TheNetw.org

Authentication for your applications

Jan Hajek

@hajekj

jan.hajek@thenetw.org

Jan “Haichi” Hajek

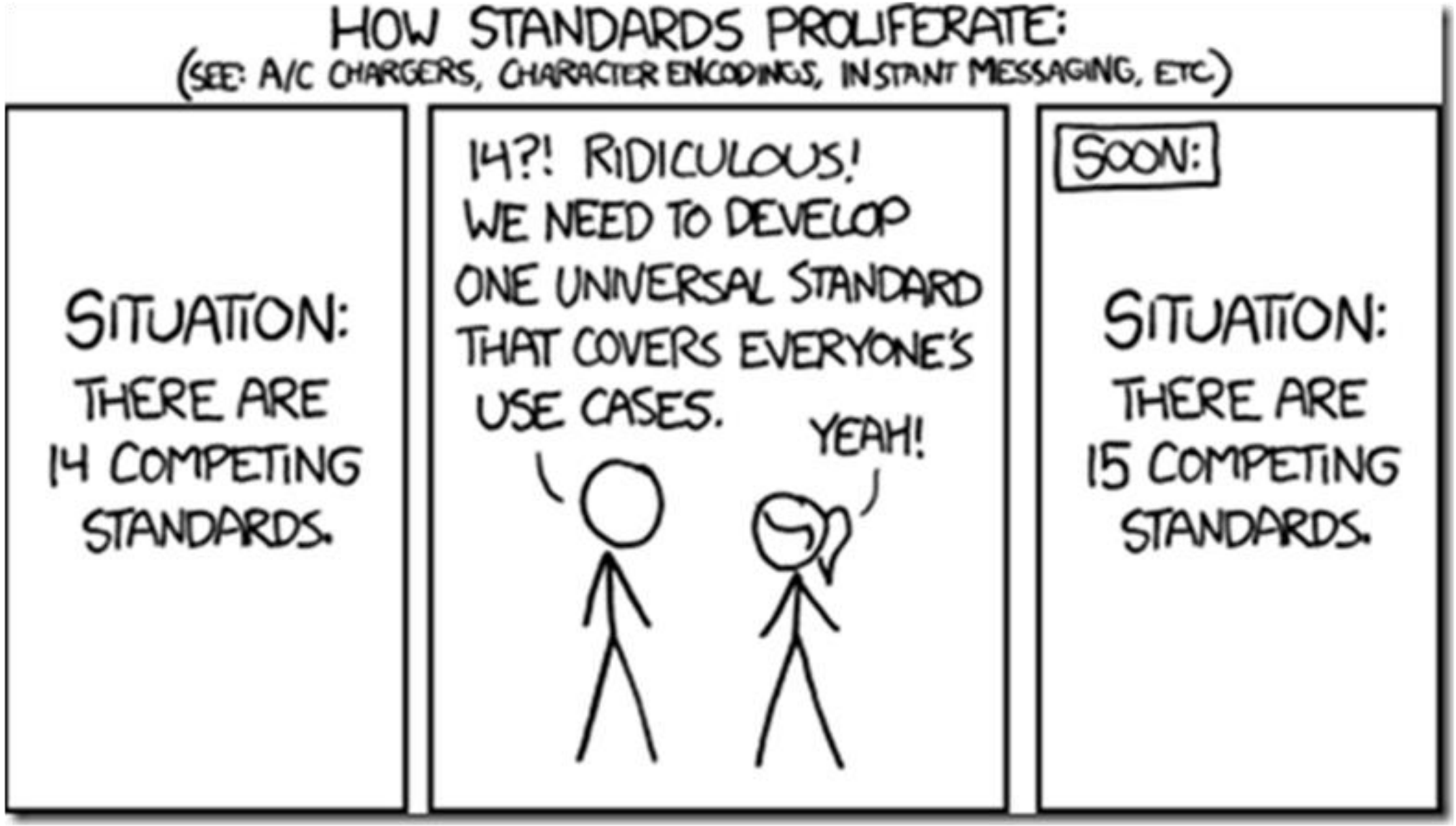
@hajekj

jan.hajek@thenetw.org

- Technical Fellow at TheNetw.org
- Microsoft MVP
- 10+ years of coding experience
- C#, JavaScript, Node.js, PHP
  - oauth2-azure author
- Co-creator of SkolniLogin.cz
- Gamer (PC and Xbox)

Many different protocols

- Windows Authentication
- SAML
- WS-Fed
- OAuth 1
- OpenID
- **OAuth 2**
- **OpenID Connect**


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Why care about Single Sign On?

- Managing usernames and passwords is pain
- Logins are universal, why reinvent the wheel?
- How to keep the passwords safe?

';--have i been pwned?

- Will users create a unique enough (and of course secure) password for my site?

Probably not. But they will log in with Twitter.

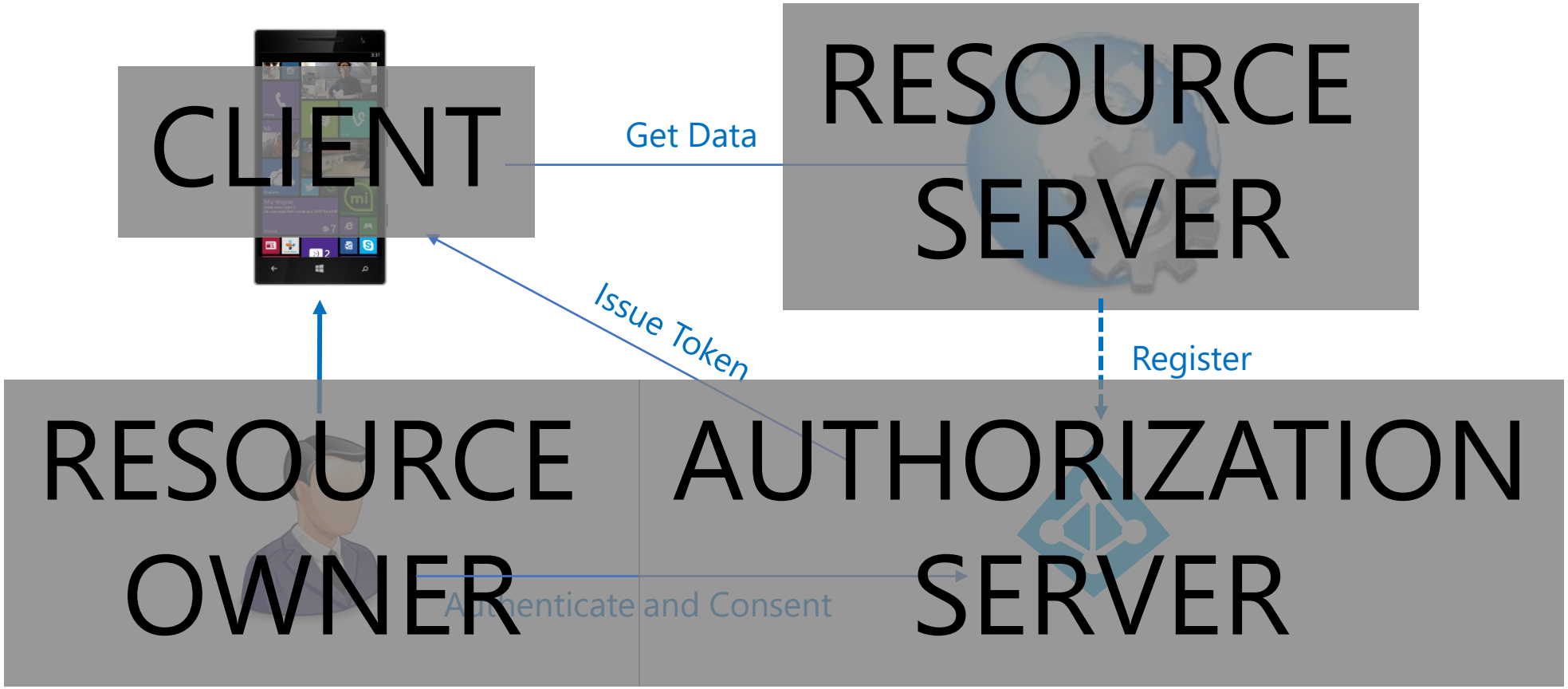
DEMO

Using various identity providers in ASP.NET Core

Identity vocabulary

- **Authentication:** Proving the user is who they say they are
- **Authorization:** Allowing users to access resources or complete actions
- **Identity:** The information we know about the user

OAuth 2.0 Concepts and Terminology



OAuth 2.0 Concepts and Terminology

- **Client**
  - Application that needs to use the resource
  - Various types –
    - browser-Web-App, Native, Daemons, etc.
  - Often end-user facing
  - E.g. Snapfish “Print shop” application
- **Resource Server**
  - Hosts the resource
  - Typically an API provider
    - E.g., Microsoft Graph API
  - Trusts tokens from an Authorization Server
  - E.g. OneDrive “Photo library”

- **Resource Owner**
  - Owner of the requested resource
  - Typically the user of the application
  - E.g. “Owner of the OneDrive account/photos”

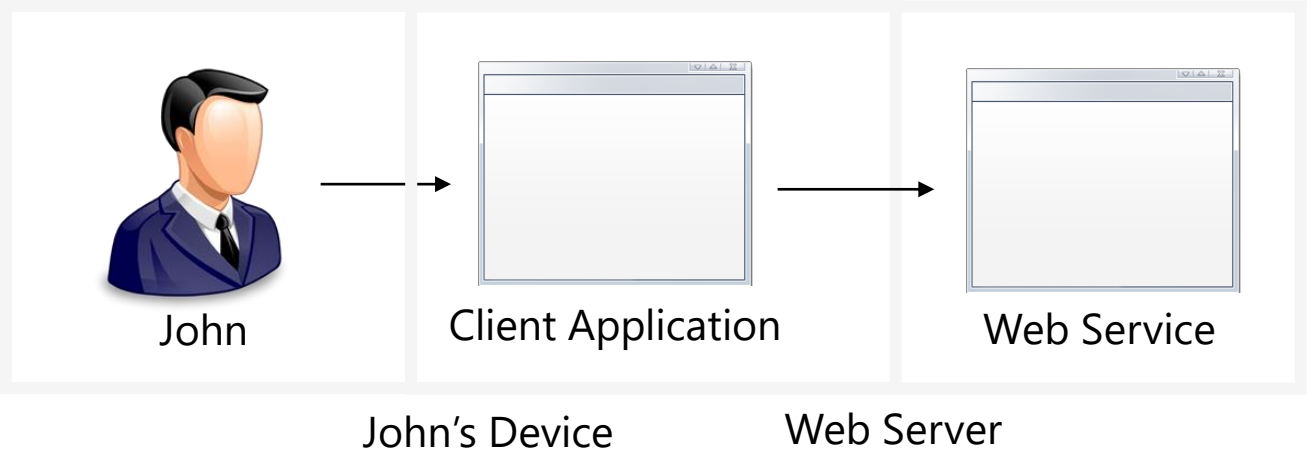
- **Authorization Server**
  - Issues access tokens to clients
  - Authenticates resource owners
  - Gets access consent from the resource owner
  - Could be “Photo library provider”

OAuth 2.0 Concepts and Terminology

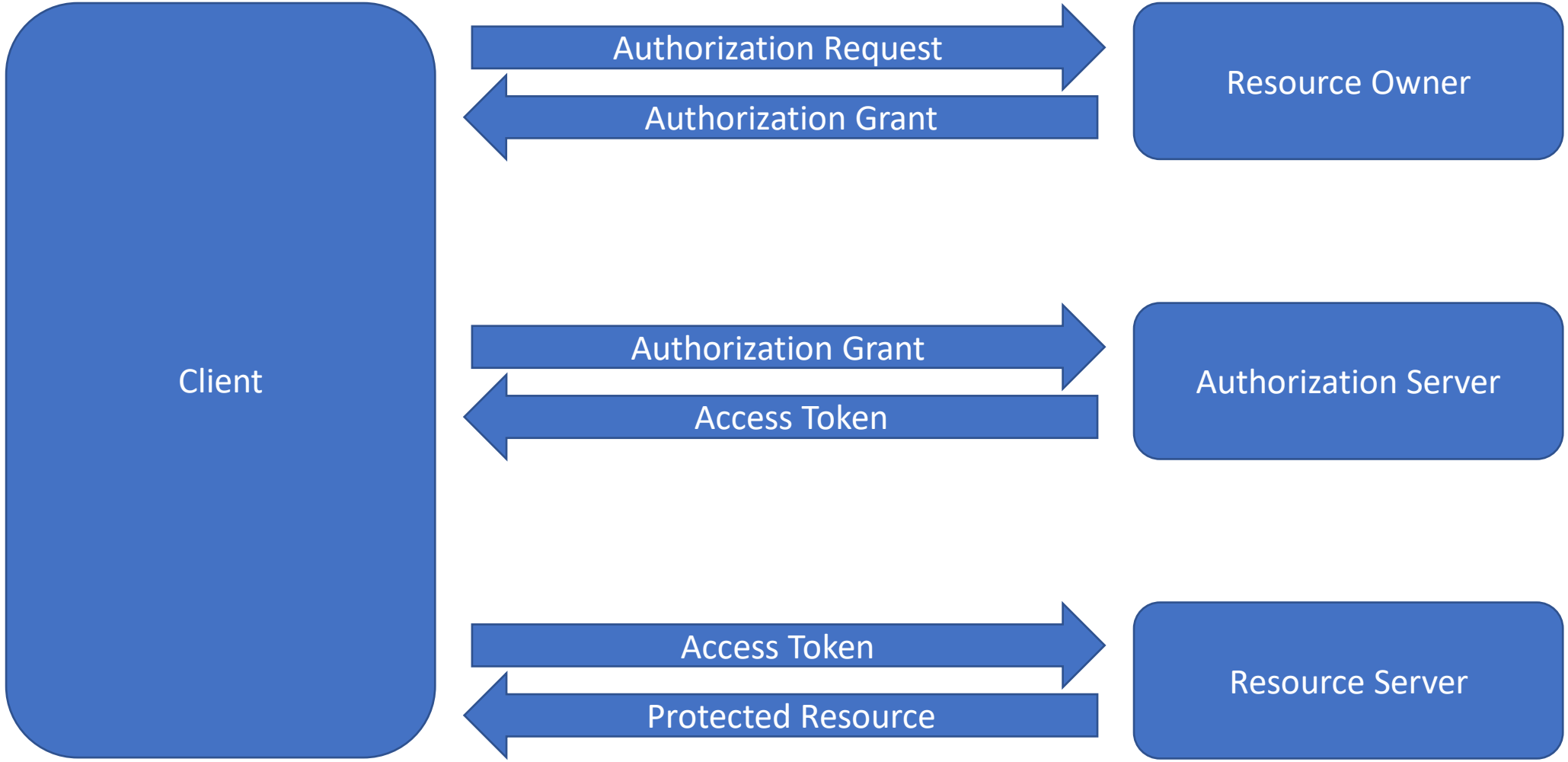
- **Client/Service Registration**
  - Tells Auth Server it is OK to issue tokens for this client to this resource server
  - May include limits on what the client can do
  - Client and service identities in AS namespace
- **Tokens**
  - JSON Web Tokens (JWT)
  - Access Tokens v. Refresh Tokens
  - “Bearer” tokens most common
  - Scope
- **Endpoints**
  - ‘Well-known’ locations at AS and in client
    - Authorization Endpoint
    - Token Endpoint
    - Client Redirect Endpoint
- **Flows**
  - Standard patterns for obtaining tokens
  - Describe how to interact w. above endpoints
  - Covers various devices + services scenarios

OAuth 2.0 Client Types

- **Public** – Can **not be trusted** to hold a secret, e.g. runs on a user device.
- **Confidential** – Can be **trusted** with secrets, e.g. web server.



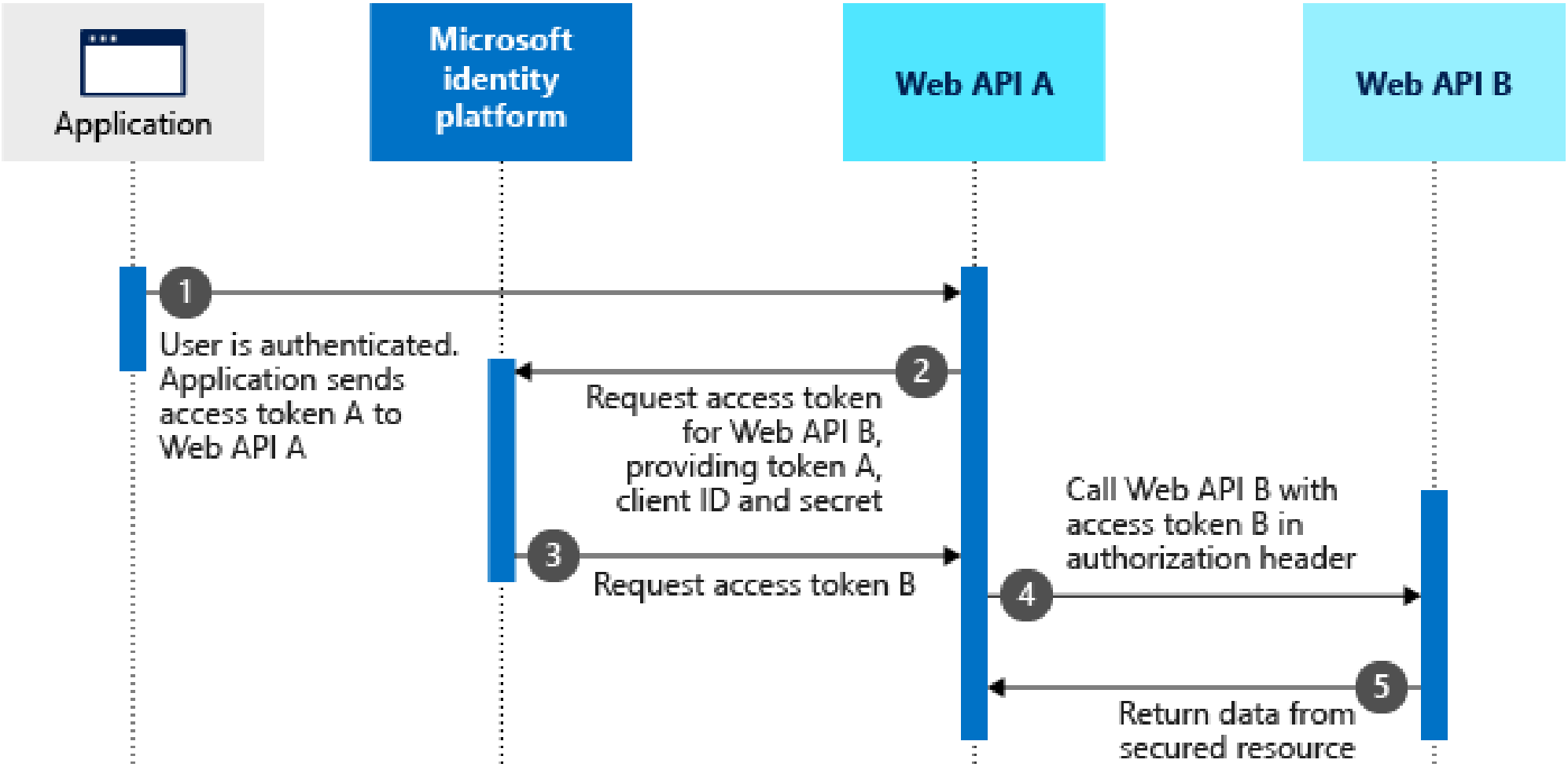
Protocol Flow



Why different flows?

- Client security – confidential vs public
- Can user enter consent in real time?
- Is http redirection possible?
- Potential for token exposure

On-behalf-of Flow



OpenID Connect

- Started as OpenID Authentication
- Became and ID wrapper around OAuth 2
- Standard way of defining identity for users

Bearer Tokens

- Added in OAuth 2
- If you have token, you are authenticated
- What are guarantees that the token came from a trusted source?

Facebook developer writes:

> *Hey guys, I'm using the freshly downloaded PHP API*
> *I've got eveything setup and when I login to authenticate, I get this error:*
> *Fatal error: Uncaught CurlException: 60: SSL certificate problem: verify that the CA cert is OK. Details:*
> *error:14090086:SSL routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed thrown in /src/facebook.php on line 512*

To which another developer replies:

> *[…] try this one:*
> *$opts[CURLOPT_SSL_VERIFYPEER] = false;*
> *$opts[CURLOPT_SSL_VERIFYHOST] = 2;*

And Facebook developer asks:

> *Is there any good reason why these shouldn't be a part of the core default $CURL_OPTS?*

JSON Web Tokens (JWTs)

- Lack of signatures mitigated by being signed…
- Bearer can be a signed JWT, which you would then verify
- Server can also require signed JWT from clients

What is JWT?!

- header + payload + key
- Header and payload are JSON objects
- Key is header + “.” + payload signed with a secret

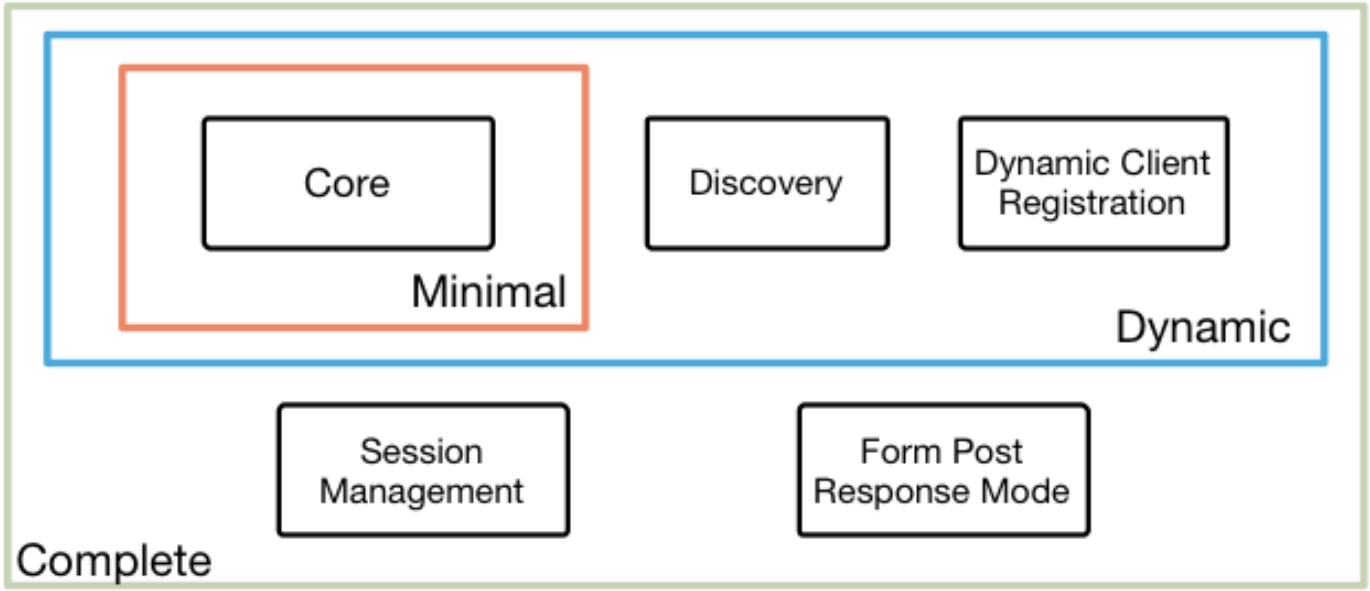
- jwt.io

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdW
IiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lI
iwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrH
DcEfxjoYZgeFONFh7HgQ

4 Feb 2014

OpenID Connect Protocol Suite

http://openid.net/connect



Underpinnings

It makes a lot of things simpler

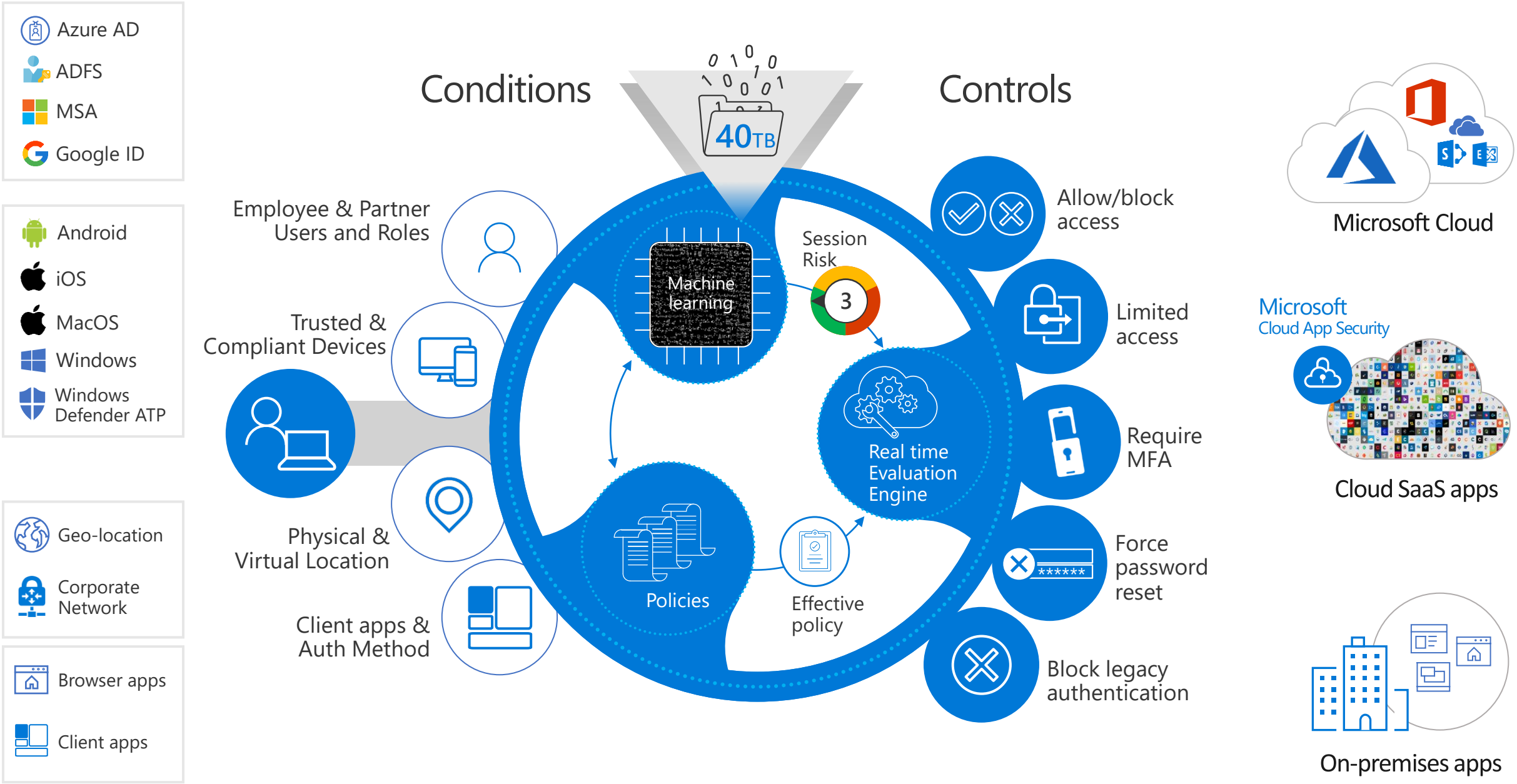
- **Dynamic**: give me URL so I can do “autodiscovery”
- **Session management**: universal session for OIDC (draft)

The difference

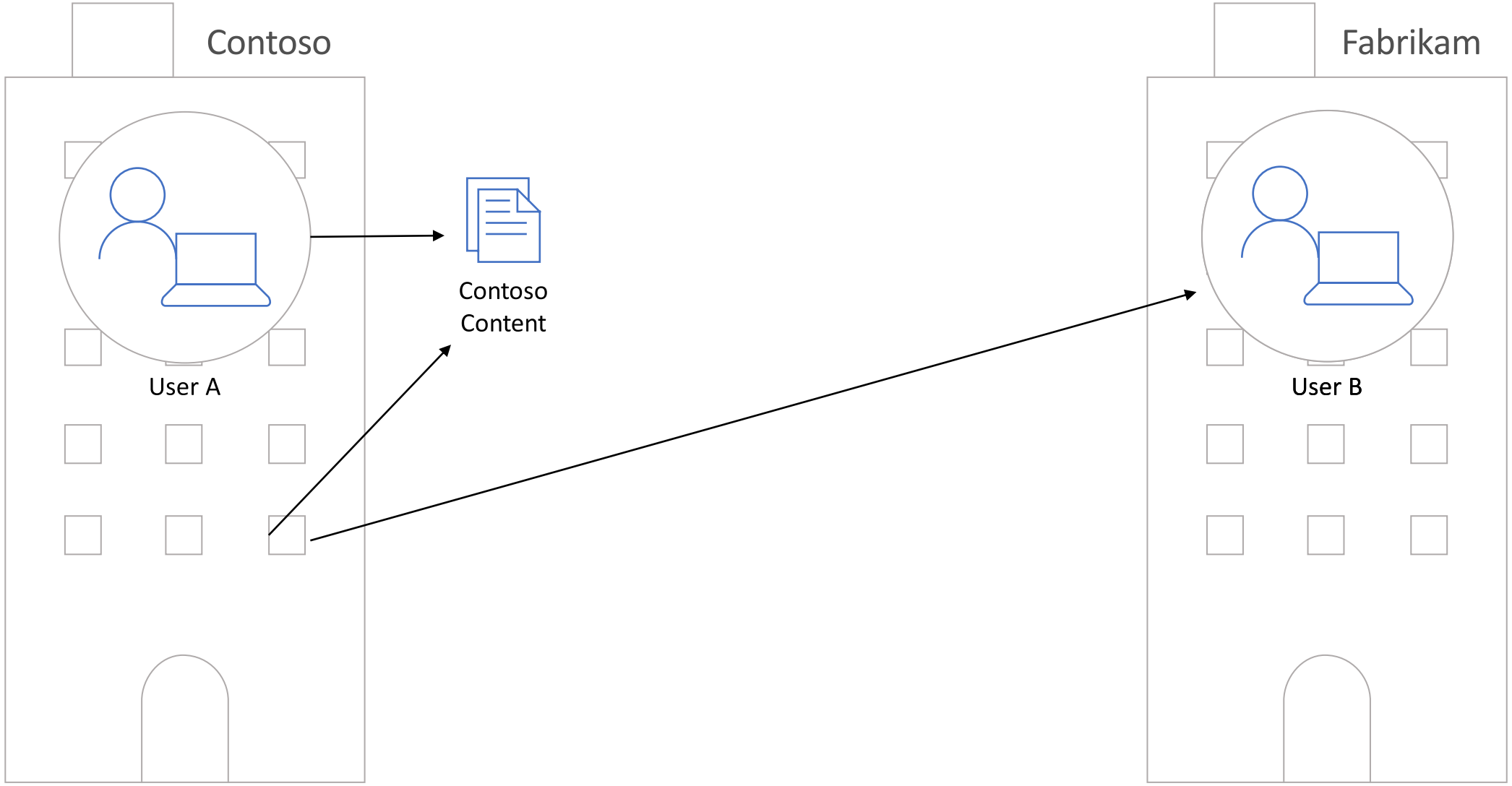
- Use discovery endpoint to dynamically grab URLs within
- Authenticate with OAuth like usual
- You get back a JWT which you validate
- You use that token to call APIs

DEMO

Registering applications and adding authentication to your applications

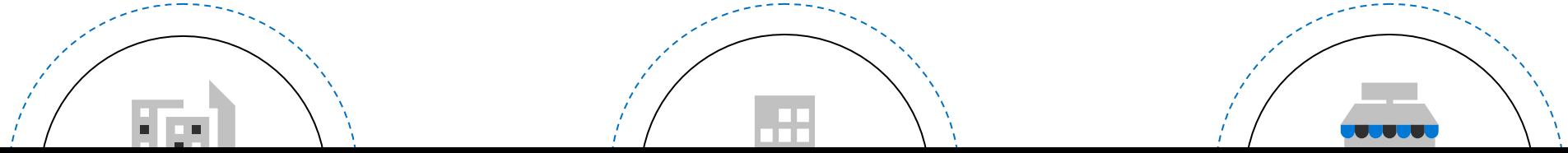

Azure AD
ADFS
MSA
Google ID
Android
iOS
MacOS
Windows
Windows Defender ATP
Geo-location
Corporate Network
Browser apps
Client apps
Conditions
40TB
Controls
Employee & Partner Users and Roles
Trusted & Compliant Devices
Physical & Virtual Location
Client apps & Auth Method
Machine learning
Session Risk
3
Real time Evaluation Engine
Policies
Effective policy
Allow/block access
Limited access
Require MFA
Force password reset
Block legacy authentication
Microsoft Cloud
Microsoft
Cloud App Security
Cloud SaaS apps
On-premises apps

External collaboration using B2B



Azure AD B2B

Enabling work with any partner on the planet



Email one-time passcode authentication (preview)

Azure AD

Direct federation

Google federation

Direct federation

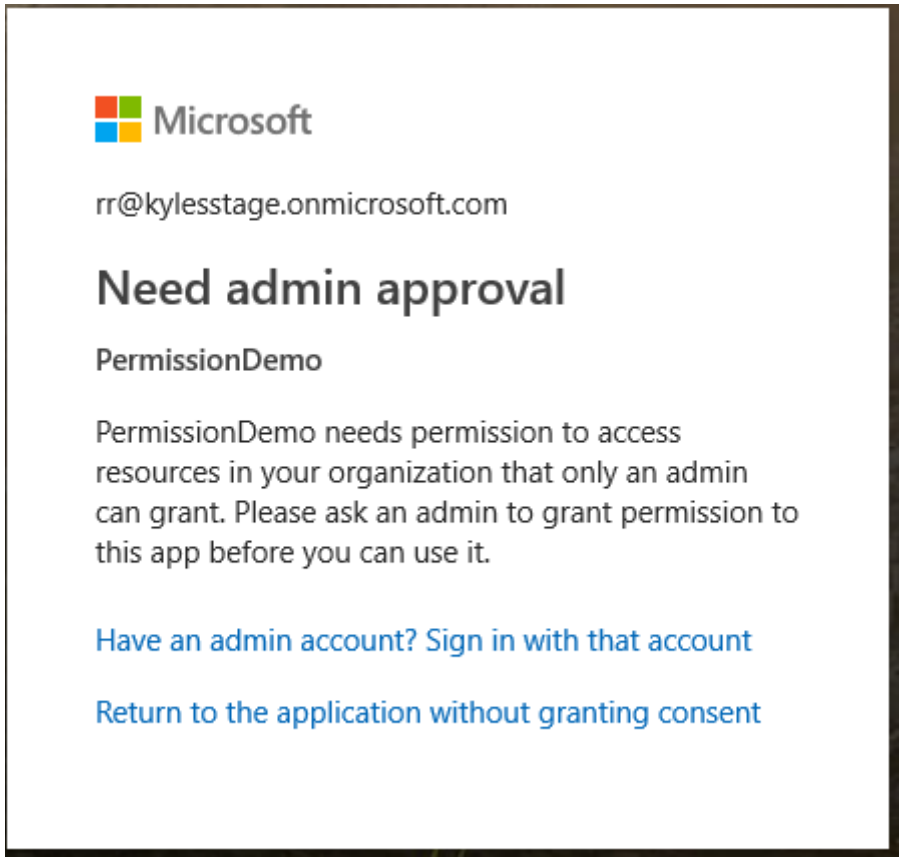
Google federation

Outlook.com (MSA)

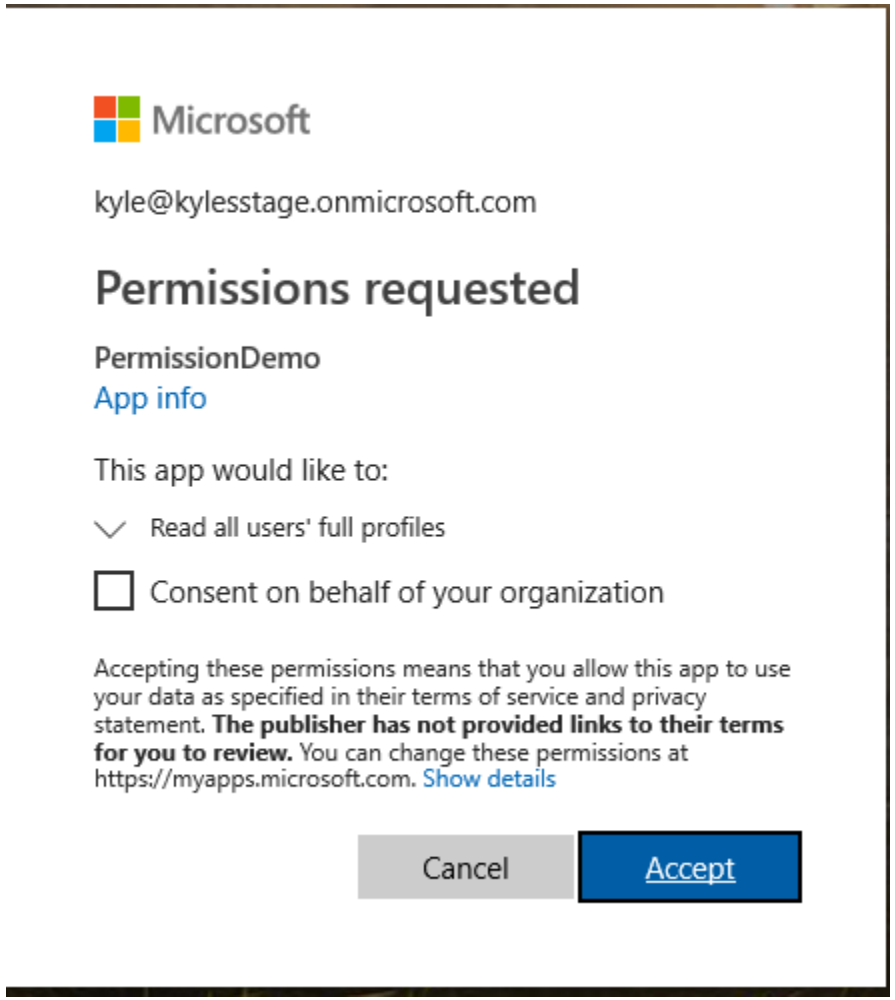
Generally available

Public Preview

Coming soon



Users can not grant consent for a permission that requires Admin consent



Users can not grant consent for a permission that requires Admin consent

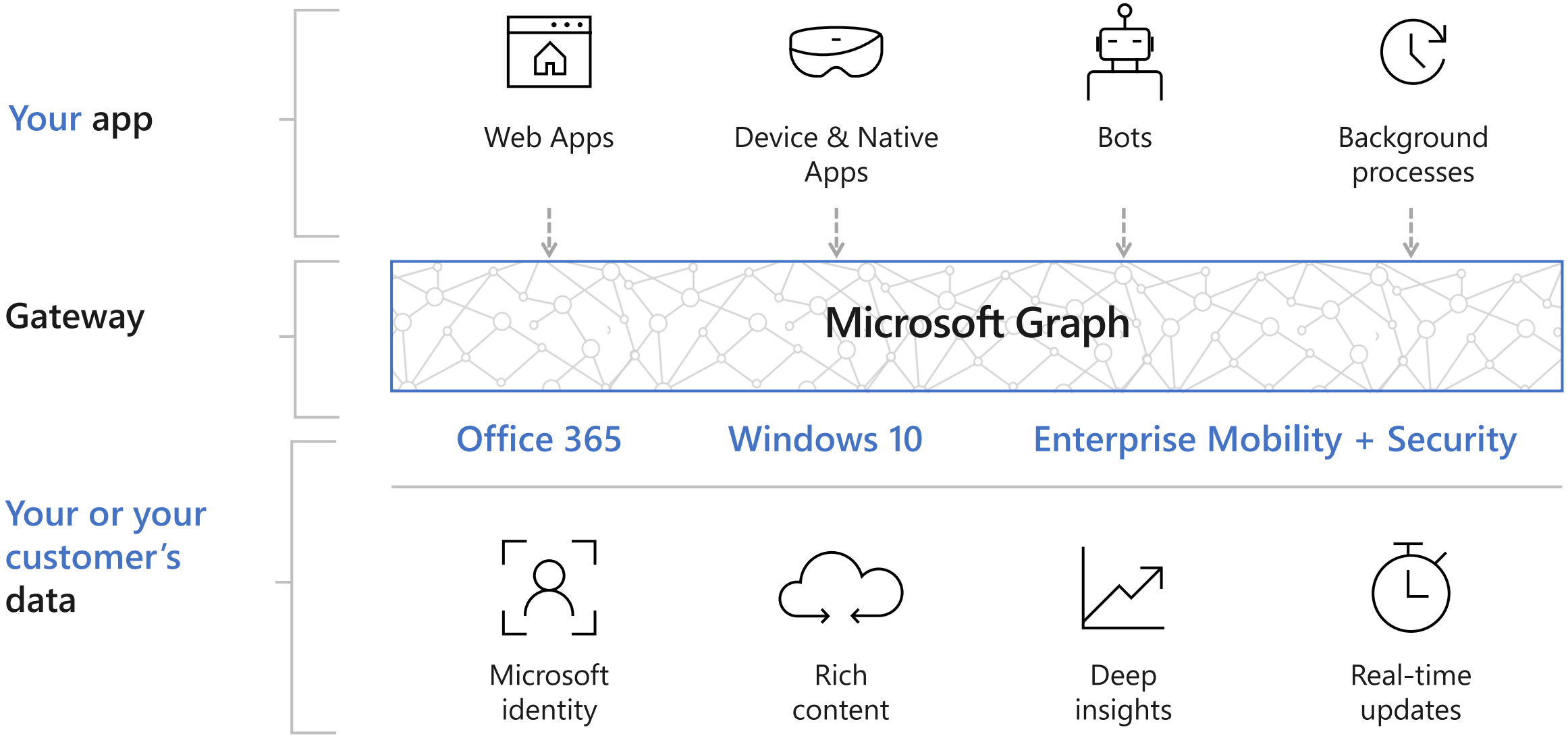
Admins can grant this consent

When running the app

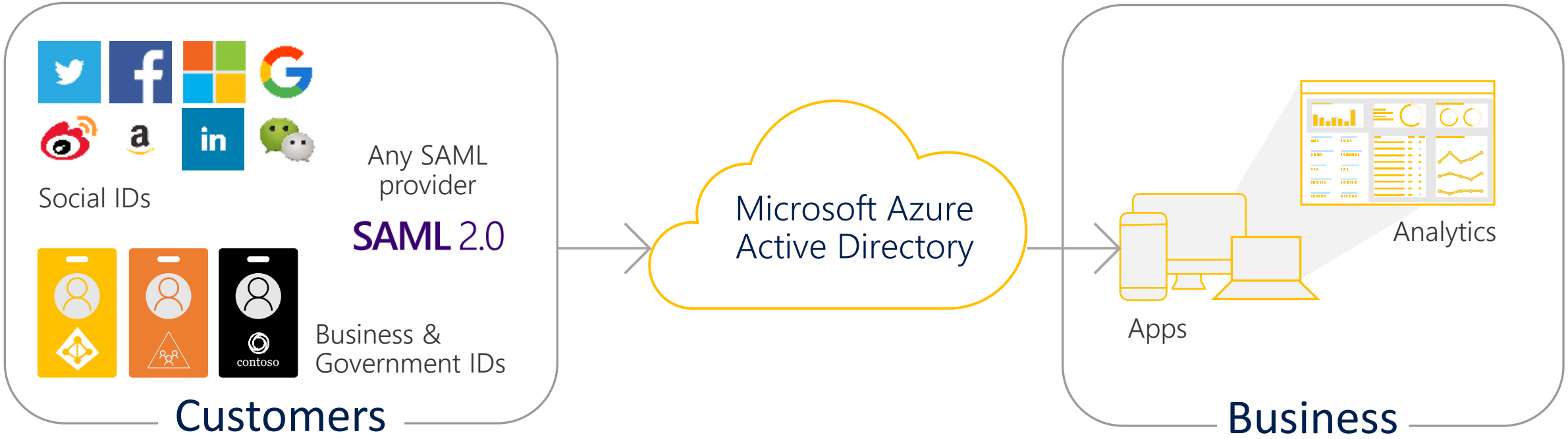
Protecting an API with Azure AD

- Register the API as an app in Azure AD
  - Define the permissions your API exposes
    - Help developers using your API to keep to least-privilege
    - Avoid “do everything” permissions where possible
    - Be conservative with permissions users can consent to
- Validate received access tokens in your API
  - Use existing libraries and middleware. They exist for most platforms.
- Apply and enforce permissions!
  - Delegated permissions must not exceed what the signed-in user is allowed to do.

Gateway to your data in the Microsoft cloud



Azure Active Directory B2C



- Securely authenticate your customers using their preferred identity provider
- Capture login, preference, and conversion data for customers
- Provide branded (white-label) registration and login experiences

Managed Identities for Azure resources

Remove credentials from your code…

Why is this important?

Sign in Get started

Sign in Get started

Setting up the token acquisition is fairly painless:

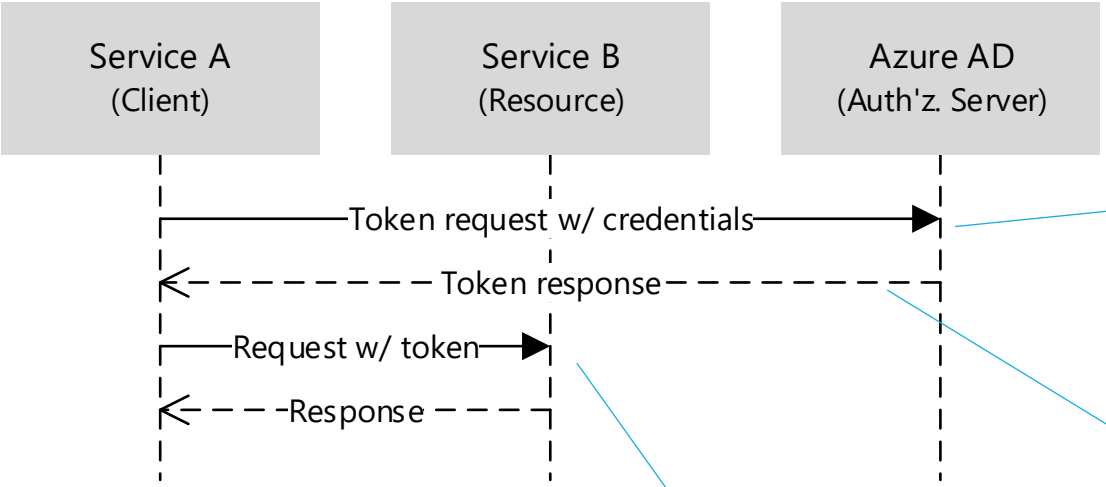
122

```
var authenticationContext = new AuthenticationContext(authString,
false);

// Config for OAuth client credentials
ClientCredential clientCred = new ClientCredential(clientId,
clientSecret);
AuthenticationResult authenticationResult = await
authenticationContext.AcquireTokenAsync(resourceId,clientCred);
string token = authenticationResult.AccessToken;
log.Verbose(token);
```

This follows the *Client Credential* flow for OAuth, and this is a silent flow for the user. Everything happens behind the scenes on the back-end. You should never use this in a client app as the clientSecret is not something you want the end-user to have access to.

Review: OAuth 2.0 Client Credentials Grant flow



Client must authenticate with the Authorization Server (AS).

No refresh token. If access token expires, just get a new one.

The access token (proof that client is authorized) is provided to the Resource Server (RS). RS must be able to verify that trusted AS issued the token.

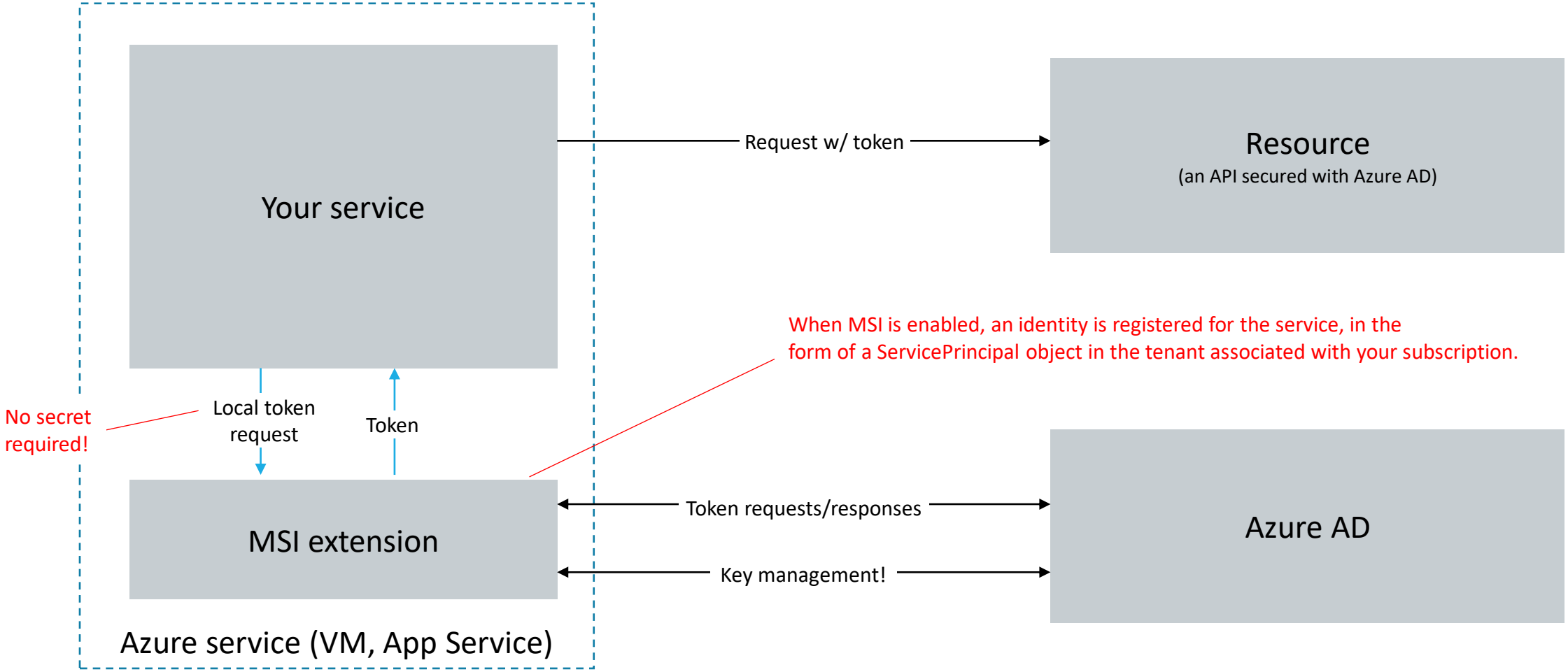
Managed identities for Azure resources

Formerly known as “Managed Service Identities”

- Gives your Azure *service* an identity
- Available for many Azure resource types (and more coming)
- System-assigned vs. user-assigned
  - System assigned identities, tied to a given resource only
  - User assigned identities, can be assigned to different resources

- No secrets in code!

Managed identities for Azure resources



DEMO

Managed Identities for Azure resources

- Use Single Sign On wherever possible
- Keep credentials AWAY from your code
- Leverage Microsoft APIs to get rich information about the organizations
- Protect your APIs with proper authorization – authentication is not always enough

- Managed Identity Demos