User-Managed Access (UMA)

http://tinyurl.com/umawg

@UMAWG

10 December 2010

Key to the talk track: Instructions are in Courier. Optional bits are in parentheses. If slides 8, 9, 11, and 14 are included, the talk is about 25 minutes. If excluded, it is about 15 minutes.

(Abstract: The User-Managed Access protocol builds on OAuth to take Web data sharing to the next level, for both users and developers. If OAuth lets you hand out valet keys to your Web resources and API endpoints, UMA aims to give you a resource doorman -- on duty 24/7, handling requests according to your strict instructions, and able to give you a full report whenever you wander through the lobby. In this session we'll discuss UMA's benefits, sample use cases, specification and implementation progress to date, and next steps.)

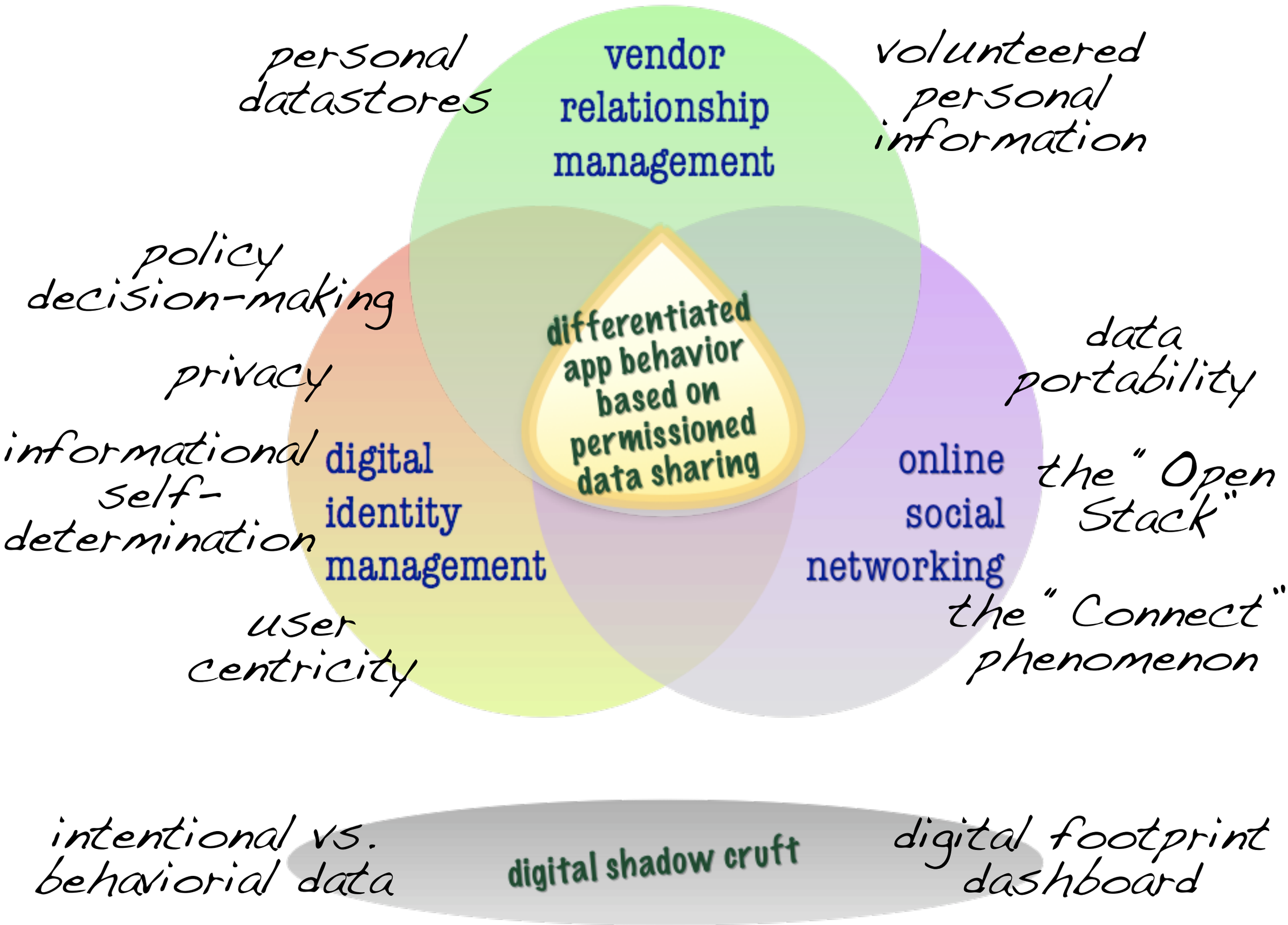
Privacy is not about secrecy

> The goal of a flexible, user-centric identity management infrastructure must be to allow the user to quickly determine what information will be revealed to which parties and for what purposes, how trustworthy those parties are and how they will handle the information, and what the consequences of sharing their information will be"
>
> – Ann Cavoukian, Information and Privacy Commissioner of Ontario, **Privacy in the Clouds** paper

It's about context, control, choice, and respect

It's becoming widely understood that privacy can't usefully be accomplished by perfect secrecy. In order to be well served in various capacities, people need to share sensitive personal information. A great example is when you share details of a health problem with a doctor. The trick in delivering privacy is to respect, and meet, a person's expectations around sharing.

This quote from Ann Cavoukian packs a lot into a small space. There's reason to believe that when she says **quickly determine** she means to **quickly find out**, so that consent can be granted or withheld. However, the UMA group is working to expand the possible solutions to include **quickly and easily controlling** what information will be revealed to which parties and for what purposes.



(4 builds)

The UMA work has been inspired by use cases coming from many different worlds.

1. Identity and access management solutions put a premium on user control of attribute sharing with a goal of minimal disclosure, often because of privacy regulations. In the enterprise, attribute sharing is more about access control than it is about personalization.

2. The Web 2.0 world encourages more and more sharing, so it exerts a pull in the opposite direction. Often the information shared is user-generated content rather than personal attributes. The Facebook Connect-style interaction, where you log in to consent to sharing from a home site to a third-party site, is a powerful pattern that UMA builds on.

3. The VRM movement takes an interestingly sophisticated view: in engaging in online commerce, individuals would likely share more information if vendors had the capacity to accept it and be respectful of it. Think of personal data sharing as **floating** to an optimal level instead of being pushed down or pulled up.

All of these realms have a problem in common: changing application behavior based on permissioned data sharing.

4. If data is shared without permission, we experience creepy effects like unwanted targeted marketing. To avoid this, UMA imagines that an individual can go to a single place on the Web -- a digital footprint dashboard -- to monitor and manage sharing rules, no matter where their actual online stuff actually lives.

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
- Striving to be simple, OAuth-based, identifier-agnostic, RESTful, modular, generative, and developed rapidly

Here you have UMA in a nutshell.

...

(Participants in the UMA Work Group are known as UMAnitarians ("ooh-man-ih-TARE-ians"). We hope you'll consider becoming one!)

(Why did the group select the IETF as a destination? Because UMA is deliberately building on top of OAuth and this is where OAuth work is being conducted.)

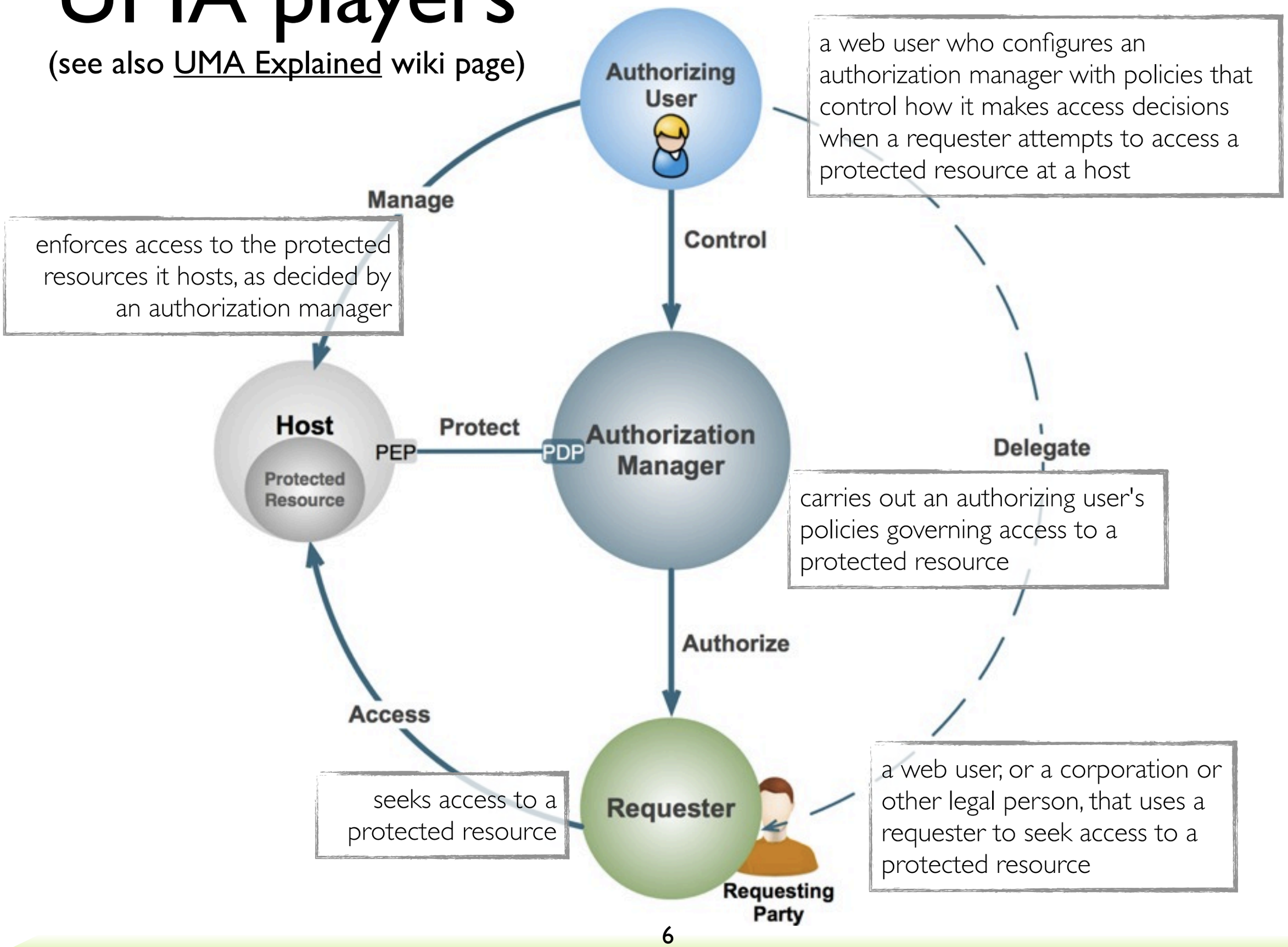
(Why did the group select OAuth as a technical substrate? Because it shares many of the same design principles UMA has: simple, doesn't dictate a single global identifier namespace, modular, and generative.)

Selective sharing shortcomings	User-Managed Access solutions
Little sophistication and consistency in Web 2.0 access control – e.g., Google Calendar vs. Flickr vs. TripIt	A way for any web app to provide sophisticated access control merely by outsourcing it, à la SSO
Rules for selective sharing can't be applied to different apps – the "family" ACL has to keep being rebuilt	Selective-sharing policies can be mapped to content at multiple hosts
Selective sharing is largely identity-based and static	Conditions for access can be "claims-based", with claims tested when access is attempted – e.g., "anyone over 18"
Individual is only in a position to consent to sharing and to site terms of service, not dictate terms of access	Sharing policies form a barrier; requesting parties have to agree to terms or otherwise prove suitability
Individual can't get a global view of every party they've said can get access to their data and content	Sharing policies and sharing authorizations all come out of a single "hub" application
OAuth today only enables the protection of singular API endpoints for web services	Any Web resource with a URL, and any access scope on it, can be protected – e.g., sharing a status update API or a single tweet

Note: ACL = access control list.

UMA players

(see also UMA Explained wiki page)



(5 builds)

The UMA protocol defines these players.

1. An individual web user, let's call her Alice, is the "user" in "User-Managed Access". She...

2. ...keeps calendars in one place, photos in another, her social graph in another, and her employer runs an app that is willing to state that she's an employee for those who need to know. These places are known as hosts and her data and content residing there are considered protected resources.

3. Alice has introduced all of her hosts to her chosen hub for selective sharing, known as an authorization manager or "AM". The participants in the UMA Work Group tend to refer to a fictitious authorization manager called CopMonkey, because it's Alice's data-sharing traffic policeman on the Web. But real authorization managers are likely to be identity providers serving lots of consumers already.

4. Alice has set up some rules for what someone needs to do in order to be considered a suitable candidate to get access to Alice's stuff. If she's sharing one of her calendars, the "someone" might be her friend Bob who watches her cats when she's away, or it might be essentially Alice herself again through a different calendar program she uses for aggregating event data, or it might be her credit-card company so that they know when she's traveling and likelier to be spending money in foreign cities so they can do better fraud protection. We call these scenarios person-to-person sharing, person-to-self sharing, and person-to-organization sharing. They have different implications for user interfaces and for liability in agreeing to Alice's terms of access.

5. The requesting party, like Alice with a browser or mobile app, needs to interact with computer systems by using a software program of its own. All of these programs have to speak the UMA protocol. The protocol endpoint that the requesting party uses is called a requester.

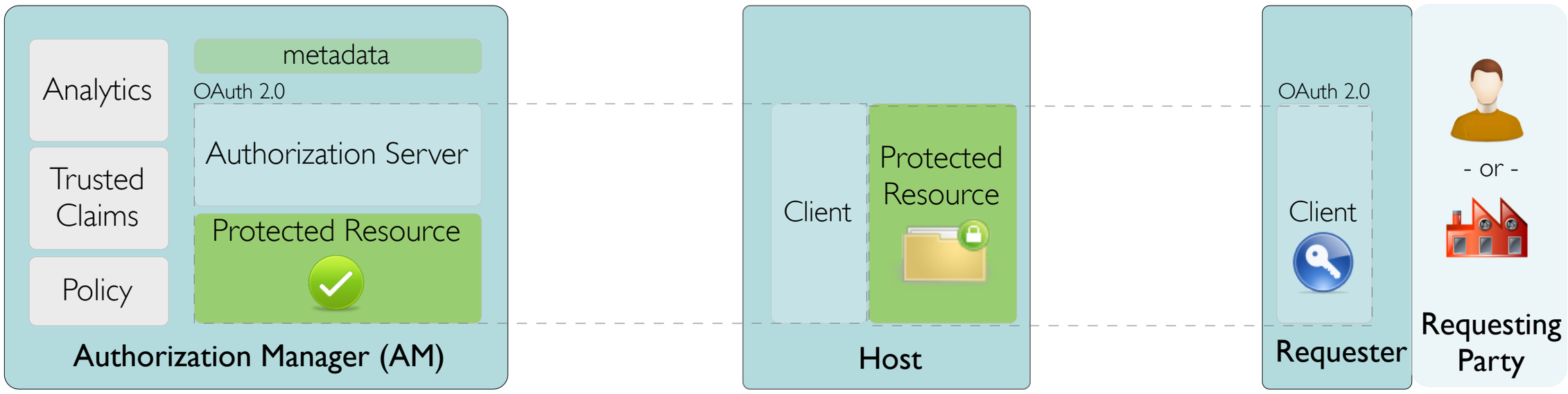
UMA has three steps

1. Trust a token
    - Alice introduces her Calendar host to CopMonkey: "When CopMonkey says whether to let someone in, do what he says"
2. Get a token
    - A travel marketing company tries to subscribe to Alice's calendar but it has to agree to her terms of use: "All right, all right, I'm clicking the 'I Agree' button"
3. Use a token
    - The marketing company now has an OAuth access token to use at the Calendar host: "This means Alice thinks it's okay"

The UMA protocol has three steps. Step one tells a host to trust Alice's chosen AM. Step two makes a requester prove it's worthy of access. Step three lets it get through to the protected resource.

Steps two and three are enhanced versions of OAuth. UMA's step one is weird and different because it **embeds** a whole new mini-OAuth interaction inside it.

The players again



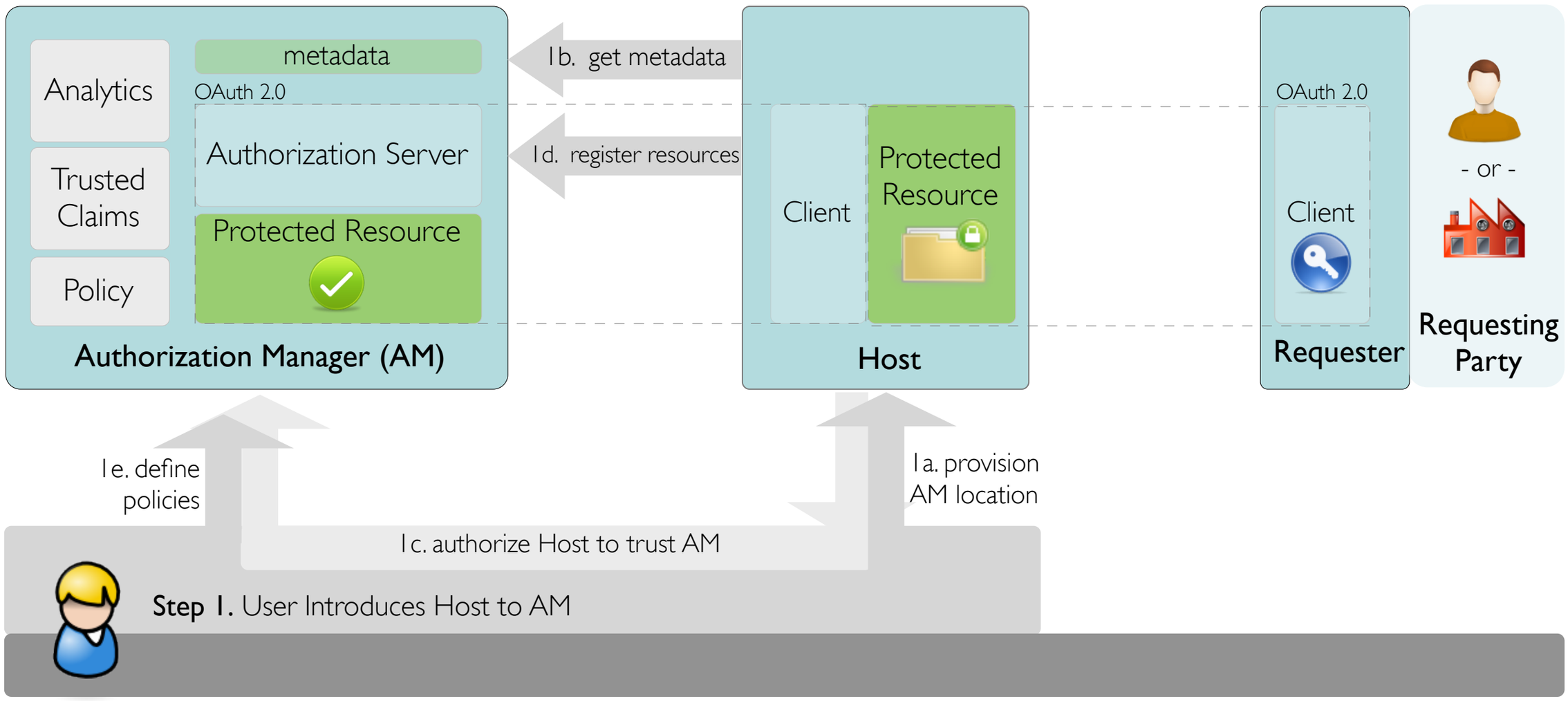
Authorizing User (user at browser or other user agent)

(slides 8, 9, 11, and 14 are optional)

Let's review the protocol flow briefly. Here you see Alice the authorizing user in the lower left corner, the requesting party along the right side, the AM on the left, and the host in the middle.

Note that UMA's entities are enhanced versions of OAuth ones. An UMA AM is basically an OAuth authorization server. An UMA host is like an OAuth resource server, and an UMA requester is a super-duper OAuth client.

Step 1 protocol flow

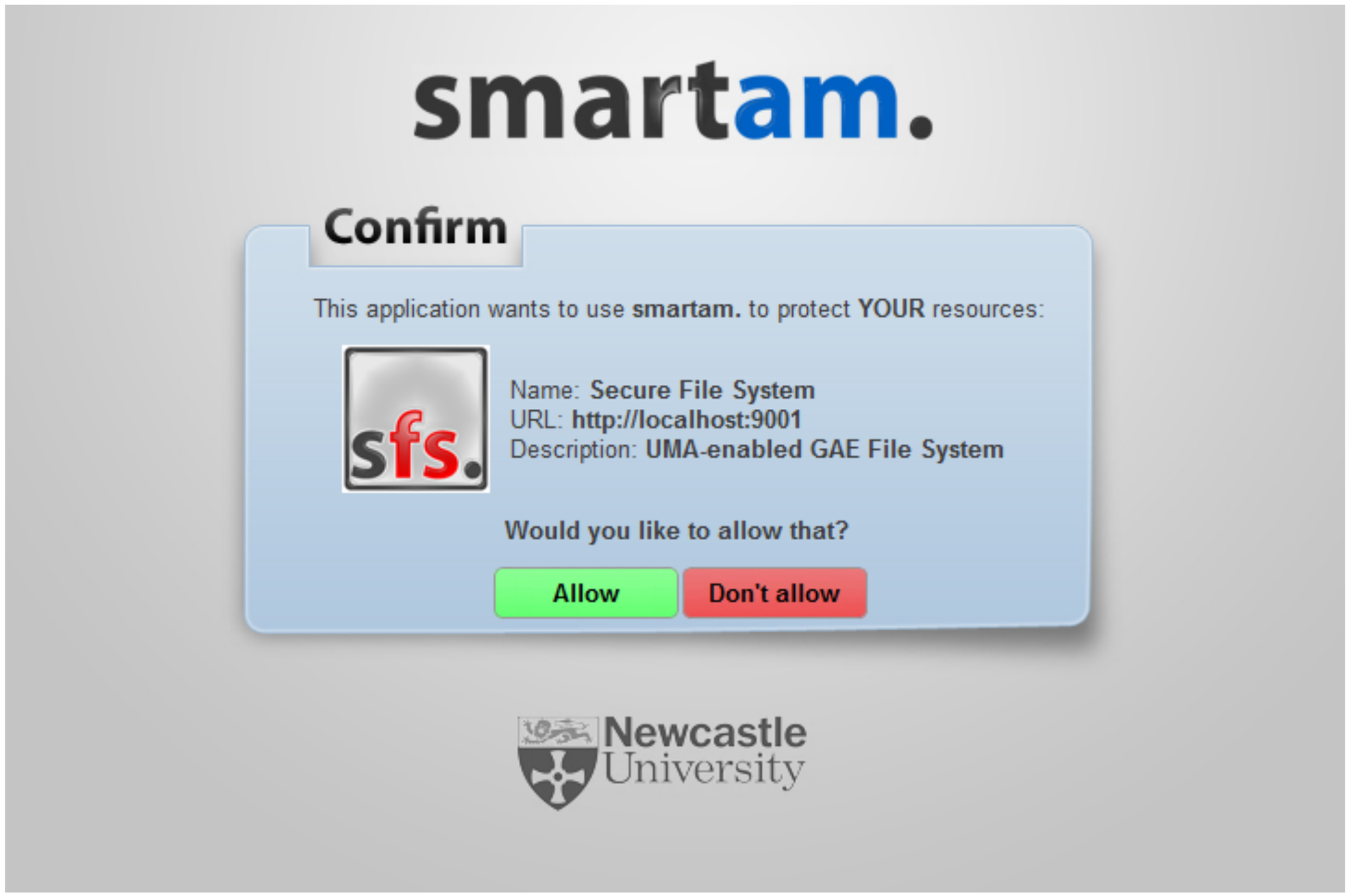


(slides 8, 9, 11, and 14 are optional)

In step 1, Alice introduces the host to the AM. This is the weird step that isn't in OAuth at all, but that includes an embedded OAuth interaction.

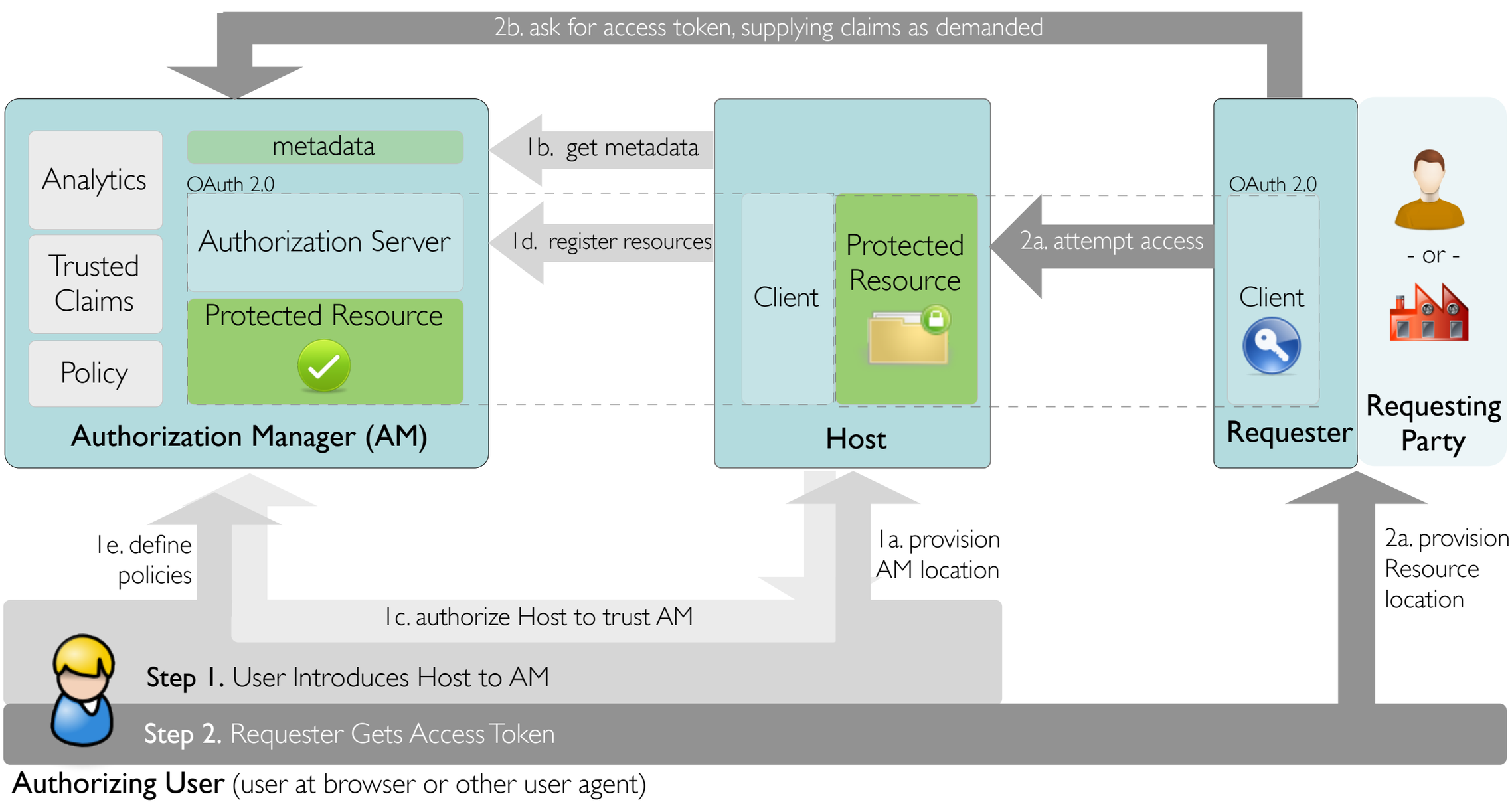
In order for the host to trust the AM, it gets its very own OAuth access token to use at a special authorization interface at the AM, meant just for hosts. One of the features it includes is a place where the host can tell the AM what resources to protect on the user's behalf.

A possible UX for host-AM introduction



A team from Newcastle University is participating in the UMA group, and they have a project called “SMART” (for “Student-Managed Access to Online Resources”) to develop an UMA-compliant Java framework. They have done a user experience study that has influenced the group’s design process. Here is a screenshot of an OAuth-style confirmation screen to introduce a host and AM.

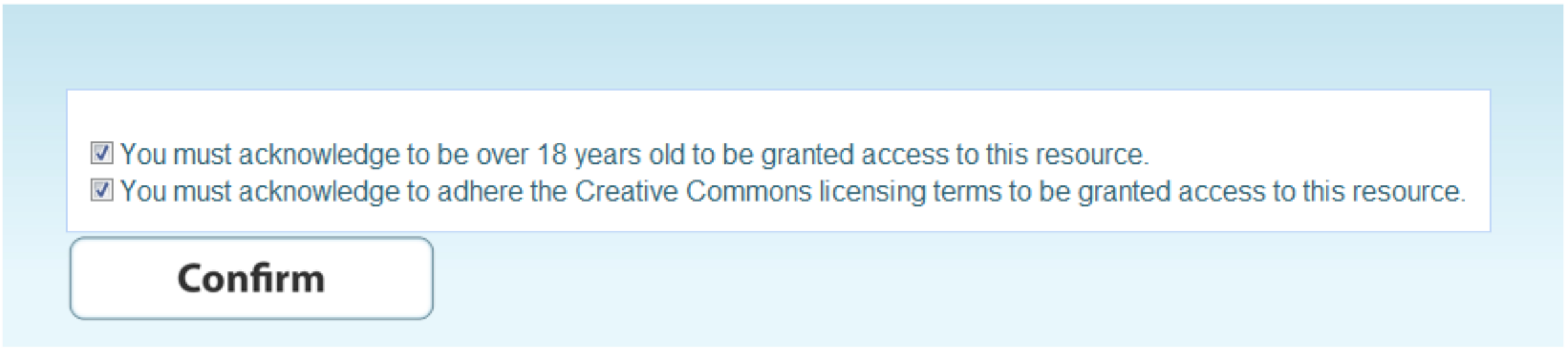
Step 2 protocol flow



(slides 8, 9, 11, and 14 are optional)

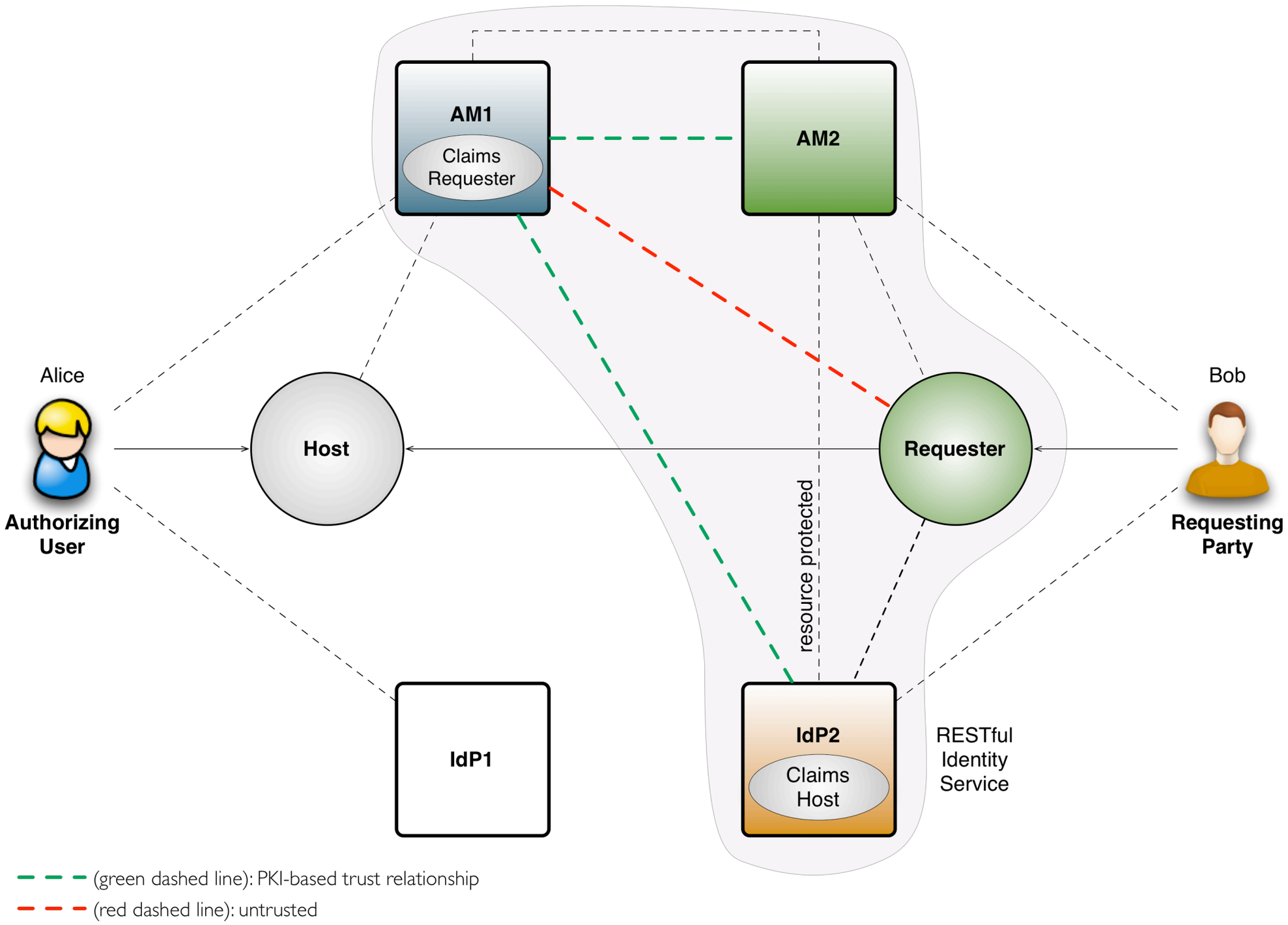
In step 2, the requester attempts access and is told where to go to try and get an OAuth access token. When it gets to the AM, it might get refused outright, or given an access token outright, or it might be asked to convey claims about the requesting party that satisfy the user's policy. This last option is an enhancement over OAuth.

A possible UX for self-asserted claims about promises



The SMART project has implemented self-asserted promissory claims as checkboxes that a requesting party is required to check.

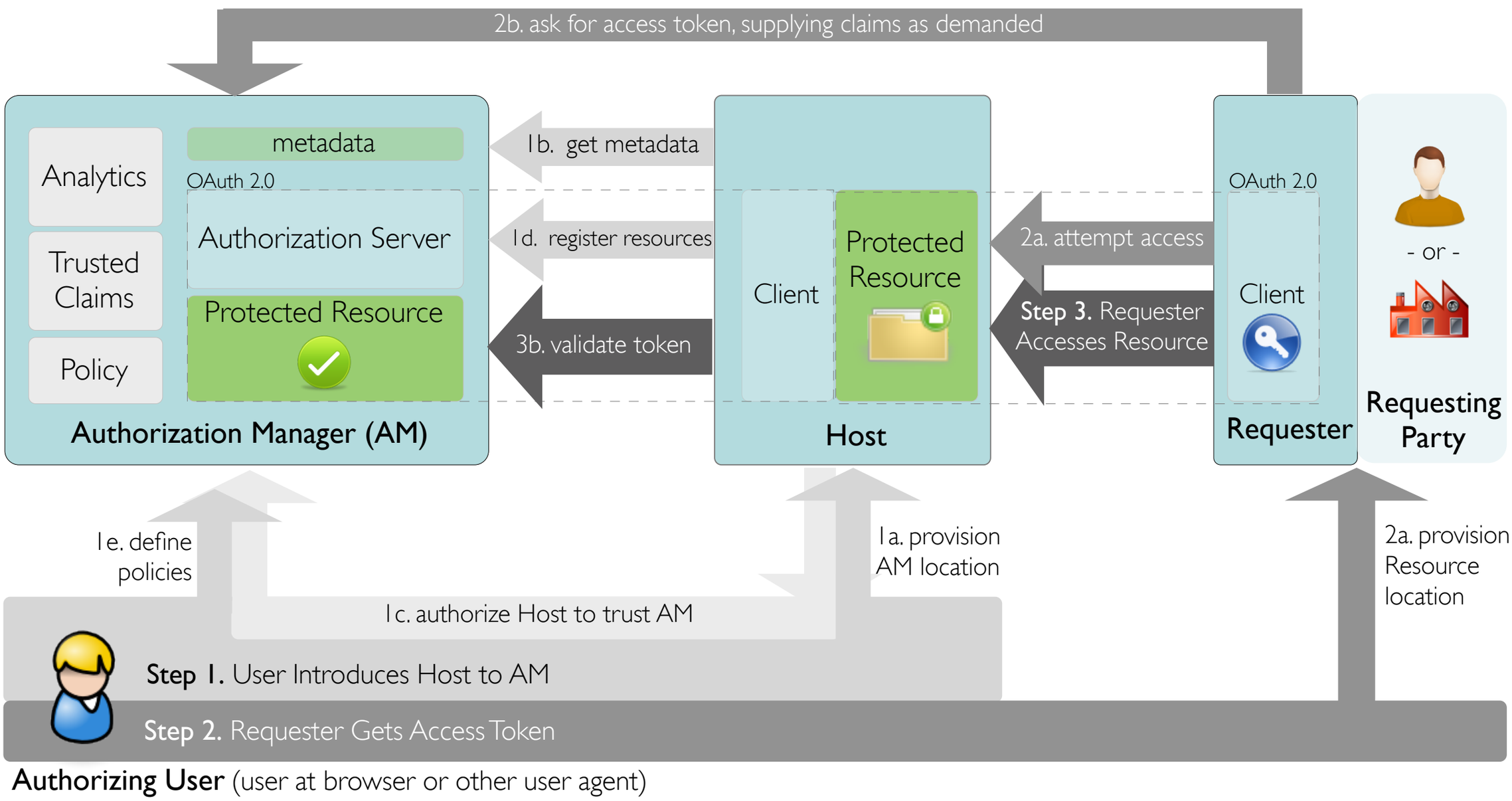
A potential claims trust model: make them UMA-protected resources



The group has begun work on an UMA extension that reuses UMA itself for “trusted claims” or “verified claims”. If Alice wants to make a calendar available to Bob, and wants Bob to prove it’s really him, she needs to ask him for something beyond just a self-asserted promise.

As we know from the identity assurance world, this is generally a hard problem! The UMA group wants to solve it as simply as possible, for simple use cases. So if Alice has a policy saying bob@gmail.com can read her travel calendar, the solution would be for Bob to treat his Google IdP as a host of UMA-protected claims about his email address there.

Step 3 protocol flow

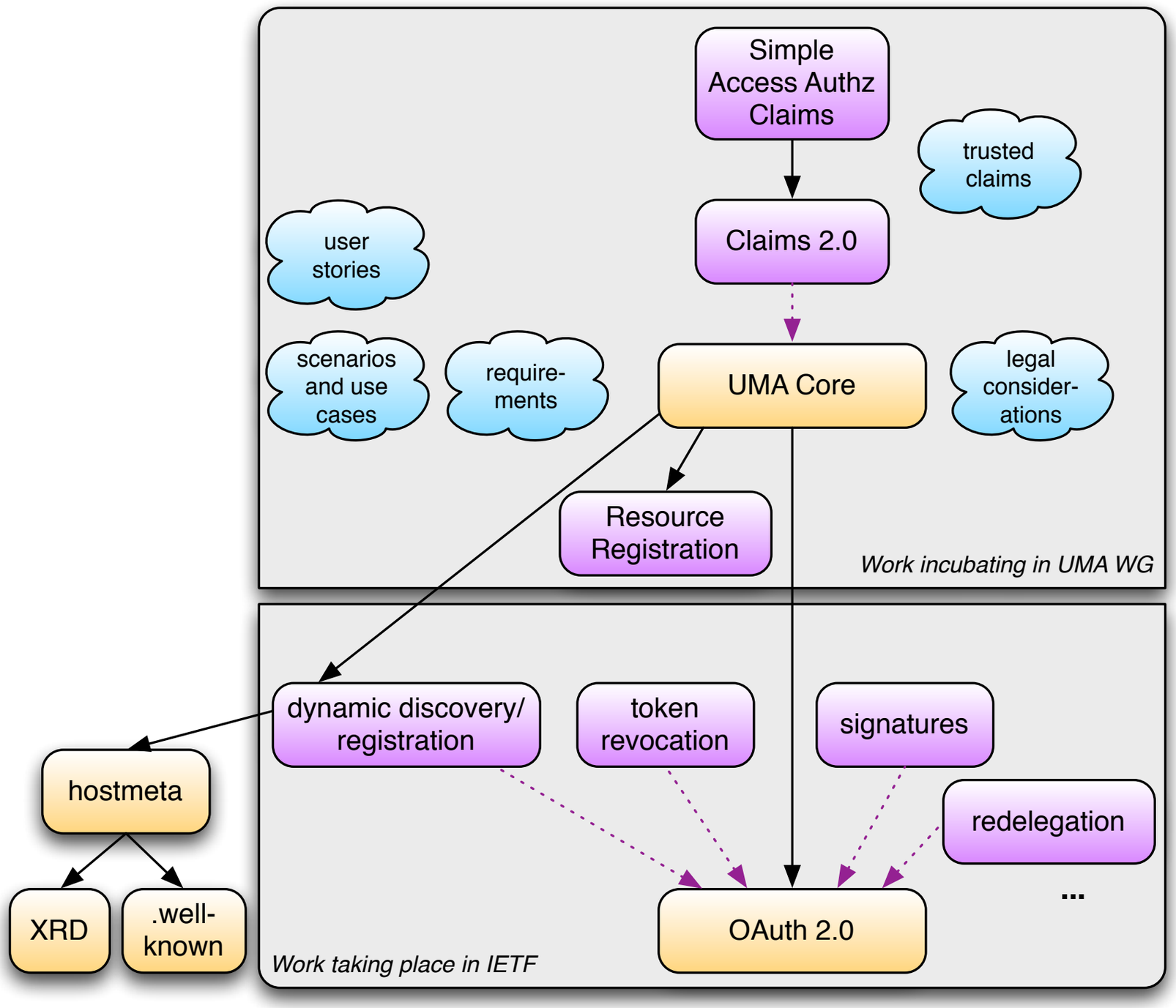


(slides 8, 9, 11, and 14 are optional)

In step 3, the requester attempts access, this time with the token in hand. OAuth doesn't standardize exactly how the resource server would validate the token. UMA builds in a standard way for the host to ask the AM outright. If it's good, the host lets the requester in.

Status of UMA development

(see also Working Drafts wiki page)



UMA Validator Bounty Program

(1 build)

The suite of UMA specs is complete, but the detail in each one varies.

The core spec outlines the three steps. The group is about to do a lot of work on conformance clauses to prepare for interoperability testing.

Participants from the group have already contributed one spec to the IETF, the dynamic registration spec. (This spec covers how an OAuth client can do dynamic self-registration at an OAuth authorization server, just like people can walk up to Web 2.0 sites and self-register to get an account. This is important when hosts and requesters both start talking to AM for the first time, to enable Alice to get them all working together the moment she wants them to.)

(The resource registration spec is new, but the group has worked on various versions of it for a very long time. It appears the OAuth group is starting to recognize the need for this piece as well.)

(The Claims 2.0 piece is optional but could be very powerful. It defines a simple JSON-based format to use in step 2 for getting claims from the requester. The UMA group hopes to leverage ongoing work on JSON tokens taking place in the OAuth group.)

The documents in blue are nontechnical. The group has collected a lot of scenarios and use cases and done a lot of analysis.

1. A number of satellite efforts are taking place.

As already mentioned, Newcastle University is doing an implementation and testing of various types. They are planning a test deployment within the university. They have already open-sourced their UMA-friendly OAuth implementation in Java, called “leeloo” (after the character in the movie The Fifth Element), and they are close to open-sourcing their larger framework, called UMA/j.

And within a few days the group will be concluding its 2010 bounty program for UMA conformance validation materials. Two submissions are anticipated, one being a conformance test plan with test cases.

Special thanks to Maciej Machulak and Domenico Catalano for screenshots and graphics

Thank you. Please visit the UMA wiki on the Kantara site for lots more information. And please consider joining the effort!

(This photo shows an implementation of the SMART authorization manager that has been ported to the Microsoft Surface.)