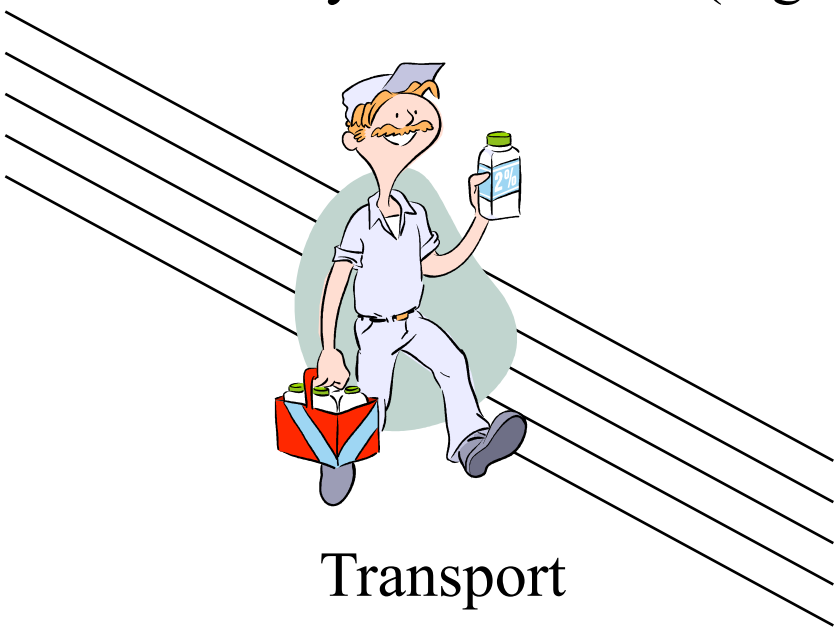
Caching Techniques for Web Content

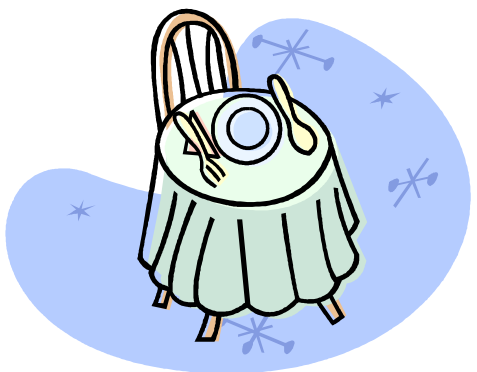
Got Milk?

Server

Annoyed cow (high server load),
Exhausted milk man (high network load),
Delayed breakfast (high access latency).



Transport



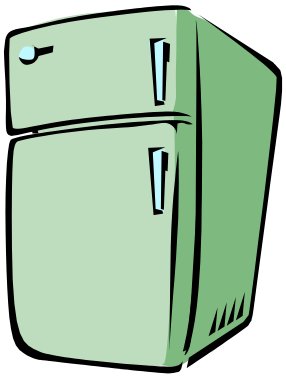
Client

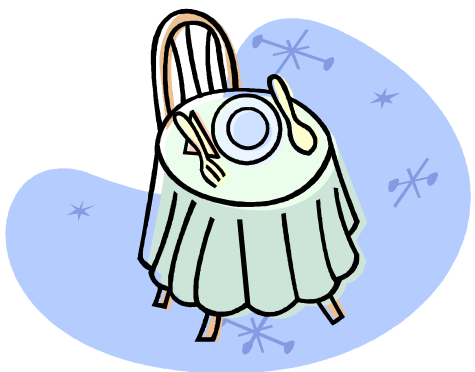
Get Milk The Better Way

Server

Transport

Cache

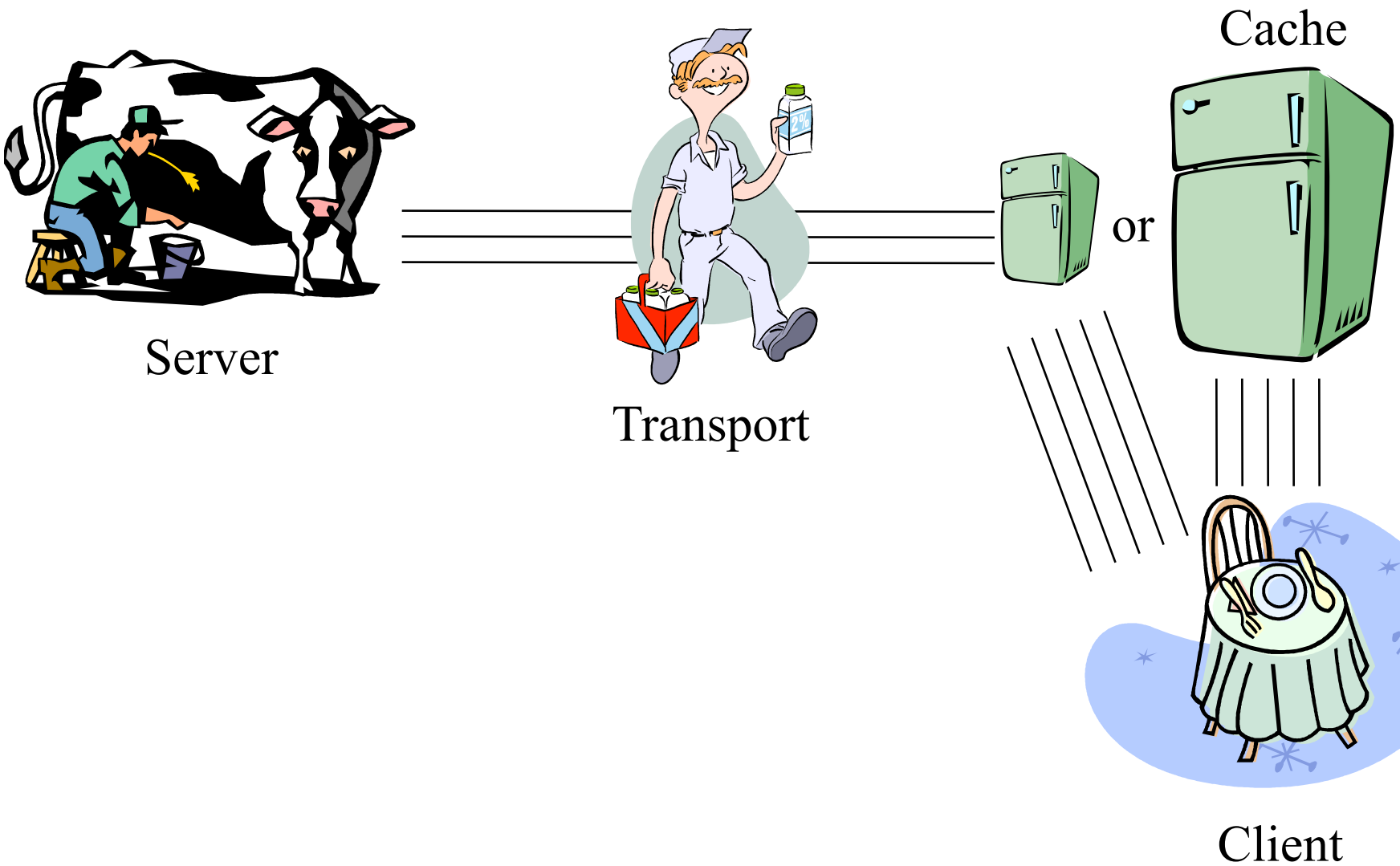




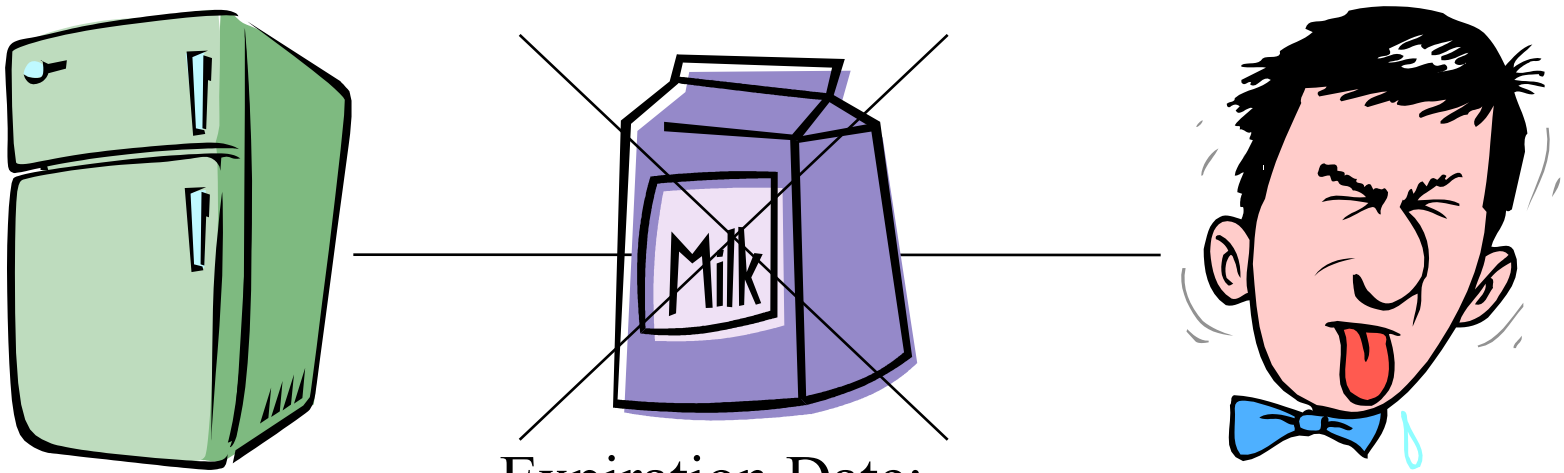
Client

Closer is better!

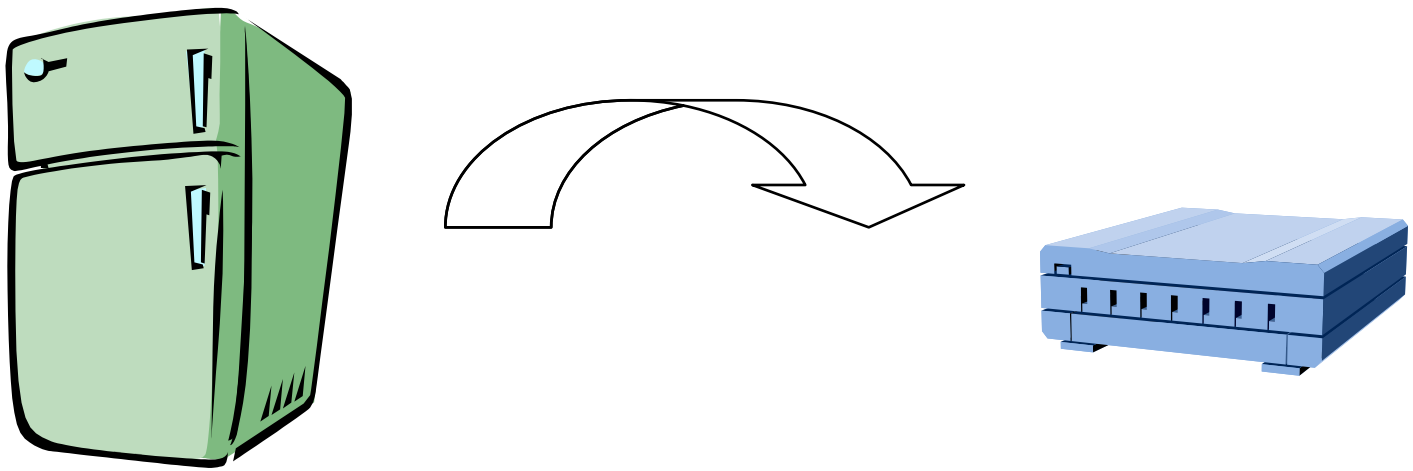
But How Big A Refrigerator?



But What About Sour Milk?

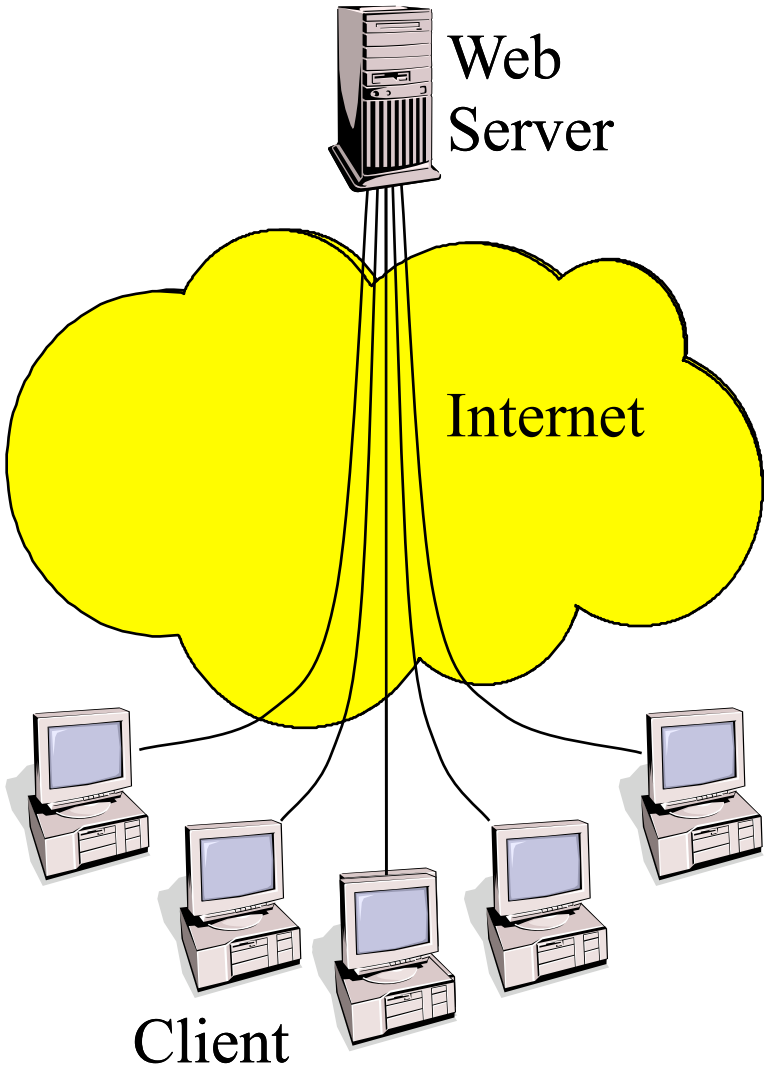


From Cows And Milk To Web Caching



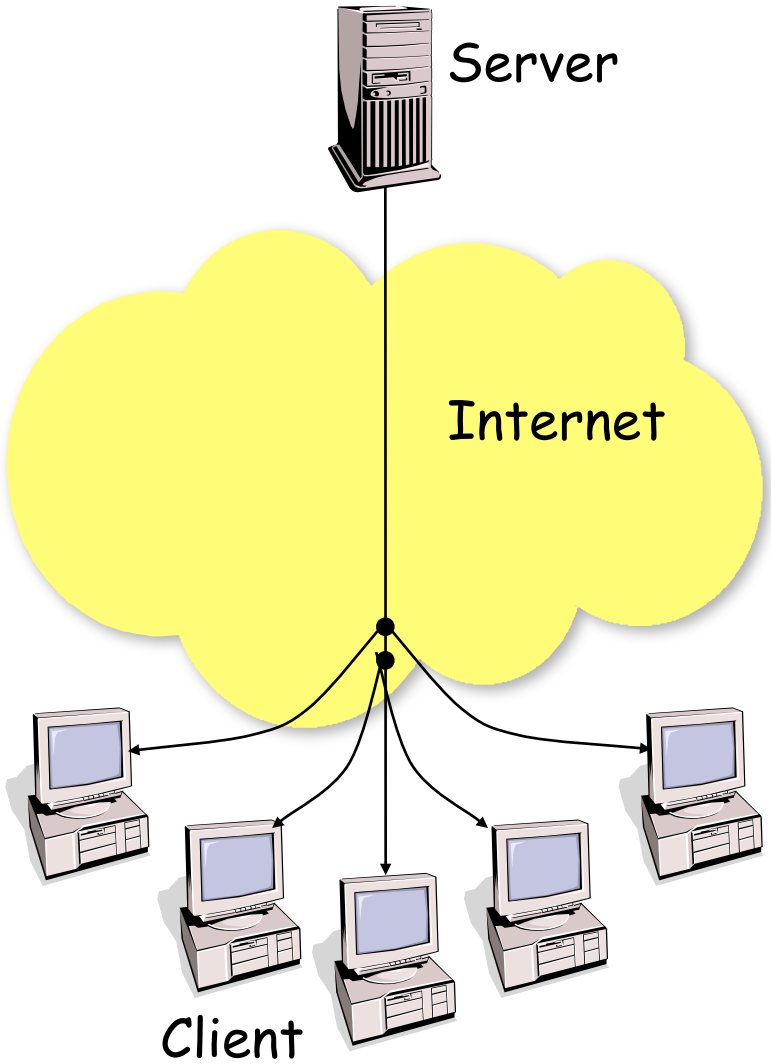
Client/Server Architecture

- The Web is based on a Client/Server architecture.
  - Client sends request to Web server,
  - Web server sends requested data back to the client.
- Data transfer is done using unicast transmission.
- Problems:
  - High server load,
  - High network load,
  - High access latency.



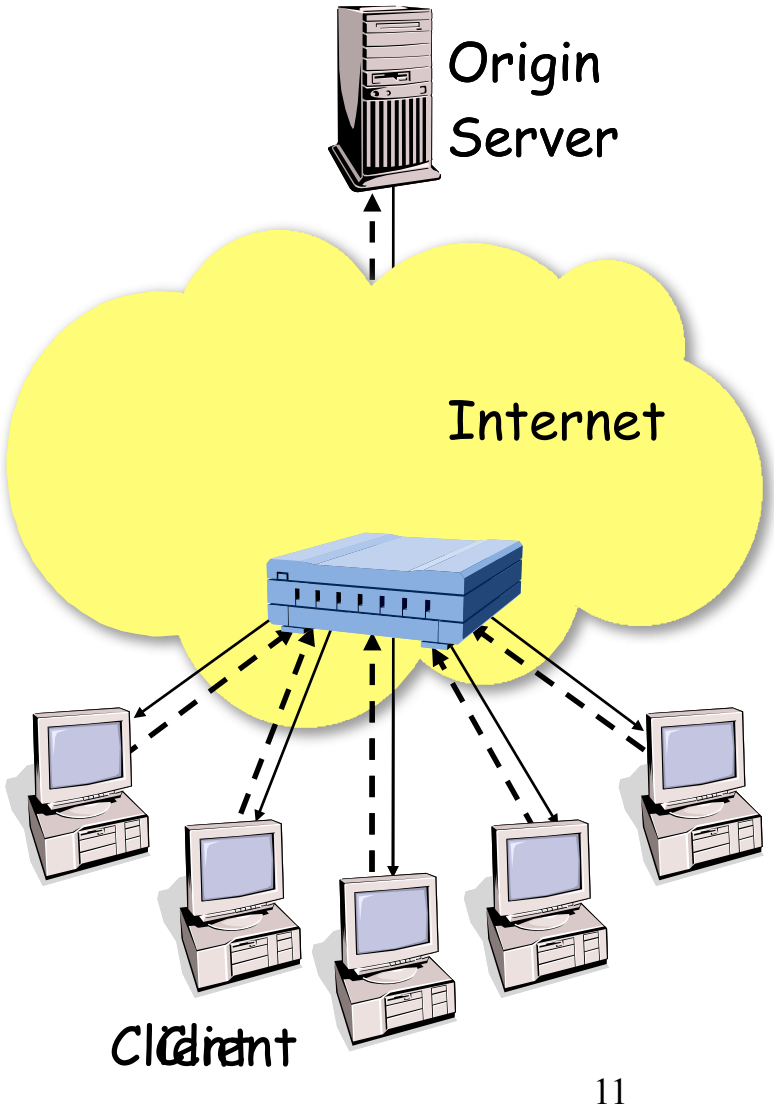
Solution Multicast?

- Benefits:
  - Reduced server load,
  - Reduced network load.
- Problems:
  - Multicast requires synchronous receiver (i.e. receivers request same data at the same time),
  - Multicast not yet widely deployed,
  - Multicast does not address the problem of high access latency.



Web Caching

- Clients request web pages via the Web cache.
- Web cache stores previously received data to serve future requests for the same data from the local disk.
- Benefits:
  - Reduced network load,
  - Reduced server load,
  - Reduced latency.



The Origins Of (Web) Caching

- Many centuries ago, the French used the word “cacher”, meaning “to hide”.
  - Became the modern word “cache”, meaning “a hiding place used especially for storing provisions”.
- Caching is a widely used technique in computer systems architecture.
  - E.g. processor cache, disk cache, graphic board cache, etc.
- Most Web browsers include a local cache mechanisms that stores Web content on the local disk.
  - Pitfall: Could be responsible for serving stale content.
- First Web server (httpd) had an associated proxy server that included a cache.
  - Built at CERN, early 1990s.
- Caching was already an important aspect of HTTP/1.0.
  - Early experiments in 1994 demonstrated significant benefits of Web caching.

Web Caching types

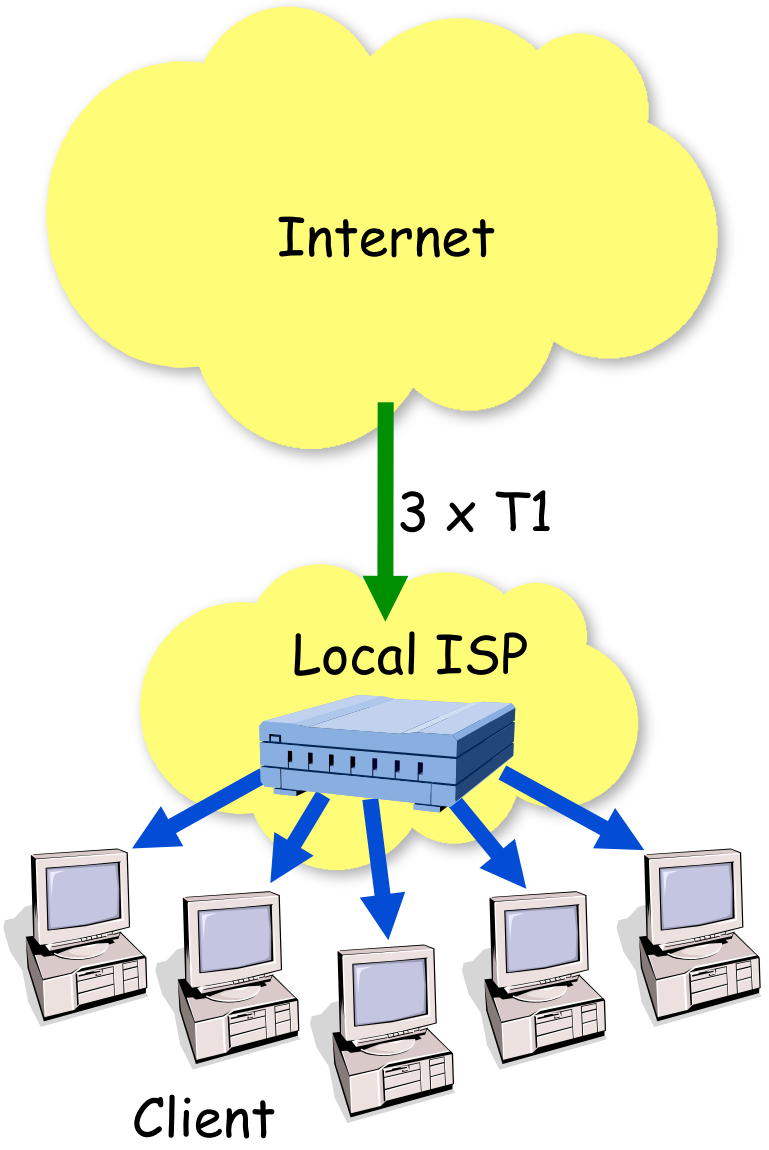
- Caching on the server:
  - Store previously generated content to avoid regenerating identical content for multiple requests
  - Cache pre-compiled code, database queries or anything else that can decrease the time necessary to dynamically create content
- Caching on the client
- Proxy caching
  - Public proxies: shared by multiple clients - located near the servers
  - Private proxies: not shared - located near the clients that they serve

Motivation And Goals Of Web Caching

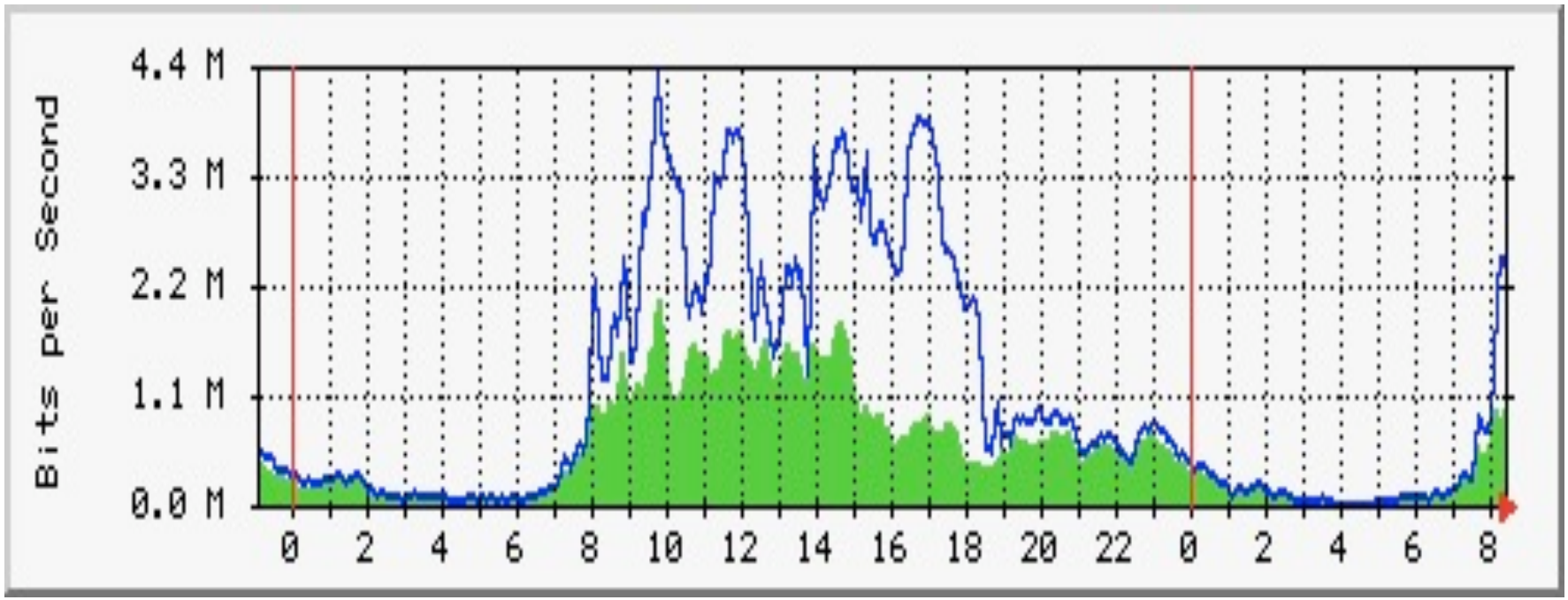
- Goals: Improve user experience and network/server scalability.
  - Speed up user access to Web content,
  - Reduce network load,
  - Reduce origin server load.
- But: In case of a cache miss, a Web cache can slow down access and increase network load.
  - Need to carefully analyze deployment scenario and properly engineer the Web caching system.
- Features to assist with system administration and monitoring are essential.
- Most important: “First, do no harm”.
  - Do not serve stale content,
  - Do not fail.

Web Caching Benefits (1)

- Cache deployed at a local ISP in New Jersey.
- ISP is connected to the Internet via three T1 links => 4.5 Mbps capacity overall.
- Incoming Data Traffic:
  - Data coming from the Internet, over the 3 x T1 connection.
- Outgoing Data Traffic:
  - Data between cache and clients.
- Without a cache: incoming data traffic = outgoing data traffic.



Web Caching Benefits (2)

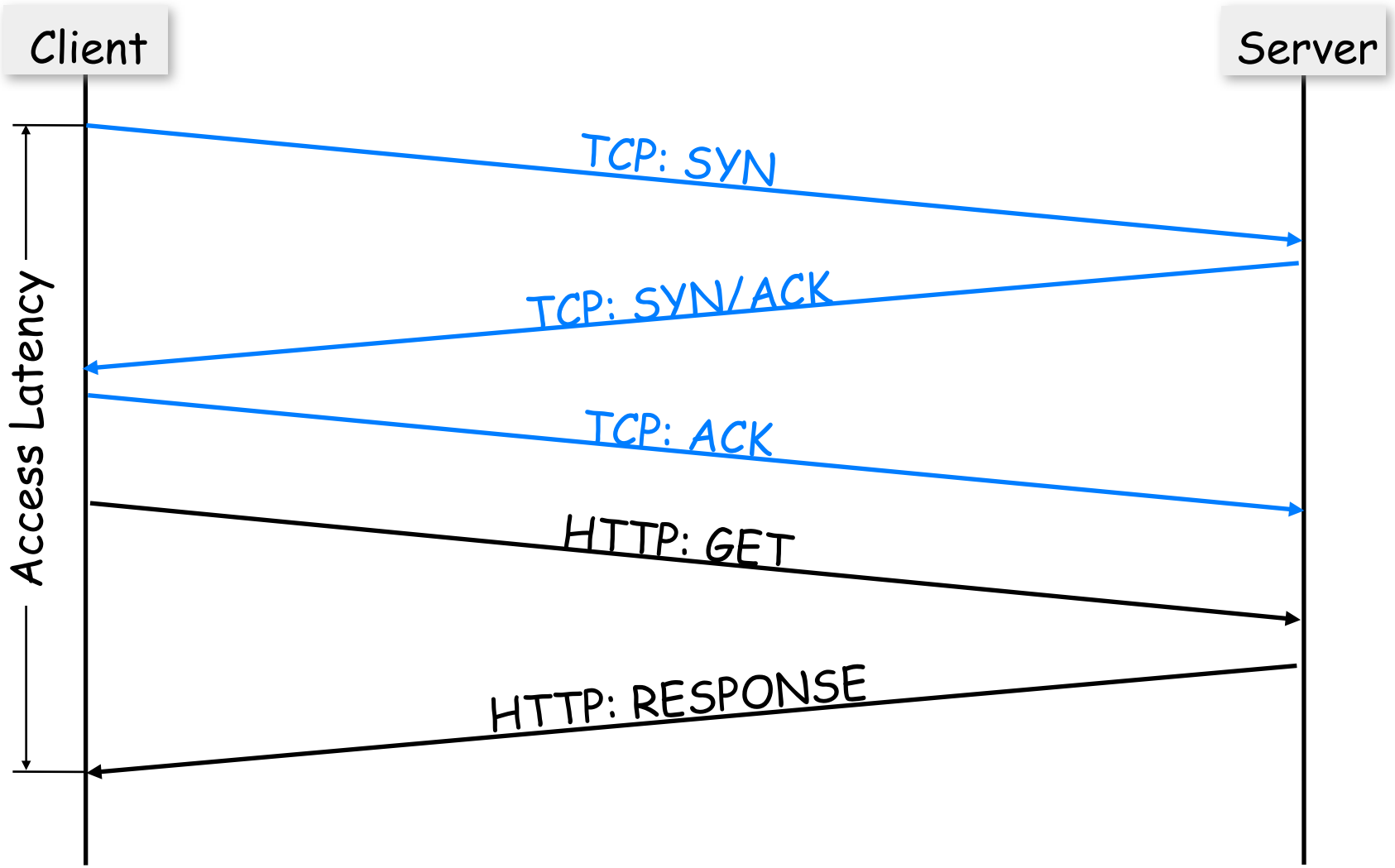


- Without a cache, ISP would have to pay for additional T1 links very soon, while there is enough bandwidth left for future growth after deploying the cache.

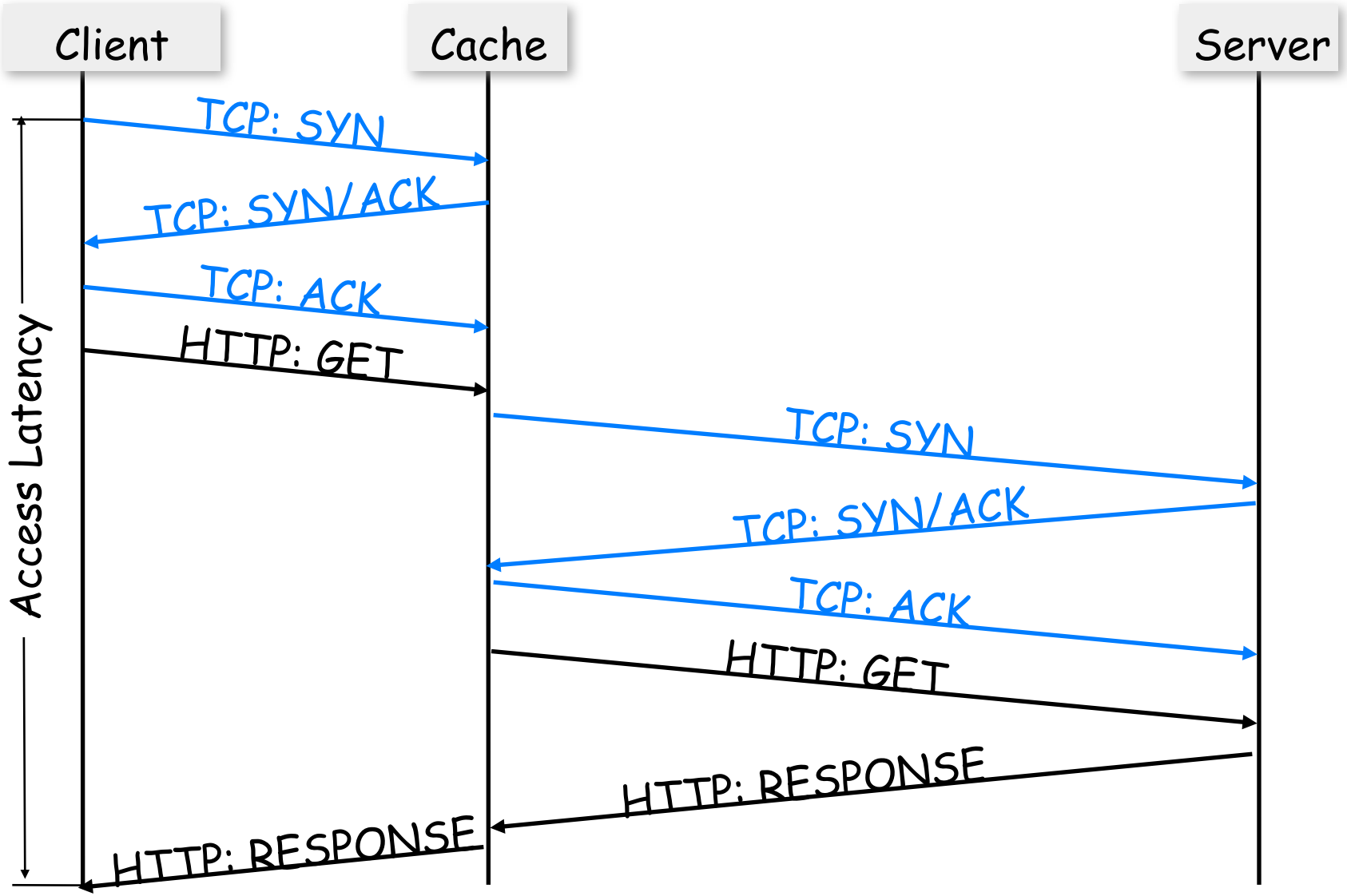
Web Caching Concerns

- User receives a stale response
- User receives personalized content intended for another user
- Sensitive information is cached, exposing it to a greater risk of compromise

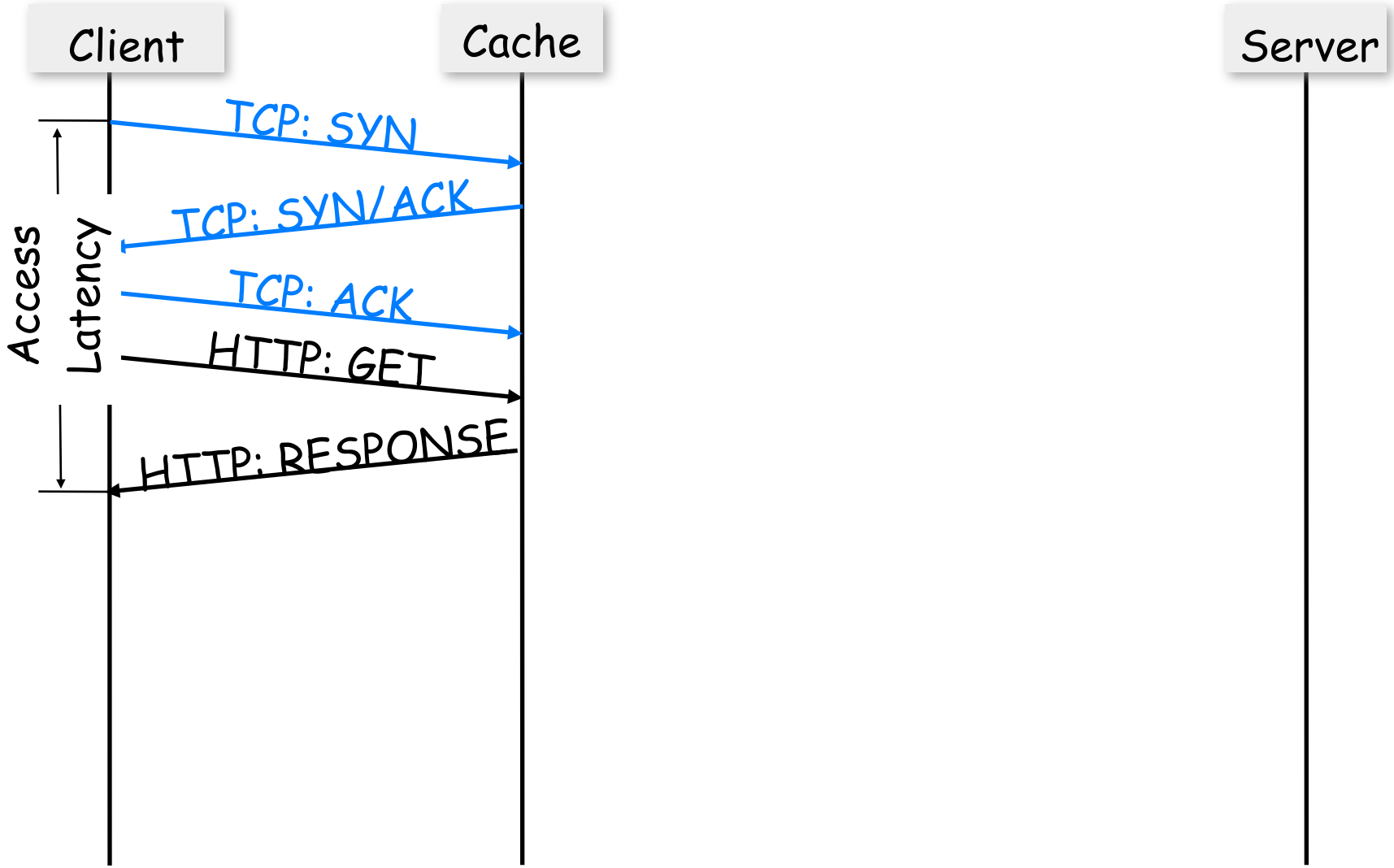
Recap: Client/Server Interaction In HTTP



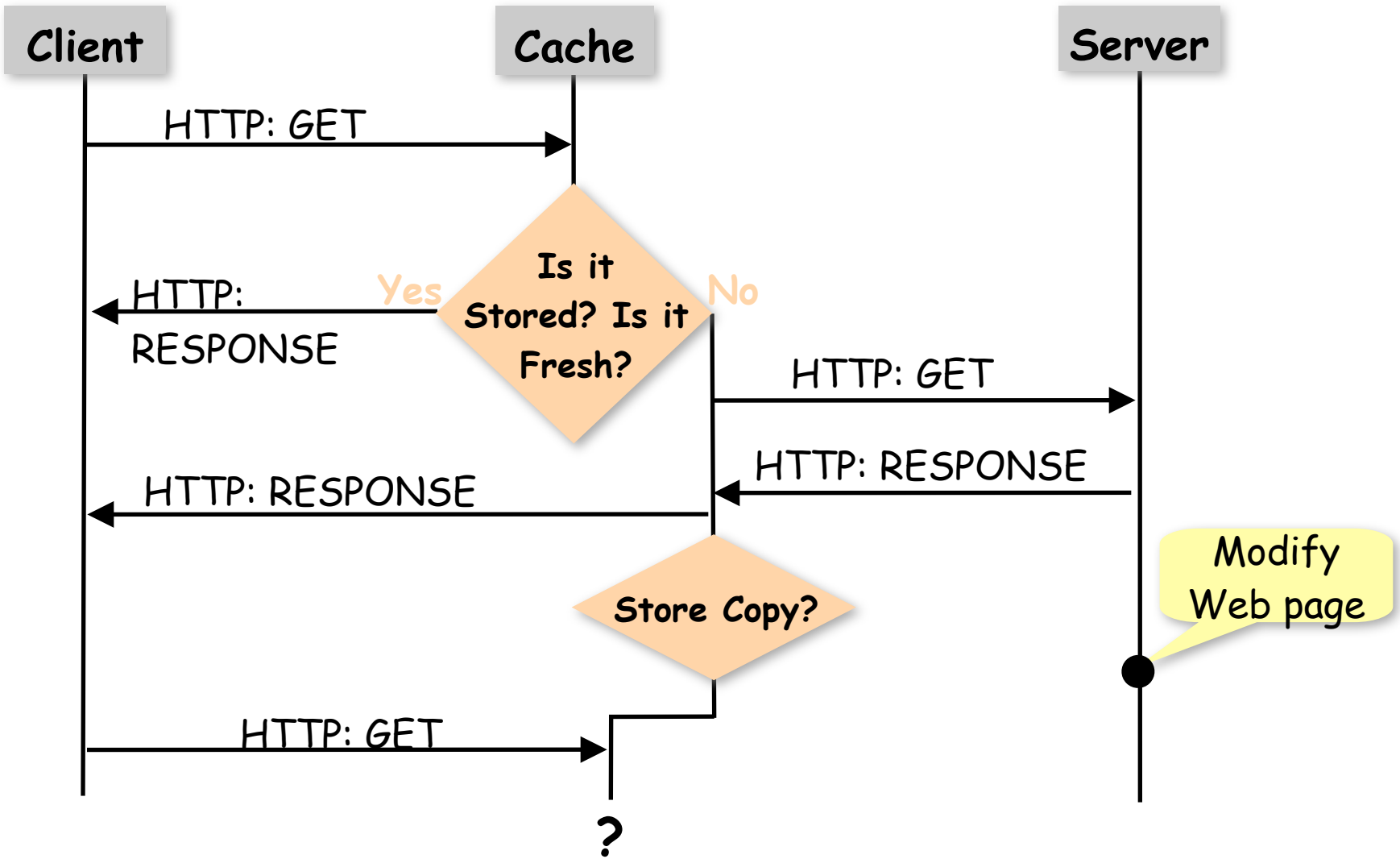
Basic Web Cache Operation: Cache Miss



Basic Web Cache Operation: Cache Hit



Basic Web Cache Operation: Flow Chart



Terminology

- Object: A file (containing HTML text, an image, an applet, an A/V clip) addressable by a single URL.
- Client: The client requests web content (e.g. web browser)
- Origin Server: Web server that is the official host of the object (identified by the host name in the URL).
- Web Cache: An intermediate server that intercepts and handles client requests; a cache acts as both a server and a client.
- Hit Ratio: Fraction of requests to a cache that can be satisfied by the cache (e.g. hit ratio of 35% means that 35% of the requests can be served by the cache without contacting the origin server).
- Dynamic Content: Objects or components of objects that are generated dynamically upon request (e.g. CGI scripts, rotating advertisement banners, etc.).

Caching Challenges

- Dynamic Content
  - Caches might decide to not cache outputs of CGI/PHP/ASP scripts.
- Cookies
  - Client requests including a COOKIE header might indicate non-cacheable content.
- Hit counts
  - Caches can cause hit counts calculations to fail.
- Less-savvy users and privacy-concerned users
  - How do you get a user to point his browser to a cache?
- Access control
  - How to make sure that the content provider gets paid?
  - Legal and security restrictions.

Caching Challenges (contd.)

- Cache replacement
  - When should a Web cache store a Web object?
  - What happens when the Web cache runs out of storage?
- Cache consistency
  - Conditional GET,
  - Expiration time,
  - Consistency heuristics,
  - Cache invalidation,
  - Cache update
  - Invalidation contracts.

Cache Replacement

- A Web cache typically attempts to store a copy of a Web object when it is requested.
  - Object store has only finite capacity.
- Web cache replacement rules determine the action to be taken when the object store is full.
  - Should the Web cache store the new Web object?
  - If yes, which Web object should be removed from the object store to make room for the new one?
- Most simple/popular replacement rules:
  - LRU (replace least recently used page),
  - LFU (replace least frequently used page).
- More sophisticated replacement rules and combinations of multiple rules are possible.

Cache Replacement Considerations

- The probability of the object being accessed in the near future.
  - Can be known a-priori or estimated based on past access patterns.
- The number of accesses to the object in the past.
  - An object that has been accessed frequently in the past is likely to be accessed again.
- The time since the last modification of the object.
  - An object that has not changed for a long time is unlikely to change in the near future.
- Cost of fetching the object (e.g. Web cache distance from Web server).
- Cost of storing the object (e.g. object size).
- Importance of client requesting the object.
- Government and enterprise policies.

Common Cache Replacement Strategies

- Least Recently Used (LRU): Replace the object that has gone without a request for the longest time – cache will fill with the most recently requested objects.
- Least Frequently Used (LFU): Replace the object that has had the fewest requests since it was first stored – cache will fill with the most frequently requested objects.
- Next To Expire (NTE): Replace the Object that is forecast to expire the soonest – cache will fill with the most stable objects.
- Largest File First (LFF): Replace the largest object, freeing up the most space for new objects – cache will fill with smallest objects.
- Hyper-G: Combines LFU, LRU, and LFF.
- GreedyDual: Associates a utility value to Web objects.

Different replacement strategies are often combined and augmented with heuristics to provide the most practical solution.

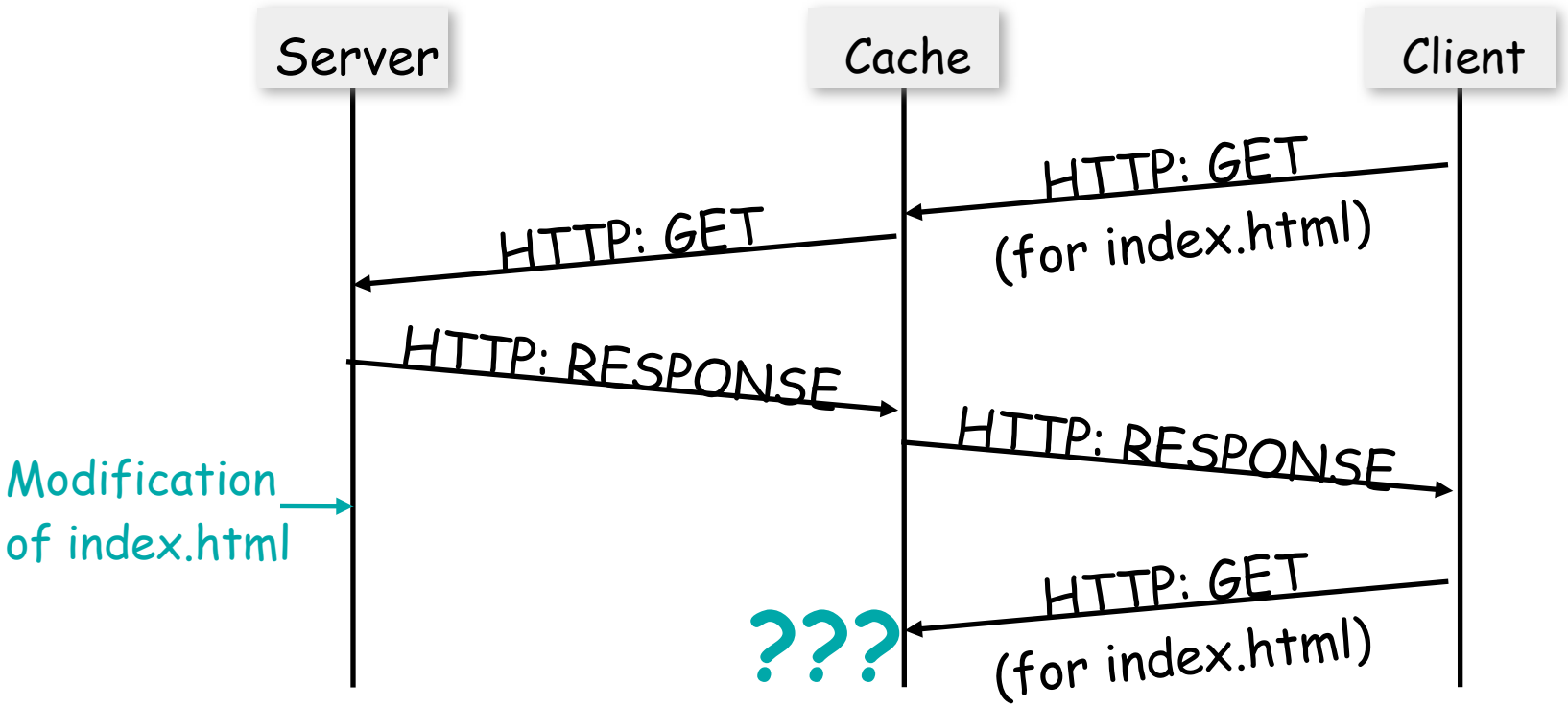
Trends In Cache Replacement

- Clever cache replacement strategies were a big issue in the early days of Web caching.
  - Many research papers focused exclusively on this topic.
  - Possible cost savings were the main driver.
- With recent developments, the focus shifts from efficiency to simplicity:
  - Steadily falling cost of hard disks,
  - Less Web traffic is cacheable,
  - Simple cache replacement strategies satisfy most situations and are less complex to implement,
  - Frequent updates of Web objects reduces the value of having large storage capacity.

Research on cache replacement has generally faded from the practical arena.

Cache Consistency

- Problem: How to make sure that the cached version of a web object is up-to-date and **not** staled?



Cache Consistency – Conditional GET (1)

- Cache uses conditional GET to check whether a cached object is stale.
- Example: First request from cache to origin server (due to a cache miss).

```
GET index.html HTTP/1.0
User-Agent: Mozilla/4.0
Accept: text/html image/gif, image/jpeg
```

- Example (contd.): Response from origin server to cache.

```
HTTP/1.0 200 OK
Date: Wed, 21 Jun 2000 04:29:01 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Wed, 21 Jun 2000 01:10:42 GMT
Content-Type: text/html <CR/LF>
<data>
```

Cache Consistency – Conditional GET (2)

- Example (contd.): Conditional GET from cache to origin server (due to a cache hit).

```
GET index.html HTTP/1.0
User-Agent: Mozilla/4.0
Accept: text/html image/gif, image/jpeg
If-modified-since: Wed, 21 Jun 2000 01:10:42
```

- Example (contd.): Response from origin server to cache.

```
HTTP/1.0 304 Not Modified
Date: Fri, 23 Jun 2000 04:29:01 GMT
Server: Apache/1.3.12 (Unix) <CR/LF>
(empty entity body)
```

Cache Consistency – Expiration Time

- **Strong consistency** (*ισχυρή συνέπεια*) and a freshness guarantee can be provided by using a conditional GET for every cache hit. However, this results in
  - Unnecessary delay and bandwidth consumption for requests yield not modified.
- Origin server can set an expiration time for web pages, which will be included in the entity header of HTTP responses:
  - A web page will not be modified before it expires.
  - There is no need for consistency checks (e.g. using conditional GET) ahead the expiration time.
- However, servers rarely use this option (except the expiration time “now”).

Why expiration time “now”?

Cache Consistency - Heuristic

- Strong consistency might not always be required.
- It is not always necessary to provide strong consistency, which allows the usage of heuristics instead of explicit freshness checks.
- Idea: Documents which have not changed for a long time are unlikely to change.
- Update heuristic: Hold document for a time proportional to the known lifetime of the object (e.g. using the 50% rule); example:

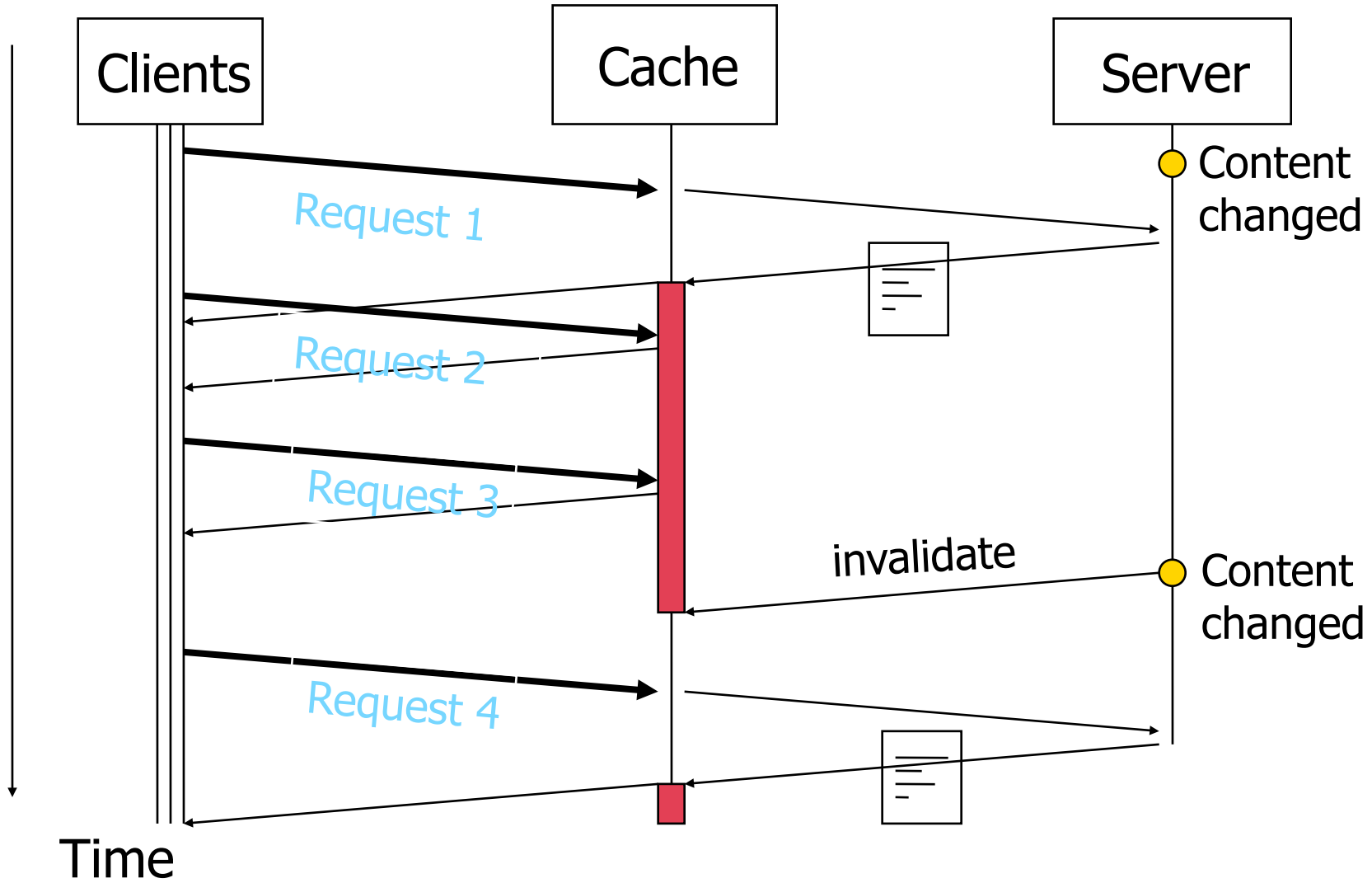
```
HTTP/1.0 200 OK
Date: Mon, 22 Jun 1998 04:29:01 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Mon, 8 Jun 1998 04:29:01 GMT
Content-Type: text/html <CR/LF>
```

  - Hold document for 7 days until 29 Jun 1998 04:29:01 GMT.

Cache Invalidation

- Idea: Server knows best when a Web page is updated!
- Server remembers where its pages are cached.
- When a page is modified, server notifies the caches which have a copy of this page.
- Caches mark page as stale.
- Subsequent client requests will fetch fresh copy from server.

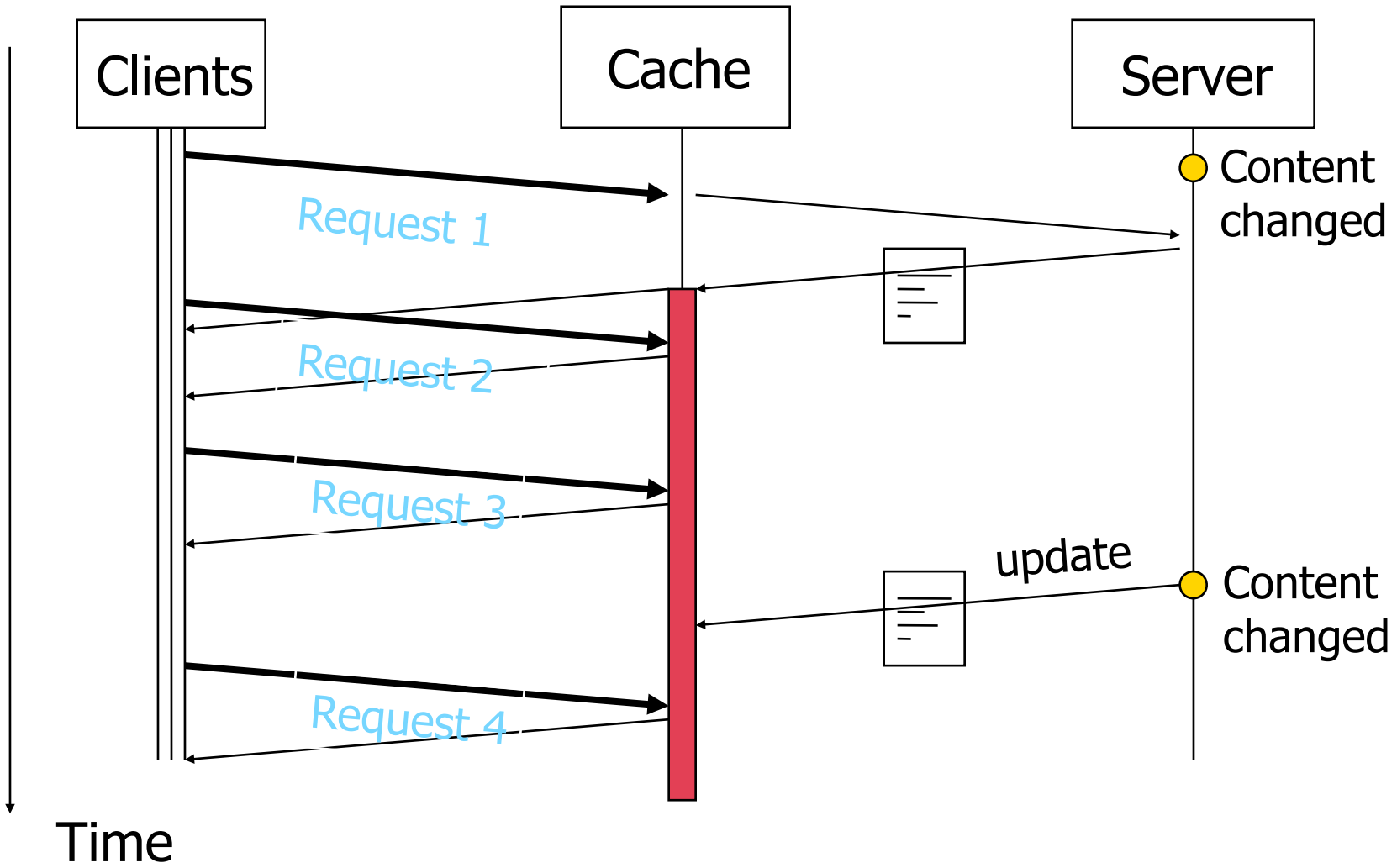
Example: Cache Invalidation



Cache Update

- Server remembers where its pages are cached.
- When a page is modified, server notifies the caches and gives them an updated version of the page (usually used together with application layer multicast).
- Caches always assume that they have the latest content.

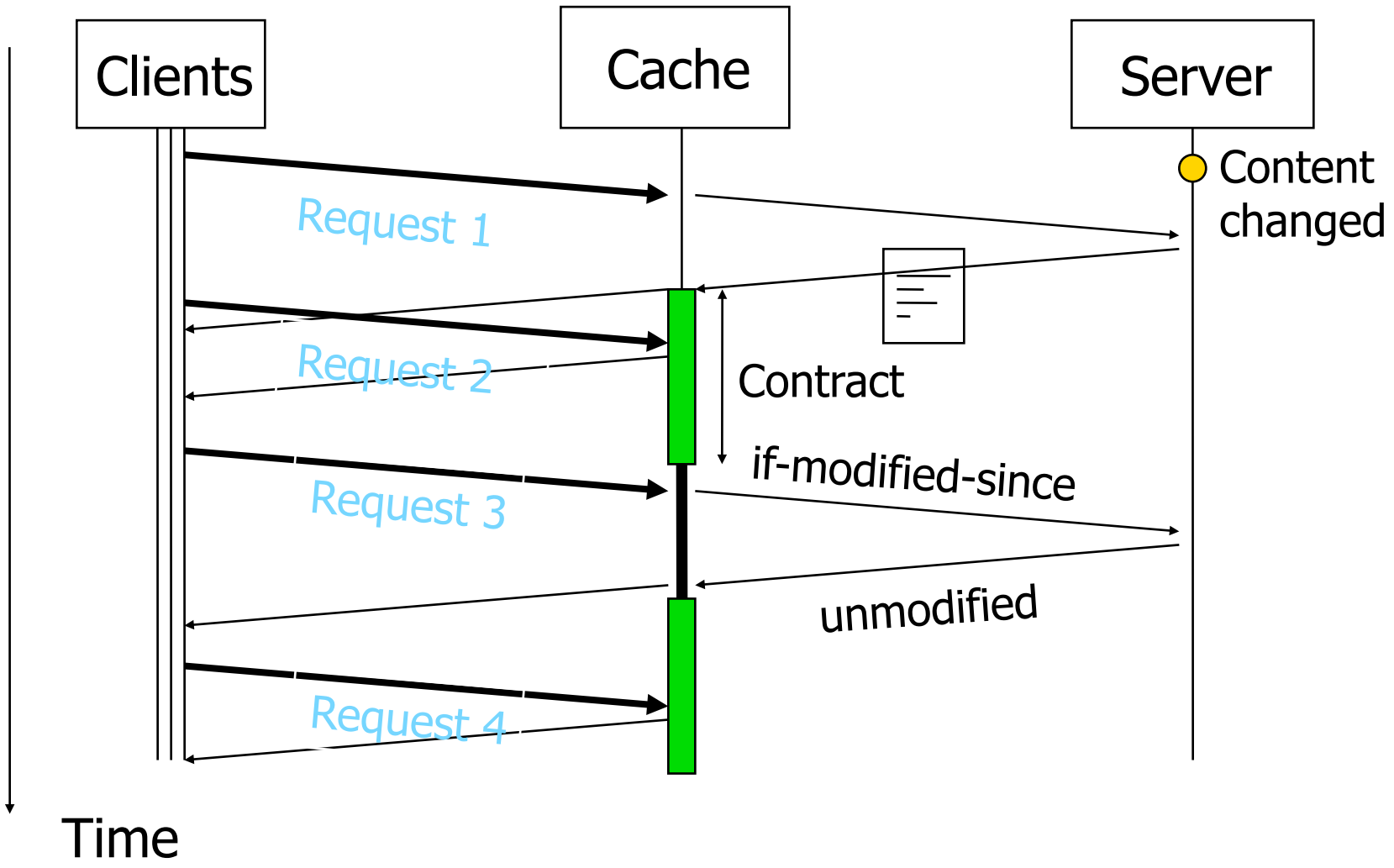
Example: Update



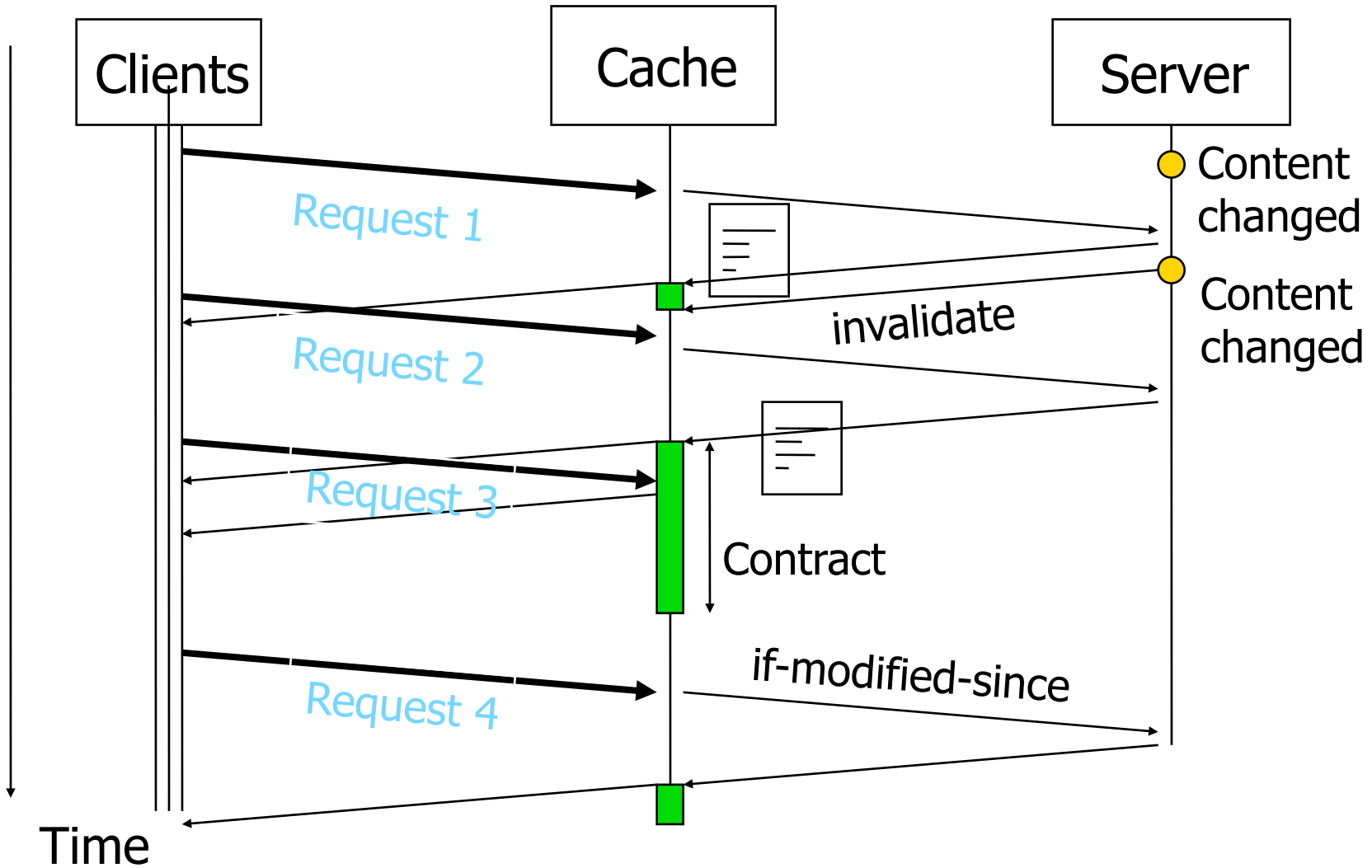
Invalidation Contracts

- Invalidation/update traffic can be very large.
- To limit invalidations server gives invalidation contracts to caches.
  - Contracts have expiration time.
- When content changes server notifies only those caches whose contract has not expired.

Example 1: Invalidation Contracts

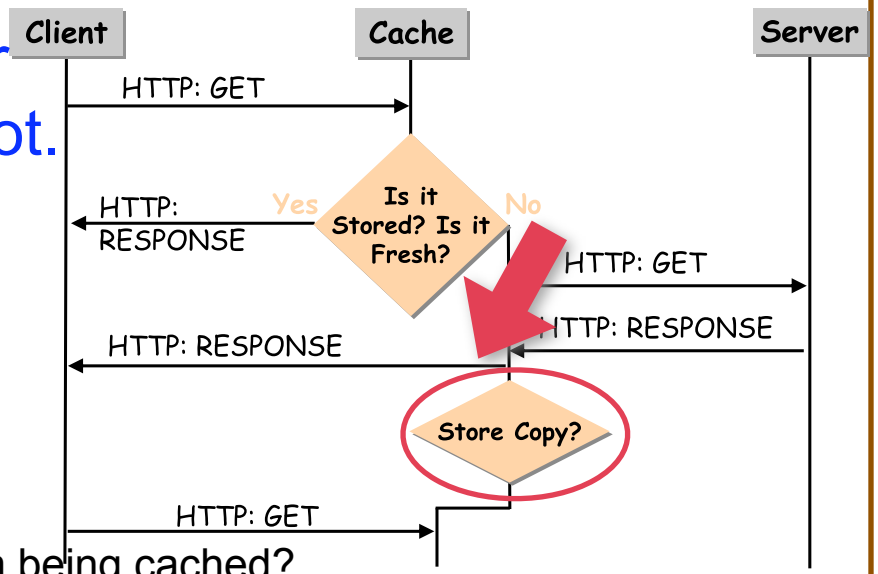


Example 2: Invalidation Contracts



Deciding Whether To Cache An Object

- Web cache must decide whether an object should be cached or not.
- Different Web caches can have different ways to make decision.
- Common criteria are:
  - Is the object cacheable?
    - Are there protocol requirements that prevent the object from being cached?
    - Is the content typically uncacheable?
  - Is the cached object likely to be used again?
  - Will the decision to cache a particular object lead to the replacement of one or more stored objects?
- A Web cache can use some or all of these criteria.



What Is Cacheable?

- Web cache can decide whether a Web object is cacheable based on two factors:
  - Content-specific considerations,
  - HTTP-specific considerations.
- Content-specific considerations are separated from the protocol issues.
  - Deal with business and/or political aspects.
  - Also affected by policies for cache consistency/revalidation.
- HTTP-specific considerations require Web caches to abide to the various directives embedded in HTTP messages.
  - Might impact hit ratio of Web cache.
  - Not all Web caches on the market strictly follow these rules.
  - Might be overwritten by content-specific considerations.

Content-Specific Considerations

- Web cache makes a local decision about the cacheability of an object separate from protocol restrictions, e.g. based on
  - Object size, URL, indicators for dynamic content, etc.
- Motivation for content-specific cacheability considerations:
  - Cache consistency and object freshness,
  - Business interests (e.g. the ability to charge for more bytes delivered if object is returned rather than “304 Not Modified”),
  - Storage considerations (e.g. do not cache large objects),
  - Privacy issues,
- Processing load on a Web cache might also have an impact.
  - Busy Web cache might decide to not cache a response coming back as result of a cache miss.

HTTP-Specific Considerations

- HTTP embeds cacheability information in the HTTP header.
  - Cache control directives (e.g. `Cache-Control: no-cache`)
- HTTP defines rules indicating what HTTP responses are cacheable.
  - A response is cacheable only if request method and headers, and response status and headers all indicate so.
    - E.g. responses to PUT, DELETE, OPTIONS are not cacheable.
    - E.g. response to POST is not cacheable unless indicated otherwise.
- Some responses include object-specific information from the origin server that may preclude caching of the message.
- A well-behaved Web cache must abide by the constraints imposed by HTTP.
  - Not all Web caches do so!

Cache Control Directives: Motivation

- Client might not want to get a cached object, e.g.
  - When hitting the [CTRL/F5] reload button in the browser.
- Content provider might not want an object to be cached, e.g.
  - Objects which rapidly change,
  - Password protected objects,
  - Hit counts.
- Server could use an expiration time “now” to increase probability of a freshness check.
  - Inefficient, makes proper usage of expiration time impossible.
- HTTP/1.1 defines cache control directives that **must** be obeyed by all systems (i.e. network caches and client software).
  - But...

Cache Control Directives In HTTP/1.1

- Cacheability of an object can be controlled using the `Cache-Control` header in an HTTP request or response.
- Some of the directives in requests/responses:
  - `no-cache`: client forces freshness check with origin server,
  - `no-store`: do not store any portion of this request or response,
  - `max-age=n`:
    - client is prepared to accept object without freshness check if age is < n seconds
    - server imposes freshness check after n seconds.
- Some of the directives in responses:
  - `public`: Object is cacheable by client and network caches,
  - `private`: Object can be cached only on private caches (client),
  - `no-cache`: Object must not be cached,
  - `no-transform`: No content transformation allowed,
  - `must-revalidate`: Cache must revalidate expired content
  - `proxy-revalidate`: similar, but revalidation not required for private caches

Cache Control Directives: Examples

- HTTP Request

```
GET index.html HTTP/1.0
User-Agent: Mozilla/4.0
Cache-Control: no-cache
Accept: text/html image/gif, image/jpeg
```

- HTTP Response

```
HTTP/1.0 200 OK
Date: Wed, 16 Feb 2005 04:29:01 GMT
Server: Apache/1.3.12 (Unix)
Last-Modified: Wed, 16 Feb 2005 01:10:42 GMT
Cache-Control: private
Content-Type: text/html <CR/LF>
<data>
```

HTTP Headers vs. HTML Meta Tags

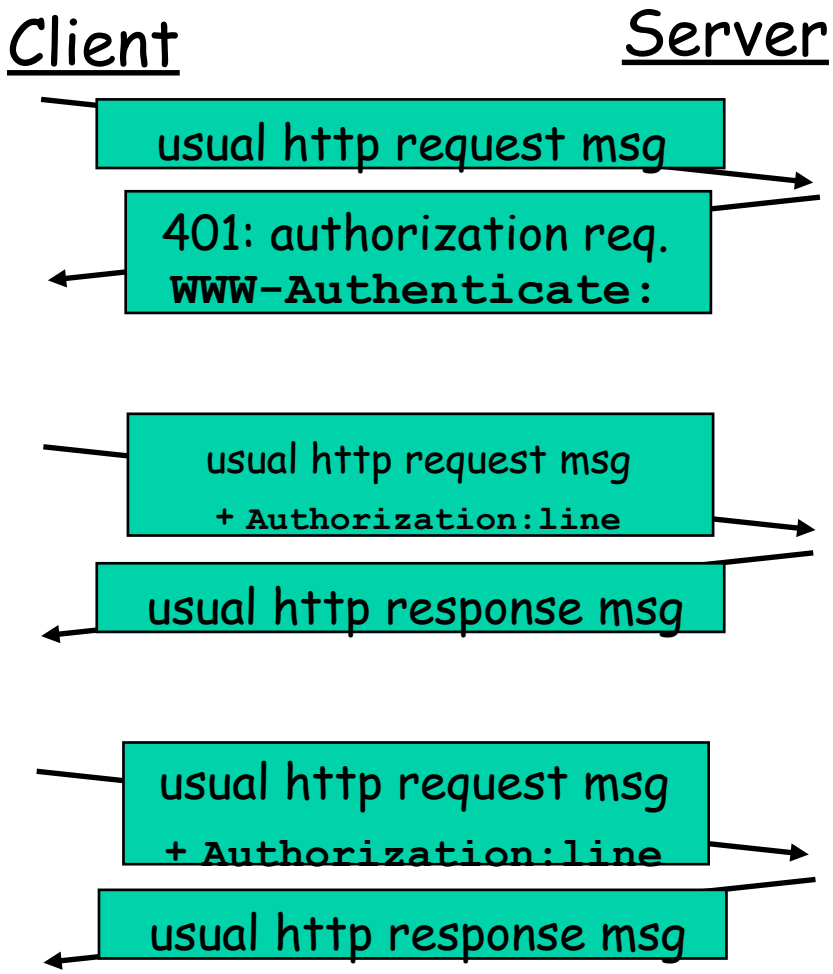
- Problem: HTTP headers can only be set by the Web server, i.e. content authors do not have direct control over setting HTTP headers.
- Content authors can “imitate” cache control directives by adding appropriate HTML Meta Tags to their content, e.g.

```
<HTML>
<HEAD>
<TITLE> HTML Meta Tags Example </TITLE>
<META HTTP-EQUIV="CACHE-CONTROL" CONTENT="NO-CACHE">
</HEAD>
<BODY>blablabla</BOFY></HTML>
```

- But: HTML meta tags are not very effective.
  - Are usually only honored by Web browser caches,
  - Web caches normally do not parse HTML payload.

Recap: HTTP Authentication

- Authentication goal: Control access to server documents.
  - Basic Authentication – a challenge/response mechanism.
- Stateless: Client must present authorization in each request.
- Authorization: typically name, password
  - **`Authorization:`** header line in request
  - if no authorization presented, server refuses access, sends **`WWW-authenticate:`** header line in response



HTTP Authentication

- Client requests object from origin server
- Server responds with “401 Authorization Required” status code
- Client software receives 401 status code and prompts user for username and password
- Client resends the request, this time with the proper AUTHORIZATION header
- Authorization is done for all subsequent requests to the server
  - Browser remembers the username and password and includes them in subsequent requests

How to make sure that a cache does not deliver password protected sites to unauthorized users?

Authentication And Web Caching

- HTTP/1.1 specifies mechanisms for handling password protected objects.
- Whenever an origin server distributes a password-protected object, it includes in the response an HTTP header `CACHE-CONTROL: proxy-revalidate.`
- A cache is allowed to cache the object, but it has to mark it with "proxy-revalidate".
- Later, when a request for the object arrives, the cache sends a conditional `GET` to the origin server.
  - If the conditional `GET` includes a valid `AUTHORIZATION` header, origin server responds with "`301 Not Modified`" and the cache forwards the object to the client.
  - If the conditional `GET` does not include valid authorization, origin server responds with "`401 Authorization Required`" and the cache forwards it to the client.

Caching and Online Advertisement

- Content providers sell web space for advertisement (banners and inline images with links to the company that the ad promotes).
- Content providers need to determine the number of hits to the page hosting the advertisement.
  - Advertisers want to know the exposure of their ad,
  - Billing to advertisers may be based on the hit count.
- Content providers also want to know from where the requests come and the number of individual user hits (e.g. using cookies).
- Caching prevents sites from getting accurate access counts.
- Solutions used in practice:
  - Mark Web pages non-cacheable, thus forcing requests to reach the origin server,
  - Providing Web cache logs to the content providers,
  - Usage of non-cacheable Web bugs.

Web Bugs

- "Web bugs" are little transparent graphics embedded in Web pages.
  - Typically marked as non-cachecable.
  - Trigger a download from the origin server

```
<HTML>
  <HEAD>
  <TITLE> Web Bug Example </TITLE>
  </HEAD>
  <BODY>
  You won't see anything on this page.
  <img src="web-bug.gif">
  </BODY>
</HTML>
```

- Web bugs have been used in controversial ways in the past, leading to privacy concerns and giving them a "bad taste".

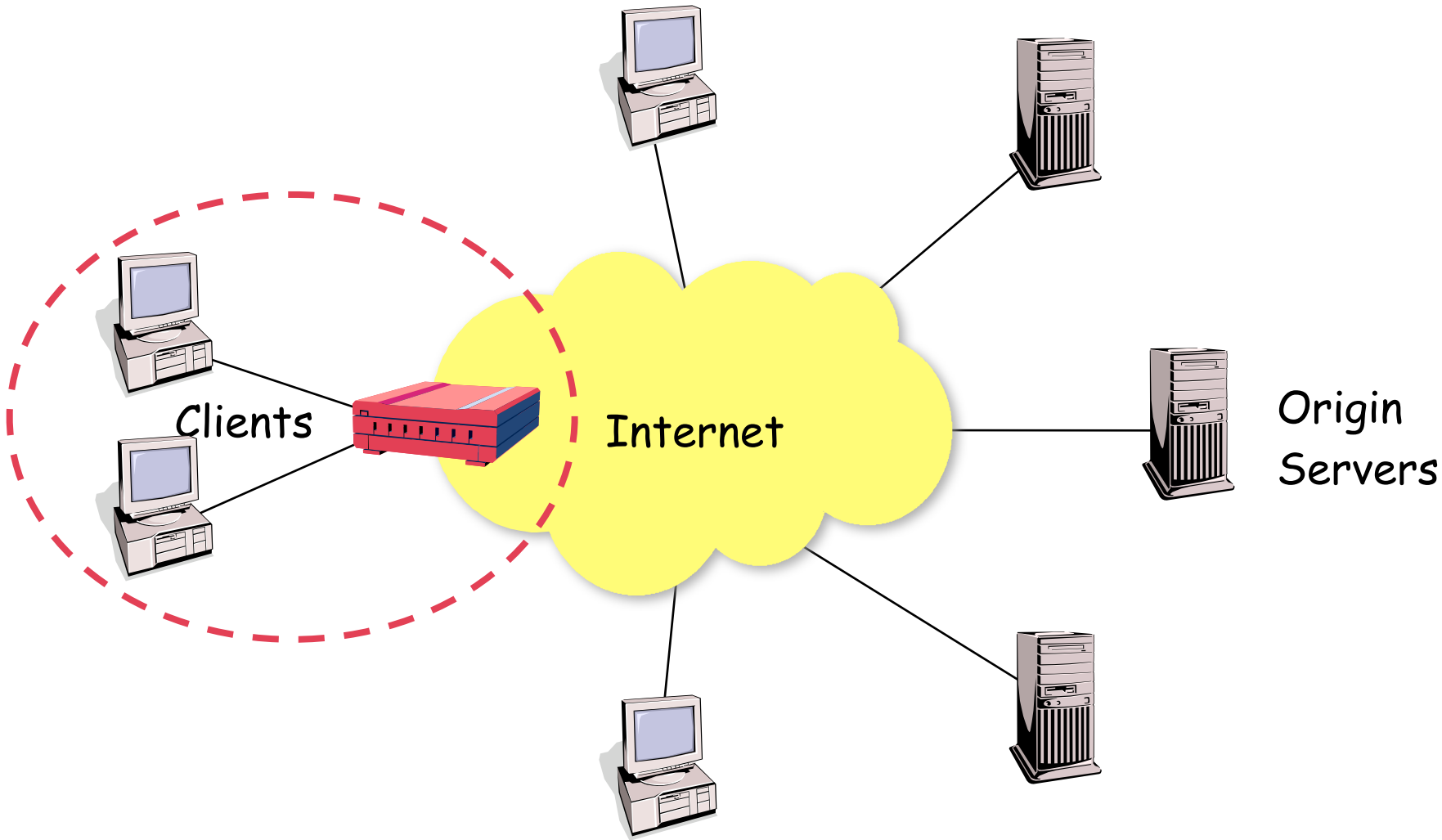
Summary: Caching Snags

- Dynamic content,
- Secure transactions and encrypted content,
- Cookies,
- Hit counters,
- Online advertisement,
- Access control,
- Privacy-concerned users.

Web Cache Deployment Options

- Two main methods to deploy Web caches, depending on whom the caching service is provided for.
  - Forward proxy
    - Proxy acts of behalf of the content consumer and helps the content consumer in retrieving requested content.
    - Also referred to as avatar or delegate.
  - Reverse proxy
    - Proxy acts on behalf of the origin server and helps the origin server to deliver its content.
    - Also referred to as surrogate.
    - Can be deployed either in front of the origin server or at the network edge.

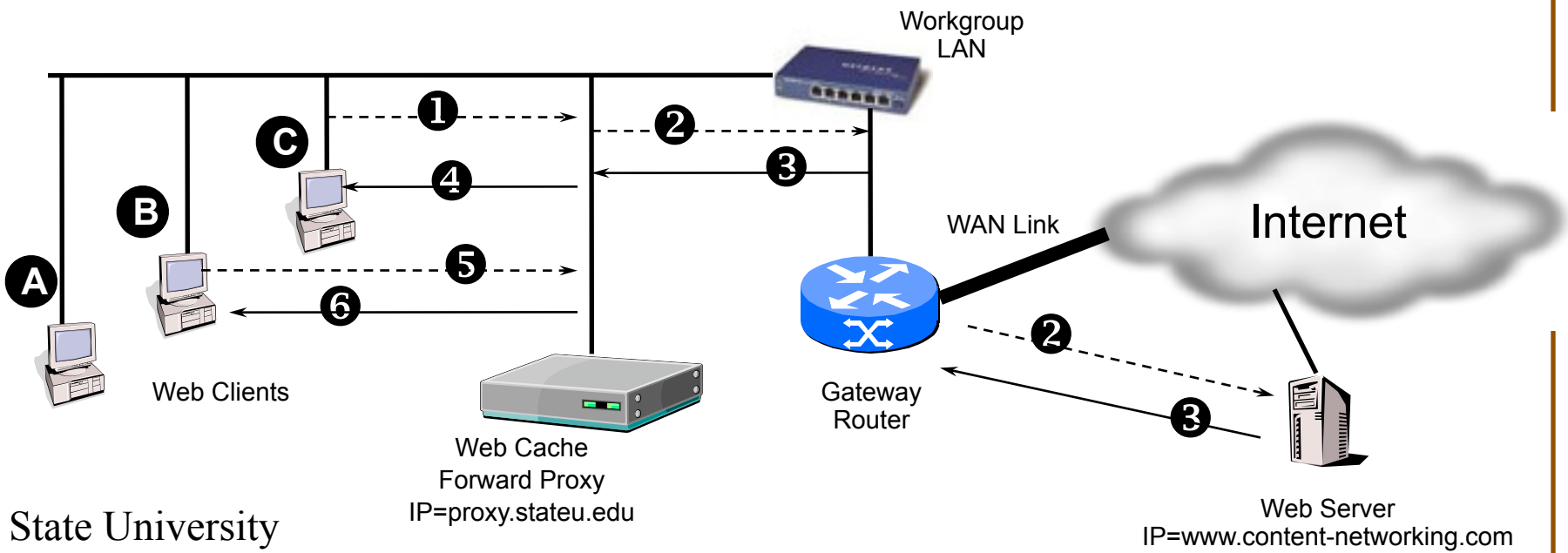
Forward Proxy



Example: Forward Proxy Deployment

HTTP Request:

```
GET http://www.content-networking.com/index.html HTTP/1.1
Host: www.content-networking.com
User-Agent: Mozilla/6.0
Accept: text/html, image/gif, image/jpeg
```

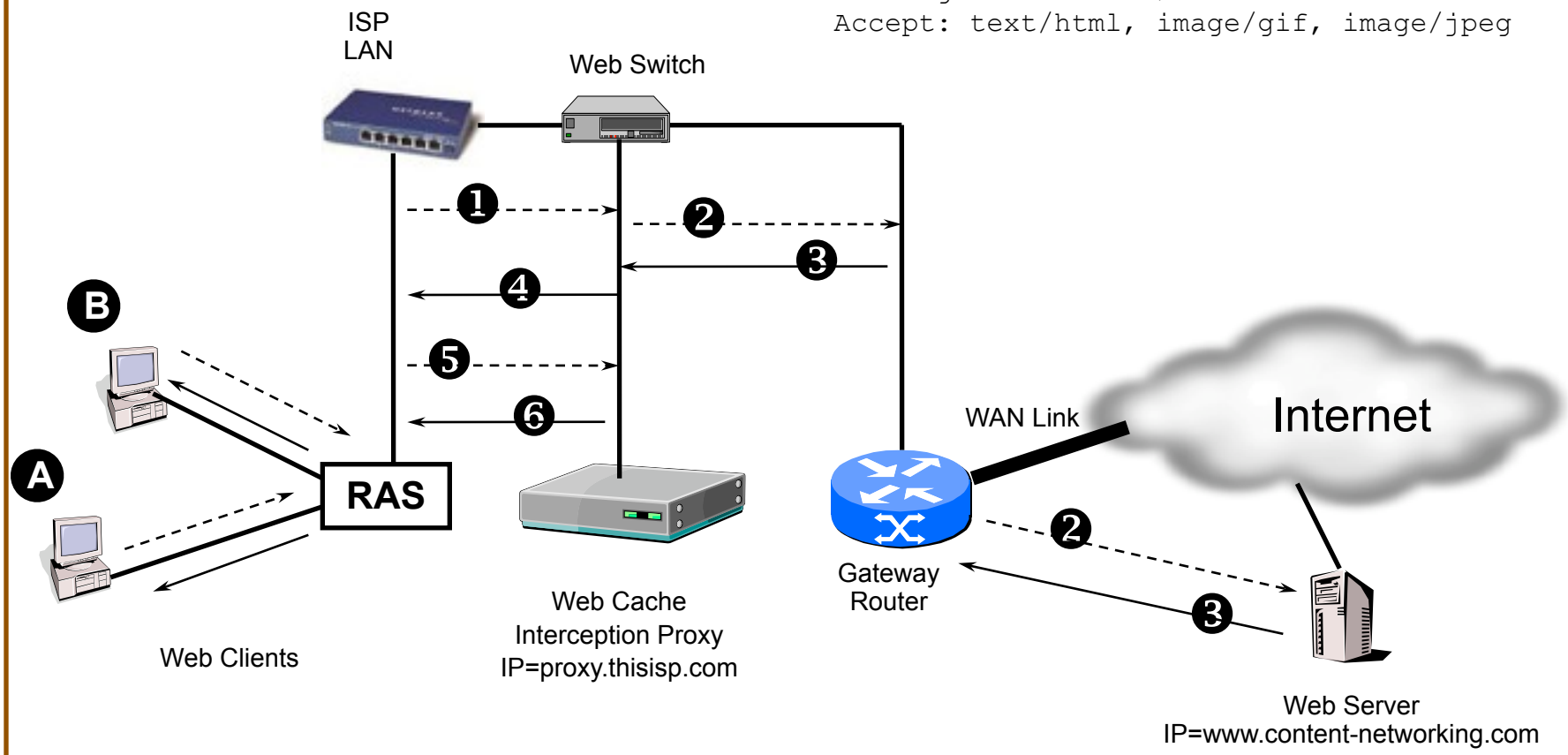


State University
Chemistry Department

Example: Interception Proxy

HTTP Request:

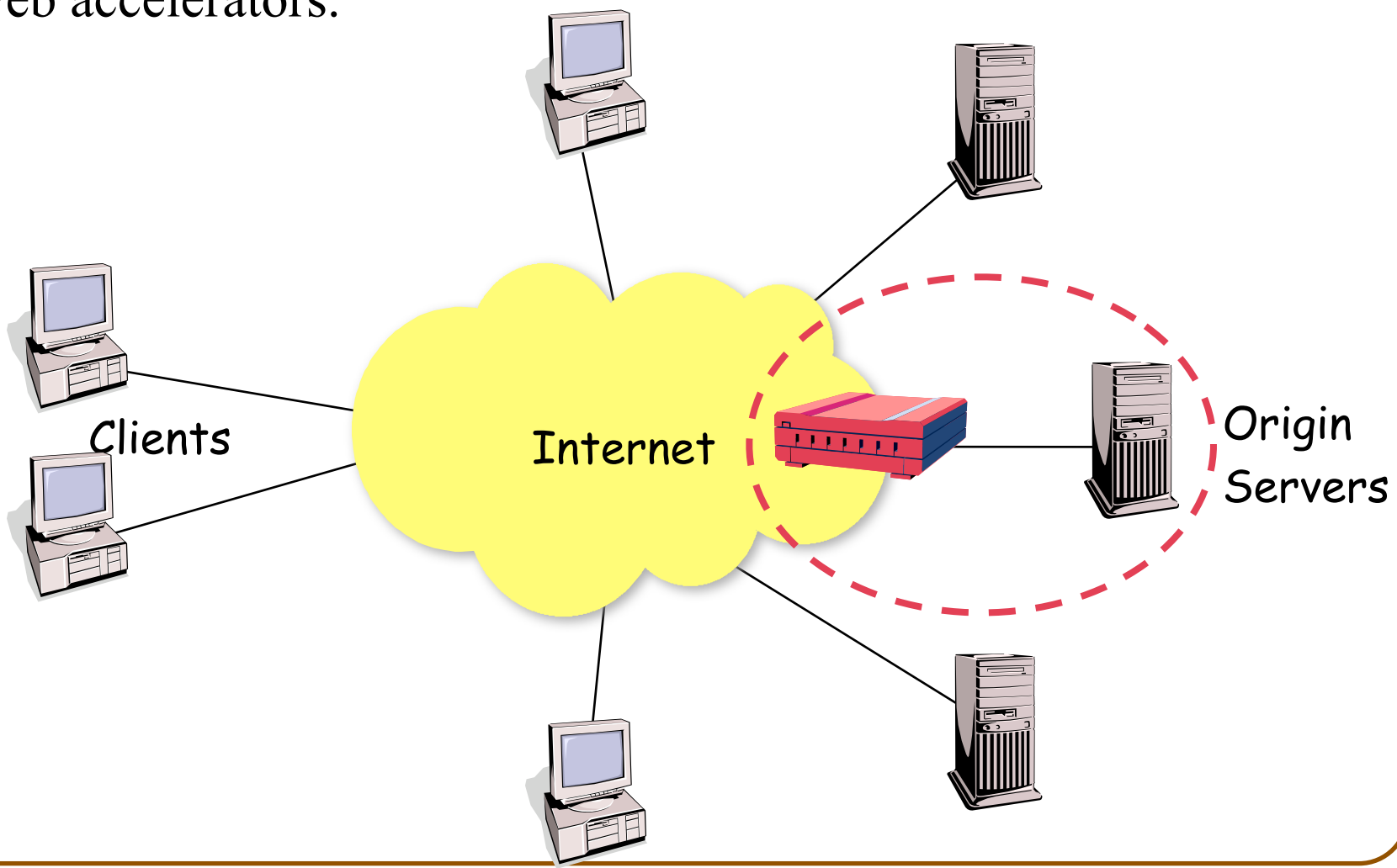
```
GET index.html HTTP/1.1
Host: www.content-networking.com
User-Agent: Mozilla/6.0
Accept: text/html, image/gif, image/jpeg
```



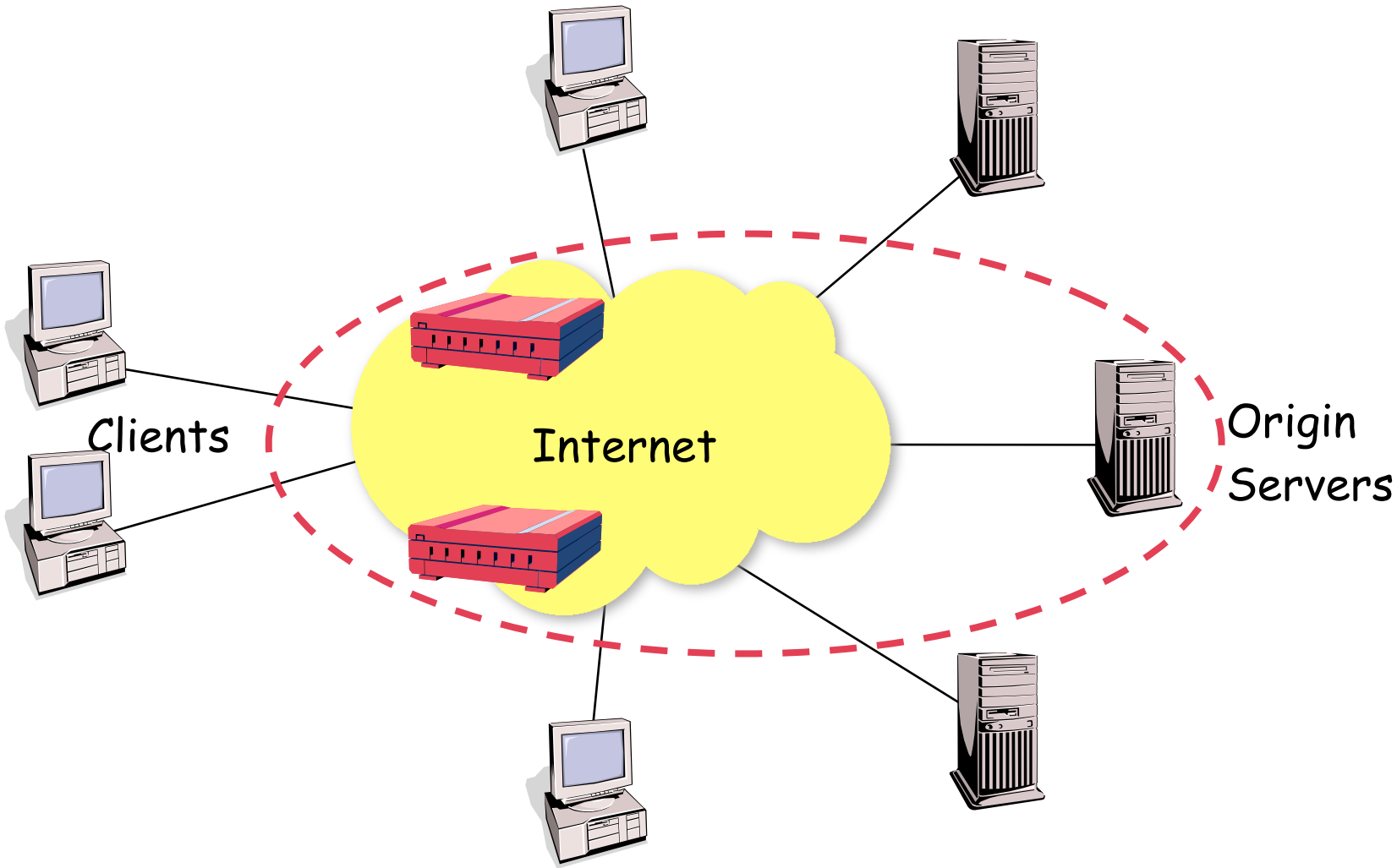
Internet Service Provider

Reverse Proxy

Also known as surrogate proxies or web accelerators.



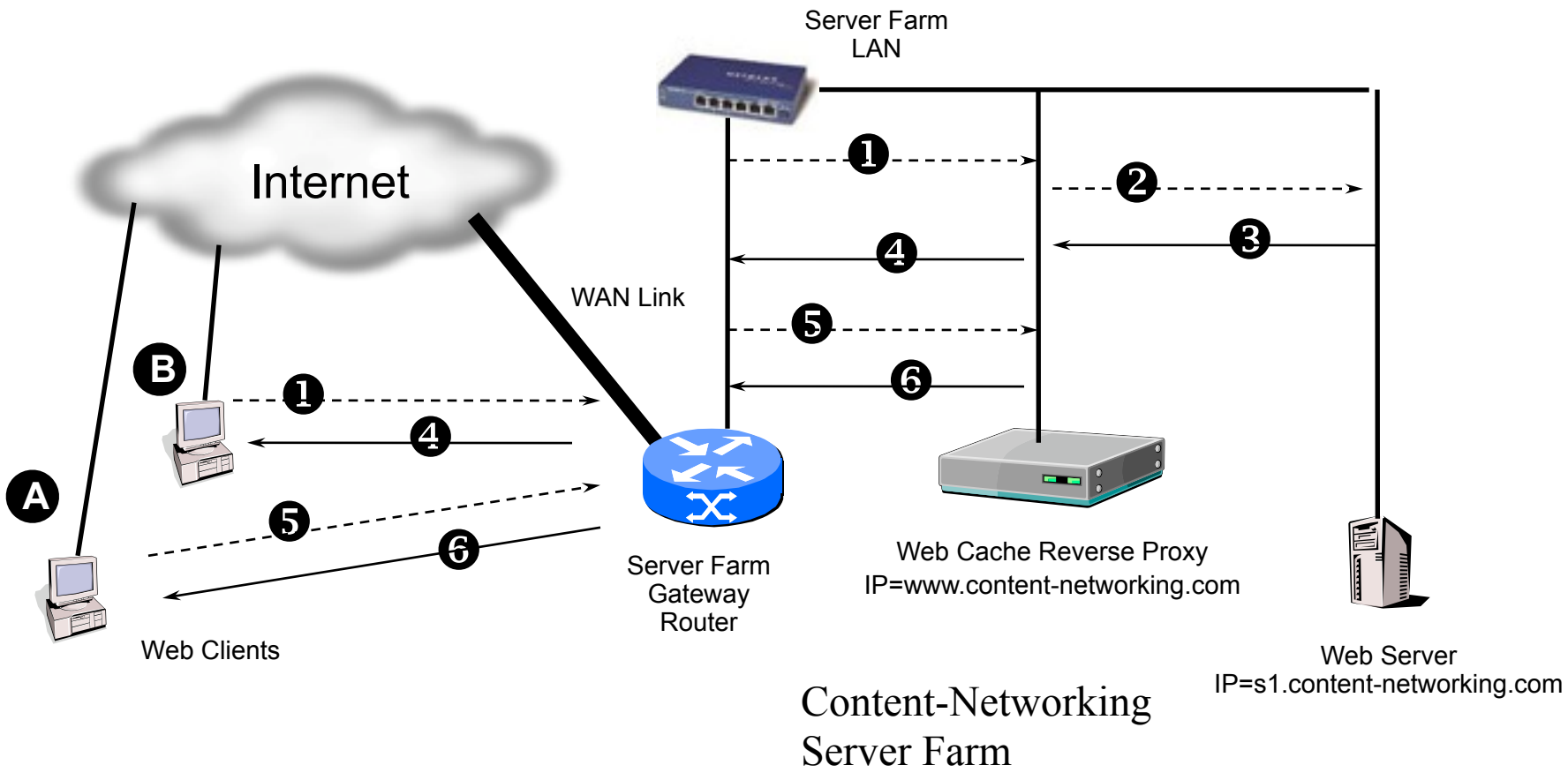
Distributed Reverse Proxy



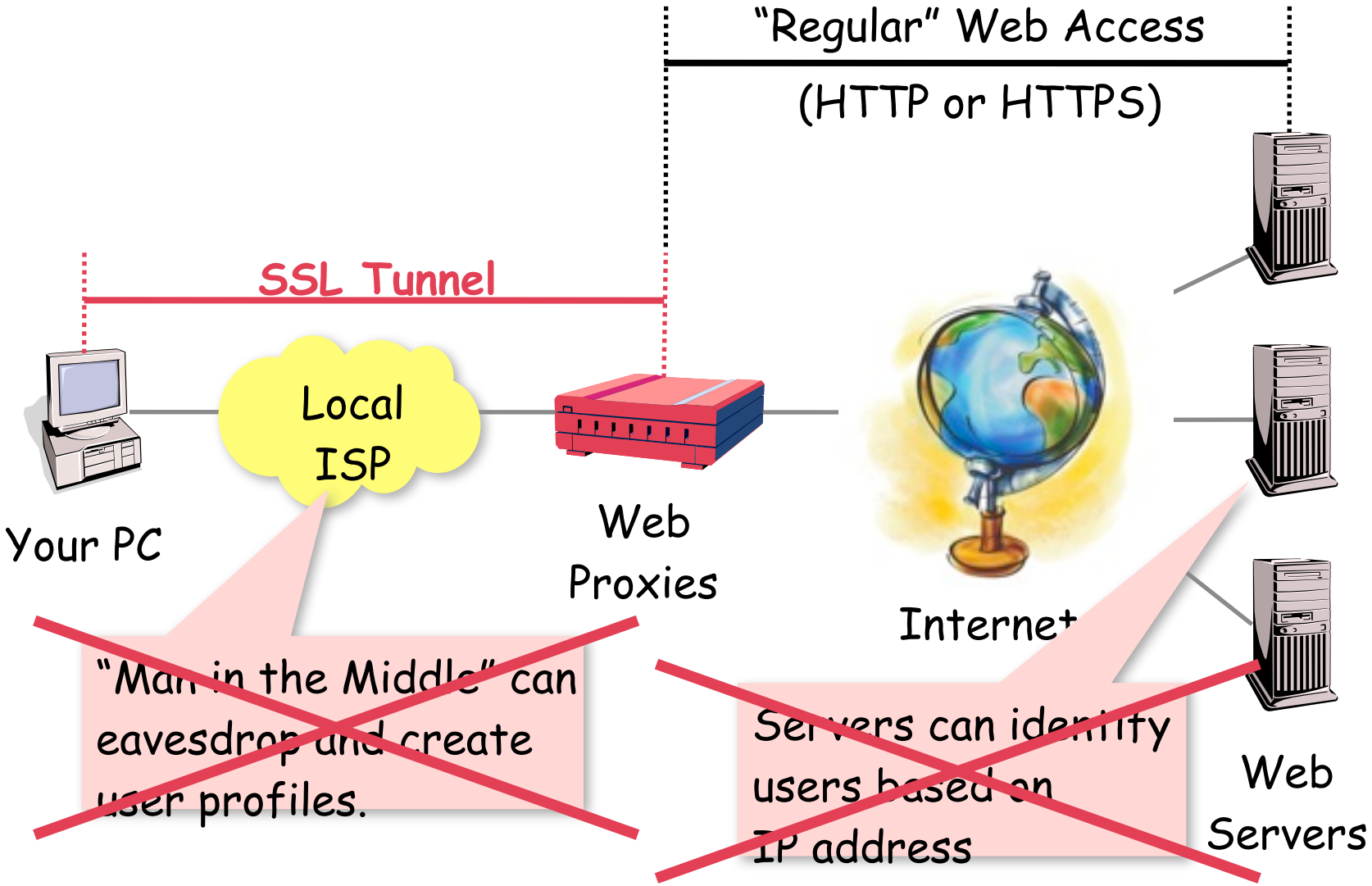
Example: Reverse Proxy Deployment

HTTP Request:

```
GET index.html HTTP/1.1
Host: www.content-networking.com
User-Agent: Mozilla/6.0
Accept: text/html, image/gif, image/jpeg
```



Anonymous Web Surfing



Caching vs. Replication

- Content replication can be seen as an alternative to Web caching.
  - Caching moves requested content closer to the user,
  - Replication copies content of an origin server to mirror sites.
- With replication, *all* content of an origin server may be available at all times at the mirror sites.
  - Protocols other than HTTP can be used for content replication.
  - Replication can be scheduled for low-traffic times (e.g. overnight).
  - Multicast can be used for efficient replication.
  - Content staleness can be avoided through update mechanisms.
- But: Typically less efficient with respect to storage needs.
  - Also more management overhead.

Measuring Web Cache Performance

- Throughput of a Web cache expressed in either
  - Requests per second – limited by the processing speed of the Web cache, or
  - Bytes per second – can be limited by the bandwidth of the network connection.
- Hit ratio is a measure of network savings provided by the cache.
  - Request hit ratio,
  - Bandwidth hit ratio.
- Response time measures time between making a request and starting to receive the corresponding response.
  - Distinguish between cache hit and cache miss.
  - Consider the distribution of response times and lowest, average, longest times.

Challenges In Measuring Cache Performance

- Definition: The workload of a Web cache consists of all inputs (e.g. HTTP requests) received by the Web cache over time.
- Web cache performance strongly depends on the workload characteristics, e.g.
  - Time between user requests,
  - Size of the requested objects,
  - Popularity of the various objects,
  - Cacheability of the various objects.
- Workloads used for performance measurements must be realistic.
  - Careful analysis of Web cache/server logs is useful.
- Popularity of Web objects is often expressed following a Zipf-like distribution.
  - $P(i) = ki^{-c}$ (exponent "c" typically close to unity).

Challenges In Benchmarking Web Caches

- Web cache selection factors include not only performance, but also
  - Price,
  - Reliability,
  - Recovery from failures,
  - Total object store size.
- Need to related price with performance.
  - Throughput normalized by price can give a rough estimate of Web cache cost effectiveness (e.g. hits per second per $1000).
- Web cache benchmarking by independent, third party organizations is helpful, e.g.,
  - “Cache-Offs” by The Measurement Factory, using the their Web Polygraph performance tool.

Polygraph Benchmark

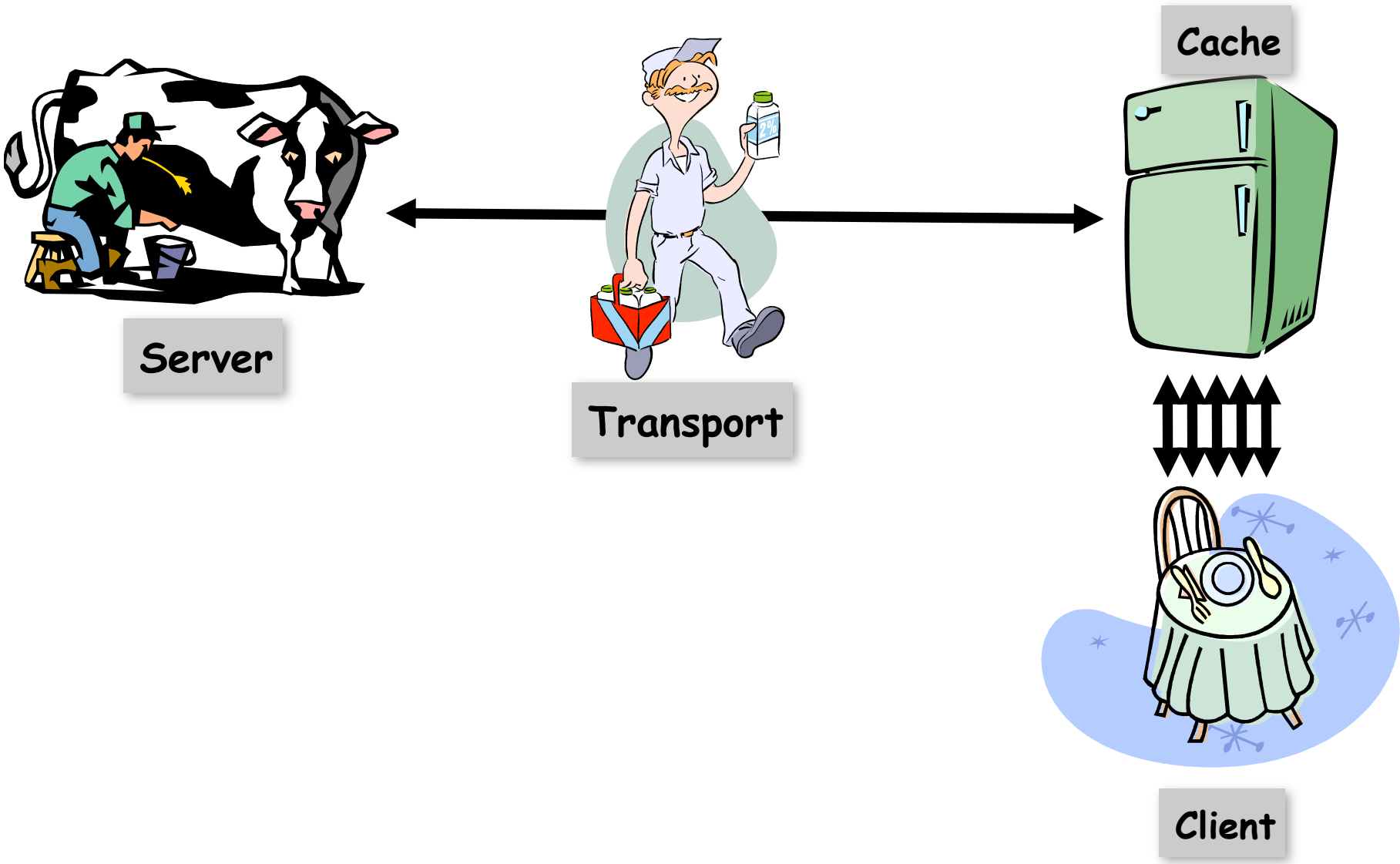
- Polygraph benchmark originally defined by IRCache, a small group of people that were employed by the University of California San Diego.
  - Funding came primarily from the National Science Foundation.
  - Now spun off into a separate company, named “The Measurement Factory”.
- Polygraph is a de-facto Industry-standard benchmark for Web caches.
- Several workload types, e.g. DataCom-1 workload:
  - Uniform request pattern,
  - 80% cacheable content,
  - 55% hit ratio,
  - 12 KB mean object size,
  - 4-hour long test,
  - Persistent connections.

Recipes for a better Cache

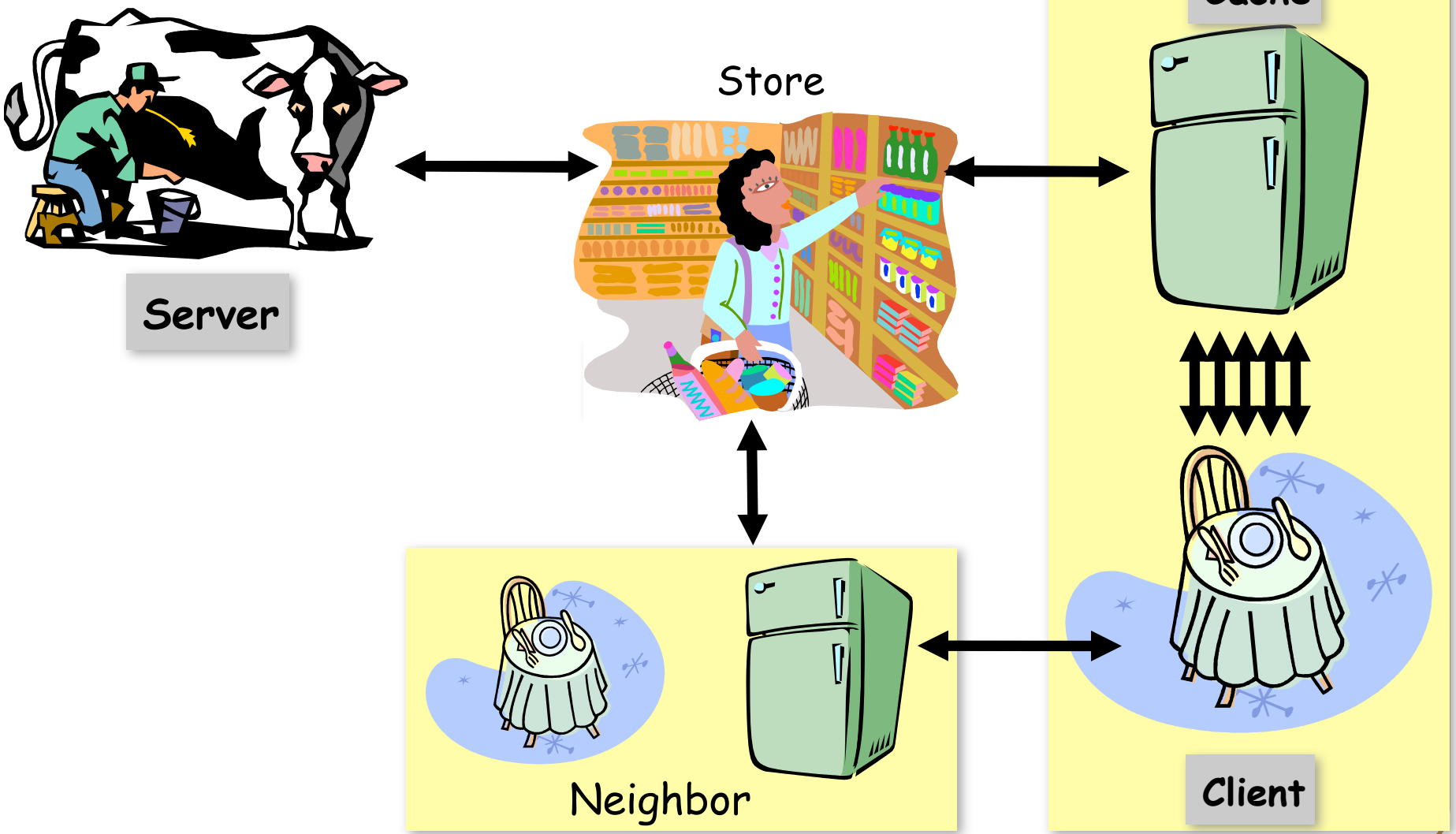
- Bigger hardware:
  - Many disks (varies from 1 disk to 14 disks .. can be more),
  - Faster NIC (1 Gbps or multiple 100 Mbps),
  - Faster CPU.
- Better software:
  - Parallel processes/multiple threads to leverage multiple disks,
  - Smarter disk read/write (e.g. read/write in large chunks),
  - Raw disk read/write,
  - More efficient data structures and smarter algorithms,
  - Efficient garbage collection in disks.
- Can provide up to two orders of magnitude higher throughput.

In a large network, it requires more than having a single efficient cache…

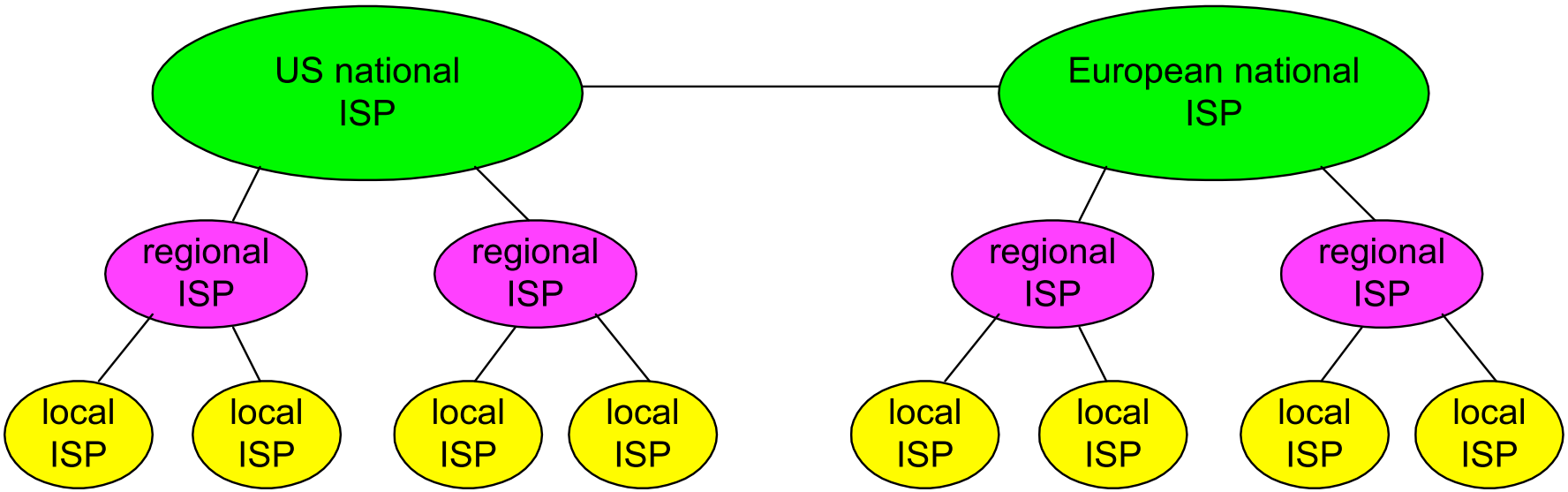
Recap: Get Milk The Better Way



Getting Milk – An Even Better Way

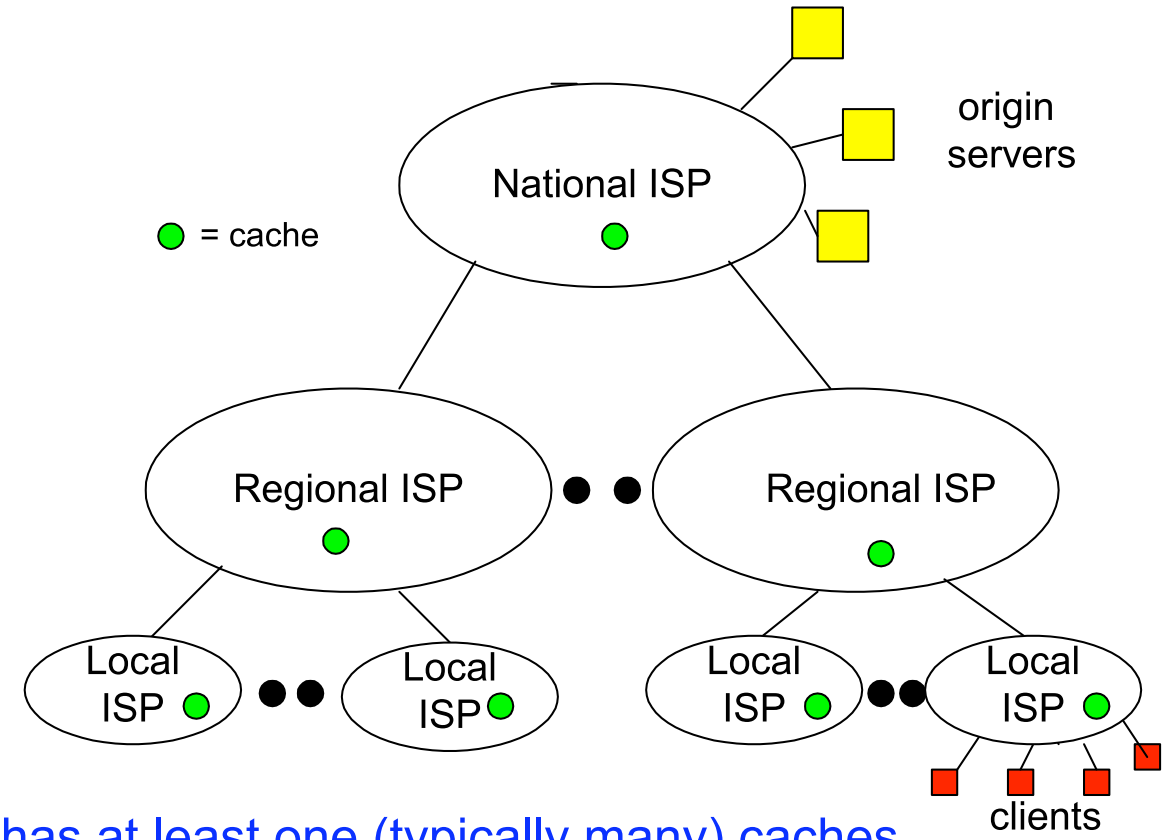


Backbones, ISPs, and NAPs



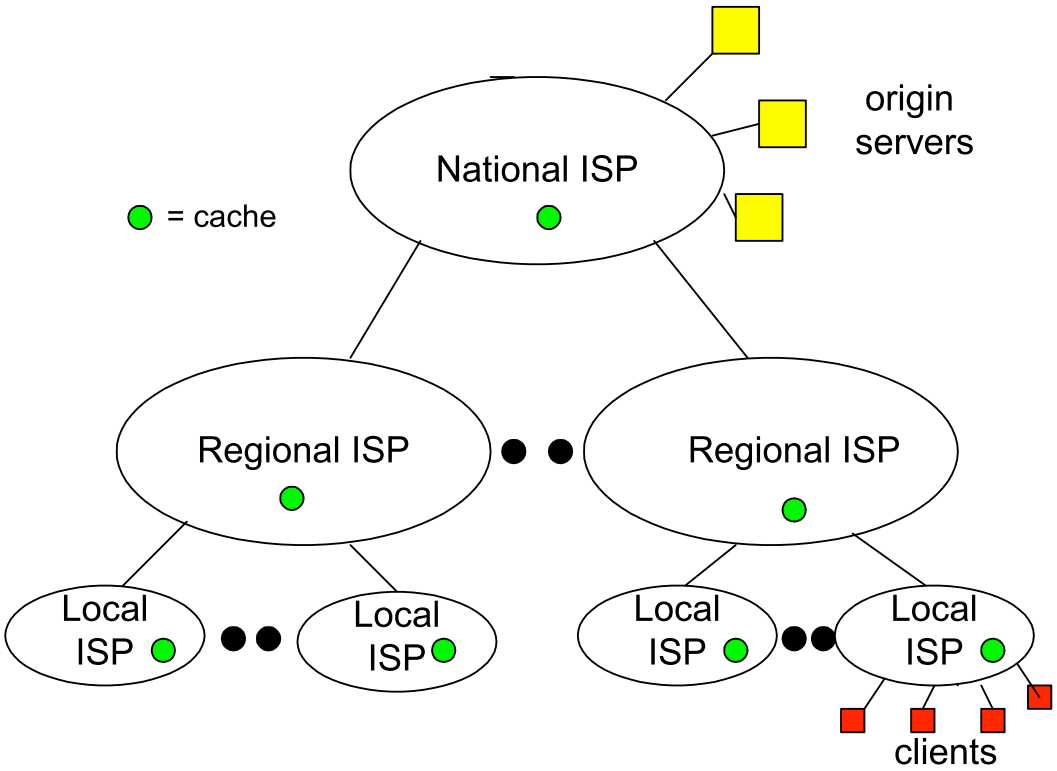
- The topology of the Internet is loosely hierarchical.
  - National, regional and local/institutional ISPs,
  - National ISPs in different countries can peer.
- Within one country there can be multiple regional ISPs.
  - They interconnect at NAPs - Network (or National) Access Points.
  - Also called MAEs: metropolitan access exchanges.

Hierarchical Caching



- Each ISP has at least one (typically many) caches.
- A cache that is located in an ISP higher up in hierarchy
  - provides service to ISPs lower in hierarchy,
  - serves as an aggregation point for all caches lower in the hierarchy.

Hierarchical Caching – Hit Rates



Illustrative
Hit Probabilities

HIT = .25

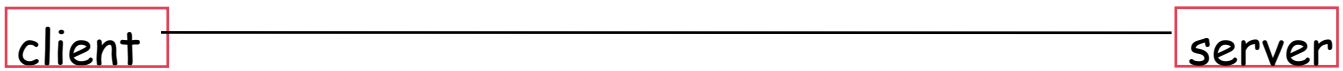
HIT = .33

HIT = .4

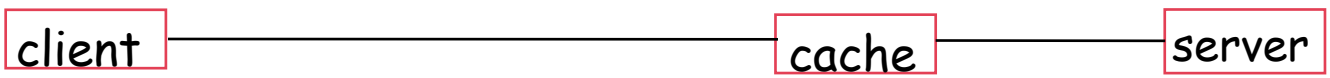
- Probability that object is found in institutional cache: .4
- Probability that object is found in regional cache: .33*(1-.4) = .2
- Probability that object is found in national cache: .25*(1-.4-.2)= .1
- Probability that object is found in the cache hierarchy: .4 + .2 + .1 = .7

Cache Chaining

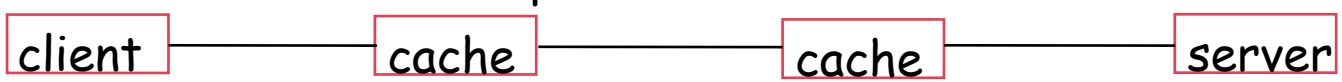
no cache:



one cache: user configures browser to point to cache:



chain of caches: user configures browser to point to cache;
first cache points to second cache:



- Hierarchies use cache chaining

Insert: Hop-by-Hop Headers

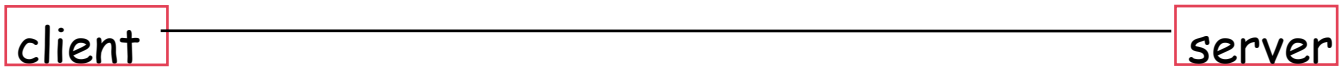
- Problem:
  - A pair of adjacent caches in a chain could use alternate way to exchange HTTP messages
    - For example, different compression techniques
  - Requires inclusion of special metadata only between those caches
- Lead to the introduction of Hop-by-Hop headers
  - Are valid only for a single transport-layer connection
  - Cannot be stored by caches or forwarded by proxies
  - Are identified by `Connection:` header field, for example

    `Connection: header 1, header2`

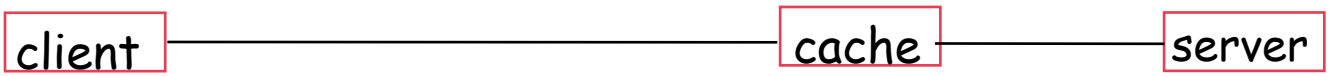
    indicates that header1 and header 2 are hop-by-hop

Cache Chaining

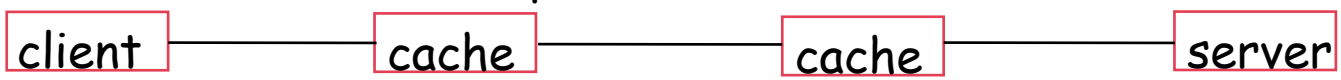
no cache:



one cache: user configures browser to point to cache:

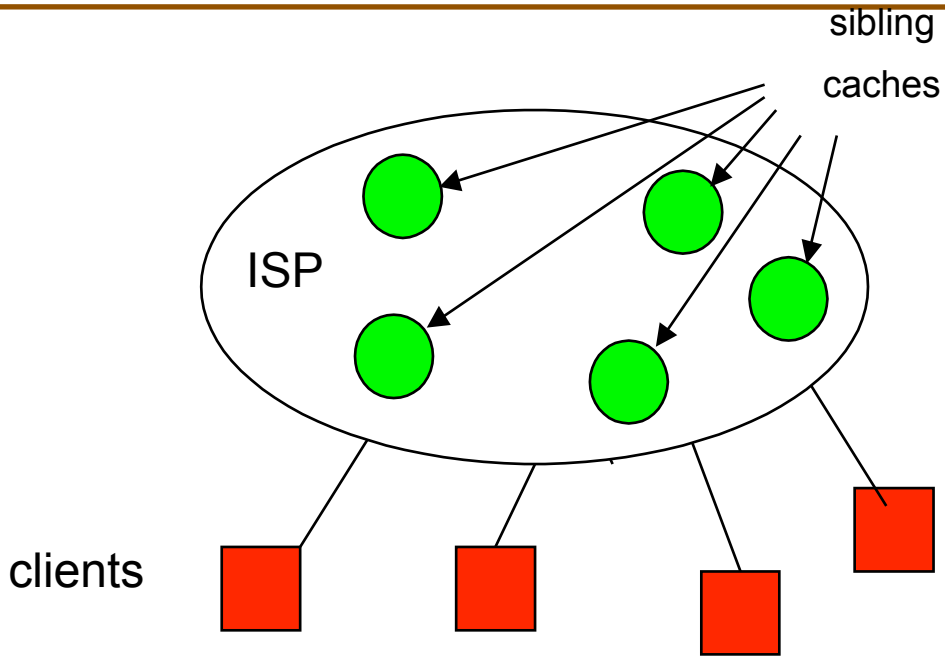


chain of caches: user configures browser to point to cache;
first cache points to second cache:



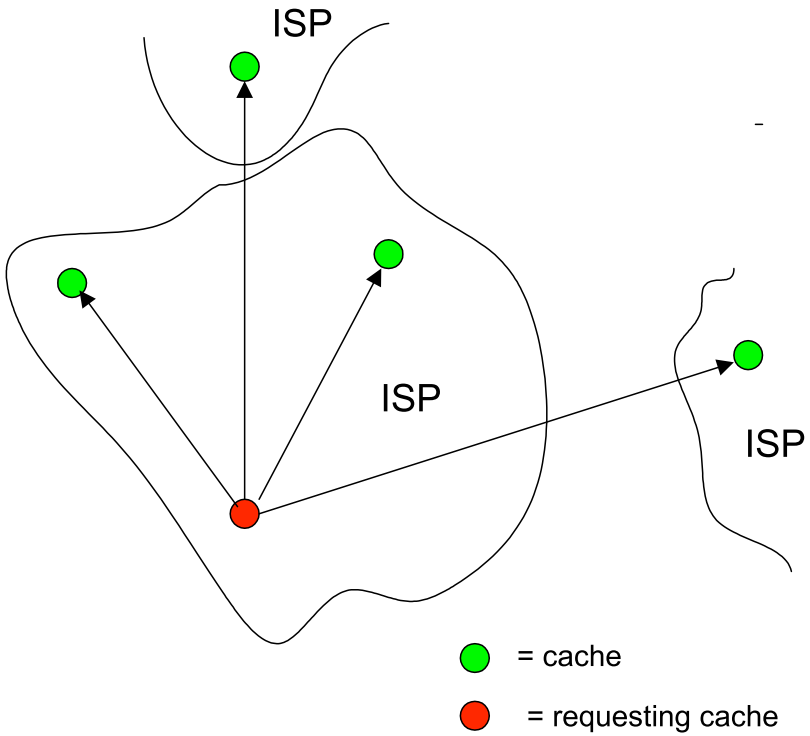
- Hierarchies use cache chaining
- All communication along chain can be over HTTP
- Leads to higher latency on cache misses
- Does not leverage cached content of peers

Cooperative Caching (1)



- Multiple sibling caches within a single ISP.
- One or more of the siblings could contain the requested object.
- Cooperation:
  - ICP (Internet Cache Protocol): siblings send messages to each other to find a copy of object.
  - Alternative: centralized control
- Can have cooperating sibling caches in each ISP in each tier of a hierarchy.

Cooperative Caching (2)



- With “pure” hierarchical caching, each client and cache points to at most one other cache.
- With cooperative caching, a cache can directly obtain an object from one of many neighboring caches. The neighboring caches can be:
  - in the ISP (sibling caches).
  - or outside the ISP

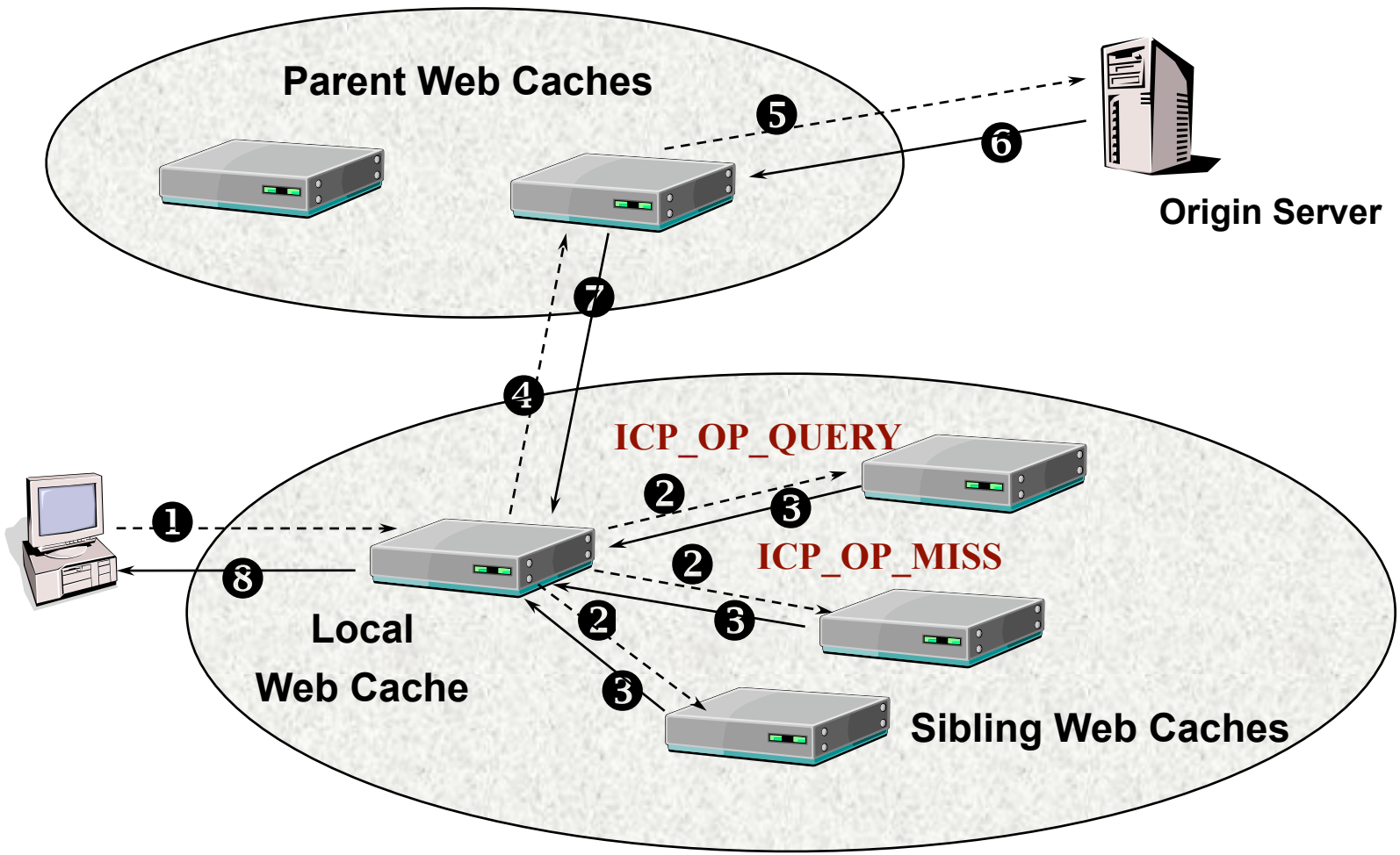
Overview of ICP (1)

- A simple protocol for querying neighboring caches (RFC 2186)
- Application-layer protocol implemented on top of UDP
- “Fast” way to determine:
  - which neighboring cache has the object;
  - the relative speed of the neighboring caches.
- When one cache queries another, there are three possible outcomes:
  - ICP hit message returned
  - ICP miss message returned
  - no response
    - proxy server down or overloaded, or
    - network connection down or overloaded

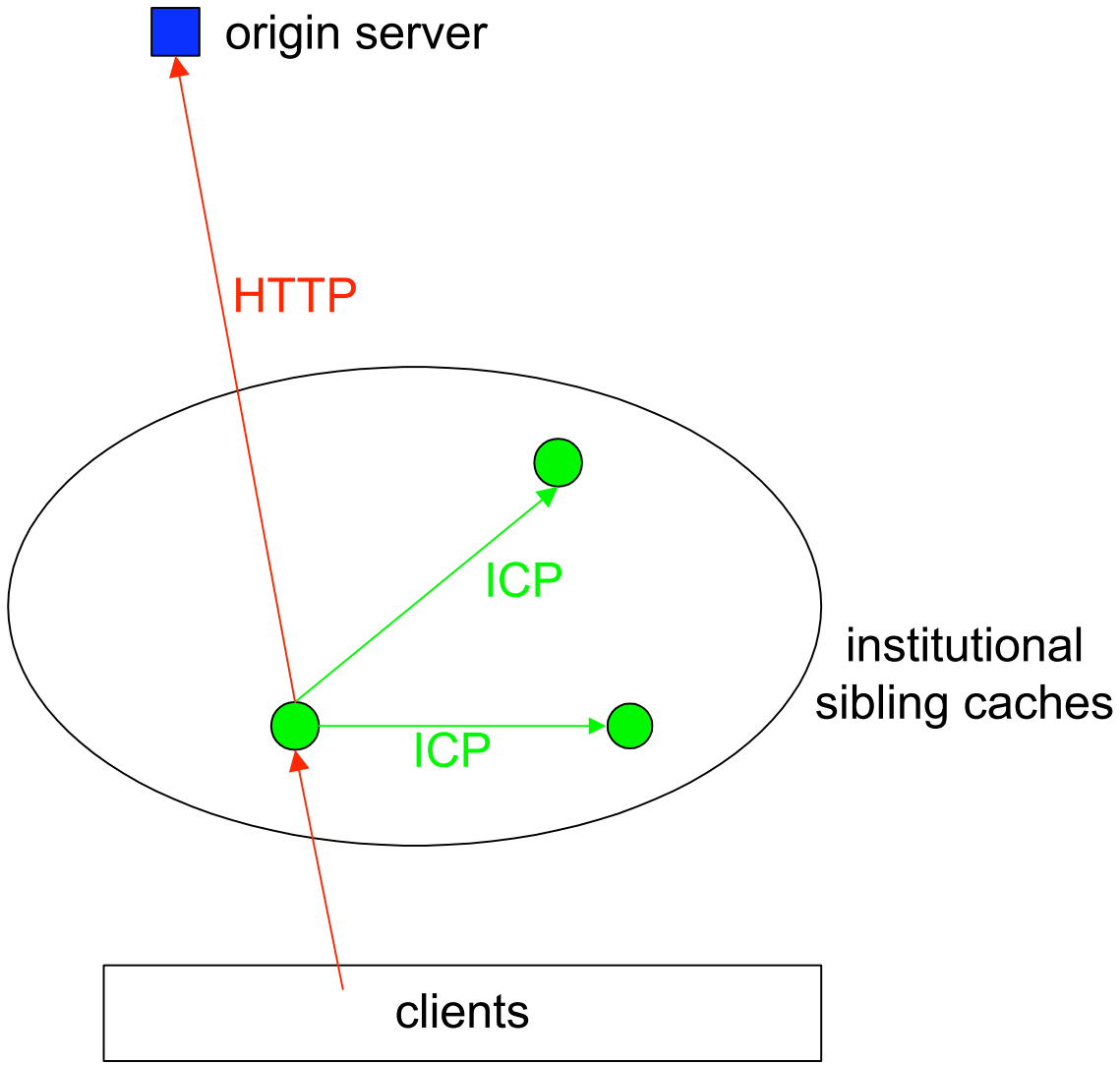
Overview of ICP (2)

- When ICP cache cannot fulfill a request from its own cache, then typically:
  - cache queries all neighbors with ICP
  - cache obtains object from first neighbor to respond with a hit
  - cache stores copy of object and forwards copy to requestor.
  - If there are no hits:
    - cache either forwards request to parent in hierarchy
    - or forwards request directly to origin server
- Cache waits up to two seconds for response to query.

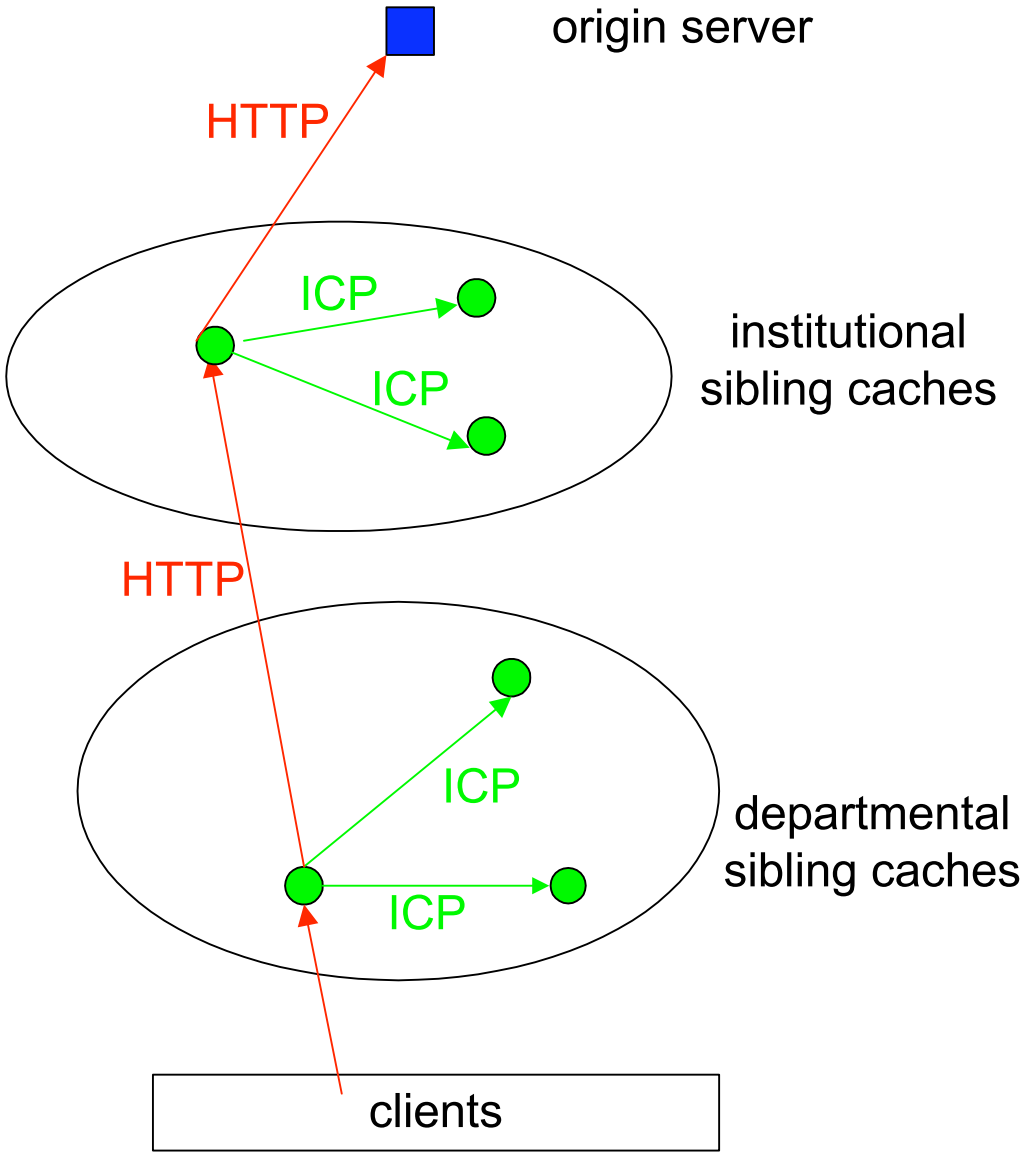
Using The ICP Protocol



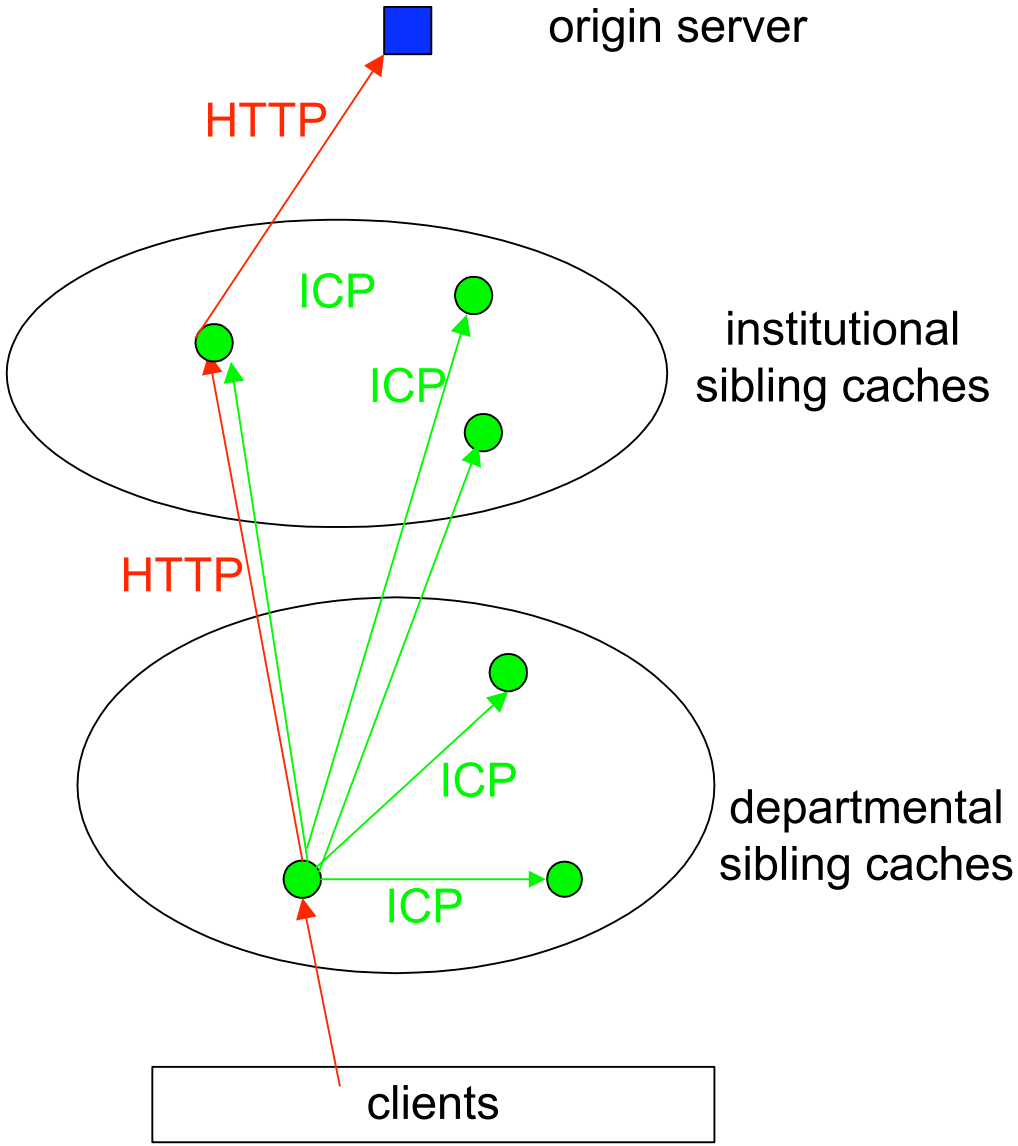
Institutional Example 1



Institutional Example 2

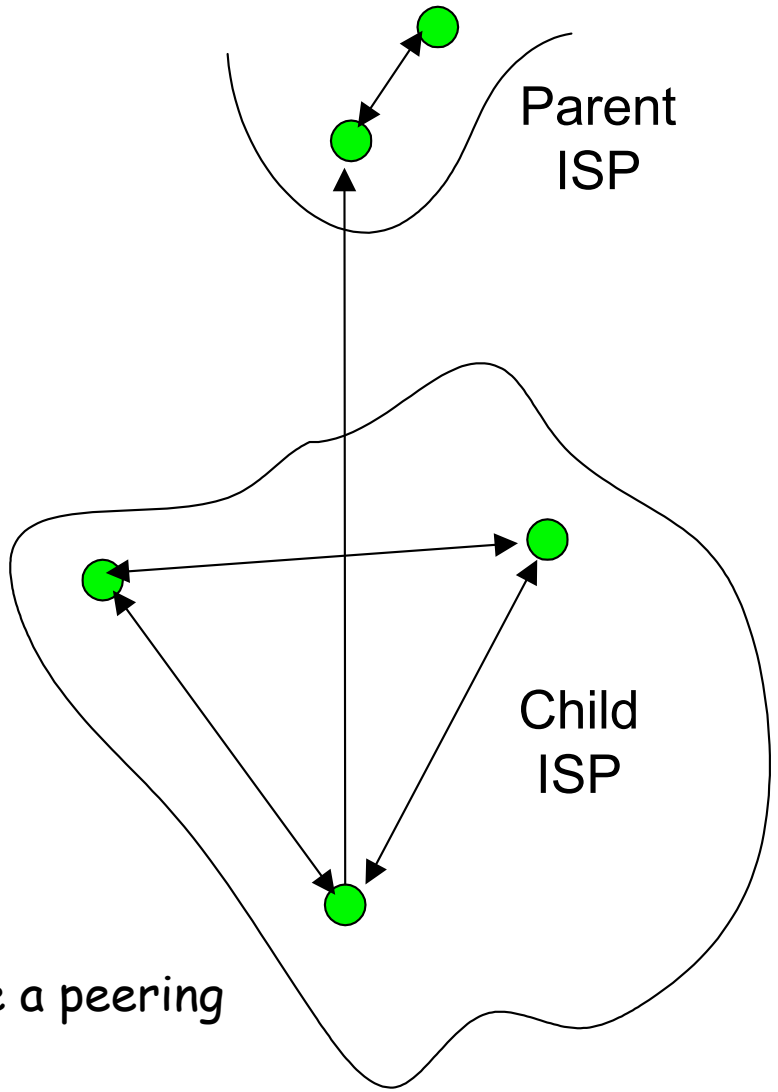


Institutional Example 3



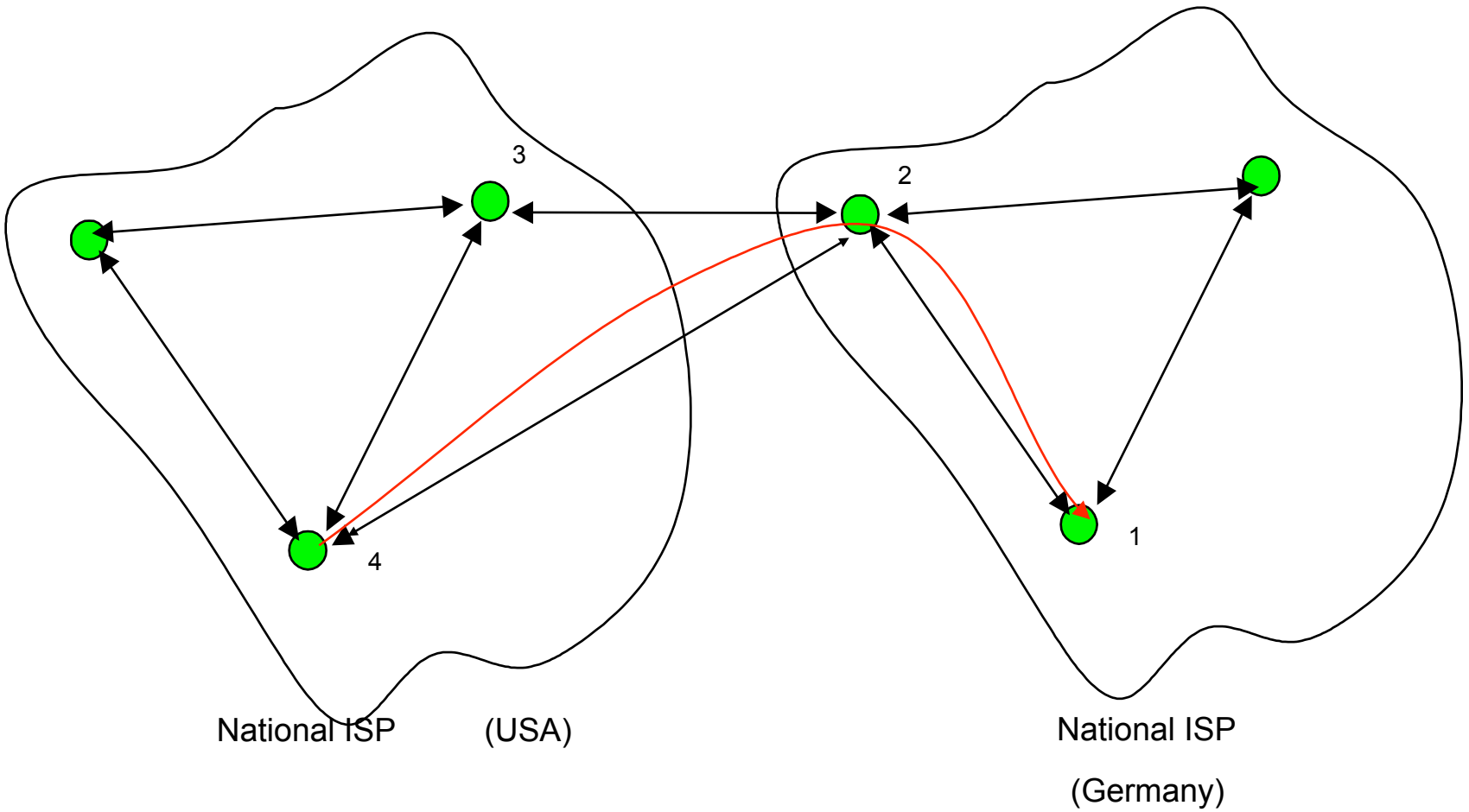
ISP Example 1 - Peering

- Siblings in child ISP query each other using ICP.
- Siblings in child ISP also query one or more caches in parent ISP using ICP.
- Siblings in parent ISP query each other using ICP.
- Typically, caches in parent ISP **do not** query caches in child ISP. (Instead, parent ISP climbs hierarchy or directly contacts origin server)



Child and parent ISP have a peering agreement.

ISP Example 2 – National Peers



Drawbacks of ICP

- Reactive nature of ICP results in increased access delays
- ICP message overhead
  - Whenever there is a miss, each sibling has to process an ICP request and response message (processing and bandwidth overhead)
- Popular objects get replicated in all the caches
  - Wastes disk and memory resources

Cache Digest Protocol

- Utilizes digests to help eliminate the need of peer queries
- Digest: a compressed summary of a cache's contents
  - A digest is usually made available by a cache as a URL
  - Digest exchange between peer caches uses HTTP/TCP
  - Example:

```
GET /cache_digest HTTP/1.1
Host: proxy.localdomain
```

- A cache keeps a digest for each of its peers

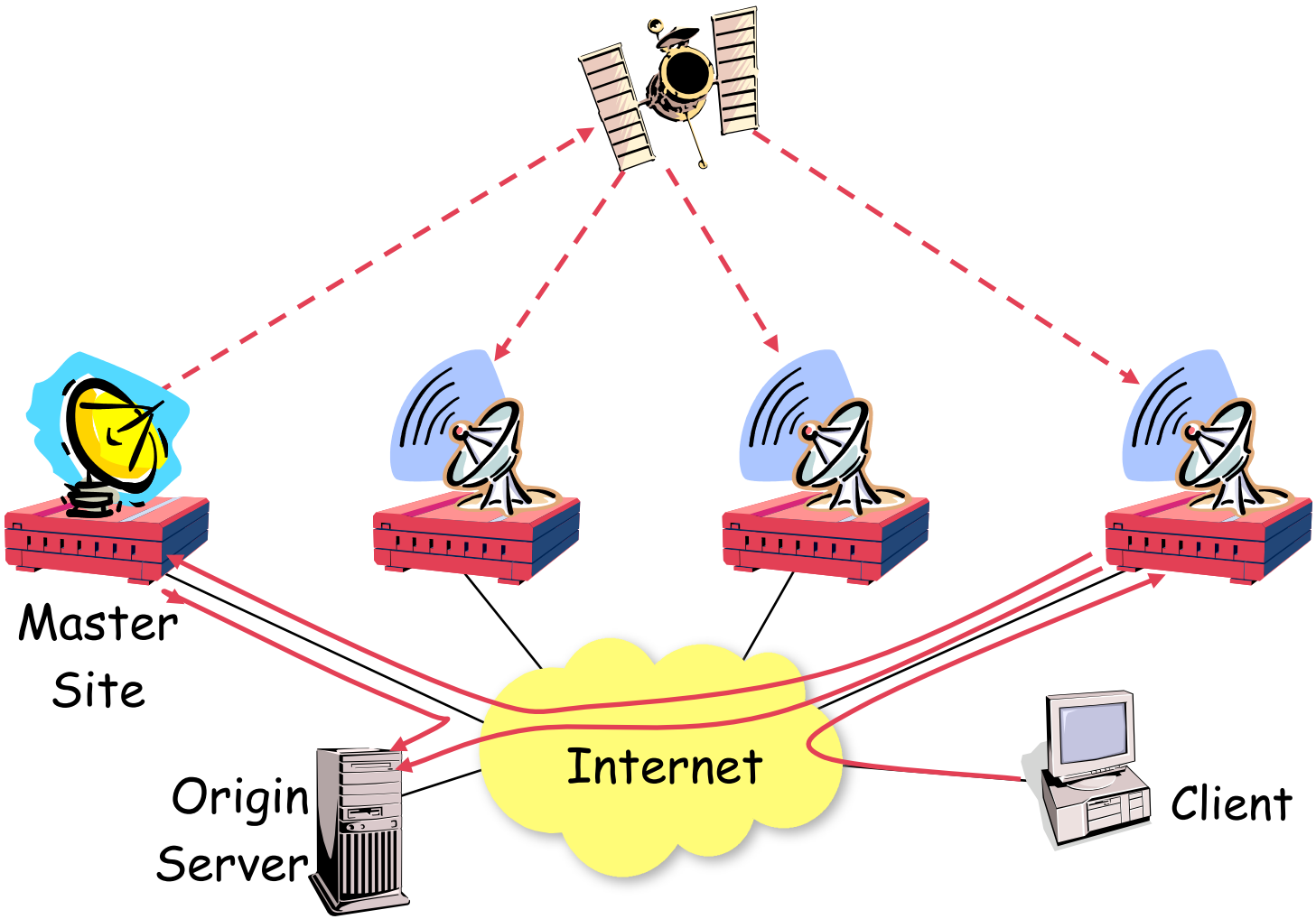
Cache Array Resolution Protocol (CARP)

- Allows multiple caching proxies to function as a single proxy.
- Works by generating a hash for each URL requested. A different hash is generated for each URL and by splitting the hash namespace into equal (or unequal parts, if uneven load is intended) the overall number of requests can be distributed to multiple servers.
- Each HTTP response that is cached by a CARP group is labeled with its URL and the identity of the cache storing it
- Participating caches use CARP to determine where a copy of a particular resource is located rather than having to query several peers
- Concerns:
  - Load balancing - it can potentially overwork some caching proxies while under-utilizing others
  - Reconfiguration if changes in participating members of the array occur

Satellite Technology: Paradigm Shift?

- **Each local ISP has a cache with:**
  - Internet connection
  - Huge storage capacity
  - Satellite dish for receiving
- **Master site has:**
  - Internet connection
  - Satellite transmitter
- **No intermediate regional or national caches.**
- **How it works: When there is a miss at some local cache:**
  - that local cache obtains document from origin server using HTTP.
  - local cache sends URL to master site.
  - master site obtains document from origin server using HTTP.
  - master site transmits document into satellite channel.
  - all local caches receive document and cache it.

Satellite Technology: Illustration



Satellite Technology: The Result

- The user populations at each of the local ISPs are aggregated together to form one huge user population.
  - The greater the user population, the greater the likelihood of repeated requests, the greater the hit rate.
  - High hit rates = low response times and less wasted bandwidth in the Internet
- Brings the Web to the edge of the network.
- And why not install the local caches in user homes?
  - Users keep their PCs on all the time and fill their disks from satellite.
  - A user applies filtering methods to reject uninteresting pages.
    - For example, only accept pages in the domains .com, .org and .edu .