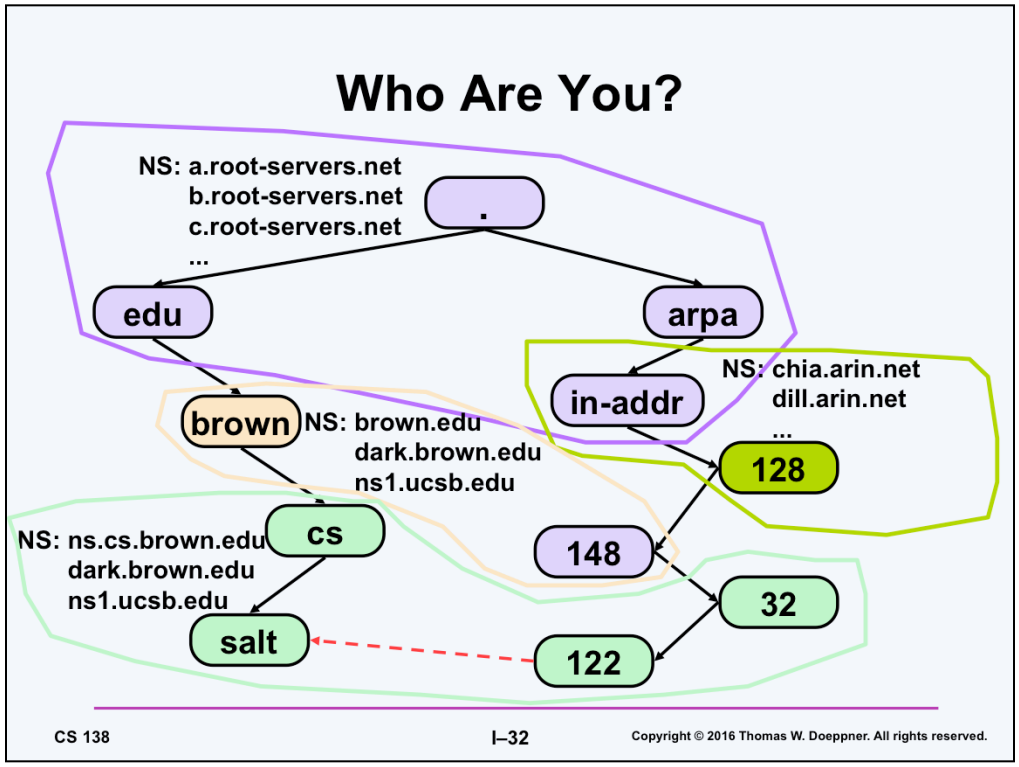

Here's a portion of the zone file describing the database maintained by name servers for the cs.brown.edu domain.

Who Are You?

- I'm 128.148.32.122
- *What's that?*



The *in-addr.arpa* domain provides a means for doing reverse lookups: given the IP address of a machine, this domain maps it into its DNS name. It's needed by servers that want to know who is contacting them (e.g., what is the domain of the caller). The leaf nodes in the *in-addr.arpa* domain contain PTR-type records referring to the actual domain names.



Resolving a name such as 122.32.148.128.in-addr.arpa, just like resolving other DNS names, requires the cooperation of a number of authorities. Names within the in-addr.arpa domain are resolved by the root name servers. To understand the next step in the resolution, we must determine who should be responsible for handling the network containing the address 128.148.32.122. Since it is a class-B address, the network identifier is the contained in the first two bytes: 128.148. Since this network address was assigned to Brown by some Internet authority, that same authority should be able to remember (and divulge) that this address belongs to Brown. Originally this authority was IANA (Internet Assigned Numbers Authority), but since the late 1990s, this authority as been split up into regional organizations. The organization handling the US, Canada, and some Caribbean and North Atlantic Islands is ARIN (American Registry for Internet Numbers). Thus their name servers divulge that 128.148 is owned by Brown and refer queries to Brown's name servers. Brown's name servers, in turn, know that subnet 32 is owned by the CS department and refer queries to CS name servers. These name servers maintain the PTR records mapping host addresses to DNS names.

More Administration

```
$ORIGIN .
$TTL 86400          ; 1 day
32.148.128.IN-ADDR.ARPA  IN SOA   ns.cs.brown.edu. root.cs.brown.edu. (
                         5874     ; serial
                         10800     ; refresh (3 hours)
                         3600     ; retry (1 hour)
                         604800    ; expire (1 week)
                         86400     ; minimum (1 day)
                         )
                    NS       dns.cs.brown.edu.
                    NS       ns1.ucsb.edu.
                    NS       knot.brown.edu.
$ORIGIN 32.148.128.IN-ADDR.ARPA.
1                   PTR      fw-32.cs.brown.edu.
110                 PTR      list.cs.brown.edu.
111                 PTR      ftp.cs.brown.edu.
120                 PTR      mx.cs.brown.edu.
121                 PTR      dns.cs.brown.edu.
122                 PTR      salt.cs.brown.edu.
```


Here is a portion of the zone file for the 32.148.128.in-addr.arpa domain.

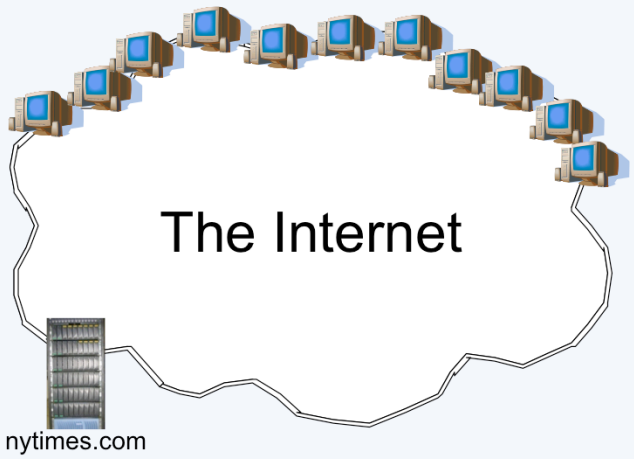
Recap: Issues

- Failure tolerance
- Decentralized management
- Speed vs. consistency

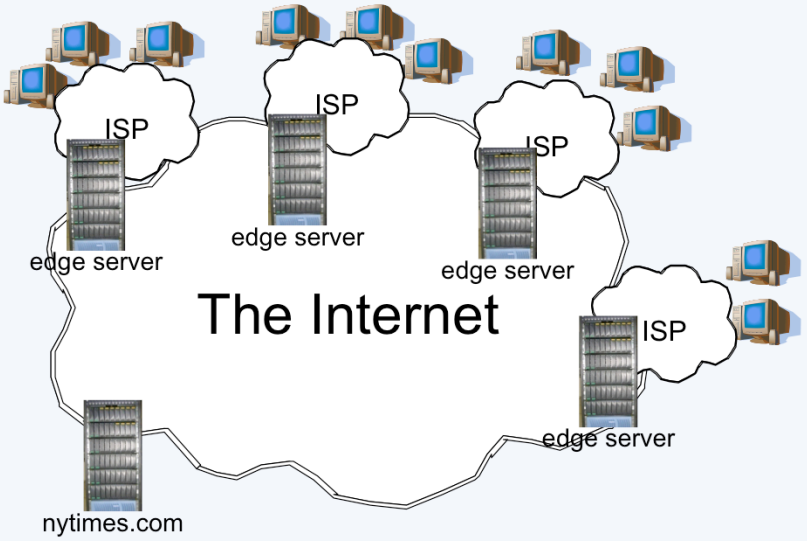
“Akamai's technology – at its core, applied mathematics and algorithms – has transformed the chaos of the Internet into a predictable, scalable, and secure platform for business and entertainment. The Akamai EdgePlatform comprises 73,000 servers deployed in 70 countries that continually monitor the Internet – traffic, trouble spots and overall conditions. We use that information to intelligently optimize routes and replicate content for faster, more reliable delivery. As Akamai can handle up to 15-20% of Web traffic on any given day, our view of the Internet is the most comprehensive and dynamic collected anywhere.”

The quote is from http://www.akamai.com/html/technology/index.html, viewed in January 2011. It’s since been updated (among other things, they’ve added a country).

Content Delivery



Content Delivery Network



What Sort of Content?

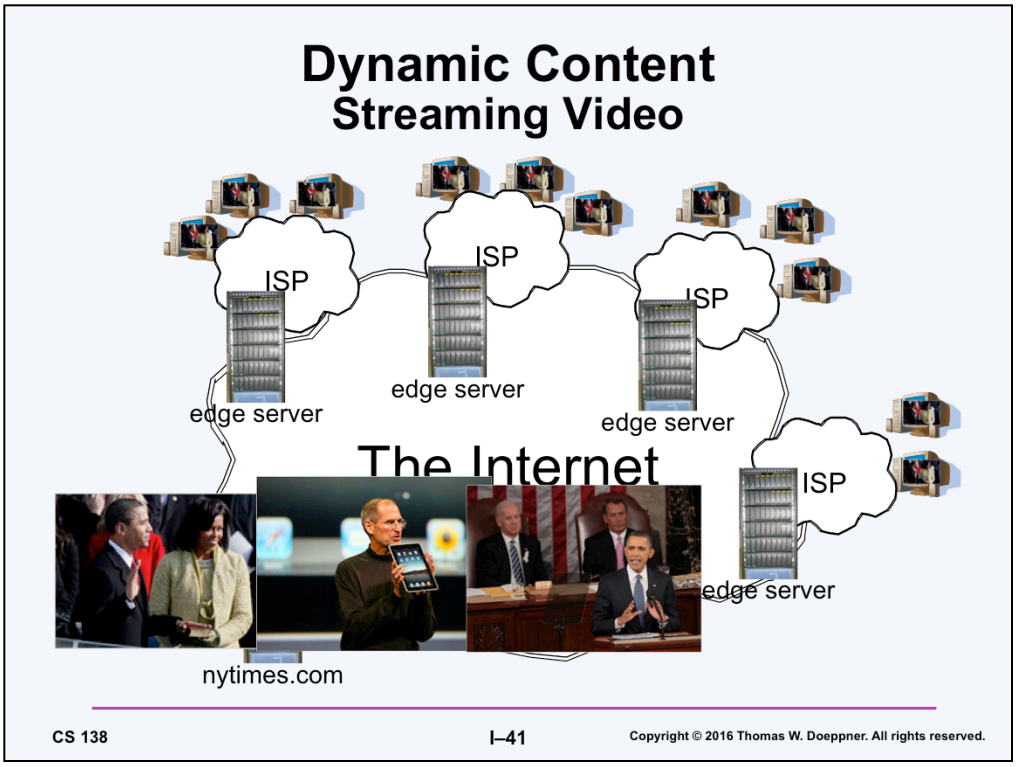
- Content aggregation
  - portals, news aggregators, etc.
- Static databases
  - store locators, product catalogs, product configurators
- Data collection
  - college applications, credit card applications, polling sites
- Two-way data exchange
  - ad serving
- All of the above

See http://www.akamai.com/dl/technical_publications/EdgeComputingExtendingEnterpriseApplicationstotheEdgeoftheInternet.pdf.

How Is It Done?

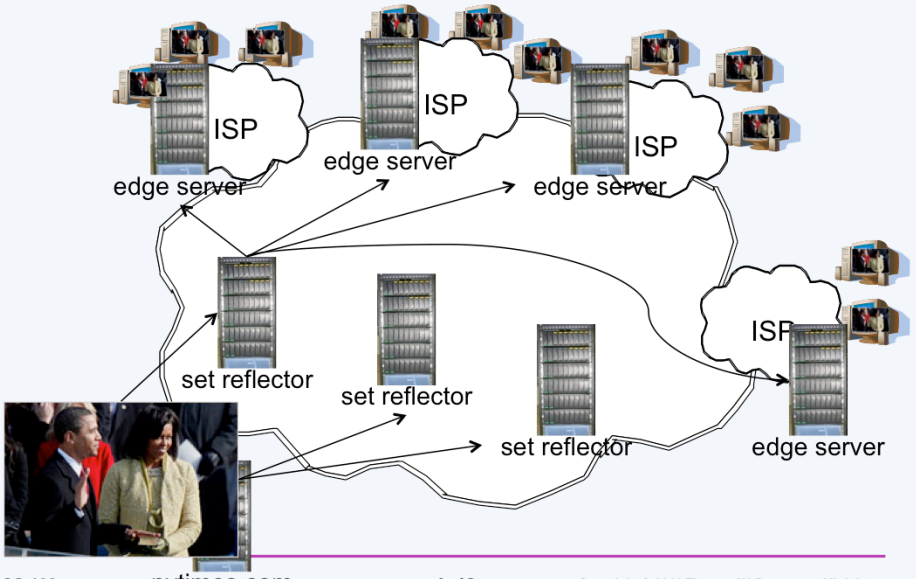
- Smoke and mirrors (courtesy of DNS)
- Example (much simplified)
  - resolve images.nytimes.com
  - DNS returns a "CNAME":
    - images.nytimes.com.g.akamai.net
  - this is resolved right to left
  - akamai.net determines which akamai server is closest to caller
  - resolves "g.akamai.net" to IP address of that server

Two papers describing this may be found at http://www.akamai.com/dl/technical_publications/GloballyDistributedContentDelivery.pdf and http://www.akamai.com/dl/technical_publications/EdgeComputingExtendingEnterpriseApplicationstotheEdgeoftheInternet.pdf.

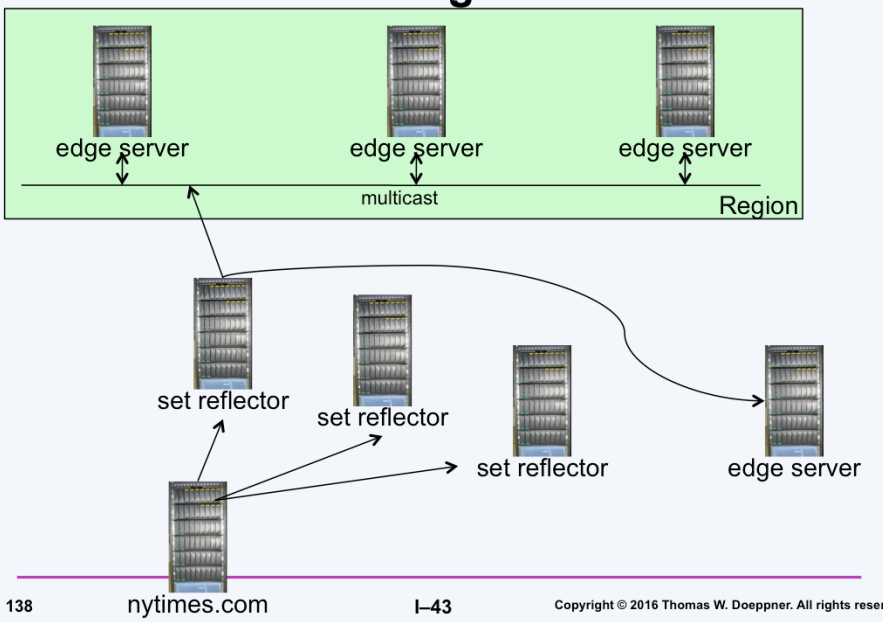


How streaming video is handled at Akamai is described in http://www.akamai.com/dl/technical_publications/ATransportLayerforLiveStreaminginaContentDeliveryNetwork.pdf.

Dynamic Content
Streaming Video



Dynamic Content
Streaming Video



Introduction to Go

Where is Go used?

- Google, of course!
- Docker (Container management)
- CloudFlare (Content Delivery Network)
- Digital Ocean (VM hosting)
- Dropbox (Cloud storage/file sharing)
- ... and many more!

Why use Go?

- Easy concurrency w/ goroutines (green threads)
- Garbage collection and memory safety
- Libraries provide easy RPC
- Channels for communication between goroutines

Example: Simple Program

```
package main

import (
	"fmt"
	"os"
)

func main() {
	for count := 1; count < 100; count++ {
		if count%2 == 0 {
			fmt.Printf("Found even number: %v\n", count)
		} else {
			fmt.Fprintf(os.Stderr, "Not an even number: %v\n", count)
		}
	}
}
```

- No parentheses
- “for { }” will loop forever
- “for condition { }” avoids initialization/afterthought, similar to a while loop

Example: Concurrency

```
package main

import (
          "fmt"
          "time"
)

func main() {
          go func() {
                    time.Sleep(time.Second * 5)
                    fmt.Printf("1")
          }()

          go func() {
                    fmt.Printf("2")
          }()

          time.Sleep(time.Second * 10)
}
```

- “go” keyword executes following function call in a separate goroutine
- Goroutines don’t necessarily run in another OS thread
- Refer to GOMAXPROCS in “runtime” package

Example: Channels

```
package main

import (
        "fmt"
        "time"
)

func message(send, recv chan string, str string) {
        for {
                send <- str
                s := <-recv
                fmt.Println(s)
        }
}

func main() {
        pingChan := make(chan string, 1)
        pongChan := make(chan string, 1)
        go message(pongChan, pingChan, "ping")
        go message(pingChan, pongChan, "pong")
        time.Sleep(time.Second)
}
```

- The channels are buffered so the goroutines don't wait on each other

Editing Go

- Syntax highlighting and formatting:
  - Vim
  - Emacs
  - Sublime
  - Eclipse
- Gotags for editors with ctags support
- Links available at:
  http://cs.brown.edu/courses/cs138/s15/syllabus.html

Tips

- `go fmt' - format source code
- `godoc' - view Go docs in localhost browser
- `runtime/pprof' - profiling package
- `go test' and `go tool cover' - test coverage
- `goimports' - add/remove imports as needed

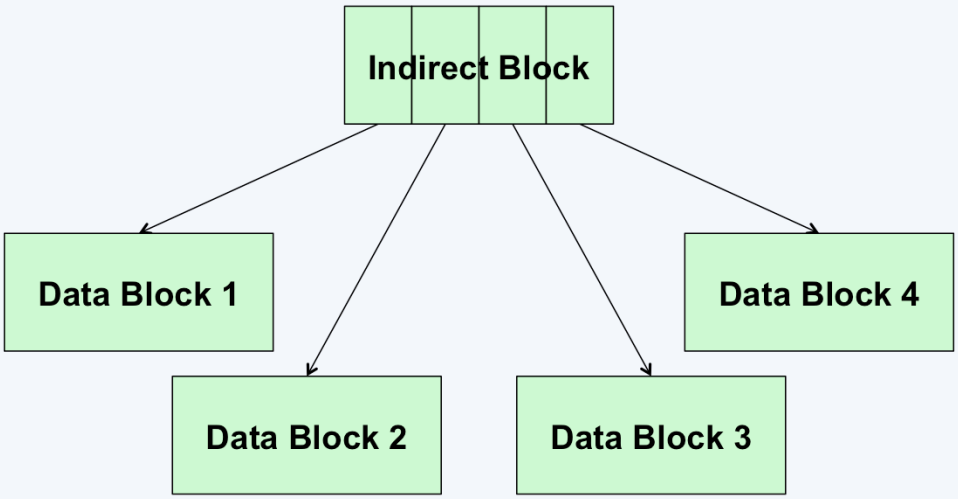
Learning Go

- Project 0: Whatsup?
- Effective Go
- golang.org/doc
- tour.golang.org

PuddleStore

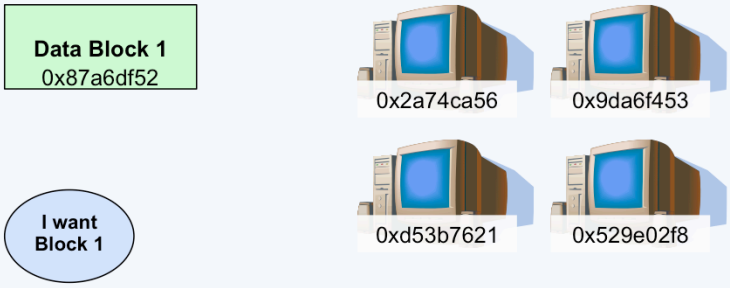
- A very distributed file system
  - thousands of computers
    - all over the world
      - (or at least throughout the SunLab)
    - no common administration
  - each holds pieces of a few files
    - pieces replicated on many computers
- Based on OceanStore
  - and its Pond prototype

A File



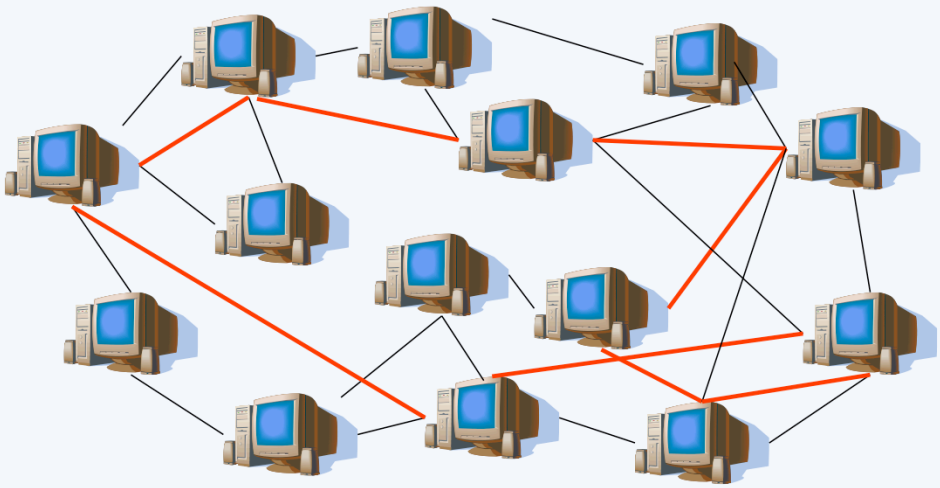
A Distributed File

Making It Work (sort of …)



- Assign each block a unique n-bit ID
  - crypto hash of its contents
- Assign each computer a unique n-bit ID
- Store block at computer that has closest ID
- Route requests for that block to that computer

Overlay Networks



Chord

- Distributed hash tables meet overlay networks
  - hash both keys and node IP addresses into identifiers
    - m-bit identifiers, where m is large enough so that probability of collision is negligible
  - lookups resolved in O(log n) messages
  - adding or deleting a node requires $O(\log^2 n)$ messages
- You implement it in the first programming assignment

We explain Chord in detail next week. The time bounds given are “with high probability.”

Making It (really) Work (with high probability)

- Assign each block a unique n-bit ID
  - crypto hash of its contents
- Assign each computer a unique n-bit ID
- Store multiple copies of blocks each at a number of computers
- Store block addresses at computer that has closest ID
  - addresses are cached at other nodes
- Route requests for that block to that computer
  - request is redirected to nearest computer that has copy of block

Tapestry

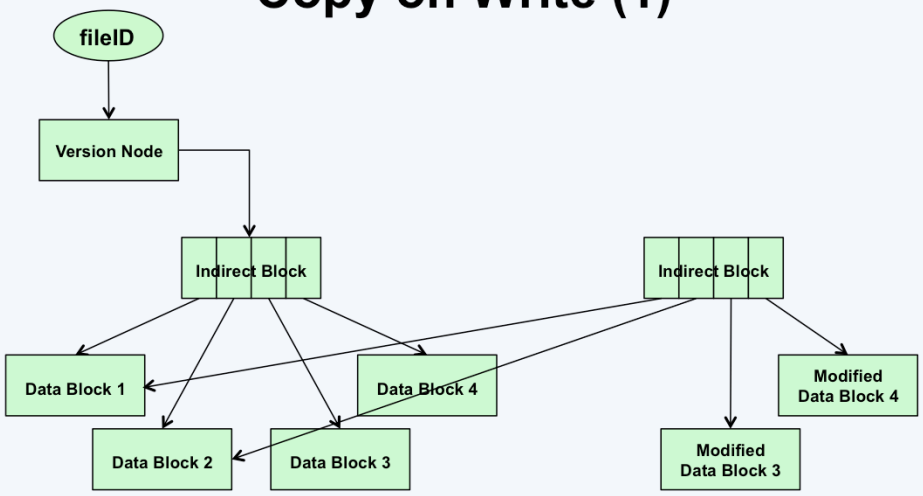
- Distributed object location and routing (DOLR)
  - you implement it in the second programming assignment

We explain Tapestry in detail next week as well. It is the overlay network used by PuddleStore.

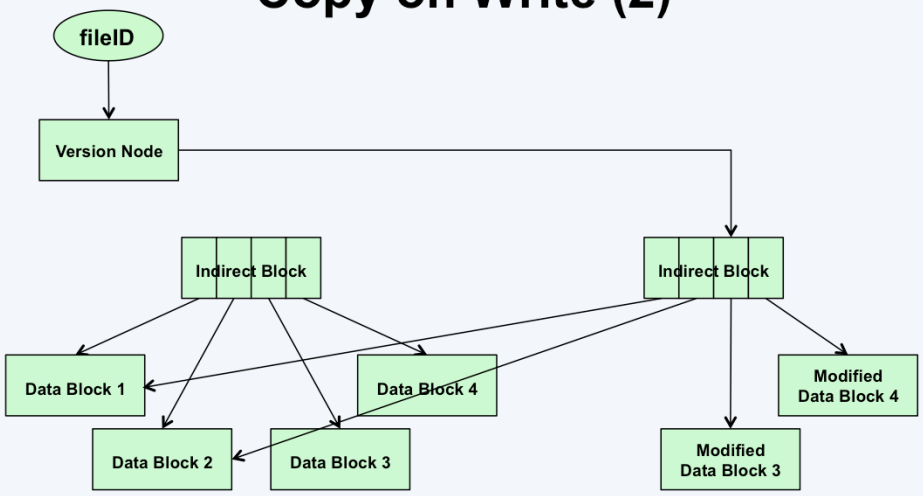
More PuddleStore Issues

- How are files named?
  - fileID = CryptoHash(file name)
- How are files updated?
  - carefully …

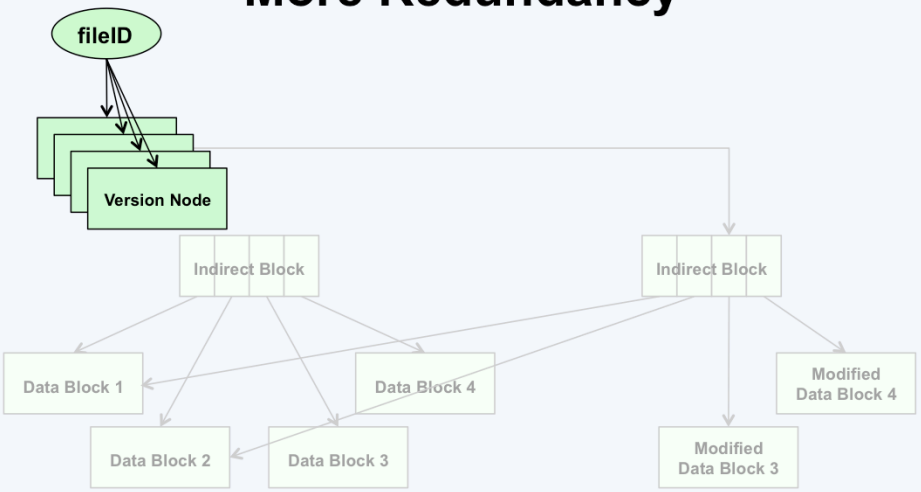
Copy on Write (1)



Copy on Write (2)



More Redundancy



Raft

- Multiple clients update file concurrently
- Each communicates with different servers
  - servers propagate changes to all copies
- How do we ensure that all copies are updated in the same order?
  - order matters ...
- Raft
  - third programming assignment

Final PuddleStore

- You put all this together
  - we give you the B design
    - if you implement it completely: you get a B
  - if you improve it (reasonably well): you get an A (and it may count as a capstone)
    - you're encouraged to discuss your design with classmates