FREQUENCY

# The Case for *Smarter*, Use *Specific*, *Sidecars*

*Wil Wade - ETHDenver 2025*

# Sidecars?

# Sidecar?

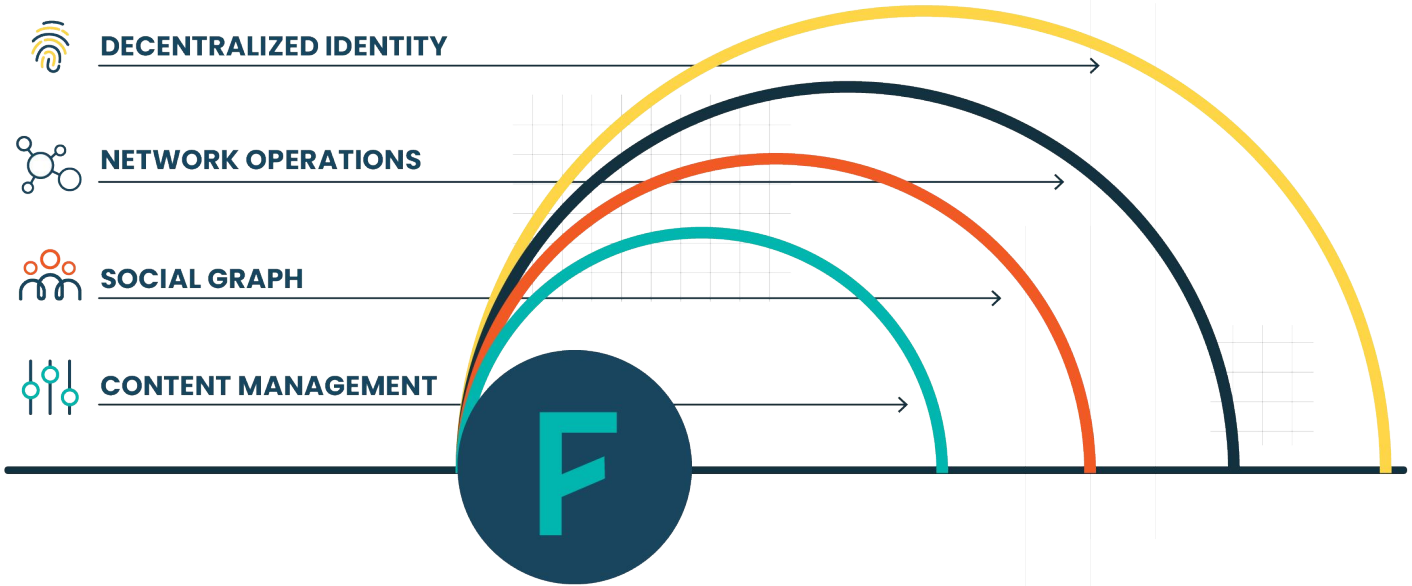
- **Standardized**
- **API Server**
- **Gateway to other servers**
- **Run by the API consumer***

# Why?

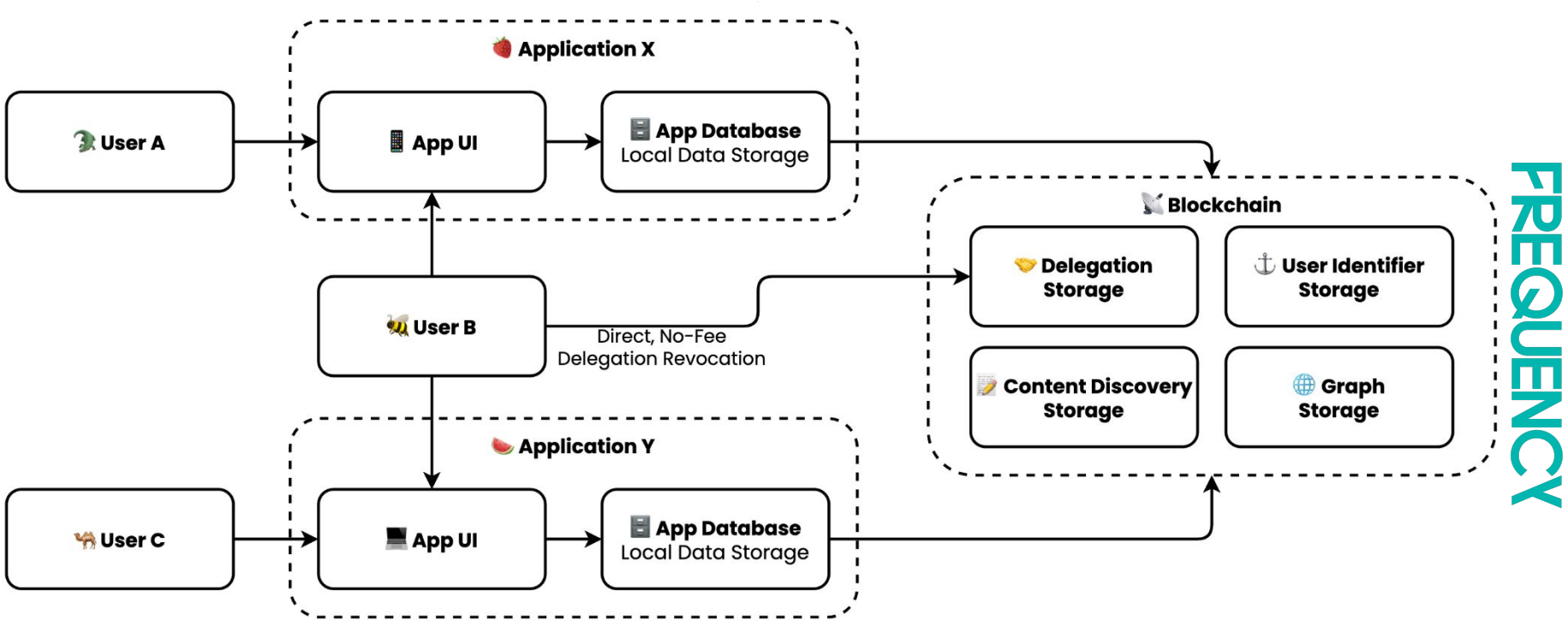
# This is a Story of Complexity

# Specific Use Case: Social Media

# Social Media: It’s a lot



# Our Use Case:
# Social Media Infrastructure for Applications

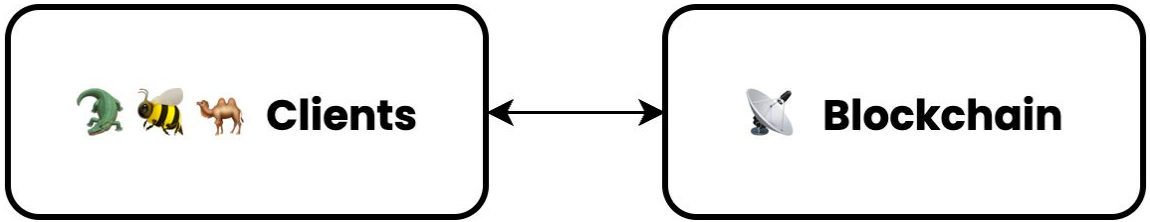

Application X
User A
App UI
App Database
Local Data Storage
Blockchain
Delegation Storage
User Identifier Storage
User B
Direct, No-Fee
Delegation Revocation
Content Discovery Storage
Graph Storage
Application Y
User C
App UI
App Database
Local Data Storage
FREQUENCY

# Problems to Solve

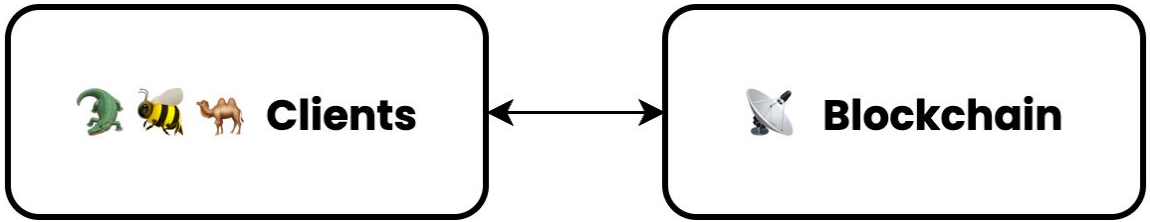
- Onboarding Speed
- Legacy Web 2 Application Integration
- Programming Language Independent
- Reduced Web3 Complexity
- Data Access
- Queuing
- Secure Key Space
- and more...

**Complexity cannot always be removed, but you *can* choose where to put it.**

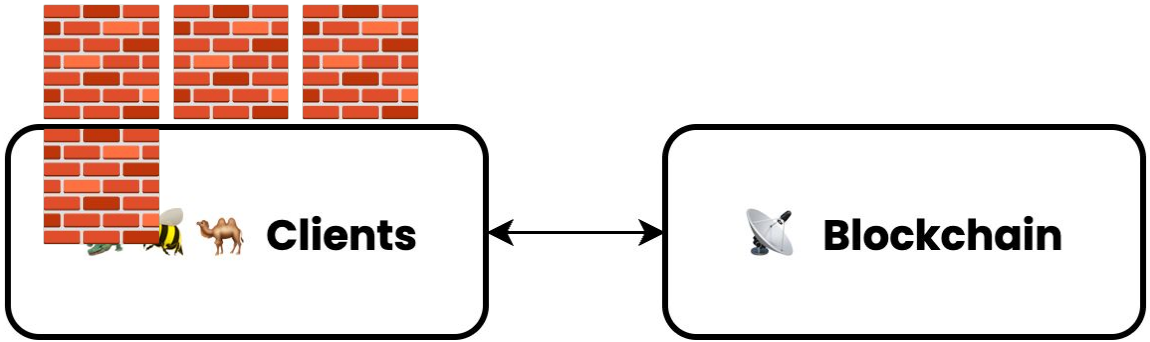
# “Classic” Web3



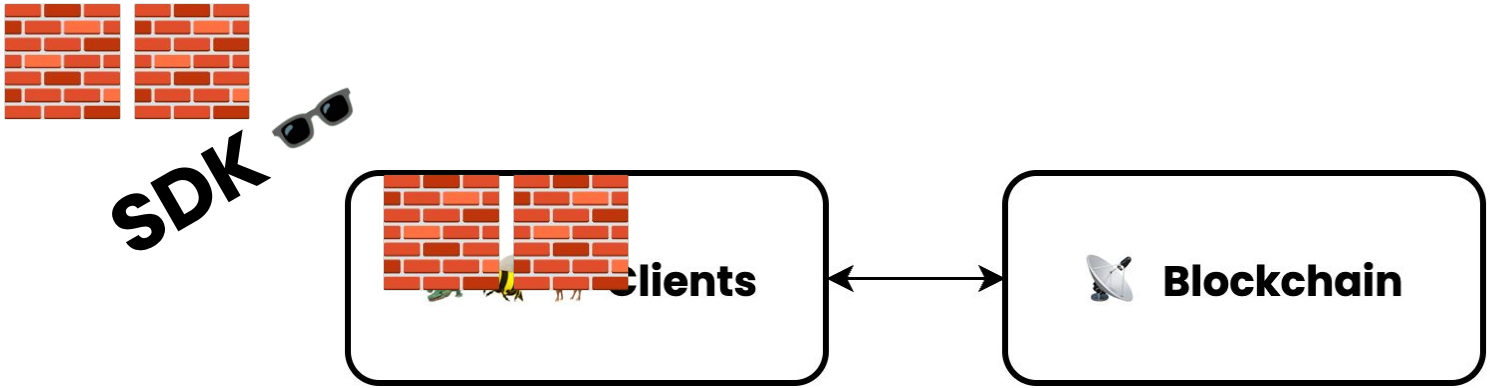
# “Classic” Web3



# “Classic” Web3



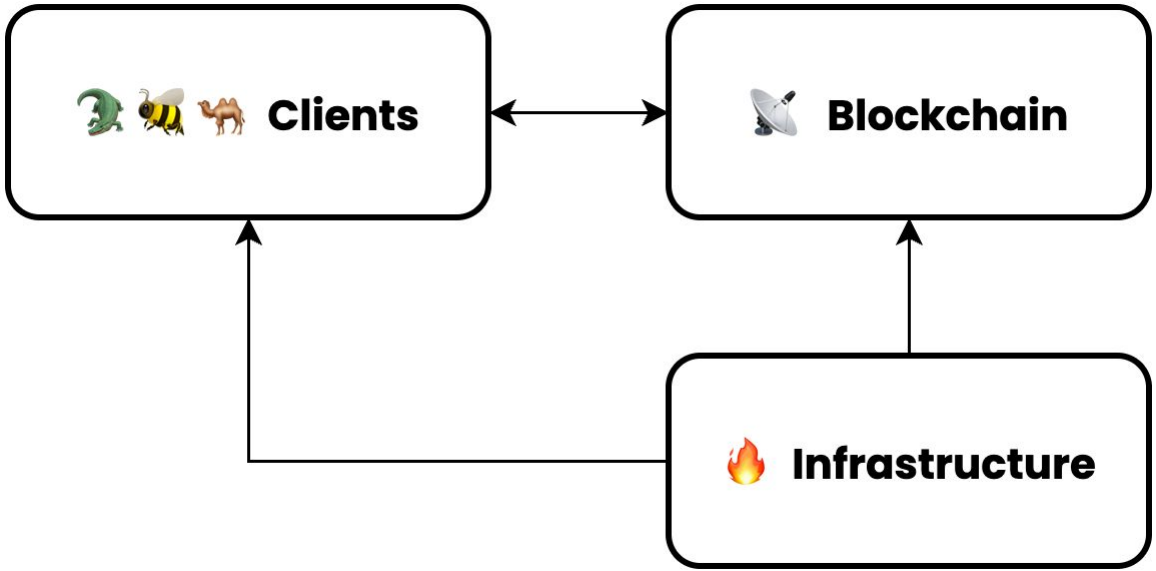
# “Classic” Web3



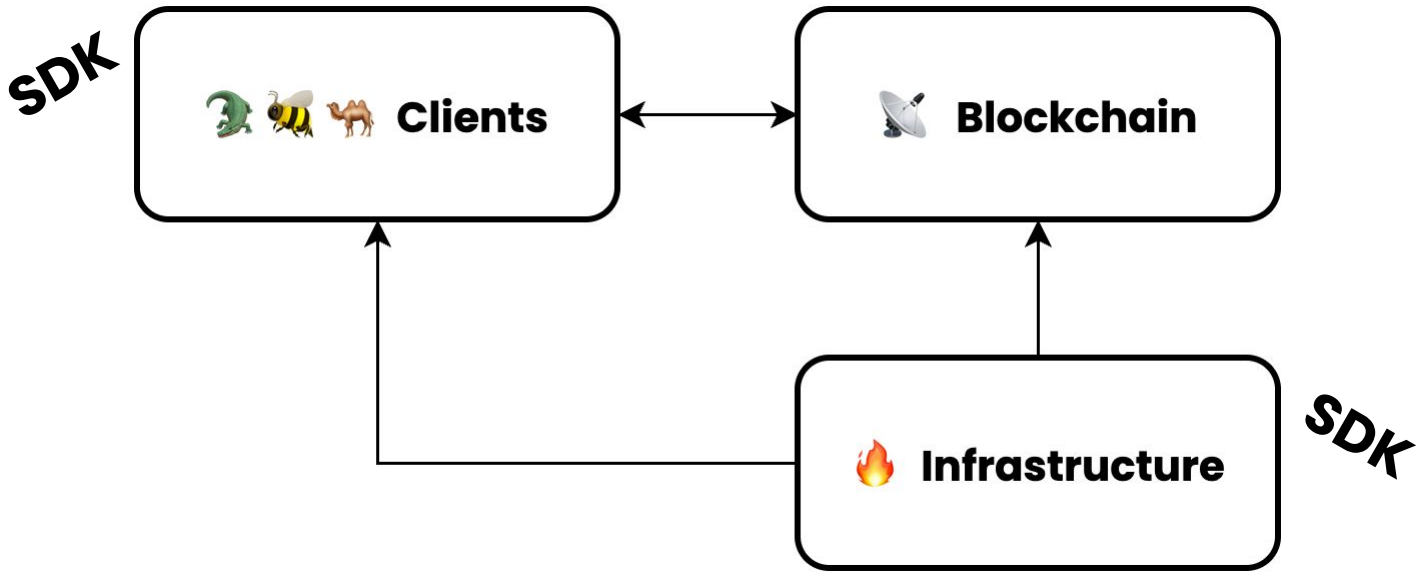
# Problems to Solve

- ✅ Onboarding Speed
- ❌ Legacy Web 2 Application Integration
- ❌ Language Independent
- ⚠️ Reduced Web3 Complexity
- ⚠️ Data Access
- ❌ Queuing
- ✅ Secure Key Space

# Web3 (Still Simplified)



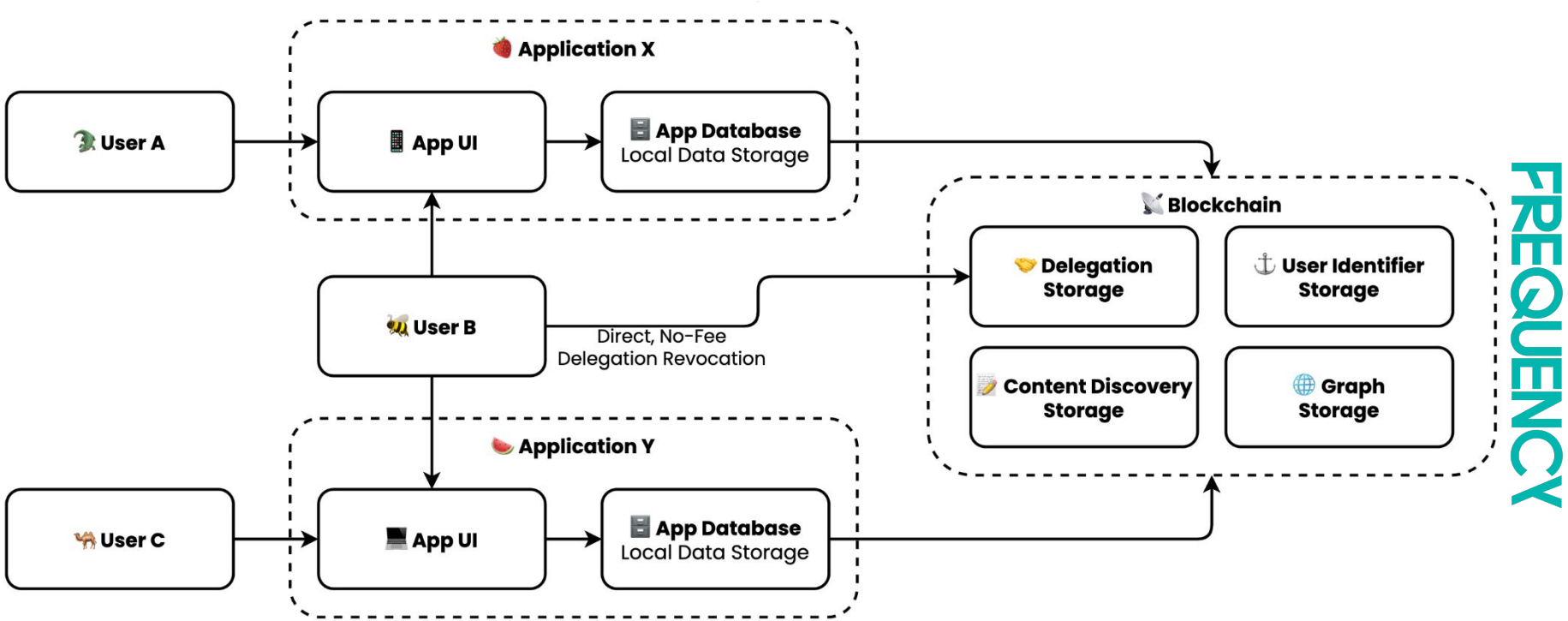
# Web3 (Still Simplified)

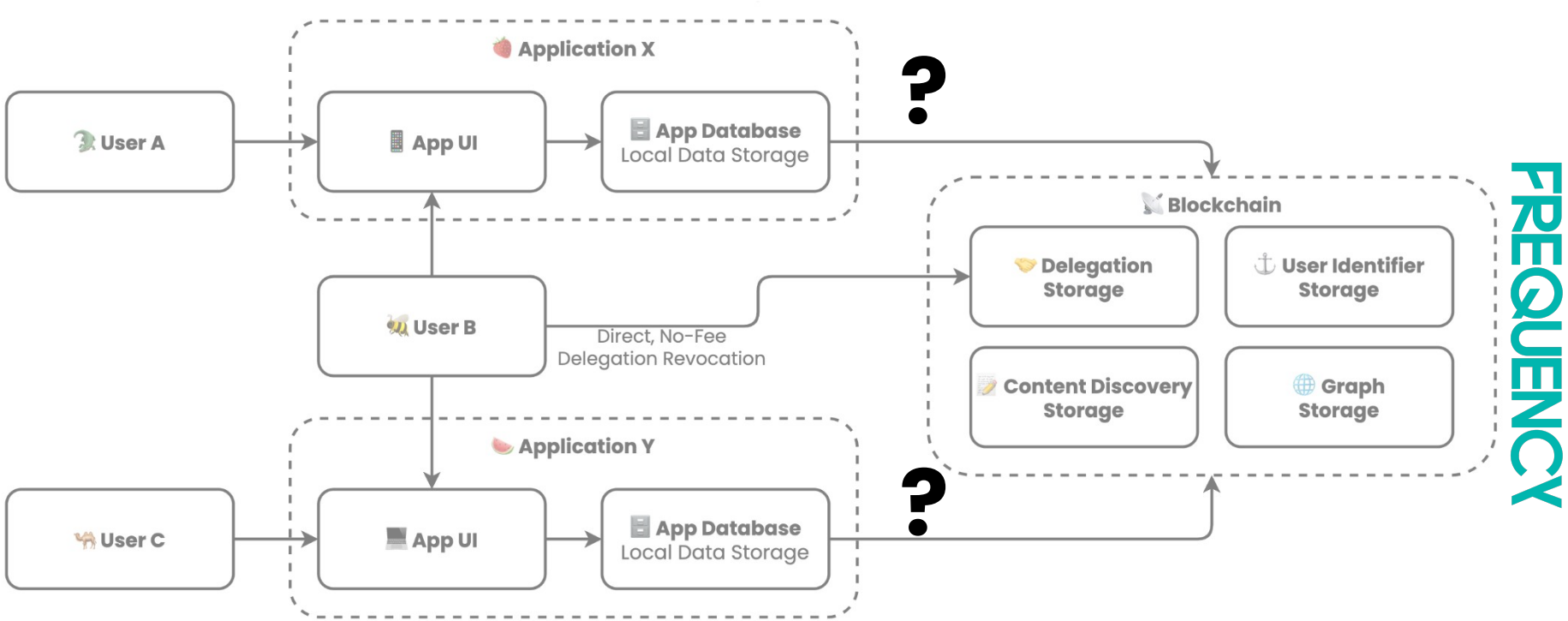


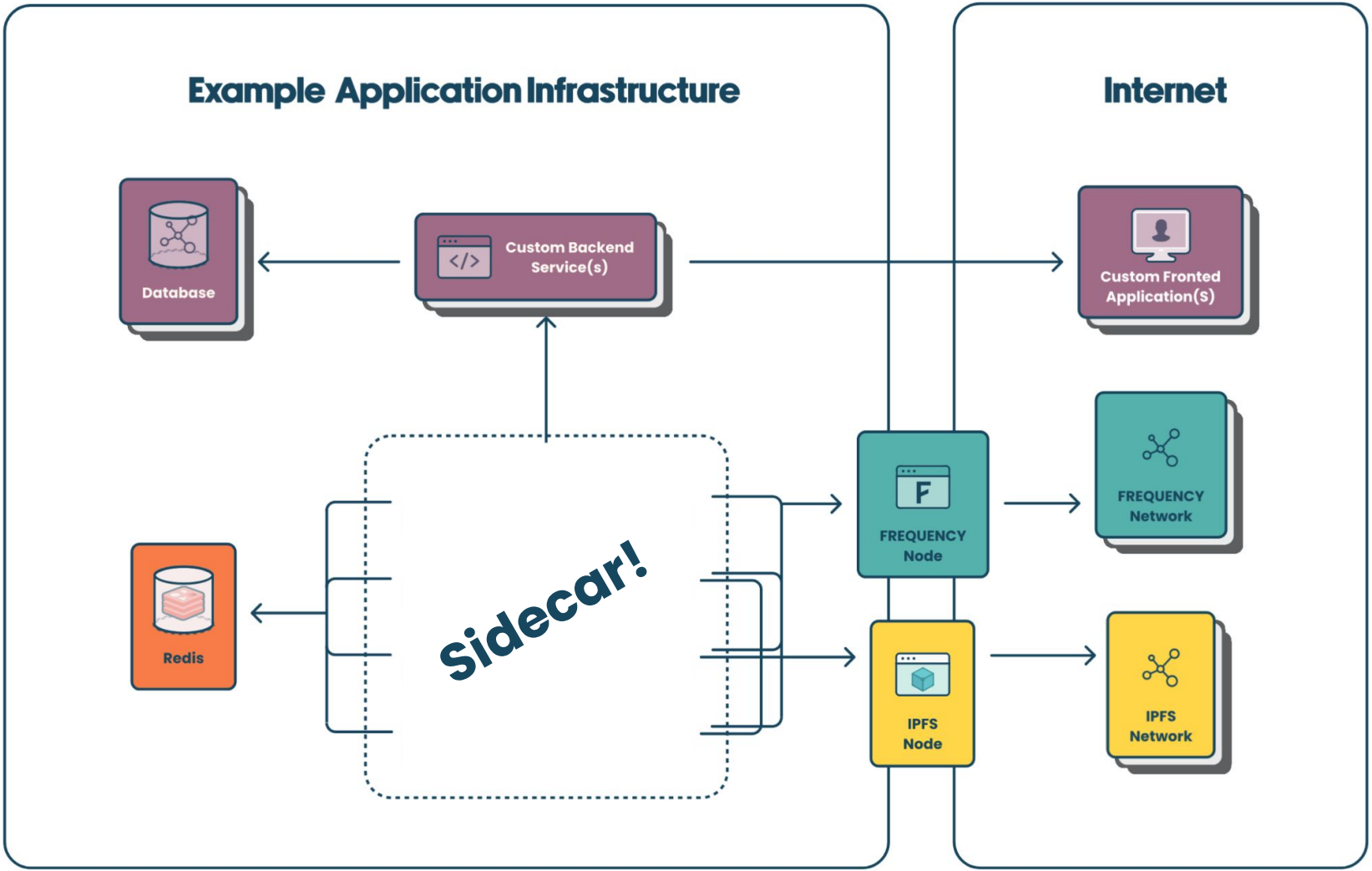
# Problems to Solve

- ✅ Onboarding Speed
- ⚠️ Legacy Web 2 Application Integration
- ✅ Language Independent
- ⚠️ Reduced Web3 Complexity
- ✅ Data Access
- ❌ Queuing
- ❌ Secure Key Space

# What if the infrastructure was interactive and smarter?


Application X
User A
App UI
App Database
Local Data Storage
Blockchain
Delegation Storage
User Identifier Storage
User B
Direct, No-Fee
Delegation Revocation
Content Discovery Storage
Graph Storage
Application Y
User C
App UI
App Database
Local Data Storage
FREQUENCY

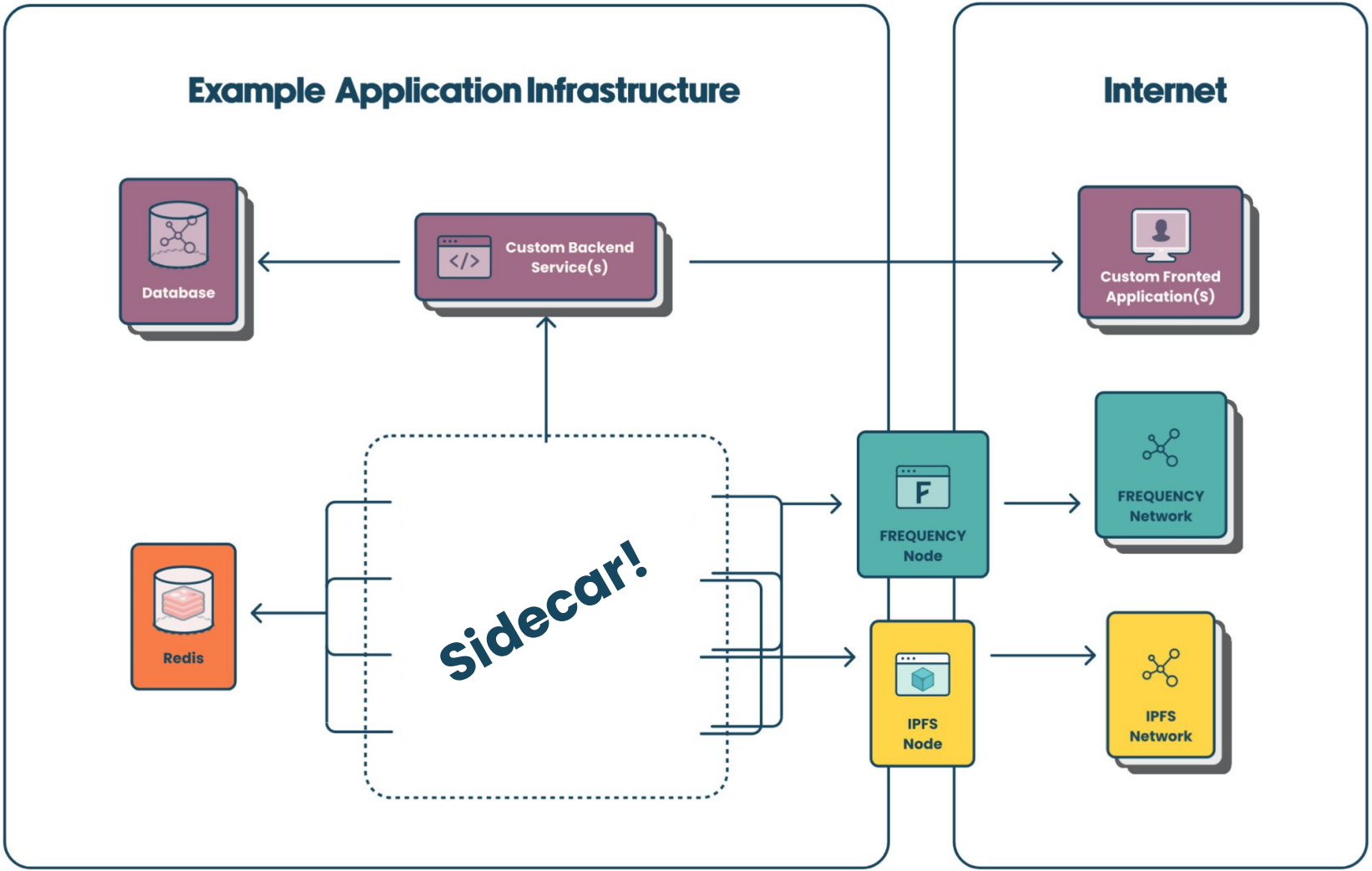

Application X
User A
App UI
App Database
Local Data Storage
?
Blockchain
Delegation Storage
User Identifier Storage
Content Discovery Storage
Graph Storage
User B
Direct, No-Fee
Delegation Revocation
Application Y
User C
App UI
App Database
Local Data Storage
?
FREQUENCY


Example Application Infrastructure
Internet
Database
Custom Backend Service(s)
Custom Fronted Application(s)
Redis
Sidecar!
FREQUENCY Node
FREQUENCY Network
IPFS Node
IPFS Network

# Problems to Solve

- ✅ Onboarding Speed
- ✅ Legacy Web 2 Application Integration
- ✅ Programming Language Independent
- ✅ Reduced Web3 Complexity
- ✅ Data Access
- ✅ Queuing
- ✅ Secure Key Space

# Shared infrastructure or shared code?


Example Application Infrastructure
Internet
Database
Custom Backend Service(s)
Custom Fronted Application(s)
Redis
Sidecar!
FREQUENCY Node
FREQUENCY Network
IPFS Node
IPFS Network

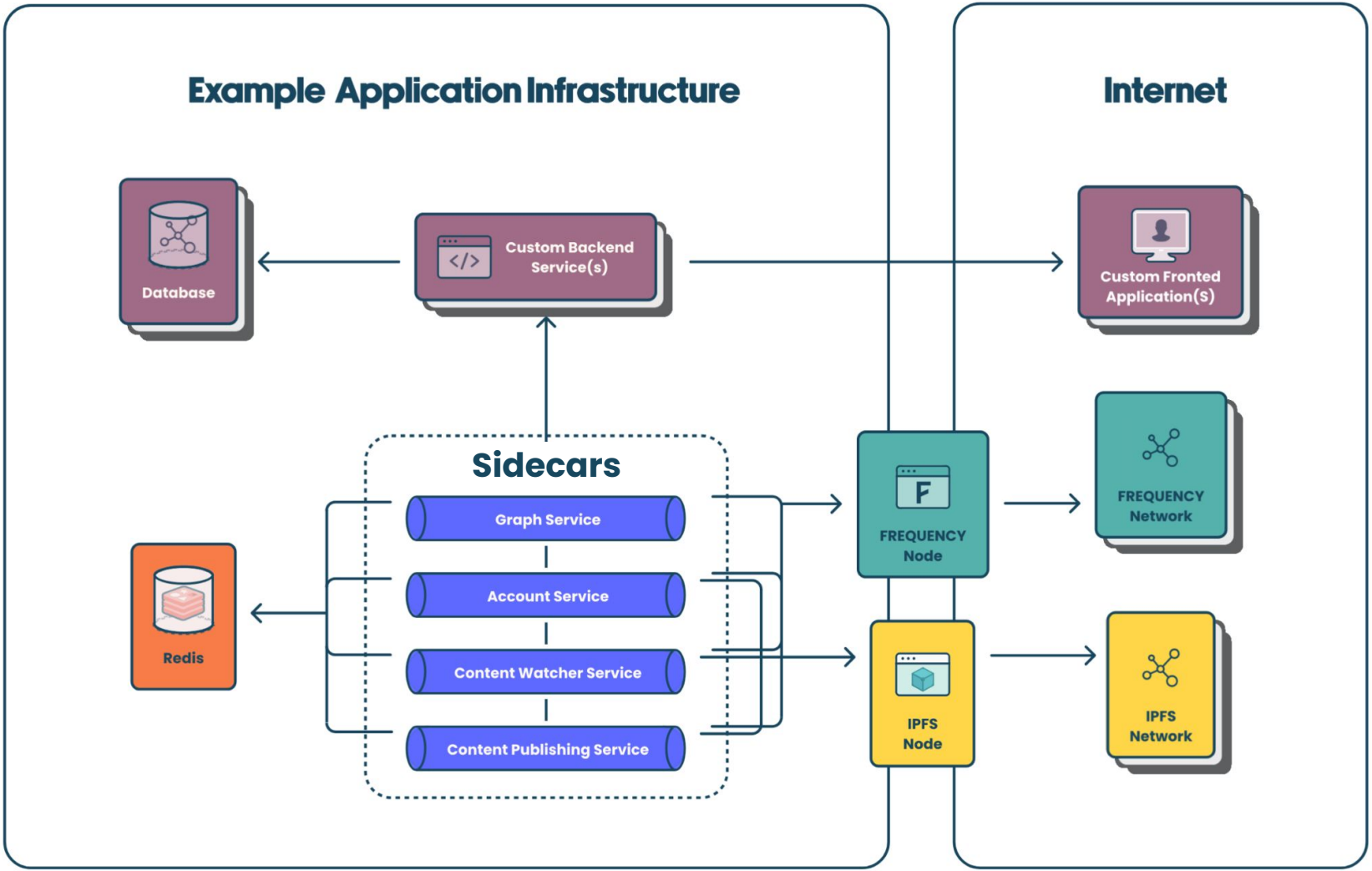
# Sidecar Separation of Concerns

DECENTRALIZED IDENTITY

NETWORK OPERATIONS

SOCIAL GRAPH

CONTENT MANAGEMENT


Example Application Infrastructure
Internet
Database
Custom Backend Service(s)
Custom Fronted Application(s)
Sidecars
Graph Service
Account Service
Content Watcher Service
Content Publishing Service
Redis
FREQUENCY Node
FREQUENCY Network
IPFS Node
IPFS Network

# When to use a Sidecar?

# When to use a Sidecar?

## Services

When your users are not humans, speed, error recovery, and risk are all different.

This can also apply if you are leveraging EIP-4337 and running an alt-mempool.

# When to use a Sidecar?

## Services

When your users are not humans, speed, error recovery, and risk are all different.

This can also apply if you are leveraging EIP-4337 and running an alt-mempool.

## Off-Chain/Complex Data

Data interactions such as submitting, retrieving, and validating are complex and requires long polls, caches, and more for performance.

Do you really want your users to deal with IPFS?

# When to use a Sidecar?

## Services

When your users are not humans, speed, error recovery, and risk are all different.

This can also apply if you are leveraging EIP-4337 and running an alt-mempool.

## Off-Chain/Complex Data

Data interactions such as submitting, retrieving, and validating are complex and requires long polls, caches, and more for performance.

Do you really want your users to deal with IPFS?

## Web2 Integration

Web2 Applications speak APIs, but they are often built assuming different timescales and response expectations.

# Not Convinced?
# Let’s take a look at ENS

# ENS SDKs

**React**

- Wagmi

**Python**

- web3.py

**Kotlin**

- KEthereum

**Go**

- go-ens
- ethereal

**JavaScript**

- Viem
- Ethers
- ENSjs
- Web3.js
- Thirdweb

**Rust**

- Ethers.rs
- Alloy

**NuGet**

- Nethereum

**Java**

- web3j

**Delphi**

- delphereum

# ENS: “Simple” Resolve

- I need a connection to Ethereum
  - Is it up?
  - Retries?
  - Finalization?
- What about off-chain resolutions?
  - What L2s do I need to connect to?
- Should I do a reverse lookup to verify the canonical address?
- Should I do the CCIP Fetch?
- Caching?

That’s just for a two parameter, read-only function from a great SDK!

# Secondary Benefit: API Docs

```yaml
openapi: 3.0.0
paths:
  /v2/accounts/siwf:
    get:
      operationId: AccountsControllerV2_getRedirectUrl
      summary: Get the Sign In With Frequency Redirect URL
      parameters:
        - name: credentials
          required: false
          in: query
          description: >-
            List of credentials using the types: "VerifiedGraphKeyCredential",
            "VerifiedEmailAddressCredential", and
            "VerifiedPhoneNumberCredential". Note that Contact related
            verifiable credentials will be nested into an anyOf request form.
          schema:
            example:
              - VerifiedGraphKeyCredential
              - VerifiedEmailAddressCredential
              - VerifiedPhoneNumberCredential
            type: array
            items:
              type: string
        - name: permissions
          required: false
          in: query
          description: >-
            The list of permissions using the Frequency Schema names and
            versions. Pattern: `<namespace>.<name>@v<version integer>` e.g.
            `dsnp.broadcast@v2`
          schema:
            example:
              - dsnp.broadcast@v2
              - dsnp.private-follows@v1
              - dsnp.reply@v2
              - dsnp.reaction@v1
              - dsnp.tombstone@v2
              - dsnp.update@v2
              - frequency.default-token-address@v1
            type: array
            items:
              type: string
        - name: callbackUrl
          required: true
          in: query
          description: >-
```

## v2/accounts

- GET `/v2/accounts/siwf` Get the Sign In With Frequency Redirect URL
- POST `/v2/accounts/siwf` Process the result of a Sign In With Frequency v2 callback

## v1/accounts

- GET `/v1/accounts/siwf` Get the Sign In With Frequency configuration
- POST `/v1/accounts/siwf` Request to Sign In With Frequency
- GET `/v1/accounts/{msaId}` Fetch an account given an MSA Id
- GET `/v1/accounts/account/{accountId}` Fetch an account given an Account Id
- GET `/v1/accounts/retireMsa/{accountId}` Get a retireMsa unsigned, encoded extrinsic payload.
- POST `/v1/accounts/retireMsa` Request to retire an MSA ID.

## v2/delegations

- GET `/v2/delegations/{msaId}` Get all delegation information associated with an MSA Id
- GET `/v2/delegations/{msaId}/{providerId}` Get an MSA's delegation information for a specific provider

## v1/delegation

- GET `/v1/delegation/{msaId}` Get the delegation information associated with an MSA Id

# Final Thought: Where are you containing your complexity?

FREQUENCY.xyz

Thank You!