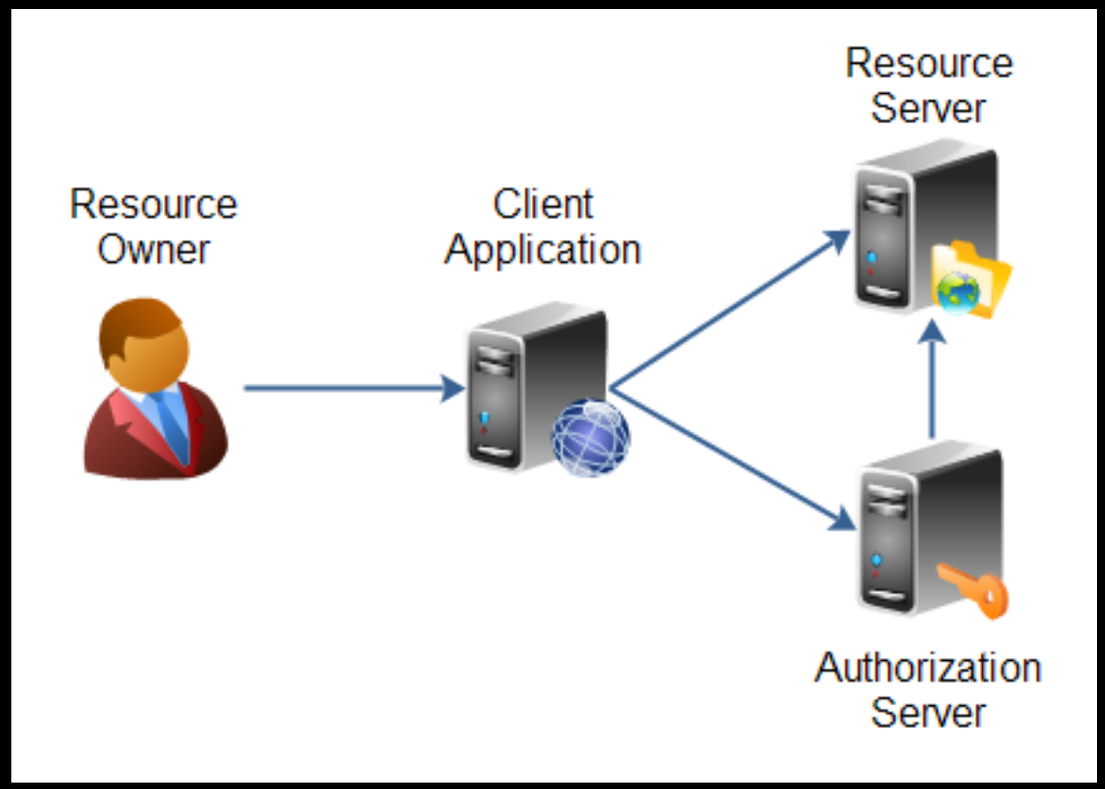
Oauth 2.0 explained

What is it again ?

- Oauth 2.0 specification is not a protocol ,it is a framework.
- Oauth 2.0 is NOT and authentication framework. It is an access delegation framework.

Who is involved?

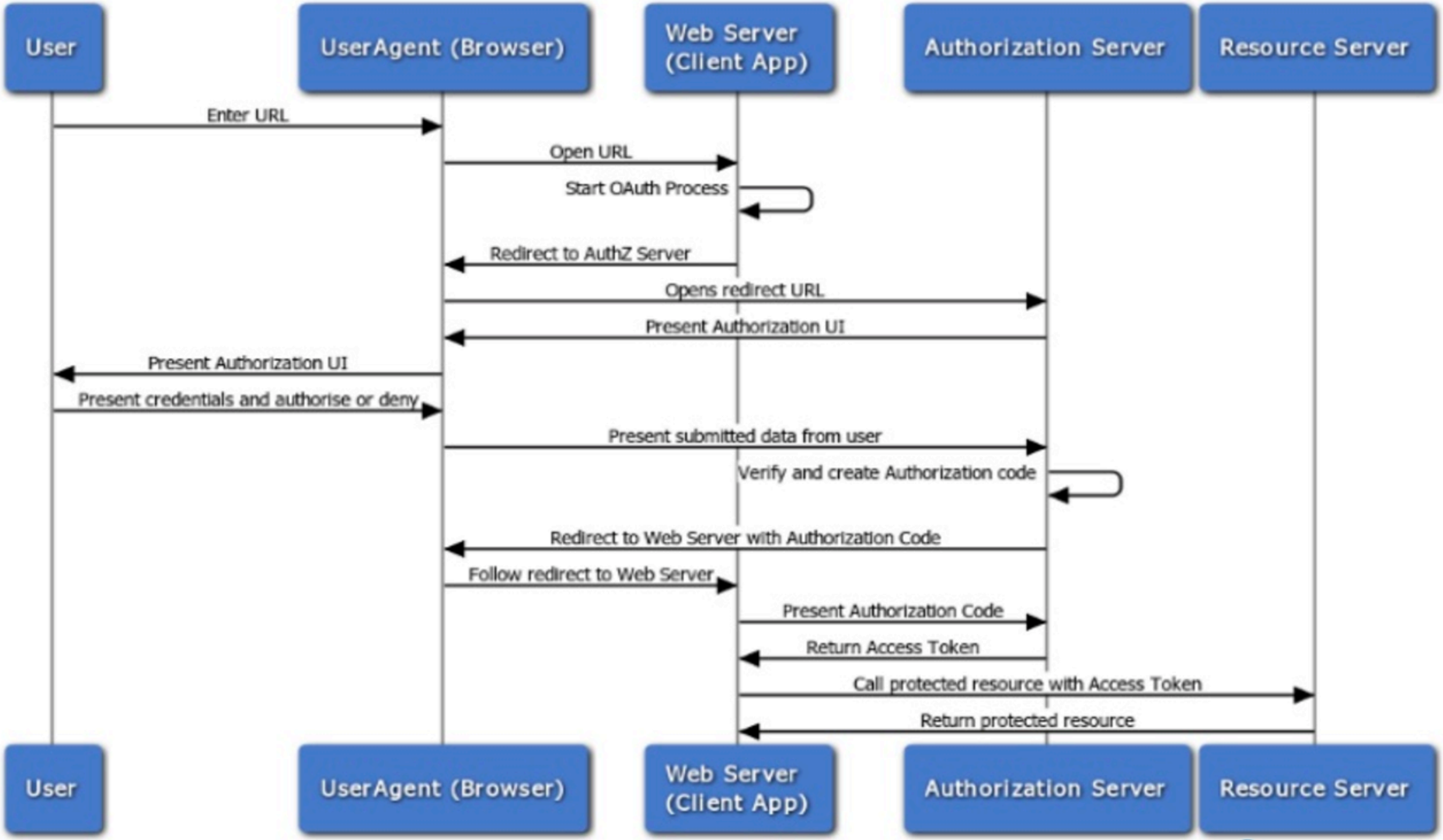
- Resource owner.
- Client application.
- Resource server.
- Authorization server.



Oauth flows

- **Authorization code grant.(for web apps)**
  - Use authorisation code from the server to obtain access token.
- **Implicit grant. (mobile apps/ native clients)**
- **Resource owner password grant.(username/ password)**
- **Client credential grant.(application specific codes)**

Authorization Code



The Flow (Part One)

The client will redirect the user to the authorization server with the following parameters in the query string:

- `response_type` with the value `code`
- `client_id` with the client identifier
- `redirect_uri` with the client redirect URI. This parameter is optional, but if not send the user will be redirected to a pre-registered redirect URI.
- `scope` a space delimited list of scopes
- `state` with a CSRF token. This parameter is optional but highly recommended. You should store the value of the CSRF token in the user's session to be validated when they return.

All of these parameters will be validated by the authorization server.

The user will then be asked to login to the authorization server and approve the client.

If the user approves the client they will be redirected from the authorisation server back to the client (specifically to the redirect URI) with the following parameters in the query string:

- `code` with the authorization code
- `state` with the state parameter sent in the original request. You should compare this value with the value stored in the user's session to ensure the authorization code obtained is in response to requests made by this client

The Flow (Part Two)

The client will now send a POST request to the authorization server with the following parameters:

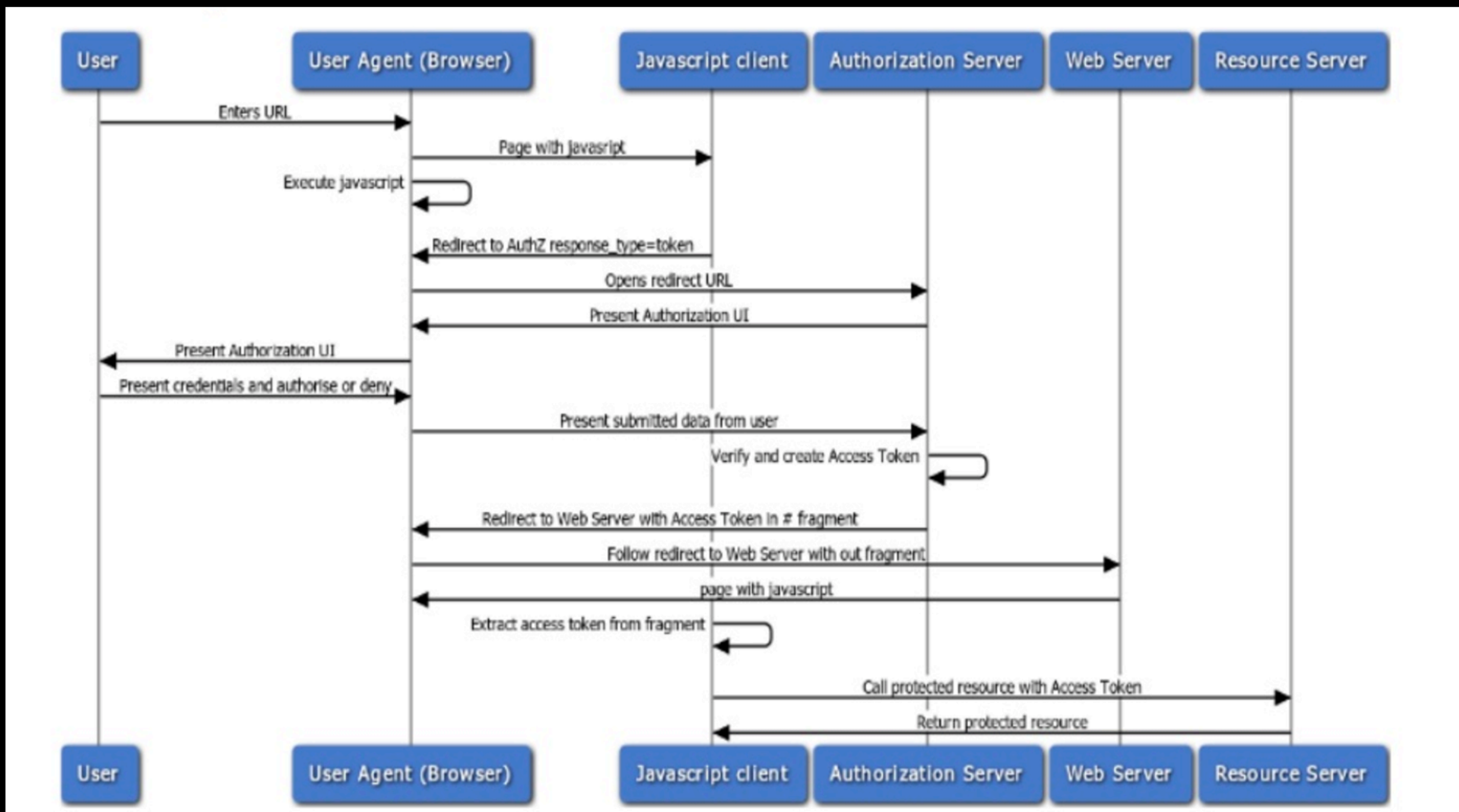
- `grant_type` with the value of `authorization_code`
- `client_id` with the client identifier
- `client_secret` with the client secret
- `redirect_uri` with the same redirect URI the user was redirect back to
- `code` with the authorization code from the query string

The authorization server will respond with a JSON object containing the following properties:

- `token_type` this will usually be the word "Bearer" (to indicate a bearer token)
- `expires_in` with an integer representing the TTL of the access token (i.e. when the token will expire)
- `access_token` the access token itself
- `refresh_token` a refresh token that can be used to acquire a new access token when the original expires

---

Implicit


User
User Agent (Browser)
Javascript client
Authorization Server
Web Server
Resource Server
Enters URL
Page with javasript
Execute javascript
Redirect to AuthZ response_type=token
Opens redirect URL
Present Authorization UI
Present Authorization UI
Present credentials and authorise or deny
Present submitted data from user
Verify and create Access Token
Redirect to Web Server with Access Token in # fragment
Follow redirect to Web Server with out fragment
page with javascript
Extract access token from fragment
Call protected resource with Access Token
Return protected resource
User
User Agent (Browser)
Javascript client
Authorization Server
Web Server
Resource Server

The client will redirect the user to the authorization server with the following parameters in the query string:

- `response_type` with the value `token`
- `client_id` with the client identifier
- `redirect_uri` with the client redirect URI. This parameter is optional, but if not sent the user will be redirected to a pre-registered redirect URI.
- `scope` a space delimited list of scopes
- `state` with a CSRF token. This parameter is optional but highly recommended. You should store the value of the CSRF token in the user's session to be validated when they return.

All of these parameters will be validated by the authorization server.

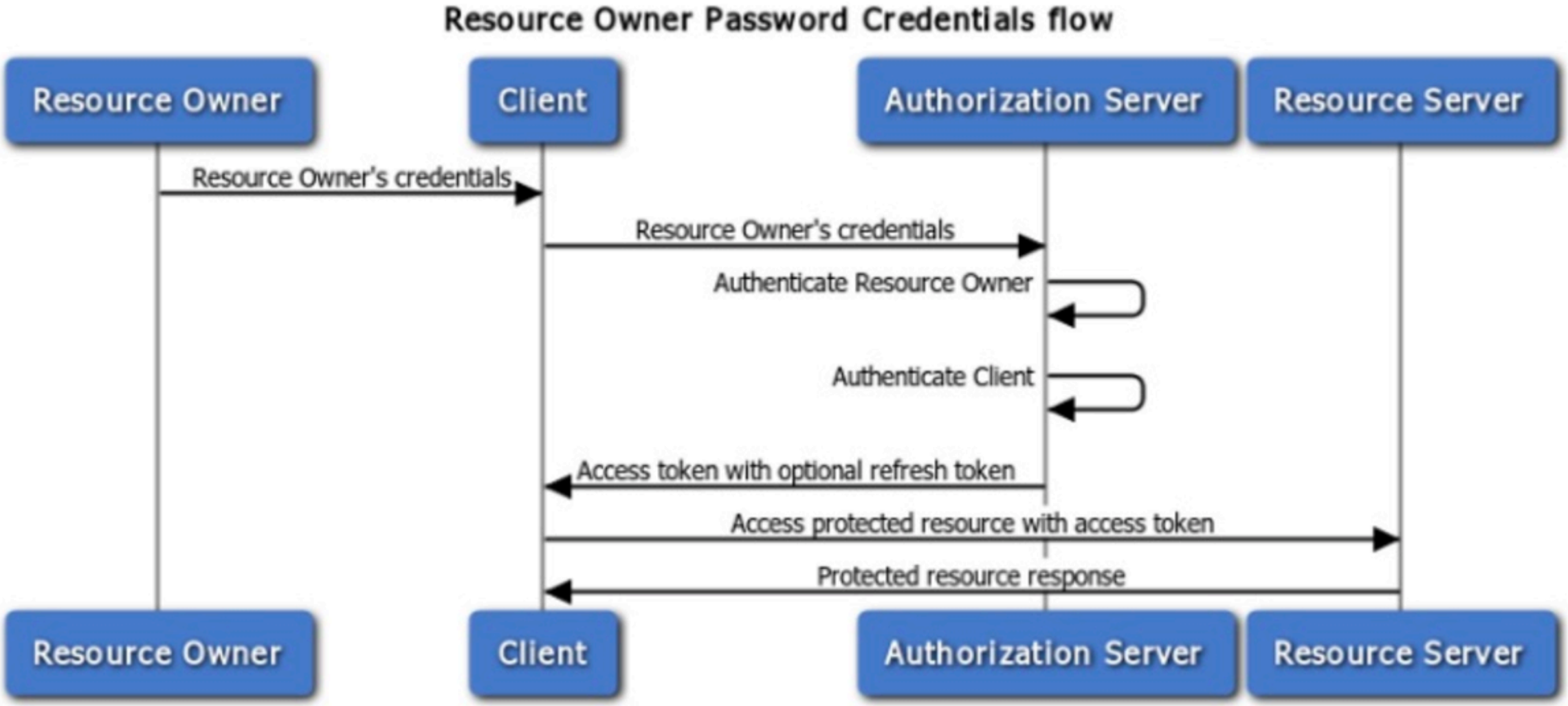
The user will then be asked to login to the authorization server and approve the client.

If the user approves the client they will be redirected back to the authorization server with the following parameters in the query string:

- `token_type` with the value `Bearer`
- `expires_in` with an integer representing the TTL of the access token
- `access_token` the access token itself
- `state` with the state parameter sent in the original request. You should compare this value with the value stored in the user's session to ensure the authorization code obtained is in response to requests made by this client rather than another client application.

Note: this grant does not return a refresh token because the browser has no means

Resource Owner Password Credentials



This grant is a great user experience for trusted first party clients both on the web and in native device applications.

The Flow

The client will ask the user for their authorization credentials (ususally a username and password).

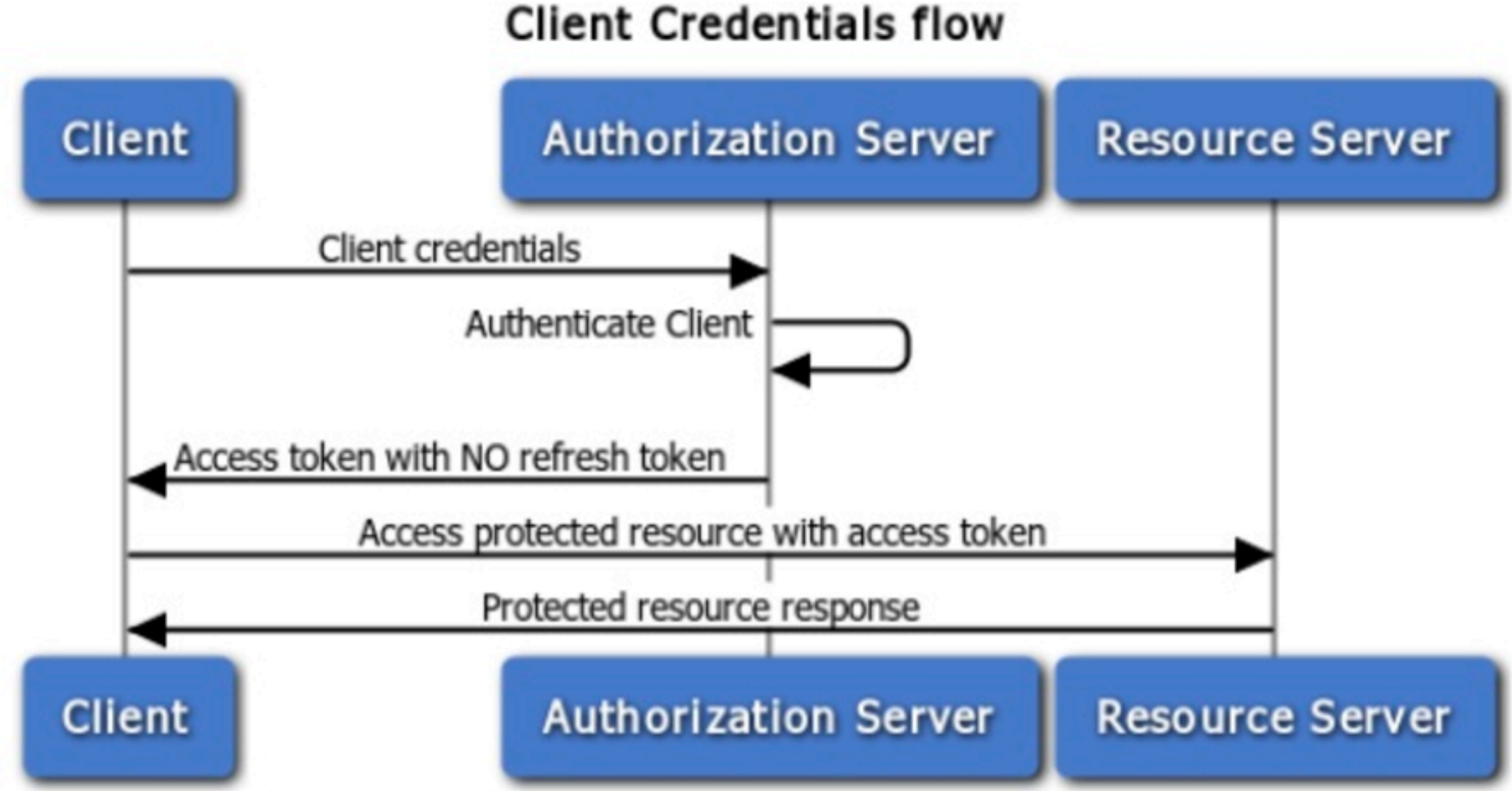
The client then sends a POST request with following body parameters to the authorization server:

- `grant_type` with the value `password`
- `client_id` with the the client's ID
- `client_secret` with the client's secret
- `scope` with a space-delimited list of requested scope permissions.
- `username` with the user's username
- `password` with the user's password

The authorization server will respond with a JSON object containing the following properties:

- `token_type` with the value `Bearer`
- `expires_in` with an integer representing the TTL of the access token
- `access_token` the access token itself
- `refresh_token` a refresh token that can be used to acquire a new access token when the original expires

Client Credentials



The simplest of all of the OAuth 2.0 grants, this grant is suitable for machine-to-machine authentication where a specific user's permission to access data is not required.

The Flow

The client sends a POST request with following body parameters to the authorization server:

- `grant_type` with the value `client_credentials`
- `client_id` with the the client's ID
- `client_secret` with the client's secret
- `scope` with a space-delimited list of requested scope permissions.

The authorization server will respond with a JSON object containing the following properties:

- `token_type` with the value `Bearer`
- `expires_in` with an integer representing the TTL of the access token
- `access_token` the access token itself

Refresh token grant (section 1.5)

Access tokens eventually expire; however some grants respond with a refresh token which enables the client to get a new access token without requiring the user to be redirected.

The Flow

The client sends a POST request with following body parameters to the authorization server:

- `grant_type` with the value `refresh_token`
- `refresh_token` with the refresh token
- `client_id` with the the client's ID
- `client_secret` with the client's secret
- `scope` with a space-delimited list of requested scope permissions. This is optional; if not sent the original scopes will be used, otherwise you can request a reduced set of scopes.

The authorization server will respond with a JSON object containing the following properties:

- `token_type` with the value `Bearer`
- `expires_in` with an integer representing the TTL of the access token
- `access_token` the access token itself
- `refresh_token` a refresh token that can be used to acquire a new access

Oauth 2.0 Tokens

- Bearer
    - Larger random token.
    - Need to be protected by SSL in transit. (Oauth 1.0 have signature validation).
    - Server need to store it securely hashed.
- Mac
    - Only supported in Oauth 1.0, so you do not need to worry about it.

- Access token : Short lived token.
- Refresh token : Long lived token.

Demo Time

- https://github.com/KalpaD/oauth_demo