OAuth in Detail

Scott Brady
IDENTITY & ACCESS CONTROL LEAD

@scottbrady91 www.scottbrady91.com

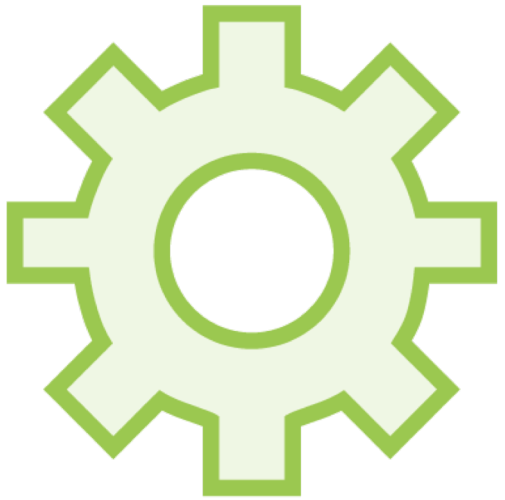
Overview

Understand each authorization grant type and when to use it

Know the different ways to receive response data as a client

Protocol Endpoints

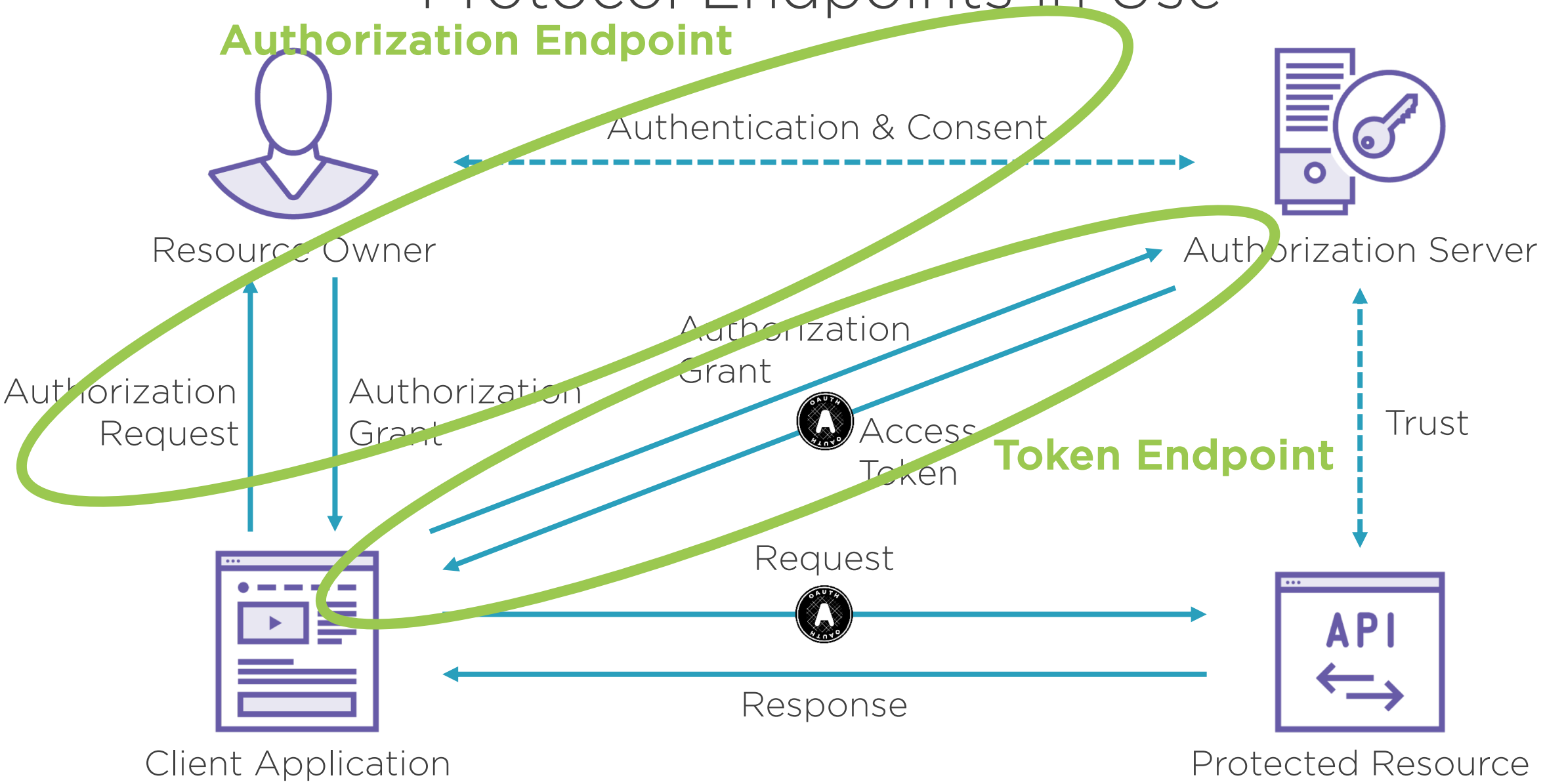
Authorization



Token

Must use Transport Layer Security

Protocol Endpoints in Use



OAuth Scope

A permission to do something within a protected resource on behalf of the resource owner

Scopes

pluralsight_api

pluralsight_api.read

pluralsight_api.feed

Scope Naming

pluralsight_api.read

Read access to just one API

read

Read access to all APIs?

The Authorization Code Grant Type

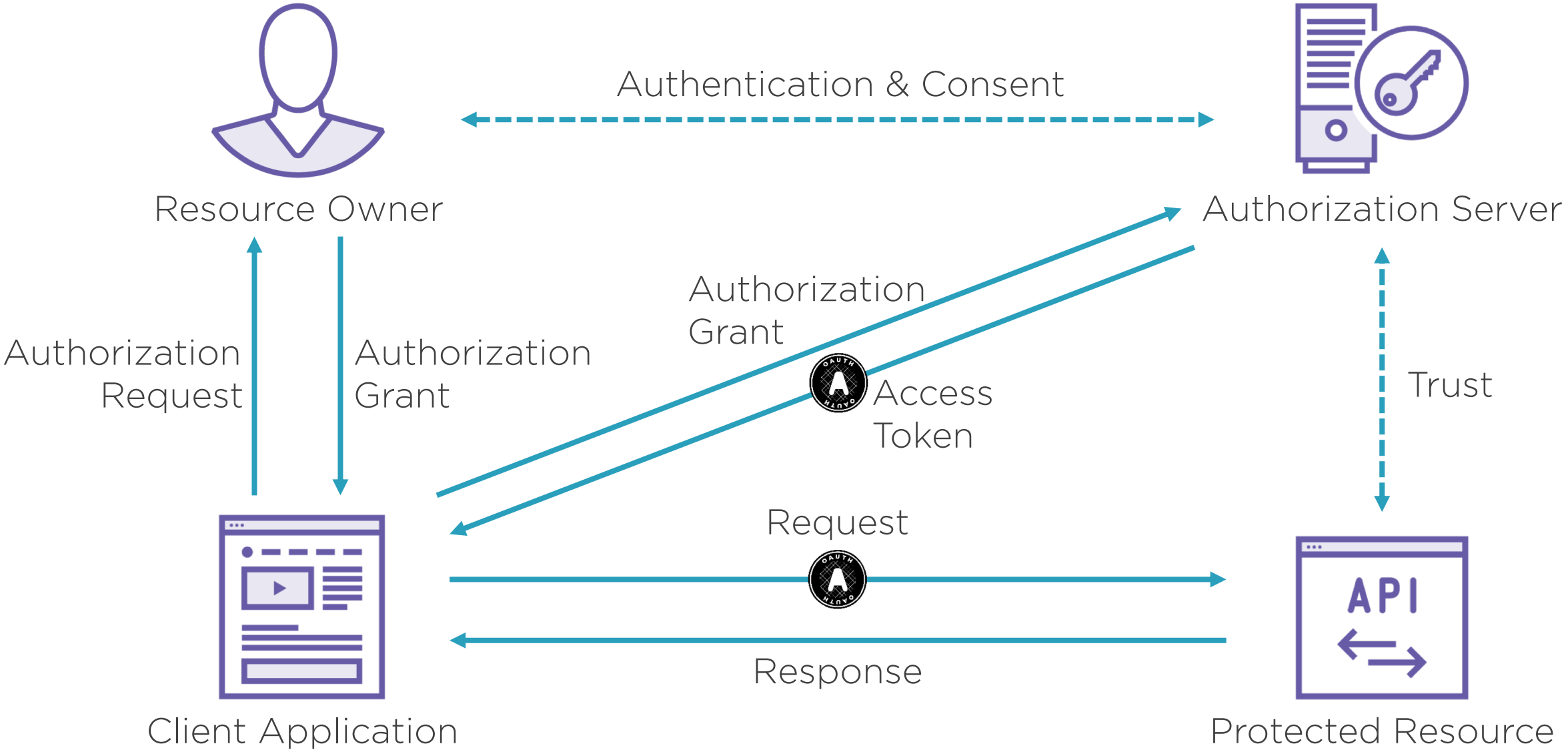
The Authorization Code Grant Type

Designed for “confidential clients”

Best for websites with a server back end

Explicit user & client authentication

The Authorization Code Flow



Authorization Request

```
https://authserver.example.com/authorize
    ?response_type=code
    &client_id=s6BhdRkqt3
    &redirect_uri=https://client.example.com/callback
    &state=xyz
    &scope=api1 api2.read
```

Authorization Response

```
https://client.example.com/callback
    ?code=SplxlOBeZQQYbYS6WxSbIA
    &state=xyz
```

Token Request

```
POST /token HTTP/1.1
  Host: server.example.com
  Content-Type: application/x-www-form-urlencoded
  Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW


 grant_type=authorization_code
 &code=SplxlOBeZQQYbYS6WxSbIA
 &redirect_uri=https://client.example.com/cb
 &client_id=s6BhdRkqt3&client_secret=gX1fBat3bV
```

Basic Authentication & OAuth

Basic Authentication Style (RFC 7617)

```
Base64(client_id + ":" + client_secret)
```

OAuth Style (RFC 6749)

```
Base64(urlformencode(client_id) + ":" + urlformencode(client_secret))
```

Token Response

```
HTTP/1.1 200 OK
  Content-Type: application/json

{
   "access_token": "2YotnFZFEjr1zCsicMWpAA",
   "token_type": "Bearer",
   "expires_in": 3600,
   "scope": "api2.read"
}
```

Demo

Authorization code flow in action

The Implicit Grant Type

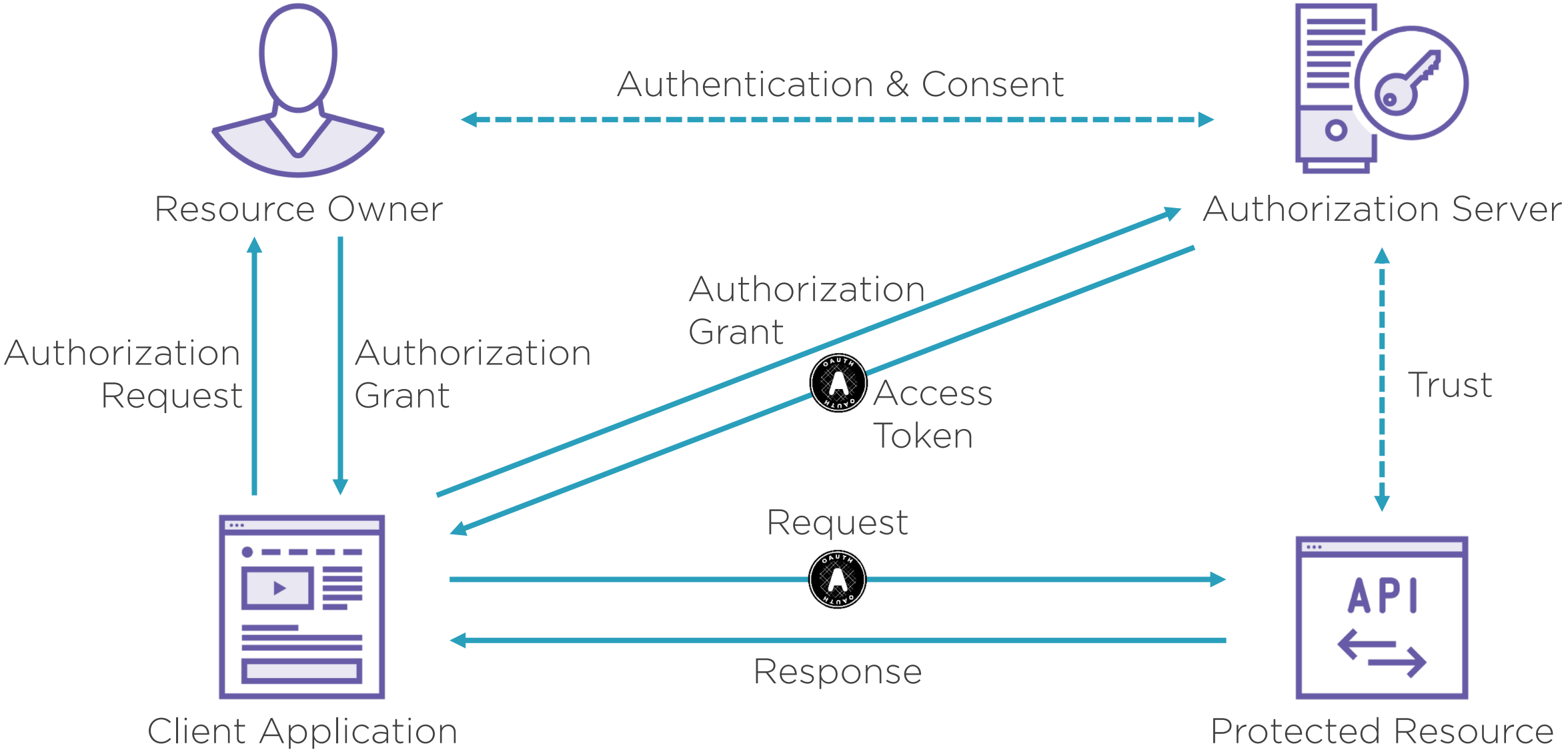
The Implicit Grant Type

Designed for “public clients”

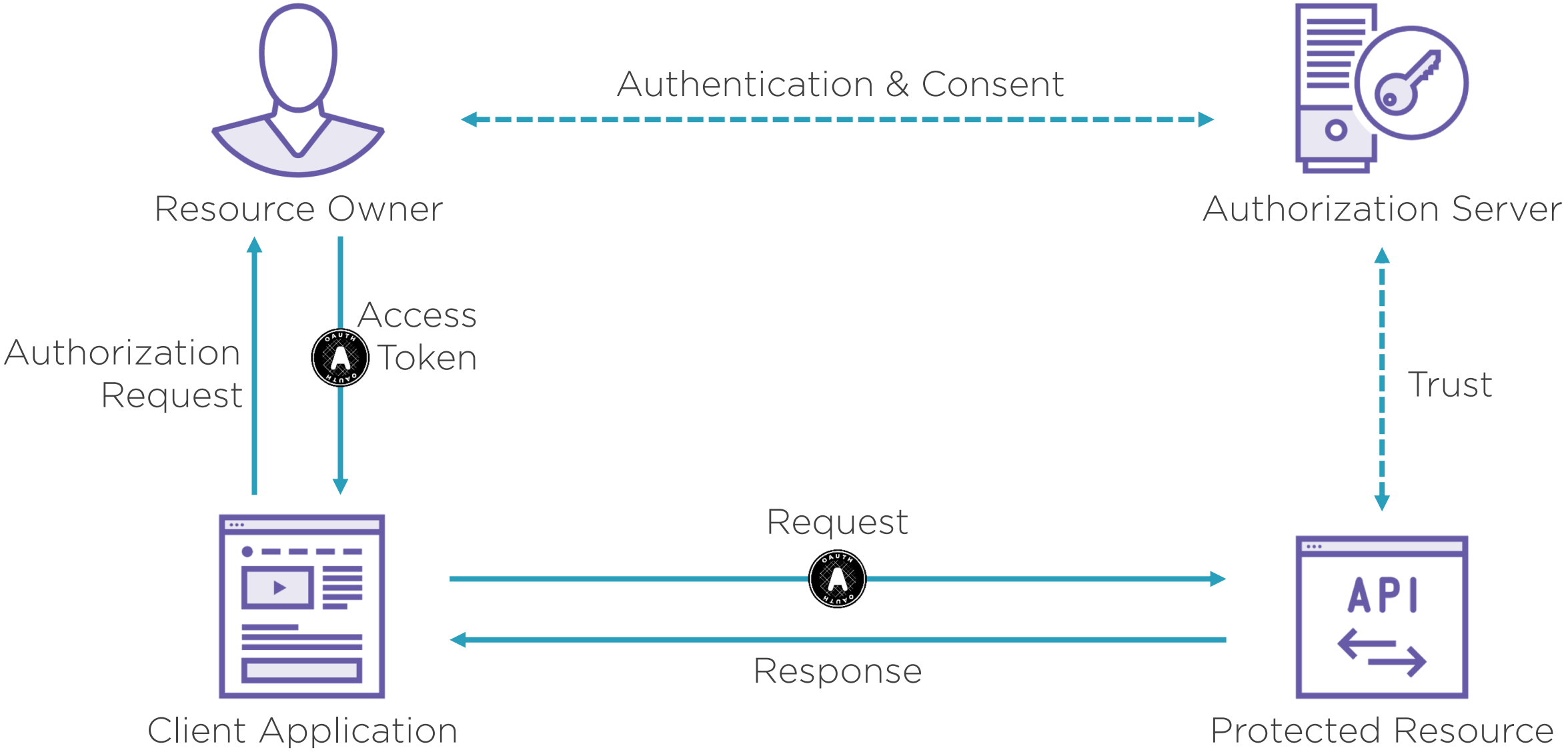
Best for clients accessing resources directly from the browser

No explicit client authentication

The Implicit Flow



The Implicit Flow



Authorization Request

```
https://authserver.example.com/authorize
    ?response_type=token
    &client_id=s6BhdRkqt3
    &redirect_uri=https://client.example.com/callback
    &state=xyz
    &scope=api1 api2.read
```

Authorization Response

```
https://client.example.com/callback
    #access_token=2YotnFZFEjr1zCsicMWpAA
    &token_type=example
    &expires_in=3600
    &state=xyz
```

Security Concerns

Access tokens exposed to resource owner

Access tokens accessible to 3rd party JavaScript

No validation that access tokens are intended for client

Demo

Implicit flow in action

The Client Credentials Grant Type

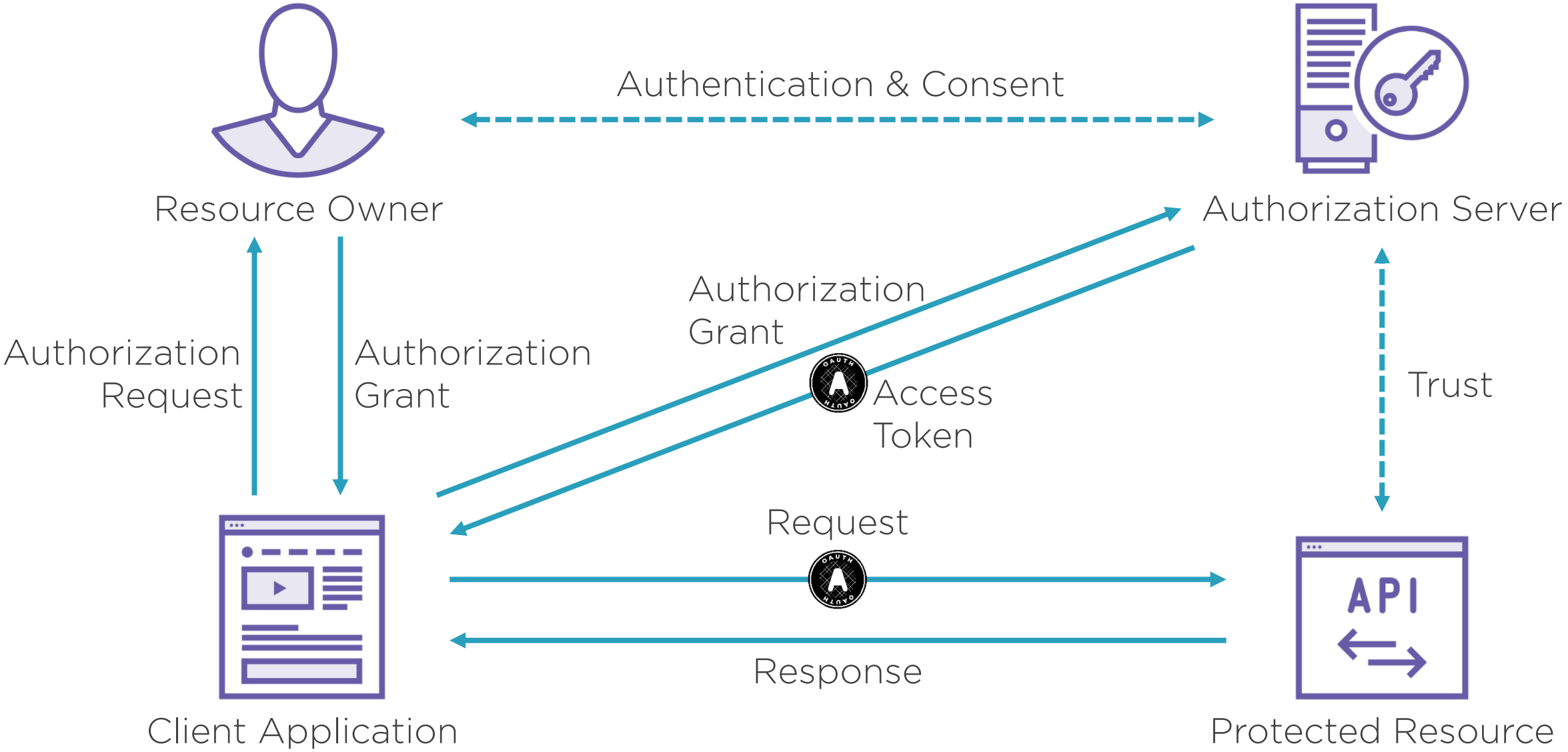
The Client Credentials Grant Type

Designed for client applications who are the resource owner

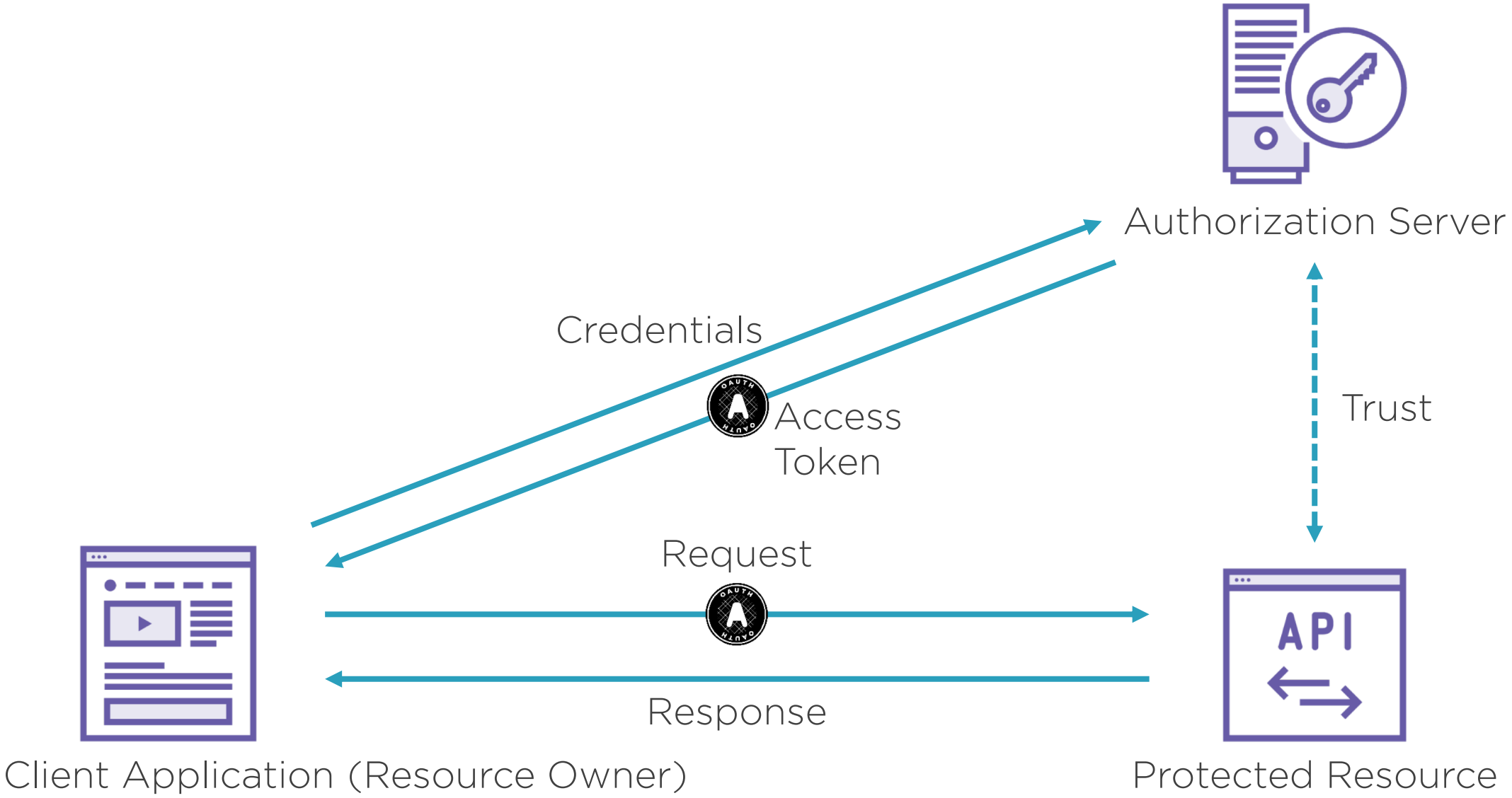
Best for machine-to-machine communication

Requires client authentication

The Client Credentials Flow



The Client Credentials Flow



Token Request

```
POST /token HTTP/1.1
  Host: server.example.com
  Content-Type: application/x-www-form-urlencoded
  Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

 grant_type=client_credentials
 &scope=api1 api2.read
```

Token Response

```
HTTP/1.1 200 OK
  Content-Type: application/json

{
   "access_token": "2YotnFZFEjr1zCsicMWpAA",
   "token_type": "Bearer",
   "expires_in": 3600,
   “scope": "api2.read"
}
```

Differences with API Keys/HTTP Basic Authentication

Not sending the credentials with every request

Access tokens

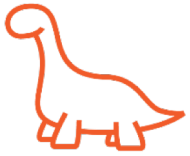
Timed access without manual input

Demo

Client credentials in action

The Resource Owner Password Credentials (ROPC) Grant Type

The Client Credentials Grant Type



Designed as a stop-gap for legacy applications

Negates most of the benefits of OAuth

Should no longer be used

Token Request

```
POST /token HTTP/1.1
  Host: server.example.com
  Content-Type: application/x-www-form-urlencoded
  Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

 grant_type=password
 &username=johndoe
 &password=A3ddj3w
 &scope=api1 api2.read
```

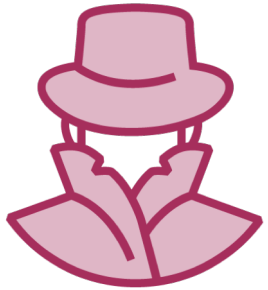
Token Response

```
HTTP/1.1 200 OK
  Content-Type: application/json

{
   "access_token": "2YotnFZFEjr1zCsicMWpAA",
   "token_type": "Bearer",
   "expires_in": 3600,
   “scope": "api2.read"
}
```

Do Not Use This!

The Problems with ~~Credential Sharing~~ Resource Owner Password Credentials



Impersonation

Revocation

Exposed user credentials

Something you know

Federation

Incompatibilities

scottbrady91.com/OAuth

Demo

Resource owner password credentials in action

Refresh Tokens

Refresh Tokens

Swap for new tokens

Allows for long-lived access

Highly confidential

User should be informed

Authorization Request

```
https://authserver.example.com/authorize
    ?response_type=code
    &client_id=s6BhdRkqt3
    &redirect_uri=https://client.example.com/callback
    &state=xyz
    &scope=api1 api2.read offline_access
```

Token Response

```
HTTP/1.1 200 OK
  Content-Type: application/json

{
   "access_token": "2YotnFZFEjr1zCsicMWpAA",
   "token_type": "Bearer",
   "expires_in": 3600,
   "refresh_token": "tGzv3JOkF0XG5Qx2TlKWIA",
   "scope": "api2.read offline_access"
}
```

Refresh Token Request

```
POST /token HTTP/1.1
  Host: server.example.com
  Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
  Content-Type: application/x-www-form-urlencoded

 grant_type=refresh_token
 &refresh_token=tGzv3JOkF0XG5Qx2TlKWIA
 &scope=api1
```

Refresh Token Response

```
HTTP/1.1 200 OK
  Content-Type: application/json

{
    "access_token": "5BaFFasdfrje2aCsicoiJefn",
    "token_type": "Bearer",
    "expires_in": 3600,
    "refresh_token": "9oih321s0i2xnoutnsglr",
    "scope": "api2.read offline_access"
}
```

Who Can Use Refresh Tokens?

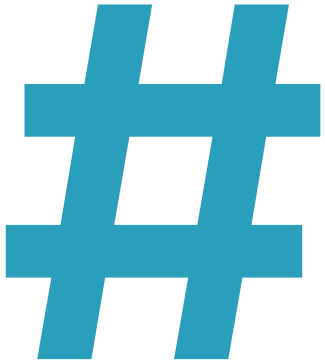
Authorization Flow	Can request `offline_access`
Authorization Code	Yes
Implicit	No
Client Credentials	No
ROPC	Yes

Demo

Refresh tokens in action

Response Modes

Query String



Hash Fragment

Form Post

Form Post HTML Example

```
<html>
   <head><title>Submit This Form</title></head>
   <body onload="javascript:document.forms[0].submit()">
    <form method="post" action="https://client.example.org/callback">
      <input type="hidden" name="state" value="xyz"/>
      <input type="hidden" name="code" value="SplxlOBeZQQYbYS6WxSbIA"/>
    </form>
   </body>
  </html>
```


When Things Go Wrong

- error
- error_description
- error_uri
- state

Authorization Error Response

```
https://authserver.example.com/authorize
    ?error=invalid_request
```

Token Error Response

```
HTTP/1.1 400 Bad Request
  Content-Type: application/json;charset=UTF-8
  Cache-Control: no-store
  Pragma: no-cache


{
    "error":"invalid_request"
}
```

Error Types

- invalid_request
- invalid_client (401)
- invalid_grant
- unauthorized_client
- unsupported_grant_type
- invalid_scope

Summary

Understand each authorization grant type and when to use it

Know the different ways to receive response data as a client

Further reading:

- RFC 6749 (OAuth 2.0)
- RFC 6819 (Threat model & security considerations)