Lecture #8: UTXOs and Accounts

COMS 4995-001: The Science of Blockchains

URL: https://timroughgarden.org/s25/

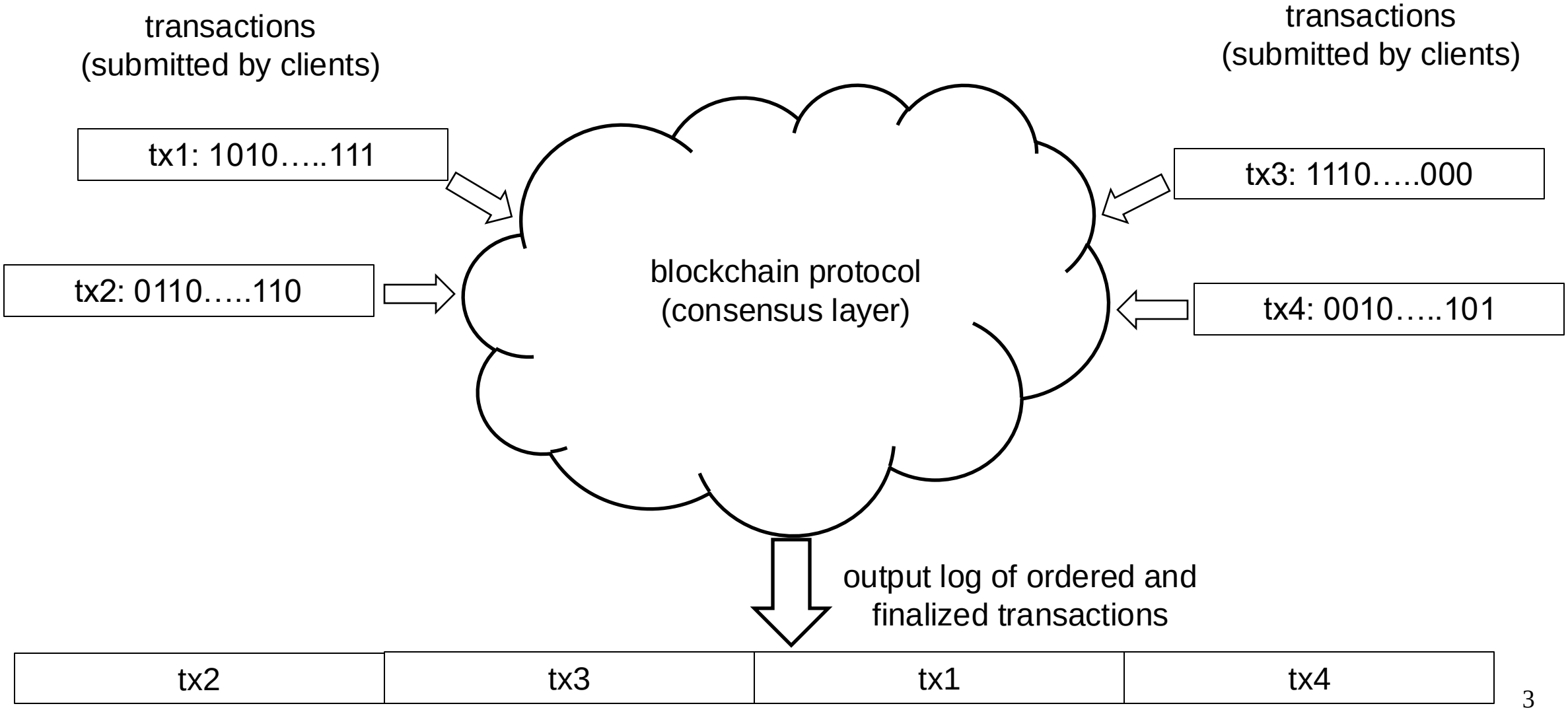
Tim Roughgarden

Responsibilities of a Blockchain Protocol

Consensus: decide on a sequence (aka “chain”) of blocks.

- note: all validators must agree on this sequence!
- blocks keeping getting added (one-by-one) as long as there are transactions to process
- SMR, Tendermint vs. longest-chain consensus, etc.

The Consensus Layer



Responsibilities of a Blockchain Protocol

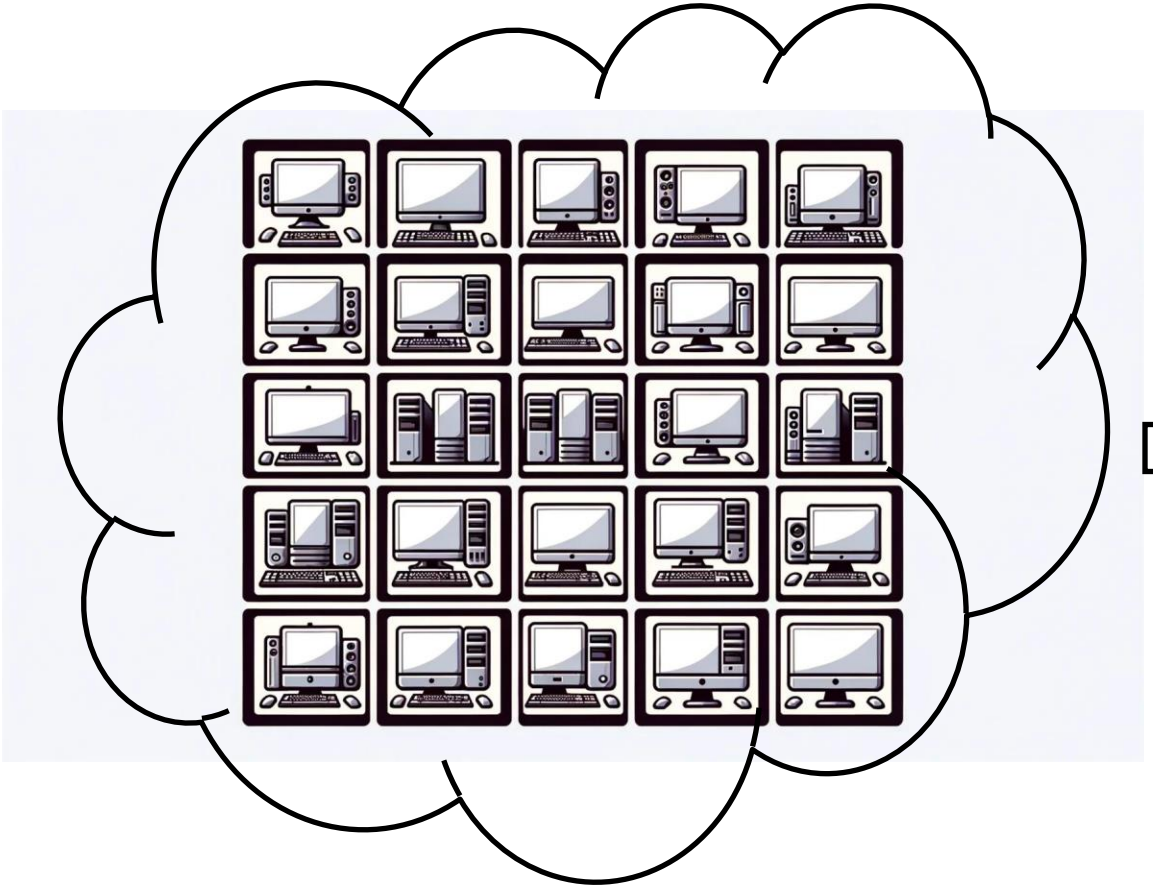
Consensus: decide on a sequence (aka “chain”) of blocks.

- note: all validators must agree on this sequence!
- blocks keeping getting added (one-by-one) as long as there are transactions to process
- SMR, Tendermint vs. longest-chain consensus, etc.

Execution: keep state of the virtual machine up-to-date.

- new block added ➔ execute the corresponding snippets of code (do computations, update variable values, etc.)
- subject of this week (concludes Part I of course)

The Computer in the Sky



network of physical computers
+ blockchain protocol

simulated (virtual) computer

The Execution Layer

tx2	tx3	tx1	tx4

consensus transaction sequence

The Execution Layer

tx2	tx3	tx1	tx4

consensus transaction sequence

blockchain protocol
(execution layer)
[replicated at each
physical machine]

The Execution Layer

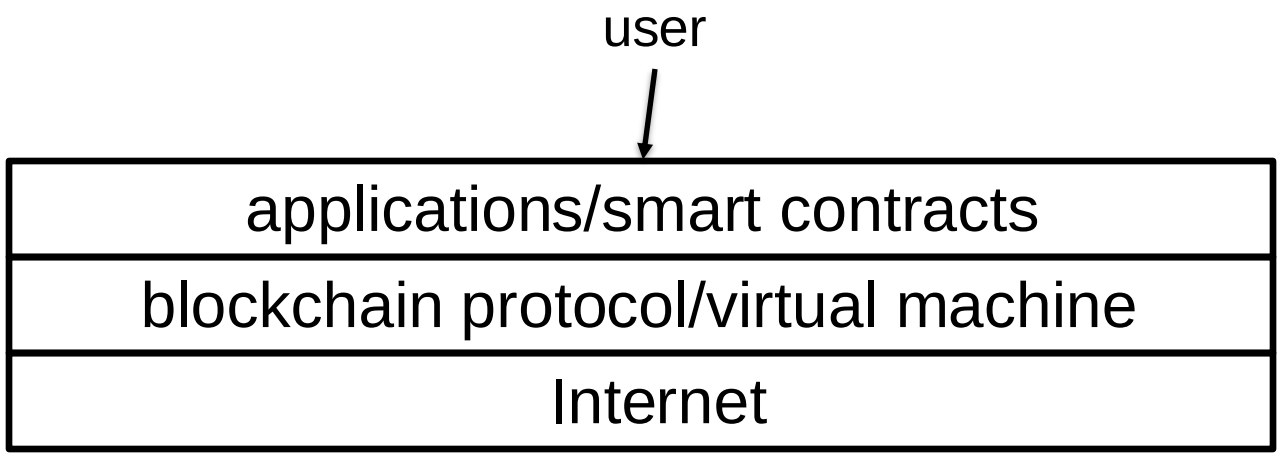
tx2	tx3	tx1	tx4

consensus transaction sequence

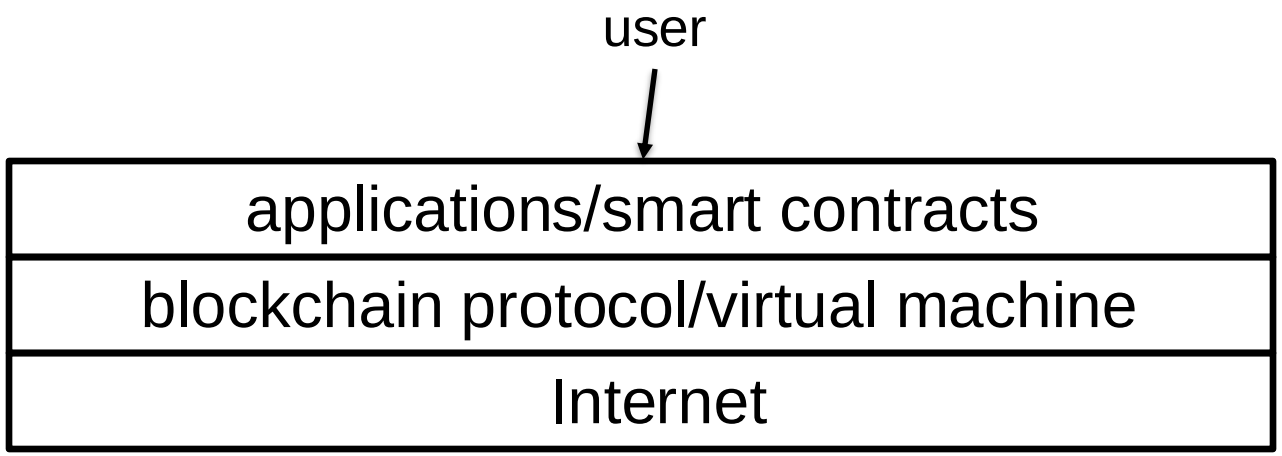
blockchain protocol
(execution layer)
[replicated at each
physical machine]

simulated (virtual) computer
[replicated at each physical
machine]

Recap: A Cartoon of Web3



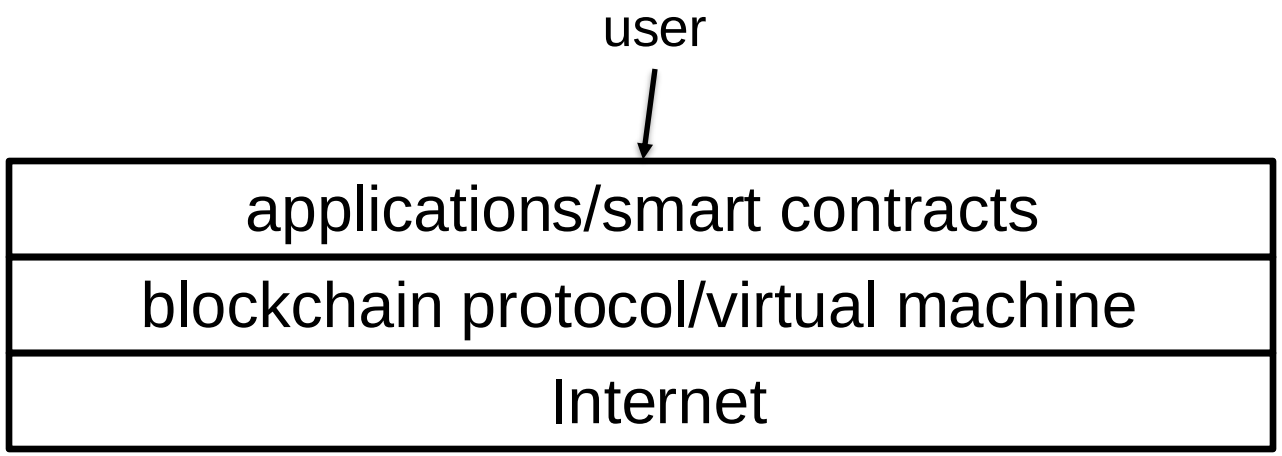
Recap: A Cartoon of Web3



Blockchain protocol:

- like an operating system, a blockchain protocol:
  - acts as a “master program” to coordinate all apps/smart contracts
  - provides a virtual machine to developers of applications

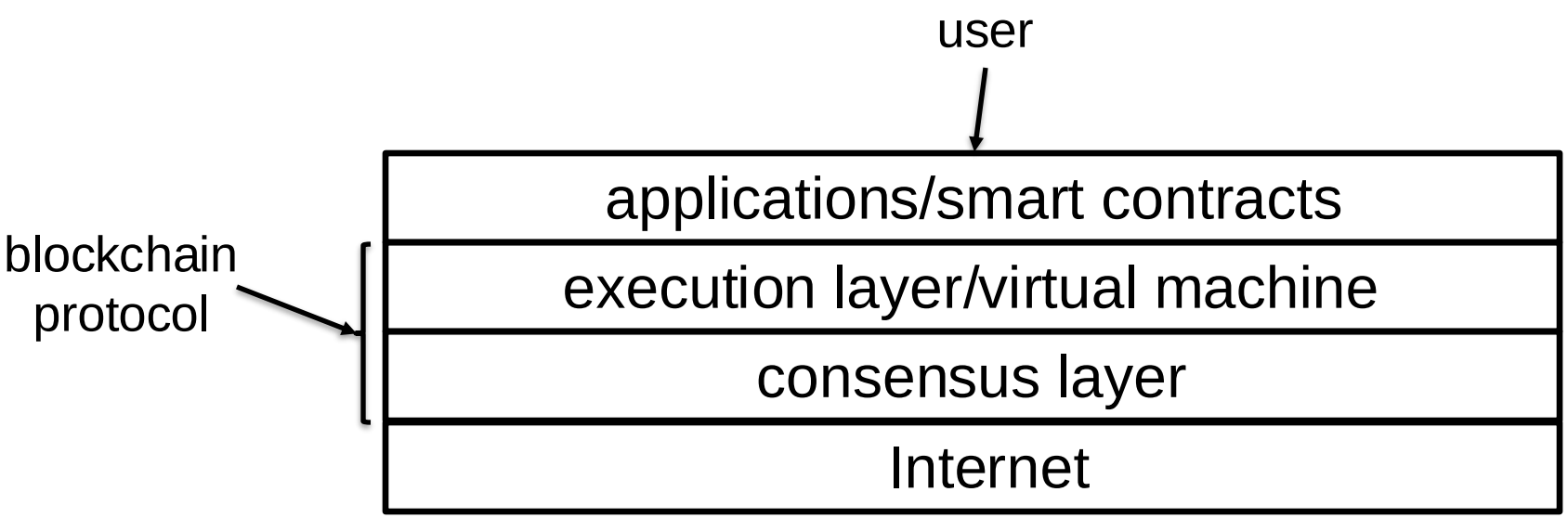
Recap: A Cartoon of Web3



Blockchain protocol:

- like an operating system, a blockchain protocol:
  - acts as a “master program” to coordinate all apps/smart contracts
  - provides a virtual machine to developers of applications
- like the Internet, “decentralized” -- the product of collaboration between many physical machines, no one owner/operator

A Cartoon of Web3 (Refined)



Blockchain protocol:

- like an operating system, a blockchain protocol:
  - acts as a “master program” to coordinate all apps/smart contracts
  - provides a virtual machine to developers of applications
- “decentralized” like the Internet

Goals for Lecture #8

1. The UTXO model (used, e.g., in Bitcoin).
   - counterintuitive but elegant VM specialized for payments
2. Measuring the size of a transaction.
   - idea: what resources are required (now and forever) by a transaction?
   - in practice, very tricky!
3. The account-based model (used in Ethereum and Solana).
   - explicit notion of account IDs and balances, programs as accounts
4. Metering computation.

How to Think About the Execution Layer

Questions:

How to Think About the Execution Layer

Questions: what are the possible “states” of the virtual machine?

How to Think About the Execution Layer

Questions: what are the possible “states” of the virtual machine?

- how are transactions described (both high-level and low-level)?

How to Think About the Execution Layer

Questions: what are the possible “states” of the virtual machine?

- how are transactions described (both high-level and low-level)?
- what state transition results from executing a transaction?

How to Think About the Execution Layer

Questions: what are the possible “states” of the virtual machine?

- how are transactions described (both high-level and low-level)?
- what state transition results from executing a transaction?
- how does a validator represent state and carry out transitions?

How to Think About the Execution Layer

Questions: what are the possible “states” of the virtual machine?

- how are transactions described (both high-level and low-level)?
- what state transition results from executing a transaction?
- how does a validator represent state and carry out transitions?

Note: will now treat the consensus layer as a “black box,” consider a single validator processing a transaction sequence.

  – separation between consensus and execution varies with protocol

How to Think About the Execution Layer

Questions: what are the possible “states” of the virtual machine?

- how are transactions described (both high-level and low-level)?
- what state transition results from executing a transaction?
- how does a validator represent state and carry out transitions?

Note: will now treat the consensus layer as a “black box,” consider a single validator processing a transaction sequence.

  – separation between consensus and execution varies with protocol

Next: warm-up with deep dive on Bitcoin’s “execution layer.”

Bitcoin Transactions

Mental model for Bitcoin transaction: Alice sends x BTC to Bob.

Bitcoin Transactions

Mental model for Bitcoin transaction: Alice sends x BTC to Bob.

Spec for execution layer (natural guess):

- current state described by key-value store
  - keys = account IDs, values = account balances

Bitcoin Transactions

Mental model for Bitcoin transaction: Alice sends x BTC to Bob.

Spec for execution layer (natural guess):

- current state described by key-value store
  - keys = account IDs, values = account balances
- transaction described by (sender ID, recipient ID, amount)
  - validity conditions: both IDs exist, sender balance ≥ amount

Bitcoin Transactions

Mental model for Bitcoin transaction: Alice sends x BTC to Bob.

Spec for execution layer (natural guess):

- current state described by key-value store
  - keys = account IDs, values = account balances
- transaction described by (sender ID, recipient ID, amount)
  - validity conditions: both IDs exist, sender balance ≥ amount
- executing transaction updates balances of sender, recipient

Bitcoin Transactions

Mental model for Bitcoin transaction: Alice sends x BTC to Bob.

~~Spec for execution layer (natural guess):~~

- ~~current state described by key-value store~~
  - ~~keys = account IDs, values = account balances~~
- ~~transaction described by (sender ID, recipient ID, amount)~~
  - ~~validity conditions: both IDs exist, sender balance ≥ amount~~
- ~~executing transaction updates balances of sender, recipient~~

Bitcoin Transactions

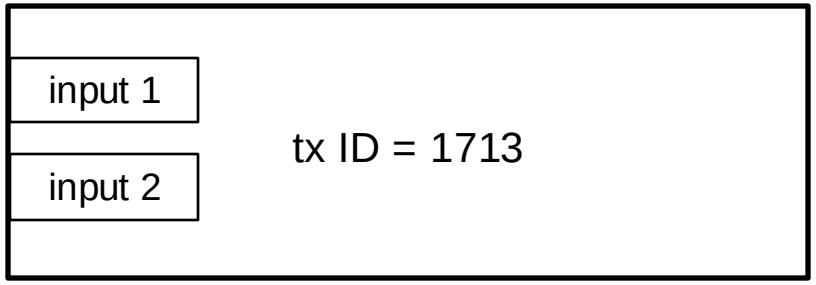
Bitcoin transaction: described by:

tx ID = 1713

Bitcoin Transactions

Bitcoin transaction: described by:

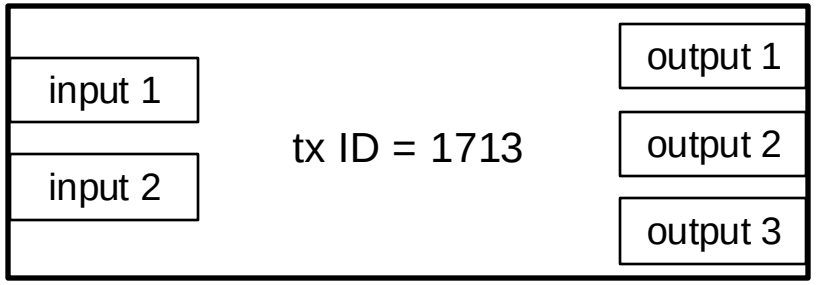
- one or more inputs



Bitcoin Transactions

Bitcoin transaction: described by:

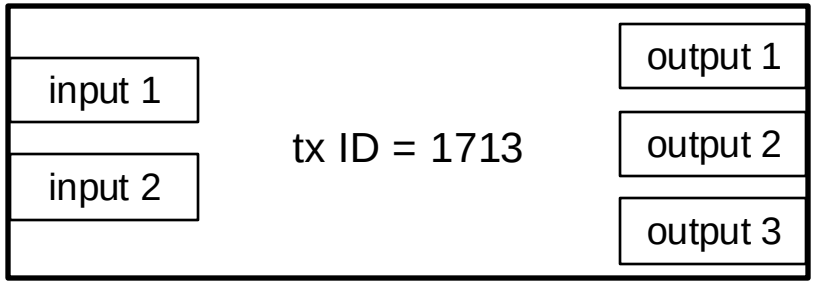
- one or more inputs
- one or more outputs



Bitcoin Transactions

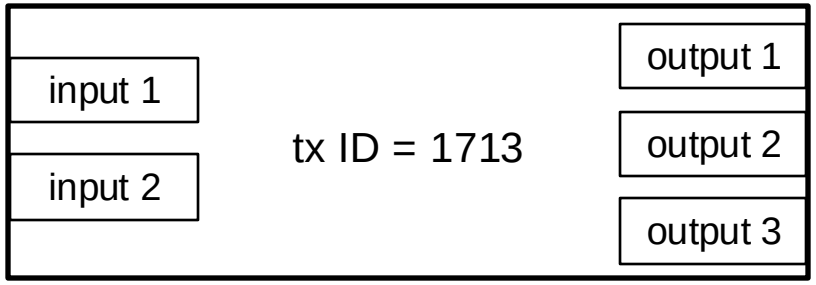
Bitcoin transaction: described by:

- one or more inputs
- one or more outputs
- format for one output
  - value (in BTC)
  - spending conditions [typically ≈ a public key, but can be more complex]



Bitcoin Transactions

Bitcoin transaction: described by:

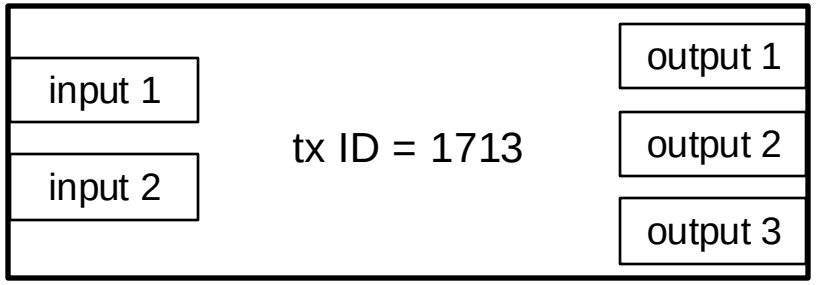


- one or more inputs
- one or more outputs
- format for one output
  - value (in BTC)
  - spending conditions [typically ≈ a public key, but can be more complex]
- format for one input
  - output of some other tx [should be unspent! ➔ “UTXO”]
  - “witness” satisfying output’s spending condition [typically, ≈ a signature]

Bitcoin Transactions

Bitcoin transaction: described by:

- outputs = (value, spending conditions)
- inputs = (UTXO, witness)

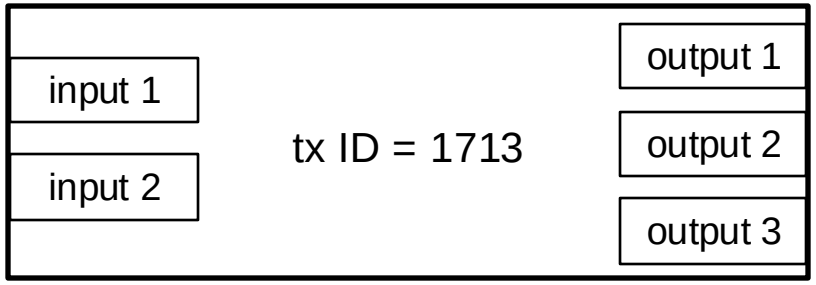


Conditions for validity:

Bitcoin Transactions

Bitcoin transaction: described by:

- outputs = (value, spending conditions)
- inputs = (UTXO, witness)



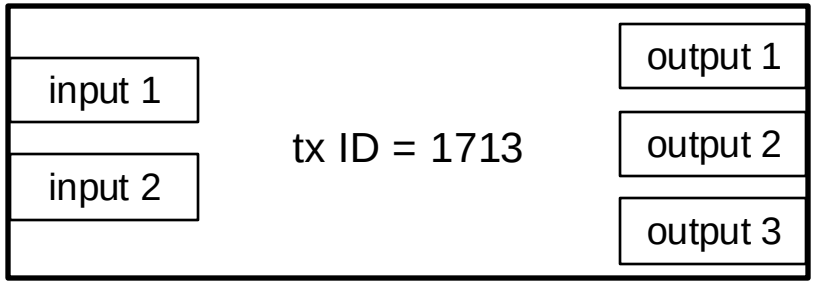
Conditions for validity:

- all inputs reference current UTXOs

Bitcoin Transactions

Bitcoin transaction: described by:

- outputs = (value, spending conditions)
- inputs = (UTXO, witness)



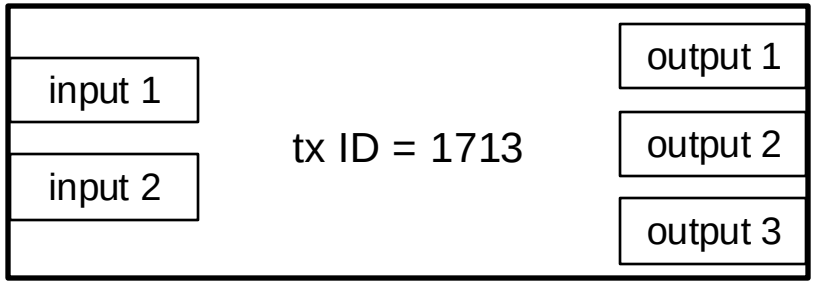
Conditions for validity:

- all inputs reference current UTXOs
- sum of input values ≥ sum of output values [difference = tx fee]
  - common that one of the outputs is a “change address”

Bitcoin Transactions

Bitcoin transaction: described by:

- outputs = (value, spending conditions)
- inputs = (UTXO, witness)



Conditions for validity:

- all inputs reference current UTXOs
- sum of input values ≥ sum of output values [difference = tx fee]
  - common that one of the outputs is a “change address”
- for each input, witness satisfies UTXO’s spending conditions

UTXOs as an Execution Layer

"State" of the Bitcoin protocol: current set of UTXOs.

- no explicit notion of accounts, user IDs, or balances

UTXOs as an Execution Layer

"State" of the Bitcoin protocol: current set of UTXOs.

- no explicit notion of accounts, user IDs, or balances

"Executing" a Bitcoin transaction: [i.e., state transition]

- remove inputs of transaction from the UTXO set
- add outputs of transaction to the UTXO set

UTXOs as an Execution Layer

"State" of the Bitcoin protocol: current set of UTXOs.

- no explicit notion of accounts, user IDs, or balances

"Executing" a Bitcoin transaction: [i.e., state transition]

- remove inputs of transaction from the UTXO set
- add outputs of transaction to the UTXO set

Note: protocol does not prescribe specific representation of state or implementation of state transitions (e.g., checking tx validity).

UTXOs as an Execution Layer

"State" of the Bitcoin protocol: current set of UTXOs.

- no explicit notion of accounts, user IDs, or balances

"Executing" a Bitcoin transaction: [i.e., state transition]

- remove inputs of transaction from the UTXO set
- add outputs of transaction to the UTXO set

Note: protocol does not prescribe specific representation of state or implementation of state transitions (e.g., checking tx validity).

- canonical implementation = "Bitcoin core"

Transaction Size

Fact: some transactions more “complex” than others.

Transaction Size

Fact: some transactions more “complex” than others.

- ideally, quantify heterogeneity via the “size” of a transaction
  - required to define a “maximum block size”
  - sensible to charge fees on a “per-unit-size” (rather than “per-tx”) basis

Transaction Size

Fact: some transactions more “complex” than others.

- ideally, quantify heterogeneity via the “size” of a transaction
  - required to define a “maximum block size”
  - sensible to charge fees on a “per-unit-size” (rather than “per-tx”) basis

Idea: “tx size” ≈ amount of resources required to process it.

Transaction Size

Fact: some transactions more “complex” than others.

- ideally, quantify heterogeneity via the “size” of a transaction
  - required to define a “maximum block size”
  - sensible to charge fees on a “per-unit-size” (rather than “per-tx”) basis

Idea: “tx size” ≈ amount of resources required to process it.

Challenge: multiple types of resources required:

- resources at the consensus layer (bandwidth)
- resources at the execution layer (computation, memory access)
- resources for long-term storage (at validators or elsewhere)

Transaction Size

Fact: some transactions more “complex” than others.

- ideally, quantify heterogeneity via the “size” of a transaction

Idea: “tx size” ≈ amount of resources required to process it.

Challenge: multiple types of resources required.

Further challenge: resource consumption may depend on external-to-protocol factors (e.g., specific client implementation and/or validator architecture).

  – in practice, “size” often defined w.r.t. some canonical implementation

Transaction Size in Bitcoin

Bitcoin (2009-2017): tx size := description length (in bytes).

- typical tx size 250 bytes
- maximum block size = 1 MB ➔ 4000 tx/block (< 7 txs/sec)

Transaction Size in Bitcoin

Bitcoin (2009-2017): tx size := description length (in bytes).

- typical tx size 250 bytes
- maximum block size = 1 MB ➔ 4000 tx/block (< 7 txs/sec)

The blocksize wars (2015-2017): heated debate over whether to increase block size (e.g., to 2MB or 8 MB).

- lowers barrier to participating, but raises barrier to validating
- benefits of innovating vs. benefits of hardening
- led to Bitcoin Cash (fork of Bitcoin with bigger blocks, now irrelevant)

Transaction Size in Bitcoin (con'd)

Bitcoin (2009-2017): tx size := description length (in bytes).

The blocksize wars (2015-2017): heated debate over whether to increase block size (e.g., to 2MB or 8 MB).

Transaction Size in Bitcoin (con'd)

Bitcoin (2009-2017): tx size := description length (in bytes).

The blocksize wars (2015-2017): heated debate over whether to increase block size (e.g., to 2MB or 8 MB).

SegWit (2017): redefined "size" of a transaction to:

.25*(# of bytes used for witness data) + (# of additional bytes used)

– ➔ maximum block size now effectively 4MB (if entirely witness data)

Transaction Size in Bitcoin (con'd)

Bitcoin (2009-2017): tx size := description length (in bytes).

The blocksize wars (2015-2017): heated debate over whether to increase block size (e.g., to 2MB or 8 MB).

SegWit (2017): redefined "size" of a transaction to:

.25*(# of bytes used for witness data) + (# of additional bytes used)

– ➔ maximum block size now effectively 4MB (if entirely witness data)

- idea: validator can discard witnesses after checking tx validity
  – "archival nodes" should still keep witness data for posterity

Transaction Size in Bitcoin (con'd)

SegWit (2017): size := .25*(# of bytes used for witness data) + (# of additional bytes used) ➔ maximum block size now effectively 4MB.

Transaction Size in Bitcoin (con'd)

SegWit (2017): size := .25*(# of bytes used for witness data) + (# of additional bytes used) ➔ maximum block size now effectively 4MB.

Taproot (2021): more general/flexible format for witness data.

Transaction Size in Bitcoin (con'd)

SegWit (2017): size := .25*(# of bytes used for witness data) + (# of additional bytes used) ➔ maximum block size now effectively 4MB.

Taproot (2021): more general/flexible format for witness data.

Ordinals/inscriptions: basically, NFTs (up to 4MB) on Bitcoin!

Transaction Size in Bitcoin (con'd)

SegWit (2017): size := .25*(# of bytes used for witness data) + (# of additional bytes used) ➔ maximum block size now effectively 4MB.

Taproot (2021): more general/flexible format for witness data.

Ordinals/inscriptions: basically, NFTs (up to 4MB) on Bitcoin!

- idea #1: ascribe "serial numbers" to Bitcoins (actually, satoshis) so that they can be viewed as non-fungible rather than fungible

Transaction Size in Bitcoin (con'd)

SegWit (2017): size := .25*(# of bytes used for witness data) + (# of additional bytes used) ➔ maximum block size now effectively 4MB.

Taproot (2021): more general/flexible format for witness data.

Ordinals/inscriptions: basically, NFTs (up to 4MB) on Bitcoin!

- idea #1: ascribe "serial numbers" to Bitcoins (actually, satoshis) so that they can be viewed as non-fungible rather than fungible
- idea #2: embed NFT data (e.g., image) into witness data of a Taproot tx

Transaction Size in Bitcoin (con'd)

SegWit (2017): size := .25*(# of bytes used for witness data) + (# of additional bytes used) ➔ maximum block size now effectively 4MB.

Taproot (2021): more general/flexible format for witness data.

Ordinals/inscriptions: basically, NFTs (up to 4MB) on Bitcoin!

- idea #1: ascribe "serial numbers" to Bitcoins (actually, satoshis) so that they can be viewed as non-fungible rather than fungible
- idea #2: embed NFT data (e.g., image) into witness data of a Taproot tx
- debate: are these good for Bitcoin?

Transaction Size in Bitcoin (con'd)

SegWit (2017): size := .25*(# of bytes used for witness data) + (# of additional bytes used) ➔ maximum block size now effectively 4MB.

Taproot (2021): more general/flexible format for witness data.

Ordinals/inscriptions: basically, NFTs (up to 4MB) on Bitcoin!

- idea #1: ascribe "serial numbers" to Bitcoins (actually, satoshis) so that they can be viewed as non-fungible rather than fungible
- idea #2: embed NFT data (e.g., image) into witness data of a Taproot tx

Point: definition of transaction size can fundamentally affect how a blockchain protocol is used!

Account-Based Execution Layers

Idea: “state” of an account-based protocol specified by:

Account-Based Execution Layers

Idea: “state” of an account-based protocol specified by:

- a set of current accounts (indexed by accountID, e.g. a pk)
  - generally, an account could correspond to a user or a program/contract

Account-Based Execution Layers

Idea: “state” of an account-based protocol specified by:

- a set of current accounts (indexed by accountID, e.g. a pk)
  - generally, an account could correspond to a user or a program/contract
- the state of each of these accounts, e.g.:
  - balance in native cryptocurrency (ETH, SOL, etc.)
  - arbitrary persistent and mutable data
  - VM code (perhaps immutable)

Account-Based Execution Layers

Idea: “state” of an account-based protocol specified by:

- a set of current accounts (indexed by accountID, e.g. a pk)
  - generally, an account could correspond to a user or a program/contract
- the state of each of these accounts, e.g.:
  - balance in native cryptocurrency (ETH, SOL, etc.)
  - arbitrary persistent and mutable data
  - VM code (perhaps immutable)

Example: in Ethereum, a user account (“EOA”) has no code, only data is a “nonce” [= # of txs sent by account, prevents “replay attacks”].

- all other data on user stored in contracts’ accounts

Typical Transaction Ingredients

Transaction: sent by a user (to a user, or a program). Includes:

Typical Transaction Ingredients

Transaction: sent by a user (to a user, or a program). Includes:

- signature by the sender (can back out pk/ID from signature)
- recipient (specified by account ID, user or contract)

Typical Transaction Ingredients

Transaction: sent by a user (to a user, or a program). Includes:

- signature by the sender (can back out pk/ID from signature)
- recipient (specified by account ID, user or contract)
- value (in native currency)
- data (e.g., which function to call and with which arguments)

Typical Transaction Ingredients

Transaction: sent by a user (to a user, or a program). Includes:

- signature by the sender (can back out pk/ID from signature)
- recipient (specified by account ID, user or contract)
- value (in native currency)
- data (e.g., which function to call and with which arguments)
- declaration of resources to be used
- transaction fee

Typical Transaction Ingredients

Transaction: sent by a user (to a user, or a program). Includes:

- signature by the sender (can back out pk/ID from signature)
- recipient (specified by account ID, user or contract)
- value (in native currency)
- data (e.g., which function to call and with which arguments)
- declaration of resources to be used
- transaction fee

Note: if programs can be arbitrary code, corresponding state transition can be extremely complex.

Metering Computation

Question: if programs can be arbitrary code, what about the halting problem? [could a tx force an infinite loop in the VM?]

Metering Computation

Question: if programs can be arbitrary code, what about the halting problem? [could a tx force an infinite loop in the VM?]

Solution: associate a cost with each VM instruction, paid by user.

Metering Computation

Question: if programs can be arbitrary code, what about the halting problem? [could a tx force an infinite loop in the VM?]

Solution: associate a cost with each VM instruction, paid by user.

Example: in Ethereum:

Metering Computation

Question: if programs can be arbitrary code, what about the halting problem? [could a tx force an infinite loop in the VM?]

Solution: associate a cost with each VM instruction, paid by user.

Example: in Ethereum:

- associate an amount of “gas” with each EVM opcode
  - EVM opcodes = instruction set for VM code in Ethereum’s VM
  - add two numbers = 3 units of gas; evaluate SHA-256 = 30 units

Metering Computation

Question: if programs can be arbitrary code, what about the halting problem? [could a tx force an infinite loop in the VM?]

Solution: associate a cost with each VM instruction, paid by user.

Example: in Ethereum:

- associate an amount of “gas” with each EVM opcode
  - EVM opcodes = instruction set for VM code in Ethereum’s VM
  - add two numbers = 3 units of gas; evaluate SHA-256 = 30 units
- user prepays for gas (part of the tx description)
- run out of gas mid-execution ➔ tx aborted and rolled back