Functional Web Apps with

webmachine

Sean Cribbs Chris Meiklejohn

Download

http://bit.ly/wmlj-pdf

http://bit.ly/wmlj-zip

Clone

github.com/cmeiklejohn/webmachine-tutorial

Introduction

Have you ever...

Have you ever...

CGI

Have you ever...

CGI

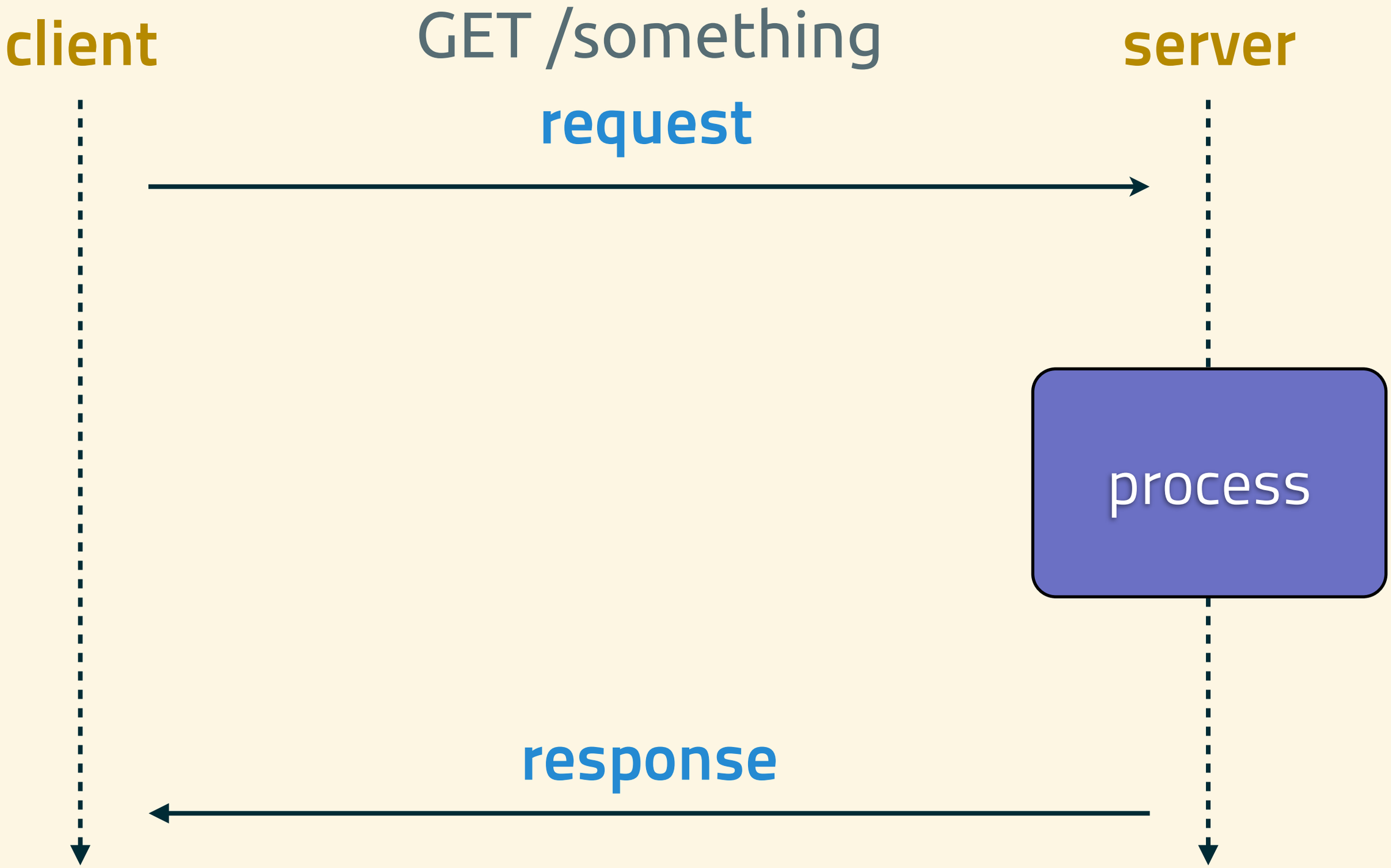
Servlet

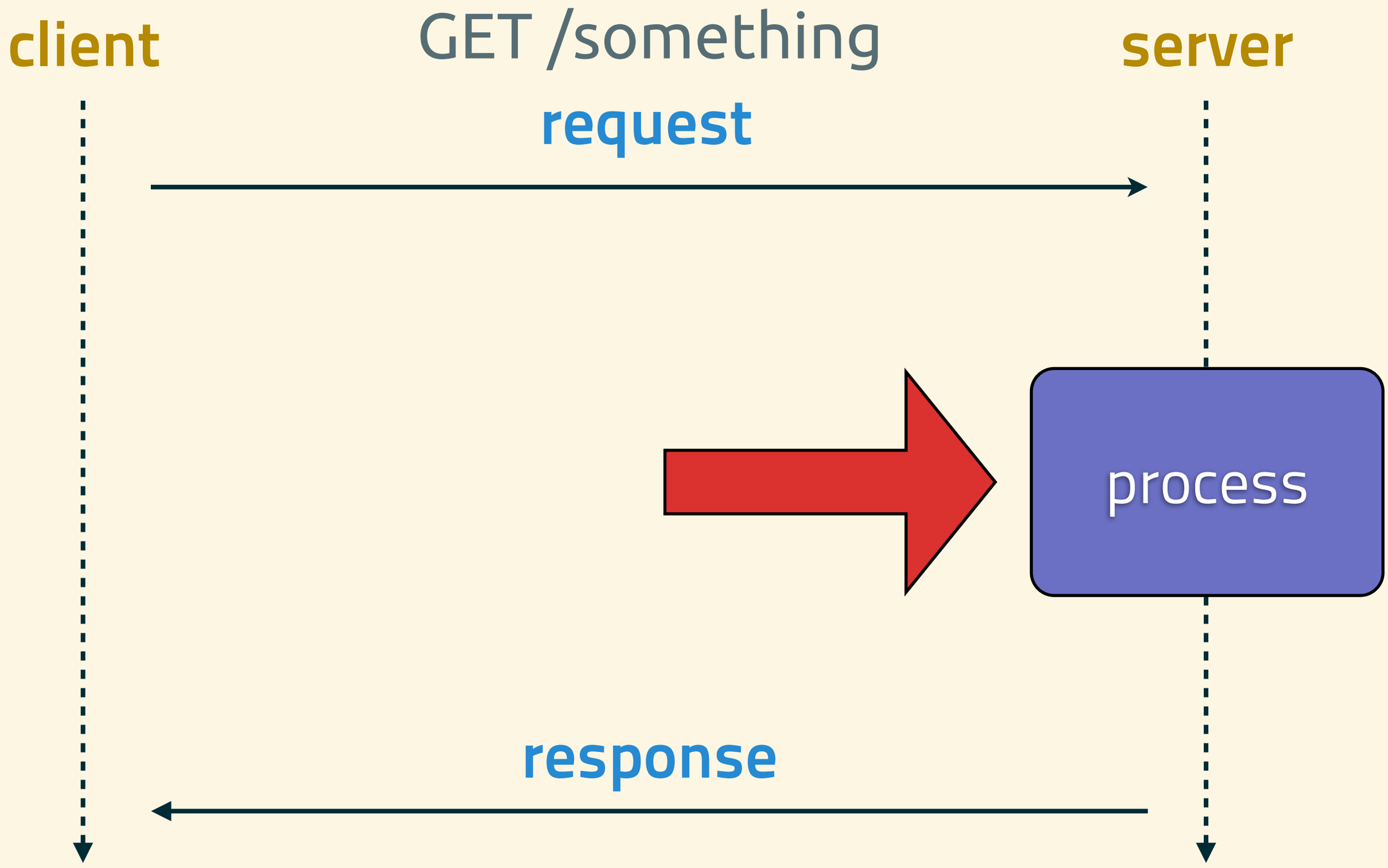
Have you ever...

CGI

Servlet

Model-2 “MVC”


client
GET /something
server
request
process
response


client
GET /something
server
request
process
response

Imperative : Actions

::

Functional : Facts

what does it DO?

Imperative : Actions

::

Functional : Facts

what does it DO?

Imperative : Actions

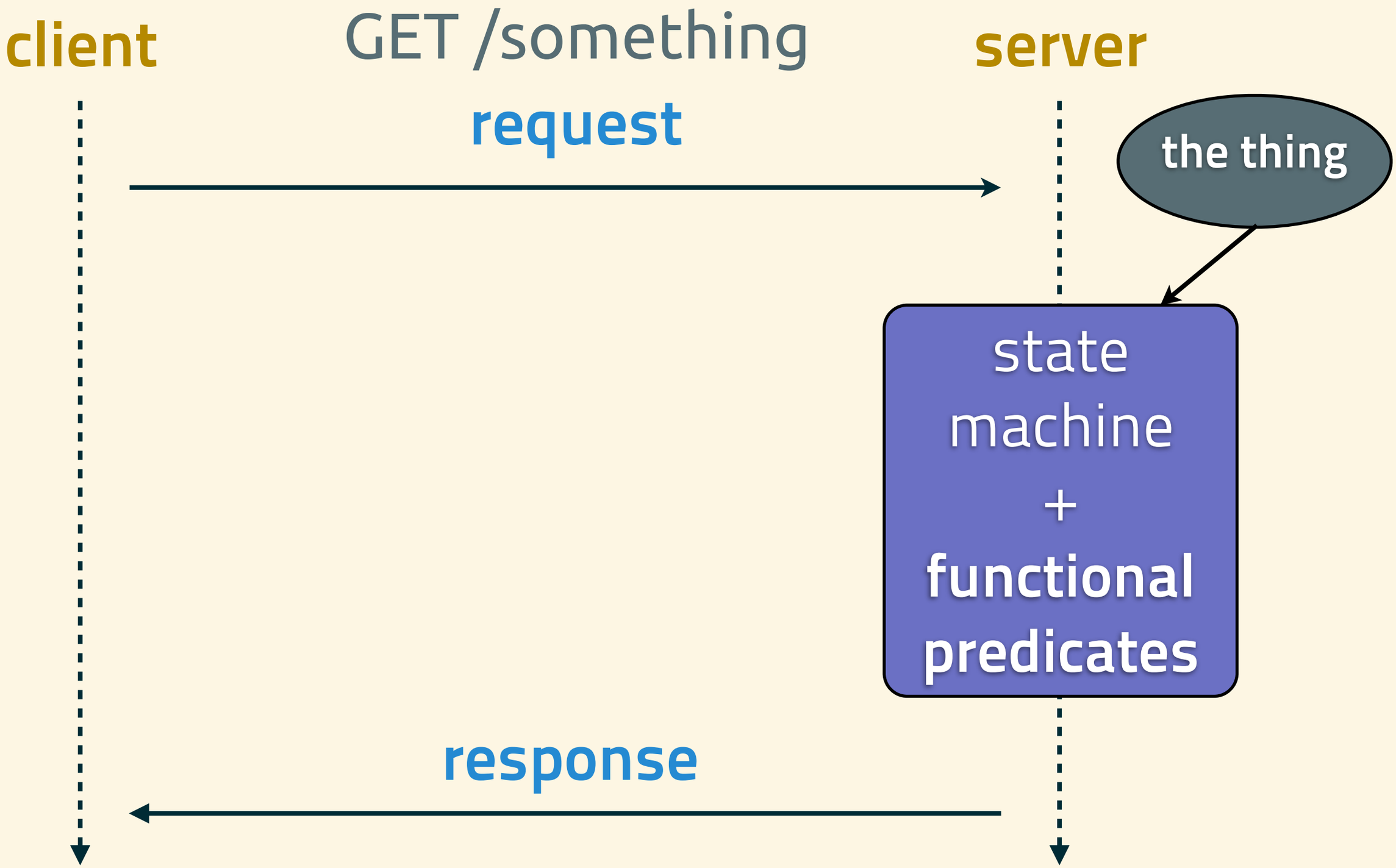
::

Functional : Facts

what IS it?

HTTP Facts: Resources

- Data or Service
- Identified by URI
- Decorated by representations and other properties/variances


client
GET /something
request
server
the thing
state machine + functional predicates
response

```
f(ReqData,State) -> {RetV,ReqData,State}.
```

function behavior + request/response data + resource state

Resource functions are **referentially transparent** and have a **uniform interface**.

Many resource functions are **optional** and use **reasonable defaults**.

```
f(ReqData,State) -> {RetV,ReqData,State}.
```

function behavior + request/response data + resource state

Resource functions are **referentially transparent** and have a **uniform interface**.

Many resource functions are **optional** and use **reasonable defaults**.

```
f(ReqData,State) -> {RetV,ReqData,State}.
```

function behavior + request/response data + resource state

Resource functions are **referentially transparent** and have a **uniform interface**.

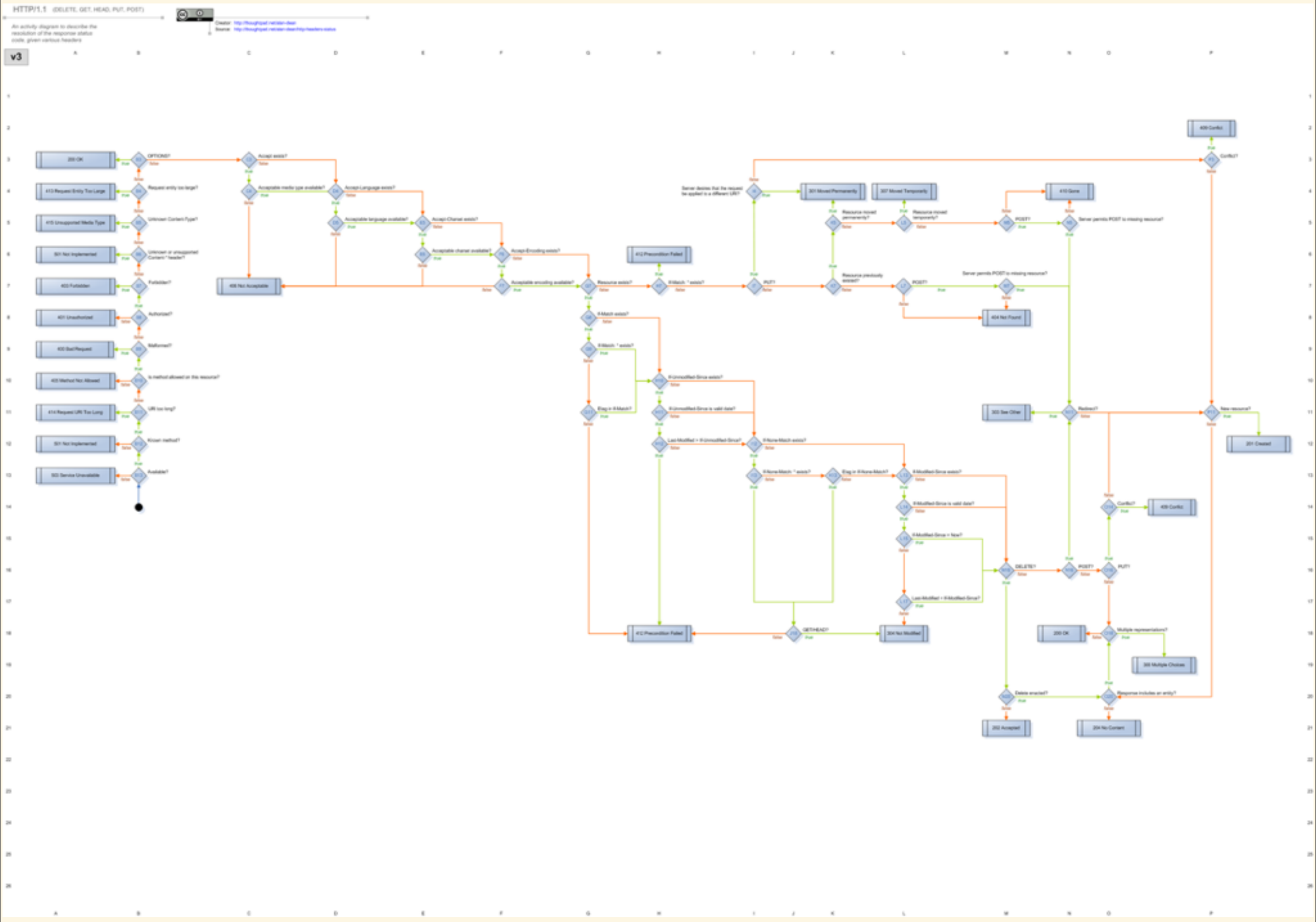
Many resource functions are **optional** and use **reasonable defaults**.

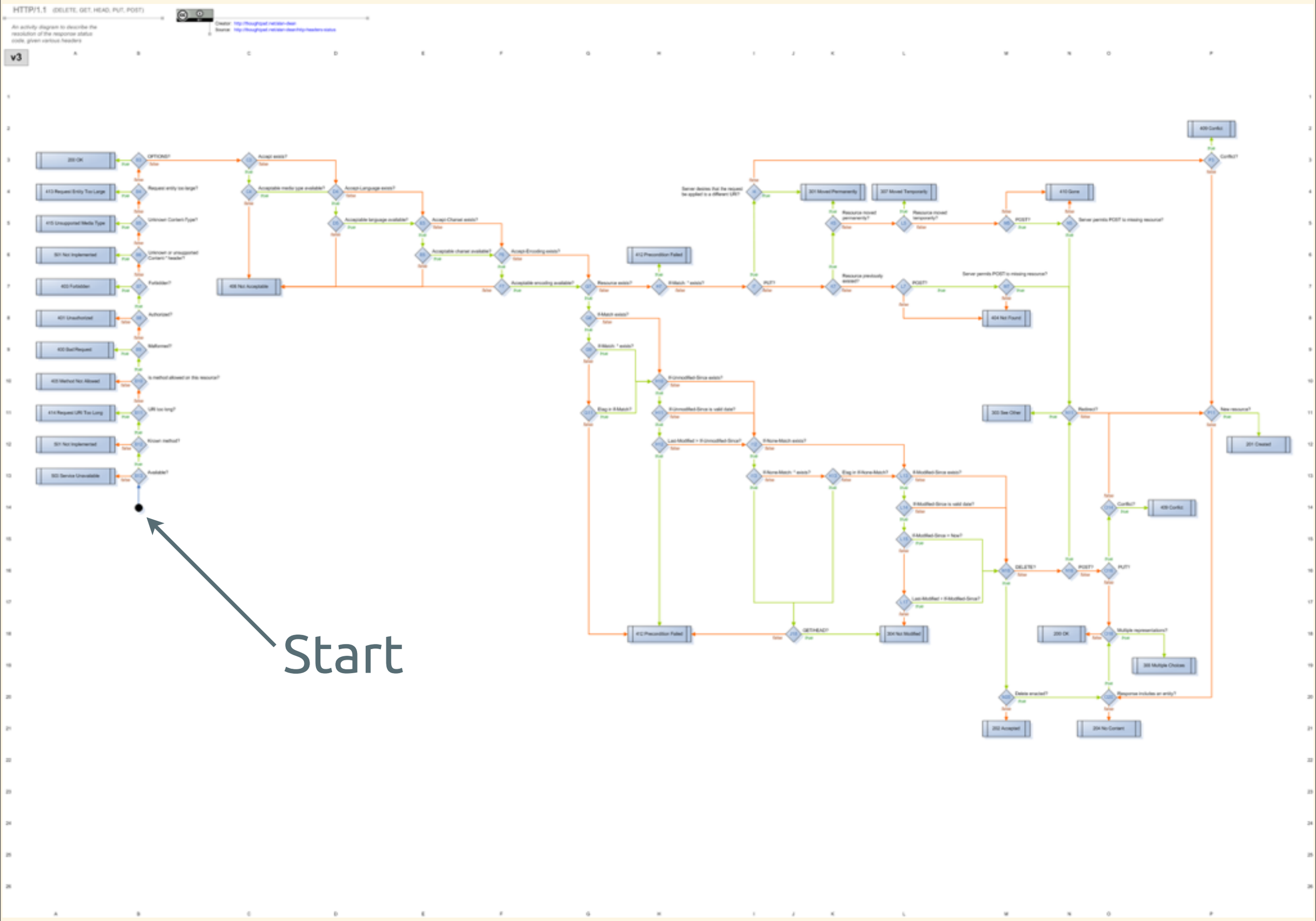
```
f(ReqData,State) -> {RetV,ReqData,State}.
```

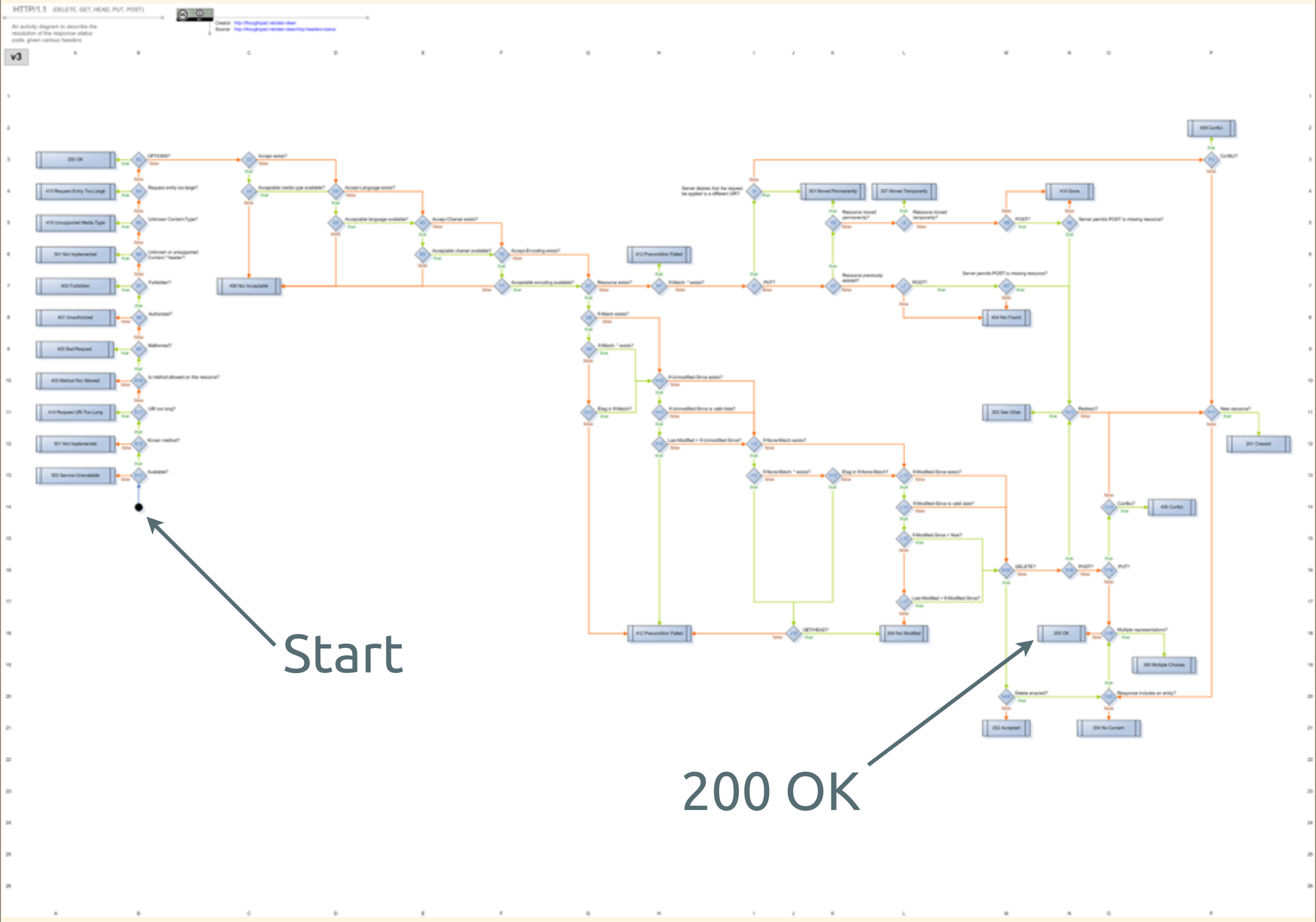
function behavior + request/response data + resource state

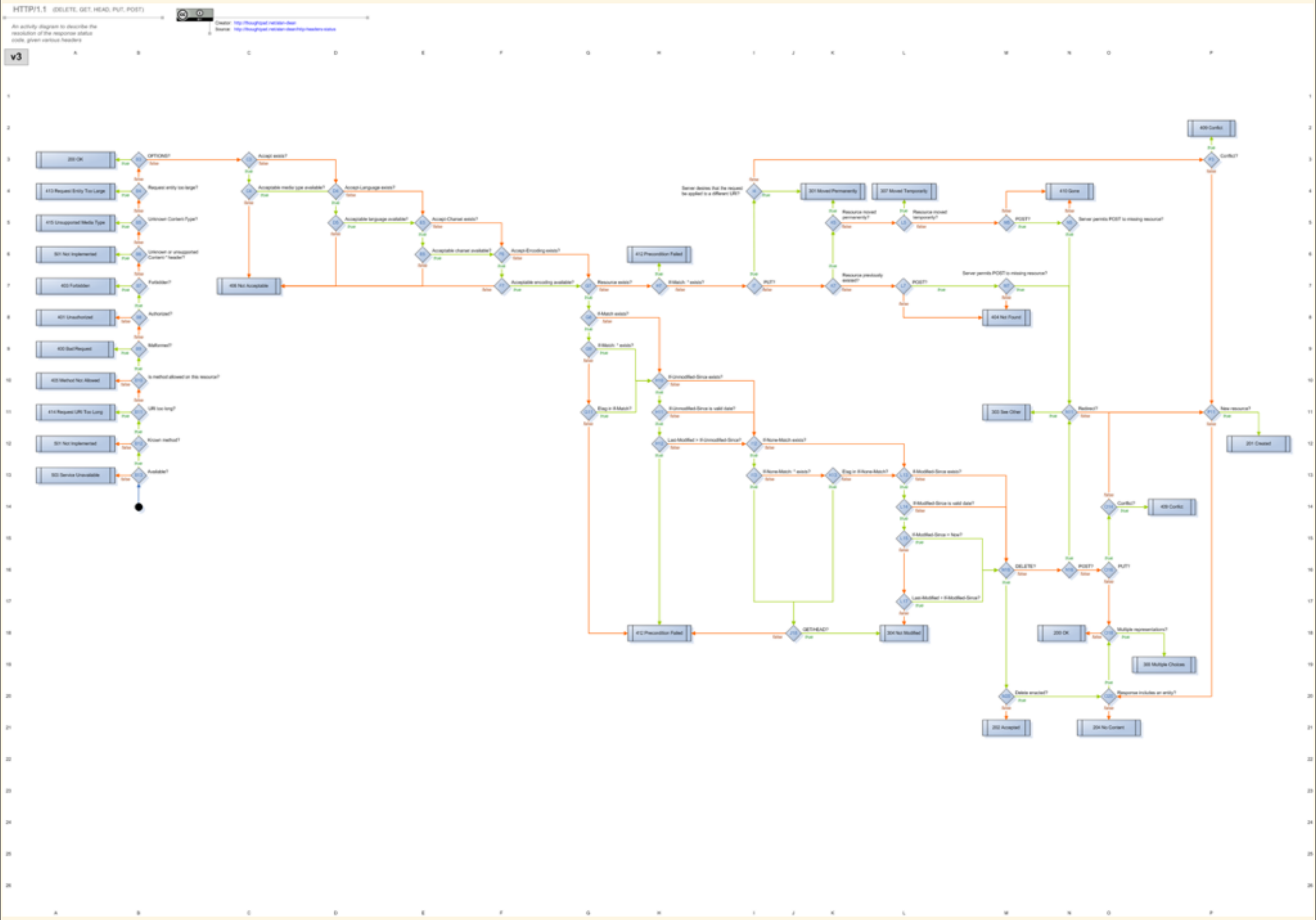
Resource functions are **referentially transparent** and have a **uniform interface**.

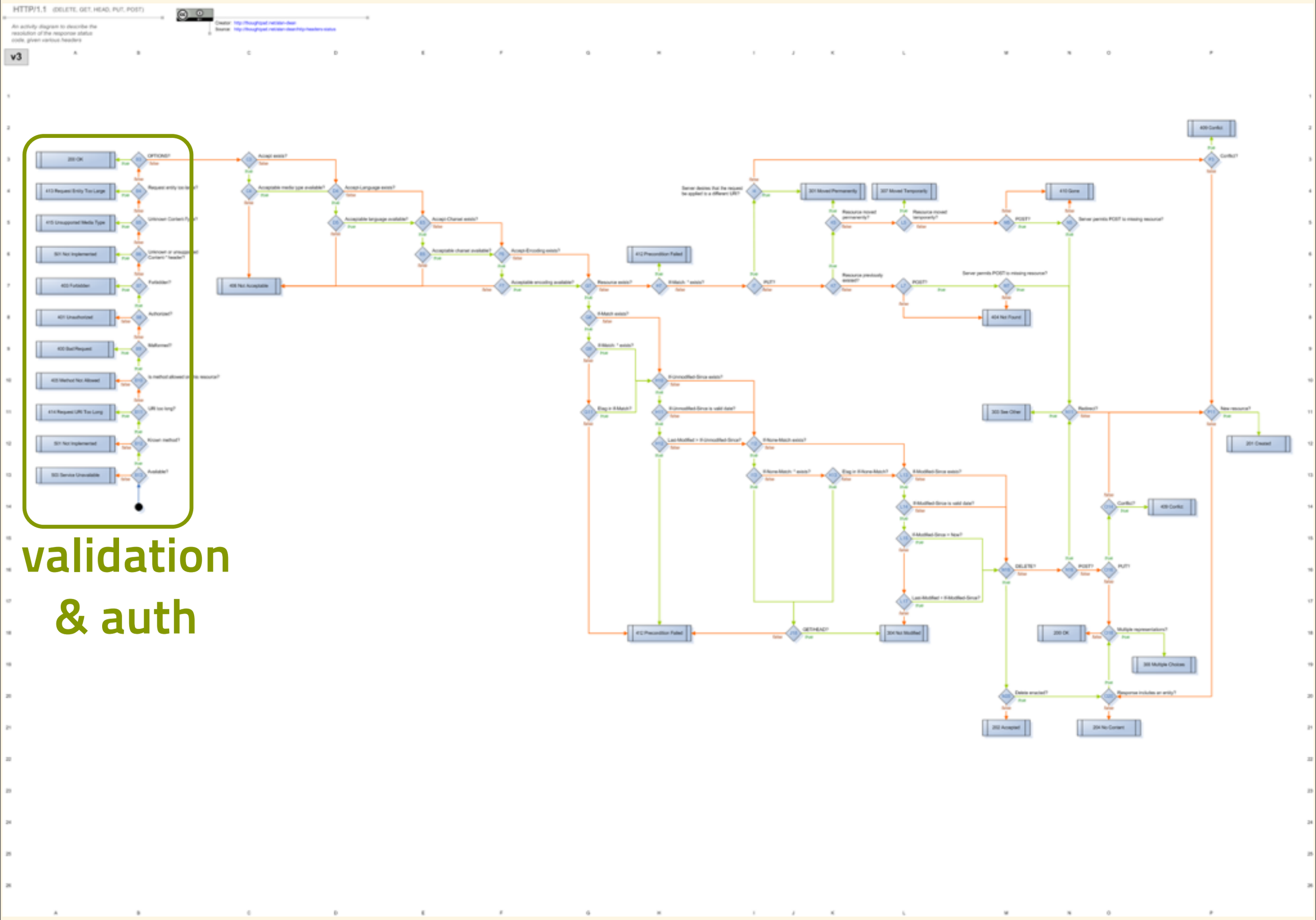
Many resource functions are **optional** and use **reasonable defaults**.

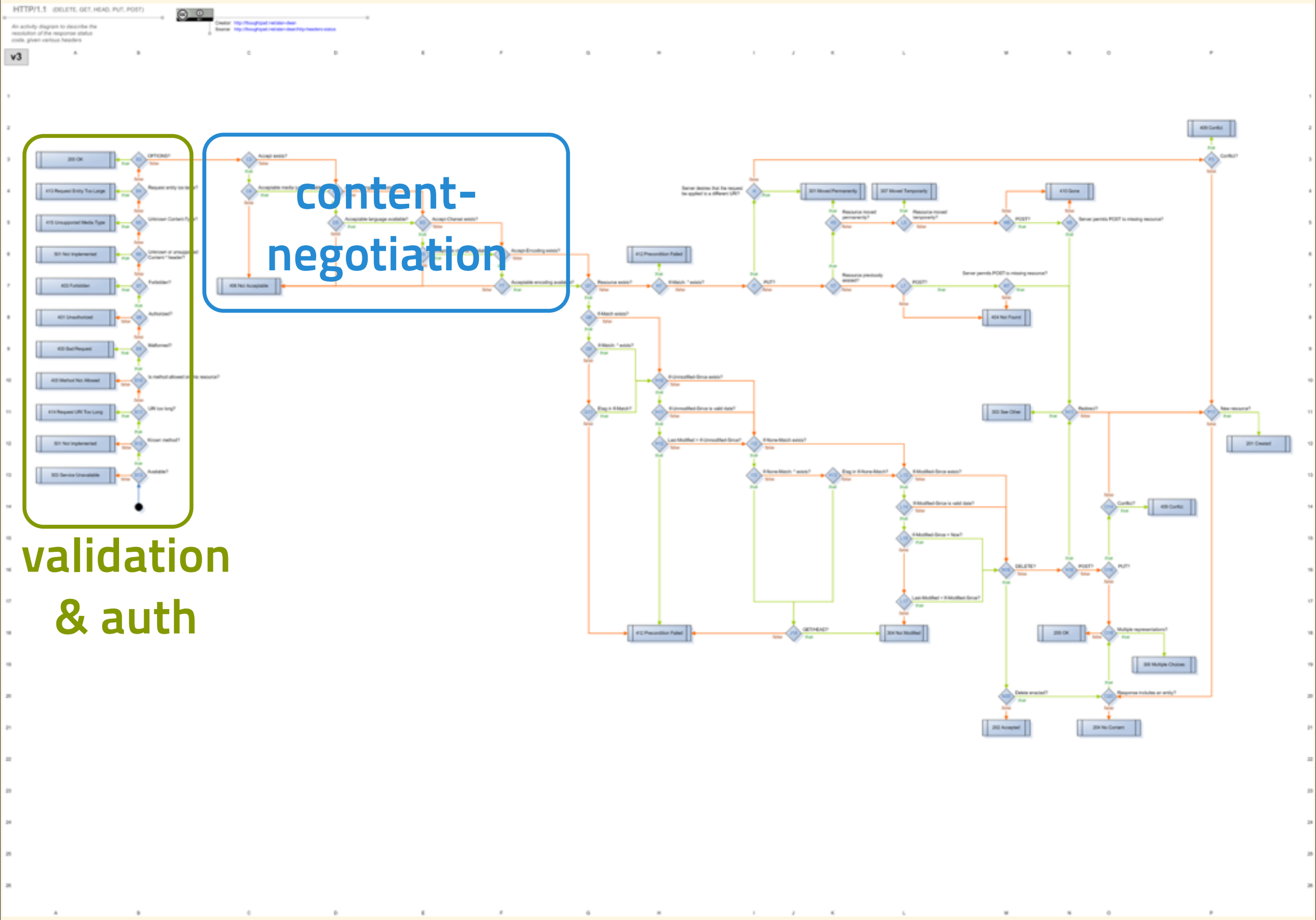

HTTP/1.1 (DELETE, GET, HEAD, PUT, POST)
An activity diagram to describe the resolution of the response status code, given various headers
v3
200 OK
OPTIONS?
413 Request Entity Too Large
Request entity too large?
415 Unsupported Media Type
Unknown Content-Type?
501 Not Implemented
Unknown or unsupported Content-* header?
403 Forbidden
Forbidden?
401 Unauthorized
Authorized?
400 Bad Request
Malformed?
405 Method Not Allowed
Is method allowed on this resource?
414 Request-URI Too Long
URI too long?
501 Not Implemented
Known method?
503 Service Unavailable
Available?
Accept exists?
406 Not Acceptable
Acceptable media type available?
Accept-Language exists?
Acceptable language available?
Accept-Charset exists?
Acceptable charset available?
Accept-Encoding exists?
Acceptable encoding available?
Resource exists?
412 Precondition Failed
If-Match: * exists?
If-Match exists?
If-Match: * exists?
If-Unmodified-Since exists?
If-Unmodified-Since is valid date?
Etag in If-Match?
Last-Modified > If-Unmodified-Since?
If-None-Match exists?
If-None-Match: * exists?
Etag in If-None-Match?
If-Modified-Since exists?
If-Modified-Since is valid date?
If-Modified-Since > Now?
Last-Modified > If-Modified-Since?
304 Not Modified
GET/HEAD?
Server desires that the request be applied to a different URI?
301 Moved Permanently
307 Moved Temporarily
Resource moved permanently?
Resource moved temporarily?
410 Gone
POST?
Server permits POST to missing resource?
Resource previously existed?
POST?
Server permits POST to missing resource?
404 Not Found
409 Conflict
Conflict?
PUT?
DELETE?
POST?
Conflict?
409 Conflict
Multiple representations?
300 Multiple Choices
200 OK
Delete enacted?
Response includes an entity?
202 Accepted
204 No Content
Redirect?
303 See Other
New resource?
201 Created

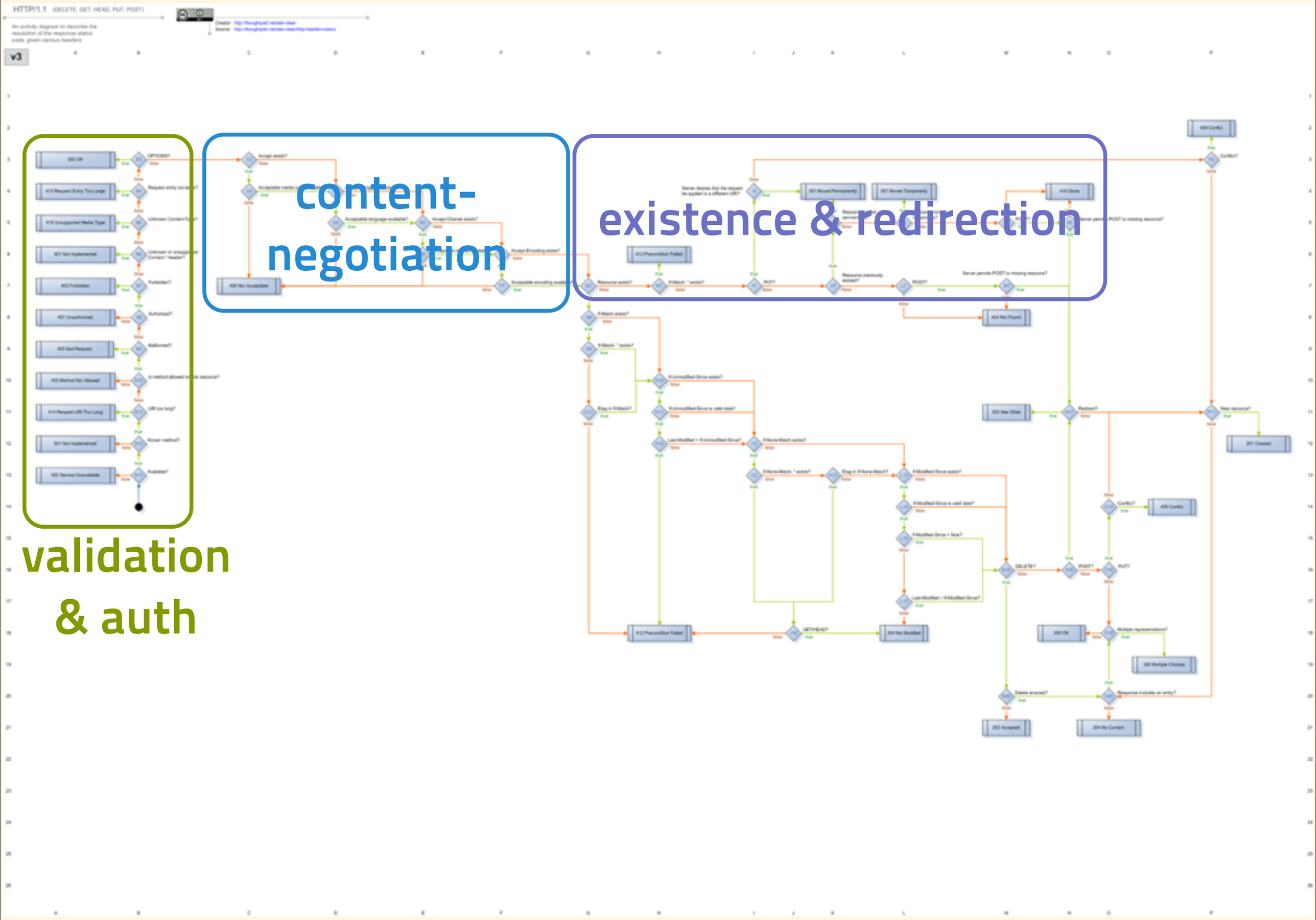

HTTP/1.1 (DELETE, GET, HEAD, PUT, POST)
An activity diagram to describe the resolution of the response status code, given various headers
v3
Start

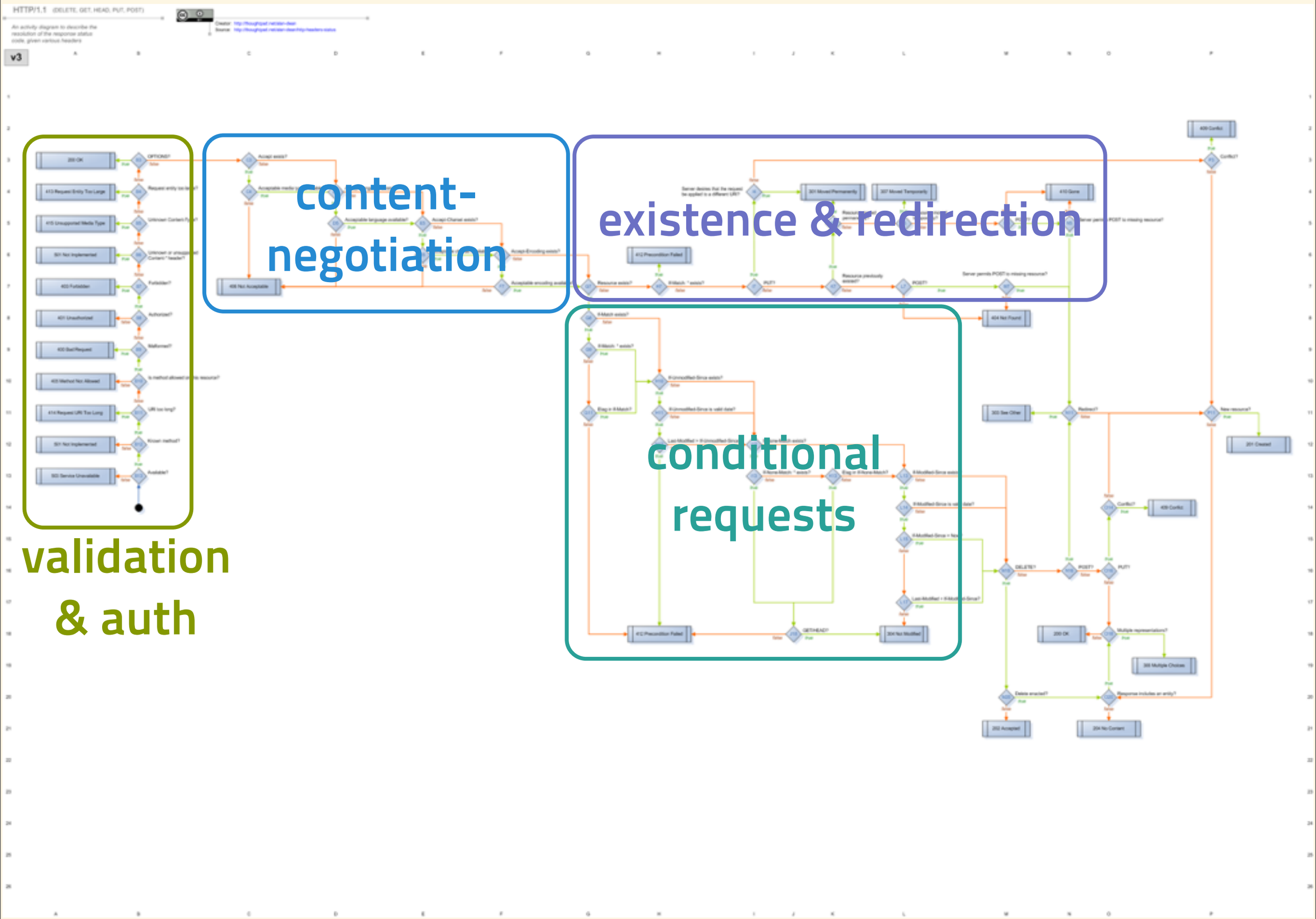

HTTP/1.1 (DELETE, GET, HEAD, PUT, POST)
An activity diagram to describe the resolution of the response status code, given various headers
v3
Start
200 OK

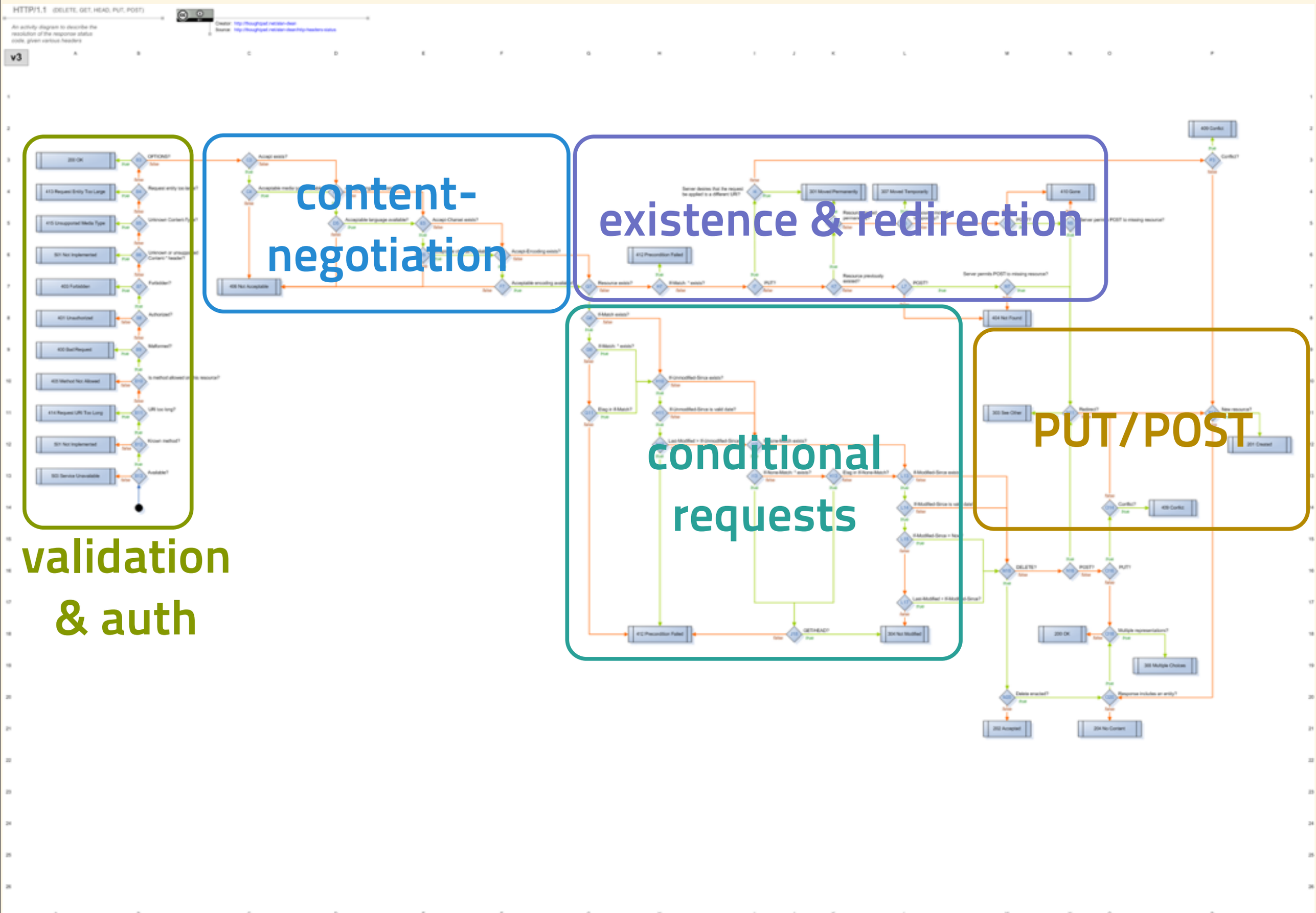

HTTP/1.1 (DELETE, GET, HEAD, PUT, POST)
An activity diagram to describe the resolution of the response status code, given various headers
v3
200 OK
OPTIONS?
413 Request Entity Too Large
Request entity too large?
415 Unsupported Media Type
Unknown Content-Type?
501 Not Implemented
Unknown or unsupported Content-* header?
403 Forbidden
Forbidden?
401 Unauthorized
Authorized?
400 Bad Request
Malformed?
405 Method Not Allowed
Is method allowed on this resource?
414 Request-URI Too Long
URI too long?
501 Not Implemented
Known method?
503 Service Unavailable
Available?
Accept exists?
406 Not Acceptable
Acceptable media type available?
Accept-Language exists?
Acceptable language available?
Accept-Charset exists?
Acceptable charset available?
Accept-Encoding exists?
Acceptable encoding available?
Resource exists?
412 Precondition Failed
If-Match: * exists?
If-Match exists?
If-Match: * exists?
If-Unmodified-Since exists?
If-Unmodified-Since is valid date?
Etag in If-Match?
Last-Modified > If-Unmodified-Since?
If-None-Match exists?
If-None-Match: * exists?
Etag in If-None-Match?
If-Modified-Since exists?
If-Modified-Since is valid date?
If-Modified-Since > Now?
Last-Modified > If-Modified-Since?
304 Not Modified
GET/HEAD?
Server desires that the request be applied to a different URI?
301 Moved Permanently
307 Moved Temporarily
Resource moved permanently?
Resource moved temporarily?
410 Gone
POST?
Server permits POST to missing resource?
Resource previously existed?
POST?
Server permits POST to missing resource?
404 Not Found
409 Conflict
Conflict?
DELETE?
POST?
PUT?
Conflict?
409 Conflict
200 OK
Multiple representations?
300 Multiple Choices
Delete enacted?
Response includes an entity?
202 Accepted
204 No Content
303 See Other
Redirect?
New resource?
201 Created

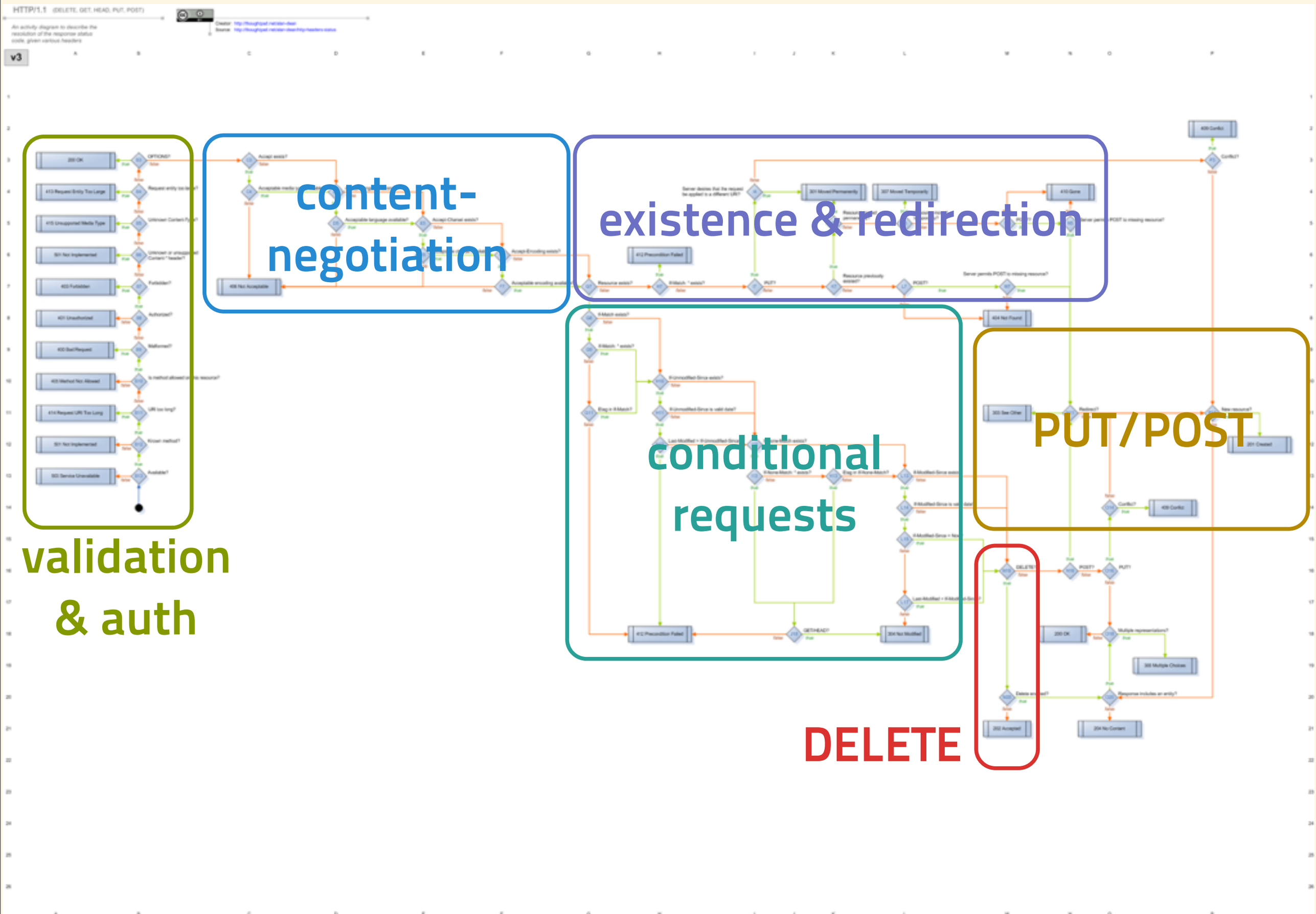

HTTP/1.1 (DELETE, GET, HEAD, PUT, POST)
An activity diagram to describe the resolution of the response status code, given various headers
v3
validation
& auth

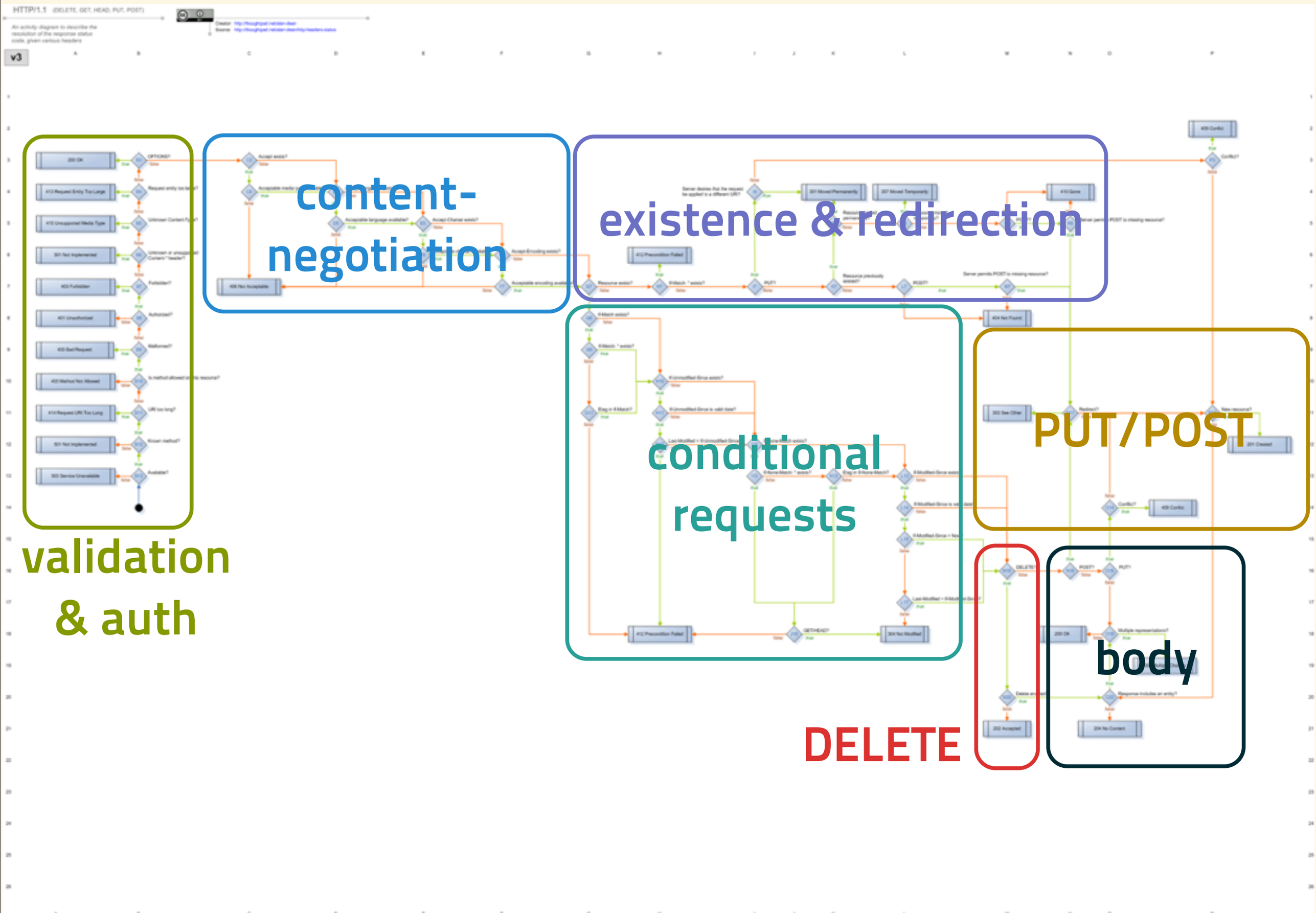

HTTP/1.1 (DELETE, GET, HEAD, PUT, POST)
An activity diagram to describe the resolution of the response status code, given various headers
v3
content-negotiation
validation & auth


HTTP/1.1 (DELETE, GET, HEAD, PUT, POST)
v3
content-negotiation
existence & redirection
validation & auth


HTTP/1.1 (DELETE, GET, HEAD, PUT, POST)
v3
content-negotiation
existence & redirection
conditional requests
validation & auth


HTTP/1.1 (DELETE, GET, HEAD, PUT, POST)
v3
validation & auth
content-negotiation
existence & redirection
conditional requests
PUT/POST


HTTP/1.1 (DELETE, GET, HEAD, PUT, POST)
v3
content-negotiation
existence & redirection
conditional requests
PUT/POST
validation & auth
DELETE


HTTP/1.1 (DELETE, GET, HEAD, PUT, POST)
v3
validation & auth
content-negotiation
existence & redirection
conditional requests
PUT/POST
DELETE
body

Hello, World

```
$ git checkout hello-world
$ $EDITOR src/tweeter_resource.erl
```

Build & Run

```
$ make
$ ./start.sh
$ $BROWSER http://localhost:8080
```

Also Heroku compatible!
(use `foreman start`)

Default resource

```
init([]) ->
    {ok, undefined}.

to_html(ReqData, State) ->
    {"<html><body>Hello, new world</body></html>",
     ReqData, State}.
```

Default resource

resource
state

```
init([]) ->
    {ok, undefined}.

to_html(ReqData, State) ->
    {"<html><body>Hello, new world</body></html>",
     ReqData, State}.
```

Default resource

```
init([]) ->
    {ok, undefined}.

to_html(ReqData, State) ->
    {"<html><body>Hello, new world</body></html>",
     ReqData, State}.
```

resource state

iolist()

Exercises

- Use the resource state as the body, setting it in `init/1`
- Put the value of the **`Host`** request header inside the response body using an `iolist()` and:

```
wrq:get_req_header(Key, ReqData)
```

* Hint: header keys are lowercase strings

UI Skeleton

```
<Ctrl-C> a  # if still running
$ git checkout -f assets
$ make; ./start.sh
$ $BROWSER http://localhost:8080
```

We'll come back to the resource at the end.

Load Tweets in UI

```
<Ctrl-C> a  # if still running
$ git checkout -f load-tweets
$ make; ./start.sh
<refresh browser>
```

Media Types

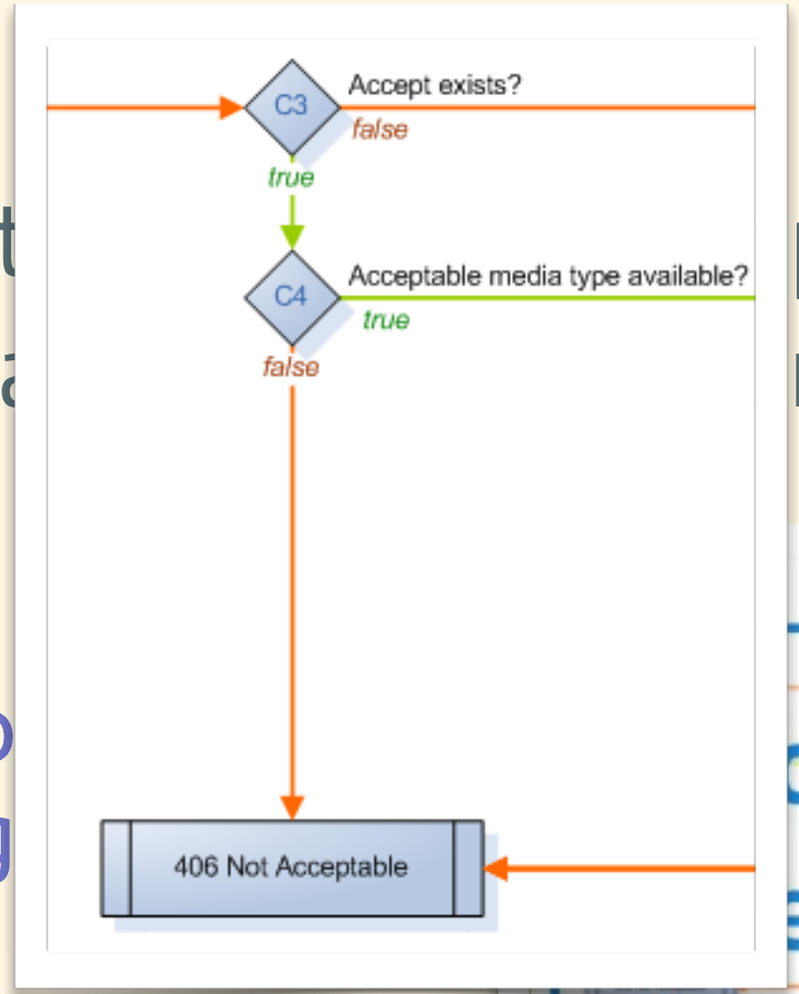
...specify alternative or multiple formats ("representations") for resources:

```
text/html
application/json
image/jpeg
```

Media Types

...specify alt ple formats
(“representa rces:

text/html
applicatio
image/jpeg



Media-types callback

```
%% default implementation
content_types_provided(ReqData, State) ->
    {[{"text/html", to_html}], ReqData, State}.
```

media type

body-producer function

```
%% As many types as you want!
content_types_provided(ReqData, State) ->
    {[{"text/html", to_html},
      {"application/json", to_json},
      {"text/xml", to_xml}], ReqData, State}.
```

Dispatching

```
%% tweeter_wm_tweet_resource.erl
routes() ->
    [{["tweets"], ?MODULE, []}].
```

```
%% tweeter_wm_asset_resource.erl
routes() ->
    [{[""], ?MODULE, []},
     {['*'], ?MODULE, []}].
```

matches any number of segments

Dispatching

```
%% tweeter_wm_tweet_resource.erl
routes() ->
    [{["tweets"], ?MODULE, []}].
```

path segments

```
%% tweeter_wm_asset_resource.erl
routes() ->
    [{[""], ?MODULE, []},
     {['*'], ?MODULE, []}].
```

matches any number of segments

Dispatching

```
%% tweeter_wm_tweet_resource.erl
routes() ->
    [{["tweets"], ?MODULE, []}].
```

path segments

resource module

```
%% tweeter_wm_asset_resource.erl
routes() ->
    [{[""], ?MODULE, []},
     {['*'], ?MODULE, []}].
```

matches any number of segments

Dispatching

```
%% tweeter_wm_tweet_resource.erl
routes() ->
    [{["tweets"], ?MODULE, []}].
```

path segments

resource module

args to `init/1`

```
%% tweeter_wm_asset_resource.erl
routes() ->
    [{[""], ?MODULE, []},
     {['*'], ?MODULE, []}].
```

matches any number of segments

Dispatching

```
%% tweeter_wm_tweet_resource.erl
routes() ->
    [{["tweets"], ?MODULE, []}].
```

path segmen

init/1

The dispatch list is set before starting up the server in `tweeter_sup`.

```
%% tweeter_wm_asset_resource.erl
routes() ->
    [{[""], ?MODULE, []},
     {['*'], ?MODULE, []}].
```

matches any number of segments

Records

```
%% definition
-record(context, {tweet, tweets}).

%% construction
#context{}.
    %% {context, undefined, undefined}
#context{tweets=[a,b,c]}.
    %% {context, undefined, [a,b,c]}

%% pattern-matching and destructuring
#context{tweets=Tweets} = Context.
    %% {context, _, Tweets} = Context.
Tweets = Context#context.tweets.
    %% Tweets = element(3,Context).

%% modification
NewContext = Context#context{tweet="foo"}.
    %% NewContext = setelement(1, Context, "foo").
```

ETS

```
%% create a table
{ok, TableID} = ets:new(table, [set]).

%% make it public, multi-reader/writer
{ok, TableID} =
    ets:new(table, [set, public, named_table,
                    {read_concurrency, true},
                    {write_concurrency, true}]).

%% read (lookup by key)
ets:lookup(TableID, a).

%% write
ets:insert(TableID, {foo, bar}).

%% query with abstract patterns
ets:match(TableID, {'$1', bar}).
```

Exercises

- Use `curl` to `GET /tweets`
- Change the **`Accept`** header (`-H` option) to exclude **`application/json`**, compare the response.
- Add the **`application/x-erlang-binary`** format to the resource. Use **`term_to_binary/1`** to generate the body.

Unique Tweet URLs

```
<Ctrl-C> a  # if still running
$ git checkout -f tweet-urls
$ make; ./start.sh
```

Resource exists?

404 Not Found
Redirection
Creation

200 OK
Condition validation
Deletion
Update / replace

Usually used for fetching the internal representation of the resource.

resource_exists

```
%% default implementation
resource_exists(ReqData, State) ->
    {true, ReqData, State}.

%% Dispatch rule that binds a path segment
%% to the atom ‘key’
routes() ->
    [{["data", key], ?MODULE, []}].

%% Do a query to get the data using the
%% bound dispatch path segment
resource_exists(ReqData, State) ->
    Key = wrq:path_info(key, ReqData),
    case query_storage(Key) of
        undefined ->
            {false, ReqData, State};
        Value ->
            {true, ReqData, State#state{data=Value}}
    end.
```

Exercises

- Find the identifier of a tweet in the JSON, request the composite URL with `curl`.
- Use `curl` to request a tweet that doesn't exist.

Create Tweets

```
$ git checkout -f tweet-urls
```

Creating Resources: PUT vs. POST

Idempotent	Non-Idempotent
Client-specified URI	Server-generated URI
204 No Content	201 Created

POSTing Resources

1. Allow the POST method
2. Specify that POST creates new resources
3. Generate a URL for the new resource
4. Accept the request body

Steps 1-3

```
%% 1. Allow the POST method
%% Default is ['HEAD', 'GET']
allowed_methods(ReqData, Context) ->
    {['HEAD', 'GET', 'POST'], ReqData, Context}.


%% 2. Specify that POST creates new resources
%% Default is false
post_is_create(ReqData, Context) ->
    {true, ReqData, Context}.


%% 3. Generate a URL for the new resource
create_path(ReqData, Context) ->
    NewID = ets:update_counter(table, curr_id, {2, 1}),
    {"/steps/" ++ integer_to_list(NewID),
     ReqData, Context#context{id=NewID}}.
```

Step 4

```
%% 4. Accept the request body
%% 4a. Specify the acceptable media types
content_types_accepted(ReqData, Context) ->
    {[{"application/json", accept_json}],
     ReqData, Context}.

%% 4b. Accept the negotiated type
accept_json(ReqData, Context) ->
    Body = wrq:req_body(),
    {struct, Props} = mochijson2:decode(Body),
    ok = store(Context#context.id, Props),
    {true, ReqData, Context}.
```

Exercises

- Use the browser UI to post tweets
- Post a tweet using `curl`...
  - Sending a JSON body
  - Sending a non-JSON body

*Hint: use `-H` and `Content-Type`

Streaming Responses

```
$ git checkout -f stream-pg2
```

Why Stream?

- Less buffering, memory usage
- Reduced latency, partial results
- Long-lived connections

Streaming Responses

```
%% Before
to_text(ReqData, Context) ->
    {"Hello, world!", ReqData, Context}.

%% After
to_text(ReqData, Context) ->
    {{stream, {<<>>, fun stream/0}}, ReqData, Context}.

%% Stream response as a "lazy sequence", with the
%% Webmachine process waiting on messages.
stream() ->
    receive
        {chat, Message} ->
            {[Message, "\n"], fun stream/0};
        quit ->
            {<<>>, done}
    end.
```

OTP: Process Groups

```
%% Create a process group
ok = pg2:create(chat).

%% Get members of the process group
Members = pg2:get_members(chat).

%% Join the process group
pg2:join(chat, self()).

%% Send a message to all members
[Member ! {chat, Msg} || Member <- Members].
```

Exercises

- Find the bug in the streaming response and fix it.
  *Hint: http://www.erlang.org/doc/man/erlang.html
- Add a new streaming response that uses HTML5 `text/event-stream` instead of `multipart/mixed`.

Caching and Preconditions

```
$ git checkout -f etag-tweets
```

HTTP Caching

- **Expiration:** `Cache-Control` + `max-age` (TTL), `Expires` (Date)
- **Validation**: `ETag`, `Last-Modified`, `If-*`
  - 304 Not Modified
  - Computing `ETag` and `Last-Modified` should be **cheap**

generate_etag & last_modified

```
%% Default is undefined, i.e. no ETag
%% Compute some hash, convert it to a hex string
generate_etag(ReqData, Context) ->
    ETag = mochihex:to_hex(erlang:phash2(Context)),
    {ETag, ReqData, Context}.
```

```
%% Default is undefined, i.e. no timestamp
%% Return some {{Y,M,D},{H,M,S}} tuple:
last_modified(ReqData, Context) ->
    {ok, #file_info{mtime=Date}} =
        file:read_file_info("somefile"),
    {Date, ReqData, Context}.
```

Exercises

- Fetch the tweets with `curl`, copy the `ETag` from response, fetch again with `If-None-Match` header.
- Add a tweet via the UI, send same `curl` request as last step.
- Add a `last_modified` callback, using ID of the latest tweet as the timestamp.

Templating

```
$ git checkout -f template
```

erlydtl

- Resembles Django's template language
- Compiles the template into an Erlang module
- `templates/foo.dtl` -> `foo_dtl` module

Exercises

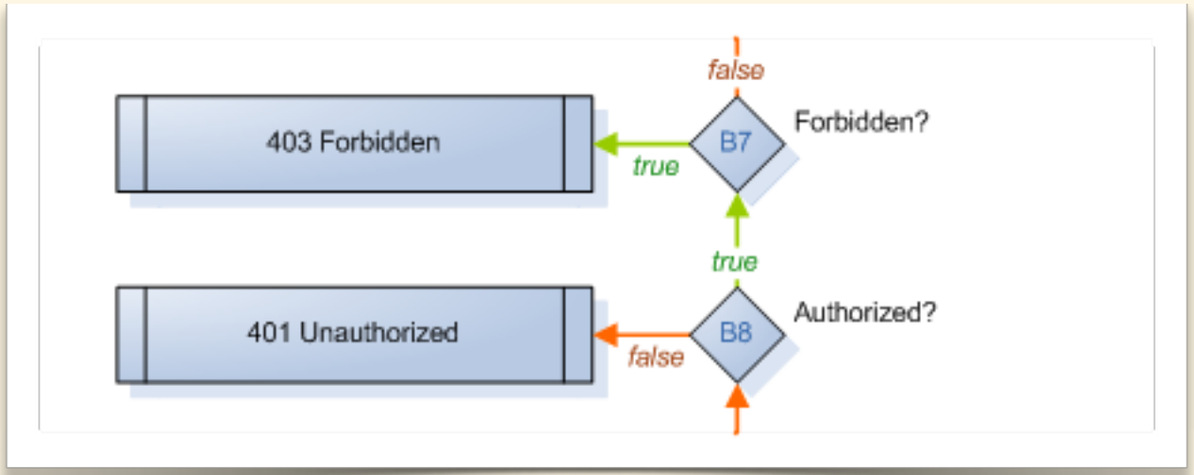
- Edit the template to change some text, recompile and refresh the browser.

Authorization & CSRF

```
$ git checkout -f csrf
```

Authorization

- 401 Unauthorized
  `Authorization`
  `WWW-Authenticate`
- 403 Forbidden
  - GTFO



is_authorized & forbidden

```
%% Defaults to true
%% If unauthorized, return a challenge string for 401
is_authorized(ReqData, Context) ->
    Auth = wrq:get_req_header("authorization", ReqData),
    case check_auth(Auth) of
        true ->
            {true, ReqData, Context};
        false ->
            {"Basic realm=\"Webmachine\"",
             ReqData, Context}
    end.

%% Defaults to false, return true for 403
forbidden(ReqData, Context) ->
    {true, ReqData, Context}.
```

Exercises

- Modify the CSRF protection to protect `DELETE` requests.
- Attempt to launch a CSRF attack to create a tweet!

Dialyzer

```
$ git checkout -f dialyzer
```

Dialyzer

- Erlang is **dynamically-typed**, but most functions have **specific parameter and return types.**
- Many bugs can be found by **static analysis** using **Dialyzer.**
- Annotations are also **documentation**.

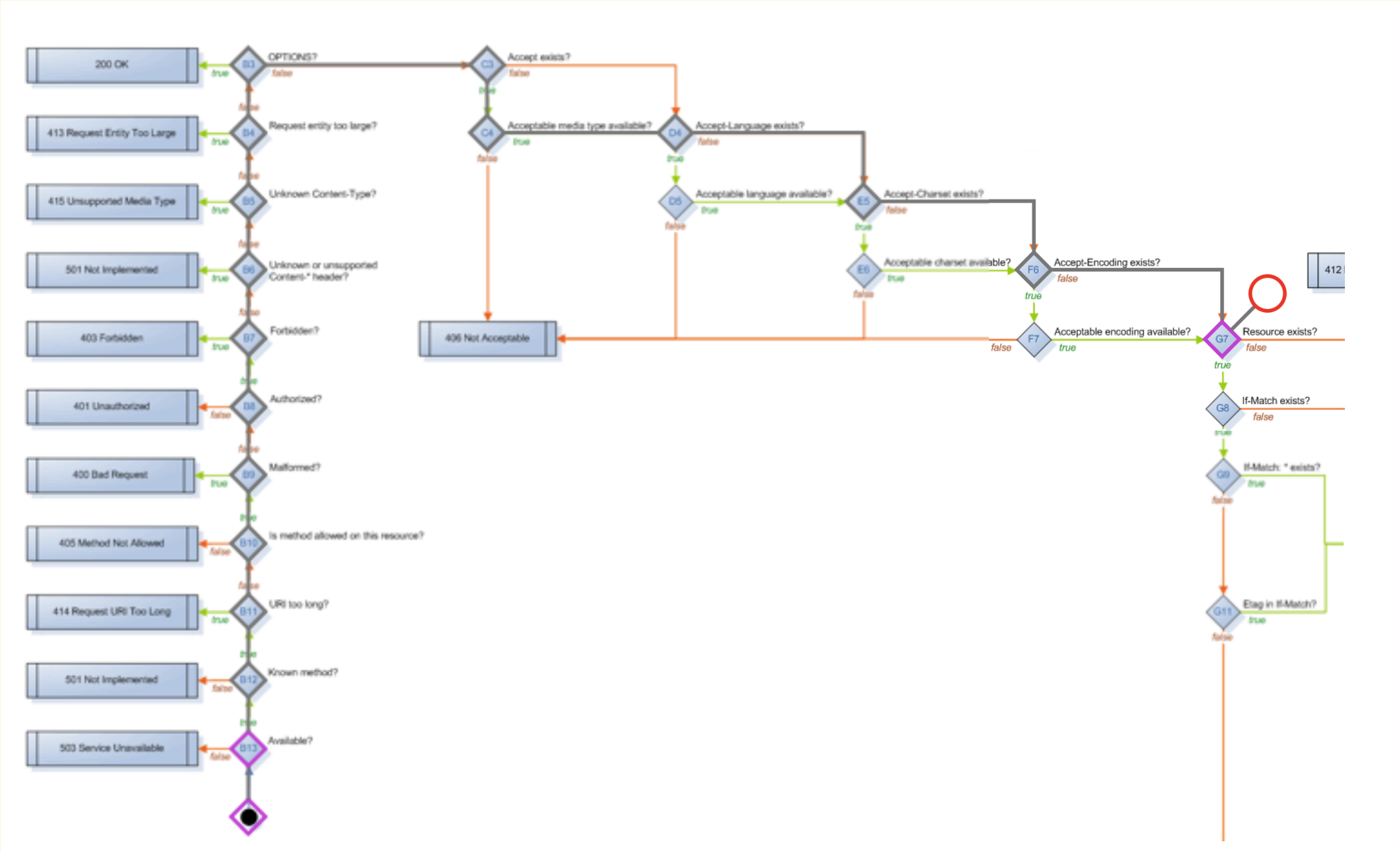
Exercises

- Run dialyzer using the `make` target.
- Break a function's types, compile, and see if dialyzer will catch it.

Visual Debugger

```
$ git checkout -f debugger
```

Debugger


200 OK
B3
OPTIONS?
C3
Accept exists?
413 Request Entity Too Large
B4
Request entity too large?
C4
Acceptable media type available?
D4
Accept-Language exists?
415 Unsupported Media Type
B5
Unknown Content-Type?
D5
Acceptable language available?
E5
Accept-Charset exists?
501 Not Implemented
B6
Unknown or unsupported Content-* header?
E6
Acceptable charset available?
F6
Accept-Encoding exists?
412
403 Forbidden
B7
Forbidden?
406 Not Acceptable
F7
Acceptable encoding available?
G7
Resource exists?
401 Unauthorized
B8
Authorized?
G8
If-Match exists?
400 Bad Request
B9
Malformed?
G9
If-Match: * exists?
405 Method Not Allowed
B10
Is method allowed on this resource?
414 Request URI Too Long
B11
URI too long?
G11
Etag in If-Match?
501 Not Implemented
B12
Known method?
503 Service Unavailable
B13
Available?
true
false

Enabling tracing

enable the trace resource

```
%% Initialize the resource, but enable tracing.
init([]) ->
    wmtrace_resource:add_dispatch_rule("wmtrace", "/tmp"),
    {{trace, "/tmp"}, #context{}}.
```

trace storage

Exercises

- Open the visual debugger at `localhost:8080/wmtrace`
- Refresh the root URL, find the bug in the resource and fix it!

Asset Resource

```
$ git checkout -f assets-final
```

Asset Resource

- Catch-all dispatch rule
- Renders erlydtl template
- ETag & Last-Modified
- Checks file existence
- Infers media type from file
- Adds CSRF token/cookie

```
{<<“class”>>, done}
```