Blockchains & Cryptocurrencies

Smart Contracts & Ethereum

Instructor: Matthew Green & Abhishek Jain
Spring 2023

News?

TECH

Crypto firm Paxos to face SEC charges, ordered to stop minting Binance stablecoin

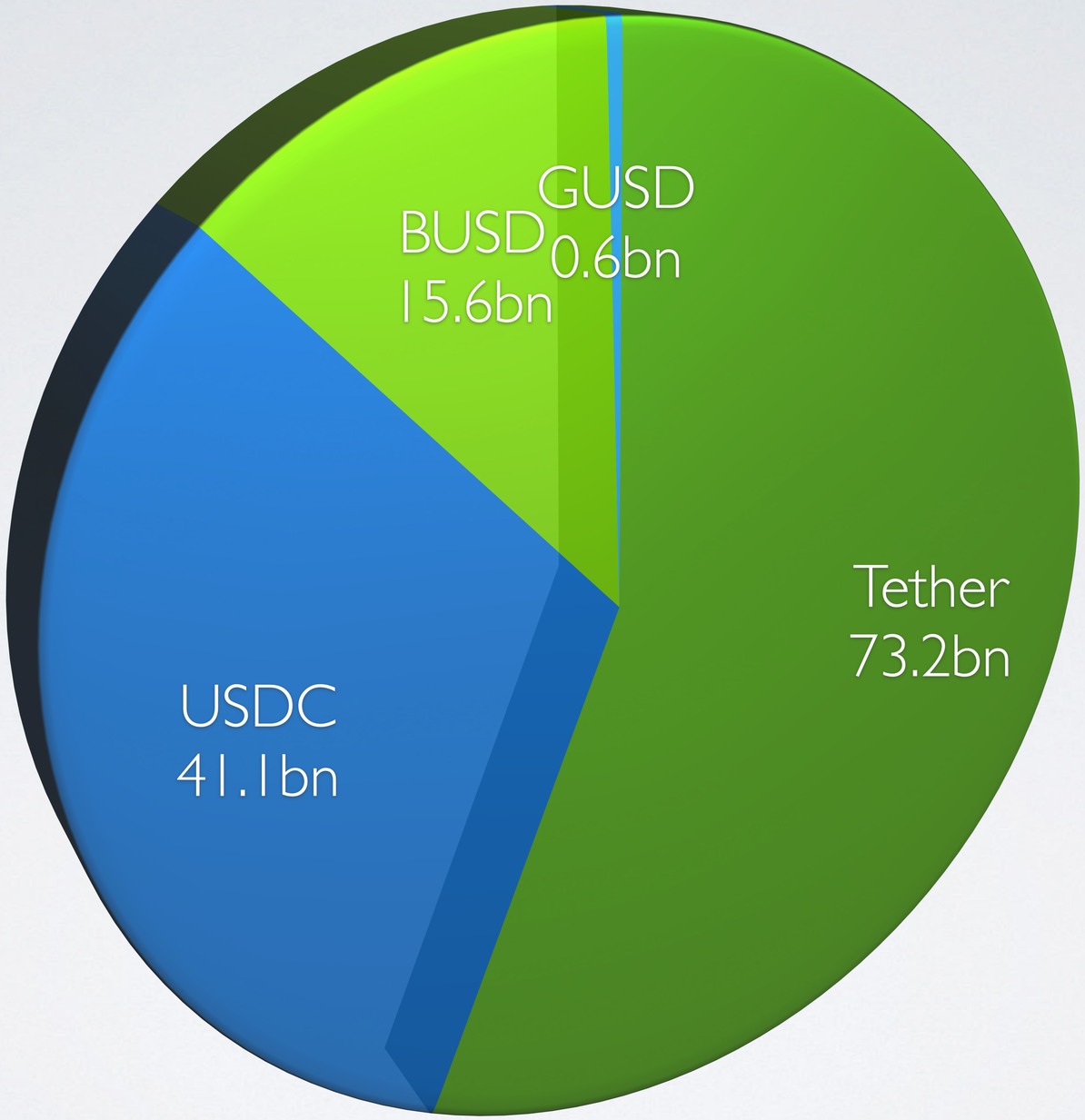
PUBLISHED MON, FEB 13 2023•9:19 AM EST | UPDATED MON, FEB 13 2023•4:47 PM EST

Rohan Goswami
@ROGOSWAMI

SHARE

KEY POINTS

- New York state regulators ordered Paxos to stop minting new Binance USD tokens, Binance CEO Changpeng Zhao said on Twitter.
- The Ethereum-built BUSD tokens are backed by some $16 billion worth of Treasurys and Treasury Reverse Repurchase Agreements.
- The regulator said it issued the order Monday “as a result of several unresolved issues related to Paxos’ oversight of its relationship with Binance.”



Stablecoin marketshare as of February 2023, data: CoinMarketCap

BUSD Binance USD (100%)

Proof of Assets

5,435,500,000 BUSD

0x47ac0fb4f2d84...ab3c24507a6d503 ETH

Wrapped Token

544,069,553 BUSD

bnb19v2ayq6k6e5...kpqn3n6mxheegvj BEP2

4,771,929,475 BUSD

0xe9e7cea3dedca...99bd69add087d56 BEP20

11,500,000 BUSD

0x9C9e5fD8bbc25...117Defa39d2db39 AVAX C-Chain

6,000,000 BUSD

0x9C9e5fD8bbc25...117Defa39d2db39 POLYGON

101,000,000 BUSD

TMz2SWatiAtZVVc...psbVtYwUPT9EdjH TRON

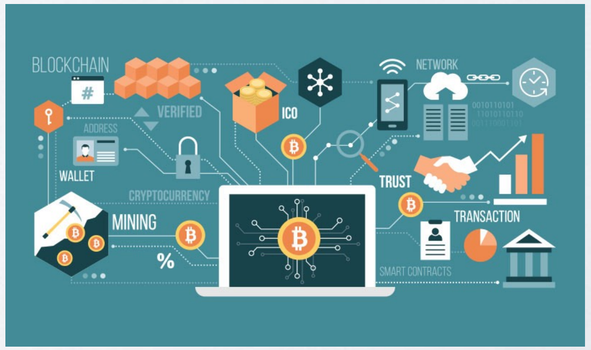
1,000,000 BUSD

0x9c9e5fd8bbc25...117defa39d2db39 OP

https://www.binance.com/en/collateral-btokens

Today

- From Bitcoin to smart contracts
- Ethereum
- Applications of smart contracts



Review: Bitcoin

- **Each block is a list of transactions**
  - Each transaction consumes one or more inputs;
  - Each transaction includes a set of outputs (amount + destination)
  - Input consumption has conditions (e.g., valid script, typically enforcing valid signature)

Review: Bitcoin

- **Each block is a list of transactions**
  - Each transaction consumes one or more inputs;
  - Each transaction includes a set of outputs (amount + destination)
  - Input consumption has conditions (e.g., valid signature)

In practice, each input/output has script:

ScriptPubKey (outputs)
ScriptSig (inputs)

Review: Bitcoin

```
  76       A9             14
OP_DUP OP_HASH160    Bytes to push

89 AB CD EF AB BA AB BA AB BA AB BA AB BA AB BA AB BA   88          AC
                       Data to push                   OP_EQUALVERIFY OP_CHECKSIG
```

- Each transaction includes a set of outputs

```
scriptPubKey: OP_2DUP OP_EQUAL OP_NOT OP_VERIFY OP_SHA1 OP_SWAP OP_SHA1 OP_EQUAL
scriptSig: <preimage1> <preimage2>
```

- Input consumption has conditions (e.g., valid signature)

```
scriptPubKey: <expiry time> OP_CHECKLOCKTIMEVERIFY OP_DROP OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
scriptSig: <sig> <pubKey>
```

ScriptPubKey (outputs)
ScriptSig (inputs)

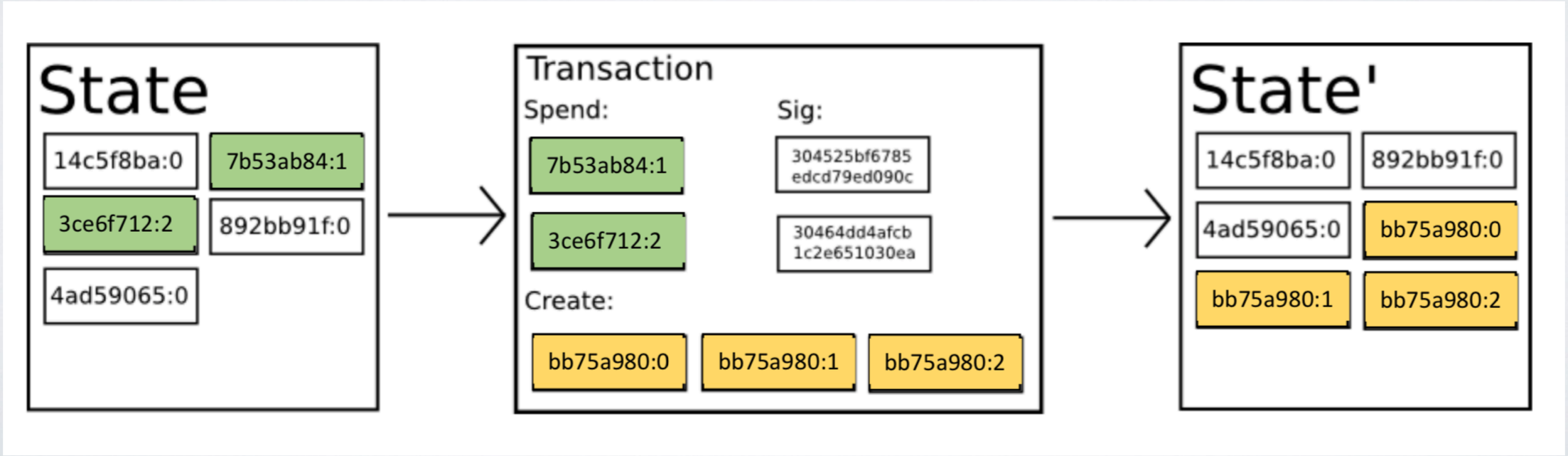
Review: Bitcoin script

- **Bitcoin script allows us to attach conditions to payments**
  - However script is deliberately *limited*
  - Stack-based FORTH-type language, many original opcodes disabled
  - Highly limited access to global data

Bitcoin transactions as “state transition”

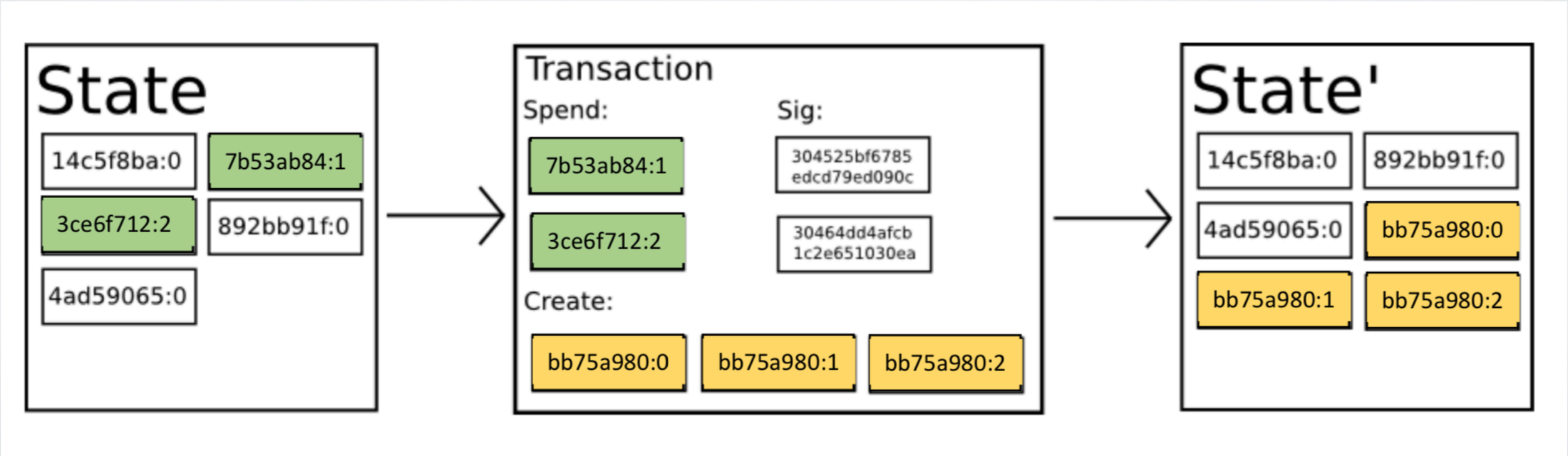
- **Let’s consider each Bitcoin transaction as a state transition function**
  - What is the input state?
  - What is the output state?
  - What does a Transaction do to the state?

- Input state: list of coins available for spending (UTXO set)
- Transaction: set of instructions for updating the UTXO set
- Output state: Updated UTXO set



- Input state: list of coins available for spending (UTXO set)
- Transaction: set of instructions for updating the UTXO set
- Output state: Updated UTXO set

(script determines if a single UTXO is satisfied)

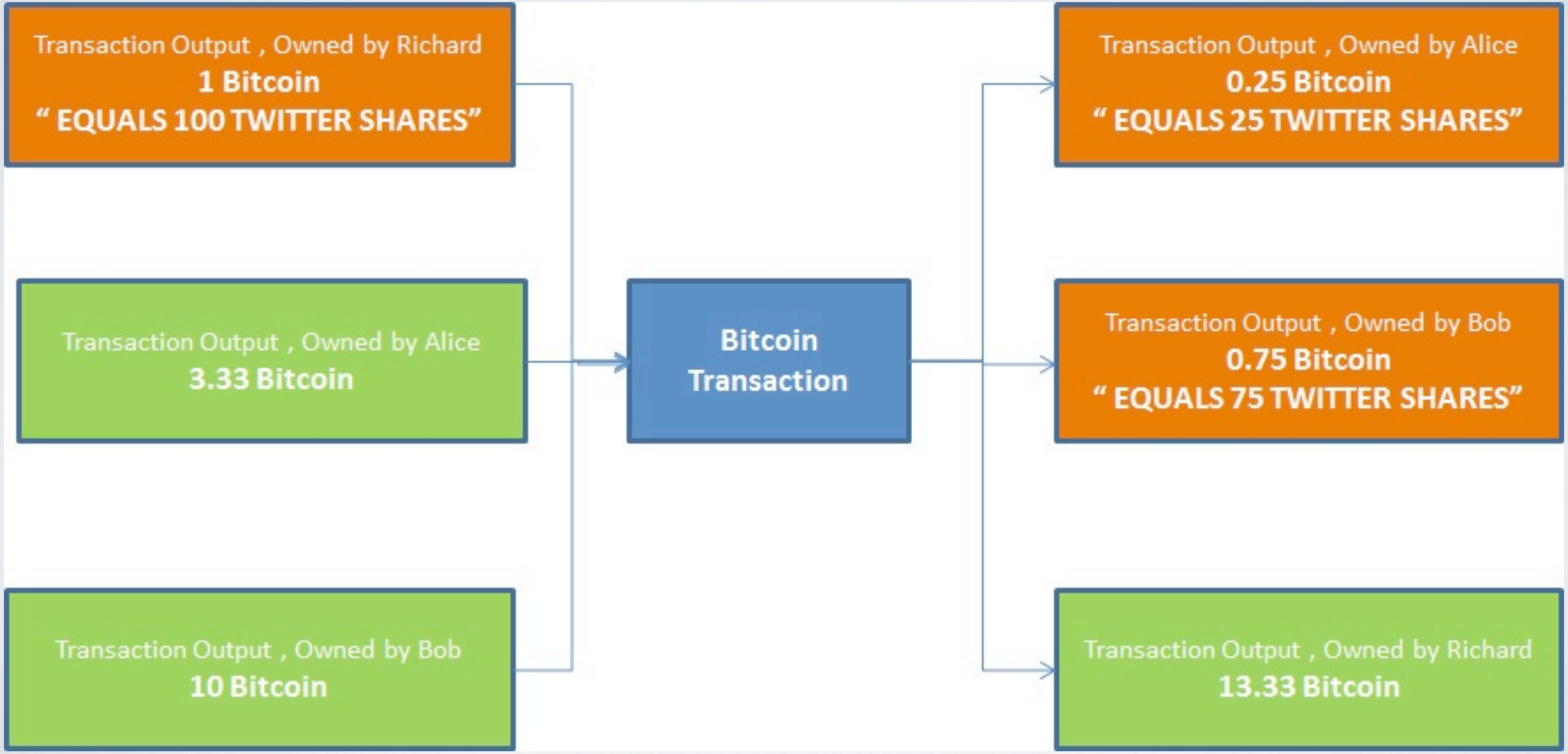


What if we want more powerful programming capabilities?

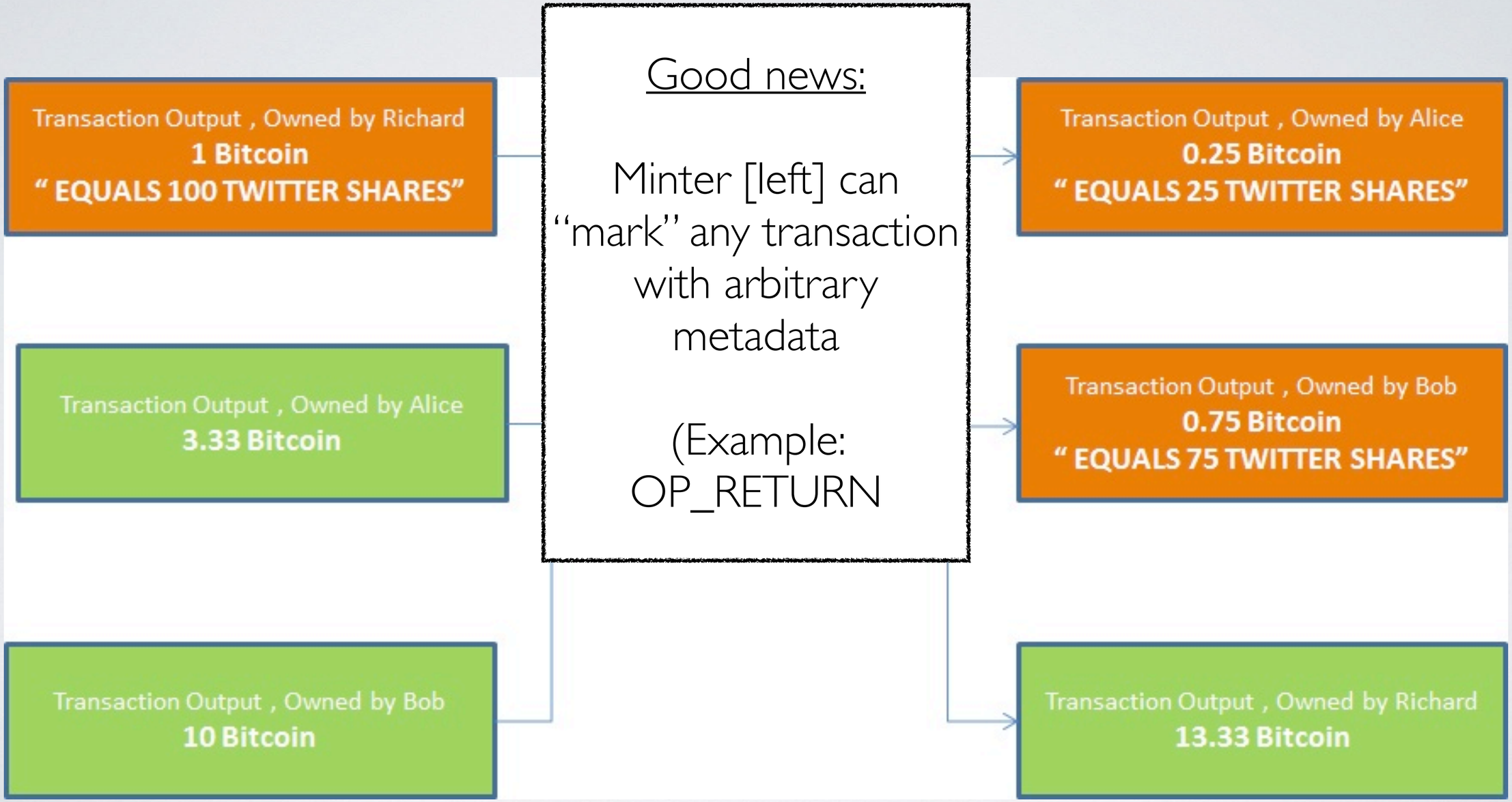
Example: custom tokens

- Bitcoin supports a single currency (BTC)
  - You can spin up a new network (e.g., Litecoin, Dogecoin, etc.)
  - Can we support a second currency on Bitcoin?
    - Applications: coupons, stablecoins, NFTs
    - Major calls: **Mint** (e.g., centralized party), **Burn**, **Pay**

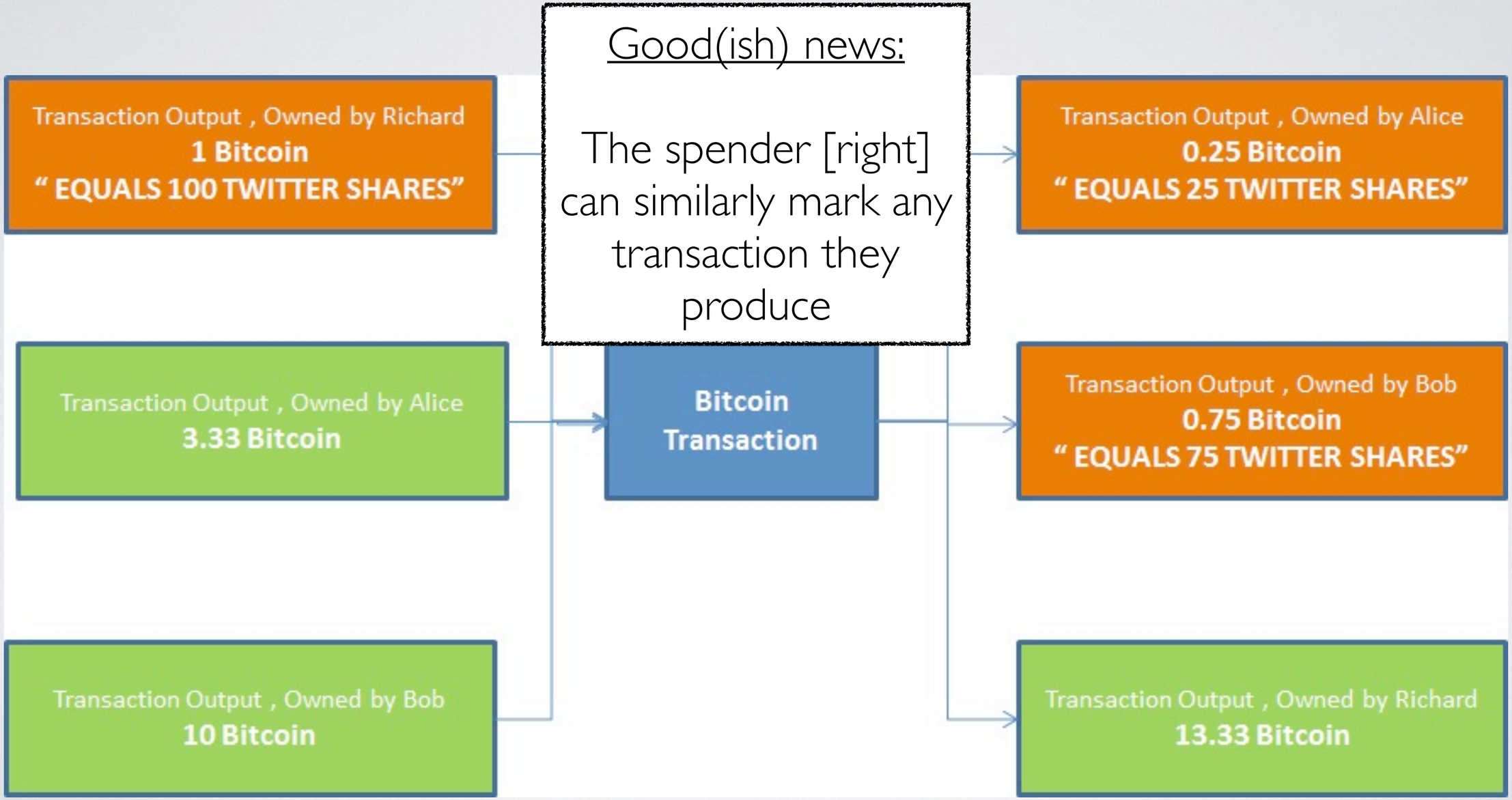
“Colored coins” on Bitcoin



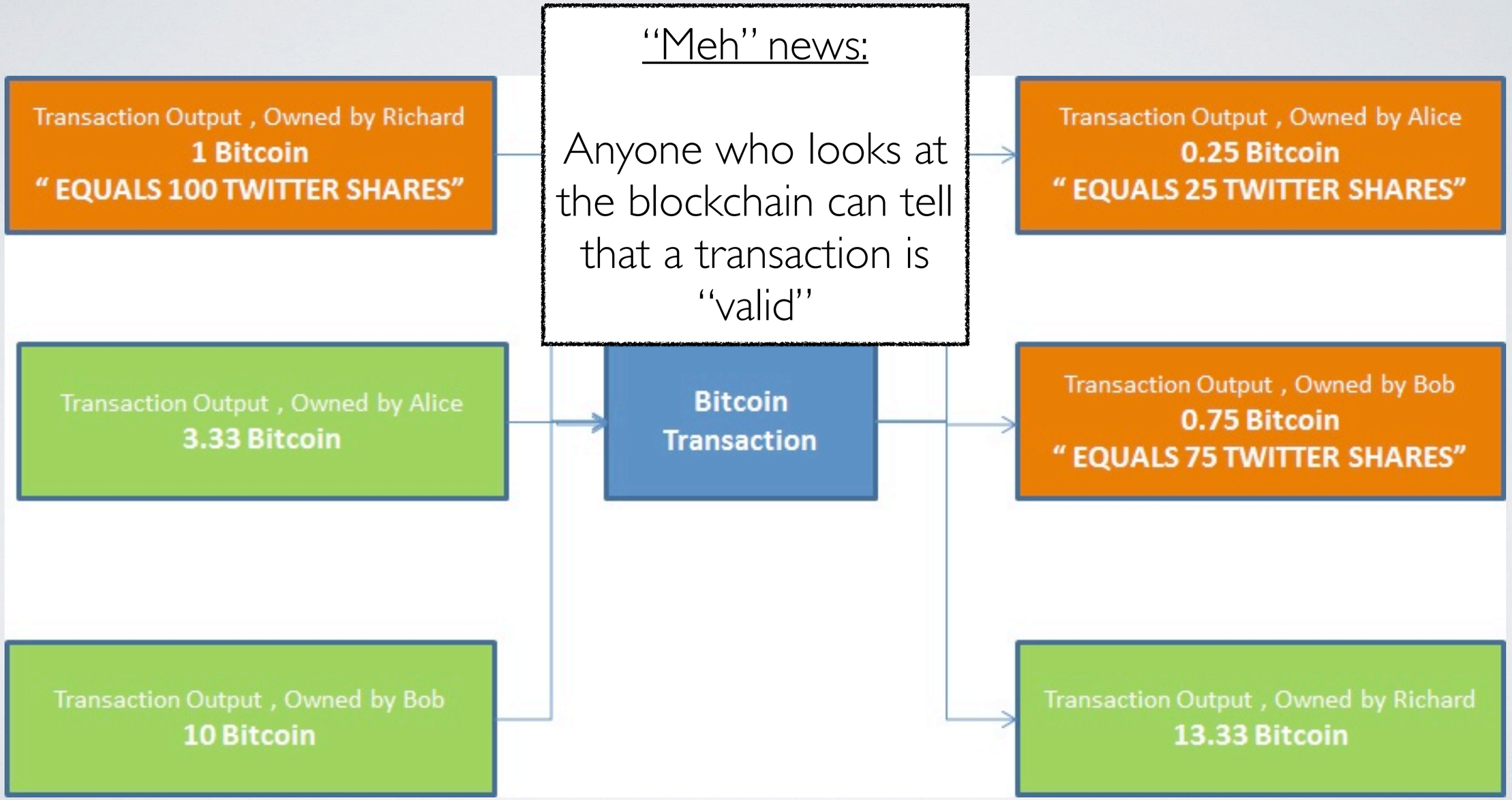
"Colored coins" on Bitcoin



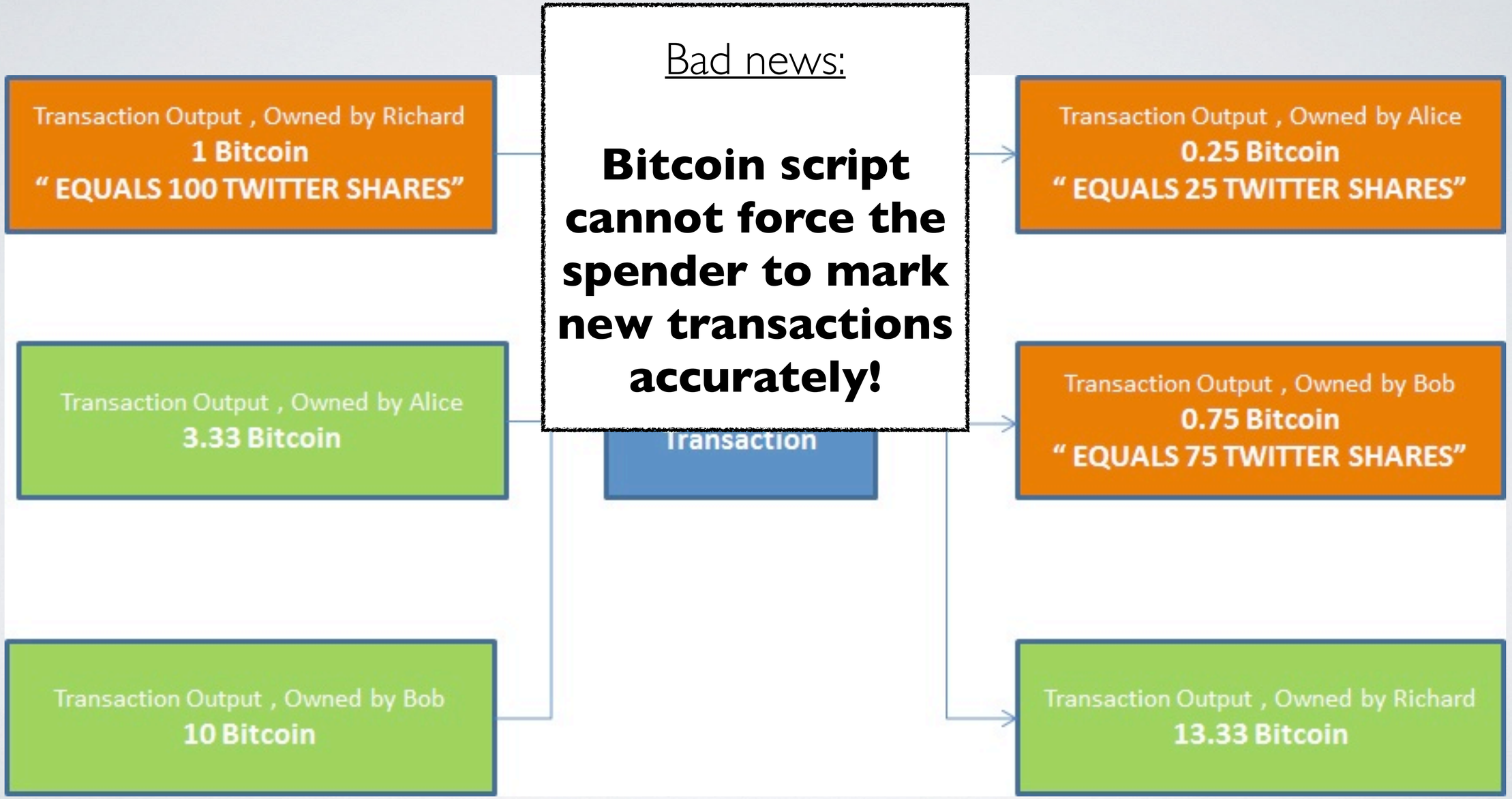
“Colored coins” on Bitcoin



"Colored coins" on Bitcoin



“Colored coins” on Bitcoin



Example 2: prediction market

- Simple program that executes the following:
  - Accepts "bets" on one of two possible outcomes (e.g., Presidential election outcomes)
  - Maintains odds based on all previous bets (simple calculation)
  - Receives a signed message from a pre-chosen judge (e.g., "Candidate A wins!")
  - Pays funds to all winners

Example 3: decentralized exchange

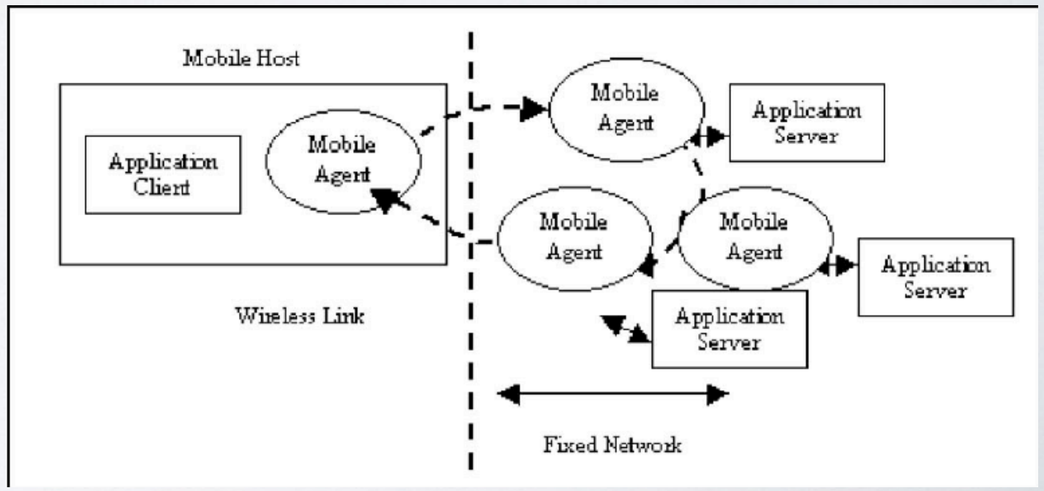
- Assume we have at least two different "tokens"
  - Accept deposits of Token A/Token B
    - Keep a separate for both tokens for each user
  - Build a program that can accept "bids" and "asks" to exchange Token A for Token B
  - Match bids to complete trades
  - Allow withdrawals of the exchanged tokens

“Smart contracts”

- Idea proposed by Nick Szabo (1994)
  - If two users wish to establish a legal contract (e.g., escrow funds, pay out on specific conditions) they can write those conditions in software
  - The software will run autonomously, respond to inputs, evaluate the conditions
  - Contract program can receive and pay out funds according to its programming
  - “Code is law”

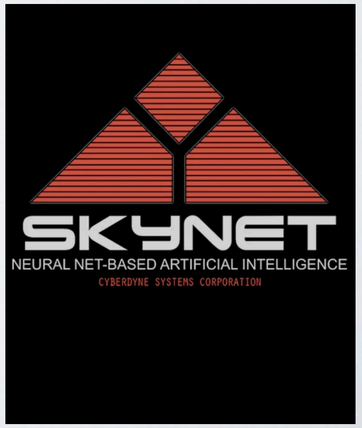
“Mobile Agents”

- Old idea from distributed computing (1990s)
  - Built software programs (“agents”) that can move between servers on a distributed computing system
  - Agents are survivable, can operate even if some computers go down
  - Agent must be portable, secure, and may need to compensate operators for resources consumed

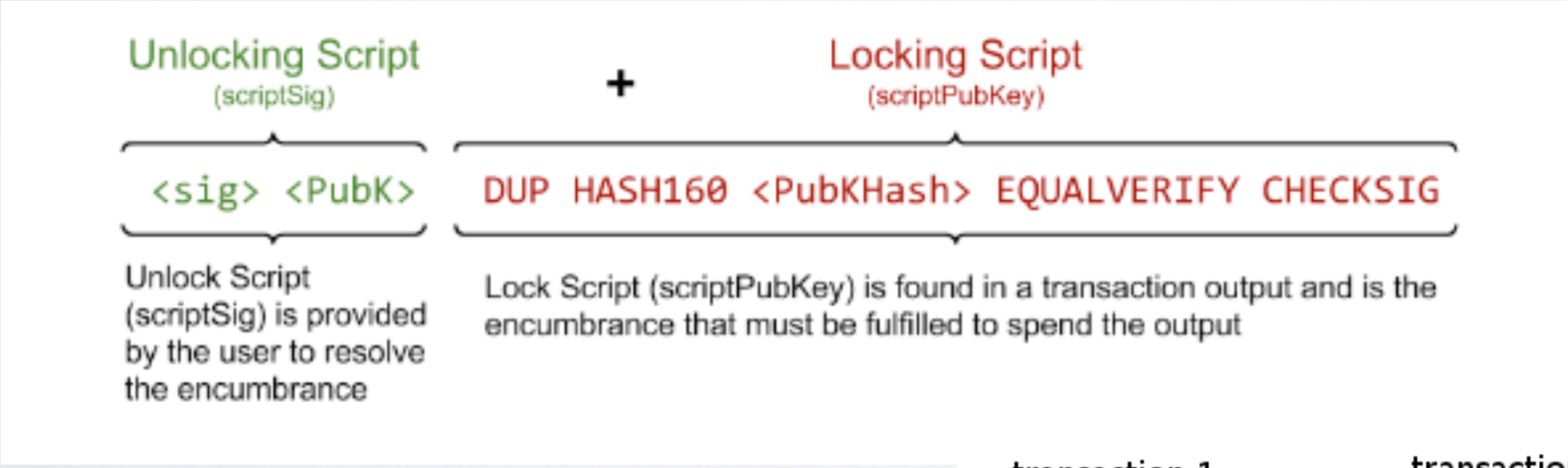


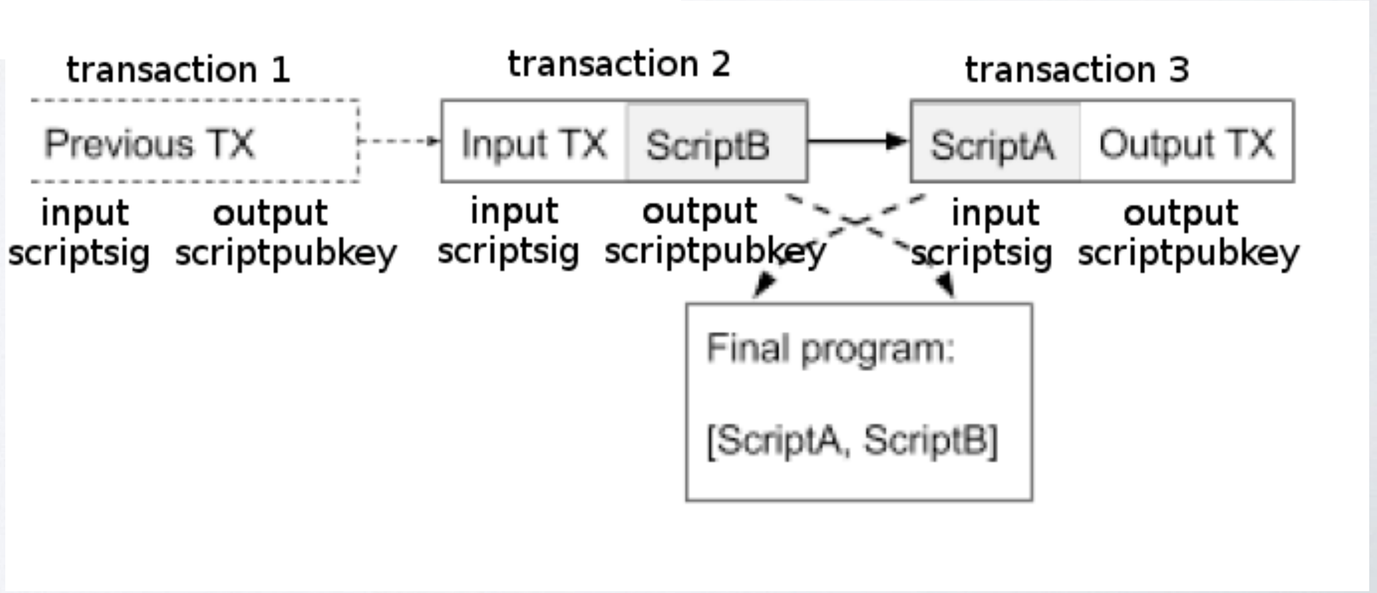
Two visions, same goal

- Run software on a computing network
  - Software is run by volunteer nodes (who come and go)
  - Software is accurate and can't be tampered with
  - Software can compensate users for the resource usage (e.g., payments)
  - Software can do useful things:
    e.g., control the disbursement of funds



Can we implement this on Bitcoin?



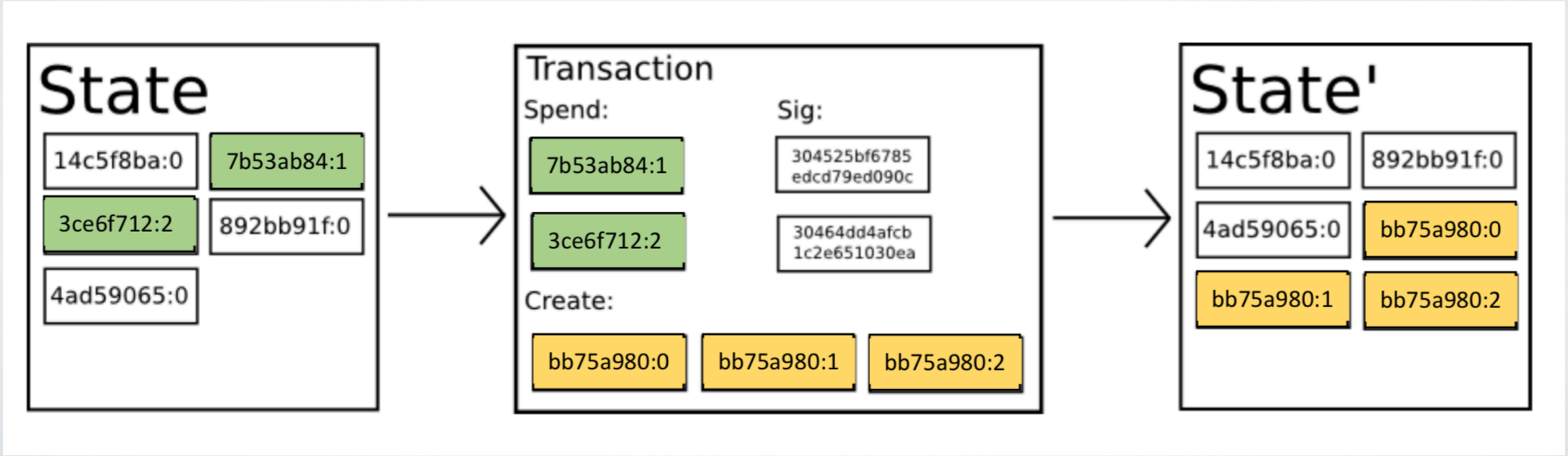


Can we implement this on Bitcoin?

- Each UTXO has a script. However:
  - Each script only runs once (when the UTXO is consumed)
  - **Scripts do not have access to "global" state** (they only have access to scriptSig and scriptPubKey data plus some limited global data like "block height".)
  - Scripts are not Turing complete (even in a limited sense)
  - Limited script opcodes (security reasons)

Bitcoin TX as state update

- Each Bitcoin TX updates the existing state
  - Takes in an input UTXO set (or a subset of it)
  - Outputs a new UTXO set

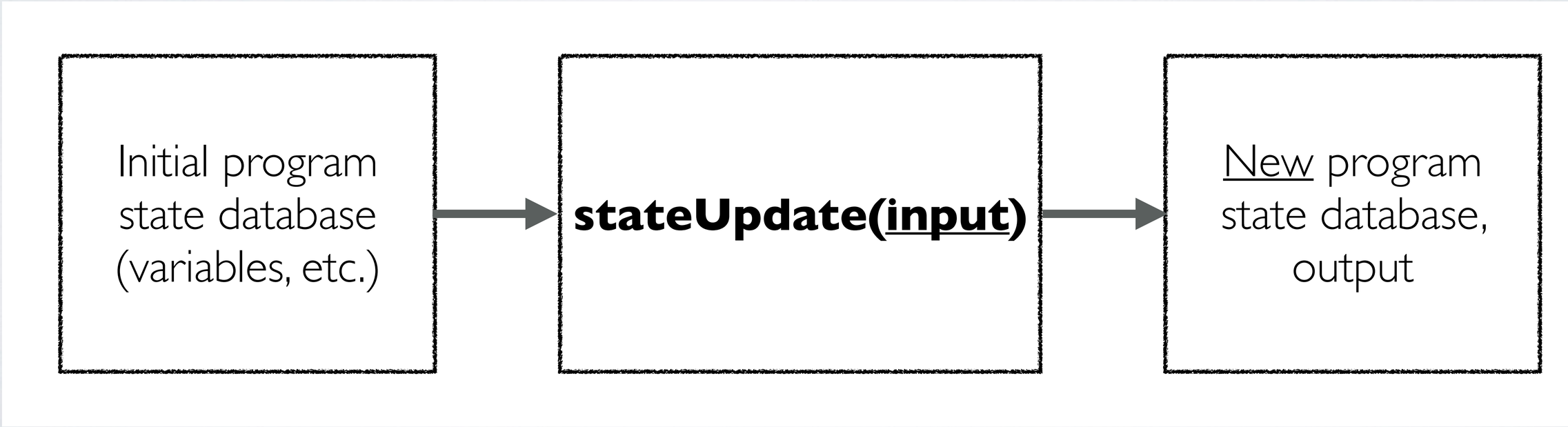


“Towards Ethereum”

Note: system we will discuss next is almost but not entirely unlike Ethereum.

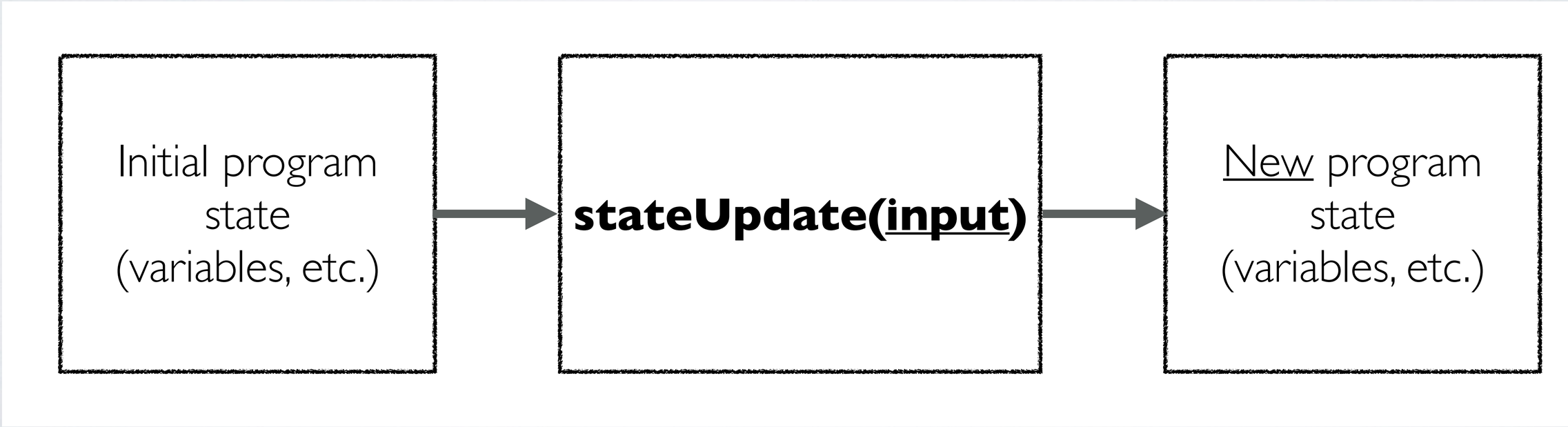
Generalizing programs

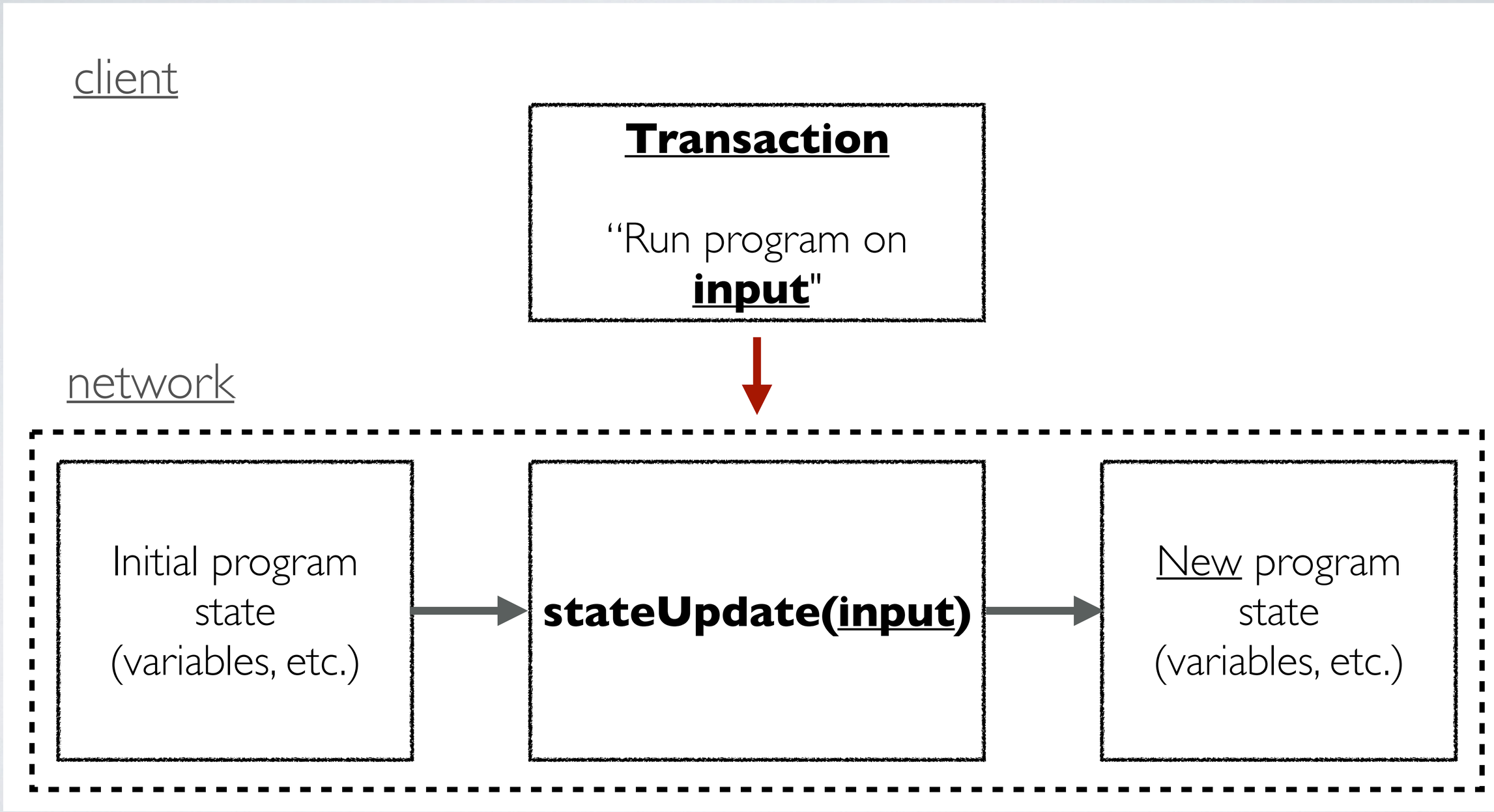
- Programs can be written as state update functions
  - Input: previous state of program, some "call input"
  - Output: new updated state for program, maybe some return value

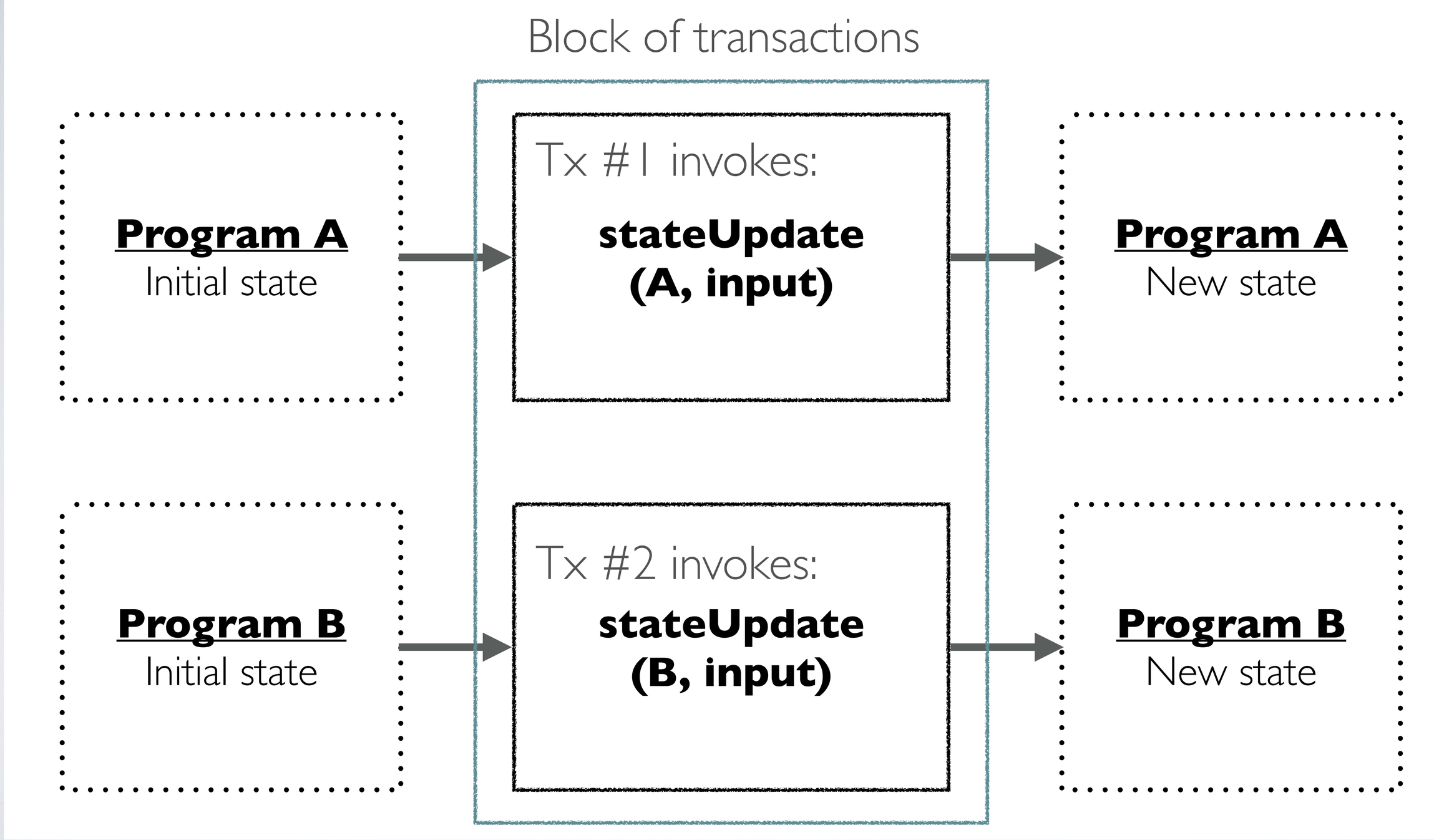


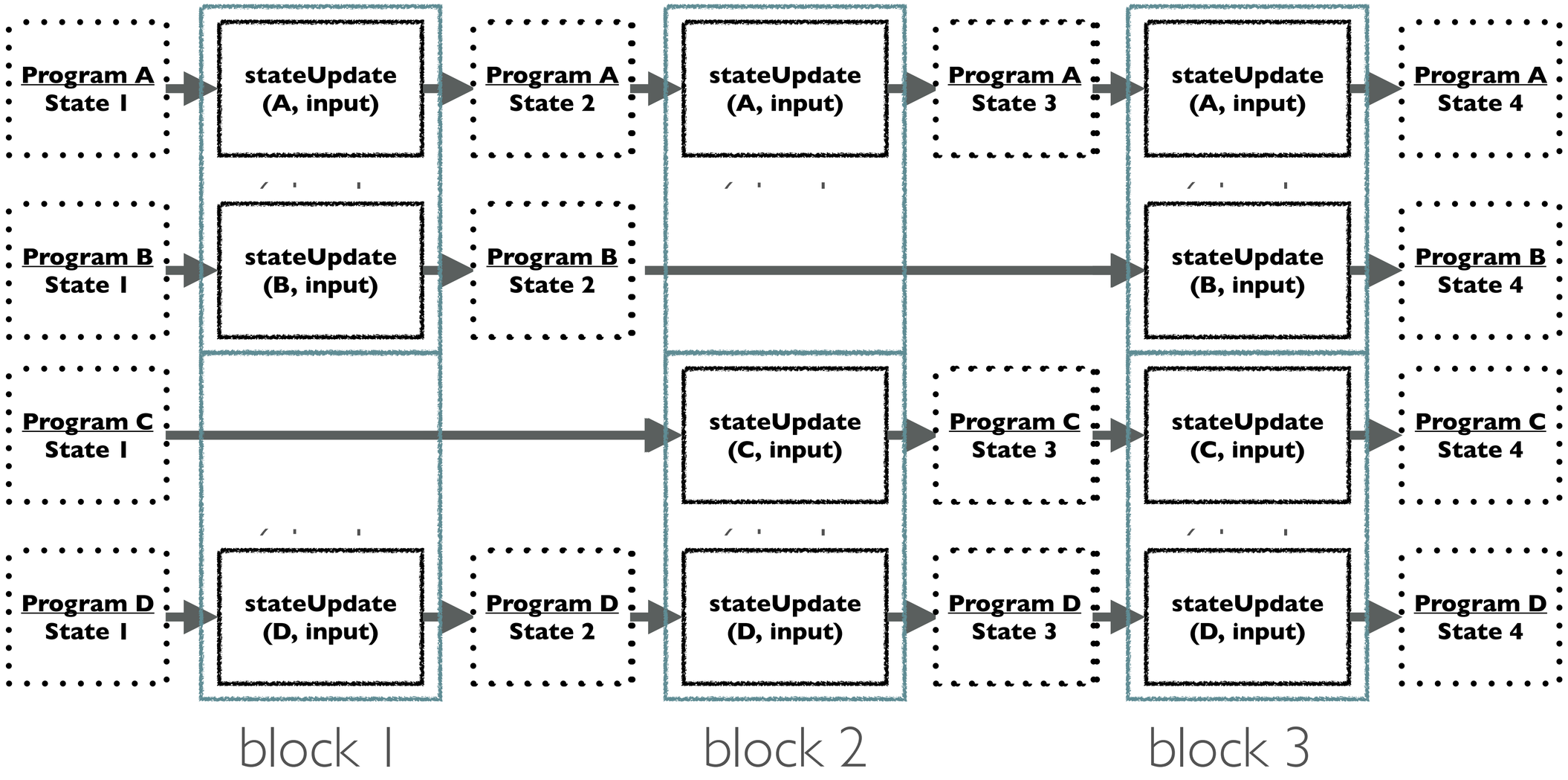
Blockchains for programs

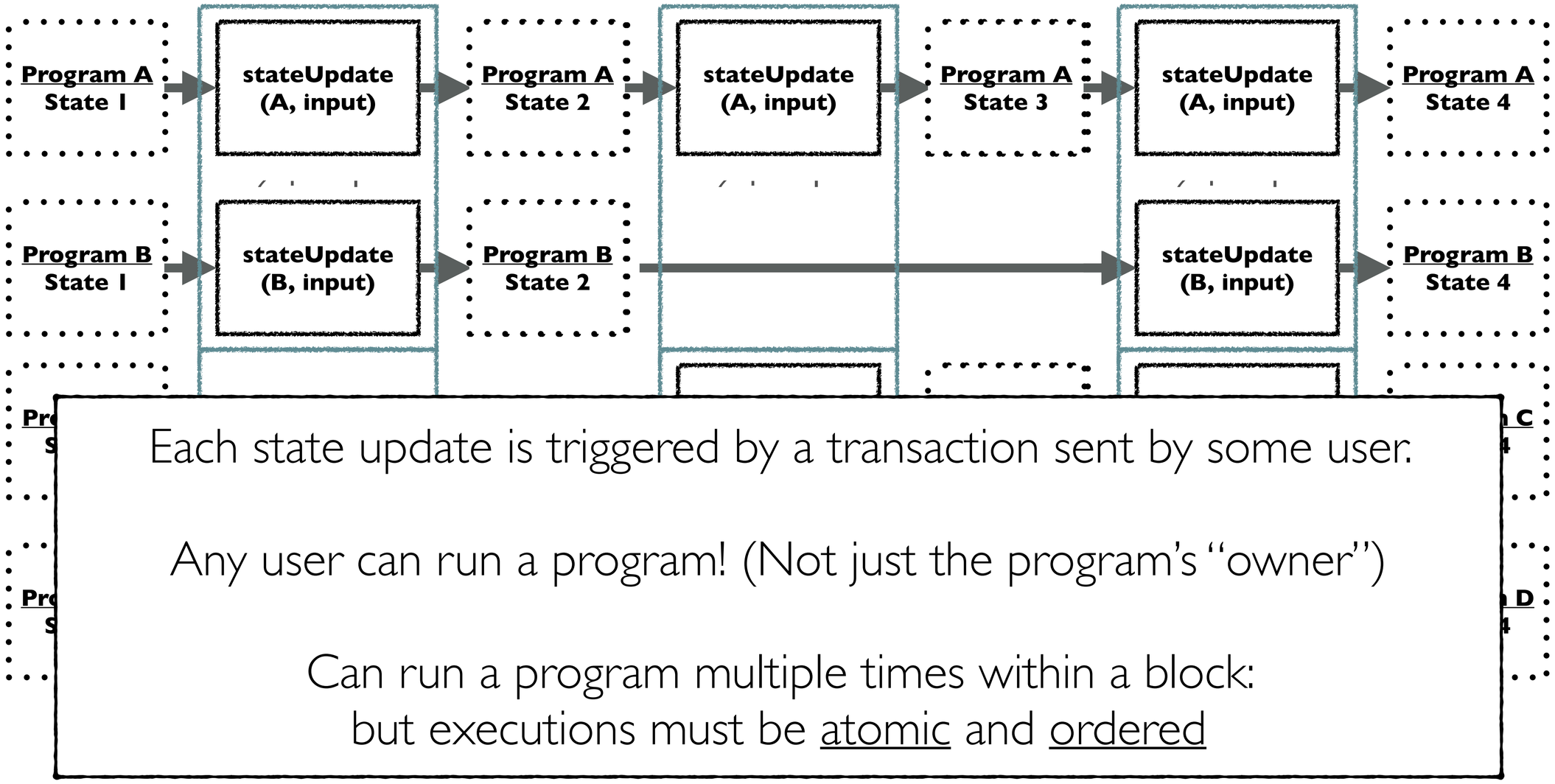
- Assume the blockchain has the program and a database
  - Each transaction simply "invokes" the program on **input**
  - The blockchain runs the program and updates its state




client
Transaction
"Run program on input"
network
Initial program state (variables, etc.)
stateUpdate(input)
New program state (variables, etc.)


Block of transactions
Program A
Initial state
Tx #1 invokes:
stateUpdate
(A, input)
Program A
New state
Program B
Initial state
Tx #2 invokes:
stateUpdate
(B, input)
Program B
New state


Program A
State 1
stateUpdate
(A, input)
Program A
State 2
stateUpdate
(A, input)
Program A
State 3
stateUpdate
(A, input)
Program A
State 4
Program B
State 1
stateUpdate
(B, input)
Program B
State 2
stateUpdate
(B, input)
Program B
State 4
Program C
State 1
stateUpdate
(C, input)
Program C
State 3
stateUpdate
(C, input)
Program C
State 4
Program D
State 1
stateUpdate
(D, input)
Program D
State 2
stateUpdate
(D, input)
Program D
State 3
stateUpdate
(D, input)
Program D
State 4
block 1
block 2
block 3

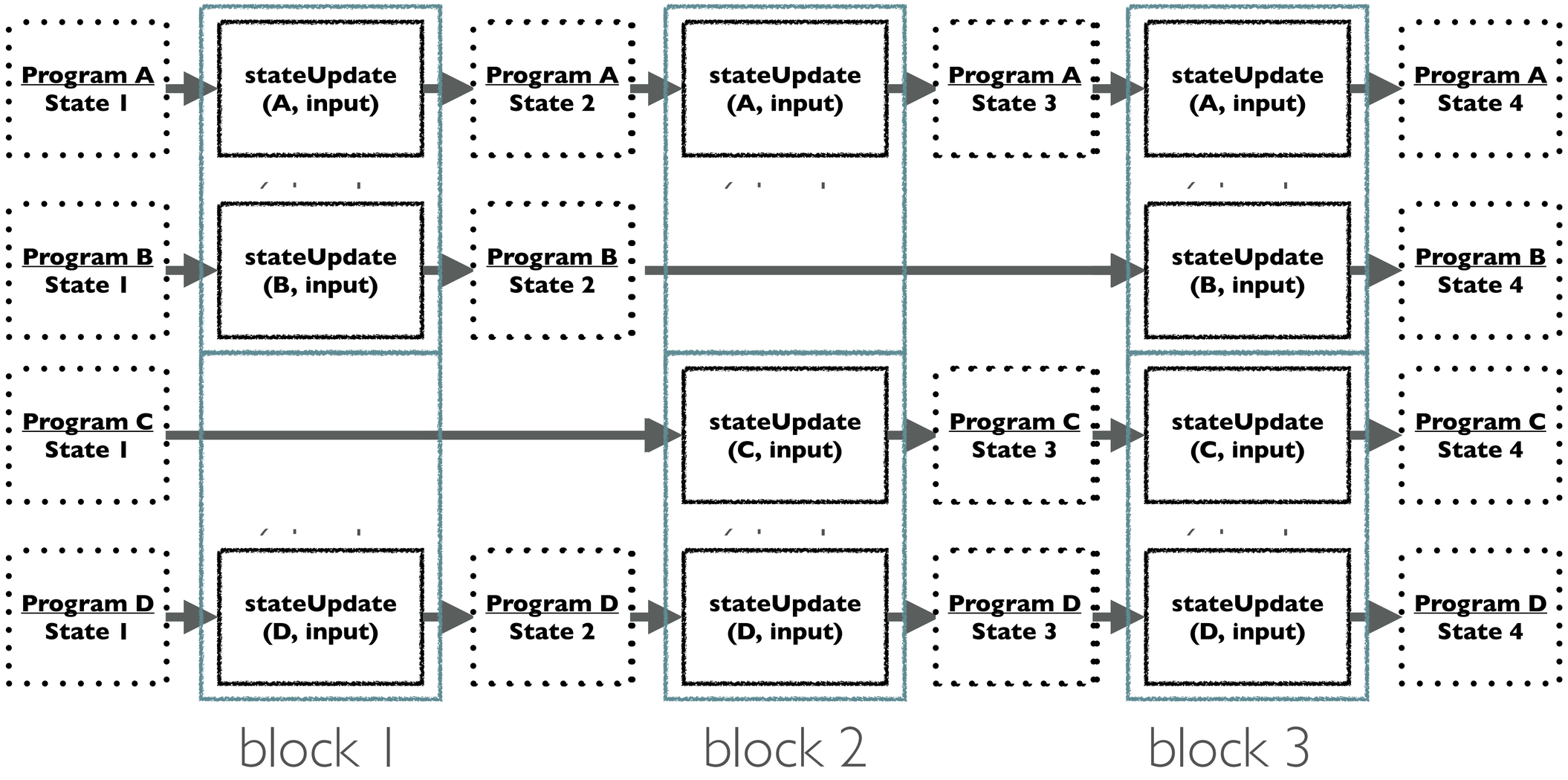


Each state update is triggered by a transaction sent by some user.

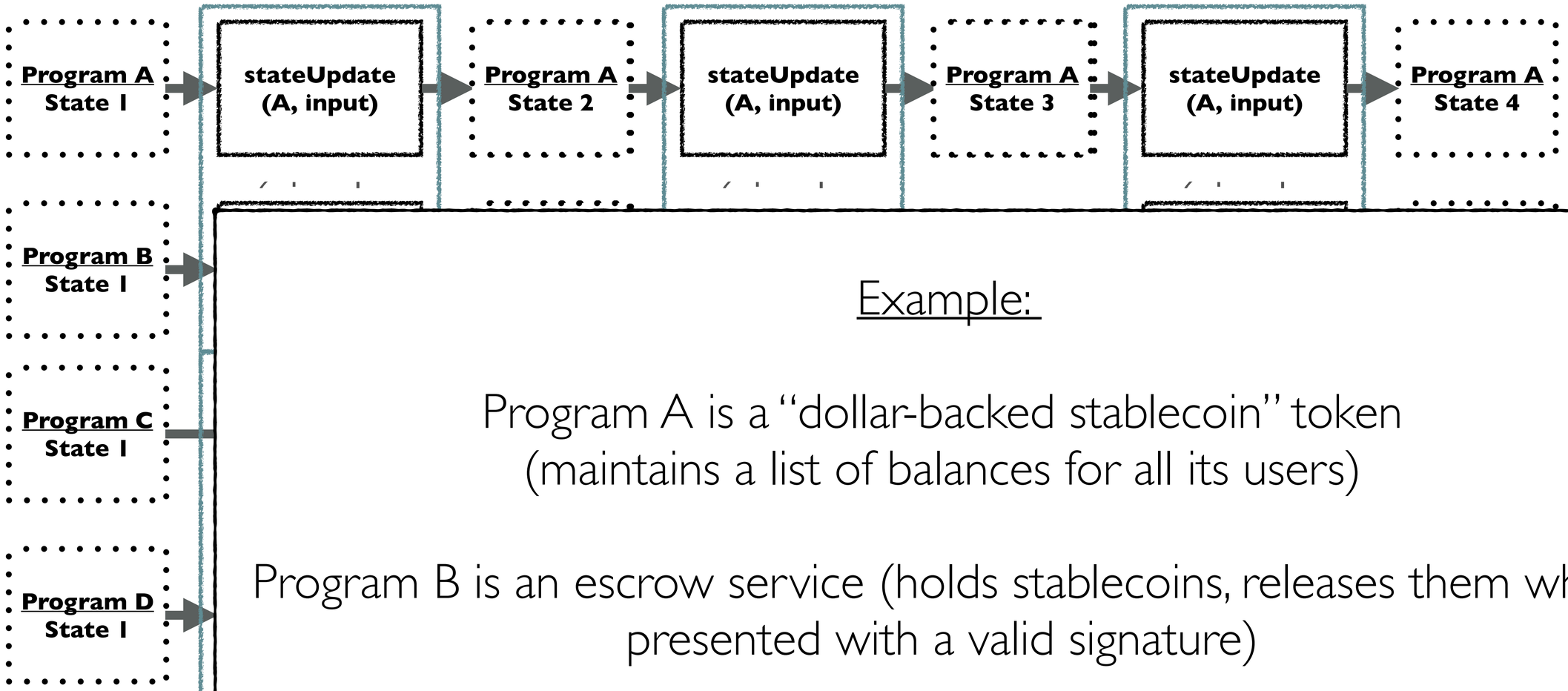
Any user can run a program! (Not just the program's "owner")

Can run a program multiple times within a block:
but executions must be atomic and ordered

What if program A wants to alter the state of program B?



What if program A wants to alter the state of program B?



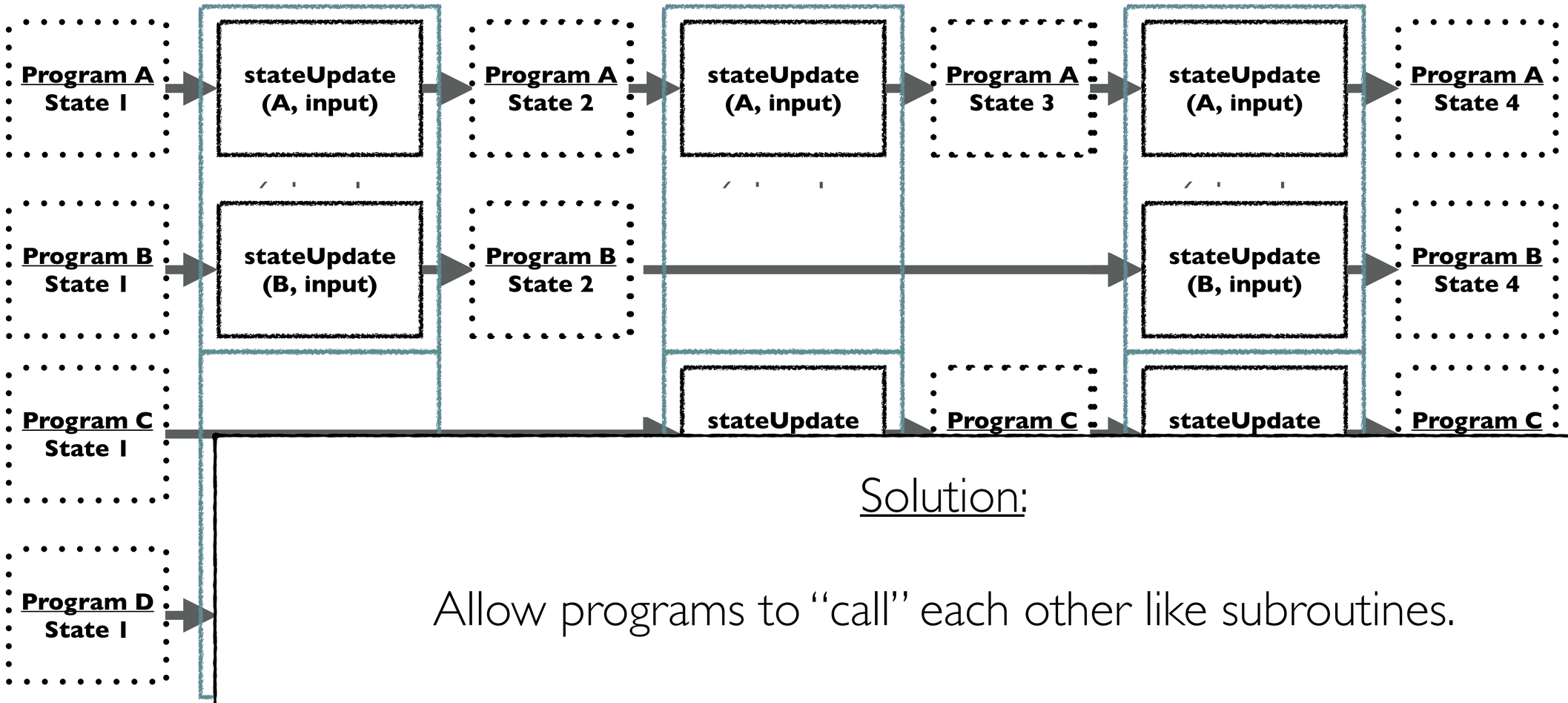
Example:

Program A is a "dollar-backed stablecoin" token
(maintains a list of balances for all its users)

Program B is an escrow service (holds stablecoins, releases them when presented with a valid signature)

How does Program B tell Program A to "pay" someone?

What if program A wants to alter the state of program B?

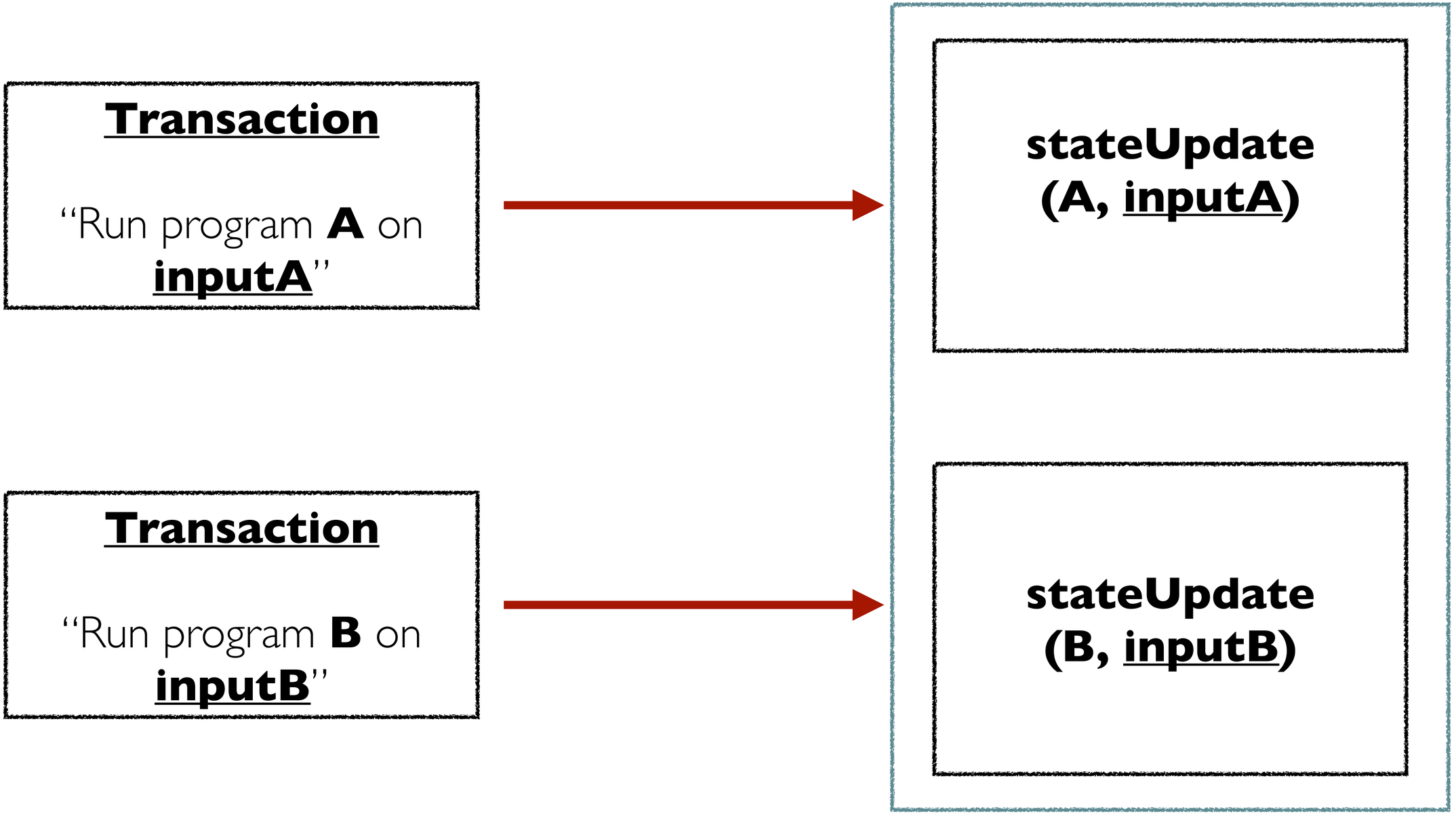


Solution:

Allow programs to "call" each other like subroutines.

Program B can tell A to "pay" another user (e.g., reduce one account balance, increase another user's account balance.)

External transactions



Transaction

"Run program **A** on **input**"

stateUpdate (A, input)

calls program B as a "subroutine"

stateUpdate (B, args)

Transaction

"Run program **A** on **input**"

stateUpdate (A, input)

calls program B as a "subroutine"

args

stateUpdate (B, args)

1. An external user calls program A
2. Program A calls program B as a subroutine

Transaction

"Run program **A** on **input**"

stateUpdate (A, input)

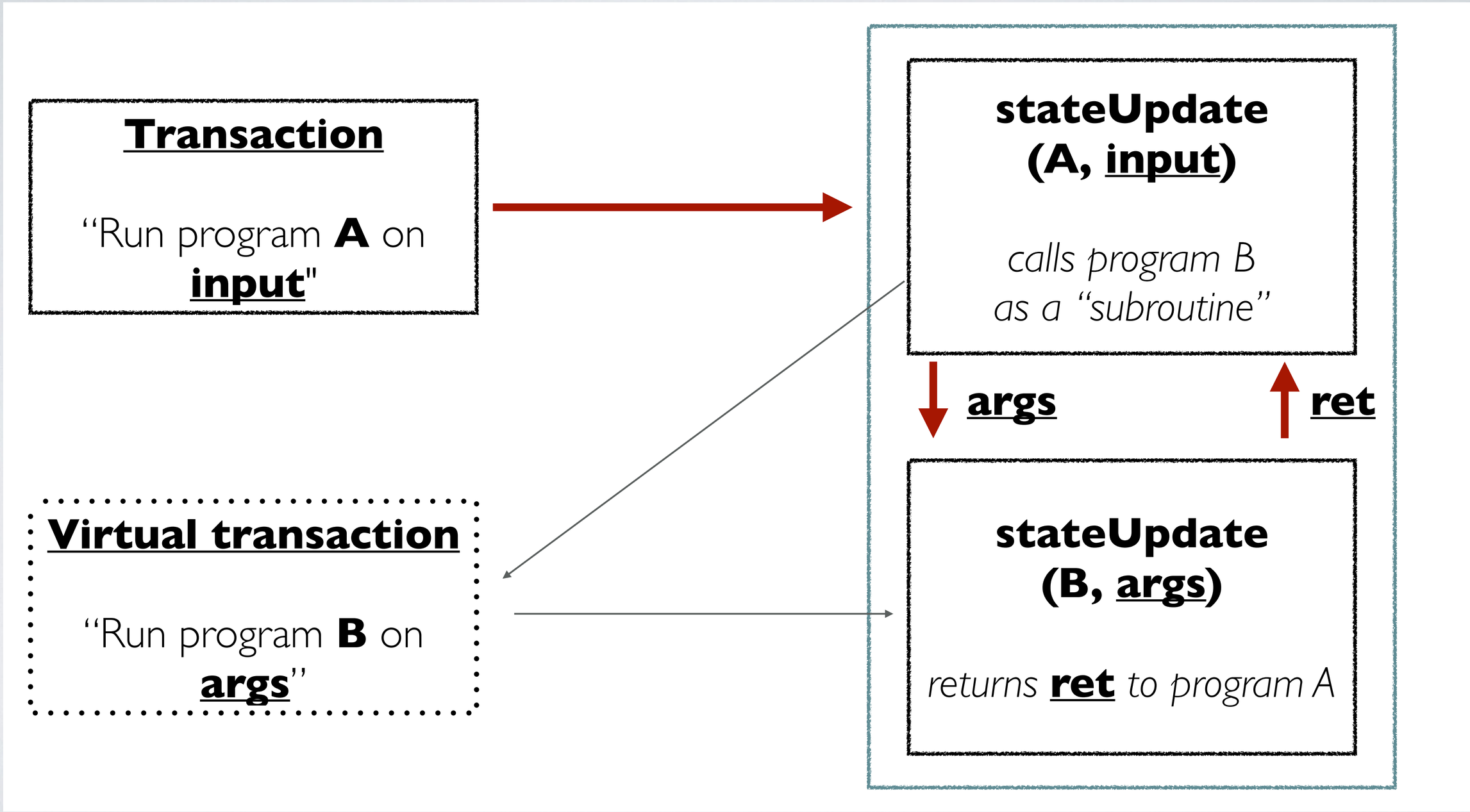
calls program B as a "subroutine"

args ↓ ↑ **ret**

stateUpdate (B, args)

returns **ret** *to program A*

1. An external user calls program A
2. Program A calls program B as a subroutine
3. Program B returns **ret** to A (and updates its own state)


Transaction
"Run program A on input"
stateUpdate (A, input)
calls program B as a "subroutine"
args
ret
Virtual transaction
"Run program B on args"
stateUpdate (B, args)
returns ret to program A

What you should be worried about right now:

When does B actually run?

Immediately?

Does A get "paused" and then B runs (maybe later) and then B generates a virtual transaction to start A up where it left off?

What if the call fails: can A get "stuck" forever?

Trans

"Run pro
inp

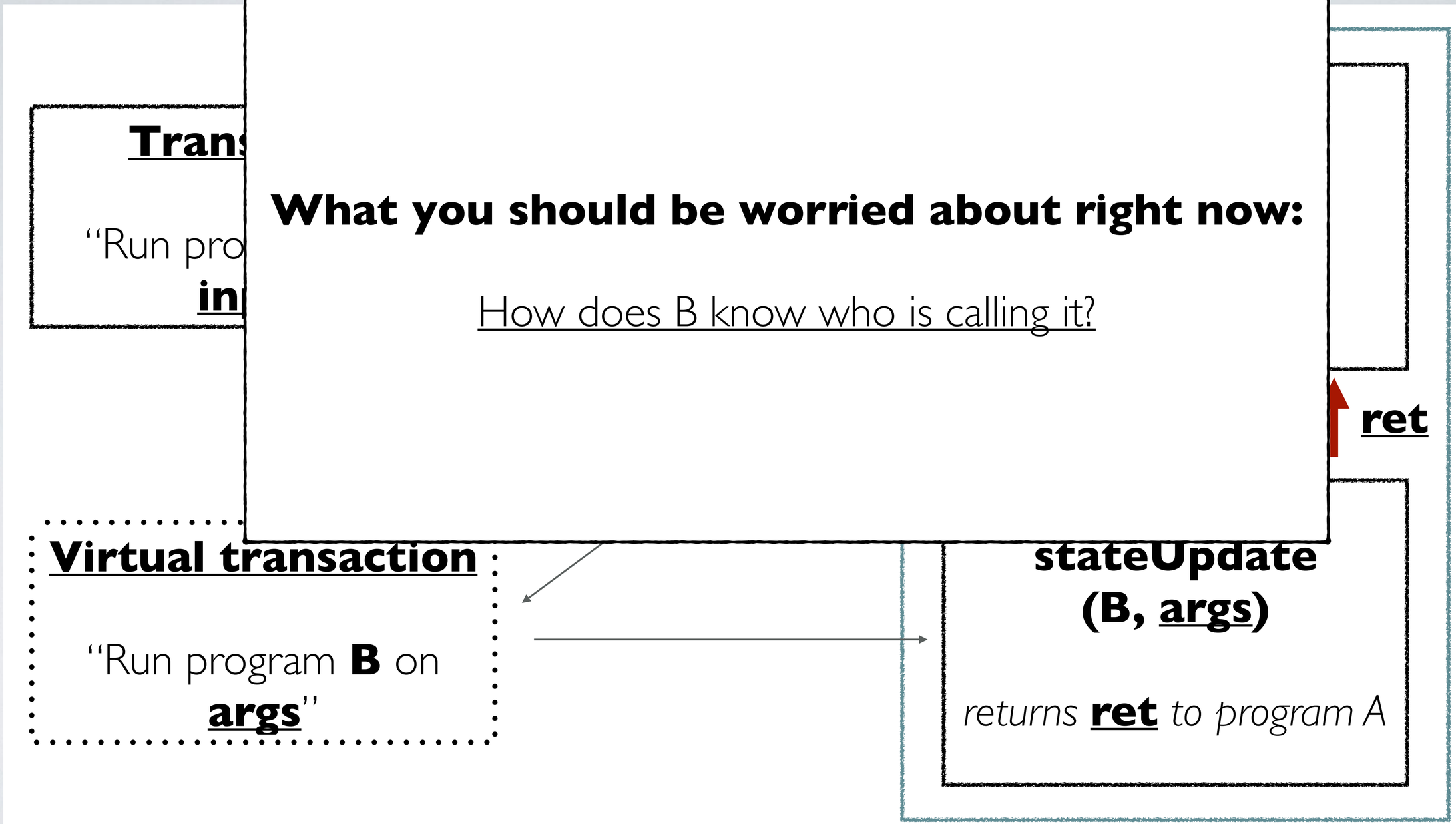
Virtual t

"Run program **B** on **args**"

ret

(B, args)

returns **ret** *to program A*


Trans
"Run pro
in
What you should be worried about right now:
How does B know who is calling it?
ret
Virtual transaction
"Run program B on args"
stateUpdate (B, args)
returns ret to program A

What you should be worried about right now:

How does B know who is calling it?

For real transactions, we can use addresses (public keys) and sign transactions with digital signatures (like Bitcoin)

For "virtual transactions", does the other program have an "address" and can it sign things?

Trans

"Run pro

inp

ret

Virtual transaction

"Run program **B** on **args**"

stateUpdate (B, args)

returns **ret** *to program A*

You should have many questions

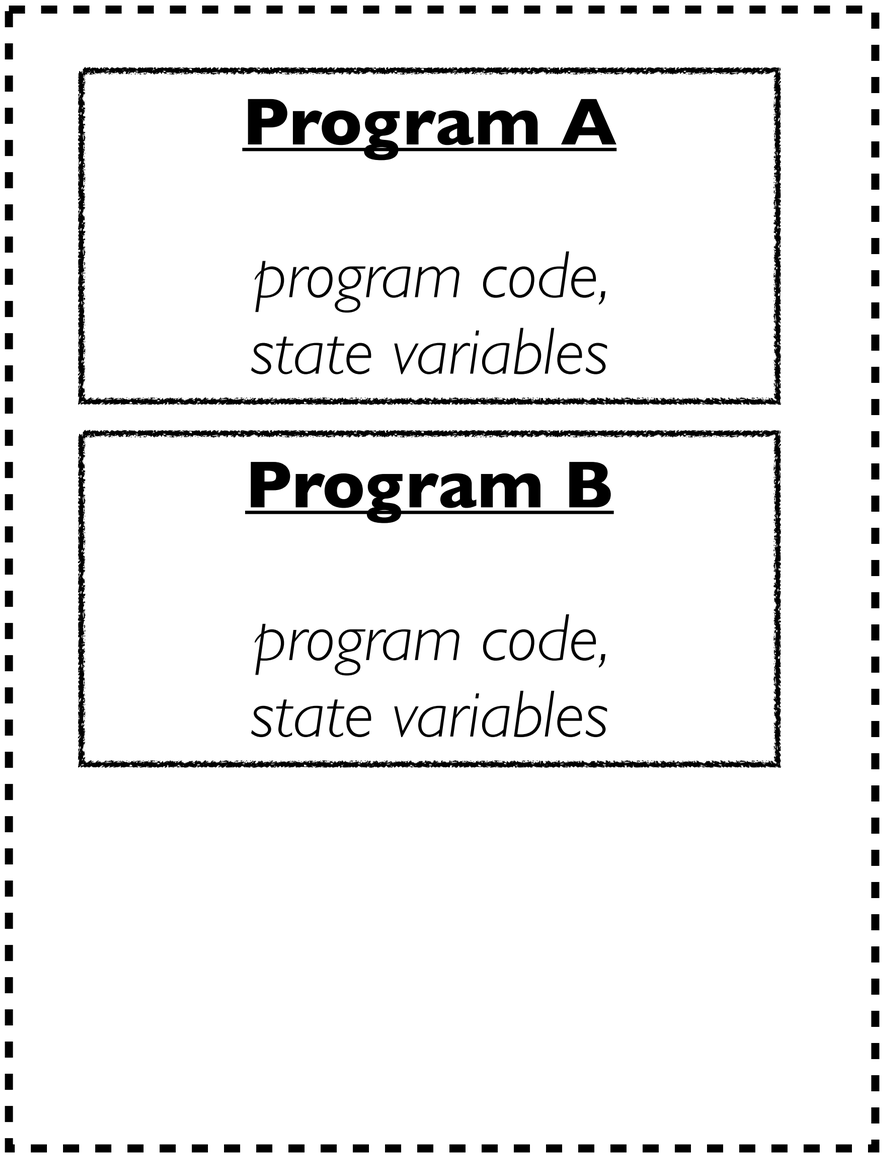
You should have many questions

- Where do these programs come from?
  - E.g., how do I submit a new program to the system
- What if running a program takes too long? Uses too much state?
- What language are these programs written in?
- Where does all the program state get stored?
- If this is a Bitcoin-like system, how do many computers stay in consensus?

Where do the programs come from?

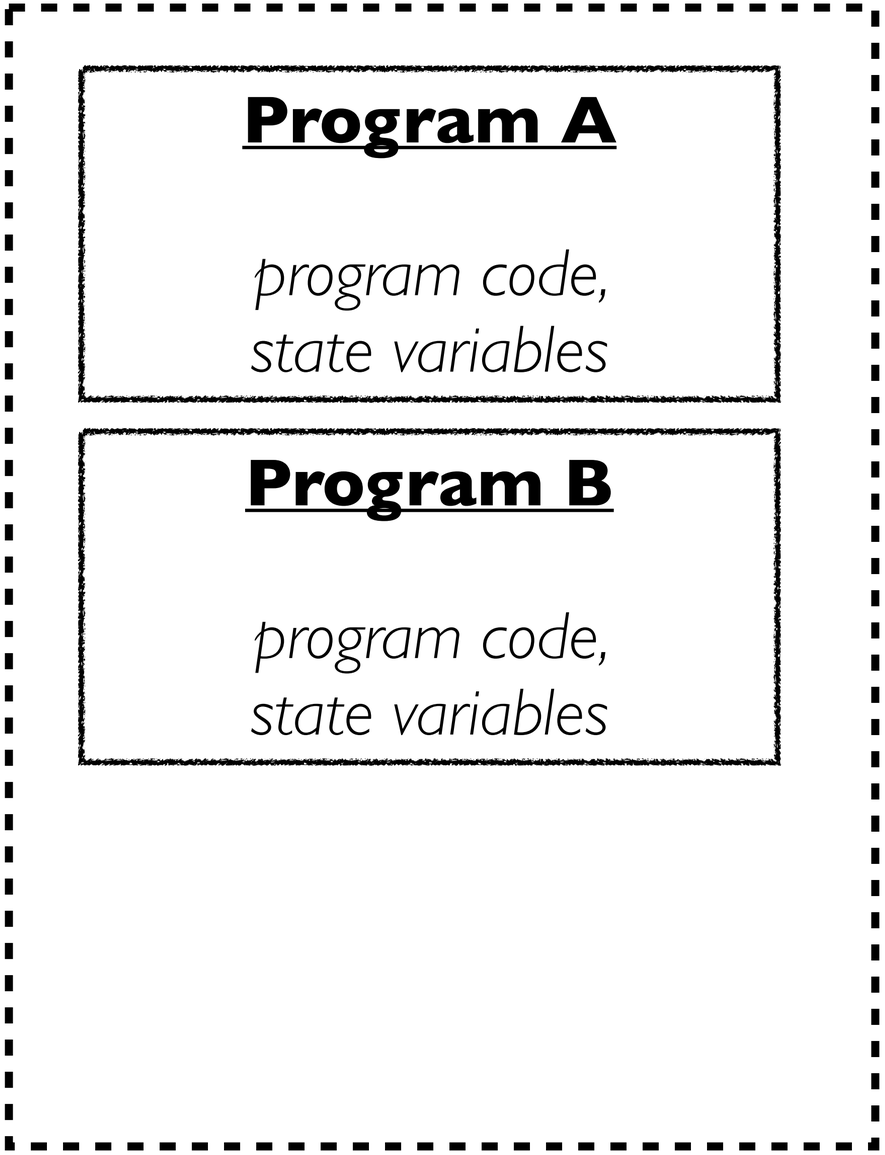
- The network maintains a database
- The database contains one "record" for each program, containing code for its stateUpdate function and all variables/data
- Users can run any program by sending an "execute" transaction (just renaming what we already saw)

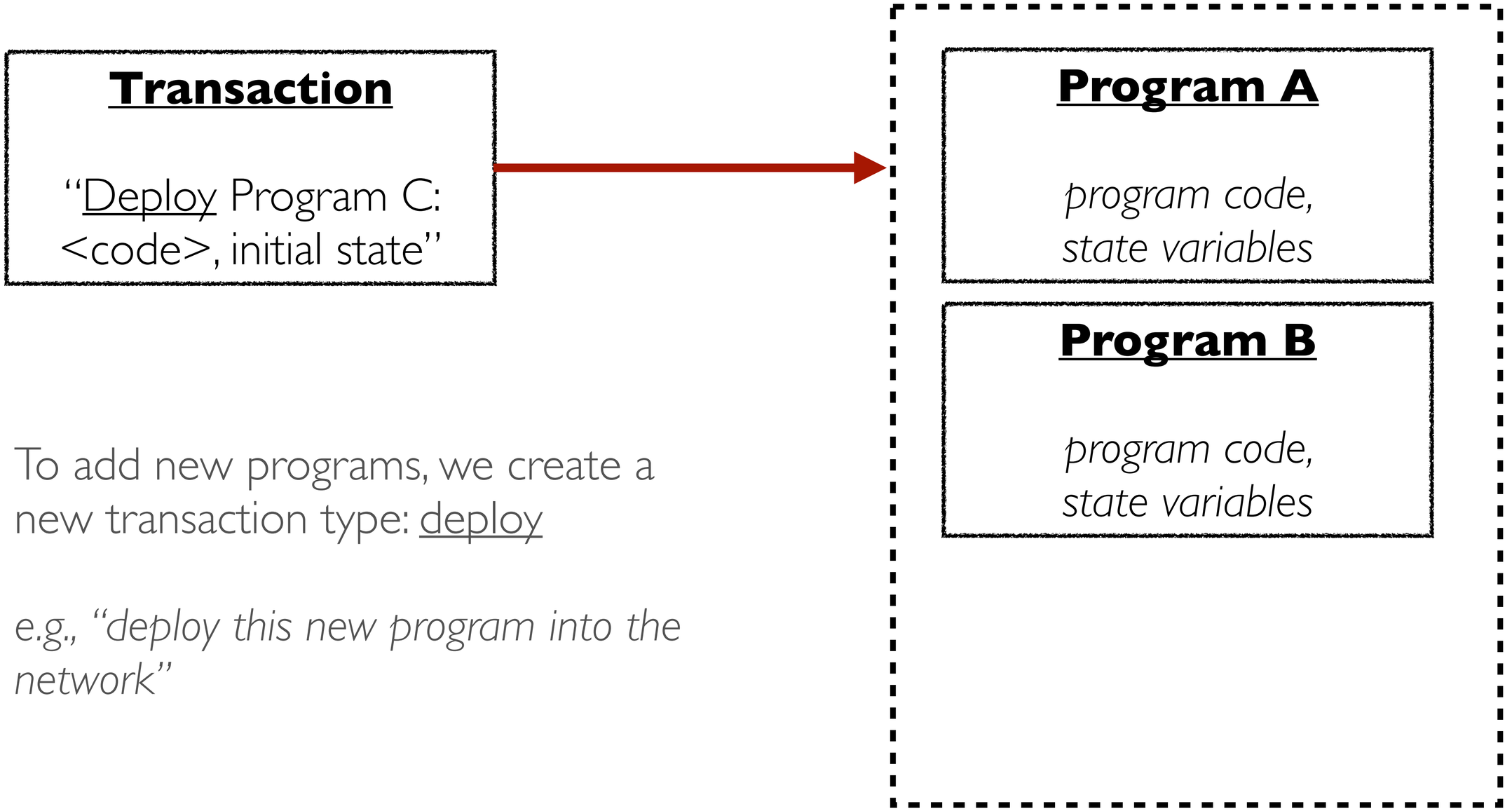
  e.g., "Execute Program B on this input"



- To add new programs, we create a new transaction type: deploy

  e.g., "deploy this new program into the network"

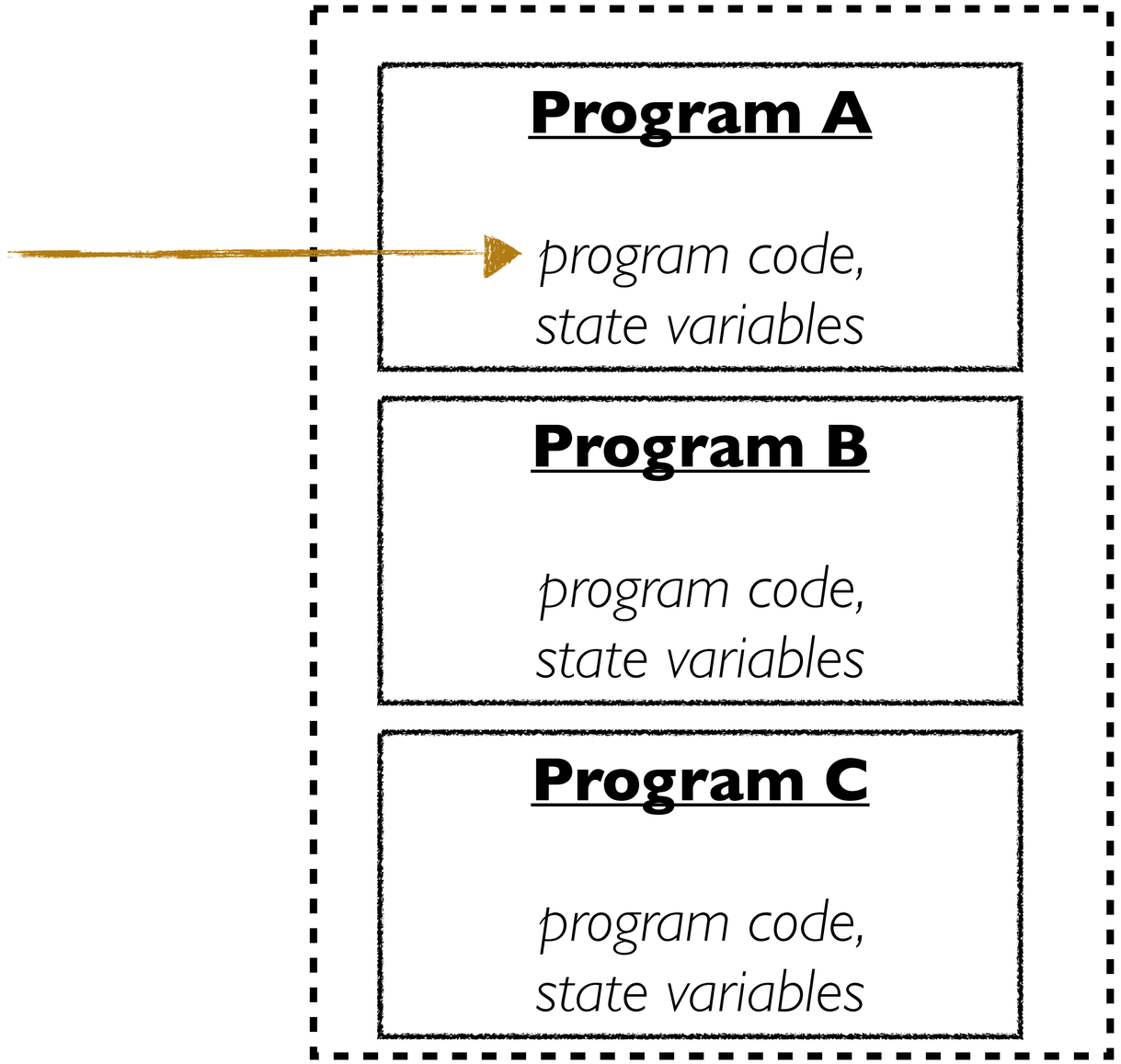




- To add new programs, we create a new transaction type: deploy

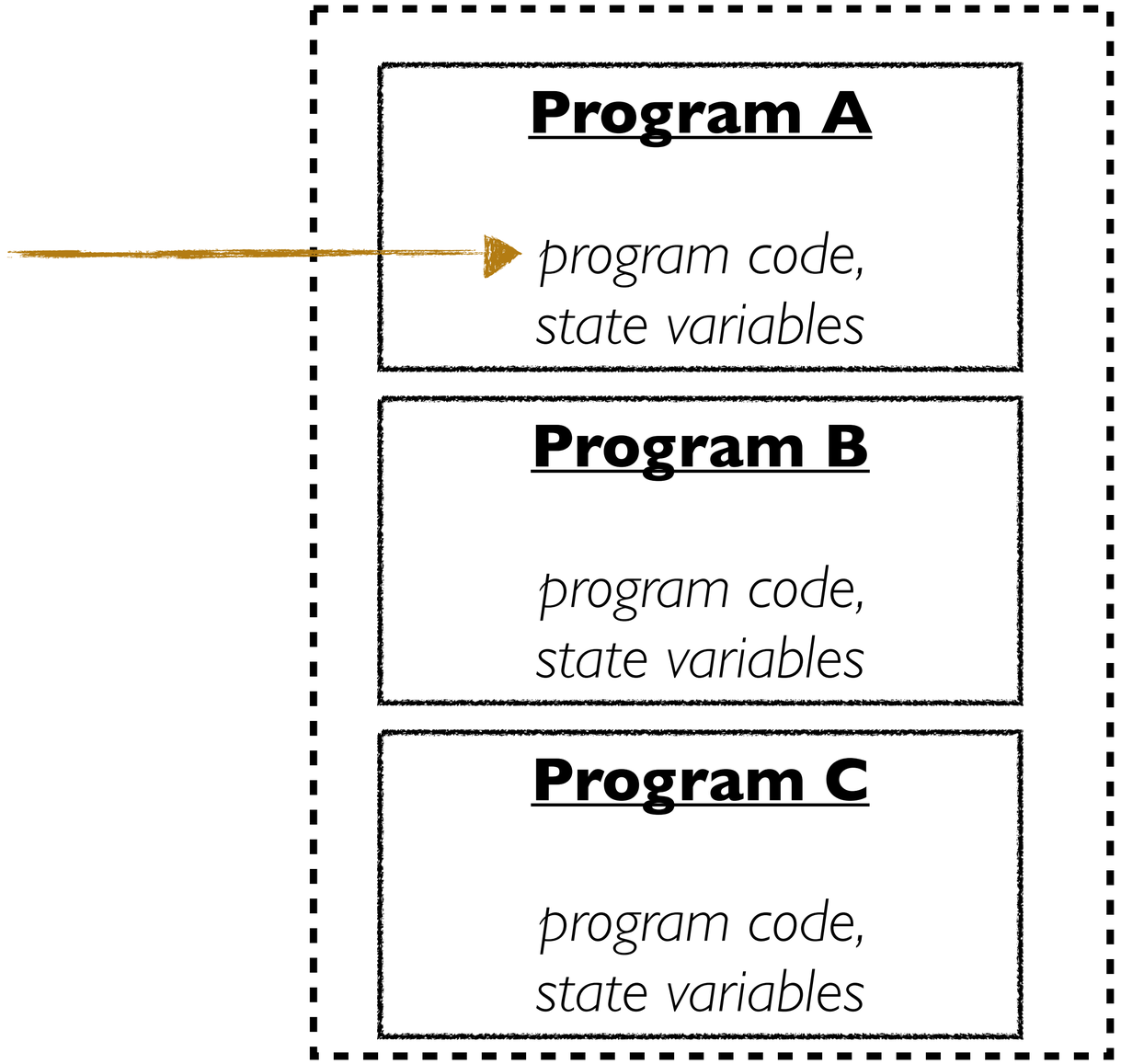
  e.g., "deploy this new program into the network"

Q: What should this code be written in?



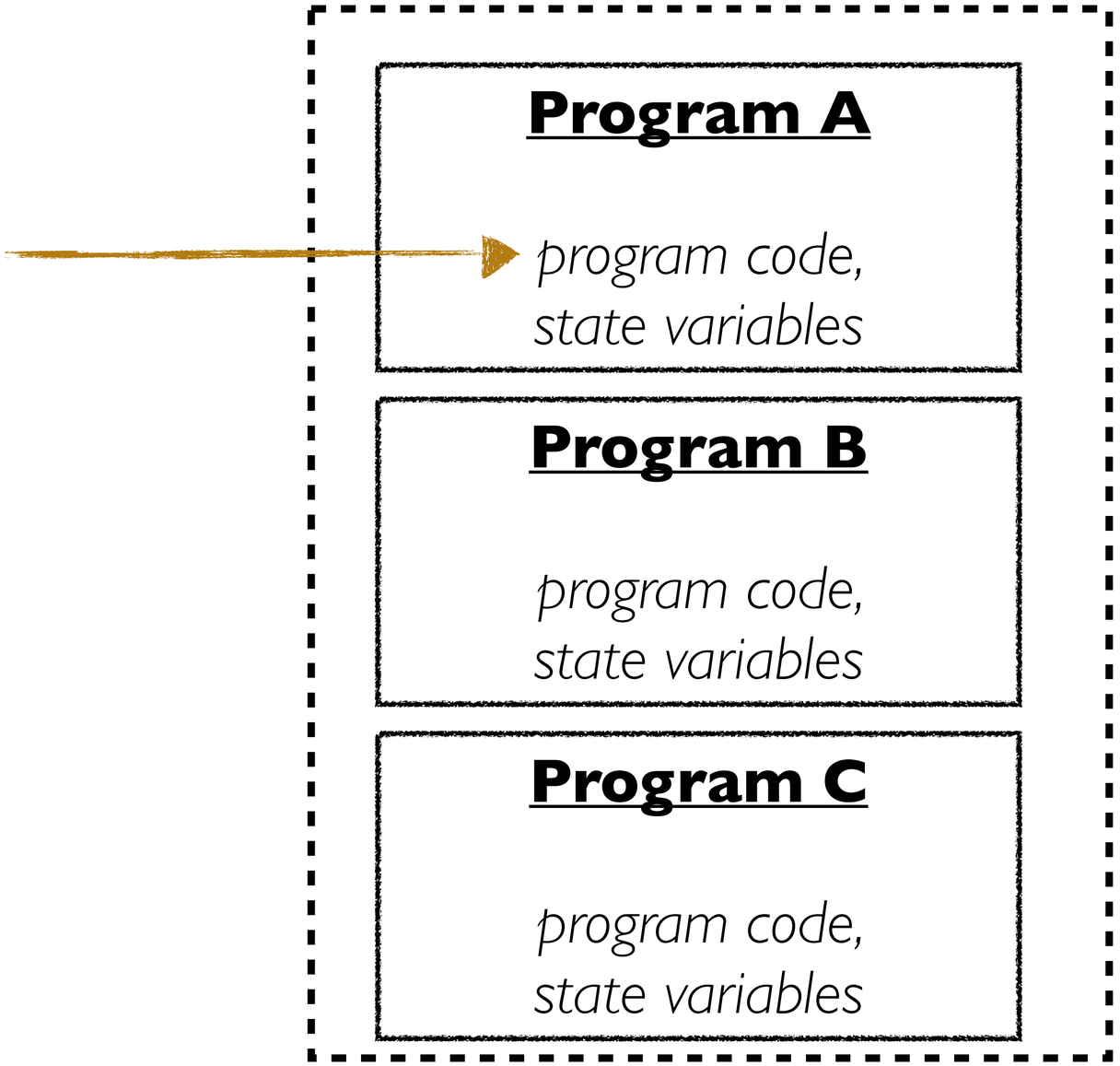
Q: What should this code be written in?

A: Javascript



Q: What should this code be written in?

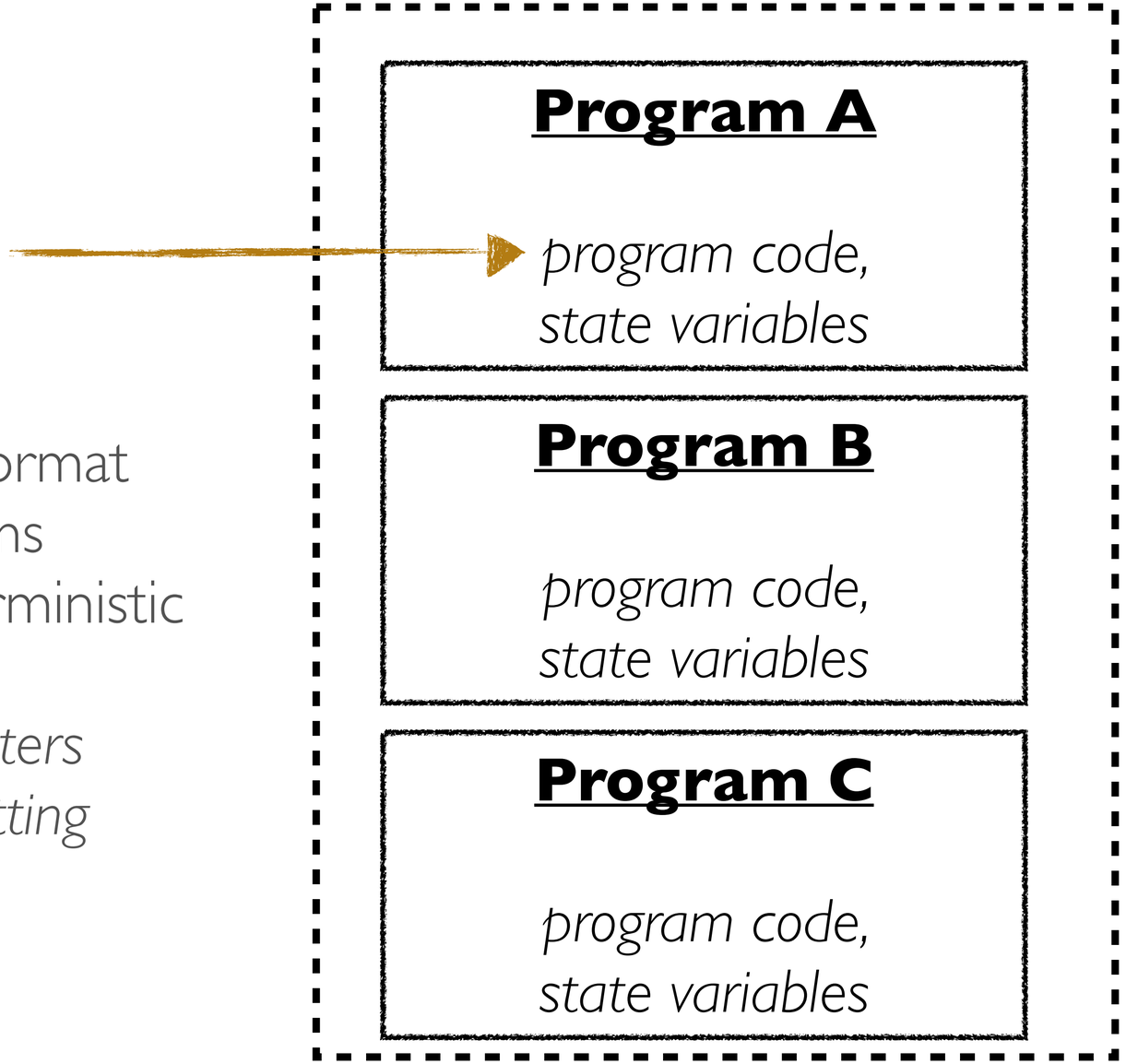
A: Javascript



Q: What should this code be written in?

A: Some kind of portable code format that runs across many platforms and is well-specified and deterministic

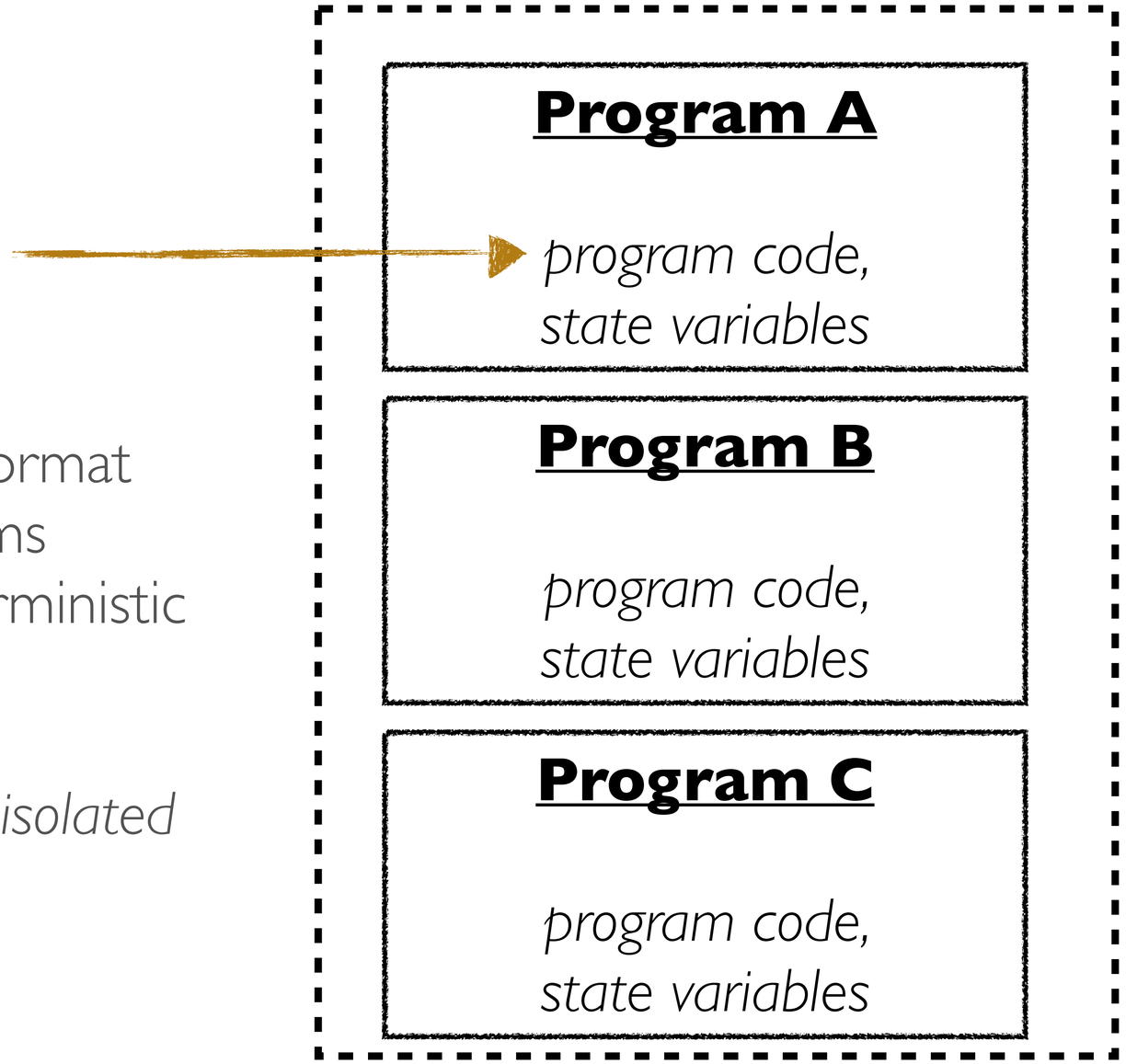
(We can't have different computers running the same code and getting different results!)



Q: What should this code be written in?

A: Some kind of portable code format that runs across many platforms and is well-specified and deterministic

(Also should be memory-safe and easily contained within an isolated virtual machine or sandbox.)



What if running a program takes too long?

What if a program "fills up" the database with junk?