The Problem

- Response Latency
- Network traffic
- Types of Caches
  - Browser
  - Proxy
  - Gateway

Caching Principle

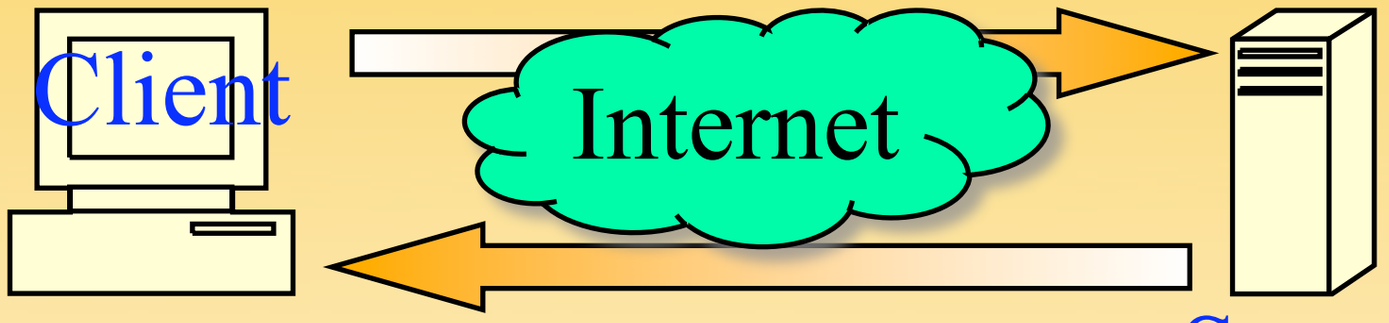
- Fast resources are scarce - e.g. storage close to clients (low propagation delay & fast links)
- Aim to locate commonly used objects in fast resource, other objects can remain in slower resources.
- Determining which objects are commonly accessed:
- Accesses are often correlated, and are said to exhibit "locality".
  - **Temporal locality**: Information accessed at one time is likely to be accessed again in the near future.
  - **Spatial locality**: Information accessed at one point is likely to be accessed also by adjacent points.

Network Content Caching

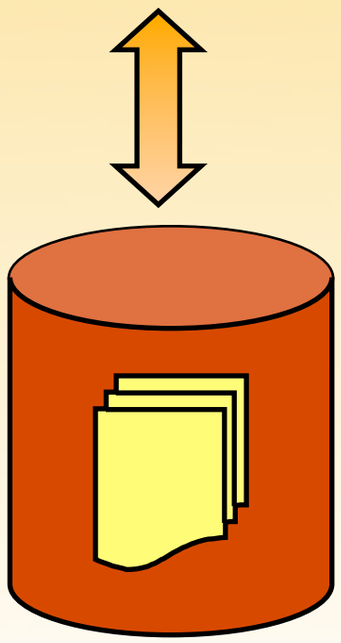
- Temporal locality: Content used once is likely to be used again, in the near future, by that node.
  - e.g. view Google logo one day; likely to view again the next day.
- Spatial locality: Content used by one user is likely to also be used by nearby users.
  - e.g. one student @ UM accesses slashdot.org => likely that others will also
- "In one study spanning more than a month, out of all the objects requested by individual users, on average close to 60 percent of those objects were requested more than once by the same user. … of the hits recorded in another caching study, up to 85 percent were the result of multiple users requesting the same object."

Browsers use private caches

GET /foo/index.html HTTP/1.1
Host: www.example.com

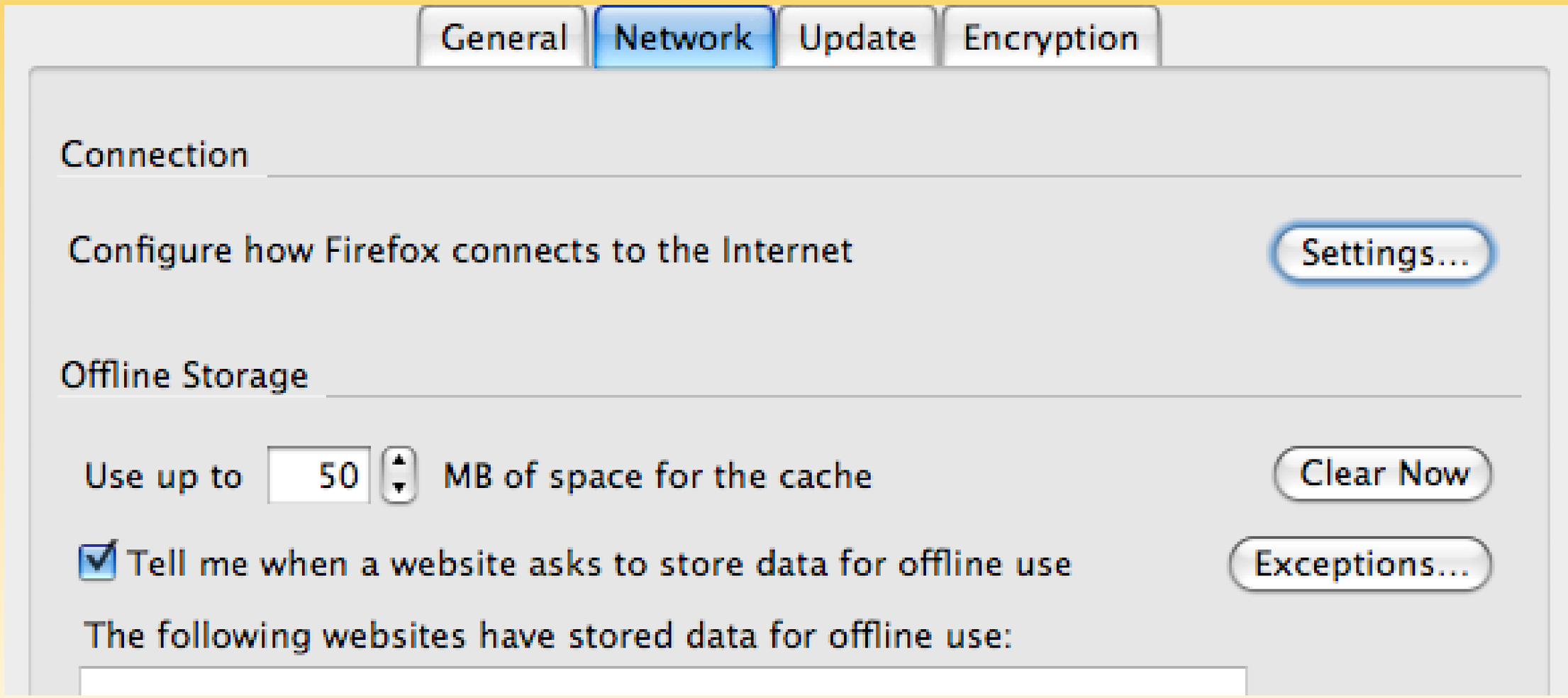


HTTP/1.1 200 OK
Last-Modified: Fri, 23 Jul 2004 01:52:37 GMT
Content-Length: 3688
Content-Type: text/html

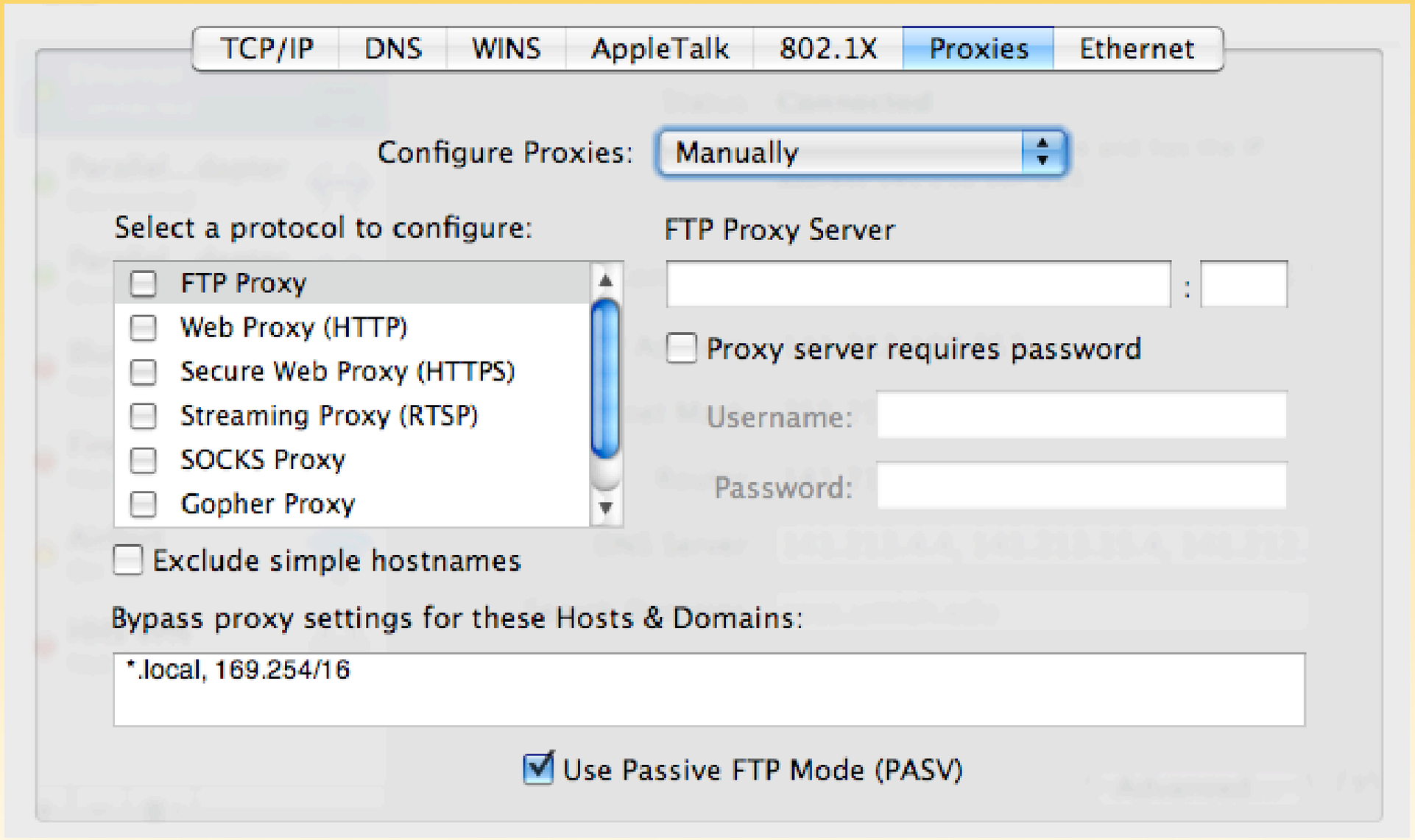


Client stores copy of **http://www.example.com/foo/index.html** on its hard disk with timestamp.

Local Cache Control

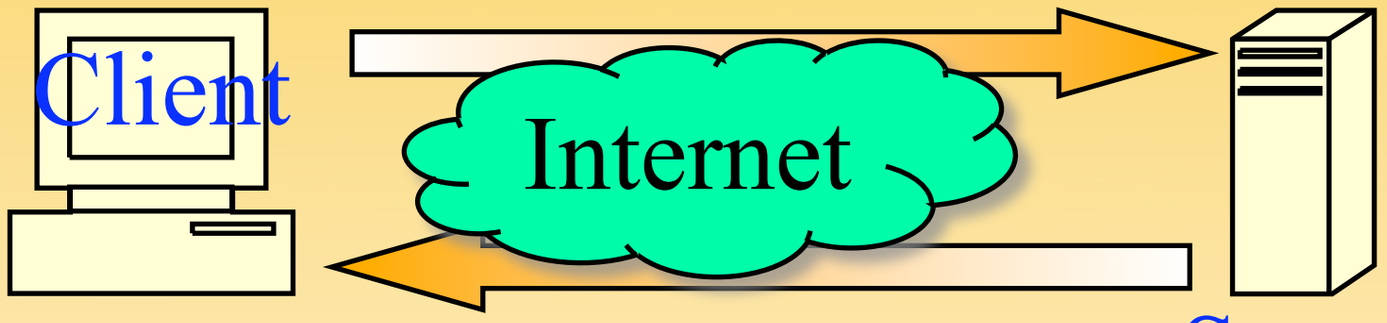


Local Proxy Control

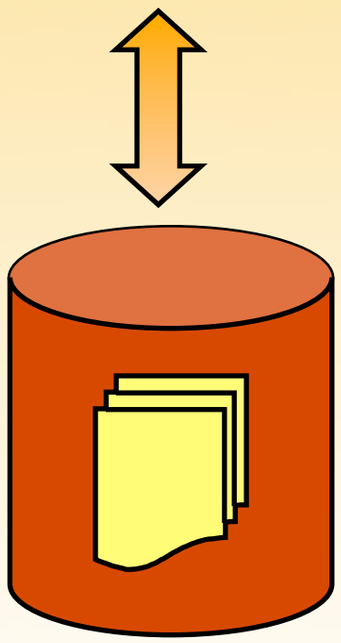


Revalidation (Conditional GET)

GET /foo/index.html HTTP/1.1
Host: www.example.com
If-Modified-Since: Fri, 23 Jul 2004 01:52:37 GMT



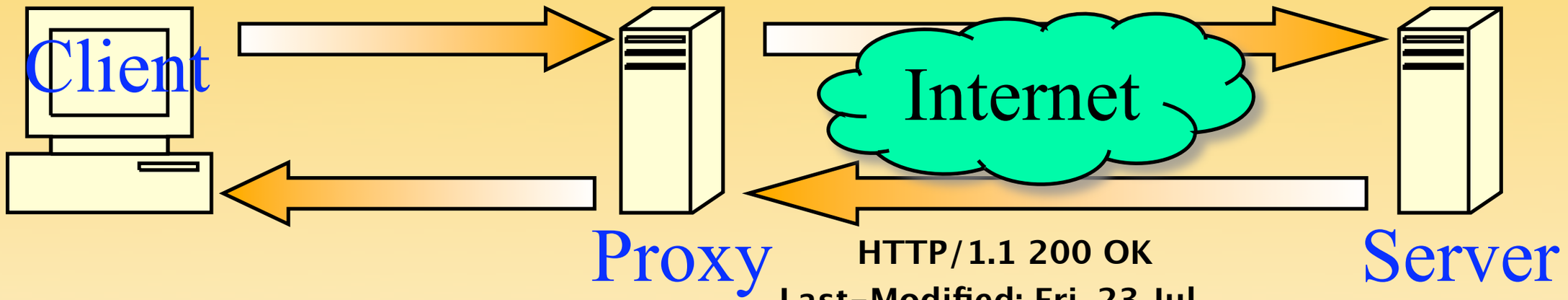
HTTP/1.1 304 Not Modified



Non-Caching Proxy

GET /foo/index.html HTTP/1.1
Host: www.example.com

GET /foo/index.html HTTP/1.1
Host: www.example.com



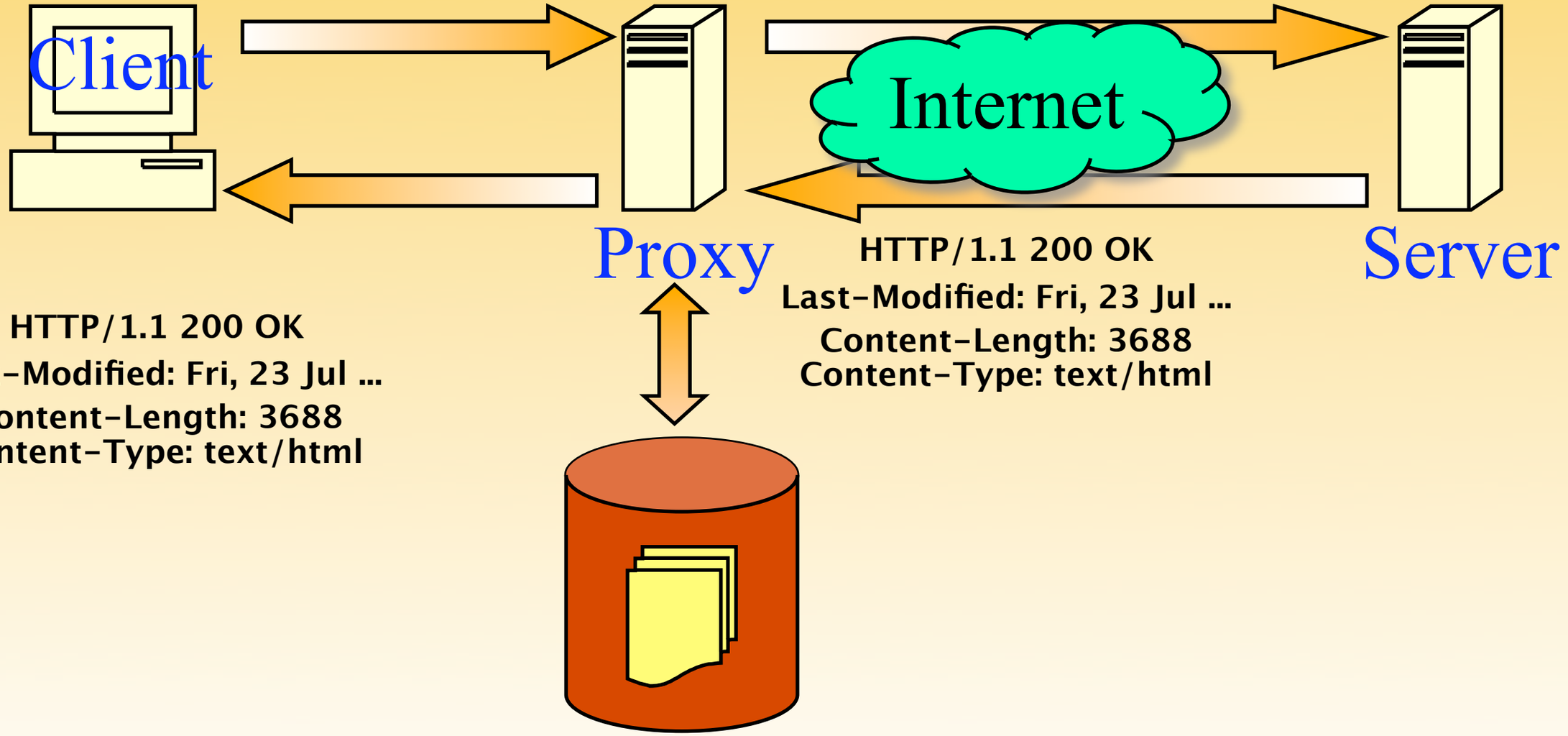
HTTP/1.1 200 OK
Last-Modified: Fri, 23 Jul ...
Content-Length: 3688
Content-Type: text/html

HTTP/1.1 200 OK
Last-Modified: Fri, 23 Jul ...
Content-Length: 3688
Content-Type: text/html

Proxy Cache Miss

GET /foo/index.html HTTP/1.1
Host: www.example.com

GET /foo/index.html HTTP/1.1
Host: www.example.com

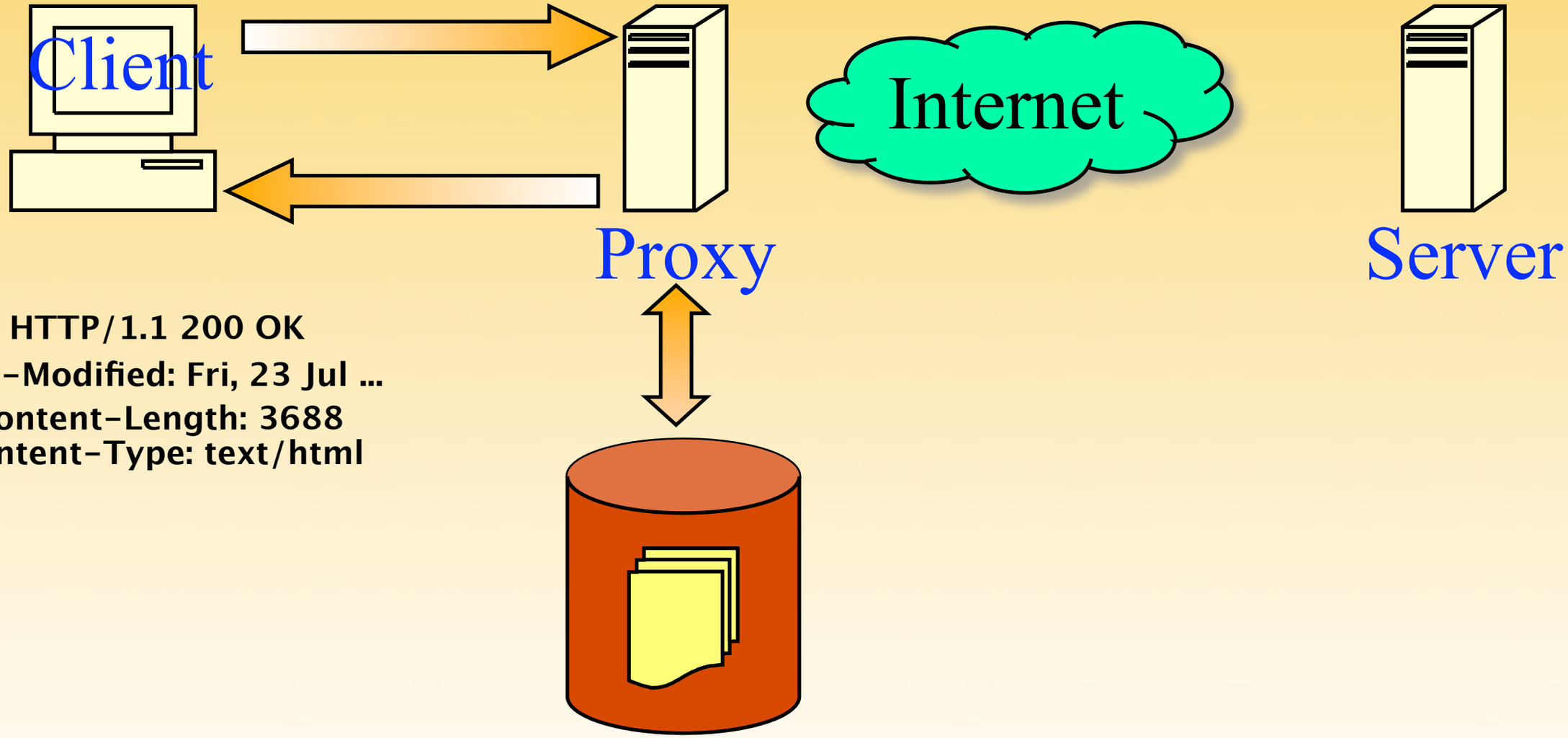


HTTP/1.1 200 OK
Last-Modified: Fri, 23 Jul ...
Content-Length: 3688
Content-Type: text/html

HTTP/1.1 200 OK
Last-Modified: Fri, 23 Jul ...
Content-Length: 3688
Content-Type: text/html

Proxy Cache Hit

GET /foo/index.html HTTP/1.1
Host: www.example.com

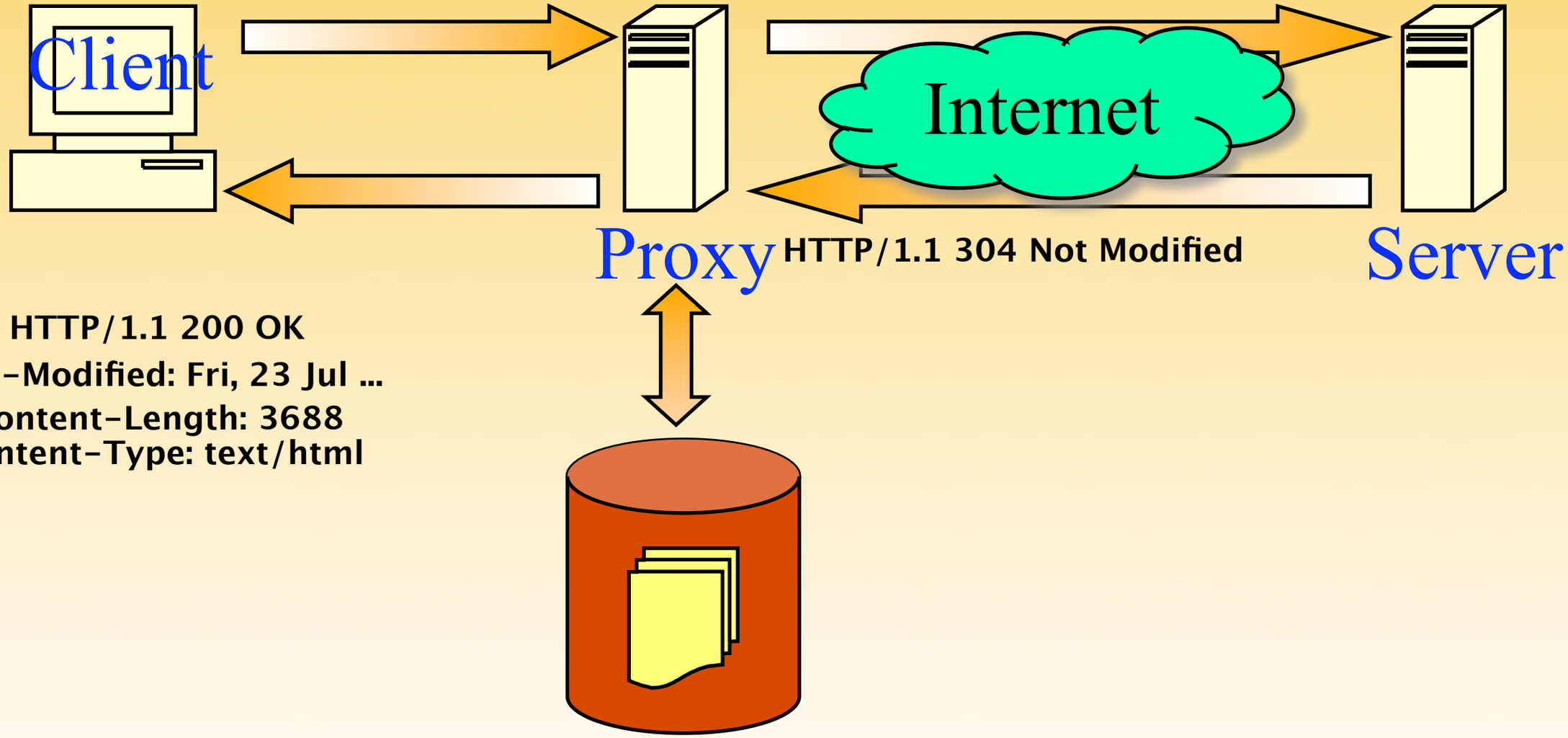


HTTP/1.1 200 OK
Last-Modified: Fri, 23 Jul ...
Content-Length: 3688
Content-Type: text/html

Proxy Cache Revalidation

GET /foo/index.html HTTP/1.1
Host: www.example.com

GET /foo/index.html HTTP/1.1
Host: www.example.com
If-Modified-Since: Fri, 23 Jul ...



HTTP/1.1 200 OK
Last-Modified: Fri, 23 Jul ...
Content-Length: 3688
Content-Type: text/html

Expires HTTP Header

- Tells all caches how long the associated representation is fresh for
- Set on web server
  - Absolute time (based on either last access or last modification time)
- Good for static images
- Also good for dynamic content
- Must be an HTTP Date; otherwise considered past
- Important for clocks on cache and server to be sync'd

Cache-Control HTTP Headers (1)

- **max-age=[seconds]** — specifies the maximum amount of time that an representation will be considered fresh. Similar to Expires, this directive is relative to the time of the request, rather than absolute. [seconds] is the number of seconds from the time of the request you wish the representation to be fresh for.
- **s-maxage=[seconds]** — similar to max-age, except that it only applies to shared (e.g., proxy) caches.
- **public** — marks authenticated responses as cacheable; normally, if HTTP authentication is required, responses are automatically uncacheable (for shared caches).

Cache-Control HTTP Headers (2)

- **no-cache** — forces caches to submit the request to the origin server for validation before releasing a cached copy, every time. This is useful to assure that authentication is respected (in combination with public), or to maintain rigid freshness, without sacrificing all of the benefits of caching.
- **no-store** — instructs caches not to keep a copy of the representation under any conditions.

Cache-Control HTTP Headers (3)

- **must-revalidate** — tells caches that they must obey any freshness information you give them about a representation. HTTP allows caches to serve stale representations under special conditions; by specifying this header, you're telling the cache that you want it to strictly follow your rules.
- **proxy-revalidate** — similar to must-revalidate, except that it only applies to proxy caches. For example:
  - Cache-Control: max-age=3600, must-revalidate

Assumptions about content types

Rate of change once published

Frequently *Occasionally* *Rarely/Never*

HTML

CSS
JavaScript

Images
Flash
PDF

Dynamic Content

Personalized

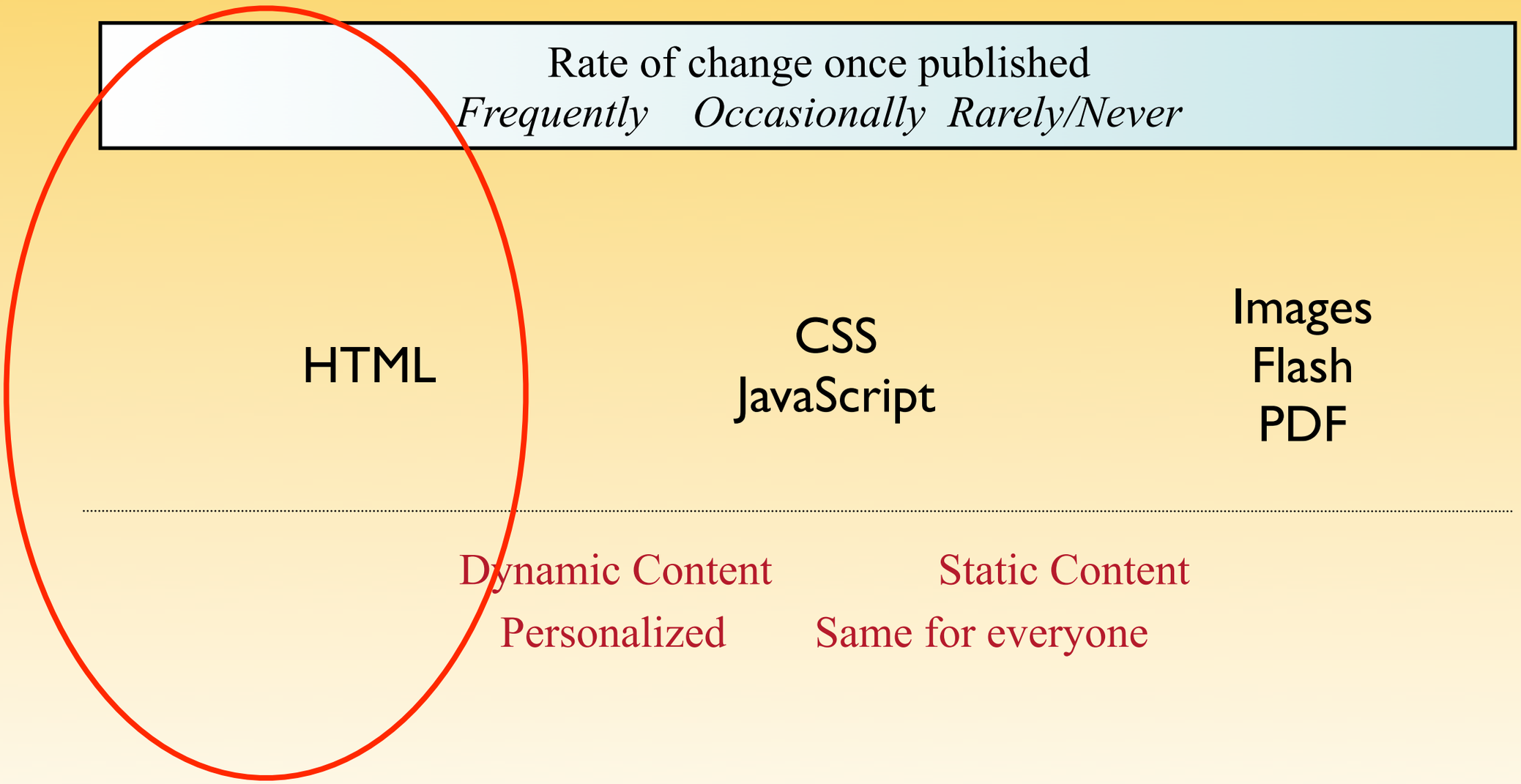
Static Content

Same for everyone

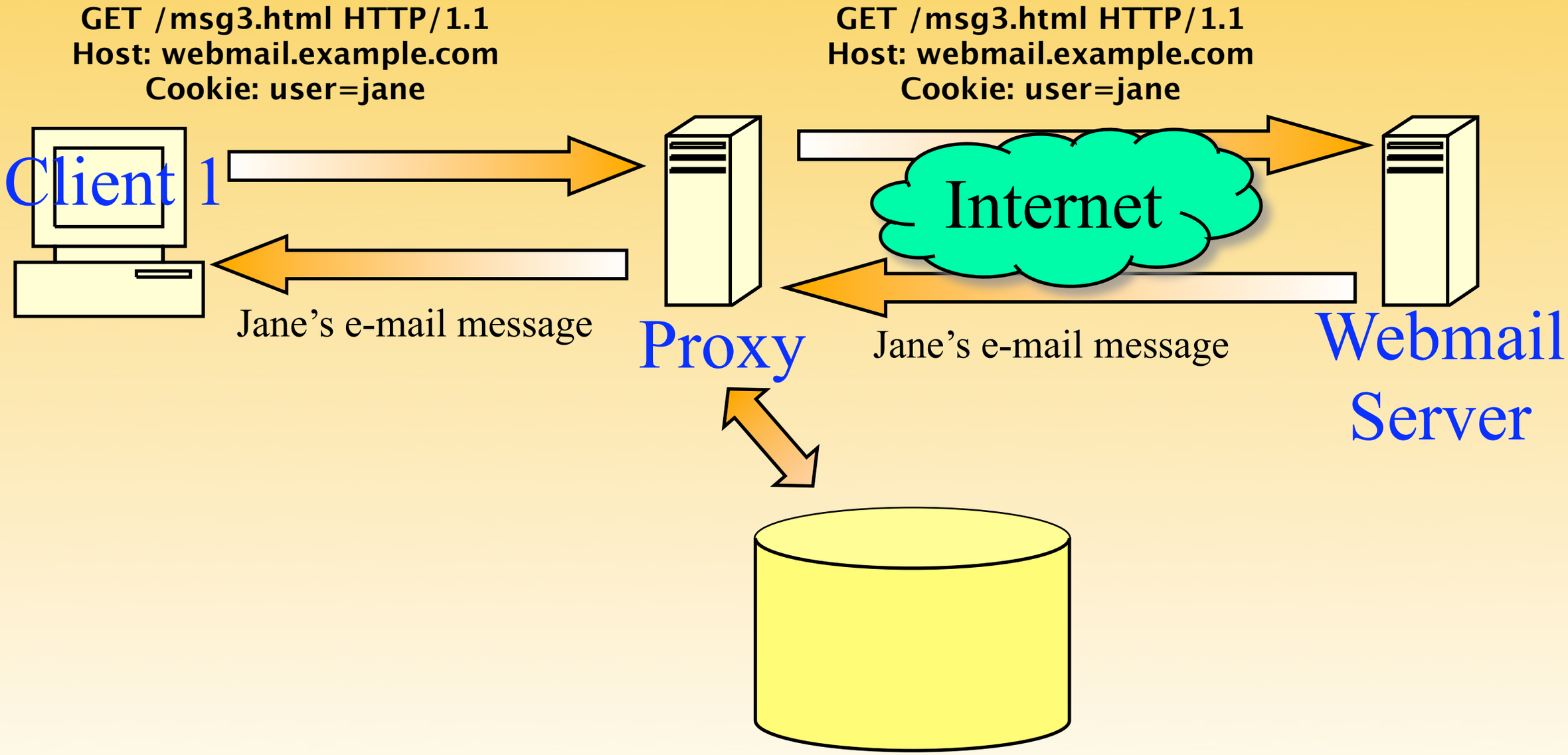
Top 5 techniques for publishers

1. Use “Cache-Control: private” for personalized content
2. Implement “Images Never Expire” policy
3. Use a cookie-free TLD for static content
4. Use Apache defaults for CSS & JavaScript
5. Use random strings in URL for accurate hit metering or very sensitive content

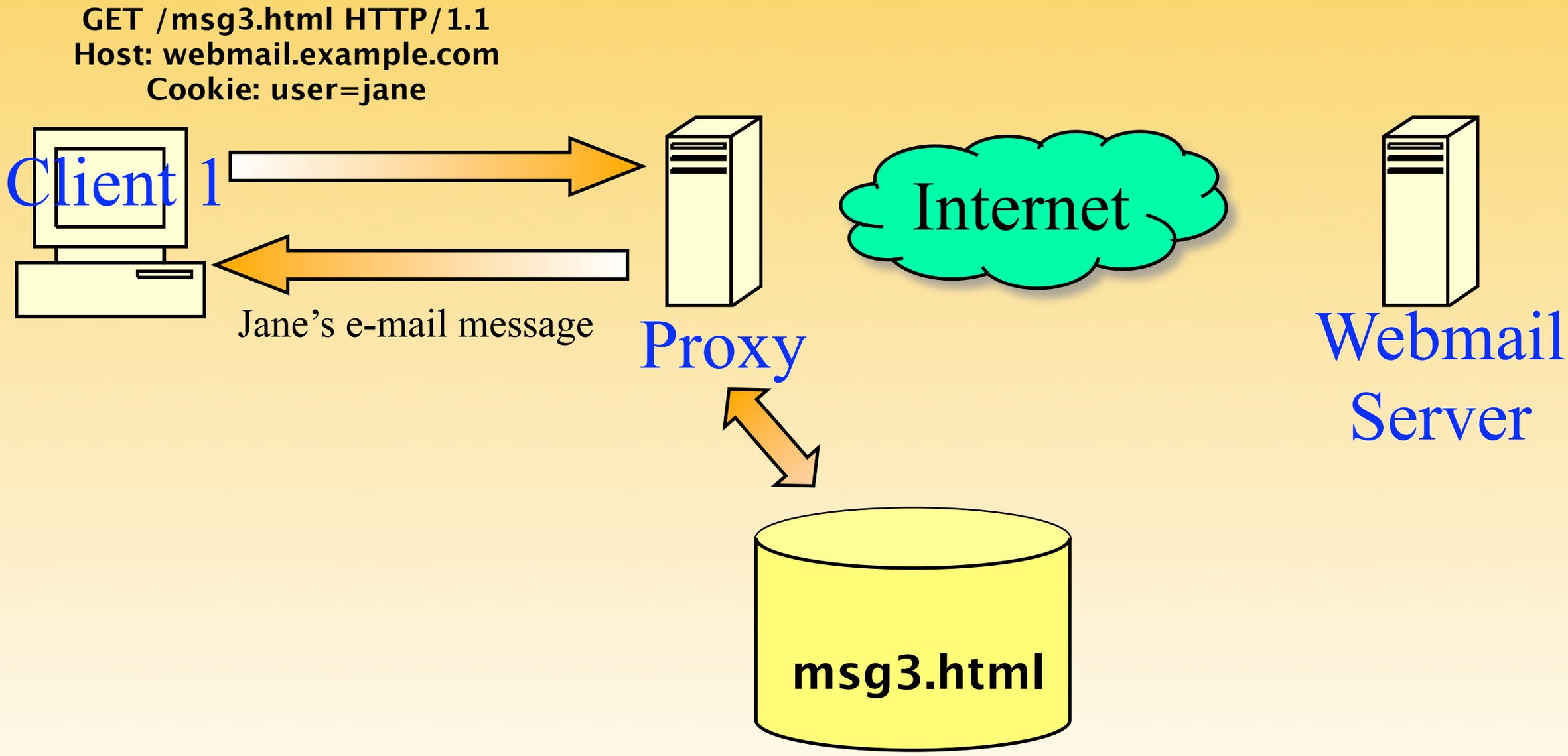
1. Use "Cache-Control:



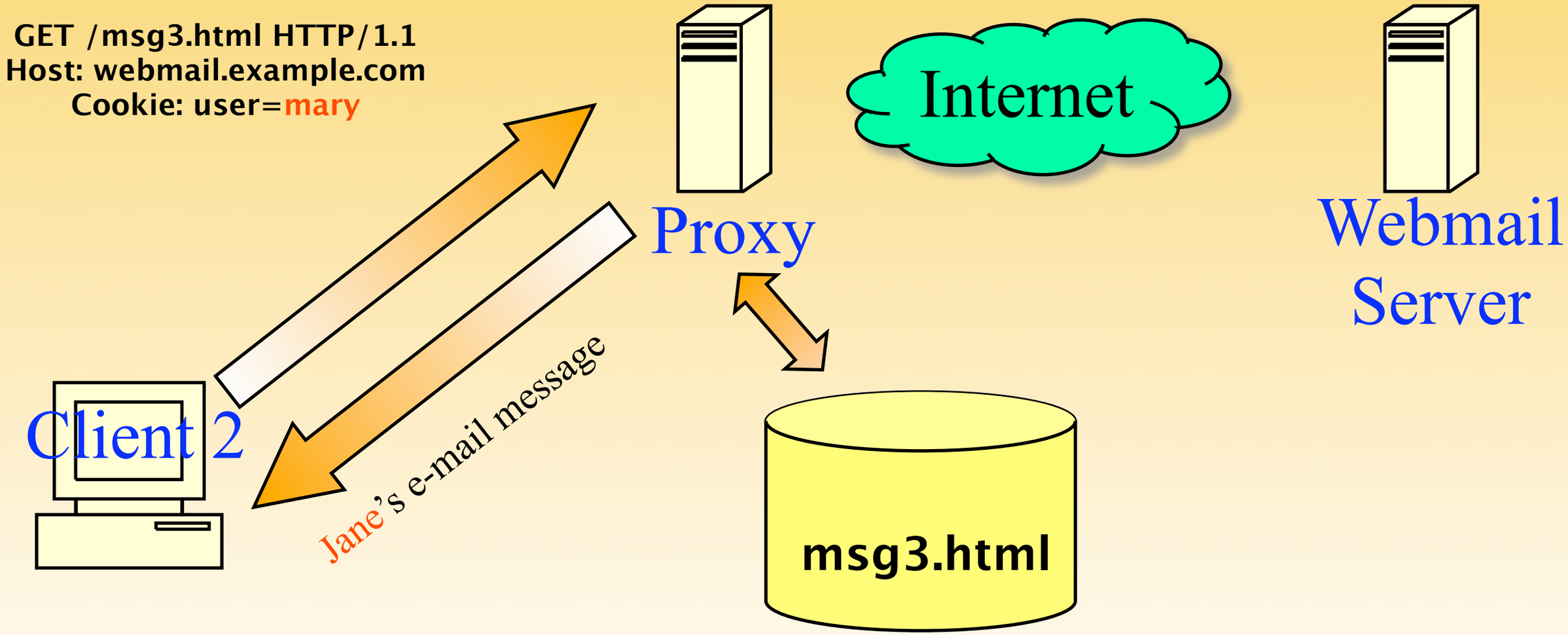
Shared caching gone awry



Shared caching gone awry



Shared caching gone awry



What's cacheable?

- HTTP/1.1 allows caching anything by default
  - Unless explicit Cache-Control header
- In practice, most caches avoid anything with
  - Cache-Control/Pragma header
  - Cookie/Set-Cookie headers
  - WWW-Authenticate/Authorization header
  - POST/PUT method
  - 302/307 status code

Cache-Control: private

- Shared caches bad for shared content
  - Mary shouldn't be able to read Jane's mail
- Private caches perfectly OK
  - Speed up web browsing experience
- Avoid personalization leakage with single line in httpd.conf or .htaccess

Header set Cache-Control private

2. “Images Never Expire” policy

Rate of change once published

Frequently *Occasionally* *Rarely/Never*

HTML

CSS
JavaScript

Images
Flash
PDF

Dynamic Content
Personalized

Static Content
Same for everyone

“Images Never Expire”

- Encourage caching of icons & logos
  - Forever ≈ 10 years in Internet biz
- Must change URL when you change img
  - http://us.yimg.com/i/new.gif
  - http://us.yimg.com/i/new2.gif
- Tradeoff
  - More difficult for designers
  - Bandwidth savings, faster user experience

3. Cookie-free static

Rate of change once published
Frequently *Occasionally* *Rarely/Never*

HTML

CSS
JavaScript

Images
Flash
PDF

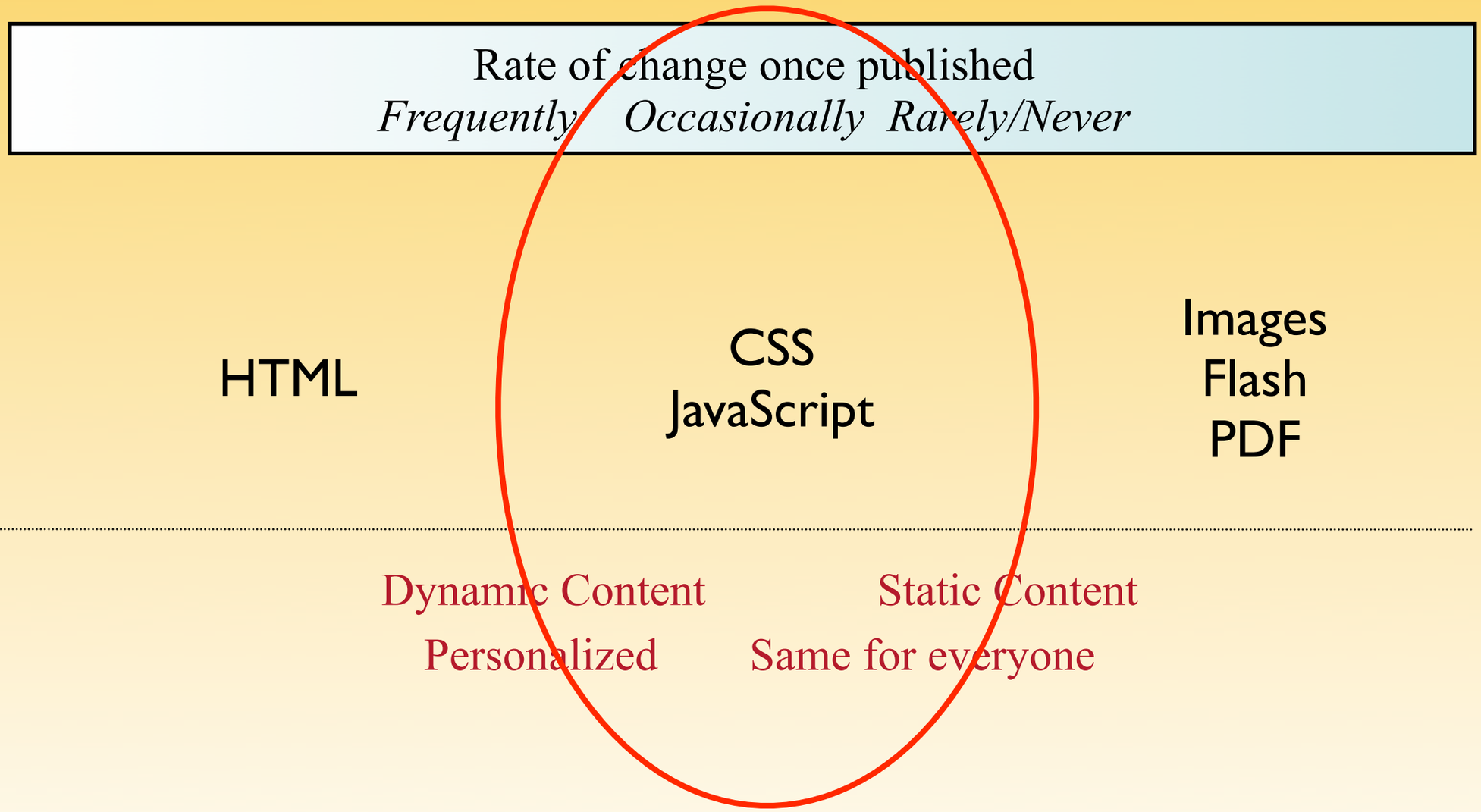
Dynamic Content
Personalized

Static Content
Same for everyone

Use a cookie-free Top Level

- For maximum efficiency use two domains
  - –www.example.com for HTML
  - –img.example.net for images
- Some proxies won't cache Cookie reqs
  - But: multimedia is never personalized
  - Cookies irrelevant for images

4. Apache defaults for



Revalidation works pretty

- Revalidation default behavior for static content
  - Browser sends If-Modified-Since request
  - Server replies with short 304 Not Modified
  - No fancy Apache config needed
- Use if you can't predict when content will change
  - Page designers can change immediately
  - No renaming necessary
- Cost: extra HTTP transaction for 304
  - Small with Keep-Alive, but large sites disable

5. Random URL strings for

Rate of change once published

Frequently *Occasionally* *Rarely/Never*

HTML

CSS
JavaScript

Images
Flash
PDF

Dynamic Content
Personalized

Static Content
Same for everyone

Accurate advertising

- If you trust proxies
  - Send Cache-Control: must-revalidate
  - Count 304 Not Modified log entries as hits
- If you don't
  - Ask client to fetch uncacheable image URL
  - Return 302 to highly cacheable image file
  - Count 302s as hits
  - Don't bother to look at cacheable server log

Defeating proxies: turning public caches into private caches

- Use distinct tokens in URL
  - No two users use same token
  - Defeats shared proxy caches
  - Works well with private caches
- Doesn't break the back button
- May break visited-link highlighting
  - e.g. JavaScript timestamps/random numbers
  - Every link is blue, no purple

Breaking the Back button

- When users click browser Back button
  - Expect to go back one page instantly
  - Private cache enables this behavior
- Aggressive cache-busting breaks Back button
  - Server sends `Pragma: no-cache` or `Expires` in past
  - Browser must re-visit server to re-fetch page
  - Hitting network much slower than hitting disk
- Use very sparingly
  - Compromising user experience is A Bad Thing

Review: Top 5 techniques

1. Use "Cache-Control: private" for personalized content
2. Implement "Images Never Expire" policy
3. Use a cookie-free TLD for static content
4. Use Apache defaults for CSS & JavaScript
5. Use random strings in URL for accurate hit metering or very sensitive content

Review: encouraging caching

- Send explicit Cache-Control or Expires
- Generate "static content" headers
  - Last-Modified, ETag
  - Content-Length
- Avoid "cgi-bin", ".cgi" or "?" in URLs
  - Some proxies (e.g. Squid) won't cache
  - Workaround: use PATH_INFO instead

Review: discouraging caching

- Use POST instead of GET
- Use random strings and "?" char in URL
- Omit Content-Length & Last-Modified
- Send explicit headers on response
  - Breaks the back button
  - Only as a last resort

Cache-Control: max-age=0,no-cache,no-store

Expires: Thu, 01 Jan 1970 12:34:56 GMT

Pragma: no-cache