Analyse technique et critique de la blockchain

Polytech Nancy 2A

Jonathan Weber

Automne 2025

Plan de la séance

1. Présentation du cours
2. La blockchain, qu'est ce que c'est ?
3. Notions de crypto ... graphie
4. La blockchain, comment ça fonctionne ?
5. Limites et critiques de la blockchain
6. Conclusion

Présentation du cours

Contenu du cours

Volume horaire

- Cours : 4h
- Travaux pratiques : 8h
- Enseignant : J. Weber

Contenu du cours

Volume horaire

- Cours : 4h
- Travaux pratiques : 8h
- Enseignant : J. Weber

Contenu

- La blockchain, qu'est ce que c'est ?
- La blockchain, comment ça fonctionne ?
- Critique de la blockchain
- ...

Contenu du cours

Volume horaire

- Cours : 4h
- Travaux pratiques : 8h
- Enseignant : J. Weber

Contenu

- La blockchain, qu'est ce que c'est ?
- La blockchain, comment ça fonctionne ?
- Critique de la blockchain
- ...

Langage support

- Python

La blockchain, qu'est ce que c'est ?

Blockchain ?

- Un registre partagé
  - C'est comme un grand cahier de comptes
  - Copié sur de nombreux ordinateurs dans le monde

- Un registre partagé
  - C'est comme un grand cahier de comptes
  - Copié sur de nombreux ordinateurs dans le monde
- Des blocs chaînés
  - Les informations sont regroupées en « blocs »
  - Chaque bloc pointe vers le précédent : on forme une chaîne

- Un registre partagé
  - C'est comme un grand cahier de comptes
  - Copié sur de nombreux ordinateurs dans le monde
- Des blocs chaînés
  - Les informations sont regroupées en « blocs »
  - Chaque bloc pointe vers le précédent : on forme une chaîne
- Immuable
  - Une fois écrit, un bloc ne peut plus être modifié sans que tout se voie
  - Changer le passé oblige à recalculer toute la chaîne

Principes fondamentaux de la blockchain

- Un registre partagé
  - C'est comme un grand cahier de comptes
  - Copié sur de nombreux ordinateurs dans le monde
- Des blocs chaînés
  - Les informations sont regroupées en « blocs »
  - Chaque bloc pointe vers le précédent : on forme une chaîne
- Immuable
  - Une fois écrit, un bloc ne peut plus être modifié sans que tout se voie
  - Changer le passé oblige à recalculer toute la chaîne
- Pas de tiers de confiance
  - Pas de banque ou d'autorité centrale
  - Ce sont les ordinateurs du réseau qui se mettent d'accord (consensus)

Principes fondamentaux de la blockchain

- Un registre partagé
  - C'est comme un grand cahier de comptes
  - Copié sur de nombreux ordinateurs dans le monde
- Des blocs chaînés
  - Les informations sont regroupées en « blocs »
  - Chaque bloc pointe vers le précédent : on forme une chaîne
- Immuable
  - Une fois écrit, un bloc ne peut plus être modifié sans que tout se voie
  - Changer le passé oblige à recalculer toute la chaîne
- Pas de tiers de confiance
  - Pas de banque ou d'autorité centrale
  - Ce sont les ordinateurs du réseau qui se mettent d'accord (consensus)
- Transparente mais pseudonyme
  - Tout le monde peut voir les transactions
  - Mais on voit des adresses, pas des noms réels

Conséquences techniques des principes

- Registre partagé
  - Chaque nœud stocke une copie quasi complète des données
  - Nécessite un protocole réseau P2P pour échanger blocs et transactions

Conséquences techniques des principes

- Registre partagé
  - Chaque nœud stocke une copie quasi complète des données
  - Nécessite un protocole réseau P2P pour échanger blocs et transactions
- Blocs chaînés par hash
  - Utilisation intensive de fonctions de hachage (ex : SHA-256)
  - Chaque bloc contient le hash du précédent dans son en-tête
  - Tout changement de données → recalcul en cascade des hash

Conséquences techniques des principes

- Registre partagé
  - Chaque nœud stocke une copie quasi complète des données
  - Nécessite un protocole réseau P2P pour échanger blocs et transactions
- Blocs chaînés par hash
  - Utilisation intensive de fonctions de hachage (ex : SHA-256)
  - Chaque bloc contient le hash du précédent dans son en-tête
  - Tout changement de données → recalcul en cascade des hash
- Immutabilité
  - On ne met pas à jour : on **ajoute** un nouveau bloc
  - Les bases de données sont pensées comme « append-only »

Conséquences techniques des principes

- Registre partagé
  - Chaque nœud stocke une copie quasi complète des données
  - Nécessite un protocole réseau P2P pour échanger blocs et transactions
- Blocs chaînés par hash
  - Utilisation intensive de fonctions de hachage (ex : SHA-256)
  - Chaque bloc contient le hash du précédent dans son en-tête
  - Tout changement de données → recalcul en cascade des hash
- Immutabilité
  - On ne met pas à jour : on **ajoute** un nouveau bloc
  - Les bases de données sont pensées comme « append-only »
- Pas de tiers de confiance
  - Besoin d'un **algorithme de consensus** (PoW, PoS, ...)
  - Vérification locale de toutes les règles par chaque nœud (signatures, balances, format)

Conséquences techniques des principes

- Registre partagé
  - Chaque nœud stocke une copie quasi complète des données
  - Nécessite un protocole réseau P2P pour échanger blocs et transactions
- Blocs chaînés par hash
  - Utilisation intensive de fonctions de hachage (ex : SHA-256)
  - Chaque bloc contient le hash du précédent dans son en-tête
  - Tout changement de données → recalcul en cascade des hash
- Immutabilité
  - On ne met pas à jour : on **ajoute** un nouveau bloc
  - Les bases de données sont pensées comme « append-only »
- Pas de tiers de confiance
  - Besoin d'un **algorithme de consensus** (PoW, PoS, ...)
  - Vérification locale de toutes les règles par chaque nœud (signatures, balances, format)
- Transparence & pseudonymat
  - Toutes les transactions sont publiques → contraintes fortes sur la vie privée
  - Gestion des identités via paires de clés (crypto asymétrique), pas via comptes classiques

À quoi sert une blockchain ?

- Enregistrer des transactions sans tiers de confiance
  - Ex : envoyer de la valeur (crypto-monnaies) sans banque
  - Tout le monde peut vérifier que les règles sont respectées

À quoi sert une blockchain ?

- Enregistrer des transactions sans tiers de confiance
  - Ex : envoyer de la valeur (crypto-monnaies) sans banque
  - Tout le monde peut vérifier que les règles sont respectées
- Garantir l'intégrité de l'historique
  - Une fois inscrit, un événement est très difficile à falsifier
  - Utile pour tracer des opérations (paiements, transferts, états)

À quoi sert une blockchain ?

- Enregistrer des transactions sans tiers de confiance
  - Ex : envoyer de la valeur (crypto-monnaies) sans banque
  - Tout le monde peut vérifier que les règles sont respectées
- Garantir l'intégrité de l'historique
  - Une fois inscrit, un événement est très difficile à falsifier
  - Utile pour tracer des opérations (paiements, transferts, états)
- Automatiser des règles avec les smart contracts
  - Programmes qui s'exécutent automatiquement quand les conditions sont réunies
  - Ex : libérer un paiement si une certaine date ou condition est atteinte

À quoi sert une blockchain ?

- Enregistrer des transactions sans tiers de confiance
  - Ex : envoyer de la valeur (crypto-monnaies) sans banque
  - Tout le monde peut vérifier que les règles sont respectées
- Garantir l'intégrité de l'historique
  - Une fois inscrit, un événement est très difficile à falsifier
  - Utile pour tracer des opérations (paiements, transferts, états)
- Automatiser des règles avec les smart contracts
  - Programmes qui s'exécutent automatiquement quand les conditions sont réunies
  - Ex : libérer un paiement si une certaine date ou condition est atteinte
- Coordonner des acteurs qui ne se font pas confiance
  - Entreprises, particuliers, organisations de pays différents
  - Chacun peut vérifier les données sans dépendre d'un acteur central

À quoi sert une blockchain ?

- Enregistrer des transactions sans tiers de confiance
  - Ex : envoyer de la valeur (crypto-monnaies) sans banque
  - Tout le monde peut vérifier que les règles sont respectées
- Garantir l'intégrité de l'historique
  - Une fois inscrit, un événement est très difficile à falsifier
  - Utile pour tracer des opérations (paiements, transferts, états)
- Automatiser des règles avec les smart contracts
  - Programmes qui s'exécutent automatiquement quand les conditions sont réunies
  - Ex : libérer un paiement si une certaine date ou condition est atteinte
- Coordonner des acteurs qui ne se font pas confiance
  - Entreprises, particuliers, organisations de pays différents
  - Chacun peut vérifier les données sans dépendre d'un acteur central
- Créer de nouveaux types d'applications
  - Finance décentralisée (DeFi), NFTs, systèmes de vote, etc.

Notions de crypto … graphie

Fonction de hachage cryptographique : c'est quoi ?

- Boîte noire mathématique :
  - Entrée : des données de taille quelconque (texte, fichier, bloc...)
  - Sortie : une "empreinte" de taille fixe (ex : 256 bits)

Fonction de hachage cryptographique : c'est quoi ?

- Boîte noire mathématique :
  - Entrée : des données de taille quelconque (texte, fichier, bloc...)
  - Sortie : une "empreinte" de taille fixe (ex : 256 bits)
- Toujours la même sortie pour la même entrée
  - Si on hash la même donnée, on obtient toujours le même résultat
  - Fonction **déterministe**

Fonction de hachage cryptographique : c'est quoi ?

- Boîte noire mathématique :
  - Entrée : des données de taille quelconque (texte, fichier, bloc...)
  - Sortie : une "empreinte" de taille fixe (ex : 256 bits)
- Toujours la même sortie pour la même entrée
  - Si on hash la même donnée, on obtient toujours le même résultat
  - Fonction **déterministe**
- Petite différence, grande conséquence
  - Changer un seul bit en entrée donne un hash totalement différent
  - On parle d'**effet avalanche**
    - Polytech Nancy → bb497074c841a70402f10a05af9784443ce2fb38 (sha1)
    - Polyt3ch Nancy → a6cc6faa8f9230b51578d4e98ce899ece40de8ca (sha1)

Fonction de hachage cryptographique : c'est quoi ?

- Boîte noire mathématique :
  - Entrée : des données de taille quelconque (texte, fichier, bloc...)
  - Sortie : une "empreinte" de taille fixe (ex : 256 bits)
- Toujours la même sortie pour la même entrée
  - Si on hash la même donnée, on obtient toujours le même résultat
  - Fonction **déterministe**
- Petite différence, grande conséquence
  - Changer un seul bit en entrée donne un hash totalement différent
  - On parle d'**effet avalanche**
    - Polytech Nancy → bb497074c841a70402f10a05af9784443ce2fb38 (sha1)
    - Polyt3ch Nancy → a6cc6faa8f9230b51578d4e98ce899ece40de8ca (sha1)
- Exemples :
  - SHA-256, SHA-3/Keccak, BLAKE2, ...

Fonction de hachage : comment ça fonctionne ?

- Compression de données
  - Prend un message de longueur quelconque
  - Le découpe en blocs et les traite avec un algorithme interne
  - Au final : un **résumé** de taille fixe (digest)

Fonction de hachage : comment ça fonctionne ?

- Compression de données
  - Prend un message de longueur quelconque
  - Le découpe en blocs et les traite avec un algorithme interne
  - Au final : un **résumé** de taille fixe (digest)
- Propriétés de sécurité
  - Résistance à la préimage
    - Donné $y = H(x)$, trouver un x' tel que $H(x') = y$ doit être calculatoirement difficile

Fonction de hachage : comment ça fonctionne ?

- Compression de données
  - Prend un message de longueur quelconque
  - Le découpe en blocs et les traite avec un algorithme interne
  - Au final : un **résumé** de taille fixe (digest)
- Propriétés de sécurité
  - Résistance à la préimage
    - Donné $y = H(x)$, trouver un x' tel que $H(x') = y$ doit être calculatoirement difficile
  - Résistance à la seconde préimage
    - Donné un message x, il doit être difficile de trouver $x' \neq x$ tel que $H(x') = H(x)$

Fonction de hachage : comment ça fonctionne ?

- Compression de données
  - Prend un message de longueur quelconque
  - Le découpe en blocs et les traite avec un algorithme interne
  - Au final : un **résumé** de taille fixe (digest)
- Propriétés de sécurité
  - Résistance à la préimage
    - Donné $y = H(x)$, trouver un x' tel que $H(x') = y$ doit être calculatoirement difficile
  - Résistance à la seconde préimage
    - Donné un message x, il doit être difficile de trouver $x' \neq x$ tel que $H(x') = H(x)$
  - Résistance aux collisions
    - Donné deux messages distincts x et x', la probabilité d'avoir $H(x') = H(x)$ doit être très faible

Fonction de hachage : comment ça fonctionne ?

- Compression de données
  - Prend un message de longueur quelconque
  - Le découpe en blocs et les traite avec un algorithme interne
  - Au final : un **résumé** de taille fixe (digest)
- Propriétés de sécurité
  - Résistance à la préimage
    - Donné $y = H(x)$, trouver un x' tel que $H(x') = y$ doit être calculatoirement difficile
  - Résistance à la seconde préimage
    - Donné un message x, il doit être difficile de trouver $x' \neq x$ tel que $H(x') = H(x)$
  - Résistance aux collisions
    - Donné deux messages distincts x et x', la probabilité d'avoir $H(x') = H(x)$ doit être très faible
- Pas une fonction de chiffrement
  - On ne déchiffre pas un hash
  - Il n'y a pas de clé secrète : c'est une fonction publique, à sens unique

Fonction de hachage : comment ça fonctionne ?

- Compression de données
  - Prend un message de longueur quelconque
  - Le découpe en blocs et les traite avec un algorithme interne
  - Au final : un **résumé** de taille fixe (digest)
- Propriétés de sécurité
  - Résistance à la préimage
    - Donné $y = H(x)$, trouver un x' tel que $H(x') = y$ doit être calculatoirement difficile
  - Résistance à la seconde préimage
    - Donné un message x, il doit être difficile de trouver $x' \neq x$ tel que $H(x') = H(x)$
  - Résistance aux collisions
    - Donné deux messages distincts x et x', la probabilité d'avoir $H(x') = H(x)$ doit être très faible
- Pas une fonction de chiffrement
  - On ne déchiffre pas un hash
  - Il n'y a pas de clé secrète : c'est une fonction publique, à sens unique
- ⚠ À ne pas confondre avec les tables de hachage
  - Les tables de hachage (structures de données) utilisent un hachage, mais pas forcément cryptographique
  - En blockchain, on a besoin de fonctions de hachage conçues pour la sécurité, pas seulement pour accélérer les recherches

Pourquoi utiliser une fonction de hachage ?

- Vérifier l'intégrité des données
  - Si les données changent, le hash change
  - Permet de détecter toute modification ou corruption

Pourquoi utiliser une fonction de hachage ?

- Vérifier l'intégrité des données
  - Si les données changent, le hash change
  - Permet de détecter toute modification ou corruption
- Chaîner les blocs dans une blockchain
  - Chaque bloc contient le hash du bloc précédent
  - Modifier un ancien bloc ⇒ tous les hash suivants deviennent faux

Pourquoi utiliser une fonction de hachage ?

- Vérifier l'intégrité des données
  - Si les données changent, le hash change
  - Permet de détecter toute modification ou corruption
- Chaîner les blocs dans une blockchain
  - Chaque bloc contient le hash du bloc précédent
  - Modifier un ancien bloc ⇒ tous les hash suivants deviennent faux
- Construire des structures efficaces
  - Arbres de Merkle pour prouver qu'une transaction est dans un bloc
  - Adresses dérivées de clés publiques, identifiants uniques, etc.

Pourquoi utiliser une fonction de hachage ?

- Vérifier l'intégrité des données
  - Si les données changent, le hash change
  - Permet de détecter toute modification ou corruption
- Chaîner les blocs dans une blockchain
  - Chaque bloc contient le hash du bloc précédent
  - Modifier un ancien bloc ⇒ tous les hash suivants deviennent faux
- Construire des structures efficaces
  - Arbres de Merkle pour prouver qu'une transaction est dans un bloc
  - Adresses dérivées de clés publiques, identifiants uniques, etc.
- Créer une preuve de travail (PoW)
  - On cherche un nonce tel que le hash respecte une certaine forme
  - Trouver ce nonce demande du calcul, mais vérifier est très rapide

Chiffrement asymétrique : idée de base

- Deux clés différentes
  - Une clé publique : peut être partagée avec tout le monde
  - Une clé privée : doit rester secrète

Chiffrement asymétrique : idée de base

- Deux clés différentes
  - Une clé publique : peut être partagée avec tout le monde
  - Une clé privée : doit rester secrète
- Objectif
  - Chiffrer des messages pour que seul le détenteur de la clé privée puisse les lire

Chiffrement asymétrique : idée de base

- Deux clés différentes
  - Une clé publique : peut être partagée avec tout le monde
  - Une clé privée : doit rester secrète
- Objectif
  - Chiffrer des messages pour que seul le détenteur de la clé privée puisse les lire
- Intuition
  - Clé publique = « boîte aux lettres » : tout le monde peut déposer un message
  - Clé privée = « clé de la boîte » : une seule personne peut l'ouvrir

Chiffrement asymétrique : idée de base

- Deux clés différentes
  - Une clé publique : peut être partagée avec tout le monde
  - Une clé privée : doit rester secrète
- Objectif
  - Chiffrer des messages pour que seul le détenteur de la clé privée puisse les lire
- Intuition
  - Clé publique = « boîte aux lettres » : tout le monde peut déposer un message
  - Clé privée = « clé de la boîte » : une seule personne peut l'ouvrir
- Exemples
  - RSA, schémas à base de courbes elliptiques (ECIES, etc.)

Chiffrement asymétrique : fonctionnement

- Génération de clés
  - L'utilisateur génère une paire : (sk, pk)
  - pk : clé publique, sk : clé privée

Chiffrement asymétrique : fonctionnement

- Génération de clés
  - L'utilisateur génère une paire : (sk, pk)
  - pk : clé publique, sk : clé privée
- Chiffrement
  - Alice veut envoyer un message m à Bob
  - Elle utilise la clé publique de Bob :

$$c = \text{encrypt}(m, pk_{\text{Bob}})$$

  - c : texte chiffré transmis sur le réseau

Chiffrement asymétrique : fonctionnement

- Génération de clés
  - L'utilisateur génère une paire : (sk, pk)
  - pk : clé publique, sk : clé privée
- Chiffrement
  - Alice veut envoyer un message m à Bob
  - Elle utilise la clé publique de Bob :

$$c = \text{encrypt}(m, pk_{\text{Bob}})$$

  - c : texte chiffré transmis sur le réseau
- Déchiffrement
  - Bob reçoit c
  - Il utilise sa clé privée :

$$m = \text{decrypt}(c, sk_{\text{Bob}})$$

  - Sans sk_{Bob}, retrouver m doit être "impossible" en pratique

Chiffrement asymétrique : propriétés et limites

- Propriétés
  - Permet d'échanger des secrets sans clé partagée au départ
  - Permet d'associer une clé publique à une identité (certificats, PKI)

Chiffrement asymétrique : propriétés et limites

- Propriétés
  - Permet d'échanger des secrets sans clé partagée au départ
  - Permet d'associer une clé publique à une identité (certificats, PKI)
- Limites
  - Coût en calcul élevé
  - En pratique : on chiffre surtout une clé symétrique, puis on chiffre les données avec un algorithme symétrique rapide (AES, ...)

Chiffrement asymétrique : propriétés et limites

- Propriétés
  - Permet d'échanger des secrets sans clé partagée au départ
  - Permet d'associer une clé publique à une identité (certificats, PKI)
- Limites
  - Coût en calcul élevé
  - En pratique : on chiffre surtout une clé symétrique, puis on chiffre les données avec un algorithme symétrique rapide (AES, ...)
- Lien avec les signatures
  - Même famille d'outils mathématiques
  - Pour signer, on exploite la clé privée dans l'autre sens : « je prouve que je possède sk lié à pk »

Signature cryptographique : c'est quoi ?

- Équivalent numérique d'une signature manuscrite
  - Permet de « signer » un message, un document, une transaction
  - Le signataire ne peut pas nier avoir signé (non-répudiation)

Signature cryptographique : c'est quoi ?

- Équivalent numérique d'une signature manuscrite
  - Permet de « signer » un message, un document, une transaction
  - Le signataire ne peut pas nier avoir signé (non-répudiation)
- Basée sur la crypto asymétrique
  - Clé privée : sert à produire la signature (doit rester secrète)
  - Clé publique : sert à vérifier la signature (peut être diffusée)

Signature cryptographique : c'est quoi ?

- Équivalent numérique d'une signature manuscrite
  - Permet de « signer » un message, un document, une transaction
  - Le signataire ne peut pas nier avoir signé (non-répudiation)
- Basée sur la crypto asymétrique
  - Clé privée : sert à produire la signature (doit rester secrète)
  - Clé publique : sert à vérifier la signature (peut être diffusée)
- Lien fort message ↔ signataire
  - La signature dépend du contenu du message
  - Toute modification du message invalide la signature

Signature cryptographique : c'est quoi ?

- Équivalent numérique d'une signature manuscrite
  - Permet de « signer » un message, un document, une transaction
  - Le signataire ne peut pas nier avoir signé (non-répudiation)
- Basée sur la crypto asymétrique
  - Clé privée : sert à produire la signature (doit rester secrète)
  - Clé publique : sert à vérifier la signature (peut être diffusée)
- Lien fort message ↔ signataire
  - La signature dépend du contenu du message
  - Toute modification du message invalide la signature
- Exemples
  - RSA, ECDSA, EdDSA (Ed25519), ...

Signature cryptographique : comment ça marche ?

- Génération de clés
  - L'utilisateur génère une paire : (sk, pk)
  - sk : clé privée, pk : clé publique

Signature cryptographique : comment ça marche ?

- Génération de clés
  - L'utilisateur génère une paire : (sk, pk)
  - sk : clé privée, pk : clé publique
- Signer un message
  - On calcule d'abord un hash du message : $h = H(m)$
  - On applique l'algorithme de signature avec sk :

$$\sigma = \text{Sign}(h, sk)$$

  - σ : la signature (suite de bits)

Signature cryptographique : comment ça marche ?

- Génération de clés
  - L'utilisateur génère une paire : (sk, pk)
  - sk : clé privée, pk : clé publique
- Signer un message
  - On calcule d'abord un hash du message : $h = H(m)$
  - On applique l'algorithme de signature avec sk :

$$\sigma = \text{Sign}(h, sk)$$

  - σ : la signature (suite de bits)
- Vérifier une signature
  - Le vérificateur connaît : m, σ, pk
  - Il recalcule $h = H(m)$, puis vérifie :

$$\text{Verify}(h, \sigma, pk) \in \{\text{Vrai}, \text{Faux}\}$$

  - Si Vrai : le message n'a pas été modifié et vient bien du détenteur de sk

Pourquoi utiliser des signatures cryptographiques ?

- Authentifier l'émetteur
  - Seul le détenteur de la clé privée peut produire une signature valide
  - On peut lier une action (transaction, message) à une identité (clé publique)

Pourquoi utiliser des signatures cryptographiques ?

- Authentifier l'émetteur
  - Seul le détenteur de la clé privée peut produire une signature valide
  - On peut lier une action (transaction, message) à une identité (clé publique)
- Garantir l'intégrité
  - Si le message change, la signature ne correspond plus
  - Toute altération est détectée automatiquement

Pourquoi utiliser des signatures cryptographiques ?

- Authentifier l'émetteur
  - Seul le détenteur de la clé privée peut produire une signature valide
  - On peut lier une action (transaction, message) à une identité (clé publique)
- Garantir l'intégrité
  - Si le message change, la signature ne correspond plus
  - Toute altération est détectée automatiquement
- Remplacer un tiers de confiance
  - Pas besoin de faire confiance à un serveur ou à une banque
  - On fait confiance aux propriétés mathématiques de l'algorithme

Pourquoi utiliser des signatures cryptographiques ?

- Authentifier l'émetteur
  - Seul le détenteur de la clé privée peut produire une signature valide
  - On peut lier une action (transaction, message) à une identité (clé publique)
- Garantir l'intégrité
  - Si le message change, la signature ne correspond plus
  - Toute altération est détectée automatiquement
- Remplacer un tiers de confiance
  - Pas besoin de faire confiance à un serveur ou à une banque
  - On fait confiance aux propriétés mathématiques de l'algorithme
- En blockchain
  - Chaque transaction est signée par l'émetteur
  - Les nœuds vérifient localement que l'émetteur autorise la dépense
  - C'est la base de la sécurité des comptes et des smart contracts

La blockchain, comment ça fonctionne ?

Structure générale d'une chaîne de blocs

- Une blockchain est une suite ordonnée de blocs
  - $B_0, B_1, B_2, \ldots, B_n$
  - Chaque nouveau bloc s'ajoute à la fin de la chaîne

Structure générale d'une chaîne de blocs

- Une blockchain est une suite ordonnée de blocs
  - $B_0, B_1, B_2, \ldots, B_n$
  - Chaque nouveau bloc s'ajoute à la fin de la chaîne
- Chaque bloc pointe vers son prédécesseur
  - Le header de B_i contient le hash de B_{i-1}
  - Ce lien forme un chaînage cryptographique

Structure générale d'une chaîne de blocs

- Une blockchain est une suite ordonnée de blocs
  - $B_0, B_1, B_2, \ldots, B_n$
  - Chaque nouveau bloc s'ajoute à la fin de la chaîne
- Chaque bloc pointe vers son prédécesseur
  - Le header de B_i contient le hash de B_{i-1}
  - Ce lien forme un chaînage cryptographique
- Immutabilité
  - Modifier un bloc change son hash
  - Donc le bloc suivant ne reconnaît plus son prédécesseur
  - Toute tentative de falsification devient immédiatement visible

Structure générale d'une chaîne de blocs

- Une blockchain est une suite ordonnée de blocs
  - $B_0, B_1, B_2, \ldots, B_n$
  - Chaque nouveau bloc s'ajoute à la fin de la chaîne
- Chaque bloc pointe vers son prédécesseur
  - Le header de B_i contient le hash de B_{i-1}
  - Ce lien forme un chaînage cryptographique
- Immutabilité
  - Modifier un bloc change son hash
  - Donc le bloc suivant ne reconnaît plus son prédécesseur
  - Toute tentative de falsification devient immédiatement visible
- Historique linéaire et vérifiable
  - Chaque nœud peut vérifier l'intégrité de toute la chaîne
  - La blockchain agit comme un registre `append-only`

Exercice : Vérification d'intégrité

Mini-blockchain :

- Block 0 : data = "A" ; prev_hash = 0
- Block 1 : data = "B" ; prev_hash = hash(Block 0)
- Block 2 : data = "C" ; prev_hash = hash(Block 1)

Tâches :

1. Décrire un algorithme qui vérifie l'intégrité de la chaîne.
2. Expliquer ce qui se passe si on modifie data = "A" dans le bloc 0.

Correction Exercice 1

- Algorithme de vérification :
  - pour i de 1 à n :
  - recalculer hash(Block $i-1$)
  - vérifier Block i.prev_hash == hash(Block $i-1$)

Correction Exercice 1

- Algorithme de vérification :
  - pour i de 1 à n :
  - recalculer hash(Block $i-1$)
  - vérifier Block i.prev_hash == hash(Block $i-1$)
- Si modification de Block 0 :
  - son hash change
  - différent de prev_hash de Block 1
  - la chaîne devient invalide à partir de Block 1

Le concept d'adresse en blockchain

- Qu'est-ce qu'une adresse ?
  - Identifiant d'un compte ou d'un destinataire sur la blockchain.
  - C'est ce que l'on donne pour « recevoir » des fonds ou interagir avec un smart contract.

Le concept d'adresse en blockchain

- Qu'est-ce qu'une adresse ?
  - Identifiant d'un compte ou d'un destinataire sur la blockchain.
  - C'est ce que l'on donne pour « recevoir » des fonds ou interagir avec un smart contract.
- Lien avec les clés cryptographiques
  - Une adresse est généralement dérivée d'une clé publique (via hash + encodage).
  - La clé privée associée permet de signer les opérations (dépenser, appeler un contrat, etc.).
  - On peut connaître l'adresse, jamais la clé privée (en théorie).

Le concept d'adresse en blockchain

- Qu'est-ce qu'une adresse ?
  - Identifiant d'un compte ou d'un destinataire sur la blockchain.
  - C'est ce que l'on donne pour « recevoir » des fonds ou interagir avec un smart contract.
- Lien avec les clés cryptographiques
  - Une adresse est généralement dérivée d'une clé publique (via hash + encodage).
  - La clé privée associée permet de signer les opérations (dépenser, appeler un contrat, etc.).
  - On peut connaître l'adresse, jamais la clé privée (en théorie).
- Pseudonymat
  - Une adresse ne contient pas le nom de la personne : c'est un identifiant pseudo-aléatoire.
  - Mais toutes les opérations liées à cette adresse sont publiques dans l'historique.

Le concept d'adresse en blockchain

- Qu'est-ce qu'une adresse ?
  - Identifiant d'un compte ou d'un destinataire sur la blockchain.
  - C'est ce que l'on donne pour « recevoir » des fonds ou interagir avec un smart contract.
- Lien avec les clés cryptographiques
  - Une adresse est généralement dérivée d'une clé publique (via hash + encodage).
  - La clé privée associée permet de signer les opérations (dépenser, appeler un contrat, etc.).
  - On peut connaître l'adresse, jamais la clé privée (en théorie).
- Pseudonymat
  - Une adresse ne contient pas le nom de la personne : c'est un identifiant pseudo-aléatoire.
  - Mais toutes les opérations liées à cette adresse sont publiques dans l'historique.
- Types d'adresses
  - Adresses de comptes utilisateurs (contrôlés par une clé privée).
  - Adresses de smart contracts (pointent vers du code on-chain).

La blockchain, comment ça fonctionne ?

Les blocs

Qu'est-ce qu'un bloc de blockchain ?

- Unité de base de l'historique
  - Un bloc regroupe un ensemble d'opérations validées
  - Il s'ajoute à la fin de la chaîne existante

Qu'est-ce qu'un bloc de blockchain ?

- Unité de base de l'historique
  - Un bloc regroupe un ensemble d'opérations validées
  - Il s'ajoute à la fin de la chaîne existante
- Deux grandes parties
  - En-tête (header) : métadonnées techniques
  - Corps (body) : contenu métier (opérations, événements, messages)

Qu'est-ce qu'un bloc de blockchain ?

- Unité de base de l'historique
  - Un bloc regroupe un ensemble d'opérations validées
  - Il s'ajoute à la fin de la chaîne existante
- Deux grandes parties
  - En-tête (header) : métadonnées techniques
  - Corps (body) : contenu métier (opérations, événements, messages)
- Rôle global
  - Structurer l'historique en « paquets » de données
  - Servir de point de référence pour la validation et la synchronisation

En-tête d'un bloc (1/2) : chaînage et identité

- Hash du bloc précédent
  - Champ `previous_block_hash`
  - Assure le lien explicite avec le bloc précédent
  - Toute modification dans un bloc casse ce chaînage

En-tête d'un bloc (1/2) : chaînage et identité

- Hash du bloc précédent
  - Champ `previous_block_hash`
  - Assure le lien explicite avec le bloc précédent
  - Toute modification dans un bloc casse ce chaînage
- Identifiant du bloc
  - Souvent défini comme le hash de l'en-tête du bloc
  - Permet de référencer un bloc de façon unique

En-tête d'un bloc (1/2) : chaînage et identité

- Hash du bloc précédent
  - Champ `previous_block_hash`
  - Assure le lien explicite avec le bloc précédent
  - Toute modification dans un bloc casse ce chaînage
- Identifiant du bloc
  - Souvent défini comme le hash de l'en-tête du bloc
  - Permet de référencer un bloc de façon unique
- Effet sur l'immutabilité
  - Modifier un bloc change son hash
  - Donc le hash stocké dans le bloc suivant ne correspond plus
  - L'historique devient facilement falsifiable à détecter

En-tête d'un bloc (2/2) : résumé et consensus

- Résumé du contenu
  - Champ de type `merkle_root` ou `state_root`
  - Hash qui résume toutes les opérations ou l'état global
  - Permet de vérifier le contenu sans tout relire

En-tête d'un bloc (2/2) : résumé et consensus

- Résumé du contenu
  - Champ de type `merkle_root` ou `state_root`
  - Hash qui résume toutes les opérations ou l'état global
  - Permet de vérifier le contenu sans tout relire
- Horodatage
  - Champ `timestamp`
  - Indique approximativement quand le bloc a été produit

En-tête d'un bloc (2/2) : résumé et consensus

- Résumé du contenu
  - Champ de type `merkle_root` ou `state_root`
  - Hash qui résume toutes les opérations ou l'état global
  - Permet de vérifier le contenu sans tout relire
- Horodatage
  - Champ `timestamp`
  - Indique approximativement quand le bloc a été produit
- Données de consensus
  - Informations dépendantes du protocole :
  - ex : preuve, signature du validateur, numéro de round, difficulté, nonce, ...
  - Utilisées pour décider si le bloc est valide au regard des règles du système

Corps d'un bloc : les opérations

- Liste d'éléments métier
  - opérations sur un registre partagé
  - messages entre participants
  - appels de contrats ou mises à jour d'état

Corps d'un bloc : les opérations

- Liste d'éléments métier
  - opérations sur un registre partagé
  - messages entre participants
  - appels de contrats ou mises à jour d'état
- Format général
  - chaque élément contient :
    - un type d'opération
    - des paramètres (données)
    - éventuellement une signature / preuve d'autorisation

Corps d'un bloc : les opérations

- Liste d'éléments métier
  - opérations sur un registre partagé
  - messages entre participants
  - appels de contrats ou mises à jour d'état
- Format général
  - chaque élément contient :
    - un type d'opération
    - des paramètres (données)
    - éventuellement une signature / preuve d'autorisation
- Regroupement
  - le bloc rassemble un lot cohérent d'opérations
  - qui seront appliquées à l'état global dans un ordre déterminé

Résumé : structure d'un bloc de blockchain

header	
`block_id`	Identifiant du bloc (souvent hash du header).
`previous_block_hash`	Lien vers le bloc précédent (chaînage).
`merkle_root` / `state_root`	Résumé des éléments du bloc / de l'état.
`timestamp`	Horodatage de création du bloc.
`consensus_data`	Données liées au consensus (nonce, signatures, ...).
body	
`éléments`	Liste d'éléments enregistrés (opérations, transactions, appels de contrats, ...).

- `header` = premiers champs (métadonnées + sécurité).
- `body` = liste d'éléments métier enregistrés dans le bloc.

Bloc, taille, état global et vérification

- Contraintes de taille
  - nombre d'opérations par bloc limité (taille max, quotas de ressources, ...)
  - permet de contrôler la charge réseau et de calcul

Bloc, taille, état global et vérification

- Contraintes de taille
  - nombre d'opérations par bloc limité (taille max, quotas de ressources, ...)
  - permet de contrôler la charge réseau et de calcul
- Lien avec l'état global
  - appliquer séquentiellement les opérations du bloc
  - met à jour l'état partagé (base de données logique de la blockchain)

Bloc, taille, état global et vérification

- Contraintes de taille
  - nombre d'opérations par bloc limité (taille max, quotas de ressources, ...)
  - permet de contrôler la charge réseau et de calcul
- Lien avec l'état global
  - appliquer séquentiellement les opérations du bloc
  - met à jour l'état partagé (base de données logique de la blockchain)
- Vérification par les nœuds
  - vérifier l'intégrité cryptographique (hash, racine, chaînage)
  - vérifier que chaque opération respecte les règles métier
  - si tout est correct, le bloc est ajouté localement à la chaîne

La blockchain, comment ça fonctionne ?

Arbres de Merklé

Arbre de Merkle : idée générale

- Structure d'arbre basée sur des hash
  - Feuilles = hash d'éléments de base (opérations, documents, états, ...)
  - Nœuds internes = hash de la concaténation de leurs enfants

Arbre de Merkle : idée générale

- Structure d'arbre basée sur des hash
  - Feuilles = hash d'éléments de base (opérations, documents, états, ...)
  - Nœuds internes = hash de la concaténation de leurs enfants
- Objectif principal
  - Résumer un grand ensemble de données dans un seul hash
  - Ce hash unique est la racine de Merkle (Merkle root)

Arbre de Merkle : idée générale

- Structure d'arbre basée sur des hash
  - Feuilles = hash d'éléments de base (opérations, documents, états, ...)
  - Nœuds internes = hash de la concaténation de leurs enfants
- Objectif principal
  - Résumer un grand ensemble de données dans un seul hash
  - Ce hash unique est la racine de Merkle (Merkle root)
- Pourquoi un arbre ?
  - Permet de retrouver et prouver des informations
  - en ne manipulant qu'une petite partie des hash

Structure d'un arbre de Merkle

- Feuilles
  - Chaque feuille contient $h_i = H(E_i)$
  - E_i = élément de base (opération, enregistrement, ...)

Structure d'un arbre de Merkle

- Feuilles
  - Chaque feuille contient $h_i = H(E_i)$
  - E_i = élément de base (opération, enregistrement, ...)
- Niveaux intermédiaires
  - Chaque nœud interne est le hash de ses deux enfants :

$$h_{parent} = H(h_{gauche} \| h_{droite})$$

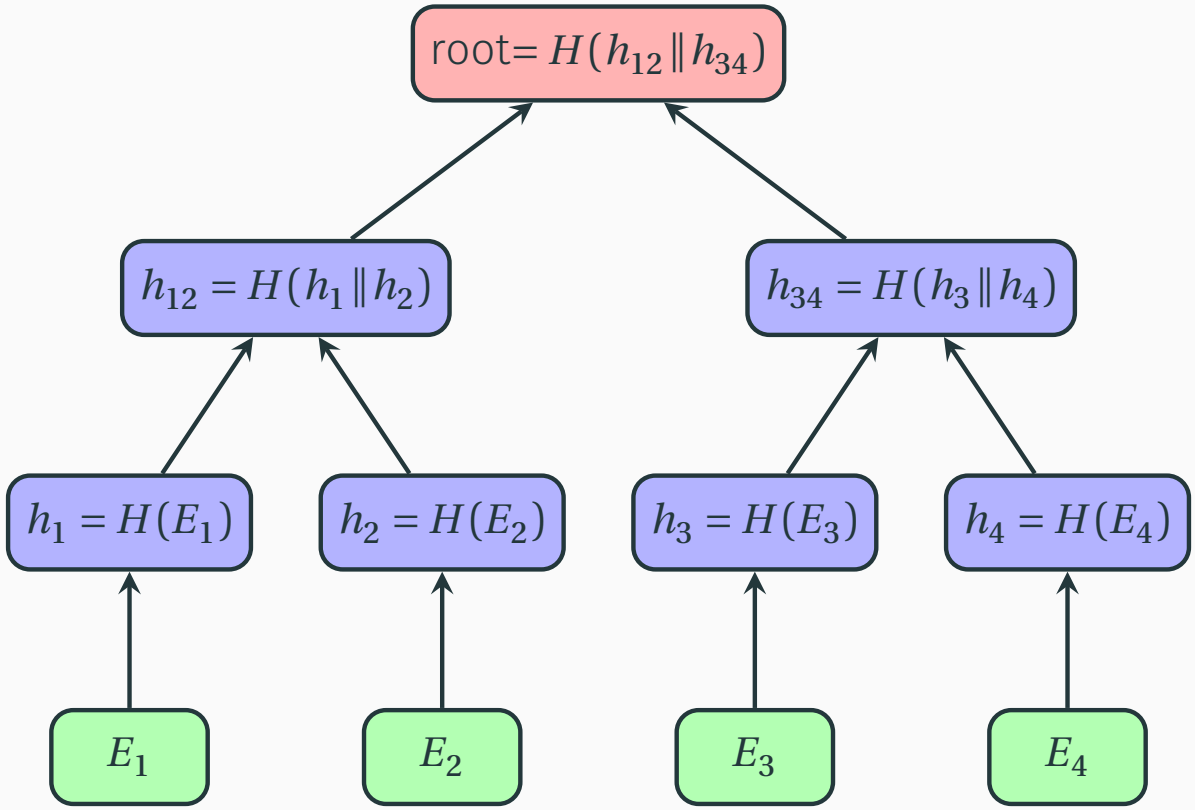
Structure d'un arbre de Merkle

- Feuilles
  - Chaque feuille contient $h_i = H(E_i)$
  - E_i = élément de base (opération, enregistrement, ...)
- Niveaux intermédiaires
  - Chaque nœud interne est le hash de ses deux enfants :

$$h_{parent} = H(h_{gauche} \| h_{droite})$$

- Racine (root)
  - Le dernier hash tout en haut de l'arbre
  - Représente l'ensemble complet des éléments
  - Une seule valeur pour vérifier un lot entier de données

Illustration : fonctionnement d'un arbre de Merkle



- ▷ Chaque élément E_i est d'abord hachée : feuilles h_i.
- ▷ Les hash sont ensuite combinés deux à deux jusqu'à obtenir la racine de Merkle.
- ▷ Cette racine (un seul hash) résume tout le contenu de l'arbre.

Construction d'un arbre de Merkle (pas à pas)

1. Calcul des feuilles
   - ▷ Pour chaque élément E_i :

$$h_i = H(E_i)$$

Construction d'un arbre de Merkle (pas à pas)

1. Calcul des feuilles
   - ▷ Pour chaque élément E_i :

$$h_i = H(E_i)$$

2. Regroupement par paires
   - ▷ On forme des paires : (h_1, h_2), (h_3, h_4), ...
   - ▷ Pour chaque paire :

$$h_{1,2} = H(h_1 \| h_2), \quad h_{3,4} = H(h_3 \| h_4)$$

   - ▷ Si le nombre de nœuds est impair, on peut dupliquer le dernier

Construction d'un arbre de Merkle (pas à pas)

1. Calcul des feuilles
   - ▷ Pour chaque élément E_i :

$$h_i = H(E_i)$$

2. Regroupement par paires
   - ▷ On forme des paires : (h_1, h_2), (h_3, h_4), ...
   - ▷ Pour chaque paire :

$$h_{1,2} = H(h_1 \| h_2), \quad h_{3,4} = H(h_3 \| h_4)$$

   - ▷ Si le nombre de nœuds est impair, on peut dupliquer le dernier
3. Répétition
   - ▷ On répète le même processus sur le niveau supérieur
   - ▷ jusqu'à ce qu'il ne reste plus qu'un hash : la Merkle root

Preuve de Merkle : prouver qu'un élément est présent

- Problème
  - Comment prouver que l'élément E_k appartient à l'ensemble
  - sans envoyer toutes les données ?

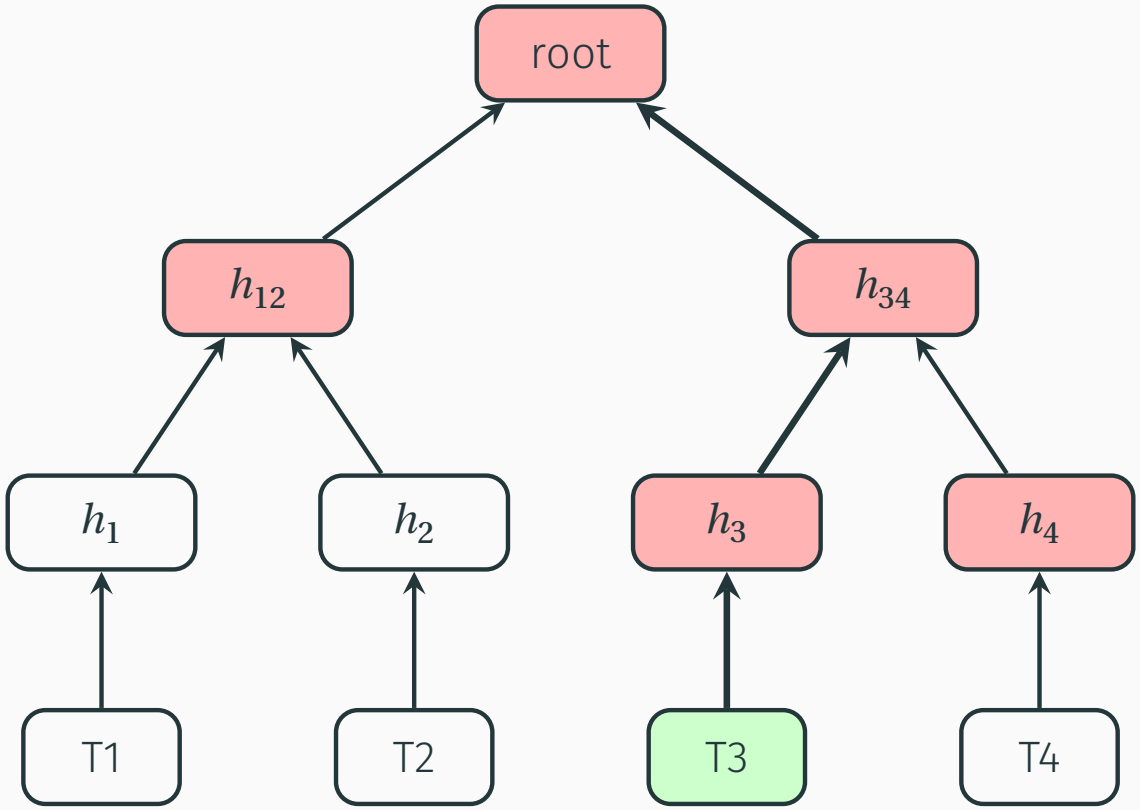
Preuve de Merkle : prouver qu'un élément est présent

- Problème
  - Comment prouver que l'élément E_k appartient à l'ensemble
  - sans envoyer toutes les données ?
- Idée de la preuve de Merkle
  - On envoie :
    - E_k lui-même
    - les hash nécessaires pour remonter jusqu'à la racine (frères sur le chemin)
  - Le vérificateur reconstruit la racine à partir de ces valeurs
  - Il compare avec la racine attendue (par exemple stockée dans le bloc)

Preuve de Merkle : prouver qu'un élément est présent

- Problème
  - Comment prouver que l'élément E_k appartient à l'ensemble
  - sans envoyer toutes les données ?
- Idée de la preuve de Merkle
  - On envoie :
    - E_k lui-même
    - les hash nécessaires pour remonter jusqu'à la racine (frères sur le chemin)
  - Le vérificateur reconstruit la racine à partir de ces valeurs
  - Il compare avec la racine attendue (par exemple stockée dans le bloc)
- Complexité
  - Longueur de la preuve $\approx O(\log n)$ pour n éléments
  - Très efficace même pour des ensembles de grande taille

Preuve de Merkle : illustrer l'inclusion d'une transaction



- ▷ On veut prouver que l'élément E3 appartient au bloc.
- ▷ Le client reçoit : E3, les hash h_4 et h_{12} (chemin de preuve).
- ▷ Il recalcule $h_3 = H(E3)$, puis $h_{34} = H(h_3 \| h_4)$, puis la racine.
- ▷ Si la racine obtenue est égale à celle stockée dans le bloc, E3 est bien inclus.

Arbres de Merkle en blockchain : à quoi ça sert ?

- Résumé compact du contenu d'un bloc
  - Le bloc stocke uniquement la racine dans son en-tête
  - Toute modification d'un élément change la racine

Arbres de Merkle en blockchain : à quoi ça sert ?

- Résumé compact du contenu d'un bloc
  - Le bloc stocke uniquement la racine dans son en-tête
  - Toute modification d'un élément change la racine
- Support des clients légers
  - Les nœuds légers téléchargent seulement les en-têtes de blocs
  - Ils utilisent des preuves de Merkle pour vérifier certaines opérations
  - Sans stocker toute la blockchain

Arbres de Merkle en blockchain : à quoi ça sert ?

- Résumé compact du contenu d'un bloc
  - Le bloc stocke uniquement la racine dans son en-tête
  - Toute modification d'un élément change la racine
- Support des clients légers
  - Les nœuds légers téléchargent seulement les en-têtes de blocs
  - Ils utilisent des preuves de Merkle pour vérifier certaines opérations
  - Sans stocker toute la blockchain
- Intégrité et auditabilité
  - Une altération locale (un seul enregistrement) se propage jusqu'à la racine
  - En comparant la racine attendue et la racine calculée, on détecte toute falsification

La blockchain, comment ça fonctionne ?

Réseau pair-à-pair

Réseau pair-à-pair : idée générale

- Pas de serveur central
  - La blockchain fonctionne sur un réseau pair-à-pair (P2P).
  - Chaque nœud peut parler directement avec plusieurs autres nœuds.

Réseau pair-à-pair : idée générale

- Pas de serveur central
  - La blockchain fonctionne sur un réseau pair-à-pair (P2P).
  - Chaque nœud peut parler directement avec plusieurs autres nœuds.
- Tous les nœuds jouent (presque) le même rôle
  - Chaque nœud peut recevoir, vérifier et relayer des informations.
  - Le réseau se configure et s'auto-organise sans point unique de contrôle.

Réseau pair-à-pair : idée générale

- Pas de serveur central
  - La blockchain fonctionne sur un réseau pair-à-pair (P2P).
  - Chaque nœud peut parler directement avec plusieurs autres nœuds.
- Tous les nœuds jouent (presque) le même rôle
  - Chaque nœud peut recevoir, vérifier et relayer des informations.
  - Le réseau se configure et s'auto-organise sans point unique de contrôle.
- Objectif
  - Diffuser rapidement les opérations et les blocs à tous les participants.
  - Permettre à chacun de maintenir une vue locale à jour de la blockchain.

Types de nœuds et connexions

- Nœud complet (full node)
  - Stocke tout ou partie substantielle de la blockchain.
  - Valide les blocs et les opérations selon les règles du protocole.

Types de nœuds et connexions

- Nœud complet (full node)
  - Stocke tout ou partie substantielle de la blockchain.
  - Valide les blocs et les opérations selon les règles du protocole.
- Nœud léger / client léger
  - Ne stocke que les en-têtes ou un sous-ensemble des données.
  - S'appuie sur des preuves (Merkle, etc.) fournies par des nœuds complets.

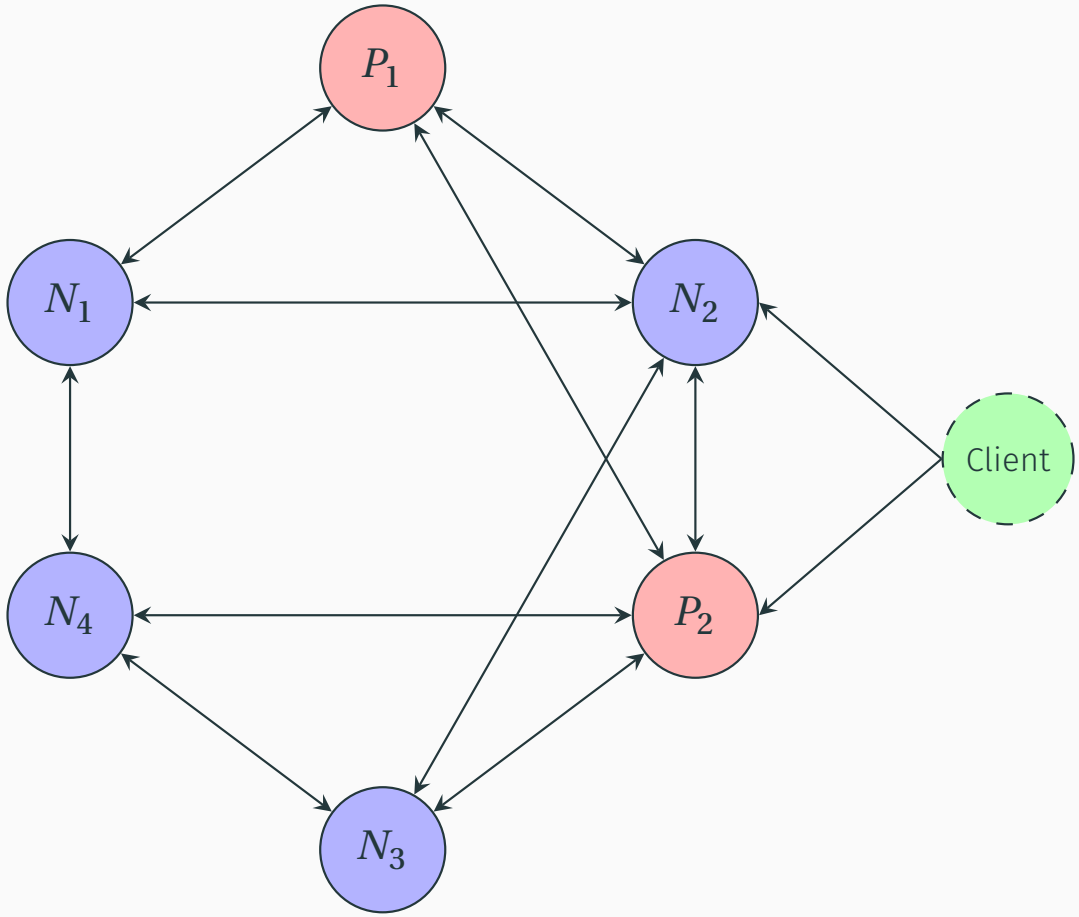
Types de nœuds et connexions

- Nœud complet (full node)
  - Stocke tout ou partie substantielle de la blockchain.
  - Valide les blocs et les opérations selon les règles du protocole.
- Nœud léger / client léger
  - Ne stocke que les en-têtes ou un sous-ensemble des données.
  - S'appuie sur des preuves (Merkle, etc.) fournies par des nœuds complets.
- Producteurs de blocs
  - Miners / validateurs / leaders selon le consensus.
  - Sélectionnent des opérations et proposent de nouveaux blocs au réseau.

Types de nœuds et connexions

- Nœud complet (full node)
  - Stocke tout ou partie substantielle de la blockchain.
  - Valide les blocs et les opérations selon les règles du protocole.
- Nœud léger / client léger
  - Ne stocke que les en-têtes ou un sous-ensemble des données.
  - S'appuie sur des preuves (Merkle, etc.) fournies par des nœuds complets.
- Producteurs de blocs
  - Miners / validateurs / leaders selon le consensus.
  - Sélectionnent des opérations et proposent de nouveaux blocs au réseau.
- Connexions
  - Chaque nœud maintient plusieurs connexions TCP/UDP avec d'autres nœuds.
  - Topologie généralement « maillée » : pas de centre, mais un graphe distribué.

Illustration : réseau pair-à-pair et producteurs de blocs



- ▷ N_1 à N_4 : nœuds complets qui stockent et valident la blockchain.
- ▷ P_1 et P_2 : producteurs de blocs (mineurs ou validateurs) mis en évidence.
- ▷ Le réseau P2P relaye les blocs et transactions entre tous les nœuds.
- ▷ Le client léger interroge quelques nœuds pour connaître l'état de la chaîne.

Propagation des opérations et des blocs

- Diffusion par gossip
  - Lorsqu'un nœud reçoit une nouvelle opération ou un nouveau bloc :
  - il le vérifie rapidement, puis le relaye à ses pairs (sauf doublons).

Propagation des opérations et des blocs

- Diffusion par gossip
  - Lorsqu'un nœud reçoit une nouvelle opération ou un nouveau bloc :
  - il le vérifie rapidement, puis le relaye à ses pairs (sauf doublons).
- Propagation des opérations
  - Un client envoie son opération signée à quelques nœuds.
  - En quelques étapes de relais, l'opération est connue d'une grande partie du réseau.

Propagation des opérations et des blocs

- Diffusion par gossip
  - Lorsqu'un nœud reçoit une nouvelle opération ou un nouveau bloc :
  - il le vérifie rapidement, puis le relaye à ses pairs (sauf doublons).
- Propagation des opérations
  - Un client envoie son opération signée à quelques nœuds.
  - En quelques étapes de relais, l'opération est connue d'une grande partie du réseau.
- Propagation des blocs
  - Quand un bloc est proposé par un producteur :
  - il est transmis aux pairs, qui le vérifient (header, signatures, opérations).
  - S'il est valide, ils l'ajoutent à leur chaîne locale et continuent à le relayer.

- Résilience
  - Pas de point de panne unique : si certains nœuds tombent, les autres continuent.
  - Le réseau peut se reconfigurer dynamiquement (nouveaux nœuds, départs, etc.).

Propriétés et défis du réseau P2P

- Résilience
  - Pas de point de panne unique : si certains nœuds tombent, les autres continuent.
  - Le réseau peut se reconfigurer dynamiquement (nouveaux nœuds, départs, etc.).
- Latence et cohérence
  - La diffusion P2P prend un certain temps.
  - Pendant la propagation, tous les nœuds n'ont pas encore la même vue de la chaîne (risque de forks temporaires).

Propriétés et défis du réseau P2P

- Résilience
  - Pas de point de panne unique : si certains nœuds tombent, les autres continuent.
  - Le réseau peut se reconfigurer dynamiquement (nouveaux nœuds, départs, etc.).
- Latence et cohérence
  - La diffusion P2P prend un certain temps.
  - Pendant la propagation, tous les nœuds n'ont pas encore la même vue de la chaîne (risque de forks temporaires).
- Sécurité du réseau
  - Faire face aux attaques Sybil (création massive de faux nœuds).
  - Détecter / ignorer les nœuds qui relaient des blocs ou des opérations invalides.

Propriétés et défis du réseau P2P

- Résilience
  - Pas de point de panne unique : si certains nœuds tombent, les autres continuent.
  - Le réseau peut se reconfigurer dynamiquement (nouveaux nœuds, départs, etc.).
- Latence et cohérence
  - La diffusion P2P prend un certain temps.
  - Pendant la propagation, tous les nœuds n'ont pas encore la même vue de la chaîne (risque de forks temporaires).
- Sécurité du réseau
  - Faire face aux attaques Sybil (création massive de faux nœuds).
  - Détecter / ignorer les nœuds qui relaient des blocs ou des opérations invalides.
- Impact sur la conception de la blockchain
  - Le protocole doit rester simple et robuste à travers un réseau P2P imparfait (latence variable, nœuds non fiables, partitions temporaires, ...).

La blockchain, comment ça fonctionne ?

Preuves

Proof of Work (PoW) : idée de base

- Objectif
  - Prouver qu'un nœud a réalisé une certaine quantité de calcul.
  - Le droit de proposer un bloc revient à celui qui trouve une solution à un puzzle.

Proof of Work (PoW) : idée de base

- Objectif
  - Prouver qu'un nœud a réalisé une certaine quantité de calcul.
  - Le droit de proposer un bloc revient à celui qui trouve une solution à un puzzle.
- Intuition
  - Le réseau impose un puzzle : trouver un nonce tel que

$$H(\text{header du bloc} \| \text{nonce}) < \text{target}$$

  - La seule manière réaliste de réussir est d'essayer beaucoup de nonces (brute force).

Proof of Work (PoW) : idée de base

- Objectif
  - Prouver qu'un nœud a réalisé une certaine quantité de calcul.
  - Le droit de proposer un bloc revient à celui qui trouve une solution à un puzzle.
- Intuition
  - Le réseau impose un puzzle : trouver un nonce tel que

$$H(\text{header du bloc} \| \text{nonce}) < \text{target}$$

  - La seule manière réaliste de réussir est d'essayer beaucoup de nonces (brute force).
- Idée clé
  - « Je prouve que j'ai travaillé » en montrant une solution difficile à trouver, mais facile à vérifier par tous les nœuds.

Proof of Work (PoW) : fonctionnement et impacts

- Fonctionnement
  - Les mineurs construisent un bloc candidat (liste d'opérations).
  - Ils essaient différents nonces jusqu'à ce que le hash respecte la condition de difficulté.
  - Le premier qui trouve diffuse son bloc ; les autres vérifient le hash en une seule opération.

Proof of Work (PoW) : fonctionnement et impacts

- Fonctionnement
  - Les mineurs construisent un bloc candidat (liste d'opérations).
  - Ils essaient différents nonces jusqu'à ce que le hash respecte la condition de difficulté.
  - Le premier qui trouve diffuse son bloc ; les autres vérifient le hash en une seule opération.
- Avantages
  - Modèle simple à comprendre, très étudié (Bitcoin, etc.).
  - Sécurité liée au coût énergétique : attaquer est très cher.

Proof of Work (PoW) : fonctionnement et impacts

- Fonctionnement
  - Les mineurs construisent un bloc candidat (liste d'opérations).
  - Ils essaient différents nonces jusqu'à ce que le hash respecte la condition de difficulté.
  - Le premier qui trouve diffuse son bloc ; les autres vérifient le hash en une seule opération.
- Avantages
  - Modèle simple à comprendre, très étudié (Bitcoin, etc.).
  - Sécurité liée au coût énergétique : attaquer est très cher.
- Inconvénients
  - Forte consommation d'énergie.
  - Débit de blocs limité, finalité seulement probabiliste (on attend plusieurs blocs).
  - Tendance à la centralisation chez les gros mineurs / pools.

Calcul de la Proof of Work dans Bitcoin

- Entrée de la PoW : le header de bloc (80 octets)
  - version, `previous_block_hash`, `merkle_root`
  - `timestamp`, `bits` (difficulté encodée), `nonce`

Calcul de la Proof of Work dans Bitcoin

- Entrée de la PoW : le header de bloc (80 octets)
  - version, `previous_block_hash`, `merkle_root`
  - `timestamp`, `bits` (difficulté encodée), `nonce`
- Fonction de hachage utilisée
  - Bitcoin calcule un double SHA-256 :

$$h = \text{SHA256}(\text{SHA256}(\text{header}))$$

Calcul de la Proof of Work dans Bitcoin

- Entrée de la PoW : le header de bloc (80 octets)
  - version, `previous_block_hash`, `merkle_root`
  - `timestamp`, `bits` (difficulté encodée), `nonce`
- Fonction de hachage utilisée
  - Bitcoin calcule un double SHA-256 :

$$h = \text{SHA256}(\text{SHA256}(\text{header}))$$

- Condition de validité
  - Le hash h (vu comme un entier sur 256 bits) doit être inférieur à une cible T :

$$h < T$$

  - En pratique : h doit commencer par un grand nombre de bits zéros.

Calcul de la Proof of Work dans Bitcoin

- Entrée de la PoW : le header de bloc (80 octets)
  - version, `previous_block_hash`, `merkle_root`
  - `timestamp`, `bits` (difficulté encodée), `nonce`
- Fonction de hachage utilisée
  - Bitcoin calcule un double SHA-256 :

$$h = \text{SHA256}(\text{SHA256}(\text{header}))$$

- Condition de validité
  - Le hash h (vu comme un entier sur 256 bits) doit être inférieur à une cible T :

$$h < T$$

  - En pratique : h doit commencer par un grand nombre de bits zéros.
- Rôle de la difficulté
  - Le champ `bits` encode T (plus la difficulté est haute, plus T est petit).
  - Tous les 2016 blocs, le protocole ajuste T pour maintenir $\sim$ 10 minutes par bloc.

Calcul de la Proof of Work dans Bitcoin

- Entrée de la PoW : le header de bloc (80 octets)
  - version, `previous_block_hash`, `merkle_root`
  - `timestamp`, `bits` (difficulté encodée), `nonce`
- Fonction de hachage utilisée
  - Bitcoin calcule un double SHA-256 :

$$h = \text{SHA256}(\text{SHA256}(\text{header}))$$

- Condition de validité
  - Le hash h (vu comme un entier sur 256 bits) doit être inférieur à une cible T :

$$h < T$$

  - En pratique : h doit commencer par un grand nombre de bits zéros.
- Rôle de la difficulté
  - Le champ `bits` encode T (plus la difficulté est haute, plus T est petit).
  - Tous les 2016 blocs, le protocole ajuste T pour maintenir $\sim$ 10 minutes par bloc.
- Processus côté mineur
  - Fixe le header (sauf `nonce` et parfois `timestamp`).
  - Boucle sur des valeurs de `nonce`, calcule h à chaque fois.
  - Quand un mineur trouve un $h < T$, il diffuse son bloc au réseau.

Exercice : probabilité de trouver un hash valide (Bitcoin)

Dans Bitcoin, on suppose que le résultat du double SHA-256 du header de bloc est un entier **aléatoire uniforme** sur 256 bits.

On peut approximer la probabilité pour qu'un hash soit valide par :

$$p \approx \frac{1}{2^{32} \times \text{difficulty}}$$

(formule classique pour Bitcoin, avec $2^{32} \approx 4{,}29 \times 10^9$).

Données (ordre de grandeur actuel) :

- ▷ Difficulté actuelle : difficulty $\approx 149 \times 10^{12}$.
- ▷ Hashrate d'un mineur : $H = 100$ TH/s $= 10^{14}$ hash/s.

Questions :

1. Calculez la probabilité p qu'un **hash unique** soit valide.
2. En déduire le **nombre moyen de hash** à essayer pour trouver un bloc.
3. Avec un hashrate de 10^{14} hash/s, estimez le **temps moyen** (en secondes puis en années) pour que ce mineur trouve un bloc.

Correction : probabilité de trouver un hash valide

Données :

$$\text{difficulty} \approx 149 \times 10^{12}, \quad 2^{32} \approx 4{,}29 \times 10^{9}, \quad H = 10^{14} \text{ hash/s}$$

1. Probabilité pour un hash d'être valide

$$p \approx \frac{1}{2^{32} \times \text{difficulty}} \approx \frac{1}{4{,}29 \times 10^{9} \times 1{,}49 \times 10^{14}} \approx \frac{1}{6{,}4 \times 10^{23}} \approx 1{,}56 \times 10^{-24}$$

2. Nombre moyen de hash nécessaires

$$\mathbb{E}[\text{hash}] = \frac{1}{p} \approx 6{,}4 \times 10^{23} \text{ hash}$$

3. Temps moyen avec $H = 10^{14}$ hash/s

$$T \approx \frac{6{,}4 \times 10^{23}}{10^{14}} = 6{,}4 \times 10^{9} \text{ s}$$

Conversion en années (avec 1 an $\approx 3{,}15 \times 10^{7}$ s) :

$$T \approx \frac{6{,}4 \times 10^{9}}{3{,}15 \times 10^{7}} \approx 203 \text{ ans}$$

Interprétation :

- ▷ Un seul mineur à 100 TH/s a une probabilité minuscule de trouver vite un bloc.
- ▷ D'où l'intérêt des pools de minage pour lisser la variance des revenus.

Proof of Stake (PoS) : idée de base

- Objectif
  - Remplacer la preuve de calcul par une preuve de mise (stake).
  - Le droit de proposer / valider un bloc dépend de la quantité de tokens immobilisés.

Proof of Stake (PoS) : idée de base

- Objectif
  - Remplacer la preuve de calcul par une preuve de mise (stake).
  - Le droit de proposer / valider un bloc dépend de la quantité de tokens immobilisés.
- Intuition
  - Plus un acteur met de valeur en jeu, plus il a de chances de participer au consensus.
  - S'il se comporte mal, une partie de son stake peut être confisquée (slashing).

Proof of Stake (PoS) : idée de base

- Objectif
  - Remplacer la preuve de calcul par une preuve de mise (stake).
  - Le droit de proposer / valider un bloc dépend de la quantité de tokens immobilisés.
- Intuition
  - Plus un acteur met de valeur en jeu, plus il a de chances de participer au consensus.
  - S'il se comporte mal, une partie de son stake peut être confisquée (slashing).
- Motivation principale
  - Obtenir une sécurité comparable à PoW
  - tout en réduisant fortement la consommation d'énergie.

- Fonctionnement (vue simplifiée)
  - Les validateurs verrouillent des tokens dans un smart contract ou un module de staking.
  - Un protocole pseudo-aléatoire sélectionne les validateurs qui proposent/vérifient les blocs.
  - Les blocs valides rapportent des récompenses ; les comportements malveillants entraînent un slashing.

Proof of Stake (PoS) : mécanismes et enjeux

- Fonctionnement (vue simplifiée)
  - Les validateurs verrouillent des tokens dans un smart contract ou un module de staking.
  - Un protocole pseudo-aléatoire sélectionne les validateurs qui proposent/vérifient les blocs.
  - Les blocs valides rapportent des récompenses ; les comportements malveillants entraînent un slashing.
- Avantages
  - Consommation de ressources physique très faible par rapport à PoW.
  - Permet des protocoles avec finalité plus rapide (BFT-like dans certains cas).

Proof of Stake (PoS) : mécanismes et enjeux

- Fonctionnement (vue simplifiée)
  - Les validateurs verrouillent des tokens dans un smart contract ou un module de staking.
  - Un protocole pseudo-aléatoire sélectionne les validateurs qui proposent/vérifient les blocs.
  - Les blocs valides rapportent des récompenses ; les comportements malveillants entraînent un slashing.
- Avantages
  - Consommation de ressources physique très faible par rapport à PoW.
  - Permet des protocoles avec finalité plus rapide (BFT-like dans certains cas).
- Inconvénients / défis
  - Risques de centralisation du stake si quelques acteurs contrôlent beaucoup de tokens.
  - Problèmes théoriques à adresser : « nothing-at-stake », attaques longues portées, etc.
  - Conception plus complexe que PoW, beaucoup de variantes (Ouroboros, Casper, Tendermint, ...).

Proof of Authority (PoA) : idée de base

- Objectif
  - Confier la production des blocs à un ensemble limité de validateurs connus.
  - Chaque bloc est signé par une autorité (ou un groupe) explicitement identifiée.

Proof of Authority (PoA) : idée de base

- Objectif
  - Confier la production des blocs à un ensemble limité de validateurs connus.
  - Chaque bloc est signé par une autorité (ou un groupe) explicitement identifiée.
- Contexte d'utilisation
  - Blockchains permissionnées ou de consortium.
  - Cas où les participants sont des organisations connues (entreprises, administrations, etc.).

Proof of Authority (PoA) : idée de base

- Objectif
  - Confier la production des blocs à un ensemble limité de validateurs connus.
  - Chaque bloc est signé par une autorité (ou un groupe) explicitement identifiée.
- Contexte d'utilisation
  - Blockchains permissionnées ou de consortium.
  - Cas où les participants sont des organisations connues (entreprises, administrations, etc.).
- Idée clé
  - On ne prouve pas du calcul ou de la mise, mais le fait d'être une entité autorisée par la gouvernance du réseau.

Proof of Authority (PoA) : fonctionnement et compromis

- Fonctionnement
  - Un ensemble de validateurs est défini (liste de clés publiques autorisées).
  - À tour de rôle, ou selon un protocole BFT, ils proposent et signent les blocs.
  - Les nœuds acceptent uniquement les blocs signés par ces autorités légitimes.

Proof of Authority (PoA) : fonctionnement et compromis

- Fonctionnement
  - Un ensemble de validateurs est défini (liste de clés publiques autorisées).
  - À tour de rôle, ou selon un protocole BFT, ils proposent et signent les blocs.
  - Les nœuds acceptent uniquement les blocs signés par ces autorités légitimes.
- Avantages
  - Très efficace : latence faible, haut débit possible.
  - Faible consommation de ressources.
  - Contrôle fin sur qui peut écrire dans la blockchain.

Proof of Authority (PoA) : fonctionnement et compromis

- Fonctionnement
  - Un ensemble de validateurs est défini (liste de clés publiques autorisées).
  - À tour de rôle, ou selon un protocole BFT, ils proposent et signent les blocs.
  - Les nœuds acceptent uniquement les blocs signés par ces autorités légitimes.
- Avantages
  - Très efficace : latence faible, haut débit possible.
  - Faible consommation de ressources.
  - Contrôle fin sur qui peut écrire dans la blockchain.
- Inconvénients
  - Décentralisation limitée : on « fait confiance » à un petit groupe d'acteurs.
  - Gouvernance hors chaîne nécessaire : qui choisit / révoque les autorités ? selon quelles règles ?
  - Moins adapté à des blockchains totalement publiques et ouvertes.

Autres types de proof (aperçu)

- Proof of Space / Capacity / Storage
  - On prouve qu'on réserve un certain espace disque pour le réseau.
  - Variante : Proof of Space-Time (espace occupé pendant une certaine durée).

Autres types de proof (aperçu)

- Proof of Space / Capacity / Storage
  - On prouve qu'on réserve un certain espace disque pour le réseau.
  - Variante : Proof of Space-Time (espace occupé pendant une certaine durée).
- Proof of History (PoH)
  - Utilise une suite de hash pour créer une sorte d'horloge cryptographique.
  - Sert souvent à optimiser l'ordre et la synchronisation des blocs (ex : combiné avec PoS).

Autres types de proof (aperçu)

- Proof of Space / Capacity / Storage
  - On prouve qu'on réserve un certain espace disque pour le réseau.
  - Variante : Proof of Space-Time (espace occupé pendant une certaine durée).
- Proof of History (PoH)
  - Utilise une suite de hash pour créer une sorte d'horloge cryptographique.
  - Sert souvent à optimiser l'ordre et la synchronisation des blocs (ex : combiné avec PoS).
- Proof of Elapsed Time (PoET)
  - Chaque nœud attend un temps aléatoire dans un environnement sécurisé (TEE).
  - Le plus rapide gagne le droit de proposer un bloc.

Autres types de proof (aperçu)

- Proof of Space / Capacity / Storage
  - On prouve qu'on réserve un certain espace disque pour le réseau.
  - Variante : Proof of Space-Time (espace occupé pendant une certaine durée).
- Proof of History (PoH)
  - Utilise une suite de hash pour créer une sorte d'horloge cryptographique.
  - Sert souvent à optimiser l'ordre et la synchronisation des blocs (ex : combiné avec PoS).
- Proof of Elapsed Time (PoET)
  - Chaque nœud attend un temps aléatoire dans un environnement sécurisé (TEE).
  - Le plus rapide gagne le droit de proposer un bloc.
- Autres variantes
  - Proof of Burn (brûler des tokens pour obtenir des droits).
  - Proof of Reputation / Importance (prendre en compte la réputation ou l'activité).
  - Hybrides PoW/PoS, PoS + BFT, etc.

Forks et blocs orphelins

- Qu'est-ce qu'un fork ?
  - Deux blocs différents sont produits à partir du même bloc parent.
  - Résultat : la chaîne se sépare temporairement en plusieurs branches.

Forks et blocs orphelins

- Qu'est-ce qu'un fork ?
  - Deux blocs différents sont produits à partir du même bloc parent.
  - Résultat : la chaîne se sépare temporairement en plusieurs branches.
- Pourquoi ça arrive ?
  - Latence réseau : deux producteurs de blocs trouvent / proposent un bloc presque en même temps.
  - Certains nœuds voient d'abord le bloc A, d'autres voient d'abord le bloc B.

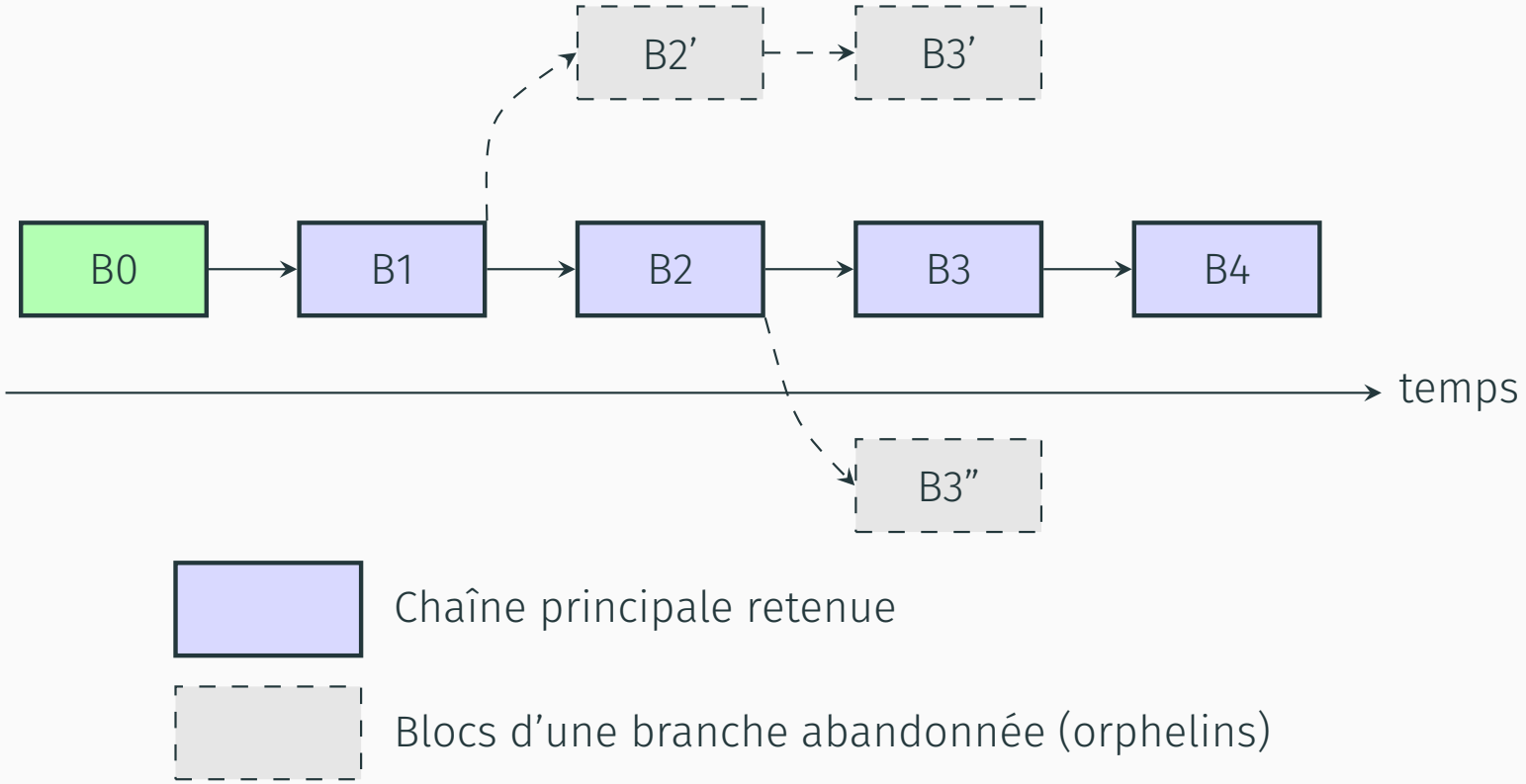
Forks et blocs orphelins

- Qu'est-ce qu'un fork ?
  - Deux blocs différents sont produits à partir du même bloc parent.
  - Résultat : la chaîne se sépare temporairement en plusieurs branches.
- Pourquoi ça arrive ?
  - Latence réseau : deux producteurs de blocs trouvent / proposent un bloc presque en même temps.
  - Certains nœuds voient d'abord le bloc A, d'autres voient d'abord le bloc B.
- Blocs orphelins (ou abandonnés)
  - Lorsque le réseau converge sur une branche, les blocs de l'autre branche deviennent orphelins.
  - Ils restent valides cryptographiquement, mais ne font plus partie de la chaîne principale.
  - Les opérations qu'ils contenaient peuvent, selon les règles, être remises en file d'attente.

Forks et blocs orphelins

- Qu'est-ce qu'un fork ?
  - Deux blocs différents sont produits à partir du même bloc parent.
  - Résultat : la chaîne se sépare temporairement en plusieurs branches.
- Pourquoi ça arrive ?
  - Latence réseau : deux producteurs de blocs trouvent / proposent un bloc presque en même temps.
  - Certains nœuds voient d'abord le bloc A, d'autres voient d'abord le bloc B.
- Blocs orphelins (ou abandonnés)
  - Lorsque le réseau converge sur une branche, les blocs de l'autre branche deviennent orphelins.
  - Ils restent valides cryptographiquement, mais ne font plus partie de la chaîne principale.
  - Les opérations qu'ils contenaient peuvent, selon les règles, être remises en file d'attente.
- Idée clef
  - Les forks sont normaux dans un réseau distribué.
  - Le protocole de consensus doit définir comment revenir à une seule histoire.

Illustration : blockchain avec forks et blocs abandonnés



- Plusieurs blocs peuvent être minés presque en même temps ⇒ **forks** temporaires.
- Le protocole choisit ensuite une **branche principale** (ici en bleu).
- Les blocs des autres branches deviennent **orphelins** : valides cryptographiquement, mais hors de la chaîne canonique.

Comment choisir la branche à garder ?

- Règle générale
  - Chaque protocole définit une règle de sélection de chaîne.
  - Objectif : que tous les nœuds honnêtes finissent par choisir la même branche.

Comment choisir la branche à garder ?

- Règle générale
  - Chaque protocole définit une règle de sélection de chaîne.
  - Objectif : que tous les nœuds honnêtes finissent par choisir la même branche.
- Exemple en Proof of Work
  - Règle du « plus de travail accumulé » (souvent simplifiée en « plus longue chaîne »).
  - Chaque bloc ajoute une certaine quantité de travail (difficulté).
  - On garde la chaîne avec la somme de travail la plus élevée.

Comment choisir la branche à garder ?

- Règle générale
  - Chaque protocole définit une règle de sélection de chaîne.
  - Objectif : que tous les nœuds honnêtes finissent par choisir la même branche.
- Exemple en Proof of Work
  - Règle du « plus de travail accumulé » (souvent simplifiée en « plus longue chaîne »).
  - Chaque bloc ajoute une certaine quantité de travail (difficulté).
  - On garde la chaîne avec la somme de travail la plus élevée.
- Exemple en PoS / BFT
  - Règle basée sur les votes / attestations des validateurs.
  - On suit la branche qui a reçu suffisamment de confirmations (quorum, finalité).

Comment choisir la branche à garder ?

- Règle générale
  - Chaque protocole définit une règle de sélection de chaîne.
  - Objectif : que tous les nœuds honnêtes finissent par choisir la même branche.
- Exemple en Proof of Work
  - Règle du « plus de travail accumulé » (souvent simplifiée en « plus longue chaîne »).
  - Chaque bloc ajoute une certaine quantité de travail (difficulté).
  - On garde la chaîne avec la somme de travail la plus élevée.
- Exemple en PoS / BFT
  - Règle basée sur les votes / attestations des validateurs.
  - On suit la branche qui a reçu suffisamment de confirmations (quorum, finalité).
- Comportement d'un nœud
  - Maintient éventuellement plusieurs branches en mémoire à court terme.
  - Si une autre branche devient préférée selon la règle, il bascule dessus.
  - L'ancienne branche devient une branche secondaire : ses blocs sont orphelins.

La blockchain, comment ça fonctionne ?

Éléments/Transactions

Cycle de vie d'un élément / transaction (avant le bloc)

1. Création côté client
   - ▷ L'utilisateur (ou une appli) prépare un élément :
     - ▷ transaction, appel de smart contract, message métier, ...
   - ▷ Il remplit les champs nécessaires : expéditeur, destinataire, montants, paramètres...
   - ▷ Il signe cet élément avec sa clé privée.

Cycle de vie d'un élément / transaction (avant le bloc)

1. Création côté client
   - ▷ L'utilisateur (ou une appli) prépare un élément :
     - ▷ transaction, appel de smart contract, message métier, ...
   - ▷ Il remplit les champs nécessaires : expéditeur, destinataire, montants, paramètres...
   - ▷ Il signe cet élément avec sa clé privée.
2. Envoi à un nœud du réseau
   - ▷ L'élément signé est envoyé à un ou plusieurs nœuds complets.
   - ▷ Ces nœuds font une validation locale rapide :
     - ▷ signature correcte ?
     - ▷ format valide ?
     - ▷ règles métier de base respectées ?
   - ▷ Si c'est invalide ⇒ rejet immédiat.

Cycle de vie d'un élément / transaction (avant le bloc)

1. Création côté client
   - ▷ L'utilisateur (ou une appli) prépare un élément :
     - ▷ transaction, appel de smart contract, message métier, ...
   - ▷ Il remplit les champs nécessaires : expéditeur, destinataire, montants, paramètres...
   - ▷ Il signe cet élément avec sa clé privée.
2. Envoi à un nœud du réseau
   - ▷ L'élément signé est envoyé à un ou plusieurs nœuds complets.
   - ▷ Ces nœuds font une validation locale rapide :
     - ▷ signature correcte ?
     - ▷ format valide ?
     - ▷ règles métier de base respectées ?
   - ▷ Si c'est invalide ⇒ rejet immédiat.
3. Mise en file d'attente (mempool)
   - ▷ Si c'est valide, l'élément est placé dans une file d'attente locale (souvent appelée mempool).
   - ▷ La mempool contient tous les éléments en attente d'inclusion dans un bloc.

Propagation dans le réseau : mempool partagée (en pratique)

4. Diffusion pair-à-pair
   - ▷ Le nœud qui reçoit un nouvel élément valide le relaye à ses pairs.
   - ▷ Chaque nœud répète le même schéma :
     - ▷ validation de base,
     - ▷ ajout à sa mempool,
     - ▷ relay aux autres (en évitant les doublons).

Propagation dans le réseau : mempool partagée (en pratique)

4. Diffusion pair-à-pair
   - ▷ Le nœud qui reçoit un nouvel élément valide le relaye à ses pairs.
   - ▷ Chaque nœud répète le même schéma :
     - ▷ validation de base,
     - ▷ ajout à sa mempool,
     - ▷ relay aux autres (en évitant les doublons).
5. Mempool = vue locale des opérations en attente
   - ▷ Chaque nœud a sa **propre** mempool :
     - ▷ contenu très proche entre nœuds honnêtes,
     - ▷ mais jamais strictement identique (latence, filtrage, limites, ...).
   - ▷ On peut voir la mempool comme un « **sas** » :
     - ▷ entre le monde extérieur (clients),
     - ▷ et la blockchain (blocs déjà confirmés).

Propagation dans le réseau : mempool partagée (en pratique)

4. Diffusion pair-à-pair
   - ▷ Le nœud qui reçoit un nouvel élément valide le relaye à ses pairs.
   - ▷ Chaque nœud répète le même schéma :
     - ▷ validation de base,
     - ▷ ajout à sa mempool,
     - ▷ relay aux autres (en évitant les doublons).
5. Mempool = vue locale des opérations en attente
   - ▷ Chaque nœud a sa **propre** mempool :
     - ▷ contenu très proche entre nœuds honnêtes,
     - ▷ mais jamais strictement identique (latence, filtrage, limites, ...).
   - ▷ On peut voir la mempool comme un « **sas** » :
     - ▷ entre le monde extérieur (clients),
     - ▷ et la blockchain (blocs déjà confirmés).
6. Priorisation
   - ▷ Les éléments peuvent être classés :
     - ▷ par frais / priorité,
     - ▷ par contraintes techniques (taille, gas, dépendances, ...).
   - ▷ Les producteurs de blocs choisiront les éléments dans cet ordre.

De la mempool au bloc : sélection des éléments

7. Rôle du producteur de bloc
   - ▷ Un mineur / validateur qui veut produire un bloc part de :
     - ▷ la mempool locale,
     - ▷ le dernier bloc jugé valide.
   - ▷ Il sélectionne un sous-ensemble d'éléments à inclure.

De la mempool au bloc : sélection des éléments

7. Rôle du producteur de bloc
   - Un mineur / validateur qui veut produire un bloc part de :
     - la mempool locale,
     - le dernier bloc jugé valide.
   - Il sélectionne un sous-ensemble d'éléments à inclure.
8. Critères de sélection typiques
   - Contraintes techniques :
     - taille maximale du bloc,
     - budget de ressources (ex : gas).
   - Intérêt économique :
     - frais les plus élevés en priorité (pour maximiser la récompense).
   - Validité métier :
     - vérifier que chaque élément reste valide dans l'ordre choisi (solde suffisant, nonce correct, pas de conflit, ...).

De la mempool au bloc : sélection des éléments

7. Rôle du producteur de bloc
   - Un mineur / validateur qui veut produire un bloc part de :
     - la mempool locale,
     - le dernier bloc jugé valide.
   - Il sélectionne un sous-ensemble d'éléments à inclure.
8. Critères de sélection typiques
   - Contraintes techniques :
     - taille maximale du bloc,
     - budget de ressources (ex : gas).
   - Intérêt économique :
     - frais les plus élevés en priorité (pour maximiser la récompense).
   - Validité métier :
     - vérifier que chaque élément reste valide dans l'ordre choisi (solde suffisant, nonce correct, pas de conflit, ...).
9. Après inclusion dans un bloc
   - Une fois le bloc accepté par le réseau :
     - les éléments inclus sont retirés de la mempool,
     - ils font désormais partie de l'historique immuable.
   - Les éléments restants restent en mempool en attendant un prochain bloc (ou finissent par expirer / être supprimés).

La blockchain, comment ça fonctionne ?

Smart contracts

Smart contract : c'est quoi ?

- Programme sur la blockchain
  - Un smart contract est un programme dont le code et les données sont stockés sur la blockchain.
  - Il est exécuté par les nœuds du réseau, pas sur une machine privée.

Smart contract : c'est quoi ?

- Programme sur la blockchain
  - Un smart contract est un programme dont le code et les données sont stockés sur la blockchain.
  - Il est exécuté par les nœuds du réseau, pas sur une machine privée.
- Règles automatiques
  - Il encode des règles métier : « si telle condition est vraie, alors exécuter telle action ».
  - L'exécution est automatique dès que les conditions sont réunies.

Smart contract : c'est quoi ?

- Programme sur la blockchain
  - Un smart contract est un programme dont le code et les données sont stockés sur la blockchain.
  - Il est exécuté par les nœuds du réseau, pas sur une machine privée.
- Règles automatiques
  - Il encode des règles métier : « si telle condition est vraie, alors exécuter telle action ».
  - L'exécution est automatique dès que les conditions sont réunies.
- Pas de confiance dans un opérateur humain
  - On ne fait pas confiance à une personne ou une entreprise.
  - On fait confiance au code du contrat et au protocole de la blockchain.

Smart contract : c'est quoi ?

- Programme sur la blockchain
  - Un smart contract est un programme dont le code et les données sont stockés sur la blockchain.
  - Il est exécuté par les nœuds du réseau, pas sur une machine privée.
- Règles automatiques
  - Il encode des règles métier : « si telle condition est vraie, alors exécuter telle action ».
  - L'exécution est automatique dès que les conditions sont réunies.
- Pas de confiance dans un opérateur humain
  - On ne fait pas confiance à une personne ou une entreprise.
  - On fait confiance au code du contrat et au protocole de la blockchain.
- Exemples de plateformes
  - Ethereum (EVM), EVM-compatibles, autres VM dédiées (Move, WASM, …)

1. Déploiement
   - ▷ Le développeur écrit le contrat (ex. en Solidity).
   - ▷ Le code est compilé puis envoyé dans une transaction spéciale.
   - ▷ Une fois inclus dans un bloc, le contrat reçoit une adresse sur la blockchain.

Structure & cycle de vie d'un smart contract

1. Déploiement
   - ▷ Le développeur écrit le contrat (ex. en Solidity).
   - ▷ Le code est compilé puis envoyé dans une transaction spéciale.
   - ▷ Une fois inclus dans un bloc, le contrat reçoit une adresse sur la blockchain.
2. Stockage & état
   - ▷ Le contrat possède des variables de stockage (état persistant).
   - ▷ Cet état est répliqué sur tous les nœuds qui exécutent la blockchain.

Structure & cycle de vie d'un smart contract

1. Déploiement
   - ▷ Le développeur écrit le contrat (ex. en Solidity).
   - ▷ Le code est compilé puis envoyé dans une transaction spéciale.
   - ▷ Une fois inclus dans un bloc, le contrat reçoit une adresse sur la blockchain.
2. Stockage & état
   - ▷ Le contrat possède des variables de stockage (état persistant).
   - ▷ Cet état est répliqué sur tous les nœuds qui exécutent la blockchain.
3. Appels de fonctions
   - ▷ Les utilisateurs envoient des transactions qui appellent des fonctions du contrat.
   - ▷ Le code s'exécute de manière déterministe sur l'état courant : lecture, mise à jour, émission d'événements, appels à d'autres contrats, ...

Structure & cycle de vie d'un smart contract

1. Déploiement
   - ▷ Le développeur écrit le contrat (ex. en Solidity).
   - ▷ Le code est compilé puis envoyé dans une transaction spéciale.
   - ▷ Une fois inclus dans un bloc, le contrat reçoit une adresse sur la blockchain.
2. Stockage & état
   - ▷ Le contrat possède des variables de stockage (état persistant).
   - ▷ Cet état est répliqué sur tous les nœuds qui exécutent la blockchain.
3. Appels de fonctions
   - ▷ Les utilisateurs envoient des transactions qui appellent des fonctions du contrat.
   - ▷ Le code s'exécute de manière déterministe sur l'état courant : lecture, mise à jour, émission d'événements, appels à d'autres contrats, ...
4. (Souvent) pas de mise à jour du code
   - ▷ Le code est en général immuable une fois déployé.
   - ▷ On met à jour la logique via de nouveaux contrats ou des mécanismes de proxy/upgrade.

À quoi servent les smart contracts ?

- Automatiser des accords
  - Exemple : séquestre (« escrow ») — libérer des fonds si une condition est remplie.
  - Exemple : abonnement — débiter périodiquement tant que le contrat n'est pas résilié.

À quoi servent les smart contracts ?

- Automatiser des accords
  - Exemple : séquestre (« escrow ») — libérer des fonds si une condition est remplie.
  - Exemple : abonnement — débiter périodiquement tant que le contrat n'est pas résilié.
- Construire des applications décentralisées (dApps)
  - Finance décentralisée (prêts, échanges, assurances on-chain).
  - Jeux, NFT, identités numériques, systèmes de vote, registres, etc.

À quoi servent les smart contracts ?

- Automatiser des accords
  - Exemple : séquestre (« escrow ») — libérer des fonds si une condition est remplie.
  - Exemple : abonnement — débiter périodiquement tant que le contrat n'est pas résilié.
- Construire des applications décentralisées (dApps)
  - Finance décentralisée (prêts, échanges, assurances on-chain).
  - Jeux, NFT, identités numériques, systèmes de vote, registres, etc.
- Coordonner plusieurs acteurs
  - Plusieurs organisations peuvent utiliser le même contrat comme source de vérité.
  - Le code agit comme un « arbitre » neutre : il applique les règles pour tout le monde.

À quoi servent les smart contracts ?

- Automatiser des accords
  - Exemple : séquestre (« escrow ») — libérer des fonds si une condition est remplie.
  - Exemple : abonnement — débiter périodiquement tant que le contrat n'est pas résilié.
- Construire des applications décentralisées (dApps)
  - Finance décentralisée (prêts, échanges, assurances on-chain).
  - Jeux, NFT, identités numériques, systèmes de vote, registres, etc.
- Coordonner plusieurs acteurs
  - Plusieurs organisations peuvent utiliser le même contrat comme source de vérité.
  - Le code agit comme un « arbitre » neutre : il applique les règles pour tout le monde.
- Traçabilité et audit
  - Toutes les interactions avec le contrat sont enregistrées sur la blockchain.
  - On peut auditer l'historique des appels et l'évolution de l'état.

Limites & risques des smart contracts

- Le code est la loi (pour de vrai)
  - Si le contrat a un bug, le réseau l'exécutera quand même.
  - Corriger un bug peut être très difficile (immutabilité, gouvernance).

Limites & risques des smart contracts

- Le code est la loi (pour de vrai)
  - Si le contrat a un bug, le réseau l'exécutera quand même.
  - Corriger un bug peut être très difficile (immutabilité, gouvernance).
- Surface d'attaque importante
  - Vulnérabilités classiques : reentrancy, mauvaises vérifications d'accès, erreurs arithmétiques, underflow/overflow (selon la plateforme), ...
  - Un bug peut entraîner la perte ou le blocage de ressources.

Limites & risques des smart contracts

- Le code est la loi (pour de vrai)
  - Si le contrat a un bug, le réseau l'exécutera quand même.
  - Corriger un bug peut être très difficile (immutabilité, gouvernance).
- Surface d'attaque importante
  - Vulnérabilités classiques : reentrancy, mauvaises vérifications d'accès, erreurs arithmétiques, underflow/overflow (selon la plateforme), ...
  - Un bug peut entraîner la perte ou le blocage de ressources.
- Contraintes techniques
  - Coût d'exécution (gas, frais) : on ne peut pas faire n'importe quel calcul.
  - Ressources limitées : temps, mémoire, taille du code.

Limites & risques des smart contracts

- Le code est la loi (pour de vrai)
  - Si le contrat a un bug, le réseau l'exécutera quand même.
  - Corriger un bug peut être très difficile (immutabilité, gouvernance).
- Surface d'attaque importante
  - Vulnérabilités classiques : reentrancy, mauvaises vérifications d'accès, erreurs arithmétiques, underflow/overflow (selon la plateforme), ...
  - Un bug peut entraîner la perte ou le blocage de ressources.
- Contraintes techniques
  - Coût d'exécution (gas, frais) : on ne peut pas faire n'importe quel calcul.
  - Ressources limitées : temps, mémoire, taille du code.
- Enjeux hors chaîne
  - Problèmes juridiques : quel statut légal pour un smart contract ?
  - Dépendance à des oracles pour connaître des faits du monde réel (prix, événements externes, etc.).

Vulnérabilité de *reentrancy*

- Principe
  - Un contrat appelle du code externe (autre contrat).
  - Ce code externe rappelle la même fonction (ou une fonction sensible) avant la fin de la première exécution.
  - L'état interne n'est pas encore mis à jour ⇒ l'attaquant peut répéter l'action plusieurs fois (ex : retirer plusieurs fois des fonds).

Vulnérabilité de *reentrancy*

- Principe
  - Un contrat appelle du code externe (autre contrat).
  - Ce code externe rappelle la même fonction (ou une fonction sensible) avant la fin de la première exécution.
  - L'état interne n'est pas encore mis à jour ⇒ l'attaquant peut répéter l'action plusieurs fois (ex : retirer plusieurs fois des fonds).
- Exemple de pattern vulnérable (pseudo-Solidity)
  - Ordre dangereux : interaction externe → mise à jour de l'état.

```
function withdraw() public {
    uint amount = balances[msg.sender];

    // 1) Appel externe : peut exécuter du code malicieux
    (bool ok, ) = msg.sender.call{value: amount}("");
    require(ok, "send failed");

    // 2) Mise à jour trop tard
    balances[msg.sender] = 0;
}
```

Vulnérabilité de *reentrancy*

- Principe
  - Un contrat appelle du code externe (autre contrat).
  - Ce code externe rappelle la même fonction (ou une fonction sensible) avant la fin de la première exécution.
  - L'état interne n'est pas encore mis à jour ⇒ l'attaquant peut répéter l'action plusieurs fois (ex : retirer plusieurs fois des fonds).
- Exemple de pattern vulnérable (pseudo-Solidity)
  - Ordre dangereux : interaction externe → mise à jour de l'état.

```
function withdraw() public {
    uint amount = balances[msg.sender];

    // 1) Appel externe : peut exécuter du code malicieux
    (bool ok, ) = msg.sender.call{value: amount}("");
    require(ok, "send failed");

    // 2) Mise à jour trop tard
    balances[msg.sender] = 0;
}
```

- Contre-mesures
  - Pattern Checks–Effects–Interactions : d'abord vérifier, ensuite mettre à jour l'état, enfin appeler l'extérieur.
  - Utiliser un reentrancy guard (verrou booléen).
  - Minimiser les appels vers des contrats tiers (ou les encapsuler).

Overflow / underflow arithmétique

- Principe
  - Les entiers ont des bornes : ex. `uint8` $\in [0, 255]$.
  - Overflow : dépasser la borne haute (255 + 1 → 0).
  - Underflow : passer sous la borne basse (0 - 1 → 255).

Overflow / underflow arithmétique

- Principe
  - Les entiers ont des bornes : ex. `uint8` $\in [0, 255]$.
  - Overflow : dépasser la borne haute (255 + 1 → 0).
  - Underflow : passer sous la borne basse (0 - 1 → 255).
- Avant Solidity 0.8 (comportement silencieux)

```
uint8 x = 255;
x = x + 1;   // overflow -> x devient 0

uint8 y = 0;
y = y - 1;   // underflow -> y devient 255
```

  - Pas d'erreur automatique : les valeurs « roulent ».
  - Peut casser les contrôles de soldes, quotas, compteurs...

Overflow / underflow arithmétique

- Principe
  - Les entiers ont des bornes : ex. `uint8` $\in [0, 255]$.
  - Overflow : dépasser la borne haute (255 + 1 → 0).
  - Underflow : passer sous la borne basse (0 - 1 → 255).
- Avant Solidity 0.8 (comportement silencieux)

```
uint8 x = 255;
x = x + 1;   // overflow -> x devient 0

uint8 y = 0;
y = y - 1;   // underflow -> y devient 255
```

  - Pas d'erreur automatique : les valeurs « roulent ».
  - Peut casser les contrôles de soldes, quotas, compteurs...
- Risques dans un smart contract
  - Création / destruction involontaire de tokens.
  - Possibilité de retirer plus que son solde effectif.
  - Bypass de limites ou de conditions métier.

Overflow / underflow arithmétique

- Principe
  - Les entiers ont des bornes : ex. `uint8` $\in [0, 255]$.
  - Overflow : dépasser la borne haute (255 + 1 → 0).
  - Underflow : passer sous la borne basse (0 - 1 → 255).
- Avant Solidity 0.8 (comportement silencieux)

```
uint8 x = 255;
x = x + 1;   // overflow -> x devient 0

uint8 y = 0;
y = y - 1;   // underflow -> y devient 255
```

  - Pas d'erreur automatique : les valeurs « roulent ».
  - Peut casser les contrôles de soldes, quotas, compteurs...
- Risques dans un smart contract
  - Création / destruction involontaire de tokens.
  - Possibilité de retirer plus que son solde effectif.
  - Bypass de limites ou de conditions métier.
- Contre-mesures
  - Solidity ≥ 0.8 : overflow/underflow déclenchent un revert par défaut.
  - Avant / cas spécifiques : utiliser `SafeMath` ou des checks explicites, n'utiliser `unchecked` que de manière très contrôlée.

Oracles : connecter la blockchain au monde réel

- Problème de départ
  - La blockchain ne voit que ses données internes : blocs, transactions, état.
  - Elle ne connaît pas les données externes : prix, météo, résultats sportifs, événements juridiques, etc.

Oracles : connecter la blockchain au monde réel

- Problème de départ
  - La blockchain ne voit que ses données internes : blocs, transactions, état.
  - Elle ne connaît pas les données externes : prix, météo, résultats sportifs, événements juridiques, etc.
- Rôle d'un oracle
  - Composant (souvent un ensemble de nœuds) qui :
    - lit une source externe (API, capteur, base de données, ...),
    - écrit la donnée correspondante on-chain,
    - via un smart contract d'oracle.
  - Les smart contracts consomment ensuite cette information comme une donnée classique.

Oracles : connecter la blockchain au monde réel

- Problème de départ
  - La blockchain ne voit que ses données internes : blocs, transactions, état.
  - Elle ne connaît pas les données externes : prix, météo, résultats sportifs, événements juridiques, etc.
- Rôle d'un oracle
  - Composant (souvent un ensemble de nœuds) qui :
    - lit une source externe (API, capteur, base de données, ...),
    - écrit la donnée correspondante on-chain,
    - via un smart contract d'oracle.
  - Les smart contracts consomment ensuite cette information comme une donnée classique.
- Types d'oracles
  - Centralisé : une seule source de données (simple, mais point unique de confiance).
  - Décentralisé : plusieurs nœuds d'oracle, agrégation (médiane, votes, ...).

Oracles : connecter la blockchain au monde réel

- Problème de départ
  - La blockchain ne voit que ses données internes : blocs, transactions, état.
  - Elle ne connaît pas les données externes : prix, météo, résultats sportifs, événements juridiques, etc.
- Rôle d'un oracle
  - Composant (souvent un ensemble de nœuds) qui :
    - lit une source externe (API, capteur, base de données, ...),
    - écrit la donnée correspondante on-chain,
    - via un smart contract d'oracle.
  - Les smart contracts consomment ensuite cette information comme une donnée classique.
- Types d'oracles
  - Centralisé : une seule source de données (simple, mais point unique de confiance).
  - Décentralisé : plusieurs nœuds d'oracle, agrégation (médiane, votes, ...).
- « Oracle problem »
  - La blockchain reste sûre, mais l'oracle devient le nouveau maillon faible.
  - On réintroduit une forme de tiers de confiance pour tout ce qui vient du monde réel.

Exemple simple de smart contract : cagnotte

- ▷ Idée : une cagnotte partagée où plusieurs personnes peuvent déposer, et où seul l'organisateur peut vider la cagnotte.
- ▷ À retenir : logique métier claire (qui peut faire quoi), état persistant (`total`, `contributions`), vérifications d'accès.

Pseudo-code (style Solidity simplifié) :

```
contract Cagnotte {
    address owner;              // créateur de la cagnotte
    uint    total;              // montant total collecté

    mapping(address => uint) contributions;

    constructor() {
        owner = msg.sender;     // l'adresse qui déploie le contrat
        total = 0;
    }

    function contribute() public payable {
        contributions[msg.sender] += msg.value;
        total += msg.value;
    }

    function withdraw() public {
        require(msg.sender == owner);  // seul l'owner peut vider la cagnotte
        uint amount = total;
        total = 0;
        payable(owner).transfer(amount);
    }

    function balance() public view returns (uint) {
        return total;
    }
}
```

La blockchain, comment ça fonctionne ?

Crypto-monnaie

- Bloc + registre = monnaie native
  - La blockchain ne stocke pas seulement des opérations métier, mais l'état des soldes d'une monnaie numérique.
  - Chaque transaction modifie ce registre : débite une adresse, crédite une autre.

- Bloc + registre = monnaie native
  - La blockchain ne stocke pas seulement des opérations métier, mais l'état des soldes d'une monnaie numérique.
  - Chaque transaction modifie ce registre : débite une adresse, crédite une autre.
- Actif natif au cœur du protocole
  - La crypto-monnaie (BTC, ETH, ...) est intégrée au protocole : création, transfert, destruction sont définis dans le code.
  - Pas besoin de banque centrale ou d'émetteur unique : les règles monétaires sont codées dans la blockchain elle-même.

- Bloc + registre = monnaie native
  - La blockchain ne stocke pas seulement des opérations métier, mais l'état des soldes d'une monnaie numérique.
  - Chaque transaction modifie ce registre : débite une adresse, crédite une autre.
- Actif natif au cœur du protocole
  - La crypto-monnaie (BTC, ETH, ...) est intégrée au protocole : création, transfert, destruction sont définis dans le code.
  - Pas besoin de banque centrale ou d'émetteur unique : les règles monétaires sont codées dans la blockchain elle-même.
- Blockchain publique, ouverte
  - N'importe qui peut (en principe) :
    - créer une adresse (clé publique/privée),
    - envoyer/recevoir des unités de la monnaie,
    - exécuter un nœud et vérifier la chaîne.
  - Forte emphase sur la résistance à la censure.

- Récompenses de bloc
  - Les producteurs de blocs (mineurs/validateurs) reçoivent :
    - une création monétaire (block reward),
    - des frais de transaction payés par les utilisateurs.
  - Ces récompenses sont versées dans la monnaie native de la blockchain.

Incentives économiques au cœur du consensus

- Récompenses de bloc
  - Les producteurs de blocs (mineurs/validateurs) reçoivent :
    - une création monétaire (block reward),
    - des frais de transaction payés par les utilisateurs.
  - Ces récompenses sont versées dans la monnaie native de la blockchain.
- Sécurité financée par la monnaie
  - Plus la monnaie a de valeur, plus :
    - il est rentable de participer honnêtement au consensus,
    - il est coûteux d'attaquer le réseau (acheter du hash power ou du stake).
  - La cryptomonnaie sert donc à aligner les incentives des participants sur la sécurité du système.

Incentives économiques au cœur du consensus

- Récompenses de bloc
  - Les producteurs de blocs (mineurs/validateurs) reçoivent :
    - une création monétaire (block reward),
    - des frais de transaction payés par les utilisateurs.
  - Ces récompenses sont versées dans la monnaie native de la blockchain.
- Sécurité financée par la monnaie
  - Plus la monnaie a de valeur, plus :
    - il est rentable de participer honnêtement au consensus,
    - il est coûteux d'attaquer le réseau (acheter du hash power ou du stake).
  - La cryptomonnaie sert donc à aligner les incentives des participants sur la sécurité du système.
- Dimension économique incontournable
  - Contrairement à une blockchain purement métier interne, une crypto-monnaie implique :
    - un marché (échanges, prix, volatilité),
    - des effets spéculatifs et des comportements d'investisseurs.

- Monnaie programmable et globale
  - Transferts potentiellement sans intermédiaire et transfrontaliers.
  - Possibilité d'écrire des logiques monétaires complexes (verrouillage, multisig, smart contracts, ...).

Spécificités et enjeux des crypto-monnaies

- Monnaie programmable et globale
  - Transferts potentiellement sans intermédiaire et transfrontaliers.
  - Possibilité d'écrire des logiques monétaires complexes (verrouillage, multisig, smart contracts, ...).
- Volatilité et spéculation
  - Le prix de la monnaie varie en fonction de l'offre, de la demande, de la spéculation et du contexte réglementaire.
  - Mélange entre outil technique (paiement, réserve de valeur) et actif spéculatif (trading, investissement).

Spécificités et enjeux des crypto-monnaies

- Monnaie programmable et globale
  - Transferts potentiellement sans intermédiaire et transfrontaliers.
  - Possibilité d'écrire des logiques monétaires complexes (verrouillage, multisig, smart contracts, ...).
- Volatilité et spéculation
  - Le prix de la monnaie varie en fonction de l'offre, de la demande, de la spéculation et du contexte réglementaire.
  - Mélange entre outil technique (paiement, réserve de valeur) et actif spéculatif (trading, investissement).
- Dimension réglementaire et sociale
  - Questions de : blanchiment, financement illicite, fiscalité, protection des utilisateurs.
  - Tension entre :
    - la promesse de décentralisation,
    - et la nécessité de réguler certains usages.

Spécificités et enjeux des crypto-monnaies

- Monnaie programmable et globale
  - Transferts potentiellement sans intermédiaire et transfrontaliers.
  - Possibilité d'écrire des logiques monétaires complexes (verrouillage, multisig, smart contracts, ...).
- Volatilité et spéculation
  - Le prix de la monnaie varie en fonction de l'offre, de la demande, de la spéculation et du contexte réglementaire.
  - Mélange entre outil technique (paiement, réserve de valeur) et actif spéculatif (trading, investissement).
- Dimension réglementaire et sociale
  - Questions de : blanchiment, financement illicite, fiscalité, protection des utilisateurs.
  - Tension entre :
    - la promesse de décentralisation,
    - et la nécessité de réguler certains usages.
- En résumé
  - Une crypto-monnaie est une blockchain où le registre principal est un système monétaire, avec tout ce que cela implique en termes de technique, d'économie et de société.

Crypto-monnaie native vs token de smart contract

- Crypto-monnaie native (L1)
  - Exemple : BTC sur Bitcoin, ETH sur Ethereum, ADA sur Cardano.
  - Intégrée dans le protocole de base de la blockchain.
  - Création, transfert, destruction gérés par les règles du consensus et du protocole, pas par un contrat.
  - Conclusion : une crypto-monnaie native n'est pas un smart contract.

Crypto-monnaie native vs token de smart contract

- Crypto-monnaie native (L1)
  - Exemple : BTC sur Bitcoin, ETH sur Ethereum, ADA sur Cardano.
  - Intégrée dans le protocole de base de la blockchain.
  - Création, transfert, destruction gérés par les règles du consensus et du protocole, pas par un contrat.
  - Conclusion : une crypto-monnaie native n'est pas un smart contract.
- Token défini par un smart contract
  - Exemple : tokens ERC-20, ERC-721, stablecoins, jetons de gouvernance, etc.
  - Un smart contract maintient un mapping `adresse → solde`.
  - Les fonctions `transfer()`, `approve()`, etc. sont du code de contrat.
  - Conclusion : ici, la "crypto-monnaie" (au sens large) est implémentée par un smart contract.

Crypto-monnaie native vs token de smart contract

- Crypto-monnaie native (L1)
  - Exemple : BTC sur Bitcoin, ETH sur Ethereum, ADA sur Cardano.
  - Intégrée dans le protocole de base de la blockchain.
  - Création, transfert, destruction gérés par les règles du consensus et du protocole, pas par un contrat.
  - Conclusion : une crypto-monnaie native n'est pas un smart contract.
- Token défini par un smart contract
  - Exemple : tokens ERC-20, ERC-721, stablecoins, jetons de gouvernance, etc.
  - Un smart contract maintient un mapping `adresse → solde`.
  - Les fonctions `transfer()`, `approve()`, etc. sont du code de contrat.
  - Conclusion : ici, la "crypto-monnaie" (au sens large) est implémentée par un smart contract.
- En résumé
  - Toutes les crypto-monnaies ne sont pas des smart contracts.
  - Les monnaies natives appartiennent au protocole, les tokens secondaires appartiennent à des smart contracts.

Limites et critiques de la blockchain

Limites techniques de la blockchain

- Scalabilité limitée
  - Débit de transactions souvent faible comparé aux systèmes centralisés.
  - Latence de confirmation élevée (secondes à minutes).

Limites techniques de la blockchain

- Scalabilité limitée
  - Débit de transactions souvent faible comparé aux systèmes centralisés.
  - Latence de confirmation élevée (secondes à minutes).
- Coût et complexité
  - Stockage redondant : chaque nœud répète une grande partie des données.
  - Exécution du même code sur de nombreux nœuds → inefficience par design.
  - Développement complexe (crypto, sécurité, modèles distribués).

Limites techniques de la blockchain

- Scalabilité limitée
  - Débit de transactions souvent faible comparé aux systèmes centralisés.
  - Latence de confirmation élevée (secondes à minutes).
- Coût et complexité
  - Stockage redondant : chaque nœud répète une grande partie des données.
  - Exécution du même code sur de nombreux nœuds → inefficience par design.
  - Développement complexe (crypto, sécurité, modèles distribués).
- Difficulté d'évolution
  - Protocole + contrats souvent **difficiles à modifier** (consensus, gouvernance).
  - Gestion des bugs et mises à jour : hard forks, mécanismes d'upgrade compliqués.

Limites techniques de la blockchain

- Scalabilité limitée
  - Débit de transactions souvent faible comparé aux systèmes centralisés.
  - Latence de confirmation élevée (secondes à minutes).
- Coût et complexité
  - Stockage redondant : chaque nœud répète une grande partie des données.
  - Exécution du même code sur de nombreux nœuds → inefficience par design.
  - Développement complexe (crypto, sécurité, modèles distribués).
- Difficulté d'évolution
  - Protocole + contrats souvent **difficiles à modifier** (consensus, gouvernance).
  - Gestion des bugs et mises à jour : hard forks, mécanismes d'upgrade compliqués.
- Problèmes de confidentialité
  - Historique quasi permanent et transparent.
  - Protéger la vie privée nécessite des couches crypto supplémentaires (zero-knowledge, mixers, etc.), encore plus difficiles à manier.

Le Scalability Trilemma des blockchains

- L'idée de base
  - Une blockchain cherche à optimiser 3 propriétés :

Le Scalability Trilemma des blockchains

- L'idée de base
  - Une blockchain cherche à optimiser 3 propriétés :
  - Scalabilité : traiter beaucoup d'opérations par seconde.

Le Scalability Trilemma des blockchains

- L'idée de base
  - Une blockchain cherche à optimiser 3 propriétés :
  - Scalabilité : traiter beaucoup d'opérations par seconde.
  - Sécurité : résister aux attaques, manipulations, pannes.

Le Scalability Trilemma des blockchains

- L'idée de base
  - Une blockchain cherche à optimiser 3 propriétés :
  - Scalabilité : traiter beaucoup d'opérations par seconde.
  - Sécurité : résister aux attaques, manipulations, pannes.
  - Décentralisation : grand nombre de nœuds, sans acteur dominant.

Le Scalability Trilemma des blockchains

- L'idée de base
  - Une blockchain cherche à optimiser 3 propriétés :
  - Scalabilité : traiter beaucoup d'opérations par seconde.
  - Sécurité : résister aux attaques, manipulations, pannes.
  - Décentralisation : grand nombre de nœuds, sans acteur dominant.
- Le trilemme
  - Il est difficile d'optimiser les 3 en même temps.
  - En pratique, la plupart des designs doivent faire des compromis :
  - « 2 forts sur 3 » est souvent plus réaliste que 3/3.

Le Scalability Trilemma des blockchains

- L'idée de base
  - Une blockchain cherche à optimiser 3 propriétés :
  - Scalabilité : traiter beaucoup d'opérations par seconde.
  - Sécurité : résister aux attaques, manipulations, pannes.
  - Décentralisation : grand nombre de nœuds, sans acteur dominant.
- Le trilemme
  - Il est difficile d'optimiser les 3 en même temps.
  - En pratique, la plupart des designs doivent faire des compromis :
  - « 2 forts sur 3 » est souvent plus réaliste que 3/3.
- **Exemples de compromis (schéma mental)**
  - Scalabilité + Sécurité ⇒ souvent moins de décentralisation (ex : peu de validateurs puissants).

Le Scalability Trilemma des blockchains

- L'idée de base
  - Une blockchain cherche à optimiser 3 propriétés :
  - Scalabilité : traiter beaucoup d'opérations par seconde.
  - Sécurité : résister aux attaques, manipulations, pannes.
  - Décentralisation : grand nombre de nœuds, sans acteur dominant.
- Le trilemme
  - Il est difficile d'optimiser les 3 en même temps.
  - En pratique, la plupart des designs doivent faire des compromis :
  - « 2 forts sur 3 » est souvent plus réaliste que 3/3.
- **Exemples de compromis (schéma mental)**
  - Scalabilité + Sécurité ⇒ souvent moins de décentralisation (ex : peu de validateurs puissants).
  - Scalabilité + Décentralisation ⇒ sécurité plus difficile à garantir (propagation lente, plus de surface d'attaque).

Le Scalability Trilemma des blockchains

- L'idée de base
  - Une blockchain cherche à optimiser 3 propriétés :
  - Scalabilité : traiter beaucoup d'opérations par seconde.
  - Sécurité : résister aux attaques, manipulations, pannes.
  - Décentralisation : grand nombre de nœuds, sans acteur dominant.
- Le trilemme
  - Il est difficile d'optimiser les 3 en même temps.
  - En pratique, la plupart des designs doivent faire des compromis :
  - « 2 forts sur 3 » est souvent plus réaliste que 3/3.
- **Exemples de compromis (schéma mental)**
  - Scalabilité + Sécurité ⇒ souvent moins de décentralisation (ex : peu de validateurs puissants).
  - Scalabilité + Décentralisation ⇒ sécurité plus difficile à garantir (propagation lente, plus de surface d'attaque).
  - Sécurité + Décentralisation ⇒ scalabilité limitée (blocs petits, rythme lent pour que tout le monde suive).

Le Scalability Trilemma des blockchains

- L'idée de base
  - Une blockchain cherche à optimiser 3 propriétés :
  - Scalabilité : traiter beaucoup d'opérations par seconde.
  - Sécurité : résister aux attaques, manipulations, pannes.
  - Décentralisation : grand nombre de nœuds, sans acteur dominant.
- Le trilemme
  - Il est difficile d'optimiser les 3 en même temps.
  - En pratique, la plupart des designs doivent faire des compromis :
  - « 2 forts sur 3 » est souvent plus réaliste que 3/3.
- **Exemples de compromis (schéma mental)**
  - Scalabilité + Sécurité ⇒ souvent moins de décentralisation (ex : peu de validateurs puissants).
  - Scalabilité + Décentralisation ⇒ sécurité plus difficile à garantir (propagation lente, plus de surface d'attaque).
  - Sécurité + Décentralisation ⇒ scalabilité limitée (blocs petits, rythme lent pour que tout le monde suive).
- Conséquence pour le design
  - Chaque projet doit **assumer** ses choix : protocoles L1 + solutions L2, sharding, sidechains, etc.
  - Question à se poser : « où place-t-on le curseur entre ces 3 axes ? »

Critiques économiques, sociales et environnementales

- Consommation de ressources
  - PoW : critique récurrente sur la consommation énergétique et l'empreinte carbone.
  - Même sans PoW : duplication des calculs et stockage chez de nombreux nœuds.

Critiques économiques, sociales et environnementales

- Consommation de ressources
  - PoW : critique récurrente sur la consommation énergétique et l'empreinte carbone.
  - Même sans PoW : duplication des calculs et stockage chez de nombreux nœuds.
- Centralisation de fait
  - Concentration du pouvoir : gros mineurs/pools, gros stakers, grandes plateformes.
  - Beaucoup d'utilisateurs passent par des acteurs centraux (exchanges, wallets custodial), ce qui va à l'encontre du discours « sans intermédiaires ».

Critiques économiques, sociales et environnementales

- Consommation de ressources
  - PoW : critique récurrente sur la consommation énergétique et l'empreinte carbone.
  - Même sans PoW : duplication des calculs et stockage chez de nombreux nœuds.
- Centralisation de fait
  - Concentration du pouvoir : gros mineurs/pools, gros stakers, grandes plateformes.
  - Beaucoup d'utilisateurs passent par des acteurs centraux (exchanges, wallets custodial), ce qui va à l'encontre du discours « sans intermédiaires ».
- Gouvernance peu claire
  - Décisions clés (forks, upgrades, paramètres) prises par un mélange flou de devs, fondations, gros acteurs économiques.
  - Difficulté à articuler la promesse d'automatisation avec la réalité de processus politiques humains.

Critiques économiques, sociales et environnementales

- Consommation de ressources
  - PoW : critique récurrente sur la consommation énergétique et l'empreinte carbone.
  - Même sans PoW : duplication des calculs et stockage chez de nombreux nœuds.
- Centralisation de fait
  - Concentration du pouvoir : gros mineurs/pools, gros stakers, grandes plateformes.
  - Beaucoup d'utilisateurs passent par des acteurs centraux (exchanges, wallets custodial), ce qui va à l'encontre du discours « sans intermédiaires ».
- Gouvernance peu claire
  - Décisions clés (forks, upgrades, paramètres) prises par un mélange flou de devs, fondations, gros acteurs économiques.
  - Difficulté à articuler la promesse d'automatisation avec la réalité de processus politiques humains.
- Usages discutables
  - Spéculation massive, bulles, scams, rug pulls.
  - Écart entre la promesse de « révolution » et la réalité de nombreux projets à faible utilité réelle.

Pourquoi ce n'est pas toujours ce qu'on nous vend

- « Sans tiers de confiance »… mais en pratique
  - On remplace souvent des institutions identifiées par :
    - des développeurs de protocole,
    - des opérateurs d'infrastructures,
    - des gros détenteurs de tokens.
  - La « confiance » se déplace plutôt qu'elle ne disparaît.

Pourquoi ce n'est pas toujours ce qu'on nous vend

- « Sans tiers de confiance »… mais en pratique
  - On remplace souvent des institutions identifiées par :
    - des développeurs de protocole,
    - des opérateurs d'infrastructures,
    - des gros détenteurs de tokens.
  - La « confiance » se déplace plutôt qu'elle ne disparaît.
- « Solution à tout »
  - Beaucoup de problèmes sont mieux résolus par :
    - une base de données classique,
    - un système distribué mais centralisé sur un acteur identifié,
    - un accord juridique plutôt qu'un smart contract.
  - Rajouter de la blockchain peut ajouter de la complexité sans gain réel.

Pourquoi ce n'est pas toujours ce qu'on nous vend

- « Sans tiers de confiance »… mais en pratique
  - On remplace souvent des institutions identifiées par :
    - des développeurs de protocole,
    - des opérateurs d'infrastructures,
    - des gros détenteurs de tokens.
  - La « confiance » se déplace plutôt qu'elle ne disparaît.
- « Solution à tout »
  - Beaucoup de problèmes sont mieux résolus par :
    - une base de données classique,
    - un système distribué mais centralisé sur un acteur identifié,
    - un accord juridique plutôt qu'un smart contract.
  - Rajouter de la blockchain peut ajouter de la complexité sans gain réel.
- Bilan
  - La blockchain est un outil intéressant pour certains cas d'usage : registres partagés, finance programmable, coordination sans acteur dominant.
  - Mais ce n'est ni magique, ni neutre, ni gratuit : coûts techniques, limites, enjeux de gouvernance doivent être assumés.
  - Question clé à se poser : « Ai-je vraiment besoin d'une blockchain ici ? »

Conclusion

Conclusion du cours

- Ce que vous devez retenir
  - La blockchain = structure de données (chaîne de blocs + hash) + réseau P2P + consensus.
  - Rôle clé des hachages, signatures, arbres de Merkle, PoW/PoS/PoA, smart contracts.
  - Immutabilité, intégrité, vérifiabilité ne sont pas magiques : elles reposent sur des hypothèses techniques et économiques.

Conclusion du cours

- Ce que vous devez retenir
  - La blockchain = structure de données (chaîne de blocs + hash) + réseau P2P + consensus.
  - Rôle clé des hachages, signatures, arbres de Merkle, PoW/PoS/PoA, smart contracts.
  - Immutabilité, intégrité, vérifiabilité ne sont pas magiques : elles reposent sur des hypothèses techniques et économiques.
- Forces et limites
  - Outil puissant pour des registres partagés, la finance programmable, la coordination sans acteur unique.
  - Mais aussi : scalabilité limitée, complexité, coûts, enjeux de gouvernance, problèmes de vie privée.
  - Toujours se demander : « Ai-je vraiment besoin d'une blockchain ici ? »

Conclusion du cours

- Ce que vous devez retenir
  - La blockchain = structure de données (chaîne de blocs + hash) + réseau P2P + consensus.
  - Rôle clé des hachages, signatures, arbres de Merkle, PoW/PoS/PoA, smart contracts.
  - Immutabilité, intégrité, vérifiabilité ne sont pas magiques : elles reposent sur des hypothèses techniques et économiques.
- Forces et limites
  - Outil puissant pour des registres partagés, la finance programmable, la coordination sans acteur unique.
  - Mais aussi : scalabilité limitée, complexité, coûts, enjeux de gouvernance, problèmes de vie privée.
  - Toujours se demander : « Ai-je vraiment besoin d'une blockchain ici ? »
- Pour aller plus loin
  - Lire des whitepapers (Bitcoin, Ethereum, protocoles PoS/BFT).
  - Expérimenter : implémenter une mini-blockchain, écrire/auditer un smart contract.
  - Garder un regard critique : distinguer les véritables innovations des effets de mode.