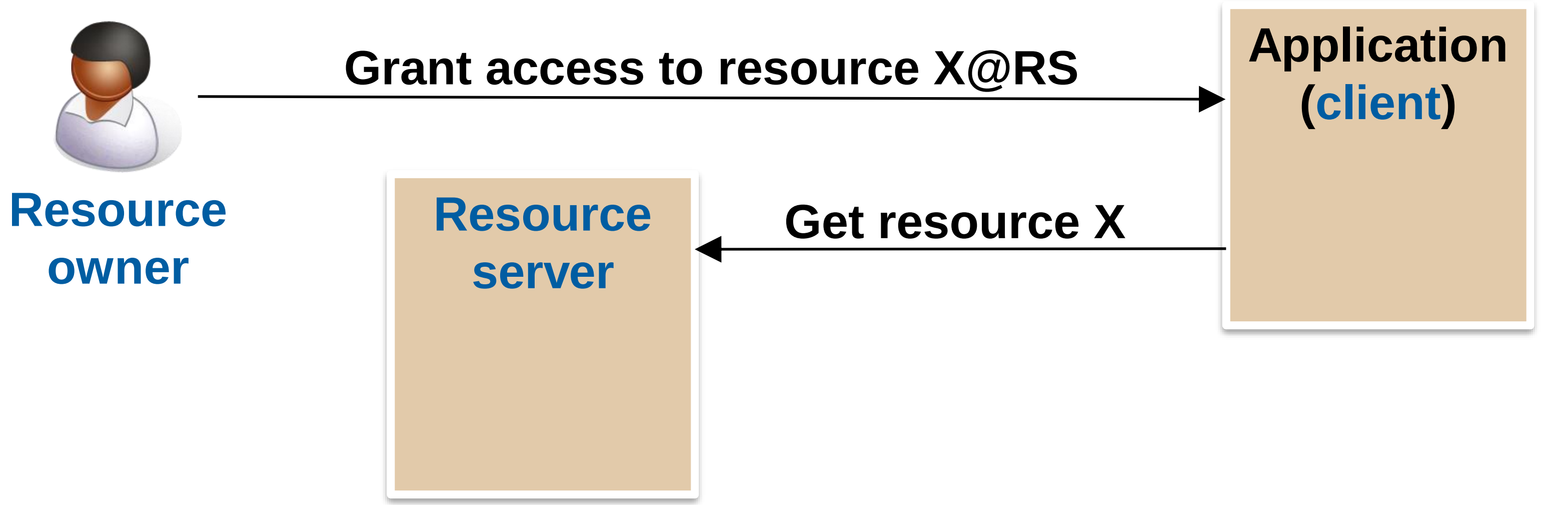
OAuth 2.0
authorization framework

Goal

- Allow an application to access user resources maintained by a service/server



- Full reference at https://oauth.net/2/

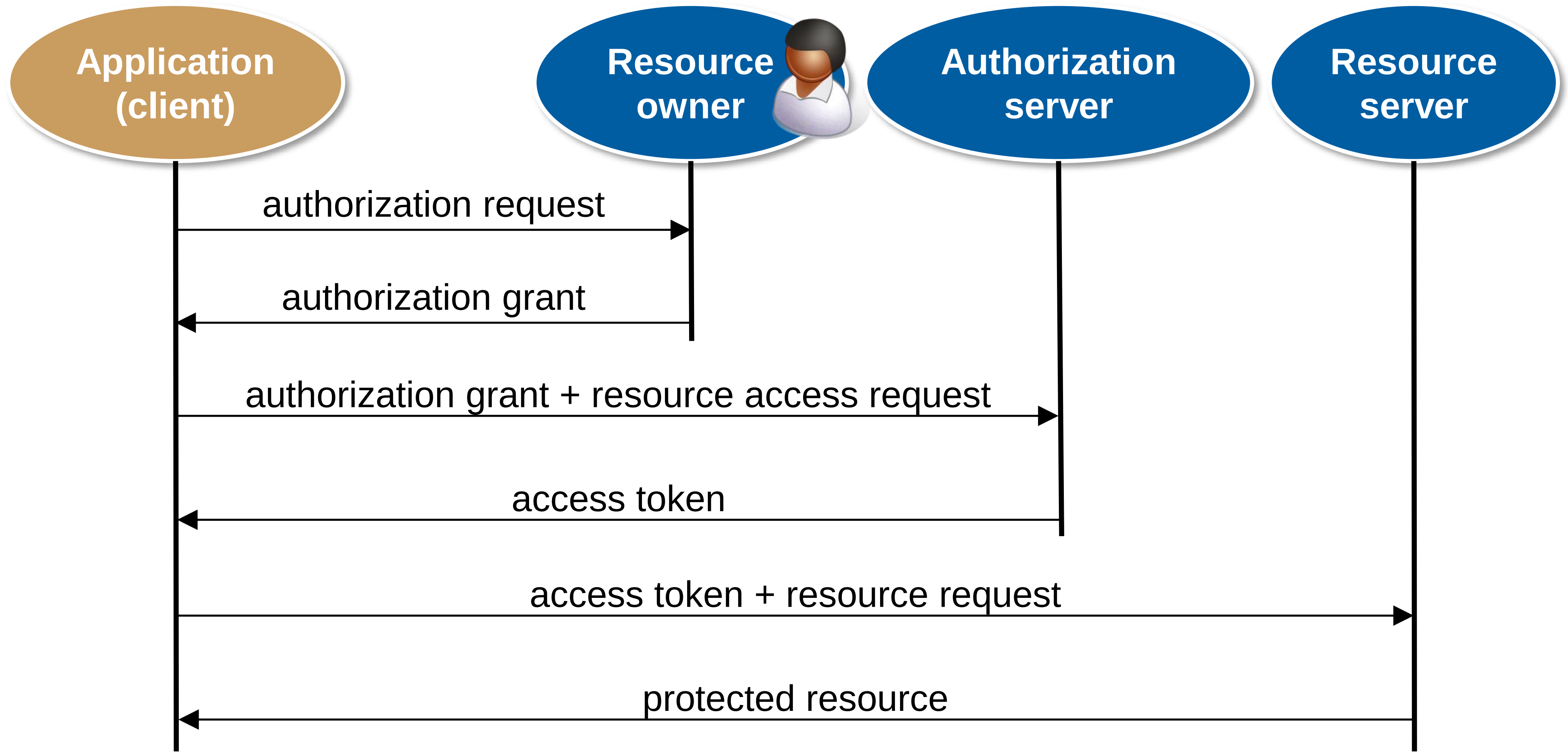
Roles (RFC 6749)

- Resource owner
  - An entity capable of **granting access** to a **protected resource**
  - **End-user**: a resource owner that is a person
- Resource server
  - The server hosting protected resources
  - Capable of accepting and responding to protected resource requests using **access tokens**

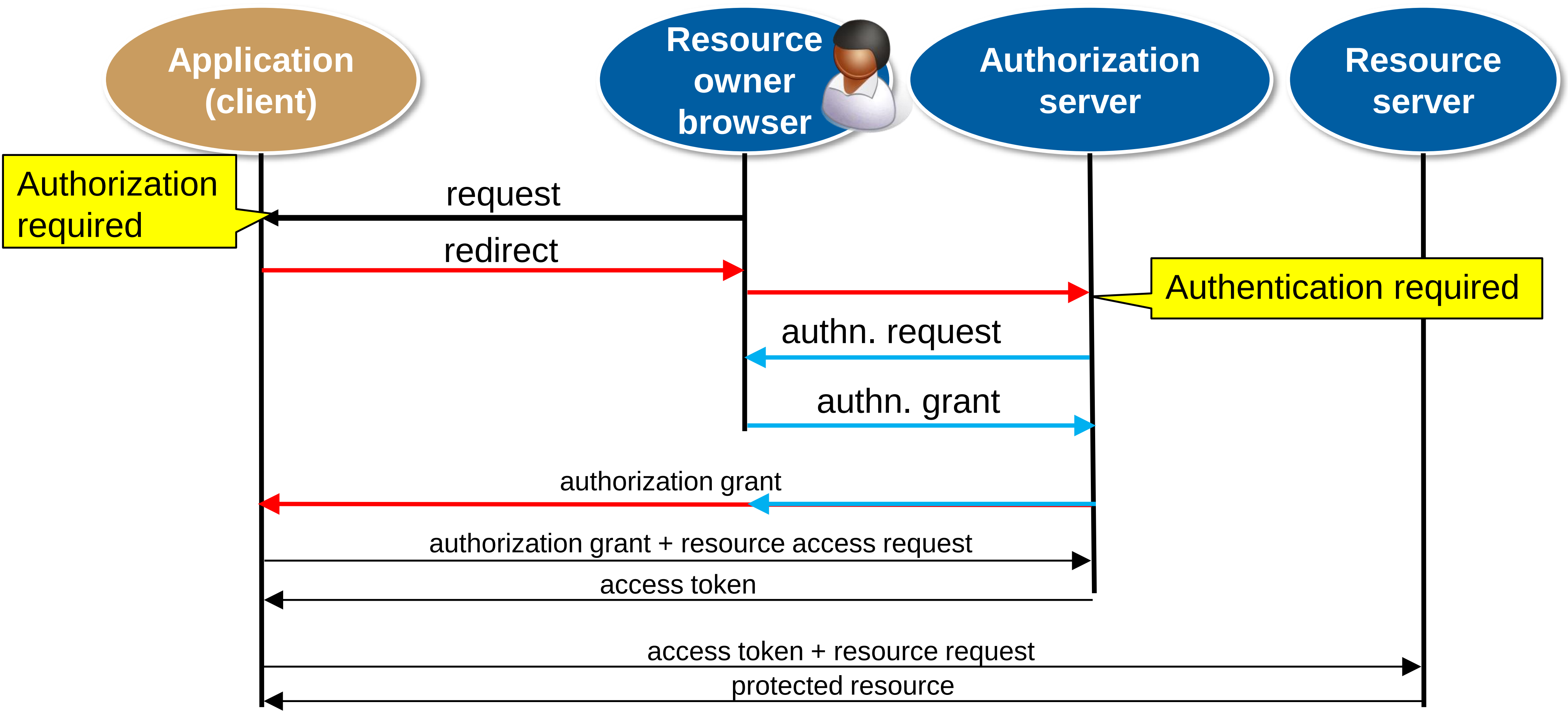
Roles (RFC 6749)

- Client
  - An **application** making requests for protected resources on behalf of the resource owner and with its authorization
- Authorization server
  (aka OAuth server or provider)
  - The server issuing **access tokens** to the client after successfully **authenticating** the resource owner and obtaining its **authorization** for the client to access one of its resources

Abstract protocol flow (RFC 6749)



Common protocol flow



Communication endpoints: Authorization endpoint

- ▷ Service provided by the **OAuth server**
  - Authenticates the resource owner (the user)
  - Asks for the delegation of access rights to its protected resources to the client
  - Send an authorization grant to the **redirection endpoint**

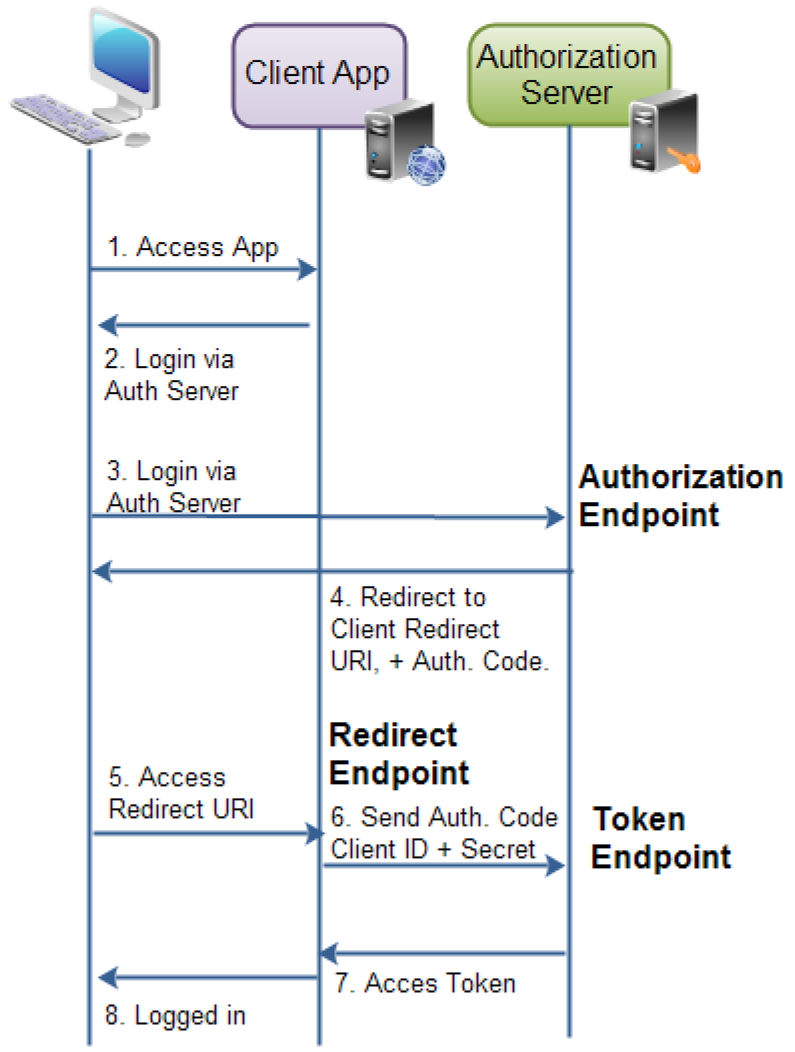
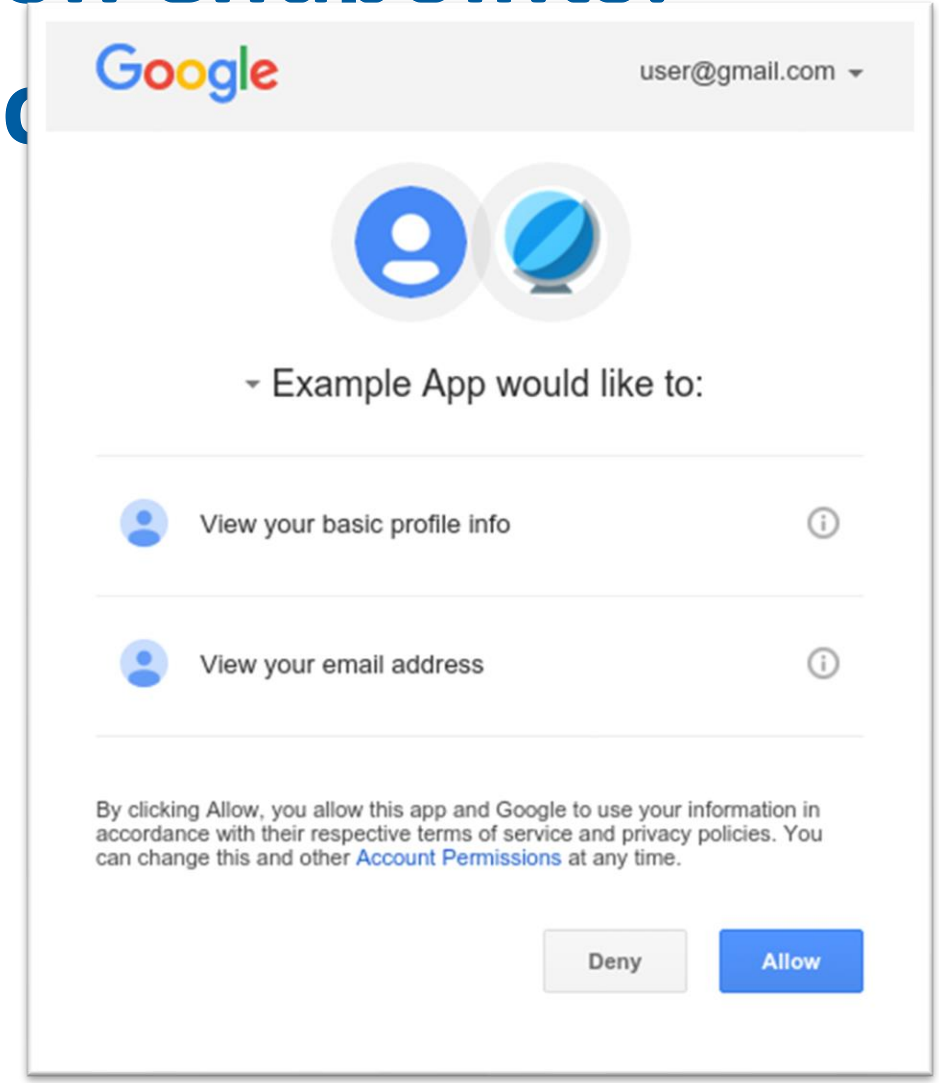


Image: https://jenkov.com/tutorials/oauth2/endpoints.html

Communication endpoints: Authorizatic



Example App would like to:

View your basic profile info

View your email address

By clicking Allow, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other Account Permissions at any time.

Communication endpoints: Token endpoint

- ▷ Service provided by the **OAuth server**
  - Produces access tokens given an authorization grant
  - It can also produce refresh tokens
  - Refresh tokens can be used to get new tokens
    - With an authorization grant

- ▷ Client authentication
  - ClientID + ClientSecret + HTTP basic authentication

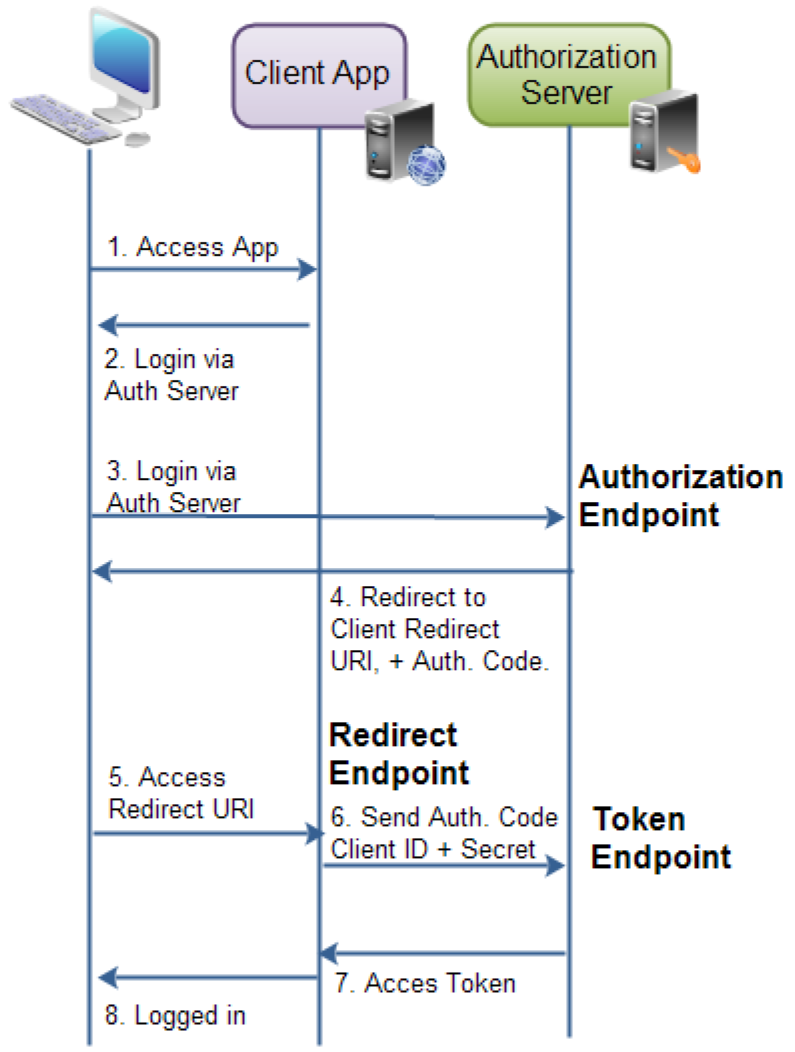


Image: https://jenkov.com/tutorials/oauth2/endpoints.html

Communication endpoints: Redirect endpoint

- ▷ Service provided by the client
  - It collects the authorization grant provided by the OAuth server
  - It should be called by the OAuth server using an HTTP redirect

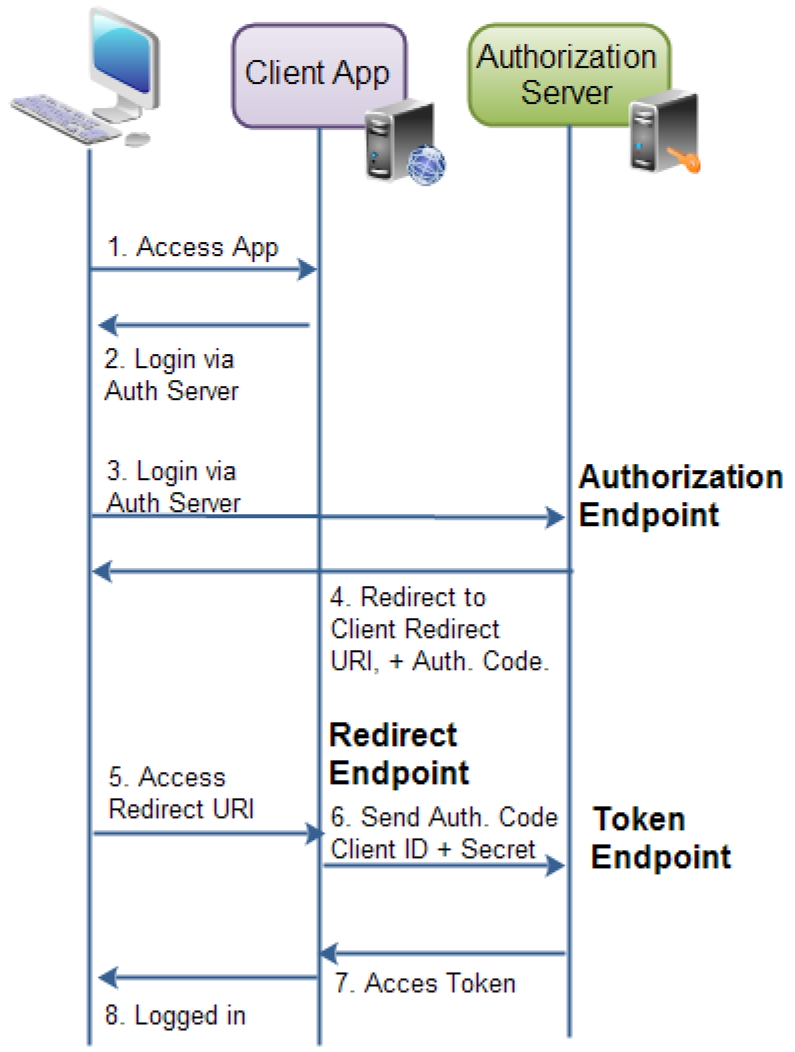


Image: https://jenkov.com/tutorials/oauth2/endpoints.html

Application (client) types

- Type is related with the ability to maintain the confidentiality of client credentials
  - Even from the resource owner

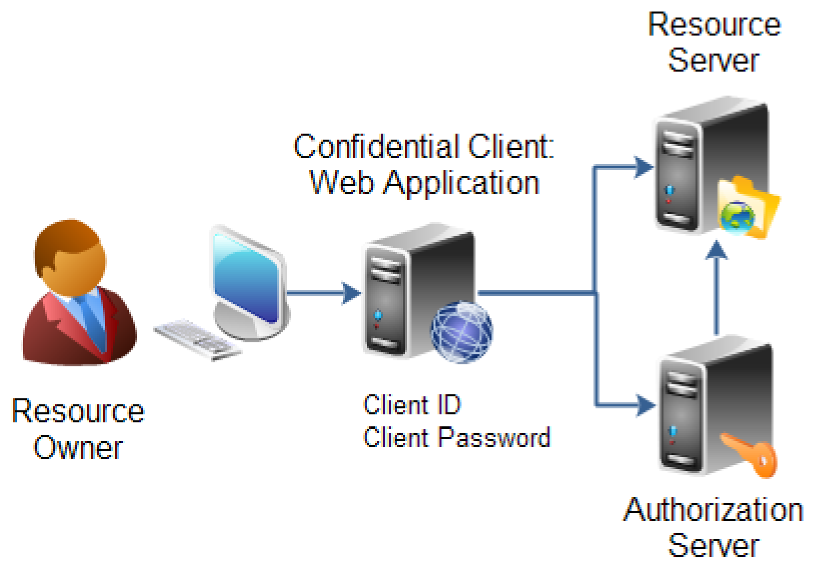
- Confidential
  - Capable
  - e.g. a secure server

- Public
  - Incapable
  - e.g. a web browser-based application, a mobile App

- Different application types will be allowed to execute different flows

Application (client) profiles

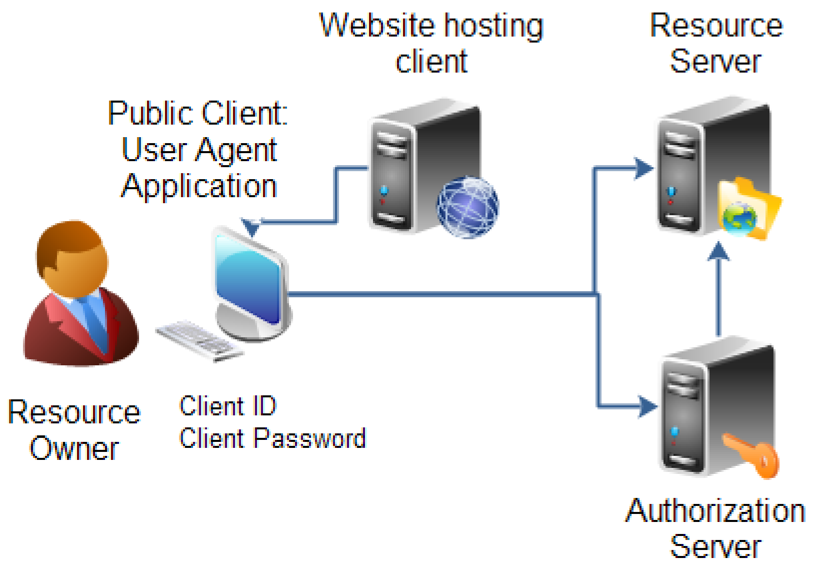
- ▷ Web application
  - Confidential client running on a web server



https://jenkov.com/tutorials/oauth2/client-types.html

Application (client) profiles

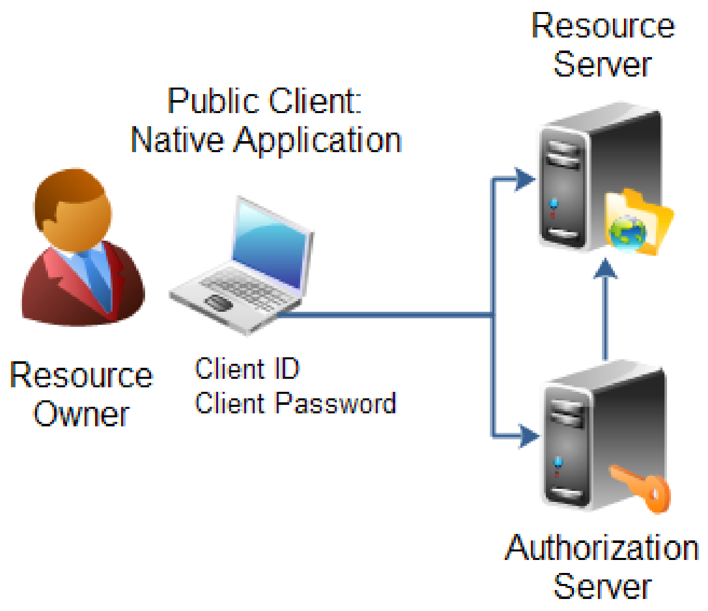
- User-agent based application
  - Public client where the client code runs on a user-agent application
    - e.g. a browser



https://jenkov.com/tutorials/oauth2/client-types.html

Application (client) profiles

- Native application
  - Public client installed and executed on the device used by the resource owner



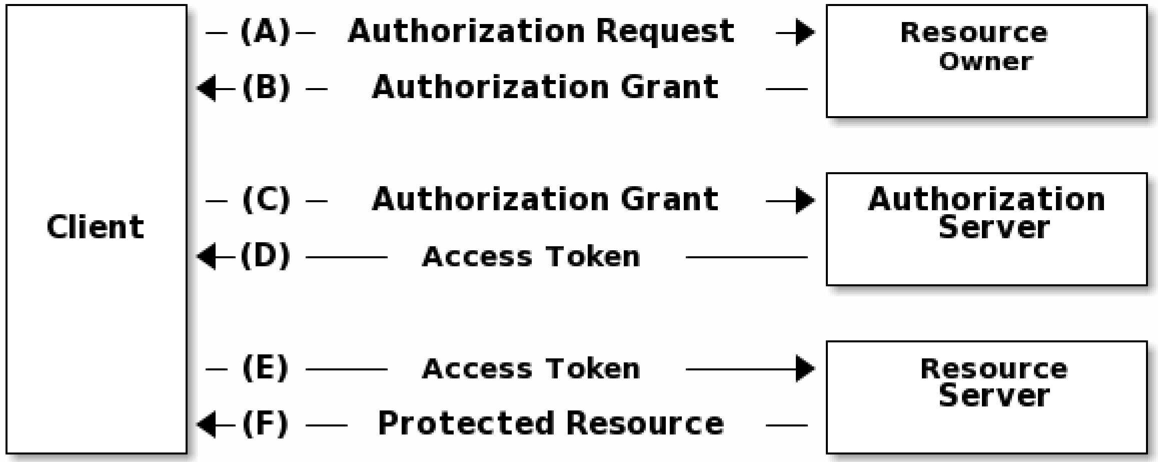
https://jenkov.com/tutorials/oauth2/client-types.html

Application (client) registration (in an OAuth server)

- Clients accessing OAuth servers must be previously registered
  - Nevertheless, the standard does not exclude unregistered clients
  - A registered client is given a unique identifier
    - ClientID
- Registration includes both informational, legal and operational information
  - Redirection URLs
  - Acceptance of legal terms
  - Application (client) name, logo, web site, description
  - Client type
  - Client authentication method (for confidential clients)

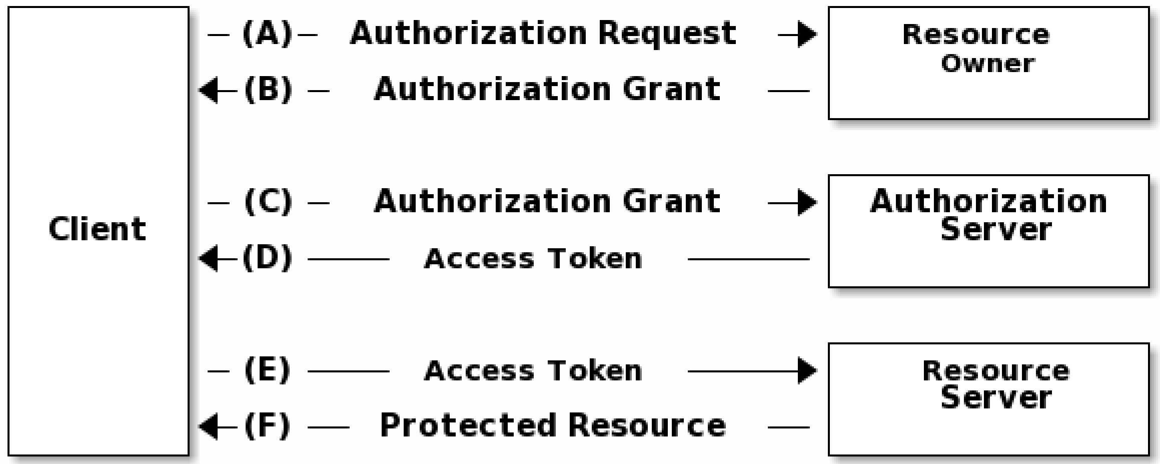
OAuth tokens: Authorization grant

- Created by an OAuth server
  - Upon authenticating a resource owner and getting its consent to grant access to a protected resource
  - An opaque byte blob that makes sense only to its issuer
- Short validity time
  - Just enough to get an access token



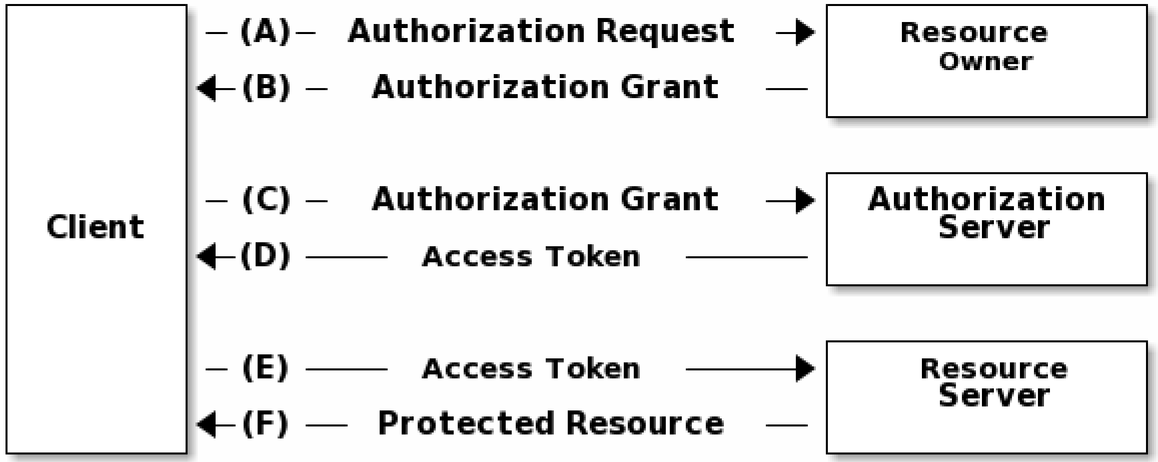
OAuth tokens:
Access token

- ▷ Created by an OAuth server
  - Upon authenticating a client and receiving an authorization grant
  - An opaque byte blob that makes sense to its issuer and to the resource owner
    - An access capability
- ▷ Bearer tokens
  - Clients need to protect their use with HTTPS
  - Clients can handover tokens to others



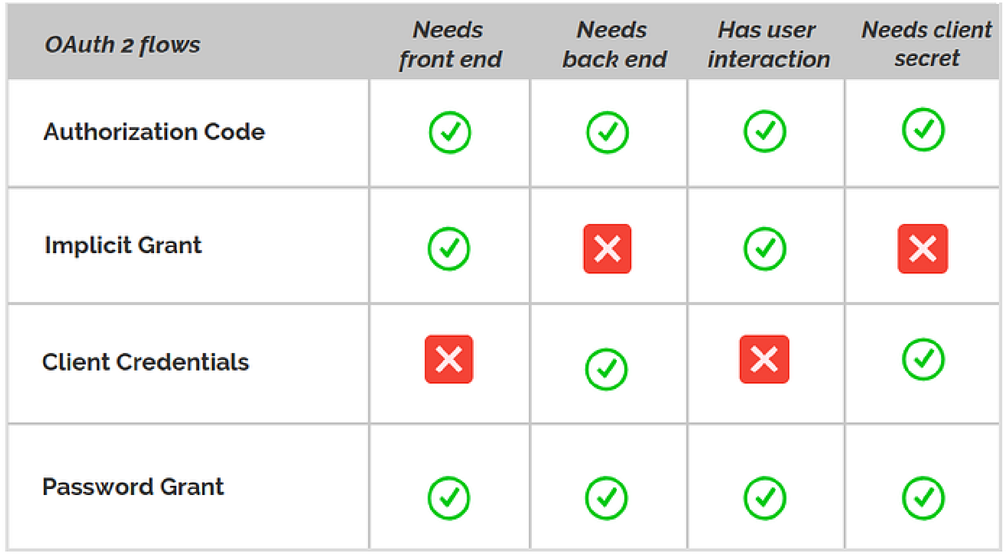
OAuth tokens: Refresh token

- Created by an OAuth server
  - When creating an access token
  - An opaque byte blob that makes sense only to its issuer
  - It can be used to collect a new access token
    - Still requiring the client authentication
- Bearer tokens
  - Clients need to protect their use with HTTPS
  - Clients can handover tokens to others



OAuth flows

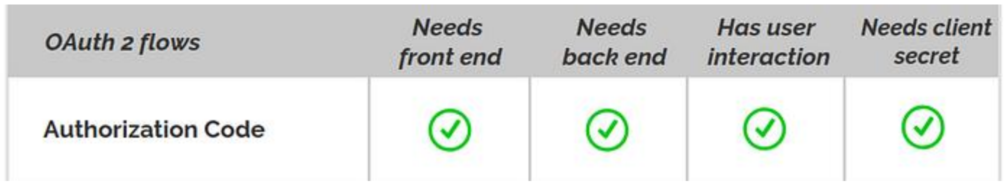
- ▷ Authorization code flow
  - 3-legged OAuth
  - Default OAuth flow
  - The most secure
- ▷ Implicit flow (grant)
- ▷ Resource owner password credentials flow
- ▷ Client credentials flow
  - 2-legged flow



OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Authorization Code	✓	✓	✓	✓
Implicit Grant	✓	✗	✓	✗
Client Credentials	✗	✓	✗	✓
Password Grant	✓	✓	✓	✓

Authorization code flow

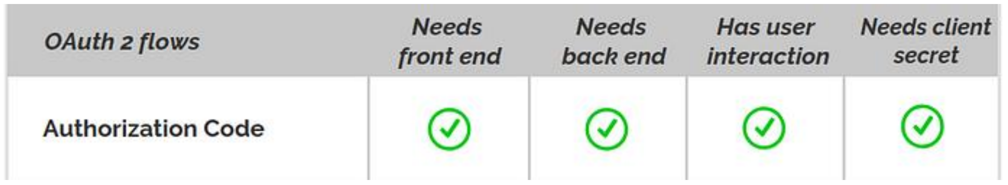
- 3-legged OAuth
  - Enables checking the identity of the 3 involved actors
- OAuth server authenticates the resource owner
  - Username + password or other means
- OAuth server authenticates the client
  - ClientID + ClientSecret + HTTP basic authorization
- Client authenticates the OAuth server
  - Certificate + URL



OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Authorization Code	✓	✓	✓	✓

Authorization code flow

- Requirements
  - Confidential application types
  - Secure storage for tokens, ClientID and ClientSecret
- Setup
  - Client registration in the OAuth server
    - Client receives ClientID and ClientSecret
    - Not regulated by OAuth

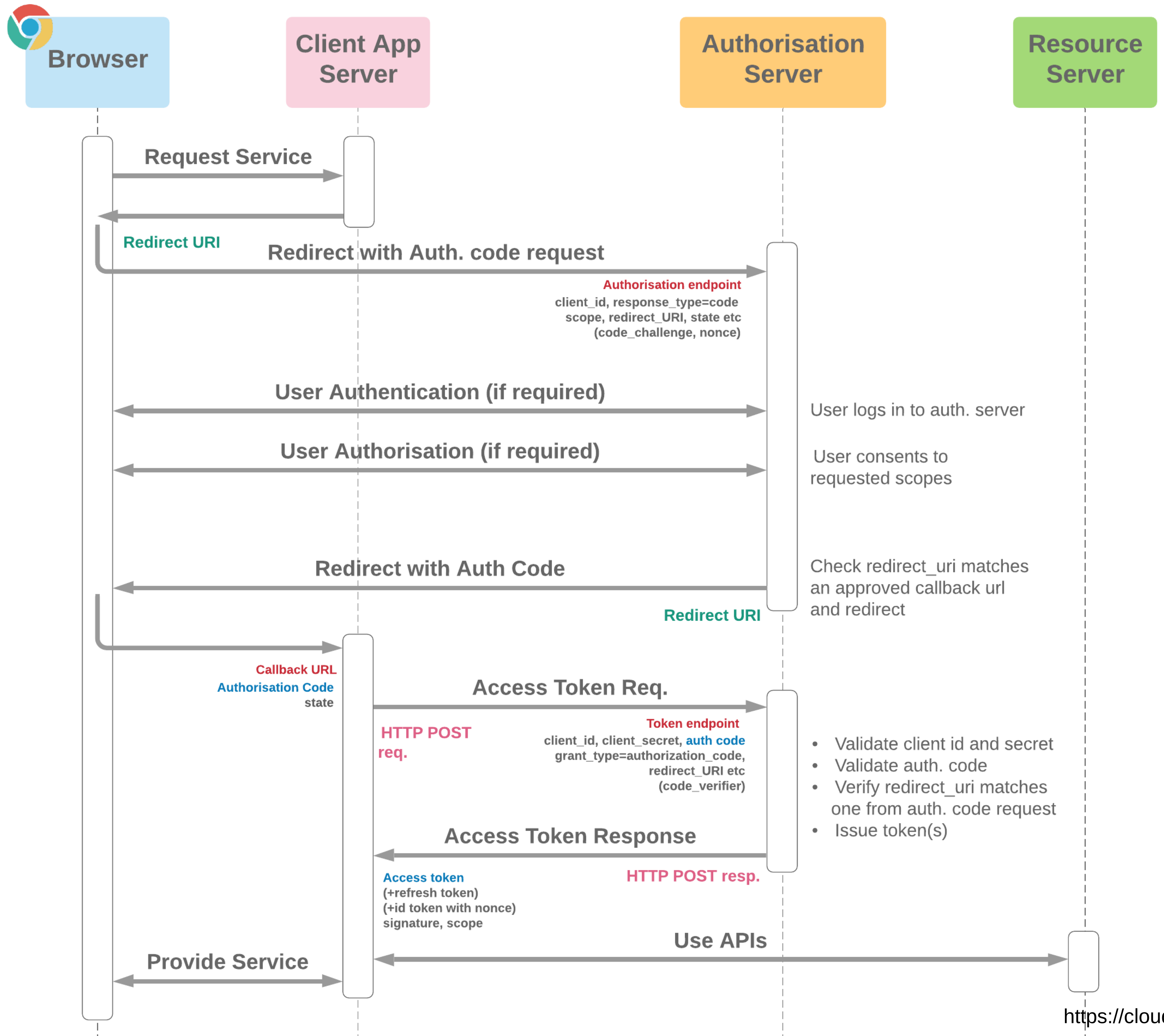


OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Authorization Code	✓	✓	✓	✓

Authorization code flow

- Resource owner uses a server-based Web App
  - The client
- The client uses the resource server API to get a resource
  - The resource server redirects the client to the OAuth server
- The OAuth server authenticates the resource owner
  - And sends an authorization grant to the client
- The client gets an access token from the OAuth server
  - Using its credentials (to have access permission)
  - Using its authorization grant
- The client uses again the resource server API to get a resource
  - This time providing an access token

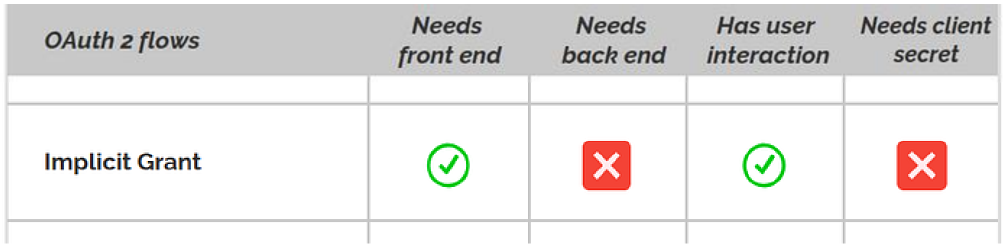
Aut



https://cloudsundial.com/salesforce-oauth-flows

Implicit flow

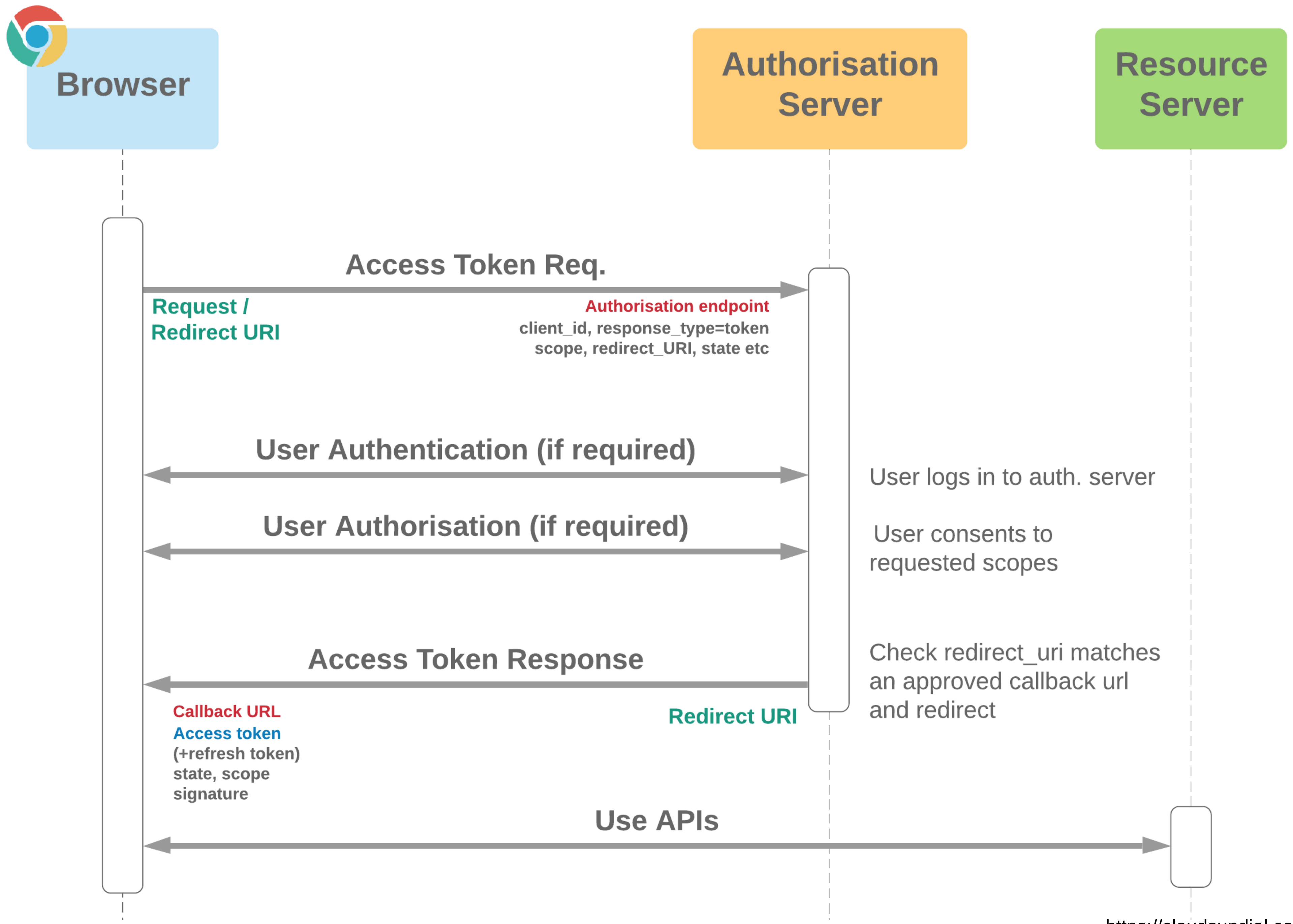
- Requirements
  - Public application types
- Setup
  - Client registration in the OAuth server
    - Client receives ClientID
    - Not regulated by OAuth
- Limitations
  - No client authentication
  - No refresh tokens



OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Implicit Grant	✓	✗	✓	✗

Implicit flow

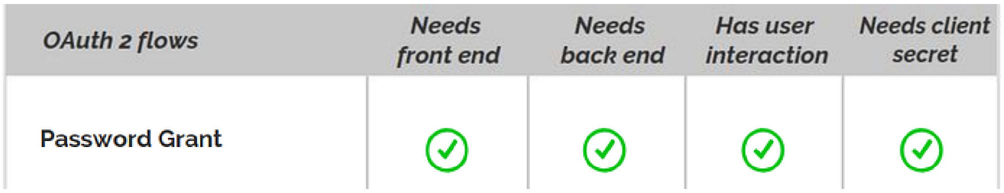
- Resource owner uses a mobile or client-based Web App
  - The client
- The client uses the resource server API to get a resource
  - The resource server redirects the client to the OAuth server
- The OAuth server authenticates the resource owner
  - And sends an access token to the client
- The client uses again the resource server API to get a resource
  - This time providing an access token


Browser
Authorisation Server
Resource Server
Access Token Req.
Request / Redirect URI
Authorisation endpoint
client_id, response_type=token
scope, redirect_URI, state etc
User Authentication (if required)
User Authorisation (if required)
User logs in to auth. server
User consents to requested scopes
Access Token Response
Check redirect_uri matches an approved callback url and redirect
Callback URL
Access token
(+refresh token)
state, scope
signature
Redirect URI
Use APIs

https://cloudsundial.com/salesforce-oauth-flows

Resource owner password flow

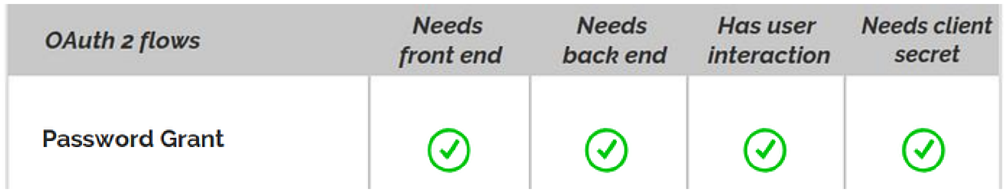
- Requirements
  - Confidential application types
  - Sharing of resource owner credentials with client applications
  - Secure storage for tokens, ClientID and ClientSecret
- Setup
  - Client registration in the OAuth server
    - Client receives ClientID and ClientSecret
    - Not regulated by OAuth
- Limitations
  - Resource owners need to trust on client applications



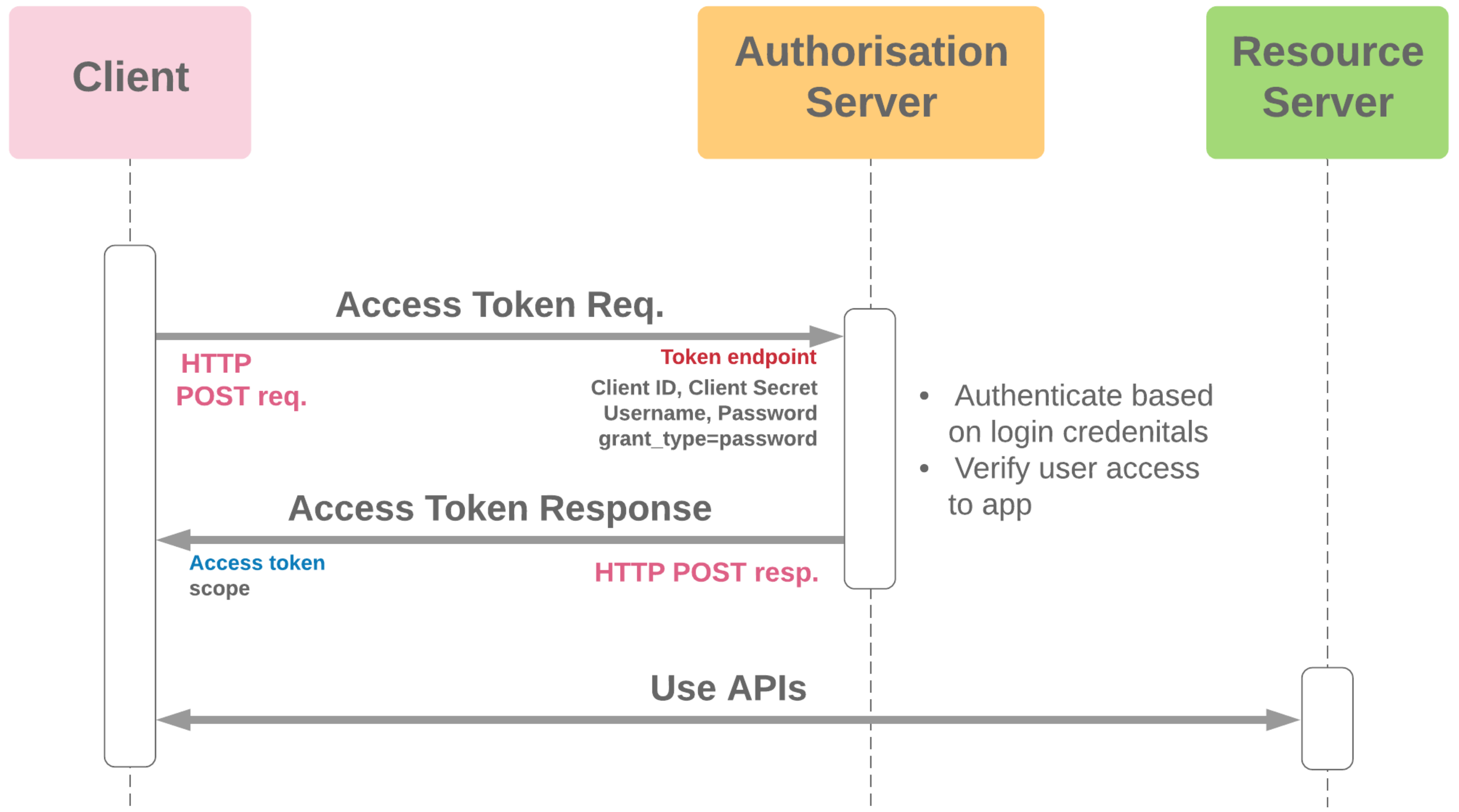
OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Password Grant	✓	✓	✓	✓

Resource owner password flow

- ▷ Resource owner uses a server-based Web App
  - The client
- ▷ The client uses the resource server API to get a resource
  - The resource server requests a token
- ▷ The client asks the resource owner for authentication credentials
- ▷ The client gets an access token from the OAuth server
  - Using its credentials (to have access permission)
  - Using the resource owner's credentials
  - These should be immediately discarded
- ▷ The client uses again the resource server API to get a resource
  - This time providing an access token

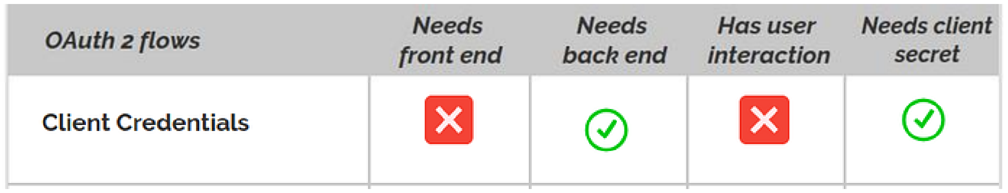


OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Password Grant	✓	✓	✓	✓


Client
Authorisation Server
Resource Server
Access Token Req.
HTTP POST req.
Token endpoint
Client ID, Client Secret
Username, Password
grant_type=password
• Authenticate based on login credentitals
• Verify user access to app
Access Token Response
Access token
scope
HTTP POST resp.
Use APIs

Client credentials flow

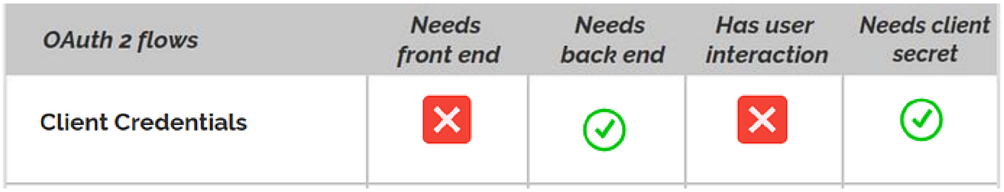
- Requirements
  - Confidential application types
  - Secure storage for tokens, ClientID and ClientSecret
- Setup
  - Client registration in the OAuth server
    - Client receives ClientID and ClientSecret
    - Not regulated by OAuth
- Limitations
  - No resource owner authentications or authorizations



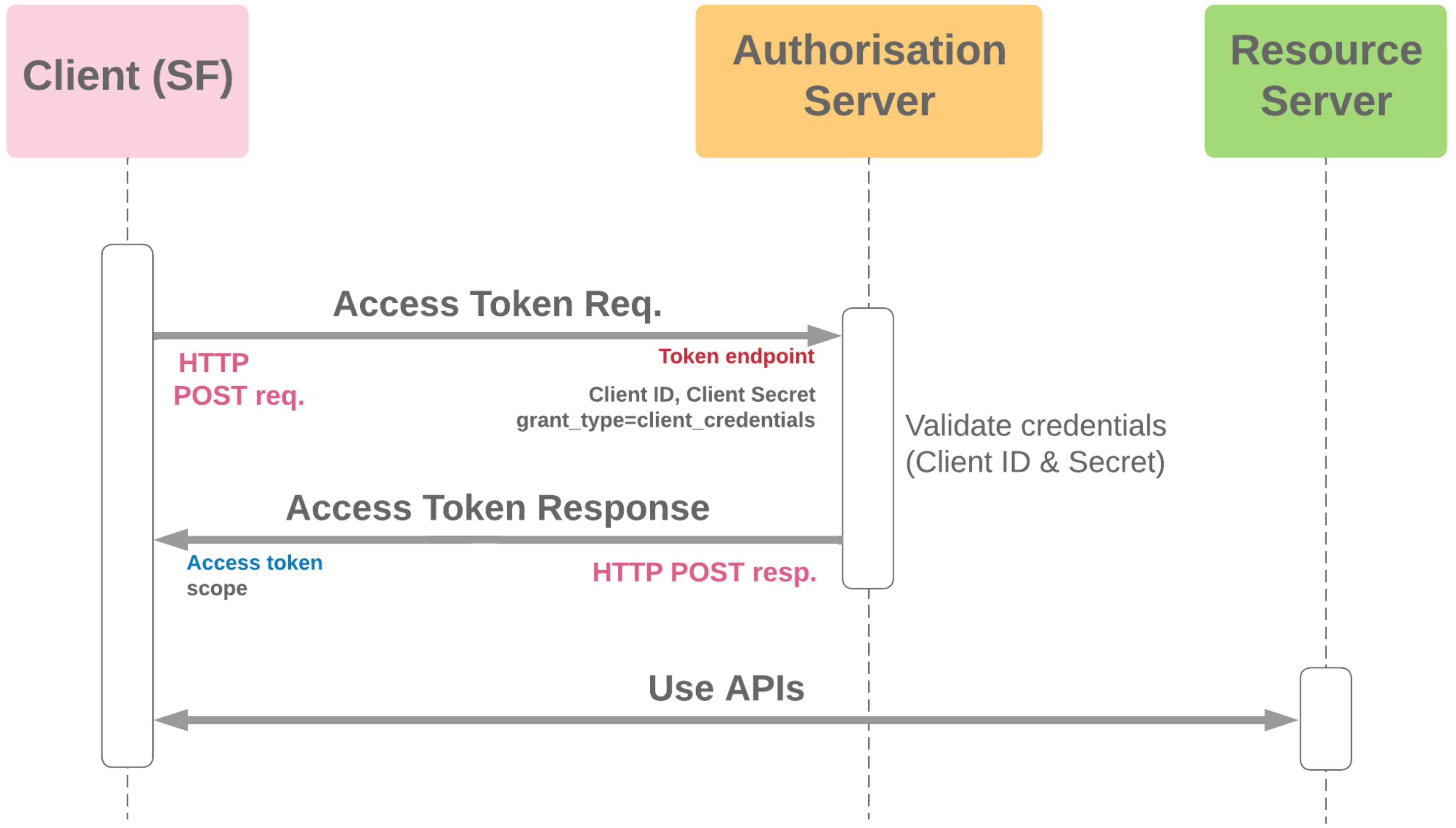
OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Client Credentials	✗	✓	✗	✓

Client credentials flow

- Resource owner uses a server-based Web App
  - The client
- The client uses the resource server API to get a resource
  - The resource server requests a token
- The client gets an access token from the OAuth server
  - Using its credentials (to have access permission)
- The client uses again the resource server API to get a resource
  - This time providing an access token



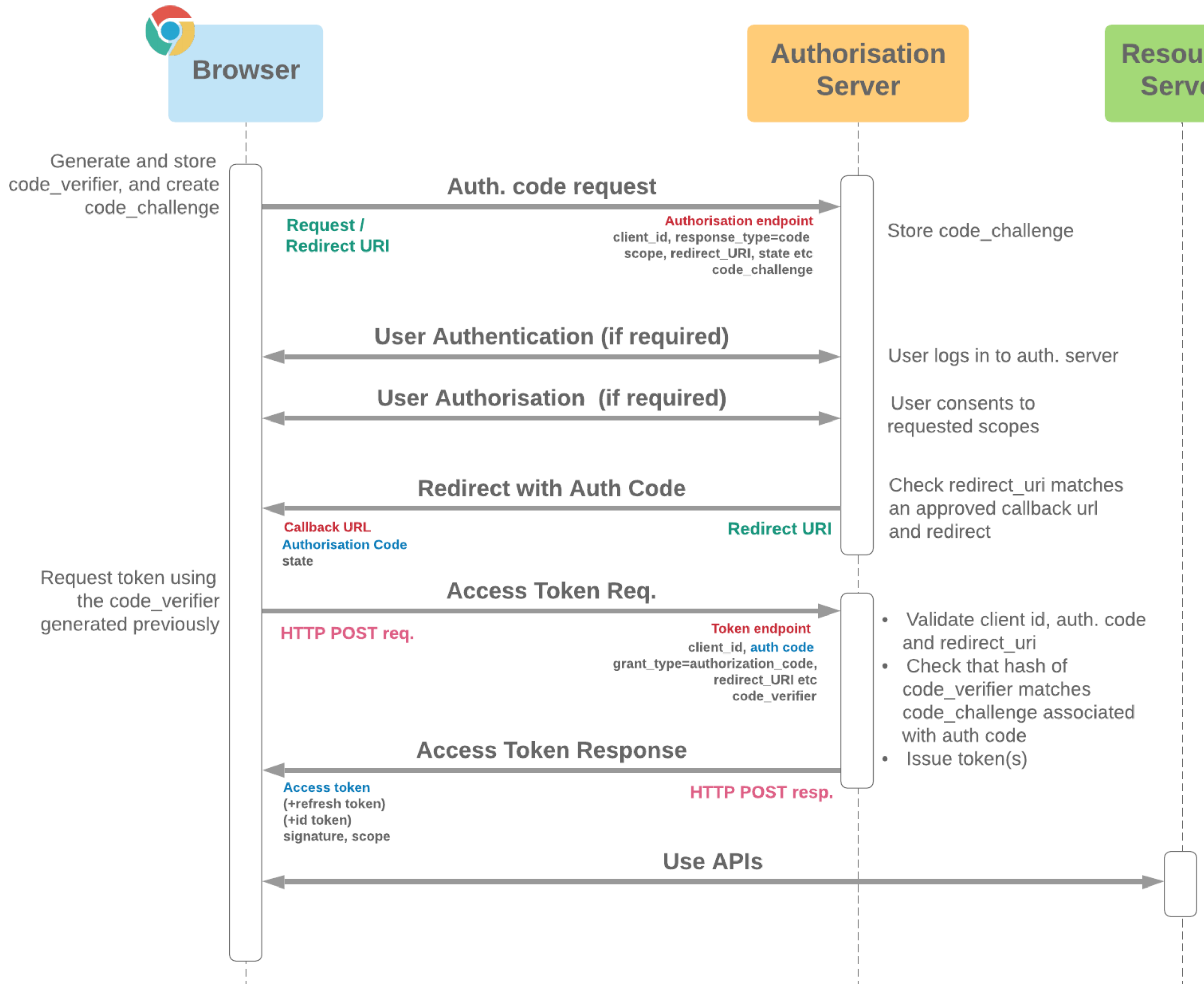
OAuth 2 flows	*Needs front end*	*Needs back end*	*Has user interaction*	*Needs client secret*
Client Credentials	✗	✓	✗	✓


Client (SF)
Authorisation Server
Resource Server
Access Token Req.
HTTP POST req.
Token endpoint
Client ID, Client Secret
grant_type=client_credentials
Validate credentials (Client ID & Secret)
Access Token Response
Access token
scope
HTTP POST resp.
Use APIs

Proof Key for Code Exchange (PKCE, RFC 7636)

- Binds authorization grants to their requesters
  - Using a Code Challenge
    - A digest of a Code Verifier
    - A bit commitment
  - Cannot the used by eavesdroppers

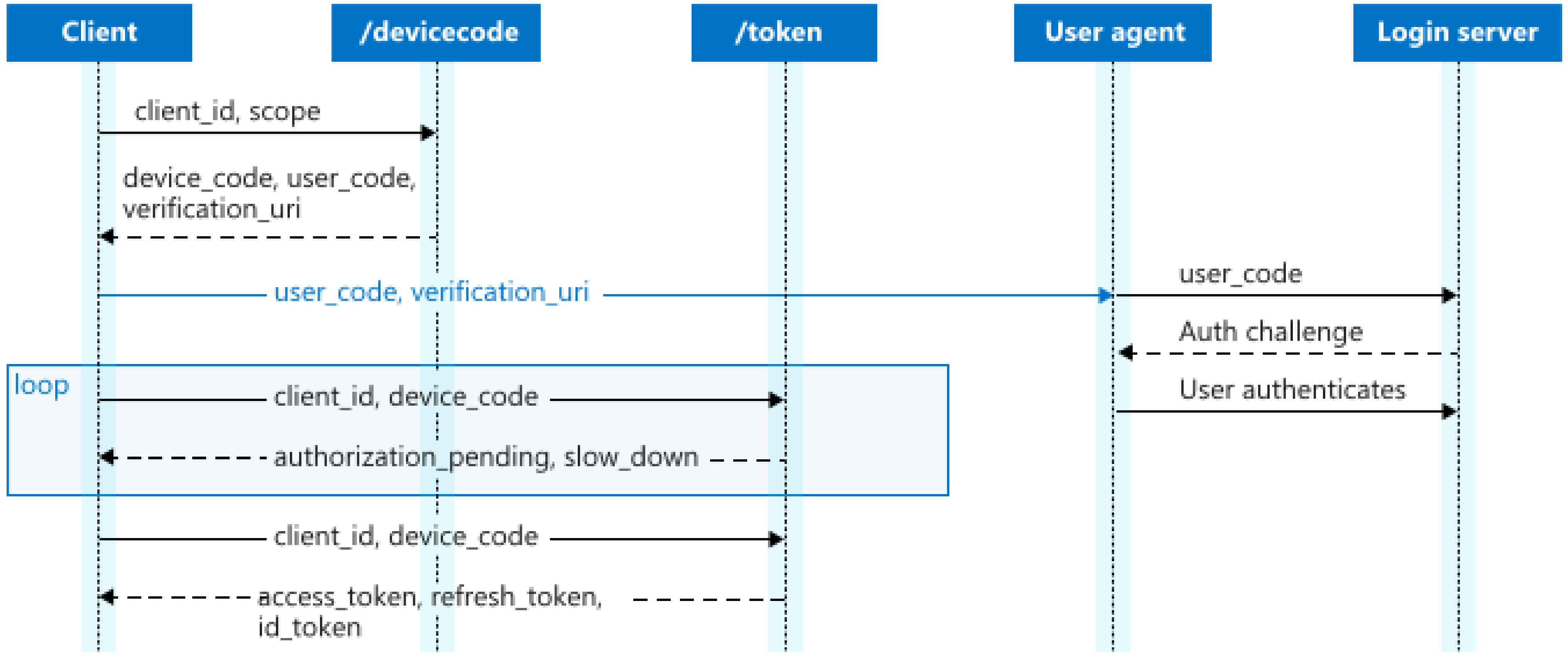
- The requester is required to demonstrate the ownership of the authorization grant when fetching the access token
  - Providing the Code Verifier


Browser
Authorisation Server
Resource Server
Generate and store code_verifier, and create code_challenge
Auth. code request
Request / Redirect URI
Authorisation endpoint
client_id, response_type=code
scope, redirect_URI, state etc
code_challenge
Store code_challenge
User Authentication (if required)
User logs in to auth. server
User Authorisation (if required)
User consents to requested scopes
Redirect with Auth Code
Check redirect_uri matches an approved callback url and redirect
Callback URL
Authorisation Code
state
Redirect URI
Request token using the code_verifier generated previously
Access Token Req.
HTTP POST req.
Token endpoint
client_id, auth code
grant_type=authorization_code,
redirect_URI etc
code_verifier
• Validate client id, auth. code and redirect_uri
• Check that hash of code_verifier matches code_challenge associated with auth code
• Issue token(s)
Access Token Response
Access token
(+refresh token)
(+id token)
signature, scope
HTTP POST resp.
Use APIs

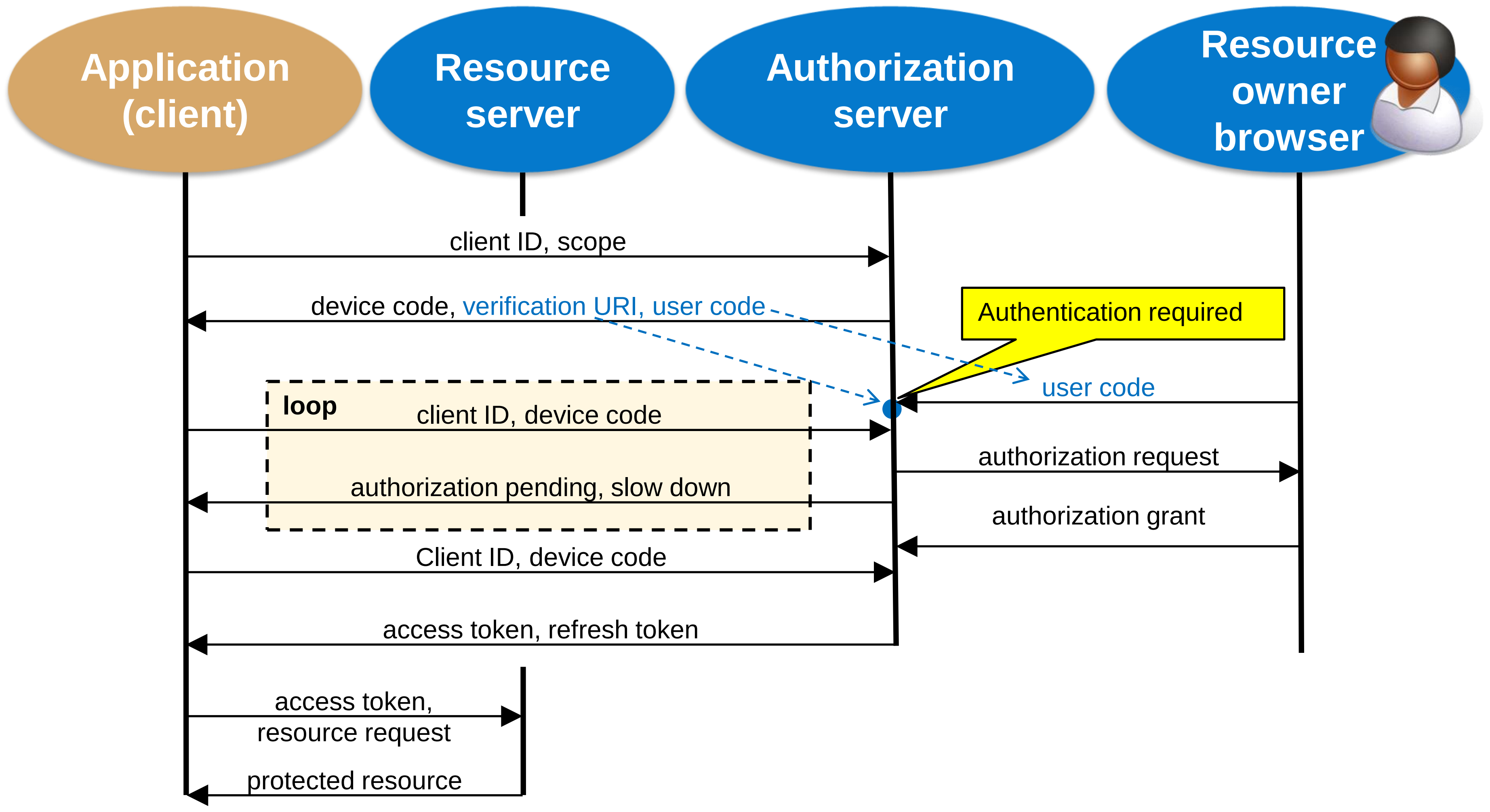
Device authorization grant (RFC 8628)

- In some cases the user is using a device with no browser to interact with a OAuth client
  - No HTTP redirections to Authorization server and back to client
  - No user interface
    - To authenticate the user
    - To review and authorize request
- Solution
  - Use a second device to perform the user authentication and to grant the authorization
    - e.g. mobile phone, tablet, etc.
  - Client fetches the access token from the Authorization server
    - Possibly with a refresh token

Device authorization grant (RFC 8628)



Actual protocol flow


Application (client)
Resource server
Authorization server
Resource owner browser
client ID, scope
device code, verification URI, user code
Authentication required
user code
loop
client ID, device code
authorization pending, slow down
authorization request
authorization grant
Client ID, device code
access token, refresh token
access token, resource request
protected resource