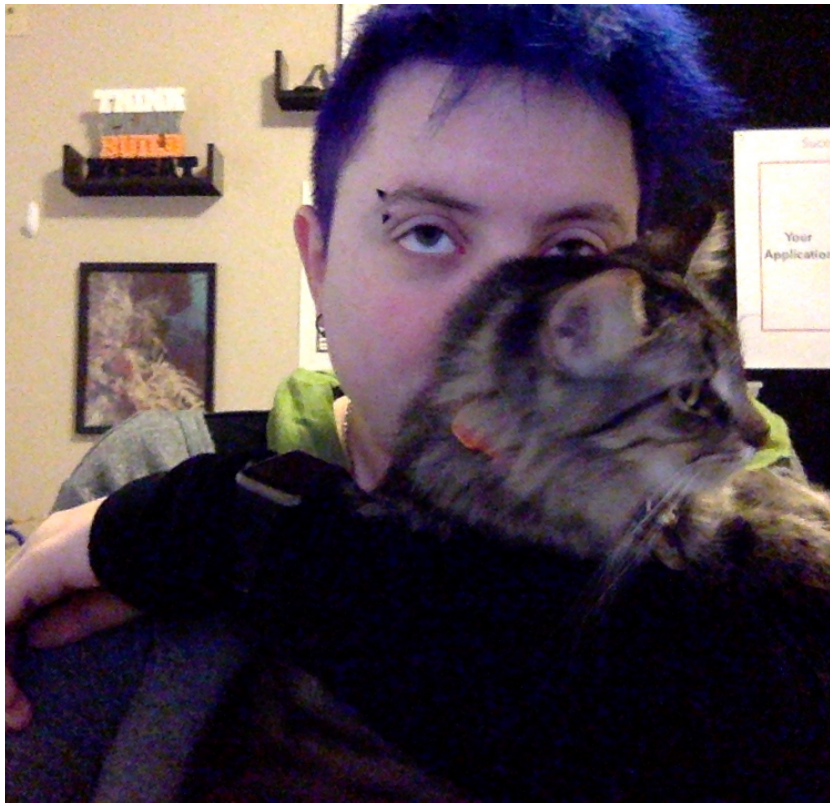
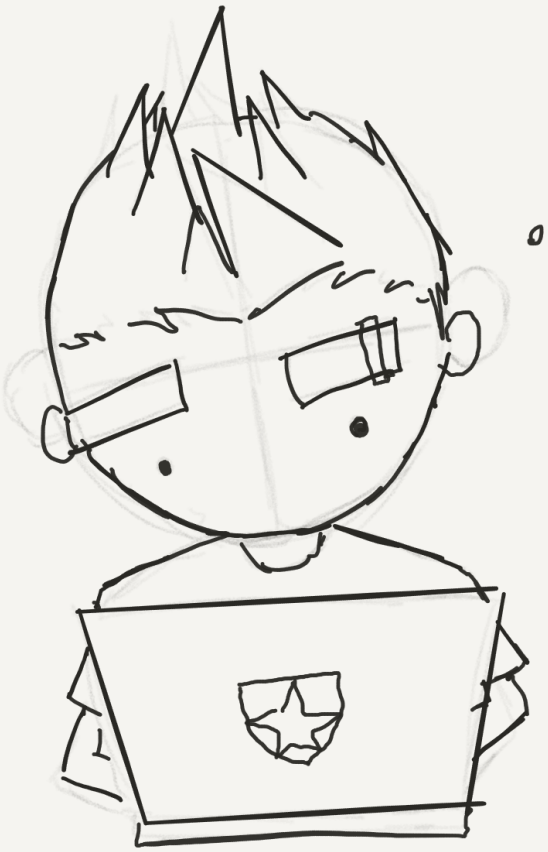
OAuth, OIDC, and .NET Core

Who *are* you, anyways?

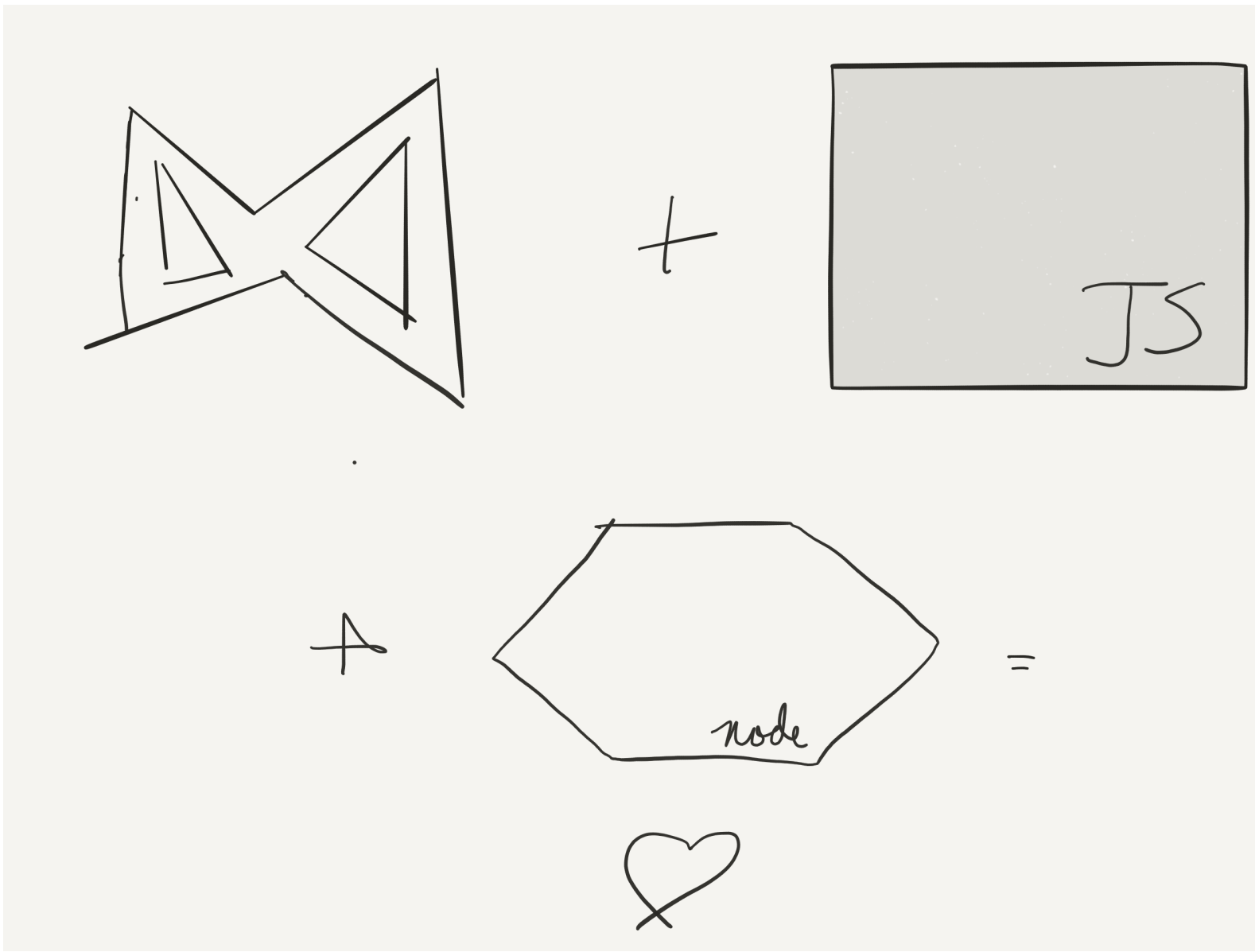
- Developer Evangelist for Auth0
- Electrical Engineering student
- Catparent
- Genderqueer (sure, go ahead, ask me about it!)



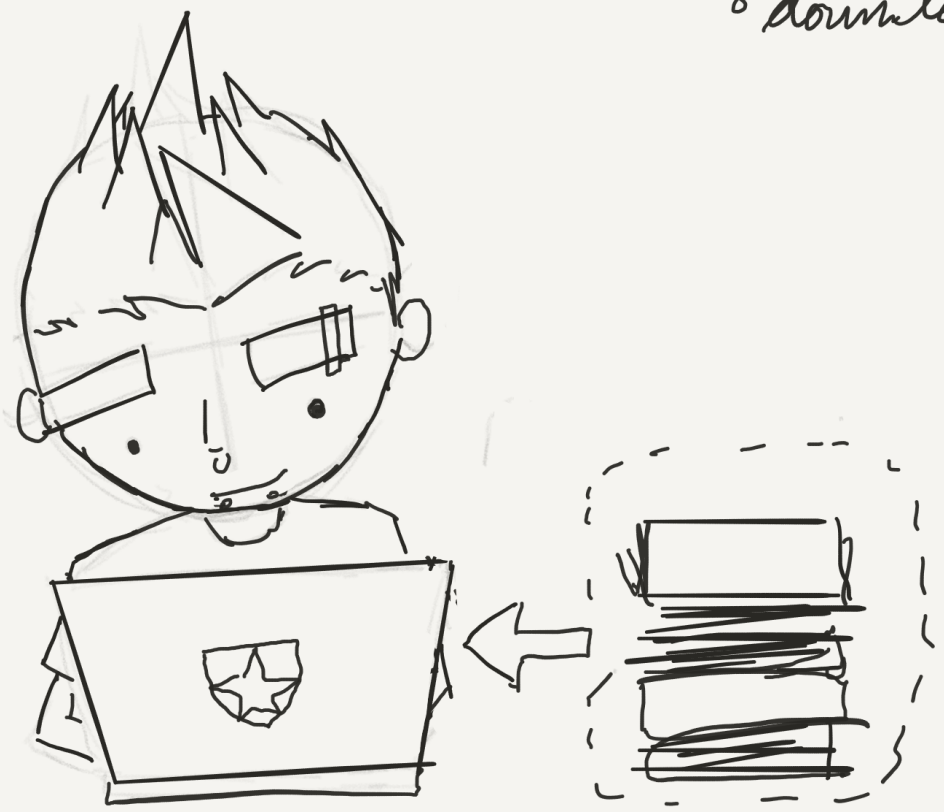
How I ended up at .NET Fringe

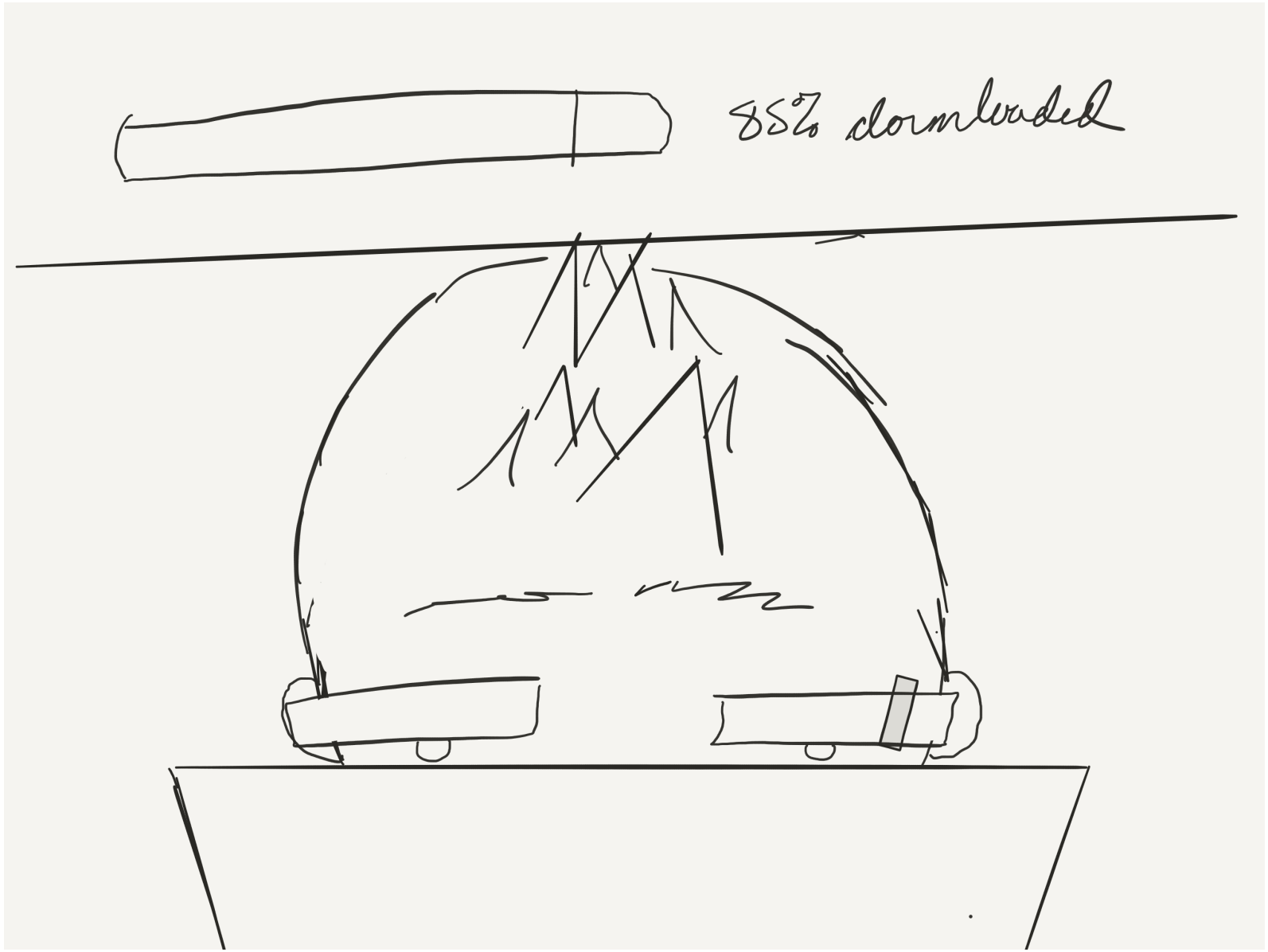

you're speaking at a .NET
conference in 10 days!
eeeee

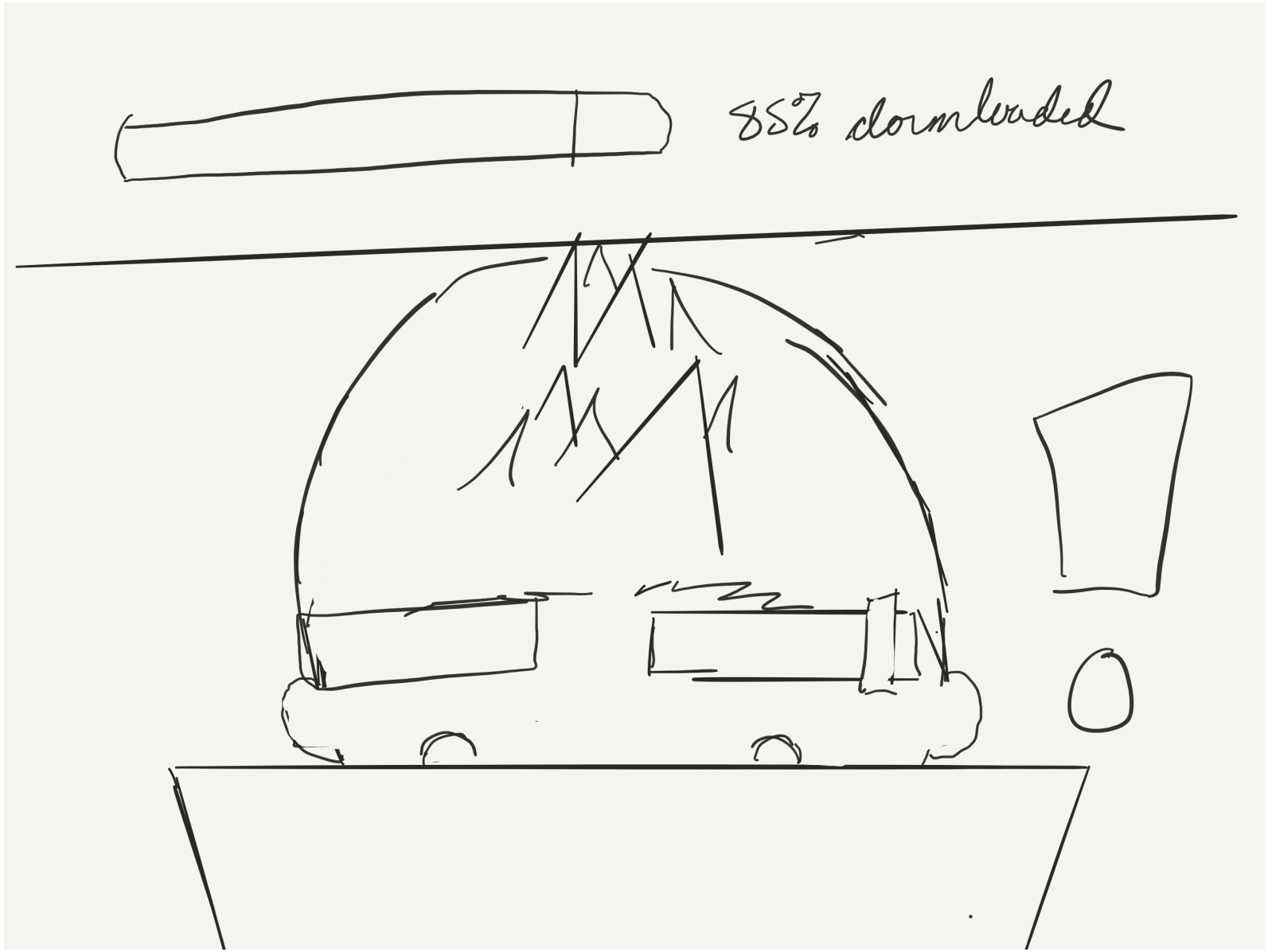
one of the things I
love about this job!


JS
node

- .NET refresh
- whats Core again
- download VM to code on




85% downloaded


85% downloaded

NO

VM

required!

Wait…so why should we listen to you on this?

I have read in full…

- The OAuth 2.0 Specification (RFC6749, 6750)
- OpenID Connect Core 1.0 incorporating errata set 1 specification
- OpenID Connect Discovery 1.0 incorporating errata set 1 specification
- OpenID Connect Dynamic Client Registration 1.0 incorporating errata set 1

(Also, I was a .NET developer in a previous life. My first conference talk was at a .NET conference!)

I hear this a LOT:
“OAuth is terrible”

Two things contribute to this

1. A lack of what OAuth is supposed to do
2. Missing context on the history of OAuth
3. Some misunderstandings of OAuth 1 vs 2, specifications vs. implementations

History of OAuth

Ok, so there's OAuth 1 and 2

- Most OAuth providers have moved to 2 (or at least support it)
- OAuth2 accounts for more than "computer + browser + server"
- We did lose signed signatures…KIND of.

Bearer Tokens

- Added in OAuth 2
- If you have the token, you're authenticated
- Made a lot of people wary
  - Where's the verification that the token came from a trusted source?

JSON Web Tokens (JWTs)

- Mitigate the lack of signatures by…well…being signed!
- Your bearer token can be a signed JWT, which consumers would then verify, and you can also required signed JWTs from them!

Wait, wait, JWT?

- Header + payload + key
- Header and payload are JSON objects
- Key is header + ‘.’ + payload, signed with a secret

Wanna learn more about JWTs?

- https://jwt.io
- RFC 7519

Okay, back to OAuth

- You care about OAuth because managing usernames and passwords is a giant pain
  - Logins are universal, but why implement it yourself?
  - How do I keep passwords safe?
  - Will users care enough to create a new (totally secure because of course it is) password?

Probably Not. But they'll log into Twitter.

The Problem with Implementations

- Some OAuth platforms ONLY use 1.0
- Some use both! Because that's not confusing AT ALL.
- Some people expect every OAuth implementation to return the same user identity.

Before we go further, there's some vocab to clear up here.

Identity Vocabulary

- Authentication: proving a user is who they sy they are
- Authorization: allowing a user to complete an action
- Identity: the information we know about a user

The “problems” with OAuth

- Just does Authentication
  - …which is what it was designed to do.
- It’s so hard to use!
  - No, not really. There are SDKs and core libraries all over the place for both .NET and JavaScript.
  - You just heard that from someone and agreed with them to sound smart (I know I did!)

The Real Problem

- We shoehorned OAuth to be an identity solver.
- Which, as it always on the web, led to 50 different "specifications" (read: because-i-said-sos) of what User Identity is and means

This is where OpenID Connect comes in.

OpenID Connect

- Started as OpenID Authentication
- Became an ID wrapper around OAuth2
- Standardized way of defining ID for users logging in with OAuth

Why was this a problem?

- …is it username, userName, or name for Twitter again?
- I want to add Facebook, but I have to map everything to fit my current models
- I need this piece of user info, but I have no idea how to get it!

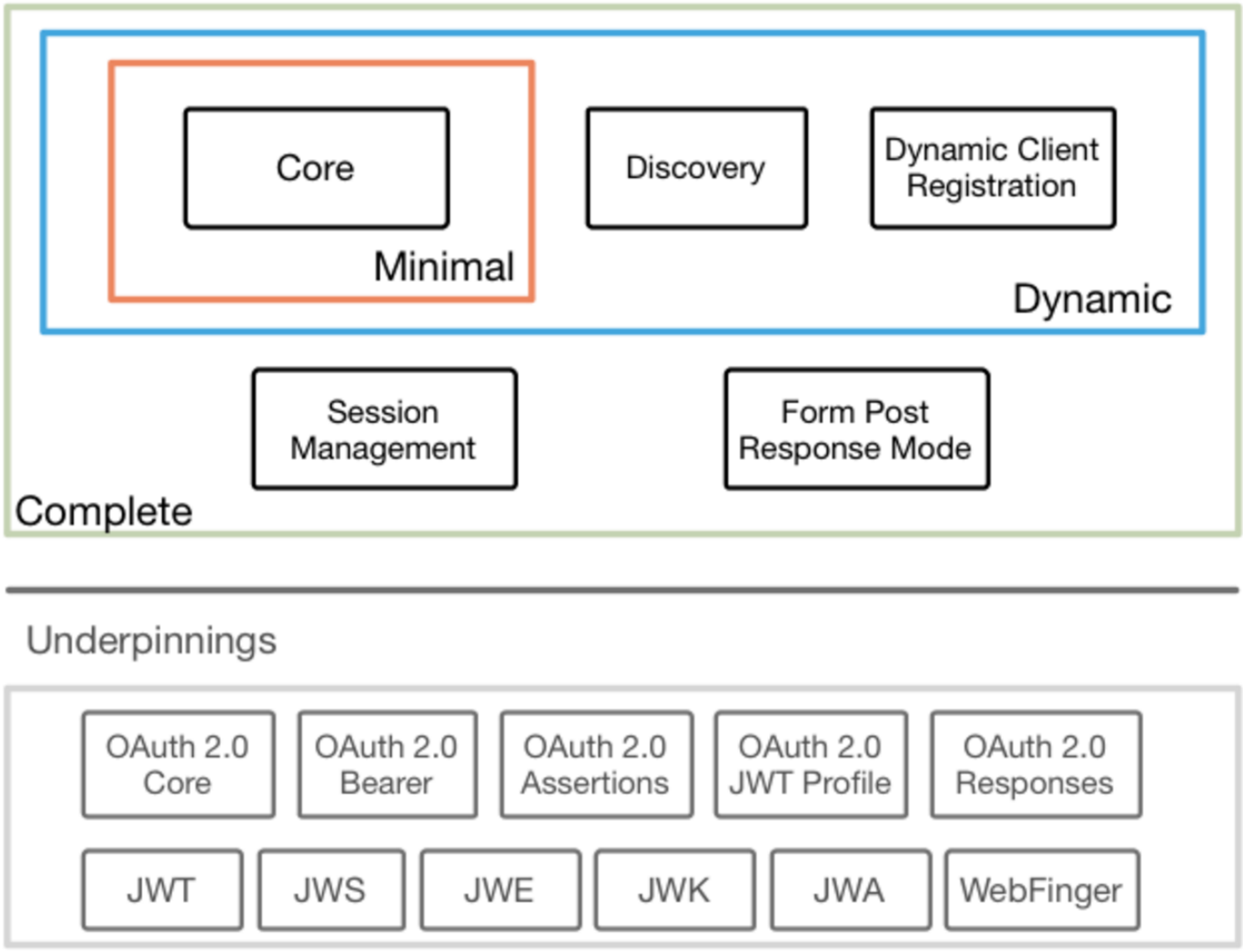
Right, but what IS it?

- “A simple Identity layer built on top of OAuth 2.0” — openid.net
- Okay but what does THAT mean?

4 Feb 2014

OpenID Connect Protocol Suite

http://openid.net/connect



…that doesn’t look simple.

But it does make things simpler!

- Dynamic: give me a URL that I can use to figure out where everything is
- Dynamic client registration: let me programmatically obtain a client ID and secret for OAuth
- Session management specs mean if you're implementing OIDC, you can share sessions between your apps!

Okay, okay. So what's different?

- Instead of pre-programming URLs, you grab a discovery document and use the URLs within
- You authenticate with OAuth like usual
- You get back a JWT ID token, which you can validate the signature of
- You use this token to obtain user information from the OAuth provider (like Google!)

Okay, so you're saying…

- You use OAuth 2.0 to Authenticate Users…
- And OpenID Connect to get information about their identity…
- Which you use in your app to authorize them to do things!

.NET Core, OAuth, and OIDC

Middleware Everywhere

- AspNet.Security.OAuth
  - (One of my coworkers contributes to this!)
- Microsoft.AspNetCore.Authentication.OAuth
  - Has a few providers, works pretty similarly
- Microsoft.AspNetCore.Authentication.OpenIdConnect
  - A little confusing, assumes discovery level

But let’s look at code instead of talking about code.

Learn you some authentication and identity for great good.

- OpenID.net
- OAuth 2 SDKs for C#, OIDC SDKs for C#
- jwt.io

Thanks for listening!

- kas@auth0.com
- @nodebotanist just about everywhere (yes, even Pokémon Go)
- https://auth0.com