

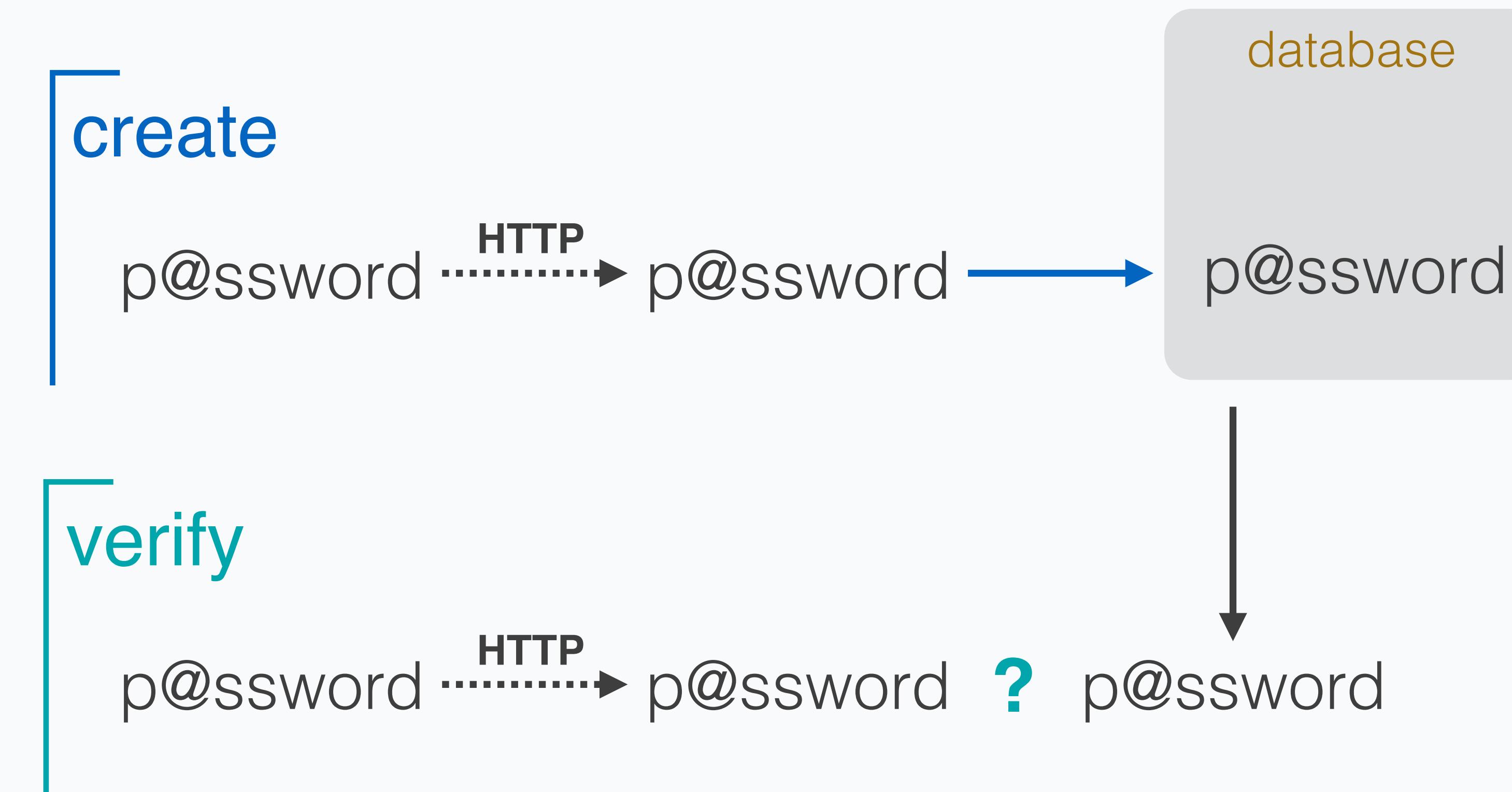
auth: data flow

Not web security!

AAA

- Authentication
 - “this person is who they say they are”
- Authorization
 - “this person is allowed to do X, Y, Z”
- Accounting
 - “who the heck is using all our bandwidth?”

login/signup



staying logged in

The Problem

What if we want to store some information about each client/server relationship (session)?

Ideas?

- script variables
- database documents

script variables?

Data will not persist across browser pages

Database Documents?

*Yes, but login credentials would have to be sent
with EVERY request*

COOKIE!



with cookies



= small text file

HTTP

client

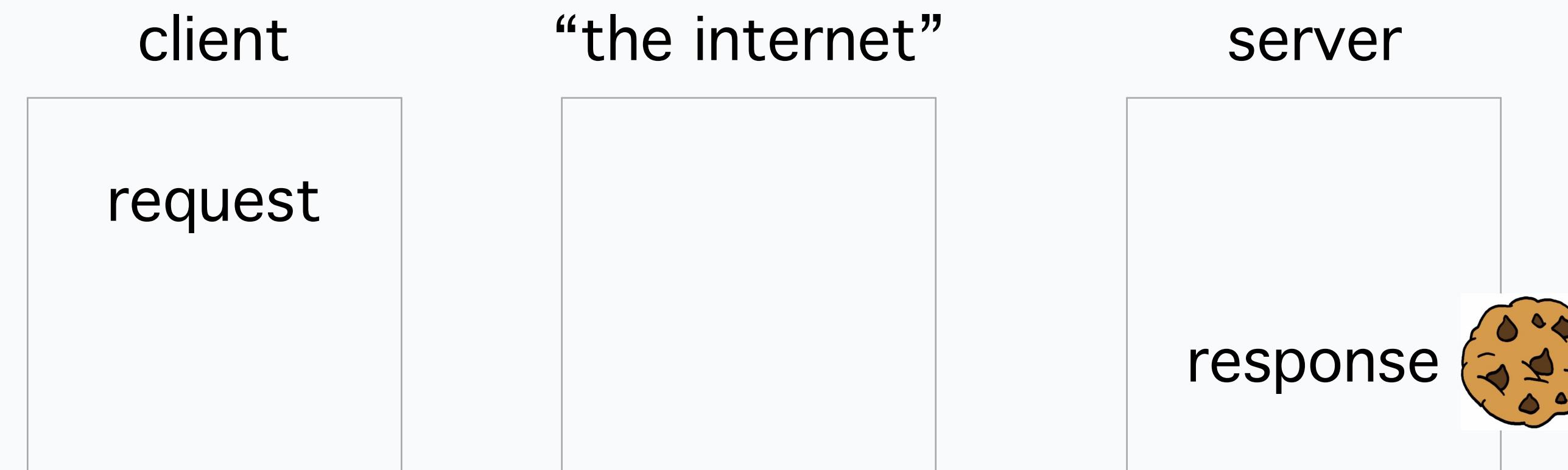
request

“the internet”

server

response

with cookies

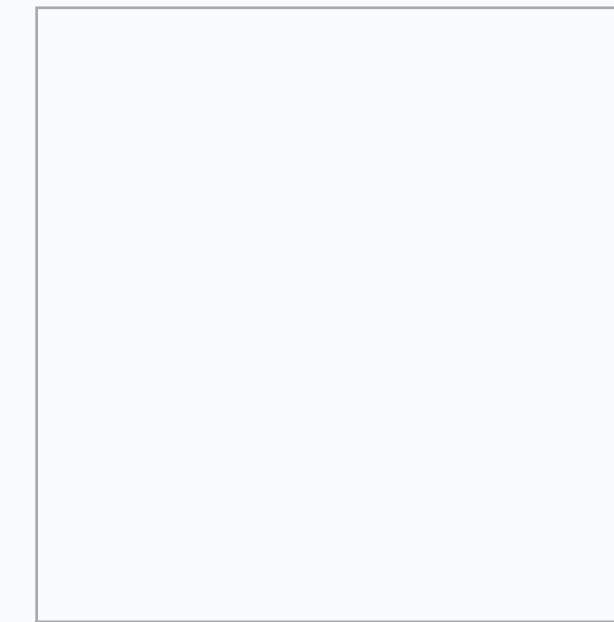


with cookies

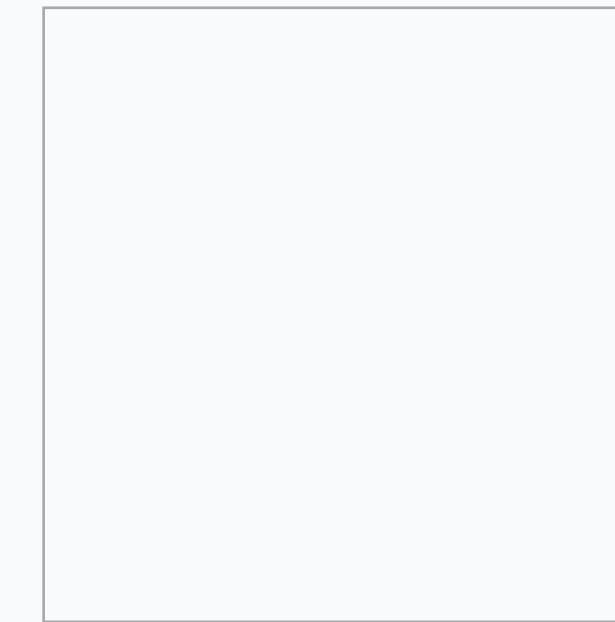
client



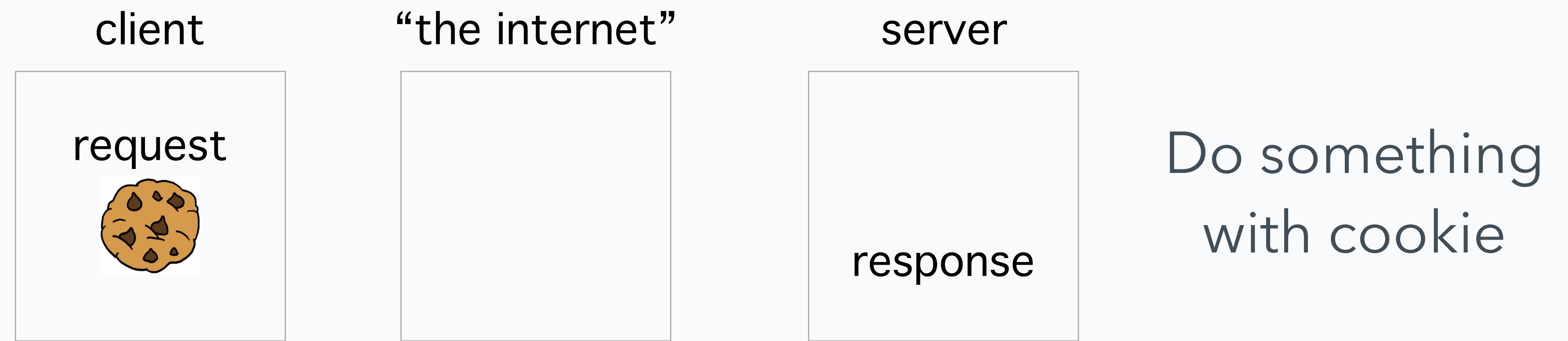
“the internet”



server



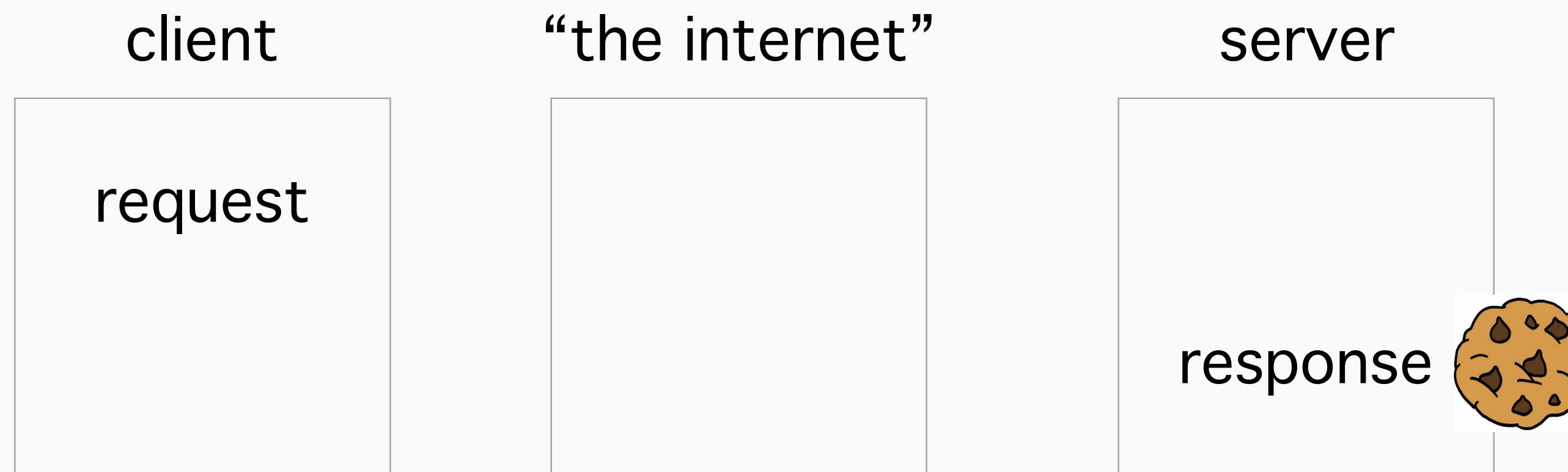
with cookies



cookies & sessions

No need to authenticate for every request

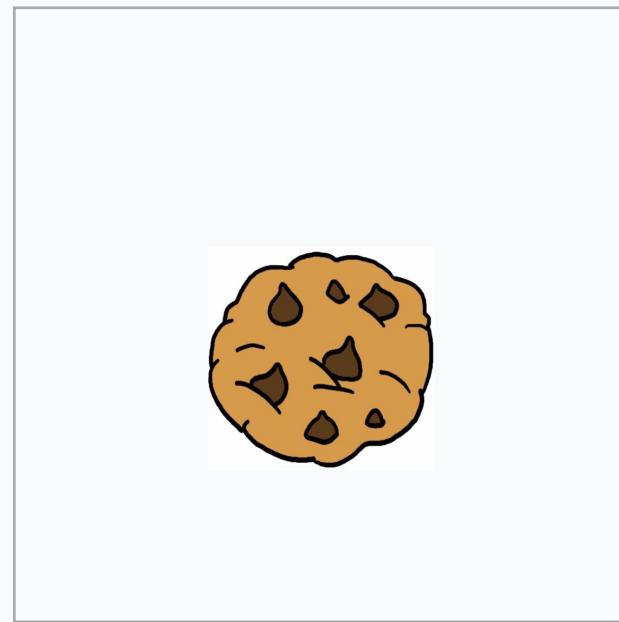
sessions



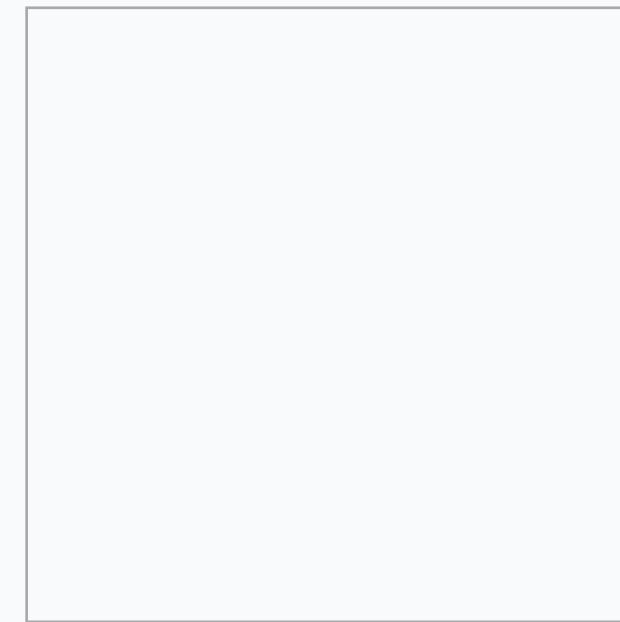
Make server
session
 $\text{sessions}[id] = \{\}$

sessions

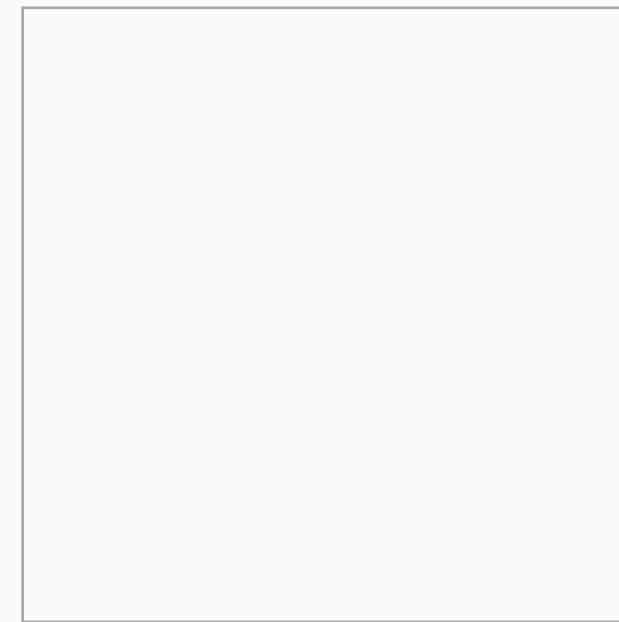
client



“the internet”



server



sessions

client

request



“the internet”

server

response

Look up session
 $\text{req.session} = \text{sessions}[\text{id}]$

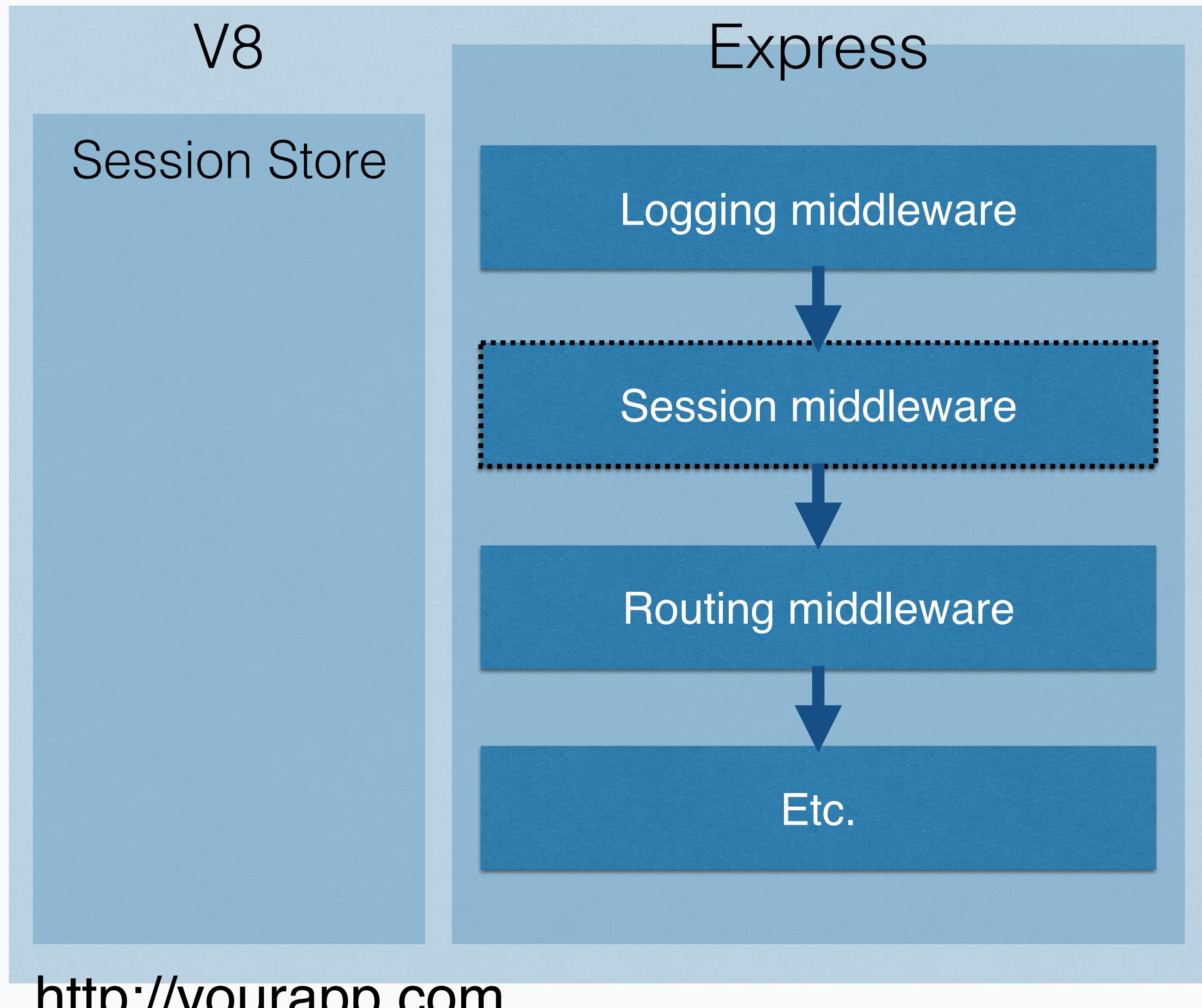
cookies & sessions

- server gives client a cookie with ID only
- Client keeps cookie and sends with all requests
- Server loads request-specific session (stored in RAM)

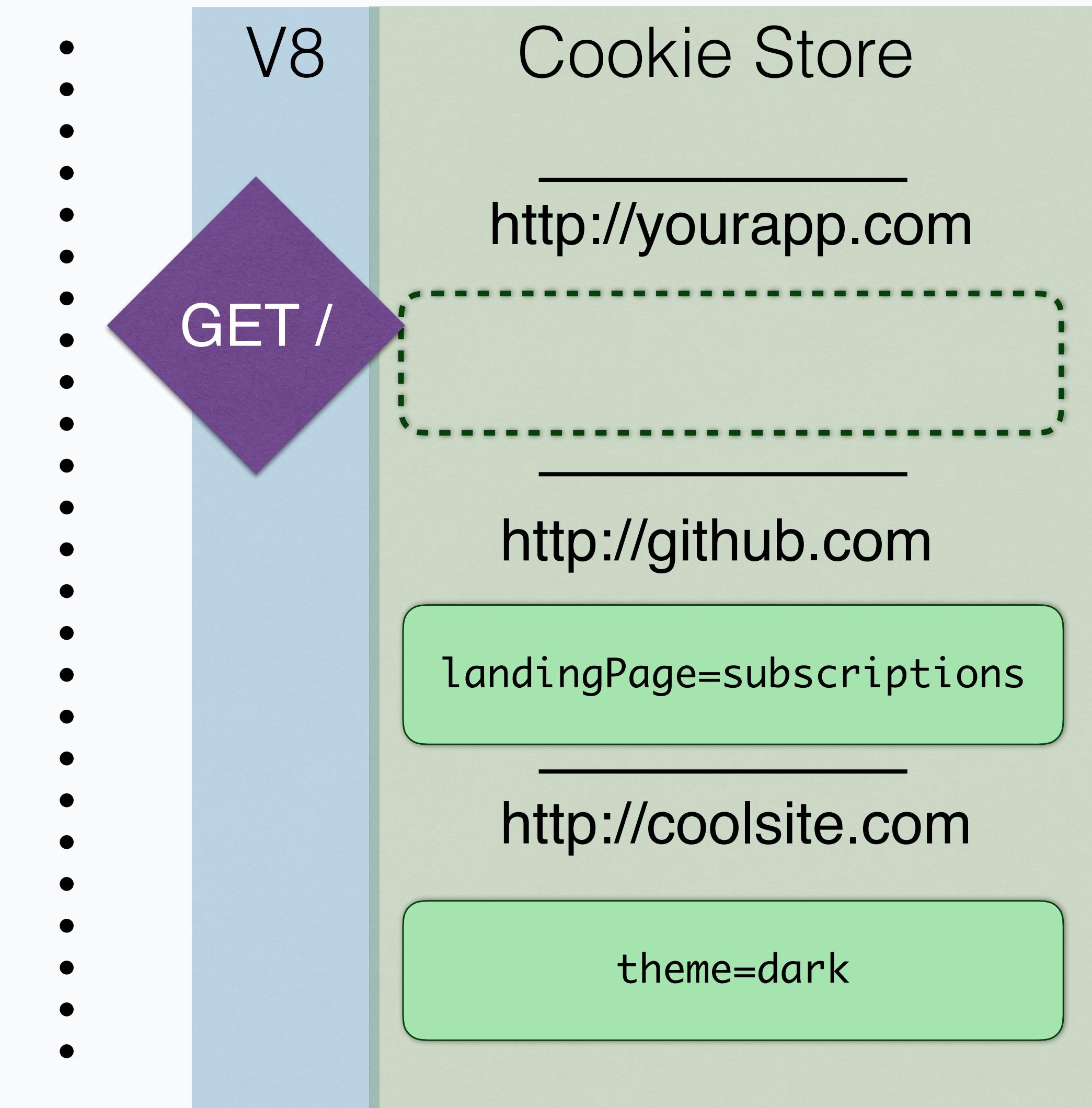


Sessions
IN
DETAIL

Server (Backend)



Internet (HTTP)



Client (Browser)

HTTP REQUEST

(headers)

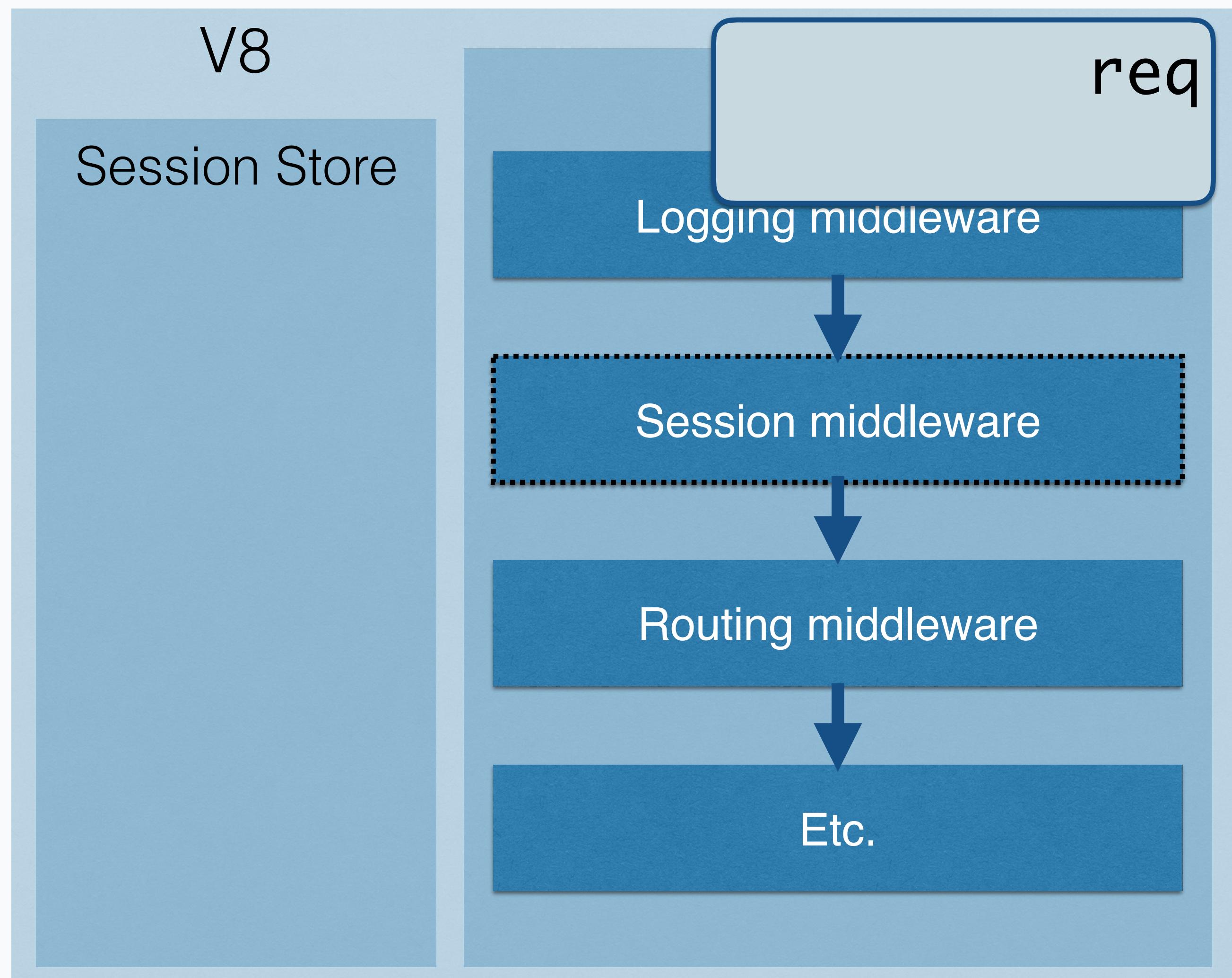
(body)

GET / HTTP/1.1
Host: yourapp.com
Connection: keep-alive
Accept: text/html
User-Agent: Chrome/
48.0.2564.116
...etc...

...no cookie info (yet)!

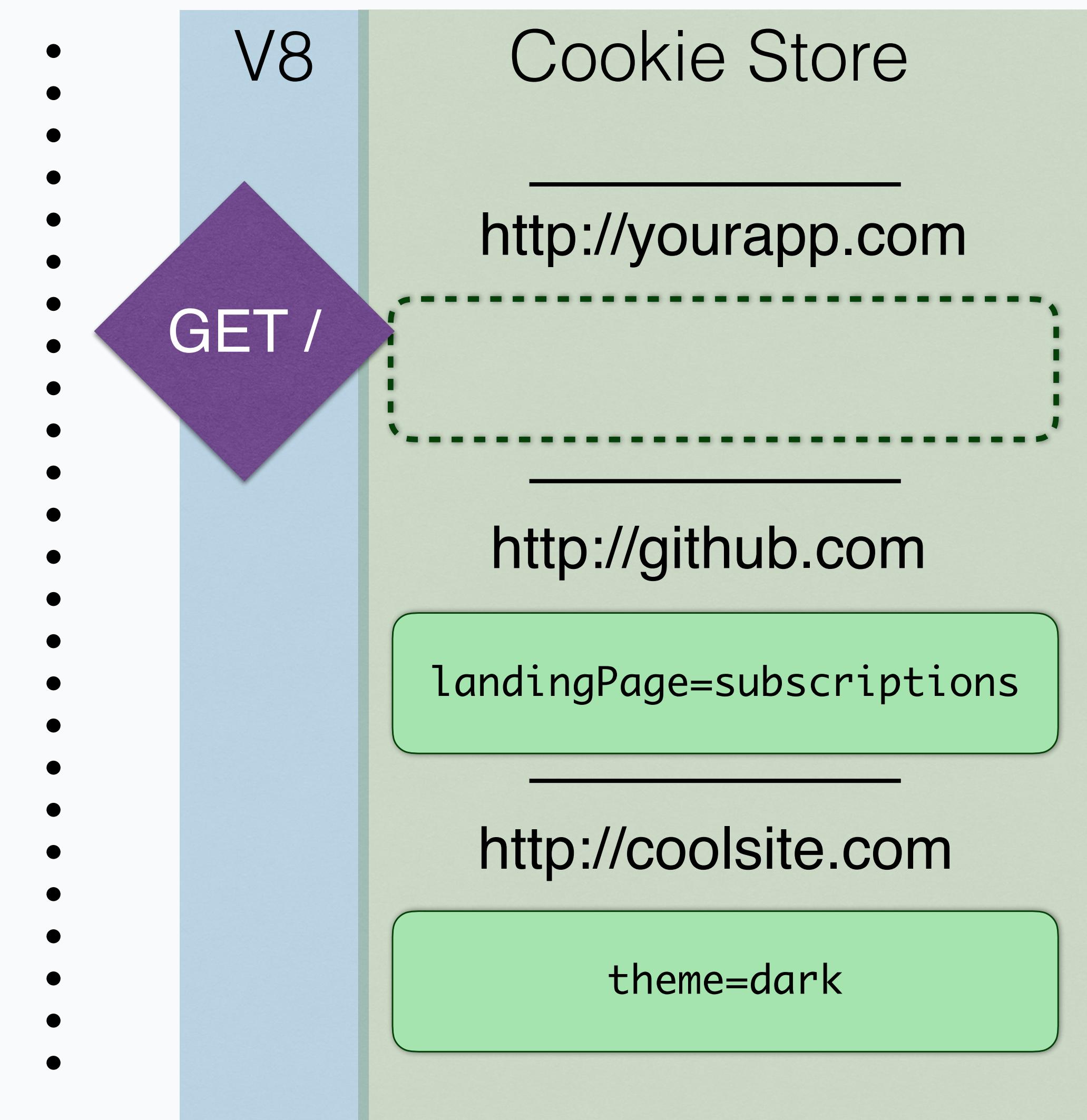


Server (Backend)



<http://yourapp.com>

Internet (HTTP)



<http://yourapp.com>

HTTP REQUEST

(headers)

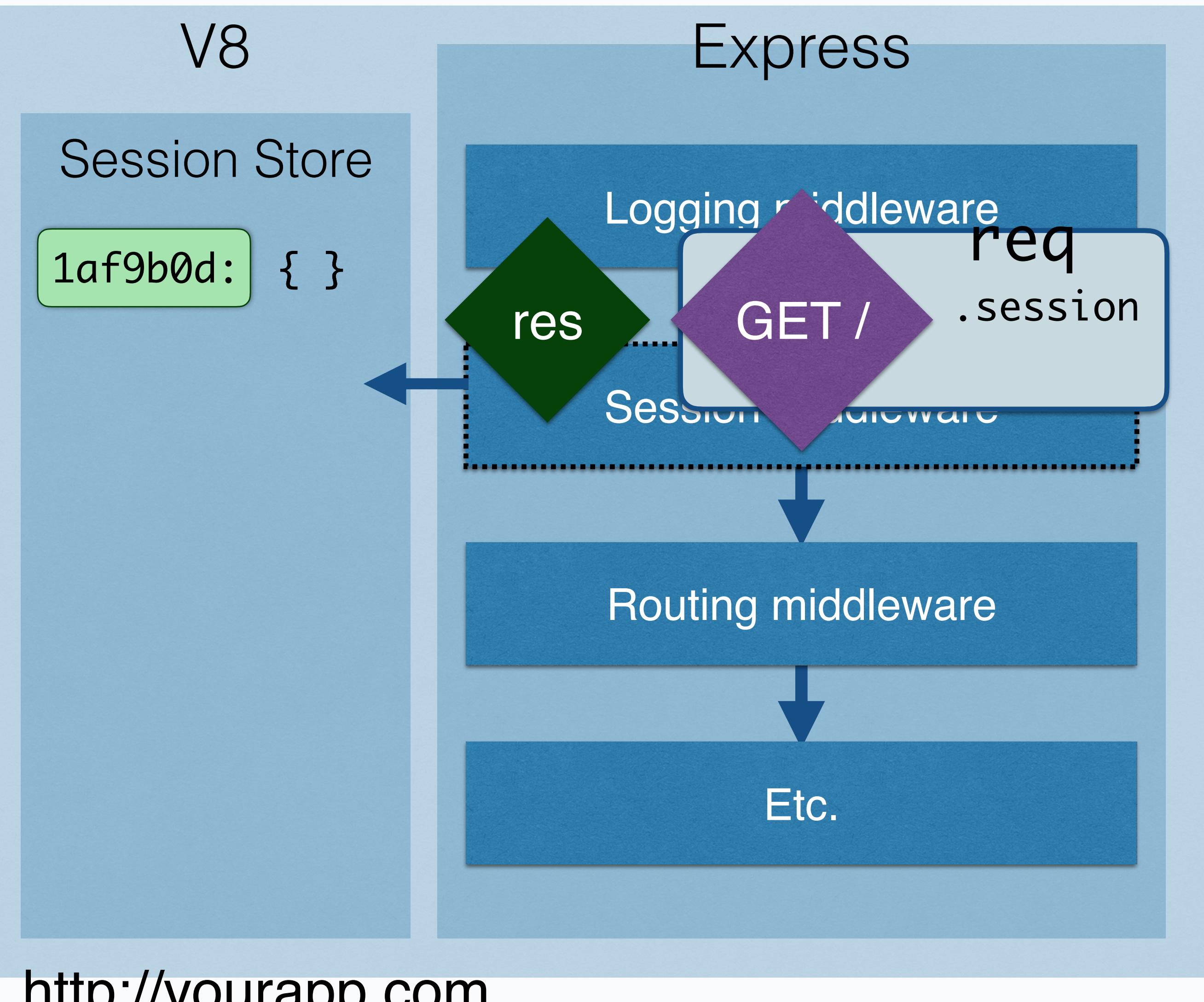
(body)

GET / HTTP/1.1
Host: yourapp.com
Connection: keep-alive
Accept: text/html
User-Agent: Chrome/
48.0.2564.116
...etc...

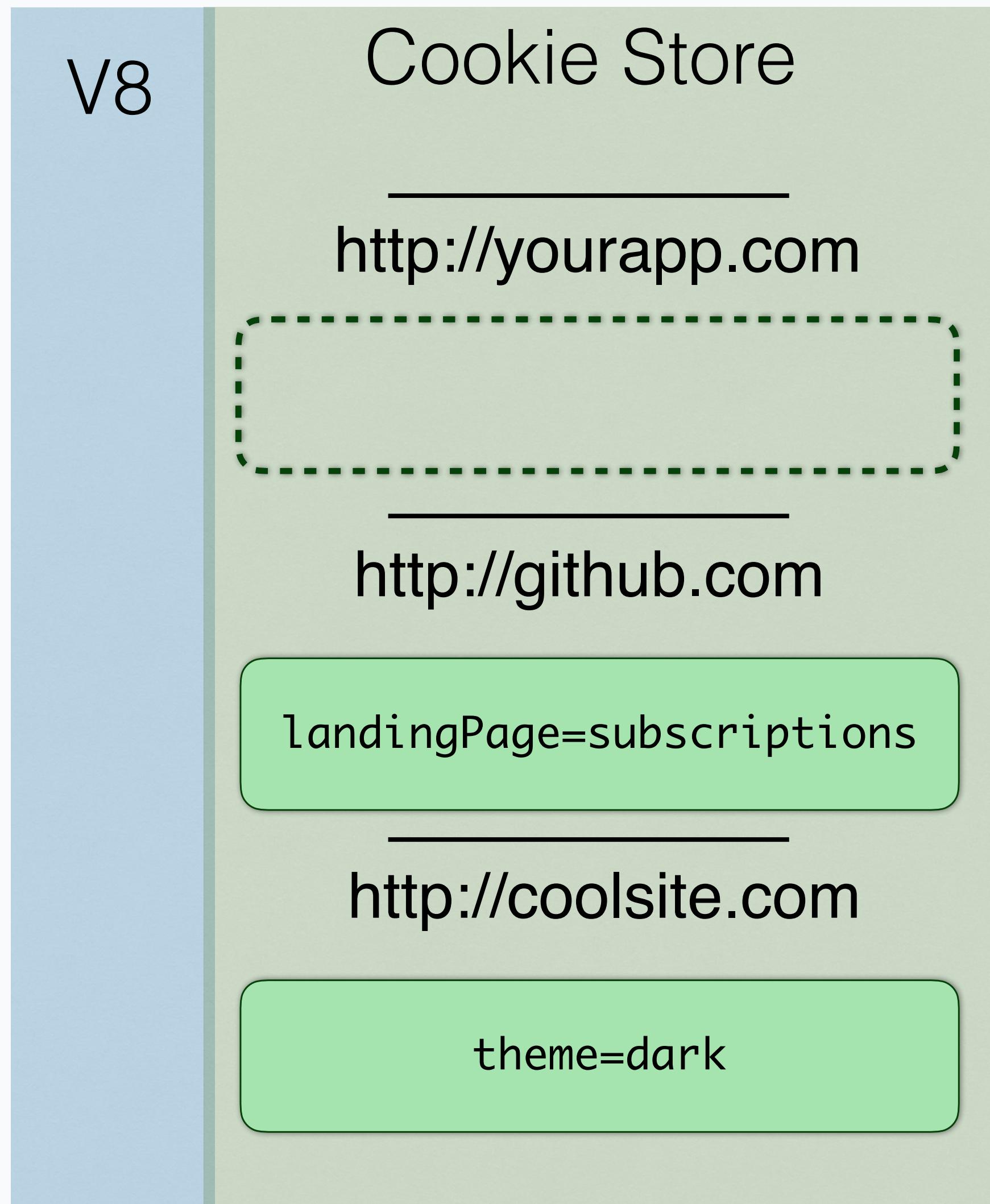
still no cookie info...
session middleware:
"let's make a session!"



Server (Backend)



Internet (HTTP)



Client (Browser)

HTTP RESPONSE

(headers)

(body, after
routing is
done)

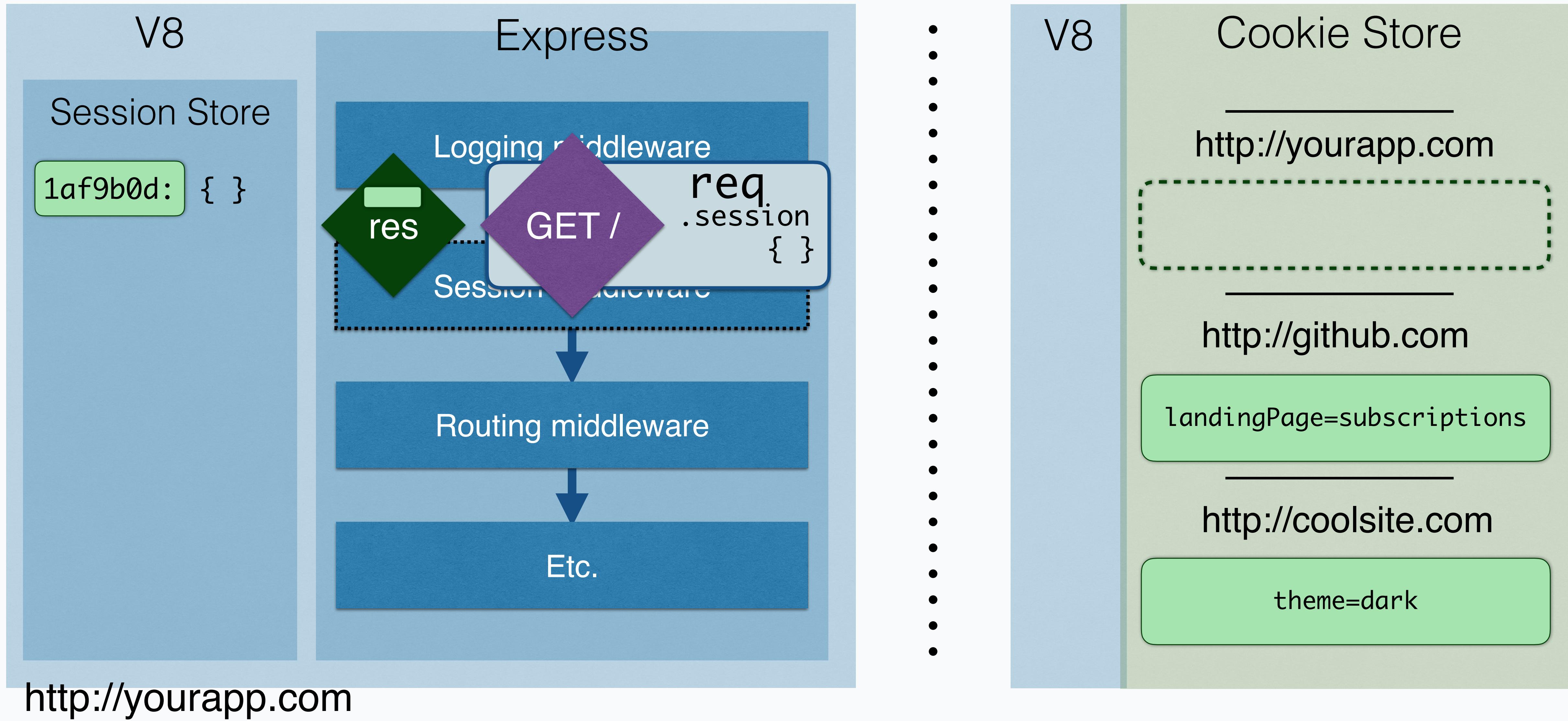
HTTP/1.x 200 OK
Transfer-Encoding: chunked
Date: Mon, 22 Feb 2016 18:30:00 GMT
Content-Type: text/html
Content-Encoding: gzip

set-cookie: myUid=1af9b0d

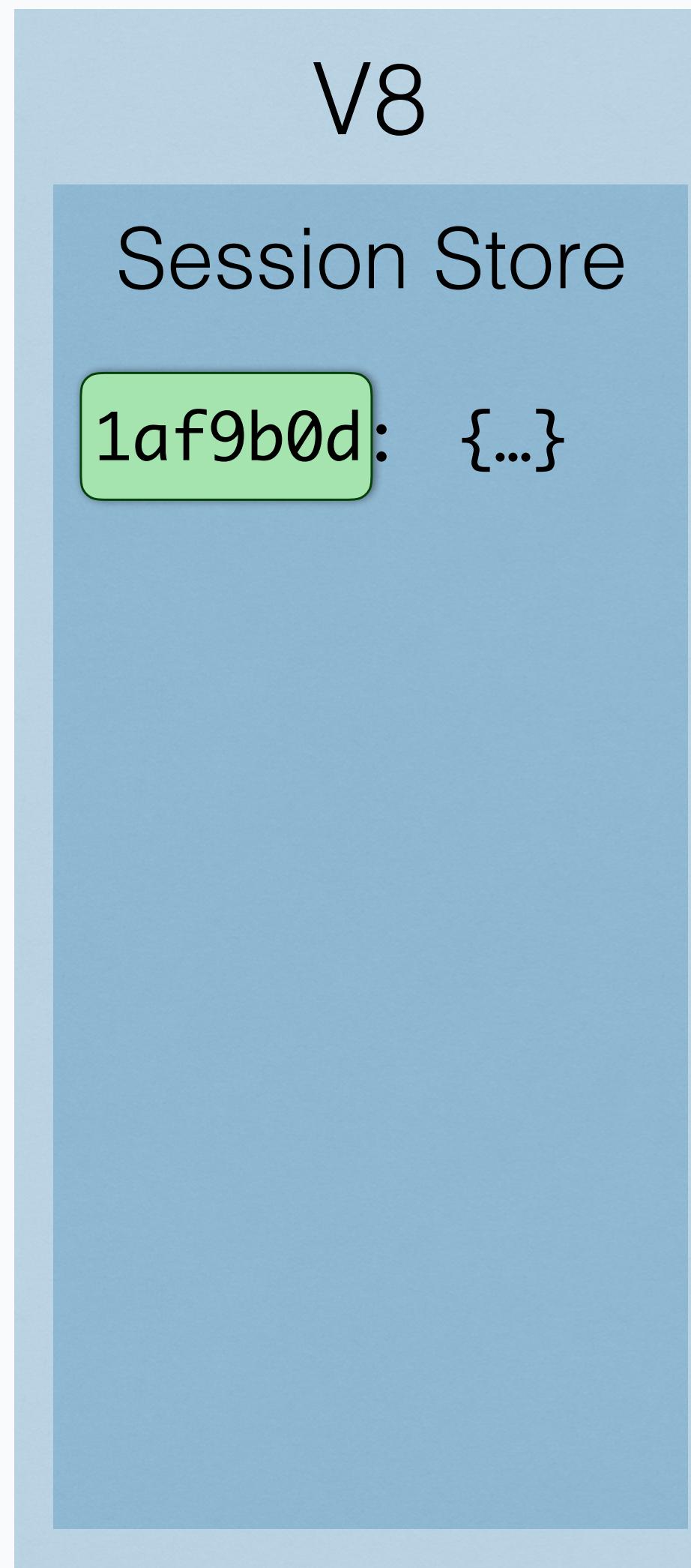
```
<!DOCTYPE html>
<html>
  <head>
    <title>Your Sweet App</title>
    <script src="/js/main.js"></script>
  </head> ...etc.
```



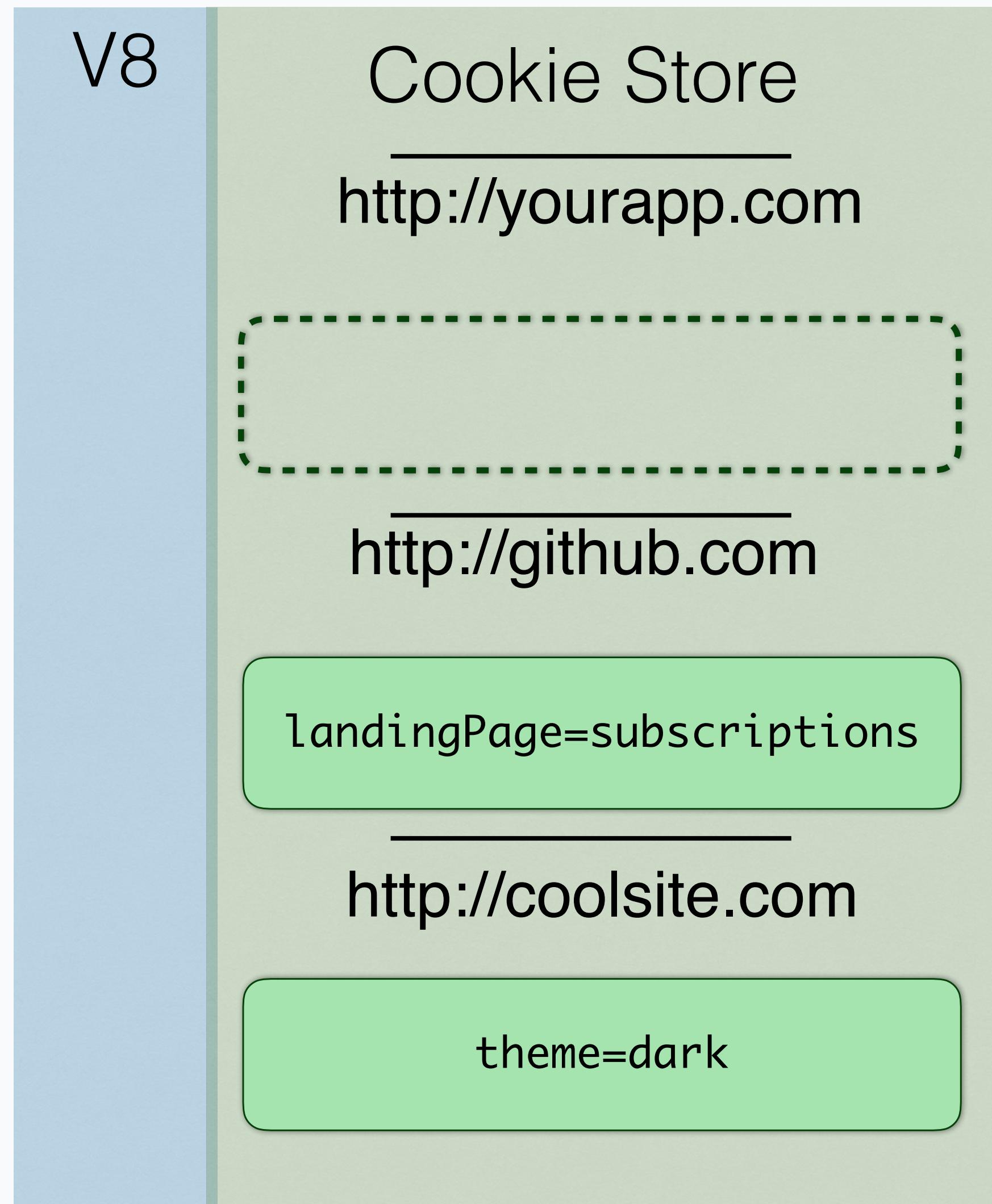
Server (Backend)



Server (Backend)



Internet (HTTP)



Client (Browser)

HTTP RESPONSE

(headers)

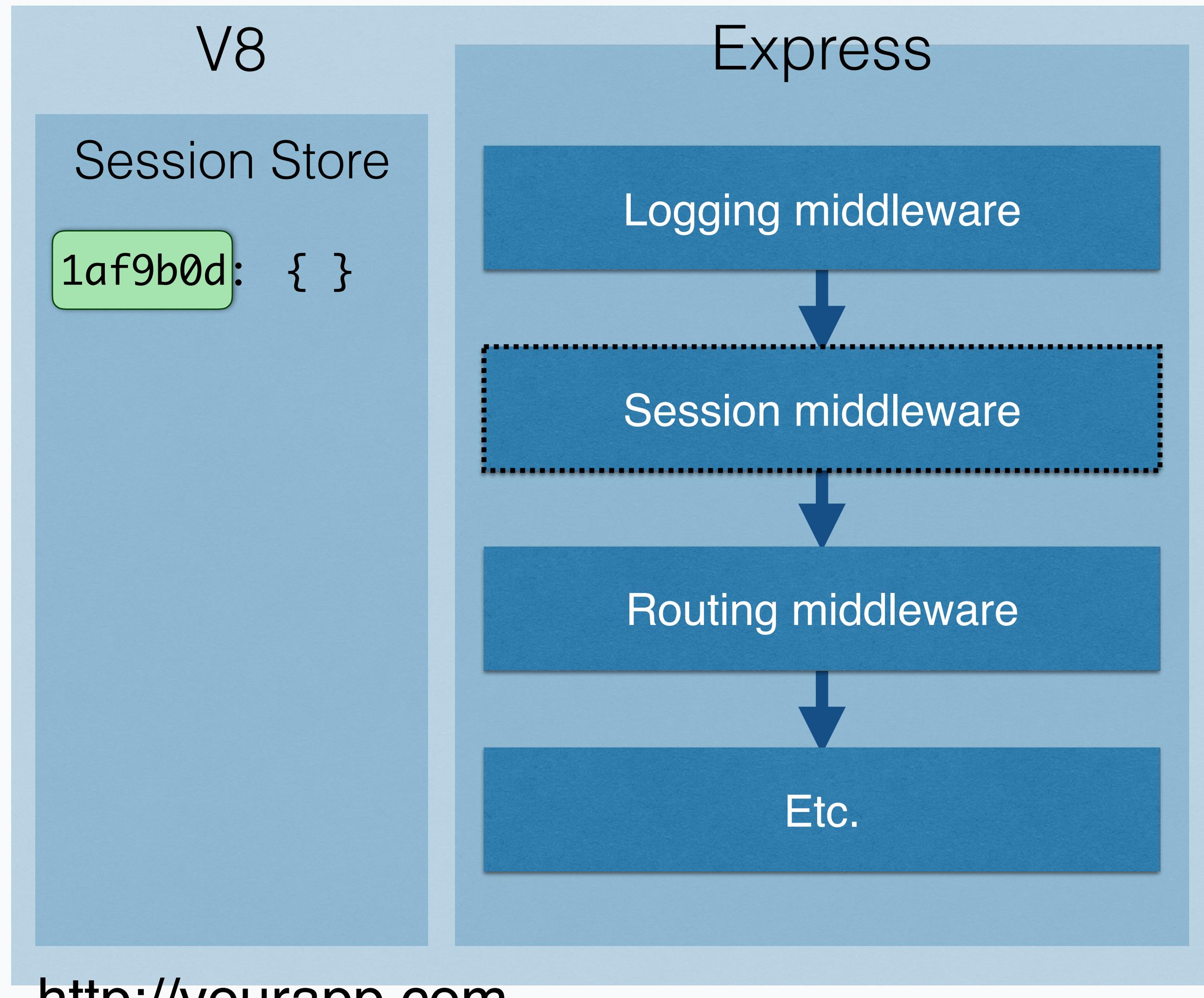
(body, after
routing is
done)

HTTP/1.x 200 OK
Transfer-Encoding: chunked
Date: Mon, 22 Feb 2016 18:30:00 GMT
Content-Type: text/html
Content-Encoding: gzip
set-cookie: myUid=1af9b0d

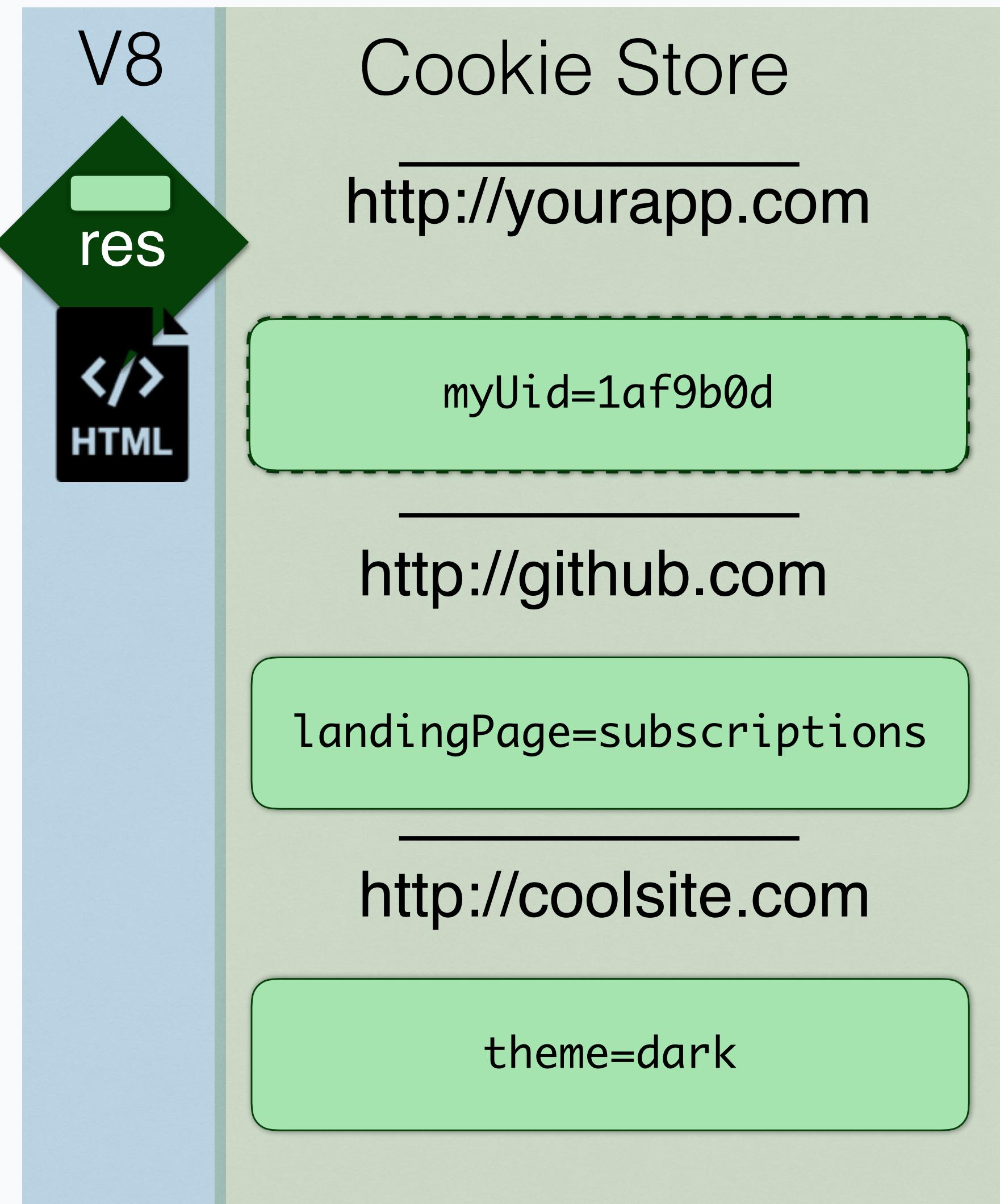
```
<!DOCTYPE html>
<html>
  <head>
    <title>Your Sweet App</title>
    <script src="/js/main.js"></script>
  </head> ...etc.
```



Server (Backend)

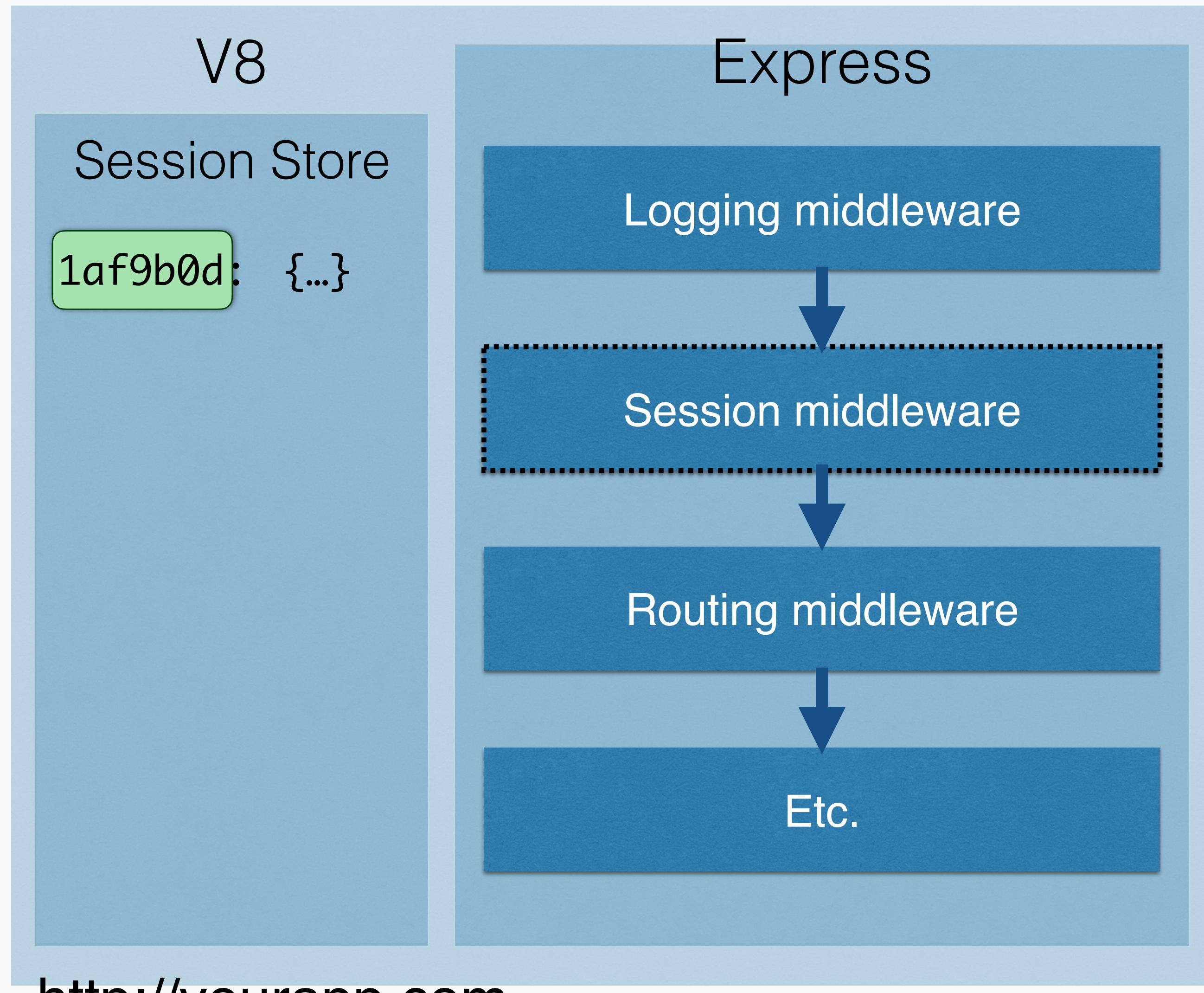


Internet (HTTP)



Client (Browser)

Server (Backend)



Internet (HTTP)



Client (Browser)

HTTP REQUEST

(headers)

(body)

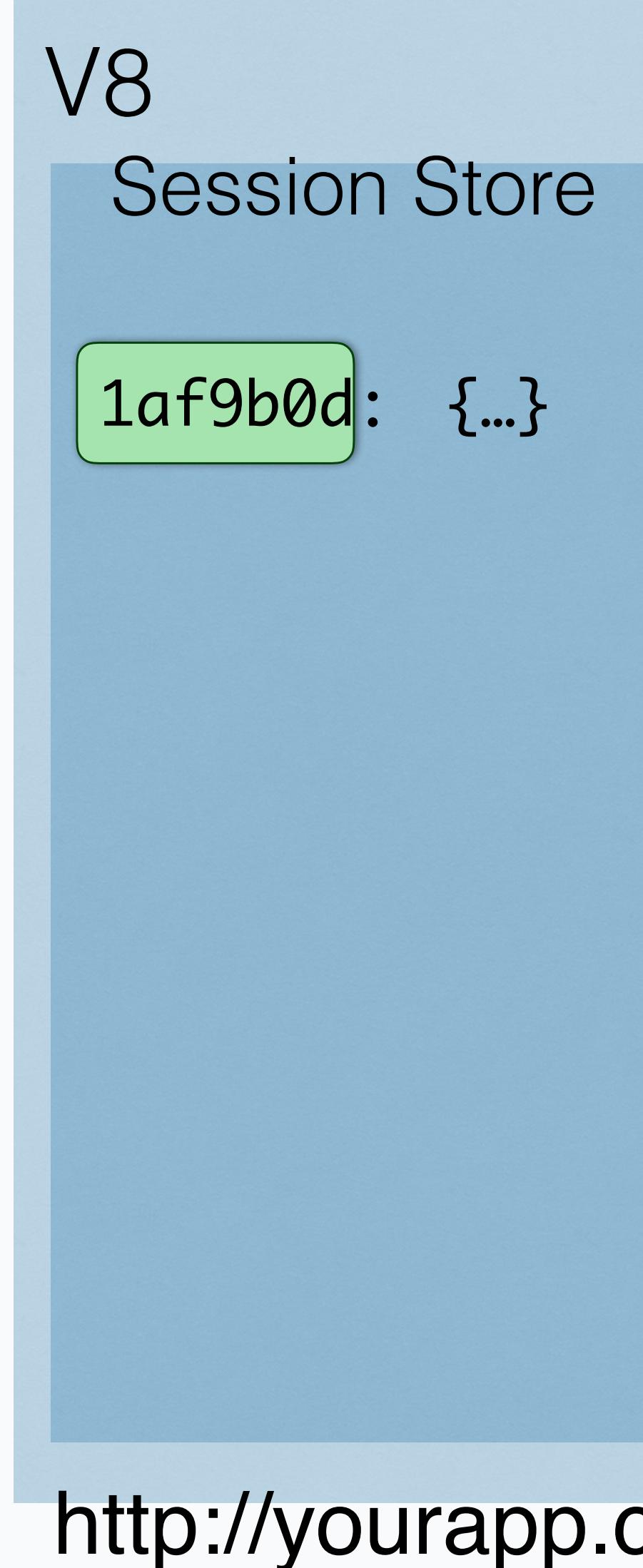
GET / HTTP/1.1
Host: yourapp.com
Connection: keep-alive
Accept: text/html
User-Agent: Chrome/
48.0.2564.116

Cookie: myUid=1af9b0d

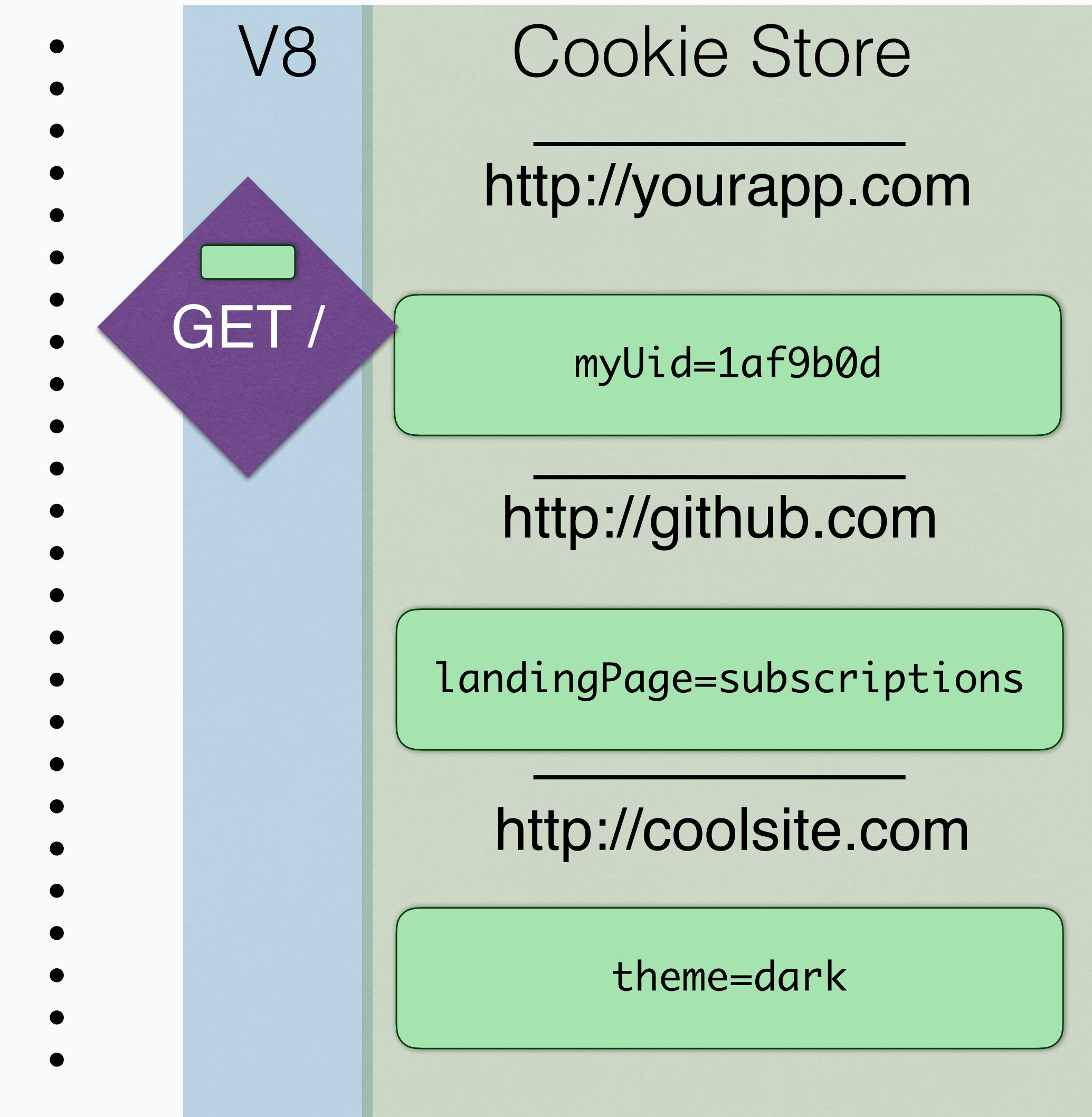
...attached cookie!



Server (Backend)



Internet (HTTP)



Client (Browser)

HTTP REQUEST

(headers)

(body)

GET / HTTP/1.1
Host: yourapp.com
Connection: keep-alive
Accept: text/html
User-Agent: Chrome/
48.0.2564.116

Cookie: myUid=1af9b0d

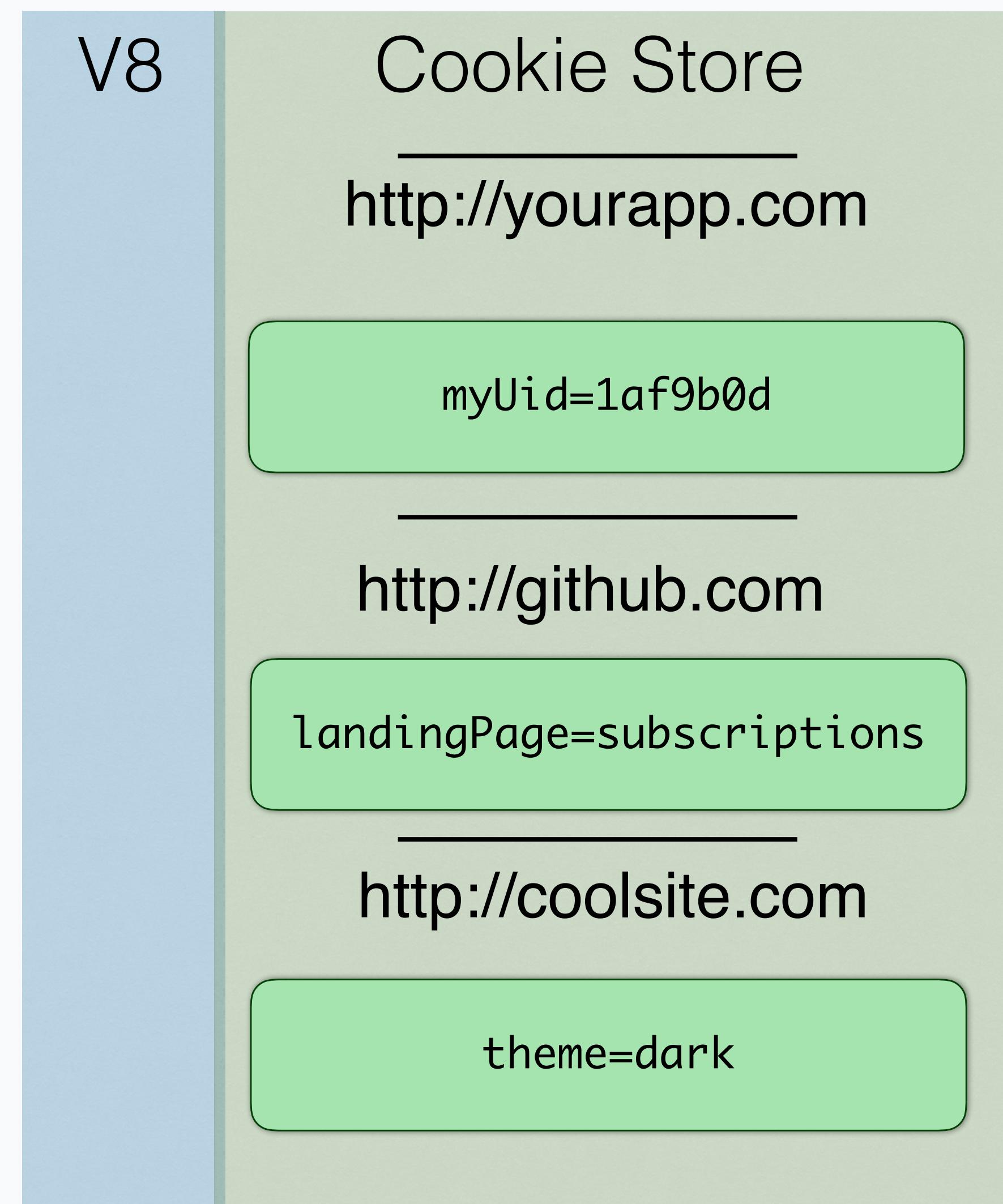
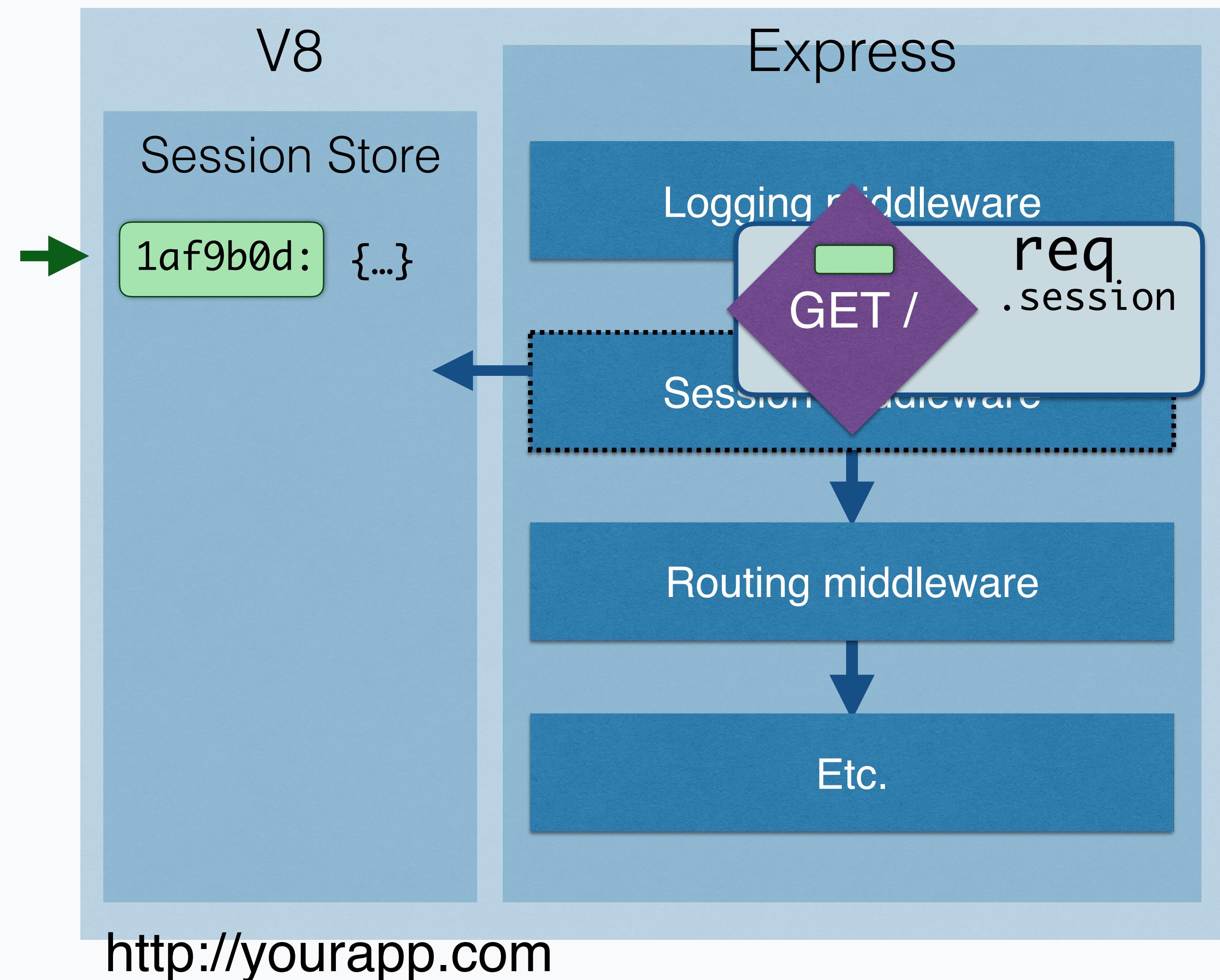
attached cookie, so
session middleware:
let's look up the session!



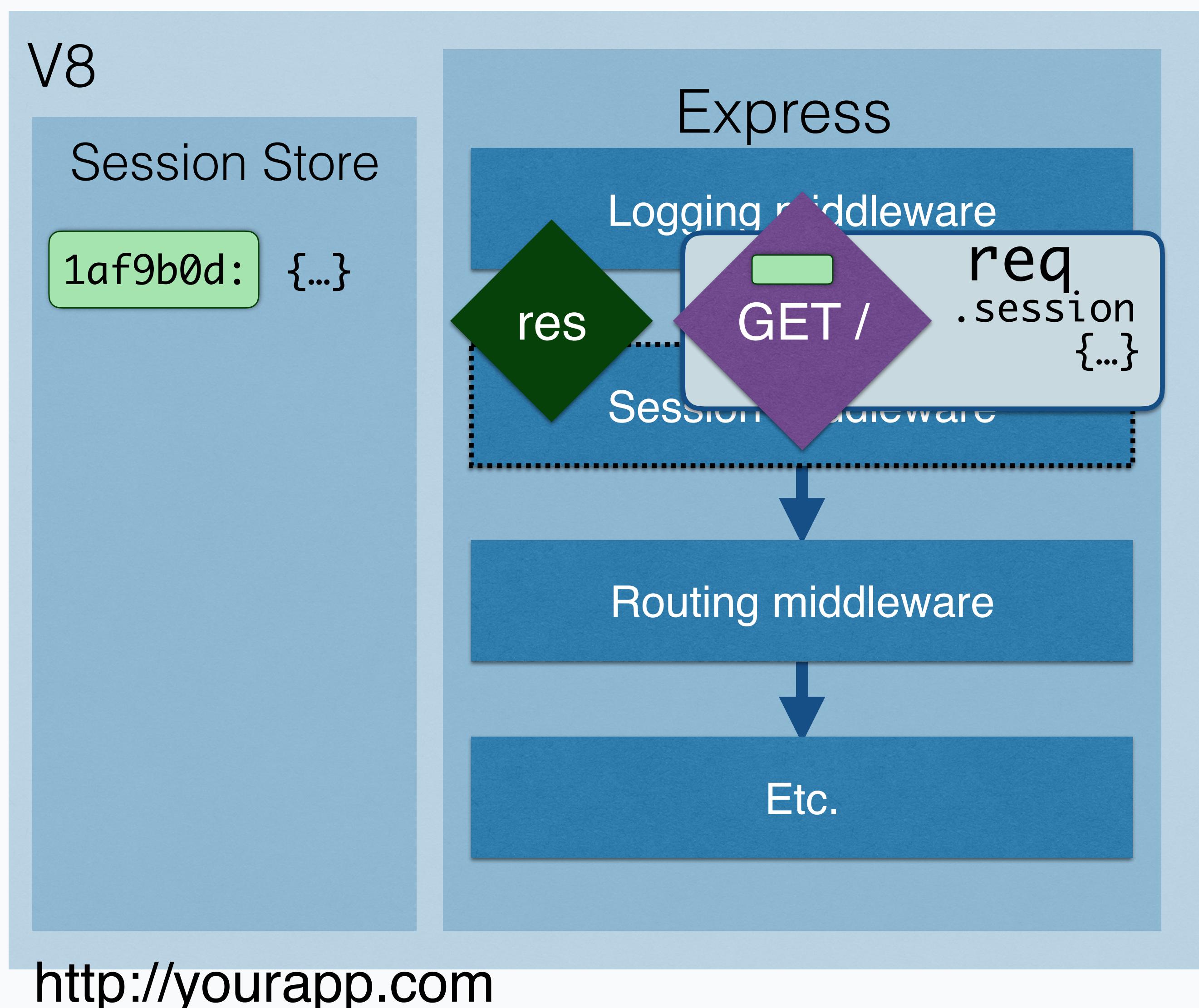
Server (Backend)

Internet (HTTP)

Client (Browser)



Server (Backend)



Internet (HTTP)

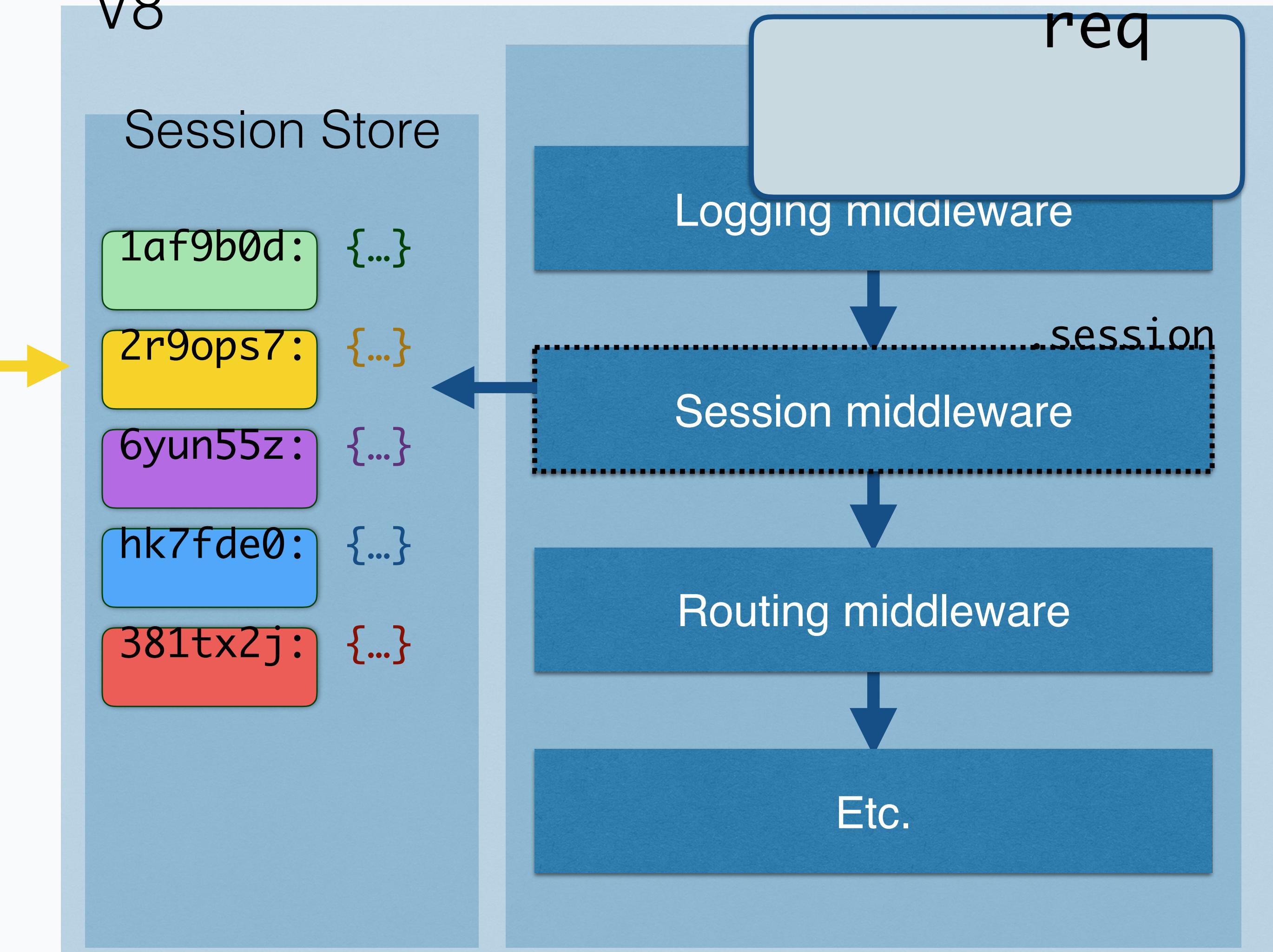


Client (Browser)

Multiple Sessions

Server (Backend)

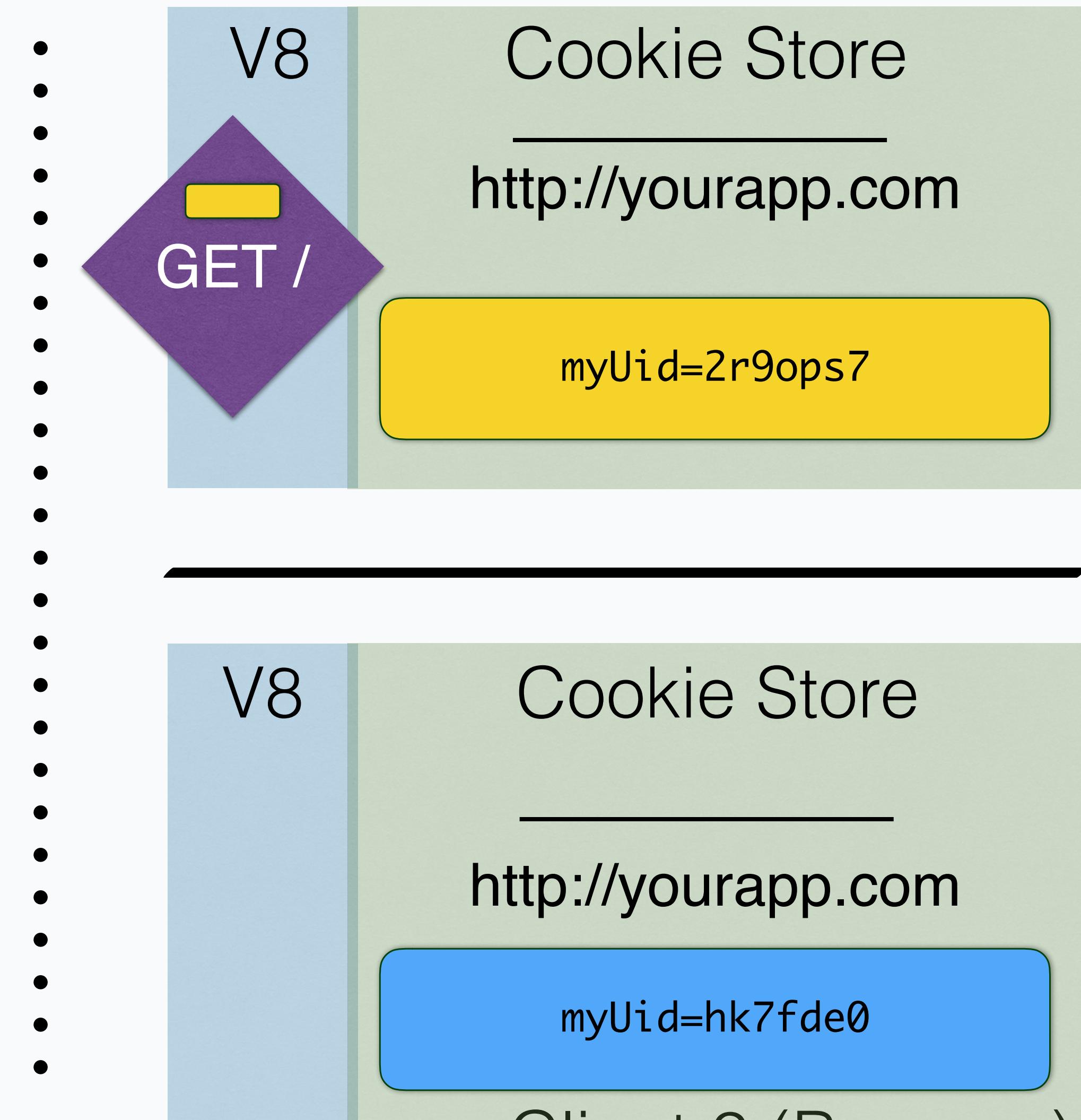
V8



http://yourapp.com

Internet (HTTP)

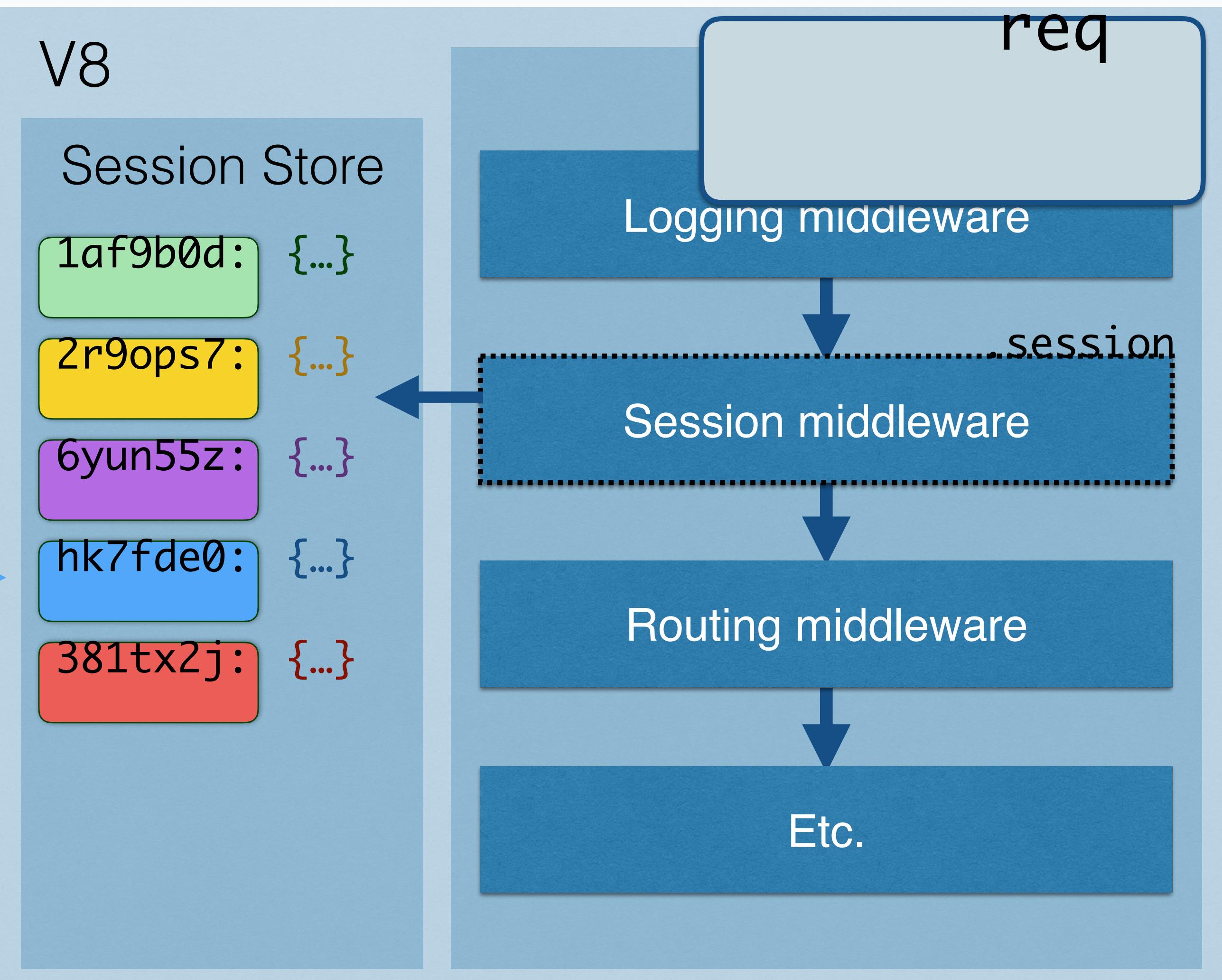
Client 1 (Browser)



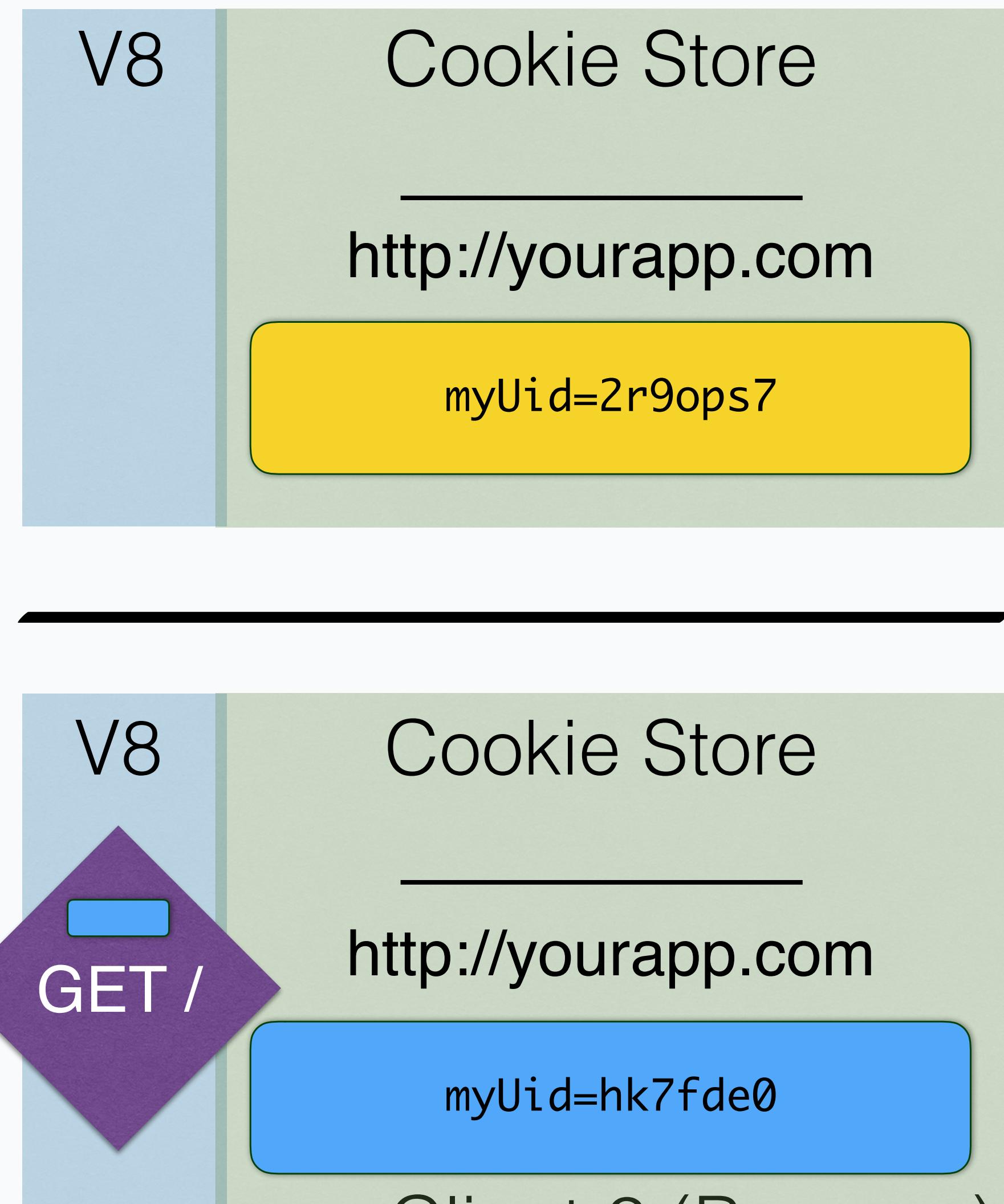
Server (Backend)

Internet (HTTP)

Client 1 (Browser)



`http://yourapp.com`



`Client 2 (Browser)`

“You get a session, you get a session,
everybody gets a session!”

-Not Oprah



Oauth

Standard protocol for auth piggybacking

Your Application

“consumer”

User

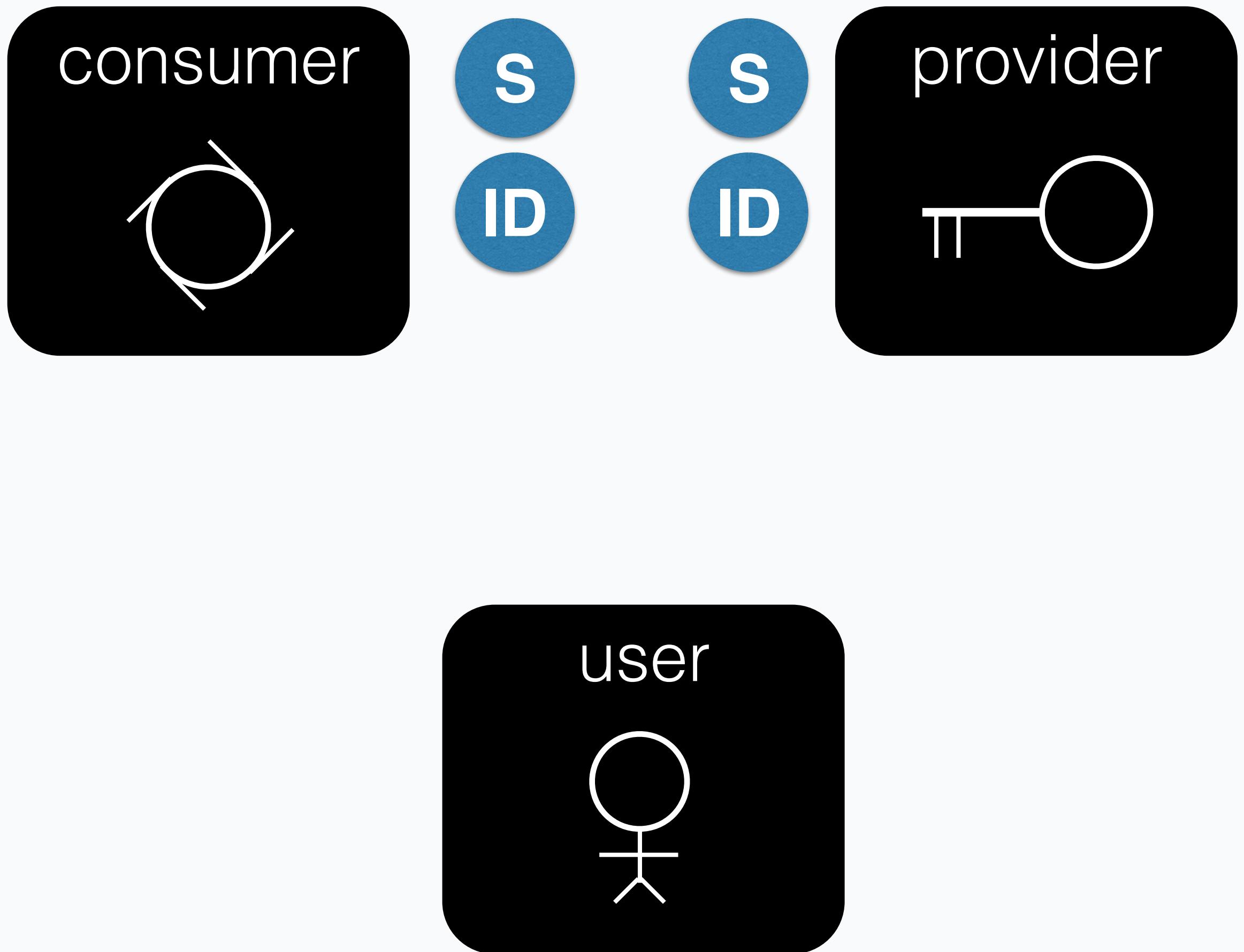
“user”

3rd Party Authority

“provider”

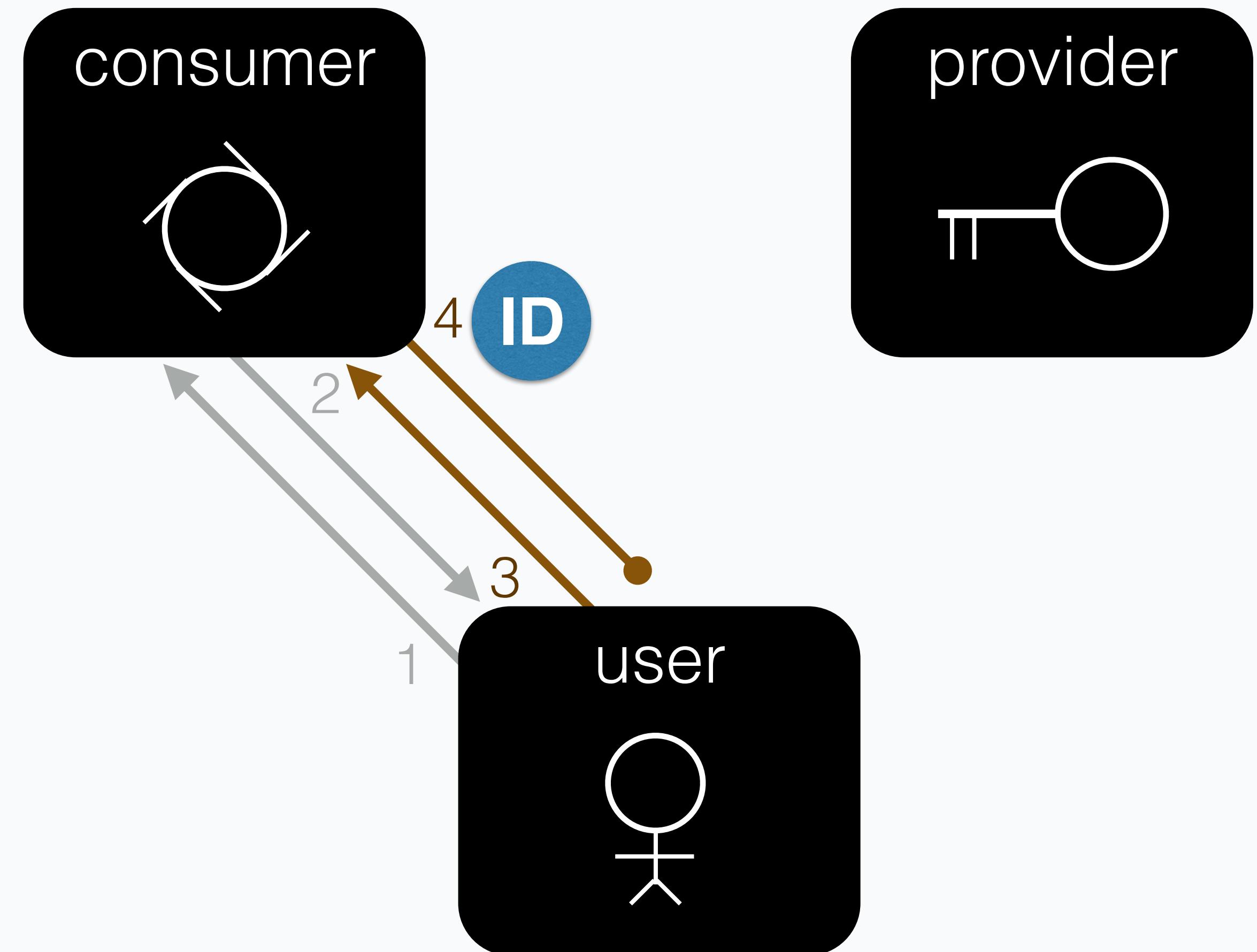
Oauth - PREP

- Consumer app has a registered dev account with the provider.
- Provider gives consumer a public client ID and private client Secret.
- Both services hold on to these credentials so the consumer can prove who it is to the provider.



Oauth - Petition

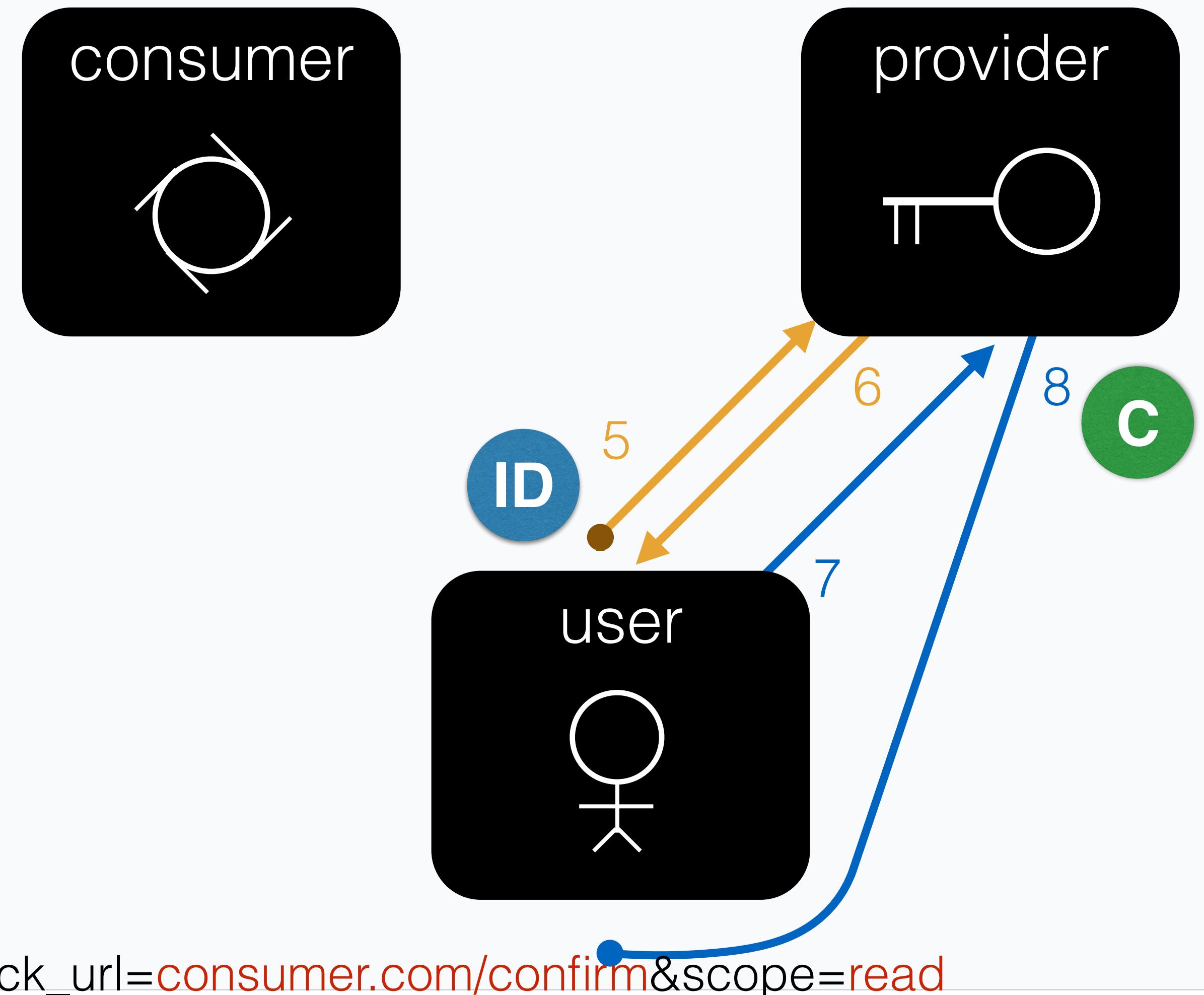
1. Request: load login
2. Response: rendered login
- 3. Request: login through provider**
- 4. Response: redirect to provider**



http://provider.com/oauth/authorize?client_id=123&callback_url=consumer.com/confirm&scope=read

Oauth - User authentication

5. Request: load login
6. Response: rendered login
7. Request: login to provider
8. Response: redirect callback URL



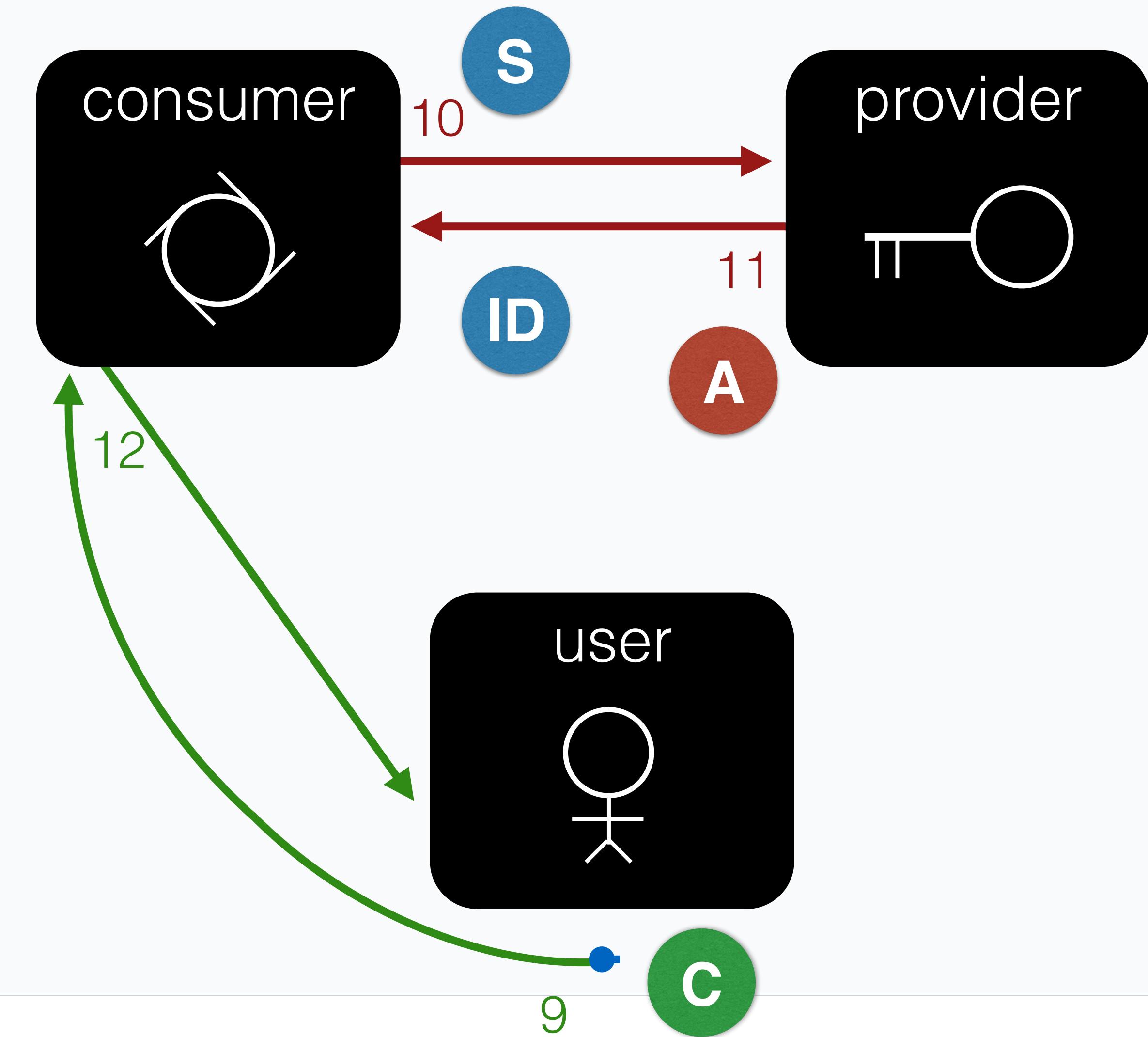
<http://consumer.com/confirm?authcode=789>

http://provider.com/oauth/authorize?client_id=123&callback_url=consumer.com/confirm&scope=read

http://provider.com/oauth/token?client_id=123&client_secret=911&authcode=789

Oauth - APP authentication

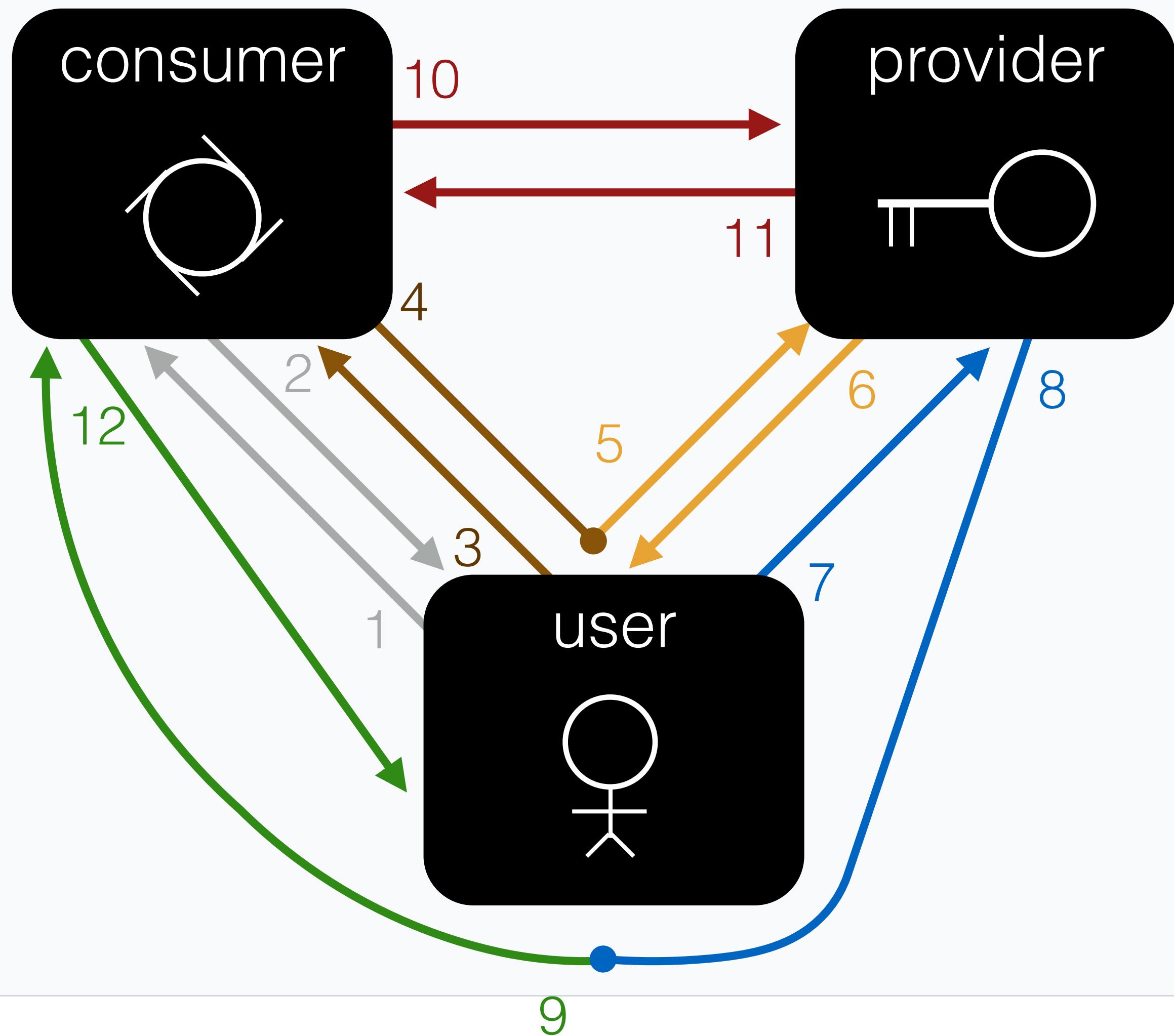
- 9. Request: callback URL
- 10. Request: authorization
- 11. Response: access token
- 12. Response: yay!



<http://consumer.com/confirm?authcode=789>

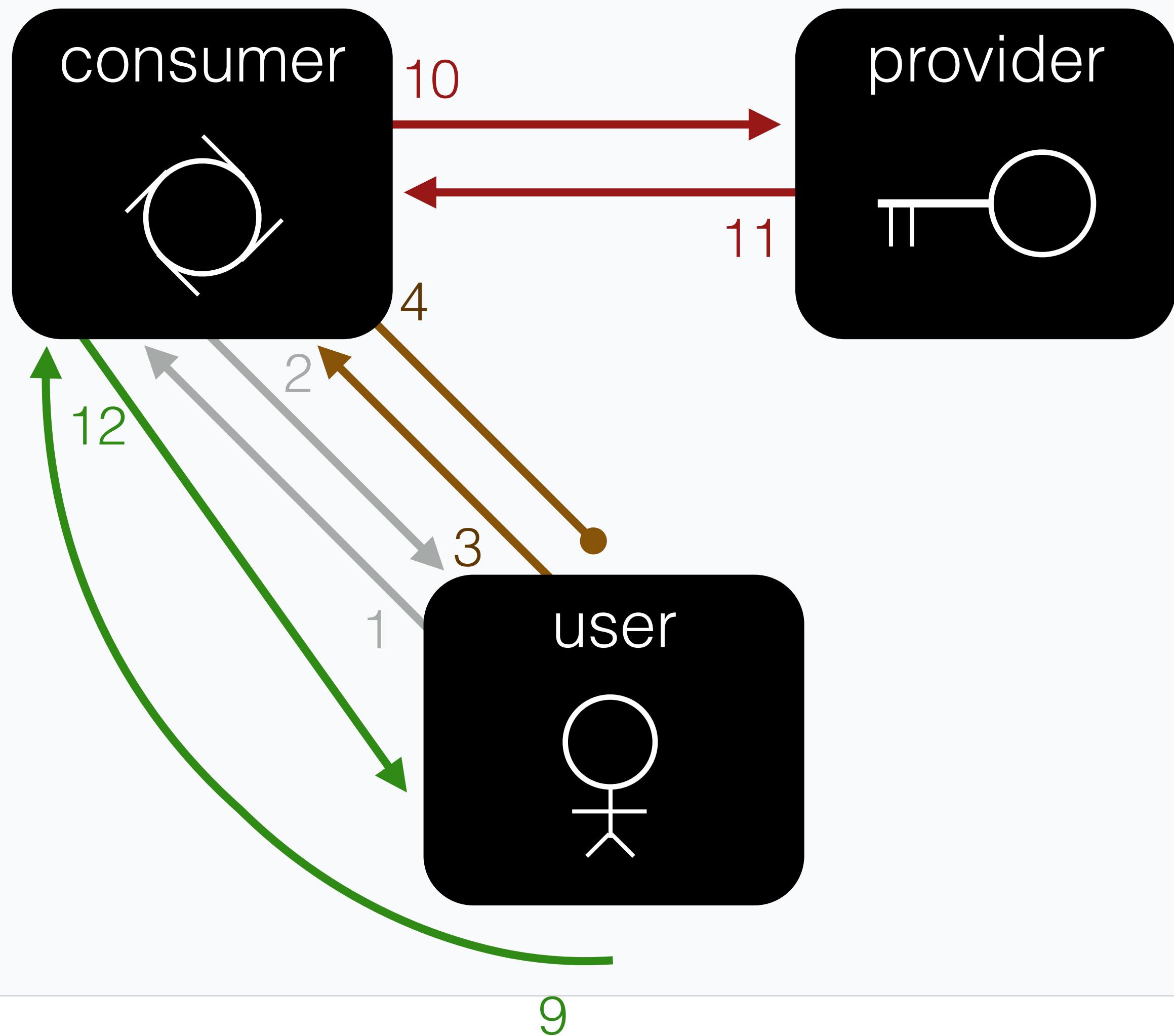
Oauth - all of it

1. Request (user to app): load login page
2. Response (app to user): rendered login page
3. Request (user to app): allow app to use provider as me [user petitions app for a special contract to allow the app to do certain things on the user's behalf]
4. Response (app to user): redirect to provider login, passing along (to provider) an app id, a success "callback URL", and a permissions "contract" [app transfers this petition to provider]
5. Request (implicit, user to provider): load login page
6. Response (provider to user): rendered login page
7. Request (user to provider): login to provider [the user signs the contract]
8. Response (provider to user): on success, redirect to callback URL, passing along a new temporary code [the provider approves the user's signature]
9. Request (implicit, user to app): initiate callback
10. Request (app to provider): request for authorization given temporary code and app secret key [the app signs the contract]
11. Response (provider to app): on success, passes back an access token [the provider approves the app's signature and puts the contract into effect]
12. Response (app to user): we're good to go!



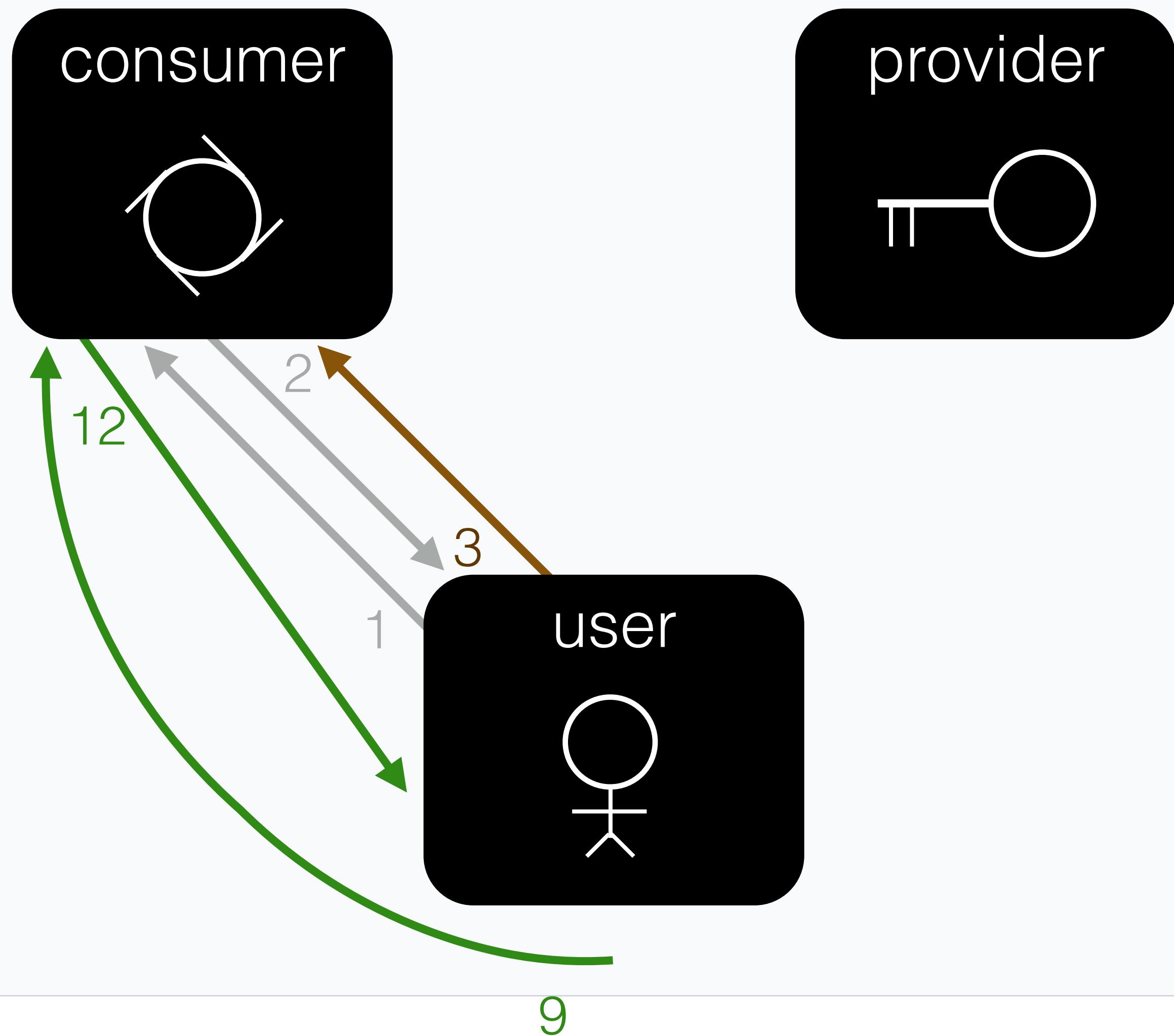
Oauth - Consumer role

1. Request (user to app): load login page
2. Response (app to user): rendered login page
3. Request (user to app): allow app to use provider as me [user petitions app for a special contract to allow the app to do certain things on the user's behalf]
4. Response (app to user): redirect to provider login, passing along (to provider) an app id, a success "callback URL", and a permissions "contract" [app transfers this petition to provider]
5. Request (implicit, user to provider): load login page
6. Response (provider to user): rendered login page
7. Request (user to provider): login to provider [the user signs the contract]
8. Response (provider to user): on success, redirect to callback URL, passing along a new temporary code [the provider approves the user's signature]
9. Request (implicit, user to app): initiate callback
10. Request (app to provider): request for authorization given temporary code and app secret key [the app signs the contract]
11. Response (provider to app): on success, passes back an access token [the provider approves the app's signature and puts the contract into effect]
12. Response (app to user): we're good to go!



Oauth - Consumer role with passport

1. Request (user to app): load login page
2. Response (app to user): rendered login page
3. Request (user to app): allow app to use provider as me [user petitions app for a special contract to allow the app to do certain things on the user's behalf]
4. Response (app to user): redirect to provider login, passing along (to provider) an app id, a success "callback URL", and a permissions "contract" [app transfers this petition to provider]
5. Request (implicit, user to provider): load login page
6. Response (provider to user): rendered login page
7. Request (user to provider): login to provider [the user signs the contract]
8. Response (provider to user): on success, redirect to callback URL, passing along a new temporary code [the provider approves the user's signature]
9. Request (implicit, user to app): initiate callback
10. Request (app to provider): request for authorization given temporary code and app secret key [the app signs the contract]
11. Response (provider to app): on success, passes back an access token [the provider approves the app's signature and puts the contract into effect]
12. Response (app to user): we're good to go!





Passport

TL;DR

- Assuming you set everything up right:
- Client can log in by requesting GET /auth/fb or similar
- In routes you will now have req.user to check user info

Passport Ingredients

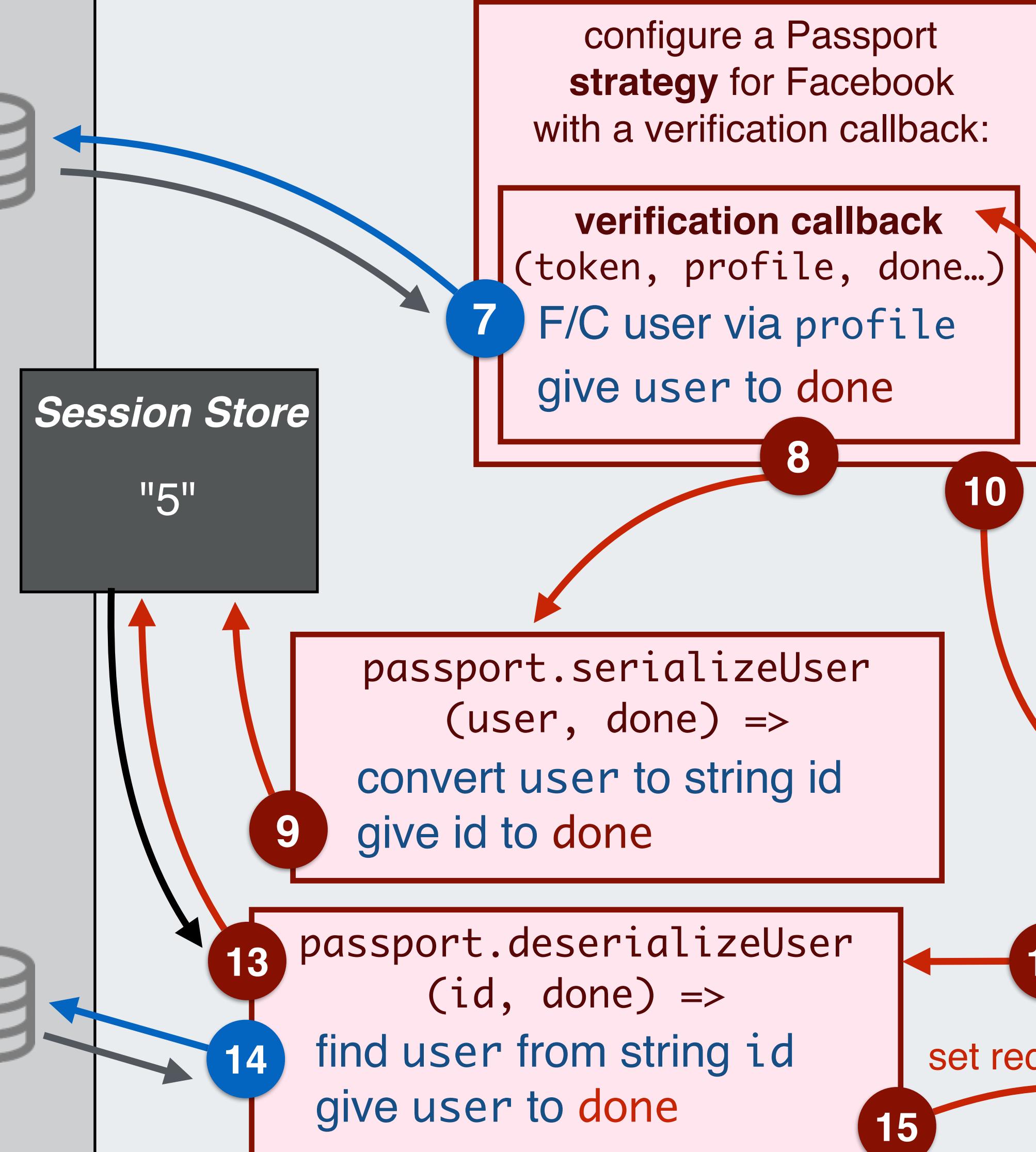
- attach `passport.session()` middleware
 - Only after `express-session`
- Define how to minimally store & look up user using session
 - `passport.serializeUser` / `passport.deserializeUser`
- Must configure a Strategy
 - Strategy needs a *verification callback* that you write – longest part
- `passport.authenticate` (in two different routes)
 - Uses strategy
 - Slightly different call in each route

DB

SERVER

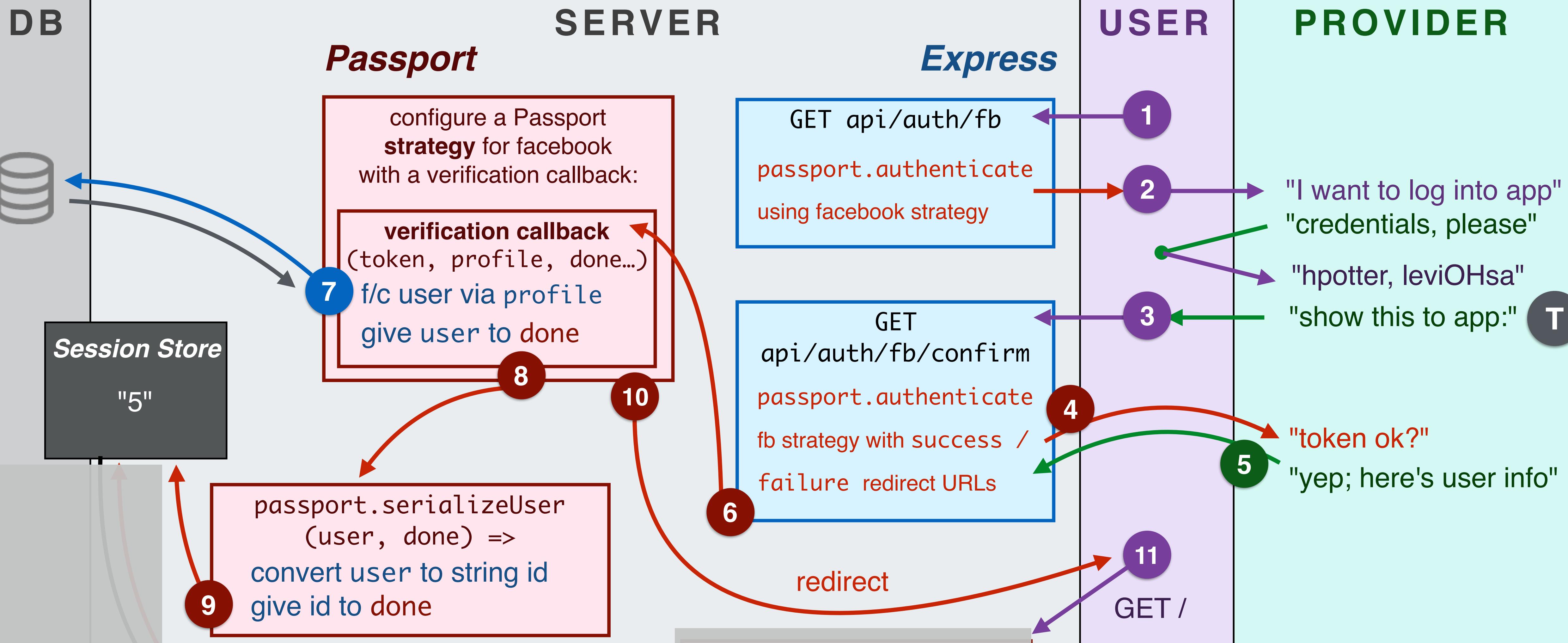
USER

PROVIDER

Passport

T





Initial login request: use `passport.authenticate` in routes. Token / profile will be passed into verification callback. Find / create user in DB based on profile info. Passport stores ID in session for easy future lookup, then tells client to redirect to the "success" URL.

configure a Passport strategy for facebook
with FacebookStrategy({})

GET api/auth/fb

1

All subsequent requests (while session persists):

Passport.session middleware calls deserializeUser

Session Store

"5"

User ID is fetched from session store

ID is used to retrieve user from DB

User is attached to req.user

Routes can now use req.user

13 passport.deserializeUser
(id, done) =>
find user from string id
give user to done

12 passport.session()

(any route)
use req.user as needed

14

15

16

11

GET /



auther™