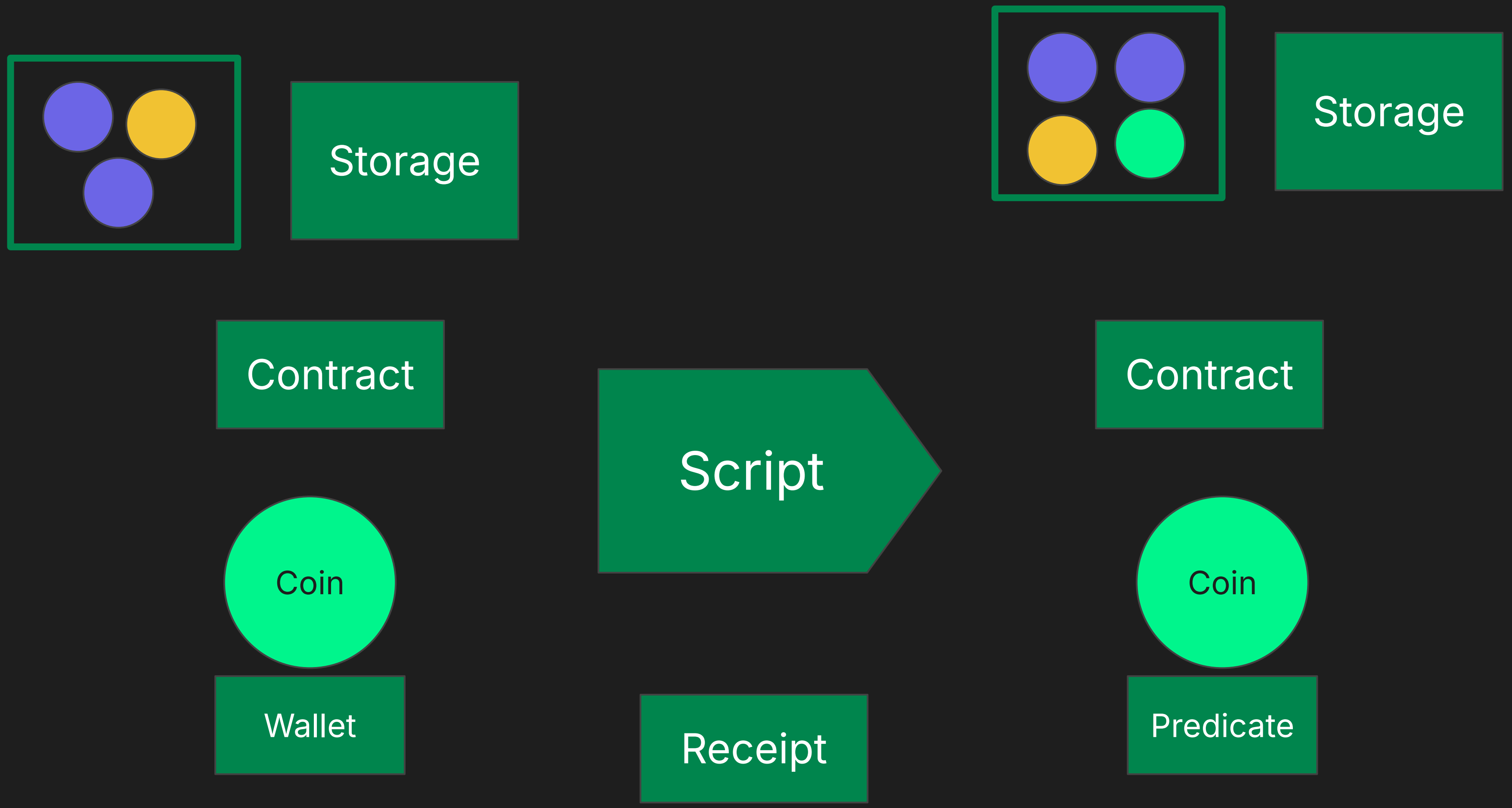
Fuel Deep Dive


Storage
Contract
Coin
Wallet
Script
Receipt
Storage
Contract
Coin
Predicate

Three Ethereum Issues

- 25 TPS (142 TPS max)
- State Bloat
- Only 1 source address, only 1 destination address

Account Model

- Ownership is represented by a balance
- Transactions update the associated account balances

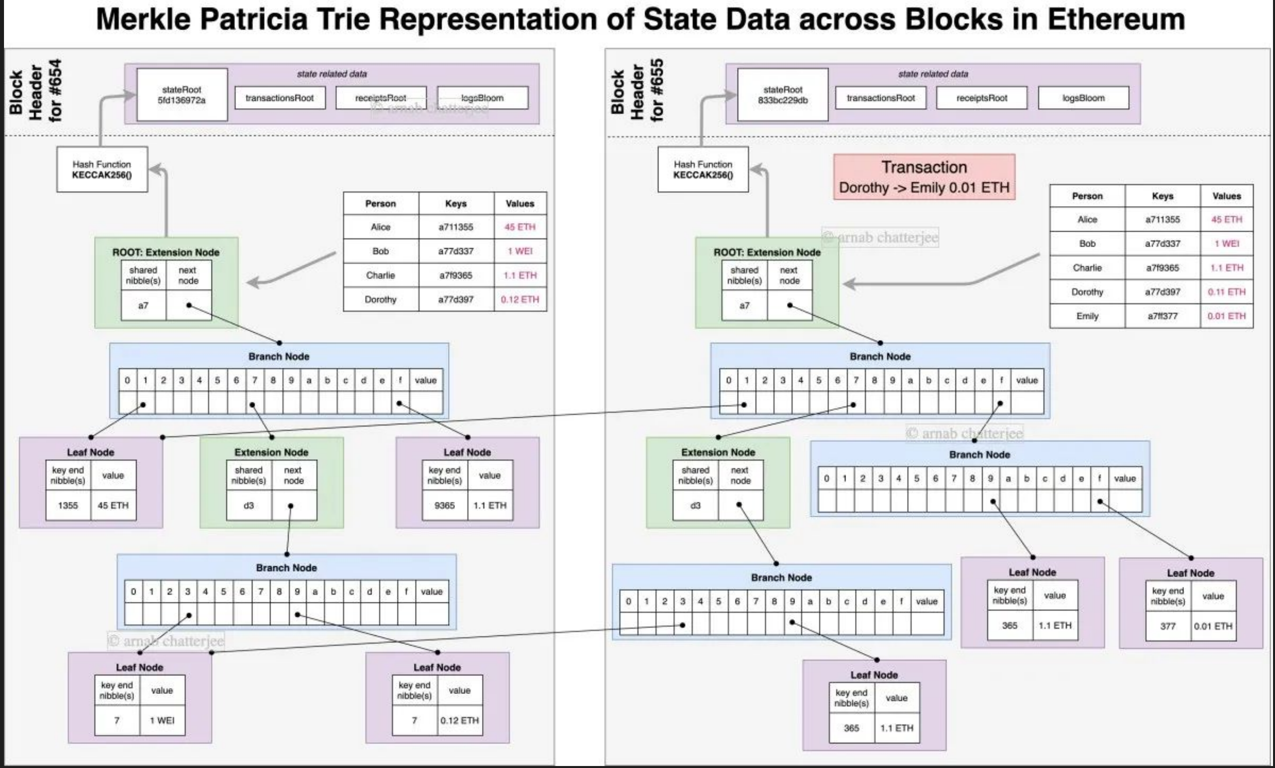
“The world state (state), is a mapping between addresses (160-bit identifiers) and account states (a data structure serialised as RLP, see Appendix B). Though not stored on the blockchain, it is assumed that the implementation will maintain this mapping in a modified Merkle Patricia tree”

A	100

A	52
B	43
C	5

A	0
B	45
C	3

Account Model



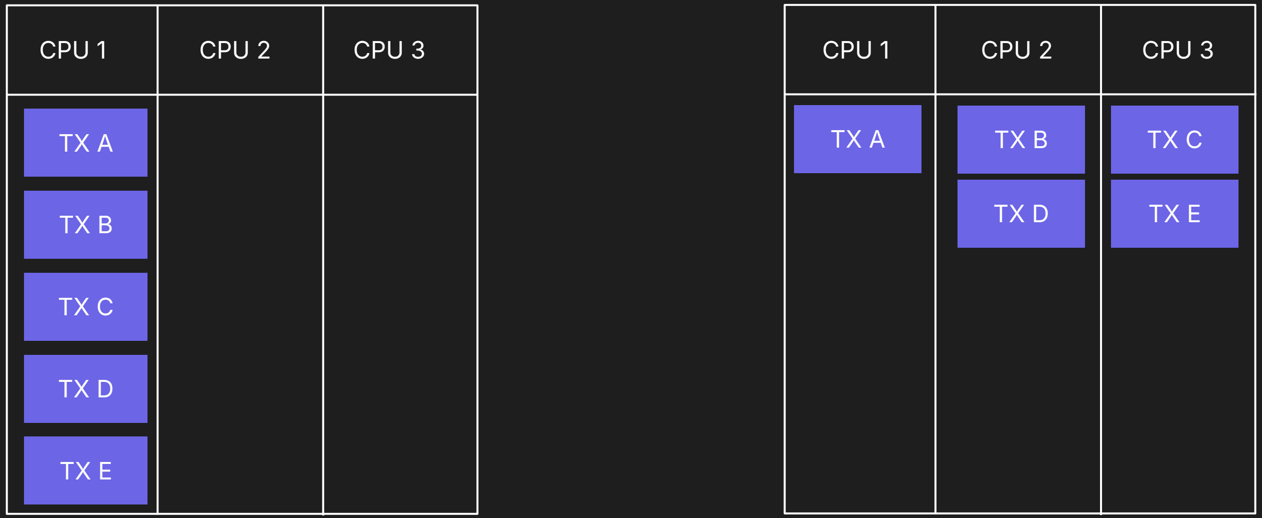
Account Model

In a nutshell:

- The state tree is large database of all accounts
- Light nodes don’t have to store the whole tree
- They can just query the account balance from a full node and get reasonable assurances with a Merkle proof

Account Model

But what about parallelism?



Account Model

Some transactions need to be processed serially.

TX A
Sent 10 to Alice

TX B
Sent the same 10
to Bob

Account Model

We can solve this with state access lists

TX A
Sent 10 from Alice to Bob

Access list:
Alice's balance
Bob's balance

TX B
Sent 10 from Alice to Uniswap

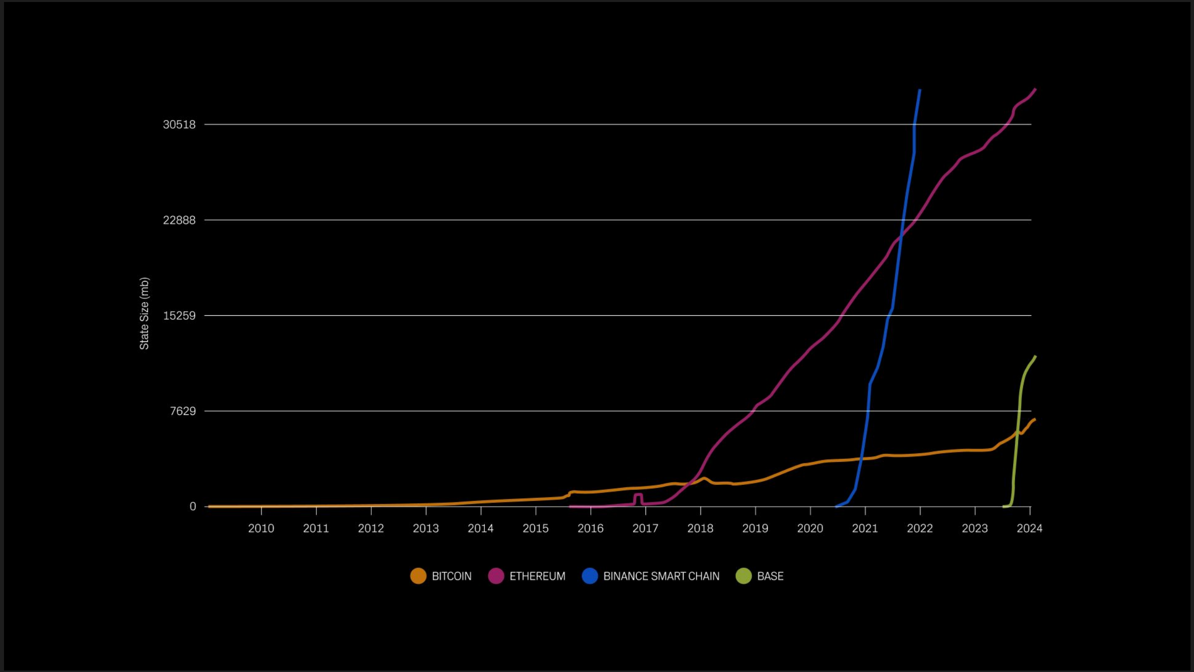
Access list:
Alice's balance
Uniswap balance

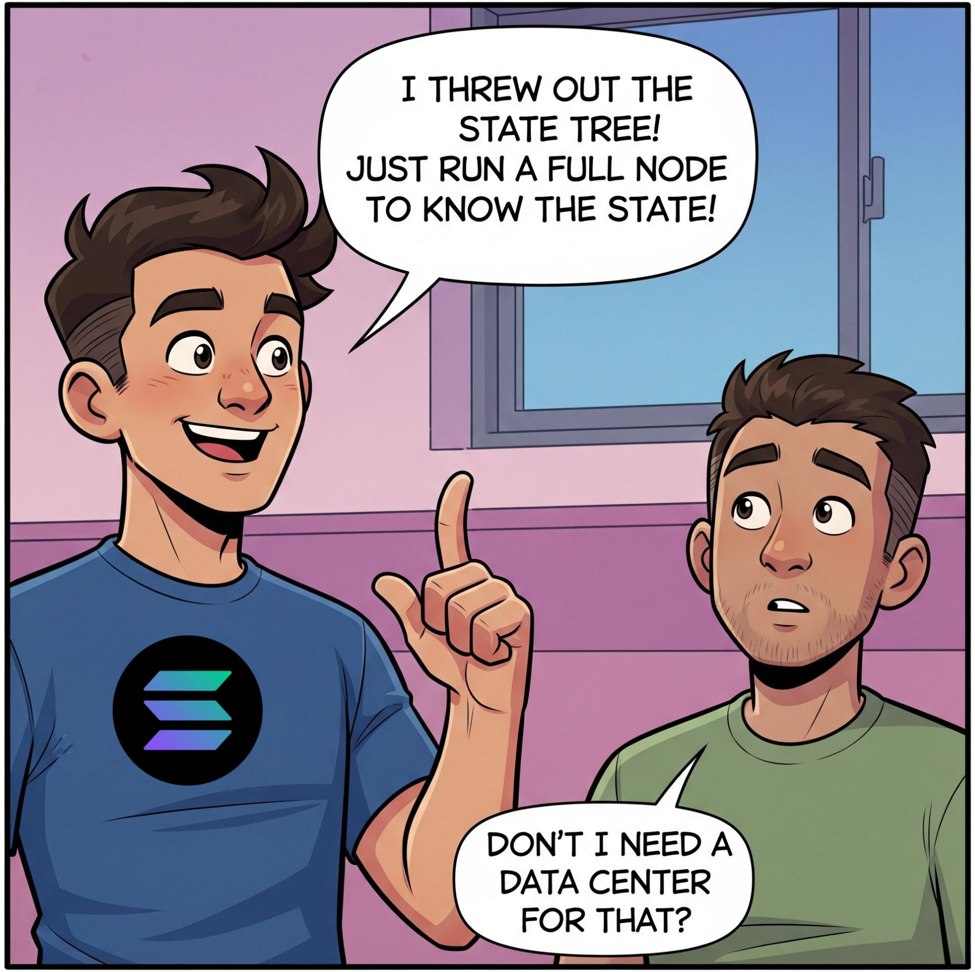
TX C
Sent 10 from Uniswap to Bob

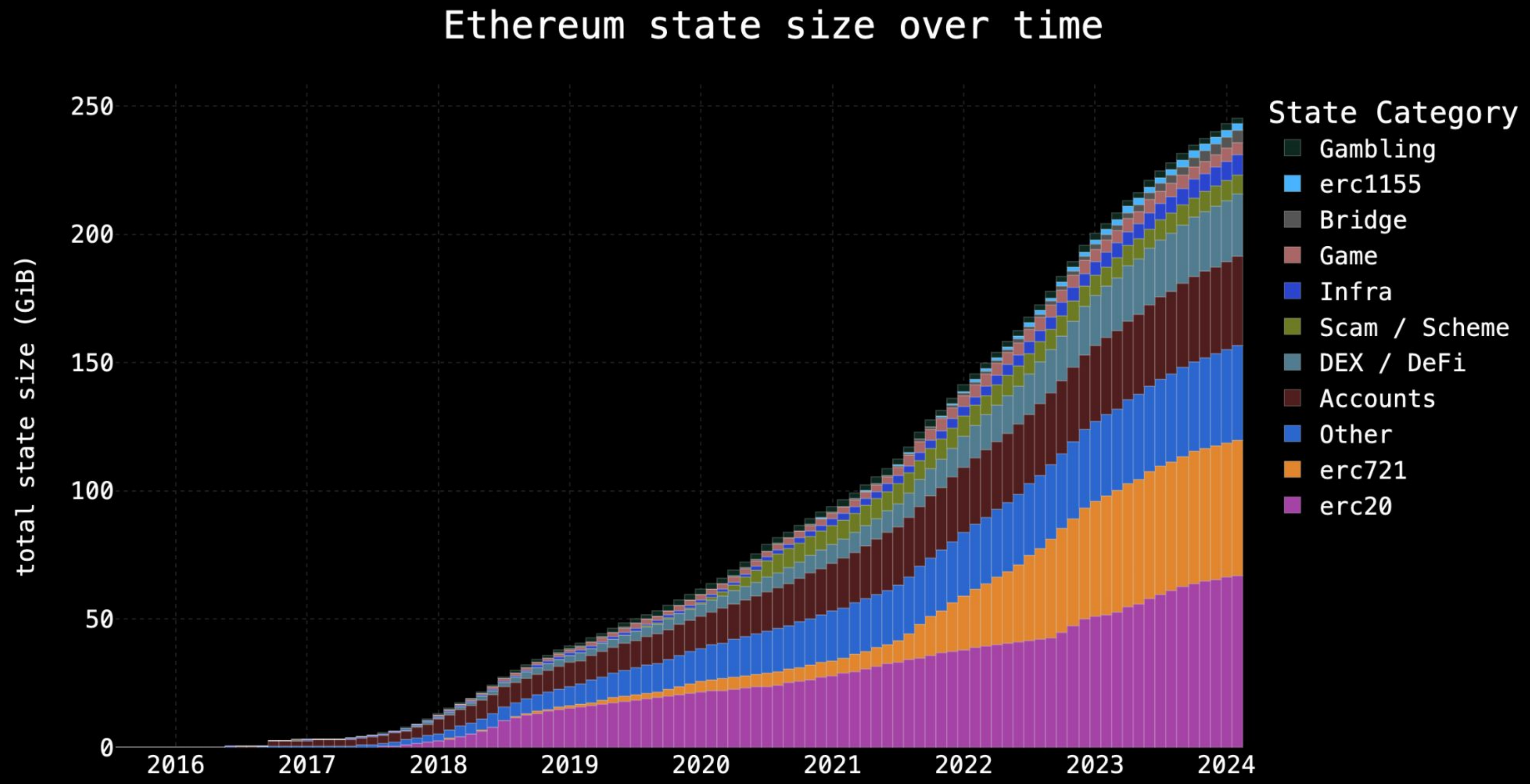
Access list:
Bob's balance
Uniswap balance

Account Model

We still have a state bloat problem




I THREW OUT THE STATE TREE! JUST RUN A FULL NODE TO KNOW THE STATE!
DON'T I NEED A DATA CENTER FOR THAT?



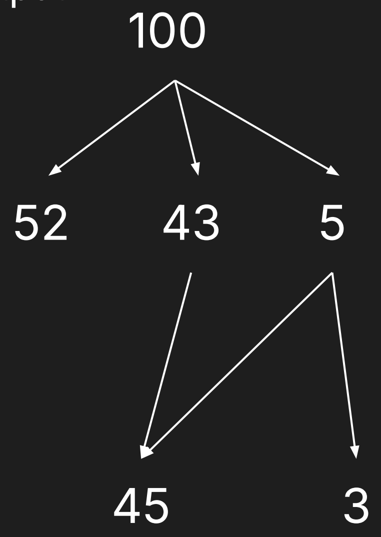
Ethereum state size over time
total state size (GiB)
0
50
100
150
200
250
2016
2017
2018
2019
2020
2021
2022
2023
2024
State Category
Gambling
erc1155
Bridge
Game
Infra
Scam / Scheme
DEX / DeFi
Accounts
Other
erc721
erc20

Can we do something else than Accounts?

UTXO Model

Unspent Transaction Outputs

- Ownership is represented by a set of Unspent Transaction Outputs
- Transactions spend UTXOs and create new ones as output
- No need to store accounts
- Easy to parallelize
- Many inputs, many outputs

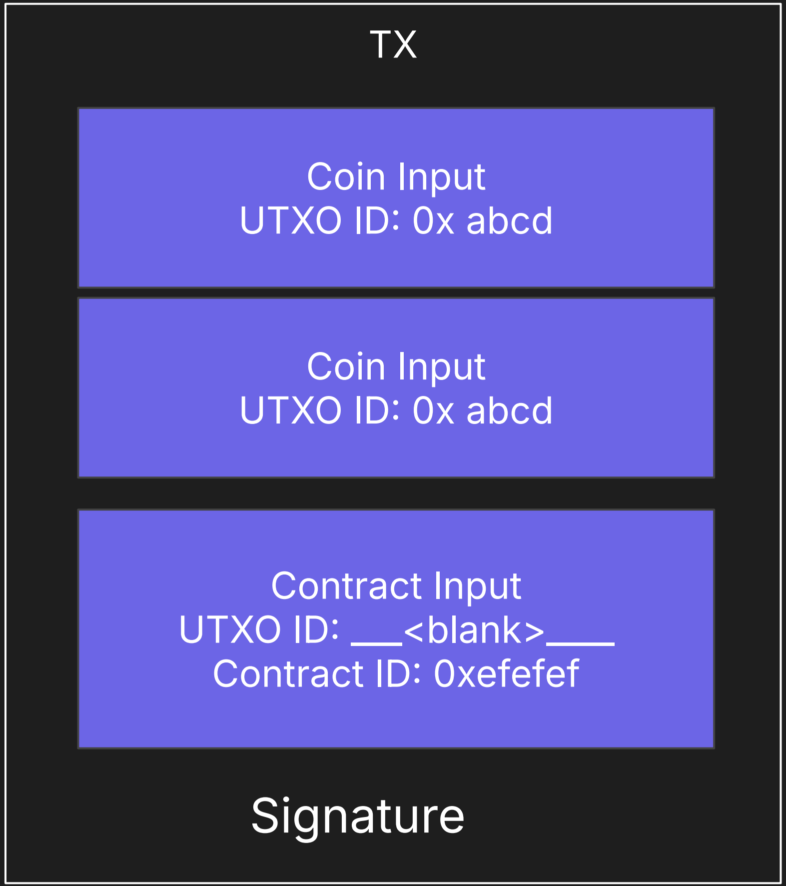


UTXO Model

- Keeping track of all your UTXOs can be complex

- The Cardano Problem
  - UXTOs are deterministic and can only be spent once
  - But then, how do users of a dapp agree who gets to spend which?

UTXO Model

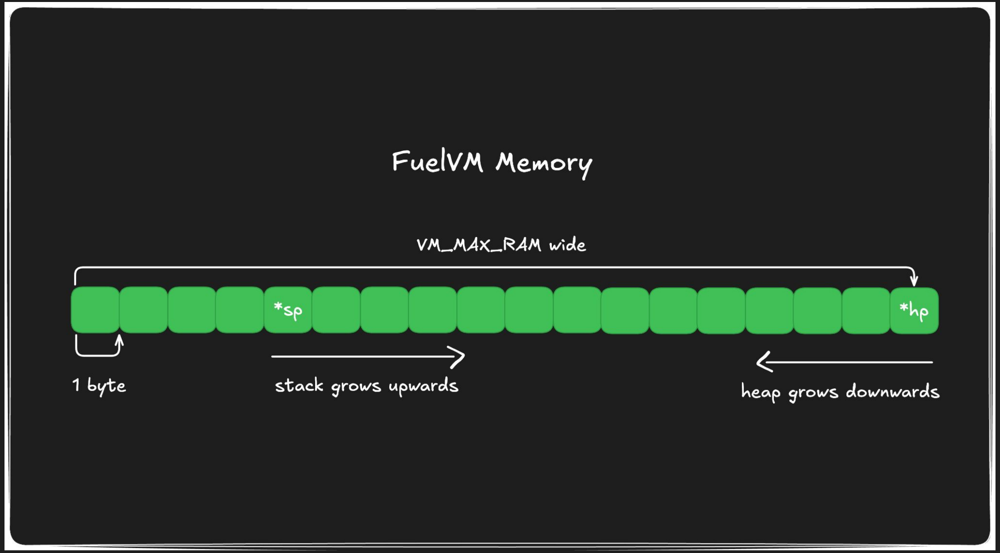


I want to use ANY Contract UTXO that matches this ID

The Virtual Machine

The FuelVM

- 64 bit
- Register based
- Byte indexed Stack and Heap model



The FuelVM

On FuelVM:

- Hashing
- Even more Signature Verification
- All assets are native!

On EVM:

- Hashing
- Signature Verification
- ETH is native, everything else is a contract

Native Assets

- MINT and BURN instructions
- `asset_id = sha256((contract_id, sub_id))`
- SRC-20

- Better DevEx: less boilerplate to make new assets
- Better Performance: less logic for the VM to process
- Better Security: no bugs or inconsistencies in transfer

Executions Contexts

- Scripts
- Predicates
- Contracts

Scripts

- Entry points for transactions (if you’re doing more than just transfers)
- Ephemeral (no state)
- Can call as many contracts as you like

Scripts are essentially “contract launchers”

Scripts

```
script;

fn main() {

    let my_contract = abi(
        MyContract,
        MY_CONTRACT_ID
    );
    my_contract.my_function();

}
```

Scripts

```
script;

fn main() {

    let my_contract = abi(
        MyContract,
        MY_CONTRACT_ID
    );
    my_contract.first_function();
    my_contract.second_function();

}
```

Scripts

Replaces
Uniswap-style router
contract

```
script;

fn main() {

    let amm = abi(MyAmm, MY_CONTRACT_ID);
    amm.read_from_pools();
    do_some_calculations();
    amm.swap(A, B);
    amm.swap(B, C);
    ensure_final_output_is_sufficient();

}
```

Predicates

- Can own coins
- not stored on chain
- stateless spending condition

Predicates

Coin held by a normal account

1 ETH

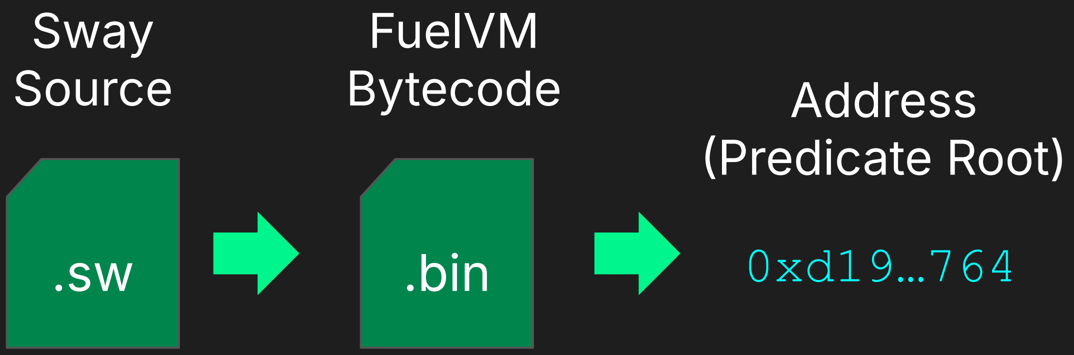
Can be spent with a signature

Coin held by a predicate root

1 ETH

Can be spent by executing a predicate matching the root and returning true

Predicates



All funds sent to a predicate root can be spent if one can execute the predicate and return true

Predicates

```
predicate;

fn main() -> bool {
    Time::now() > UNLOCK_TIMESTAMP
}
```

Anybody can spend the funds after this date

Predicates

```
predicate;

fn main(sig: b256) -> bool {
    validate_signature(sig)
}
```

Contracts

Smart contracts with their own:

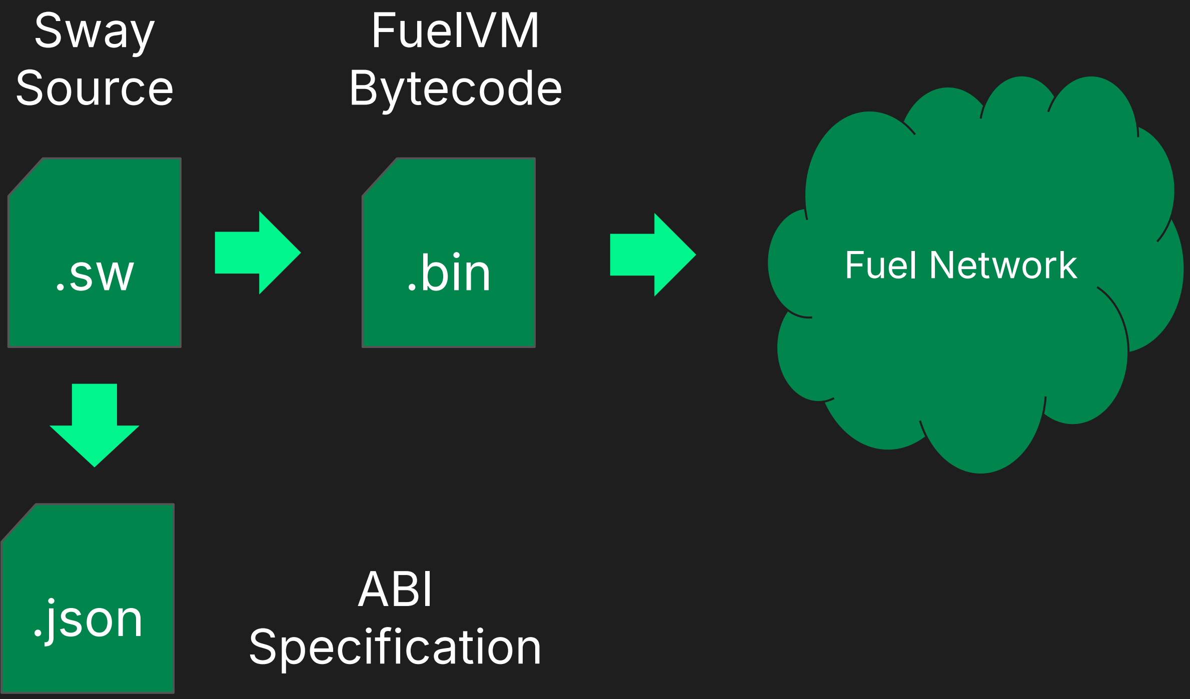
- Coins (contracts can own assets)
- ABI
- Execution context
- Storage context

Contract ABI

```
contract;

impl Contract {
    fn foobar() {
        log(“hello”);
    }
}
```

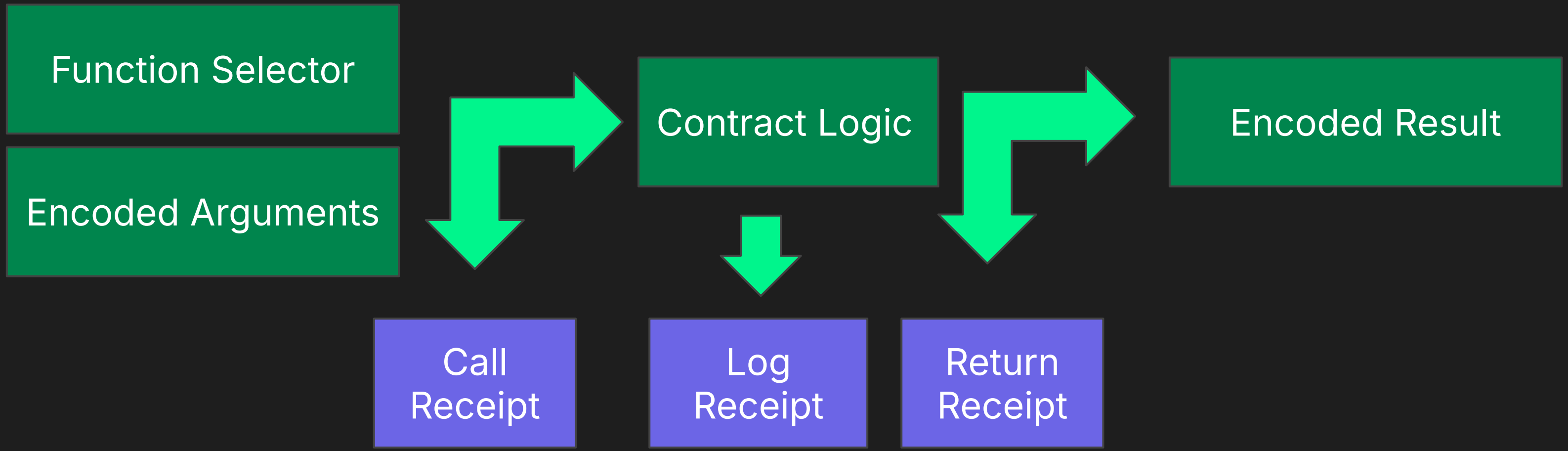
Contract ABI



Contract ABI

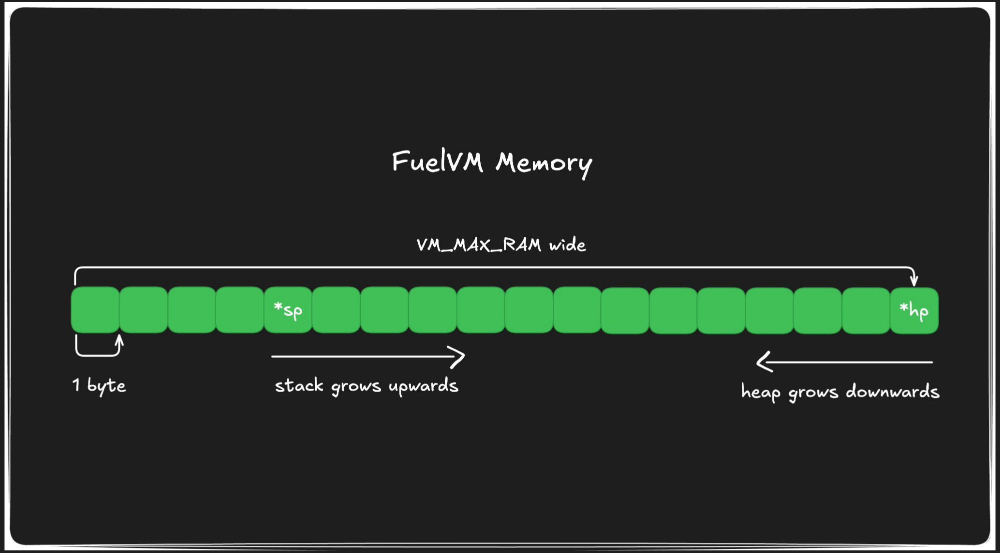
```
…
"functions": [
   {
      "name": "foobar",
      "inputs": [],
      "output": "2e3…f5d",
      "attributes": null
    }
  ],
…
```

Contract ABI

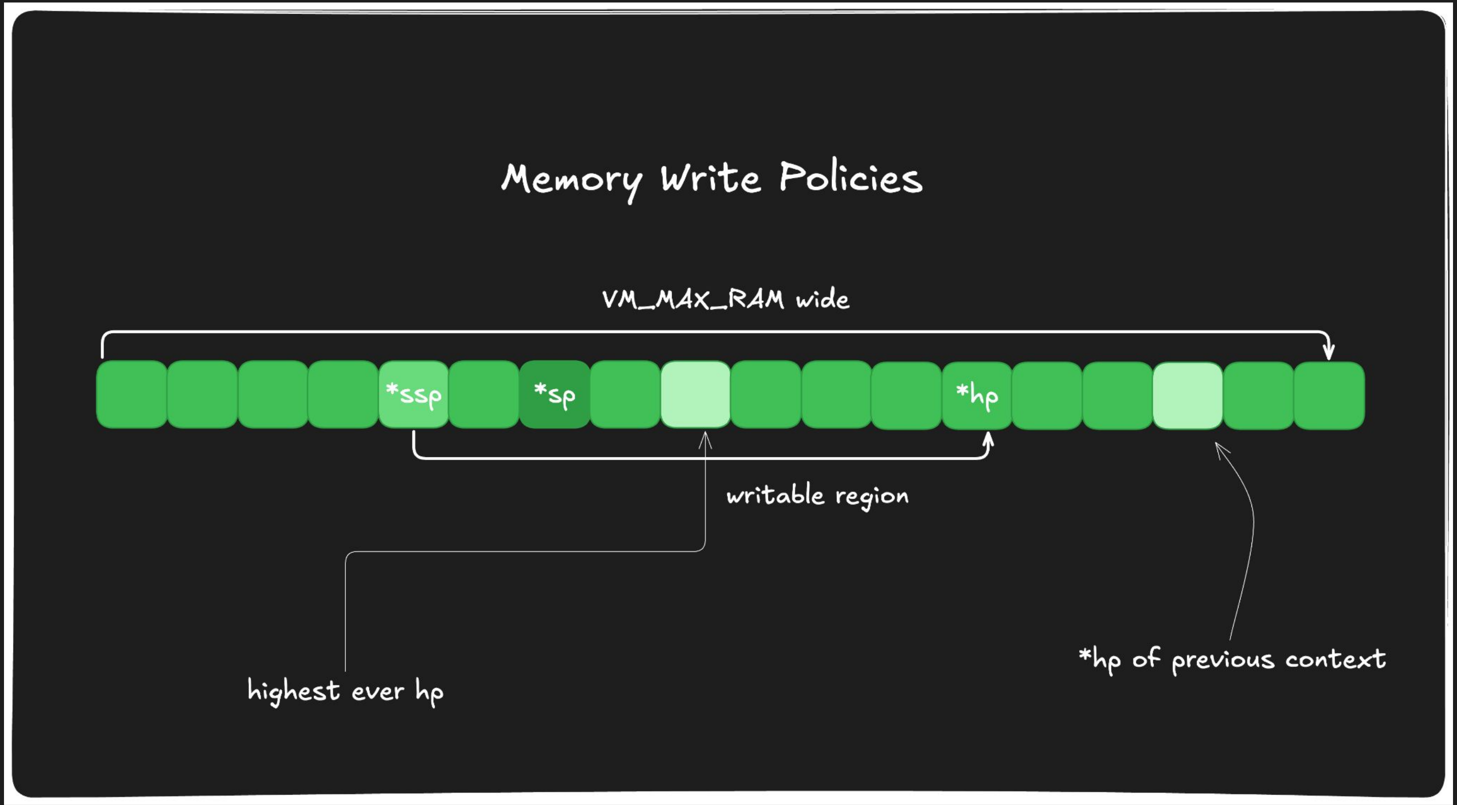


Memory

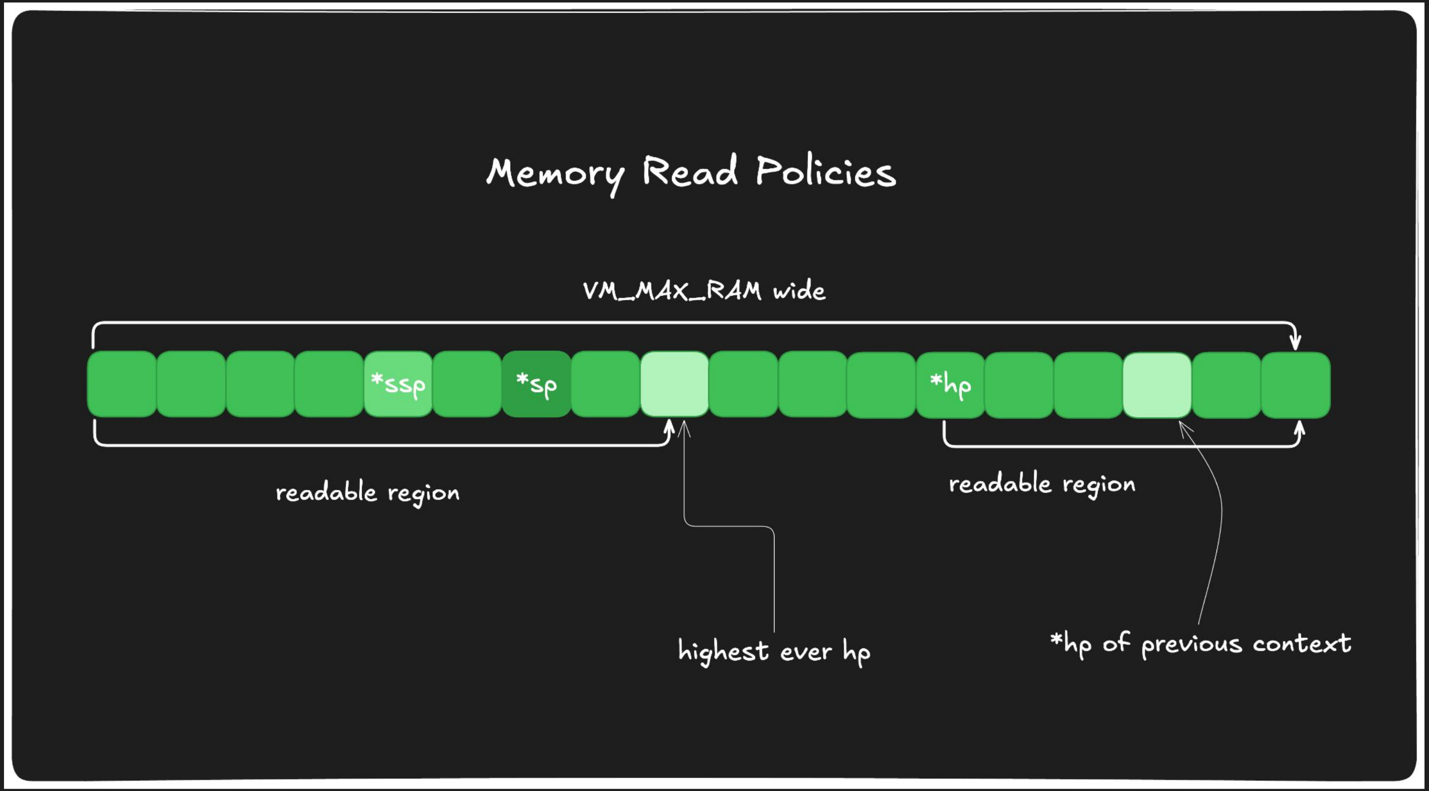
- ssp: bottom of the current writable stack area
- sp: on top of the stack
- hp: below the current bottom of the heap



Memory


Memory Write Policies
VM_MAX_RAM wide
*ssp
*sp
*hp
writable region
highest ever hp
*hp of previous context

Memory



Memory Read Policies
VM_MAX_RAM wide
*ssp
*sp
*hp
readable region
readable region
highest ever hp
*hp of previous context

Memory

...
Unwritable Area

Contract ID for this call

Asset ID of forwarded coins

Saved registers from previous contexts

Code Size

Function Selector

Encoded Arguments Pointer

Padding

Call Frame Stack
...

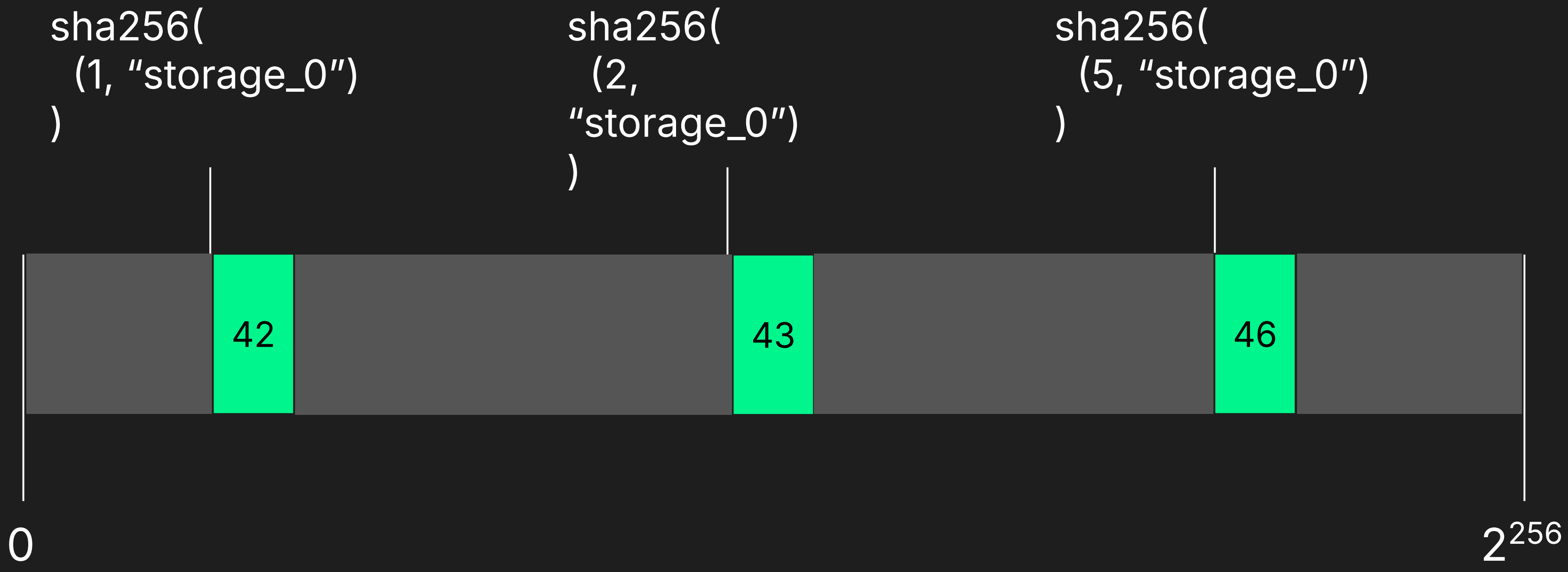
A typical call frame

Storage

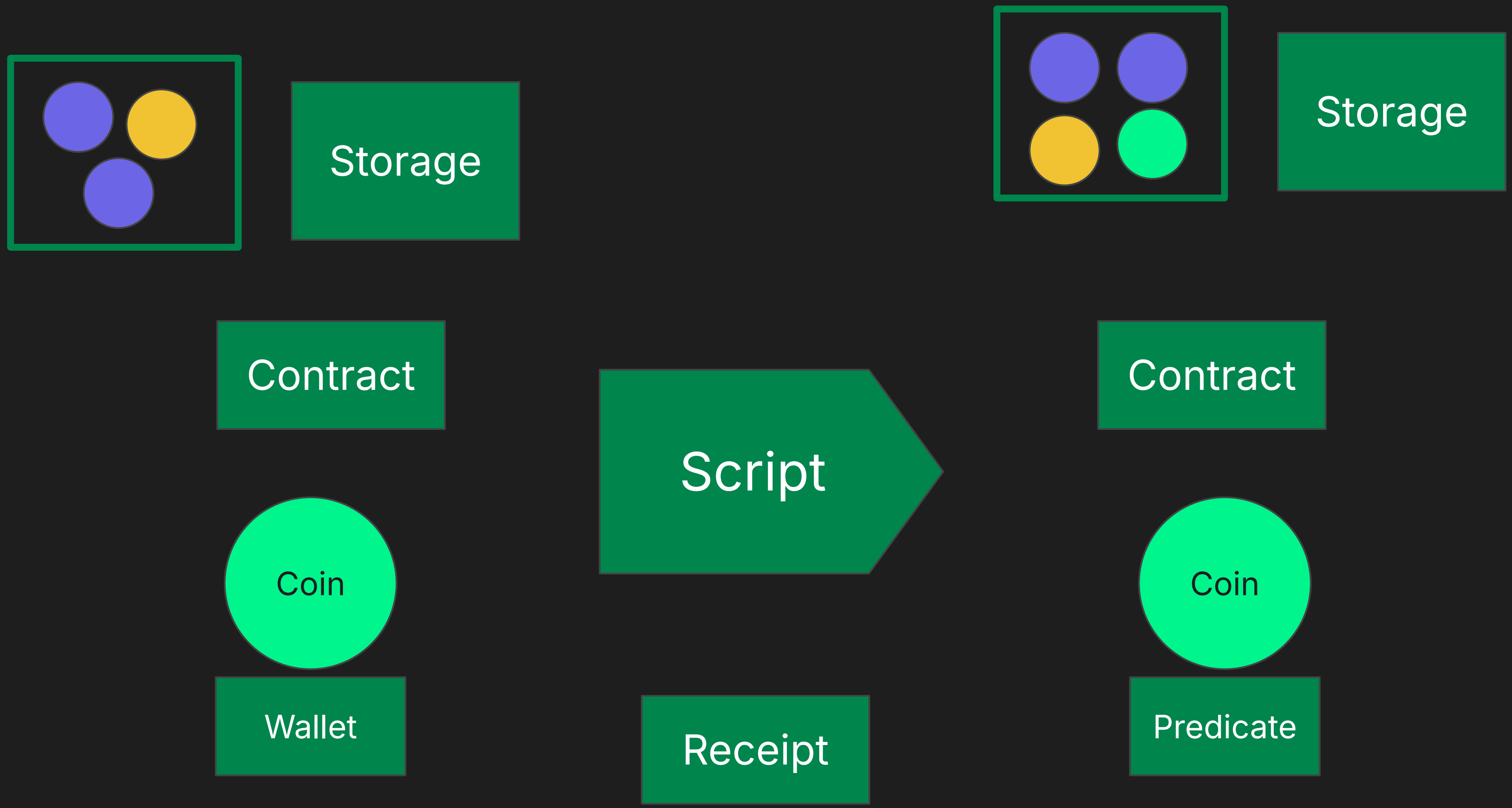
- Persistent storage
- Every contract has its own independent storage context
- 2^{256} slots of 256 bits (sparse)

```
storage {
    foo: StorageMap<u64, u64> = StorageMap{},
}
```

1	2	5
42	43	46



Putting it all together


Storage
Contract
Coin
Wallet
Script
Receipt
Storage
Contract
Coin
Predicate