Software Security
Lecture 5 - OAuth

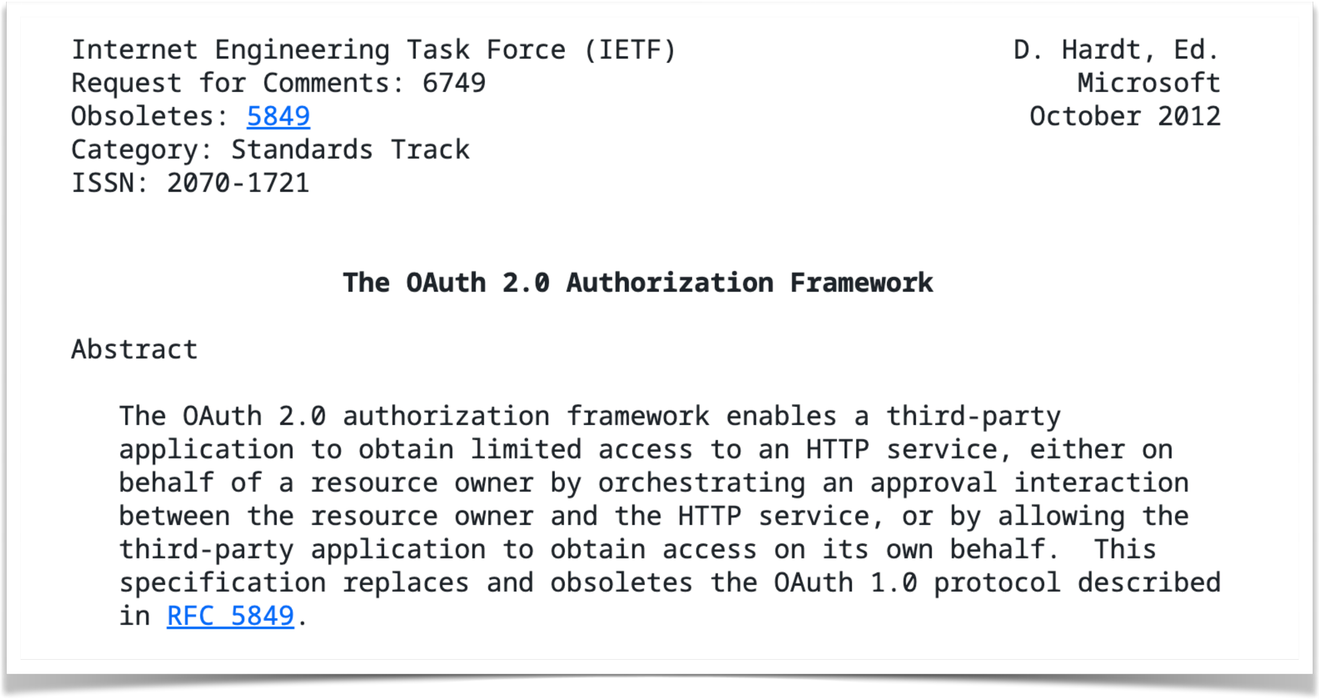
MIEI - Integrated Master in Computer Science and Informatics
MEI - Master in Computer Science and Engineering
Specialisation Course

Carla Ferreira & João Costa Seco

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

OAuth 2.0

- OAuth 2.0 is the industry-standard protocol for authorisation.
- OAuth 2.0 provides simple authorisation flows for web applications, desktop applications, mobile phones, and domotic devices.
- OAuth 2.0 is being developed by the IETF OAuth Working Group.



Internet Engineering Task Force (IETF) D. Hardt, Ed.
Request for Comments: 6749 Microsoft
Obsoletes: 5849 October 2012
Category: Standards Track
ISSN: 2070-1721

The OAuth 2.0 Authorization Framework

Abstract

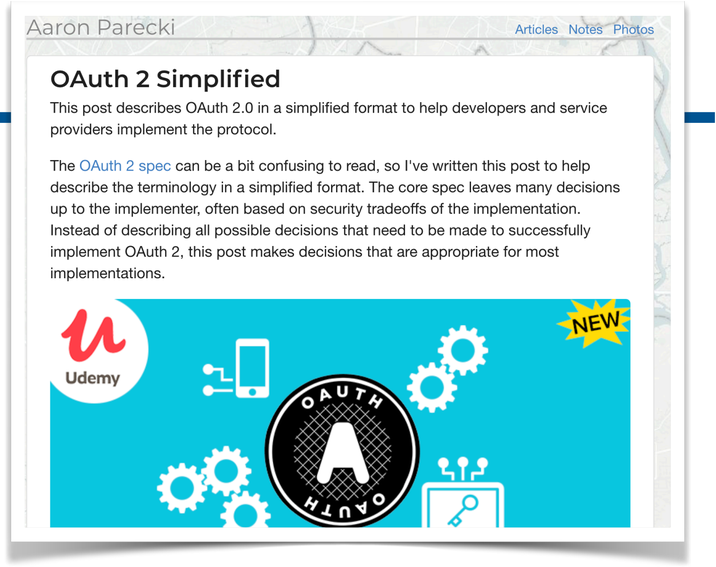
The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in RFC 5849.

OAuth 2.0 Roles

- **resource owner:** An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.
- **resource server:** The server hosting the protected resources is capable of accepting and responding to protected resource requests using access tokens.
- **client:** An application making protected resource requests on behalf of the resource owner and with its authorisation. It needs permission from the resource owner (user) before it can be done.
- **Authorisation server:** The server that issues access tokens to the client after successfully authenticating the resource owner and obtaining authorisation. This service interacts with the user to approve or reject the request. In small-scale applications, it may be co-located with the resource server, but in large-scale applications, it is often a separate component.

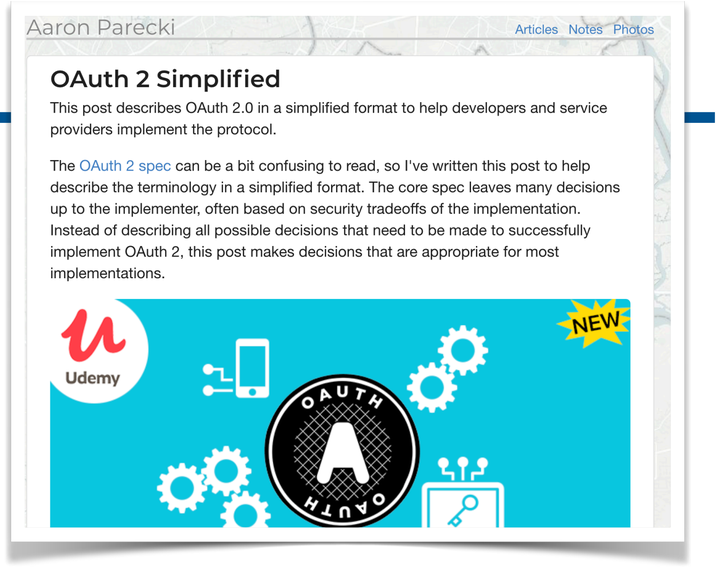
https://tools.ietf.org/html/rfc6749

OAuth 2.0



- The purpose is to provide access to resources (APIs). It is meant for the resource server to give access to the token's bearer.
- It contains a minimal amount of information to authorise access. It can be opaque or follow the JWT rules.
- OAuth 2.0 is an open specification that supports the implementation of different components.

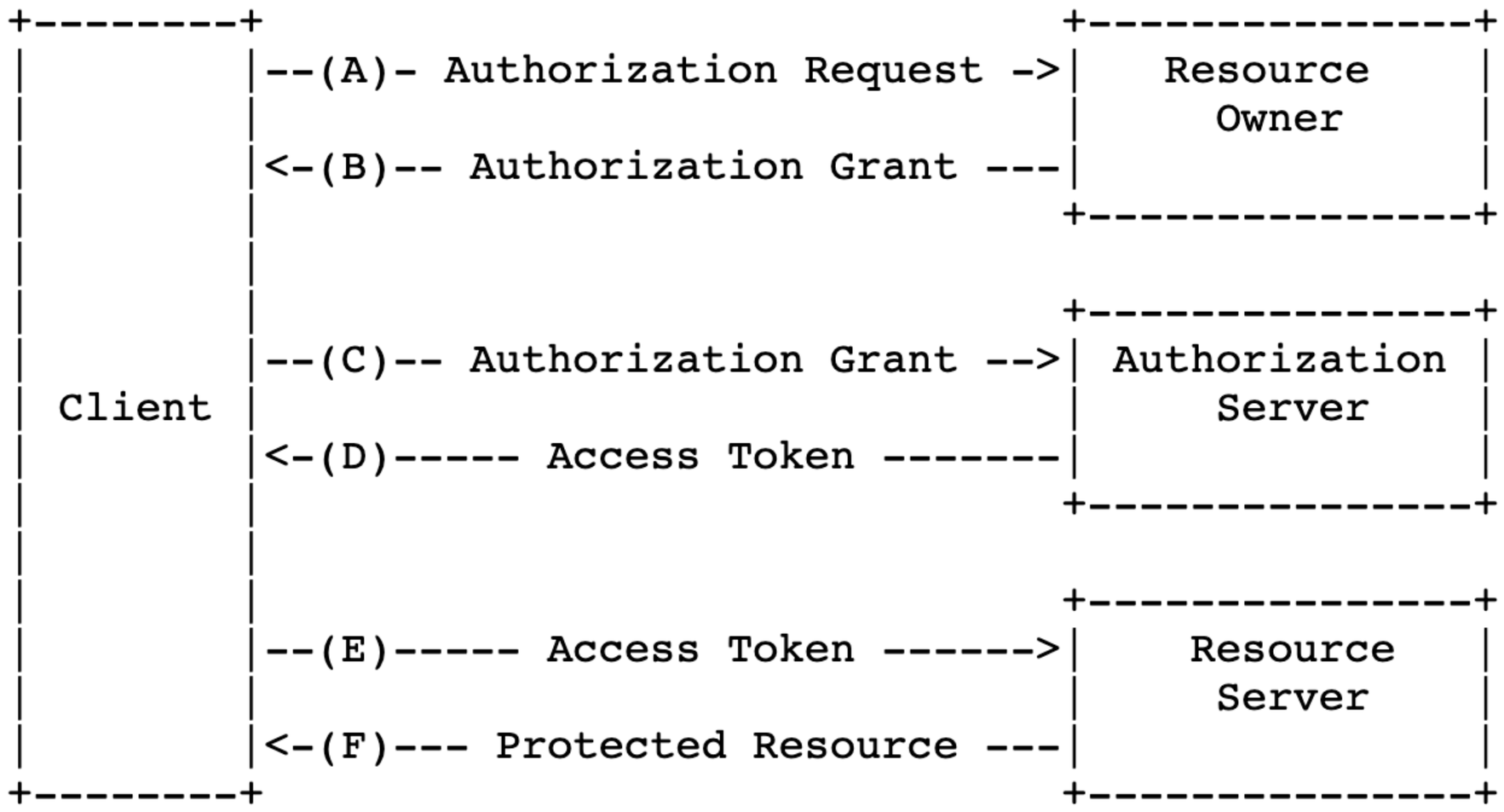
OAuth 2.0 + OpenID Connect



- The purpose is to provide the identity of the authenticated user. The goal is the client application to know who is authenticated.
- A JWT containing detailed claims about the authenticated user and meta-data about the authentication event.
- Used to support secure authentication (token-based) and single sign-on.

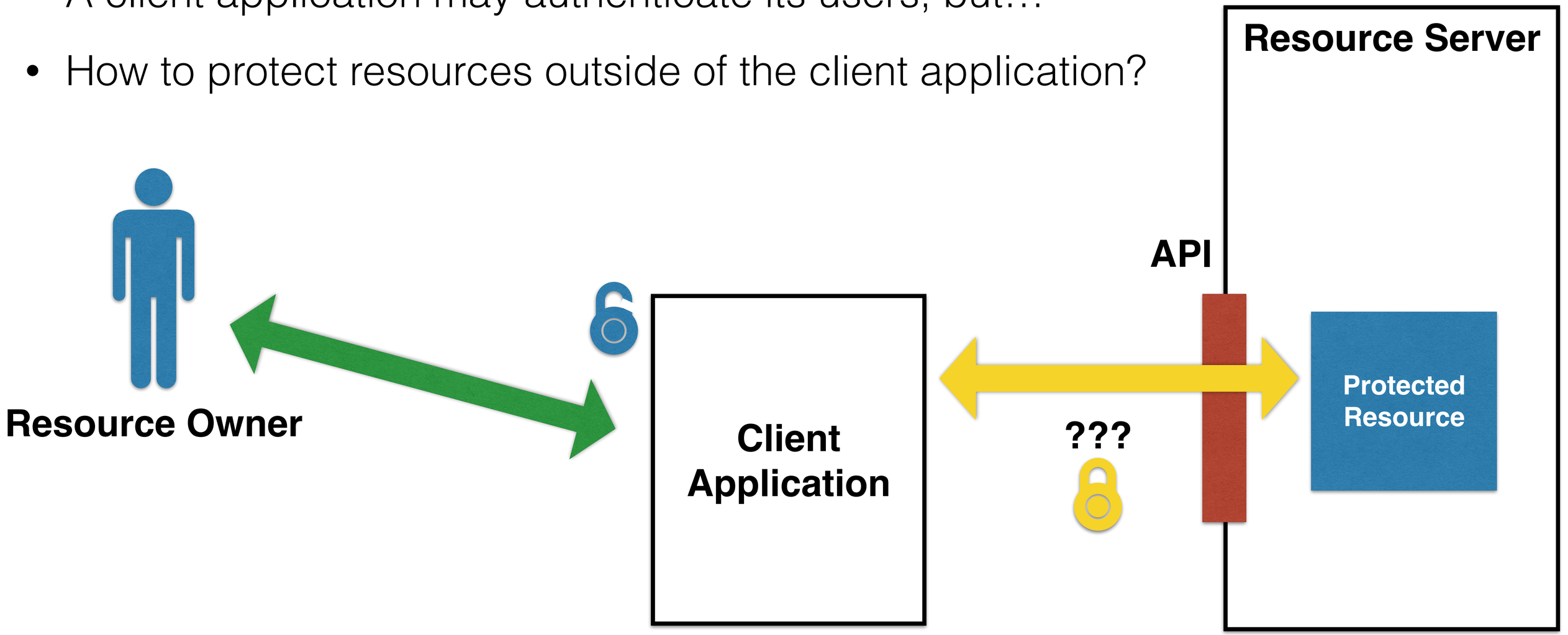
Auth 2.0 protocol flow

Auth 2.0 protocol flow



OAuth 2.0

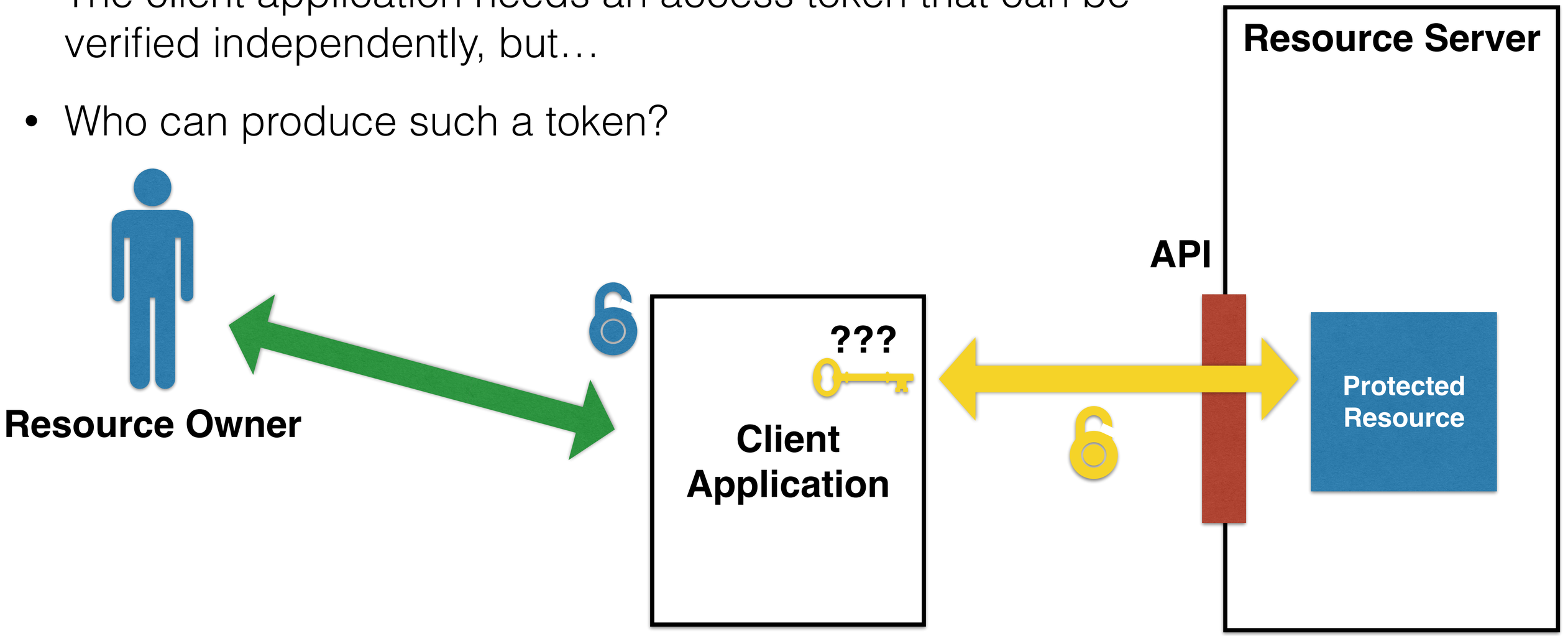
- A client application may authenticate its users, but…
- How to protect resources outside of the client application?



https://darutk.medium.com/the-simplest-guide-to-oauth-2-0-8c71bd9a15bb

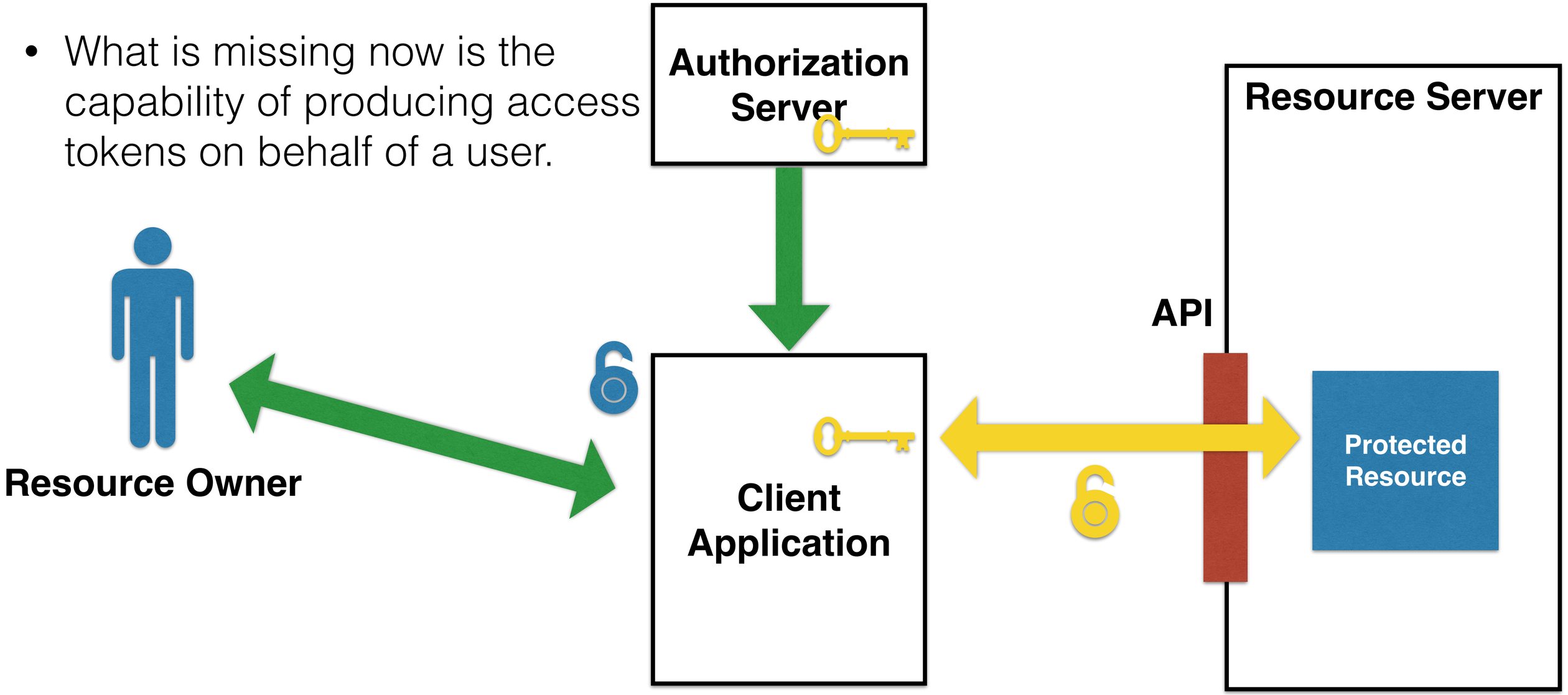
OAuth 2.0

- The client application needs an access token that can be verified independently, but…
- Who can produce such a token?



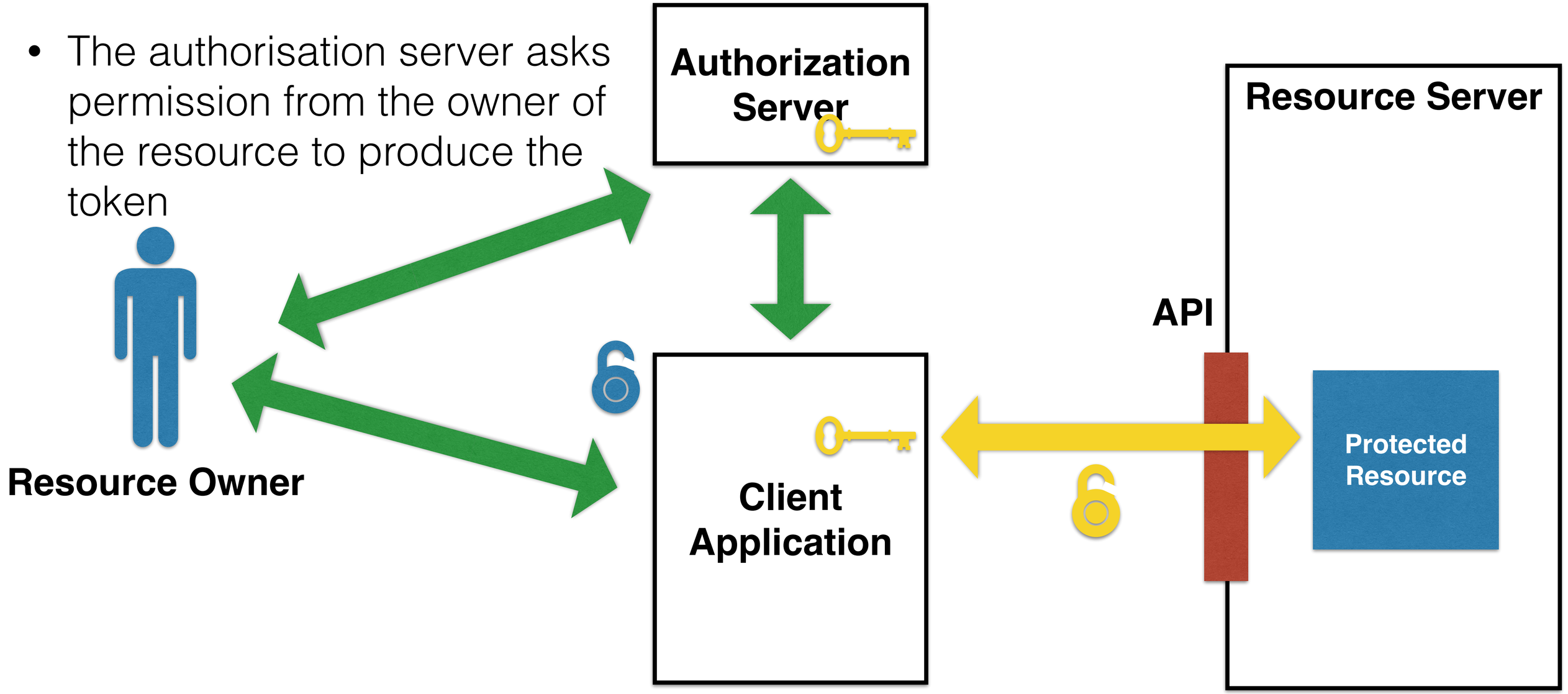
OAuth 2.0

- What is missing now is the capability of producing access tokens on behalf of a user.



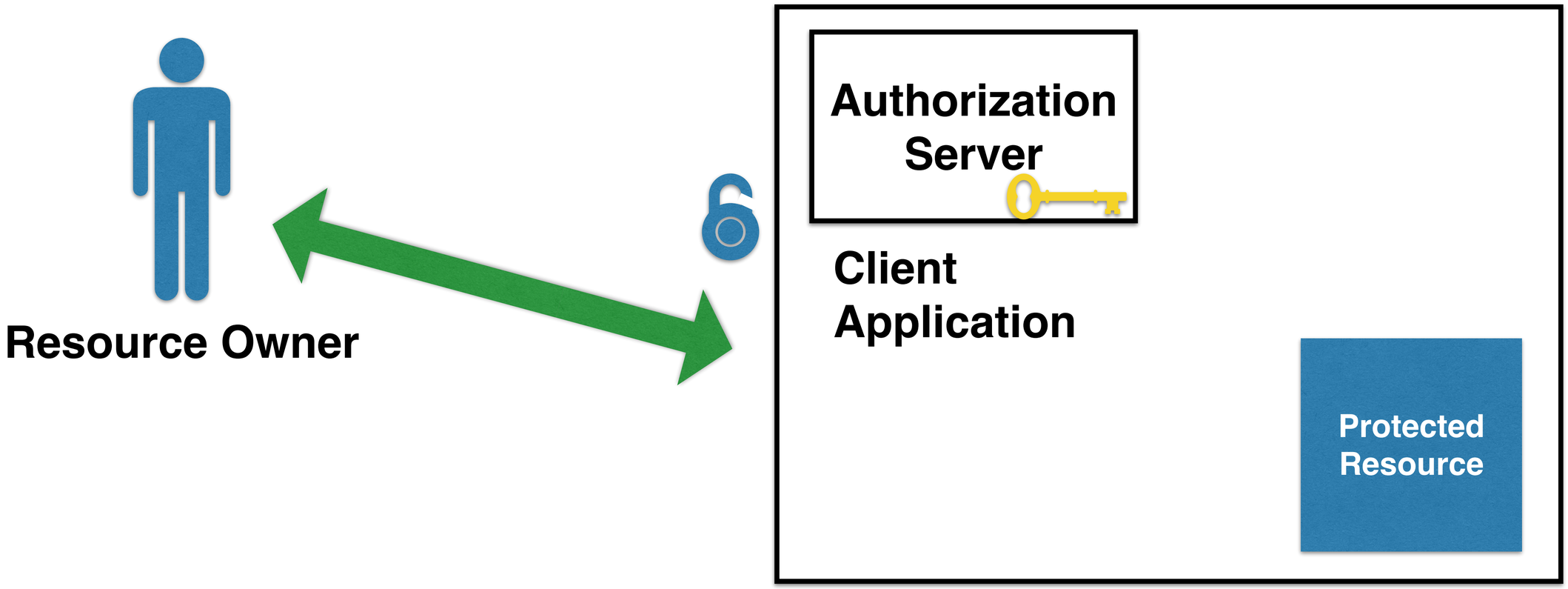
OAuth 2.0

- The authorisation server asks permission from the owner of the resource to produce the token



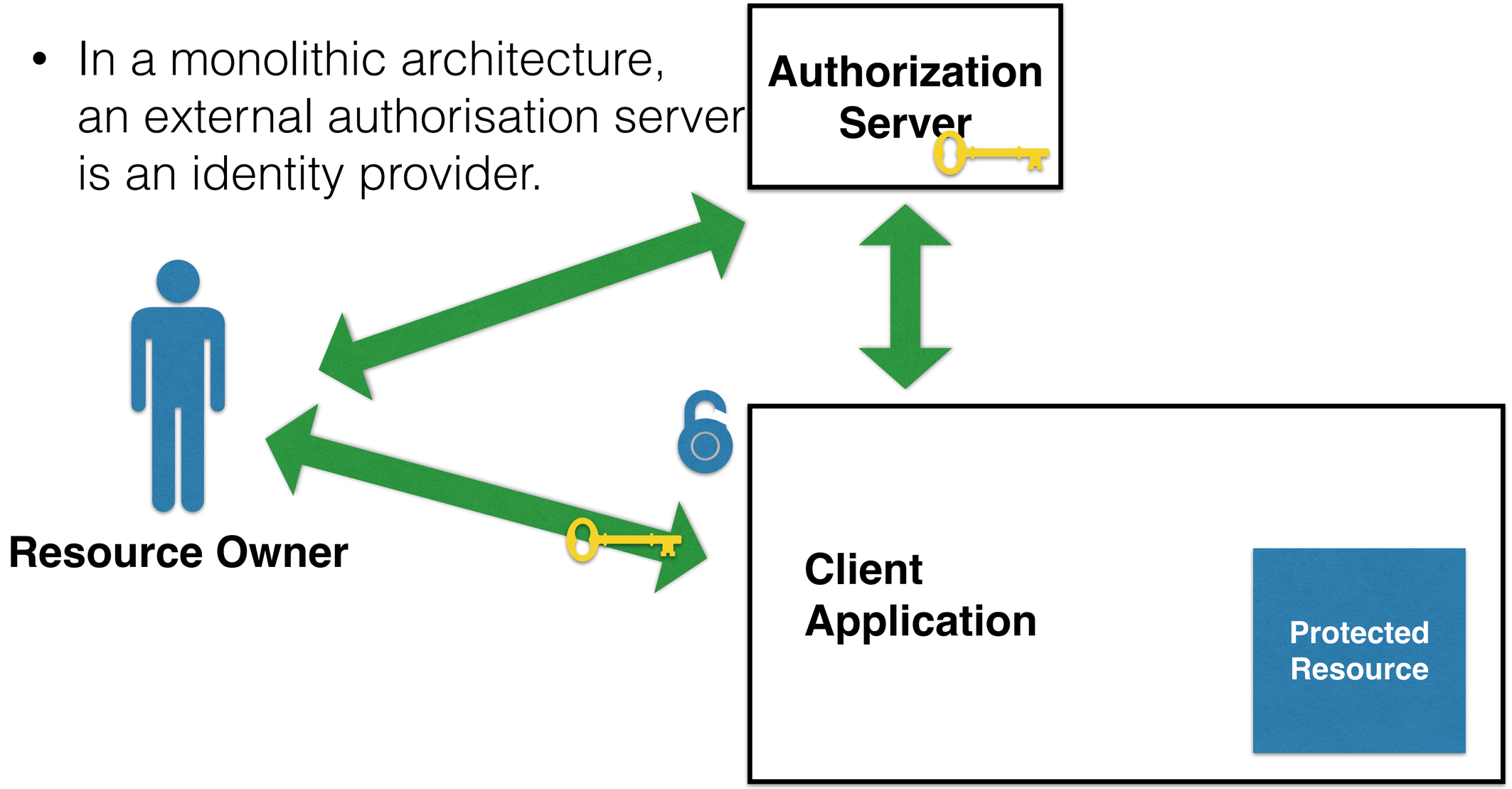
OAuth 2.0 (small-scale monolithic)

- In a small-scale monolithic architecture, the authorisation server and the protected resources are all in one place.



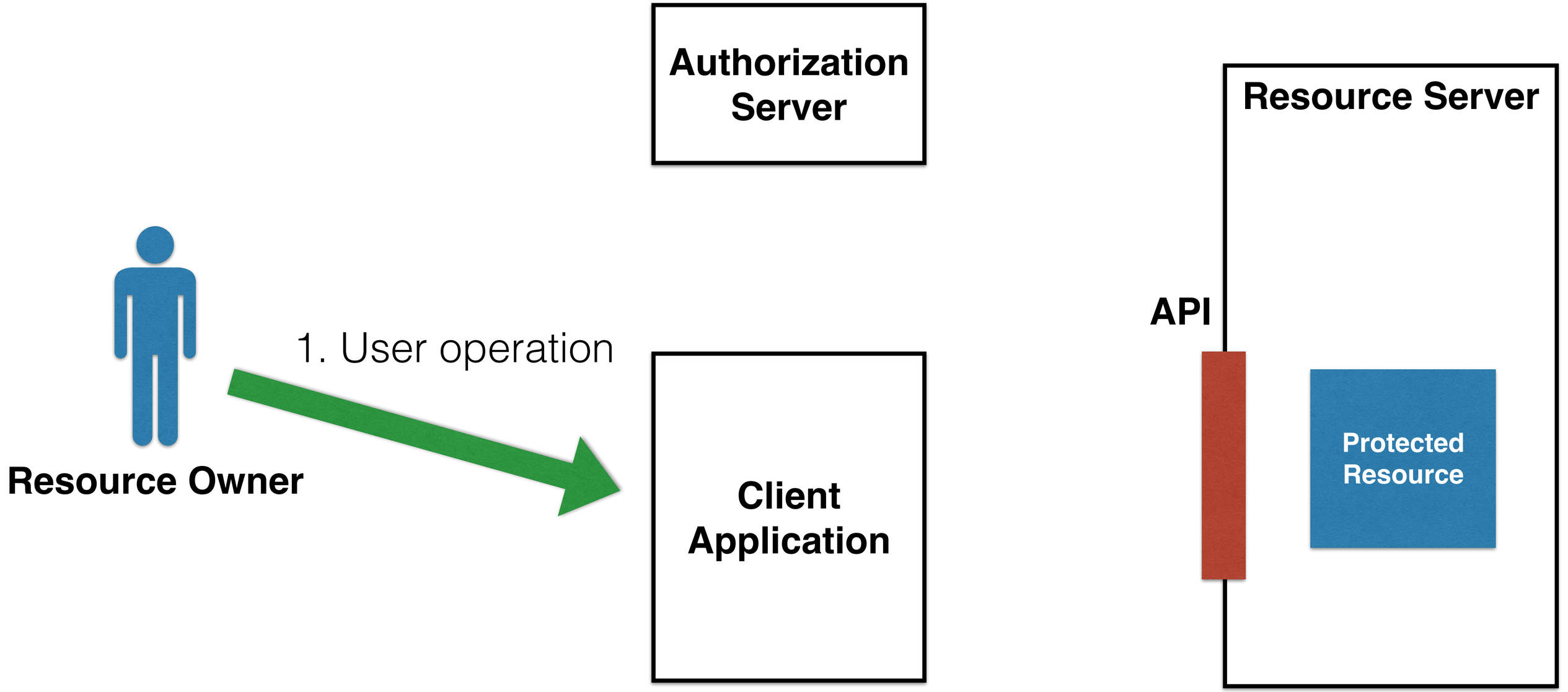
OAuth 2.0 (monolithic w/ external identification)

- In a monolithic architecture, an external authorisation server is an identity provider.

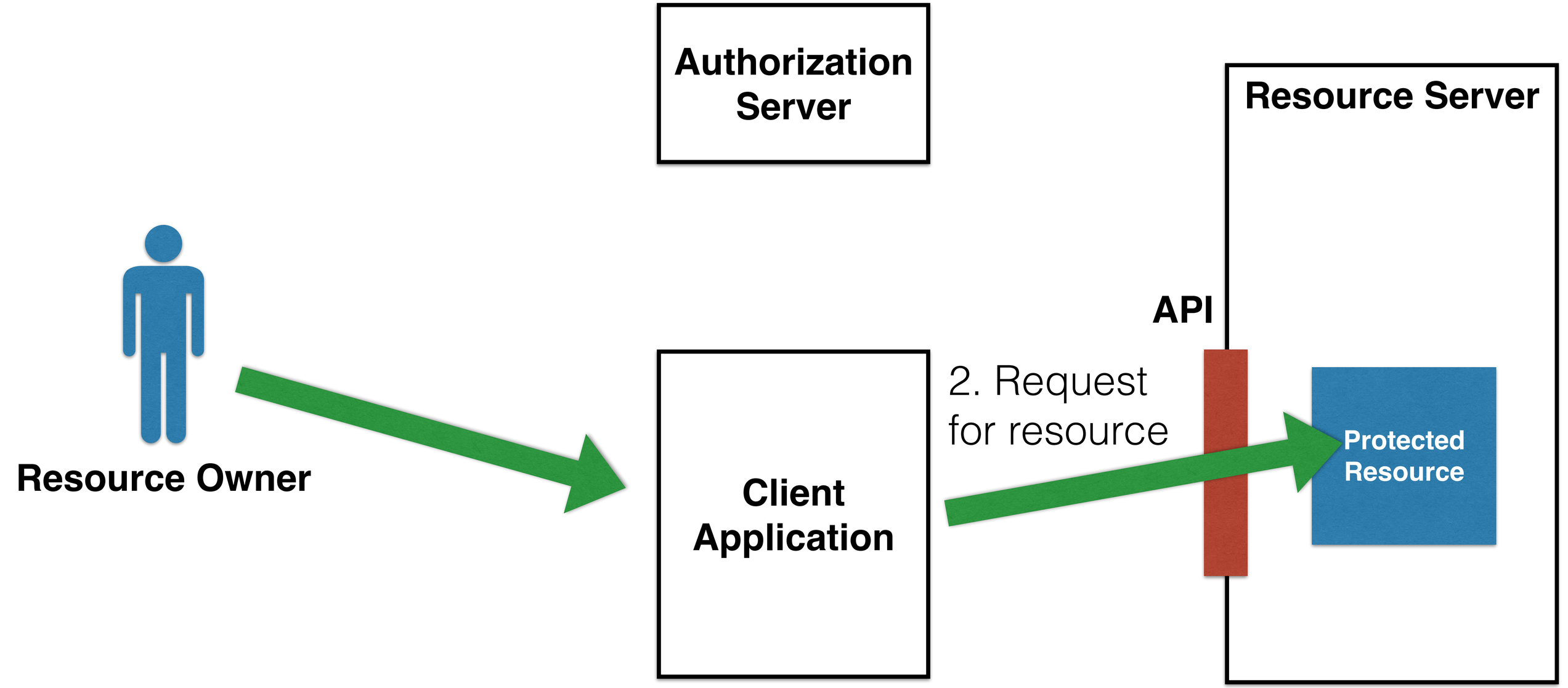


OAuth 2.0 flow

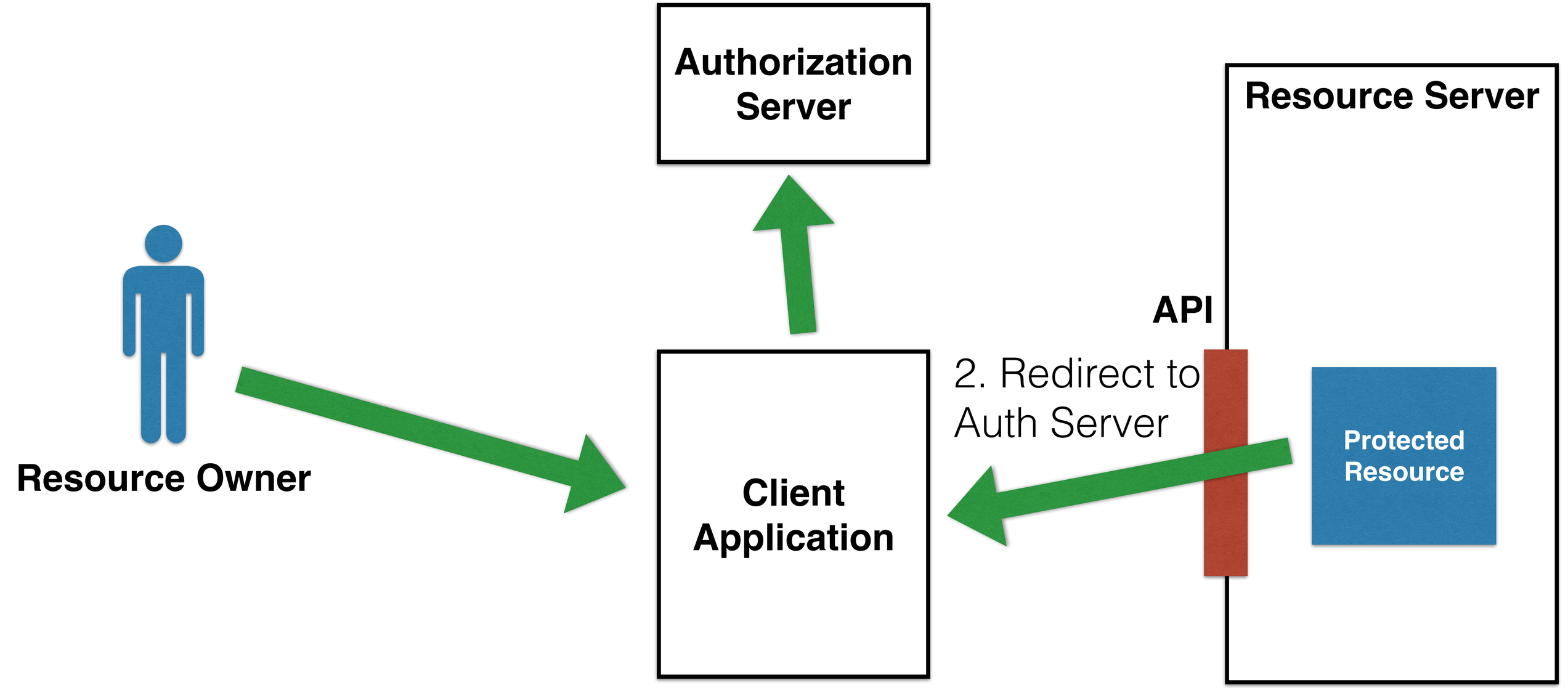
OAuth 2.0 flow



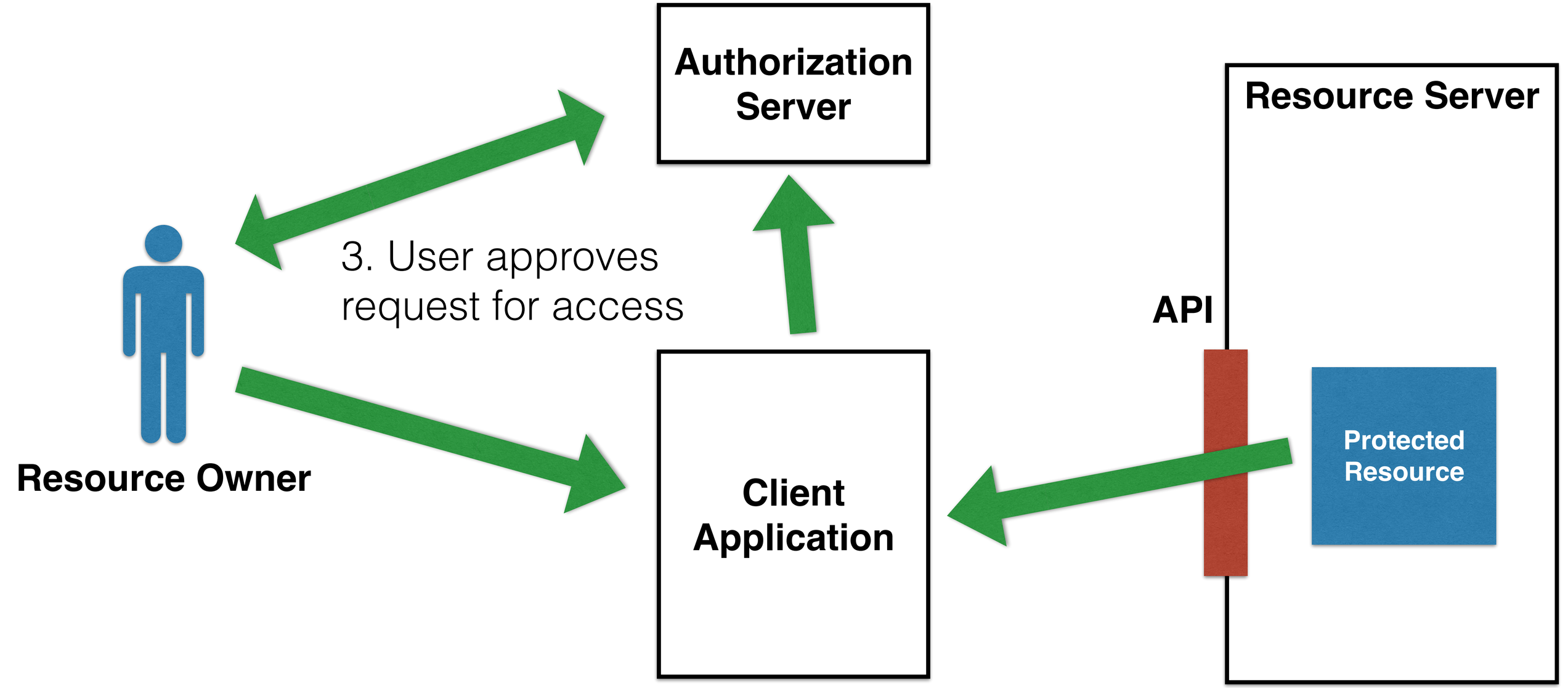
OAuth 2.0 flow



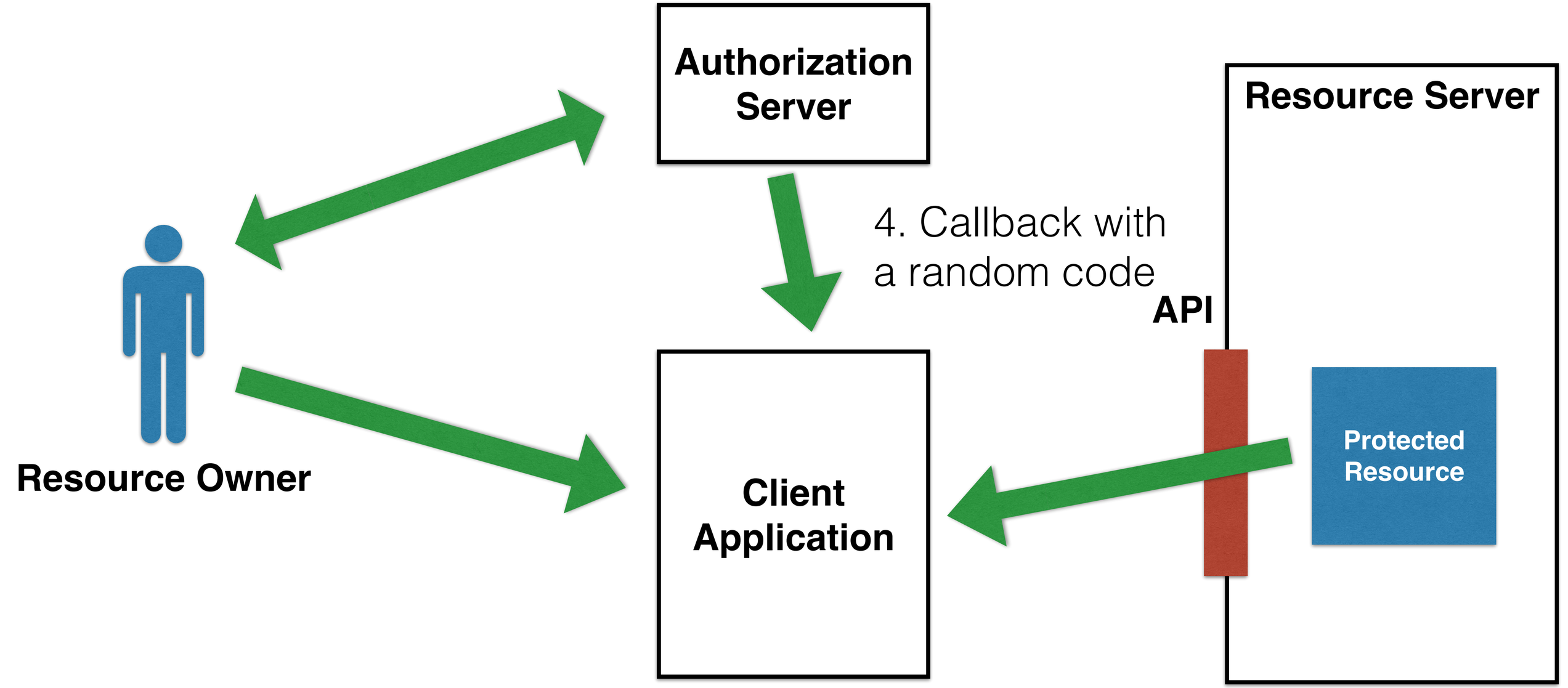
OAuth 2.0 flow



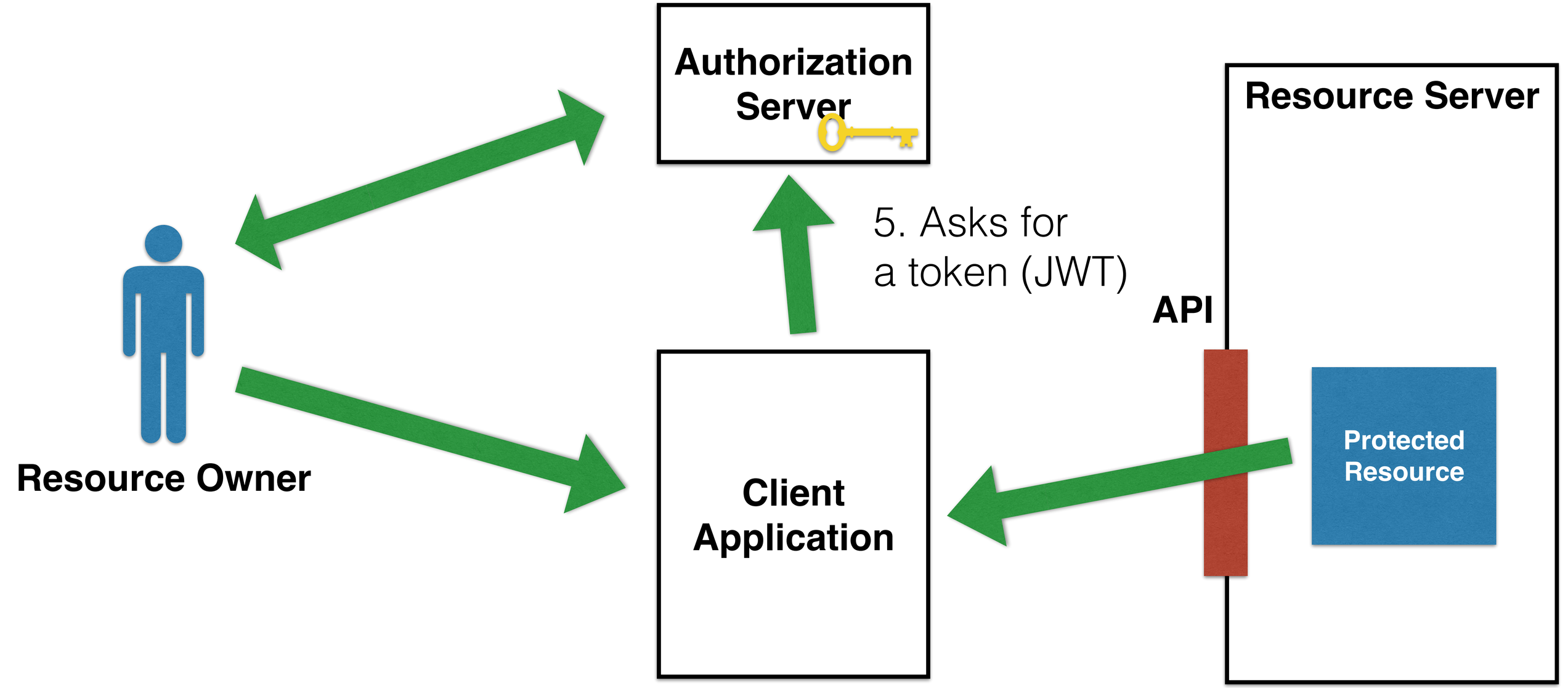
OAuth 2.0 flow



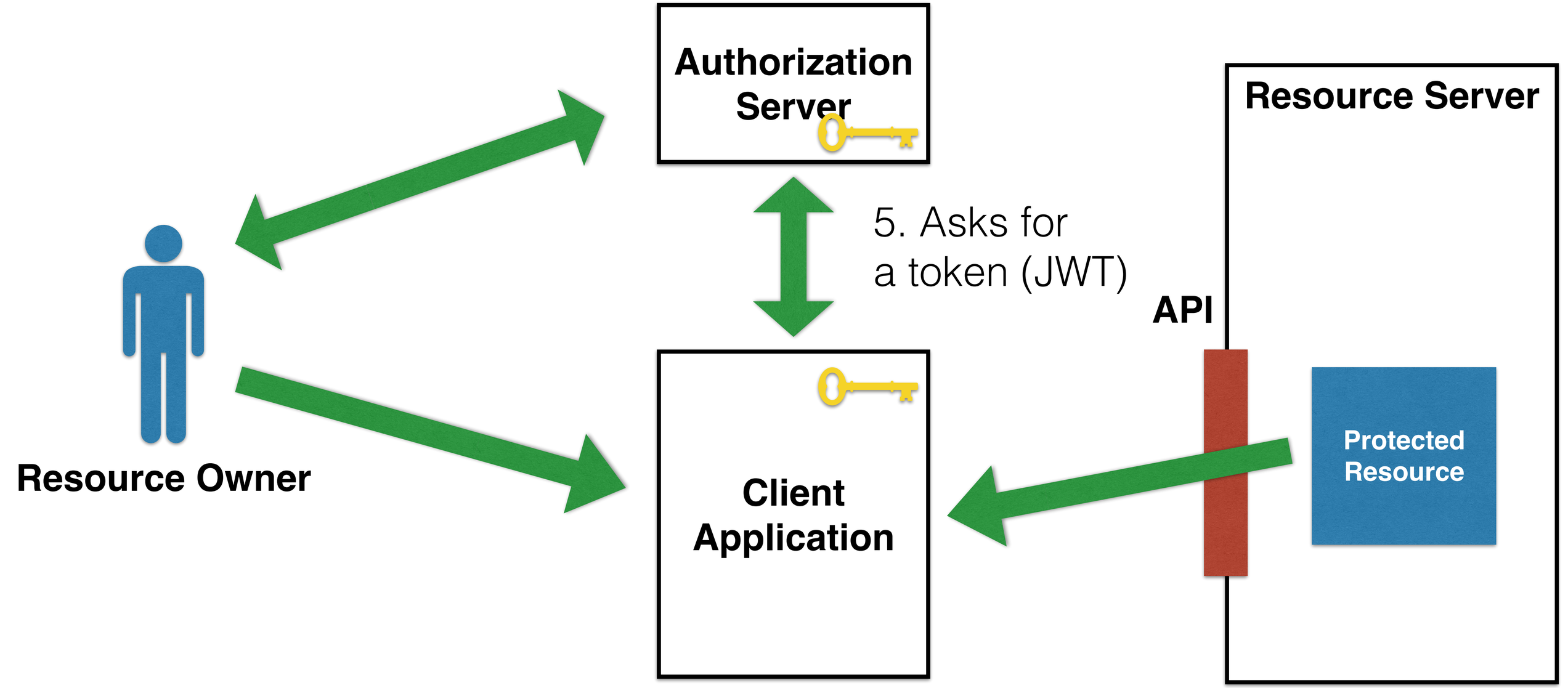
OAuth 2.0 flow



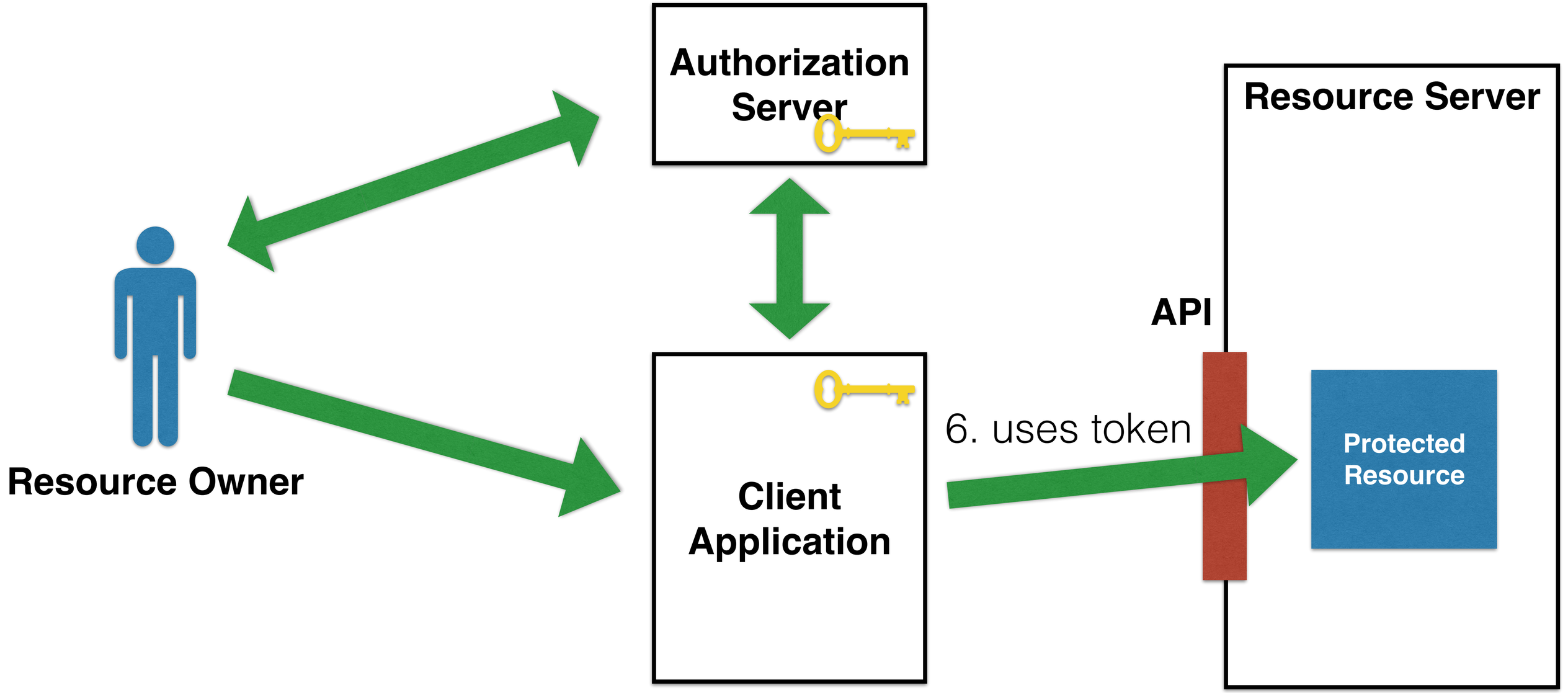
OAuth 2.0 flow



OAuth 2.0 flow



OAuth 2.0 flow



How does it start? With a unauthenticated request!

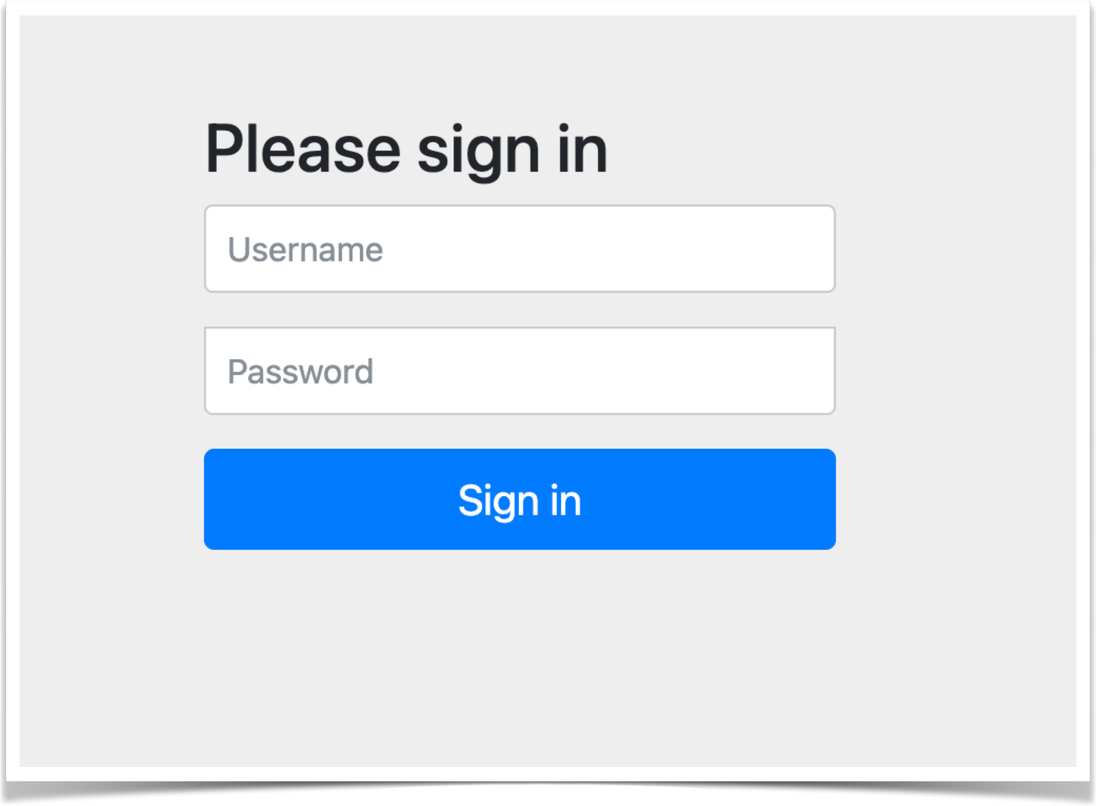
```
const authenticated = (req, res, next) => {
    if (!req.user) res.redirect('/auth/provider')
    else next()
}

app.get('/profile', authenticated, (req, res) => {
  res.send(`
    <h1>Profile</h1>
    <pre>${JSON.stringify(req.user, null, 2)}</pre>
    <a href="/logout">Logout</a>
  `);
});

app.get('/auth/provider', (req, res, next) => {
    passport.authenticate('oauth2', { scope: ['profile'] })(req, res, next);
});
```

`req.user` Is the conventional way express applications to store the user session

A request to /profile causes…

- the Application (the OAuth client) to redirect the user to

```
http://localhost:9000/authorize?
response_type=code&
redirect_uri=http%3A%2F%2Flocalhost%3A3000%2Fauth%2Fprovider%2Fcallback&
scope=profile&
state=TSYPoK8q8DK6WTRh9bmgneEy&
client_id=my-client2
```



Next, after logging in…

- The Authorization Server redirects the browser to

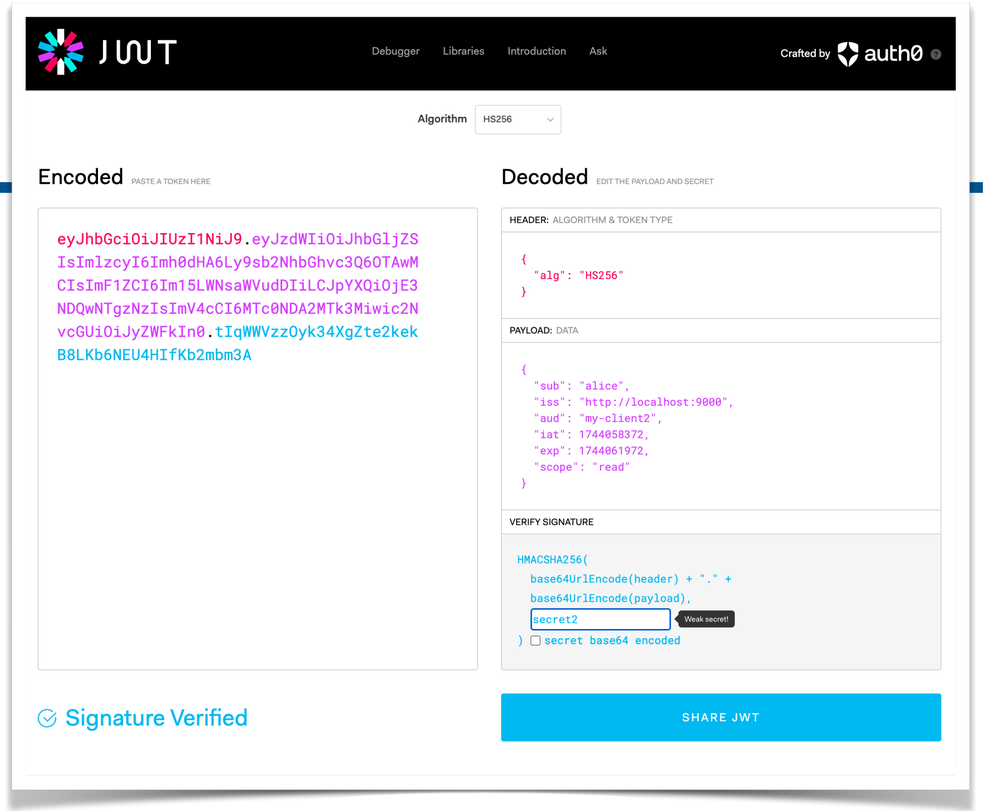
```
http://localhost:3000/auth/provider/callback?
code=cff3984e-f0cb-4110-9e72-4d179314a7d6&
state=TSYPoK8q8DK6WTRh9bmgneEy
```

- The Application gets the token from

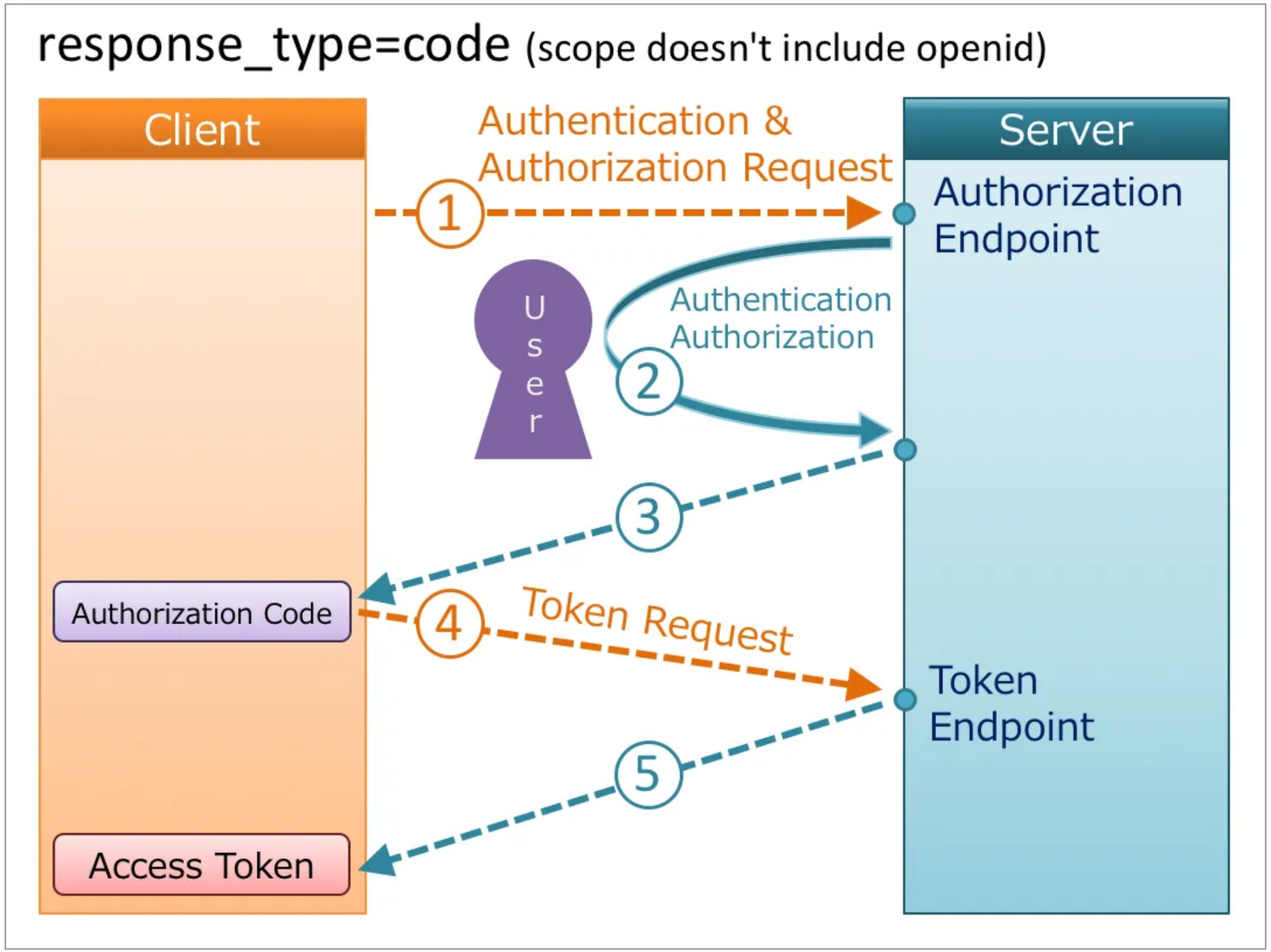
```
http://localhost:9000/token?
code=cff3984e-f0cb-4110-9e72-4d179314a7d6
```

  with basic authentication

- And proceeds to the initial request URL while authenticated…

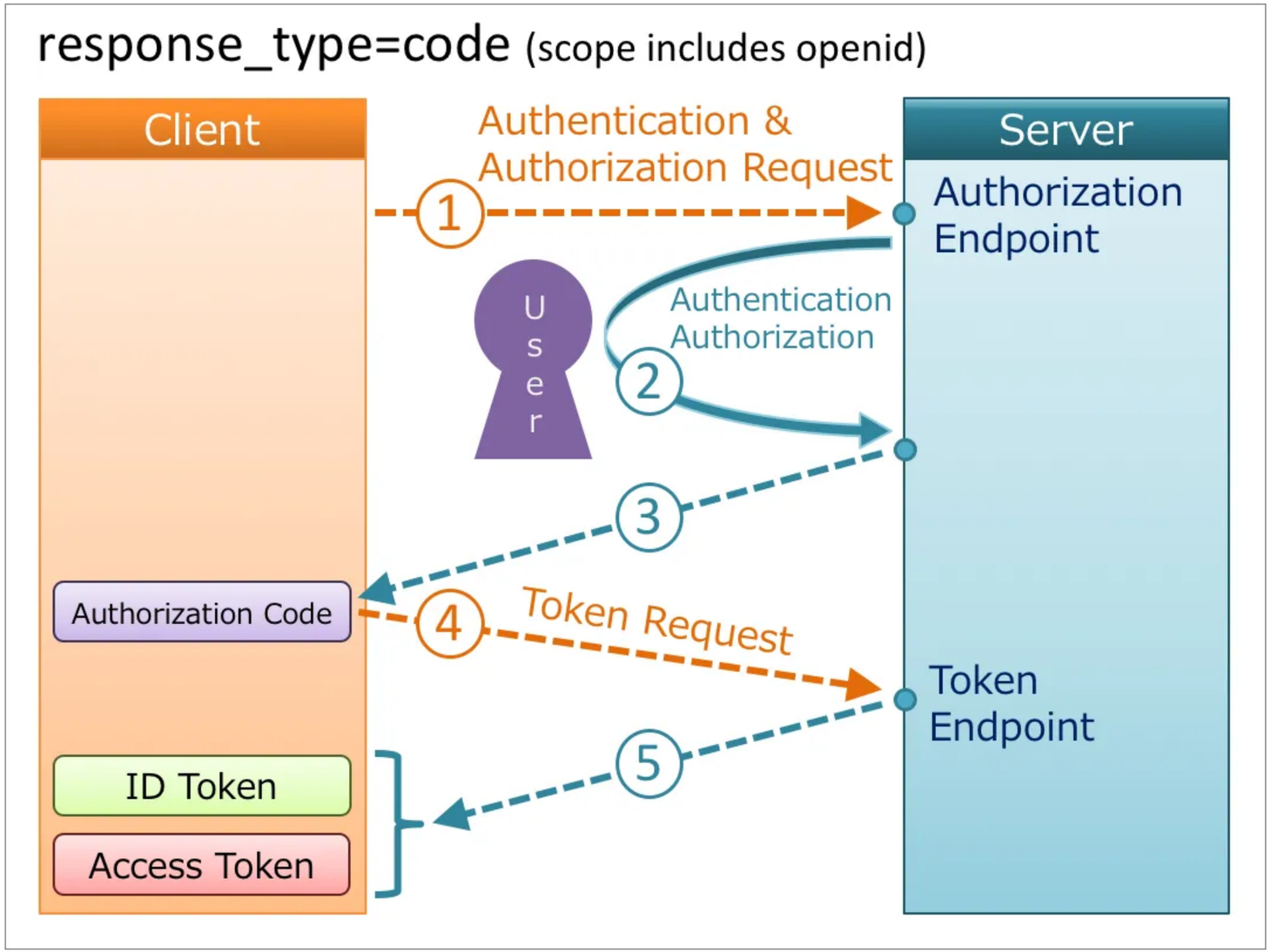


OAuth 2 flow



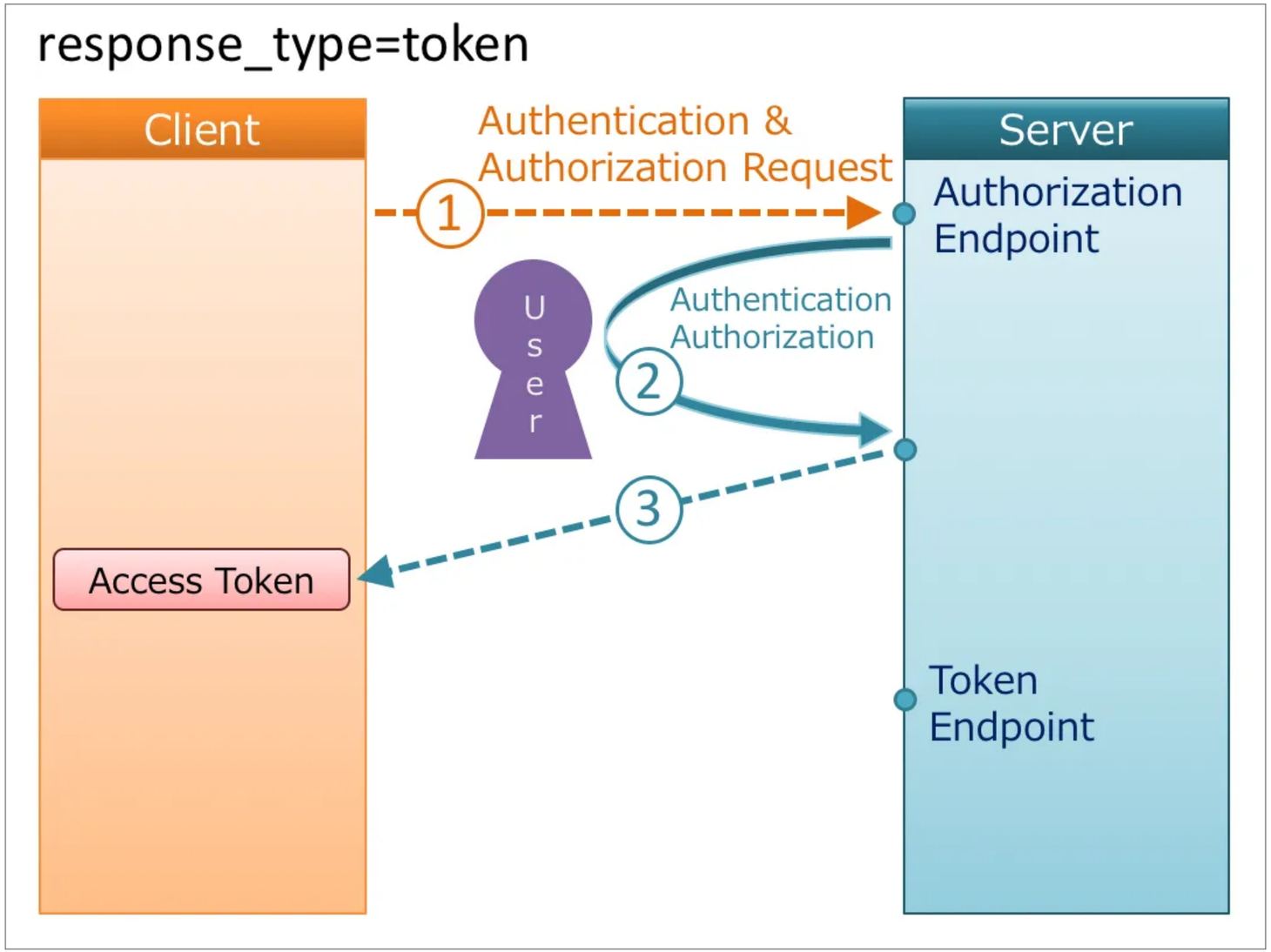
https://darutk.medium.com/diagrams-of-all-the-openid-connect-flows-6968e3990660

OAuth 2 flow



https://darutk.medium.com/diagrams-of-all-the-openid-connect-flows-6968e3990660

OAuth 2 flow



Implicit mode, only for serverless applications

https://darutk.medium.com/diagrams-of-all-the-openid-connect-flows-6968e3990660

Configuration of a client using Spring

```yaml
spring:
  security:
    oauth2:
      client:
        registration:
          github:
            clientId: github-client-id
            clientSecret: github-client-secret
```

```
dependencies {
    implementation("org.springframework.boot:spring-boot-starter-web")
    implementation("org.springframework.boot:spring-boot-starter-security")
    implementation("org.springframework.boot:spring-boot-starter-oauth2-client")
}
```

Configuration of a client using Spring

```
spring:
  security:
    oauth2:
      client:
        registration:
          custom:
            client-id: my-client
            client-secret: secret
            authorization-grant-type: authorization_code
            redirect-uri: "{baseUrl}/login/oauth2/code/custom"
         provider:
            custom:
              authorization-uri: http://localhost:9000/authorize
              token-uri: http://localhost:9000/token
```

Configuration of a client using Express

```
const strategy = new OAuth2Strategy(
    {
      authorizationURL: 'http://localhost:9000/authorize',
      tokenURL: 'http://localhost:9000/token',
      clientID: 'my-client2',
      clientSecret: 'secret2',
      callbackURL: 'http://localhost:3000/auth/provider/callback',
      state: true,
      customHeaders: {
        Authorization: 'Basic ' + Buffer.from('my-client2:secret2').toString('base64')
      }
    },
    (accessToken, refreshToken, profile, cb) => {
      const user = { accessToken, refreshToken };
      return cb(null, user);
    }
  )
passport.use(strategy);
```

Project, Part 1 (due April 28th)

- Implement an Authorization Server from scratch using a stack of your choice to comply with OAuth 2.0. Implement a client application using a standard library client for OAuth 2.0.
  - Your Authorization Server should allow the registration of new applications (API).
  - Your Authorization Server should allow new users to register (API).
  - The instructions for this API must be made public to the class (via a link on Discord and a README online).
- **Tier 1 (15)**: Your Authorization server is used locally by your client application
- **Tier 2 (17)**: Your Authorization server is deployed (e.g. using Heroku) on the Internet and used by your client application using HTTPS
- **Tier 3 (20)**: Someone else's client application uses your deployed Authorization server (1 point per client up to 3)
- **Extra points**: Your client application can connect to multiple deployed Authorization Servers (0.5 per server up to 3)
- Proof of connection should be in JWT tokens (identifying the authors of the server) and checked in person.
- Submit a report with the steps of construction and configurations needed to make the Authorization server work.
- The grade limit is just a limit. The final grade depends on the manual evaluation of your code.