Security – from What Used to Work to What Works Today

Kevin Dockx

@KevinDockx | http://blog.kevindockx.com/

A Few Important Definitions

Authentication establishes the identity of a user

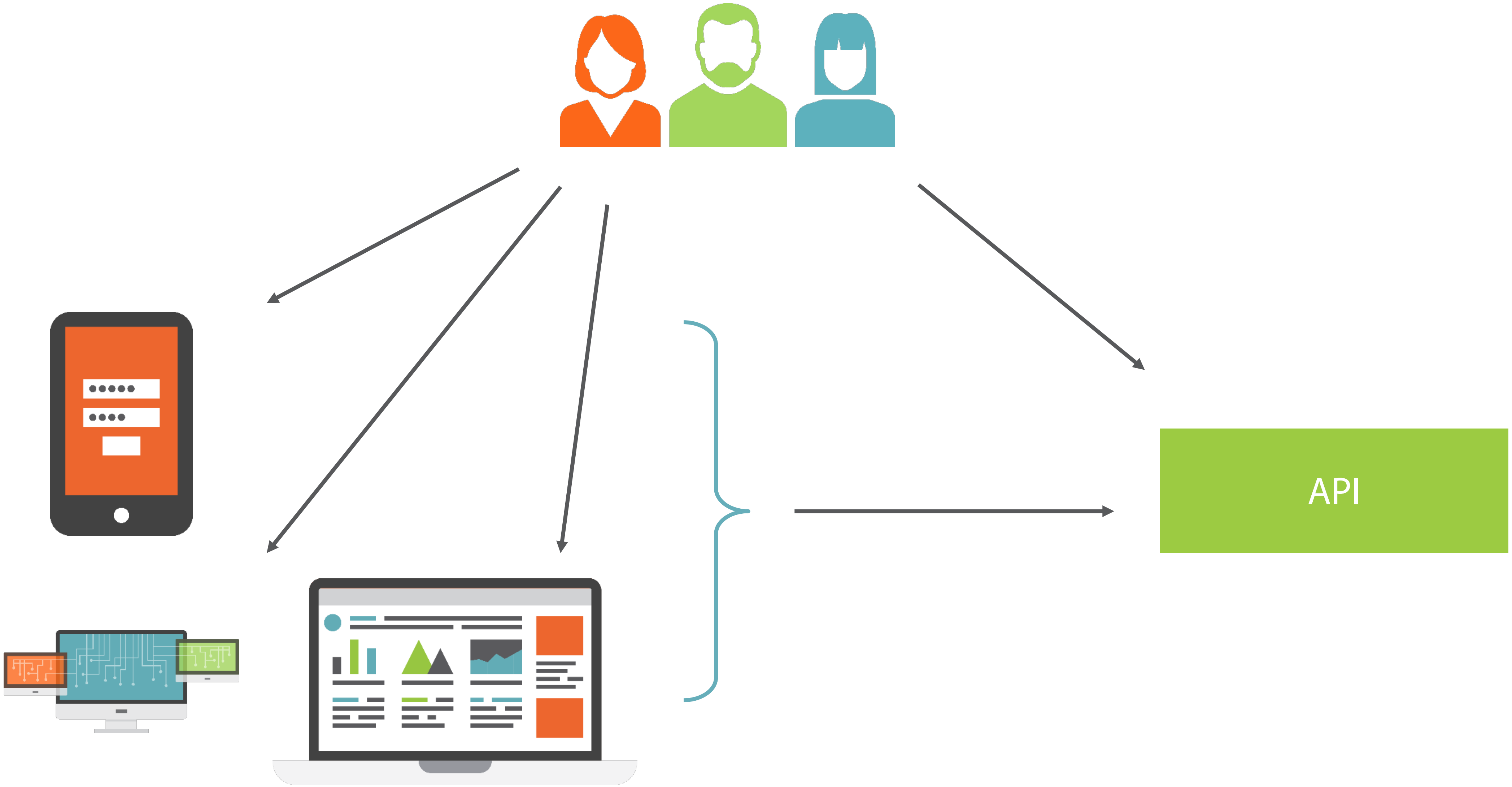
Authorization helps control what a user can or cannot access

Encryption helps protect data as it passes over the wire

Always SSL

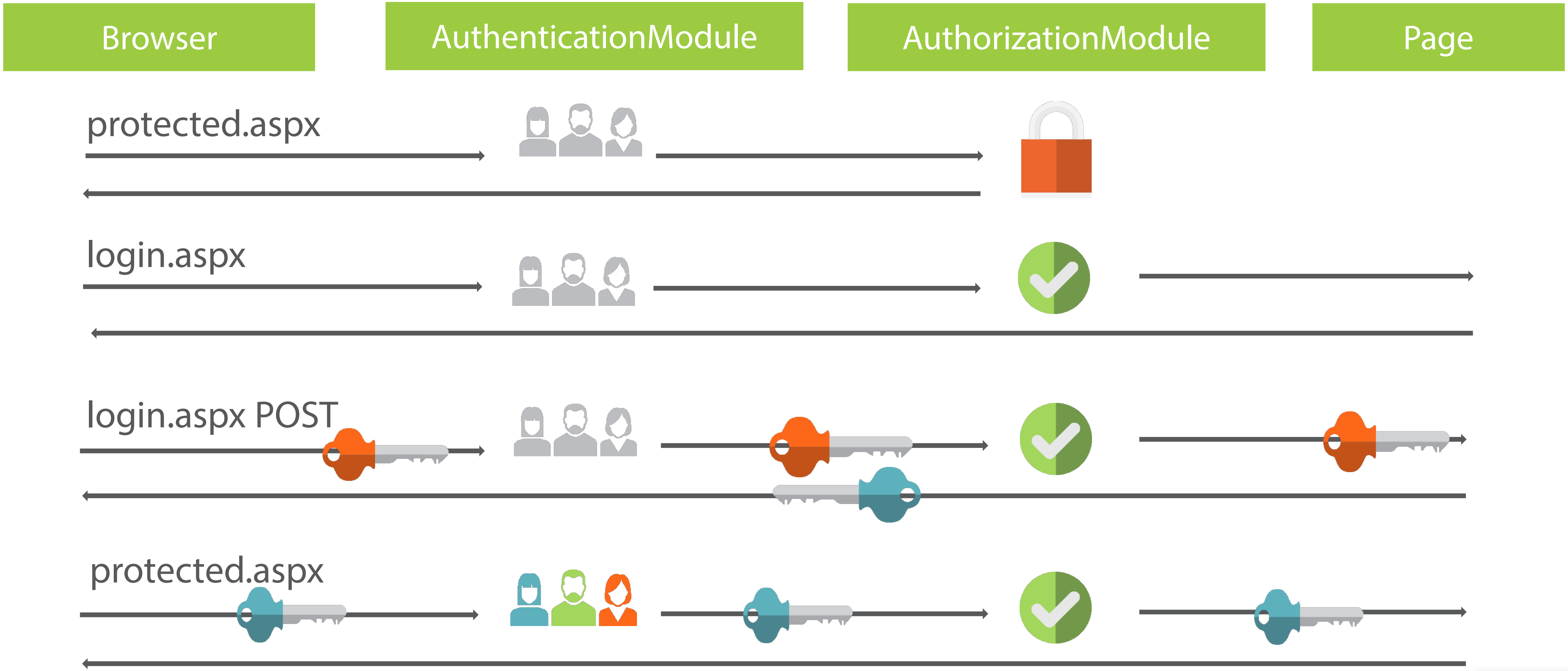
-

ALWAYS


API

The thing with security is that a lot of approaches will work, but most are not a good idea

An Old, Familiar Approach



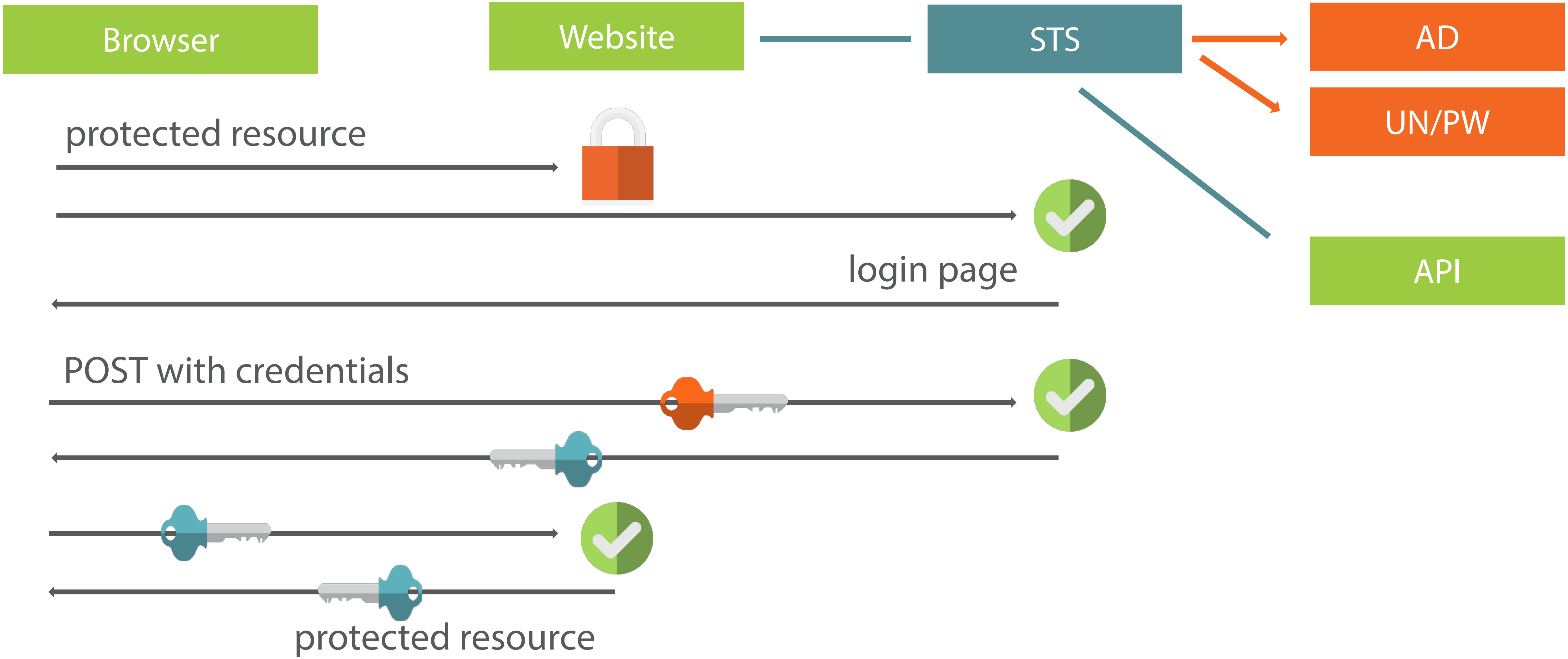
An Old, Familiar, yet Insufficient Approach

- We're not in a web-only world anymore
- Our API is still free for all

- Our application shouldn't be responsible for authentication!

Federated security allows for clean separation between the service a client is accessing and the associated authentication and authorization procedures

A Better Approach



A Better, yet Insufficient Approach

- .NET: ws-federation, ws-security, SOAP, WCF
- … but we're not living in a .NET world anymore
- We don't want our mobile clients to be part of the trusted subsystem
  - A phone = untrusted by default

OAuth 2.0 is an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications

A Few Important Definitions

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

Resource owner: an entity capable of granting access to a protected resource

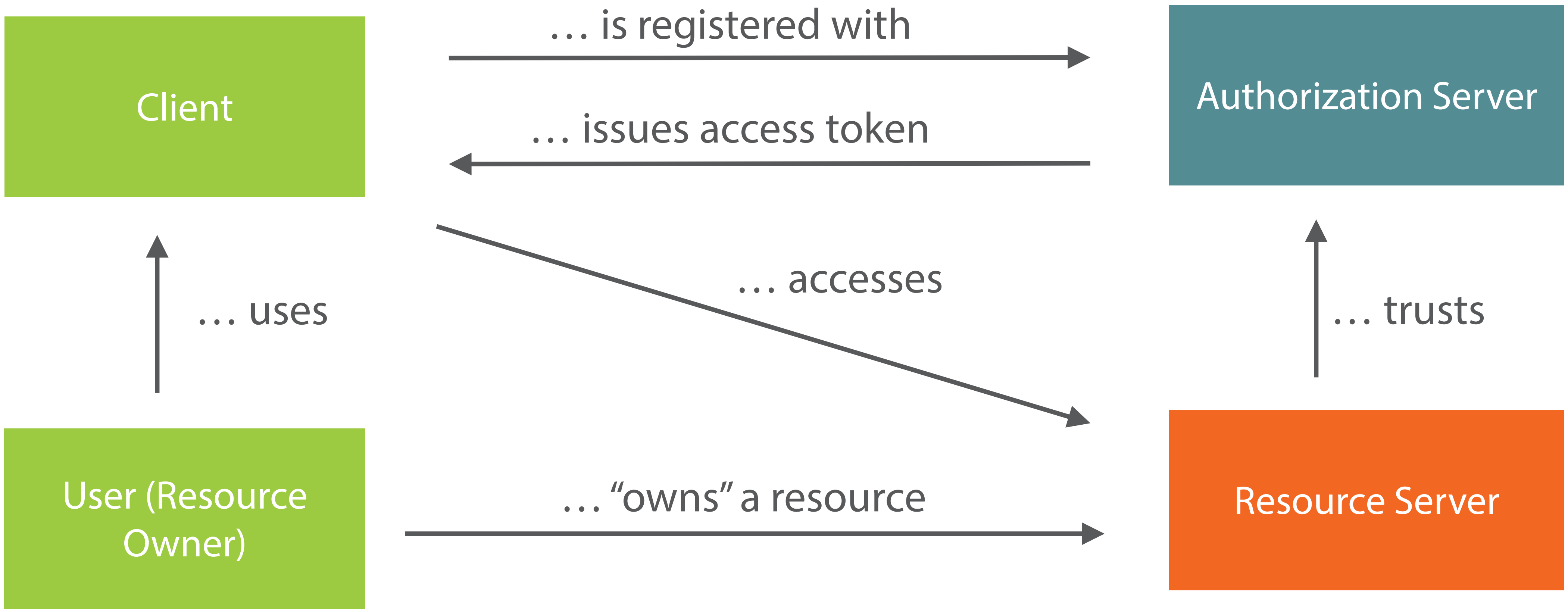
Resource server: the server hosting the protected resources

A Few Important Definitions

Client: an application making protected resource requests on behalf of the resource owner and with its authorization

Authorization server: the server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization

Interaction



Client

… is registered with

Authorization Server

… issues access token

… uses

… accesses

… trusts

User (Resource Owner)

… "owns" a resource

Resource Server

Client Types

Confidential Clients

- Clients capable of maintaining the confidentiality of their credentials
  - Web application
    - Eg: our MVC application

Public Clients

- Clients incapable of maintaining the confidentiality of their credentials
  - Native application
    - Eg: our Windows Phone Client
  - User-Agent based Application
    - Eg: javascript-based application

Protocols

Authorization Server

- Authorization endpoint
  - used by the client to obtain authorization from the resource owner via user-agent redirection
- Token endpoint
  - used by the client to exchange an authorization grant for an access token, typically with client authentication

Client

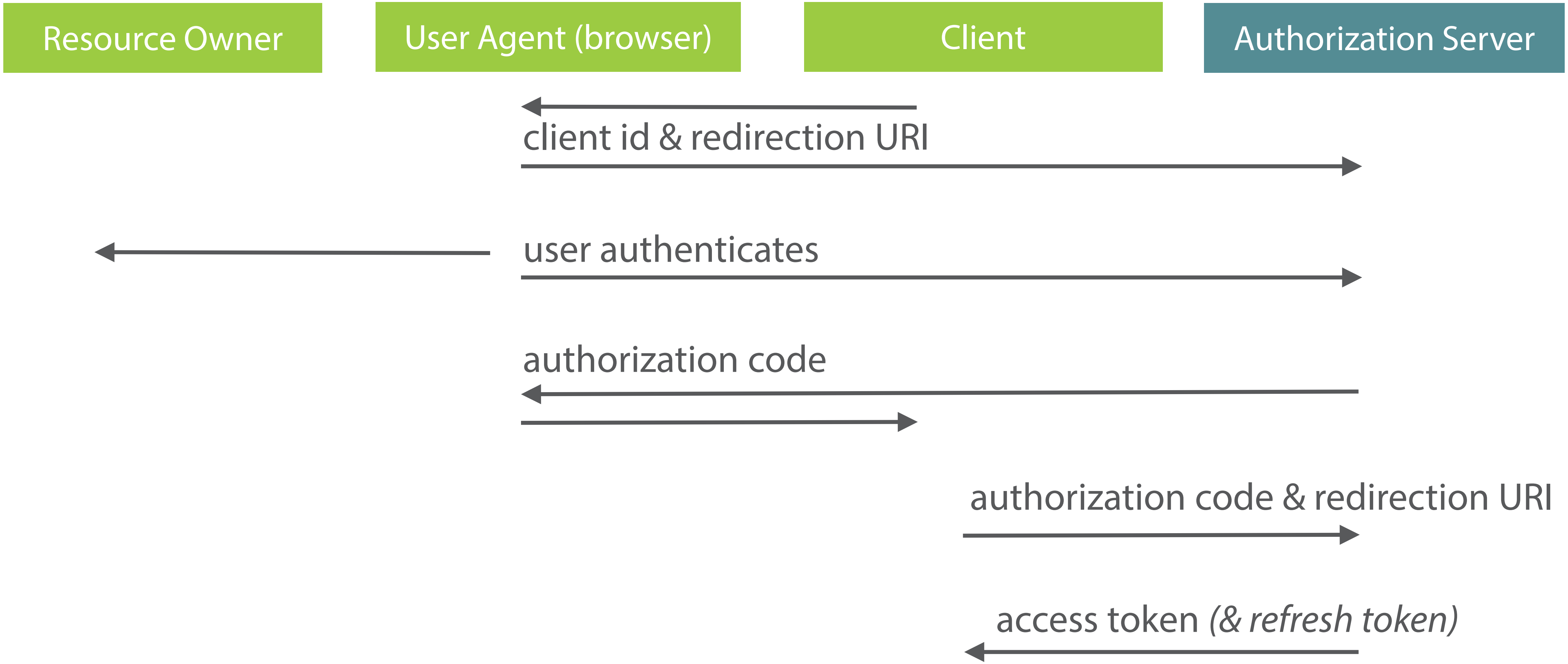
- Redirection endpoint
  - used by the authorization server to return responses containing authorization credentials to the client via the resource owner user-agent

Authorization Code Grant

- Can be used to obtain access tokens & refresh tokens
- Optimized for **confidential** clients

- Client must be able to
  - Interact with the resource owner's user-agent (browser)
  - Receive incoming requests (via redirection) from the authorization server

Authorization Code Grant



Resource Owner

User Agent (browser)

Client

Authorization Server

client id & redirection URI

user authenticates

authorization code

authorization code & redirection URI

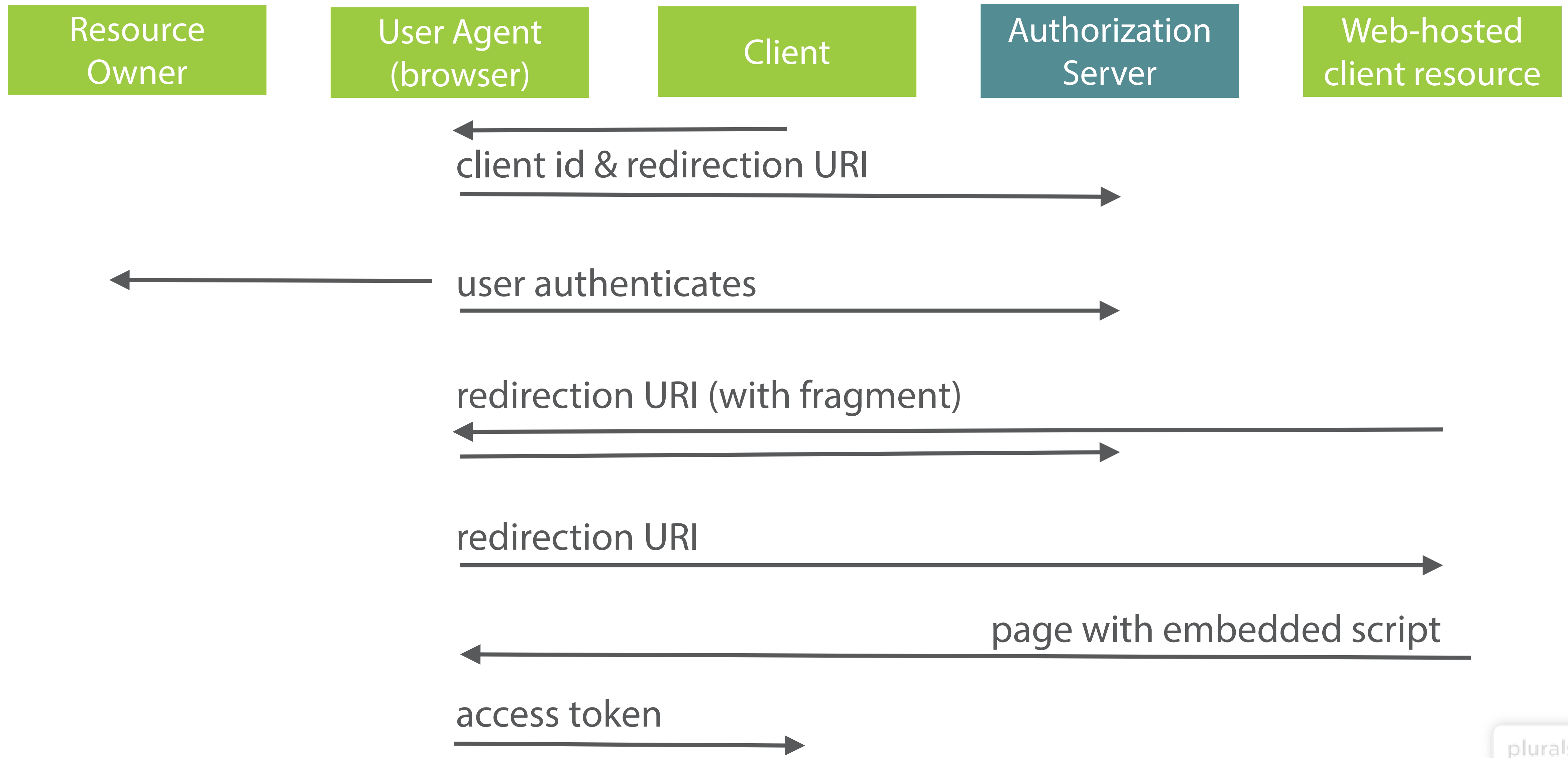
access token *(& refresh token)*

Implicit Grant

- Can be used to obtain access tokens, but no refresh tokens
- Optimized for **public** clients
- The client gets the access token as result of the authorization request (rather than making separate requests like in the authorization code flow)

- Client must be able to
  - Interact with the resource owner's user-agent (browser)
  - Receive incoming requests (via redirection) from the authorization server

Implicit Grant



Resource Owner

User Agent (browser)

Client

Authorization Server

Web-hosted client resource

client id & redirection URI

user authenticates

redirection URI (with fragment)

redirection URI

page with embedded script

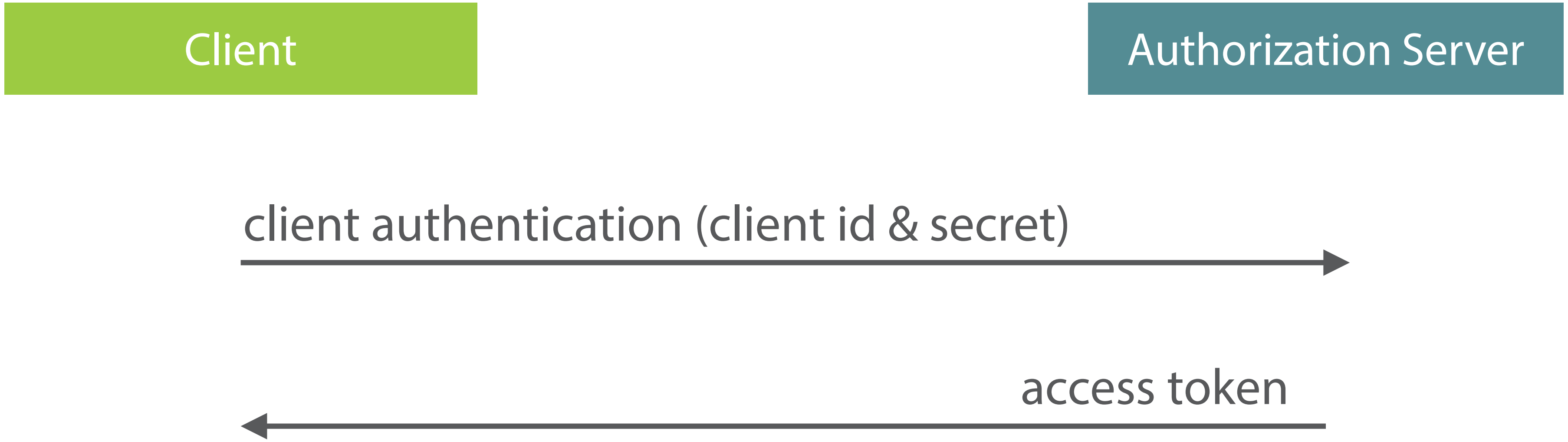
access token

Client Credentials Grant

- Can be used to obtain access tokens using Client Credentials (id, secret)
- Must only be used by **confidential** clients

- Can only be used when the client is requesting access to the protected resources
  - under its control
  - or belonging to another resource owner that has been previously arranged with the authorization server

Client Credentials Grant



Client

Authorization Server

client authentication (client id & secret)

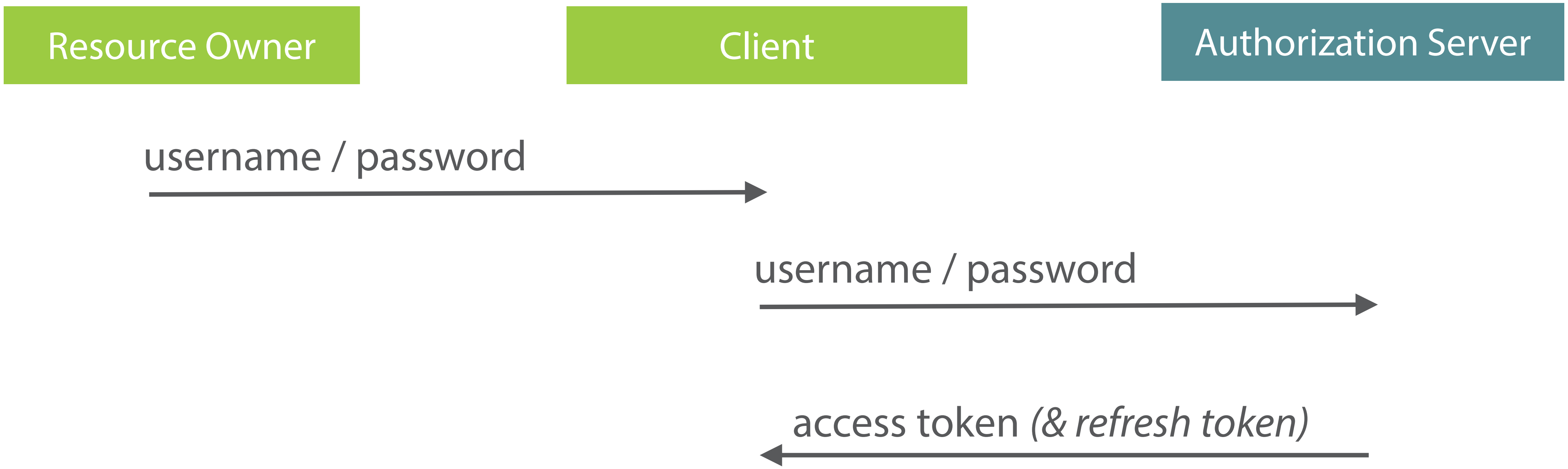
access token

Resource Owner Password Credentials Grant

- Only suitable for **trusted** applications!
- Don’t use this unless other flows are not viable

- The client must be capable of obtaining the **resource owner’s credentials** (username/password)
  - This makes this a no-go for most applications

Resource Owner Password Credentials Grant



Resource Owner

Client

Authorization Server

username / password

username / password

access token *(& refresh token)*

The thing with security is that a lot of approaches will work, but most are not a good idea

OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol

OpenID Connect Specification

- http://openid.net/specs/openid-connect-core-1_0.html

- Defines an id_token
- Defines a UserInfo endpoint to get user information (email, nickname, website, picture, …)

- Authorization Code & Implicit

Hybrid Flow

- Combination of Authorization Code (auth code returned to the client that can be exchanged for id_token & access_token) & Implicit (id_token and access_token directly returned to the client)

- Some tokens are returned from the authorization endpoint, others from the token endpoint

- This allows a client app to use the id_token to get access to a user's identity, but at the same time receive an authorization code through which a refresh token can be obtained

Summary

OAuth 2 is the way to go for authorization

Defines 4 grants:

- Authorization Code
- Implicit
- Client Credentials
- Resource Owner Password Credentials

OpenID Connect adds identity to OAuth2

Defines a new flow: Hybrid