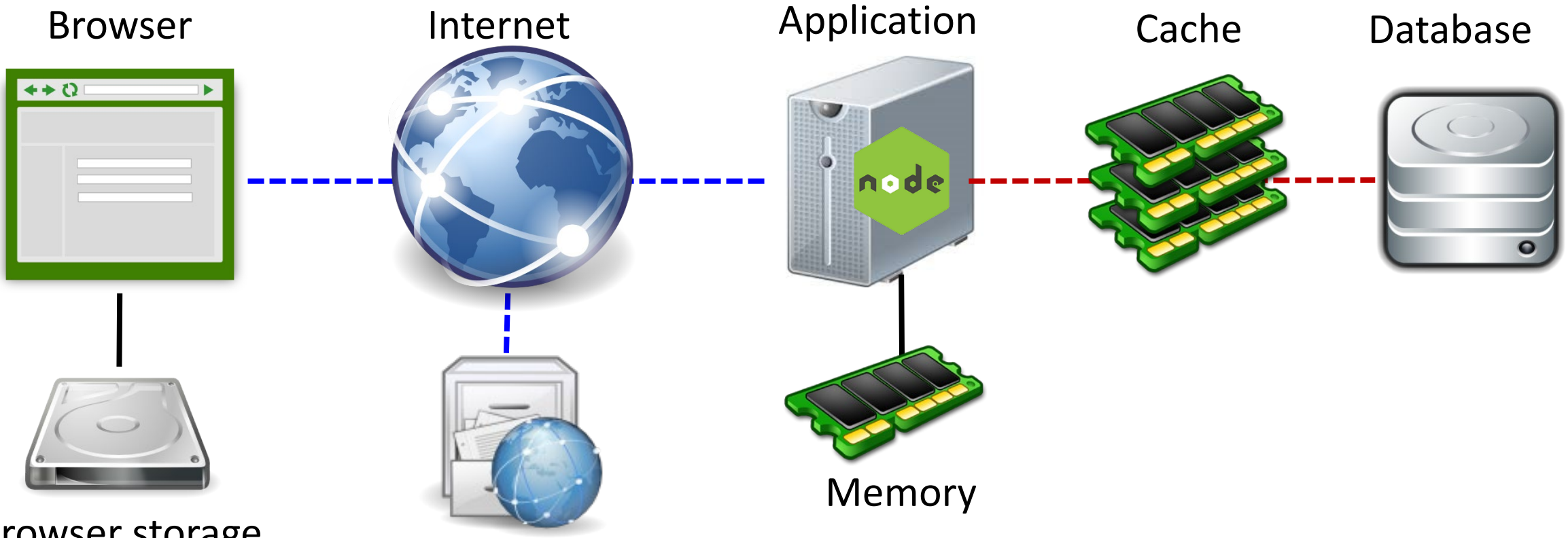
Cache

Where Can Data be Persisted?



Cache
Local storage
Indexed DB

Cloudflare
Akamai
AWS CloudFront

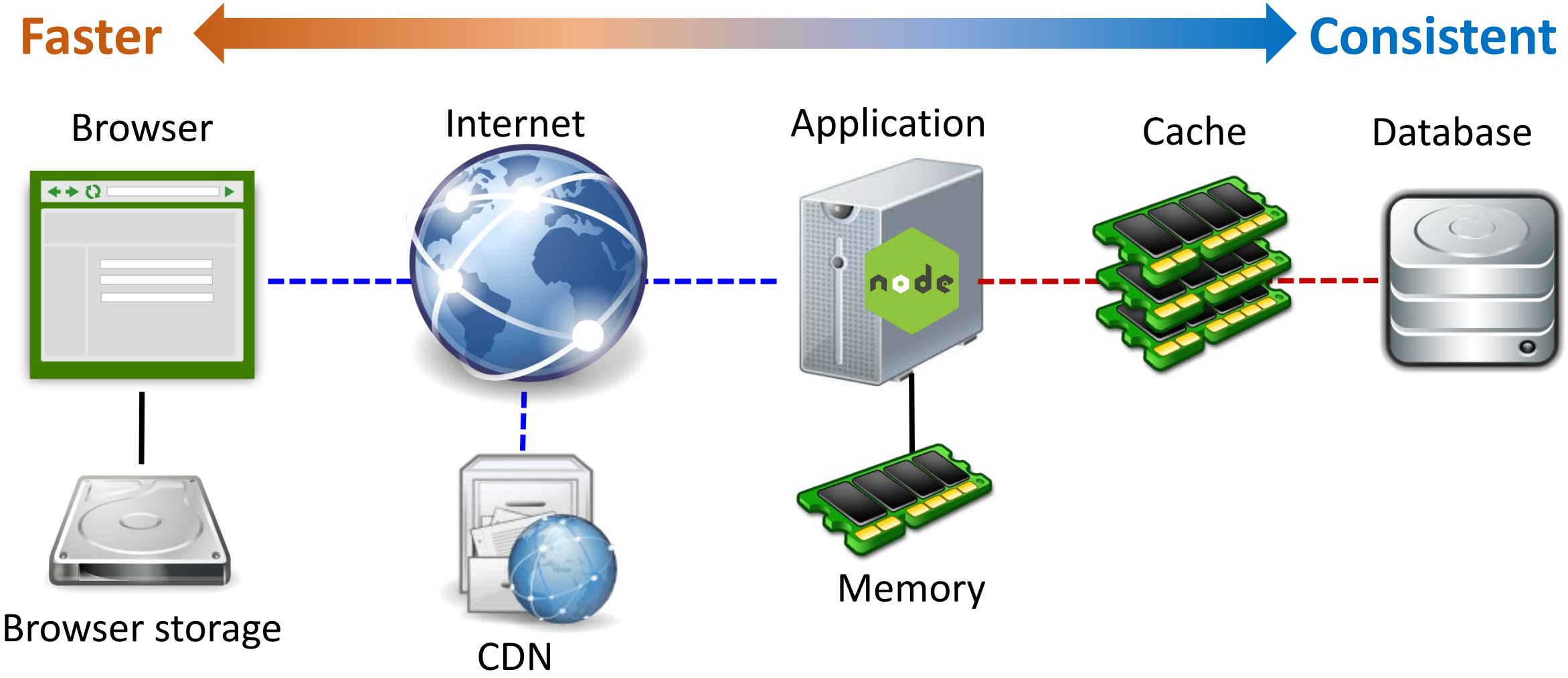
Memcached
Redis
Mongo

MySQL
PostgreSQL
SQLite

Caching

- Temporary storing data/information such as images, video clips, audio, web pages
  - Referring to information and data generated by application
- To speed up the delivery of data to the client
  - As the information are stored ‘closer’ to the client
  - The required data may be satisfied by what is in the cache
- Validity of the data
  - Hold a copy of the data, not the source of truth
  - Data may change without knowledge of the cache
- Cache can be controlled by
  - Application - either on the client or server
  - Internet - content delivery network

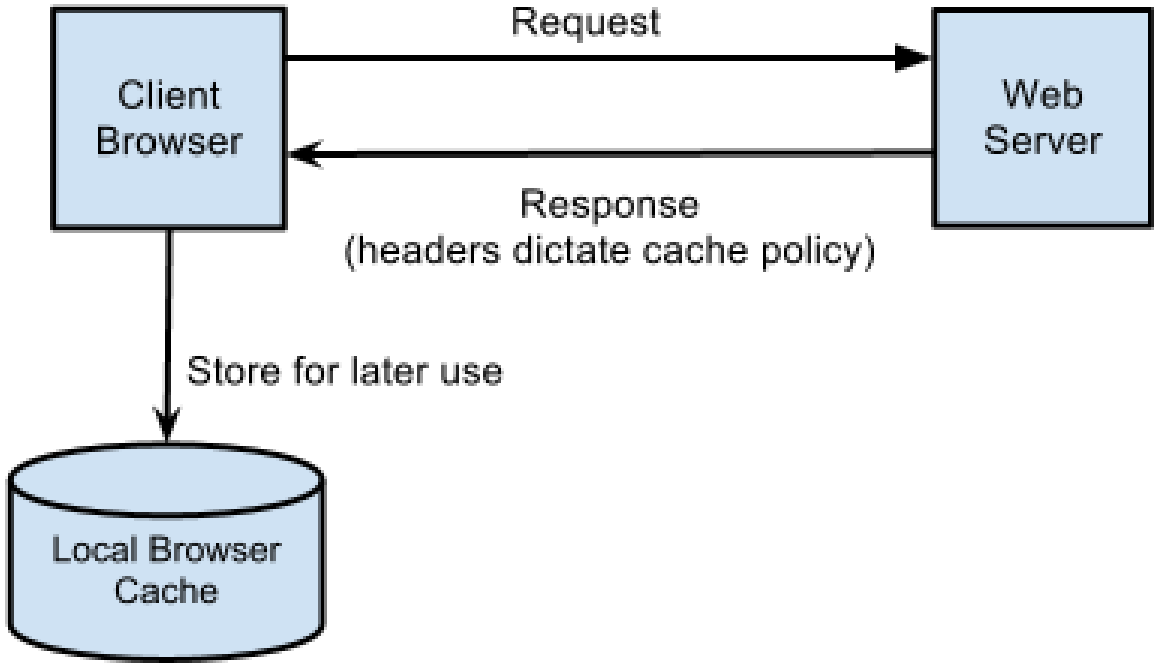
Persistence Characteristics

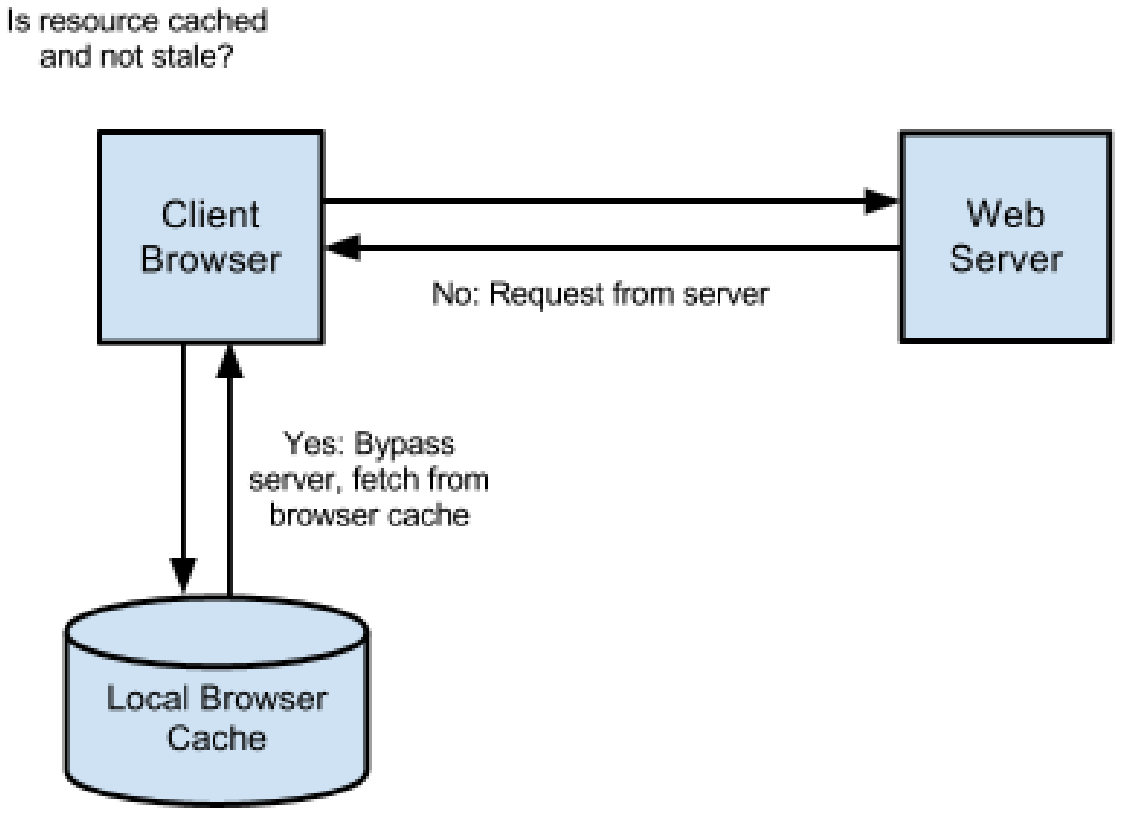


What to Cache?

- Immutable data - data that do not change
  - Eg. images, audio, news article
- Expensive to generate data - data that takes a long time to generate or computationally expensive to generate
  - Eg. weekly sales report
- Heavily used data that required fast response - data that are required frequently
  - Eg. last sell price of a stock

How Does Caching Works?

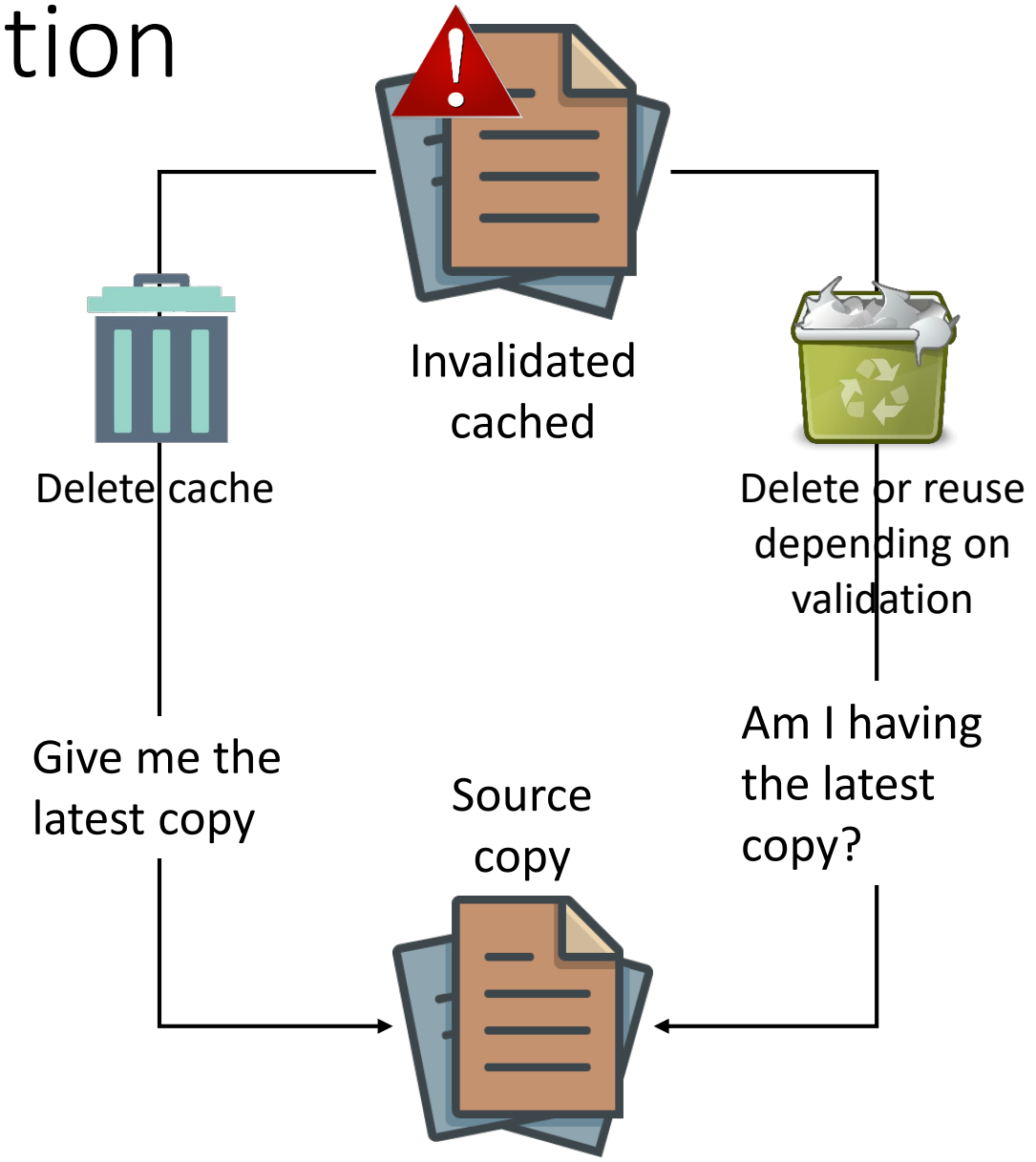




Images from https://devcenter.heroku.com/articles/increasing-application-performance-with-http-cache-headers

Invalidation and Revalidation

- Cache can be invalidated
  - Manually by the user
  - Past the ‘used-by’ date - freshness
- Handling invalidated cache
  - Re-fetch the latest version
  - Revalidate the invalidate copy
- Revalidate - ask the source if the current cached copy has changed
  - No extra download if copy is unchanged



Caching By Time and Content

- Cache will serve data for a fixed period of time
- Will be considered stale after the expiry period - discarded
- Does not require revalidation from the server

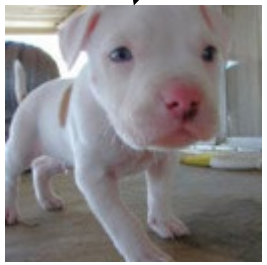
- Cache will serve data as long as the content has not changed
- A digital signature is generated for the cached content
- Will be considered stale if the digital signature of the content changes
- Has to be revalidated before using

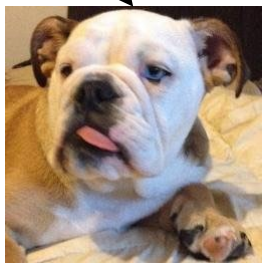
Caching By Time and Content

- Resource URL uniquely identifies the content
  - Change the URL and the content changes
  - Static assets

`www.pets.com/dog/pitbull.png`

`www.pets.com/dog/bulldog.png`





- Resource URL is independent of the underlying content
  - Changes in the content is not reflected by the URL
  - Dynamically generated content

`www.news.com/latest`

Cache Sharing - Public or Private

- Data in public caches are shared by many client
  - Eg. News headline
- Data is either immutable or change infrequently

- Privately cache data are data that is only cached for a single client
  - Eg. shopping cart of a customer
- Typically data requires authentication

`Cache-Control` Header

- **`Cache-Control`** header uses the following directive to control caching
  - `public` - result can be cached by public caches and can be shared eg. CDN
  - `private` - result can be cached by the browser and is intended for the recipient only. Should not be cached by public caches.
  - `no-cache` - do not use cached unless it has been revalidated
  - `no-store` - do not cache
  - `max-age` - in seconds. Used to control cache invalidation
- The meaning of the directive changes depending on whether the header is in a request or response

Example **Cache-Control**

```
GET /api/customers
```

```
200 OK
Cache-Control: public, max-age=90
```

Response can be cached and result can be shared with other requests for the next 90 seconds

```
200 OK
Cache-Control: private, max-age=90
```

Response is intended for client only and therefore should only be cached by the client/browser and no one else

```
200 OK
Cache-Control: no-store
```

Response should not be cached

Example **Cache-Control**

```
GET /api/customers
Cache-Control: max-age=30
```

Client is not willing to accept response from caches if the result has been in the cached longer than 30 seconds

```
GET /api/customers
Cache-Control: no-store
```

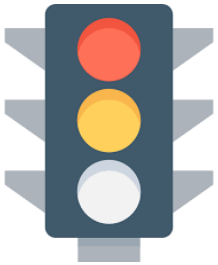
Client is indicating that both the request and the corresponding response should not be cached

```
GET /api/customers
Cache-Control: no-cache
```

Response to the request must not be a cached copy viz. the response must always originate from the server

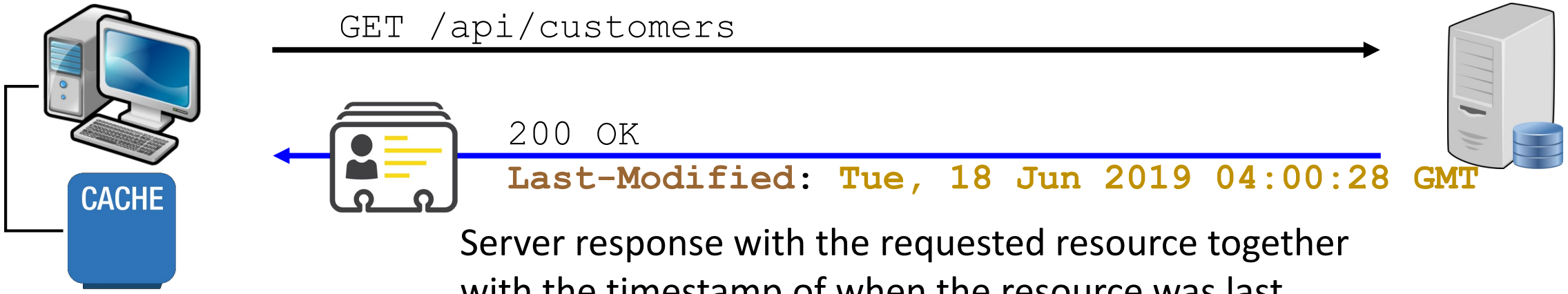
```
200 OK
```

Time Based Conditional GET

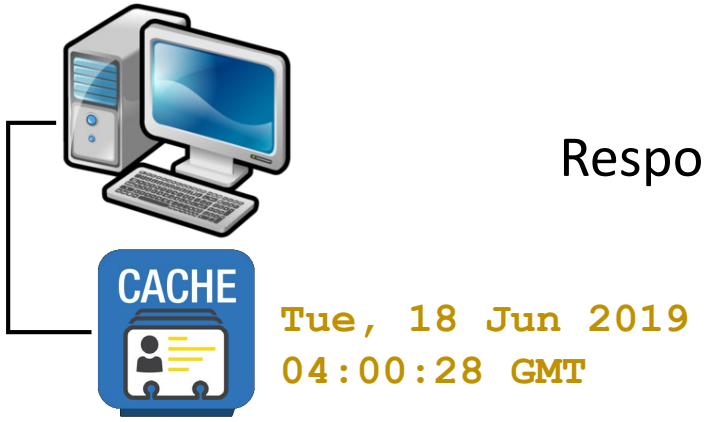


- Ask the server if the resource has been modified since a certain time. Use
  - `Last-Modified` HTTP header in the response
  - `If-Modified-Since` HTTP header in the request

Time Based Conditional **GET**

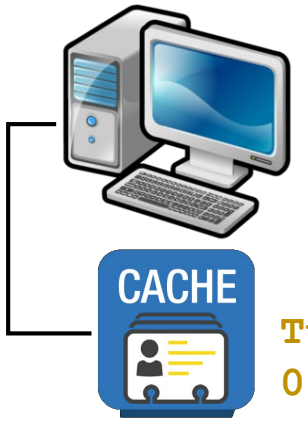


Server response with the requested resource together with the timestamp of when the resource was last modified



Response is cached along with the last modified date

Time Based Conditional **GET**

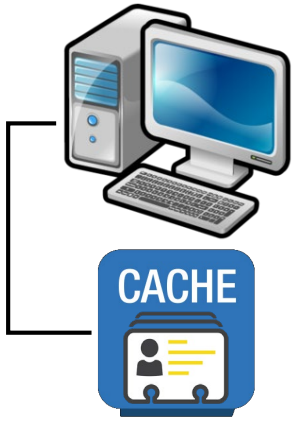


Tue, 18 Jun 2019
04:00:28 GMT

```
GET /api/customers
If-Modified-Since: Tue, 18 Jun 2019 04:00:28 GMT
```

```
304 Not Modified
```

Returns a 304 status if the document has not changes since the timestamp. Note no payload is sent. The client will get its result from the cache




```
GET /api/customers
If-Modified-Since: Tue, 18 Jun 2019 04:00:28 GMT
```

A new copy of the resource is returned with a different `Last-Modified` timestamp

```
200 OK
Last-Modified: Wed, 19 Jun 2019 01:25:46 GMT
```

Example Time Based Caching

Express

```
const cacheControl = require('express-cache-controller');
const preconditions = require('express-preconditions');

const options = {
  stateAsync: (req) => new Promise((resolve, reject) => {
    resolve({ lastModified: /* last modified date of the request */ })
  })
}

app.get('/api/employee/:id',
  cacheControl({ maxAge: 30, private: true }),
  preconditions(options),
  (req, resp) => {
    const result = //get employee with lastModified date
    resp.setHeader('Last-Modified', result.lastModified)
    resp.status(200).json(result);
});
```

Last modified date of the resource requested by the request

Return the timestamp

Create a function to be used by preconditions middleware to get the last modified timestamp

`Cache-Control: private, max-age=30`

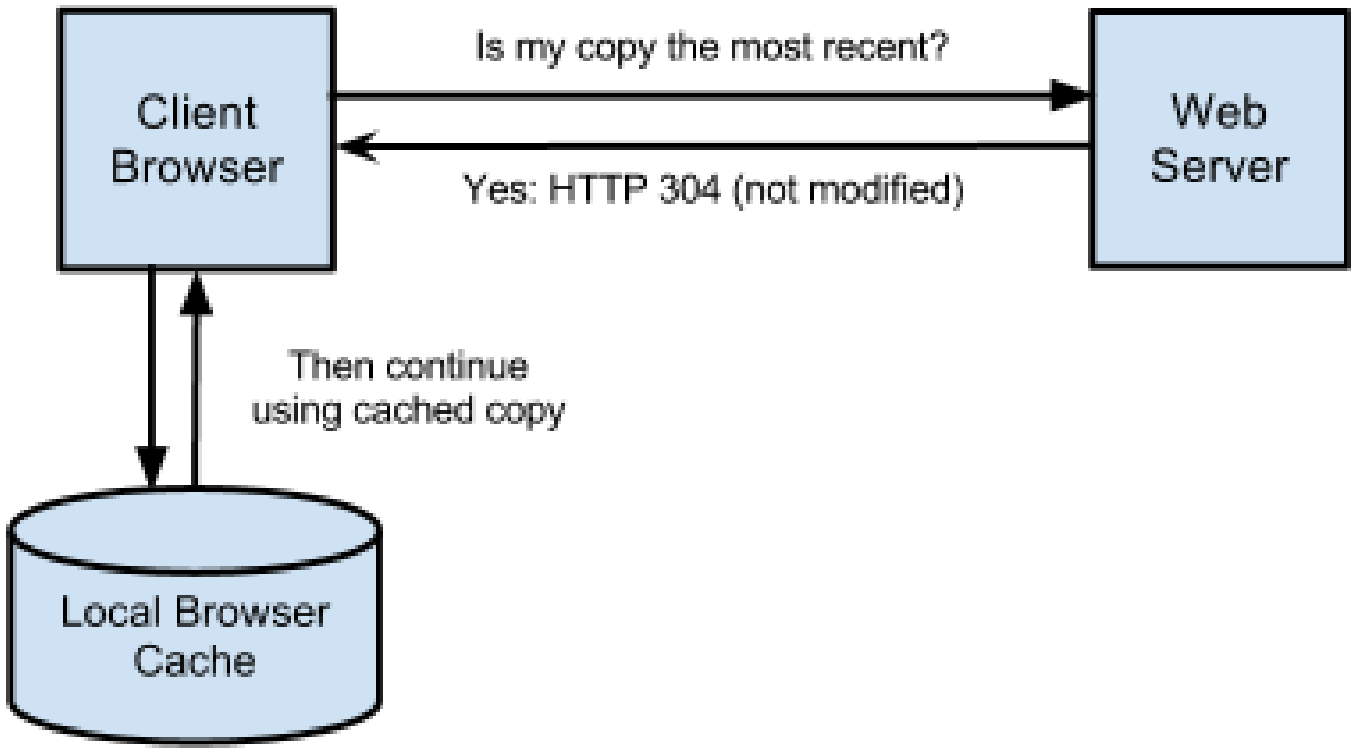
Will return a 304 if the current modified date is earlier that the timestamp in the request

Add the `Last-Modified` header to the response

Caching Dynamic Content

- **`Cache-Control`** good for caching responses that do not change that often, eg. images
  - Caching dynamically generated responses may not be very effective
  - Eg. Aggregating news
- Caches cannot determine the validity of the response
  - Only the application that generated the response know
  - Have to set `max-age=0` in the request to get the latest copy
- Dynamically generated responses may be very volatile
  - Allow server/application to determine if it needs to regenerate response

Caching By Content

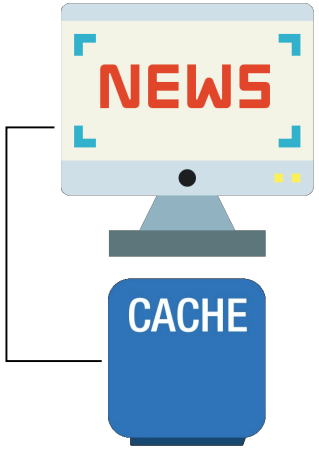


Images from https://devcenter.heroku.com/articles/increasing-application-performance-with-http-cache-headers

Etag

- Etag (entity tag) is an identifier assigned to a response
  - Eg. `abc123`
- Generated by the server for identifying a response in time
  - Can also be generated by caches
- Like a digital signature
  - On the content of the response
- Can be expensive to generate, so
  - Use for resources that are expensive to produce
  - Requested frequently
  - Large response/payload

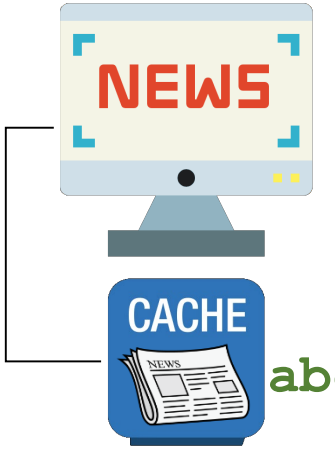
How Does ETag Work?




```
GET /api/headlines?category=asia
```


```
200 OK
ETag: "abc123"
Cache-Control: no-cache
```

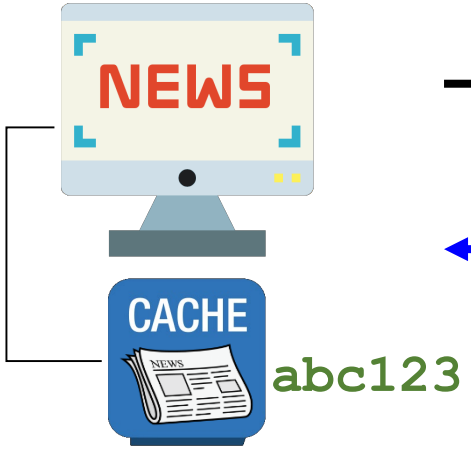
The requested content is returned with a tag that uniquely identifies the resource at that point in time.
Note the **`no-cache`** directive to force revalidation. ETag is the validator



The resource is cached along with its tag by the client

Inspired from https://keshvica.wordpress.com/2015/07/12/web-cache-validation-with-etags/

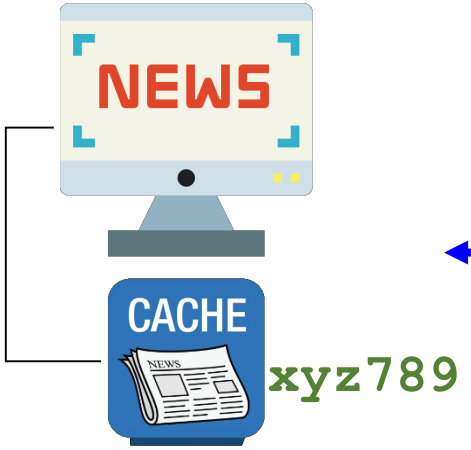
Revalidating ETags




```
GET /api/headlines?category=asia
ETag: "abc123"
```

```
304 Not Modified
```

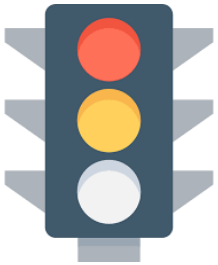
Application checks if the ETag presented by the client is the same as the current resource state. If they are the same, then a 304 status will be returned




```
200 OK
ETag: "xyz789"
Cache-Control: no-cache
```

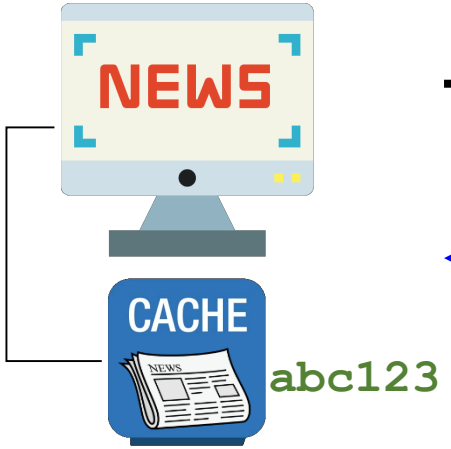
Inspired from https://keshvica.wordpress.com/2015/07/12/web-cache-validation-with-etags/

Content Based Conditional GET



- Explicitly ask if the content has changed
- Ask the server if the content has changed based on the ETag of the resource
- Uses `If-None-Match` HTTP header in the request
- Server response with either
  - 304 viz. content not modified, or
  - New resource with a matching ETag if content changes

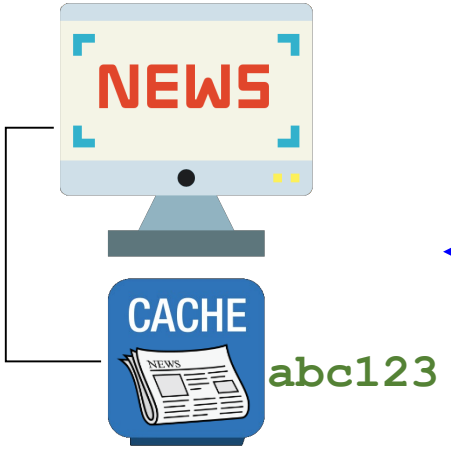
Content Based Conditional **GET**




```
GET /api/headlines?category=asia
If-None-Match: "abc123"
```

```
304 Not Modified
```

A request is for the same resource at a later point in time. The ETag of the resource is pass with the request in `If-None-Match` header
If the resource has not change, the application returns a 304 status indicating to the client that it already has the resource. Not data is passed back



```
200 OK
ETag: "xyz789"
Cache-Control: no-cache
```

If the resource has changed, then the server will return a new copy of the resource along with the corresponding ETag

Inspired from https://keshvica.wordpress.com/2015/07/12/web-cache-validation-with-etags/

Types of ETag

`"12345678"`

`W/"12345678"`

- Strong ETag indicates that the response and the source content is byte-for-byte identical

- ETag is prefixed with `W/` to indicate that it's a 'weak' ETag
- Indicates that the response and the source content are semantically equivalent
  - Allow caches to serve similar resources
- Use when its impossible to get content to be 100% the same
  - E.g front page of newspaper with advertisement

Generating ETag

- Use the last modified date
  - Keep a `last-modified` field for records which is updated whenever a record is updated

```
create table customer (
  custId int primary key,
  ...
  lastModified default current_timestamp
      on update current_timestamp
)
```

  - ORMs like Sequalized automatically adds a update timestamp field
    - `updateAt` field

Generating ETag

- Capture the content of the page
  - Hash with MD5 - may be too ‘heavyweight’ if the content changes frequently or private
  - First ‘few lines’ of the content if the content are ordered eg. latest news ordered by timestamp
- Rule of thumb - should be fast

Example Content Based Caching

Express

```
const cacheControl = require('express-cache-controller');
const preconditions = require('express-preconditions');

const options = {
   stateAsync: (req) => new Promise((resolve, reject) => {
      resolve({ etag: /* return the etag of the current resource */ })
   })
}

app.get('/api/employee/:id',
   cacheControl({ maxAge: 30, private: true }),
   preconditions(options),
   (req, resp) => {
      const result = //get employee with lastModified date
      resp.setHeader('ETag', new Date(result.lastModified).toGMTString());
      resp.status(200).json(result);
});
```

Using the `lastModified` data as ETag.

Updating Resources

Fred

Application

Wilma

GET /api/customer/1

GET /api/customer/1

200 OK

200 OK

PUT /api/customer/1

PUT /api/customer/1

204 No Content

Wilma's updates are lost

204 No Content

Conditional PUT

- Uncoordinated distributed updates will result in data loss
- Optimistic lock/concurrency
  - Server should only update a resource if the resource have not changed
- Content based optimistic lock
  - Uses `ETag` to identify the state of the resource
  - Request uses `If-Match` header to notify the server to only perform the update if the resource to be update has the same ETag as the `If-Match`
- Time based optimistic lock
  - Use `Last-Modified` to timestamp when the resource was last updated
  - Request use `If-Unmodified-Since` header to notify the server the last modified time of the client's copy
- If either of the condition fails, server will return a 412 status

Updating Resources

Fred

Application

Wilma

GET /api/customer/1

GET /api/customer/1

200 OK
ETag: "123"

200 OK
ETag: "123"

PUT /api/customer/1
If-Match: "123"

PUT /api/customer/1
If-Match: "123"

204 No Content
ETag: "abc"

412 Precondition Failed

Caching Summary

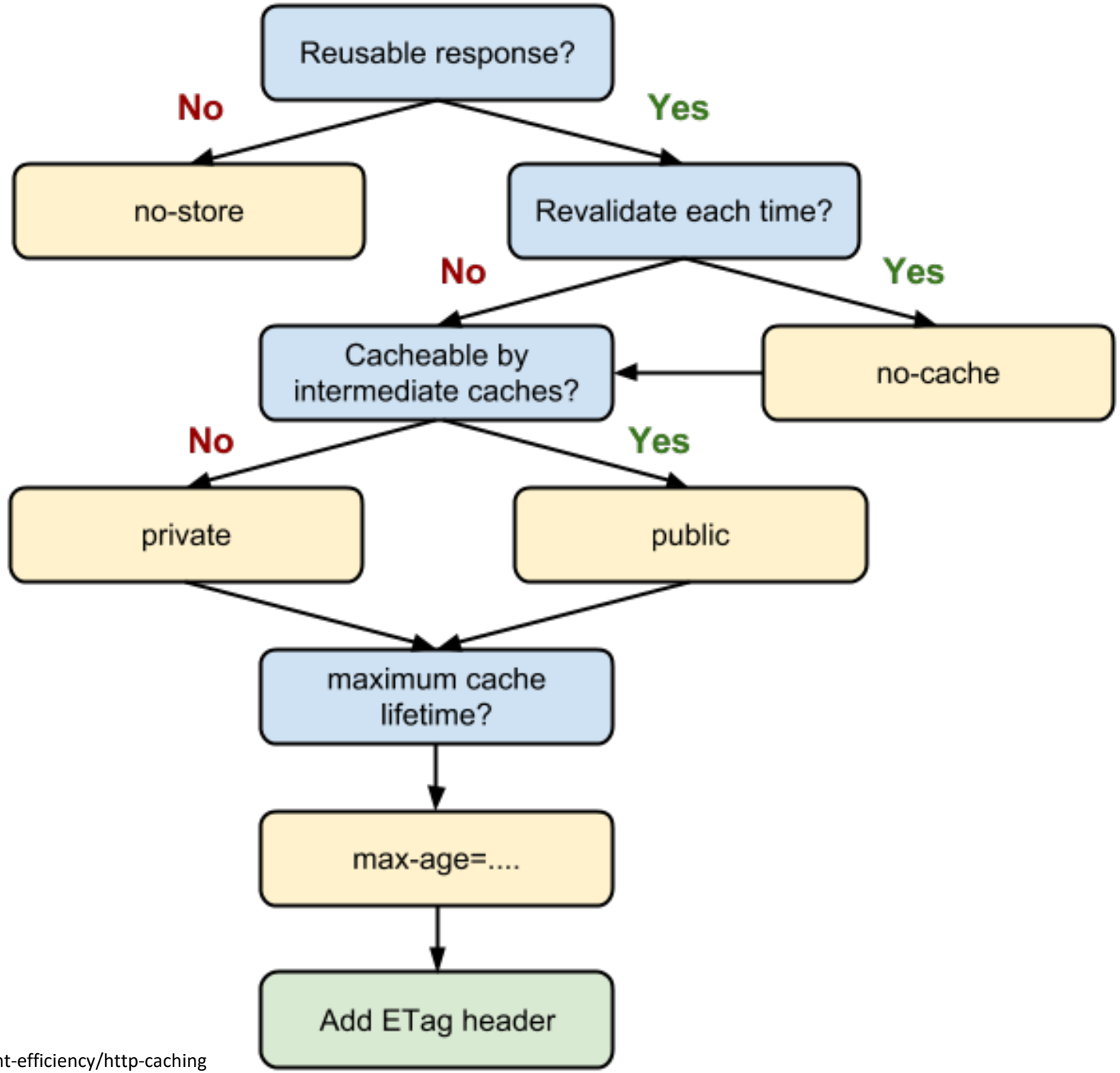
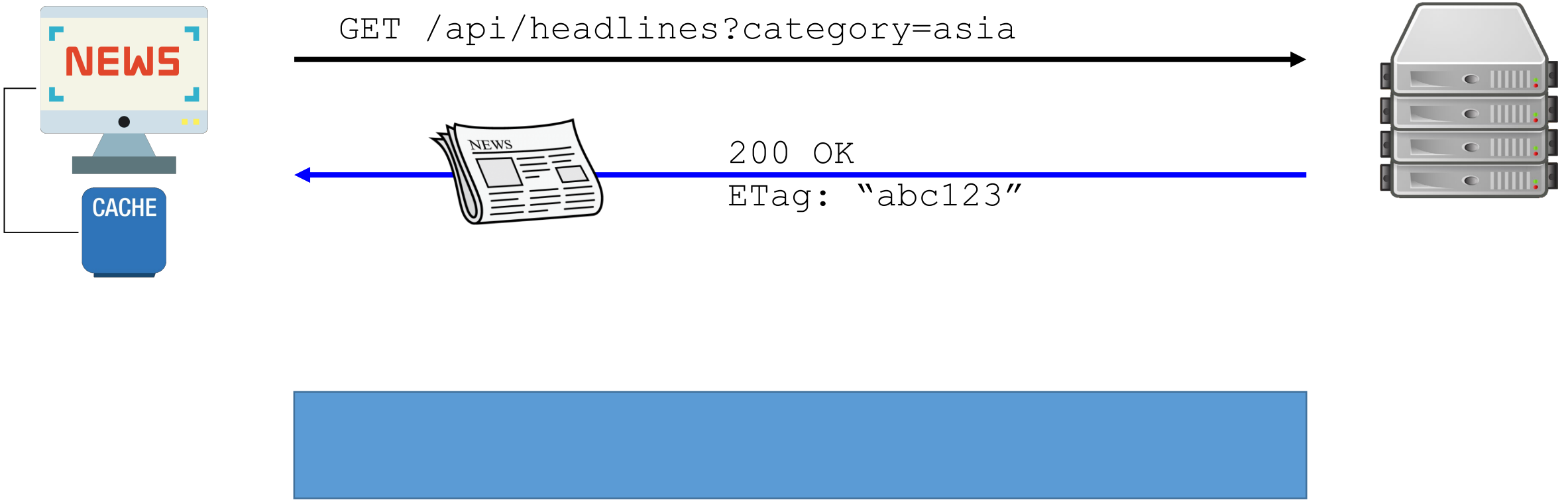


Image from https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/http-caching

List of Modules

- express-cache-controller - https://github.com/DaMouse404/express-cache-controller
- express-preconditions - https://github.com/richardschneider/express-conditional-request

How Does Etag Work?

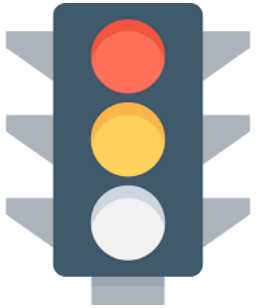


Inspired from https://keshvica.wordpress.com/2015/07/12/web-cache-validation-with-etags/

Caching By Time, Content, Condition

- Cache will serve data for a defined period of time
- Will be considered stale after the expiry period - discarded
- Does not require recheck/revalidate from the server

- Cache will serve data as long as the content has not changed
- A digital signature is generated for the cached content
- Will be considered stale if the digital signature of the content changes



- Cache will serve data as long as the data has not changed
- A time is associated with the data
- If the data has changed since that time, then it is considered stale