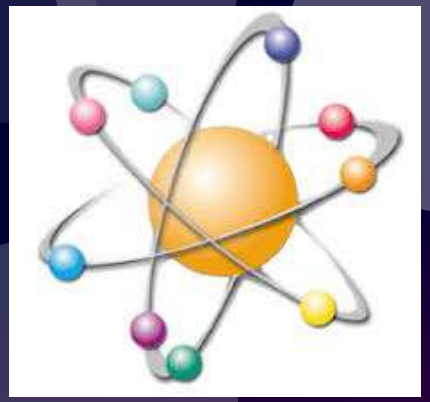
INITIATION BLOCKCHAIN

Présentation



- Master Phys. Fond.
- Master 2 Traitement du Signal
- Web child
- Ingé. Études et dev. SI/Web2

Présentation

- Intérêt personnel Crypto 2016
- Alyra 2021-2022
- Dev Web3/Blockchain

Présentation

- Lead dev backend YesOrNo
- App mobile Web3
- Betting Web3

Présentation

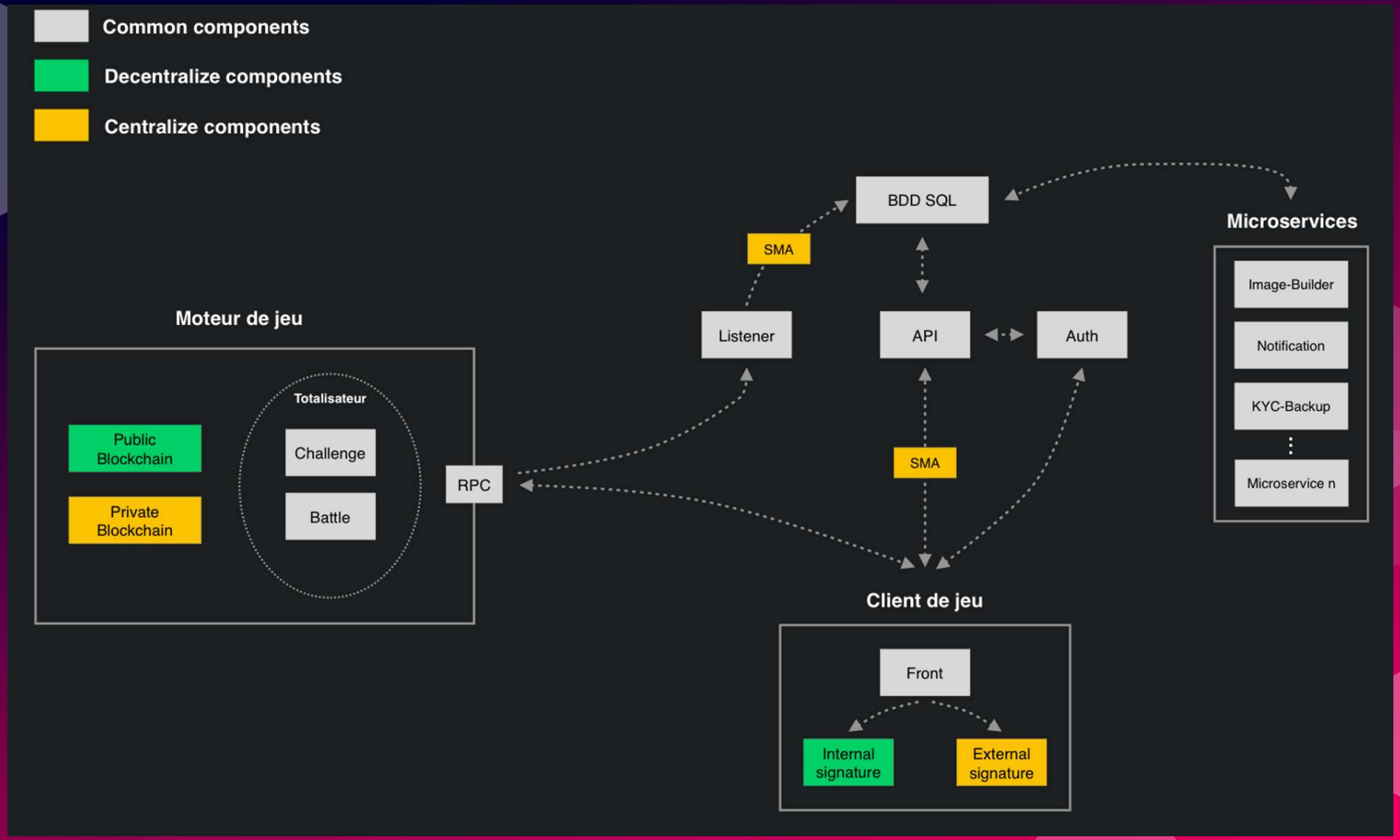
Stack technique

- BC : Polygon
- Front : Flutter
- Back : NestJs
- Infra, Auth, Storage : GCP
- DB : Mysql

Présentation

Stack technique

- BC : Polygon
- Front : Flutter
- Back : NestJs
- Infra, Auth, Storage : GCP
- DB : Mysql

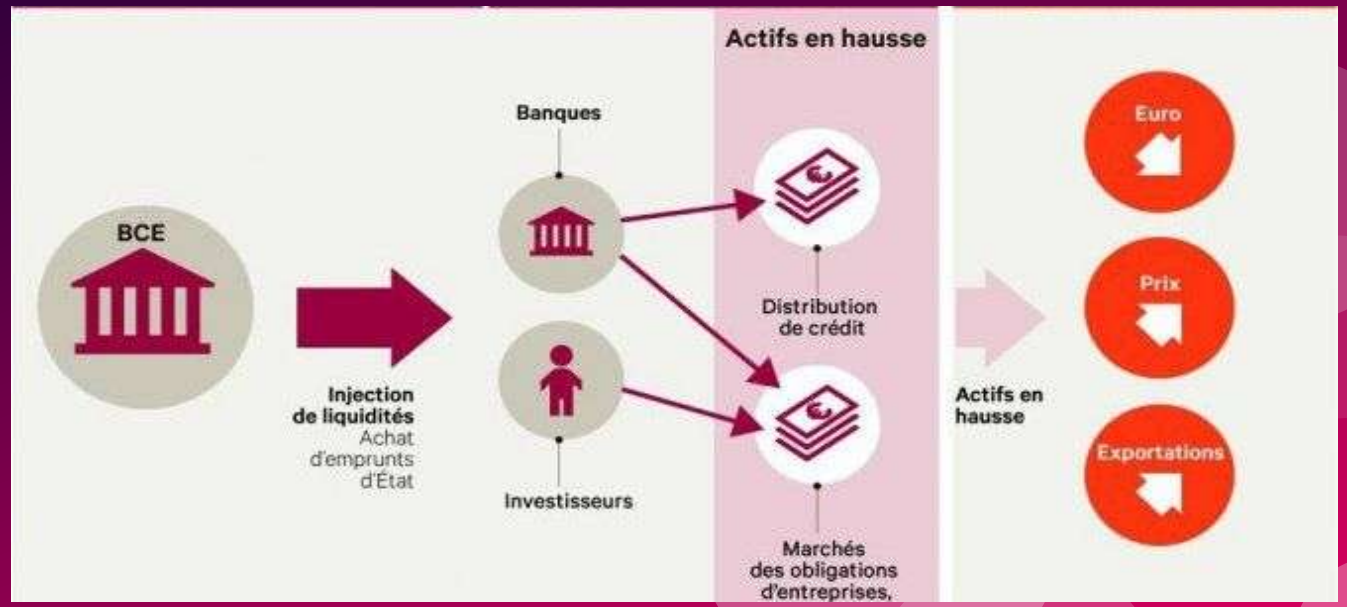


CONTEXTE

CONTEXTE

- Economie florissante depuis 2002-2003
- Croissance tirée par le haut par l'Asie notamment Chine (PIB à 2 chiffres / an)
- Politique facilitante envers les banques
- Injection monétaire massive (QE)
- Impression monétaire ex nihilo

Et puis ...



CONTEXTE

… en 2007-2008

- Accord massif de prêt immobilier US
- Foyers revenus faibles
- Taux intérêt variable

- Bulle spéculative
- Hausse du prix de l’immo.
- Hausse des taux intérêt
- Défaut de paiement

Crise des subprimes 2007-2008

15 Septembre 2008

- Panique inédite (1929)
- Propagation Reste du monde
- Injection monétaire BC

CONTEXTE

On a perdu quelque chose …. ??

CONTEXTE

La confiance aux entités centralisées

Naissance d’un mouvement libertarien (11/2008)

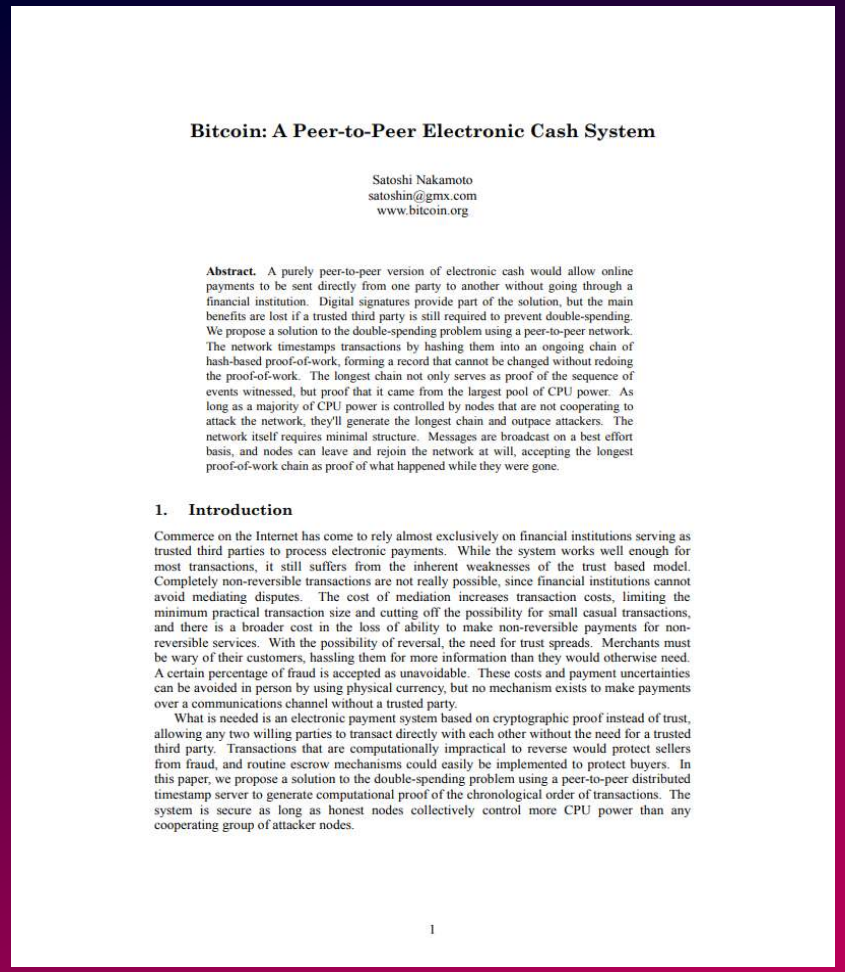
Satoshi Nakamoto

- Système de transfert d’argent peer-2-peer
- Sans tiers de confiance
- Anonymat des pairs
- Emission de nouvelles « pièces »

Naissance de la blockchain (11/2008)

- White paper : https://bitcoin.org/bitcoin.pdf
- Double dépense ??
- Certitude des transferts
- Inscription des transactions dans une chaîne de blocs

Naissance de la blockchain (11/2008) Whitepaper



Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

1

Genesis Block : Janvier 2009

- « *Times 03/Jan/2009 : Chancellor on brink of second bailout for banks* »
- Volonté de contre-pouvoir
- 12/12/2010 ⇒ No more Satoshi

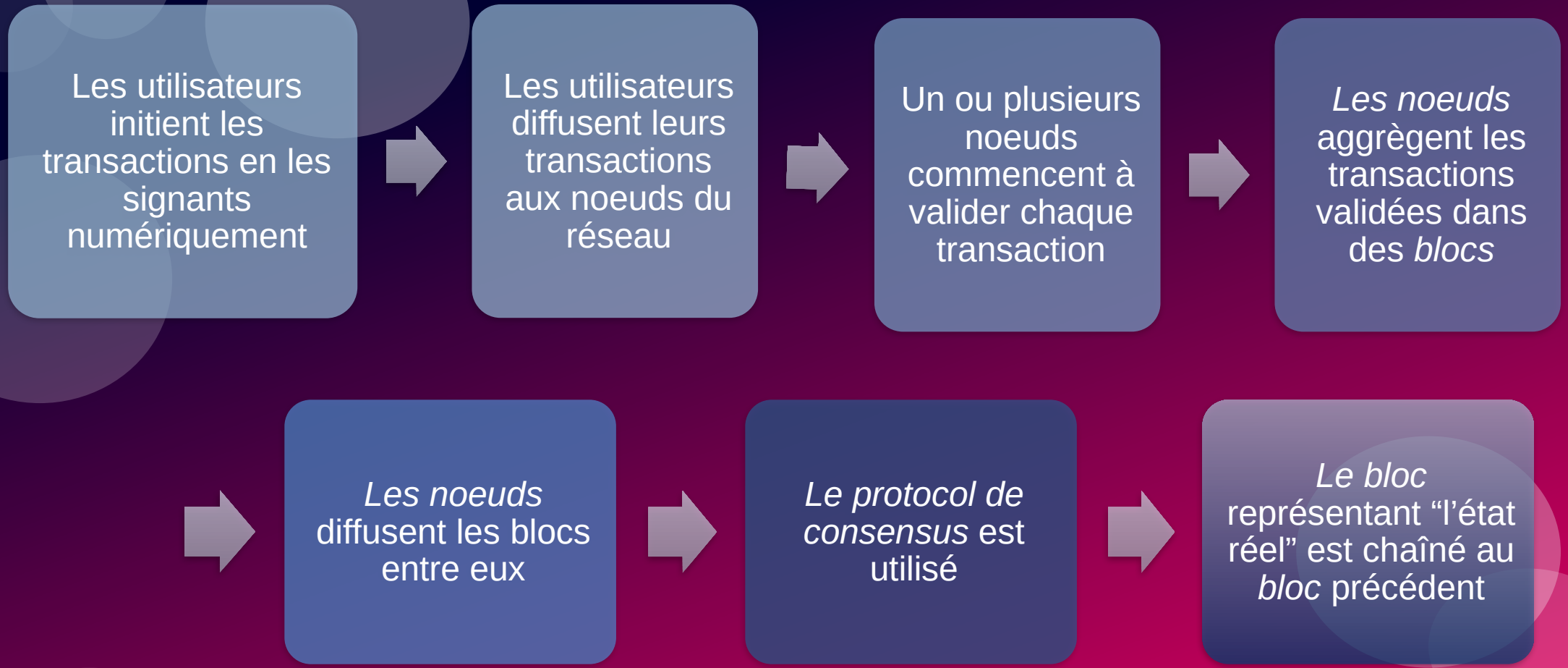
Succès rapide

Fonctionnement d’une blockchain

- Registre distribué sur l’ensemble du réseau
- Information répliquée et identique sur chaque nœud du réseau
- Consensus pour s’accorder sur les données inscrites
- Immutabilité des données
- Certitude de l’émetteur et récepteur
- Aucune autorité centrale ne régit le réseau

- Les transactions sont inscrites dans des blocs
- Reliés les uns aux autres
- « BlockChain »
- Chaque Tx est vérifiable, horodatée et infalsifiable

Blockchain : schéma workflow généraliste



Eléments de cryptographie (fonctions de hashage, signatures numériques)

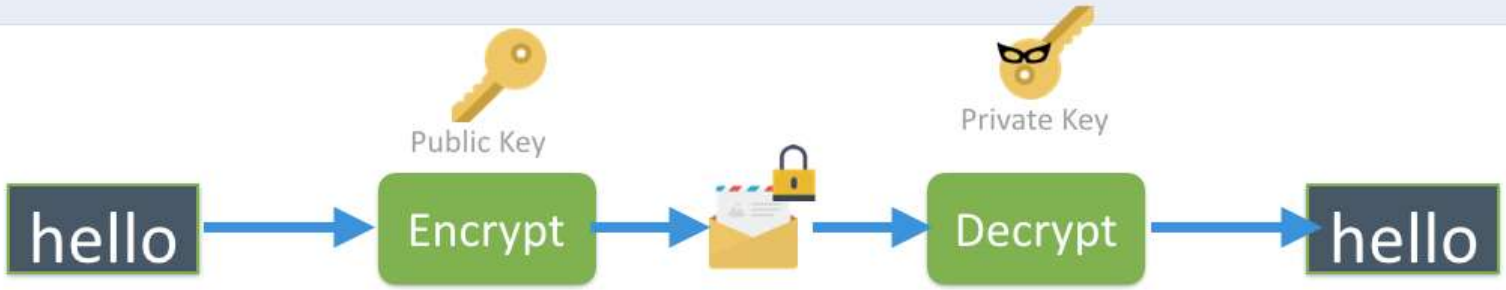
- Cryptographie :

  Chiffrement et déchiffrement des données

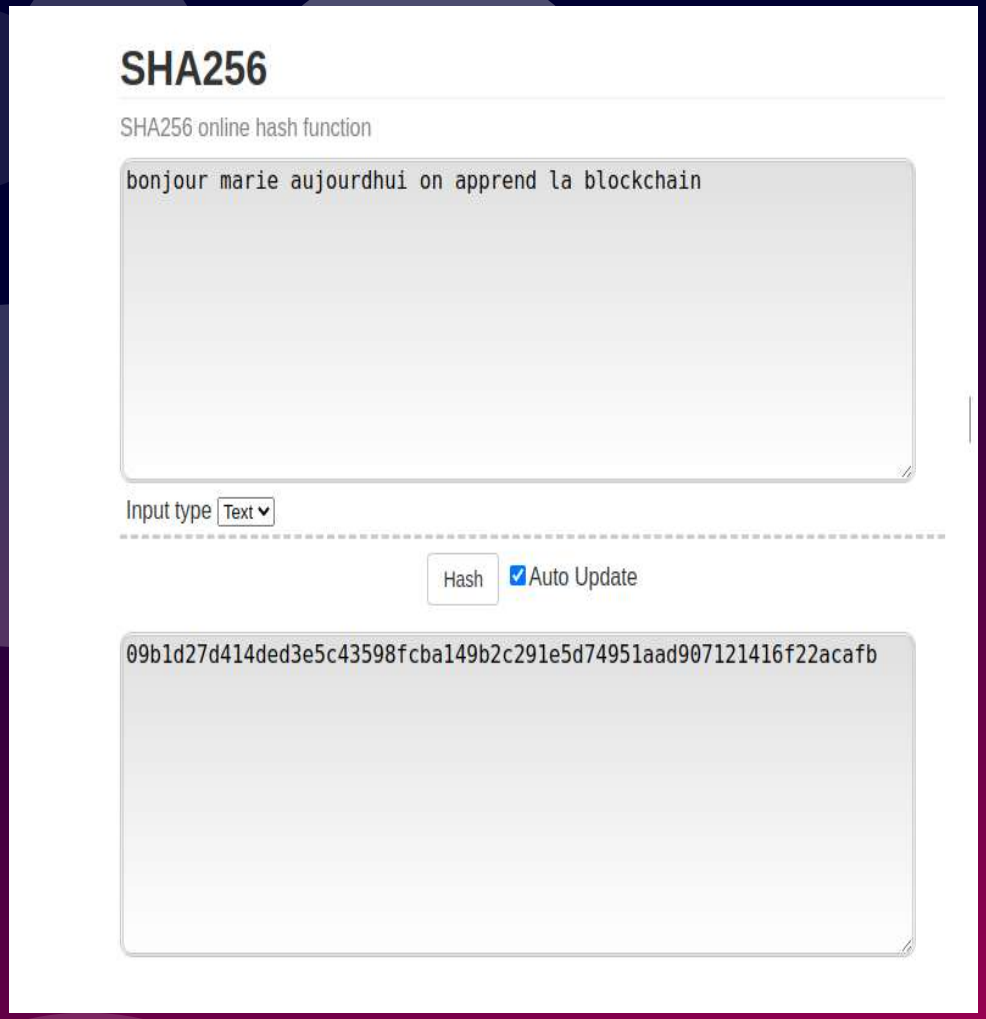
Eléments de cryptographie

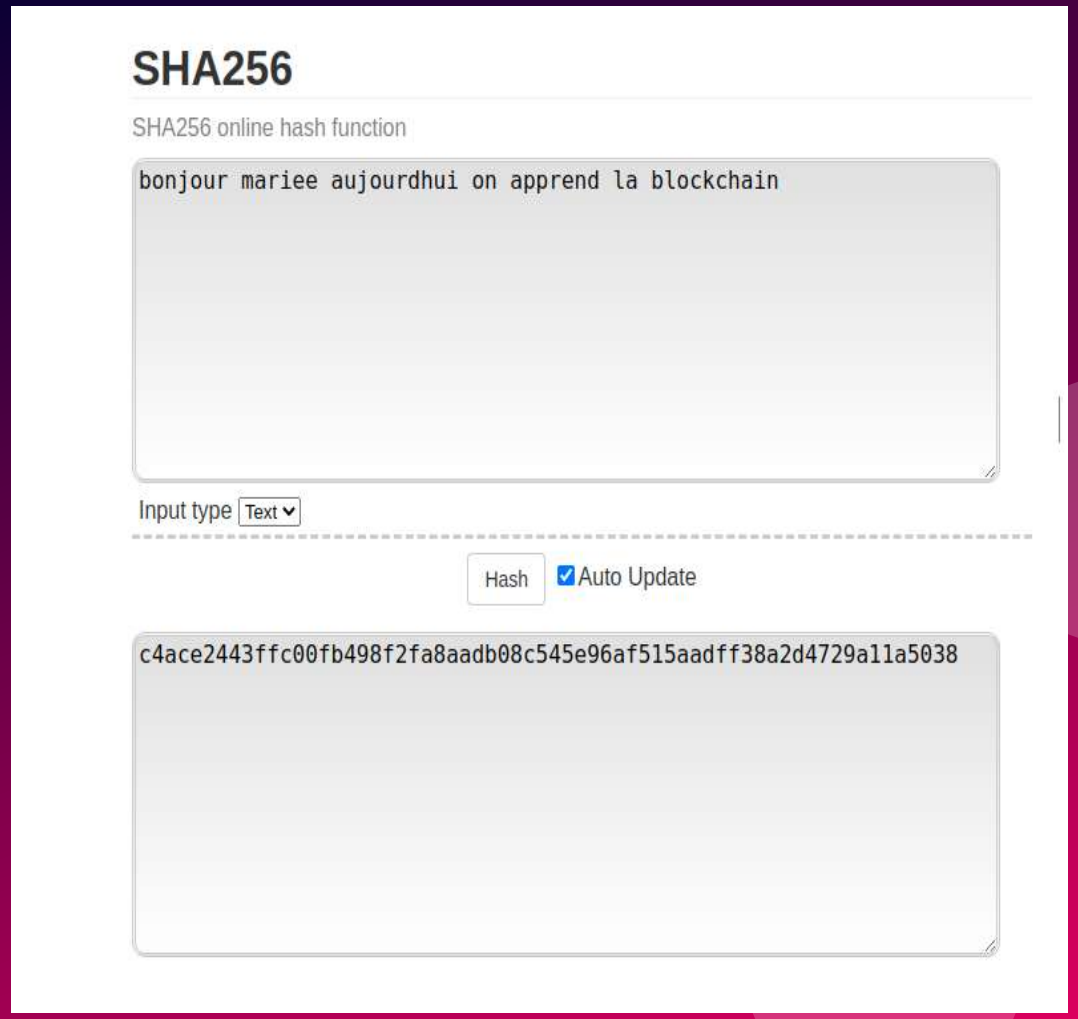
Cryptography symétrique	Cryptography asymétrique	Hashage
Le même mot de passe sert à chiffrer et déchiffrer les données	Un mot de passe sert à chiffrer les données, un autre à les déchiffrer	Projette vers un espace de dimension fixe
Fonction bidirectionnelle	Les mots de passe vont par pair (clé publique / clé privée)	Fonction monodirectionnelle

Eléments de cryptographie

	Description
Hash	
Symmetric Encryption	
Asymmetric Encryption	

Eléments de cryptographie (SHA 256)





Eléments de cryptographie (SHA 256)

Propriétés

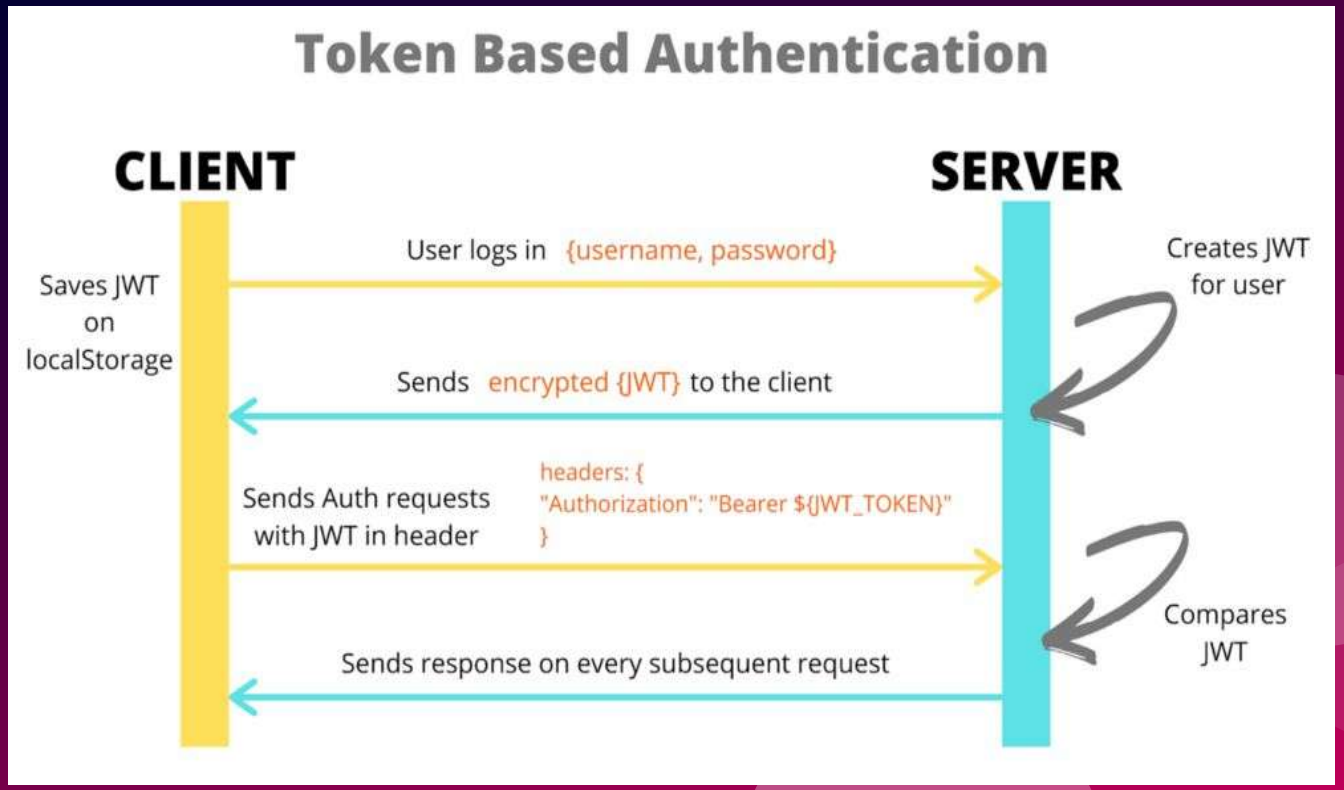
- Déterministe
- Sortie = 256 bits
- Facile à encoder
- Très difficile à décoder
- Unidirectionnel
- Même entrée = même sortie
- Non inversable = diff. de Cryptage

Eléments de cryptographie (Use Case IRL)

A vous de jouer ...

Eléments de cryptographie (Sécurité Web)

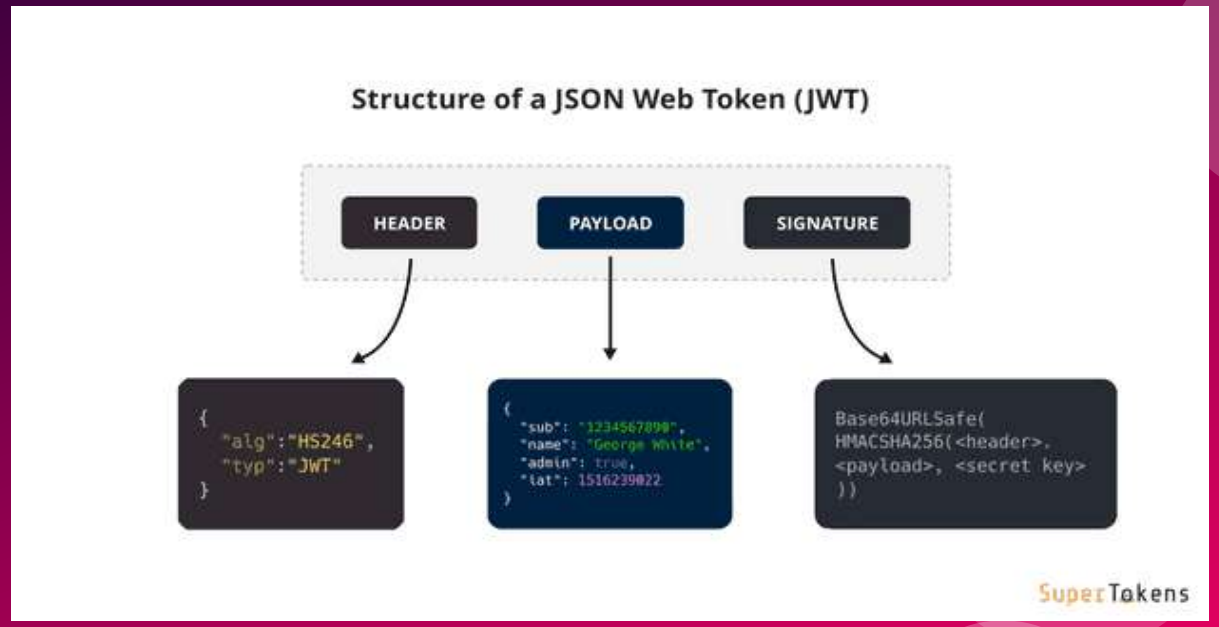
JWT



Eléments de cryptographie (Sécurité Web)

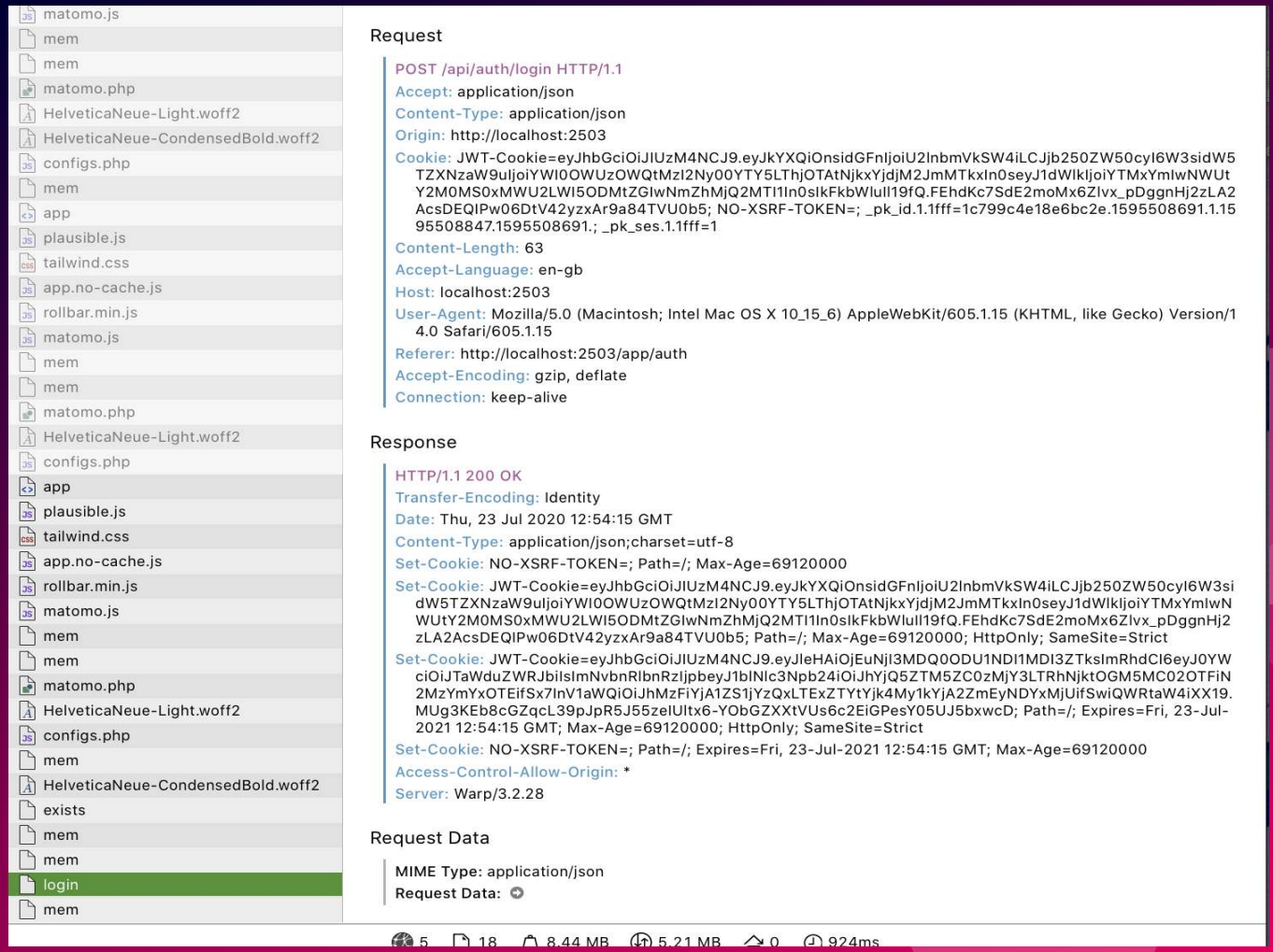
JWT

- Partage infos entre 2 entités : client et serveur
- JWT utilise la cryptographie et est généré par le serveur
- Signé à l'aide d'une clé privée



Eléments de cryptographie (Sécurité Web)

JWT



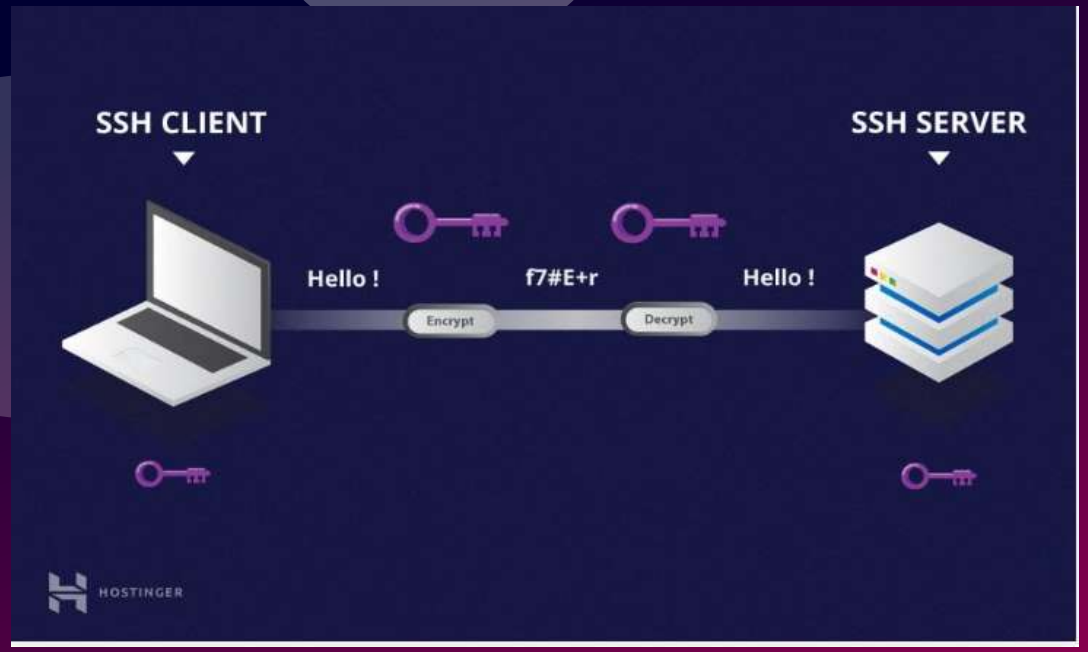
Eléments de cryptographie (Sécurité Web)

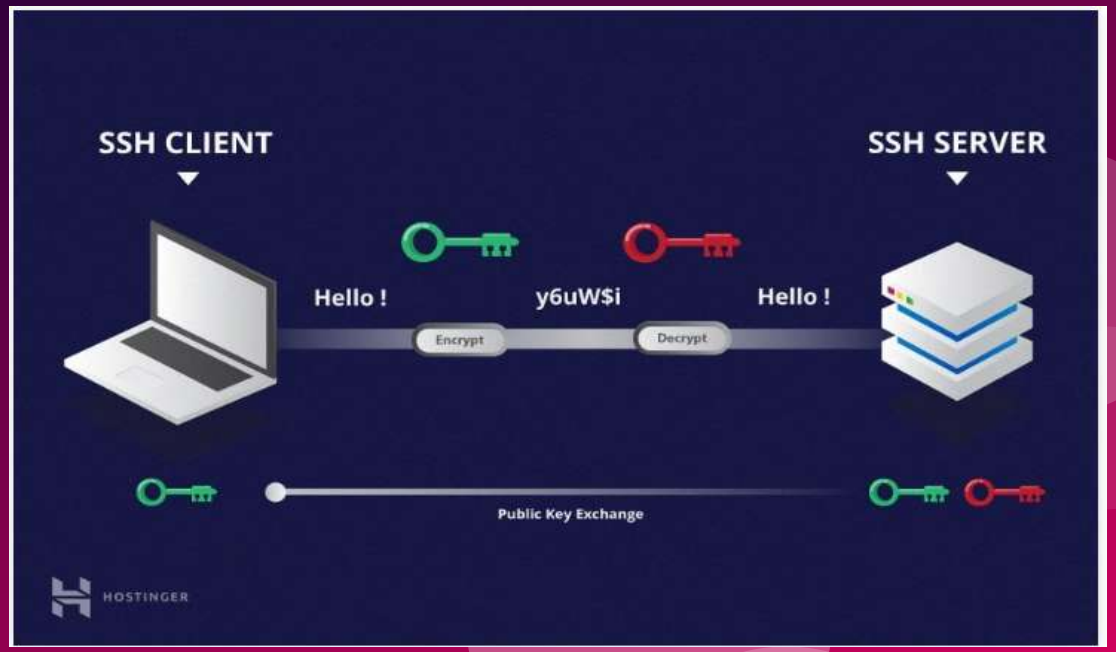
JWT

https://supertokens.com/blog/what-is-jwt

Eléments de cryptographie (SSH)

```
ssh {user}@{host}
```





https://www.hostinger.fr/tutoriels/ssh-linux

Eléments de cryptographie (Messagerie)

Telegram

Eléments de cryptographie (Hack)

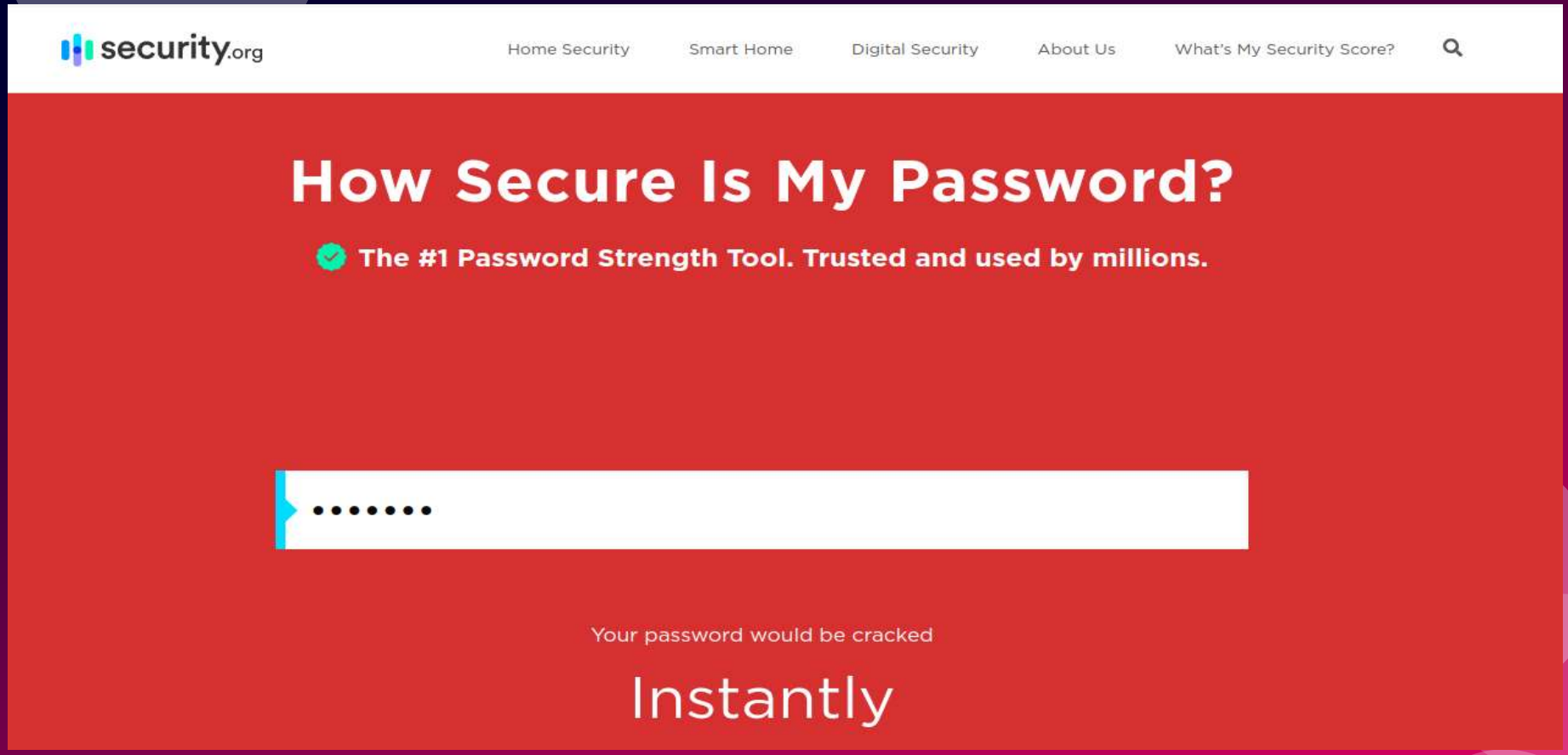
- Fonctions de hachage ==> Unidirectionnel
- Possibilité de hack par *brute force*
- Tables de mots/hachage ⇒ Correspondance

- Génération Clé publique/Clé privée avecune chaîne de caractères aléatoire
- 2^256 clés privées possibles (> nb atomes dans l'univers)
- Introduction de la phrase mnémonique (24 mots) en 2013 avec BIP 39

Eléments de cryptographie (Hack)

https://www.security.org/how-secure-is-my-password/

« bonjour »



Eléments de cryptographie (Hack)

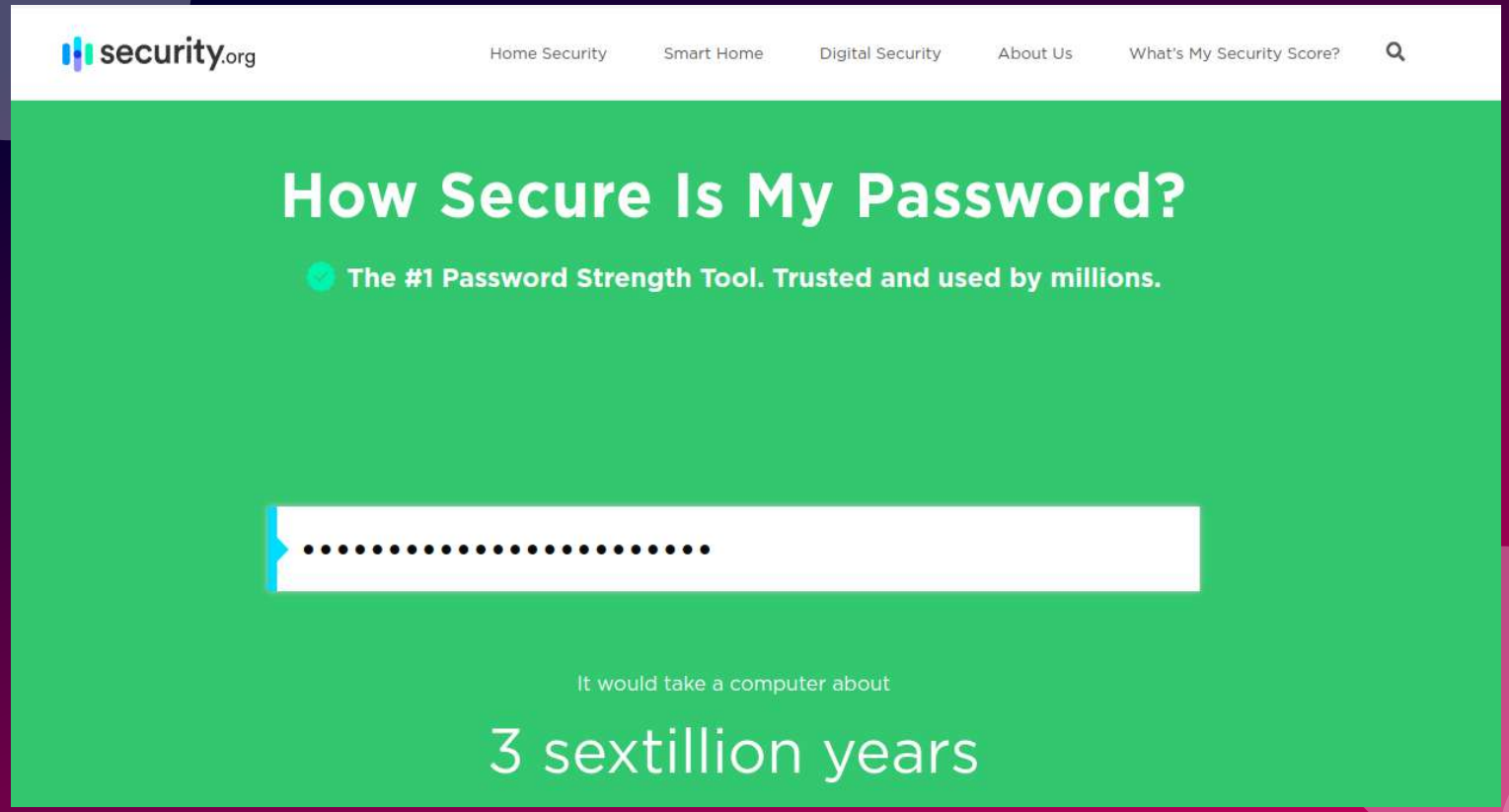
https://www.security.org/how-secure-is-my-password/

« bonjourtoto »

Eléments de cryptographie (Hack)

https://www.security.org/how-secure-is-my-password/

« bonjourtotocommentvastu ? »



1 000 000 000 000 000 000 000 000 000 000 000 000 années

Eléments de cryptographie (quantum computer)

Menace de l’informatique quantique ??

⇒ Informatique classique : 2 états (0 ou 1)
⇒ qubits : infinité d’états (superposition d’états entre 0 et 1)

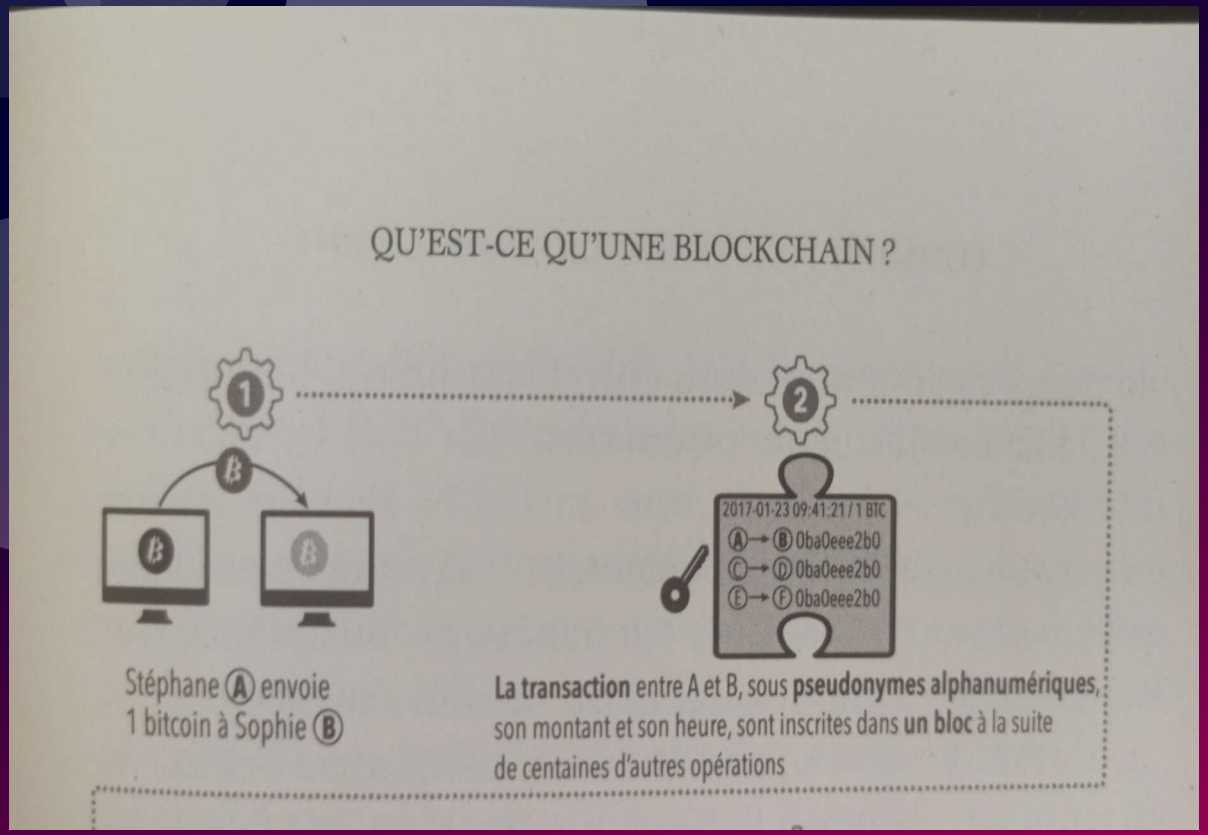
L’ordinateur quantique Borealis de la société canadienne Xanadu
a réalisé un calcul en 36 micro sec.
Un supercalculateur aurait mis 9000 ans

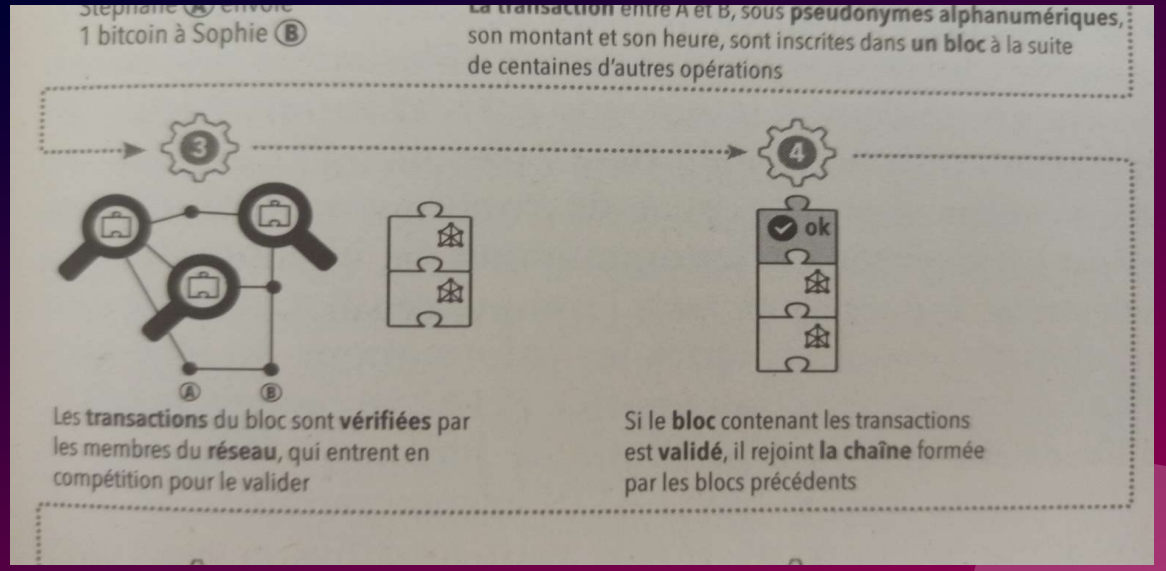
La blockchain pas à pas

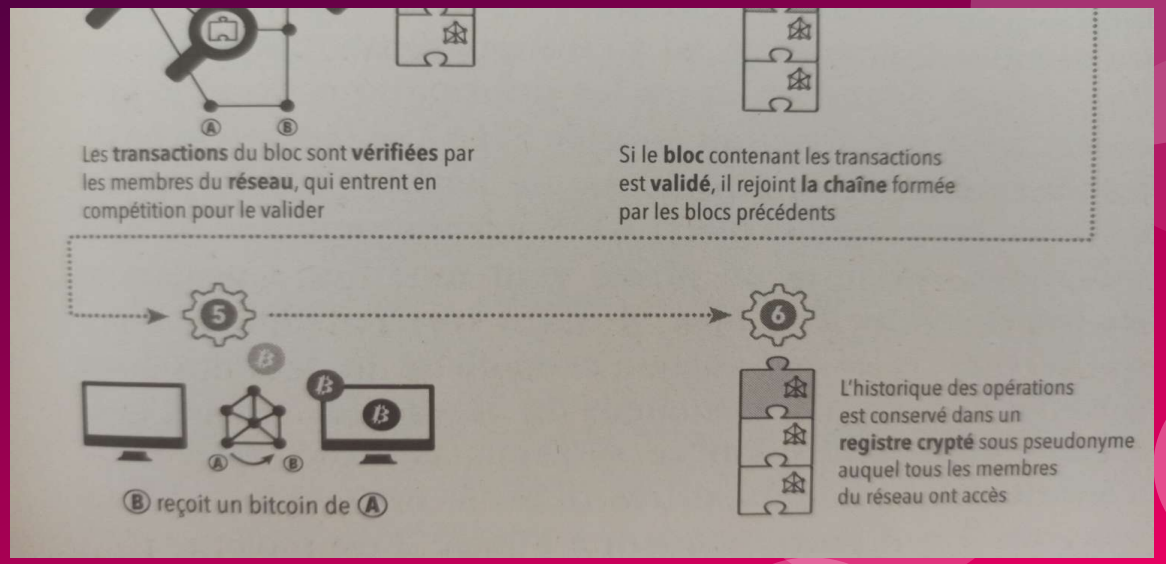
- Stéphane (02017cb6a243de2d0e7014650a36ee2315cdb6ff869794c685b0fe00fa2679741e)
- Sophie (0254abc59904f2a90aac69698f3238be298cfb487c65d44e18a84d6ecd69fb8033)

- Envoi 1 BTC

La blockchain pas à pas

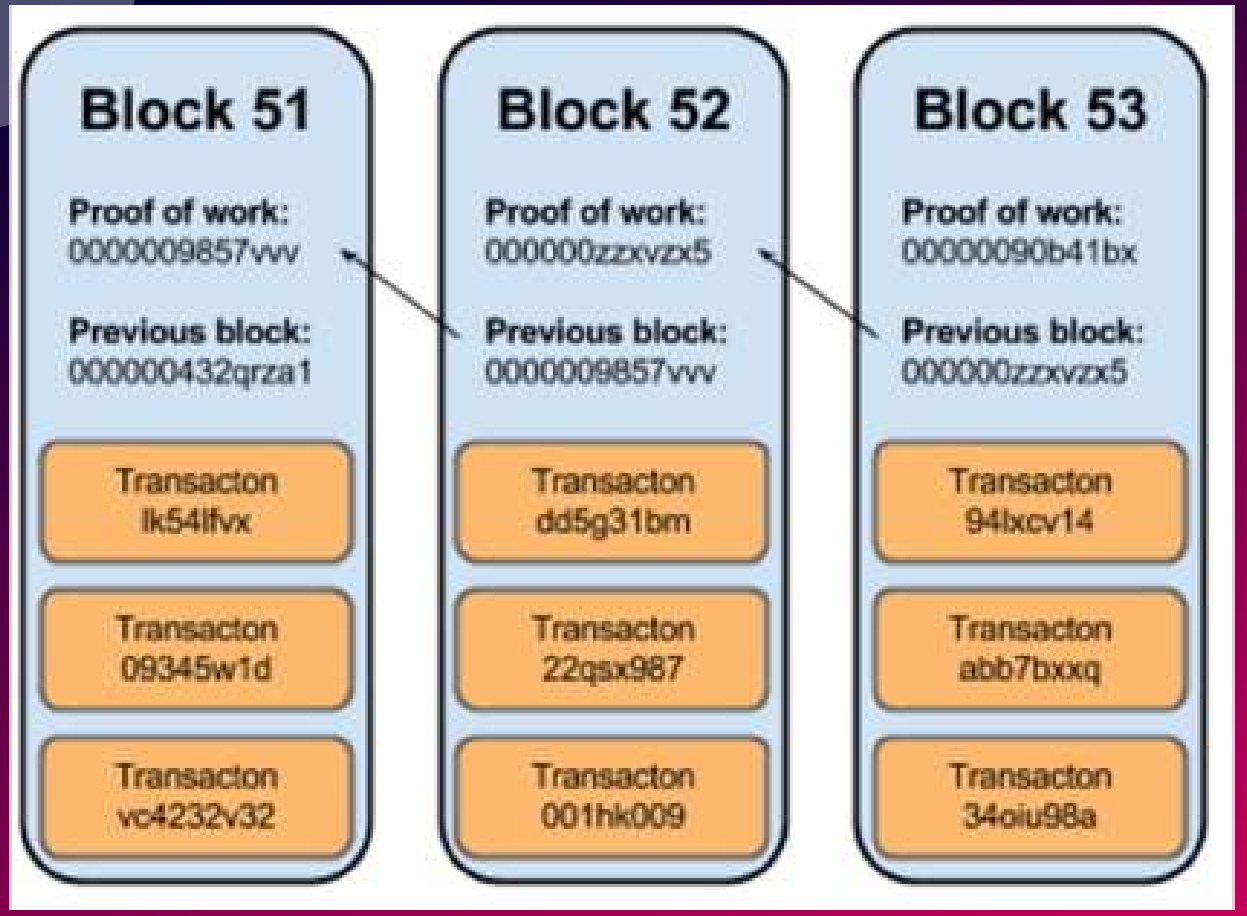






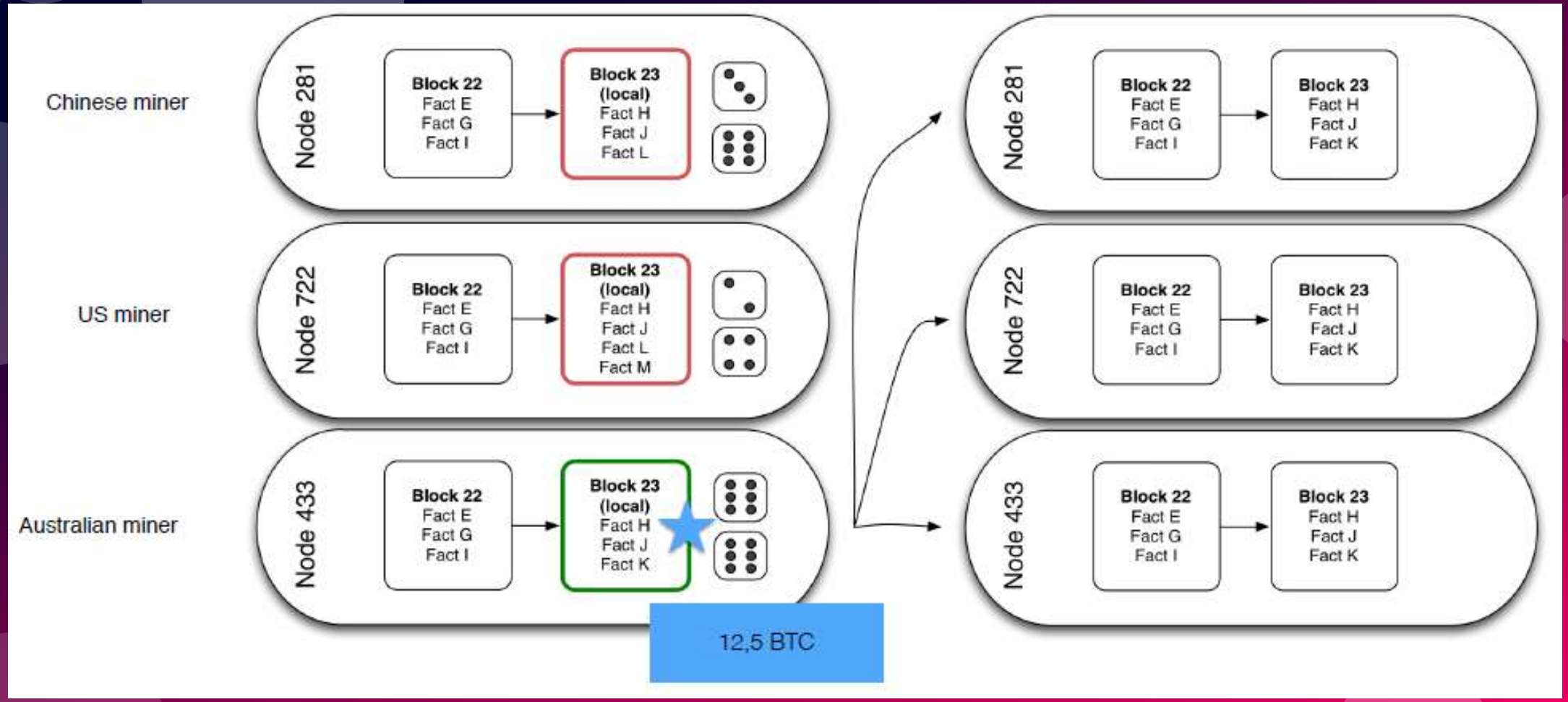
La blockchain pas à pas (Minage)

Minage : processus par lequel les tx sont vérifiées et inscrites dans la BC



La blockchain pas à pas (Minage)

Minage : Check tx, Check fonds émetteur, check double dépense ===> Incitation rémunératrice par résolution d'un pb mathématique avec puissance de calcul ⇒ *Proof of work*



La blockchain pas à pas (Propriétés intrinsèques)

- Décentralisation
- Sécurité
- Transparence
- Consensus

La blockchain pas à pas (Contenu d’un bloc)

BLOC

(Unité de la Blockchain, comme des pages de transactions au sein d’un registre)

Header

(Numéro de version / Hash du bloc précédent / Hash de la racine de l’arbre de Merkle / Time Stamp / Valeur cible du hash / Nonce)

Body

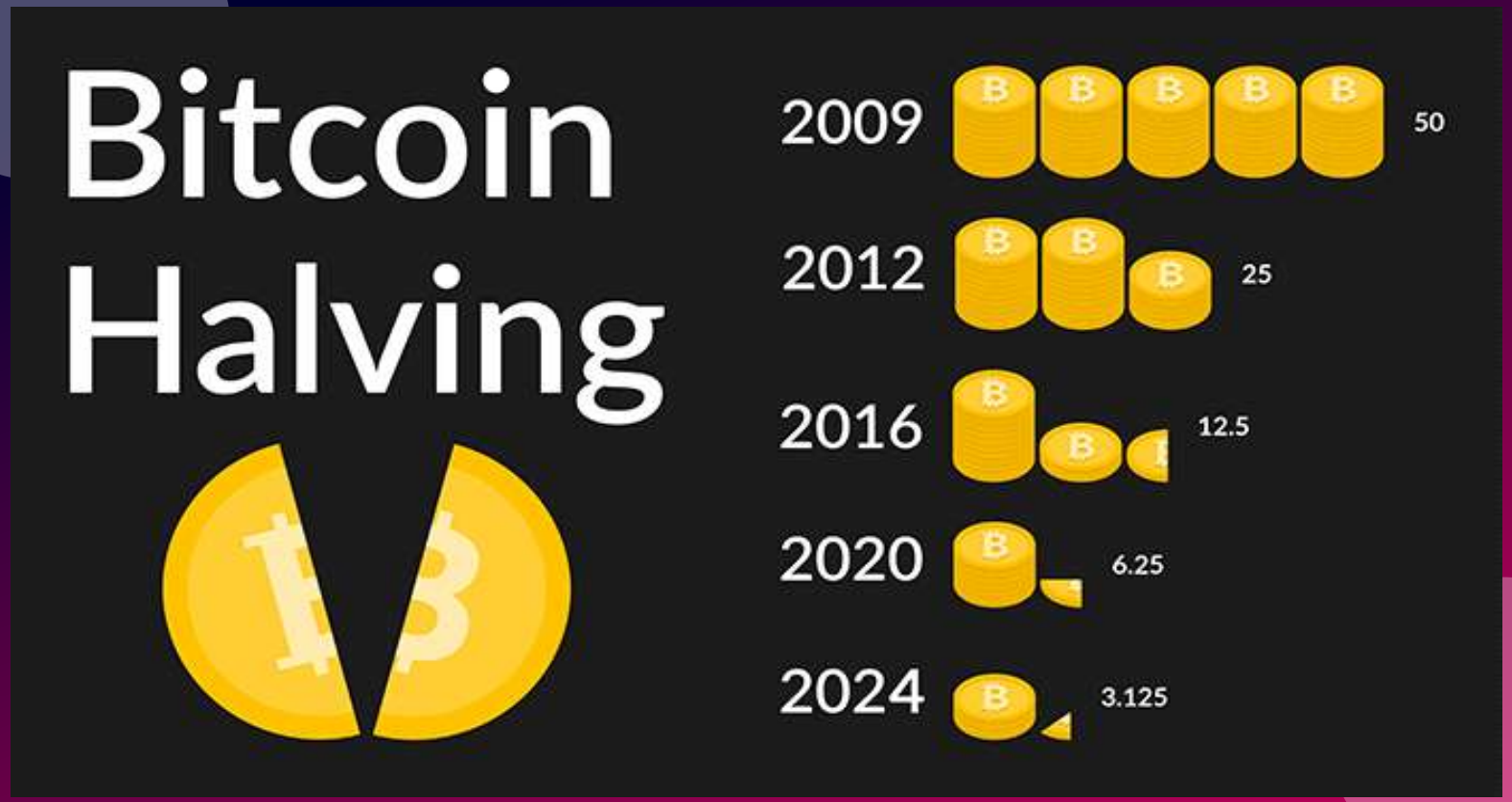
1) Le nombre de transactions et les transactions
2) Récompense par bloc
3) Hauteur du bloc
4) Taille du bloc

La blockchain pas à pas (Explorateur de blockchain)

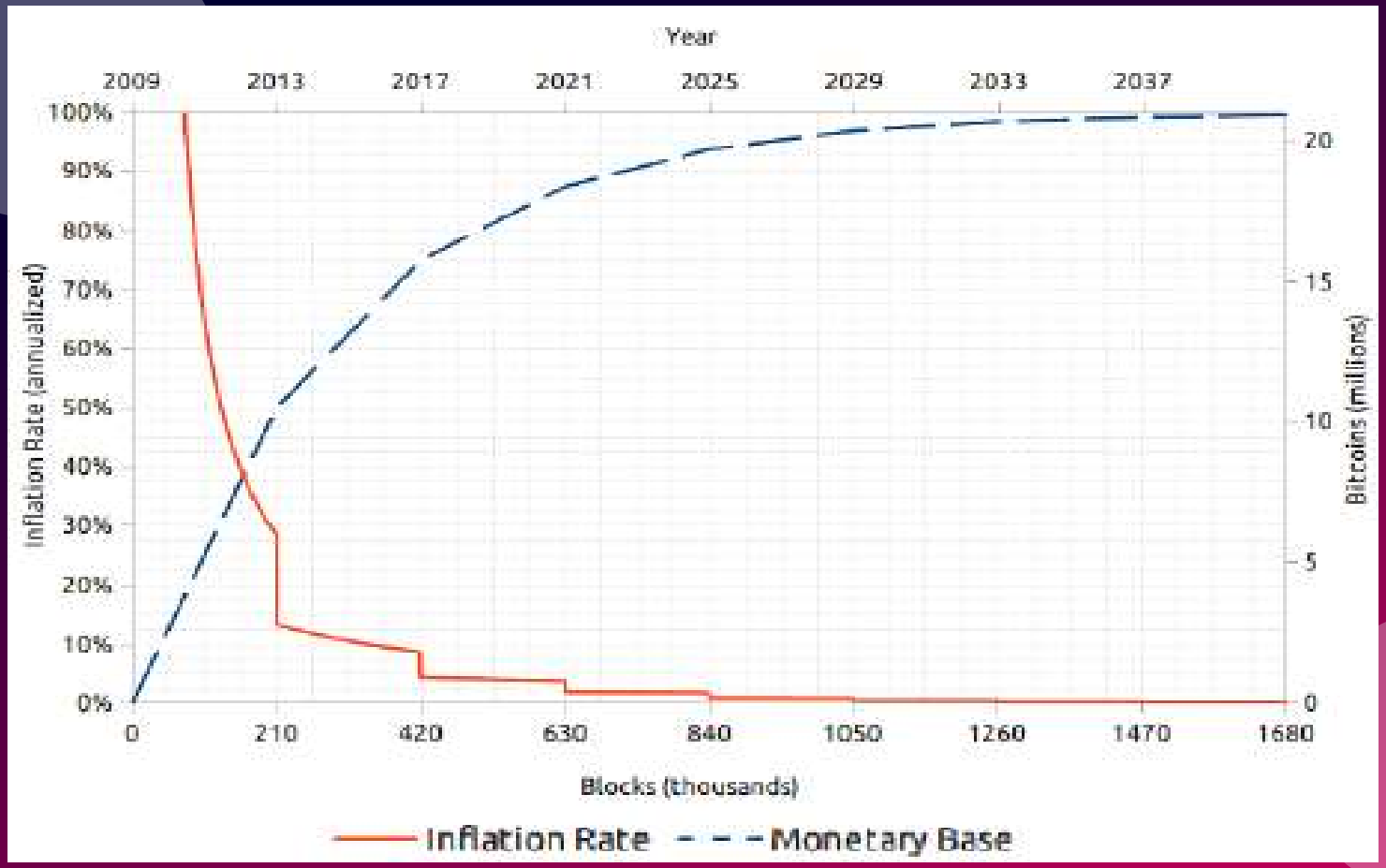
⇒ Blockcypher (https://live.blockcypher.com/btc)

⇒ Blockstream (https://https://blockstream.info/)

La blockchain pas à pas (Halving)



La blockchain pas à pas (Inflation rate)



La blockchain pas à pas (Inflation rate)

Bitcoin : conclusion

Bitcoin : La concurrence

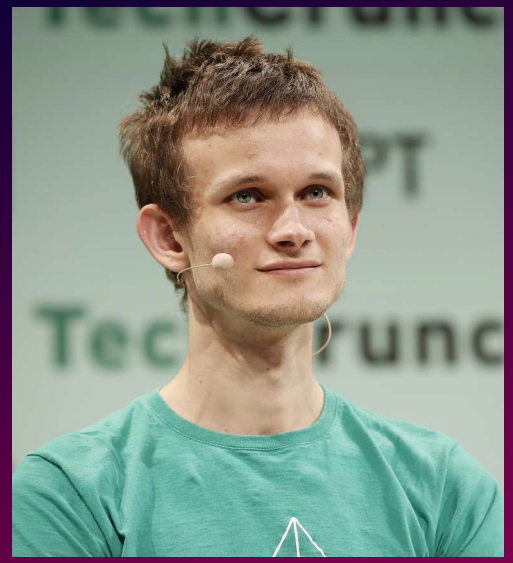
Bitcoin : La concurrence

Bitcoin : Limites

- Cas d’usage limités
- Sécurité VS simplicité

Ethereum : les origines

Mettre en valeur la puissance de la BC via une protocole « Proof Of Stake »



Vitalik Buterin

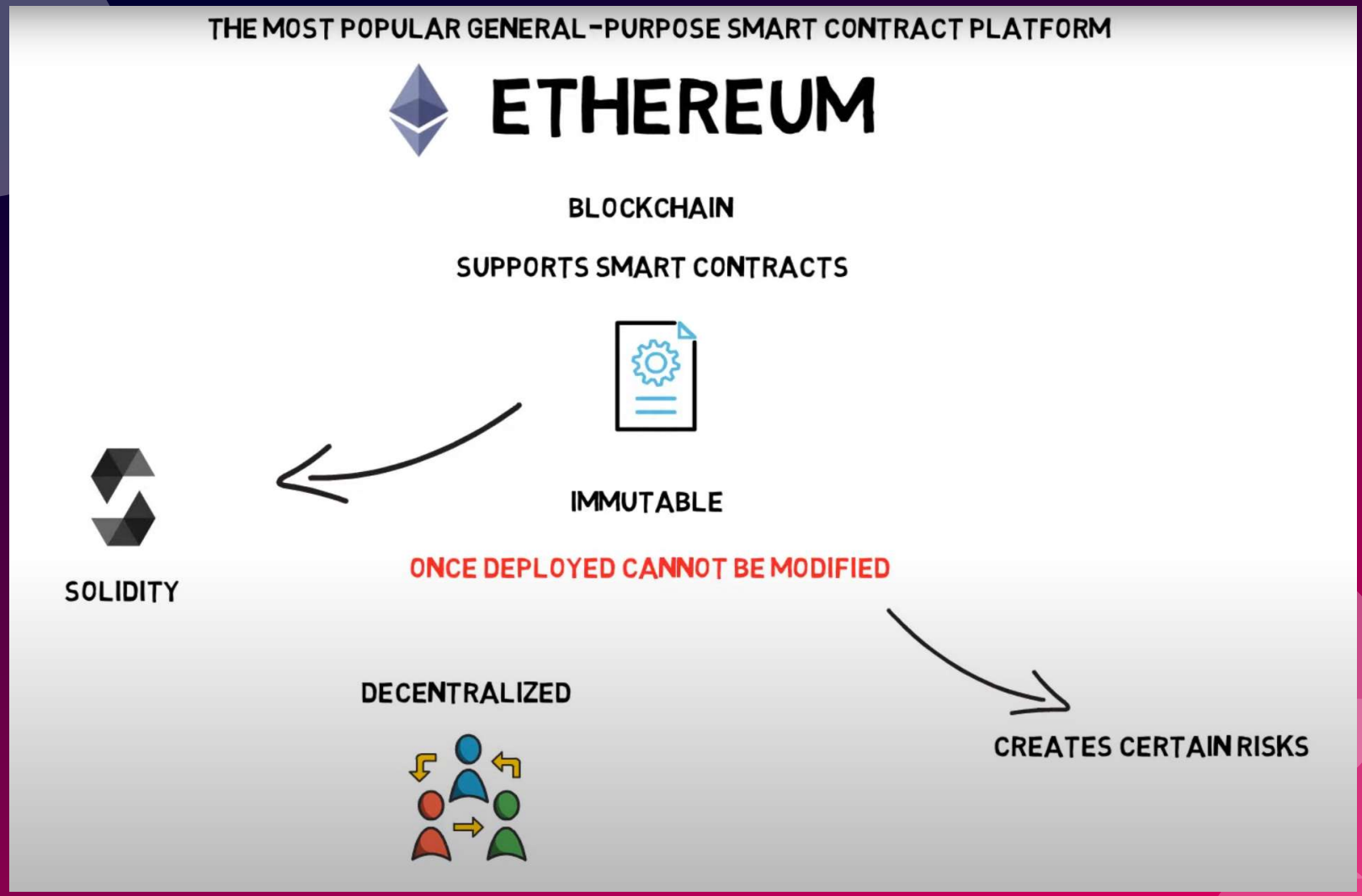
Ethereum : les origines

- Proposer la création de *smart contracts*
- Executer programmatiquement des TX qui s’apparentent aux clauses d’un contrat
- Automatiser les executions sans autorité tierce

- Fees de TX avec crypto native ETH
- Permet + de Use case : Financement participatif, prêt/emprunt
- Gavin Wood
- Launch Date : janvier 2014
- Genesis bloc : 30/07/2015

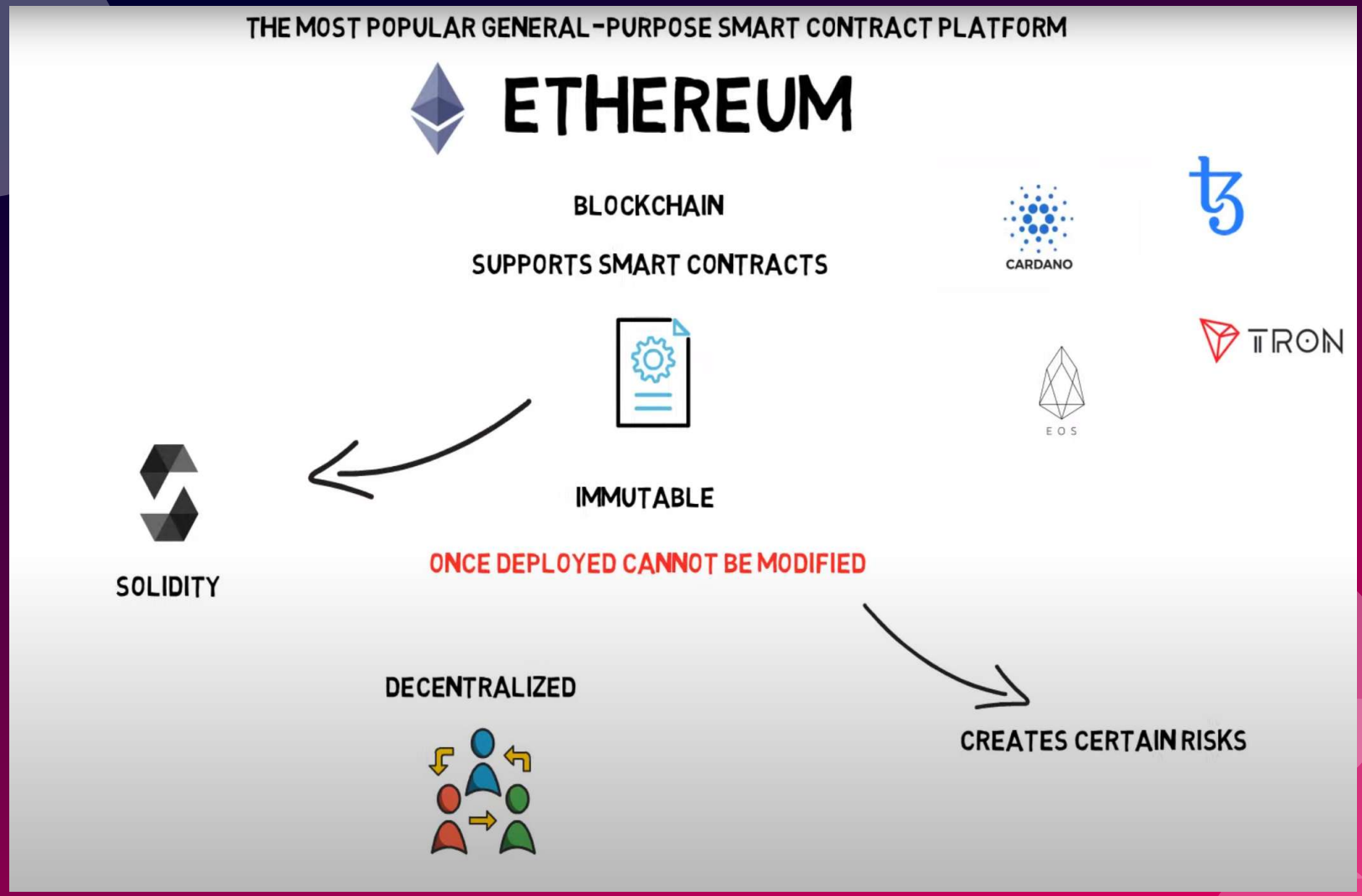
Ethereum : smart contract

Code is law ?



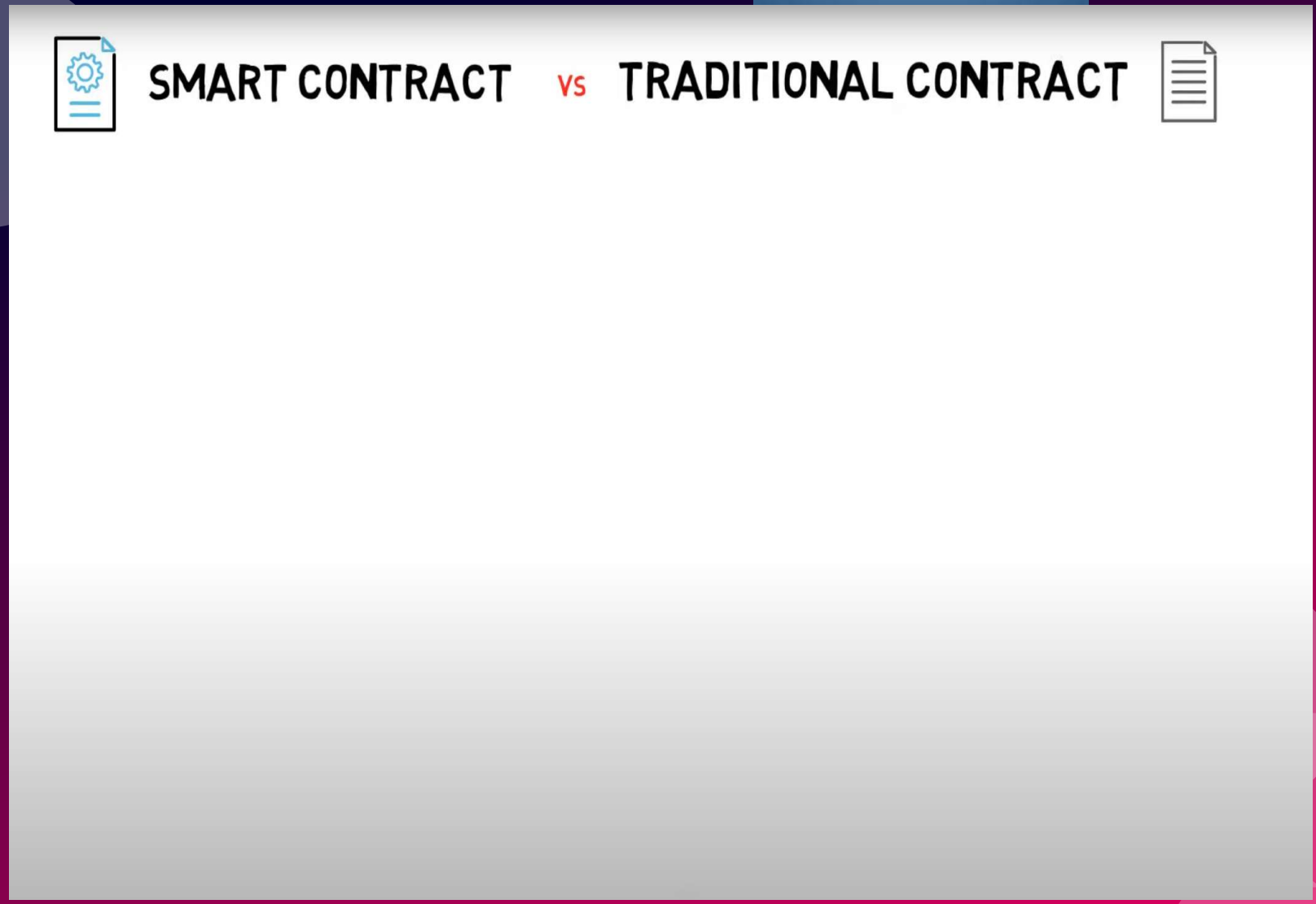
Ethereum : smart contract

Code is law ?



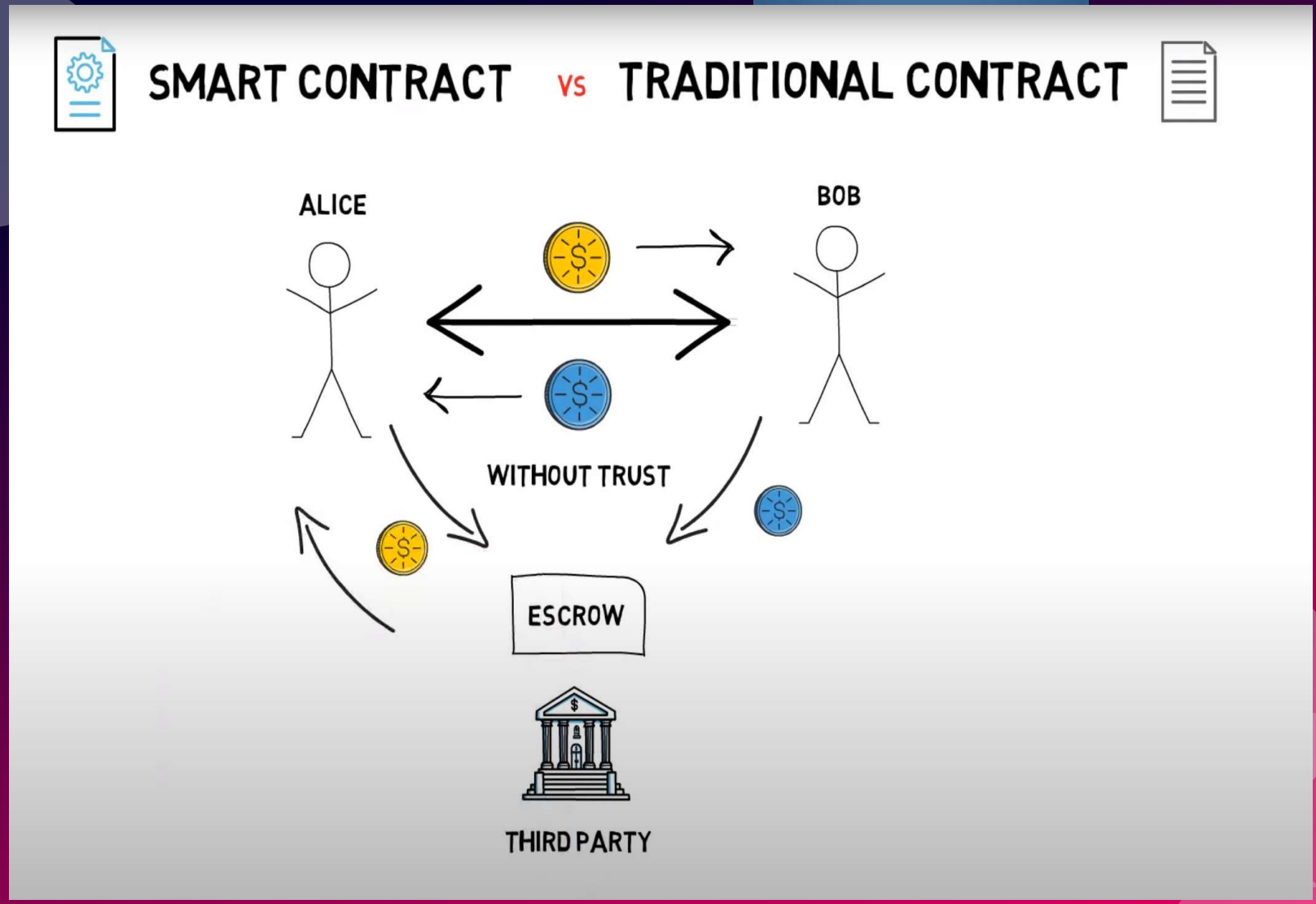
Ethereum : smart contract

Code is law ?



Ethereum : smart contract

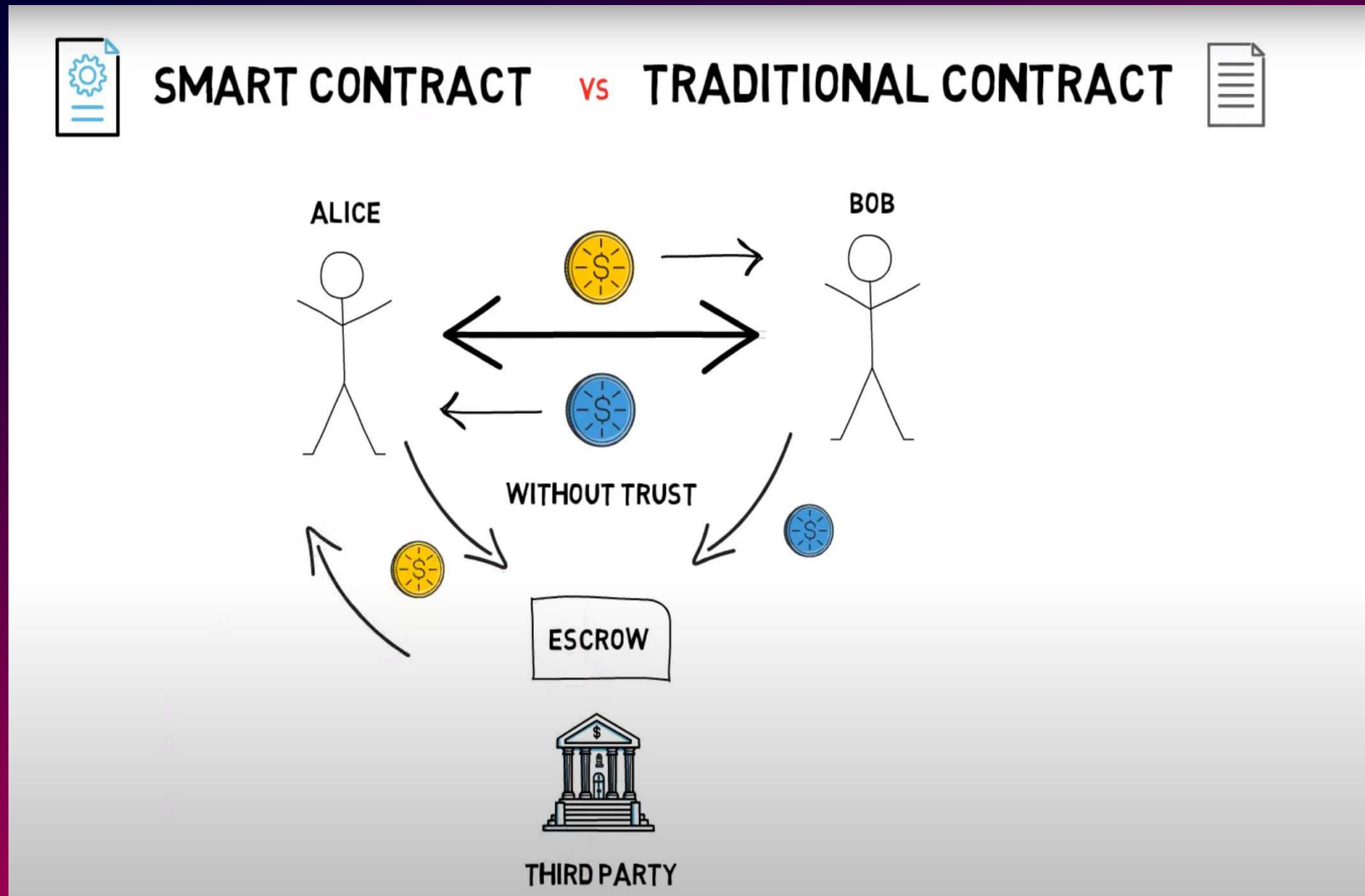
Code is law ?



Ethereum : smart contract

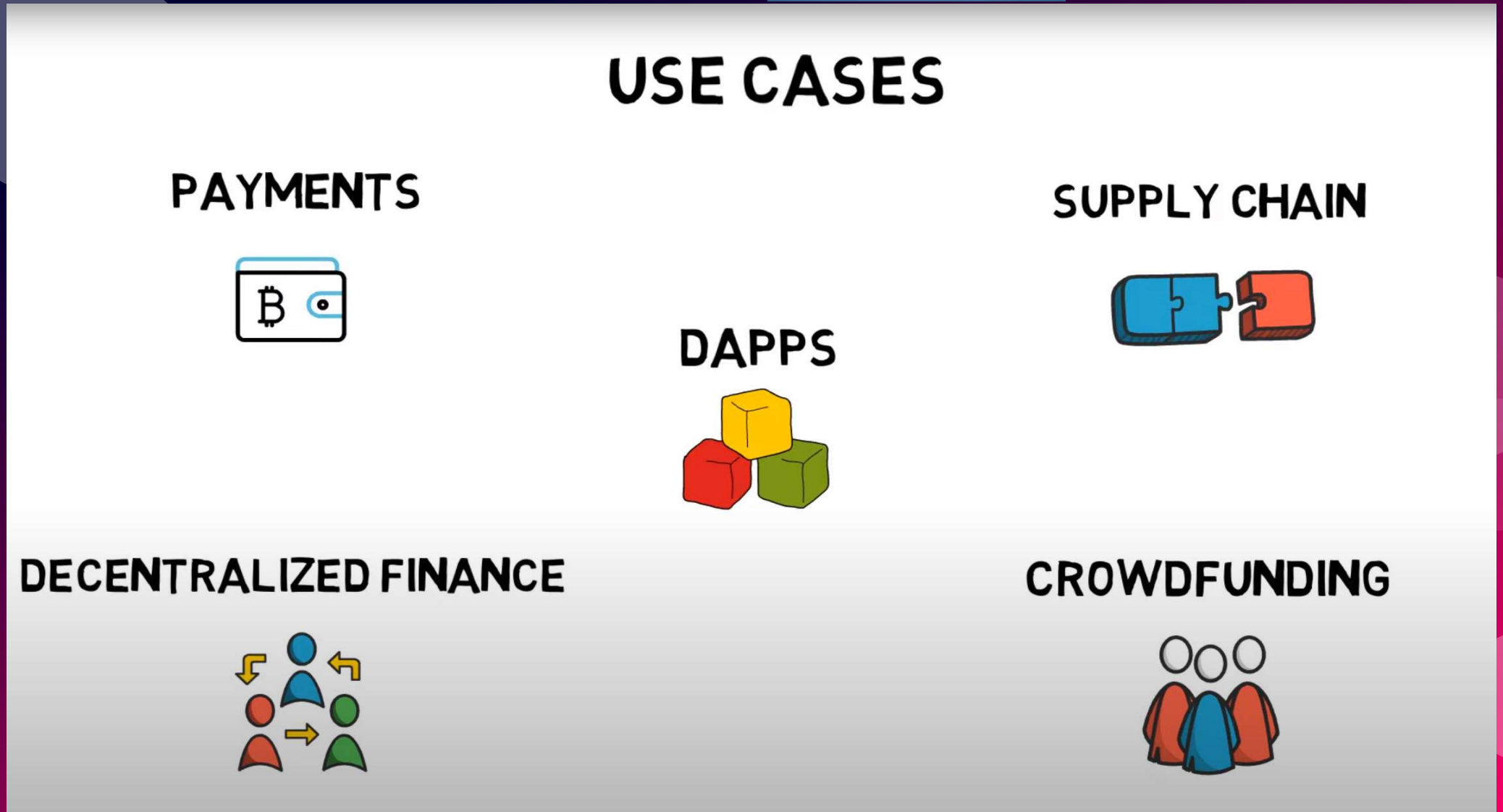
- Pb approche tradi
- Confiance en l'intermédiaire
- Réputation de l'intermédiaire
- Not deterministic
- Lenteur tx (⇒ fermé le dimanche!)
- Coût élevé (risque de coûts cachés)
- No reusability
- Huge team dedicated to check scammers (=expensive)

- Deterministic
- Totalement automatisé
- Rapidité tx
- Coût faible et transparent
- Reusability
- Scam is Impossible



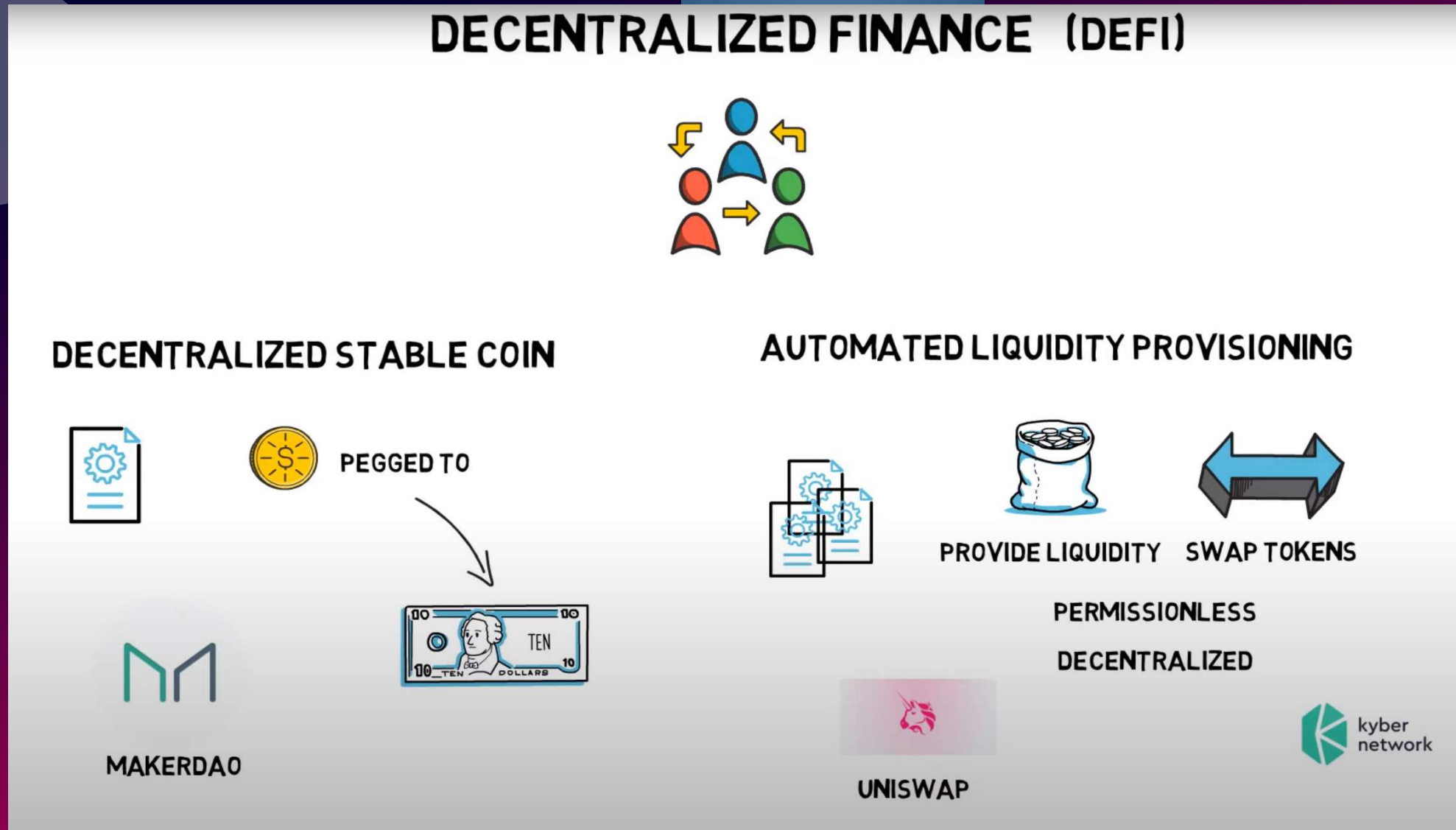
Ethereum : smart contract

Use Case



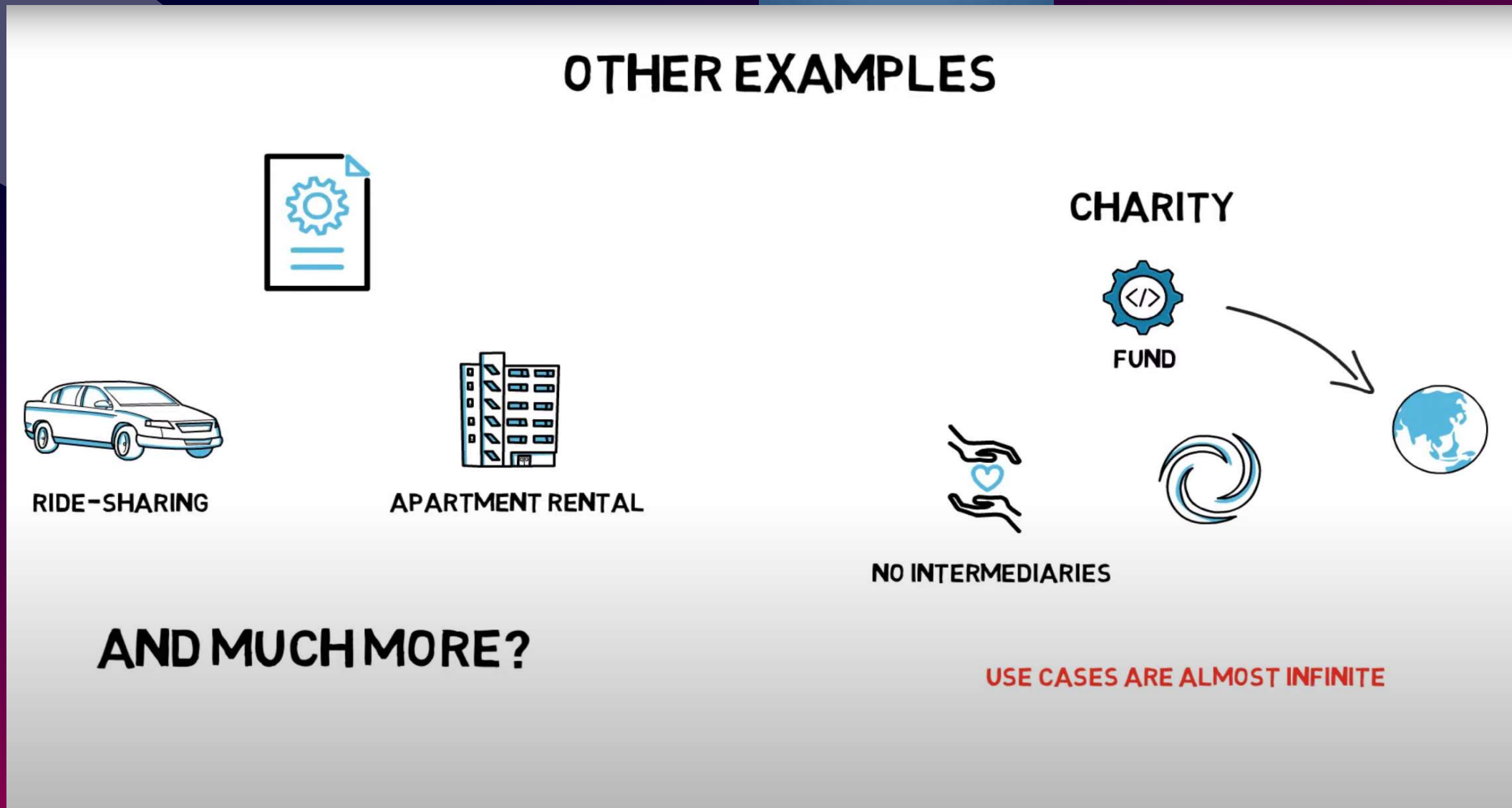
Ethereum : smart contract

Use Case



Ethereum : smart contract

Use Case

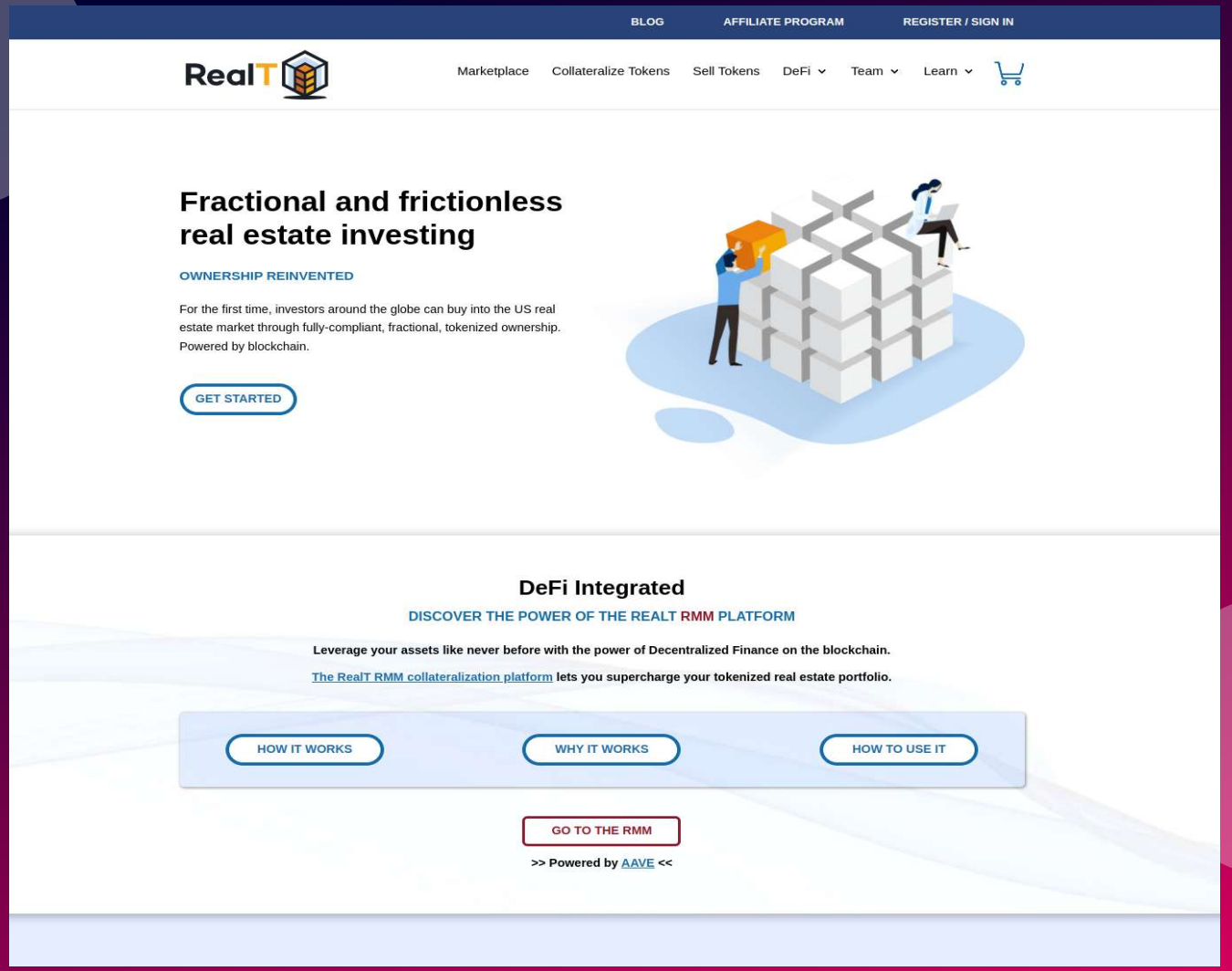


Ethereum : smart contract

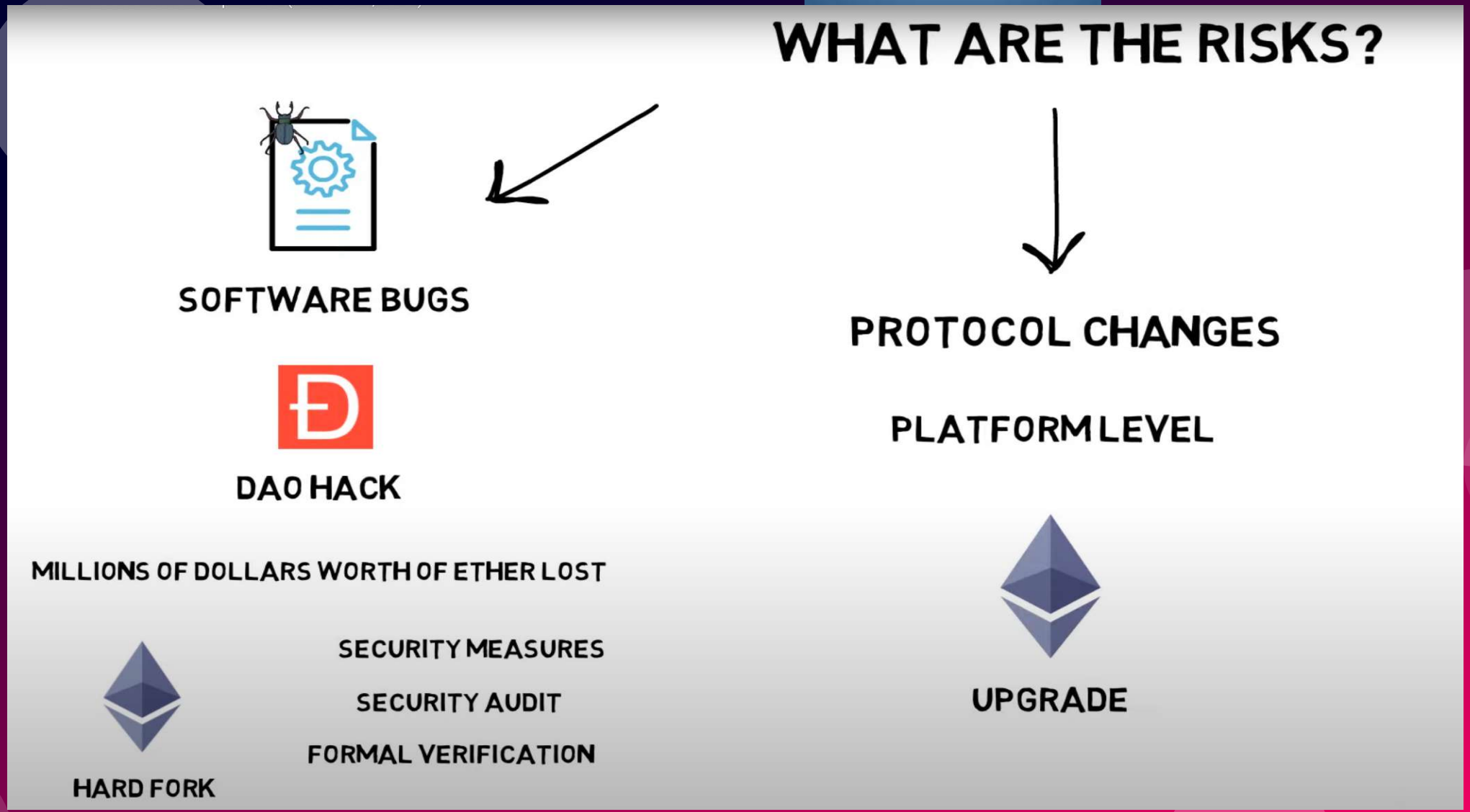
Exemple concret : https://realt.co/

Leader dans la tokenisation de l’immobilier

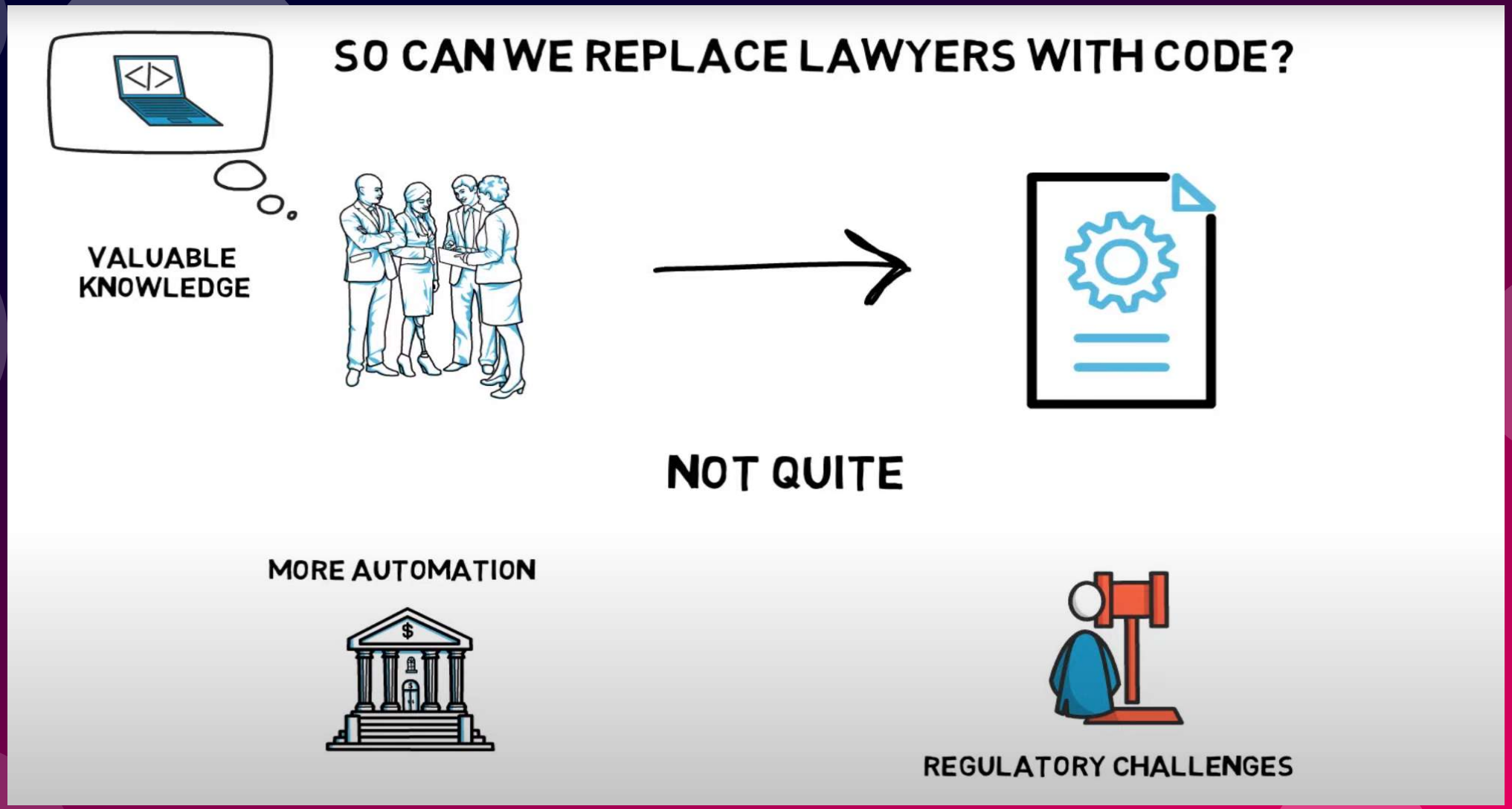
Ethereum : smart contract



Ethereum : smart contract



Ethereum : smart contract



Ethereum : smart contract

QUICK SUMMARY

PROS

FULLY AUTOMATED

DETERMINISTIC RESULTS

TRUSTLESS

FAST

PRECISE

SECURE

COST EFFICIENT

TRANSPARENT

CONS

SOFTWARE BUGS

PROTOCOL CHANGES

UNCLEAR REGULATION

UNCLEAR TAX

SOLVABLE

Ethereum

Une blockchain d’infrastructure

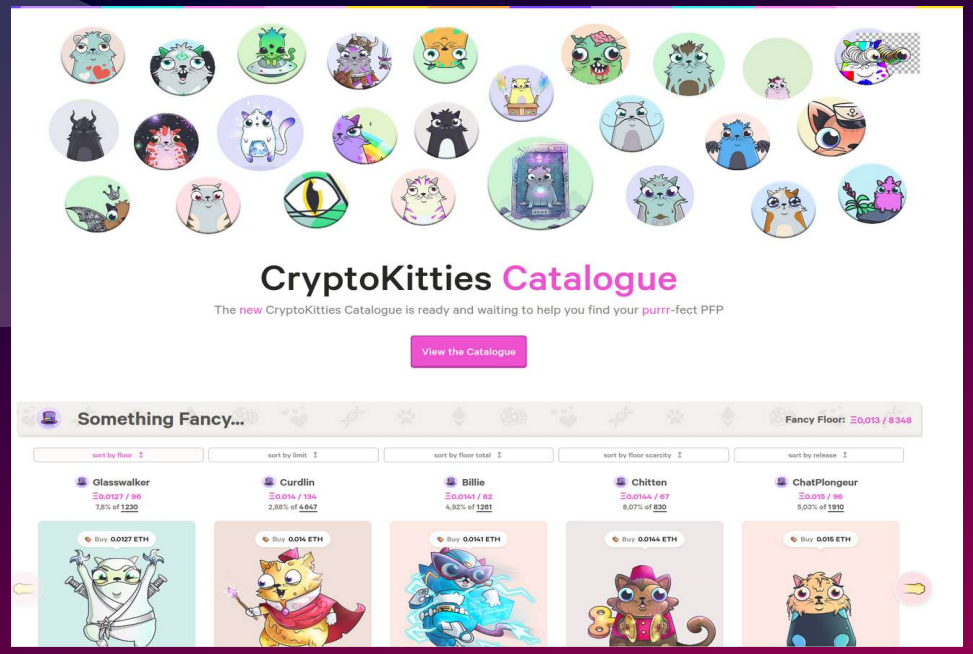
- Dapp (Decentralized Application)

Ethereum

La folie des ICO (2017)

Ethereum

Les CryptoKitties



- Révélation des limites d’ETH (gas fees, tx longues, trafic
- Scalabilité d’ETH remise en cause
- Prémices des NFT ?

Ethereum

L’émergence de la révolution (2018-2019)

- Prêt avec intérêt
- Emprunt en échange d’intérêt

Ethereum

L’émergence de la révolution (2018-2019)

- Exchange centralisé
- Kyc
- Risque lié au CEX
- Avantage : très liquide

Ethereum

L’émergence de la révolution (2018-2019)

- Exchange décentralisé
- Pool de liquidités

Ethereum

L’émergence de la révolution (2018-2019)

- Oracle décentralisé
- Pont offchain <==> onchain

Ethereum

L’émergence de la révolution (2018-2019)

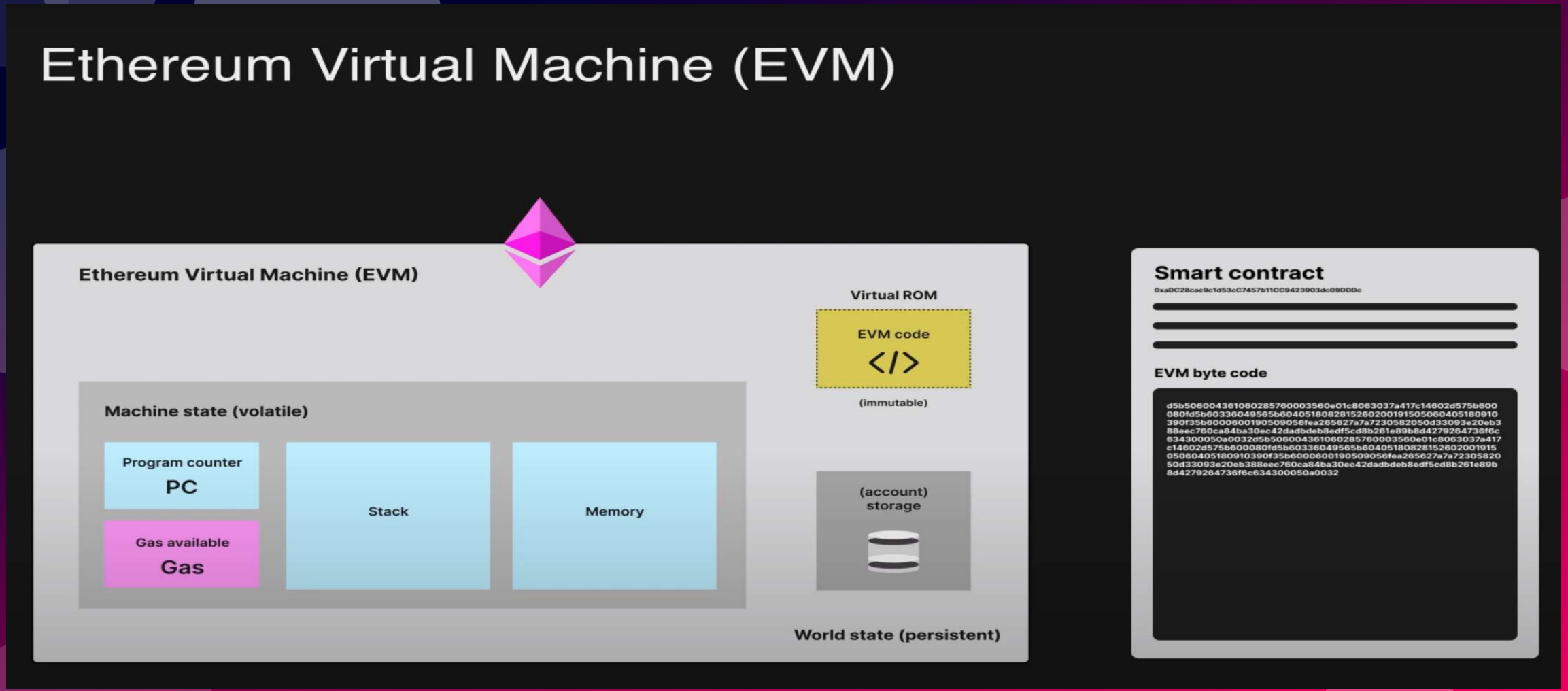
- Oracle décentralisé
- Pont offchain <==> onchain

Solidity

SOLIDITY

EVM (Ethereum Virtual Machine

Déf : Environment that runs SC on the ETH BC. Backbone d'ETH



EVM (Ethereum Virtual Machine)

Smart contracts

SOLIDITY

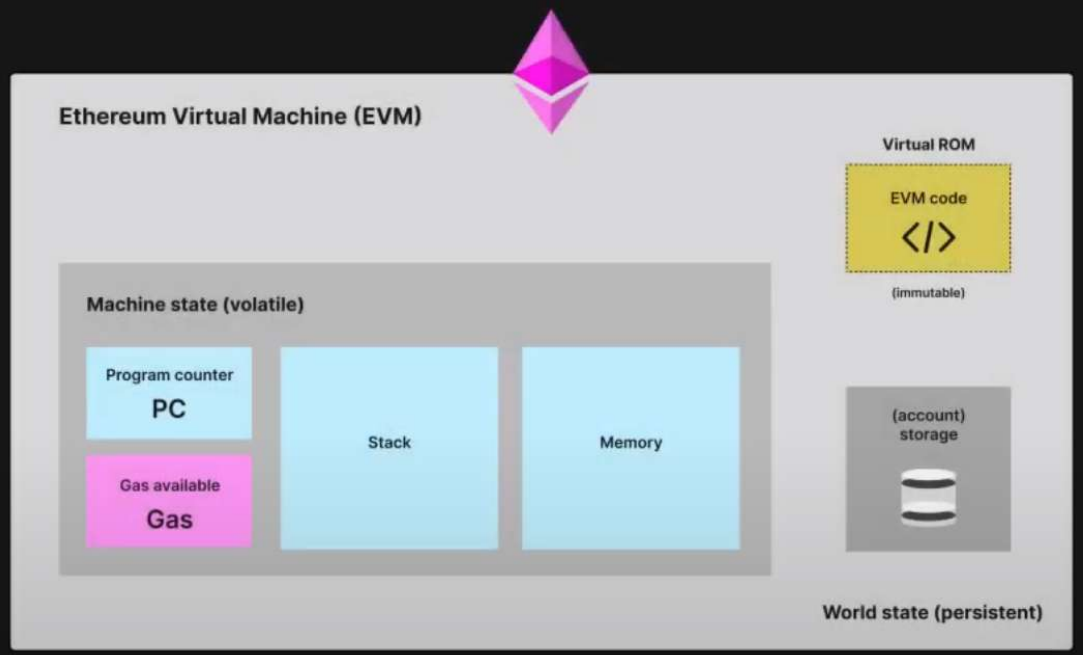
```
// SPDX-License-Identifier: MIT

pragma solidity >=0.8.9 <0.9.0;

import 'erc721a/contracts/ERC721A.sol';
import '@openzeppelin/contracts/access/Ownable.sol';
import '@openzeppelin/contracts/utils/cryptography/MerkleProof.sol';
import '@openzeppelin/contracts/security/ReentrancyGuard.sol';

contract VeryCoolApesAA is ERC721A, Ownable, ReentrancyGuard {

  using Strings for uint256;

  bytes32 public merkleRoot;
  mapping(address => bool) public whitelistClaimed;

  string public uriPrefix = '';
  string public uriSuffix = '.json';
  string public hiddenMetadataUri;

  uint256 public cost;
  uint256 public maxSupply;
  uint256 public maxMintAmountPerTx;

  bool public paused = true;
  bool public whitelistMintEnabled = false;
  bool public revealed = false;

  constructor(
    string memory _tokenName,
    string memory _tokenSymbol,
    uint256 _cost,
    uint256 _maxSupply,
    uint256 _maxMintAmountPerTx,
    string memory _hiddenMetadataUri
  ) ERC721A(_tokenName, _tokenSymbol) {
```

=>

EVM bytecode

```
6080604052348015600f57600080fd5b5060878061001e6000396000f3fe
052348015600f57600080fd5b5060878061001e6000396000f3fe6080604
57600080fd5b5060878061001e6000396000f3fe6080604052348015600f5
052348015600f57600080fd5b5060878061001e6000396000f3fe6080604
6080604052348015600f57600080fd5b5060878061001e6000396000f3fe
052348015600f57600080fd5b5060878061001e6000396000f3fe6080604
57600080fd5b5060878061001e6000396000f3fe6080604052348015600f5
052b5060878061001e6000396000f3fe6080604052348015600f57600080f
052348015600f57600080fd5b5060878061001e6000396000f3fe6080604
6080604052348015600f57600080fd5b5060878061001e6000396000f3fe
052348015600f57600080fd5b5060878061001e6000396000f3fe6080604
576000600f57600080fd5b5060878061001e6000396000f3fe60806040523
6080604052348015600f57600080fd5b5060878061001e6000396000f3fe
052348015234801560f57600080fd5b5060878061001e6000396000f3fe
57600080fd5b5060878061001e6000396000f3fe6080604052348015600f5
052348015600f57600080fd5b5060878061001e6000396000f3fe6080604
6080604052348015600f57600080fd5b5060878061001e6000396000f3fe
052348015600f57600080fd5b5060878061001e6000396000f3fe6080604
57600080fd5b5060878061001e6000396000f3fe6080604052348015600f5
052b5060878061001e6000396000f3fe6080604052348015600f57600080f
052348015600f57600080fd5b5060878061001e6000396000f3fe6080604
6080604052348015600f57600080fd5b5060878061001e6000396000f3fe
052348015600f57600080fd5b5060878061001e6000396000f3fe6080604
576000600f57600080fd5b5060878061001e6000396000f3fe60806040523
6080604052348015600f57600080fd5b5060878061001e6000396000f
```

=>



SOLIDITY

GAS (Unité : ETH, Gwei = 10^-9 eth, Wei = 10^-18 eth)

Mais c'est quoi le gaz ? 😊

- Unité qui mesure la quantité d'efforts de calculs requis pour exécuter des opérations spécifiques sur la Blockchain Ethereum
  - Chaque transaction nécessite des ressources informatiques pour s'exécuter, cela implique des frais (gaz)
  - Fonctionne comme l'essence d'une voiture
- Les frais de gaz sont payés en ether (ETH). Les prix du gaz sont indiqués en gwei (1 gwei = 0,00000001 ETH (10-9 ETH))

- Deux buts :
  - Empêcher la surcharge du réseau ou les erreurs
  - Rémunérer les validateurs du réseau

SOLIDITY

EVM - Solidity - Opcodes - Bytecodes

- Les opérations sur la Blockchain Ethereum sont effectuées par l'EVM
- Ces opérations coûtent du gaz
  - Multiplier 2 nombres coûte 5 Gaz
  - Une soustraction coûte 3 Gaz
  - Une transaction coûte 21 000 Gaz
- Les Smarts contracts combinent de nombreuses opérations et peuvent rapidement coûter beaucoup de Gaz
- Gaz != Ether
- Le Gaz se paie en Ether

SOLIDITY

Un exemple : une transaction entre deux comptes

- Prix en Gaz = Montant de Gaz X Prix du Gaz (en Gwei)
- Une transaction coûte 21 000 GAZ
- Prix du gaz maintenant : 16 Gwei/GAZ
  - Fast
  - Medium
  - Slow
- La transaction coûte : 21 000 * 16 = 336 000 Gwei
- 336 000 Gwei = 0.000336 Ether
- Prix en € = 0.000336 * 1146€ = 0.385056€

SOLIDITY

etherscan.io/gastracker

ETH Price: $1,792.24 (-3.46%) Gas: 60 Gwei

Search by Address / Txn Hash / Block / Token / Domai

Etherscan

Home Blockchain Tokens NFTs Resources Developers More | Sign In

Ethereum Gas Tracker ⛽

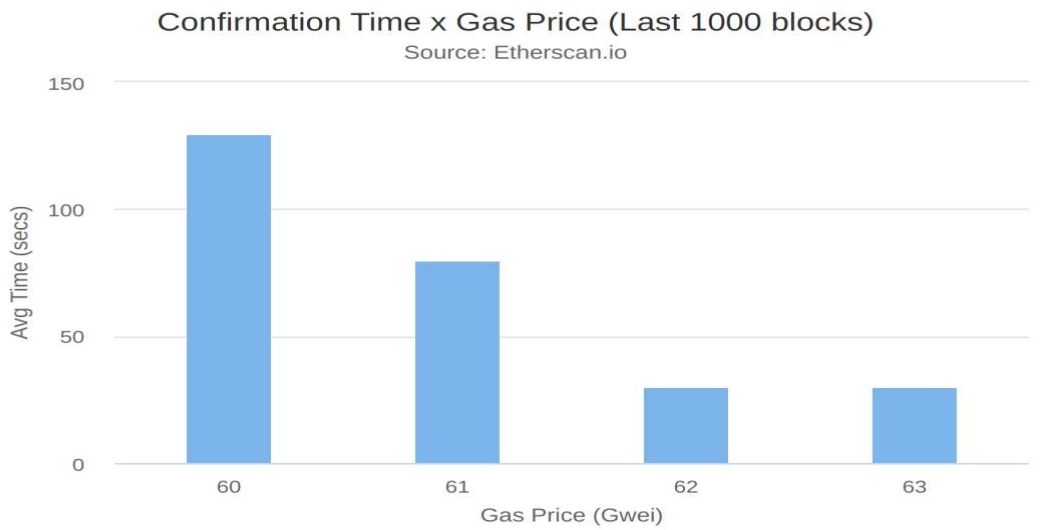
Next update in 2s

Wed, 24 May 2023 14:37:59 UTC

😁 Low	😀 Average	🙂 High
60 gwei	60 gwei	63 gwei
Base: 59 | Priority: 1 $2.26 | ~ 3 mins: 0 secs	Base: 59 | Priority: 1 $2.26 | ~ 3 mins: 0 secs	Base: 59 | Priority: 4 $2.41 | ~ 30 secs

Estimated Cost of Transaction Actions: View APIs ↗

Action	Low	Average	High
OpenSea: Sale	$7.83	$7.83	$8.35
Uniswap V3: Swap	$20.17	$20.17	$21.50



SOLIDITY

Métaphore de la voiture

- Une voiture permet d'aller d'un point A à un point B
  - Prix d'un litre d'essence : 1.9€
  - Nombre de litres qui sera consommé : 10 litres
  - Prix final : 1.9€ * 10 = 19€
- Faire une transaction sur la Blockchain Ethereum
  - Prix d'une unité de Gaz : 19 Gwei
  - Nombre d'unités de Gaz qui seront consommées : 21 000
  - Prix final : 19 Gwei * 21 000 = 399000 Gwei = 0.000399 * 1500€ = 0.5985€

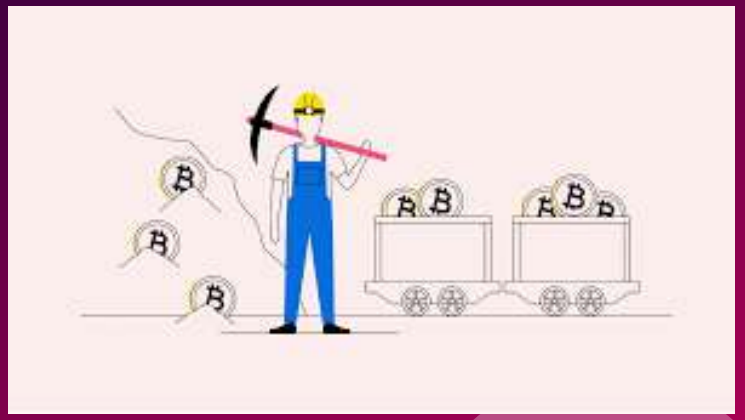
SOLIDITY

Le prix du Gaz

- MemPool
  - Piscine des transactions
  - Les validateurs prennent les transactions les plus rémunératrices
  - Gas Limit / Block : 30 000 000
    - 30 000 000 / 21 000 = 1428 transactions simples par Block
- Le prix du Gaz varie en fonction de la congestion du réseau Ethereum
  - Si gros projet NFT en cours ☝
  - Si gros projet DeFi en cours ☝

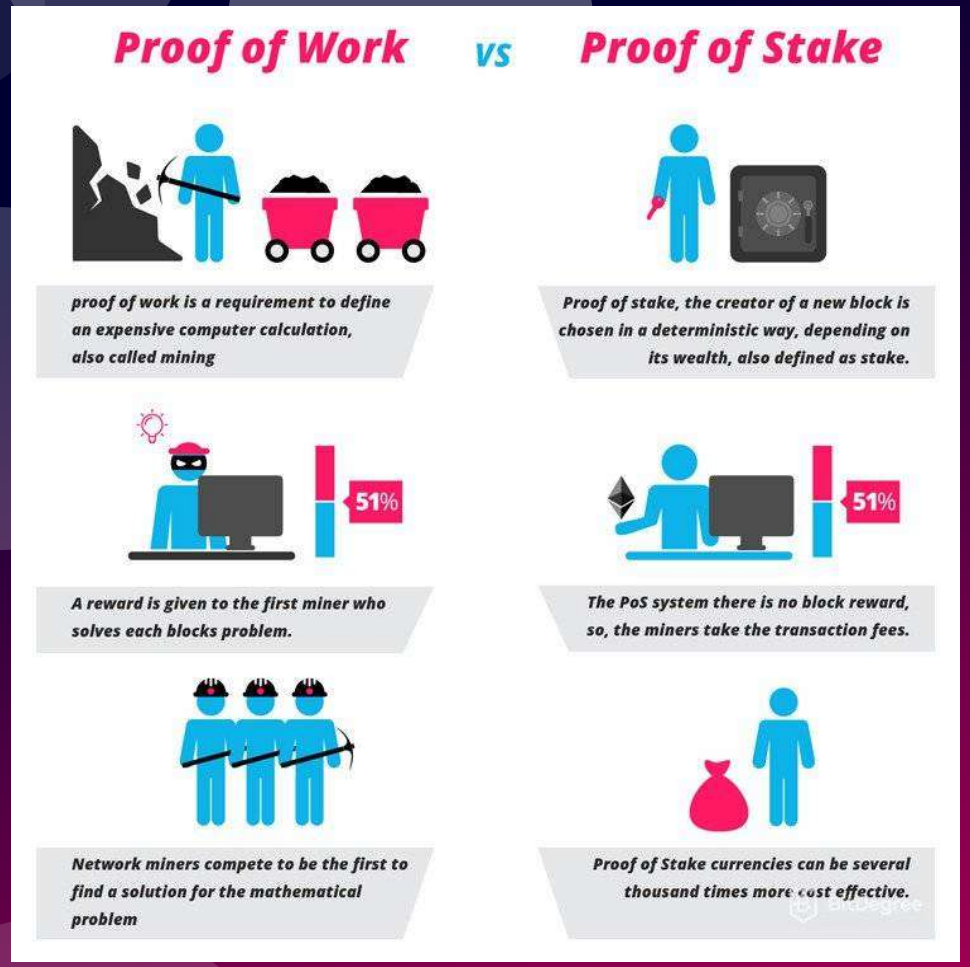
Blockchain et Consensus

Proof of work



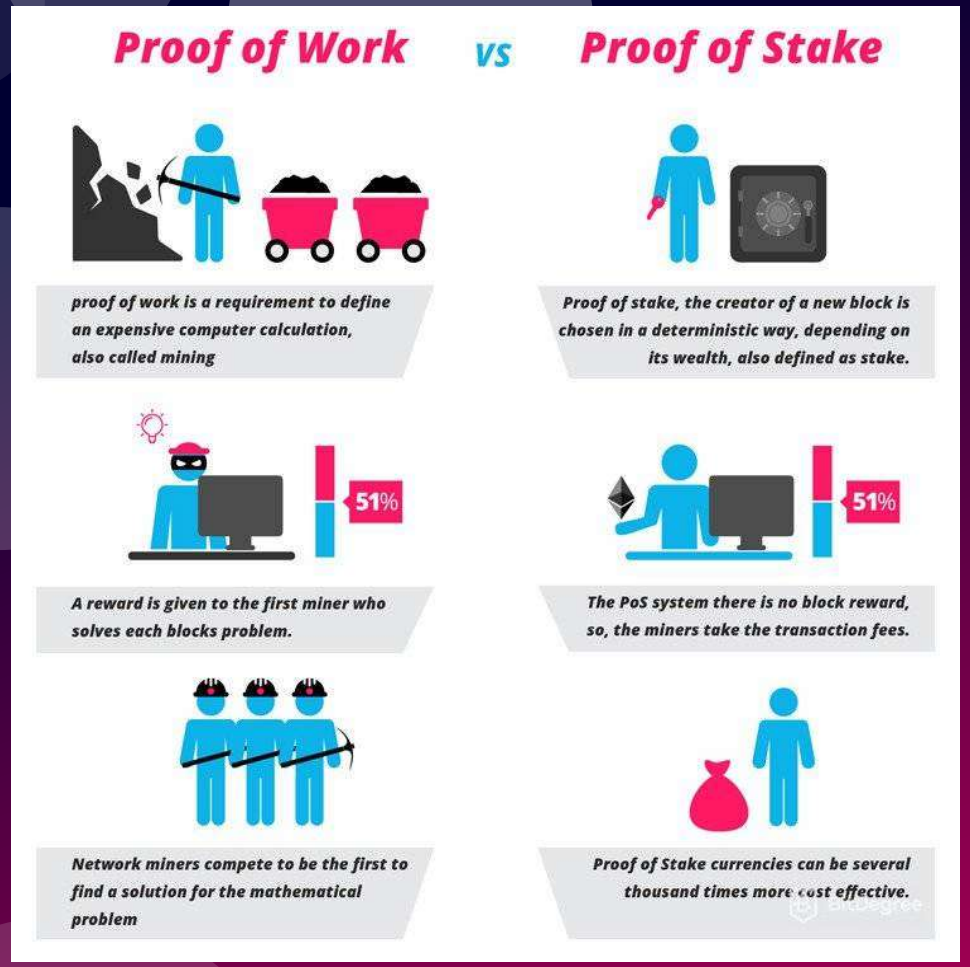
Blockchain et Consensus

Proof of stake



Blockchain et Consensus

Proof of stake



Blockchain et Consensus

Pure Proof of stake

Idem Proof of stake
mais conditions pour être 1 validateur
……. 1 ALGO

Blockchain et Consensus

Delegated Proof of stake (DPoS)

- Détenteurs du token natif élisent des délégués (⇒ validateurs)
- Risque de Centralisation

TRON

Blockchain et Consensus

Liquid Proof of stake

- Les users déléguent leurs tokens en touchant une récompense % au montant mise en jeu
- Avantages : Incitation à la délégation

Tezos

Blockchain et Consensus

Proof of history (PoH)

- Bad experience …. :-)

Blockchain et Consensus

Proof of authority (PoA)

- blocs et les tx sont validés par des comptes approuvés à l’avance
- Identité soit formellement vérifiée et affichée sur la blockchain
- Basé sur identité et réputation
- Risque : Très centralisé
- Avantages : très rapide

Sources

- The main reference for this slide set is the Solidity language documentation
  - URL: https://docs.soliditylang.org/en/latest/index.html
- This section is the Types section therein
  - https://docs.soliditylang.org/en/latest/types.html
  - Many of the examples herein are from there

SOLIDITY

A NOTE ABOUT THESE SLIDES

- These slides are meant as a explanation
  - You are not expected to remember everything!
- Since you all know OOP, it's just showing what is different in Solidity
- You can come back to use this as a reference later

Value types: integers

- Signed and unsigned ints from 8 bits (1 byte) to 256 bits (32 bytes) in size
  - And in all byte sizes between
- Unsigned: `uint8`, `uint16`, `uint24`, ... `uint256`
  - `uint` is an alias for `uint256`
- Signed: `int8`, `int16`, `int24`, ..., `int256`
  - `int` is an alias for `int256`
- Standard operations (arithmetic, comparisons, bit operators, shift operators)
  - Exponentiation is `**`
- `type(int256).max` and `type(uint160).min` for min/max

SOLIDITY

Other value types

- Booleans: type is `bool`, values are `true` and `false`
- All comparisons evaluate to a Boolean
- Fixed point numbers: not yet fully supported
  - Types are `fixed` and `ufixed`
  - But don't use them!
- No floating point numbers!
- Note that all variables have an initial value of 0

SOLIDITY

Addresses

- Types are `address` and `address payable`
- It's just a number: a 160-bit (20 byte) Ethereum address
  - Allows standard comparison operators
- Used for:
  - Keeping track of who is the contract initiator
  - Addresses of other contracts to call
- Can cast between `address` and: `uint160` or `bytes20`

```
address a = address(0);
uint160 u = uint160(a);
bytes20 b = bytes20(u);
```

- The address of a caller of a function is in `msg.sender`

Address examples

- You can convert to a payable address:

```
address a;
// set a = something...
address payable b = payable(a);
```

- Conversions:

```
uint160 b = uint160(msg.sender);
bytes20 c = bytes20(msg.sender);
```

- More examples:

```
if ( a == address(0) ) { ... }
address initiator = msg.sender;
```

Address fields

- Although a primitive type, it has fields and methods:
  - `balance`: the balance, in wei, for that address

```
address payable x = payable(0x123);
address myAddress = address(this);
if (x.balance < 10 && myAddress.balance >= 10) {...}
```

- Other methods we won't see in this course:
  - `transfer()` to send wei (requires a `receive()` function on the receiving contract)
  - `send()`: low-level and not safe version of `transfer()`
  - `call()`, `delegatecall()`, and `staticcall()`: calling a contract with an unknown ABI
  - `code()` and `codehash()` to get the bytecode of the contract

Fixed-sized Byte Arrays

- Fixed sized array of bytes: `bytes1`, `bytes2`, up to `bytes32`
  - The size is the integer in the type name; max of 32 bytes
  - `.length` to get the length
  - Indexing with [ ]
- Operators:
  - Comparisons, bit operators, shift operators
  - Treats the byte array like a similarly sized `uint` when performing the operation

SOLIDITY

Dynamically-sized arrays

- `bytes`: dynamically sized array of bytes
- `string`: dynamically sized array of UTF-8 characters
  - We'll see string functions in a bit...

SOLIDITY

String literals

- Strings can be enclosed in single quotes or double quotes
- Can use the backslash for escape characters
- Two strings next to each other is concatenation:

```
string s = "CS" '4501'; // equals "CS4501"
```

More general arrays

- We can create arrays of any type:

```
uint8[6] memory p = [ 1, 2, 3, 4, 5, 6 ];
```

  - We'll see the `memory` keyword shortly...

Enums

- Enumerated type, which is mapped to a `uint`

```
enum ActionChoices { GoLeft, GoRight,
                     GoStraight, SitStill }
//
ActionChoices choice;
ActionChoices constant default = ActionChoices.GoLeft;
//
function setGoStraight() public {
  choice = ActionChoices.GoStraight;
}
```

SOLIDITY

Mappings

- An associative array (like a hash table) that holds a key-value pair

```
mapping (address => bool) public override voted;
mapping (uint => Choice) internal _choices;
```

  - We'll talk about `public`, `internal`, and `override` later
- To access a mapping value:

```
if ( voted[msg.sender] ) { ... }
```

- To write to a mapping:

```
voted[msg.sender] = true;
```

Mappings of mappings

- You can have mappings to mappings:

```
mapping (uint => mapping(string => address) )
                public twodmap;
```

- It's just like a 2-dimensional array:

```
twodmap[3]["foo"]
```

SOLIDITY

Is an element in a mapping?

- Consider:

```
mapping (address => bool) public override voted;
// this next one was not in the original Poll.sol:
mapping (address => uint) public how_voted;
//
// or:
//
mapping (uint => Choice) internal _choices;
uint public override num_choices;
```

- The `voters` tells if an address has voted
  - If they haven't, then that key's value will be 0 (false)
  - The `how_voted` will tell what they voted for
- For the Choices, the uint tells how many keys there are
  - Our code has keys 0 to num_choies-1

SOLIDITY

Structs

- Like a class without methods

```
struct Choice {
  uint id;
  string name;
  uint votes;
}
```

  - Structs can contain mappings as well
- They are often kept in a mapping:

```
mapping (uint => Choice) internal _choices;
```

- If you put the same `Choice` struct into different mappings, you *might* now have *two* copies of that struct!

Time

- All time is kept as a UNIX timestamp
  - The number of seconds since January 1st, 1970
  - This is how it's kept in the blockchain as well
- Only time available in a contract is the timestamp of the block
  - Via `block.timestamp`
- Can use `seconds`, `minutes`, `hours`, `days`, and `weeks` as time units
  - They are always pluralized
  - But the number before each must be a literal, not a variable!
  - Example on the next slide

SOLIDITY

Time units example

```
uint a = 1;
uint b = 5;
uint c = 2;

// valid:
uint endTime = block.timestamp +
               1 seconds + 5 minutes + 2 days;

// not valid:
uint endTime = block.timestamp +
               a seconds + b minutes + c days;

// valid:
uint endTime = block.timestamp +
               a * 1 seconds + b * 1 minutes + c * 1 days
```

Ether units

- Can convert ether units as well
- All convert to wei (recall that 1 ether is 10^{18} wei)

```
assert(1 wei == 1);
assert(1 gwei == 1e9);
assert(1 ether == 1e18);
```

Memory locations

- All reference types must specify a memory location
  - `storage`: for all state variables, it is stored on the blockchain
    - statically allocated, like the stack on x86
    - two sub-types:
      - local storage (local subroutine variable)
      - state storage (state/class variable on the blockchain)
  - `memory`: a local variable, it only exists while the subroutine is executing
    - dynamically allocated, like the heap in x86
  - `calldata`: read-only, used for function parameters

SOLIDITY

Assignments of References

- If you assign one reference type to another, what happens depends on where the data is stored:

To (lhs)	From (rhs)	Result
memory	memory	reference aliasing
memory	any storage	full copy
memory	calldata	full copy
any storage	memory	full copy
any storage	calldata	full copy
local storage	any storage	reference aliasing
state storage	any storage	full copy
calldata	(any)	not allowed

Assignments of references

- How to remember:
  - You can't ever write to `calldata`
  - Anything assigned between *different* types of memory is always a copy, as references can't point from one memory type to another
  - State storage, which is on the blockchain, never stores references -- so anything copied to state storage is always a full copy
  - Reference aliasing works in the 2 cases when the memory types are the same and it's not going into state storage: memory to memory, and any storage to local storage

Type names

- For certain types, you have to give a qualifier as to where it is stored
  - `bytes` and `string`
- The qualifier is `memory`, `storage`, or `calldata`
- The type name is *two words*
  - `string memory`
  - `bytes storage`

String functions

- No (direct) built-in string comparison!
  - Reasons:
    - It is difficult / expensive to compare across memory locations
  - Instead, compare their hashes:

```
if ( keccak256(x) == keccak256(y) ) { ... }
```

    - Note that `keccak256()` uses a lot of gas
- Can concatenate two strings:

```
string memory s = string.concat(s1, s2);
```

  - Only returns a `string memory`
- There are 3rd party string function libraries
  - Which contain comparisons, among other functions

Code Example

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity >=0.5.0 <0.9.0;

contract C {
  // The data location of x is storage; this is the
  // only place where the data location can be omitted
  uint[] x;

  // The data location of memoryArray is memory.
  function f(uint[] memory memoryArray) public {
    x = memoryArray; // works, copies the whole array to storage
    uint[] storage y = x; // works, assigns a pointer, y's location is storage
    y[7]; // fine, returns the 8th element
    y.pop(); // fine, modifies x through y
    delete x; // fine, clears the array, also modifies y
    // The following doesn't work; it would need to create a new temporary /
    // unnamed array in storage, but storage is "statically" allocated:
    // y = memoryArray;
    // This does not work either, since it would "reset" the pointer,
    // but there is no sensible location it could point to.
    // delete y;
    g(x); // calls g, handing over a reference to x
    h(x); // calls h and creates an independent, temporary copy in memory
  }
```

Learning Solidity

- The easy part: learning the language
  - It's just an OO language with familiar syntax
- The hard part is understanding:
  - The restrictions that blockchain programs have
    - No time or random numbers
    - No print statements!
  - How to best interact with the blockchain
  - What the heck do you do with it?
- The easy part will be done in the next 20 minutes
- The hard part will be the rest of this course

Sources

- The main reference is the Solidity language documentation
  - URL: https://docs.soliditylang.org/en/latest/index.html
- Ethereum developer resources
  - URL: https://ethereum.org/en/developers/
- Solidity reference
  - URL: https://docs.soliditylang.org
- Remix documentation (the Solidity IDE)
  - URL: https://remix-ide.readthedocs.io/en/latest/

SOLIDITY

Solidity Overview

- An object-oriented language with C++/Java like syntax
  - Same set of control structures with the same syntax (for, if/else, while, etc.)
- A class is called a "contract"
- Methods are called functions

SOLIDITY

License identifier

- All Solidity smart contracts must have a license line as the first line
  - SPDX is the Software Package Development Exchange
- Examples:

```
// SPDX-License-Identifier: MIT
```

```
// SPDX-License-Identifier: CC BY-SA
```

```
// SPDX-License-Identifier: GPL
```

- A compiler will issue a warning if it's not there
- The actual license type is not checked, so you can also use:

```
// SPDX-License-Identifier: Unlicensed
```

Pragma

- We'll only use the following Solidity version pragma:

```
pragma solidity ^0.8.17;
```

- The three numbers are: major version, minor version, bugfix version
- By default, it forces a compiler error if the wrong compiler version is being used
  - Some platforms, such as Truffle or Remix, will try to find the right compiler version
- This will only compile with version 0.8.17 or later
  - But will NOT allow compilation with version 0.9.0 or later
- Note that breaking changes (very few!) are only (intentionally) introduced on major versions

Pragma

- More complicated examples are possible:

```
pragma solidity >=0.4.0 <0.6.0;
```

- One to avoid: note there is no carat before the version number
  - Reason: a later bugfix version, 0.8.12, will not compile this program

```
pragma solidity 0.8.11;
```

- In this course, we'll use ONLY the following:
  - As this will compile on all of the platforms we will be using

```
pragma solidity ^0.8.17;
```

Comments

- C++ and Java style comments

```
// this is a comment

/* this is a
   multi-line
   comment
 */

/* this is a
 * multi-line
 * comment with better
 * formatting
 */
```

SOLIDITY

Import

- Default import statement:

```
import "./filename.sol";
```

  - Note that this will pollute the namespace
- We can also use:

```
import * as symbolName from "./filename.sol";
```

  - All names from the file can be accessed via `symbolName.thing`
  - An equivalent version of this is:

```
import "./filename.sol" as symbolName;
```

- Naming collision? Then rename it upon import:

```
import {symbol1 as alias, symbol2} from "./filename.sol
```

High-level "things"

- `contract`
  - Essentially an OO class
  - Can also be `abstract`
- `interface`
  - Just like interfaces in Java, or pure abstract classes in C++
  - By convention, their names always start with a capital I (India)
- `library`
  - Can contain ONLY functions that do not access state
    - Meaning no contract field access
    - Specifically, they can only contain `pure` functions (we'll see `pure` shortly)
  - Unless you have a reason otherwise, have everything inside have `internal` visibility
    - We'll see visibilities shortly...

Inheritance

- Contracts can inherit from interfaces and other contracts
  - Interfaces can inherit from (only) other interfaces
  - Via the `is` keyword
- The super-class (or super-interface) must be defined above or `import`'ed

```
interface Foo is Bar {
// ...
}
```

```
contract Student is Person, Learner {
// ...
}
```

- Any contract that inherits from an interface, but does not implement all the methods, must be declared `abstract`

Inheritance

- `virtual`: if a *thing* (function, field, etc.) in a contract can be overridden
  - All function prototypes in interfaces are assumed to be `virtual`
- `override`: when a sub-class is replacing an inherited function
  - If the function is multiply inherited, you may have to specify which (or all) that it overrides:

```
function foo() public override(IERC165,ERC721) {
```

    - This will be needed in the Tokens assignment
  - The requirement to specify `override` varies by compiler and version

Abstract Contracts

- A `contract` must have all the methods implemented
- An `interface` must have no methods implemented
- An `abstract contract` can have some methods implemented and some not
  - Part interface, part contract
  - Example: if you inherit from an interface, but only implement some of the methods

```
abstract contract Foo is Bar {
  // ...
}
```

What we have so far

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

interface IERC165 {
  // ...
}

interface IERC721 is IERC165 {
  // ...
}

contract ERC721 is IERC721 {
  // ...
}

library Address {
  // ...
}
```

Visibilities

- `public`: like any other OO language, it can be called by anybody or anyone
  - For fields, this is only the *readability* of the field; only contract functions can *write* to the field
- `private`: like any other OO language, it can be called only by code in that class
- `internal`: like `protected` in other OO languages: it can be called by that class or it's sub-classes
- `external`: like `public`, but can ONLY be called by code *outside* the contract; code in the contract can not call an `external` function
  - Functions in interfaces must be declared `external`
  - The implementing contract can then "raise" the visibility to `public`

Visibilities

- The stated visibility in an interface can be expanded (only) in the contract
  - Example: if an interface specifies a function as `external`...
    - Then the implementing contract can specify it either as `external` or `public`
      - As `public` is *more* visible than `external`
- Anything field defined as `public` has a getter function defined for it -- see the next slide
  - If you have `uint public k`, then you can call the `k()` getter function to access its value
- The default, if unspecified, is `public`
  - But without a getter function
- Note that all data is still visible on the blockchain, even for `private` fields!

Getter functions

- The following code:

```
uint public num_assignments;
mapping (address => string) public aliases;
mapping (uint => mapping(string => address) )
            public twodmap;
```

- Creates the following getter functions:

```
function num_assignments() public view returns (uint);
function aliases(address a) public view
                returns (string memory);
function twodmap(uint id, string s) public view
                returns (address);
```

Function Qualifiers

- `view`: this function does not *modify* the state of the contract
  - It can read state variables, but not write to any state variables
  - It can write to local variables, of course
  - Like `const` in C++
- `pure`: this function does not *read or write* the state of the contract
  - Any function in a `library` must be `pure`
  - `view` and `pure` are mutually exclusive
- Using these qualifiers allows calls without having to make a transaction
  - And *may* allow optimizations which reduce contract size and gas costs

Declarations for fields

- Format:

```
<type> <visibility> <override?> <name> [= <value>];
```

- Examples:

```
uint public duration = 86400;
uint public override k;
address public override initiator;
mapping (address => uint) internal values;
```

- Local variables are similar
  - But without the visibility and `override` qualifiers

SOLIDITY

Declarations for functions

- Format:

```
function <name> ([<parameter_list>]) <visibility>
                <qualifiers> [returns (<list>)] {
```

- Examples:

```
function addChoice (string memory _name) public
                { /* ... */ }

function unnecessaryFunction() public view
                returns (string memory) { /* ... */ }

function overridePureFunction() external pure
                override returns (uint,bool) { /* ... */ }
```

Require and revert

- A reversion means the transaction method fails, and all state is rolled back to before it started
  - Still costs gas fees!
- `revert()` will do this
- Also:
  - `require(a>0);` will revert if a is not greater than zero
  - Better to use the two-parameter version:
  - `require(a>0,"a must be greater than zero");`
    - Now, if/when it reverts, it will tell you why
  - Use this all the time!!!

Receiving ether

- The `payable` qualifier means the function can accept ether as part of the call
- The amount of ether received is in `msg.value`
  - And is added to the contract's balance
    - ... assuming it doesn't revert

Constructors

- Syntax:

```
constructor() {
  // ...
}
```

- No `function` keyword, no return type
- Default, if not specified, is an empty body
- Can take parameters:

```
constructor(uint foo) {
  // ...
}
```

- Only one constructor is allowed per class!
  - They may eventually allow multiple, differentiated by the parameter list

SOLIDITY

Destructor

- `selfdesctruct()`
  - Only if enabled (declared)
  - Disables the contract permanently
  - Caller claims the ether balance

Contract elements

- A contract can have:
  - Fields
  - Functions
  - A constructor and a `selfdestruct()` method
  - A few other special-purpose methods we won't see here
  - Modifiers

SOLIDITY

Modifiers

- A modifier changes how a function works
- You put in conditions (or whatever) and then indicate where the function body goes
- That indication is with the underscore
- Example:

```
modifier onlyOwner() {
    require(msg.sender == owner, "Not owner");
    _;
}
```

Modifiers

- This code adapted from solidity-by-example.org:

```
contract Modifiers {
    address public owner;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Not owner");
        _;
    }

    modifier validAddress(address _addr) {
        require(_addr != address(0), "Not valid address");
        _;
    }

    function changeOwner(address _newOwner) public onlyOwner
                                            validAddress(_newOwner) {
        owner = _newOwner;
    }
}
```

Events & emit

- An *event* is when something happens
- Allows for logging of what has happened to a contract
- They are declared in contracts or interfaces:

```
event votedEvent (uint indexed _id);
event choiceAddedEvent (uint indexed _id);
```

  - The `indexed` keyword allows for searching the event log by that value
- Events are called via `emit`:

```
emit choiceAddedEvent(num_choices);
emit votedEvent(_id);
```

- Events can be "watched" for (or "subscribed to")
  - We'll see this later this semester

Try-catch

- Try-catch:

```
try <something> {
  // what to do if it succeeds
} catch {
  // what to do if it fails
}
```

- The "something" can be:
  - Calling a function
  - Creating a new contract
  - Or anything else
- If the *something* reverts, the catch clause is executed

Transferring ether in Solidity

- To transfer ether from a Solidity contract:

```
(bool success, ) = payable(a).call{value: v}("");
require(success, "Failed to transfer ETH");
```

- The amount, in v, is in wei
- The address to pay it to is in a
- This reverts if the contract does not have sufficient balance
  - Or if it fails for any other reason, such as an invalid address

Contracts creating contracts

- We can have a contract `Foo` deploy another contract `Bar`
- Given a field in Foo such as either:

```
Bar public b;
address public b;
```

- Then we can initialize it via either (respectively):

```
b = new Bar();
b = address(new Bar());
```

Most useful global variables

From here

- block.timestamp (uint)
  - current block timestamp as seconds since unix epoch
- block.difficulty (uint)
  - current block difficulty
- msg.sender (address)
  - sender of the message (current call)
- msg.value (uint)
  - number of wei sent with the message
- this
  - the current contract (example: address(this).balance)

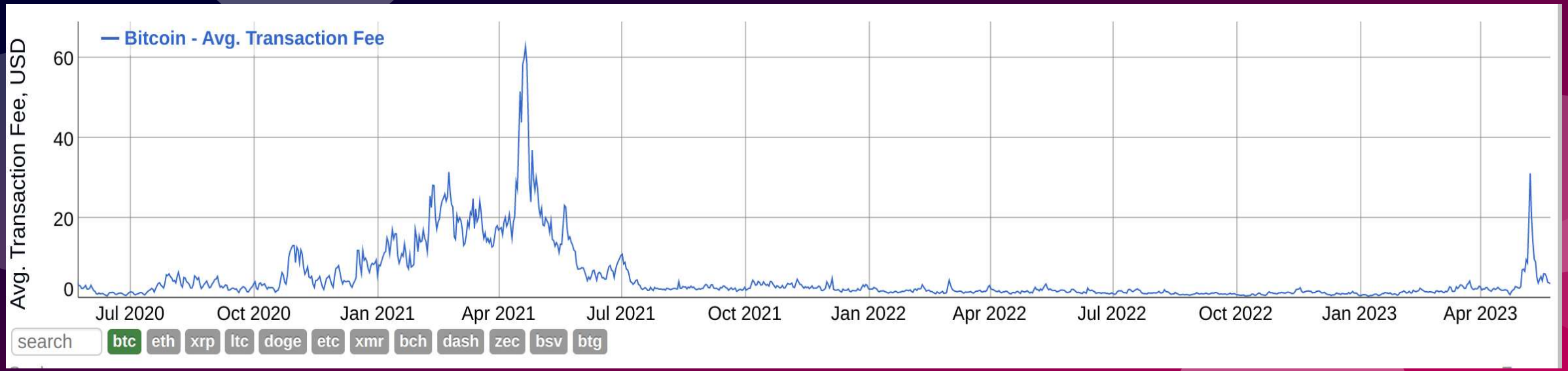
Other global variables

- block.basefee (uint): current block's base fee (EIP-3198 and EIP-1559)
- block.chainid (uint): current chain id
- block.coinbase (address payable): current block miner's address
- block.gaslimit (uint): current block gaslimit
- block.number (uint): current block number
- gasleft() returns (uint256): remaining gas
- msg.data (bytes calldata): complete calldata
- msg.sig (bytes4): first four bytes of the calldata (i.e. function identifier)
- tx.gasprice (uint): gas price of the transaction
- tx.origin (address): sender of the transaction (full call chain)

Global functions

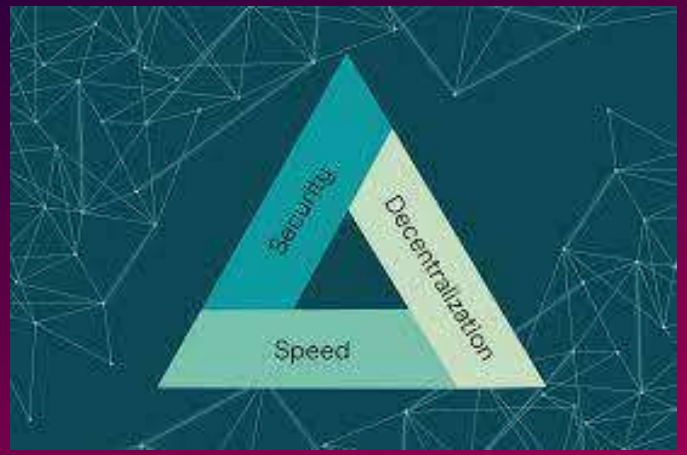
- blockhash(uint blockNumber) returns (bytes32)
  - hash of the given block when blocknumber is one of the 256 most recent blocks; otherwise returns zero
- keccak256(bytes memory) returns (bytes32)

Blockchain : la scalabilité en question



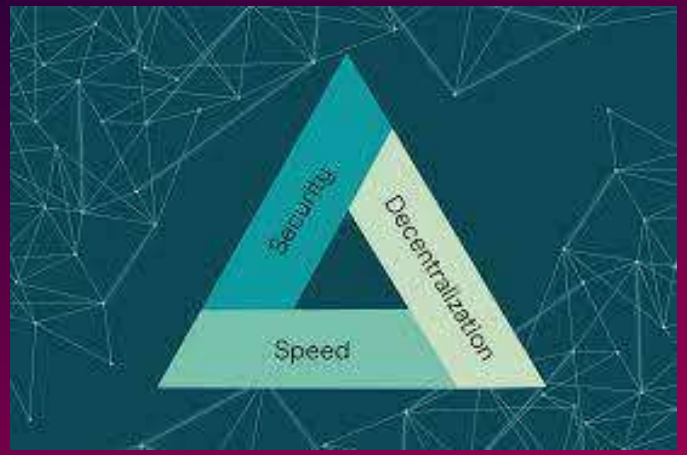
Blockchain : la scalabilité en question

Concept évoqué par V. Buterin dès le début d'ETH



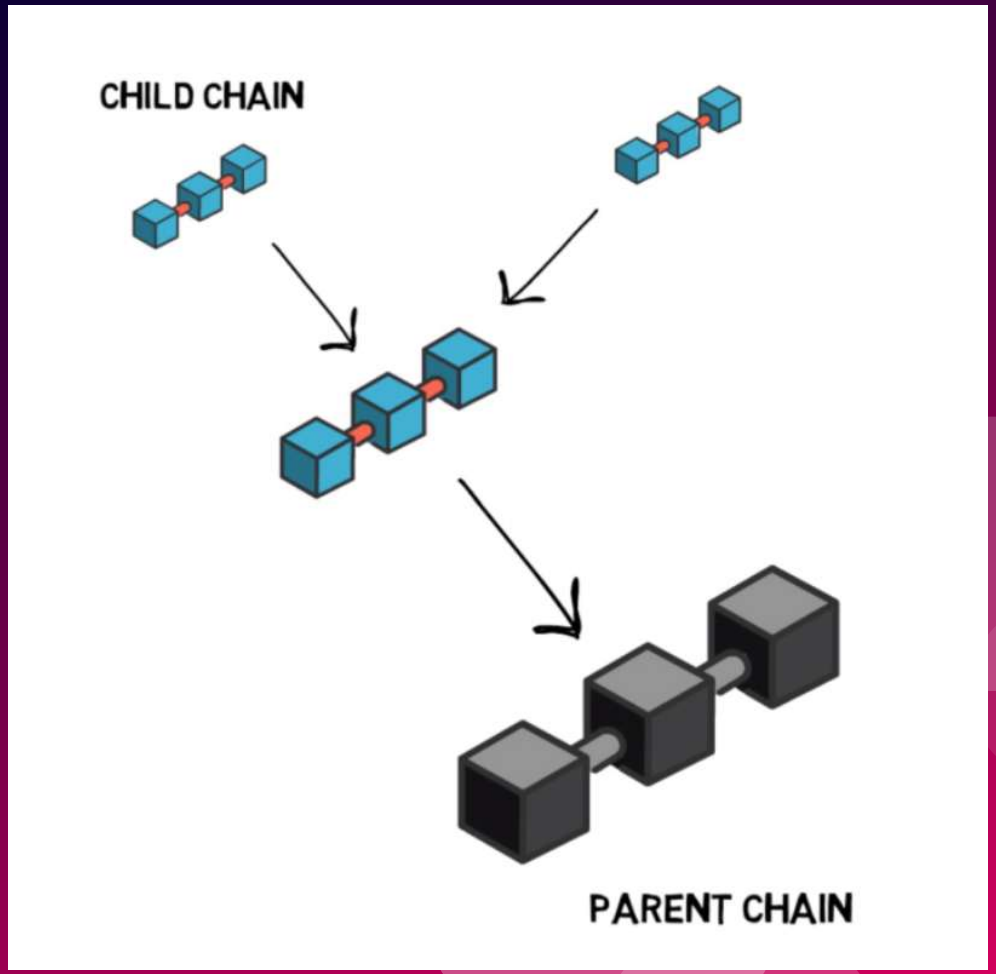
Blockchain : la scalabilité en question

- 3 composantes inhérentes à la Blockchain
- Décomposer les propriétes hors Sécurité à d’autres « couches »



Blockchain : Layer 2 « Plasma »

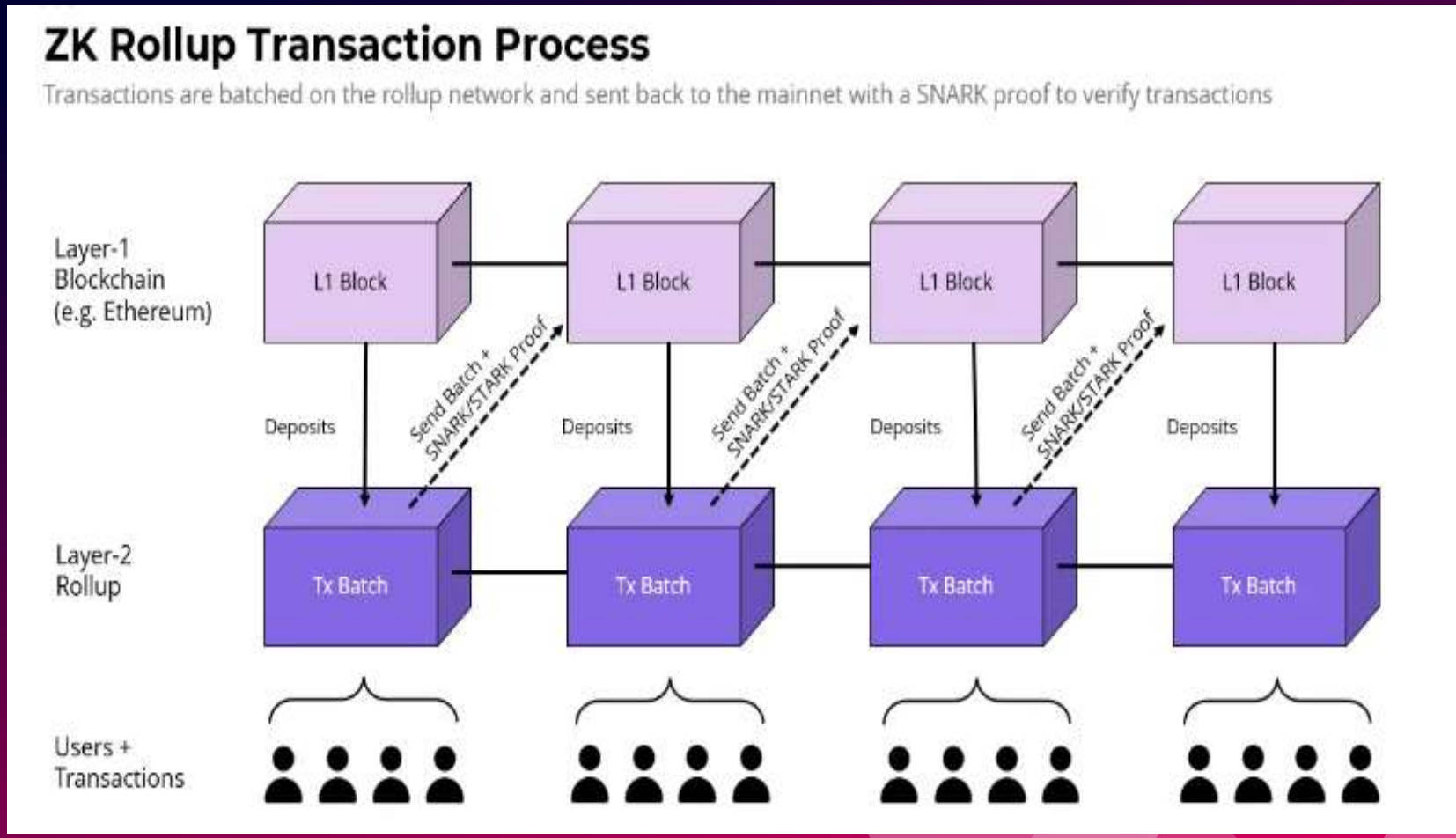
- Proposé par V. Buterin et Joseph Poon
- Duplication de la *parent chain*
- Décharge des tx de la *parent chain*



polygon

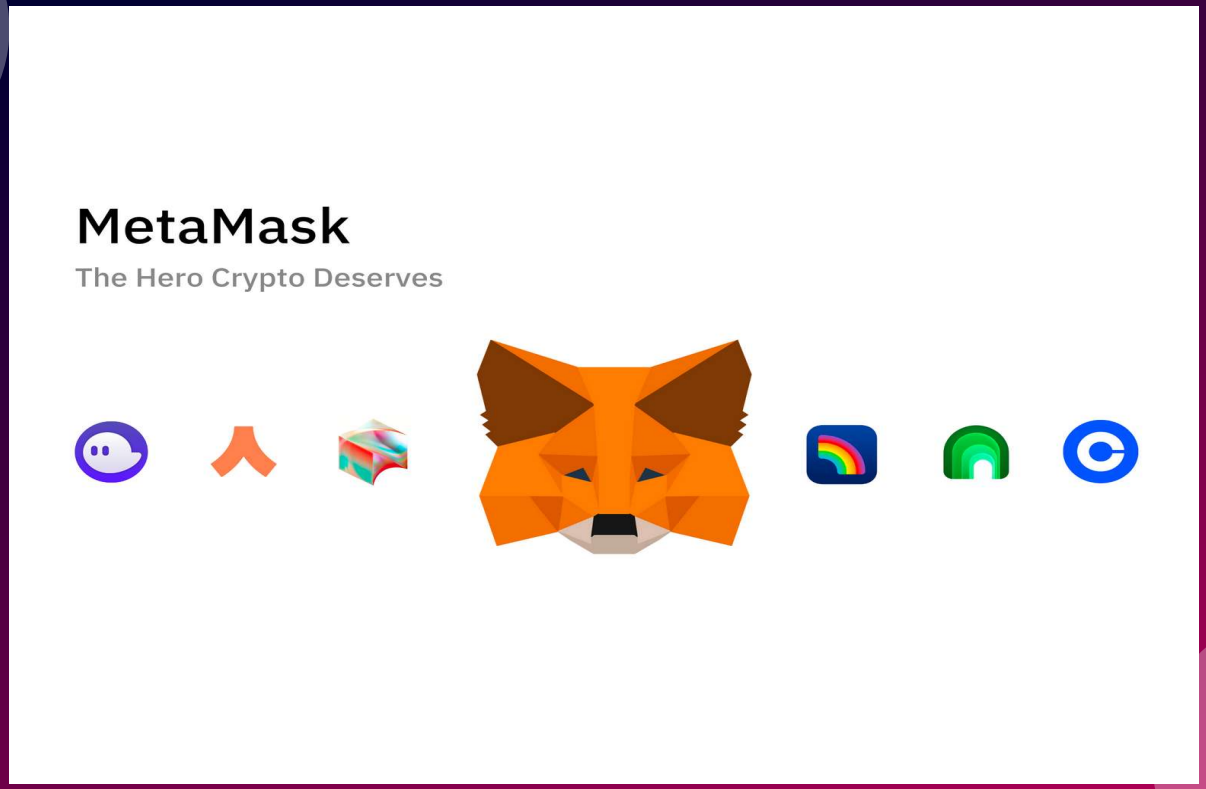
Blockchain : Layer 2 « ZK Rollups»

- Solution de scaling
- Aggrège de multiples Tx off-chain avec compression par Merkle Root
- Renvoie la preuve cryptographique on-chain via les « relayers » (Snark)



Blockchain : Layer 2 « ZK Rollups»

Wallet Crypto



Wallet Crypto

- Installer Metamask sur browser : https://metamask.io/download
- Créer 3 comptes à partir de la clé privée
- Ajouter le réseau testnet de Fantom : https://chainlist.org
- Obtenir des FTM de testnet sur le faucet officiel : https://faucet.fantom.network/
- Envoyer des FTM entre les différents comptes
- Observer le résulat dans l’explorateur de blocks : https://testnet.ftmscan.com/

Test Smart Contract : RemixIDE (https://remix.ethereum.org/)

Github :
https://github.com/gilbruno/SolidityProjects/blob/master/Voting.sol

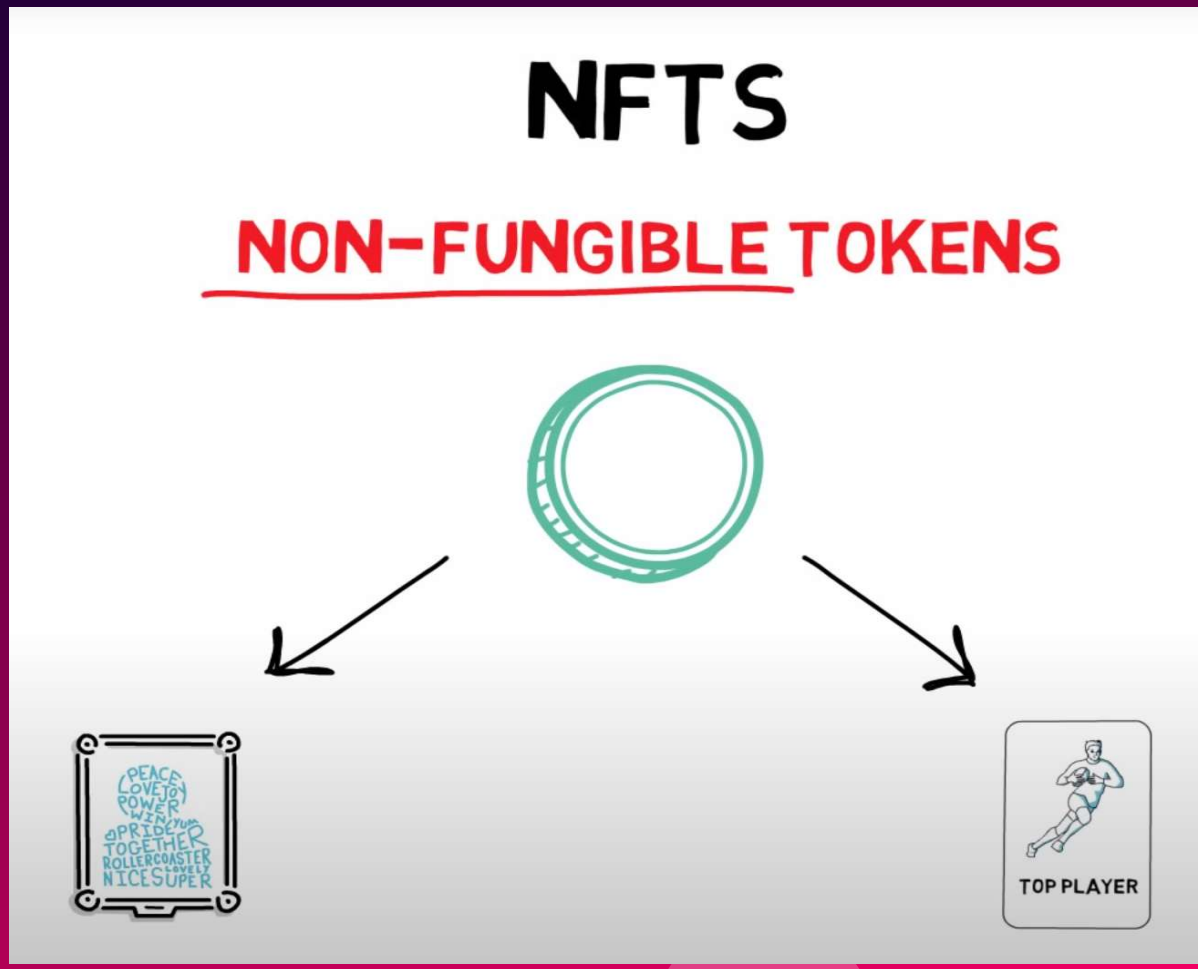
et

https://github.com/gilbruno/SolidityVoting

Puis test via Metamask

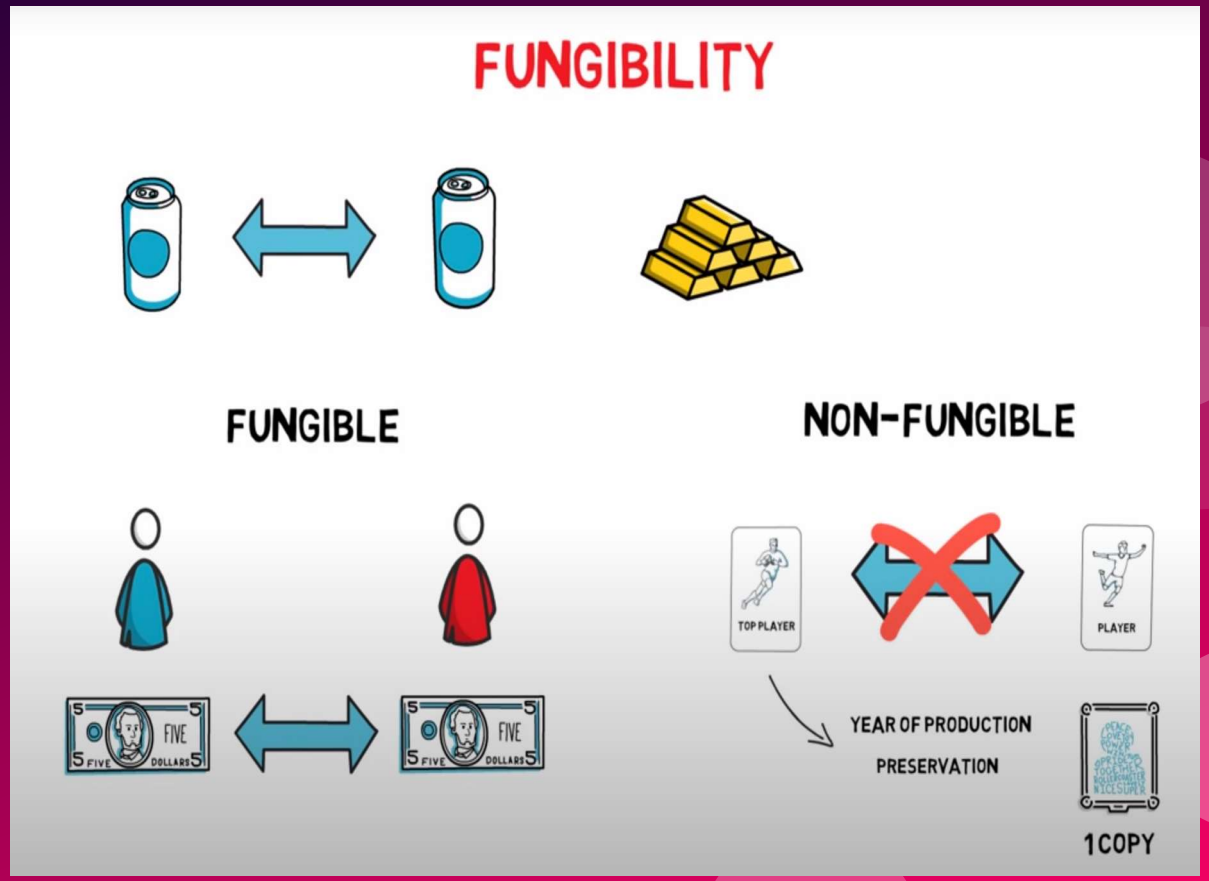
Blockchain : Les NFT

- Autre type de token : Non Fungible Token
- Propriété cryptographique de Non Fungibility
- S’applique principalement pour les œuvres d’art numérique et *collectible*



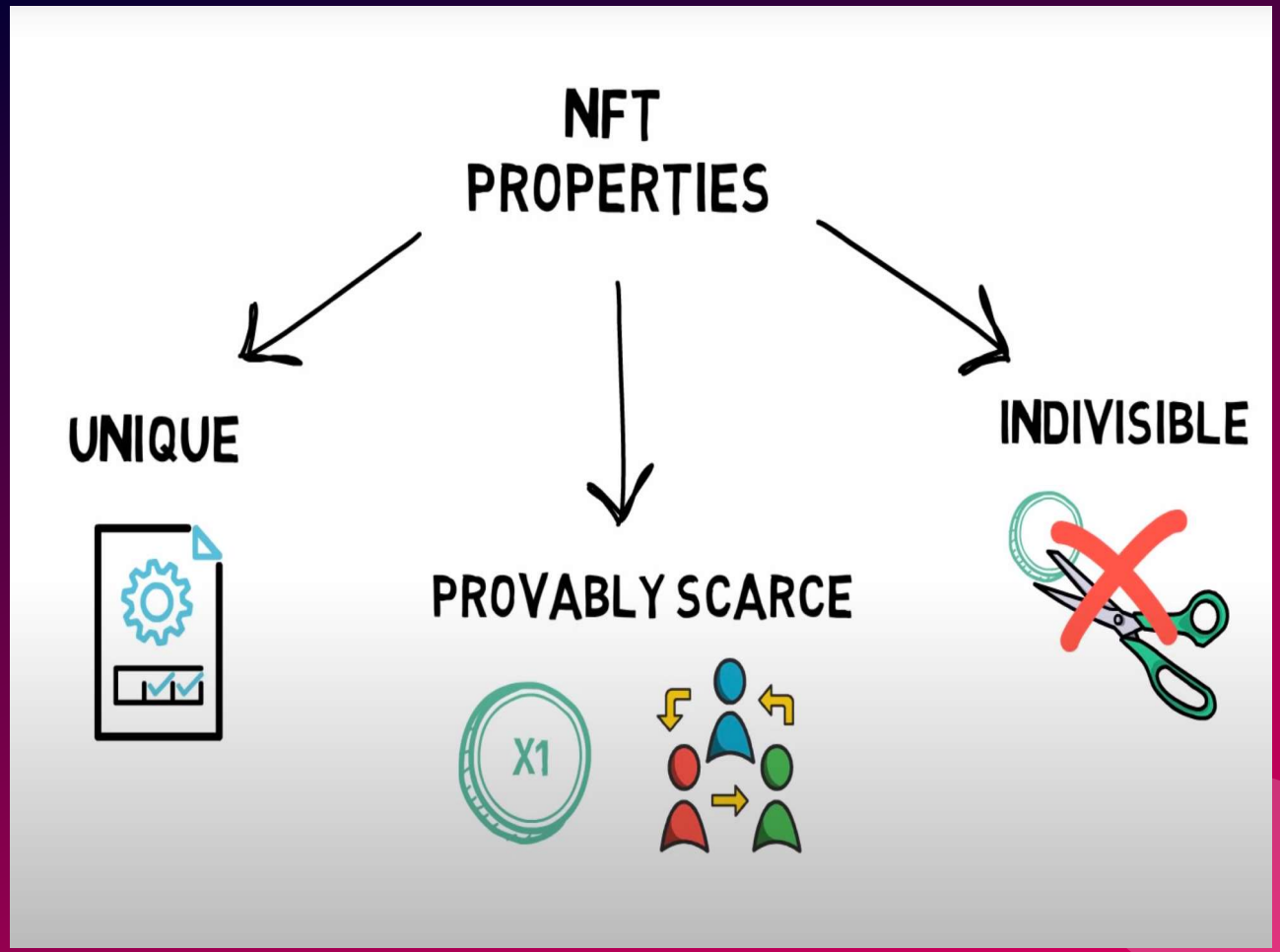
Blockchain : Les NFT

- Propriété qui confère un caractère non unique, interchangeable avec une autre unité sans altérer le posséseur de cette unité
- Exemple fungible : billet de banques
- Exemple : Non fungible (carte pokémon)



Blockchain : Les NFT

- Propriétés des NFTs
- Unique, rare, indivisible

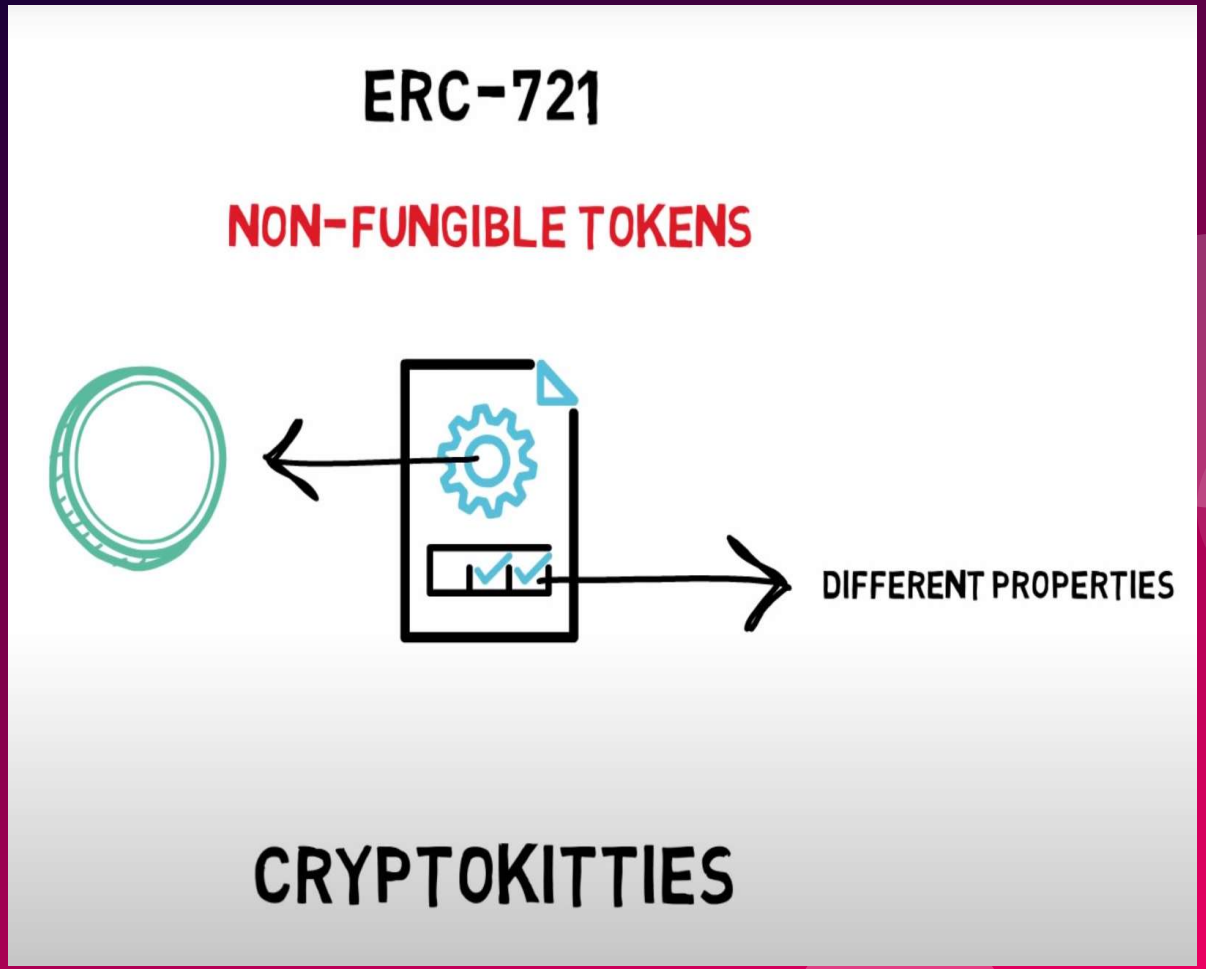


Blockchain : Les NFT

- Standardisé par EIP avec les normes ERC-721 et ERC-115
- https://eips.ethereum.org/EIPS/eip-721
- ERC20 ⇒ Standard pour créer un token sur la BC ETH (Ex : Stable coin)

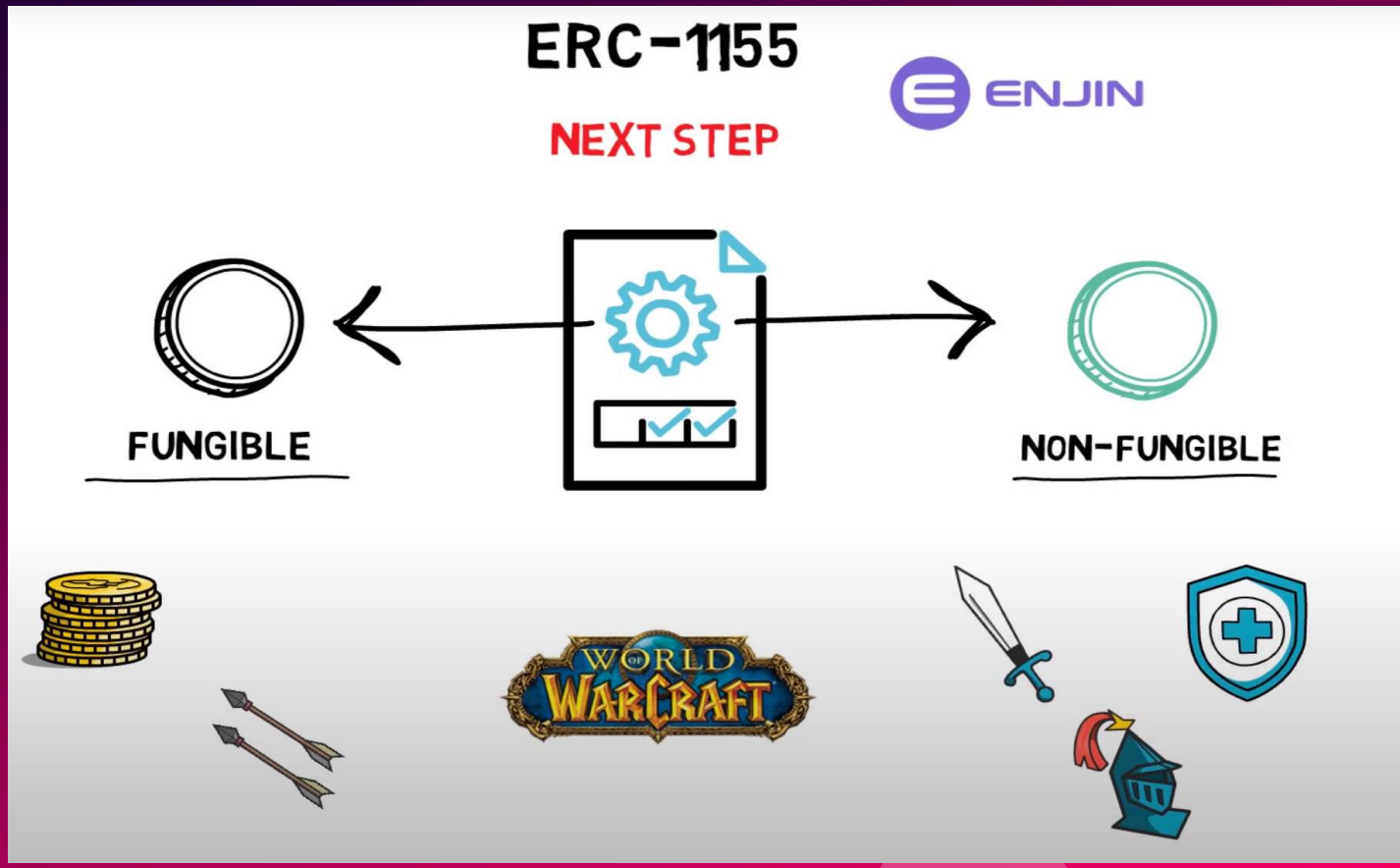
Blockchain : Les NFT ERC-721

- Standard pour permettre à un Smart contract de créer un NFT purement non fongible
- Pionniers : CryptoKitties
- Marketplace : Opensea (https://opensea.io/)



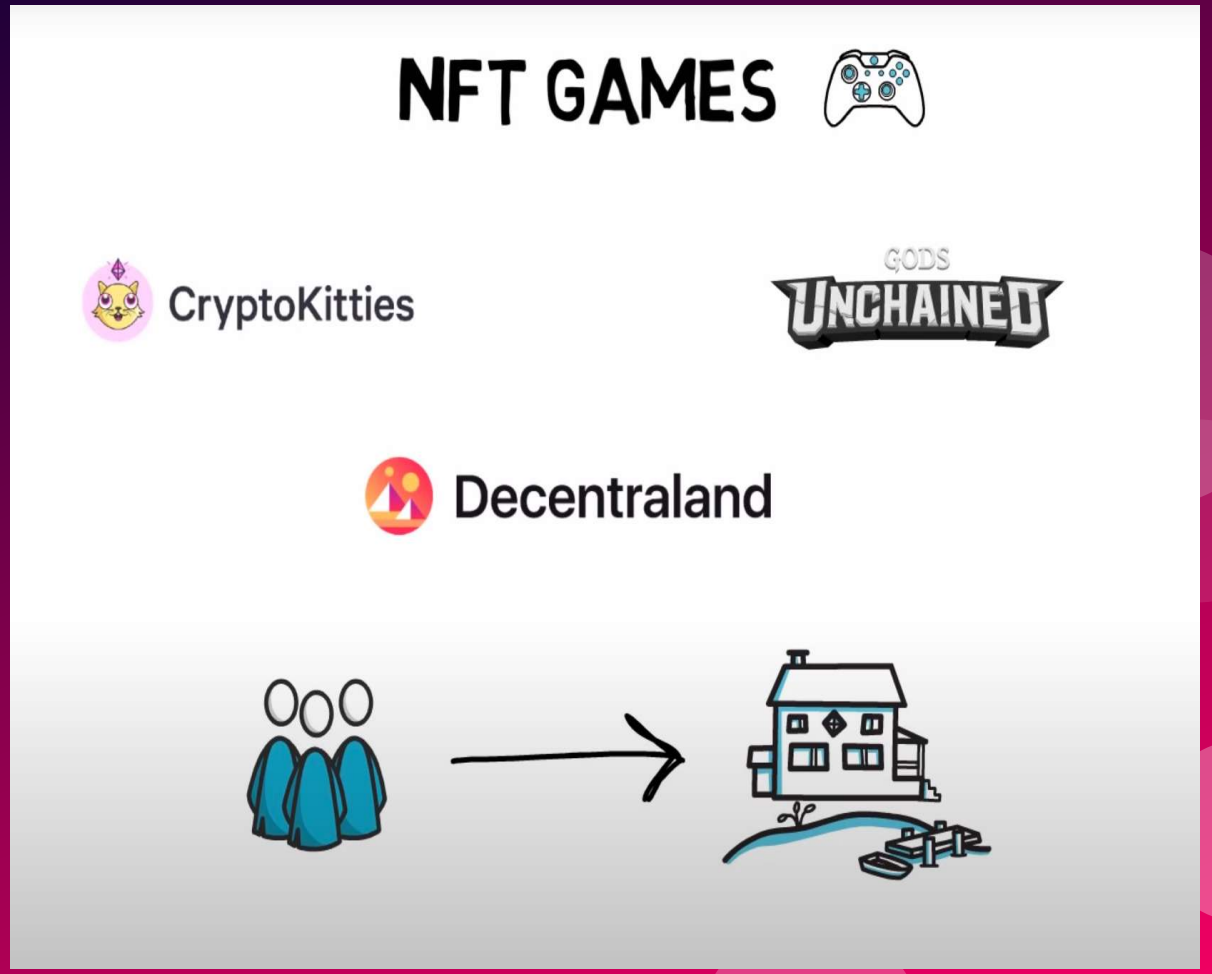
Blockchain : Les NFT ERC-1155

- Standard pour permettre à un Smart contract de créer un NFT mix de non fongible et fongible



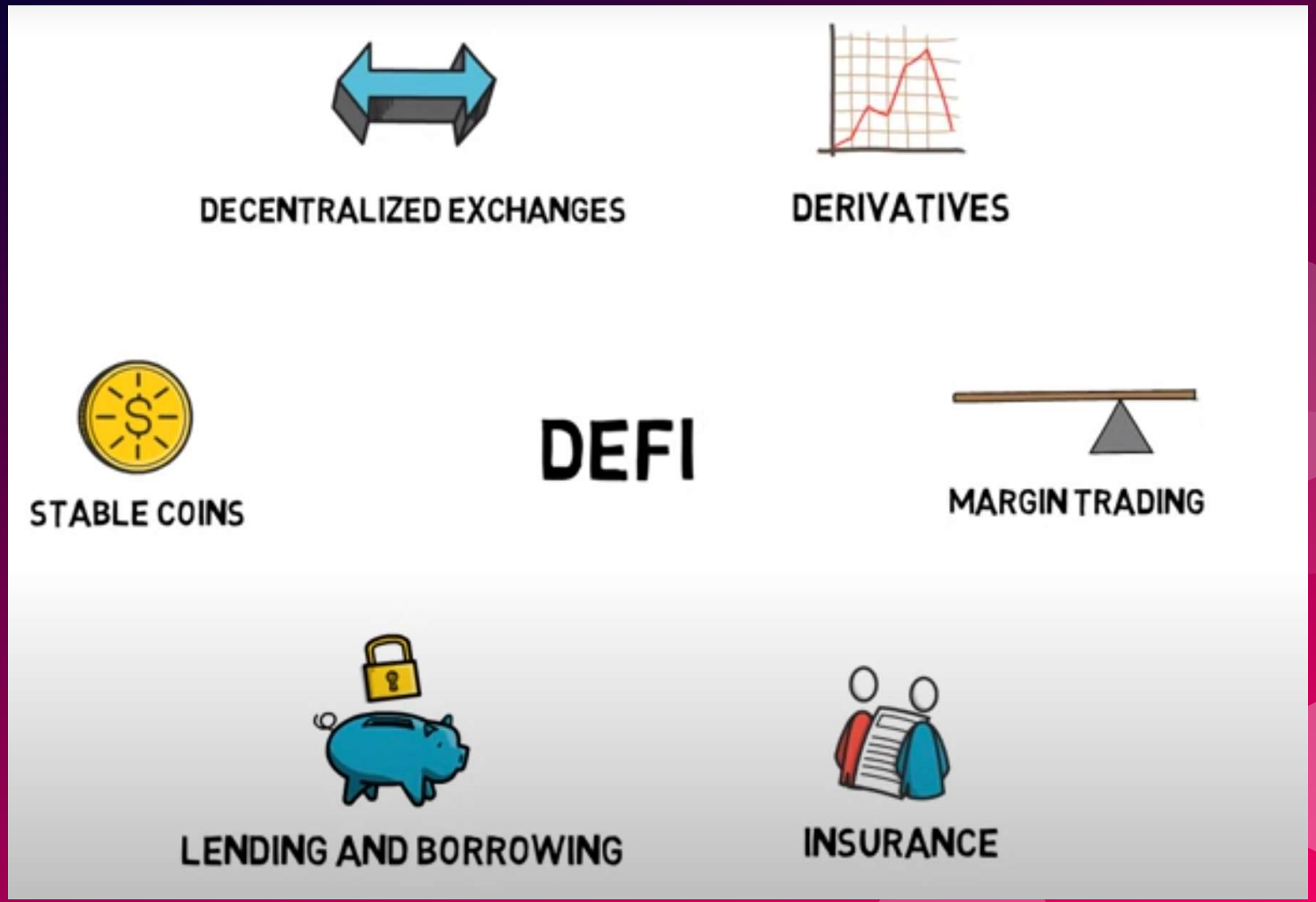
Blockchain : Les NFT et Dapps

- Très utilisé dans le domaine de jeu videos
- Decentraland : permet d’acheter un terrain virtuel
- https://www.phonandroid.com/metavers-un-bout-de-terrain-virtuel-se-vend-24-millions-de-dollars-un-record.html
- https://cryptonaute.fr/nouveau-record-cryptokitties-600-ethereum/
-



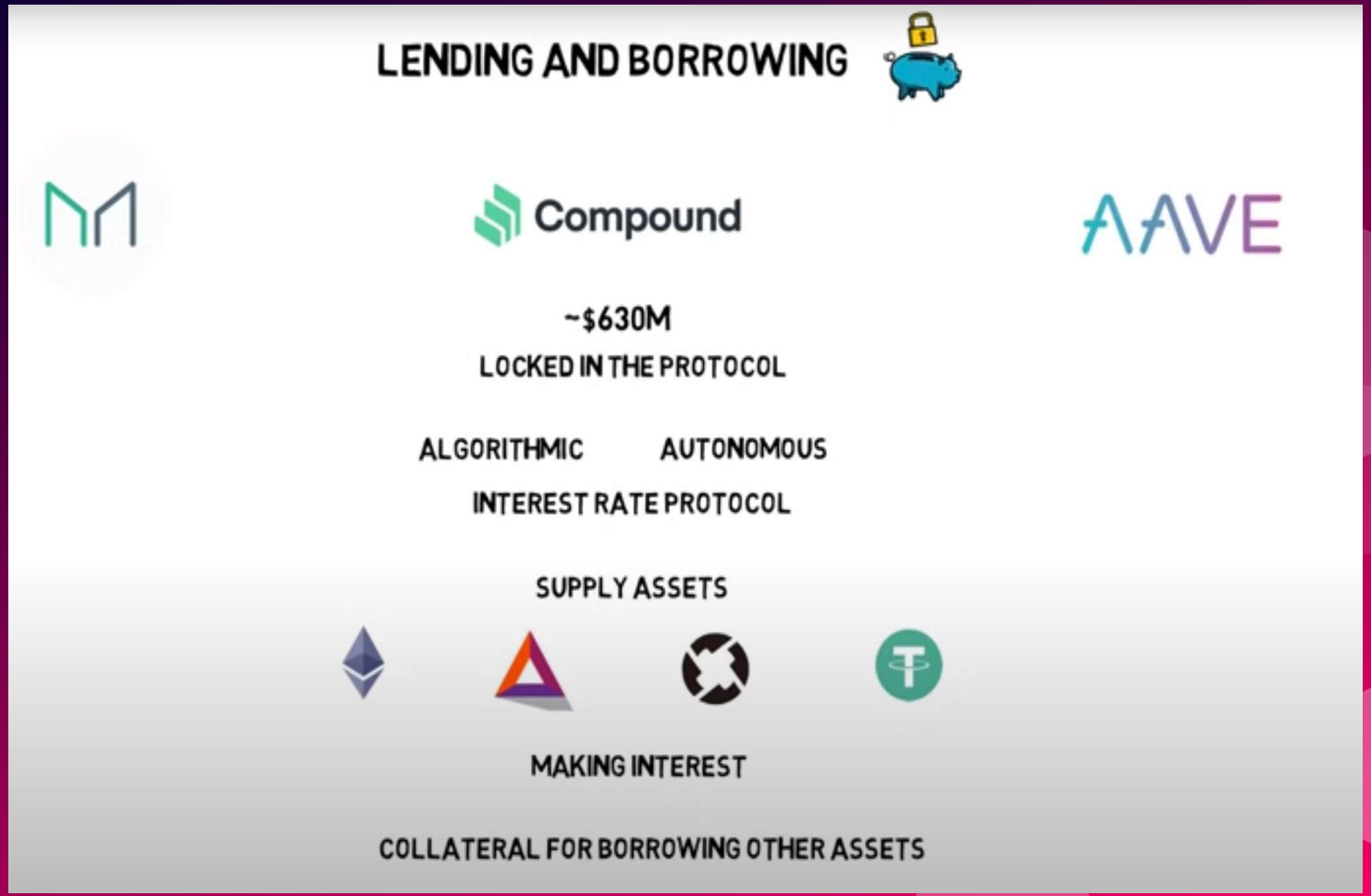
Blockchain : La DeFi

- Définition Finance décentralisée
- Fournit les services de la TradFi sans intermédiraire, avec une totale décentralisation
- Lending, Borrowing, Stable coins, etc ...



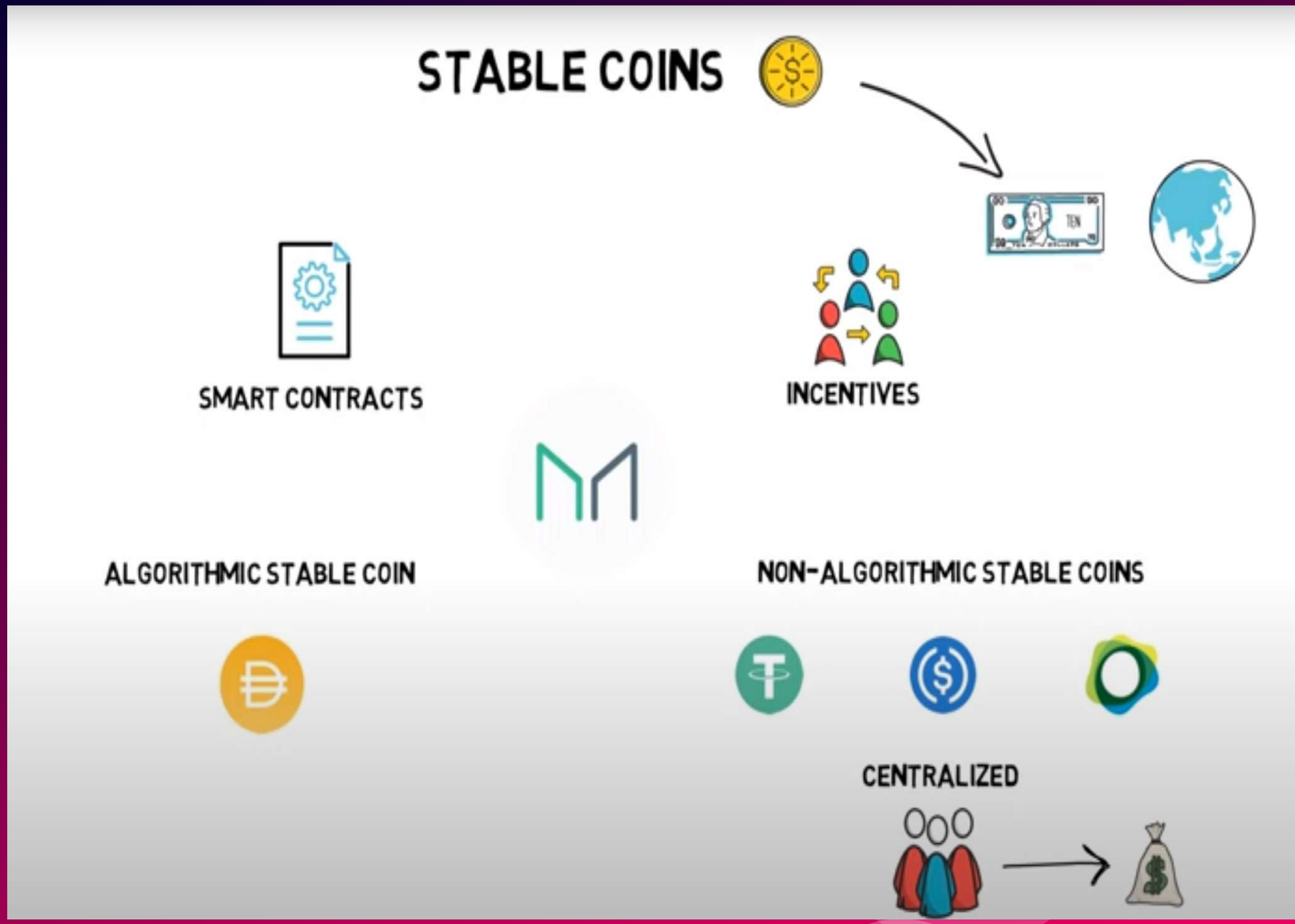
Blockchain : La DeFi

- MakerDAO pionnier ⇒ Stable coin DAI
- Compound & Aave : Lending & Borrowing
- Permet un accès plus facile au crédit que la tradfi



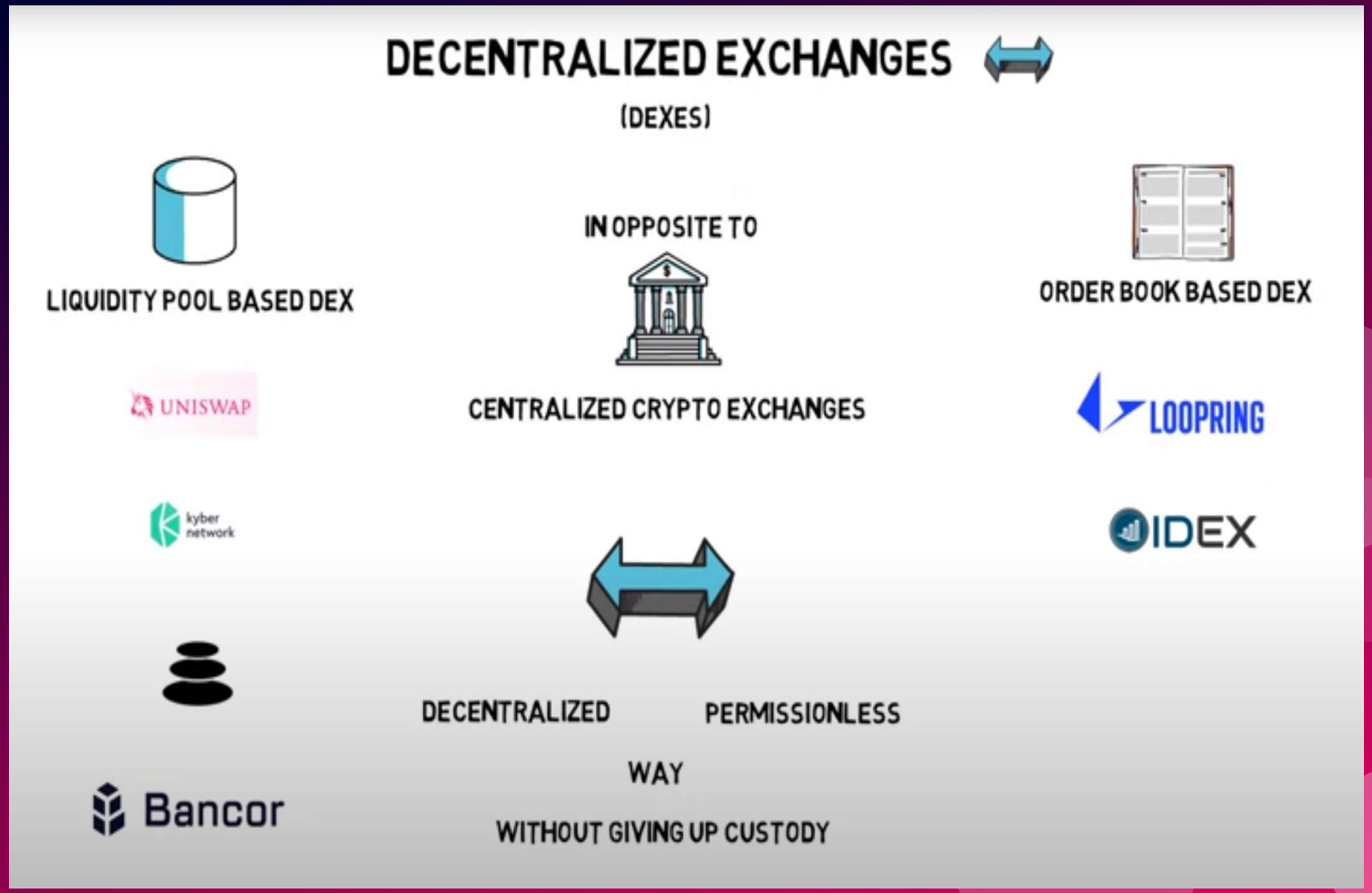
Blockchain : La DeFi

- MakerDAO pionnier ⇒ Stable coin DAI
- Lock ETH ⇒ Mint DAI (avec intérêt)
- DAI est algorithmique =/ USDT non algo
- DAI est backé par de la crypto



Blockchain : La DeFi

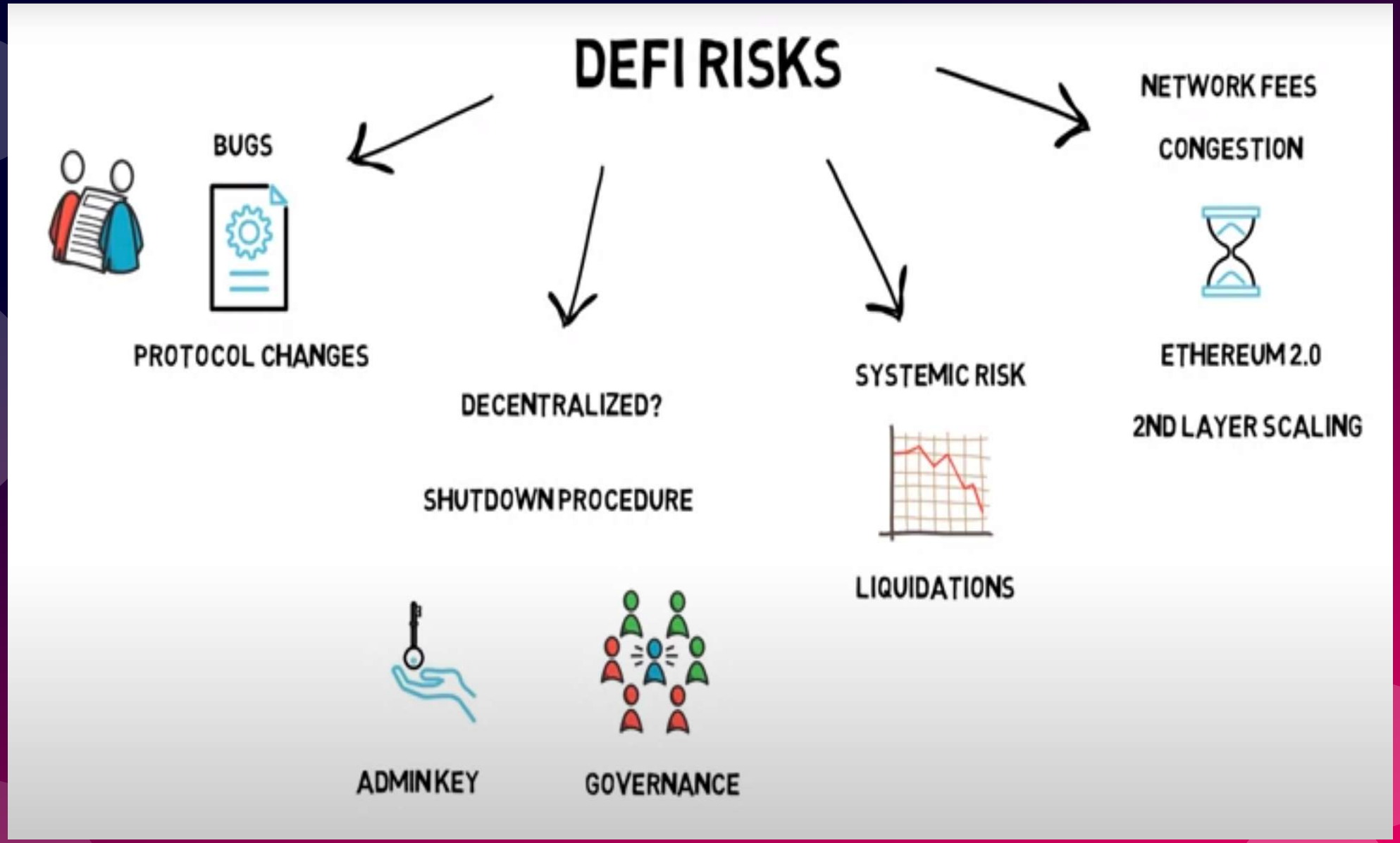
- DEX > CEX
- CEX célèbre : Binance, FTX (RIP), Bybit



Blockchain : La DeFi

DEFI	CEFI
PERMISSIONLESS	PERMISSIONED
 OPEN	CLOSE 
CENSORSHIP-RESISTANT	CAN BE CENSORED
CHEAPER	EXPENSIVE
BUILT ON THE BLOCKCHAIN	BUILT ON OLD FOUNDATIONS

Blockchain : La DeFi et ses risques



Blockchain : La DeFi (conclusion)

- Technologie disruptive
- Open, permissionless, DAO (démocratique)
- High-risk / High-reward
- Est présente sur quasi toutes les BC
- TradiFi killer

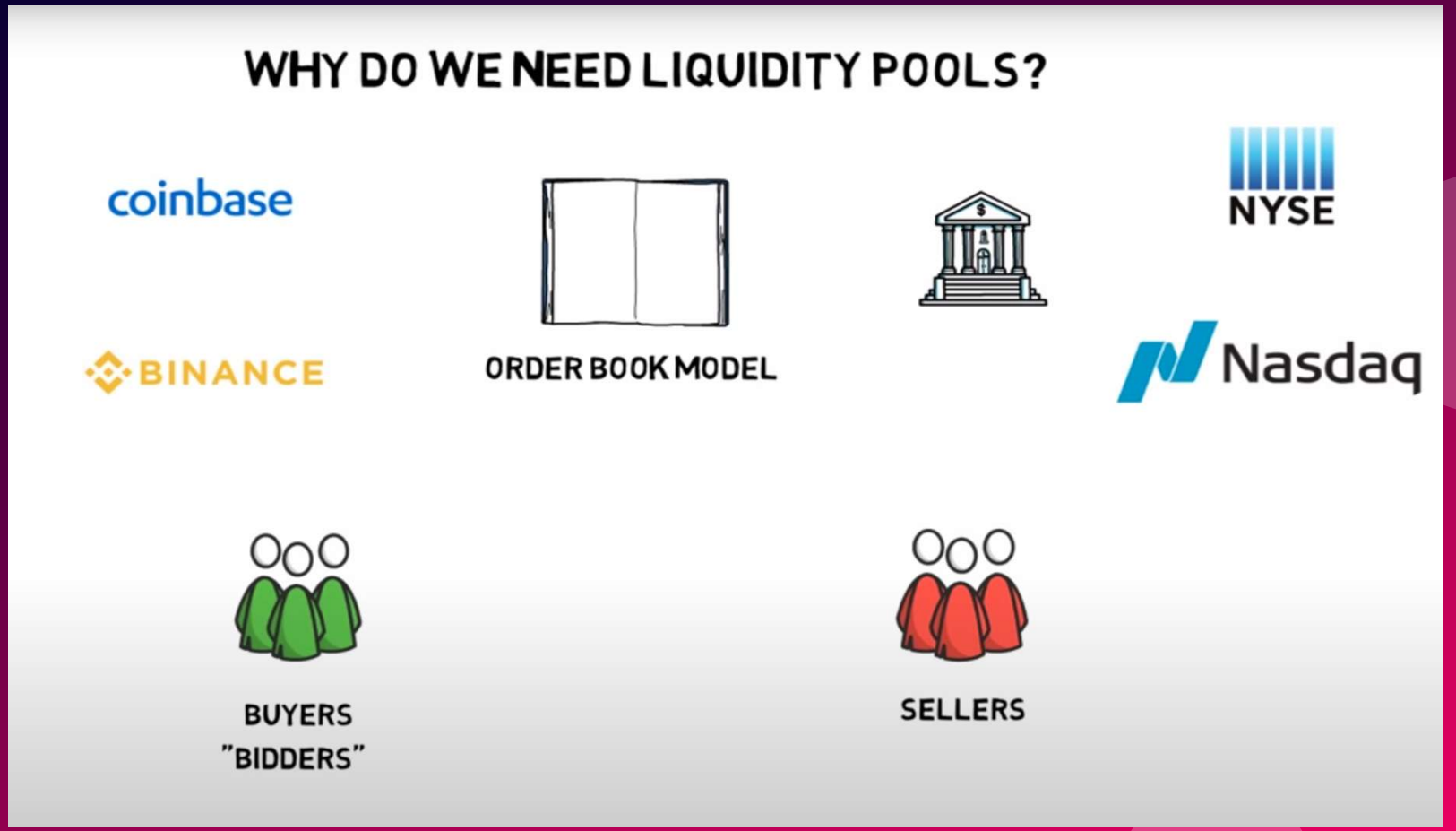
Les projets cryptos

Les projets cryptos

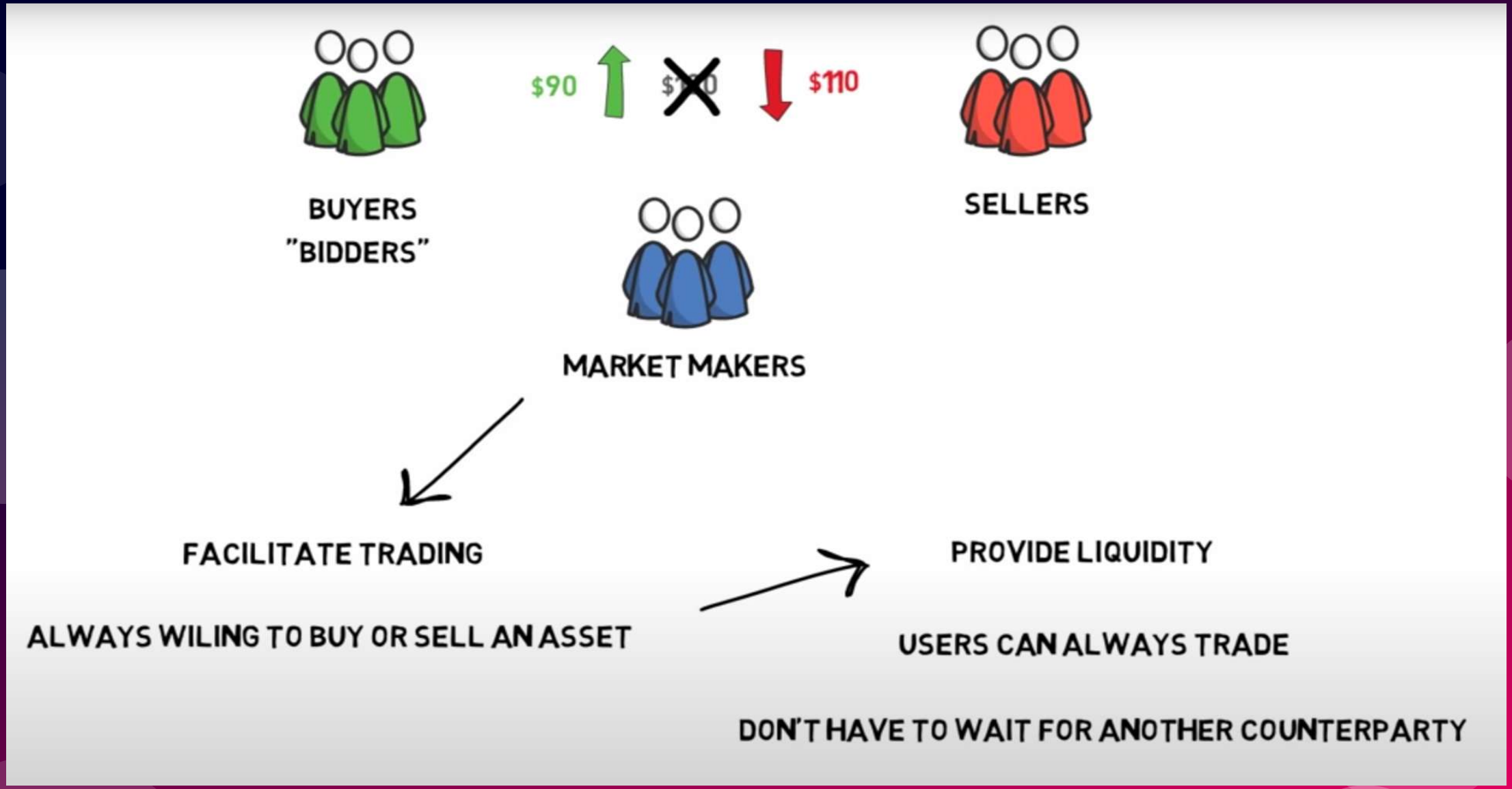
- Pionnier dans l’interopérabilité entre BC
- Propose un SDK (Tendermint) poyur créer une BC rapidement
- Créé par Jae Kwon en 2014
- ICO en 2017
- Internet des BC

Blockchain : Liquidity Pools

- CEX tels que Binance ou ou Crypto.com == Brokers Nyse / Nasdaq



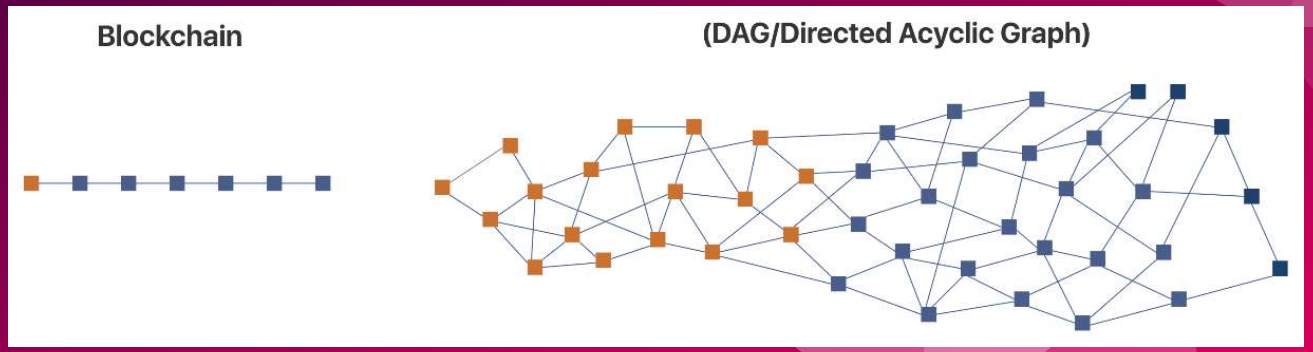
Blockchain : Liquidity Pools



Les projets cryptos

Les projets cryptos

- Layer1 avec objectif de scalabilité
- BC différente : DAG (Directed Acyclic Graph)
- Eth Killer
- Consensus « Lachesis » spécifique (PoS)
- EVM Compatible



Les projets cryptos

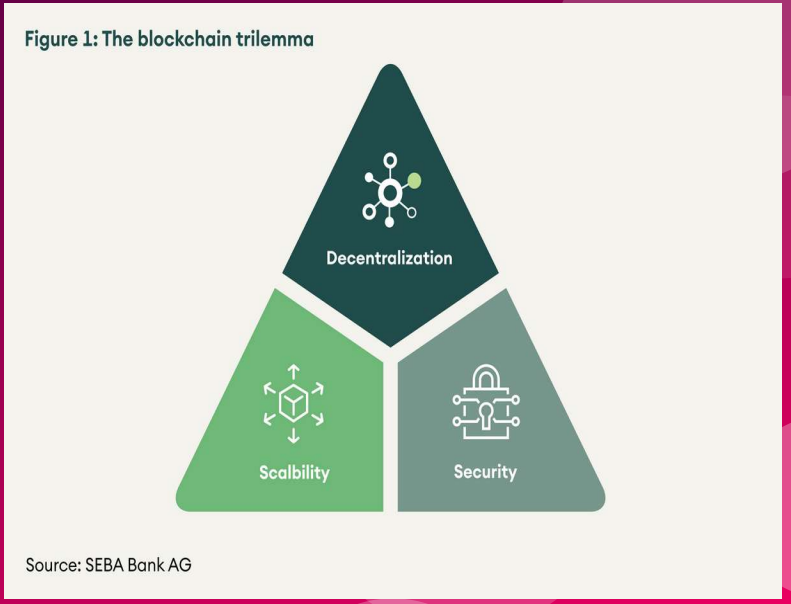
Les projets cryptos

- Lancé en 2014
- Token natif : XMR
- BC focalisée sur l’anonymat
- Impossible d’identifier les émetteurs et destinataires
- Consensus PoW
- 18M de XMR ~
- Inflation Rate de 0,86 % / an

Les projets cryptos

Les projets cryptos

- Le « sauveur » d'ETH ⇒ Layer 2
- BC différente : DAG (Directed Acyclic Graph)
- Eth Killer
- Trilemme de la BC : Scalabilité / Décentralisation / Sécurité
- Basé sur les « plasmas »
- Fragments de la BC principale dupliquée
- Soulage la BC principale



Les projets cryptos

Les projets cryptos

- Permet une interopérabilité entre BC

Les projets cryptos

Les projets cryptos

- Protocole de prêt et emprunt
- Créé en 2017

Les projets cryptos

Les projets cryptos

- BC dédiée au stockage décentralisé des données
- Permet à qui le souhaite de prêter son espace de stockage
- Sans entité centralisée pour gérer ça
- Repose sur le protocole IPFS
- « Loueur » payé en FIL

Les projets cryptos

Les projets cryptos

- A démocratisé la Defi
- 1er DEX sur la BNB Chain

FIN ...