Securing Clients to the API Server

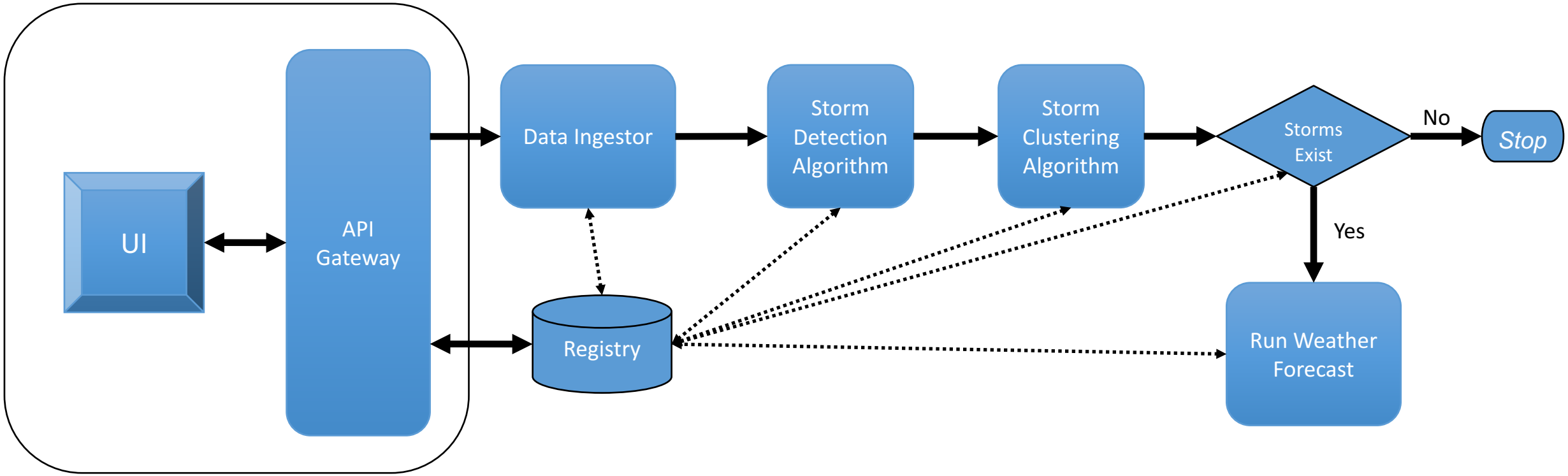
Applications for OAuth2 and OpenID Connect to Science Gateways

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

Remember...

- This is a class about science gateway architectures
- We have two major divisions in the architecture.
- **Science gateway tenants** are what the end user interacts with.
  - Domain specific: SEAGrid.org, Geo-Gateway.org, etc
  - Maintain their own user bases
- **Science gateway middleware** provides general purpose services that are used by gateway tenants.
- One middleware instance can support multiple science gateway tenants.
- Our focus in this class is on the middleware.

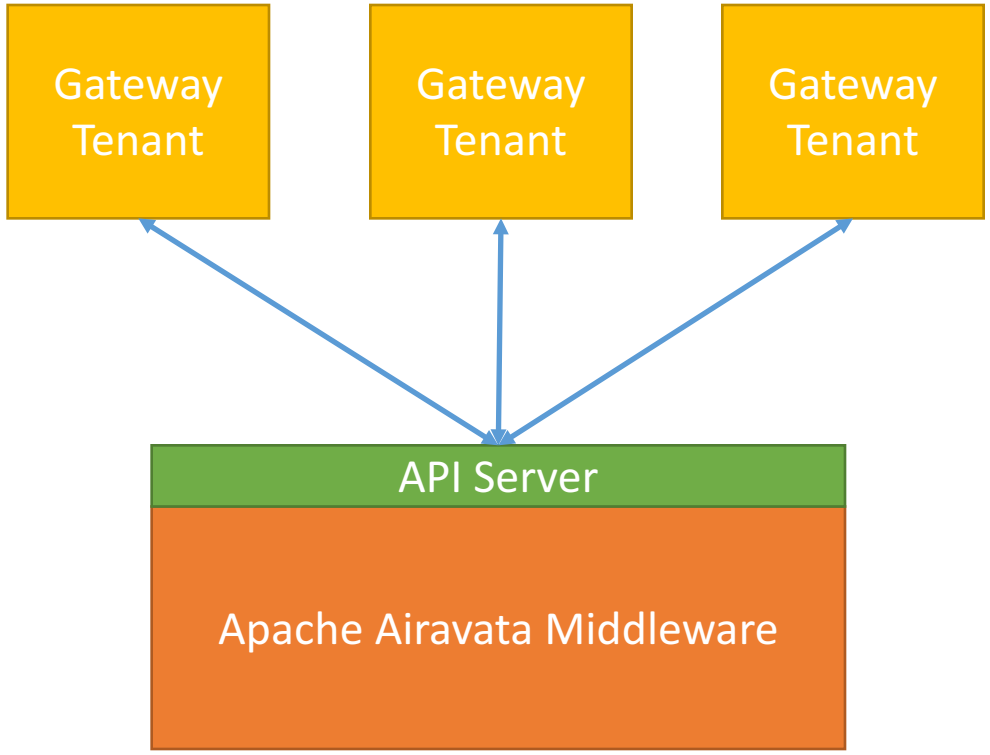
Your Microservice Layout



We are interested in how to secure the UI-API Gateway Connection

Zoom in on the UI and API Server

- UI: this is the gateway tenant
- The API Server can communicate with multiple tenants.
- Tenants can be Web servers, mobile applications, native browser JavaScript apps, or desktop applications.
- Tenants and the API server communicate over network connections.



Security Challenges for This Architecture

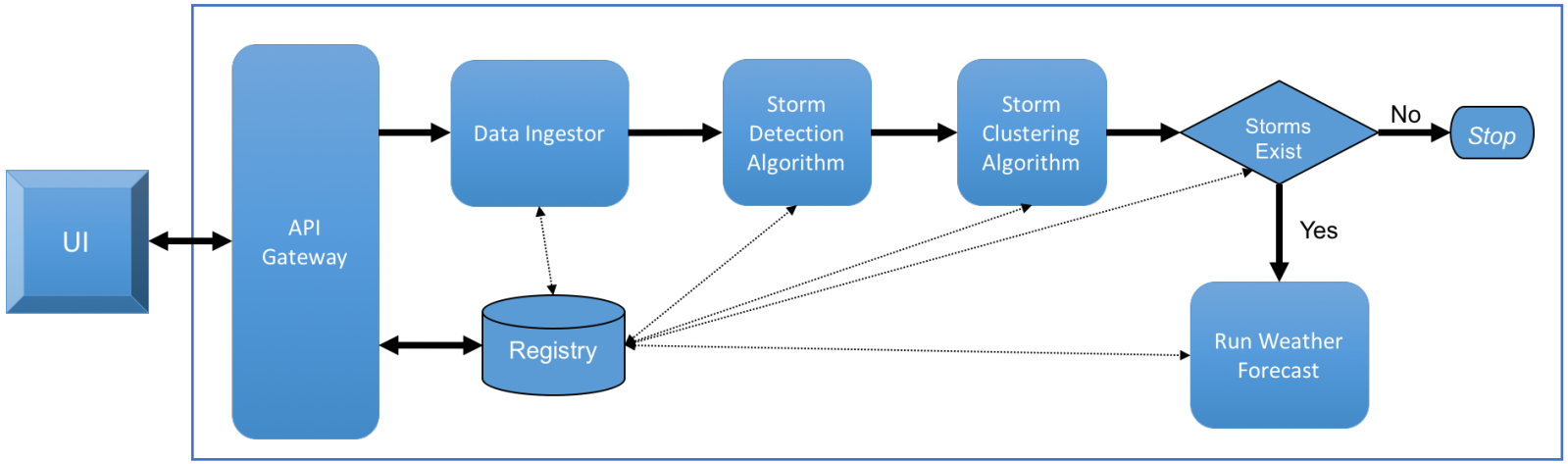
- We need to establish trust between a gateway tenant and the API server.
- The gateway tenant may manage its own user base, but these must be communicated to the API server.
- Gateway tenants can be a single web server for an entire community
  - Web server tenants are single security repositories
- Or they can be desktop or browser clients that get distributed to every user.
  - Need unique credentials for each client
  - Credentials are more vulnerable

Simplifying Assumptions

- We don’t consider the problem of securing the microservices themselves.
  - This could use OAuth2
- Assume all the microservices run under a single administrative domain.
  - That is, you deploy to EC2 instances under your control.
- Use “operational security” rather than “architectural security”
  - Firewalls, closed networks and similar approaches to limit access to services to trusted VMs.

Operational security does introduce some interesting problems

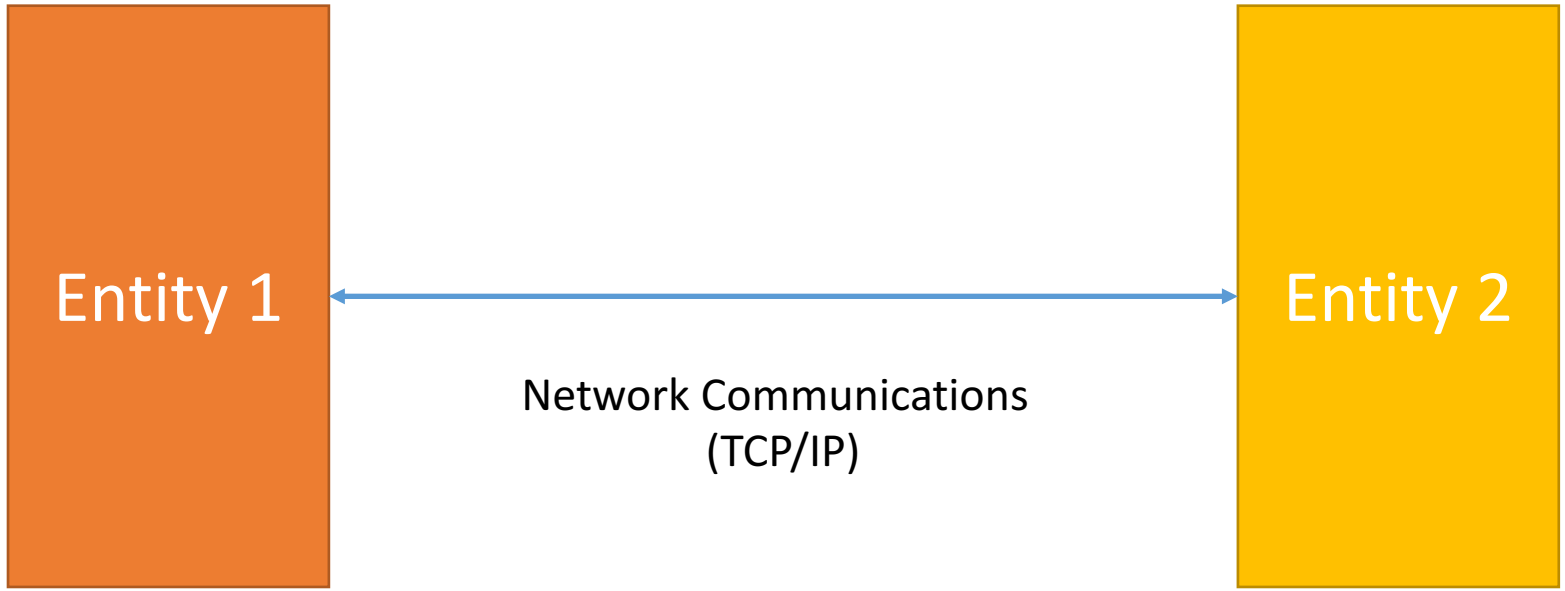
- Rogue services
- Scaling your operational perimeter
- Integrating trusted third party services.



Network Security and OAuth2

A basic introduction

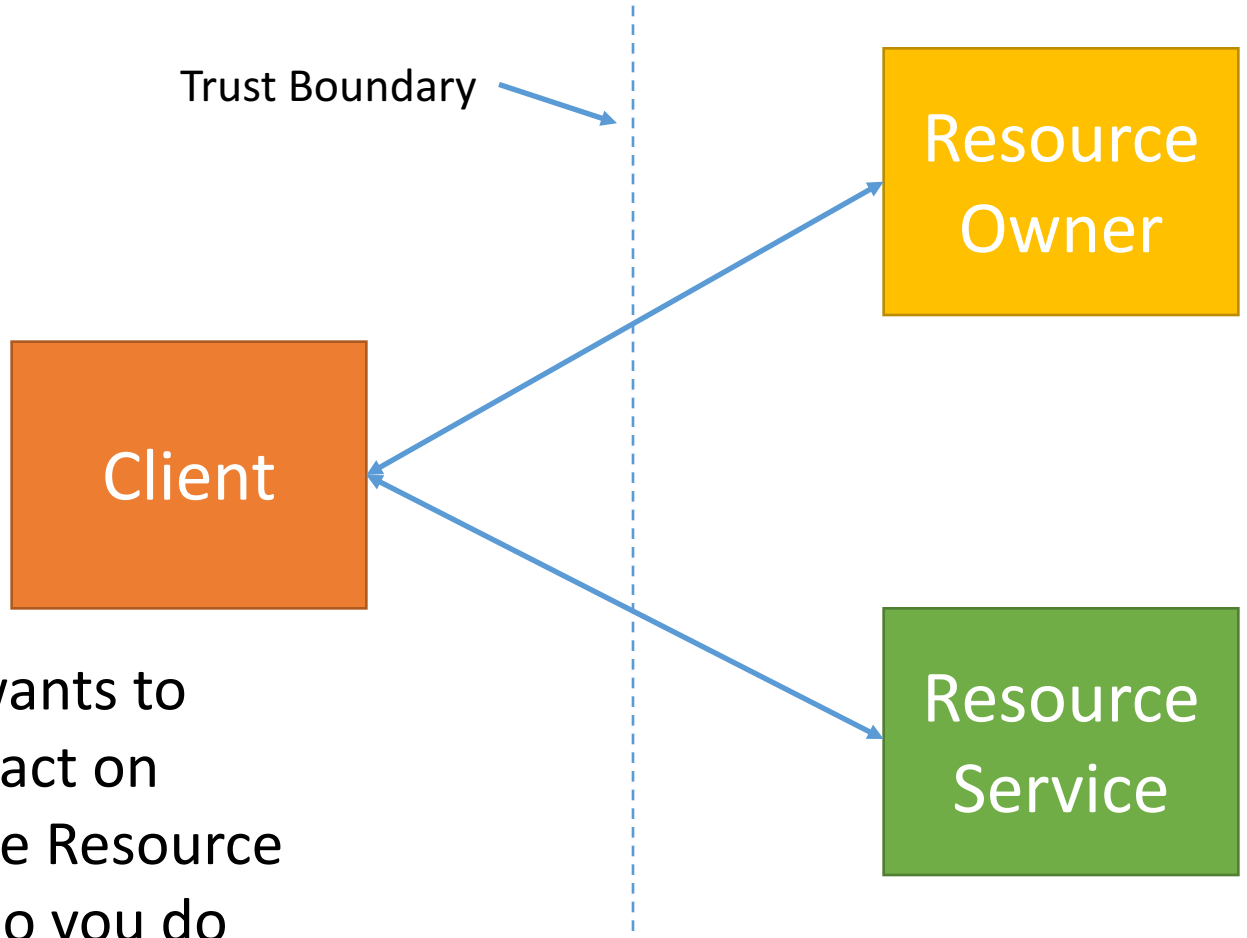
Entities on a Network



Security Concept	Description
Identity	Entities have unique identities.
Authentication (AuthN)	Entities can establish and prove their identities
Authorization (AuthZ)	How an entity responds to a request from another entity. Usually coupled with authentication.
Message Signing	Entities can verify that messages came from a particular authenticated entity. Implemented with cryptographic keys
Message Integrity	Detecting if the network message between entities has been altered. Implemented with message digests.
Message Privacy	Communications between entities can only be read by those entities. Implemented with encryption.
Message Singularity	Each message between entities is unique. Avoids accidental or malicious replays. Uses nonces, timestamps, etc.

Some Basic Network Security Concepts

The Authorization Problem



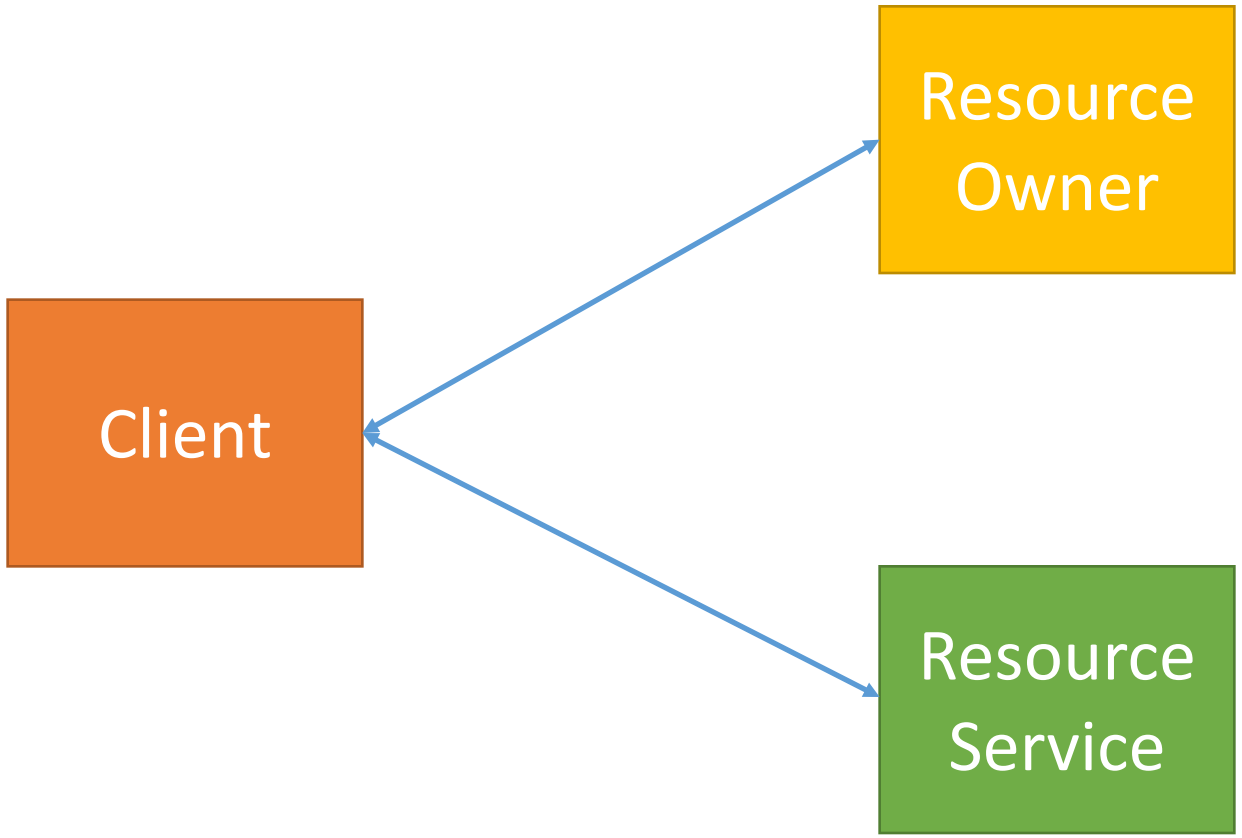
The **Resource Owner** wants to authorize the **Client** to act on **Resource Service** on the Resource Owner's behalf. How do you do delegate this authority?

Authorization and 3rd Party Services

- This scenario has become very common.
- Driven by social networking, PaaS and SaaS, and mobile devices
- Platforms and devices such as Facebook, Google, and Apple hold your personal data.
- Third party applications need to access some of this data.
- You decide which applications to authorize
  - “Facebook, it is ok for this application to access the names of my Facebook friends and other personal information.”
  - “IPhone, it is OK for this app to know my location”

Problems Delegating Authority

- Straightforward Approach: Client requests an access-restricted resource by authenticating **using the resource owner's credentials**.
  - The resource owner shares its credentials with the third party Client.
- This is a really bad solution
  - What are some problems with this approach?

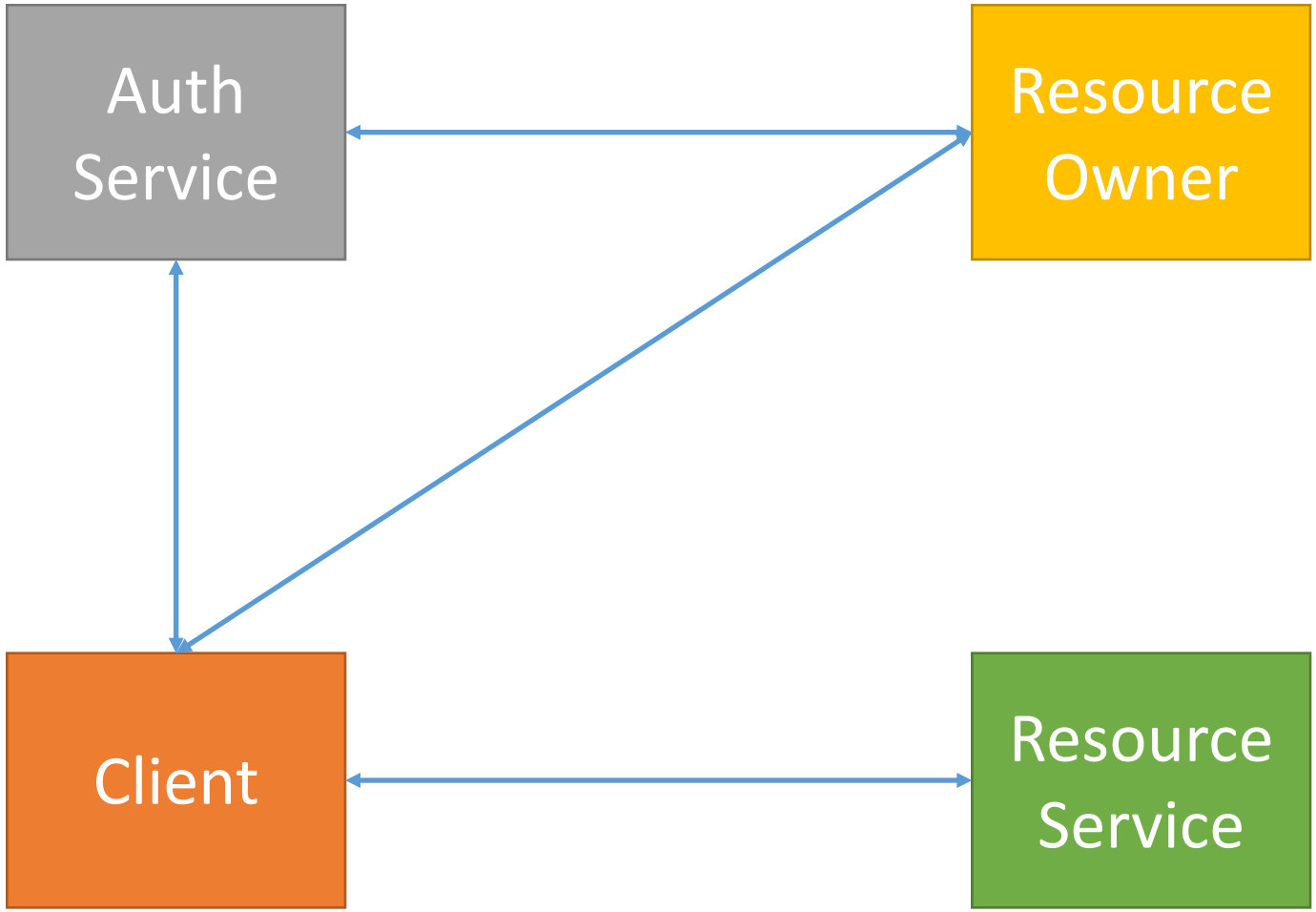


Problems with Credential Sharing

- Third-party applications store the resource owner's credentials for future use.
- Third-party applications gain overly broad access to the resource owner's protected resources.
  - No ability to restrict duration or access to a limited subset of resources.
- Resource owners cannot revoke access to an individual third party without revoking access to all third parties
  - Changing passwords.
- Compromise of any third-party application results in compromise of the end-user's long term credentials and all of the data protected by that password.

Introducing OAuth2

OAuth2 solves this problem by introducing a mutually trusted Authorization Service



OAuth2 Main Concepts

- OAuth2 introduces an authorization layer
  - Separates the role of the client from that of the resource owner.
- In OAuth2, the client is issued a different set of credentials than those of the resource owner.
  - OAuth2 access tokens rather than passwords
- An OAuth2 access token has a specific scope, lifetime, and other access attributes.
- Access tokens are issued to third-party clients by an Authorization Server with the approval of the Resource Owner.
- The Client uses the access token to access the protected resources hosted by the Resource Server.

Types of Clients

Client Type	Description
Web Application	Client runs on a Web server. Client credentials and access tokens are stored on a Web server.
Native Applications	Client runs on a device used by the Resource Owner. Client credentials and access tokens are stored on the device.
User Agent Applications	Client code is downloaded from a server and runs on the user's device (Web browser). Client credentials and access tokens are stored on the user's device.

These clients have different security implications

Client Registration: Trusting the Client

- Clients register with the Authorization Server
  - This is a one time operation.
- The Client can be either confidential or public
  - Confidential: a web server-based Client, for example
  - Public: Browser, desktop or mobile clients
- The Authorization Server issues a client identifier to the Client
  - Unique string representing the information provided by the client.
- Confidential Clients authenticate to the Authorization Server
  - Passwords, key pairs, etc.

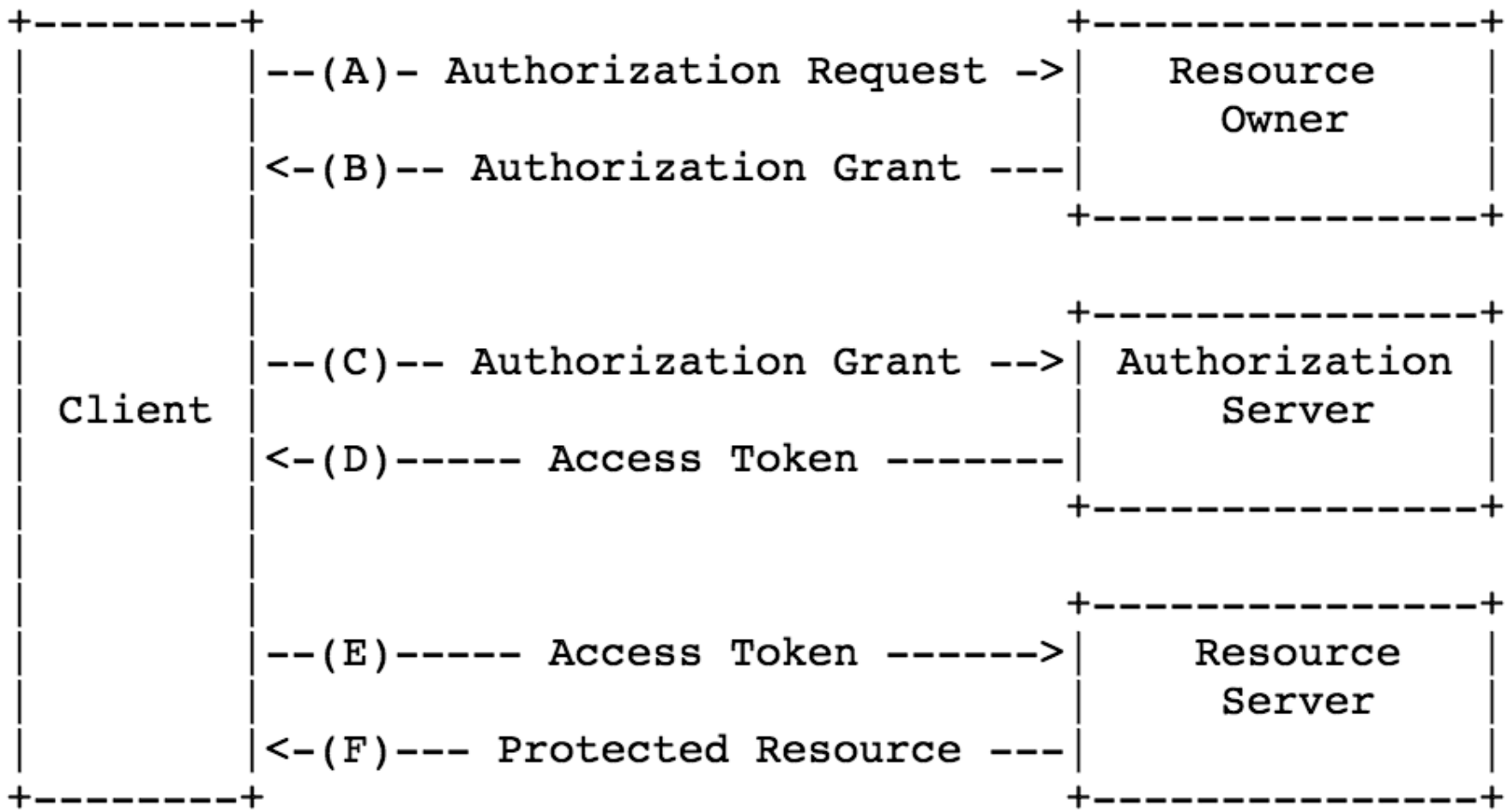


Figure 1: Abstract Protocol Flow

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

OAuth2 In Brief...

- The Resource Owner issues a **grant** to the client.
- The Client uses the grant to get an **access token** from Authorization Service.
- The Client uses the access token to make requests from the Resource Service.
- OAuth2 has several **grant types** that are appropriate for different scenarios.

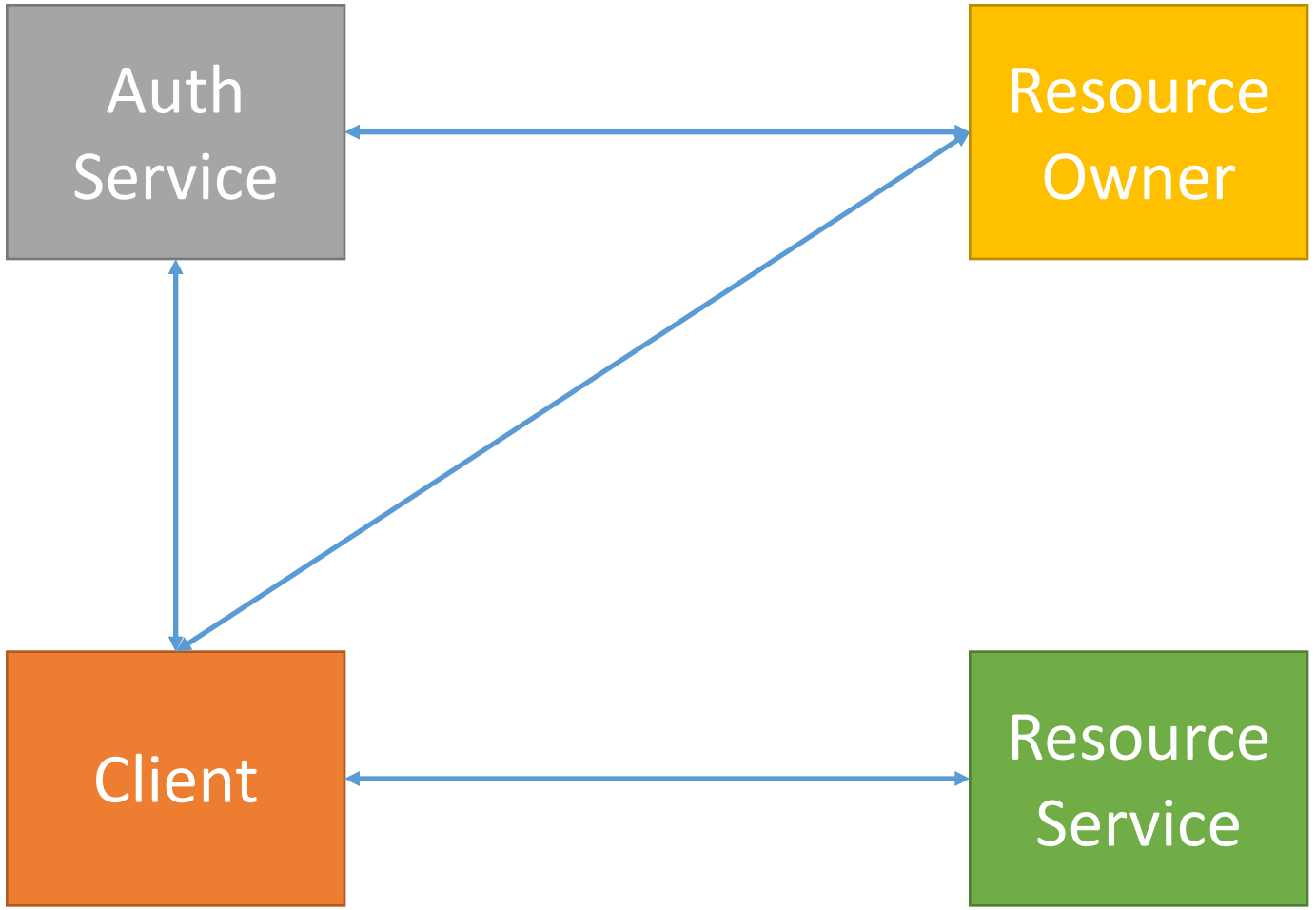
Grant Type	Description
Authorization Code	• Client directs the Resource Owner to an Authorization Server. • Resource Owner authenticates to the Authorization Server • Auth Server issues an auth code to the Resource Owner and then directs the Resource Owner back to the Client. • The Client uses the auth code to get an access token directly from the Authorization Server
Implicit	Authorization flow suitable for JavaScript clients in a browser. Client gets the access token directly in a redirect URL, skipping the authorization code step. Convenient but less secure.
Resource Owner Password Credentials	Resource Owner gives the Client its full credentials. Client uses these to obtain an access token. Owner must trust the Client, and Client can use the credentials only once. Included for backward compatibility.
Client Credentials	Client and Resource Server are owned by the same entity, or Client and Resource Owner are the same. Ex: Facebook services only access your personal data if you authorize them.

Refresh Tokens

- Used to obtain new access tokens after the access token has expired.
- Only sent to the Authorization Server, not the Resource Server.
- Issued to the Client by the Authorization Server when the Access Token is issued.
- Refresh tokens are optional

Assumptions in OAuth2

What are some ways to attack OAuth2? How can OAuth2 defend against these attacks?



OAuth2 Network Security Considerations

- Resource Owner, Resource Server, Client, and Authorization Server must all trust each other.
  - This is the mutual authentication problem for entities not in the same administrative domain.
- Examples:
  - Resource Server must know that the access token came from a legitimate client.
- Authorization codes must be single use and short lived
- Message privacy is required when transmitting access tokens.
  - TLS security
- Message integrity and nonces also important.

OpenID Connect: A Summary

An OAuth2-Based Authentication Protocol

http://openid.net/connect/

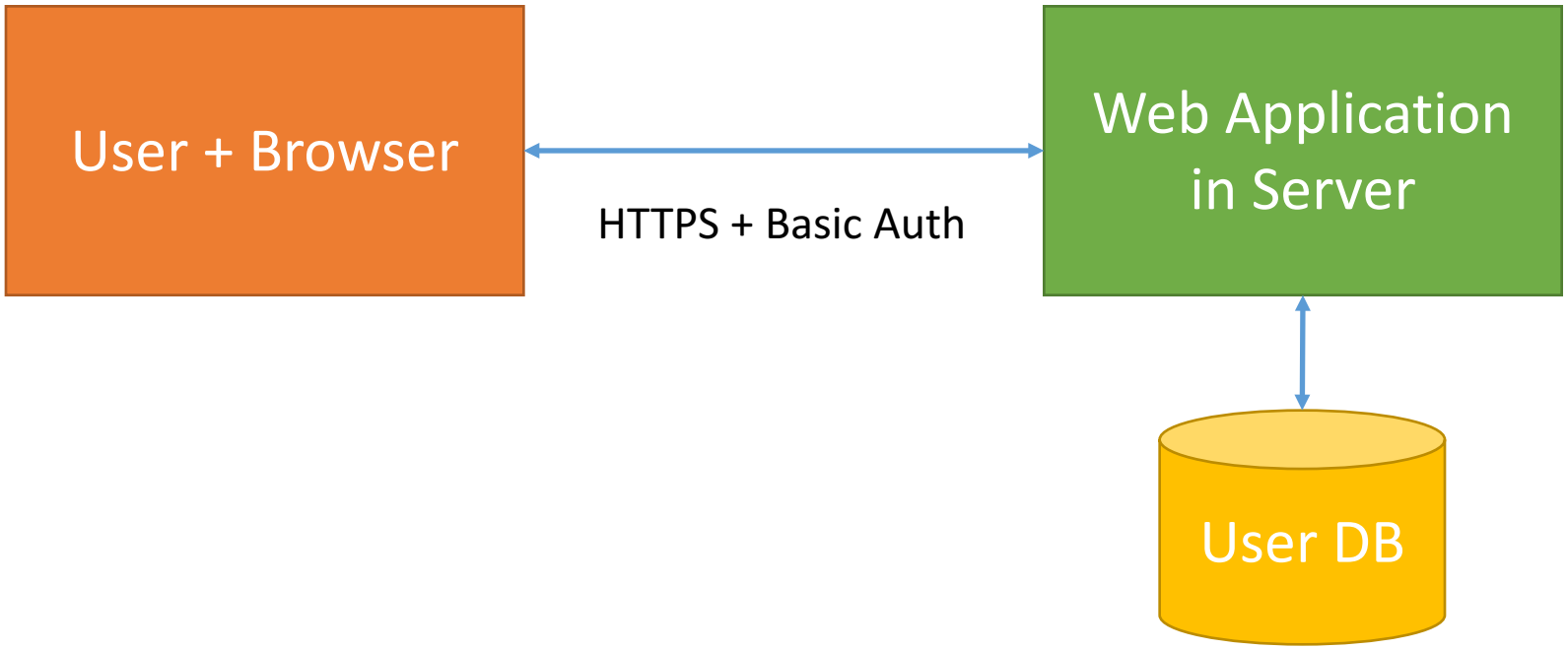
Why OpenID Connect?

- Authentication as a Service
  - Don’t run your own authentication service
  - Use a trusted service instead
  - Authentication mechanisms and details handled by the service.
- Why? The trusted Identity Provider (IdP) absorbs lots of headaches
  - Best practices and implementations for securing user accounts and information.
  - Avoids the need to provide separate identity management for every application
  - Handles federated identities.
  - Handles advanced authentication mechanisms such as two-factor authentication
- Examples
  - CAS: not OpenID Connect based, but similar
  - WSO2 Identity Server: Open source software for running your own IdP. We use this for Apache Airavata.
  - Google, Microsoft, Salesforce, Paypal, Yahoo (whoops...)

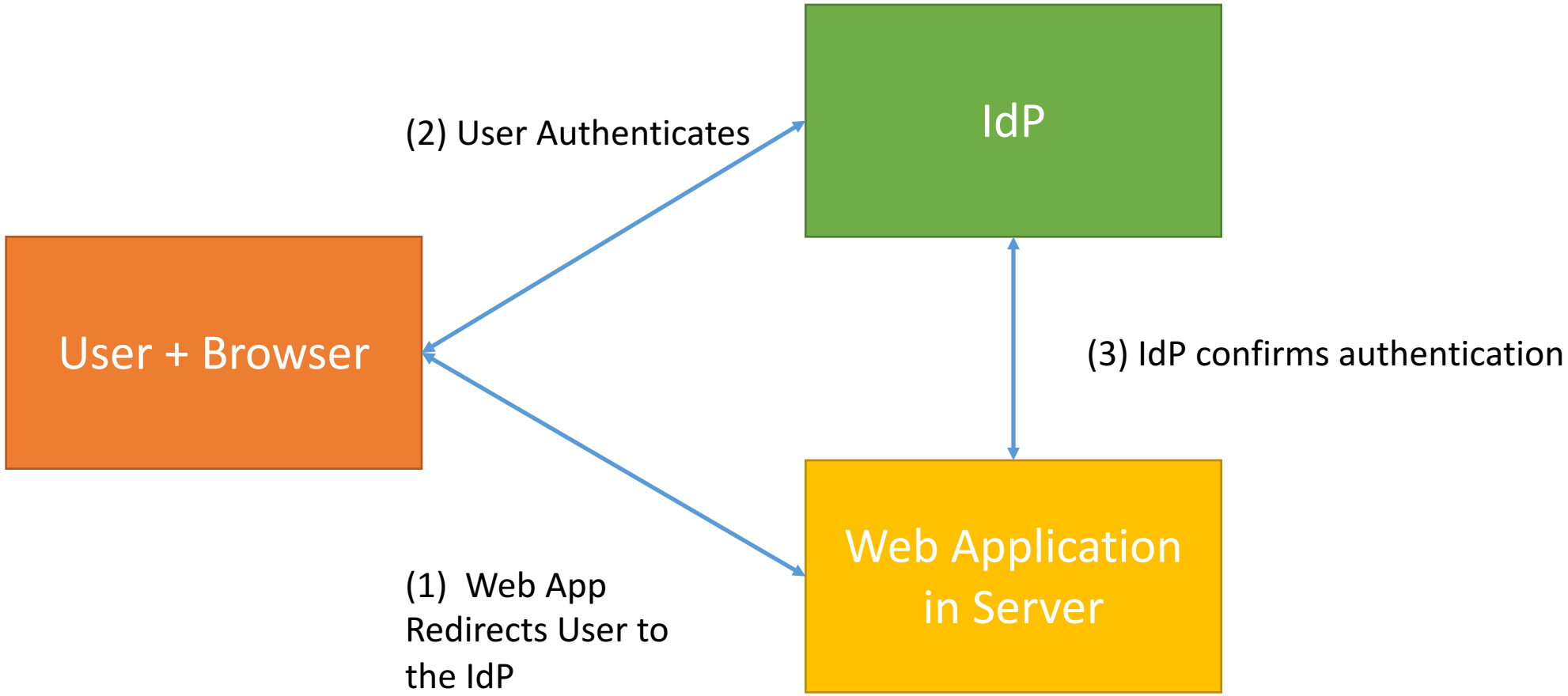
OAuth2 and OpenID Connect

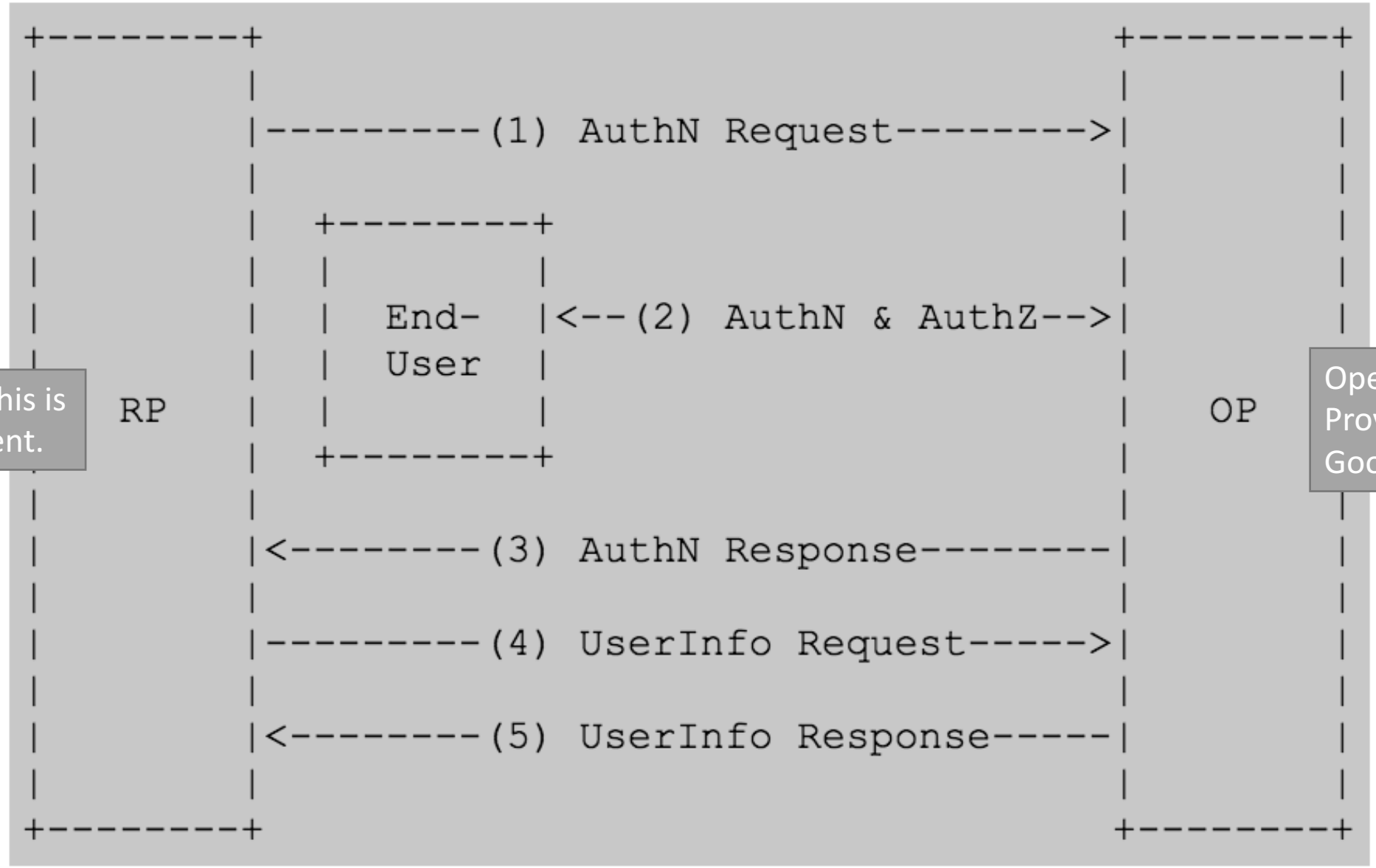
- OAuth2 is used to authorize clients to access resources using access tokens.
  - Establishing client identity is a one-time operation
  - Access tokens are used to access services.
- OpenID Connect uses the same ideas to authenticate users before they can access services.
- Clients can also obtain basic profile information about the user in an interoperable and REST-like manner.
  - Suitable for APIs, not just browser clients

Direct Authentication



Authentication as a Service





Relying Party. This is the OAuth2 Client.

OpenID Connect Provider (i.e., Google)

Basic OIDC Flow

Basic OIDC Steps

- The RP (Client) sends a request to the OpenID Provider (OP).
  - This is the science gateway
- The OP authenticates the End-User and obtains authorization.
- The OP responds with an ID Token and usually an Access Token.
  - Verifies to the client that the user authenticated correctly.
- The RP can send a request with the Access Token to the UserInfo Endpoint.
- The UserInfo Endpoint returns Claims about the End-User.

OIDC Mappings to OAuth2

- The OIDC server is the Authorization Server.
- The Science Gateway is the Client
- Grant Types used by OIDC
  - Authorization Code: most common code, useful for server-side Web applications
  - Implicit: Use this with browser-side JavaScript applications that need to interact with the OIDC Server directly.

The OIDC ID Token (1/2)

- ID Token data structure is the primary extension that OpenID Connect makes to OAuth 2.0 to enable End-Users to be authenticated.
- The ID Token is a security token that contains **Claims** about the authentication of an End-User by an Authorization Server when using a Client, and potentially other requested Claims.

The OIDC ID Token (2/2)

- The ID Token is represented as a JSON Web Token (JWT)
- JWT: compact claims representation format intended for space constrained environments such as HTTP Authorization headers and URI query parameters.
  - https://tools.ietf.org/html/draft-ietf-oauth-json-web-token-32

Sample OIDC ID Token


```
{
 "iss": "https://server.example.com",
 "sub": "24400320",
 "aud": "s6BhdRkqt3",
 "nonce": "n-0S6_WzA2Mj",
 "exp": 1311281970,
 "iat": 1311280970,
 "auth_time": 1311280969,
 "acr": "urn:mace:incommon:iap:silver"
}
```

Parameter	Value
iss	Issuer Identifier for the Issuer of the response. The iss value is a case sensitive URL using the https scheme that contains scheme, host, and optionally, port number and path components and no query or fragment components.
sub	Subject Identifier. A locally unique and never reassigned identifier within the Issuer for the End-User, which is intended to be consumed by the Client
aud	Audience(s) that this ID Token is intended for. It must contain the OAuth 2.0 client_id of the Relying Party as an audience value. It may contain other values.
nonce	String value used to associate a Client session with an ID Token, and to mitigate replay attacks.
exp	Expiration time
iat	Time at which the JWT was issued.
auth_time	Time when the End-User authentication occurred.
acr	Authentication Context Class Reference. You can used an RFC 6711 Registered Name here. This is an established Level of Assurance identifier.

Additional Claims

- OIDC ID Tokens can also contain additional claims about the user.
  - Examples: Full name, preferred name, profile page URL, picture, website, birthday, etc.
  - These are stored by the UserInfo Endpoint. Not all may stored, and sharing decisions are another story.
- OIDC clients (science gateways) can also make subsequent requests for this information from a UserInfo Endpoint.

UserInfo Endpoint

- The UserInfo Endpoint is an OAuth 2.0 Protected Resource that returns Claims about the authenticated End-User.
- The Client makes a request to the UserInfo Endpoint using an Access Token obtained through OpenID Connect Authentication.

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

{
 "sub": "248289761001",
 "name": "Jane Doe",
 "given_name": "Jane",
 "family_name": "Doe",
 "preferred_username": "j.doe",
 "email": "janedoe@example.com",
 "picture": "http://example.com/janedoe/me.jpg"
}
```

OAuth2, OpenID Connect and Science Gateway API Servers

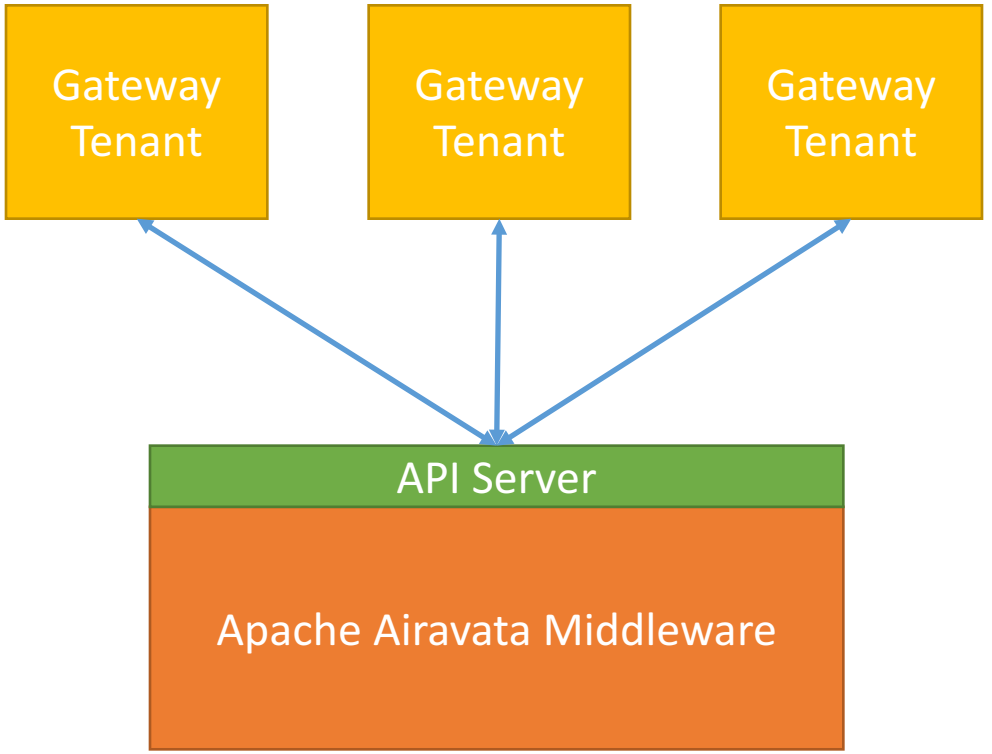
Variations on the OAuth2 Scenarios

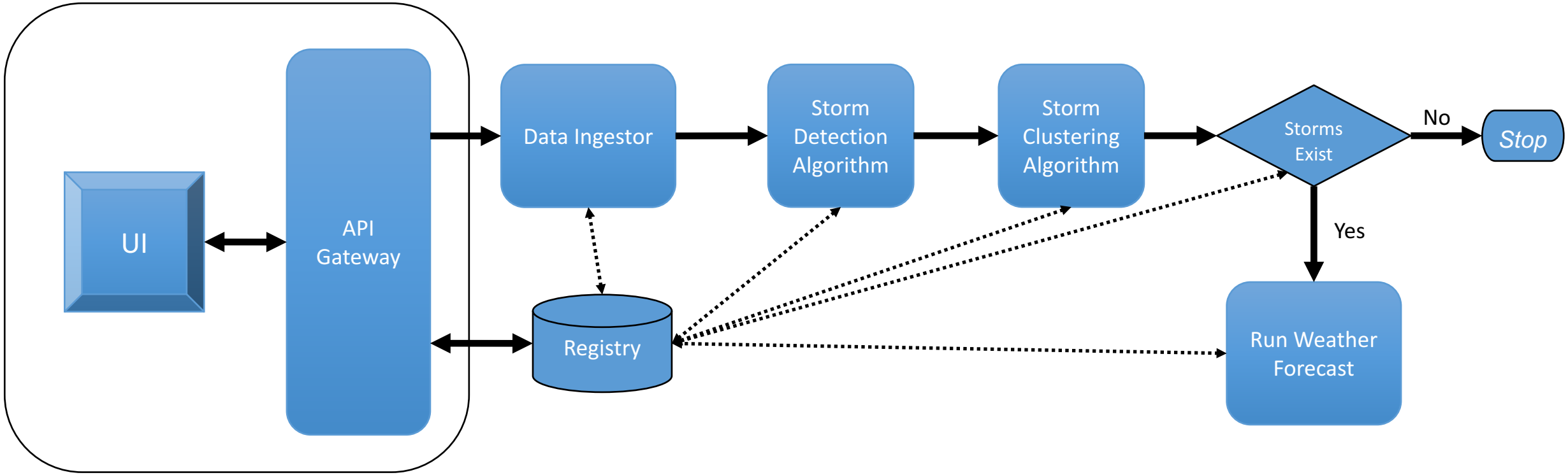
Apache Airavata API Security: Exploring Identity and Access Management Solutions for Multi-Tenanted eScience Framework, *Supun Nakandala, Indiana University; Hasini Gunasinghe, Purdue University; Suresh Marru*, Indiana University; Marlon Pierce, Indiana University*

http://escience-2016.idies.jhu.edu/program/hot-topics-invited-talks/

General Gateway Issues

- Science Gateways use middleware for common, generic functions.
  - Execute jobs, manage data and metdata
- Middleware (Airavata) needs a scalable way to establish trust with numerous science gateway tenants.
- Gateway tenants can be Web clients but also desktop clients.
  - These have very different security concerns.
- Science gateways need a way to authenticate users.





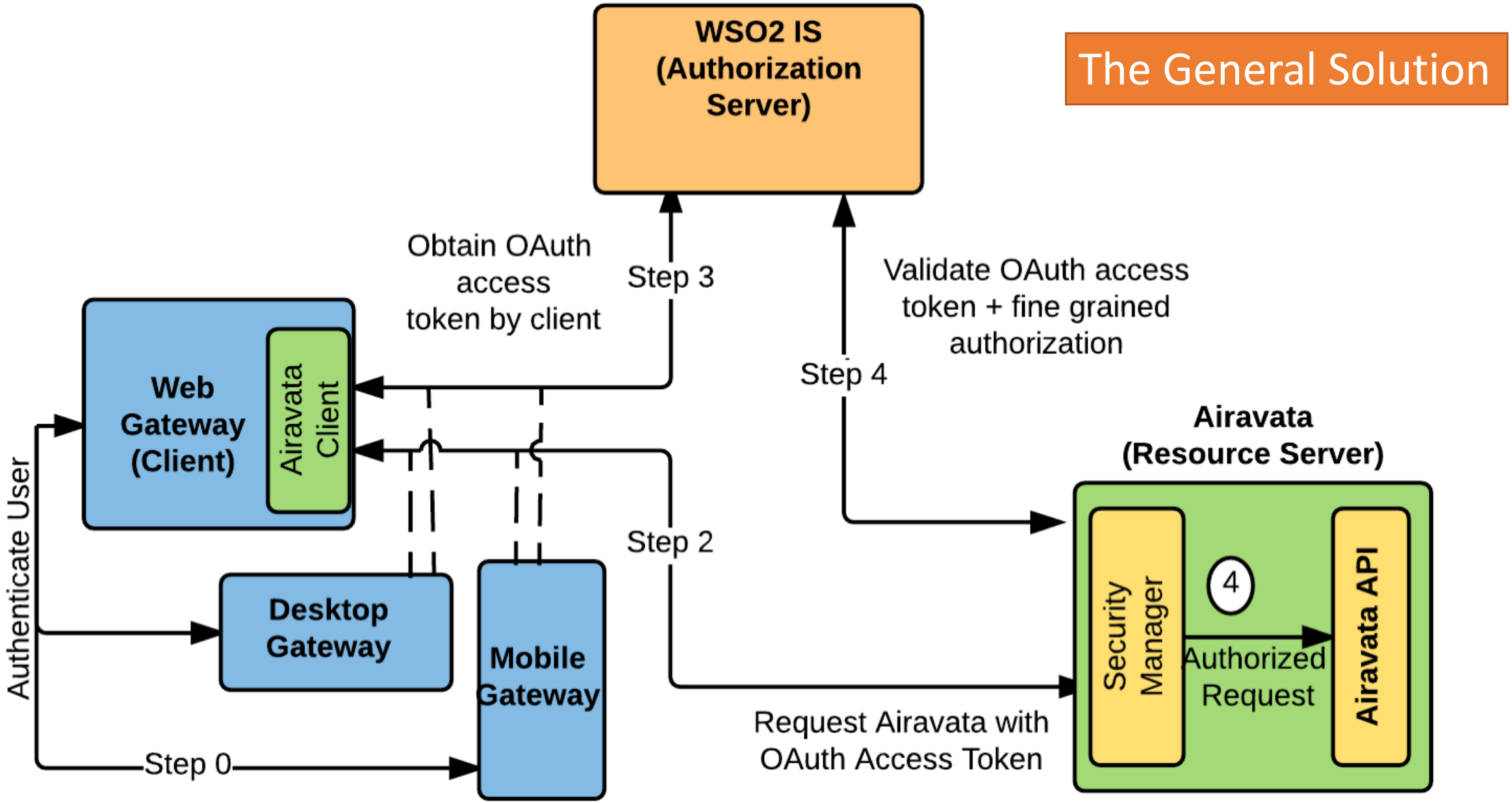
Science Gateways use OAuth2+OpenID Connect to establish trust between gateway clients and the API server.

- API Server is multi-tenanted. It's a platform.
- Gateways are clients to the API server
- Gateways have separate user bases.
- Gateways can be Web servers, desktop applications, or browser clients.

SEAGrid Scenarios

- SEAGrid needs to authenticate users
- SEAGrid has both Web and desktop (JavaFX) clients.
  - These are clients to Apache Airavata services.
- Web client (PHP) runs on a server under the control of the SEAGrid administrator.
- But desktop clients run under the user's control.
    - User could lose them.
- Apache Airavata needs to issue access tokens to invoke API calls to both the SEAGrid web site and to SEAGrid desktop clients.

The General Solution



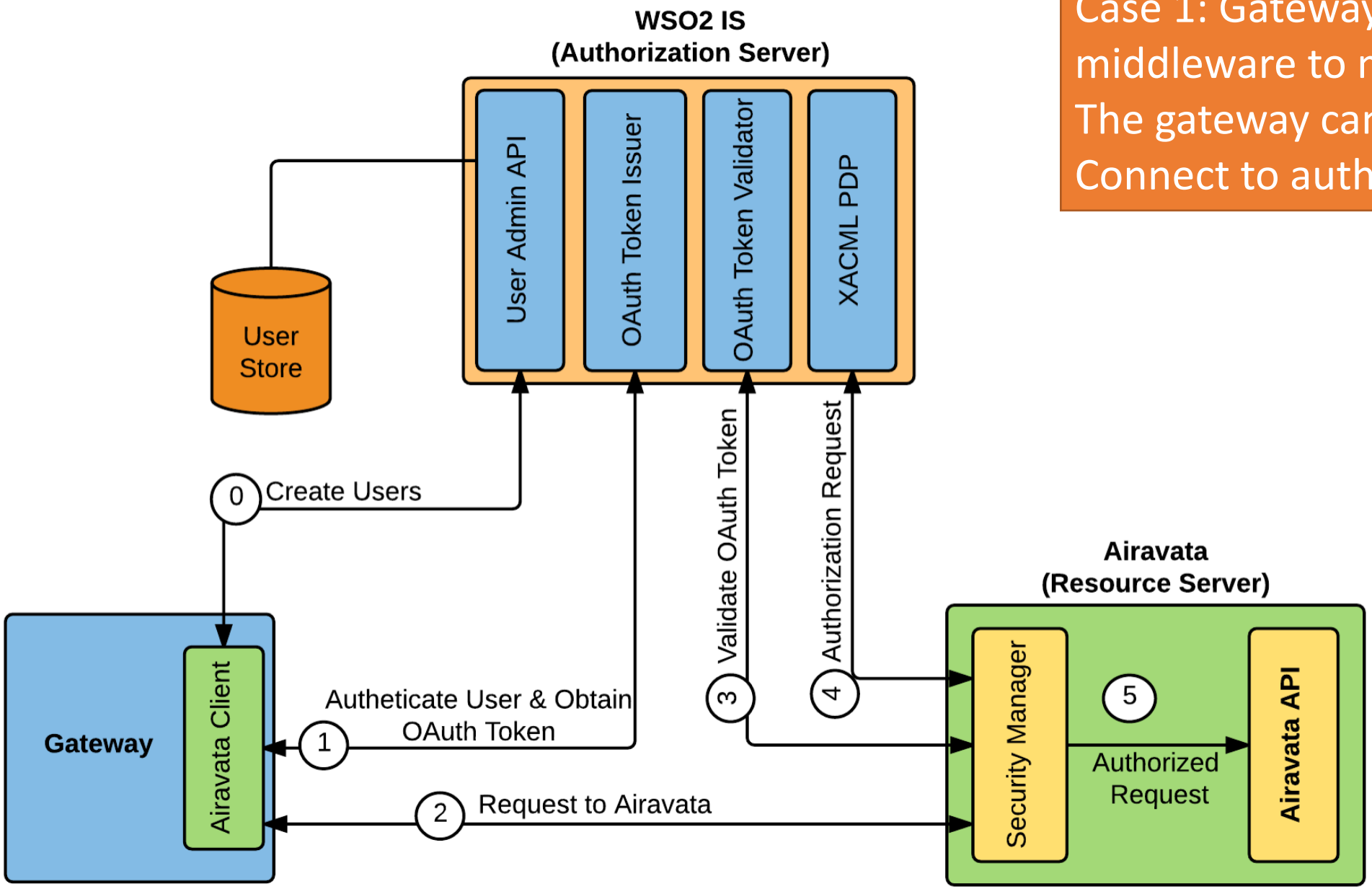
Our Conclusions About OAuth2 and Gateways

OAuth2 Grant Type	Science Gateway
Authorization Code	Web-based, server side gateway implemented with PHP, JSP/servlets, Django, etc. The Airavata client SDK is on the server under the gateway operator's control
Implicit	Client is a browser using JavaScript client SDKs to make direct connections to the Airavata server; no Web server in the middle
Resource Owner Password	Client is a trusted non-browser application under the user's control, such as a mobile device or a desktop application.
Client Credential	Machine-to-machine authentication

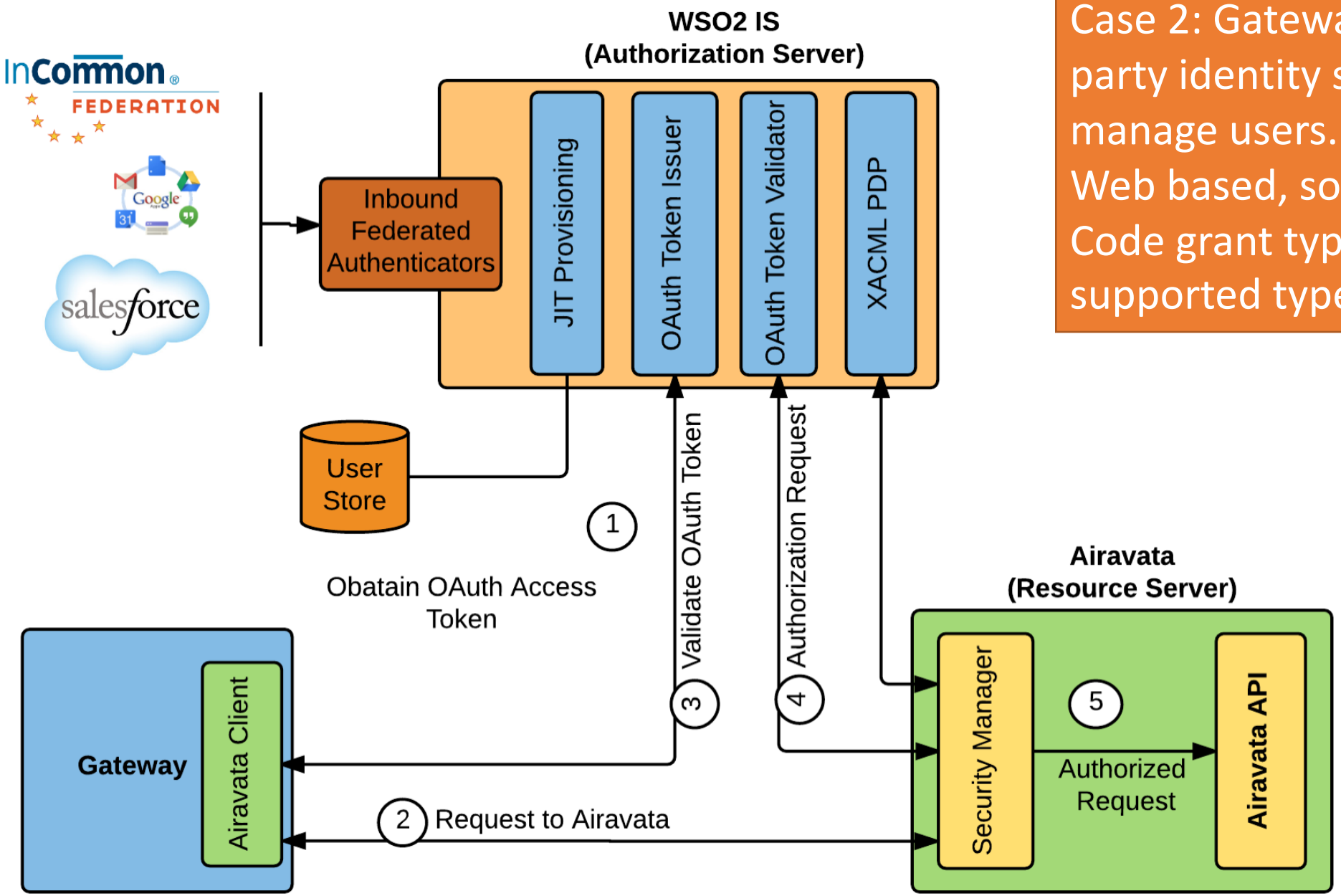
How Do We Handle Authentication and User Management

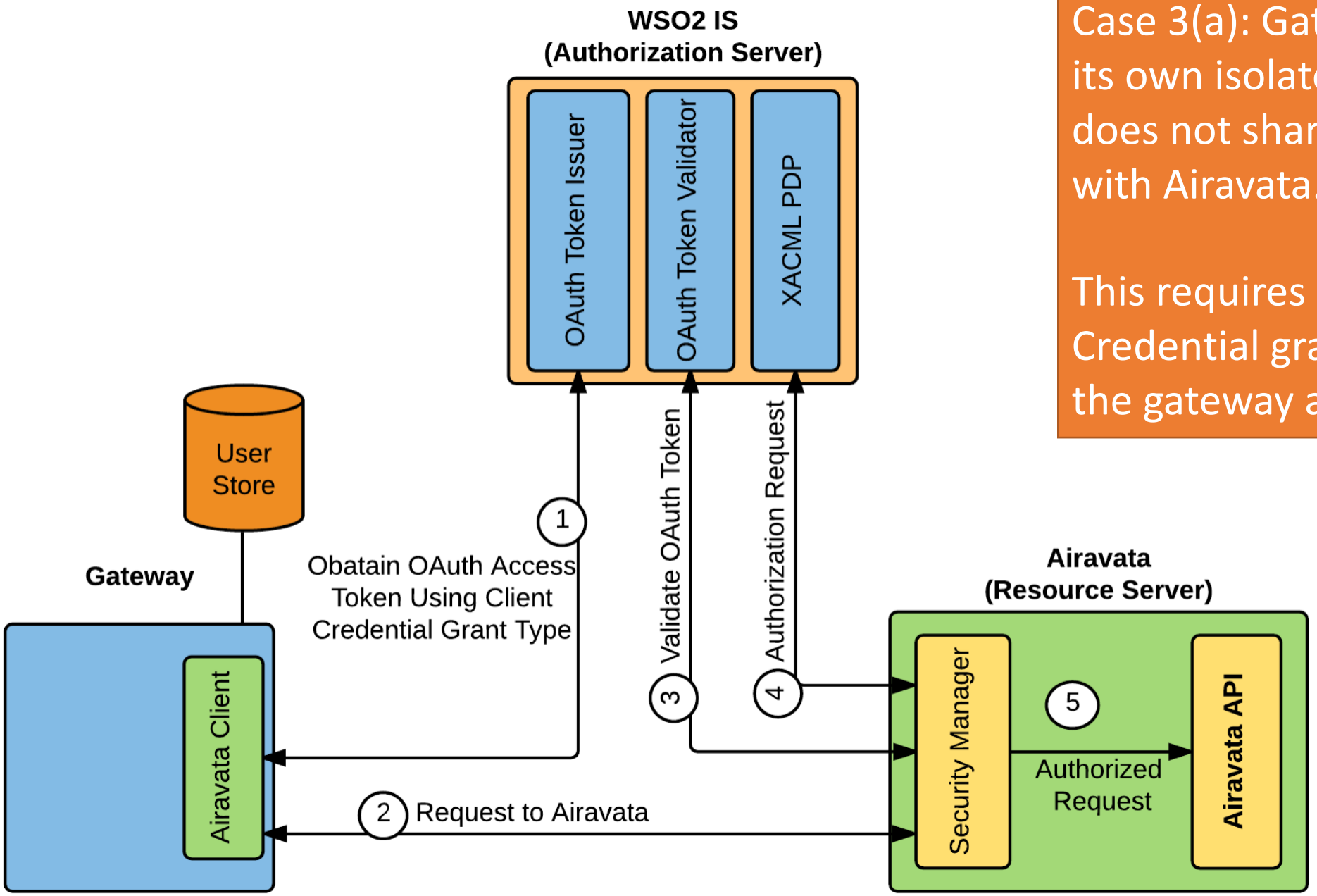
- There are several possibilities.
- Some gateways will use this as a middleware service.
  - Case 1
- Other gateways will use third party identity providers
  - Case 2
- Still other gateways will bring their own user stores.
  - Case 3

Case 1: Gateway uses Airavata middleware to manage users. The gateway can use OpenID Connect to authenticate users.



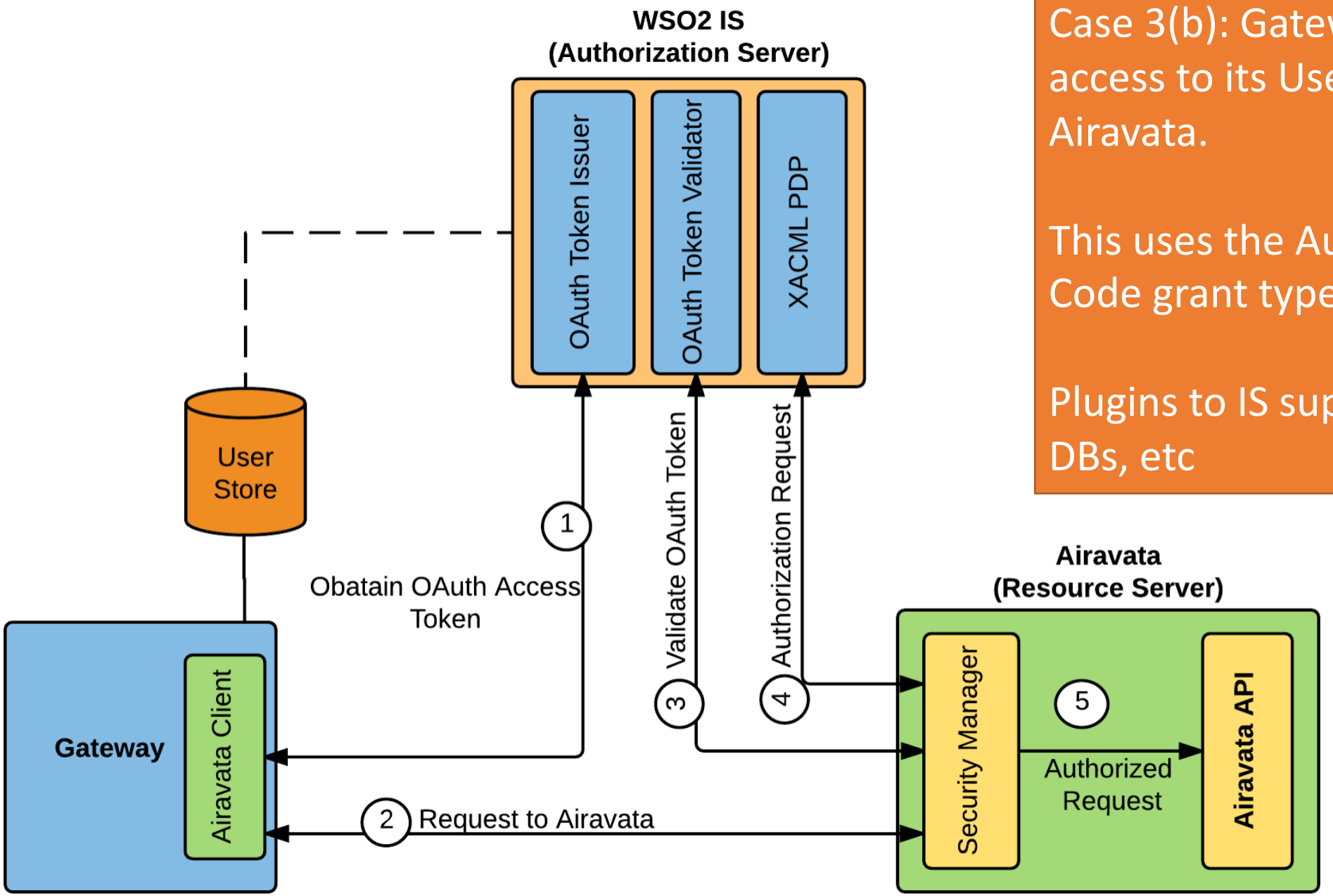
Case 2: Gateway uses third party identity service to manage users. This must be Web based, so Authorization Code grant types are the only supported type.





Case 3(a): Gateway maintains its own isolated User Store and does not share information with Airavata.

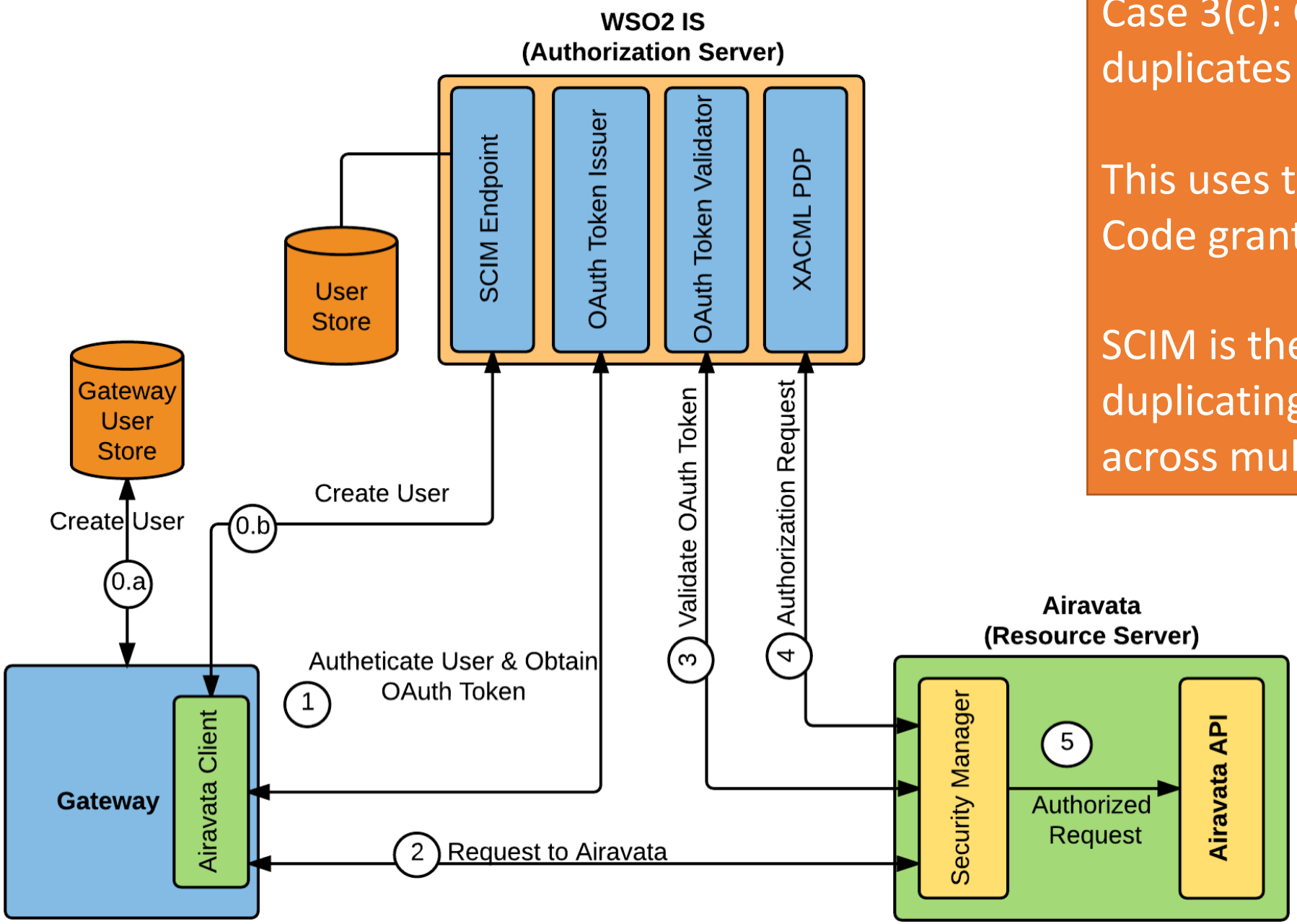
This requires a Client Credential grant type between the gateway and Airavata.



Case 3(b): Gateway shares read access to its User Store with Airavata.

This uses the Authorization Code grant type.

Plugins to IS support LDAP, DBs, etc



Case 3(c): Gateway shares duplicates its user store to IS.

This uses the Authorization Code grant type.

SCIM is the protocol for duplicating user information across multiple user stores.

Applications to Milestone 2

- Your UI should be a separate entity from your API server.
  - The UI communications go over the
- I recommend using an existing OAuth2 provider like Google.
- You should only need to worry about implementing Case 1.

Parameter	Value
client_id	A client identifier established between the OIDC server and the client app.
response_type	The value “code” for Authorization Code grant types. Use “id_token” for Implicit grant types.
redirect_uri	The HTTP endpoint on your server that will receive the response from the OIDC server.
scope	In a basic request should be openid email.
state	Should include the value of the anti-forgery unique session token, as well as any other information needed to recover the context when the user returns to your application, e.g., the starting URL.

OpenID Connect Client Request Parameters