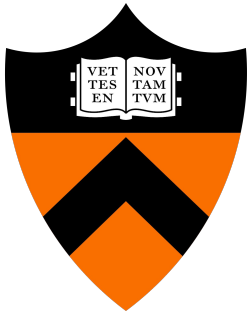
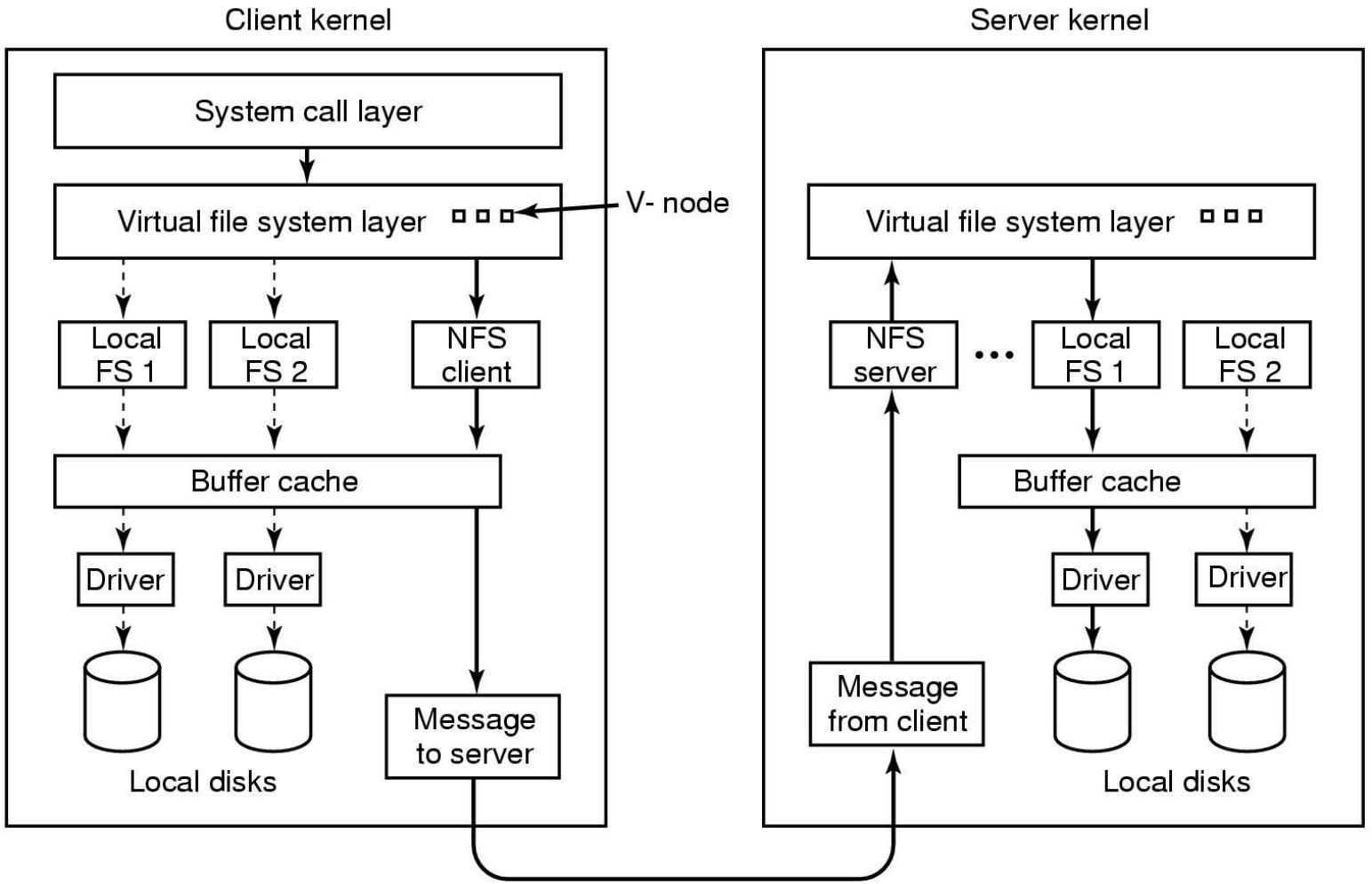
Caching:
Network Files Systems + the Web



COS 316

Michael Freedman

Using a network file system



3 Goals: Make remote operations appear

- Local
- Consistent
- Fast

Good performance? Caching to rescue!

- Distributed file systems
  - Make a remote file system look local
  - Today: NFS (Network File System)

- Web servers
  - Make remote content look local

TANSTANFL

(There ain't no such thing as a free lunch)

- With local FS, `read` sees data from "most recent" `write`, even if performed by different process
  - "Read/write coherence", linearizability
- Achieve the same with NFS?
  - Perform all reads & writes synchronously to server
  - Huge cost: high latency, low scalability
- And what if the server doesn't return?
  - Options: hang indefinitely, return ERROR

Caching GOOD

Lower latency, better scalability

Consistency HARDER

No longer one single copy of data, to which all operations are serialized

Caching options

- Read-ahead: Pre-fetch blocks before needed
- Write-through: All writes sent to server
- Write-behind: Writes locally buffered, send as batch

- Consistency challenges:
  - When client writes, how do others caching data get updated? (Callbacks, …)
  - Two clients concurrently write? (Locking, overwrite, …)

NFS

- Stateless protocol
  - Recovery easy: crashed == slow server
  - Messages over UDP (unencrypted)
- Read from server, caching in NFS client
- NFSv2 was write-through (i.e., synchronous)
- NFSv3 added write-behind
  - Delay writes until `close` or `fsync` from application

Exploring the consistency tradeoffs

- Write-to-read semantics too expensive
  - Give up caching, require server-side state, or …
- Close-to-open “session” semantics
  - Ensure an ordering, but only between application `close` and `open`, not all `writes` and `reads`.
  - If B opens after A closes, will see A’s writes
  - But if two clients open at same time? No guarantees
    - And what gets written? “Last writer wins”

NFS Cache Consistency

- Recall challenge: Potential concurrent writers
- Cache validation:
  - Get file's last modification time from server: `getattr(fh)`
  - Both when first open file, then poll every 3-60 seconds
    - If server's last modification time has changed, flush dirty blocks and invalidate cache
- When reading a block
  - Validate: (current time – last validation time < threshold)
  - If valid, serve from cache. Otherwise, refresh from server

Some problems…

- "Mixed reads" across version
  - A reads block 1-10 from file, B replaces blocks 1-20, A then keeps reading blocks 11-20.
- Assumes synchronized clocks. Not really correct.
- Writes specified by offset
  - Concurrent writes can change offset

When statefulness helps

Leases

- Client obtains ***lease*** on file for read or write
  - "A lease is a ticket permitting an activity; the lease is valid until some expiration time."

- Read lease allows client to cache clean data
  - *Guarantee:* no other client is modifying file

- Write lease allows safe delayed writes
  - Client can locally modify than batch writes to server
  - *Guarantee:* no other client has file cached

Using leases

- Client requests a lease
  - May be implicit, distinct from file locking
  - Issued lease has file version number for cache coherence
- Server determines if lease can be granted
  - *Read leases* may be granted concurrently
  - *Write leases* are granted exclusively
- If conflict exists, server may send *eviction* notices
  - Evicted write lease must write back
  - Evicted read leases must flush/disable caching
  - Client acknowledges when completed

Bounded lease term simplifies recovery

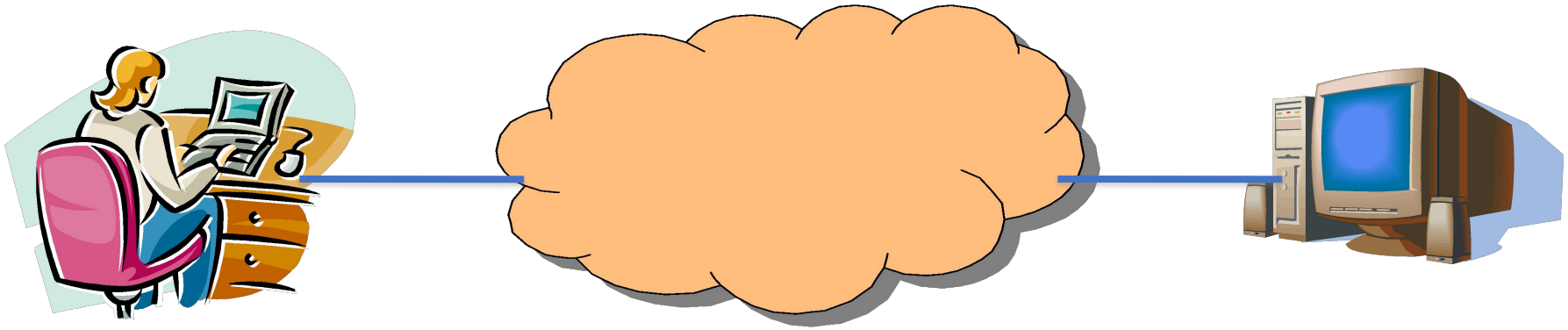
- Before lease expires, client must *renew* lease

- Client fails while holding a lease?
  - Server waits until the lease expires, then unilaterally reclaims
  - If client fails during eviction, server waits then reclaims

- Server fails while leases outstanding? On recovery:
  - Wait *lease period* + *clock skew* before issuing new leases
  - Absorb renewal requests and/or writes for evicted leases

Statelessness: Web caching

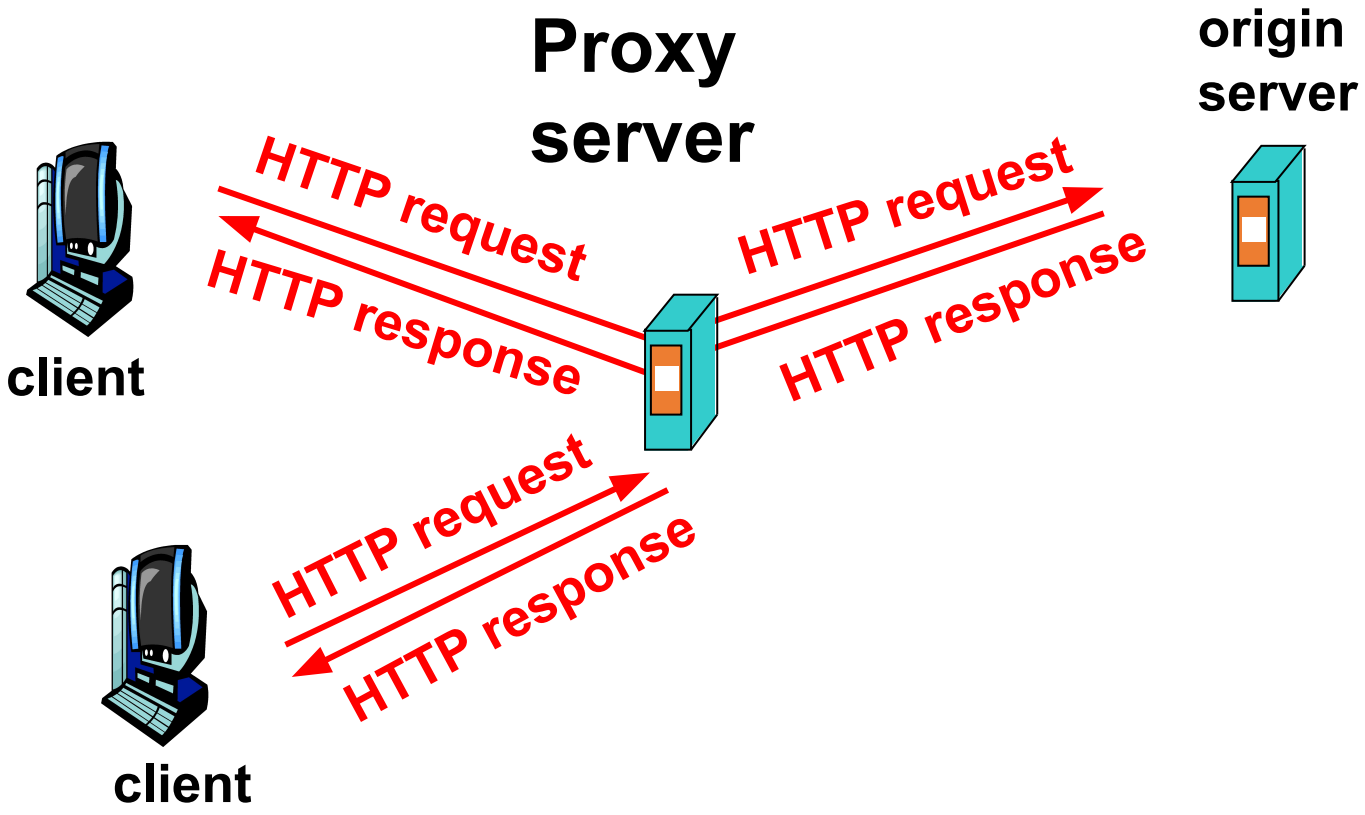
Single Server, Poor Performance



- Single server
  - Single point of failure
  - Easily overloaded
  - Far from most clients
- Popular content
  - Popular site
  - Flash crowd
  - Denial of Service attack

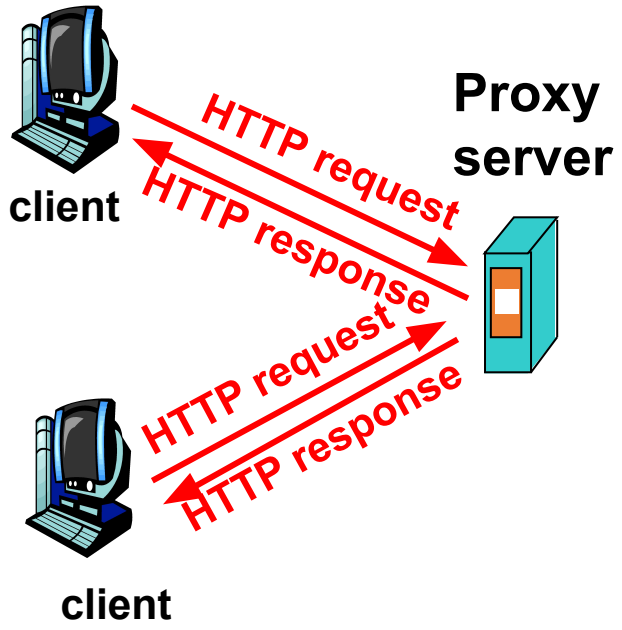
Proxy Caches

- Accept requests from multiple clients
- Takes request and reissues it to server
- Takes response and forwards to client



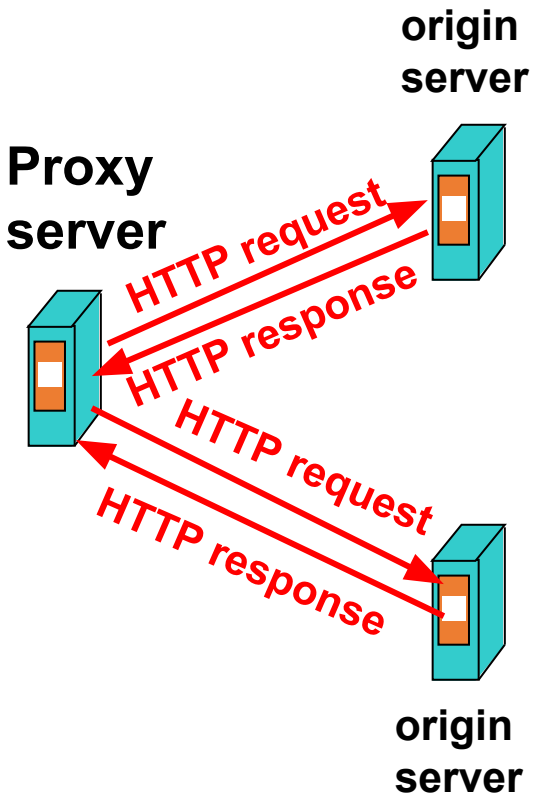
Forward Proxy

- Cache “close” to the client
  - Under administrative control of client-side AS
- Explicit proxy
  - Requires configuring browser
- Implicit proxy
  - Service provider deploys an “on path” proxy
  - … that intercepts and handles Web requests



Reverse Proxy

- Cache “close” to server
  - Either by proxy run by server or in third-party content distribution network (CDN)
- Directing clients to the proxy
  - Map the site name to the IP address of the proxy



Limitations of Web Caching

- Much content is not cacheable
  - Dynamic data: stock prices, scores, web cams
  - CGI scripts: results depend on parameters
  - Cookies: results may depend on passed data
  - SSL: encrypted data is not cacheable
  - Analytics: owner wants to measure hits
- Stale data
  - Or, overhead of refreshing the cached data

Modern HTTP Video-on-Demand

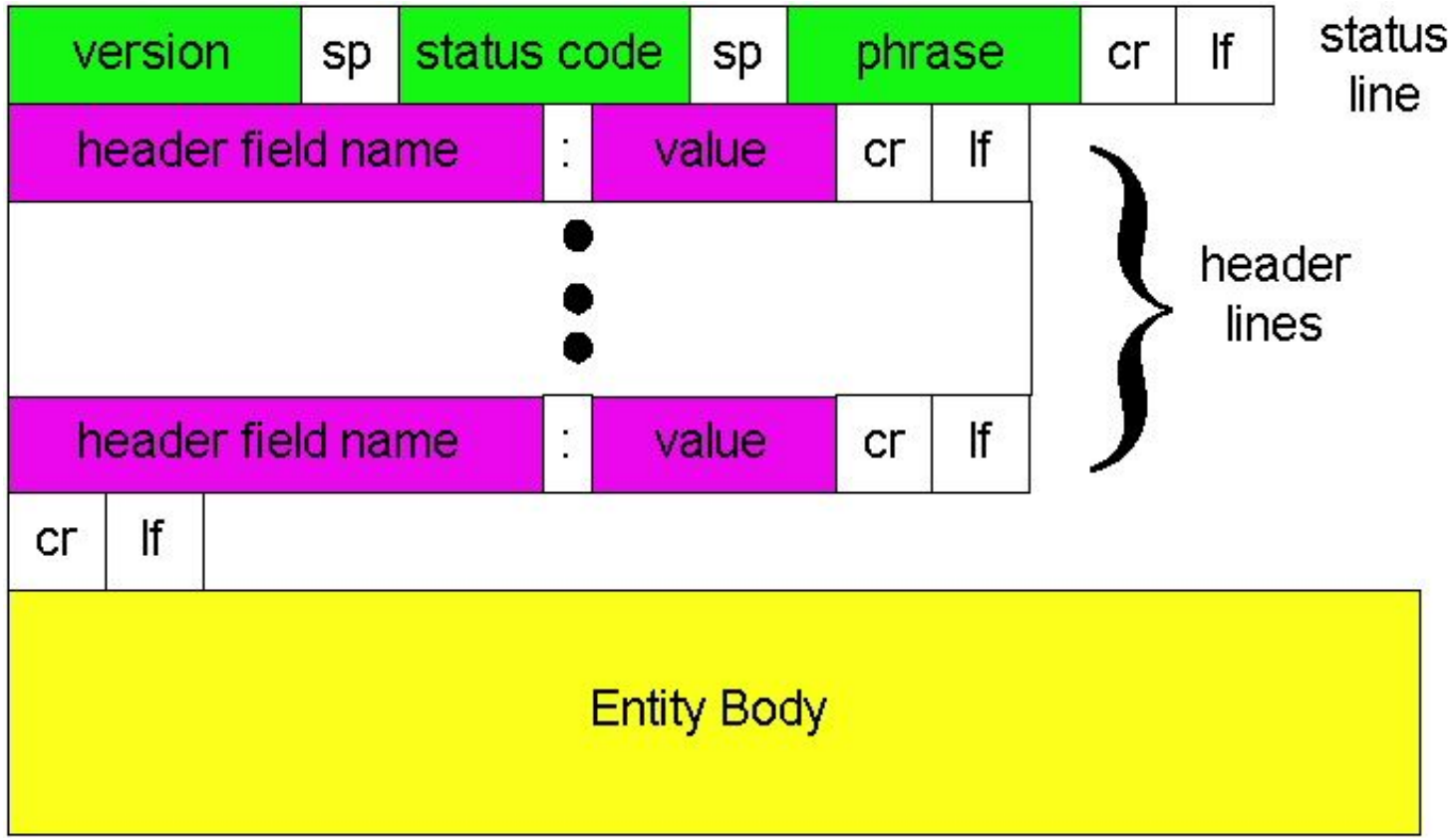
- Download “content manifest” from origin server
- List of video segments belonging to video
  - Each segment 1-2 seconds in length
  - Client can know time offset associated with each
  - Standard naming for different video resolutions: 320dpi, 720dpi, 1040dpi
- Client downloads segment (at certain res) using standard HTTP
  - HTTP request can be satisfied by cache: it’s a static object
- Client observes download time vs. segment duration, increases/decreases resolution if appropriate

HTTP Caching

- Clients (and proxies) cache documents
  - When should origin be checked for changes?
  - Every time? Every session? Date?

- HTTP includes caching information in headers
  - HTTP 0.9/1.0 used: “Expires: <date>”; “Pragma: no-cache”
  - HTTP/1.1 has “Cache-Control”
    - “No-Cache”, “Max-age: <seconds>”
    - “E-tag: <opaque value>

HTTP Response includes headers



HTTP Caching

- If not expired: use cached copy
- If expired, use condition GET request to origin
  - “If-Modified-Since: <date>”, “If-None-Match: <etag>”
  - 304 (“Not Modified”) or 200 (“OK”) response

GET / HTTP/1.1

Host: sns.cs.princeton.edu

Connection: Keep-Alive

If-Modified-Since: Tue, 1 Feb 2011 …

HTTP/1.1 304 Not Modified

Date: Wed, 02 Feb 2011 ….

Server: Apache/2.2.3 (CentOS)

Accept-Ranges: bytes

Cache validation in many-server world

- What happens in many servers and basing cache validation on "modification time"?
- Enter stronger validators based on content, not time

GET / HTTP/1.1

Host: sns.cs.princeton.edu

Connection: Keep-Alive

If-Modified-Since: Tue, 1 Feb 2011 …

If-None-Match: "7a11f-10ed-3a75ae4a"

HTTP/1.1 304 Not Modified

Date: Wed, 02 Feb 2011 ….

Server: Apache/2.2.3 (CentOS)

Accept-Ranges: bytes

ETag: "7a11f-10ed-3a75ae4a"

Caching GOOD

Lower latency, better scalability

Consistency HARDER

No longer one single copy of data, to which all operations are serialized