Concordium.com

Concordium @ Graviton

An introduction to:

- Blockchains,
- Concordium,
- & Rust!

Kasper Dissing Bargsteen
Doris Benda

In this Presentation

Here's what we'll cover:

Why Concordium?

The Rust programming language

Building a voting-themed smart contract

Testing and deploying the smart contract

Additional resources

Questions

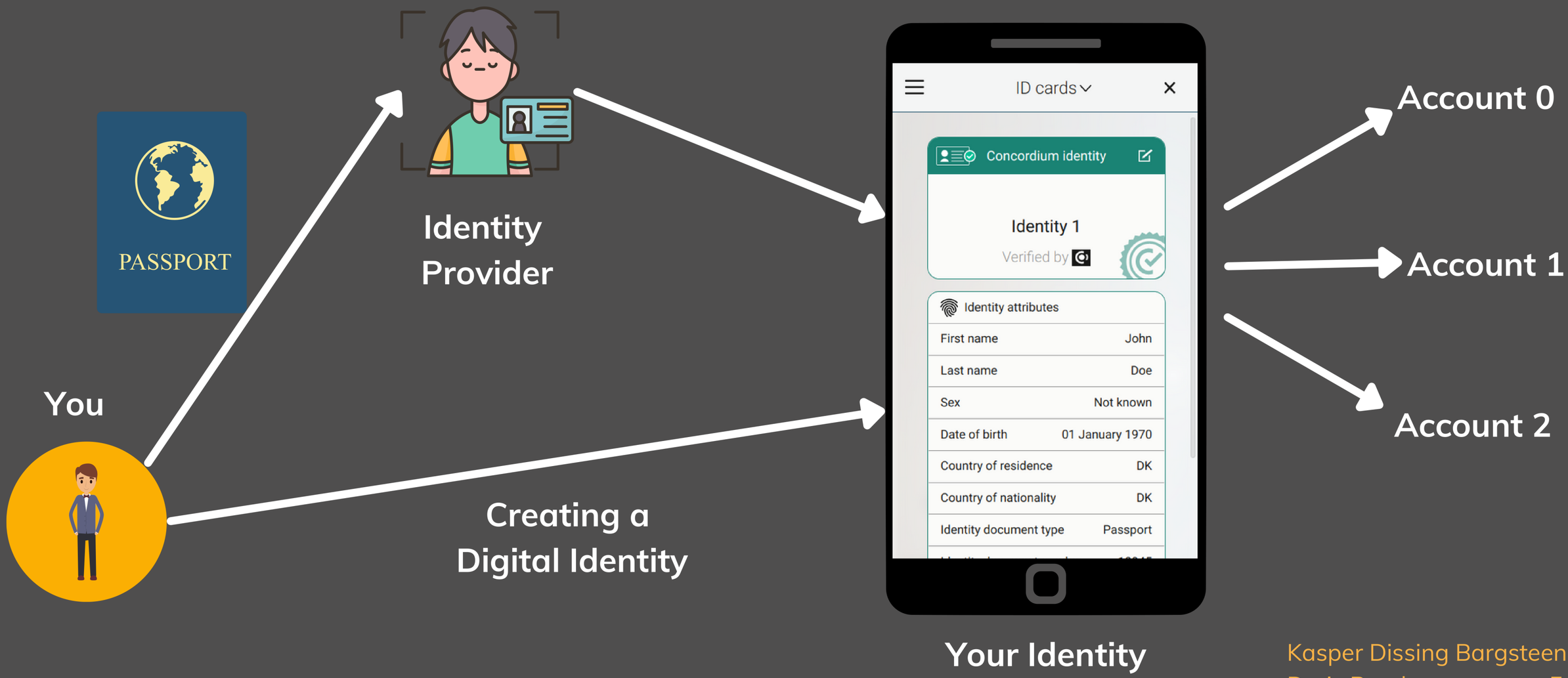
Why Concordium?

> “Concordium is the Layer 1, science-backed blockchain creating a safer digital world”

Why Concordium?

- Most important features for what we are building today:
  - Concordium is the only blockchain to have ***identity verification built into the protocol level.***
  - You can proof statements about yourself (and your id) via Concordiums **zero-knowledge proof technology**.

Concordiums ID-layer and account creation



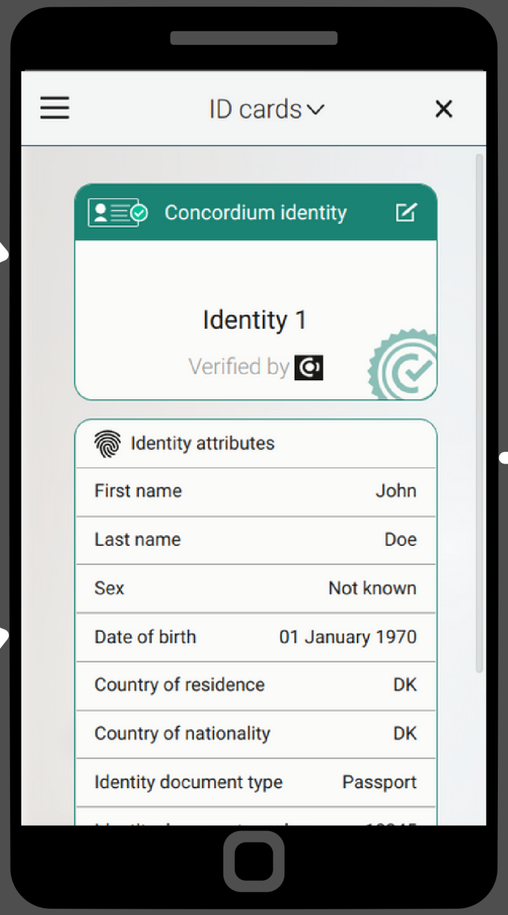
Concordium.com

Revoking anonymity

Anonymity Revokers

Law Enforcement

Identity Provider



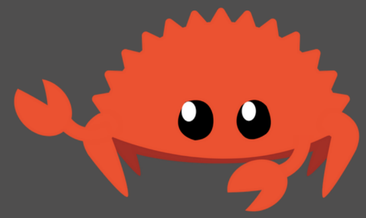
Your Identity

Legal Warrant

You

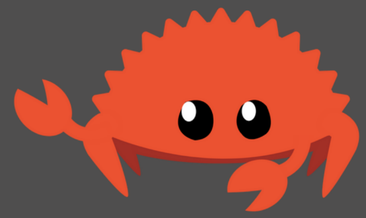
The Rust programming language

Ferris, the official Rust mascot



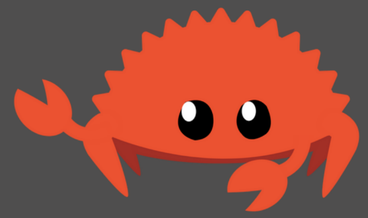
What is Rust?

- Systems programming language
  - So you can build an OS in it! (like C/C++)
- Most popular language seven years in a row (StackOverflow dev survey)
- Strong type system
  - Everything has a type, and the compiler will catch a lot of your mistakes
- Nice, high-level abstractions
  - At ***zero runtime cost*** 😮 (but how?)
- *Ownership* model for memory handling
  - No garbage collector, no manual freeing of data
  - Gives strong guarantees against common bugs in your code



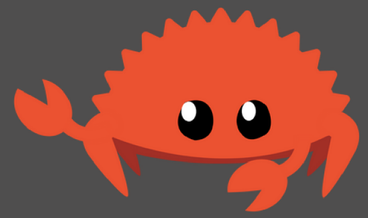
Memory models

- Manual handling of memory
  - C / C ++
  - Allocate (malloc, alloc)
  - Free memory (free)
- Garbage collection
  - Python, JavaScript, C#, Go, Java, ...
  - A garbage collector program runs in the background and cleans up (frees) memory
- Rust's ownership model
  - No manual freeing of data
  - No garbage collection



The ownership model

- Three rules:
  - Each value has an *owner*
  - There can only be one owner at a time
  - When the owner goes out of scope, the value is dropped (freed)



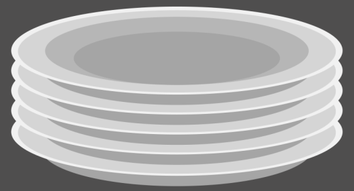
The ownership model

- Three rules:
  - Each value has an *owner*
  - There can only be one owner at a time
  - When the owner goes out of scope, the value is dropped (freed)

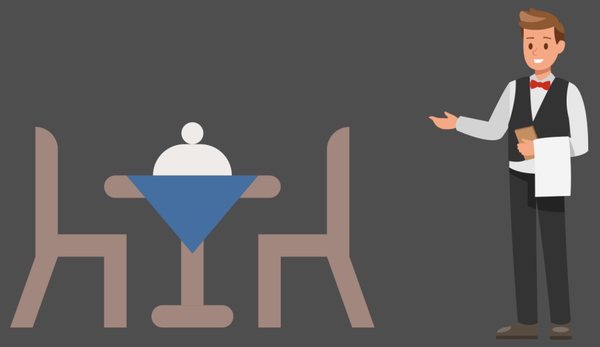
```
{
    let s = String::from("hello"); // s is valid from this point forward

    // do stuff with s
}                                  // this scope is now over, and s is no
                                   // longer valid
```

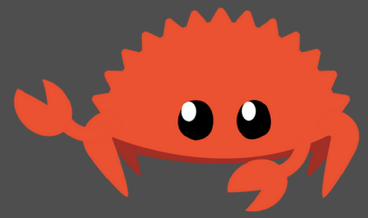
Stack vs heap

- The memory (RAM) available for a program is split in two: the stack and the heap
- The stack:
  - Data of known, fixed sizes (numbers, bools, pointers, etc.)
  - Put on top of each other, like a stack of plates.
  - Last-in-first-out => Fast



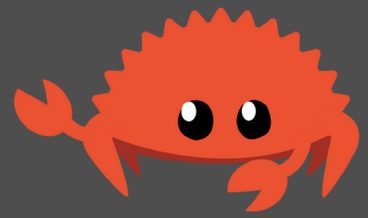
- The heap:
  - Data of variable and changing sizes (strings, lists, trees, etc.)
  - Less organized, you request a certain amount of space and get a pointer to it
  - Example: getting a table at a restaurant
  - The pointer can be stored on the stack
  - Following pointers => Slower





The Ownership model

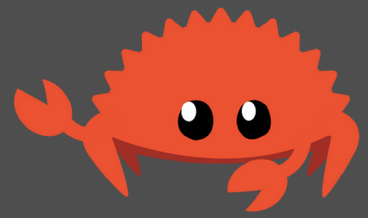
```
let x = 5;
let y = x;

println!("x = {}, y = {}", x, y);
```



The Ownership model

```
let x = 5;
let y = x;

println!("x = {}, y = {}", x, y);
```

```
x = 5, y = 5
```

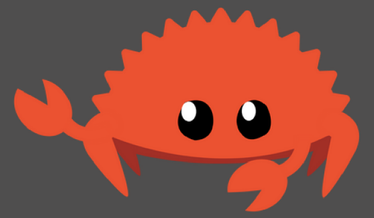


The Ownership model

```rust
let x = 5;
let y = x;

println!("x = {}, y = {}", x, y);
```

```
x = 5, y = 5
```

```rust
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```



The Ownership model

```rust
let x = 5;
let y = x;

println!("x = {}, y = {}", x, y);
```

```
x = 5, y = 5
```

```rust
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

```
$ cargo run
   Compiling ownership v0.1.0 (file:///projects/ownership)
error[E0382]: borrow of moved value: `s1`
 --> src/main.rs:5:28
  |
2 |     let s1 = String::from("hello");
  |         -- move occurs because `s1` has type `String`, which does not implement the
3 |     let s2 = s1;
  |              -- value moved here
4 |
5 |     println!("{}, world!", s1);
  |                            ^^ value borrowed here after move
  |
  = note: this error originates in the macro `$crate::format_args_nl` which comes from t

For more information about this error, try `rustc --explain E0382`.
error: could not compile `ownership` due to previous error
```

The Ownership model

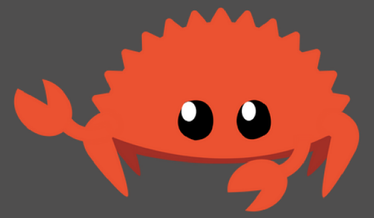
```rust
let x = 5;
let y = x;

println!("x = {}, y = {}", x, y);
```

```
x = 5, y = 5
```

Numbers have a known, fixed size and live on the stack. So they implement the Copy trait. The value of x is copied to y.

```rust
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

```
$ cargo run
   Compiling ownership v0.1.0 (file:///projects/ownership)
error[E0382]: borrow of moved value: `s1`
 --> src/main.rs:5:28
  |
2 |     let s1 = String::from("hello");
  |         -- move occurs because `s1` has type `String`, which does not implement the
3 |     let s2 = s1;
  |              -- value moved here
4 |
5 |     println!("{}, world!", s1);
  |                            ^^ value borrowed here after move
  |
  = note: this error originates in the macro `$crate::format_args_nl` which comes from t

For more information about this error, try `rustc --explain E0382`.
error: could not compile `ownership` due to previous error
```



The Ownership model

```rust
let x = 5;
let y = x;

println!("x = {}, y = {}", x, y);
```

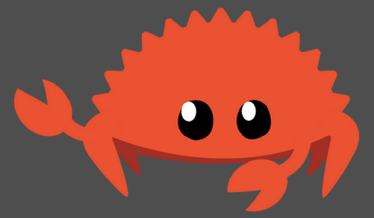
```
x = 5, y = 5
```

Numbers have a known, fixed size and live on the stack. So they implement the Copy trait. The value of x is copied to y.

trait = interface

```rust
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

```
$ cargo run
   Compiling ownership v0.1.0 (file:///projects/ownership)
error[E0382]: borrow of moved value: `s1`
 --> src/main.rs:5:28
  |
2 |     let s1 = String::from("hello");
  |         -- move occurs because `s1` has type `String`, which does not implement the
3 |     let s2 = s1;
  |              -- value moved here
4 |
5 |     println!("{}, world!", s1);
  |                            ^^ value borrowed here after move
  |
  = note: this error originates in the macro `$crate::format_args_nl` which comes from t

For more information about this error, try `rustc --explain E0382`.
error: could not compile `ownership` due to previous error
```



The Ownership model

```
let x = 5;
let y = x;

println!("x = {}, y = {}", x, y);
```

```
x = 5, y = 5
```

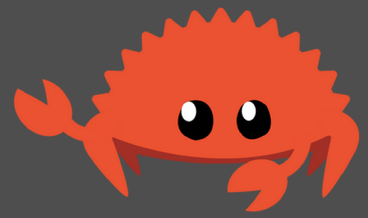
Numbers have a known, fixed size and live on the stack. So they implement the Copy trait.
The value of x is copied to y.

trait = interface

```
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

Strings have a variable size and live on the heap.
The value of s1 is moved to s2.

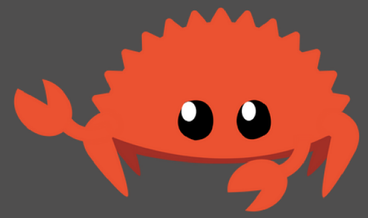
```
$ cargo run
   Compiling ownership v0.1.0 (file:///projects/ownership)
error[E0382]: borrow of moved value: `s1`
 --> src/main.rs:5:28
  |
2 |     let s1 = String::from("hello");
  |         -- move occurs because `s1` has type `String`, which does not implement the
3 |     let s2 = s1;
  |              -- value moved here
4 |
5 |     println!("{}, world!", s1);
  |                            ^^ value borrowed here after move
  |
  = note: this error originates in the macro `$crate::format_args_nl` which comes from t

For more information about this error, try `rustc --explain E0382`.
error: could not compile `ownership` due to previous error
```



The Ownership model

```
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

How to fix this?

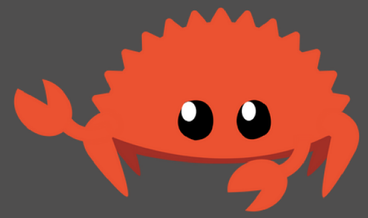


The Ownership model

Structure your program
so you don't print s1!

```
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

How to fix this?



The Ownership model

```
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

Structure your program
so you don't print s1!

Or, use the *Clone* trait:

```
let s1 = String::from("hello");
let s2 = s1.clone();

println!("s1 = {}, s2 = {}", s1, s2);
```

How to fix this?

The Ownership model

```
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

Structure your program
so you don't print s1!

Or, use the *Clone* trait:

```
let s1 = String::from("hello");
let s2 = s1.clone();

println!("s1 = {}, s2 = {}", s1, s2);
```

How to fix this?

The Ownership model

```
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

Structure your program
so you don't print s1!

Or, use the *Clone* trait:

```
let s1 = String::from("hello");
let s2 = s1.clone();

println!("s1 = {}, s2 = {}", s1, s2);
```

```
s1 = hello, s2 = hello
```

How to fix this?

The Ownership model

```
let s1 = String::from("hello");
let s2 = s1;

println!("{}, world!", s1);
```

How to fix this?

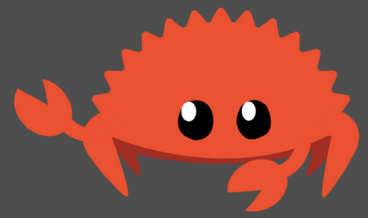
Structure your program
so you don't print s1!

Or, use the *Clone* trait:

```
let s1 = String::from("hello");
let s2 = s1.clone();

println!("s1 = {}, s2 = {}", s1, s2);
```

```
s1 = hello, s2 = hello
```

Use references!

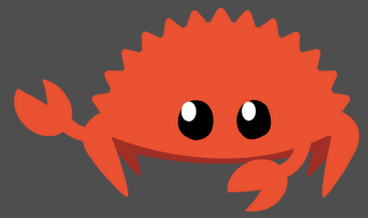


The Ownership model: references

Transfer ownership:

```rust
fn takes_ownership(s: String) {
    println!("I now own: {}", s);
} // s goes out of scope and the String is dropped
```

dropped = freed



The Ownership model: references

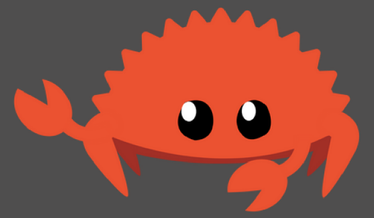
Transfer ownership:

```
fn takes_ownership(s: String) {
    println!("I now own: {}", s);
} // s goes out of scope and the String is dropped
```

dropped = freed

Borrow immutably:

```
fn just_borrow(s: &String) { // Notice the & !
    println!("Just borrowed for a while: {}", s);
} // s is still alive after this!
```



The Ownership model: references

Transfer ownership:

```rust
fn takes_ownership(s: String) {
    println!("I now own: {}", s);
} // s goes out of scope and the String is dropped
```

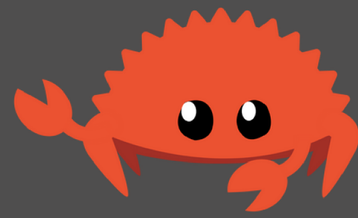
dropped = freed

Borrow immutably:

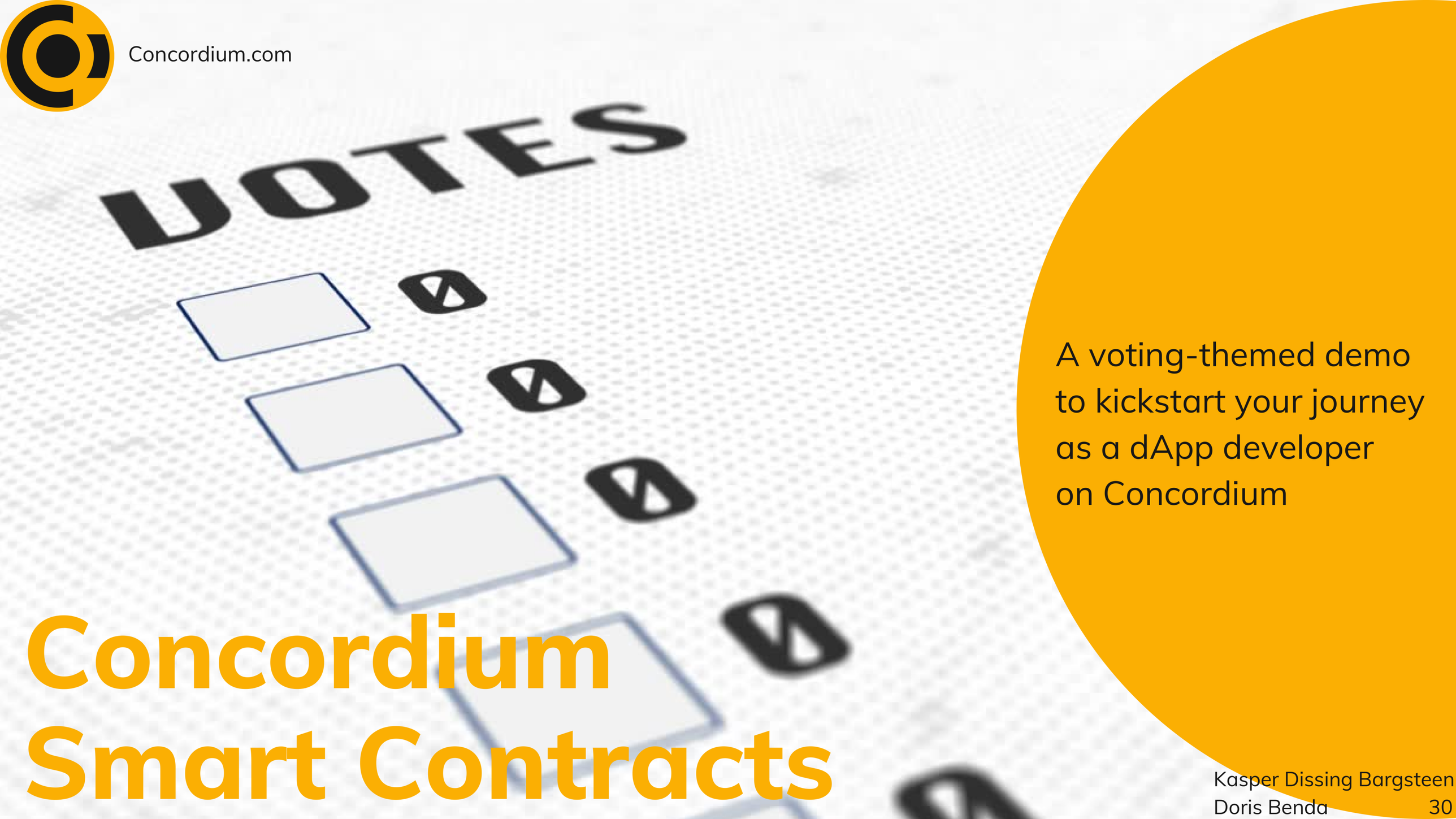
```rust
fn just_borrow(s: &String) { // Notice the & !
    println!("Just borrowed for a while: {}", s);
} // s is still alive after this!
```

Borrow mutably:

```rust
fn change(s: &mut String) { // Notice the &mut !
    s.push_str(" ← was changed");
} // s has changed but is still alive.
```



We'll cover the rest in practice!



Concordium Smart Contracts

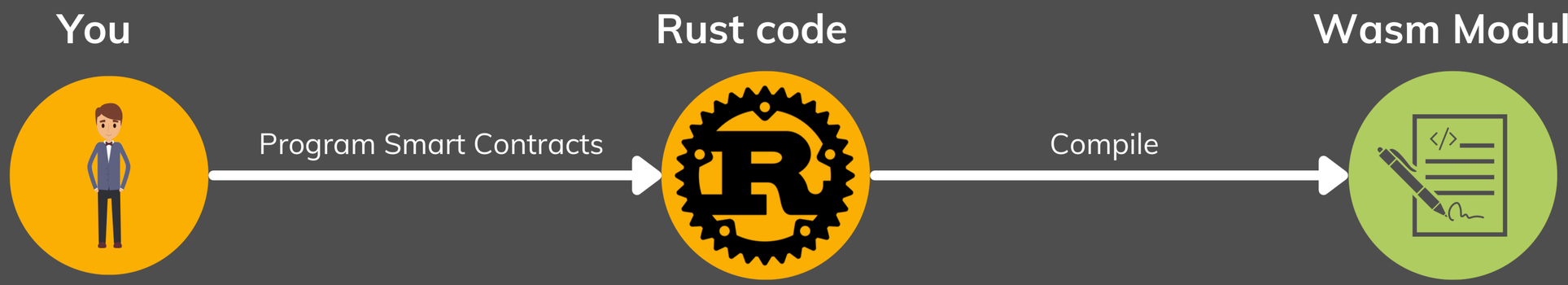
A voting-themed demo to kickstart your journey as a dApp developer on Concordium

Kasper Dissing Bargsteen
Doris Benda

Introduction To Smart Contracts

A smart contract is a user-supplied piece of code submitted to the Concordium blockchain, used to define additional behavior that is not directly part of the core protocol.

Smart contracts are deployed as Wasm modules on the Concordium chain. Rust currently has the best support to write and compile your smart contract into a Wasm module that then can be deployed to the Concordium chain.



Life Cycle Of A Smart Contract On Concordium

You

Deploy (A smart contract is first deployed to the chain as part of a Wasm module.)

Wasm Module

You

Initialize (After this,a smart contract can be initialized to obtain a smart contract instance.)

Smart Contract Instance

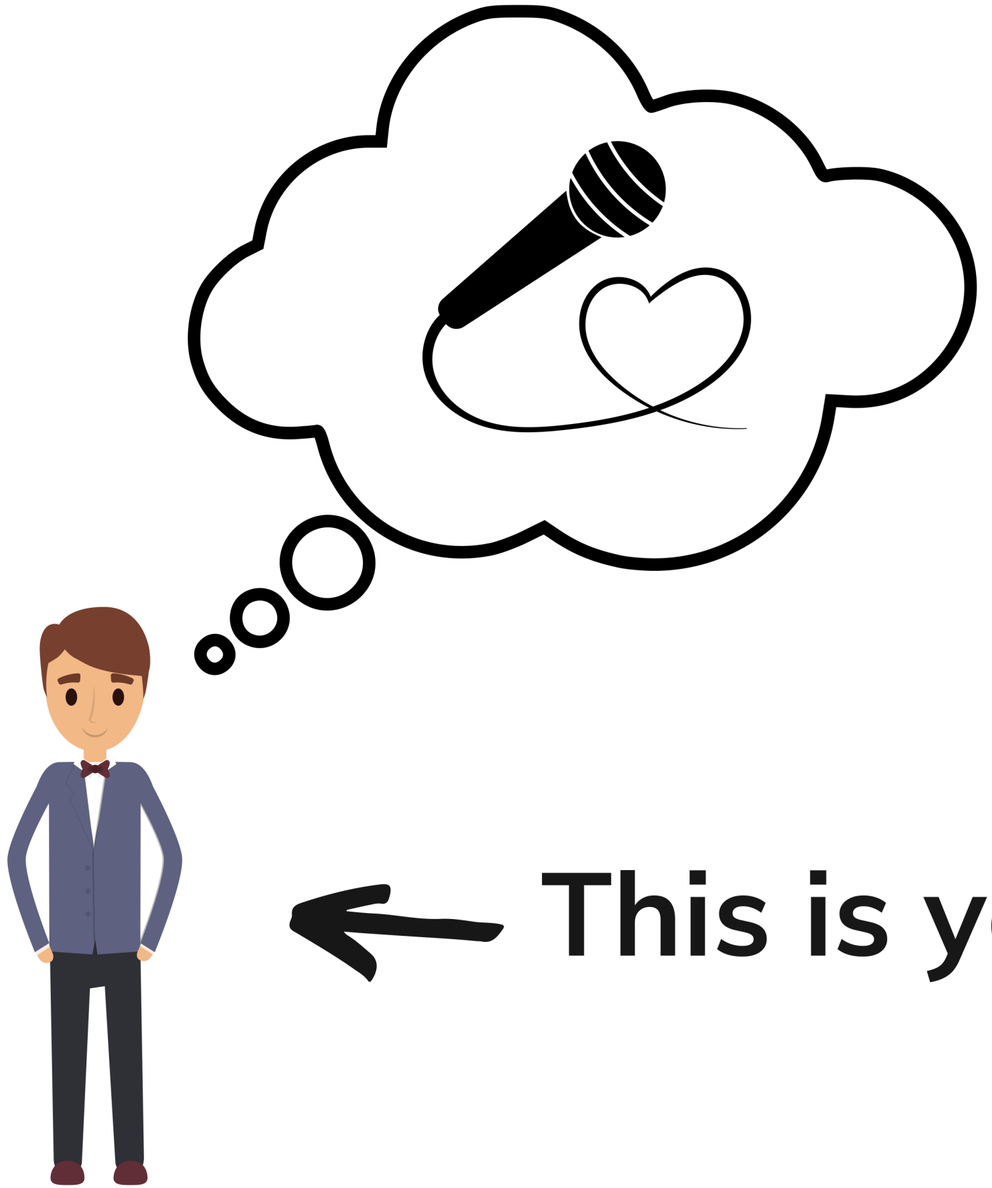
Users

Update (A smart contract instance can be repeatedly updated according to its own logic. The smart contract logic can include authentication in form of logic on who can update the smart contract under which conditions.)

Smart Contract Instance

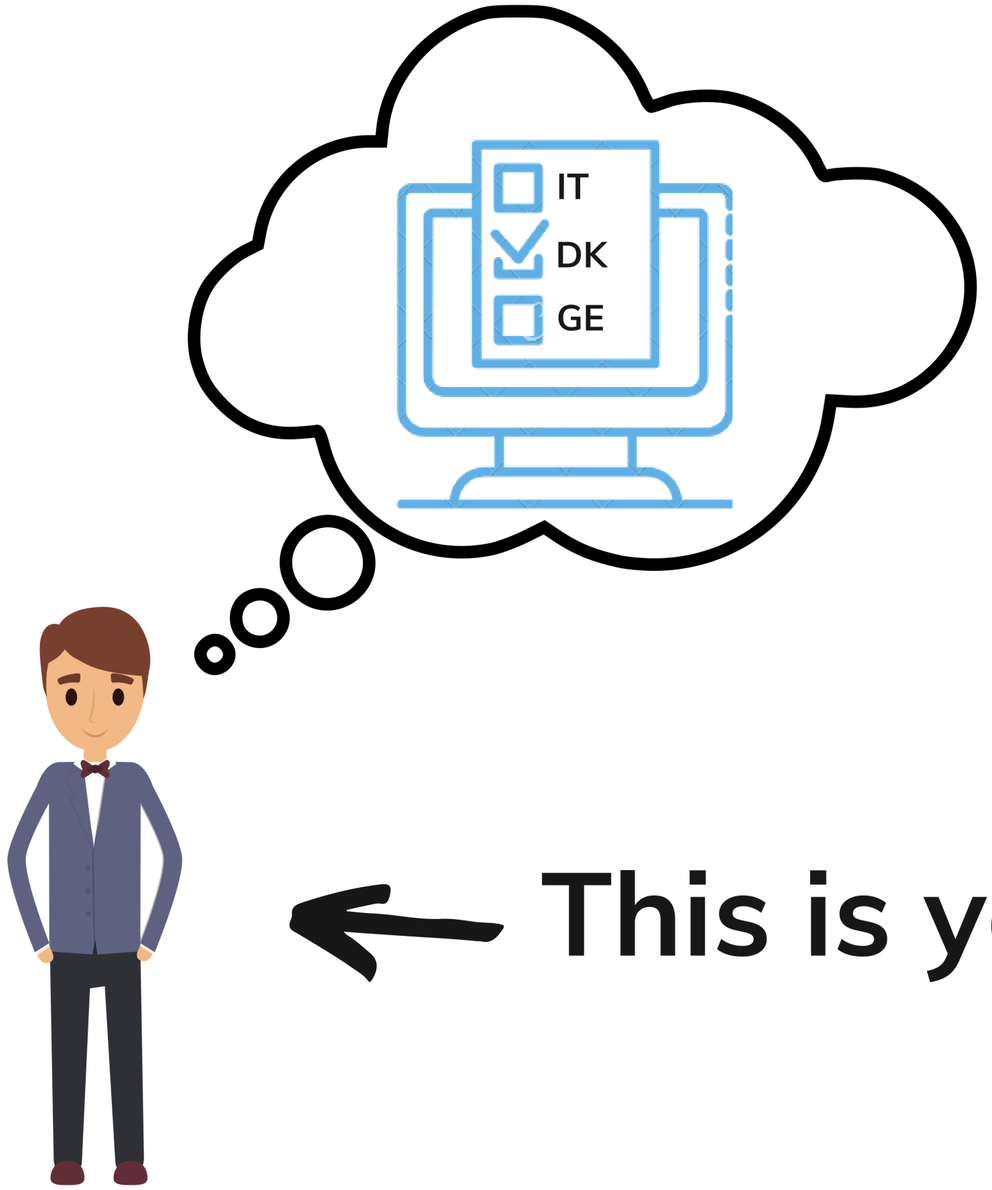
A Very Realistic Problem

This is you..



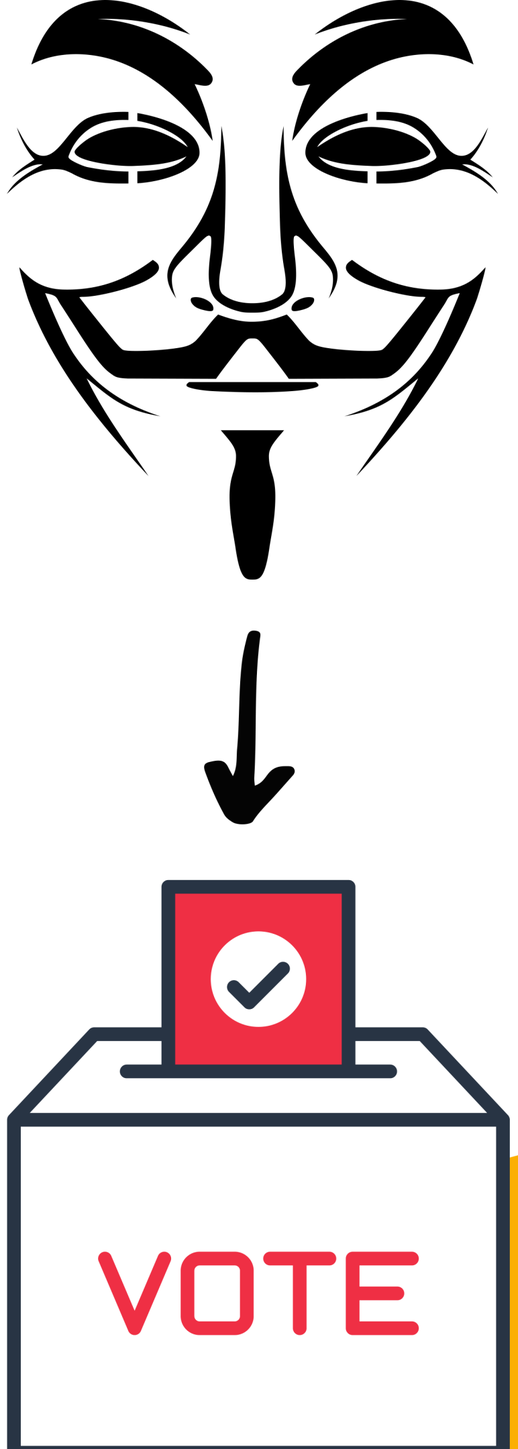
This is you..

This is you..

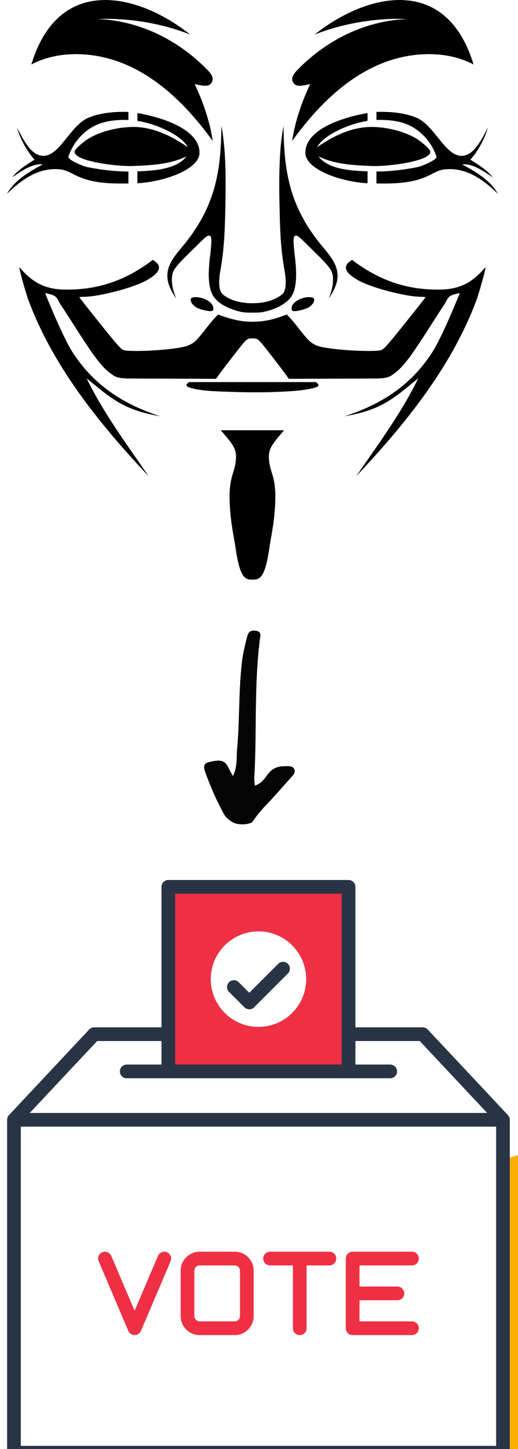


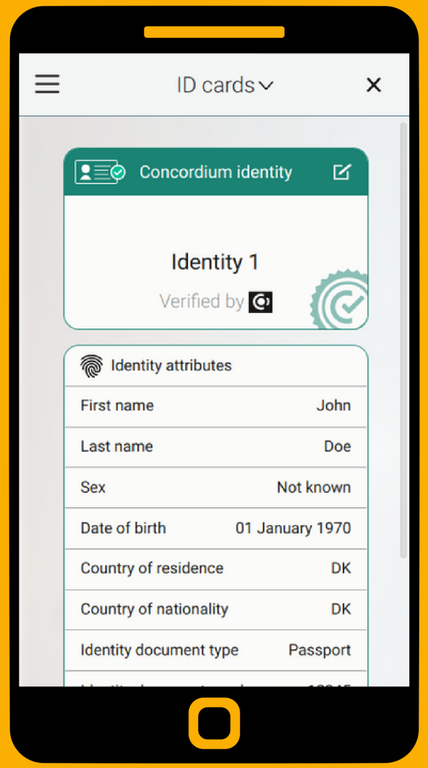
This is you..

Mister X should not be able to vote for his own country.

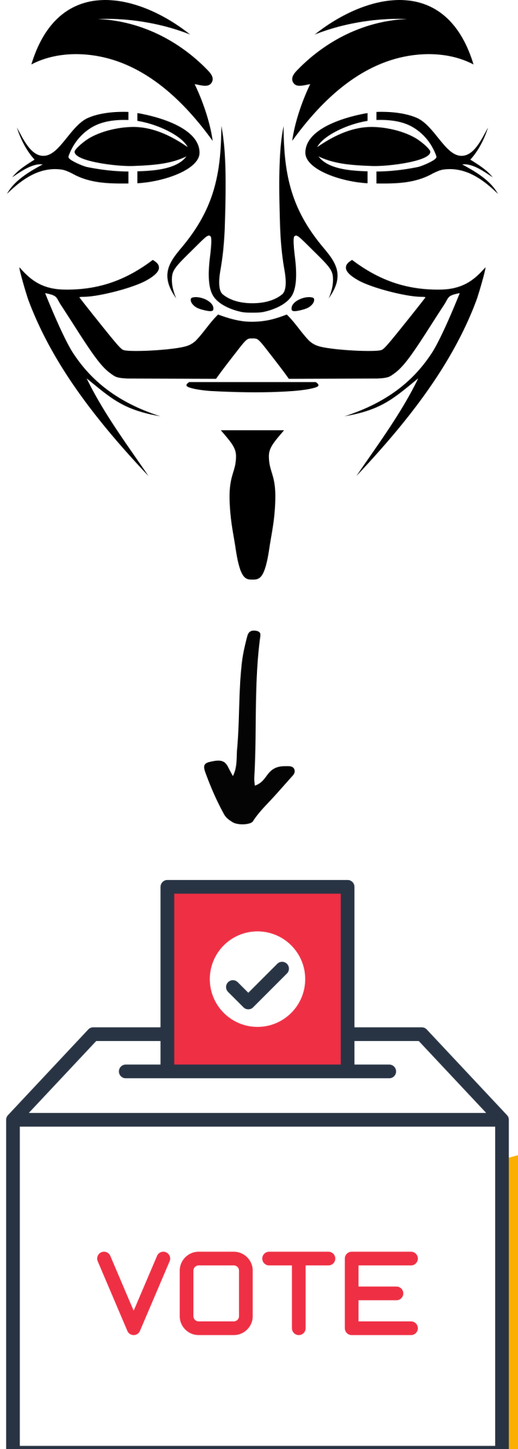


Concordium and its identity layer





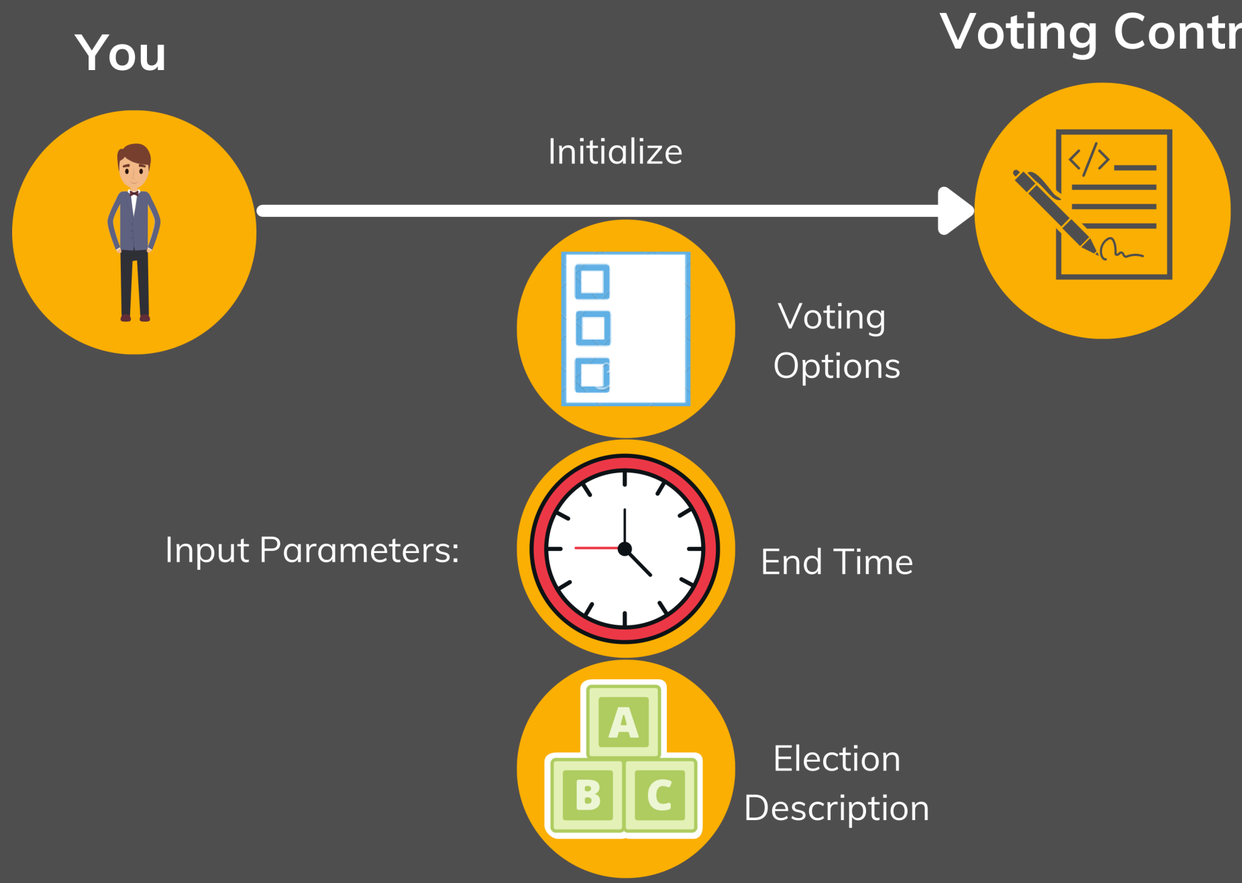
Concordium and its identity layer



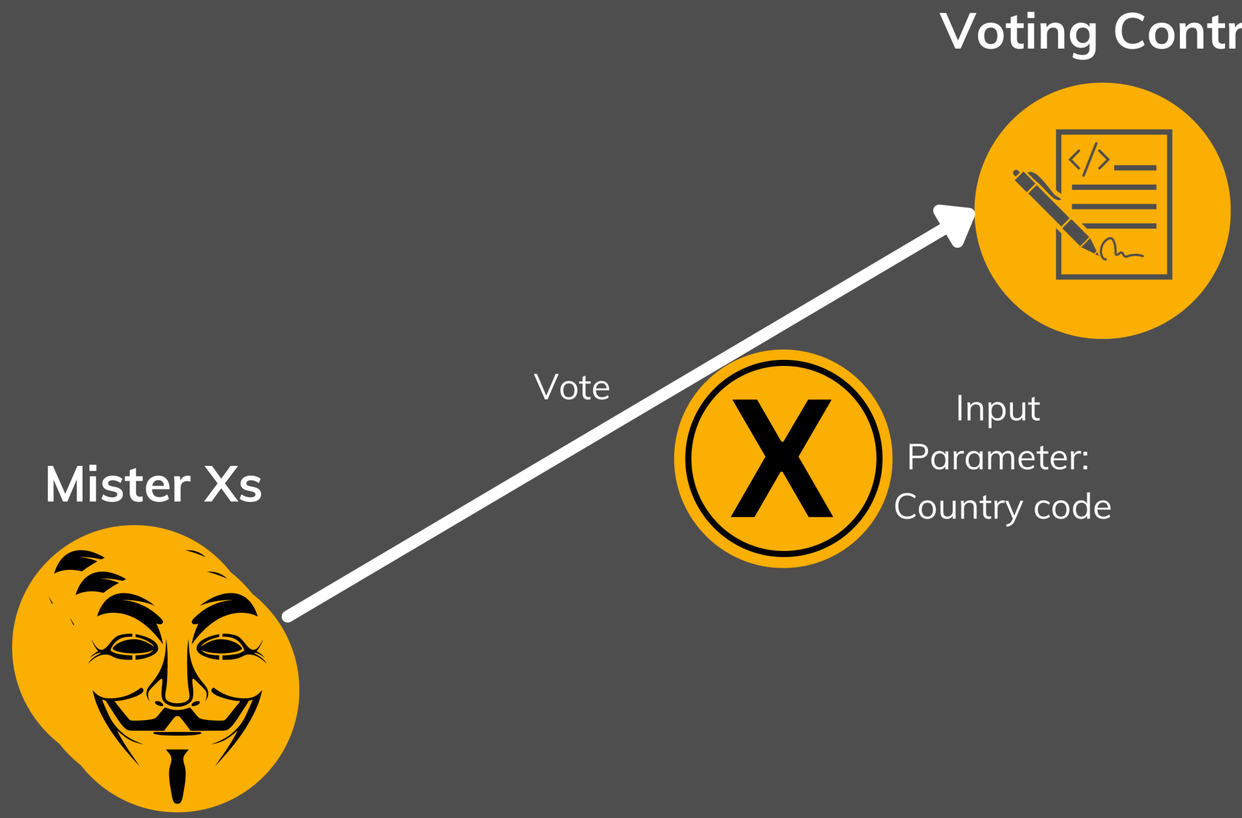
Identity Based Voting™

You

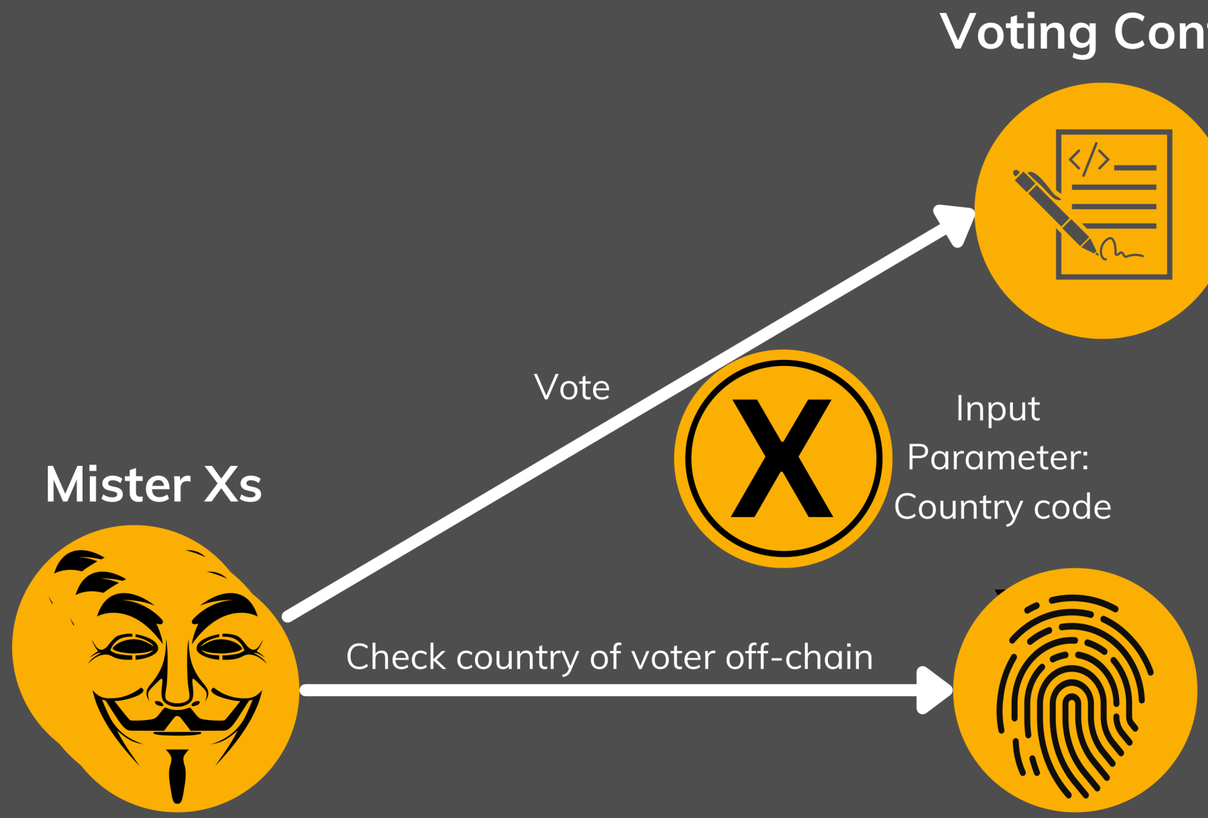
Identity Based Voting™



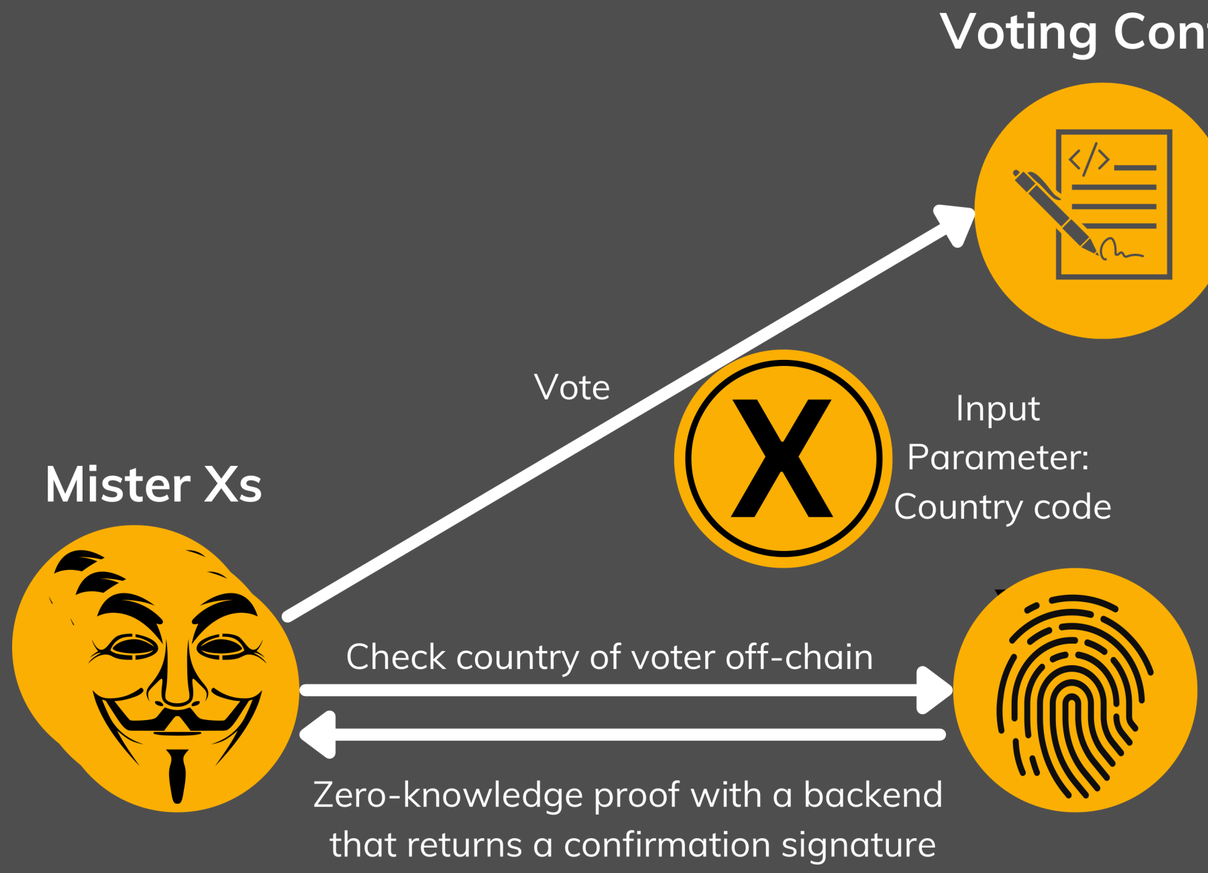
Identity Based Voting™



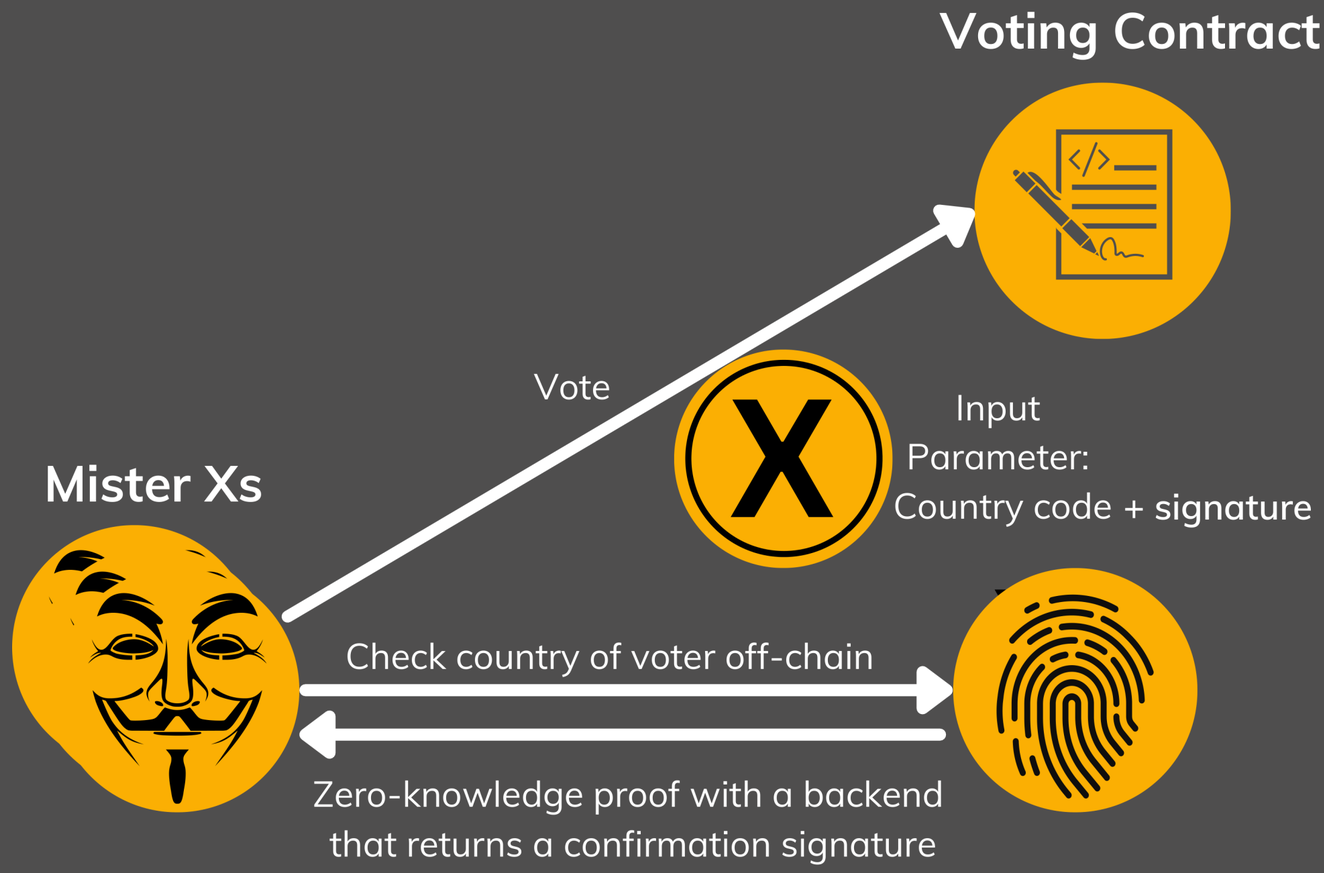
Identity Based Voting™



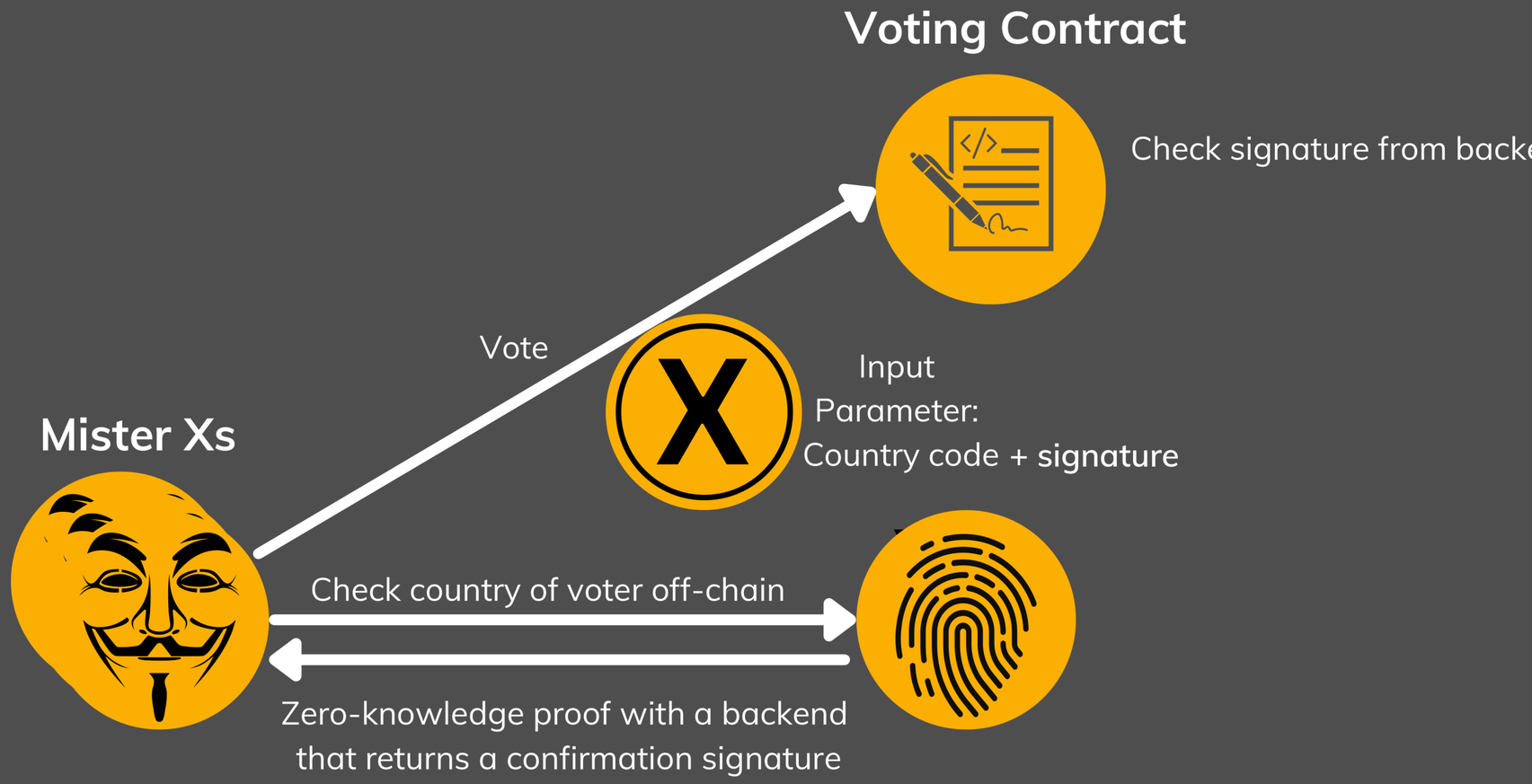
Identity Based Voting™



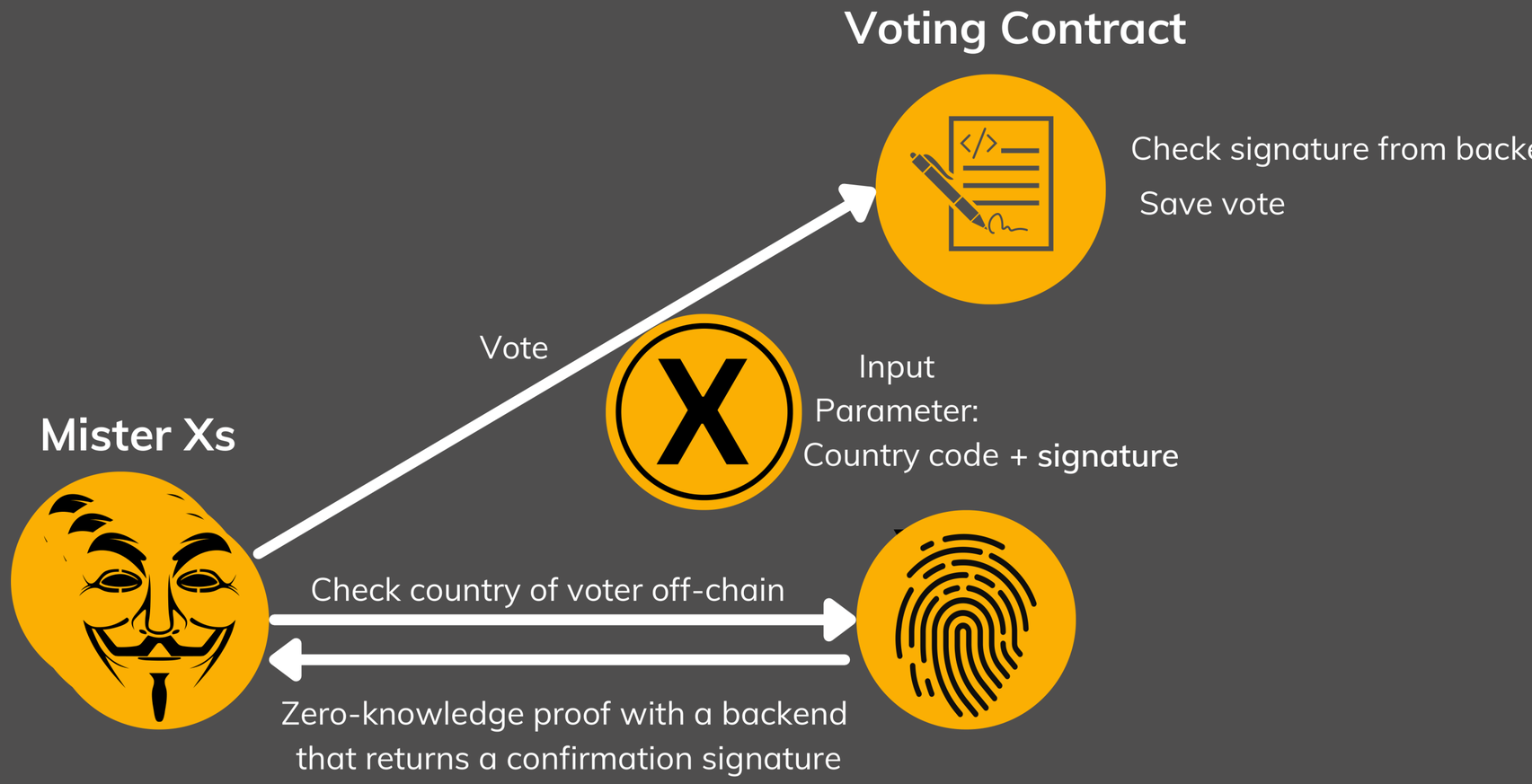
Identity Based Voting™



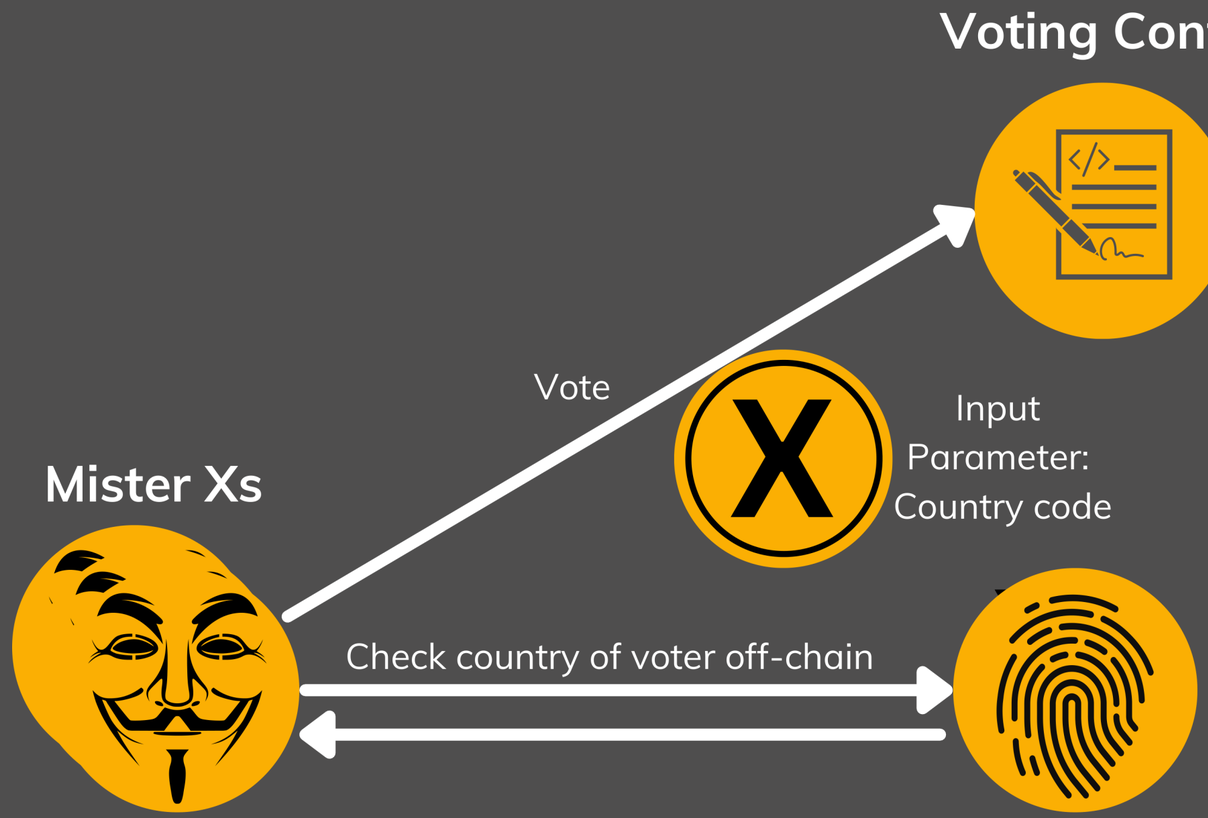
Identity Based Voting™



Identity Based Voting™



Identity Based Voting™



Not part of this tutorial

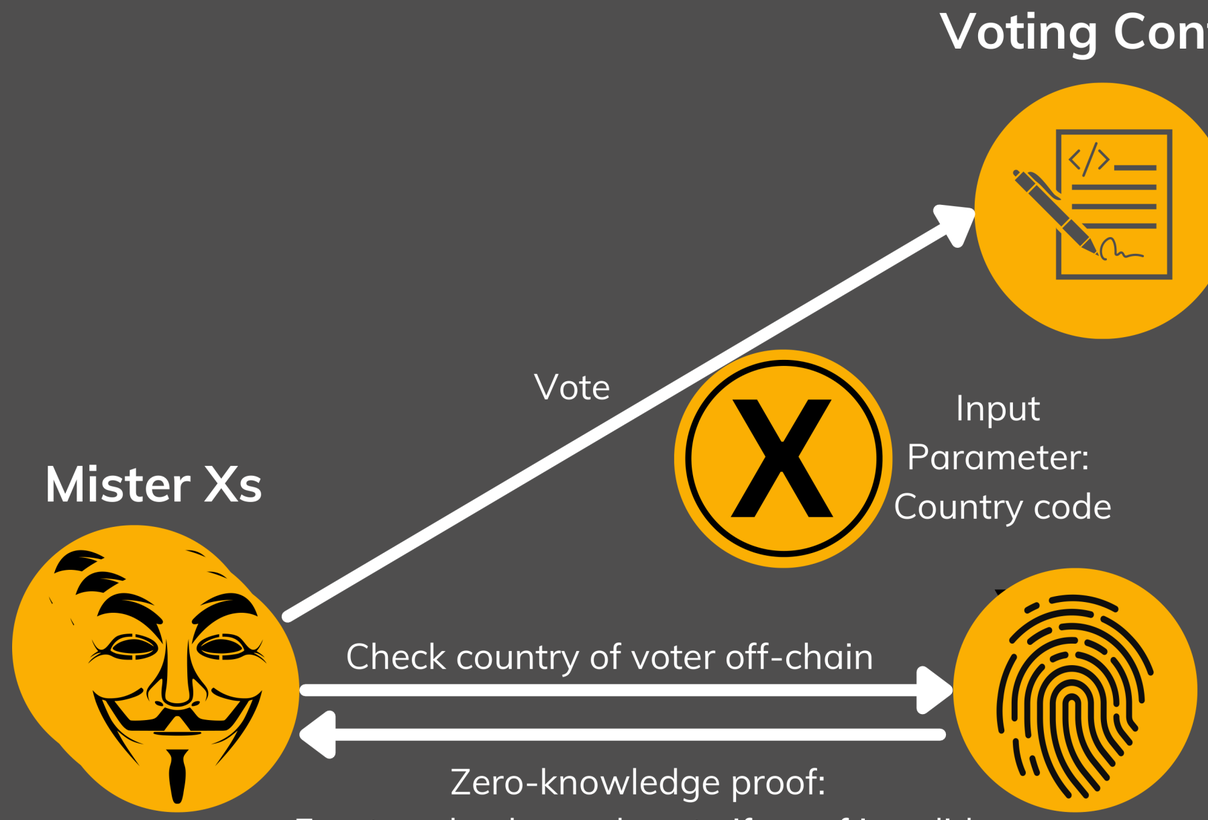
Check signature from backend

+ signature

Zero-knowledge proof with a backend that returns a confirmation signature

But the example repository has a full example of it!

Identity Based Voting™



Not part of this tutorial

Check signature from backend

+ signature

Zero-knowledge proof with a backend that returns a confirmation signature

But the example repository has a full example of it!

Installing prerequisites

Installing prerequisites

- Rust programming language
  - https://www.rust-lang.org/tools/install
- Concordium Client
  - https://developer.concordium.software/en/mainnet/net/installation/downloads-testnet.html
- Cargo Concordium
  - $ cargo install --locked cargo-concordium
- Cargo Generate
  - $ cargo install --locked cargo-generate
- Concordium Visual Studio Code Extension
  - https://marketplace.visualstudio.com/items?itemName=Concordium.concordium-smart-contracts
- Concordium Browser Wallet (shown tomorrow)
  - https://chrome.google.com/webstore/detail/concordium-wallet/mnnkpffndmickbiakofclnpoiajlegmg

Writing the Voting Smart Contract

Smart contracts onto the chain!

Additional Resources

- Our documentation:
  - https://developer.concordium.software/en/mainnet/net/guides/graviton-hackathon.html
- Repository for the smart contract:
  - https://github.com/Concordium/voting-workshop
- General support:
  - https://support.concordium.software/

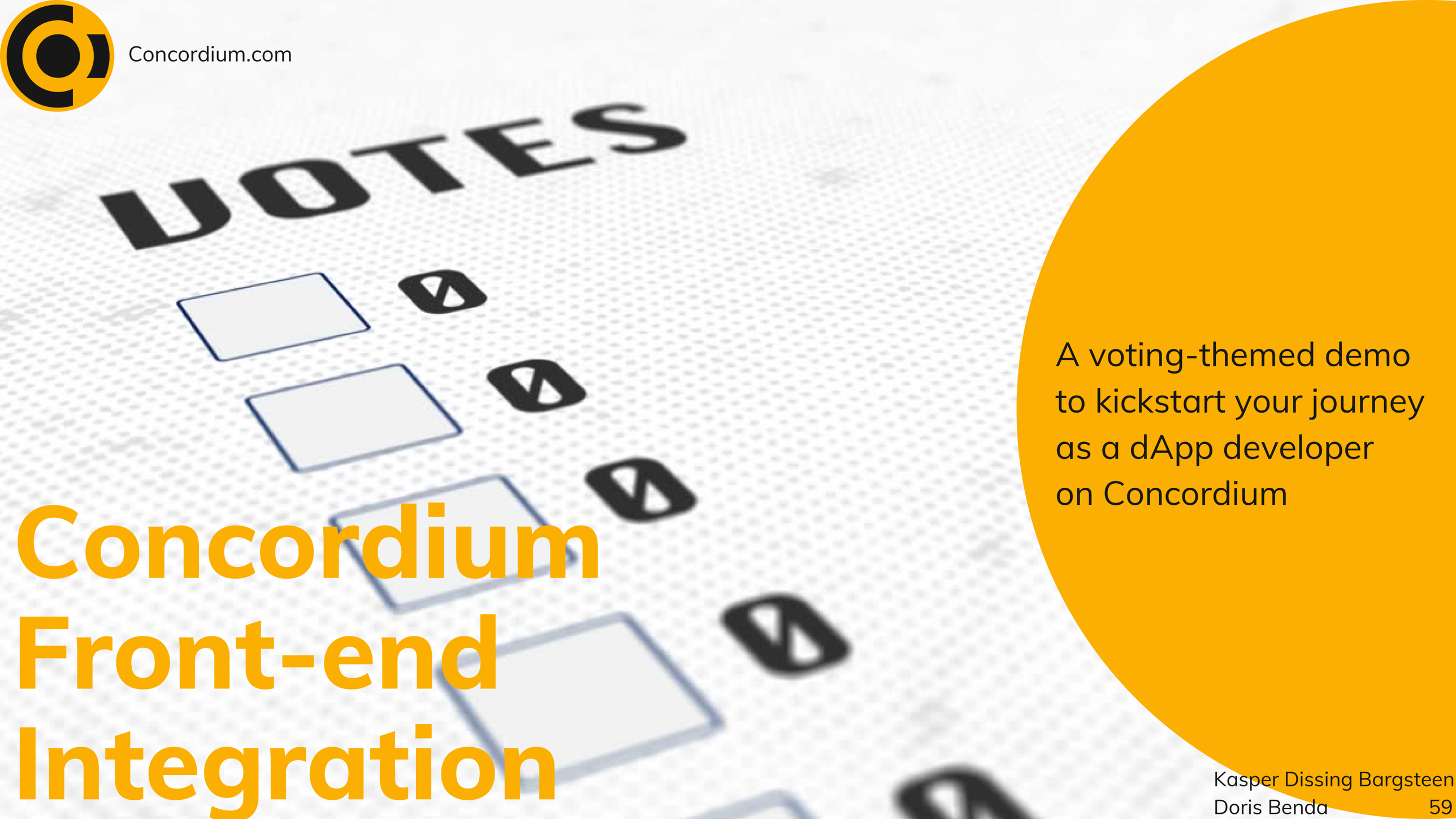
Additional Resources

- Testnet dashboard:
  - dashboard.testnet.concordium.com
  - testnet.ccdscan.io
- Public Concordium testnet node:
  - node.testnet.concordium.com
    port 10000 (GRPCv1) and port 20000 (gRPCv2 and gRPC-web)

Questions?

The end

Happy hacking & see you tomorrow!



Concordium Front-end Integration

A voting-themed demo to kickstart your journey as a dApp developer on Concordium

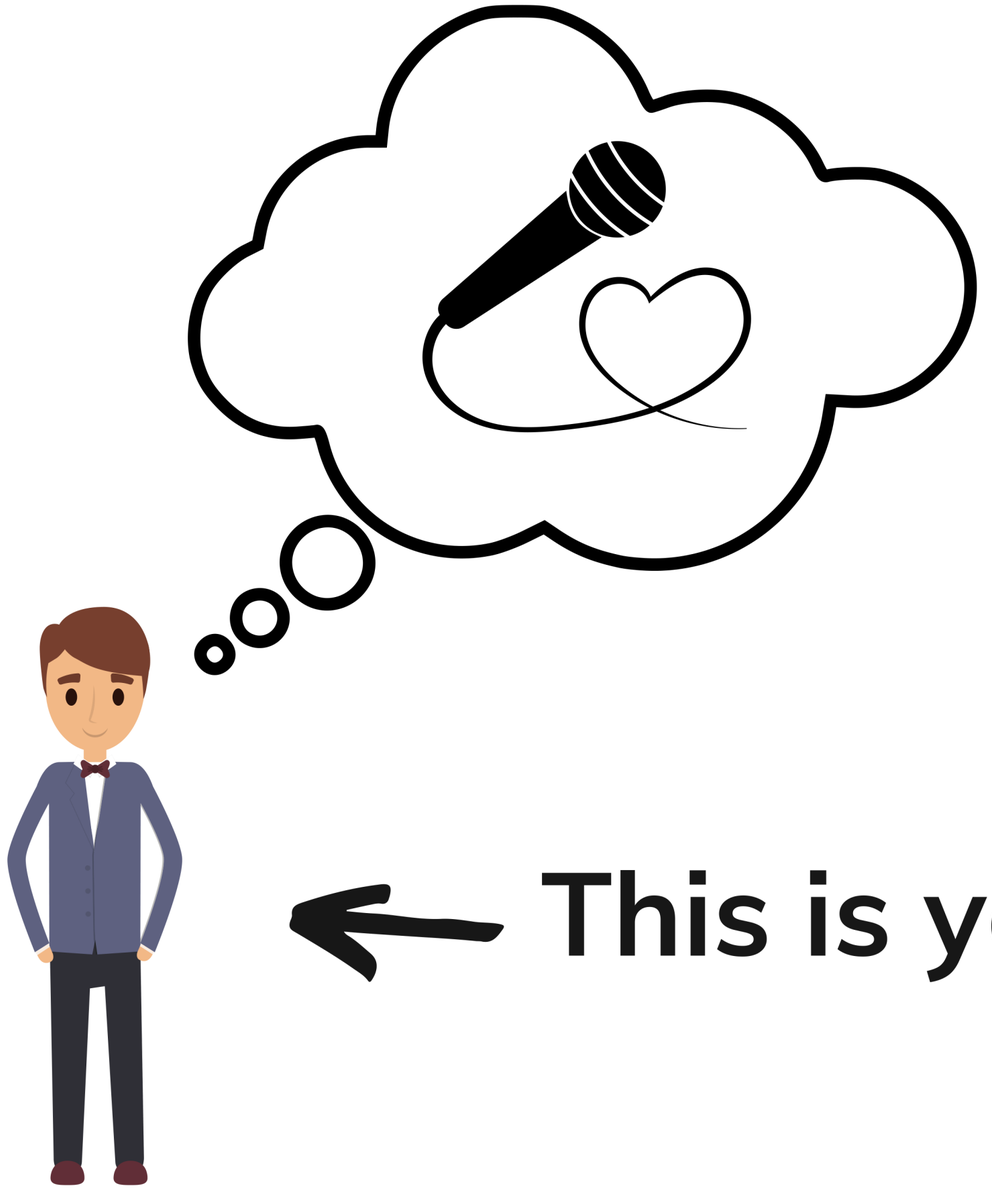
Kasper Dissing Bargsteen
Doris Benda

In this Presentation

Here's what we'll cover:

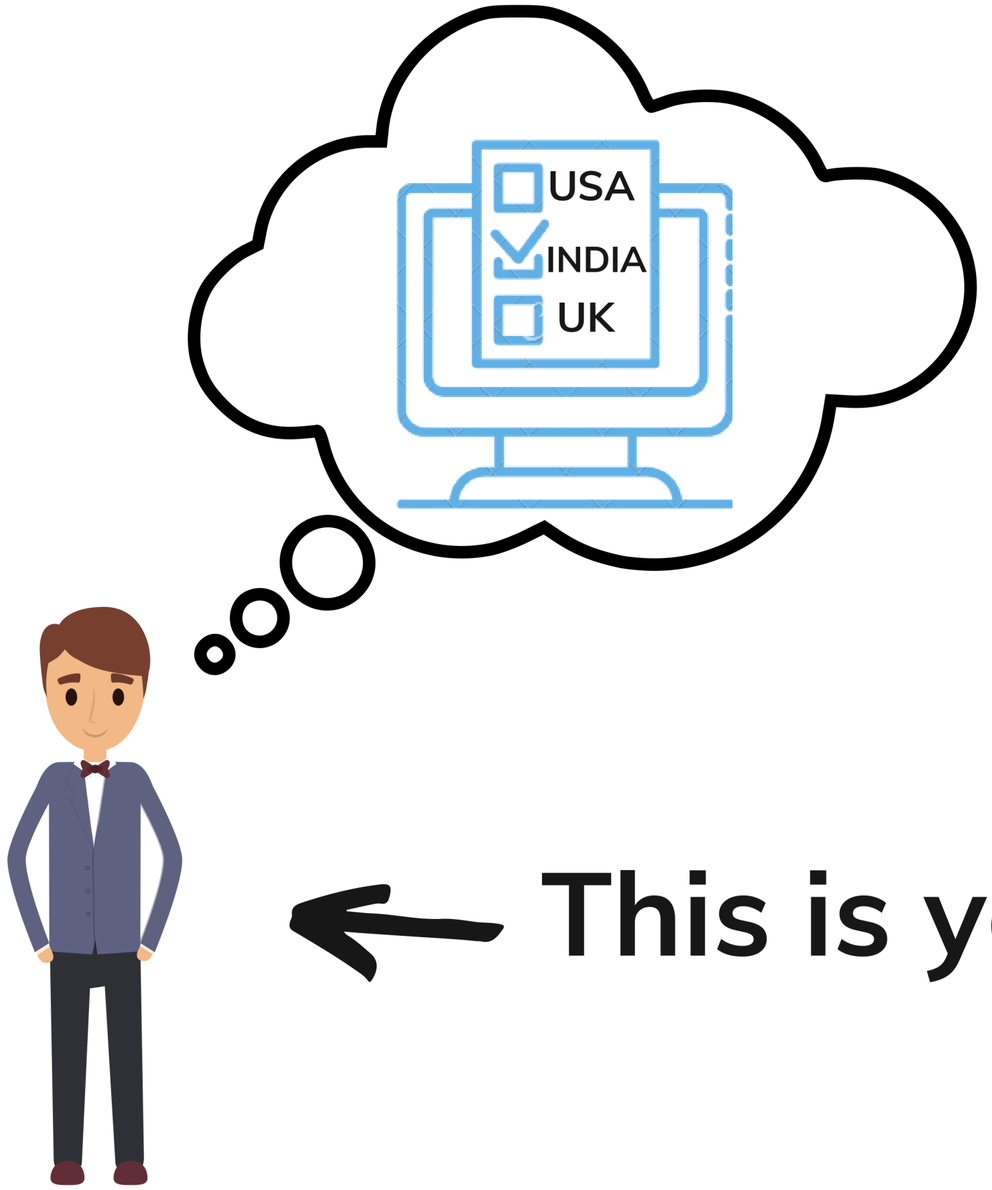
Recap of The Problem

← This is you..



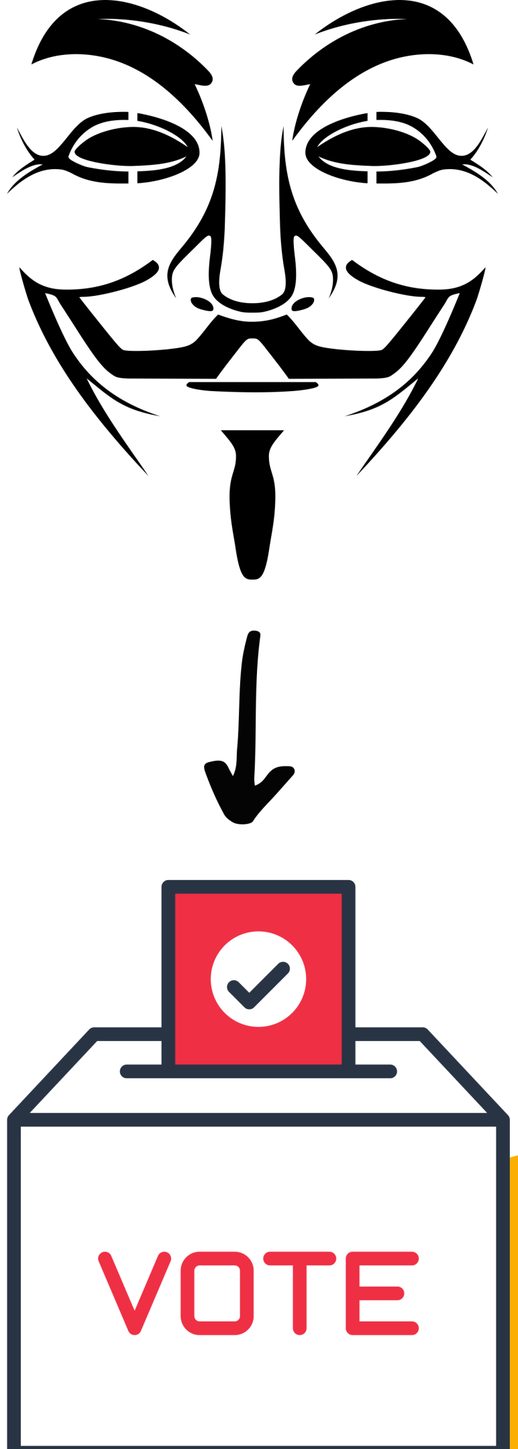
This is you..

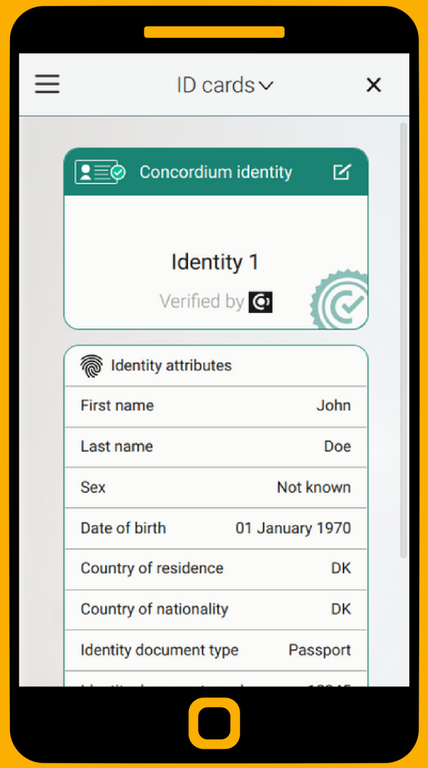
This is you..



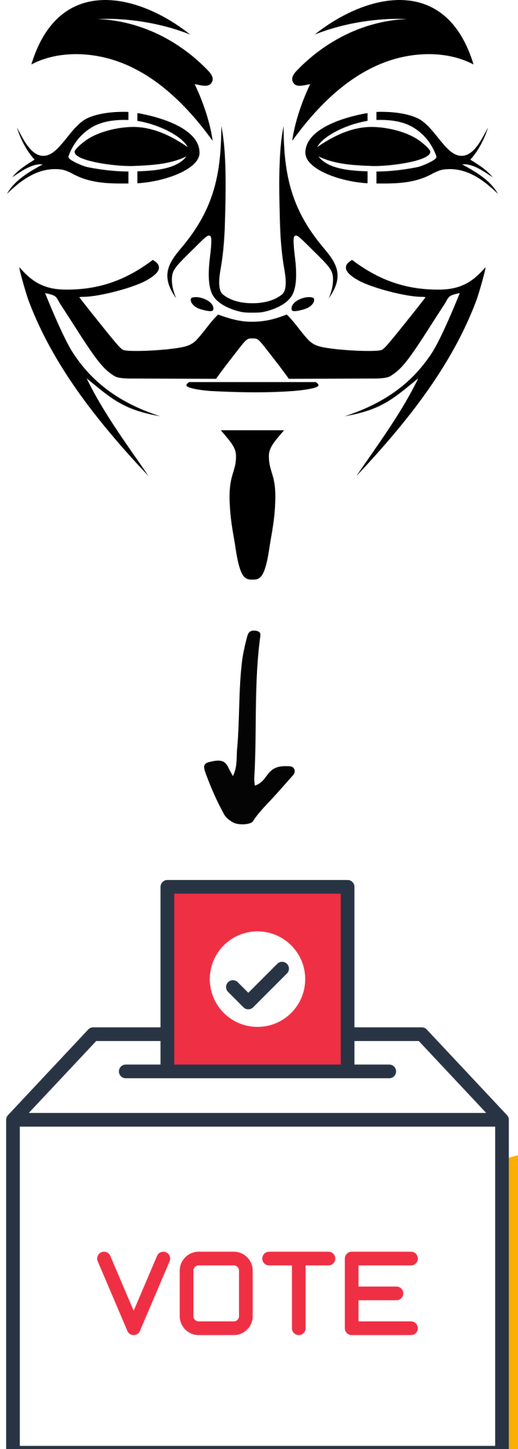
This is you..

Mister X should not be able to vote for their own country.





Concordium and its identity layer



Front-end Integration

Browser Wallet

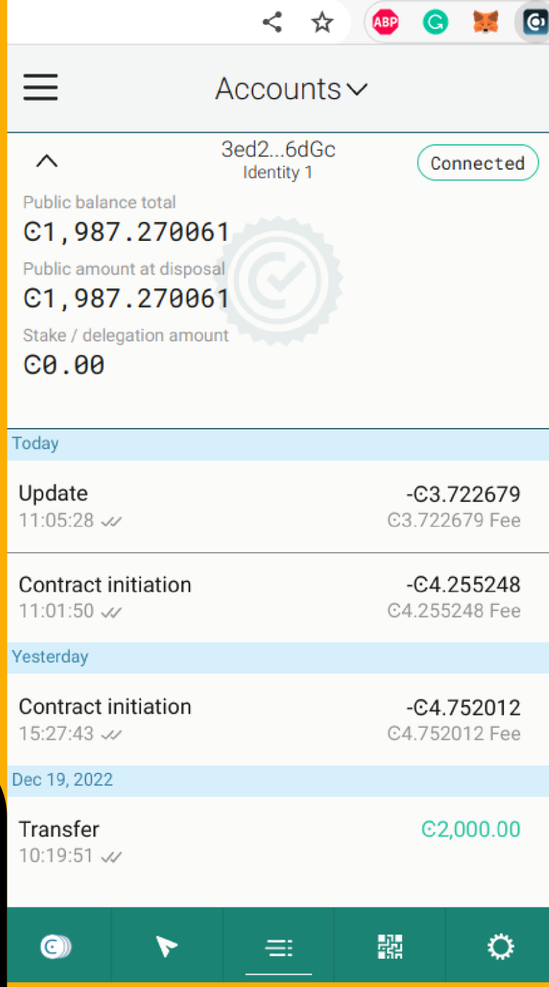
Concordium.com

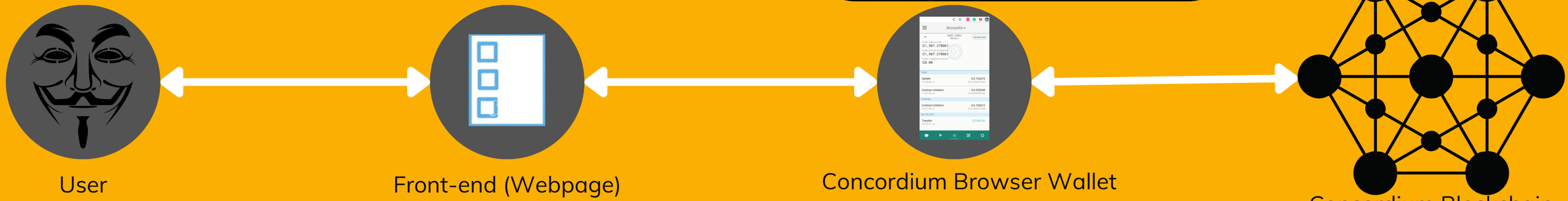
Concordium Browser Wallet

- A browser wallet is a piece of code that can be added as an extension to supported browsers such as Chrome.
- The browser wallet allows you to interact with the chain:
  - -> Reading data from the chain
  - -> Writing data to the chain (aka. sending transactions)
- The browser wallet hosts the private keys corresponding to the accounts of the user and a link that points to a server that hosts a Concordium node.

Let's download the testnet browser wallet:
https://developer.concordium.software/en/mainnet/net/installation/downloads-testnet.html#bw

Scan me!





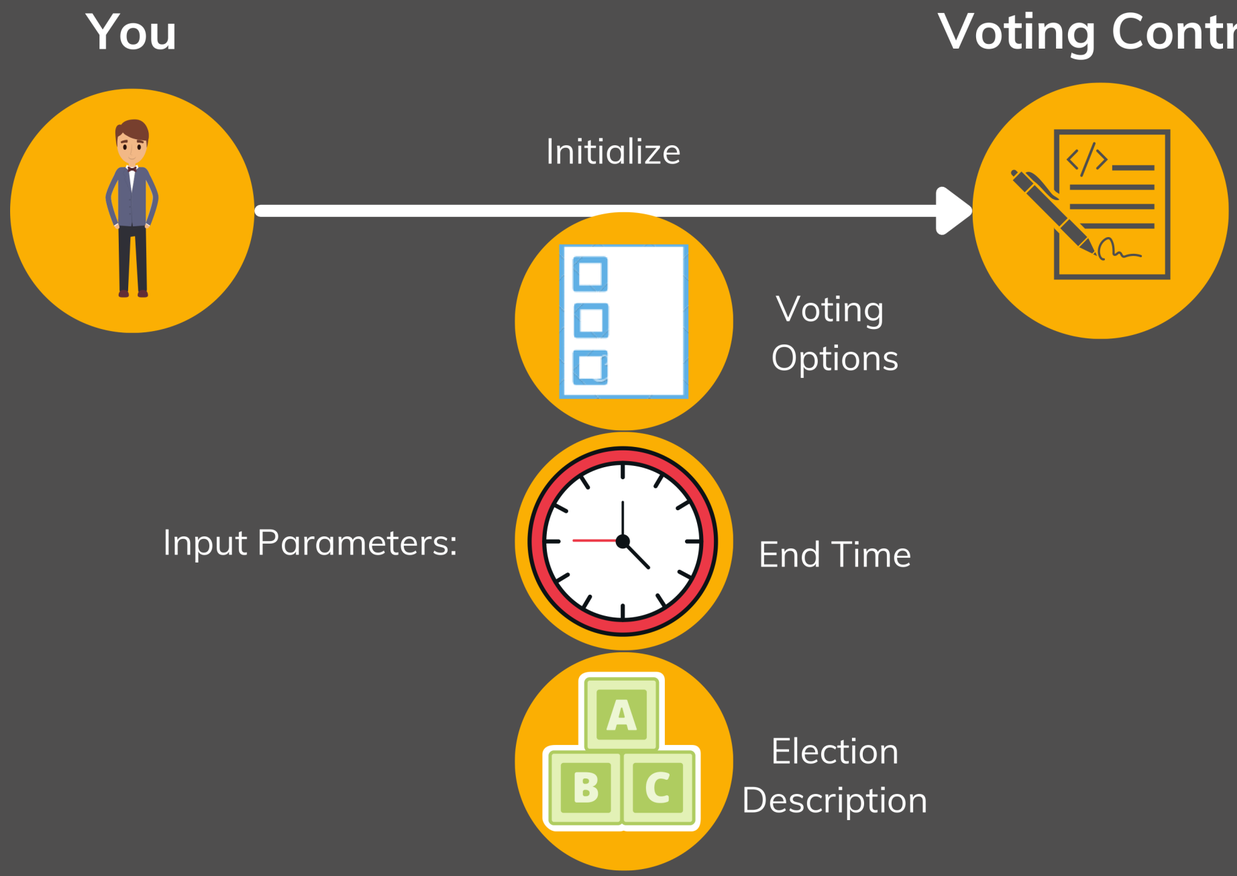
Initializing The Smart Contract

Identity Based Voting™

You

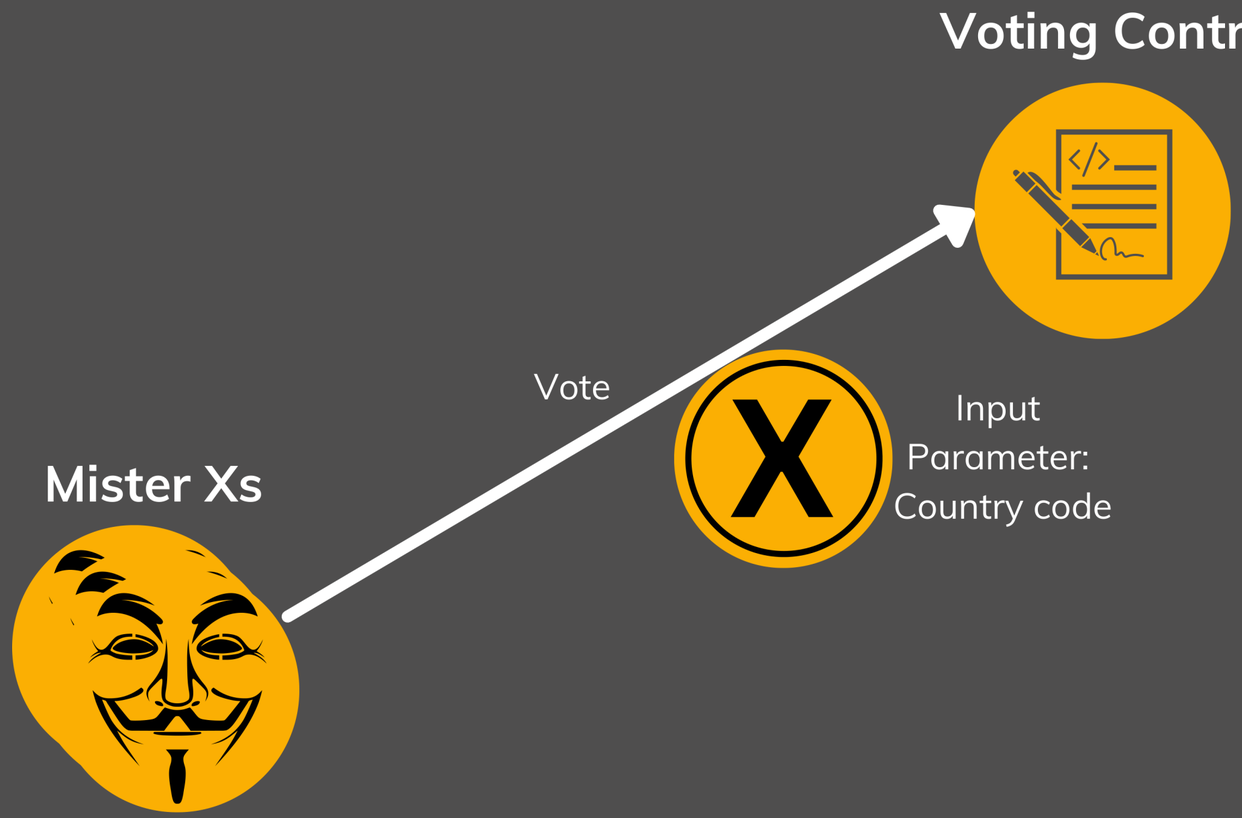
Concordium.com

Identity Based Voting™



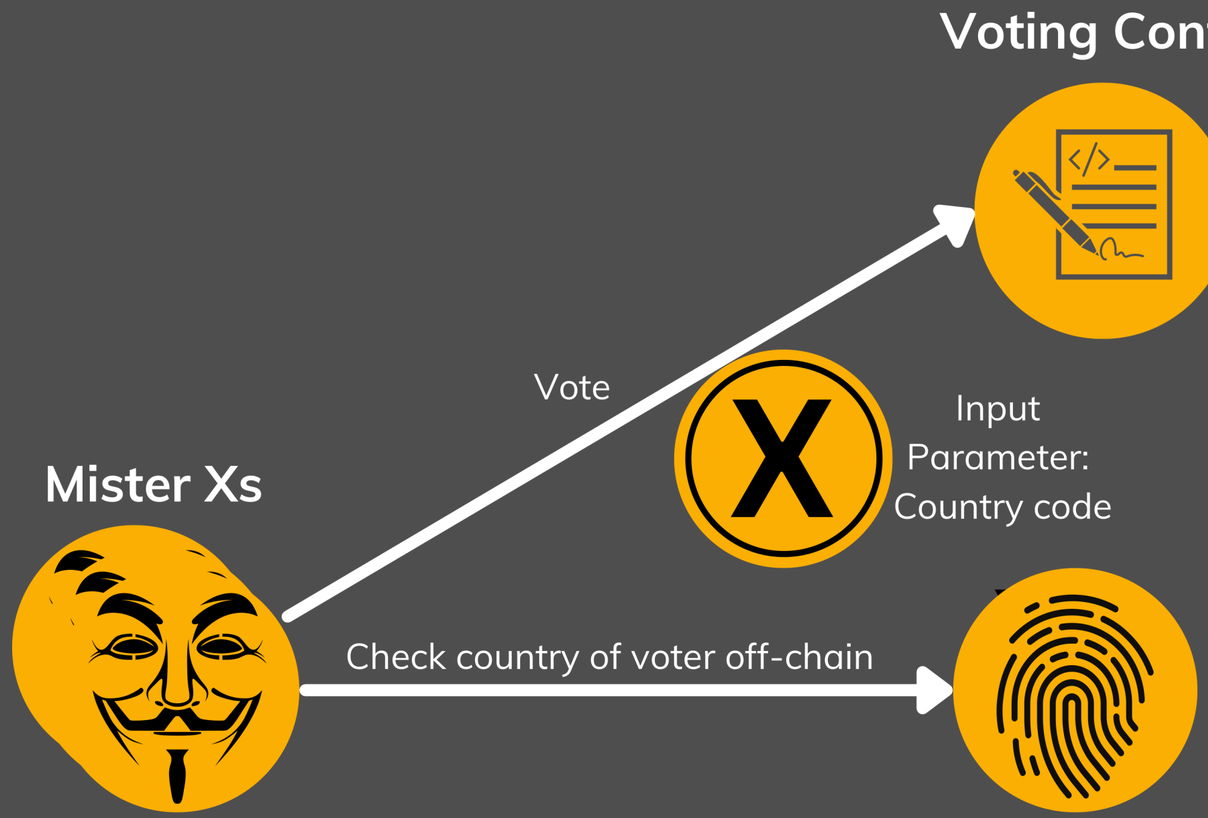
Voting

Identity Based Voting™

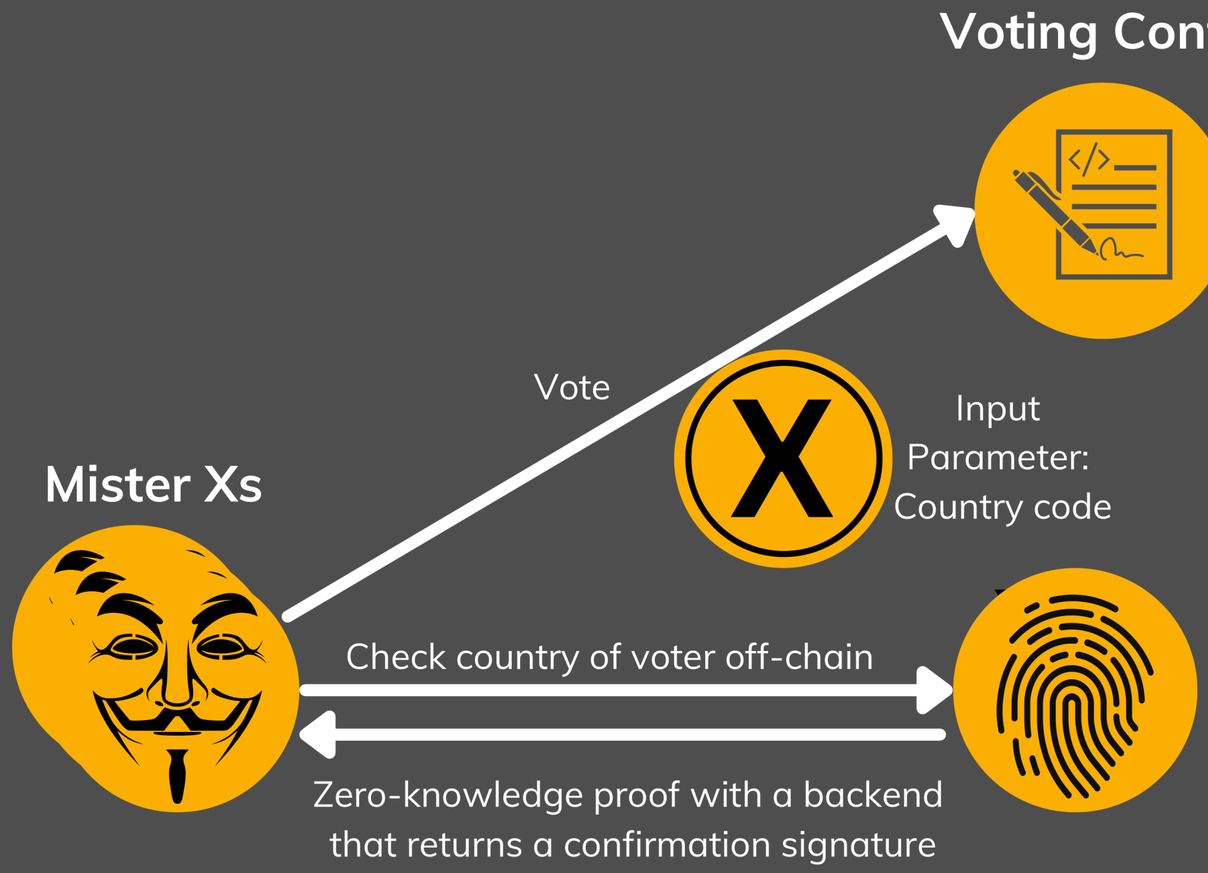


Identity-checked voting

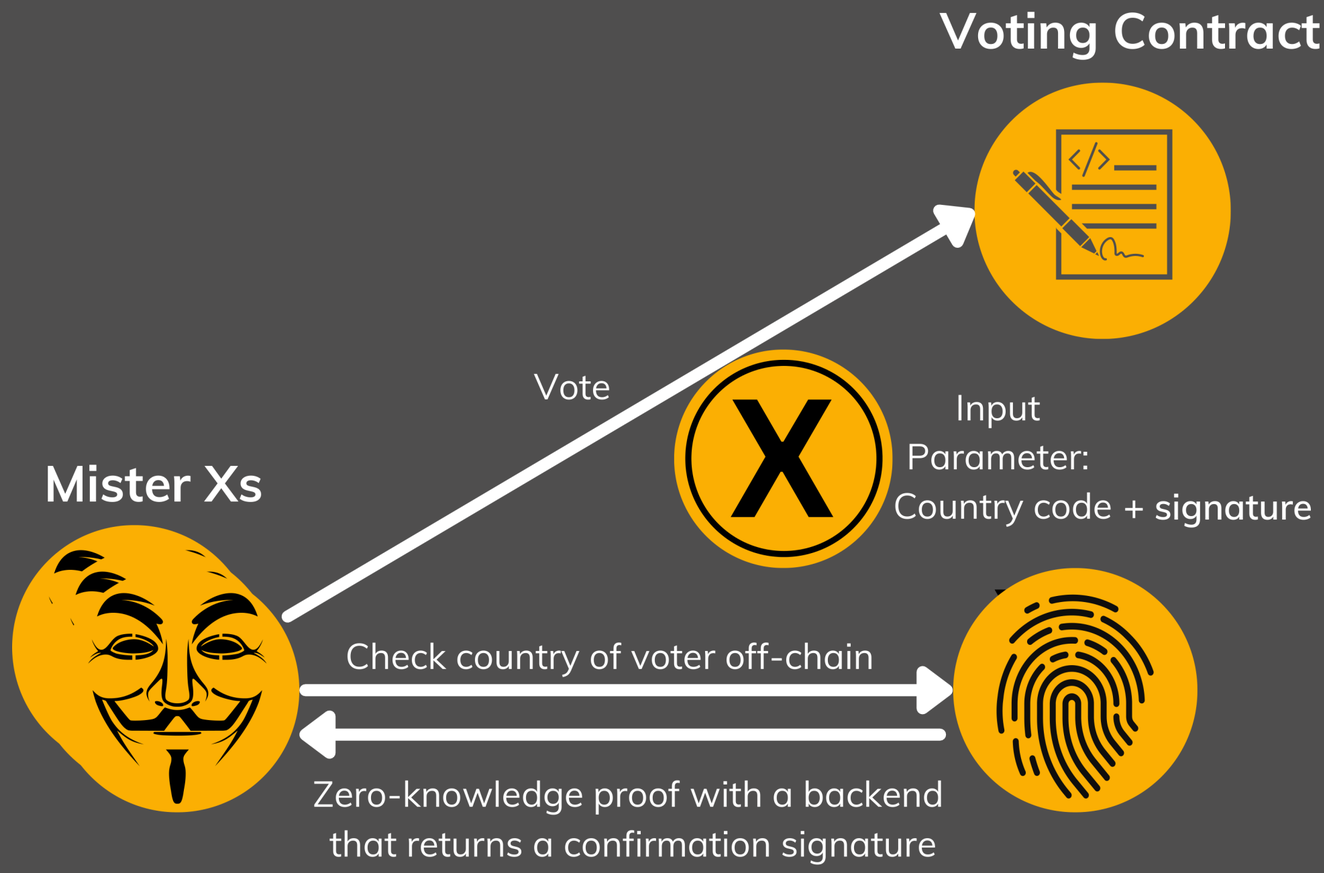
Identity Based Voting™



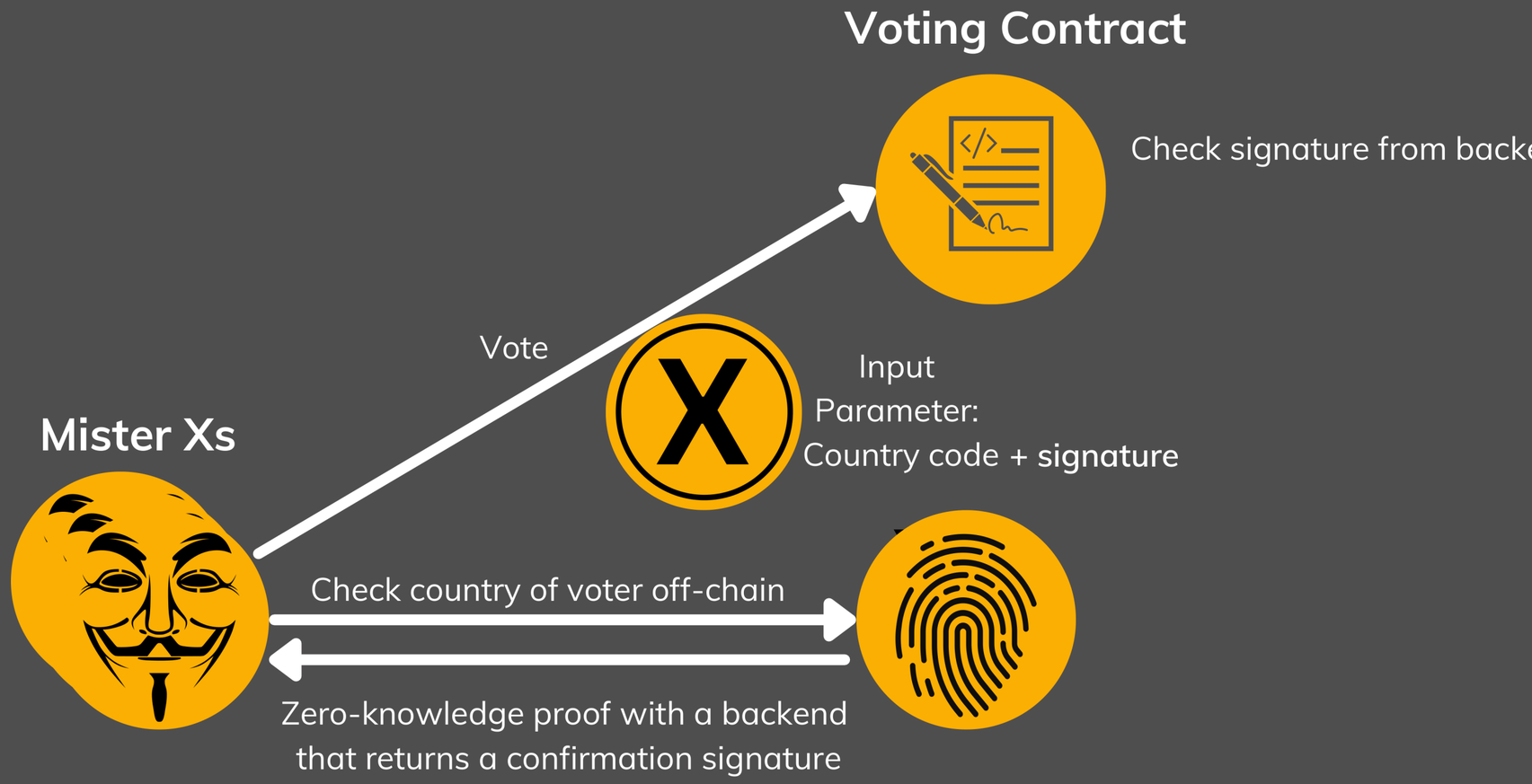
Identity Based Voting™



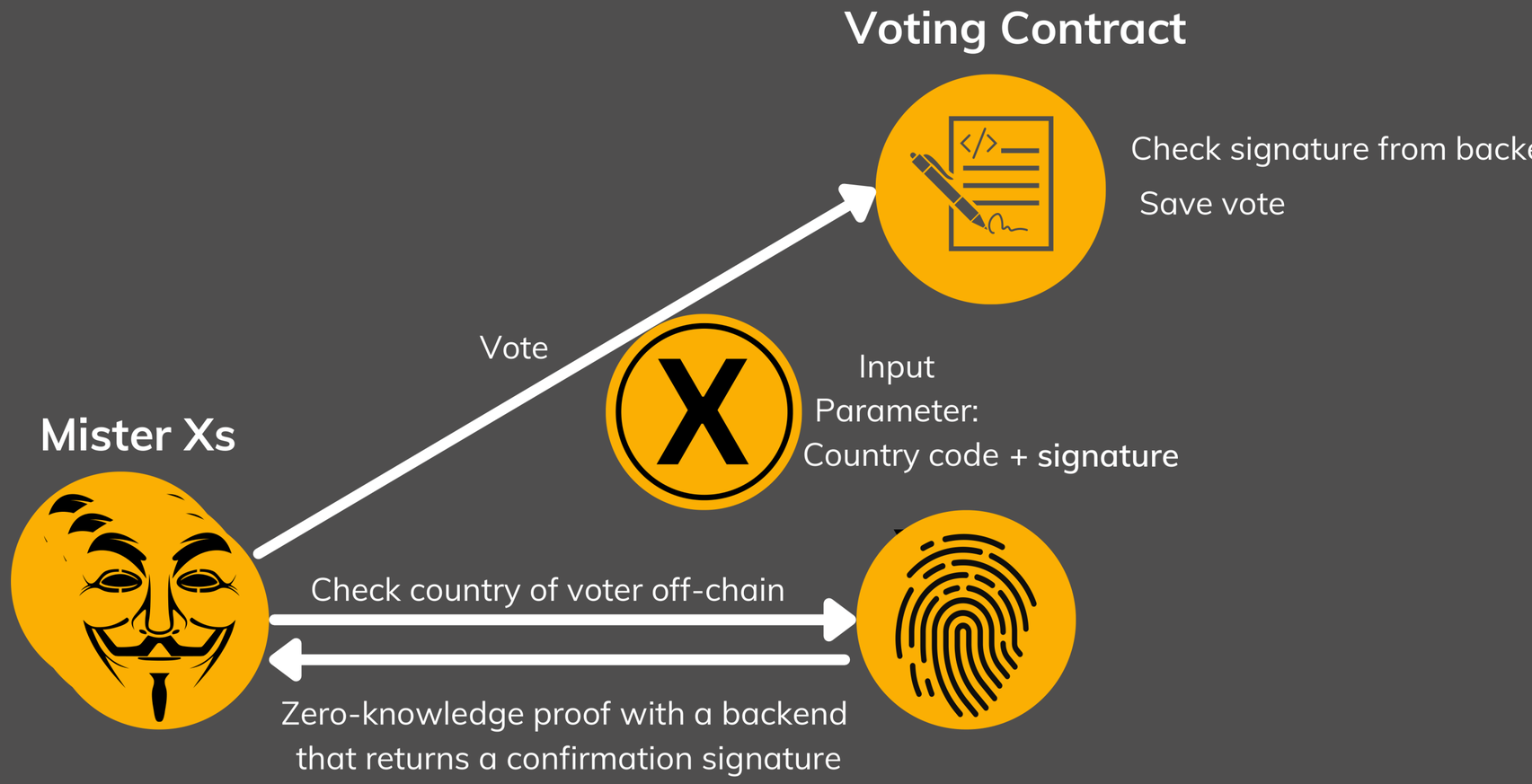
Identity Based Voting™



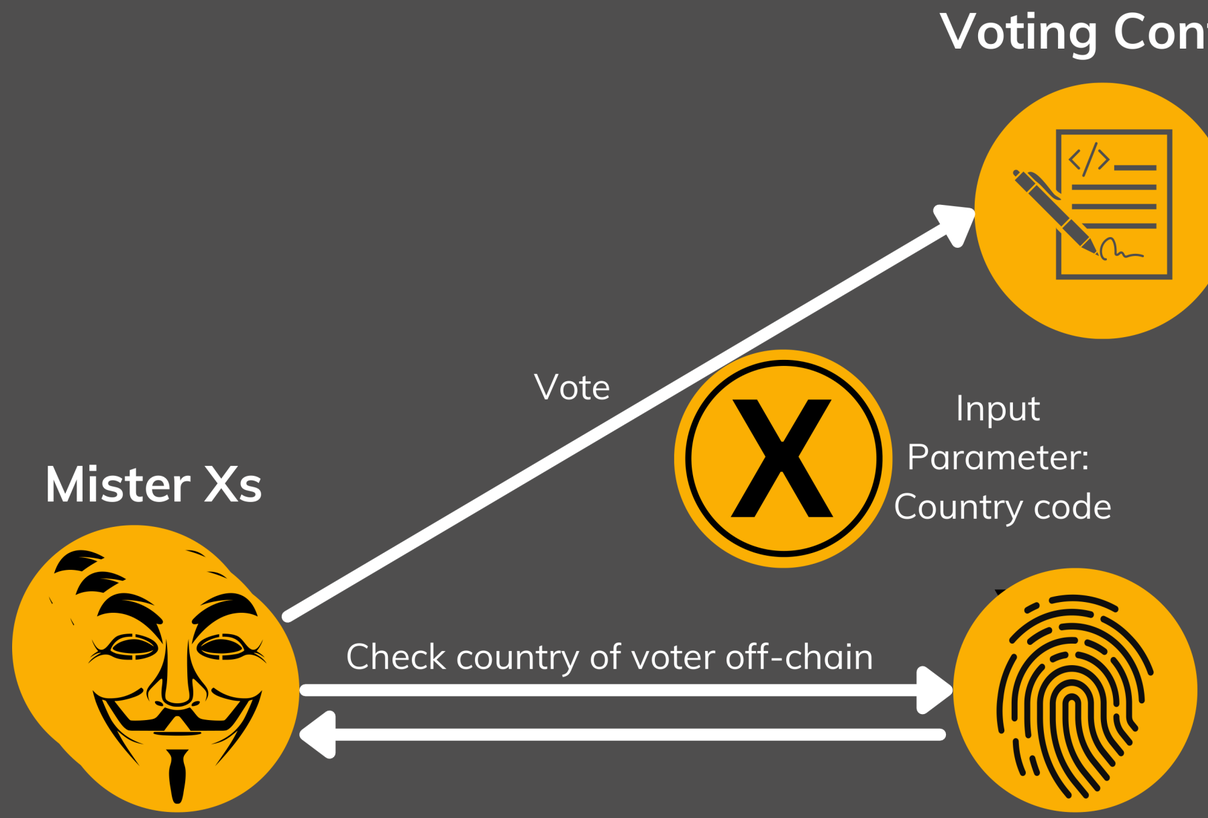
Identity Based Voting™



Identity Based Voting™



Identity Based Voting™



Not part of this tutorial

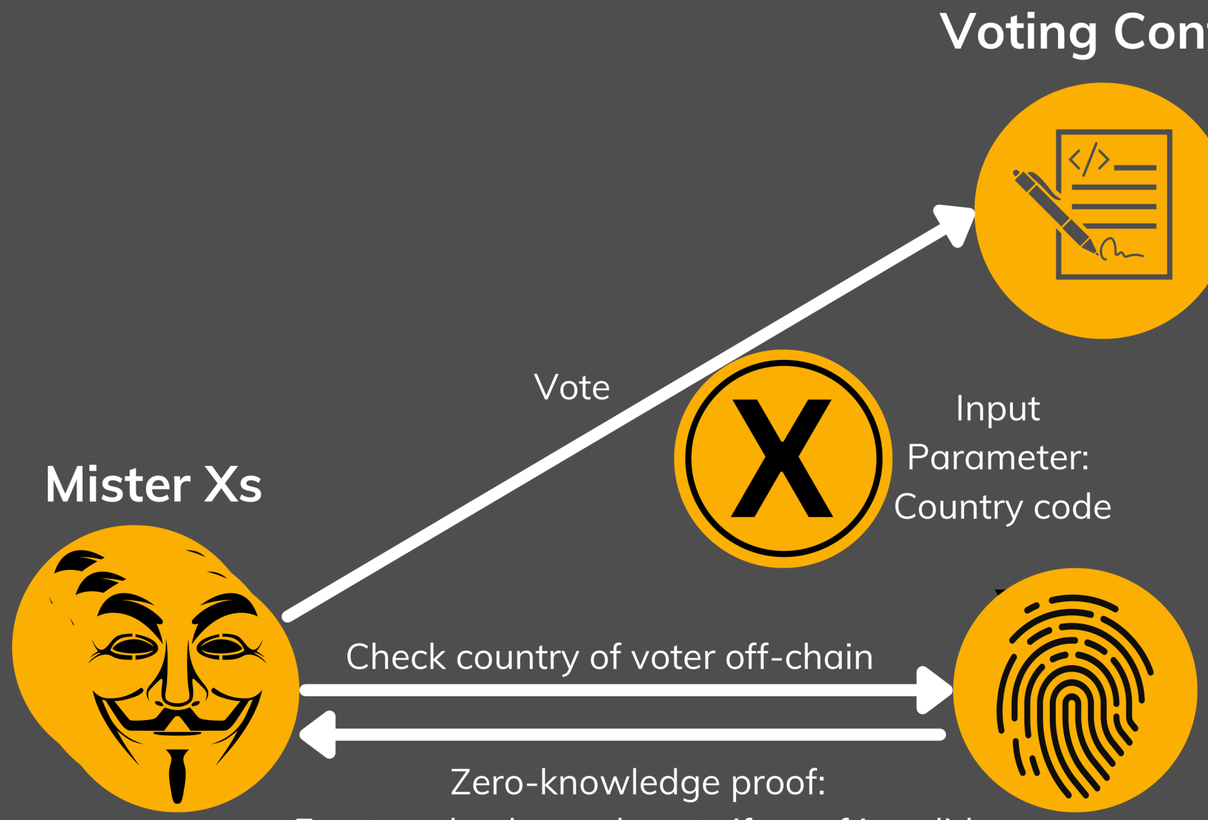
Check signature from backend

+ signature

Zero-knowledge proof with a backend that returns a confirmation signature

But the example repository has a full example of it!

Identity Based Voting™



Not part of this tutorial

Check signature from backend

+ signature

Zero-knowledge proof with a backend that returns a confirmation signature

But the example repository has a full example of it!

Why Building On Concordium

Blockchain + Identity =

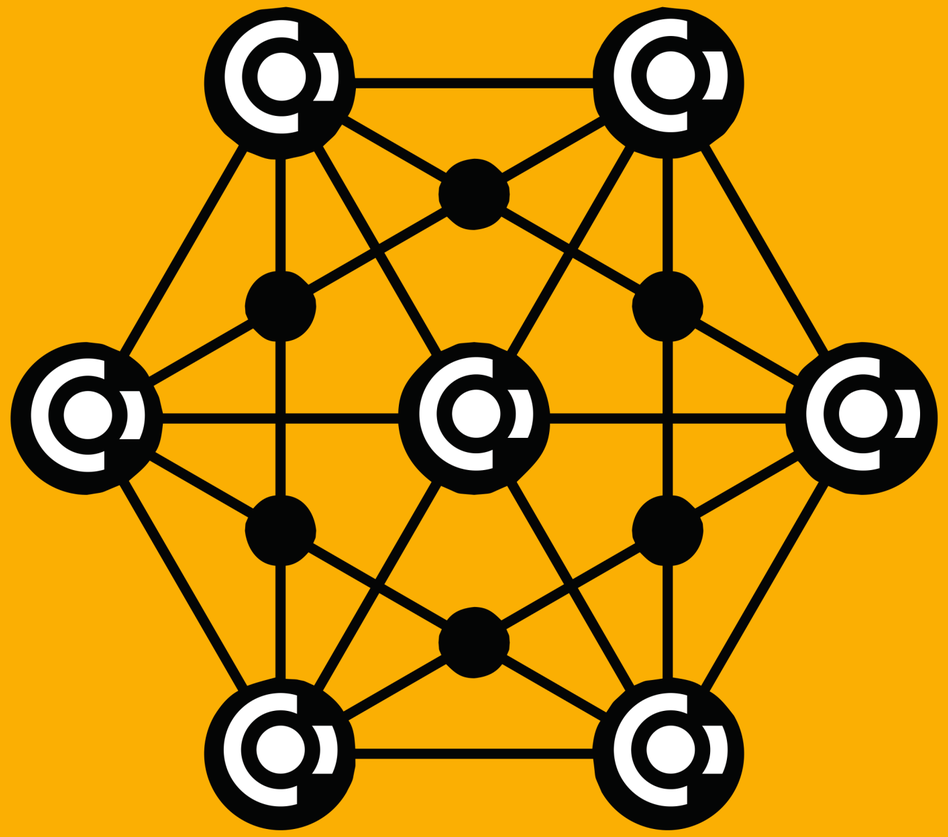
“Concordium is a science-based, Proof-of-stake, Public, Permissionless Blockchain with Identity built into the core protocol layer …”

Blockchain + Identity =

Blockchain: Stores transactions that cannot be deleted

Proof-of-stake: Eco-friendly. Weighted lottery tokenomics

Public: Anyone can run a node



Permissionless: No central control

Science-based: Built by scientists

Identity: Requires ID check to open Accounts

Blockchain + Identity =

- Unique and Innovative Identity Features:
  - Balance privacy with accountability through its ID layer
  - Zero-knowledge proofs
- Smart contract capabilities and the growing ecosystem of dApps and partners:
  - https://concordium.com/developer-ecosystem/
  - https://concordium.com/partners/
- Growing company - Always looking for new talent:
  - Great Tech Team (https://concordium.com/team/)
  - Explore our open positions or just apply (https://concordium.com/careers)
- Roadmap and company website:
  - https://concordium.com/

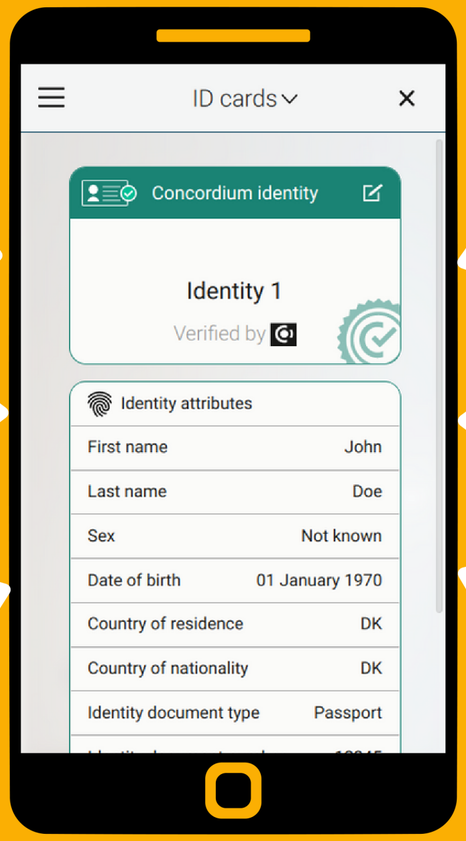
Concordium.com

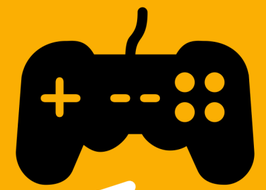
Ideation session

Frequent
Flyer Card

Employment Relation

Education
Certificates





Are you older
than 18?

Who are you? Really?

Are you a
participant of
this event?

Additional Resources

- **Developer documentation:**
  - https://developer.concordium.software/en/mainnet/net/guides/learn-about-concordium.html
- **Hackathon info:**
  - https://developer.concordium.software/en/mainnet/net/guides/graviton-hackathon.html
- **Front-end examples:**
  - https://github.com/Concordium/concordium-browser-wallet/tree/main/examples
  - https://github.com/Concordium/concordium-dapp-examples
- **Hosted front-ends:**
  - https://piggybank.testnet.concordium.com/
  - https://wccd.testnet.concordium.com/
  - https://voting.testnet.concordium.com
  - https://gallery.testnet.concordium.com/
  - http://signmessage.testnet.concordium.com/

Additional Resources

- **Node-SDK:**
  - https://github.com/Concordium/concordium-node-sdk-js
  - https://github.com/Concordium/concordium-node-sdk-js/tree/main/packages/common
- **Tutorials:**
  - https://developer.concordium.software/en/mainnet/smart-contracts/tutorials/index.html
- **Concordium Academy:**
  - https://academy.concordium.software/
- **This presentation with code examples:**
  - https://github.com/Concordium/voting-workshop
- **General software support:**
  - https://support.concordium.software/

Full dApps

http://developer.concordium.software/en/mainnet/net/guides/dapp-examples.html

Questions?

Good Luck For The Hackathon

Reach Out To Us If You Have Questions