Rust 1

Ms. Jignasha Dalal

Corporate IT Trainer,

Rapid Innovation

Rust vs Solidity

Rust

- Compiles to WASM
- Low level language
- Fast execution
- Memory safe

Solidity

- Compiles to EVM
- High level language
- slow execution
- Programmer has to be careful

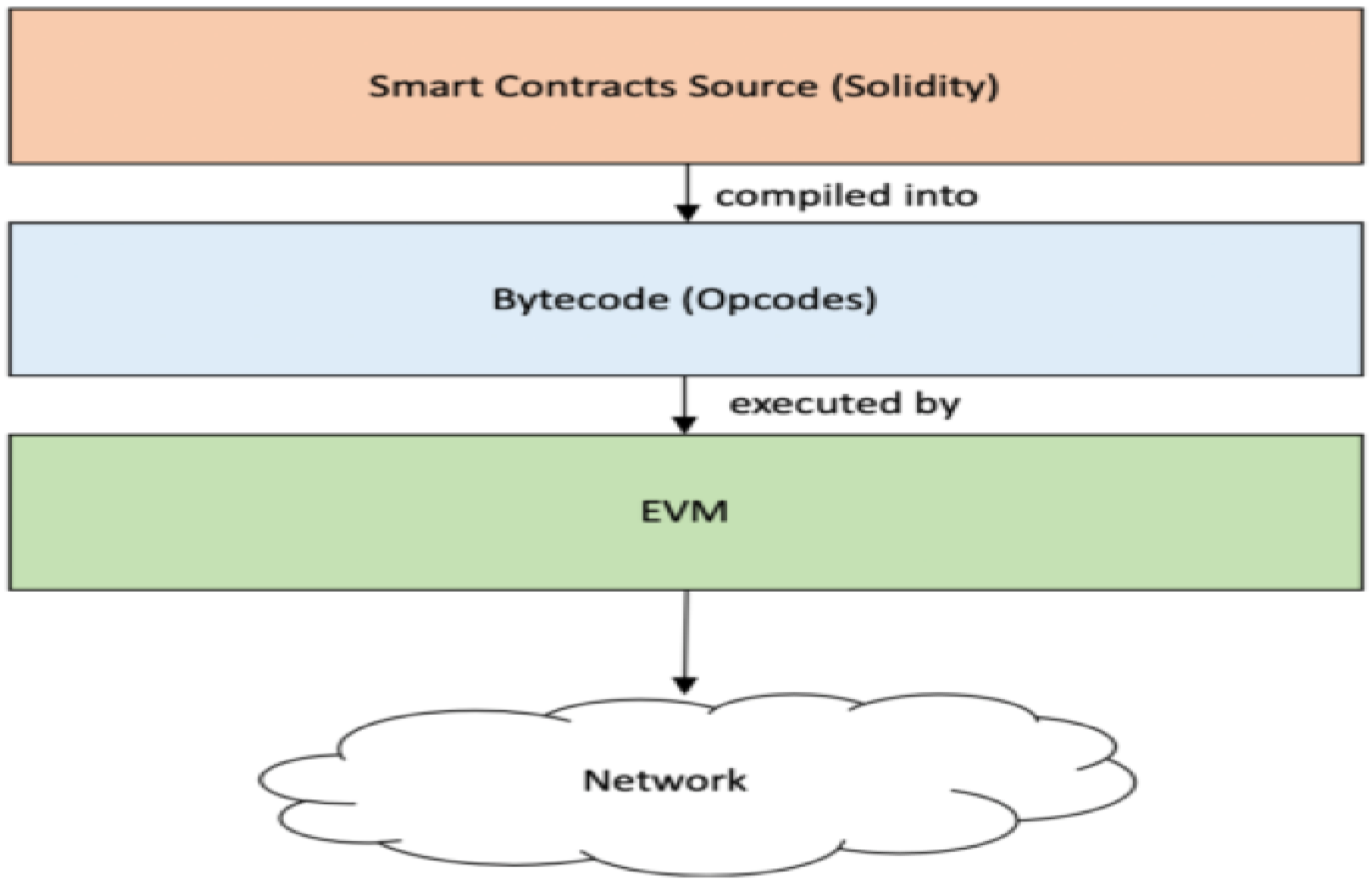
Rust vs Solidity

Sr. No	Solidity	Rust
1	Object Oriented, High level and statically typed	Multi paradigm (object oriented, functional and imperative), Low level
2	Low memory safety and efficiency	High Memory Safety and efficiency (no garbage collector)
3	Easier To learn	Hard to learn
4	Speed is low	Speed is high
6	Integer overflow and underflow problems	Not an issue here
7	Difficult to perform static analysis (analysing and troubleshooting code without running)	Easy to do
8	Polygon, Ethereum, Avalanche, Hashgraph, etc.	Solana, Near, Substrate, Cosmos

EVM and WASM

EVM

- State and stack machine



EVM and WASM

EVM challenges

- EVM call stack has limitation of 1024 items which limits the complexity of smart contracts
- All operations are 256 bit.
    - Our processors are 32/64 bits, needs to process 256 bit instructions
    - Result --- Reduced performance
- Very few languages are compiled to EVM (Solidity is most popular)

EVM and WASM

WASM (Web Assembly)

- WebAssembly or WASM is a Compiler Target (code generated by compilers) with a binary format which allows us to execute **C, C++, C#, typescript, Haxe, Kotlin and Rust** on the browsers with a performance close to native code.
- It is memory-safe, sandboxed, and platform-independent;
- The environment possesses 64 and 32-bit integer operation support that maps one-to-one with CPU instructions
- Quickly adapts to any machine-level architecture making it extremely high-performing.
- Comes with an instruction set that's compatible with most modern hardware architectures.
- Runs close to native speed on most platforms.

Rust Installation

1. **Mac OS or Linux**

   $ curl --proto '=https' --tlsv1.3 https://sh.rustup.rs -sSf | sh

   $ xcode-select –install

2. **Windows**

   To acquire the build tools, you’ll need to install Visual Studio 2022. When asked which workloads to install, include:

   “Desktop Development with C++”

   The Windows 10 or 11 SDK

   The English language pack component, along with any other language pack of your choosing

   Install rust using https://www.rust-lang.org/tools/install

Rust Installation

1. **Check the version**

   ```
   rustc –version
   ```

2. Rust update

   ```
   rustup update
   ```

3. Uninstall rust

   ```
   rustup self uninstall
   ```

Rust Hello world

1. **Create project directory**
2. **Open main.rs file in the choice of your editor**

```
fn main() {
    println!("Hello, world!");
}
```

On mac Os or Linux

```
$ rustc main.rs
$ ./main

Hello, world!
```

On Windows

```
$ rustc main.rs
$ .\main.exe

Hello, world!
```

Rust cargo

```
$ cargo new hello_cargo
$ cd hello_cargo
$ cargo run
Hello, world!
```

Main.rs file

```
fn main() {
    println!("Hello, world!");
}
```

Cargo.toml

```
[package]
name = "hello_cargo"
version = "0.1.0"
edition = "2021"

# See more keys and their definitions at https://doc.rust-lang.org/cargo/reference/manifest.html

[dependencies]
```

Immutable vs mutable variables

let apples=5;   //immutable by default– similar to constants

References are immutable by default

let mut guess : String = String :: new (); // mutable, similar to variable

guess--- mutable variable of type string

= -- binding some value to the variable

String::new() --- empty instance of type String

let x = 5;

let y = 10;

println!("x = {x} and y + 2 = {}", y + 2);

```
use std::io;

fn main() {
    println!("Guess the number!");

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin()
        .read_line(&mut guess)
        .expect("Failed to read line");

    println!("You guessed: {guess}");
}
```

Crates

crate is a collection of Rust source code files

Cargo’s coordination of **external crates** is where Cargo really shines

Filename: Cargo.toml

```
[dependencies]

rand = "0.8.5“
```

[dependencies] tell Cargo which external crates your project depends on and which versions of those crates you require

```
$ cargo build
```

Crates

- When we include an external dependency, Cargo fetches the latest versions of everything that dependency needs from the registry, which is a copy of data from Crates.io.
- Crates.io is where people in the Rust ecosystem post their open source Rust projects for others to use.
- After updating the registry, Cargo checks the [dependencies] section and downloads any crates listed that aren’t already downloaded
- **Cargo only compiles the changes made in the code or cargo.toml (efiicient)**

Cargo.lock

- When you build a project for the first time, Cargo figures out all the versions of the dependencies that fit the criteria and then writes them to the **Cargo.lock** file.
- When you build your project in the future, Cargo will see that the **Cargo.lock** file exists and will use the versions specified there rather than doing all the work of figuring out versions again.
- This lets you have a reproducible build automatically.

Cargo.lock and updating of crate

```
$ cargo update
```

will ignore the Cargo.lock file and figure out all the latest versions that fit your specifications in Cargo.toml.

Generating a secret number

```
use std::io;
use rand::Rng; //Rng is trait

fn main() {
    println!("Guess the number!");

    let secret_number = rand::thread_rng().gen_range(1..=100); // gen_range function is defined in Rng
trait

    println!("The secret number is: {secret_number}");

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin() .read_line(&mut guess).expect("Failed to read line");

    println!("You guessed: {guess}");
}
```

Crate documentation

Note: You won’t just know which traits to use and which methods and functions to call from a crate, so each crate has documentation with instructions for using it.

Another neat feature of Cargo is that running the `cargo doc --open` command will build documentation provided by all your dependencies locally and open it in your browser.

If you’re interested in other functionality in the rand crate, for example, run `cargo doc --open` and click rand in the sidebar on the left.

match

- A match expression is made up of arms.
- An arm consists of a pattern to match against, and the code that should be run if the value given to match fits that arm’s pattern.
- Rust takes the value given to match and looks through each arm’s pattern in turn.

Compare guess to secret number

```
use rand::Rng; // Rng is trait
use std::cmp::Ordering; // Ordering is enum with three values: "Less", "Greater"
"Equal"
use std::io;

fn main() {
    // --snip--

    println!("You guessed: {guess}");

    match guess.cmp(&secret_number) {
        Ordering::Less => println!("Too small!"),
        Ordering::Greater => println!("Too big!"),
        Ordering::Equal => println!("You win!"),
    }
}
```

Compare guess to secret number

```
// --snip--

   let mut guess = String::new();

   io::stdin().read_line(&mut guess).expect("Failed to read line");

   let guess: u32 = guess.trim().parse().expect("Please type a number!");  //shadowing a variable

   println!("You guessed: {guess}");

   match guess.cmp(&secret_number) {
       Ordering::Less => println!("Too small!"),
       Ordering::Greater => println!("Too big!"),
       Ordering::Equal => println!("You win!"),
   }
```

loop

- The **loop** keyword creates an infinite loop.
- **Stopping the loop:**
  - Give non number input and program will crash
  - Use break statement