How to Write Blazing Fast Web Apps with Ruby on Rails

Presentation by
Jesse Hallett
@hallettj
Portland Code Camp 2009

Cache, cache, cache, and more cache.

Cached content is faster than rendered content.

Two kinds of caches:

- Fragment cache - e.g. memcached
- Page cache - a.k.a. HTTP cache

Fragment Caching

- Cache any part of a page
- Key / value store of arbitrary content

Pros

- More flexible than page caching
- Allows mixing dynamic and cached content

Cons

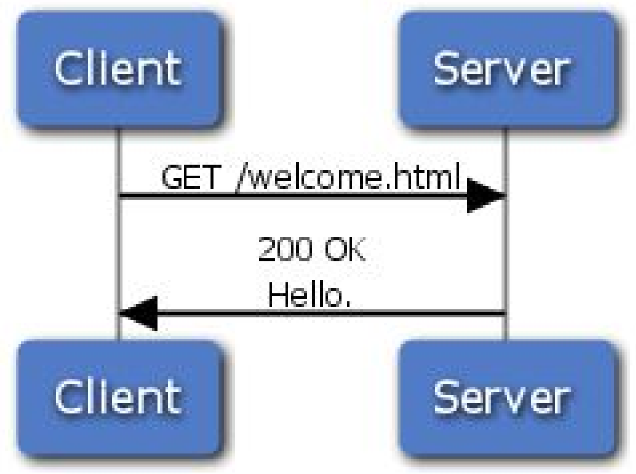
- Slower than page caching
- Requires deeper integration with your app

HTTP Caching

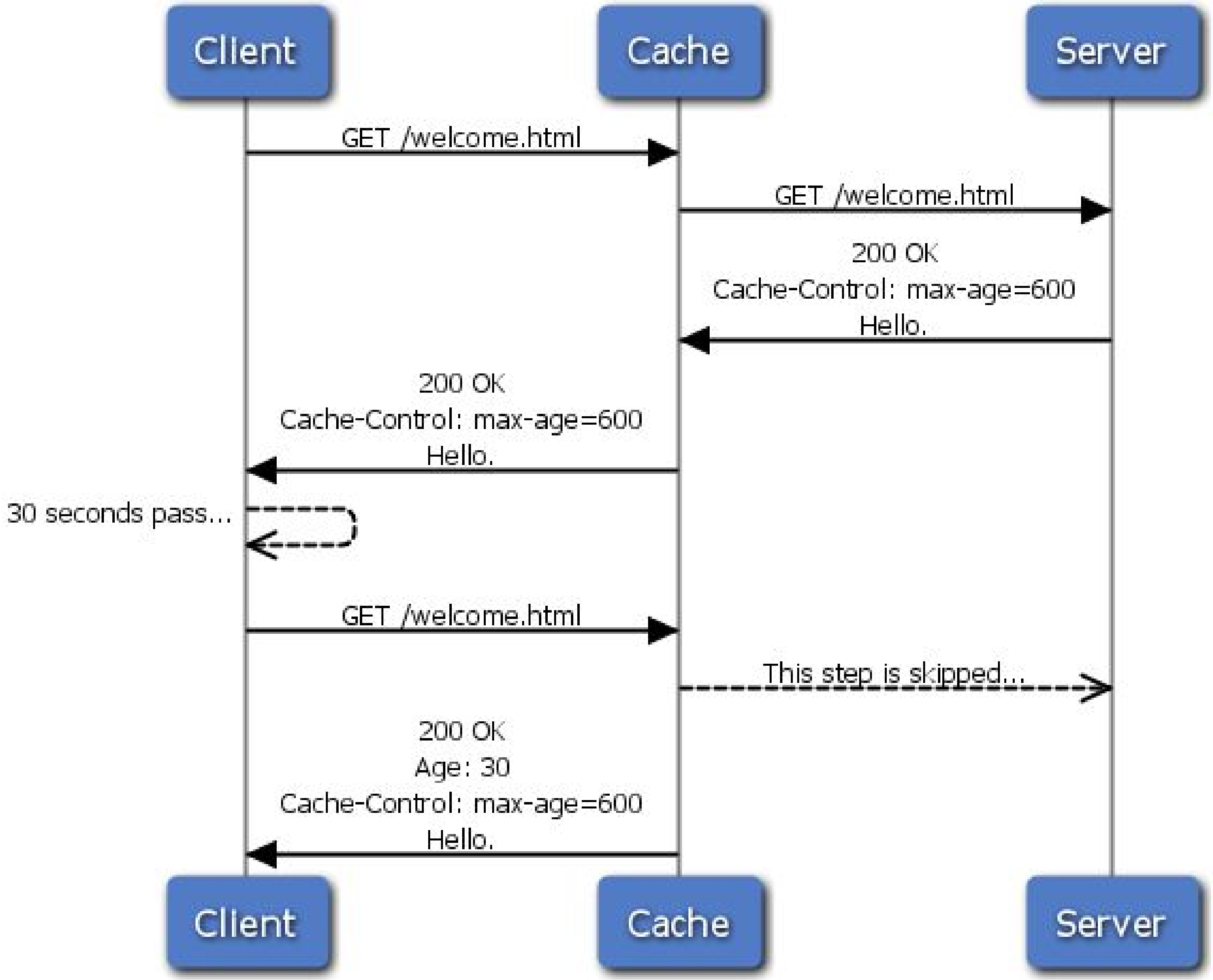
- External to your application
- Standardized

Pros
- Serving cached content is really fast
- Works on the client side for maximum fastness
- Easy to set up
  - `headers['Cache-Control'] = 'max-age: 600'`

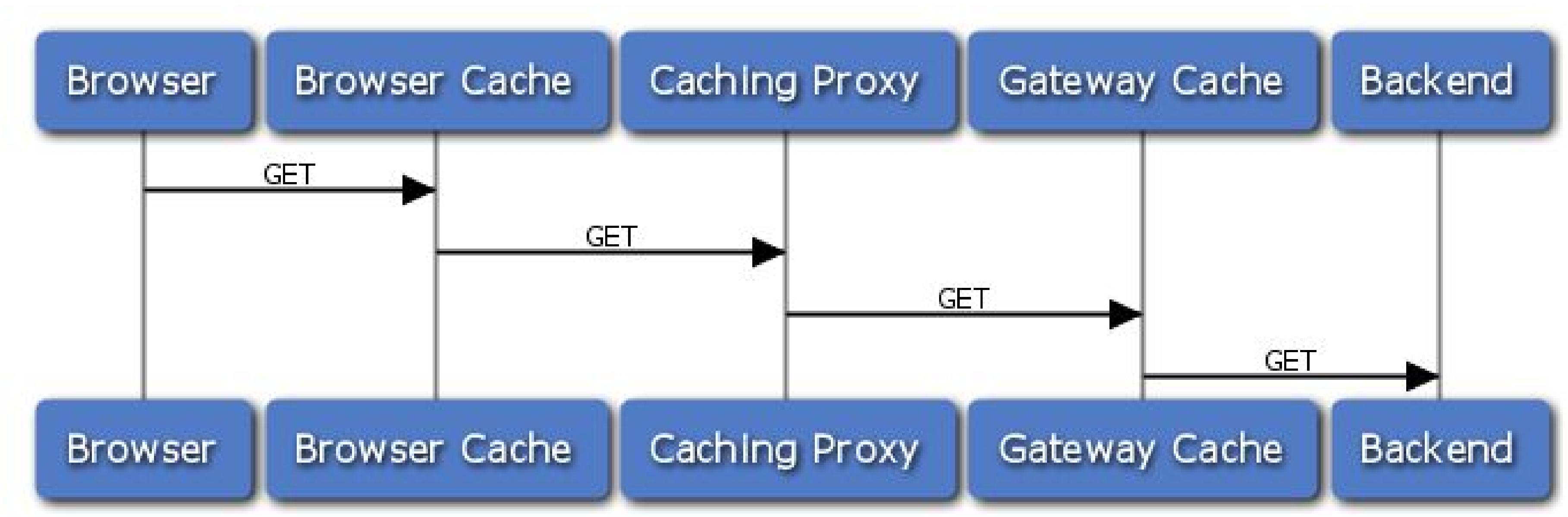
Cons
- Cannot mix dynamic content with cached content
- Not all cache implementations are RFC compliant
- Can be negated by, e.g., cookies



A very basic request/response cycle



The same request with caching



Caches can hide anywhere.

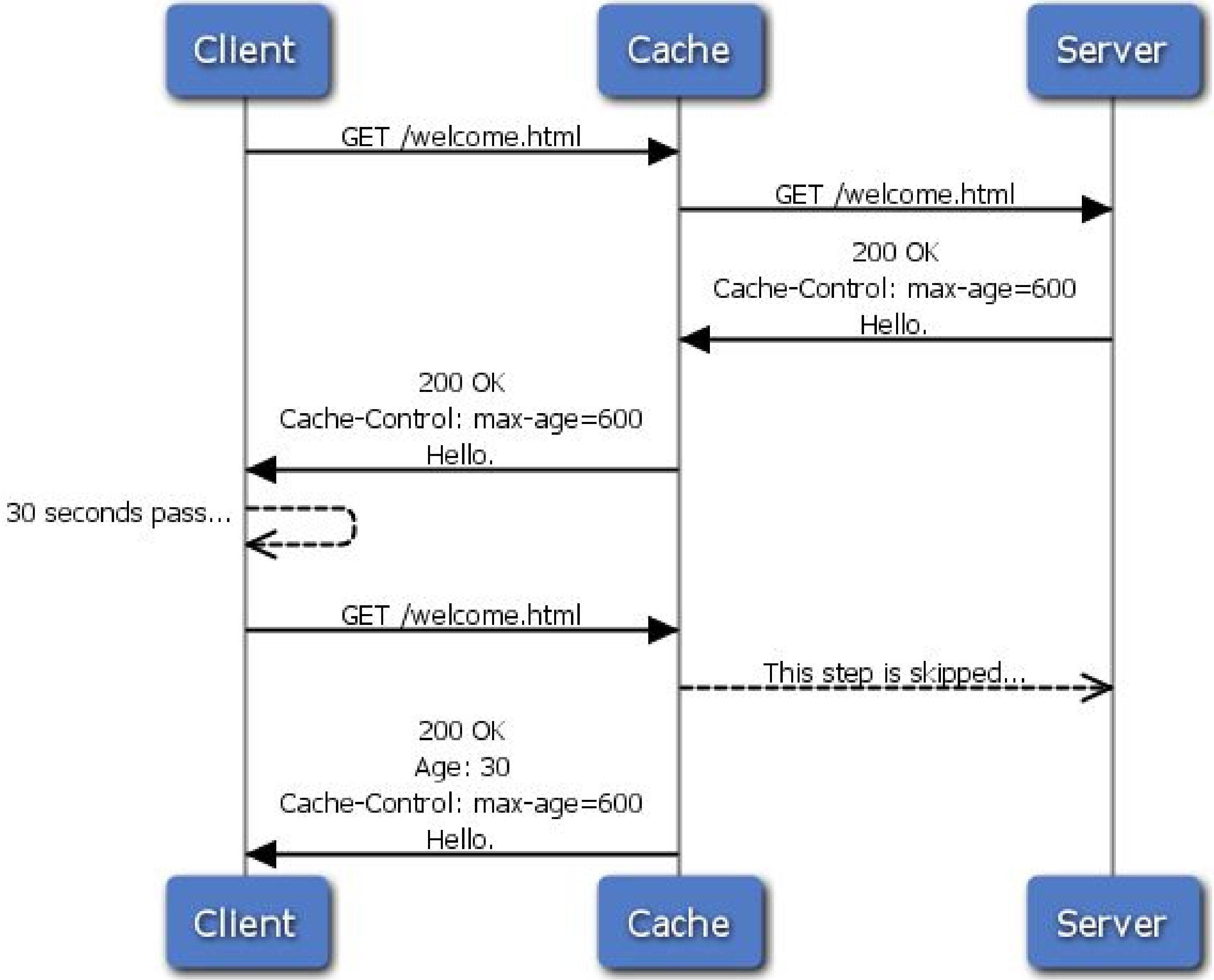
Gateway Caches

Cache that sits on your server in front of your application.

- The same cache applied to all clients
- Client's first page load is as fast as the second
- A cache you can rely on
  - `/query?are-you-caching-this`

Implementations

- Varnish - http://varnish-cache.com/
- Squid - http://www.squid-cache.org/
- rack-cache - http://tomayko.com/src/rack-cache/



Now imagine requests are coming from different clients.

Cache Headers

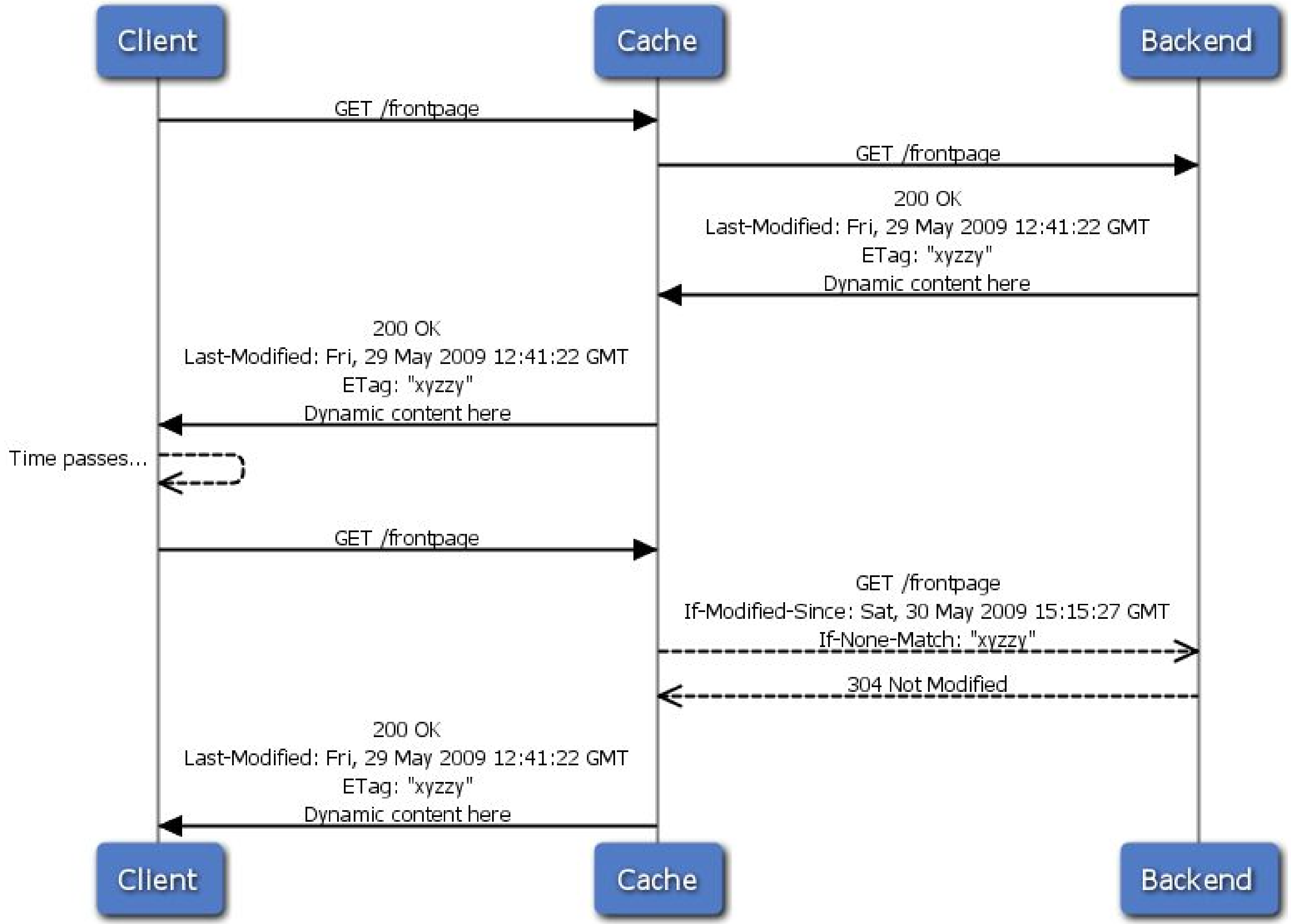
Expiration

- Cache-Control
- Expires

Validation

- Last-Modified / If-Modified-Since
- ETag / If-None-Match

Expiration and validation can be combined. Expiration takes precedence.

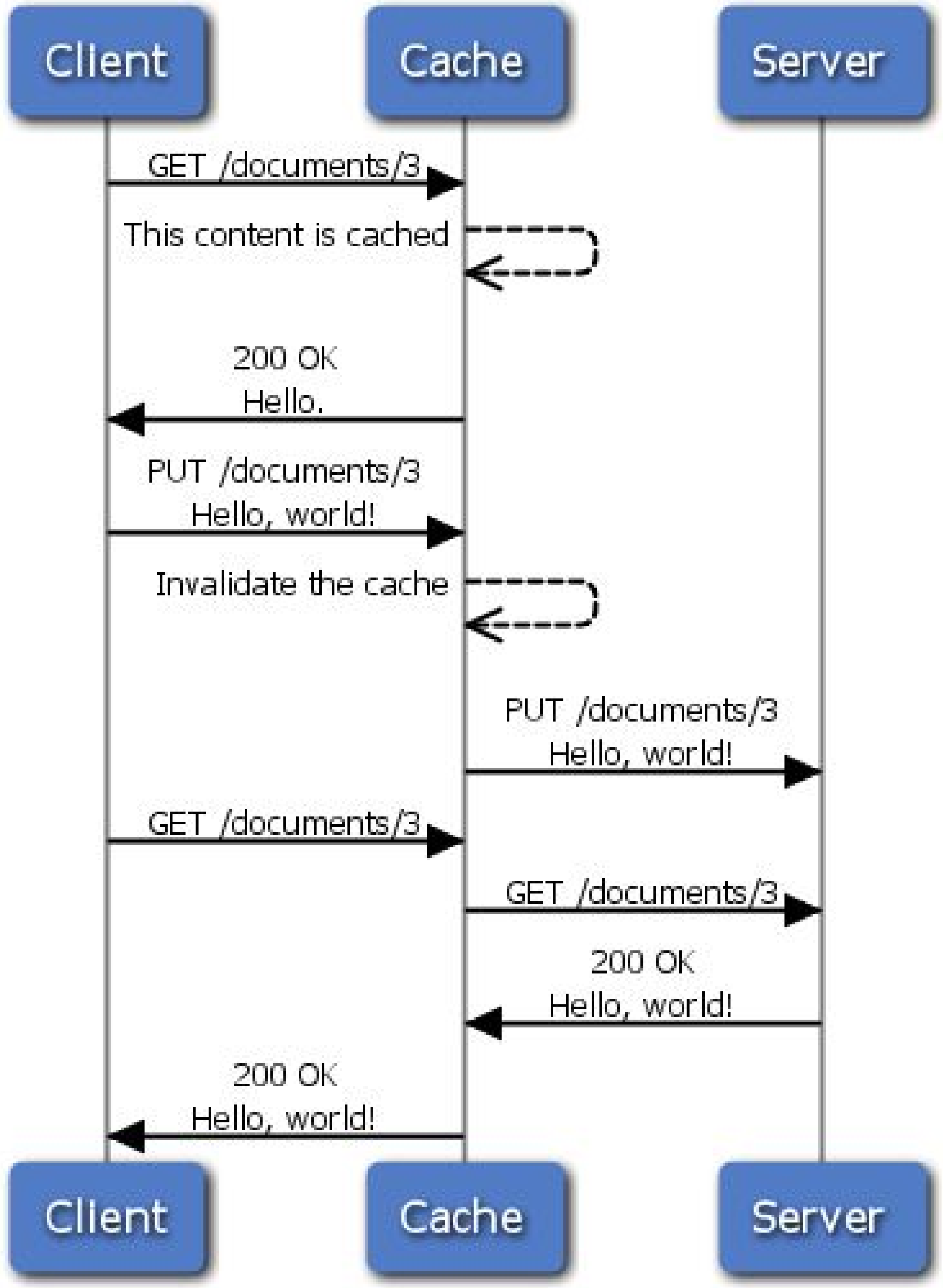


Example of validation caching

Caching REST

Aside: REST works nicely with compliant caches.

- Designed to work with HTTP
- GET from a URL to read
- PUT to the same URL to update
- PUT operations invalidate cached content

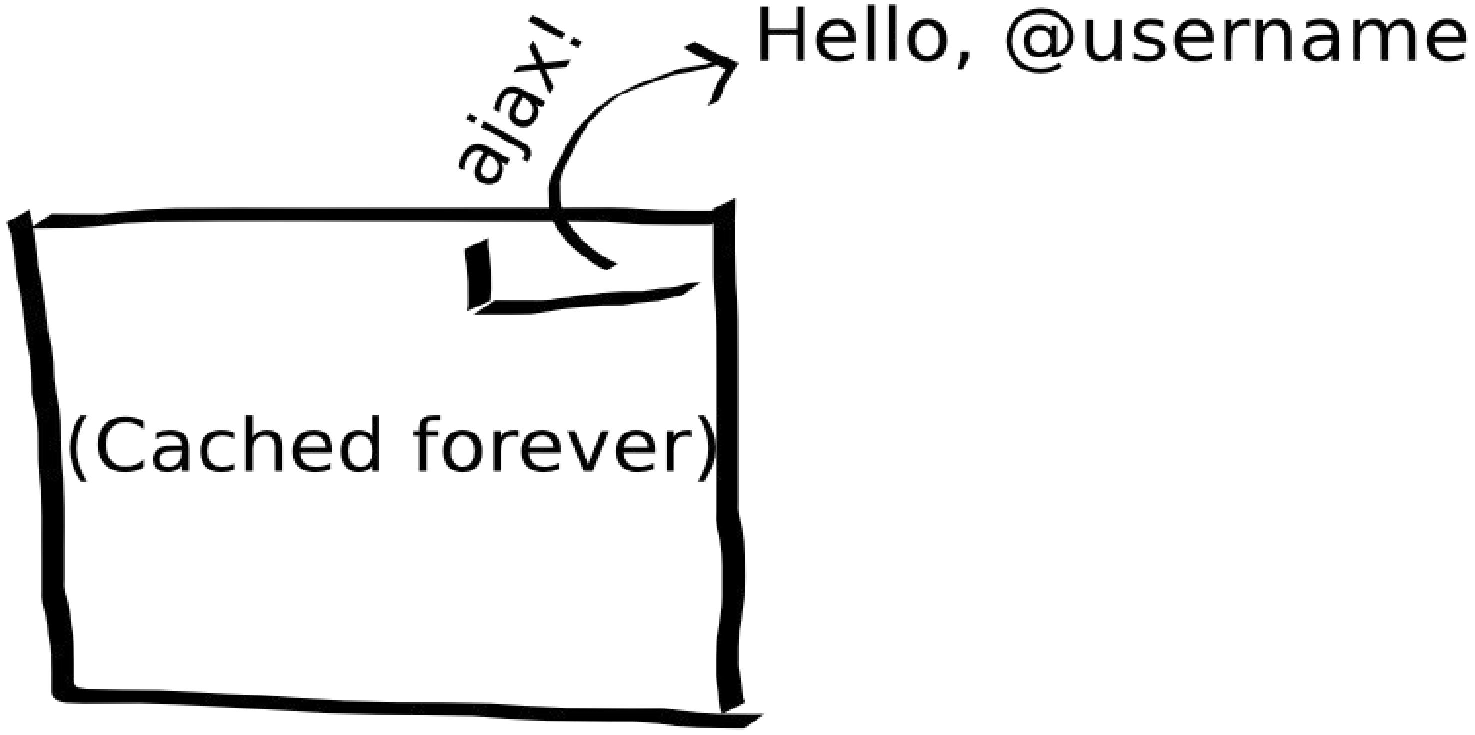


REST requests mesh nicely with HTTP caching

Thick Clients

Load data asynchronously with Ajax

- Can combine dynamic and cached content
- View and data can be cached separately



Gzip Your Content

- Big Bandwidth Savings
- Part of HTTP/1.1 spec
- Easy to set up in your web server configuration

In you Apache configuration

```
AddOutputFilterByType DEFLATE text/html text/plain text/xml
text/css text/javascript application/javascript
application/x-javascript
BrowserMatch ^Mozilla/4 gzip-only-text/html
BrowserMatch ^Mozilla/4\.0[678] no-gzip
BrowserMatch \bMSIE !no-gzip !gzip-only-text/html
```

References

HTTP caching

http://tomayko.com/writings/things-caches-do - Concise writeup of HTTP caching

http://www.mnot.net/cache_docs/ - More detailed information about HTTP caching

http://tools.ietf.org/html/rfc2616#section-13 - RFC describing HTTP caching

http://www.ircache.net/cgi-bin/cacheability.py - Tool for testing your app's cacheability

http://www.ibm.com/developerworks/web/library/wa-rails2/ - Rails-specific strategies for page caching

References (cont.)

Gateway cache implementations

http://varnish-cache.com/ - Varnish

http://www.squid-cache.org/ - Squid

http://tomayko.com/src/rack-cache/ - rack-cache

References (cont.)

Code Examples

http://github.com/hallettj/qrcode-rails - Source of QRCode-generating application

More resources

http://www.engineyard.com/blog/2009/5-tips-to-scale-your-ror-application/ - A list of tips for improving Rails application performance.

http://www.websequencediagrams.com/ - UML diagram generator