Beyond X.509:
A token-based AAI for WLCG

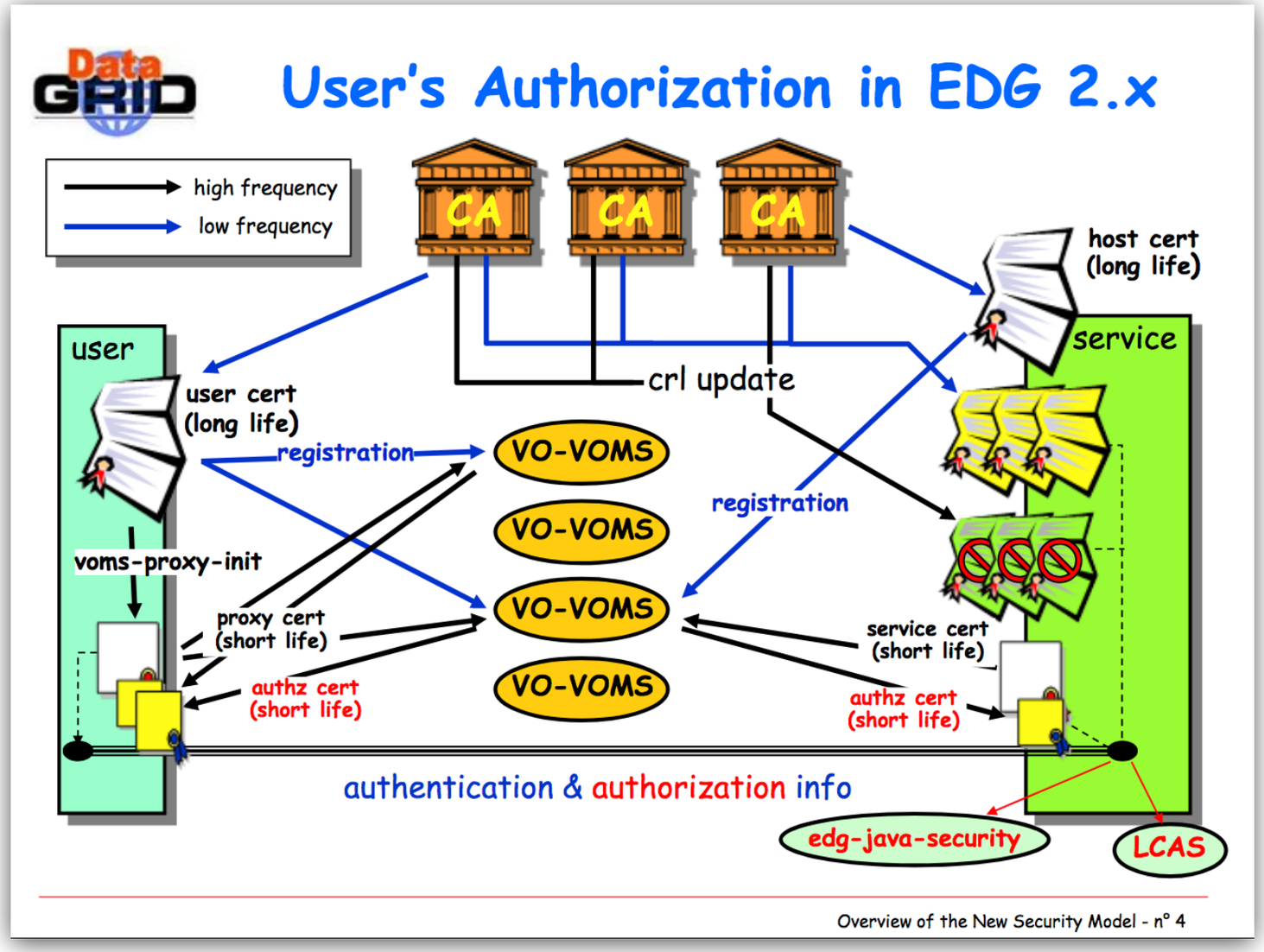
Andrea Ceccanti
DPM Workshop

June 14th 2019

The current WLCG AAI

In operation since ~2003, and still working nicely:

- **X.509 trust fabric** provided by **IGTF** (tells services which CAs are trusted)
- **X.509 certificates** provided to users for authentication
- **Proxy certificates** for Single Sign-On (SSO) and delegation
- **VOMS attribute certificates** for attribute-based authorization (issued and signed by VO-scoped VOMS servers)



Slide by Ákos Frohner

Current WLCG AAI: the weak points

Usability

- X.509 certificates are **difficult** to handle for users
- VOMS does not work in browsers

Inflexible authentication

- Only one authentication mechanism supported: X.509 certificates
- Hard to integrate identity federations

Authorization tightly bound to authentication mechanism

- VOMS attributes are inherently linked to an X.509 certificate subject

Ad-hoc solution

- We had to invent our own standard and develop ad-hoc libraries and central services to implement our own AAI

Can we do better today?

A novel AAI for WLCG: main challenges

Authentication

- **Flexible**, able to accomodate various authentication mechanisms
  - X.509, username & password, EduGAIN, social logins (Google, GItHub), ORCID, …

Identity harmonization & account linking

- Harmonize multiple identities & credentials in a single account, providing a **persistent identifier**

Authorization

- **Orthogonal** to authentication, **attribute** or **capability-based**

Delegation

- Provide the ability for **services to act on behalf of users**
- Support for **long-running applications**

Provisioning

- Support provisioning/de-provisioning of identities to services/relying resources

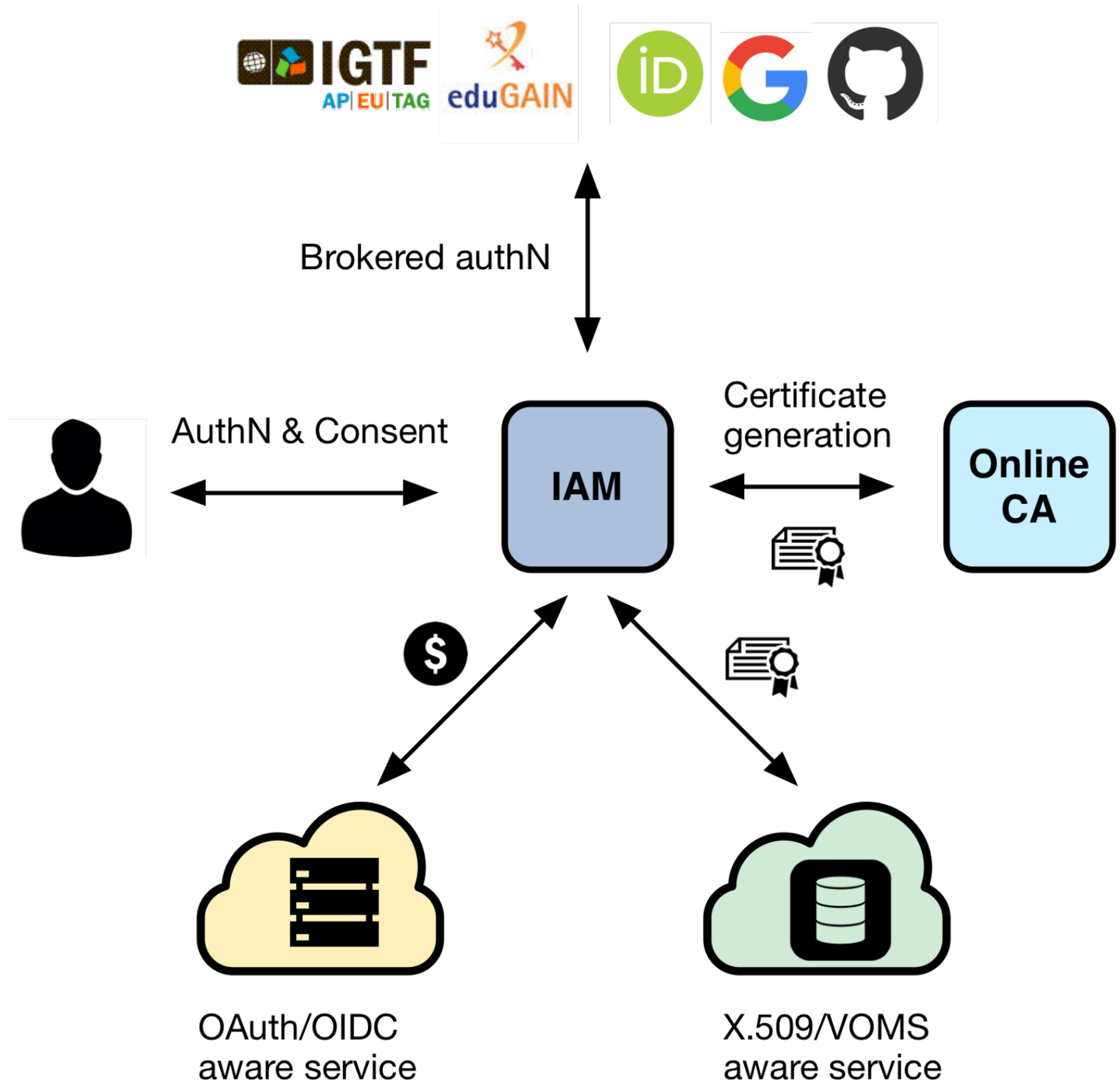
Token translation

- Enable **integration with legacy services through controlled credential translation**

The future token-based WLCG AAI

Introduce a central VO-scoped authz service that

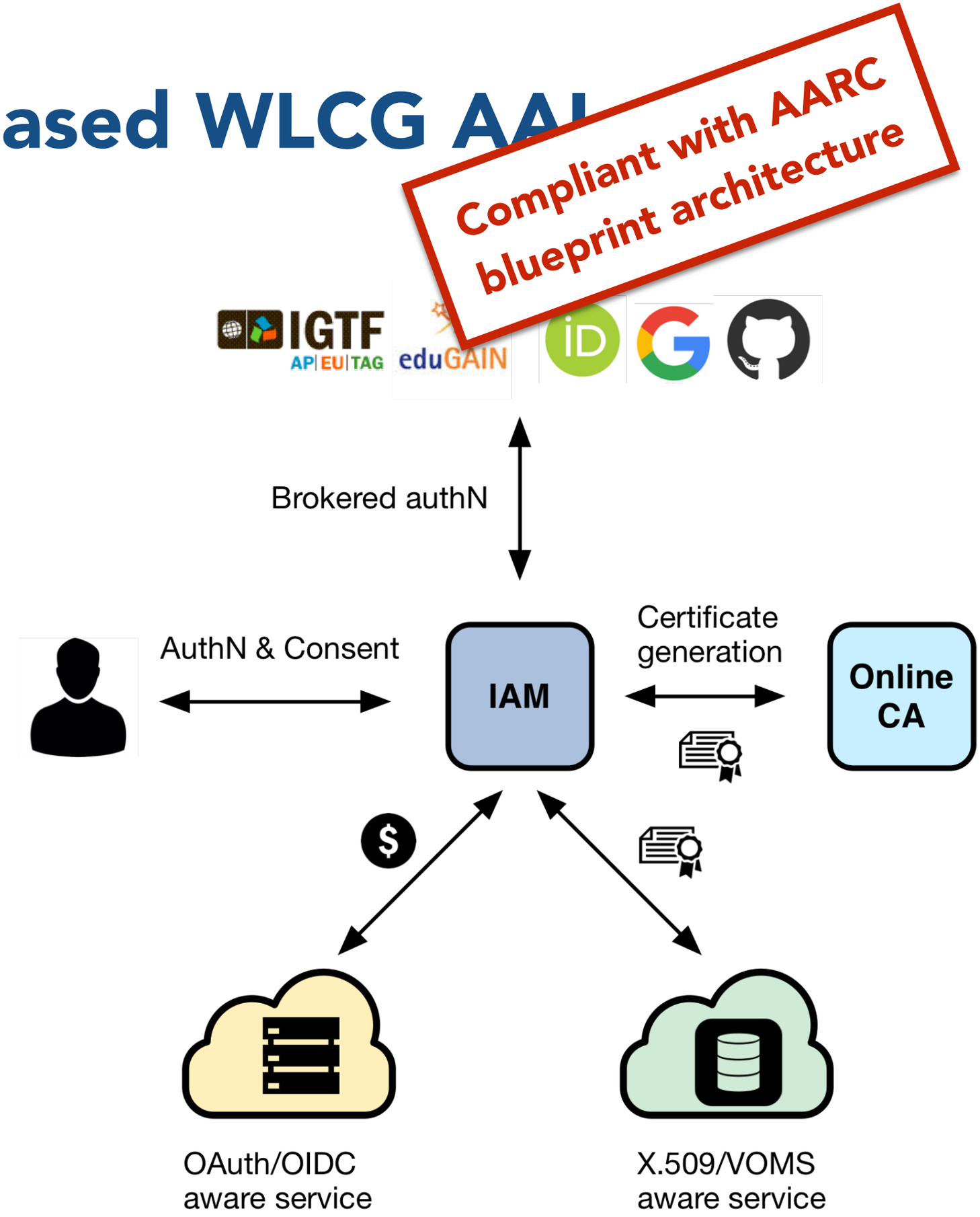
- supports **multiple authentication mechanisms**
- provides users with a **persistent, VO-scoped** identifier
- exposes **identity information**, **attributes** and **capabilities** to services via **JWT** tokens and standard **OAuth & OpenID Connect** protocols
- can integrate existing **VOMS**-aware services
- supports **Web** and **non-Web access**, **delegation** and **token renewal**



The future token-based WLCG AAI

Introduce a central VO-scoped authz service that

- supports **multiple authentication mechanisms**
- provides users with a **persistent, VO-scoped** identifier
- exposes **identity information**, **attributes** and **capabilities** to services via **JWT** tokens and standard **OAuth & OpenID Connect** protocols
- can integrate existing **VOMS**-aware services
- supports **Web** and **non-Web access**, **delegation** and **token renewal**



Enabling technologies: an overview

Enabling technologies in one slide

OAuth 2.0

- a standard framework for **delegated authorization**
- widely adopted in industry

OpenID Connect

- an **identity layer** built on top of OAuth 2
- “OAuth-based authentication done right”

JSON Web Tokens (JWTs)

- a **compact**, **URL-safe** means of representing **claims** to be transferred between two (or more) parties

OAuth: a delegated authorization framework

OAuth defines how **controlled delegation of privileges** can happen among collaborating services

Provides answers to questions like:

- How can an application request access to protected resources?
  - How can I obtain **an access token**?
- How is authorization information exchanged across parties?
  - How is the **access token** presented to **protected resources**? (i.e. APIs)

OpenID Connect: an identity layer for OAuth

OAuth is a **delegated authorization** protocol

- an **access token** states the **authorization rights** of the client application presenting the token to access some resources

OpenID Connect extends OAuth to provide a standard **identity layer**

- i.e. information about **who the user is** and **how it was authenticated** via an additional **ID token (JWT)** and a dedicated **user information query endpoint** at the OpenID Connect Identity provider
- provides ability to establish **login sessions** (SSO)

JSON Web Tokens (JWT)

JSON Web Token (JWT) is an open standard that defines a compact, self-contained way of securely transmitting information between parties as a JSON object

JWTs are typically **signed** and, if confidentiality is a requirement, can be **encrypted**.

Header

```
{
  "kid": "rsa1",
  "alg": "RS256"
}
```

.

Body

```
{
  "sub": "e1eb758b-b73c-4761-bfff-adc793da409c",
  "iss": "https://iam-test.indigo-datacloud.eu/",
  "exp": 1482163788,
  "iat": 1482160188,
  "jti": "e7bcb54c-8f67-4a77-8415-37adeb4b958c"
}
```

.

Signature

```
QbOfPrha9kp4e7TknXe88
d8v_9e7V2v2xMAKX10xY4
M3P1wragAhQmyoVQwq-uk
```

Why OAuth, OpenID Connect and JWT?

Standard, widely adopted in industry

- Do not reinvent the wheel, reuse existing knowledge and tools, extend when needed

Reduced integration complexity at relying services

- Off-the-shelf libraries and components

Authentication-mechanism agnostic

- The AAI is not bound to a specific authentication mechanism

Distributed verification of access and identity tokens

- It scales

Back to our token-based AAI...

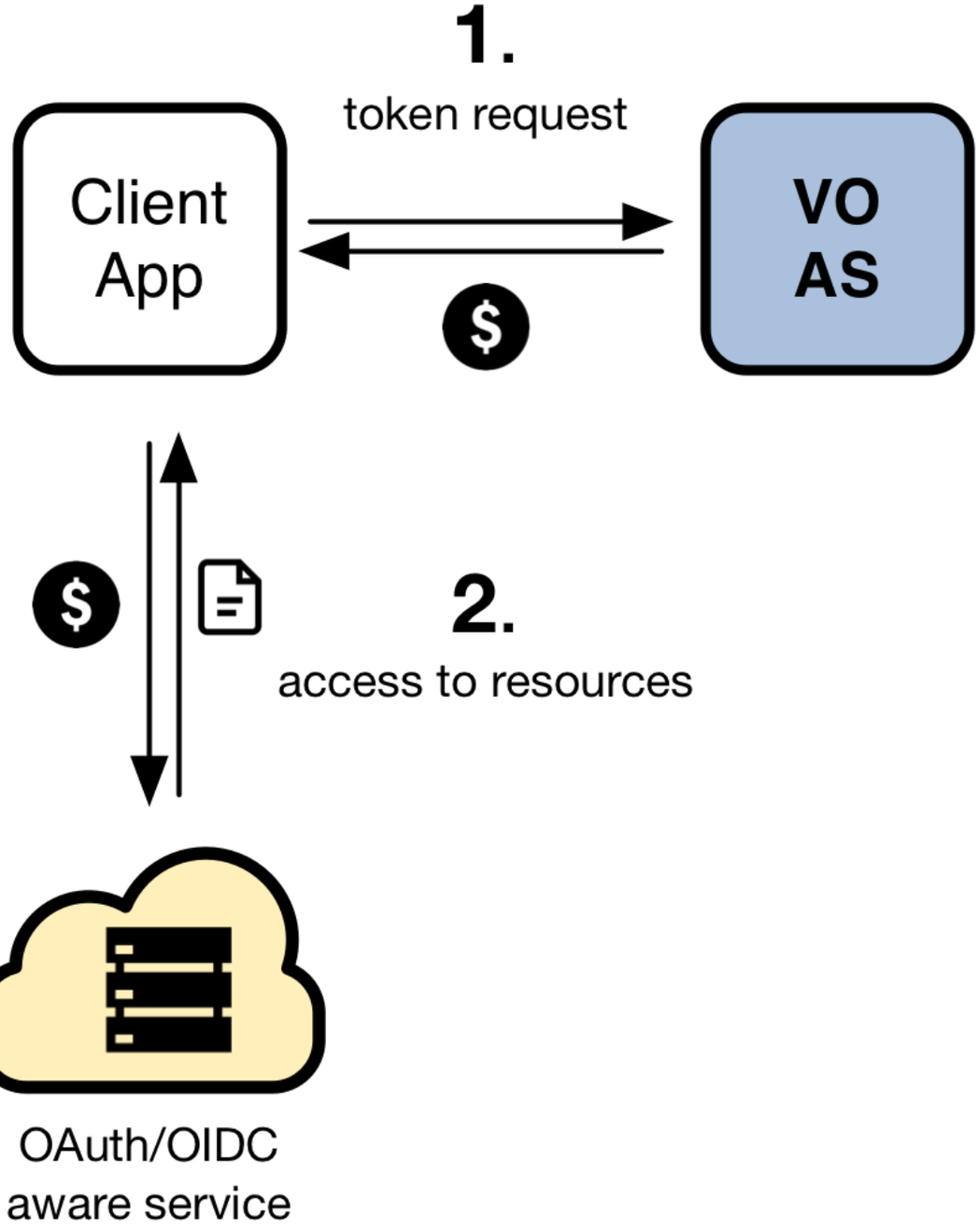
OAuth and OpenID Connect for WLCG

In order to access resources/services, a **client application** needs an **access token**

The token is obtained from **a VO** (which acts as an OAuth Authorization Server) using standard **OAuth/OpenID Connect** flows

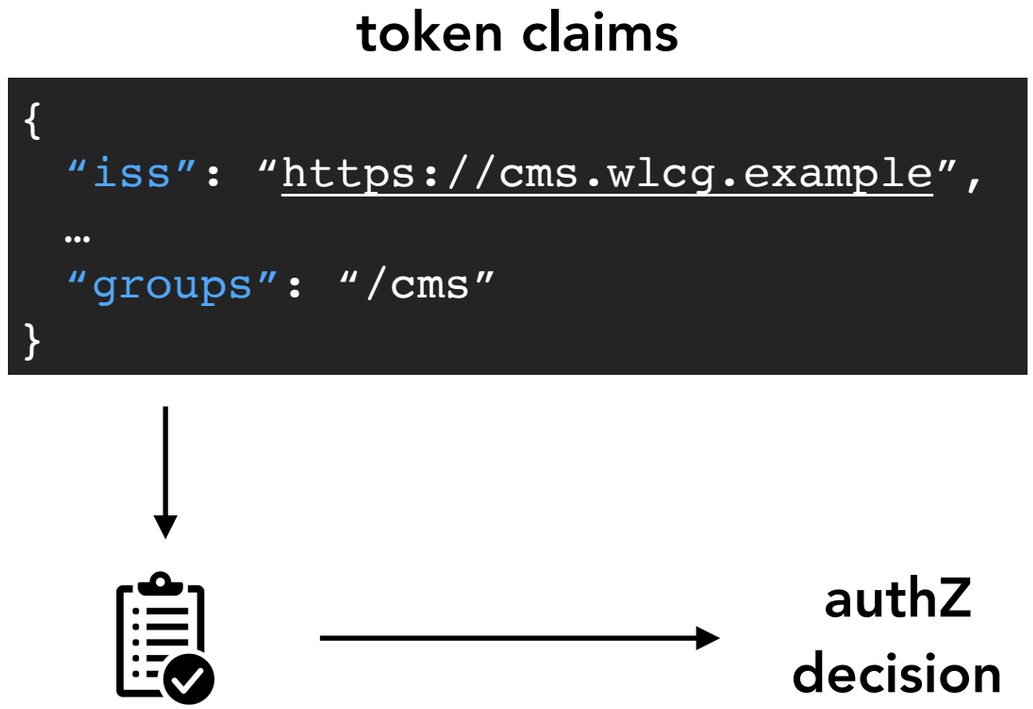
Authorization is then **performed at the services** leveraging info extracted from the token:

- **Identity attributes**: e.g., **groups**, roles, …
- **OAuth scopes**: capabilities linked to access tokens at token creation time

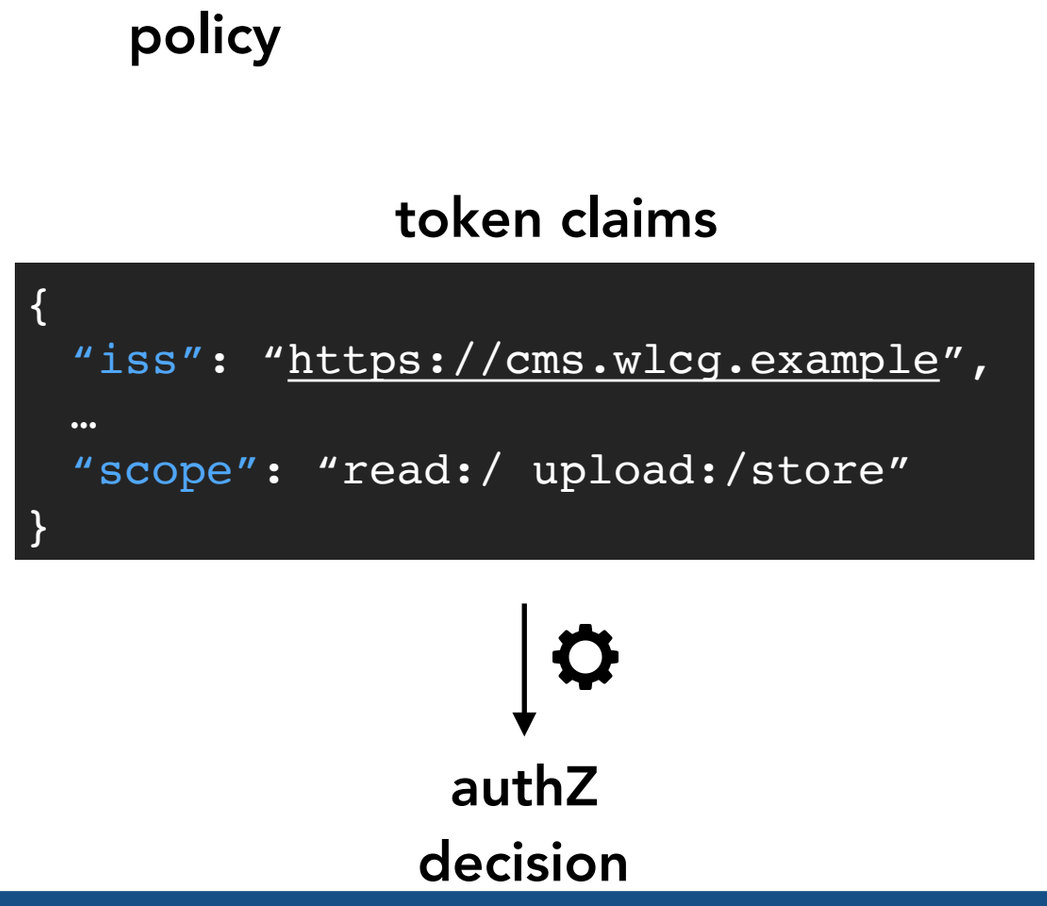


Identity-based vs Scope-based Authorization

Identity-based authorization: the token brings information about attribute ownership (e.g., groups/role membership), the service maps these attributes to a local authorization policy



Scope-based authorization: the token brings information about which actions should be authorized at a service, the service needs to understand these capabilities and honor them. The authorization policy is managed at the VO level

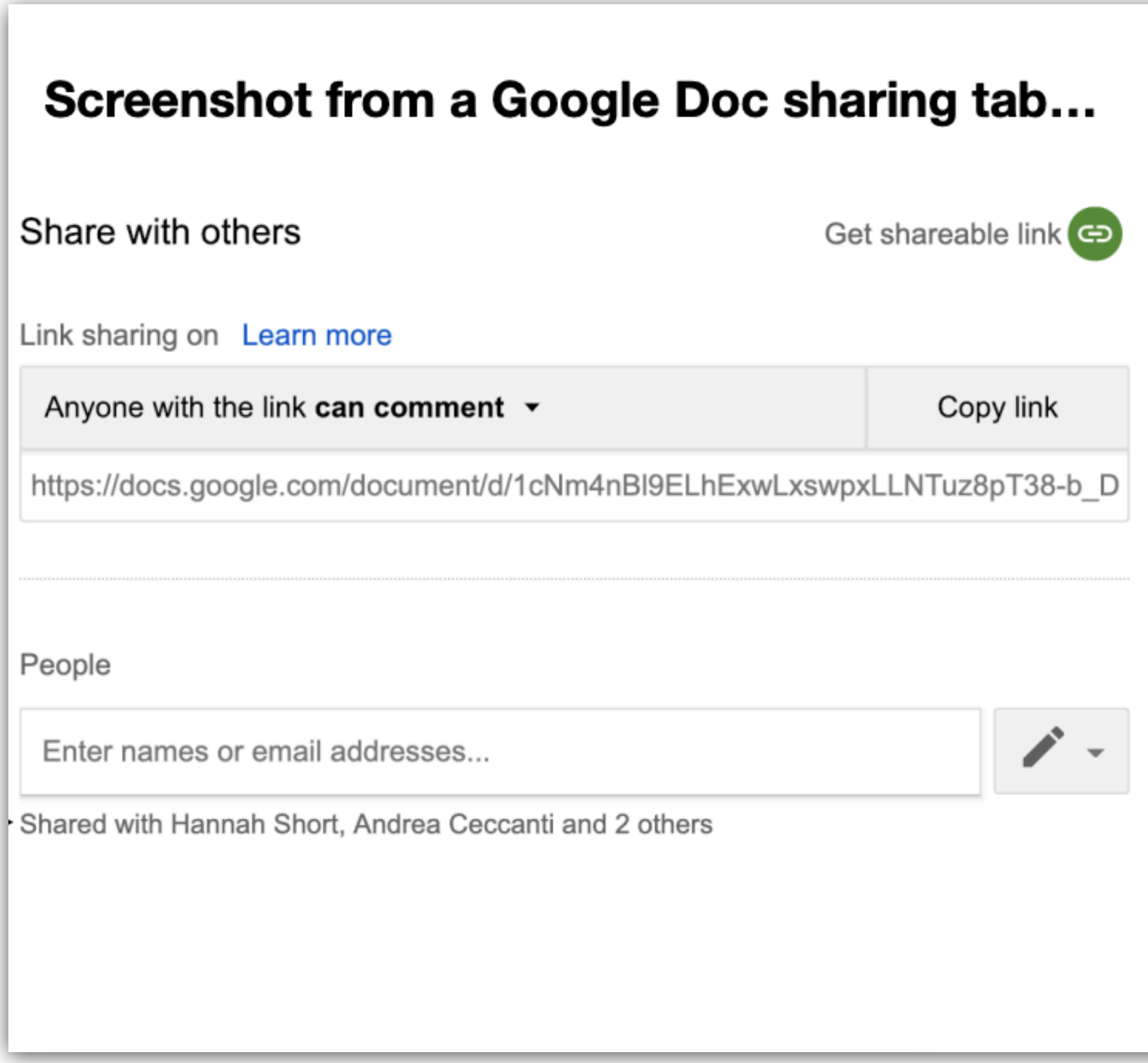


Identity-based vs Scope-based Authorization

The two models can coexist, even in the context of the same application!

scope-based authZ

identity-based authZ



* Slide courtesy of B. Bockelman

Token-based AuthN/AuthZ in practice

DOMA Third Party Copy

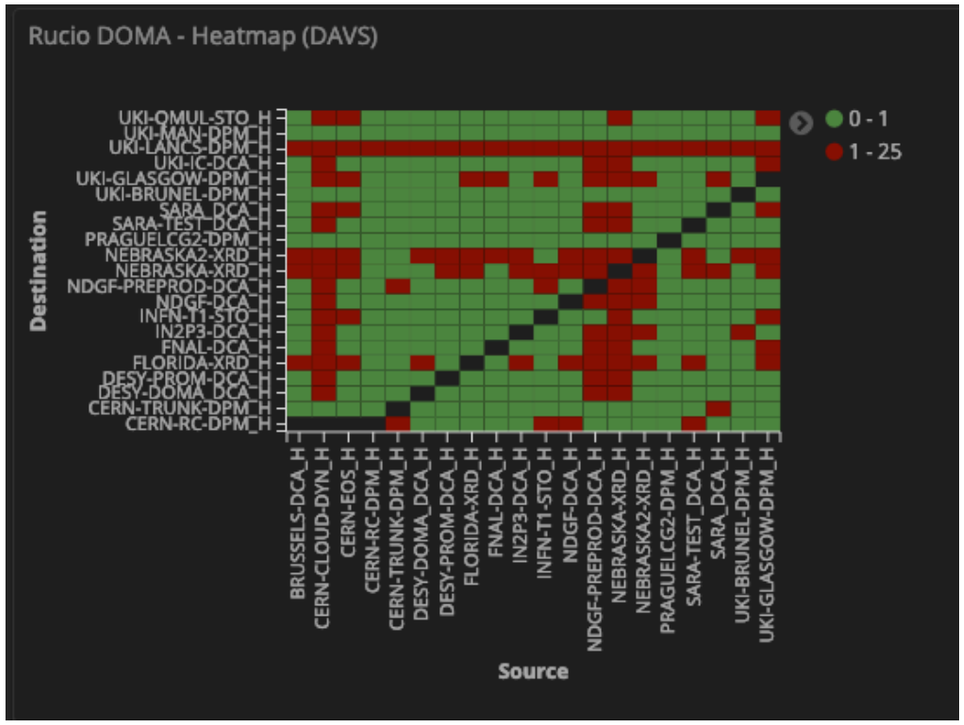
Token-based AuthN/Z in DOMA TPC WG

Bearer tokens used for authorization and delegation in HTTP third-party transfers across WLCG storage elements

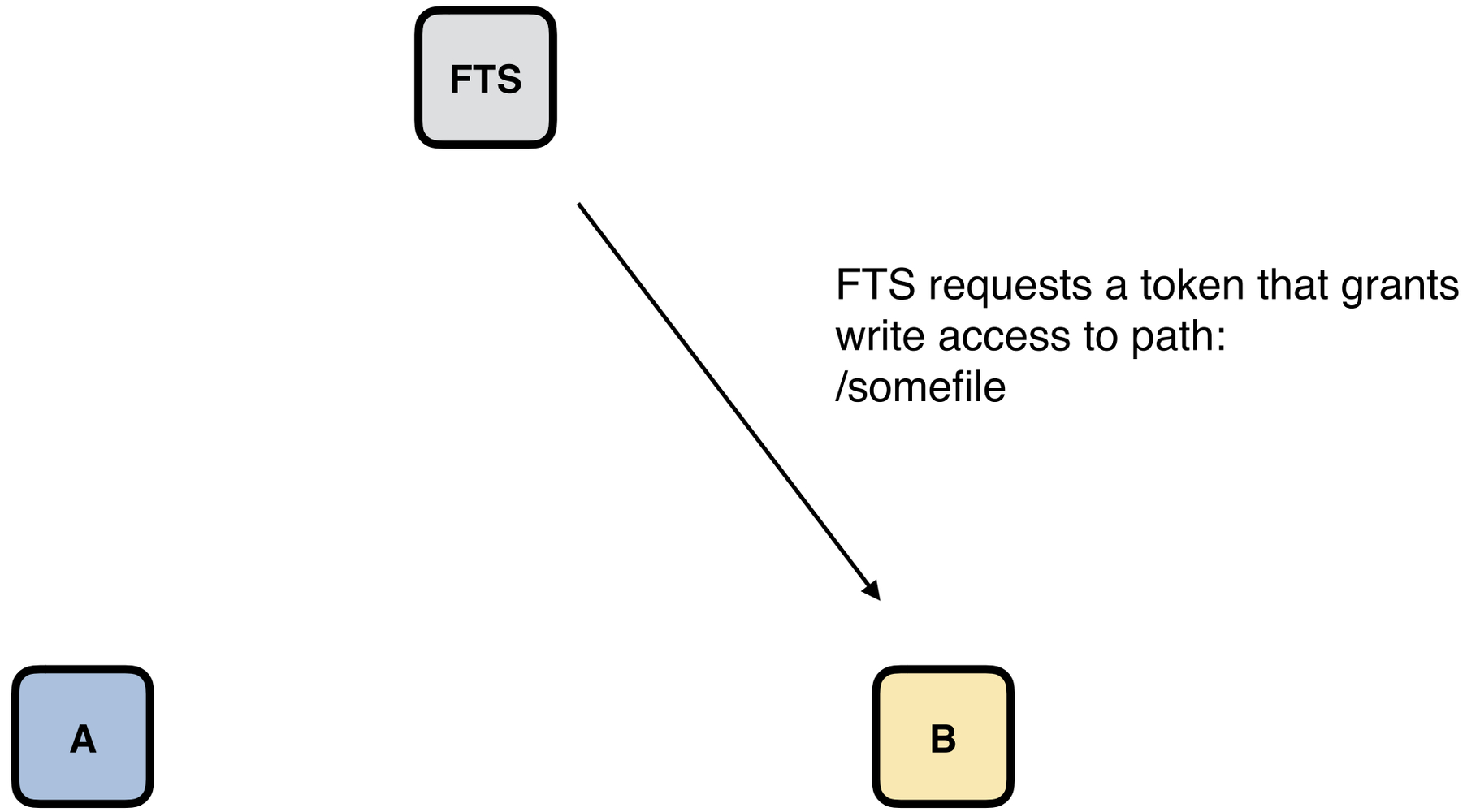
AuthZ already working fine for most SEs: dCache, DPM, StoRM, XRootD

FTS exchanges a VOMS proxy with an **SE-issued** authorization token, which grants (a subset of) the privileges granted by the VOMS proxy on such storage element

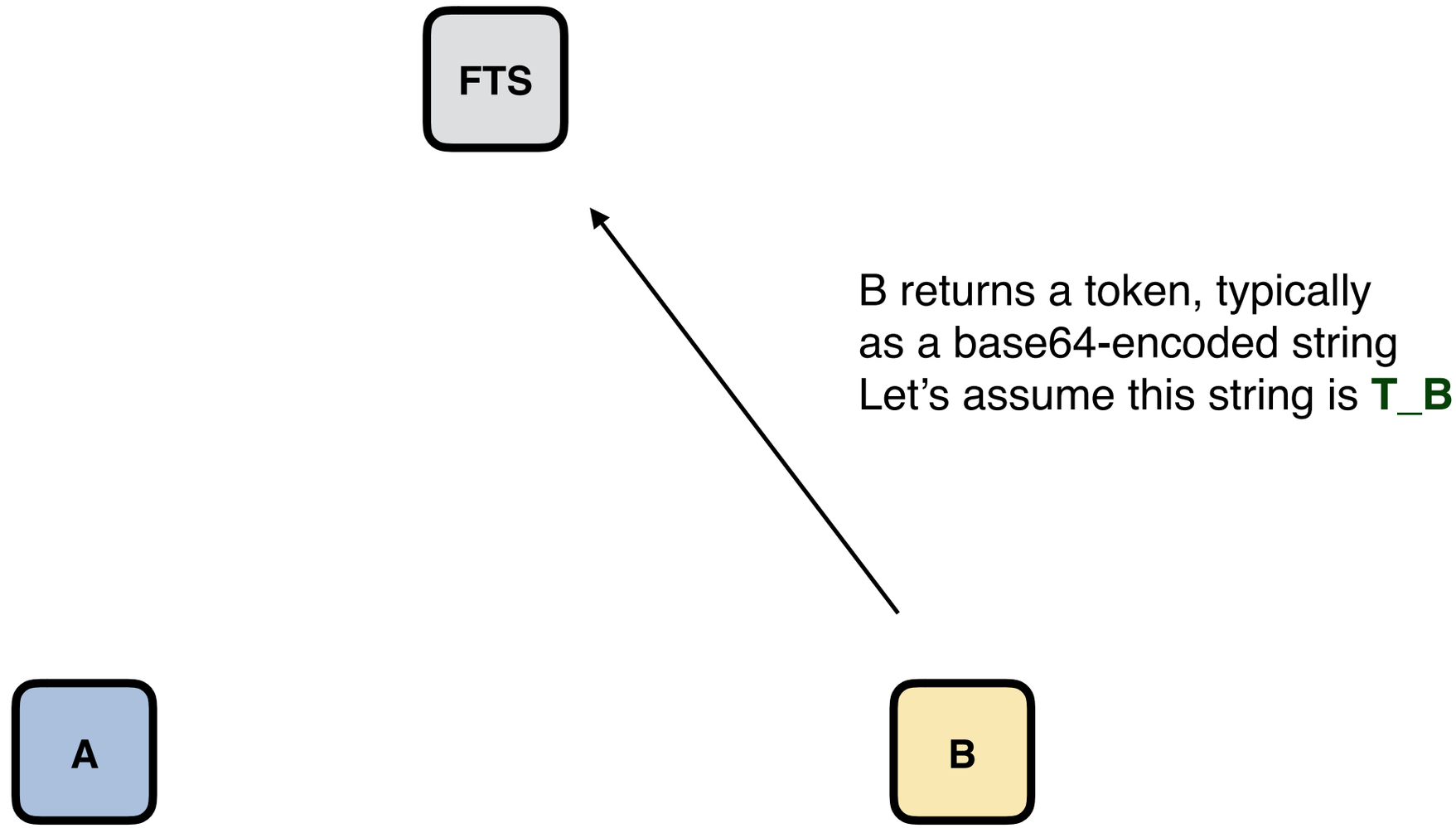
Agreement to converge on a common OAuth-based flow to request tokens



Token-based delegated AuthZ example

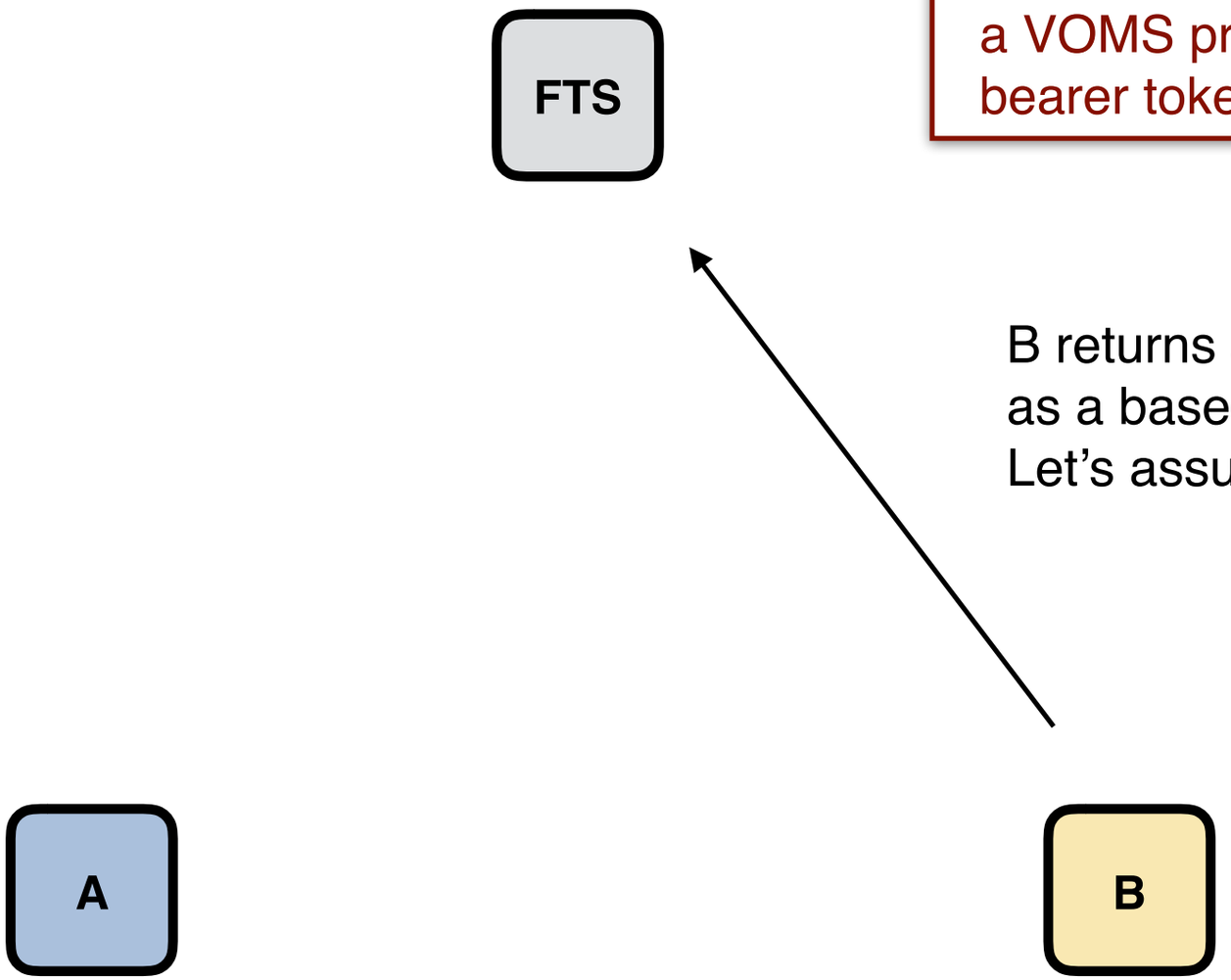


Token-based delegated AuthZ example

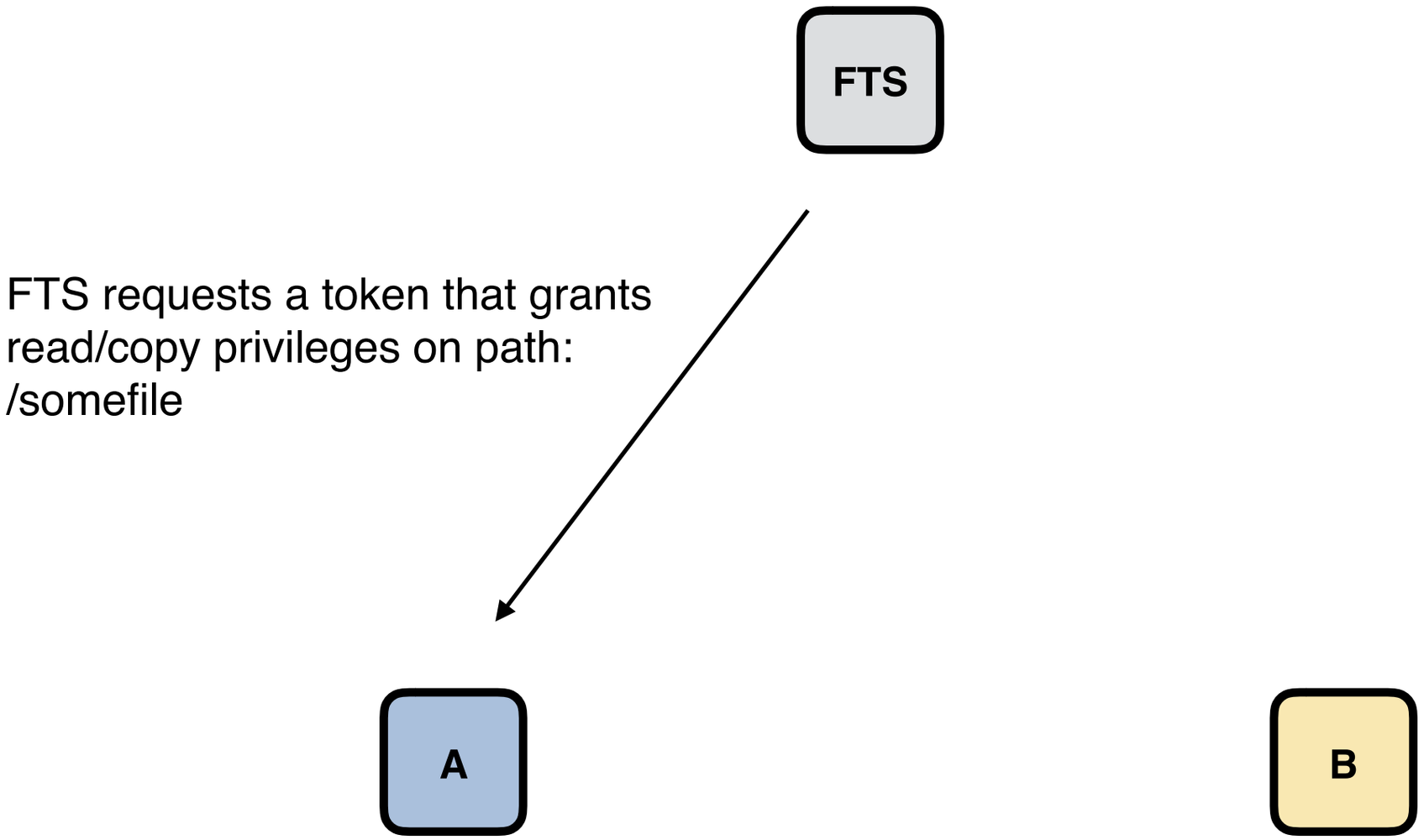


Token-based delegated AuthZ example

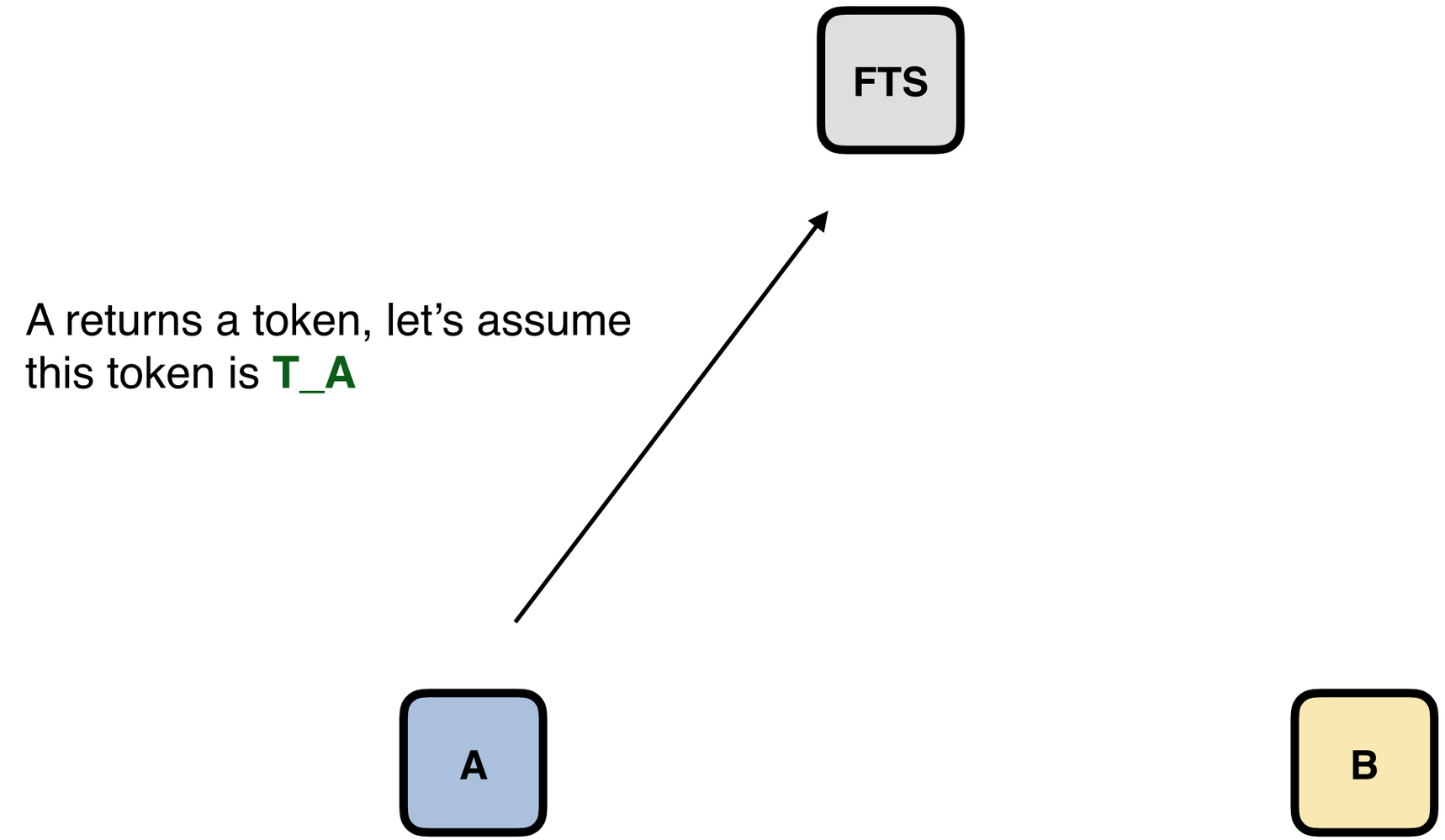
This is a **token translation step**:
FTS authenticates with
a VOMS proxy and gets back a
bearer token



Token-based delegated AuthZ example

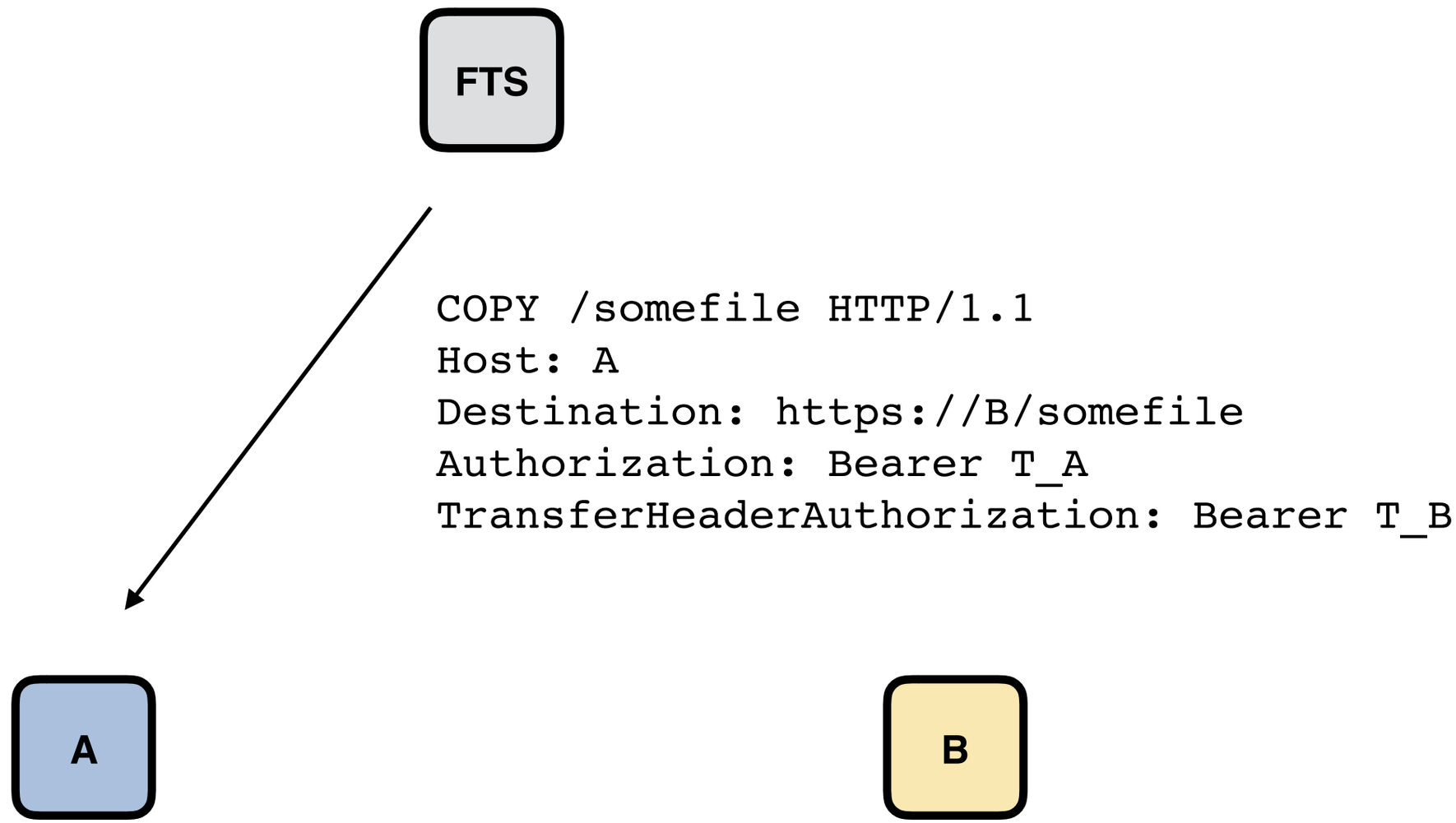


Token-based delegated AuthZ example



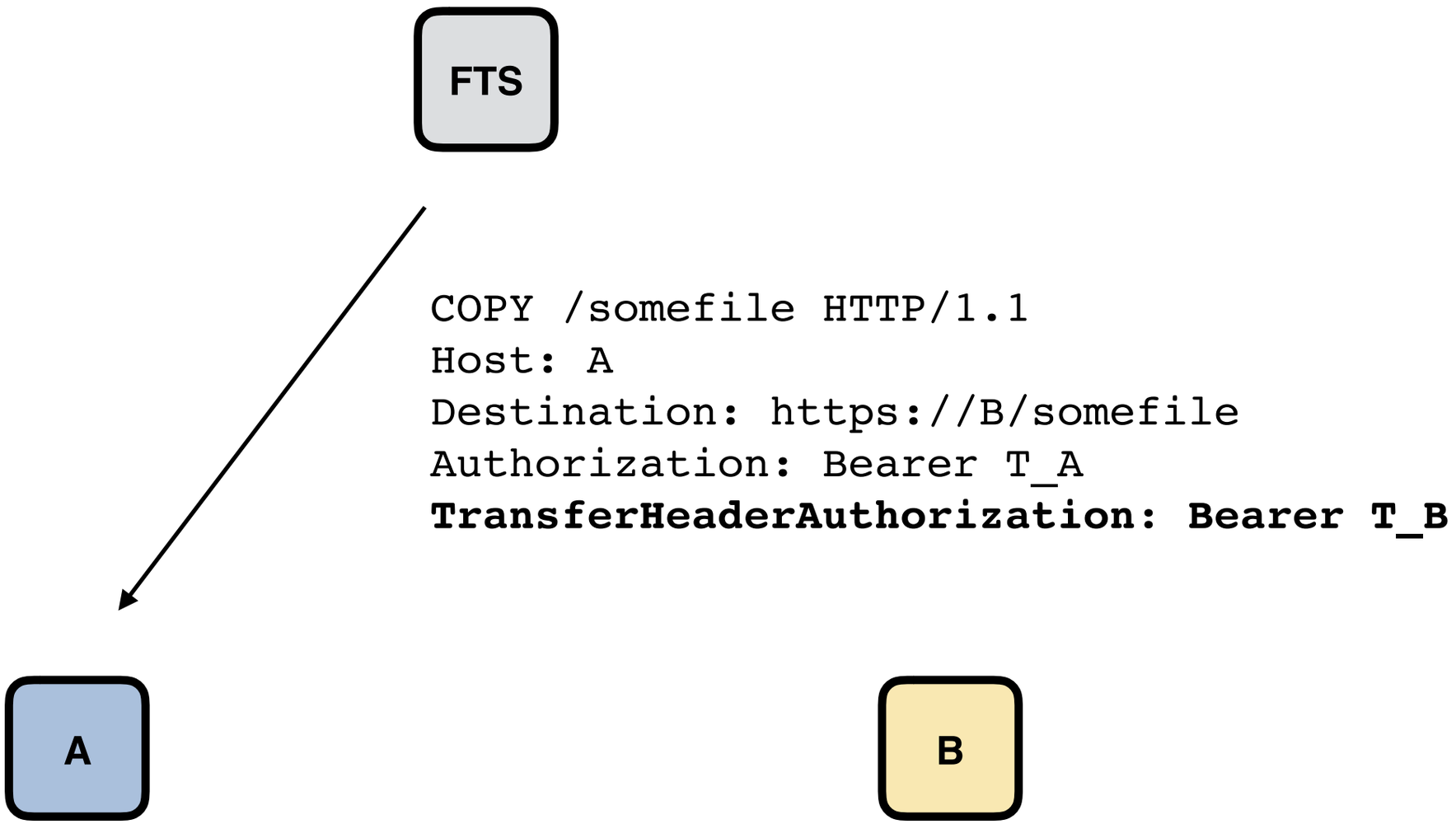
Token-based delegated AuthZ example

FTS can now request a TPC
from A/somefile to B/somefile

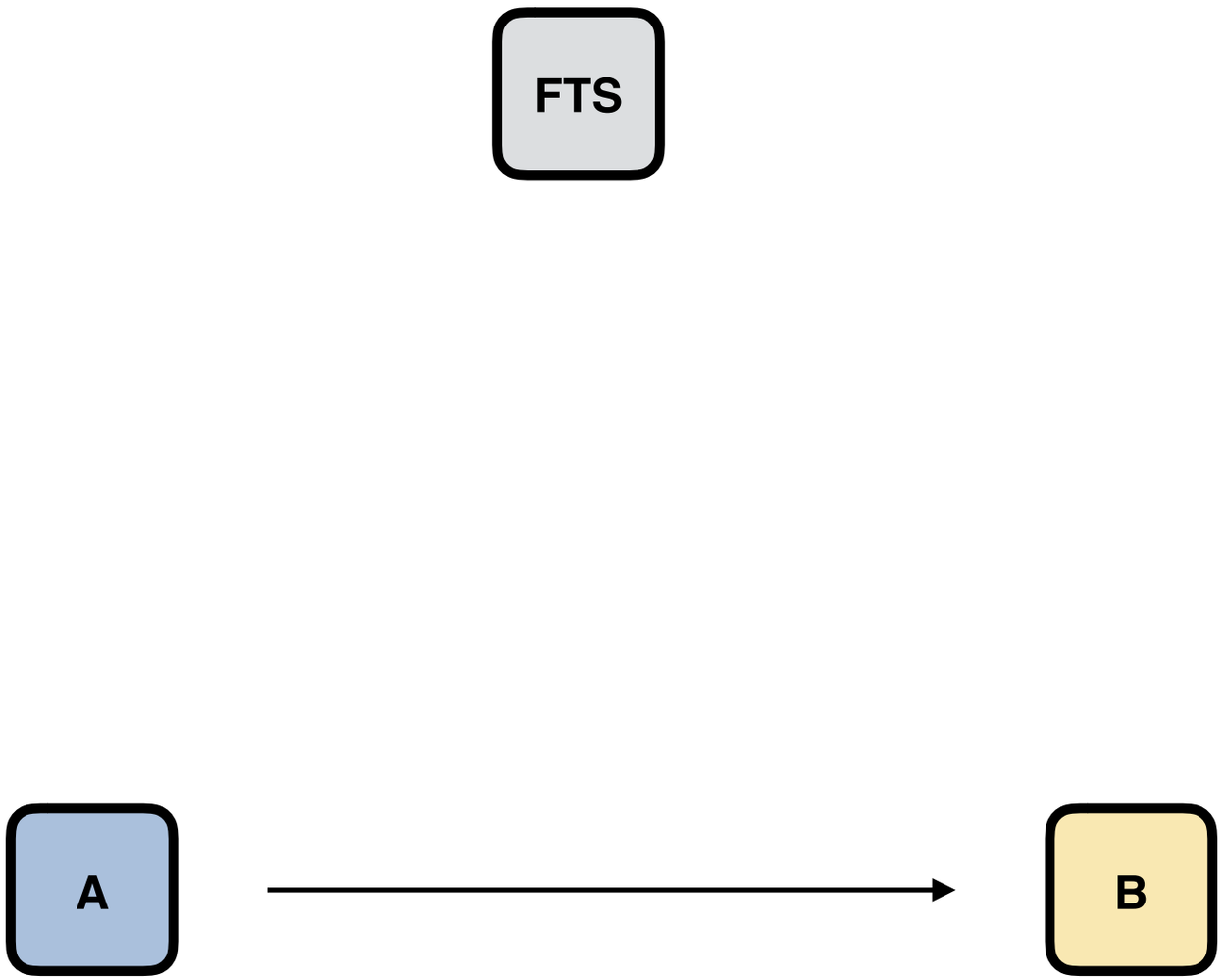


Token-based delegated AuthZ example

The protocol provides a way to request that certain headers in the COPY request are included in related transfer requests: all headers in the copy request starting with **TransferHeader** will be copied in the transfer request without such prefix.



Token-based delegated AuthZ example



Service-issued tokens vs VO-issued tokens

Service-issued tokens (like the ones used in DOMA TPC currently) only grant privileges on the service that issued the token

VO-issued tokens, OTOH, will grant access to all resources that trust the VO authorization server

- in a way similar to how VOMS attribute certificates grants access to all VO resources

In the future we will likely see both types of tokens used, and token translation services to translate between the two

- as it happens today at SEs to translate between VOMS ACs and an SE-issued tokens in support of DOMA TPC

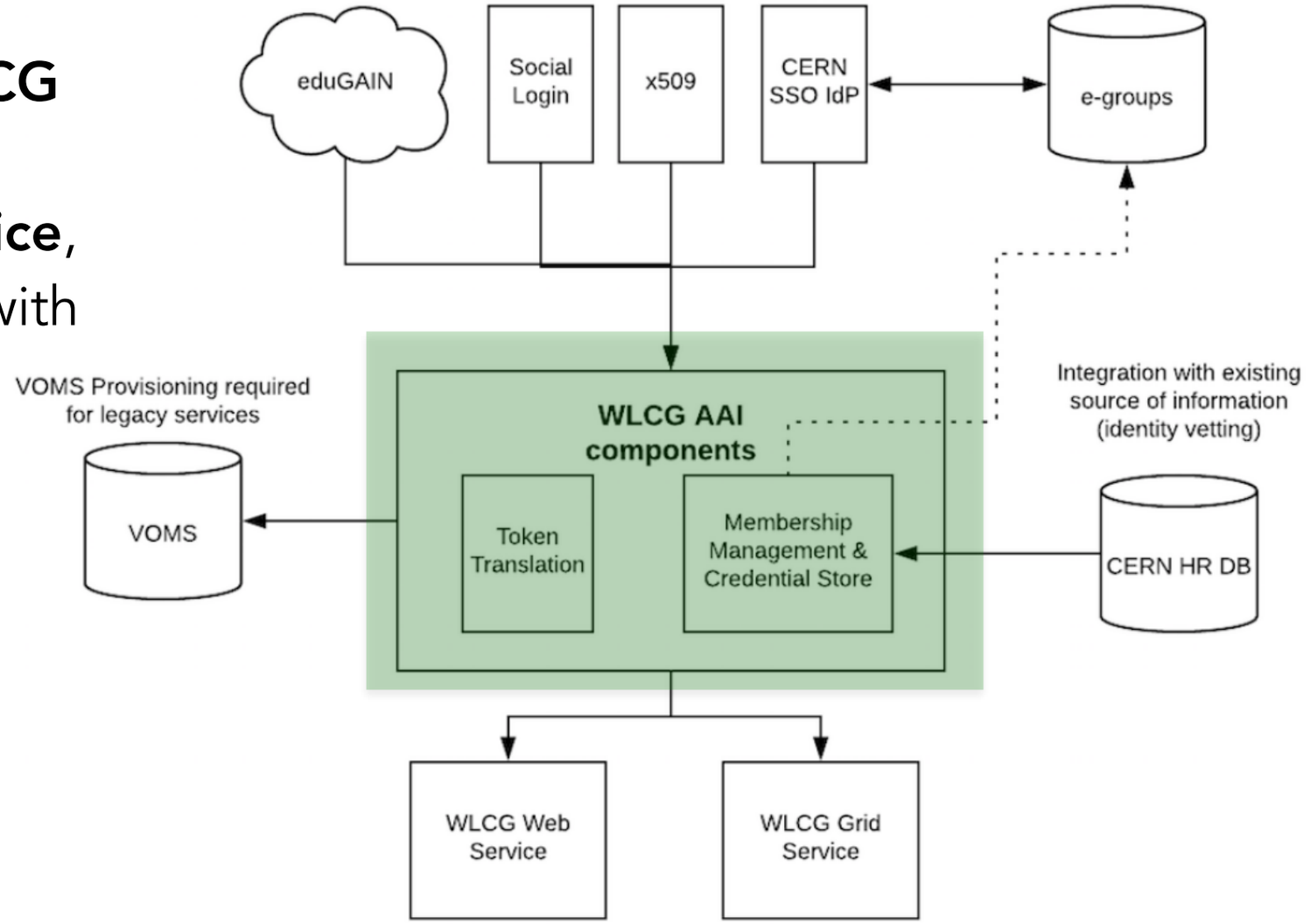
Standardization activities

The WLCG Authorization WG

https://twiki.cern.ch/twiki/bin/view/LCG/WLCGAuthorizationWG

Main objectives:

- Design and testing of a **WLCG Membership Management and Token Translation service**, facilitated by pilot projects with the support of AARC
- Definition of a **token-based authentication and authorization profile for WLCG**

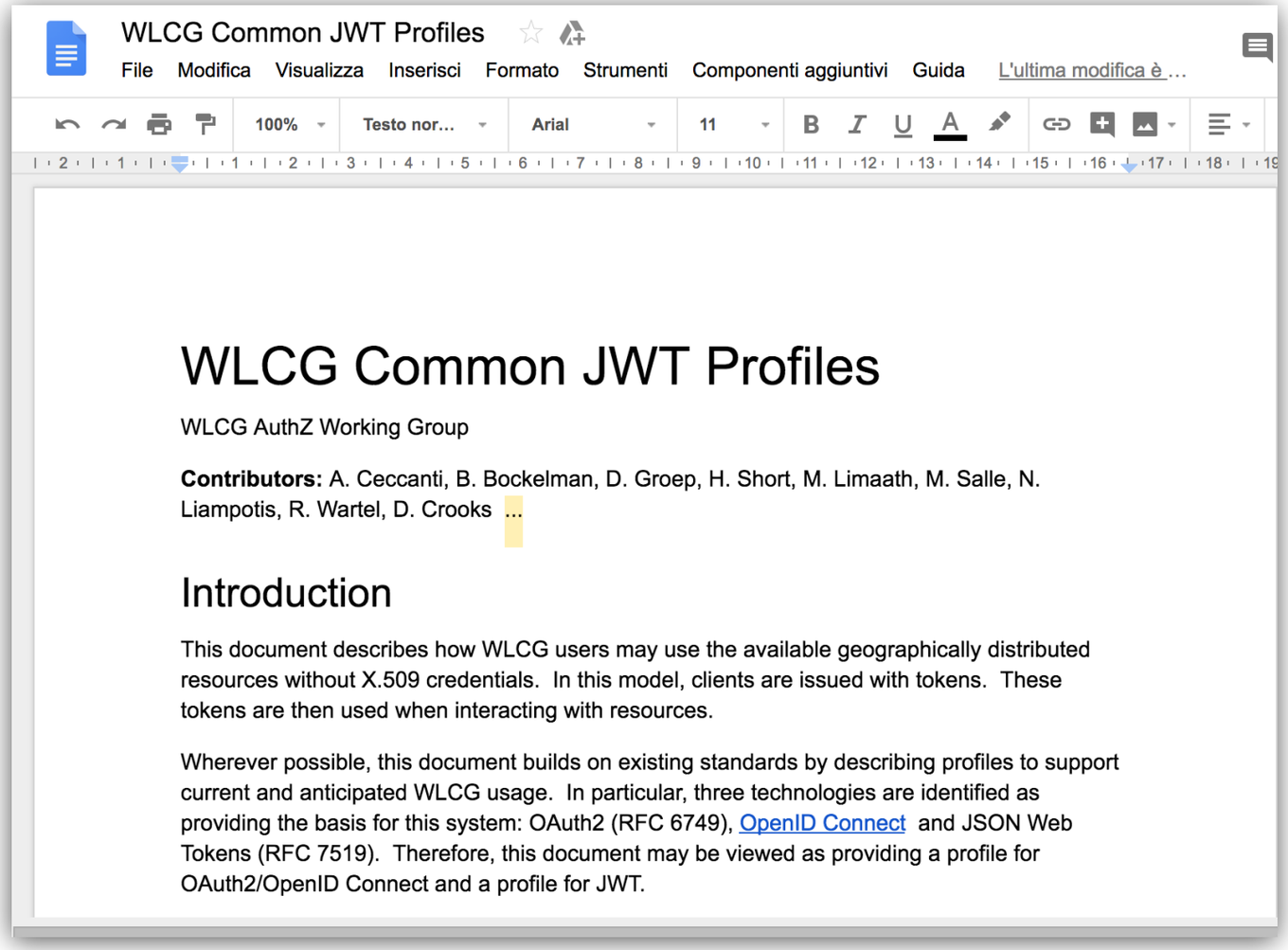


A common profile for Token-based AuthN/AuthZ

How is **authentication** and **authorization** information encoded in **identity** and **access tokens**?

How is **trust** established between parties exchanging tokens?

What's the recommended **token lifetime?**



Approach:

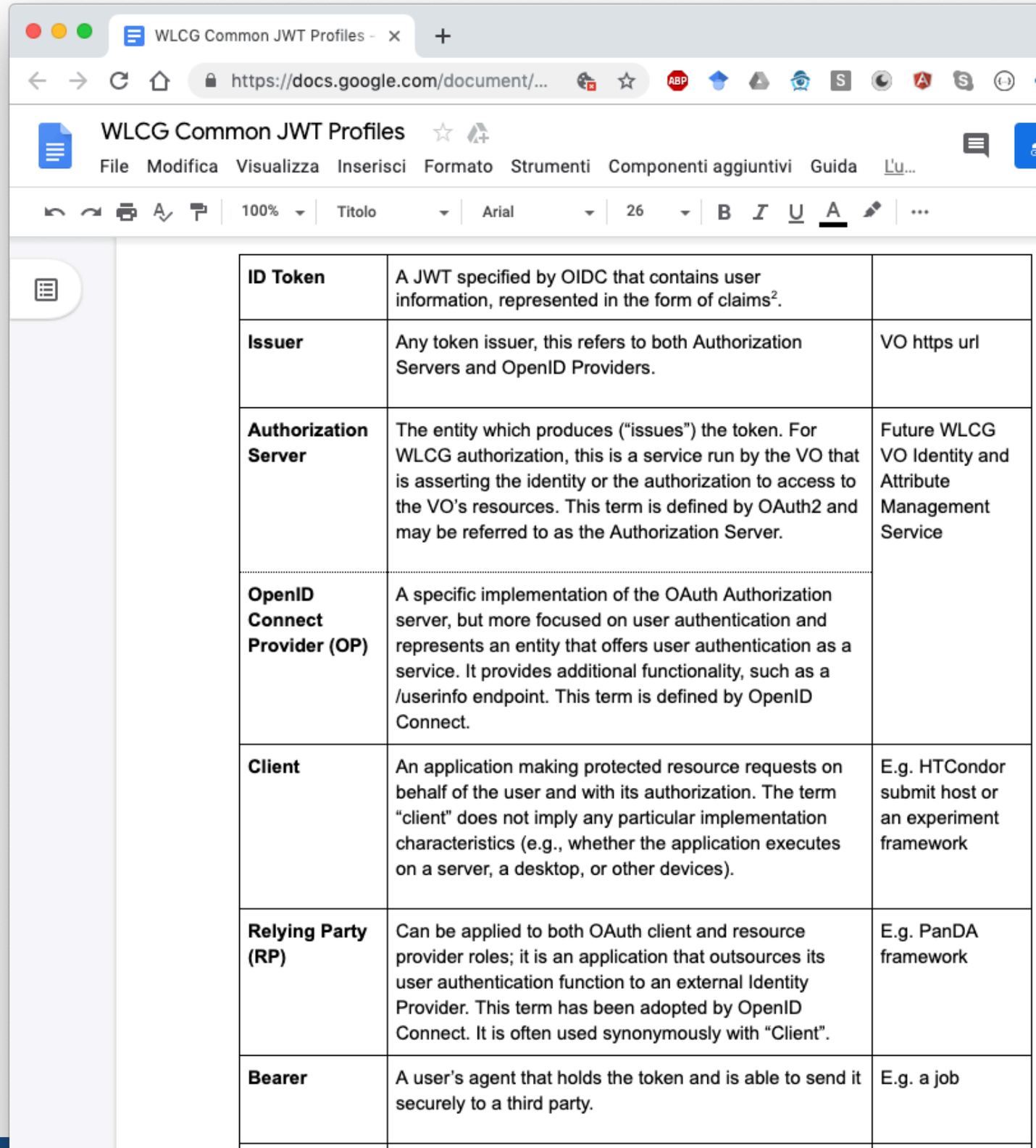
rely on existing standards as much as possible, extend only when needed

WLCG JWT profile: glossary

Define common terms and meaning

Leverage standard definitions wherever possible

Map general concepts to our use cases



ID Token	A JWT specified by OIDC that contains user information, represented in the form of claims[2].	
Issuer	Any token issuer, this refers to both Authorization Servers and OpenID Providers.	VO https url
Authorization Server	The entity which produces (“issues”) the token. For WLCG authorization, this is a service run by the VO that is asserting the identity or the authorization to access to the VO’s resources. This term is defined by OAuth2 and may be referred to as the Authorization Server.	Future WLCG VO Identity and Attribute Management Service
OpenID Connect Provider (OP)	A specific implementation of the OAuth Authorization server, but more focused on user authentication and represents an entity that offers user authentication as a service. It provides additional functionality, such as a /userinfo endpoint. This term is defined by OpenID Connect.	
Client	An application making protected resource requests on behalf of the user and with its authorization. The term “client” does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).	E.g. HTCondor submit host or an experiment framework
Relying Party (RP)	Can be applied to both OAuth client and resource provider roles; it is an application that outsources its user authentication function to an external Identity Provider. This term has been adopted by OpenID Connect. It is often used synonymously with “Client”.	E.g. PanDA framework
Bearer	A user’s agent that holds the token and is able to send it securely to a third party.	E.g. a job

WLCG JWT profile: token claims

What are the **required** claims to be included in access tokens and ID tokens, and what is the meaning.

Common claims: claims commons to access and ID tokens

ID token claims: claims specific to ID tokens (mainly focusing on user authentication and identity)

Access token claims: claims specific to access tokens (mainly focusing on authorization capabilities or attributes)

The profile mostly **reuses existing, standard** claims, with some WLCG specific additions. Additional, application-specific claims are allowed

WLCG specific token claims

wlcg.ver: the version of the WLCG token profile the relying party must understand to validate the token. Example:

```
wlcg.ver = “WLCG:1.0”
```

wlcg.groups: group information about an authenticated end-user, following a UNIX-like path syntax. Example:

```
wlcg.groups = {“/cms”, “/cms/itcms”}
```

Other claims used in the profile come from JWT and OpenID connect core standard

Scope-based authorization

OAuth provides **scopes** as a standard mechanism to express authorization permissions granted to client applications.

In practice, scopes are a set of strings included in an access token that **limit what are the operations that can be authorized by clients presenting such access token.**

OAuth scopes are commonly used in industry to define the authorization on service APIs. Examples:

https://api.slack.com/docs/oauth-scopes

https://developer.github.com/apps/building-oauth-apps/understanding-scopes-for-oauth-apps/#available-scopes

https://developers.google.com/identity/protocols/googlescopes

WLCG OAuth scopes

Work in progress: can change!

Building on the SciTokens experience, define scopes that would match our computing use-cases. First use case: **storage access**

storage.read: Read data. Only applies to "online" resources such as disk (as opposed to "nearline" such as tape where the storage.stage authorization should be used in addition).

storage.modify: Change data. This includes renaming files and writing data. This permission includes overwriting or replacing stored data in addition to deleting or truncating data.

storage.create: Upload data. This includes renaming files if the destination file does not already exist. This authorization DOES NOT permit overwriting or deletion of stored data.

storage.stage: Cause data to be staged from a nearline resource to an online resource.

Storage scopes and resource paths

Storage scopes may additionally provide a resource path*, which further limits the authorization. The resource path is provided respecting the following format:

scope:path

Examples:

storage.read:/

storage.write:/protected

 * the path is required for selected scopes, more details in the profile

Path semantics

Work in progress: can change!

Following the Scitokens model, permissions granted on a path **apply transitively to subpaths,** e.g.:

storage.read:/cms

grants read access to the /cms directory and to all its content, but does not grant read access to the /atlas directory.

This approach is **not equivalent** with POSIX semantics, but matches well with our experiments data access authorization models.

Path semantics

Work in progress: can change!

Following the Scitokens model, permissions granted on a path **apply transitively to subpaths,** e.g.:

storage.read:/cms

grants read access to the /cms directory and to all its content, but does not grant read access to the /atlas directory.

This approach is **not equivalent** with POSIX semantics, but matches well with our experiments data access authorization models.

Note that implementing this semantic is up to client applications, i.e. dCache, DPM, EOS, StoRM, EOS, XRootD, etc…., the token just provides a (signed) string!

The WebDAV interface exercise

To understand whether the currently defined capabilities match well with a real world storage access protocol implementation, we started matching the above scopes to the WebDAV interface currently implemented by many SEs

Discussion ongoing in the WLCG Authz WG
Feedback is appreciated!

Scope-based attribute selection

Work in progress: can change!

Use scopes to implement a group selection mechanism for groups equivalent to the one provided by VOMS, following the approach outlined in the OpenID Connect standard.

Two types of groups:

- **Default groups:** whose membership is always asserted (similar to VOMS groups)
- **Optional groups:** whose membership is asserted only when explicitly requested by the client application (similar to VOMS roles)

Scope-based attribute selection

Work in progress: can change!

A parametric `wlcg.groups` scope is introduced with the following form:

`wlcg.groups[:<group_name>]?`

With the following rules:

- If the scope does not have the parametric part, i.e. its value is `wlcg.groups`, the authorization server will return the list of default groups for the user being authenticated for the target client.
- if the scope is parametric, i.e. it has the form `wlcg.groups:<group_name>`, in addition to the default groups as described in the previous point, the authorization server will also return the requested group as a value in the `wlcg.groups` claim if the user is member of such group.

Scope-based attribute selection

Work in progress: can change!

…with the following rules:

- To request multiple groups, multiple `wlcg.groups:<group_name>` scopes are included in the authorization request
- The order of the groups in the returned `wlcg.groups` claim complies with the order in which the groups were requested
- the returned groups claim will not contain duplicates

This seems complex, but it’s the attribute selection mechanism we use everyday with VOMS

Scope-based attribute selection

Work in progress: can change!

…with the following rules:

- To request multiple groups, multiple `wlcg.groups:<group_name>` scopes are included in the authorization request
- The order of the groups in the returned `wlcg.groups` claim complies with the order in which the groups were requested
- the returned groups claim will not contain duplicates

This seems complex, but it's the attribute selection mechanism we use everyday with VOMS

Note that implementing this semantic is (mostly) up to the WLCG AuthZ server (i.e., IAM).

Scope-based attribute selection: examples

An authorization request with the following scope:

```
scope=wlcg.groups:/cms/uscms wlcg.groups:/cms/ALARM wlcg.groups
```

will return the following wlcg.groups claim

```
"wlcg.groups": ["/cms/uscms","/cms/ALARM", “/cms"]
```

assuming /cms is the only default group defined at the authorization server

Trust & security

The profile document also provides recommendations on token lifetimes and trust establishment and other important aspects

Token Type	Recommended Lifetime	Minimum Lifetime	Maximum Lifetime	Justification
Access Token & ID Token	20 minutes	5 minutes	6 hours	Access token lifetime should be short as there is no revocation mechanism. The granted lifetime has implications for the maximum allowable downtime of the Access Token server.
Refresh Token	10 days	1 day	30 days	Refresh token lifetimes should be kept bounded, but can be longer-lived as they are revocable. Meant to be long-lived enough to be on a "human timescale".
Issuer Public Key Cache	6 hours	1 hour	1 day	The public key cache lifetime defines the minimum revocation time of the public key. The actual lifetime is the maximum allowable downtime of the public key server
Issuer Public Key	6 months	2 days	12 months	JWT has built-in mechanisms for key rotation; these do not need to live as long as CAs. This may evolve following operational experience, provision should be made for flexible lifetimes.

* Slide courtesy of H. Short

The INDIGO IAM service

INDIGO Identity and Access Management service

Flexible authentication support

- (SAML, X.509, OpenID Connect, username/password, …)

Account linking

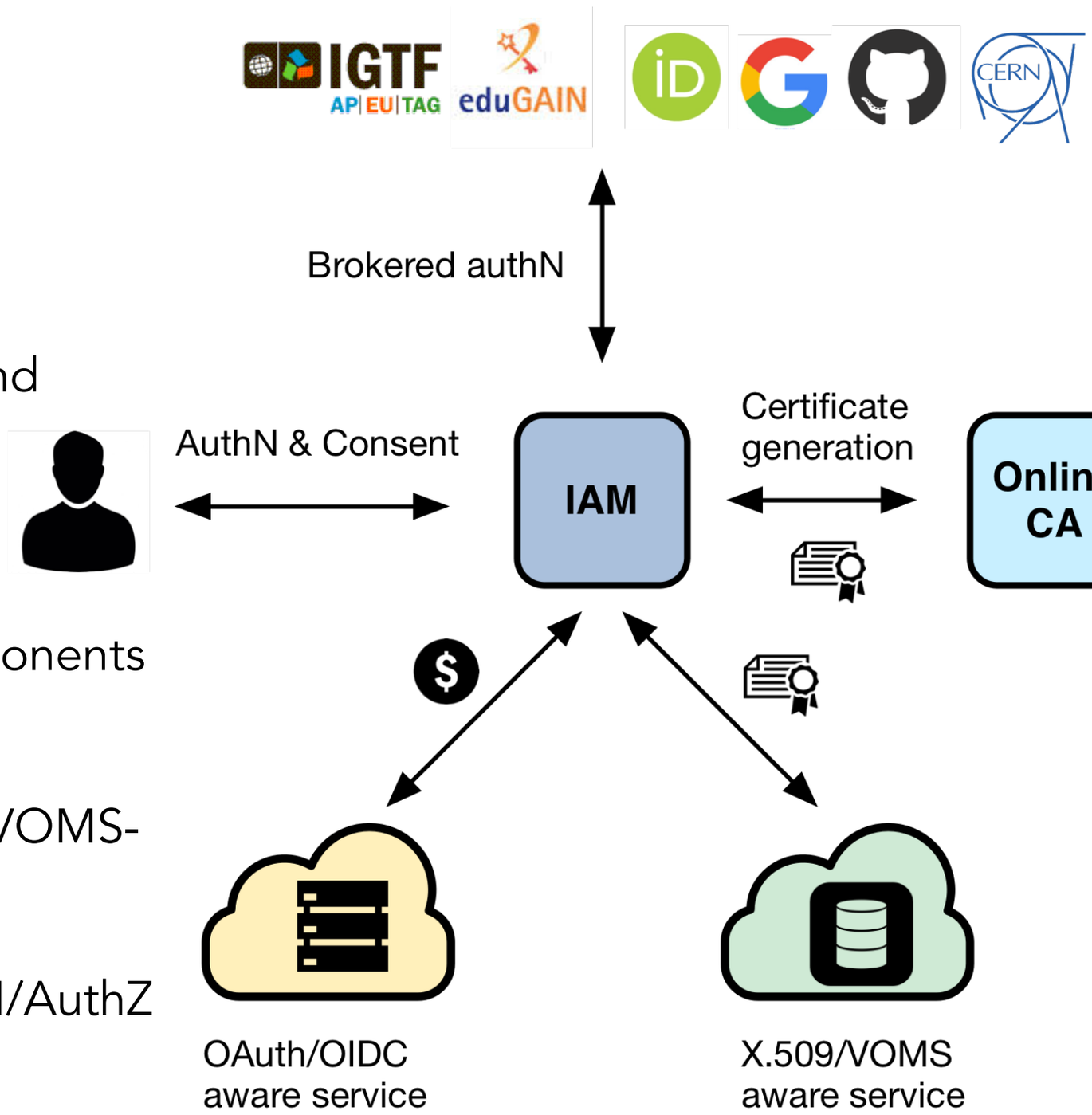
Registration service for moderated and automatic user enrollment

Enforcement of AUP acceptance

Easy integration in off-the-shelf components thanks to **OpenID Connect/OAuth**

VOMS support, to integrate existing VOMS-aware services

Self-contained, comprehensive AuthN/AuthZ solution



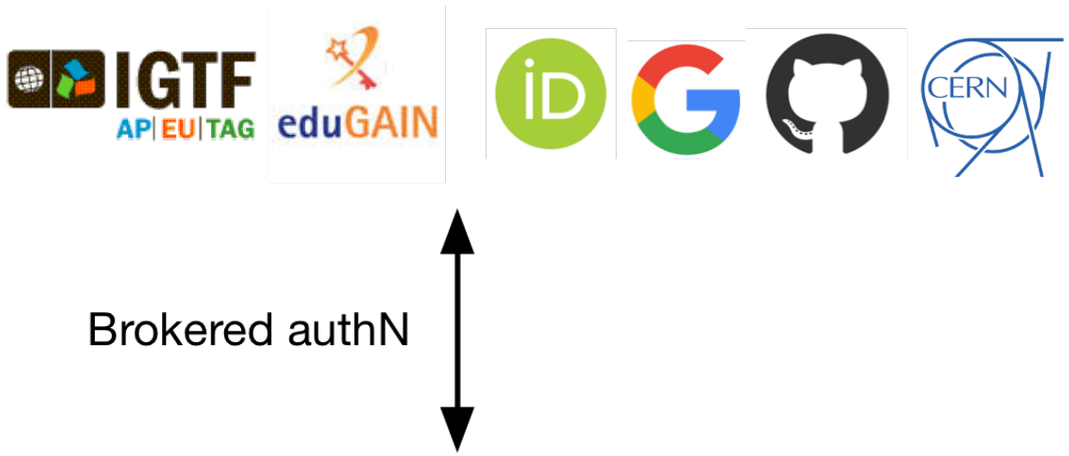
INDIGO Identity and Access Management service

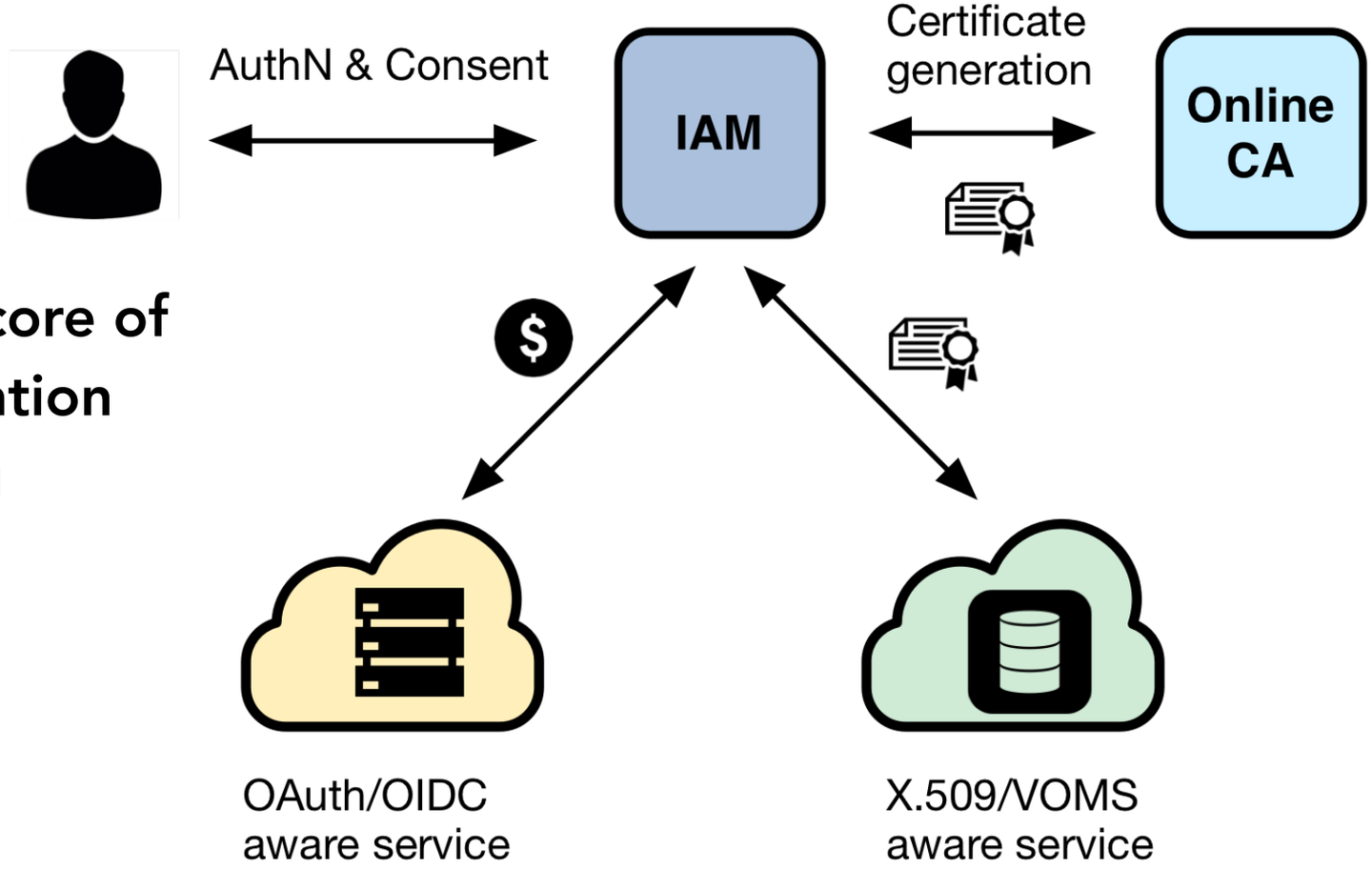
Originally developed in the context of the INDIGO DataCloud project

Sustained by INFN for the foreseeable future with support from:

- EOSC-Hub
- ESCAPE

Selected by WLCG to be the at the core of the next-generation WLCG authorization service in support of LHC computing





Testing integration with IAM

IAM has already been integrated with off-the-shelf components (Openstack, Kubernetes, …) and WLCG middleware

- DODAS,FTS, dCache, StoRM WebDAV
- RUCIO (integration ongoing now)

Integration with IAM can be tested **NOW**

Thanks for your attention.

Questions?

Useful references

IAM @ GitHub: https://github.com/indigo-iam/iam

IAM documentation: https://indigo-iam.github.io/docs

WLCG Authorization WG: https://twiki.cern.ch/twiki/bin/view/LCG/WLCGAuthorizationWG

WLCG AuthZ WG Demos: https://indico.cern.ch/event/791175/attachments/1806605/2948665/demos.mp4 (IAM starts at minute 46)

IAM in action video: https://www.youtube.com/watch?v=1rZlvJADOnY

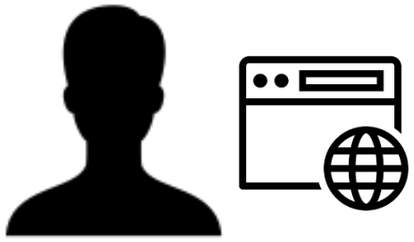
Contacts:

- andrea.ceccanti@cnaf.infn.it
- indigo-aai.slack.com

Backup slides

Web application integration scenario

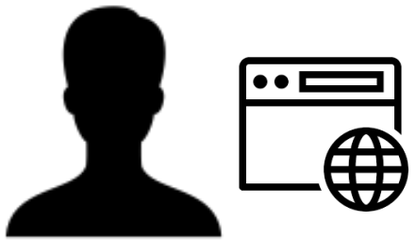
Web application: authorization code flow



A Web App integrates with IAM to **delegate user authentication management** and **obtain authorization** information

Home IdP

Web application: authorization code flow



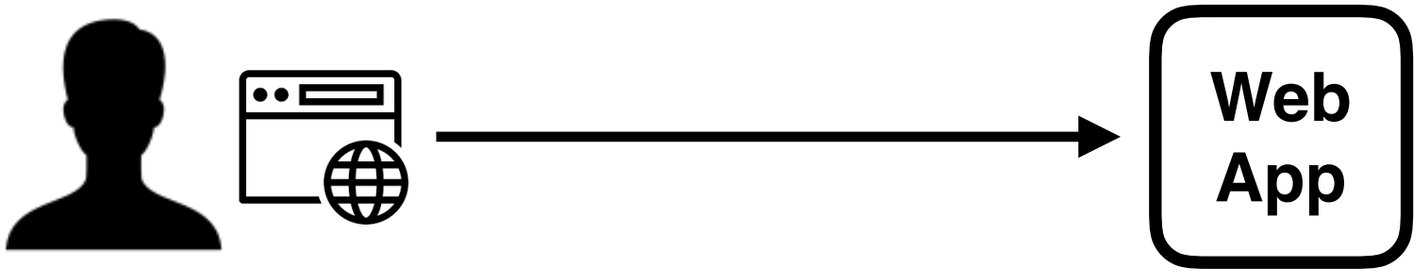
Web App

OAuth and OpenID connect provide the **authorization code flow** in support of this integration use case

Home IdP

IAM

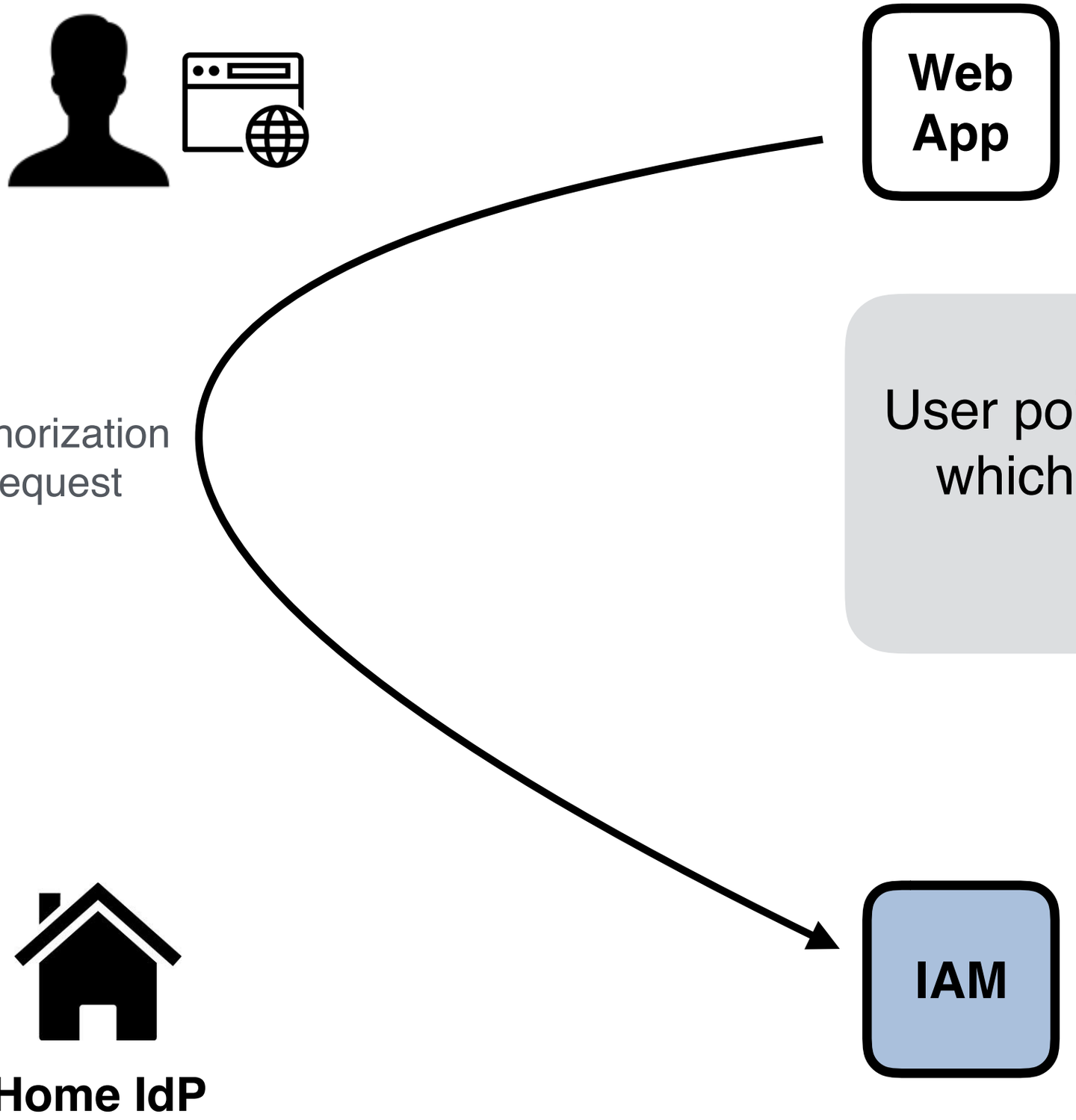
Authorization code flow



User points its browser to web app, which redirects back to IAM for authentication

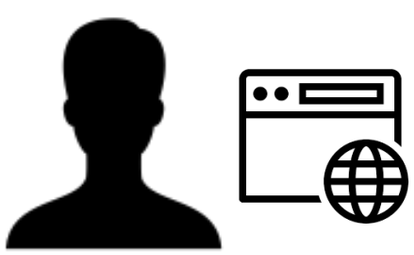
Home IdP

Authorization code flow



User points its browser to web app, which redirects back to IAM for authentication

Authorization code flow



Web App

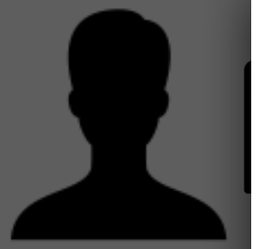
authorization request

User does not have a valid session at IAM, so IAM shows the login page

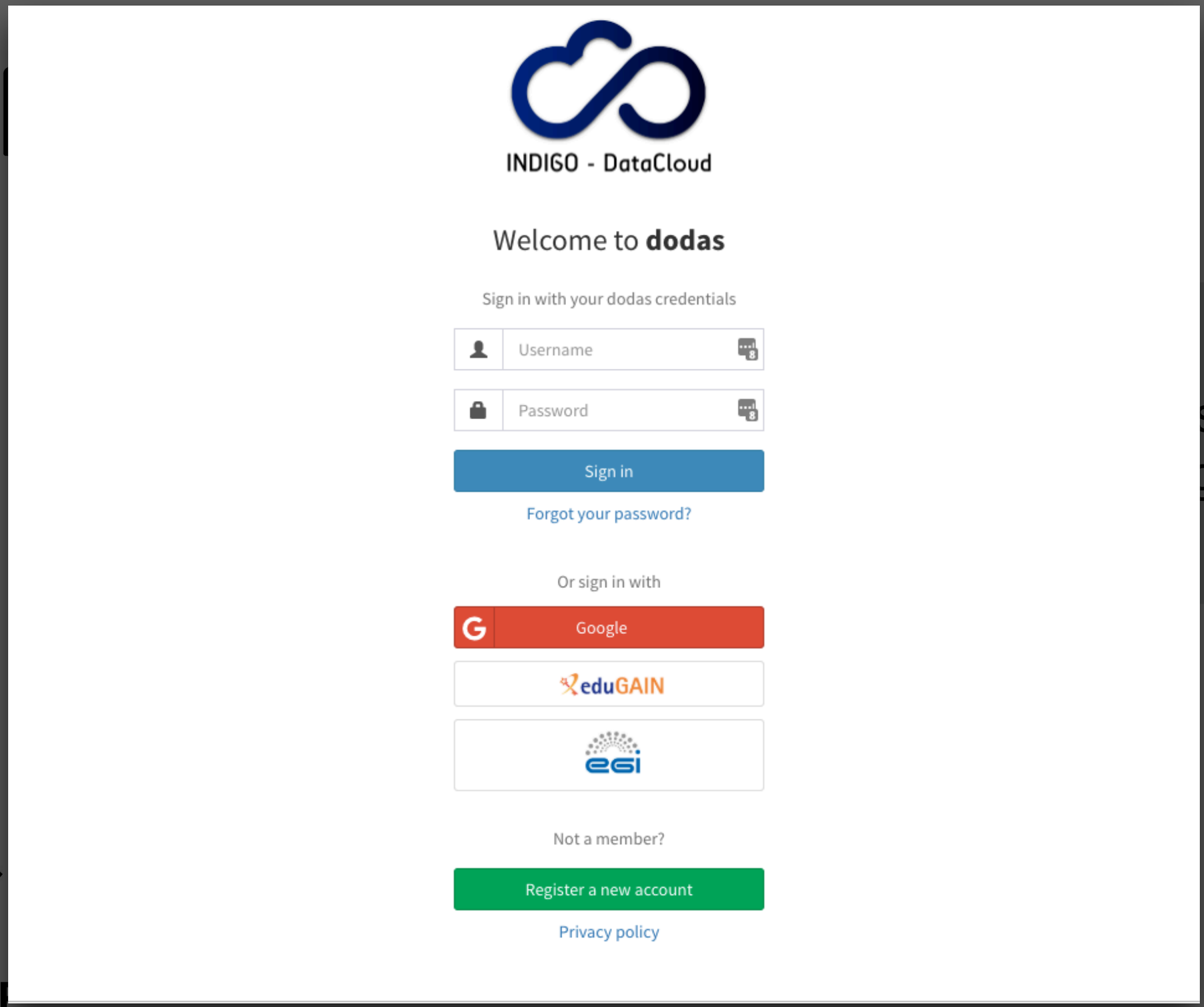
Home IdP

IAM

Authorization code flow



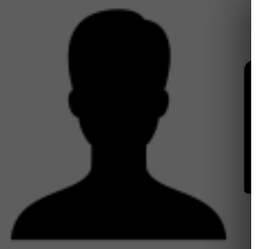
authorization request



session at gin page

Home Idp

Authorization code flow



auth

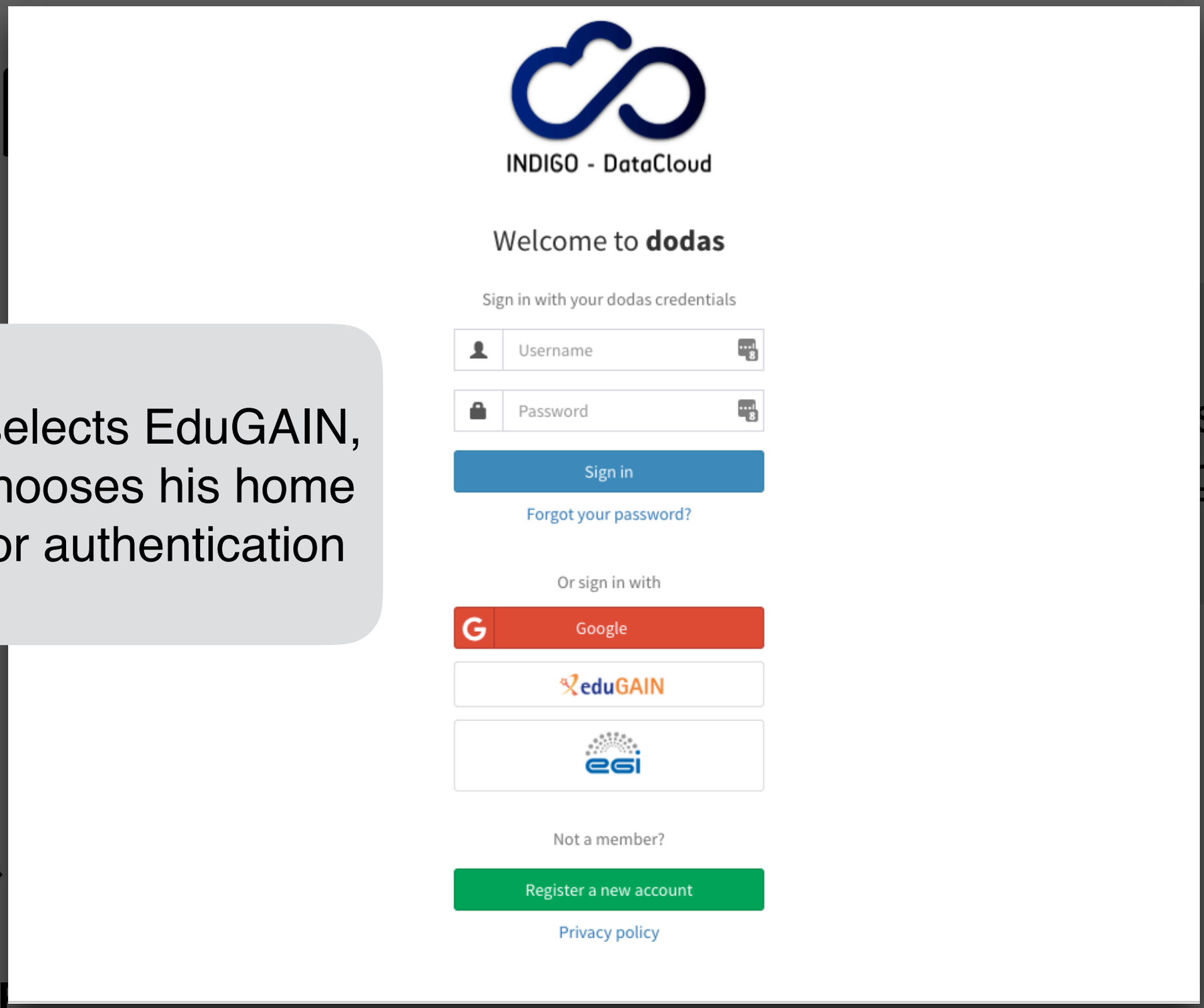
r

User selects EduGAIN, and chooses his home IDP for authentication

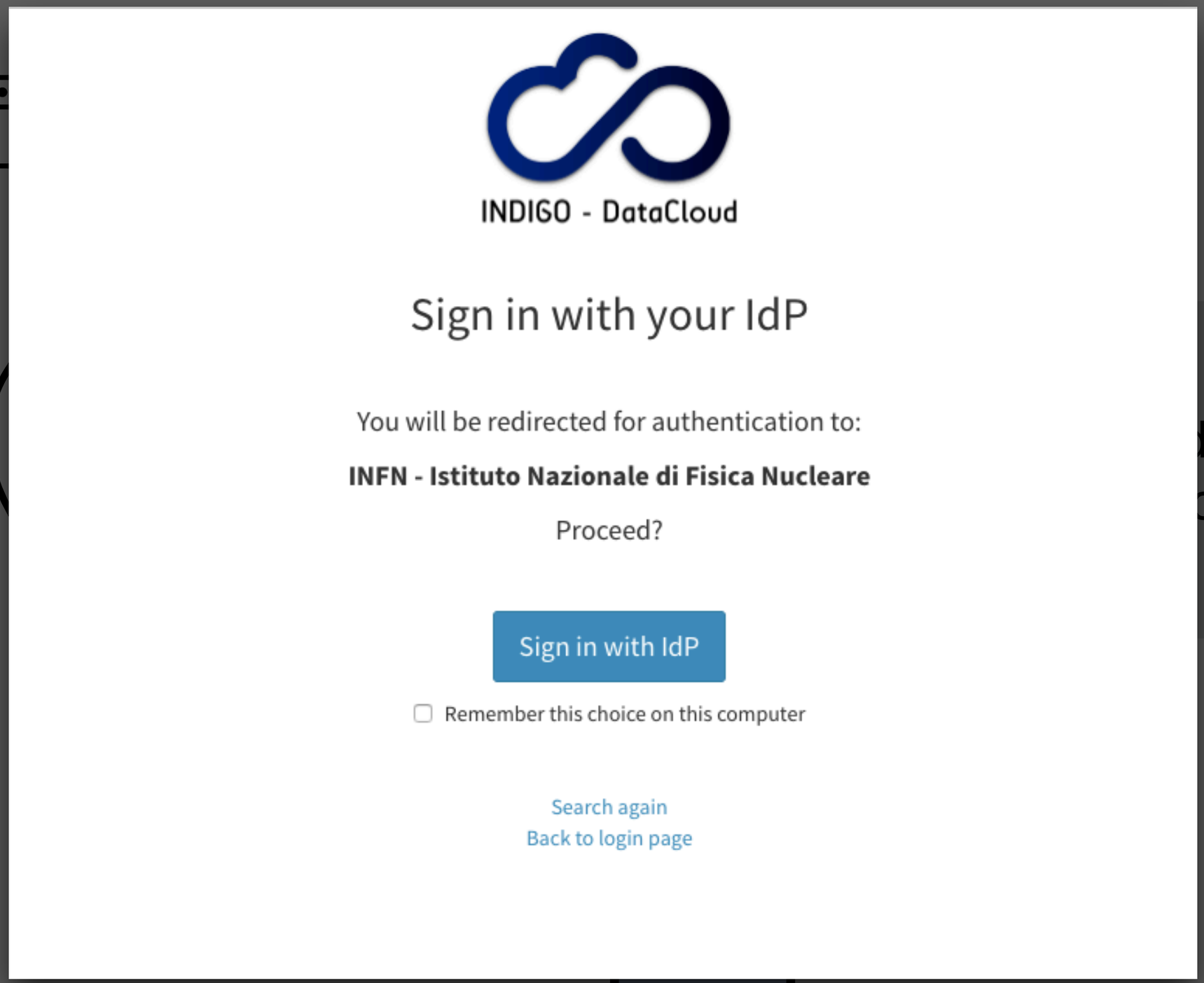
session at

gin page

Home IdP



Authorization code flow

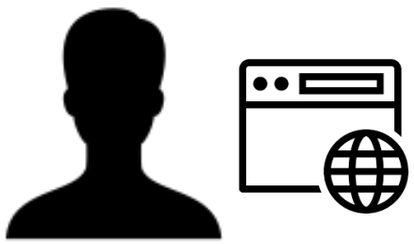


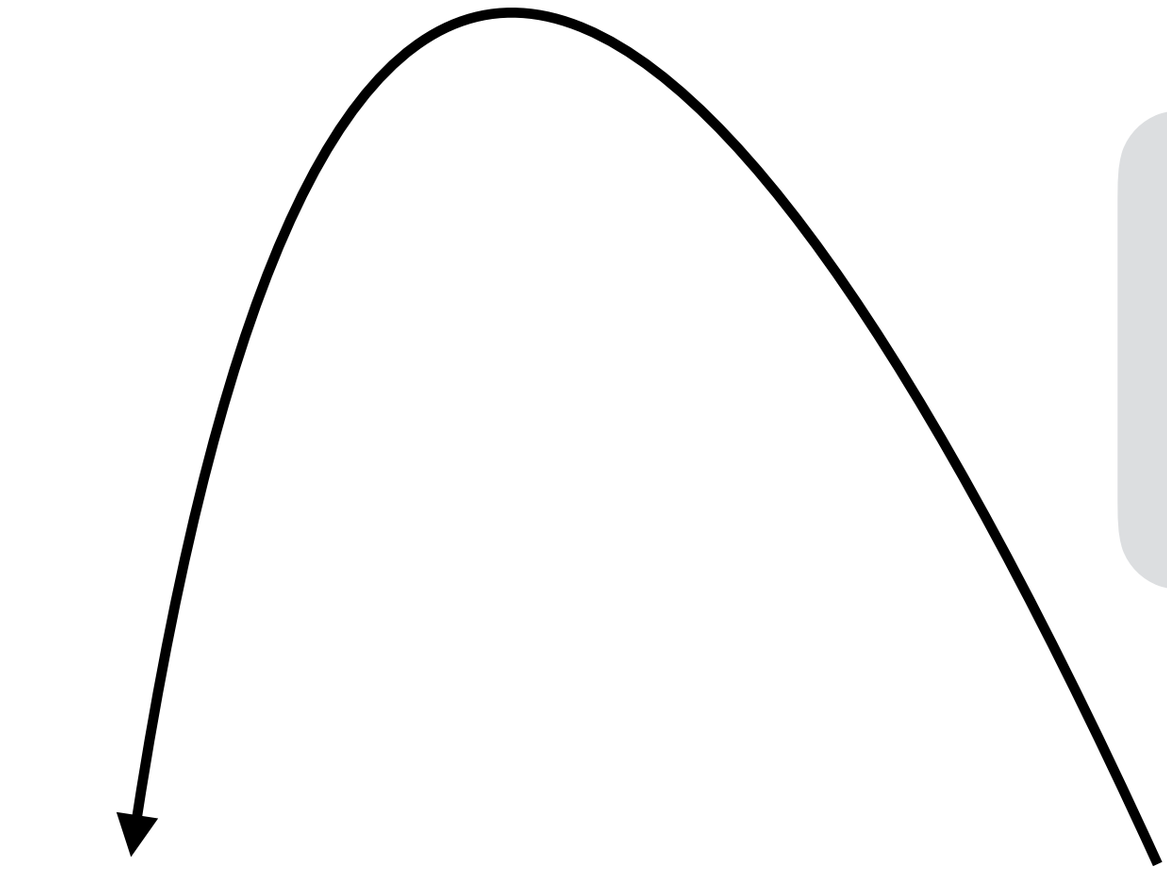
authorization request

session at login page

Home IdP

Authorization code flow

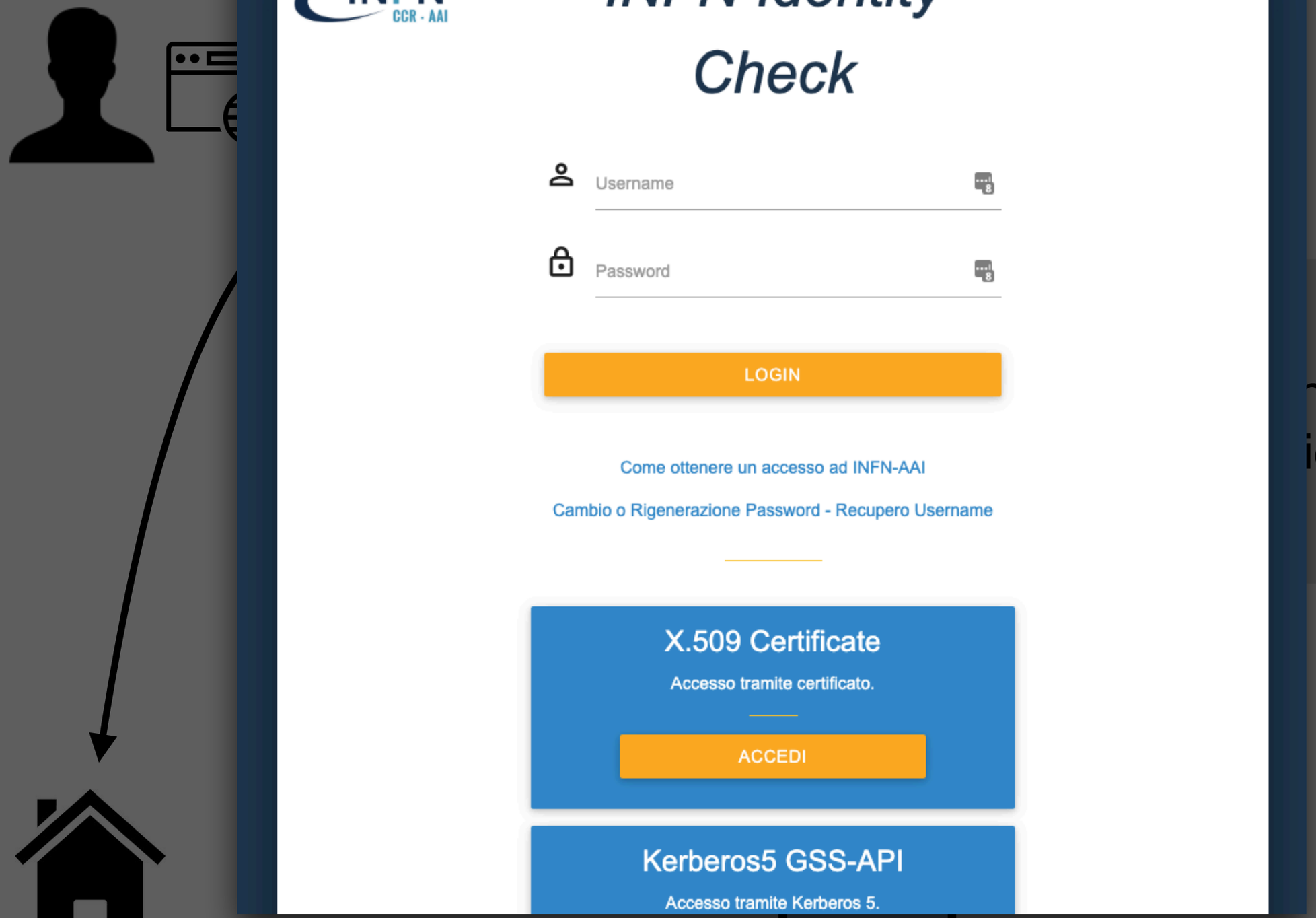




Home IdP

User is redirected to home IDP for authentication

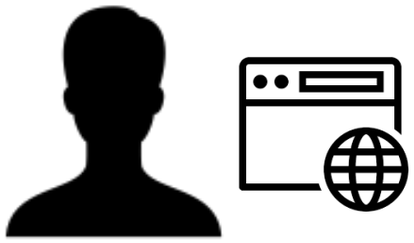
Authorization code flow



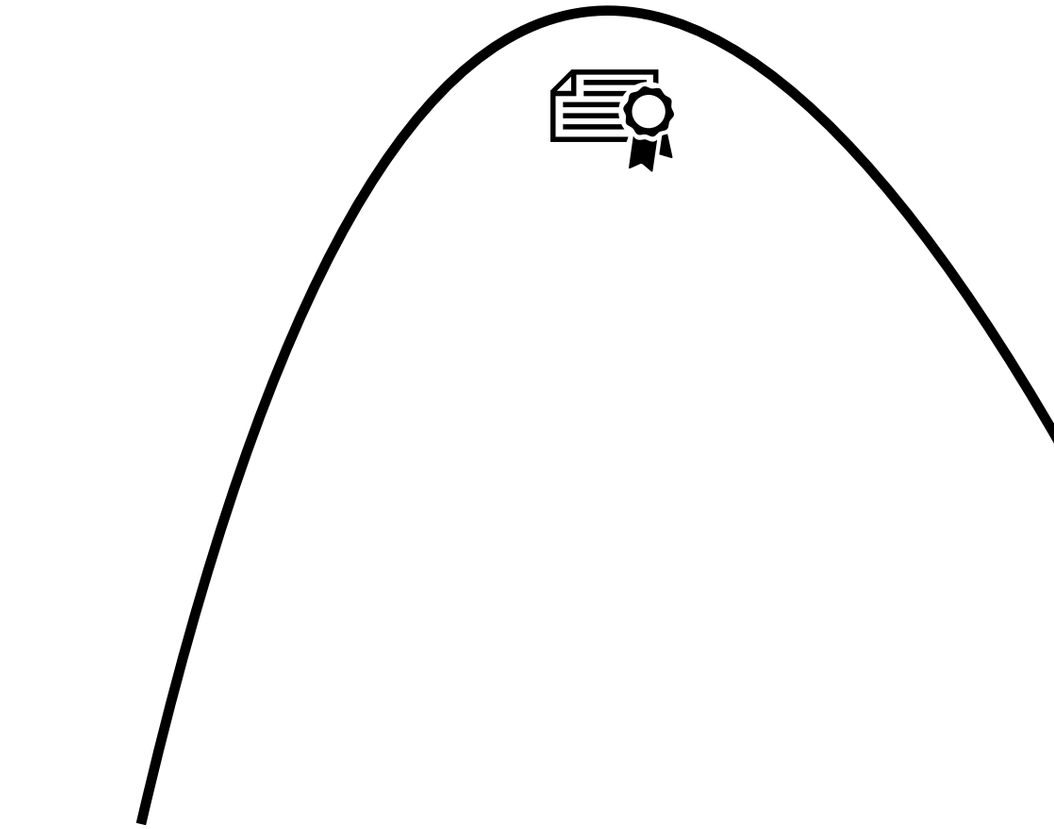
Home IDP

ome IDP
ion

Authorization code flow



Web App

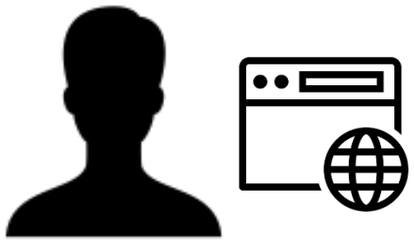


Home IdP

IAM

Home IDP authenticates user and sends back an authentication assertion, via redirection and possibly other interactions between IAM and the IDP

Authorization code flow



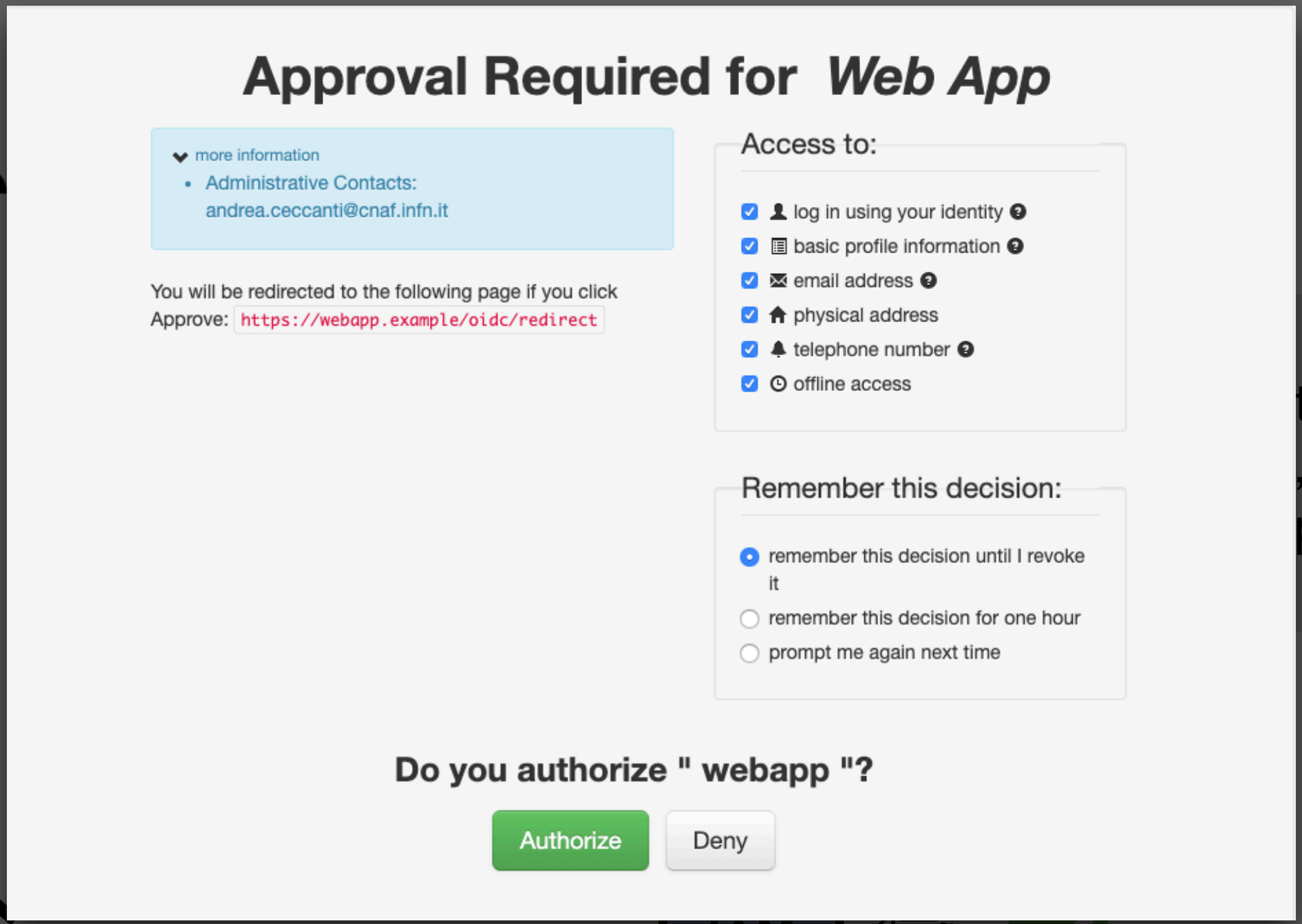
Web App

IAM validates the assertion, the user is a registered one, so IAM shows a “Give consent” page

Home IdP

IAM

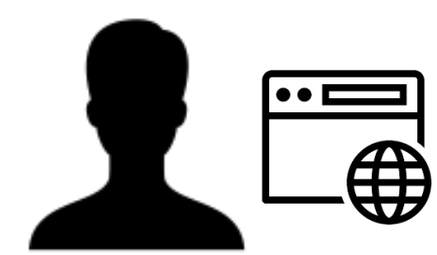
Authorization code flow



tion,
, so IAM
page

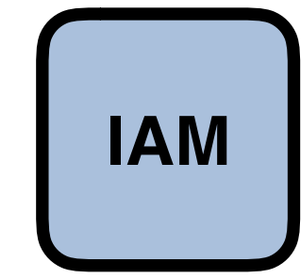
Home IdP

Authorization code flow

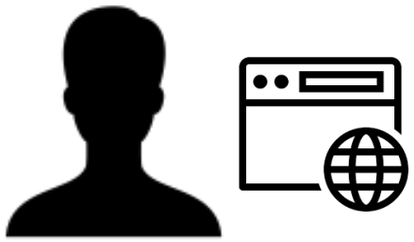


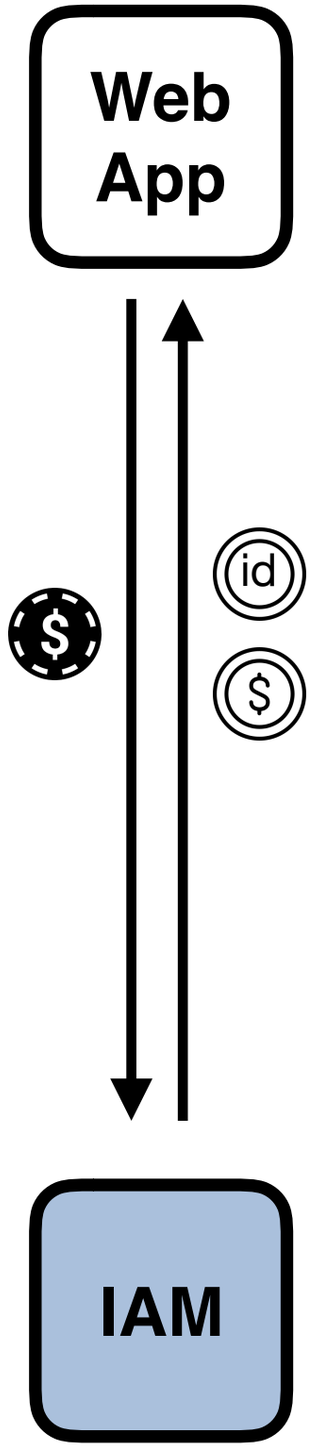
IAM generates an
authorization code
and sends it back to web app using
an HTTP redirect

Home IdP



Authorization code flow

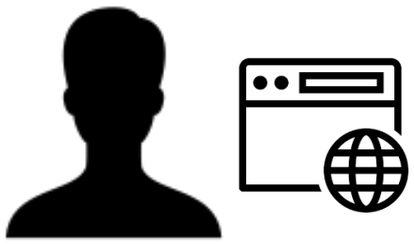


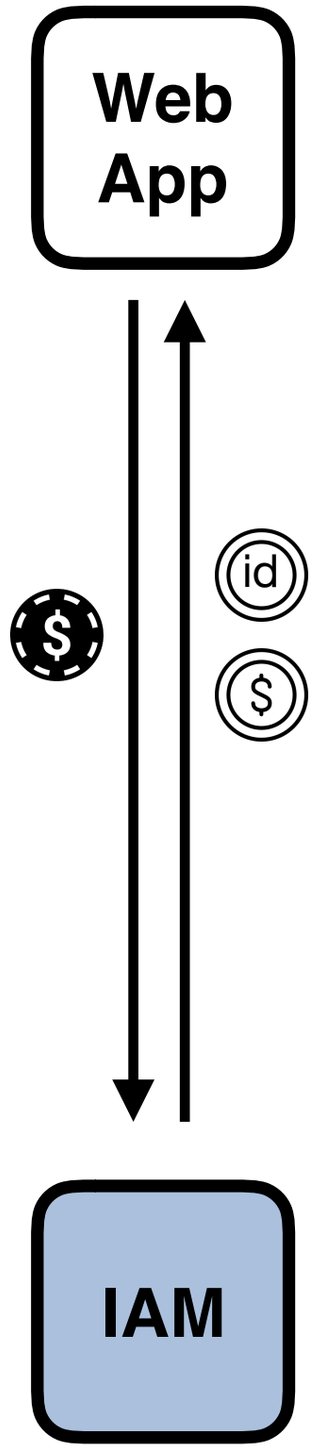


The Web App exchanges the **authorization code** with a couple of tokens:
an **access token** and an **id token**

Home IdP

Authorization code flow

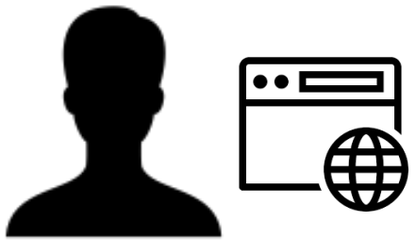


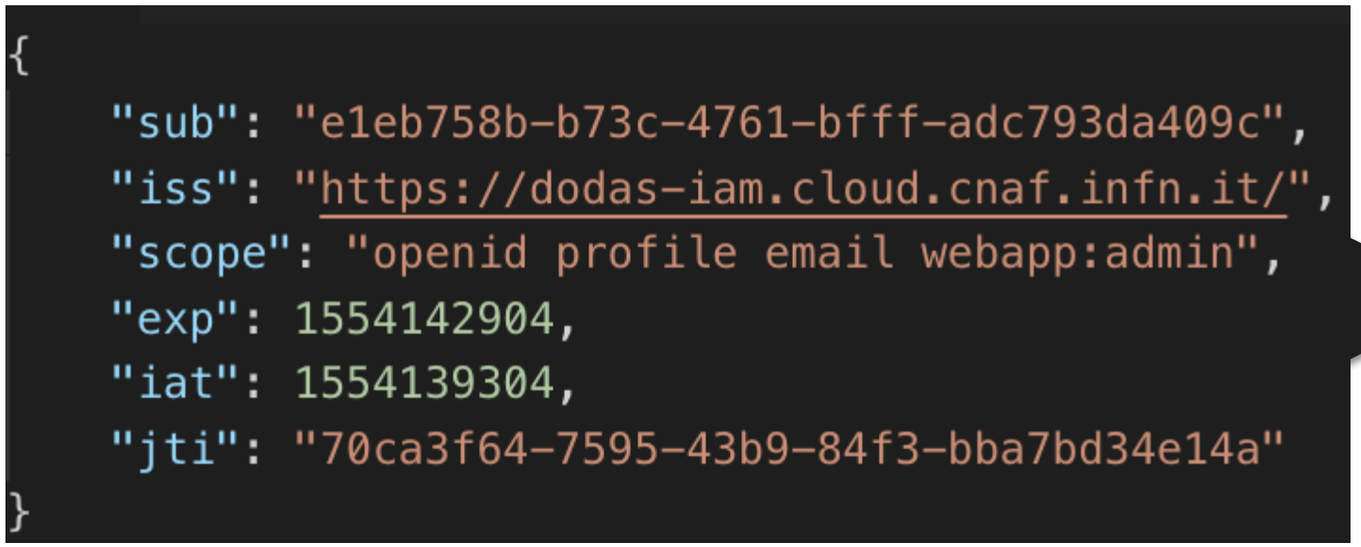


In the IAM implementation, both tokens are **JWT tokens**.

Home IdP

Authorization code flow





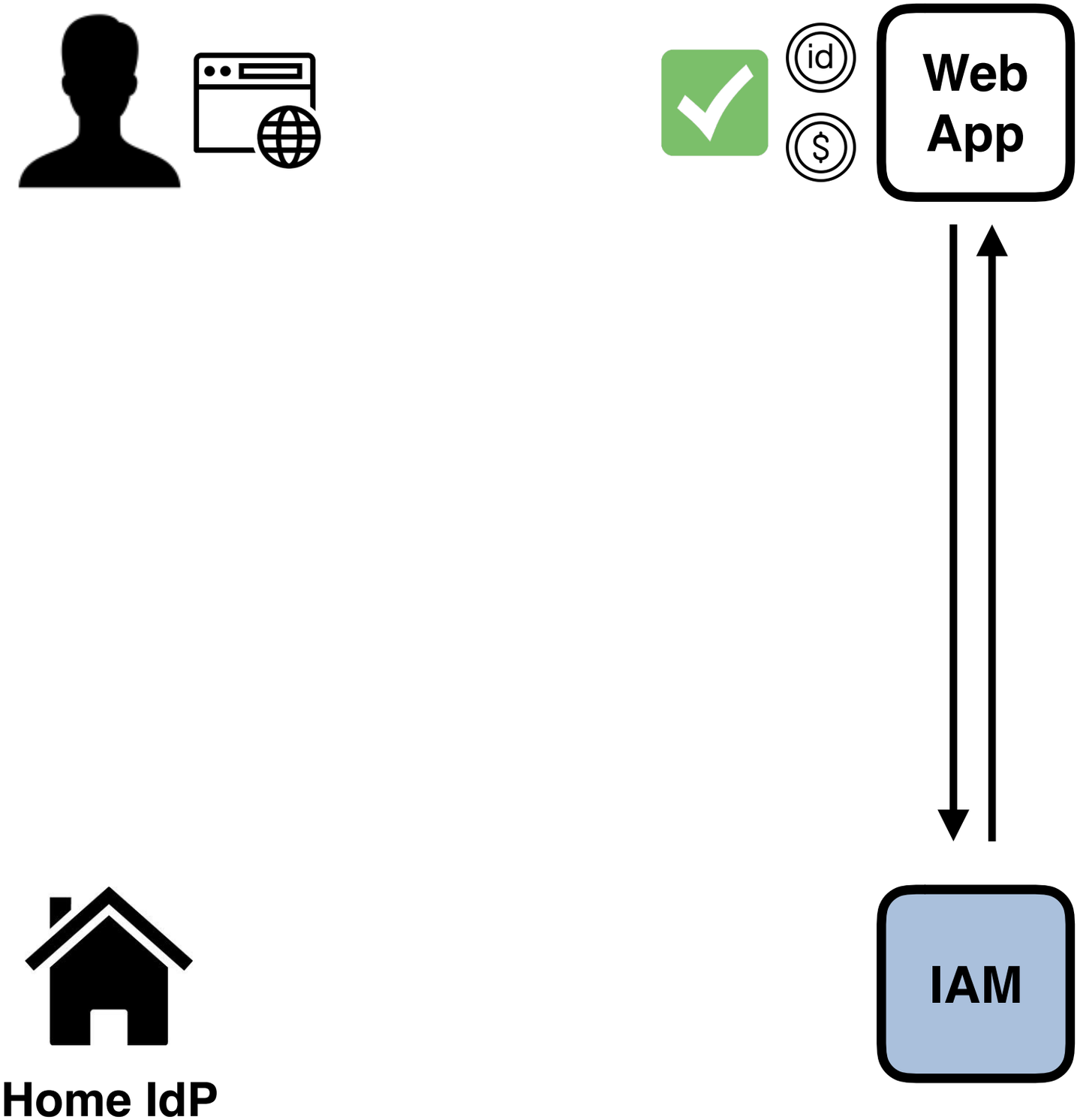
Home IdP

The **access token** provides (mainly) authorization information

Authorization code flow

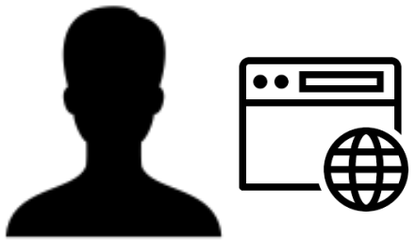
The **id token** provides (mainly) authentication information

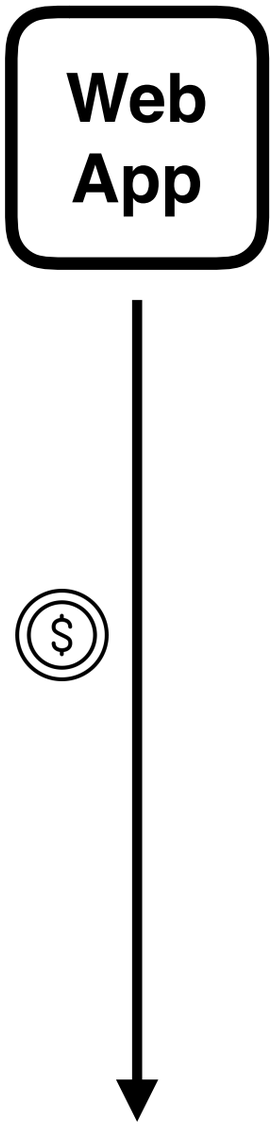
Authorization code flow



Both tokens are **validated** following to the OpenID Connect guidelines, checking **temporal validity**, **token signature**, **audience**, etc…

Authorization code flow

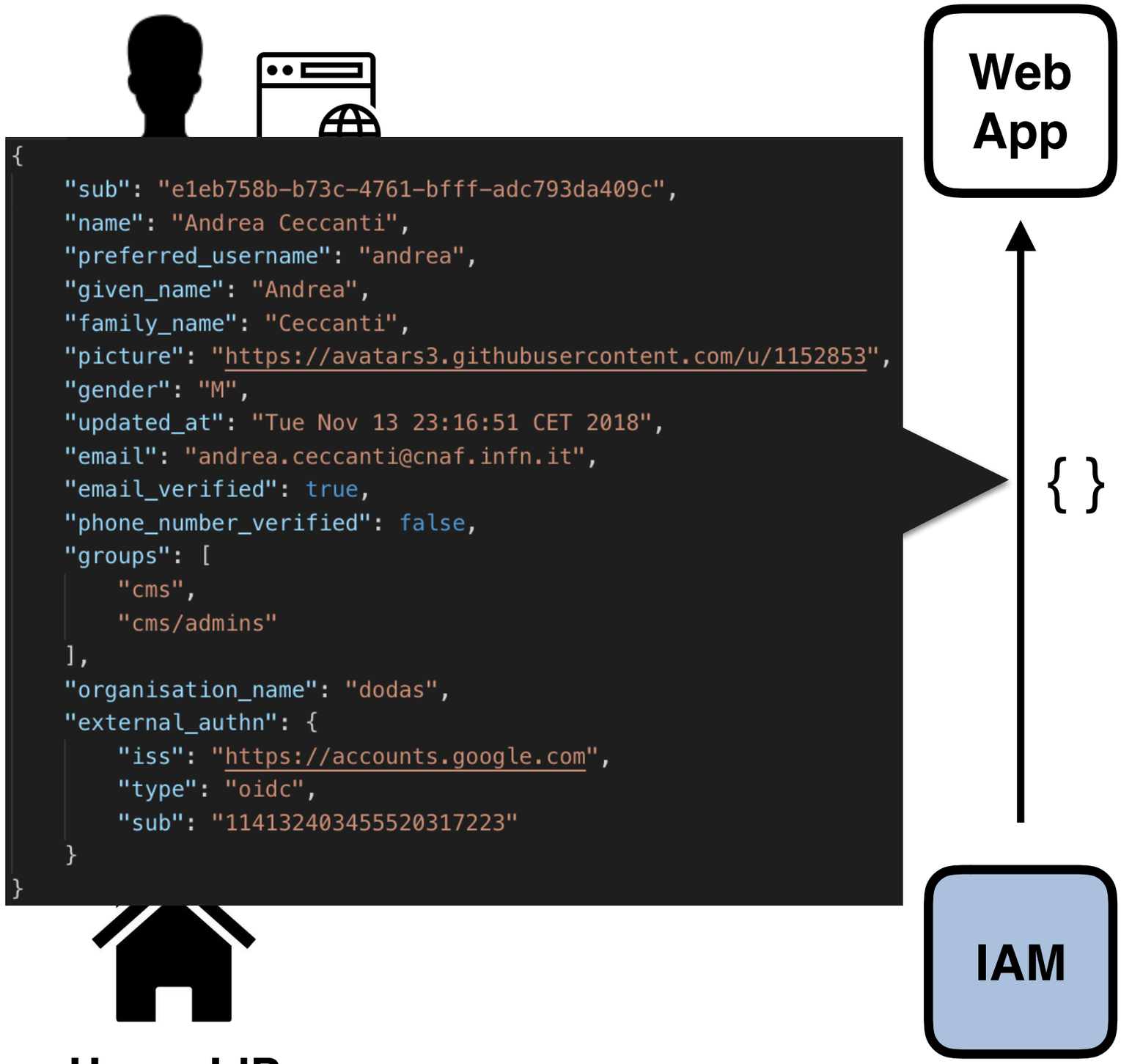




Additional information about the user can be requested by querying the **/userinfo** endpoint and providing the just obtained **access token** for authentication/ authorization purposes

Home IdP

Authorization code flow



The returned JSON object contains authentication information that can overlap with the contents of the **id token**, depending on the IAM configuration

Authorization code flow in practice

In practice, decent OAuth/OpenID Connect client libraries implement all the above **behind the scenes.**

As an example, Apache mod_auth_openidc requires the following information to enable a working OpenID Connect integration

- The OpenID Connect provider discovery/metadata URL
- Client credentials

The library then takes care of exchanging messages with the OpenID provider, implementing verification checks, and provides the obtained authentication/authorization information to the protected web application

- typically via env variables or HTTP headers