An Introduction to Smart Contracts on Nexa

What is Nexa ?

Nexa Overview

- Fair-launched L1 blockchain
- UTXO-based PoW model
- Native token (NEXA) with denominations in satoshis
- Group tokens (fungible and NFTs)
- High tx throughput with very low fees
- Script-based smart contracts with introspection capabilities
- Brief history: 

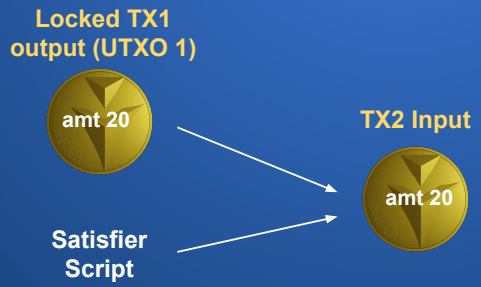
Foundation Concepts

UTXOs

- A UTXO is a *coin* - it has an amount and a locking script (*constraint script*)

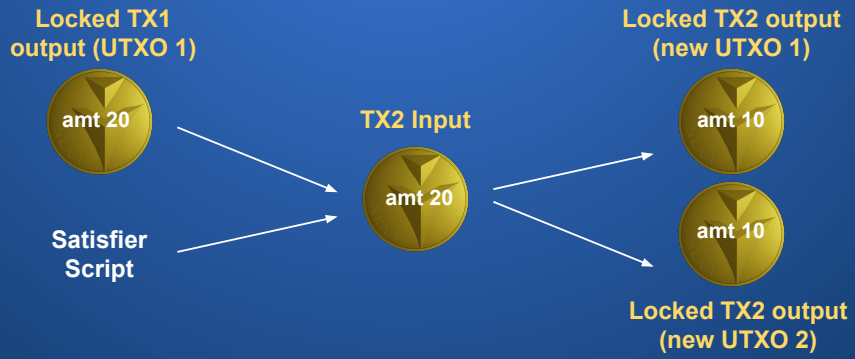
UTXOs

- A UTXO is a *coin* - it has an amount and a locking script (*constraint script*)
- To spend it, you provide a *satisfier script* (spending proof) that makes the combined script evaluate to true



UTXOs

- A UTXO is a *coin* - it has an amount and a locking script (*constraint script*)
- To spend it, you provide a *satisfier script* (spending proof) that makes the combined script evaluate to true
- Once spent, a UTXO is consumed and new UTXOs are created



UTXOs

- A UTXO is a *coin* - it has an amount and a locking script (*constraint script*)
- To spend it, you provide a *satisfier script* (spending proof) that makes the combined script evaluate to true
- Once spent, a UTXO is consumed and new UTXOs are created

Analogy: Imagine you hand over a $20 bill to a merchant, the $20 bill is permanently destroyed, and two new $10 bills are printed. The merchant keeps a new $10 bill and you receive a new $10 bill as change

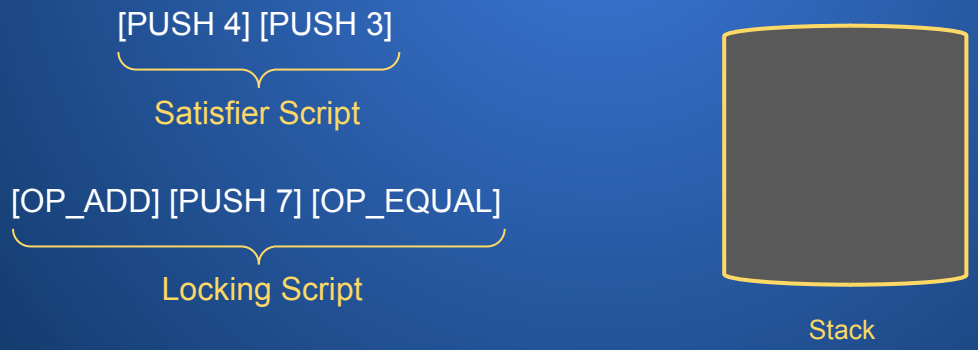
A Note on Nexa Contracts

IMPORTANT TO NOTE:

On NEXA, there is no distinction between 'sending coins' and 'executing a smart contract.' Both are the same thing - *proving you have the right to spend a UTXO by satisfying its constraint (locking) script*.

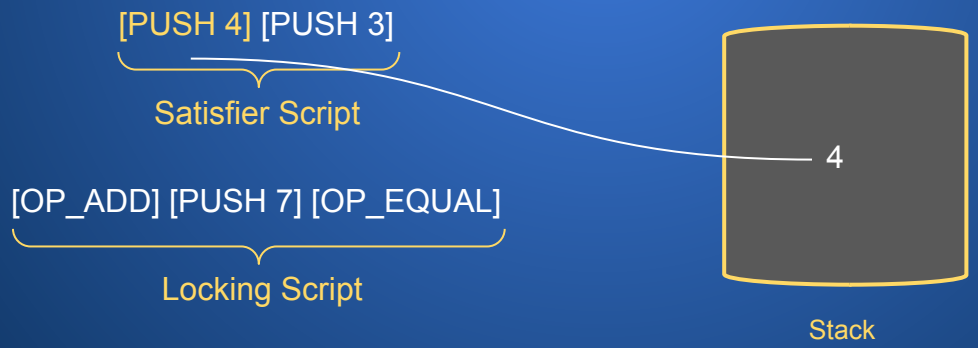
Stack-Based Script Evaluation

- Scripts execute on a stack machine (push data, execute operations)
- No loops - scripts always terminate (bounded execution)



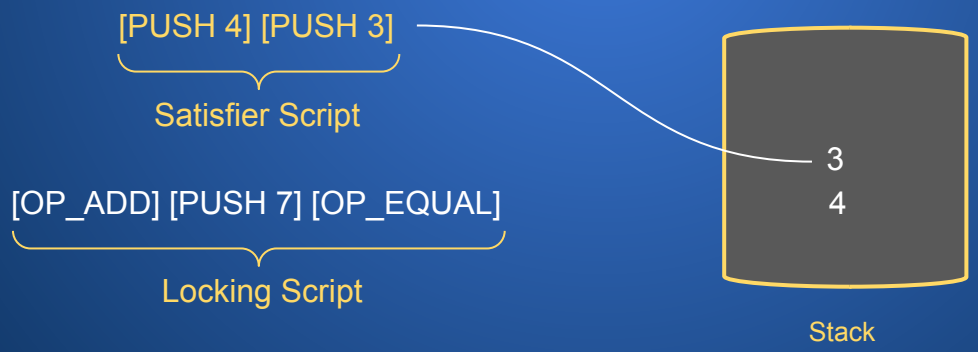
Stack-Based Script Evaluation

- Scripts execute on a stack machine (push data, execute operations)
- No loops - scripts always terminate (bounded execution)
- Satisfier script runs first, pushes data onto the stack



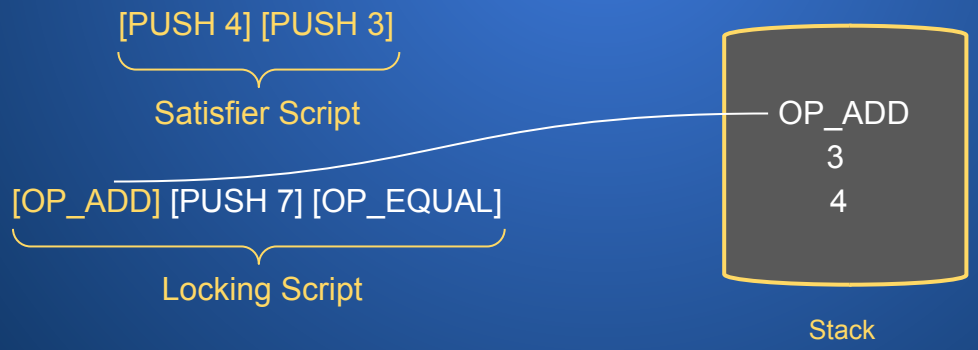
Stack-Based Script Evaluation

- Scripts execute on a stack machine (push data, execute operations)
- No loops - scripts always terminate (bounded execution)
- Satisfier script runs first, pushes data onto the stack



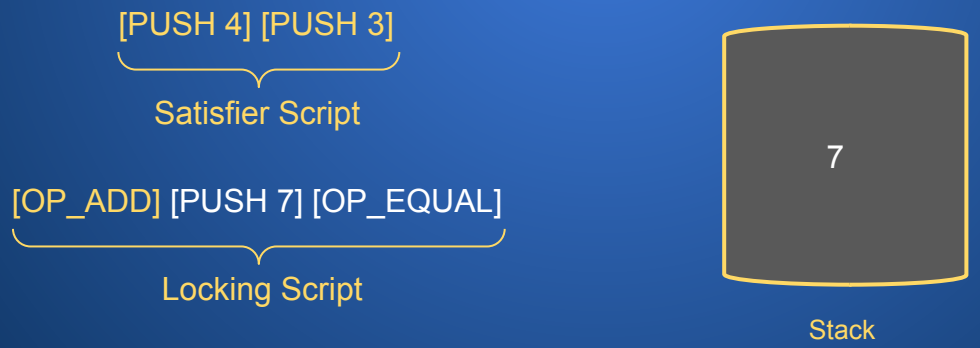
Stack-Based Script Evaluation

- Scripts execute on a stack machine (push data, execute operations)
- No loops - scripts always terminate (bounded execution)
- Satisfier script runs first, pushes data onto the stack
- Then the constraint (locking) script runs and validates conditions



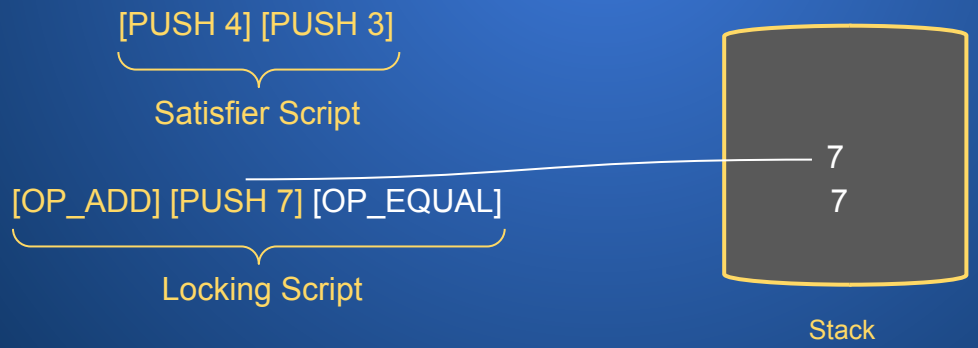
Stack-Based Script Evaluation

- Scripts execute on a stack machine (push data, execute operations)
- No loops - scripts always terminate (bounded execution)
- Satisfier script runs first, pushes data onto the stack
- Then the constraint (locking) script runs and validates conditions



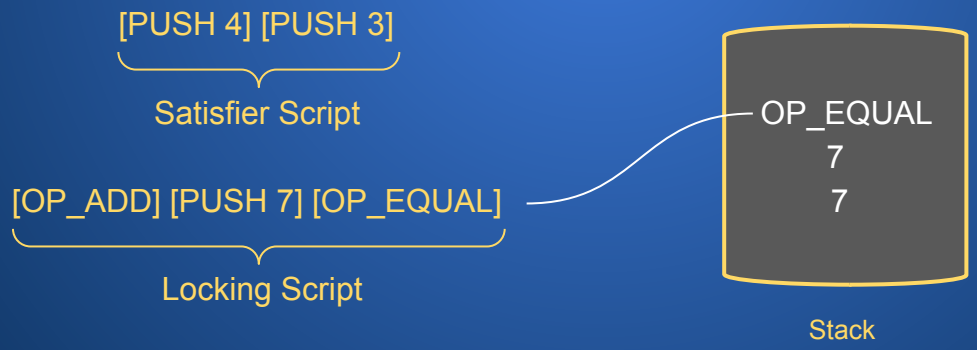
Stack-Based Script Evaluation

- Scripts execute on a stack machine (push data, execute operations)
- No loops - scripts always terminate (bounded execution)
- Satisfier script runs first, pushes data onto the stack
- Then the constraint (locking) script runs and validates conditions



Stack-Based Script Evaluation

- Scripts execute on a stack machine (push data, execute operations)
- No loops - scripts always terminate (bounded execution)
- Satisfier script runs first, pushes data onto the stack
- Then the constraint (locking) script runs and validates conditions



Stack-Based Script Evaluation

- Scripts execute on a stack machine (push data, execute operations)
- No loops - scripts always terminate (bounded execution)
- Satisfier script runs first, pushes data onto the stack
- Then the constraint (locking) script runs and validates conditions
- If the stack ends empty (clean stack), the spend is valid

[PUSH 4] [PUSH 3]

Satisfier Script

[OP_ADD] [PUSH 7] [OP_EQUAL]

Locking Script

Stack

ALL TRANSACTIONS ARE FULLY VALIDATED VIA NETWORK-WIDE CONSENSUS

Key OP Codes

- *OP_PARSE*: NEXA's introspection opcode (used on P2T transactions)
  - P2T Field 3: Template hash
  - P2T Field 4: Args hash
  - P2T Fields 8+: Individual "visible args"

- *OP_VERIFY*

- *OP_CHECKLOCKTIMEVERIFY*

- *OP_CHECKSIGVERIFY / OP_CHECKDATASIGVERIFY*

- Standard arithmetic and comparison ops (ADD, SUB, DIV, GREATERTHAN, etc.)

Decentralized Oracles

- Trusted external data providers that sign statements about real-world events
- On NEXA: oracle signs a structured message
- Contract verifies the signature against the oracle's well-known public key using checkDataSigVerify
- Oracle doesn't know or care about any contracts - it simply publishes signed facts
- Oracles can be used to sign the outcome of *ANY* event or piece of data

Check out the NEXA Price Oracle here:
https://wallywallet.org/_api/v0/now/hourlyavg/usdt/nexa

Decentralized Oracles


```
{
        "type": "Hourly Average",
        "msg": {
                "data": "4e4558415553445 41f76ff690000000001886ea1e01000000",
                "Signature":"8b3adcf3673dd081872131820f6e817be017b7109cdbb53481f9dcebb67ea96518d25f470608ee0588102c30cd29b14935421feae0c08d5305b6c39b6dd17220"
        },
        "epochSeconds": 1778349599,
        "Price":"0.0000004813653528",
        "pairPriceUnit": "USDT/NEXA"
}
```

Nexa Smart Contracts vs Ethereum Smart Contracts

Nexa vs EVM Contracts

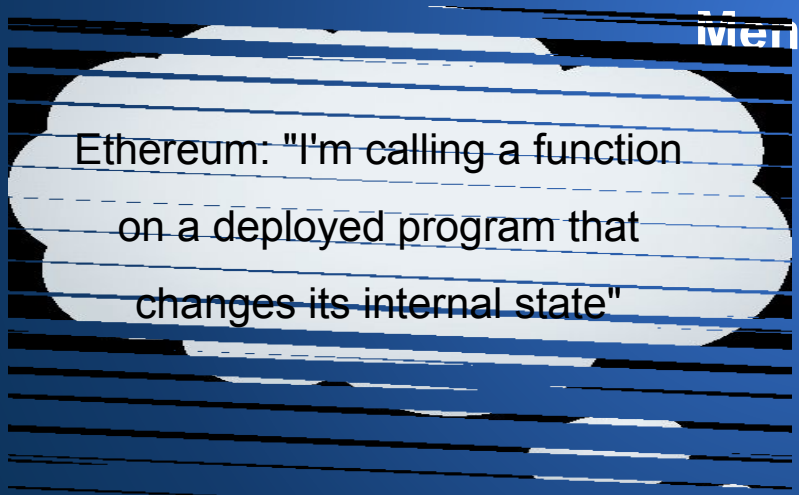
Aspect	NEXA (UTXO Script)	Ethereum (EVM)
Model	UTXO-based	Account-based
Execution	Stack-based script evaluated per-input at spend time	Turing-complete VM with persistent storage
State	Stateless	Stateful
Parallelism	Inputs validated independently (parallelizable)	Sequential execution within blocks
Gas / Fees	Minimal standard transaction fees, no dynamic gas fees	Gas model, variable costs based on computation
Introspection	OP_PARSE lets scripts inspect their own transaction's inputs/outputs	Contracts can call other contracts and read state
Deployment	Template hash identifies contract type; parameters embedded in UTXOs	Contract deployed to an address with constructor args
Composability	Delegation pattern	Direct contract-to-contract calls

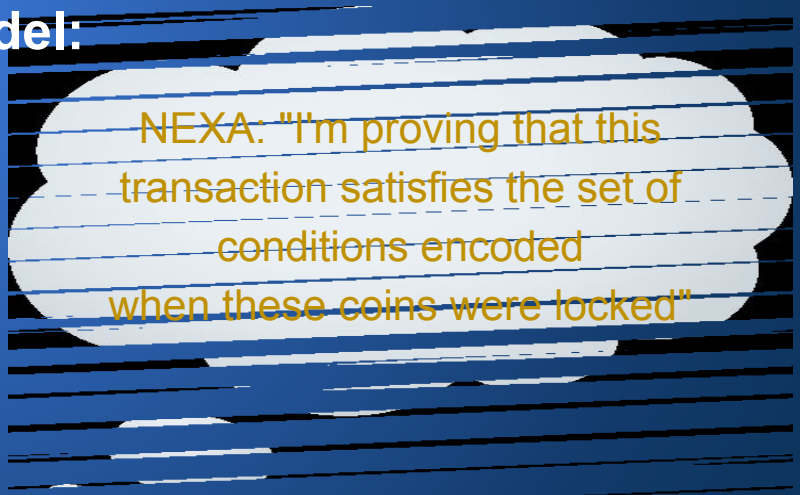
Nexa vs EVM Contracts (cont)

Similarities:

Both support programmable money with enforced rules, cryptographic verification, capable of oracle integration for external data

Mental Model:





Nexa vs EVM Contracts (cont)

Problems With Ethereum Contracts, and How Nexa Provides a Solution:

- Reentrancy Attacks: NEXA scripts don't call other contracts - they validate transaction structure and provable conditions. There's no reentrant execution path to exploit.
- State Mutation Bugs: NEXA contracts are stateless. A UTXO exists or it's been spent. There's no mutable state to corrupt.
- Bounded Execution, No Gas Fees: These issues simply don't exist on NEXA.
- No front-running via MEV: NEXA's UTXO model eliminates this because you'd need to reference the same UTXO, which creates a direct conflict - only a single transaction on the same UTXO can succeed.
- Explicit Value Transfer: NEXA's contract scripts can explicitly check output amounts and destinations. There is no implicit token transfer function which can be tricked. The script verifies the mappings between inputs and outputs directly from the transaction structure - what you see is what you get!
- Audit Complexity: NEXA contracts are comprised of straightforward, short scripts which compile directly to OP codes.

Source Code to On-Chain Contract:
An End-to-End Demonstration

Contract Arg Types

- Template Args:
- Holder Args: hashed together (field 4)
- Holder Public Args: also known as visible args (fields 8+)
- Spender Args:

Fixed Time Exchange Rate Prediction

Lets walk through a contract in which two participants engage in a prediction on the exchange rate of the trading pair NEXA/USDT at a specific future timestamp…

Fixed Time Prediction Contract

Fixed Time Exchange Rate Prediction Delegation

Lets walk through a contract in which a participant makes a *public offer* to engage any willing taker in a prediction on the exchange rate of the trading pair NEXA/USDT at a specific future timestamp…

Fixed Time Prediction Delegation Contract

Smart Contract Participants & Ecosystem Roles

Contract-Related Participant Roles

- Prediction Participants (Alice & Bob)
- Deployer (Platform)
- Facilitator (Any on-chain miner)
- Oracle
- Super Facilitator (Trusted third-party)

Value to the Nexa Ecosystem

- User attraction

- Low barrier

- Demonstrating capability

- Miner revenue

Why Facilitators Matter:

- Permissionless settlement
- Anyone can claim on behalf of the winner
- Creates a natural market for transaction execution
- Miners get additional revenue beyond block rewards

Current & Future Uses of Nexa Smart Contracts

Current Applications

[Nexa Warriors]

[NexPredict]

Other Implementations

Nexa's smart contract capabilities are extensive, and there exists nearly unlimited opportunities for further implementations and applications to DeFi, blockchain gaming, and more! Here are a couple of simple extensions related to currently implemented contracts:

- OP_PARSE's introspection capabilities further utilized for group tokens or NFTs
- Delegation circular-spends
- Delegation-bound Multi-party contracts

Learn More About Nexa and Smart Contracts

[Nexa on Discord]

[Nexa on Gitlab]

[Nexa on Telegram]

[Nexa.org]

[Nexa's Technical Documentation]

Q & A