

Fullstack Development

Authentication / Authorization

Part 2: Social signing up/in

Something like this

We need OAuth 2.0.



Sign in with Google



Sign in with Facebook



Sign in with Apple



Sign in with Twitter



Sign in with email


Part 2: Social signing up/in

Section 2A: OAuth 2.0

OAuth 2.0

Example of consent screen

3rdPartApp wants to access your
Google Account

 some@email.com

This will allow **3rdPartApp** to:

 31

View and edit events on all your calendars



Make sure you trust 3rdPartApp

You may be sharing sensitive info with this site or app.
Learn about how calendly.com will handle your data by
reviewing its [terms of service](#) and [privacy policies](#). You
can always see or remove access in your [Google Account](#).

[Learn about the risks](#)

Cancel

Allow

OAuth 2.0

- "Open Authorization"
- Standard designed to allow application to access resources hosted by other web apps on behalf of a user.
 - Standard for `author`
 - Not for `authn`
- Replaced OAuth 1.0 in 2012.

OAuth 2.0

- Specifies many "flows"
 - **Authorization Code Flow**
 - Client Credentials Flow
 - Refresh Token Flow
 - JWT Bearer Flow
 - Device Code Flow
- We will use "Authorization Code Flow" for social login.

Recommended resources

- <https://engineering.backmarket.com/oauth2-explained-with-cute-shapes-7eae51f20d38>
- https://developer.okta.com/blog/2019/10/21/illustrated-guide-to-oauth-and-oidc?utm_source=pocket_shared
- <https://youtu.be/8aCyojTIW6U?si=YPxkcLPcAoK5jixl>
- <https://youtu.be/t18YB3xDfXI?si=pD1JnFP0GrnBXW2v>

Wait

Are we using OAuth (standard for `author`) and **authorization** code flow for `authn`?

Yes, we kind of "misusing" it.

Authorization code flow

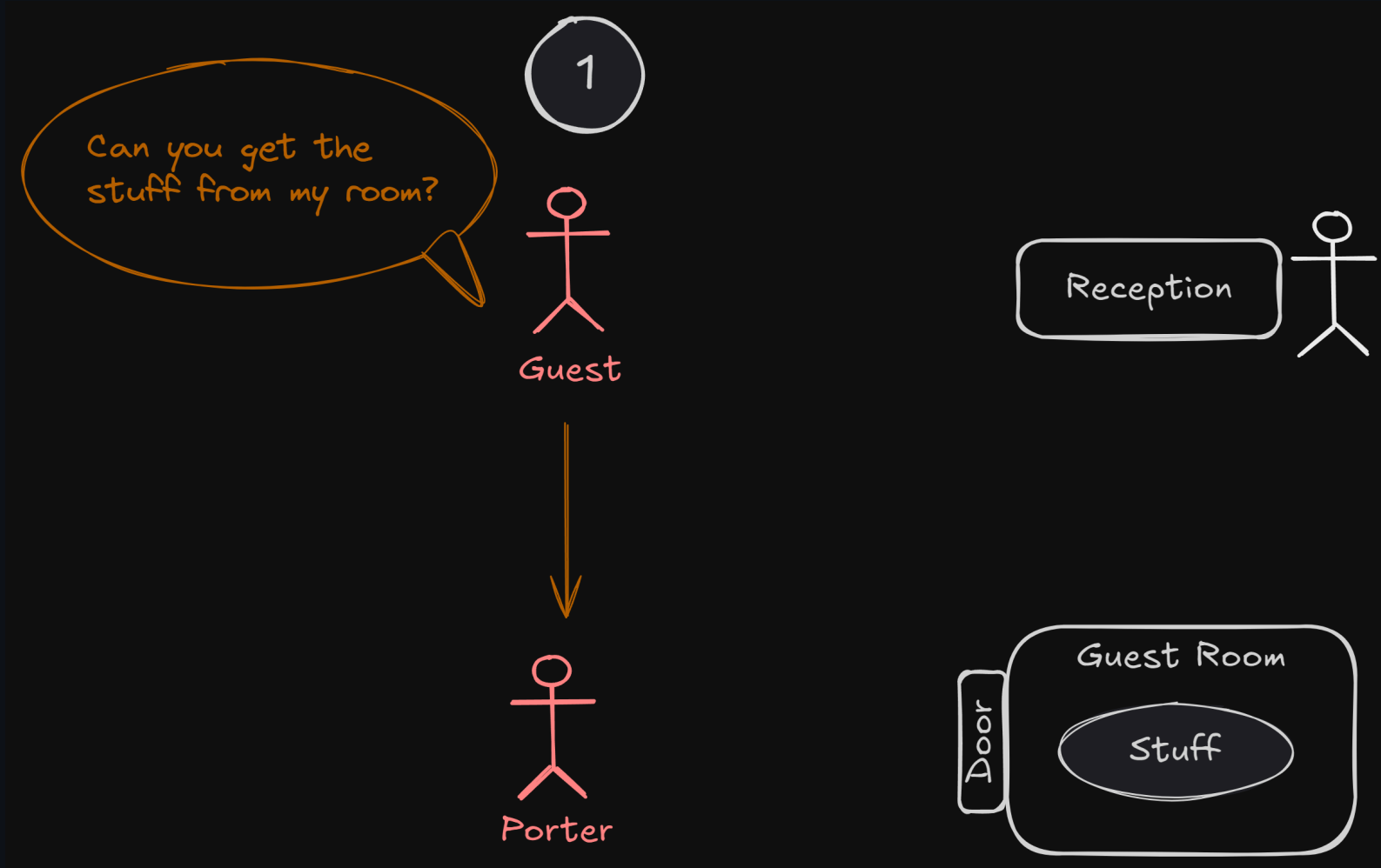
| In real life

Setup

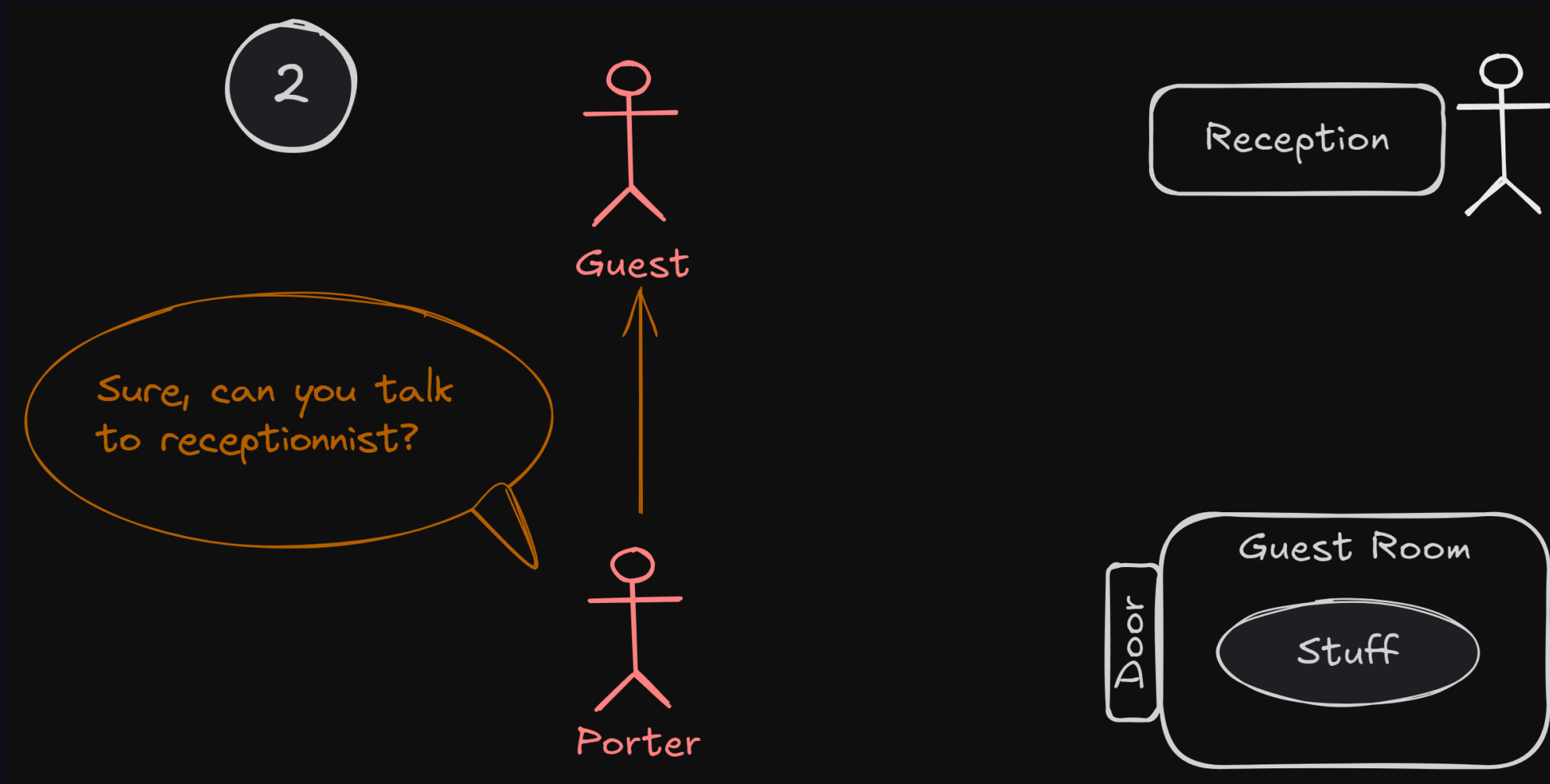
- You are a guest at a hotel.
- You already checked out.
- You forgot your stuff in the room.
- You want a porter to get your stuff for you.



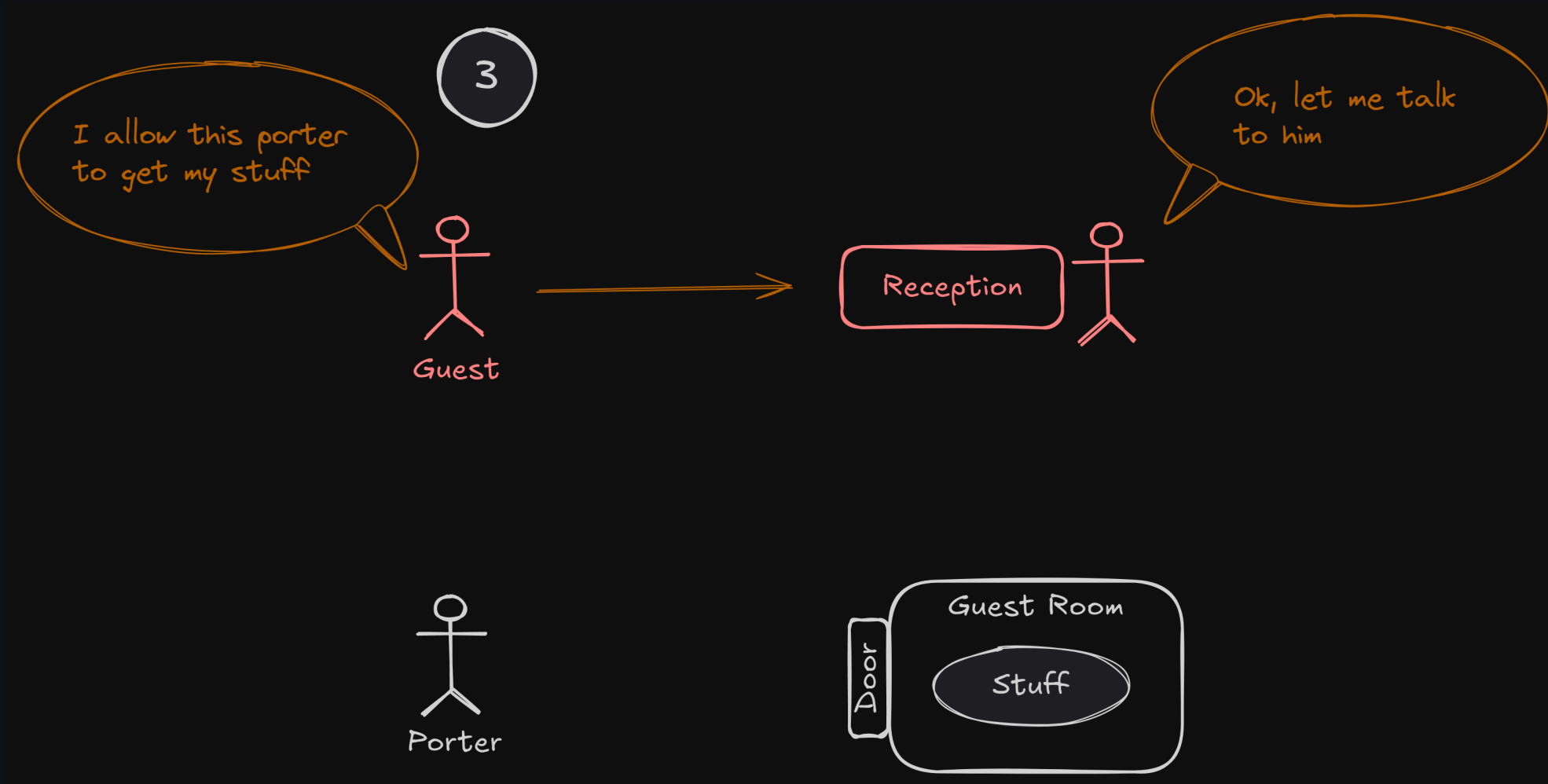
OAuth 2.0 in real life



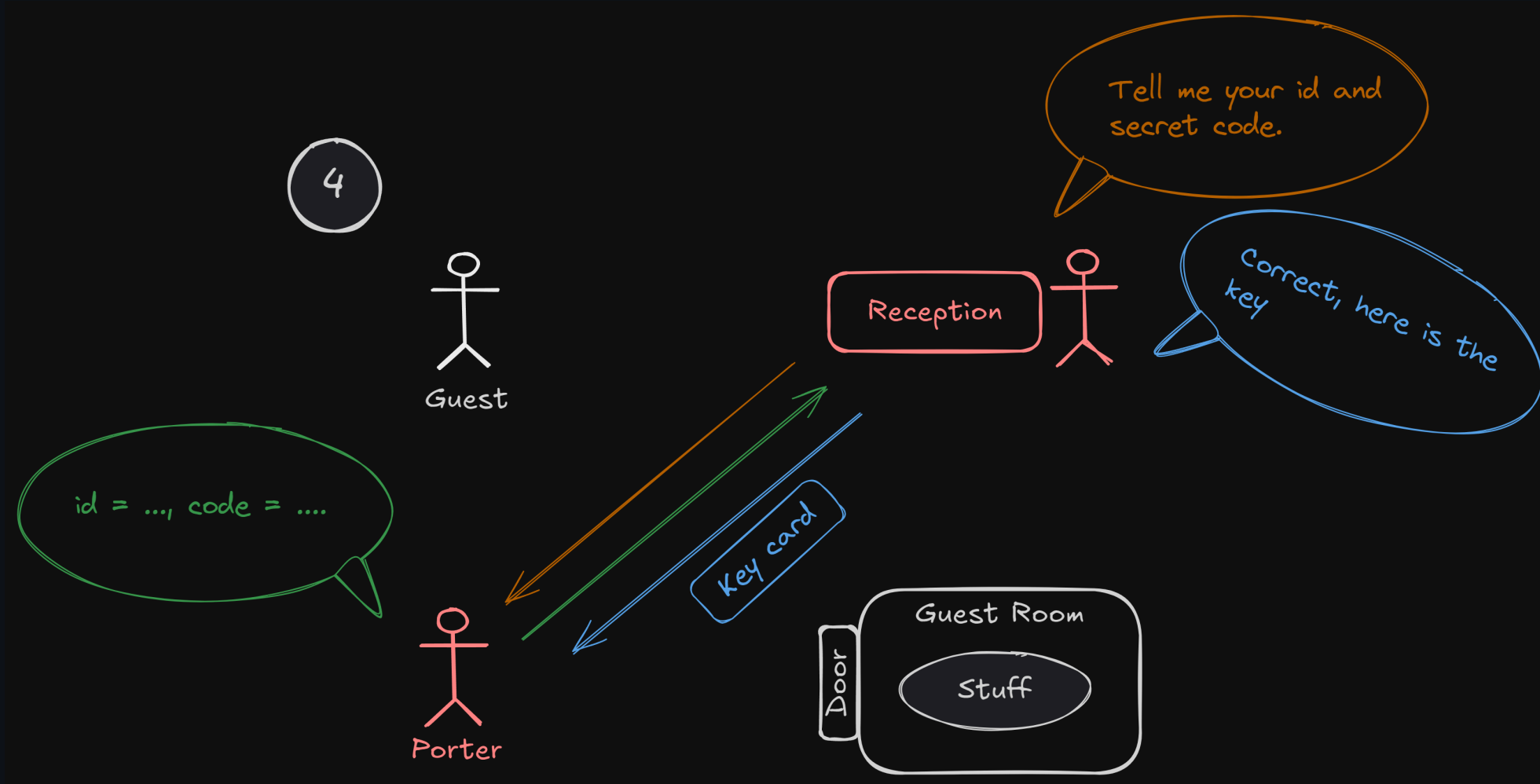
OAuth 2.0 in real life



OAuth 2.0 in real life



OAuth 2.0 in real life



OAuth 2.0 in real life

5

Guest

Reception

Porter

Key card

Guest Room

Door

Stuff

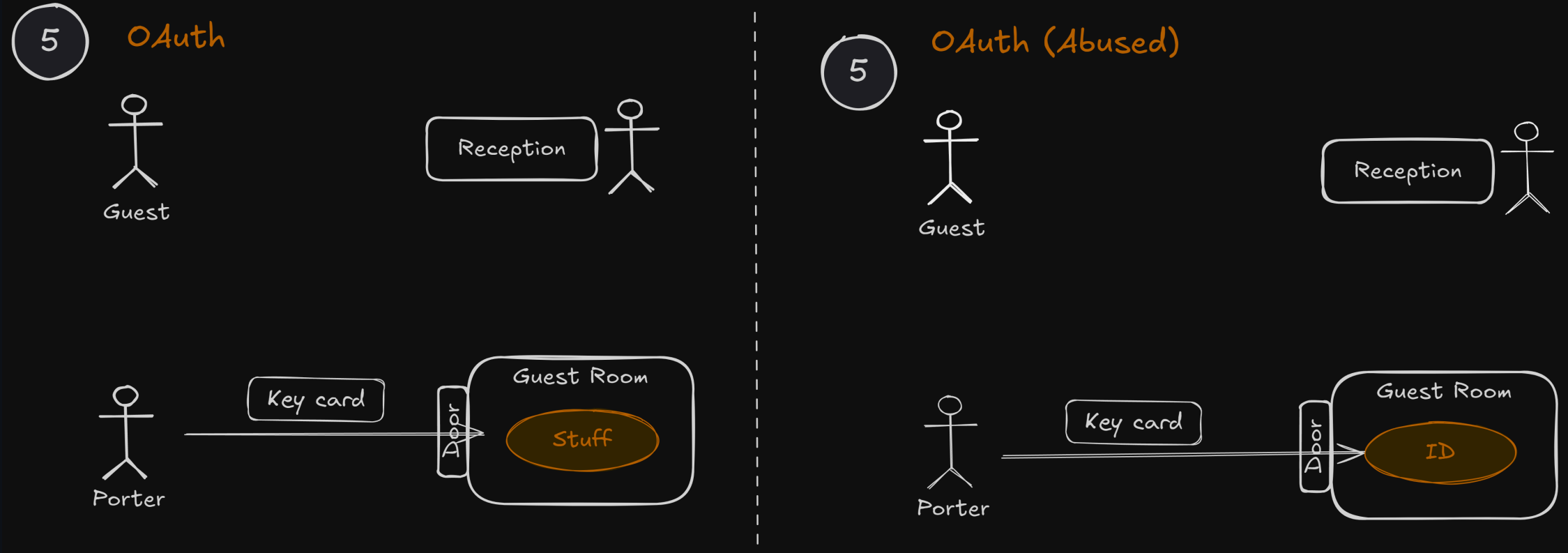
Authorization code flow

- You (`guest`) authorize `porter` to access your resource.
- `porter` does not need to know who you are.
- The keycard reader at the door also doesn't need to have your information.

Authentication?

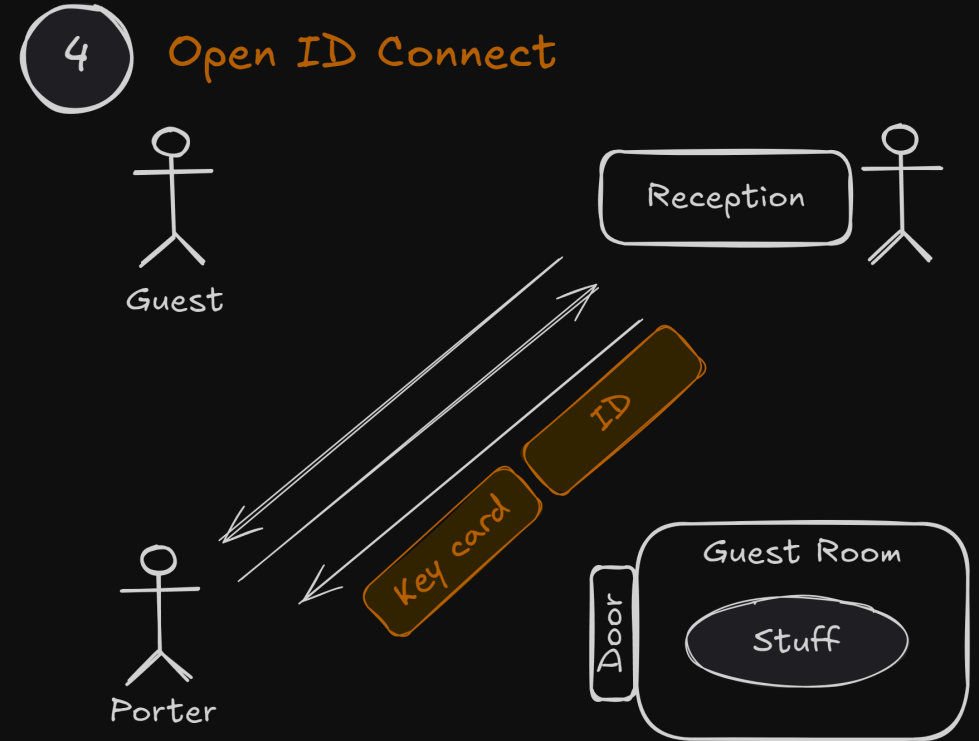
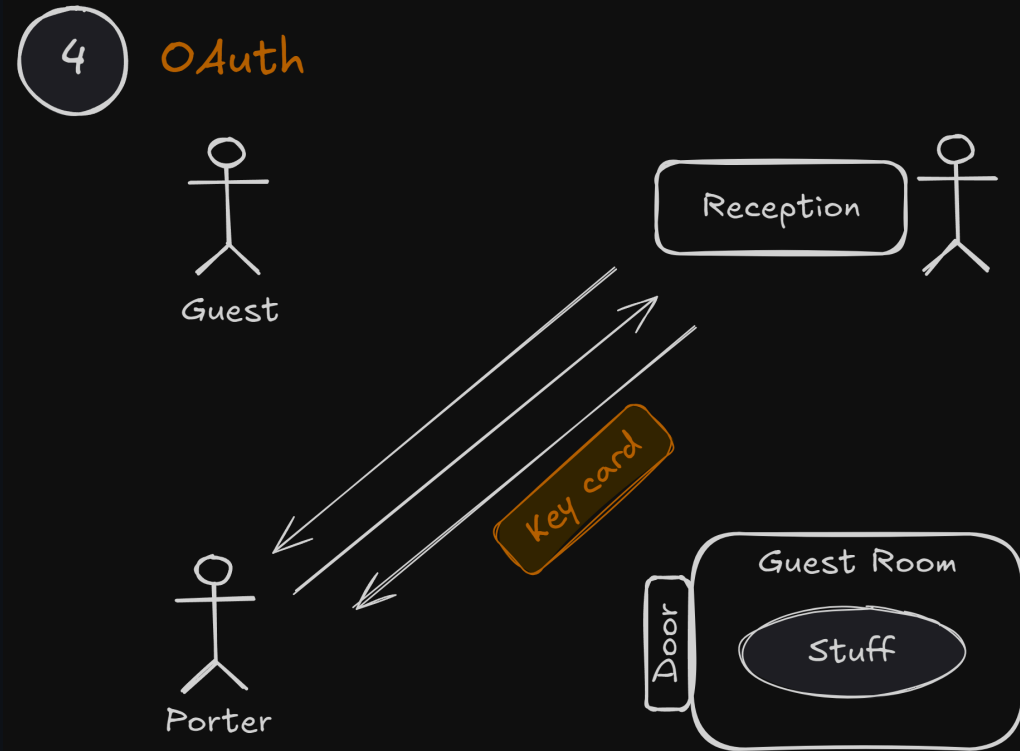
- But what if the porter wants to know who you are.
- There are two ways.

Oauth 2.0 in real life



This is what we are using, but is there a better way?

Oauth 2.0 in real life



OpenID Connect (OIDC)

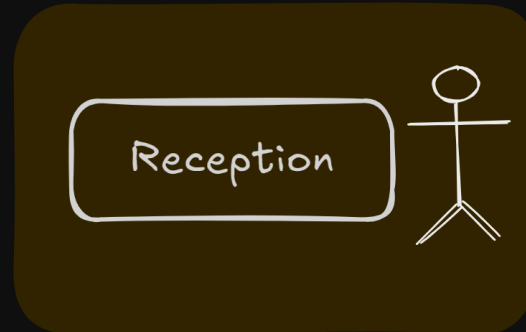
- Thin layer that sits on top of OAuth 2.0
 - Adds login and profile information about the person who is logged in.
- When a "Authorization Server" supports OIDC, it is sometimes called an "Identity Provider".
- Not all servers support OIDC.

Terminology

Resource owner (user)



Authorization server



Clinet (application)



Resource server



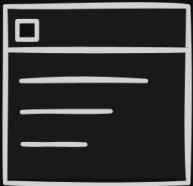
Terminology



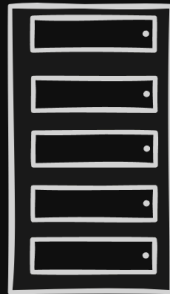
Resource owner (user)



Authorization server

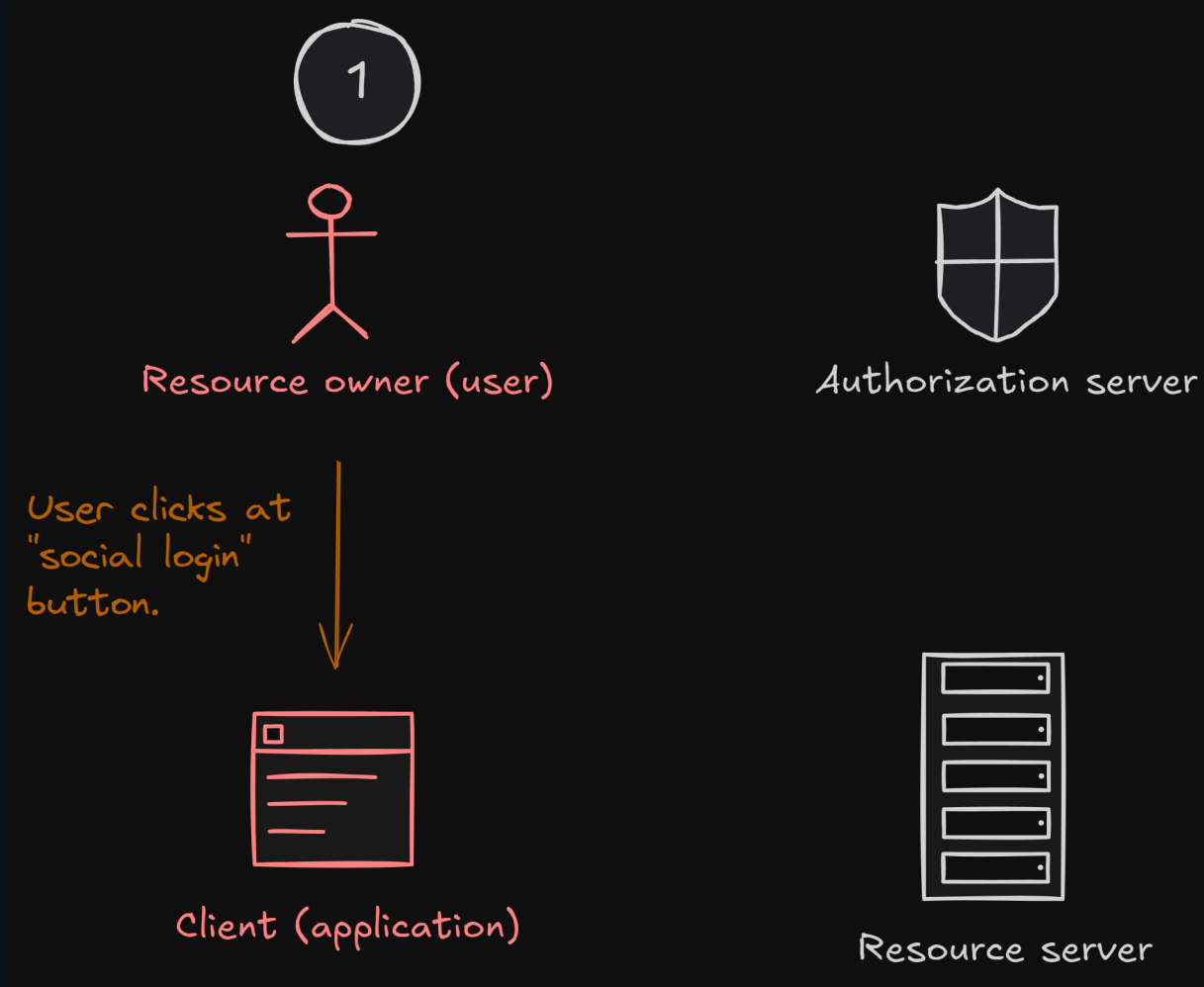


Client (application)

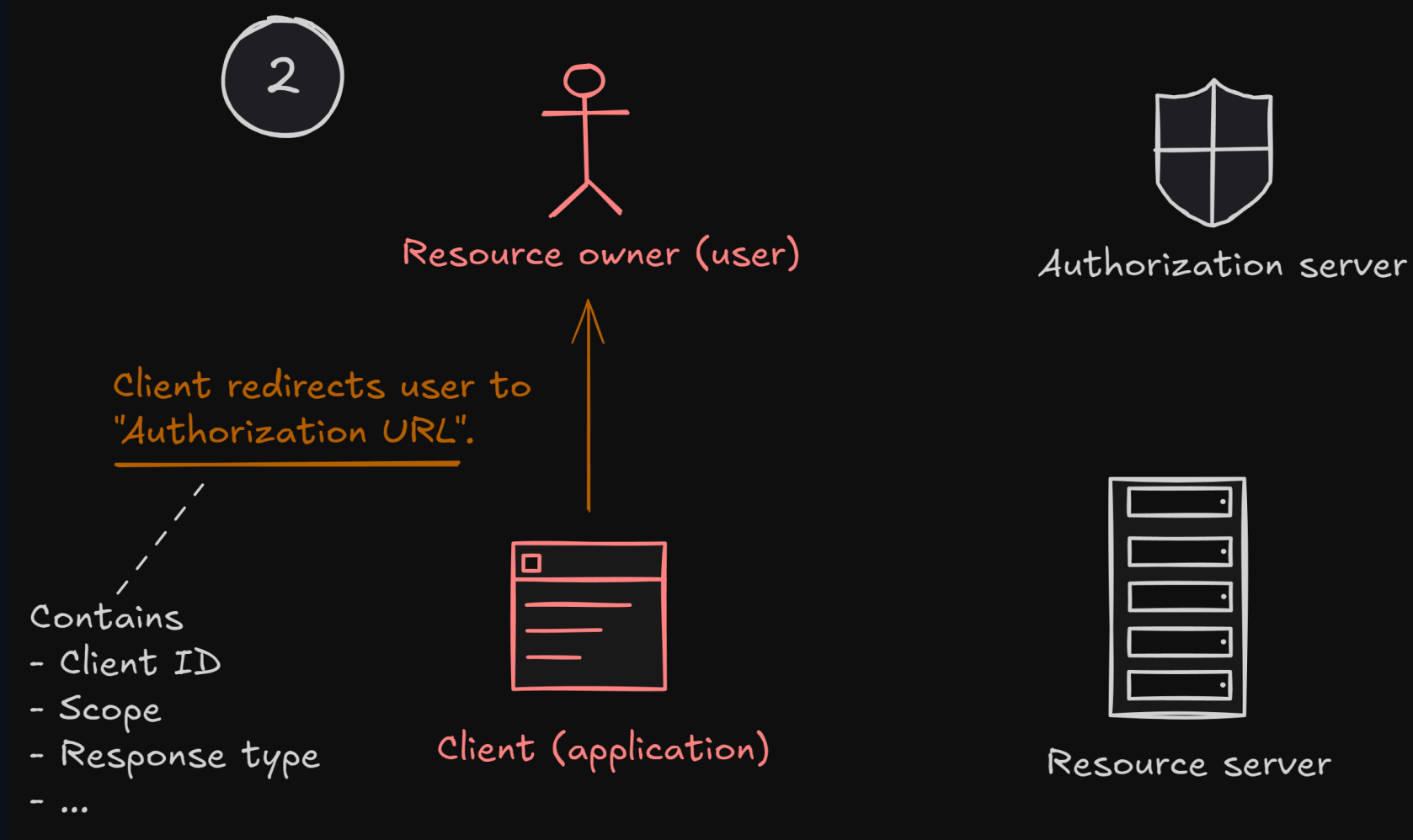


Resource server

OAuth 2.0 (actual)

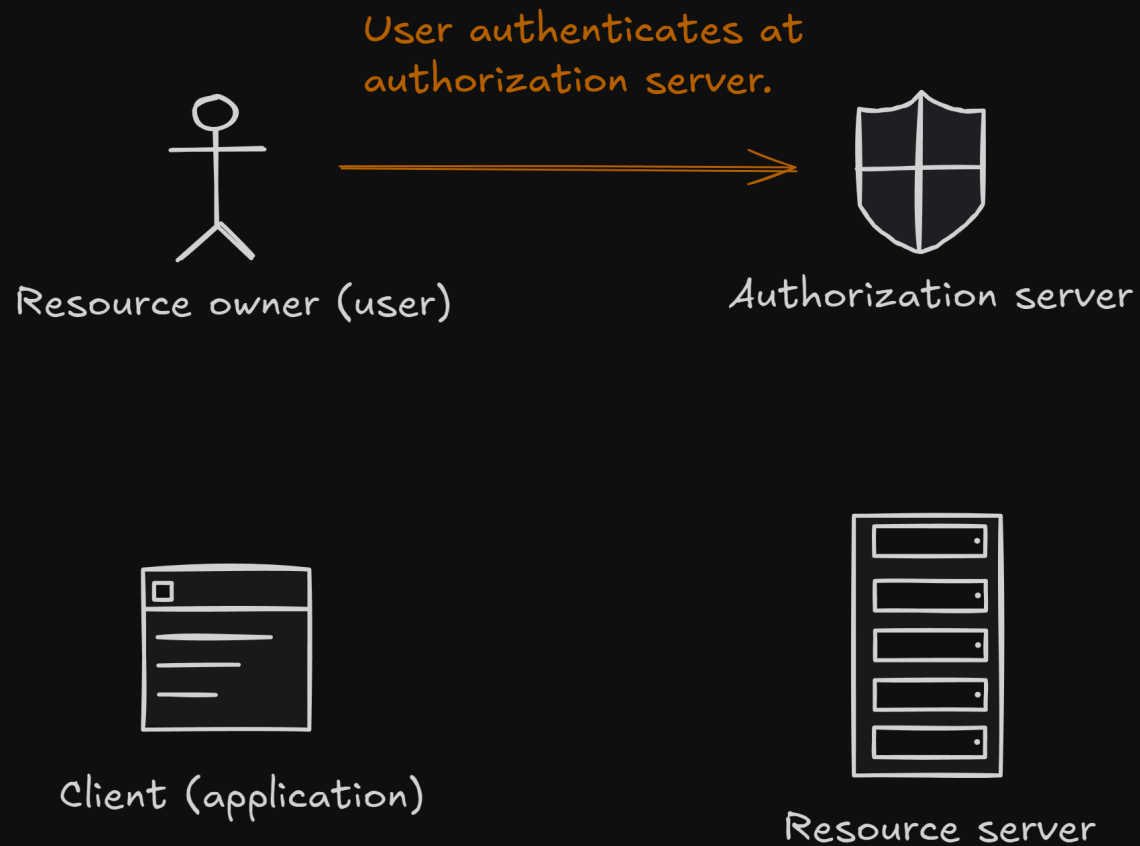


OAuth 2.0 (actual)

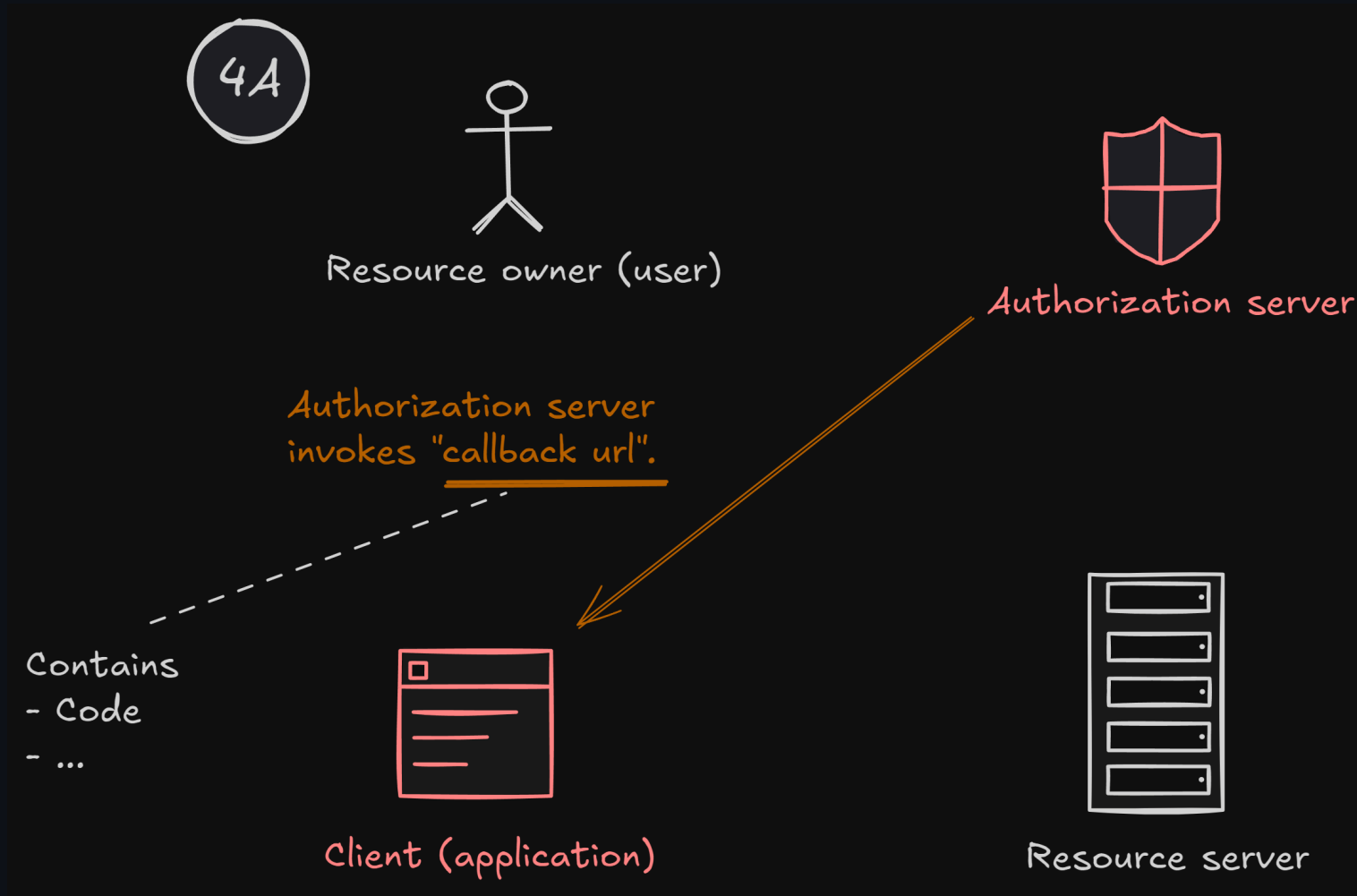


OAuth 2.0 (actual)

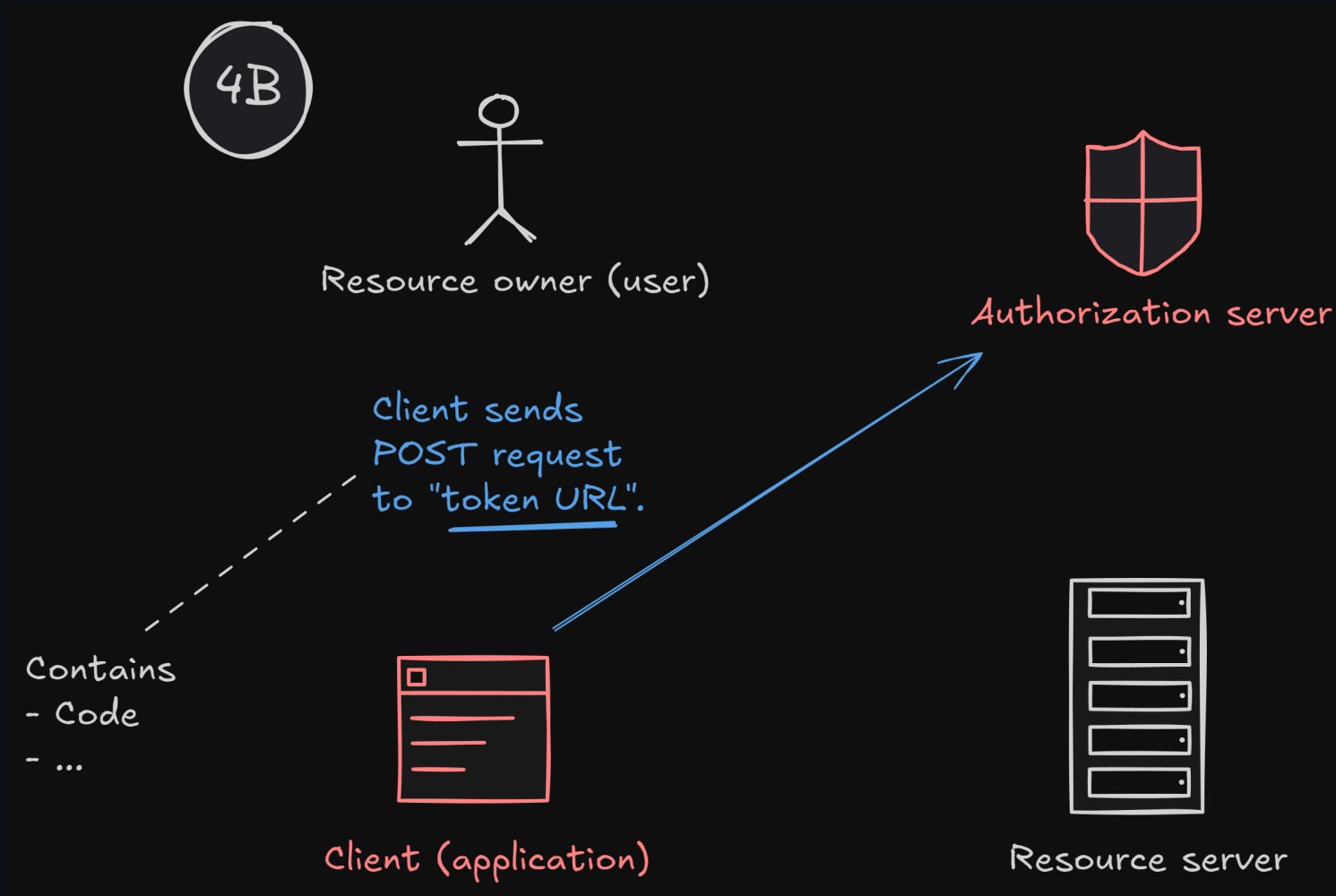
3



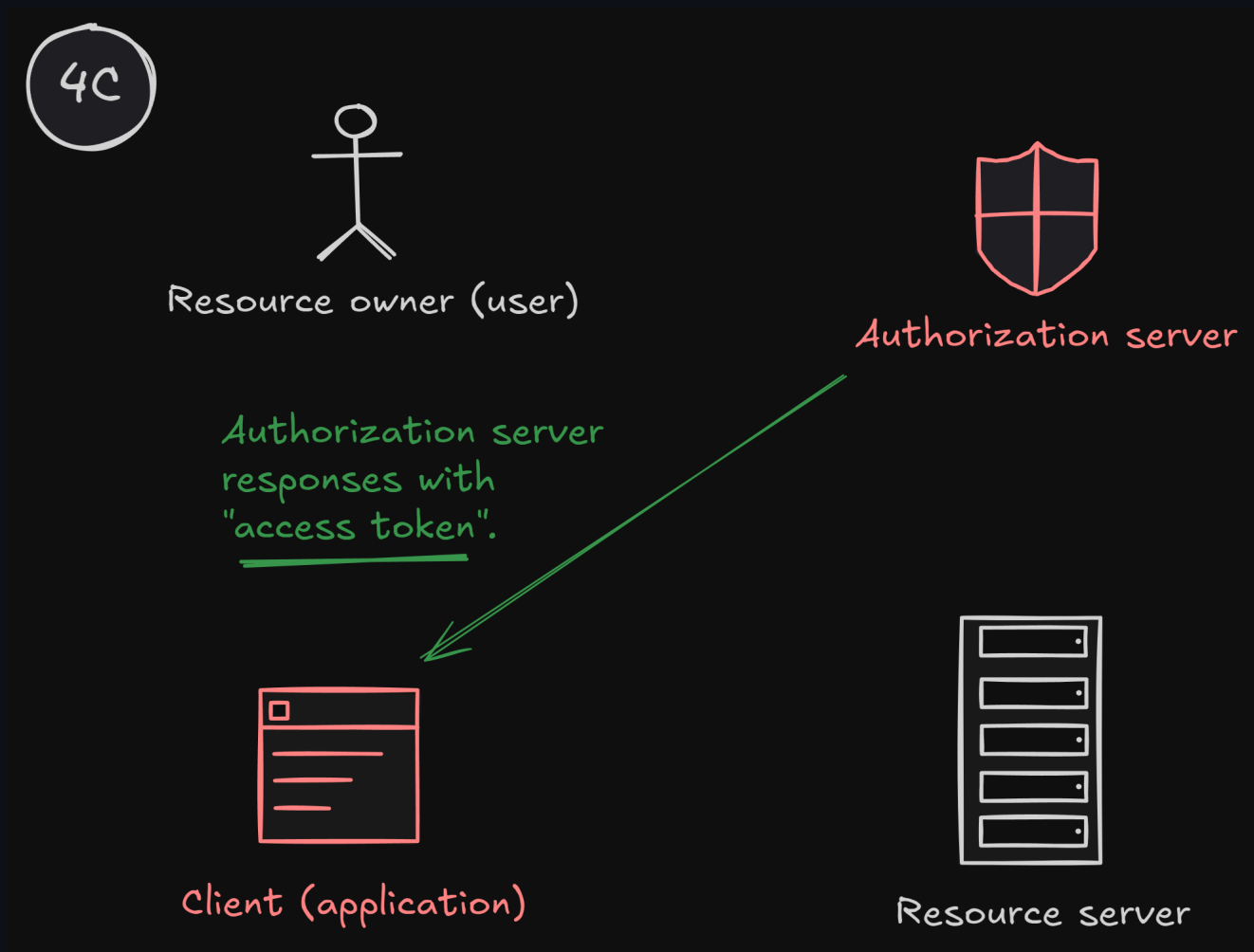
OAuth 2.0 (actual)



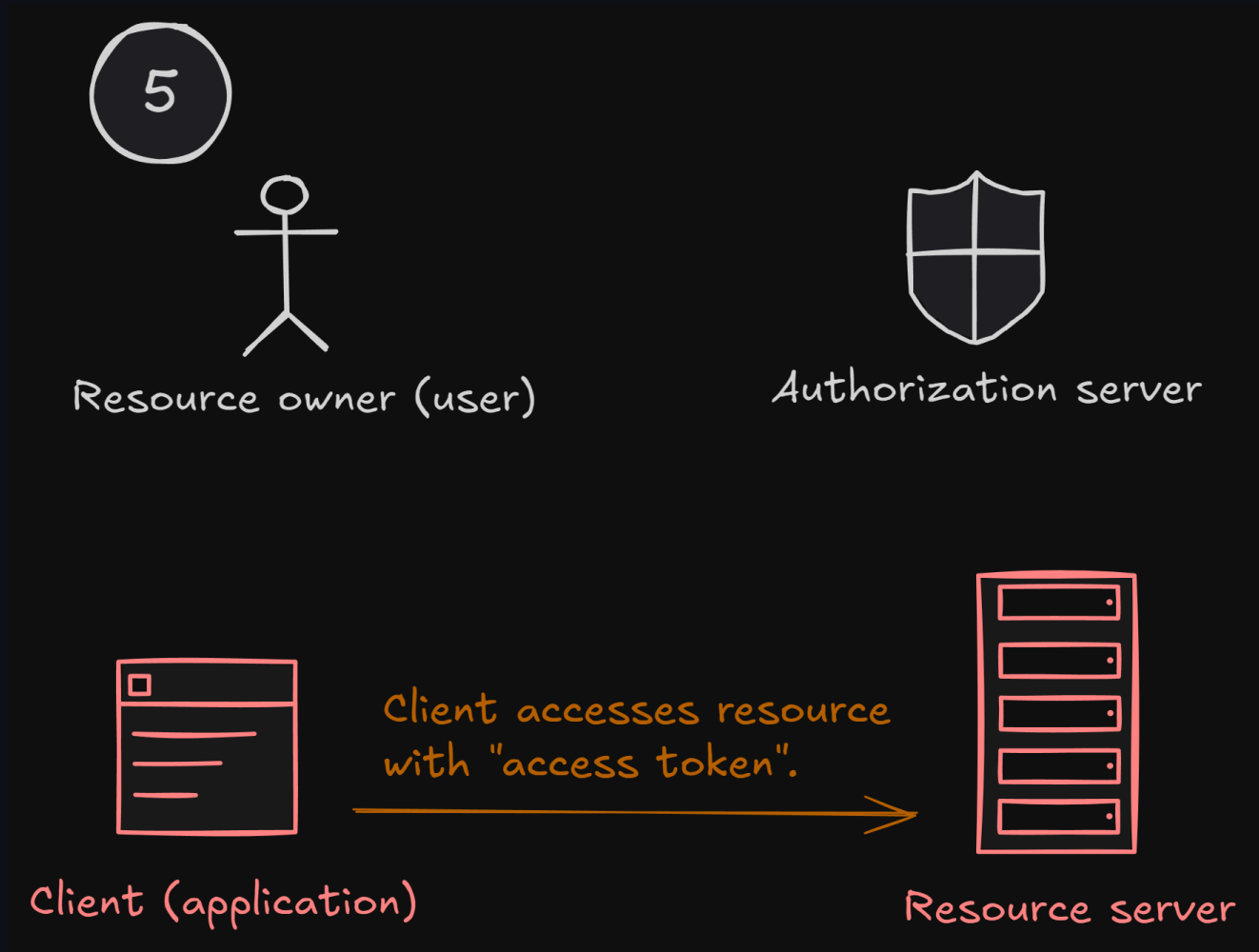
OAuth 2.0 (actual)



OAuth 2.0 (actual)



OAuth 2.0 (actual)



Part 2: Social signing up/in

Section 2B: Oauth 2.0 with Github (Lab)

You will need

- Client ID = ...
- Client Secret = ...
- Callback URL = http://localhost:5001/whatever
- Scope = ...
- Authoriation URL = ...
- Token URL = ...
- Resource URL
 - https://api.github.com/user
 - https://api.github.com/user/emails
- Access Token = ...

Setup

- Register your app [here](#).
- Homepage URL and Callback URL can be whatever for now.

Register a new OAuth application

Application name *

fs-auth-2

Something users will recognize and trust.

Homepage URL *

http://localhost:5001

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *

http://localhost:5001/whatever

Your application's callback URL. Read our [OAuth documentation](#) for more information.

☐ **Enable Device Flow**

Allow this OAuth App to authorize users via the Device Flow.

Read the [Device Flow documentation](#) for more information.

Register application

Cancel

Setup

- Get Client ID and Client Secret

fs-auth-2



nnnpooh owns this application.

Transfer ownership

You can list your application in the [GitHub Marketplace](#) so that other users can discover it.

List this application in the Marketplace

0 users

Revoke all user tokens

Client ID

0v231iwDq5jdIXDiQXI2

Client secrets

Generate a new client secret

Make sure to copy your new client secret now. You won't be able to see it again.



Client secret

✓ 23b7d193673020a537c62697312eb50a14353937

Added now by nnnpooh

Never used

You cannot delete the only client secret. Generate a new client secret first.

Delete

Setup

- Choose `scope`.
 - `scope=user,user:email`
- Construct `Authorization URL`
 - `https://github.com/login/oauth/authorize?client_id=CLIENT_ID&redirect_uri=REDIRECT_URL&response_type=code&scope=SCOPE`

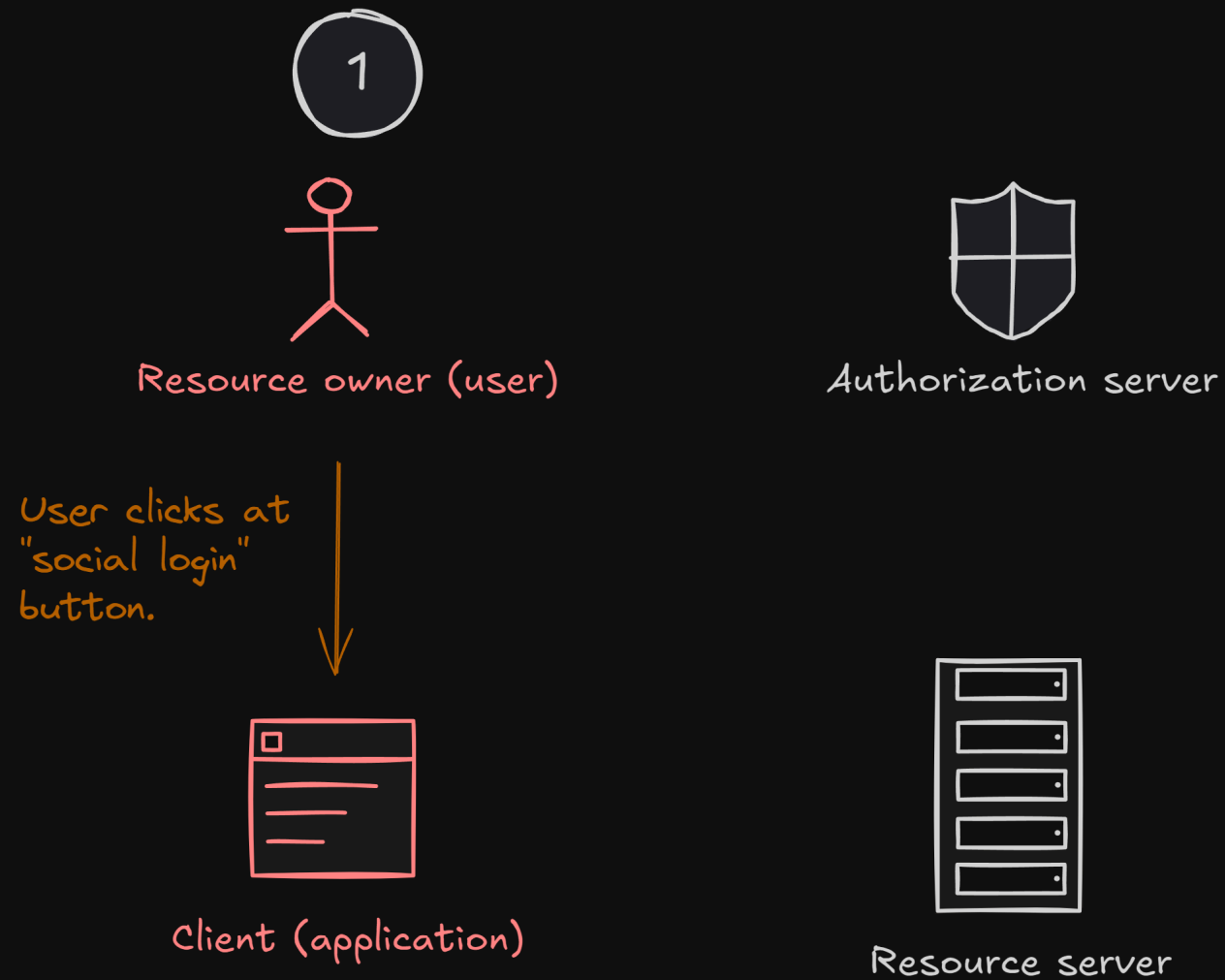
Setup

- Construct Token URL (*incompleted*)
 - `https://github.com/login/oauth/access_token?`
`client_id=CLIENT_ID&client_secret=CLIENT_SECRET&code=CODE&redirect_uri=CAL`
`LBACK_URL`

Let's go

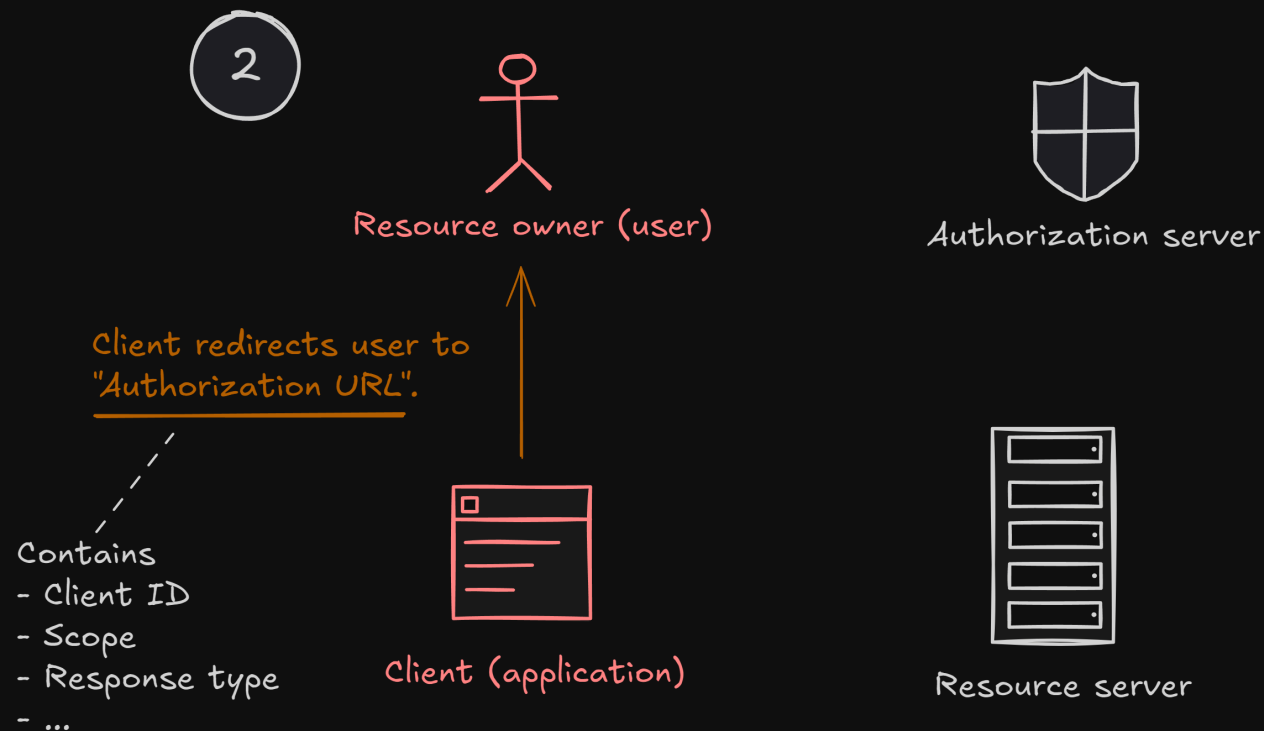
Step 1

- Do nothing.
- *Pretend that your app has social signin button.*



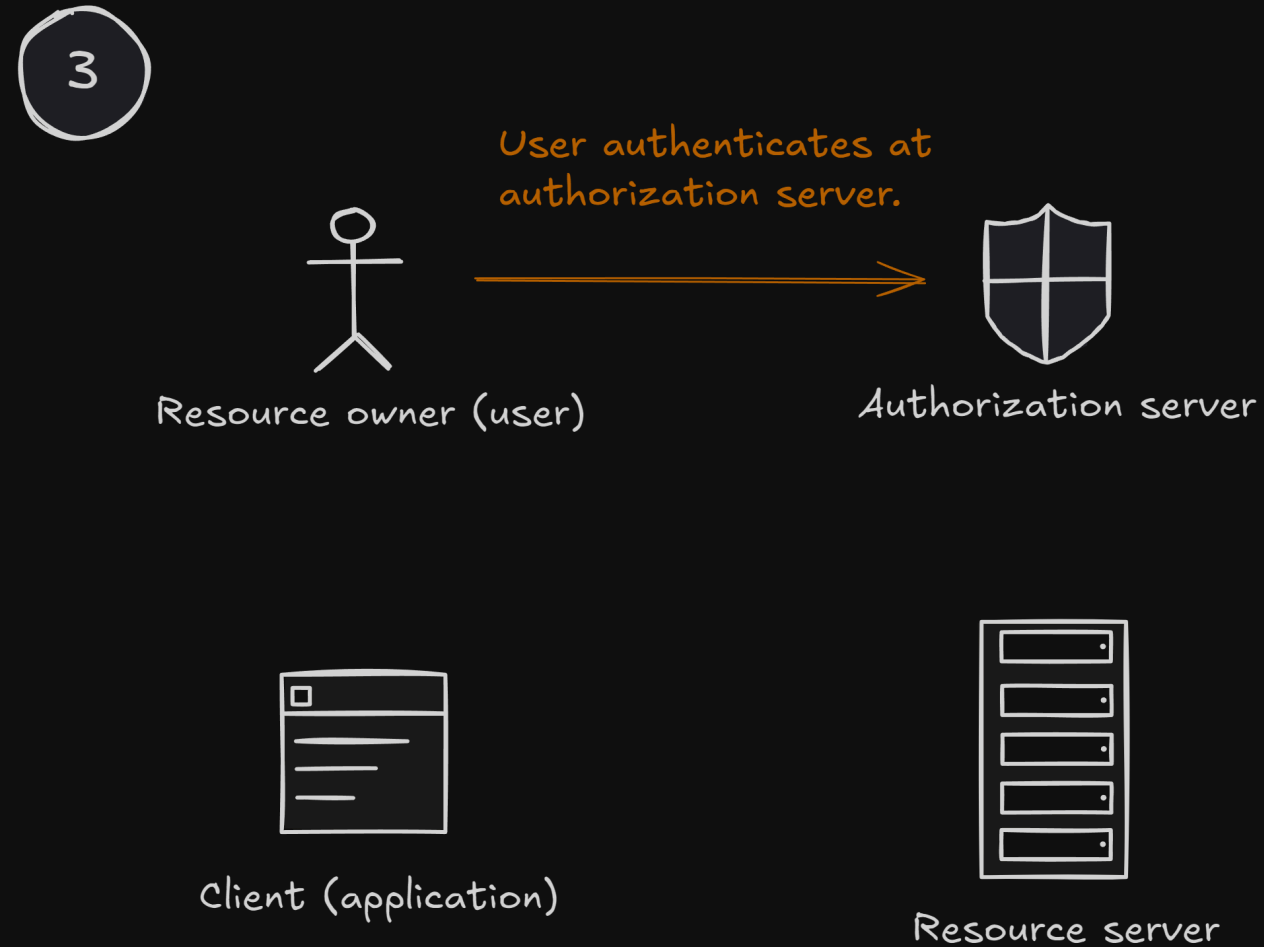
Step 2

- Paste the `Authorization URL` in the address bar.
- *Pretend that this is done from url redirection.*



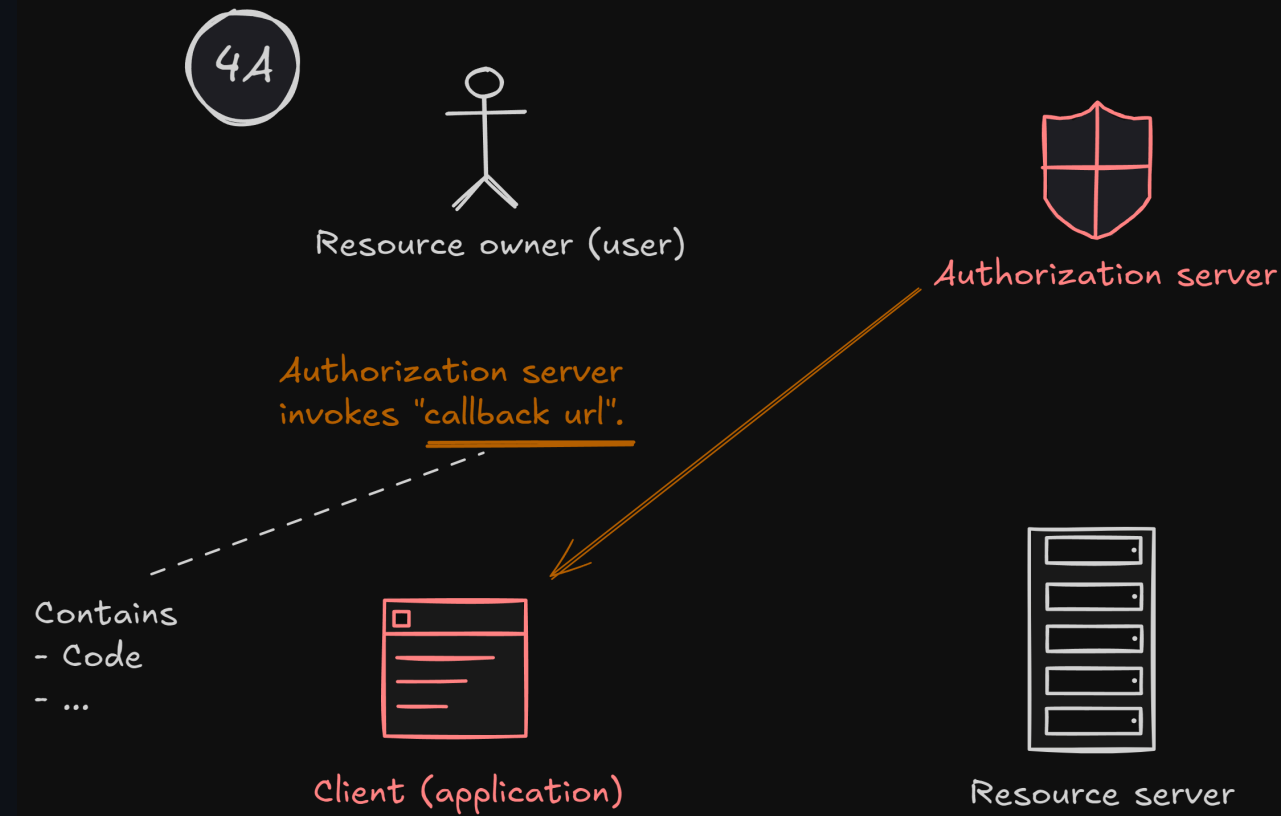
Step 3

- Authenticate at Github.



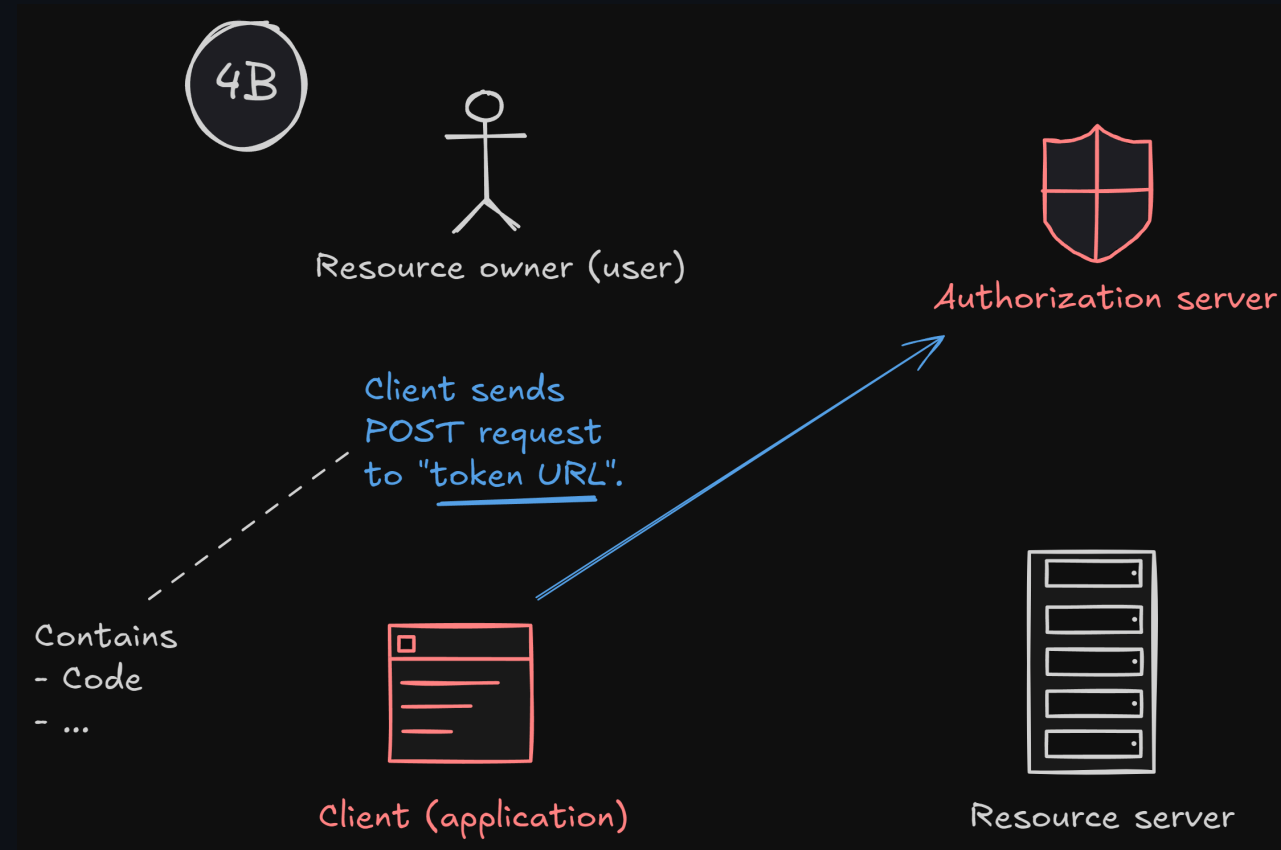
Step 4A

- Extract **Code** and keep it.
- **Code** is usually very short-lived.



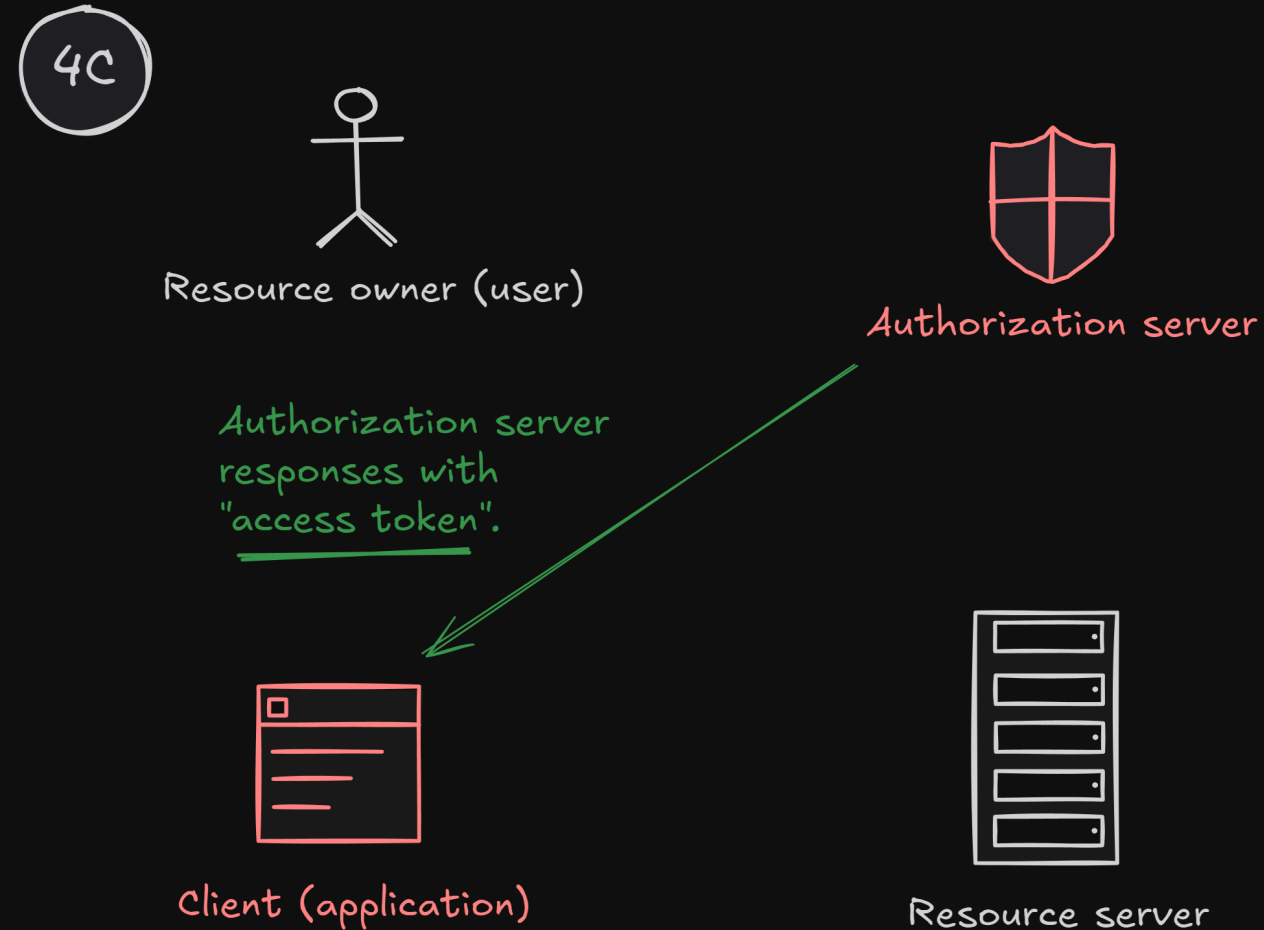
Step 4B

- Send **POST** request to **Token URL** with actual **Code**.
- [Reference](#)



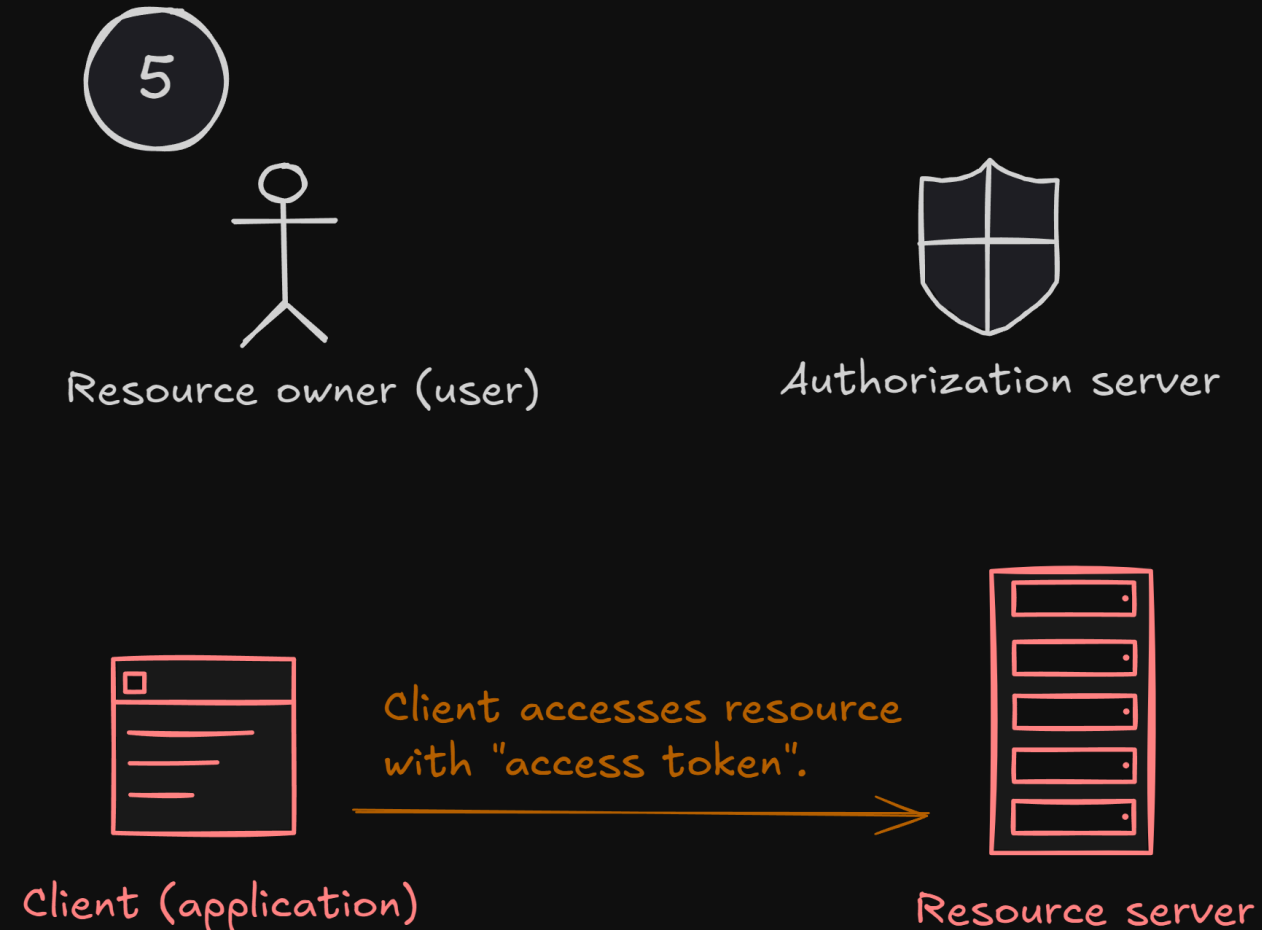
Step 4C

- Keep **Access Token** from the response.



Step 5

- Send **GET** request to **Resources URL**.
- Use **Access Token** as bearer token.
- [Reference](#)



Search... Ctrl + P

Invite nnnpooh@gmail.com

Auth GET https://api.github.com/user Send 200 OK 491 ms 1421 B 5 Minutes Ago

Base Environment Add Cookies Add Certificates

Filter GET New Request POST New Request

URL PREVIEW https://api.github.com/user

Params Body Auth Headers 3 Scripts Docs

QUERY PARAMETERS

name	value

PATH PARAMETERS

Path parameters are url path segments that start with a colon ':' e.g. 'id'

Preview

```
12 "starred_url": "https://api.github.com/users/nnnpoooh/starred{/owner}/{/repo}",
13 "subscriptions_url": "https://api.github.com/users/nnnpoooh/subscriptions",
14 "organizations_url": "https://api.github.com/users/nnnpoooh/orgs",
15 "repos_url": "https://api.github.com/users/nnnpoooh/repos",
16 "events_url": "https://api.github.com/users/nnnpoooh/events{/privacy}",
17 "received_events_url": "https://api.github.com/users/nnnpoooh/received_events",
18 "type": "User",
19 "site_admin": false,
20 "name": "Nirand",
21 "company": "Chiang Mai University",
22 "blog": "",
23 "location": "Thailand",
24 "email": null,
25 "hireable": null,
26 "bio": null,
27 "twitter_username": null,
28 "notification_email": null,
29 "public_repos": 49,
30 "public_gists": 32,
31 "followers": 6,
32 "following": 0,
33 "created_at": "2019-10-05T05:21:33Z".
```

\$.store.books[*].author

Online Made with ❤ by Kong

Google OAuth 2.0

- Authorization URL = `https://accounts.google.com/o/oauth2/v2/auth?client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&response_type=code&scope=openid+https://www.googleapis.com/auth/userinfo.email+https://www.googleapis.com/auth/userinfo.profile`
- Token URL = `https://oauth2.googleapis.com/token?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&code=CODE&redirect_uri=CALLBACK_URL&grant_type=authorization_code`
- Resource URL = `https://www.googleapis.com/oauth2/v2/userinfo`

- ```
id_token .
```

# JSON Web Token



Part 2: Social signing up/in

## Section 2C: passport implementation

# Setup

- `git clone -b signin-oauth https://github.com/fullstack-67/auth-mpa-v2.git`  
`auth-signin-oauth`
- Fill in `.env`
  - Make sure to update `Callback URL` in your Github OAuth app.
- `pnpm i`
- `npm run db:reset`
- `npm run dev`

# Note

- Database tables
  - `users` and `accounts` tables.
  - `many-to-one` relations.
  - Composite key in `accounts` table to avoid duplicated providers / user.
- Types of response objects from API.
  - `./src/types/github.ts`
  - `JSON schema`, `QuickType`
- Add type definition to `req.user`.
  - `./src/types/express.d.ts` `What?`

# Highlighted packages

```
{
 "passport": "^0.7.0",
 "passport-oauth2": "^1.8.0"
}
```

# Middleware setup

```
// * Passport
passport.use("github", github);
app.use(passport.initialize());
```

# Strategy setup

```
export const github = new OAuthStrategy(
 {
 // Option
 },
 async function (
 accessToken: string,
 refreshToken: string,
 profile: any,
 done: VerifyCallback
) {
 // Do something with accessToken
 }
);
```

# Routing

Redirect to `Authorization URL`

```
app.get("/login/oauth/github", passport.authenticate("github"));
```

# Routing

Receive `code` from `Callback URL`

```
app.get(
 "/callback/github",
 passport.authenticate("github", {
 failureRedirect: "/login",
 }),
 function (req, res) {
 // If successful, do something with req.user.
 });
```



# Using OAuth library

- No need to construct `Authorization URL` manually.
- No need to write logic for steps `4A`, `4B` and `4C`.
- If you use `passport-github2`, you can also skip step `5`.

# Shortcoming

- You will see that users need to constantly sign in to access the main page.
- We need to persist users' auth states.

**Next: Part 3**