OAuth 2.0 - Overview

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 program that starts/stops Spotify playback when lecture ends/starts

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 with 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

User

register/login

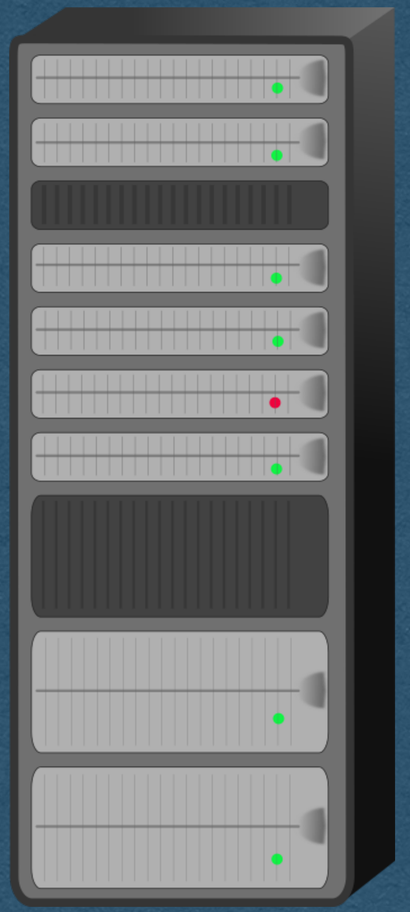
auth token

Request API key
-Auth token

API key

API endpoint
-API key

200 OK
-Private data



Server

OAuth?

- This setup works well enough

- 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 for your users
- Examples:
  - You're building a scrum board app that creates/updates 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 create/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 on your own app.
  - 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

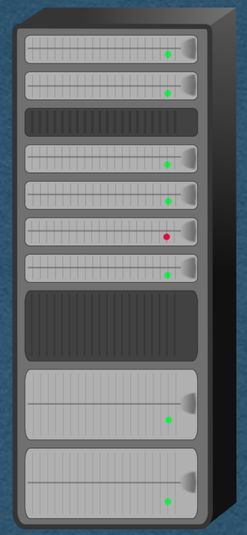
- **Never do this!!**

- ..Have the user give you their API key

- Not nearly as bad as storing their password
- Lack of accountability
  - API accessing made by our app will look like they come from the user
  - Rouge apps can abuse your key without detection
- API key rate limiting will count against the user when we use the key
  - User gets denied access if your app overuses the key
  - **Bigger problem** if the API charges $ per access

Simple Overview [BAD Attempt]:

- Our app asks the user for their API key



Your App / Client

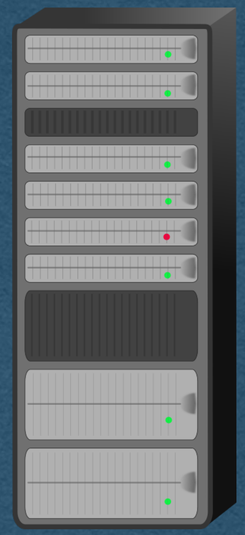
1. Request API key

User / Resource Owner

3rd Party API / Auth Server / Resource Server

Simple Overview [BAD Attempt]:

- Our app asks the user for their API key
- User visits the API and obtains an API key for their account



Your App / Client

1. Request API key

User / Resource Owner

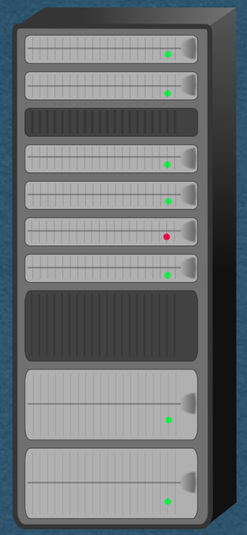
2. Request API key

3. API key

3rd Party API / Auth Server / Resource Server

Simple Overview [BAD Attempt]:

- Our app asks the user for their API key
- User visits the API and obtains an API key for their account
- User hands their key over to us



Your App / Client

1. Request API key

4. API key

User / Resource Owner

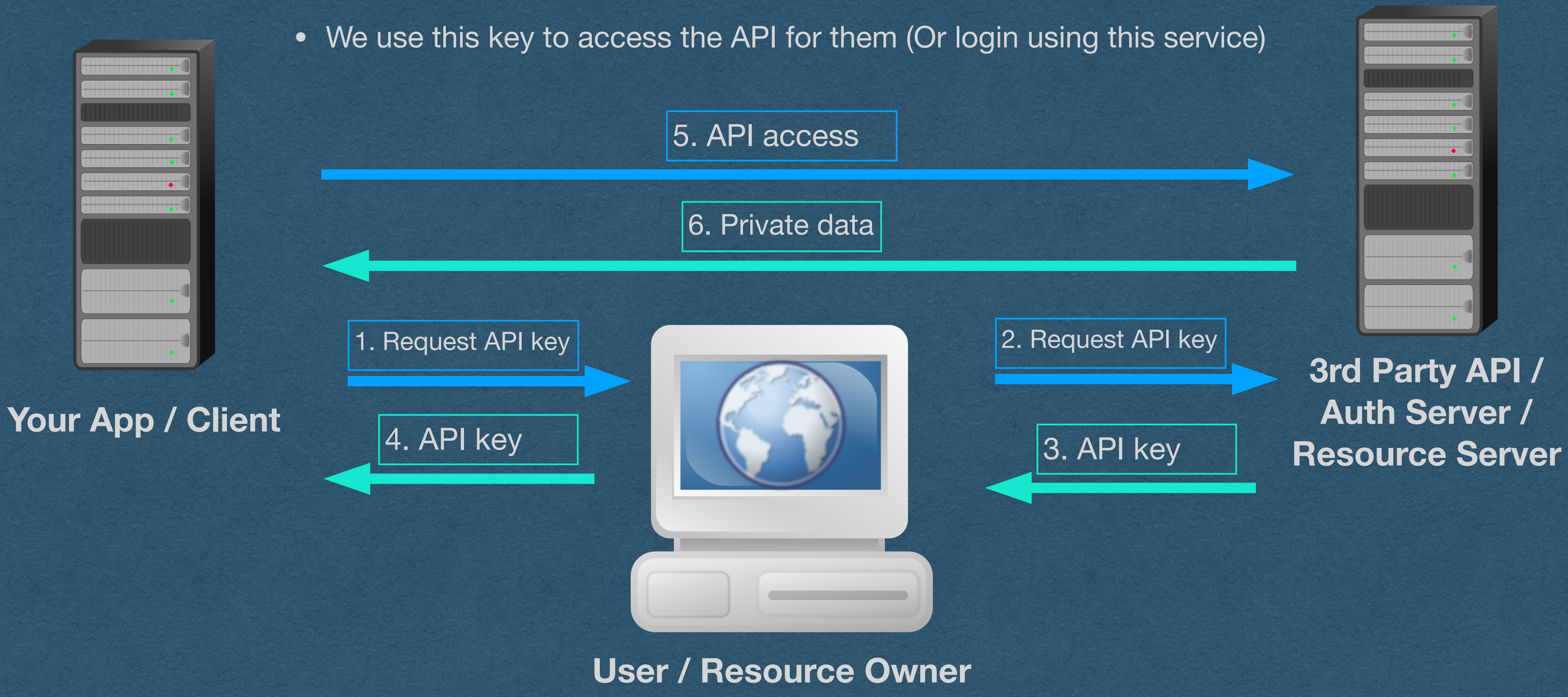
2. Request API key

3. API key

3rd Party API / Auth Server / Resource Server

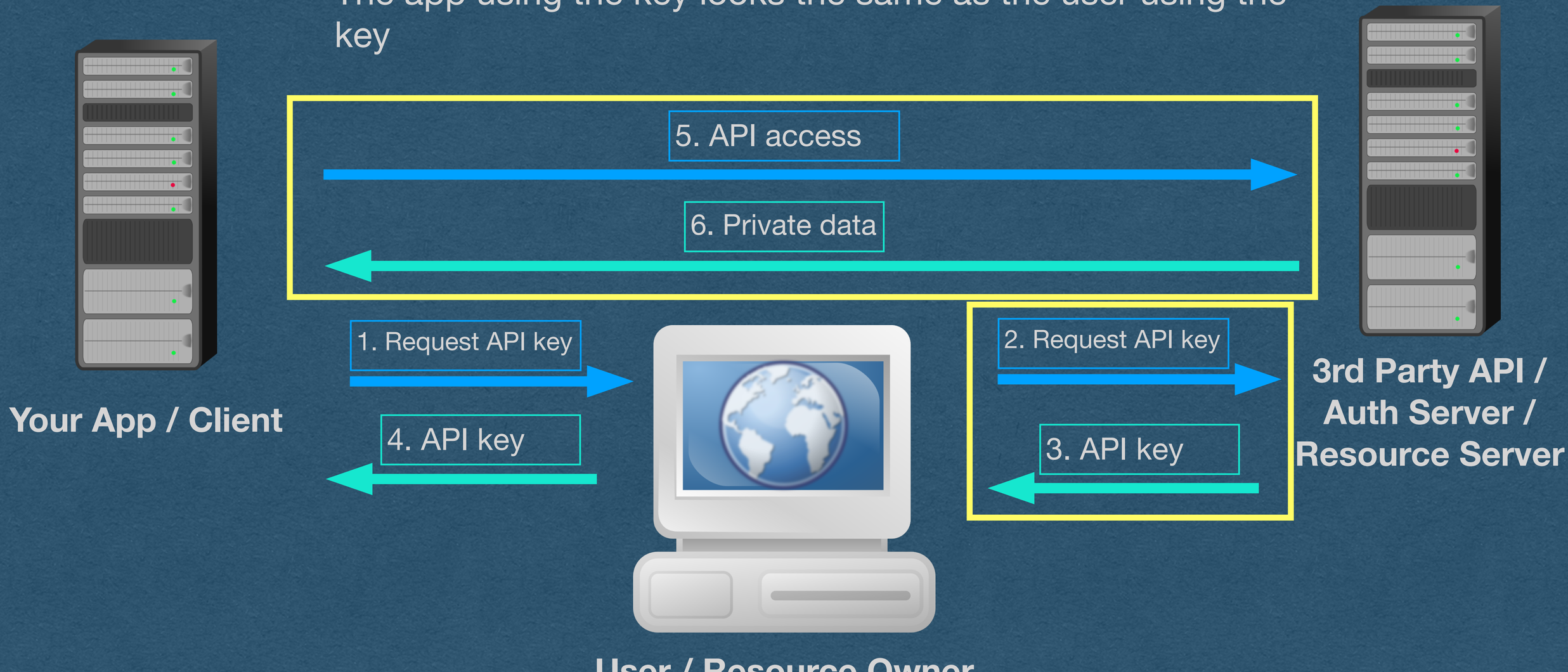
Simple Overview [BAD Attempt]:

- Our app asks the user for their API key
- User visits the API and obtains an API key for their account
- User hands their key over to us
- We use this key to access the API for them (Or login using this service)



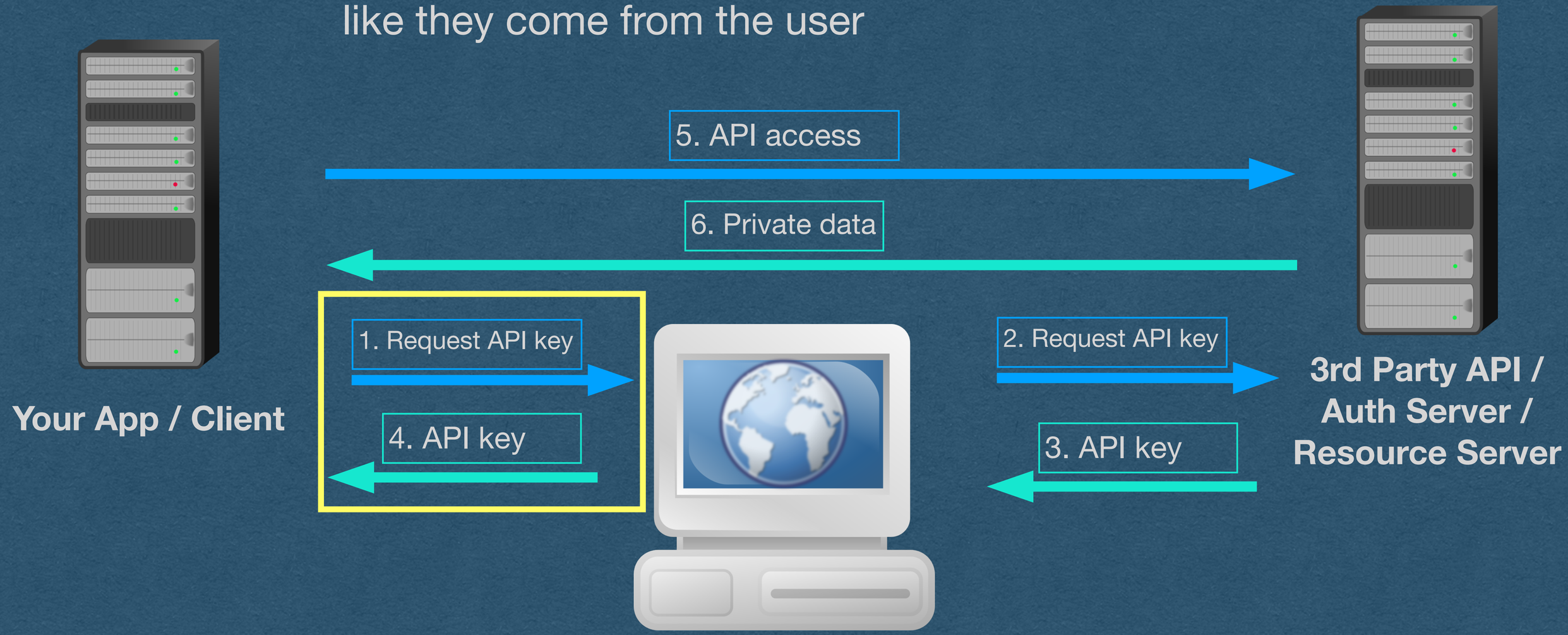
Security issue - Lack of accountability of the client

- The API has no idea that the user allowed your app to use this key
- The app using the key looks the same as the user using the key



Security issue - No compromise detection

- Your server is responsible for handling the API key
- If the key is compromised, attackers requests look like they come from the user



Security issue - Never trust your users

- The user has to handle their own API key
  - Key can be compromised at this point
- Never trust your users. Not even with their own security

OAuth 2.0

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

- Designed specifically to allow apps to use 3rd party APIs for 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

OAuth 2.0

OAuth 2.0 will fix the security issues with one simple fix

- **The API issues API keys (Called access tokens) to your app directly**
  - Our app is accountable for the use, and secure handling, of the access token
  - User never handles their access token (No need to trust them)

OAuth 2.0

- There are several ways to use OAuth 2.0 called flows

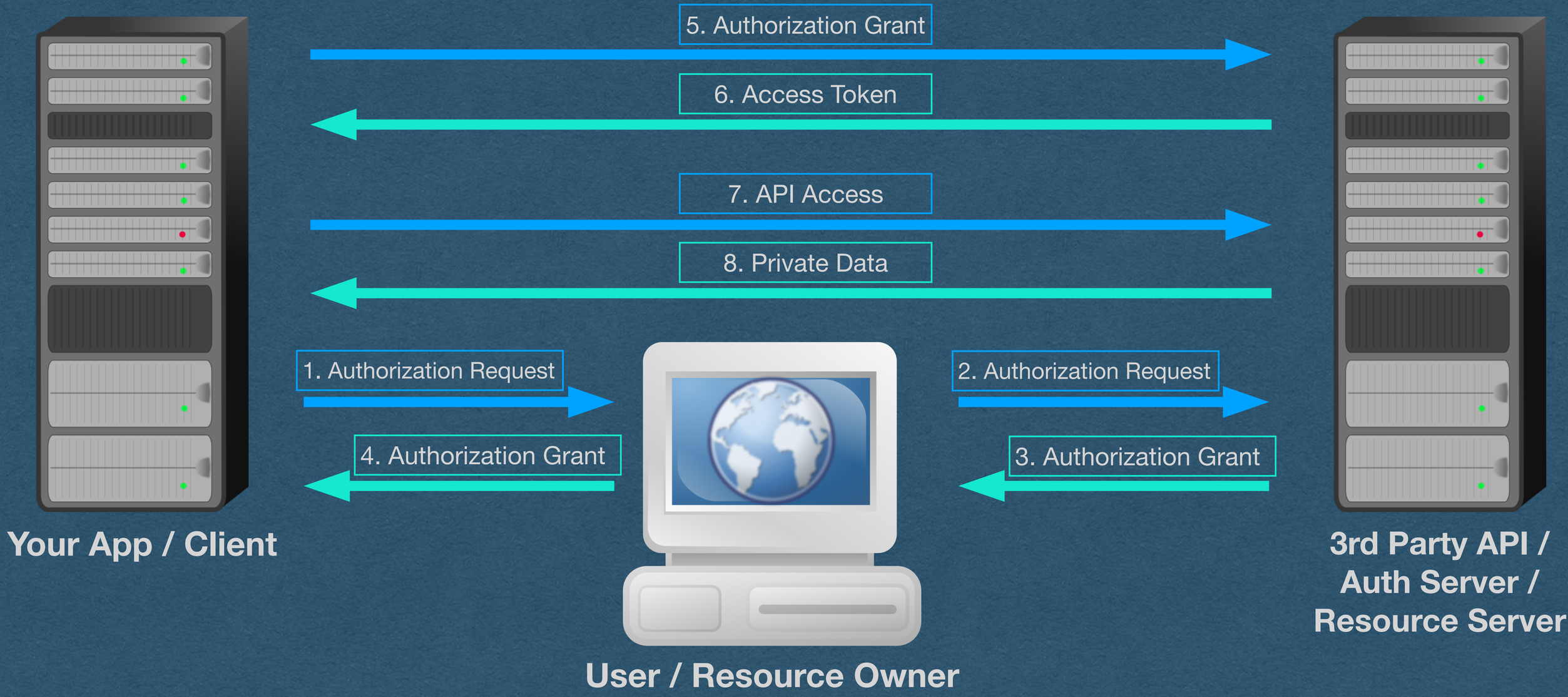
- We'll use the **authorization code flow**
  - The proper flow for our use case of a private server
  - We can store a secret on our server that the user can never access

- If, for example, we build a stand-alone app with no server:
  - The app must be self-contained (There is no back end)
  - User has access to every part of the app, including any secret information
  - Use the **implicit grant flow** (This is not allowed on the HW)

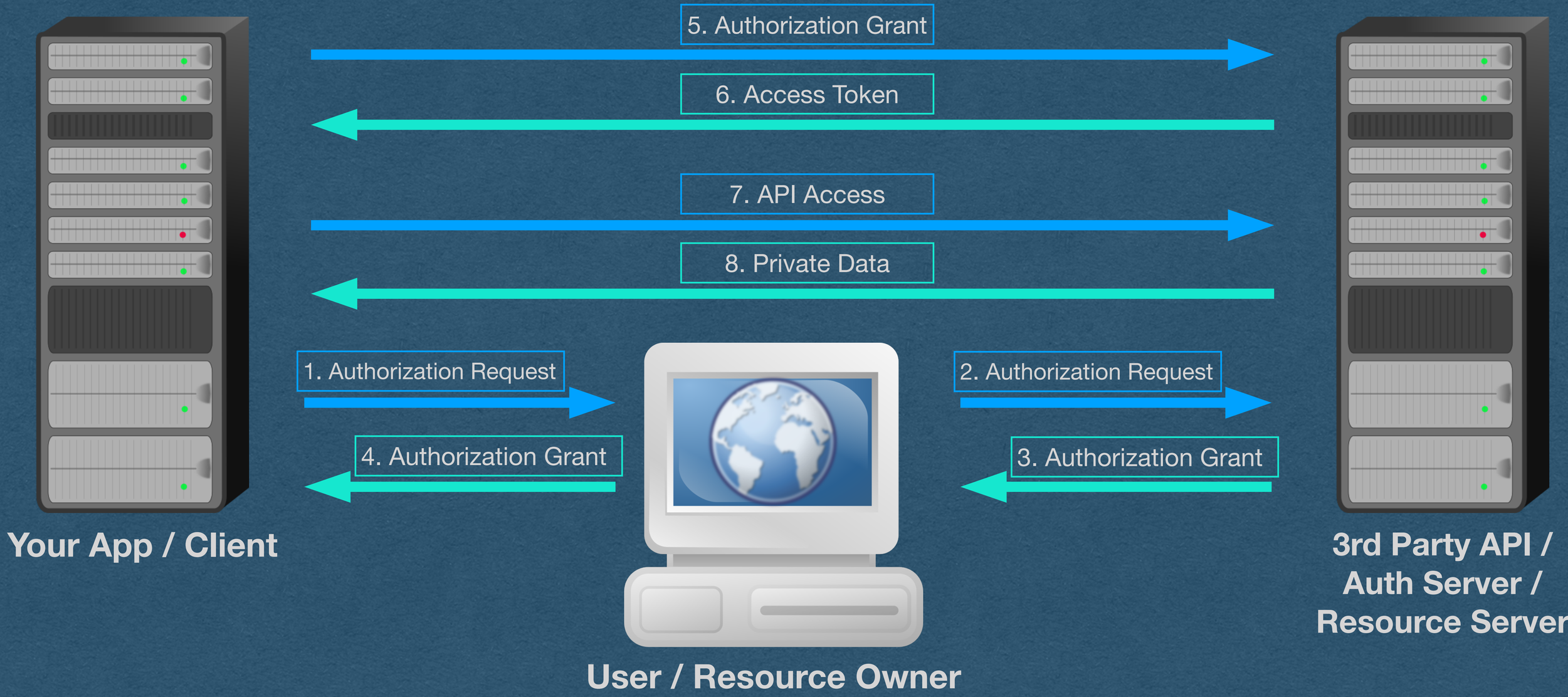
OAuth 2.0 - Client Registration

- Before starting the authorization process with your users, you must register your app with the 3rd party API
- During the registration process, there are 3 key pieces of data:
  - **Client ID**: A unique id generated by the API and assigned to your app. This is public information
  - **Client Secret**: A high-entropy random value generated by the API. This is effectively a password that will be used to authenticate your app. [If this value cannot be kept secret, use a flow that doesn't involve a secret]
  - **Redirect URI**: Provided by you. This is where the API will send your user after they grant you access to the API

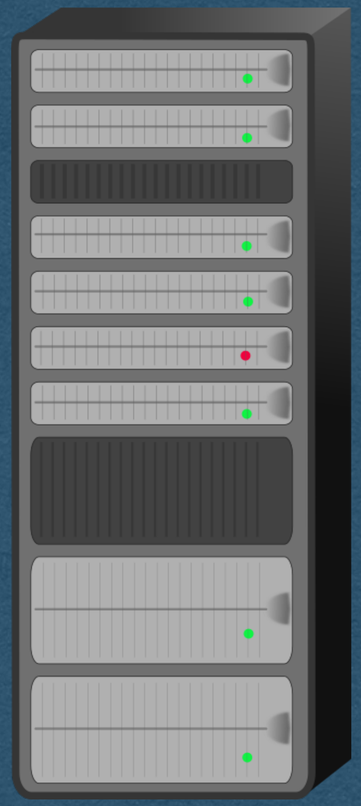
- We'll update our picture with the full OAuth authentication code flow



- Let's break this down



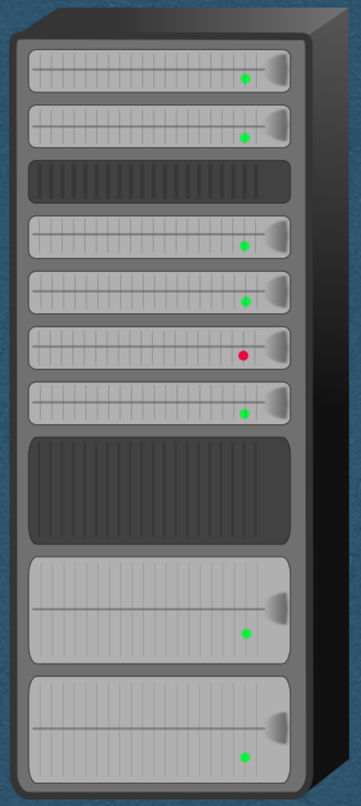
- 1: Your app asks the user to obtain an authorization grant allowing the app to use the API on their behalf



Your App / Client

1. Authorization Request

User / Resource Owner



3rd Party API / Auth Server / Resource Server

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?

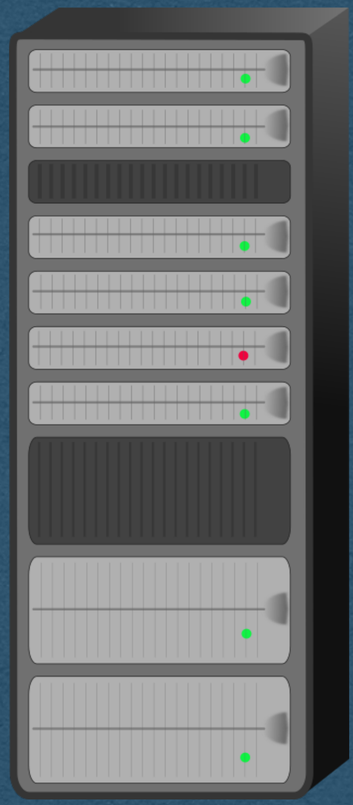
AGREE

CANCEL

- 2. The user sends the request to the authentication server
- The user is authenticated by the API (username/password or auth token)
- User is asked if they want to allow access
  - Contains a list of scopes requested by our app

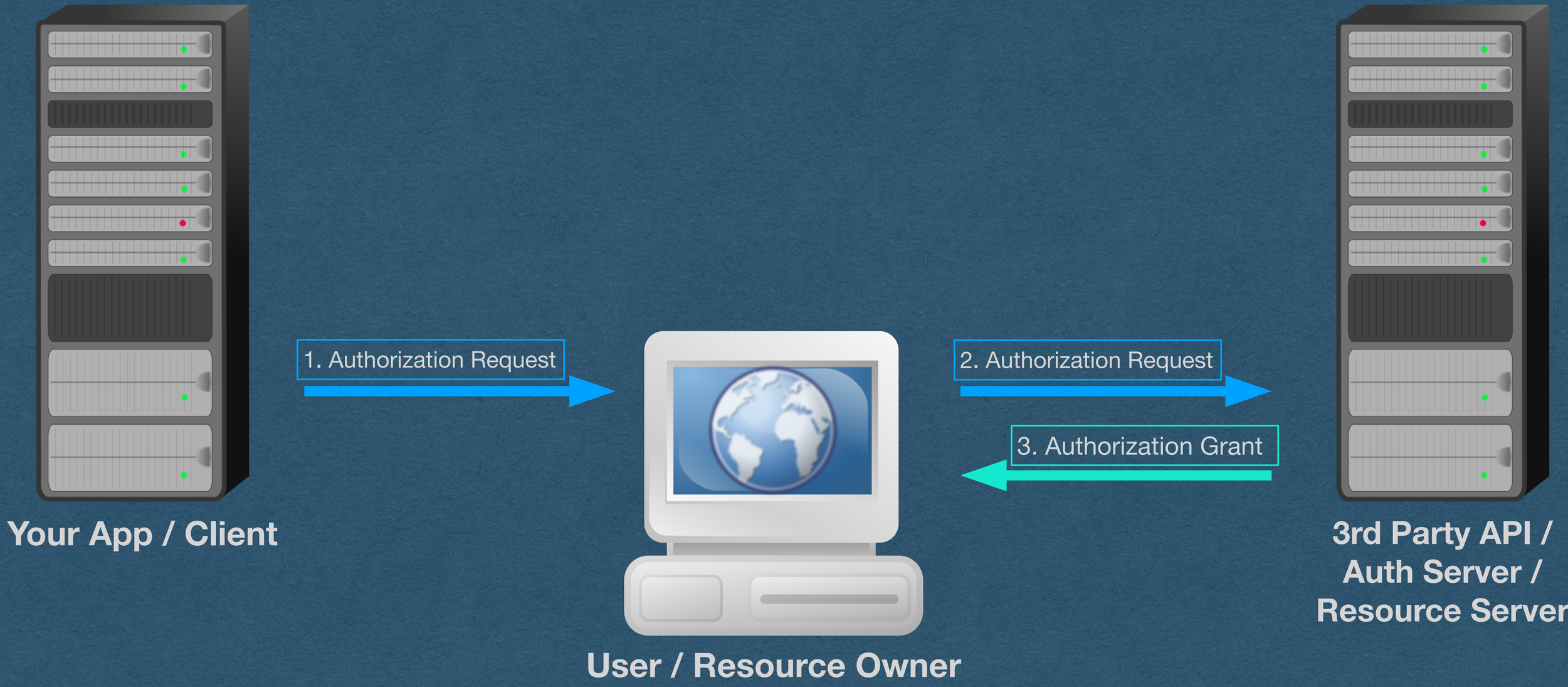
User / Resource Owner

2. Authorization Request

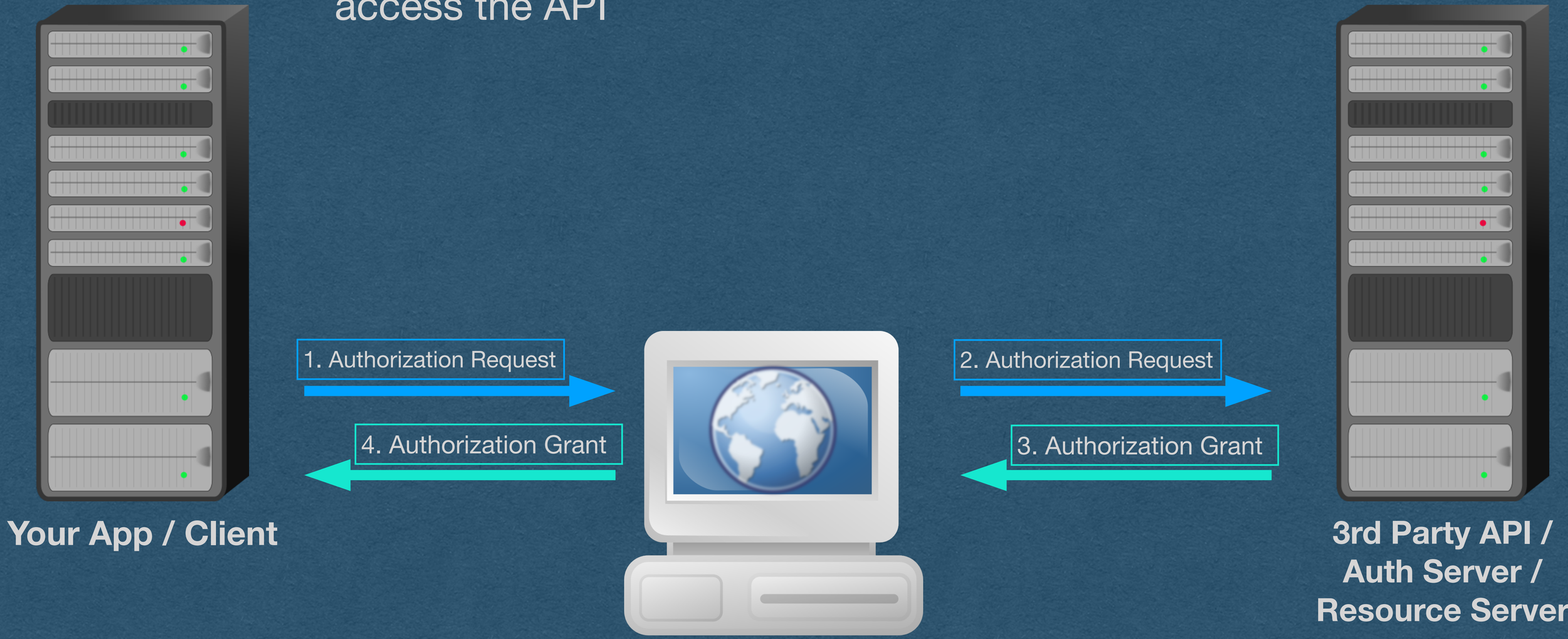


3rd Party API / Auth Server / Resource Server

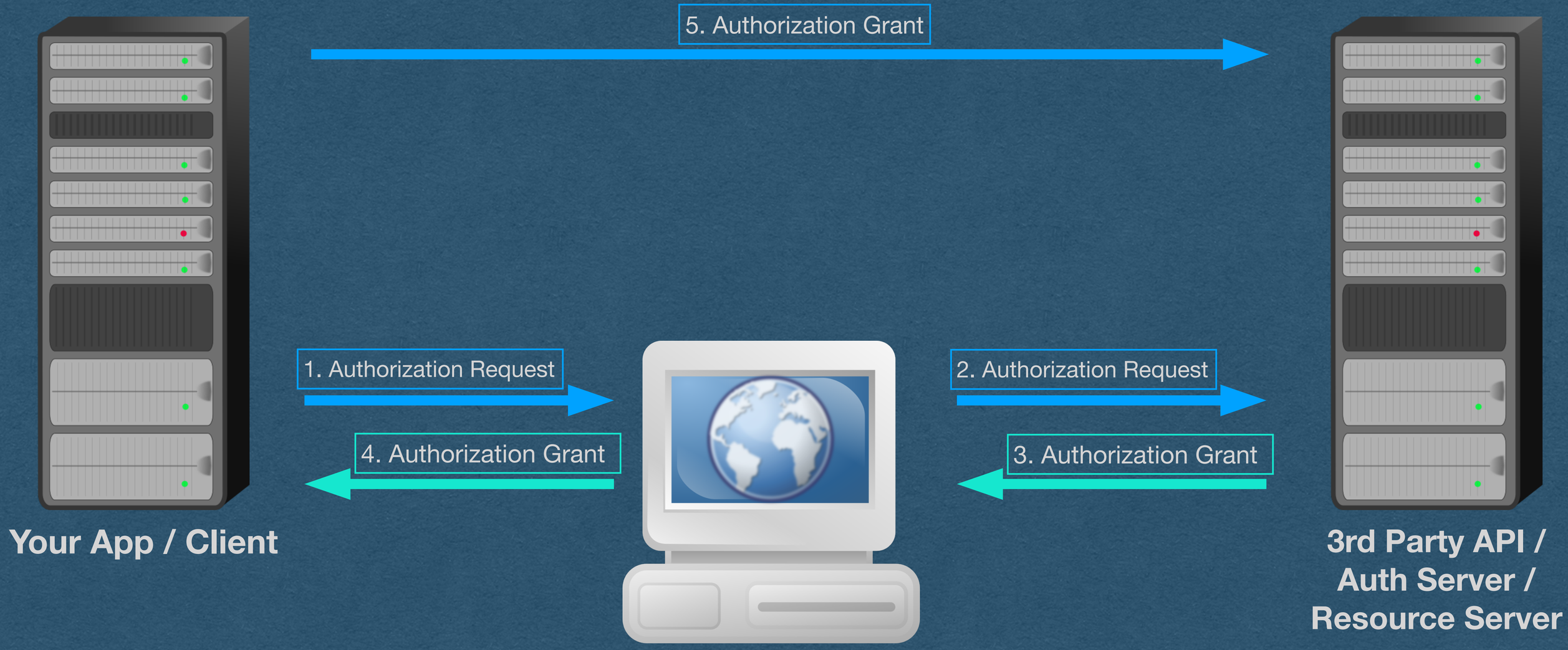
- 3: If the user is authenticated and accepts, an authorization grant is sent to the user
- The grant contains an authorization code



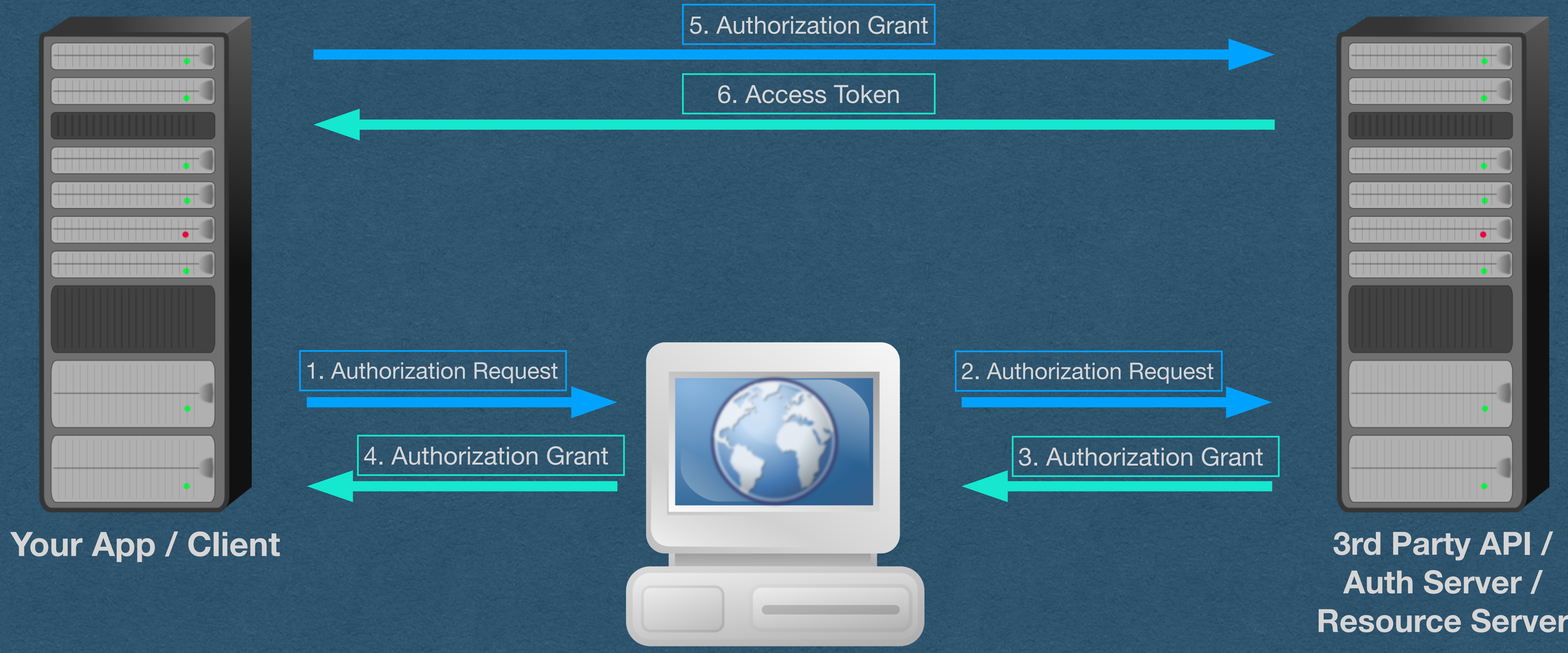
- 4: Your app receives a request from the user at the redirect URI containing the authorization grant
- Your app now has permission from the user to access the API



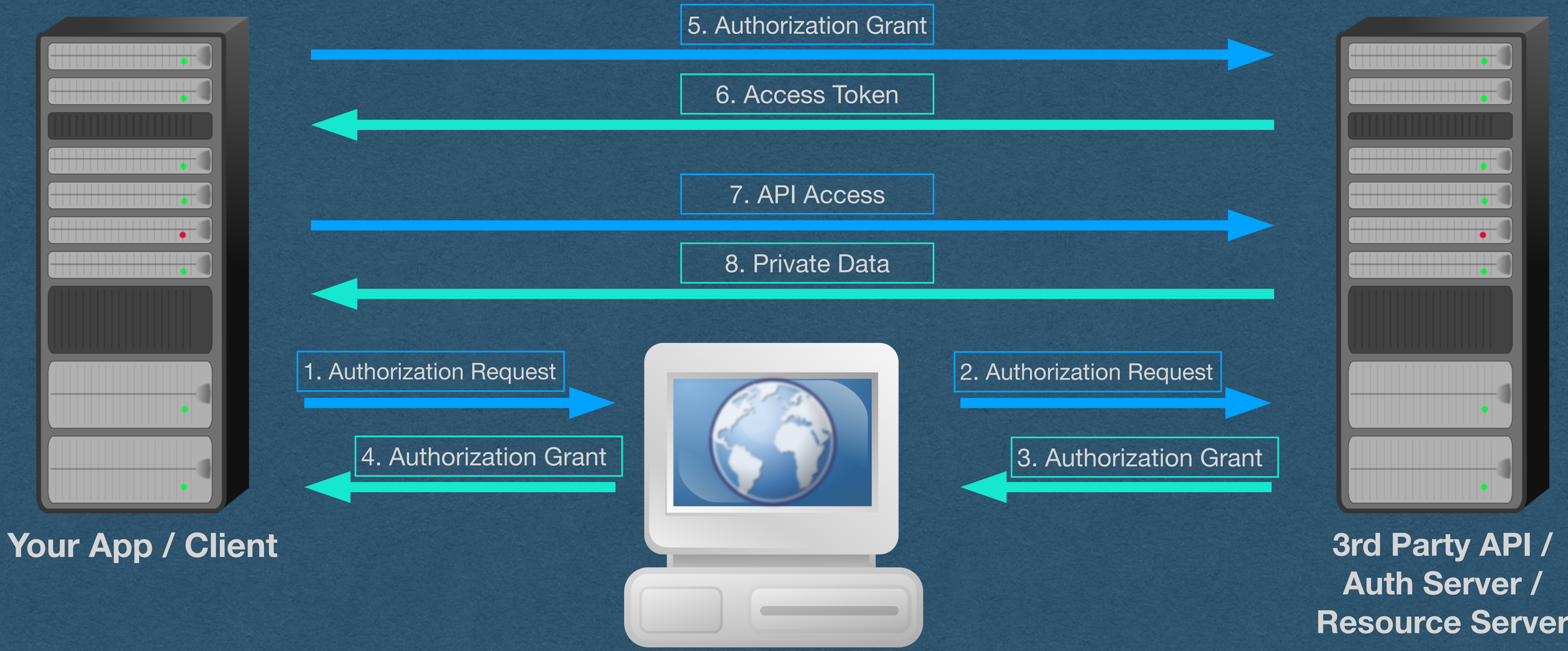
- 5: Your app will connect to the auth server and "cash in" the grant for an access token
- This step prevents the user from ever handling their access token



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



- 7/8: Your app can now use the access token to access the API for your user
- If the user clicked "login with 3rd party", verify their identity through the API
  - Create an account for them based on this identity



OAuth 2.0 - Authorization Code Flow