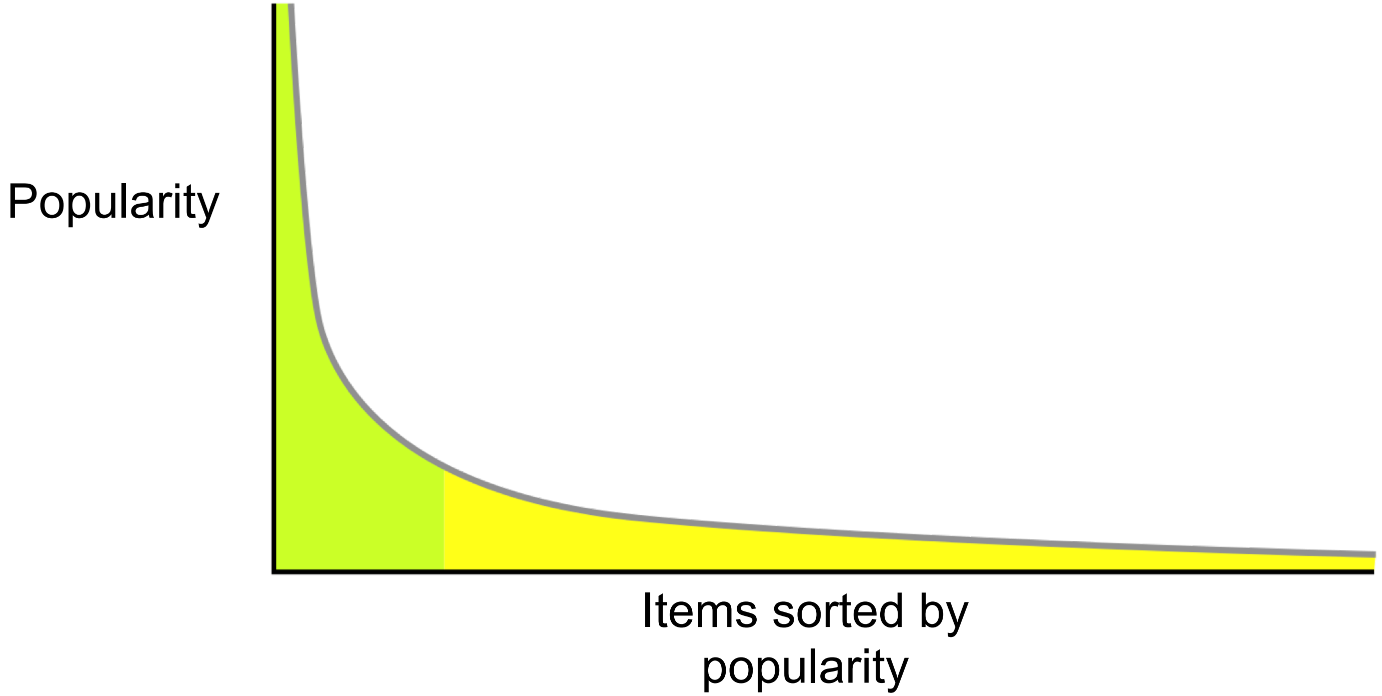
Web Content Delivery

Yao Liu

Content Distribution Problem

- Power law (Zipf distribution)
  - Models a lot of natural phenomena
  - Social graphs, media popularity, wealth distribution, etc.
  - Happens in the Web too.

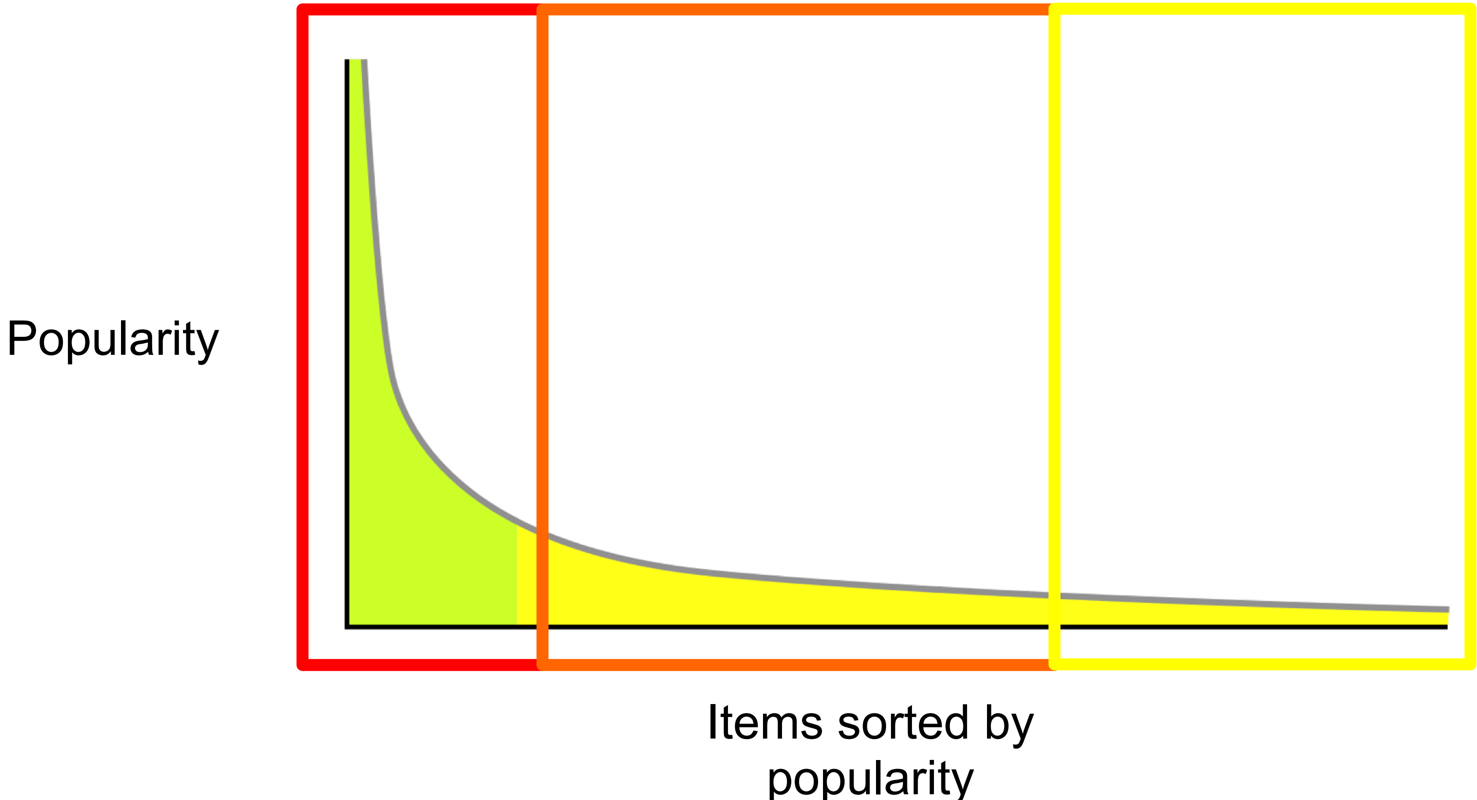


Content Distribution Workload

- What are the most frequent things you do on Facebook?
  - Read/write wall posts/comments/likes
  - View/upload photos
  - Very different in their characteristics
- Read/write wall posts/comments/likes
  - Mix of reads and writes so more care is necessary in terms of consistency
  - But small in size so probably less performance sensitive
- Photos
  - Write-once, read-many so less care is necessary in terms of consistency
  - But large in size so more performance sensitive

Facebook's Photo Distribution Problem

- "Hot" vs. "very warm" vs. "warm" photos
  - Hot: Popular, a lot of views
  - Very warm: Somewhat popular, still a lot of views
  - Warm: Unpopular, but still a lot of views in aggregate



"Hot" Photos

- How would you serve these photos?
- Caching should work well.
  - Many views for popular photos
- Where should you cache?
  - Close to users
- What's commonly used these days?
  - CDN
  - CDN mostly relies on DNS, so we'll look at DNS then CDN

DNS and CDN

Domain Name System (DNS)

- On the Internet, hosts are identified by IP addresses.
- However, addresses are hard for users to remember.
- A **user-friendly name** is also typically assigned to each host in a network.
- Use **DNS protocol** to map between host names and IP addresses.
- DNS is a *distributed database* implemented in hierarchy of many ***N**ame **S**ervers* (NS).
- It is an application-layer protocol that provides core Internet function: hosts, name servers communicate to ***resolve*** names (address/name translation)

DNS Service

- Hostname to IP address translation
- Host aliasing
  - canonical, alias names
- Mail server aliasing
- Load distribution
  - replicated Web servers: many IP addresses correspond to one name

Strawman Solution #1: Local File

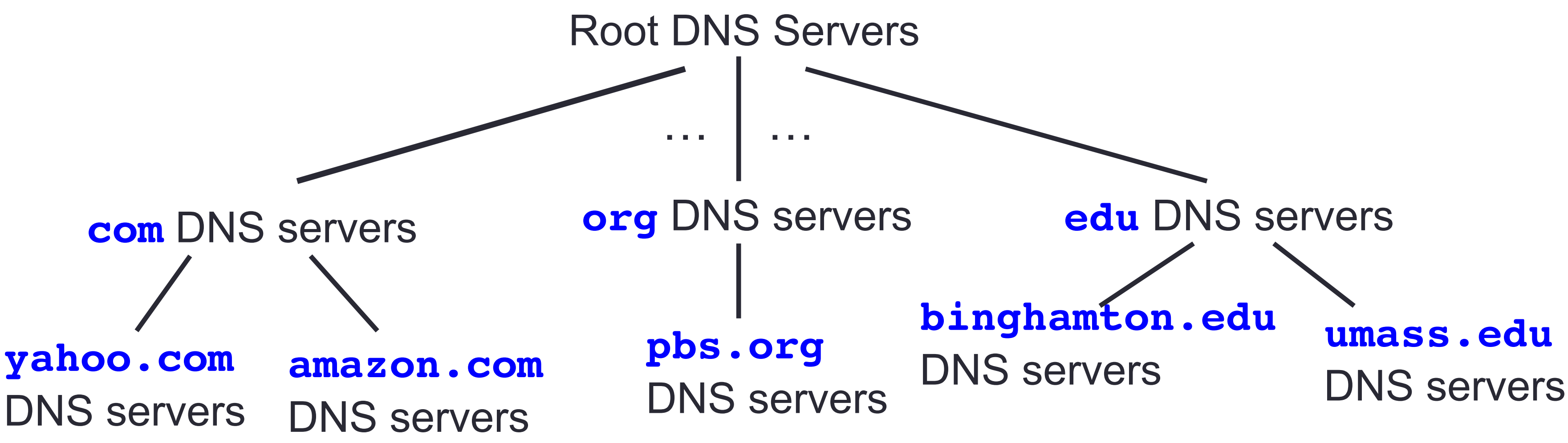
- Original name to address mapping
  - Flat namespace
  - /etc/hosts
  - SRI kept main copy
  - Downloaded regularly
- Count of hosts was increasing: moving from a machine per domain to machine per user
  - Many more downloads
  - Many more updates

Strawman Solution #2: Central Server

- Central server
  - One place where all mappings are stored
  - All queries go to the central server
- Many practical problems
  - Single point of failure
  - High traffic volume
  - Distant centralized database
  - Single point of update
  - Does not scale

Need a distributed, hierarchical collection of servers

DNS: a Distributed, Hierarchical Database



- Client wants IP for www.amazon.com
  - client queries root server to find **.com** DNS server
  - client queries **.com** DNS server to get **amazon.com** DNS server
  - client queries **amazon.com** DNS server to get IP address for **www.amazon.com**

DNS Root Servers

- 13 root servers (see http://www.root-servers.org/)
- Labeled A through M

A Verisign, Dulles, VA
C Cogent, Herndon, VA (also Los Angeles)
D U Maryland College Park, MD
G US DoD Vienna, VA
H ARL Aberdeen, MD
J Verisign, (11 locations)

K RIPE London (+ Amsterdam, Frankfurt)

I Autonomica, Stockholm
(plus 3 other locations)

E NASA Mt View, CA
F Internet Software C. Palo Alto, CA (and 17 other locations)

m WIDE Tokyo

B USC-ISI Marina del Rey, CA
L ICANN Los Angeles, CA

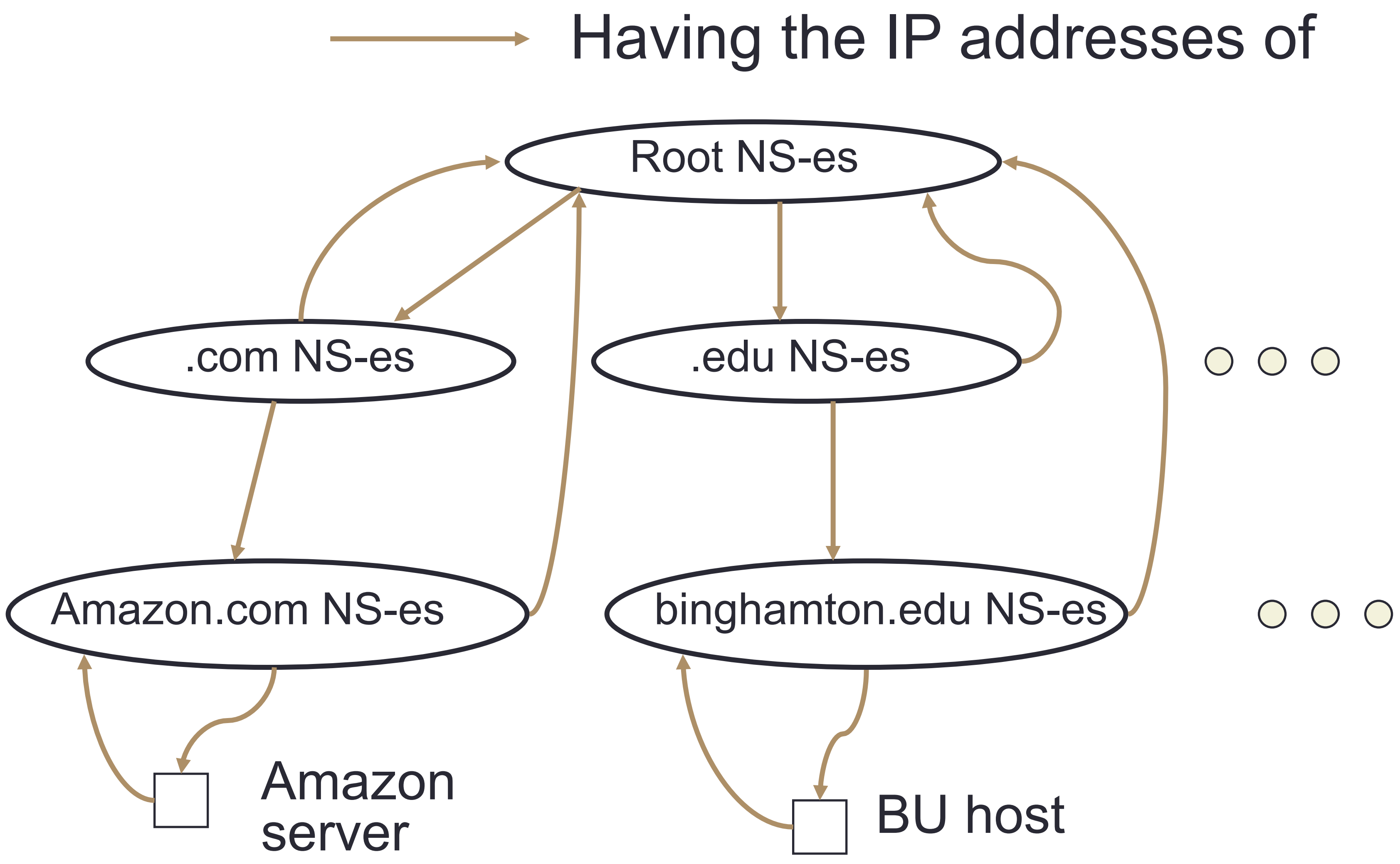
- There are 13 root NS around the world, maintaining 13 identical databases of top-level domain NS.
- Every root NS *knows* all `.com` NS, `.edu` NS, `.net` NS, `.org` NS, …
- Each .com NS is also "complete", it *knows* the NS of all 2nd-level .com domains.
  - It *knows* the NS of `amazon.com`, `google.com`, etc.
- The same applies to every `.net` NS, `.edu` NS, `.jp` NS, and so on.

"Knows" means having the IP addresses of

TLD and Authoritative DNS Servers

- Top-level domain (TLD) servers
  - Generic domains (e.g., com, org, edu)
  - Country domains (e.g., uk, fr, ca, jp)
  - Typically managed professionally
    - Network Solutions maintains servers for “com”
    - Educause maintains servers for “edu”
- Authoritative DNS servers
  - Provide public records for hosts at an organization
  - For the organization’s servers (e.g., Web and mail)
  - Can be maintained locally or by a service provider

- It is the lower-level NS that actually maintain machine addresses.
  - An Amazon NS knows the exact IP address of **`www.amazon.com`**
  - A BU NS knows the exact IP address of **`cs.binghamton.edu`**
- Each low-level NS *knows* all machines in its domain.
- Every NS in the world has the list of root NS.
- Each host is typically configured with the IP addresses of one or two local NSes.


Having the IP addresses of
Root NS-es
.com NS-es
.edu NS-es
Amazon.com NS-es
binghamton.edu NS-es
Amazon server
BU host

Local Name Server

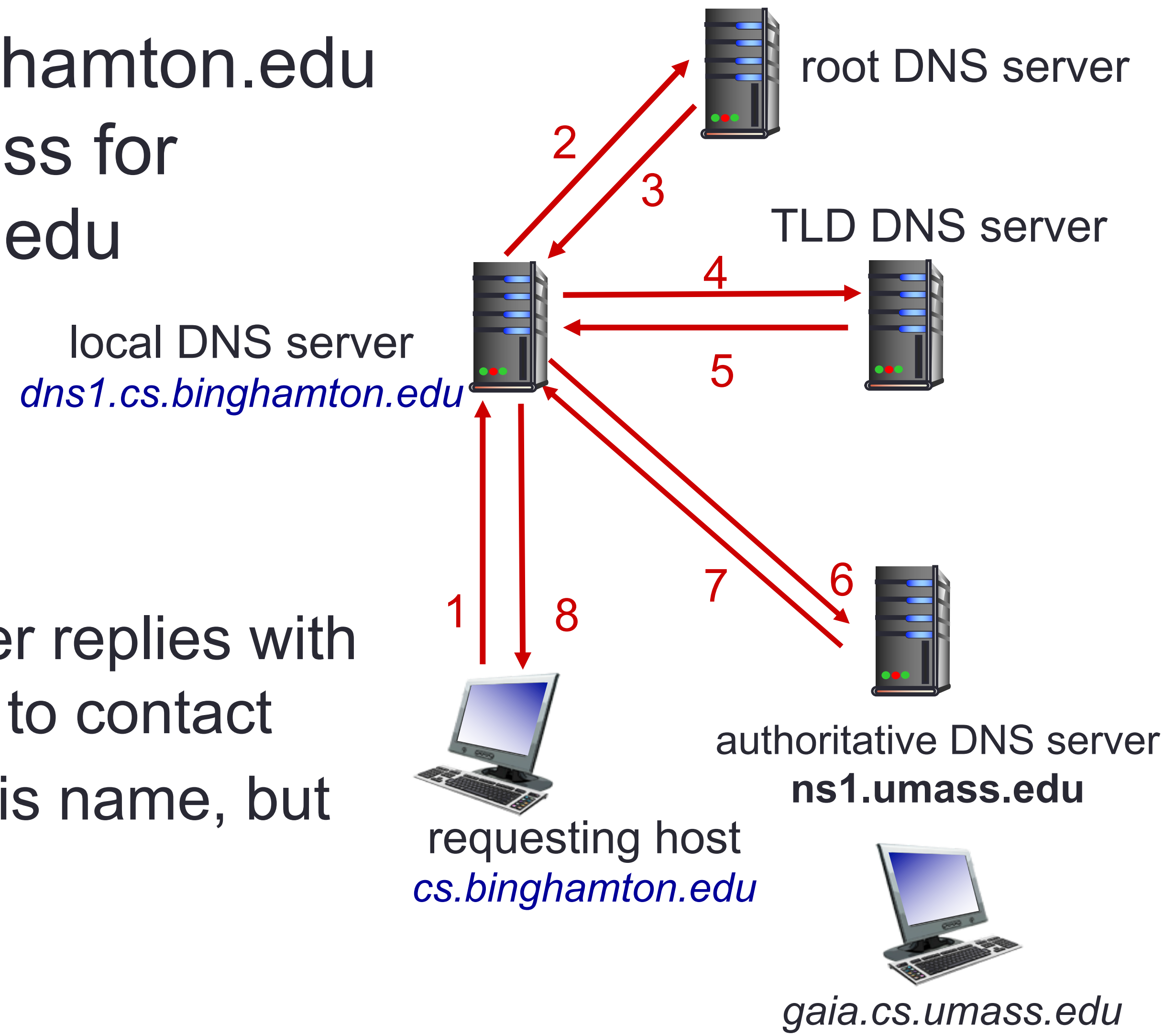
- At least one per ISP (residential ISP, company, university)
- Hosts learn local name server via:
  - DHCP (same protocol that tells host its IP address)
  - Static configuration (e.g., can use Google's "local" name server at 8.8.8.8 or 8.8.4.4)
- When a host makes a DNS query, the query is sent to its local DNS server
  - Acts as a proxy, forwards query into hierarchy
  - Reduces lookup latency for commonly searched hostnames

Application's Use of DNS

- Extract server name (e.g., from the URL)
- Do `gethostbyname()` or `getaddrinfo()` to trigger resolver code, sending messages to the local name server

DNS Name Resolution Example

- Host at cs.binghamton.edu wants IP address for gaia.cs.umass.edu
- Iterated query:
  - contacted server replies with name of server to contact
  - "I don't know this name, but ask this server"



DNS: Caching

- Performing all these queries takes time
  - And all this before the actual communication takes place
  - E.g., 1-second latency before starting Web download
- Once (any) name server learns mapping, it caches mapping
  - e.g., .com NS → IP, .edu NS → IP, google.com NS → IP, www.google.com → IP, …
  - Cache entries timeout (disappear) after some time (TTL)
  - TLD servers typically cached in local name servers
  - Thus, root name servers not often visited

DNS Record Types

Type of record	**Associated entity**	**Description**
SOA	Zone	Holds information on the represented zone
A	Host	Contains an IP address of the host this node represents
MX	Domain	Refers to a mail server to handle mail addressed to this node
SRV	Domain	Refers to a server handling a specific service
NS	Zone	Refers to a name server that implements the represented zone
CNAME	Node	Symbolic link with the primary name of the represented node
PTR	Host	Contains the canonical name of a host
HINFO	Host	Holds information on the host this node represents
TXT	Any kind	Contains any entity-specific information considered useful

The most important types of resource records forming the contents of nodes in the DNS name space.

Reliability

- DNS servers are replicated
  - Name service available if at least one replica is up
  - Queries can be load balanced between replicas
- Try alternate servers on timeout

Inserting Resource Records into DNS

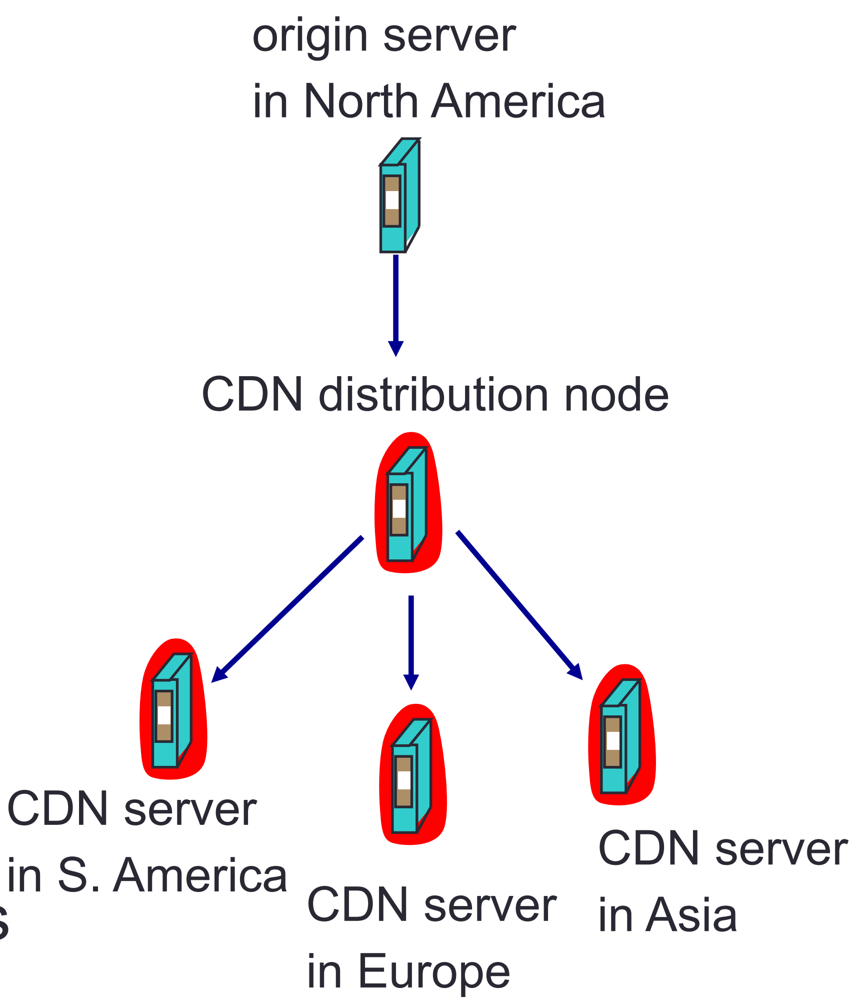
- Example: just created startup “FooBar”
- Register foobar.com at Network Solutions
  - Provide registrar with names and IP addresses of your authoritative name server (primary and secondary)
  - Registrar inserts two RRs into the com TLD server:
    - (foobar.com, dns1.foobar.com, NS)
    - (dns1.foobar.com, 212.212.212.1, A)
- Put in authoritative server dns1.foobar.com
  - Type A record for www.foobar.com
  - Type MX record for foobar.com
- Play with “dig” on UNIX

Content Distribution Networks (CDNs)

- The content providers are the CDN customers.

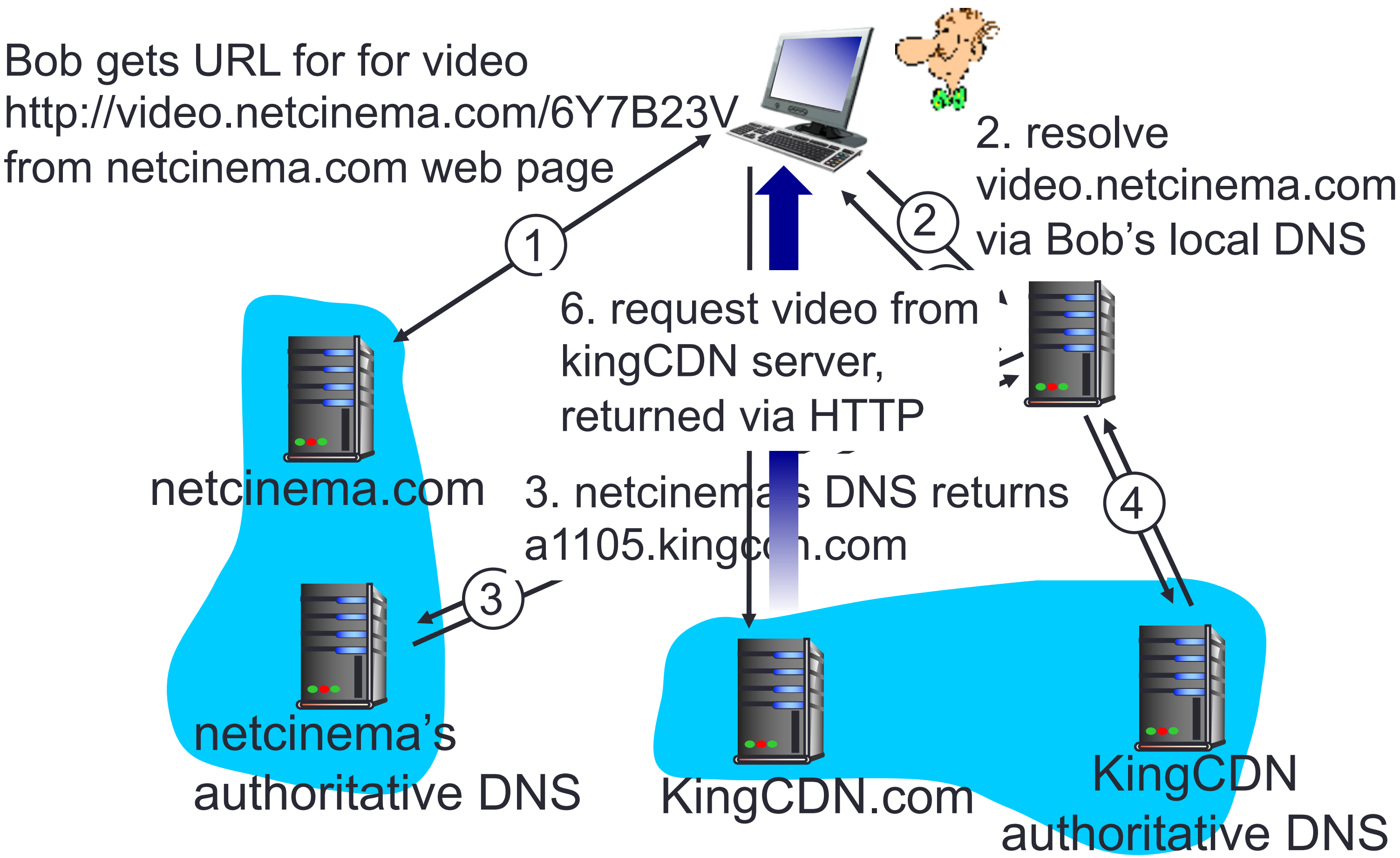
Content replication

- CDN company installs hundreds of CDN servers throughout Internet
  - in lower-tier ISPs, close to users
- CDN replicates its customers' content in CDN servers.
- When provider updates content, CDN updates servers



A (Simple) Example

- Bob (client) requests video http://video.netcinema.com/6Y7B23V
- Video served from CDN at http://a1105.kingCDN.com/6Y7B23V



CDN Server Selection

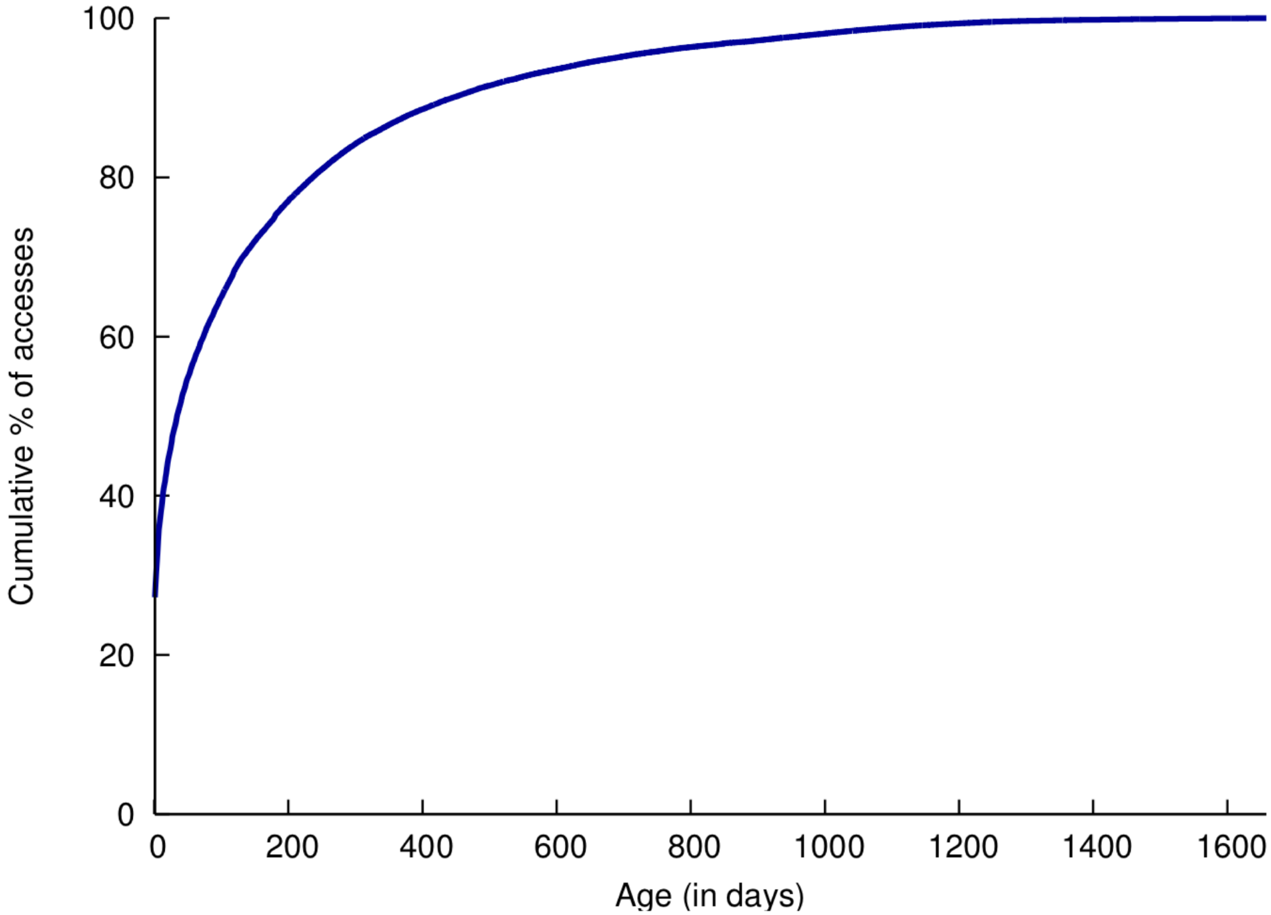
- Which server?
  - Lowest load: to balance load on servers
  - Best performance: to improve client performance
    - pick CDN node geographically closest to client
    - pick CDN node with shortest delay (or min # hops) to client (CDN nodes periodically ping access ISPs, reporting results to CDN DNS)
- How to direct clients to a particular server?
  - As part of application: HTTP redirect
  - As part of naming: DNS

Facebook Haystack

“Very warm” and “warm” Photos

- Hot photos in Facebook are served by a CDN.
- (Very) Warm photo characteristics
  - Not so much popular
  - Not entirely “cold,” i.e., occasional views
  - A lot in aggregate
  - Does not want to cache everything in CDN due to diminishing returns
- Facebook stats (in their 2010 paper)
  - 260 billion images (~20 PB)
  - 1 billion new photos per week (~60 TB)
  - One million image views per second at peak
  - Approximately 10% not served by CDN, but still a lot

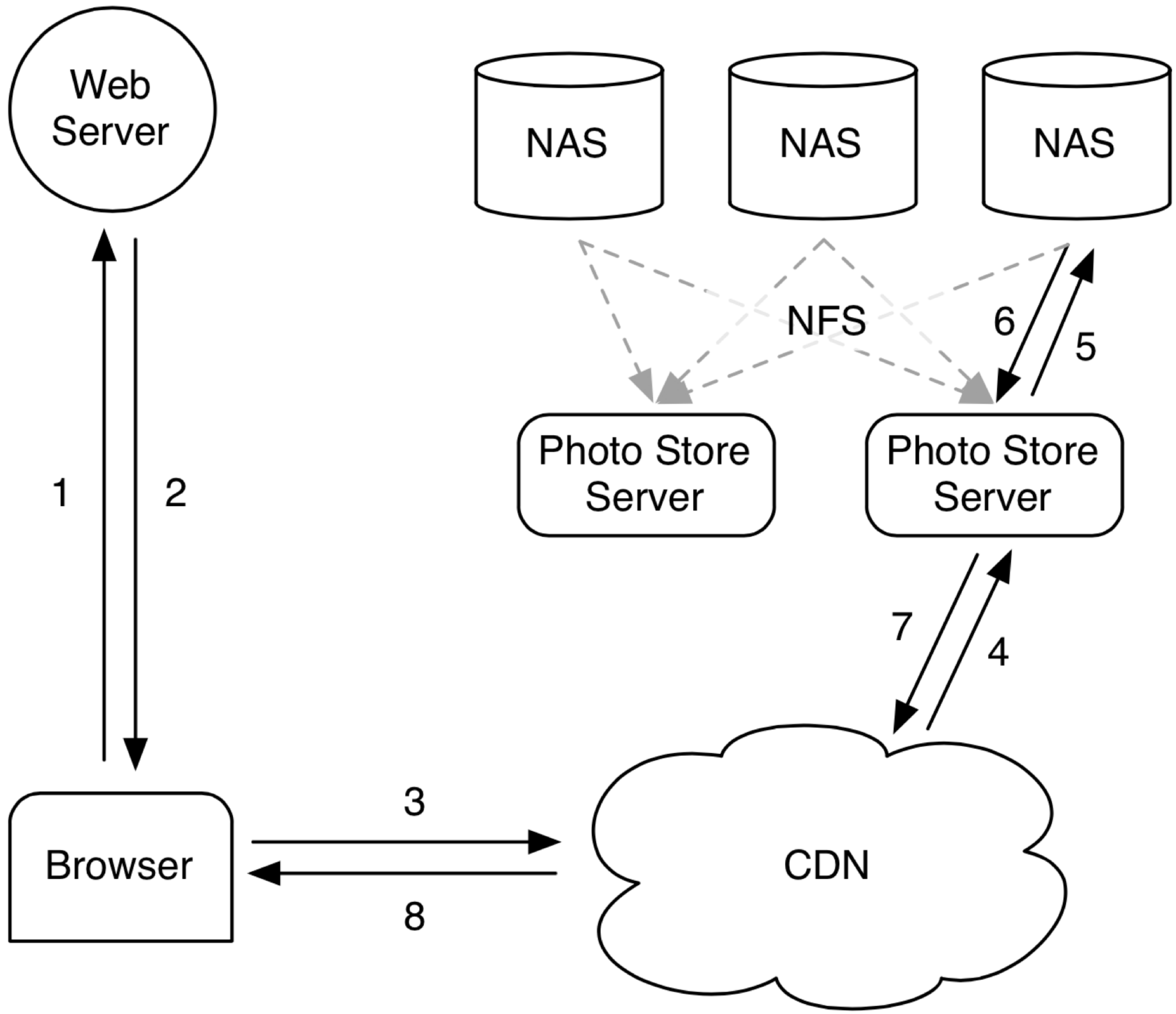
Popularity Comes with Age



Facebook Photo Storage

- Three generations of photo storage
  - NFS-based
  - Haystack: Very warm photos
  - f4: Warm photos
- Characteristics
  - After-CDN storage
  - Each generation solves a particular problem observed from the previous generation.

1st Generation: NFS-Based



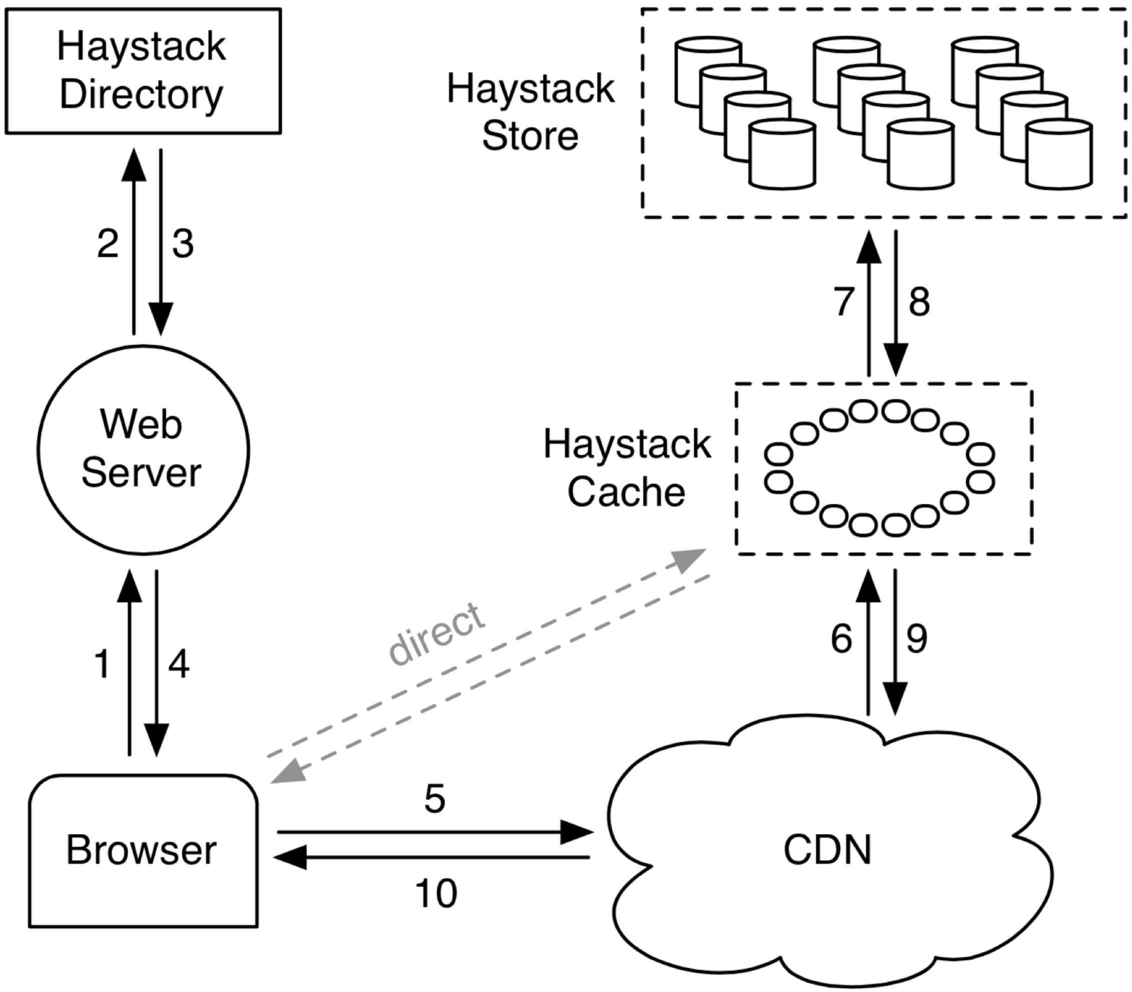
1st Generation: NFS-Based

- Each photo → single file
- Observed problem
  - Thousands of files in each directory
  - Extremely inefficient due to meta data management
  - 10 disk operations for a single image: chained filesystem inode reads for its directory and itself & the file read

2nd Generation: Haystack

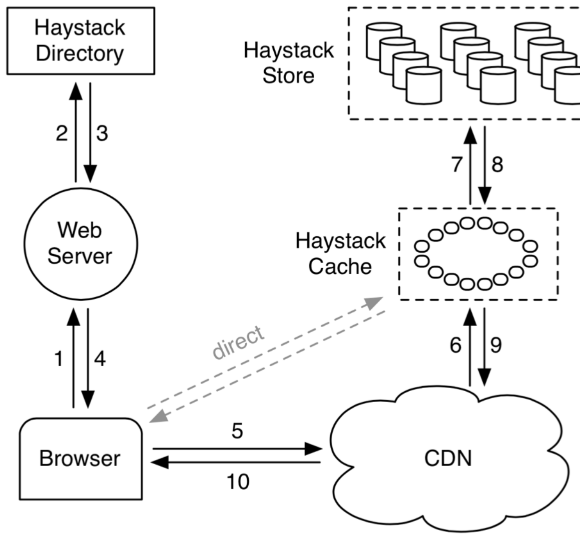
- Custom-designed photo storage
- What would you try? (Hint: too many files!)
  - Starting point: One big file with many photos
- Reduces the number of disk operations required to one
  - All meta data management done in memory
- Design focus
  - Simplicity
  - Something buildable within a few months
- Three components
  - Directory
  - Cache
  - Store

Haystack Architecture



Haystack Directory

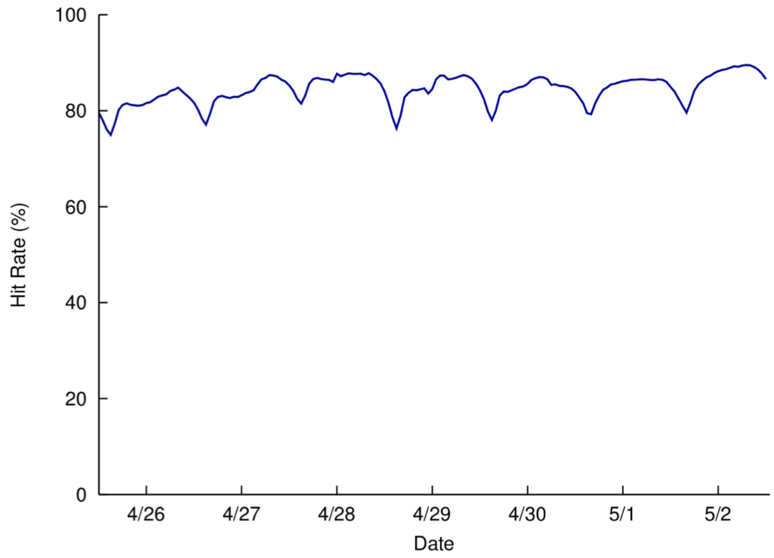
- Helps the URL construction for an image
  - http://⟨CDN⟩/⟨Cache⟩/⟨Machine id⟩/⟨Logical volume, Photo⟩
  - Staged lookup
  - CDN strips out its portion.
  - Cache strips out its portion.
  - Machine strips out its portion



- Logical & physical volumes
  - A logical volume is replicated as multiple physical volumes
  - Physical volumes are stored.
  - Each volume contains multiple photos.
  - Directory maintains this mapping

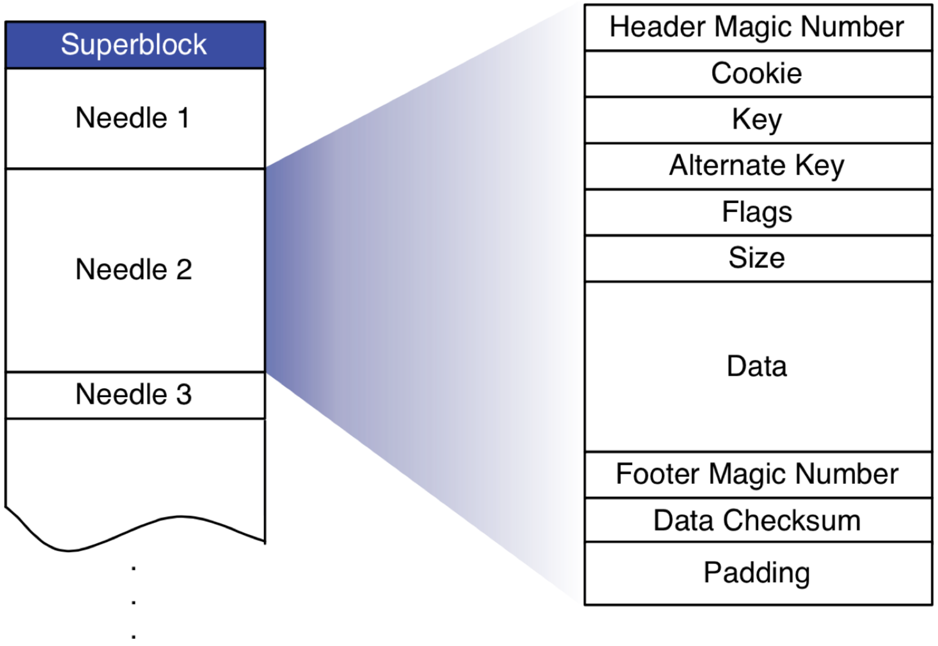
Haystack Cache

- Facebook-operated CDN using distributed hash table (DHT)
  - Photo IDs as the key
- Further removes traffic to Store
  - Mainly caches newly-uploaded photos
- High cache hit rate (due to caching new photos)



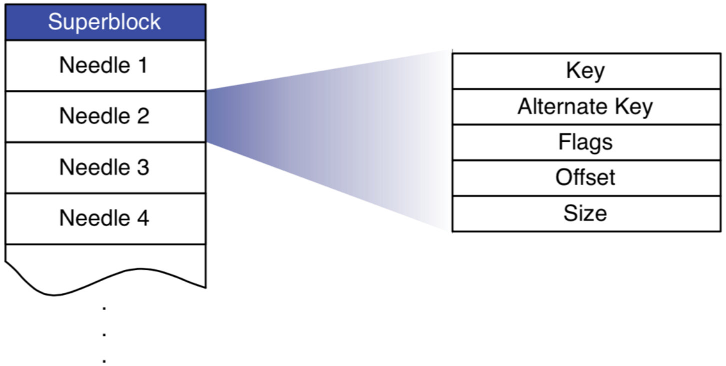
Haystack Store

- Maintains physical volumes
- One volume is a single large file (100GB) with many photos (needles)



Haystack Store

- Metadata managed in memory
  - (key, alternate key) to (flags, size, volume offset)
  - Quick lookup for both read and write
  - Disk operation only required for actual image read
- Write/delete
  - Append-only
  - Delete is marked, later garbage-collected.
- Indexing
  - For fast memory metadata construction



Daily Stats with Haystack

- Photos uploaded: ~120 Million
- Haystack photos written: ~1.44 Billion
  - each photo in 4 sizes and saves each size in 3 different locations.
- Photos viewed: 80 – 100 Billion
  - Thumbnails: 10.2%
  - Small: 84.4%
  - Medium: 0.2%
  - Large: 5.2%
- Haystack photos read: 10 Billion

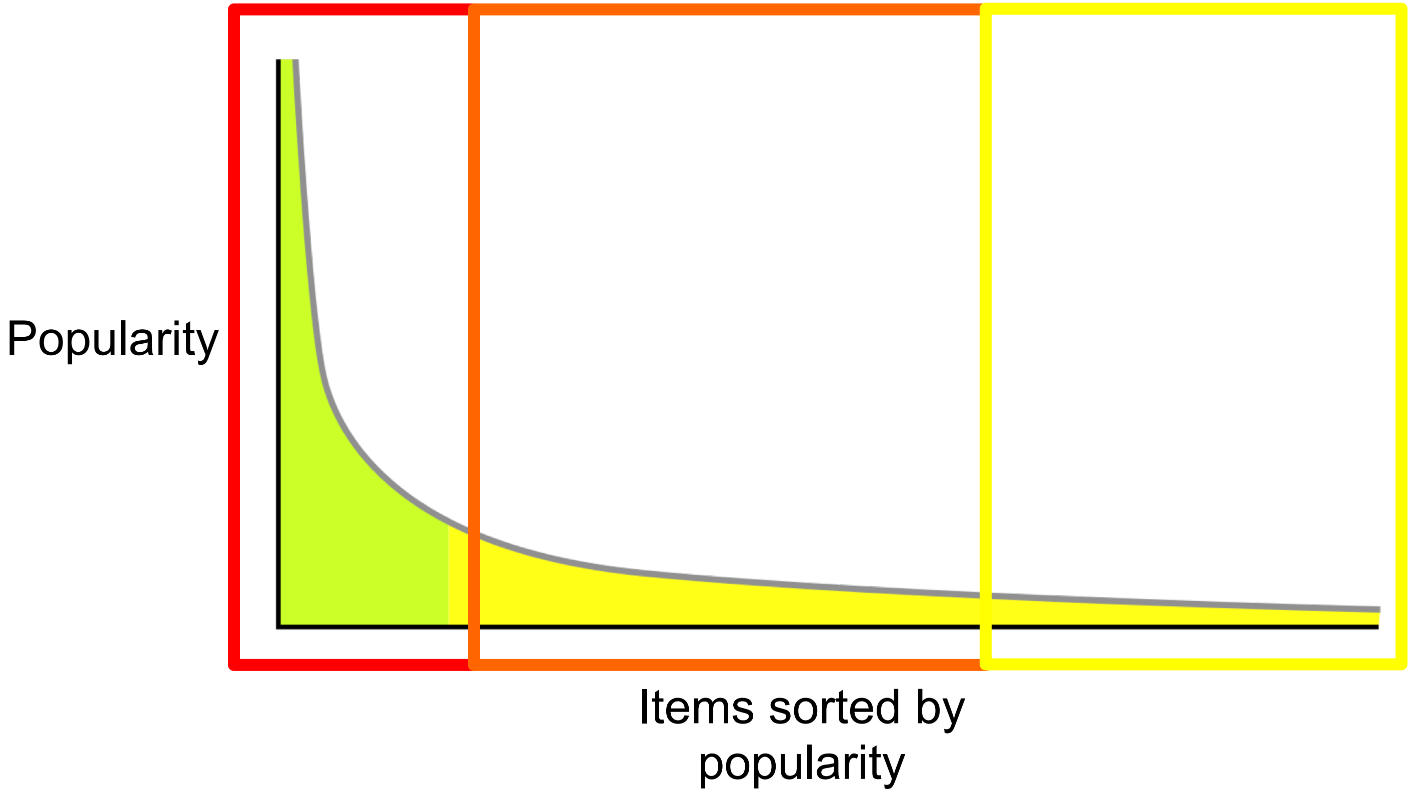
Summary

- Photo workload
  - Zipf distribution
  - “Hot” photos can be handled by CDN
  - “Warm” photos have diminishing returns.
- Haystack: Facebook’s 2nd generation photo storage
  - Goal: reducing disk I/O for warm photos
  - One large file with many photos
  - Metadata stored in memory
  - Internal CDN

Facebook f4

f4: Breaking Down Even Further

- Hot photos: CDN
- Very warm photos: Haystack
- Warm photos: f4
- Why? Storage efficiency

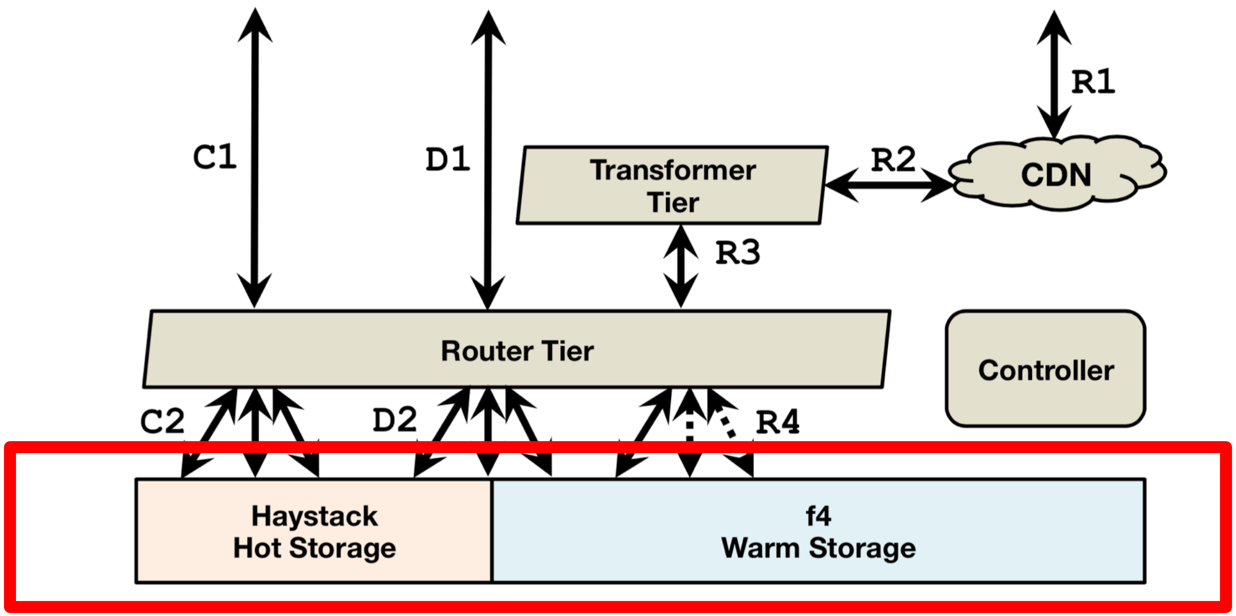


CDN / Haystack / f4

- Storage efficiency became important.
  - Static contents (photos & videos) grew quickly.
- Haystack is concerned about throughput, but not efficiently using storage space.
- Warm photos: Don’t quite need a lot of throughput.
- Design question: Can we design a system that is more optimized for storage efficiency for warm photos?

CDN / Haystack / f4

- CDN absorbs much traffic for hot photos/videos.
- Haystack's tradeoff: good throughput, but somewhat inefficient storage space usage.
- f4's tradeoff: less throughput, but more storage efficient.
    - ~ 1 month after upload, photos/videos are moved to f4.



Why Not Just Use Haystack?

- Haystack
  - Haystack store maintains large files (many photos in one file).
  - Each file is replicated 3 times, two in a single data center, and one additional in a different data center.
- Each file is placed in RAID disks.
  - RAID: Redundant Array of Inexpensive Disks
  - RAID provides better throughput with good reliability.
  - Haystack uses RAID-6, where each file block requires 1.2X space usage.
  - With 3 replications, each file block spends 3.6X space usage to tolerate 4 disk failures in a datacenter as well as 1 datacenter failure.
- f4 reduces this to 2.1X space usage with the same fault-tolerance guarantee.

Haystack & f4

- Haystack uses RAID-6, which has 2 parity bits, with 12 disks.
  - Stripe: 10 data disks, 2 parity disks, failures tolerated: 2
  - Each data block is replicated twice in a single datacenter, and one additional is placed in a different datacenter.
- Storage usage
  - Single block storage usage: 1.2X
  - 3 replications: 3.6X
- How to improve upon this storage usage?
  - RAID parity disks are basically using error-correcting codes
  - Other (potentially more efficient) error-correcting codes exist, e.g., Hamming codes, Reed-Solomon codes, etc.
  - f4 does not use RAID, rather handles individual disks.
  - f4 uses more efficient Reed-Solomon code.

Haystack & f4

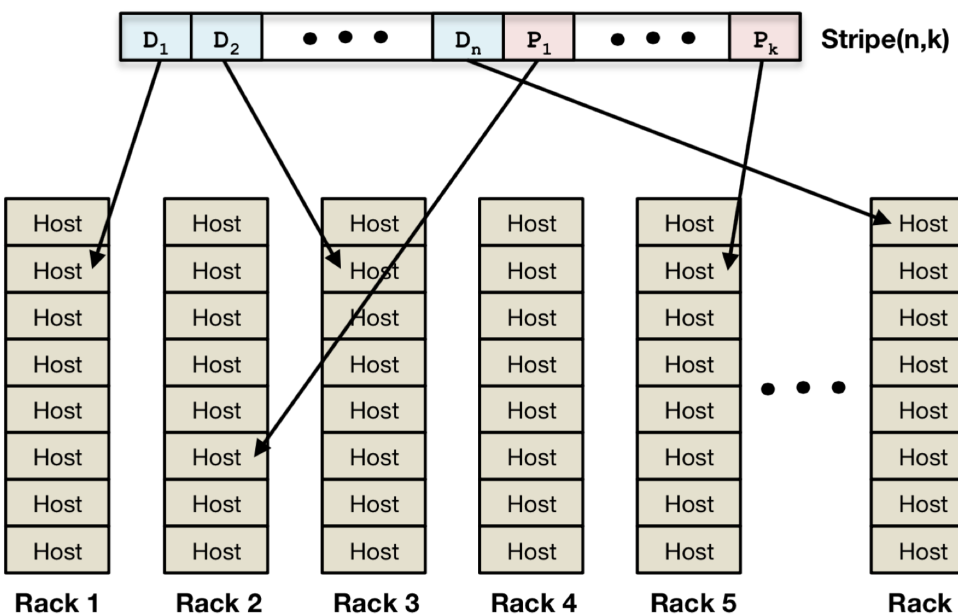
- (n, k) Reed-Solomon code
  - k data blocks, f==(n-k) parity blocks, n total blocks
  - Can tolerate up to f block failures
  - Need to go through coder/decoder for read/write, which affects the throughput
  - Upon a failure, any k blocks can reconstruct the lost block.

k data blocks	*f* parity blocks

- f4 reliability with a Reed-Solomon code
  - Disk failure/host failure
  - Rack failure
  - Datacenter failure
  - Spread blocks across racks and across data centers

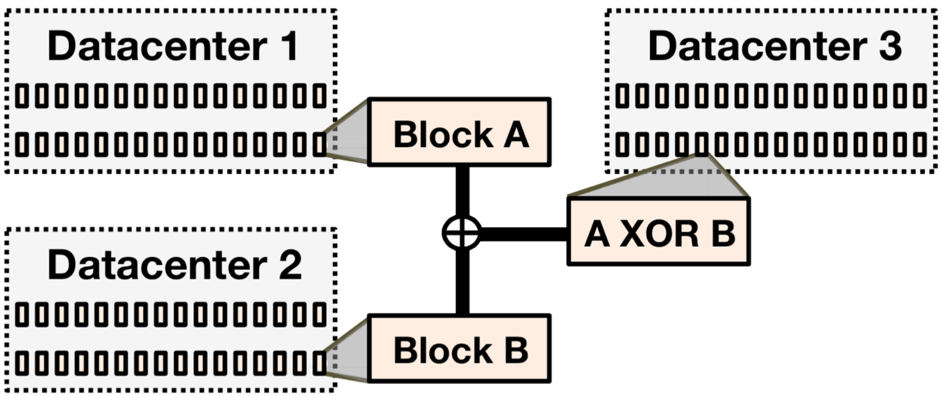
f4: Single Datacenter

- Within a single data center, (14, 10) Reed-Solomon code
  - This tolerates up to 4 block failures
  - 1.4X storage usage per block
- Distribute blocks across different racks
  - This tolerates four host/rack failures



f4: Cross-Datacenter

- Additional parity block
  - Can tolerate a single datacenter failure



- Average space usage per block: 2.1X
  - E.g., average for block A & B: (1.4*2 + 1.4)/2 = 2.1
- With 2.1X space usage,
  - 4 host/rack failures tolerated
  - 1 datacenter failure tolerated

Haystack vs. f4

- Haystack
  - Per stripe: 10 data disks, 2 parity disks, 2 failures tolerated
  - Replication degree within a datacenter: 2
  - 4 total disk failures tolerated within a datacenter
  - One additional copy in another datacenter (for tolerating one datacenter failure)
  - Storage usage: 3.6X (1.2X for each copy)
- f4
  - Per stripe: 10 data disks, 4 parity disks, 4 failures tolerated
  - Reed-Solomon code achieves replication within a datacenter
  - One additional copy XOR'ed to another datacenter, tolerating one datacenter failure
  - Storage usage: 2.1X (previous slide)

Summary

- Facebook photo storage
  - CDN
  - Haystack
  - f4
- Haystack
  - RAID-6 with 3.6X space usage
- f4
  - Reed-Solomon code
  - Block distribution across racks and datacenters
  - 2.1X space usage

Reading

- Finding a needle in Haystack: Facebook's photo storage
  - https://www.usenix.net/legacy/events/osdi10/tech/full_papers/Beaver.pdf
- f4: Facebook's Warm BLOB Storage System
  - https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-muralidhar.pdf

Acknowledgement

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC), Michael Freedman (Princeton), Jennifer Rexford (Princeton), and Steve Ko (Buffalo).