# Nexus: A domain-specific language for financial smart contracts

Noah-Vincenz Noeh

# 1. Nexus

# 1. Nexus

- based on Peyton Jones et al. paper
- leverages Ethereum smart contracts to compose financial contracts
- any arbitrary complex contract can be trivially composed from set of simpler ones
- end user not required to write smart contract code
- allows for safer smart contracts
- lower cost smart contracts

# 1.1 Horizon: Contract -> DateTime

| | | |
|---|---|---|
| H(zero) | = | infinite |
| H(one) | = | infinite |
| H(give c) | = | H(c) |
| H(scaleK k c) | = | H(c) |
| H(truncate t c) | = | min(t, H(c)) |
| H(get c) | = | H(c) |
| H($c_1$ and $c_2$) | = | max(H($c_1$), H($c_2$)) |
| H($c_1$ or $c_2$) | = | max(H($c_1$), H($c_2$)) |

# 1.2 Value: Contract -> Number

| | | | |
|---|---|---|---|
| V(zero) | = | 0 | |
| V(one) | = | 1 | |
| V(give c) | = | -V(c) | |
| V(scaleK k c) | = | k * V(c) | |
| V(truncate t c) | = | V(c) | on $\{t \mid t \leq H(c)\}$ |
| | | 0 | on $\{t \mid t > H(c)\}$ |
| V(get c) | = | V(c) | on $\{t \mid t = H(c)\}$ |
| | | 0 | on $\{t \mid t > H(c) \wedge t < H(c)\}$ |
| V($c_1$ and $c_2$) | = | $V(c_1) + V(c_2)$ | on $\{t \mid t \leq H(c_1) \wedge t \leq H(c_2)\}$ |
| | | $V(c_1)$ | on $\{t \mid t \leq H(c_1) \wedge t > H(c_2)\}$ |
| | | $V(c_2)$ | on $\{t \mid t > H(c_1) \wedge t \leq H(c_2)\}$ |
| V($c_1$ or $c_2$) | = | $\max(V(c_1), V(c_2))$ | on $\{t \mid t \leq H(c_1) \wedge t \leq H(c_2)\}$ |
| | | $V(c_1)$ | on $\{t \mid t \leq H(c_1) \wedge t > H(c_2)\}$ |
| | | $V(c_2)$ | on $\{t \mid t > H(c_1) \wedge t \leq H(c_2)\}$ |

# 1.3 Conditionals

$$
\begin{aligned}
&\textit{if} \;\; \textit{boolean condition} \\
&\qquad \textit{consequent} \\
&\{ \; \textit{else} \\
&\qquad \textit{alternative} \; \}
\end{aligned}
$$

# 1.4 Comparing Horizon: (Contract, Contract) -> Bool

$$c_1\{>\}c_2 \iff H(c_1) > H(c_2)$$

- zero {>} one ✗
- scaleK 10 (one) {>} truncate “24/12/2019 23:33:33" ( one ) ✓
- ( truncate “20/12/2019 23:33:33" ( one ) and truncate “24/12/2019 23:33:33" ( zero ) ) {>} truncate “24/12/2019 23:33:33" ( one ) ✗
- ( truncate “23/12/2019 23:33:33" ( one ) and truncate “26/12/2019 23:33:33" ( one ) ) > truncate “24/12/2019 23:33:33" ( one ) ✓

# 1.5 Comparing Value: (Contract, Contract) -> Bool

$$c_1[>]c_2 \iff V(c_1) > V(c_2)$$

- zero [>] one ✗
- scaleK 10 (one) [>] truncate “24/12/2019 23:33:33" ( one ) ✓
- ( truncate “20/12/2019 23:33:33" ( one ) and truncate “24/12/2019 23:33:33" ( zero ) ) [>] truncate “24/12/2019 23:33:33" ( one ) ✗
- ( truncate “23/12/2019 23:33:33" ( one ) and truncate “26/12/2019 23:33:33" ( one ) ) [>] truncate “24/12/2019 23:33:33" ( one ) ✓

# 1.6 Comparing Domination: (Contract, Contract) -> Bool

$$c_1 > c_2 \iff H(c_1) > H(c_2) \wedge \forall t < H(c_2).V(c_1)(t) > V(c_2)(t)$$

- zero > one ✗
- scaleK 10 (one) > truncate “24/12/2019 23:33:33" ( one ) ✓
- ( truncate “20/12/2019 23:33:33" ( one ) and truncate “24/12/2019 23:33:33" ( zero ) ) > truncate “24/12/2019 23:33:33" ( one ) ✗
- ( truncate “23/12/2019 23:33:33" ( one ) and truncate “26/12/2019 23:33:33" ( one ) ) > truncate “24/12/2019 23:33:33" ( one ) ✓

# 2. Nexus Parsing

# 2.1 Conditional Evaluation

# 2.1 Conditional Evaluation

one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | | | | |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one | ▯ | ▯ | ▯ |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ǁ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | ▯ | ▯ | ▯ |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) || (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | [0] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | [0] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | [0] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ǁ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | [0] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | [0,3] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | [0,3] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ǁ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| zero | one and | [0,3] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | [0,3] | [zero] | [[>]] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| one | one and | [0,3] | [zero] | [[>]] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| one | one and | [0] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| one | one and | [0] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) || (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | [0] | [one] | [<] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019-23:33:33”(one)) ǁ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| truncate | one and | [0] | [one] | [<] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) ∥ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| truncate “24/03/2019 23:33:33” | one and | [0] | [one] | [<] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) ∥ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| truncate “24/03/2019 23:33:33” | one and | [0] | [one] | [<] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) ∥ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| truncate “24/03/2019 23:33:33” one | one and | [0] | [one] | [<] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| truncate “24/03/2019 23:33:33” one | one and | [0] | [one] | [<] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) || (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| truncate “24/03/2019 23:33:33” one | one and | [0] | [one] | [<] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | [0] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) ǁ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | [0] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) ∥ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| zero | one and | [0] | [] | [] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and | [0] | [zero] | [{<=}] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| one | one and | [0] | [zero] | [{<=}] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) ‖ (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| one | one and | [0] | [zero] | [{<=}] |

# 2.1 Conditional Evaluation

**one and if(((if(zero[>]one) {zero} else {one}) [<] truncate ”24/03/2019 23:33:33”(one)) || (zero{<=}one)) {zero} else {give(one)}**

| ifCondition | contractString | NOPS | FPS | COPS |
|---|---|---|---|---|
| | one and zero | [] | [] | [] |

# 2.2 Contract Decomposition

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| | | | |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| truncate | [] | [] | [] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| truncate “24/12/2019 23:33:33” | [] | [] | [] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| | [truncate “24/12/2019 23:33:33”] | [)] | [] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| scaleK | [truncate “24/12/2019 23:33:33”] | [)] | [] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| scaleK 10 | [truncate “24/12/2019 23:33:33”] | [)] | [] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| | [truncate “24/12/2019 23:33:33”, truncate “24/12/2019 23:33:33” (scaleK 10] | [),))] | [] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| one | [truncate “24/12/2019 23:33:33”, truncate “24/12/2019 23:33:33” (scaleK 10] | [),))] | [] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| | [truncate “24/12/2019 23:33:33”] | [)] | [truncate “24/12/2019 23:33:33” (scaleK 10 (one))] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| | [truncate “24/12/2019 23:33:33”] | [)] | [truncate “24/12/2019 23:33:33” (scaleK 10 (one))] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| | [truncate “24/12/2019 23:33:33”] | [)] | [truncate “24/12/2019 23:33:33” (scaleK 10 (one))] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| scaleK | [truncate “24/12/2019 23:33:33”] | [)] | [truncate “24/12/2019 23:33:33” (scaleK 10 (one))] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| scaleK 7 | [truncate “24/12/2019 23:33:33”] | [)] | [truncate “24/12/2019 23:33:33” (scaleK 10 (one))] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| | [truncate “24/12/2019 23:33:33”, truncate “24/12/2019 23:33:33” (scaleK 7] | [),))] | [truncate “24/12/2019 23:33:33” (scaleK 10 (one))] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| one | [truncate “24/12/2019 23:33:33”, truncate “24/12/2019 23:33:33” (scaleK 7] | [),))] | [truncate “24/12/2019 23:33:33” (scaleK 10 (one))] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| | [truncate “24/12/2019 23:33:33”, truncate “24/12/2019 23:33:33” (scaleK 7] | [),))] | [truncate “24/12/2019 23:33:33” (scaleK 10 (one)), truncate “24/12/2019 23:33:33” (scaleK 7 (one))] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| zero | [truncate “24/12/2019 23:33:33”, truncate “24/12/2019 23:33:33” (scaleK 7] | [),))] | [truncate “24/12/2019 23:33:33” (scaleK 10 (one)), truncate “24/12/2019 23:33:33” (scaleK 7 (one))] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| | [truncate “24/12/2019 23:33:33”] | [)] | [truncate “24/12/2019 23:33:33” (scaleK 10 (one)), truncate “24/12/2019 23:33:33” (scaleK 7 (one)), truncate “24/12/2019 23:33:33” (scaleK 7 (zero))] |

# 2.2 Contract Decomposition

**truncate “24/12/2019 23:33:33” (scaleK 10(one) and (scaleK 7 (one and zero)))**

| contractString | parseStack | CPS | resultArr |
|---|---|---|---|
| | [] | [] | [truncate “24/12/2019 23:33:33” (scaleK 10 (one)), truncate “24/12/2019 23:33:33” (scaleK 7 (one)), truncate “24/12/2019 23:33:33” (scaleK 7 (zero))] |

# 2.3 Compilation into IR

# 2.3 Compilation into IR

- uses language identities (ie. *give(give(c))* == *c*)
- standardised contract representation
- contract processing optimisation
- creates *Contract* class instance
- IR is easily executable by Rust smart contract

# 2.3 Compilation into IR

scaleK 100 (give (get (scaleK 10 (give (one))))) => scaleK 1000 (get (one))

scaleK 5 (zero) => zero

# 2.4 Rust Contract Execution

# 2.5 Move IR Code Creation

# 2.5 Move IR Code Creation

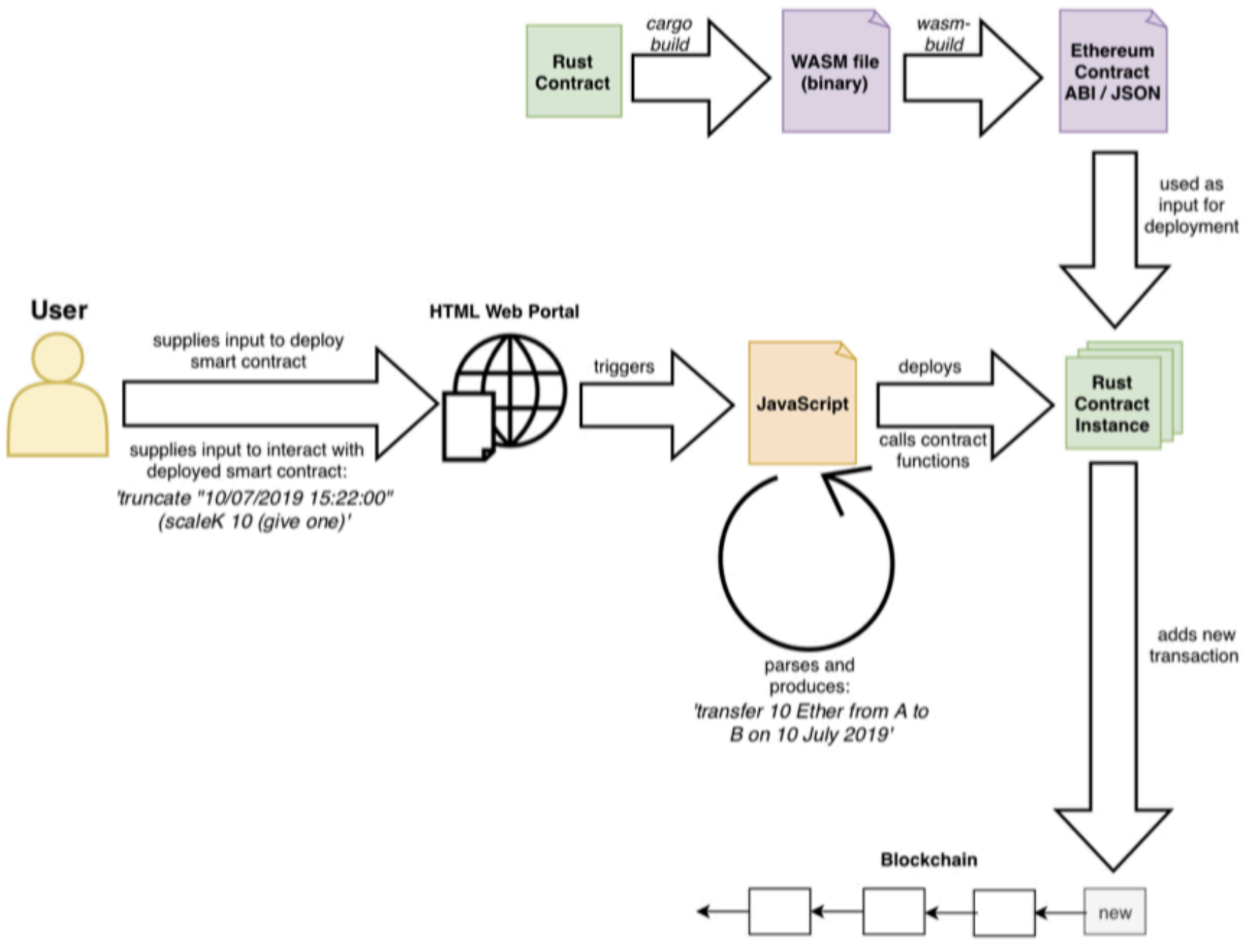
- downloaded onto user's machine
- executable via Libra command line interface
- Nexus as Move IR source language?

# 2.5 Move IR Code Creation

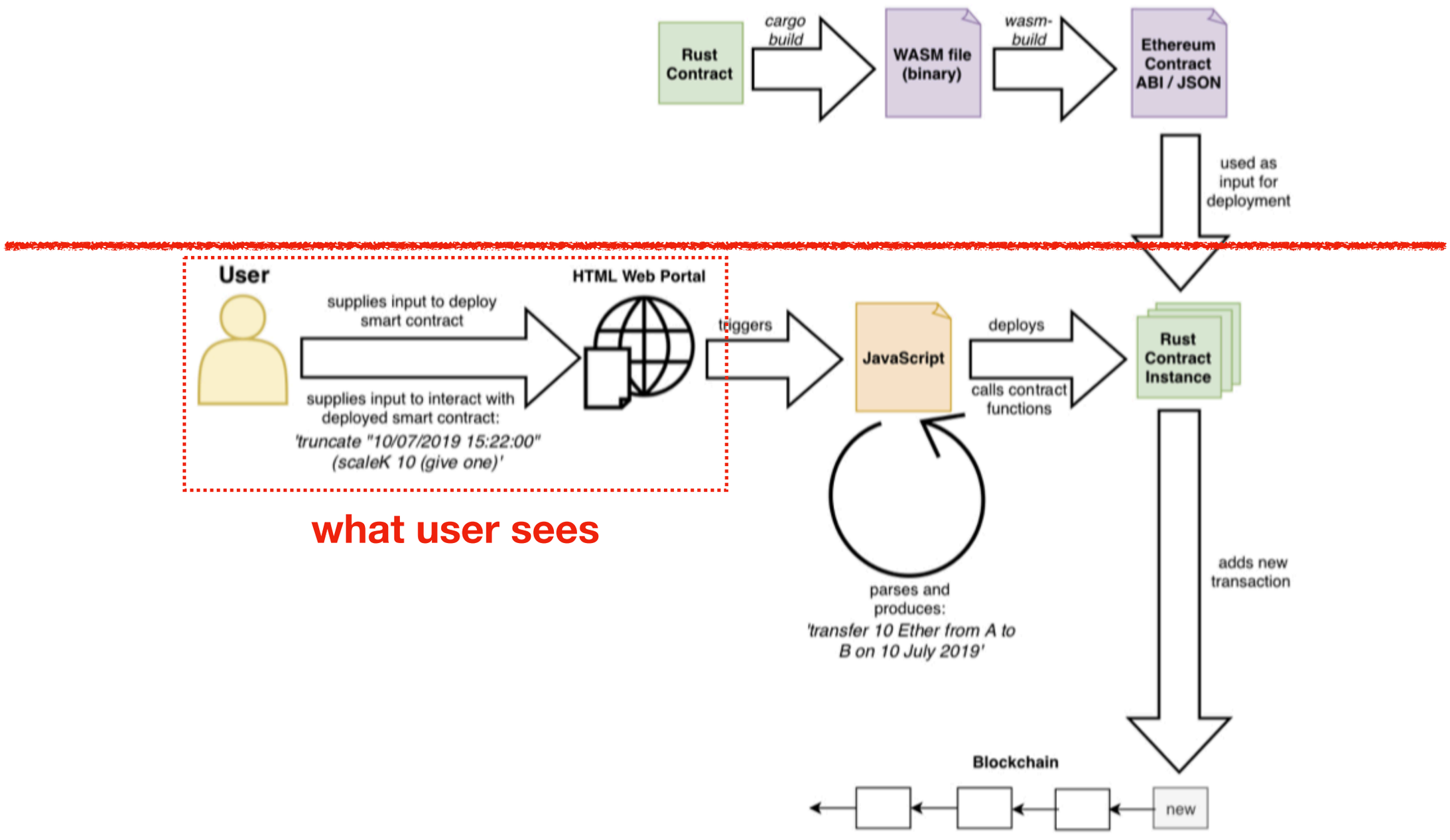
```
//! no-execute
import 0x0.LibraAccount;
import 0x0.LibraCoin;

main(payee: address) {
      let coin: R#LibraCoin.T;
      let account_exists: bool;
      let recipient: address;
      let sender: address;
      sender = 0x7f023262356b002a4b7deb7ce057eb8b1aabb427;
      recipient = 0x004ec07d2329997267ec62b4166639513386f32e;
      coin = LibraAccount.withdraw_from_sender(1);
      account_exists = LibraAccount.exists(copy(recipient));
      if (!move(account_exists)) {
           create_account(copy(recipient));
      }
      LibraAccount.deposit(move(recipient), move(coin));
      return;
}
```

# 3. System Architecture

# 3. System Architecture



# 3. System Architecture

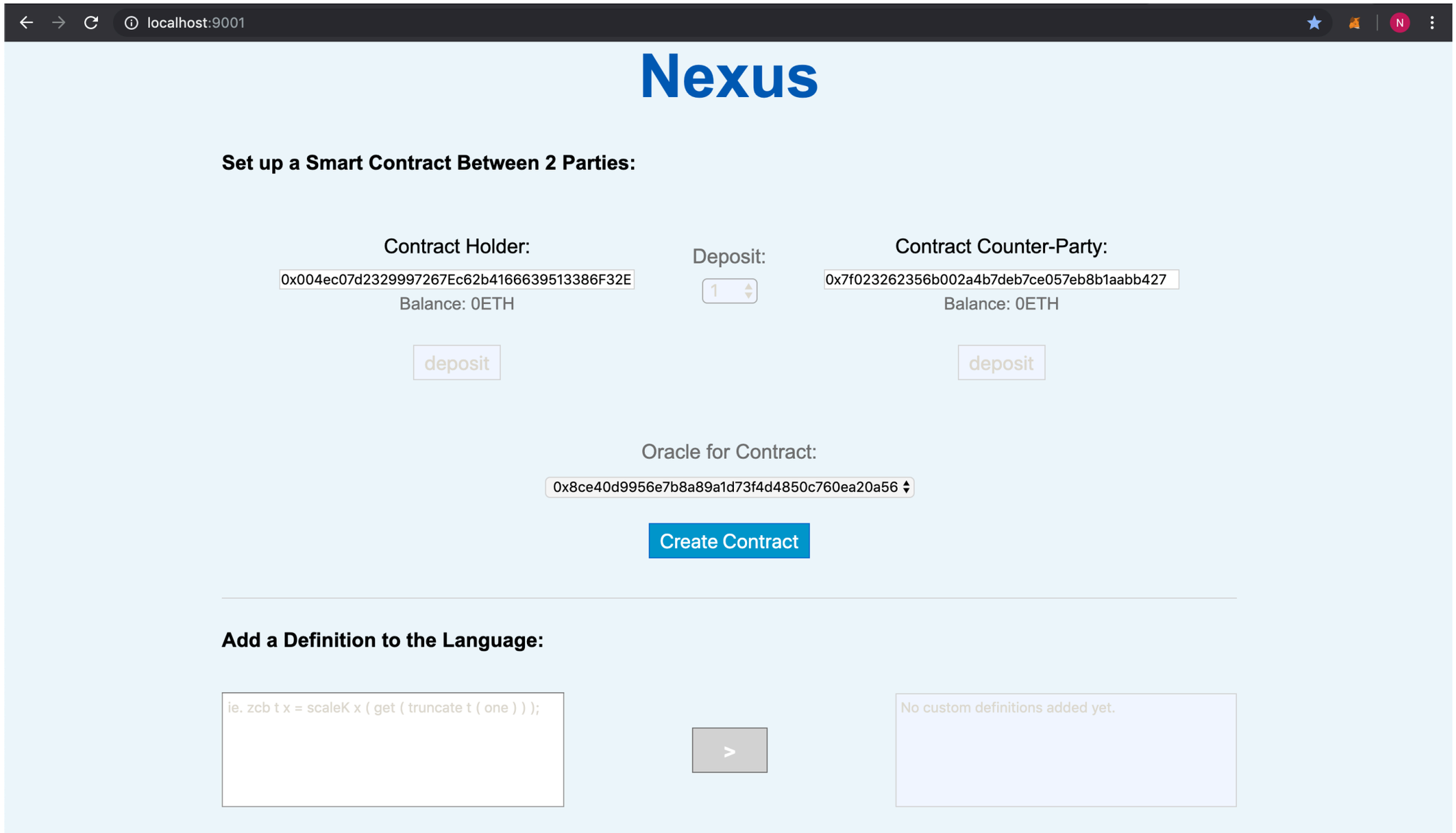


# 3. System Architecture

- isolation of high-level Nexus code from lower-level Rust smart contract code
- no smart contract code required to be written by end user
- enhanced portability, limited security threats, low tx costs

# 4. Web Application

# 4. Web Application



# 4.1 Collateral

- first step when using web app
- backup / insurance for both parties to receive funds promised
- can only add new contract if holding enough Ether to execute pending transactions + newly added transaction at any given time in future

  ie. abs value of new tx (negative from sender POV) < ( sender current balance - sum of all maximum absolute values of all pending transactions (from sender POV))

# 4.1 Collateral

**Set up a Smart Contract Between 2 Parties:**

Contract Holder:

0x004ec07d2329997267Ec62b4166639513386F32E

Balance: 0ETH

deposit

Deposit:

1

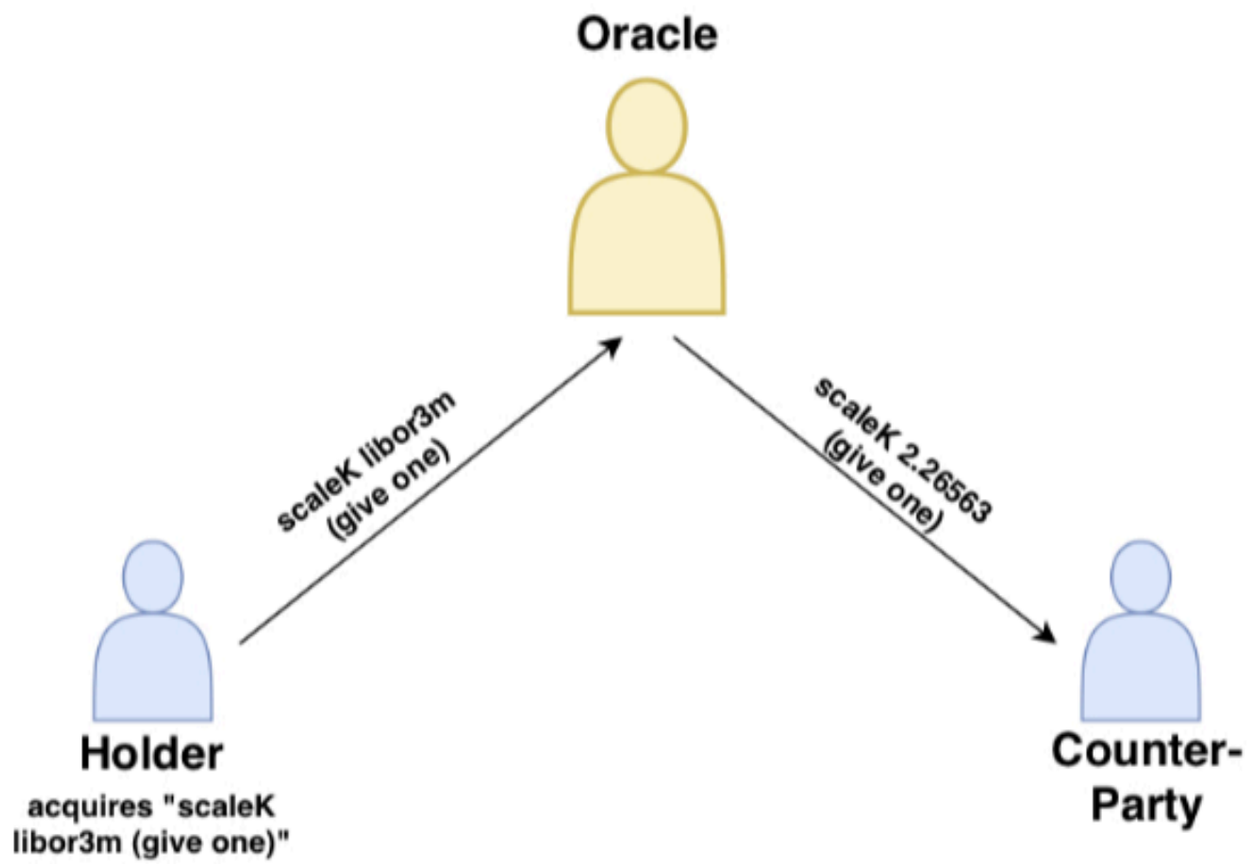
Contract Counter-Party:

0x7f023262356b002a4b7deb7ce057eb8b1aabb427

Balance: 0ETH

deposit

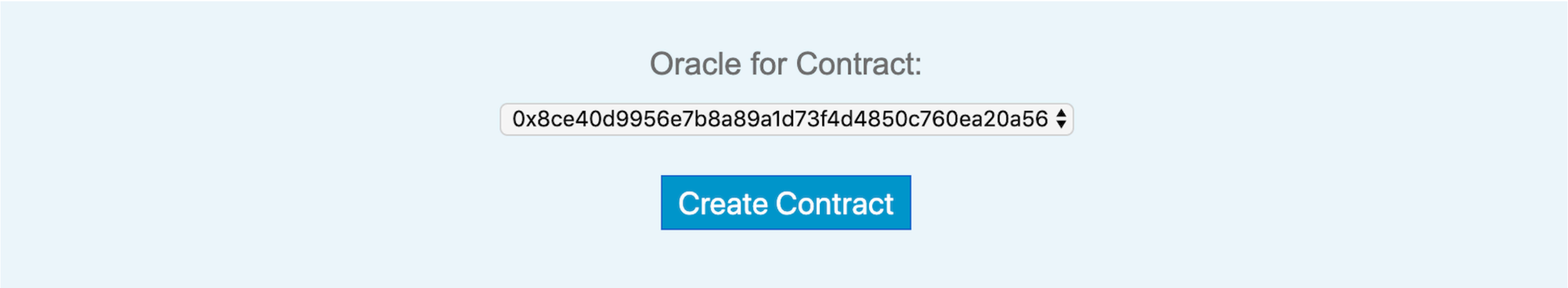
# 4.2 Oracle


Oracle
scaleK libor3m
(give one)
scaleK 2.26563
(give one)
Holder
acquires "scaleK
libor3m (give one)"
Counter-
Party

# 4.2 Oracle

- contract address
- agreed upon prior creation of contract
- provides observable values for contracts at time of acquirement
- no dispute between parties about observable values

# 4.2 Oracle


Oracle for Contract:
0x8ce40d9956e7b8a89a1d73f4d4850c760ea20a56
Create Contract

# 4.3 Extensibility

- custom-defined combinators composed of multiple already existing definitions ie. *andGive = and give;*
- enhanced usability by allowing users to define frequently used contracts such as *Zero-Coupon Discount Bond*, *European Options* etc.
- map maintains custom definitions
- custom definitions are replaced before decomposition

# 4.3 Extensibility

**Add a Definition to the Language:**

```
ie. zcb t x = scaleK x ( get ( truncate
t ( one ) ) );
```

>

```
andGive = and give
```

# 4.4 Contract Choices

- displayed when disjunction contract is being decomposed ie. *one or zero*
- holder must make this choice before chosen subcontract is made visible in web app and added to list of pending contracts — usability vs performance

# 4.4 Contract Choices

**Construct Smart Contract Transactions:**

( zero or give one ) or ( ( scaleK 10 ( one ) ) or zero )

Make Transaction

Contract choice:

( zero or give one )

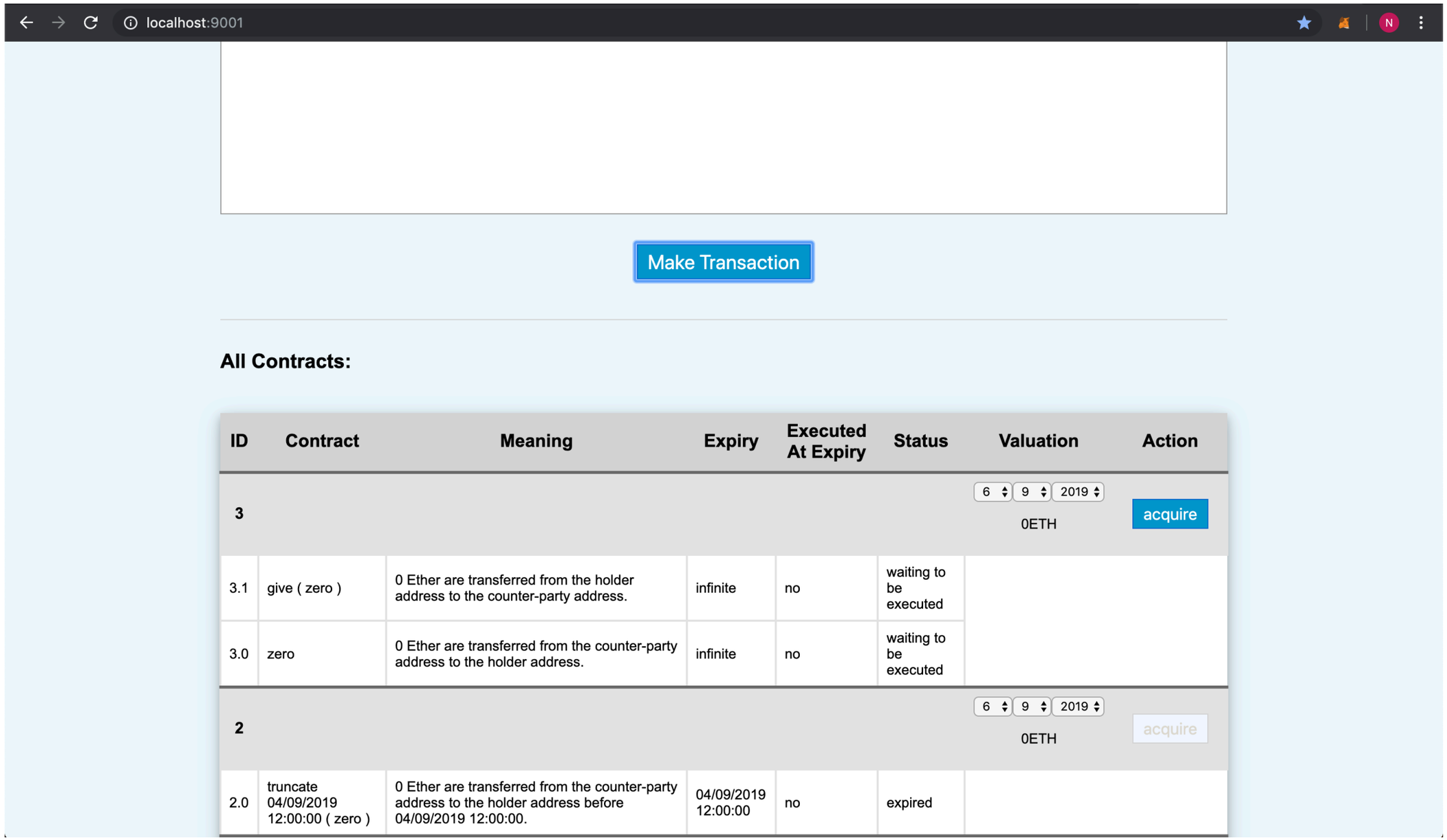
OR

( ( scaleK 10 ( one ) ) or zero )

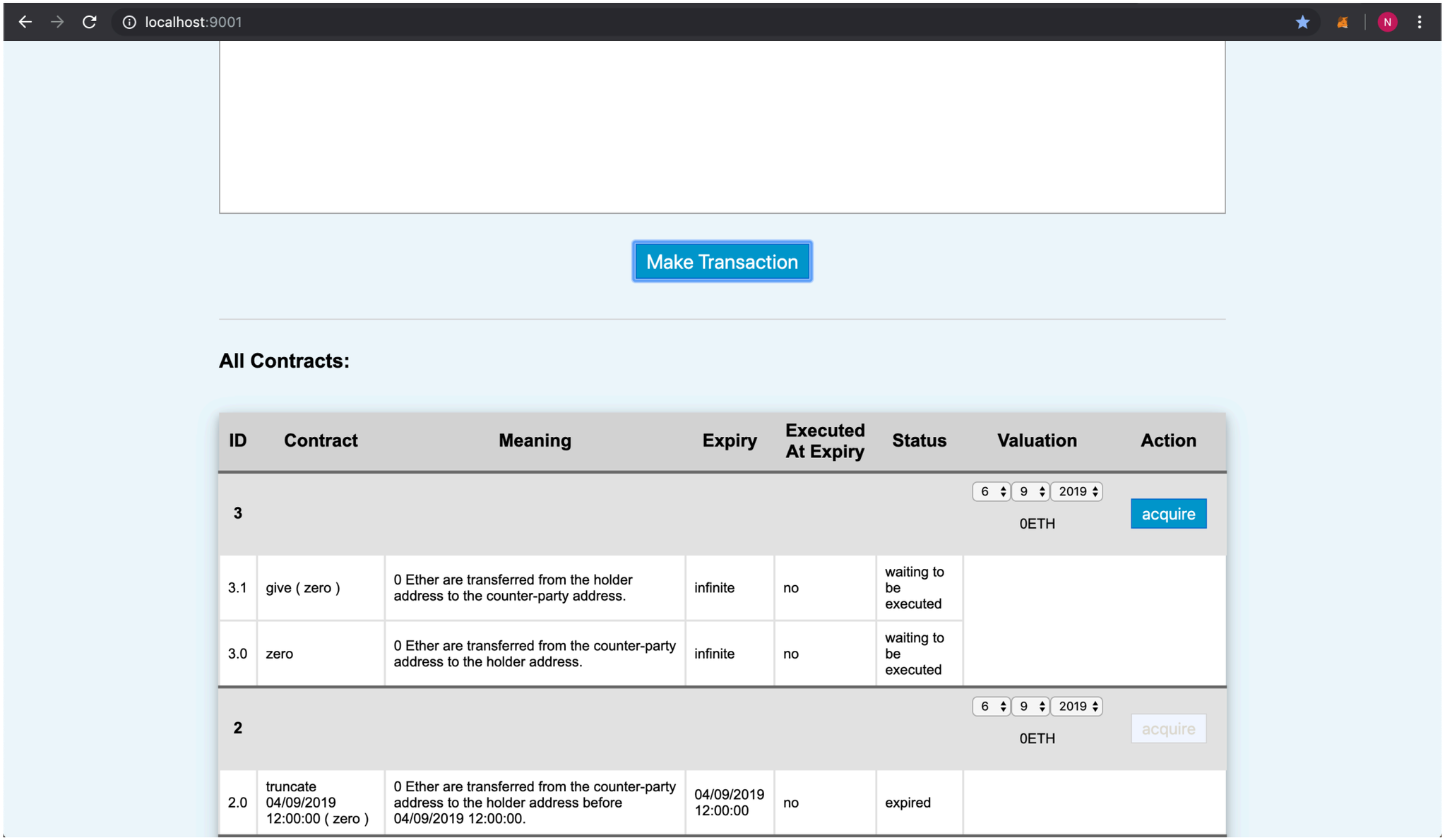
# 4.5 Contract Valuation

- valuation model suggested by Peyton Jones et al. only abstract and not implemented in a real-world financial setting
- Nexus: allows users to evaluate pending smart contracts for any given time in the future
- horizon, value, domination
- can be used for transformation optimisations to allow contracts to be “valued more cheaply”
- helps users understand more about evolution of contracts during their life & learn more about general market activity
- offers way of simulating + visualising different transaction orderings in the web app

# 4.5 Contract Valuation

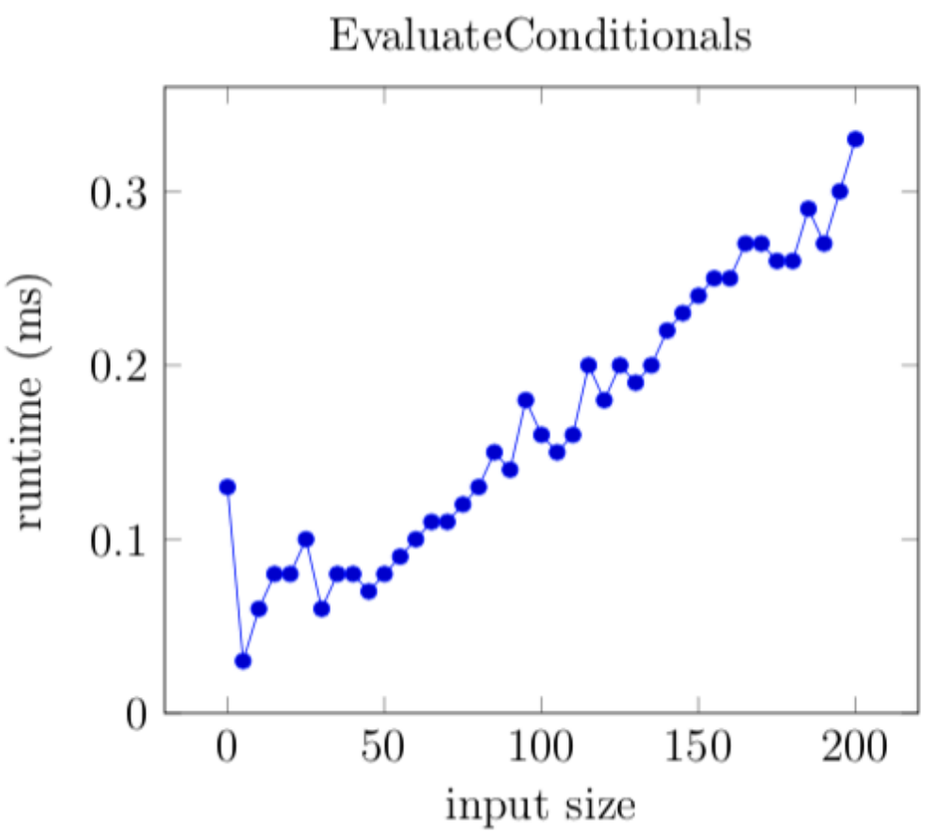


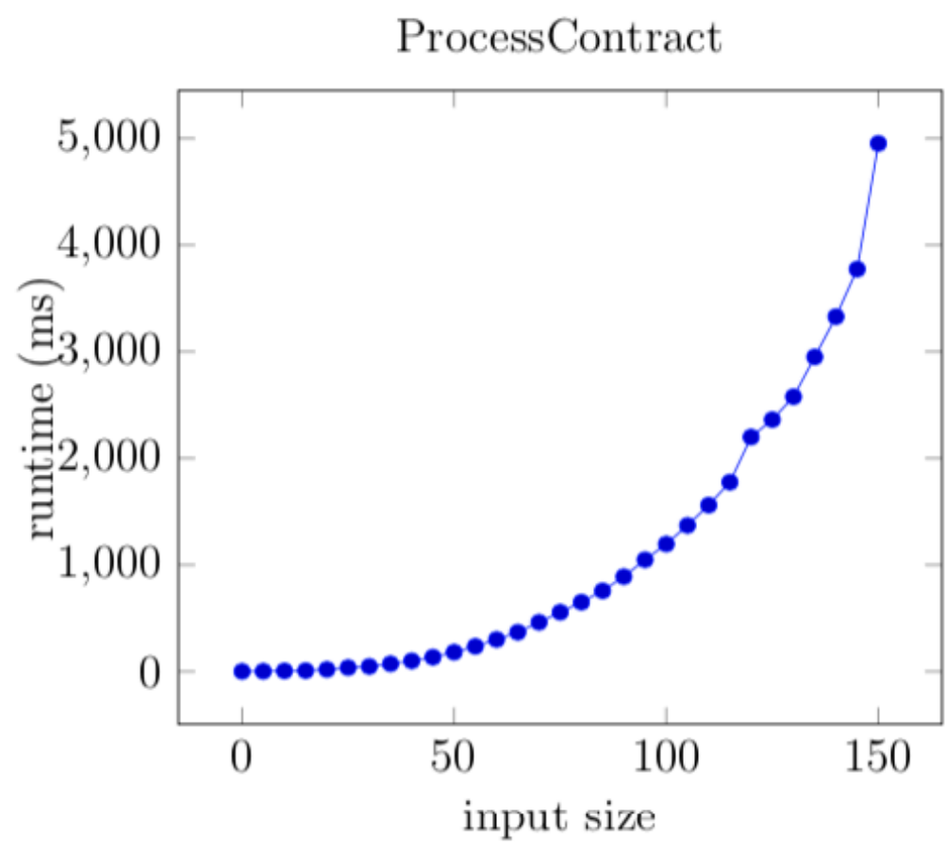
# 4.6 Acquirement & Expiration



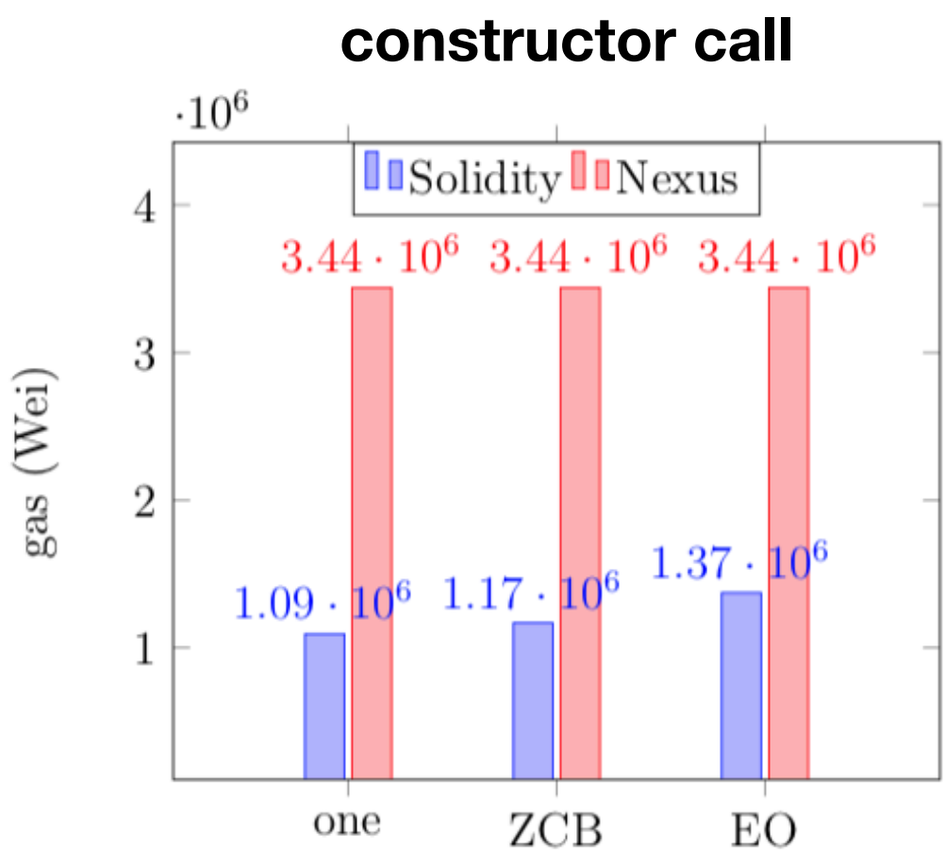
# 5. Results

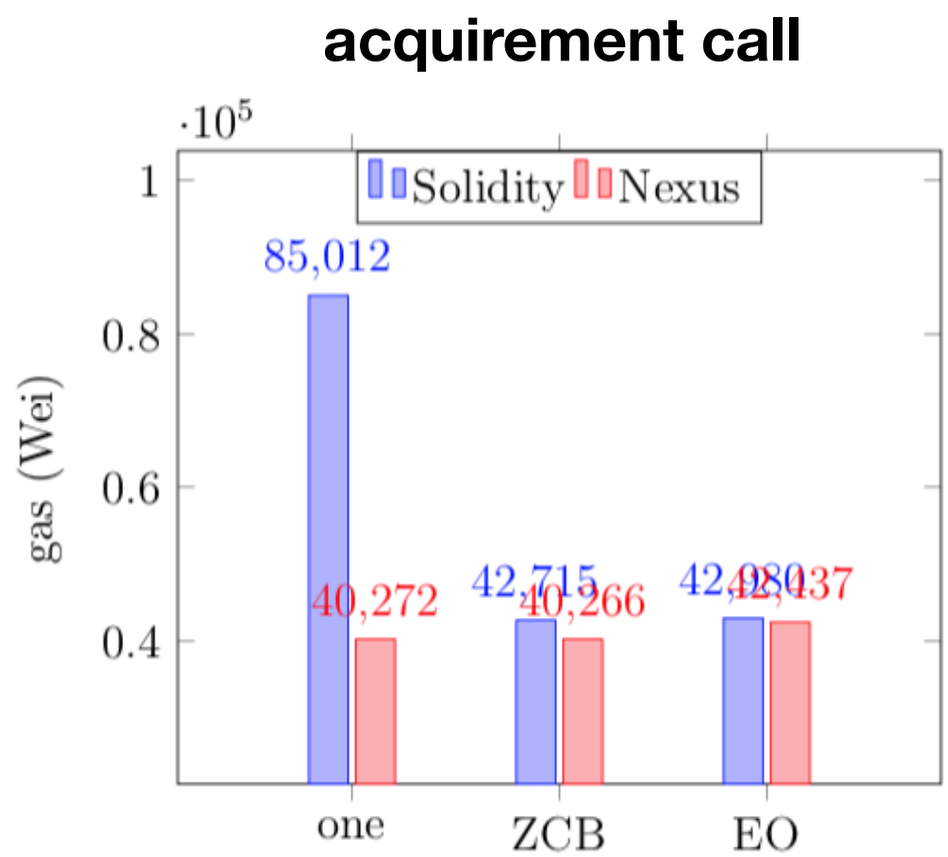
# 5.1 Performance



# 5.2 Costs


constructor call
$\cdot 10^6$
Solidity
Nexus
gas (Wei)
4
3
2
1
$3.44 \cdot 10^6$
$3.44 \cdot 10^6$
$3.44 \cdot 10^6$
$1.09 \cdot 10^6$
$1.17 \cdot 10^6$
$1.37 \cdot 10^6$
one
ZCB
EO


acquirement call
$\cdot 10^5$
Solidity
Nexus
gas (Wei)
1
0.8
0.6
0.4
85,012
40,272
42,715
40,266
42,980
42,437
one
ZCB
EO

# 5.3 Usability

Nexus: one

Solidity:

D. Dean, “Smartfin - implementing a financial domain-specific language for smart contracts,”

```
pragma solidity >=0.4.22 <0.6.0;

// Represents the contract 'one'
contract One {
    // Static values
    int256 MAX_INT256 = int256(~(uint256(1) << 255));
    int256 MIN_INT256 = int256(uint256(1) << 255);

    // The contract holder
    address holder;

    // The counter-party
    address counterParty;

    // The stakes of the holder and counter-party
    mapping(address => int256) stakes;

    // Whether or not this contract has been acquired
    bool acquired;

    // Constructor, takes the contract holder address
    constructor(address contractHolder) public {
        require(
            contractHolder != msg.sender,
            "Holder and counter-party cannot have the same address."
        );
        // Set the holder and counter-party
        holder = contractHolder;
        counterParty = msg.sender;

        // Initialise stakes to 0 and acquired to false
        stakes[counterParty] = 0;
        stakes[holder] = 0;
        acquired = false;
    }

    // Only allows the holder or counter-party to call a function
    modifier onlyParties() {
        require(
            msg.sender == counterParty || msg.sender == holder,
            "This function can only be called by the holder or the counter-party."
        );

        _;
    }

    // Returns the balance of one of the two parties
    function getBalance(bool holderBalance) public view returns (int256) {
        if (holderBalance) {
            return stakes[holder];
        } else {
            return stakes[counterParty];
        }
    }

    // Acquires this contract
    function acquire() public {
        require(
            msg.sender == holder,
            "Only the holder may call this function."
        );
        require(
            !acquired,
            "This function can only be called before acquisition."
        );

        acquired = true;

        // Update balances
        transferToHolder(1);
    }

    // Stake Ether in the contract
    function stake() public payable onlyParties() {
        require(
            uint256(MAX_INT256) >= msg.value,
            "Value being staked is too big to be stored as an int256 value."
        );

        // Update balance
        stakes[msg.sender] = safeAddSigned(stakes[msg.sender], int256(msg.value));
    }

    // Withdraw Ether from the contract
    function withdraw(uint64 amount) public onlyParties() {
        require(
            address(this).balance > 0,
            "Contract does not have enough funds."
        );
        require(
            stakes[msg.sender] > 0,
            "The caller does not have enough stake."
        );

        uint64 finalAmount = amount;

        // Clamp withdrawal amount to total contract balance
        if (address(this).balance < finalAmount) {
            finalAmount = uint64(address(this).balance);
        }

        // Clamp withdrawal amount to party's balance
        if (stakes[msg.sender] < finalAmount) {
            finalAmount = uint64(stakes[msg.sender]);
        }

        // Adjust balance first to prevent re-entrancy bugs
        stakes[msg.sender] = safeSubSigned(stakes[msg.sender], int256(finalAmount));

        // Send Ether (with no gas)
        msg.sender.call.value(finalAmount).gas(0);
    }

    // Transfers the given amount from the holder to the counter-party
    function transferToHolder(int256 amount) private {
        stakes[holder] = safeAddSigned(stakes[holder], amount);
        stakes[counterParty] = safeSubSigned(stakes[counterParty], amount);
    }

    // Add two signed integers if no overflow or underflow can occur
    function safeAddSigned(int256 a, int256 b) private view returns (int256) {
        require(
            (b >= 0 && a <= MAX_INT256 - b) ||
            (b < 0 && a >= MIN_INT256 - b),
            "Integer overflow or underflow."
        );

        return a + b;
    }

    // Subtract one signed integer from another if no overflow or underflow can occur
    function safeSubSigned(int256 a, int256 b) private view returns (int256) {
        require(
            b != MIN_INT256,
            "Integer overflow or underflow."
        );

        return safeAddSigned(a, -b);
    }
}
```

# 5.3 Usability

- web app distributed among 10 external participants with different technical ability
- each was given 5 minutes to play around with app before filling out a survey and ranking criteria out of 10

| Comprehensibility | Functionality | Design |
|---|---|---|
| 8.2/10 | 9.8/10 | 8.4/10 |

# 6. Demo

# 7. Future Work

# 7.1 Nexus Language

- expand set of combinators
- apply to and evaluate in terms of additional use cases ie. *American Options*
- add more mathematical proofs about language
- implement *else if {* syntax
- add exchange & interest rates

# 7.2 Performance

- merge *EvaluateConditionals* and *Decompose* algorithms into one

- payment channel

# 7.3 Usability

- allow contracts between more than two parties (non-trivial)

- gas estimation

# 7.4 Libra Integration

- implement system for Libra (permissioned) & compare implementations in terms of usability, scalability, performance

# 8. Conclusion

# 8. Conclusion

- trivial to use (even for non-technical users)
- secure
- efficient
- low-cost
- portable
- traceability & transparency
- room for improvement