XYZ

@justin__richer

https://bspk.io/

This is not an extension to OAuth 2

Why not?

- JSON
- Flexibility
- Consistency
- Reification of the transaction itself
- We have learned a lot about what works and what doesn't in the last decade

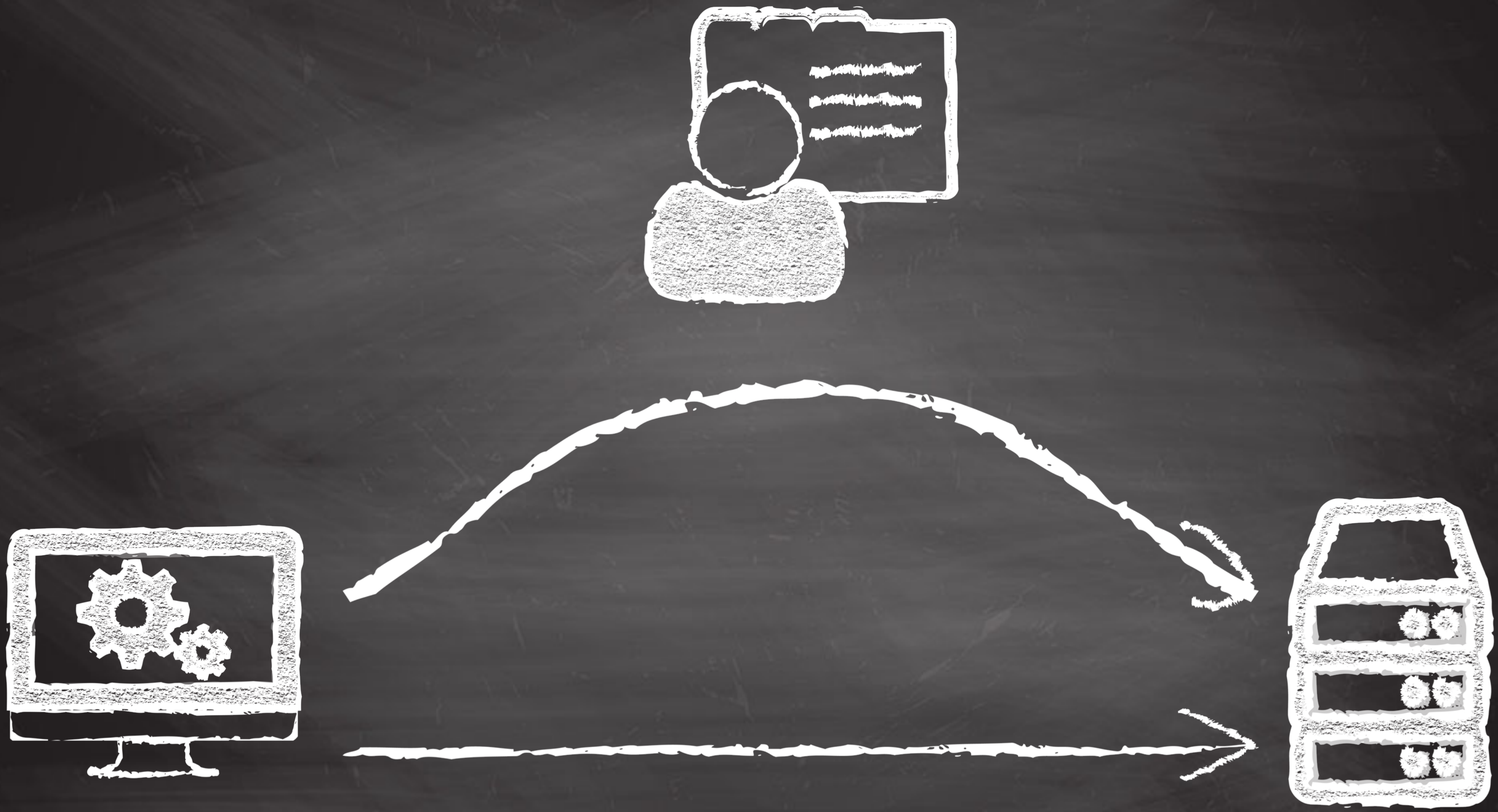
Status

- Implementations in Java and Node.js
  - Redirect client
  - User code client
  - AS (transaction, interaction, and user code)
  - Signing methods
- Details at https://oauth.xyz/

Transactions

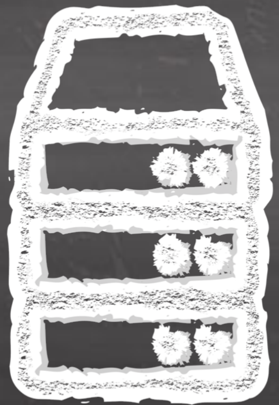
@justin__richer https://bspk.io/

OAuth has always been transactional



Registering Intent

The client starts at the AS



Start a Transaction

```
{
    "resources": [ ... ]
    "key": ...
    "display": ...
    "interact": ...
    "user": ...
}
```

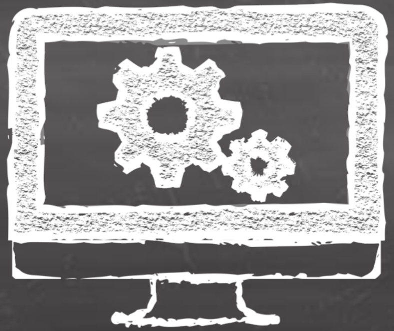
"What I want"

```
"resources": [{
    "actions": ["read", "write", "dolphin"],
    "locations": ["https://server.example.net/",
                  "https://resource.local/other"],
    "datatypes": ["metadata"]
},
...
]
```

"How to recognize me"

```
"key": {
    "proof": "jwsd",
    "jwks": {
      "keys": [ {
        "kty": "RSA",
        "e": "AQAB",
        "kid": "xyz-1",
        "alg": "RS256",
        "n": "kOB5rR4Jv0GMeLaY6_It_..."
      } ]
    }
}
```

The client has to prove possession of all referenced keys



Sign the request body and present a header

```
Detached-JWS: eyJiNjQiOmZhbHNlLCJhbGciOiJSU...
```

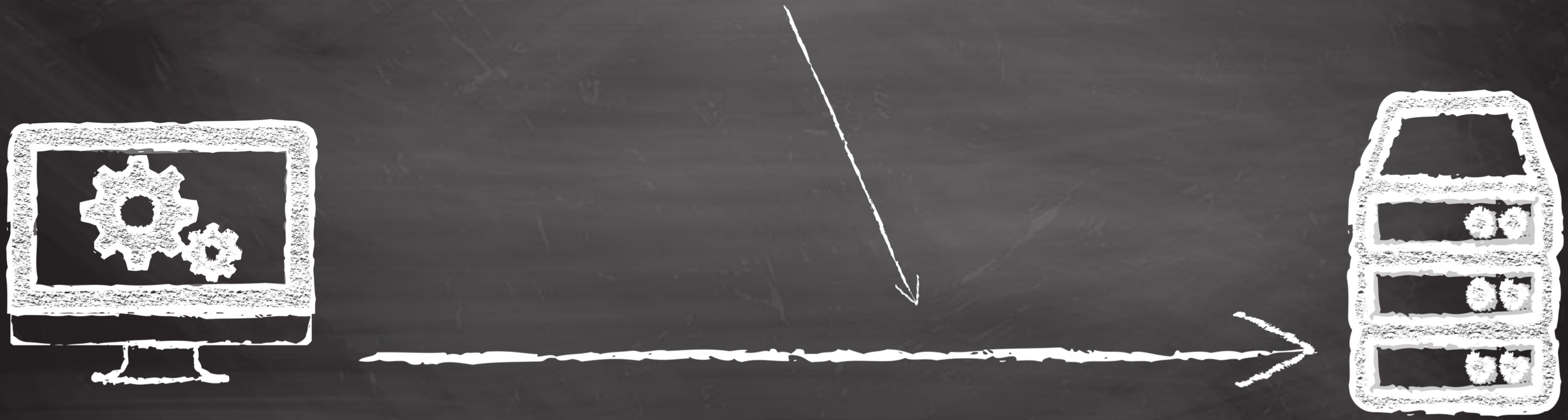
Use DPoP key proofing

```
DPoP: eyJiNjQiOmZhbHNlLCJhbGciOiJSU...
```

Use Cavage signatures

```
Signature: keyId="xyz-client", algorithm="rsa-sha256",
    headers="(request-target) digest content-length",
    signature="TkehmgK7GD/...
Digest: SHA=oZz2O3kg5SEFAhmr0xEBbc4jEfo=
```

Use MTLS



“What to show the user”

```
"display": {
    "name": "My Client Display Name",
    "uri": "https://example.net/client"
}
```

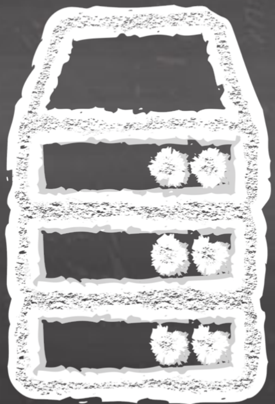
"What I know about the user"

```
"user": {
    "assertion": "eyJraWQiOiIxZTlnZGs3IiwiYWxnIjoi..."
    "type": "oidc_id_token"
}
```

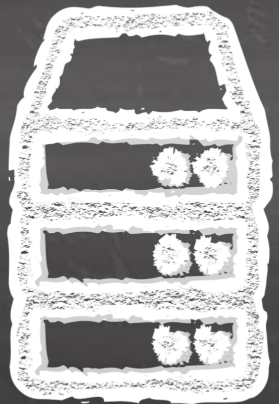
"How I can interact with the user"

```
"interact": {
    "redirect": true,
    "user_code": true,
    "callback": {
        "uri": "https://client.example.net/return/123455",
        "nonce": "LKLTI25DK82FX4T4QFZC"
    }
}
```

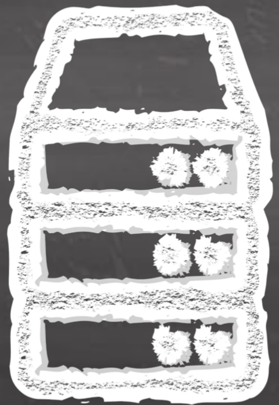
Process all aspects of the transaction request



Maybe we can already issue an access token



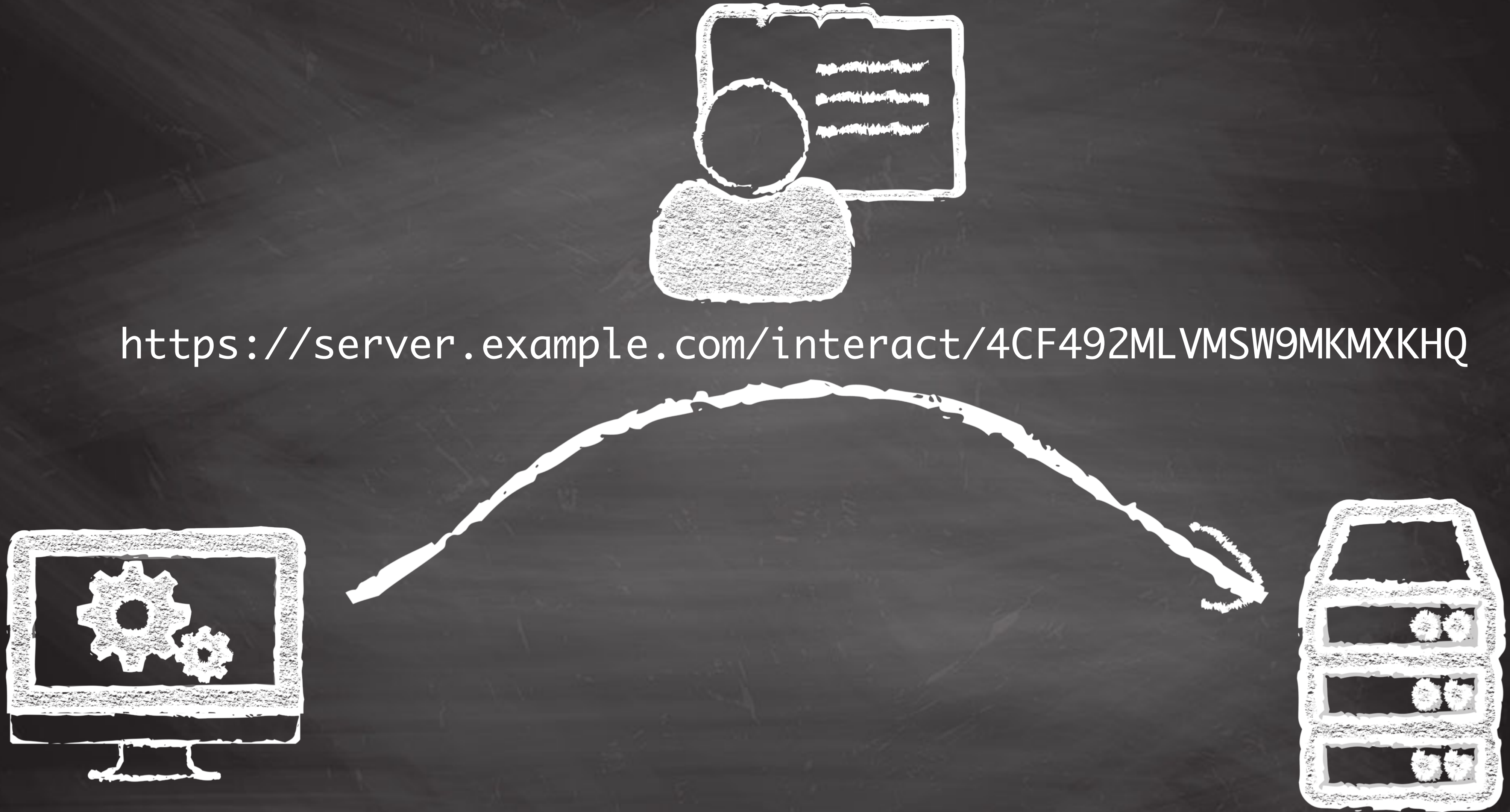
Or:
"I need to talk to the user"



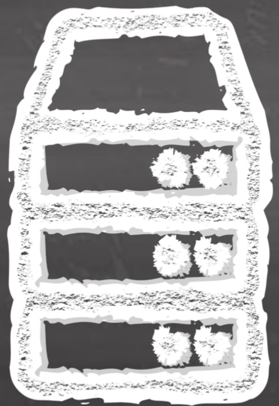
"Go fetch me the user"

```
{
    "interaction_url":
"https://server.example.com/interact/4CF492MLVMSW9MKMXKHQ",
    "server_nonce": "MBDOFXG4Y5CVJCX821LH",
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    }
}
```

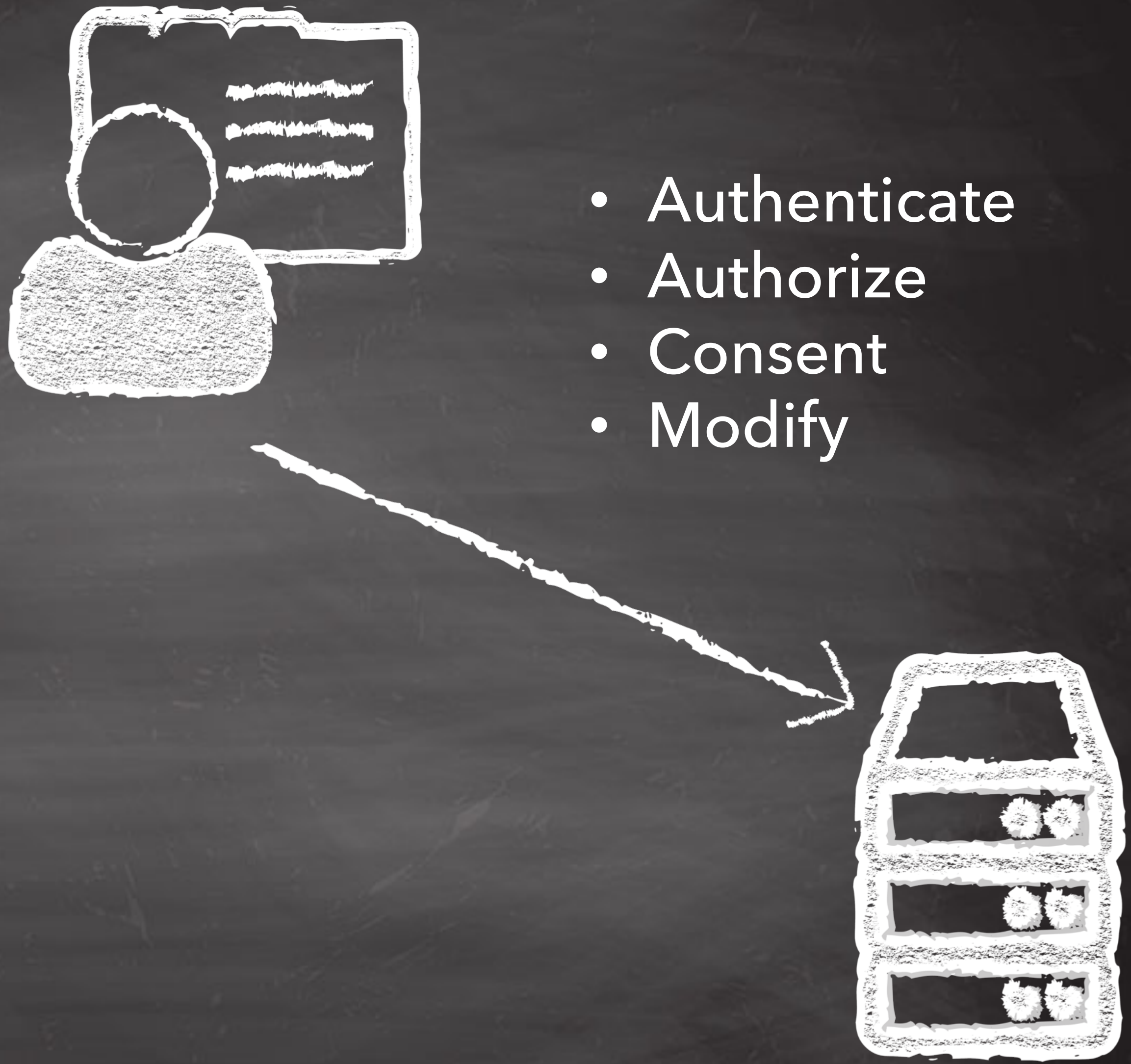
The Front Channel



Look up the transaction based on the incoming interaction URL

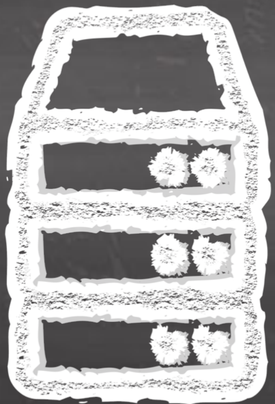


User interacts like you'd expect

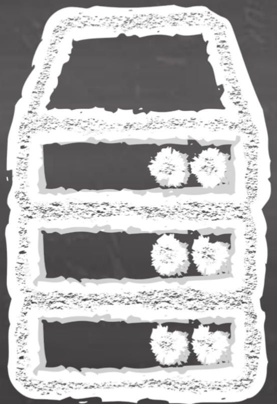


- Authenticate
- Authorize
- Consent
- Modify

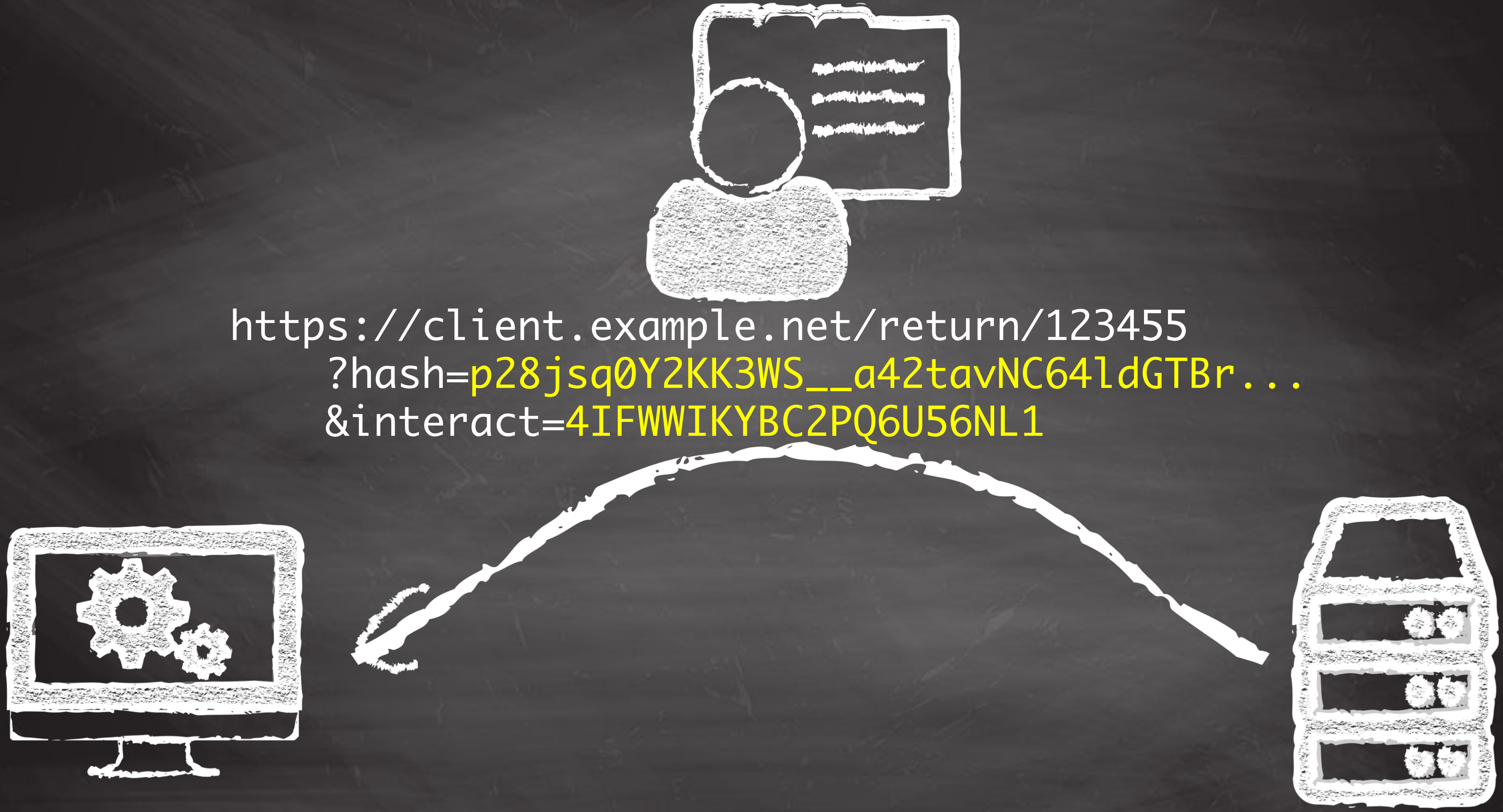
Generate a handle



Calculate a hash

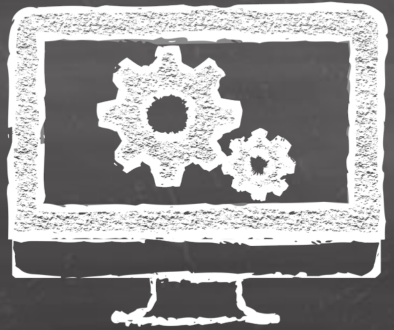


```
client_nonce
server_nonce
interact_handle
```



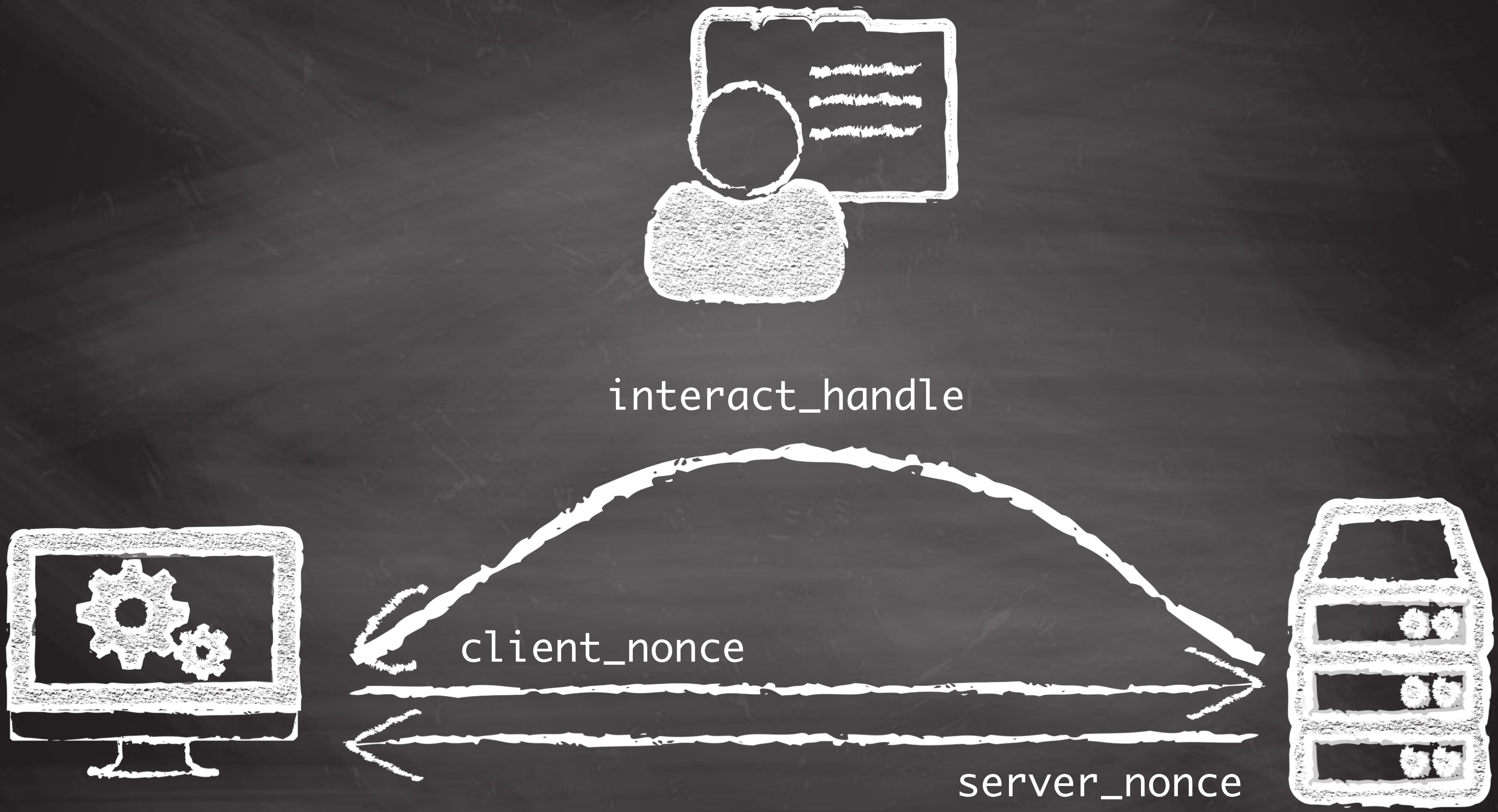
```
https://client.example.net/return/123455
    ?hash=p28jsq0Y2KK3WS__a42tavNC64ldGTBr...
    &interact=4IFWWIKYBC2PQ6U56NL1
```

Validate the hash



```
client_nonce
server_nonce
interact_handle
```

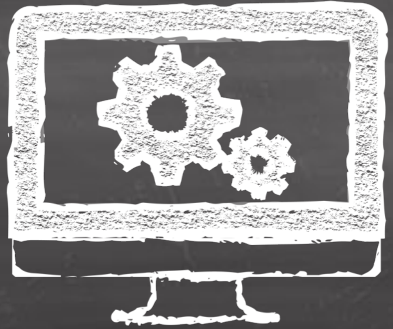
Connect the legs of the triangle



Continue the Transaction

```
{
    "handle": "80UPRY5NM33OMUKMKSKU",
    "interact_handle": "4IFWWIKYBC2PQ6U56NL1"
}
```

The client STILL has to prove possession of all referenced keys



"Here's an access token"

```
{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "type": "bearer"
    }
}
```

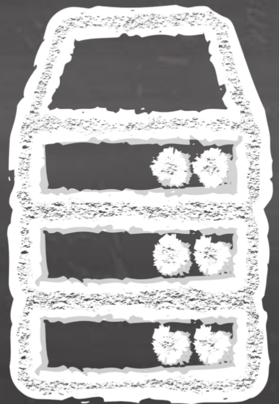
Handles:

Referencing previous state

"Use this, I'll remember you"

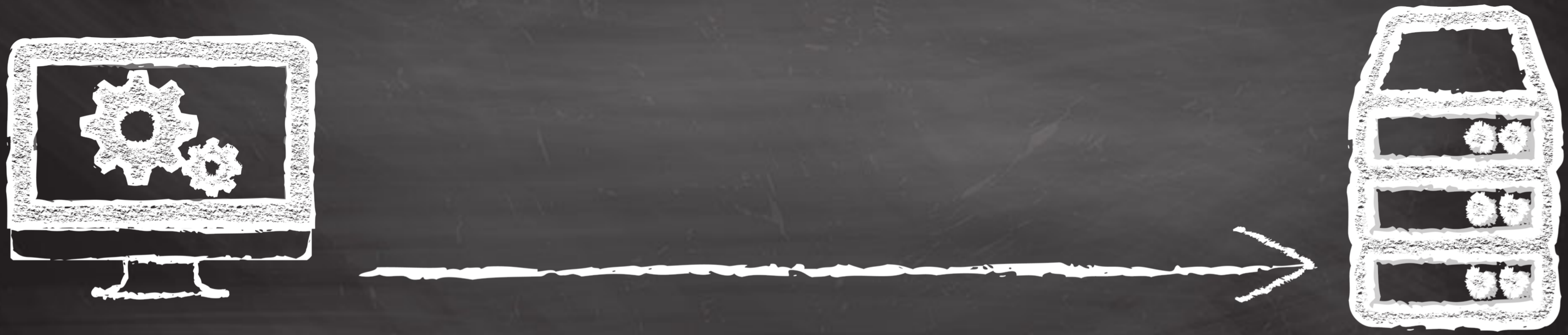
```
{
    "display_handle": {
        "value": "VBUEOIQA82PBY2ZDJW7Q", "type": "bearer"
    },
    "key_handle": {
        "value": "7C7C4AZ9KHRS6X63AJA0", "type": "bearer"
    }
    ...
}
```

This could happen out of band

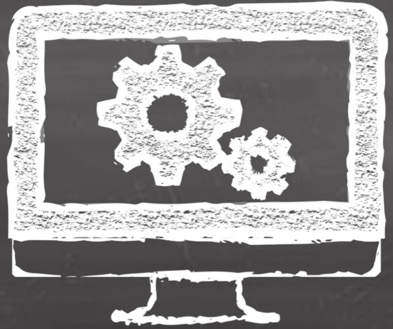


Starting a new transaction with handles

```
{
    "display": "VBUEOIQA82PBY2ZDJW7Q",
    "key": "7C7C4AZ9KHRS6X63AJA0"
}
```



The client STILL has to prove possession of all referenced keys

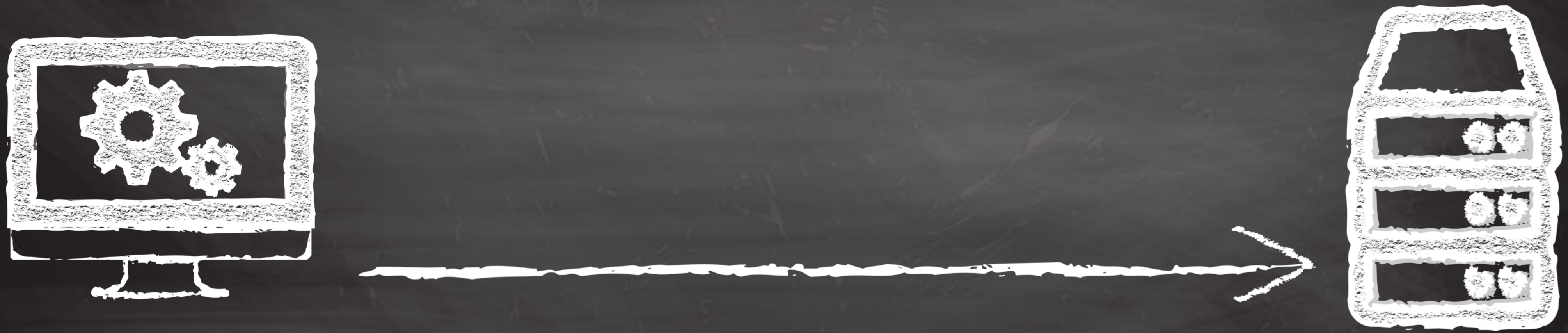


An access token and a transaction handle

```
{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "type": "bearer”
    },
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    }
}
```

Refreshing a Token

```
{
    "handle": "80UPRY5NM33OMUKMKSKU"
}
```



Remembering or identifying the user

```
{
    "user_handle": {
        "value": "XUT2MFM1XBIKJKSDU8QM",
        "type": "bearer"
    }
}
```

Resource handles: Scopes, redux

```
"resources": [
    "read", "write", "dolphin"
]
```

Complex combined requests

```
"resources": [
    "read", "write", "dolphin",
    {
        "actions": ["read", "write", "dolphin"],
        "locations": ["https://server.example.net/",
                      "https://resource.local/other"],
        "datatypes": ["metadata"]
    }
]
```

Other interaction modes

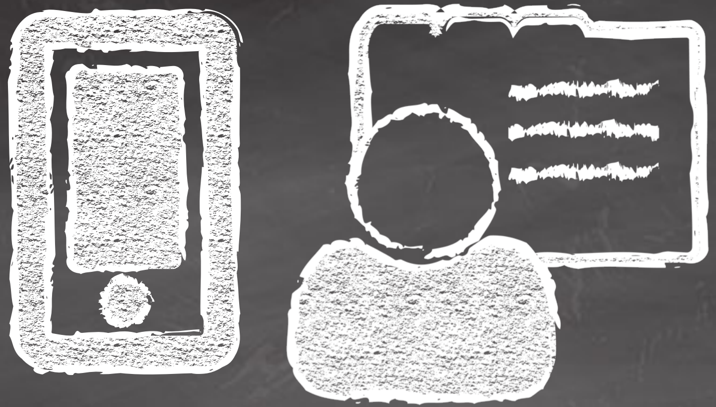
User code style interaction

```
"interact": {
    "redirect": true,
    "user_code": true
}
```

"Go fetch me the user"

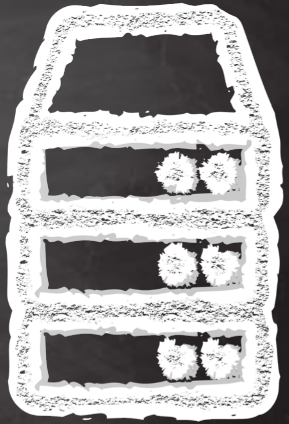
```
{
    "user_code": {
        "url": "https://server.example.com/interact/device",
        "code": "A1BC-3DFF”
    }
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    }
}
```

Tell the user

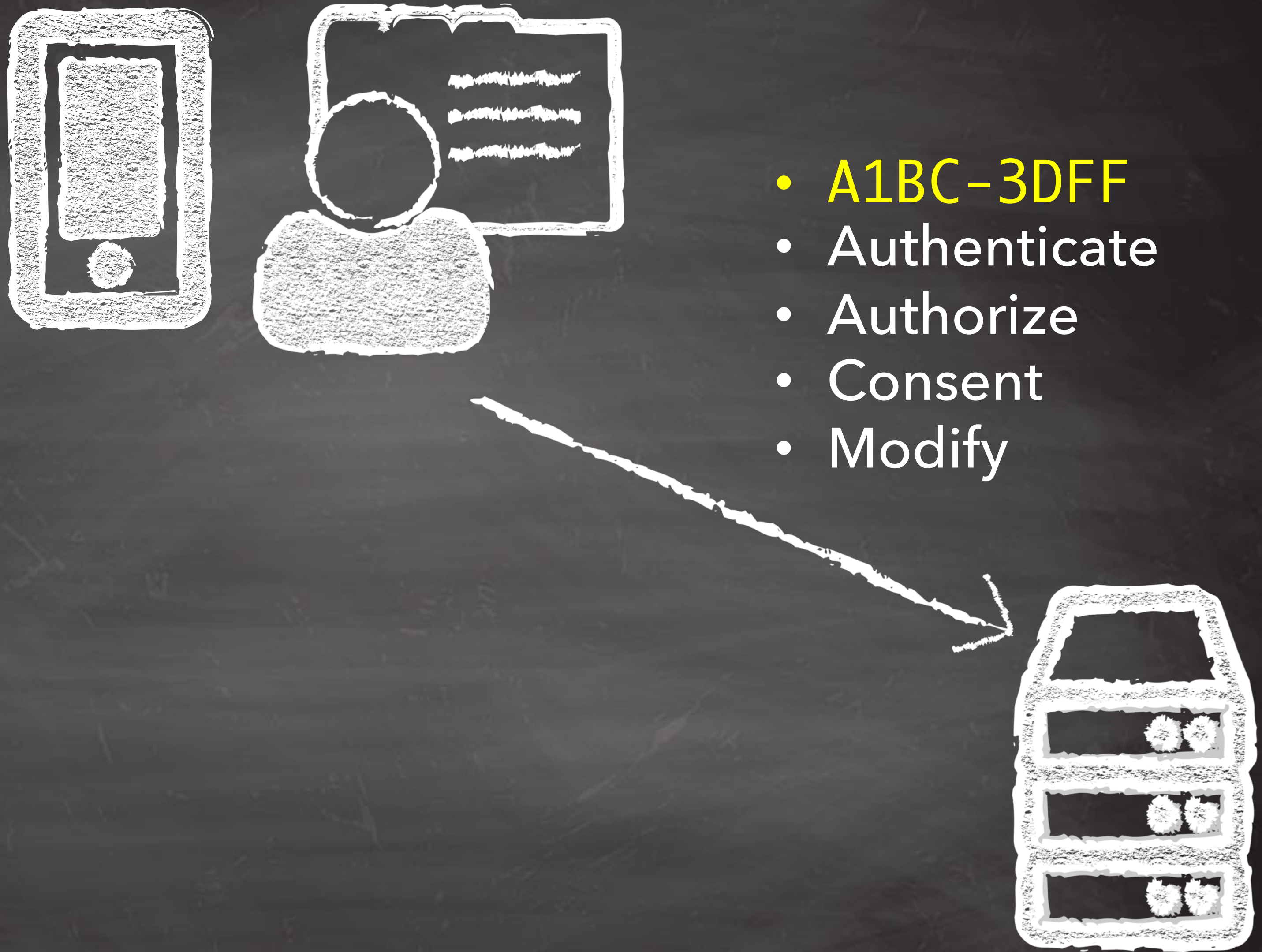


https://server.example.com/interact/device

A1BC-3DFF

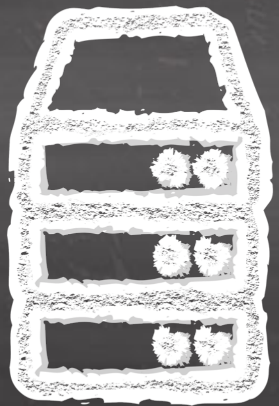


User interacts like you'd expect



- A1BC-3DFF
- Authenticate
- Authorize
- Consent
- Modify

Look up the transaction based on the user code



Meanwhile: Are we ready yet?

```
{
    "handle": "80UPRY5NM33OMUKMKSKU"
}
```

Not yet

```
{
    "wait": 30,
    "handle": {
        "value": "BI9QNW6V9W3XFJK4R02D",
        "type": "bearer"
    }
}
```

What about a combined URL?

We can use the regular interaction URL

```
{
    "interaction_url":

"https://server.example.com/interact/4CF492MLVMSW9MKMXKHQ",

    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    }
}
```

Tell the user

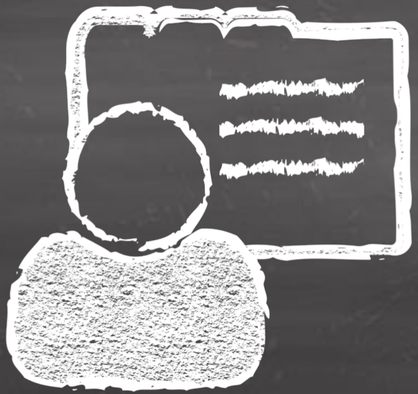
"Get me stuff from the user's agent"

```
{
    "didcomm": "...",
    "handle": {
        "value": "80UPRY5NM33OMUKMKSKU",
        "type": "bearer"
    }
}
```

"Prove that the user has an authenticator"

```
{
    "webauthn": {
        "origin": ...
        "challenge": ...
    },
    "handle": {
       "value": "80UPRY5NM33OMUKMKSKU",
       "type": "bearer"
    }
}
```

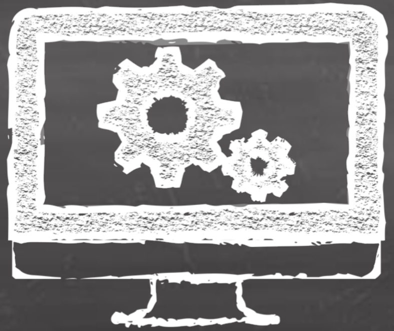
What about identity?



Pass identity assertions back like OIDC, VC

```
{
    "access_token": ...
    "id_token": "eyj0...”
    "user_info": {
        "sub": "BA293-123AAZ",
        "profile_uri": "http://..."
    }
    "verifiable_claims": "..."
}
```

What about binding tokens?



Access token bound to a key

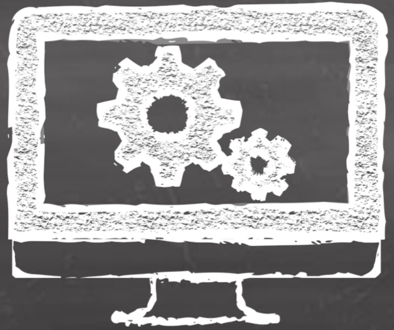
```
{
    "access_token": {
        "value": "OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
        "type": "jwsd",
        "key": {
            "kid": "token-1234", ...
        }
    }
}
```

Key proof is presented alongside token

```
Authorization: JWSD OS9M2PMHKUR64TB8N6BW7OZB8CDFONP219RP1LT0
Detached-JWS: eyJiNjQiOmZhbHNlLCJhbGciOiJSU...
```

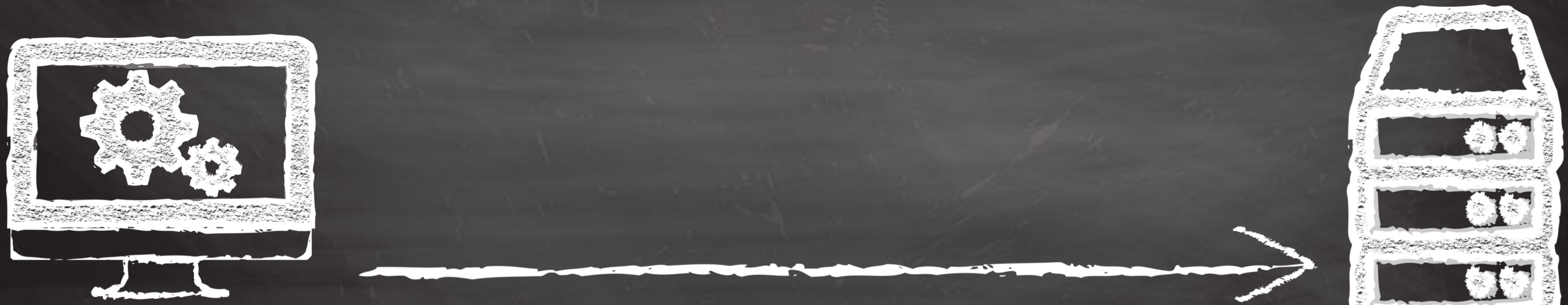
What about discovery?

Client needs only one URL



Client presents what it can do

```
{
    "capabilities": [
        "foo",
        "bar",
        "ext-1",
        "ext-2",
        ...
    ]
}
```



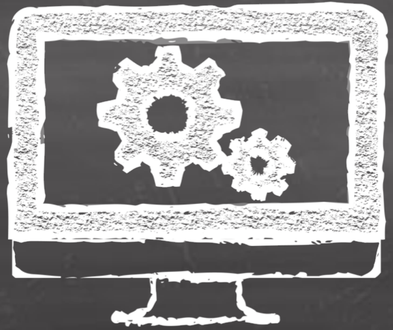
AS chooses from the list

```
{
    "capabilities": [
        "foo",

        "ext-1",

        ...
    ]
}
```

Mapping concepts to OAuth 2



Mapping to OAuth2

client_id client_secret	display_handle
client_assertion dpop mtls	key proofing
scope	resource_handle
refresh_token	transaction handle
PCT (UMA)	user_handle
id_token	user assertions

Mapping to OAuth2

```
{
    "display": "client_id"
    "resources": ["scope1", "scope2"],
    "key": "client_id"
}
```

Pros and Cons

✓

- Wider set of use cases
- More secure by default
- Built on existing experience
- Simpler data model
- Fewer moving parts
- Static and dynamic scenarios
- Multimodal JSON

✗

- Not backwards compatible
- Different assumptions
- Different data model
- Multimodal JSON
- Unknown in large deployment scale
- We don’t know what’s broken yet

Making XYZ from OAuth 2

- PAR + RAR + JAR + JARM
- DPoP + PoP + MTLS
- Auth Code, Device, Exchange, Refresh, Assertion, CIBA, OB/FAPI, Client Credentials, and UMA flows
- PKCE + State
- Plus a few things we haven't invented yet
- This is unwieldy at best…

OAUTH
3
OAUTH