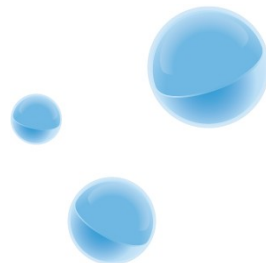
VARNISH

Makes Websites Fly

Demystifying cache

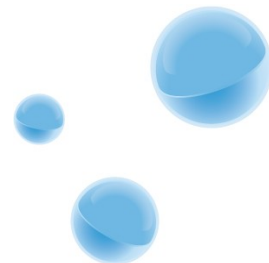
Kristian Lyngstøl
Product Specialist
Varnish Software AS

Montreal, March 2013

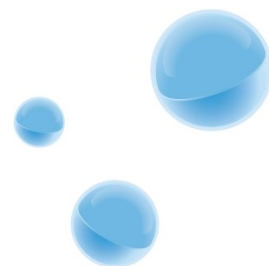


Agenda

- The types of caches involved
- The benefits of a cache
- HTTP
- Reverse proxy specifics



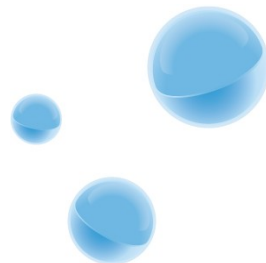
Not: L1/L2 cache. Disk cache, etc.

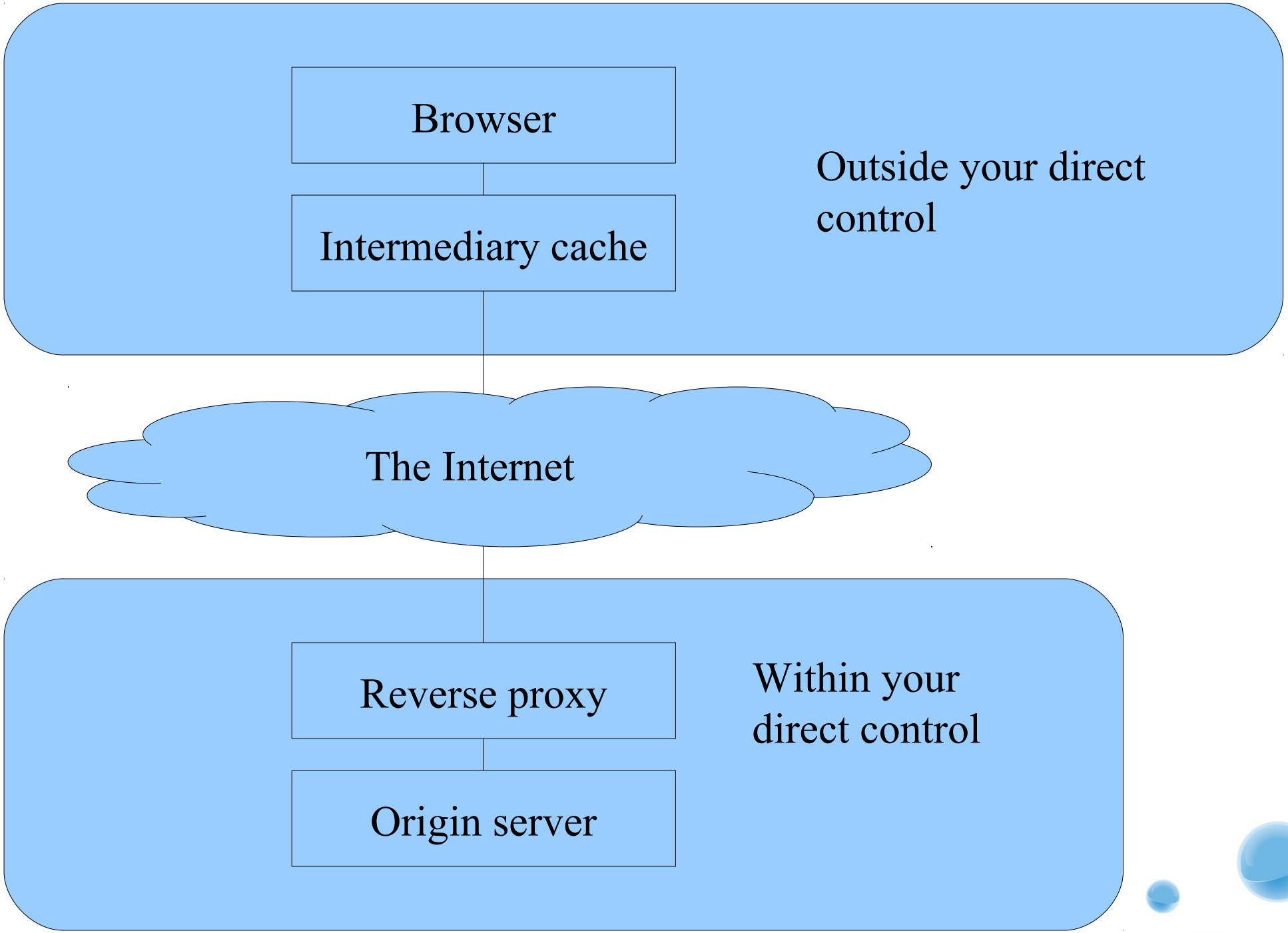


Browser cache

Intermediary caches

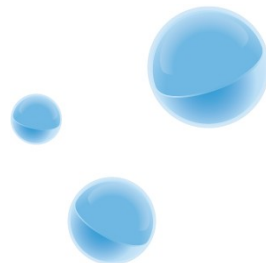
Reverse proxy




Browser
Intermediary cache
Outside your direct control
The Internet
Reverse proxy
Origin server
Within your direct control

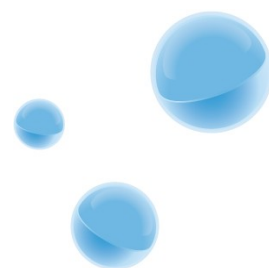
Pro: Reduced server load

Con: Increased complexity



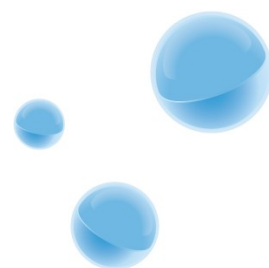
Pro: Increased robustness

Con: Increased complexity == harder problems



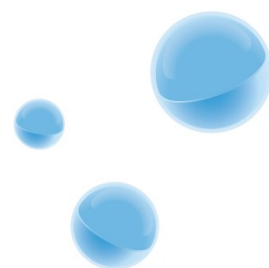
Pro: Easier scaling

Con: Complex architecture

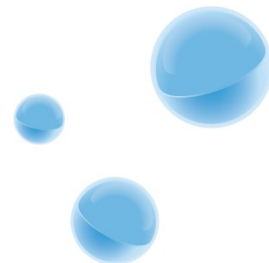


Pro: Reduced latency by geographic distribution

Con: Increased latency on cache misses



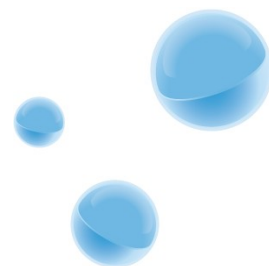
HTTP: Written for cache.



HTTP

The original REST interface

Addresses browser cache, intermediary caches but **NOT** reverse proxies.

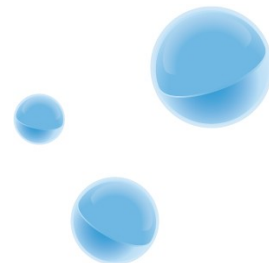


`Vary:`

Browser: GET /foo, Accept-Encoding: gzip

Server: Here's /foo. It's compressed. `Vary: Accept-encoding`.

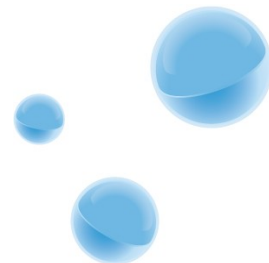
Vary is a way for a server to signal that there might be different variants of the content depending on a specific header.



`Vary:` examples

- `Vary: Accept-Encoding` Compression.
- `Vary: User-Agent` Mobile content?
- `Vary: Cookie`
- `Vary: Accept-Language`
- All of the above

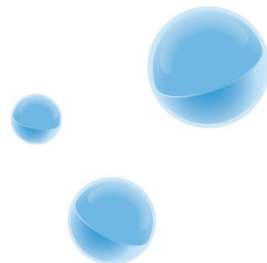
Most common by far: Vary: Accept-Encoding



Vary-challenges

Content is provided either in gzip or uncompressed form, yet “Accept-Encoding” can contain any number of potential algorithms.

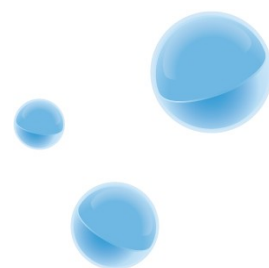
Content varies depending on a specific cookie, but no way to tell which by using just Vary. (all or nothing).



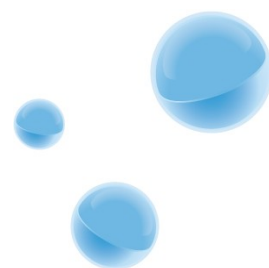
“Accept-Encoding: gzip, deflate”

“Accept-Encoding: deflate, gzip”

Two different things!

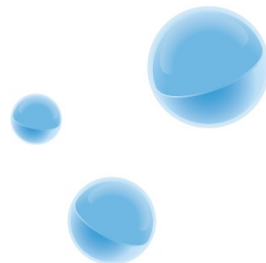


Without a cache, vary has almost no side effects.
Thus often ignored.



What we should do:

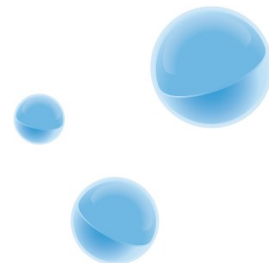
Vary: Cookie, User-Agent



Technique number 0: FAIL SAFE

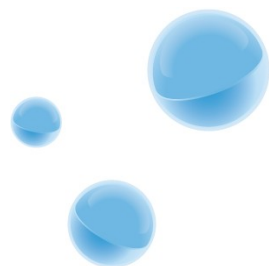
You will fail at some point. Make sure you fail in an acceptable manner.

Is it better to disable a cache or artificially inflate the cache than to deliver user-specific content to the wrong user?

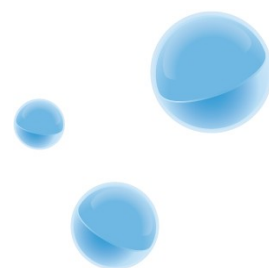


Hypothetical example

- Site allows students to register for a summer event.
- Last minute change.
- At launch, the cache is forced to cache content with “Set-Cookie” headers present.
- Students end up taking over each others' sessions
- Lawsuit to distribute blame lasts for years.

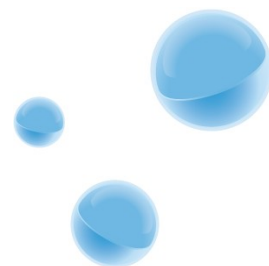


The by far most common cache mistake is ignoring cookies.

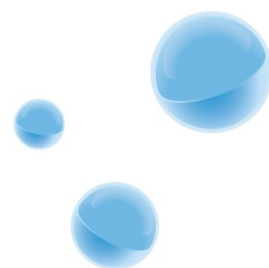


“If content has cookies, clean out all but THESE cookies”

“If content still has cookies, either DO NOT cache, or add the entire cookie string to the hash key, then cache.”

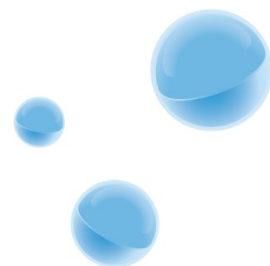


Never. Ever. Cache. Set-Cookie. (unless, of course, you have a good reason!)

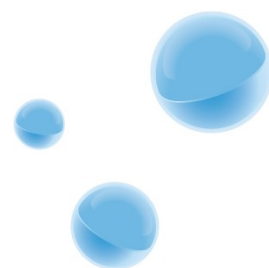


Example: Kenneth (36)

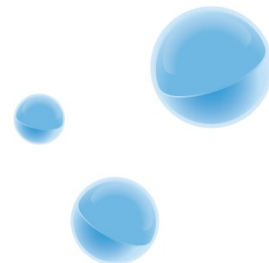
Kenneth (36) became famous when his tax returns were incorrectly cached and thousands of users got to see his tax returns instead of their own....



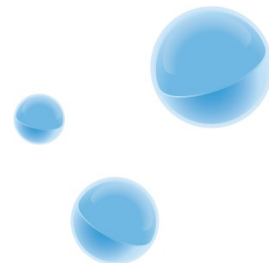
Some would argue that the site going down might have been better.



Cache-Control header

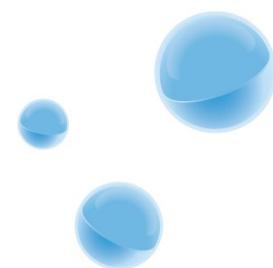


max-age: For browsers, also used by intermediary caches.



s-maxage: For caches, used by both intermediary caches AND reverse proxies.

Tip: Use it for reverse proxies, then remove it before exposing it to intermediary caches.

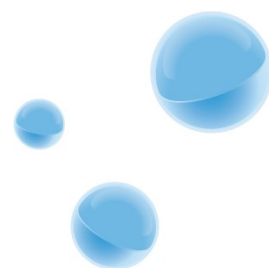


must-revalidate: You can cache, but revalidate before using it.

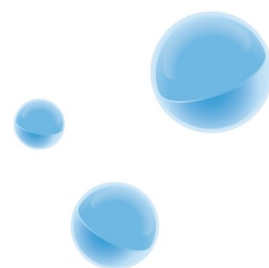
private: Only browser cache. No shared cache.

public: Can be cached.

no-cache: Similar to must-revalidate.



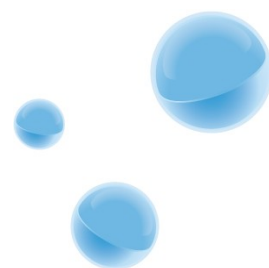
Emerging: Surrogate-Control, a “Cache-Control” for surrogate caches (aka: reverse proxies) only.



Conditional requests

“Give me /pictures/cats, but only if it's newer than X”

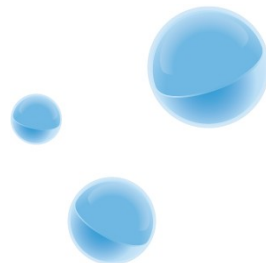
If-Modified-Since:



Conditional requests

“Give me /pictures/cats, but only if it doesn't match Etag FOO.”

If-None-Match: Foo

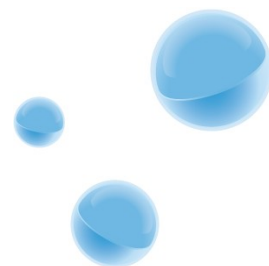


ETags

Essentially a Unique ID for an asset/resource/url.

Not only useful for caches.

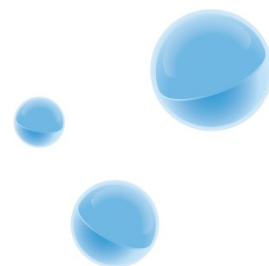
“UPDATE this resource, but only if the old version matches what I had.”



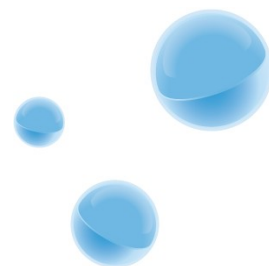
```
GET /foobar HTTP/1.1
ETag: FOO-version1
```

(time passes)

```
DELETE /foobar HTTP/1.1
If-Match: FOO-version1
```



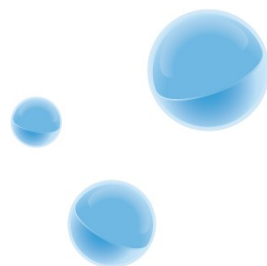
Handling high loads == handling bugs, DoS attacks and more.



Example: Counting comments

20-ish news sites, each with comment section. To display “X comments on this article” on the frontpage, a resource contained a list of all articles and the counters.

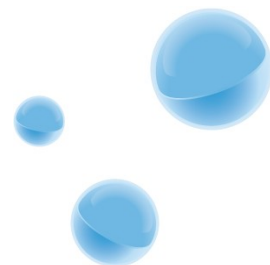
“This updates constantly, impossible to cache!”



What about Expires and Pragma?

Pragma: Not defined anywhere. Do not use.

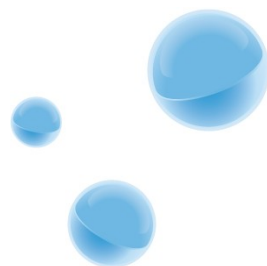
Expires: Troublesome at best. Usually set to some developer's birthday or that 1997-date that seems to have originated from an example on php.net.



Common Expires values

```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Expires: Sun, 19 Nov 1978 05:00:00 GMT
Expires: Sat, 26 Jul 1997 05:00:00 GMT
Expires: Mon, 26 Jul 1997 05:00:00 GMT
```

(Mon, 26, Jul 1997 does not exist)



Contact information

Kristian Lyngstøl

kristian@bohemians.org

@kristianlyng

http://kly.no