Native Account Abstraction in Pectra, rollups and beyond

Combining EOF, EIP-7702 and RIP-7560 to achieve the AA endgame

Alex Forshtat

Account Abstraction Team

Okay, let's say we want Full Native Account Abstraction. How do we get it?

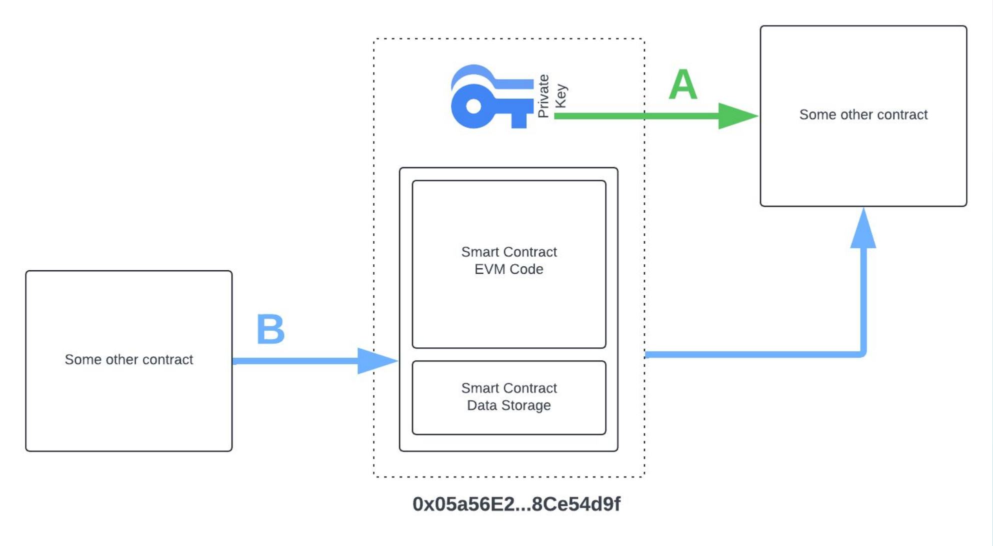
What part of AA is coming to Ethereum in the next hard-fork?
What is still needed for the “endgame”?
How do we want to achieve it?
How can you participate?
Are there alternatives?

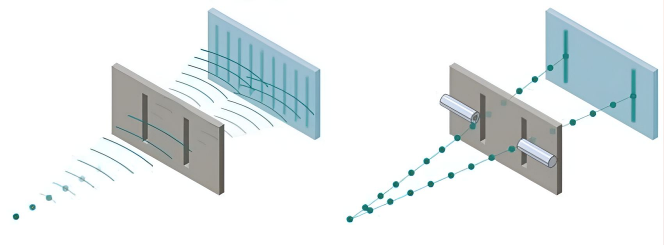
TLDR: Status of Account Abstraction standards

- ERC-4337:
  - Solved most AA use-cases
  - No longer “new” - used by serious projects
- EIP-7702:
  - Allows EOAs to role-play as Smart Accounts
  - Scheduled for inclusion with Pectra
- RIP-7560:
  - Enshrines ERC-4337 design - becomes native part of L2 rollups
  - Ready for devnet
- EIP-7701:
  - Similar to RIP-7560 but less opinionated
  - Targets Ethereum L1 with EOF
  - Feedback requested

EIP-7702 authorizations and Account Abstraction

- Changes the behaviour of existing EOAs - allows them to have code
- Fully solves the “execution” part of Account Abstraction
- Does not solve the “security” part of Account Abstraction
    - ECDSA key can override the contract code
    - EOA needed to create Type-4 transactions
- Works well with ERC-4337





EIP-7702 addresses have dual nature and can appear as EOAs or as Smart Contracts in different transactions.

Can we wait for them to add full AA to Ethereum?

List of EIPs in either “Scheduled for Inclusion” or “Considered for Inclusion” status for the next hardforks (Prague, Osaka, Amsterdam)

- EIP-2537: Precompile for BLS12-381 curve operations
- EIP-2935: Save historical block hashes in state
- EIP-6110: Supply validator deposits on chain
- EIP-7002: Execution layer triggerable exits
- EIP-7251: Increase the MAX_EFFECTIVE_BALANCE
- EIP-7549: Move committee index outside Attestation
- EIP-7685: General purpose execution layer requests
- EIP-7702: Set EOA account code
- EIP-7623: Increase calldata cost
- EIP-7742: Uncouple blob count between CL and EL
- EIP-7762: Increase MIN_BASE_FEE_PER_BLOB_GAS
- EIP-7594: PeerDAS - Peer Data Availability Sampling
- EIP-7692: EOF EIPs, namely:
- EIP-663: SWAPN, DUPN and EXCHANGE instructions
- EIP-3540: EOF - EVM Object Format v1
- EIP-3670: EOF - Code Validation
- EIP-4200: EOF - Static relative jumps
- EIP-4750: EOF - Functions
- EIP-5450: EOF - Stack Validation
- EIP-6206: EOF - JUMPF and non-returning functions
- EIP-7069: Revamped CALL instructions
- EIP-7480: EOF - Data section access instructions
- EIP-7620: EOF Contract Creation
- EIP-7698: EOF - Creation transaction
- RIP-7212: Precompile for secp256r1 Curve Support
- EIP-4762: Statelessness gas cost changes
- EIP-6800: Ethereum state using a unified verkle tree
- EIP-6873: Preimage retention
- EIP-7545: Verkle proof verification precompile
- EIP-7667: Raise gas costs of hash functions

Can we wait for them to add full AA to Ethereum?

Full Native Account Abstraction will hit L1:

- after it has been tested in production
- after this roadmap is complete
- after there's consensus among core devs

Such things take years.

This does not mean we have to sit and wait!

There are many ways for the community to advance with full AA in the meantime.

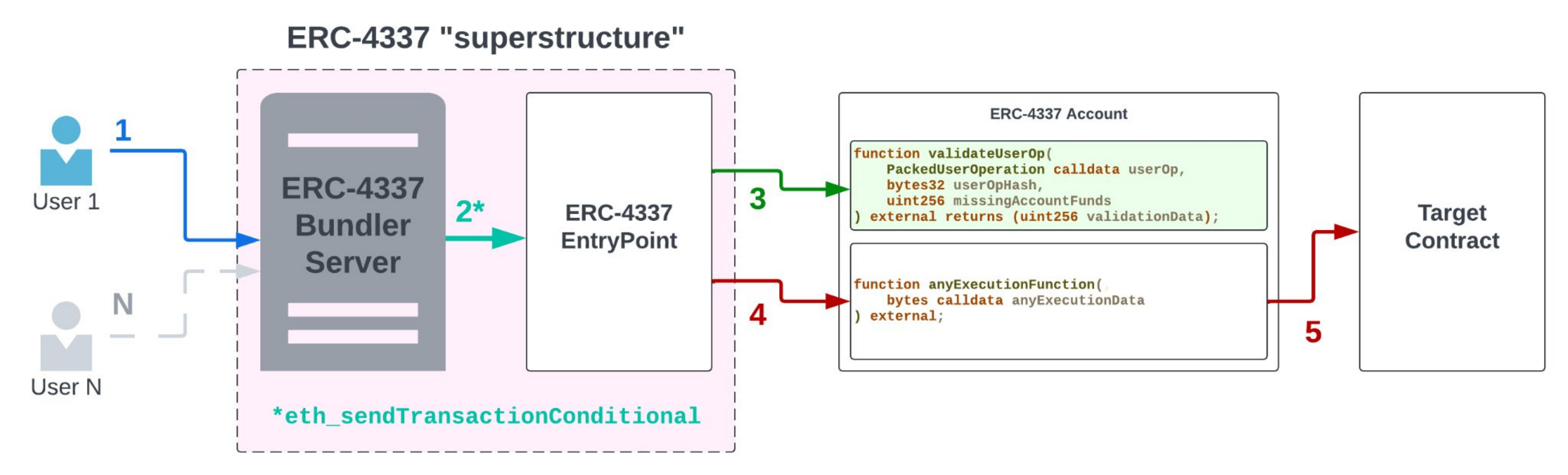
Can we keep using ERC-4337 forever?

Yes, ERC-4337 is often “good enough”.

It is not perfect:

- Relies on EOAs to act as “bundlers”
- Adds cost and complexity overhead
- Forms a parallel tech stack
- Incompatible full AA solutions are created on L2s causing fragmentation
- Will not support future features (see “inclusion lists” EIP-7547 or EIP-7805)

A simple ERC-4337 transaction flow

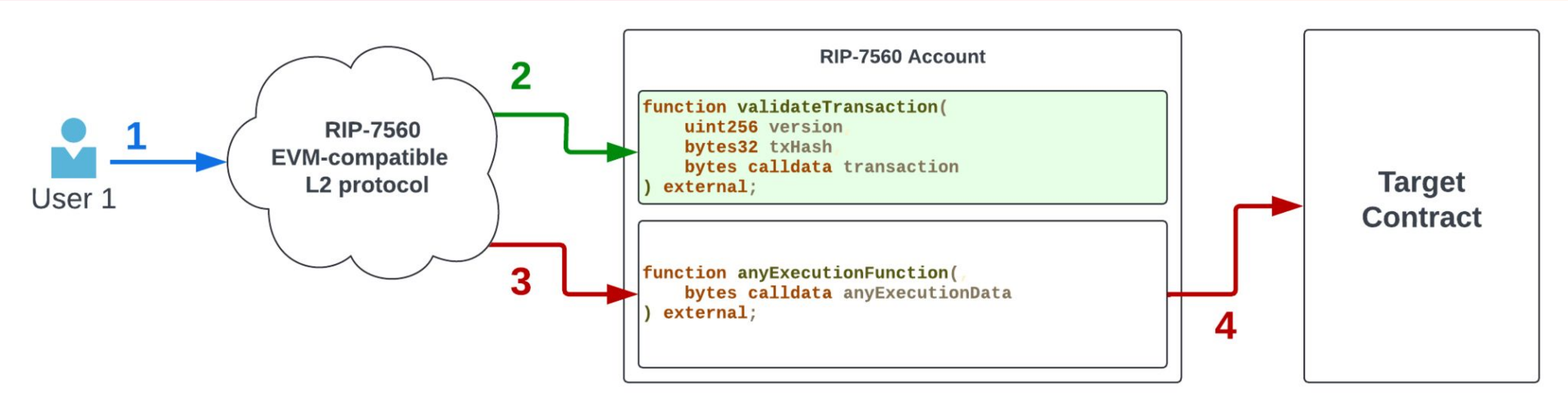


```
function validateUserOp(
    PackedUserOperation calldata userOp,
    bytes32 userOpHash,
    uint256 missingAccountFunds
) external returns (uint256 validationData);
```

```
function anyExecutionFunction(
    bytes calldata anyExecutionData
) external;
```

ERC-4337 bundler servers participate in a p2p UserOps mempool. Bundlers must inject `'EntryPoint::handleOps'` tx directly into the block builder using an ERC-7796 conditional API.

RIP-7560: Full Native AA for Layer-2s



```
function validateTransaction(
    uint256 version,
    bytes32 txHash,
    bytes calldata transaction
) external;
```

```
function anyExecutionFunction(,
    bytes calldata anyExecutionData
) external;
```

With RIP-7560 the Account Abstraction stack becomes a part of the chain protocol.

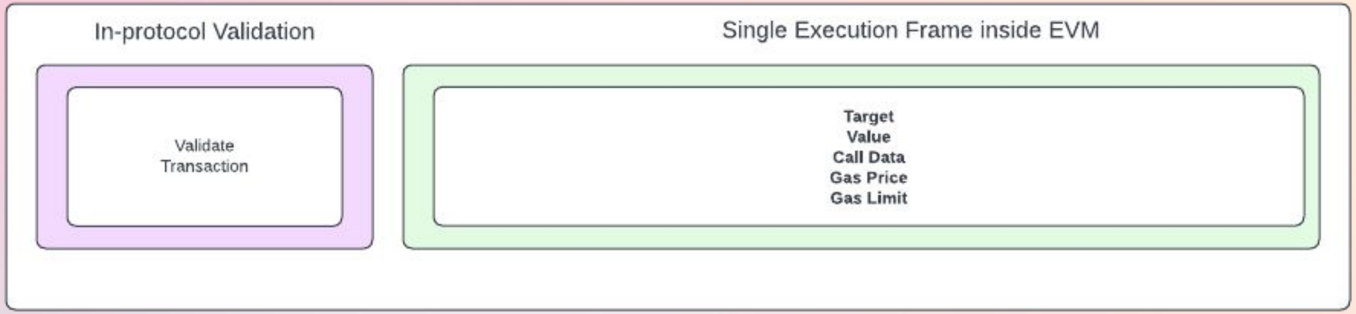
RIP-7560: Splitting up a transaction into multiple frames

Protocol includes validation of:

- Signature
- Nonce
- Balance
- Gas limits
- Gas fee payment

This is still a single Ethereum transaction

Externally Owned Account Initiated Transaction



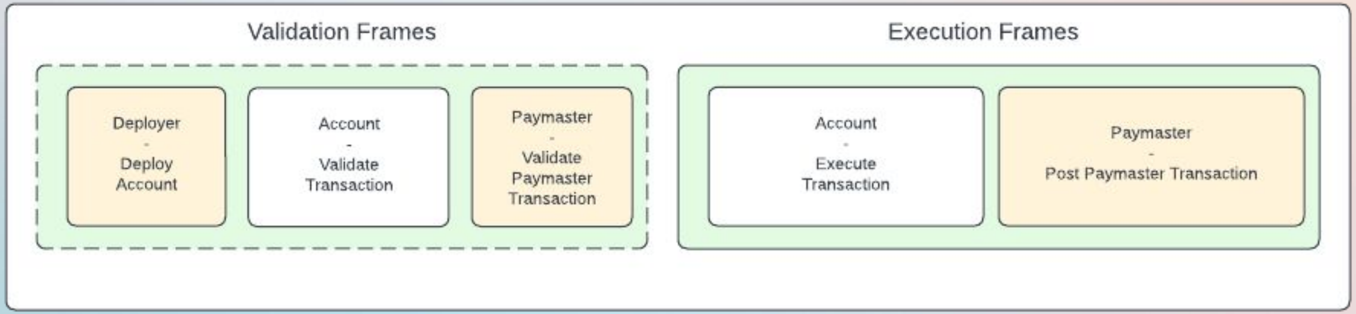
In-protocol Validation

Validate Transaction

Single Execution Frame inside EVM

Target
Value
Call Data
Gas Price
Gas Limit

Native Account Abstraction Transaction



Validation Frames

Deployer - Deploy Account

Account - Validate Transaction

Paymaster - Validate Paymaster Transaction

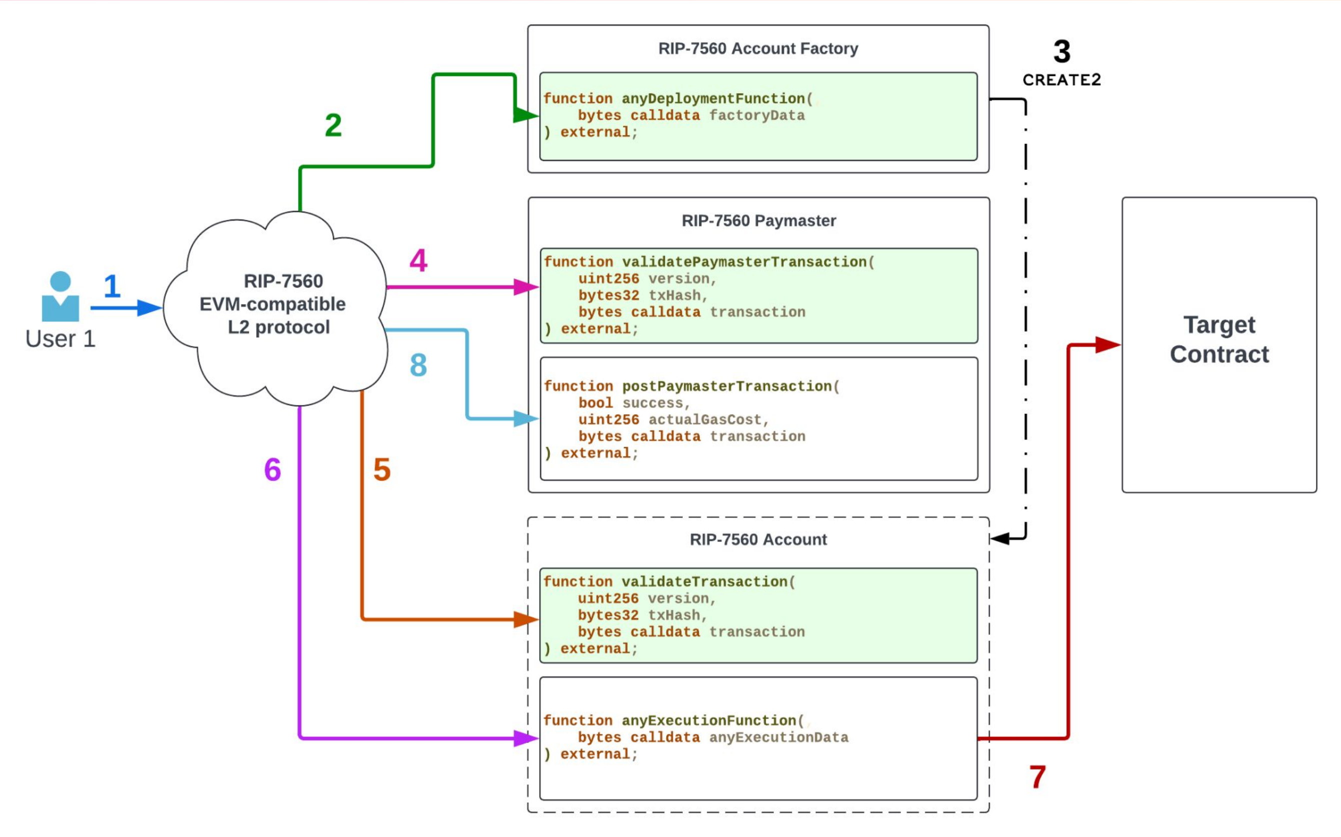
Execution Frames

Account - Execute Transaction

Paymaster - Post Paymaster Transaction

RIP-7560: All possible code paths

chainId, nonce,
executionData,
sender, **builderFee**,
paymaster,
paymasterData,
deployer,
deployerData,
maxPriorityFeePerGas,
maxFeePerGas,
validationGasLimit,
paymasterGasLimit,
postOpGasLimit,
callGasLimit,
accessList, authList
authorizationData



```
function anyDeploymentFunction(
    bytes calldata factoryData
) external;
```

```
function validatePaymasterTransaction(
    uint256 version,
    bytes32 txHash,
    bytes calldata transaction
) external;
```

```
function postPaymasterTransaction(
    bool success,
    uint256 actualGasCost,
    bytes calldata transaction
) external;
```

```
function validateTransaction(
    uint256 version,
    bytes32 txHash,
    bytes calldata transaction
) external;
```

```
function anyExecutionFunction(
    bytes calldata anyExecutionData
) external;
```

RIP-7560 is not meant for Ethereum Mainnet

- The RIP process coordinates common new features between various L2s
- RIPs are opt-in - unanimity not needed
- RIPs can take some liberties with the protocols
- RIPs may evolve into EIPs

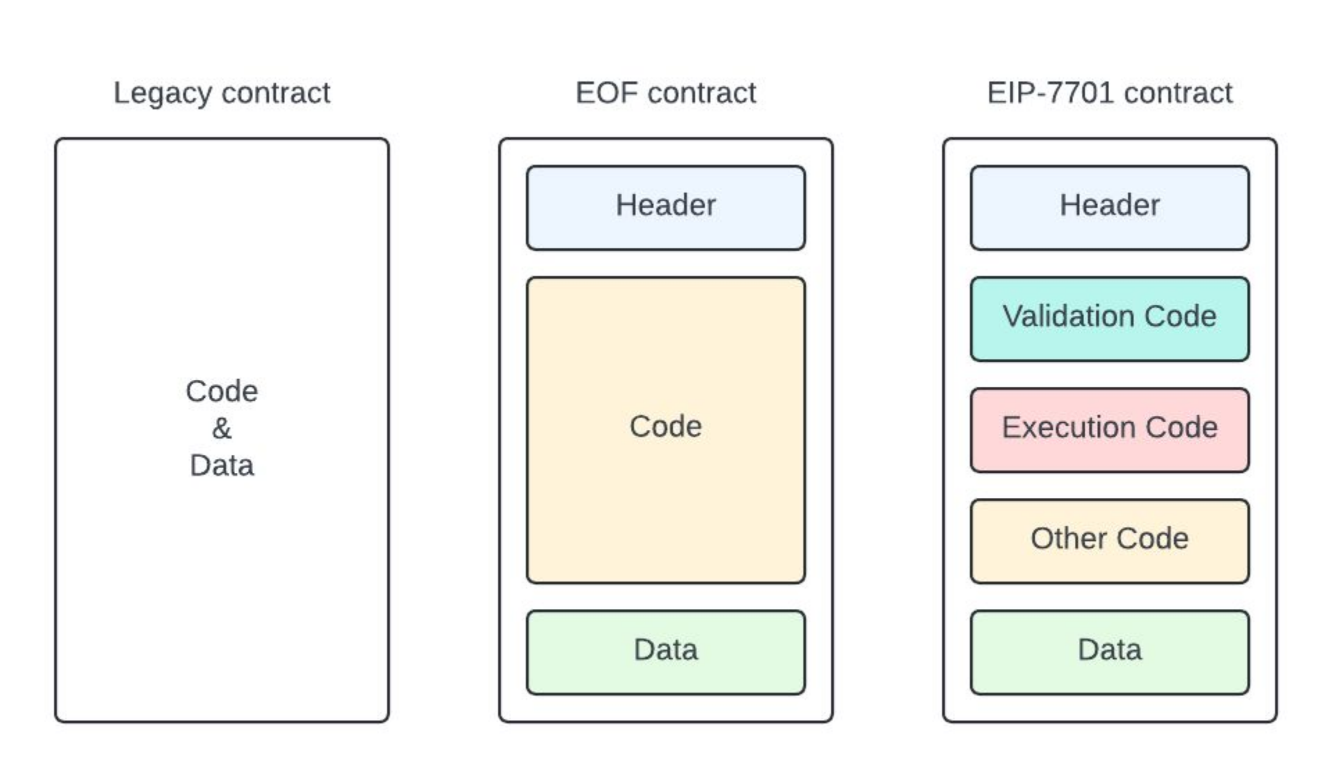
What prevents RIP-7560 from being a mainnet EIP?

- Depends on Solidity method selectors for contract roles
  - EVM must remain language-agnostic
    - Most L2s introduce Solidity-like precompiles anyways
  - Threat of method selector collisions
  - Easy to hide Indication of “successful validation” in code
- Validation code not distinguishable during deployment
  - EOF verifies contract code during deployment
  - We can use AA-specific checks for validation code
- Validation code is exposed as an external function
  - This is unnecessary and can be abused by some implementations

EIP-7701: EOF containers and Account Abstraction

Using EOF allows us to:

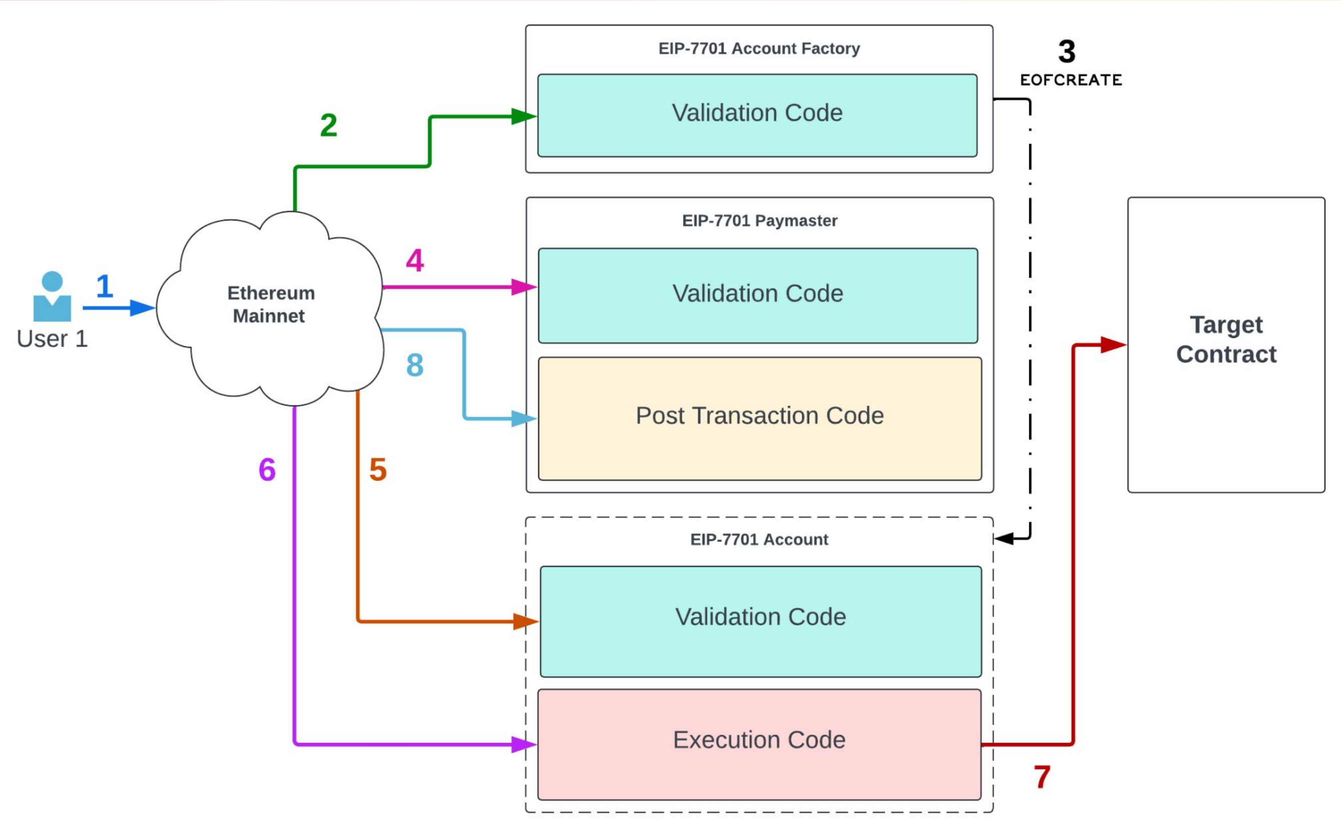
- Create code sections with roles determined by the protocol
- Verify the code in these containers with different rules
- Hide code in these sections from ABI
- Non-reverting code as indication of “successful validation”



EIP-7701: All possible code paths

Transaction flow is exactly the same.

The only changes are related to transition from “magic” method selectors to EOF-based dispatch.



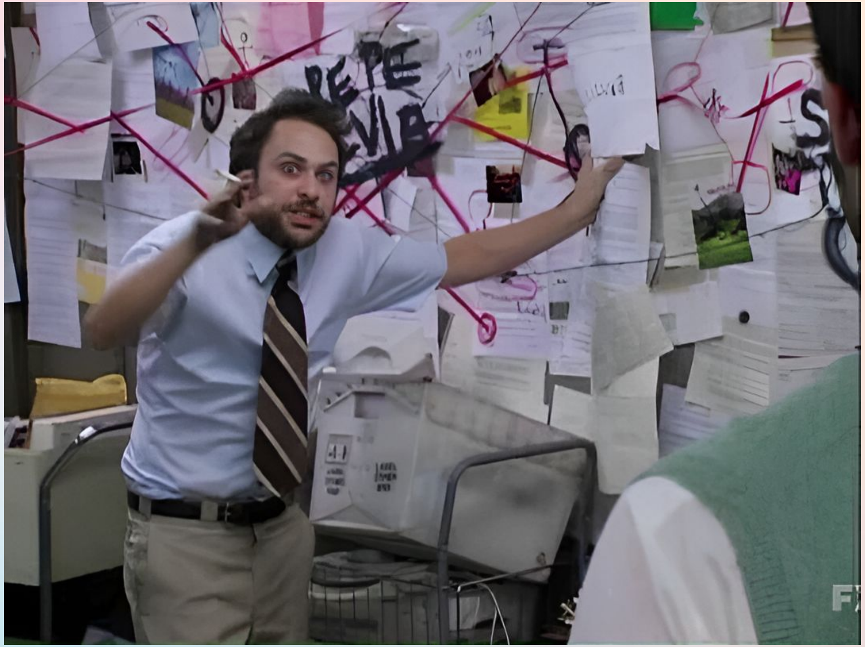
Native Account Abstraction - Big Challenges

AA would have been on mainnet years ago if it was that simple. It is not.

AA allows for cross-transaction dependencies in mempool and in blocks.

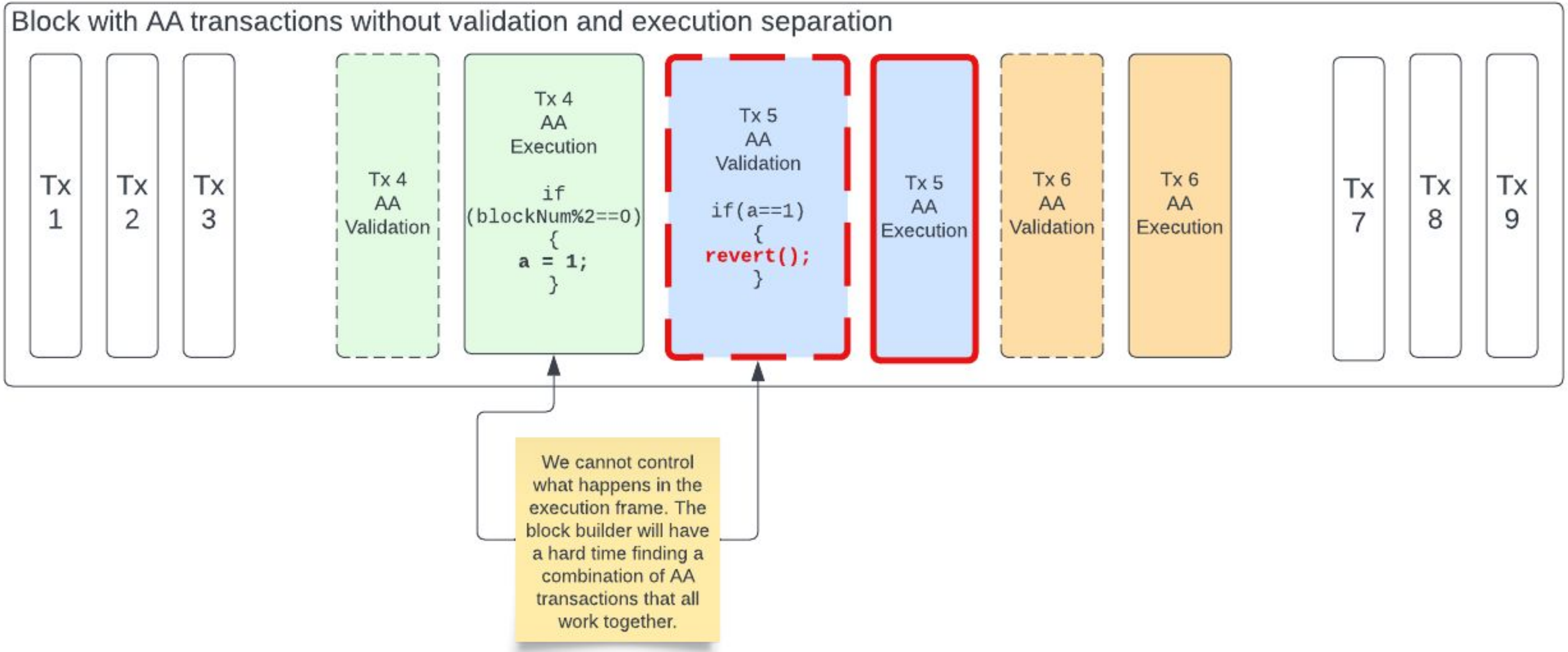
The main challenges it poses:

- Efficient block building
- DoS-resilient p2p mempool



ERC-7562: Account Abstraction Validation Scope Rules

Cross-transaction dependencies and block building



Block with AA transactions without validation and execution separation

Tx 1

Tx 2

Tx 3

Tx 4 AA Validation

Tx 4 AA Execution

```
if
(blockNum%2==0)
{
a = 1;
}
```

Tx 5 AA Validation

```
if(a==1)
{
revert();
}
```

Tx 5 AA Execution

Tx 6 AA Validation

Tx 6 AA Execution

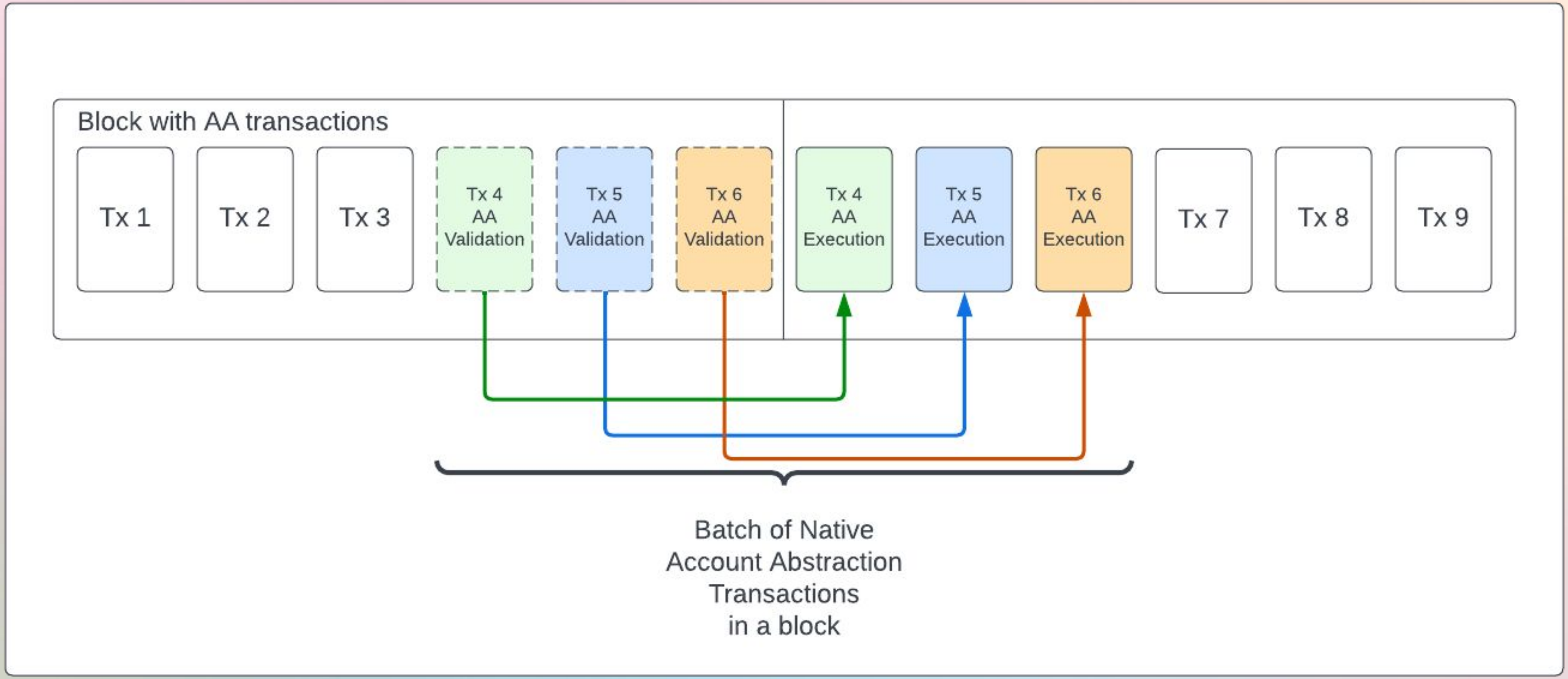
Tx 7

Tx 8

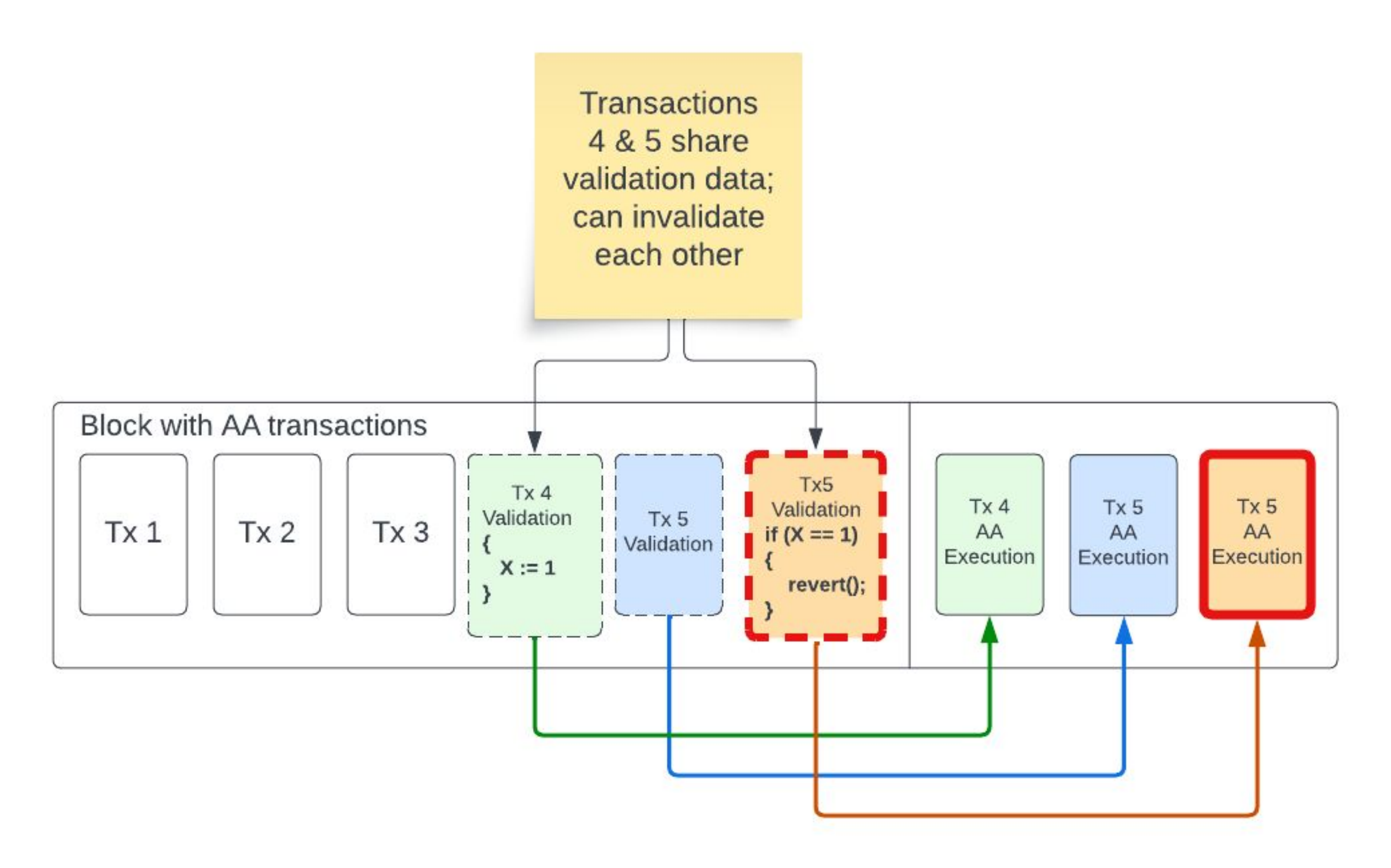
Tx 9

We cannot control what happens in the execution frame. The block builder will have a hard time finding a combination of AA transactions that all work together.

RIP-7711: Validation-Execution separation



Cross-validation dependencies and validation rules



Transactions 4 & 5 share validation data; can invalidate each other

Block with AA transactions

Tx 1

Tx 2

Tx 3

Tx 4 Validation

```
{
   X := 1
}
```

Tx 5 Validation

Tx5 Validation

```
if (X == 1)
{
   revert();
}
```

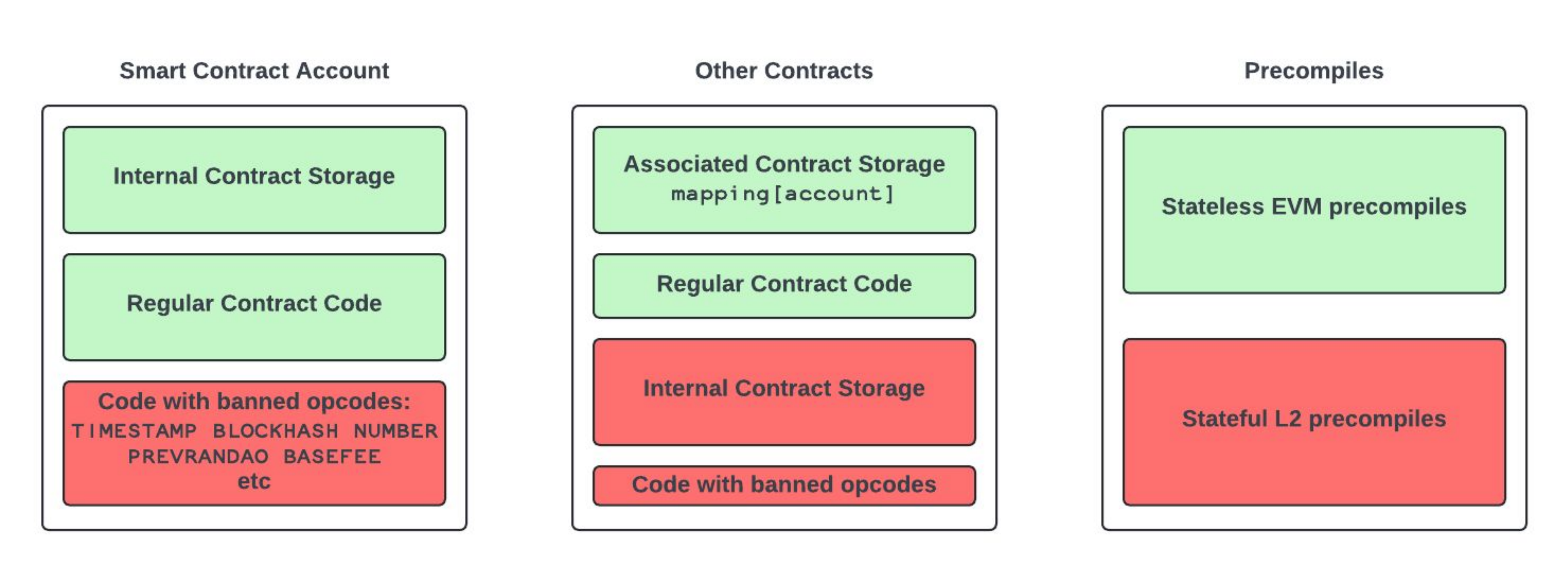
Tx 4 AA Execution

Tx 5 AA Execution

Tx 5 AA Execution

Cross-validation dependencies and validation rules

Validation rules allow the block builders to immediately detect and reject transactions that may have validation cross-dependencies and pose a DoS threat.



Smart Contract Account

Internal Contract Storage

Regular Contract Code

Code with banned opcodes:
`TIMESTAMP BLOCKHASH NUMBER PREVRANDAO BASEFEE`
etc

Other Contracts

Associated Contract Storage
`mapping[account]`

Regular Contract Code

Internal Contract Storage

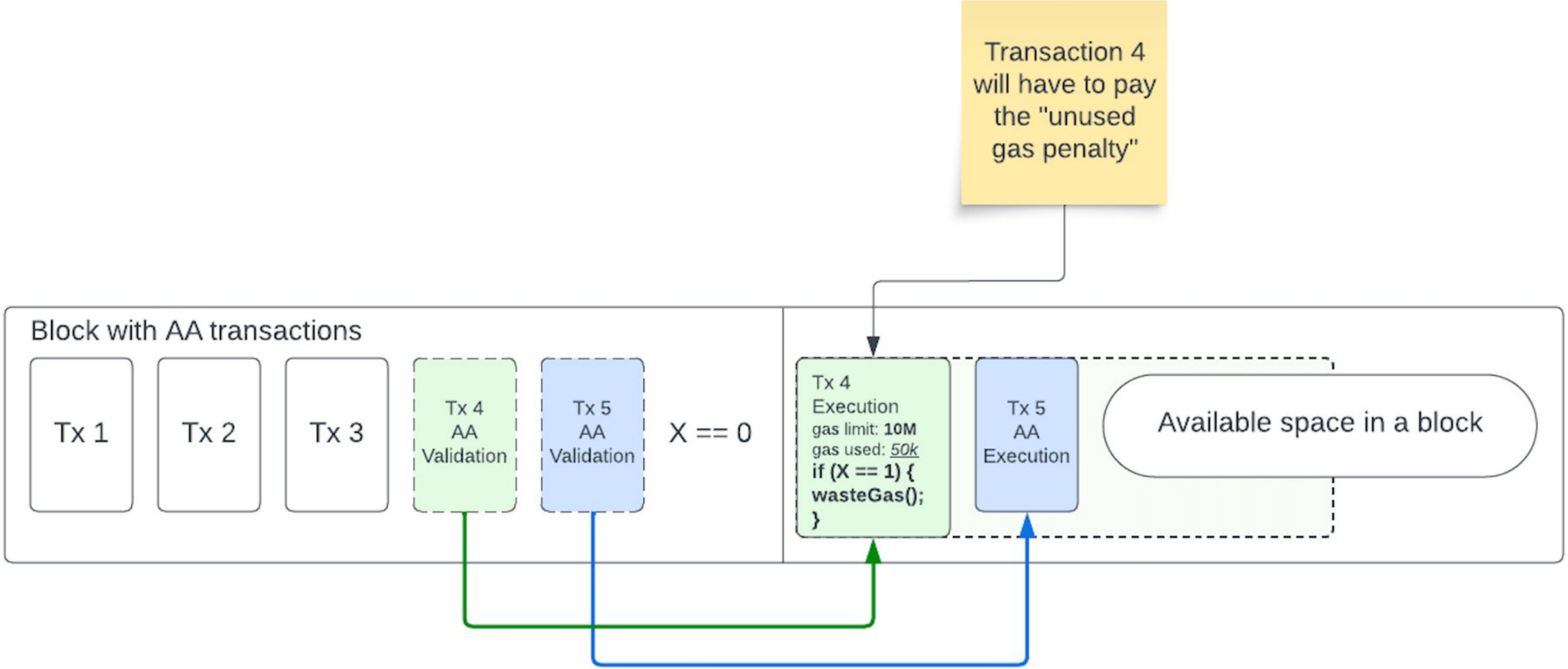
Code with banned opcodes

Precompiles

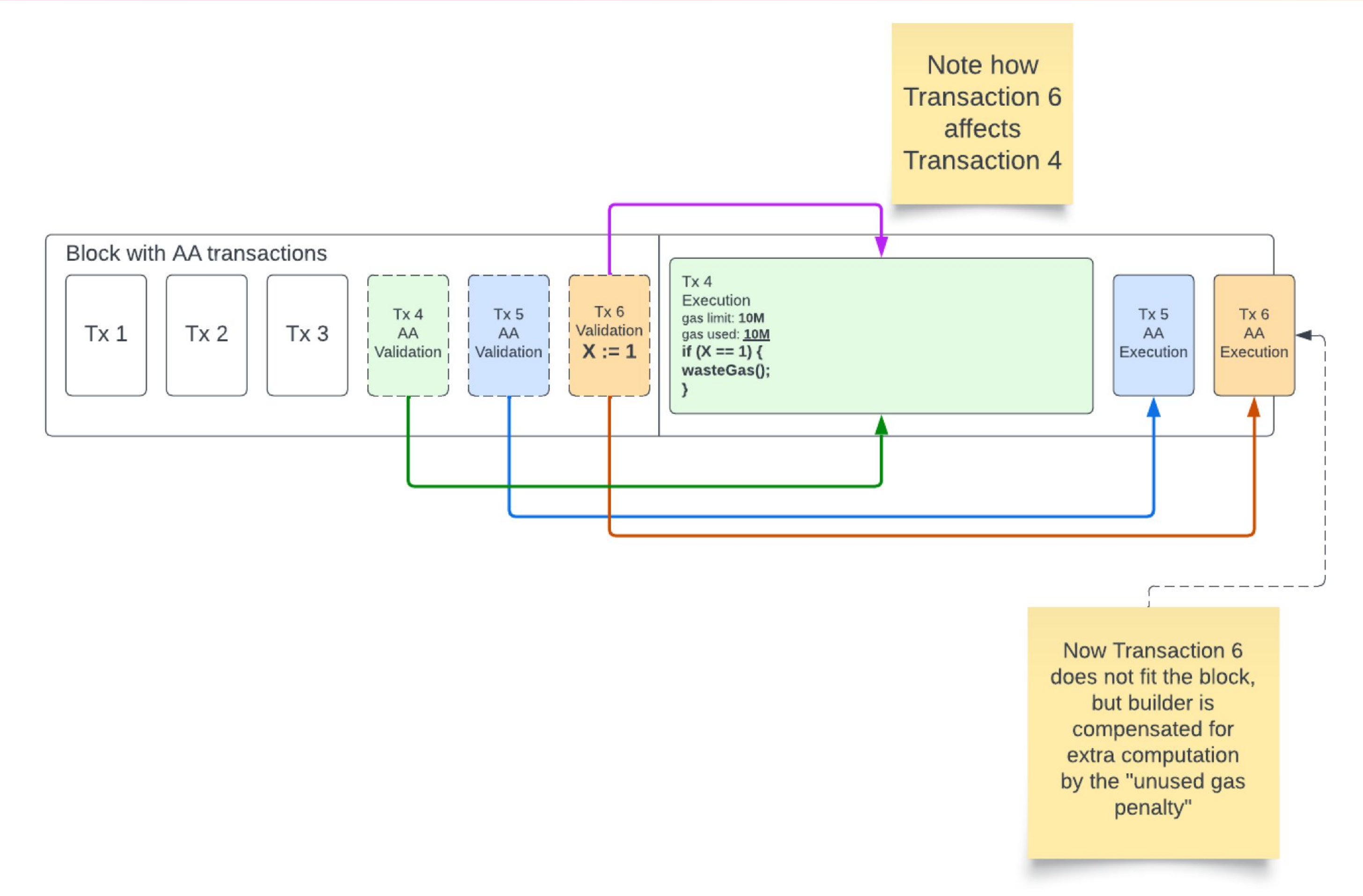
Stateless EVM precompiles

Stateful L2 precompiles

Example of a side channel attack: unused gas as a vector



Example of a side channel attack: unused gas as a vector

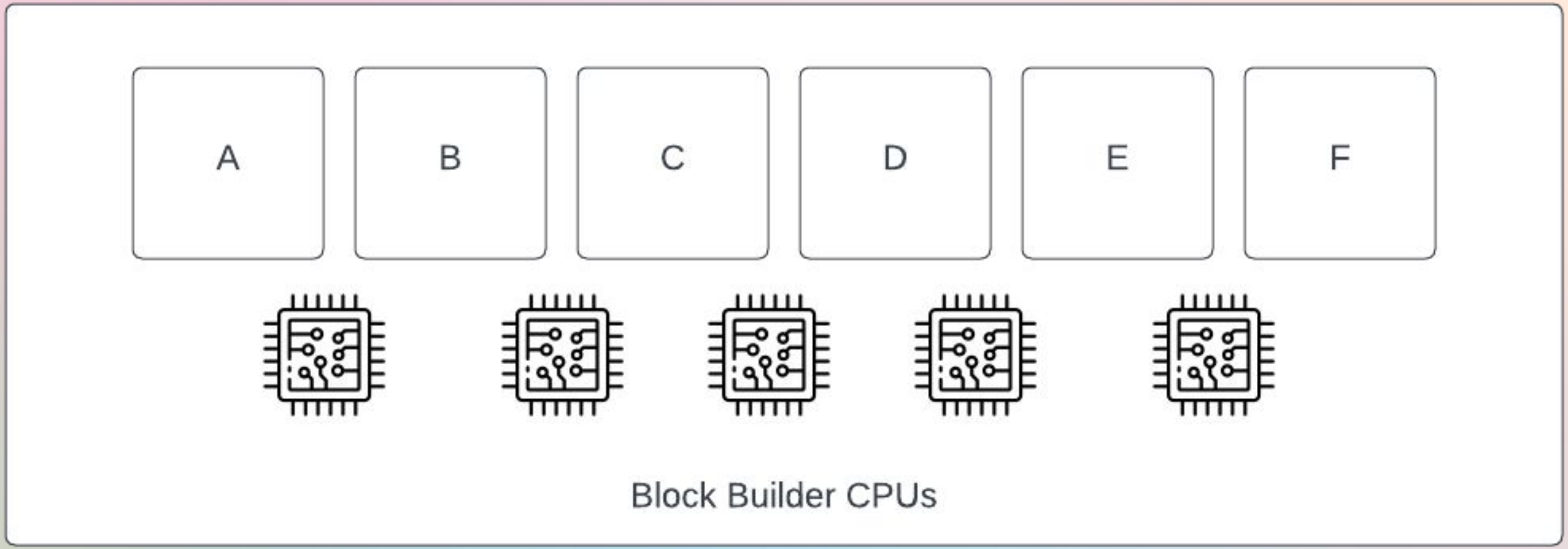


Note how Transaction 6 affects Transaction 4

Now Transaction 6 does not fit the block, but builder is compensated for extra computation by the "unused gas penalty"

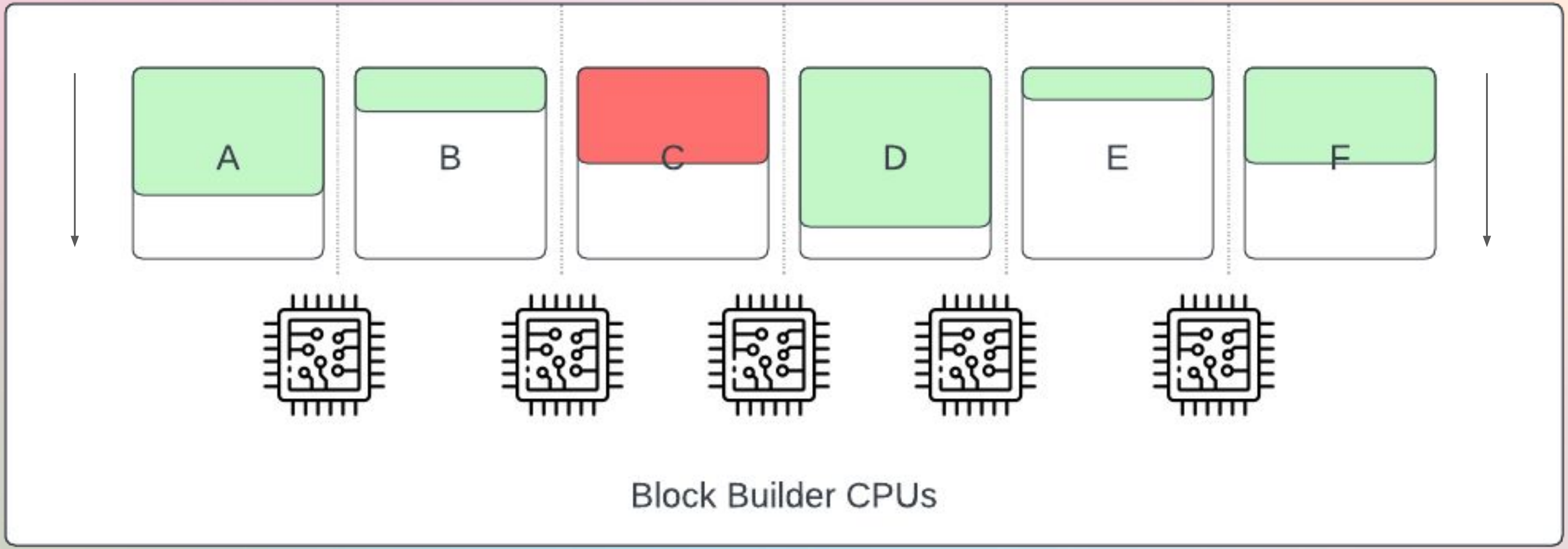
Secure, efficient and parallelized transaction validation

Block builders must be able to validate each transaction independently.



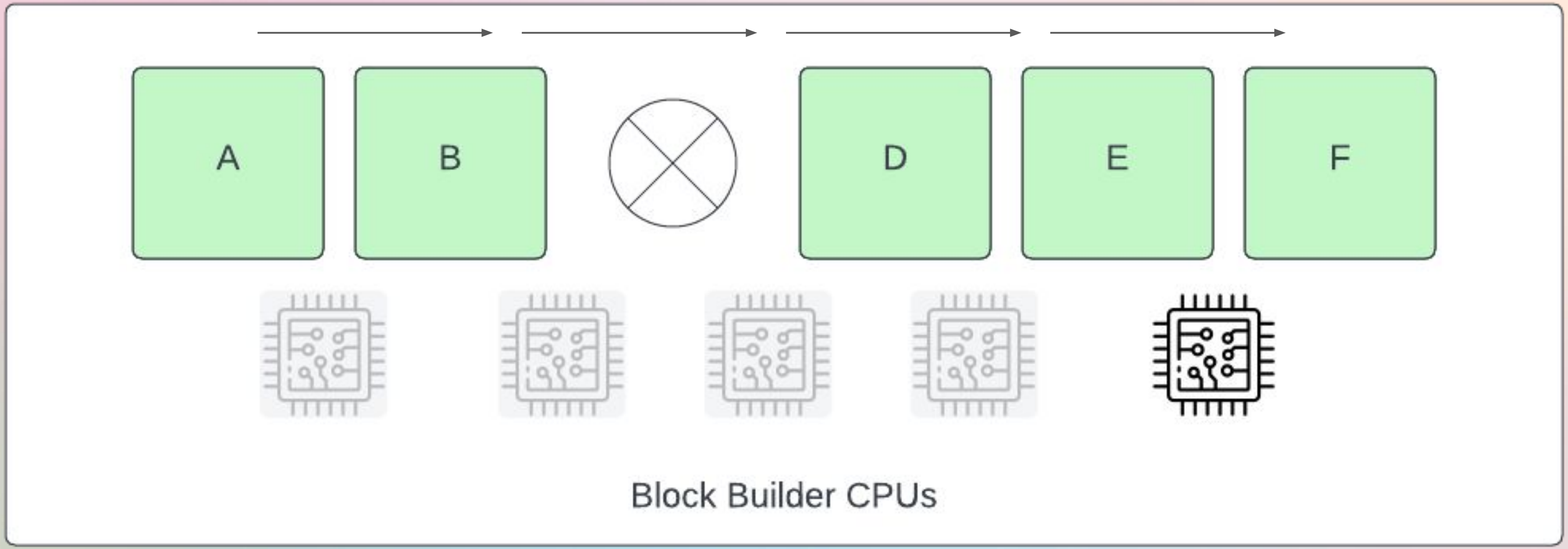
Secure, efficient and parallelized transaction validation

Transactions that do not pass validations are not included in a block.



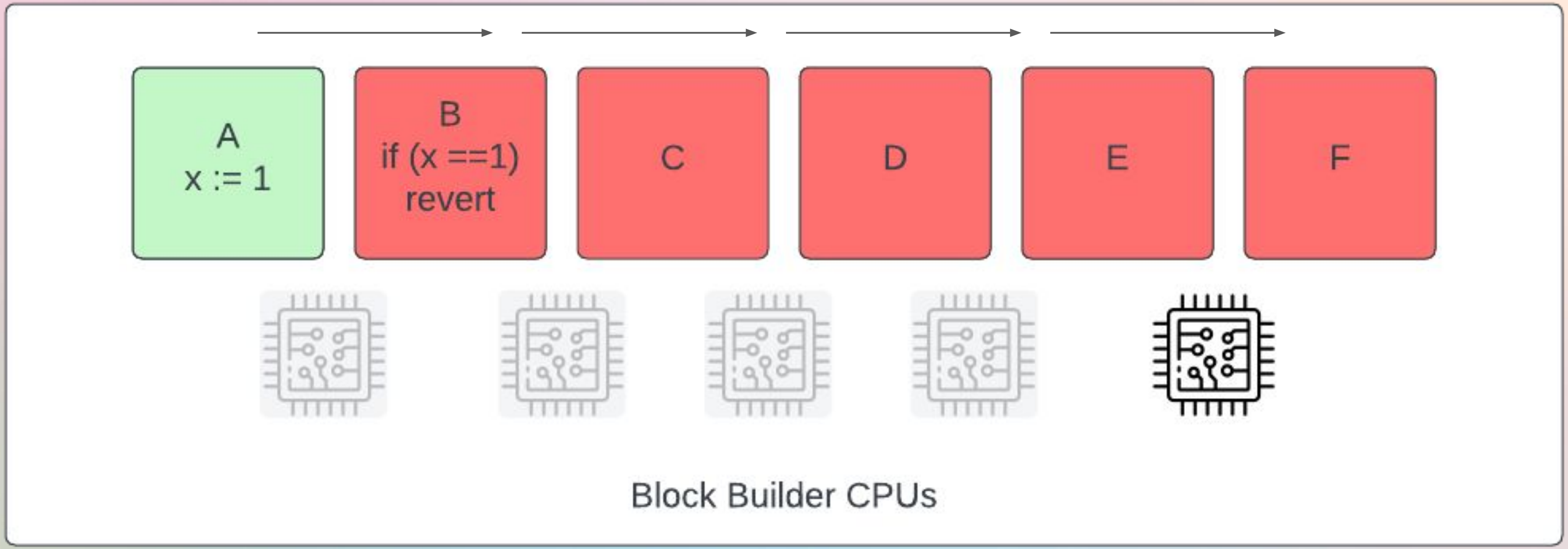
Secure, efficient and parallelized transaction validation

If the sandboxing works correctly the resulting block will be valid.



Mass transaction invalidation problem

If validation accessible data overlaps transactions can invalidate each other.
This is also a DoS attack vector if mempool is filled with mutually exclusive txs.



What should interested L2 developers do now ?

- Get in touch with us and other L2s interested in RIP-7560
- Get familiar with RIP-7560/EIP-7701 and provide feedback
- Have a look at a prototype ‘go-ethereum’ fork with RIP-7560 in it
  - https://github.com/eth-infinitism/go-ethereum/tree/RIP-7560-revision-2
- Take part in a RollCall event and add RIP-7560 to the call’s agenda
  - https://github.com/ethereum/pm
- Just build it

Get in touch

https://www.erc4337.io/

https://www.rip7560.io/

https://discord.gg/kUwZhJU2gc