UMA and Dynamic Client Registration

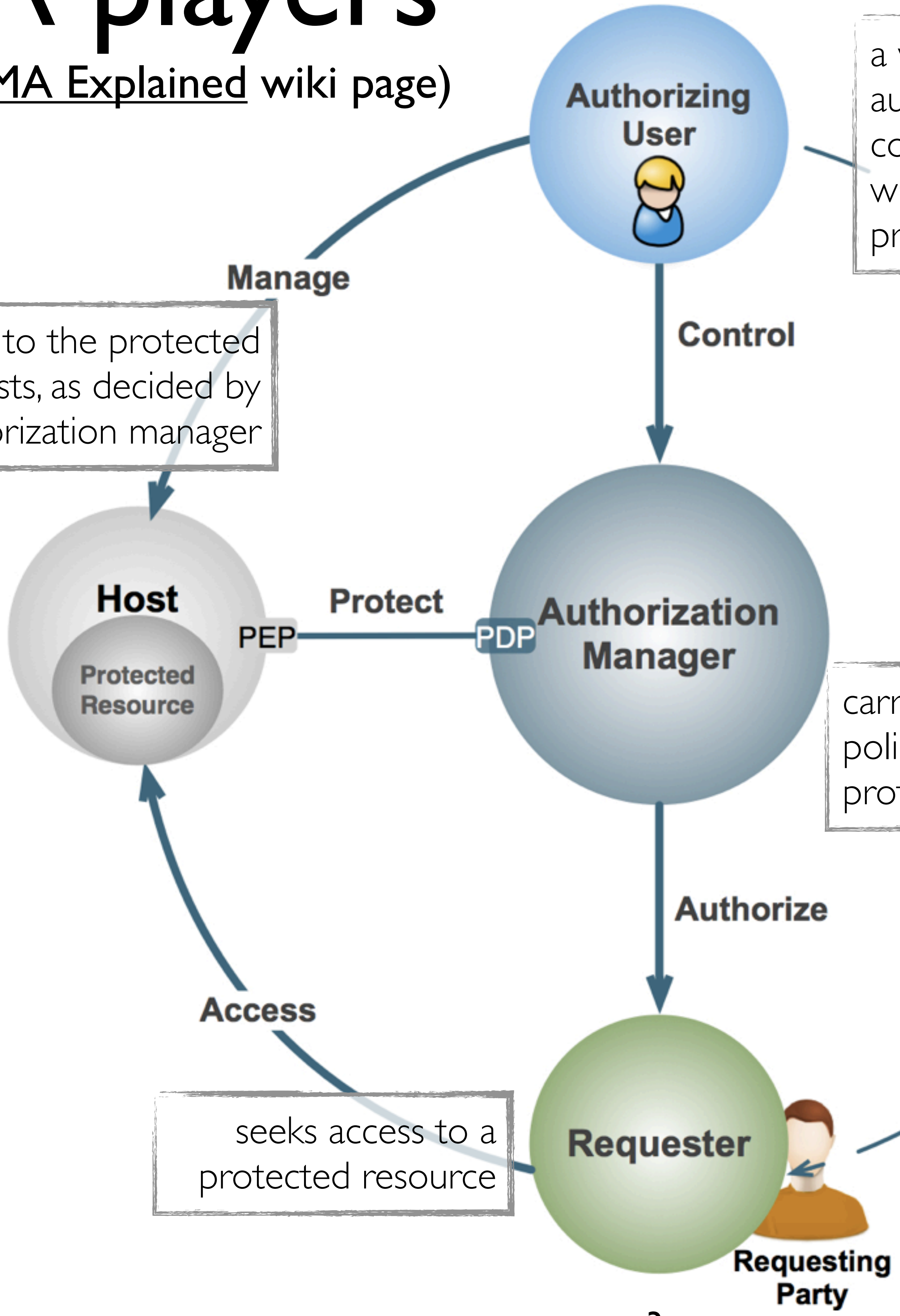
Thomas Hardjono
on behalf of the UMA Work Group

UMA is...

- A web protocol that lets you control authorization of data sharing and service access made on your behalf
- A Work Group of the Kantara Initiative that is free for anyone to **join** and contribute to
- A set of draft specifications that is free for anyone to implement
- Undergoing multiple implementation efforts
- Slated to be contributed to the IETF once "incubated" (roughly by August, in modular pieces over time)
- Striving to be simple, OAuth-based, identifier-agnostic, RESTful, modular, generative, and developed rapidly

UMA players

(see also UMA Explained wiki page)



a web user who configures an authorization manager with policies that control how it makes access decisions when a requester attempts to access a protected resource at a host

enforces access to the protected resources it hosts, as decided by an authorization manager

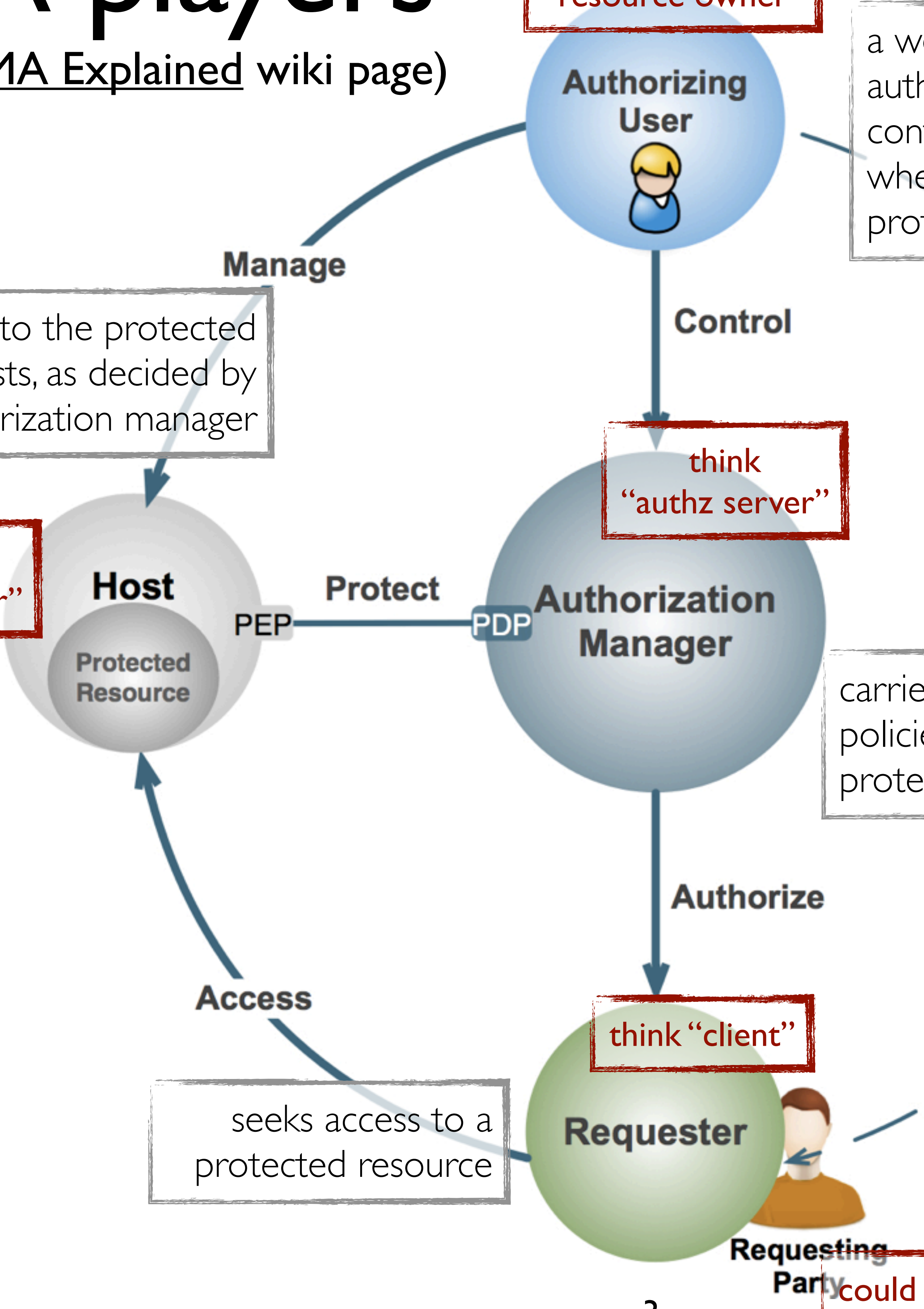
carries out an authorizing user's policies governing access to a protected resource

seeks access to a protected resource

a web user, or a corporation or other legal person, that uses a requester to seek access to a protected resource

UMA players

(see also UMA Explained wiki page)



think "resource owner"

a web user who configures an authorization manager with policies that control how it makes access decisions when a requester attempts to access a protected resource at a host

enforces access to the protected resources it hosts, as decided by an authorization manager

think "resource server"

think "authz server"

carries out an authorizing user's policies governing access to a protected resource

think "client"

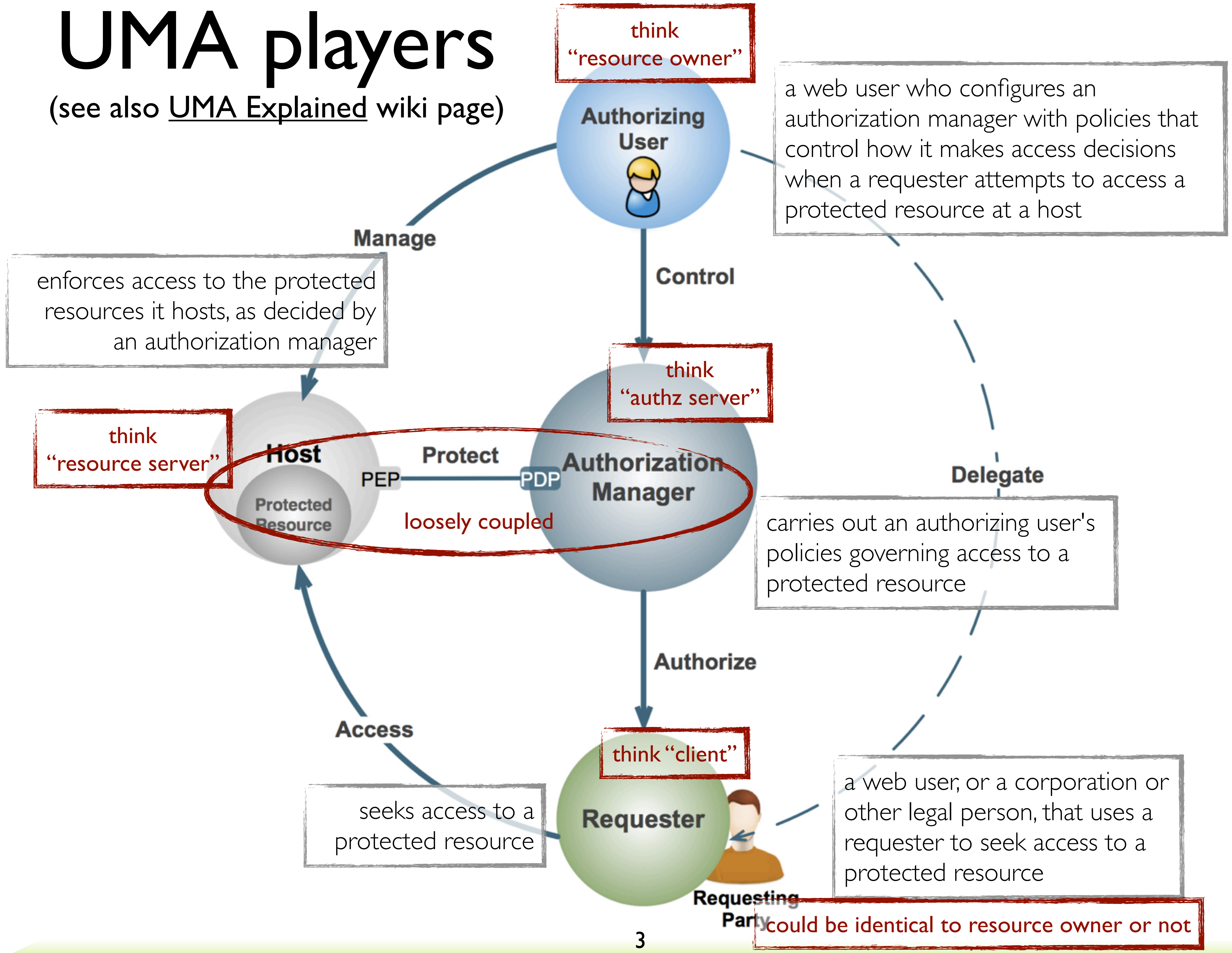
seeks access to a protected resource

a web user, or a corporation or other legal person, that uses a requester to seek access to a protected resource

could be identical to resource owner or not

UMA players

(see also UMA Explained wiki page)



a web user who configures an authorization manager with policies that control how it makes access decisions when a requester attempts to access a protected resource at a host

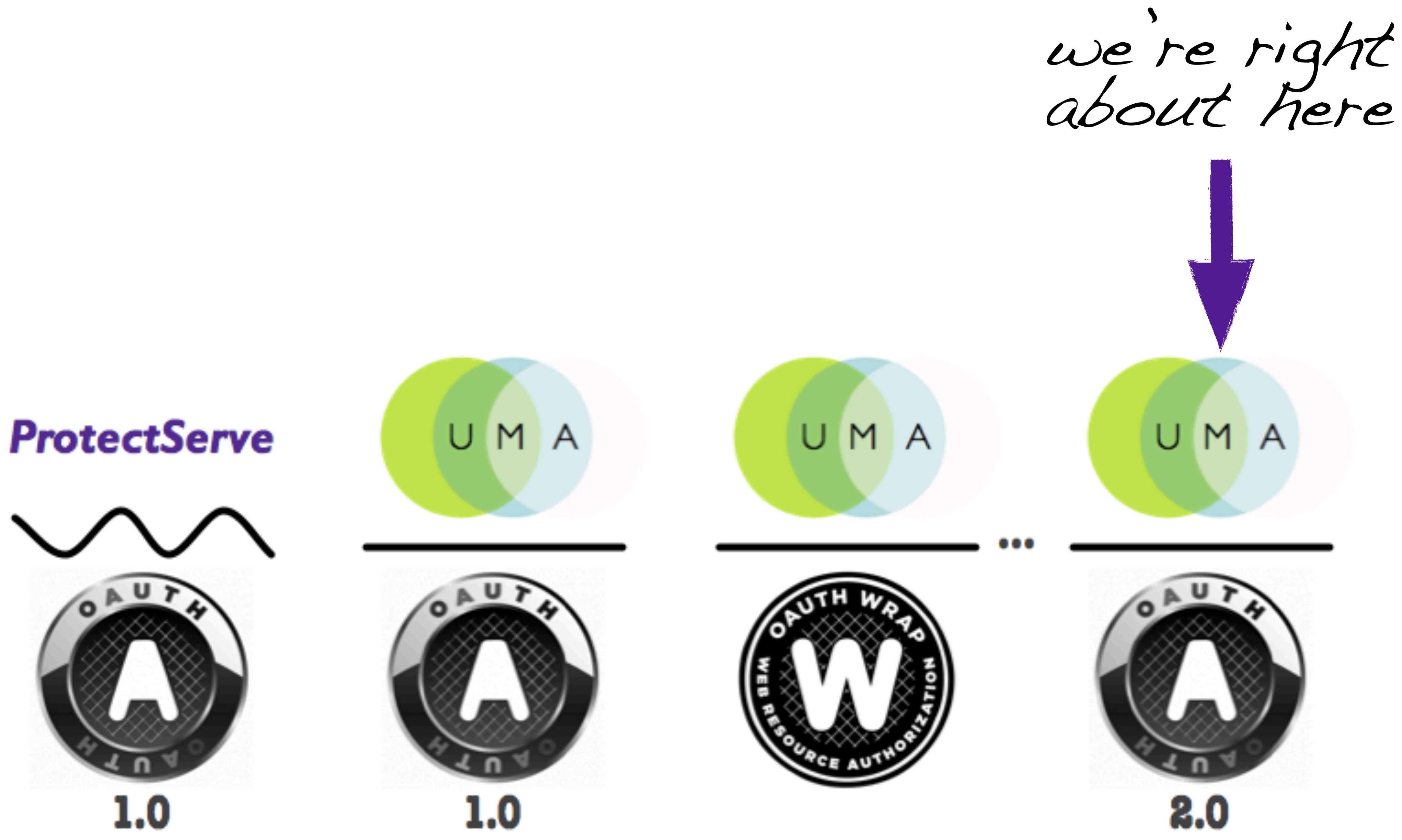
enforces access to the protected resources it hosts, as decided by an authorization manager

carries out an authorizing user's policies governing access to a protected resource

seeks access to a protected resource

a web user, or a corporation or other legal person, that uses a requester to seek access to a protected resource

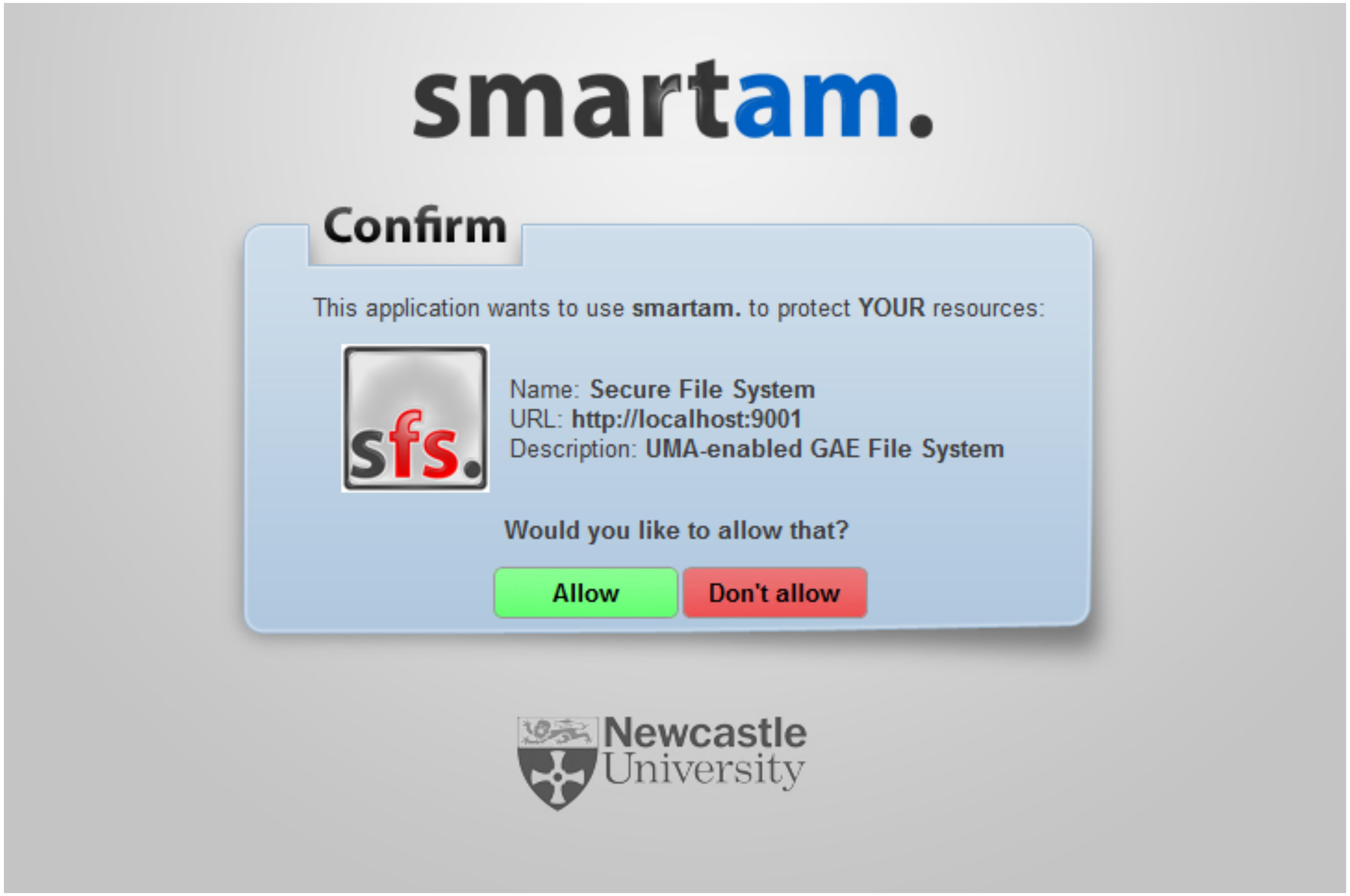
UMA's history with OAuth



UMA has three steps

1. Protect a resource
   - Alice introduces her Calendar host to CopMonkey: "When CopMonkey says whether to let someone in, do what he says"
2. Get authorization
   - Bob tries to subscribe to Alice's calendar but his client has to get a token for him, and he has to present (say) an identity claim to CopMonkey meets Alice's policy
3. Get access
   - Bob now has an access token with the necessary scope to use at the Calendar host: "This means Alice thinks it's okay"

UMA leverages OAuth twice: host-AM and requester-AM

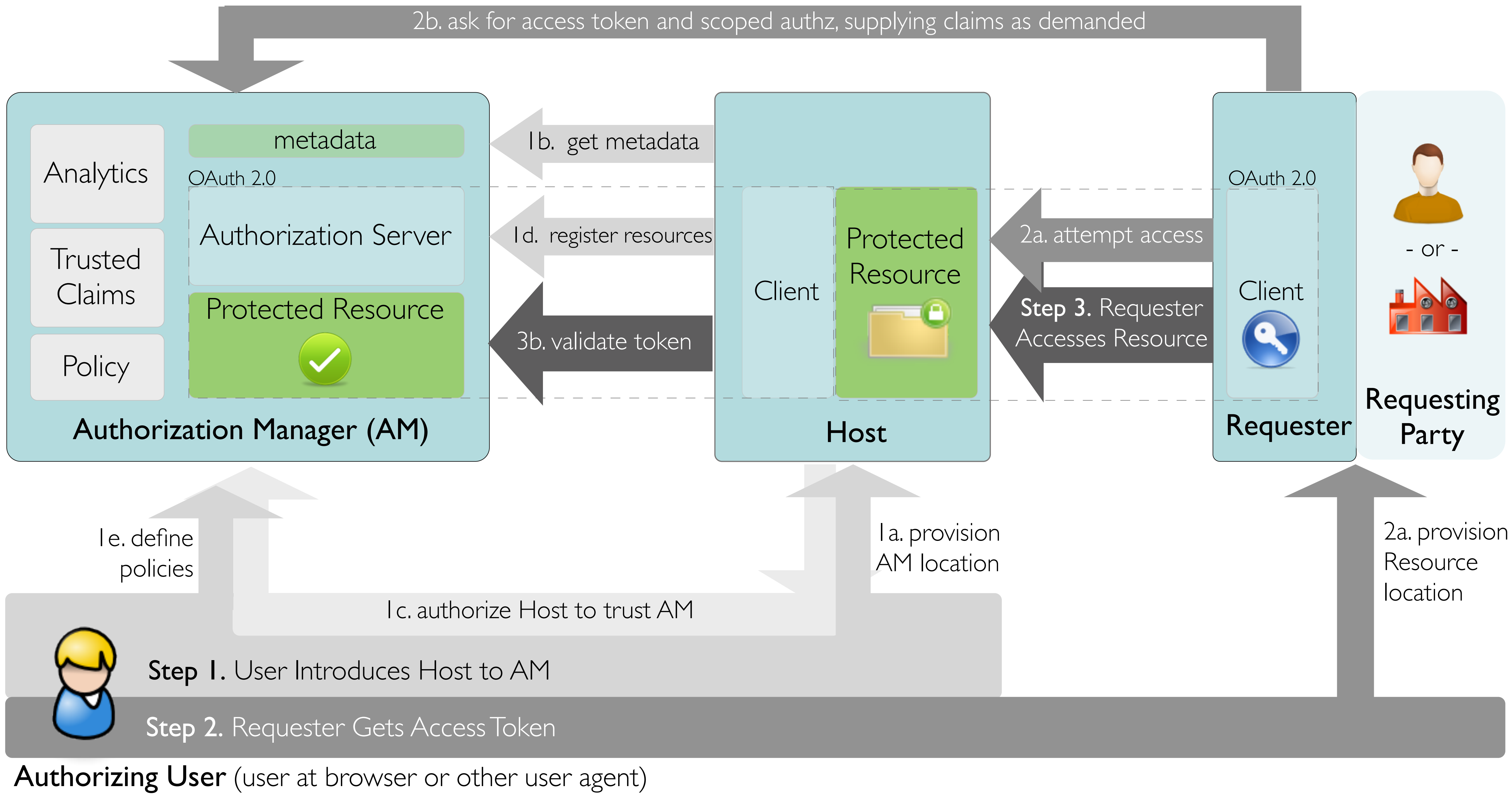


This application wants to use **smartam.** to protect **YOUR** resources:

Name: **Secure File System**
URL: **http://localhost:9001**
Description: **UMA-enabled GAE File System**

Would you like to allow that?

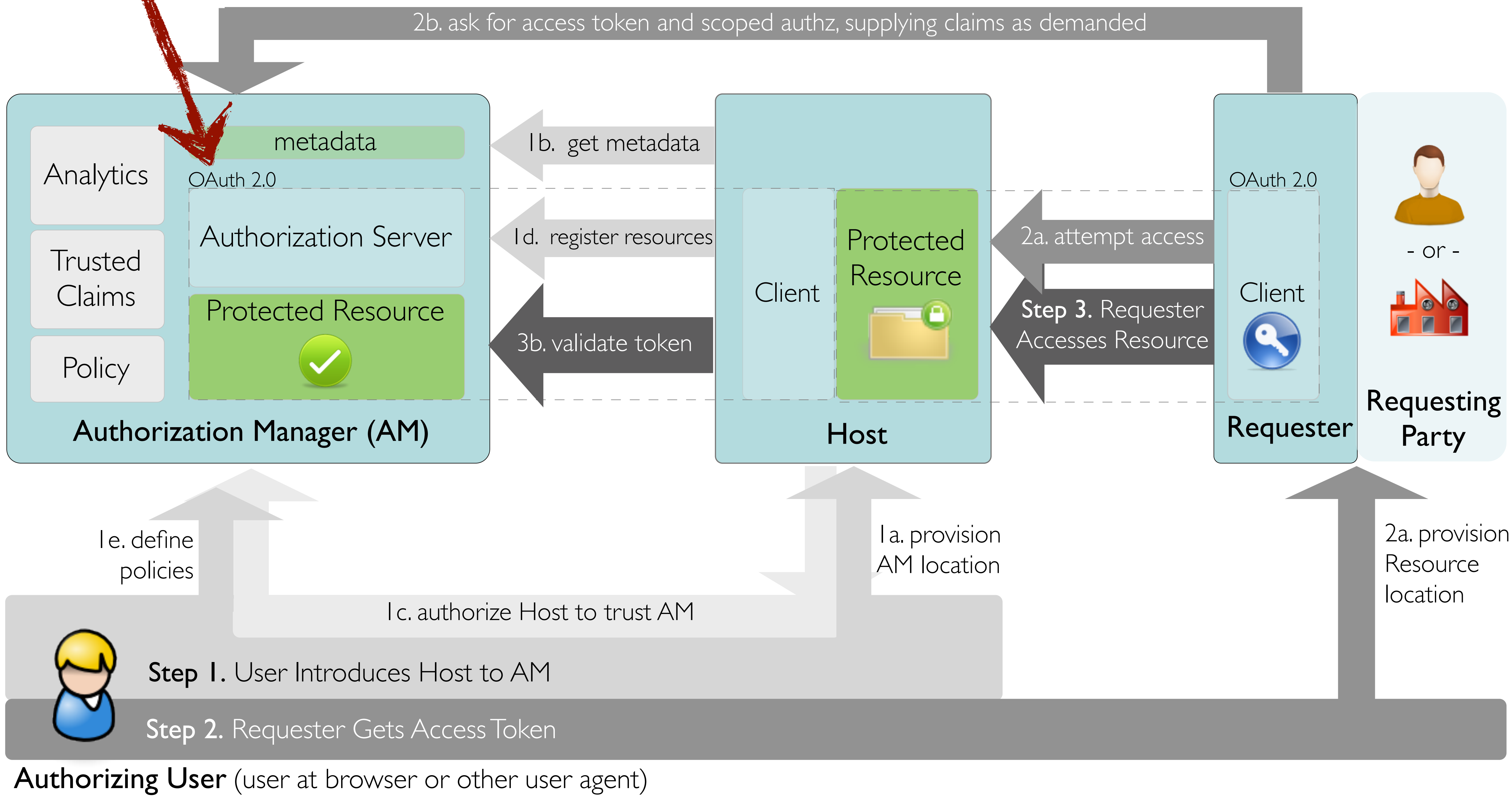
Overall UMA flow



2b. ask for access token and scoped authz, supplying claims as demanded
Analytics
metadata
OAuth 2.0
Authorization Server
Trusted Claims
Protected Resource
Policy
Authorization Manager (AM)
1b. get metadata
1d. register resources
3b. validate token
Client
Protected Resource
Host
2a. attempt access
Step 3. Requester Accesses Resource
OAuth 2.0
Client
Requester
- or -
Requesting Party
1e. define policies
1a. provision AM location
2a. provision Resource location
1c. authorize Host to trust AM
Step 1. User Introduces Host to AM
Step 2. Requester Gets Access Token
Authorizing User (user at browser or other user agent)

host gets access token to use at AM's protect authz API; ideally useser can tell host dynamically to use this AM, a la OpenID Provider discovery

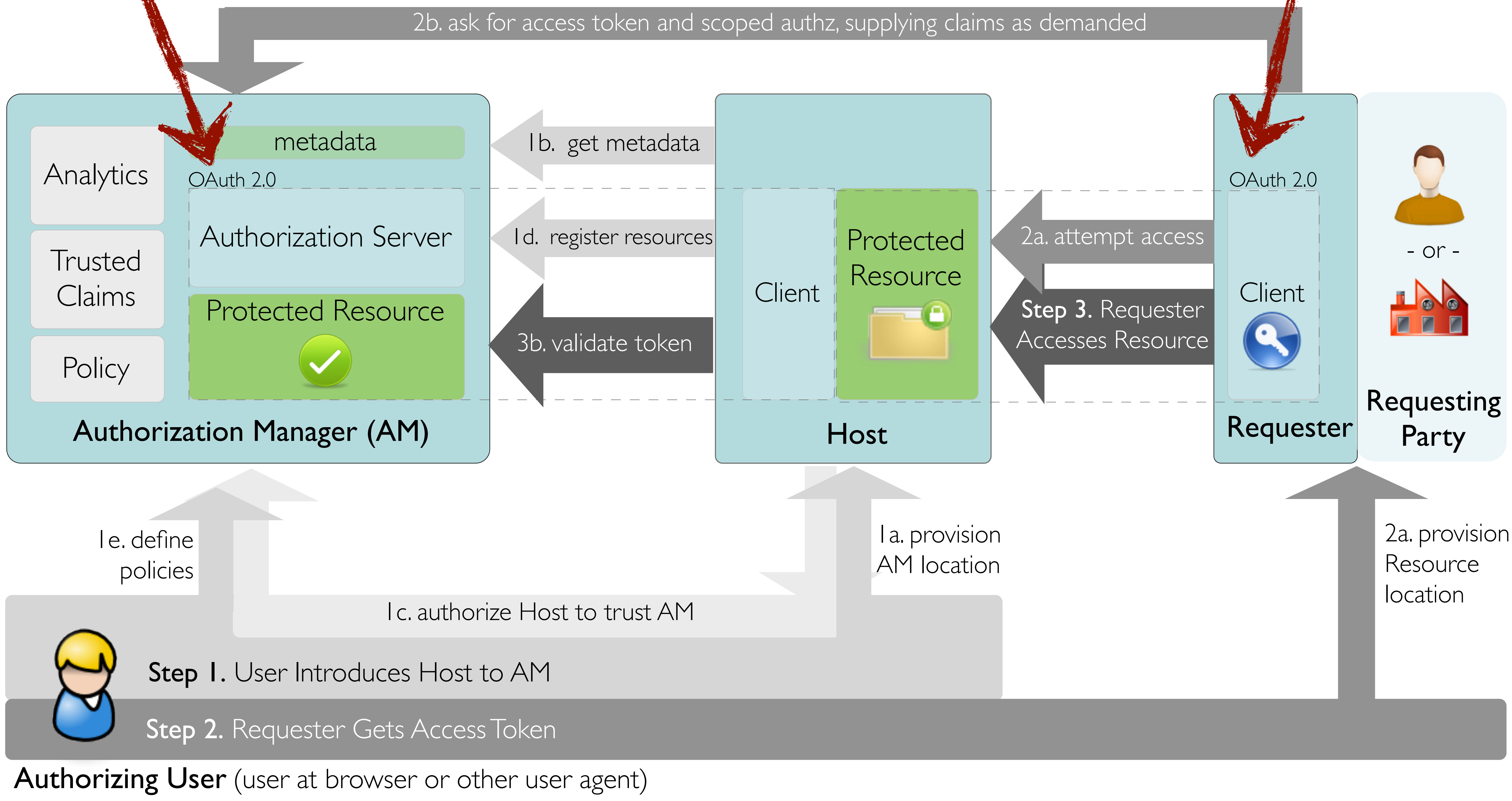
Overall UMA flow


2b. ask for access token and scoped authz, supplying claims as demanded
Analytics
Trusted Claims
Policy
metadata
OAuth 2.0
Authorization Server
Protected Resource
Authorization Manager (AM)
1b. get metadata
1d. register resources
3b. validate token
Client
Protected Resource
Host
2a. attempt access
Step 3. Requester Accesses Resource
OAuth 2.0
Client
Requester
- or -
Requesting Party
1e. define policies
1c. authorize Host to trust AM
1a. provision AM location
2a. provision Resource location
Step 1. User Introduces Host to AM
Step 2. Requester Gets Access Token
Authorizing User (user at browser or other user agent)

Overall UMA flow

host gets access token to use at AM's protect authz API; ideally useser can tell host dynamically to use this AM, a la OpenID Provider discovery

requesting party needs to approach host dynamically, to allow authz user to advertise resource availability widely, judging access suitability through claims alone


2b. ask for access token and scoped authz, supplying claims as demanded
Analytics
Trusted Claims
Policy
metadata
OAuth 2.0
Authorization Server
Protected Resource
Authorization Manager (AM)
1b. get metadata
1d. register resources
3b. validate token
Client
Protected Resource
Host
2a. attempt access
Step 3. Requester Accesses Resource
OAuth 2.0
Client
Requester
- or -
Requesting Party
1e. define policies
1a. provision AM location
2a. provision Resource location
1c. authorize Host to trust AM
Step 1. User Introduces Host to AM
Step 2. Requester Gets Access Token
Authorizing User (user at browser or other user agent)

UMAnitarians submitted draft-oauth-client-registration

- We need the option of dynamically issued client credentials for roughly the same reasons and with the same constraints (battle DoS attacks) as others do
- rev 00 of the I-D (it expired recently)
- We would be happy to revise it to state our emerging understanding of our requirements
- (Our emerging process for claims negotiation, leading to meaningful authz-scoped access, means we can push off strong client-side authentication to later in the process; stay tuned for more on this)

Backup slides

Comparing OAuth2 and UMA: terms

• resource owner	➡ authorizing user
• resource server	➡ host
• authorization server	➡ authorization manager
• client	➡ requester

Comparing OAuth2 and UMA: concepts

- There is one resource owner in the picture, on "both sides"

➡ The authorizing user may be granting access to a truly autonomous party

- The resource server respects access tokens from "its" authz server

➡ The host outsources authz jobs to an authz manager chosen by the user

- The authz server issues access tokens based on the client's ability to authenticate

➡ The authz manager issues tokens based on user policy and "claims" conveyed by the requester

Comparing OAuth2 and UMA: dynamic trust

- The client and server sides must meet outside the resource-owner context ahead of time
- The resource server meets its authz server ahead of time and is tightly coupled with it
- The resource server validates tokens in an unspecified manner, assumed locally

➡ A requester can walk up to a protected resource and attempt to get access without registering first

➡ The authz user can mediate the introduction of each of his hosts to the authz manager he wants it to use

➡ The host asks the authz manager to validate tokens in real time (JWT will allow us to avoid this)

Comparing OAuth2 and UMA: protocol

- Two major steps: token-getting (with multiple flow options) and token-using
- User delegation flows and autonomous client flows
- Resource and authz servers are generally not expected to communicate directly vs. through the token

➡ Three major steps: host/authz manager introduction (trust), token/authz-getting, and token-using

➡ (Emerging:) Token issuance is between autonomous parties; claims negotiation deals with delegation piece

➡ Authz manager gives host its own access token; host uses it to supply resource details and request token validation