STARKWARE

Camp 1: Getting Started

StarkNet EDU

@starknet_edu

January 2023

Programming Languages

Imperative

- C++, Java, Solidity, etc.

Functional

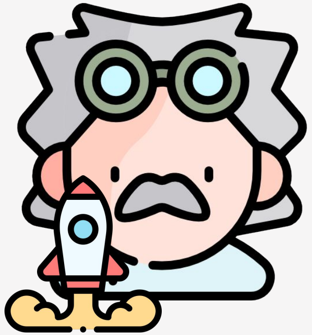
- Haskell, Lisp, Vyper, etc.

Provable

- Cairo

STARKWARE

The Space Exploration Problem



How to save fuel?

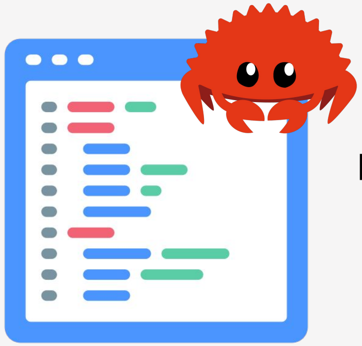
Best launch window

Best trajectory

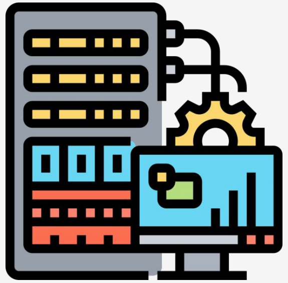
Engineers create algorithm in Rust

STARKWARE

Cooperating with a Rival



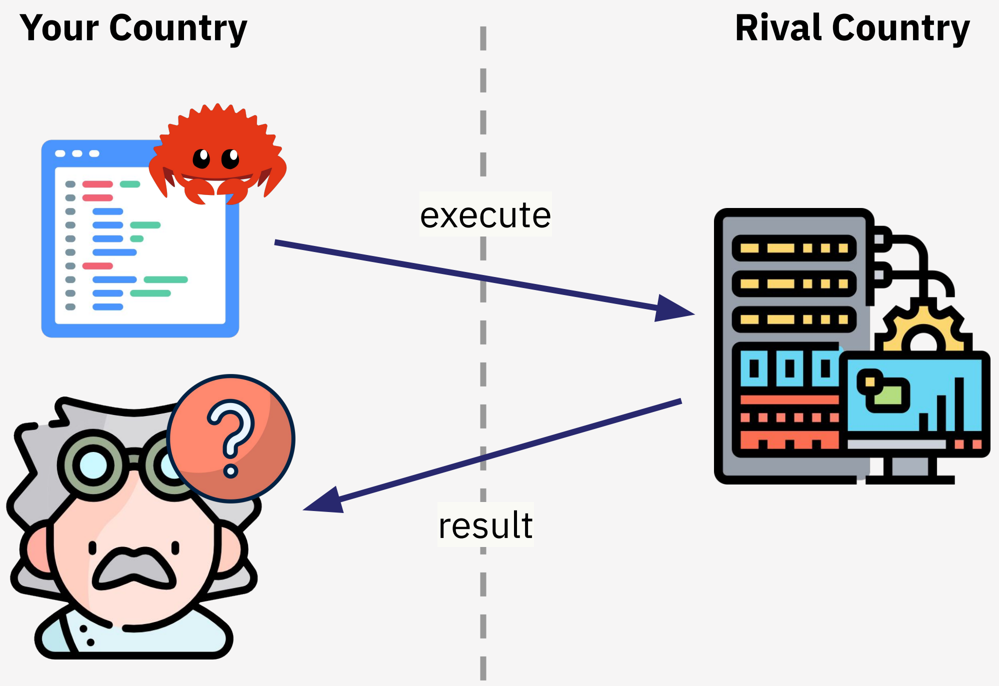
Execution needs supercomputer



Only rival country has one

STARKWARE

To Trust or not to Trust?



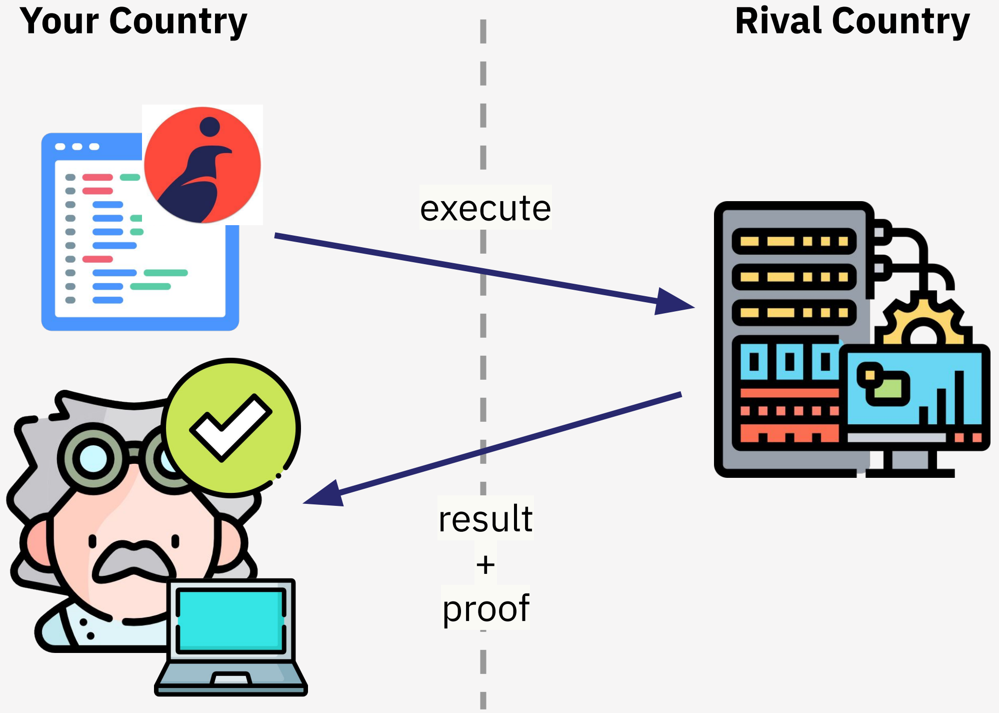
How do you know if...?

Supercomputer malfunction

Rival spy agency sabotage

YOU CAN'T

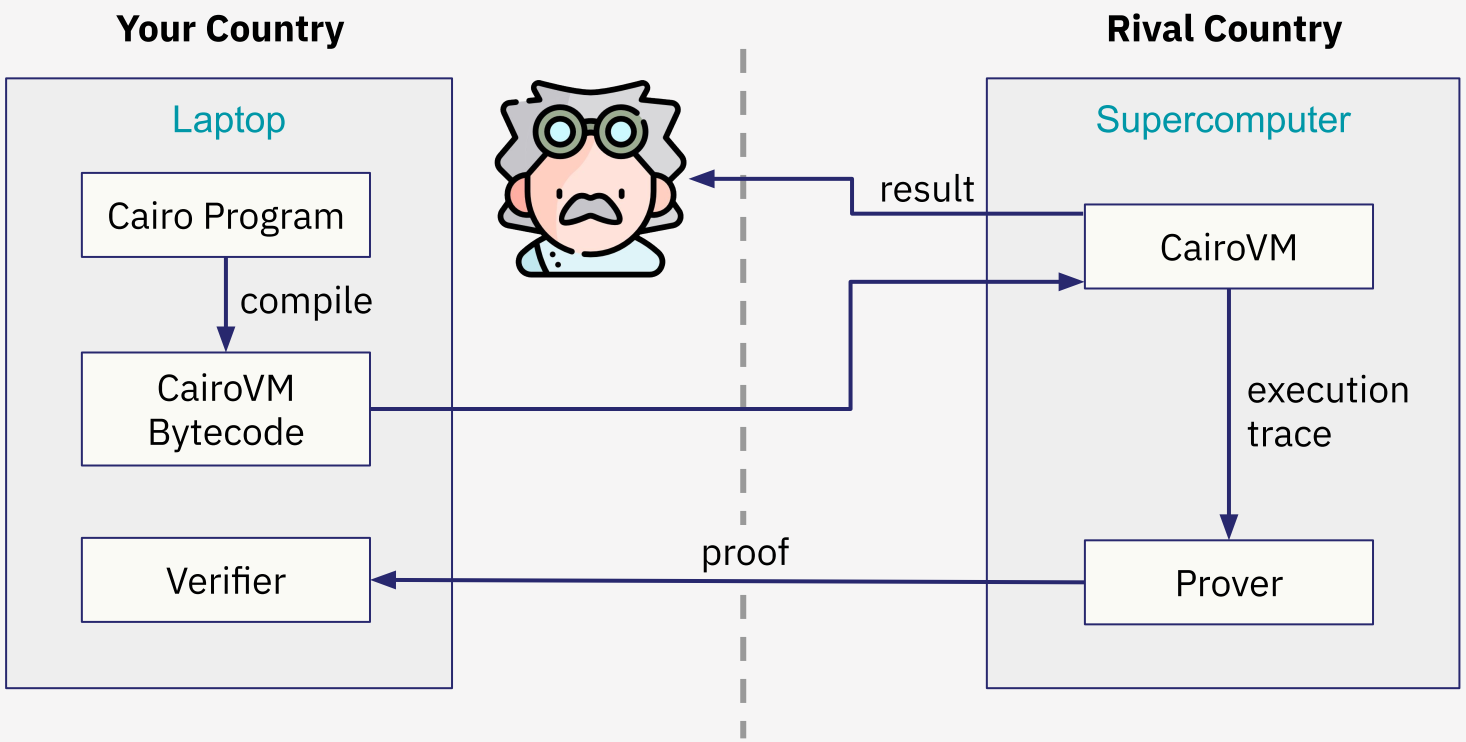
Trustless Cooperation



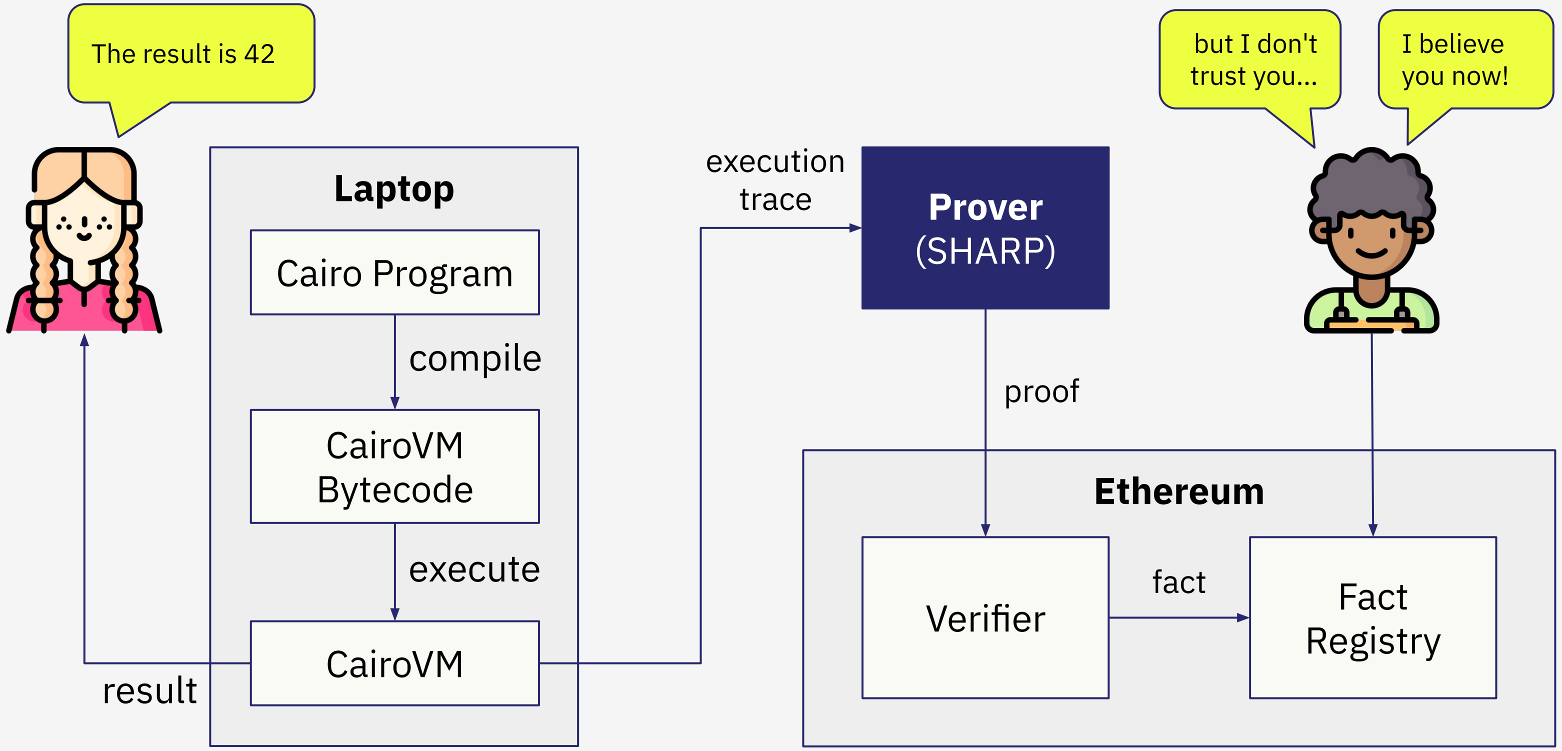
If the execution is **intentionally** or **unintentionally** modified, the proof will be **invalid**

A regular computer is able to keep a supercomputer **honest**

Trustless Cooperation



Convincing Strangers



Cairo 0.10 First Look

```
%builtins output

from starkware.cairo.common.serialize import serialize_word

func sum_two_nums(num1: felt, num2: felt) -> (sum: felt) {
   alloc_locals;
   local sum = num1 + num2;
   return (sum=sum);
}

func main{output_ptr: felt*}() {
   alloc_locals;

   const NUM1 = 1;
   const NUM2 = 10;

   let (sum) = sum_two_nums(num1=NUM1, num2=NUM2);
   serialize_word(sum);
   return ();
}
```

builtins pass special pointers to **main**

import works like in **python**

felt (field element) is Cairo's **native** type

variables are defined as **let**, **local** or **tempvar**

functions have **regular** and **implicit** arguments

implicit arguments are always passed to internal functions

Executing the First Cairo Program

STARKWARE

Goals

1) Create a dev environment

2) Write Cairo program

3) Submit program to Prover

4) Check validity of proof

Useful Links

Python dev environment article

Fact Registry on Ethereum Goerli

Cairo CLI Cheat Sheet

```
# compile
cairo-compile src/example.cairo --output build/example.json

# run & print output
cairo-run --program build/example.json --print_output --layout=small

# debug memory
cairo-run --program build/example.json --print_memory --relocate_prints

# debug memory of unbound program
cairo-run --program build/example.json --print_memory --relocate_prints --no_end --steps 16

# compile, run and submit to sharp
cairo-sharp submit --source src/example.cairo

# sharp job status
cairo-sharp status <job-key>

# proof validity status
cairo-sharp is_verified <fact> --node_url <eth-rpc-url>
```

To Speed Up Time...

Job Key: 70f88a8b-6261-44b0-b3b1-e52ec84e55fe

Fact: 0x8f6b78593719c8e46080237f2a6338e6c7f651c39be788a8029f6f7c713feb45

Summary

STARKWARE

Cairo Fundamentals

Comments

```
%builtins output

// single line comment
from starkware.cairo.common.serialize import serialize_word

// multiple
// line comment
func sum_two_nums(num1: felt, num2: felt) -> (sum: felt) {
   alloc_locals;
   local sum = num1 + num2;
   return (sum=sum); // in-line comment
}
...
```

There **isn't** a special symbol for multi line comment

Functions

STARKWARE

```
%builtins output
from starkware.cairo.common.serialize import serialize_word

func main{output_ptr: felt*}() {
    alloc_locals;
    let var1 = return_value();
    let (var2, var3) = return_tuple();
    let (local var4) = return_named_tuple();
    return ();
}
```

Avoids issue with revoked references (`alloc_locals;`)

Returned value can be turned into **local** (`let (local var4) = return_named_tuple();`)

```
func return_value() -> felt {
    return 5;
}
```

```
func return_tuple() -> (felt, felt) {
    return (10, 15);
}
```

```
func return_named_tuple() -> (res: felt) {
    return (res=20);
}
```

STARKWARE

Felt (field element)

Default data type of Cairo. Behaves like an integer.

```
%builtins output
from starkware.cairo.common.serialize import serialize_word

func main{output_ptr: felt*}() {
   alloc_locals;
   local foo: felt = 5;
   local bar = 10;
   local baz = 'Hello there';

   serialize_word(foo); // 5
   serialize_word(bar); // 10
   serialize_word(baz); // 87521618088882658227876453
   return ();
}
```

Explicitly defined as felt

Implicitly defined as felt

Also a felt, not a <u>real</u> string. **Max 31 chars**

Felt representation of the "short string"

```
func sum_two_nums(num1: felt, num2: felt) -> (sum: felt) { ... }
```

Variable Declaration

const, let, tempvar & local

```
%builtins output
from starkware.cairo.common.serialize import serialize_word

func main{output_ptr: felt*}() {
   alloc_locals;

   const a = 5;
   let b = a + 5;
   tempvar c = a + b;
   local d = a + b + c;

   serialize_word(a); // 5
   serialize_word(b); // 10
   serialize_word(c); // 15
   serialize_word(d); // 30
   return ();
}
```

alloc_locals;	Needed when using **local** variables
const a = 5;	Can't be redeclared. Resolved by compiler
let b = a + 5;	Works as a reference. Resolved by compiler
tempvar c = a + b;	Stored in memory. Revoking issues
local d = a + b + c;	Stored in memory. No revoking issues

Revoked References

STARKWARE

Compiler loses track of variables

```
func foo() -> felt {
    return 10;
}
```

```
func main{output_ptr: felt*}() {
    tempvar a = 5;
    let b = foo();
    serialize_word(a);    ◄-------------------------------------------------- Fail. Revoked reference
    return ();
}
```

```
func main{output_ptr: felt*}() {
    alloc_locals;
    local a = 5;
    let b = foo();
    serialize_word(a);    ◄-------------------------------------------------- Success
    return ();
}
```

Builtins

Virtual ASICs for expensive computations

STARKWARE

```
%builtins output

from starkware.cairo.common.serialize import serialize_word

func main{output_ptr: felt*}() {
   serialize_word(5);
   return ();
}
```

- `%builtins output` — Defining a builtin
- `func main{output_ptr: felt*}() {` — Builtin pointer passed to **main** automatically
- `serialize_word(5);` — Builtin pointer passed passed implicitly

```
%builtins output pedersen

from starkware.cairo.common.serialize import serialize_word
from starkware.cairo.common.cairo_builtins import HashBuiltin
from starkware.cairo.common.hash import hash2

func main{output_ptr: felt*, pedersen_ptr: HashBuiltin*}() {
   let (foo) = hash2{hash_ptr=pedersen_ptr}(1, 2);
   serialize_word(foo);
   return ();
}
```

- `%builtins output pedersen` — Defining more than one builtin
- `func main{output_ptr: felt*, pedersen_ptr: HashBuiltin*}() {` — Builtin pointers are passed in the same order
- `let (foo) = hash2{hash_ptr=pedersen_ptr}(1, 2);` — Builtin pointer can't be passed implicitly

Implicit Arguments I

```
func serialize_word{output_ptr: felt*}(word) { ... }
```

Library requires a builtin pointer

```
%builtins output
from starkware.cairo.common.serialize import serialize_word

func main{output_ptr: felt*}() {
   alloc_locals;
   local val1 = 5;
   local val2 = 10;

   serialize_word{output_ptr=output_ptr}(val1);
   serialize_word(val2);
   return ();
}
```

Multiple builtins can be declared

%builtins directive passes declared pointers to **main**

Pointers can be passed **explicitly**

Pointers can be passed **implicitly**

STARKWARE

Implicit Arguments II

```
%builtins output
from starkware.cairo.common.serialize import serialize_word

func main{output_ptr: felt*}() {
   foo(5);
   return ();
}

func foo{output_ptr: felt*}(val: felt) {
   bar(val);
   return ();
}

func bar{output_ptr: felt*}(val: felt) {
   serialize_word(val);
   return ();
}
```

Pointer is first passed here

Pointer is passed implicitly

Pointer declared as implicit argument

Pointer is passed implicitly

Pointer declared as implicit argument

Pointer is required here

Tuples

Finite, ordered, unalterable list of elements

```
// A tuple with three elements
local tuple0: (felt, felt, felt) = (7, 9, 13)

// A tuple with a single element
local tuple1: (felt) = (5,)

// A named tuple
local tuple2: (a : felt) = (a=5)

// Tuple that contains another tuple.
local tuple3: (felt, (felt, felt, felt), felt) = (1, tuple0, 5)
local tuple4: ((felt, (felt, felt, felt), felt), felt, felt) = (tuple3, 2, 11)

let a = tuple0[2]  // let a = 13
let b = tuple4[0][1][2]  // let b = 13
```

(5) is not a valid tuple. It needs the comma

Named tuples don't need trailing comma when single item

Accessing values of a tuple

STARKWARE

Type Alias

A custom data type for tuples

```
%builtins output

from starkware.cairo.common.serialize import serialize_word

using MyType = (a: felt, b: felt);

func main{output_ptr: felt*}() {
   alloc_locals;
   local my_val: MyType = (a=1, b=2);
   serialize_word(my_val.a);
   return ();
}
```

Structs

Custom data types

```
%builtins output

from starkware.cairo.common.serialize import serialize_word

struct MyType {
    a: felt,
    b: felt,
}

func main{output_ptr: felt*}() {
    alloc_locals;
    local my_val: MyType* = new MyType(a=1, b=2);
    serialize_word(my_val.a);
    return ();
}
```

Struct definition

Creating a pointer to a struct with **new**

Accessing a member of a struct

Nested Structs

STARKWARE

Complex data types

```
%builtins output
from starkware.cairo.common.serialize import serialize_word

struct Nested {
   c: felt,
}

struct MyType {
   a: felt,
   b: Nested,
}

func main{output_ptr: felt*}() {
   alloc_locals;
   local my_val: MyType* = new MyType(
      a = 1,
      b = Nested(c=2)
   );
   serialize_word(my_val.b.c);
   return ();
}
```

Declaring a member with a custom type

Don't use **new**
We don't want a pointer, we want the object

Accessing a nested member

Array of Felts

STARKWARE

```
%builtins output
from starkware.cairo.common.serialize import serialize_word
from starkware.cairo.common.alloc import alloc

func main{output_ptr: felt*}() {
   fixed_lenght_felt();
   var_lenght_felt();
   return ();
}
```

Library to allocate contiguous memory for arrays
Returns pointer to first memory address

```
func fixed_lenght_felt{output_ptr: felt*}() {
   alloc_locals;
   local felt_array: felt* = new (5, 10);
   serialize_word(felt_array[1]);
   return ();
}
```

For arrays of **fixed** size use **new** keyword
Variable must be **local** or **tempvar**, not **let**

```
func var_lenght_felt{output_ptr: felt*}() {
   let felt_array: felt* = alloc();
   assert felt_array[0] = 6;
   assert felt_array[1] = 12;
   serialize_word(felt_array[1]);
   return ();
}
```

For arrays of **variable** size use **alloc** keyword
Variable must be **let**, not **local** or **tempvar**

assert must be used to manipulate array

Array of Objects

STARKWARE

```
struct MyType {
   a: felt,
   b: felt,
}
```

```
func fixed_lenght_type{output_ptr: felt*}() {
   alloc_locals;
   local obj_array: MyType* = cast(
      new(
         MyType(a=1, b=2),
         MyType(a=3, b=4)
      ),
      MyType*
   );
   serialize_word(obj_array[1].b);
   return ();
}
```

new returns **felt*** by default
We need to **cast** to **MyType***

Multiple instances can be created with a single **new**

Custom type to cast to as pointer

```
func var_lenght_type{output_ptr: felt*}() {
   let felt_array: MyType* = alloc();
   assert felt_array[0] = MyType(a=1, b=2);
   assert felt_array[1] = MyType(a=3, b=4);
   serialize_word(felt_array[1].a);
   return ();
}
```

No need to use **new** because we don't need a pointer

If-Else

Only available control structure (no **for**, **while**, **switch**, etc.)

```
%builtins output
from starkware.cairo.common.serialize import serialize_word

func main{output_ptr: felt*}() {
   alloc_locals;
   local a = 2;
   local b;

   if (a == 5) {
      b = 10;
   } else {
      b = 20;
   }
   serialize_word(b); // 20
   return ();
}
```

Variable can be declared with no value using **local**

Recursion

STARKWARE

Only alternative to for loops in Cairo

```
%builtins output
from starkware.cairo.common.serialize import serialize_word

func main{output_ptr: felt*}() {
    let value = pow_n(x=2, n=3);
    serialize_word(value); // 8      <------------------------------------------ 2^3
    return ();
}

func pow_n(x: felt, n: felt) -> felt {
    return pow_n_tail(acc=x, iter=n, base=x);    <-------------------------- Tail recursion friendly signature
}

func pow_n_tail(acc: felt, iter: felt, base: felt) -> felt {
    if (iter == 1) {    <-------------------------------------------------- End recursion and return result
        return acc;
    }
    let acc = acc * base;
    let iter = iter - 1;
    return pow_n_tail(acc, iter, base);    <------------------------------ Tail recursion. Performance boost
}
```

STARKWARE

Assert

Assignment, comparison and complex operations

```
let felt_array: felt* = alloc();
assert felt_array[0] = 6;
```

Assignment for array elements

```
let x = 3;
let y = 5;
let z = 4;
assert x * x = y + z;
```

comparing value successfully

```
let x = 3;
let y = 5;
let z = 5;
assert x * x = y + z;
```

execution stops with fail status

Cairo Summary

Allows to write **provable programs**

Basic type is **felt**

Max integer is 251 bits prime number

No for loops, only recursion

Define variables with **let**, **tempvar** or **local**

Builtins work like virtual ASICs

Functions have regular and implicit arguments

Failed **asserts** stop execution

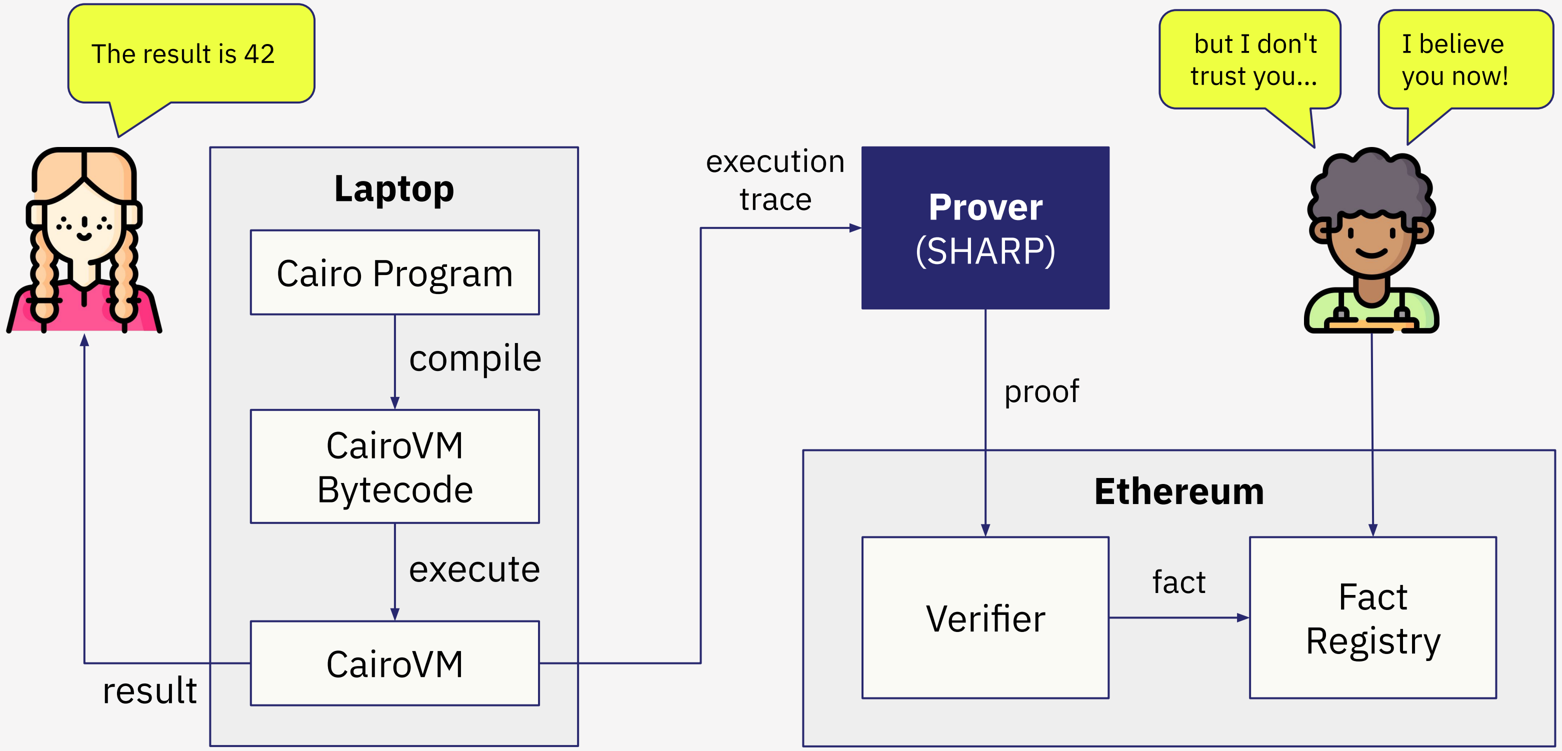
Hints

STARKWARE

StarkNet

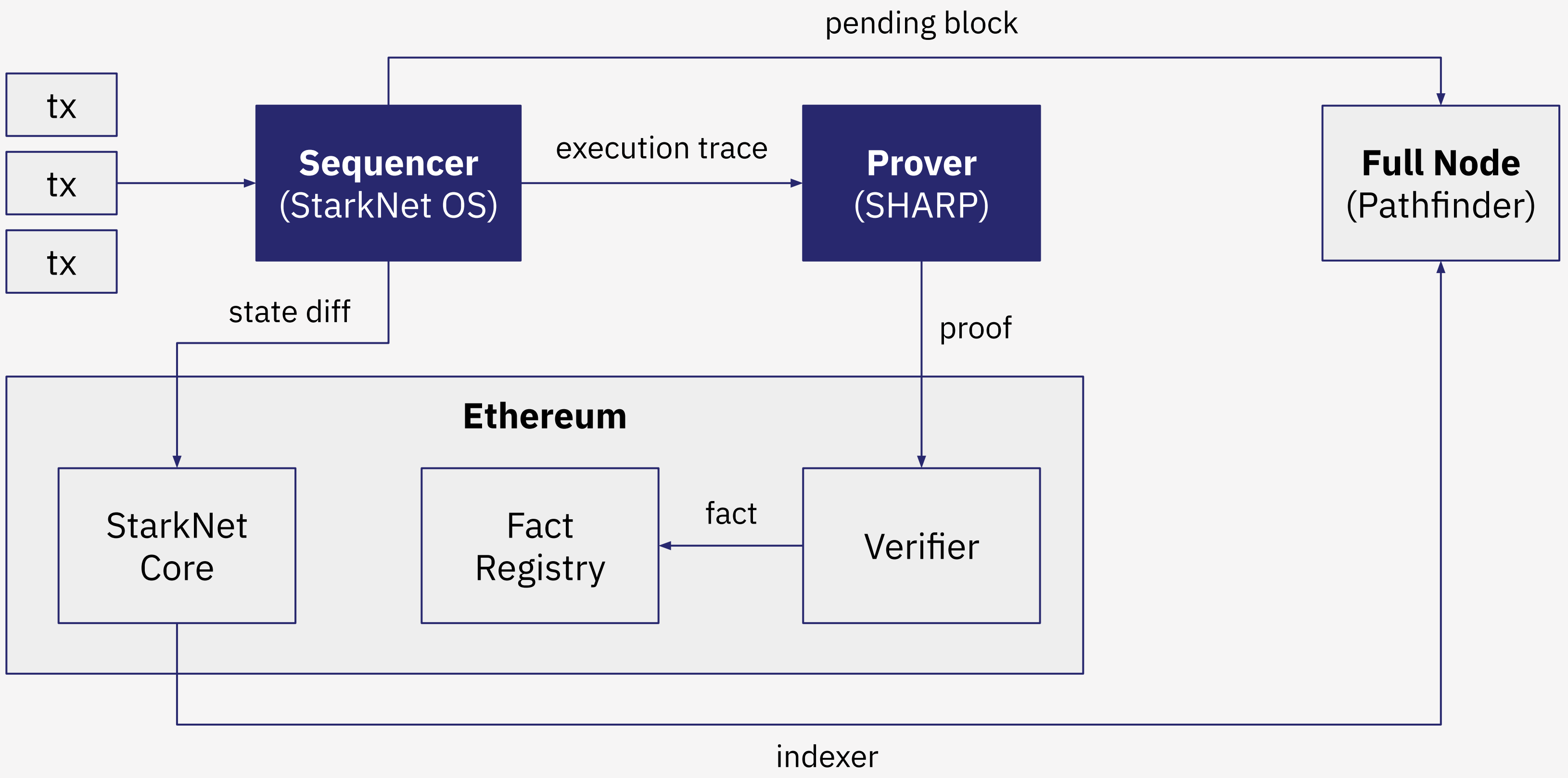
Cairo Architecture Recap

Only on testnet. No real use case



StarkNet's Architecture

Real use case of STARK proofs



Cairo for StarkNet

STARKWARE

Additional features and APIs when using Cairo for StarkNet

```
%lang starknet

@storage_var
func animals(token_id: Uint256) -> (animal: Animal) {
}

@constructor
func constructor{
    syscall_ptr: felt*,
    pedersen_ptr: HashBuiltin*,
    range_check_ptr: felt
}(
    name: felt
) { ... }

@view
func name{...}() -> (name: felt) { ... }

@external
func register_me_as_breeder{...}() -> (is_added: felt) { ... }
```

No need to declare builtins for contracts

Persistent storage

Mapping

Only one constructor function allowed

Commonly used builtin pointers

Does not change internal or global state (call)

Changes internal or global state (invoke)

Creating a User Account

```
// Define testnet as target network for all commands
$ export STARKNET_NETWORK=alpha-goerli

// Define signature scheme to use for all commands
$ export STARKNET_WALLET=starkware.starknet.wallets.open_zeppelin.OpenZeppelinAccount

// Calculate wallet address before deployment
$ starknet new_account

// Fund wallet using faucet

// Estimate gas fees for deployment
$ starknet deploy_account --estimate_fee

// Deploy user account
$ starknet deploy_account
```

Declare Contract

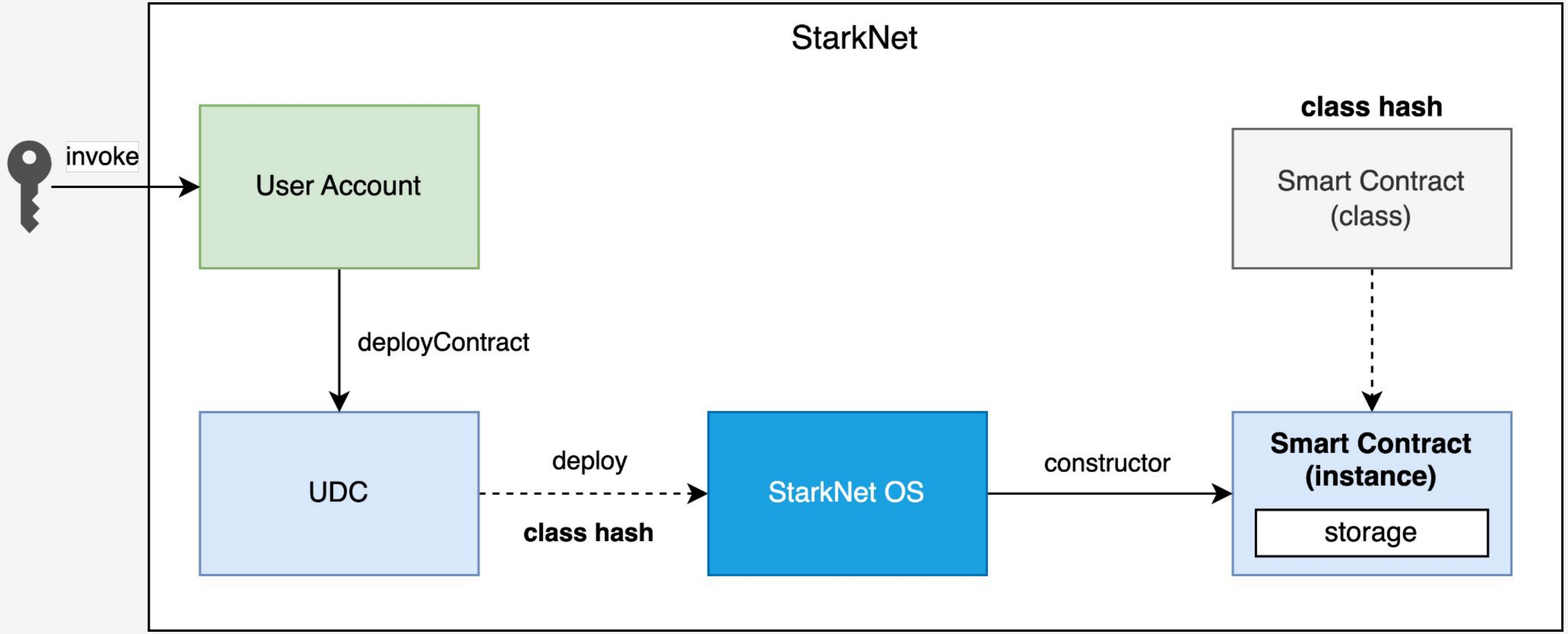
```
// Compile contract
$ starknet-compile starknet/voting.cairo --output compiled/voting.json

// Declare contract
$ starknet declare --contract compiled/voting.json

// Calculate class hash (if declared already)
$ starknet-class-hash compiled/voting.json
```

Voting Contract Class Hash
hex: 0x60101ef5dc96cff8224d3d48e06b8091ffecbea578545b56a42ecf1032d3bf2
felt: 2715657293614881836478276174981963700795133668158601100932833519980176554994

The Universal Deployer Contract (UDC)



Deploy Contract

STARKWARE

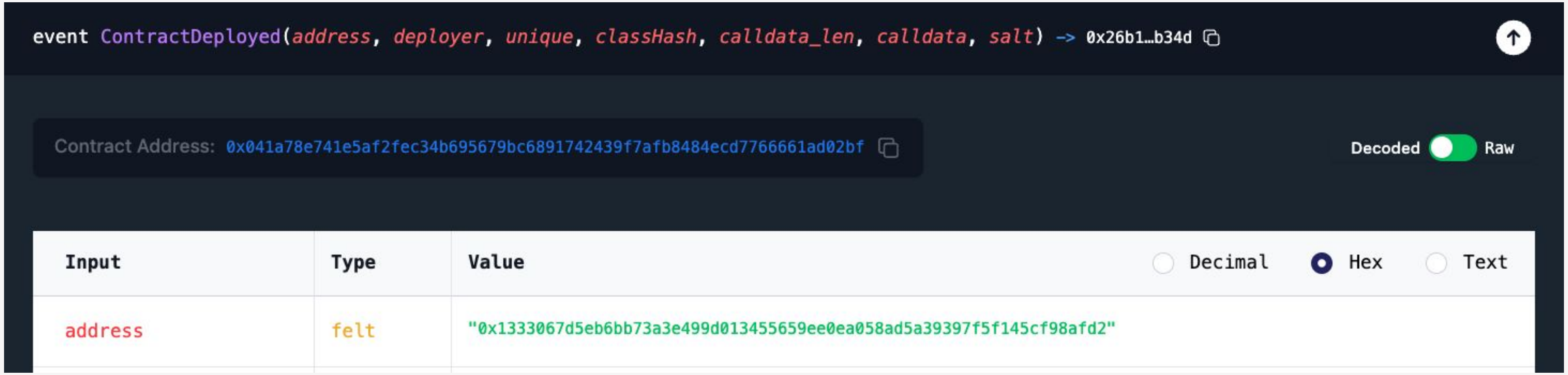
```
func deployContract{ ... }(
   classHash: felt,
   salt: felt,
   unique: felt,
   calldata_len: felt,
   calldata: felt*
) -> (address: felt) {
```

```
func constructor{ ... }(
   admin_address: felt,
   registered_addresses_len: felt,
   registered_addresses: felt*
) { ... }
```

```
starknet invoke \
   --address 0x041a78e741e5af2fec34b695679bc6891742439f7afb8484ecd7766661ad02bf \  ◄---- UDC address
   --abi abis/udc.json \
   --function deployContract \
   --inputs
      0x60101ef5dc96cff8224d3d48e06b8091ffecbea578545b56a42ecf1032d3bf2  ◄---- Voting class hash
      0  ◄---- Salt
      0  ◄---- Unique (boolean)
      5  ◄---- calldata_len
      0x0732b42ffe95457c1cD8788383fC9b53e70F7331deb4cbb76644bED1b528681C  ◄---- admin_address
      3
      0x031409d4bD912a707d9aD6F0ae57E471ac6aEf0eE462098B2874000af338cEF7  ◄---- Voter 1 address
      0x015b5097AfCaFc6fca7b5E0c0EAaa3990d57d5212F86f89d4971d6523eDf58fc  ◄---- Voter 2 address
      0x020beEdabD63ad5fE5359d6f8f145Ee3532eB10788dA103fc30dCF97a898f5D8  ◄---- Voter 3 address
```

Interacting with the Voting Contract

Check "events" tab of deploy tx



```
event ContractDeployed(address, deployer, unique, classHash, calldata_len, calldata, salt) -> 0x26b1…b34d
```

Contract Address: 0x041a78e741e5af2fec34b695679bc6891742439f7afb8484ecd7766661ad02bf

Input	Type	Value
address	felt	"0x1333067d5eb6bb73a3e499d013455659ee0ea058ad5a39397f5f145cf98afd2"

Voting contract on Starkscan

StarkNet Summary

Cairo for StarkNet uses additional APIs

Smart contracts can have only one constructor

A **view** function doesn't change the state (read-only) and is **called** for free

An **external** function changes the state (write) is **invoked** paying gas fees

Account abstraction separates Signer from User Account

Deployment is done with the UDC

Error Handling

Authorization

STARKWARE

Thanks!

StarkNet EDU

@starknet_edu

January 2023