SMART CONTRACTS

Kai Mast
CS639/839
Spring 2023

Announcements

- First Miniproject will be out this week
  - Due next Friday

- New homeworks will be out on weekends
  - Due on Tuesdays
  - Should not take more than 15 mins
  - First homework this weekend!

- Midterm will be held the week after spring break
  - Thursday, 3/23 @ 5:45pm
  - Alternate midterm planned for 3/28

- Final is set for 5/10 @ 7:25-9:25pm
  - Let me know if this does not work for you

- I will post a form about the alternate midterm and final soon

TODAY'S AGENDA

1. Recap of last week's content
2. Overview of Ethereum's blockchain
3. A short break
4. Introduction to smart contracts

DECENTRALIZED LEDGERS

Blockchains (or Decentralized Ledgers)

- Stores a set of transactions and their order
- Transactions represent the updates to the state

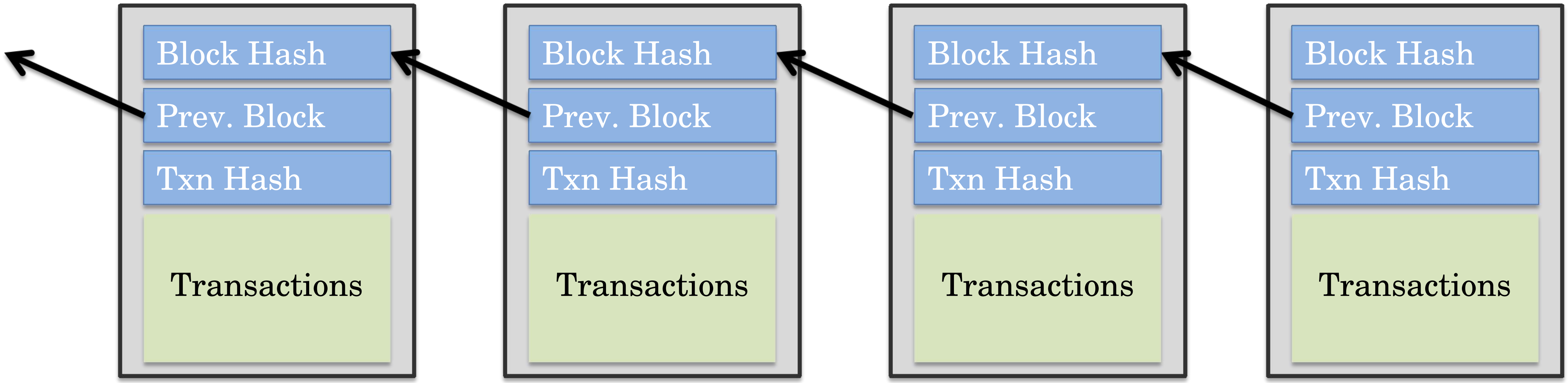
State

- Data that persists after a transaction has finished execution
- E.g., account balances, UTXOs, or smart contract data

Decentralized Ledger Technologies

- Agree on what transaction to aceept and in which order
- Propagate new blocks across the network
- Much more on this later in the semester

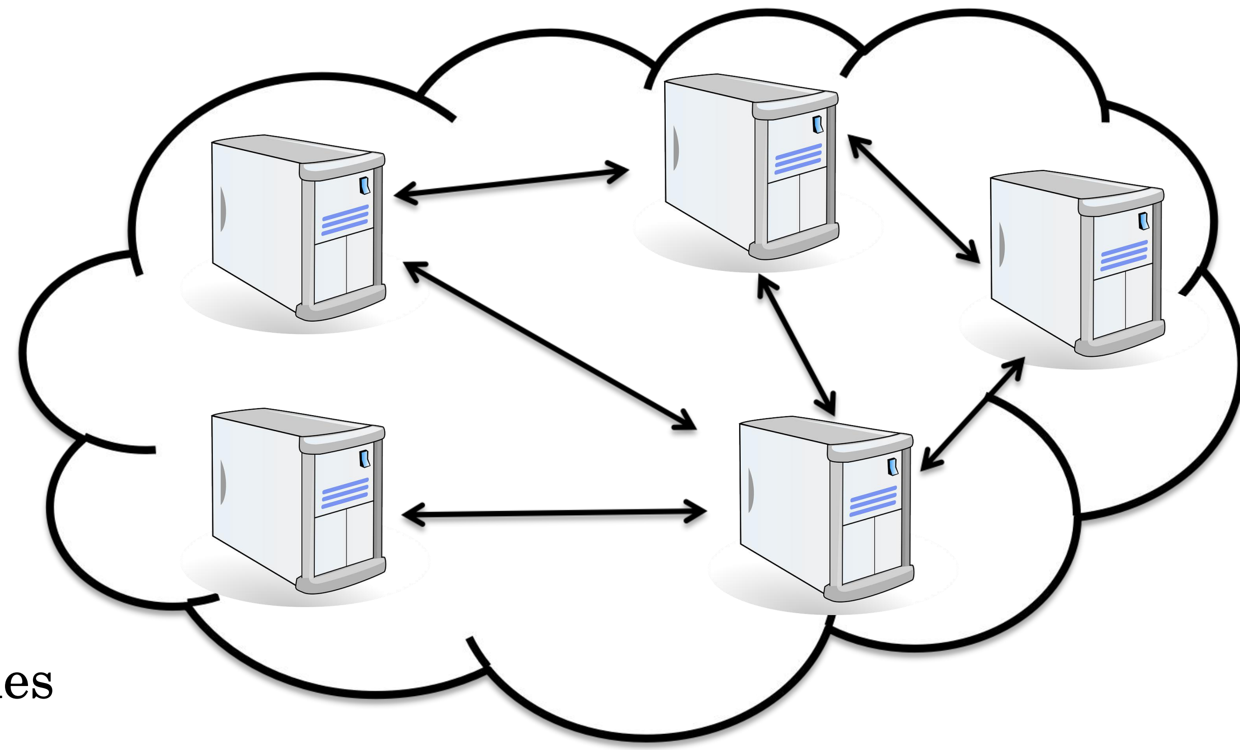
BITCOIN-STYLE LEDGERS



- Ledger contents are stored in a chain of blocks
- Each block contains a (possibly large) number of transactions
- Transaction are ordered using their position within the block and the blocks position within the blockchain

BLOCKCHAIN NETWORKS

Network is public (**permissionless**)
- Not every participant is known
- Anyone can join or leave
- Not everyone is connected to everyone (**peer-to-peer**)



New nodes join by:
- Connecting to a small number of existing nodes
- Fetching and executing all past blocks and transactions

FAILURE MODELS

Blockchains are up here

What we talked about in OS

Byzantine Failures
Any kind of failure can happen

Omission Failures
Messages might get dropped

Crash Failures
Failures might happen "silently"

Fail-Stop
- Failures are immediately detected
- Nodes stop execution as soon as they fail

LEDGER PROPERTIES

Immutability

- No past state can be changed
- Transactions cannot be reordered

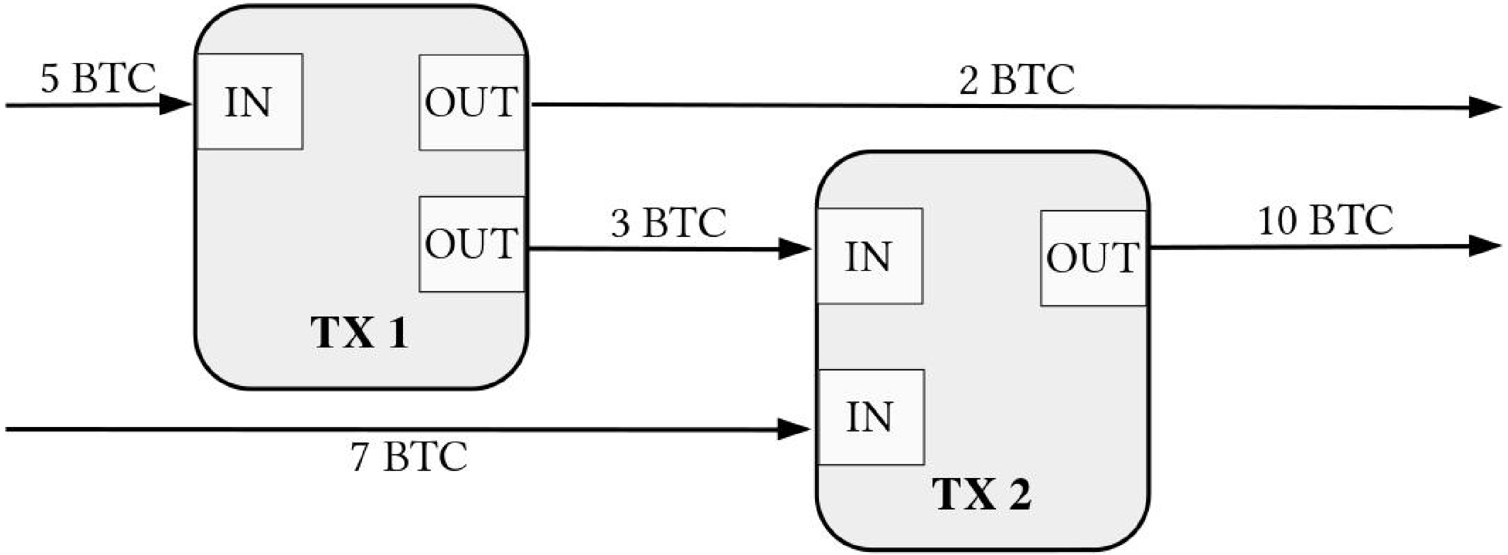
Auditability

- Past transaction can be inspected to replay history

Consistency

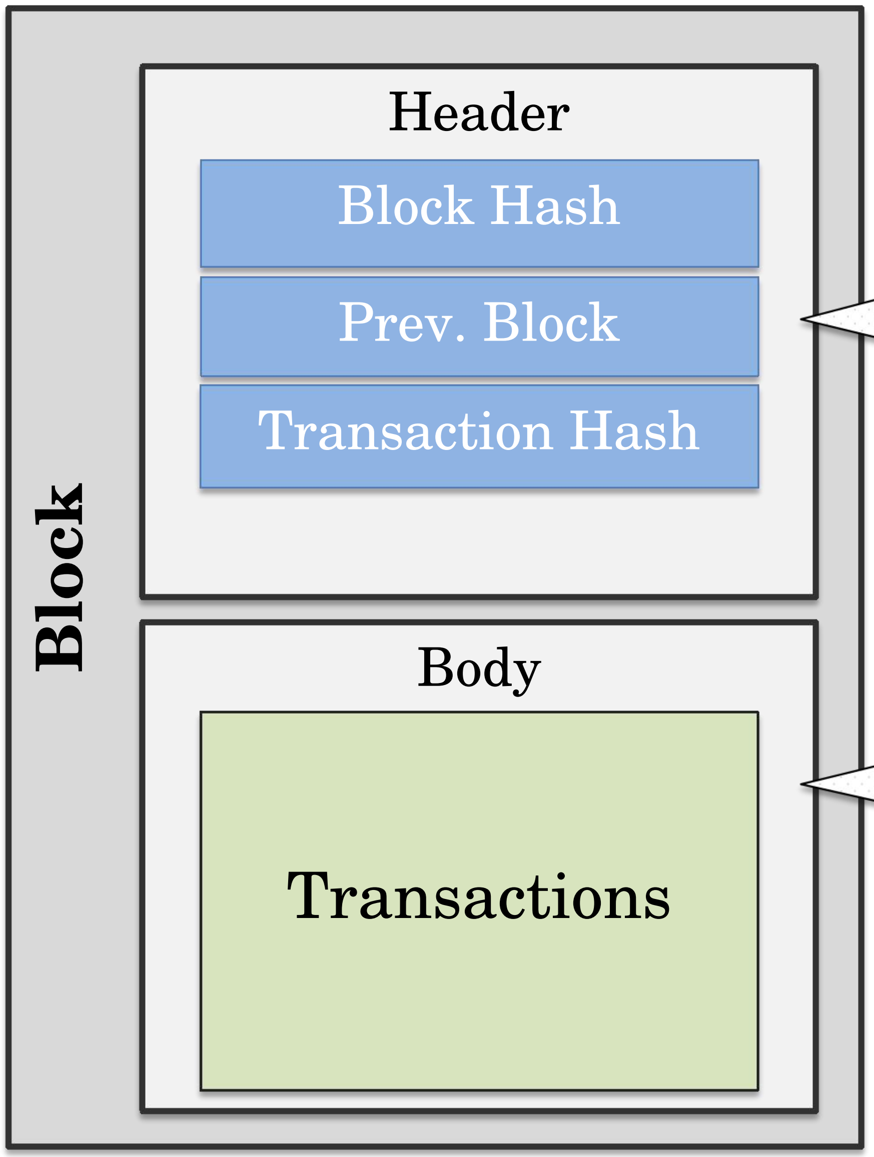
- Application-specific constraints are enforced
- E.g., no double spends are allowed in Bitcoin

UTXO MODEL



- Your account balance is the combined value of all UTXOs you control
- Each transactions consumes at least one UTXO and creates at least one UTXO
- Each UTXO can only be consumed at most once and only in its entirety
- The sum of a transactions inputs must be greater or equal to the sum of its outputs
  - The difference is the *transaction fee*

BLOCK STRUCTURE



Contains **metadata** of the block

- Location of the block within the chain
- Time the block was created and who created it
- Proof-of-Work (if the blockchain uses mining)

Contains **transaction data**

- Each transaction and its required signatures and arguments
- Much bigger than the header

NODE TYPES

Full Nodes
- Hold all data (i.e., all transactions ever accepted to the ledger)
- Can also participate in consensus

Light Nodes
- Only store metadata (block headers)
- Use block headers to verify any data received from full nodes

Why light nodes?

- Blockchains can get large (Bitcoin's is 100s of Gigabytes!)
- Nodes might not have enough compute power to process the entire chain
- Headers are sufficient information for clients

HASH FUNCTIONS IN BLOCKCHAINS

H(Message (Variable length)) = Hash Value (Fixed Size)

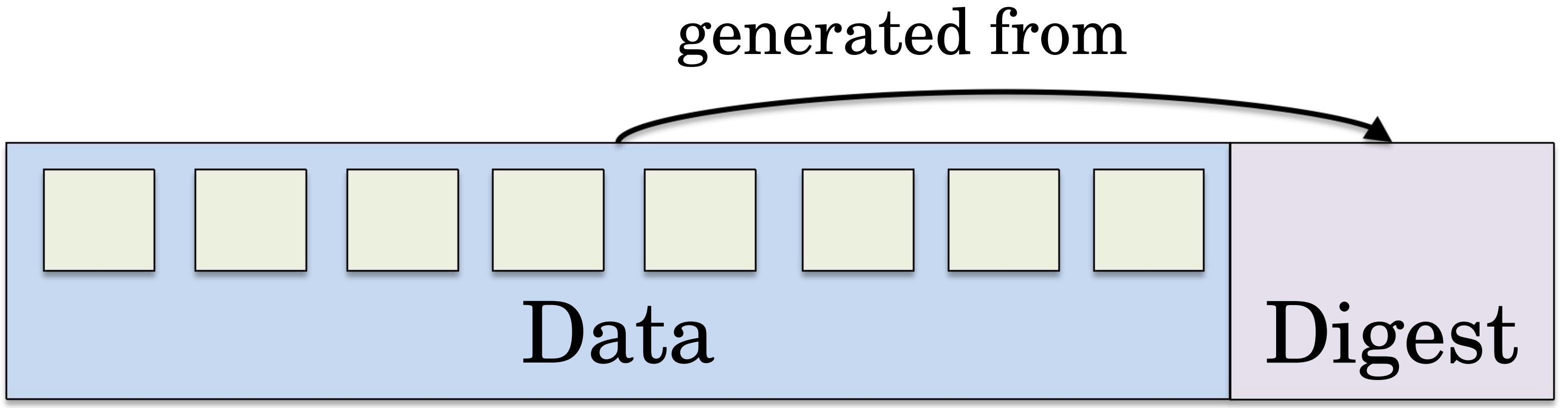
Hash Functions

- Take some input string and generates a fixed size integer value from it
- One way function: No (easy) way to generate the input from hash value

Cryptographic Hash Functions vs. Ordinary Hash Functions

- Hard to find a *collisions*
  - Useful to prevent against attacks
- But, more expensive to compute

AUTHENTICATED DATA STRUCTURES



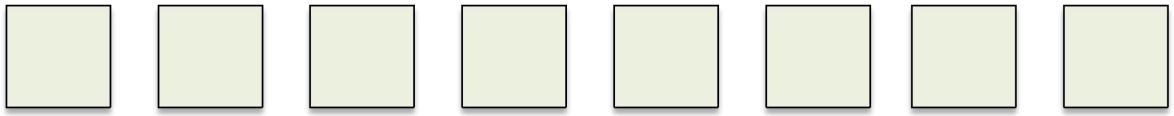
Goal: Provide a way to verify the integrity and authenticity of some data

- Similar (but not identical to) checking integrity of a filesystem/disk
- Data can consists of a large number of items (e.g., all transaction in a block)

Approach:

- Create some additional authentication data (or *digest*) that allows checking for correctness
- To verify, re-generate authentication data and compare

SIMPLISTIC APPROACH

h() = Digest

Compute a single digest from the entire data, e.g. using a hash function

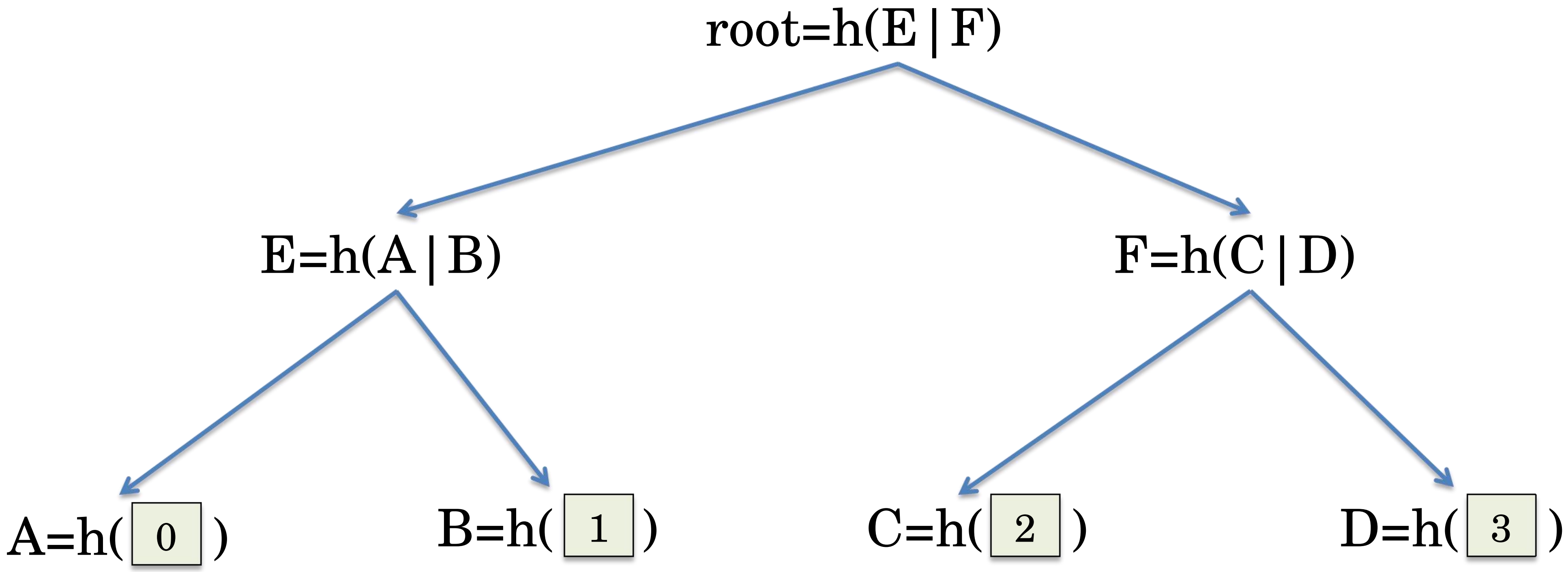
Problems:

- Data can be very big (e.g., the entire state of the blockchain)
- Need to recompute the digest every time any part of the data changes
- To verify any piece of the data we need all of it

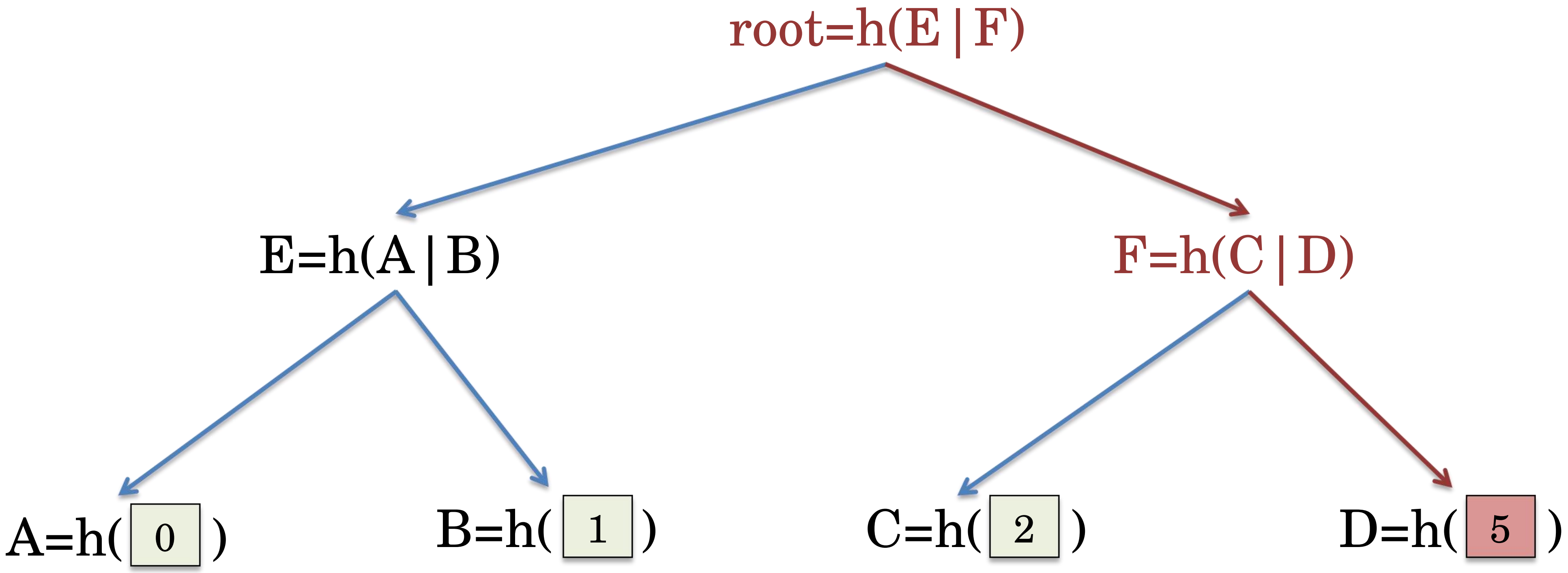
HASH TREES

Idea: Generate a recursive tree structure that recursively hashes data

- Changing data only requires us to recompute the affected branch
- A binary hash tree is also called a *Merkle-tree*

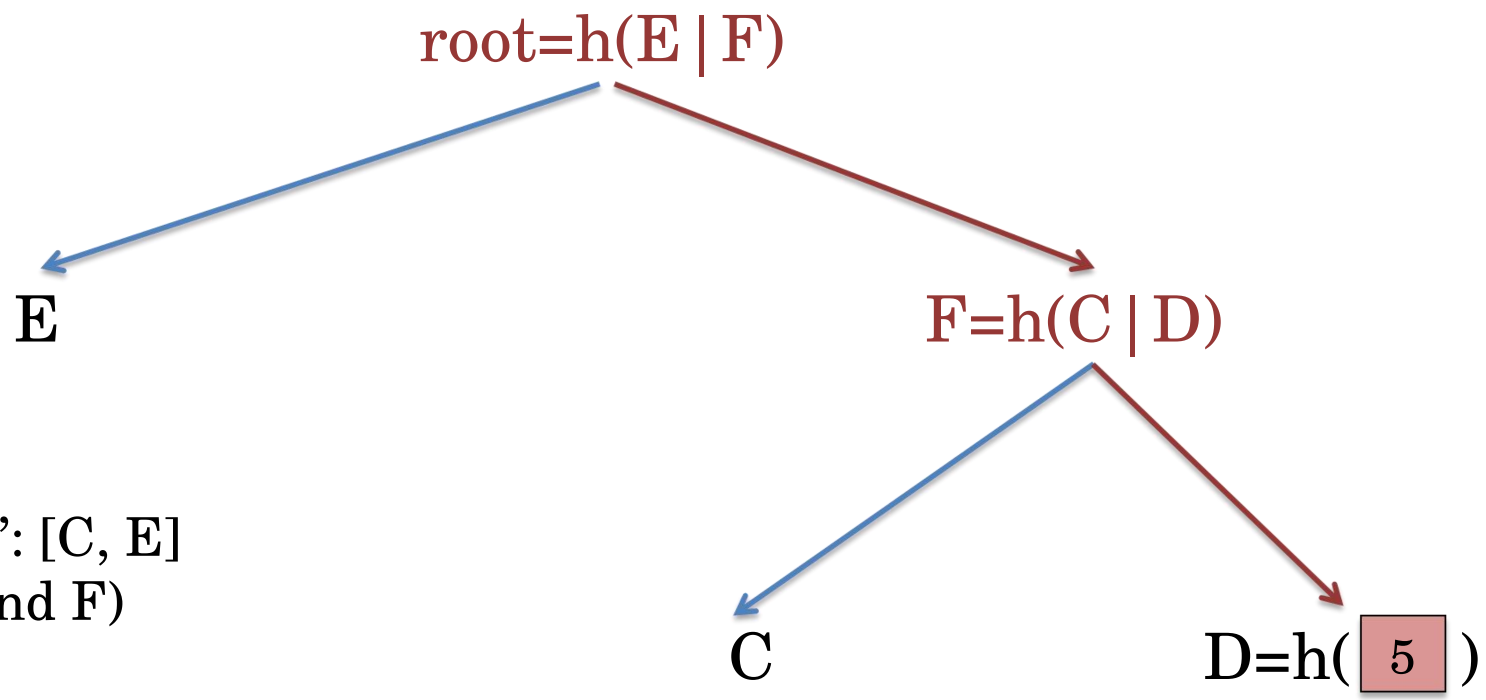


UPDATING A MERKLE TREE



data[“D”] = 5

MERKLE PROOFS

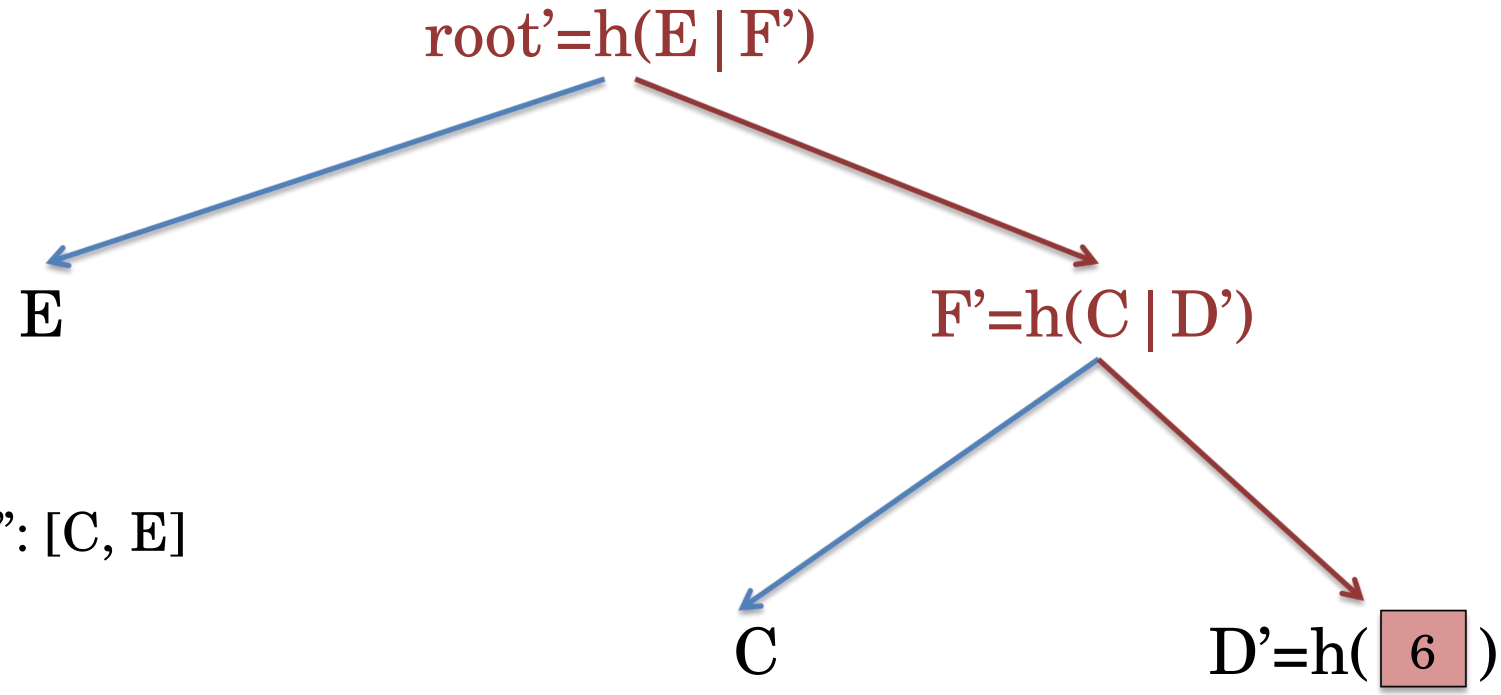


Proof for "D=5 in A" ": [C, E]
(can re-compute D, and F)
(root is stored)

We can verify a single data item by comparing its branch with the root of the tree

- Verifier only needs to have the root stored

INCONSISTENCIES IN MERKLE PROOFS



Proof for “D=5 in A” ”: [C, E]

root’ != root
F’ != F
D’ != D

Computed root’ will differ from stored root
=> verifier will detect inconsistency

PUBLIC/PRIVATE KEY PAIRS

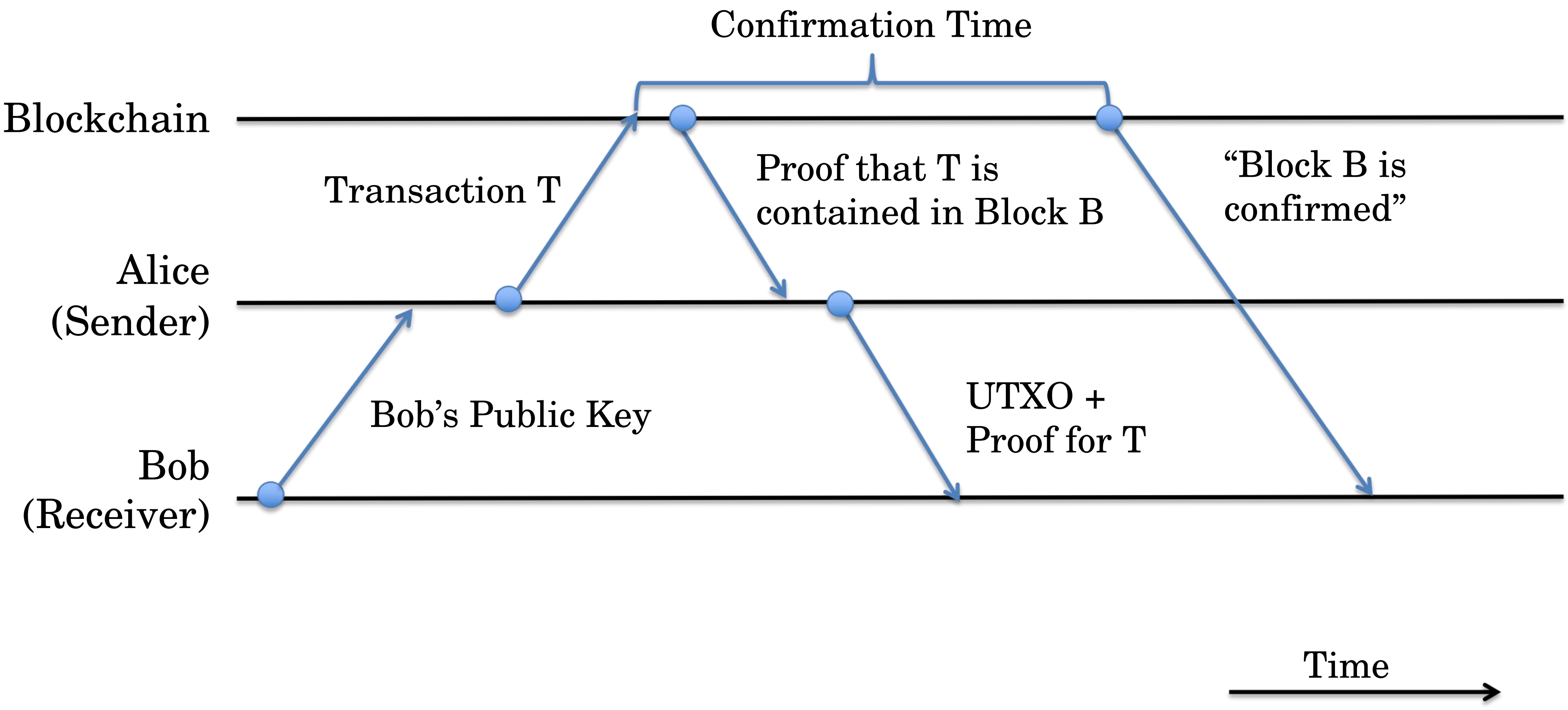
Each pair has one public and one private key

Each type of key has different capabilities
- Also called **asymmetric** cryptography

Anyone with the public key:
- Can verify signatures
- Can encrypt data

Owner of the private key (e.g., a Bitcoin Wallet):
- Can sign data
- Can decrypt data

MONEY TRANSFERS IN THE UTXO MODEL



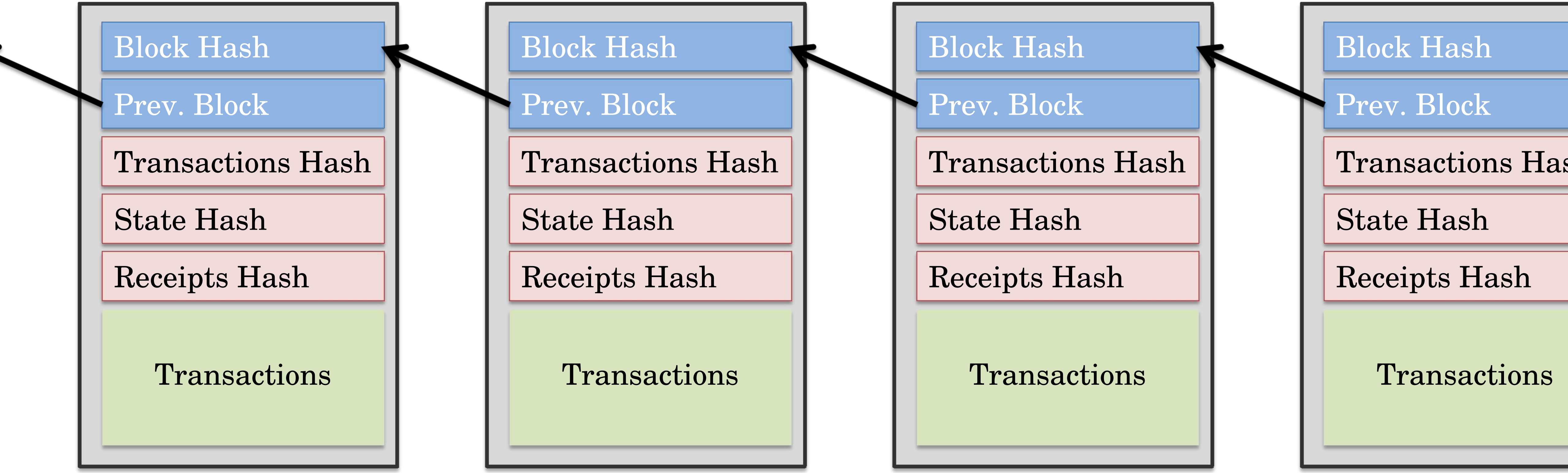
SMART CONTRACTS

So far:
- Execute financial transactions (w/ some scriptability)
- Not enough to build arbitrary applications

Idea:
- Support execution of Turing complete code
- Allow storing state on the blockchain

But how?
- Does not work (easily) with the UTXO model (UTXOs are removed once consumed)
- Bitcoin Script misses many features (no loops, or function calls)
- We need a different data and execution model

THE ETHEREUM BLOCKCHAIN

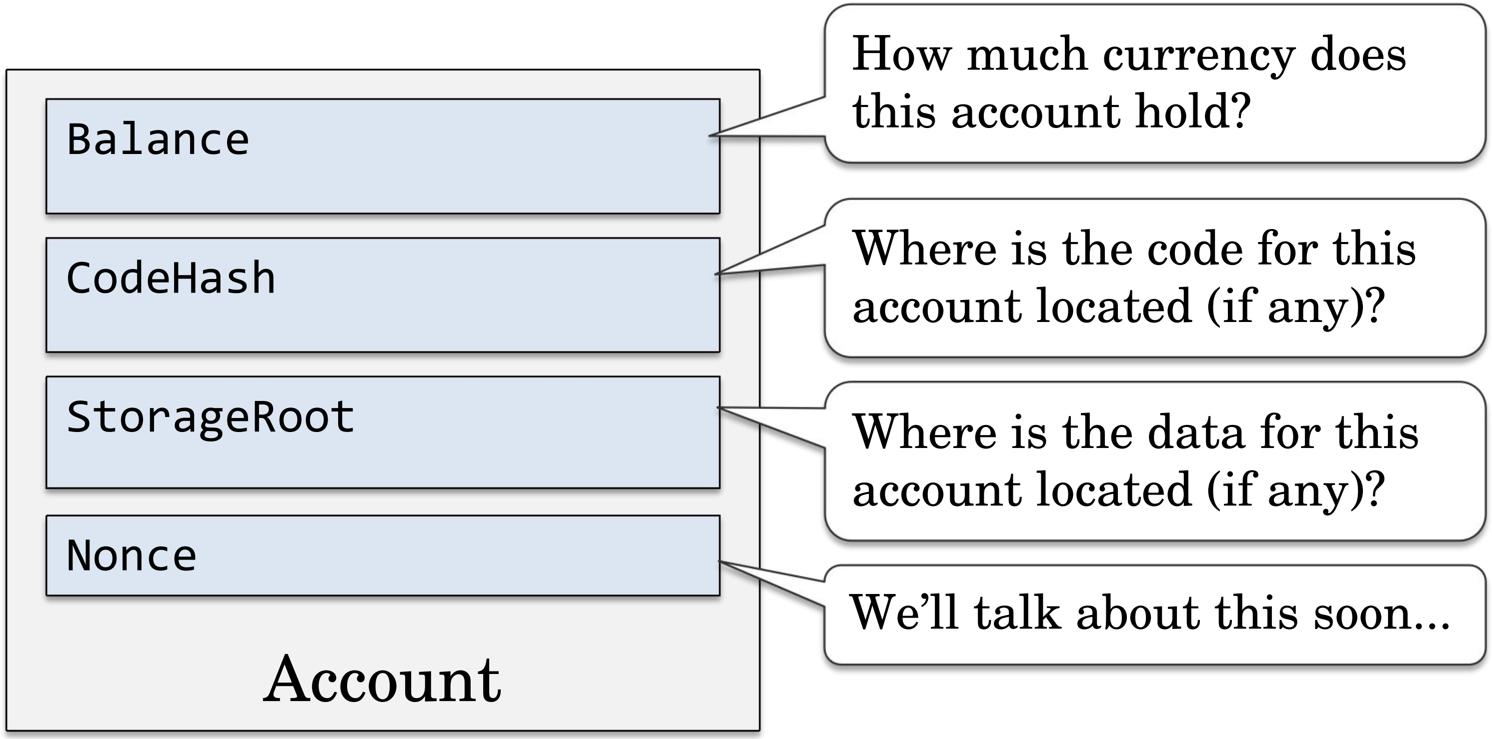


Blocks contain additional information about (account/contract) state and transaction receipts (transaction outputs)

Why store the hash, but not the data?

Can recompute receipts and state by (re-)executing the transactions

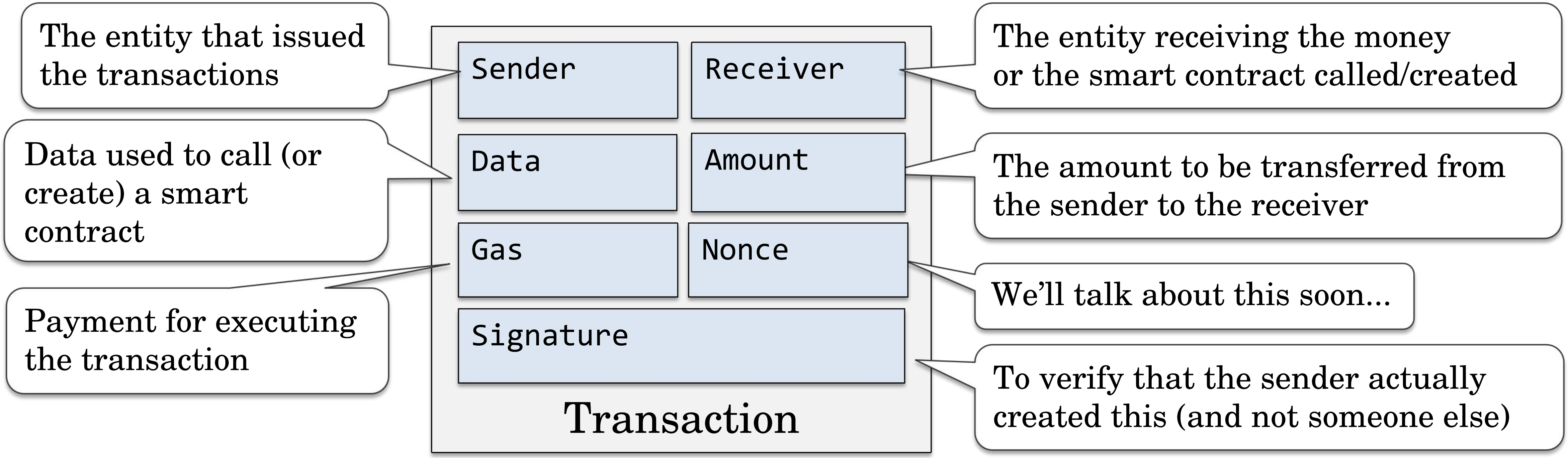
THE ACCOUNTS MODEL



Two Types of Accounts:

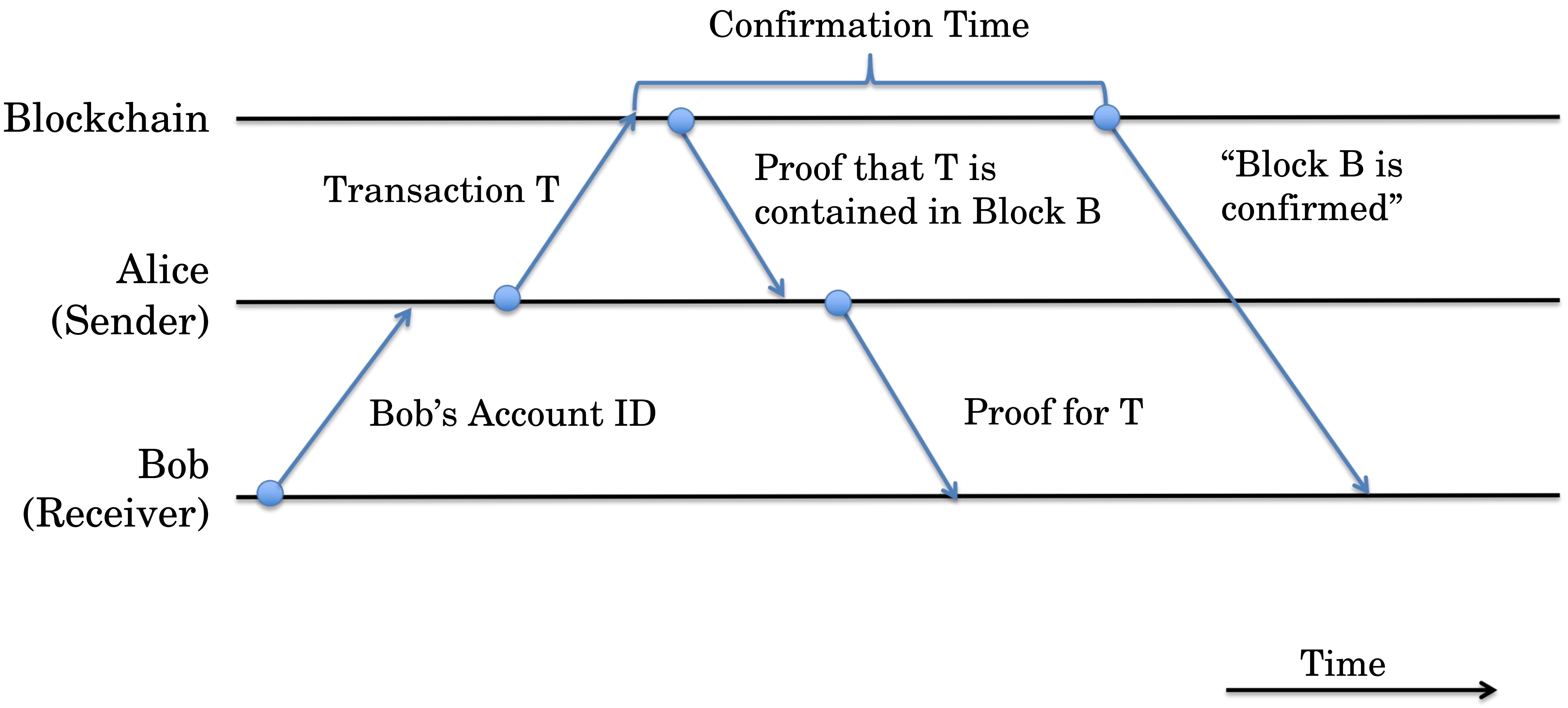
- **Externally-owned:** Controlled by one or multiple users
- **Smart Contracts:** Controlled by the blockchain

TRANSACTIONS IN THE ACCOUNTS MODEL

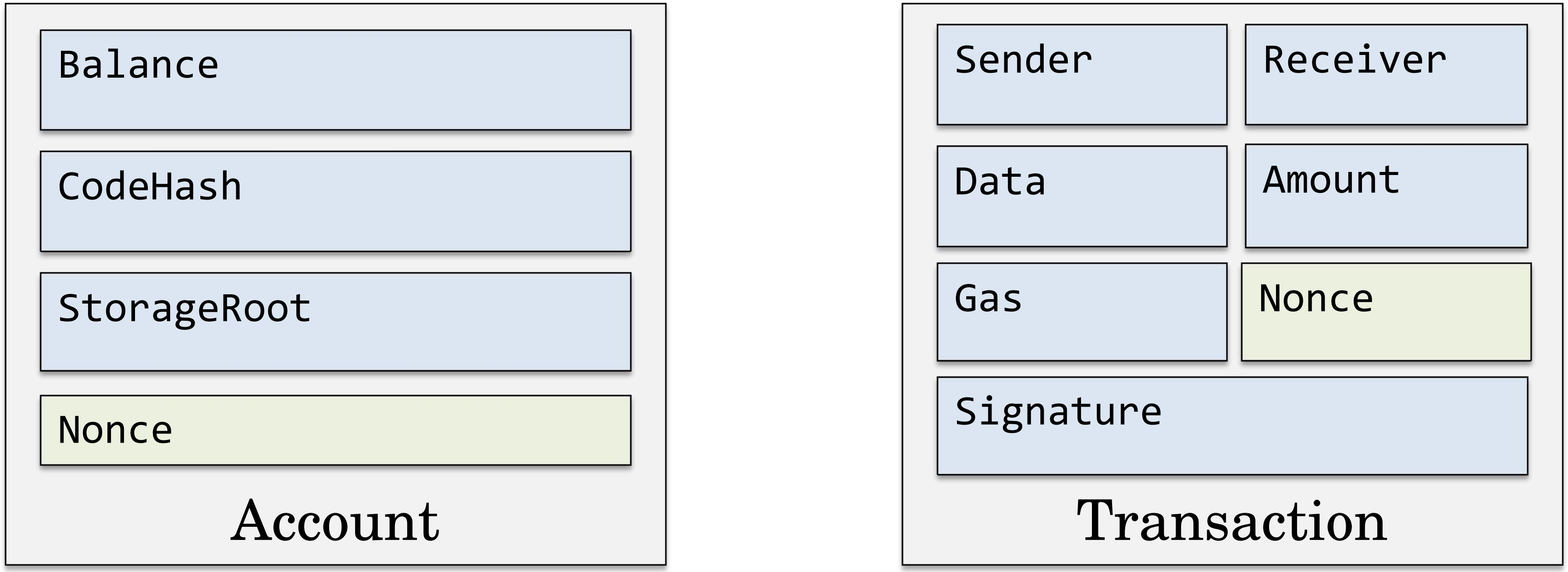


The Accounts model is more complicated, but also more expressive (as we will see soon)

MONEY TRANSFERS IN THE ACCOUNTS MODEL



NONCES IN ETHEREUM



Problem with the accounts model: **Replay attacks**

- No way to differentiate between two similar transactions and the same transaction being included multiple times by an attacker.

Nonce is a “**n**umber only used **once**”

- We increment the account’s nonce whenever a transaction is send “from” it
- A transaction is only valid if its nonce is equal to the sending accounts nonce

Gas in Ethereum

Gas pays for processing of transactions and execution of smart contracts

A transaction has some **base cost** (for validation etc.)

Each **execution step** has some gas cost

- Roughly proportional to the CPU cycles required to execute it
- Not all instructions have the same cost
  - e.g., addition (`ADD`) is much cheaper than exponentials (`EXP`)
  - We'll learn more about the EVM op codes later

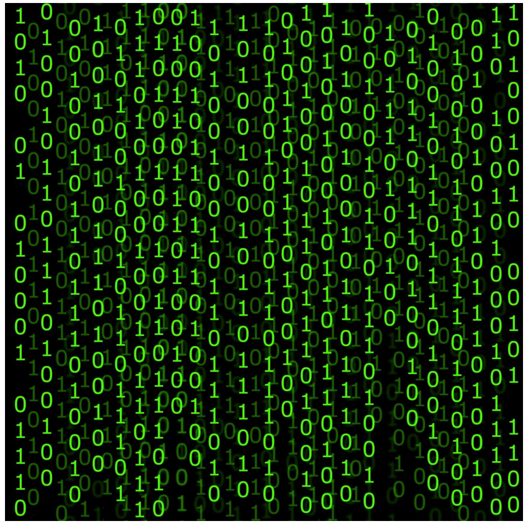
TRANSACTIONS VALIDITY IN ETHEREUM

Three things must hold for an Ethereum transaction to be valid

1. Sending account must exist and have at least `amount+gas` in its balance
2. Nonce must match the sending accounts nonce
3. Signature must match the sending accounts public key

SMART CONTRACT DEPLOYMENT

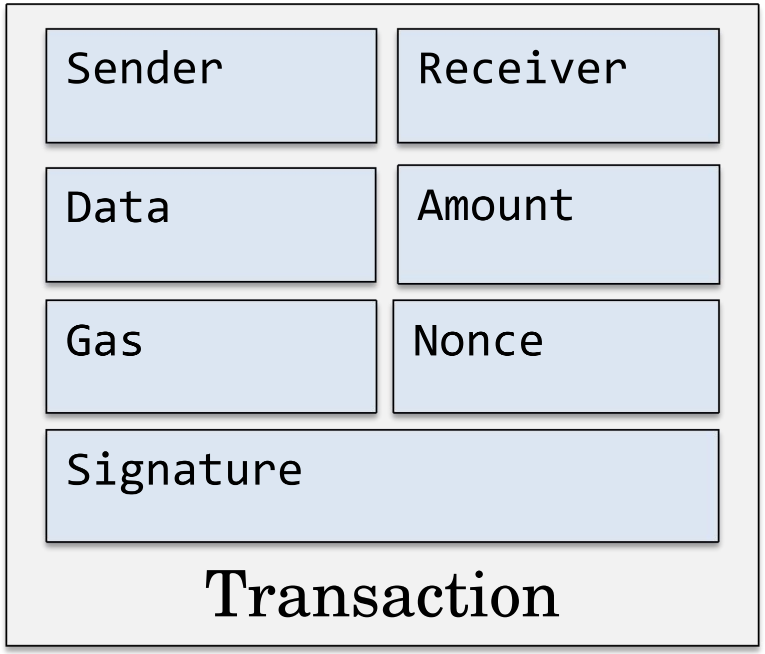
Step 1: Write code in a high-level language



Step 2: Compile program to byte code

Step 3: Store byte code on the blockchain

INTERACTING WITH CONTRACTS



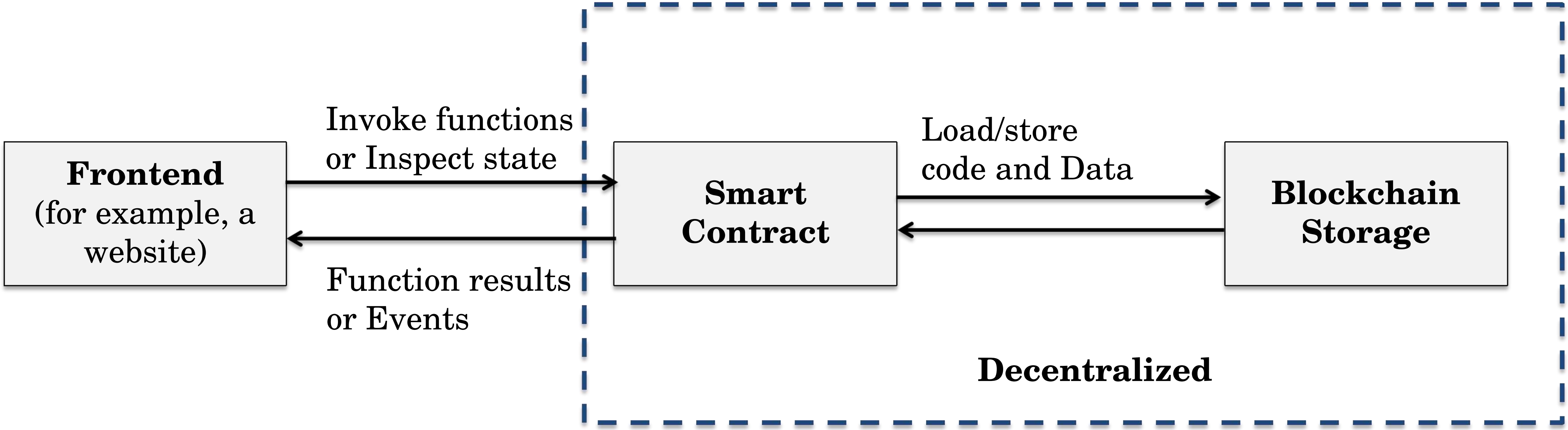
How do we get code onto the blockchain?

- Set `Receiver` to an unused address
- Store contract code in `Data`
- `Amount` is the initial balance of the smart contract

How do we call a smart contract?

- Set `Receiver` to the contract's address
- `Data` contains call information (function identifier and arguments)
- `Gas` allows paying for computation

DECENTRALIZED APPLICATIONS



Frontends are **stateless**

- Store no data and can be replaced easily

A decentralized app can consist of multiple smart contracts (not shown here)

THE VYPER PROGRAMMING LANGUAGE

- Most popular smart contract language after Solidity
- Less complex (=less features) than solidity
  - Easier to understand and harder to maker errors (hopefully)
- Syntax similar to Python
- I will use this for most examples, but you can use Solidity for the projects as well

VYPER SYNTAX

"Python with types"

```
def get_value() -> int128:
      # Defining a list
      example_list: int128[3]

      # Setting values
      example_list = [10, 11, 12]
      example_list[2] = 42

      # Returning a value
      return example_list[0]
```

VYPER STORAGE

- Simply define state as global variables
- Access it using the `self` keyword.

Other contracts and accounts can call this

```
 # cannot be changed after the contract is created
value: immutable(bool)

 # other contracts can read this
another_value: public(int256)

@external
def __init__(val1: bool, val2: int256):
      # Constructor will be called when
      # the contract is created
      self.value = val1
      self.another_value = val2
```

VYPER DECORATORS

We can use decorators to limit what a function can do

Can only read contract state

Can't access contract state at all

```
value: bool
another_value: public(int256)1

[..]

@view
@external
def get_value() -> bool
     return self.value

@pure
@external
def get_constant() -> uint128:
     return 1
```

ACCESSING TRANSACTION DATA

You can use the `msg` keyword to access information about the caller

```
@external
def get_caller() -> account:
     return msg.sender
```

DEMO

THAT'S ALL FOR TODAY

Next Time:

- Smart contracts calling other smart contracts
- (Non-fungible) Tokens
- Decentralized Exchanges