Consensus

Abdelkader Ouared

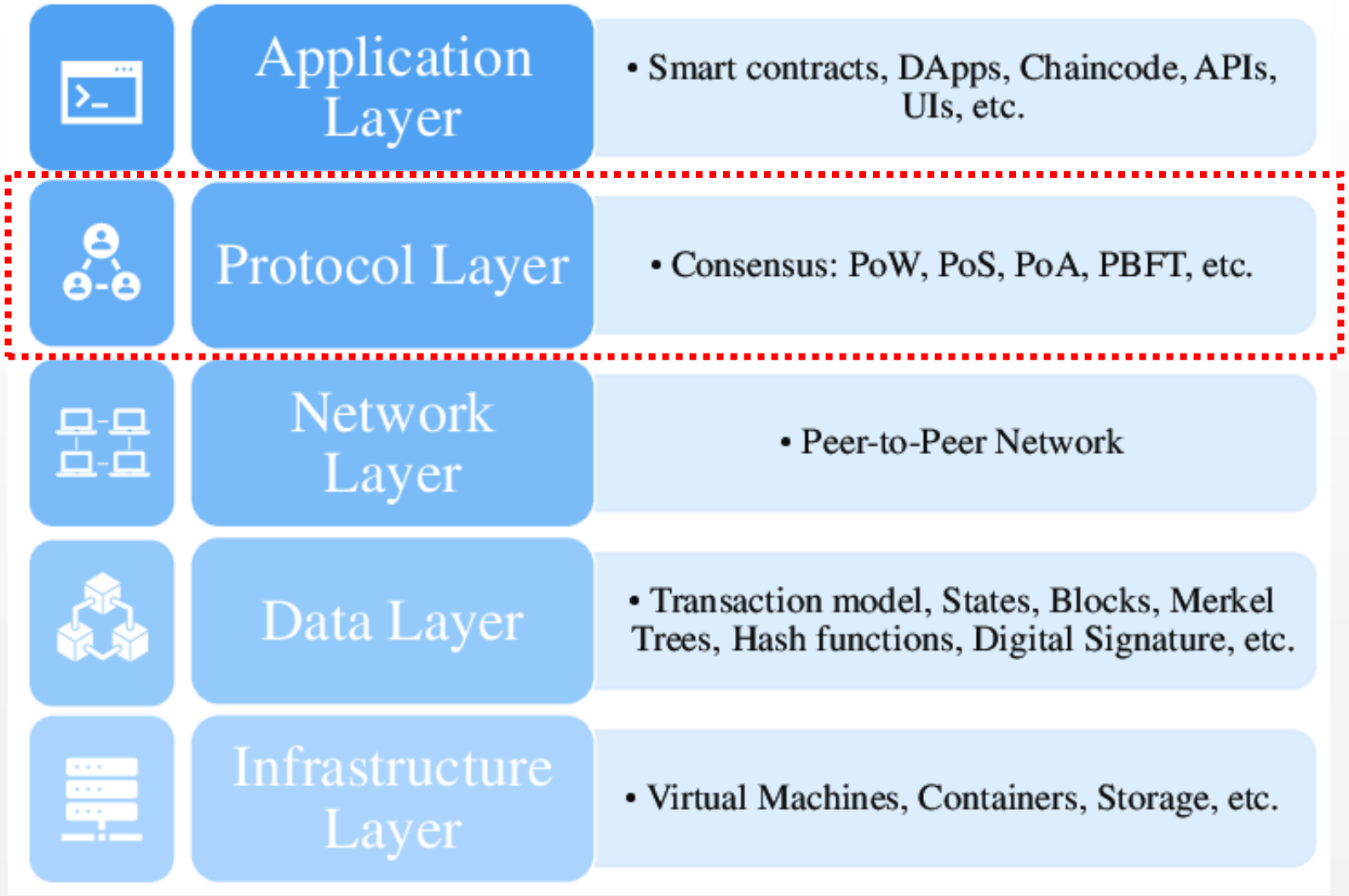
abdelkader.ouared@univ-tiaret.dz

 @ouared

Agenda

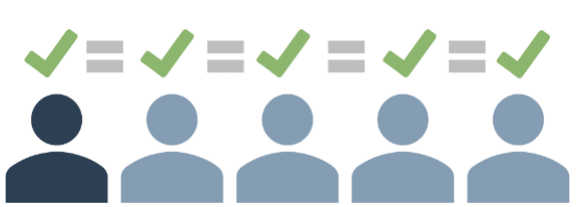
- Qu’est-ce que le consensus
- Pourquoi est-il ***nécessaire*** pour les systèmes blockchain ?
- Le problème des généraux byzantins: ***Byzantine Generals Problem (BGP)***
- Tolérance aux pannes: ***Fault Tolerant Broadcast (FTB)***
- Machinee d’état Répliquée: ***State Machine Replication (STR)***
- Algorithmes de consensus

Les couches de blockchain



Qu'est-ce que le consensus ?

- Le consensus est un accord entre un groupe de personnes sur une idée, une déclaration ou un plan d'action.



  - **Majorité** : 51%
  - **Supermajorité**: 66% (sometimes higher)
  - **Collectif** : 100%
  - **Pondérée**: tous les votes n'ont pas le même poids ou plusieurs votes par agent

Qu'est-ce que le consensus ?

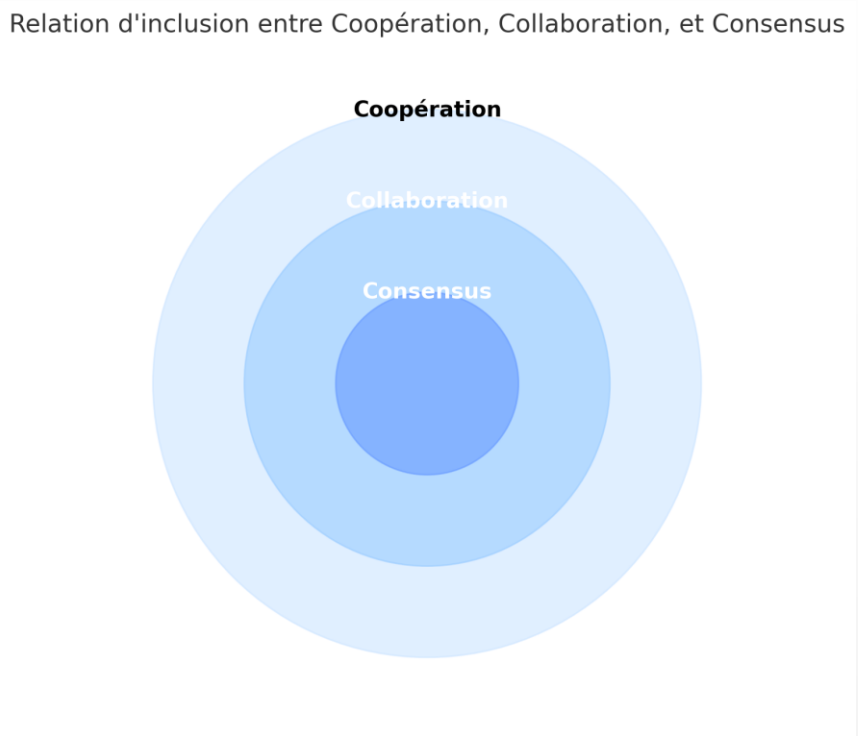
- Le consensus n’est généralement pertinent que lorsqu’il n’y a pas de **leader** distingué
  - Un jury doit se mettre d'accord sur une décision commune du **tribunal**.
  - Le **Sénat** doit parvenir à un consensus sur l’adoption de nouveaux projets de loi (majorité ou supermajorité)
- Particulièrement important lorsqu'il existe un **désaccord important** ou **un risque de parties** peu fiables dans les discussions autour de la décision

Consensus vs Collaboration vs Coopération

- **Le consensus** cherche un **accord collectif/général/total**, ou d'une décision partagée par l'ensemble d'un groupe Cela ne signifie pas que chacun est parfaitement satisfait, mais que tous sont d'accord pour avancer avec la décision commune.

- **La collaboration** est un effort **commun** ou un **travail conjoint et intégré interactif** vers un objectif commun, même si tout le monde n'est pas toujours d'accord sur chaque point.

- La coopération est un effort **coordonné (**les rôles sont souvent mieux définis et les contributions sont moins imbriquées **)**, où chaque partie (individus ou les groupes) travaillent ensemble pour un objectif commun/partagé, mais ils peuvent accomplir leurs tâches de manière plus **indépendante/** autonome.



Consensus vs Collaboration vs Consensus

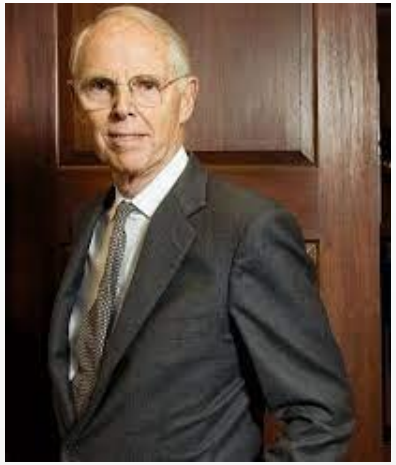
Types d'interaction ont été définis et analysés à travers divers composantes (Ferber, 1995)

Buts	Ressources	Compétences	Situation
Compatibles	Suffisantes	Suffisantes	Indépendance
Compatibles	Suffisantes	Insuffisantes	Collaboration simple
Compatibles	Insuffisantes	Suffisantes	Encombrement
Compatibles	Insuffisantes	Insuffisantes	Collaboration coordonnée
Incompatibles	Suffisantes	Suffisantes	Compétition individuelle pure
Incompatibles	Suffisantes	Insuffisantes	Compétition collective pure
Incompatibles	Insuffisantes	Suffisantes	Conflits individuels pour des ressources
Incompatibles	Insuffisantes	Insuffisantes	Conflits collectifs pour des ressources

Getting to Yes: Negotiating Agreement

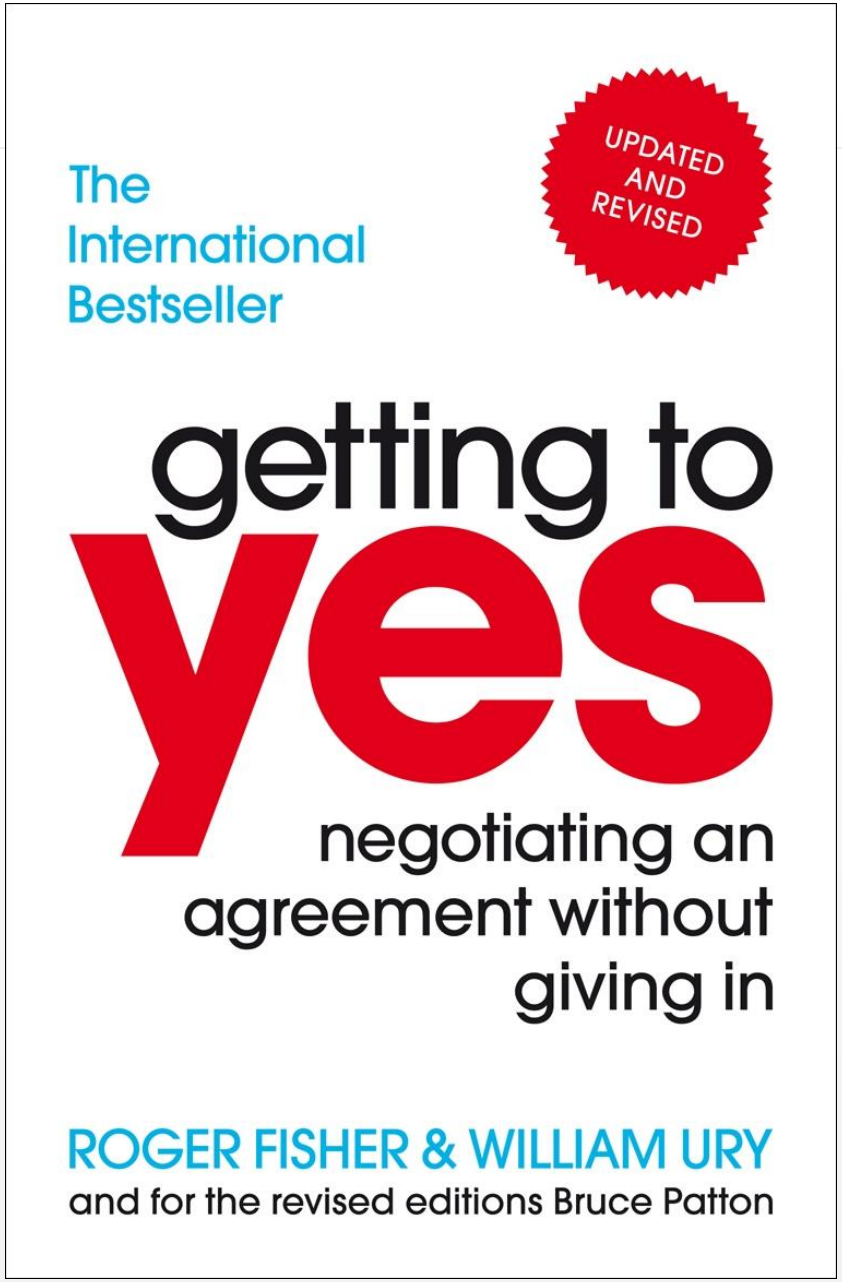
Without Giving In *by Roger Fisher*

Cité 16480 fois



Quatre principes de *Getting to Yes* *adaptés* pour les entretiens d'embauche :

- ❑ **Rester professionnel** : Séparez les émotions des questions posées. Restez calme et positif, même face à des questions difficiles.
- ❑ **Comprendre les besoins de l'employeur** : Concentrez-vous sur ce que l'entreprise recherche réellement et montrez comment vous pouvez y répondre.
- ❑ **Proposer des idées gagnant-gagnant** : Suggérez des solutions ou des idées qui profitent à l'entreprise et mettent en valeur vos compétences.
- ❑ **Appuyer vos arguments avec des faits** : Utilisez des résultats concrets ou des exemples de réussite pour démontrer vos compétences.



Qu’est-ce que le ***consensus*** dans **la blockchain** ?

Le processus par lequel nous parvenons à un accord sur l'état du système **entre des machines non fiables** connectées par des réseaux asynchrones

Le consensus dans la blockchain ?

- L'accord des composants du système (nœuds) sur l'état [suivant] du système, ou la **transition entre l'état actuel et l'état suivant**
- Les nœuds doivent s'accorder sur un ensemble de **transactions valides** représentant le **changement de l'état actuel du système** à l'état suivant du système
- Le consensus doit être atteint **automatiquement** (sans surveillance humaine)

Le consensus dans la blockchain ?

- Le consensus **est irréversible** : les transactions publiées sont définitives
- Le consensus de la blockchain est un sous-ensemble du consensus des systèmes distribués
  - **Système distribué :** un certain nombre d'ordinateurs indépendants reliés par un réseau
- Doit être résistant aux acteurs malveillants ou faux

« Le consensus est le processus par lequel tous les nœuds s'accordent sur le même registre »

Le consensus dans la blockchain ?

A Hundred Impossibility Profs for Distributed Computing (1989)

Cent Preuves d'Impossibilité en Calcul Distribué

- ***Objectif** : Exposer les limites fondamentales des systèmes distribués.*
- *100 preuves montrant les tâches impossibles (ex : consensus, tolérance aux pannes).*
- *Guide la conception d'algorithmes réalistes: concevoir des systèmes plus robustes*

Distributed Consensus Making Impossible Possible
Heidi Howard - JOTB16

The Part-Time Parliament · 7

defined to be the largest vote v in $Votes(\mathcal{B})$ cast by p with $v_{bal} < b$, or to be $null_p$ if there was no such vote. Since $null_p$ is smaller than any real vote cast by p, this means that $MaxVote(b, p, \mathcal{B})$ is the largest vote in the set

$$\{v \in Votes(\mathcal{B}) : (v_{pst} = p) \wedge (v_{bal} < b)\} \cup \{null_p\}$$

For any nonempty set Q of priests, $MaxVote(b, Q, \mathcal{B})$ was defined to equal the maximum of all votes $MaxVote(b, p, \mathcal{B})$ with p in Q.

Conditions $B1(\mathcal{B})$–$B3(\mathcal{B})$ are stated formally as follows.*

The Part-Time Parliament · 29

$I3(p)$ [Associated variables: $prevBal[p]$, $prevDec[p]$, $nextBal[p]$]
$\wedge\ prevBal[p] = MaxVote(\infty, p, \mathcal{B})_{bal}$
$\wedge\ prevDec[p] = MaxVote(\infty, p, \mathcal{B})_{dec}$
$\wedge\ nextBal[p] \geq prevBal[p]$

$I4(p)$ [Associated variable: $prevVotes[p]$]
$(status[p] \neq idle) \Rightarrow$
$\forall v \in prevVotes[p] : \wedge\ v = MaxVote(lastTried[p], v_{pst}, \mathcal{B})$
$\wedge\ nextBal[v_{pst}] \geq lastTried[p]$

$I5(p)$ [Associated variables: $quorum[p]$, $voters[p]$, $decree[p]$]
$(status[p] = polling) \Rightarrow$
$\wedge\ quorum[p] \subseteq \{v_{pst} : v \in prevVotes[p]\}$
$\wedge\ \exists B \in \mathcal{B} : \wedge\ quorum[p] = B_{prs}$
$\wedge\ decree[p] = B_{dec}$
$\wedge\ voters[p] \subseteq B_{vot}$
$\wedge\ lastTried[p] = B_{bal}$

$I6$ [Associated variable: $\mathcal{B}$]
$\wedge\ B1(\mathcal{B}) \wedge B2(\mathcal{B}) \wedge B3(\mathcal{B})$
$\wedge\ \forall B \in \mathcal{B} : B_{prs}$ is a majority set

$I7$ [Associated variable: $\mathcal{M}$]
$\wedge\ \forall NextBallot(b) \in \mathcal{M} : (b \leq lastTried[owner(b)])$
$\wedge\ \forall LastVote(b, v) \in \mathcal{M} : \wedge\ v = MaxVote(b, v_{pst}, \mathcal{B})$
$\wedge\ nextBal[v_{pst}] \geq b$
$\wedge\ \forall BeginBallot(b, d) \in \mathcal{M} : \exists B \in \mathcal{B} : (B_{bal} = b) \wedge (B_{dec} = d)$
$\wedge\ \forall Voted(b, p) \in \mathcal{M} : \exists B \in \mathcal{B} : (B_{bal} = b) \wedge (p \in B_{vot})$
$\wedge\ \forall Success(d) \in \mathcal{M} : \exists p : outcome[p] = d \neq$ BLANK

The Paxons had to prove that I satisfies the three conditions given above. The first condition, that I holds initially, requires checking that each conjunct is true for the initial values of all the variables. While not stated explicitly, these initial values can be inferred from the variables' descriptions, and checking the first condition is straightforward. The second condition, that I implies consistency, follows from $I1$, the first conjunct of $I6$, and Theorem 1. The hard part was proving the third condition, the invariance of I, which meant proving that I is left true by every action. This condition is proved by showing that, for each conjunct of I, executing any action when I is true leaves that conjunct true. The proofs are sketched below.

$I1(p)$ $\mathcal{B}$ is changed only by adding a new ballot or adding a new priest to B_{vot} for some $B \in \mathcal{B}$, neither of which can falsify $I1(p)$. The value of $outcome[p]$ is changed only by the **Succeed** and **Receive** *Success* **Message** actions. The enabling condition and $I5(p)$ imply that $I1(p)$ is left true by the **Succeed** action. The enabling condition, $I1(p)$, and the last conjunct of $I7$ imply that $I1(p)$ is left true by the **Receive** *Success* **Message** action.

A Hundred Impossibility Proofs for Distributed Computing

Nancy A. Lynch *
Lab for Computer Science
MIT, Cambridge, MA 02139
lynch@tds.lcs.mit.edu

1 Introduction

This talk is about impossibility results in the area of distributed computing. In this category, I include not just results that say that a particular task cannot be accomplished, but also lower bound results, which say that a task cannot be accomplished within a certain bound on cost.

I started out with a simple plan for preparing this talk: I would spend a couple of weeks reading all the impossibility proofs in our field, and would categorize them according to the ideas used. Then I would make wise and general observations, and try to predict where the future of this area is headed. That turned out to be a bit too ambitious: there are many more such results than I thought. Although it is often hard to say what constitutes a "different result", I managed to count over 100 such impossibility proofs! And my search wasn't even very systematic or exhaustive.

It's not quite as hopeless to understand this area as it might seem from the number of papers. Although there are 100 different results, there aren't 100 different ideas. I thought I could contribute something by identifying some of the commonality among the different results.

So what I will do in this talk will be an incomplete version of what I originally intended. I will give you a tour of the impossibility results that I was able to collect. I apologize for not being comprehensive, and in particular for placing perhaps undue emphasis on results I have been involved in (but those are the ones I know best!). I will describe the techniques used, as well as giving some historical perspective. I'll intersperse this with my opinions and observations, and I'll try to collect what I consider to be the most important of these at the end. Then I'll make some suggestions for future work.

*This work was supported in part by the National Science Foundation (NSF) under Grant CCR-86-11442, by the Office of Naval Research (ONR) under Contract N00014-85-K-0168 and by the Defense Advanced Research Projects Agency (DARPA) under Contract N00014-83-K-0125.

2 The Results

I classified the impossibility results I found into the following categories: shared memory resource allocation, distributed consensus, shared registers, computing in rings and other networks, communication protocols, and miscellaneous.

2.1 Shared Memory Resource Allocation

This was the area that introduced me not only to the possibility of doing impossibility proofs for distributed computing, but to the entire distributed computing research area.

In 1976, when I was at the University of Southern California, Armin Cremers and Tom Hibbard were playing with the problem of *mutual exclusion* (or allocation of one resource) in a shared-memory environment. In the environment they were considering, a group of asynchronous processes communicate via shared memory, using operations such as read and write or test-and-set.

The previous work in this area had consisted of a series of papers by Dijkstra [36] and others, each presenting a new algorithm guaranteeing mutual exclusion, along with some other properties such as progress and fairness. The properties were specified somewhat loosely; there was no formal model used for

https://www.youtube.com/watch?v=XUQJvMALfUA&t=1287s

Pourquoi un mécanisme de consensus?

- **Verrouillage distribué**
- **Banque**
- **Systèmes critiques pour la sécurité**
- **Ordonnancement et coordination distribués**
- **Architecture des certificats numériques**
  - **Architecture avec une seule autorité de certification (CA) ☹**
  - **Architecture à entité unique ☹**
  - **Architecture à autorité multiple (*Cross-domain*)**
  - **Architecture distribuée**

Tout ce qui nécessite un ***accord garanti***

Consensus sur la blockchain : les nœuds

Les types courants sont :

- **Noeud minier (Mining Node):** Ces nœuds sont ceux qui ont le plus investi dans le système et, par conséquent, ont le plus de pouvoir. Ces nœuds calculent et proposent de nouveaux blocs à la chaîne selon les règles du système. Ils reçoivent des récompenses de blocs, qui incluent souvent des frais de minage.
- **Nœud complet (Full Node)** : ces nœuds constituent une étape intermédiaire entre les nœuds miniers et les nœuds légers. **Ils stockent la copie complète de la blockchain** et **vérifient tous les blocs pour garantir la validité des transactions**. Ils ne sont souvent pas rémunérés pour leur contribution au système blockchain.
- **Nœud léger (Light Node:)** : ces nœuds ne stockent généralement **qu'une copie partielle de la blockchain** ou s'occupent simplement **des soldes**. Ils peuvent souvent vérifier les transactions, mais ne participent généralement **pas au consensus**. Ils ne sont presque jamais rémunérés pour leur contribution au système blockchain et agissent principalement comme un point d'accès pour la diffusion des transactions

Pourquoi le consensus est-il nécessaire ?

- Le consensus est un problème très difficile lorsque les parties ne sont pas dignes de confiance.
- Le réseau doit maintenir son intégrité pour conserver sa valeur
- Les transactions passées doivent être fiables pour que le réseau fonctionne
- Ainsi, la capacité de vérifier les transactions sans confiance est nécessaire
- Ce problème est résolu avec diverses formes de consensus
- Le problème du consensus peut souvent être reformulé comme la capacité à faire confiance au résultat d'une transaction ou d'un bloc, sans faire confiance aux parties impliquées dans la transaction, ou à la partie qui l'a vérifiée.

Algorithmes de consensus

“Le but d'un algorithme de consensus, en général, est de permettre la mise à jour sécurisée d'un état selon certaines règles de transition d'état spécifiques, où le droit d'effectuer les transitions d'état est réparti entre un ensemble économique.”

- Vitalik Buterin (Co-Founder of Ethereum)

https://bcfocus.com/wp-content/uploads/2018/05/2018051515042388.jpg

Qu’est-ce que le consensus dans la blockchain ?

Paxos

Lamport’s original consensus algorithm

The Part-Time Parliament
Leslie Lamport
ACM Transactions on Computer Systems
May 1998

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament’s protocol provides a new way of implementing the state-machine approach to the design of distributed systems.

Categories and Subject Descriptors: C2.4 [**Computer-Communications Networks**]: Distributed Systems—*Network operating systems*; D4.5 [**Operating Systems**]: Reliability—*Fault-tolerance*; J.1 [**Administrative Data Processing**]: Government

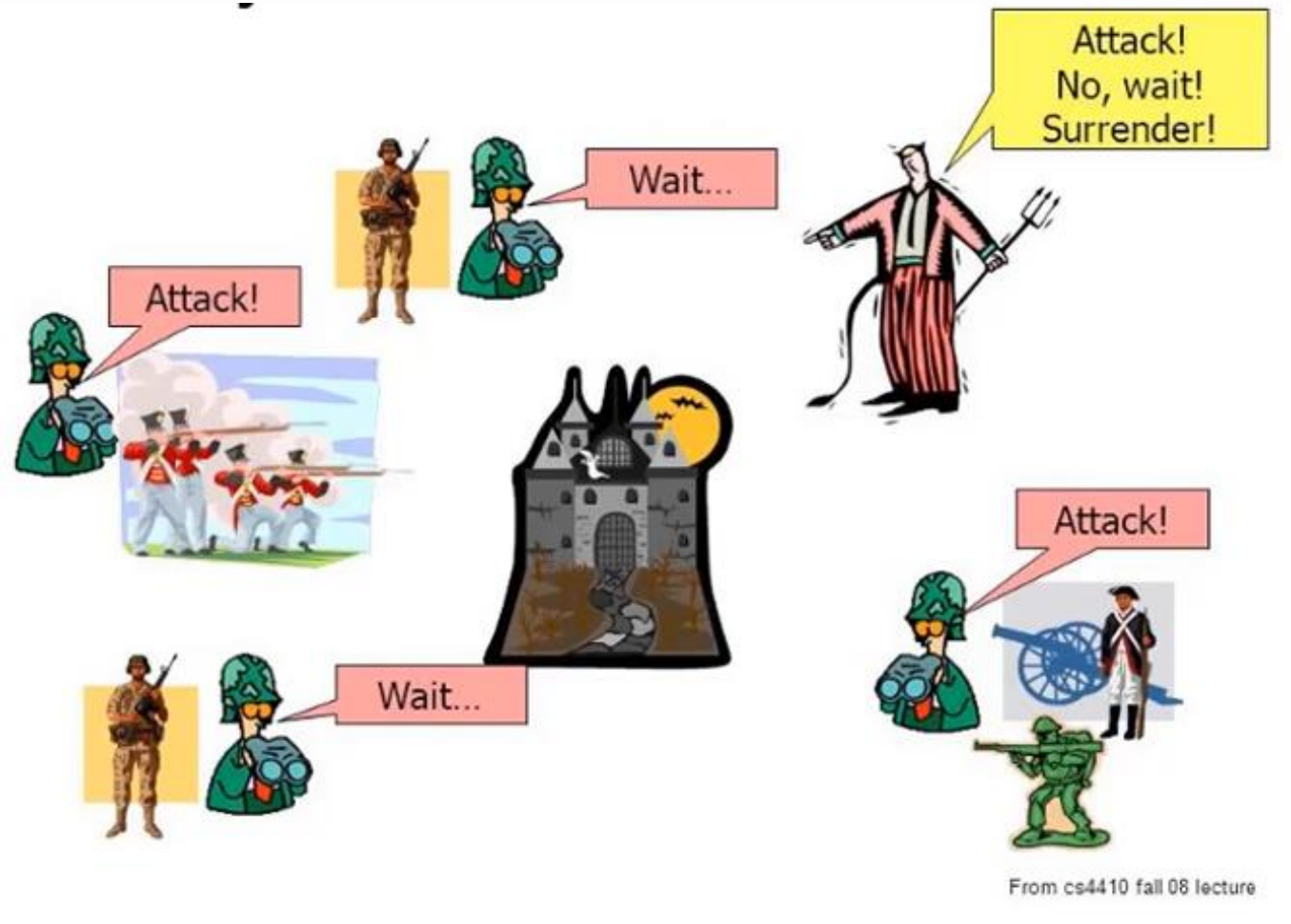
General Terms: Design, Reliability

Additional Key Words and Phrases: State machines, three-phase commit, voting

Problème des généraux byzantins: Byzantine Generals Problem (BGP)

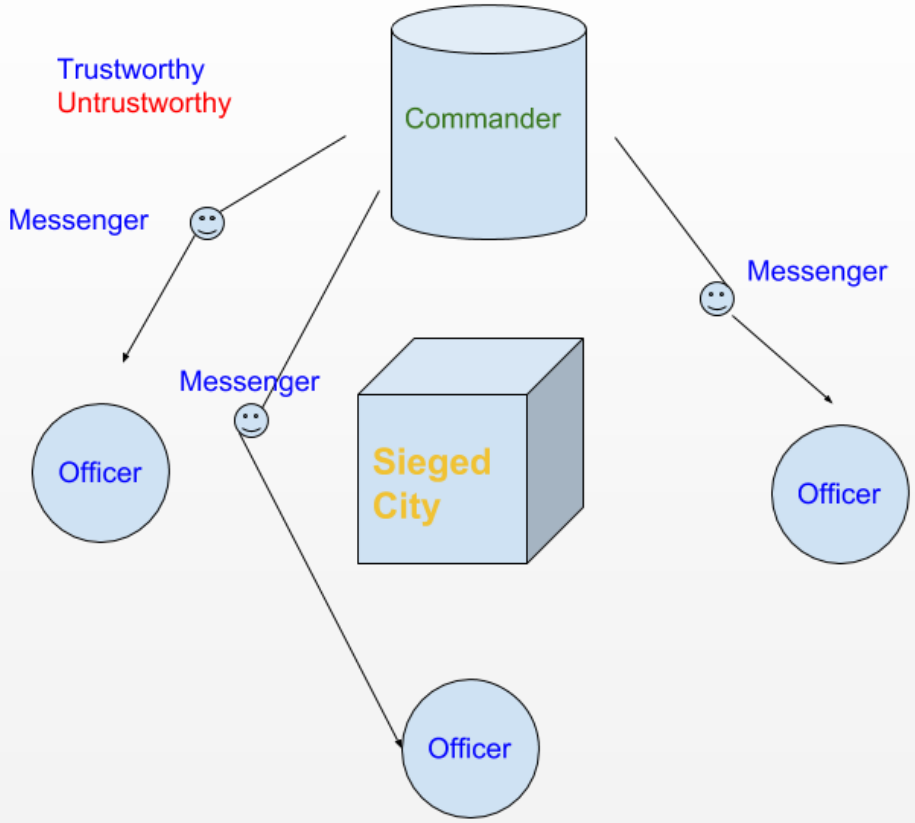
Byzantine General's Problem

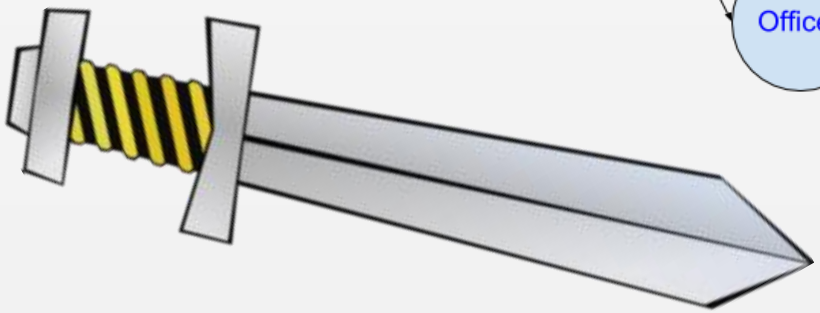
[Lamport, Shostak, and Pease 1982]



Problème des généraux byzantins: Byzantine Generals Problem (BGP)

- Un général et ses armées ont encerclé une ville
- La ville est très forte et a jusqu'à présent résisté à leurs attaques
- Ils doivent choisir entre battre en retraite ou attaquer, et quand le faire
- Une attaque ou une retraite ***non coordonnée*** entraînera des pertes inacceptables



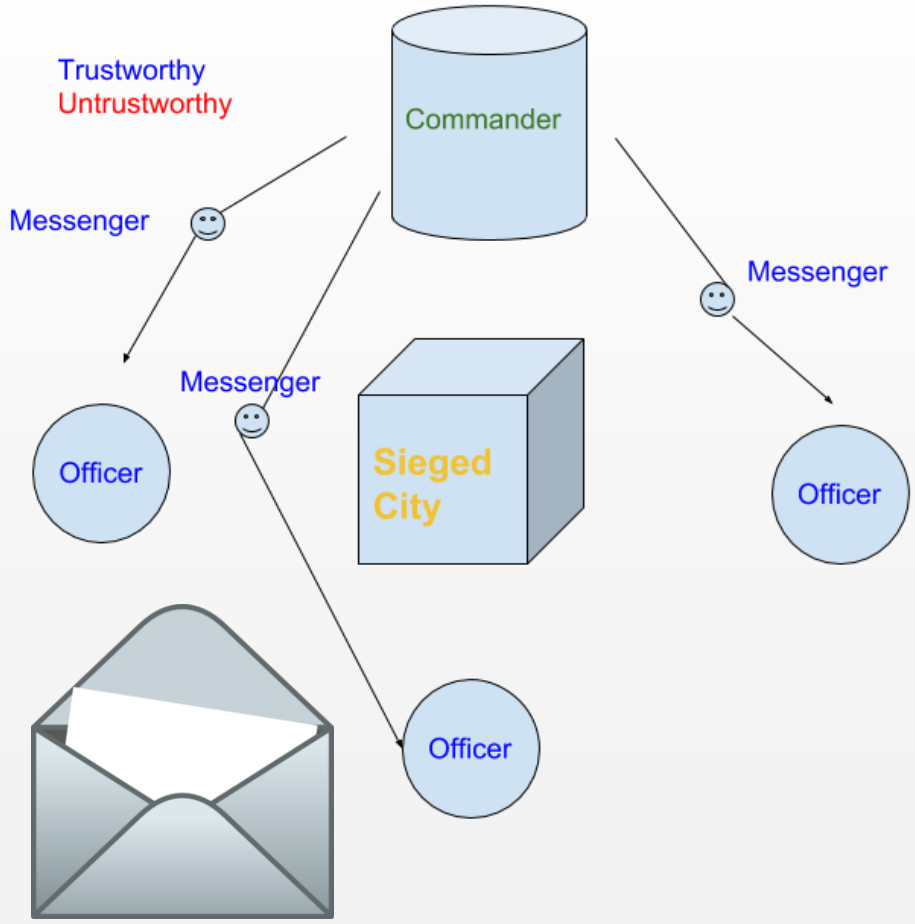


Problème des généraux byzantins: Byzantine Generals Problem (BGP)

- Les généraux et les officiers doivent communiquer par l'intermédiaire de messagers
  - Les officiers peuvent également communiquer entre eux à l'aide de **messagers**.

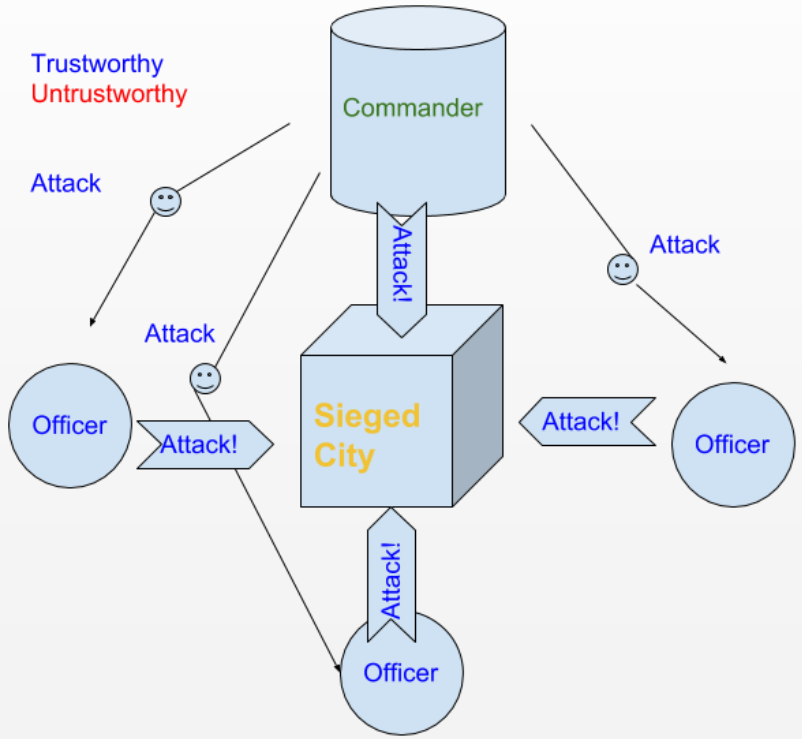
Toute partie impliquée peut être non fiable:

- Le général lui-même est peut-être non fiable
- Les officiers qui contrôlent ses armées peuvent être non fiables
- Les messagers peuvent être non fiable et peuvent également être enlevés par les forces ennemies

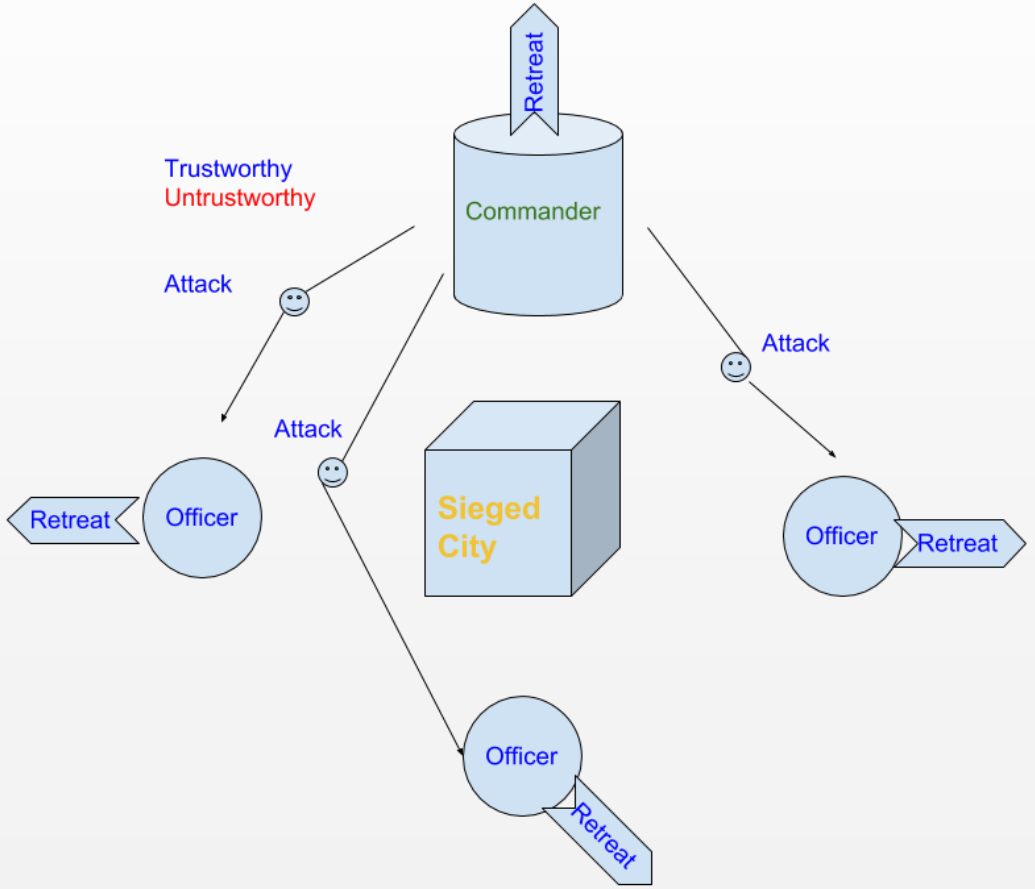


Résultats du BGP

Victoire!

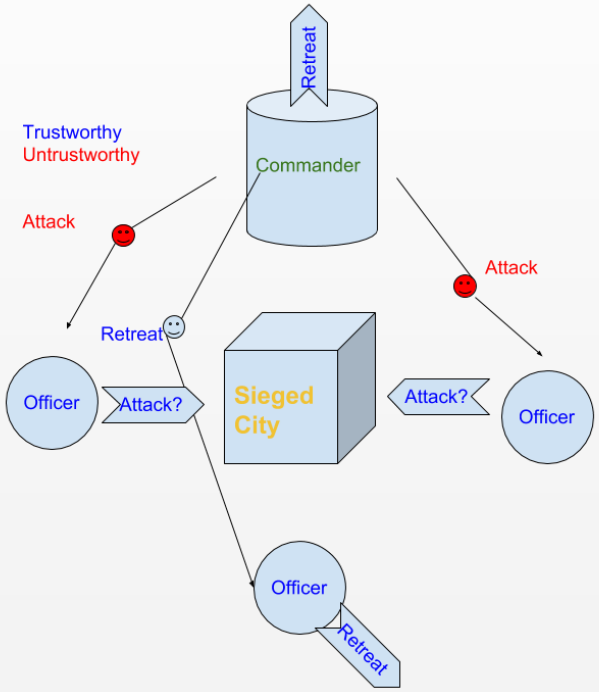


Échec: Accepter le Retour

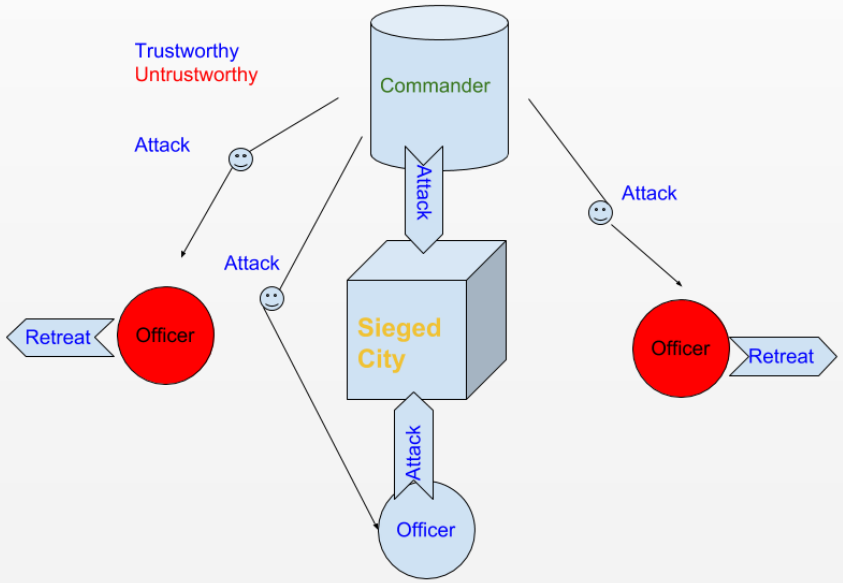


Résultats du BGP

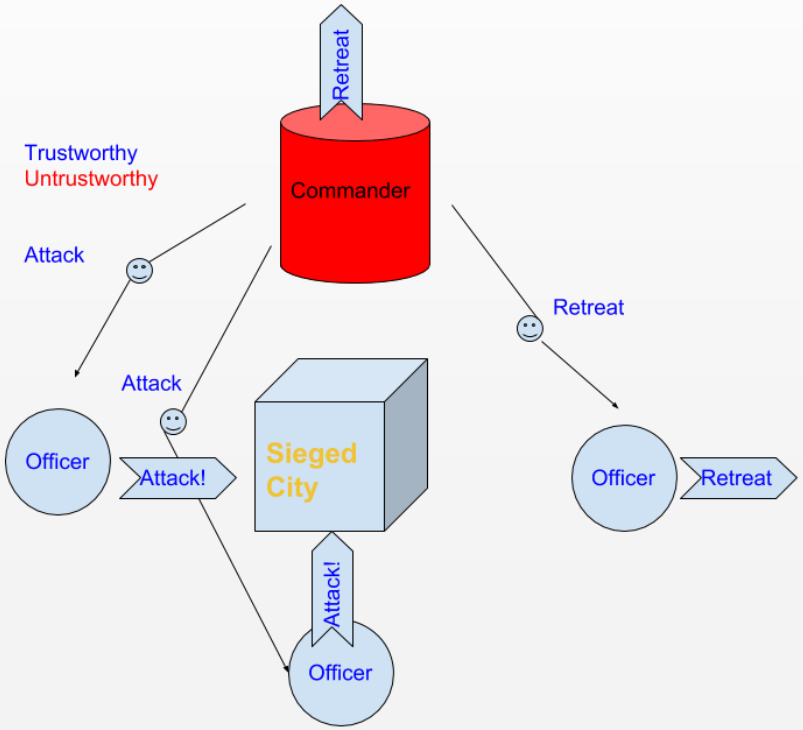
Échec ! (Messagers non fiables)



Échec ! (Officiers non fiables)



Échec ! (Général non fiable)



Diffusion tolérante aux pannes: Fault Tolerant Broadcast

Sécurité : tous les officiers décident de la même commande (s'ils décident).

Validité : si le général est honnête, tous les officiers décident de sa commande.

Disponibilité :

Tous les officiers finissent par décider [***Diffusion***].

Si le général est honnête, tous les officiers finissent par décider [**Diffusion *fiable***].

Scénarios et Situations de Consensus

- **En cas de désaccord sur l'état du réseau, un consensus permet de…**
  - Suspendre le réseau.
  - Imposer la décision du nœud le plus rapide.
  - Parvenir à une version unique acceptée par tous.
  - Mettre en place une gouvernance centralisée.
- **Dans un réseau sans consensus, chaque nœud peut…**
  - Modifier ses propres transactions.
  - Avoir sa propre version de l'historique des transactions.
  - Valider toutes les transactions sans confirmation.
  - Contrôler le réseau indépendamment.

Tolérance aux pannes byzantines: Byzantine Fault Tolerance (BFT)

- Un système est considéré comme tolérant aux pannes byzantines (BFT) s'il résiste aux dilemmes du problème des généraux byzantins
- Une faute byzantine est définie comme un mode de défaillance du système, soit une défaillance ou un fonctionnement incorrect, causé par une incapacité à atteindre un consensus au sein du système.

Tolérance aux pannes byzantines: Byzantine Fault Tolerance (BFT)

Il s'agit souvent du mode de tolérance aux pannes le plus difficile, car il n'inclut aucune règle sur les actions qu'un nœud peut entreprendre ni aucune hypothèse sur la manière dont une partie malveillante/défectueuse du système peut se comporter.

- ***Par exemple:*** le consensus peut être plus facile à atteindre si l’on suppose que les nœuds seront systématiquement honnêtes ou malhonnêtes dans leurs communications avec d’autres nœuds.

Tolérance aux pannes byzantines: Byzantine Fault Tolerance (BFT)

Doit être résilient dans un environnement extrêmement incertain

Les nœuds et les messages peuvent être bons ou mauvais à n'importe quelle fréquence

Des changements sans avertissement

Souvent considéré comme le critère de sécurité le plus strict dans les systèmes distribués pour cette raison

- Ce serait plus facile s'il y avait des hypothèses sur la fréquence/régularité de certains nœuds étant bons ou mauvais

Tolérance aux pannes byzantines: Byzantine Fault Tolerance (BFT)

La plupart des algorithmes de consensus utilisés dans les technologies blockchain sont tolérants aux pannes byzantines

- Certains algorithmes supposent que les nœuds malhonnêtes effectuent certaines actions ou envoient certains messages et ne sont donc pas des BFT.
- Parfois, la manière dont un système implémente l'algorithme de consensus détermine sa tolérance aux pannes

Tolérance aux pannes byzantines: Byzantine Fault Tolerance (BFT)

L'objectif des algorithmes de consensus de la blockchain est d'être BFT

La plupart des algorithmes sont (PoW, PoS, etc.)

Certains algorithmes incluent des hypothèses supplémentaires qui brisent le BFT

- Permettant souvent des accélérations
- Vitesse vs sécurité

Tolérance aux pannes

Nous ne pouvons pas garantir un accord dans un système asynchrone où même un seul hôte pourrait échouer.

Pourquoi ?

Nous ne pouvons pas détecter de manière fiable les pannes. Nous ne pouvons pas faire la distinction entre un hôte ou un ***réseau lent*** et un ***hôte en panne***.

Note : Nous pouvons toujours garantir la sécurité, le problème est limité à la garantie de la réactivité.

Solution pour ***Tolérance aux pannes***

En pratique :

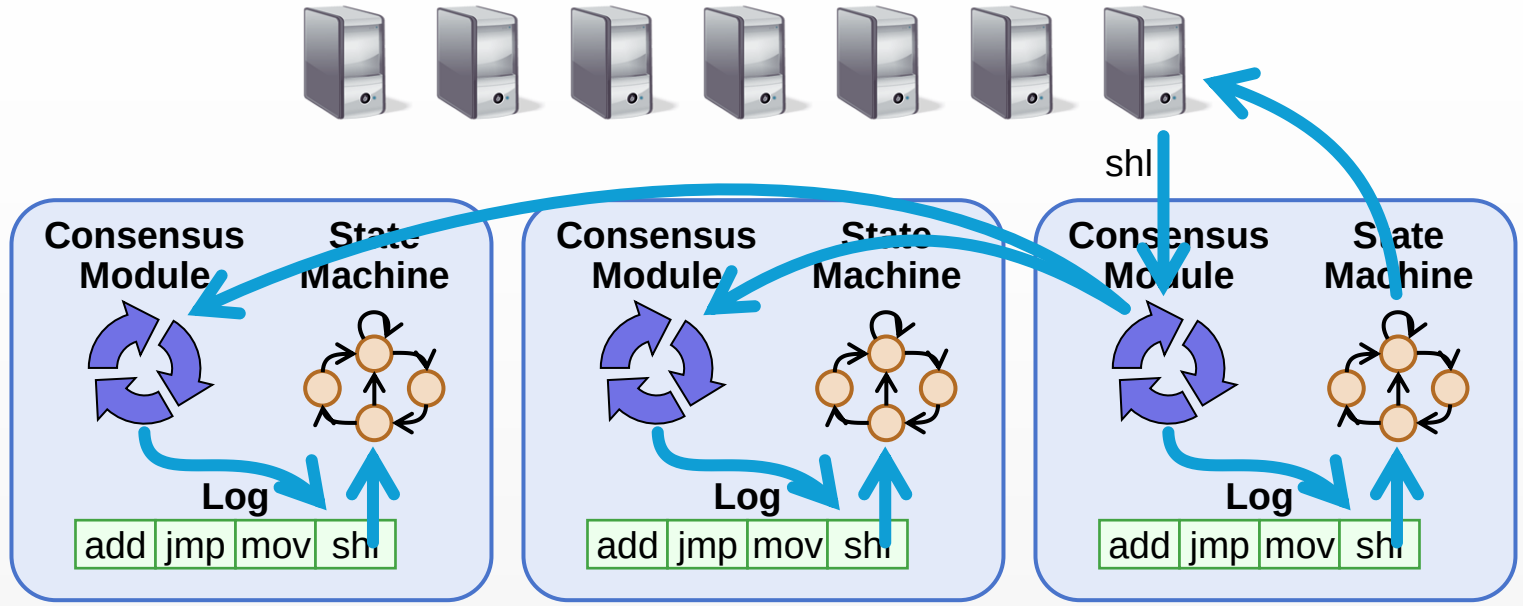
Nous acceptons que parfois le système ne soit pas disponible. Nous atténuons cela en utilisant des délais et des reprises.

En théorie :

Nous faisons des hypothèses plus faibles sur la synchronisation du système, par exemple, les messages arrivent dans un délai raisonnable.

Par analogie avec le Problème des généraux byzantins

- **Journal répliqué**



- **Journal répliqué => machine à états répliquée**
- Tous les serveurs exécutent les mêmes commandes dans le même ordre.
- Le module de consensus garantit une réplication correcte du journal.
- Le système progresse tant qu'une majorité de serveurs reste opérationnelle.
- **Modèle de défaillance : échec-arrêt (pas byzantin), messages retardés/perdus.**

State Machine Replication Problem (SMR Problem)

- **Overview of SMR Problem:**
  - Clients submit transactions (txs) to nodes in a distributed system.
  - Each node maintains a local append-only data structure, which is an ordered sequence of transactions.
  - The goal is to keep all nodes in sync, meaning each node should have identical local histories.
- **Définition d'un protocole :**
  - **Un protocole est défini comme le code qui s'exécute sur chaque nœud, responsable de :**
    - Effectuer des calculs locaux,
      Recevoir des messages d'autres nœuds et clients,
      Envoyer des messages à d'autres nœuds..
- **Objectifs du SMR :**
  - **Cohérence** : tous les nœuds doivent être d'accord sur l'historique, ce qui signifie qu'ils partagent la même séquence de transactions.
  - **Disponibilité :** *chaque transaction valide soumise par un client doit éventuellement être ajoutée à l'historique.*

Types d'algorithmes de consensus

- **Preuve de Travail (PoW) :** Mécanisme où les participants résolvent des puzzles mathématiques pour ajouter des blocs, nécessitant une grande puissance de calcul.
- **Preuve d'Enjeu (PoS) :** Les participants valident les transactions selon la quantité de cryptomonnaie qu'ils "mettent en jeu".
- **Preuve d'Enjeu Déléguée (DPoS) :** Variante de la PoS où un petit groupe de validateurs élus gère la validation des transactions.
- **Preuve d'Autorité (PoA) :** Quelques nœuds de confiance valident les transactions et ajoutent des blocs.
- **Preuve de Temps Écoulé (PoET) :** Les participants attendent une période aléatoire avant de proposer un bloc.
- **Preuve de Capacité (PoC) et Preuve d'Espace (PoSpace) :** Les participants utilisent leur espace de stockage pour le consensus.
- **Preuve de Brûlage (PoB)** : Les participants "brûlent" des tokens pour prouver leur engagement.

Types d'algorithmes de consensus

Questions ?!!