Building Smart Contracts in JavaScript

near.org

Agenda

2022 - Q2 - May 23

1. Languages Overview
2. NEAR JavaScript SDK
3. FT & NFT & XCC
4. Example: Guest Book
5. Future Plan

The Road Not Taken

Two roads diverged in a wood, and I–
I took the one less traveled by,
And that has made all the difference.

– by Robert Frost

NEAR Contract Languages Overview

AssemblyScript

Pros

- Easier for prototyping
- Trivial for JS and TS devs to learn
- Smaller binaries in Wasm
- Binaries are easier to read / debug

Cons

- Immature compiler and ecosystem
- Debugging tools are immature

Rust

Pros

- Mature, battle-hardened compiler
- Thriving ecosystem
- Real world use cases
- near-sdk-rs makes life easy

Cons

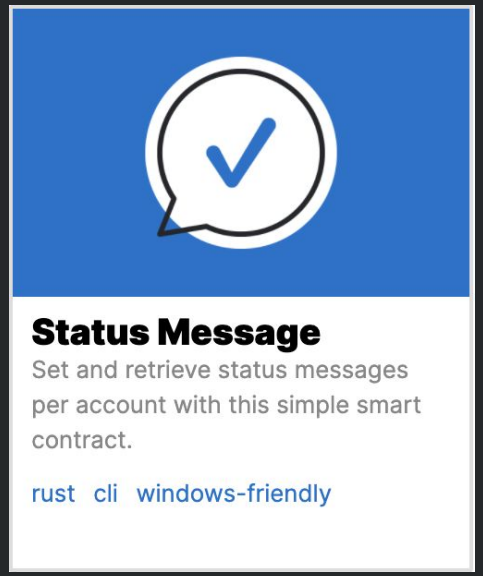
- Steep learning curve
- Even experts fight the compiler

Why Building Smart Contract in JS?

A Language for Mass Adoption

1. One of the most popular programming languages in the world
2. Minimal learning cost for Web 2.0 developers and newbies
3. A large and mature ecosystem comparing to AssemblyScript
4. Build DApps in a single language
5. Better developer experience comparing to AssemblyScript

Status Message



Code
- `package.json`
- `src/index.js`

Build
- `yarn`
- `yarn build`

Test
- `yarn test`

https://github.com/near/near-sdk-js/blob/master/examples/status-message

Status Message: Deploy to Testnet

Deploy

- `near create-account status-message.alice.near —-masterAccount alice.near —-initialBalance 1`
- `(near-sdk-js/examples/status-message) near call $JSVM_ACCOUNT deploy_js_contract --accountId $CONTRACT_ID --args $(cat build/status-message.base64) --base64 --deposit 0.1`
- `(near-sdk-js) near call $JSVM_ACCOUNT call_js_contract --accountId $ACCOUNT_ID --base64 --args $(node encode_call.js $CONTRACT_ID init '')`

Call

- `(near-sdk-js) near call $JSVM_ACCOUNT call_js_contract --accountId $ACCOUNT_ID --base64 --args $(node encode_call.js $CONTRACT_ID set_status '["near-sdk-js is cool"]') --deposit 0.01`

View

- `(near-sdk-js) near call $JSVM_ACCOUNT call_js_contract --accountId $ACCOUNT_ID --base64 --args $(node encode_call.js $CONTRACT_ID get_status '["bot.testnet"]')`

NEAR JavaScript SDK

Docs

- https://github.com/near/near-sdk-js
- https://github.com/near/near-sdk-js/blob/master/src/api.js

Decorators:

```
@NearBidgen
@call, @view
```

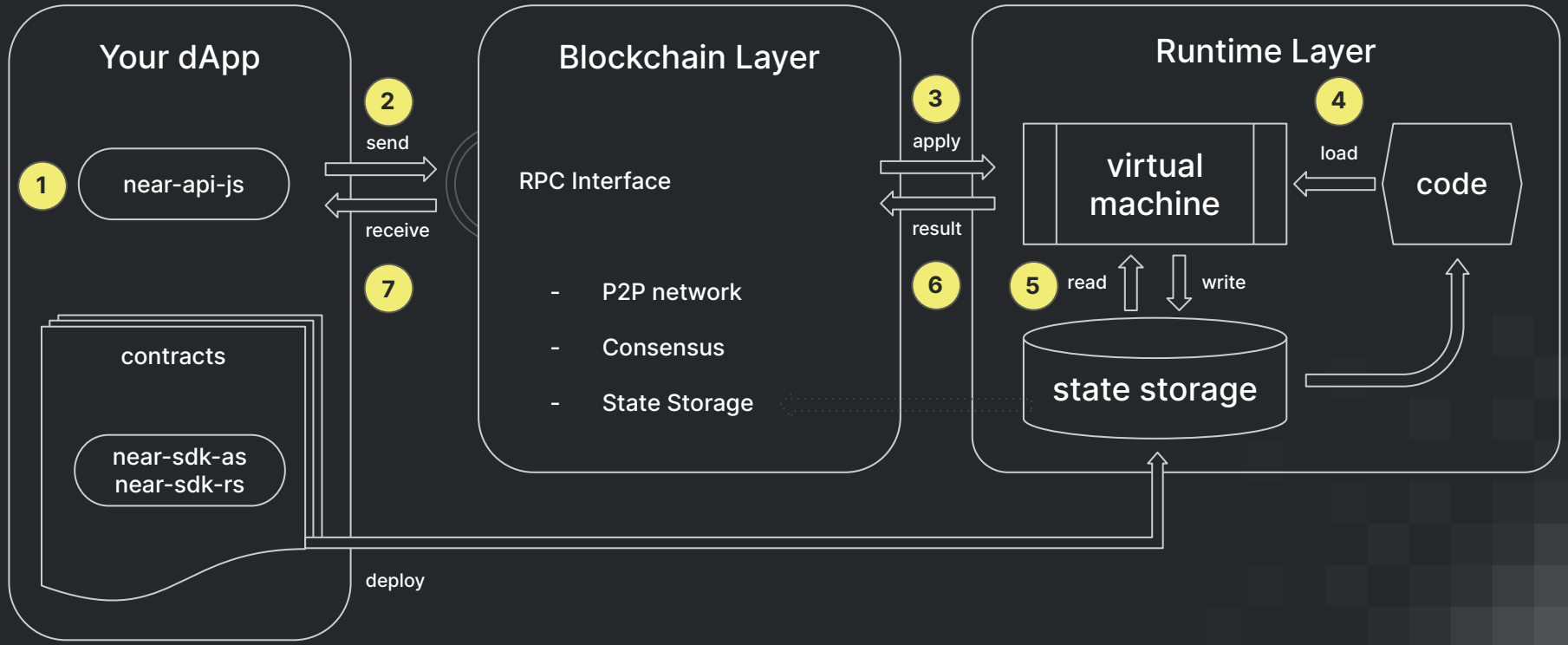
3 Types of Account IDs

1. `near.signerAccountId()`
2. `near.predecessorAccountId()`
3. `near.jsvmJsContractName()`

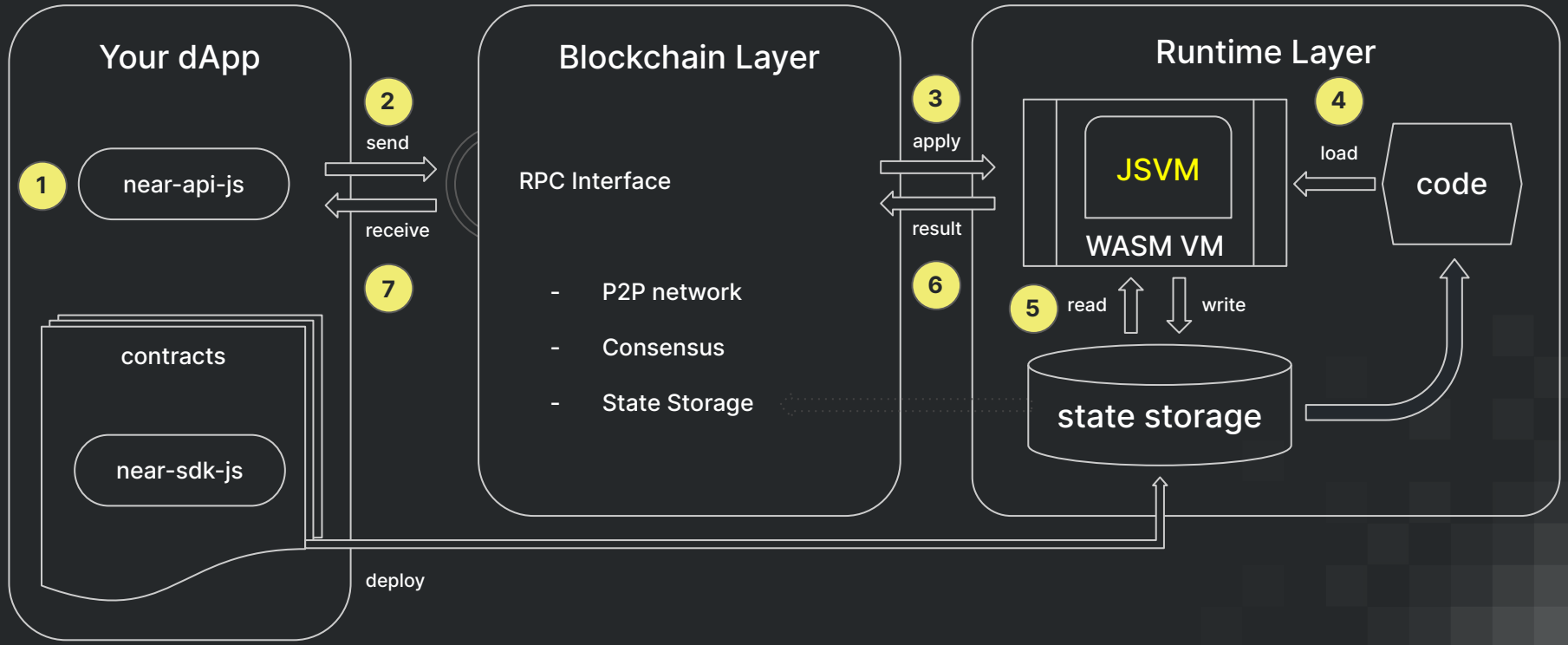
Collections

```
LookupMap
LookupSet
Vector
```

NEAR dApps at a lower level



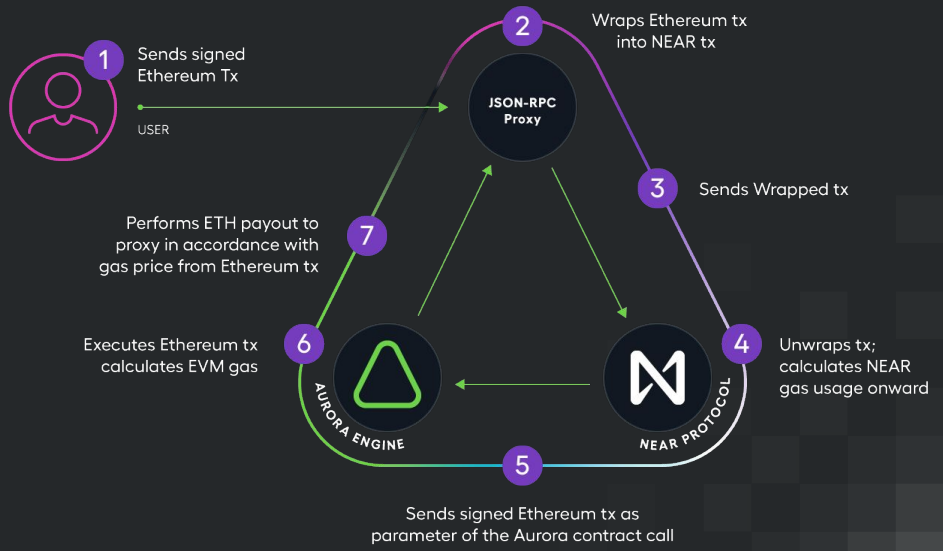
How JavaScript Contract Works?



How is JSVM implemented?

JavaScript Virtual Machine built as one Smart Contract on NEAR

1. A standalone environment for JavaScript contracts
2. An embedded lightweight JavaScript Engine based on QuickJS
3. All JS contracts deployed to JSVM contract
4. JS contract calls routed via JSVM
5. Similar to Aurora Engine for EVM / Solidity



Build Contracts with JavaScript v.s. Solidity

JavaScript

Pros

- Top common language
- Powerful ecosystem
- Frontend/contract in one language
- Synchronous cross-contract call

Cons

- Still at early stage
- Immature tools and practices

Solidity

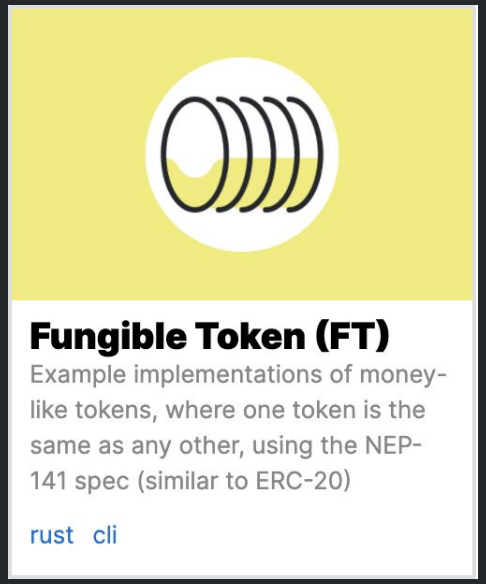
Pros

- Dominant smart contract language
- Flourish tools and tutorials
- Real world use cases
- Low learning curve

Cons

- Not a common language
- Security concerns

Fungible Token (FT)



https://github.com/near/near-sdk-js/blob/master/examples/fungible-token

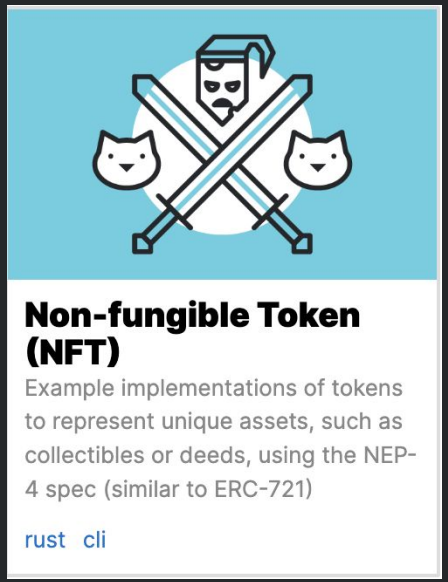
Cross-Contracl Call (XCC) is **sync within JSVM**

Standard: **NEP-141**

Another XCC Example: **cross-contract-call**

Testing: workspaces-js

Non-Fungible Token (NFT)



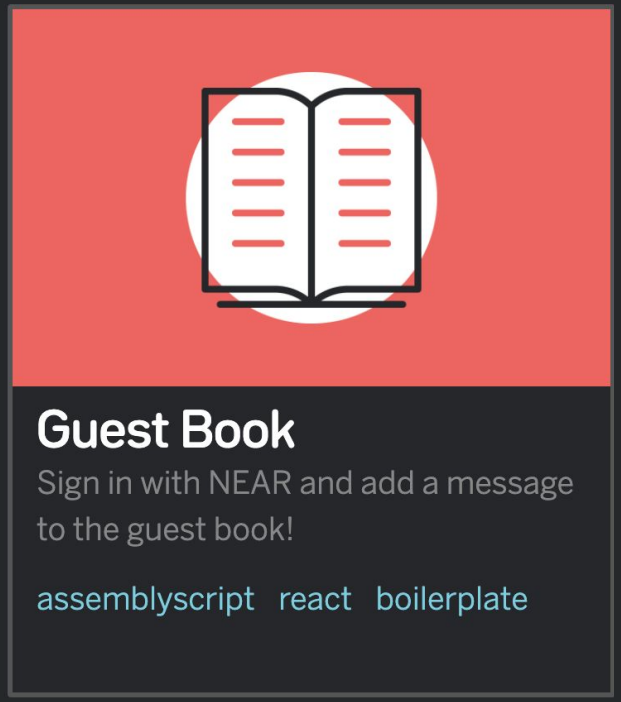
https://github.com/near/near-sdk-js/blob/master/examples/non-fungible-token

Cross-Contracl Call (XCC) is **sync within JSVM**

Standard: **NEP-171**

Testing: `workspaces-js`

Guest Book



https://github.com/think-in-universe/guest-book/tree/js-demo

NEAR Guest Book

Log out

Sign the guest book, rando.testnet!

Message:

Donation (optional): 1 Ⓝ

Sign

Messages

rando.testnet:
Hello, world!

rando.testnet:
And here's a donation for ya

Future Plan

1. NEAR Standards
    a. Fungible Token / Non-Fungible Token
    b. Events
    c. Storage Management
2. Tooling
    a. CLI
    b. Explorer
    c. near-api-js
3. Security
    a. Function Call Key

Thank you!