Code d'Armor

Đapps

on Ethereum with Solidity

Version 2 - 20/06/2018

Code d'Armor

Pierre-Yves Lapersonne

software developer

pylapp.github.io

He has a beard, so he is an expert!

Code d'Armor

a story of Đaleks

Code d'Armor

back to the past

back to the past...

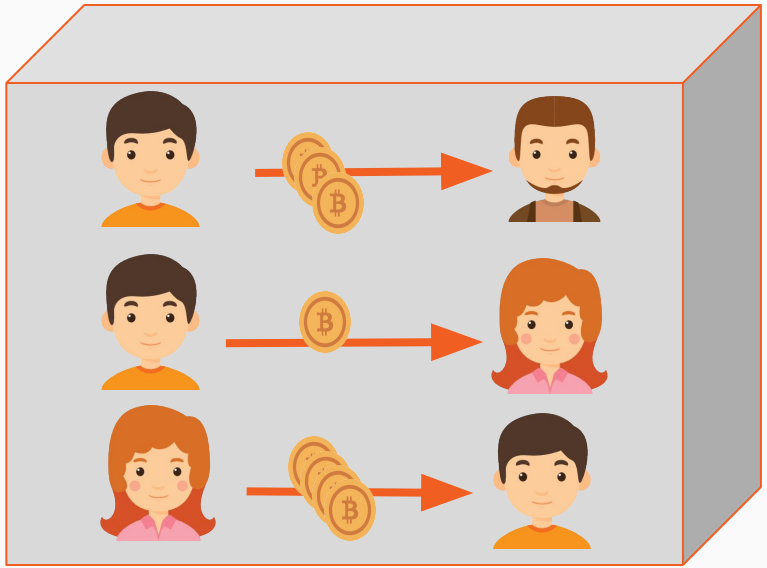
- 2008 - subprimes crisis
- 2008 - Bitcoin 
  - **Bitcoin: A Peer to Peer Electronic Cash System**
  - Satoshi Nakamoto
  - 2009/01/03 - Block Genesis
- Bitcoin: the 1st blockchain
  - **decentralized**
  - **without regulation**
  - **unfalsifiable**
  - not anonymous
  - holds only bitcoin transactions

1 - transaction

- sender
- receiver
- balance of bitcoin

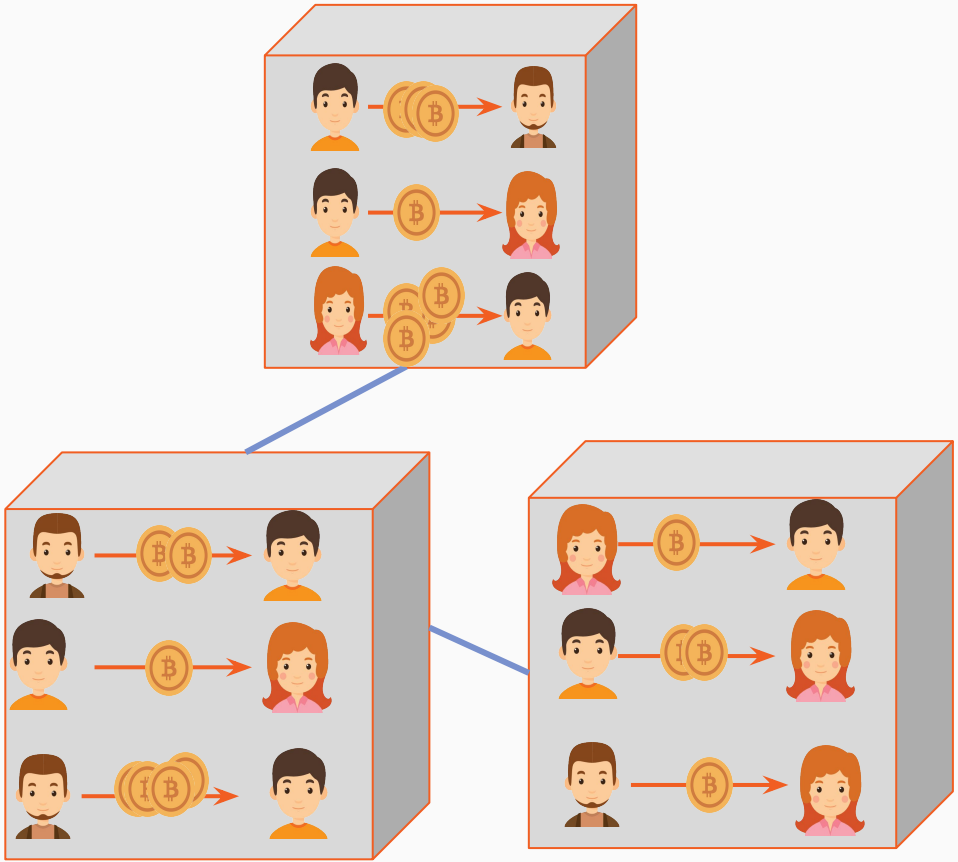
2 - block mining

- gather several transactions in a block
- miners solve a hard mathematical problem to choose the one who will add the block and trigger the transactions inside (Proof Of Work)



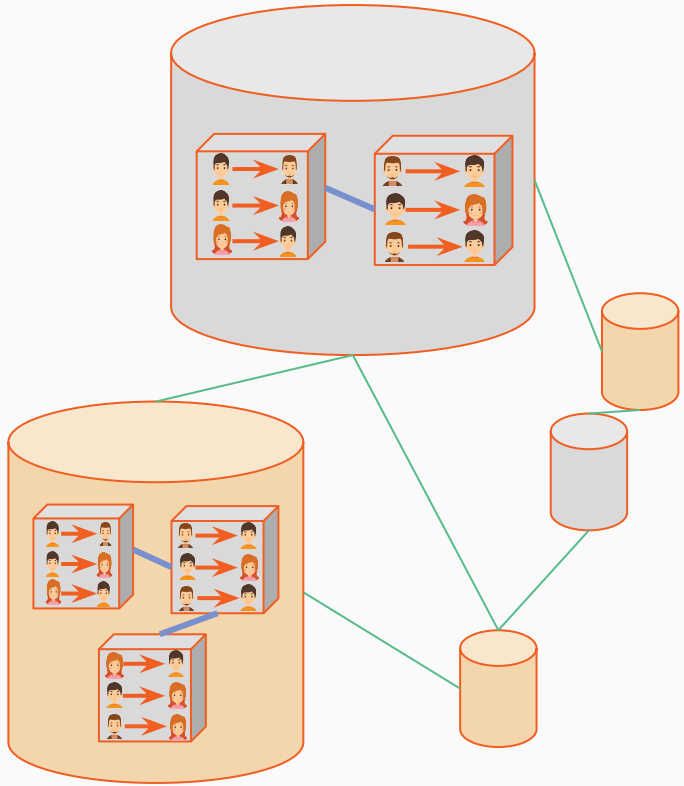
3 - blockchain

- each block has a fingerprint of the previous block (hash)
- to corrupt a block, we must modify all the following blocks



4 - network

- the blockchain is replicated in each node of the network
- if we want to corrupt a blockchain, we have to corrupt all blockchains before the next block adding



Code d'Armor

Ethereum blockchain

a new type of blockchain

- 2013 - Vitalik Buterin
  - was only 19 y.o.!
- blockchains are:
  - **decentralized**
  - replicated (distributed)
  - **not regulated**
  - **use tokens**
  - have blocks with financial transactions
- and what if we put programs in blockchains?
  - **Đ**ecentralized **app**lications

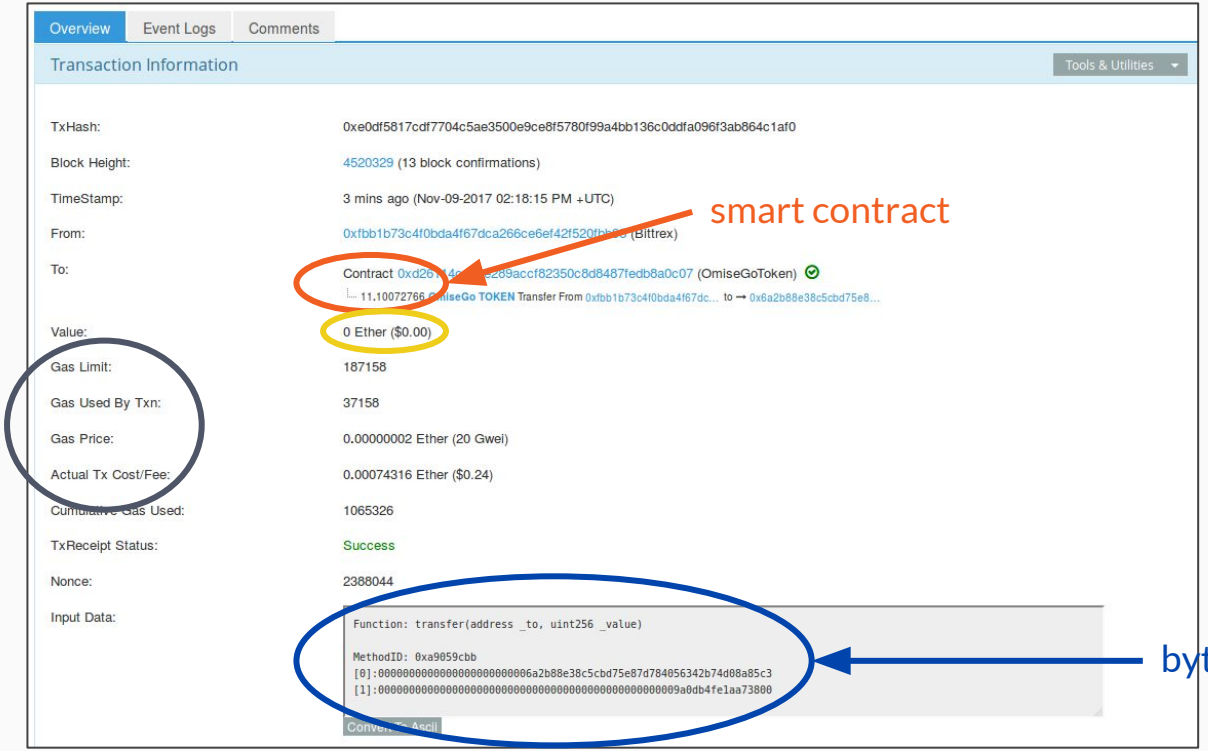
The Ethereum Experience

programs in blockchain

- use of virtual machine
  - Ethereum Virtual Machine (**EVM**)
- transactions have the bytecode in payload
- execution of programs are conditioned
  - **tokens** are used to process instructions
    - ETH, ETC, ...
  - each instruction has a cost
    - **gas**
- nodes of network check outputs of programs
- Ethereum can be seen as kind of **“slow” database/register**

- Ethereum Yellow Paper
- Gas Costs from Yellow Paper

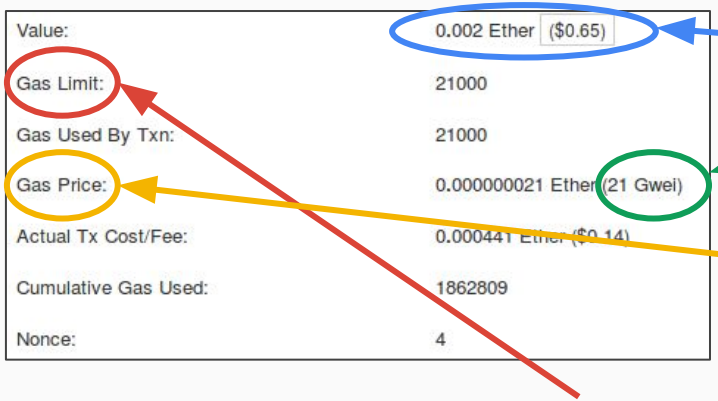
transactions



etherscan.io

einen Kraftstoff wie Benzin 🎶

- **gas** is used so as to evaluate:
  - costs of instructions
  - fees of transactions (**TxFees**)
  - value of transactions earned by miners



final cost of the transaction, earned by miner

wei = unit for gas

- the lower the price is, the less gainful the transaction is, the later the transaction will be mined
- allow to involve the cost of the transaction

the amount of gas to burn to process the program of the smart contract, which depends to the amount of code to process

TxFees in Ether based on gas

Ether = Tx Fees = Gas Limit * Gas Price

```
var unitMap = {
    'wei':          '1',
    'kwei':         '1000',
    'ada':          '1000',
    'femtoether':   '1000',
    'mwei':         '1000000',
    'babbage':      '1000000',
    'picoether':    '1000000',
    'gwei':         '1000000000',
    'shannon':      '1000000000',
    'nanoether':    '1000000000',
    'nano':         '1000000000',
    'szabo':        '1000000000000',
    'microether':   '1000000000000',
    'micro':        '1000000000000',
    'finney':       '1000000000000000',
    'milliether':    '1000000000000000',
    'milli':         '1000000000000000',
    'ether':        '1000000000000000000',
    'kether':       '1000000000000000000000',
    'grand':        '1000000000000000000000',
    'einstein':     '1000000000000000000000',
    'mether':       '1000000000000000000000000',
    'gether':       '1000000000000000000000000000',
    'tether':       '1000000000000000000000000000000'
};
```

ETH Gas Station

Đapps and gas

- Đapps must be **optimized**
  - choose the best functions
  - be aware with storage of data
  - use the more suitable types
  - decrease complexity of functions

- If the program is:
  - dirty
  - heavy
  - not enough well written

  ➔ may **burn a lot of gas**
  ➔ must **be expensive in Ether**

fungible tokens: ERC20

- implemented by a lot of **cryptocurrencies**
- Ether ---> ERC20
- looks like **common currencies**
- e.g. we can burn 0.001 ETH

```
contract ERC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);
    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```


non-fungible tokens: ERC721

- more for **Đapps**
- CryptoKitties ---> ERC721
- looks like a **token**
- e.g. we cannot burn 0.001 CryptoKitty

```
contract ERC721 {
    event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);
    event Approval(address indexed _owner, address indexed _approved, uint256 indexed _tokenId);
    event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);
    function balanceOf(address _owner) external view returns (uint256);
    function ownerOf(uint256 _tokenId) external view returns (address);
    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable;
    function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;
    function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
    function approve(address _approved, uint256 _tokenId) external payable;
    function setApprovalForAll(address _operator, bool _approved) external;
    function getApproved(uint256 _tokenId) external view returns (address);
    function isApprovedForAll(address _owner, address _operator) external view returns (bool);
}
```

- cryptokitties.co
- ledgerlegends.com
- decentraland.org

long live the chains...

- blockchains world is very bubbling
- Ethereum has an hardfork

- 2016/07: The DAO
- steal of 3,600,000 ETH
- upgrade made to fix the issue and make a rollback
- part of community wanted to remain to the original fork (Ethereum Classic) but not the others (Ethereum)

figures

for Ethereum network

- blockchain size is ~ 667 GB
- 15,722 nodes on Ethereum network
- 274.247 TH/s of hash rate
- 5,809,818 blocks
- +252,740,000 transactions
- 14.4 s for block time
- ~ 3 ETH for reward
- 1 ETH ~ 495 USD
- PoW hash algorithm: Ethash (keccak-256)

- etherscan.io
- bitinfocharts.com
- coinbase

On 2018/06/18

Code d'Armor

Ðecentralized applications

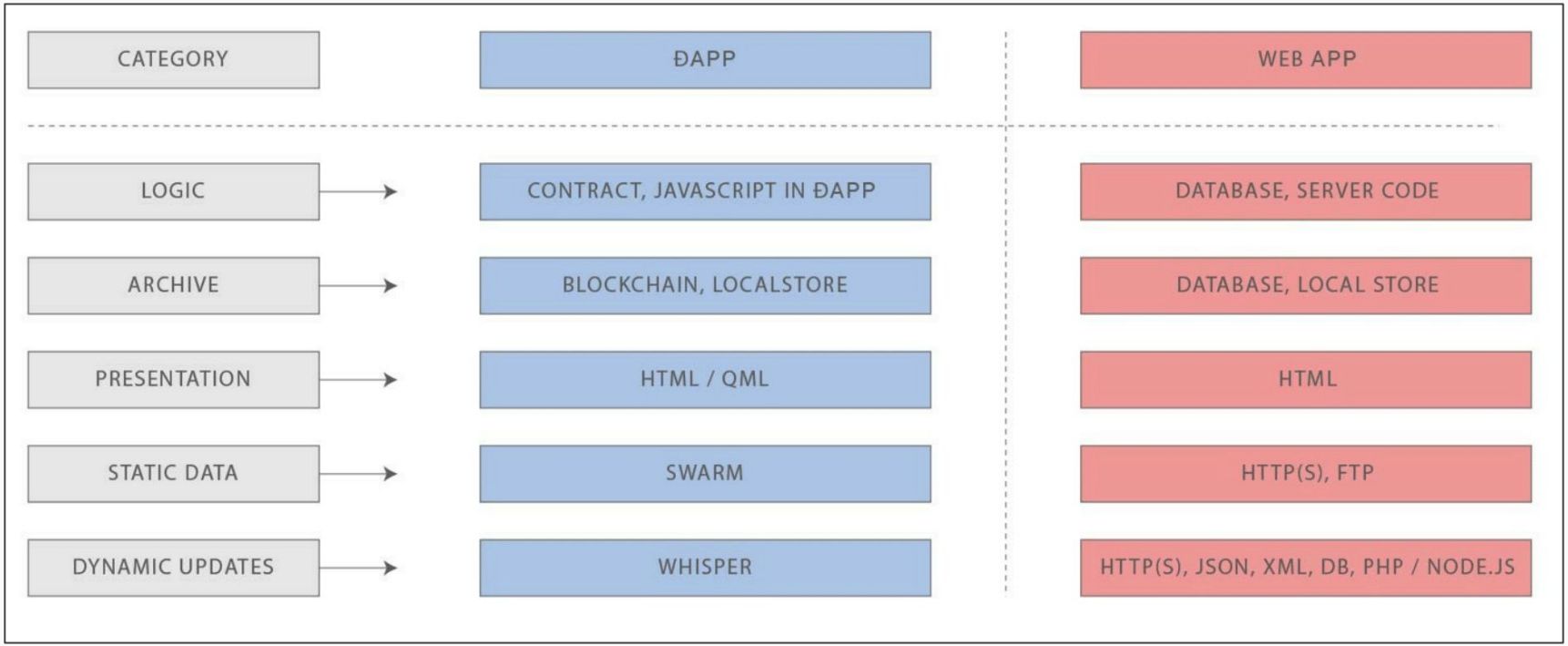
Decentralized applications

- “Old” paradigms
  - sequential -> 1 computer processing series of N instructions
  - parallel -> 1 computer processing K (in N) instructions at the same time (threads)
  - distributed -> X computers processing samples of the N instructions

- Decentralized paradigms
  - ➔ X computers processing at the same time the N instructions
  - ➔ programs are **duplicated** in computers
  - ➔ **no more unique server** or backend which hosts the program

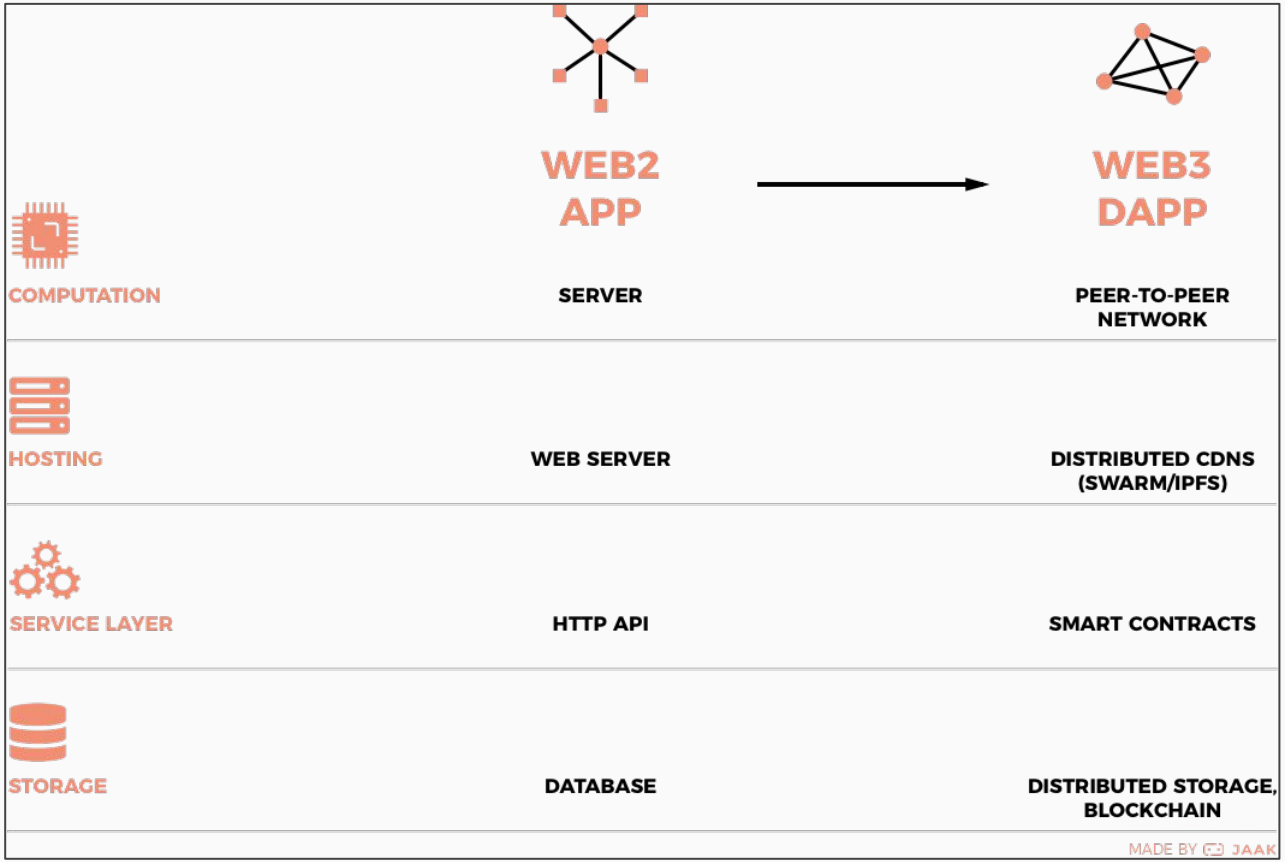
  - ➔ **no authorities** to trust
  - ➔ **no centralisation to fail**
  - ➔ but **slower than centralized solutions**

layers of Đapps (1)

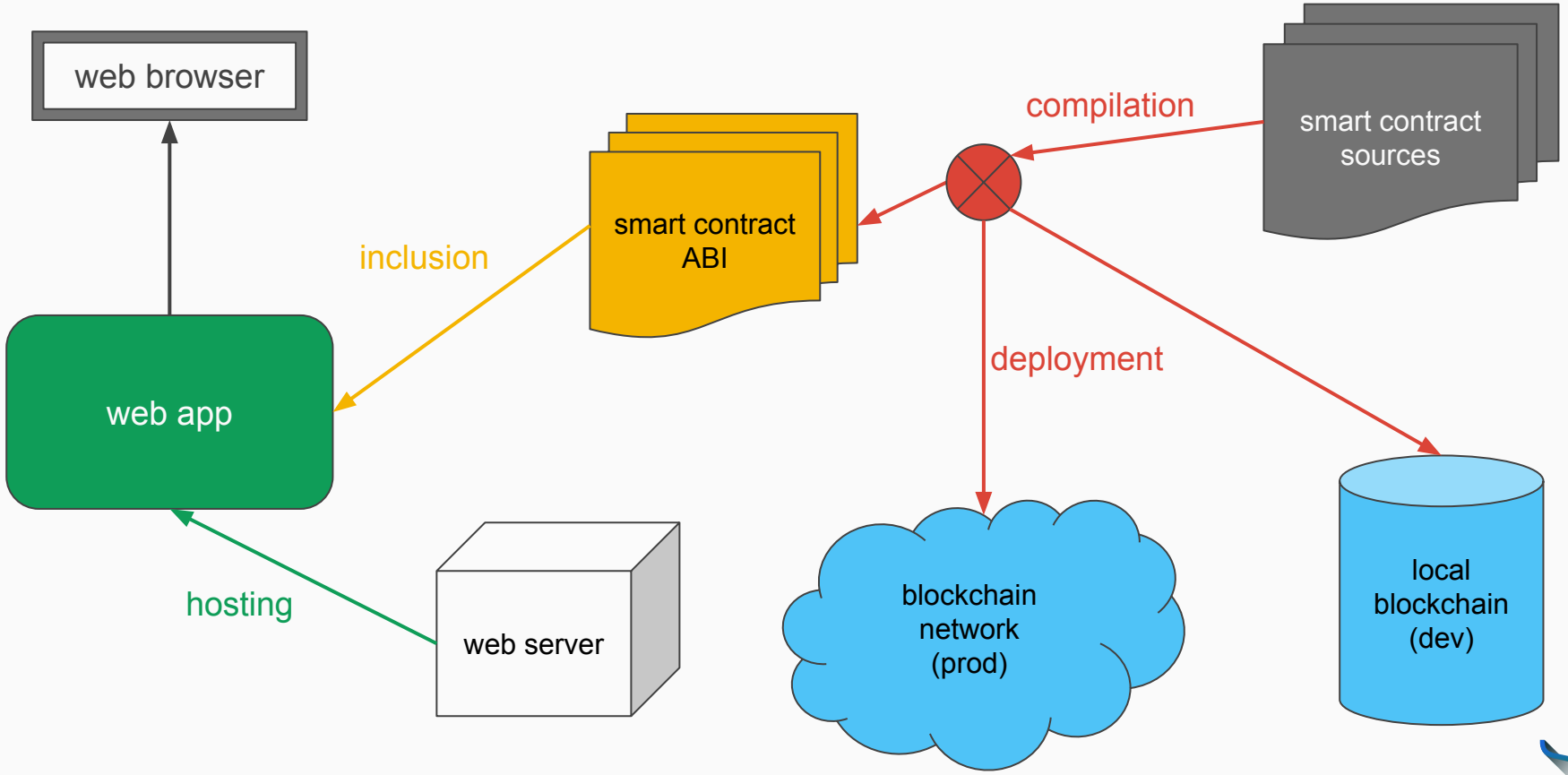


The Ethereum Experience

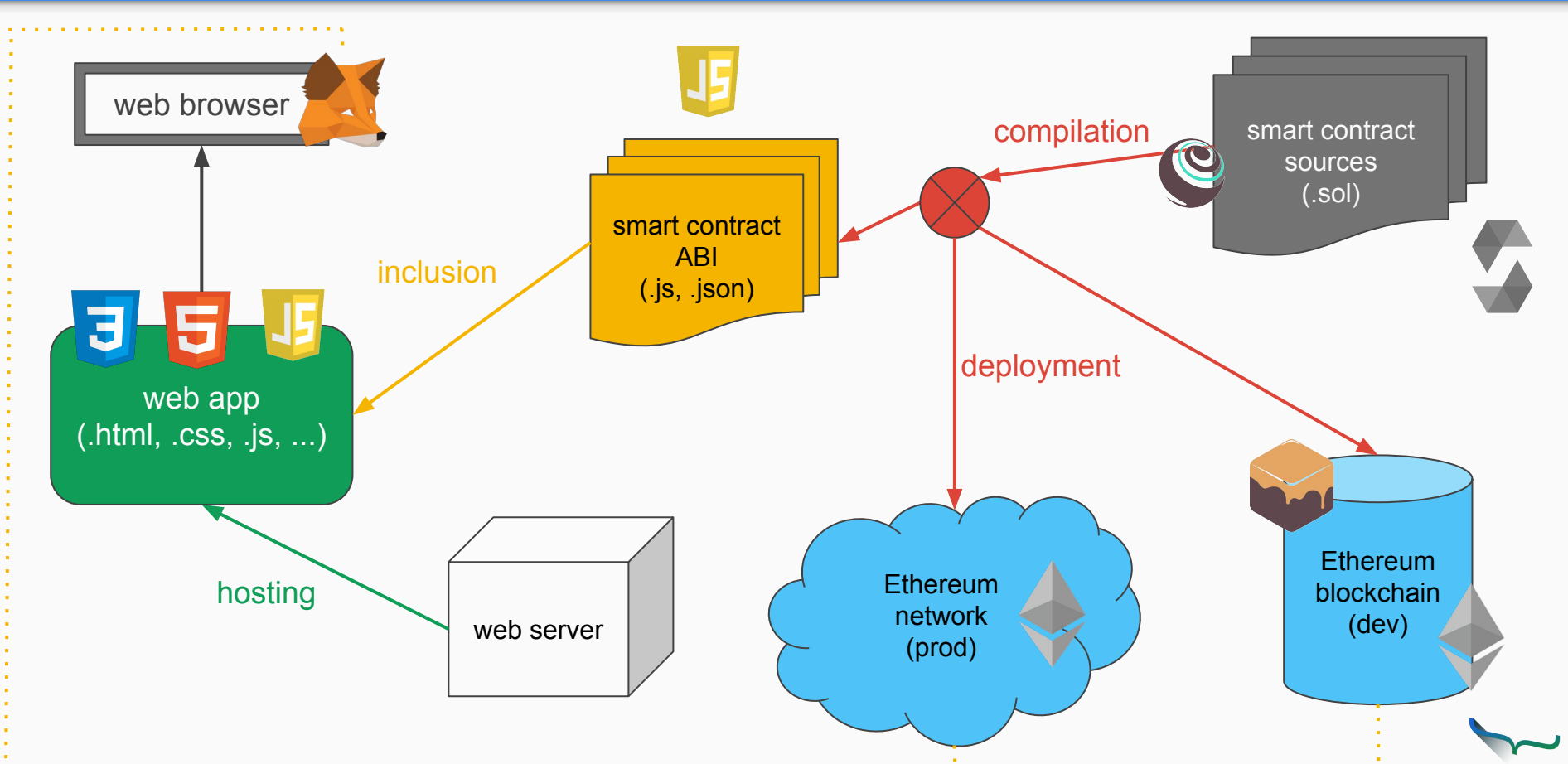
layers of Đapps (2)



architecture of web Đapps (1)



architecture of web Đapps (2)



Code d'Armor

smart contracts

smart contracts

common things

- have **balance** of tokens and unique **address**
- **the code is law**
- hosted in the blockchain
- once deployed, **cannot be**
  - **removed**
  - **modified**
- if bugs or flaws appear
  - write a new contract!
- must be **optimized and well written**
  - ownership
  - costs
  - tokens
  - ...

Solidity

Solidity

- **2014**, Gavin Wood, Ethereum project
- Influenced by JavaScript, Python, C++
- Last release: **0.4.24**
- For **smart contracts** supported on
  - Ethereum blockchains
  - Hedera hashgraph
  - ...
- Light API, few data structures, few data types

- on GitHub
- web site

On 2018/06/18

Solidity - types

```
enum DalekColors { Red, Blue, Yellow, White, Black, Orange }

struct Dalek {
  DalekColors color;
  bool isAlive;
  string name;
  // Gather fields by types to lower transactions costs
  // The smalled the type is, the cheaper (gas) the contract is
  uint32 power;
  uint32 level;
  // And also bytes, string, address, hex, bool...
  // But not yet floating point values! (06/2018)
}

mapping (uint => address) internal dalekToOwner;
Dalek[] public daleks;
```


Solidity - functions

```
/// @notice Compute the power and the level of a poney
/// @dev Modify the array of ponies in the arena
/// @param _poneyId The id of the poney in the arena's array
/// @return (uint32, uint32) The computed power and level
function _computePowerAndLevel(uint _poneyId) internal
returns (uint32 power, uint32 level) {
  // memory, storage = only for arrays and structs because they are heavy
  Poney storage poney = ponies[_poneyId];
  bytes memory nameAsArray = bytes(poney.name);
  assert(nameAsArray.length > 0);
  // keccack = Ethereum SHA3 mapping a string to a random 256 bit hex number
  uint32 newPower = uint32(keccak256(poney.name));
  uint32 newLevel = uint32(keccak256(poney.name));
  poney.power = newPower;
  poney.level = newLevel;
  return (newPower, newLevel);
}

/// @notice Create a new poney and add it in the arena
/// @param _name The name of the poney, should be unique
/// @dev No controls on the name of the poney, to do.
/// @return uint The id of the poney (to use in the array)
function createPoney(string _name) public onlyOwner() returns (uint) {
  bytes memory nameAsArray = bytes(_name);
  require(nameAsArray.length > 0);
  uint32 newPower = uint32(blockhash(block.number-1))% 5 + 1;
  uint32 newLevel = uint32(blockhash(block.number-1))% 5 + 1;
  Poney memory poney = Poney(true, _name, newPower, newLevel);
  uint poneyId = ponies.push(poney);
  /*uint32 level;
  (,level) = _computePowerAndLevel(poneyId);*/
  poneyToOwner[poneyId] = msg.sender;
  return poneyId;
}
```


Solidity - modifiers

```
// A modifier is called before the execution of a function
// and is used to check pre-conditions
modifier mustBeAlive(uint _dalekId, uint _poneyId) {
  // view = read data fom contract, without modifying them, does not burn gas
  // internal = private and visible to inheriting contracts
  // memory = copy object by value, not by reference
  Dalek memory dalek = daleks[_dalekId];
  Poney memory poney = ponies[_poneyId];
  // require = If false, gas refunded
  require(dalek.isAlive);
  require(poney.isAlive);
  _; // Go to caller function
}

function pewPewPew(uint _dalekId, uint _poneyId)
external mustBeAlive(_dalekId, _poneyId) {
  // external = callable only from outside
  // storage = use references, will modify the objects
  Dalek storage dalek = daleks[_dalekId];
  Poney storage poney = ponies[_poneyId];
  if (dalek.power >= poney.power) {
    poney.isAlive = false;
    dalek.level = dalek.level.add32(1);
    dalek.power = dalek.power.sub32(5);
    dalekVictims[dalek.name] = dalekVictims[dalek.name].add(1);
    emit DalekWin(_dalekId, _poneyId);
    } else {
      dalek.isAlive = false;
      poney.level = poney.level.mul32(2); // Holly poney!
      poney.power = poney.power.add32(10);
      poneyVictims[poney.name] = poneyVictims[poney.name].add(1);
      emit PoneyWin(_poneyId, _dalekId);
    }
  }
```


Solidity - inheritance

```
pragma solidity ^0.4.24;

import "./ERC721.sol";
import "./SafeMath.sol";
import "./PoneyFactory.sol";

contract PoneyOwnership is ERC721, PoneyFactory {

  using SafeMath for uint256;
```


Solidity - payable

```
// payable = we can receive Ether from this call
function giveCandiesToPoney(uint _poneyId) external payable {
  // msg.value = the value of the transaction, i.e. the call
  require (msg.value <= 100 ether);
  // this.balance = the amount of Ether in the contract
  owner.transfer(address(this).balance); // withdraw to our address
  Poney storage unicorn = ponies[_poneyId];
  unicorn.power *= 10; // Hazardous with overflows!
}
```


Solidity - events

```
// Events are signals thrown within the app
// We can use them also for history instead of storing too much data in the contract
// indexed = track events with this value
event DalekWin(uint indexed _dalekId, uint _poneyId);
event PoneyWin(uint indexed _poneyId, uint _dalekId);
```


Solidity - ownership and ERC721

```
contract Ownable {

  address public owner;

  event OwnershipRenounced(address indexed previousOwner);
  event OwnershipTransferred(
    address indexed previousOwner,
    address indexed newOwner
    );

    constructor() public {
      owner = msg.sender;
    }

    modifier onlyOwner() {
      require(msg.sender == owner);
      _;
    }

    function renounceOwnership() public onlyOwner {
      emit OwnershipRenounced(owner);
      owner = address(0);
    }

    function transferOwnership(address _newOwner) public onlyOwner {
      _transferOwnership(_newOwner);
    }

    function _transferOwnership(address _newOwner) internal {
      require(_newOwner != address(0));
      emit OwnershipTransferred(owner, _newOwner);
      owner = _newOwner;
    }

  }
```

```
contract PoneyOwnership is ERC721, PoneyFactory {

  using SafeMath for uint256;

  mapping (uint => address) internal poneyApprovals;

  function balanceOf(address _owner) public view returns (uint256 _balance) {
    return ownerPoneyCount[_owner];
  }

  function ownerOf(uint256 _tokenId) public view returns (address _owner) {
    return poneyToOwner[_tokenId];
  }

  function _transfer(address _from, address _to, uint256 _tokenId) private {
    ownerPoneyCount[_to] = ownerPoneyCount[_to].add(1);
    ownerPoneyCount[msg.sender] = ownerPoneyCount[msg.sender].sub(1);
    poneyToOwner[_tokenId] = _to;
    emit Transfer(_from, _to, _tokenId);
  }

  function transfer(address _to, uint256 _tokenId) public onlyOwnerOf(_tokenId) {
    _transfer(msg.sender, _to, _tokenId);
  }

  function approve(address _to, uint256 _tokenId) public onlyOwnerOf(_tokenId) {
    poneyApprovals[_tokenId] = _to;
    emit Approval(msg.sender, _to, _tokenId);
  }

  function takeOwnership(uint256 _tokenId) public {
    require(poneyApprovals[_tokenId] == msg.sender);
    address owner = ownerOf(_tokenId);
    _transfer(owner, msg.sender, _tokenId);
  }

  modifier onlyOwnerOf(uint256 _tokenId) {
    require(msg.sender == poneyToOwner[_tokenId]);
    _;
  }
}
```


Solidity - Application Binary Interface


```
  ],
  "name": "pewPewPew",
  "outputs": [],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
{
  "constant": true,
  "inputs": [],
  "name": "sayHello",
  "outputs": [
    {
      "name": "",
      "type": "string"
    }
  ],
  "payable": false,
  "stateMutability": "pure",
  "type": "function"
},
{
  "constant": false,
  "inputs": [
    {
      "name": "_nameOfPoney",
      "type": "string"
    }
  ],
  "name": "getNewPoney",
  "outputs": [
    {
      "name": "",
      "type": "uint256"
    }
  ],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
},
```


Web3

web3

common things about the decentralised web

- Ethereum **JavaScript** API
- implementing **JSON RPC** specifications
- used so as to **deal with smart contracts** on blockchains
- allows to interact with a local/remote Ethereum node

web3 - set up

```
window.addEventListener('load', function() {
  // if (typeof web3 !== 'undefined') {
  //   console.log("Use Web3 with Web3js current provider");
  //   web3js = new Web3(web3.currentProvider);
  // } // else : kabooooom
  if (typeof web3 !== 'undefined') {
    web3 = new Web3(web3.currentProvider);
  } else {
    // set the provider you want from Web3.providers
    web3 = new Web3(new Web3.providers.HttpProvider(HTTP_PROVIDER));
  }
  startApp();
})
```

```
// To get the address, compile and deploy to the blockchain the contract "Arena"
const TOP_CONTRACT_ADDRESS = "0x340bc26a0afe4ef3304a47740b7ad1ae31c1c52e";
// The JSON-based signatures of the smart contract we want to use
// ABI = Application Binary Interface
const TOP_CONTRACT_ABI_FILE = "./build/contracts/Arena.json";
// In case the browser is not compatible with Web3 :()
const HTTP_PROVIDER = "http://localhost:8545";

let web3js;
let userAccount;
let arenaContract;
let arenaContractInstance;
let transactionConfig;

// Initializes the Dapp with addresses, ABI...
function startApp(){
  console.log("Loading ABI...");
  $.getJSON(TOP_CONTRACT_ABI_FILE, function(topContractJson){
    let topContractAbi = topContractJson.abi;
    console.log("Gotten ABI: "+JSON.stringify(topContractJson));
    arenaContract = web3.eth.contract(topContractAbi);
    arenaContractInstance = arenaContract.at(TOP_CONTRACT_ADDRESS);
    sayHello();
    // Beware, accounts may be switched, not dealt here
    userAccount = web3.eth.defaultAccount; // web3.eth.accounts[0];
    console.log("Listening to signals...");
    listenToSignals();
  });
} // End of start app
```

web3 - call external functions of contracts

```
function getNumberOfFighters(){
  arenaContractInstance.getNumberOfPonies(function(error, result){
    console.log("> Called 'getNumberOfPonies': "+result)
  });
  arenaContractInstance.getNumberOfDaleks(function(error, result){
    console.log("> Called 'getNumerOfDaleks': "+result)
  });
}

function fight(){
  let dalek = $("#whichDalek").val();
  let poney = $("#whichPoney").val();
  arenaContractInstance.pewPewPew(dalek, poney, function(error, result){
    console.log("> Called 'pewPewPew': "+result)
  });
}

function boost(){
  let poney = $("#whichPoney").val();
  let weiValue = web3.toWei(5,'ether');
  arenaContractInstance.giveCandiesToPoney(poney, {from: userAccount, value: weiValue},  function(error, result){
    console.log("> Called 'giveCandiesToPoney': "+result)
  });
}
```

web3 - listen to events thrown by contracts

```
let signalDalekVictory, signalPoneyVictory;
function listenToSignals(){
  signalDalekVictory = arenaContractInstance.DalekWin();
  signalPoneyVictory = arenaContractInstance.PoneyWin();
  signalDalekVictory.watch(function(err, result){ console.log("> EXTERMINATE !");})
  signalPoneyVictory.watch(function(err, result){ console.log("> LOOK AT MY HORSE !");})
}
```


Code d'Armor

conclusion

about Đapps

and blockchains

- blockchains provide
  - **persistance**
  - **decentralization**
  - missing of regulation
  - **immutability** of records

- blockchains **may be slow**
  - need to mine a block containing the transaction to trigger contracts and calls

- apps **may have decentralized parts**:
  - logic
  - traces
  - some features

- apps **might need centralized parts:**
  - massive storage
  - legacy features

then?

new types of ledgers

- other solutions may be used with blockchains, e.g.:

  ➔ Hyperledger
  - Linux Foundation
  - massive **toolbox**
  - private blockchain, ...

- **hashgraph** can replace blockchains and host Đapps:

  ➔ faster, higher number of transactions, more regulated

  ➔ faster consensus

Code d'Armor

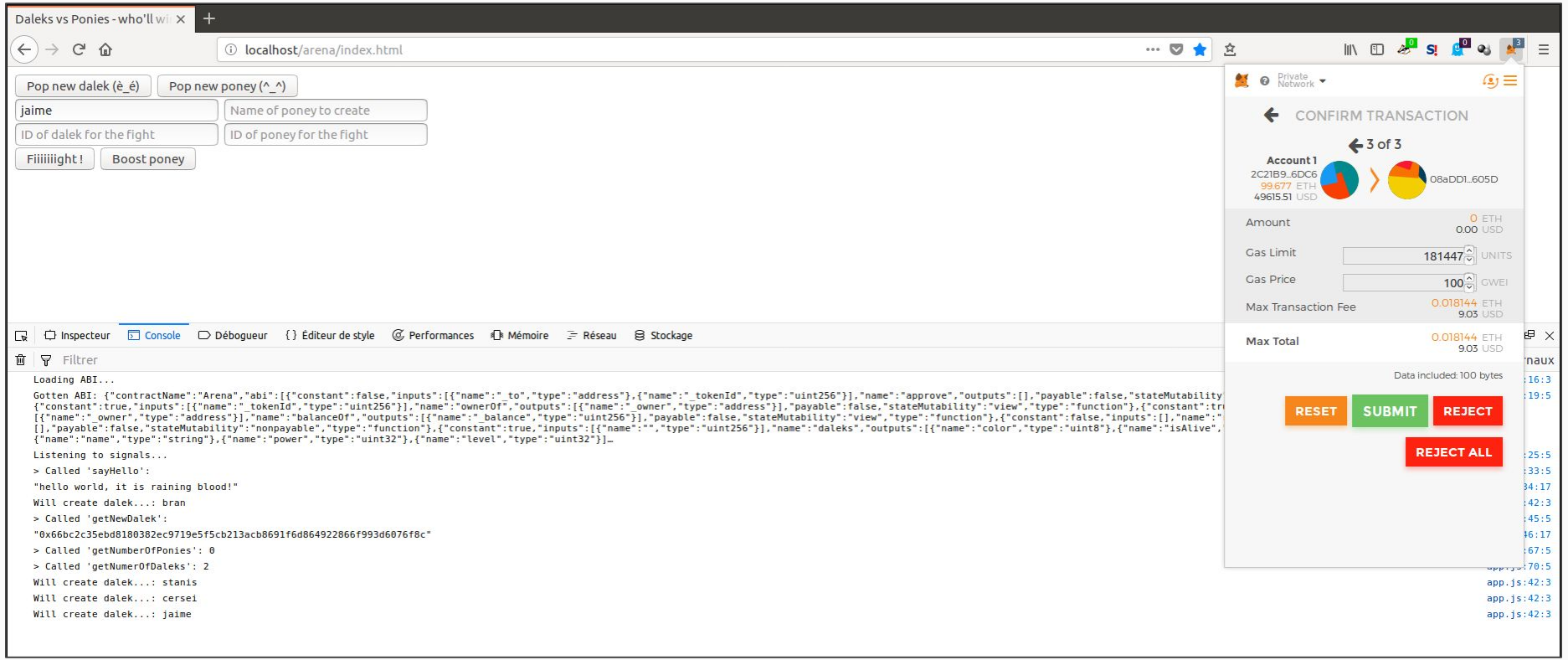
thanks!

:)

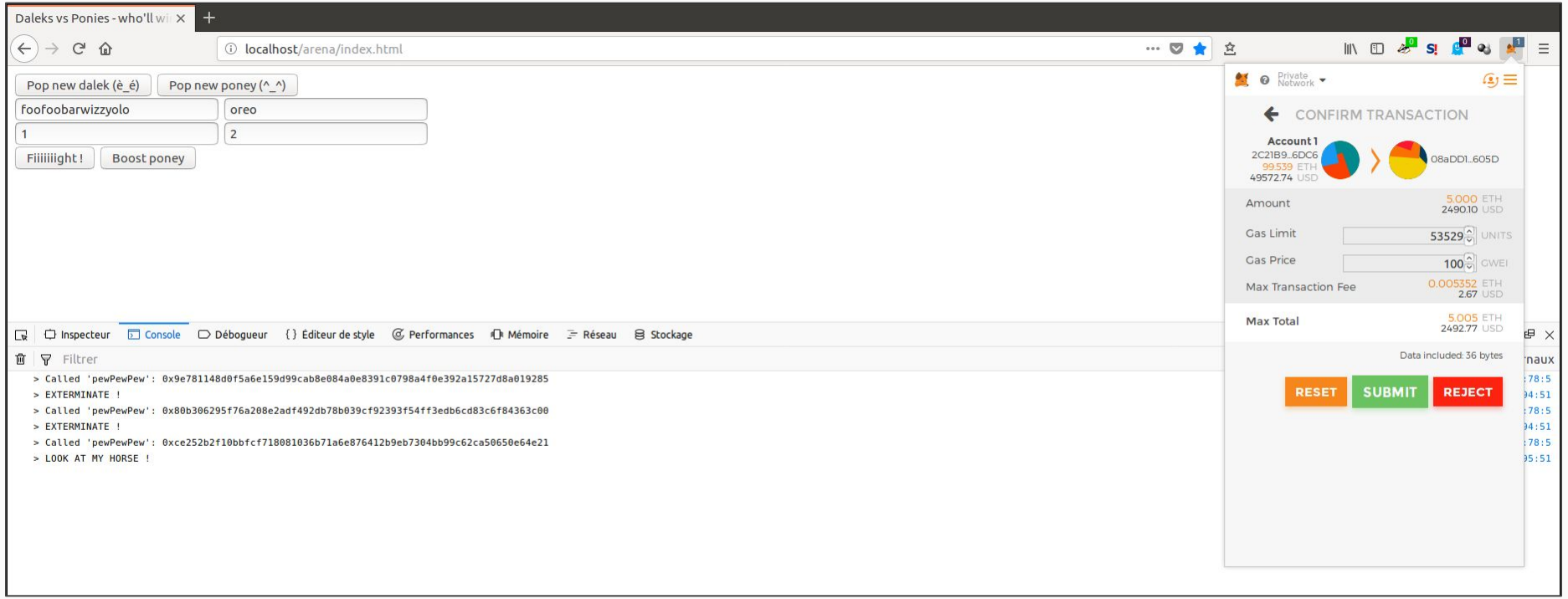
Code d'Armor

demos

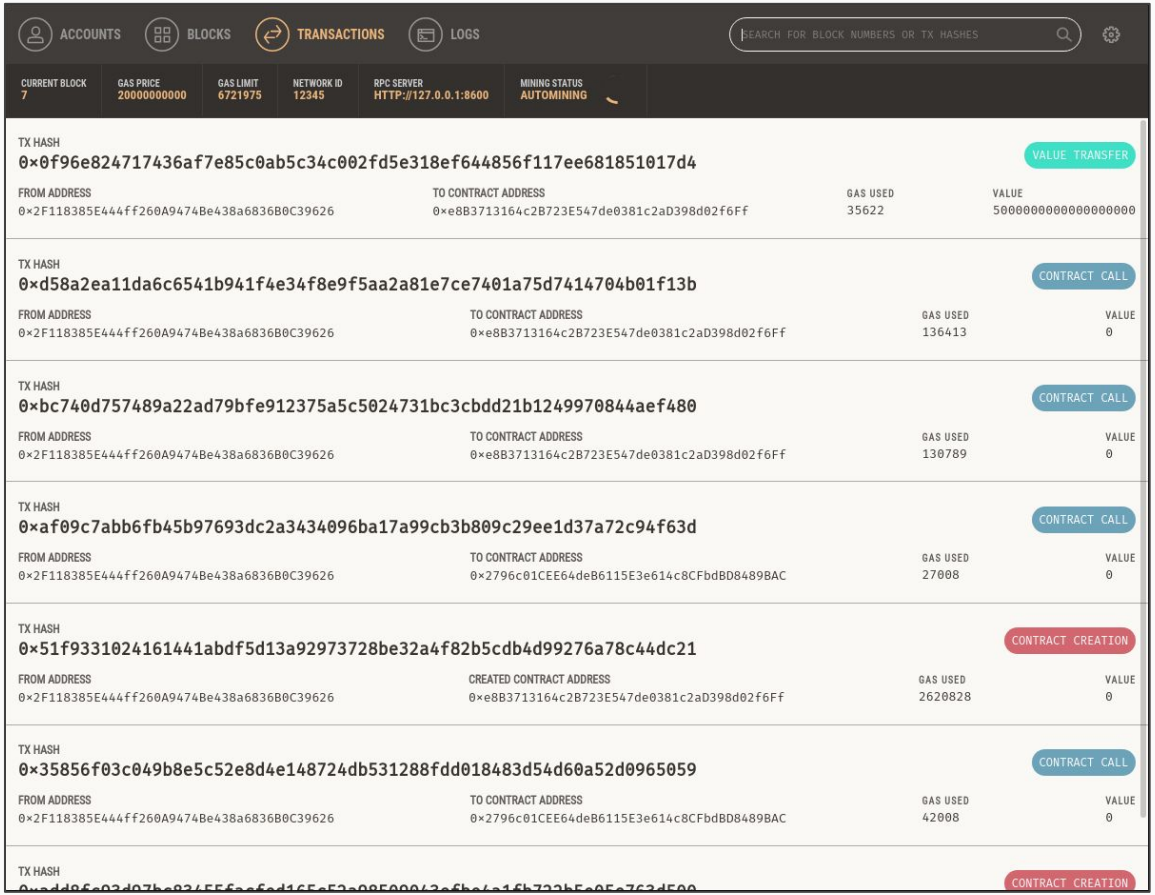
demo - creation of new dalek (transaction, MetaMask)



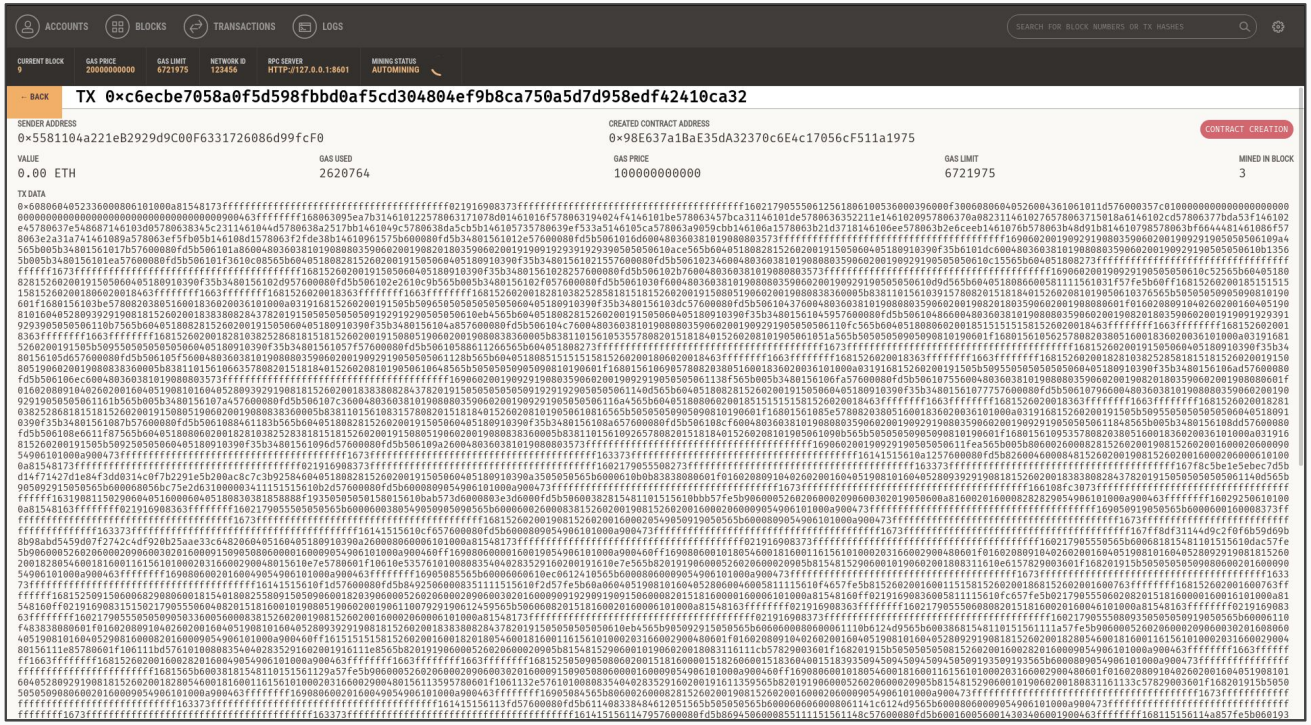
demo - boost a poney (transaction with ETH, MetaMask)



demo - Ganache



demo - bytecode (Ganache + Etherscan)



Decoded Bytecode :

[1] PUSH1 0x80
[3] PUSH1 0x40
[4] MSTORE
[5] CALLER
[7] PUSH1 0x00
[8] DUP1
[11] PUSH2 0x0100
[12] EXP
[13] DUP2
[14] SLOAD
[15] DUP2
[36] PUSH20 0xffffffffffffffffffffffffffffffffffffffff
[37] MUL
[38] NOT
[39] AND
[40] SWAP1
[41] DUP4
[62] PUSH20 0xffffffffffffffffffffffffffffffffffffffff
[63] AND
[64] MUL
[65] OR
[66] SWAP1
[67] SSTORE
[68] POP
[71] PUSH2 0x2561
[72] DUP1
[75] PUSH2 0x0053
[77] PUSH1 0x00
[78] CODECOPY
[80] PUSH1 0x00
[81] RETURN
[82] STOP
[84] PUSH1 0x80
[86] PUSH1 0x40
[87] MSTORE
[89] PUSH1 0x04
[90] CALLDATASIZE
[91] LT
[94] PUSH2 0x011d
[95] JUMPI
[97] PUSH1 0x00
[98] CALLDATALOAD
[128] PUSH29 0x0100000000000000000000000000000000000000000000000000000000
[129] SWAP1
[130] DIV
[135] PUSH4 0xffffffff
[136] AND
[137] DUP1

demo - script (use Truffle console and Ganache)

```
# Here are workflows of demos to dodo
0. Make things easier
   ```solidity
   Arena.deployed().then(inst => { Arena = inst });
   ```

1. Using MetaMask, may a transfer of ETH tokens from an account to another using the addresses
of the accounts displayed within Ganache. Show the variations of balances.
2. Migrate (= deploy) a smart contract within the blockchain hosted and managed by Ganache.
Find the transaction in the blockchain, get its big payload (in hex format) and convert it to Ethereum bytecode
thanks to https://etherscan.io/opcode-tool
3. Hello World!
   ```solidity
   Arena.sayHello();
   ```

4. Create 3 ponies and 3 daleks, read their values
   ```solidity
   Arena.getNewPoney("cookie");
   Arena.getNewPoney("patachou");
   Arena.getNewPoney("leulipaupe");
   Arena.getNewDalek("The Doctor");
   Arena.getNewDalek("Foo");
   Arena.getNewDalek("Trump");
   Arena.getDetailsOfPoney(0);
   Arena.getDetailsOfPoney(1);
   Arena.getDetailsOfPoney(2);
   Arena.getDetailsOfDalek(0);
   Arena.getDetailsOfDalek(1);
   Arena.getDetailsOfDalek(2);
   Arena.getNumberOfDaleks();
   Arena.getNumberOfPonies();
   ```

5. Make some fights and check the variations of fields
   ```solidity
   Arena.getDetailsOfDalek(1);
   Arena.getDetailsOfPoney(2);
   Arena.pewPewPew(1,2);
   Arena.getDetailsOfDalek(1);
   Arena.getDetailsOfPoney(2);
   ```

6. Listen to siggnals (Solidity events) and make a new fight so as to see the thrown event
   ```solidity
   let signalDalekVictory = Arena.DalekWin();
   let signalPoneyVictory = Arena.PoneyWin();
   signalDalekVictory.watch(function(err, result){ console.log("EXTERMINATE !"); console.log(result.args); })
   signalPoneyVictory.watch(function(err, result){ console.log("LOOK AT MY HORSE !"); console.log(result.args); })
   Arena.pewPewPew(0,0);
   ```

7. Boost a poney using candies baught in ETH
   ```solidity
   Arena.getDetailsOfPoney(0);
   Arena.giveCandiesToPoney(0, {from: "0x675623D8d1ee9eA678364B5391907448488BbbFe2", value: 1300000000000000000});
   Arena.getDetailsOfPoney(0);

and show the dedicated transaction with a non-null ETH value and the new balances
```



```

Code d'Armor

links

(⌐□_□)

Tools

- BitInfoCharts
- Coinbase
- Ethereum Natural Specification Format
- Ethereum Network Status
- Ethernodes
- Etherscan
- Etherscan gas price
- Etherscan ByteCode to Opcode Disassembler
- ETH Gas station
- Ganache
- Geth
- Infura
- Loom
- Metamask
- Mist
- Open Zeppelin
- Porosity
- Remix IDE
- State of the ĐApps
- Truffle
- Web3

References

- A 101 Noob Intro to Programming Smart Contracts on Ethereum, ConsenSys
- Bloc Genesis, bitcoin.fr
- Create Your Own Crypto-currency with Ethereum, ethereum.org
- Crossing Over to Web3 - An Introduction to Decentralised Development, Luke Hedger
- CryptoKitties, cryptokitties.co
- Cryptomonnaies - mode d'emploi en 20', Plerre-Yves Lapersonne
- Decentraland, decentraland.org
- Demystifying Hashgraph: Benefits and Challenges, Yaoqi Jia
- Ethash, github.com/ethereum
- Ethereum: Ether, Ether Gas, Gas Limit, Gas Price & Fees [All you need to know to get into an ICO], coinsutra.com
- Ethereum "Gas" - How it works, steemit.com
- Full-stack smart contract development, Júlio Santos
- Hashgraph wants to give you the benefits of blockchain without the limitations, Samantha Stein
- Hedera Hashgraph, hederahashgraph.com
- How to build your own Ethereum based ERC20 Token and launch an ICO in next 20 minutes, Sandeep Panda
- Hyperledger, hyperledger.org
- Introduction to Smart Contracts, solidity.readthedocs.io

Web sites visited on 2018/06/20

References

- The Next Generation of Distributed Ledger Technology, iota.org
- La Révolution des Blockchains, Pierre-Yves Lapersonne
- Ledger Legends
- Proof of Work vs Proof of Stake: Basic Mining Guide, blockgeeks.com
- Reversing EVM bytecode with radare2, positive.com
- Swarm, swarm-guide.readthedocs.io
- The Ethereum Experience, Ethereum
- Then Tangle, Serguei Popov
- The Tangle: an Illustrated Introduction, Alon Gal
- Understanding Blockchain Fundamentals, Part 1: Byzantine Fault Tolerance, Georgios Konstantopoulos
- Walking Through the ERC721 Full Implementation, Karen Scarbrough
- What are Sidechains?, Shaan Ray
- Whisper, github.com/ethereum

Web sites visited on 2018/06/20

Credits

- Logo of Bitcoin - Copyright © **Bitcoin Project**. All Rights Reserved.
- Logo of Code d'Armor - Copyright © **Code d'Armor**. All Rights Reserved.
- Logo of CSS3 - **icones8.fr**
- Logo of Ganache - Copyright © **Consensys**. All Rights Reserved.
- Logo of HTML5 - Copyright © **W3C**, *CC BY 3.0*
- Logo of Hedera Hashgraph - Copyright © **Hashgraph Consortium**. All Rights Reserved.
- Logo of Hyperledger - Copyright © **The Linux Foundation**. All Rights Reserved.
- Logo of IOTA - Copyright © **IOTA Foundation**. All Rights Reserved.
- Logo of JavaScript - **javatpoint.com**, modified with Gimp
- Logo of MetaMask - **metamask.io**
- Logo of Solidity - Copyright © **Ethereum Foundation**, *CC BY-SA 3.0*
- Logo of Truffle - Copyright © **Consensys**. All Rights Reserved.
- Pictures by **Freepik**, *Flaticon Basic License*

Code d'Armor

source code

(^_^)

```
pragma solidity ^0.4.24;

import "./DalekFactory.sol";
import "./PoneyOwnership.sol";
import "./Ownable.sol";

contract Arena is Ownable, PoneyOwnership, DalekFactory {
  mapping (string => uint256) internal dalekVictims;
  mapping (string => uint256) internal poneyVictims;
  event DalekWin(uint indexed _dalekId, uint _poneyId);
  event PoneyWin(uint indexed _poneyId, uint _dalekId);
  function giveCandiesToPoney(uint _poneyId) external payable {
    require (msg.value <= 100 ether);
    owner.transfer(address(this).balance); // withdraw to our address
    Poney storage unicorn = ponies[_poneyId];
    unicorn.power *= 10; // Hazardous with overflows!
  }
  modifier mustBeAlive(uint _dalekId, uint _poneyId) {
    Dalek memory dalek = daleks[_dalekId];
    Poney memory poney = ponies[_poneyId];
    require(dalek.isAlive);
    require(poney.isAlive);
    _;
  }
  function pewPewPew(uint _dalekId, uint _poneyId)
    external mustBeAlive(_dalekId, _poneyId) {
      Dalek storage dalek = daleks[_dalekId];
      Poney storage poney = ponies[_poneyId];
      if (dalek.power >= poney.power) {
        poney.isAlive = false;
        dalek.level = dalek.level.add32(1);
        dalek.power = dalek.power.sub32(5);
        dalekVictims[dalek.name] = dalekVictims[dalek.name].add(1);
        emit DalekWin(_dalekId, _poneyId);
        } else {
          dalek.isAlive = false;
          poney.level = poney.level.mul32(2); // Holly poney!
          poney.power = poney.power.add32(10);
          poneyVictims[poney.name] = poneyVictims[poney.name].add(1);
          emit PoneyWin(_poneyId, _dalekId);
        }
    }
```

sources - Arena.sol (1)

```
44    function sayHello() external pure returns (string) {
45      return "hello world, it is raining blood!";
46    }
47    function getNewPoney(string _nameOfPoney) external returns (uint){
48      return createPoney(_nameOfPoney);
49    }
50    function getNewDalek(string _nameOfDalek) external returns (uint){
51      return createDalek(_nameOfDalek);
52    }
53    function getNumberOfPonies() external view returns (uint) {
54      return ponies.length;
55    }
56    function getNumberOfDaleks() external view returns (uint) {
57      return daleks.length;
58    }
59    function getDetailsOfDalek(uint _dalekId) external view
60      returns (string name, bool isAlive, uint32 level, uint32 power) {
61        Dalek memory dalek = daleks[_dalekId];
62        return (dalek.name, dalek.isAlive, dalek.level, dalek.power);
63    }
64    function getDetailsOfPoney(uint _poneyId) external view
65      returns (string name, bool isAlive, uint32 level, uint32 power) {
66        Poney memory poney = ponies[_poneyId];
67        return (poney.name, poney.isAlive, poney.level, poney.power);
68    }
69  }
70
```

```
pragma solidity ^0.4.24;
import "./Arena.sol";
import "./SafeMath.sol";
import "./Ownable.sol";
contract DalekFactory is Ownable {
  using SafeMath for uint32;
  using SafeMath for uint256;
  enum DalekColors { Red, Blue, Yellow, White, Black, Orange }
  struct Dalek {
    DalekColors color;
    bool isAlive;
    string name;
    uint32 power;
    uint32 level;
  }
  mapping (uint => address) internal dalekToOwner;
  Dalek[] public daleks;
  function createDalek(string _name) public onlyOwner() returns (uint){
    bytes memory nameAsArray = bytes(_name);
    assert(nameAsArray.length > 0);
    Dalek memory newDalek = Dalek(DalekColors.Black, true, _name, 7, 7);
    uint dalekId = daleks.push(newDalek);
    dalekToOwner[dalekId] = msg.sender;
    return dalekId;
  }
}

```

sources - DalekFactory.sol

```
2  pragma solidity ^0.4.24;
3  contract ERC721 {
4    event Transfer(address indexed _from, address indexed _to, uint256 _tokenId);
5    event Approval(address indexed _owner, address indexed _approved, uint256 _tokenId);
6
7    function balanceOf(address _owner) public view returns (uint256 _balance);
8    function ownerOf(uint256 _tokenId) public view returns (address _owner);
9    function transfer(address _to, uint256 _tokenId) public;
10   function approve(address _to, uint256 _tokenId) public;
11   function takeOwnership(uint256 _tokenId) public;
12 }
```

sources - ERC721.sol

```
pragma solidity ^0.4.24;
contract Ownable {
  address public owner;
  event OwnershipRenounced(address indexed previousOwner);
  event OwnershipTransferred(
    address indexed previousOwner,
    address indexed newOwner
    );
    constructor() public {
      owner = msg.sender;
    }
    modifier onlyOwner() {
      require(msg.sender == owner);
      _;
    }
    function renounceOwnership() public onlyOwner {
      emit OwnershipRenounced(owner);
      owner = address(0);
    }
    function transferOwnership(address _newOwner) public onlyOwner {
      _transferOwnership(_newOwner);
    }
    function _transferOwnership(address _newOwner) internal {
      require(_newOwner != address(0));
      emit OwnershipTransferred(owner, _newOwner);
      owner = _newOwner;
    }
  }
```

sources - Ownable.sol

```
pragma solidity ^0.4.24;
import "./SafeMath.sol";
import "./Ownable.sol";
contract PoneyFactory is Ownable {
  using SafeMath for uint32;
  using SafeMath for uint256;
  struct Poney {
    bool isAlive;
    string name;
    uint32 power;
    uint32 level;
  }
  mapping (address => uint) internal ownerPoneyCount;
  mapping (uint => address) internal poneyToOwner;
  Poney[] public ponies;
  function _computePowerAndLevel(uint _poneyId) internal
    returns (uint32 power, uint32 level) {
      Poney storage poney = ponies[_poneyId];
      bytes memory nameAsArray = bytes(poney.name);
      assert(nameAsArray.length > 0);
      uint32 newPower = uint32(keccak256(poney.name));
      uint32 newLevel = uint32(keccak256(poney.name));
      poney.power = newPower;
      poney.level = newLevel;
      return (newPower, newLevel);
  }
  function createPoney(string _name) public onlyOwner() returns (uint) {
    bytes memory nameAsArray = bytes(_name);
    require(nameAsArray.length > 0);
    uint32 newPower = uint32(blockhash(block.number-1))% 5 + 1;
    uint32 newLevel = uint32(blockhash(block.number-1))% 5 + 1;
    Poney memory poney = Poney(true, _name, newPower, newLevel);
    uint poneyId = ponies.push(poney);
    poneyToOwner[poneyId] = msg.sender;
    return poneyId;
  }
}
```

sources - PoneyFactory.sol

```
pragma solidity ^0.4.24;
import "./ERC721.sol";
import "./SafeMath.sol";
import "./PoneyFactory.sol";
contract PoneyOwnership is ERC721, PoneyFactory {
  using SafeMath for uint256;
  mapping (uint => address) internal poneyApprovals;
  function balanceOf(address _owner) public view returns (uint256 _balance) {
    return ownerPoneyCount[_owner];
  }
  function ownerOf(uint256 _tokenId) public view returns (address _owner) {
    return poneyToOwner[_tokenId];
  }
  function _transfer(address _from, address _to, uint256 _tokenId) private {
    ownerPoneyCount[_to] = ownerPoneyCount[_to].add(1);
    ownerPoneyCount[msg.sender] = ownerPoneyCount[msg.sender].sub(1);
    poneyToOwner[_tokenId] = _to;
    emit Transfer(_from, _to, _tokenId);
  }
  function transfer(address _to, uint256 _tokenId) public onlyOwnerOf(_tokenId) {
    _transfer(msg.sender, _to, _tokenId);
  }
  function approve(address _to, uint256 _tokenId) public onlyOwnerOf(_tokenId) {
    poneyApprovals[_tokenId] = _to;
    emit Approval(msg.sender, _to, _tokenId);
  }
  function takeOwnership(uint256 _tokenId) public {
    require(poneyApprovals[_tokenId] == msg.sender);
    address owner = ownerOf(_tokenId);
    _transfer(owner, msg.sender, _tokenId);
  }
  modifier onlyOwnerOf(uint256 _tokenId) {
    require(msg.sender == poneyToOwner[_tokenId]);
    _;
  }
}
```

sources - PoneyOwnership.sol

```html
<!DOCTYPE html>
<html lang="en">

  <head>
    <meta charset="UTF-8">
    <title>Daleks vs Ponies - who'll win?</title>
    <meta name="description" content="Proof Of Concept of Dapp with quick and dirty written code where daleks fight ponies." />
    <!-- JS voodoo glue -->
    <script language="javascript" type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <!-- Web3 - The project to use to run Dapp
        Web3 is the entry of all the API and features provding cools things to use with blockchains.
        Contract, blocks, batch, plenty of cool things!
        See https://github.com/ethereum/web3.js/blob/1.0/dist/web3.min.js -->
    <!--<script language="javascript" type="text/javascript" src="web3.min.js"></script>-->
    <script src="https://cdn.jsdelivr.net/gh/ethereum/web3.js/dist/web3.min.js"></script>
    <!-- Some logic -->
    <script language="javascript" type="text/javascript" src="app.js"></script>
  </head>

  <body>

    <!-- Some dumb fields -->
    <button id="popDalek" type="button" onclick="createDalek();">Pop new dalek (è_é)</button>
    <button id="popPoney" type="button" onclick="createPoney();">Pop new poney (^_^)</button>
    <br />
    <input id="newDalekName" type="text" placeholder="Name of dalek to create"/>
    <input id="newPoneyName" type="text" placeholder="Name of poney to create"/>
    <br />
    <input id="whichDalek" type="text" placeholder="ID of dalek for the fight"/>
    <input id="whichPoney" type="text" placeholder="ID of poney for the fight"/>
    <br />
    <button id="fight" type="button" onclick="fight();">Fiiiiiight !</button>
    <button id="boost" type="button" onclick="boost();">Boost poney</button>

  </body>

</html>
```

sources - index.html

```
const TOP_CONTRACT_ADDRESS = "0xe8b3713164c2b723e547de0381c2ad398d02f6ff";
const TOP_CONTRACT_ABI_FILE = "./build/contracts/Arena.json";
const HTTP_PROVIDER = "http://localhost:8545";

let web3js;
let userAccount;
let arenaContract;
let arenaContractInstance;
let transactionConfig;

window.addEventListener('load', function() {
  if (typeof web3 !== 'undefined') {
    web3 = new Web3(web3.currentProvider);
  } else {
    // set the provider you want from Web3.providers
    web3 = new Web3(new Web3.providers.HttpProvider(HTTP_PROVIDER));
  }
  startApp();
})


function startApp(){
  console.log("Loading ABI...");
  $.getJSON(TOP_CONTRACT_ABI_FILE, function(topContractJson){
    let topContractAbi = topContractJson.abi;
    console.log("Gotten ABI: "+JSON.stringify(topContractJson));
    arenaContract = web3.eth.contract(topContractAbi);
    arenaContractInstance = arenaContract.at(TOP_CONTRACT_ADDRESS);
    sayHello();
    userAccount = web3.eth.defaultAccount; // web3.eth.accounts[0];
    console.log("Listening to signals...");
    listenToSignals();
  });
}
```

```
function sayHello(){
  arenaContractInstance.sayHello(function(error, result){
    console.log("> Called 'sayHello': ");
    if (!error) console.log(JSON.stringify(result));
    else console.error(error);
  });
}

function createDalek(){
  let name = $("#newDalekName").val();
  console.log("Will create dalek...: "+name);
  arenaContractInstance.getNewDalek(name, function(error, result){
    console.log("> Called 'getNewDalek': ");
    if (!error) console.log(JSON.stringify(result));
    else console.error(error);
    getNumberOfFighters();
  });
}

function createPoney(){
  let name = $("#newPoneyName").val();
  console.log("Will create poney...: "+name);
  arenaContractInstance.getNewPoney(name, function(error, result){
    console.log("> Called 'getNewPoney': ");
    if (!error) console.log(JSON.stringify(result));
    else console.error(error);
    getNumberOfFighters();
  });
}

```

sources - app.js (2)

```
function getNumberOfFighters(){
  arenaContractInstance.getNumberOfPonies(function(error, result){
    console.log("> Called 'getNumberOfPonies': "+result)
  });
  arenaContractInstance.getNumberOfDaleks(function(error, result){
    console.log("> Called 'getNumerOfDaleks': "+result)
  });
}

function fight(){
  let dalek = $("#whichDalek").val();
  let poney = $("#whichPoney").val();
  arenaContractInstance.pewPewPew(dalek, poney, function(error, result){
    console.log("> Called 'pewPewPew': "+result)
  });
}

function boost(){
  let poney = $("#whichPoney").val();
  let weiValue = web3.toWei(5,'ether');
  arenaContractInstance.giveCandiesToPoney(poney, {from: userAccount, value: weiValue},  function(error, result){
    console.log("> Called 'giveCandiesToPoney': "+result)
  });
}

let signalDalekVictory, signalPoneyVictory;
function listenToSignals(){
  signalDalekVictory = arenaContractInstance.DalekWin();
  signalPoneyVictory = arenaContractInstance.PoneyWin();
  signalDalekVictory.watch(function(err, result){ console.log("> EXTERMINATE !");})
  signalPoneyVictory.watch(function(err, result){ console.log("> LOOK AT MY HORSE !");})
}
```

1_initial_migration.js

```
var Migrations = artifacts.require("./Migrations.sol");

module.exports = function(deployer) {
  deployer.deploy(Migrations);
};
```

2_deploy_contract.js

```
const Arena = artifacts.require("./Arena.sol")

module.exports = function(deployer) {
  deployer.deploy(Arena);
};

```

sources - migrations JS file

 # Init folder with Truffle framework
$ truffle init

 # Create a Foo contract with Truffle (creates test files, ...)
$ truffle create contract Foo

 # Define a migration to use to deploy to blockchain
$ truffle create migration code_migration_xxx

 # Compile Solidity sources with Truffle
$ rm -rf build/contracts/*
$ truffle compile

 # Add in truffle.js a new network (name, host, port, network id)

 # Migrate contracts to the blockchain
$ truffle migrate --network name-of-network

 # Debug mode on the blockchain
$ truffle console --network name-of-network