ETHEREUM DEV BARCELONA Meetup

µtraining 4 devs

adria@codecontext.io

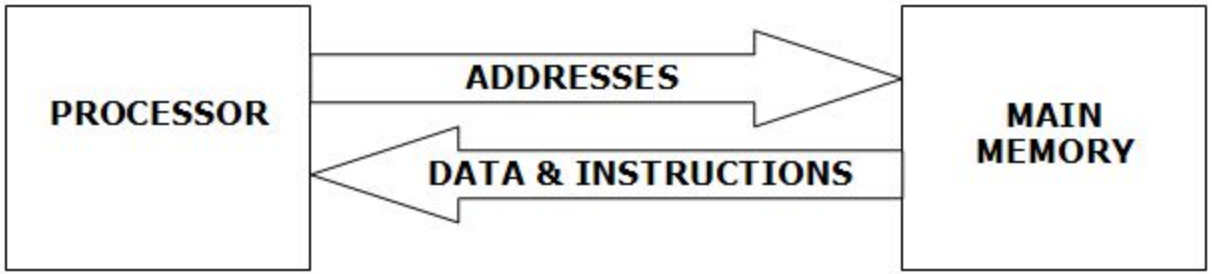
PART 0

quick intro

PART I

ethereum

- Ethereum is a blockchain technology that
  - Has an embeeded cryptocurrency: **ETHER** (finney, swabo,... wei)
  - You can hold ethers with a wallet with a private key, like bitcoin
  - Ether accounts are called **ADDRESS** (like Bank IBAN)
  - And transfer them from an account to another
  - In some implementations (nets) has a market value
- But you also can:
  - Store programs (called **SMARTCONTRACTS**) in the blockchain
  - Call to functions of this programs
  - You need to pay for it, and you pay with Ether
  - The “CPU consume” of a function is calculated with **GAS**, eg
    - i=i+2 costs 3 GAS
    - i=i*2 costs 5 GAS
  - Call to functions of this programs
    - Using your account that has positive ETH balance
    - Using MAXGAS=200 GASPRICE=0.0002 ETHS
    - The function consumed 100 GAS, you pay 100*0.0002=0.02ETHS = 0,24€
  - An SMARTCONTRACT is also an accout that can hold ethers, so is also
    - identified by an ADDRESS
  - An SMARTCONTRACT can make automatic transfers between accounts


PROCESSOR
ADDRESSES
DATA & INSTRUCTIONS
MAIN
MEMORY

```
int i=1
func inc() {
  i++;
}
func get() {
  return i;
}
```

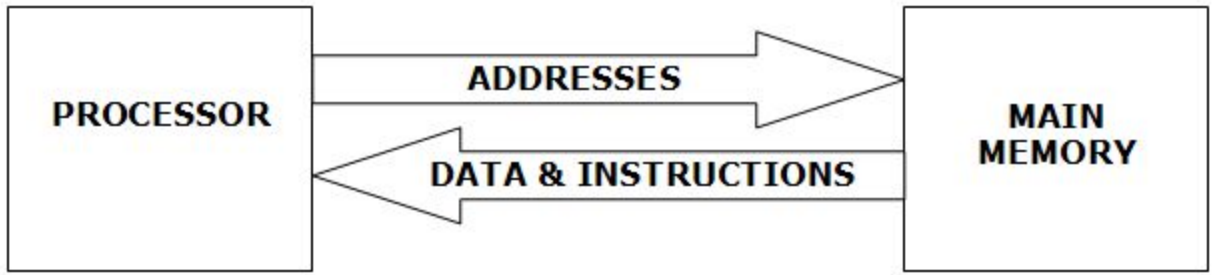
To execute this program:

- Bootstrap
  - Need to compile this code into CPU instruction set
  - CPU copy [allocates] to memory the compiled code
  - CPU copy [allocates] to memory the ‘1’ for the ‘i’ variable
    the default values
- User calls the method inc()
  - CPU retrieves the code from the memory, and run the code
    - CPU gets the value from the memory of ‘i’ variable
    - CPU increments the value
    - CPU stores the value to the memory of ‘i’ variable

Memory is the blockchain

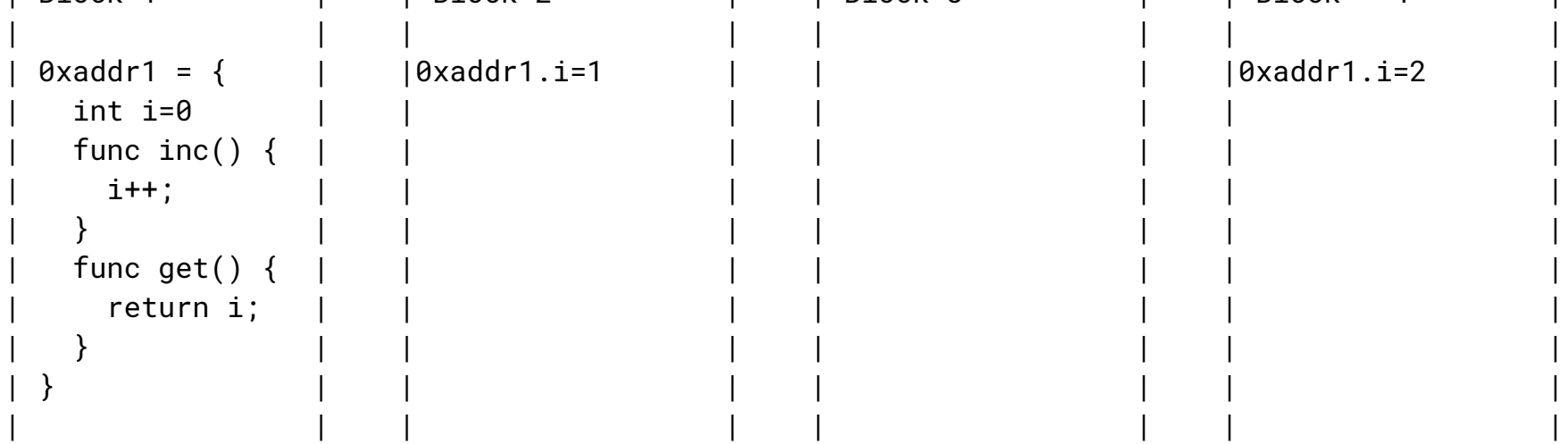
- It’s a secure, append-only database maintained by peers (like log file)
- Append operations are done each 15 seconds, so writes are grouped with blocks

Blockchain & Von Neumann

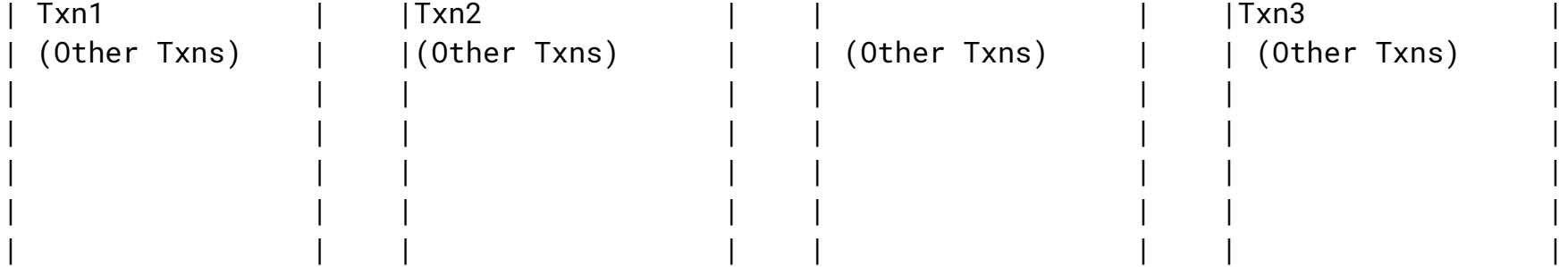


All nodes executes
the same
EVM = Etehreum
Virtual
Machine

```
create_contract("int i=0;func inc(){i++;}func get(){return i;}") -> 0xaddr1
0xaddr1.inc()
println 0xaddr1.get()
0xaddr1.inc()
```



```
Txn1: create_contract("int i=0;func inc(){i++;}func get(){return i;}") -> 0xaddr1
Txn2: addr1.inc()
println 0xaddr1.get()
Txn3: 0xaddr1.inc()
```



Tx content

Changes to blockchain are done with TRANSACTIONS, that contains:

1) Source account
2) Target account
3) Ethers sent from an account to another account
4) Data
5) The signature of source account

	Src	Targ	Eth	Data	Sig
Transfer ethers between accounts	0x000001	0x000002	1		….
Create contract	0x000001			0a 0f 1e 0d...	...
Call function	0x000001	contract address (0xaddr1)		name function+parameters	...

PART II

let’s play

- Install metamask google chrome plugin
- Create an account
- Get some ETH from http://faucet.ropsten.be:3001/
- Go to https://ethereum.github.io/browser-solidity
- Drop

```
pragma solidity ^0.4.9;
contract Counter {
     uint i=1;
     function inc() {
        i++;
     }
     function get() constant returns (uint) {
          return i;
     }
}
```

- Create a contract, play
- Attach to an existing contract, play
- Let’s see the transactons in https://testnet.etherscan.io/

PART III

solidity

```
bool              | true, false
uint              | alias for uint256
int               | alias for int256
address           |
string            | "ETHS GO!"
enum              | enum State { Open, Closed } ; State u = State.Open;
struct            | struct Point { uint x, uint y}; Point p1,p2;
mapping           | mapping (string => int) ages; ages["juan"]=12; ages["pepe"]=60
vector            | int[] v; v.push(1); v[0]
var               | any variable

All composite types (struct, mapping, vector) are references, so data will be not copied
Not needed to initialize, default values are applied

Point p1; // p1.x ==0  && p1.y==0
p.1x = 1
Point p2 = p1;
if (p1.y==1) {} // true, p1 and p2 points to the same memory
var p3 = p1;
```

Base code

```
pragma solidity ^0.4.9;
contract Counter {
  uint i;

  function get() returns (uint) { return i; }
  function reset() {
     i = 0;
  }
  function inc() {
     if (i==5) {
        i=0;
    } else { i++; }
  }
}
```

constructor

```
pragma solidity ^0.4.9;
contract Counter {
  uint i;
  string name;

  function Counter(string _name)          { name = _name; }
  function get() returns (uint)           { return i; }
  function reset() {
     i = 0;
  }
  function inc() {
     if (i==5) {
        i=0;
    } else { i++; }
  }
}
```

```
pragma solidity ^0.4.9;
contract Counter {
  uint i;
  string name;
  address owner;

  function Counter(string _name)          { name = _name; owner = msg.sender; }
  function get() returns (uint)           { return i; }
  function reset() {
     if (msg.sender != owner) throw;
     i = 0;
  }
  function inc() {
     if (i==5) {
        i=0;
    } else { i++; }
  }
}
```

```
pragma solidity ^0.4.9;
contract Counter {
  uint i;
  string name;
  address owner;

  function Counter(string _name)          { name = _name; owner = msg.sender; }
  function get() constant returns (uint) { return i; }
  function reset() {
     if (msg.sender != owner) throw;
     i = 0;
  }
  function inc() {
     if (i==5) {
        i=0;
    } else { i++; }
  }
}
```

payable, <address.send>

```
pragma solidity ^0.4.9;
contract Counter {
  uint i;
  string name;
  address owner;

  function Counter(string _name)         { name = _name; owner = msg.sender; }
  function get() constant returns (uint) { return i; }
  function reset() {
     if (msg.sender != owner) throw;
     owner.send(this.balance);
     i = 0;
  }
  function inc() payable {
     if (i==5) {
        msg.sender.send(this.balance);
        i=0;
    } else { i++; }
  }
}
```

```
pragma solidity ^0.4.9;
contract Counter {
  uint i;
  string name;
  address owner;

  event LogPrize(indexed address _addr, uint256 _amount);
  function Counter(string _name)         { name = _name; owner = msg.sender; }
  function get() constant returns (uint) { return i; }
  function reset() {
     if (msg.sender != owner) throw;
     owner.send(this.balance);
     i = 0;
  }
  function inc() payable {
     if (i==5) {
        msg.sender.send(this.balance);
        LogPrize(msg.sender, this.balance);
        i=0;
    } else { i++; }
  }
}
```

```
pragma solidity ^0.4.9;
contract Counter {
  uint i;
  string name;
  address owner;

  event LogPrize(address _addr, uint256 _amount);
  function Counter(string _name)         { name = _name; owner = msg.sender; }
  function get() constant returns (uint) { return i; }
  function reset() {
     if (msg.sender != owner) throw;
     owner.send(this.balance);
     i = 0;
  }
  function inc() payable {
     if (i==5) {
        msg.sender.send(this.balance);
        LogPrize(msg.sender, this.balance);
        i=0;
    } else { i++; }
  }
  function () payable { msg.sender.send(msg.value}; }
}
```

Small coin example

```
pragma solidity ^0.4.9;
contract MiniCoin {

  mapping (address => uint256) balances;

  function MiniCoin(uint256 _initialBalance) {
    balances[msg.sender]=_initialBalance;
  }

  function transfer(address _to, uint256 _amount) {
      if (balances[msg.sender] <  _amount ) throw;
      balances[msg.sender] -= _amount;
      balances[_to] += _amount;
  }

  function balance() constant returns (uint256) {
      return balances[msg.sender];
  }
}
```

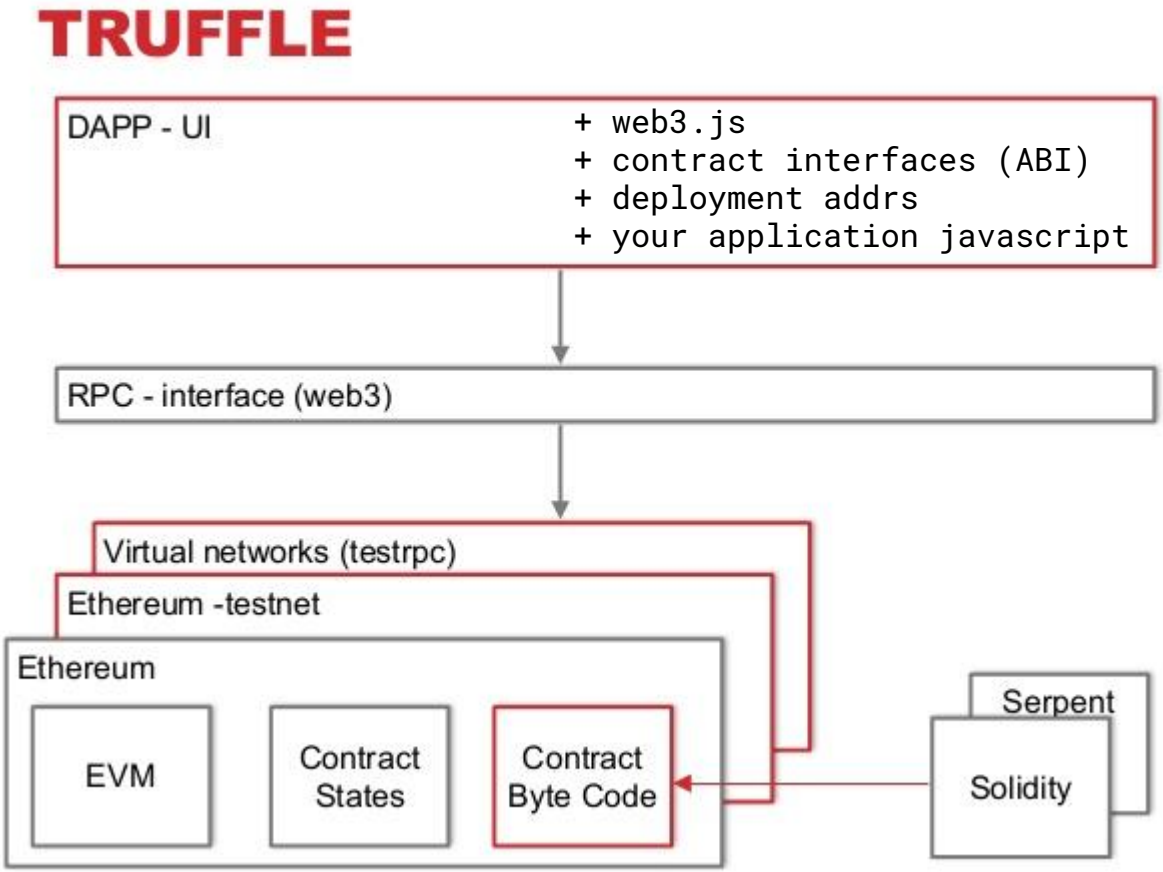
Small smartcontract

- Make an exercise together
- I lend you a tool (e.g. Hammer)
  - but I want you to make a security deposit of at least 1000 wei
- When returned
  - if both agree, the deposit is sent to me (if broken) or to you (if ok)
  - if we don’t agree a neighbor will act as judge

PART IV

truffle 3.x

Truffle schema



```
> npm install -g truffle
> npm install -g ethereumjs-testrpc
> npm install truffle-default-builder --save
```

let’s see the contents inside

```
> truffle init
> truffle compile
> truffle migrate --reset
```

let’s see the Lend example in truffle

```
> truffle compile
> truffle migrate --reset
> truffle serve
```