WLCG Authorization WG progress report

Andrea Ceccanti (INFN-CNAF)
on behalf of the WLCG AuthZ WG

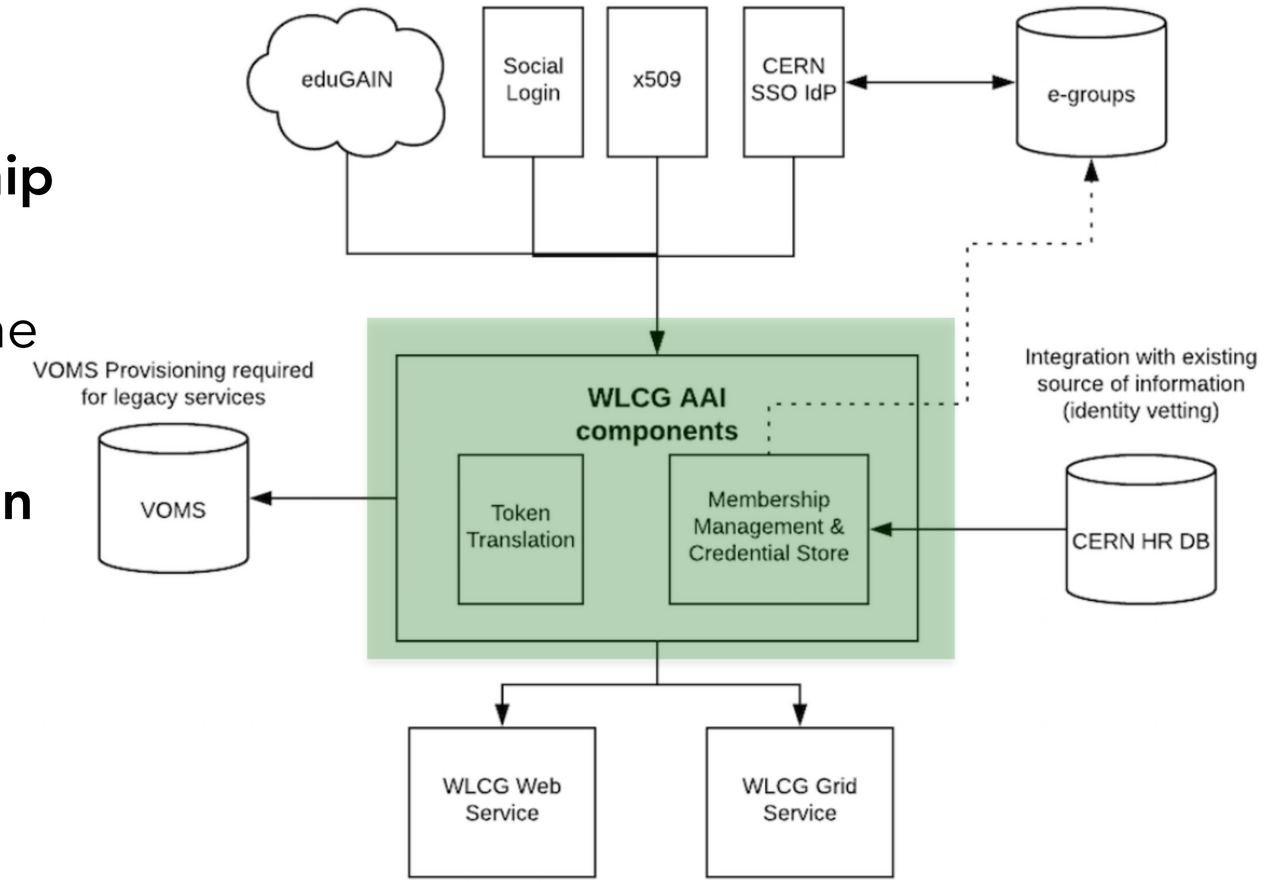
GDB meeting
December, 11th 2019

The WLCG Authorization WG

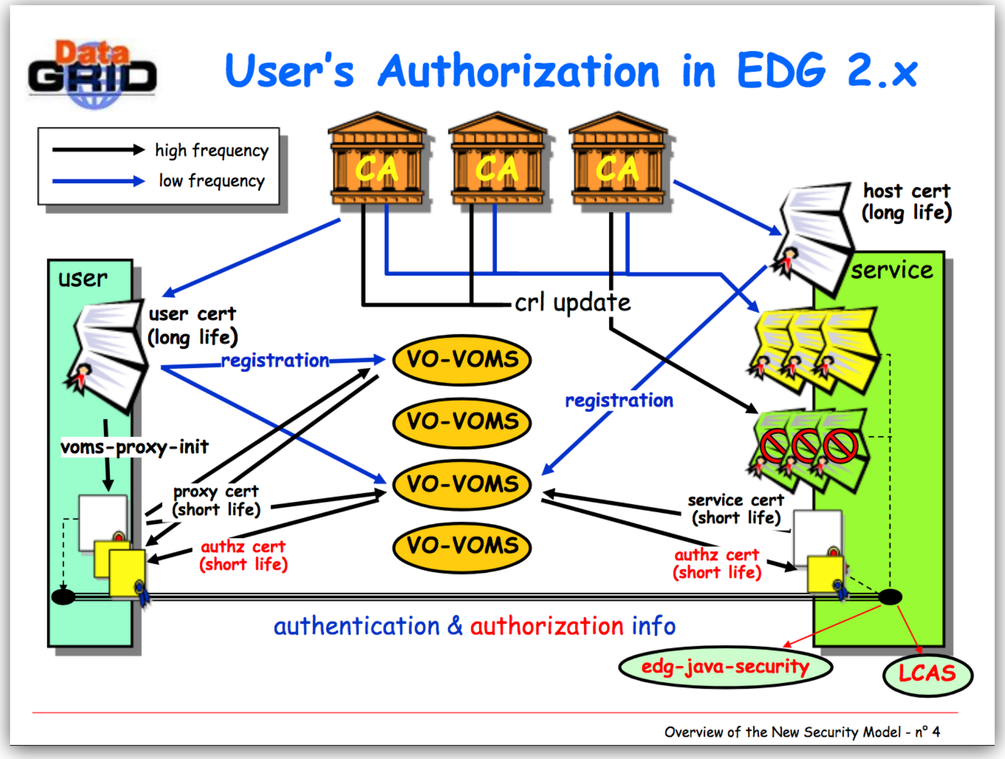
https://twiki.cern.ch/twiki/bin/view/LCG/WLCGAuthorizationWG

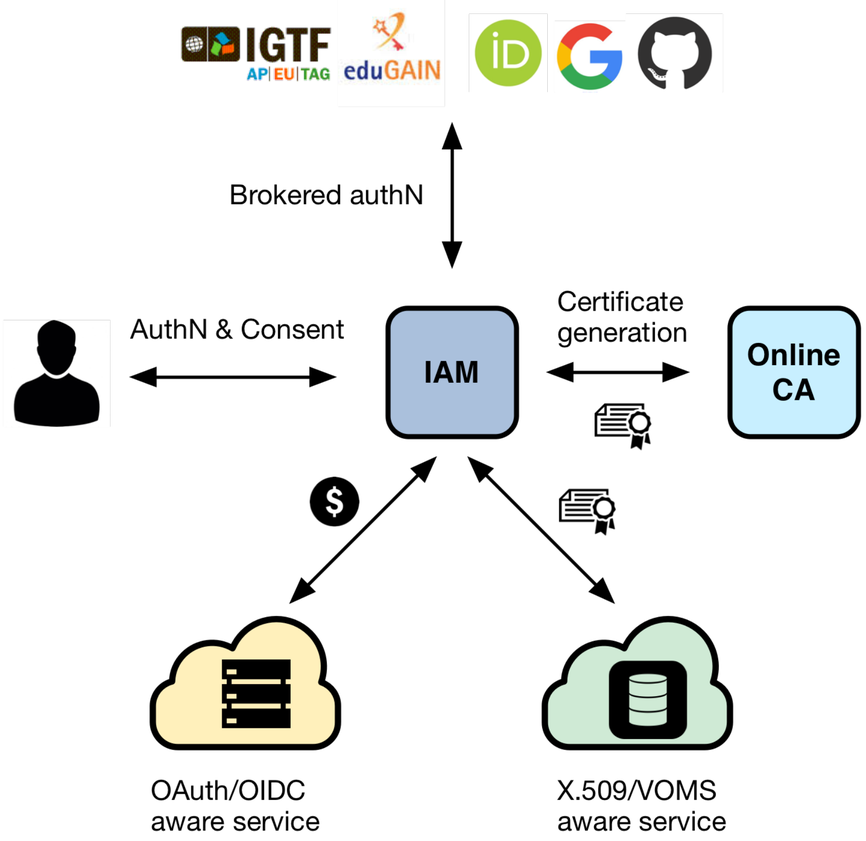
Main objectives:

- 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**



Objective: evolution of the WLCG AAI beyond X.509





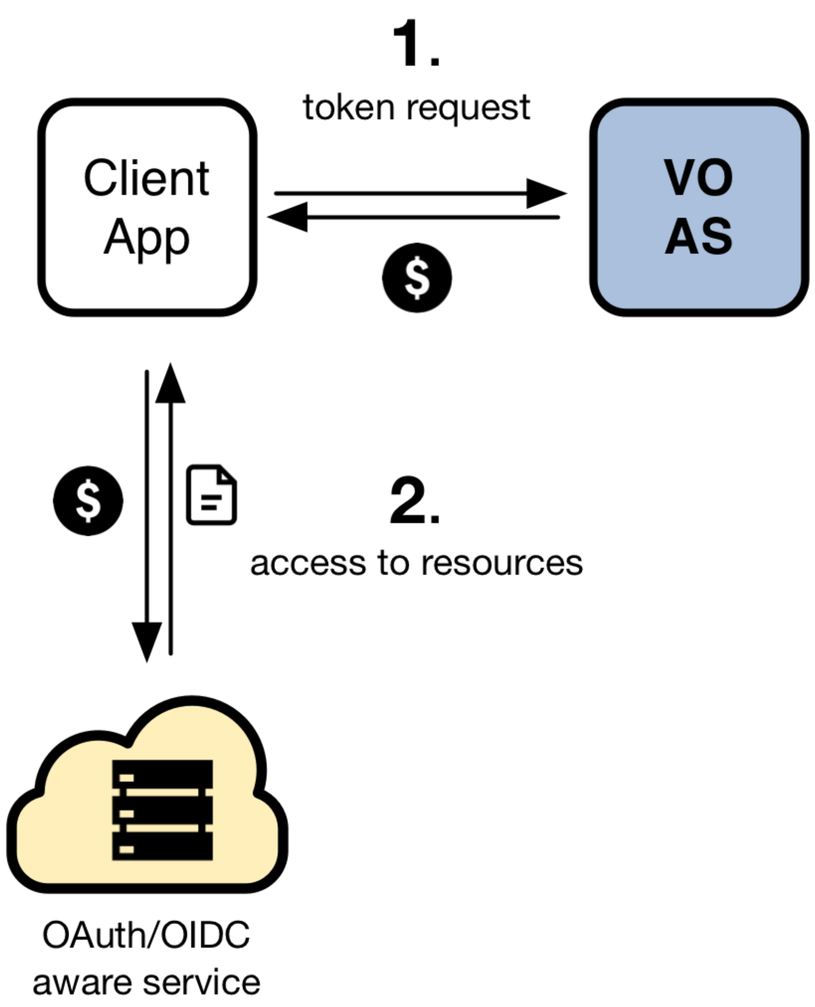
Token-based AuthN/Z for WLCG

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**
- **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

token claims

```
{
 "iss": "https://cms.wlcg.example",
 …
 "wlcg.groups": "/cms"
}
```

local policy

authZ decision

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

token claims

```
{
 "iss": "https://cms.wlcg.example",
 …
 "scope": "storage.read:/ storage.modify:/store"
}
```

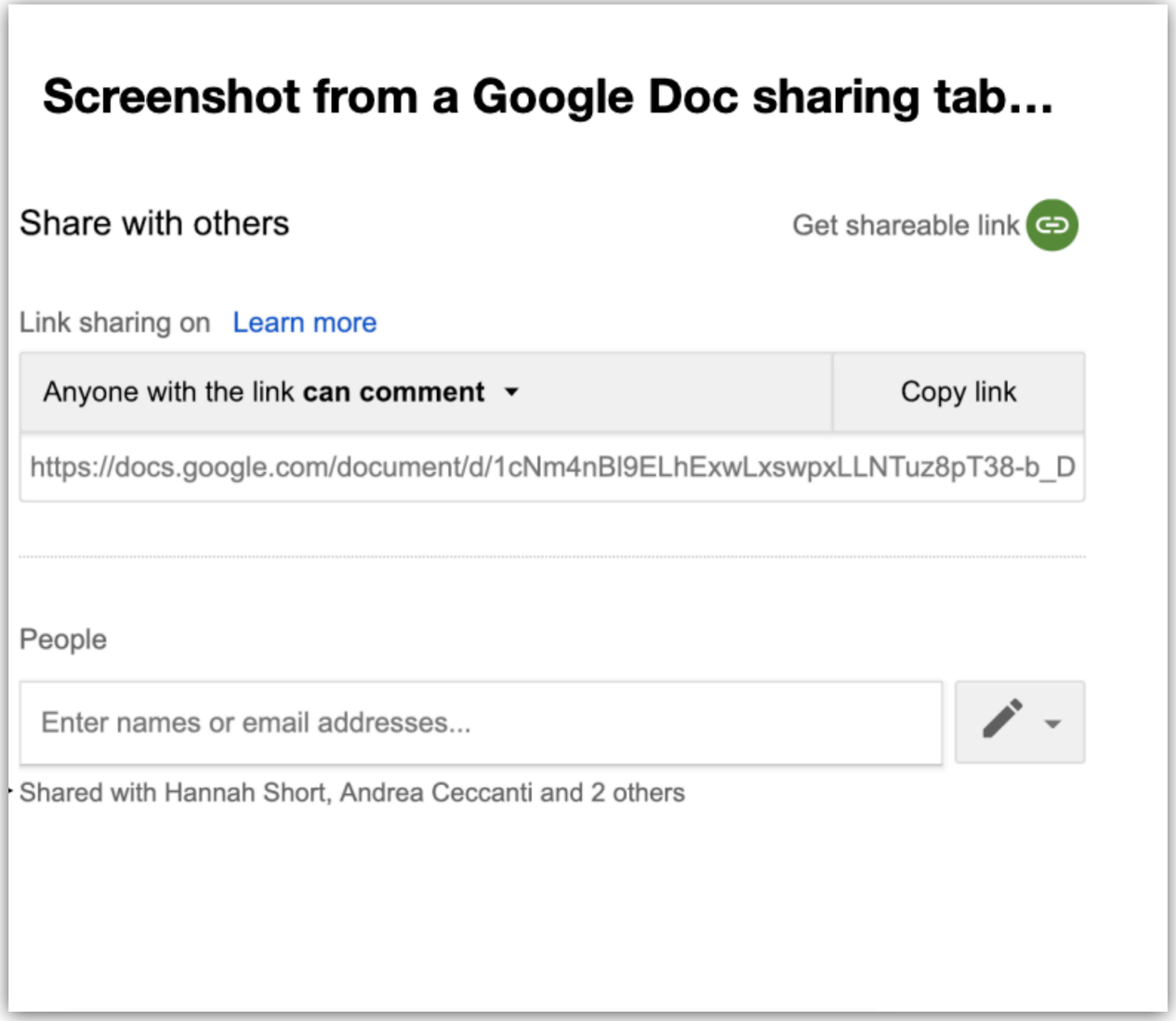
authZ decision

Identity-based vs Scope-based Authorization

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

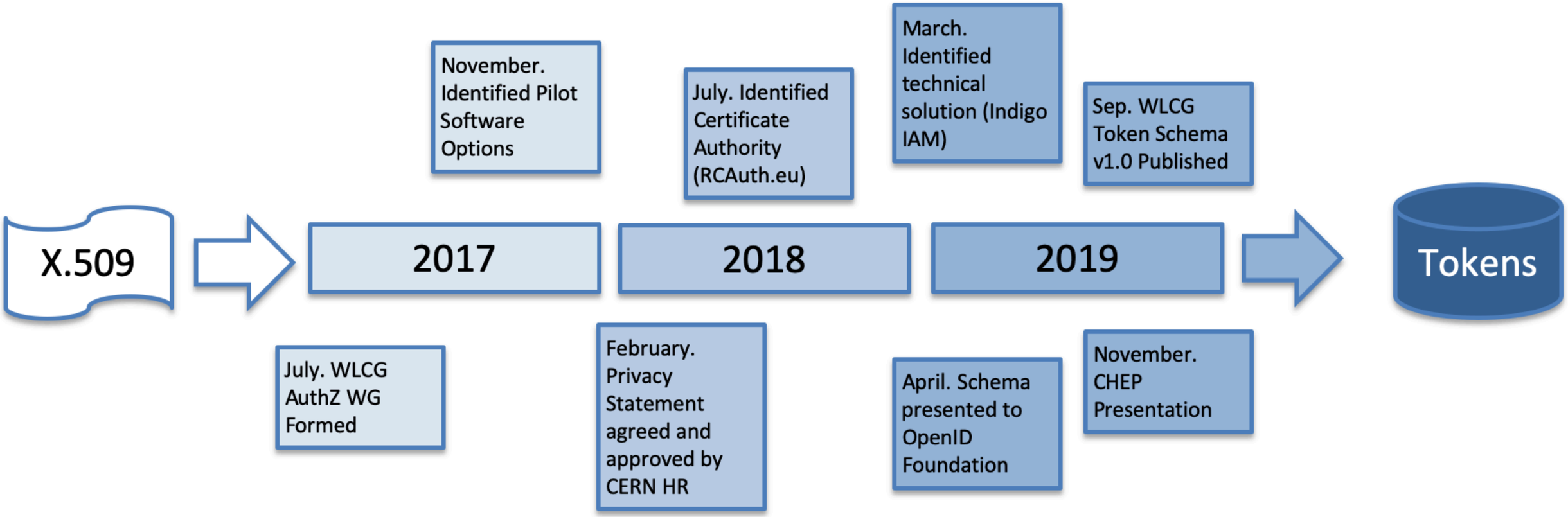
scope-based authZ

identity-based authZ



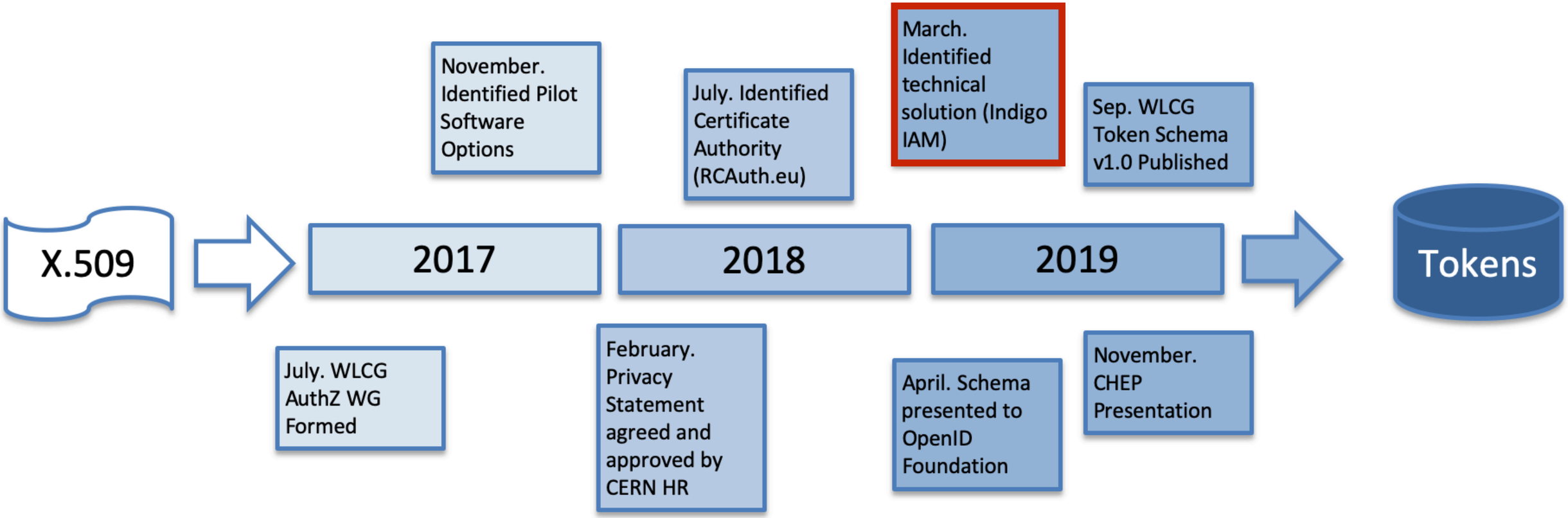
* Slide courtesy of B. Bockelman

Towards Tokens



* Slide courtesy of H. Short

Towards Tokens

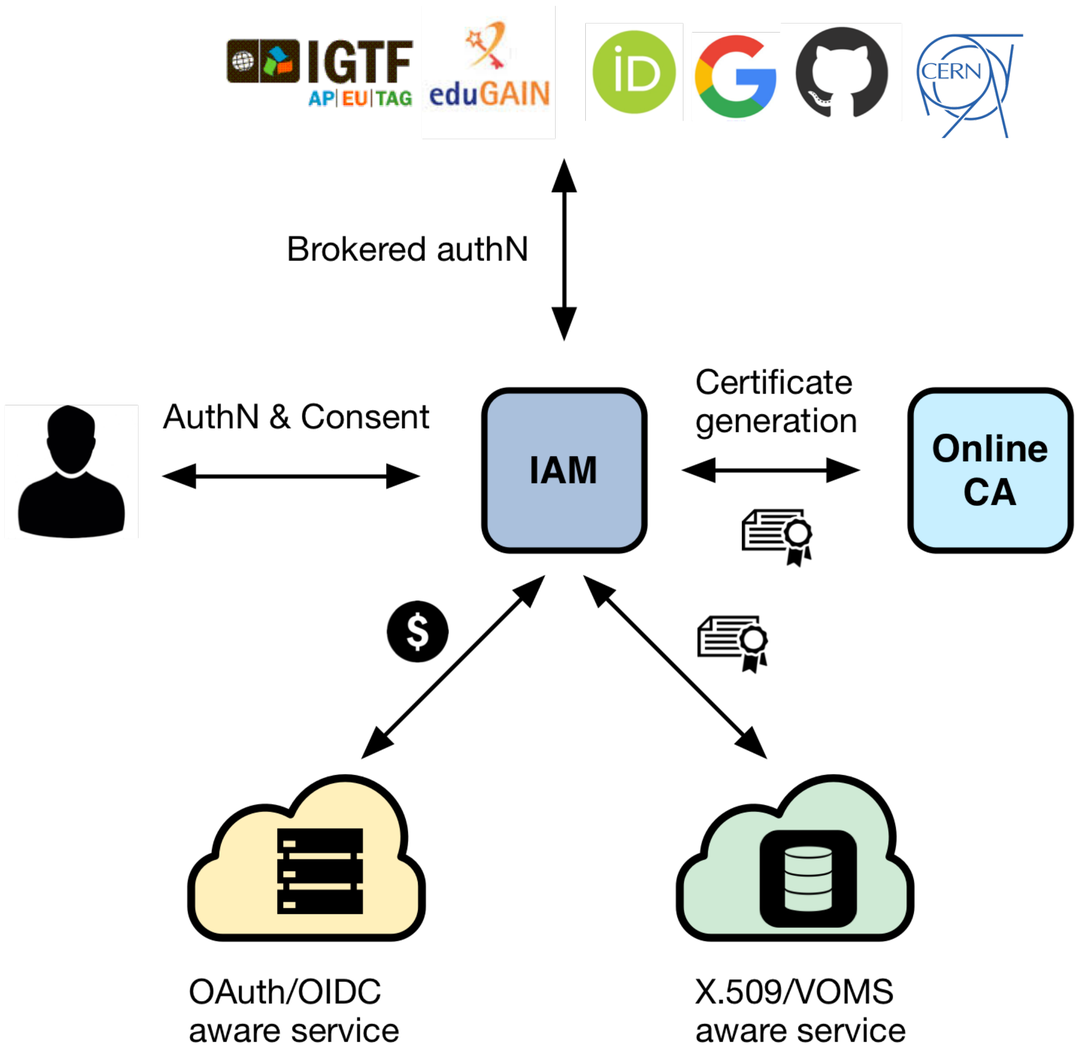


* Slide courtesy of H. Short

INDIGO Identity and Access Management Service

A **VO-scoped** authentication and authorization service that

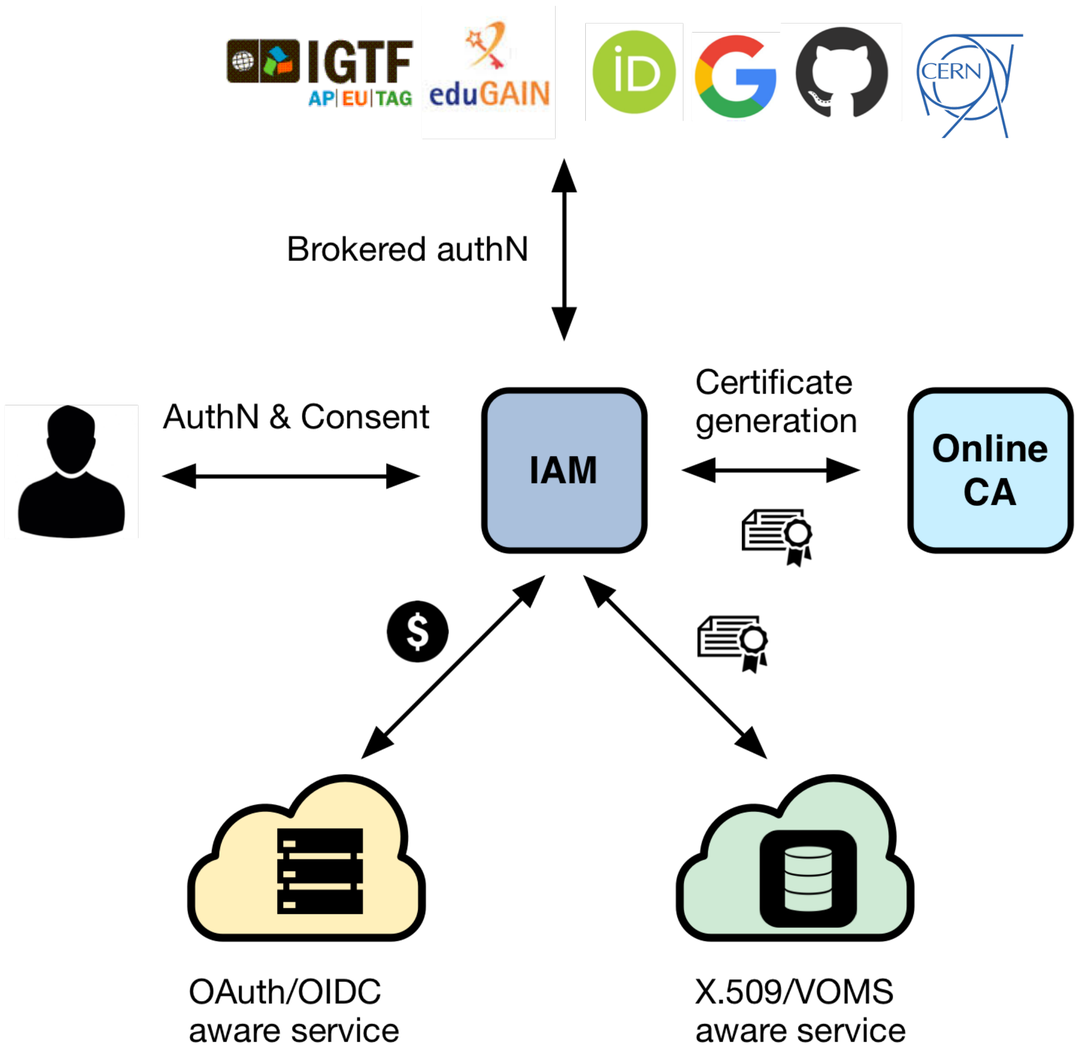
- supports **multiple authentication mechanisms**
- provides users with a **persistent, VO-scoped** identifier
- exposes **identity information**, **attributes** and **capabilities** to services via **JWT** tokens and standard **OAuth & OpenID Connect** protocols
- can integrate existing **VOMS**-aware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



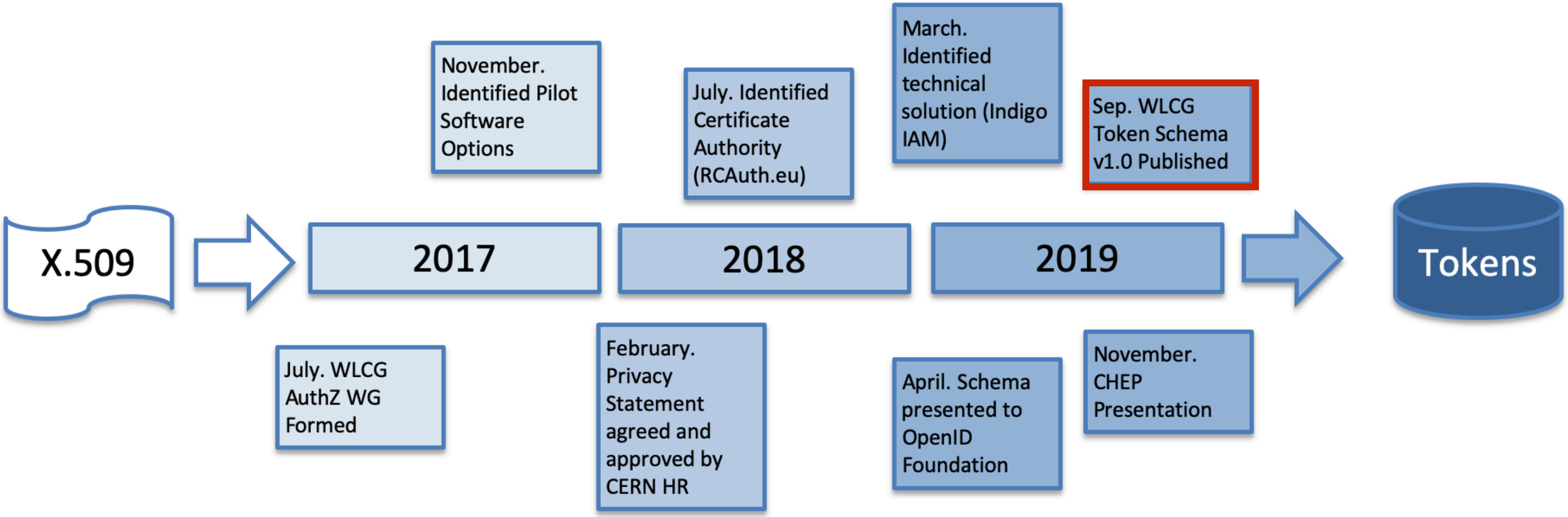
INDIGO Identity and Access Management Service

Selected by the WLCG MB to be the core of the future, token-based WLCG AAI

Sustained by INFN for the foreseeable future, with current support from:



Towards Tokens



* Slide courtesy of H. Short

The JWT common profile has reached v1.0

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

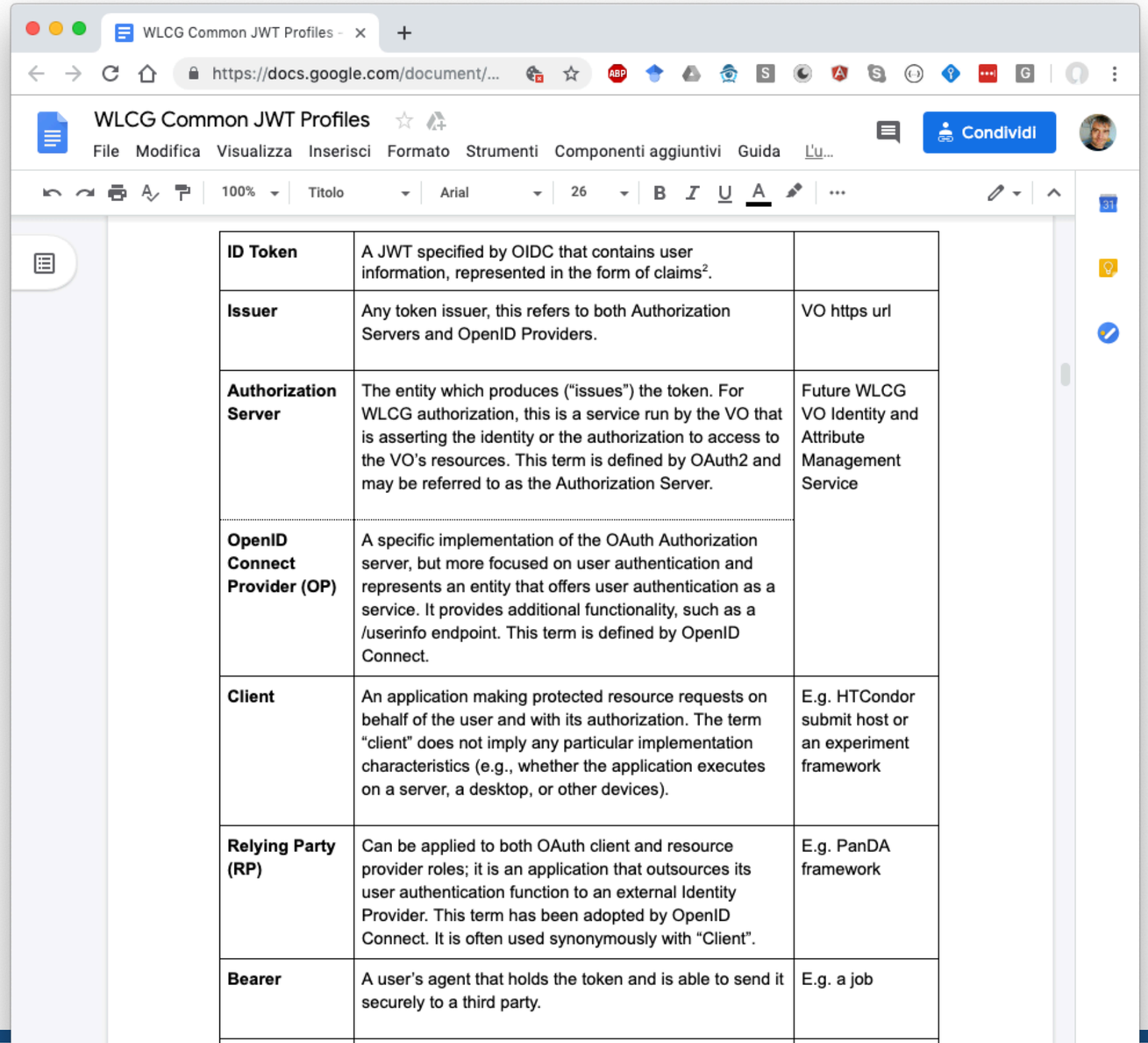
The WLCG JWT profile in more detail

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

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.modify:/protected

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

Path semantics

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

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, XRootD, etc…., the token just provides a (signed) string!

Scope-based group selection

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 group selection

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 group selection

…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 group selection

…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 group 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

Implementing the JWT profile

What does it mean supporting the WLCG profile?

Depends on the **role** of your service:

- OAuth resource server
  - The typical example is an **HTTP Restful API**
  - Does not need the ability to start an OAuth/OpenID Connect authentication flow
  - Does not need to be registered in IAM
  - Needs to extract token from incoming requests and validate token and map authn/authz info in the token to local authz enforcement

- OAuth/OpenID Connect client:
  - The typical example is a **Web application (a portal)** that wants to delegate authentication to IAM
  - Needs to be registered in IAM
  - Needs the ability to start OAuth/OpenID Connect auhn/z flow, store securely client credentials, validate tokens, refresh them when needed …

- Some services will naturally fit in **both roles** defined above
  - e.g., RUCIO, FTS, dCache

What does it mean supporting the WLCG profile?

As an **OAuth resource server** (RS):

- Ability to extract an access token from an incoming HTTP request
- Ability to parse and validate the incoming access token
  - identify if it has been issue by a trusted and recognized authorization server
  - verify temporal validity
  - verify signature, following OAuth/OIDC conventions
- Ability to honour access token audience restrictions
  - the RS needs the ability to identity itself with (one or multiple) audience labels and honour audience restrictions in access tokens
- Ability to map defined scopes to local authZ
  - e.g., storage.read:/cms grants read access to the /cms namespace (and any subdirectory)
- Ability to map group-based to local authZ
  - e.g., /cms group membership as stated grants read access to the /cms namespace

What does it mean supporting the WLCG profile?

As an **OAuth resource server** (RS):

- Ability to extract an access token from an incoming HTTP request
- Ability to parse and validate the incoming access token
  - identify if it has been issue by a trusted and recognized authorization server
  - verify temporal validity
  - verify signature, following OAuth/OIDC conventions
- Ability to honour access token audience restrictions
  - the RS needs the ability to identity itself with (one or multiple) audience labels and honour audience restrictions in access tokens
- Ability to map defined scopes to local authZ
  - e.g., storage.read:/cms grants read access to the /cms namespace (and any subdirectory)
- Ability to map group-based to local authZ
  - e.g., /cms group membership as stated grants read access to the /cms namespace

This is typically sorted out by OAuth/OIDC libraries

What does it mean supporting the WLCG profile?

This is typically sorted out by OAuth/OIDC libraries

As an **OAuth/OpenID Connect client**:

- Ability to store client credentials securely
- Ability to start and manage an OAuth/OpenID Connect flow to obtain tokens from the Authorization Server (i.e., IAM)
  - Authorization code flow, for most use cases
  - Refresh token flow, to refresh access tokens about the expire
  - Client credentials flow, to obtain tokens linked not linked to user identities, but to the service itself
- Ability to parse and validate ID tokens resulting from OpenID Connect authentication flows in compliance with the OpenID connect spec
- Ability to honour audience restrictions
  - the ability to identity itself with (one or multiple) audience labels and honour audience restrictions in ID tokens
- (Optional) Ability to implement Level Of Assurance (LoA) policies

The WLCG IAM instance

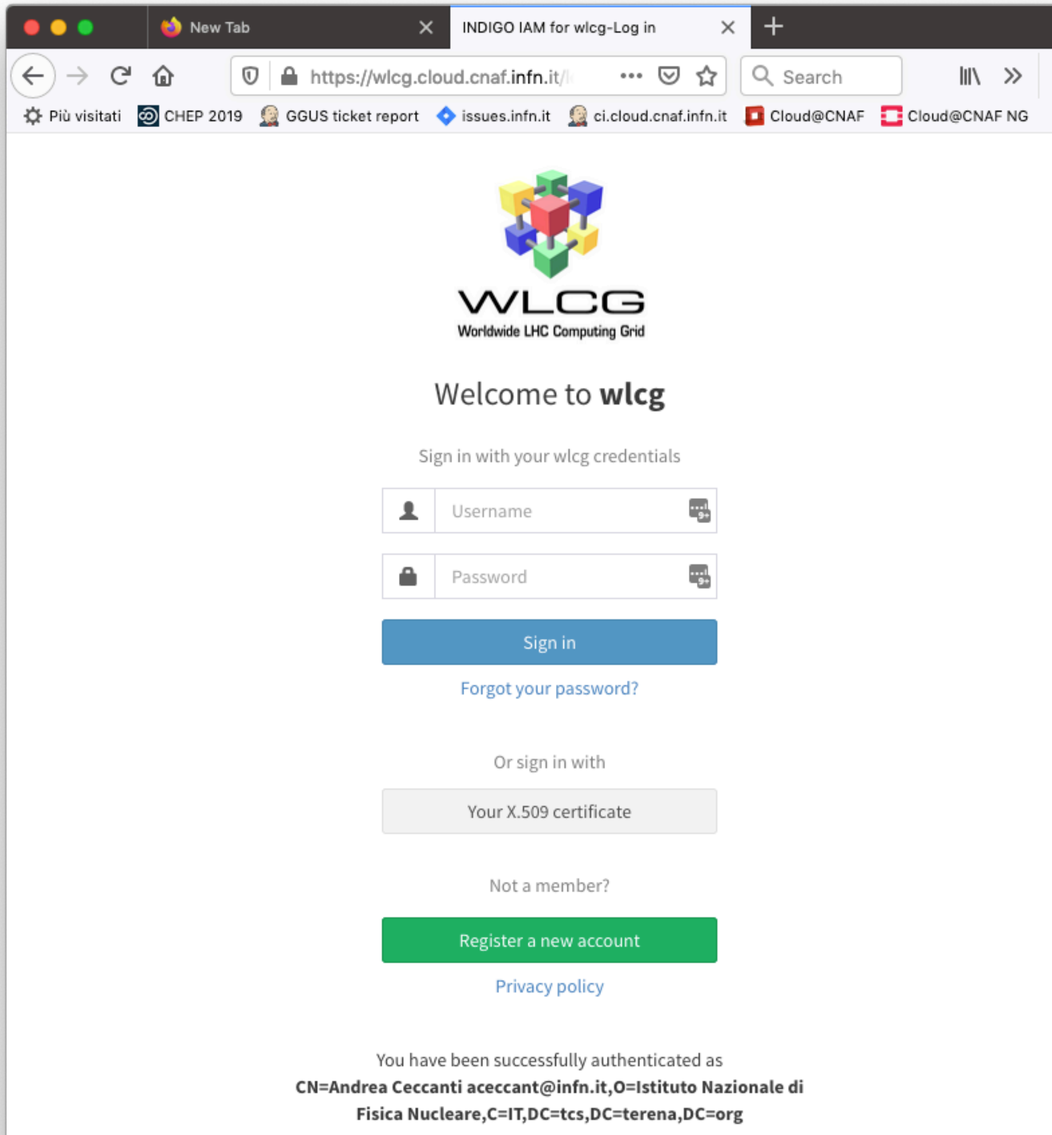
A WLCG-managed, **experiment-agnostic VO in support of WLCG development, integration and testing activities** focusing on **the transition to token-based AuthN/Z**

https://wlcg.cloud.cnaf.infn.it

Deployed at INFN-CNAF, **integrated with CERN SSO**

- Provides support for VOMS and token/based AuthN/AuthZ
- **Supports the WLCG JWT profile v1.0**

Reference for WLCG token-based authn/z integration activities



Ongoing integration activities

DOMA Third-Party Copy WG

- Token-based authN/Z testbed leveraging the WLCG IAM instance in support of bulk transfers
  - RUCIO, FTS, XRootD, dCache, DPM, Echo, StoRM, EOS
- Now focus is on integrating support for the WLCG JWT profile, and on the flows that must be used to obtain/exchange tokens with IAM (see this draft document and this slide deck)
- **Token-based AuthN/Z "hackathon" @ CERN in January**
  - more on this in next slide

HTCondor

- Successful submission of a job using a token issued by IAM demonstrated last month at CHEP

Token-based AuthN/Z “Hackathon” @ CERN in January

What: sort out as many problems as possible while discussing things and coding together in a room with the objective of demonstrating **a full stack HTTP X509-free data transfer management chain**

- RUCIO->FTS->SEs
- SEs: EOS, dCache, DPM, StoRM, XRootD, Echo

Who: Developers of the above components

When/Where: **January, 16th 2020 @ CERN**

Conclusions

Huge progress in the recent months towards enabling the transition to token-based AuthN/Z

WLCG JWT profile has reached v1.0

- Ongoing work to integrate support for it in many key Grid middleware components

IAM WLCG instance **available** as a reference for integration activities

- implementing the WLCG JWT profile

Hackathon @ CERN in January focused on enabling X509-free data management

Thanks for your attention.
Questions?

References

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

The WLCG JWT profile: https://zenodo.org/record/3460258

OAuth/OIDC for DOMA TPC transfers: https://indico.cern.ch/event/858953/contributions/3617085/attachments/1956518/3250785/DOMA-TPC-041219.pdf

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

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