

# Introduction aux Blockchains

---

Rohan Fossé - 09 Mai 2022

# Domaines dans la blockchain

- Cryptographie
- Réseau
- Bases de données
- Systèmes distribués
- Langages de programmation / Vérification formelle

# Blockchain : définition

- Base de données distribuée : chaque utilisateur possède une copie locale d'un registre ;
- Les utilisateurs forment un réseau pair-à-pair et s'échangent des blocs (agrégats d'opérations) pour faire évoluer ce registre ;
- Chaîne de blocs: chaque nouveau bloc doit succéder à un bloc existant et connu du réseau ;
- Protection contre la falsification: tous les participants sont tenus de vérifier la validité des blocs reçus ;
- Implémentation connue : toute personne doit pouvoir vérifier le fonctionnement.

# Une application : les crypto-monnaies (1/3)

Monnaie digitale décentralisée :

- Le registre de la blockchain est un *livre de comptes* associant une adresse à un solde en *jetons* ;
- Les comptes sont des objets cryptographiques permettant de signer des transactions et d'en vérifier leur validité.
- Les participants ont une **incitation économique** à participer au réseau : e.g. récompense à la création d'un bloc ;

**Hypothèse** : les participants accordent une valeur aux jetons.

⇒ Les actions nuisibles au réseau doivent se traduire par une perte économique.

# Une application : les crypto-monnaies (2/3)

## **Critique des systèmes bancaires**

- Centralisation: une seule autorité
- Système en boîte noire
- Aucune garantie de fiabilité

## **Avantages de la blockchain**

- Décentralisation : chacun participe au système ;
- Sécurité : chaque participant valide localement les changements et peut rejouer l'état ;
- Transparence : chaque action effectuée est connue de tous les participants.

# Une application : les crypto-monnaies (3/3)

## Problèmes ouverts

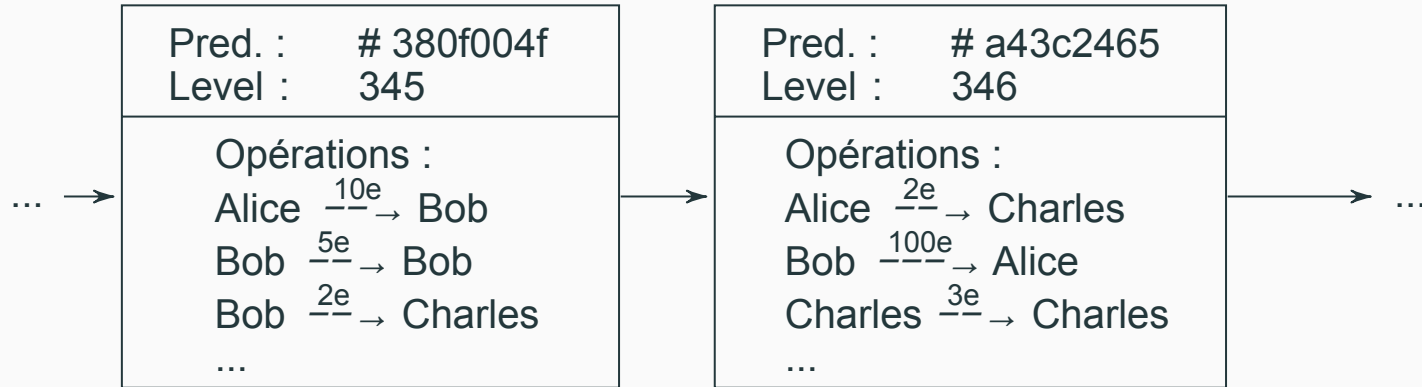
- *Privacy* :
  - L'historique de chaque personne est accessible : comment préserver l'anonymat ?
- Passage à l'échelle :
  - Débit réduit d'opérations ;
  - Rapidité du réseau pair-à-pair.
- Dissensions sociales :
  - Fonctionnement et évolution du réseau.

# Quelques projets de cryptomonnaies

- Bitcoin : 2009 ⇒ Bitcoin Cash, ...
- Ethereum : 2014 ⇒ Ethereum Classic, ...
- Dogecoin : 2015
- Zcash : 2016
- Tezos : 2018
- ...

# Structure d'un *block*

## Agrégats d'opérations ordonnés





# État de la blockchain

- L'état (ou registre) d'une blockchain est stocké dans une base de données
- La base de données est modifiée après chaque application de bloc
- L'application d'un bloc revient à appliquer l'ensemble des opérations contenues

# Application d'un bloc

Bloc

Pred. : # 380f004f
Level : 345
Opérations : Alice $\xrightarrow{10e}$ Bob Bob $\xrightarrow{5e}$ Bob Bob $\xrightarrow{2e}$ Charles

État initial

Alice	1293e
Bob	5432e
Charles	543e

État après application du bloc

Alice	1220e
Bob	5440e
Charles	545e

**Il faut vérifier ce que l'on reçoit !**

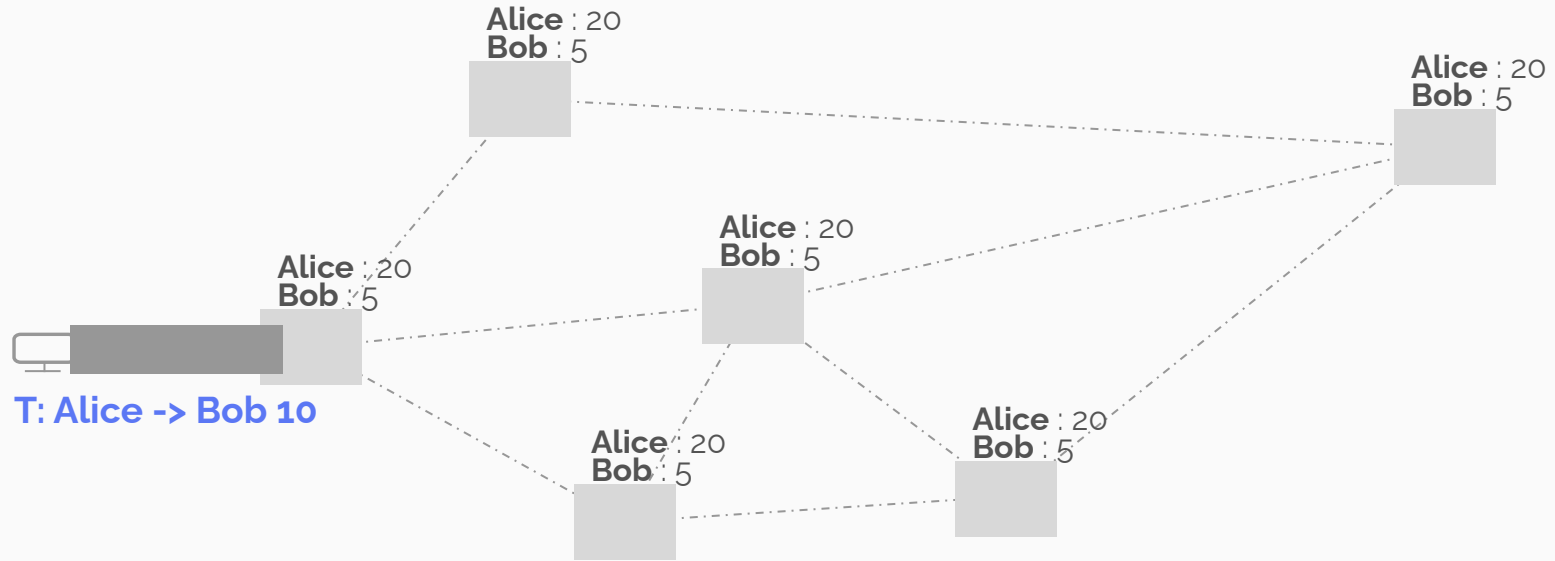
# Base de donnée répliquée

- Chaque utilisateur possède une copie locale de l'état de la blockchain
- Cela permet de vérifier localement ce que les autres utilisateurs envoient
- Cela permet également de créer des transactions cohérentes avec l'état actuel de la chaîne

# Réseau pair-à-pair

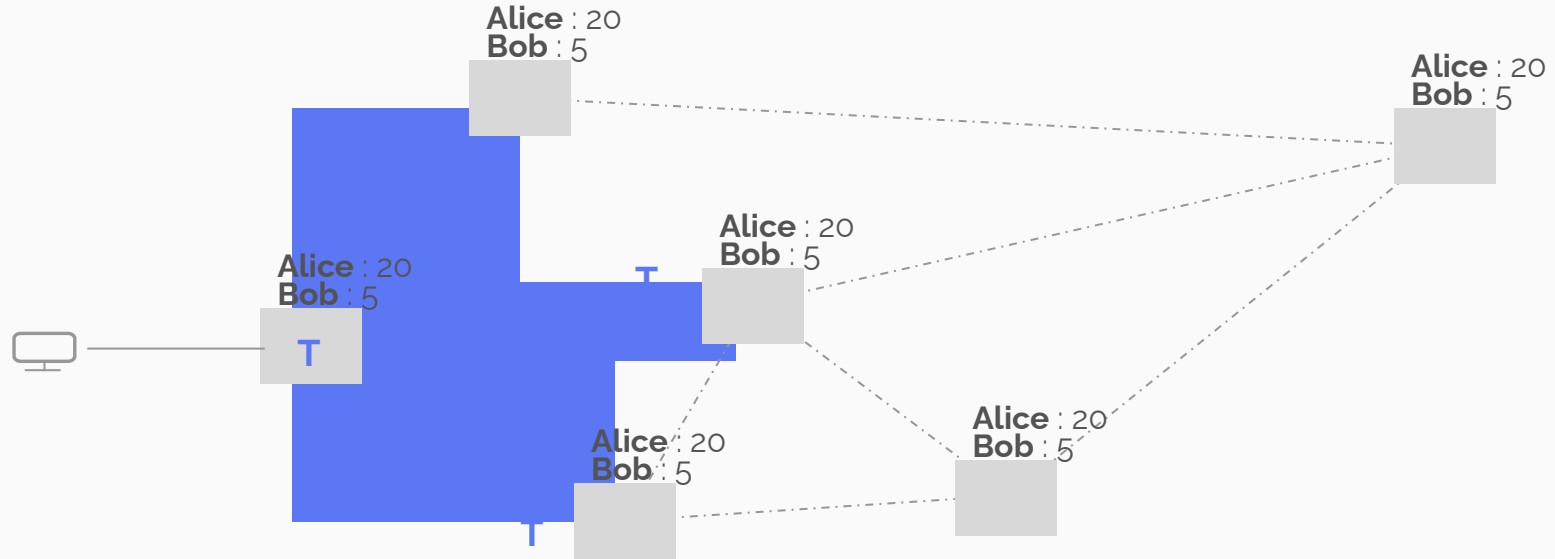
- Les utilisateurs doivent rester synchronisés entre eux
- Ils communiquent via un réseau pair-à-pair en utilisant un **nœud**
- Chaque participant se connecte à un nœud agissant comme un client pair-à-pair
- Plus il y a de nœuds dans le réseau, plus le réseau est sûr

# Propagation dans le réseau



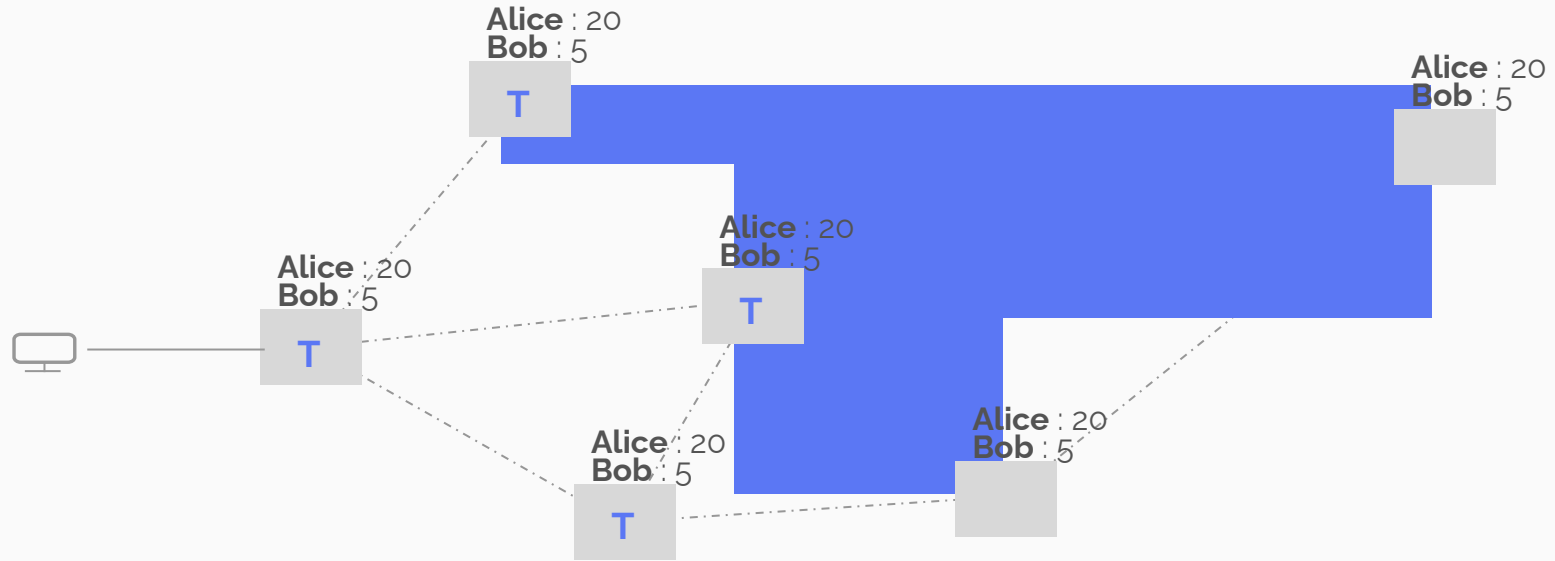
L'utilisateur crée et envoie une transaction à son nœud

# Propagation dans le réseau



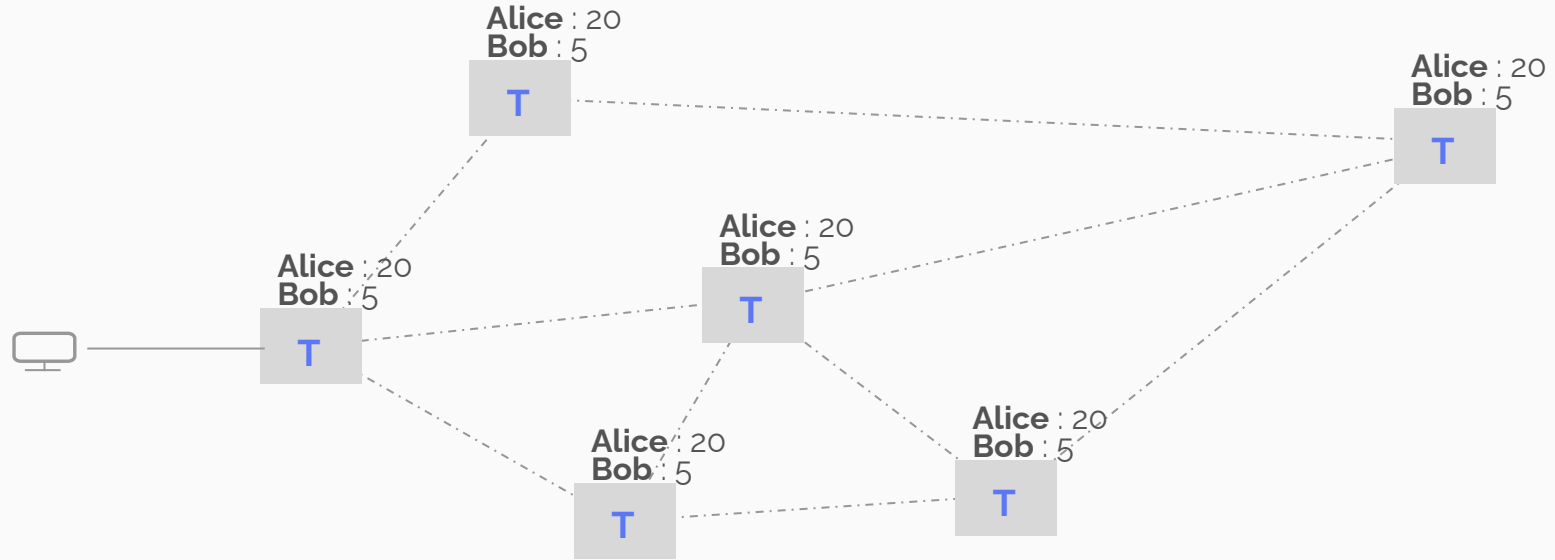
Le nœud propage la transaction à ses pairs qui la mette dans leur liste de transactions

# Propagation dans le réseau



Les autres nœuds informent à leur tour leur pairs...

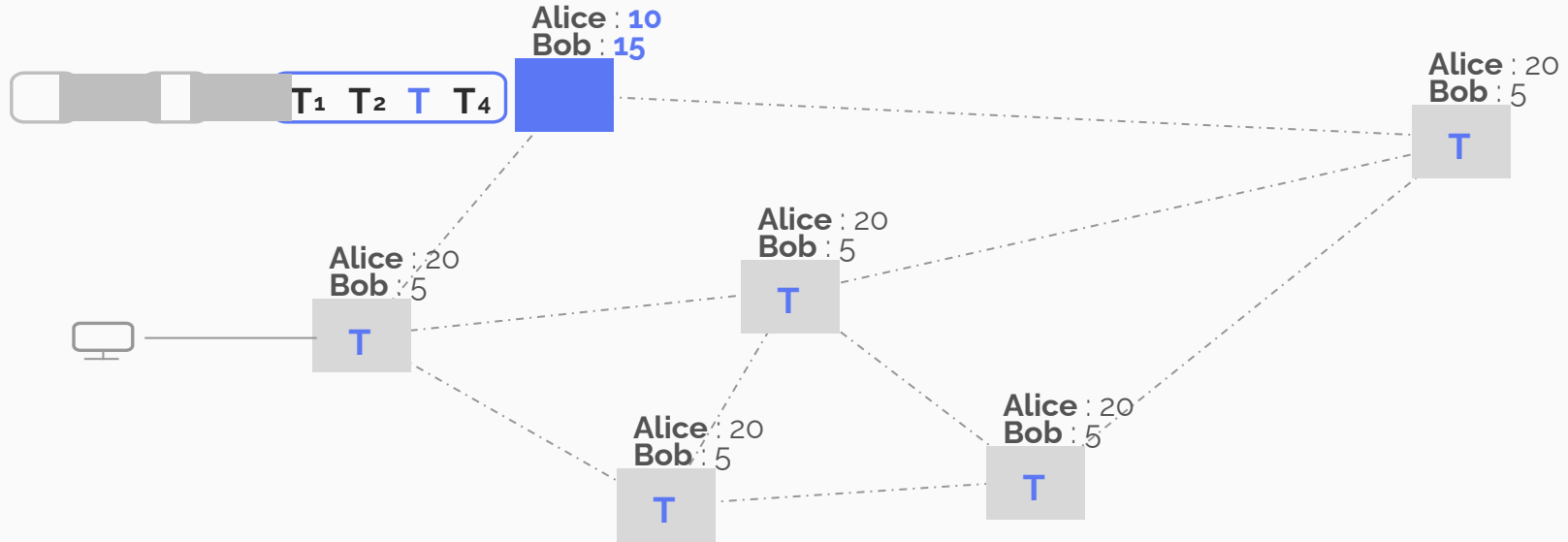
# Propagation dans le réseau



... jusqu'à la propagation globale dans le réseau



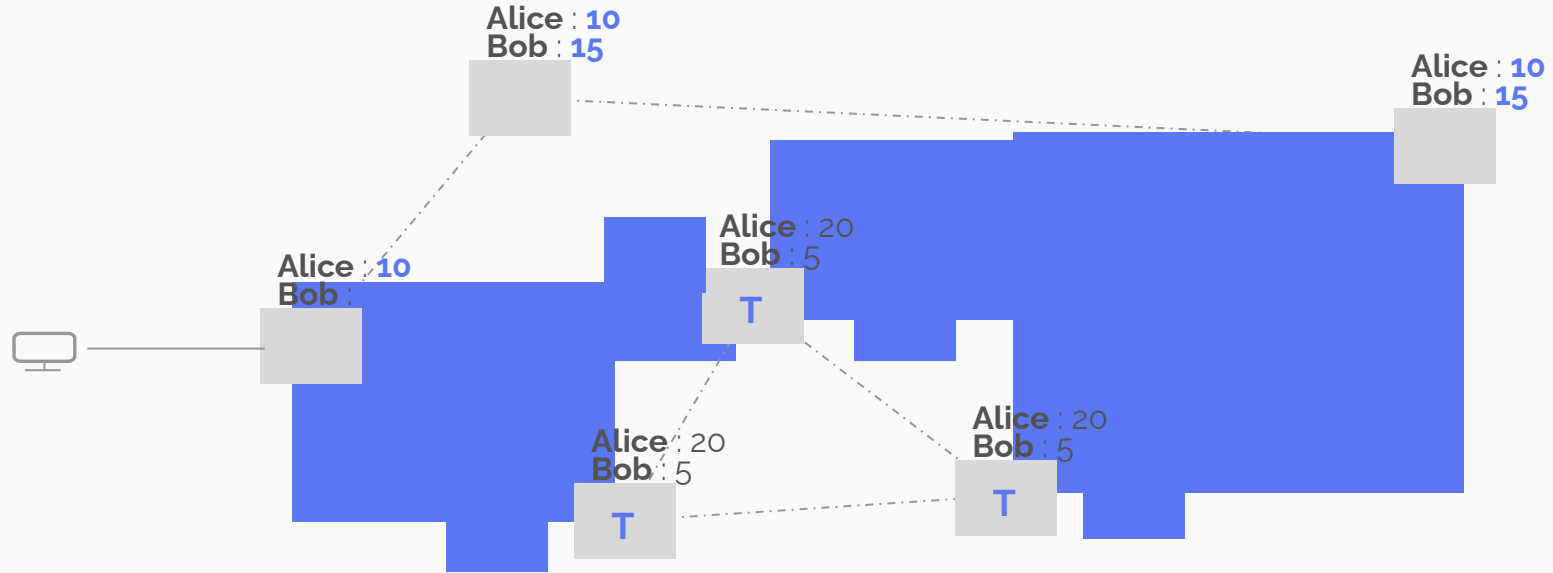
# Propagation dans le réseau



À un moment, un nœud va décider de créer un bloc avec les transactions reçues

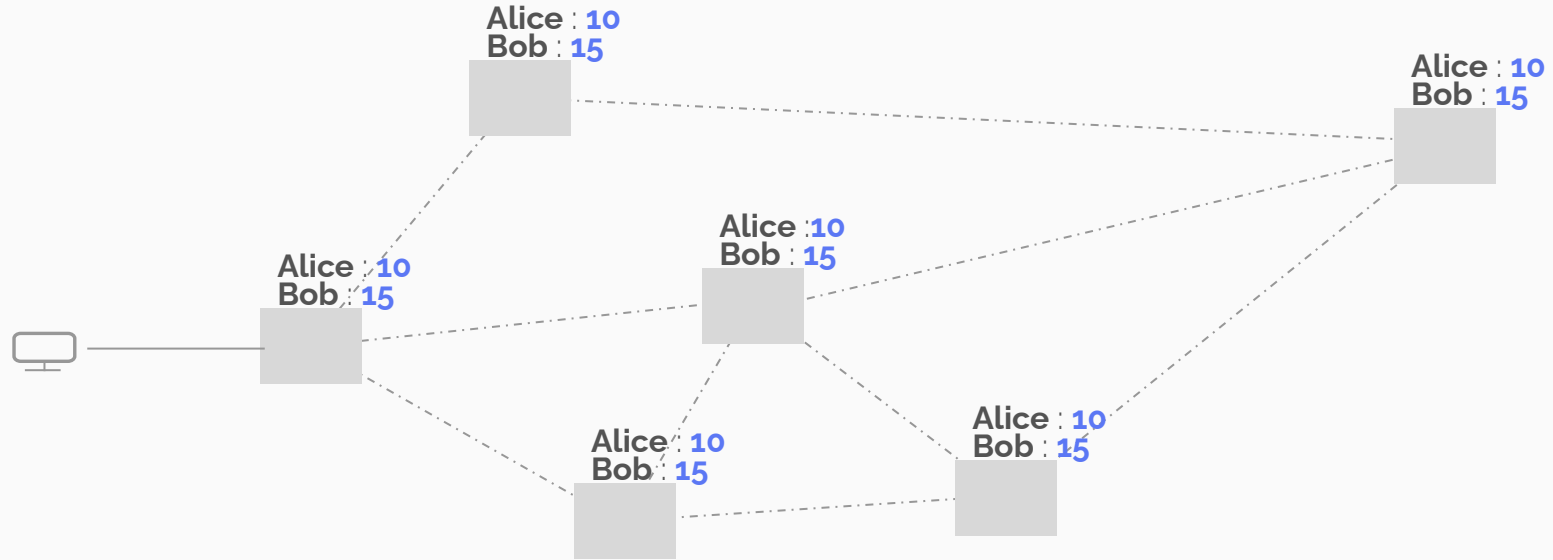


# Propagation dans le réseau



Chaque participant vérifie et applique localement ce bloc à sa base de données

# Propagation dans le réseau



Au bout d'un moment, chaque participant a reçu le bloc et la majorité du réseau est à jour

# Architecture du système

Une blockchain a besoin de deux composants principaux :

- Un **nœud**
- Un **protocole économique**

# Caractéristiques du nœud

## Une couche réseau et un stockage

- Couche réseau P2P : *Gossip*
  - Propagation des opérations
  - Propagation des blocs
  - Découverte de pairs réseau
- Stockage :
  - Stocke les blocs et les opérations pour les envoyer au besoin aux autres participants
  - Maintient les états passés de la chaîne pour être capable de revenir en arrière au besoin

# Protocole économique

Le protocole économique est l'ensemble des règles de la chaîne  
Chaque participant accepte de suivre ses règles pour maintenir  
le bon comportement du réseau

Il détermine notamment :

- la validité des transactions
- la validité des blocs
- attribue un score aux blocs appliqués
- les récompenses à la participation
- ...

# Interaction entre le nœud et le protocole

- Le protocole valide les blocs et les opérations
- Le nœud stocke les blocs et les états de la chaîne

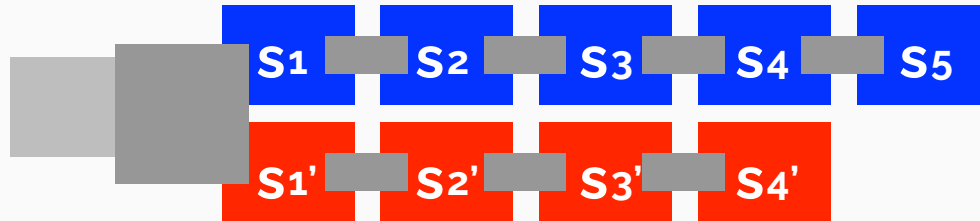
## Scénario : un nouveau bloc arrive du réseau

1. Le nœud demande au protocole de valider ce bloc ;
2. Le protocole valide le bloc et retourne un **score** ou si le bloc est invalide rejette le bloc et *kick* au besoin le pair ;
3. Le nœud teste si le nouveau **score** est meilleur que le score du bloc résultant de son état courant :
  - Si oui : le bloc est sélectionné comme nouvelle tête de chaîne et propagé dans le réseau,
  - Sinon : le bloc est ignoré.



# Problème des chaînes concurrentes

Il est possible qu'à un moment le réseau ne soit plus d'accord sur l'état de la chaîne : quel bloc est en tête ?



Pour résoudre ce problème, il est nécessaire d'établir un **algorithme de consensus** que tous les participants sont encouragés à appliquer pour le bon fonctionnement du réseau.

# Algorithme de consensus

Un algorithme de consensus doit définir deux notions clés :

- Établir un score à chaque bloc
- Déterminer la validité d'un bloc

Mais pourquoi avoir envie de publier des blocs ?

**Incitation économique** : chaque personne participant au bon fonctionnement de la chaîne (vivacité) reçoit une récompense

# Proof-of-Work

La majorité des implémentations de blockchains actuelles dispose d'un algorithme de consensus basé sur le **proof-of-work**

- Le premier qui arrive à résoudre un puzzle “cryptographique” a le droit de publier un bloc
- Bonnes propriétés : dur de tricher, facile à vérifier, ...

# Fonctions de hachage cryptographique

Une fonction de hachage  $h$  prend une **donnée** en entrée de taille arbitraire  $n$  et produit une image de taille constante  $c$  en sortie : un **hash**

$$h : \{0, 1\}^n \rightarrow \{0, 1\}^c$$

Propriétés attendues :

- Déterminisme :  $x = y \Rightarrow h(x) = h(y)$
- Uniformité :  $x \neq y \Rightarrow h(x) \neq h(y)$  (impossible en pratique)
- Non-inversibilité : pour un hash donné, aucune information sur l'entrée ne doit pouvoir être déduite (cryptographie)

# Fonction de hachage cryptographique (2)

Il existe un grand nombre d'algorithmes de hachage cryptographiques : SHA, Blake2, HMAC, MD5, etc.

## Exemple de hash avec SHA (Secure Hash Algorithm)

```
$ echo "bla" | sha256sum  
00 e 3 2 6 1 a 6 e 0 d 7 9 c 3 2 9 4 4 5 a c d 5 4 0 f b 2 b 0 ...
```

```
$ echo "bli" | sha256sum  
5 f 2 c 0 6 5 3 c 7 f 7 0 3 a b f 9 a c 2 b 0 8 3 c 2 5 6 a 7 f ...
```

# Bitcoin – Algorithme de consensus

Dans Bitcoin, un bloc est dit valide si les bits de son hash commencent par  $n$  zéros (*difficulté*)

## Block $B$

Opérations :

Alice  $\xrightarrow{1bc}$  Bob

Roger  $\xrightarrow{5bc}$  John

$h(B) = 001010111010101\dots$

Si  $n = 2$  alors **valide**

Si  $n = 5$  alors **invalide**

**Comment peut-on rendre ce bloc valide ?**

# Bitcoin – Algorithme de consensus (2)

On ajoute au bloc une valeur (**nonce**) facilement modifiable

Block  $B$

Nonce : <entier>

---

Opérations :

Alice  $\xrightarrow{1bc}$  Bob

Roger  $\xrightarrow{5bc}$  John

$B_i$  = Bloc  $B$  avec  $i$  le nonce :

- $h(B_0) = 10101011... \Rightarrow 5$
- $h(B_1) = 00011101... \Rightarrow 5$
- ...
- $h(B_{23}) = 0000010... \Rightarrow X$

# Bitcoin – Algorithme de consensus (2)

On ajoute au bloc une valeur (**nonce**) facilement modifiable

Block  $B$

Nonce : <entier>

---

Opérations :

Alice  $\xrightarrow{1bc}$  Bob

Roger  $\xrightarrow{5bc}$  John

$B_i$  = Bloc  $B$  avec  $i$  le nonce :

- $h(B_0) = 10101011... \Rightarrow 5$
- $h(B_1) = 00011101... \Rightarrow 5$
- ...
- $h(B_{23}) = 0000010... \Rightarrow X$

Plus  $n$  est grand, plus il est “difficile” de trouver un bloc valide



# Bitcoin – Algorithme de consensus (3)

Bitcoin vise à ce que le réseau dispose d'un bloc toutes les 10 minutes. Pour arriver à ce but, la difficulté est adaptée dynamiquement :

- Après la publication d'un bloc, la difficulté pour le bloc suivant démarre très haut et décroît chaque minute ;
- Si un bloc valide est trouvé en dessous de 10 minutes, la difficulté sera accrue pour les blocs futurs ;
- Au contraire, si un bloc est publié au-delà de 10 minutes, la difficulté sera réduite.

# Bitcoin – Algorithme de consensus (4)

## Critiques

- Course au calcul : premier arrivé, premier servi
- Favorise les personnes disposant d'un matériel dédié
- Consomme énormément d'énergie
- Entraîne une centralisation du réseau

## Quelques chiffres

- 1 Bitcoin  $\approx$  32800e
- La récompense pour un bloc est actuellement 6.25 Bitcoin

# Différents algorithmes de consensus

Proof-of-Work : coûteux mais simple et robuste

D'autres algorithmes de consensus existent :

- Proof-of-Stake : droit de créer un bloc selon l'investissement des utilisateurs
- Proof-of-Space : similaire au PoW mais espace disque plutôt que calcul
- Proof-of-Authority : droit de créer un bloc basé sur la réputation

# Protocoles Économiques

# Protocole Économique

Protocole (code) exécuté par tous les participants de la chaîne

## **Contient l'ensemble des règles de la chaîne**

- Algorithme de consensus
- Représentation des données (+ API)
- Validité et applications des opérations/blocs
- *Smart-contracts*
- ...

# Représentation des données

Le protocole doit définir le format de l'état de la chaîne et des structures de données (e.g. opérations, blocs, ..)

## **Exemple : un *état* de la chaîne**

/accounts:

{id => balance}

/constants:

rewards-per-block

min-time-between-blocks

max-op-per-block

...

# Représentation des données

Le protocole doit définir le format de l'état de la chaîne et des structures de données (e.g. opérations, blocs, ..)

## Exemple : un *état* de la chaîne

/accounts:

{id => balance}

/constants:

rewards-per-block

min-time-between-blocks

max-op-per-block

...

## Pourquoi ?

# Cohérence des données

La validité d'un bloc et des opérations dépendent de la cohérence des données.

Comment s'en assurer ?

$B$

Pred. : # 380f004f
Level : 345
Opérations :
Alice $\xrightarrow{10e}$ Bob
Bob $\xrightarrow{5e}$ Bob
Bob $\xrightarrow{2e}$ Charles

$S$

Alice	1293e
Bob	5432e
Charles	543e

$B(S)$

Alice	1220e
Bob	5440e
Charles	545e



# Cohérence des données

La validité d'un bloc et des opérations dépendent de la cohérence des données.

- On *hash* l'état résultant
- Ce hash est inclus dans le bloc

$B$

Pred. :	# 380f004f
Level :	345
State <sub>H</sub> :	$H(B(S))$
Opérations : ...	

$B(S)$

Alice	1220e
Bob	5440e
Charles	545e

# Validité des opérations

## Format d'une opération :

$X \xrightarrow{\leq \text{montant}} Y$  + signature

## Propriétés que le protocole doit vérifier :

- $X$  existe dans l'état
- $X$  possède au moins  $\text{<montant>}$  dans son compte
- La signature est bien celle de  $X$

# Validité des opérations

**Format d'une opération :**

$X \xrightarrow{\text{<montant>}} Y$  + signature

**Propriétés que le protocole doit vérifier :**

- $X$  existe dans l'état
- $X$  possède au moins <montant> dans son compte
- La signature est bien celle de  $X$

Que se passe-t-il si  $Y$  n'existe pas ?

# Validité des opérations

**Format d'une opération :**

$X \xrightarrow{\leq \text{montant} >} Y$  + signature

**Propriétés que le protocole doit vérifier :**

- $X$  existe dans l'état
- $X$  possède au moins  $<\text{montant}>$  dans son compte
- La signature est bien celle de  $X$

Que se passe-t-il si  $Y$  n'existe pas ?

Que se passe-t-il si  $Y$  ré-injecte à nouveau cette opération ?

# Mécanismes d'*anti-replay*

## On associe un compteur à chaque compte

- À chaque débit, le compteur du compte est incrémenté
- Les opérations doivent déclarer pour quel compteur elles supposent être valides
- Pour être valide, l'opération doit avoir le même compteur que celui du compte dans l'état
- Un entier par compte à stocker dans l'état

Alice(**25**)  $\xrightarrow{10e}$  Bob

Alice(**24**)  $\xrightarrow{10e}$  Bob

<i>S</i>		
Alice	<b>#25</b>	1293e
Bob	#32	5432e
Charles	#10	543e

# Validité des blocs

$B$

Pred. :	# 380f004f
Level :	345
State <sub>H</sub> :	# 9932c2ad
Opérations : op <sub>1</sub> ...op <sub>n</sub>	

## Propriétés à vérifier :

- Prédécesseur est cohérent
- Le niveau est cohérent
- Les opérations sont valides
- L'état résultant est cohérent

⇒ On doit stocker le bloc prédécesseur dans l'état

# Application des blocs

En général, on distribue une récompense au créateur du bloc.  
**Comment peut-on implémenter cela ?**

# Application des blocs

En général, on distribue une récompense au créateur du bloc

## **Comment peut-on implémenter cela ?**

1. On ajoute aux blocs l'identifiant du créateur
2. La fonction d'application de bloc crédite la récompense
3. L'état doit également être mis-à-jour

Rappel : Création monétaire  $\Rightarrow$  Inflation



# Protocole Bitcoin

Rappel : le protocole Bitcoin veut un bloc toutes les 10min

**Comment faire ?**

# Protocole Bitcoin

Rappel : le protocole Bitcoin veut un bloc toutes les 10min

## Comment faire ?

1. On ajoute à l'état la difficulté actuelle
2. On ajoute aux blocs un temps de création
3. On ajoute à l'état une moyenne des temps
4. La fonction d'application doit :
  - Calculer le delta de temps entre les deux blocs
  - Vérifier que ce delta est cohérent
  - Vérifier le hash vis-à-vis de la difficulté et du temps
  - Mettre à jour la moyenne et, la difficulté au besoin

# Déterminer les constantes du protocole

## Vitesse de création des blocs

- Création d'un bloc toutes les heures  $\Rightarrow$  Effet ?

# Déterminer les constantes du protocole

## Vitesse de création des blocs

- Création d'un bloc toutes les heures  $\Rightarrow$  Effet ?  
 $\Rightarrow$  Lenteur d'inclusion des transactions
- Création d'un bloc toutes les secondes  $\Rightarrow$  Effet ?

# Déterminer les constantes du protocole

## Vitesse de création des blocs

- Création d'un bloc toutes les heures  $\Rightarrow$  Effet ?  
 $\Rightarrow$  Lenteur d'inclusion des transactions
- Création d'un bloc toutes les secondes  $\Rightarrow$  Effet ?  
 $\Rightarrow$  Latence réseau, synchronisation difficile, ...

# Déterminer les constantes du protocole (2)

## Taille d'une opération

2 adresses ( $2 \times 32$  octets) + signature (32 octets)  
compteur (8 octets) + montant (8 octets)

= 112 octets

## Nombre d'opérations autorisées dans un bloc

- 10 opérations autorisées ( $\sim 1\text{Ko}$ )  $\Rightarrow$  Effet ?

# Déterminer les constantes du protocole (2)

## Taille d'une opération

2 adresses ( $2 \times 32$  octets) + signature (32 octets)  
compteur (8 octets) + montant (8 octets)

= 112 octets

## Nombre d'opérations autorisées dans un bloc

- 10 opérations autorisées ( $\sim 1\text{Ko}$ )  $\Rightarrow$  Effet ?
- 10.000 opérations autorisées ( $\sim 1,1\text{Mo}$ )  $\Rightarrow$  Effet ?

# Déterminer les constantes du protocole (2.2)

## Il faut prendre en compte :

- Le temps de validation d'une opération et d'un bloc
- La propagation et l'aggrégation dans le réseau
- Les effets sur le consensus :
  - Exemple : les blocs vides sont plus rapides à créer que des blocs remplis
- Le *throughput* (débit en français) =  $\frac{\text{Nb. opérations par bloc}}{\text{Vitesse de création d'un bloc}}$   
(e.g. Visa = 1.736 op/s)
- . . .



# Déterminer les constantes du protocole (3)

## **Conclusion – Déterminer les constantes est très difficile :**

- Incitation économique vs. limitations techniques
- Variables économiques du “vrai” monde
- Considérer les états possibles du réseau  
(surchargé, attaqué, . . . )
- Choix technologiques, architectures matérielles, . . .

# Protocole économique – Conclusion

## Implémentation des règles

- Tout le monde choisit d'exécuter le même code ;
- Le code implémente la logique de la chaîne ;
- Le code est loi.

## Criticité du protocole

- Bug de création d'argent  $\Rightarrow$  dévalorisation monétaire ;
- Une exception non-rattrapée  $\Rightarrow$  arrêt de la chaîne, . . .

# Smart-contracts

# Définition

**Un *Smart-Contract* est un compte régit par un script possédant :**

- Une adresse (identifiant) et un montant ;
  - Du **code** à exécuter ;
  - Un **espace de stockage**.
- 
- Le code et le stockage sont également visibles de tous.
  - Le protocole définit :
    - Le langage du smart-contract : syntaxe & sémantique ;
    - Son modèle d'exécution.

# Modèle d'exécution

## Création

- Tout le monde peut créer un smart-contract<sup>1</sup> ;
- Le créateur (manager) doit fournir le code du *smart-contract* à créer et fournir un stockage initial.

---

1. En fonction des contraintes établies par le protocole

# Modèle d'exécution (ii)

## Exécution

- Un *smart-contract* est exécuté via une transaction ;
- Seul le stockage varie entre chaque appel ;
- Selon son code, un *smart-contract* peut :
  - Émettre des transactions ;
  - Appeler un autre *smart-contract* ;
  - Créer de nouveaux *smart-contracts*.
- Son exécution est généralement atomique.

# Exemple

## Code

```
1  /*  add  the  parameter
2      to  the  storage
3      for  each  call  */
4  int  storage ;
5  code ( int  parameter ) {
6      storage  +=  parameter  ;
7      return  ;
8  }
```

## Opération de création :

- Un *manager*
- Un montant initial
- Un stockage initial
- Le code à exécuter

## Exemple :

op\_crea = *hmanager* : Alice, balance : 0, storage : 0, code : *Pi*

⇒ Propagation de l'opération sur la chaîne

# Exemple – Création

$B$

Pred. :	# 380f424f
Level :	4201
State <sub>H</sub> :	# 2952a3de
Opérations : op_crea	

$B(S)$

Alice	1220e
Bob	5440e
Charles	545e
Alice <sub>sc</sub>	0e      Storage : 0

- Tout le monde va stocker le nouveau smart-contract
- Il sera présent dans la chaîne pour toujours



# Exemple – Appels

$op_1 = \text{Alice} \xrightarrow{h0e, param:10i} \text{Alice}_{sc}$

$op_2 = \text{Bob} \xrightarrow{h2e, param:17i} \text{Alice}_{sc}$

Après inclusion des opérations dans un bloc  $B$  :

$B(S)$

Alice	1220e
Bob	5440e
Charles	545e
Alice <sub>sc</sub>	2e      Storage : 27

Tous les participants doivent **exécuter les appels au smart-contract** pour valider et mettre à jour leur état.

# Coût des Smart-contracts

**Que se passe-t-il si :**

- On appelle un smart-contract très long à exécuter ou qui ne termine pas ?

# Coût des Smart-contracts

## Que se passe-t-il si :

- On appelle un smart-contract très long à exécuter ou qui ne termine pas ?
- Le code du smart-contract est très gros ou stocke (ou va stocker) énormément de données ?

# Coût des Smart-contracts

**Que se passe-t-il si :**

- On appelle un smart-contract très long à exécuter ou qui ne termine pas ?
- Le code du smart-contract est très gros ou stocke (ou va stocker) énormément de données ?

**On doit faire payer**

# Notion de *Gas*

## Gas

- Les personnes appelant des S-C doivent fournir du *gas* **en payant**
- Chaque exécution de smart-contract a une limite de *gas*
- Chaque instruction exécutée engendre un coût en *gas*
- Si l'exécution dépasse le *gas* fourni par l'utilisateur, l'appel au S-C échoue

En général, le montant payé pour le *gas* est reversé au validateur du bloc.

# Stockage des Smart-contracts

## On paye l'espace de stockage utilisé :

- À la création, on paye en proportion de :
  - La taille du code injectée
  - Le stockage initial à créer
- À l'appel, on paye le stockage qui va être généré

En général, le montant payé pour le stockage est « brûlé ».

# Exemples de « Dapps » ( Decentralized applications)

## Vote électronique (version naïve)

- Le contrat contient les adresses des participants autorisés à voter ;
- Lorsqu'un participant vote, le contrat marque dans son stockage que le participant à voter ;
- Après une certaine date, le vote est terminé et le contrat n'accepte plus d'appel. L'issue du vote et son historique restent consultables.

# Exemples de « Dapps » ( Decentralized applications)

## Multi-signature M-N

- Un compte commun géré par  $N$  comptes ;
- Un transfert ne peut s'effectuer que si  $M$  parmi  $N$  signatures sont récoltées.



# Exemples de « Dapps » ( Decentralized applications)

## ***Escrow (séquestre)***

- Permet de sécuriser des échanges
- L'acheteur et le vendeur passent par le contrat :
  - Chacun donne  $2 \times$  le montant de la vente au contrat
  - Si les deux partis confirment le bon déroulement alors le transfert est effectué et le surplus est reversé sinon l'argent est maintenu dans le contrat tant qu'aucun accord n'est trouvé.

# Exemples de « Dapps » ( Decentralized applications)


**Crypto-kitties...** <https://www.cryptokitties.co/>

# Langages de Smart-contract

- Bitcoin Script – Bitcoin
- Solidity, Vyper, EVM – Ethereum
- SmartPy, Ligo, Michelson – Tezos
- . . .

# Criticité des Smart-contracts

## anyone can kill your contract #6995

 **Closed** ghost opened this issue on 6 Nov 2017 · 17 comments



ghost commented on 6 Nov 2017 • edited by ghost ▾

I accidentally killed it.

<https://etherscan.io/address/0x863df6bfa4469f3ead0be8f9f2aae51c91a907b4>



69



3



115



59



24



46



1



2

Coût : 930 millions de dollars en tokens Ethereum

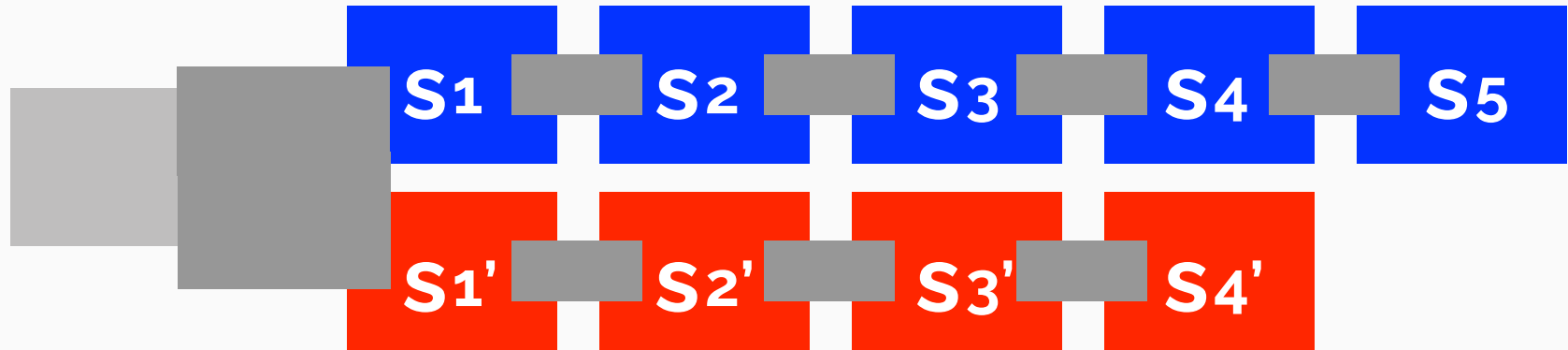
# Conclusion

- Les smart-contracts permettent d'automatiser des actions sur la chaîne
- Chacun peut observer le code et les interactions des smart-contracts
- Les smart-contracts ont un coût d'utilisation et de potentielles failles
  - ⇒ Il est important de s'assurer de leur correction

# Algorithme de consensus

# Rappel : nécessité du consensus

Il est possible qu'à un moment le réseau ne soit plus d'accord sur l'état de la chaîne : quel bloc est en tête ?



## **Problème fondamental de calcul distribué**

Tous les processus doivent se mettre d'accord (atteindre un consensus) afin d'être sûres qu'elles ont obtenu la même valeur.

### **Objectif**

- Parvenir à la fiabilité d'un système.
- Doit pouvoir tolérer la présence de “pannes”.



# Tolérance aux pannes

## Deux types de pannes

- Crash : un processus s'arrête brusquement.
- Panne Byzantine : un processus agit de manière imprévue  
mauvais messages envoyés, délais de réponse, etc.

## Tolérances aux pannes

Le système doit continuer à fonctionner même en présence de pannes.

# Problème des généraux Byzantins

Des généraux de l'armée Byzantine campent autour d'une cité ennemie. Ils ne peuvent communiquer qu'à l'aide de messagers et doivent établir un plan de bataille commun, faute de quoi la défaite sera inévitable. Cependant un certain nombre de ces généraux peuvent s'avérer être des traîtres, qui essayeront donc de semer la confusion parmi les autres. Le problème est donc de trouver un algorithme pour s'assurer que les généraux loyaux arrivent tout de même à se mettre d'accord sur un plan de bataille.

# Propriété attendue : sûreté

## Sûreté (Safety)

- Finalité : pas de retour en arrière possible.
  - Cohérence : les processus disposent de la même valeur.
  - Entente : les processus sont d'accord sur la valeur.
- ⇒ Il n'y a pas de longues “branches” (*fork*) dans une chaîne.

# Propriété de finalité

## Définition

- Un bloc (ou une transaction) est dite finale si elle **reste** sur la chaîne de chaque processus.
- Confirmations : nombre de blocs qui doivent arriver avant d'atteindre la finalité.

# Finalité probabiliste

## Finalité probabiliste

- Après  $n$  confirmations, un bloc est *probablement* final.
- La probabilité de *fork* décroît (exponentiellement) en fonction de  $n$ .

Exemple : Proof of Work (PoW)

# Finalité à la Nakamoto

## Nakamoto-style

- Tolère au maximum,  $1/2$  d'acteurs *Byzantin*.
- La production de bloc dépend proportionnellement d'une ressource : e.g. puissance de calcul, *stake*
- Stratégie des attaquants : construire une plus longue chaîne en secret en utilisant ses propres ressources
- Les attaquants disposent d'une minorité des ressources : au bout d'un moment, ils vont perdre.

⇒ Attaquable si au moins 51% des ressources du réseau sont contrôlés par des attaquants.

# Finalité déterministe

## Finalité déterministe (ou absolue)

- Après  $n$  confirmations, un bloc est **final**.
- $n$  est une constante connue  
(finalité immédiate quand  $n = 0$ )

Exemple : Tendermint <sup>1</sup>

---

1. Algorithme de consensus de la blockchain *Cosmos*

# Algorithmes BFT classiques

## Classic BFT-style

- Tolère au maximum  $1/3$  d'acteurs *Byzantin*
- $n$  acteurs,  $f$  acteurs Byzantins,  $n = 3f + 1$
- Basé sur une notion de *quorum*
- Un quorum est atteint lorsque  $2f + 1$  d'acteurs sont d'accords.



# Propriété attendue : Progrès

## Progrès (*Liveness*)

- Le système doit être capable de progresser en un temps borné.
- Peut-être impossible sous certaines conditions :
  - Trop d'acteurs Byzantins.
  - Asynchronie des messages.

# Propriété attendue : Équité (Blockchains)

## Équité (*Fairness*)

- Les acteurs honnêtes sont récompensés de manière équitable.
- Les acteurs malhonnêtes doivent être punis proportionnellement à leur nuisance (Proof of Stake).

## Synchrone

- Tous les messages arrivent **et** en un temps borné.

## Asynchrone

- Les messages peuvent dupliqués/retardés/perdus.

## Partiellement synchrone

- Synchrone mais la borne de temps n'est pas connue.

# Théorème FLP

## Théorème FLP (1985 - Fischer, Lynch and Patterson)

- Le consensus est impossible pour des processus asynchrones s'il peut se produire au moins une défaillance.

# Protocoles Proof of Work à la Bitcoin

## Prover

- “Mine” un hash de bloc qui sera préfixé par  $n$  zéros.
- Peut prouver aux autres l’effort de calcul.
- Récompensé pour son travail.

## Vérifier

- Vérifie l’effort de calcul instantanément

# Protocoles Proof of Stake

## Sélection du producteur de bloc

- Sélection basée sur la quantité de jetons (*stake*).
- Processus de décision déterministe via le protocole économique
- Aucune puissance de calcul requise : la validité dépend de l'identité du producteur.

Problème :

Que faire si le producteur publie deux blocs différents ?

# Problème du *Nothing at stake*

Créer deux blocs au même niveau :

- En Proof of Work, créer deux blocs au même niveau (fork) coûte cher en puissance de calcul.
- En Proof of Stake, si il n'y a pas de coût à créer un bloc, il n'y a rien à perdre à créer de fork (Nothing at stake).

Rappel : Fork  $\Rightarrow$  Réduit la possibilité de finalité.

## **Solution**

- Punir (économiquement) ce comportement.

# **Algorithme de consensus : Emmy (Tezos)**



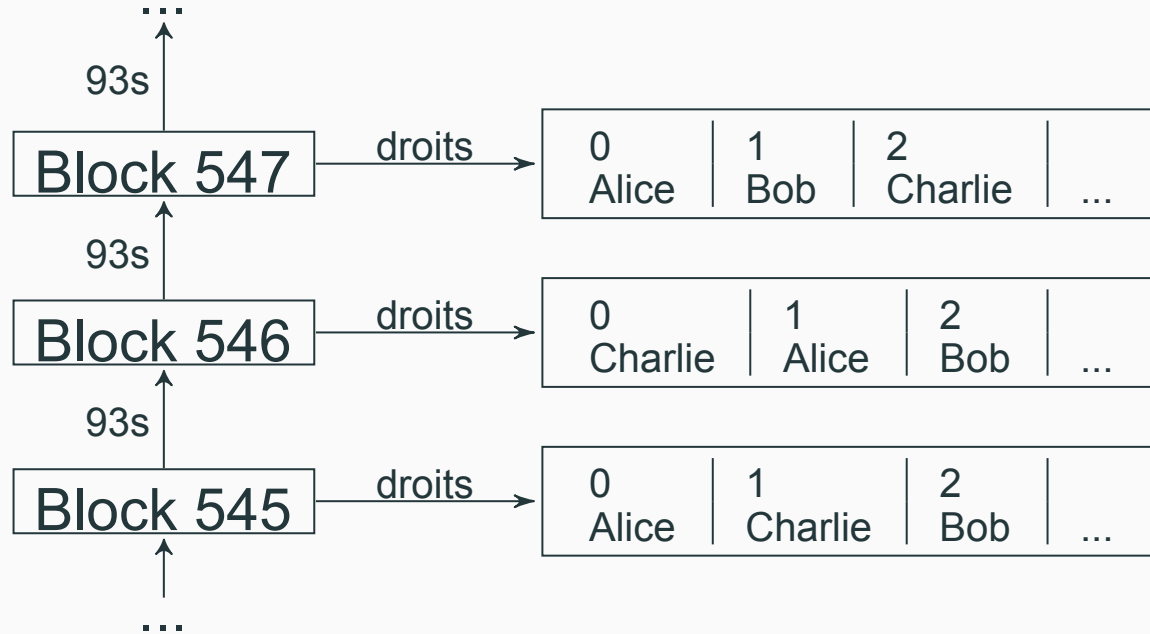
# Emmy : caractéristiques

- “Liquid” Proof of Stake
- Finalité probabiliste
- Droits déterminés proportionnellement en fonction du *stake*
- Mining  $\Rightarrow$  **Baking**

# Droits de “baking”

- Pour chaque level, on tire, au hasard, une liste de comptes.
- Le premier compte de la liste peut commencer à “baker” au temps  $t$  (93s après le prédécesseur), le deuxième compte à  $(t + 93s)$ , le troisième à  $(t + 60s)$ , etc.
- Liste infinie pour assurer la **liveness** : si un “baker” est inactif, on attend le prochain.

# Droits de “baking” – exemple



# Calcul des droits de baking

## Le montant de jetons varie dans le temps

- “Snapshots” régulier de la distribution d’argents des participants

## Les droits de baking ne doivent pas être manipulable

- Tous les 32 blocs, le baker désigné doit fournir le hash d’une graine d’aléa (secret).
- Après un **cycle** (4096 blocs), tous les bakers ayant injecté des hashes de graines doivent révéler leurs graines.
- Toutes les graines sont rassemblées pour former la prochaine graine d’aléa qui permettra de tirer les nouveaux droits (à partir d’un snapshot aléatoire).

# Dépôt de garantie

Pour résoudre le problème de “Nothing at stake” :

- Dès qu'un bloc est produit, un dépôt de garantie est gelé ( $640_{tz}$  par bloc).
- Si un “baker” produit deux blocs au même niveau, on enlève tous les dépôts encore gelés.
- Les dépôts sont rendus au fur et à mesure, après un certain temps, si le “baker” n'a pas triché.
- Relation entre le stake et la somme des dépôts de garantie.

Incitation économique à rester honnête

# Delegated Proof of Stake

- Pour avoir des droits de production de bloc, les participants doivent avoir  $8,000_{tz}$  : un **rouleau**
- On détermine les droits en fonction du nombre de rouleaux.

Si un participant n'a pas assez de jetons pour participer (et donc obtenir des récompenses), il peut **déléguer** son *stake* ou encore l'associer à d'autres participants.

À l'heure actuelle, il existe des “services de délégations” qui acceptent les délégations et gardent une petite partie des récompenses (entre 5% et 15%) en échange du service.

# *Liquid Proof of Stake*

Delegated PoS mais :

- Les déléguants peuvent utiliser leurs jetons (pas de gel).
- Les “bakers” ne peuvent pas utiliser les jetons délégués.
- Les déléguants peuvent changer de délégués à tout moment.

# Activité

- Un baker devient inactif lorsqu'il n'a pas produit de bloc depuis un certain temps.
- Seuls les bakers actifs sont sélectionnés pour participer.



# Inflation

Chaque participant peut gagner jusqu'à 6% de ses jetons chaque année :

- Dès qu'un bloc est produit, la récompense est distribuée.
- L'inflation est maximisé à 6% (constante de protocole)

# Finalité probabiliste

Un bloc est final lorsqu'il a +99% de chances de faire partie de la chaîne finale.

Pour augmenter rapidement cette probabilité :

- On demande aux participants de désigner les blocs qu'ils souhaitent finaliser via des votes : **endorsements**.
- Plus un bloc est “endorsé”, plus sa probabilité de finalité est grande :  
fitness de bloc = fitness du prédécesseur  
+ 1  
+ nombre d'endorsements

# Finalité probabiliste (2)

Un bloc est final lorsqu'il a +99% de chances de faire partie de la chaîne finale.

Plus une séquence de blocs possède d'endorsements, plus sa probabilité d'être finale est grande.

## **Hypothèse Nakamoto-style**

- +50% des participants sont honnêtes.

# Endorsements

- Les droits d'endorsements sont déterminés de la même manière que les droits de baking.
- Un endorsement pour un bloc  $n$  doit être inclus dans le bloc  $n + 1$ .
- On récompense également les endorsements.
- On prend également un dépôt de garantie.
- Deux endorsements à un même niveau pour deux blocs différents  $\Rightarrow$  punition
- Seuls les délégués actifs sont sélectionnés.