Chapitre 1 : Introduction et vue d'ensemble

> Ce chapitre présente Bitcoin Core, le logiciel de référence du réseau Bitcoin, et offre une vision d'ensemble de son architecture avant d'entrer dans les détails techniques.
>
> Fil conducteur : d'abord, comprendre ce qu'est un nœud complet (`bitcoind`) et ce qu'il garantit. Ensuite, voir comment le code de Bitcoin Core est organisé. Enfin, suivre le trajet d'une transaction, de sa création jusqu'à ses confirmations.

1.1 Qu'est-ce que Bitcoin Core ?

Constat observable

Bitcoin Core est un logiciel libre.

Il permet de se connecter au réseau Bitcoin, de valider intégralement les transactions et les blocs, et de relayer des informations entre les participants du réseau.

Il constitue l'implémentation de référence du protocole Bitcoin.

Dans Bitcoin, une **transaction** est un message qui propose de déplacer des bitcoins.

Un **bloc** est un lot de transactions validées, ajouté à la **chaîne de blocs** (blockchain), l'historique public du système.

Le programme principal, appelé `bitcoind` (pour "Bitcoin daemon", c'est-à-dire "démon Bitcoin" — un programme qui s'exécute en arrière-plan), remplit le rôle de **nœud complet**.

Il télécharge et vérifie la chaîne de blocs depuis 2009, puis maintient une copie locale de l'état du réseau.[1]

Il existe deux manières courantes de configurer le stockage d'un nœud complet.

Dans une configuration dite « archive », `bitcoind` conserve l'ensemble des blocs sur disque : c'est une bibliothèque qui garde tous les volumes depuis 2009.

C'est utile pour analyser l'historique et pour aider d'autres nœuds à se synchroniser.

En mode **pruning** (élagage), `bitcoind` vérifie les blocs exactement de la même manière, puis supprime progressivement les plus anciens pour économiser de l'espace disque.

Il conserve tout de même l'essentiel pour fonctionner au quotidien : l'état courant (quels fonds sont dépensables, et sous quelles conditions).

La contrepartie est qu'il ne peut plus servir l'historique complet à d'autres nœuds.

Mécanisme

Quand `bitcoind` démarre, il effectue les opérations suivantes :

1. **Initialisation** : chargement de la configuration, vérification de l'environnement
2. **Connexion au réseau** : découverte d'autres nœuds et établissement de connexions **pair-à-pair** (peer-to-peer, P2P). En pratique, un nœud parle directement à plusieurs voisins, sans serveur central.
3. **Synchronisation** : téléchargement et validation des blocs manquants
4. **Fonctionnement normal** : réception, validation et relais des nouvelles transactions et blocs

Le nœud maintient plusieurs structures de données essentielles :

- La **chaîne de blocs** (blockchain) : l'historique complet et ordonné de tous les blocs, depuis l'origine. Chaque bloc référence le précédent par son hash, ce qui relie l'historique. Imaginez un grand registre public qui s'épaissit à chaque nouveau bloc.
- L'**ensemble des sorties non dépensées** (UTXO set) : la liste de tous les **UTXO** ("Unspent Transaction Outputs" — sorties non dépensées) disponibles. C'est l'état courant : quels fonds sont dépensables, et quelles preuves seront nécessaires pour les dépenser.
- La **mempool** ("memory pool") : la « salle d'attente » locale des transactions valides reçues mais pas encore confirmées. Elle peut légèrement différer d'un nœud à l'autre.

Conséquence directe

Un nœud complet **ne fait confiance à personne**. Il vérifie lui-même chaque signature (preuve cryptographique d'autorisation), chaque montant, chaque règle. C'est cette propriété qui permet à Bitcoin de fonctionner sans autorité centrale.

Ce que cela **garantit** : toute transaction ou bloc invalide sera rejeté, quelle que soit sa source.

Ce que cela **ne garantit pas** : que les transactions non confirmées seront effectivement incluses dans un bloc futur (cela dépend des mineurs, qui tentent de produire les blocs).

> **À retenir**
>
> Bitcoin Core est le logiciel qui fait fonctionner le réseau Bitcoin. En tant que nœud complet, il vérifie indépendamment toutes les règles du protocole sans faire confiance à aucun tiers.

> **Références**
>
> [1] Point d'entrée du démon : src/bitcoind.cpp, lignes 260-290 — fonction `MAIN_FUNCTION` qui initialise le `NodeContext` et lance l'application.

1.2 Architecture générale du code source

Constat observable

Le code source de Bitcoin Core est organisé en modules distincts, chacun responsable d'une fonction précise. Cette séparation permet de comprendre le système par parties et facilite la maintenance du logiciel.

Le répertoire principal `src/` contient plus de 180 fichiers et une vingtaine de sous-répertoires.[1]

Mécanisme

Les principaux modules sont :

Répertoire	Rôle
`crypto/`	Fonctions cryptographiques (hachage, signatures)
`consensus/`	Règles de consensus (ce qui rend un bloc valide)
`primitives/`	Structures de base (transactions, blocs)
`script/`	Interpréteur du langage Script, qui exprime les règles pour dépenser des fonds
`wallet/`	Gestion des clés et des fonds de l'utilisateur
`node/`	Logique du nœud (gestion de la chaîne, mempool)
`net/`	Communication réseau pair-à-pair (peer-to-peer, P2P)
`rpc/`	Interface de commande à distance (Remote Procedure Call, RPC)
`policy/`	Règles locales de relais (*policy*), souvent plus strictes que le consensus
`secp256k1/`	Bibliothèque de cryptographie sur courbe elliptique

Le cœur du nœud est représenté par une structure appelée `NodeContext`.

C'est un objet C++ interne qui regroupe les références vers les composants principaux (gestionnaire de chaîne `ChainstateManager`, mempool `CTxMemPool`, gestionnaire de connexions `CConnman`, etc.).[2]

Conséquence directe

Cette architecture modulaire signifie que :

- Les règles de **consensus** (dans `consensus/`) sont séparées des règles de **politique** (dans `policy/`)
- La **cryptographie** est isolée et peut être auditée indépendamment
- Le **portefeuille** est optionnel — un nœud peut fonctionner sans

On peut voir le **consensus** comme une loi commune : si un bloc la viole, tous les nœuds le rejetteront.

La **politique** (*policy*) ressemble plutôt à un règlement intérieur : un nœud peut être plus strict sur ce qu'il accepte en mempool et ce qu'il relaye, sans changer ce qui est valide dans un bloc.

Ce que cela **garantit** : une modification dans le portefeuille ne peut pas affecter les règles de validation des blocs.

Ce que cela **ne garantit pas** : que le code est exempt de bugs (d'où l'importance des audits et des tests).

À retenir

Le code de Bitcoin Core est organisé en modules : cryptographie, consensus, réseau, portefeuille. Cette séparation permet d'auditer chaque partie indépendamment et limite les risques d'erreur.

Références

[1] Structure du répertoire source : src/

[2] Structure `NodeContext` regroupant tous les composants : src/node/context.h, lignes 56-99

1.3 Le cycle de vie d'une transaction

Constat observable

Une transaction Bitcoin passe par plusieurs étapes distinctes avant d'être considérée comme définitivement enregistrée.

Quand vous « envoyez » un paiement, la transaction ne part pas vers une banque : elle circule de nœud en nœud, jusqu'à être incluse dans un bloc.

Une **transaction** est un message qui décrit un transfert de fonds ; un **bloc** est un paquet de transactions ajouté à la chaîne de blocs.

Comprendre ce cycle de vie est essentiel pour saisir le fonctionnement global du système.

Mécanisme

Étape 1 : Création

L'utilisateur (via son portefeuille) construit une transaction qui :

- Référence des fonds qu'il possède (des UTXO, "Unspent Transaction Outputs" — sorties de transactions non dépensées)
- Spécifie les destinataires et les montants
- Inclut une **signature cryptographique** — une sorte de sceau numérique mathématiquement inviolable qui prouve qu'il a le droit de dépenser ces fonds[1]

Étape 2 : Propagation

La transaction est envoyée aux nœuds connectés, qui la vérifient et la relaient à leurs propres voisins.

C'est un bouche-à-oreille technique : chaque nœud répète l'information à d'autres.

En quelques secondes, la transaction atteint une grande partie du réseau.[2]

Chaque nœud qui reçoit la transaction :

- Vérifie sa validité (signatures, montants, format)
- La stocke dans sa mempool si elle est valide
- La transmet à ses pairs

Étape 3 : Inclusion dans un bloc

Un **mineur** est un participant qui tente de produire le prochain bloc.

Il sélectionne des transactions dans sa mempool, puis construit un **bloc candidat** : un bloc « en brouillon » qui respecte déjà les règles de validité.

Ensuite, il cherche une **preuve de travail** (proof-of-work, PoW) : une énigme mathématique coûteuse à résoudre, mais facile à vérifier.

Ce processus de recherche s'appelle le **minage**.

On peut l'imaginer comme une loterie où l'on essaie de trouver un « ticket gagnant ».

Quand il y parvient, il diffuse le bloc au réseau.[3]

Étape 4 : Confirmation

Le bloc contenant la transaction est ajouté à la chaîne.

À partir de là, chaque nouveau bloc empilé par-dessus rend un retour en arrière plus coûteux. On dit alors que la transaction gagne une "confirmation" supplémentaire.

Confirmations	Signification
0	Transaction en mempool, non confirmée
1	Incluse dans le dernier bloc
6	Considérée comme pratiquement irréversible

Conséquence directe

Ce que chaque étape **garantit** :

- Après **propagation** : la transaction est connue du réseau, mais peut encore être remplacée ou ignorée
- Après **1 confirmation** : la transaction fait partie de la chaîne, mais une **réorganisation** — un événement rare où la chaîne se restructure temporairement quand deux blocs sont trouvés simultanément — reste possible
- Après **6 confirmations** : une réorganisation nécessiterait une puissance de calcul colossale

Ce que cela **ne garantit pas** :

- Une transaction non confirmée n'est **jamais** garantie d'être incluse
- Le nombre de confirmations nécessaires dépend du montant : pour quelques euros, 1 confirmation suffit souvent
- Pour des montants élevés, on attend fréquemment 6 confirmations (~1 heure), afin que le coût d'une attaque reste supérieur au gain potentiel

> **À retenir**
>
> Une transaction passe par quatre étapes : création, propagation sur le réseau, inclusion dans un bloc par un mineur, puis confirmation par les blocs suivants. Plus il y a de confirmations, plus la transaction est difficile à annuler.
>
> ---
>
> **Références**
>
> [1] Construction des transactions : src/wallet/spend.cpp — fonctions de création et signature
>
> [2] Propagation des transactions : src/net_processing.cpp — gestion des messages `inv`, `getdata`, `tx`
>
> [3] Sélection des transactions pour le minage : src/node/miner.cpp — création des blocs candidats

Synthèse du chapitre

Ce premier chapitre a posé les fondations :

1. **Bitcoin Core** est le logiciel de référence qui fait fonctionner le réseau Bitcoin en tant que nœud complet indépendant
2. **L'architecture** est modulaire : cryptographie, consensus, réseau et portefeuille sont séparés
3. **Le cycle de vie** d'une transaction comprend quatre étapes : création → propagation → inclusion → confirmation

Le chapitre suivant plongera dans les **primitives cryptographiques** qui sous-tendent la sécurité de l'ensemble du système.

Termes techniques introduits

Terme	Définition
Nœud complet (full node)	Programme qui télécharge et vérifie intégralement toute la chaîne de blocs
Daemon	Programme qui s'exécute en arrière-plan sans interface graphique
UTXO (Unspent Transaction Output)	Sortie de transaction non encore dépensée ; représente des fonds disponibles
UTXO set	Ensemble des UTXO disponibles ; état courant des fonds dépensables
Mempool (memory pool)	Ensemble des transactions valides en attente d'inclusion dans un bloc
Signature cryptographique	Preuve mathématique qu'une transaction est autorisée, sans révéler le secret utilisé pour la produire
Confirmation	Chaque bloc ajouté après celui contenant une transaction
Consensus	Ensemble des règles que tous les nœuds doivent respecter
Politique (policy)	Règles locales plus strictes que le consensus, pour le relais
Chaîne de blocs (blockchain)	Historique ordonné des blocs, mis à jour au fil du temps
Pair-à-pair (peer-to-peer, P2P)	Réseau sans serveur central, où les nœuds se connectent directement
Pruning (élagage)	Mode où le nœud vérifie les blocs mais ne conserve pas l'historique complet sur disque
Mineur	Participant qui produit des blocs en cherchant une preuve de travail
RPC (Remote Procedure Call)	Interface de commande permettant de piloter `bitcoind` à distance
Script	Langage qui exprime les règles à respecter pour dépenser des fonds
Preuve de travail (proof-of-work, PoW)	Mécanisme coûteux à produire mais facile à vérifier, utilisé pour sécuriser l'ajout de blocs
Bloc candidat	Bloc en préparation, avant de trouver une preuve de travail valide
Réorganisation (reorg)	Réécriture temporaire de la partie la plus récente de la chaîne, pouvant retirer des confirmations