Ethereum on Python

中国Python开发者大会

jan@cryptape.com

@janx
peatio
pyethapp
pyethereum
ruby-ethereum
cita

Bitcoin
Decentralized Data Layer

Ethereum
Decentralized Computation Layer

Pattern

Cryptography

P2P Network

Consensus

Authenticated Data Structure

DEVP2P

Node Discovery (UDP)

- Kademlia
- Bootstrap nodes

Secure Transport (TCP)

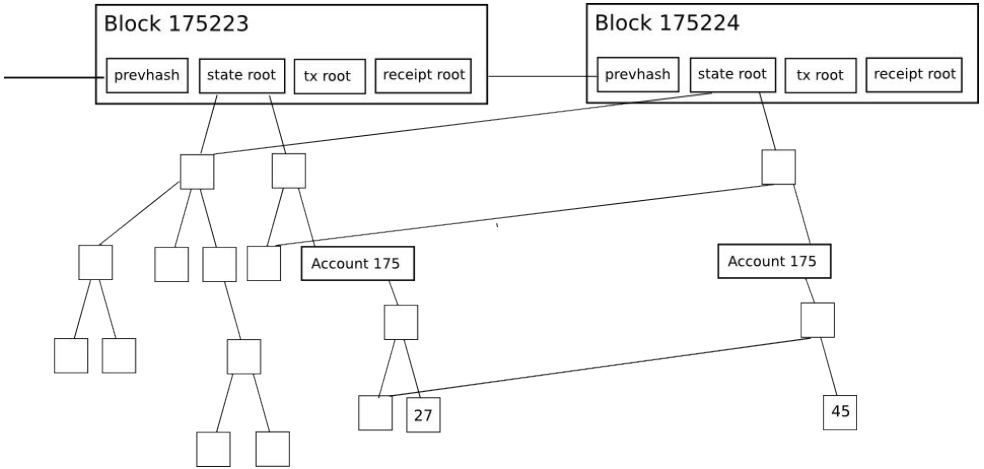
- Handshake
- Transport
- Multiplexer
- Flow Control

Authenticated Data Structure

Hash

List

Tree



github.com/ethereum/
pyethereum

```
class Block(rlp.Serializable):

    fields = [
        ('header', BlockHeader),
        ('transactions', CountableList(Transaction)),
        ('uncles', CountableList(BlockHeader))
    ]
```

ethereum/block.py

```
class BlockHeader(rlp.Serializable):

    fields = [
        ('prevhash', hash32),
        ('uncles_hash', hash32),
        ('coinbase', address),
        ('state_root', trie_root),
        ('tx_list_root', trie_root),
        ('receipts_root', trie_root),
        ('bloom', int256),
        ('difficulty', big_endian_int),
        ('number', big_endian_int),
        ('gas_limit', big_endian_int),
        ('gas_used', big_endian_int),
        ('timestamp', big_endian_int),
        ('extra_data', binary),
        ('mixhash', binary),
        ('nonce', binary)
    ]
```

ethereum/block.py

```
class Account(rlp.Serializable):

    fields = [
        ('nonce', big_endian_int),
        ('balance', big_endian_int),
        ('storage', trie_root),
        ('code_hash', hash32)
    ]
```

ethereum/state.py

```
class Transaction(rlp.Serializable):

    fields = [
        ('nonce', big_endian_int),
        ('gasprice', big_endian_int),
        ('startgas', big_endian_int),
        ('to', utils.address),
        ('value', big_endian_int),
        ('data', binary),
        ('v', big_endian_int),
        ('r', big_endian_int),
        ('s', big_endian_int),
    ]
```

ethereum/transactions.py

Account Model

Contract Account

Machine

Program

External Account

Human

Private Key

PoW
Ethash

15s

Uncles

ASIC Resistance

EVM
Ethereum Virtual Machine

Minimal

Safe

Deterministic

Bounded Turing-complete

Gas Mechanism

Halting Problem

Opcode Pricing

Example:

Tx cost = 21000 gas * 20 shannon per gas = 0.00042 ether

```
opcodes = {
    0x00: ['STOP', 0, 0, 0],
    0x01: ['ADD', 2, 1, 3],
    0x02: ['MUL', 2, 1, 5],
    0x03: ['SUB', 2, 1, 3],
    0x04: ['DIV', 2, 1, 5],
    0x05: ['SDIV', 2, 1, 5],
    0x06: ['MOD', 2, 1, 5],
    0x07: ['SMOD', 2, 1, 5],
    0x08: ['ADDMOD', 3, 1, 8],
    0x09: ['MULMOD', 3, 1, 8],
    0x0a: ['EXP', 2, 1, 10],
    0x0b: ['SIGNEXTEND', 2, 1, 5],
```

ethereum/opcodes.py

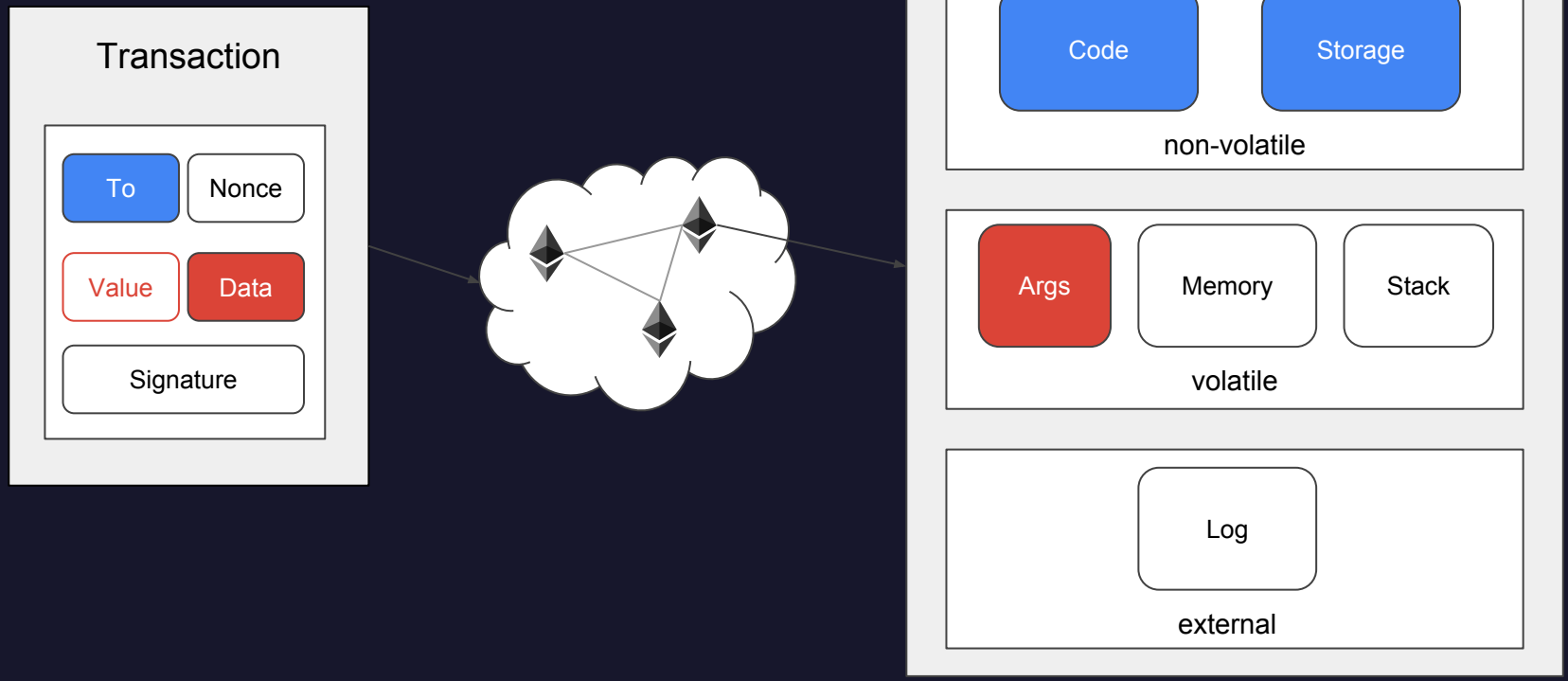
Precompile Contract

“System Applications”

- 0x01 - ecrecover
- 0x02 - sha256
- 0x03 - ripemd160
- 0x04 - identity

Metropolis

- Bigint modular exponentiation
- Addition and scalar multiplication on alt_bn128
- Ate pairing check on alt_bn128



Submit & Transaction Execution

```python
# Applies the block-level state transition function
def apply_block(state, block):
    # Pre-processing and verification
    snapshot = state.snapshot()
    cs = get_consensus_strategy(state.config)
    try:
        # Start a new block context
        cs.initialize(state, block)
        # Basic validation
        assert validate_header(state, block.header)
        assert cs.check_seal(state, block.header)
        assert cs.validate_uncles(state, block)
        assert validate_transaction_tree(state, block)
        # Process transactions
        for tx in block.transactions:
            apply_transaction(state, tx)
        # Finalize (incl paying block rewards)
        cs.finalize(state, block)
        # Verify state root, tx list root, receipt root
        #print('std', state.to_dict())
        assert verify_execution_results(state, block)
        # Post-finalize (ie. add the block header to the state for now)
        post_finalize(state, block)
    except (ValueError, AssertionError) as e:
        state.revert(snapshot)
        raise e
    return state
```

ethereum/meta.py

```
def apply_transaction(state, tx):
    ...
    validate_transaction(state, tx)
    ...

    message_data = vm.CallData([safe_ord(x) for x in tx.data], 0, len(tx.data))
    message = vm.Message(
        tx.sender,
        tx.to,
        tx.value,
        tx.startgas -
        intrinsic_gas,
        message_data,
        code_address=tx.to)

    # MESSAGE
    ext = VMExt(state, tx)

    if tx.to != b'':
        result, gas_remained, data = apply_msg(ext, message)
    else:  # CREATE
        result, gas_remained, data = create_contract(ext, message)

    ...
    gas_used = tx.startgas - gas_remained
    ...
```

ethereum/messages.py

```
def apply_transaction(state, tx):
    ...
    validate_transaction(state, tx)
    ...

    message_data = vm.CallData([safe_ord(x) for x in tx.data], 0, len(tx.data))
    message = vm.Message(
        tx.sender,
        tx.to,
        tx.value,
        tx.startgas -
        intrinsic_gas,
        message_data,
        code_address=tx.to)

    # MESSAGE
    ext = VMExt(state, tx)

    if tx.to != b'':
        result, gas_remained, data = apply_msg(ext, message)
    else:  # CREATE
        result, gas_remained, data = create_contract(ext, message)

    ...
    gas_used = tx.startgas - gas_remained
    ...
```

ethereum/messages.py

```
def _apply_msg(ext, msg, code):
    # Transfer value, instaquit if not enough
    snapshot = ext.snapshot()
    if msg.transfers_value:
        if not ext.transfer_value(msg.sender, msg.to, msg.value):
            log_msg.debug('MSG TRANSFER FAILED', have=ext.get_balance(msg.to),
                          want=msg.value)
            return 1, msg.gas, []

    # Main loop
    if msg.code_address in ext.specials:
        res, gas, dat = ext.specials[msg.code_address](ext, msg)
    else:
        res, gas, dat = vm.vm_execute(ext, msg, code)

    if res == 0:
        log_msg.debug('REVERTING')
        ext.revert(snapshot)

    return res, gas, dat
```

ethereum/messages.py

pyethapp
pyethereum
pydevp2p
py-evm

High Level Languages

Solidity, Serpent, Bamboo, Viper, …

ABI - Solidity Calling Convention

Formal Verification

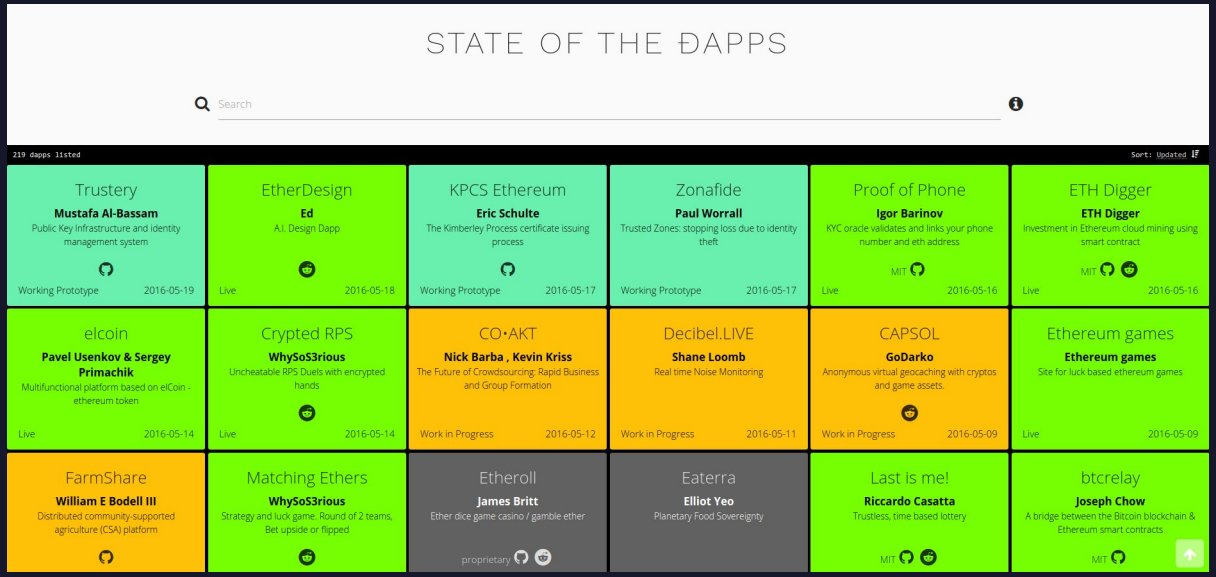
```
contract MyToken {
    /* This creates an array with all balances */
    mapping (address => uint256) public balanceOf;

    /* Initializes contract with initial supply tokens to the creator of the contract */
    function MyToken(
        uint256 initialSupply
        ) {
        balanceOf[msg.sender] = initialSupply;              // Give the creator all initial tokens
    }

    /* Send coins */
    function transfer(address _to, uint256 _value) {
        require(balanceOf[msg.sender] >= _value);           // Check if the sender has enough
        require(balanceOf[_to] + _value >= balanceOf[_to]); // Check for overflows
        balanceOf[msg.sender] -= _value;                    // Subtract from the sender
        balanceOf[_to] += _value;                           // Add the same to the recipient
    }
}
```

MyToken.sol

DApp

Smart Contract

UI + web3.js



CHINA
print('Hello PyConChina 2017')

CRYPTAPE

CITA on Rust

We're hiring!

join@cryptape.com