How I Used Formal Methods for Blockchain Smart Contract Security

Daejun Park

March 1, 2022

Background

Developing and applying practical formal methods for improving quality, reliability, and security of production systems software

Background

- Master's thesis: scalable, context-/path-sensitive program analysis
- 

(2008–2011): static analysis for detecting memory safety bugs for embedded systems software
- PhD thesis: language-parametric formal methods
- 

(2018–Present): formal verification of blockchain consensus protocols and smart contracts

Background

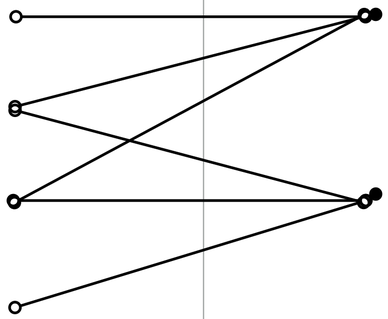
- Master's thesis: scalable, context-/path-sensitive program analysis
- Sparrow (2008–2011): static analysis for detecting memory safety bugs for embedded systems software
- PhD thesis: language-parametric formal methods
- runtime verification (2018–Present): formal verification of blockchain consensus protocols and smart contracts

Language-parametric Formal Methods

PhD thesis: language-parametric formal methods

Executable Semantics of Production Languages

- JavaScript [PLDI'15]
- Ethereum VM [CSF'16]
- x86 [PLDI'19]
- LLVM [ASPLOS'21]
- …



Language-parametric Formal Methods

Distinguished Paper Award

- Deductive verification [OOPSLA'16, FSE'18, CAV'20]
- Translation validation [ASPLOS'21]
- …

Developed once, and reused for all others

PhD thesis: language-parametric formal methods

- Complete
- Extensively tested
- Found bugs in language implementations

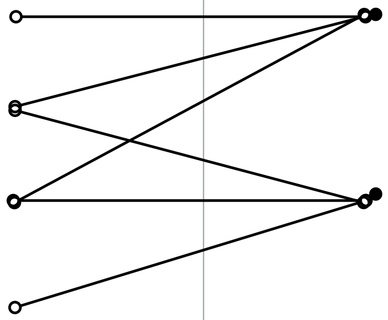
Executable Semantics of Production Languages

- JavaScript [PLDI'15]
- Ethereum VM [CSF'16]
- x86 [PLDI'19]
- LLVM [ASPLOS'21]
- …

Language-parametric Formal Methods

Distinguished Paper Award

- Deductive verification [OOPSLA'16, FSE'18, CAV'20]
- Translation validation [ASPLOS'21]
- …



Developed once, and reused for all others

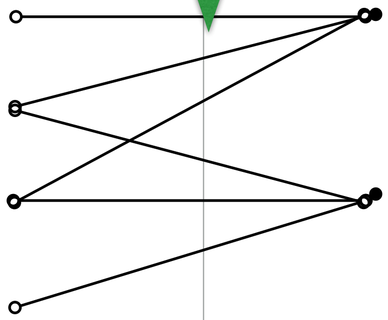
PhD thesis: language-parametric formal methods

- Complete
- Extensively tested
- Found bugs in language implementations

Executable Semantics of Production Languages

- JavaScript [PLDI'15]
- Ethereum VM [CSF'16]
- x86 [PLDI'19]
- LLVM [ASPLOS'21]
- …

- JavaScript verifier
- Ethereum smart contract verifier



Language-parametric Formal Methods

Distinguished Paper Award

- Deductive verification [OOPSLA'16, FSE'18, CAV'20]
- Translation validation [ASPLOS'21]
- …

Developed once, and reused for all others

PhD thesis: language-parametric formal methods

- Complete
- Extensively tested
- Found bugs in language implementations

Executable Semantics of Production Languages

- JavaScript [PLDI'15]
- Ethereum VM [CSF'16]
- x86 [PLDI'19]
- LLVM [ASPLOS'21]
- …

- JavaScript verifier
- Ethereum smart contract verifier

- Translation validation for x86 backend of LLVM compiler pipeline

Language-parametric Formal Methods

Distinguished Paper Award

Deductive verification [OOPSLA'16, FSE'18, CAV'20]

Translation validation [ASPLOS'21]

Developed once, and reused for all others

PhD thesis: language-parametric formal methods

- Complete
- Extensively tested
- Found bugs in language implementations

Executable Semantics of Production Languages

- JavaScript verifier
- Ethereum smart contract verifier

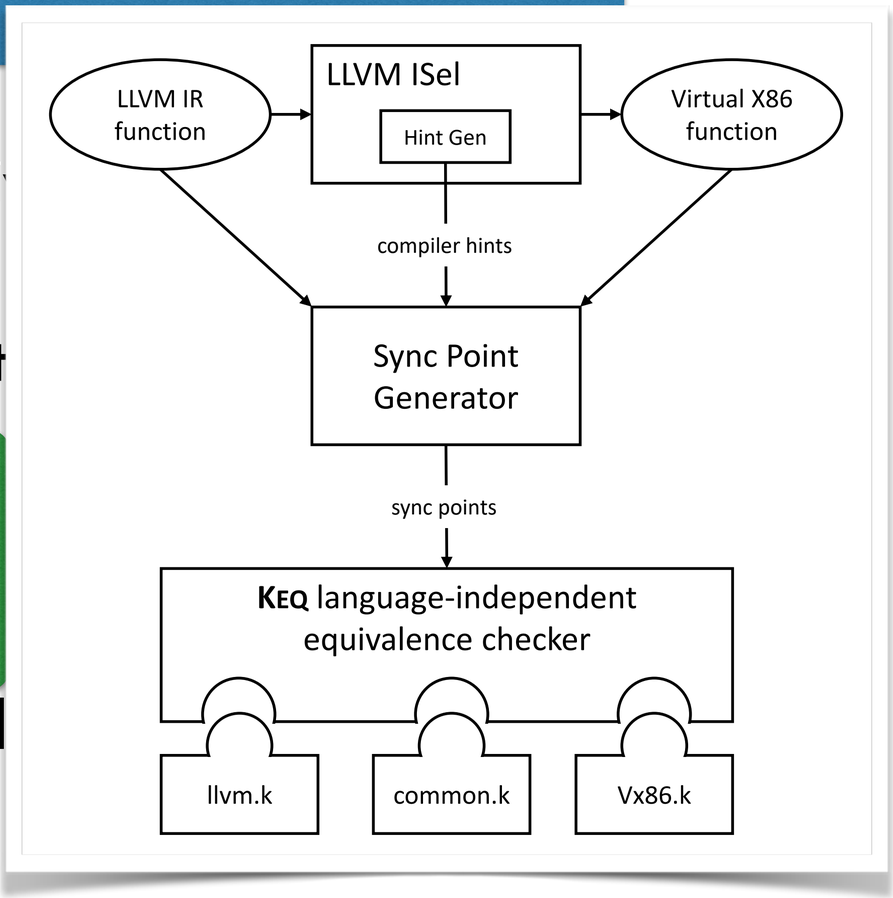
Language-parametric Formal Methods

- JavaScript [PLDI'15]
- Ethereum VM [CSF'16]
- x86 [PLDI'19]
- LLVM [ASPLOS'21]
- …

Deducti

Translat

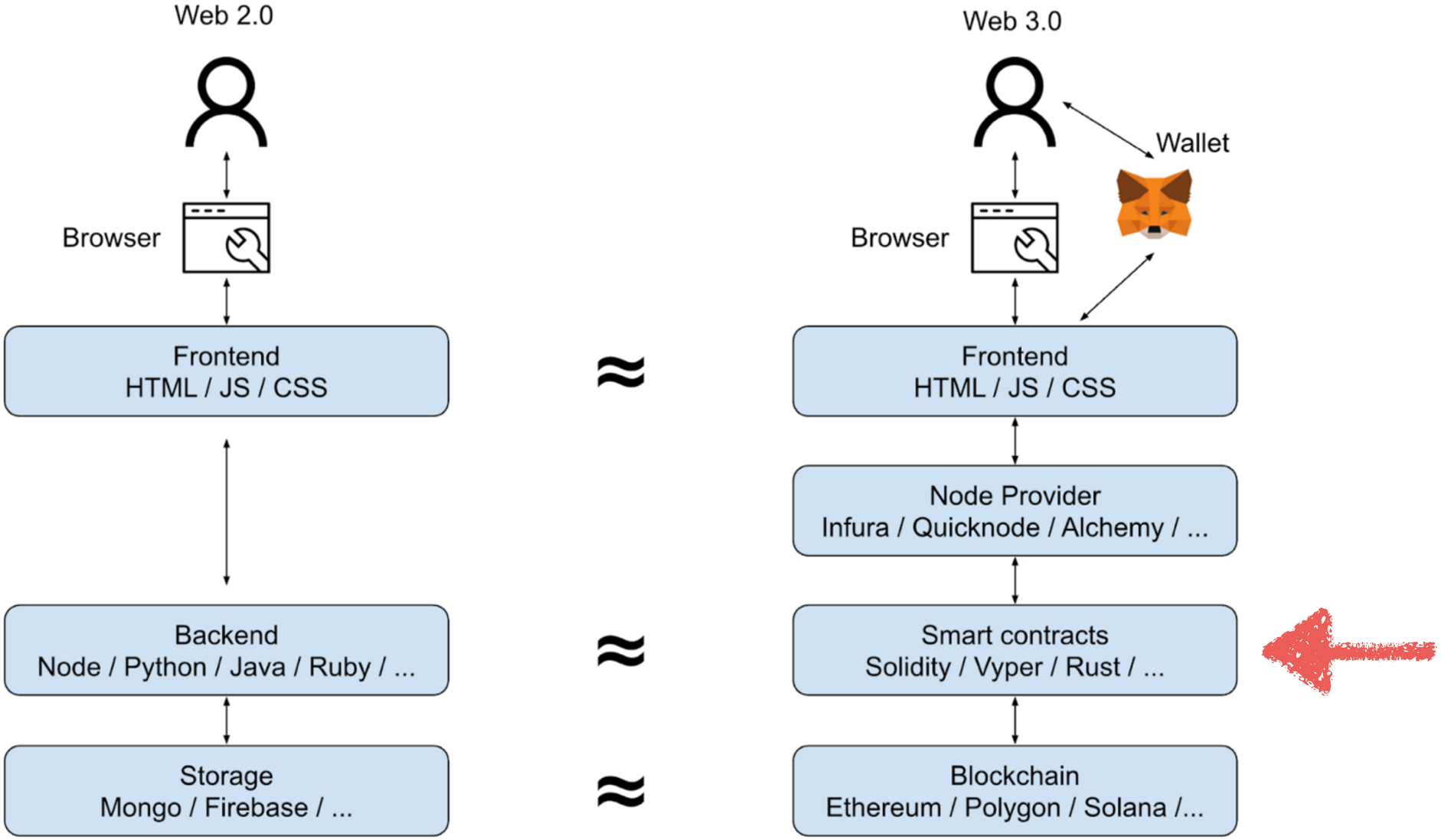
- Translation validation for x86 backend of LLVM compiler pipeline



Developed once, and reused for al

Smart Contract Verification

Smart contracts



* Image by https://towardsdatascience.com/decoding-ethereum-smart-contract-data-eed513a65f76

Smart contract example

```
 1 mapping(address => uint) balance;
 2
 3 function batchTransfer(address from, address[] to, uint amount) {
 4     uint total = amount * to.length;        // compute the total amount
 5
 6     if (balance[from] < total) revert();    // check if balance is enough
 7
 8     balance[from] -= total;                 // deduct the total amount
 9
10     for (int i = 0; i < to.length; i++) {
11         balance[to[i]] += amount;           // credit to each receiver
12     }
13 }
```

Smart contract example

```
 1 mapping(address => uint) balance;
 2
 3 function batchTransfer(address from, address[] to, uint amount) {
 4     uint total = amount * to.length;        // compute the total amount
 5
 6     if (balance[from] < total) revert();    // check if balance is enough
 7
 8     balance[from] -= total;                 // deduct the total amount
 9
10     for (int i = 0; i < to.length; i++) {
11         balance[to[i]] += amount;           // credit to each receiver
12     }
13 }
```

Smart contract example

```
 1 mapping(address => uint) balance;
 2
 3 function batchTransfer(address from, address[] to, uint amount) {
 4     uint total = amount * to.length;        // compute the total amount
 5
 6     if (balance[from] < total) revert();    // check if b
 7
 8     balance[from] -= total;                 // deduct the
 9
10     for (int i = 0; i < to.length; i++) {
11         balance[to[i]] += amount;           // credit to
12     }
13 }
```

* BatchTransfer Overflow (CVE-2018-10299): https://nvd.nist.gov/vuln/detail/CVE-2018-10299

Smart contract example

```
 1 mapping(address => uint) balance;
 2
 3 function transfer(address from, address to, uint amount) {
 4     uint newBalanceFrom, newBalanceTo;
 5
 6     if (balance[from] >= amount) {
 7         newBalanceFrom = balance[from] - amount; // compute new balance of sender
 8         newBalanceTo   = balance[to]   + amount; // compute new balance of receiver
 9     } else {
10         revert();
11     }
12
13     balance[from] = newBalanceFrom;  // update the sender's balance
14     balance[to]   = newBalanceTo;    // update the receiver's balance
15 }
```

Smart contract example

$$\{\texttt{from} \mapsto x \ * \ \texttt{to} \mapsto y \ * \ M\} \ \texttt{transfer}(\texttt{from}, \texttt{to}, v) \ \{\texttt{from} \mapsto (x - v) \ * \ \texttt{to} \mapsto (y + v) \ * \ M\}$$

```
 1 mapping(address => uint) balance;
 2
 3 function transfer(address from, address to, uint amount) {
 4     uint newBalanceFrom, newBalanceTo;
 5
 6     if (balance[from] >= amount) {
 7         newBalanceFrom = balance[from] - amount; // compute new balance of sender
 8         newBalanceTo   = balance[to]   + amount; // compute new balance of receiver
 9     } else {
10         revert();
11     }
12
13     balance[from] = newBalanceFrom;  // update the sender's balance
14     balance[to]   = newBalanceTo;    // update the receiver's balance
15 }
```

Smart contract example

$$\{\texttt{from} \mapsto x \ * \ \texttt{to} \mapsto y \ * \ M\} \ \texttt{transfer}(\texttt{from}, \texttt{to}, v) \ \{\texttt{from} \mapsto (x - v) \ * \ \texttt{to} \mapsto (y + v) \ * \ M\}$$

```
 1 mapping(address => uint) balance;
 2
 3 function transfer(address from, address to, uint amount) {
 4     uint newBalanceFrom, newBalanceTo;
 5
 6     if (balance[from] >= amount) {
 7         newBalanceFrom = balance[from] - amount; // compute new balance of sender
 8         newBalanceTo   = balance[to]   + amount; // compute new balance of receiver
 9     } else {
10         revert();
11     }
12
13     balance[from] = newBalanceFrom;  // update the sender's balance
14     balance[to]   = newBalanceTo;    // update the receiver's balance
15 }
```

Smart contract example

$$\{\texttt{from} \mapsto x \ * \ \texttt{to} \mapsto y \ * \ M\} \ \texttt{transfer}(\texttt{from}, \texttt{to}, v) \ \{\texttt{from} \mapsto (x - v) \ * \ \texttt{to} \mapsto (y + v) \ * \ M\}$$

```
 1 mapping(address => uint) balance;
 2
 3 function transfer(address from, address to, uint amount) {
 4     uint newBalanceFrom, newBalanceTo;
 5
 6     if (balance[from] >= amount) {
 7         newBalanceFrom = balance[from] - amount; // compute new balance of sender
 8         newBalanceTo   = balance[to]   + amount; // compute new balance of receiver
 9     } else {
10         revert();
11     }
12
13     balance[from] = newBalanceFrom;  // update the sender's balance
14     balance[to]   = newBalanceTo;    // update the receiver's balance
15 }
```

what if "from == to"?

Smart contract example

$$\{\texttt{from} \mapsto x \ * \ \texttt{to} \mapsto y \ * \ M\}\ \texttt{transfer}(\texttt{from}, \texttt{to}, v)\ \{\texttt{from} \mapsto (x - v) \ * \ \texttt{to} \mapsto (y + v) \ * \ M\}$$

```
 1 mapping(address => uint) balance;
 2
 3 function transfer(address from, address to, uint amount) {
 4     uint newBalanceFrom, newBalanceTo;
 5
 6     if (balance[from] >= amount) {
 7         newBalanceFrom = balance[from] - amount; // compute new balance of sender
 8         newBalanceTo   = balance[to]   + amount; // compute new balance of receiver
 9     } else {
10         revert();
11     }
12
13     balance[from] = newBalanceFrom;  // update the sender's balance
14     balance[to]   = newBalanceTo;    // update the receiver's balance
15 }
```

what if "from == to"?

$$\texttt{sum}(\texttt{balance}) = \texttt{sum}(\texttt{balance}')$$

How to obtain quality specifications?

- Best-effort approach, e.g.:
  - Identify functional correctness specs (for core business logic)
  - Generalize known security vulnerabilities and turn them into specs
  - Clarify reasonable assumptions on external environment
- Peer review
- Client communication

How to obtain quality specifications?

- Best-effort approach, e.g.:
  - Identify functional correctness specs (for core business logic)
  - Generalize known security vulnerabilities and turn them into specs
  - Clarify reasonable assumptions on external environment
- Peer review
- Client communication

Can we get clients involved in verification process?

From property-based testing to verification

```
mapping(address => uint) balance;

function transfer(address from, address to, uint amount) { ... }


// test total balance preservation: sum(old(balances)) == sum(balances)
function testTransfer(address from, address to, address other, uint amount) {
    // snapshot old balances
    uint oldBalFrom  = balance[from];
    uint oldBalTo    = balance[to];
    uint oldBalOther = balance[other];
    require(other != from && other != to); // ensure `other` represents the rest

    // run the target function (concretely or symbolically)
    transfer(from, to, amount);

    // check total balance preserved
    assert(Math.add(oldBalFrom, oldBalTo) == Math.add(balance[from], balance[to]));
    assert(oldBalOther == balance[other]);
}
```

From property-based testing to verification

```
1  mapping(address => uint) balance;
2
3  function transfer(address from, address to, uint amount) { ... }
4
5
6  // test total balance preservation: sum(old(balances)) == sum(balances)
7  function testTransfer(address from, address to, address other, uint amount) {
8      // snapshot old balances
9      uint oldBalFrom  = balance[from];
10     uint oldBalTo    = balance[to];
11     uint oldBalOther = balance[other];
12     require(other != from && other != to); // ensure `other` represents the rest
13
14     // run the target function (concretely or symbolically)
15     transfer(from, to, amount);
16
17     // check total balance preserved
18     assert(Math.add(oldBalFrom, oldBalTo) == Math.add(balance[from], balance[to]));
19     assert(oldBalOther == balance[other]);
20 }
```

From property-based testing to verification

```
1  mapping(address => uint) balance;
2
3  function transfer(address from, address to, uint amount) { ... }
4
5
6  // test total balance preservation: sum(old(balances)) == sum(balances)
7  function testTransfer(address from, address to, address other, uint amount) {
8      // snapshot old balances
9      uint oldBalFrom  = balance[from];
10     uint oldBalTo    = balance[to];
11     uint oldBalOther = balance[other];
12     require(other != from && other != to); // ensure `other` represents the rest
13
14     // run the target function (concretely or symbolically)
15     transfer(from, to, amount);
16
17     // check total balance preserved
18     assert(Math.add(oldBalFrom, oldBalTo) == Math.add(balance[from], balance[to]));
19     assert(oldBalOther == balance[other]);
20 }
```

From property-based testing to verification

```
1  mapping(address => uint) balance;
2
3  function transfer(address from, address to, uint amount) { ... }
4
5
6  // test total balance preservation: sum(old(balances)) == sum(balances)
7  function testTransfer(address from, address to, address other, uint amount) {
8      // snapshot old balances
9      uint oldBalFrom  = balance[from];
10     uint oldBalTo    = balance[to];
11     uint oldBalOther = balance[other];
12     require(other != from && other != to); // ensure `other` represents the rest
13
14     // run the target function (concretely or symbolically)
15     transfer(from, to, amount);
16
17     // check total balance preserved
18     assert(Math.add(oldBalFrom, oldBalTo) == Math.add(balance[from], balance[to]));
19     assert(oldBalOther == balance[other]);
20 }
```

From property-based testing to verification

```
1  mapping(address => uint) balance;
2
3  function transfer(address from, address to, uint amount) { ... }
4
5
6  // test total balance preservation: sum(old(balances)) == sum(balances)
7  function testTransfer(address from, address to, address other, uint amount) {
8      // snapshot old balances
9      uint oldBalFrom  = balance[from];
10     uint oldBalTo    = balance[to];
11     uint oldBalOther = balance[other];
12     require(other != from && other != to); // ensure `other` represents the rest
13
14     // run the target function (concretely or symbolically)
15     transfer(from, to, amount);
16
17     // check total balance preserved
18     assert(Math.add(oldBalFrom, oldBalTo) == Math.add(balance[from], balance[to]));
19     assert(oldBalOther == balance[other]);
20 }
```

{P} {C} {Q}

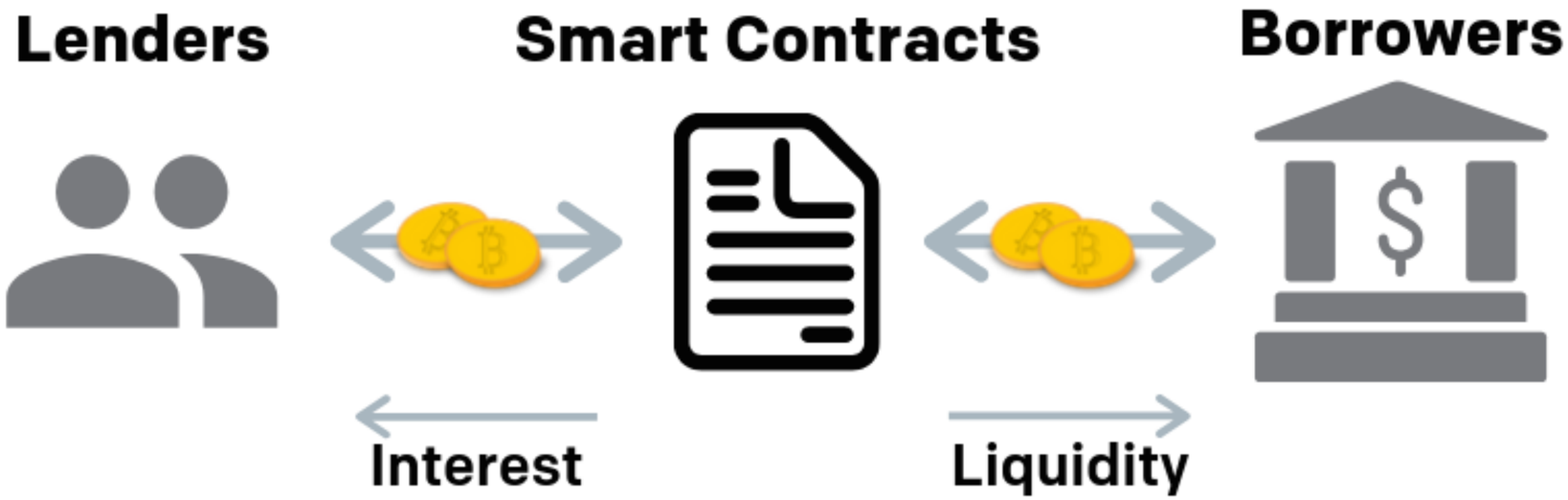
From property-based testing to verification

- This can be written and maintained by developers.
- This can be run on CI.
- This can be (easily translated to) specification.

```
1  mapping(address => uint) balanc
2
3  function transfer(address from,
4
5
6  // test total balance preservation: sum(old(balances)) == sum(balances)
7  function testTransfer(address from, address to, address other, uint amount) {
8      // snapshot old balances
9      uint oldBalFrom  = balance[from];
10     uint oldBalTo    = balance[to];
11 {p}
12     uint oldBalOther = balance[other];
13     require(other != from && other != to); // ensure `other` represents the rest
14 {c}
15     // run the target function (concretely or symbolically)
16     transfer(from, to, amount);
17 {o}
18     // check total balance preserved
19     assert(Math.add(oldBalFrom, oldBalTo) == Math.add(balance[from], balance[to]));
20     assert(oldBalOther == balance[other]);
   }
```

Refinement-based Verification

Example: simple lending protocol



* Image by https://medium.com/@goldfarbas/everything-you-need-to-know-about-defi-lending-protocols-6a5e8404f2b0

Example: simple lending protocol

Reference Model

```
mapping(address => num) principal;
mapping(address => num) interest;



function deposit(address user, num amount) {

    principal[user] += amount;
}

function reward(num amount) {
    num rate = amount / sum(principal);
    for (address user : keys(principal)) {
        interest[user] += principal[user] * rate;
    }
}
```

Example: simple lending protocol

Reference Model

```
mapping(address => num) principal;
mapping(address => num) interest;



function deposit(address user, num amount) {

    principal[user] += amount;
}

function reward(num amount) {
    num rate = amount / sum(principal);
    for (address user : keys(principal)) {
        interest[user] += principal[user] * rate;
    }
}
```

Optimized Code

```
mapping(address => num) principal;
mapping(address => num) interest;
mapping(address => num) lastRateSum;
num currRateSum;

function deposit(address user, num amount) {
    update(user);
    principal[user] += amount;
}

function reward(num amount) {
    num rate = amount / sum(principal);
    currRateSum += rate;
}

function update(address user) {
    num rate = currRateSum - lastRateSum[user];
    interest[user] += principal[user] * rate;
    lastRateSum[user] = currRateSum;
}
```

Example: simple lending protocol

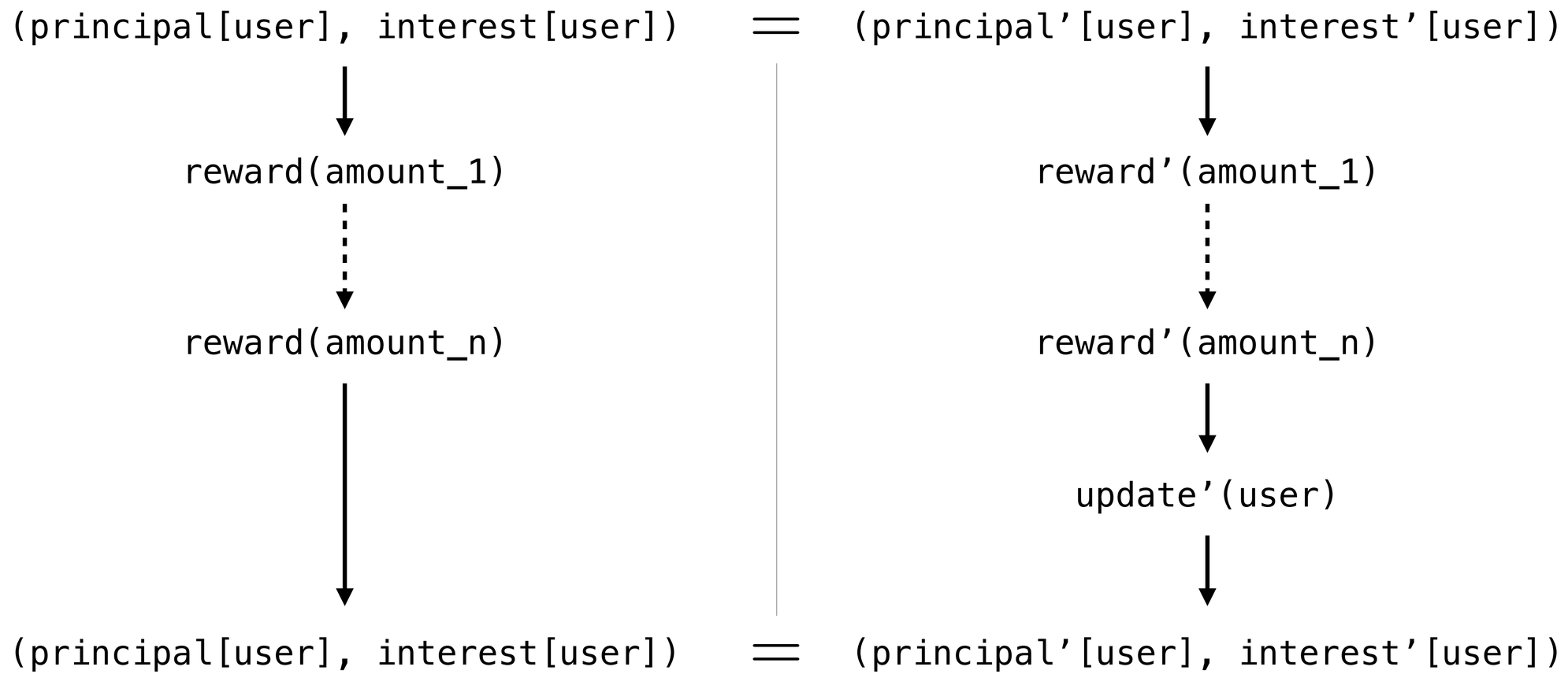
Reference Model

```
mapping(address => num) principal;
mapping(address => num) interest;



function deposit(address user, num amount) {

    principal[user] += amount;
}

function reward(num amount) {
    num rate = amount / sum(principal);
    for (address user : keys(principal)) {
        interest[user] += principal[user] * rate;
    }
}
```

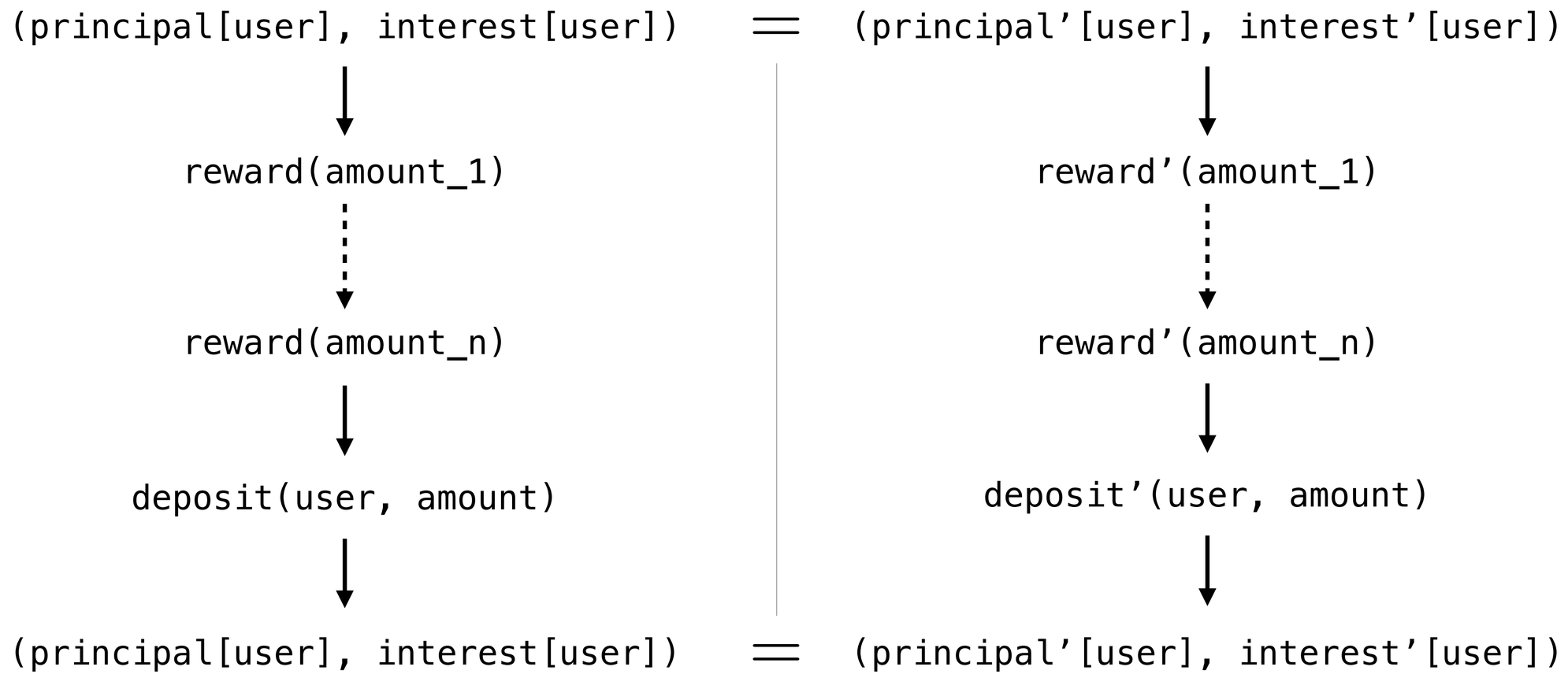
Optimized Code

```
mapping(address => num) principal;
mapping(address => num) interest;
mapping(address => num) lastRateSum;
num currRateSum;

function deposit(address user, num amount) {
    update(user);
    principal[user] += amount;
}

function reward(num amount) {
    num rate = amount / sum(principal);
    currRateSum += rate;
}

function update(address user) {
    num rate = currRateSum - lastRateSum[user];
    interest[user] += principal[user] * rate;
    lastRateSum[user] = currRateSum;
}
```

Refinement proof

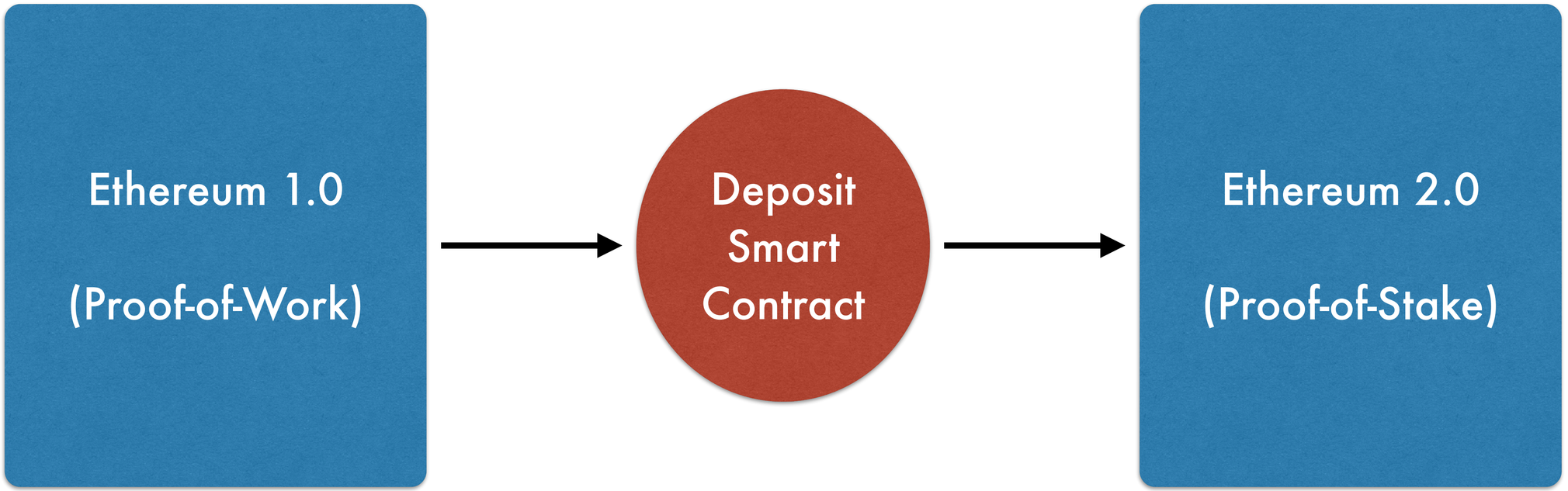


Refinement proof



End-to-End Formal Verification of Ethereum 2.0 Deposit Smart Contract

Ethereum 2.0 deposit contract



Top Accounts by ETH Balance

A total of > 1,999,999 accounts found (118,148,868.055 Ether)

(Showing the last 10,000 top accounts only)

First < Page 1 of 400 > Last

Rank	Address	Name Tag	Balance	Percentage
1	0x00000000219ab540356cbb839cbe05303d7705fa	Eth2 Deposit Contract	9,366,114.000069 Ether	7.92738361%
2	0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2	Wrapped Ether	7,711,580.31105652 Ether	6.52700313%
3	0xda9dfa130df4de4673b89022ee50ff26f6ea73cf	Kraken 13	2,113,030.001 Ether	1.78844710%
4	0x73bceb1cd57c711feac4224d062b0f6ff338501e		2,003,504.57090949 Ether	1.69574589%
5	0xbe0eb53f46cd790cd13851d5eff43d12404d33e8	Binance 7	1,996,008.34195866 Ether	1.68940116%
6	0x9bf4001d307dfd62b26a2f1307ee0c0307632d59		1,490,000.0180927 Ether	1.26112086%
7	0x4ddc2d193948926d02f9b1fe9e1daa0718270ed5	Compound: cETH Token	976,993.92778433 Ether	0.82691772%
8	0x61edcdf5bb737adffe5043706e7c5bb1f1a56eea	Gemini 3	929,498.95358134 Ether	0.78671846%
9	0xdc24316b9ae028f1497c275eb9192a3ea0f67022	Lido: Curve Liquidity Farming Pool Contract	603,436.15441902 Ether	0.51074222%

Etherscan

Eth: $3,227.56 (+3.97%) | 82 Gwei

Contract Overview

Eth2 Deposit Contract

Balance: 9,366,114.000069000000000069 Ether

Value: $30,220,047,804.64 (@ $3,226.53/ETH)

Token: >$592,202.12 >107

Top Accounts by ETH Balance

A total of > 1,999,999 accounts found (118,1

(Showing the last 10,000 top accounts only)

Rank	Address			
1	0x00000000219ab540356cbb839cbe05303d7705fa	Eth2 Deposit Contract	9,366,114.000069 Ether	7.92738361%
2	0xc02aaa39b223fe8d0a0e5c4f27ead9083c756cc2	Wrapped Ether	7,711,580.31105652 Ether	6.52700313%
3	0xda9dfa130df4de4673b89022ee50ff26f6ea73cf	Kraken 13	2,113,030.001 Ether	1.78844710%
4	0x73bceb1cd57c711feac4224d062b0f6ff338501e		2,003,504.57090949 Ether	1.69574589%
5	0xbe0eb53f46cd790cd13851d5eff43d12404d33e8	Binance 7	1,996,008.34195866 Ether	1.68940116%
6	0x9bf4001d307dfd62b26a2f1307ee0c0307632d59		1,490,000.0180927 Ether	1.26112086%
7	0x4ddc2d193948926d02f9b1fe9e1daa0718270ed5	Compound: cETH Token	976,993.92778433 Ether	0.82691772%
8	0x61edcdf5bb737adffe5043706e7c5bb1f1a56eea	Gemini 3	929,498.95358134 Ether	0.78671846%
9	0xdc24316b9ae028f1497c275eb9192a3ea0f67022	Lido: Curve Liquidity Farming Pool Contract	603,436.15441902 Ether	0.51074222%

Ethereum 2.0 deposit contract

deposit ledger

Name	Amount
Alice	10 ETH
Bob	20 ETH
-	-
-	-

Ethereum 2.0 deposit contract

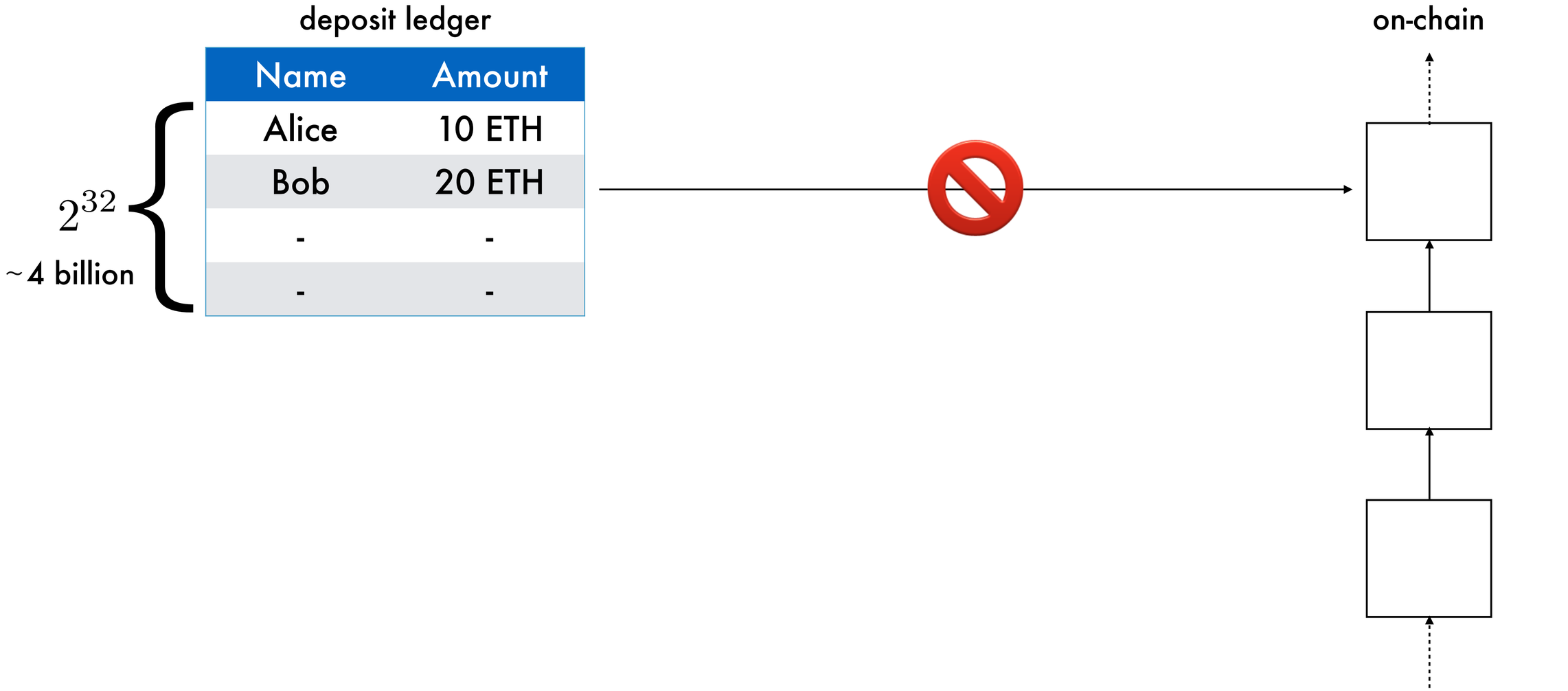
deposit ledger

2^{32}

~4 billion

Name	Amount
Alice	10 ETH
Bob	20 ETH
-	-
-	-

Ethereum 2.0 deposit contract



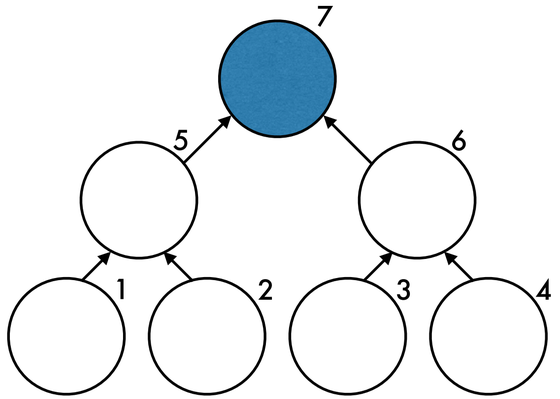
Ethereum 2.0 deposit contract

deposit ledger

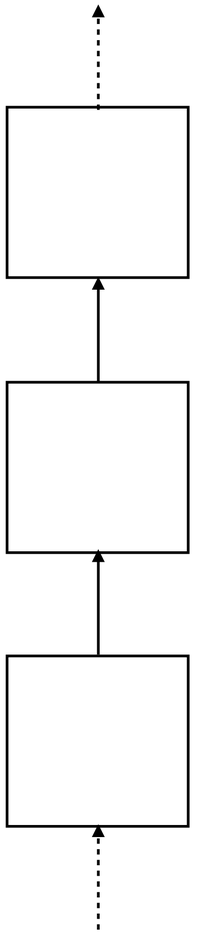
2^{32} ~4 billion

Name	Amount
Alice	10 ETH
Bob	20 ETH
-	-
-	-

Merkle tree



on-chain



Ethereum 2.0 deposit contract

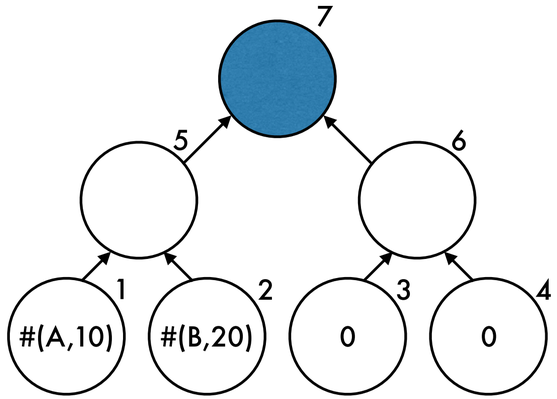
deposit ledger

2^{32}

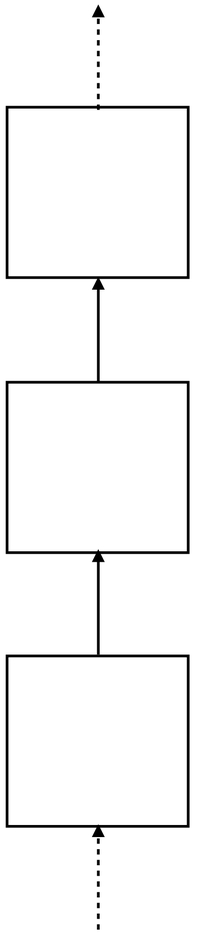
~4 billion

Name	Amount
Alice	10 ETH
Bob	20 ETH
-	-
-	-

Merkle tree



on-chain



Ethereum 2.0 deposit contract

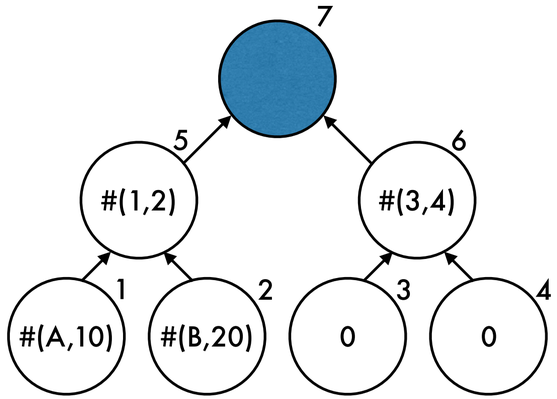
deposit ledger

2^{32}

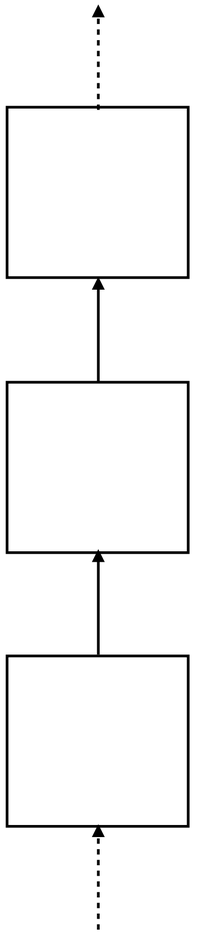
~4 billion

Name	Amount
Alice	10 ETH
Bob	20 ETH
-	-
-	-

Merkle tree



on-chain



Ethereum 2.0 deposit contract

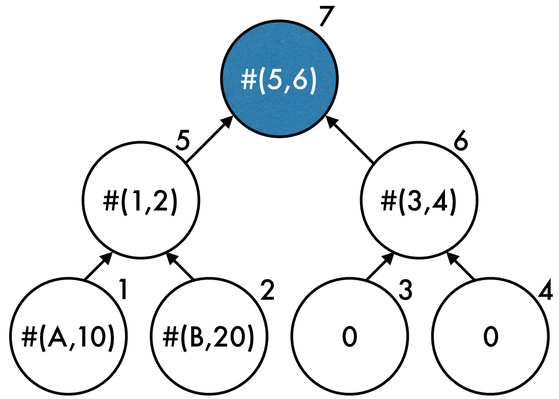
deposit ledger

2^{32}

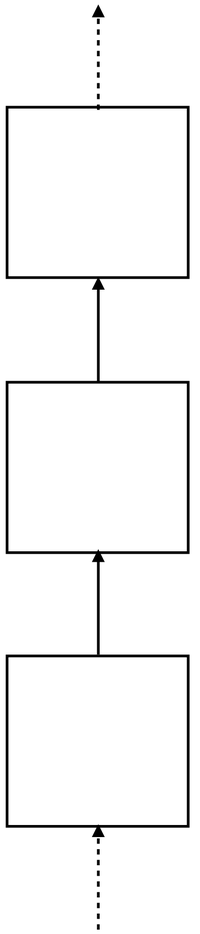
~4 billion

Name	Amount
Alice	10 ETH
Bob	20 ETH
-	-
-	-

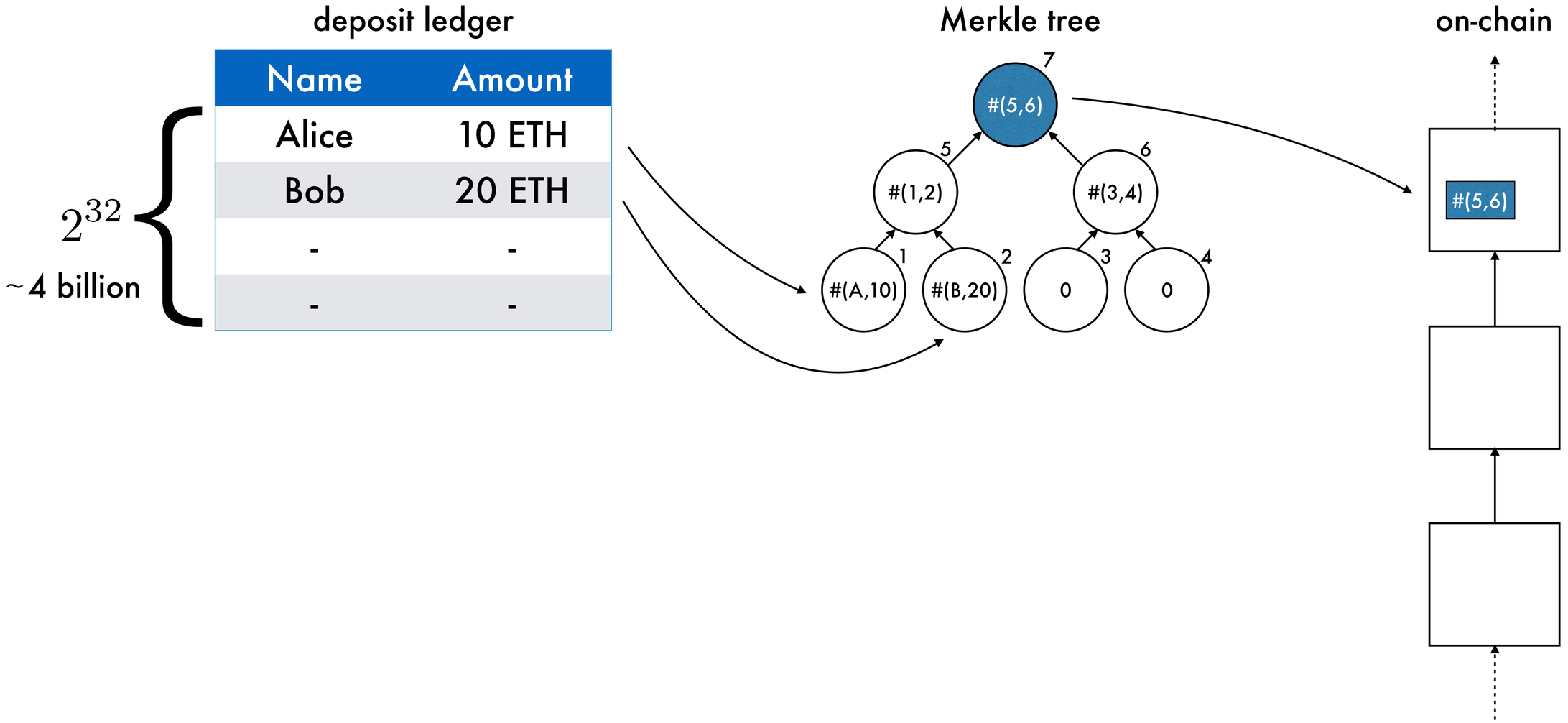
Merkle tree



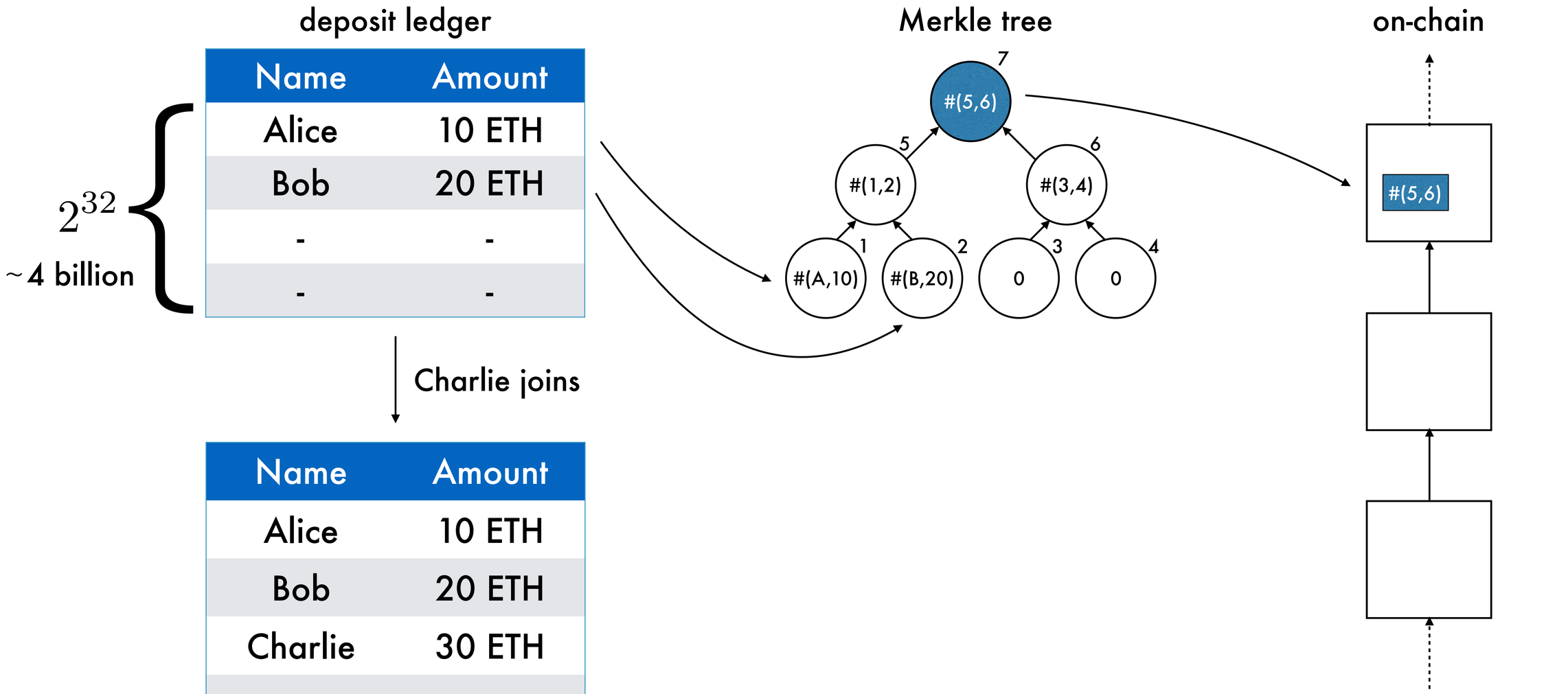
on-chain



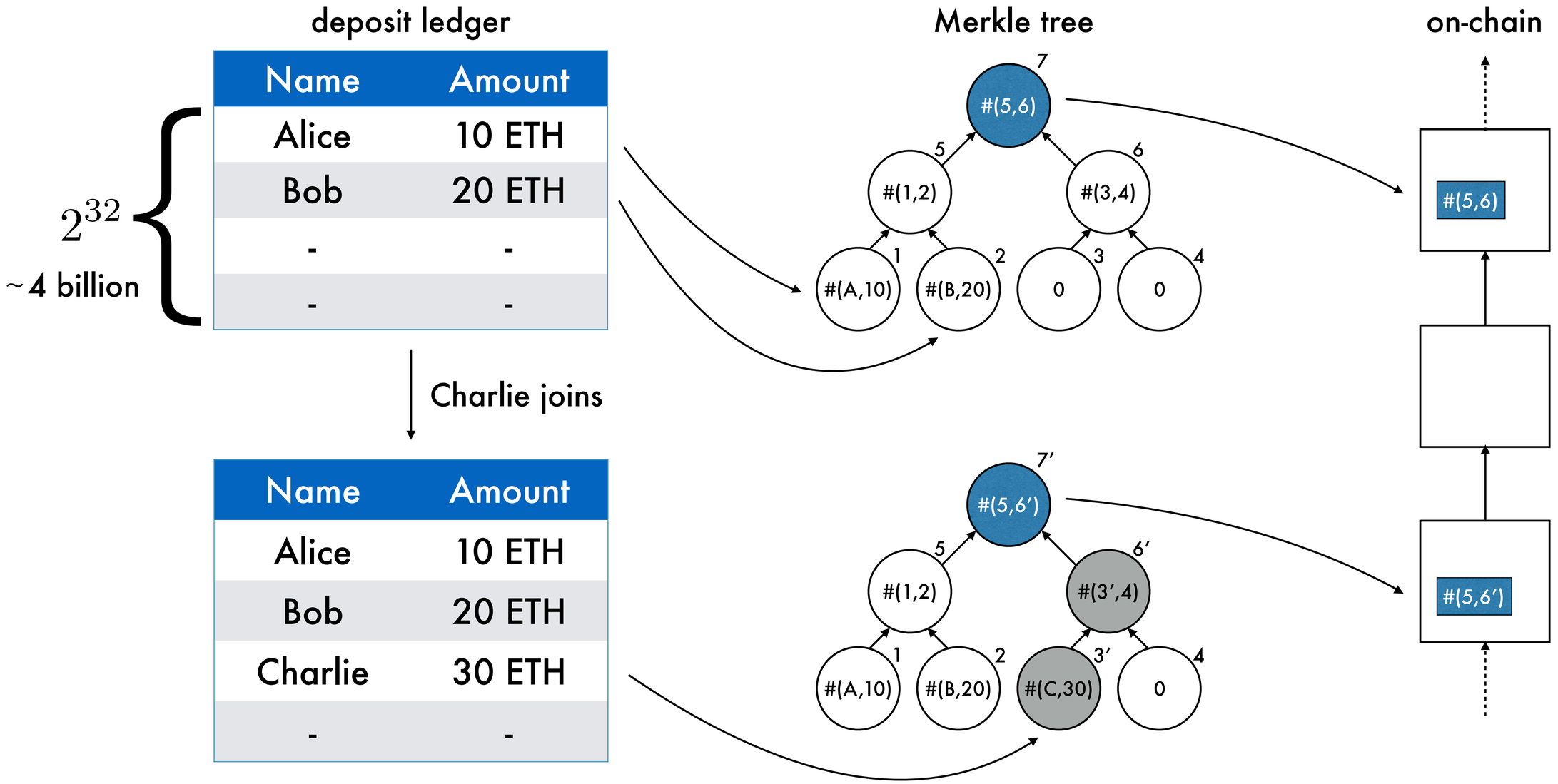
Ethereum 2.0 deposit contract



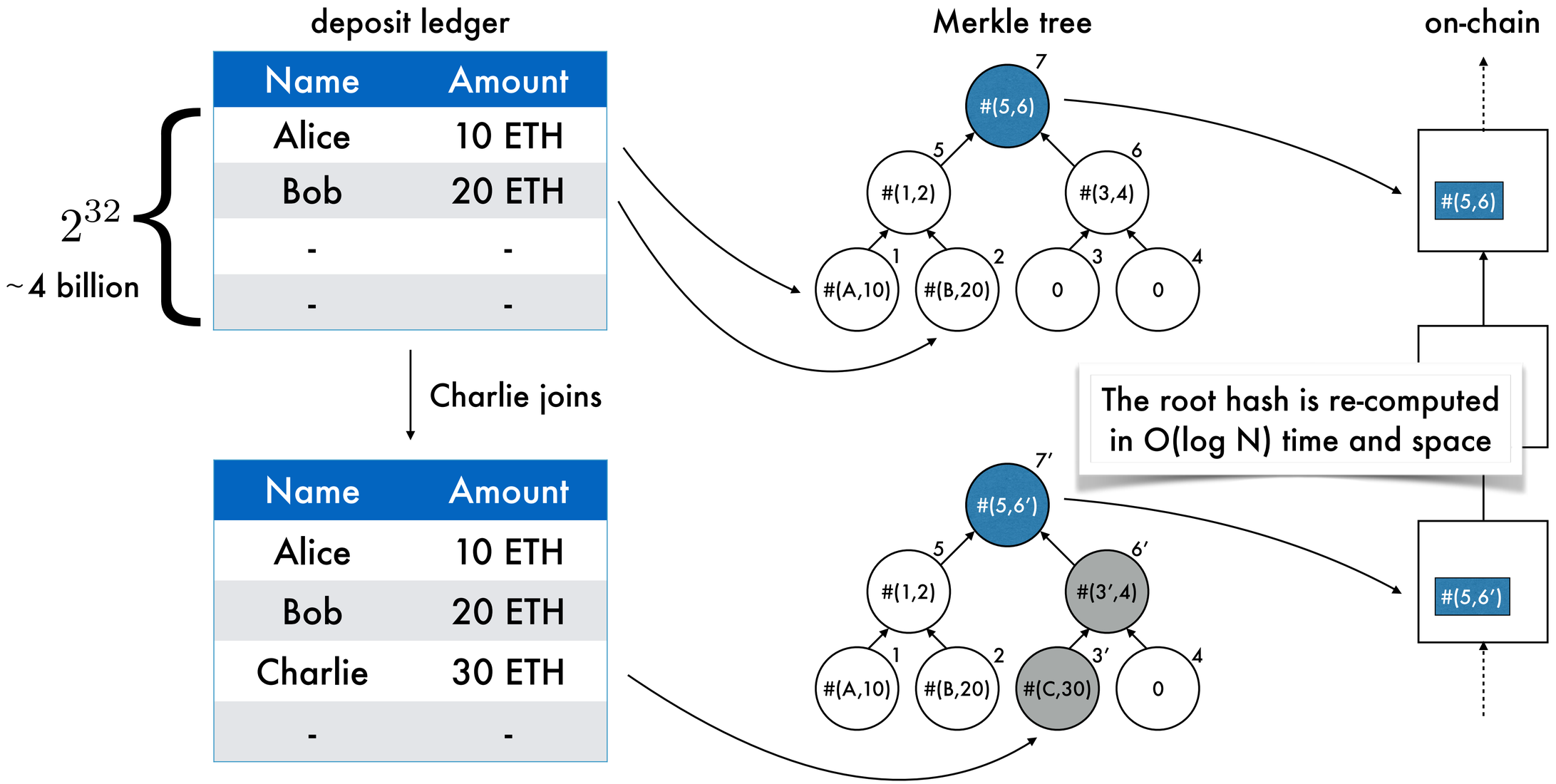
Ethereum 2.0 deposit contract



Ethereum 2.0 deposit contract



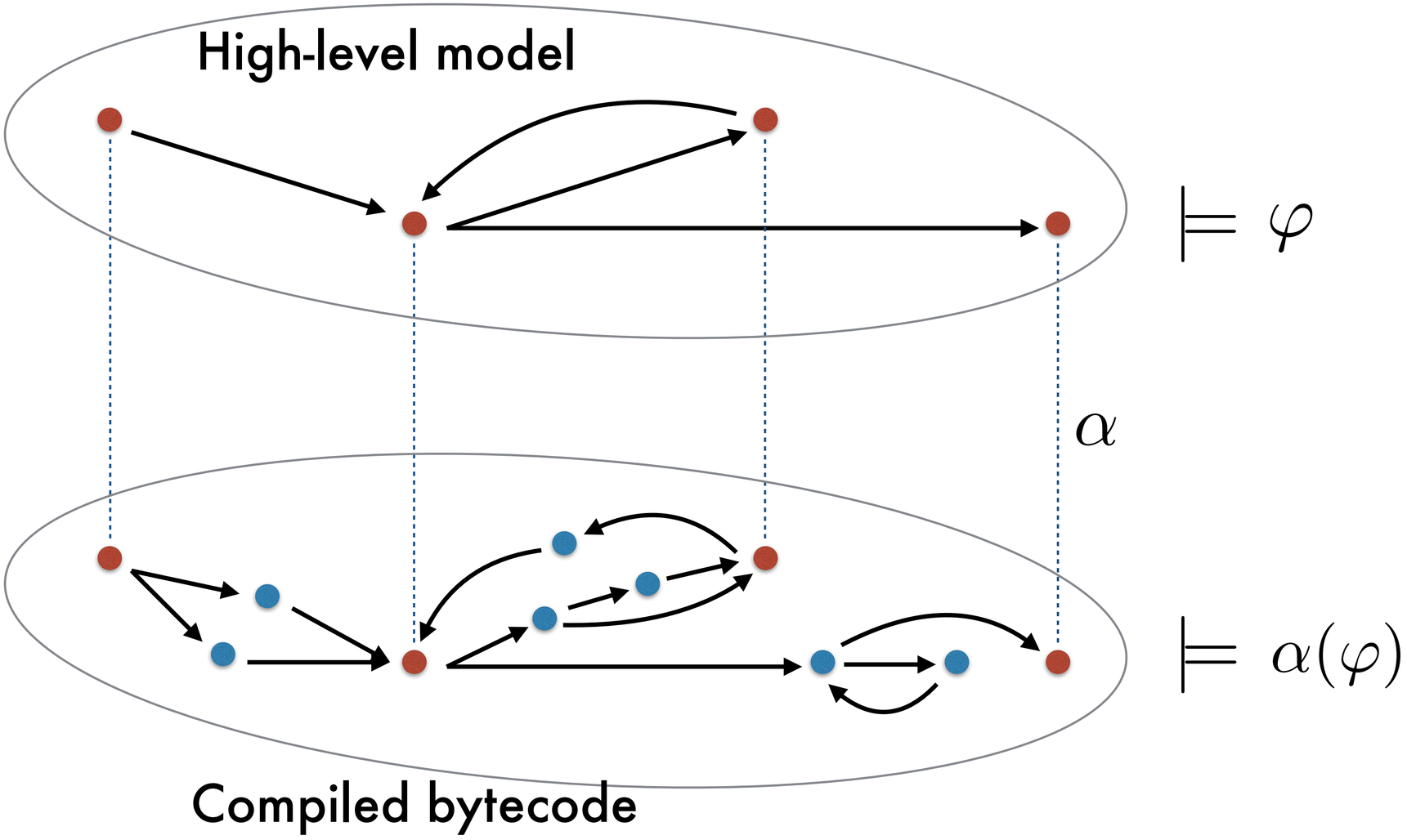
Ethereum 2.0 deposit contract



Formal verification of deposit contract

- *End-to-end* verification via *refinement-based* approach
  - Formalize and verify the incremental Merkle tree algorithm
  - Verify the contract *bytecode* faithfully implements the algorithm
    - No need to trust compiler
- This separation of concerns helped to reduce verification effort

Refinement-based verification



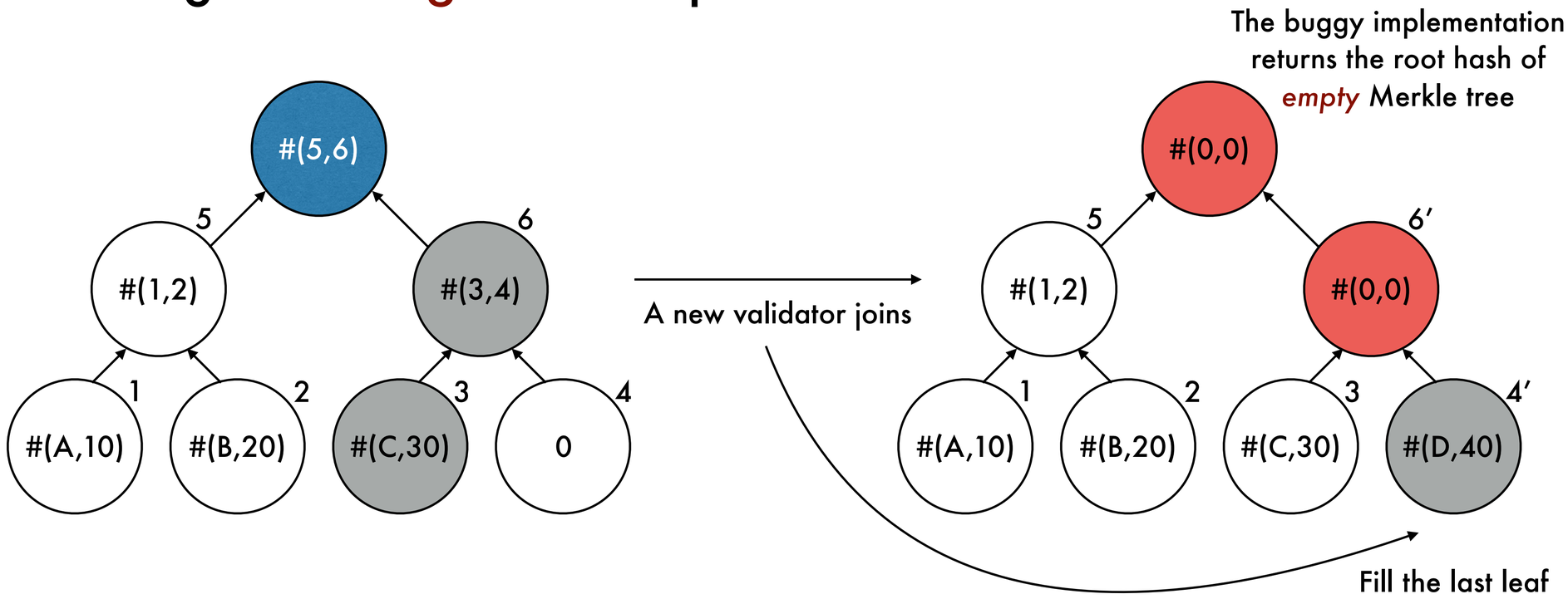
Verification effort

Correctness proof (over formal model)	2 person-weeks
Refinement proof (over bytecode)	5 person-weeks
Total	7 person-weeks

Source code	~100 LOCs
Bytecode	~3,000 instructions
Mechanized proofs	~1,000 LOCs

Findings

- Critical bug in the *algorithm* implementation



- Several bugs in the compiled *bytecode*
  - Mostly due to *compiler bugs*

Thank you!