Introducing TxVM
Cathie Yun, Software Engineer
Background

Chain builds ledger services for financial institutions. Underlying technology: transactions on a blockchain. Supports high transaction volumes, issued assets, and smart contracts.
Existing approaches

Two popular blockchains use different models for transactions and contracts.
Bitcoin

Transactions and contracts are **declarative**.
Bitcoin

Programs are simple predicates. Resulting state is known before publication.
Ethereum

Transactions and contracts are imperative.
Ethereum

Contracts can call other contracts. Resulting state is unknown until tx is published.

deployed contracts

<table>
<thead>
<tr>
<th>nonce</th>
<th>gas price</th>
<th>start gas</th>
<th>destination</th>
<th>amount</th>
<th>signature</th>
</tr>
</thead>
</table>

data
# Introducing TxVM

<table>
<thead>
<tr>
<th>Feature</th>
<th>BTC</th>
<th>EVM</th>
<th>TxVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deterministic &amp; isolated</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
</tr>
<tr>
<td>Expressive language</td>
<td>✘</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Safe environment</td>
<td>✔</td>
<td>✘</td>
<td>✔</td>
</tr>
</tbody>
</table>
Transaction **is** the program: executes contracts, controls value flow and provides signatures.
Executing the program produces a **deterministic** transaction log.
First class **values** and **contracts** as a part of the VM type system.

1. Values and contracts have **constrained operations** that preserve a "law of conservation"

2. Values and contracts **must be cleared** from the VM by the end of execution.
First-class Values

Example ride-sharing transaction

1. Operations preserve "law of conservation"
2. Objects must be cleared from the VM by the end
First-class Values

Example ride-sharing transaction

1. Operations preserve "law of conservation"
2. All objects must be cleared from the VM
First-class Values

Example ride-sharing transaction

1. Operations preserve "law of conservation"
2. All objects must be cleared from the VM
First-class Values

Example ride-sharing transaction

1. Operations preserve "law of conservation"
2. All objects must be cleared from the VM
First-class Values

Example ride-sharing transaction

1. Operations preserve "law of conservation"
2. All objects must be cleared from the VM
First-class Values

Example ride-sharing transaction

1. Operations preserve "law of conservation"

2. All objects must be cleared from the VM
First-class Values

Impossible application violating rule #1

1. Operations preserve "law of conservation"
2. All objects must be cleared from the VM
First-class Values

Impossible application violating rule #1

1. Operations preserve "law of conservation"

2. All objects must be cleared from the VM
First-class Values

Impossible application violating rule #1

1. Operations preserve "law of conservation"

2. All objects must be cleared from the VM
First-class Values

Impossible application violating rule #1

1. Operations preserve "law of conservation"
2. All objects must be cleared from the VM
First-class Values

Invalid transaction violating rule #2

1. Operations preserve "law of conservation"

2. All objects must be cleared from the VM
First-class Values

Invalid transaction violating rule #2

1. Operations preserve "law of conservation"

2. All objects must be cleared from the VM
First-class Values
Invalid transaction violating rule #2

1. Operations preserve "law of conservation"
2. All objects must be cleared from the VM
First-class Values

Invalid transaction violating rule #2

1. Operations preserve "law of conservation"
2. All objects must be cleared from the VM
First-class Values

Invalid transaction violating rule #2

1. Operations preserve "law of conservation"
2. All objects must be cleared from the VM
First-class Values

Invalid transaction violating rule #2

1. Operations preserve "law of conservation"
2. All objects must be cleared from the VM
First-class Values

Invalid transaction violating rule #2

1. Operations preserve "law of conservation"
2. All objects must be cleared from the VM
First-class Values

Invalid transaction violating rule #2

1. Operations preserve "law of conservation"

2. All objects must be cleared from the VM
TxVM internals

1. VM parts
   stacks
   tx log
   runlimit
   code
TxVM internals

1. VM parts
   stacks
   tx log
   runlimit
   code

2. VM rules
   empty stacks
   tx is finalized
   runlimit not exceeded
   no failures
TxVM internals

1. VM parts
   - stacks
   - tx log
   - runlimit
   - code

2. VM rules
   - empty stacks
   - tx is finalized
   - runlimit not exceeded
   - no failures

3. Blockchain updates
   - all effects in tx log
   - remove inputs
   - add outputs
TxVM walkthrough

Alice’s $10 input call

Alice’s $5 input call

merge_values

Carol’s account contract call

finalize

<signatures>... call
TxVM walkthrough

Alice’s $10 input call
Alice’s $5 input call
merge_values
Carol’s account contract call
finalize
<signatures>... call

stack

tx log
Claim and open an input

- Alice’s $10
- Alice’s $5
- Carol’s account

merge_values

contract call

finalize

<signatures>... call

Input $10

Alice’s contract

$10

stack

tx log
Claim a second input

input call

input call

merge_values

contract call

contract call

finalize

<signatures>... call

Alice’s $10

Alice’s $5

Carol’s account

Input $10

Input $5

Alice's contract

$5

Alice's contract

$10

stack

tx log
Merge and split values

- Alice’s $10
  - input call
- Alice’s $5
  - input call
- Carol’s account
  - contract call

merge_values

finalize

<signatures>... call

$15
- Alice's contract
- Alice's contract

Input $10
Input $5

stack

tx log
Lock value in a new contract

Alice’s $10 input call

Alice’s $5 input call

merge_values

carol’s account contract call

finalize

<signatures>... call

Input $10

Input $5

Output $15

Alice’s contract

Alice’s contract

Alice’s contract

stack

tx log
Freeze modifications, compute transaction ID

- Alice’s $10 input call
- Alice’s $5 input call
- merge_values
- Carol’s account contract call
- finalize
- <signatures>... call

Input $10
Input $5
Output $15
Tx ID

Alice’s contract
Alice’s contract
Alice’s contract

stack

Tx ID

tx log
Clear remaining contracts

Alice’s $10
input call

Alice’s $5
input call

merge_values

Carol’s account
contract call

finalize

<signatures>... call

Input $10
Input $5
Output $15
Tx ID

Alice’s contract
Alice’s contract
Alice’s contract

stack
tx log
TxVM recap

- **Declared inputs**: Alice's $10, Alice's $5

- **Value flow**: Merge values

- **New outputs**: Carol's account

- **Freeze tx log**: Contract call

- **Clear remaining contracts**: Finalize, <signatures>... call
Diving into contracts

Input call

Alice’s $10

Alice’s $5

Input call

merge_values

Carol’s account

contract call

finalize

<signatures>... call

Input $10

Input $5

Output $15

Tx ID

Alice’s contract

Alice’s contract

stack

tx log
Signature contract
Checks a simple signature on transaction ID.

Get the Tx ID as calculated from the finalized tx log

Get Alice’s signature

Clear contract if valid, fail VM if invalid
Diving into contracts

input call

input call

merge_values

contract call

finalize

<signatures>... call

Alice’s $10

Alice’s $5

Carol’s account

C1

C2

C3

Input $10

Input $5

Output $15

Tx ID

Alice’s contract

Alice’s contract

stack

tx log
Unspent output contract

Unlocks the value and defers a checksig contract.

<$10> put_value
[txid <Alice’s pubkey> get_sig checksig verify]
put_contract
Unspent output contract

Unlocks the value and defers a checksig contract.

txid <Alice’s pubkey> get_sig checksig verify

<$10> put_value

[ C1 ] put_contract

c2

txvm stack
Unspent output contract

Unlocks the value and defers a checksig contract.

Release the $10 value
Unspent output contract

Unlocks the value and defers a checksig contract.

Alice's contract

C2

<$10> put_value

[ C1 ] put_contract

Release the checksig contract

$10

txvm stack
Diving into contracts

Alice’s $10 input call

Alice’s $5 input call

merge_values

Carol’s account contract call

finalize

<signatures>... call

Input $10

Input $5

Output $15

Tx ID

Alice's contract

Alice's contract

stack

tx log
New output contract

Locks the value and creates an output.

get_value
    [put_value
        [txid <pubkey> get_sig checksig verify]
    ]
put_contract]
output
New output contract
Locks the value and creates an output.

```plaintext
[put_value
  [txid <pubkey> get_sig checksig verify]
  put_contract]

[get_value
  [output]]
```
New output contract

Locks the value and creates an output.

get_value

\[ \text{output} \]

Input $10
Input $5

$15

txvm stack

tx log
New output contract

Locks the value and creates an output.

Receive the value to be locked

get_value

[ C2 ] output

$10

Input $10

$5

Input $5

$15

txvm stack

$15

tx log
New output contract

Locks the value and creates an output.

Output contract C2 with $15 value

get_value

Output $15

Input $5

Input $10

Output contract C2 with $15 value

txvm stack

tx log
Diving into contracts

- Alice’s $10 input call
- Alice’s $5 input call
- merge_values
- Carol’s account contract call
- finalize
- <signatures>... call

Input $10
Input $5
Output $15
Tx ID
## TxVM instruction set

<table>
<thead>
<tr>
<th>Opcode</th>
<th>mnemonics</th>
<th>operand(s)</th>
<th>description</th>
<th>byte size</th>
<th>TxVM size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (false)</td>
<td>16</td>
<td>int</td>
<td>nonce</td>
<td>verify</td>
<td>eq</td>
</tr>
<tr>
<td>1 (true)</td>
<td>17</td>
<td>add</td>
<td>merge</td>
<td>jumpif</td>
<td>dup</td>
</tr>
<tr>
<td>2</td>
<td>18</td>
<td>neg</td>
<td>split</td>
<td>exec</td>
<td>drop</td>
</tr>
<tr>
<td>3</td>
<td>19</td>
<td>mul</td>
<td>issue</td>
<td>call</td>
<td>peek</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>div</td>
<td>retire</td>
<td>yield</td>
<td>tuple</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>mod</td>
<td>amount</td>
<td>wrap</td>
<td>untuple</td>
</tr>
<tr>
<td>6</td>
<td>22</td>
<td>gt</td>
<td>assetid</td>
<td>input</td>
<td>len</td>
</tr>
<tr>
<td>7</td>
<td>23</td>
<td>not</td>
<td>anchor</td>
<td>output</td>
<td>field</td>
</tr>
<tr>
<td>8</td>
<td>24</td>
<td>and</td>
<td>vmhash</td>
<td>contract</td>
<td>encode</td>
</tr>
<tr>
<td>9</td>
<td>25</td>
<td>or</td>
<td>sha256</td>
<td>seed</td>
<td>cat</td>
</tr>
<tr>
<td>10</td>
<td>26</td>
<td>roll</td>
<td>sha3</td>
<td>self</td>
<td>slice</td>
</tr>
<tr>
<td>11</td>
<td>27</td>
<td>bury</td>
<td>checksum</td>
<td>caller</td>
<td>bitnot</td>
</tr>
<tr>
<td>12</td>
<td>28</td>
<td>reverse</td>
<td>log</td>
<td>contractprog.</td>
<td>bitand</td>
</tr>
<tr>
<td>13</td>
<td>29</td>
<td>get</td>
<td>peeklog</td>
<td>timerange</td>
<td>bitor</td>
</tr>
<tr>
<td>14</td>
<td>30</td>
<td>put</td>
<td>txid</td>
<td>prv</td>
<td>bitxor</td>
</tr>
<tr>
<td>15</td>
<td>31</td>
<td>depth</td>
<td>finalize</td>
<td>ext</td>
<td>0 bytes</td>
</tr>
</tbody>
</table>

### Opcode Table

- **smallints**: 0x09-0x1f
- **ints**: 0x20-0x2f
- **stack**: 0x30-0x37
- **values**: 0x38-0x3b
- **crypto**: 0x3c-0x3f
- **tx**: 0x40-0x4d
- **control flow**: 0x4e-0x4f
- **extension**: 0x50-0x5f
- **data**: 0x60-0x6f
- **pushdata**: 0x70-0x7f
Conclusion

- TxVM is a **new model** for blockchain transactions.
- Borrows **good parts** from both Bitcoin and Ethereum.
- Powerful abstractions: first-class **values** and **contracts**.
- Safe and straightforward **language** for writing smart contracts.
- **Open source & whitepaper** soon.
Cathie Yun
cathie@chain.com

Bob Glickstein
bob@chain.com

Dan Robinson
@danrobinson

Oleg Andreev
@oleganza