OAuth 2.0

Web API

- Many apps have APIs that can be used to interact with user data programmatically

- Such apps will [typically] allow users to access their data in 2 ways:
  - Using the app itself - loading the page and interacting with the UI (User Interface)
  - Connecting to the API - Sending HTTP requests directly to the server without using the front end

- Using the API allows us to write custom programs that interact with the app
  - eg. A programs that starts/stops Spotify playback

Web API

- Web APIs use endpoints
- API Endpoint: A combination of path and HTTP method that has specific behavior

- Examples:
  - POST /chat-message - Adds a message to chat
  - DELETE /chat-message/<mesageId> - delete the chat message with an id matching <messageId>
  - PUT api.spotify.com/v1/me/player/play - begin music playback
  - POST api.github.com/repos/<owner>/<repo>/issues - create an issue in the repo <owner>/<repo>
  - GET api.github.com/repos/<owner>/<repo>/issues - get all issues in the repo <owner>/<repo>

Web API

- How do we securely consume an API?

- The API server *can* verify an existing authentication token
  - These tokens were not designed for API access
    - Gives full access to the account without restriction

- More commonly, the API will issue an API key to the user
  - Send this key with each API access
  - Server verifies the user associated with the key for *authorization*
  - Keys can have restricted functionality and are only used for API access (Not as detrimental if compromised)

Web API

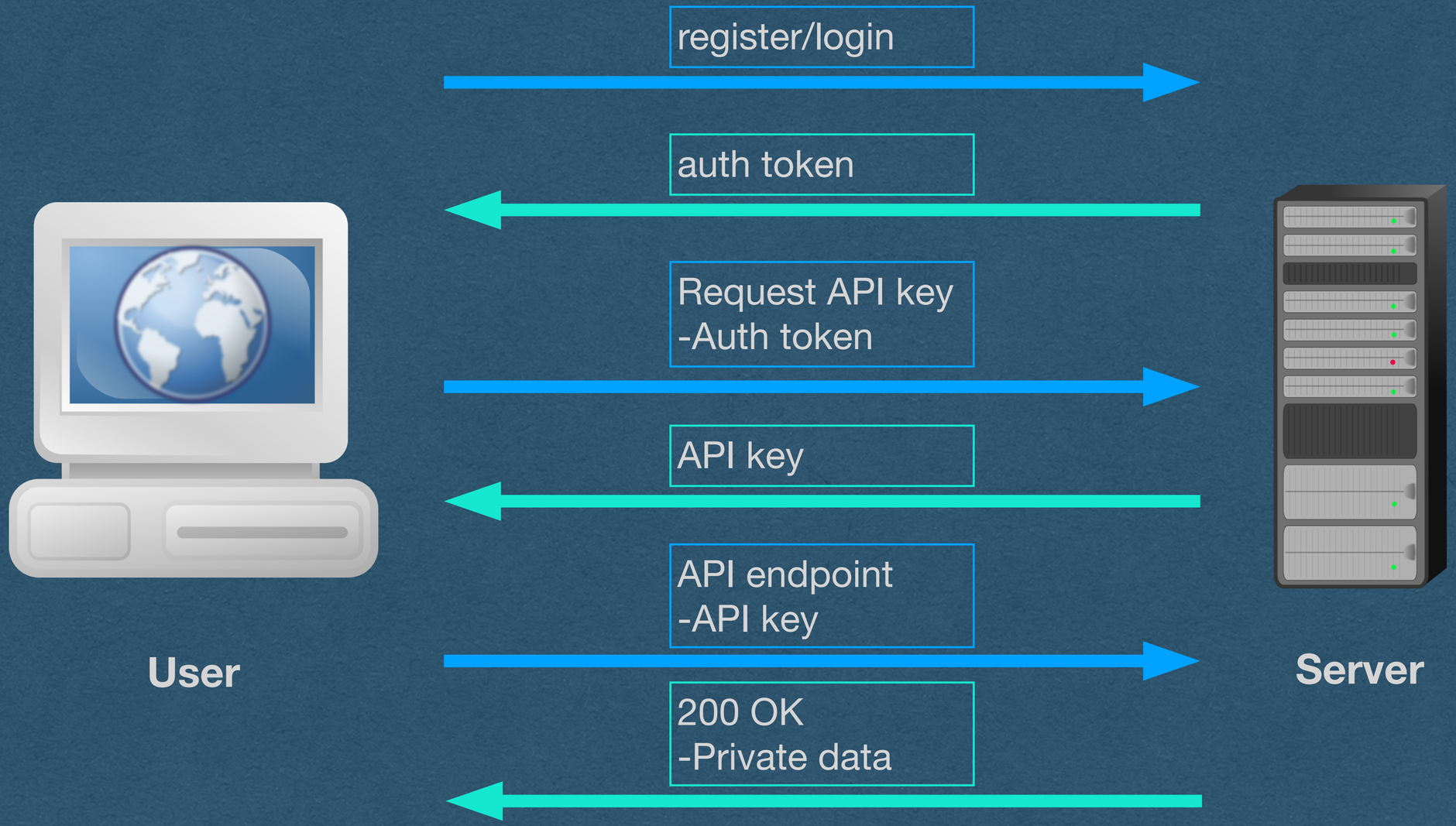
- How do we securely consume an API?

- The API server *can* verify an existing authentication token
  - These tokens were not designed for API access
    - Gives full access to the account without restriction

- More commonly, the API will issue an API key to the user
  - Send this key with each API access
  - Server verifies the user associated with the key for *authorization*
  - Keys can have restricted functionality and are only used for API access (Not as detrimental if compromised)

Web API

- Authenticate with username and password
- Request an API key
- Use the API key to access private data from the API



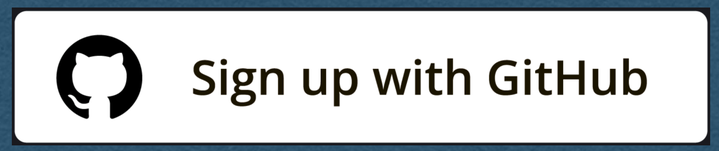
OAuth?

- This setup works well

- So where does OAuth come in?

The Problem

- A user enjoys an app (eg. GitHub) that has a web API
- You want to write a app that consumes the GitHub API on behalf of your users. Examples:
  - You're building a bug reporting app that creates GitHub issues for your users
  - You want users to access their private repos through your app
  - You want a "sign up with GitHub" button on your app



The Problem

In general, You want to write an app that uses a 3rd party API to access/modify your users private data

How do we do this ***securely***?

A BAD Attempt

- **Never do this!!**

- ..Have your users give you their GitHub username and password

- Effective, but **very insecure**
- Never ask users for their password outside of registration/login.
  - We did a lot of work hashing/salting to make sure we can't know passwords
- This would require us to store plaintext passwords so we can reuse them each time a user wants to access the API through our app

Another BAD Attempt

- **Never do this!!**

- ..Have the user give you their GitHub API key

- No nearly as bad as storing their password
- Vulnerable to several different attacks (Discussed as we introduce solutions)
- API key rate limiting will count against the user when we use the key
  - Problem if the user gets denied access because your app overused the key
  - **Big problem** if the API charges $ per access

OAuth 2.0

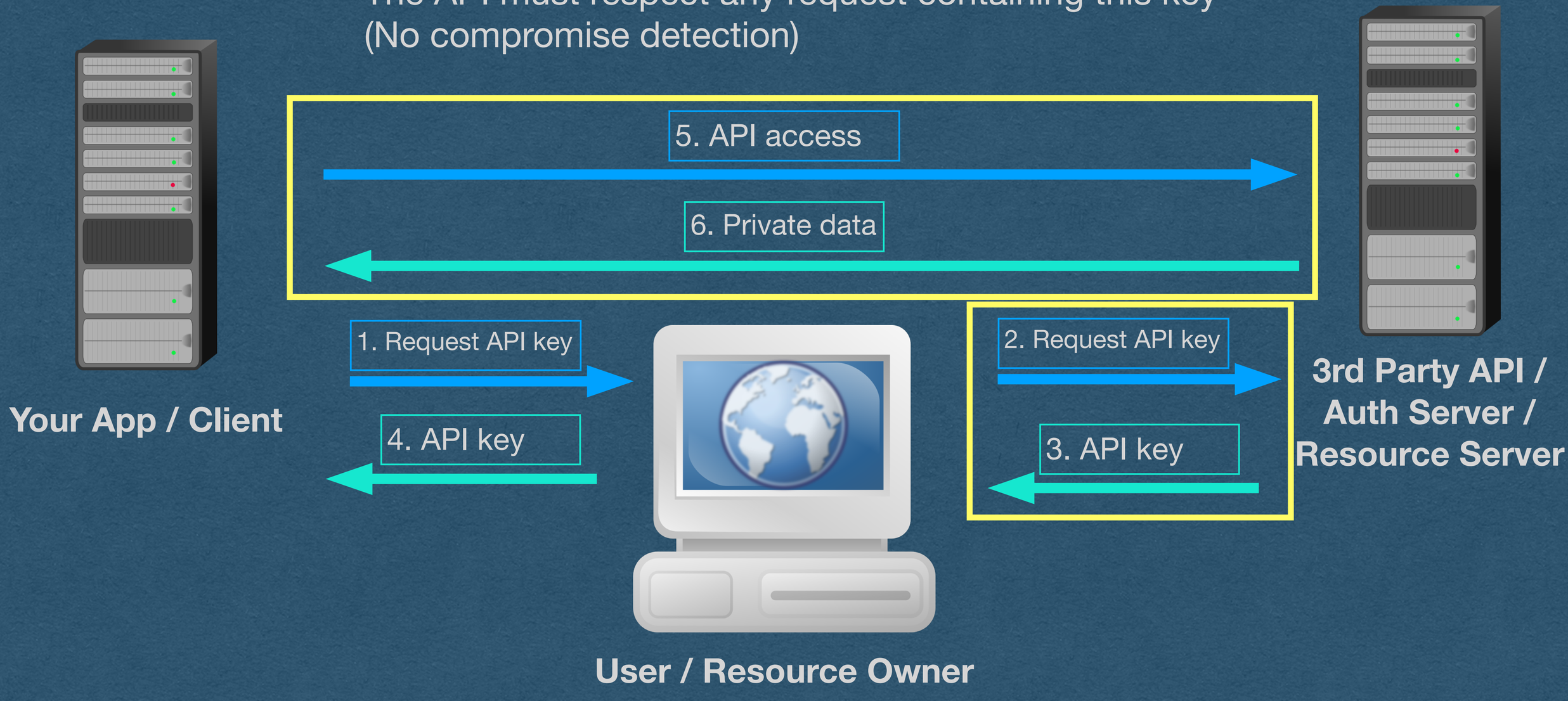
- OAuth 2.0 (Open Authorization 2.0)
- The current, most widely used, solution to this problem

- Designed to allow apps to use 3rd party APIs on behalf of their users in a secure way
  - User still has to trust the app with their data
    - They are explicitly giving the app permission to access their private data so this should be assumed

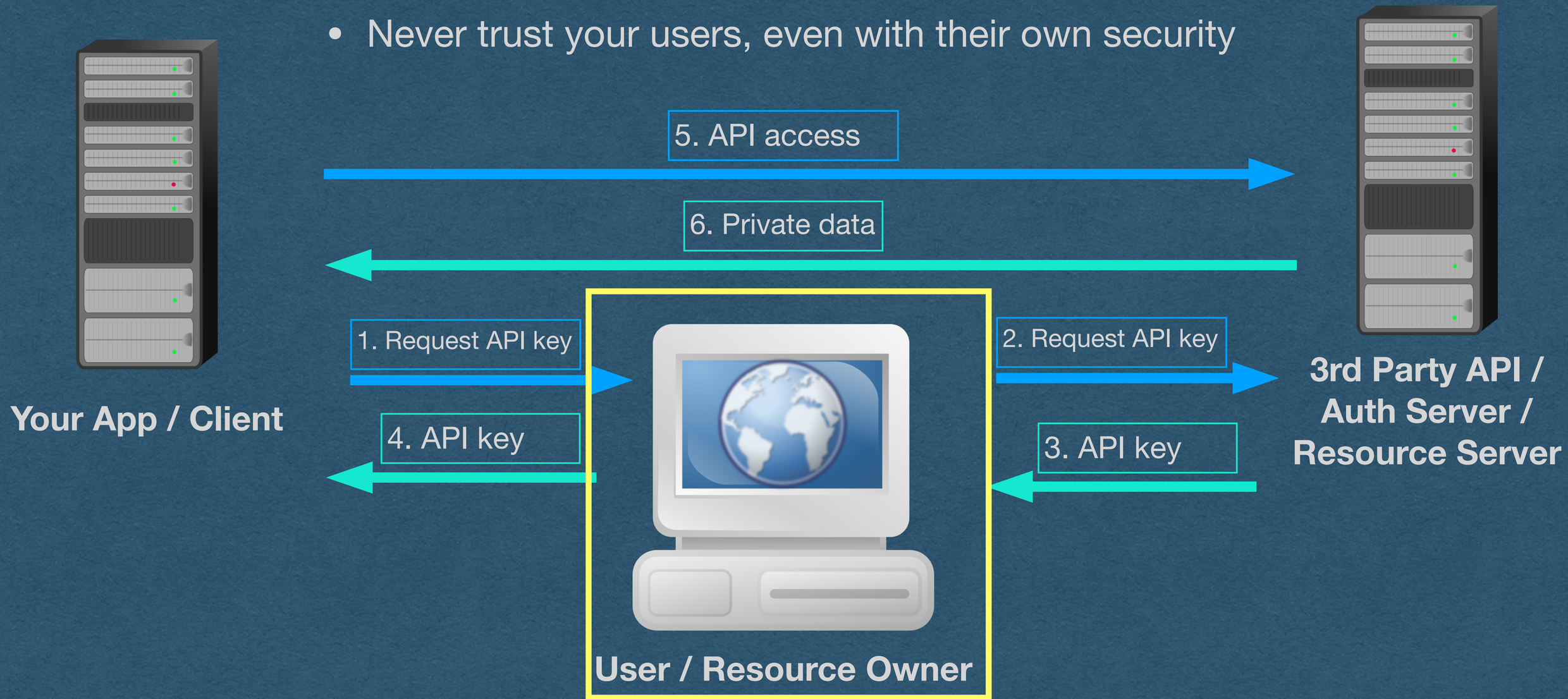
- The handling of this access is secure
  - Protected from outside attackers

- What we want:
  - App asks user for their API key
  - User obtains an API key and sends it to our app
  - Our app uses the key to make API requests

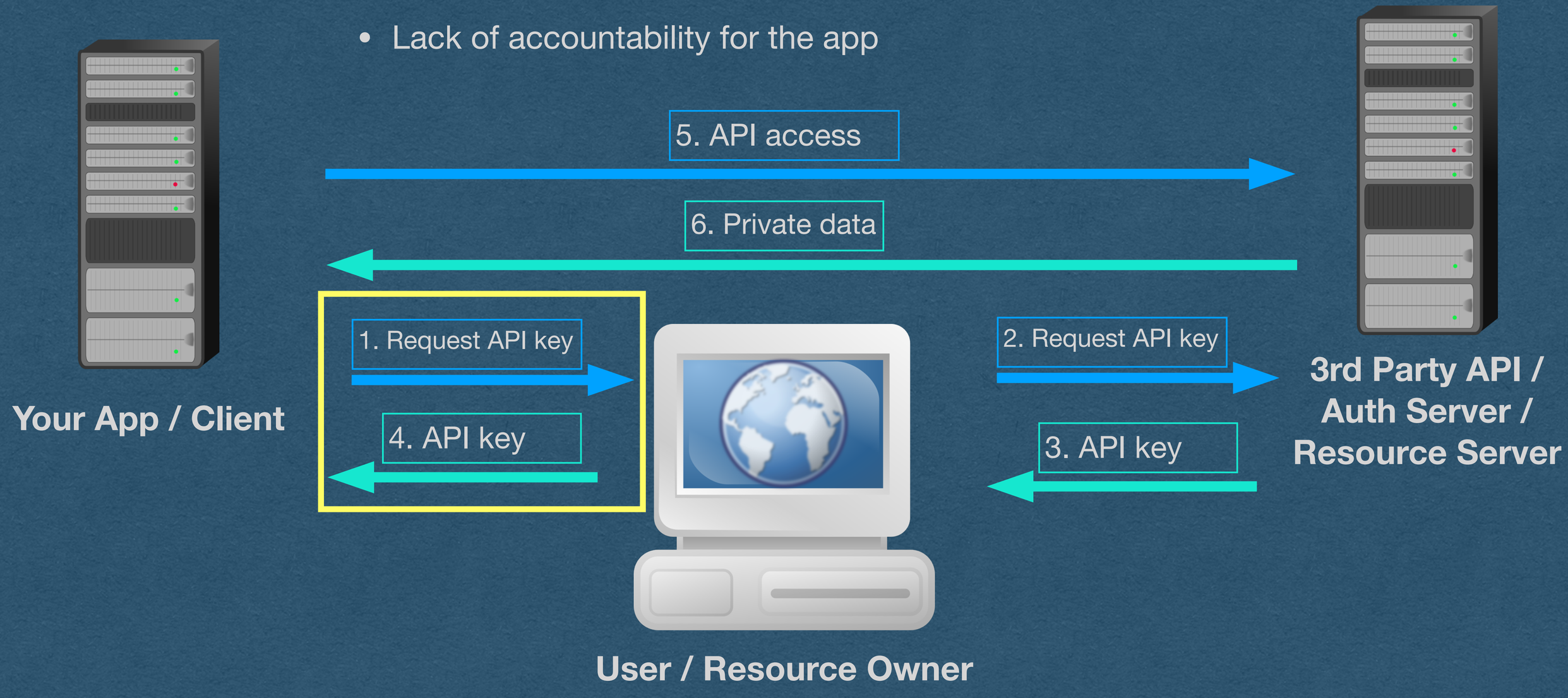
- Security issues:
  - The API has no idea that user allowed your app to use this key
  - The API must respect any request containing this key (No compromise detection)



- Security issues:
  - The user has to handle their own API key
    - Key can be compromised at this point
  - Never trust your users, even with their own security



- Security issues:
  - Your server obtains the user's API key
  - Make API requests *as* the user (As opposed to *on behalf* of the user)
    - Lack of accountability for the app

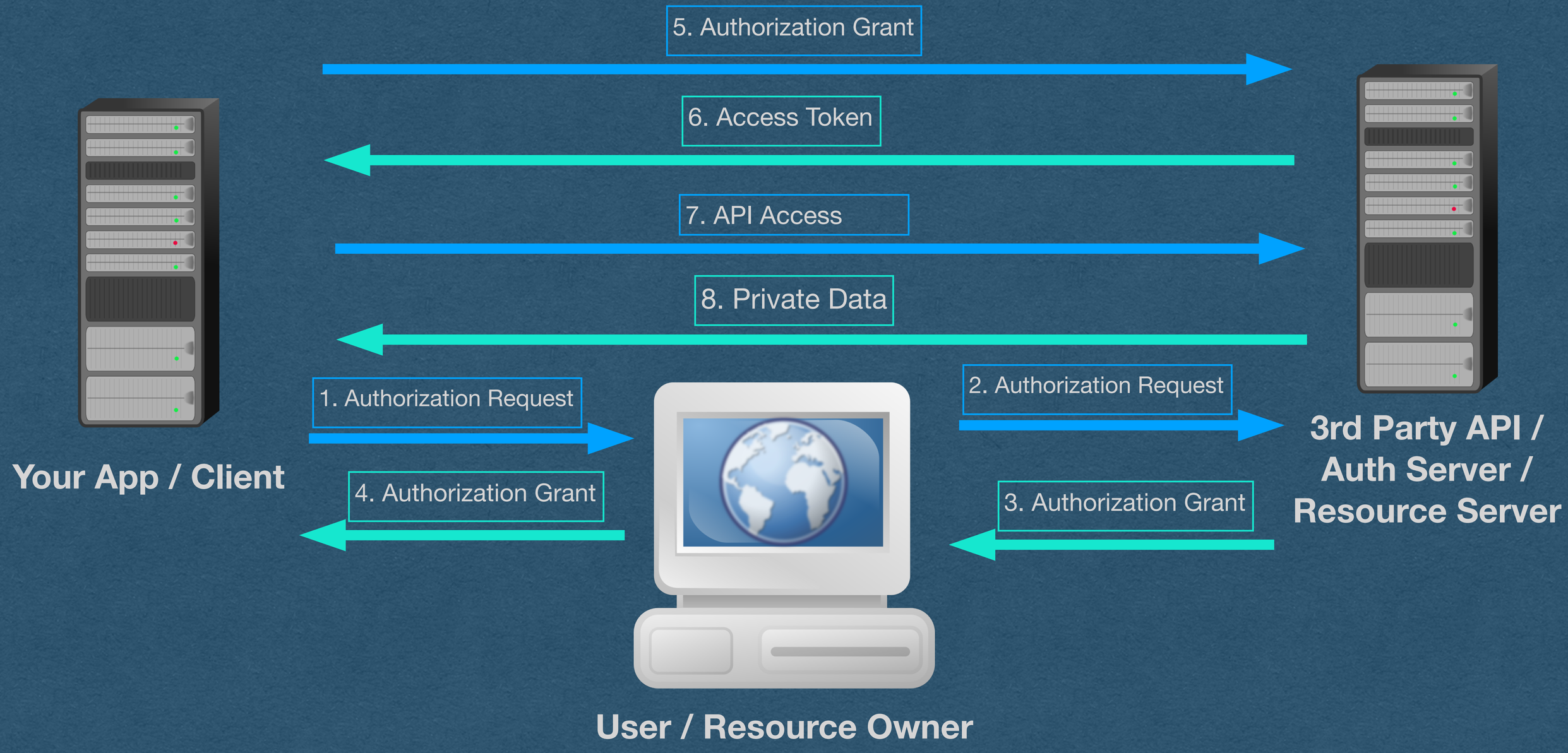


OAuth 2.0

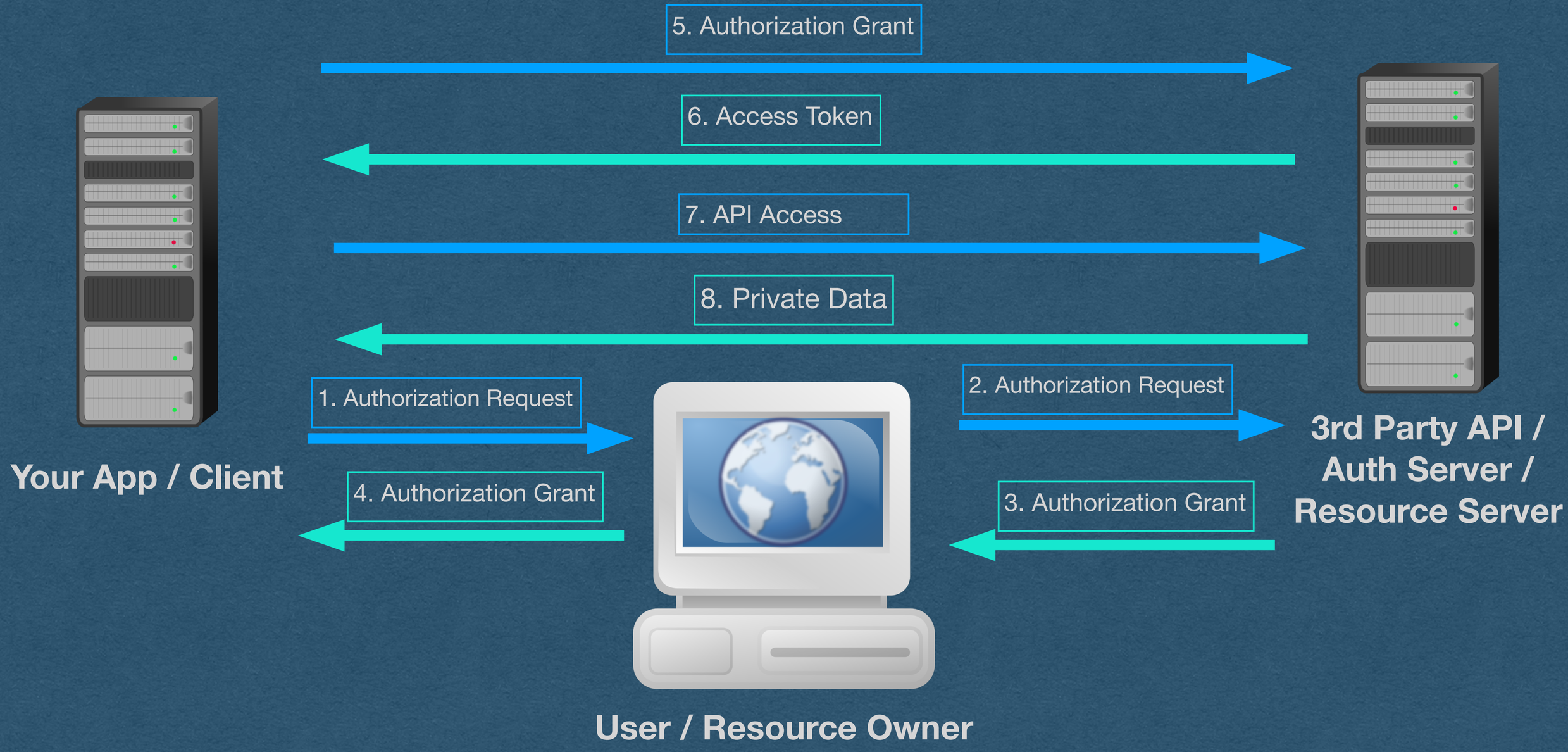
OAuth 2.0 will fix these issues with one simple fix

- **The API issue API keys (Called access tokens) to your app directly**
  - If another app tries to use the key, the request is denied
  - User never handles the access token (No need to trust them)
  - Our app is accountable for the use of the access token

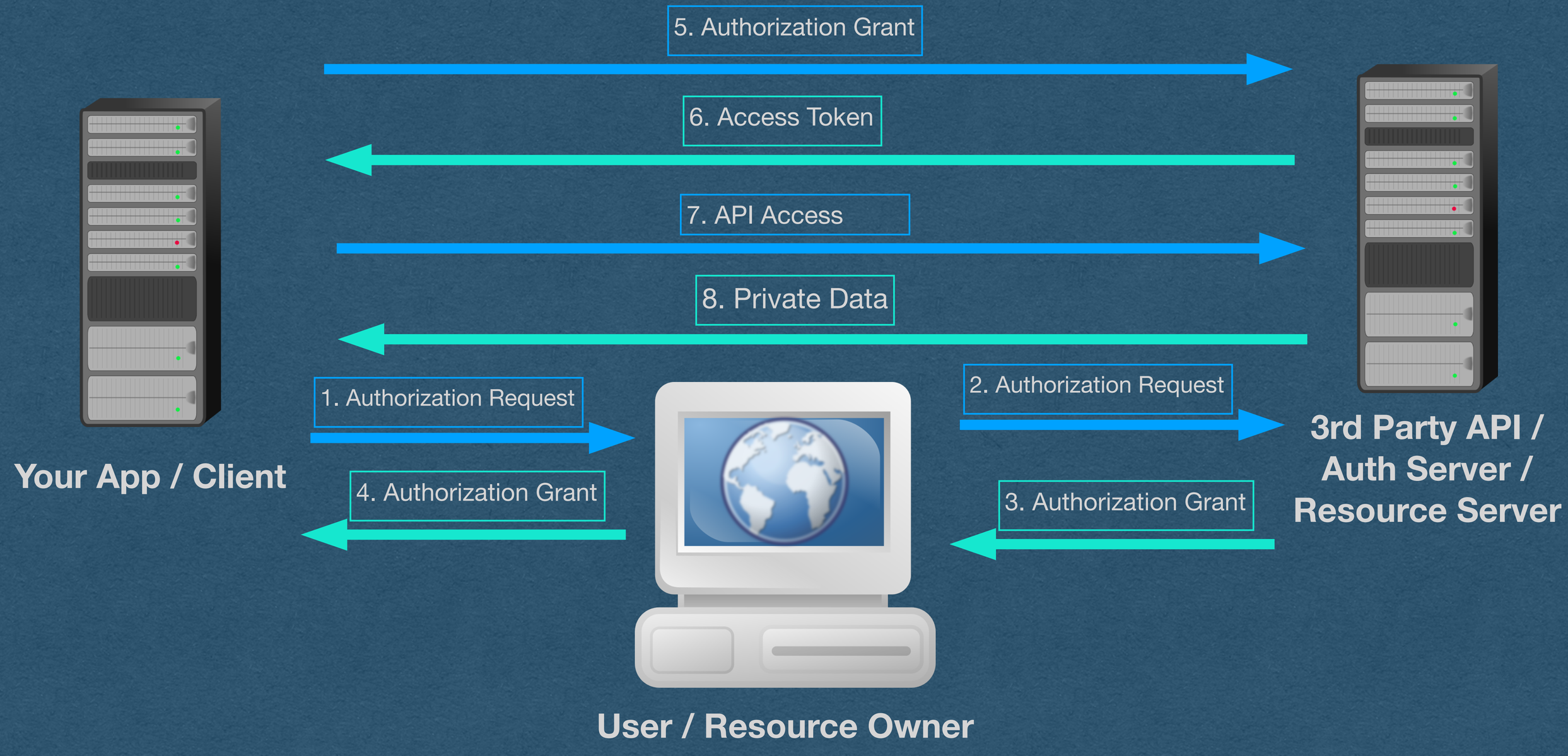
- Let's update this picture with the API issuing an access token to our client



- That's better!
- But how does it all work??



- 1: Your app asks the user to obtain an authorization grant allowing the app to use the API on their behalf



Spotify

Music Timer

You agree that Music Timer will be able to:

View your Spotify account data

Your name and username, your profile picture, how many followers you have on Spotify and your public playlists

View your activity on Spotify

Content you have recently played
The content you are playing
The content you are playing and Spotify Connect devices information

Take actions in Spotify on your behalf

Control Spotify on your devices
Create, edit, and follow playlists

You can remove this access at any time at spotify.com/account.

For more information about how Music Timer can use your personal data, please see Music Timer's privacy policy.

Logged in as Emily.
Not you?

AGREE

CANCEL

- 2. The user sends the request to the authentication server
- The user is authenticated (username/password or auth token)
- User is asked if they want to allow access
  - Contains a list of roles that the request is asking for permission to use

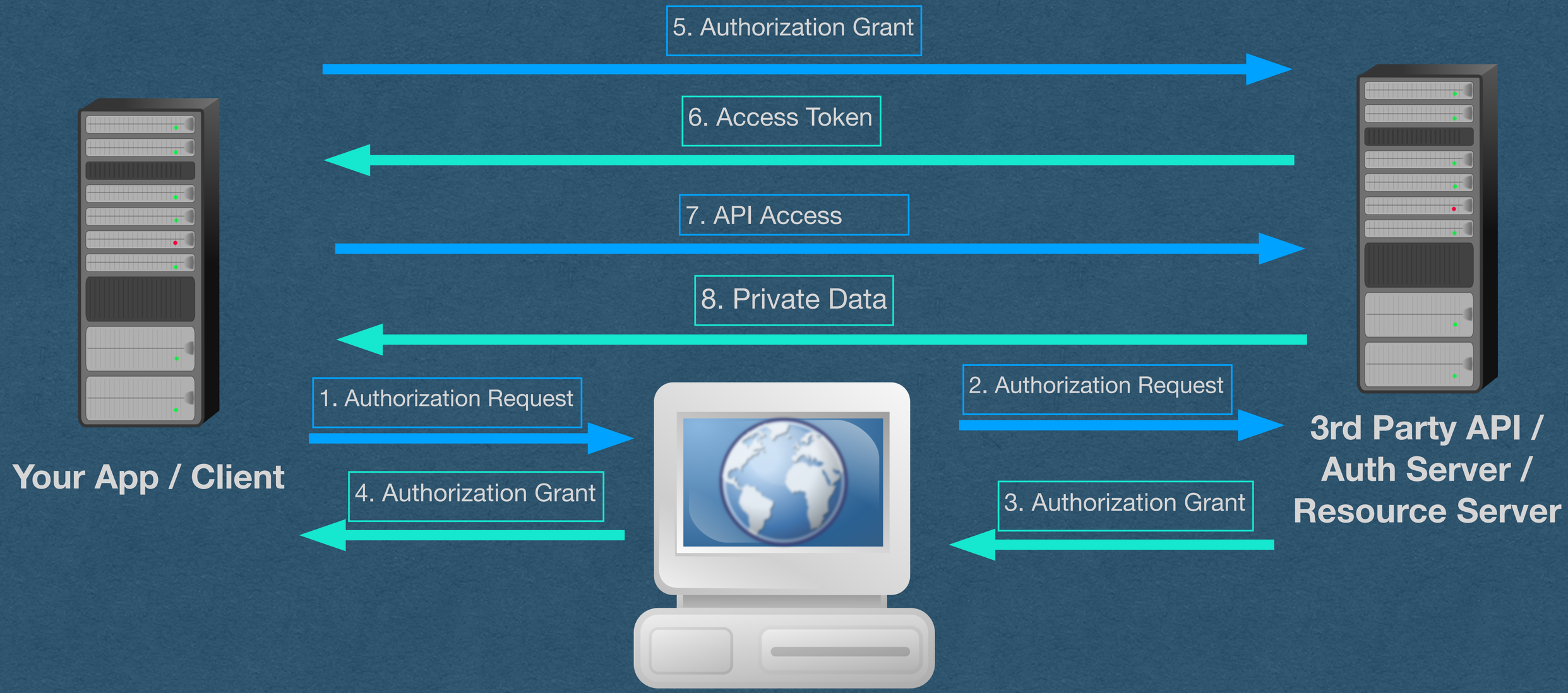
User / Resource Owner

2. Authorization Request

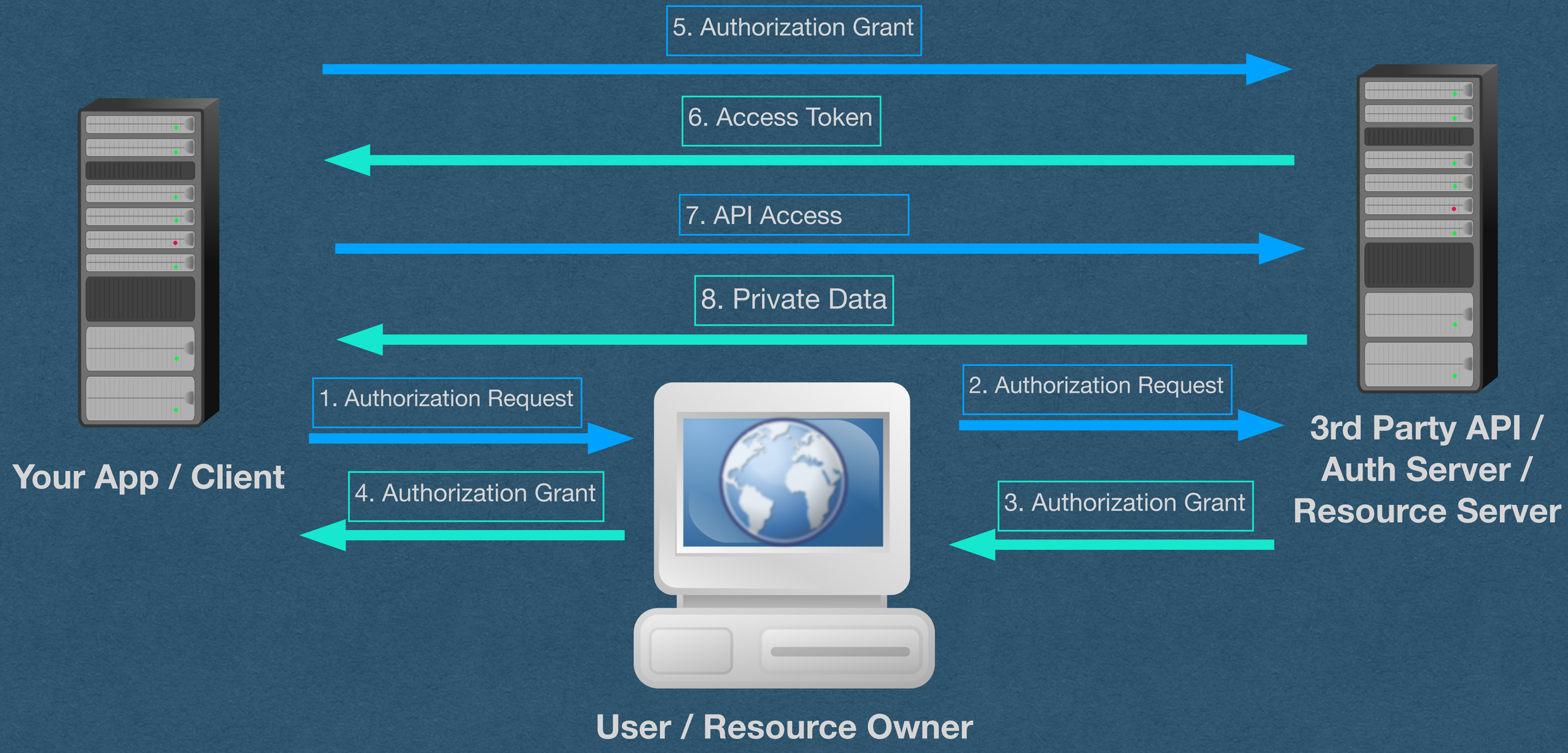
3. Authorization Grant

3rd Party API / Auth Server / Resource Server

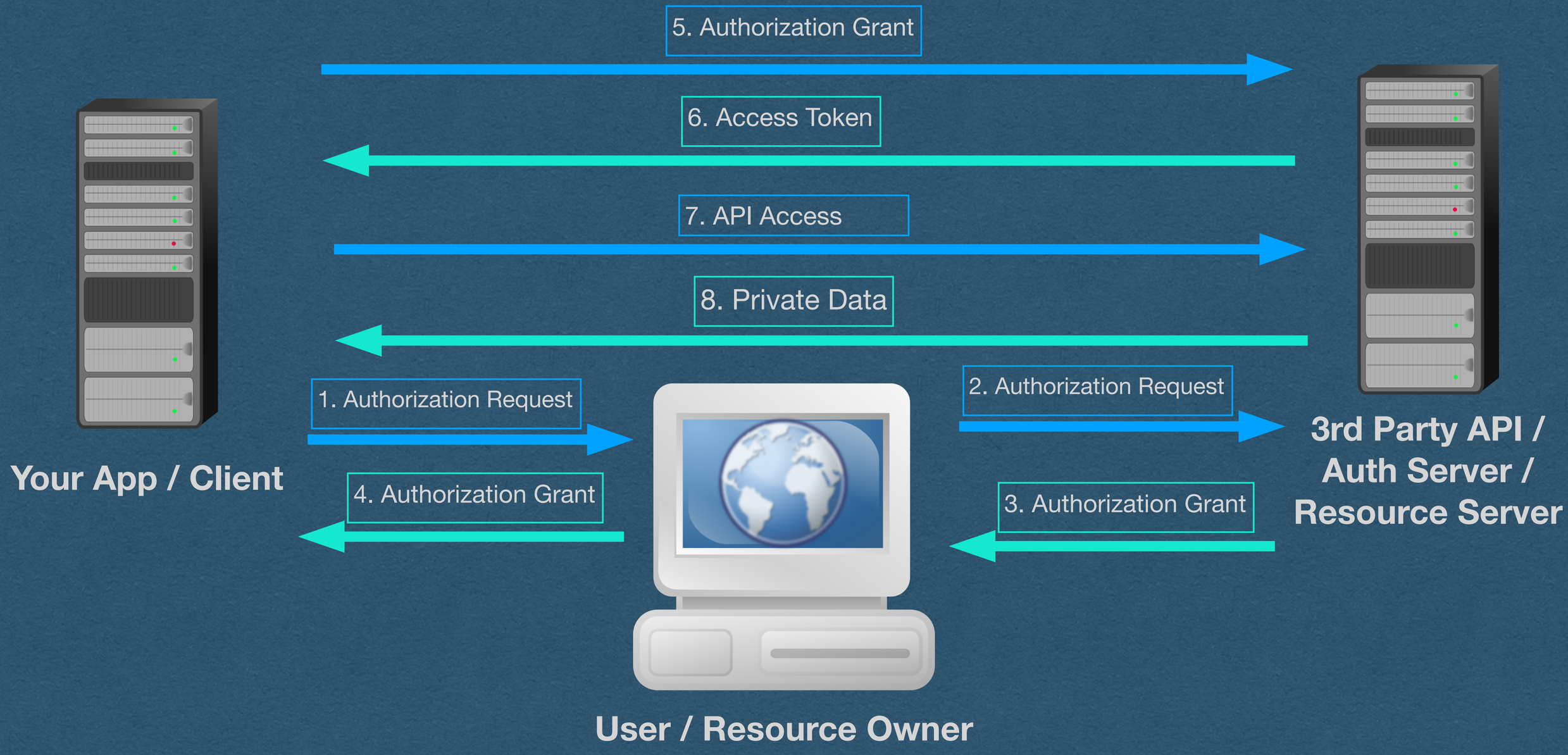
- 3: If the user is authenticated and accepts, an authorization grant is send in the url of redirect URI registered by the client



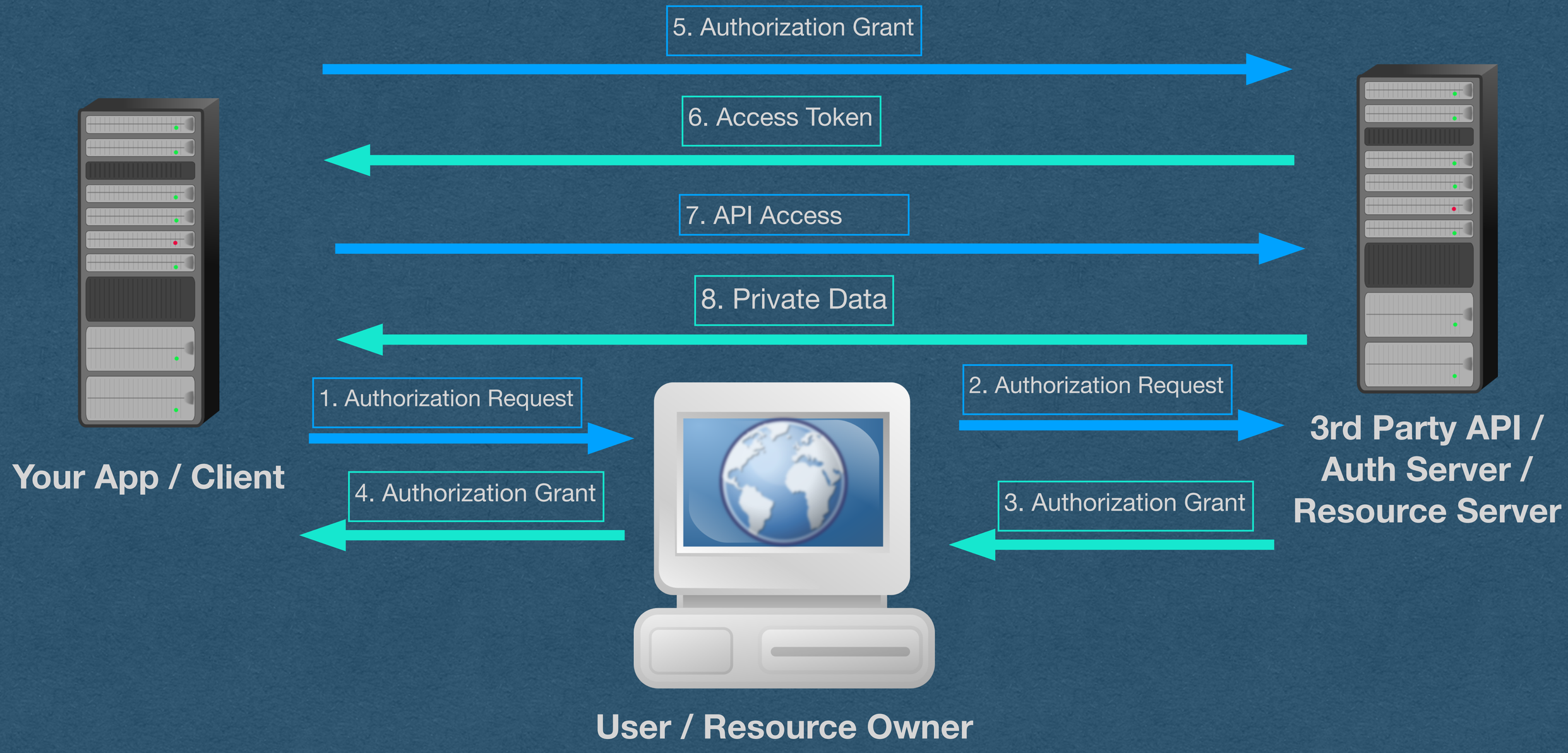
- 4: Your app receives a request from the user at the redirect URI containing the authorization grant
- Your app now has permission from the user to access the API



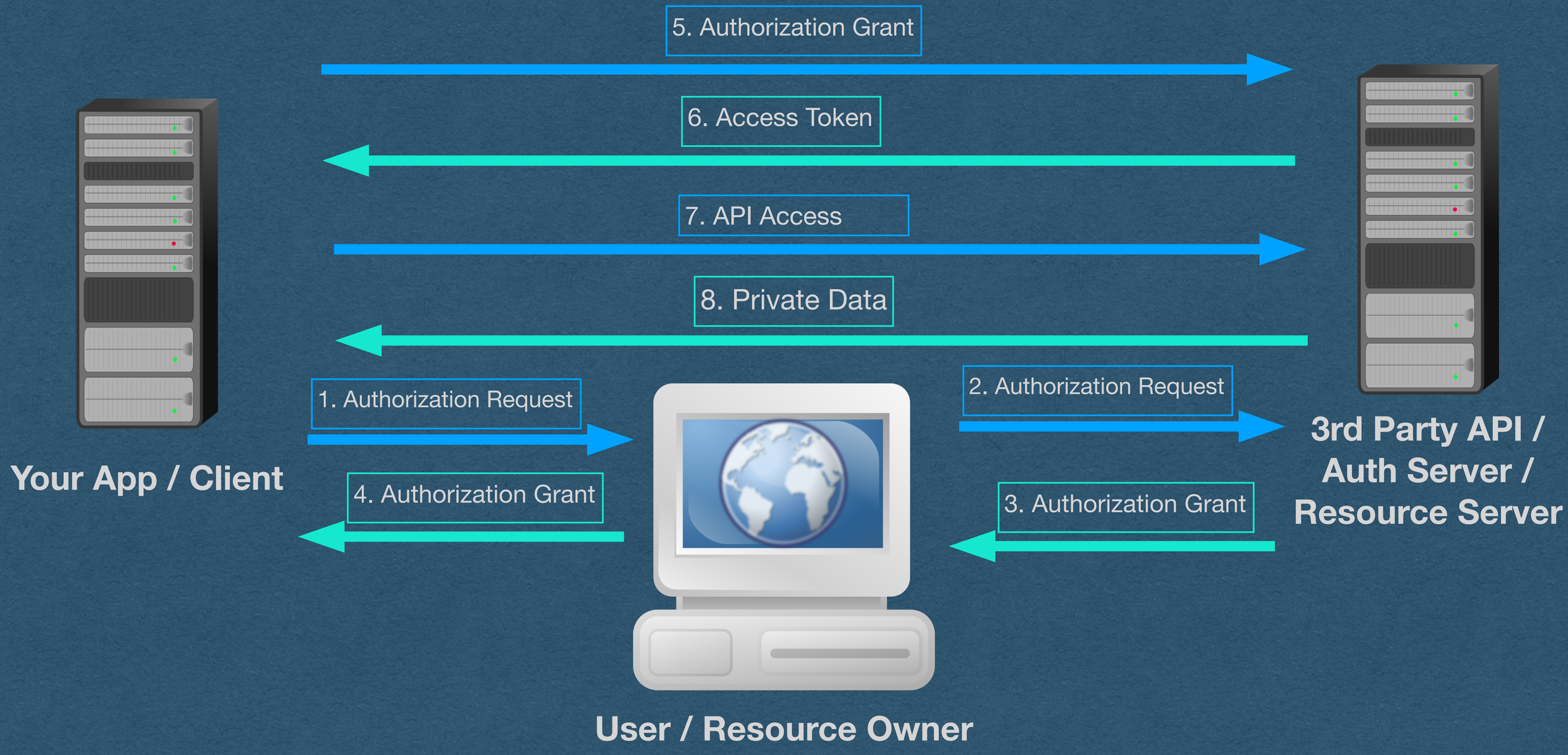
- 5: Your app will connect to the auth server and "cash in" the grant for an access token
- This step prevents the user from ever handling their access token



- 6: The auth server will verify the identity of the client using a client secret and send the access token directly to your app



- 7/8: Your app can now use the access token to access the API on behalf of your user



Client Registration

- Before any of this process can start, you must register your app with the API
- 3 Key pieces of data:
  - Client ID: A unique id assigned for your app. This is public information
  - Client Secret: Generated by the API. It *should* be kept secret if possible and can be used to authenticate your app
  - Redirect URI: Provided by you. This is where the API will send your user after generating the authorization grant