STUDIO SU SISTEMI DISTRIBUITI E DECENTRALIZZATI BASATI SU BLOCKCHAIN

Cos’è la blockchain ?

La blockchain è un database immutabile, decentralizzato, protetto da crittografia e accettato tramite consenso da una rete di peer.

Scripting

Bitcoin

P2PKH
+
P2SH

Ethereum

EVM assembly
+
Account
+
SmartContract

Vecchio-Nuovo paradigma

- **One-point-of-failure**
- **Trusted-third-party**
- **Censura delle opere**
- **Centralizzazione dell’auth**
- **Enti centralizzati**

- **Decentralizzato**
- **Ogni utente è responsabile per l’auth**
- **Enti autonomi e trasparenti**

Digichain

- **La gestione dei diritti sulle opere autoriali**
- **Cosa permette:**
  - Registrare un opera
  - Guadagnare dalla condivisione
  - Gestisce il trasferimento dei diritti
  - Ognuno può osservare gli stati delle compravendite
  - Confrontare lo sfruttamento delle opere

Registrazione

- **La prova del diritto acquisito**
- **Gestisce:**
  - Il diritto
  - Trattiene i fondi investiti
  - Gestisce eventuali refund
  - Può essere gestito solo tramite Digichain

GlobalRegistrar

- **ENS Ethereum Name System**
- **Permette:**
  - La registrazione di smartcontract sotto nome simbolico
  - Permette la consultazione
  - Facilita l’utilizzo di contratti
  - Può implementare forme di bidding

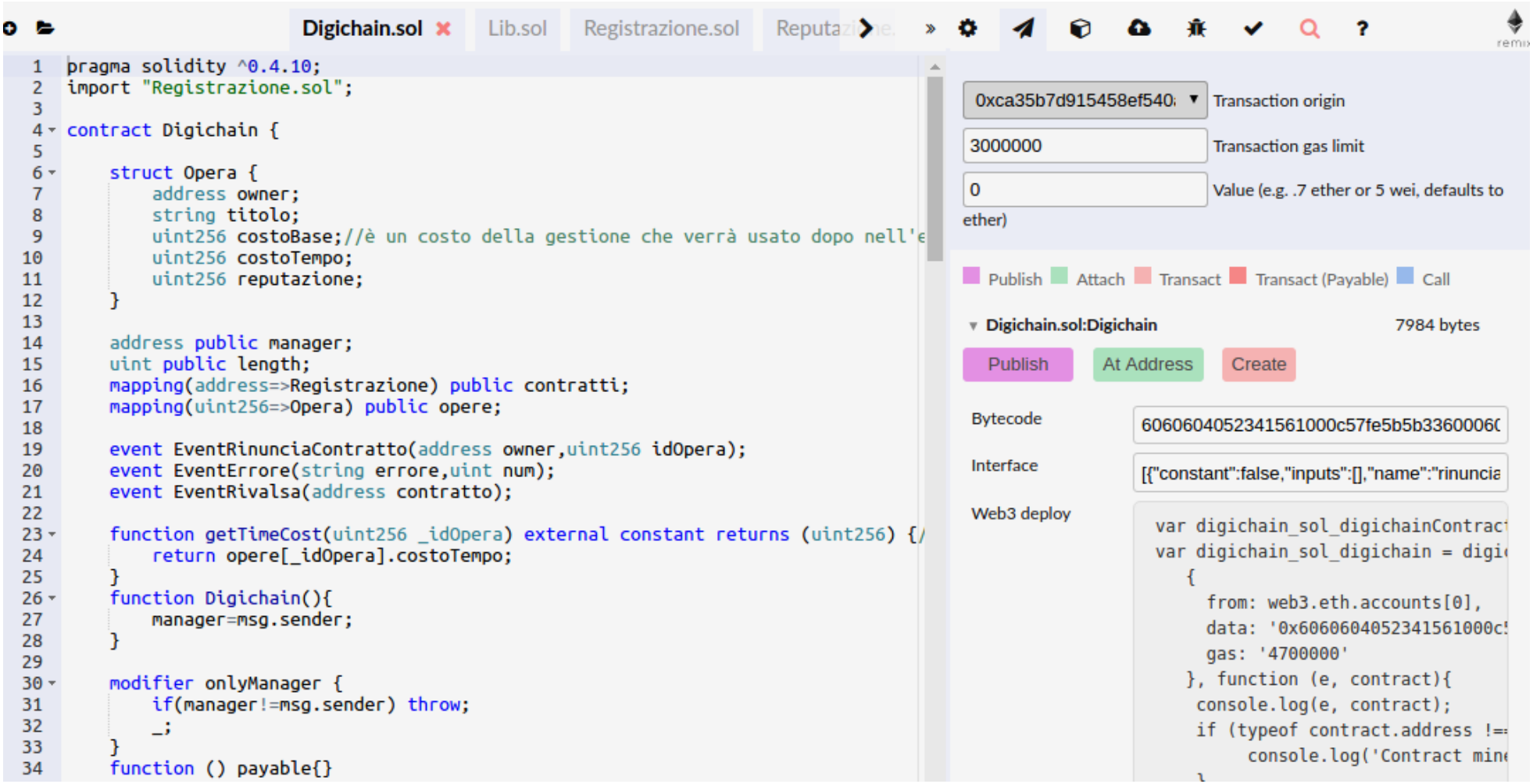
Ambiente di sviluppo

Solidity

Ethereum

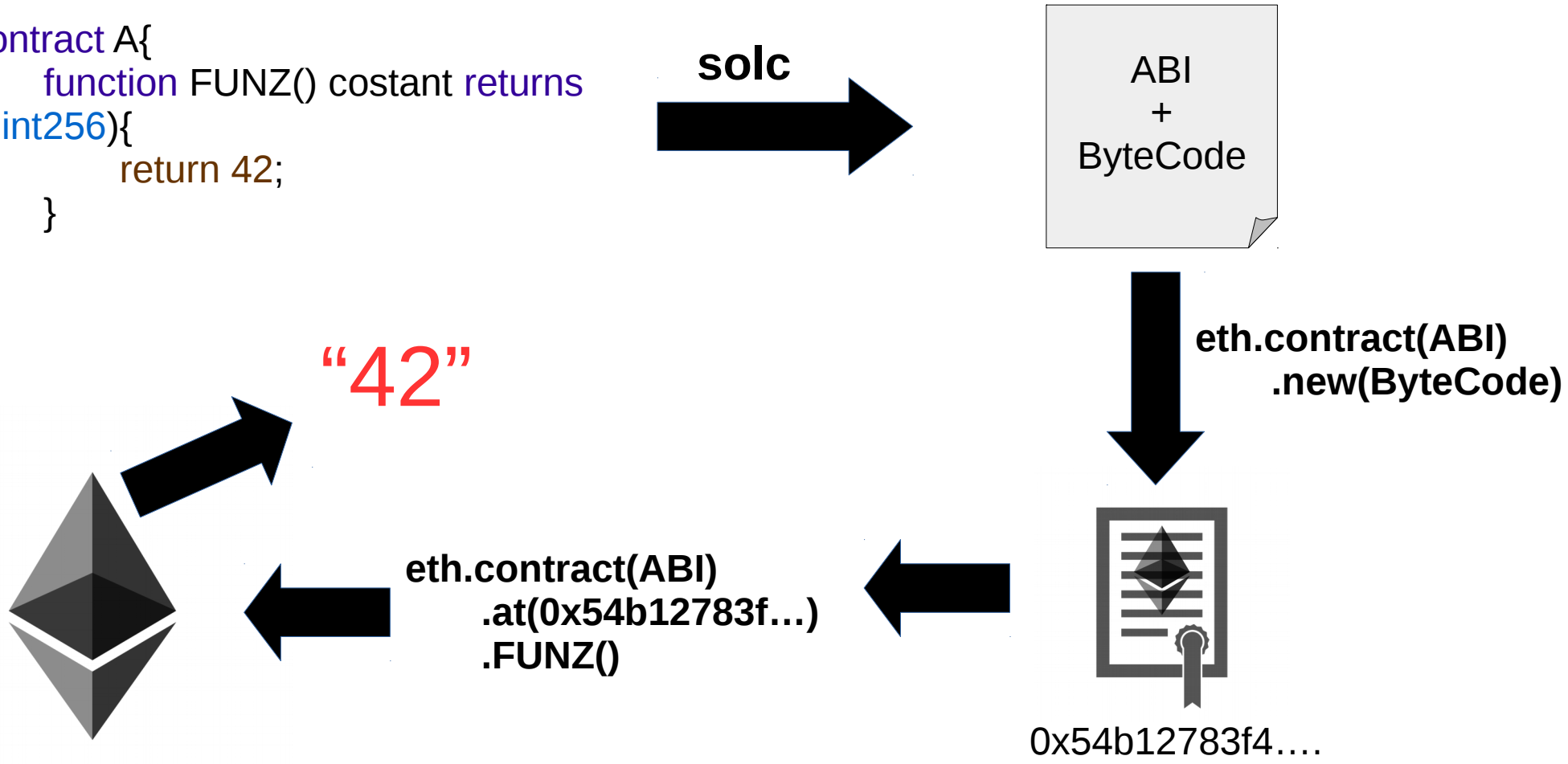
Geth + Web3 + Solc + Mocha

IDE – Solidity



Geth (Command line interface)

```
contract A{
	function FUNZ() costant returns
(uint256){
		return 42;
	}
}
```



Che problemi ha il paradigma di Ethereum

- **Numeri random**
- **Programmazione ad eventi**
- **Scalabilità**
- **Ogni operazione ha un costo**
- **Tempi != numero_blocco**
- **Immutabilità del codice**

Che problemi sono stati risolti

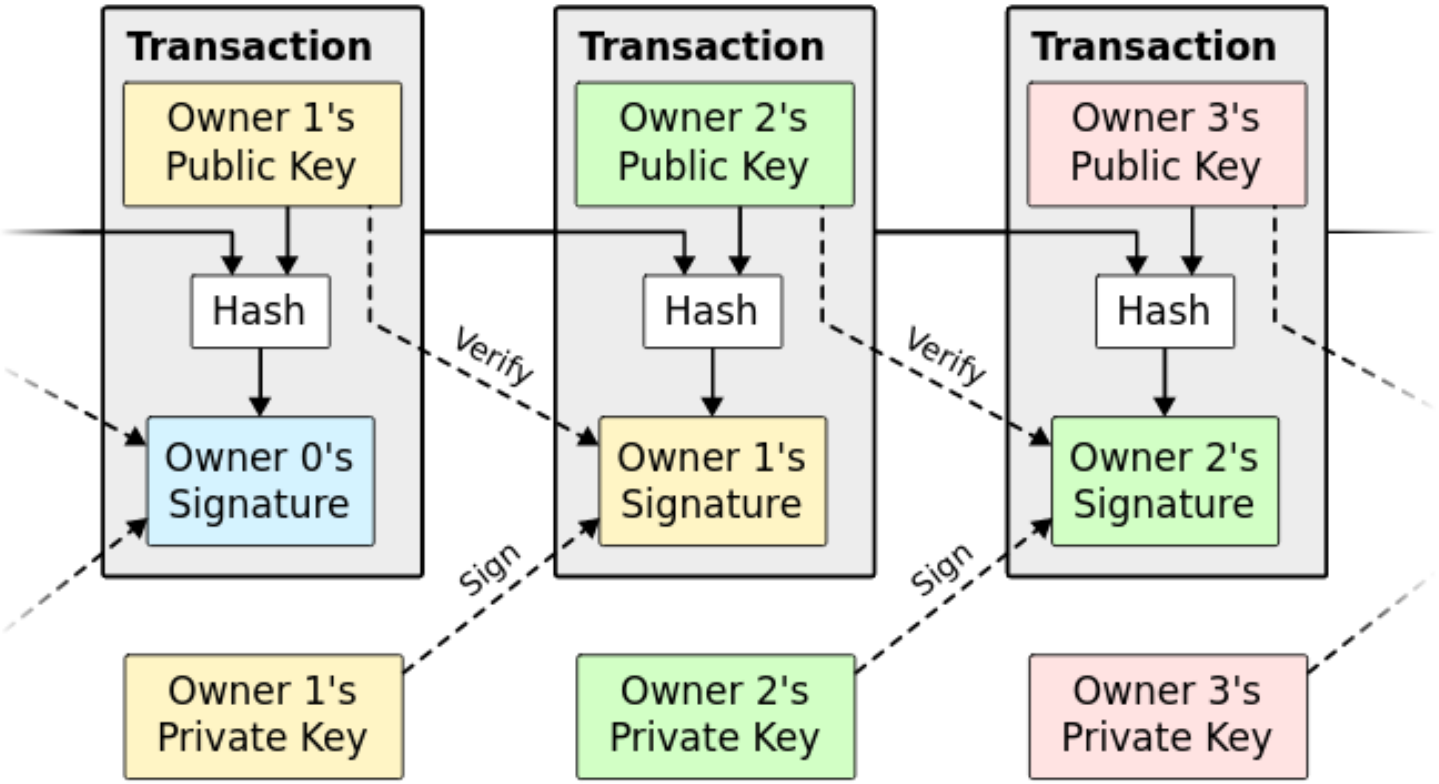
- **Pattern**
  - System of contracts
  - Escrow
- **Costo dell’utilizzo a tempo**
  - Gestione del refund
- **Incentivazione dei nodi ad automatizzare le funzioni.**
  - Nessuna garanzia
  - Partecipazione volontaria come la Blockchain

Possibili sviluppi

- **Interfaccia web**
- **Integrazione con browser**
- **Votazioni tracciabili**

EOF

Transazioni e la base della criptomoneta



Ethereum Blockchain Mechanism (Proof Of Work)

An interpretation of the Ethereum Project Yellow Paper

G. Wood, "Ethereum: A secure decentralised generalised transaction ledger", 2014.

Lee Thomas

2016-06-21

***Ver 0.1** 2016-06-22*

Successful miner's computer

Takes the following steps and broadcasts block header, H, to network

Determine Transactions

Miner picks transactions to process (from those broadcasted)

Determine Ommers

Miner finds and includes valid ommers

Apply Rewards

Update account balance(s) to reward valid blocks

Compute a Valid State

Block Finalisation Defines result of all selected state transitions

State Transition Cycle Defines result of a single transaction $\sigma_{t+1} = \Upsilon(\sigma_t, T)$

Execution Cycle Defines result of a single cycle of the state machine

`Ethereum Virtual Machine, EVM`

Instruction Set

```
0x00 STOP
0X01 ADD
0x02 MUL
0X03 SUB
0x04 DIV
...
```

Execution Environment, I Tuple

- **Code owner, I_a** Address of accounts that owns executing code
- **sender, I_o** Sender address of transaction that originated this execution
- **Gas price, I_p** Price of gas in the transaction that originated this execution
- **Input data, I_d** Byte array (transaction data if execution agent is transaction)
- **causer, I_s** Address of account that caused the code to be executing (I_o if transaction)
- **value, I_v** Wei pass to this account as part of execution procedure (transaction value if execution agent is transaction)
- **Machine code, I_b** Byte array of machine code to be executed
- **Block header, I_H** Block header of the present block
- **Message-call depth, I_e** Number of CALLS or CREATES being executed at present

Substate, A Tuple

- **Suicide Set, A_s** Accounts to be discarded post transaction
- **Log Series, A_l** A series of archived indexable checkpoints (allow contract calls to be easily tracked externally)
- **Refund Balance, A_r** Price of gas in the transaction that originated this execution

Tuple

World State, σ

- **Addresses** scalar; the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account.
- **Account States** RLP serialised data structures
- **Balances** scalar; the number of Wei owned by this address
- **Storage and Code**

Machine state, μ Tuple

- **Gas available, g**
- **Program counter, pc** 256 max
- **Memory contents, m**
- **No words, i** Active number of words in memory, counting from position 0
- **Stack contents, s**

Iterator Function, O Defines result of a single cycle of the state machine

- Get next instruction from I_b
- Get items to add to /remove from stack, $\Delta = \alpha + \delta$
- Update machine stack
- Subtract gas used
- Increment Program counter

Halt? — Yes

All transactions ? — No / Yes

Get next Transaction

Proof of Work

$\text{PoW}(H_n, \mathbf{n}_{rand}, \mathbf{d}) \leq 2^{256}/H_d$?

No

Choose Random Nonce $\mathbf{n}_{rand}$ 8 bytes

Mining Network

"Oh look, some transactions have been broadcast to the network. Let's race each other to create a new valid block... GO"

Ommers (or Uncles)

Valid, yet redundant blocks where $\text{PoW}(H_n, \mathbf{n}_{rand}, \mathbf{d}) < 2^{256}/H_d$

time

Ethereum Network

"Oh look another miner has been successful. Therefore there is little point in continuing with mining this block – we'll start on the next one"

"note – we had to run all the same code on our copies of the EVM to prove this!"

Red indicates `KECCAK256` Hash

Block Header, H

- **parentHash, H_p** Keccak256 hash of parent block header
- **ommersHash, H_o** Keccak256 hash of the ommers list portion of this block
- **beneficiary, H_c** 160-bit address, for fees collected from successful mining
- **stateRoot, H_r** Keccak 256-bit hash of the root node of the state trie, after all transactions are executed and finalisations applied
- **transactionsRoot, H_t** Keccak 256-bit hash of the root node of the trie structure populated with each transaction in the transactions list
- **receiptsRoot, H_e** Keccak 256-bit hash of the root node of the trie structure populated with the receipts of each transaction in the transactions list
- **logsBloom, H_b** Bloom filter from indexable info (logger address & log topics) contained in each log entry from the receipt of each tx in the tx list
- **difficulty, H_d** Scalar value corresponding to the difficulty level of this block
- **number, H_i** scalar value equal to the number of ancestor blocks (genesis block = 0)
- **gasLimit, H_l** scalar value equal to the current limit of gas expenditure per block
- **gasUsed, H_g** scalar value equal to the total gas used in transactions in this block
- **timestamp, H_s** scalar value equal to the reasonable output of Unix's time() at this block's inception
- **extraData, H_x** arbitrary byte array (<32 bytes) containing data relevant to this block
- **mixHash, H_m** 256-bit hash which proves combined with the nonce that a sufficient amount of computation has been carried out on this block
- **nonce, H_n** 64-bit hash which proves combined with the mixHash that a sufficient amount of computation has been carried out on this block

Block, B

- **Block Header, H or B_H**
- **Transaction List, B_T**
- **Ommers List, B_U**

Information required to derive Block Header

Account storage contents Trie A mapping between integer keys (KEC) and integer values (RLP)

Root Node

Account, σ[address] RLP data structure

- **nonce, σ[address]$_n$** scalar; the number of transactions sent from this address or, in the case of accounts with associated code, the number of contract-creations made by this account.
- **balance, σ[address]$_b$** scalar; the number of Wei owned by this address
- **storageRoot, σ[address]$_s$** 256-bit hash of the root node of a Merkle Patricia tree that encodes the storage contents of the account (a mapping between 256-bit integer values), encoded into the trie as a mapping from the Keccak 256-bit hash of the 256-bit integer keys to the RLP-encoded 256-bit integer values.
- **codeHash, σ[address]$_c$** Hash of the EVM code of this account – the code that gets executed should this address receive a message call; it is immutable and thus, unlike all other fields, cannot be changed after construction. All such code fragments are contained in the state database under their corresponding hashes for later retrieval.

World State Trie, σ A mapping between addresses and account states. Stored as a merkle-partrica tree

Root Node

Transaction, T

- **nonce, T_n** Scalar; the number of transactions sent by the sender
- **gasPrice, T_p** scalar; the number of Wei to be paid per unit of gas for all computation costs incurred as a result of execution of this transaction
- **gasLimit, T_g** scalar; the max amount of gas that should be used in executing this transaction. Paid before any computation is done, may not be increased later
- **to, T_t** 160-bit address of the message call's recipient or, for a contract creation transaction, ∅ (zero bytes).
- **value, T_v** scalar; the number of Wei to be transferred to the message call's recipient or, in the case of contract creation, as an endowment to the newly created account
- **init, T_i** Contract creation transaction only: unlimited size byte array specifying the EVM-code for the account initialisation
- **data, T_d** Message-call transaction only: unlimited size byte array specifying the input data of the message call
- **v, r, s, T_w, T_r, T_s** Outputs from ECDSA signature of $\text{KEC}(T_n, T_p, T_g, T_t, T_v, T_i \text{ or } T_d)$ & identifies sender.

Transaction Trie, T A merkle-partrica tree of transactions to include

Root Node

Transaction Receipts Trie Index keyed trie

Root Node

Transaction Receipt, $B_R[i]$ *(i=transaction no)* Tuple

- **post-transaction state, R_σ** Tree
- **cumulative gas used, R_u** positive integer
- **transaction logs, R_l** Series of log entries (tuples).
- **bloom filter of log info, R_b** 256-byte hash. Reduction of a log entry

Log Entry, O Tuple

- **Logger's address, O_a** address
- **Log topics, O_t** A series of 32-byte log topics (O_{t0}, O_{t1},...)
- **Log data, O_d** Some number of bytes data.

Transazioni e la base della criptomoneta

```
contract A{
	function FUNZ() costant returns
(uint256){
		return 42;
	}
}
```

```
[{"constant":false
,"inputs":[]
,"name":"FUNZ"
,"outputs":[{"name":""
,"type":"uint256"}]
,"payable":false
,"type":"function"}]
```