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

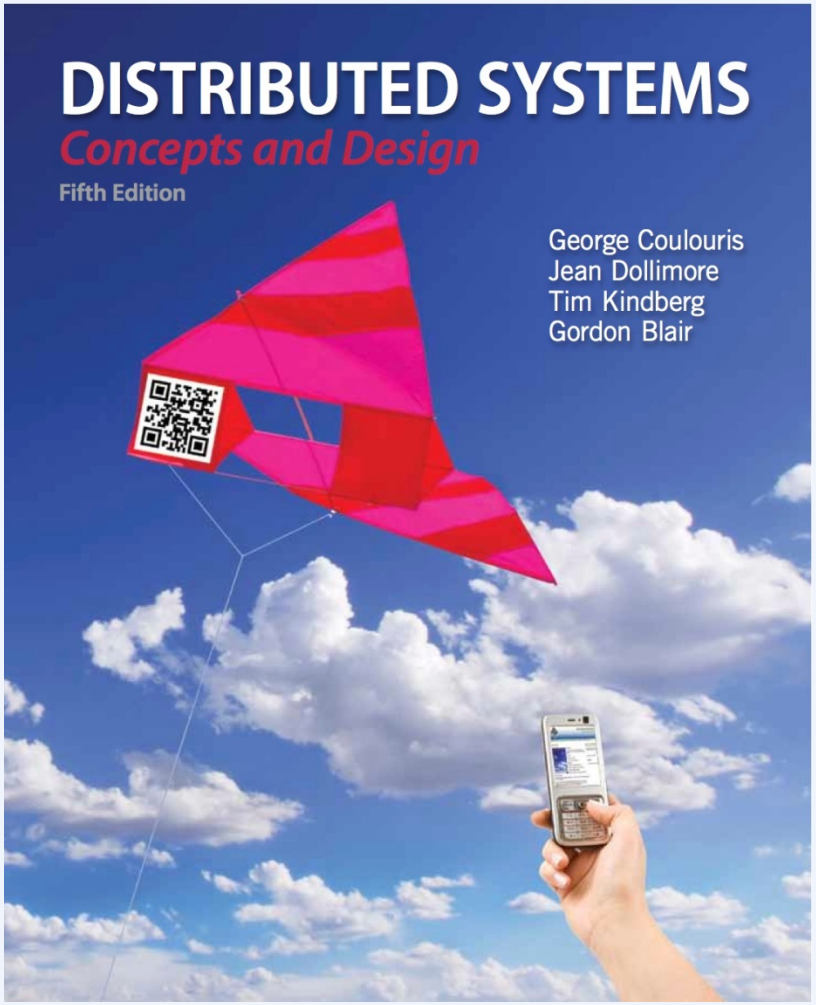
Workload

- 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

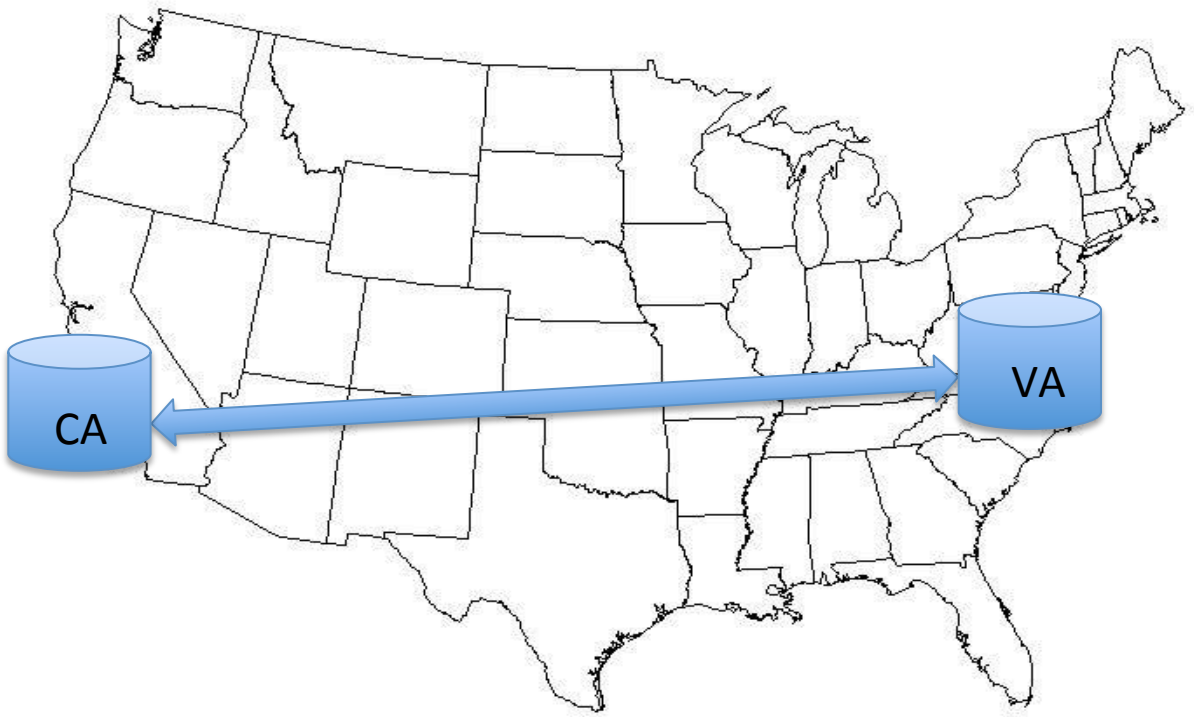
- Ability to write and debug largish programs with threads
  - CS 32 or 33
- Ability to prove a theorem
  - there won't be many
  - CS 22 is helpful
- Willingness to learn a new programming language
  - Go

Textbook



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

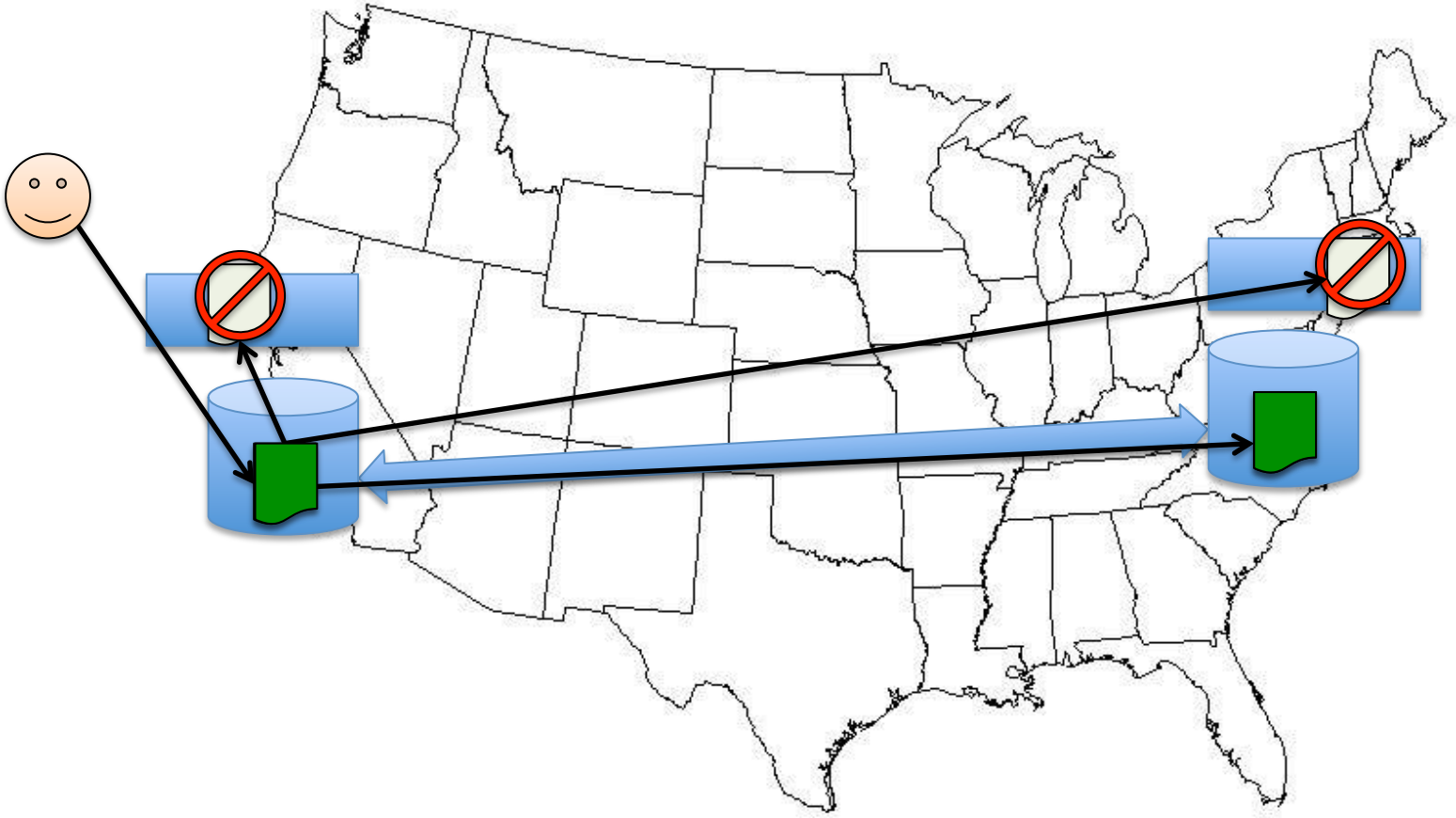
- Facebook handles reads via memcached

Caching objects

- Cache invalidated on a new write

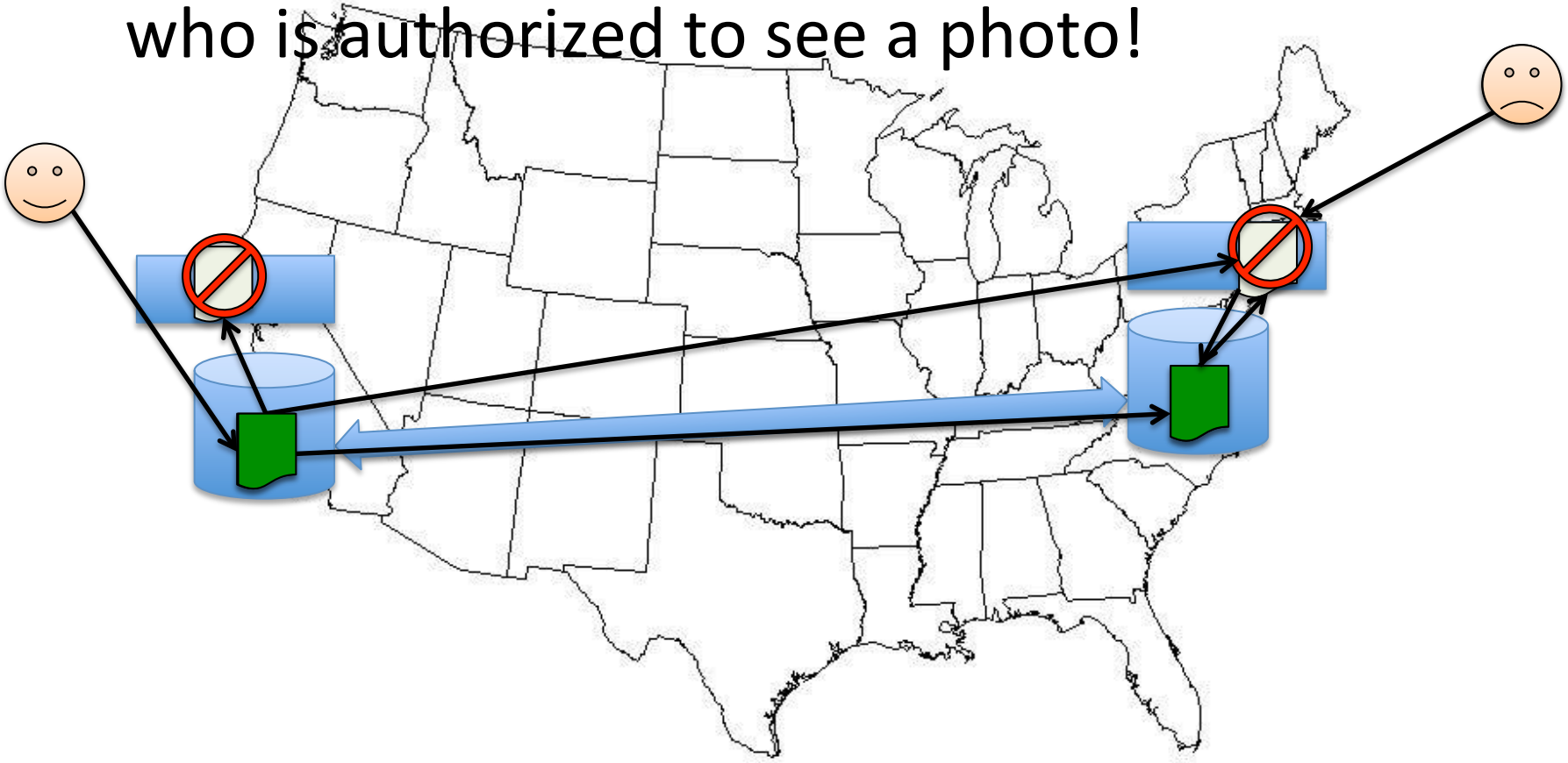
Adding a new Datacenter

- Initial design had a bug



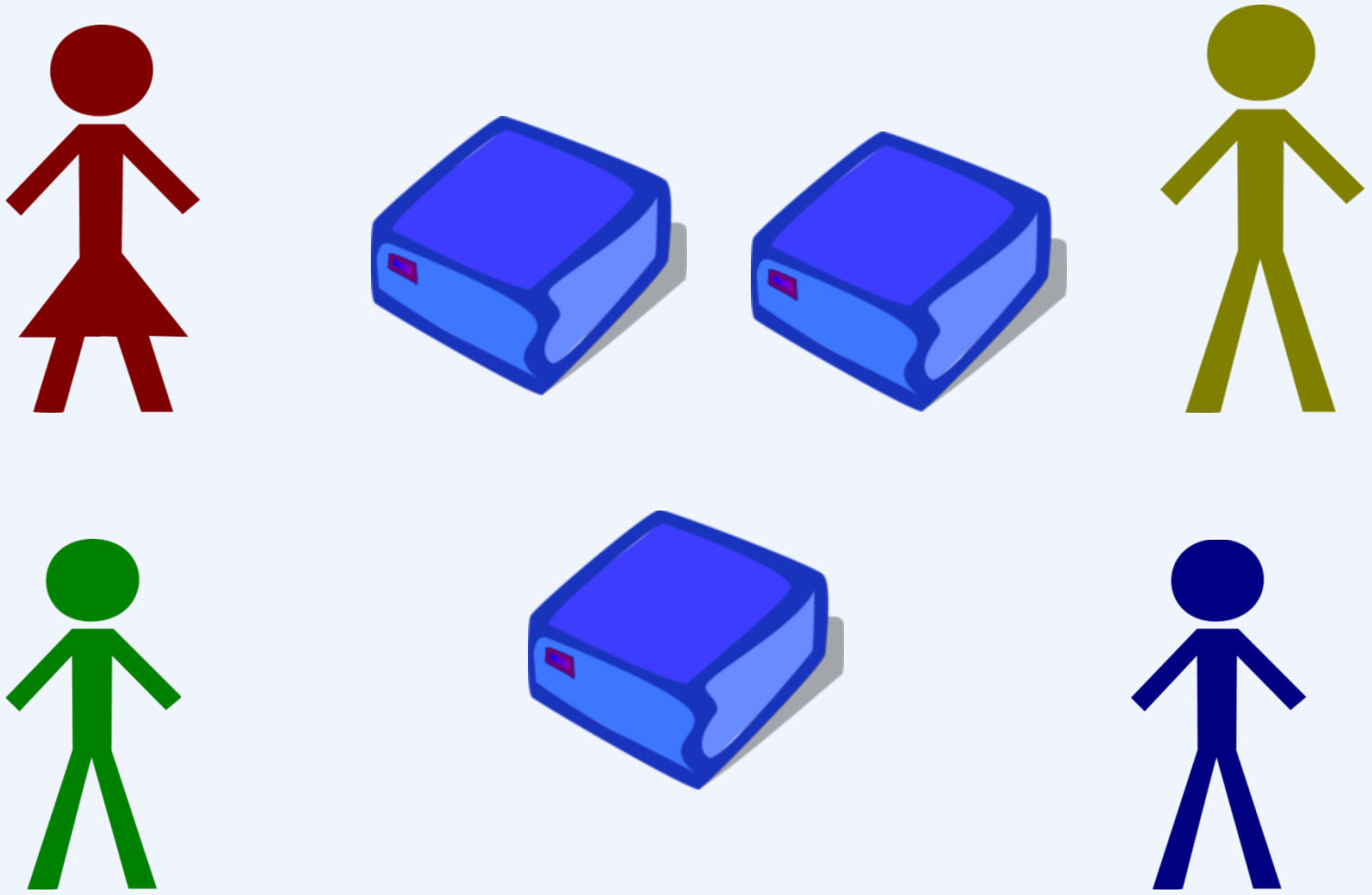
Adding a new Datacenter

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

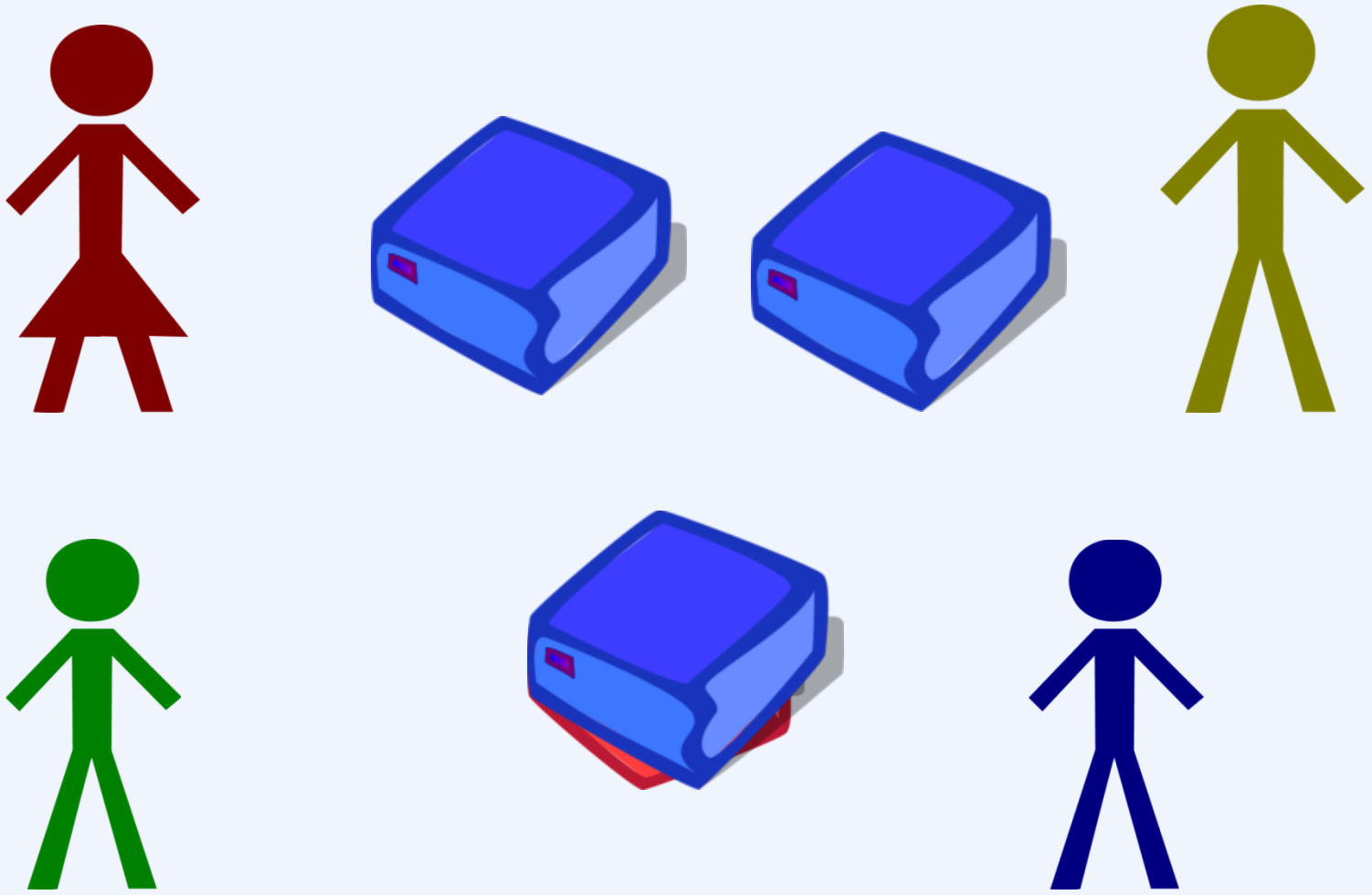


Grades Database

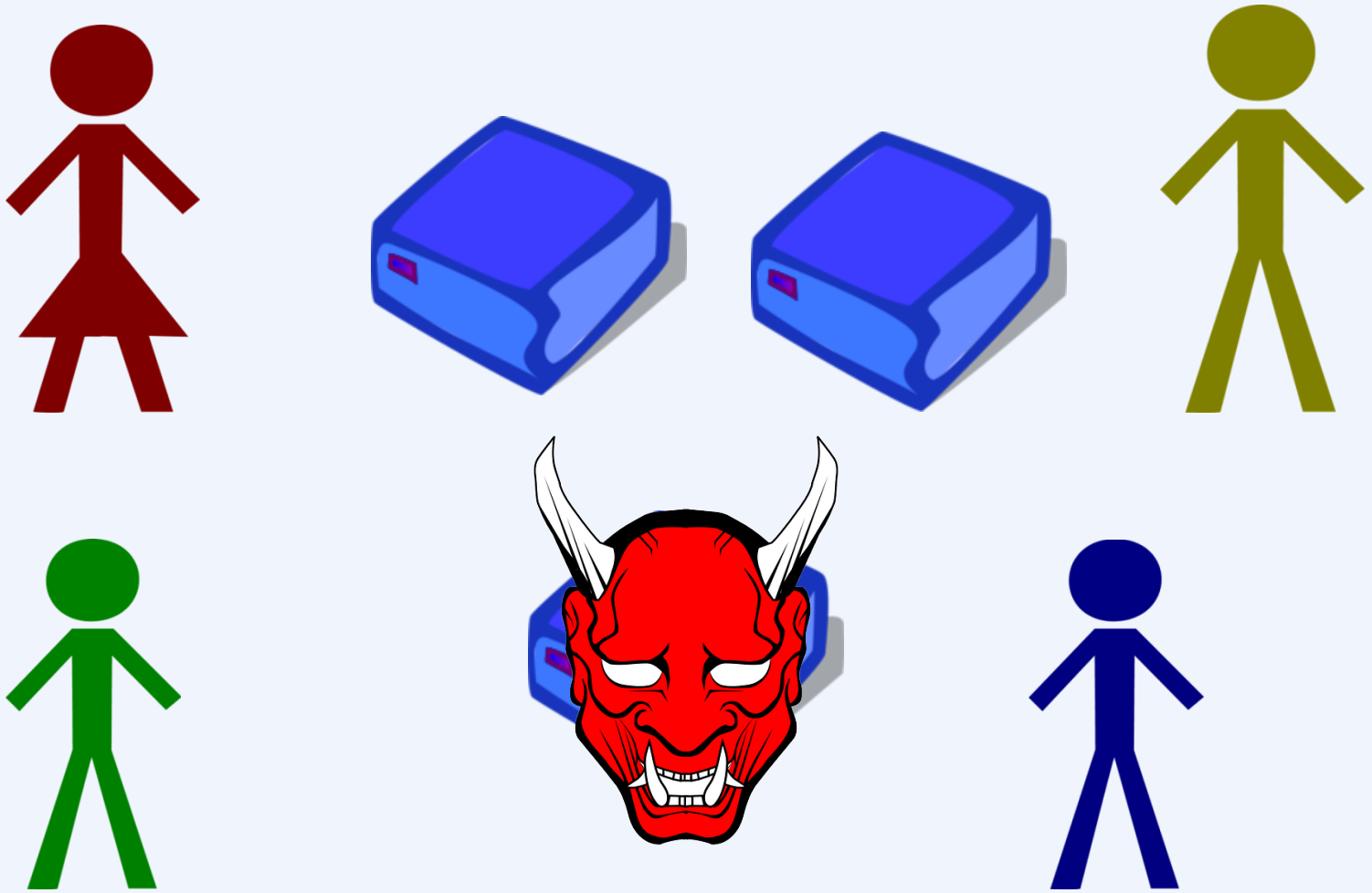
Distributed Grades Database



Failure



Byzantine Failure



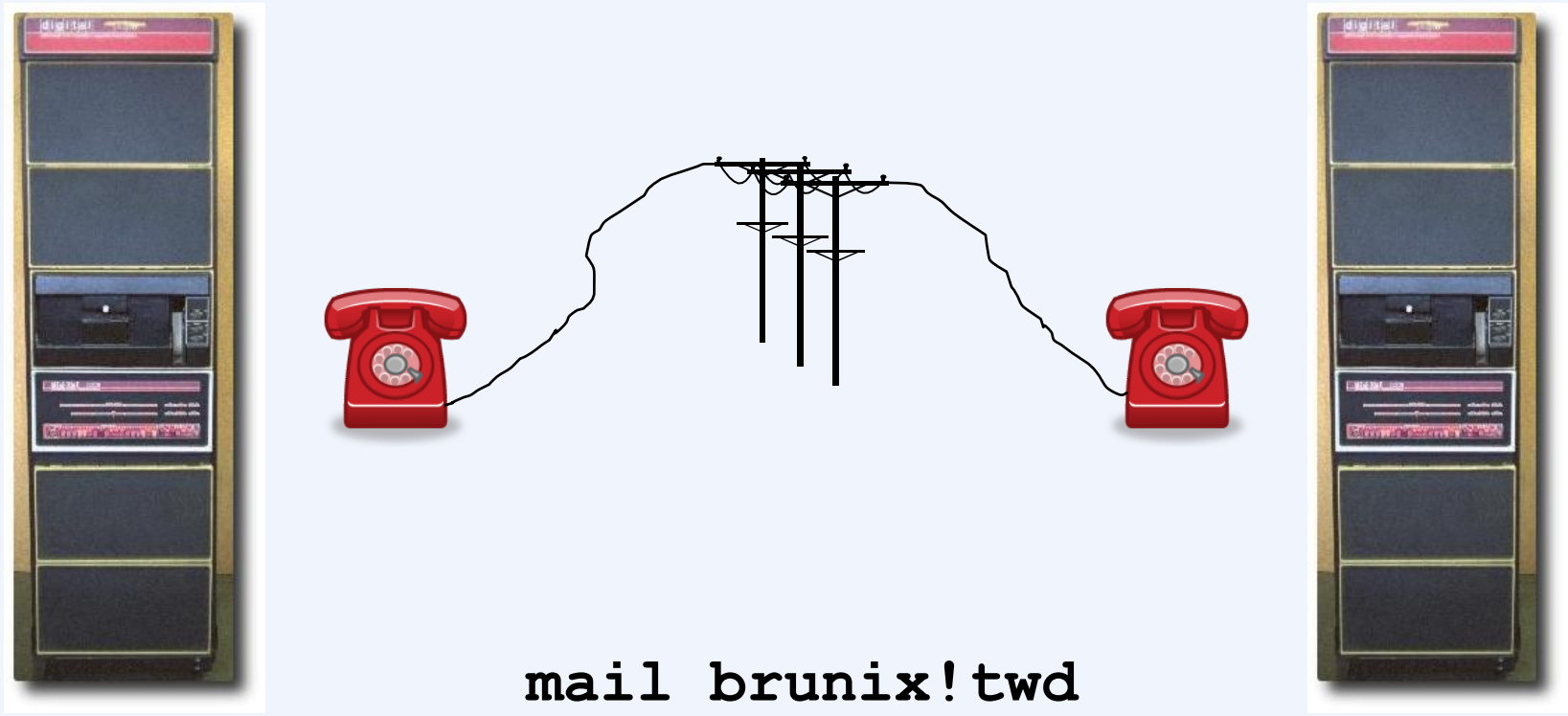
Application Examples

- Email
- DNS
- Content Distribution Networks

Email: Ancient History

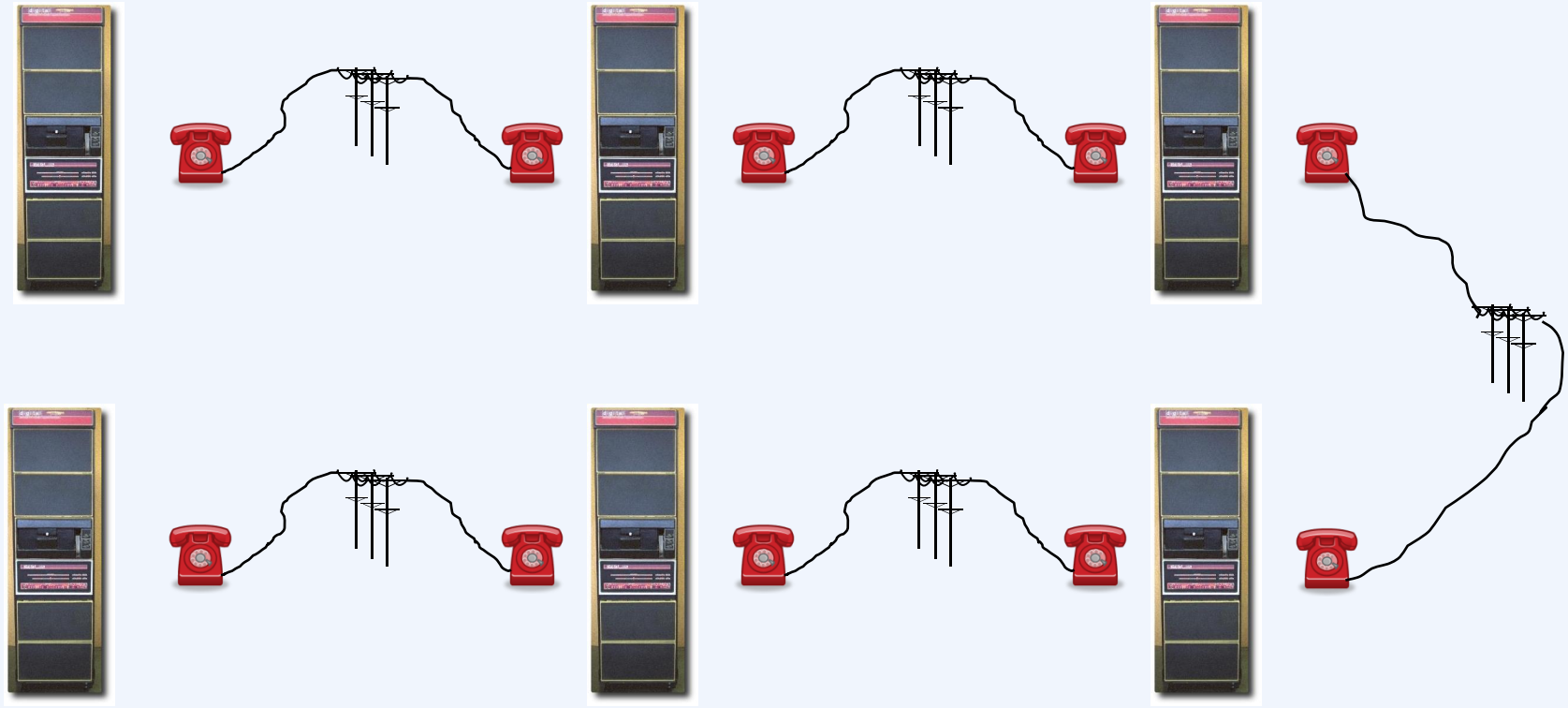
```
mail twd
```

Enter UUCP: Distributed Email



```
mail brunix!twd
```

But ...



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

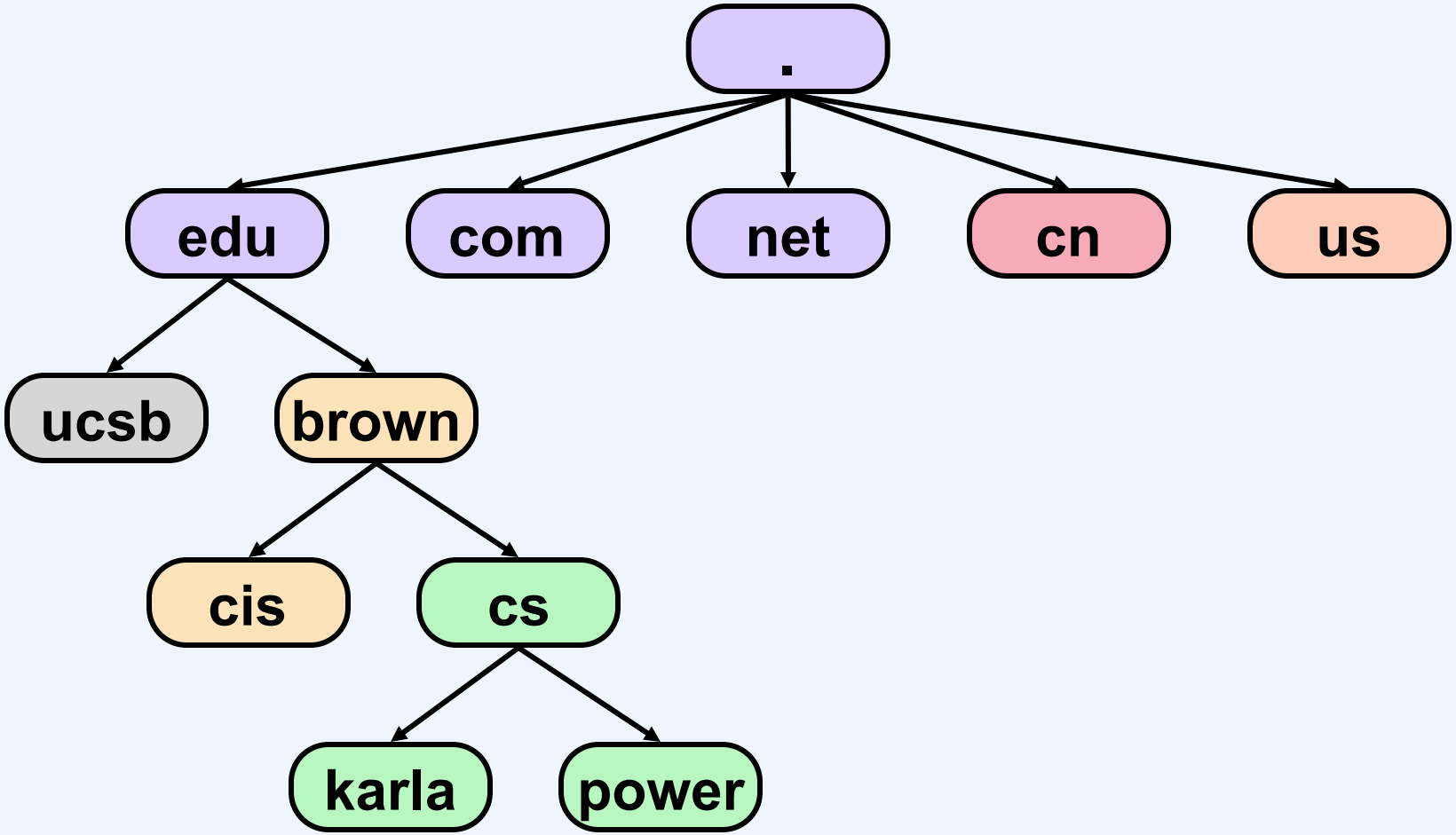
On My 1989 Business Card …

```
{decvax,ihnp4}!brunix!twd

twd@cs.brown.edu

twd@browncs
```

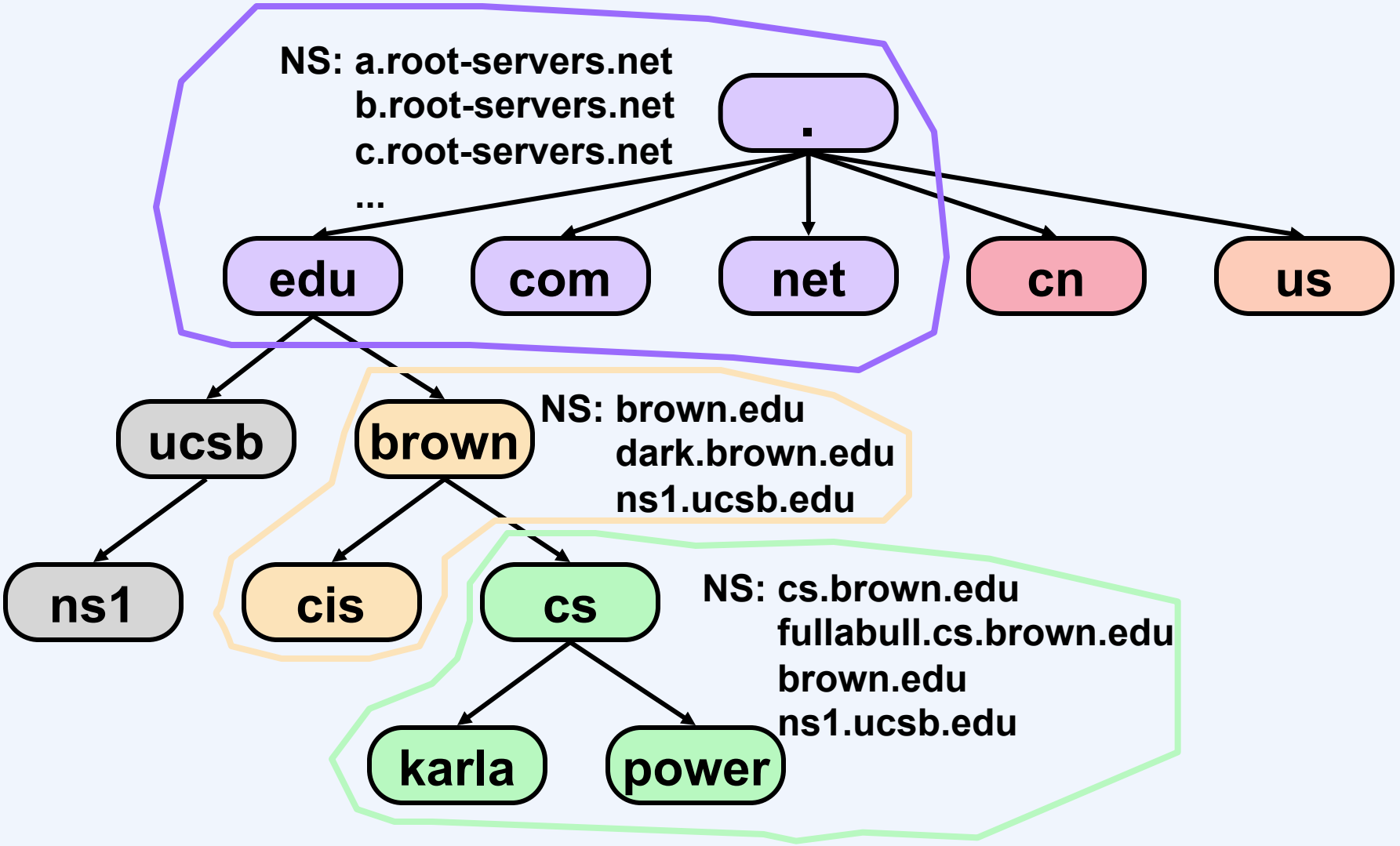
Domain Name System

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

Example



Name Servers



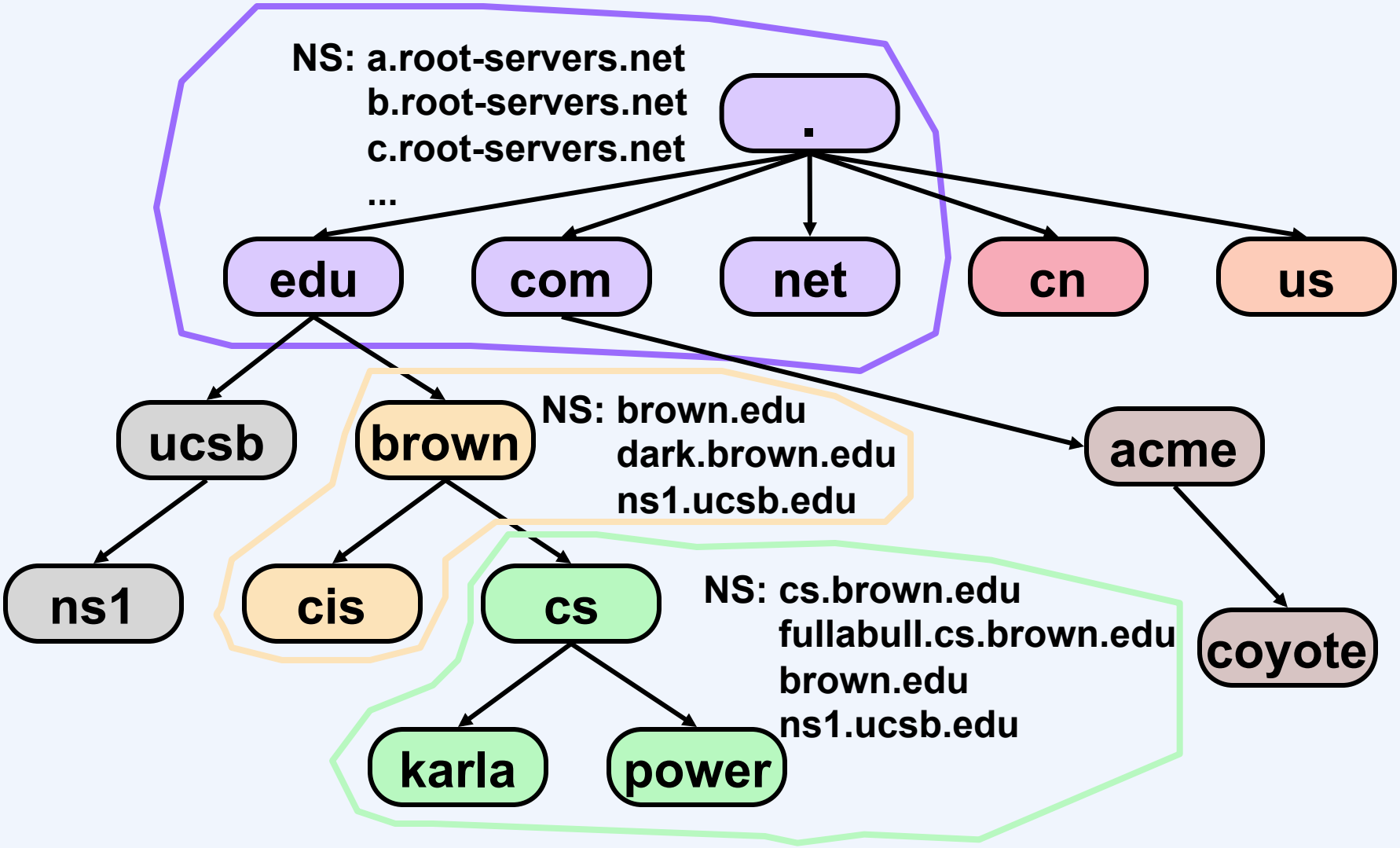
Replicating Name Servers

- 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!)

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

Lookups (Example)



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)

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**

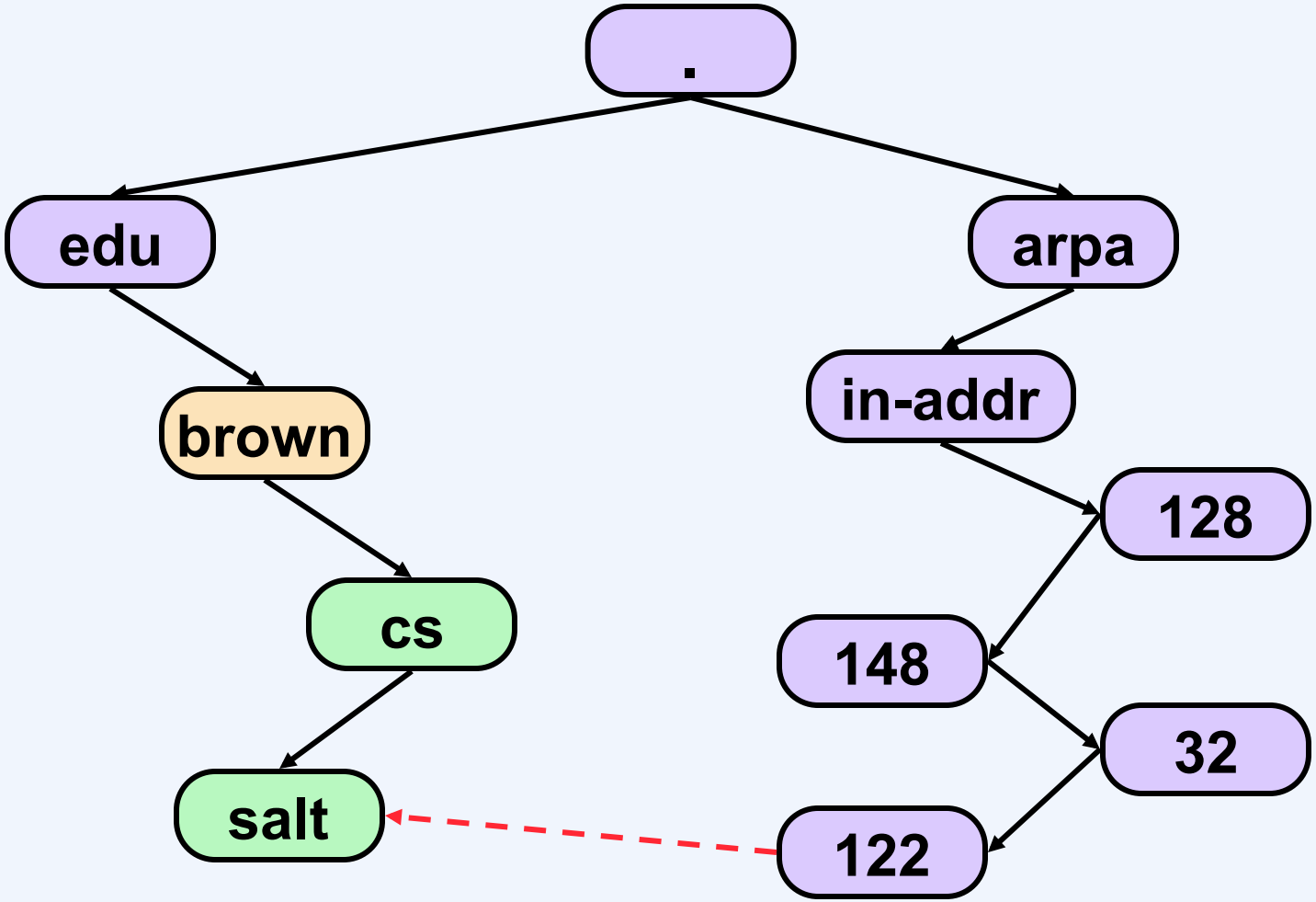
Administration

```
$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
```

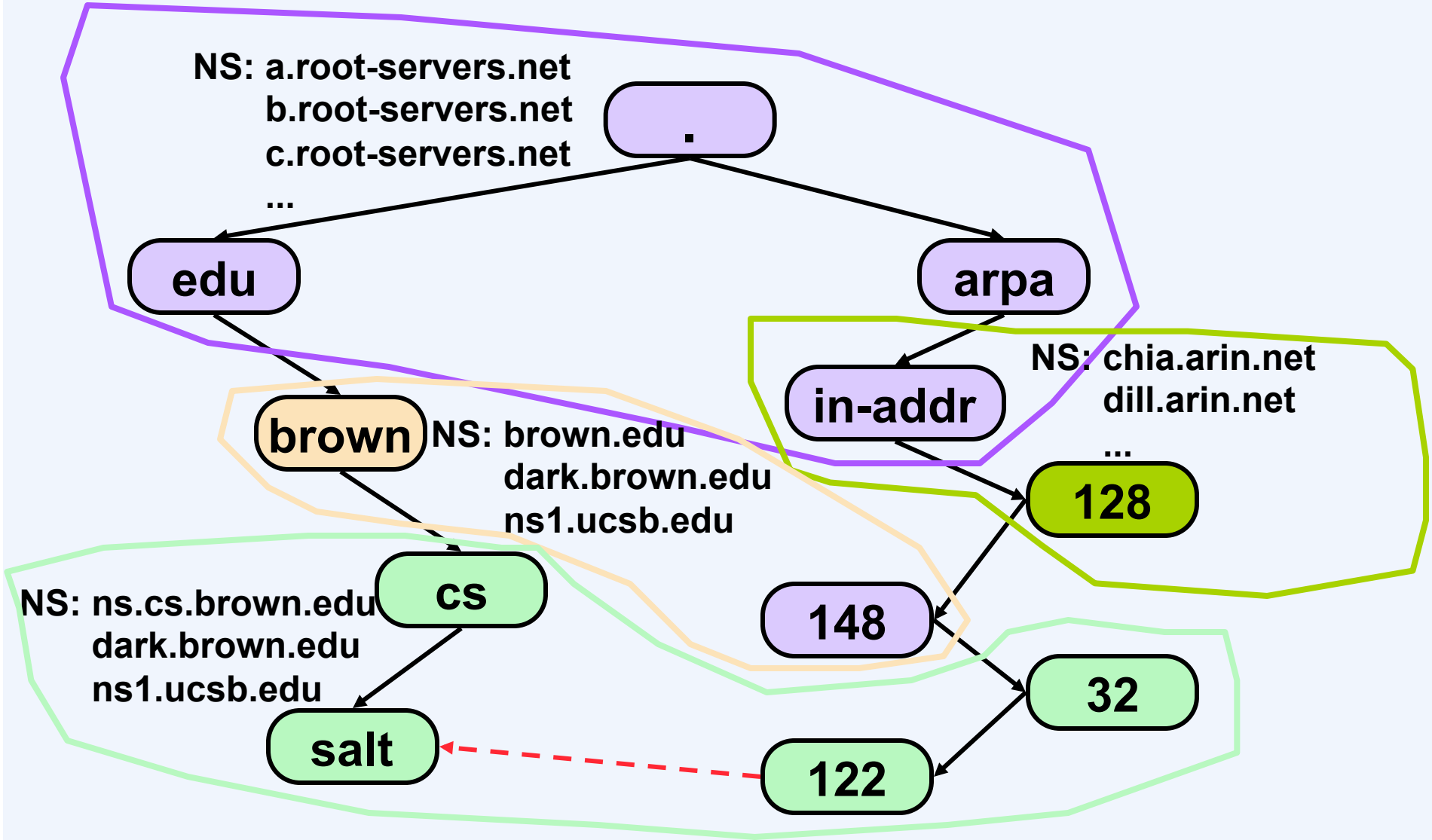
Who Are You?

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

Who Are You?



Who Are You?



More Administration

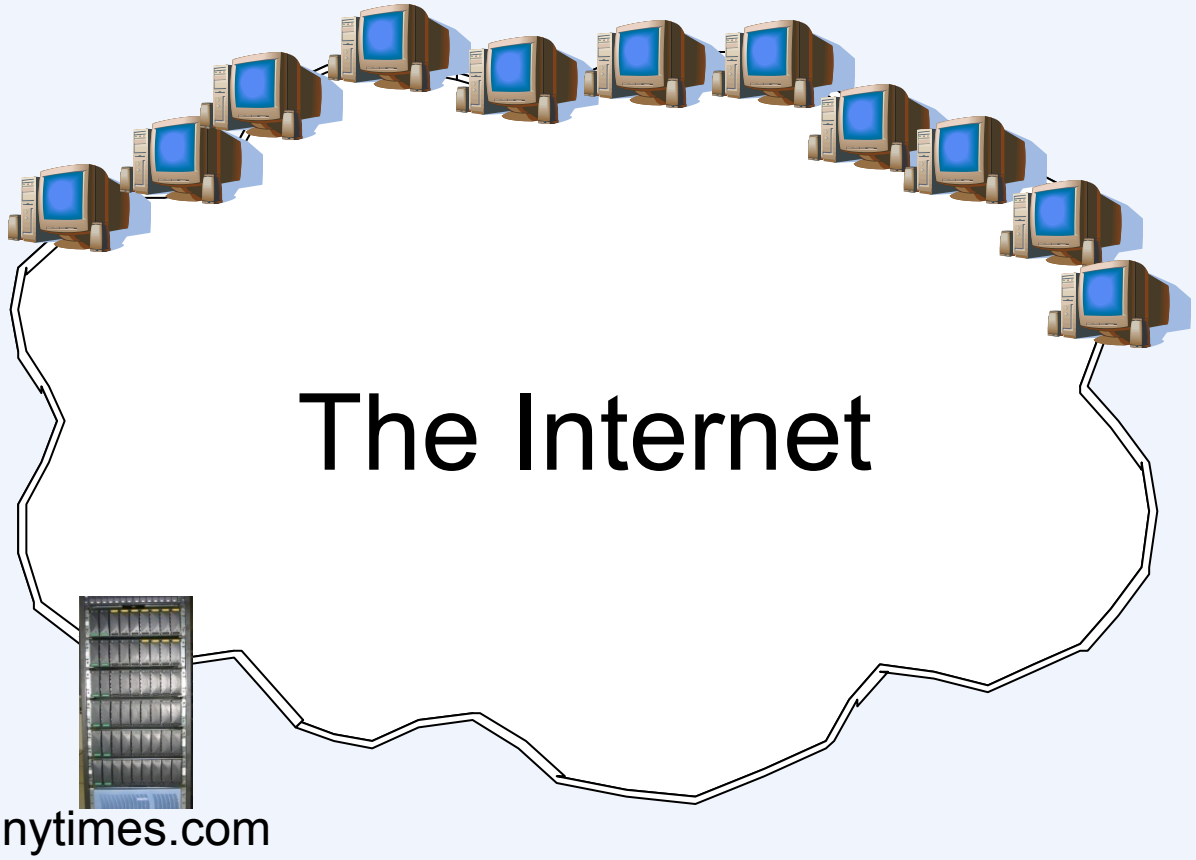
```
$ORIGIN .
$TTL 86400          ; 1 day
32.148.128.IN-ADDR.ARPA  IN SOA   ns.cs.brown.edu. root.cs.brown.edu. (
                         5874     ; serial
                         10800     ; refresh (3 hours)
                         3600     ; retry (1 hour)
                         604800    ; expire (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.
```

Recap: Issues

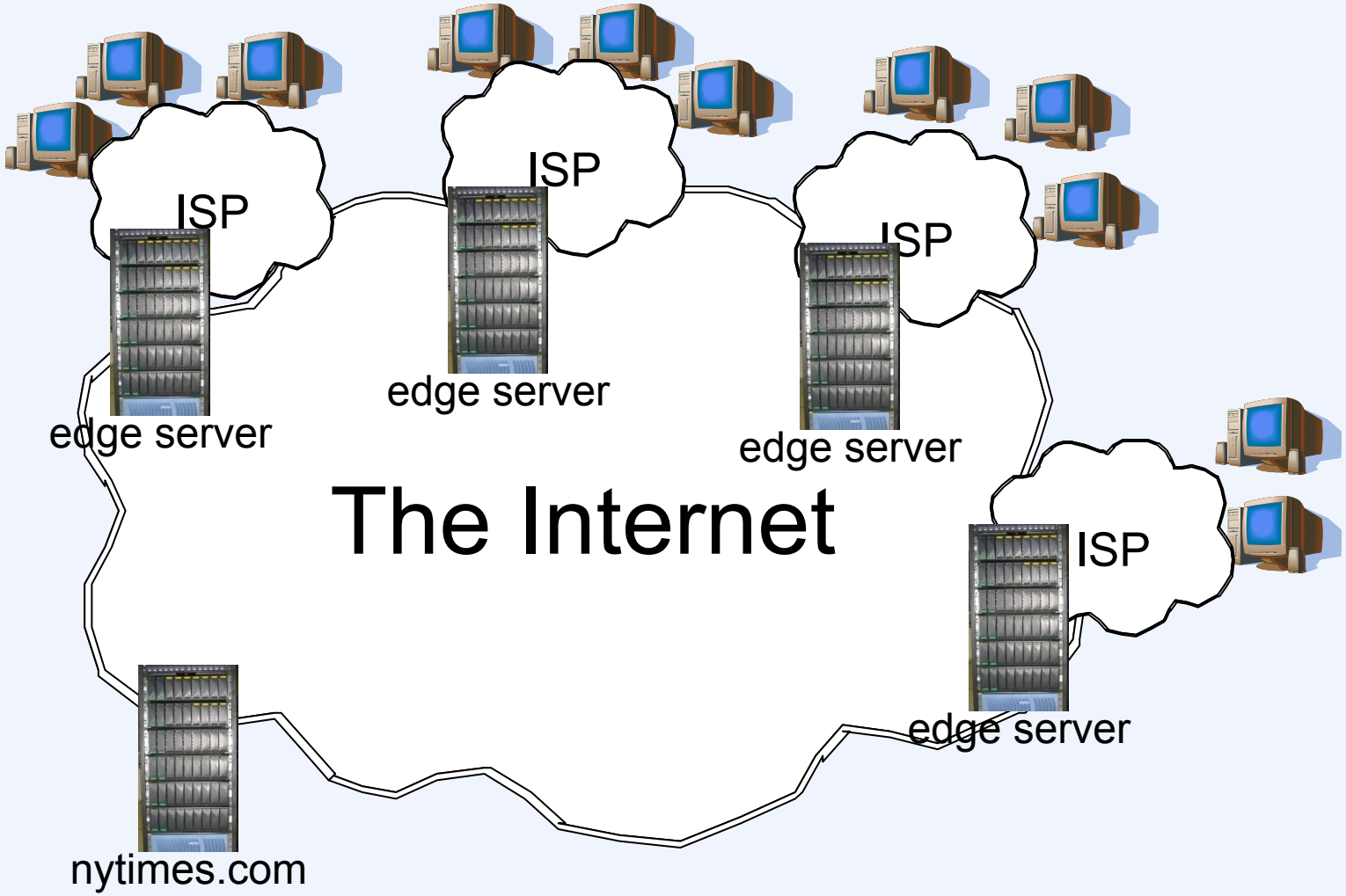
- Failure tolerance
- Decentralized management
- Speed vs. consistency

“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.”

Content Delivery



Content Delivery Network



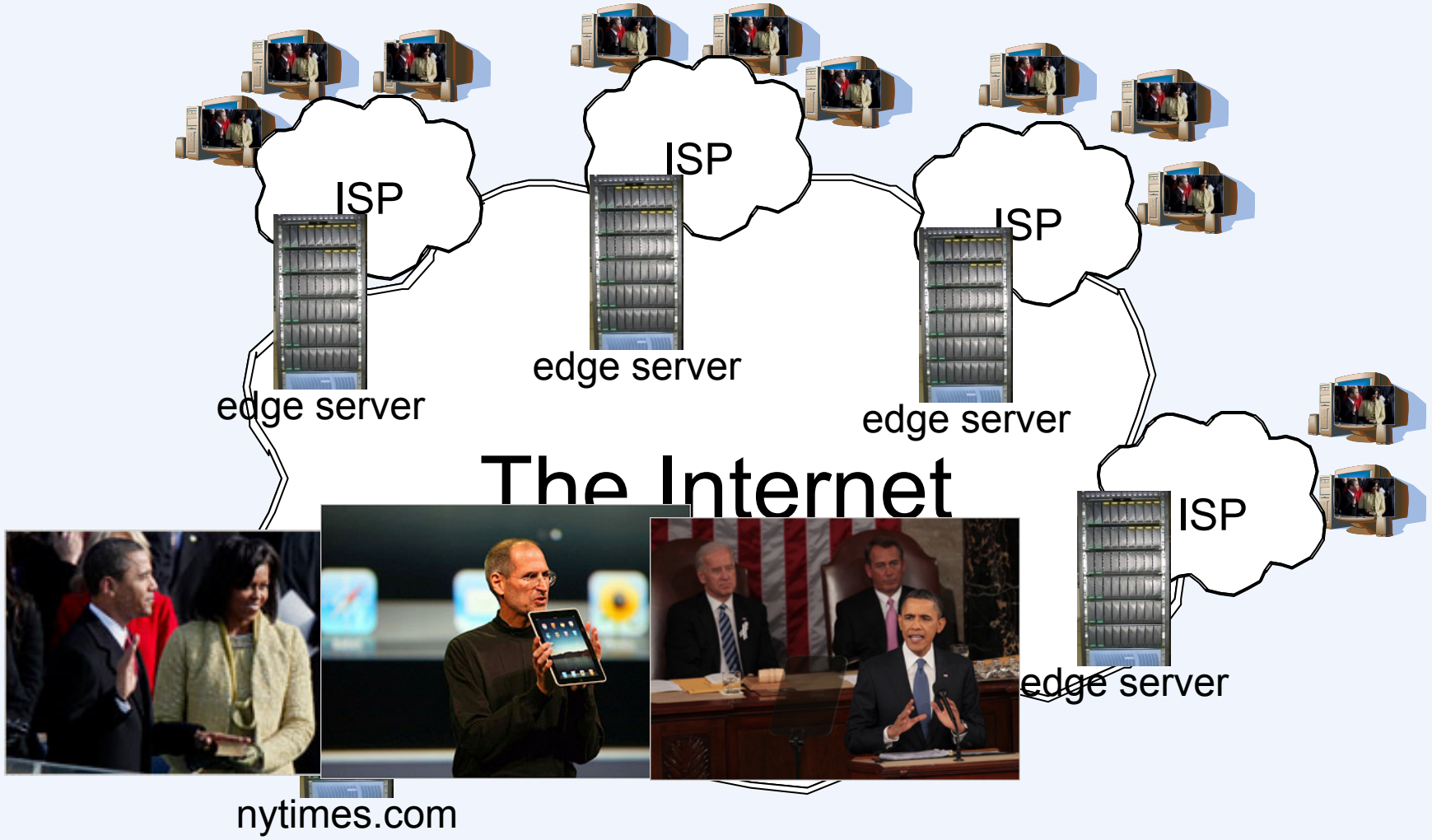
What Sort of Content?

- 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

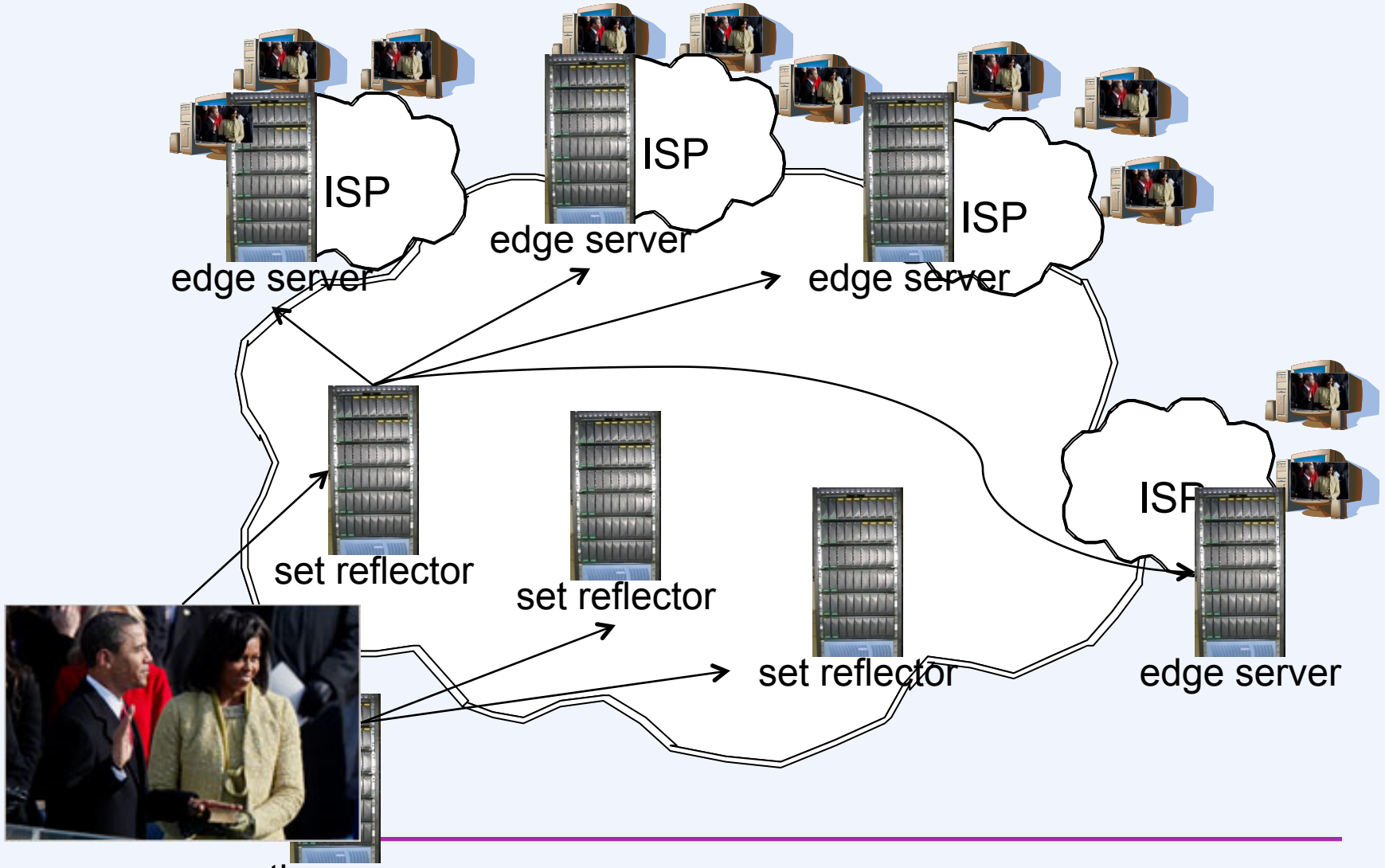
How Is It Done?

- 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

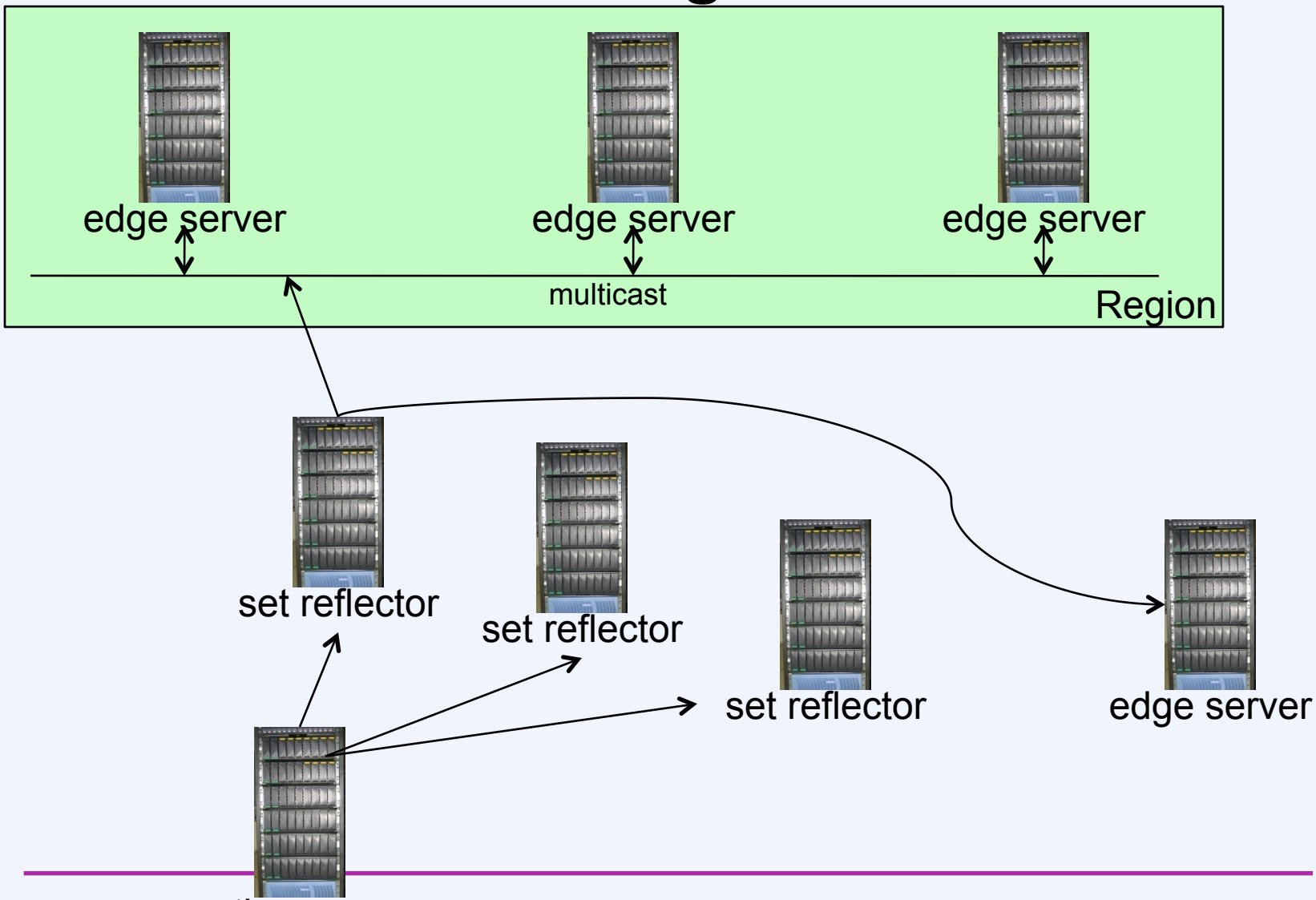
Dynamic Content
Streaming Video



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

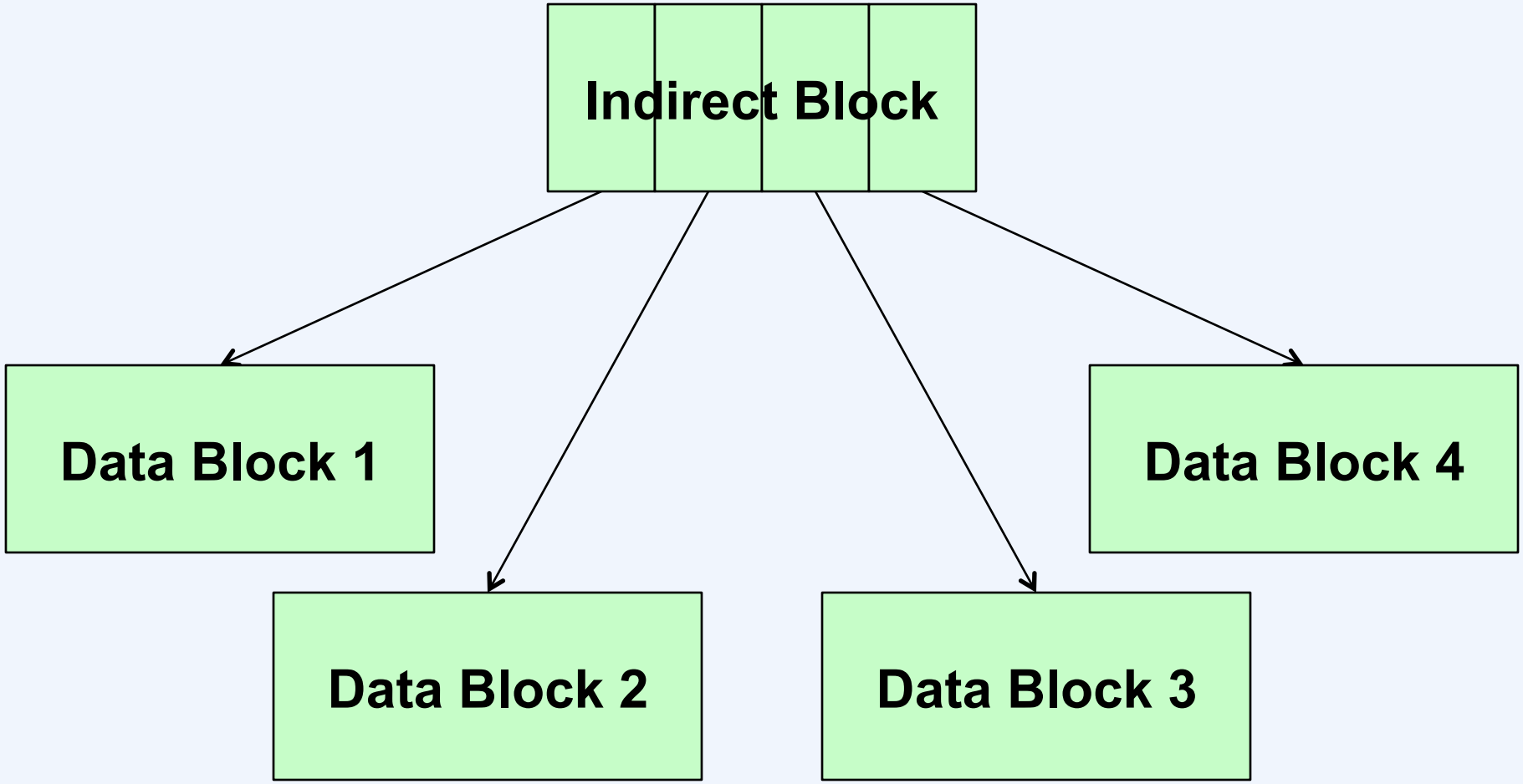
Learning Go

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

PuddleStore

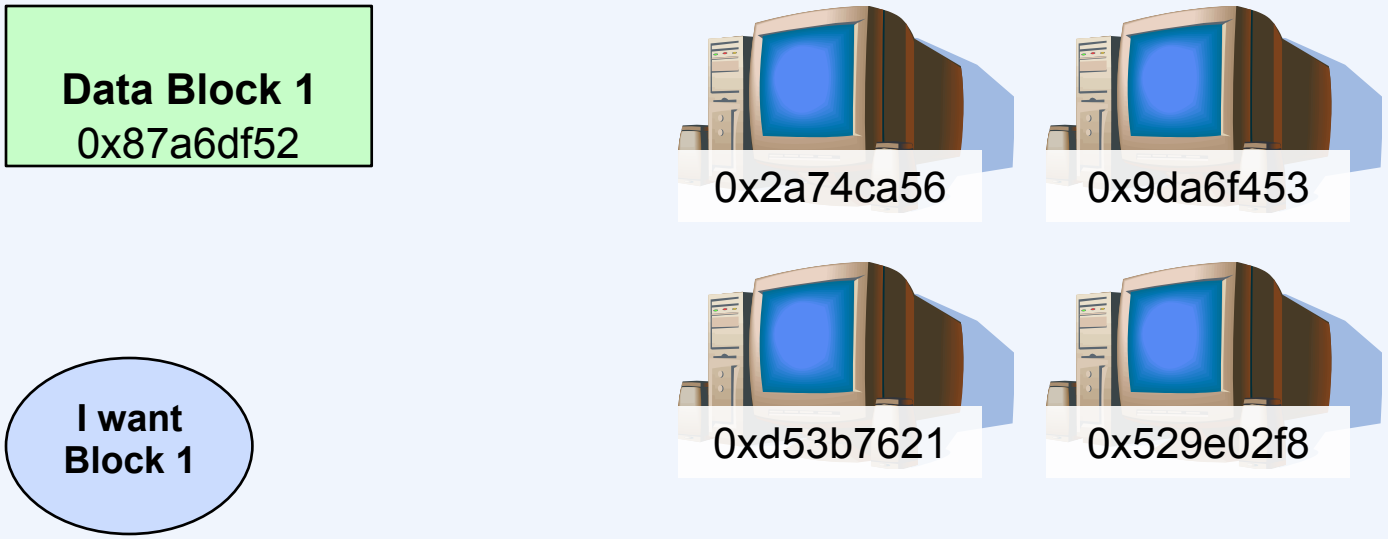
- A very distributed file system
  - thousands of computers
    - all over the world
      - (or at least throughout the SunLab)
    - no common administration
  - each holds pieces of a few files
    - pieces replicated on many computers
- Based on OceanStore
  - and its Pond prototype

A File



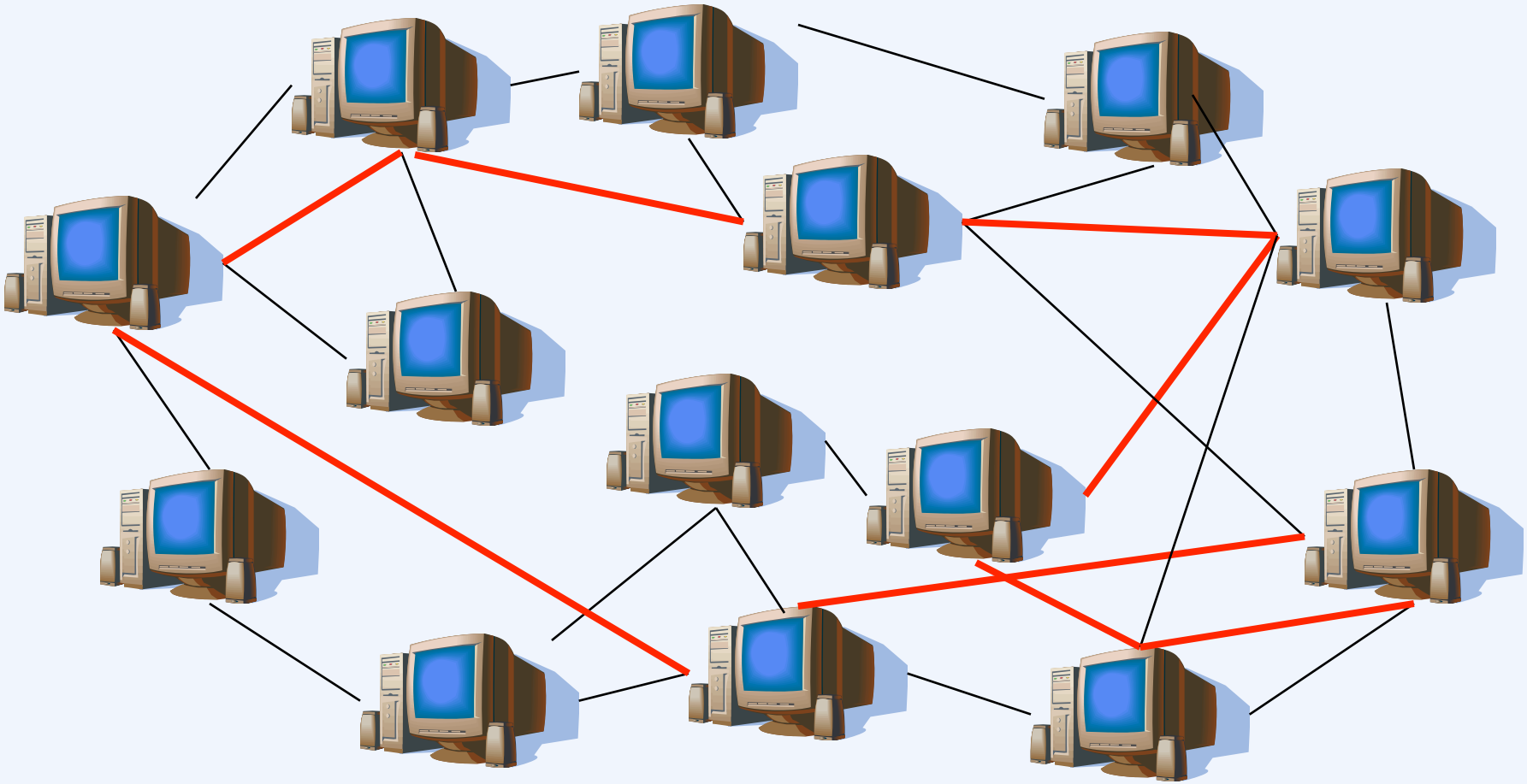
A Distributed File

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

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

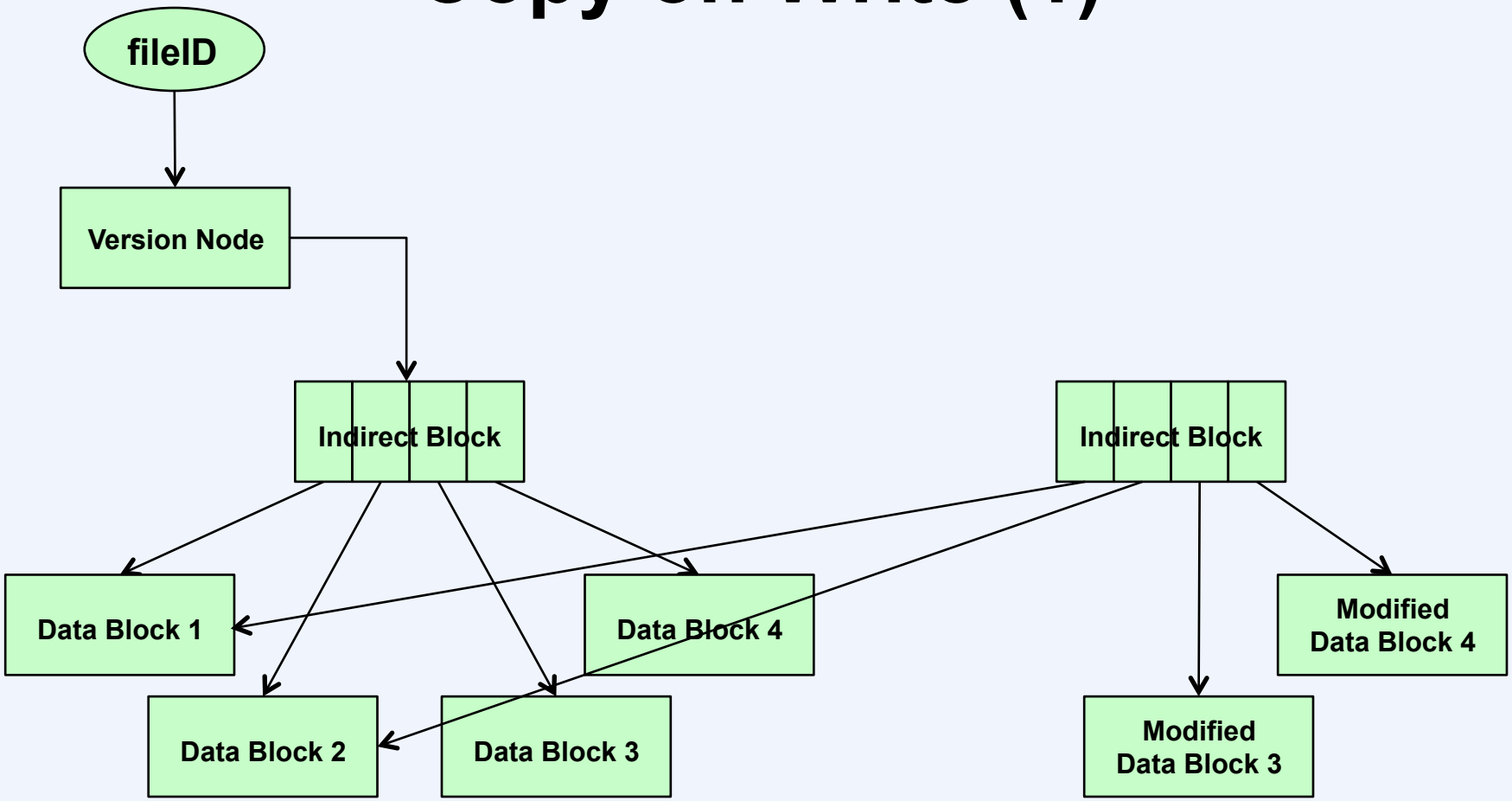
Tapestry

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

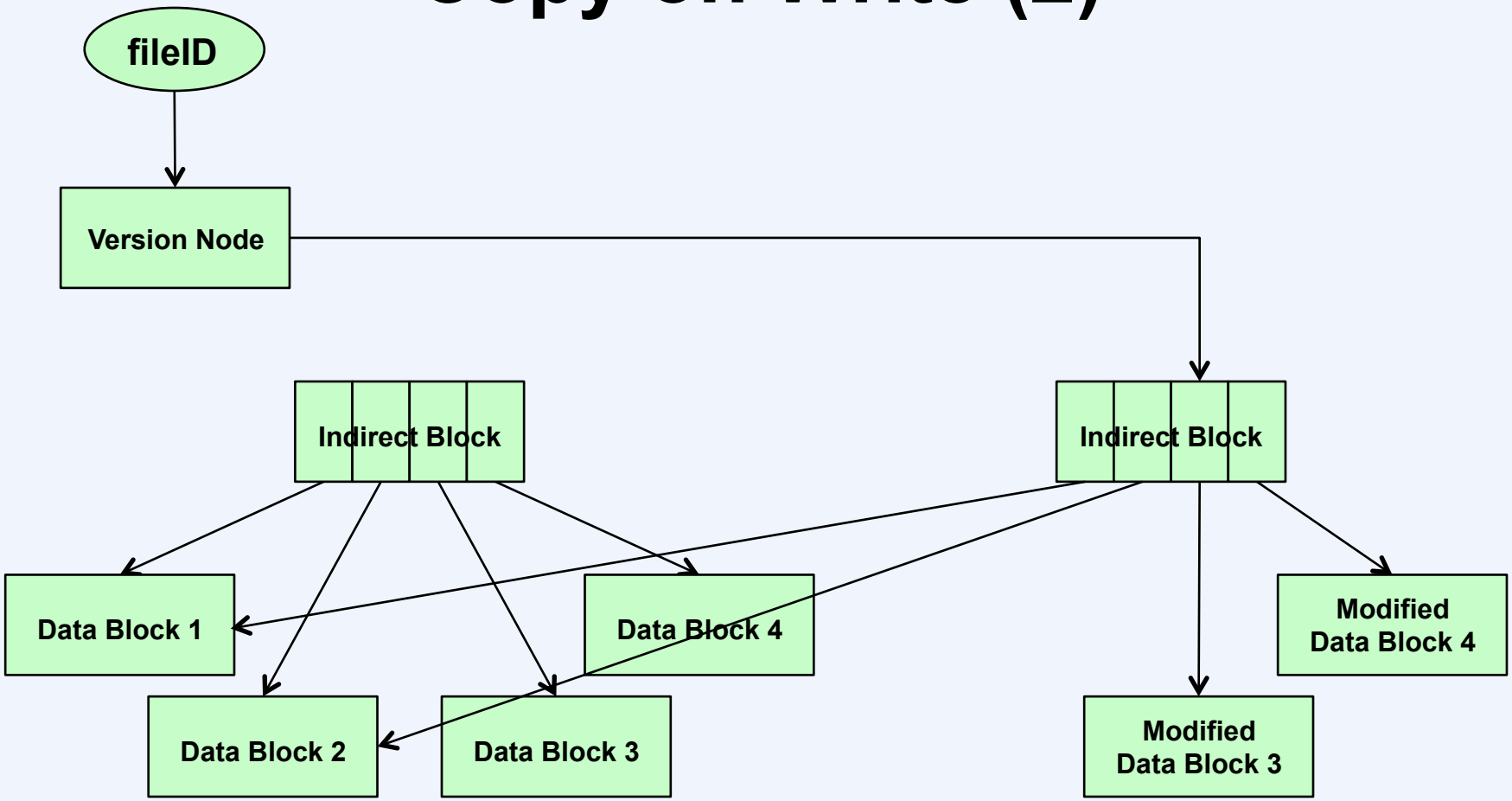
More PuddleStore Issues

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

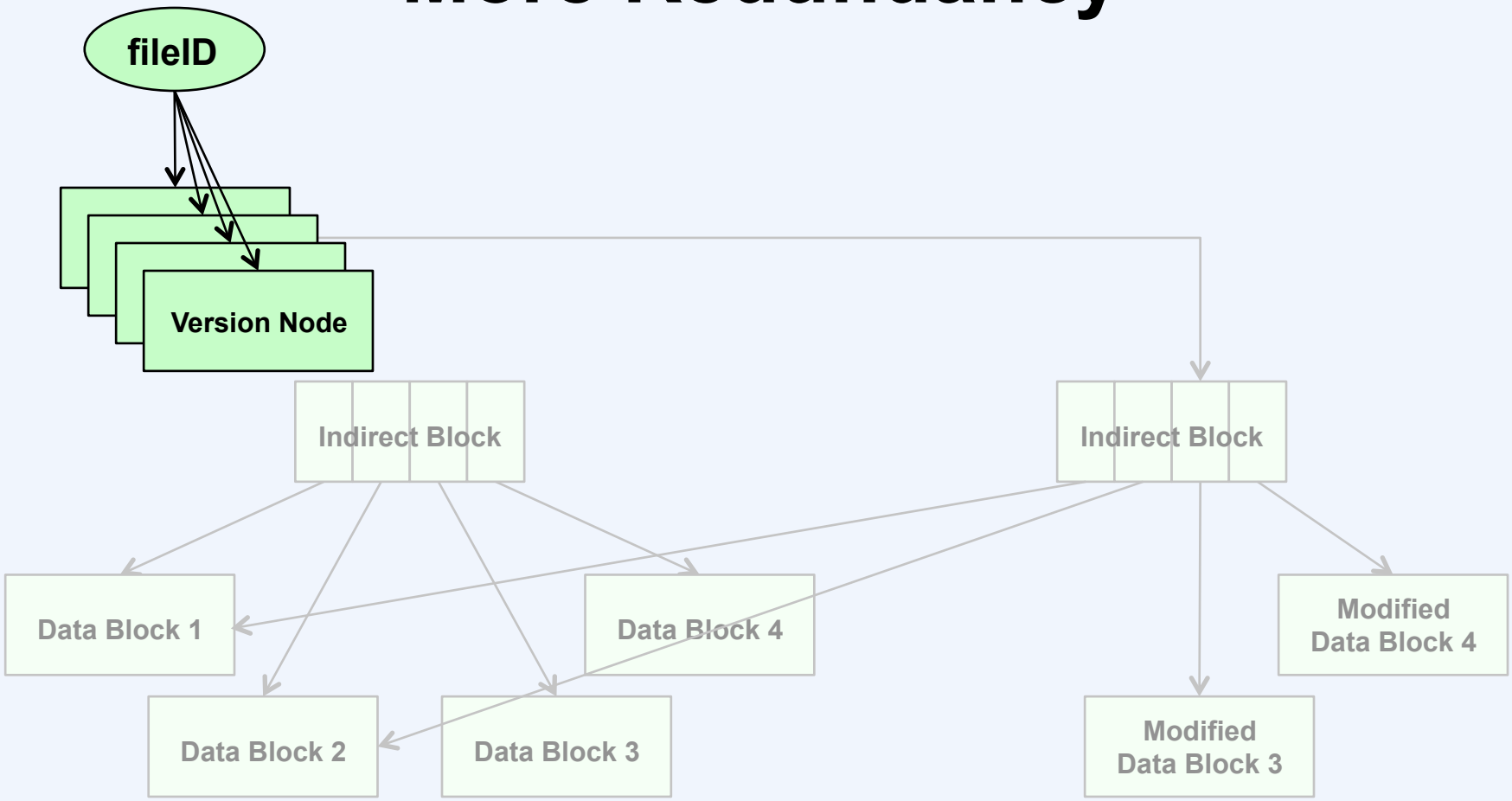
Copy on Write (1)



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