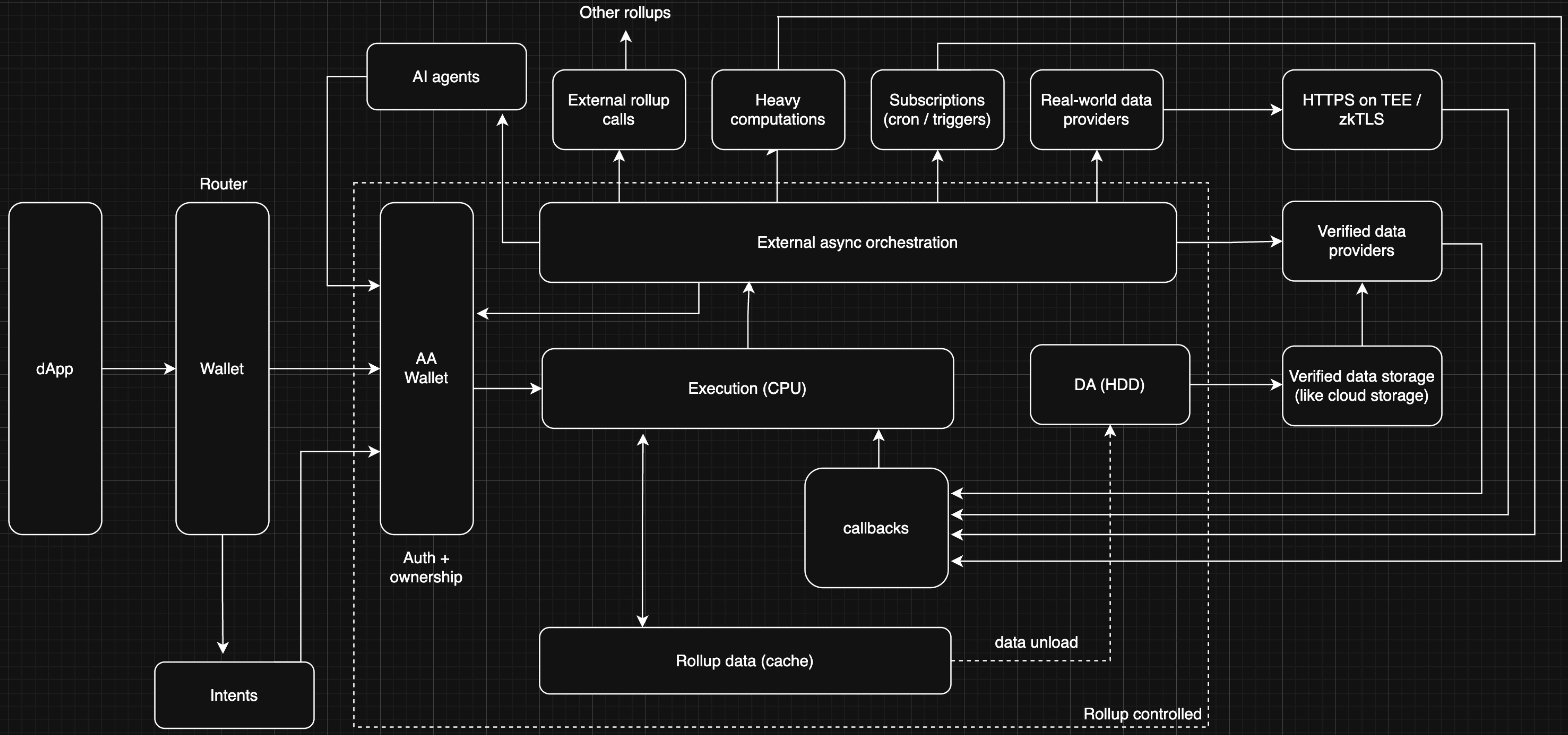
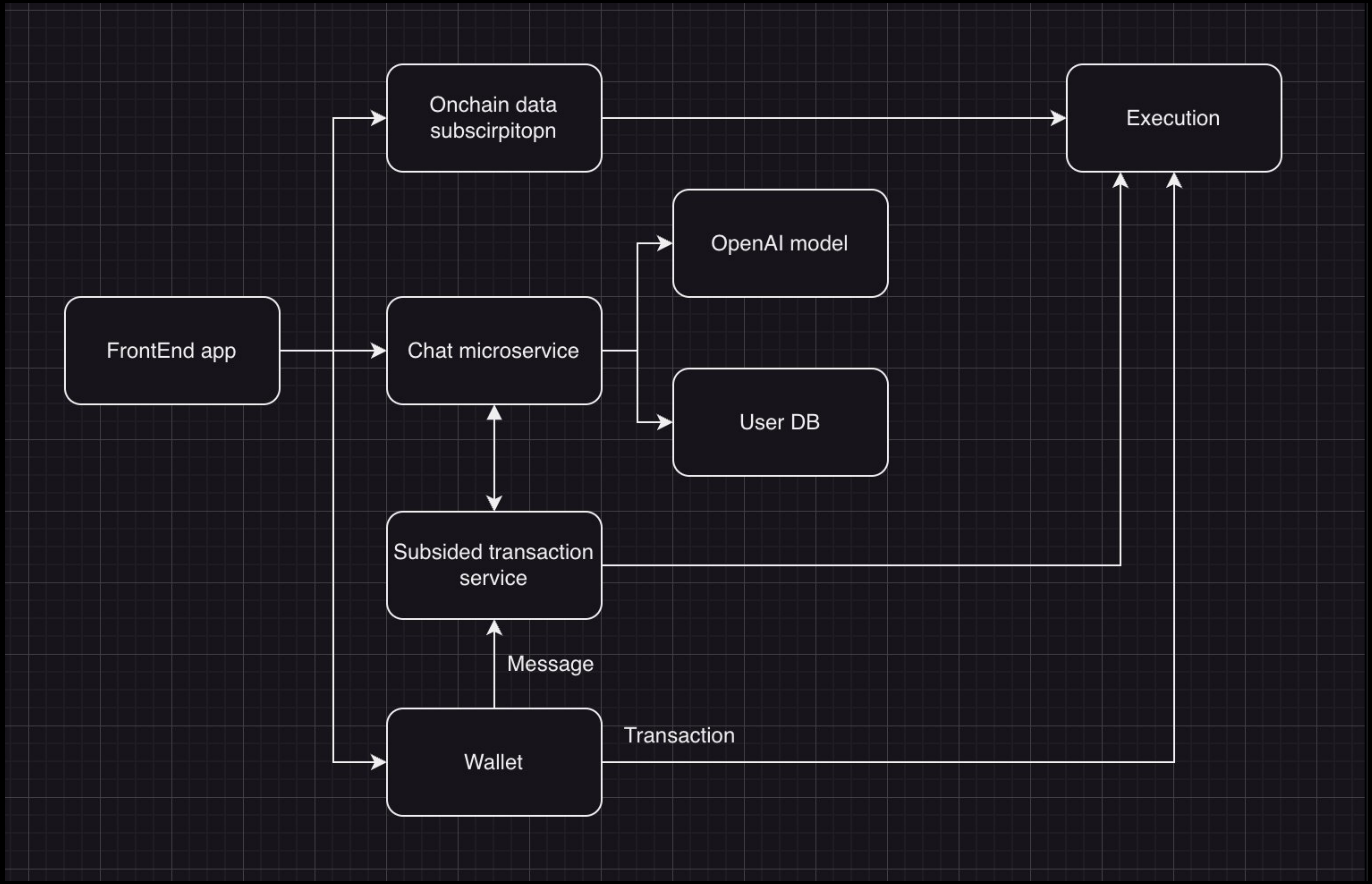
Building rollup-centric app

Gearbox Foundation, 2025

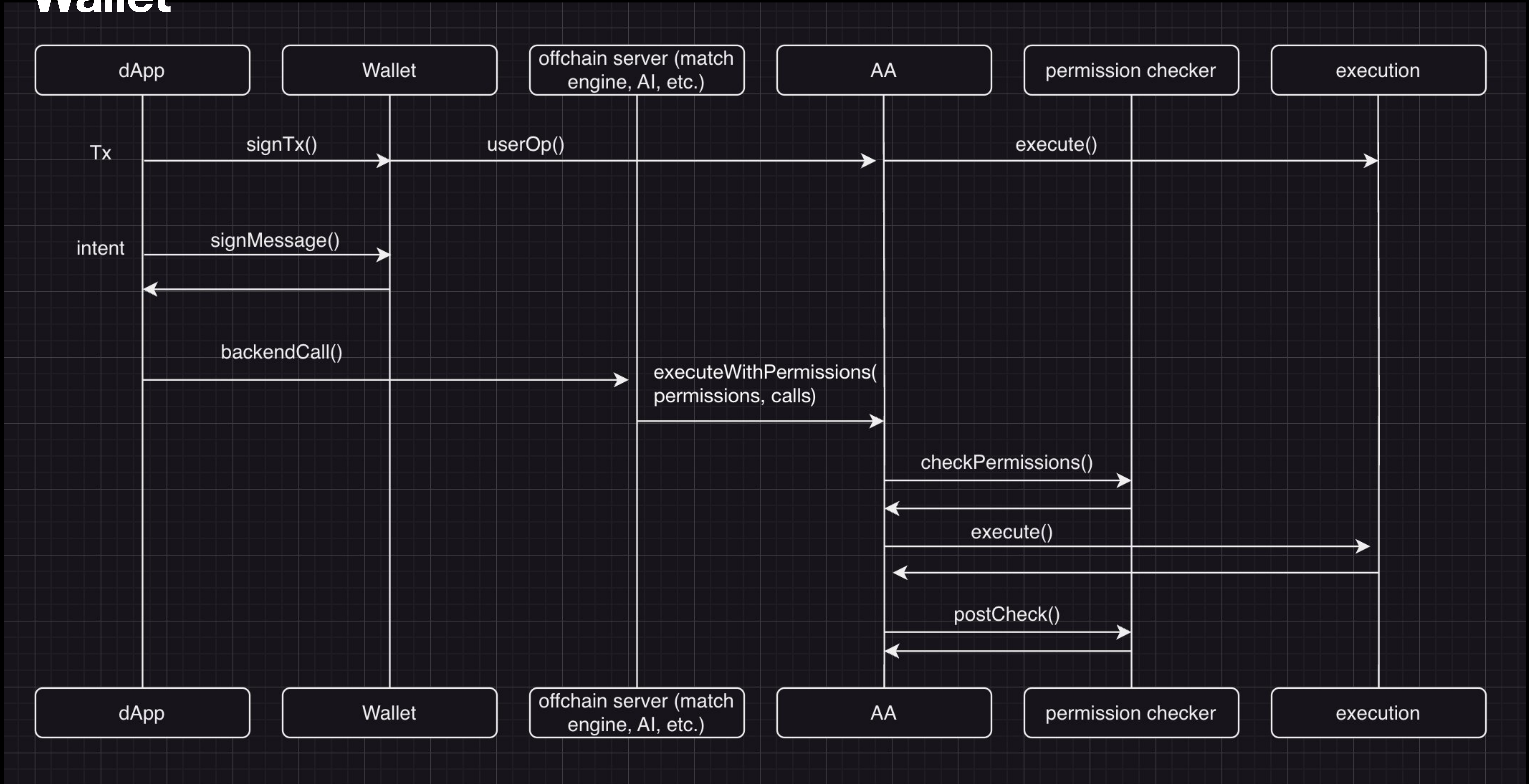
Architecture



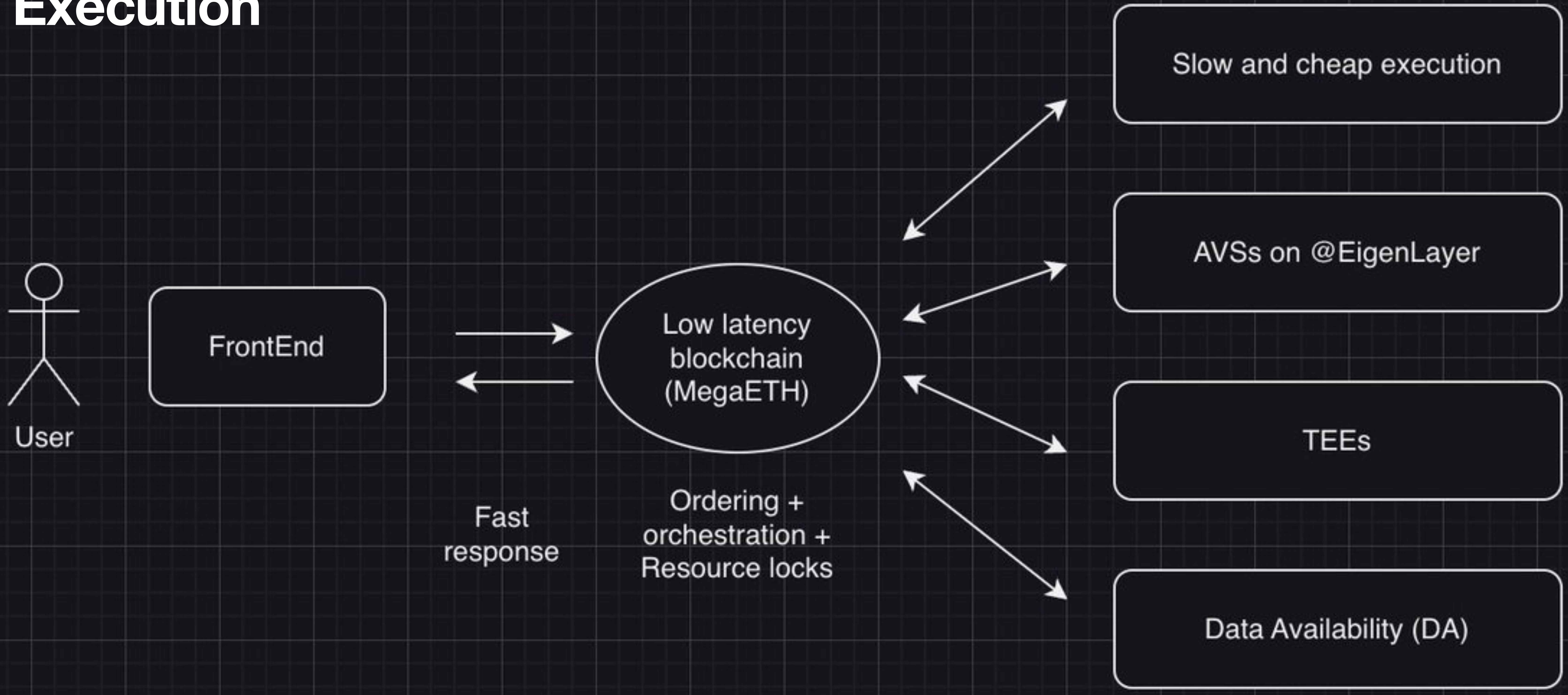
Frontend


Onchain data subscirpitopn
Execution
OpenAI model
FrontEnd app
Chat microservice
User DB
Subsided transaction service
Message
Wallet
Transaction

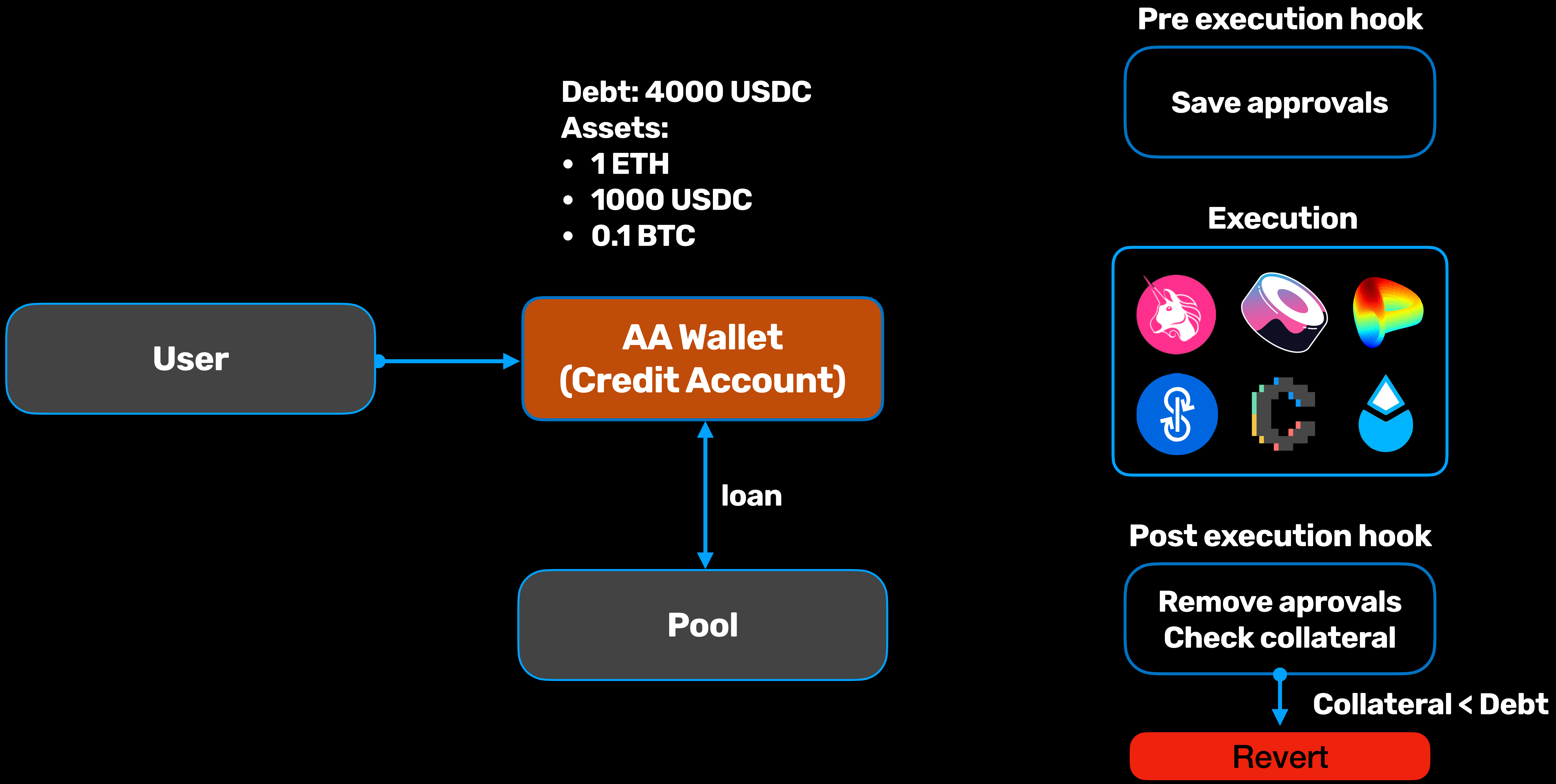
Wallet


dApp
Wallet
offchain server (match engine, AI, etc.)
AA
permission checker
execution
Tx
signTx()
userOp()
execute()
intent
signMessage()
backendCall()
executeWithPermissions(
permissions, calls)
checkPermissions()
execute()
postCheck()
dApp
Wallet
offchain server (match engine, AI, etc.)
AA
permission checker
execution

Execution


User
FrontEnd
Low latency blockchain (MegaETH)
Fast response
Ordering + orchestration + Resource locks
Slow and cheap execution
AVSs on @EigenLayer
TEEs
Data Availability (DA)

Gearbox 101


Debt: 4000 USDC
Assets:
• 1 ETH
• 1000 USDC
• 0.1 BTC
User
AA Wallet (Credit Account)
loan
Pool
Pre execution hook
Save approvals
Execution
Post execution hook
Remove aprovals
Check collateral
Collateral < Debt
Revert

Credit account vs Pool based models

	Pool based model (Aave)	Credit account model (Gearbox)
Transaction	<u>Account</u> • 1 ETH • 0.1 BTC → <u>Account</u> • 1 locked ETH • 3000 USDC • 0.1 BTC	<u>Credit account</u> • 1 ETH • 0.1 BTC → <u>Credit account</u> • 1 ETH • 3000 USDC • 0.1 BTC Debt: 3000 USDC
ETH	You can't use ETH until repay debt	You can use ETH until collateral > debt
USDC	You can use USDC as you want	You can use USDC until collateral > debt
BTC	You can use BTC as you want	As you want if it's not used as collateral, otherwise until collateral > debt

Transaction example (simplified)

```
batch = [
  IERC20(WETH).approve(UniswapRouter, type(uint256).max),
  UniswapRouter.swapExactTokenToTokens(WETH, USDC, 1 eth)
];
```

Pre-execution hook:

```
- iterate across the batch, stores WETH approval to callApprovals array.
```

Execution:

```
- batch executed as usual
```

Post-execution hook:

```
- remove allowance based on callApprovals
- check if it’s enough collateral:
      tvw = (0 eth) * 3000 USDC/eth * 92% LTV + (3000 + 3000) USDC * 98% LTV
               + 0.1 BTC * 100_000 * 92% LTV) USDC = 15,080 USDC
      require(debt (3000 USDC) < 15,080 USDC)
```

Fat account thesis

Account Abstraction

Safe Biconomy coinbase

1inch TRUST AMBIRE

Credit Abstraction

Gearbox

Chain Abstraction

INFINEX SOCKET OneBalance

Everclear LI.FI ACROSS

Ethereum Standards

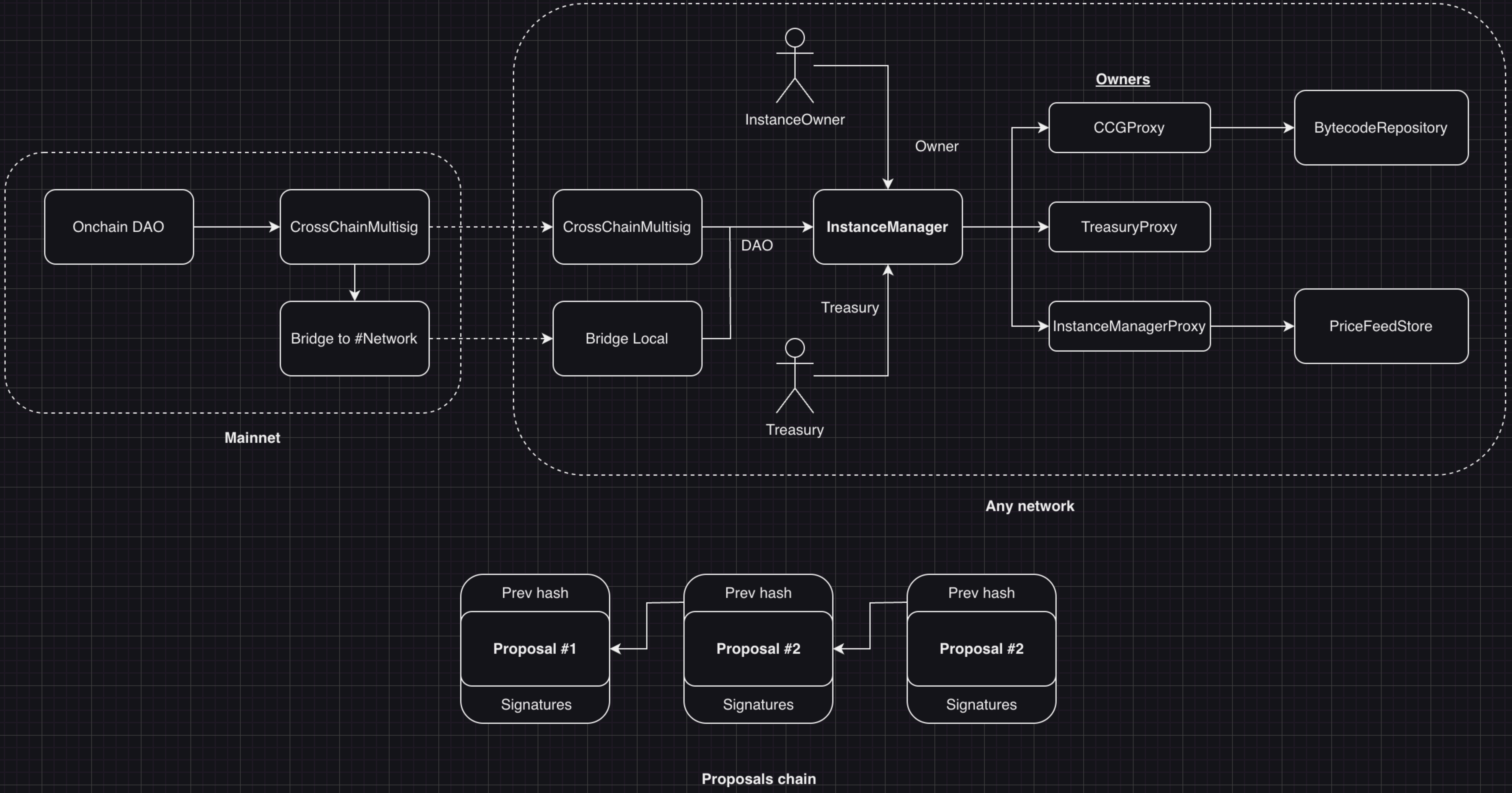
ERC-4337 ERC-7579

ERC-7702 ERC-3074

Goals

- RAAS ready. Permissionless deploy on any chain
- Permissionless management & plugins
- Liquidity layer across whole Ethereum ecosystem
- Supporting initial intents infra

RAAS ready


Onchain DAO
CrossChainMultisig
Bridge to #Network
Mainnet
CrossChainMultisig
Bridge Local
DAO
InstanceOwner
Owner
InstanceManager
Treasury
Treasury
Owners
CCGProxy
BytecodeRepository
TreasuryProxy
InstanceManagerProxy
PriceFeedStore
Any network
Prev hash
Proposal #1
Signatures
Prev hash
Proposal #2
Signatures
Prev hash
Proposal #2
Signatures
Proposals chain

New rollup deployment process

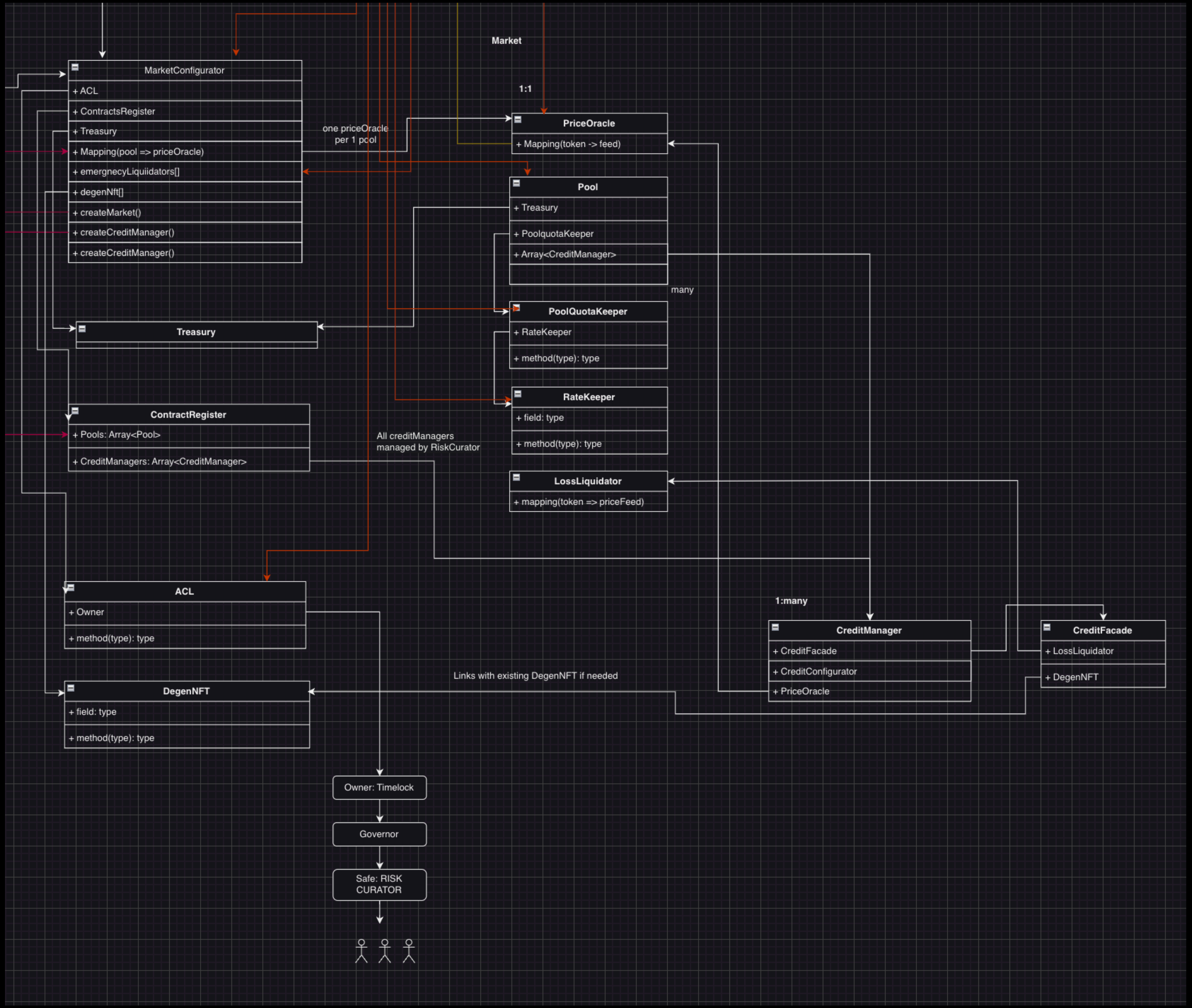
1. Deploy CrossChainMultisig via CREATE2 -> deterministic address
2. Deploy InstanceManager(ccg) -> determinitsic address
3. InstanceManager deploys BytecodeRepository (which is storage of verified bytecode)
4. Execute all signed proposals on local CCG -> they will be automatically applied
5. Proposals:
   - Add auditors (party who can sign bytecode in repository)
   - Add system contracts (system factories could be added only by DAO)
   - Deploy system contracts.

As result: proposal chain creates deterministic setup on any rollup without dev involvement.
Then DAO votes for instance manager and this proposal delivered as latest, so we have
a warranty that everything was deployed and setup properly

Modularity needs BytecodeRepository

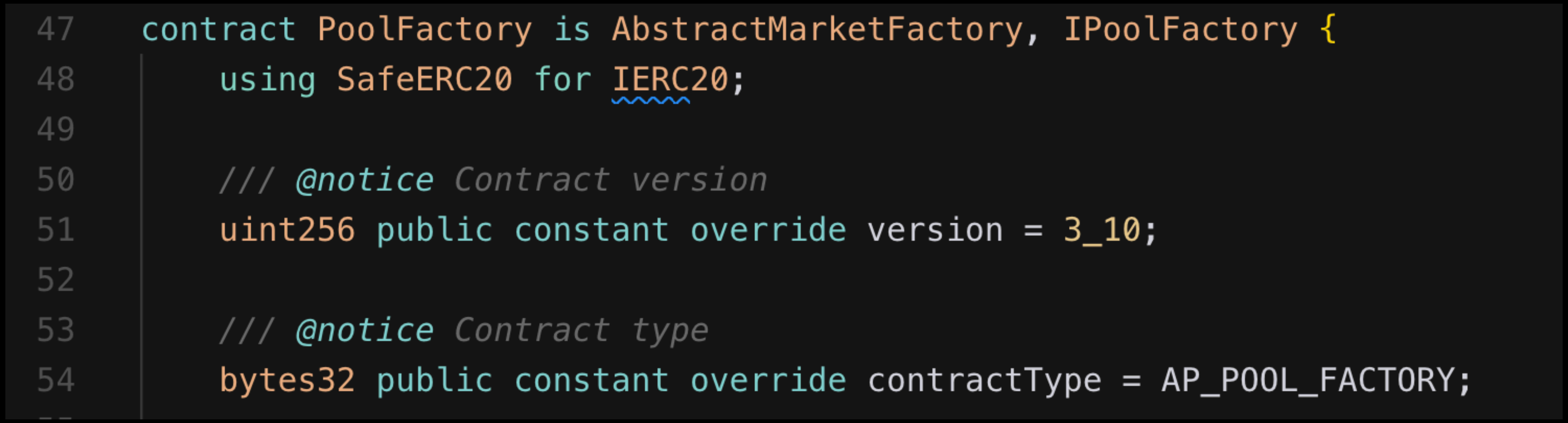
Plugins:

- Interest rate models
- Rate models
- Price feeds (based on some conracts)
- Loss policies
- Core contracts migration



Bytecode repository & version control

1. Each contract has contractType (bytes32) and version.

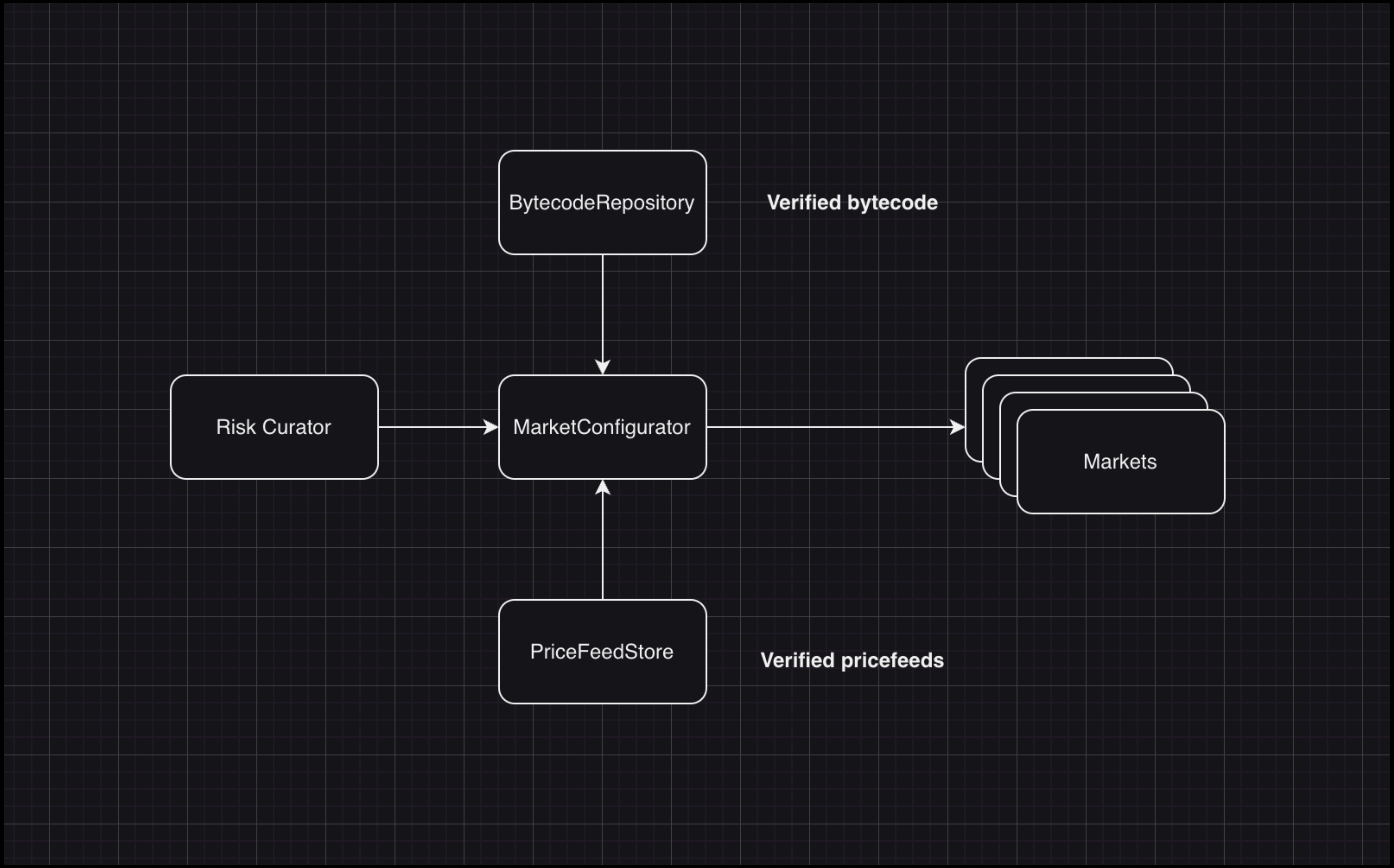


```
contract PoolFactory is AbstractMarketFactory, IPoolFactory {
    using SafeERC20 for IERC20;

    /// @notice Contract version
    uint256 public constant override version = 3_10;

    /// @notice Contract type
    bytes32 public constant override contractType = AP_POOL_FACTORY;
```

2. Everyone can upload a contract to BCR (permissionless).
3. Contracts are listed in BCR only with one auditor signature. Once it’s done, particular bytecode is assigned with contractType / version.
4. System contracts could be added by DAO only (requires voting)
5. Domain system: “PRICE_FEED::ERC4626”, “IRM::LINEAR”. Each domain represents supported interface.

Permissonless management across ecosystem

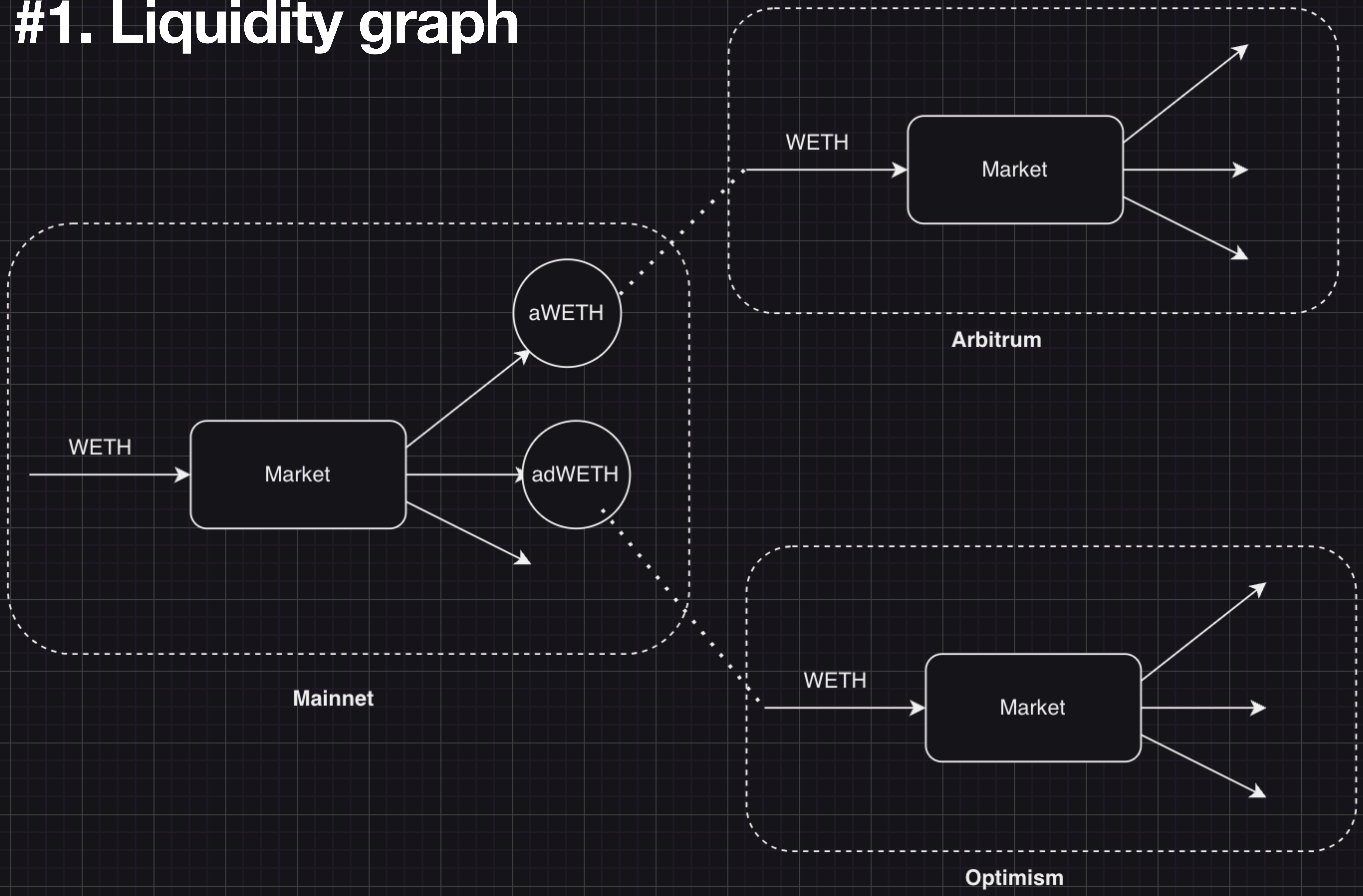
Verified bytecode & pricefeed on any network



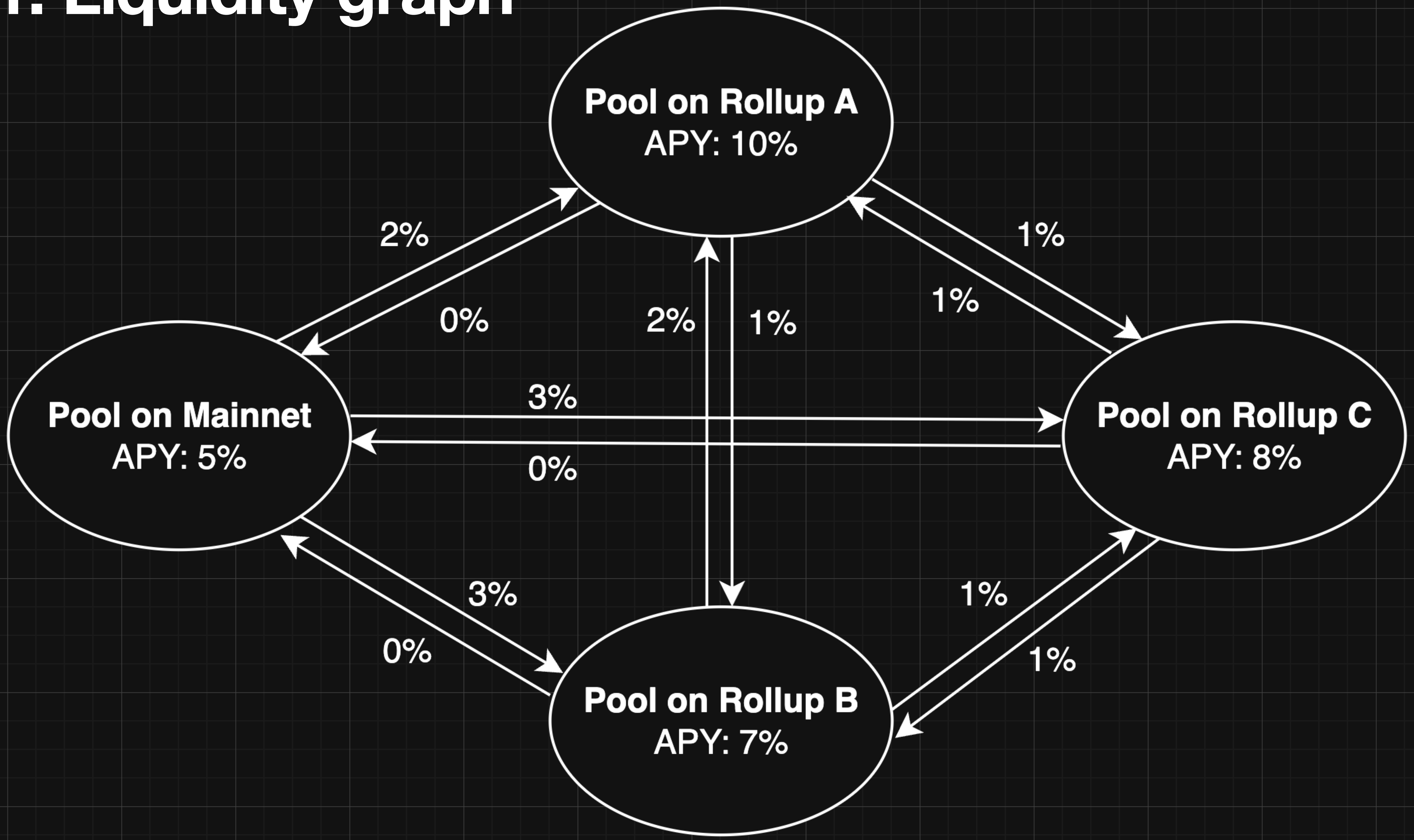
Rollup centric protocol management

1. InstanceManager has the same address on any EVM-network, could be used for market discovery
2. Anybody can creates and manage markets.
3. All bytecode is verifiable
4. New plugin contracts could be spreaded across all connected chains
5. Contact updates propagated automatically, the last decision is always on Risk curators.
6. Everyone can verify contract bytecode, who is auditor, etc.
7. Everyone can contribute to protocol plugins without any permissions
8. DAO can control fees management and crucial decisions, however, it’s authority is restricted

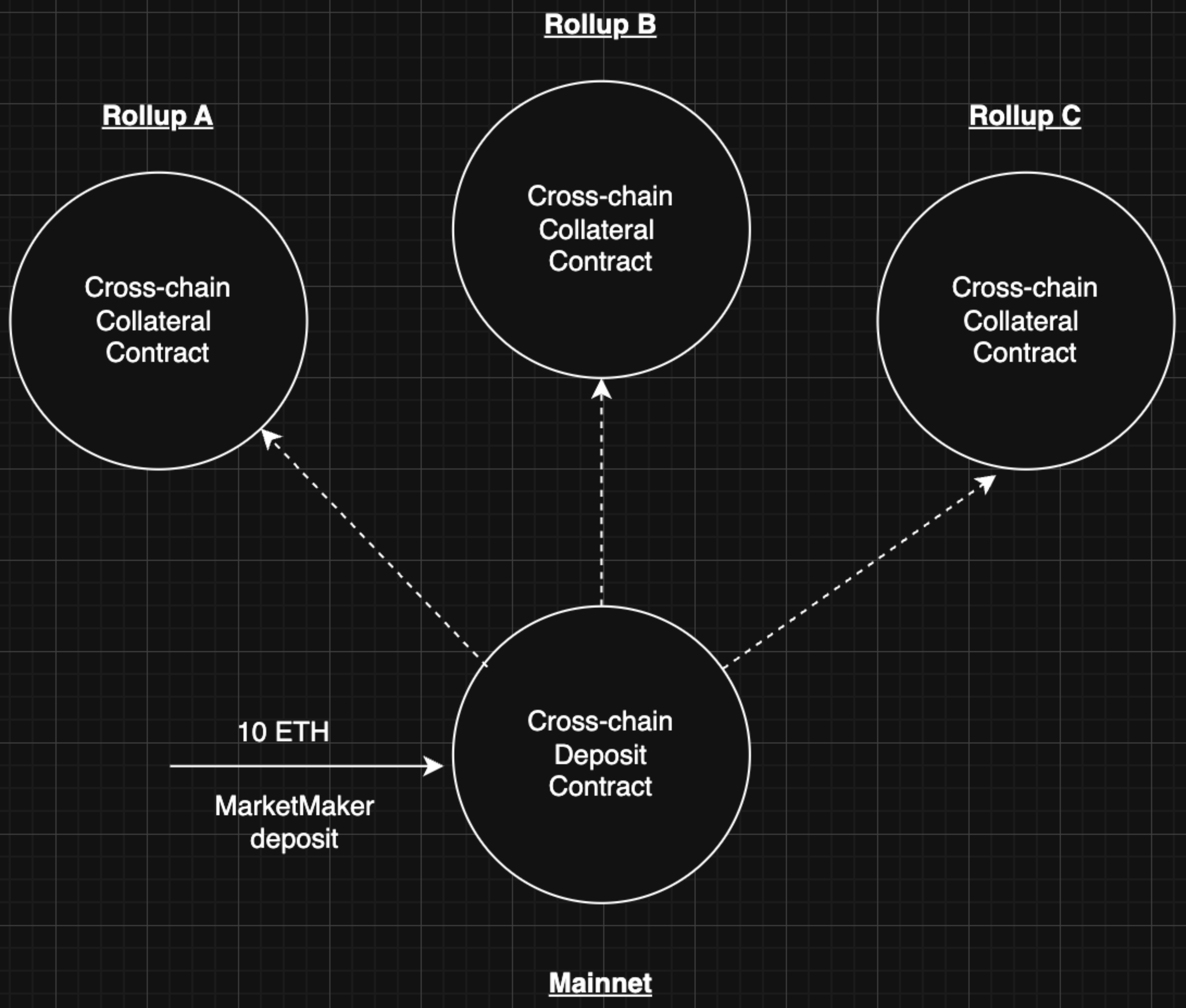
App #1. Liquidity graph



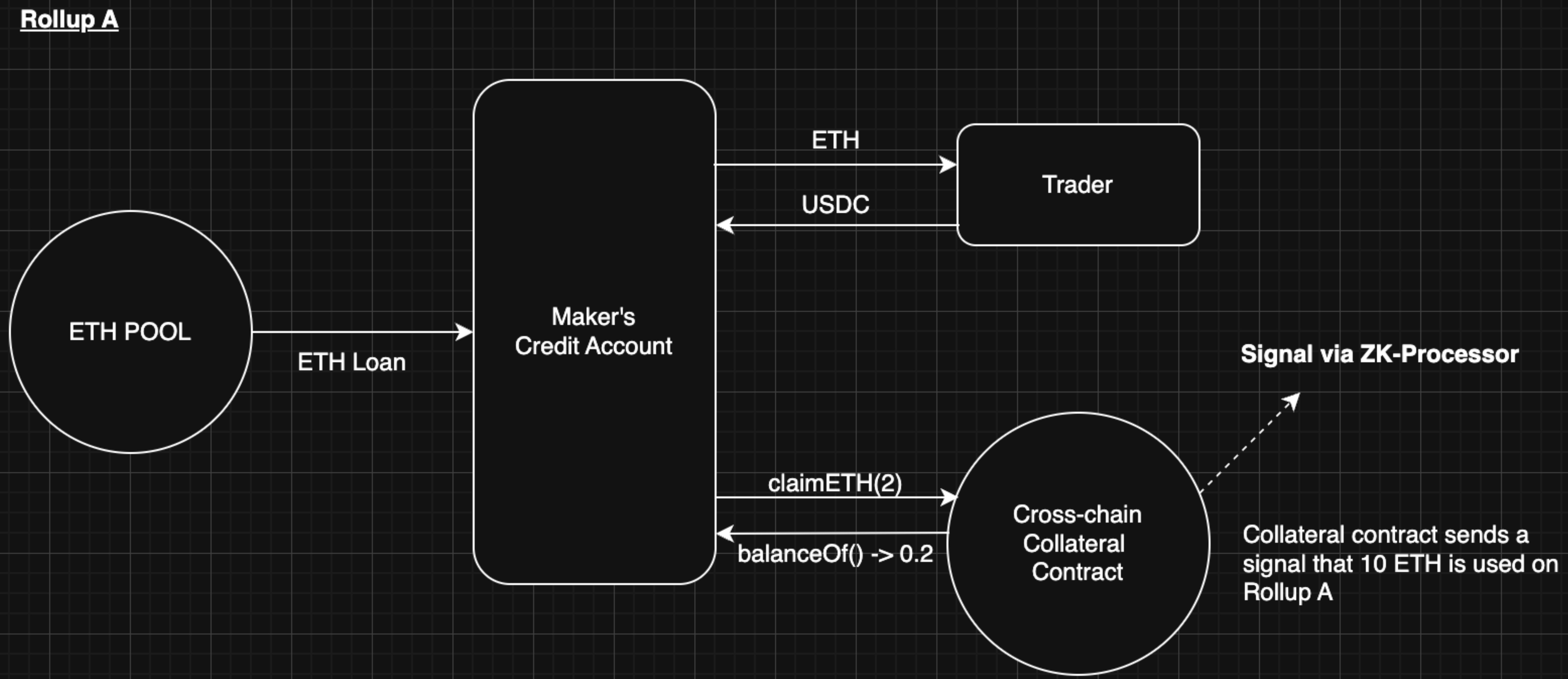
App #1. Liquidity graph



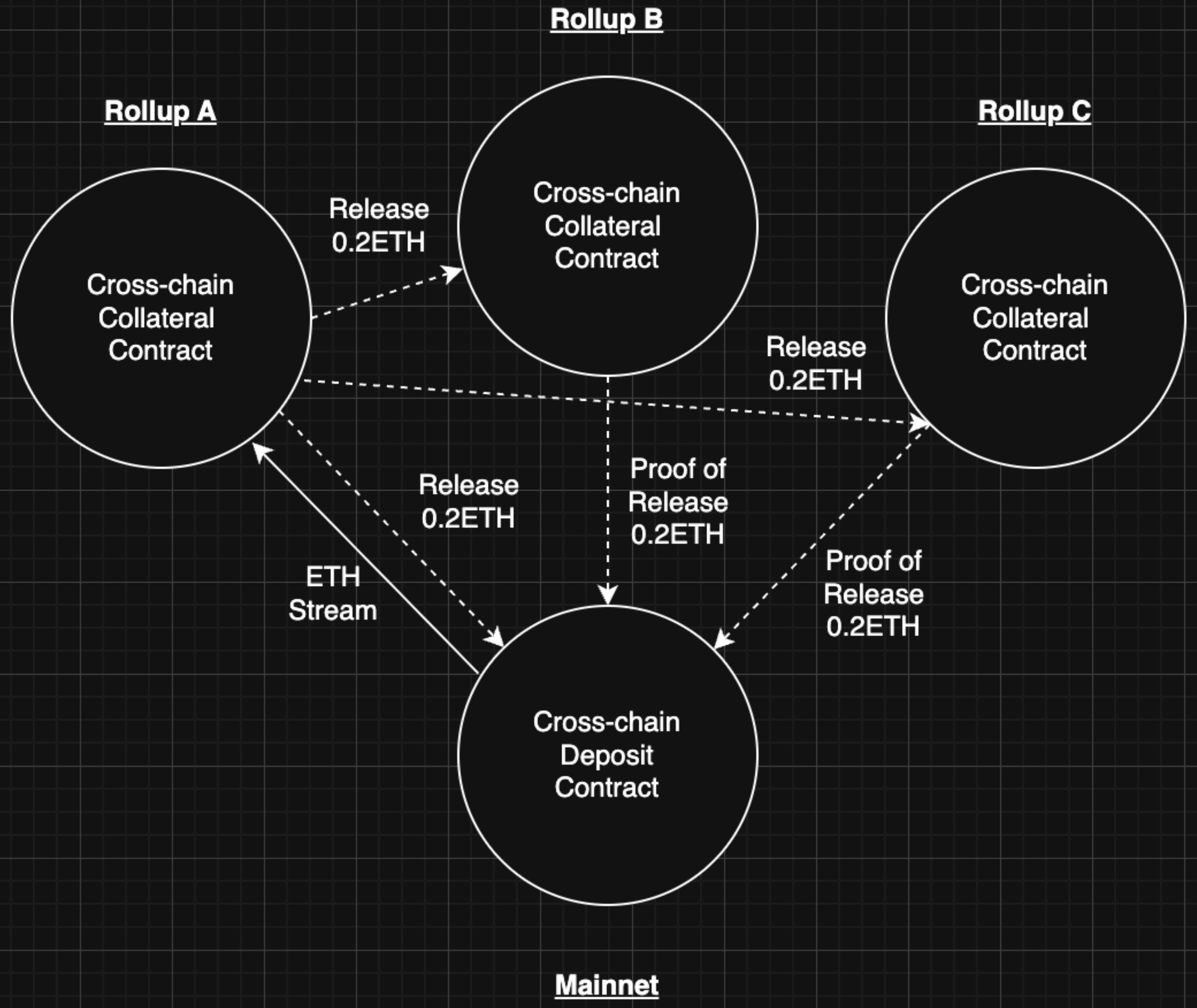
App #2. Solver credit


Rollup B
Rollup A
Rollup C
Cross-chain Collateral Contract
Cross-chain Collateral Contract
Cross-chain Collateral Contract
10 ETH
MarketMaker deposit
Cross-chain Deposit Contract
Mainnet

App #2. Solver credit



App #2. Solver credit



Thanks for your attention!

Let’s stay in touch: @0xmikko_eth