IC3

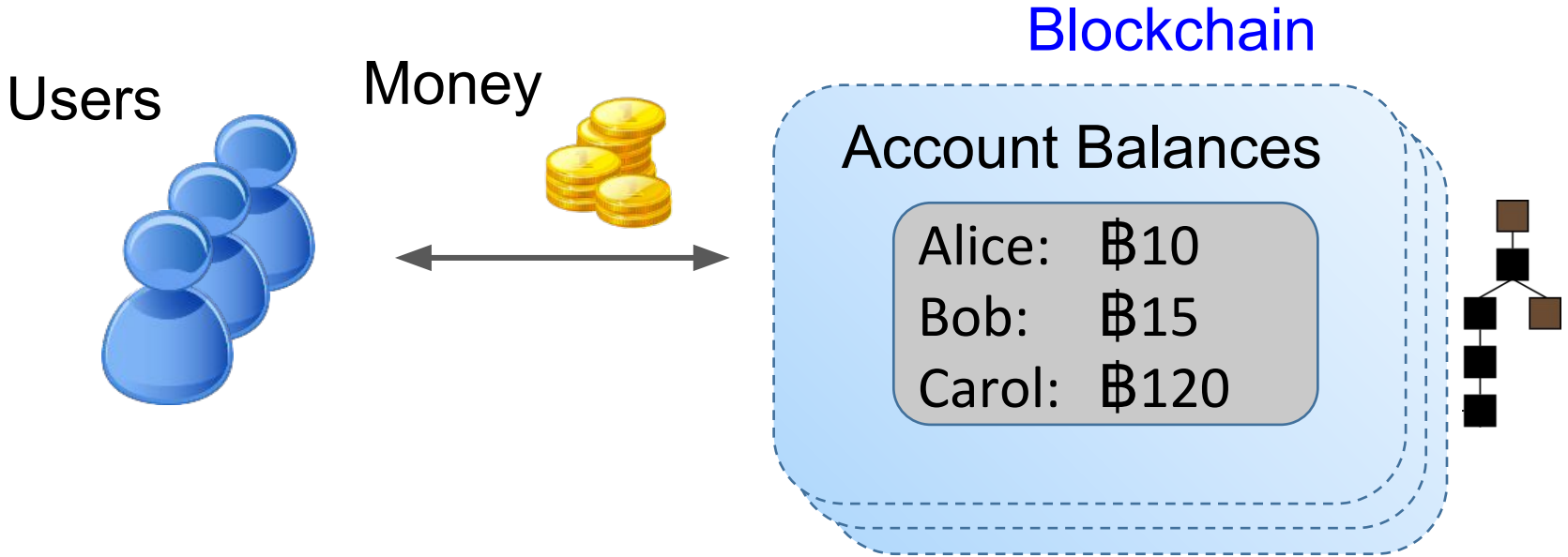
Ethereum isn't Turing Complete and it doesn't matter anyway

Andrew Miller

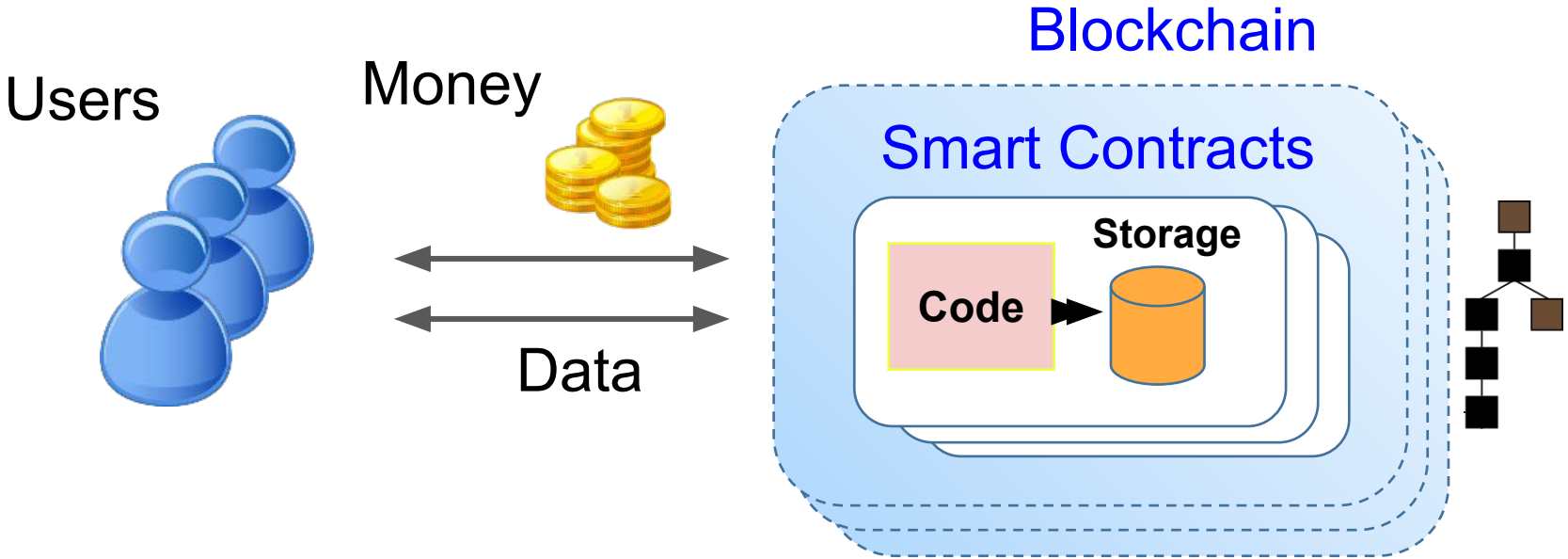
Aug 2, 2016

IC3 NYC Blockchain Meetup

Digital currency is just one application running a top a blockchain



Blockchain: a virtual trusted third party
Smart contracts: programs running on a blockchain



- IC3-Ethereum Crypto Boot Camp and Workshop - July 20-28, 2016

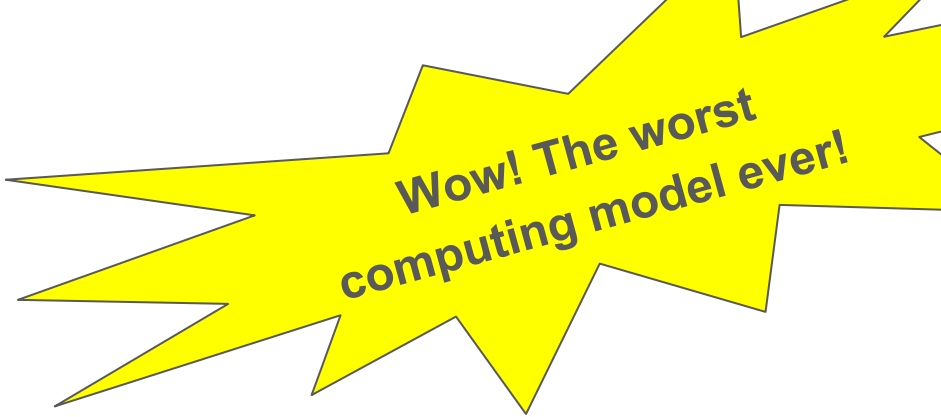
"One of the most productive hackathons I have ever attended!"

- Vitalik Buterin, Ethereum Foundation

Why Turing-complete smart contracts are doomed

(1) Turing-complete languages are fundamentally inappropriate for writing "smart contracts" - because such languages are inherently *undecidable*[1] , which makes it impossible to know what a "smart contract" will do *before* running it.

Turing Machines



“Hello World” in (high level) Turing Machine code

```
--<-<<+[+[<+>--->->->-<<<]>]<<--.<++++
++.<<-..<<.<+.>>.>>.<<<.+++.>>.>>-.<<<
+.
```

Turing Machines are universal*

* … for computable *functions*

Church-Turing thesis:

"All reasonable computing machines can be simulated* by each other"

*With ***polynomial asymptotic*** efficiency

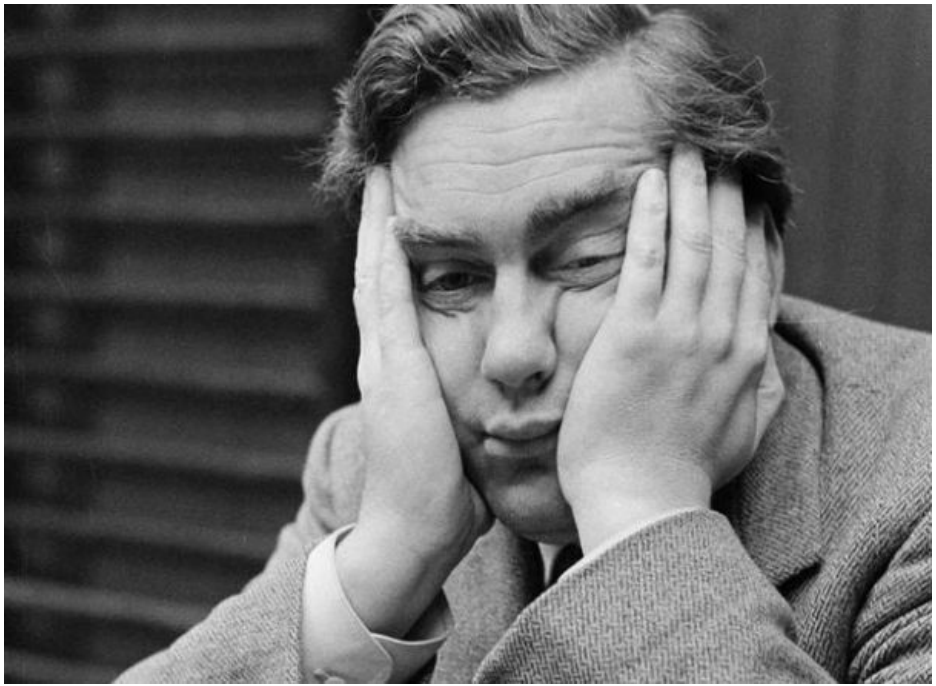
In computability theory, the **halting problem** is the problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running or continue to run forever.

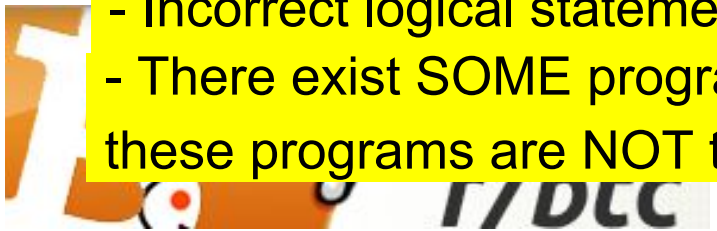
Why Turing-complete smart contracts are doomed

(1) Turing-complete languages are fundamentally inappropriate for writing "smart contracts" - because such languages are inherently *undecidable*[1] , which makes it impossible to know what a "smart contract" will do *before* running it.

Ethereum isn’t Turing complete

Actually, the Ethereum Virtual Machine always halts after a bounded time.





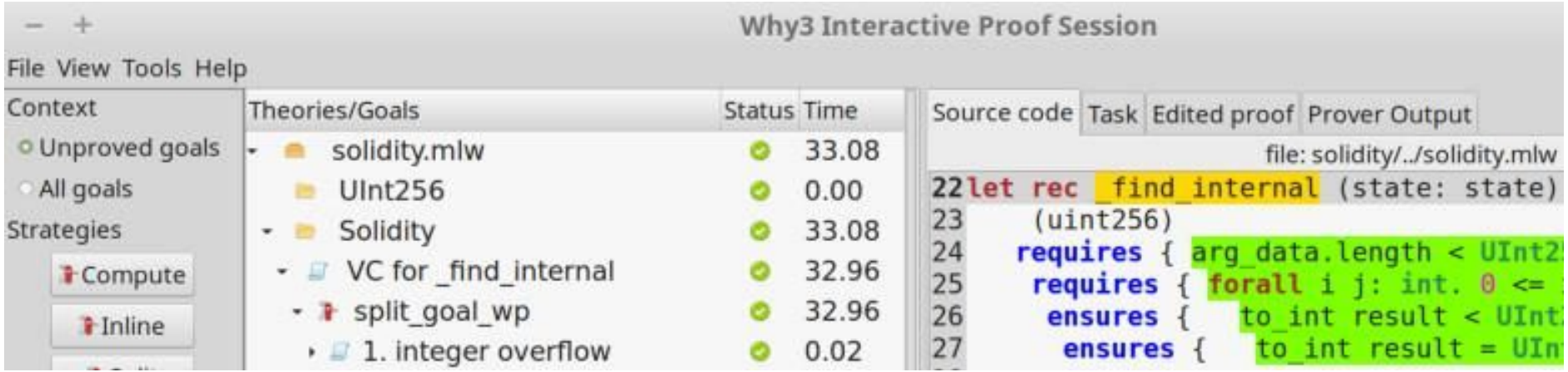
- Incorrect logical statement. No points awarded.
- There exist SOME programs we can't decide, but in practice these programs are NOT the programs we care about.

Why Turing-complete smart contracts are doomed

(1) Turing-complete languages are fundamentally inappropriate for writing "smart contracts" - because such languages are inherently *undecidable*[1] , which makes it impossible to know what a "smart contract" will do *before* running it.

Halting problem is not an excuse to avoid formal analysis

Many formal analysis tools apply to Turing complete languages



No efficiently* solvable problem requires Turing completeness

*polynomial time

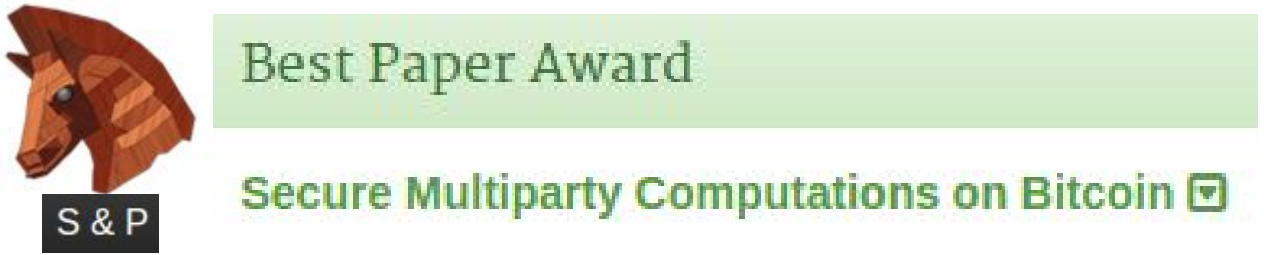
Turing Machines are NOT Universal for Smart Contracts

Neithereum - a Turing complete and yet utterly useless smart contract system

0xf0	CREATE	3	1	Create a new account with associated code.
0x01	ADD	2	1	Addition operation. $\boldsymbol{\mu}'_{\mathbf{s}}[0] \equiv \boldsymbol{\mu}_{\mathbf{s}}[0] + \boldsymbol{\mu}_{\mathbf{s}}[1]$
0x02	MUL	2	1	Multiplication operation.
0xf1	CALL	7	1	Message-call into an account.
0x50	POP	1	0	Remove item from stack.
0x51	MLOAD	1	1	Load word from memory.
0x56	JUMP	1	0	Alter the program counter. $J_{\text{JUMP}}(\boldsymbol{\mu}) \equiv \boldsymbol{\mu}_{\mathbf{s}}[0]$ This has the effect of writing said value to $\boldsymbol{\mu}_{pc}$.
0x57	JUMPI	2	0	Conditionally alter the program counter.
0x5a	GAS	0	1	Get the amount of available gas, includi for the cost of this instruction

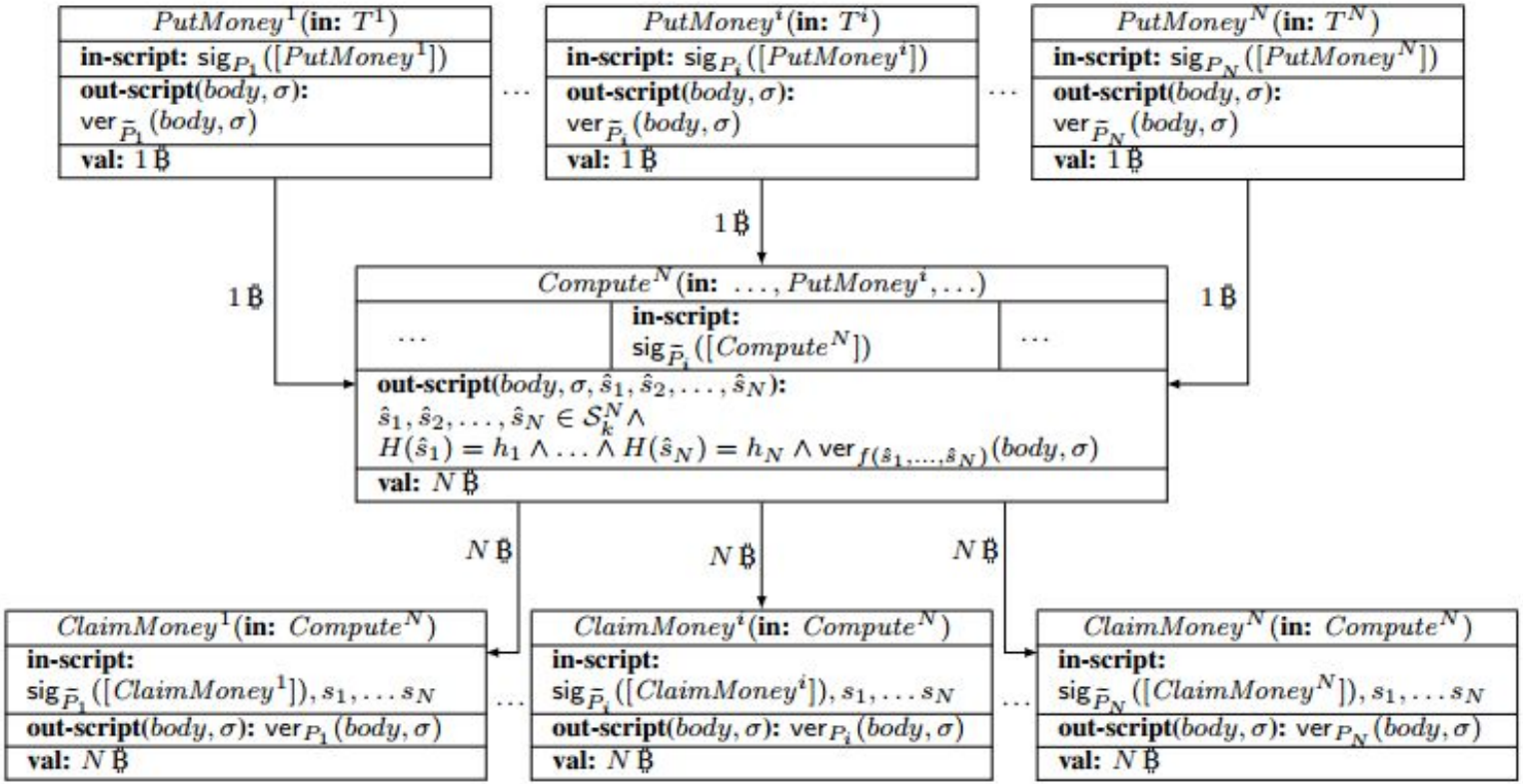
Ethereum is more powerful than Bitcoin for reasons that have nothing to do with Turing completeness

Due to the limitations of Bitcoin's UTXO structure, even celebrated Bitcoin smart contracts are inefficient.

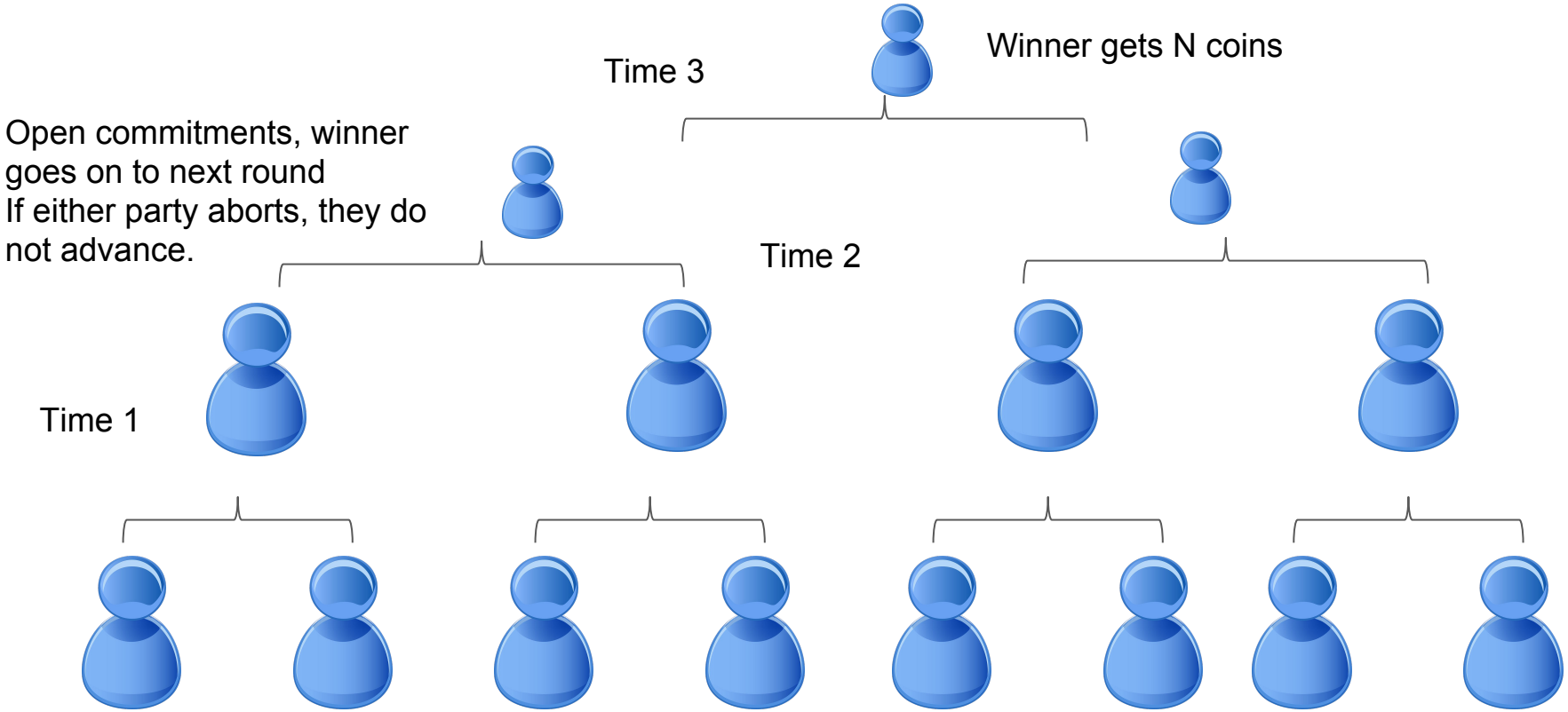


A fair lottery game with $O(N^3)$ collateral required

Each of N parties deposits N^2 coins (N^3 in total) and commits to a random string
If any party does not open their commitment in time, they forfeit their deposit, others compensated



Cheaper lottery on Ethereum using persistent state



Ethereum isn't Universal - Contrived example

```
contract WantToPlayAGame {
    int private switch = 1;
    int private balance = 0;
    function doubleMyMoney() onlyAlice {
        assert msg.value == 1000000;
        assert block.number < B;
        if (switch) balance = 1000000 * 2;
    }
    function withdraw() onlyAlice {
        assert block.number >= B;
        send(msg.sender, balance);
    }
    function tricky() onlyJigsaw { switch = 0; }
}
```

Only safe to call **doubleMyMoney** if "**switch**" is set.

There is no way to inspect **switch**!

- Cannot observe effects this block
- Cannot check if 1st tx in block
- Cannot load current block hash

Moral of the story

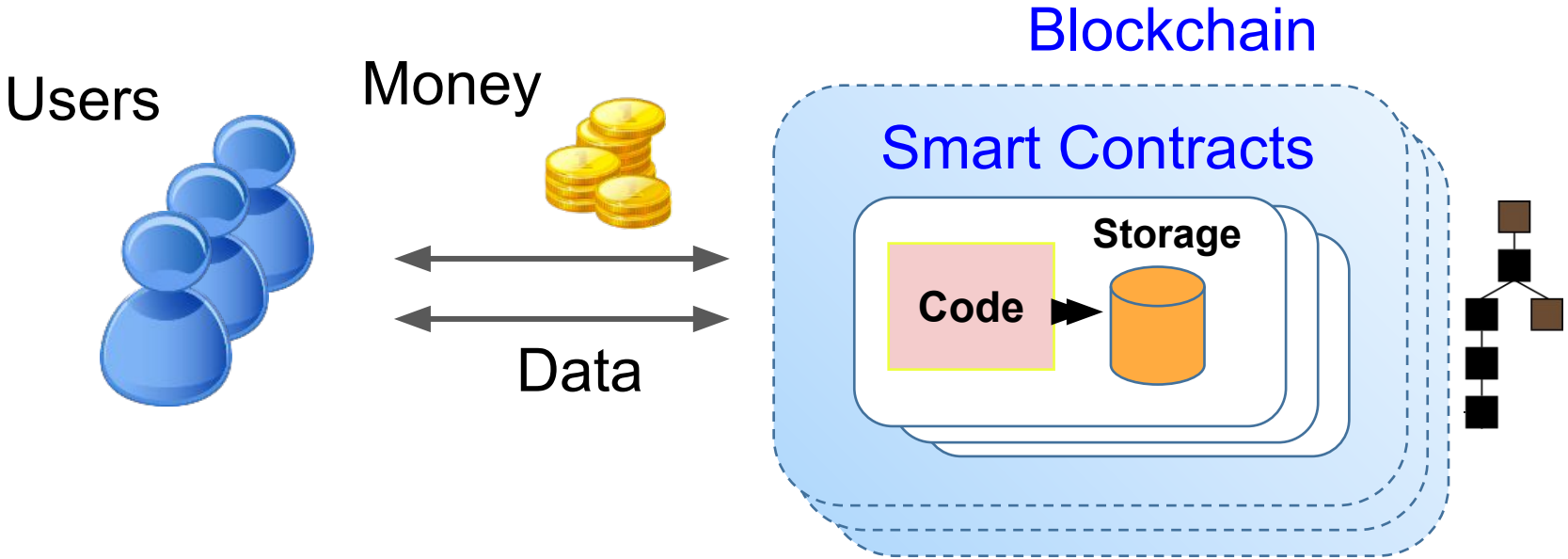
Difficult tradeoffs between ***expressiveness*** and ***transparency*** in programming language design

HAWK:
Privacy-Preserving Smart Contracts

Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen,
Charalampos Papamanthou

IEEE S&P, 2016 https://eprint.iacr.org/2015/675

Blockchain: a virtual TTP - *correctness, not privacy*
Smart contracts: programs running on a blockchain



July 16, 2016

~$1B Market Cap

TOTAL SUPPLY OF 82,028,272.37 ETHER

$11.92 @ 0.01789 BTC/ETH

LAST BLOCK

1893477 (14.08s Avg)

Hash Rate

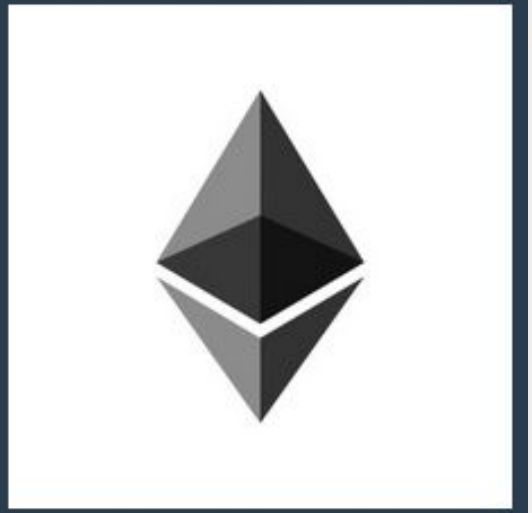
4,122.04 GH/s

TRANSACTIONS

7222997

Contracts

98326



DAPPS FOR BEGINNERS

Ethereum contract tutorials.

```
contract decentralisedAuction{
        struct auction {
                uint deadline;
                uint highestBid;
                address highestBidder;
                address recipient;
        }
        mapping(uint => auction) Auctions;
        uint numAuctions;

        function startAuction(uint timeLimit) returns
                auctionID = numAuctions++;
                Auctions[auctionID].deadline = block
                Auctions[auctionID].recipient = msg.
        }
        function bid(uint id) returns (address highe
                auction a = Auctions[id];
                if (a.highestBid + 1*10^18 > msg.val
```

Big concern with smart contracts is **privacy**

- **Code** of the contract is public
- **Data** sent to the contract is public
- **Money** sent/received is public

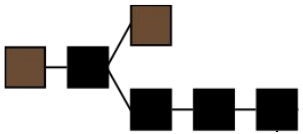
Implications of privacy leaks

- **What stocks** you are invested in
- **What risks** you are hedging against
- An attacker can **“front-run”** your order

Naïve auction

Blockchain Contract

Winner pays 2nd price
Other bidders get refund
Seller receives 2nd price



1 Input

bids

leaked

HAZARD

2 Output

refunds

2nd price

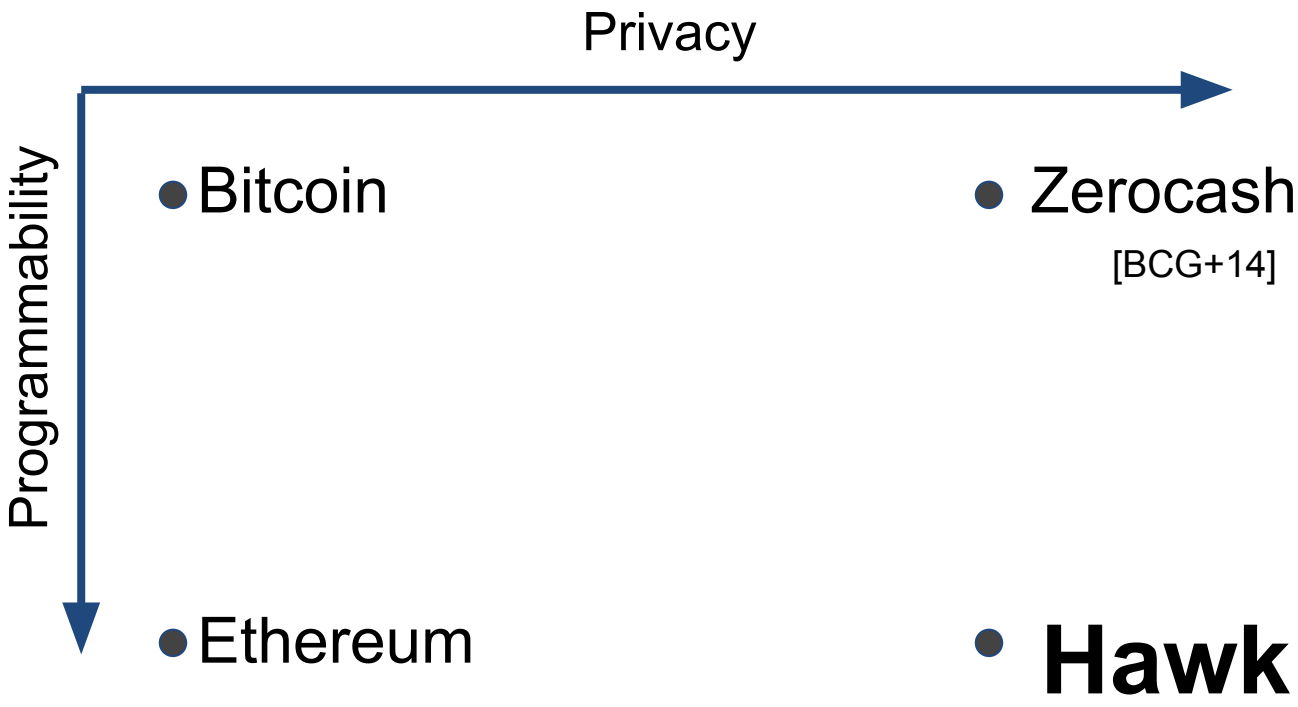
Hawk:

a framework for **privacy-preserving smart contracts**

Main idea:

Use **Zero-Knowledge Proofs (ZKP)**

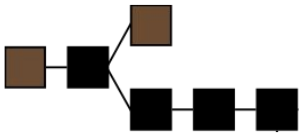
Prove statements about a ciphertext without revealing information about the plaintext


Privacy
Programmability
Bitcoin
Zerocash
[BCG+14]
Ethereum
Hawk

Hawk auction

Blockchain Contract

ZKP Parameters



1 Freeze

Commitment( , r), ZKP

2 Open

$\text{Encrypt}_{\text{Manager}}$( , r)

3 Finalize

$\text{Encrypt}_{\text{User}}$(), ZKP

Minimally trusted manager (i.e., auctioneer)

- Not for correctness
- Not for input independence
- Not for the security of the currency
- Not for fairness
- Trusted only for posterior privacy in *this* contract

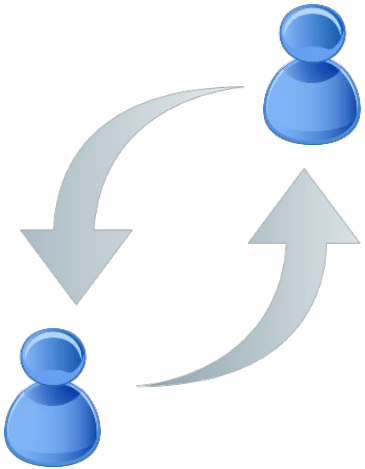
Could implement with MPC, hardware

Develop applications without thinking about crypto

Smart contract program

Hawk Compiler

Cryptographic protocol



```
private contract auction(Inp &in, Outp &out) {
  int winner = -1;
  int bestprice = -1;
  int secondprice = -1;

  for (int i = 0; i < N; i++) {
    if (in.party[i].$val > bestprice) {
      secondprice = bestprice;
      bestprice = in.party[i].$val;
      winner = i;
    } else if (in.party[i].$val > secondprice) {
      secondprice = in.party[i].$val;
    }
  }

  // Winner pays secondprice to seller
  // Everyone else is refunded
  out.Seller.$val = secondprice;
  out.party[winner].$val = bestprice - secondprice;
  out.winner = winner;
  for (int i = 0; i < N; i++) {
    if (i != winner)
      out.party[i].$val = in.party[i].$val;
  }
}
```

Sealed-bid auction

Rock-paper-scissors

```
    if (in.Bob.$val != $1)    out.out = B_CHEAT;
    Outcome o = outcome(in.Alice.move, in.Bob.move);
    if   (o == WIN)  out.Alice.$val = $2;
    else if (o == LOSE)  out.Bob.$val  = $2;
    else out.Alice.$val = out.Bob.$val = $1;
  }

  public contract deposit() {
    // Alice and Bob each deposit $2
    // Manager deposits $4
    def check(Output o):
      send $4 to Manager
      if (o == A_CHEAT): send $4 to Bob
      if (o == B_CHEAT): send $4 to Alice
      if (o == OK):
        send $2 to Alice
        send $2 to Bob
    def managerTimedOut():
      send $4 to Bob
      send $4 to Alice
  }
```

```
1  // Raise $10,000 from up to N donors
2  #define BUDGET $10000

3  HawkDeclareParties(Entrepreneur, /* N Parties */);
4  HawkDeclareTimeouts(/* hardcoded timeouts */);

5  private contract crowdfund(Inp &in, Outp &out) {
6    int sum = 0;
7    for (int i = 0; i < N; i++) {
8      sum += in.p[i].$val;
9    }
10   if (sum >= BUDGET) {
11     // Campaign successful
12     out.Entrepreneur.$val = sum;
13   } else {
14     // Campaign unsuccesful
15     for (int i = 0; i < N; i++) {
16       out.p[i].$val = in.p[i].$val; // refund
17     }
18   }
19 }
```

Kick-starter

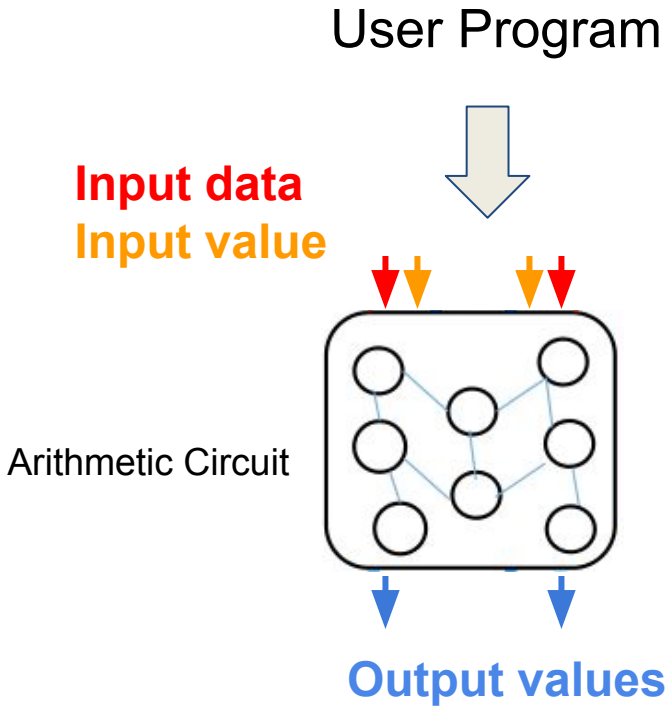
Financial Swap

```
6   private contract swap(Inp &in, Outp &out) {
7     if (sha1(in.Alice.threshold) != threshold_comm)
        out.$ = A_CHEAT;
7     if (in.Alice.$val != $10) out.$ = A_CHEAT;
8     if (in.Bob.$val != $10) out.$ = B_CHEAT;
8
9     if (in.stockprice < in.Alice.threshold[0])
        out.Alice.$val = $20;
10    else out.Bob.$val = $20;
11  }

12  public contract deposit {
13    def receiveStockPrice(stockprice):
14      // Alice and Bob each deposits $10
15      // Assume the stock price authority is trusted
16      // to send this contract the price
17      assert msg.sender == StockPriceAuthority
18      self.stockprice = stockprice
19    def check(int stockprice, Output o):
20      assert stockprice == self.stockprice
21      if (o == A_CHEAT): send $20 to Bob
22      if (o == B_CHEAT): send $20 to Alice
23      if (o == OK):
```

Hawk compiler

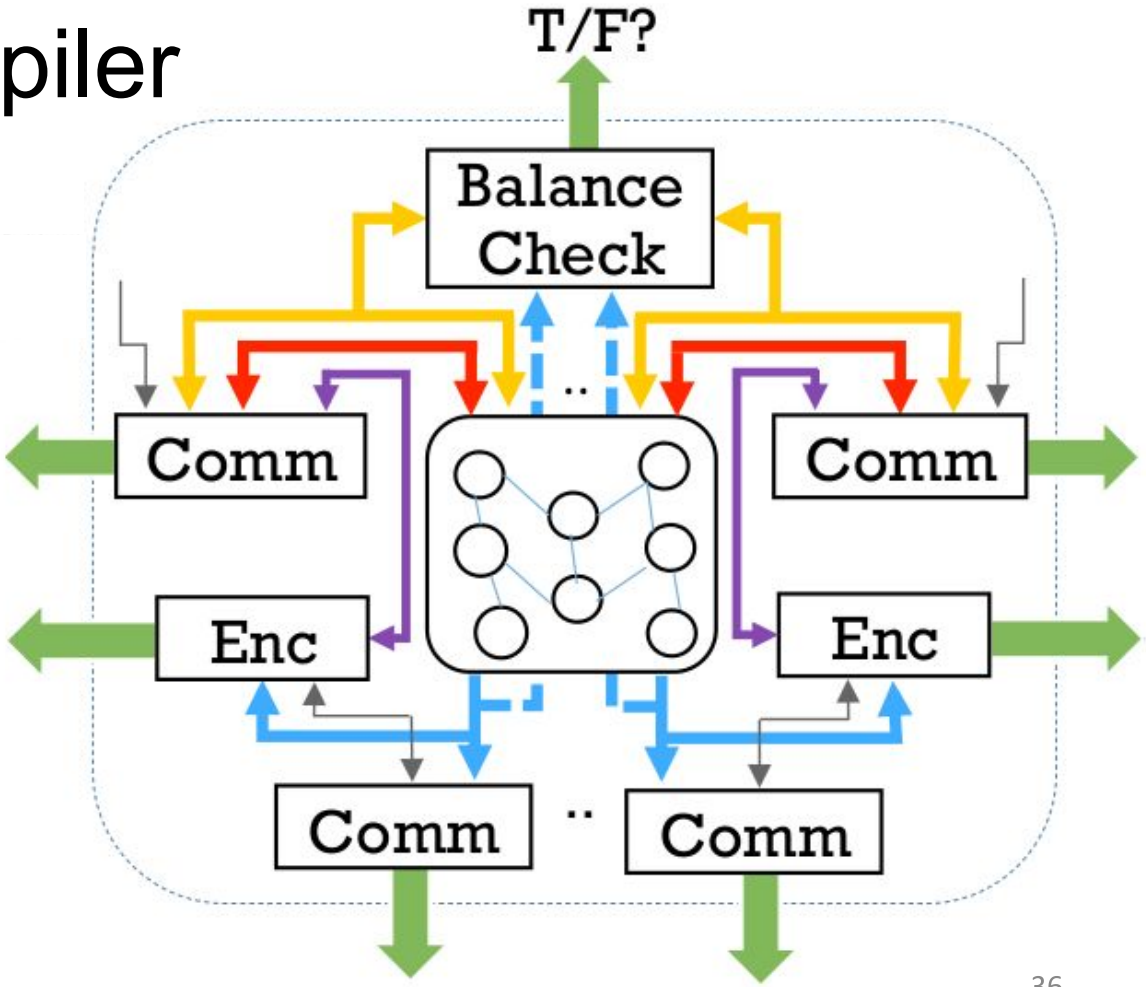
Step 1: User provided program compiled to arithmetic circuit



Hawk compiler

Step 1: User provided program compiled to arithmetic circuit

Step 2: Augment arithmetic with ZKP-friendly crypto



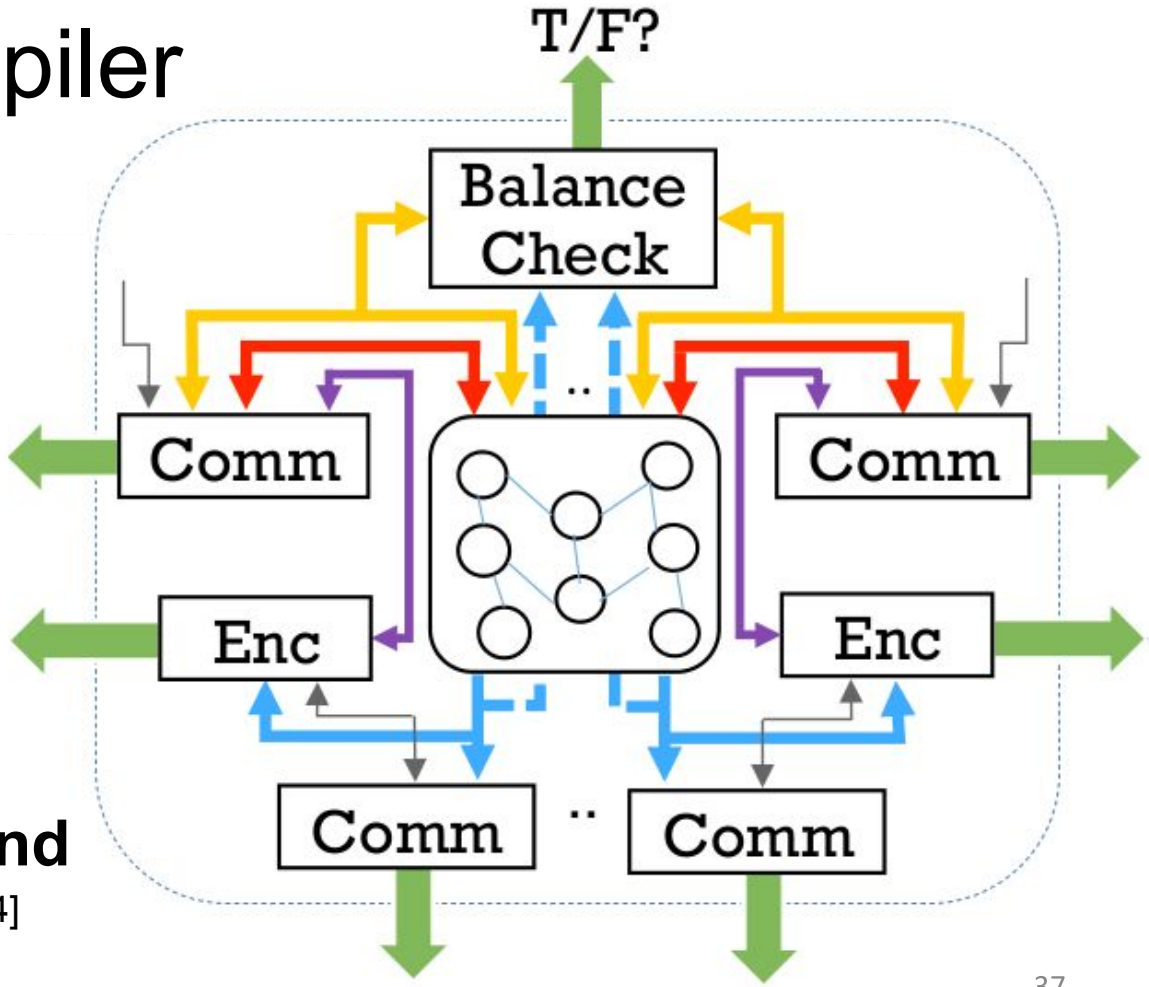
Hawk compiler

Step 1: User provided program compiled to arithmetic circuit

Step 2: Augment arithmetic with ZKP-friendly crypto

Step 3: libsnark backend

[BCTV14]



Performance

@ 112-bit security

100-user auction or crowd-funding

Auctioneer: **2.8 min**

User: **40 secs** (independent of # users)

On-chain cost: **≈ 1.5 seconds**

≈ 220 kilobytes

Main overhead: ZK proofs

More in the paper:

- Formal models and proofs in UC
  including multi-party variation
- Efficient UC-secure zero knowledge proofs
- SNARK-friendly cryptography

Software available:

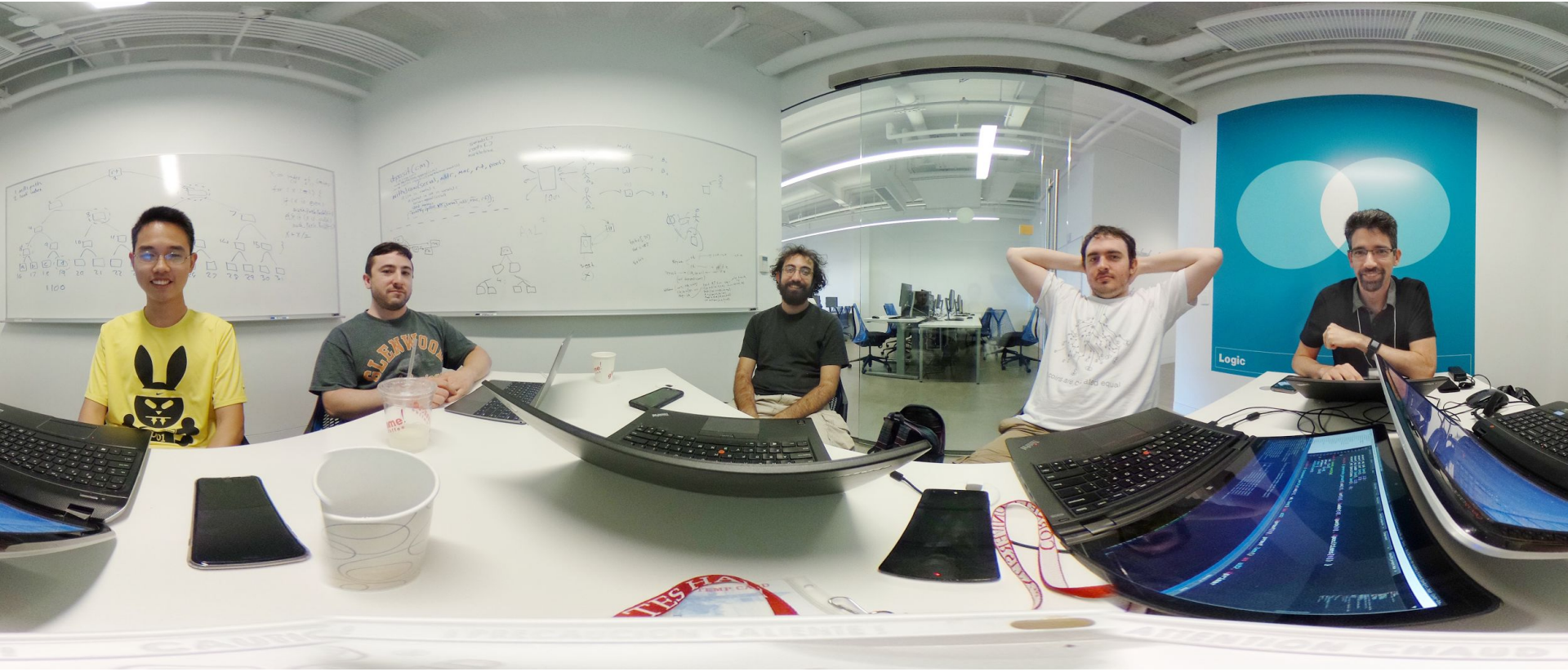
jSNARK: http://oblivm.com/jsnark

Hawk: http://oblivm.com/hawk (soon)

IC3 ZCASH

zkSNARKs in Ethereum

Sean Bowe | Jul 28, 2016



HAWK: Privacy-Preserving Smart Contracts

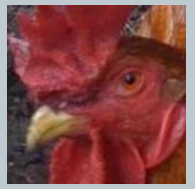
https://eprint.iacr.org/2015/675

Software available:

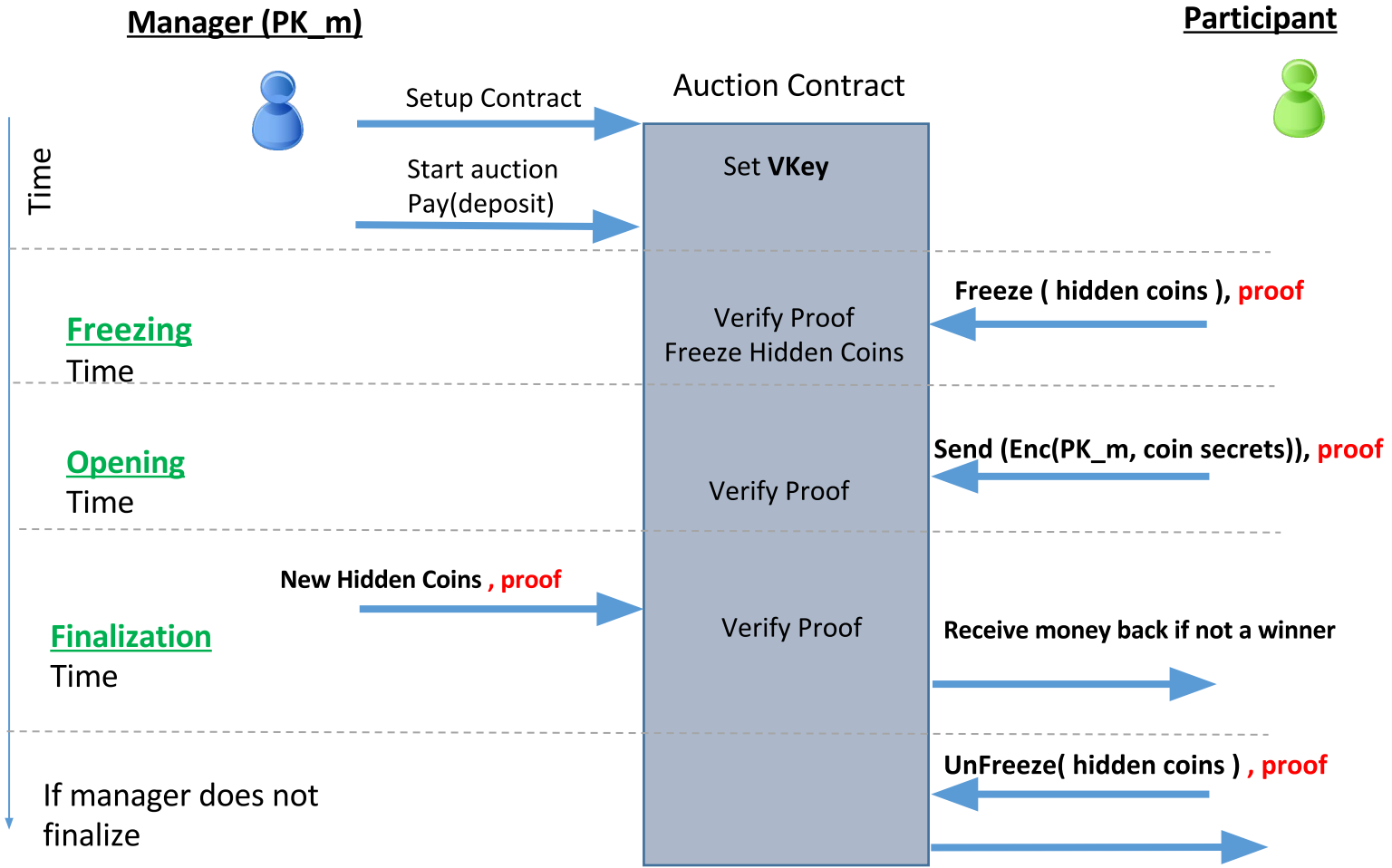
jSNARK: http://oblivm.com/jsnark

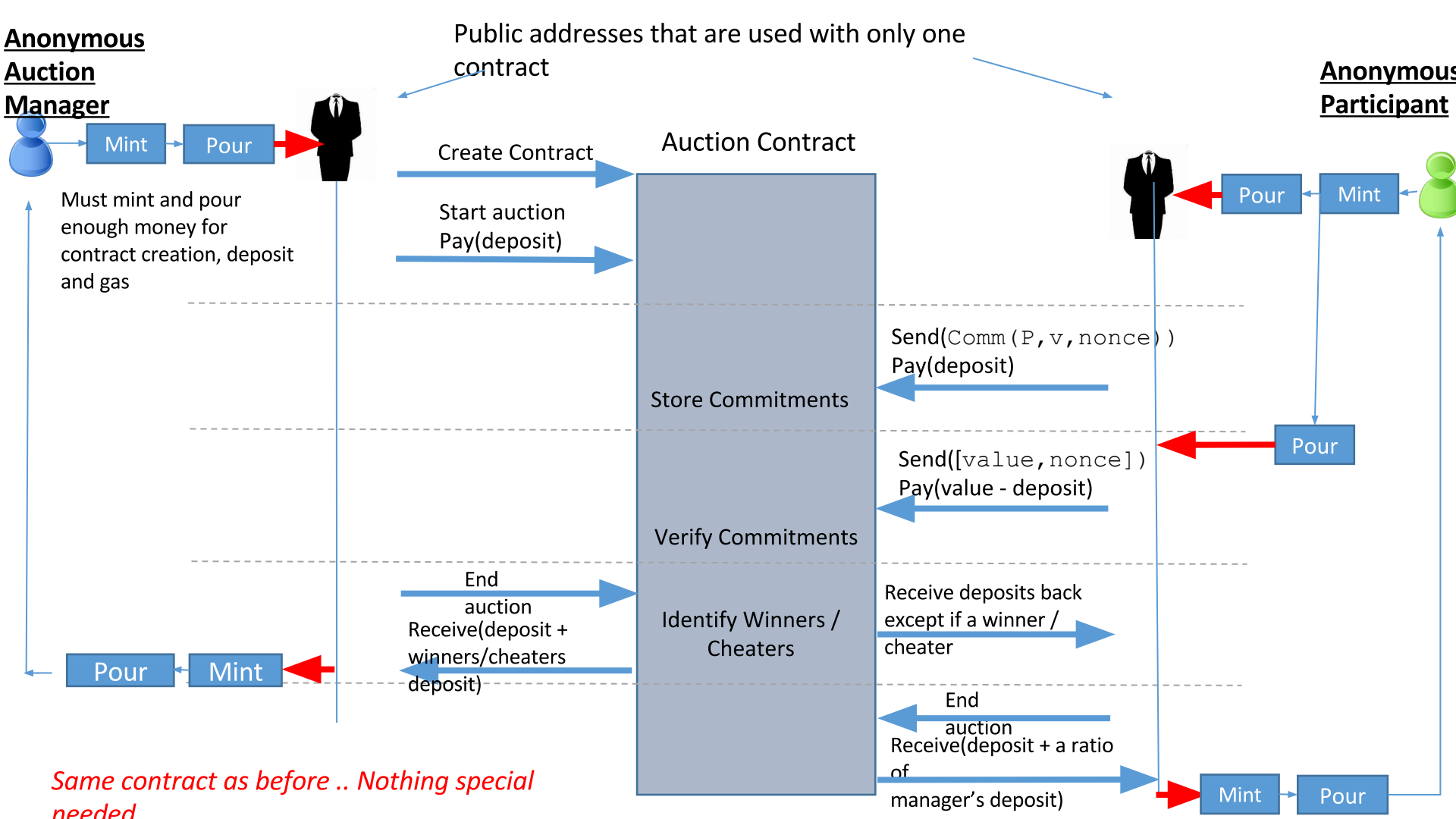
Hawk: http://oblivm.com/hawk (soon)

@socrates1024



Privacy-preserving Auction




Anonymous Auction Manager
Public addresses that are used with only one contract
Anonymous Participant
Mint
Pour
Create Contract
Auction Contract
Must mint and pour enough money for contract creation, deposit and gas
Start auction
Pay(deposit)
Pour
Mint
Send(Comm(P,v,nonce))
Pay(deposit)
Store Commitments
Pour
Send([value,nonce])
Pay(value - deposit)
Verify Commitments
End auction
Receive deposits back except if a winner / cheater
Identify Winners / Cheaters
Receive(deposit + winners/cheaters deposit)
Pour
Mint
End auction
Receive(deposit + a ratio of manager's deposit)
Mint
Pour
Same contract as before .. Nothing special needed

Weird Machines

The Language-theoretic approach (LANGSEC) regards the Internet insecurity epidemic as a consequence of ad hoc *programming of input handling at all layers of network stacks, and in other kinds of software stacks.*