Ξthereum - Yellow Paper

Outline

- Motivation
- Blockchain
- Block, State, Transactions
- Gas, Payment
- Contracts
- Execution Model
- Block Finalization
- Applications

Motivation

"What Ethereum intends to provide is a blockchain with a built-in fully fledged ***Turing-complete programming language*** *which can be used to create* ***"contracts"*** *that encode* ***arbitrary state transition functions****, allowing users to create any of the systems described [below], as well as many others that we have not yet imagined, simply by writing up the logic in a few lines of code."*

- *custom currencies and financial instruments (coloured coins)*
- *ownership of an underlying physical device (smart property)*
- *non-fungible assets (e.x. Namecoin)*
- *digital assets controlled by code (smart contracts)*
- *decentralized autonomous organizations (DAOs)*

Blockchain

- "Transaction-based state machine"
  - Account balances
  - Information of physical world
  - Etc.
- Transactions collected into blocks + previous block ID
- Miners dedicate effort (work) to bolster one block over a competitor's block
- Blocks include reward to incentivize miners

Blockchain

New state after Ethereum state transition function,

$$\sigma_{t+1} \equiv \Upsilon(\sigma_t, T)$$

Block as a collection of transactions,

$$B \equiv (..., (T_0 , T_1 , \ldots))$$

Block transition = state transition (transaction) + block finalization

$$\Pi(\sigma, B) \equiv \Omega(B, \Upsilon(\Upsilon(\sigma, T_0), T_1)\ldots)$$

New state after block transition

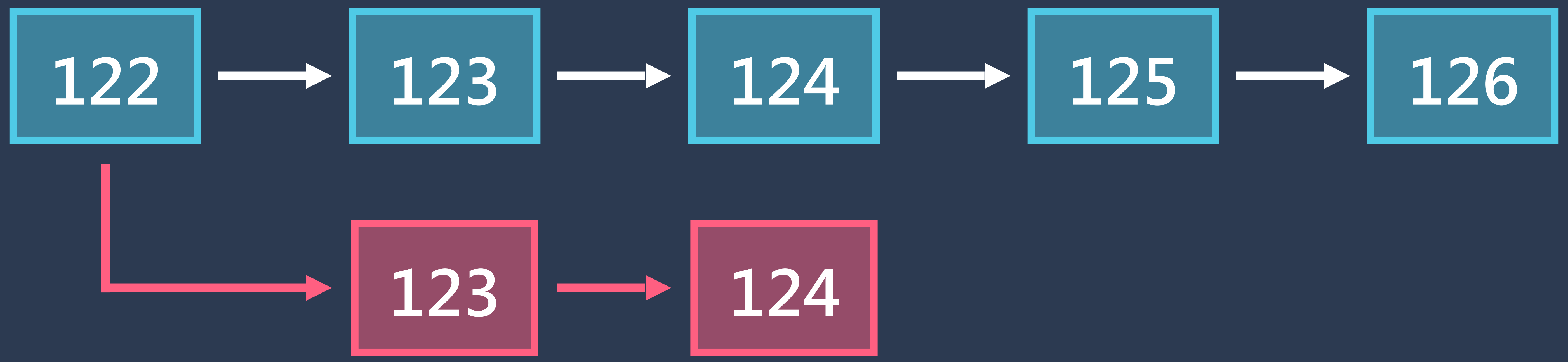
$$\sigma_{t+1} \equiv \Pi(\sigma_t, B)$$

Blockchain

- **Decentralized** system; all miners have the opportunity to create a new block
- To form **consensus** over which **path** from genesis block to leaf to use an **agreed upon scheme** is required
- Disagreements = **forks**; kills **confidence** in the system
- **GHOST protocol** is the agreed-upon scheme

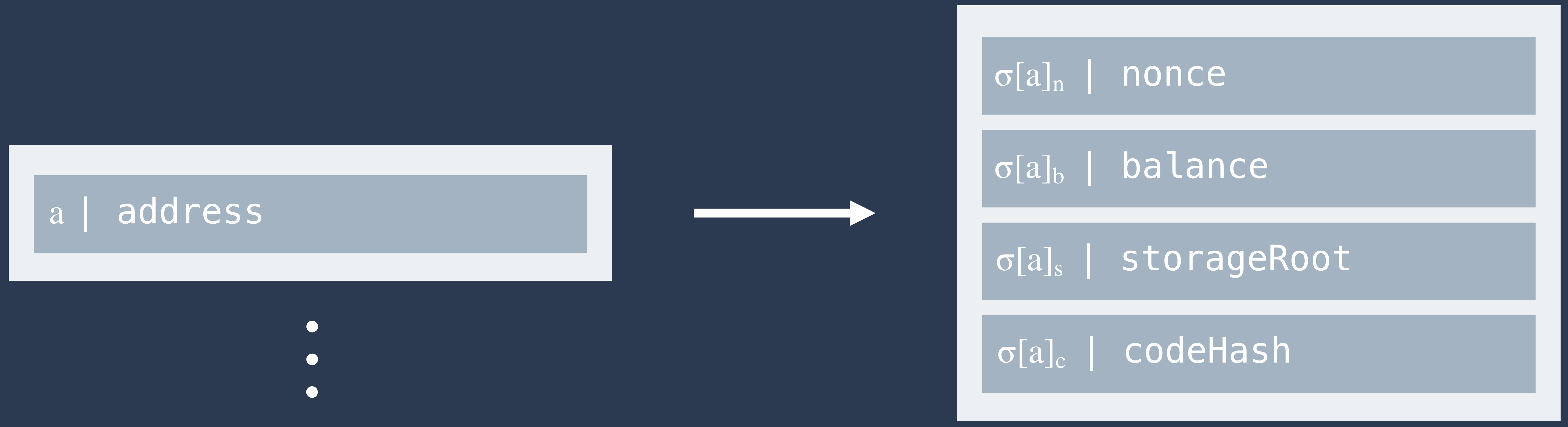
GHOST Protocol

- “Greedy Heaviest Observed Subtree”
- Fast confirmation times lead to **stale blocks** (wasted), and reduced security
- Exacerbated by miner centralization (pools)
- Include stale blocks (“Uncles”) to determine longest (“heaviest”) chain
- Uncle receives 87.5% of its base reward, nephew including it receives remaining 12.5%
- Only considers up to 7 generations



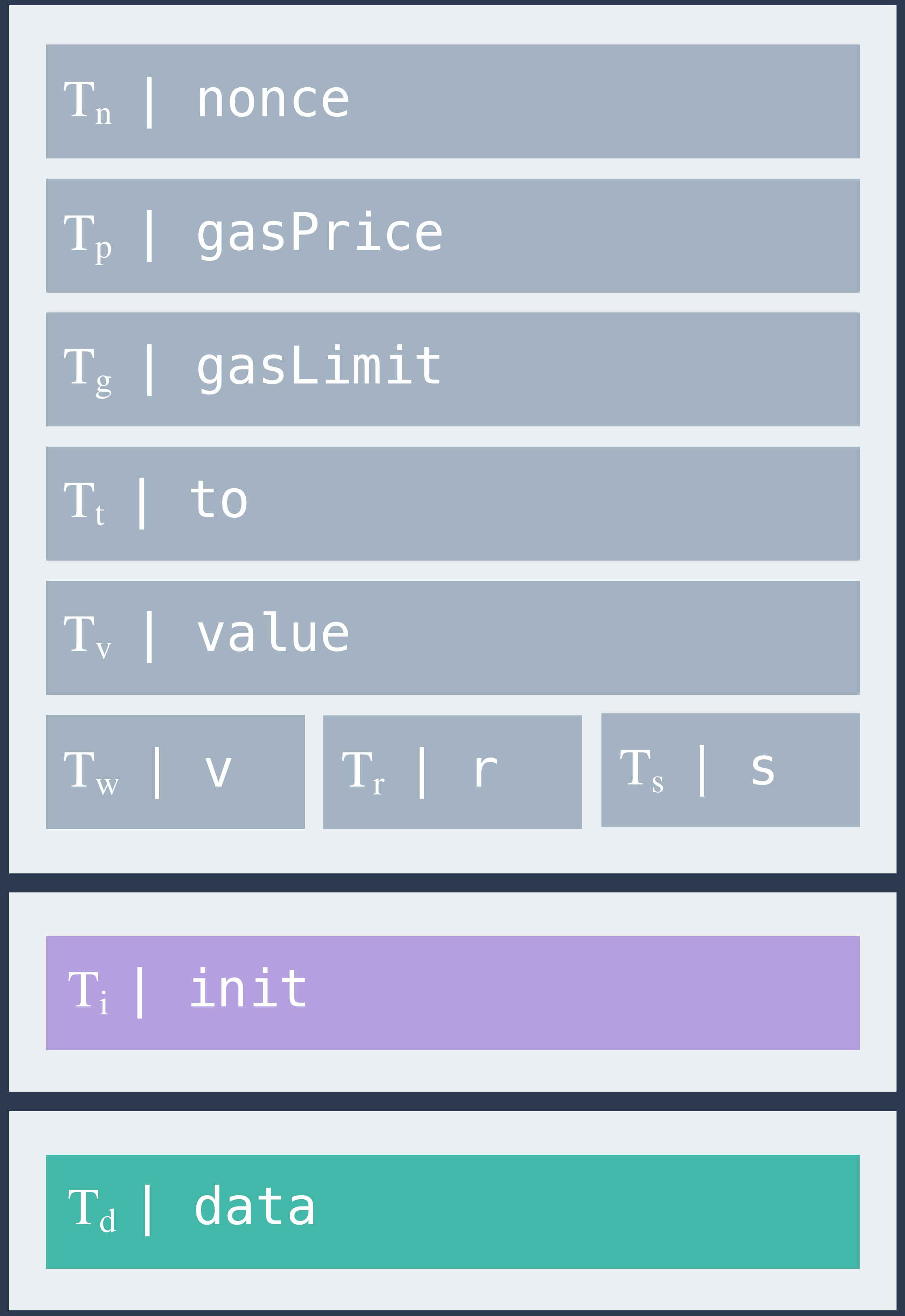
Block, State, Transactions: Account State

- **World state** (state tree) maps addresses (160-bit identifiers) and account states
- Mapping stored in a Merkle Patricia tree **off of the blockchain**
- Tree depends on database backend ("state database")



Block, State, Transactions: The Transaction

- Two types of transactions
  1. **contract creation**
  2. **message calls**
- Resources cost gas: computation, and storage (transaction data)
- `gasPrice`, `gasLimit` put bounds on computation (to prevent DoS attacks)



Block, State, Transactions: The Block

- Block consists of header, $\mathbf{H}$, transactions, $\mathbf{T}$, ommer block headers, $\mathbf{U}$

H_p | parentHash	H_l | gasLimit
H_o | ommersHash	H_g | gasUsed
H_c | beneficiary	H_s | timestamp
H_r | stateRoot	H_x | extraData
H_t | transactionsRoot	H_m | mixHash, H_n | nonce
H_e | receiptsRoot	$\mathbf{T}$ | transactions ⋮
H_b | logsBloom	
H_d | difficulty	$\mathbf{U}$ | ommerBlockHeaders ⋮
H_i | number	

Block, State, Transactions: The Block

`parentHash`

- Hash of parent block's header

`ommersHash`

- Hash of ommer block header list

`beneficiary`

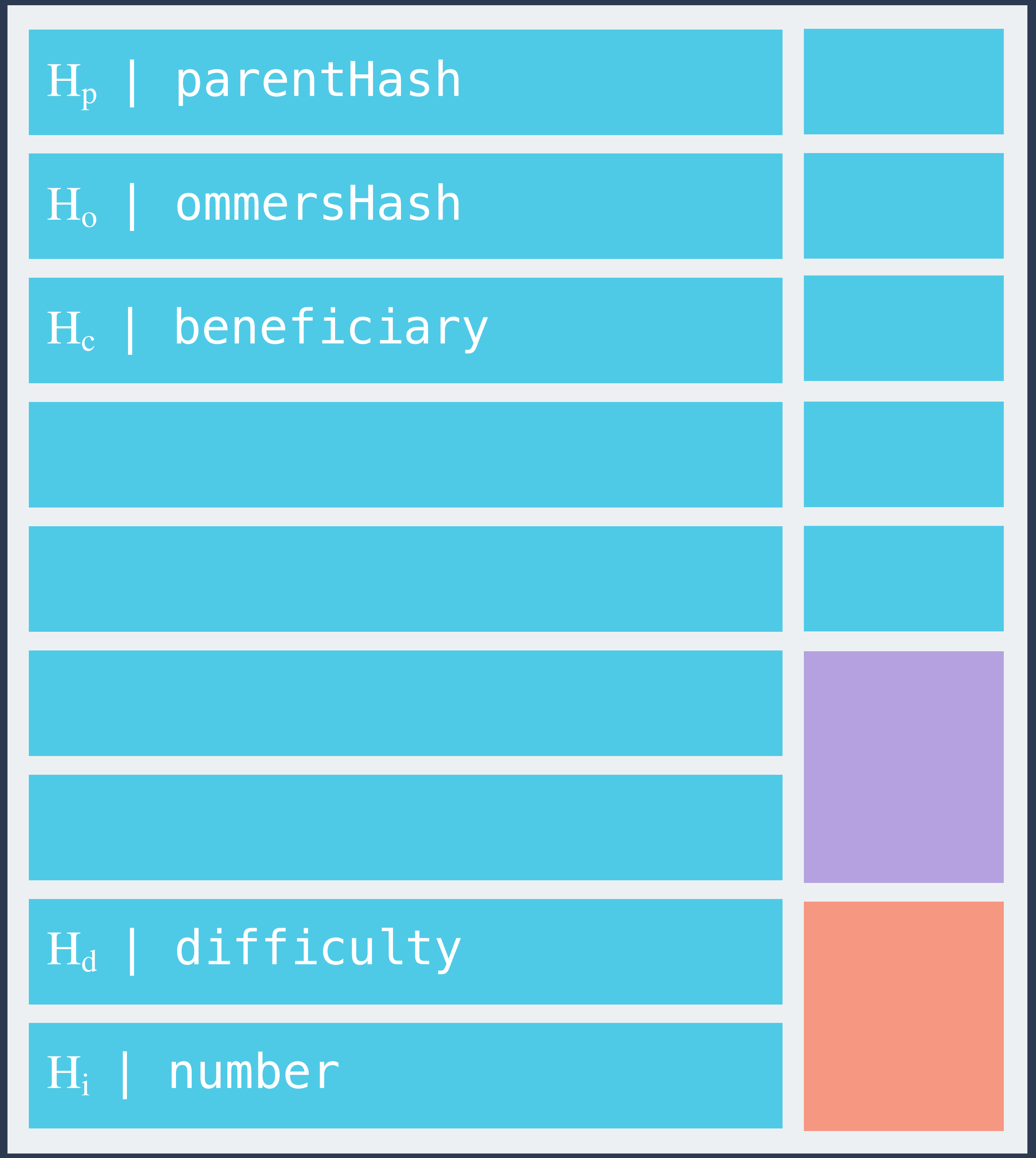
- Address where fees will be sent to

`difficulty`

- Derived from parent block's difficulty and timestamp

`number`

- Number of ancestor blocks



Block, State, Transactions: The Block

`stateRoot`

- Hash of root node of state tree after transactions are executed and finalizations applied
- $\sigma_{t+1} \equiv \Pi(\sigma_t, B)$

`transactionsRoot`

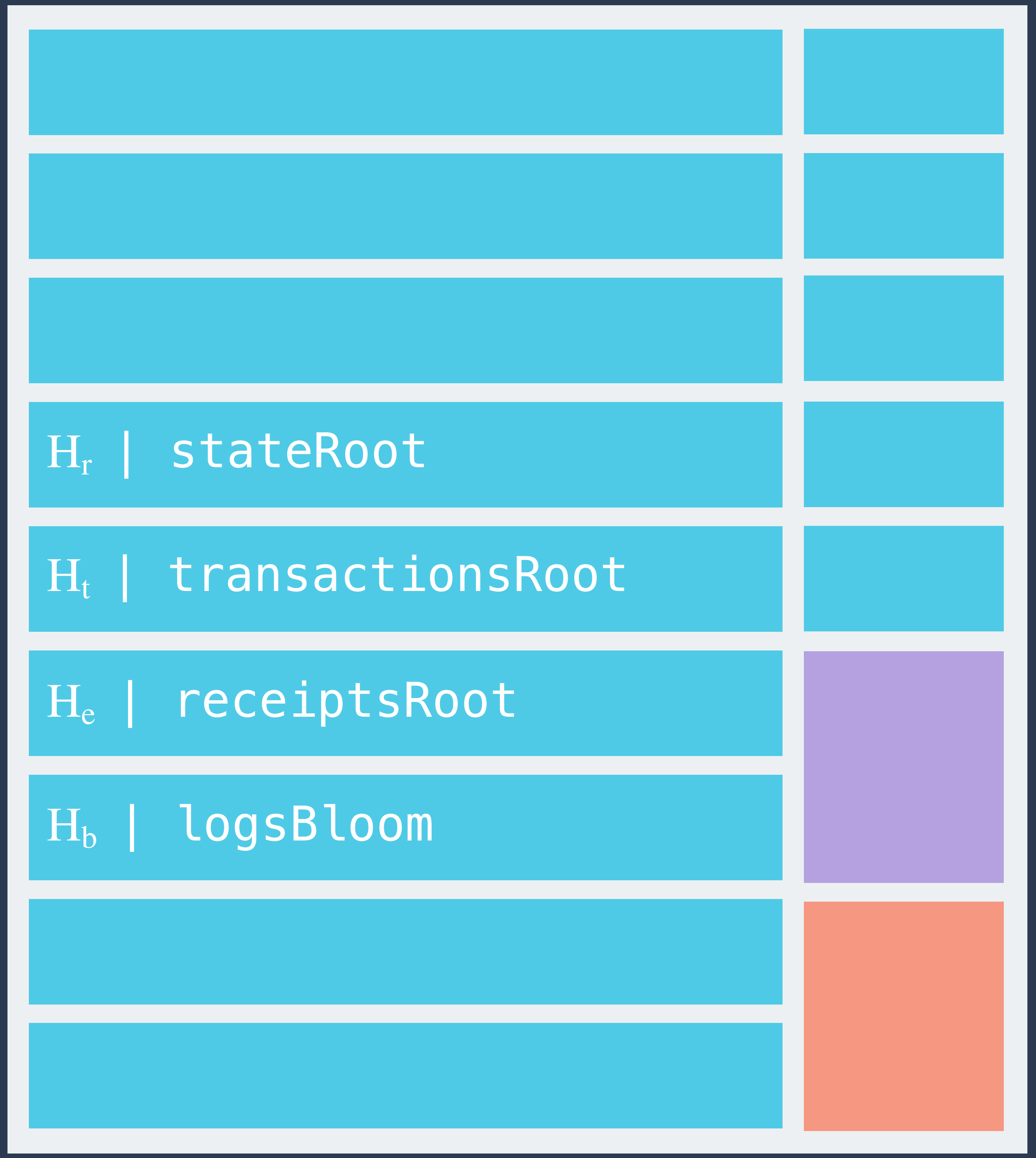
- Hash of root node of transaction tree

`receiptsRoot`

- Hash of root node of receipts tree populated from receipts of each transaction
- $L_R(R) \equiv (\mathrm{TRIE}(L_S(R_\sigma)), R_u, R_b, R_l)$

`logsBloom`

- Bloom filter composed of indexable data in each log entry
- Node can quickly scan block headers check bloom filter if block may contain logs
- Re-executes transactions to generate logs, then returns



Block, State, Transactions: The Block

`gasLimit`

- Max gas all transactions combined are allowed to consume
- Akin to Bitcoin's blocksize (in bytes) but controllable by miner within constraints

`gasUsed`

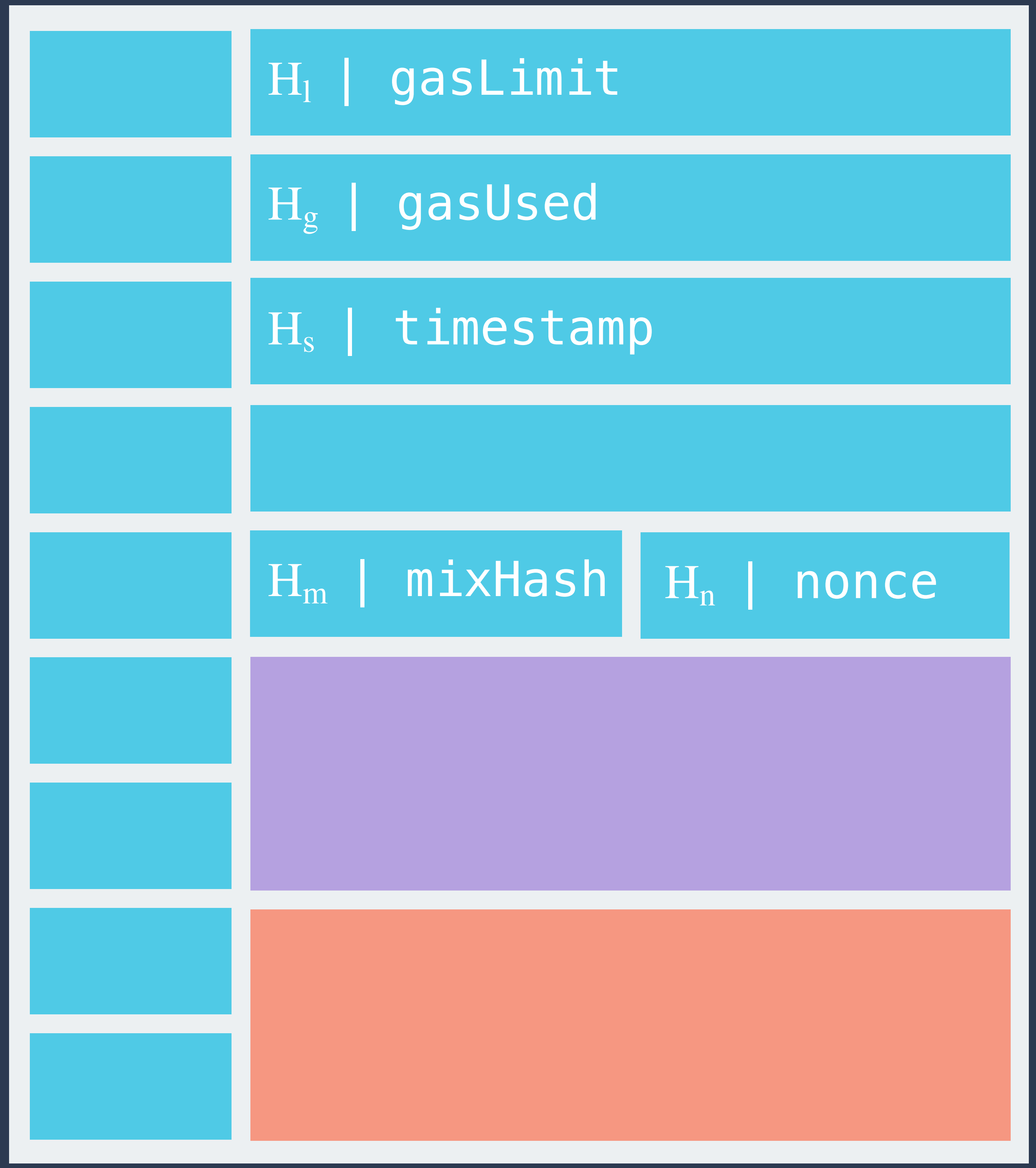
- Total gas used from executing transactions

`timestamp`

- Unix time() at this block's inception

`mixHash, nonce`

- Proof a sufficient amount of work went into mining this block



Gas, Payment

- Ether is used to purchase gas
- Transactors are free to set any **`gasPrice`** and miners are free to ignore any transactions
- **Intrinsic gas**, g_0 (“cost to pay for existence of transaction”)
  - Amount of gas transaction requires to be paid prior to execution
  - Certain cost per operation (e.x. `SHA3(…)` costs 30 gas)
  - Remaining gas used to pay for computation $g \equiv T_g - g_0$
- Upfront transaction cost: **`gasLimit * gasPrice + value`**
- **Unused gas** ($^*g$) is refunded
- In-case of **insufficient funds** during execution, state is reverted, but fees are kept by miner

Execution Model

- Specifies how system state is altered given
  - Bytecode instructions
  - Environment tuple data
- Ethereum Virtual Machine (EVM) - *quasi*-Turing-complete machine
- Stack-based, 256-bit word item size, 1024-item max stack size
- Word-addressable byte array memory model
- Word-addressable word array storage model (non-volatile, stored separately in system state, K/V, σ)
- Fees associated with memory and storage access

Execution Model

Execution model defines,

$$(\sigma', g', A, o) \equiv \Xi(\sigma, g, I)$$

Which can compute the resultant state, σ', remaining gas, g', accrued substate, A, and resultant output, o

Execution Environment

σ	System state
g	Remaining gas for computation
I_a	Code-owning account address
I_o	Transactor (sender) address
I_p	Gas price of transaction
I_d	Input data byte array
I_s	Execution-initiator address (E.x. I_o)
I_v	Value in Wei sent to account
I_b	Machine code to be executed
I_H	Block header of present block
I_e	Depth of present message-call

Execution Model

- Stack items are added/removed
- Single run loop,
  - Check *exception*-halting state at each step (Eq. 128)
  - Program counter increments for each operation
  - Gas is reduced by the instruction’s gas cost
  - Check *normal*-halting state at each step (Eq. 132)
- `JUMPDEST` command allows jumping to arbitrary positions in contract code
- Appendix H.2 for complete instruction set

Block Finalization

Finalizing a block involves,

1. Validate (miners: determine) ommers
   - Valid headers, <= 7 generations
2. Validate (miners: determine) transactions
   - Gas used in block (gasUsed) == accumulated gas used according to final transaction
3. Apply rewards
   - Reward miner 5 ether + 1/32 * 5 ether per ommer block
   - Reward ommer depending on block number
4. Verify (miners: compute valid) state and nonce

Block Finalization - Mining

- The mining PoW exists as a cryptographically-secure nonce proving a particular amount of computation was expended
- Mining new blocks comes with an attached reward to incentivize work, and distribute wealth
- Goals of a PoW function
  - Accessible, not requiring specialized hardware
  - Prevent super-linear profits, and gaining large % of mining power
  - ASIC-resistant

Block Finalization - Ethash

- Ethash is the PoW algorithm for Ethereum
- Algorithm,
  1. Compute a seed by scanning through block headers up until current block
  2. Compute 16MB pseudorandom cache (stored by light clients)
  3. Generate 1 GB dataset (DAG) from cache, where dataset item depends on small number of cache items - updated every 30k blocks ~ 100Hrs - grows linearly with time
  4. Mining involves hashing together random slices of the dataset - memory hard task

Applications

- Financial applications
  - sub-currencies, financial derivatives, hedging contracts, wallets, wills, employment contracts, insurance
- Online voting, gambling, decentralized governance
- Decentralized storage, cloud computing
- "Decentralized Autonomous Organizations"
- Smart property
- Dapps (e.x. Golem, Augur)
- ICOs

Applications - Developer Tools

- **Writing smart contracts (languages)**
  - Solidity
  - Serpent
  - LLL (Low-level Lisp-like Language)
- **Clients**
  - Geth
  - testRPC
- **Frameworks**
  - Truffle
  - OpenZepplin
- **Dapps**
  - Web3.js
  - MetaMask
  - Mist

Solidity Example - MetaCoin.sol

```
pragma solidity ^0.4.4;

import “./ConvertLib.sol";

contract MetaCoin {
   mapping (address => uint) balances;

   event Transfer(address indexed _from, address indexed _to, uint256 _value);

   function MetaCoin() {
      balances[tx.origin] = 10000;
   }

   function sendCoin(address receiver, uint amount) returns(bool sufficient) {
      if (balances[msg.sender] < amount) return false;
      balances[msg.sender] -= amount;
      balances[receiver] += amount;
      Transfer(msg.sender, receiver, amount);
      return true;
   }

   function getBalanceInEth(address addr) returns(uint){
      return ConvertLib.convert(getBalance(addr), 2);
   }

   function getBalance(address addr) returns(uint) {
      return balances[addr];
   }
}
```