Algorand

Algorand School

2022

Cosimo Bassi
Solutions Architect at Algorand Inc.

cosimo.bassi@algorand.com

Open Source: github.com/cusma/algorand-school



Algorand

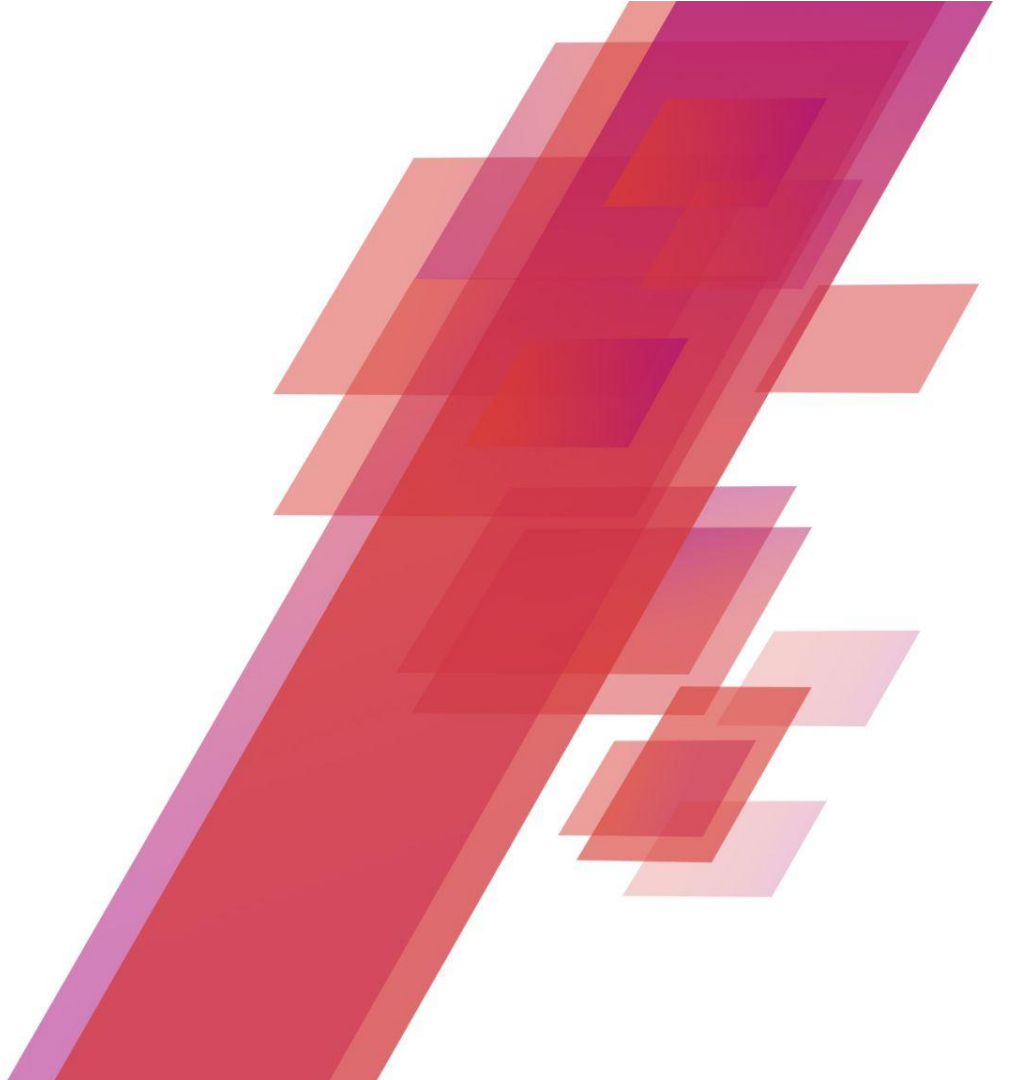
Our journey today:

understanding Algorand Consensus and Algorand Networks, how to use Algorand Dev Tools, how to develop decentralized applications on the Algorand Virtual Machine

Algorand

Agenda

- Blockchain as an infrastructure
- Analog properties for Digital things
- Algorand Consensus
- Algorand Sustainability
- Algorand Decentralized Governance
- Algorand Networks
- Algorand Interoperability
- Algorand Transactions
- Algorand Accounts
- Algorand Standard Assets
- Algorand Virtual Machine
- Algorand Smart Contracts on Layer-1
- Smart Signatures & Smart Contracts
- TEAL
- PyTEAL & ABI & Beaker
- dApp Example & Use Cases

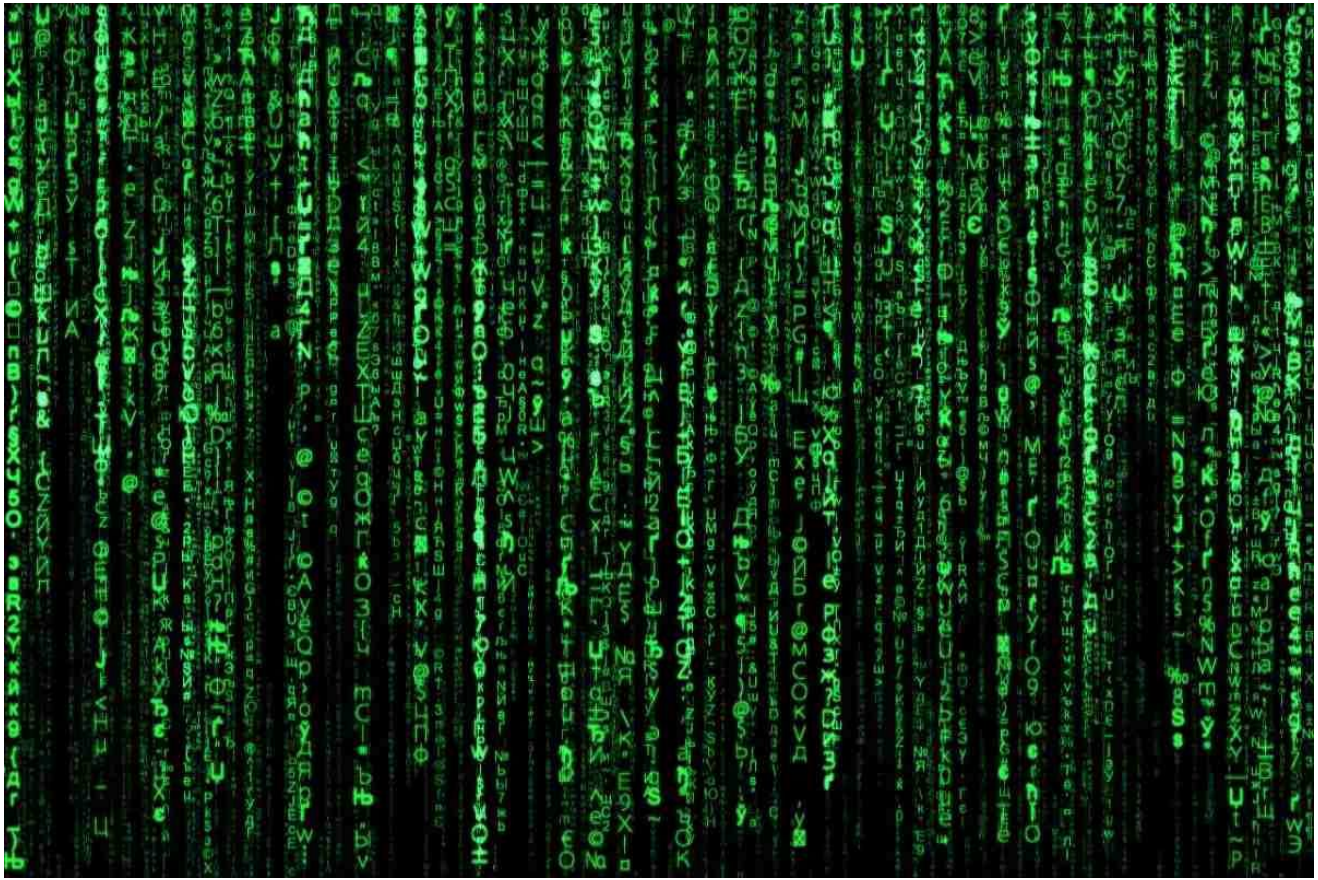


Algorand

BLOCKCHAIN AS INFRASTRUCTURE

For native digital value

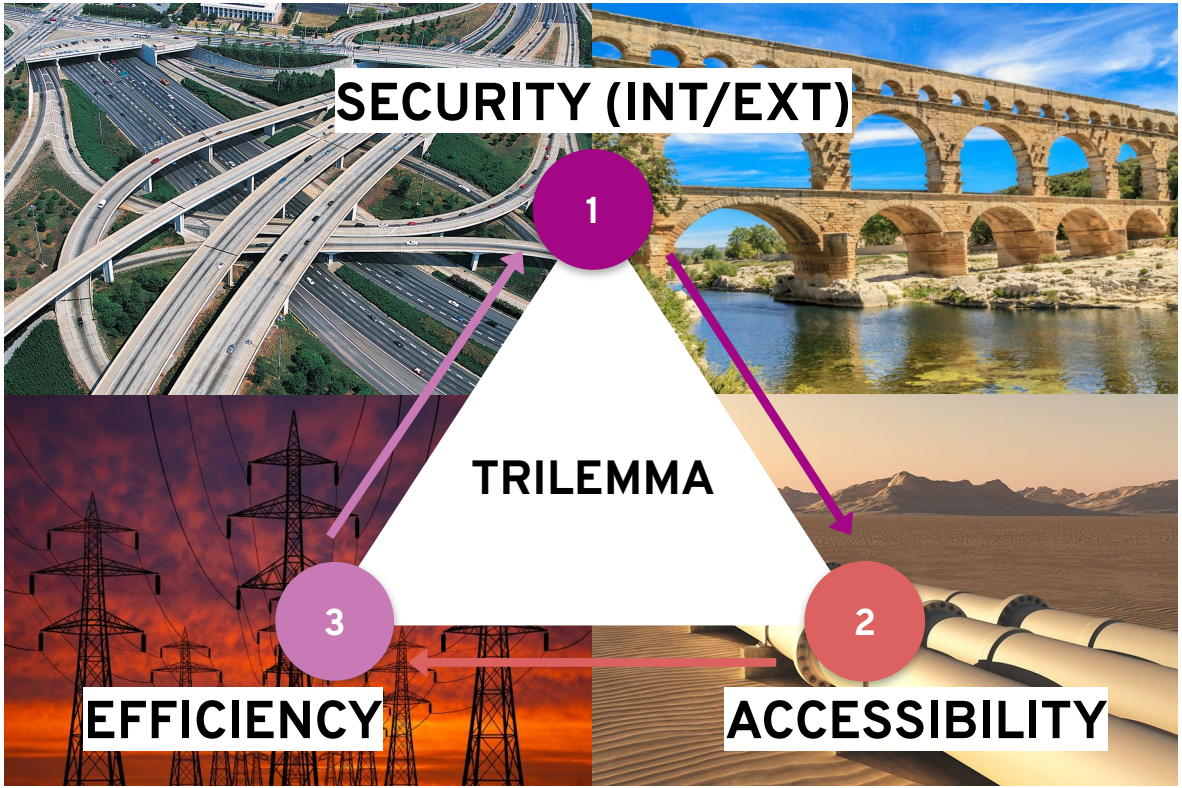
Blockchain *is a digital infrastructure* for value



"The ability to create something which is not duplicable in the digital world has enormous value."

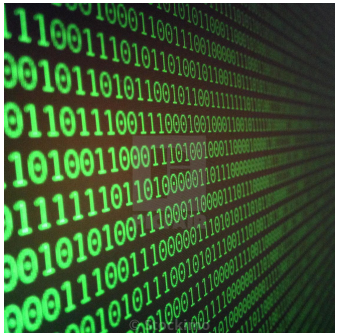
Eric Schmidt

The *infrastructure trilemma*



Public or private? Who has the control? Duty and responsibility? Aligned incentives?

The problem of *native digital value*



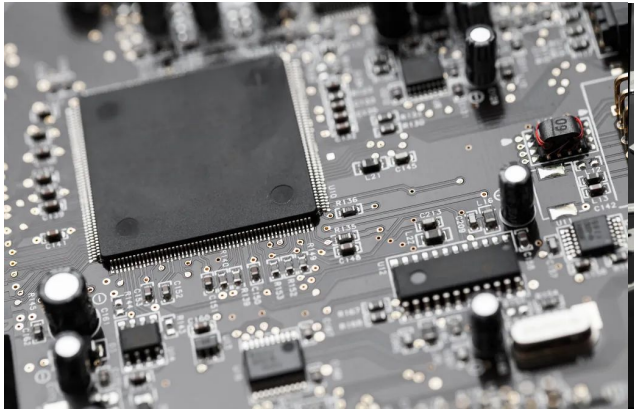
In the digital age
everything can be represented in bits
as a string of 0 and 1

Strings of 0 and 1 are useful
because you can
duplicate them easily

Value is therefore difficult to represent in the digital age:
SCARCITY, AUTHENTICITY, UNICITY

How to build such infrastructure? *Protocol* is the answer

This is not just an *information technology* problem



DISTRIBUTED SYSTEMS

CRYPTOGRAPHY

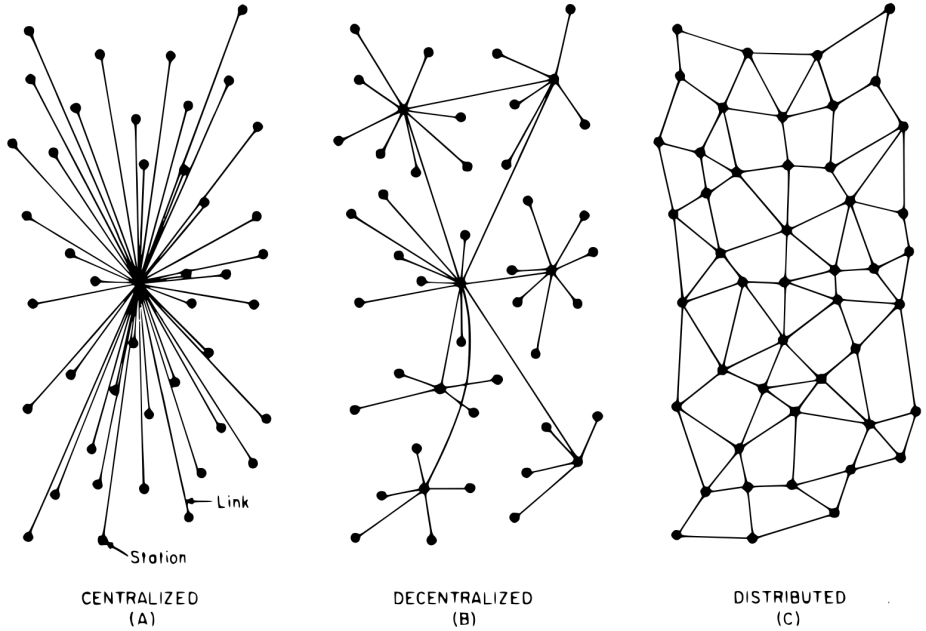
GAME THEORY

PUBLIC
TAMPER-PROOF
TRANSPARENT
TRUSTLESS
LEDGER

Who controls the system? Nobody… everybody!

DISTRIBUTED SYSTEMS

PAST

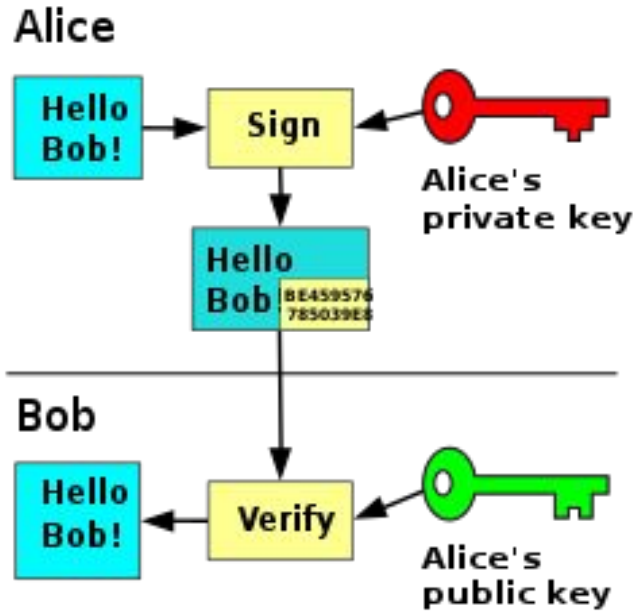


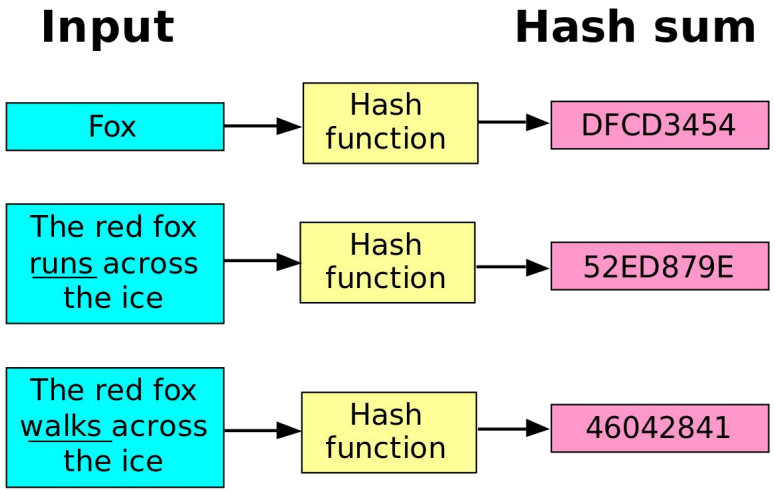
FUTURE

No absolute power, no single point of failure.

Don’t trust, verify!

CRYPTOGRAPHY





Nobody can break the rules…

and

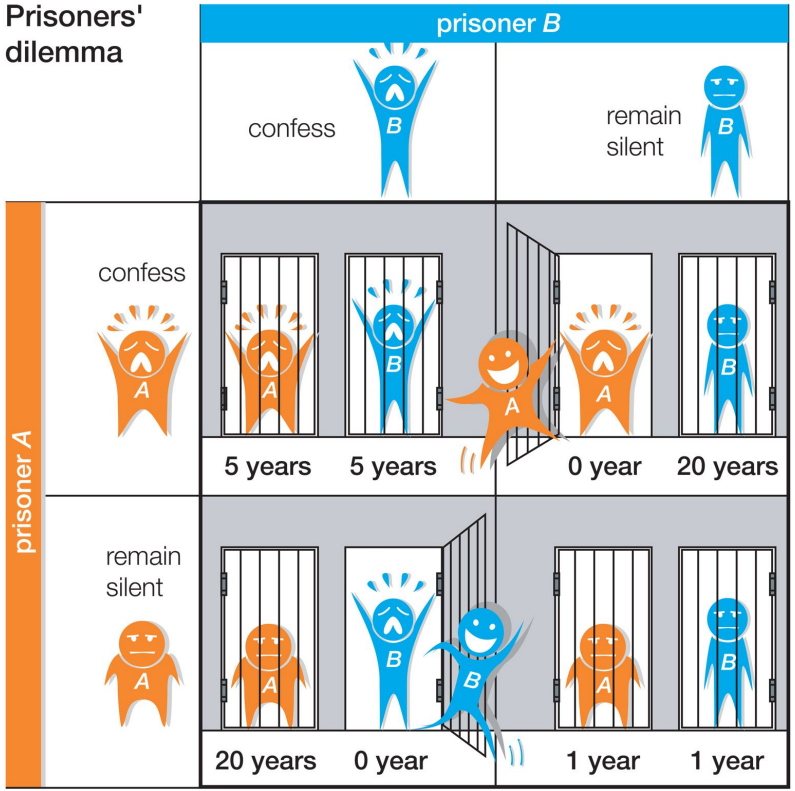
…everybody can verify.

Align incentives: collective self-protecting system

GAME THEORY

Equilibrium in which **attacking** the system is **less convenient than protecting** it.

Cost of the attack: make malicious behaviours expensive.



Algorand

ANALOG PROPERTIES FOR DIGITAL THINGS

Consensus as “law of Physics” for digital world

Who owns what? Let's write it down!

A **blockchain** is a **public ledger** of transactional data.

WHO

OWNS

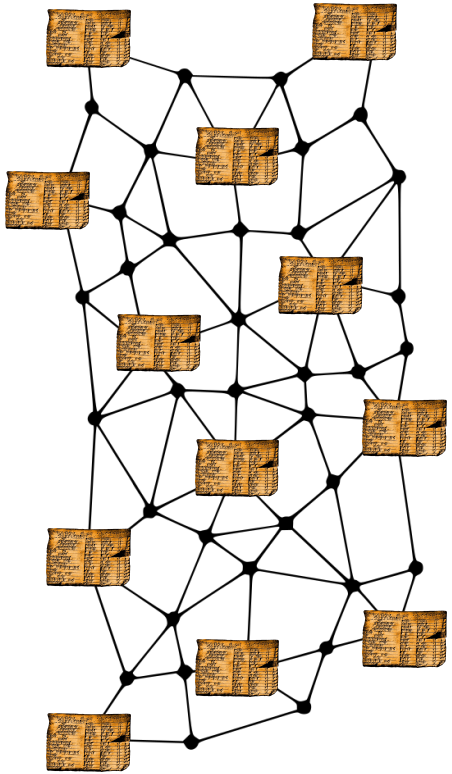
WHAT

World's first writing — cuneiform — traces its beginnings back to an ancient system of accounting.

More copies are better than one!

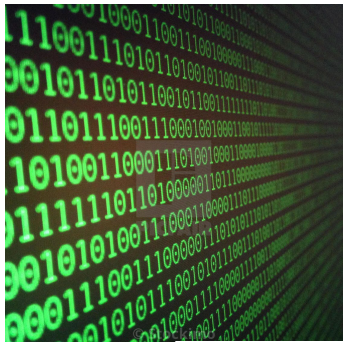
Distributed and **replicated** across a **system of multiple nodes** in a network.

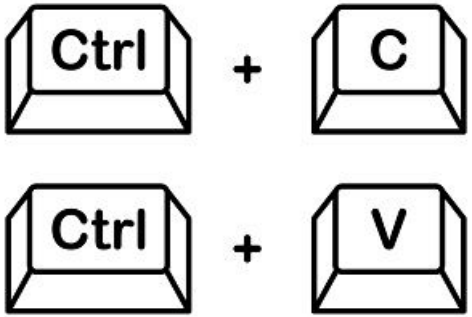
All *"ledger keepers"* should work together, using the **same set rules**, to **verify** the transactions to add to each copy of the finalized ledger.

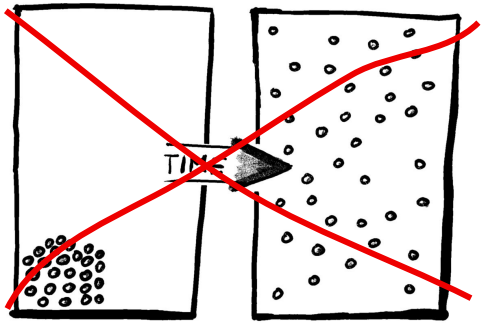


Entropy, irreversibility and the arrow of time

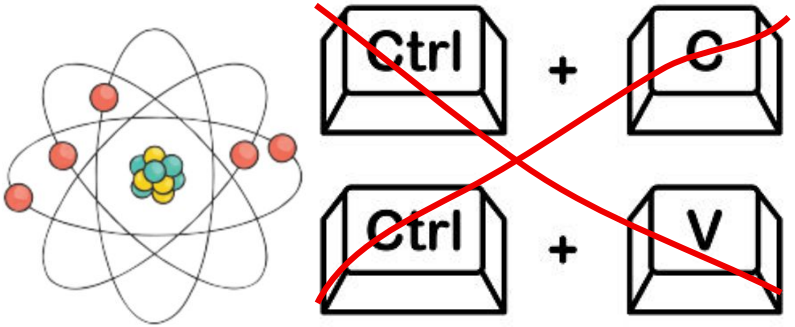
Bits can be copy & pasted and **are not obliged** to follow the **arrow of time**!

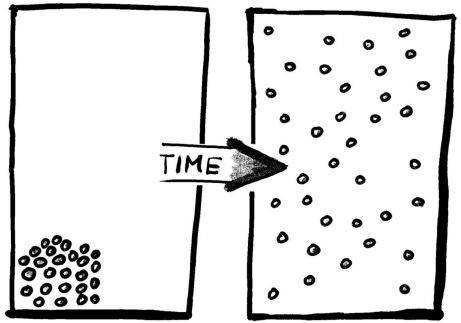






Atoms can't be copy & pasted and **are obliged** to follow the **arrow of time**!

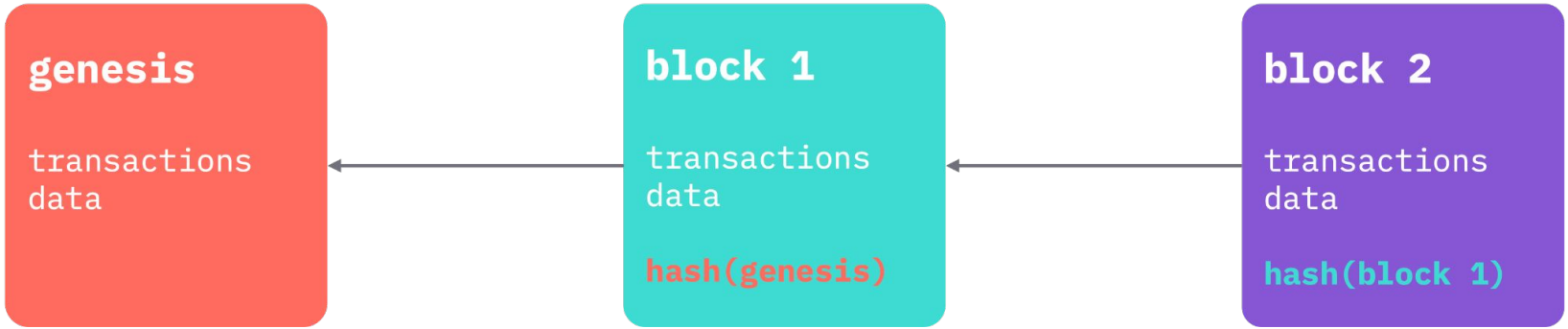




A chain of transactions organized in blocks

The “**block**” refers to a set of transactions that are proposed and verified by the other nodes and eventually added to the ledger (**no copy & paste**).

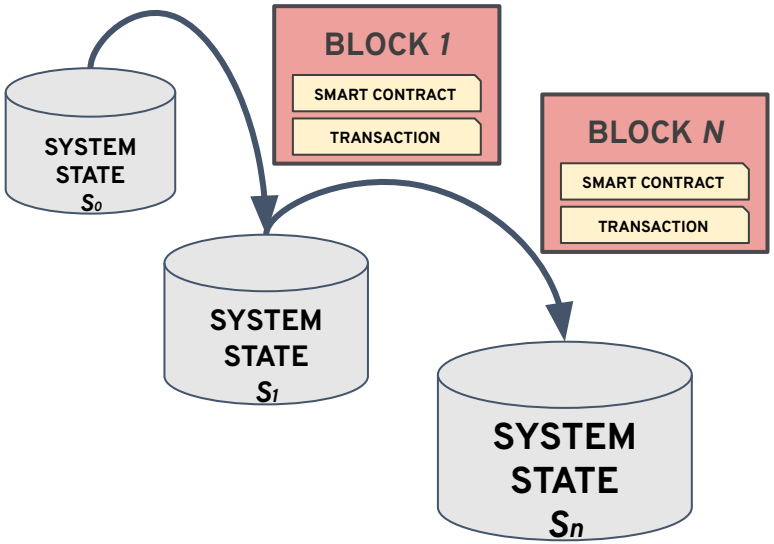
The “**chain**” refers to the fact that each block of transactions contains proof (a cryptographic hash) of what was in the previous block (**arrow of time**).



Can transactions be reverted or modified ex-post? No! *Dura lex, sed lex!*

A distributed state machine

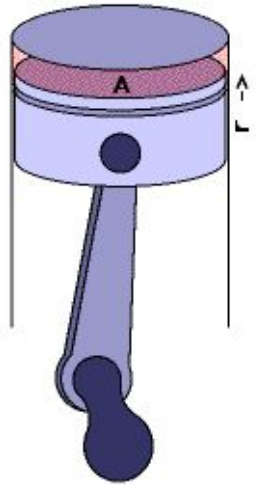
The **evolution** of the **states of the system** is determined…

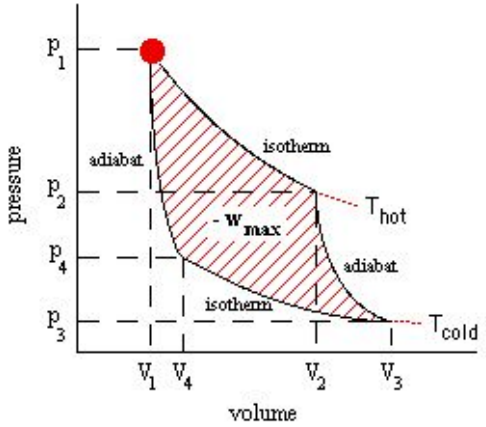


…**acting on bits**, through **what?**

A machine

The **evolution** of the **states of the system** is determined…



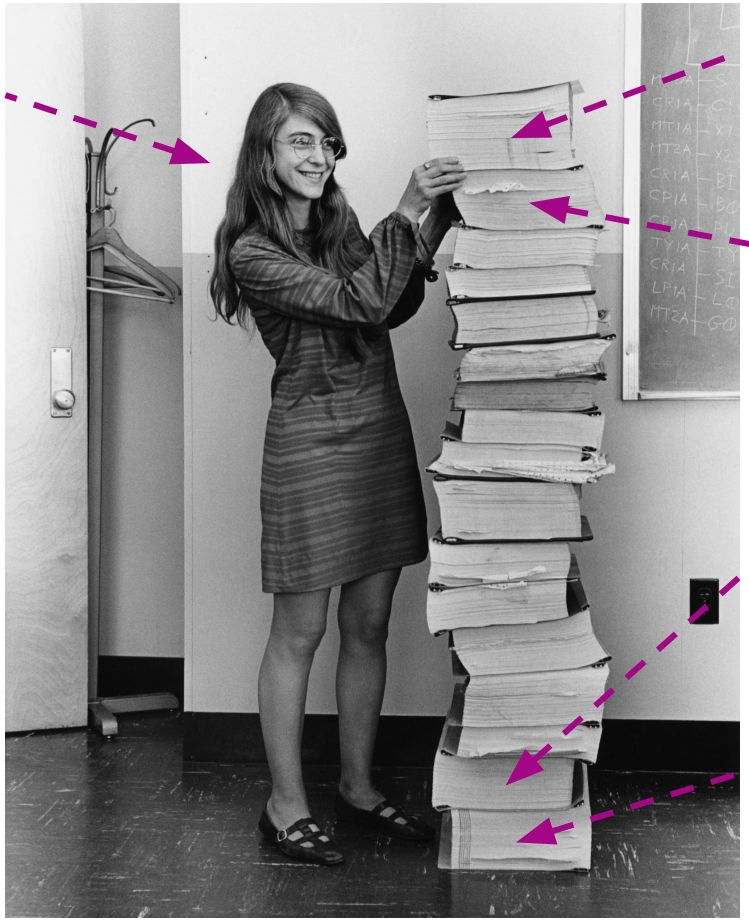


…**acting on atoms**, through the **inviolable laws of Physics!**

The responsibility of correct information

How should we **replace** the role played by the **law of Physics** in evolving the state of a machine?

With a **set of software rules**, called **Consensus Protocol** that evolves the **state** of the system.



Margaret Hamilton in 1969, standing next to listings of the software she and her MIT team produced for the Apollo project.

The architecture of consensus

1. How to choose the **proposer for the next block** for a **public and permissionless** blockchain?

2. How to ensure that **there is no ambiguity** in the choice of the next block?

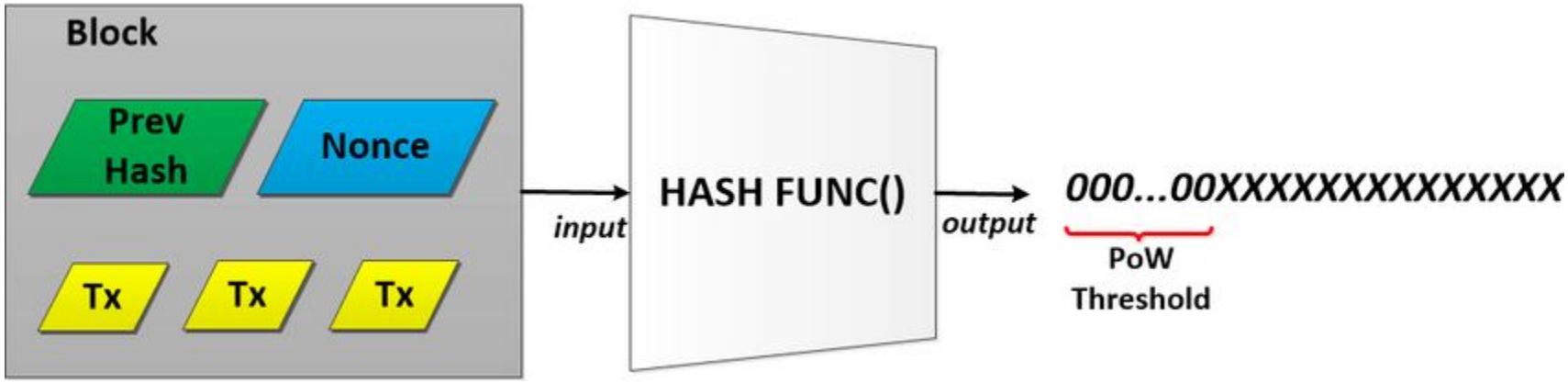
3. How to ensure that the **blockchain stays unique** and has **no forks**?

4. How to ensure that **consensus mechanism itself can evolve** over time while the blockchain is an immutable ledger?

Temple of Concordia

Valley of Temples (Agrigento), 440-430 B.C.

Proof of Work consensus mechanism



Miners **compete with each other** to append the next block and **earn a reward for the effort**, fighting to win an **expensive computational battle**.

The **more computational power**, the **higher the probability** of being elected as block proposer.

Proof of Work limits

- Huge electrical consumption
- Concentration of governance in few mining farms
- Soft-forking of the blockchain

Proof of Stake consensus mechanism

Network participants **show their commitment and interest** in keeping the ledger safe and secure **proving the ownership of value stored on the ledger itself**.

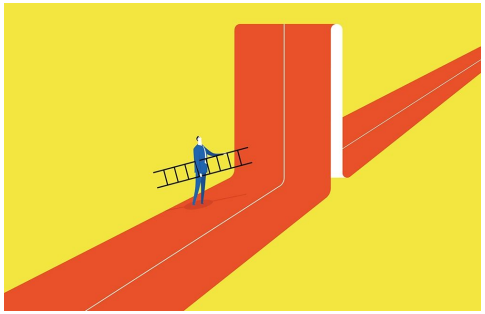
The **higher the skin in the game** the **higher the probability** of being elected as block proposer.

Proof of Stake limits

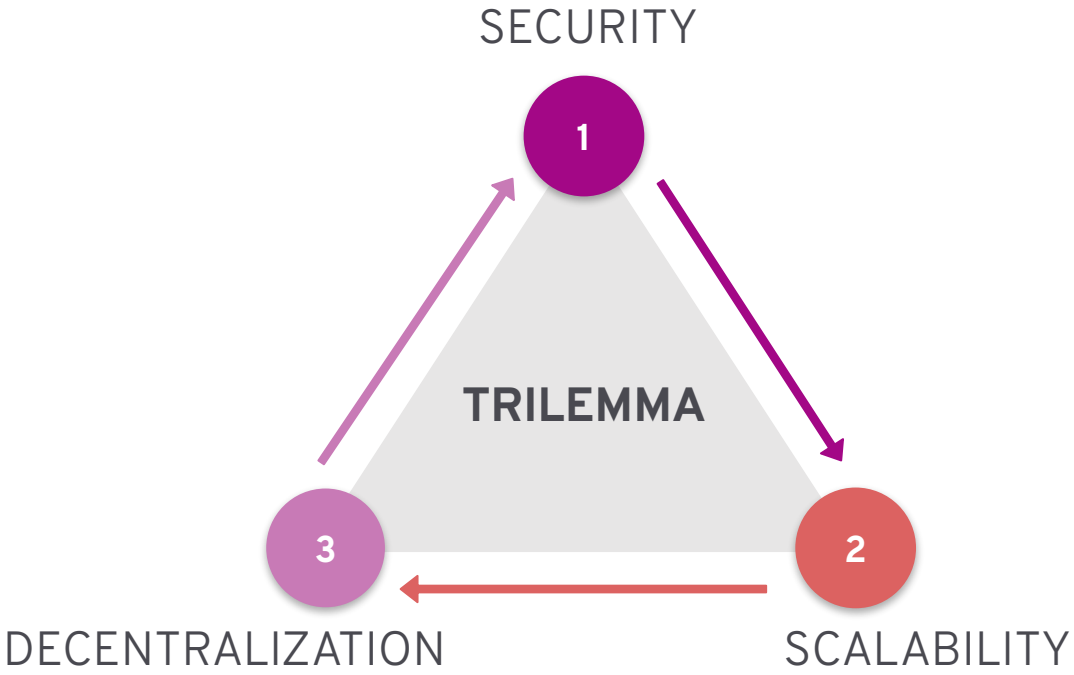
	CRITICAL ISSUES
BONDED PROOF OF STAKE Validators **bind their stake**, to **show their commitment** in validating and appending a new block. Misbehaviors are punished.	• Participating in the consensus protocol makes users' stakes illiquid • Risk of economic barrier to entry
DELEGATED PROOF OF STAKE Users **delegate the validation** of new blocks to a **fixed committee**, through weighted voting based on their stakes.	• Known delegate nodes, therefore exposed to DDoS attacks • Centralization of governance

BPoS

DPoS



Is the *Blockchain Trilemma* unsolvable?



Algorand

ALGORAND CONSENSUS

Pure Proof of Stake (PPoS)

Algorand PPoS Consensus

- **Scalable** 6000 TPS, billions of users
- **Fast** < 3.9 s per block
- **Secure** 0 downtime for over 23M blocks
- **Low fees** 0.001 ALGO per txn
- **No Soft Forks** prob. < 10^{-18}
- **Instant Transaction Finality**
- **Carbon neutral**
- **Minimal hardware node requirements**
- **No delegation or binding of the stake**
- **No minimum stake**
- **Secure with respect DDoS**
- **Network Partitioning resilience**

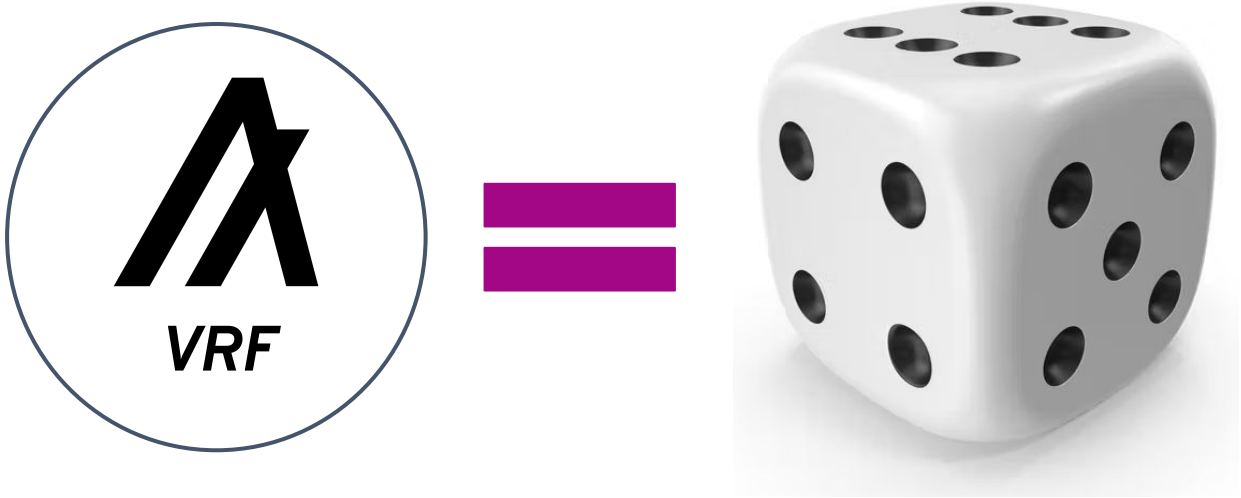
Silvio Micali

Algorand Founder

Professor MIT, Turing Award, Gödel Prize

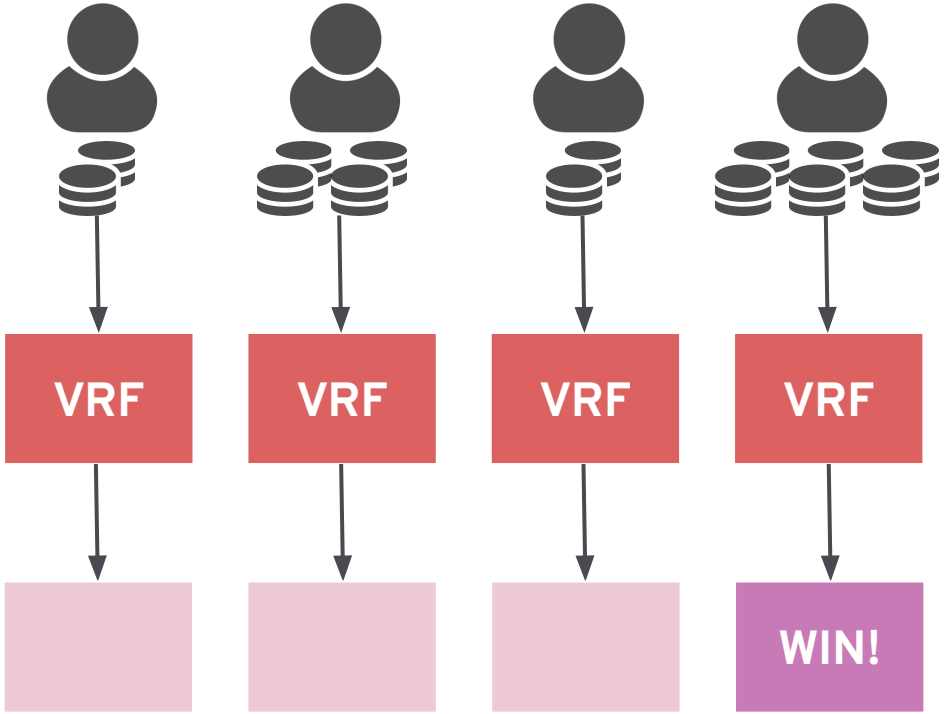
Digital Signatures, Probabilistic Encryption, Zero-Knowledge Proofs, Verifiable Random Functions and other primitives of modern cryptography.

Tamper-proof, unique and verifiable dices



1. Dices are **perfectly balanced** and **equiprobable**, nobody could tamper their result!
2. Keep **observing dice rolls** by no means increases the chance of guessing the next result!
3. Each dice is **uniquely signed** by its owner, nobody can roll someone else dices!
4. Dices are **publicly verifiable**, everybody can read the results of a roll!

Who chose the *next block*?



1. Each **ALGO** could be assimilated to a **tamper-proof dice** participating in a safe and secret **cryptographic dice roll**. More ALGOs more dices to roll.

2. For **each new block**, dice rolls are performed in a **distributed, parallel and secret** and manner, directly on online accounts' hardware (in microseconds).

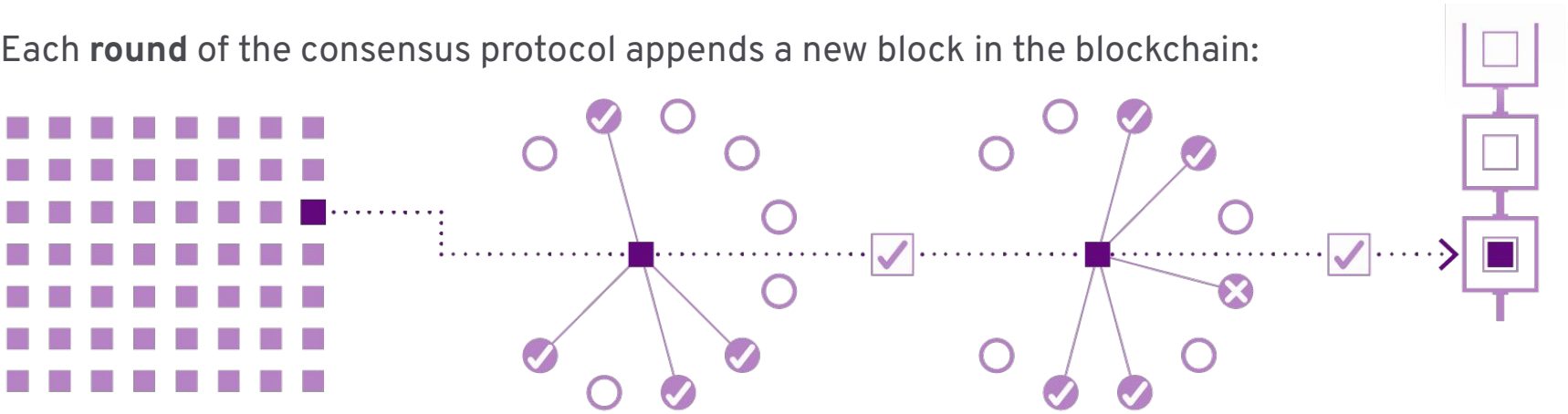
3. The **winner is revealed** in a safe and **verifiable way** only after winning the dice roll, proposing the next block.

A glimpse on “simplified” VRF sortition

1. A **secret key (SK) / public verification key (VK)** pair is associated **with each ALGO in the account**
2. For each new round ***r*** of the **consensus protocol** a **threshold *L(r)*** is defined
3. **Each ALGO** in the account **performs a VRF**, using its own **secret key (SK)**, to generate:
   a. a pseudo-random number: $Y = VRF_{SK}(seed)$
   b. the verifiable associated proof: $\rho_{SK}(seed)$
4. If $Y = VRF_{SK}(seed) < L(r)$, that specific ALGO **“wins the lottery”** and **viraly propagates the proof** of its victory $\rho_{SK}(seed)$ to other **network’s nodes**, through “gossiping” mechanism
5. Others node can use the **public verification key (VK)** to verify, through $\rho_{SK}(seed)$, that the number **Y** was generated by **that specific ALGO, owned by the winner of the lottery**

Pure Proof of Stake, in short

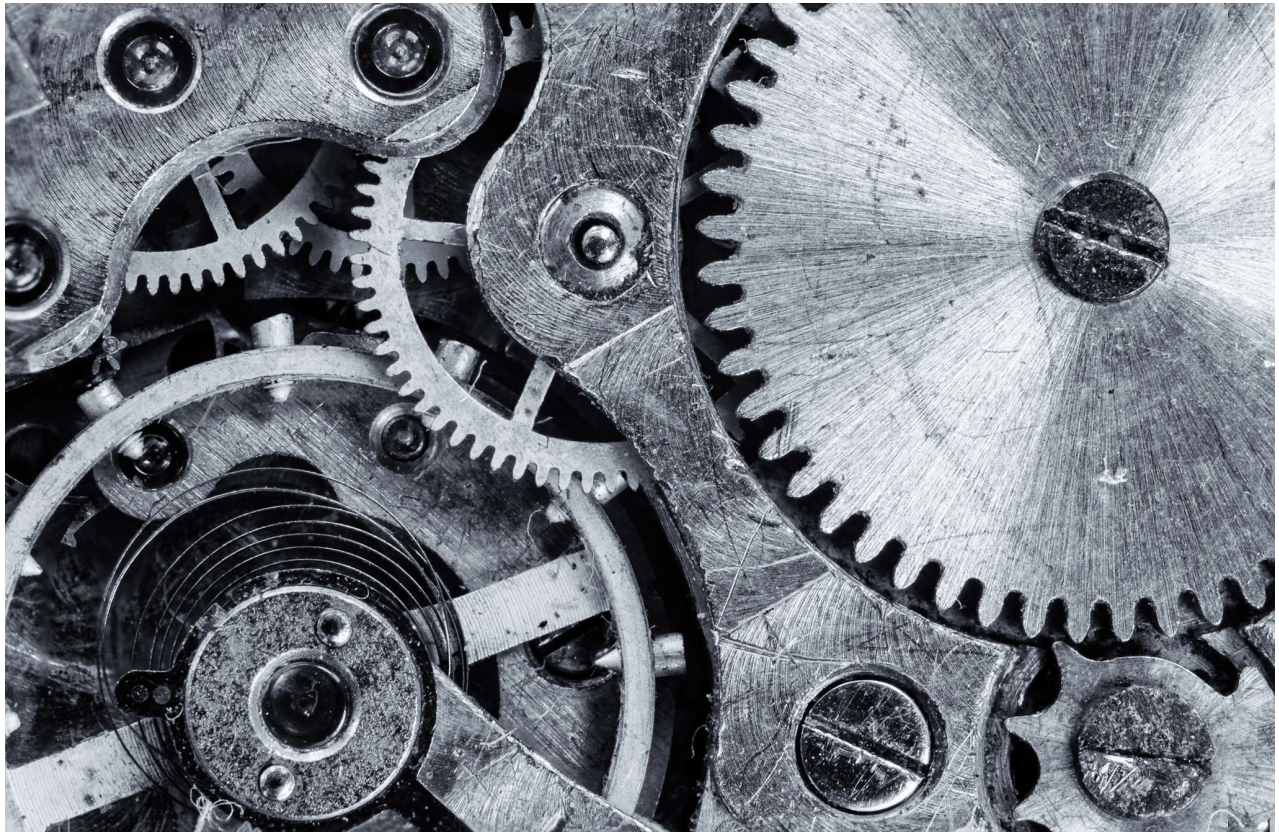
Each **round** of the consensus protocol appends a new block in the blockchain:



An account is elected to **propose** the **next block**	A **committee** is elected to **filter** and **vote** on the **block proposals**	A **new committee** is elected to reach a quorum and **certify** the **block**	The **new block** is **appended** to the **blockchain**

Through the **cryptographic lottery**, an **online account** is elected with probability directly proportional to its stake: **each ALGO corresponds to an attempt** to win the lottery!

Ok, but… how long does it take?



Less than 3.9 seconds!

Pure Proof of Stake security

Algorand's decentralized Byzantine consensus protocol can tolerate an **arbitrary number of malicious users** as long as honest users hold a **super majority of the total stake** in the system.

1. The adversary **does not know which users he should corrupt**.
2. The adversary **realizes which users are selected too late** to benefit from attacking them.
3. Each **new set of users will be privately and individually elected**.
4. During a network partition in Algorand, the adversary is **never able to convince two honest users to accept two different blocks** for the same round.
5. Algorand is able to **recover shortly after network partition** is resolved and guarantees that new blocks will be generated at the same speed as before the partition.

Pure Proof of Stake: the output pre-upgrade!

BLOCKS	> 23M with 0 downtime
BLOCKCHAIN SIZE	~ 1 TB
ADDRESSES	> 27M with ~ 2M monthly active addresses
AVG. BLOCK FINALIZATION	~ 3.8 sec per block
TXNS WEEKLY VOLUME	~ 11M transactions (March 2022)
TPS WEEKLY PEAK	~ 6000 transactions per second

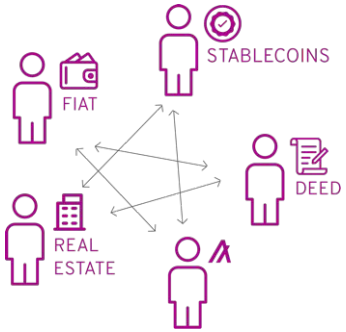
** up to August 2022*

Algorand Layer-1 primitives

Algorand Standard Assets (ASA)

- SECURITIES
- CURRENCIES
- STABLECOINS
- UTILITY TOKENS
- CERTIFICATIONS
- REAL ESTATE

Atomic Transfers (AT)



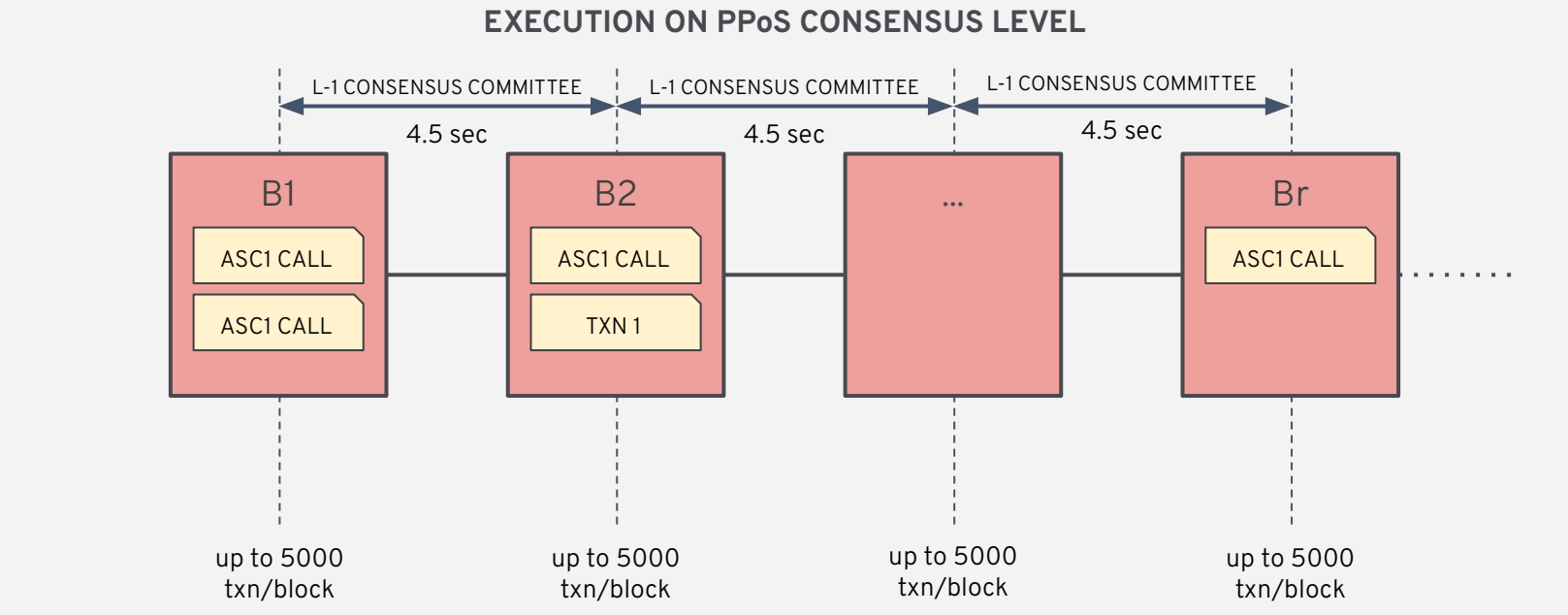
Algorand Virtual Machine (AVM)

- FAST
- SECURE
- LOW COST

Algorand State Proof (ASP)

- TRUSTLESS
- INTEROPERABLE
- POST-QUANTUM SECURE

What does *execution on Layer-1* mean?



AVM execution does not slow down the whole blocks production!

What does *execution on Layer-1* mean?

- Smart Contracts are executed *“at consensus level”*
- Benefit from network's speed, security, and scalability
- Fast trustless execution (~4.5 seconds per block)
- Low cost execution (0.001 ALGO regardless SC’s complexity)
- Instant Finality of Smart Contracts’ effects
- Native interoperability with Layer-1 primitives
- Safe high level languages (PyTeal, Reach, Clarity)
- Low energy consumption

Algorand

ALGORAND SUSTAINABILITY

"Permission-less" is not "Responsibility-less"

Full paper: "Proof of Stake Blockchain Efficiency Framework"

“Permission-less” is not *“Responsibility-less”*

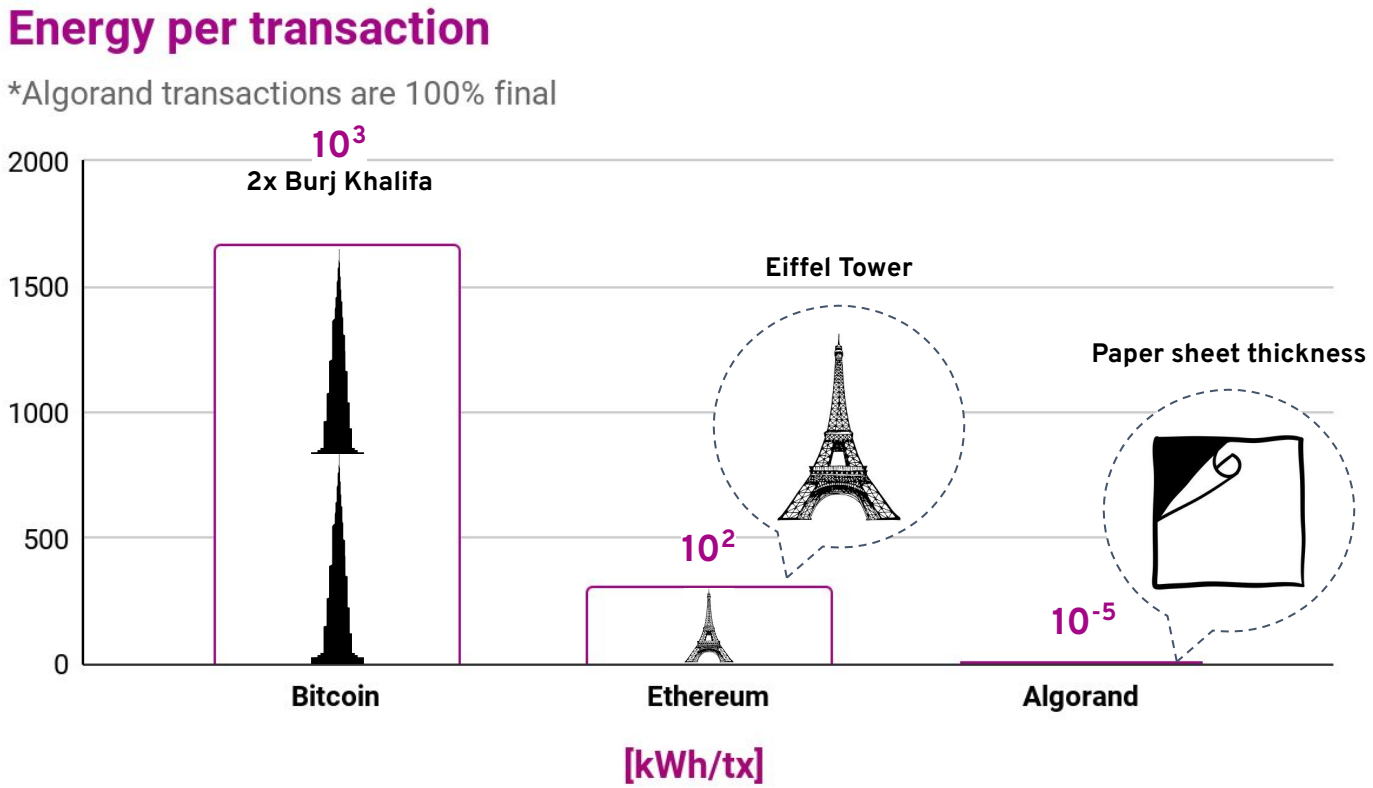
- Algorand is a *permission-less* network 🕸️, so **no centralized authority can know or impose** how node runners should power their nodes 💻.
- As decentralized infrastructure, Algorand is **responsible for its impact** on Planet Earth 🌏, although no centralized authority controls it.
- Algorand **should acts proactively** (not hiding behind the *“permission-less excuse”*) and set blockchains’ sustainability bar high 🌱.

Proof of unsustainable Work

- **Proof of Work** is a planetary **wasteful computational battle**, in which miners **MUST burn energy** to secure the blockchain.
- Showing off personal commitment in the ecosystem through the **consumption of computational and energetic resources** is at the **core of PoW** consensus mechanism.

Our planet Earth can no longer afford unsustainable technologies.

A matter of orders of magnitude (PoW vs PPoS)



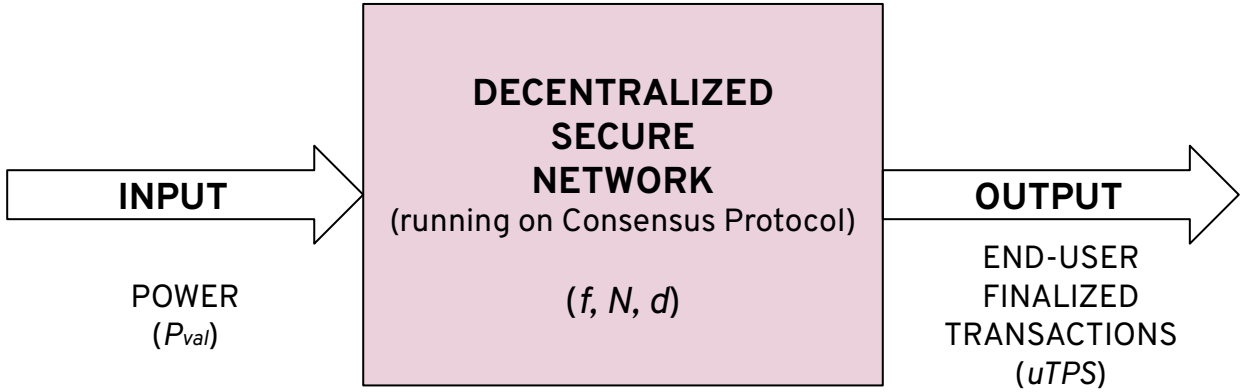
Full paper: "Proof of Stake Blockchain Efficiency Framework"

You like to win easy!

What about others Proof of Stake?

When the going gets tough, the tough get going (PoS vs PPoS)

Blockchain **sustainability** must consider **scalable end-user transactions** (*uTPS*) **finality** (*f*), **nodes hardware** (*N*), and **secure network decentralization** (*d*).



Being sustainable while centralized, insecure or not scalable is worthless!

Reframing the question

Is Algorand blockchain efficient
at consuming energy
to finalize end user useful transactions
in a secure, scalable and decentralized way?

Algorand solves *blockchain trilemma* sustainably

- **Algorand transactions are 100% available to end-users**

  (other PoS blockchains consume their own transactions for consensus)
- **Algorand transactions are 100% instantly final**

  (other PoS must consume the energy of several blocks to ensure transactions' finality)
- **Algorand transactions are secured by a very decentralized network**

  (some PoS blockchain have only few validators)
- **Algorand security is a feature of its own efficiency**

  (Algorand never experienced downtime since the genesis block)

Full paper: "Proof of Stake Blockchain Efficiency Framework"

Algorand

ALGORAND NETWORKS

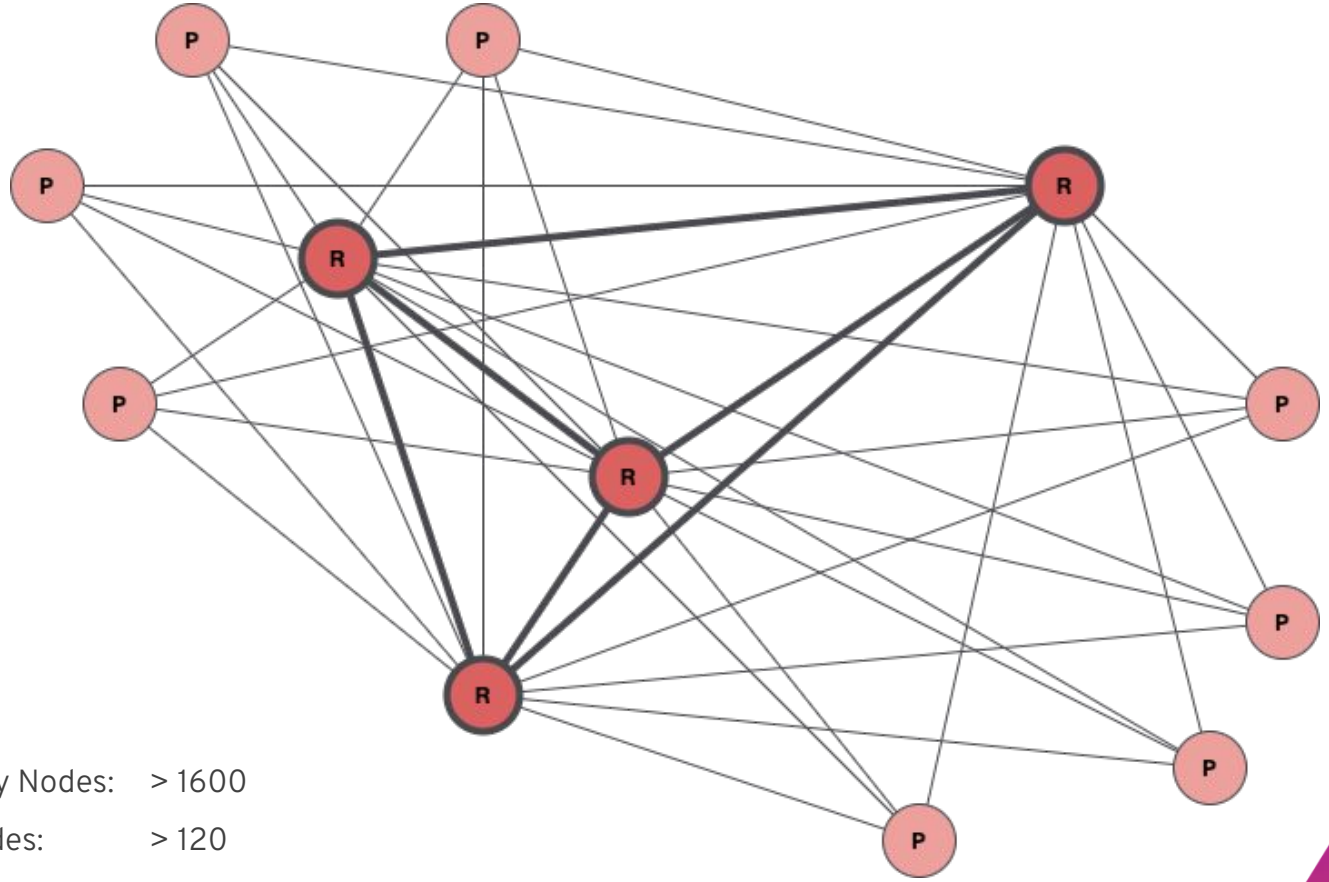
Nodes, Indexer and APIs

Algorand Node configurations

1. **Non-Relay Nodes**
   - Participate in the PPoS consensus (if hosting participation keys)
   - Connect only to Relay Nodes
   - Light Configuration: store just the lastest 1000 blocks (Fast Catch-Up)
   - Archival Configuration: store all the chain since the genesis block

2. **Relay Nodes**
   - Communication routing to a set of connected Non-Relay Nodes
   - Connect both with Non-Relay Nodes and Relay Nodes
   - Route blocks to all connected Non-Relay Nodes
   - Highly efficient communication paths, reducing communication hops

Example of Algorand Network topology



Node Metrics

- Non-Relay Nodes: > 1600
- Relay Nodes: > 120

Access to Algorand Network

The **Algorand blockchain is a distributed system of nodes** each maintaining their **local state** based on validating the history of blocks and the transactions therein. **Blockchain state integrity** is maintained by the **consensus protocol** which is implemented within the **Algod daemon** (often referred to as the node software).

An application **connects to the Algorand blockchain** through an **Algod client**, requiring:

- **a valid Algod REST API endpoint IP address**
- **an Algod token from an Algorand node connected to the network you plan to interact with**

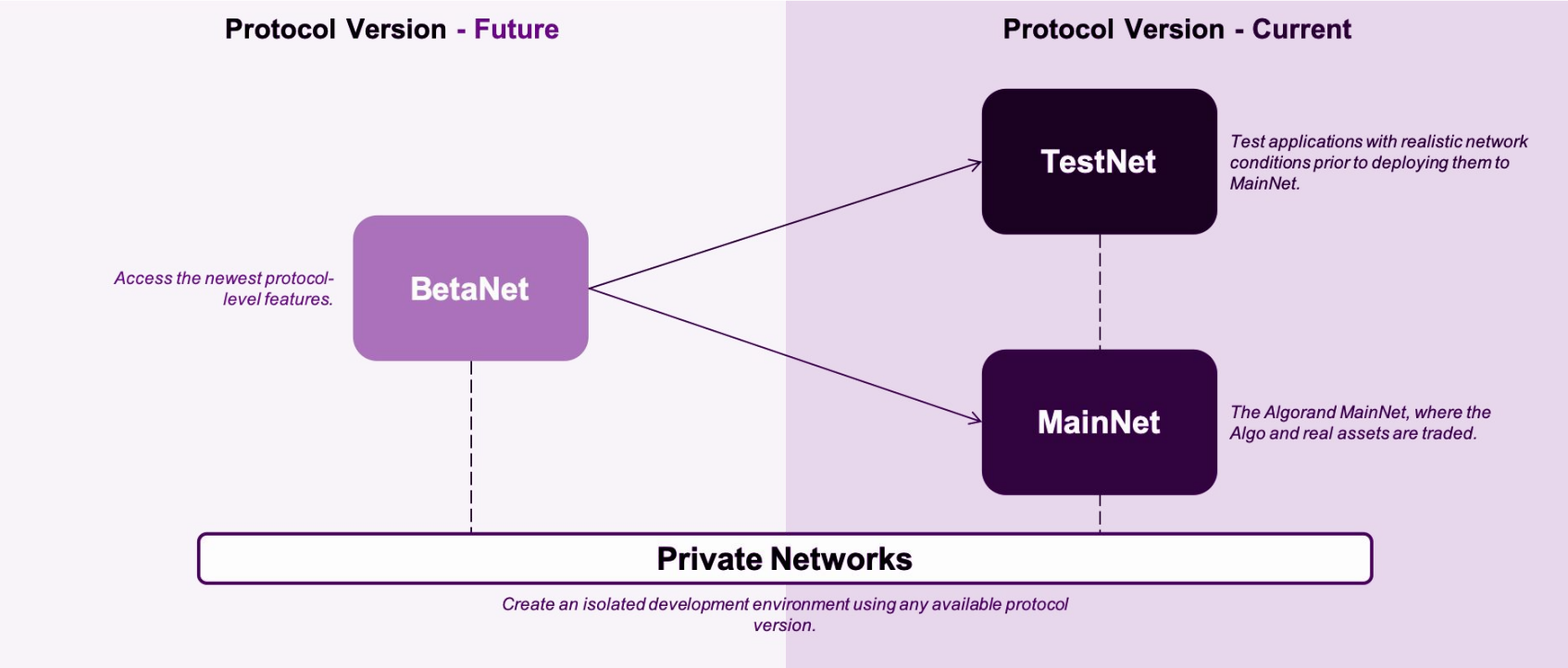
These **two pieces of information** can be provided by **your local node** or by a **third party node aaS**.

How to get an Algod Client?

There are **three ways** to get a REST API Algod **endpoint IP address / access token**, each with their respective pros and cons depending on development goals.

	Use a third-party service	Use Docker Sandbox	Run your own node
Time	**Seconds** - Just signup	**Minutes** - Same as running a node with no catchup	**Days** - need to wait for node to catchup
Trust	1 party	1 party	Yourself
Cost	Usually free for development; pay based on rate limits in production	Variable (with free option) - see node types	Variable (with free option) - see node types
Private Networks	❌	✅	✅
`goal`, `algokey`, `kmd`	❌	✅	✅
Platform	Varies	MacOS; Linux	MacOS; Linux
Production Ready	✅	❌	✅

Algorand Networks



Algorand Node - Writing on the blockchain

1. Install (Linux, MacOS, Windows)
2. Choose a **network** (MainNet, TestNet, BetaNet, PrivateNet)
3. Start & Sync with the network, Fast Catchup

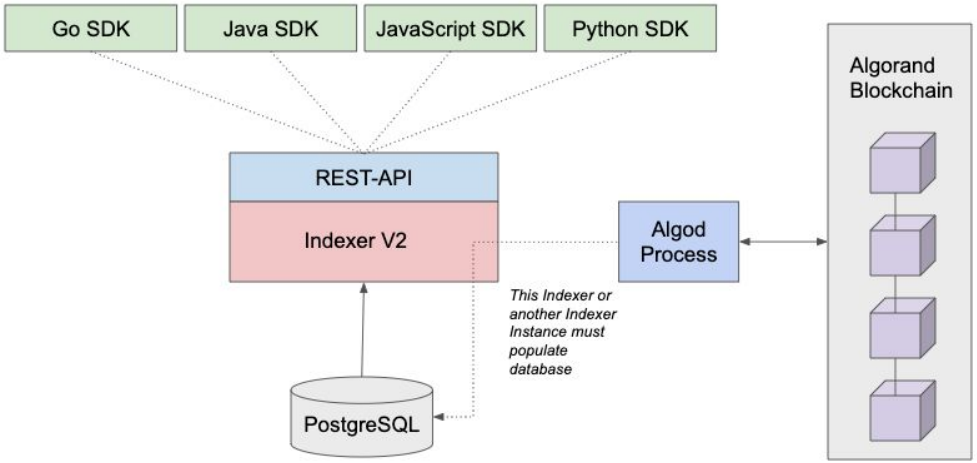
genesis.json (mainnet)


```
{
  "alloc": [
    {
      "addr": "7377777777777777...77777777UFEJ2CI",
      "comment": "RewardsPool",
      "state": {
        "algo": 10000000000000,
        "onl": 2
      }
    },
    {
      "addr": "Y76M3MSY6DKBRHBL7C3...F2QWNPL226CA",
      "comment": "FeeSink",
      "state": {
        "algo": 1000000,
        "onl": 2
      }
    },
    ...
  ],
  "fees": "Y76M3MSY6DKBRHBL7C3NNDX...F2QWNPL226CA",
  "id": "v1.0",
  "network": "mainnet",
  "proto":
"https://github.com/algorandfoundation/specs/tree/5615ad
c36bad610c7f165fa2967f4ecfa75125f0",
  "rwd": "737777777777777777777...77777777UFEJ2CI",
  "timestamp": 1560211200
}
```


Interacting with Algorand Nodes

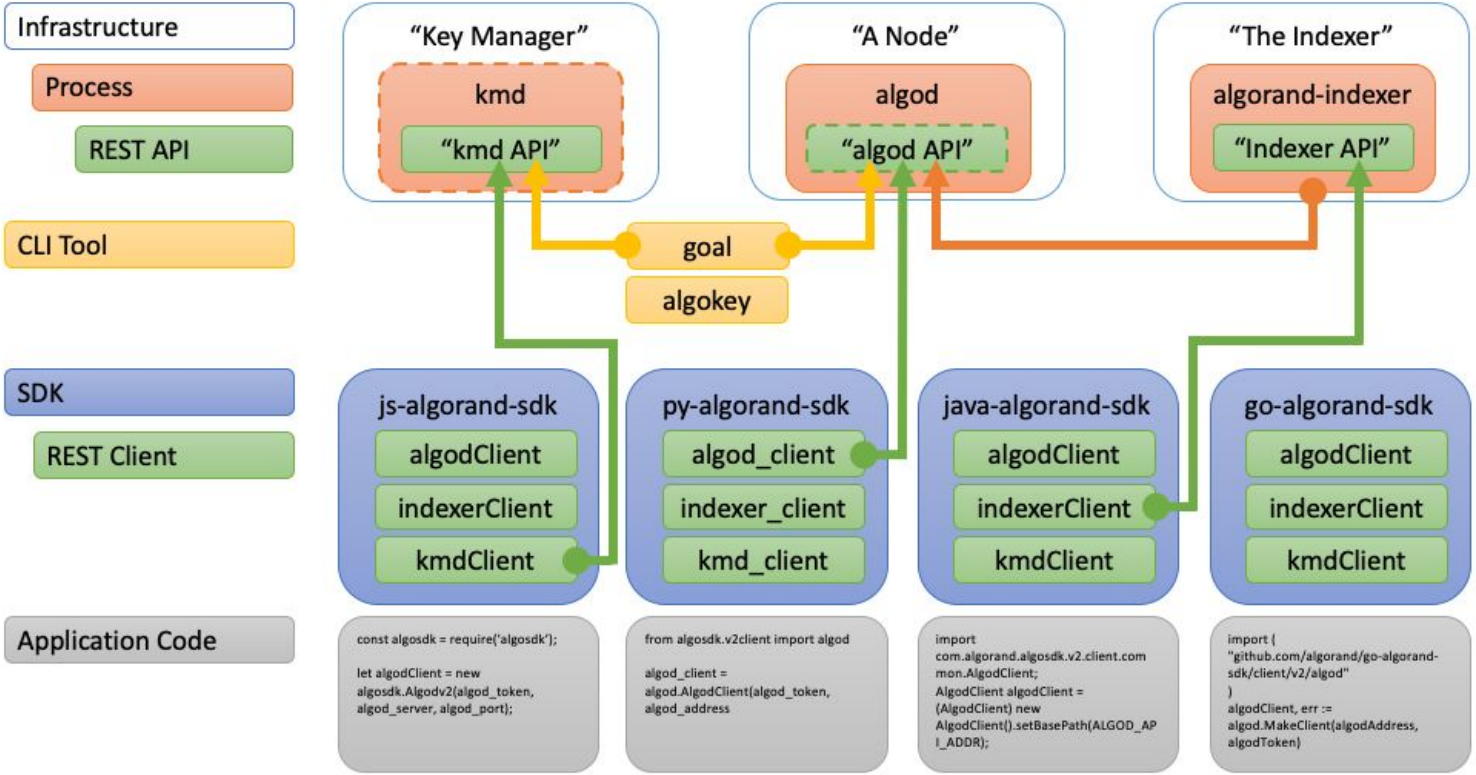
1. CLI utilities: **goal**, **kmd** and **algokey**
2. REST API interface: **algod V2**, **kmd**, **indexer**
3. Algorand SDKs: **JavaScript**, **Python**, **Java** o **Go**

Algorand Indexer - Reading from the blockchain



The **Indexer** provides a **REST API interface** of API calls to **query the Algorand blockchain**. The Indexer REST APIs retrieves blockchain data from a **PostgreSQL database**, populated using the Indexer instance connected to an **Archival Algorand node that reads blocks' data**. As with the Nodes, the Indexer can be used as a **third-party service**.

How to interact with Algorand Node and Indexer

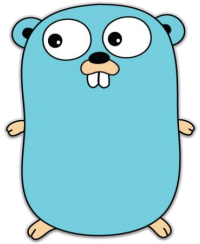


Algorand SDKs

Java

JavaScript

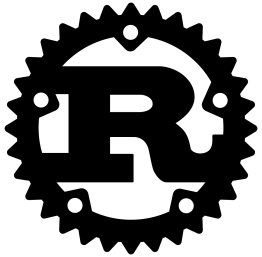
Python



Go

Algorand Community SDKs

C#



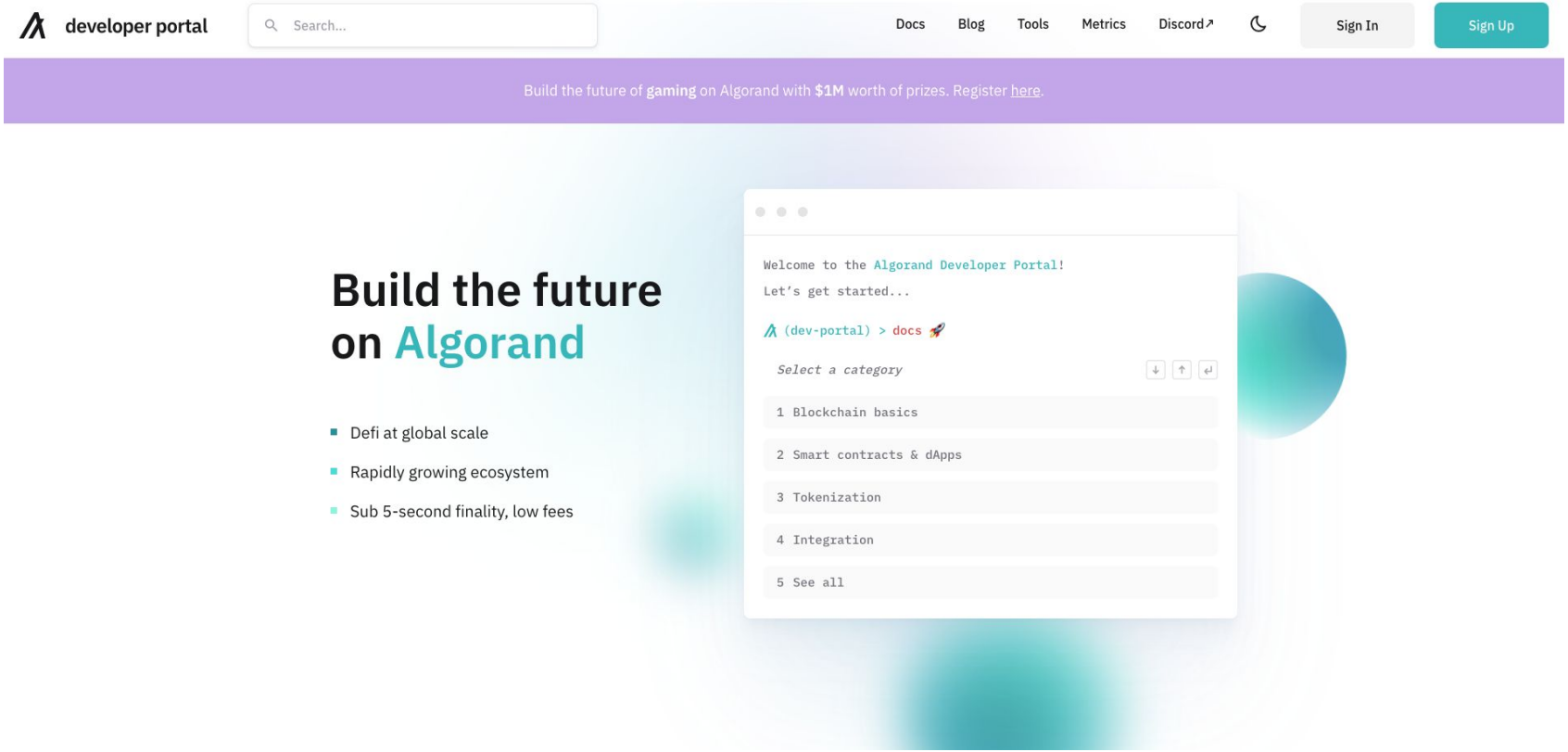
Rust

Dart

PHP

Swift

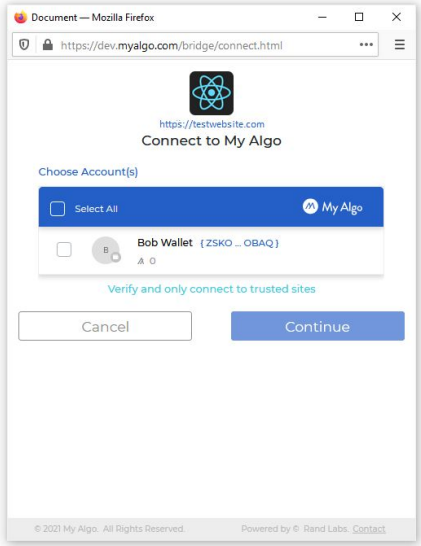
Algorand Developer Portal



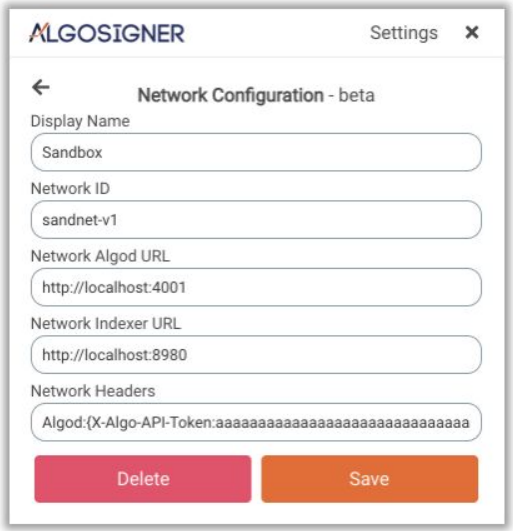
Awesome Algorand

Algorand Wallets

[Mobile Wallet + Wallet Connect]



[MyAlgo Wallet]



[AlgoSigner]

Algorand Explorers

Algorand

ALGORAND INTEROPERABILITY

State Proofs and Post-Quantum Security

Credits to Noah Grossman for contens

Trustless interoperability

Algorand approach to interoperability:

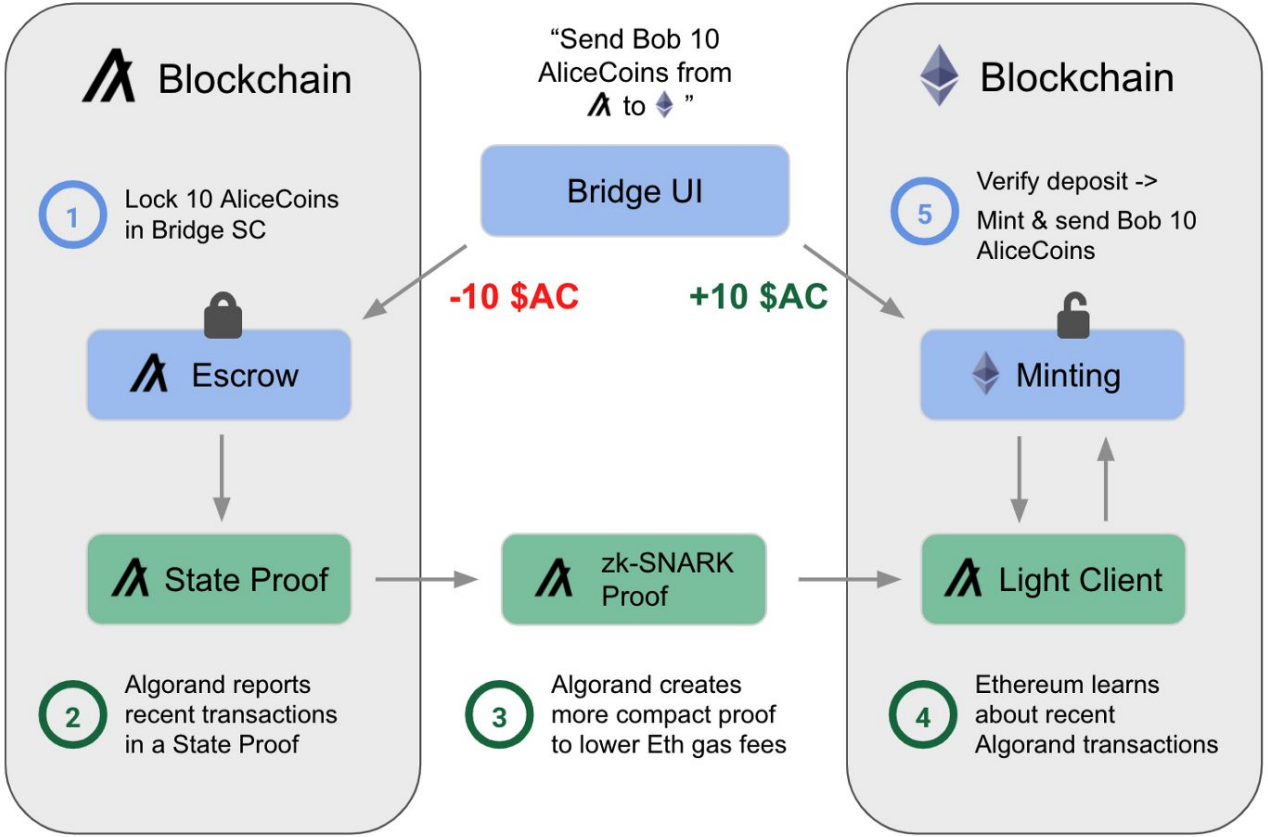
cross-chain transactions should rely just on

1. Trust in **departing** consensus protocol
2. Trust in **arrival** consensus protocol

without centralized bridges or validator networks, to handle the assets.

Algorand State Proofs remove trusted centralized operators becoming the **first trustless post-quantum secure L1 interoperability standard**.

Post-Quantum Secure Algorand State Proofs



Post-Quantum secure and **immutable proofs**, attesting blockchain state, generated by Pure Proof of Stake consensus protocol.

Algorand

ALGORAND TRANSACTIONS

Core element of blocks

Changing blockchain state

Transactions are the **core element of blocks**, which define the **evolution** of distributed ledger **state**. There are **six transaction types** in the Algorand Protocol:

1. Payment
2. Key Registration
3. Asset Configuration
4. Asset Freeze
5. Asset Transfer
6. Application Call

These six transaction types can be specified in particular ways that result in more granular perceived transaction types.

Signature, fees and round validity

In order to be approved, Algorand's transactions must comply with:

1. **Signatures**: transactions must be correctly signed by its sender, either a **Single Signature**, a **Multi Signature** or a **Smart Signature / Smart Contract**
2. **Fees**: in Algorand transactions fees are a way to protect the network from DDoS. In Algorand Pure PoS fees are not meant to pay "validation" (as it happens in PoW blockchains). In Algorand you can **delegate fees**.
3. **Round validity**: to handle transactions' idempotency, letting Non-Archival nodes participate in Algorand Consensus, transactions have an intrinsic validity of 1000 blocks (at most).

Browse through a transaction

Transactions are characterized by two kind of **fields** (codec):

- **common** (header)
- **specific** (type)

Field	Required	Type	codec	Description
Fee	*required*	uint64	`"fee"`	Paid by the sender to the FeeSink to prevent denial-of-service. The minimum fee on Algorand is currently 1000 microAlgos.
FirstValid	*required*	uint64	`"fv"`	The first round for when the transaction is valid. If the transaction is sent prior to this round it will be rejected by the network.
GenesisHash	*required*	[32]byte	`"gh"`	The hash of the genesis block of the network for which the transaction is valid. See the genesis hash for MainNet, TestNet, and BetaNet.
LastValid	*required*	uint64	`"lv"`	The ending round for which the transaction is valid. After this round, the transaction will be rejected by the network.
				the account that pays the fee and amount.
				ype of transaction. This value is automatically generated e developer tools.

Field	Required	Type	codec	Description
Receiver	*required*	Address	`"rcv"`	The address of the account that receives the amount.
Amount	*required*	uint64	`"amt"`	The total amount to be sent in microAlgos.
CloseRemainderTo	*optional*	Address	`"close"`	When set, it indicates that the transaction is requesting that the Sender account should be closed, and all remaining funds, after the fee and amount are paid, be transferred to this address.

Payment Transaction example

Here is a transaction that sends 5 ALGO from one account to another on MainNet.


```
{
  "txn": {
    "amt": 5000000,
    "fee": 1000,
    "fv": 6000000,
    "gen": "mainnet-v1.0",
    "gh": "wGHE2Pwdvd7S12BL5FaOP20EGYesN73ktiC1qzkkit8=",
    "lv": 6001000,
    "note": "SGVsbG8gV29ybGQ=",
    "rcv": "GD64YIY3TWGDMCNPP553DZPPR6LDUSFQOIJVFDPPXWEG3FVOJCCDBBHU5A",
    "snd": "EW64GC6F24M7NDSC5R3ES4YUVE3ZXXNMARJHDCCCLIHZU6TBEOC7XRSBG4",
    "type": "pay"
  }
}
```

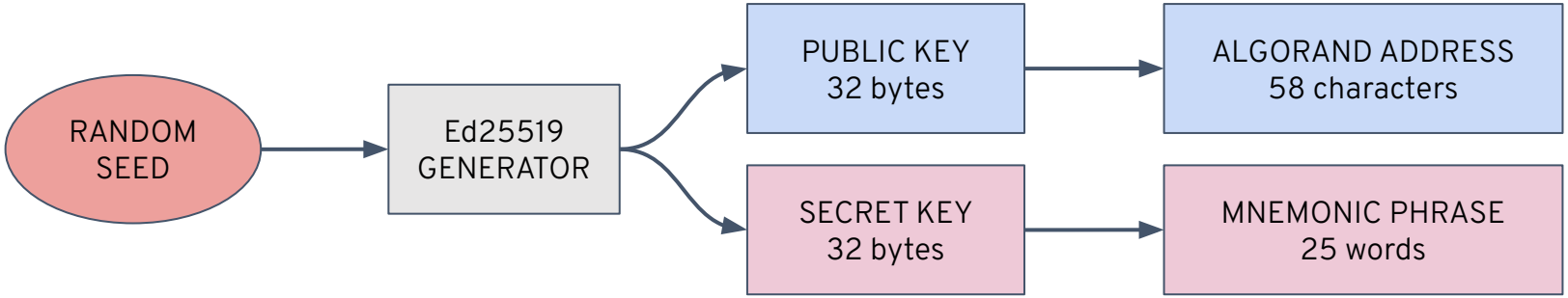
Algorand

ALGORAND ACCOUNTS

Transactions' Authorization

Signatures

Algorand uses **Ed25519** high-speed, high-security elliptic-curve signatures.

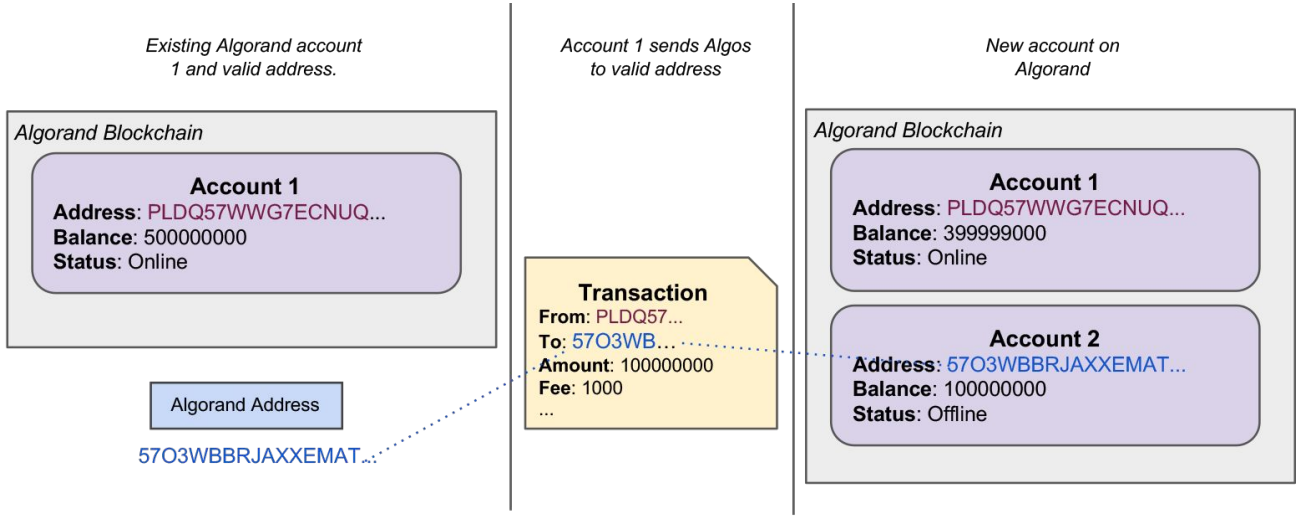


- **ADDRESS:** the **public key** is transformed into an **Algorand Address**, by adding a 4-byte checksum to the end of the public key and then **encoding it in base32**.

- **MNEMONIC:** the 25-word **mnemonic** is generated by converting the **private key bytes** into **11-bit integers** and then **mapping** those integers to the bip-0039 English word list.

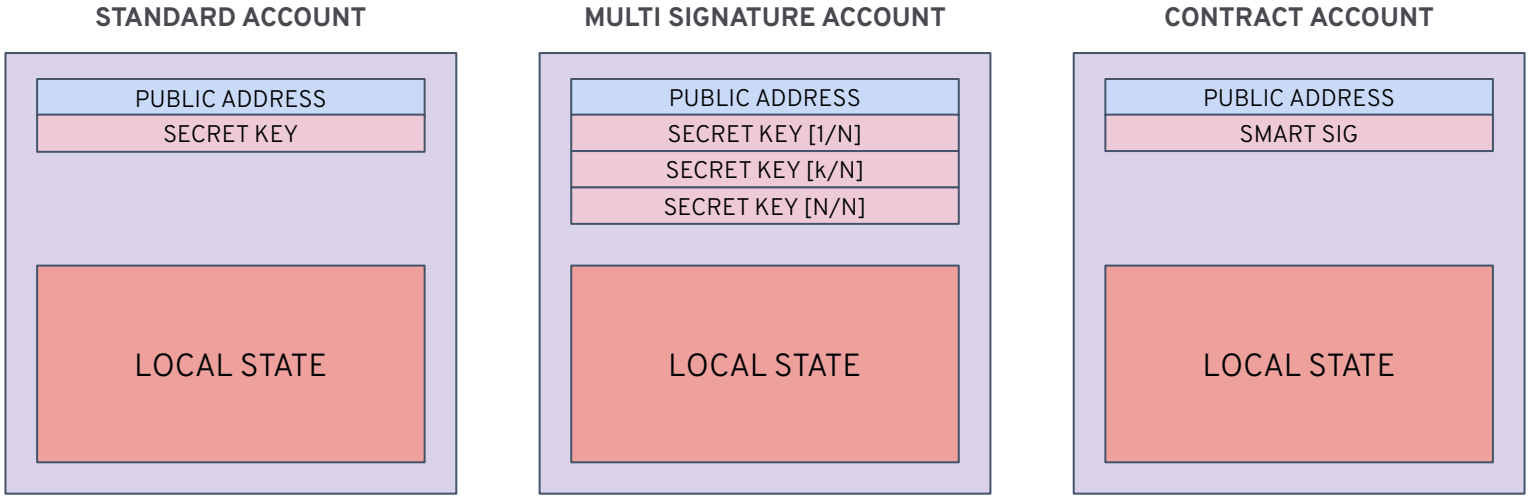
Algorand Accounts

Accounts are **entities on the Algorand blockchain** associated with specific **on-chain local sate**. An **Algorand Address** is the **unique identifier** for an **Algorand Account**.



All the potential keys pairs "already exists" mathematically, we just keep discovering them.

Transactions Authorization and Rekey-To



Algorand Rekeying: powerful **Layer-1 protocol feature** which enables an Algorand account to **maintain a static public address** while dynamically **rotating the authoritative private spending key(s)**. Any Account can Rekey either to a Standard Account, MultiSig Account or LogicSig Contract Account.

Account State Minimum Balance

Name	Current value	Developer doc	Consensus parameter name in (.go)	Note
Default	0.1 Algos	reference	MinBalance	
Opt-in ASA	+ 0.1 Algos	reference	MinBalance	
Created ASA	+ 0.1 Algos	reference	MinBalance	creator of ASA does not need to opt in

Name	Current value	Developer doc	Consensus parameter name in (.go)	Note
Per page application creation fee	0.1 Algos	reference	AppFlatParamsMinBalance	
Flat for application opt-in	0.1 Algos	reference	AppFlatOptInMinBalance	
Per state entry	0.025 Algos	reference	SchemaMinBalancePerEntry	
Addition per integer entry	0.0035 Algos	reference	SchemaUintMinBalance	
Addition per byte slice entry	0.025 Algos	reference	SchemaBytesMinBalance	

Algorand

ASA

Algorand Standard Assets on Layer-1

ARC-3

Algorand Standard Asset Parameters Conventions for Fungible and Non-Fungible Tokens

- Status: *Final*
- Defines **Fungible Tokens**, **Pure NFT**, **Fractional NFTs**
- Very comprehensive definition of NFT's metadata (e.g. images, videos, audio tracks, etc.) and traits as JSON structure
- Binding between the ASA and the JSON structure **uploaded on external storage** (e.g. IPFS, Arweave, etc.)
- Circulating tokens: ~ 42,000
- Generators: https://arc3.xyz/, https://app.algodesk.io/
- Example: https://www.nftexplorer.app/asset/429087615

ARC-69

Community Algorand Standard Asset Parameters Conventions for Digital Media Tokens

- Status: *Living*
- Adopted both in art and games (https://algoseas.io/)
- Simple and succinct definition of NFT's metadata and traits as JSON structure. Allows tratits configuration after minting
- Binding between the ASA and the JSON structure **uploaded directly on-chain** as *transaction notefield*
- Circulating tokens: ~ 565,000 (most popular standard)
- Generators: https://app.algodesk.io/
- Example: https://www.nftexplorer.app/asset/420625533

ARC-19

Templating of NFT ASA URLs for mutability

- Status: *Final*
- Has been adopted as building block of **non-fungible-domain** standard proposed by NFDomains
- Makes use of ASA Reserve Address to **update the NFT metadata binding** over time.
- Circulating tokens: ~ 26,000
- Generators: https://app.nf.domains/
- Example: https://app.nf.domains/name/john.algo

ARC-20 / ARC-18

Smart ASA / Royalty Enforcement Specification

- Status: ARC-18 (*Draft*), ARC-20 (*Draft*)
- Binds a native ASA with a Smart Contract creating a *"Smart ASA"*, useful whenever a **decentralized transferability policy** must be enforced on-chain (e.g. *royalties*, *vesting*, *limited amount per day*, etc.)
- Allows ASA programmable full reconfigurability on the AVM (e.g. enforcing a rule to upgrade a trait of a NFT character in game, etc.)
- ARC-20 Reference Implementation
- ARC-18 Reference Implementation (using *Beaker*)

Algorand

ALGORAND VIRTUAL MACHINE (AVM)

Programming on Algorand

What's a *Smart Contract*?

Smart Contracts are **deterministic** programs through which complex **decentralized** trustless **applications** can be executed on the **AVM**.

What's the *AVM*?

The **Algorand Virtual Machine** is a **Turing-complete** secure **execution environment** that runs on Algorand **consensus layer**.

What the AVM *actually* does?

Algorand Virtual Machine purpose: **approving** or **rejecting** transactions' effects **on the blockchain** according to Smart Contracts' logic.

AVM **approves** transactions' effects if and only if:

1. There is a single non-zero value on top of AVM's stack;

AVM **rejects** transactions' effects if and only if:

1. There is a single zero value on top of AVM's stack;
2. There are multiple values on the AVM's stack;
3. There is no value on the AVM's stack;

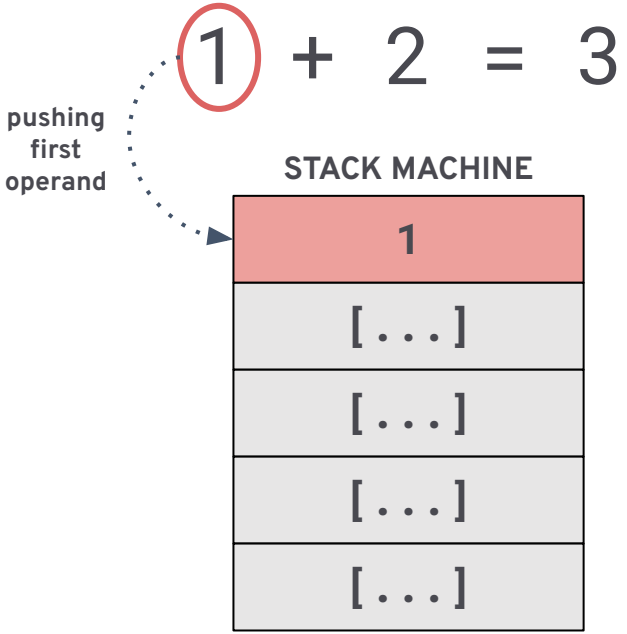
How the AVM *works*?

Suppose we want the AVM to **check** the following assertion:

$$1 + 2 = 3$$

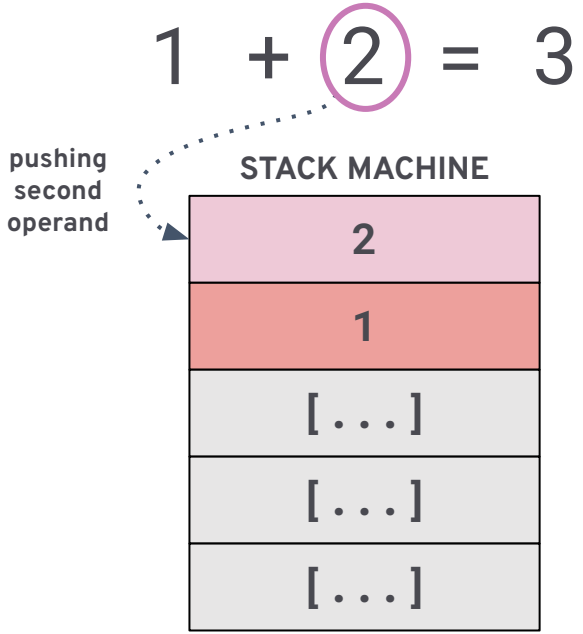
How the AVM *works*?

Suppose we want the AVM to **check** the following assertion:



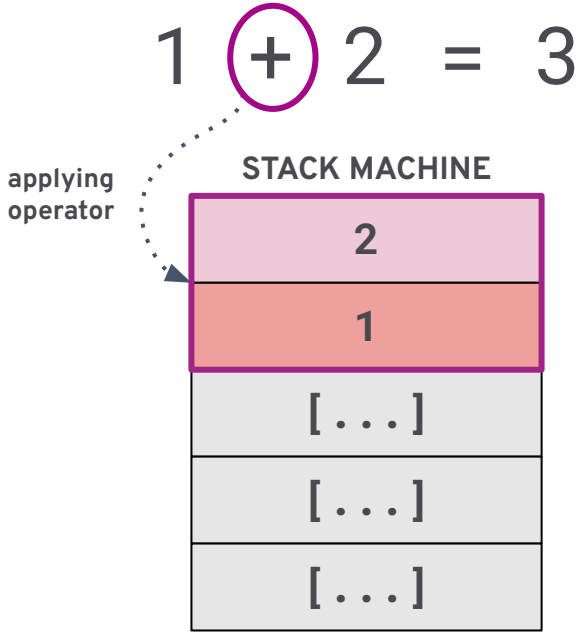
How the AVM *works*?

Suppose we want the AVM to **check** the following assertion:



How the AVM *works*?

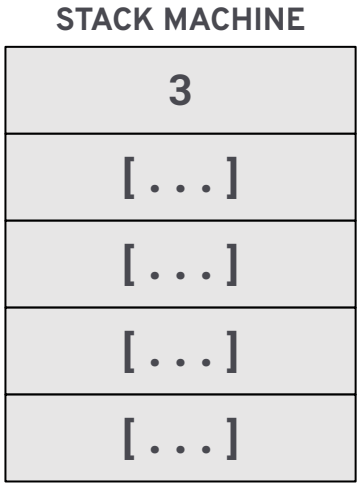
Suppose we want the AVM to **check** the following assertion:



How the AVM *works*?

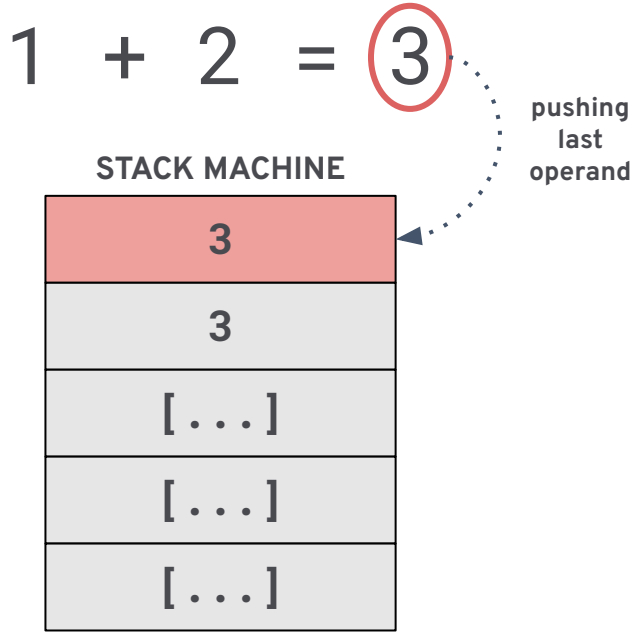
Suppose we want the AVM to **check** the following assertion:

$$1 + 2 = 3$$



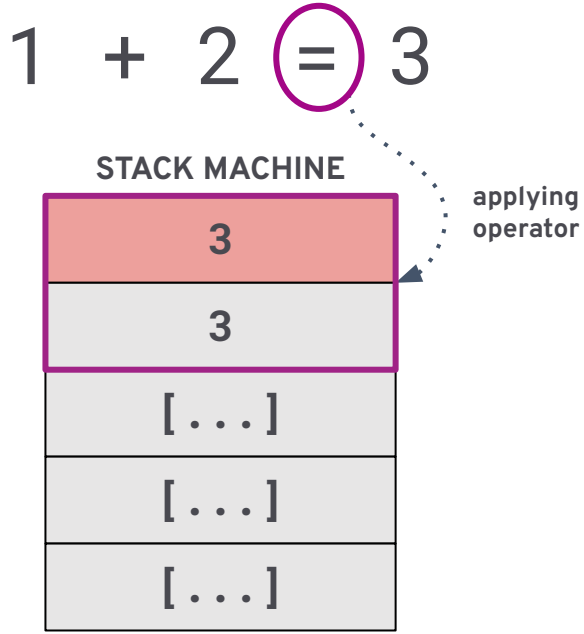
How the AVM *works*?

Suppose we want the AVM to **check** the following assertion:



How the AVM *works*?

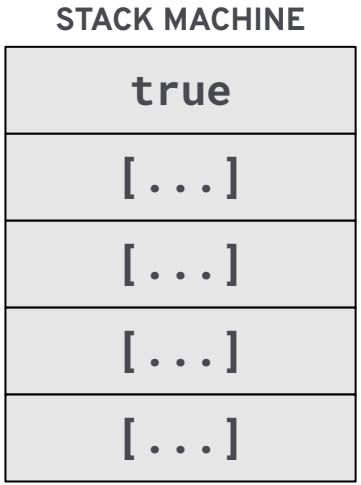
Suppose we want the AVM to **check** the following assertion:



How the AVM *works*?

Suppose we want the AVM to **check** the following assertion:

$$1 + 2 = 3$$



AVM architecture

TRANSACTION

1. Sender
2. Receiver
3. Fee
4. FirstValid
5. LastValid
6. Amount
7. Lease
8. Note
9. TypeEnum
10. …

TRANSACTION ARGS

[0]:	Bytes
[i]:	Bytes
[255]:	Bytes

Stateless properties

APP ARG ARRAY

[0]:	UInt64/ Bytes
[i]:	UInt64/ Bytes
[15]:	UInt64/ Bytes

ACCOUNT ARRAY

[0]:	Bytes
[i]:	Bytes
[3]:	Bytes

ASSET ARRAY

[0]:	UInt64
[i]:	UInt64
[7]:	UInt64

APP IDs ARRAY

[0]:	UInt64
[i]:	UInt64
[7]:	UInt64

APP GLOBAL K/V PAIRS

[0]:	UInt64 / Bytes
[i]:	UInt64 / Bytes
[63]:	UInt64 / Bytes

APP LOCAL K/V PAIRS

[0]:	UInt64 / Bytes
[i]:	UInt64 / Bytes
[15]:	UInt64 / Bytes

Max Key + Value size: 128 bytes

Stateful properties

PROGRAM

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
&&
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
txn Amount
int 42
==
txn Amount
int 77
==
||
&&
```

Processing

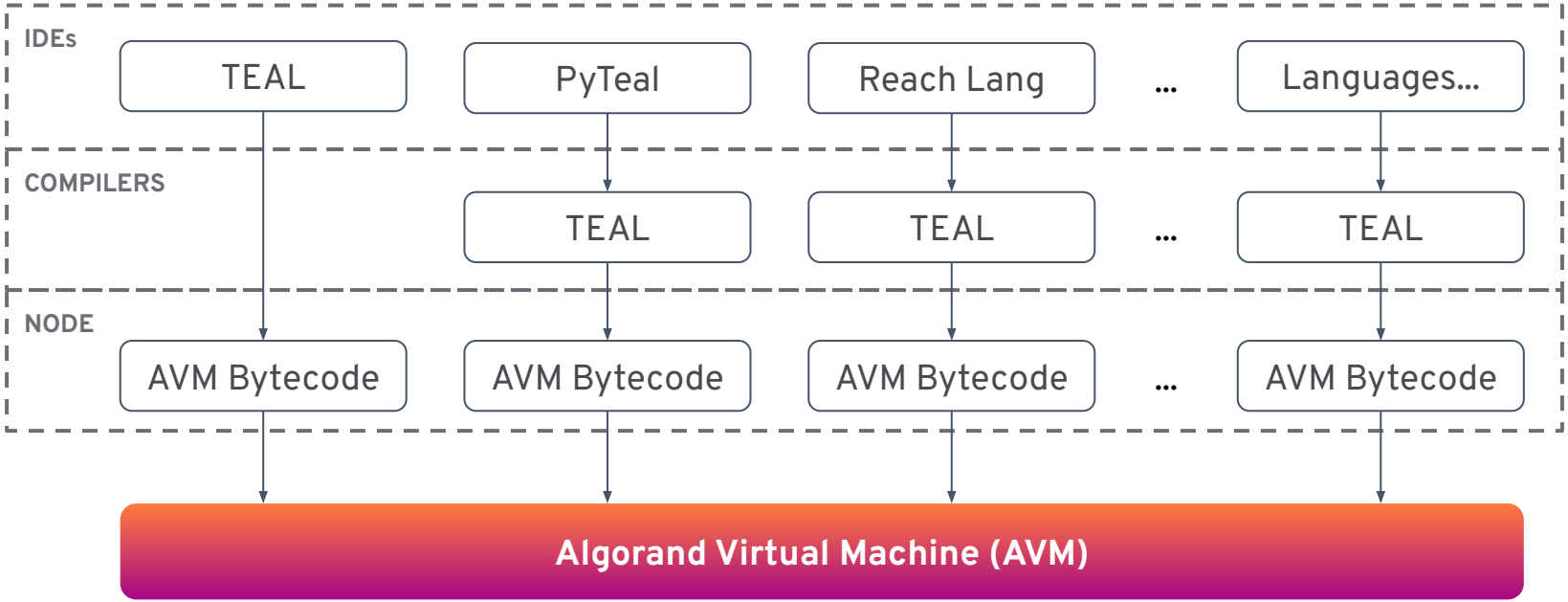
STACK MACHINE

[0]:	UInt64 / Bytes
[i]:	UInt64 / Bytes
[999]:	UInt64 / Bytes

SCRATCH SPACE

[0]:	UInt64 / Bytes
[i]:	UInt64 / Bytes
[255]:	UInt64 / Bytes

How to program the AVM?



AVM vs EVM

	Algorand Virtual Machine	Ethereum Virtual Machine
TURING COMPLETENESS	YES	YES
EXECUTION SPEED	~ 4.5 sec regardless dApp complexity	> 20 sec depends on dApp complexity
ENERGY EFFICIENCY	~10^{-5} [kWh/txn] all final	~10^{2} [kWh/txn] not all final
EXECUTION COSTS	Flat Fee for Smart Contract Calls and Inner Transactions	Depends on dApp complexity
INTEROPERABILITY	native interoperability ASA, AT, MultiSig, RekeyTo...	user defined solutions / standards
EFFECTS FINALITY	instant	~ 6 blocks
MATHEMATICAL PRECISION	512 bits	256 bits
PROGRAMMABILITY	TEAL, PyTEAL, Reach, ...	Solidity, Viper, Reach, ...

What can be built on the AVM?

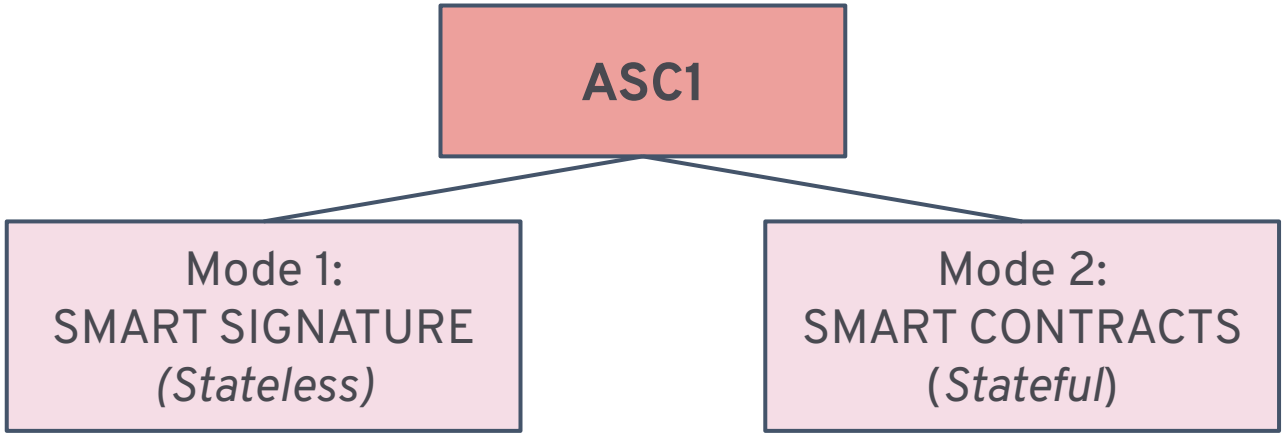
- Escrow accounts
- KYC processes
- Financial instruments (Bonds, ETFs, etc.)
- Loan payment
- Voting applications
- Auctions
- Multiparty or Delegated fund management
- Programmatic recurring fees / recurring debt
- And more...

Algorand

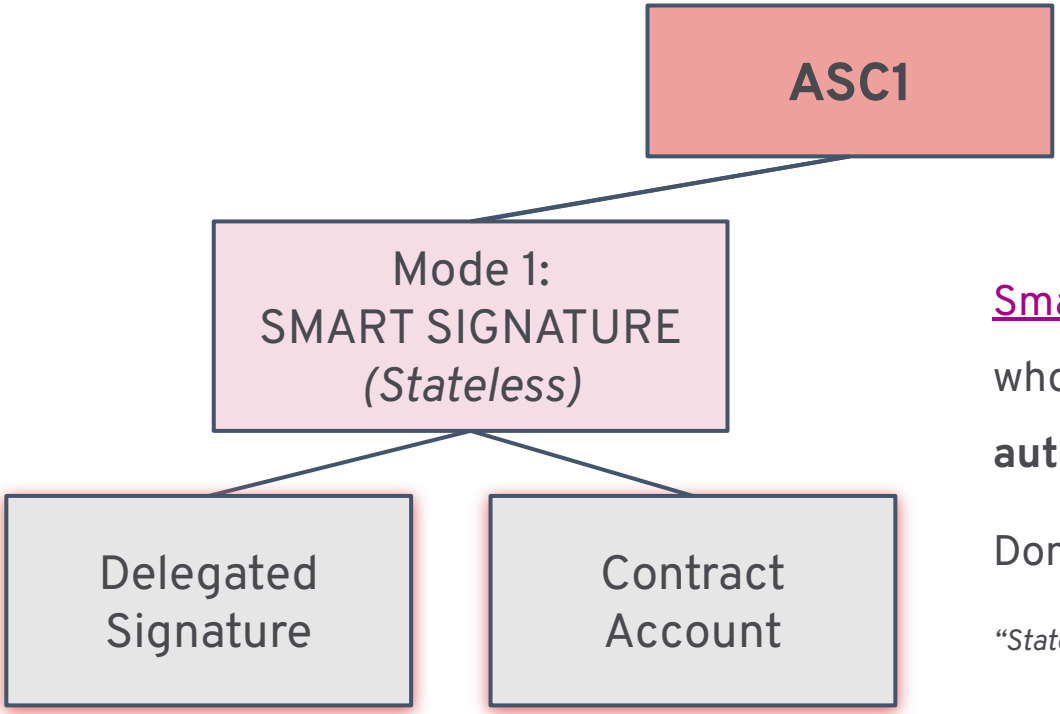
ASC1

Algorand Smart Contracts on Layer-1

AVM Modes


ASC1
Mode 1:
SMART SIGNATURE
(Stateless)
Mode 2:
SMART CONTRACTS
(Stateful)

Stateless ASC1

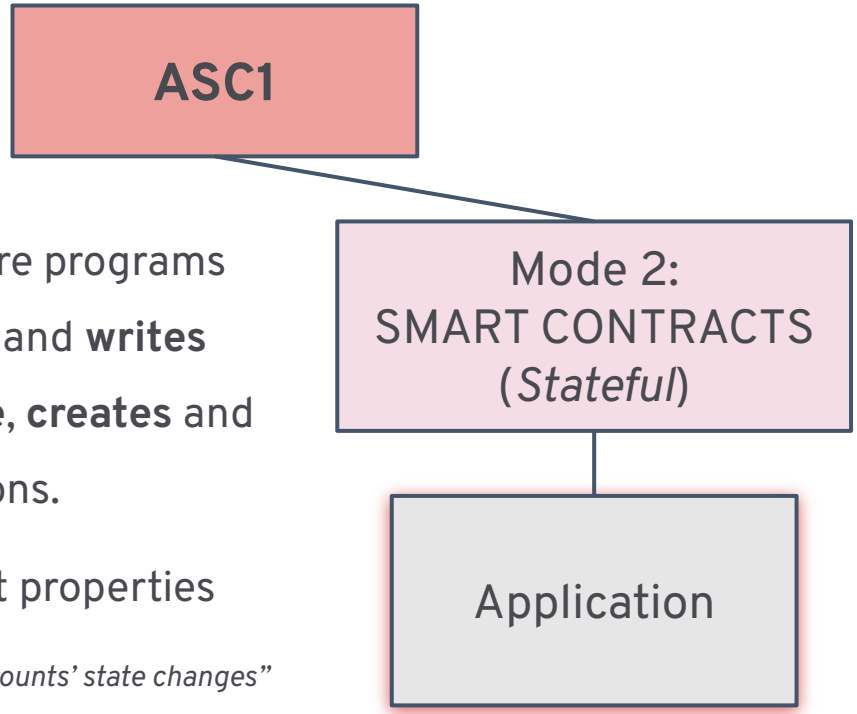


Smart Signatures are programs whose logic **governs transactions' authorization.**

Domain: transient properties

"Stateless ASC1 deal with assets' spending approvals"

Stateful ASC1



Smart Contracts are programs whose logic **reads** and **writes blockchain's state**, **creates** and **submits** transactions.

Domain: persistent properties

"Stateful ASC1 deal with accounts' state changes"

Algorand

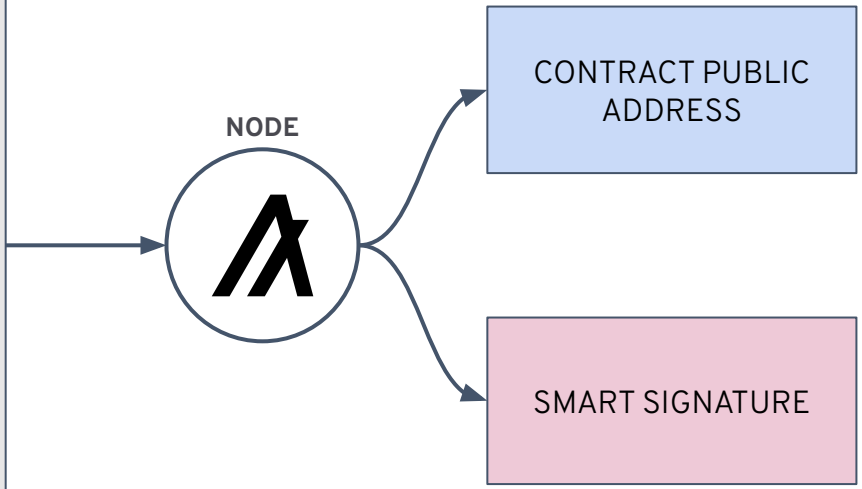
SMART SIGNATURES

Authorizing transactions through TEAL logic

Creating Smart Signature

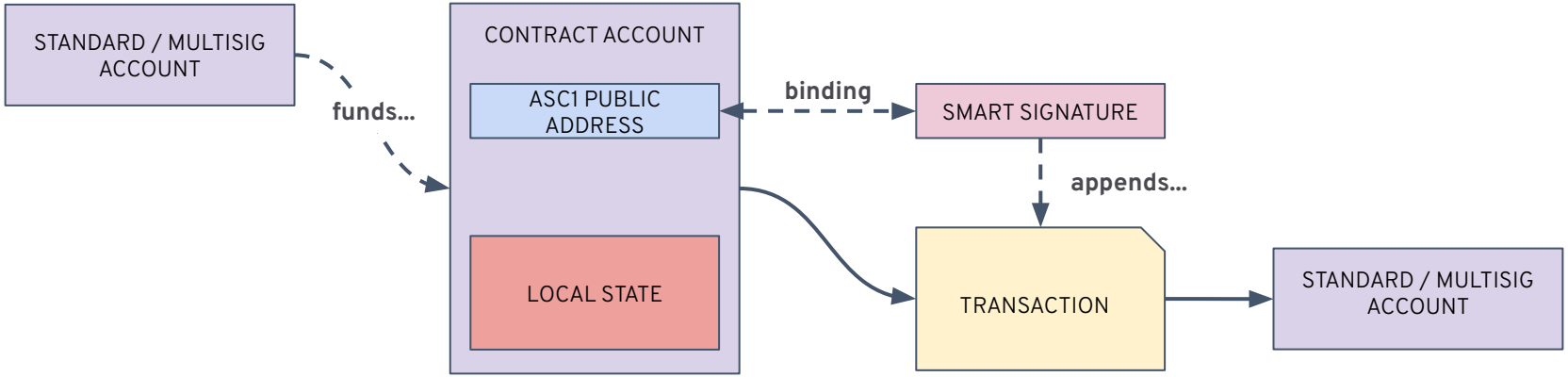
PROGRAM

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
&&
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
txn Amount
int 42
==
txn Amount
int 77
==
||
&&
```

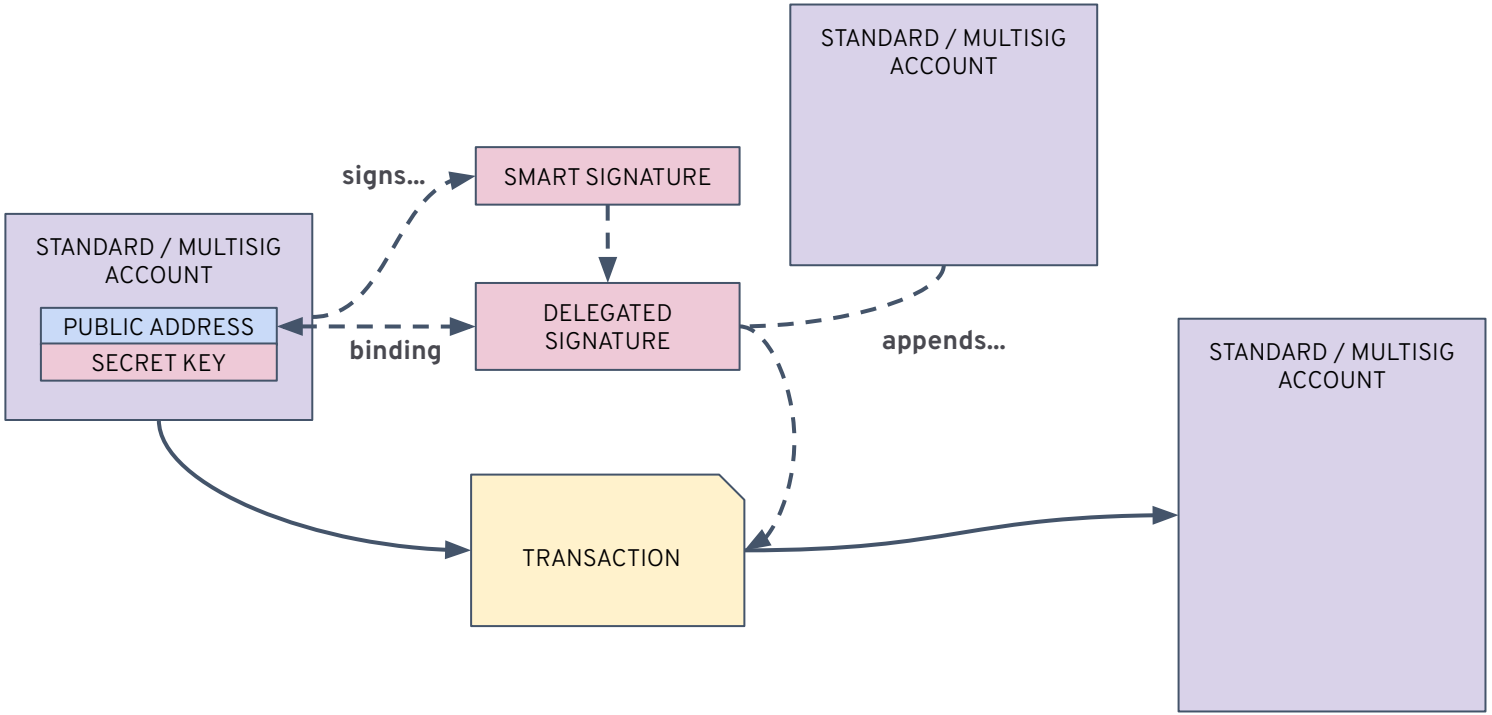


Compiled TEAL Max Size: 1000 bytes
OpCode Max Budget: 20.000

Contract Account



Delegated Signature

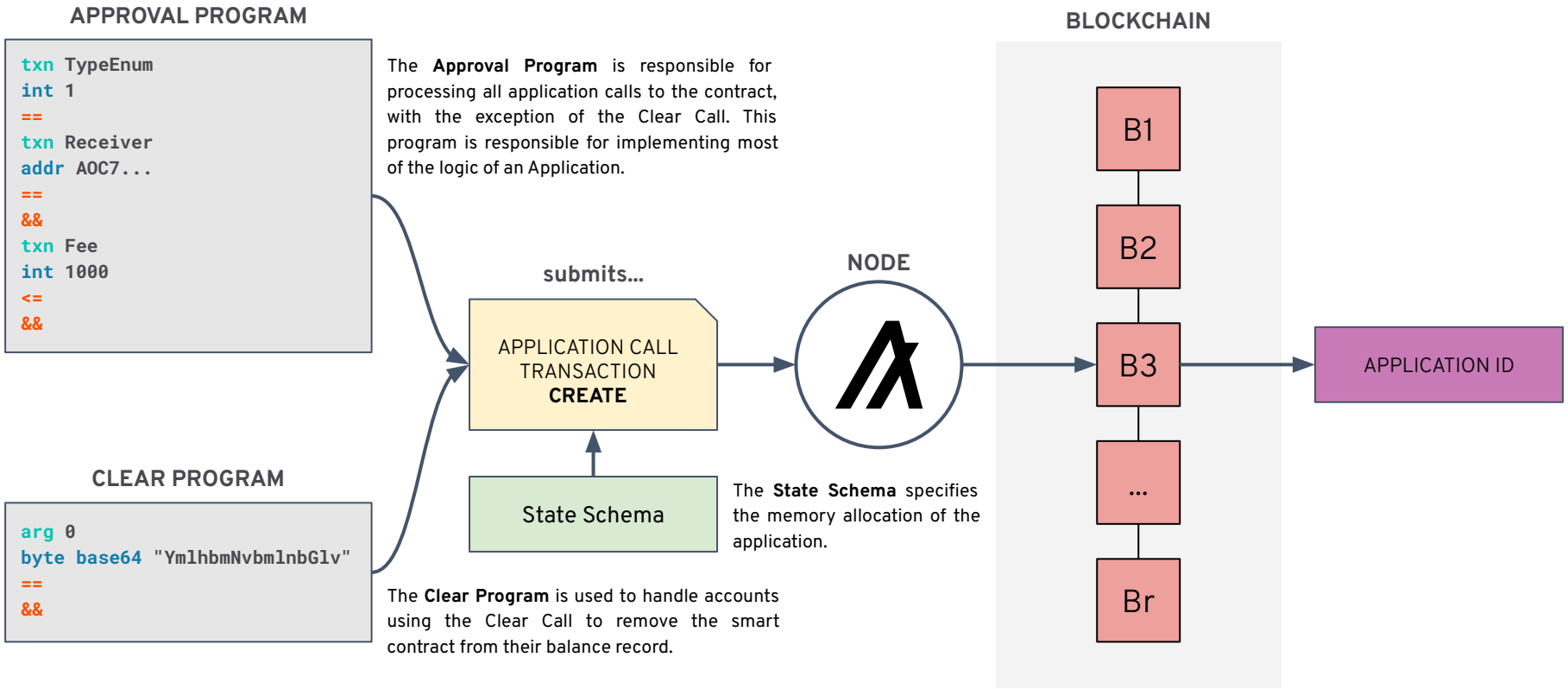


Algorand

SMART CONTRACTS

Decentralized Applications on Algorand

Deploying Applications on chain

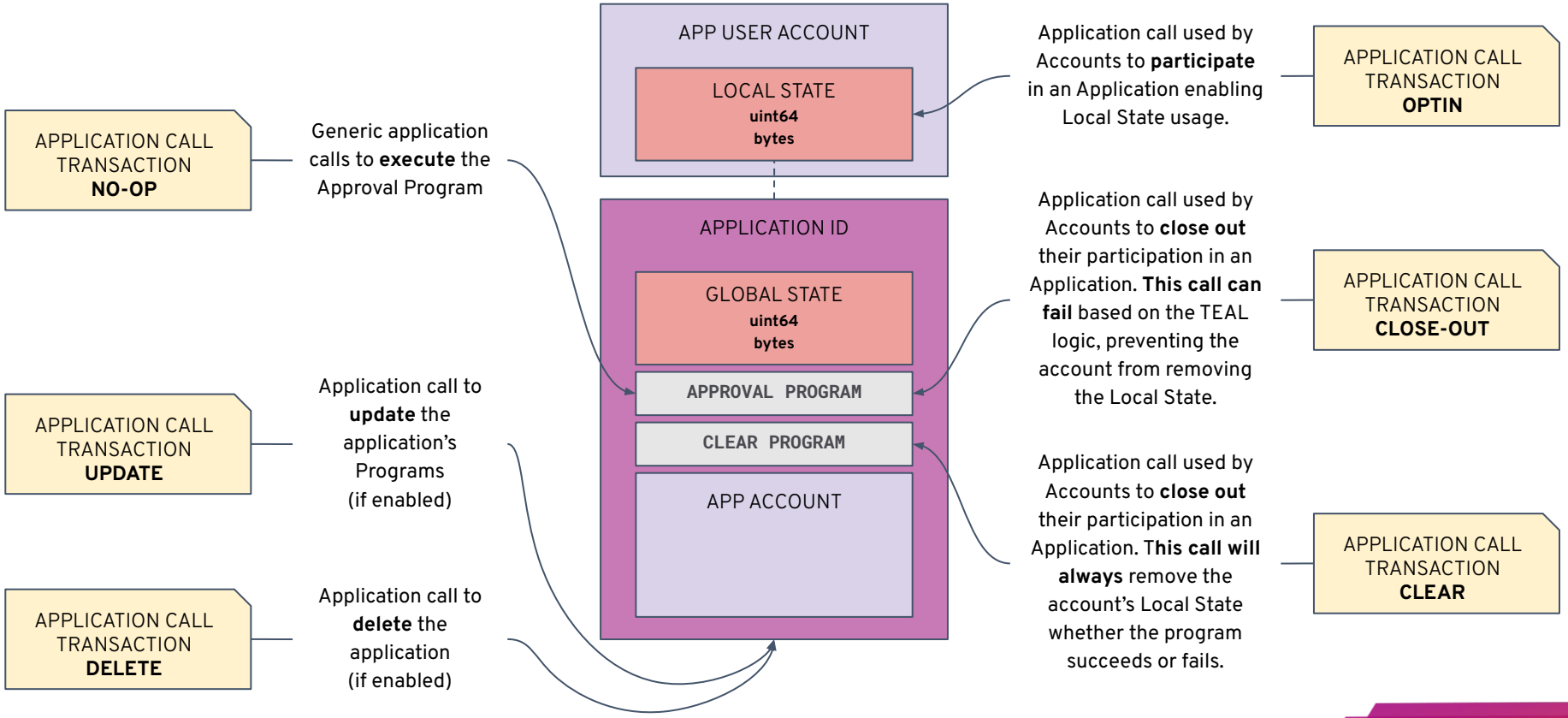


Compiled TEAL Max Size: 2048 bytes (+3 Extra Pages)
OpCode Max Budget: 700 (x16 Atomic Calls)

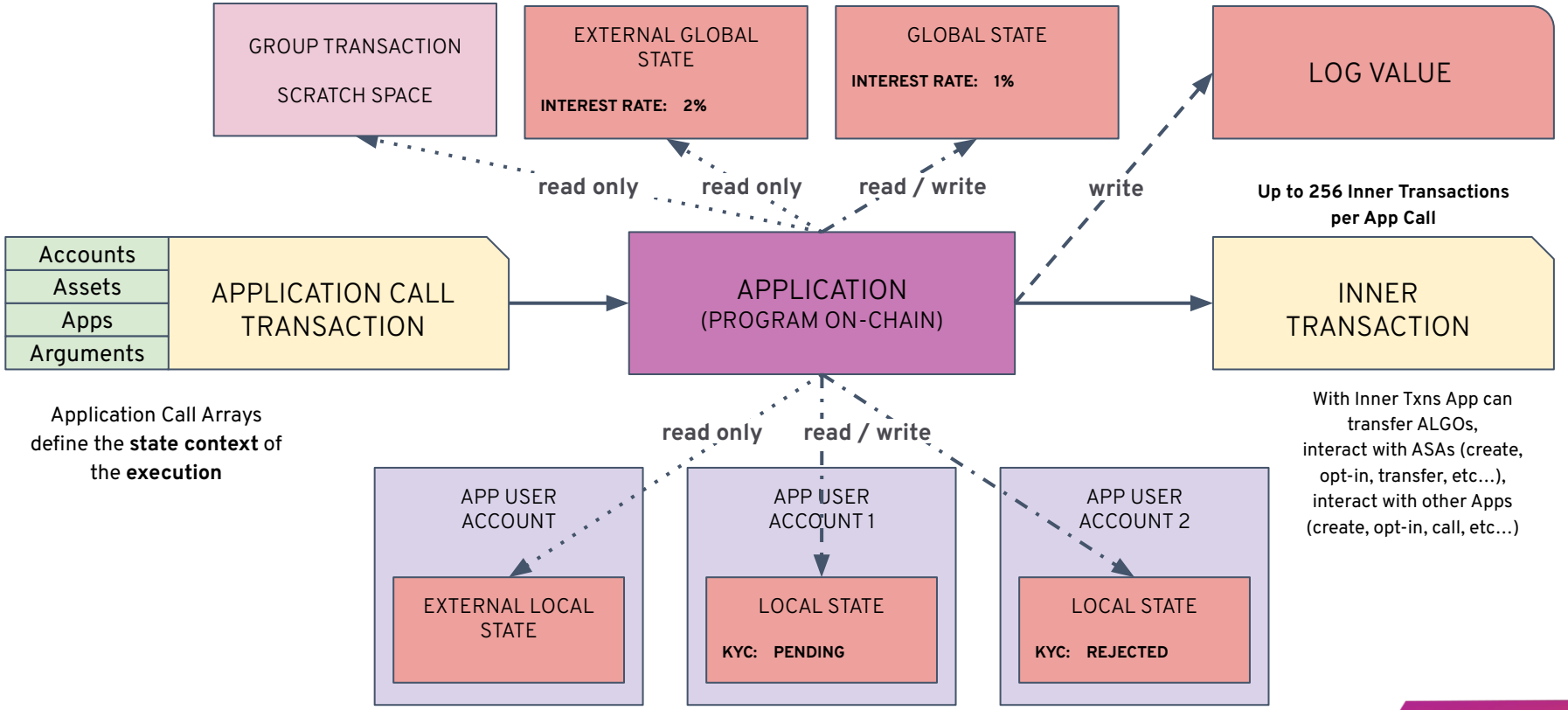
Application Minimum Balance requirements

Name	Current value	Developer doc	Consensus parameter name in (.go)	Note
Per page application creation fee	0.1 Algos	reference	AppFlatParamsMinBalance	
Flat for application opt-in	0.1 Algos	reference	AppFlatOptInMinBalance	
Per state entry	0.025 Algos	reference	SchemaMinBalancePerEntry	
Addition per integer entry	0.0035 Algos	reference	SchemaUintMinBalance	
Addition per byte slice entry	0.025 Algos	reference	SchemaBytesMinBalance	

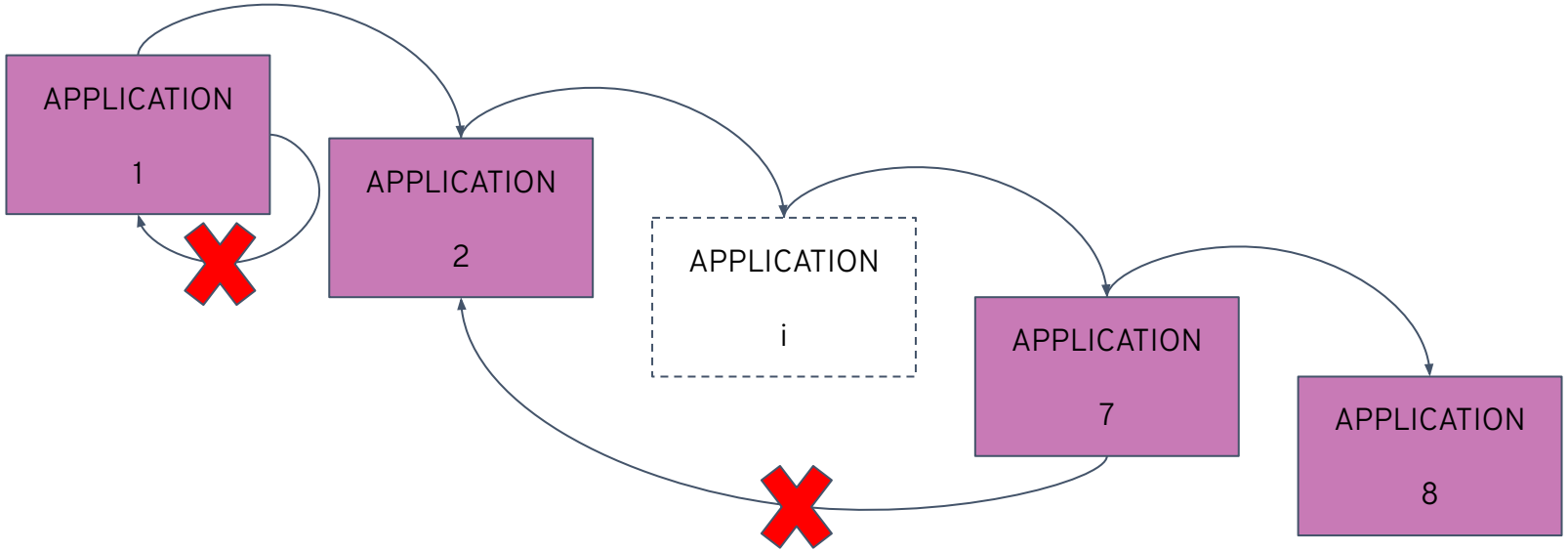
Calling Applications



Interacting with Applications



Contract 2 Contract interaction



1. An application may not call itself, even indirectly. This is referred to as **re-entrancy** adn is explicitly forbidden.
2. An application may only call into other applications **up to a stack depth of 8**.
3. An application **may issue up to 256 inner transactions** to increase its budget (max budget of 179.2k!), but the **max call budget is shared for all applications in the group**.

Smart Contract ABI

The **ABI** (*Application Binary Interface*) defines the **encoding/decoding of data types** and a standard for **exposing and invoking methods** in a Smart Contract. The specification is defined in ARC-4.

```
{
    "name":"super-awesome-contract",
    "networks":{
        "MainNet":{
            "appID": 123456
        }
    },
    "methods":[
        {
            "name":"add",
            "desc":"Add 2 integers",
            "args":[ { "type":"uint64" }, { "type":"uint64" } ],
            "returns": {"type":"uint64"}
        },
        {
            "name":"sub",
            "desc":"Subtract 2 integers",
            "args":[ { "type":"uint64" }, { "type":"uint64" } ],
            "returns": {"type":"uint64"}
        },
```

At a high level, the ABI allows contracts to define an API so clients know exactly what the Smart Contract is expecting to be passed (like a "*Smart Contract Swagger*").

Algorand

TEAL

AVM assembly-like language

Smart Signature Example

Suppose we want to develop a **Smart Signature** that approves a transaction **if and only if**:

1. is "*Payment*" **type** transaction;
2. the **receiver** is a specific "*ADDR*";
3. **fees** are less or equal to "*1000 microALGO*";
4. first **argument** is equal to "*bianconiglio*";
5. **amount** is equal to "*42 ALGO*";
6. or **amount** is equal to "*77 ALGO*";

Where do we start?

Smart Signature as “Transaction Observer”

Smart Signatures can be defined as a *“transactions’ observers”*: programs that meticulously check all **fields in the transaction** (or in a group of transactions) that intend to *“approve”* or *“reject”* based on TEAL logic.

To translate those 6 **semantically defined** example’s conditions into **TEAL** we need to check which transaction fields are going to be controlled by Smart Signature’s logic.

Let's start with the translation...

Translating conditions into TEAL…

1. is "*Payment*" **type** transaction;

TxType	*required*	string	`"type"`	Specifies the type of transaction. This value is automatically generated using any of the developer tools.

```
txn TypeEnum
int 1
==
```
1

Translating conditions into TEAL...

2. the **receiver** is a specific "*ADDR*";


```
txn Receiver
addr AOC7...
==
```

2

Translating conditions into TEAL…

3. **fees** are less or equal to "*1000 microALGO*";

Fee	*required*	uint64	`"fee"`	Paid by the sender to the FeeSink to prevent denial-of-service. The minimum fee on Algorand is currently 1000 microAlgos.

```
txn Fee
int 1000
<=
```

3

Translating conditions into TEAL...

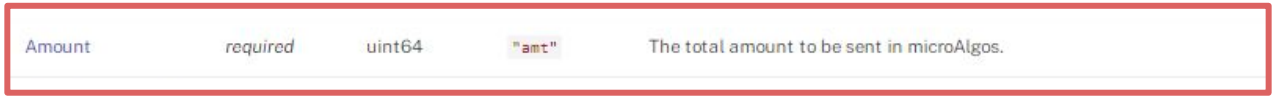
4. first **argument** is equal to "*bianconiglio*";

`arg`	push Args[N] value to stack by index

```
arg 0                                    4
byte base64 "YmlhbmNvbmlnbGlv"
==
```

Translating conditions into TEAL…

5. **amount** is equal to "*42 ALGO*";



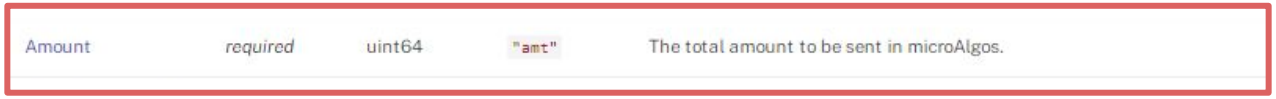
Amount	*required*	uint64	`"amt"`	The total amount to be sent in microAlgos.

```
txn Amount
int 42000000
==
```

5

Translating conditions into TEAL…

6. **amount** is equal to “*77 ALGO*”;



Amount	*required*	uint64	`"amt"`	The total amount to be sent in microAlgos.

```
txn Amount
int 77000000
==
```

6

Logic connectors...

```
txn TypeEnum
int 1
==                                          1

txn Receiver
addr AOC7...
==                                          2

txn Fee
int 1000
<=                                          3

arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==                                          4

txn Amount
int 42000000
==                                          5

txn Amount
int 77000000
==                                          6
```

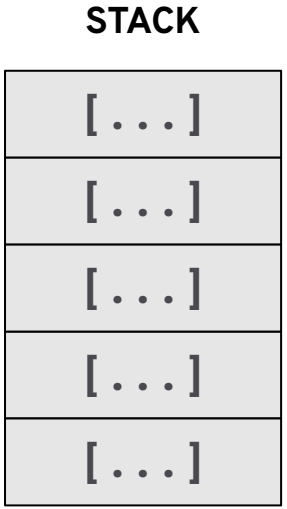
STACK

Logic connectors...

```
txn TypeEnum
int 1
==                                          1
txn Receiver
addr AOC7...
==                                          2
&&
txn Fee
int 1000
<=                                          3
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==                                          4
&&
&&
txn Amount
int 42000000
==                                          5
txn Amount
int 77000000
==                                          6
||
&&
```

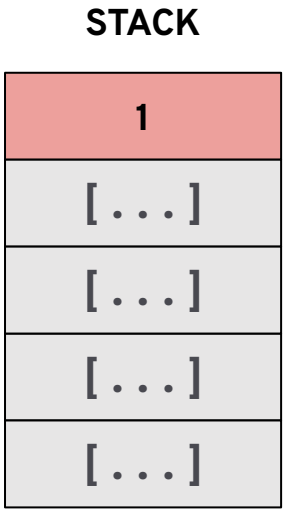
This is probably the most complex phase in **TEAL** programming, because you need to keep in mind the **state of the stack**.

This phase is drastically simplified with the use of **PyTEAL**, Python binding for TEAL, which automatically performs this concatenation, saving us the effort of thinking about the state of the stack.



Execution...

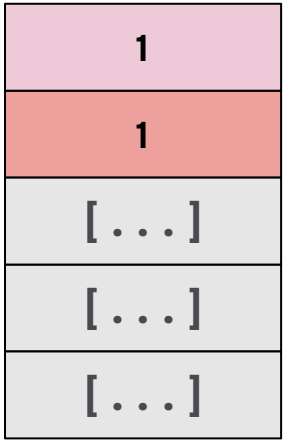
```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Execution...

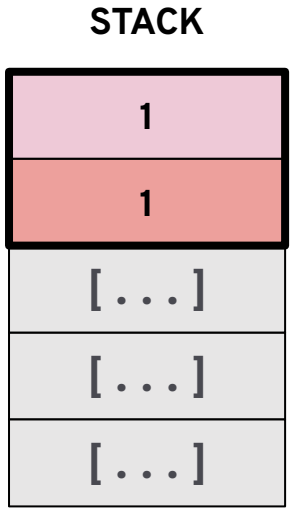
```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK



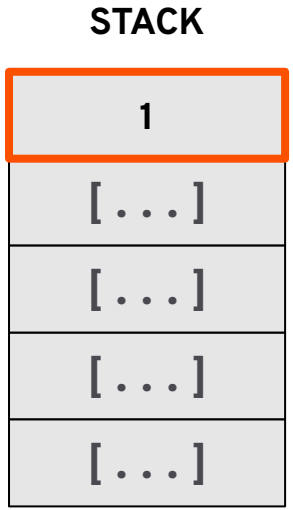
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK

AOC7...
1
[. . .]
[. . .]
[. . .]

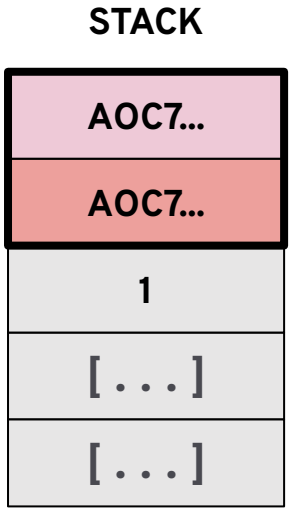
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK

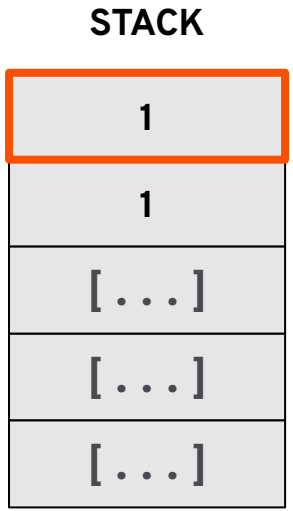
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



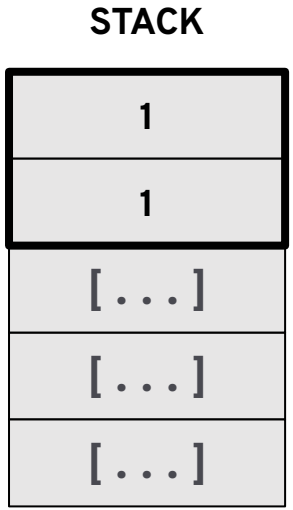
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



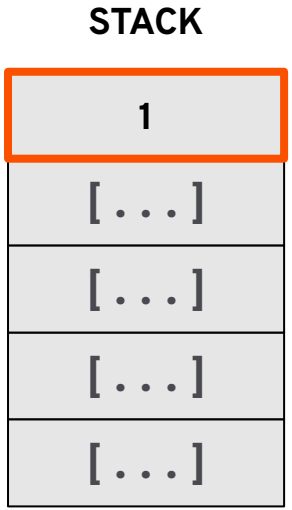
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK

1000
1
[. . .]
[. . .]
[. . .]

Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

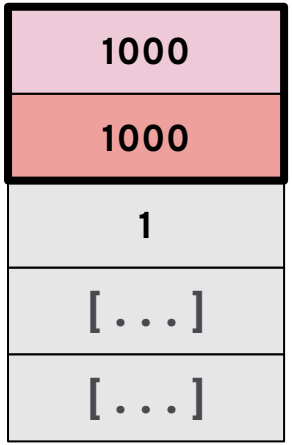
STACK

1000
1000
1
[. . .]
[. . .]

Execution...

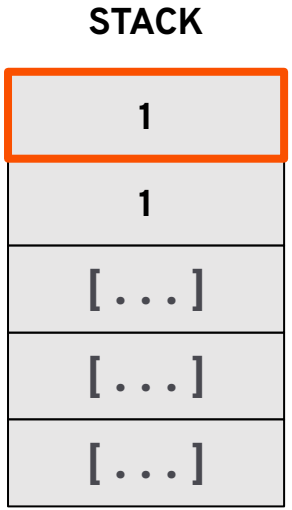
```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK



Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK

bianconiglio
1
1
[. . .]
[. . .]

Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK

bianconiglio
bianconiglio
1
1
[. . .]

Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

Execution...

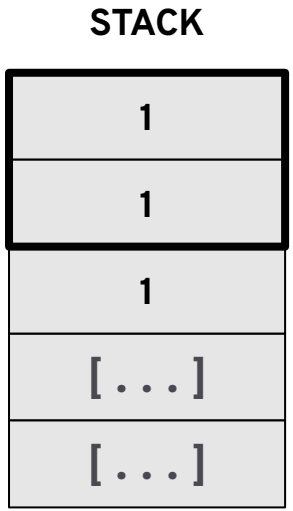
```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK

1
1
1
[. . .]
[. . .]

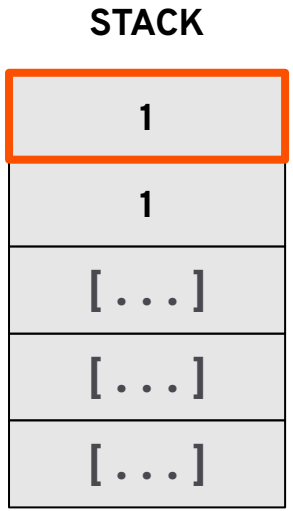
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



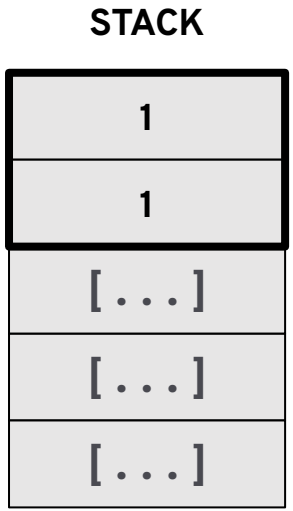
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



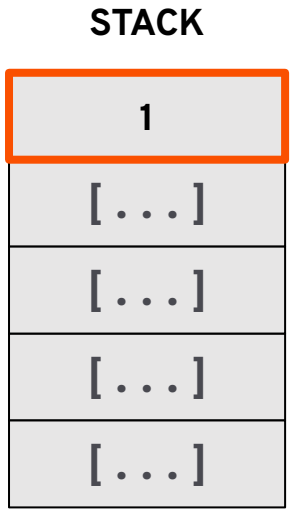
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK

42000000
1
[...]
[...]
[...]

Execution...

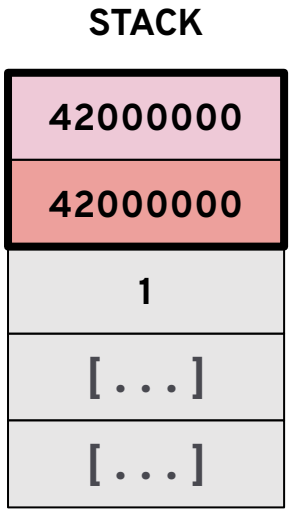
```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK

42000000
42000000
1
[. . .]
[. . .]

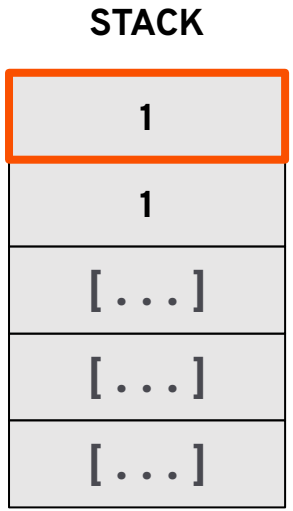
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK

42000000
1
1
[. . .]
[. . .]

Execution...

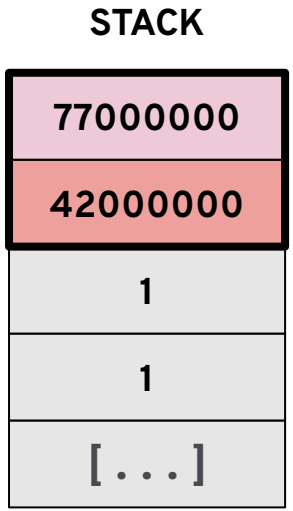
```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK

77000000
42000000
1
1
[. . .]

Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Execution...

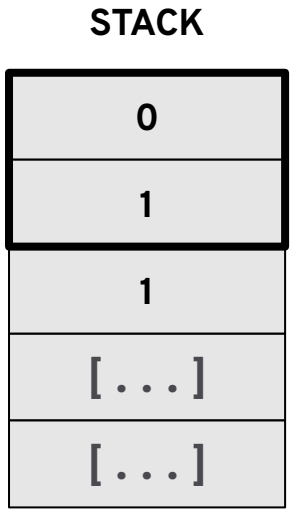
```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```

STACK

0
1
1
[. . .]
[. . .]

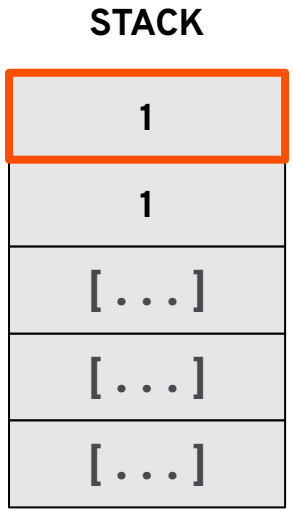
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



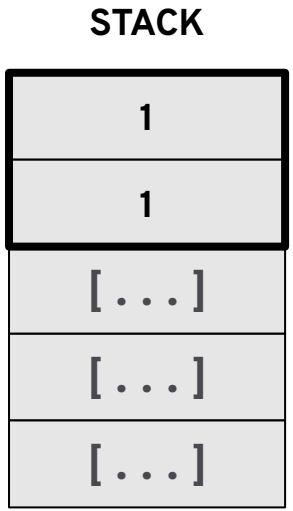
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



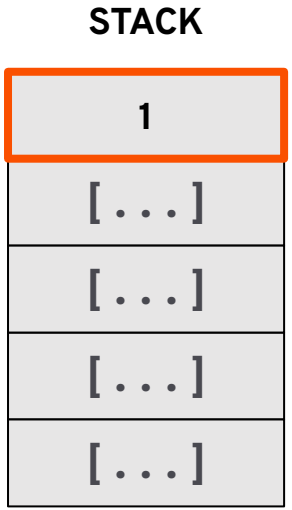
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



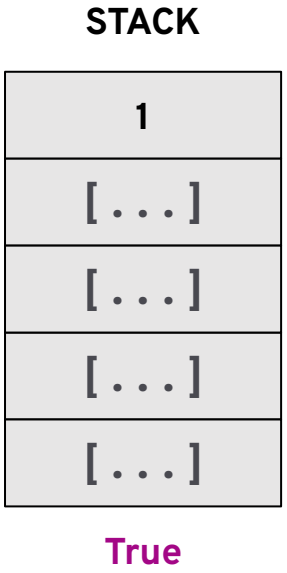
Execution...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "YmlhbmNvbmlnbGlv"
==
&&
&&
txn Amount
int 42000000
==
txn Amount
int 77000000
==
||
&&
```



Conclusion...

```
txn TypeEnum
int 1
==
txn Receiver
addr AOC7...
==
&&
txn Fee
int 1000
<=
arg 0
byte base64 "
==
&&
&&
txn Amount
int
==
txn Amount
int
==
||
&&
```



Algorand

PyTEAL

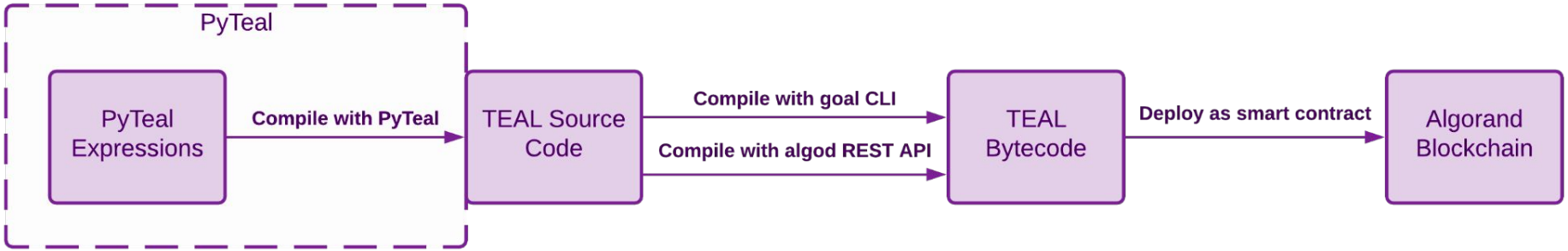
Writing Smart Contracts with Python

Credits to Jason Paulos for the PyTEAL presentation

What's *PyTEAL*?

PyTEAL is a **Python** language binding for **Algorand Virtual Machine**.

PyTEAL allows **Smart Contracts** and **Smart Signatures** to be written in Python and then compiled to **TEAL**.



It’s easier with PyTEAL!

TEAL Source Code

```
txn Receiver
addr AOC7...
==
txn Amount
int 1000
<=
&&
```

COMPILE...

AVM bytecode

PyTEAL Source Code

```
And(
        Txn.Receiver == Addr(AOC7...),
        Txn.Amount <= Int(1000),
)
```

COMPILE...

TEAL Source Code

```
txn Receiver
addr AOC7...
==
txn Amount
int 1000
<=
&&
```

COMPILE...

AVM bytecode

Smart Signature written in PyTEAL

```python
import base64
from pyteal import *

"""Esempio ASC1 in PyTeal"""


def asc1(tmpl_receiver):
    is_payment = Txn.type_enum() == Int(1)
    is_correct_receiver = Txn.receiver() == Addr(tmpl_receiver)
    max_fee = Txn.fee() <= Int(1000)
    follow_the = Arg(0) == Bytes(
        "base64", str(base64.b64encode('bianconiglio'.encode()), 'utf-8'))
    amount_option1 = Txn.amount() == Int(42000000)
    amount_option2 = Txn.amount() == Int(77000000)
    is_correct_amount = Or(amount_option1, amount_option2)
    asc1_logic = And(is_payment,
                     is_correct_receiver,
                     max_fee,
                     follow_the,
                     is_correct_amount)
    return asc1_logic
```

PyTEAL Basics - Intro

PyTEAL expressions represent an **abstract syntax tree** (AST)

You're writing **Python code** that produces **TEAL code**.

```
from pyteal import *

program = ...
teal_source = compileTeal(program, mode=Mode.Application, version=5)
```

PyTEAL Basics - Types (1/2)

Two basic types:

- **uint64**
- **byte strings**

```
i = Int(5)
x = Bytes("content")
y = Bytes(b"\x01\x02\x03")
z = Bytes("base16", "05")
```

PyTEAL Basics - Types (2/2)

Conversion between types

- **Itob** - integer to bytes (8-byte big endian)
- **Btoi** - bytes to integer

```
Itob(i) # produces the byte string 0x0000000000000005
Btoi(z) # produces the integer 5
```

PyTEAL Basics - Math operators

Python **math** operators

```
i = Int(10)
j = i * Int(2) + Int(1)
k = And(Int(1), Or(Int(1), Int(0)))
```

PyTEAL Basics - Byte string manipulation

Byte string **manipulation**

```
x = Bytes("content")
y = Concat(Bytes("example "), x) # "example content"
z = Substring(y, Int(2), Len(y)) # "ample content"
```

PyTEAL Basics - Crypto utilities

Built-in **crypto** utilities

```
h_sha256 = Sha256(z)
h_sha512_256 = Sha512_256(z)
h_keccak = Keccak256(z)
```

PyTEAL Basics - Fields (1/3)

Fields from the **current transaction**

```
Txn.sender()
Txn.accounts.length()
Txn.application_args.length()
Txn.accounts[1]
Txn.application_args[0]
Txn.group_index()
```

PyTEAL Basics - Fields (2/3)

Fields from transactions in the **current atomic group**

```
Gtxn[0].sender()
Gtxn[Txn.group_index() - Int(1)].sender()
Gtxn[Txn.group_index() - Int(1)].accounts[2]
```

PyTEAL Basics - Fields (3/3)

Fields from **execution context**

```
Global.group_size()
Global.round() # current round number
Global.latest_timestamp() # UNIX timestamp of last round
```

PyTEAL Basics - Logs

Log publicly viewable messages to the chain

```
Log(Bytes("message"))
```

PyTEAL Basics - State (1/2)

Global - one instance per application

```
App.globalPut(Bytes("status"), Bytes("active")) # write to global key "status"

status = App.globalGet(Bytes("status")) # read global key "status"

App.globalDel(Bytes("status")) # delete global key "status"
```

PyTEAL Basics - State (2/2)

Local - one instance per opted-in account per application

```
App.localPut(Txn.sender(), Bytes("level"), Int(1)) # write to sender's local key "level"
App.localPut(Txn.accounts[1], Bytes("level"), Int(2))  # write to other account's local key "level"

sender_level = App.localGet(Txn.sender(), Bytes("level"))  # read from sender's local key "level"

App.localDel(Txn.sender(), Bytes("level"))  # delete sender's local key "level"
```

PyTEAL Basics - Control Flow (1/5)

Approve the transaction and **immediately exit**

```
Approve()
```

Reject the transaction and **immediately exit**

```
Reject()
```

PyTEAL Basics - Control Flow (2/5)

Multiple expressions can be joined into a **sequence**

```
program = Seq(
    App.globalPut(Bytes("count"), App.globalGet(Bytes("count")) + Int(1)),
    Approve()
)
```

PyTEAL Basics - Control Flow (3/5)

Basic conditions can be expressed with **If**, **Then**, **Else**, **Elself**

```
program = Seq(
    If(App.globalGet(Bytes("count")) == Int(100))
    .Then(
        App.globalPut(Bytes("100th caller"), Txn.sender())
    )
    .Else(
        App.globalPut(Bytes("not 100th caller"), Txn.sender())
    ),
    App.globalPut(Bytes("count"), App.globalGet(Bytes("count")) + Int(1)),
    Approve(),
)
```

PyTEAL Basics - Control Flow (4/5)

Larger conditions can be expressed with **Cond**

```
program = Cond(
    [Txn.application_id() == Int(0), on_create],
    [Txn.on_completion() == OnComplete.UpdateApplication, on_update],
    [Txn.on_completion() == OnComplete.DeleteApplication, on_delete],
    [Txn.on_completion() == OnComplete.OptIn, on_opt_in],
    [Txn.on_completion() == OnComplete.CloseOut, on_close_out],
    [Txn.on_completion() == OnComplete.NoOp, on_noop],
    # error if no conditions are met
)
```

PyTEAL Basics - Control Flow (5/5)

Loops can be expressed with **For** and **While**

```
i = ScratchVar(TealType.uint64)

on_create = Seq(
    For(i.store(Int(0)), i.load() < Int(16), i.store(i.load() + Int(1)))
    .Do(
        App.globalPut(Concat(Bytes("index"), Itob(i.load())), Int(1))
    ),
    Approve(),
)
```

PyTEAL Basics - Subroutines (1/2)

Sections of code can be put into **subroutines** (Python decorators)

```
@Subroutine(TealType.uint64)
def isEven(i):
    return i % Int(2) == Int(0)

App.globalPut(Bytes("value_is_even"), isEven(Int(10)))
```

PyTEAL Basics - Subroutines (2/2)

Recursion is allowed

```
@Subroutine(TealType.uint64)
def recursiveIsEven(i):
    return (
        If(i == Int(0))
        .Then(Int(1))
        .ElseIf(i == Int(1))
        .Then(Int(0))
        .Else(recursiveIsEven(i - Int(2)))
    )
```

PyTEAL Basics - Inner Transactions (1/3)

Every **application** has control of an **account**

```
Global.current_application_address()
```

PyTEAL Basics - Inner Transactions (2/3)

Applications can **send transactions** from this **account**

```
Seq(
    InnerTxnBuilder.Begin(),
    InnerTxnBuilder.SetFields(
        {
            TxnField.type_enum: TxnType.Payment,
            TxnField.receiver: Txn.sender(),
            TxnField.amount: Int(1_000_000),
        }
    ),
    InnerTxnBuilder.Submit() # send 1 Algo from the app account to the transaction sender
)
```

PyTEAL Basics - Inner Transactions (3/3)

```
appAddr = Global.current_application_address()
Seq(
    InnerTxnBuilder.Begin(),
    InnerTxnBuilder.SetFields(
        {
            TxnField.type_enum: TxnType.AssetConfig,
            TxnField.config_asset_name: Bytes("PyTEAL Coin"),
            TxnField.config_asset_unit_name: Bytes("PyTEAL"),
            TxnField.config_asset_url: Bytes("https://pyteal.readthedocs.io/"),
            TxnField.config_asset_decimals: Int(6),
            TxnField.config_asset_total: Int(800_000_000),
            TxnField.config_asset_manager: appAddr,
        }
    ),
    InnerTxnBuilder.Submit(), # create a PyTEAL Coin asset
    App.globalPut(Bytes("PyTealCoinId"), InnerTxn.created_asset_id()) # remember the asset ID
)
```

PyTeal ABI Support

PyTeal natively supports the ABI standard: it handles methods signatures and ABI JSON auto-generation.

```python
from pyteal import *

@Subroutine(TealType.uint64)
def minimum(a: abi.Uint64, b: abi.Uint64) -> Expr:
    """Return the minimum value of the two arguments."""
    return (
        If(a.get() < b.get())
        .Then(a.get())
        .Else(b.get())
    )
```

Beaker: Smart Contract development framework

Beaker improves the PyTeal Smart Contract development experience, making it **simpler**, **cleaner** and **faster** both to **write and debug**:

- Provides a **standard way to organize code** by using a **Application** class with **@external** (methods) and **@internal** (subroutine) decorators, with utilities for call **authorization**;
- Provides an **ApplicationClient** to deal with common needs like *creating/opting-in to/calling* methods;
- Provides a **standard way to declare Global and Local sate**, with typed values as class variables and utilities as: *default values, static variables, variable initialization, etc.*;
- Improves the *pc=xxx* error message using the **SourceMap endpoint** during compilation and mapping the *pc* back to the source TEAL with a **LogicException**.

A pythonic Algorand stack

PyCharm IDE & AlgoDEA plug-in

Algorand Sandbox Docker in dev mode

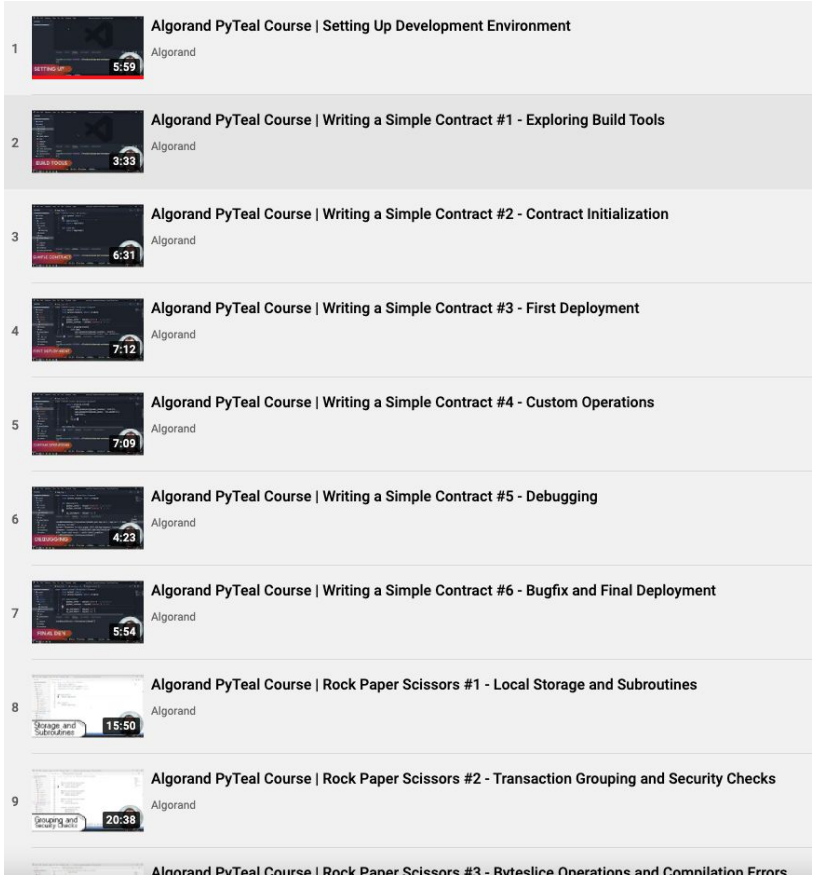
Algorand Python SDK

PyTEAL & Beaker

PyTest for Smart Contracts unit-tests and e2e-tests

TEAL Debugger

“Zero to Hero PyTEAL” crash course



Join and learn in 12 lessons!

Algorand

DAPP EXAMPLE & USE CASES

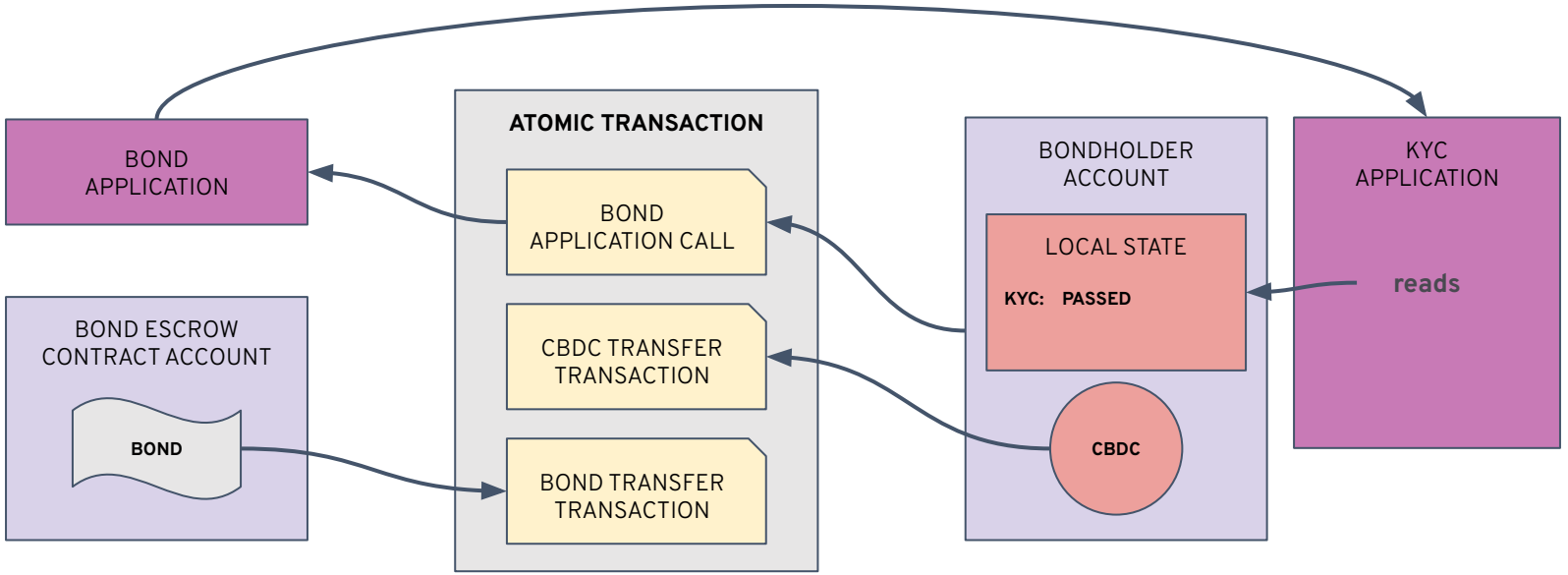
Example: zero coupon bond

Zero coupon bond

1. Bondholder purchase a bond (only if KYC has been verified)
2. Bond application checks for bond's maturity
3. Bond Issuer re-pays the Bondholder at given rate

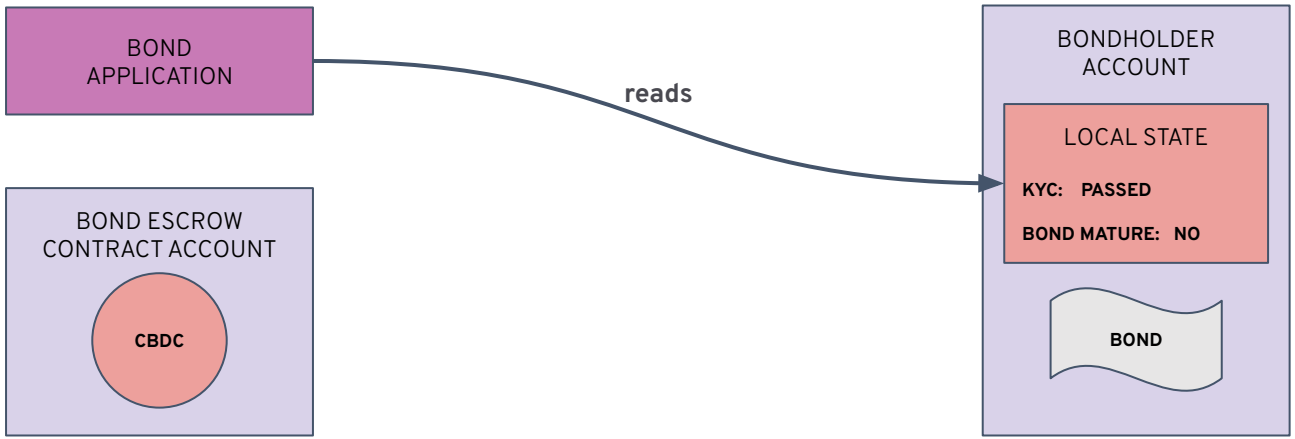
Zero coupon bond

1. Bondholder purchase a bond (only if KYC has been passed)



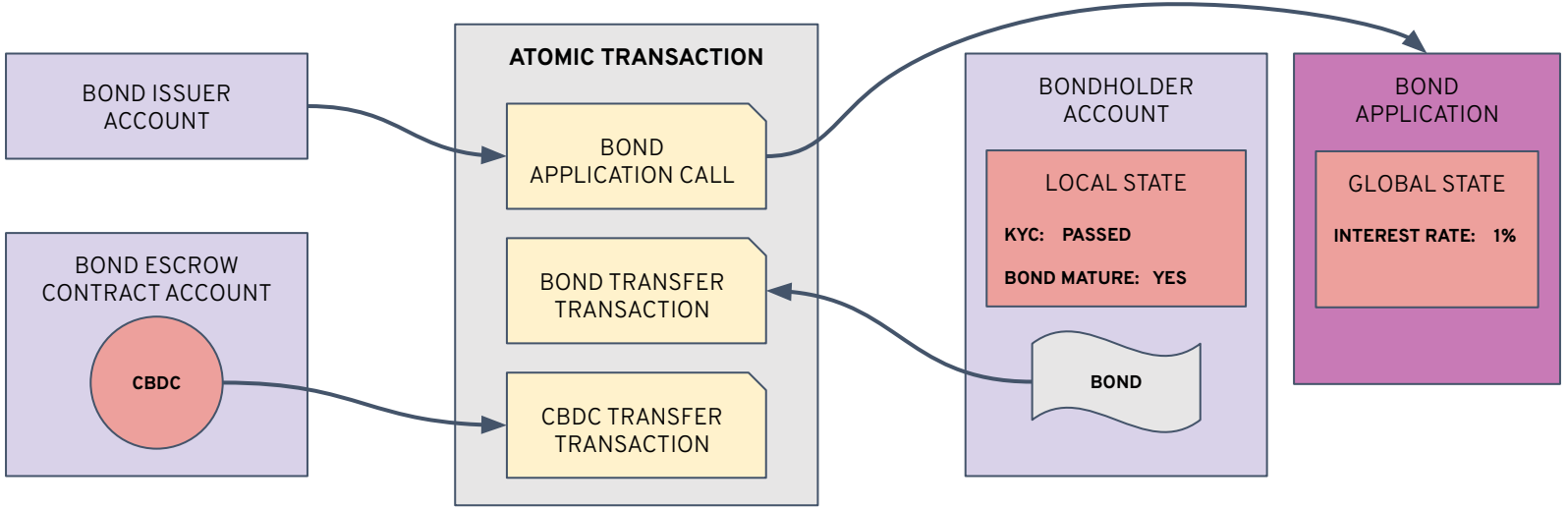
Zero coupon bond

2. Bond application checks for bond's maturity



Zero coupon bond

3. Bond Issuer re-pays the Bondholder at given rate



Algorand

Thanks to the contributors!

- Jason Paulos
- Pietro Grassano