

E-Payment Systems and Cryptocurrency Technologies Spring Semester, 2021

<https://course.ie.cuhk.edu.hk/~ftec4004>

Prof. Wing C. Lau

wclau@ie.cuhk.edu.hk

<http://www.ie.cuhk.edu.hk/~wclau>

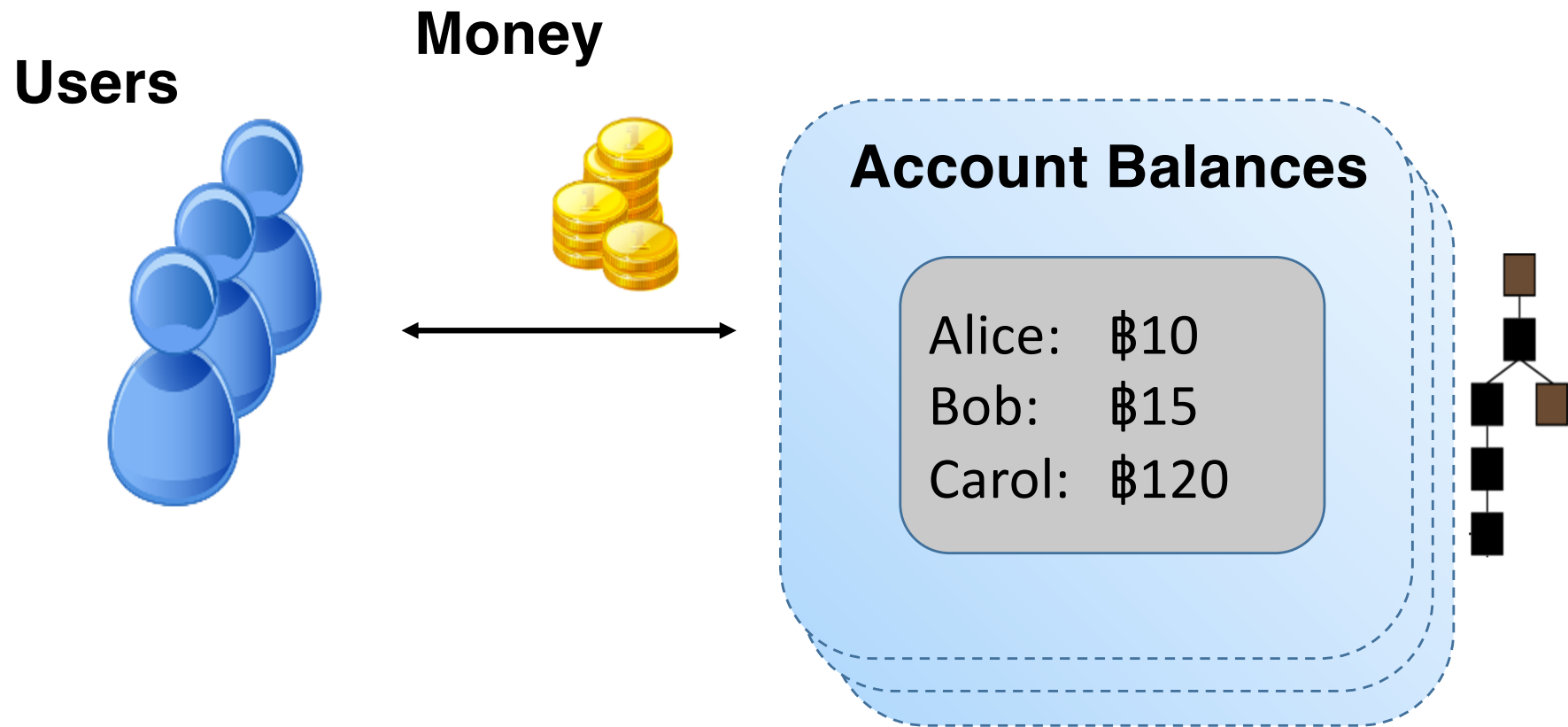
Introduction to Smart Contract and Ethereum

Acknowledgements

- The slides used in this lecture are mostly adapted from the following sources. The copyrights and contribution of the original authors are hereby acknowledged and recognized:
 - ◆ Sherman S.M. Chow, IERG5590 Advanced Blockchain, CUHK, 2020.
 - ◆ Andrew Miller, ECE398 SC: Smart Contracts Security and Blockchain Security, UIUC, Spring 2018.
 - ◆ Andreas Antonopoulos, Gavin Wood, *Mastering Ethereum – Building Smart Contracts and DApps*, Publisher: O'Reilly, Dec 2018.
 - ◆ Loi Luu, Ethereum and Smart Contracts, Winter School on Cryptocurrency and Blockchain Technologies, Shanghai, Jan 2017.
 - ◆ Foteini Baldimtsi, CS795 Blockchain Technologies, George Mason University, 2017, http://www.baldimtsi.com/teaching/cs795_sp17
 - ◆ Andrew Miller, ECE/CS 598AM: Cryptocurrency Security, UIUC, Fall 2016.
 - ◆ A. Narayanan, J. Bonneau, E.W. Felten, A. Miller, S. Goldfeder, J. Clark, *Bitcoin and Cryptocurrency Technologies*, Princeton Press, July 2016
<http://bitcoinbook.cs.princeton.edu>
 - ◆ Stefan Dziembowski, University of Warsaw, <https://www.crypto.edu.pl/dziembowski-talks>
 - ◆ ©2016 by Stefan Dziembowski. Permission to make digital or hard copies of part or all of this material is currently granted without fee *provided that copies are made only for personal or classroom use, are not distributed for profit or commercial advantage, and that new copies bear this notice and the full citation.*

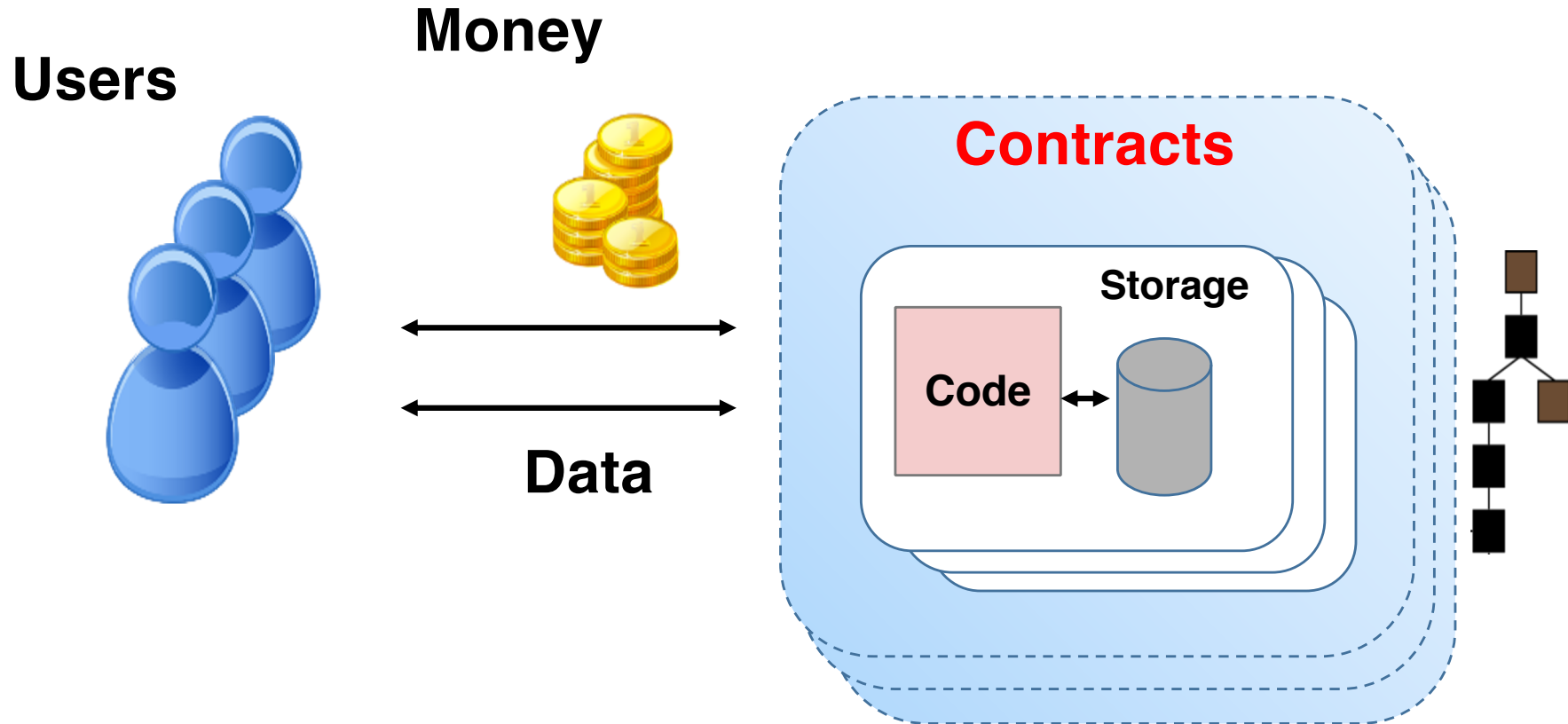
Digital currency is just one application on top of a blockchain

Decentralized Consensus “Blockchain”



Smart Contracts: user-defined programs running on top of a blockchain

Decentralized Consensus “Blockchain”



Definition:

A **Smart Contract** is a **computer program** executed in a **secure environment** that directly controls **digital assets**

Bitcoin transaction syntax

From the previous lecture -- “standard” transactions:



P₂

T₂ =

(User P₁ sends **1 BTC** from T₁ to P₂ signature of P₁ on [T₂])



P₃

T₃ =

(User P₂ sends **1 BTC** from T₂ to P₃ signature of P₂ on [T₃])

Standard transactions:



$T_2 =$

(User P_1 sends 1 BTC from T_1 to P_2 signature of P_1 on $[T_2]$)



$T_3 =$

(User P_2 sends 1 BTC from T_2 to P_3 signature of P_2 on $[T_3]$)

Strange transactions:

a Boolean function



$T_2 =$

T_1

1 BTC

a condition C_2 to spend T_2 a "witness W_2 "



$T_3 =$

T_2

1 BTC

a condition C_3 to spend T_3 a "witness W_3 "

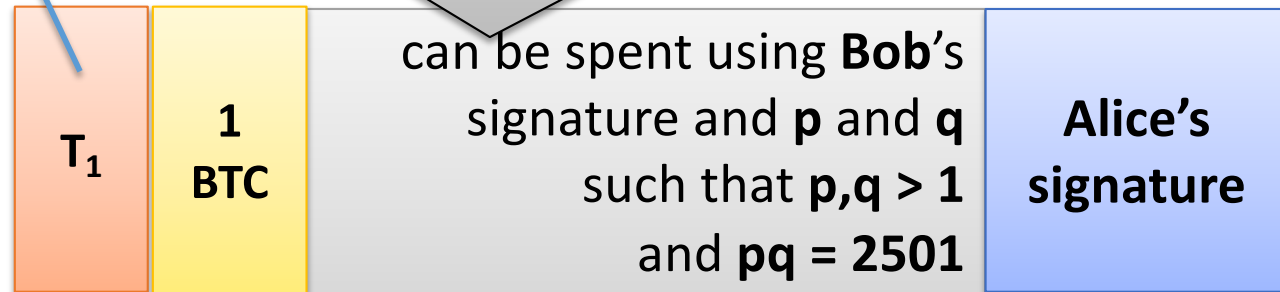
Example: "Alice gives 1 BTC to the Bob if he factors 2501."

T_1 --- earlier transaction that can be spent by Alice



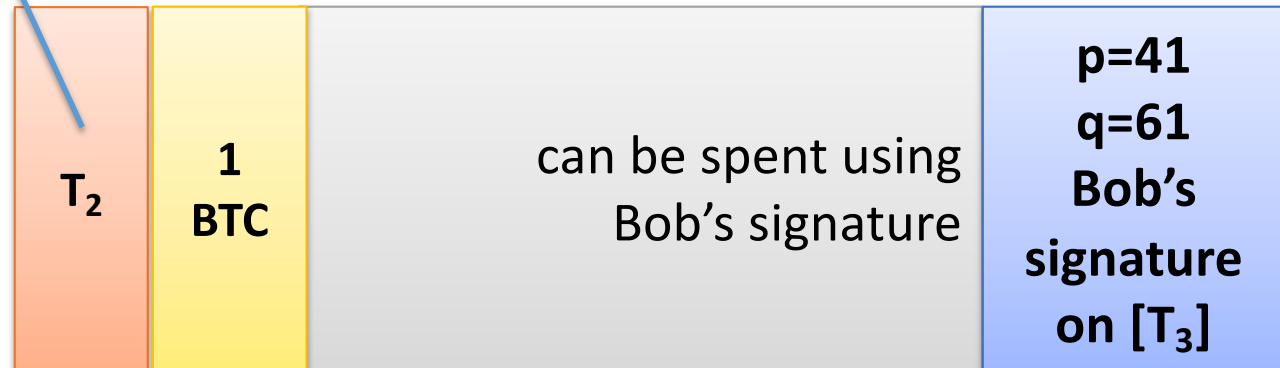
Alice posts:

$T_2 =$



Bob claims the money by posting:
aka:

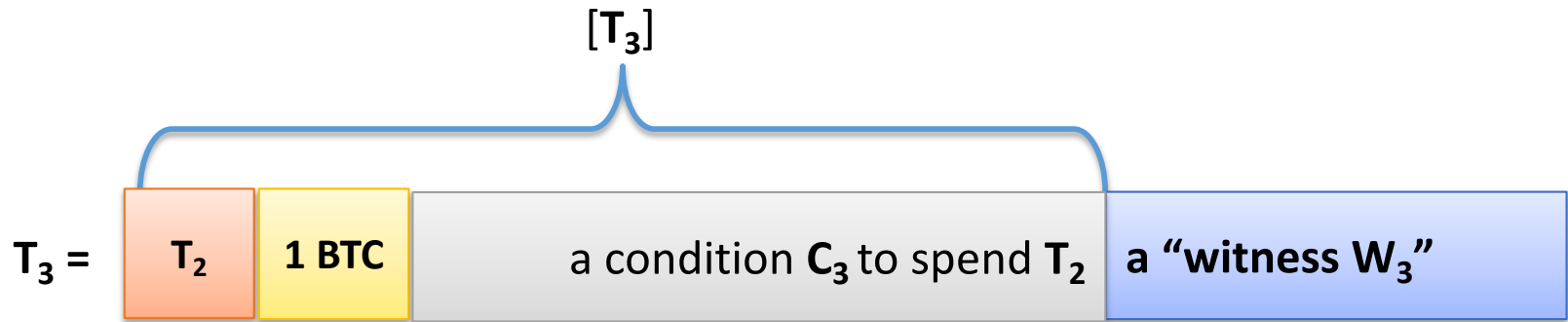
$T_3 =$



T_3 redeems T_2

formally:
 $C([T],(p,q)) = \text{true}$ iff
 $p, q > 1$ & $pq = 2501$
and is Bob's signature on $[T]$

Redeeming condition



T_3 redeems T_2 if

C_2 evaluates to **true** on input **($[T_3], W_3$)**.

Note: in the the standard transactions:

$$C_2([T_3], W_3) = \text{Vrfy}(\text{pk}_2, [T_3], W_3)$$

How are the conditions written?

In **Bitcoin scripting language** for
its Strange Transactions
(**non-Turing complete** stack-based)

Example:

```
OP_DUP OP_HASH160  
02192cfd7508be5c2e6ce9f1b6312b7f268476d2  
OP_EQUALVERIFY OP_CHECKSIG
```

Bitcoin contracts

The “**Strange transactions**” can be used to create the “**Bitcoin contracts**”.

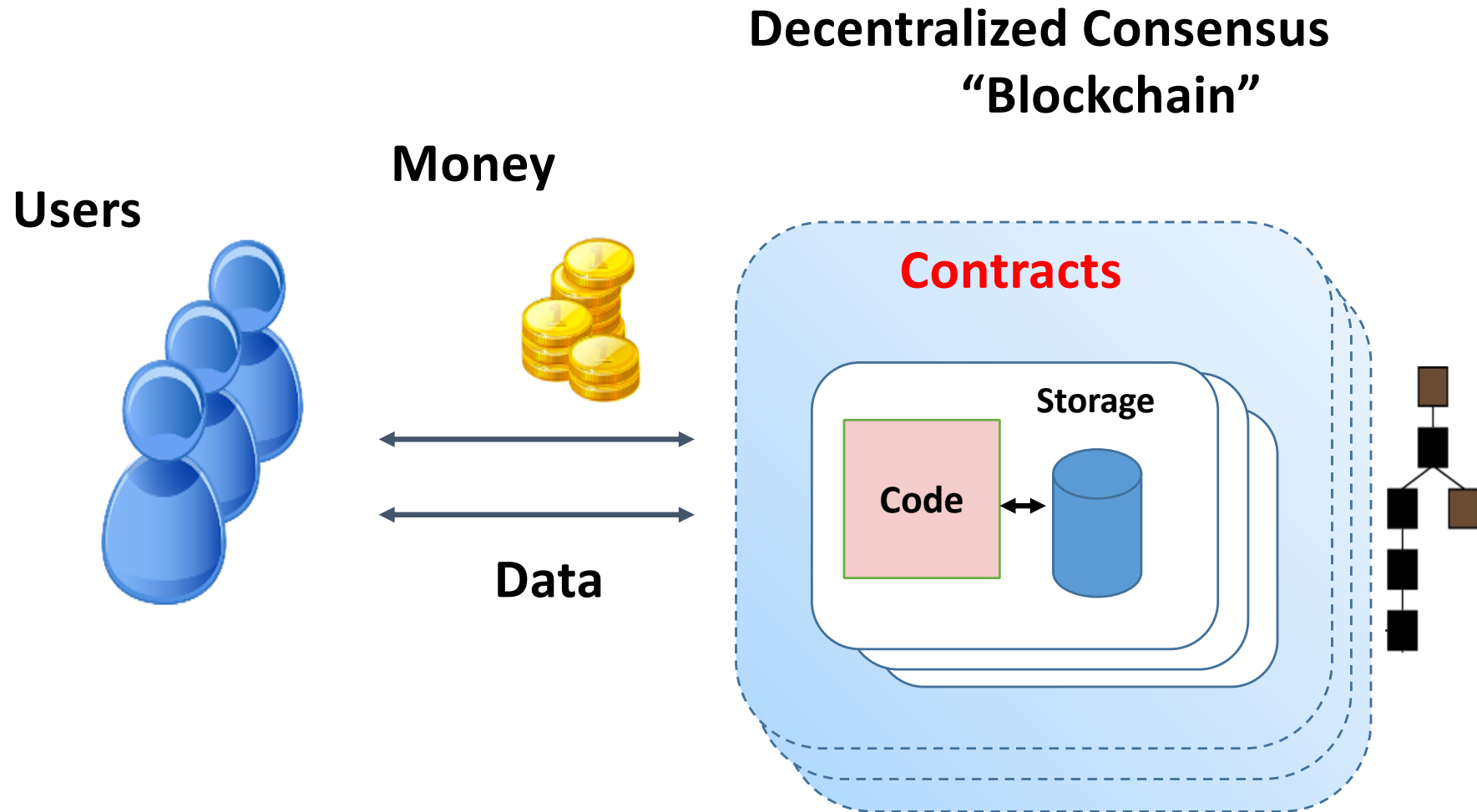
Simple examples:

- **Payment channels**
Pay money to anyone who knows some **password**.
- **Assurance contracts.**
Put a “**deposit**” to prove you are not a spammer.
Pay money only **if some event happens** (may require an oracle).

More advanced examples:

- “**Decentralized organizations**”
- **Secure multiparty computation** protocols [**Andrychowicz, D., Malinowski, Mazurek, 2014, Bentov and Kumaresan 2014**].

Smart Contracts: user-defined programs running on top of a blockchain



Definition:

A **Smart Contract** is a **computer program** executed in a **secure environment** that directly controls **digital assets**

History of “Smart contracts”: conceptualized by Szabo in 1994

A smart contract is a computerized transaction protocol that executes the terms of a contract. The general objectives are to satisfy common contractual conditions (such as payment terms, liens, confidentiality, and even enforcement), minimize exceptions both malicious and accidental, and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitrations and enforcement costs, and other transaction costs.

-Nick Szabo “The Idea of Smart Contracts”

A Concise and General Definition of Smart Contract

A smart contract is a computer program executed in a secure environment that directly controls digital assets

A Concise and General Definition of Smart Contract (cont'd)

A smart contract is a **computer program** executed in a secure environment that directly controls digital assets

A Concise and General Definition of Smart Contract (cont'd)

*A **computer program** is a collection of instructions that performs a specific task when executed by a computer. A computer requires programs to function, and typically executes the program's instructions in a central processing unit.*

[Wikipedia](#)

Example: Bet on an event

```
if HAS_EVENT_X_HAPPENED() is true:  
    send(party_A, 1000)  
else:  
    send(party_B, 1000)
```

A Concise and General Definition of Smart Contract (cont'd)

A smart contract is a computer program executed in a **secure environment** that directly controls digital assets

Properties of Secure Environments

- Correctness of execution
 - The execution is done correctly, is not tampered
- Integrity of code and data
- *Optional* properties
 - Confidentiality of code and data
 - Verifiability of execution
 - Availability for the programs running inside

Examples of Secure Environments

- Servers run by Trusted Parties
- Decentralized computer network (i.e. **Blockchains**)
- Quasi-decentralized computer network (i.e. Consortium (permission-based) Blockchains)
- Servers secured by trusted hardware (e.g. SGX)



A Concise and General Definition of Smart Contract (cont'd)

A smart contract is a computer program executed in a secure environment that **directly controls** digital assets

Example

- Legal contract: “I promise to send you \$100 if my lecture is rated 1*”
- Smart contract: “I send \$100 into a computer program executed in a secure environment which sends \$100 to you if the rating of my lecture is 1*, otherwise it eventually sends \$100 back to me”

A Concise and General Definition of Smart Contract (cont'd)

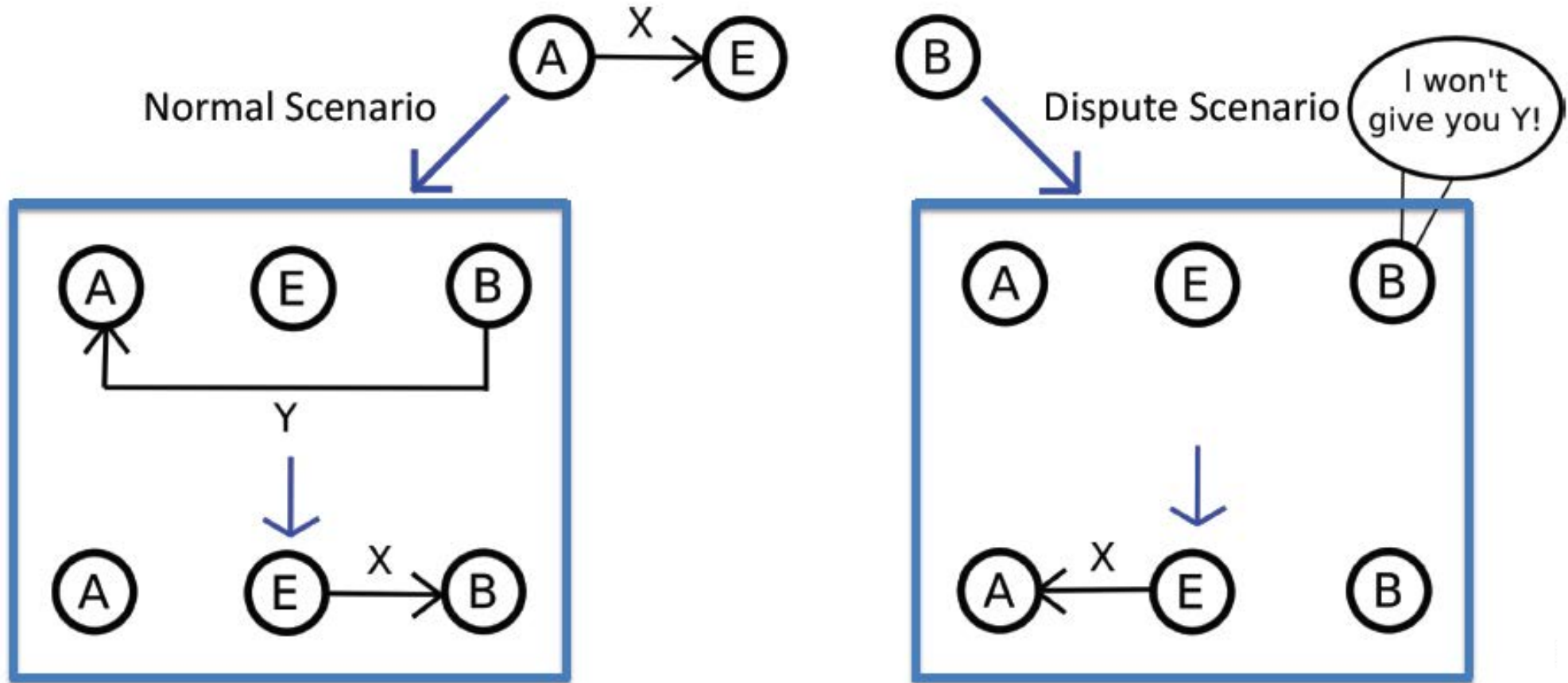
A smart contract is a computer program executed in a secure environment that directly controls **digital assets**

What are Digital Assets?

- A broad category
 - Domain name
 - Website
 - E-Money
 - Anything Tokenizable (e.g. Gold, Silver, Stock share etc)
 - Game items
 - Network bandwidth
 - Computation cycles
 - ...

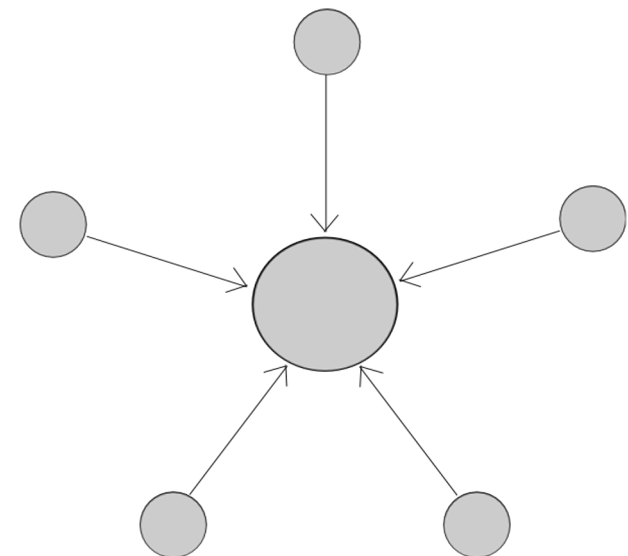
Smart Contract-based Application, an Example: Escrow Service for Exchange

Two individuals A and B are exchanging their corresponding object X and Y



Another Application Example of Smart Contract: Multi-Signature

- Require M out of N “owners” to agree for a particular Digital Asset to be transferred:
 - Intra-organizational Use Cases
 - Make sense even for some Use Cases involved only 1 single Individual, e.g.
 - Two-Factor Authentication
 - Two private Signing Keys stored in different storage media/ system...



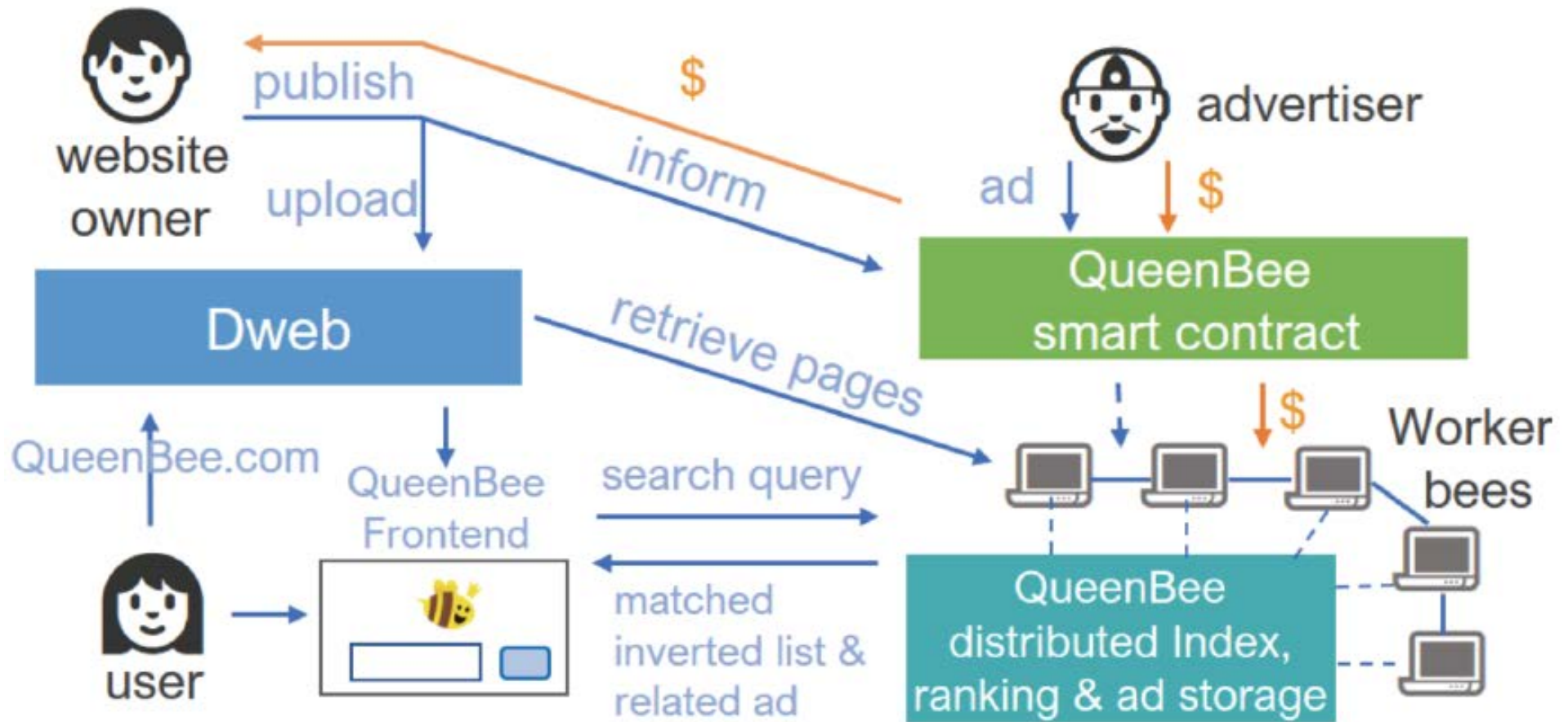
A lot of other Interesting Applications of Smart Contracts

- Individual/intra-organizational
 - Complex access policies depending on amount, withdrawal limits, etc
 - Dead man's switch, "digital will"
 - e.g., When the owner dies, transfer all assets to someone
- General
 - Prediction markets
 - Insurance
 - Micro-payments for computational services (file storage, bandwidth, computation, etc)
 - Games Puzzles with Incentives
 - Gambling and Decentraized Casinos

General Theme of these Applications: Decentralized Exchange

- Permissionless Blockchain => Decentralization
=> Democratization ??
- Democratization access to Financial Services
- How do you buy/sell Foreign Currencies now ?
- How “Sharing Economy Applications” (SEA) works ?
 - Airbnb, Zipcar, Uber
- Decentralized Exchange can provide the platform for “SEA”
 - People exchange goods/services/assets among themselves over a decentralized platform, i.e. a Blockchain, instead of via a dealer or centralized authorities.

Another Example (Idea): QueenBee: Decentralized Search on Decentralized Web



<http://cidrdb.org/cidr2019/gongshow.html>

Also see: The Web3.0 idea proposed by Gavin Wood (co-founder of Ethereum)
<https://gavwood.com/dappsweb3.html> ; <https://gavwood.com/web3lt.html>
<https://medium.com/@gavofyork/why-we-need-web-3-0-5da4f2bf95ab>

Why are they called “Smart” Contracts ?

- Automated processing
 - Facilitate, Verify and/or Enforce the Execution of a contract
- Trust reduction
 - Trust the Secure Execution Environments, not depending on a very large number of Contract Enforcement mechanisms
- Trackable and Irreversible
- Unambiguous, terms clearly expressed in code
 - Question: how to express terms clearly in code?

Smart Contracts vs Legal Contracts

	Legal	Smart
Specification	Natural language + “legalese”	Code
Identity & Consent	Signatures	Digital signatures
Dispute resolution	Judges, Arbitrators	Decentralized platform
Nullification	By Judges	????
Payment	Carried out by parties separately	Built-in
Escrow	Trusted third party, settled in \$	Built-in

Smart Contracts vs Legal Contracts

- A smart contract is more like a vending machine
 - Follow predetermined rules

Legal contracts	Smart contracts
Good at subjective (i.e. requiring human judgement) claims	Good at objective (i.e. mathematically evaluable) claims
High cost	Low cost
May require long legal process	Fast and automated
Relies on penalties	Relies on collateral/security deposits
Jurisdiction-bound	Potentially International

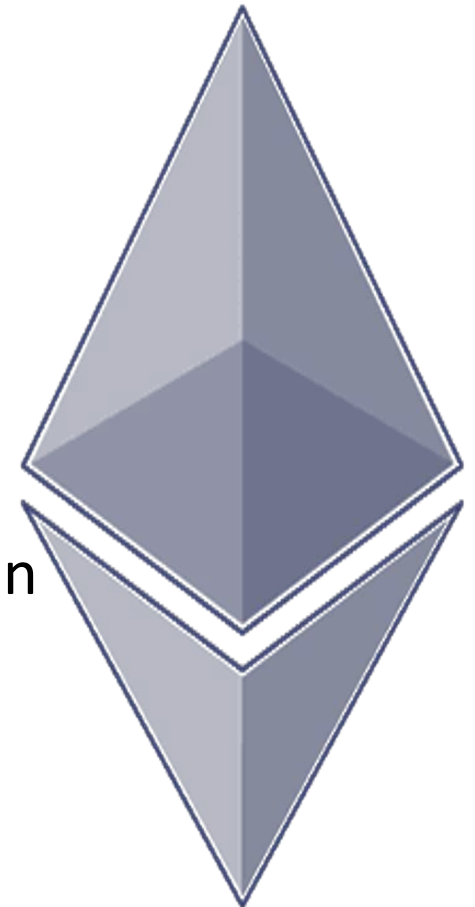
Smart Contracts vs Legal Contracts

- Smart contracts are not very effective for loans
 - Has the capital to provide liquid collateral for a loan, do not need the loan in the first place
 - Can use illiquid collateral though (eg. domain names)
- Legal contracts are not very effective for the anti-spam use case
 - Amounts at stake are so small
 - Spammers can locate themselves in favorable jurisdictions and evade detection

Ethereum:
The First Blockchain-based
Smart Contract Platform

Ethereum

- Blockchain with expressive programming language
 - Programming language makes it ideal for smart contracts
- Why? (back then, circa 2016)
 - Most public blockchains are cryptocurrencies
 - Mainly designed for transferring coins between users
 - Smart contracts enable much more applications



About Ethereum

Crowdfunded ~\$20M in ~ a month
Popularized a grand vision of
“generalized” cryptocurrency

Flexible scripting language
“pyethereum” simulator, 2014



Co-founder of Ethereum:
Vitaly Dmitriyevich “Vitalik” Buterin



[HOME](#)

[TXS](#)



TOTAL SUPPLY OF 73,891,107.34 ETHER
\$0.66 @ 0.00271 BTC/ETH

LAST BLOCK

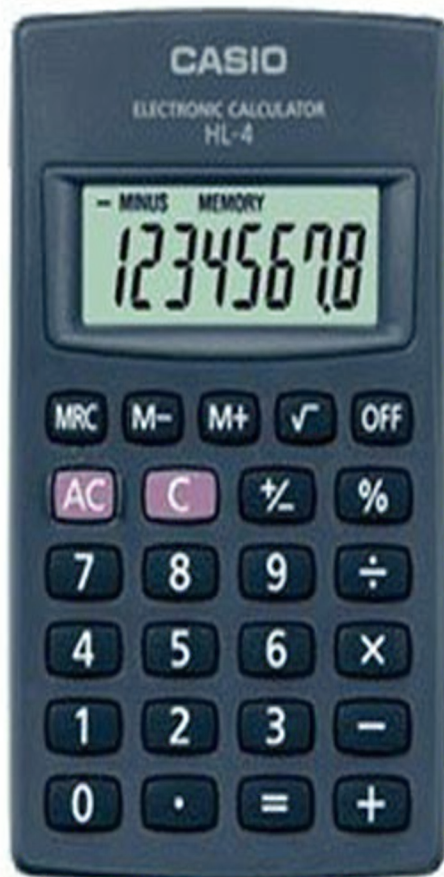
358122 (14.563s Avg)

TRANSACTIONS

324551

Analogy: Most existing blockchain protocols were designed like

 **bitcoin**



OR THIS



Why not make a protocol that works like



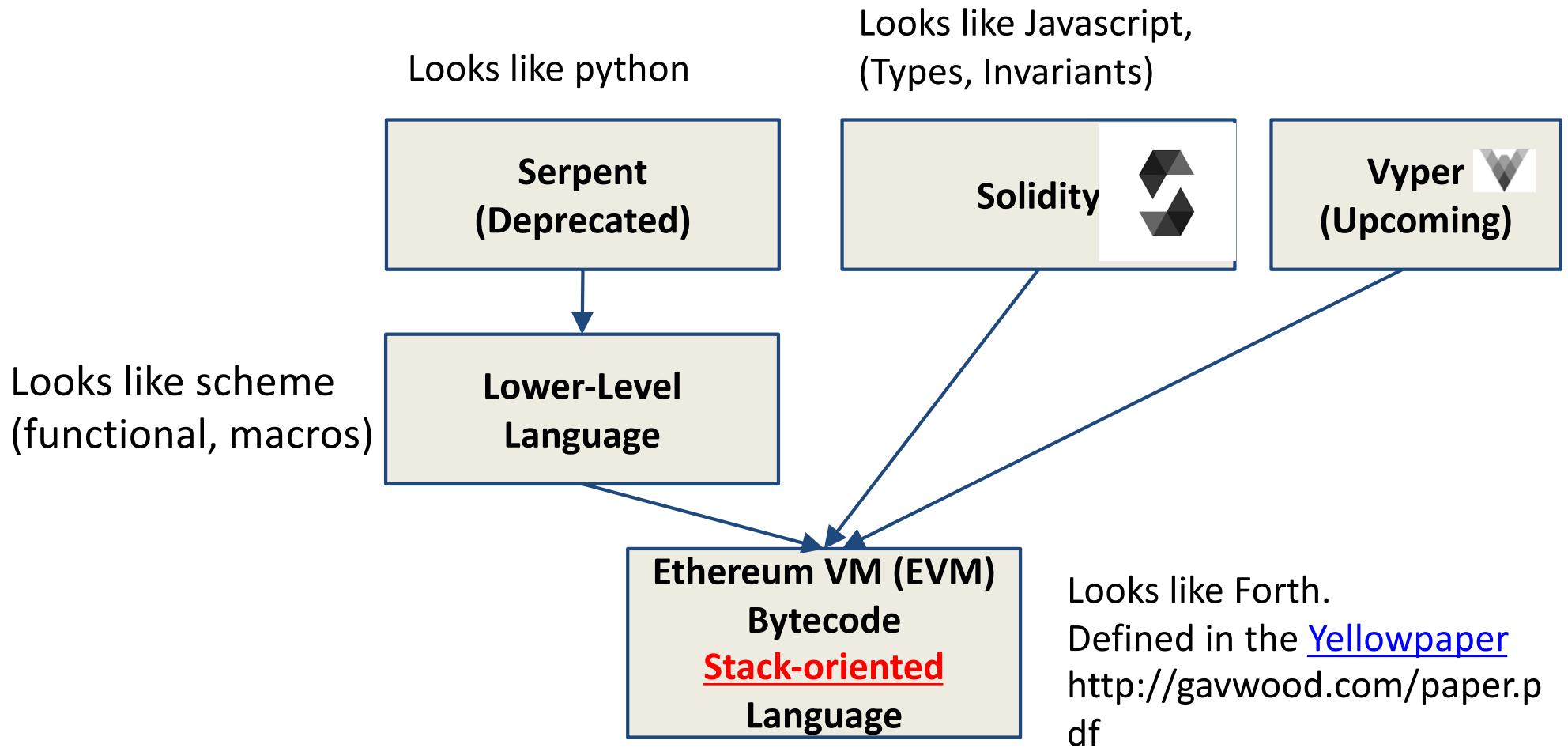
OR THIS



OR THIS



Ethereum Languages



The Solidity Language



- Currently the most common (default) language used for Ethereum Smart Contract Programming
- Syntax looks like JavaScript
- Contracts look like classes/objects
- Static typing
 - Most types can be cast e.g. `bool(x)`
- `bool`, `uint8`, `uint16`, ... `uint256`, `int8`, ... `int256`
- `address`
- `string`
- `byte[]`
- `mapping(keyType => valueType)`

Solidity Contract programming model

```
1 pragma solidity ^0.4.19;
2 contract TestContract {
3
4     // declare storage
5     int[] myStorage1;
6
7     // define functions
8     function testFunction() {
9         // Code goes here
10    }
11
12    // ...
13 }
```

- Contract class

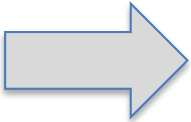
 Create an object of this class by making a transaction

- Define functions you can call

Workflow

```
1 contract Greetings {  
2     string greeting;  
3     function Greetings (string _greeting) public {  
4         greeting = _greeting;  
5     }  
6  
7     /* main function */  
8     function greet() constant returns (string) {  
9         return greeting;  
10    }  
11 }
```

What you write



What other see on the blockchain

60606040526040516102503
80380610250833981016040
528.....



PUSH 60
PUSH 40
MSTORE
PUSH 0
CALLDATALOAD
.....

What people get from the disassembler

How Ethereum Works

- Two types of account:
 - **Normal account** like in Bitcoin,
 - aka Externally Owned Account (EOA)
 - has balance and address
 - **Smart Contract account**
 - like an object: containing (i) code, and (ii) private storage (key-value storage)
 - Code can
 - Send ETH to other accounts
 - Read/write storage
 - Call (ie. start execution in) other contracts

Ethereum's Account-based Model vs. Bitcoin's UTXO Model

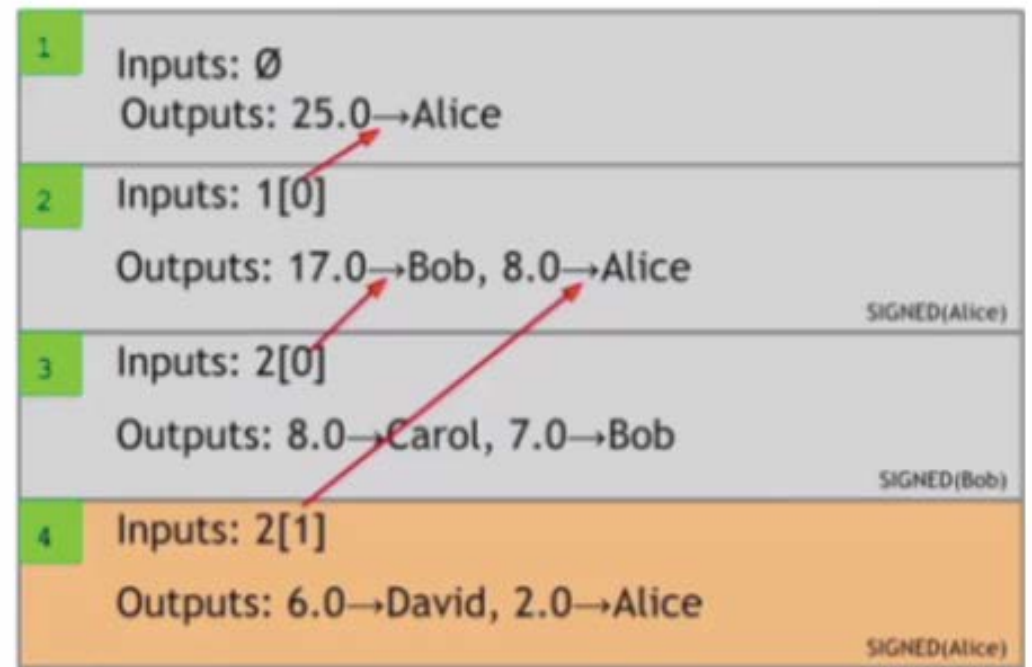
Ethereum

Alice		
Date	Description	Balance
01/01/19	...	\$25.0
01/02/19	\$17 -> Bob	\$8.0

Bob		
Date	Description	Balance
01/01/19	...	\$25.0
01/02/19	Alice -> \$17	\$17.0

time ↓

Bitcoin



SIMPLIFICATION: only one transaction per block

DNS: The “Hello World” of Ethereum

```
data domains[](owner, ip)
```

Private
Storage

```
def register(domainname):
```

```
    if not self.domains[domainname].owner:  
        self.domains[domainname].owner = msg.sender
```

```
def set_ip(domainname, ip):
```

Can be invoked by
other accounts

```
    if self.domains[domainname].owner ==  
msg.sender:  
        self.domains[domainname].ip = ip
```

Transactions in Ethereum

- Normal transactions like Bitcoin transactions
 - Send tokens between accounts
- Transactions to contracts
 - like function calls to objects
 - specify which object you are talking to, which function, and what data (if possible)
- Transactions to create contracts

Transactions

- **nonce** (anti-replay-attack)
- **to** (destination address)
- **value** (amount of ETH to send)
- **data** (readable by contract code)
- **gasprice** (amount of ether per unit gas)
- **startgas** (maximum gas consumable)
- **v, r, s** (ECDSA signature values)

How to Create a Contract?

- Submit a transaction to the blockchain
 - **nonce**: previous nonce + 1
 - **to**: empty
 - **value**: value sent to the new contract
 - **data**: contains the code of the contract
 - **gasprice** (amount of Ether (ETH) per unit gas)
 - **startgas** (maximum gas consumable)
 - **v, r, s** (ECDSA signature values)
- If tx is successful
 - Returns the address of the new contract (derived from Creator Address and nonce)

How to Interact With a Contract?

- Submit a transaction to the blockchain
 - **nonce**: previous nonce + 1
 - **to**: contract address
 - **value**: value sent to the new contract
 - **data**: data supposed to be read by the contract
 - **gasprice** (amount of ether per unit gas)
 - **startgas** (maximum gas consumable)
 - **v, r, s** (ECDSA signature values)
- If tx is successful
 - Returns outputs from the contract (if applicable)

Blockchain State

Bitcoin's state consists of key value mapping addresses to account balance

Address	Balance (BTC)
0x123456...	10
0x1a2b3f...	1.0
0xab123d...	1.1

Ethereum's state consists of key value mapping addresses to account objects

Address	Object
0x123456...	X
0x1a2b3f...	Y
0xab123d...	Z

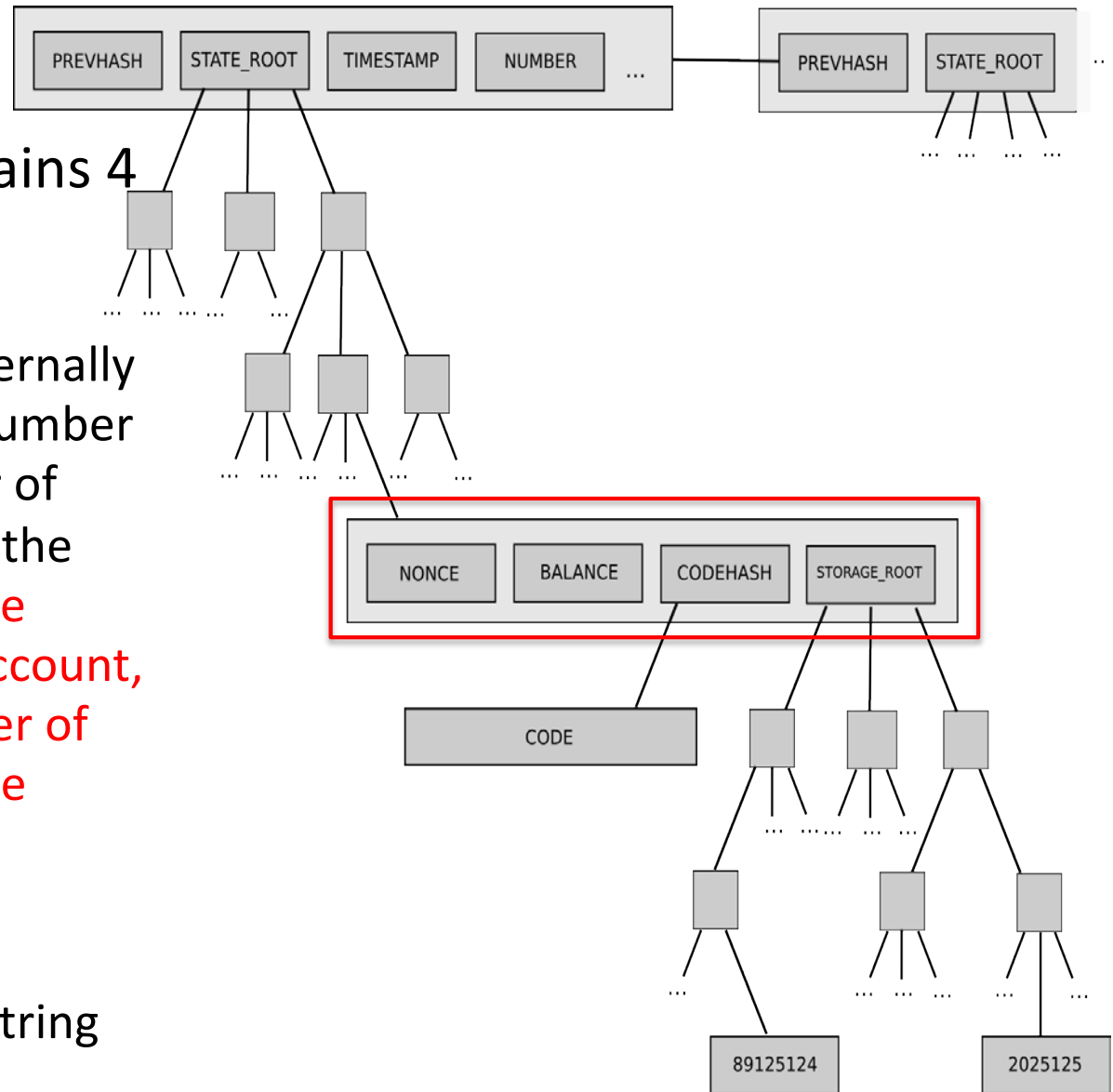
Blockchain != Blockchain State

Account Object

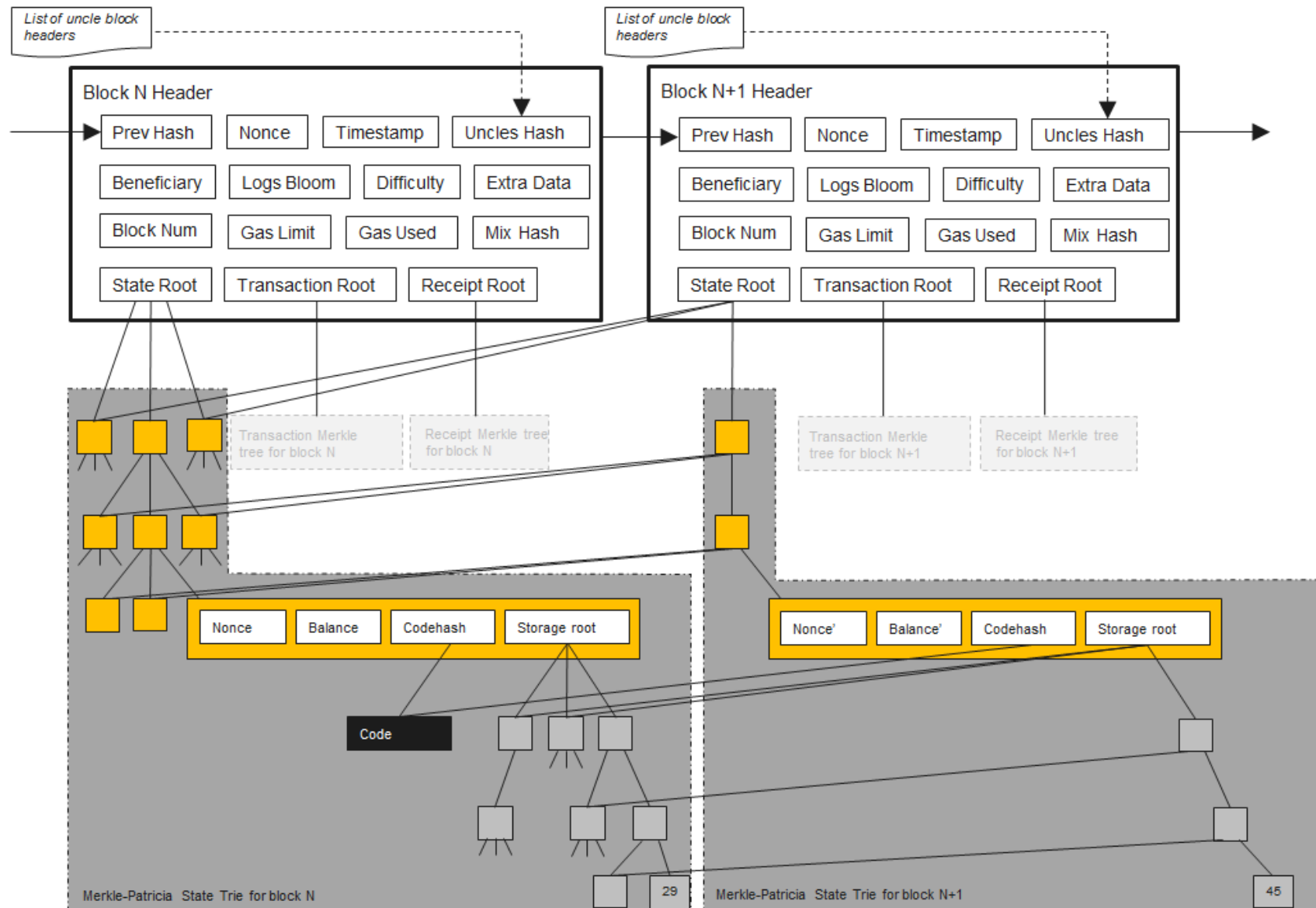
- Every account object contains 4 pieces of data:

- Nonce
 - If the account is an Externally Owned Account, this number represents the number of transactions sent from the account's address. **If the account is a contract account, the nonce is the number of contracts created by the account.**

- Balance
- Code hash (code = empty string for normal accounts)
- Storage trie root



Data-structure of the Ethereum Blockchain



Data-structure of the Ethereum Blockchain

```
type Blockchain struct {
    config *params.ChainConfig // chain & network configuration

    hc *HeaderChain
    chainDB ethdb.Database
    relLogsFeed event.Feed
    chainFeed event.Feed
    chainSideFeed event.Feed
    chainHeadFeed event.Feed
    logsFeed event.Feed
    scope event.SubscriptionScope
    genesisBlock *types.Block

    mu sync.RWMutex // global mutex for locking chain operations
    chainmu sync.RWMutex // blockchain insertion lock
    procMu sync.RWMutex // block processor lock

    checkpoint int // checkpoint counts towards the new checkpoint
    currentBlock *types.Block // Current head of the block chain
    currentFastBlock *types.Block // Current head of the fastsync chain (may be above the block chain)

    stateCache state.Database // State database to reuse between imports (contains state cache)
    bodyCache *lru.Cache // Cache for the most recent block bodies
    bodyRLPCache *lru.Cache // Cache for the most recent block bodies in RLP encoded format
    blockCache *lru.Cache // Cache for the most recent entire blocks
    futureBlocks *lru.Cache // Future blocks are blocks added for later processing

    quit chan struct{} // blockchain quit channel
    running int32 // running will be called atomically
    // procInterrupt must be atomically called
    procInterrupt int32 // interrupt signaler for block processing
    wg sync.WaitGroup // chain processing wait group for shutting down

    engine consensus.Engine
    processor Processor // block processor interface
    validator Validator // block and state validator interface
    vmConfig vm.Config

    badBlocks *lru.Cache // Bad block cache
}

// Block represents an entire block in the Ethereum blockchain.
type Block struct {
    header *Header
    uncles []*Header
    transactions Transactions

    // caches
    hash atomic.Value
    size atomic.Value

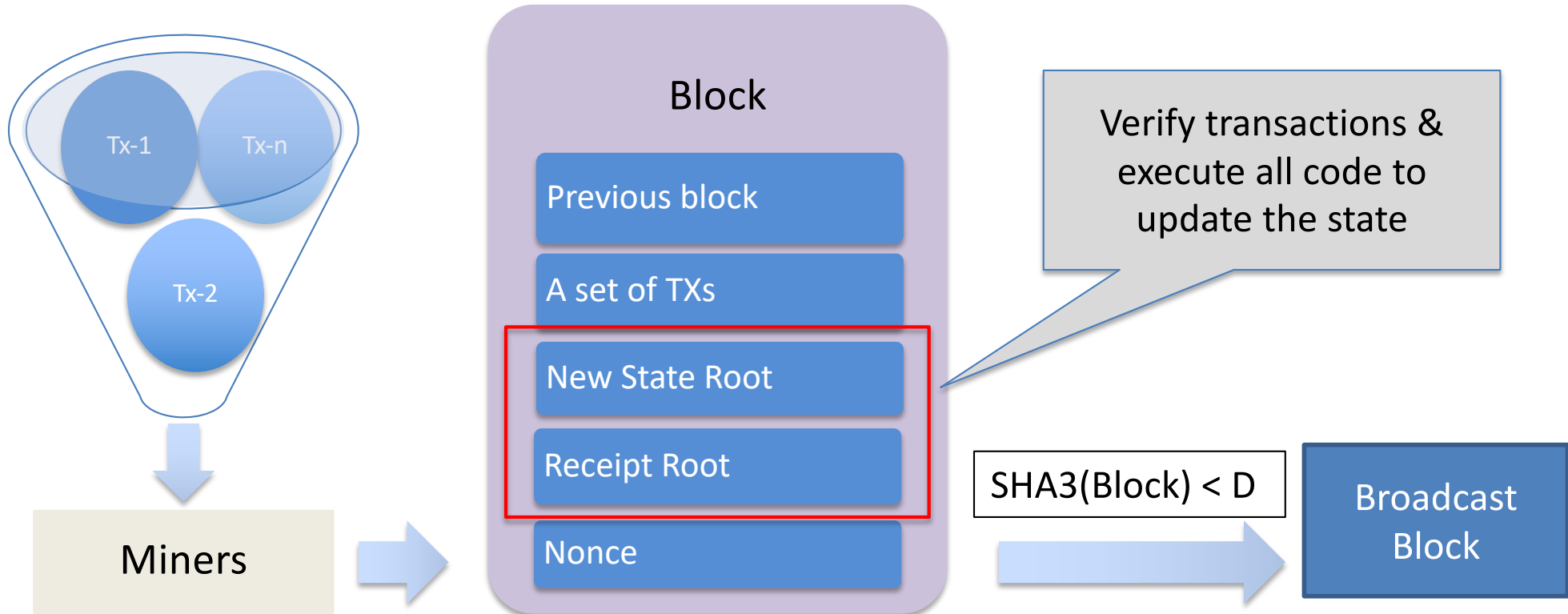
    // Id is used by package core to store the total difficulty
    // of the chain up to and including the block.
    td *big.Int

    // These fields are used by package eth to track
    // inter-peer block relay.
    ReceivedAt time.Time
    ReceivedFrom interface{}
}

// Header represents a block header in the Ethereum blockchain.
type Header struct {
    ParentHash common.Hash      json:"parentHash"    gencodec:"required"
    UncleHash common.Hash      json:"sha3Uncles"    gencodec:"required"
    Coinbase common.Address    json:"miner"         gencodec:"required"
    Root common.Hash      json:"stateRoot"     gencodec:"required"
    TxHash common.Hash      json:"transactionsRoot" gencodec:"required"
    ReceiptHash common.Hash    json:"receiptsRoot"  gencodec:"required"
    Bloom Bloom          json:"logsBloom"     gencodec:"required"
    Difficulty *big.Int        json:"difficulty"    gencodec:"required"
    Number *big.Int        json:"number"        gencodec:"required"
    GasLimit *big.Int        json:"gasLimit"      gencodec:"required"
    GasUsed *big.Int        json:"gasUsed"       gencodec:"required"
    Time *big.Int        json:"timestamp"     gencodec:"required"
    Extra []byte          json:"extraData"     gencodec:"required"
    MixDigest common.Hash      json:"mixHash"       gencodec:"required"
    Nonce BlockNonce     json:"nonce"         gencodec:"required"
}
```

ETHEREUM BLOCK STRUCTURE

Block Mining

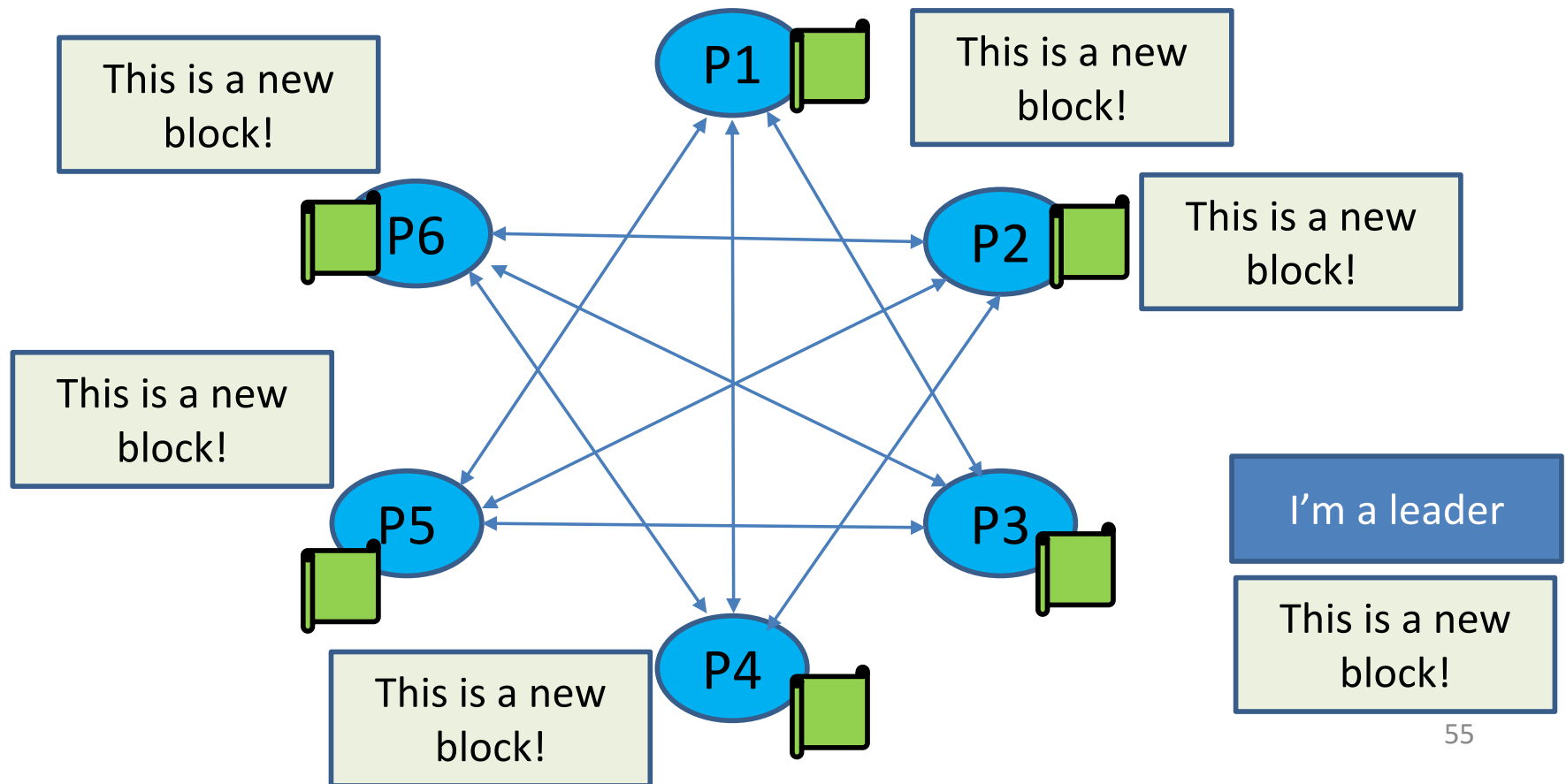


Receipt root is the root of the tree of all “receipts”.

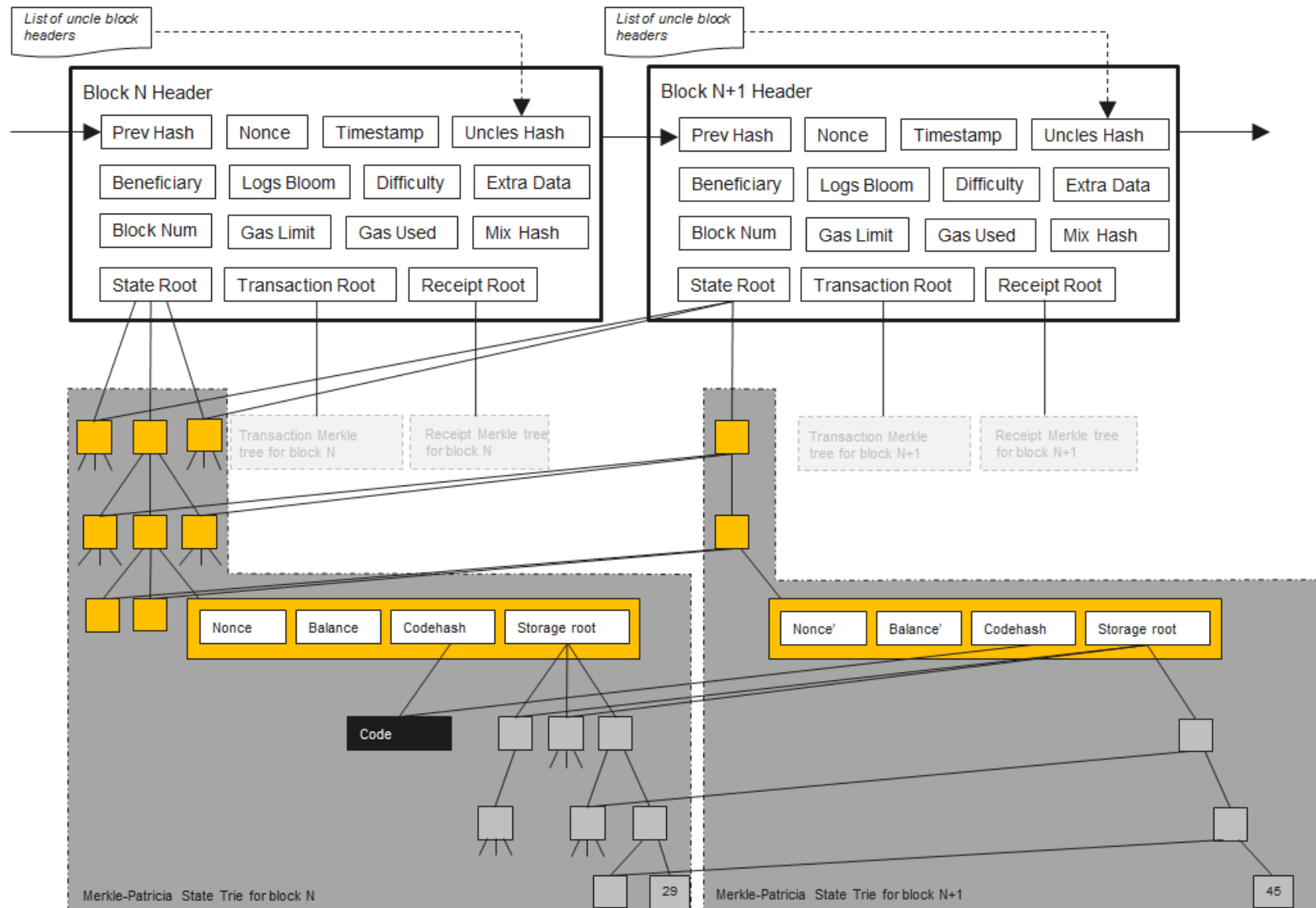
Each receipt represents an intermediate state root after 1 transaction is executed.

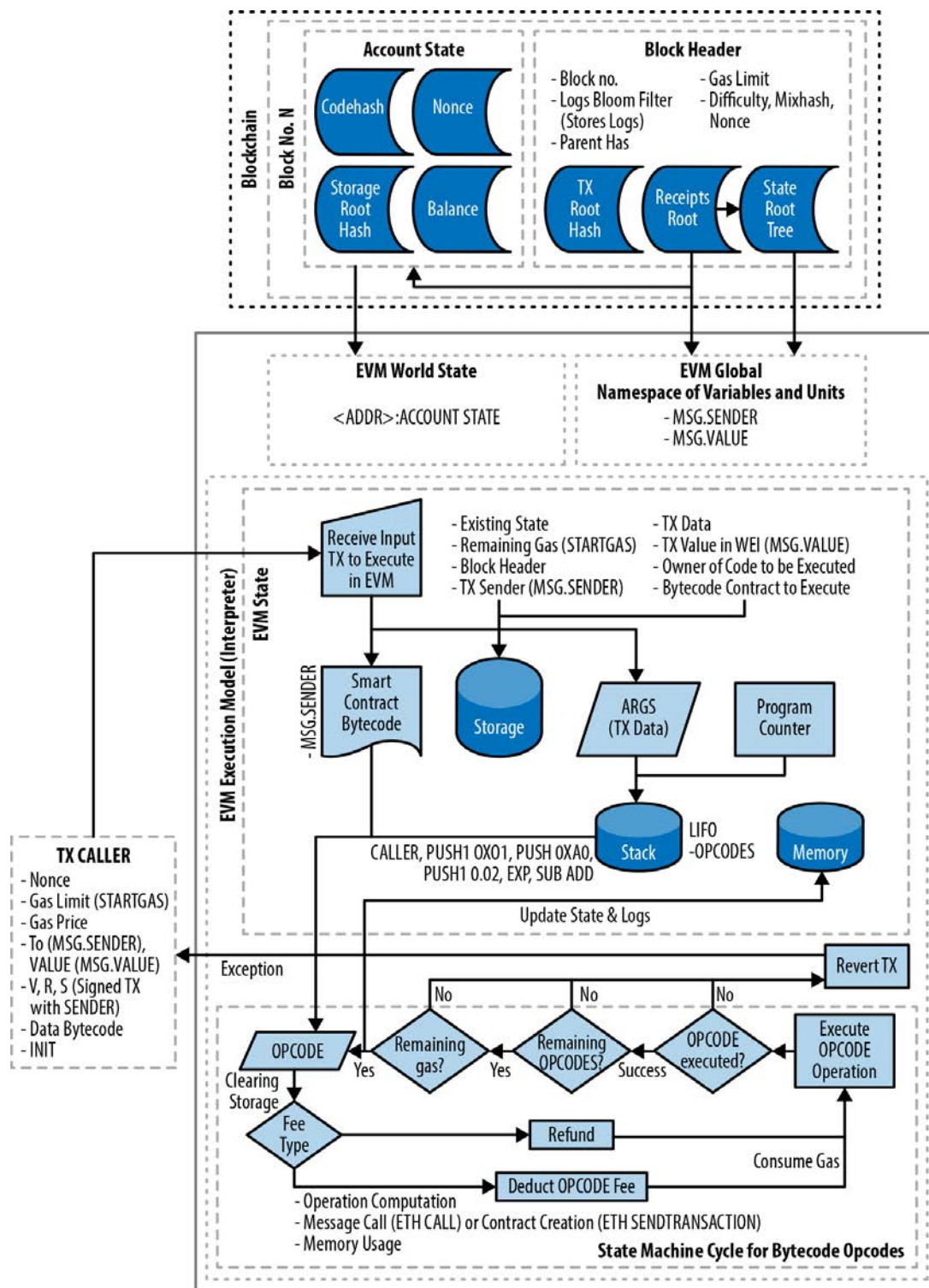
Code execution

- Every (full) node on the blockchain processes every transaction and stores the entire state



Data-structure of the Ethereum Blockchain





Architecture of the Ethereum Virtual Machine (EVM) and its Code Execution Context

DoS Attack Vector

- The “Halting Problem”
 - Cannot tell whether or not a program will run infinitely
- A malicious miner can DoS attack full nodes by including lots of computation in their txs
 - Full nodes attacked when verifying the block

```
uint i = 1;
while (i++ > 0) {
    donothing();
}
```


Solution: “Gas”

- Charge fee per computational step (“gas”)
 - Special gas fees for operations that take up storage

operation	gas cost	description
ADD	3	Add the top 2 items on stack
SUB	3	The 1st item subtracts the 2nd item on stack
AND	3	Bitwise AND operation of the top 2 items on stack
OR	3	Bitwise OR operation of the top 2 items on stack
LT	3	Whether the 1st item is smaller than the 2nd item on stack
EQ	3	Whether the top 2 items on stack are equal
ADDRESS	2	Get the address of current account
ORIGIN	2	Get the address of transaction sender
JUMP	8	Unconditional jump
JUMPI	10	Conditional jump
PUSH1	3	Place 1 byte item on stack
POP	2	Remove 1 item from stack
MLOAD	3	Load a word from memory
MSTORE	3	Save a word to memory
SLOAD	200	Load a word from storage
SSTORE	20,000/ 5,000	Save a word to storage
CREATE	32,000	Create a smart contract
CALL	700	Call a smart contract

Gas in Ethereum is a necessary evil

- All miners and full nodes must evaluate all transactions
 - limit computation cost
- All miners must store all state
 - limit storage use
- Short-cut the halting problem
 - There is an upper GAS_LIMIT, so all programs will halt

Sender has to pay for the gas

- **gasprice**: amount of Ether (ETH) per unit gas
- **startgas**: maximum gas consumable
 - If **startgas** is less than needed
 - Out of gas exception, revert the state as if the TX has never happened **BUT this failed TX will still be recorded in the Ethereum blockchain**
 - Sender still pays all the gas
- TX fee = gasprice * consumedgas
- Gas limit: similar to block size limit in Bitcoin
 - Total gas spent by all transactions in a block < Gas Limit

All transactions specify `START_GAS`, `GAS_PRICE`

1. If $\text{START_GAS} \times \text{GAS_PRICE} > \text{caller.balance}$, halt
2. Deduct $\text{START_GAS} \times \text{GAS_PRICE}$ from `caller.balance`
3. Set $\text{GAS} = \text{START_GAS}$
4. Run code, deducting from `GAS`
5. For negative values, add to `GAS_REFUND`
 - a. `GAS` only decreases
6. After termination, add `GAS_REFUND` to `caller.balance`

Back of envelope numbers a Solidity programmer should know

Average gas price (as of March '18): <https://etherscan.io/chart/gasprice>

<http://ethgasstation.info/>

~20gigawei = 0.00000002 ether

Price of Ether (as of March '18):

~\$500 per Ether

Cost per transaction:

21000 gas "base" for a transaction = \$0.21 **(21 cents per transaction)**

Cost of data:

~75 gas per byte of data stored = \$0.77/kB **(77 cents per kilobyte)**

Gas limit per block: 4,000,000 \Rightarrow 53 kilobytes per block (2.66MB per 10 min)

<http://ethgasstation.info/>

Recommended Gas Prices

(based on current network conditions)

Speed	Gas Price (gwei)
SafeLow (<30m)	2
Standard (<5m)	3
Fast (<2m)	20

Polite contracts call `revert` on errors

```
uint8 numCandidates;
uint32 votingFee;
mapping(address => bool) hasVoted;
mapping(uint8 => uint32) numVotes;

/// Cast a vote for a designated candidate
function castVote(uint8 candidate) {
    if (msg.value < votingFee)
        return;
    if (hasVoted[msg.sender])
        revert();

    hasVoted[msg.sender] = true;
    numVotes[candidate] += 1;
}
```

`revert()` ensures no effects persisted except gas consumption

Out-of-gas exceptions are bad news



- State reverts to previous value
 - Except that $START_GAS * GAS_PRICE$ is still deducted

Built-in support for calling other contracts

- `a.transfer(x)` sends `x` to address `a`
 - returns 0 if this fails due to call stack
- `foo.call.value(3).gas(20764)(bytes4(sha3("bar()")));`
 - also `callcode`, `delegatecall`
 - default is 0 value, all available gas
- `new` constructor deploys a new contract
 - Careful, it's expensive!

Remember:

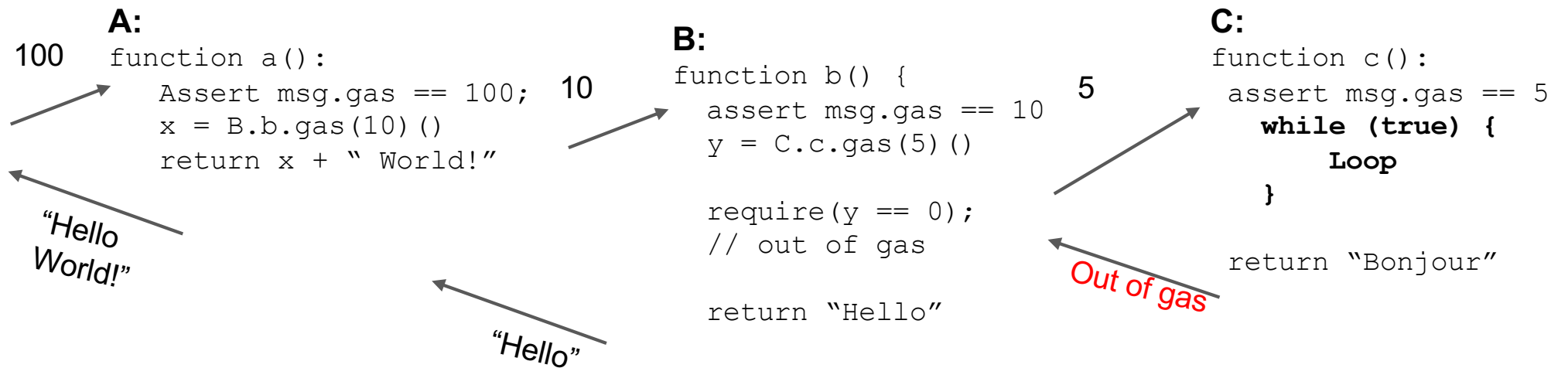
Smart contracts code is fixed *forever*.
Calls required to update functionality

Built-in support for calling other contracts (cont'd)

- Contract member variables `public` if public, automatically defines a “getter”
- Modifiers `payable, constant, returns()`, also modifiers can be user defined
- Macros / Internal Functions `internal` modifier -> does not require a “message”
- Type conversions `int(x), uint256(x), bool(x)`
- Structs, arrays, mappings, memory vs storage

```
array: int[2] x; hashmap mapping ( int[2] => int );
```
- Throwing exceptions `throw;` // exceptions
contain no data
- Units (currency: “eth”, “wei”, etc.) `3 * (2 eth)`

Callers can choose how much gas to send



(pseudocode: exact syntax used here does not work in Solidity)

Economics of gas are similar to transaction fees

- Miners choose transactions based on GAS_PRICE
- In theory, they should not care which opcodes are used
 - In practice, some “overpriced” opcodes may be preferred
- Maximum gas limit per block
 - Miners can slowly raise it, each block votes

References for Solidity syntax

- **Storage and stateful methods**

public instance variables, public methods

constant, pure, view methods

<https://solidity.readthedocs.io/en/v0.4.21/contracts.html#functions>

<https://solidity.readthedocs.io/en/v0.4.21/contracts.html#visibility-and-getters>

- **Control flow, for loops and if statements**

<https://solidity.readthedocs.io/en/v0.4.21/control-structures.html#control-structures>

References for Solidity syntax (cont'd)

- Events and printf debugging

event Debug(string); ... emit Debug("fail at point A")

<https://solidity.readthedocs.io/en/v0.4.21/contracts.html#events>

Debugging strategies:

1. Use Log Events
2. Use pure functions
3. Use the Remix debugger

References for Solidity syntax (cont'd)

- Declaring variables and conversion between types

<https://solidity.readthedocs.io/en/v0.4.21/types.html#value-types>

<https://solidity.readthedocs.io/en/v0.4.21/types.html#conversions-between-elementary-types>

- Integers can overflow <https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/math/SafeMath.sol>

- Mappings

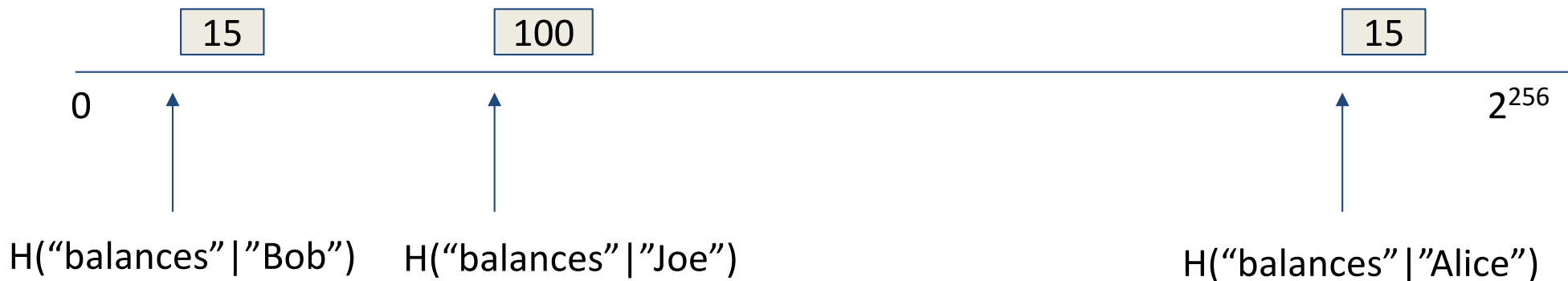
mapping (address => bool) hasAlreadyVoted;

<https://solidity.readthedocs.io/en/v0.4.21/types.html#mappings>

Clever implementation of maps in Solidity

```
mapping(string => uint256) balances;
```

Alice	15
Bob	15
Joe	100



- Every item requires at least one 256-bit word
- Balances["Andrew"] is 0 if "Andrew" doesn't exist or if "Andrew" has 0 balance
- To delete a key, set balances["Andrew"] = 0
- Cannot delete an entire map!

References for Solidity syntax (cont'd)

- Payable and transferring currency

address x; x.transfer(msg.value)

<https://solidity.readthedocs.io/en/v0.4.21/types.html#address>

<https://solidity.readthedocs.io/en/v0.4.21/units-and-global-variables.html#address-related>

- Arrays in storage and in memory

<https://solidity.readthedocs.io/en/v0.4.21/types.html#reference-types>

<https://solidity.readthedocs.io/en/v0.4.21/types.html#arrays>

References for Solidity syntax (cont'd)

- Calling methods of other contracts, the Gas model

Tx.gas, gasLeft()

<https://solidity.readthedocs.io/en/v0.4.21/units-and-global-variables.html#special-variables-and-functions>

- Extern / abstract contracts

<https://solidity.readthedocs.io/en/v0.4.21/contracts.html#abstract-contracts>

- Access controls

msg.sender, tx.origin

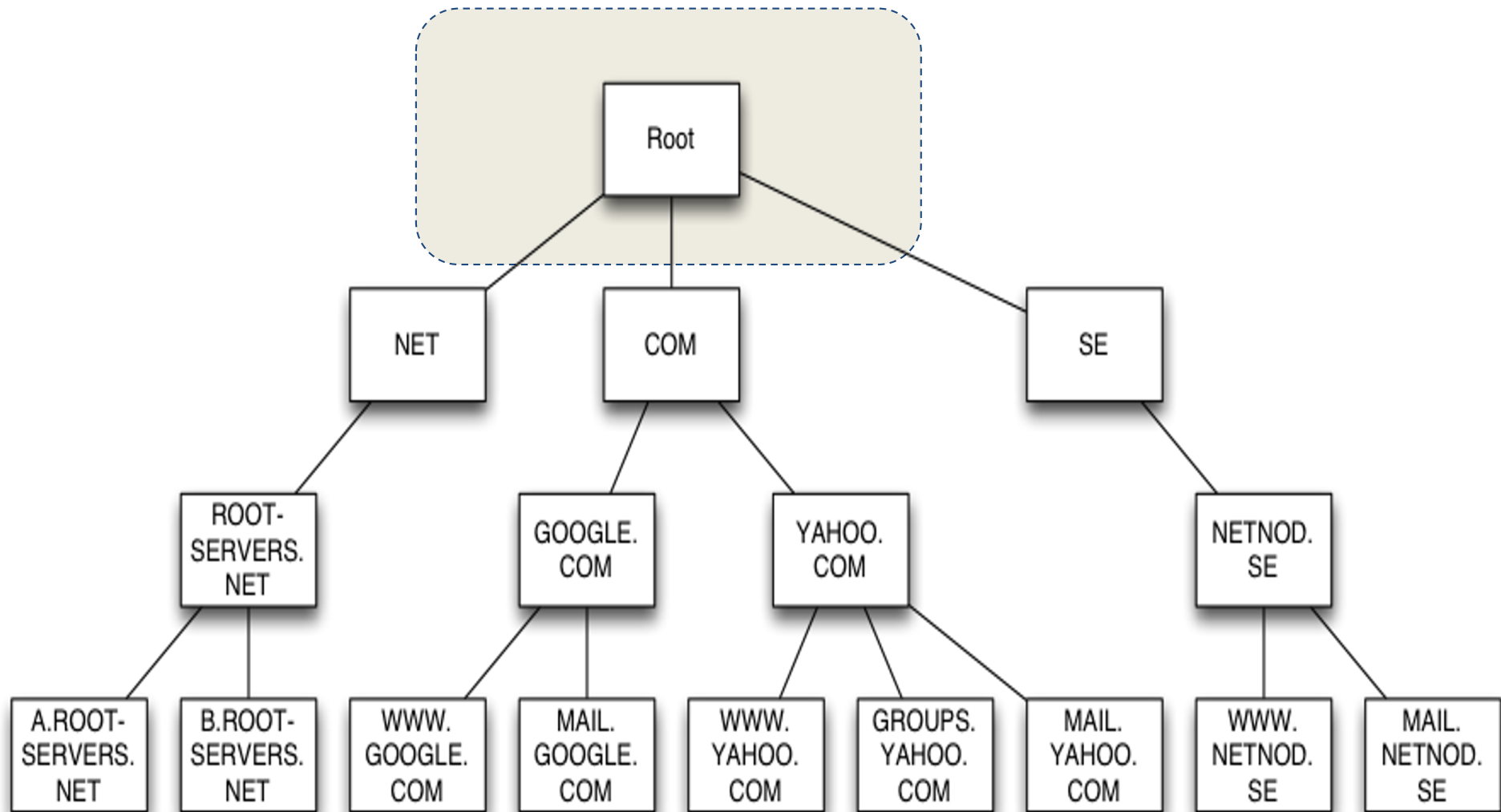
<https://solidity.readthedocs.io/en/v0.4.21/common-patterns.html?highlight=payable#restricting-access>

Solidity gotchas (Many more !!)

- Member variables can be marked `public`
 - Getter methods automatically provided
- Functions must be marked `payable` to accept funds
- Member variables go to storage by default
 - Method variables go to memory
- Fallback function `function()`
 - Called if no function specified (e.g. `send`)
 - Called if non-existent function called
- `msg.sender` vs. `tx.origin`

An Example:
Namecoin in Ethereum

What is the Domain Name System?



“Namecoin”: a simplistic DNS replacement

Initially, all names are unregistered.

Anyone can claim an unregistered name.

Once it's registered, no one can change it.

Namecoin pseudocode

```
def register(k, v):  
    if !self.storage[k]: # Is the key not yet taken?  
        # Then take it!  
  
        self.storage[k] = v  
  
        return(1)  
  
    else:  
  
        return(0) // Otherwise do nothing
```

```
contract Namespace {  
  
    struct NameEntry {  
        address owner;  
        bytes32 value;  
    }  
  
    uint32 constant REGISTRATION_COST = 100;  
    uint32 constant UPDATE_COST = 10;  
    mapping(bytes32 => NameEntry) data;  
  
    function nameNew(bytes32 hash){  
        if (msg.value >= REGISTRATION_COST){  
            data[hash].owner = msg.sender;  
        }  
    }  
  
    function nameUpdate(bytes32 name, bytes32 newValue, address newOwner){  
        bytes32 hash = sha3(name);  
        if (data[hash].owner == msg.sender && msg.value >= UPDATE_COST){  
            data[hash].value = newValue;  
            if (newOwner != 0){  
                data[hash].owner = newOwner;  
            }  
        }  
    }  
  
    function nameLookup(bytes32 name){  
        return data[sha3(name)];  
    }  
  
}
```

Key challenges in Smart Contract Design and Implementation:

- Smart contracts on public blockchains can be trusted for **correctness** and **availability**, but not **privacy (yet)**
- Blockchain resources are **expensive**
- On the blockchain, “**Code is law**”

****Uncertain delays, Race conditions (e.g. Front Running), Temporary Forks, ...**

****Obscure and counterintuitive VM rules**

=> ****** can be (and have been) exploited to cause Smart Contract “Security” vulnerabilities which resulted in **substantial monetary losses !**

Some Security Flaws

(Many more a Solidity Programmer needs to know !)

- Due to abstraction of semantic
 - Transaction ordering dependence
 - Reentrancy bug
 - Which exploited the DAO
- Obscure VM rules
 - Maximum stack depth is 1024: not many devs know
 - Inconsistent Exception Handling in EVM



The DAO Attacked: Code Issue Leads to \$60 Million Ether Theft

Michael del Castillo (@DelRayMan) | Published on June 17, 2016 at 14:00 GMT

NEWS

For details, refer to:

<https://dasp.co>

<https://www.cryptocompare.com/coins/guides/the-dao-the-hack-the-soft-fork-and-the-hard-fork/>

The Reentrancy Bug which stole 3.5M ETH from DAO ; Led to a Hard Fork and Split of Ethereum blockchain [Ethereum Classic (ETC) was born] (circa June-July 2016)

Example:

1. A **smart contract** tracks the balance of a number of external addresses and allows users to retrieve funds with its public `withdraw()` function.
2. A **malicious smart contract** uses the `withdraw()` function to retrieve its entire balance.
3. The **victim contract** executes the `call.value(amount)()` **low level function** to send the ether to the **malicious contract before updating the balance of the malicious contract.**
4. The **malicious contract** has a payable `fallback()` function that accepts the funds and then calls back into the **victim contract's** `withdraw()` function.
5. This second execution triggers a transfer of funds: remember, the balance of the **malicious contract** still hasn't been updated from the first withdrawal. As a result, the **malicious contract** successfully withdraws its entire balance a second time.

```
function withdraw(uint _amount) {  
    require(balances[msg.sender] >= _amount);  
    msg.sender.call.value(_amount)();  
    balances[msg.sender] -= _amount;  
}
```

For details, refer to:

<https://dasp.co>

<https://www.cryptocompare.com/coins/guides/the-dao-the-hack-the-soft-fork-and-the-hard-fork/>

Ongoing Efforts to mitigate Security Flaws

- Create developer tools
 - Smart contract analyser based on symbolic exec: [Oyente](#)
 - Testing and deployment framework: [truffle](#)
 - Formal verification for smart contracts: [eth-isabelle](#), [why3](#)
- Design better semantic [CCS'16]
- Educate users
- Idea
 - Create security certificates for smart contracts?

**A Popular and Serious Application of
Ethereum Smart Contracts:
Initial Coin Offering (ICO)**

ERC20 Tokens

- A Token Contract is a smart contract that contains **a map of account address and their balances**
 - The **balance represents a value** that is defined by the contract creator, e.g. the value may represent physical objects, another monetary value or the holder's reputation.
 - This balance is commonly called a **token**.

- ERC-20 defines a common set of features and interfaces for Token Contracts in Ethereum

NB: ERC = Ethereum Request for Comments

Holder address	Balance
0x0000...0000	0
0x1f59...3492	110
0x2299...3ab7	90
0x4ba5...ae22	100
0x4919...413d	100
0x93f1...1b09	100
0xd8f0...c028	100
0xe20b...93b6	100
Total supply	700

Transfer of 10 tokens from 0x2299...3ab7 to 0x1f59...3492; changes shown in red

ERC20 defines interfaces for basic token behavior

<https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md>

Basic functionality:

```
function totalSupply() constant returns (uint256 totalSupply)
```

```
function balanceOf(address _owner) constant returns (uint256 balance)
```

Delegating control:

```
function transfer(address _to, uint256 _value) returns (bool success)
```

```
function transferFrom(address _from, address _to, uint256 _value) returns (bool success)
```

Delegating control:

```
function approve(address _spender, uint256 _value) returns (bool success)
```

```
function allowance(address _owner, address _spender) constant returns (uint256  
remaining)
```

Refer to:

<https://github.com/ethereumbook/ethereumbook/blob/develop/10tokens.asciidoc>
for more details



Search for Account, Tx Hash or Data

ERC20 Standard Token Explorer

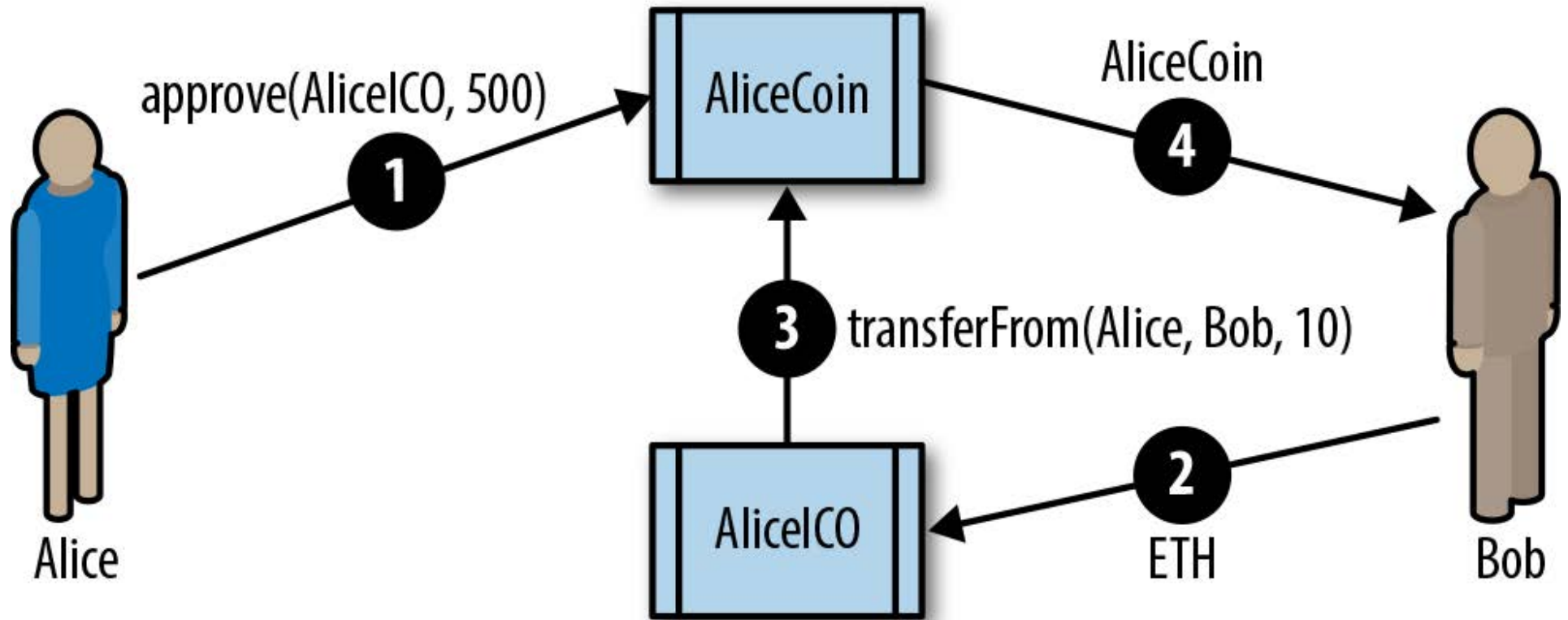
[Home](#) / [TokenTracker](#) / [Search](#)

The TOKEN TRACKER can search, discover and track all contracts that conform to the [ERC20 Token Standard](#)

Please enter the contract address you wish to search for above.

Note: To set your token's divisor value, name and logo (28x28px PNG format) please verify your contract source and [provide us](#) with a link to your official site. We will also use the token's '**decimals**' value if it is present in the contract. For an example see <https://ethereum.org/token>

Two-Step Approve & TransferFrom Workflow of Initial Coin Offering (ICO) of an ERC20 Token



Two-Step Approve & TransferFrom Workflow of Initial Coin Offering (ICO) of an ERC20 Token (cont'd)

For the approve & transferFrom workflow, two transactions are needed. Let's say that Alice wants to allow the AliceICO contract to sell 50% of all the AliceCoin tokens to buyers like Bob and Charlie. First, Alice launches the AliceCoin ERC20 contract, issuing all the AliceCoin to her own address. Then, Alice launches the AliceICO contract that can sell tokens for ether. Next, Alice initiates the approve & transferFrom workflow. She sends a transaction to the AliceCoin contract, calling approve with the address of the AliceICO contract and 50% of the totalSupply as arguments. This will trigger the Approval event. Now, the AliceICO contract can sell AliceCoin.

When the AliceICO contract receives ether from Bob, it needs to send some AliceCoin to Bob in return. Within the AliceICO contract is an exchange rate between AliceCoin and ether. The exchange rate that Alice set when she created the AliceICO contract determines how many tokens Bob will receive for the amount of ether sent to the AliceICO contract. When the AliceICO contract calls the AliceCoin transferFrom function, it sets Alice's address as the sender and Bob's address as the recipient, and uses the exchange rate to determine how many AliceCoin tokens will be transferred to Bob in the value field. The AliceCoin contract transfers the balance from Alice's address to Bob's address and triggers a Transfer event. The AliceICO contract can call transferFrom an unlimited number of times, as long as it doesn't exceed the approval limit Alice set. The AliceICO contract can keep track of how many AliceCoin tokens it can sell by calling the allowance function.

Cryptocurrency Market Capitalizations









All ▾

Coins ▾

Tokens ▾

USD ▾

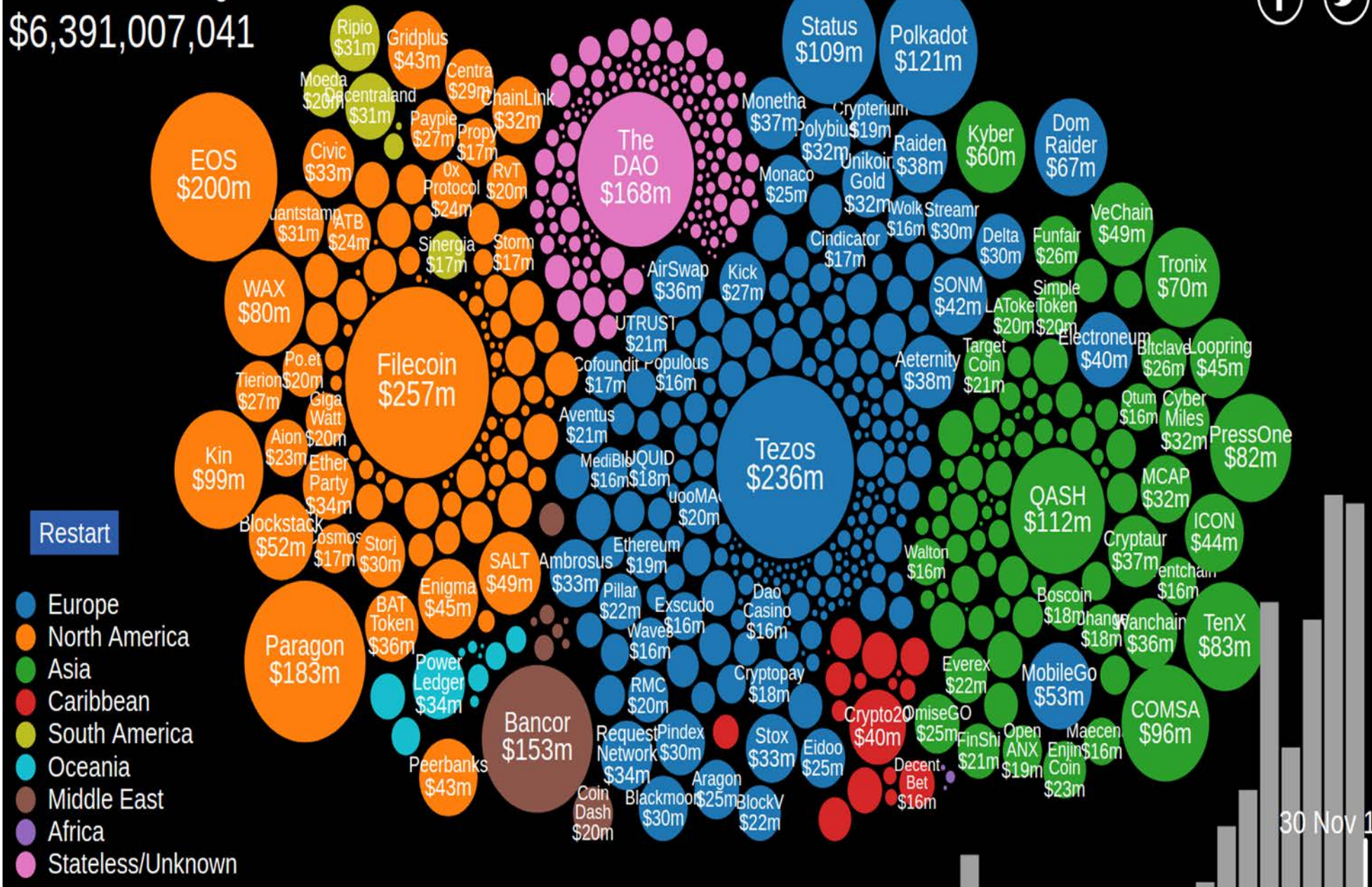
[← Back to Top 100](#)

#	Name	Platform	Market Cap	Price	Circulating Supply	Volume (24h)	% 1h	% 24h	% 7d
1	 EOS	Ethereum	\$4,452,467,817	\$5.80	767,354,631	\$309,579,000	0.63%	2.41%	1.73%
2	 Tether	Omni	\$2,292,973,023	\$1.00	2,287,140,814	\$1,537,220,000	0.11%	-0.01%	0.12%
3	 TRON	Ethereum	\$2,252,388,428	\$0.034258	65,748,192,476	\$225,229,000	0.35%	6.56%	-17.96%
4	 Rinace Coin	Ethereum	\$1,434,935,596	\$12.34	116,261,604	\$113,461,000	0.84%	11.30%	7.17%
<hr/>									
678	 Musiconomi	Ethereum	\$?	\$0.031992	?	\$0	0.55%	1.74%	-7.51%
679	 SpherePay	Ethereum	\$?	\$0.002235	?	\$0	?	?	?
680	 Nexus	Ethereum	\$?	\$0.008240	?	\$?	?	?	-85.68%
681	 Monero Gold	Ethereum	\$?	\$0.000185	?	\$?	?	?	?

Total Market Cap: \$30,011,931,539

Four Years of Initial Coin Offerings i

Total fundraising
\$6,391,007,041



30 Nov 17

Statement on Cryptocurrencies and Initial Coin Offerings

SEC Chairman Jay Clayton

Dec. 11, 2017

The world's social media platforms and financial markets are abuzz about cryptocurrencies and "initial coin offerings" (ICOs). There are tales of fortunes made and dreamed to be made. We are hearing the familiar refrain, "this time is different."

SEC Halts Fraudulent Scheme Involving Unregistered ICO

FOR IMMEDIATE RELEASE

2018-53

Washington D.C., April 2, 2018 — The Securities and Exchange Commission today charged two co-founders of a purported financial services start-up with orchestrating a fraudulent initial coin offering (ICO) that raised more than \$32 million from thousands of investors last year. Criminal authorities separately charged and arrested both defendants.

Lifecycle of an ICO Pre-launch Token

Presale / private investment

Whitepaper released

Public crowdsale

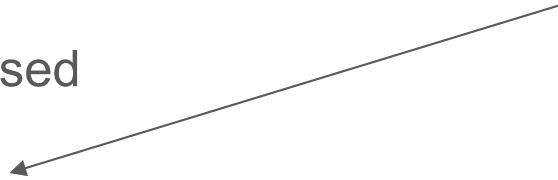
Optionally: tokens can be traded on exchanges

Development continues...

Product launches, you can use your tokens

Most likely need to register with SEC!

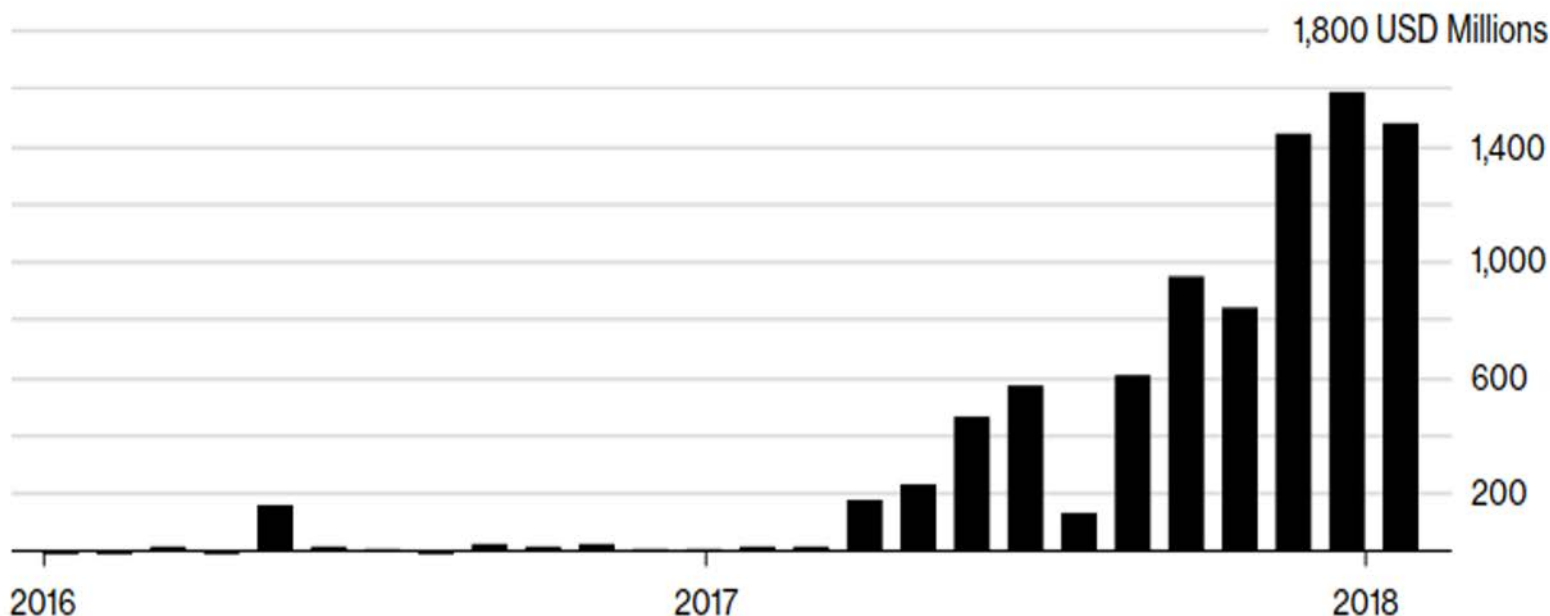
Typically implemented with ERC20



What Clampdown?

Initial coin offering funding is soaring even as regulators tighten grasp

■ Monthly ICO funding



Source: CoinDesk

In 2017, 46 percent of token startups either failed after their ICOs or weren't able to complete funding, according to Bitcoin.com, which drew on statistics gathered by website TokenData. So far this year, 50 of 340 completed ICOs have failed, according to TokenData.

<https://www.bloomberg.com/news/articles/2018-04-02/crypto-hedge-fund-bubble-begins-to-deflate-as-returns-tumble>

<https://news.bitcoin.com/46-last-years-icos-failed-already/>

Interested in Dissecting the Useless Ethereum Token ?



<https://uetoken.com/>

<https://etherscan.io/address/0x27f706edde3aD952EF647Dd67E24e38CD0803DD6#code>

You can also build your own tETH-backed ERC20
Token:

<https://ropsten.etherscan.io/token/0xEFeaEF27c453eB96AEa340d03E1724B81973cD61#balances>

Many ERC20 templates on the Internet

This is a widely adopted standard, and so tons of tools/service will
???"just work"???" if you adhere to ERC20 standard

**Beware: any bug for a popular template can jeopardize all smart
contracts/ ICOs of that template !!**

<http://imgtfy.com/?q=erc20+token+template>

[https://github.com/bitfwdcommunity/Issue-your-own-ERC20-
token/blob/master/contracts/erc20_tutorial.sol](https://github.com/bitfwdcommunity/Issue-your-own-ERC20-token/blob/master/contracts/erc20_tutorial.sol)

The Short Address Attack on ERC20 token exchanges

Several online exchanges allow you to transfer tokens from a web form

Assumption: All interactions with contracts respects function interface

```
function transfer(address _to, uint256 _value) returns (bool success)
```

Reality: compiled EVM interprets all data as array of 32-byte blocks

Attack: attacker finds a key for an address of the form `0xAAAAAAAA...AAAAA000` and types the 17-byte prefix into the `#userAddr` field

0000...0000 **AAAAAAAA...AAAAA000** 00000...0000000055000

Example: 0000...0000 **AAAAAAAA...AAAAA000** 00000000...0000000055

```
var callData = "0"*12 + ($(#userAddr).val()) + toHex($(#amt).val(), 32)
```

padding

20 byte address

value (little endian, 32 bytes)

For details, refer to:

<https://blog.golemproject.net/how-to-find-10m-just-by-reading-the-blockchain/>

<https://vessenes.com/the-erc20-short-address-attack-explained/>

https://www.reddit.com/r/ethereum/comments/6r9nhj/cant_understand_the_erc20_short_address_attack/

Many ERC20 Tokens are stuck in Unspendable Contracts

TOP 5 tokens that got stuck inside contracts:

1. 257 221 GNT^[4] inside Golem contract. (\$109 149 lost)
2. 439 GNO^[5] inside Gnosis contract. (\$105 029 lost)
3. 782 DGD^[6] inside DigixDAO contract. (\$62500 lost)
4. 259 335 SNGLS^[7] inside SingularDTV contract. (\$49 008 lost)
5. 222 REP^[8] inside Augur contract. (\$4900 lost)

To be continued...

<https://www.reddit.com/r/ethereum/6e8y9o/>

Yet another Story on ERC20 Token Thefts due to “Integer Overflow” bug

```
255 function batchTransfer(address[] _receivers, uint256 _value) public whenNotPaused returns (bool) {
256     uint cnt = _receivers.length;
257     uint256 amount = uint256(cnt) * _value;
258     require(cnt > 0 && cnt <= 20);
259     require(_value > 0 && balances[msg.sender] >= amount);
260
261     balances[msg.sender] = balances[msg.sender].sub(amount);
262     for (uint i = 0; i < cnt; i++) {
263         balances[_receivers[i]] = balances[_receivers[i]].add(_value);
264         Transfer(msg.sender, _receivers[i], _value);
265     }
266     return true;
267 }
268 }
```

Details available at:

<https://medium.com/@peckshield/alert-new-batchoverflow-bug-in-multiple-erc20-smart-contracts-cve-2018-10299-511067db6536>

<https://news.bitcoin.com/exchanges-suspend-erc20-token-deposits-after-discovery-of-smart-contract-bug/>

Conclusions

- Smart Contract generalizes the use of Blockchain beyond transferring cryptocurrency among different users
- Ethereum is the 1st general purpose programmable platform built to enable blockchain-based Smart Contracts
- A Smart Contract platform enables Decentralized Exchange/ Trading of Digital Assets in an Automated manner
- The notion of “Gas” is introduced in Ethereum to side-step the potential Denial-of-Service attacks due to the “Halting Problem”.
- A lot of serious Gotchas in the design and implementation of Solidity/ Smart Contracts due to System/Programming Language idiosyncrasies + Bugs
 - ◆ Being a popular (ICO) template does not mean its safe !
- If you want to design/ program Smart Contracts for a living, you need to know much more and deeply about Smart Contract Security, Vulnerabilities and Pitfalls

Additional References on Smart Contract Security and Best Practices

- Decentralized Application Security Project Top 10 of 2018, *<https://dasp.co/index.html>*
- Smart Contract Best Practices, *<https://consensys.github.io/smart-contract-best-practices/>*
- SWC Registry: Smart Contract Weakness Classification and Test Cases, *<https://swcregistry.io>*

Recommended Texts/ References on Bitcoin, Ethereum and Smart Contracts

- Satoshi Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” Oct 2008, <https://bitcoin.org/bitcoin.pdf>
- Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, Steven Goldfeder, Bitcoin and Cryptocurrency Technologies - A Comprehensive Introduction, Princeton University Press, 2016, <https://bitcoinbook.cs.princeton.edu>
- Preethi Kasireddy, “How does Ethereum work anyway”, Sept 13, 2017, <https://www.preethikasireddy.com/post/how-does-ethereum-work-anyway>
- Andreas M. Antonopoulos, Mastering Bitcoin, 2nd Edition, Published by O'Reilly, July 2017, <https://github.com/bitcoinbook/bitcoinbook>
- The Ethereum White Paper - A Next-Generation Smart Contract and Decentralized Application Platform, <https://github.com/ethereum/wiki/wiki/White-Paper>
- Andreas M. Antonopoulos, Gavin Wood, Mastering Ethereum - Building Smart Contracts and DApps, Published by O'Reilly, 2018, <https://github.com/ethereumbook/ethereumbook/blob/develop/book.asciidoc>