Sway

A modern programming language for blockchain

1. What is Sway?
2. Why a new language?
3. A Deep Dive
4. Where to learn more?

What is Sway?

- DSL for blockchain programming
- Fully featured type system
  - Generics
  - Trait based polymorphism
  - Algebraic types
- Built-in static analysis
  - Compiler enforcement of the CEI pattern
- Main target is FuelVM

Why a new language?

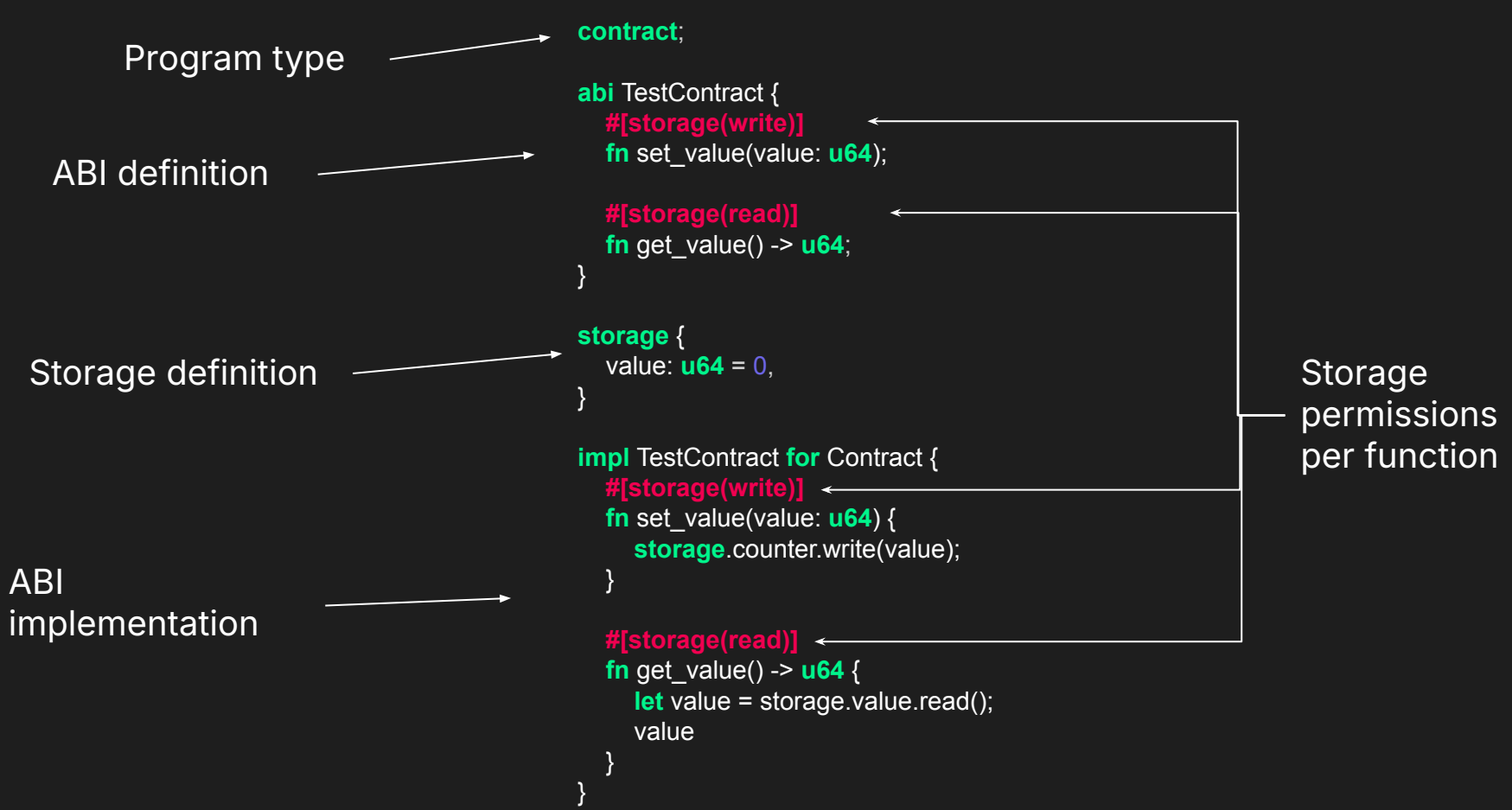
- Solidity is the standard but it has big shortcomings
  - Inexpressive type system
  - Fragmented tooling ecosystem
- Rust and other GPPLs are not suited for blockchain
  - Complicated memory model
  - Optimized for runtime

A Deep Dive into Sway

Program types

- `library`
  - Common behavior that can be imported elsewhere
- `script`
  - Runnable bytecode that executes once to perform some task
  - Can call contracts
  - No persistent storage
- `contract`
  - On chain bytecode with a defined interface (ABI)
  - Persistent storage
  - Can call other contracts
- `predicate`
  - `Has an address based on the bytecode hash`
  - Returns a boolean value
  - Represents ownership of a resource
  - No side effects allowed

A simple example


Program type
ABI definition
Storage definition
ABI implementation
contract;
abi TestContract {
#[storage(write)]
fn set_value(value: u64);
#[storage(read)]
fn get_value() -> u64;
}
storage {
value: u64 = 0,
}
impl TestContract for Contract {
#[storage(write)]
fn set_value(value: u64) {
storage.counter.write(value);
}
#[storage(read)]
fn get_value() -> u64 {
let value = storage.value.read();
value
}
}
Storage permissions per function

Memory Model

One Big Arena

Everything is transient and operations are costly, hence:

- No lifetime management
- No deallocation
- References behave more like C++ less like Rust

Type System

Built-ins

- Primitives
  - `u8 u16 u32 u64 u256 str str[] bool b256 raw_ptr raw_slice`
- Tuples
  - `(u64, bool, u64)`
  - `()`
- Arrays
  - `[u64; 5]`

User defined

- Structs
  - `struct MyStruct { a: u64, b: u32 }`
- Enums
  - `enum MyEnum { Foo: (), Bar: (u64, u64) }`

```
struct Foo {
    bar: u64,
    baz: bool,
}

impl Foo {
    fn is_baz_true(self) -> bool {
        self.baz
    }

    fn new_foo(number: u64, boolean: bool) -> Foo {
        Foo {
            bar: number,
            baz: boolean,
        }
    }
}
```

Traits

- Methods a type must implement
- Can be used with parametric polymorphism
- Can provide own methods

```
trait Compare {
    fn equals(self, b: Self) -> bool;
} {
    fn not_equals(self, b: Self) -> bool {
        !self.equals(b)
    }
}
```

Abis

- Similar but conceptually different from traits
- Can inherit from traits
- Implementation means adding methods to the interface of a contract
- Can be used to call contract methods

```
abi Counter {
    #[storage(read, write)]
    fn increment();

    #[storage(read)]
    fn count() -> u64;
}
```

Generics and Associated Types

- Similar to Rust
- No GATs

```
fn into_rectangle<T>(t: T) -> Rectangle
where
    Rectangle: Convert<T>;
```

```
trait MyTrait {
    type AssociatedType;
}
```

```
trait Convert<T> {
    fn from(t: T) -> Self;
}
```

Constants and Configurables

- Data section values
- Configurables
  - use the ABI encoding
  - are decoded at runtime
  - can be changed at “configuration-time”
  - useful with predicates
  - useful for factory patterns

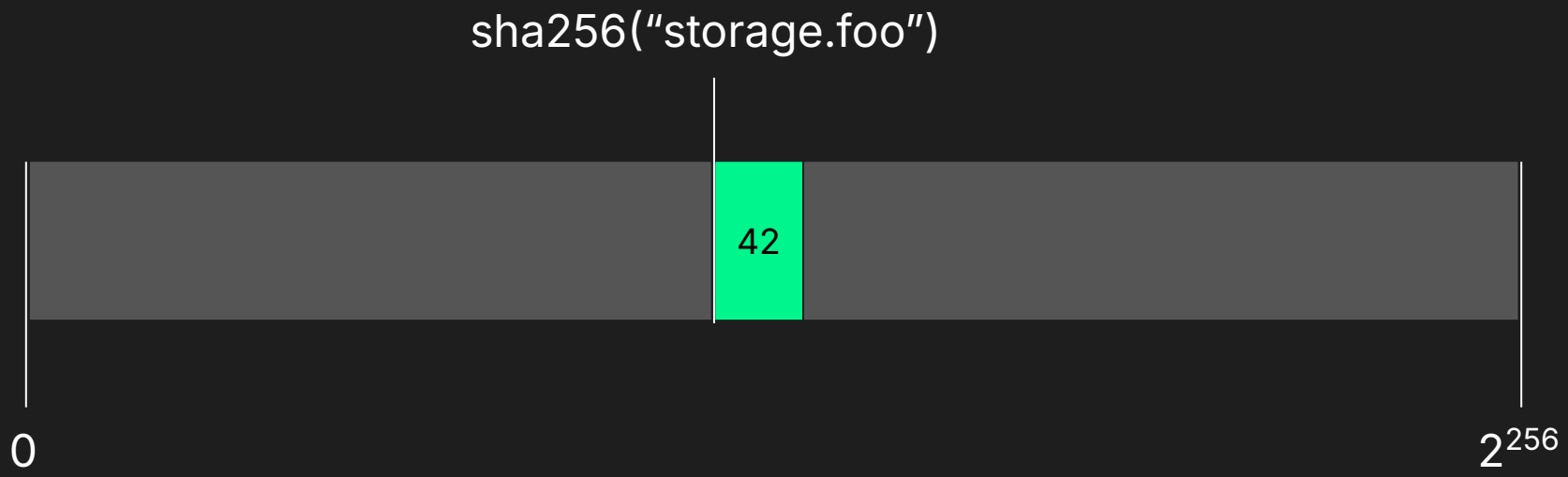
```
const CONSTANT: u64 = 42;
configurable {
    CONFIGURABLE: u64 = 42,
}
```

Storage

Persistent Storage for Contracts

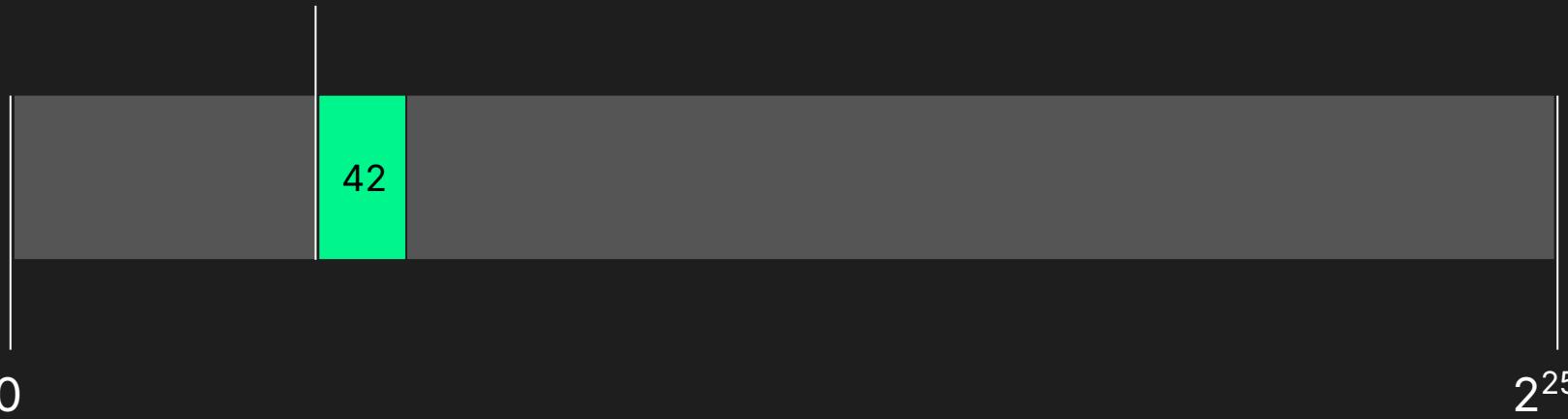
- One **`storage`** definition per contract source
- Permissions effects carried as part of function type
  - `#[storage(read)]`
  - `#[storage(write)] #[storage(read, write)]`
- A series of 256 bit slots

```
storage {
    foo: u64 = 42,
}
```



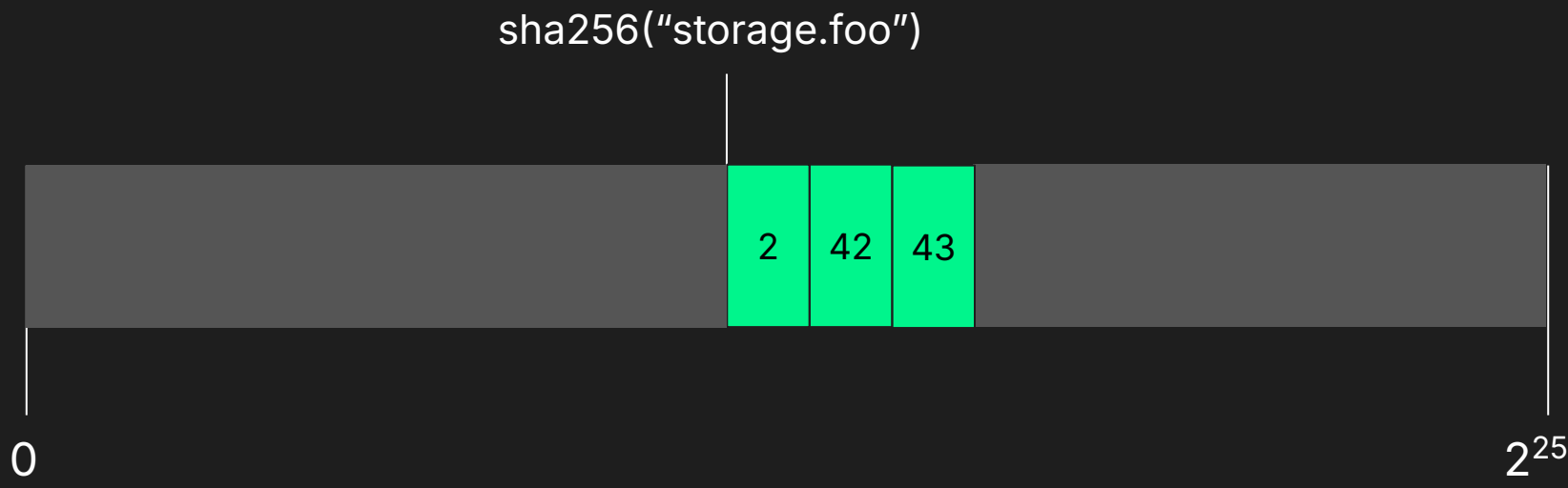
```
storage {
    bar {
        foo: u64 = 42,
    }
}
```

sha256(“storage::bar.foo”)



```
storage {
    foo: StorageVec<u64> = StorageVec{},
}
```

42	43



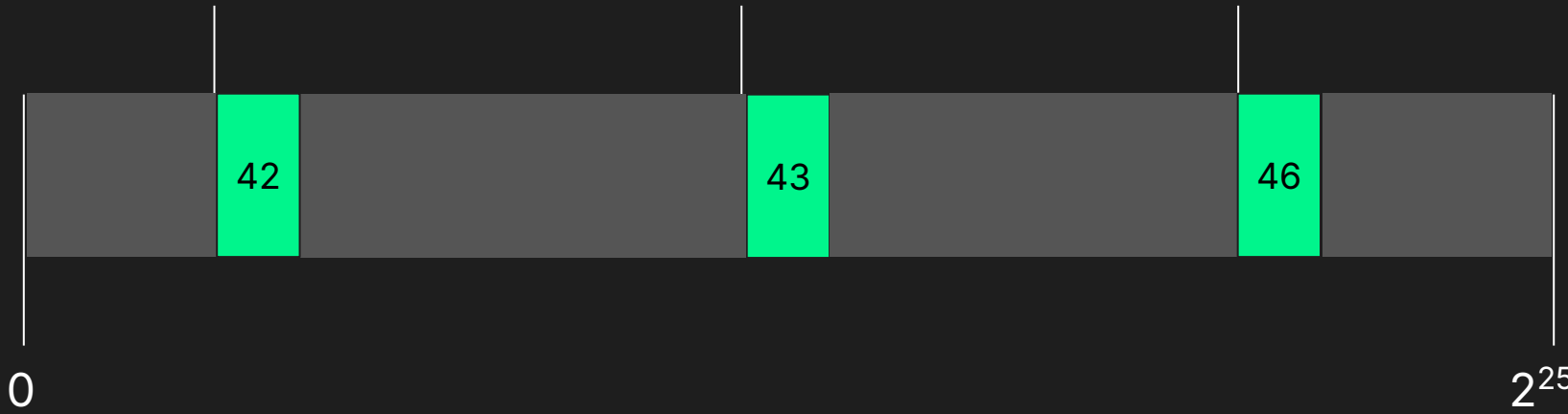
```
storage {
    foo: StorageMap<u64, u64> = StorageMap{},
}
```

1	2	5
42	43	46

sha256(
  (1, “storage.foo”)
)

sha256(
  (2, “storage.foo”)
)

sha256(
  (5, “storage.foo”)
)



ABI

ABI Files

- The compiler generates ABI files alongside contract bytecode
- Equivalent to header files
- Lists available contract methods, types, etc

ABI Encoding

- Lightweight, self contained, dynamically sized
- Three implementations:
  - Rust
  - TypeScript
  - Sway

Calling Contracts

- CALL vs LDC

```
let x = abi(MyContractAbi, contract_id);
let return_value = x.foo(42);
```

```
run_external(contract_id);
```

Standards

Standards

- SRC-20: Native assets
  - SRC-3: Mint and Burn
  - SRC-7, SRC-9: Metadata (for NFTs)
  - SRC-6: Vaults
- SRC-5: Ownership
- SRC-14: Simple Proxies

...and a lot more

Where to learn more?

Documentation Resources

- The Sway Book and the unified Fuel Docs
  - docs.fuel.network
- The playground
  - sway-playground.org
- Sway by example
  - swaybyexample.com
- The standard library docs
  - fuellabs.github.io/sway/master/std/