Minage et Consensus

2022-03-09

# Minage et Consensus

## Présentation

Le mot "exploitation minière" est quelque peu trompeur. En évoquant l'extraction de métaux précieux, il focalise notre attention sur la récompense du minage, le nouveau bitcoin créé dans chaque bloc. Bien que l'exploitation minière soit incitée par cette récompense, l'objectif principal de l'exploitation minière n'est pas la récompense ou la génération de nouvelles pièces. Si vous considérez l'exploitation minière uniquement comme le processus par lequel les pièces sont créées, vous confondez les moyens (incitations) avec le but du processus. Le minage est le mécanisme qui sous-tend la chambre de compensation décentralisée, par laquelle les transactions sont validées et compensées. Le minage est l'invention qui rend le bitcoin spécial, un mécanisme de sécurité décentralisé qui est à la base de l'argent numérique P2P.

L'exploitation minière *sécurise le système Bitcoin* et permet l'émergence d'un *consensus à l'échelle du réseau sans autorité centrale*. La récompense des pièces nouvellement frappées et des frais de transaction est un système d'incitation qui aligne les actions des mineurs sur la sécurité du réseau, tout en mettant en œuvre simultanément l'offre monétaire.

Le but du minage n'est pas la création de nouveaux bitcoins. C'est le système d'incitation. Le minage est le mécanisme par lequel la *sécurité* du bitcoin est *décentralisée*.

Les mineurs valident les nouvelles transactions et les enregistrent dans le grand livre mondial. Un nouveau bloc, contenant les transactions qui se sont produites depuis le dernier bloc, est "miné" toutes les 10 minutes en moyenne, ajoutant ainsi ces transactions à la blockchain. Les transactions qui font partie d'un bloc et ajoutées à la blockchain sont considérées comme "confirmées", ce qui permet aux nouveaux propriétaires de bitcoin de dépenser le bitcoin qu'ils ont reçu dans ces transactions.

Les mineurs reçoivent deux types de récompenses en échange de la sécurité fournie par le minage : de nouvelles pièces créées avec chaque nouveau bloc, également appelées *récompense de bloc* ou *récompense de coinbase*, et les frais de transaction de toutes les transactions incluses dans le bloc. Pour gagner cette récompense, les mineurs s'affrontent pour résoudre un problème mathématique difficile basé sur un algorithme de hachage cryptographique. La solution au problème, appelée Proof-of-Work, est incluse dans le nouveau bloc et agit comme preuve que le mineur a déployé des efforts informatiques importants. La compétition pour résoudre l'algorithme Proof-of-Work pour gagner la récompense et le droit d'enregistrer les transactions sur la blockchain est la base du modèle de sécurité de Bitcoin.

Le processus est appelé minage parce que la récompense (nouvelle génération de pièces) est conçue pour simuler des rendements décroissants, tout comme le minage de métaux précieux. La masse monétaire de Bitcoin est créée par l'exploitation minière, de la même manière qu'une banque centrale émet de la nouvelle monnaie en imprimant des billets de banque. La quantité maximale de bitcoins nouvellement créés qu'un mineur peut ajouter à un bloc diminue environ tous les quatre ans (ou précisément tous les 210 000 blocs). Il a commencé à 50 bitcoins par bloc en janvier 2009 et a été réduit de moitié à 25 bitcoins par bloc en novembre 2012. Il a été réduit de moitié à 12,5 bitcoins en juillet 2016 et de nouveau à 6,25 bitcoins en mai 2020. Sur la base de cette formule, les récompenses de minage de bitcoins diminuent de façon exponentielle. jusqu'à environ l'an 2140, lorsque tous les bitcoins (20,99999998 millions) auront été émis. Après 2140, aucun nouveau bitcoin ne sera émis.

Les mineurs de Bitcoin gagnent également des frais sur les transactions. Chaque transaction comprend généralement des frais de transaction, sous la forme d'un surplus de bitcoin entre les entrées et les sorties de la transaction. Le mineur de bitcoin gagnant peut "garder la monnaie" sur les transactions incluses dans le bloc gagnant. Aujourd'hui, les frais représentent 0,5% ou moins des revenus d'un mineur de bitcoin, la grande majorité provenant du bitcoin nouvellement créé. Cependant, à mesure que la récompense de bloc diminue avec le temps et que le nombre de transactions par bloc augmente, une plus grande proportion des revenus de minage de bitcoins proviendra des frais. Progressivement, la récompense minière sera dominée par les frais de transaction, qui constitueront la principale incitation pour les mineurs. Après 2140, la quantité de nouveaux bitcoins dans chaque bloc tombe à zéro et l'extraction de bitcoins ne sera encouragée que par les frais de transaction.

Dans ce chapitre, nous examinerons d'abord l'exploitation minière en tant que mécanisme d'approvisionnement monétaire, puis nous examinerons la fonction la plus importante de l'exploitation minière : le mécanisme de consensus décentralisé qui sous-tend la sécurité du bitcoin.

Pour comprendre l'exploitation minière et le consensus, nous suivrons la transaction d'Alice telle qu'elle est reçue et ajoutée à un bloc par l'équipement minier de Jing. Ensuite, nous suivrons le bloc tel qu'il est extrait, ajouté à la blockchain et accepté par le réseau Bitcoin à travers le processus de consensus émergent.

### Économie Bitcoin et création de devises

Bitcoin sont "frappés" lors de la création de chaque bloc à un taux fixe et décroissant. Chaque bloc, généré en moyenne toutes les 10 minutes, contient des bitcoins entièrement nouveaux, créés à partir de rien. Tous les 210 000 blocs, soit environ tous les quatre ans, le taux d'émission des devises est diminué de 50 %. Pendant les quatre premières années de fonctionnement du réseau, chaque bloc contenait 50 nouveaux bitcoins.

En novembre 2012, le nouveau taux d'émission de bitcoins a été réduit à 25 bitcoins par bloc. En juillet 2016, il a été réduit à 12,5 bitcoins par bloc, et en mai 2020, il a de nouveau été réduit à 6,25 bitcoins par bloc. Le taux de nouvelles pièces diminue ainsi de manière exponentielle sur 32 "halvings" jusqu'au bloc 6 720 000 (miné environ en 2137), lorsqu'il atteint l'unité monétaire minimale de 1 satoshi. Enfin, après 6,93 millions de blocs, en 2140 environ, près de 2 099 999 997 690 000 satoshis, soit près de 21 millions de bitcoins, seront émis. Par la suite, les blocs ne contiendront plus de nouveaux bitcoins et les mineurs seront récompensés uniquement par les frais de transaction. <<bitcoin\_money\_supply> > montre le bitcoin total en circulation au fil du temps, à mesure que l'émission de devises diminue.

image

images/mbc2\_1001.png["BitcoinMoneySupply"]

Le nombre maximum de pièces minées est la *limite supérieure* des récompenses minières possibles pour le bitcoin. En pratique, un mineur peut intentionnellement exploiter un bloc en prenant moins que la récompense complète. Ces blocs ont déjà été extraits et d'autres pourraient l'être à l'avenir, ce qui entraînerait une émission totale inférieure de la monnaie.

Dans l'exemple de code dans <<max\_money> >, nous calculons le montant total de bitcoin qui sera émis.

inclure::code/max\_money.py[]

<<max\_money\_run> > affiche la sortie produite en exécutant ce script.

$ python max\_money.py  
Total BTC à créer : 2100000000000000.0 Satoshis

L'émission finie et décroissante crée une masse monétaire fixe qui résiste à l'inflation. Contrairement à une monnaie fiduciaire, qui peut être imprimée en nombre infini par une banque centrale, le bitcoin ne peut jamais être gonflé par l'impression.

La conséquence la plus importante et la plus débattue d'une émission monétaire fixe et décroissante est que la monnaie a tendance à être intrinsèquement déflationniste. La déflation est le phénomène d'appréciation de la valeur due à une inadéquation entre l'offre et la demande qui fait monter la valeur (et le taux de change) d'une monnaie. Le contraire de l'inflation, la déflation des prix, signifie que l'argent a plus de pouvoir d'achat au fil du temps.

De nombreux économistes soutiennent qu'une économie déflationniste est une catastrophe qu'il faut éviter à tout prix. En effet, en période de déflation rapide, les gens ont tendance à accumuler de l'argent au lieu de le dépenser, en espérant que les prix chuteront. Un tel phénomène s'est déroulé au cours de la « décennie perdue » du Japon, lorsqu'un effondrement complet de la demande a poussé la monnaie dans une spirale déflationniste.

Les experts du Bitcoin affirment que la déflation n'est pas mauvaise en soi. La déflation est plutôt associée à un effondrement de la demande car c'est le seul exemple de déflation que nous ayons à étudier. Dans une monnaie fiduciaire avec possibilité d'impression illimitée, il est très difficile d'entrer dans une spirale déflationniste à moins qu'il n'y ait un effondrement complet de la demande et une réticence à imprimer de l'argent. La déflation du bitcoin n'est pas causée par un effondrement de la demande, mais par une offre limitée de manière prévisible.

L'aspect positif de la déflation, bien sûr, c'est qu'elle est le contraire de l'inflation. L'inflation provoque une dépréciation lente mais inévitable de la monnaie, se traduisant par une forme de taxation cachée qui punit les épargnants afin de renflouer les débiteurs (y compris les plus gros débiteurs, les gouvernements eux-mêmes). Les monnaies sous contrôle gouvernemental souffrent de l'aléa moral d'une émission facile de dette qui peut ensuite être effacée par la dépréciation aux dépens des épargnants.

Il reste à voir si l'aspect déflationniste de la monnaie est un problème lorsqu'il n'est pas entraîné par une rétraction économique rapide, ou un avantage parce que la protection contre l'inflation et l'avilissement l'emporte largement sur les risques de déflation.

## Consensus décentralisé

Dans le chapitre précédent, nous avons examiné la blockchain, le registre public mondial (liste) de tous transactions, que tout le monde dans le réseau Bitcoin accepte comme enregistrement de propriété faisant autorité.

Mais comment tout le monde dans le réseau peut-il s'entendre sur une seule « vérité » universelle sur qui possède quoi, sans avoir à faire confiance à personne ? Tous les systèmes de paiement traditionnels dépendent d'un modèle de confiance qui a une autorité centrale fournissant un service de chambre de compensation, vérifiant et compensant essentiellement toutes les transactions. Bitcoin n'a pas d'autorité centrale, mais d'une manière ou d'une autre, chaque nœud complet possède une copie complète d'un grand livre public auquel il peut faire confiance en tant qu'enregistrement faisant autorité. La blockchain n'est pas créée par une autorité centrale, mais est assemblée indépendamment par chaque nœud du réseau. D'une manière ou d'une autre, chaque nœud du réseau, agissant sur les informations transmises via des connexions réseau non sécurisées, peut arriver à la même conclusion et assembler une copie du même registre public que tout le monde. Ce chapitre examine le processus par lequel le réseau Bitcoin parvient à un consensus mondial sans autorité centrale.

La principale invention de Satoshi Nakamoto est le mécanisme décentralisé de *consensus émergent*. Emergent, parce que le consensus n'est pas atteint explicitement - il n'y a pas d'élection ou de moment fixe où le consensus se produit. Au lieu de cela, le consensus est un artefact émergent de l'interaction asynchrone de milliers de nœuds indépendants, qui suivent tous des règles simples. Toutes les propriétés du bitcoin, y compris la monnaie, les transactions, les paiements et le modèle de sécurité qui ne dépend pas de l'autorité centrale ou de la confiance, découlent de cette invention.

Le consensus décentralisé de Bitcoin émerge de l'interaction de quatre processus qui se produisent indépendamment sur les nœuds du réseau :

* Vérification indépendante de chaque transaction, par chaque nœud complet, sur la base d'une liste complète de critères
* Agrégation indépendante de ces transactions dans de nouveaux blocs par des nœuds de minage, associée à un calcul démontré via un algorithme de preuve de travail
* Vérification indépendante des nouveaux blocs par chaque nœud et assemblage dans une chaîne
* Sélection indépendante, par chaque nœud, de la chaîne avec le calcul le plus cumulatif démontré par la preuve de travail

Dans les prochaines sections, nous examinerons ces processus et la manière dont ils interagissent pour créer la propriété émergente du consensus à l'échelle du réseau qui permet à n'importe quel nœud Bitcoin d'assembler sa propre copie du grand livre public mondial faisant autorité.

## Vérification indépendante des transactions

Dans <<transactions> >, nous avons vu comment un logiciel de portefeuille crée des transactions en collectant des UTXO, en fournissant les scripts de déverrouillage appropriés, puis en construisant de nouvelles sorties attribuées à un nouveau propriétaire. La transaction résultante est ensuite envoyée aux nœuds voisins du réseau Bitcoin afin qu'elle puisse être propagée sur l'ensemble du réseau Bitcoin.

Cependant, avant de transmettre les transactions à ses voisins, chaque nœud Bitcoin qui reçoit une transaction vérifiera d'abord la transaction. Cela garantit que seules les transactions valides sont propagées sur le réseau, tandis que les transactions non valides sont rejetées au premier nœud qui les rencontre.

Chaque nœud vérifie chaque transaction par rapport à une longue liste de critères :

* La syntaxe et la structure des données de la transaction doivent être correctes.
* Ni les listes d'entrées ni de sorties ne sont vides.
* La taille de la transaction est inférieure à la taille maximale autorisée pour un bloc excluant les données témoins, comme indiqué dans [*tx\_check.cpp*](https://bit.ly/3t623VL).
* Chaque valeur de sortie, ainsi que le total, doit être dans la plage de valeurs autorisée (moins de 21 millions de pièces, plus que le seuil de *poussière*).
* Aucune des entrées n'a hash=0, N=–1 (les transactions coinbase ne doivent pas être relayées).
* nLocktime est égal à INT\_MAX, ou les valeurs nLocktime et nSequence sont satisfaites selon MedianTimePast.
* La taille de la transaction en octets est supérieure ou égale à 82.
* Le nombre d'opérations de signature (SIGOPS) contenues dans la transaction est inférieur à la limite d'opérations de signature.
* Le script de déverrouillage (scriptSig) ne peut pousser que des nombres sur la pile, et le script de verrouillage (scriptPubkey) doit correspondre aux formes IsStandard (cela rejette les transactions "non standard").
* Une transaction correspondante dans le pool ou dans un bloc de la branche principale doit exister.
* Pour chaque entrée, si la sortie référencée existe dans une autre transaction du pool, la transaction doit être rejetée.
* Pour chaque entrée, regardez dans la branche principale et le pool de transactions pour trouver sa transaction parente. Si la transaction parent est manquante pour une entrée, il s'agira d'une transaction orpheline. Ajouter au pool de transactions orphelines, si une transaction correspondante n'est pas déjà dans le pool.
* Pour chaque entrée, si sa transaction parente est une transaction coinbase, elle doit avoir au moins COINBASE\_MATURITY (100) confirmations.
* Pour chaque entrée, la sortie référencée doit exister et ne peut pas déjà être dépensée.
* En utilisant les transactions parentes pour obtenir les valeurs d'entrée, vérifiez que chaque valeur d'entrée, ainsi que la somme, se situent dans la plage de valeurs autorisée (moins de 21 millions de pièces, plus de 0).
* Rejeter si la somme des valeurs d'entrée est inférieure à la somme des valeurs de sortie.
* Rejeter si les frais de transaction seraient trop bas (minRelayTxFee) pour entrer dans un bloc vide.
* Les scripts de déverrouillage pour chaque entrée doivent être validés par rapport aux scripts de verrouillage de sortie correspondants.

Ces conditions peuvent être vues en détail dans les fonctions AcceptToMemoryPool, CheckTransaction et CheckInputs dans Bitcoin Core. Notez que les conditions changent avec le temps, pour faire face à de nouveaux types d'attaques par déni de service ou parfois pour assouplir les règles afin d'inclure davantage de types de transactions.

En vérifiant indépendamment chaque transaction lors de sa réception et avant de la propager, chaque nœud crée un pool de transactions valides (mais non confirmées) connu sous le nom de *transaction pool*, *memory pool* ou *mempool*.

## Nœuds miniers

Certains des nœuds du réseau Bitcoin sont des nœuds spécialisés appelés *miners*. Dans <<ch01\_intro\_what\_is\_bitcoin> > nous avons présenté Jing, un étudiant en génie informatique à Shanghai, en Chine, qui est un mineur de bitcoin. Jing gagne du bitcoin en exécutant une "plate-forme minière", qui est un système informatique spécialisé conçu pour exploiter le bitcoin. Le matériel minier spécialisé de Jing est connecté à un serveur exécutant un nœud Bitcoin complet. Contrairement à Jing, certains mineurs minent sans nœud complet, comme nous le verrons dans <<mining\_pools> >. Comme tous les autres nœuds complets, le nœud de Jing reçoit et propage les transactions non confirmées sur le réseau Bitcoin. Le nœud de Jing, cependant, agrège également ces transactions dans de nouveaux blocs.

Le nœud de Jing est à l'écoute de nouveaux blocs, propagés sur le réseau Bitcoin, comme tous les nœuds. Cependant, l'arrivée d'un nouveau bloc a une signification particulière pour un nœud minier. La compétition entre mineurs se termine effectivement par la propagation d'un nouveau bloc qui fait office d'annonce d'un gagnant. Pour les mineurs, recevoir un nouveau bloc valide signifie que quelqu'un d'autre a gagné la compétition et qu'il a perdu. Cependant, la fin d'un tour d'une compétition est aussi le début du tour suivant. Le nouveau bloc n'est pas seulement un drapeau à damier, marquant la fin de la course ; c'est aussi le pistolet de départ dans la course pour le bloc suivant.

## Agréger les transactions en blocs

Après avoir validé les transactions, un nœud Bitcoin les ajoutera au *memory pool*, ou *transaction pool*, où les transactions attendent jusqu'à ce qu'elles puissent être incluses (exploitées) dans un bloc. Le nœud de Jing collecte, valide et relaie les nouvelles transactions comme n'importe quel autre nœud. Contrairement aux autres nœuds, cependant, le nœud de Jing regroupera ensuite ces transactions dans un *bloc candidat*.

Suivons les blocs qui ont été créés à l'époque où Alice achetait une tasse de café au Bob's Cafe (voir <<cup\_of\_coffee> >). La transaction d'Alice a été incluse dans le bloc 277 316. Dans le but de démontrer les concepts de ce chapitre, supposons que ce bloc a été miné par le système de minage de Jing et suivons la transaction d'Alice au fur et à mesure qu'elle fait partie de ce nouveau bloc.

Le nœud minier de Jing maintient une copie locale de la blockchain. Au moment où Alice achète la tasse de café, le nœud de Jing a assemblé une chaîne jusqu'au bloc 277 314. Le nœud de Jing écoute les transactions, essaie d'extraire un nouveau bloc et écoute également les blocs découverts par d'autres nœuds. Comme le nœud de Jing est en train d'exploiter, il reçoit le bloc 277 315 via le réseau Bitcoin. L'arrivée de ce bloc signifie la fin de la compétition pour le bloc 277,315 et le début de la compétition pour créer le bloc 277,316.

Au cours des 10 minutes précédentes, alors que le nœud de Jing cherchait une solution pour bloquer 277 315, il collectait également des transactions en vue du bloc suivant. À ce jour, il a collecté quelques centaines de transactions dans le pool de mémoire. Lors de la réception du bloc 277,315 et de sa validation, le nœud de Jing le comparera également à toutes les transactions du pool de mémoire et supprimera celles qui étaient incluses dans le bloc 277,315. Quelles que soient les transactions restantes dans le pool de mémoire, elles ne sont pas confirmées et attendent d'être enregistrées dans un nouveau bloc.

Le nœud de Jing construit immédiatement un nouveau bloc vide, un candidat pour le bloc 277 316. Ce bloc est appelé *bloc candidat* car ce n'est pas encore un bloc valide, car il ne contient pas de preuve de travail valide. Le bloc ne devient valide que si le mineur réussit à trouver une solution à l'algorithme Proof-of-Work.

Lorsque le nœud de Jing regroupe toutes les transactions du pool de mémoire, le nouveau bloc candidat contient 418 transactions avec des frais de transaction totaux de 0,09094928 bitcoin. Vous pouvez voir ce bloc dans la blockchain à l'aide de l'interface de ligne de commande du client Bitcoin Core, comme indiqué dans <<block277316> >.

$ bitcoin-cli getblockhash 277316  
  
0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4  
  
$ bitcoin-cli getblock 0000000000000001b6b9a13b095e96db41c4a928b97ef2d9\  
44a9b31b2cc7bdc4

{  
 &quot;hachage&quot; : &quot;0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4&quot;,  
 &quot;confirmations&quot; : 35561,  
 &quot;taille&quot; : 218629,  
 &quot;hauteur&quot; : 277316,  
 &quot;version 2,  
 &quot;merkleroot&quot; : &quot;c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e&quot;,  
 &quot;tx&quot; : [  
 &quot;d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f&quot;,  
 &quot;b268b45c59b39d759614757718b9918caf0ba9d97c56f3b91956ff877c503fbe&quot;,  
  
 ... 417 transactions supplémentaires ...  
  
 ],  
 &quot;heure&quot; : 1388185914,  
 &quot;nonce&quot; : 924591752,  
 &quot;bits&quot; : &quot;1903a30c&quot;,  
 &quot;difficulté&quot; : 1180923195.25802612,  
 &quot;chaînette&quot; : &quot;00000000000000000000000000000000000000000934695e92aaf53afa1a&quot;,  
 &quot;hachage de bloc précédent&quot; : &quot;0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569&quot;  
}

### La transaction Coinbase

La première transaction d'un bloc est une transaction spéciale, appelée *coinbase transaction*. Cette transaction est construite par le nœud de Jing et contient sa *récompense* pour l'effort de minage.

Lorsque le bloc 277 316 a été miné, la récompense était de 25 bitcoins par bloc. Depuis, deux périodes de "halving" se sont écoulées. La récompense globale est passée à 12,5 bitcoins en juillet 2016 et à 6,25 bitcoins en mai 2020.

Le nœud de Jing crée la transaction coinbase en tant que paiement sur son propre portefeuille : "Pay Jing's address 25.09094928 bitcoin." Le montant total de la récompense que Jing collecte pour l'extraction d'un bloc est la somme de la récompense coinbase (25 nouveaux bitcoins) et des frais de transaction (0,09094928) de toutes les transactions incluses dans le bloc, comme indiqué dans <<generation\_tx\_example> >.

Transaction .Coinbase

$ bitcoin-cli getrawtransaction d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f 1

{  
 &quot;Hex&quot;: &quot;01000000010000000000000000000000000000000000000000000000000000000000000000ffffffff0f03443b0403858402062f503253482fffffffff0110c08d9500000000232102aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac00000000&quot;,  
 &quot;txid&quot; : &quot;d5ada064c6417ca25c4308bd158c34b77e1c0eca2a73cda16c737e7424afba2f&quot;,  
 &quot;version 1,  
 &quot;temps de verrouillage&quot; : 0,  
 &quot;vin&quot; : [  
 {  
 &quot;coinbase&quot; : &quot;03443b0403858402062f503253482f&quot;,  
 &quot;séquence&quot; : 4294967295  
 }  
 ],  
 &quot;vouter&quot; : [  
 {  
 &quot;valeur&quot; : 25.09094928,  
 &quot;n&quot; : 0,  
 &quot;scriptPubKey&quot; : {  
 &quot;asm&quot; : &quot;02aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21OP\_CHECKSIG&quot;,  
 &quot;hex&quot; : &quot;2102aa970c592640d19de03ff6f329d6fd2eecb023263b9ba5d1b81c29b523da8b21ac&quot;,  
 &quot;reqSigs&quot; : 1,  
 &quot;type&quot; : &quot;pubkey&quot;,  
 &quot;adresses&quot; : [  
 &quot;1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N&quot;  
 ]  
 }  
 }  
 ]  
}

Contrairement aux transactions régulières, la transaction coinbase ne consomme pas (dépense) UTXO comme entrées. Au lieu de cela, il n'a qu'une seule entrée, appelée *coinbase*, qui crée du bitcoin à partir de rien. La transaction coinbase a une sortie, payable à la propre adresse Bitcoin du mineur. La sortie de la transaction coinbase envoie la valeur de 25,09094928 bitcoin à l'adresse Bitcoin du mineur ; dans ce cas c'est 1MxTkeEP2PmHSMze5tUZ1hAV3YTKu2Gh1N.

### Récompense et frais Coinbase

transaction coinbase, le nœud de Jing calcule d'abord le montant total des frais de transaction en ajoutant toutes les entrées et sorties des 418 transactions qui ont été ajoutées au bloc. Les frais sont calculés comme suit :

Frais totaux = Somme (entrées) - Somme (sorties)

Dans le bloc 277 316, le total des frais de transaction est de 0,09094928 bitcoin.

Ensuite, le nœud de Jing calcule la récompense correcte pour le nouveau bloc. La récompense est calculée en fonction de la hauteur du bloc, à partir de 50 bitcoins par bloc et réduite de moitié tous les 210 000 blocs. Parce que ce bloc est à la hauteur 277 316, la récompense correcte est de 25 bitcoins.

Le calcul peut être vu dans la fonction GetBlockSubsidy dans le client Bitcoin Core, comme indiqué dans <<getblocksubsidy\_source> >.

CAmount GetBlockSubsidy(int nHeight, const Consensus ::Params&amp; consensusParams)  
{  
 int halvings = nHeight / consensusParams.nSubsidyHalvingInterval ;  
 // Force la récompense du bloc à zéro lorsque le décalage vers la droite n&#39;est pas défini.  
 si (moitiés &gt;= 64)  
 renvoie 0 ;  
  
 CAmount nSubsidy = 50 \* COIN ;  
 // La subvention est réduite de moitié tous les 210 000 blocs, ce qui se produira environ tous les 4 ans.  
 nSubvention &gt;&gt;= moitiés ;  
 retourner nSubvention ;  
}

La subvention initiale est calculée en satoshis en multipliant 50 par la constante COIN (100 000 000 satoshis). Cela fixe la récompense initiale (nSubsidy) à 5 milliards de satoshis.

Ensuite, la fonction calcule le nombre de halvings qui se sont produits en divisant la hauteur actuelle du bloc par l'intervalle de halving (SubsidyHalvingInterval). Dans le cas du bloc 277 316, avec un intervalle de moitié tous les 210 000 blocs, le résultat est 1 moitié.

Le nombre maximum de moitiés autorisé est de 64, donc le code impose une récompense nulle (ne renvoie que les frais) si les 64 moitiés sont dépassées.

Ensuite, la fonction utilise l'opérateur binaire de décalage vers la droite pour diviser la récompense (nSubsidy) par deux pour chaque cycle de réduction de moitié. Dans le cas du bloc 277 316, cela entraînerait un décalage binaire vers la droite de la récompense de 5 milliards de satoshis une fois (une réduction de moitié) et donnerait 2,5 milliards de satoshis, soit 25 bitcoins. L'opérateur de décalage binaire vers la droite est utilisé car il est plus efficace que plusieurs divisions répétées. Pour éviter un bogue potentiel, l'opération de décalage est ignorée après 63 moitiés et la subvention est définie sur 0.

Enfin, la récompense coinbase (nSubsidy) est ajoutée aux frais de transaction (nFees), et la somme est restituée.

Si le nœud minier de Jing écrit la transaction coinbase, qu'est-ce qui empêche Jing de se "récompenser" lui-même 100 ou 1000 bitcoins ? La réponse est qu'une récompense incorrecte aurait pour conséquence que le bloc serait considéré comme invalide par tout le monde, gaspillant l'électricité de Jing utilisée pour la preuve de travail. Jing ne peut dépenser la récompense que si le blocage est accepté par tout le monde.

### Structure de la transaction Coinbase

Avec ces calculs, le nœud de Jing construit ensuite la transaction coinbase pour se payer 25,09094928 bitcoins.

Comme vous pouvez le voir dans <<generation\_tx\_example> >, la transaction coinbase a un format spécial. Au lieu d'une entrée de transaction spécifiant un UTXO précédent à dépenser, il a une entrée "coinbase". Nous avons examiné les entrées de transaction dans <<tx\_in\_structure> >. Comparons une entrée de transaction régulière avec une entrée de transaction coinbase. <<table\_8-1> > montre la structure d'une entrée de transaction normale, tandis que <<table\_8-2> > montre la structure de l'entrée de la transaction coinbase.

La structure d'une entrée de transaction "normale"

|  |  |  |
| --- | --- | --- |
| Taille | Champ | La description |
| 32 octets | Hachage de transaction | Pointeur vers la transaction contenant l'UTXO à dépenser |
| 4 octets | Indice de sortie | Le numéro d'index de l'UTXO à dépenser, le premier est 0 |
| 1–9 octets (VarInt) | Taille du script de déverrouillage | Longueur du script de déverrouillage en octets, à suivre |
| variables | Déverrouillage-Script | Un script remplissant les conditions du script de verrouillage UTXO |
| 4 octets | Numéro de séquence | Généralement défini sur 0xFFFFFFFF pour désactiver BIP 125 et BIP 68 |

La structure d'une entrée de transaction coinbase

|  |  |  |
| --- | --- | --- |
| Taille | Champ | La description |
| 32 octets | Hachage de transaction | Tous les bits sont à zéro : pas une référence de hachage de transaction |
| 4 octets | Indice de sortie | Tous les bits sont des uns : 0xFFFFFFFF |
| 1–9 octets (VarInt) | Taille des données Coinbase | Longueur des données coinbase, de 2 à 100 octets |
| variables | Données Coinbase | Données arbitraires utilisées pour les nonce supplémentaires et les balises minières. Dans les blocs v2 ; doit commencer par la hauteur du bloc |
| 4 octets | Numéro de séquence | Réglé sur 0xFFFFFFFF |

Dans une transaction coinbase, les deux premiers champs sont définis sur des valeurs qui ne représentent pas une référence UTXO. Au lieu d'un "hachage de transaction", le premier champ est rempli de 32 octets tous mis à zéro. "L'index de sortie" est rempli de 4 octets tous mis à 0xFF (255 décimal). Le "Script de déverrouillage" (scriptSig) est remplacé par les données coinbase, un champ de données utilisé par les mineurs, comme nous le verrons ensuite.

### Données Coinbase

Les transactions Coinbase n'ont pas de champ de script de déverrouillage (aka, scriptSig). Au lieu de cela, ce champ est remplacé par des données coinbase, qui doivent être comprises entre 2 et 100 octets. À l'exception des premiers octets, le reste des données de coinbase peut être utilisé par les mineurs comme ils le souhaitent ; ce sont des données arbitraires.

("bloc de genèse" ) Dans le bloc de genèse, par exemple, Satoshi Nakamoto a ajouté le texte "The Times 03/Jan/2009 Chancellor on bound of second bailout for banks" dans les données coinbase, l'utilisant comme preuve de la date et pour transmettre un message. Actuellement, les mineurs utilisent les données coinbase pour inclure des valeurs nonce supplémentaires et des chaînes identifiant le pool de minage.

Les premiers octets de la coinbase étaient auparavant arbitraires, mais ce n'est plus le cas. Conformément à BIP-34, les blocs de version 2 (blocs avec le champ de version défini sur 2) doivent contenir l'index de hauteur de bloc en tant qu'opération "push" de script au début du champ coinbase.

Dans le bloc 277 316, nous voyons que la coinbase (voir <<generation\_tx\_example> >), qui se trouve dans le script de déverrouillage ou le champ scriptSig de l'entrée de transaction, contient la valeur hexadécimale 03443b0403858402062f503253482f. Décodons cette valeur.

Le premier octet, 03, indique au moteur d'exécution de script de pousser les trois octets suivants sur la pile de script (voir <<tx\_script\_ops\_table\_pushdata> >). Les trois octets suivants, 0x443b04, sont la hauteur de bloc encodée au format little-endian (en arrière, octet le moins significatif en premier). Inversez l'ordre des octets et le résultat est 0x043b44, soit 277 316 en décimal.

Les quelques chiffres hexadécimaux suivants (0385840206) sont utilisés pour coder un *nonce* supplémentaire (voir <<extra\_nonce> >), ou valeur aléatoire, utilisée pour trouver une solution de preuve de travail appropriée.

<span class="keep-together"><code>/P2SH/</code></span> ], ce qui indique que le nœud de minage qui a miné ce bloc prend en charge l'amélioration P2SH définie dans BIP-16. L'introduction de la capacité P2SH a nécessité une signalisation par les mineurs pour approuver le BIP-16 ou le BIP-17. Ceux qui approuvaient la mise en œuvre du BIP-16 devaient inclure la chaîne /P2SH/ dans leurs données coinbase. Ceux qui approuvaient l'implémentation BIP-17 de P2SH devaient inclure la chaîne p2sh/CHV dans leurs données coinbase. Enfin, le BIP-16 a été élu vainqueur, et de nombreux mineurs ont continué à inclure la chaîne /P2SH/ dans leur coinbase pour indiquer qu'ils prennent en charge cette fonctionnalité.

<<satoshi\_words> > utilise la bibliothèque libbitcoin introduite dans <<alt\_libraries> > pour extraire les données coinbase du bloc genesis, affichant le message de Satoshi. Notez que la bibliothèque libbitcoin contient une copie statique du bloc genesis, de sorte que l'exemple de code peut récupérer le bloc genesis directement à partir de la bibliothèque.

inclure::code/satoshi-words.cpp[]

Nous compilons le code avec le compilateur GNU C++ et exécutons l'exécutable résultant, comme indiqué dans <<satoshi\_words\_run> >.

# Compiler le code  
$ g++ -o satoshi-words satoshi-words.cpp $(pkg-config --cflags --libs libbitcoin)  
# Lancer l&#39;exécutable  
$ ./mots satoshi  
^D��<GS> ^A^DEThe Times 03/Jan/2009 La chancelière est sur le point de renflouer les banques

## Construction de l'en-tête de bloc

Pour construire l'en-tête de bloc, le nœud d'exploration de données doit remplir six champs, comme indiqué dans <<block\_header\_structure\_ch10> >.

La structure de l'entête du bloc

|  |  |  |
| --- | --- | --- |
| Taille | Champ | La description |
| 4 octets | Version | Un numéro de version pour suivre les mises à niveau du logiciel/protocole |
| 32 octets | Hachage de bloc précédent | Une référence au hachage du bloc précédent (parent) dans la chaîne |
| 32 octets | Racine de Merkel | Un hachage de la racine de l'arbre Merkle des transactions de ce bloc |
| 4 octets | Horodatage | Le temps de création approximatif de ce bloc (secondes à partir d'Unix Epoch) |
| 4 octets | Cible | La cible de l'algorithme de preuve de travail pour ce bloc |
| 4 octets | Nonce | Un compteur utilisé pour l'algorithme Proof-of-Work |

Au moment où le bloc 277 316 a été miné, le numéro de version décrivant la structure du bloc est la version 2, qui est codée au format petit-boutiste sur 4 octets sous la forme 0x02000000.

Ensuite, le nœud de minage doit ajouter le "Previous Block Hash" (également appelé prevhash). C'est le hachage de l'en-tête de bloc du bloc 277,315, le bloc précédent reçu du réseau, que le nœud de Jing a accepté et sélectionné comme *parent* du bloc candidat 277,316. Le hachage d'en-tête de bloc pour le bloc 277 315 est :

0000000000000002a7bbd25a417c0374cc55261021e8a9ca74442b01284f0569

En sélectionnant le bloc *parent* spécifique, indiqué par le champ Hash du bloc précédent dans l'en-tête du bloc candidat, Jing engage sa puissance minière à étendre la chaîne qui se termine dans ce bloc spécifique. En substance, c'est ainsi que Jing "vote" avec son pouvoir minier pour la chaîne valide la plus longue.

L'étape suivante consiste à résumer toutes les transactions avec un arbre merkle, afin d'ajouter la racine merkle à la en-tête de bloc. La transaction coinbase est répertoriée comme la première transaction du bloc. Ensuite, 418 transactions supplémentaires sont ajoutées après, pour un total de 419 transactions dans le bloc. Comme nous l'avons vu dans le <<merkle\_trees> >, il doit y avoir un nombre pair de nœuds "feuilles" dans l'arbre, donc la dernière transaction est dupliquée, créant 420 nœuds, chacun contenant le hachage d'une transaction. Les hachages de transaction sont ensuite combinés, par paires, créant chaque niveau de l'arbre, jusqu'à ce que toutes les transactions soient résumées en un nœud à la "racine" de l'arbre. La racine de l'arborescence merkle résume toutes les transactions en une seule valeur de 32 octets, que vous pouvez voir répertoriée comme "racine merkle" dans <<block277316> >, et ici :

c91c008c26e50763e9f548bb8b2fc323735f73577effbc55502c51eb4cc7cf2e

Le nœud minier de Jing ajoutera alors un horodatage de 4 octets, codé comme un horodatage "d'époque" Unix, qui est basé sur le nombre de secondes écoulées depuis minuit UTC, le jeudi 1er janvier 1970. L'heure 1388185914 est égale à vendredi , 27 décembre 2013, 23:11:54 UTC.

Le nœud de Jing remplit ensuite la cible, qui définit la preuve de travail requise pour en faire un bloc valide. La cible est stockée dans le bloc en tant que métrique de "bits cibles", qui est un codage d'exposant de mantisse de la cible. Le codage a un exposant de 1 octet, suivi d'une mantisse de 3 octets (coefficient). Dans le bloc 277 316, par exemple, la valeur des bits cibles est 0x1903a30c. La première partie 0x19 est un exposant hexadécimal, tandis que la partie suivante, 0x03a30c, est le coefficient. Le concept de cible est expliqué dans <<target> > et la représentation des "bits cibles" est expliquée dans <<target\_bits> >.

Le dernier champ est le nonce, qui est initialisé à zéro.

Une fois tous les autres champs remplis, l'en-tête du bloc est maintenant terminé et le processus d'extraction peut commencer. L'objectif est maintenant de trouver une valeur pour le nonce qui se traduit par un hachage d'en-tête de bloc égal ou inférieur à la cible. Le nœud de minage devra tester des milliards ou des billions de valeurs de nonce avant de trouver un nonce qui réponde à l'exigence.

## Miner le bloc

Maintenant qu'un bloc candidat a été construit par le nœud de Jing, il est temps pour la plate-forme de minage matériel de Jing de "miner" le bloc, pour trouver une solution à l'algorithme de preuve de travail qui rend le bloc valide. Tout au long de ce livre, nous avons étudié les fonctions de hachage cryptographique utilisées dans divers aspects du système Bitcoin. La fonction de hachage SHA256 est la fonction utilisée dans le processus d'extraction de bitcoin.

Dans les termes les plus simples, le minage est le processus de hachage répété de l'en-tête de bloc, en modifiant un paramètre, jusqu'à ce que le hachage résultant corresponde à une cible spécifique. Le résultat de la fonction de hachage ne peut pas être déterminé à l'avance, pas plus qu'un modèle ne peut être créé qui produira une valeur de hachage spécifique. Cette caractéristique des fonctions de hachage signifie que la seule façon de produire un résultat de hachage correspondant à une cible spécifique est d'essayer encore et encore, en modifiant de manière aléatoire l'entrée jusqu'à ce que le résultat de hachage souhaité apparaisse par hasard.

### Algorithme de preuve de travail

Un algorithme de hachage prend une entrée de données de longueur arbitraire et produit un résultat déterministe de longueur fixe, une empreinte numérique de l'entrée. Pour toute entrée spécifique, le hachage résultant sera toujours le même et peut être facilement calculé et vérifié par toute personne mettant en œuvre le même algorithme de hachage. La principale caractéristique d'un algorithme de hachage cryptographique est qu'il est impossible de trouver deux entrées différentes qui produisent la même empreinte digitale (appelée *collision*). En corollaire, il est également pratiquement impossible de sélectionner une entrée de manière à produire une empreinte digitale souhaitée, autrement qu'en essayant des entrées aléatoires.

Avec SHA256, la sortie a toujours une longueur de 256 bits, quelle que soit la taille de l'entrée. Dans <<sha256\_example1> >, nous utiliserons l'interpréteur Python pour calculer le hachage SHA256 de la phrase "Je suis Satoshi Nakamoto".

Exemple .SHA256

$ python

Python 3.7.3  
&gt;&gt;&gt; importer hashlib  
&gt;&gt;&gt; hashlib.sha256(b&quot;Je suis Satoshi Nakamoto&quot;).hexdigest()  
&#39;5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e&#39;

<<sha256\_example1> > montre le résultat du calcul du hachage de &quot;Je suis Satoshi Nakamoto&quot; : 5d7c7ba21cbbcd75d14800b100252d5b428e5b1213d27c385bc141ca6b47989e. Ce nombre de 256 bits est le *hash* ou *digest* de la phrase et dépend de chaque partie de la phrase. L'ajout d'une seule lettre, d'un signe de ponctuation ou de tout autre caractère produira un hachage différent.

Maintenant, si nous changeons la phrase, nous devrions nous attendre à voir des hachages complètement différents. Essayons cela en ajoutant un nombre à la fin de notre phrase, en utilisant le script Python simple dans <<sha256\_example\_generator> >.

Script .SHA256 pour générer de nombreux hachages en itérant sur un nonce

inclure::code/hash\_example.py[]

L'exécution de ceci produira les hachages de plusieurs phrases, rendues différentes en ajoutant un nombre à la fin du texte. En incrémentant le nombre, nous pouvons obtenir différents hachages, comme indiqué dans <<sha256\_example\_generator\_output> >.

$ python hash\_example.py

Je suis Satoshi Nakamoto0 =&gt; a80a81401765c8eddee25df36728d732...  
Je suis Satoshi Nakamoto1 =&gt; f7bc9a6304a4647bb41241a677b5345f...  
Je suis Satoshi Nakamoto2 =&gt; ea758a8134b115298a1583ffb80ae629...  
Je suis Satoshi Nakamoto3 =&gt; bfa9779618ff072c903d773de30c99bd...  
Je suis Satoshi Nakamoto4 =&gt; bce8564de9a83c18c31944a66bde992f...  
Je suis Satoshi Nakamoto5 =&gt; eb362c3cf3479be0a97a20163589038e...  
Je suis Satoshi Nakamoto6 =&gt; 4a2fd48e3be420d0d28e202360cfbaba...  
Je suis Satoshi Nakamoto7 =&gt; 790b5a1349a5f2b909bf74d0d166b17a...  
Je suis Satoshi Nakamoto8 =&gt; 702c45e5b15aa54b625d68dd947f1597...  
Je suis Satoshi Nakamoto9 =&gt; 7007cf7dd40f5e933cd89fff5b791ff0...  
Je suis Satoshi Nakamoto10 =&gt; c2f38c81992f4614206a21537bd634a...  
Je suis Satoshi Nakamoto11 =&gt; 7045da6ed8a914690f087690e1e8d66...  
Je suis Satoshi Nakamoto12 =&gt; 60f01db30c1a0d4cbce2b4b22e88b9b...  
Je suis Satoshi Nakamoto13 =&gt; 0ebc56d59a34f5082aaef3d66b37a66...  
Je suis Satoshi Nakamoto14 =&gt; 27ead1ca85da66981fd9da01a8c6816...  
Je suis Satoshi Nakamoto15 =&gt; 394809fb809c5f83ce97ab554a2812c...  
Je suis Satoshi Nakamoto16 =&gt; 8fa4992219df33f50834465d3047429...  
Je suis Satoshi Nakamoto17 =&gt; dca9b8b4f8d8e1521fa4eaa46f4f0cd...  
Je suis Satoshi Nakamoto18 =&gt; 9989a401b2a3a318b01e9ca9a22b0f3...  
Je suis Satoshi Nakamoto19 =&gt; cda56022ecb5b67b2bc93a2d764e75f...

Chaque phrase produit un résultat de hachage complètement différent. Ils semblent complètement aléatoires, mais vous pouvez reproduire les résultats exacts de cet exemple sur n'importe quel ordinateur avec Python et voir les mêmes hachages exacts.

Le nombre utilisé comme variable dans un tel scénario est appelé un *nonce*. Le nonce est utilisé pour faire varier la sortie d'une fonction cryptographique, dans ce cas pour faire varier l'empreinte SHA256 de la phrase.

Pour relever le défi de cet algorithme, fixons-nous un objectif : trouver une phrase qui produit un hachage hexadécimal commençant par un zéro. Heureusement, ce n'est pas difficile ! <<sha256\_example\_generator\_output> > montre que la phrase "I am Satoshi Nakamoto13" produit le hash 0ebc56d59a34f5082aaef3d66b37a661696c2b618e62432727216ba9531041a5, ce qui correspond à nos critères. Il a fallu 13 tentatives pour le trouver. En termes de probabilités, si la sortie de la fonction de hachage est uniformément distribuée, nous nous attendrions à trouver un résultat avec un 0 comme préfixe hexadécimal une fois tous les 16 hachages (un sur 16 chiffres hexadécimaux de 0 à F). En termes numériques, cela signifie trouver une valeur de hachage inférieure à 0x100000000000000000000000000000000000000000000000000000000000. Nous appelons ce seuil la *cible* et le but est de trouver un hachage qui est numériquement égal ou inférieur à la cible. Si nous diminuons la cible, la tâche de trouver un hachage inférieur à la cible devient de plus en plus difficile.

Pour donner une analogie simple, imaginez un jeu où les joueurs lancent une paire de dés à plusieurs reprises, en essayant de lancer autant ou moins qu'une cible spécifiée. Au premier tour, l'objectif est de 11. À moins que vous ne lanciez un double-six, vous gagnez. Au tour suivant, l'objectif est de 10. Les joueurs doivent lancer 10 ou moins pour gagner, encore une fois une tâche facile. Disons que quelques tours plus tard, l'objectif est tombé à 5. Maintenant, plus de la moitié des lancers de dés dépasseront l'objectif et seront donc invalides. Il faut exponentiellement plus de lancers de dés pour gagner, plus la cible est basse. Finalement, lorsque l'objectif est de 2 (le minimum possible), un seul lancer sur 36, soit 2 % d'entre eux, produira un résultat gagnant.

Du point de vue d'un observateur qui sait que la cible du jeu de dés est 2, si quelqu'un a réussi à lancer un lancer gagnant, on peut supposer qu'il a tenté, en moyenne, 36 lancers. En d'autres termes, on peut estimer la quantité de travail nécessaire pour réussir à partir de la difficulté imposée par la cible. Lorsque l'algorithme est basé sur une fonction déterministe telle que SHA256, l'entrée elle-même constitue la *preuve* qu'une certaine quantité de *travail* a été effectuée pour produire un résultat égal ou inférieur à la cible. Par conséquent, *Preuve de travail*.

Même si chaque tentative produit un résultat aléatoire, la probabilité de tout résultat possible peut être calculée à l'avance. Par conséquent, un résultat d'une difficulté spécifiée constitue la preuve d'une quantité de travail spécifique.

Dans <<sha256\_example\_generator\_output> >, le "nonce" gagnant est 13 et ce résultat peut être confirmé par n'importe qui indépendamment. N'importe qui peut ajouter le nombre 13 comme suffixe à la phrase "Je suis Satoshi Nakamoto" et calculer le hachage, en vérifiant qu'il est inférieur à la cible. Le résultat réussi est également une preuve de travail, car cela prouve que nous avons fait le travail pour trouver ce nonce. Alors qu'il ne faut qu'un seul calcul de hachage pour vérifier, il nous a fallu 13 calculs de hachage pour trouver un nonce qui a fonctionné. Si nous avions une cible inférieure (difficulté plus élevée), il faudrait beaucoup plus de calculs de hachage pour trouver un nonce approprié, mais un seul calcul de hachage pour que quiconque puisse le vérifier. De plus, en connaissant la cible, n'importe qui peut estimer la difficulté à l'aide de statistiques et donc savoir combien de travail a été nécessaire pour trouver un tel nonce.

La preuve de travail doit produire un hachage qui est *égal ou inférieur à* la cible. Une cible plus élevée signifie qu'il est moins difficile de trouver un hachage égal ou inférieur à la cible. Une cible inférieure signifie qu'il est plus difficile de trouver un hachage égal ou inférieur à la cible. La cible et la difficulté sont inversement liées.

La preuve de travail de Bitcoin est très similaire au défi présenté dans <<sha256\_example\_generator\_output> >. Le mineur construit un bloc candidat rempli de transactions. Ensuite, le mineur calcule le hachage de l'en-tête de ce bloc et voit s'il est égal ou inférieur à la *target* actuelle. Si le hachage est supérieur à la cible, le mineur modifiera le nonce (généralement en l'incrémentant simplement de un) et réessayera. À la difficulté actuelle du réseau Bitcoin, les mineurs doivent essayer des quadrillions de fois avant de trouver un nonce qui se traduit par un hachage d'en-tête de bloc suffisamment bas.

Un algorithme de preuve de travail très simplifié est implémenté en Python dans <<pow\_example1> >.

#!/usr/bin/env python  
# example of proof-of-work algorithm  
  
import hashlib  
import time  
  
max\_nonce = 2 \*\* 32 # 4 billion  
  
def proof\_of\_work(header, difficulty\_bits):  
   
 # calculate the difficulty target  
 target = 2 \*\* (256-difficulty\_bits)  
   
 for nonce in xrange(max\_nonce):  
 hash\_result = hashlib.sha256(str(header)+str(nonce)).hexdigest()  
   
 # check if this is a valid result, below the target  
 if long(hash\_result, 16) < target:  
 print "Success with nonce %d" % nonce  
 print "Hash is %s" % hash\_result  
 return (hash\_result,nonce)  
   
 print "Failed after %d (max\_nonce) tries" % nonce  
 return nonce  
  
   
if \_\_name\_\_ == '\_\_main\_\_':  
   
 nonce = 0  
 hash\_result = ''  
   
 # difficulty from 0 to 31 bits   
 for difficulty\_bits in xrange(32):  
   
 difficulty = 2 \*\* difficulty\_bits  
 print "Difficulty: %ld (%d bits)" % (difficulty, difficulty\_bits)  
   
 print "Starting search..."  
   
 # checkpoint the current time  
 start\_time = time.time()  
   
 # make a new block which includes the hash from the previous block  
 # we fake a block of transactions - just a string  
 new\_block = 'test block with transactions' + hash\_result   
   
 # find a valid nonce for the new block  
 (hash\_result, nonce) = proof\_of\_work(new\_block, difficulty\_bits)   
   
 # checkpoint how long it took to find a result  
 end\_time = time.time()  
   
 elapsed\_time = end\_time - start\_time  
 print "Elapsed Time: %.4f seconds" % elapsed\_time  
   
 if elapsed\_time > 0:  
   
 # estimate the hashes per second  
 hash\_power = float(long(nonce)/elapsed\_time)  
 print "Hashing Power: %ld hashes per second" % hash\_power

En exécutant ce code, vous pouvez définir la difficulté souhaitée (en bits, combien de bits de tête doivent être nuls) et voir combien de temps il faut à votre ordinateur pour trouver une solution. Dans <<pow\_example\_outputs> >, vous pouvez voir comment cela fonctionne sur un ordinateur portable moyen.

$ python preuve-de-travail-exemple.py\*

Difficulté : 1 (0 bits)  
  
[...]  
  
Difficulté : 8 (3 bits)  
Lancement de la recherche...  
Succès avec nonce 9  
Le hachage est 1c1c105e65b47142f028a8f93ddf3dabb9260491bc64474738133ce5256cb3c1  
Temps écoulé : 0,0004 seconde  
Puissance de hachage : 25 065 hachages par seconde  
Difficulté : 16 (4 bits)  
Lancement de la recherche...  
Succès avec nonce 25  
Le hachage est 0f7becfd3bcd1a82e06663c97176add89e7cae0268de46f94e7e11bc3863e148  
Temps écoulé : 0,0005 seconde  
Puissance de hachage : 52 507 hachages par seconde  
Difficulté : 32 (5 bits)  
Lancement de la recherche...  
Succès avec nonce 36  
Le hachage est 029ae6e5004302a120630adcbb808452346ab1cf0b94c5189ba8bac1d47e7903  
Temps écoulé : 0,0006 seconde  
Puissance de hachage : 58 164 hachages par seconde  
  
[...]  
  
Difficulté : 4194304 (22 bits)  
Lancement de la recherche...  
Succès avec nonce 1759164  
Le hachage est 0000008bb8f0e731f0496b8e530da984e85fb3cd2bd81882fe8ba3610b6cefc3  
Temps écoulé : 13,3201 secondes  
Puissance de hachage : 132 068 hachages par seconde  
Difficulté : 8388608 (23 bits)  
Lancement de la recherche...  
Succès avec nonce 14214729  
Le hachage est 000001408cf12dbd20fcba6372a223e098d58786c6ff93488a9f74f5df4df0a3  
Temps écoulé : 110,1507 secondes  
Puissance de hachage : 129 048 hachages par seconde  
Difficulté : 16777216 (24 bits)  
Lancement de la recherche...  
Succès avec nonce 24586379  
Le hachage est 0000002c3d6b370fccd699708d1b7cb4a94388595171366b944d68b2acce8b95  
Temps écoulé : 195,2991 secondes  
Puissance de hachage : 125 890 hachages par seconde  
  
[...]  
  
Difficulté : 67108864 (26 bits)  
Lancement de la recherche...  
Succès avec nonce 84561291  
Le hachage est 0000001f0ea21e676b6dde5ad429b9d131a9f2b000802ab2f169cbca22b1e21a  
Temps écoulé : 665,0949 secondes  
Puissance de hachage : 127 141 hachages par seconde

Comme vous pouvez le constater, augmenter la difficulté de 1 bit entraîne un doublement du temps nécessaire pour trouver une solution. Si vous pensez à l'ensemble de l'espace numérique de 256 bits, chaque fois que vous contraignez un bit de plus à zéro, vous réduisez l'espace de recherche de moitié. Dans <<pow\_example\_outputs> >, il faut 84 millions de tentatives de hachage pour trouver un nonce qui produit un hachage avec 26 bits de tête comme zéro. Même à une vitesse de plus de 120 000 hachages par seconde, il faut tout de même 10 minutes sur un portable pour trouver cette solution.

Au moment de la rédaction, le réseau tente de trouver un bloc dont le hachage d'en-tête est égal ou inférieur à :

0000000000000000029AB900000000000000000000000000000000000000000

Comme vous pouvez le voir, il y a beaucoup de zéros au début de cette cible, ce qui signifie que la plage acceptable de hachages est beaucoup plus petite, il est donc plus difficile de trouver un hachage valide. Il faudra en moyenne plus de 1,8 hachages zêta (mille milliards de milliards de hachages) pour que le réseau découvre le bloc suivant. Cela semble être une tâche impossible, mais heureusement, le réseau apporte 3 exa-hachages par seconde (EH/sec) de puissance de traitement, ce qui permettra de trouver un bloc en environ 10 minutes en moyenne.

### Représentation cible

Dans <<block277316> >, nous avons vu que le bloc contient la cible, dans une notation appelée "target bits" ou simplement "bits", qui dans le bloc 277 316 a la valeur 0x1903a30c. Cette notation exprime la cible de preuve de travail sous la forme d'un format coefficient/exposant, avec les deux premiers chiffres hexadécimaux pour l'exposant et les six chiffres hexadécimaux suivants pour le coefficient. Dans ce bloc, par conséquent, l'exposant est 0x19 et le coefficient est 0x03a30c.

La formule pour calculer la cible de difficulté à partir de cette représentation est :

cible = coefficient \* 2

(8\*(exposant–3))

En utilisant cette formule et la valeur des bits de difficulté 0x1903a30c, nous obtenons :

cible = 0x03a30c \* 2

0x08\*(0x19-0x03)

=> cible = 0x03a30c \* 2

(0x08\*0x16)

=> cible = 0x03a30c \* 2

0xB0

qui en décimal est :

=> cible = 238 348 \* 2

176

=> cible =

22 829 202 948 393 929 850 749 706 076 701 368 331 072 452 018 388 575 715 328

retour en hexadécimal :

=> cible =

0x0000000000000003A30C000000000000000000000000000000000000000000

Cela signifie qu'un bloc valide pour la hauteur 277 316 est celui qui a un hachage d'en-tête de bloc inférieur à cette cible. En binaire, ce nombre doit avoir plus de 60 bits de tête mis à zéro. Avec ce niveau de difficulté, un seul mineur traitant 1 billion de hachages par seconde (1 terahash par seconde ou 1 TH/sec) ne trouverait une solution qu'une fois tous les 8 496 blocs ou une fois tous les 59 jours, en moyenne.

### Reciblage pour ajuster la difficulté

Comme nous l'avons vu, la cible détermine la difficulté et affecte donc le temps qu'il faut pour trouver une solution au Proof-of- Algorithme de travail. Cela conduit à des questions évidentes : pourquoi la difficulté est-elle ajustable, qui l'ajuste et comment ?

Les blocs de Bitcoin sont générés toutes les 10 minutes, en moyenne. C'est le battement de cœur de bitcoin et sous-tend la fréquence d'émission de devises et la rapidité de règlement des transactions. Il doit rester constant non seulement à court terme, mais sur une période de plusieurs décennies. Au cours de cette période, on s'attend à ce que la puissance des ordinateurs continue d'augmenter à un rythme rapide. De plus, le nombre de participants à l'exploitation minière et les ordinateurs qu'ils utilisent changeront également constamment. Pour maintenir le temps de génération de bloc à 10 minutes, la difficulté de minage doit être ajustée pour tenir compte de ces changements. En fait, la cible de preuve de travail est un paramètre dynamique qui est périodiquement ajusté pour atteindre un objectif d'intervalle de bloc de 10 minutes. En termes simples, l'objectif est défini de sorte que la puissance minière actuelle se traduise par un intervalle de bloc de 10 minutes.

Comment, alors, un tel ajustement s'effectue-t-il dans un réseau totalement décentralisé ? Le reciblage se produit automatiquement et sur chaque nœud indépendamment. Tous les 2 016 blocs, tous les nœuds reciblent la preuve de travail. L'équation de reciblage mesure le temps qu'il a fallu pour trouver les 2 016 derniers blocs et le compare au temps prévu de 20 160 minutes (2 016 blocs multipliés par l'intervalle de bloc de 10 minutes souhaité). Le rapport entre la durée réelle et la durée souhaitée est calculé et un ajustement proportionnel (à la hausse ou à la baisse) est apporté à la cible. En termes simples : si le réseau trouve des blocs plus rapidement que toutes les 10 minutes, la difficulté augmente (la cible diminue). Si la découverte de blocs est plus lente que prévu, la difficulté diminue (la cible augmente).

L'équation peut se résumer ainsi :

Nouvel objectif = Ancien objectif \* (Heure réelle des 2016 derniers blocs / 20160 minutes)

<<retarget\_code> > montre le code utilisé dans le client Bitcoin Core.

// Limiter le pas de réglage  
 int64\_t nActualTimespan = pindexLast-&gt;GetBlockTime() - nFirstBlockTime ;  
 LogPrintf(&quot; nActualTimespan = %d avant les limites\n&quot;, nActualTimespan);  
 si (nActualTimespan &lt; params.nPowTargetTimespan/4)  
 nActualTimespan = params.nPowTargetTimespan/4 ;  
 si (nActualTimespan &gt; params.nPowTargetTimespan\*4)  
 nActualTimespan = params.nPowTargetTimespan\*4 ;  
  
 // Recibler  
 const arith\_uint256 bnPowLimit = UintToArith256(params.powLimit);  
 arith\_uint256 bnNew ;  
 arith\_uint256 bnOld ;  
 bnNew.SetCompact(pindexLast-&gt;nBits);  
 bnAncien = bnNouveau ;  
 bnNew \*= nActualTimespan ;  
 bnNew /= params.nPowTargetTimespan ;  
  
 si (bnNew &gt; bnPowLimit)  
 bnNouveau = bnPowLimit ;

Bien que l'étalonnage cible se produise tous les 2 016 blocs, en raison d'une erreur de un par un dans le client Bitcoin Core d'origine, il est basé sur le temps total des 2 015 blocs précédents (et non 2 016 comme il se doit), ce qui entraîne un biais de reciblage vers une difficulté plus élevée de 0,05 %.

Les paramètres Interval (2 016 blocs) et TargetTimespan (deux semaines en 1 209 600 secondes) sont définis dans *chainparams.cpp*.

Pour éviter une volatilité extrême de la difficulté, l'ajustement de reciblage doit être inférieur à un facteur quatre (4) par cycle. Si l'ajustement cible requis est supérieur à un facteur de quatre, il sera ajusté d'un facteur de 4 et pas plus. Tout autre ajustement sera effectué au cours de la prochaine période de reciblage, car le déséquilibre persistera au cours des 2 016 prochains blocs. Par conséquent, des écarts importants entre la puissance de hachage et la difficulté peuvent nécessiter plusieurs cycles de blocs de 2 016 pour s'équilibrer.

La difficulté d'extraction d'un bloc de bitcoins est d'environ '10 minutes de traitement' pour l'ensemble du réseau, sur la base du temps qu'il a fallu pour extraire les 2 016 blocs précédents, ajusté tous les 2 016 blocs. Ceci est réalisé en abaissant ou en élevant la cible.

Notez que la cible est indépendante du nombre de transactions ou de la valeur des transactions. Cela signifie que la quantité de puissance de hachage et donc d'électricité dépensée pour sécuriser le bitcoin est également entièrement indépendante du nombre de transactions. Bitcoin peut évoluer, parvenir à une adoption plus large et rester sécurisé sans aucune augmentation de la puissance de hachage par rapport au niveau actuel. L'augmentation de la puissance de hachage représente les forces du marché alors que de nouveaux mineurs entrent sur le marché pour concourir pour la récompense. Tant qu'une puissance de hachage suffisante est sous le contrôle des mineurs agissant honnêtement à la recherche de la récompense, cela suffit pour empêcher les attaques de "prise de contrôle" et, par conséquent, c'est suffisant pour sécuriser le bitcoin.

La difficulté du minage est étroitement liée au coût de l'électricité et au taux de change du bitcoin vis-à-vis de la monnaie utilisée pour payer l'électricité. Les systèmes de minage hautes performances sont à peu près aussi efficaces que possible avec la génération actuelle de fabrication de silicium, convertissant l'électricité en calcul de hachage au taux le plus élevé possible. La principale influence sur le marché minier est le prix d'un kilowattheure d'électricité en bitcoin, car cela détermine la rentabilité du minage et donc les incitations à entrer ou à sortir du marché minier.

## Réussir à miner le bloc

Comme nous l'avons vu précédemment, Jing's node a construit un bloc candidat et l'a préparé pour le minage. Jing possède plusieurs plates-formes de minage de matériel avec des circuits intégrés spécifiques à l'application, où des centaines de milliers de circuits intégrés exécutent l'algorithme SHA256 en parallèle à des vitesses incroyables. Beaucoup de ces machines spécialisées sont connectées à son nœud minier via USB ou un réseau local. Ensuite, le nœud de minage exécuté sur le bureau de Jing transmet l'en-tête de bloc à son matériel de minage, qui commence à tester des milliards de nonces par seconde. Parce que le nonce n'est que de 32 bits, après avoir épuisé toutes les possibilités de nonce (environ 4 milliards), le matériel de minage modifie l'en-tête de bloc (en ajustant l'espace ou l'horodatage supplémentaire du nonce coinbase) et réinitialise le compteur de nonce, en testant de nouvelles combinaisons.

Près de 11 minutes après avoir commencé à miner le bloc 277 316, l'une des machines de minage de matériel trouve une solution et la renvoie au nœud de minage.

Lorsqu'il est inséré dans l'en-tête de bloc, le nonce 924 591 752 produit un hachage de bloc de :

0000000000000001b6b9a13b095e96db41c4a928b97ef2d944a9b31b2cc7bdc4

qui est inférieur à la cible :

0000000000000003A30C000000000000000000000000000000000000000000

Immédiatement, le nœud minier de Jing transmet le bloc à tous ses pairs. Ils reçoivent, valident puis propagent le nouveau bloc. Au fur et à mesure que le bloc se répand sur le réseau, chaque nœud l'ajoute à sa propre copie de la blockchain, l'étendant à une nouvelle hauteur de 277 316 blocs. Au fur et à mesure que les nœuds miniers reçoivent et valident le bloc, ils abandonnent leurs efforts pour trouver un bloc à la même hauteur et commencent immédiatement à calculer le bloc suivant dans la chaîne, en utilisant le bloc de Jing comme "parent". En construisant sur le bloc nouvellement découvert de Jing, les autres mineurs "votent" essentiellement avec leur puissance minière et approuvent le bloc de Jing et la chaîne qu'il prolonge.

Dans la section suivante, nous examinerons le processus utilisé par chaque nœud pour valider un bloc et sélectionner la chaîne la plus longue, créant ainsi le consensus qui forme la blockchain décentralisée.("", startref="jingtentwo")

## Validation d'un nouveau bloc

La troisième étape du mécanisme de consensus de bitcoin est indépendante validation de chaque nouveau bloc par chaque nœud du réseau. Au fur et à mesure que le bloc nouvellement résolu se déplace sur le réseau, chaque nœud effectue une série de tests pour le valider avant de le propager à ses pairs. Cela garantit que seuls les blocs valides sont propagés sur le réseau. La validation indépendante garantit également que les mineurs qui agissent honnêtement obtiennent leurs blocs incorporés dans la blockchain, gagnant ainsi la récompense. Les mineurs qui agissent de manière malhonnête voient leurs blocs rejetés et non seulement perdent la récompense, mais gaspillent également les efforts déployés pour trouver une solution de preuve de travail, encourant ainsi le coût de l'électricité sans compensation.

Lorsqu'un nœud reçoit un nouveau bloc, il valide le bloc en le comparant à une longue liste de critères qui doivent tous être remplis ; sinon, le bloc est rejeté. Ces critères peuvent être vus dans le client Bitcoin Core dans les fonctions CheckBlock et CheckBlockHeader et incluent :

* La structure de données du bloc est syntaxiquement valide
* Le hachage de l'en-tête de bloc est égal ou inférieur à la cible (applique la preuve de travail)
* L'horodatage du bloc est inférieur à deux heures dans le futur (en tenant compte des erreurs de temps)
* La taille du bloc est dans les limites acceptables
* La première transaction (et seulement la première) est une transaction coinbase
* Toutes les transactions dans le bloc sont valides en utilisant la liste de contrôle des transactions décrite dans <<tx\_verification> >

La validation indépendante de chaque nouveau bloc par chaque nœud du réseau garantit que les mineurs ne peuvent pas tricher. Dans les sections précédentes, nous avons vu comment les mineurs peuvent écrire une transaction qui leur attribue le nouveau bitcoin créé dans le bloc et réclamer les frais de transaction. Pourquoi les mineurs n'écrivent-ils pas eux-mêmes une transaction pour mille bitcoins au lieu de la bonne récompense ? Parce que chaque nœud valide les blocs selon les mêmes règles. Une transaction coinbase invalide rendrait l'ensemble du bloc invalide, ce qui entraînerait le rejet du bloc et, par conséquent, cette transaction ne ferait jamais partie du grand livre. Les mineurs doivent construire un bloc parfait, basé sur les règles partagées que tous les nœuds suivent, et l'exploiter avec une solution correcte à la preuve de travail. Pour ce faire, ils dépensent beaucoup d'électricité dans l'exploitation minière, et s'ils trichent, toute l'électricité et les efforts sont gaspillés. C'est pourquoi la validation indépendante est un élément clé du consensus décentralisé.

## Assemblage et sélection de chaînes de blocs

Le La dernière étape du mécanisme de consensus décentralisé de Bitcoin est l'assemblage de blocs en chaînes et la sélection de la chaîne avec le plus de preuves de travail. Une fois qu'un nœud a validé un nouveau bloc, il tentera alors d'assembler une chaîne en connectant le bloc à la blockchain existante.

Les nœuds maintiennent trois ensembles de blocs : ceux connectés à la blockchain principale, ceux qui forment des branches de la blockchain principale (chaînes secondaires) et enfin, les blocs qui n'ont pas de parent connu dans les chaînes connues (orphelins). Les blocs invalides sont rejetés dès que l'un des critères de validation échoue et ne sont donc inclus dans aucune chaîne.

La "chaîne principale" à tout moment est la chaîne de blocs *valide* à laquelle est associée la preuve de travail la plus cumulative. Dans la plupart des cas, il s'agit également de la chaîne contenant le plus de blocs, à moins qu'il n'y ait deux chaînes de longueur égale et que l'une ait plus de preuve de travail. La chaîne principale aura également des branches avec des blocs qui sont des "frères et sœurs" des blocs de la chaîne principale. Ces blocs sont valides mais ne font pas partie de la chaîne principale. Elles sont conservées pour référence future, au cas où l'une de ces chaînes serait étendue au-delà de la chaîne principale en cours. Dans la section suivante (<<forks> >), nous verrons comment des chaînes secondaires se produisent à la suite d'une extraction quasi simultanée de blocs à la même hauteur.

Lorsqu'un nouveau bloc est reçu, un nœud essaie de l'insérer dans la blockchain existante. Le nœud examinera le champ "hachage du bloc précédent" du bloc, qui est la référence au parent du bloc. Ensuite, le nœud tentera de trouver ce parent dans la blockchain existante. La plupart du temps, le parent sera la "pointe" de la chaîne principale, ce qui signifie que ce nouveau bloc étend la chaîne principale. Par exemple, le nouveau bloc 277,316 a une référence au hachage de son bloc parent 277,315. La plupart des nœuds qui reçoivent 277 316 auront déjà le bloc 277 315 comme pointe de leur chaîne principale et relieront donc le nouveau bloc et étendront cette chaîne.

Parfois, comme nous le verrons dans <<forks> >, le nouveau bloc prolonge une chaîne qui n'est pas la chaîne principale. Dans ce cas, le nœud attachera le nouveau bloc à la chaîne secondaire qu'il prolonge, puis comparera le travail de la chaîne secondaire à la chaîne principale. Si la chaîne secondaire a plus de travail cumulé que la chaîne principale, le nœud *reconvergera* sur la chaîne secondaire, ce qui signifie qu'il sélectionnera la chaîne secondaire comme sa nouvelle chaîne principale, faisant de l'ancienne chaîne principale une chaîne secondaire. Si le nœud est un mineur, il construira maintenant un bloc étendant cette nouvelle chaîne plus longue.

Si un bloc valide est reçu et qu'aucun parent n'est trouvé dans les chaînes existantes, ce bloc est considéré comme "orphelin". Les blocs orphelins sont enregistrés dans le pool de blocs orphelins où ils resteront jusqu'à ce que leur parent soit reçu. Une fois que le parent est reçu et lié aux chaînes existantes, l'orphelin peut être retiré du pool d'orphelins et lié au parent, ce qui en fait une partie d'une chaîne. Les blocs orphelins se produisent généralement lorsque deux blocs extraits dans un court laps de temps sont reçus dans l'ordre inverse (enfant avant parent).

En sélectionnant la chaîne valide de plus grand travail cumulé, tous les nœuds finissent par parvenir à un consensus à l'échelle du réseau. Les divergences temporaires entre les chaînes sont finalement résolues au fur et à mesure que du travail est ajouté, étendant l'une des chaînes possibles. Les nœuds de minage « votent » avec leur puissance de minage en choisissant quelle chaîne étendre en minant le bloc suivant. Lorsqu'ils exploitent un nouveau bloc et étendent la chaîne, le nouveau bloc lui-même représente leur vote.

Dans la section suivante, nous verrons comment les divergences entre les chaînes concurrentes (forks) sont résolues par la sélection indépendante de la chaîne de plus grand travail cumulatif.

### Fourchettes Blockchain

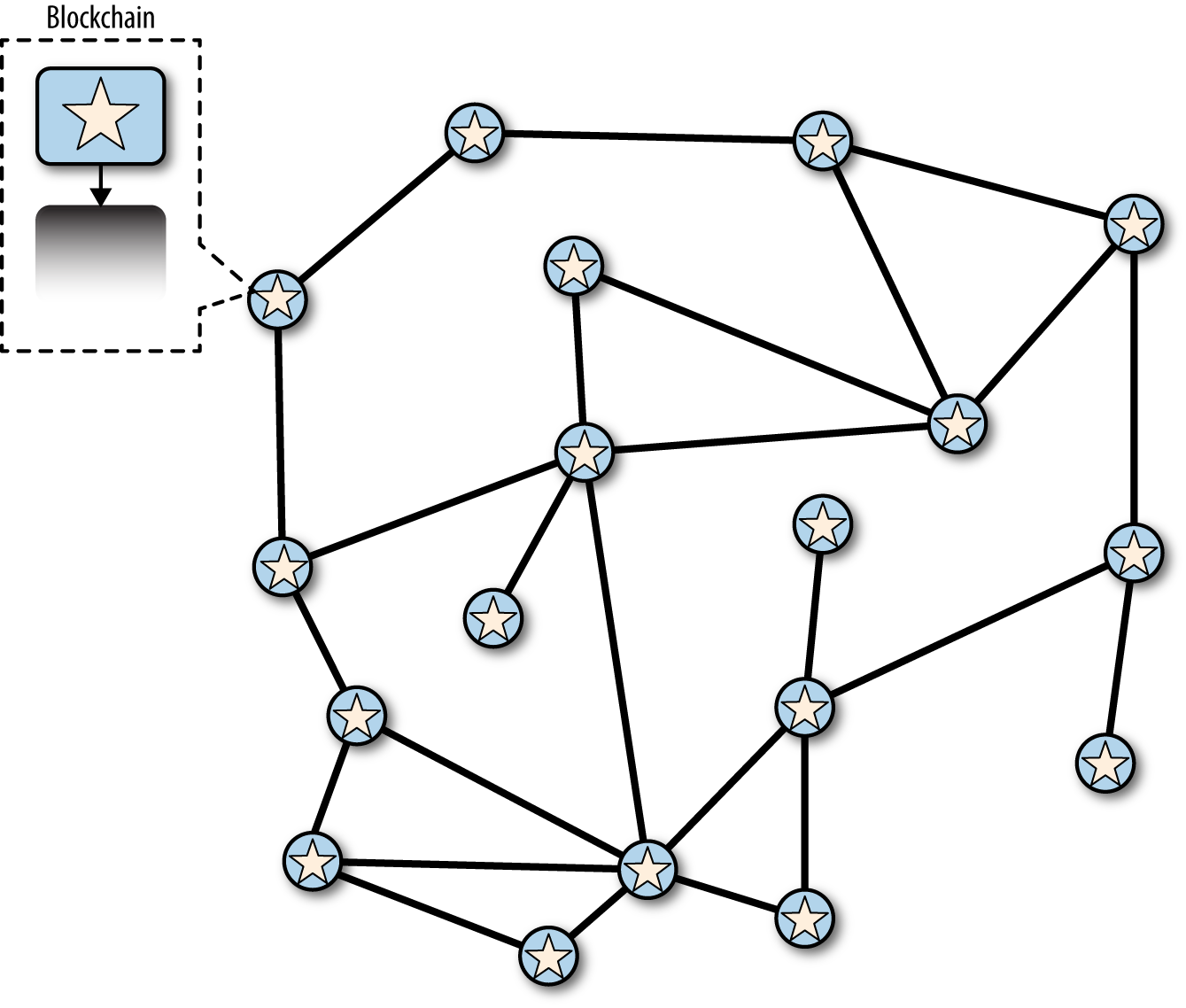
("forks", "blockchain fork events", id="forks10")Parce que la blockchain est une structure de données décentralisée, différentes copies de celle-ci ne sont pas toujours cohérentes. Les blocs peuvent arriver à différents nœuds à des moments différents, ce qui amène les nœuds à avoir différentes perspectives de la blockchain. Pour résoudre ce problème, chaque nœud sélectionne et tente toujours d'étendre la chaîne de blocs qui représente le plus de preuve de travail, également connue sous le nom de chaîne la plus longue ou de chaîne de travail cumulative la plus importante. En additionnant le travail enregistré dans chaque bloc d'une chaîne, un nœud peut calculer la quantité totale de travail qui a été dépensée pour créer cette chaîne. Tant que tous les nœuds sélectionnent la plus grande chaîne de travail cumulatif, le réseau mondial Bitcoin finit par converger vers un état cohérent. Les fourches se produisent sous forme d'incohérences temporaires entre les versions de la blockchain, qui sont résolues par une éventuelle reconvergence à mesure que d'autres blocs sont ajoutés à l'une des fourches.

Les fourches de blockchain décrites dans cette section se produisent naturellement (accidentellement) en raison de retards de transmission dans le réseau mondial. Plus loin dans ce chapitre, nous verrons également les forks délibérément induits (hard forks et soft forks), qui sont utilisés pour modifier les règles de consensus.

Dans les quelques diagrammes suivants, nous suivons la progression d'un événement "fork" à travers le réseau. Le diagramme est une représentation simplifiée du réseau Bitcoin. À des fins d'illustration, différents blocs sont représentés sous différentes formes (étoile, triangle, triangle inversé, losange), répartis sur le réseau. Chaque nœud du réseau est représenté par un cercle.

Chaque nœud a sa propre perspective de la blockchain mondiale. Au fur et à mesure que chaque nœud reçoit des blocs de ses voisins, il met à jour sa propre copie de la blockchain, en sélectionnant la plus grande chaîne de travail cumulée. À des fins d'illustration, chaque nœud contient une forme qui représente le bloc qui, selon lui, est actuellement la pointe de la chaîne principale. Donc, si vous voyez une forme d'étoile dans le nœud, cela signifie que le bloc en étoile est la pointe de la chaîne principale, en ce qui concerne ce nœud.

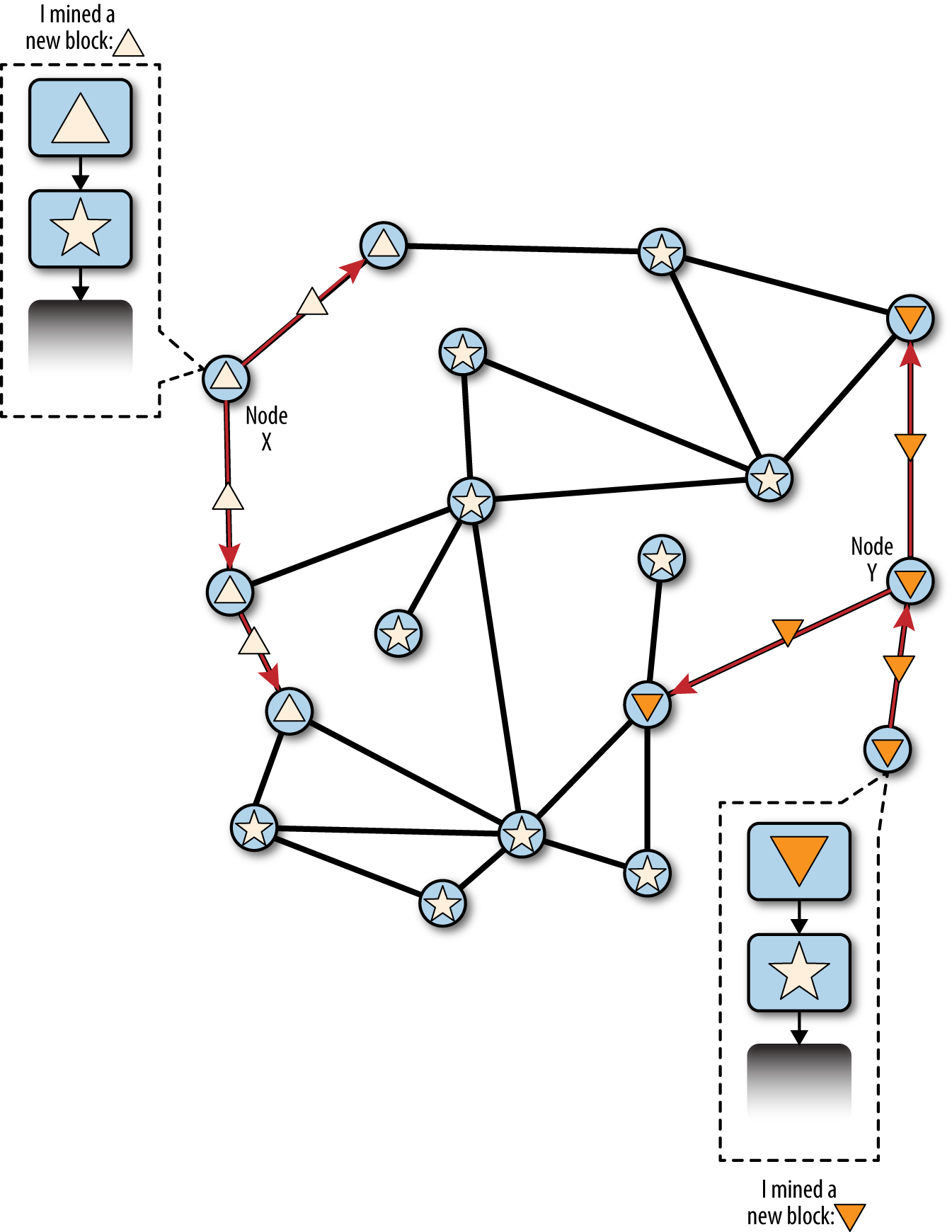
Dans le premier diagramme (<<fork1> >), le réseau a une perspective unifiée de la blockchain, avec le bloc étoile comme pointe de la chaîne principale.



Avant la bifurcation—tous les nœuds ont la même perspective

Un "fork" se produit chaque fois qu'il y a deux blocs valides différents à la même hauteur de bloc en compétition pour former la blockchain la plus longue. Cela se produit dans des conditions normales chaque fois que deux mineurs résolvent l'algorithme de preuve de travail dans un court laps de temps l'un de l'autre. Alors que les deux mineurs découvrent une solution pour leurs blocs candidats respectifs, ils diffusent immédiatement leur propre bloc "gagnant" à leurs voisins immédiats qui commencent à propager le bloc sur le réseau. Chaque nœud qui reçoit un bloc valide l'incorpore dans sa blockchain, étendant la blockchain d'un bloc. Si ce nœud voit plus tard un autre bloc valide étendre le même parent (à la même hauteur de bloc), il connecte le deuxième bloc sur une chaîne secondaire, bifurquant sa chaîne principale. En conséquence, certains nœuds "verront" un bloc gagnant en premier, tandis que d'autres nœuds verront l'autre bloc gagnant en premier, et deux versions concurrentes de la blockchain émergeront.

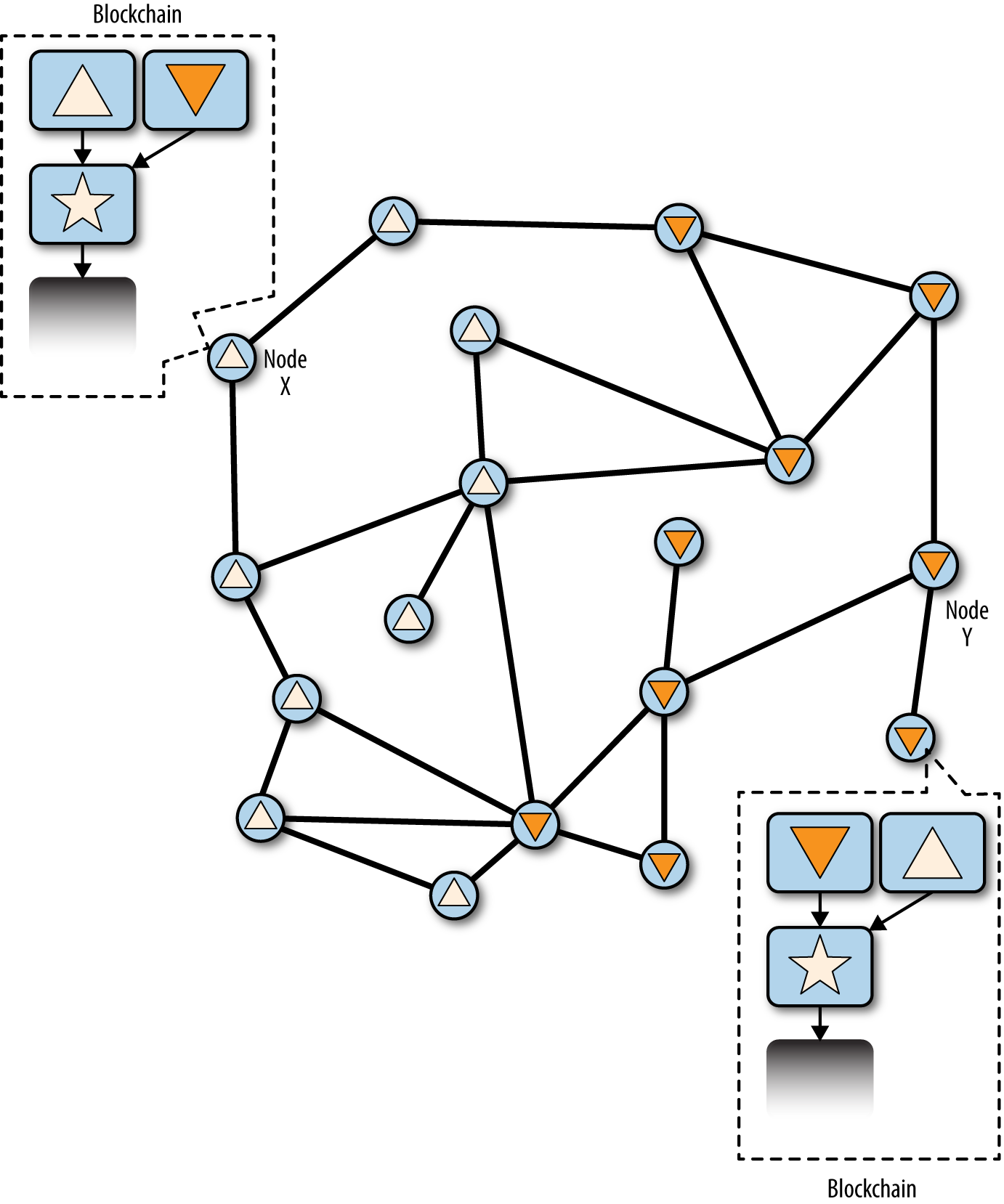
Dans <<fork2> >, on voit deux mineurs (Node X et Node Y) qui minent deux blocs différents presque simultanément. Ces deux blocs sont des enfants du bloc étoile et étendent la chaîne en s'appuyant sur le bloc étoile. Pour nous aider à le suivre, l'un est visualisé comme un bloc triangulaire provenant du nœud X, et l'autre est représenté comme un bloc triangulaire inversé provenant du nœud Y.



Visualisation d'un événement fork blockchain : deux blocs trouvés simultanément

Supposons, par exemple, que le mineur Node X trouve une solution de preuve de travail pour un bloc "triangle" qui étend la blockchain, en s'appuyant sur le bloc parent "étoile". Presque simultanément, le mineur Node Y qui étendait également la chaîne du bloc "étoile" trouve une solution pour le bloc "triangle inversé", son bloc candidat. Maintenant, il y a deux blocs possibles ; un que nous appelons "triangle", originaire du Nœud X ; et un que nous appelons "triangle inversé", originaire du nœud Y. Les deux blocs ont été extraits avec succès, les deux blocs sont valides (contiennent une solution valide à la preuve de travail), et les deux blocs étendent le même parent (bloc " Star"). Les deux blocs contiennent probablement la plupart des mêmes transactions, avec seulement peut-être quelques différences dans l'ordre des transactions.

Au fur et à mesure que les deux blocs se propagent, certains nœuds reçoivent d'abord le bloc "triangle" et d'autres reçoivent le bloc "triangle inversé" en premier. Comme indiqué dans <<fork3> >, le réseau se scinde en deux perspectives différentes de la blockchain ; un côté surmonté du bloc triangle, l'autre du bloc triangle renversé.



Visualization of a blockchain fork event: two blocks propagate, splitting the network

In [figure\_title](#fork3), the miner Node X mined (created) the triangle block and extended the star chain with it. Therefore, Node X initially considers the chain with "triangle" block as the main chain. Later, Node X also received the "upside-down triangle" block that was mined by Node Y. Since it was received second, it is assumed to have "lost" the race. Yet, the "upside-down triangle" block is not discarded. It is linked to the "star" block parent and forms a secondary chain. While Node X assumes its main chain is the winning chain, it keeps the "losing" chain so that it has the information needed to reconverge if the "losing" chain ends up "winning."

On the other side of the network, the miner Node Y constructs a blockchain based on its own perspective of the sequence of events. The miner Node Y mined (created) the "upside-down triangle" and initially considers that chain as the main chain (the "winner" chain). When it later received the "triangle" block that was mined by Node X, it connected it to the "star" block parent as a secondary chain.

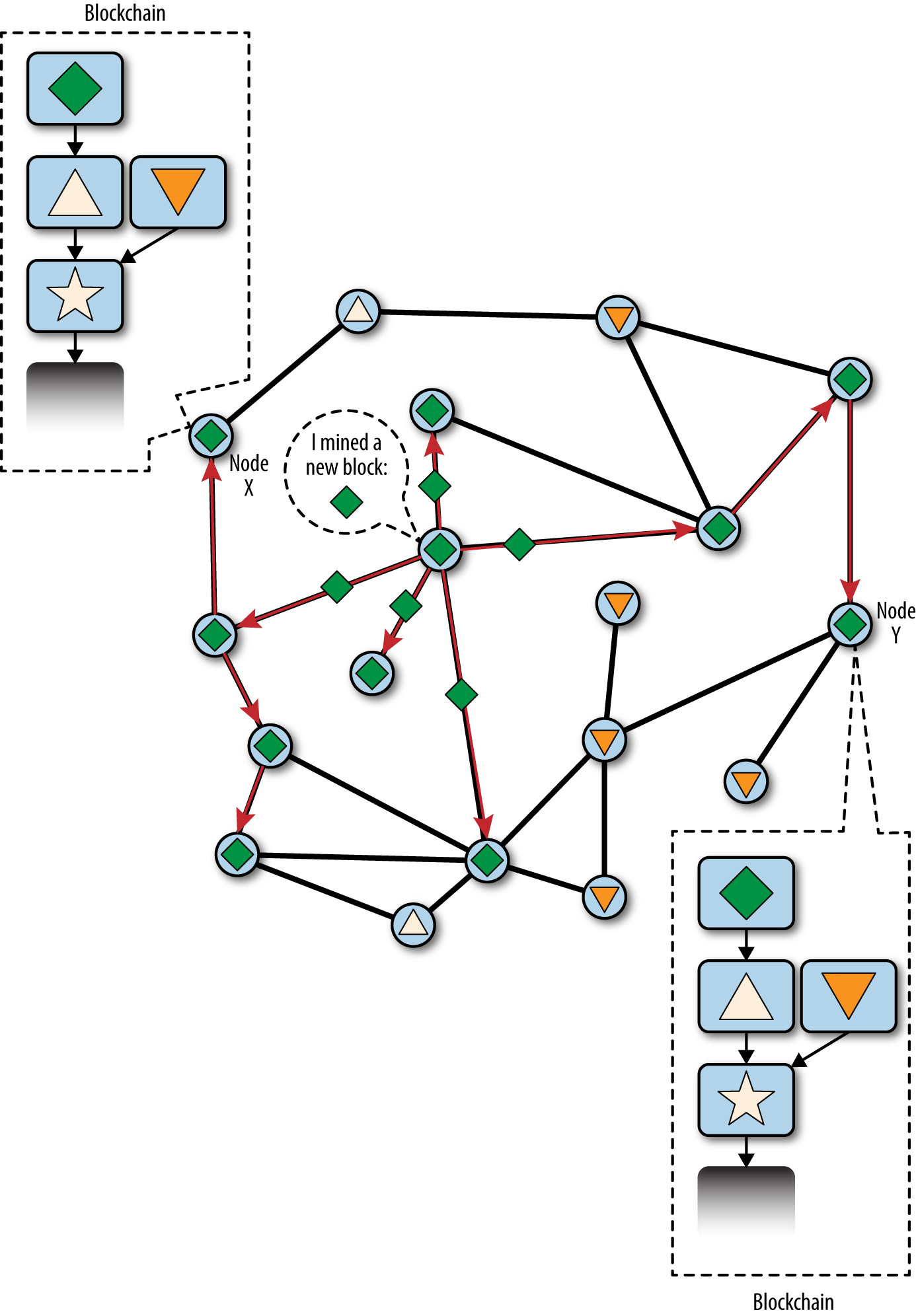
Neither side is "correct," or "incorrect." Both are valid perspectives of the blockchain. Only in hindsight will one prevail, based on how these two competing chains are extended by additional work.

Each mining node whose perspective resembles Node X will immediately begin mining a candidate block that extends the chain with "triangle" as its tip. By linking "triangle" as the parent of their candidate blocks, they are voting with their hashing power. Their vote supports the chain that they have elected as the main chain.

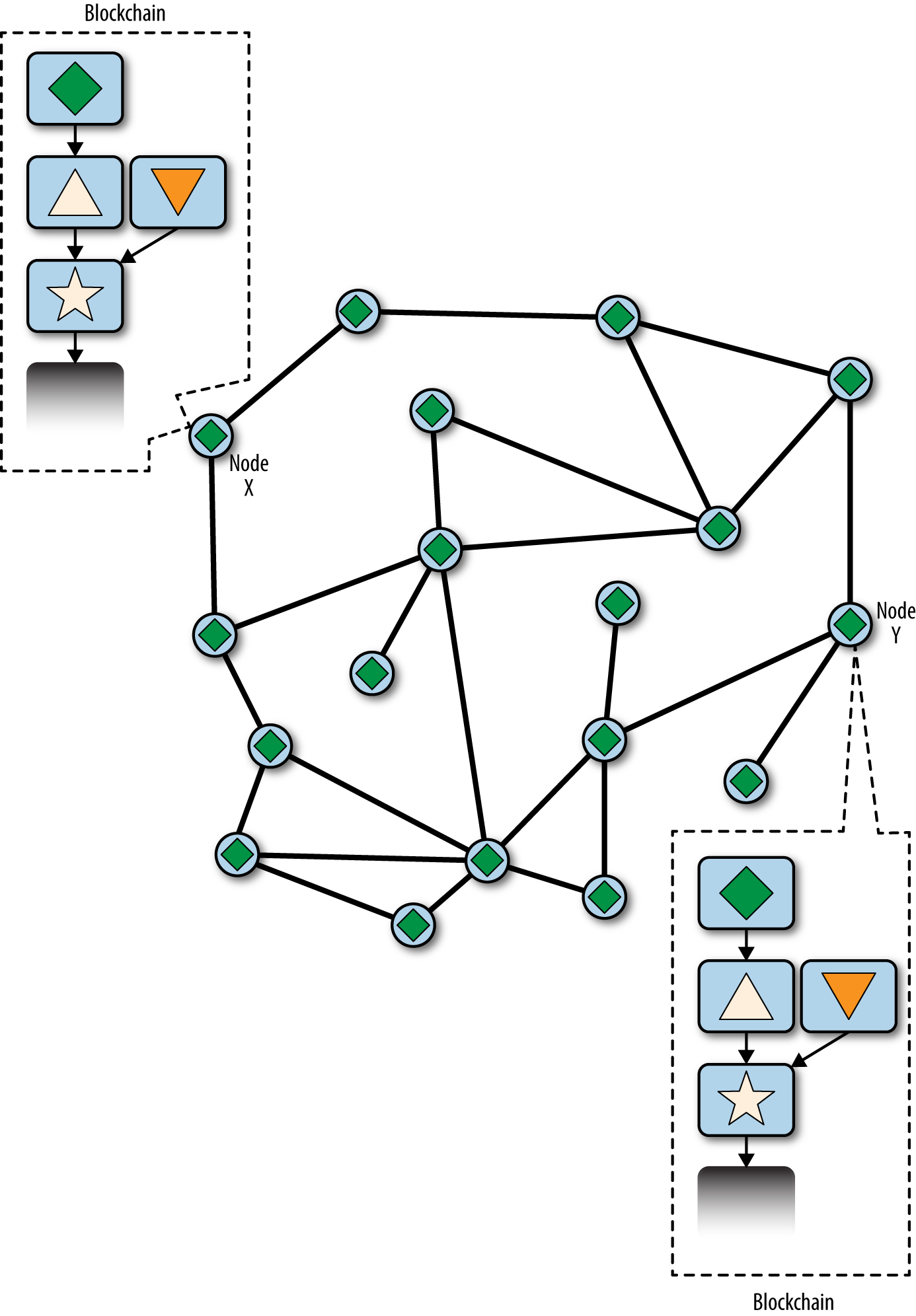
Any mining node whose perspective resembles Node Y will start building a candidate block with "upside-down triangle" as its parent, extending the chain that they believe is the main chain. And so, the race begins again.

Forks are almost always resolved within one block time (10 minutes on average). While part of the network’s hashing power is dedicated to building on top of "triangle" as the parent, another part of the hashing power is focused on building on top of "upside-down triangle." Even if the hashing power is almost evenly split, it is likely that one set of miners will find a solution and propagate it before the other set of miners have found any solutions. Let’s say, for example, that the miners building on top of "triangle" find a new block "rhombus" that extends the chain (e.g., star-triangle-rhombus). They immediately propagate this new block and the entire network sees it as a valid solution as shown in [figure\_title](#fork4).

All nodes that had chosen "triangle" as the winner in the previous round will simply extend the chain one more block. The nodes that chose "upside-down triangle" as the winner, however, will now see two chains: star-triangle-rhombus and star-upside-down-triangle. The chain star-triangle-rhombus is now longer (more cumulative work) than the other chain. As a result, those nodes will set the chain star-triangle-rhombus as the main chain and change the star-upside-down-triangle chain to a secondary chain, as shown in [figure\_title](#fork5). This is a chain reconvergence, because those nodes are forced to revise their view of the blockchain to incorporate the new evidence of a longer chain. Any miners working on extending the chain star-upside-down-triangle will now stop that work because their candidate block is now considered a child of a stale block, as its parent "upside-down-triangle" is no longer on the longest chain. Since the upside-down-triangle block is now obsolete, the miner Node Y (which mined this block) will not be able to spend the mining reward for this block, even though this block was valid and was successfully mined. The transactions within "upside-down-triangle" that are not within "triangle" are re-inserted in the mempool for inclusion in the next block to become a part of the main chain. The entire network reconverges on a single blockchain star-triangle-rhombus, with "rhombus" as the last block in the chain. All miners immediately start working on candidate blocks that reference "rhombus" as their parent to extend the star-triangle-rhombus chain.



Visualization of a blockchain fork event: a new block extends one fork, reconverging the network. Both Node X and Node Y now consider "upside-down-triangle" block as a stale block.



Visualization of a blockchain fork event: the network reconverges on a new longest chain

It is theoretically possible for a fork to extend to two blocks, if two blocks are found almost simultaneously by miners on opposite "sides" of a previous fork. However, the chance of that happening is very low. Whereas a one-block fork might occur every day, a two-block fork occurs at most once every few weeks.

Bitcoin’s block interval of 10 minutes is a design compromise between fast confirmation times (settlement of transactions) and the probability of a fork. A faster block time would make transactions clear faster but lead to more frequent blockchain forks, whereas a slower block time would decrease the number of forks but make settlement slower.

## Mining and the Hashing Race

Bitcoin mining is an extremely competitive industry. The hashing power has increased exponentially every year of bitcoin’s existence. Some years the growth has reflected a complete change of technology, such as in 2010 and 2011 when many miners switched from using CPU mining to GPU mining and field programmable gate array (FPGA) mining. In 2013 the introduction of ASIC mining lead to another giant leap in mining power, by placing the SHA256 function directly on silicon chips specialized for the purpose of mining. The first such chips could deliver more mining power in a single box than the entire Bitcoin network in 2010.

The following list shows the total hashing power of the Bitcoin network in terahashes/sec (TH/sec), since its inception in 2009 (source: Blockchain.com):

2009

0.000004 – 0.00001 TH/sec (2.40× growth)

2010

0.00001 – 0.14 TH/sec (14,247× growth)

2011

0.14 – 9.49 TH/sec (63.92× growth)

2012

9.49 – 22 TH/sec (2.32× growth)

2013

22.04 – 15,942 TH/sec (723.32× growth)

2014

15,942 – 306,333 TH/sec (19.21× growth)

2015

306,333 – 881,232 TH/sec (2.87× growth)

2016

881,232 – 2,807,540 TH/sec (3.18× growth)

2017

2,807,540 – 18,206,558 TH/sec (6.48× growth)

2018

18,206,558 – 41,801,528 TH/sec (2.29× growth)

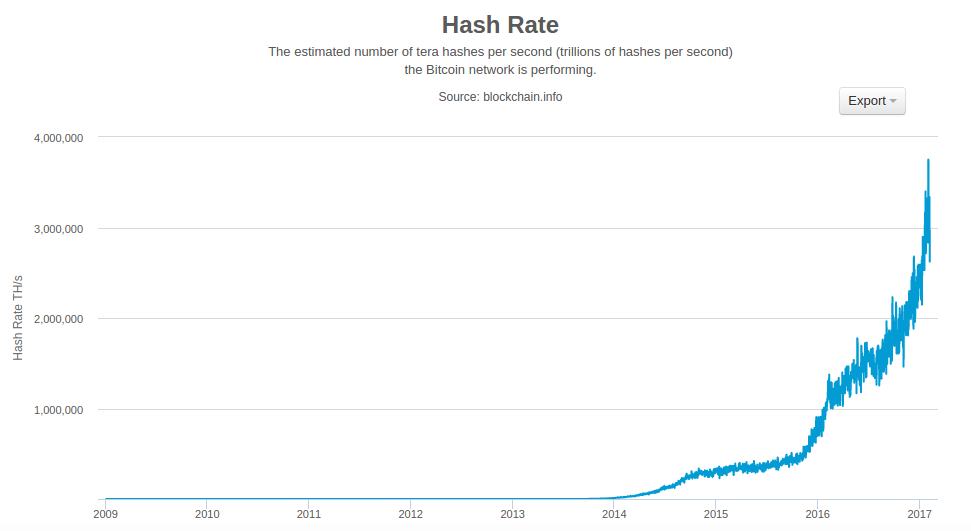
2019

41,801,528 – 109,757,127 TH/sec (2.62× growth)

2020

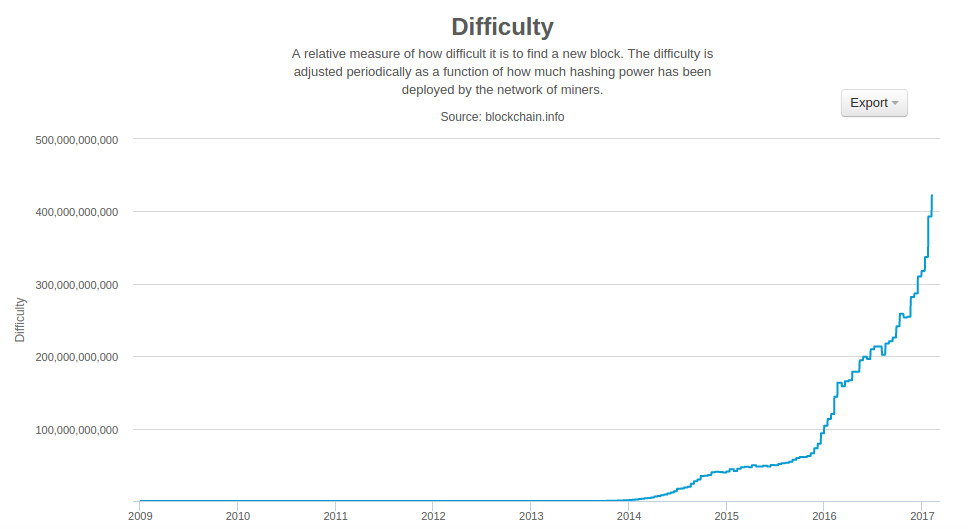
109,757,127 – 149,064,869 TH/sec (1.35× growth)

In the chart in [figure\_title](#network_hashing_power), we can see that Bitcoin network’s hashing power increased over the past two years. As you can see, the competition between miners and the growth of bitcoin has resulted in an exponential increase in the hashing power (total hashes per second across the network).



Total hashing power, terahashes per second (TH/sec) (chart on a linear scale)

As the amount of hashing power applied to mining bitcoin has exploded, the difficulty has risen to match it. The difficulty metric in the chart shown in [figure\_title](#bitcoin_difficulty) is measured as a ratio of current difficulty over minimum difficulty (the difficulty of the first block).



Bitcoin’s mining difficulty metric (chart on a logarithmic scale)

In the last two years, the ASIC mining chips have become increasingly denser, approaching the cutting edge of silicon fabrication with a feature size (resolution) of 7 nanometers (nm). Currently, ASIC manufacturers are aiming to overtake general-purpose CPU chip manufacturers, designing chips with a feature size of 5 nm, because the profitability of mining is driving this industry even faster than general computing. There are no more giant leaps left in bitcoin mining, because the industry has reached the forefront of Moore’s Law, which stipulates that computing density will double approximately every 18 months. Still, the mining power of the network continues to advance at an exponential pace as the race for higher density chips is matched with a race for higher density data centers where thousands of these chips can be deployed. It’s no longer about how much mining can be done with one chip, but how many chips can be squeezed into a building, while still dissipating the heat and providing adequate power.

### The Extra Nonce Solution

Since 2012, bitcoin mining has evolved to resolve a fundamental limitation in the structure of the block header. In the early days of bitcoin, a miner could find a block by iterating through the nonce until the resulting hash was equal to or below the target. As difficulty increased, miners often cycled through all 4 billion values of the nonce without finding a block. However, this was easily resolved by updating the block timestamp to account for the elapsed time. Because the timestamp is part of the header, the change would allow miners to iterate through the values of the nonce again with different results. Once mining hardware exceeded 4 GH/sec, however, this approach became increasingly difficult because the nonce values were exhausted in less than a second. As ASIC mining equipment started pushing and then exceeding the TH/sec hash rate, the mining software needed more space for nonce values in order to find valid blocks. The timestamp could be stretched a bit, but moving it too far into the future would cause the block to become invalid. A new source of "change" was needed in the block header. The solution was to use the coinbase transaction as a source of extra nonce values. Because the coinbase script can store between 2 and 100 bytes of data, miners started using that space as extra nonce space, allowing them to explore a much larger range of block header values to find valid blocks. The coinbase transaction is included in the merkle tree, which means that any change in the coinbase script causes the merkle root to change. Eight bytes of extra nonce, plus the 4 bytes of "standard" nonce allow miners to explore a total 296 (8 followed by 28 zeros) possibilities *per second* without having to modify the timestamp. If, in the future, miners could run through all these possibilities, they could then modify the timestamp. There is also more space in the coinbase script for future expansion of the extra nonce space.

### Mining Pools

In this highly competitive environment, individual miners working alone (also known as solo miners) don’t stand a chance. The likelihood of them finding a block to offset their electricity and hardware costs is so low that it represents a gamble, like playing the lottery. Even the fastest consumer ASIC mining system cannot keep up with commercial systems that stack tens of thousands of these chips in giant warehouses near hydroelectric powerstations. Miners now collaborate to form mining pools, pooling their hashing power and sharing the reward among thousands of participants. By participating in a pool, miners get a smaller share of the overall reward, but typically get rewarded every day, reducing uncertainty.

Let’s look at a specific example. Assume a miner has purchased mining hardware with a combined hashing rate of 14,000 gigahashes per second (GH/s), or 14 TH/s. In 2017 this equipment costs approximately $2,500 USD. The hardware consumes 1375 watts (1.3 kW) of electricity when running, 33 kW-hours a day, at a cost of $1 to $2 per day at very low electricity rates. At current bitcoin difficulty, the miner will be able to solo mine a block approximately once every 4 years. How do we work out that probability? It is based on a network-wide hashing rate of 3 EH/sec (in 2017), and the miner’s rate of 14 TH/sec:

P = (14 \* 10

12

/ 3 \* 10

18

) \* 210240 = 0.98

…​where 210240 is the number of blocks in four years. The miner has a 98% probability of finding a block over four years, based on the global hash rate at the beginning of the period.

If the miner does find a single block in that timeframe, the payout of 6.25 bitcoin, at approximately $1,000 per bitcoin, will result in a single payout of $6,250, which will produce a net profit of about $750. However, the chance of finding a block in a 4-year period depends on the miner’s luck. He might find two blocks in 4 years and make a larger profit. Or he might not find a block for 5 years and suffer a big financial loss. Even worse, the difficulty of the bitcoin Proof-of-Work algorithm is likely to go up significantly over that period, at the current rate of growth of hashing power, meaning the miner has, at most, one year to break even before the hardware is effectively obsolete and must be replaced by more powerful mining hardware. Financially this only makes sense at very low electricity cost (less than 1 cent per kW-hour) and only at very large scale.

Mining pools coordinate many hundreds or thousands of miners, over specialized pool-mining protocols. The individual miners configure their mining equipment to connect to a pool server, and specify a Bitcoin address, which will receive their share of the rewards. Their mining hardware remains connected to the pool server while mining, synchronizing their efforts with the other miners. Thus, the pool miners share the effort to mine a block and then share in the rewards.

Successful blocks pay the reward to a pool Bitcoin address, rather than individual miners. The pool server will periodically make payments to the miners' Bitcoin addresses, once their share of the rewards has reached a certain threshold. Typically, the pool server charges a percentage fee of the rewards for providing the pool-mining service.

Miners participating in a pool split the work of searching for a solution to a candidate block, earning "shares" for their mining contribution. The mining pool sets a higher target (lower difficulty) for earning a share, typically more than 1,000 times easier than the Bitcoin network’s target. When someone in the pool successfully mines a block, the reward is earned by the pool and then shared with all miners in proportion to the number of shares they contributed to the effort.

Pools are open to any miner, big or small, professional or amateur. A pool will therefore have some participants with a single small mining machine, and others with a garage full of high-end mining hardware. Some will be mining with a few tens of a kilowatt of electricity, others will be running a data center consuming a megawatt of power. How does a mining pool measure the individual contributions, so as to fairly distribute the rewards, without the possibility of cheating? The answer is to use bitcoin’s Proof-of-Work algorithm to measure each pool miner’s contribution, but set at a lower difficulty so that even the smallest pool miners win a share frequently enough to make it worthwhile to contribute to the pool. By setting a lower difficulty for earning shares, the pool measures the amount of work done by each miner. Each time a pool miner finds a block header hash that is equal to or less than the pool target, she proves she has done the hashing work to find that result. More importantly, the work to find shares contributes, in a statistically measurable way, to the overall effort to find a hash equal to or lower than the Bitcoin network’s target. Thousands of miners trying to find low-value hashes will eventually find one low enough to satisfy the Bitcoin network target.

Let’s return to the analogy of a dice game. If the dice players are throwing dice with a goal of throwing equal to or less than four (the overall network difficulty), a pool would set an easier target, counting how many times the pool players managed to throw equal to or less than eight. When pool players throw equal to or less than eight (the pool share target) but higher than four (higher than the overall network difficulty), they earn shares, but neither they nor the pool win the game because they don’t achieve the game target (equal to or less than four). The pool players will achieve the easier pool target much more often, earning them shares very regularly, even when they don’t achieve the harder target of winning the game. Every now and then, one of the pool players will throw a combined dice throw of equal to or less than four, the pool player wins a share and the whole pool wins the game. Then, the earnings can be distributed to the pool players based on the amount of shares each one has earned. Even though the target of eight-or-less wasn’t winning, it was a fair way to measure dice throws for the players, and it occasionally produces a four-or-less throw.

Similarly, a mining pool will set a (higher and easier) pool target that will ensure that an individual pool miner can find block header hashes that are equal to or less than the pool target often, earning shares. Every now and then, one of these attempts will produce a block header hash that is equal to or less than the Bitcoin network target, making it a valid block and the whole pool wins.

#### Managed pools

Most mining pools are "managed," meaning that there is a company or individual running a pool server. The owner of the pool server is called the *pool operator*, and he charges pool miners a percentage fee of the earnings.

The pool server runs specialized software and a pool-mining protocol that coordinate the activities of the pool miners. The pool server is also connected to one or more full Bitcoin nodes and has direct access to a full copy of the blockchain database. This allows the pool server to validate blocks and transactions on behalf of the pool miners, relieving them of the burden of running a full node. For pool miners, this is an important consideration, because a full node requires a dedicated computer with at least 300 to 350 GB of persistent storage (disk) and at least 2 to 4 GB of memory (RAM). Furthermore, the bitcoin software running on the full node needs to be monitored, maintained, and upgraded frequently. Any downtime caused by a lack of maintenance or lack of resources will hurt the miner’s profitability. For many miners, the ability to mine without running a full node is another big benefit of joining a managed pool.

Pool miners connect to the pool server using a mining protocol such as Stratum (STM) or GetBlockTemplate (GBT). An older standard called GetWork (GWK) has been mostly obsolete since late 2012, because it does not easily support mining at hash rates above 4 GH/s. Both the STM and GBT protocols create block *templates* that contain a template of a candidate block header. The pool server constructs a candidate block by aggregating transactions, adding a coinbase transaction (with extra nonce space), calculating the merkle root, and linking to the previous block hash. The header of the candidate block is then sent to each of the pool miners as a template. Each pool miner then mines using the block template, at a higher (easier) target than the Bitcoin network target, and sends any successful results back to the pool server to earn shares.

#### Peer-to-peer mining pool (P2Pool)

Managed pools create the possibility of cheating by the pool operator, who might direct the pool effort to double-spend transactions or invalidate blocks (see [Consensus Attacks](#consensus_attacks)). Furthermore, centralized pool servers represent a single-point-of-failure. If the pool server is down or is slowed by a denial-of-service attack, the pool miners cannot mine. In 2011, to resolve these issues of centralization, a new pool mining method was proposed and implemented: P2Pool, a peer-to-peer mining pool without a central operator.

P2Pool works by decentralizing the functions of the pool server, implementing a parallel blockchain-like system called a *share chain*. A share chain is a blockchain running at a lower difficulty than the Bitcoin blockchain. The share chain allows pool miners to collaborate in a decentralized pool by mining shares on the share chain at a rate of one share block every 30 seconds. Each of the blocks on the share chain records a proportionate share reward for the pool miners who contribute work, carrying the shares forward from the previous share block. When one of the share blocks also achieves the Bitcoin network target, it is propagated and included on the Bitcoin blockchain, rewarding all the pool miners who contributed to all the shares that preceded the winning share block. Essentially, instead of a pool server keeping track of pool miner shares and rewards, the share chain allows all pool miners to keep track of all shares using a decentralized consensus mechanism like bitcoin’s blockchain consensus mechanism.

P2Pool mining is more complex than pool mining because it requires that the pool miners run a dedicated computer with enough disk space, memory, and internet bandwidth to support a full Bitcoin node and the P2Pool node software. P2Pool miners connect their mining hardware to their local P2Pool node, which simulates the functions of a pool server by sending block templates to the mining hardware. On P2Pool, individual pool miners construct their own candidate blocks, aggregating transactions much like solo miners, but then mine collaboratively on the share chain. P2Pool is a hybrid approach that has the advantage of much more granular payouts than solo mining, but without giving too much control to a pool operator like managed pools.

Even though P2Pool reduces the concentration of power by mining pool operators, it is conceivably vulnerable to 51% attacks against the share chain itself. A much broader adoption of P2Pool does not solve the 51% attack problem for bitcoin itself. Rather, P2Pool makes bitcoin more robust overall, as part of a diversified mining ecosystem.

## Consensus Attacks

Bitcoin’s consensus mechanism is, at least theoretically, vulnerable to attack by miners (or pools) that attempt to use their hashing power to dishonest or destructive ends. As we saw, the consensus mechanism depends on having a majority of the miners acting honestly out of self-interest. However, if a miner or group of miners can achieve a significant share of the mining power, they can attack the consensus mechanism so as to disrupt the security and availability of the Bitcoin network.

It is important to note that consensus attacks can only affect future consensus, or at best, the most recent past (tens of blocks). Bitcoin’s ledger becomes more and more immutable as time passes. While in theory, a fork can be achieved at any depth, in practice, the computing power needed to force a very deep fork is immense, making old blocks practically immutable. Consensus attacks also do not affect the security of the private keys and signing algorithm (ECDSA). A consensus attack cannot steal bitcoin, spend bitcoin without signatures, redirect bitcoin, or otherwise change past transactions or ownership records. Consensus attacks can only affect the most recent blocks and cause denial-of-service disruptions on the creation of future blocks.

One attack scenario against the consensus mechanism is called the "51% attack." In this scenario a group of miners, controlling a majority (51%) of the total network’s hashing power, collude to attack bitcoin. With the ability to mine the majority of the blocks, the attacking miners can cause deliberate "forks" in the blockchain and double-spend transactions or execute denial-of-service attacks against specific transactions or addresses. A fork/double-spend attack is where the attacker causes previously confirmed blocks to be invalidated by forking below them and re-converging on an alternate chain. With sufficient power, an attacker can invalidate six or more blocks in a row, causing transactions that were considered immutable (six confirmations) to be invalidated. Note that a double-spend can only be done on the attacker’s own transactions, for which the attacker can produce a valid signature. Double-spending one’s own transactions is profitable if by invalidating a transaction the attacker can get an irreversible exchange payment or product without paying for it.

Let’s examine a practical example of a 51% attack. In the first chapter, we looked at a transaction between Alice and Bob for a cup of coffee. Bob, the cafe owner, is willing to accept payment for cups of coffee without waiting for confirmation (mining in a block), because the risk of a double-spend on a cup of coffee is low in comparison to the convenience of rapid customer service. This is similar to the practice of coffee shops that accept credit card payments without a signature for amounts below $25, because the risk of a credit-card chargeback is low while the cost of delaying the transaction to obtain a signature is comparatively larger. In contrast, selling a more expensive item for bitcoin runs the risk of a double-spend attack, where the buyer broadcasts a competing transaction that spends the same inputs (UTXO) and cancels the payment to the merchant. A double-spend attack can happen in two ways: either before a transaction is confirmed, or if the attacker takes advantage of a blockchain fork to undo several blocks. A 51% attack allows attackers to double-spend their own transactions in the new chain, thus undoing the corresponding transaction in the old chain.

In our example, malicious attacker Mallory goes to Carol’s gallery and purchases a beautiful triptych painting depicting Satoshi Nakamoto as Prometheus. Carol sells "The Great Fire" paintings for $250,000 in bitcoin to Mallory. Instead of waiting for six or more confirmations on the transaction, Carol wraps and hands the paintings to Mallory after only one confirmation. Mallory works with an accomplice, Paul, who operates a large mining pool, and the accomplice launches a 51% attack as soon as Mallory’s transaction is included in a block. Paul directs the mining pool to remine the same block height as the block containing Mallory’s transaction, replacing Mallory’s payment to Carol with a transaction that double-spends the same input as Mallory’s payment. The double-spend transaction consumes the same UTXO and pays it back to Mallory’s wallet, instead of paying it to Carol, essentially allowing Mallory to keep the bitcoin. Paul then directs the mining pool to mine an additional block, so as to make the chain containing the double-spend transaction longer than the original chain (causing a fork below the block containing Mallory’s transaction). When the blockchain fork resolves in favor of the new (longer) chain, the double-spent transaction replaces the original payment to Carol. Carol is now missing the three paintings and also has no bitcoin payment. Throughout all this activity, Paul’s mining pool participants might remain blissfully unaware of the double-spend attempt, because they mine with automated miners and cannot monitor every transaction or block.

To protect against this kind of attack, a merchant selling large-value items must wait at least six confirmations before giving the product to the buyer. Alternatively, the merchant should use an escrow multisignature account, again waiting for several confirmations after the escrow account is funded. The more confirmations elapse, the harder it becomes to invalidate a transaction with a 51% attack. For high-value items, payment by bitcoin will still be convenient and efficient even if the buyer has to wait 24 hours for delivery, which would correspond to approximately 144 confirmations.

In addition to a double-spend attack, the other scenario for a consensus attack is to deny service to specific bitcoin participants (specific Bitcoin addresses). An attacker with a majority of the mining power can simply ignore specific transactions. If they are included in a block mined by another miner, the attacker can deliberately fork and remine that block, again excluding the specific transactions. This type of attack can result in a sustained denial-of-service against a specific address or set of addresses for as long as the attacker controls the majority of the mining power.

Despite its name, the 51% attack scenario doesn’t actually require 51% of the hashing power. In fact, such an attack can be attempted with a smaller percentage of the hashing power. The 51% threshold is simply the level at which such an attack is almost guaranteed to succeed. A consensus attack is essentially a tug-of-war for the next block and the "stronger" group is more likely to win. With less hashing power, the probability of success is reduced, because other miners control the generation of some blocks with their "honest" mining power. One way to look at it is that the more hashing power an attacker has, the longer the fork he can deliberately create, the more blocks in the recent past he can invalidate, or the more blocks in the future he can control. Security research groups have used statistical modeling to claim that various types of consensus attacks are possible with as little as 30% of the hashing power.

The massive increase of total hashing power has arguably made bitcoin impervious to attacks by a single miner. There is no possible way for a solo miner to control more than a small percentage of the total mining power. However, the centralization of control caused by mining pools has introduced the risk of for-profit attacks by a mining pool operator. The pool operator in a managed pool controls the construction of candidate blocks and also controls which transactions are included. This gives the pool operator the power to exclude transactions or introduce double-spend transactions. If such abuse of power is done in a limited and subtle way, a pool operator could conceivably profit from a consensus attack without being noticed.

Not all attackers will be motivated by profit, however. One potential attack scenario is where an attacker intends to disrupt the Bitcoin network without the possibility of profiting from such disruption. A malicious attack aimed at crippling bitcoin would require enormous investment and covert planning, but could conceivably be launched by a well-funded, most likely state-sponsored, attacker. Alternatively, a well-funded attacker could attack bitcoin’s consensus by simultaneously amassing mining hardware, compromising pool operators, and attacking other pools with denial-of-service. All of these scenarios are theoretically possible, but increasingly impractical as the Bitcoin network’s overall hashing power continues to grow exponentially.

Undoubtedly, a serious consensus attack would erode confidence in bitcoin in the short term, possibly causing a significant price decline. However, the Bitcoin network and software are constantly evolving, so consensus attacks would be met with immediate countermeasures by the bitcoin community, making bitcoin more robust.

## Changing the Consensus Rules

The rules of consensus determine the validity of transactions and blocks. These rules are the basis for collaboration between all Bitcoin nodes and are responsible for the convergence of all local perspectives into a single consistent blockchain across the entire network.

While the consensus rules are invariable in the short term and must be consistent across all nodes, they are not invariable in the long term. In order to evolve and develop the Bitcoin system, the rules have to change from time to time to accommodate new features, improvements, or bug fixes. Unlike traditional software development, however, upgrades to a consensus system are much more difficult and require coordination between all the participants.

### Hard Forks

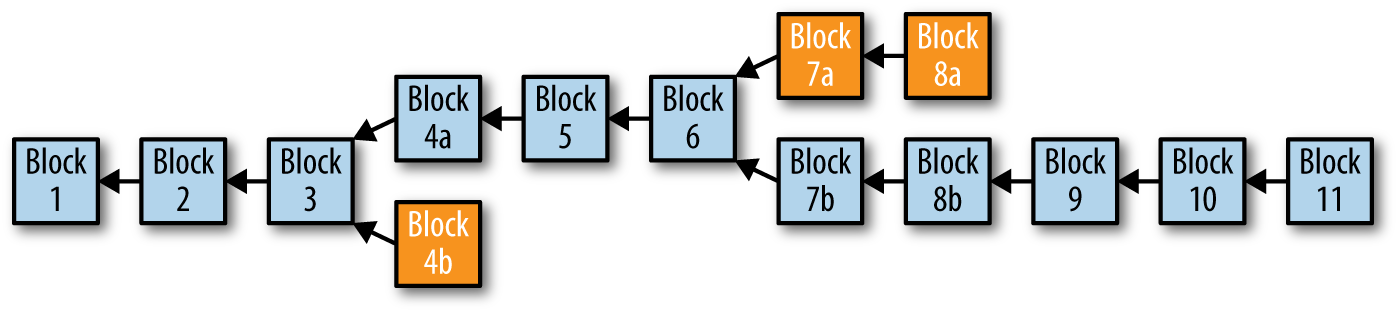
In [???](#forks) we looked at how the Bitcoin network may briefly diverge, with two parts of the network following two different branches of the blockchain for a short time. We saw how this process occurs naturally, as part of the normal operation of the network and how the network reconverges on a common blockchain after one or more blocks are mined.

There is another scenario in which the network may diverge into following two chains: a change in the consensus rules. This type of fork is called a *hard fork*, because after the fork the network does not reconverge onto a single chain. Instead, the two chains evolve independently. Hard forks occur when part of the network is operating under a different set of consensus rules than the rest of the network. This may occur because of a bug or because of a deliberate change in the implementation of the consensus rules.

Hard forks can be used to change the rules of consensus, but they require coordination between all participants in the system. Any nodes that do not upgrade to the new consensus rules are unable to participate in the consensus mechanism and are forced onto a separate chain at the moment of the hard fork. Thus, a change introduced by a hard fork can be thought of as not "forward compatible," in that non-upgraded systems can’t process the new consensus rules after the hard fork event.

Let’s examine the mechanics of a hard fork with a specific example.

[figure\_title](#blockchainwithforks) shows a blockchain with two forks. At block height 4, a one-block fork occurs. This is the type of spontaneous fork we saw in [???](#forks). With the mining of block 5, the network reconverges on one chain and the fork is resolved.



A blockchain with forks

Later, however, at block height 6, a hard fork occurs. Let’s assume that a new implementation of the client is released with a change in the consensus rules. Starting on block height 7, miners running this new implementation will accept a new type of digital signature, let’s call it a "Smores" signature, that is not ECDSA based. Immediately after, a node running the new implementation creates a transaction that contains a Smores signature and a miner with the updated software mines block 7b containing this transaction.

Any node or miner that has not upgraded the software to validate Smores signatures is now unable to process block 7b. From their perspective, both the transaction that contained a Smores signature and block 7b that contained that transaction are invalid, because they are evaluating them based upon the old consensus rules. These nodes will reject the transaction and the block and will not propagate them. Any miners that are using the old rules will not accept block 7b and will continue to mine a candidate block whose parent is block 6. In fact, miners using the old rules may not even receive block 7b if all the nodes they are connected to are also obeying the old rules and therefore not propagating the block. Eventually, they will be able to mine block 7a, which is valid under the old rules and does not contain any transactions with Smores signatures.

The two chains continue to diverge from this point. Miners on the "b" chain will continue to accept and mine transactions containing Smores signatures, while miners on the "a" chain will continue to ignore these transactions. Even if block 8b does not contain any Smores-signed transactions, the miners on the "a" chain cannot process it. To them it appears to be an orphan block, as its parent "7b" is not recognized as a valid block.

### Hard Forks: Software, Network, Mining, and Chain

For software developers, the term "fork" has another meaning, adding confusion to the term "hard fork." In open source software, a fork occurs when a group of developers choose to follow a different software roadmap and start a competing implementation of an open source project. We’ve already discussed two circumstances that will lead to a hard fork in bitcoin: a bug in the consensus rules and a deliberate modification of the consensus rules. In the case of a deliberate change to the consensus rules, a software fork precedes the hard fork. However, for this type of hard fork to occur, a new software implementation of the consensus rules must be developed, adopted, and launched.

Examples of software forks that have attempted to change consensus rules include Bitcoin XT, Bitcoin Classic, and most recently Bitcoin Unlimited. However, none of these software forks have resulted in a hard fork. While a software fork is a necessary precondition, it is not in itself sufficient for a hard fork to occur. For a hard fork to occur, the competing implementation must be adopted and the new rules activated, by miners, wallets, and intermediary nodes. Conversely, there are numerous alternative implementations of Bitcoin Core, and even software forks, that do not change the consensus rules and barring a bug, can coexist on the network and interoperate without causing a hard fork.

Consensus rules may differ in obvious and explicit ways, in the validation of transactions or blocks. The rules may also differ in more subtle ways, in the implementation of the consensus rules as they apply to bitcoin scripts or cryptographic primitives such as digital signatures. Finally, the consensus rules may differ in unanticipated ways because of implicit consensus constraints imposed by system limitations or implementation details. An example of the latter was seen in the unanticipated hard fork during the upgrade of Bitcoin Core 0.7 to 0.8, which was caused by a limitation in the Berkeley DB implementation used to store blocks.

Conceptually, we can think of a hard fork as developing in four stages: a software fork, a network fork, a mining fork, and a chain fork.

The process begins when an alternative implementation of the client, with modified consensus rules, is created by developers.

When this forked implementation is deployed in the network, a certain percentage of miners, wallet users, and intermediate nodes may adopt and run this implementation. A resulting fork will depend upon whether the new consensus rules apply to blocks, transactions, or some other aspect of the system. If the new consensus rules pertain to transactions, then a wallet creating a transaction under the new rules may precipitate a network fork, followed by a hard fork when the transaction is mined into a block. If the new rules pertain to blocks, then the hard fork process will begin when a block is mined under the new rules.

First, the network will fork. Nodes based on the original implementation of the consensus rules will reject any transactions and blocks that are created under the new rules. Furthermore, the nodes following the original consensus rules will temporarily ban and disconnect from any nodes that are sending them these invalid transactions and blocks. As a result, the network will partition into two: old nodes will only remain connected to old nodes and new nodes will only be connected to new nodes. A single transaction or block based on the new rules will ripple through the network and result in the partition into two networks.

Once a miner using the new rules mines a block, the mining power and chain will also fork. New miners will mine on top of the new block, while old miners will mine a separate chain based on the old rules. The partitioned network will make it so that the miners operating on separate consensus rules won’t likely receive each other’s blocks, as they are connected to two separate networks.

### Diverging Miners and Difficulty

As miners diverge into mining two different chains, the hashing power is split between the chains. The mining power can be split in any proportion between the two chains. The new rules may only be followed by a minority, or by the vast majority of the mining power.

Let’s assume, for example, an 80%–20% split, with the majority of the mining power using the new consensus rules. Let’s also assume that the fork occurs immediately after a retargeting period.

The two chains would each inherit the difficulty from the retargeting period. The new consensus rules would have 80% of the previously available mining power committed to them. From the perspective of this chain, the mining power has suddenly declined by 20% vis-a-vis the previous period. Blocks will be found on average every 12.5 minutes, representing the 20% decline in mining power available to extend this chain. This rate of block issuance will continue (barring any changes in hashing power) until 2016 blocks are mined, which will take approximately 25,200 minutes (at 12.5 minutes per block), or 17.5 days. After 17.5 days, a retarget will occur and the difficulty will adjust (reduced by 20%) to produce 10-minute blocks again, based on the reduced amount of hashing power in this chain.

The minority chain, mining under the old rules with only 20% of the hashing power, will face a much more difficult task. On this chain, blocks will now be mined every 50 minutes on average. The difficulty will not be adjusted for 2016 blocks, which will take 100,800 minutes, or approximately 10 weeks to mine. Assuming a fixed capacity per block, this will also result in a reduction of transaction capacity by a factor of 5, as there are fewer blocks per hour available to record transactions.

### Contentious Hard Forks

This is the dawn of consensus software development. Just as open source development changed both the methods and products of software and created new methodologies, new tools, and new communities in its wake, consensus software development also represents a new frontier in computer science. Out of the debates, experiments, and tribulations of the bitcoin development roadmap, we will see new development tools, practices, methodologies, and communities emerge.

Hard forks are seen as risky because they force a minority to either upgrade or remain on a minority chain. The risk of splitting the entire system into two competing systems is seen by many as an unacceptable risk. As a result, many developers are reluctant to use the hard fork mechanism to implement upgrades to the consensus rules, unless there is near-unanimous support from the entire network. Any hard fork proposals that do not have near-unanimous support are considered too "contentious" to attempt without risking a partition of the system.

The issue of hard forks is highly controversial in the bitcoin development community, especially as it relates to any proposed changes to the consensus rules controlling the maximum block size limit. Some developers are opposed to any form of hard fork, seeing it as too risky. Others see the mechanism of hard fork as an essential tool for upgrading the consensus rules in a way that avoids "technical debt" and provides a clean break with the past. Finally, some developers see hard forks as a mechanism that should be used rarely, with a lot of advance planning and only under near-unanimous consensus.

Already we have seen the emergence of new methodologies to address the risks of hard forks. In the next section, we will look at soft forks, and the BIP-34 and BIP-9 methods for signaling and activation of consensus modifications.

### Soft Forks

Not all consensus rule changes cause a hard fork. Only consensus changes that are forward-incompatible cause a fork. If the change is implemented in such a way that a non-upgraded client still sees the transaction or block as valid under the previous rules, the change can happen without a fork.

The term *soft fork* was introduced to distinguish this upgrade method from a "hard fork." In practice, a soft fork is not a fork at all. A soft fork is a forward-compatible change to the consensus rules that allows non-upgraded clients to continue to operate in consensus with the new rules.

One aspect of soft forks that is not immediately obvious is that soft fork upgrades can only be used to constrain the consensus rules, not to expand them. In order to be forward compatible, transactions and blocks created under the new rules must be valid under the old rules too, but not vice versa. The new rules can only limit what is valid; otherwise, they will trigger a hard fork when rejected under the old rules.

Soft forks can be implemented in a number of ways—the term does not specify a particular method, rather a set of methods that all have one thing in common: they don’t require all nodes to upgrade or force non-upgraded nodes out of consensus.

#### Soft forks redefining NOP opcodes

A number of soft forks have been implemented in bitcoin, based on the re-interpretation of NOP opcodes. Bitcoin Script had ten opcodes reserved for future use, NOP1 through NOP10. Under the consensus rules, the presence of these opcodes in a script is interpreted as a null-potent operator, meaning they have no effect. Execution continues after the NOP opcode as if it wasn’t there.

A soft fork therefore can modify the semantics of a NOP code to give it new meaning. For example, BIP-65 (CHECKLOCKTIMEVERIFY) reinterpreted the NOP2 opcode. Clients implementing BIP-65 interpret NOP2 as OP\_CHECKLOCKTIMEVERIFY and impose an absolute locktime consensus rule on UTXO that contain this opcode in their locking scripts. This change is a soft fork because a transaction that is valid under BIP-65 is also valid on any client that is not implementing (ignorant of) BIP-65. To the old clients, the script contains an NOP code, which is ignored.

#### Other ways to soft fork upgrade

The reinterpretation of NOP opcodes was both planned for and an obvious mechanism for consensus upgrades. Recently, however, another soft fork mechanism was introduced that does not rely on NOP opcodes for a very specific type of consensus change. This is examined in more detail in [???](#segwit). Segwit is an architectural change to the structure of a transaction, which moves the unlocking script (witness) from inside the transaction to an external data structure (segregating it). Segwit was initially envisioned as a hard fork upgrade, as it modified a fundamental structure (transaction). In November 2015, a developer working on Bitcoin Core proposed a mechanism by which segwit could be introduced as a soft fork. The mechanism used for this is a modification of the locking script of UTXO created under segwit rules, such that non-upgraded clients see the locking script as redeemable with any unlocking script whatsoever. As a result, segwit can be introduced without requiring every node to upgrade or split from the chain: a soft fork.

It is likely that there are other, yet to be discovered, mechanisms by which upgrades can be made in a forward-compatible way as a soft fork.

### Criticisms of Soft Forks

Soft forks based on the NOP opcodes are relatively uncontroversial. The NOP opcodes were placed in Bitcoin Script with the explicit goal of allowing non-disruptive upgrades.

However, many developers are concerned that other methods of soft fork upgrades make unacceptable tradeoffs. Common criticisms of soft fork changes include:

Technical debt

Because soft forks are more technically complex than a hard fork upgrade, they introduce *technical debt*, a term that refers to increasing the future cost of code maintenance because of design tradeoffs made in the past. Code complexity in turn increases the likelihood of bugs and security vulnerabilities.

Validation relaxation

Non-upgraded clients see transactions as valid, without evaluating the modified consensus rules. In effect, the non-upgraded clients are not validating using the full range of consensus rules, as they are blind to the new rules. This applies to NOP-based upgrades, as well as other soft fork upgrades.

Irreversible upgrades

Because soft forks create transactions with additional consensus constraints, they become irreversible upgrades in practice. If a soft fork upgrade were to be reversed after being activated, any transactions created under the new rules could result in a loss of funds under the old rules. For example, if a CLTV transaction is evaluated under the old rules, there is no timelock constraint and it can be spent at any time. Therefore, critics contend that a failed soft fork that had to be reversed because of a bug would almost certainly lead to loss of funds.

## Soft Fork Signaling with Block Version

Since soft forks allow non-upgraded clients to continue to operate within consensus, the mechanism for "activating" a soft fork is through miners signaling readiness: a majority of miners must agree that they are ready and willing to enforce the new consensus rules. To coordinate their actions, there is a signaling mechanism that allows them to show their support for a consensus rule change. This mechanism was introduced with the activation of BIP-34 in March 2013 and replaced by the activation of BIP-9 in July 2016.

### BIP-34 Signaling and Activation

The first implementation, in BIP-34, used the block version field to allow miners to signal readiness for a specific consensus rule change. Prior to BIP-34, the block version was set to "1" by *convention* not enforced by *consensus*.

BIP-34 defined a consensus rule change that required the coinbase data field of a coinbase transaction input to contain the block height. Prior to BIP-34, the coinbase data could contain any arbitrary data the miners chose to include. After activation of BIP-34, valid blocks had to contain a specific block-height at the beginning of the coinbase data and be identified with a version number greater than or equal to "2."

To signal the change and activation of BIP-34, miners set the block version to "2," instead of "1." This did not immediately make version "1" blocks invalid. Once activated, version "1" blocks would become invalid and all version "2" blocks would be required to contain the block height in the coinbase to be valid.

BIP-34 defined a two-step activation mechanism, based on a rolling window of 1000 blocks. A miner would signal his or her individual readiness for BIP-34 by constructing blocks with "2" as the version number. Strictly speaking, these blocks did not yet have to comply with the new consensus rule of including the block-height in the coinbase transaction because the consensus rule had not yet been activated. The consensus rules activated in two steps:

* If 75% (750 of the most recent 1000 blocks) are marked with version "2," then version "2" blocks must contain block height in the coinbase transaction or they are rejected as invalid. Version "1" blocks are still accepted by the network and do not need to contain block-height. The old and new consensus rules coexist during this period.
* When 95% (950 of the most recent 1000 blocks) are version "2," version "1" blocks are no longer considered valid. Version "2" blocks are valid only if they contain the block-height in the coinbase (as per the previous threshold). Thereafter, all blocks must comply with the new consensus rules, and all valid blocks must contain block-height in the coinbase transaction.

After successful signaling and activation under the BIP-34 rules, this mechanism was used twice more to activate soft forks:

* [BIP-66](https://github.com/bitcoin/bips/blob/master/bip-0066.mediawiki) Strict DER Encoding of Signatures was activated by BIP-34 style signaling with a block version "3" and invalidating version "2" blocks.
* [BIP-65](https://github.com/bitcoin/bips/blob/master/bip-0065.mediawiki) CHECKLOCKTIMEVERIFY was activated by BIP-34 style signaling with a block version "4" and invalidating version "3" blocks.

After the activation of BIP-65, the signaling and activation mechanism of BIP-34 was retired and replaced with the BIP-9 signaling mechanism described next.

The standard is defined in [BIP-34 (Block v2, Height in Coinbase)](https://github.com/bitcoin/bips/blob/master/bip-0034.mediawiki).

### BIP-9 Signaling and Activation

The mechanism used by BIP-34, BIP-66, and BIP-65 was successful in activating three soft forks. However, it was replaced because it had several limitations:

* By using the integer value of the block version, only one soft fork could be activated at a time, so it required coordination between soft fork proposals and agreement on their prioritization and sequencing.
* Furthermore, because the block version was incremented, the mechanism didn’t offer a straightforward way to reject a change and then propose a different one. If old clients were still running, they could mistake signaling for a new change as signaling for the previously rejected change.
* Each new change irrevocably reduced the available block versions for future changes.

BIP-9 was proposed to overcome these challenges and improve the rate and ease of implementing future changes.

BIP-9 interprets the block version as a bit field instead of an integer. Because the block version was originally used as an integer, versions 1 through 4, only 29 bits remain available to be used as a bit field. This leaves 29 bits that can be used to independently and simultaneously signal readiness on 29 different proposals.

BIP-9 also sets a maximum time for signaling and activation. This way miners don’t need to signal forever. If a proposal is not activated within the TIMEOUT period (defined in the proposal), the proposal is considered rejected. The proposal may be resubmitted for signaling with a different bit, renewing the activation period.

Furthermore, after the TIMEOUT has passed and a feature has been activated or rejected, the signaling bit can be reused for another feature without confusion. Therefore, up to 29 changes can be signaled in parallel and after TIMEOUT the bits can be "recycled" to propose new changes.

While signaling bits can be reused or recycled, as long as the voting period does not overlap, the authors of BIP-9 recommend that bits are reused only when necessary; unexpected behavior could occur due to bugs in older software. In short, we should not expect to see reuse until all 29 bits have been used once.

Proposed changes are identified by a data structure that contains the following fields:

name

A short description used to distinguish between proposals. Most often the BIP describing the proposal, as "bipN," where N is the BIP number.

bit

0 through 28, the bit in the block version that miners use to signal approval for this proposal.

starttime

The time (based on Median Time Past, or MTP) that signaling starts after which the bit’s value is interpreted as signaling readiness for the proposal.

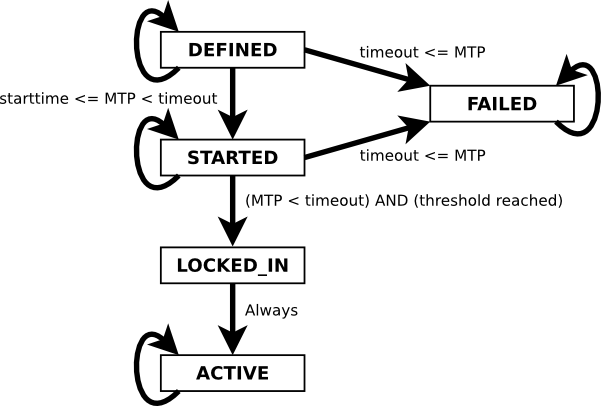
endtime

The time (based on MTP) after which the change is considered rejected if it has not reached the activation threshold.

Unlike BIP-34, BIP-9 counts activation signaling in whole intervals based on the difficulty retarget period of 2016 blocks. For every retarget period, if the sum of blocks signaling for a proposal exceeds 95% (1916 of 2016), the proposal will be activated one retarget period later.

BIP-9 offers a proposal state diagram to illustrate the various stages and transitions for a proposal, as shown in [figure\_title](#bip9states).

Proposals start in the DEFINED state, once their parameters are known (defined) in the bitcoin software. For blocks with MTP after the start time, the proposal state transitions to STARTED. If the voting threshold is exceeded within a retarget period and the timeout has not been exceeded, the proposal state transitions to LOCKED\_IN. One retarget period later, the proposal becomes ACTIVE. Proposals remain in the ACTIVE state perpetually once they reach that state. If the timeout elapses before the voting threshold has been reached, the proposal state changes to FAILED, indicating a rejected proposal. FAILED proposals remain in that state perpetually.



BIP-9 state transition diagram

BIP-9 was first implemented for the activation of CHECKSEQUENCEVERIFY and associated BIPs (68, 112, 113). The proposal named "csv" was activated successfully in July of 2016.

The standard is defined in [BIP-9 (Version bits with timeout and delay)](https://github.com/bitcoin/bips/blob/master/bip-0009.mediawiki).

## Consensus Software Development

Consensus software continues to evolve and there is much discussion on the various mechanisms for changing the consensus rules. By its very nature, bitcoin sets a very high bar on coordination and consensus for changes. As a decentralized system, it has no "authority" that can impose its will on the participants of the network. Power is diffused between multiple constituencies such as miners, core developers, wallet developers, exchanges, merchants, and end users. Decisions cannot be made unilaterally by any of these constituencies. For example, while miners can theoretically change the rules by simple majority (51%), they are constrained by the consent of the other constituencies. If they act unilaterally, the rest of the participants may simply refuse to follow them, keeping the economic activity on a minority chain. Without economic activity (transactions, merchants, wallets, exchanges), the miners will be mining a worthless coin with empty blocks. This diffusion of power means that all the participants must coordinate, or no changes can be made. Status quo is the stable state of this system with only a few changes possible if there is strong consensus by a very large majority. The 95% threshold for soft forks is reflective of this reality.

It is important to recognize that there is no perfect solution for consensus development. Both hard forks and soft forks involve tradeoffs. For some types of changes, soft forks may be a better choice; for others, hard forks may be a better choice. There is no perfect choice; both carry risks. The one constant characteristic of consensus software development is that change is difficult and consensus forces compromise.

Some see this as a weakness of consensus systems. In time, you may come to see it as I do, as the system’s greatest strength.