共識機制與工作量證明及模擬挖礦實作

盧瑞山 教授

認識區塊結構

區塊頭

- 比特幣的區塊由區塊頭及該區塊所包含的交易列表組成。
- 區塊頭的大小為80字節，
- 由4字節的版本號、
- 32字節的上一個區塊的散列值
- 32字節的Merkle Root Hash
- 4字節的時間綴（當前時間）
- 4字節的當前難度值
- 4字節的隨機數組成。

區塊結構

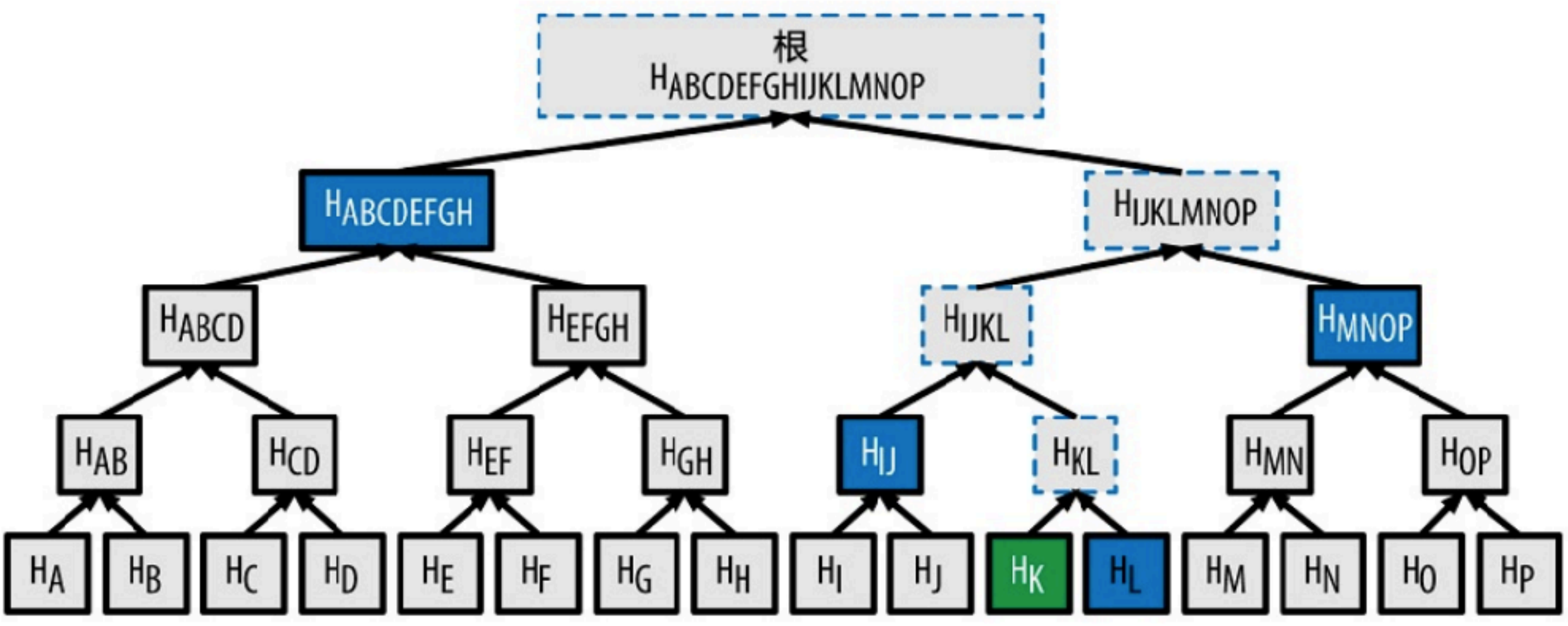
version	02000000
previous block hash (reversed)	17975b97c18ed1f7e255adf297599b55 330edab87803c81701000000000000000
Merkle root (reversed)	8a97295a2747b4f1a0b3948df3990344 c0e19fa6b2b92b3a19c8e6badc141787
timestamp	358b0553
bits	535f0119
nonce	48750833
transaction count	63
coinbase transaction	
transaction	
...	

Block hash

000000000000000 e067a478024addfe cdc93628978aa52d 91fabd4292982a50

Field	Description	Size
Magic no	value always 0xD9B4BEF9	4 bytes
Blocksize	number of bytes following up to end of block	4 bytes
Blockheader	consists of 6 items	80 bytes
Transaction counter	positive integer VI = VarInt	1 - 9 bytes
transactions	the (non empty) list of transactions	<Transaction counter>-many transactions

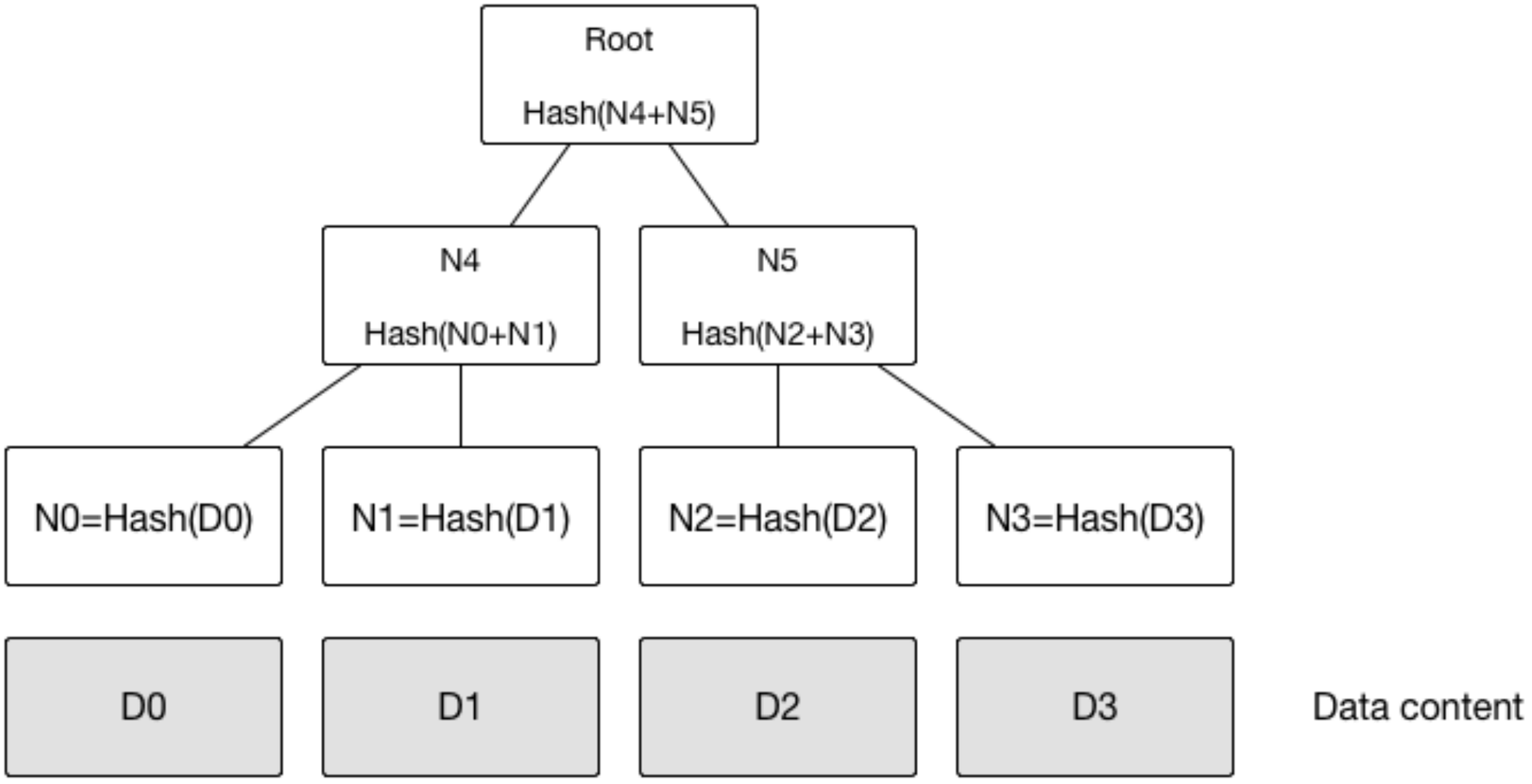
可證明交易存在的Merkle path



Merkle tree的效率

交易数量	区块的近似大小	路径大小（哈希数量）	路径大小（字节）
16笔交易	4KB	4个哈希	128字节
512笔交易	128KB	9个哈希	288字节
2048笔交易	512KB	11个哈希	352字节
65,535笔交易	16MB	16个哈希	512字节

梅根樹範例


Root
Hash(N4+N5)
N4
Hash(N0+N1)
N5
Hash(N2+N3)
N0=Hash(D0)
N1=Hash(D1)
N2=Hash(D2)
N3=Hash(D3)
D0
D1
D2
D3
Data content

梅根樹的應用場景

默克爾樹的典型應用場景包括：

快速比較大量數據：當兩個默克爾樹根相同時，則意味著所代表的數據必然相同。

快速定位修改：例如上例中，如果 D1 中數據被修改，會影響到 N1，N4 和 Root。因此，沿著 Root --> N4 --> N1，可以快速定位到發生改變的 D1；

零知識證明：例如如何證明某個數據（D0……D3）中包括給定內容 D0，很簡單，構造一個默克爾樹，公佈 N0，N1，N4，Root，D0 擁有者可以很容易檢測 D0 存在，但不知道其它內容。

哈希現金

雜湊現金

文獻回顧

- • 哈希現金是一種工作量證明機制，它是亞當·貝克（Adam Back）在1997年發明的，用於抵抗郵件的拒絕服務攻擊及垃圾郵件網關濫用。在比特幣之前，哈希現金被用於垃圾郵件的過濾，也被微軟用於hotmail/exchange/outlook等產品中（微軟使用一種與哈希現金不兼容的格式並將之命名為電子郵戳

Challenge and Nonce

工作量證明主要用於防止拒絕服務攻擊和反垃圾信息
通常這種「工作證明」會花費一定的時間計算才能得到
最常見的例子是CAPTCHA

另外用於防止DoS和垃圾信息的機制是HashCash
比特幣使用的原理就類似於HashCash

哈希現金（hashcash ）的靈感來自於這樣一個想法，即一些數學結果難於發現而易於校驗
一個眾所周知的例子是因數分解一個大的數字（尤其是因數較少的數字）
將數字相乘來獲得它們的積的代價是低廉的，但首先找到那些因數的代價卻要高得多

Consensus

共識機制

共識機制

中本聰在提出工作量證明(POW)機製作為共識算法後，部分人認為耗能過大，Sunny King就設計出「環保」的權益證明(POS)機制，後續到Bitshares改進的代理權益證明(DPOS)，並衍生到更多的類POS機制。從公有鏈角度，共識算法就是公平和效率孰重孰輕的決策，技術實現不是難點, 難點在於如何從社會學從人性出發去設計激勵機制。各種共識算法的支持者都有其合理的理由，不同共識的爭論即使到現在還一直存在。

工作量證明

Proof of Work

工作量證明的理論基礎

- 一種很難算出答案，但很好驗證的演算法
- 哈希現金 Hashcash
- Adam Beck 1997提出Hashcash的觀念
- 應用：拒絕垃圾郵件的轉發，拒絕阻斷式服務攻擊
- Hash = sha256(data+nonce)
- Hash:000012345678
- more info go visit to: hashcash.org

中本聰的思維

中本聰在其所撰寫最原始的比特幣論文中寫到：

The proof-of-work involves scanning for a value that when hashed, such as with SHA-256, the hash begins with a number of zero bits. The average work required is exponential in the numberof zero bits required and can be verified by executing a single hash.

意思就是：工作量證明過程包括掃描SHA-256的哈希數由多少個0開頭，每增加一個0，平均工作量都會有指數級的增加，就是二的四次方，增加了多少個零就是多少個二的四次方乘在一起倍數的工作量增加，這些將在解一個哈希數（也就是挖一個比特幣的block過程）中得到證明

區塊100000

区块 #100000

概览	
交易次数	4
总输出量	103.01 BTC
预计交易量	5.53 BTC
交易费	0 BTC
高度	100000 (主链)
时间戳	2010-12-29 11:57:43
难度系数	14,484.16
计算目标	453281356
大小	0.957 KB
版本	1
随机数	274148111

哈希值	
哈希值	000000000003ba27aa200b1cecaad478d2b00432346c3f1f3986da1afd33e506
上一区块	00000000000 2d01c1fccc21636b607dfd930d31d01c3a62104612a1719011250
下一区块	000000000000 30b66c911bd5ba14a74260057311eaeb1982802f7010f1a91090
二进制哈希树根	f3e94742aca4b5ef85438dc37c06c3282295ffec960994b2c0d5ac2a25a95766

网络传播算法 (按此浏览)

區塊100001

区块 #100001

概览	
交易次数	12
总输出量	837.06 BTC
预计交易量	281.1 BTC
交易费	0 BTC
高度	100001 (主链)
时间戳	2010-12-29 12:06:44
难度系数	14,484.16
计算目标	453281356
大小	3.308 KB
版本	1
随机数	2613872960
新区块奖励	50 BTC

哈希值	
哈希值	000000000003ba27aa200b1cecaad478d2b00432346c3f1f3986da1afd33e506
上一区块	000000000003ba27aa200b1cecaad478d2b00432346c3f1f3986da1afd33e506
下一区块	0000000000013b8ab2cd513b0261a14096412195a72a0c4827d229dcc7e0f7af
二进制哈希树根	7fe79307acb300d910d9e4bcc5bacb4c7c114c7dfd6789c19f3a733dcbb3bb6a

网络传播算法 (按此浏览)

- Bitcoin consensus algorithm (simplified)

- This algorithm is simplified in that it assumes the ability to select a random node in a manner that is not vulnerable to Sybil attacks.

- 1. New transactions are broadcast to all nodes

- 2. Each node collects new transactions into a block

- 3. In each round a random node gets to broadcast its block

- 4. Other nodes accept the block only if all transactions in it are valid (unspent, valid signatures)

- 5. Nodes express their acceptance of the block by including its hash in the next block they create

工作量证明解題的步驟

- 生成Coinbase交易，並與其他所有準備打包進區塊的交易組成交易列表，通過Merkle Tree算法生成Merkle Root Hash

- 把Merkle Root Hash及其他相關字段組裝成區塊頭，將區塊頭的80字節數據（Block Header）作為工作量證明的輸入

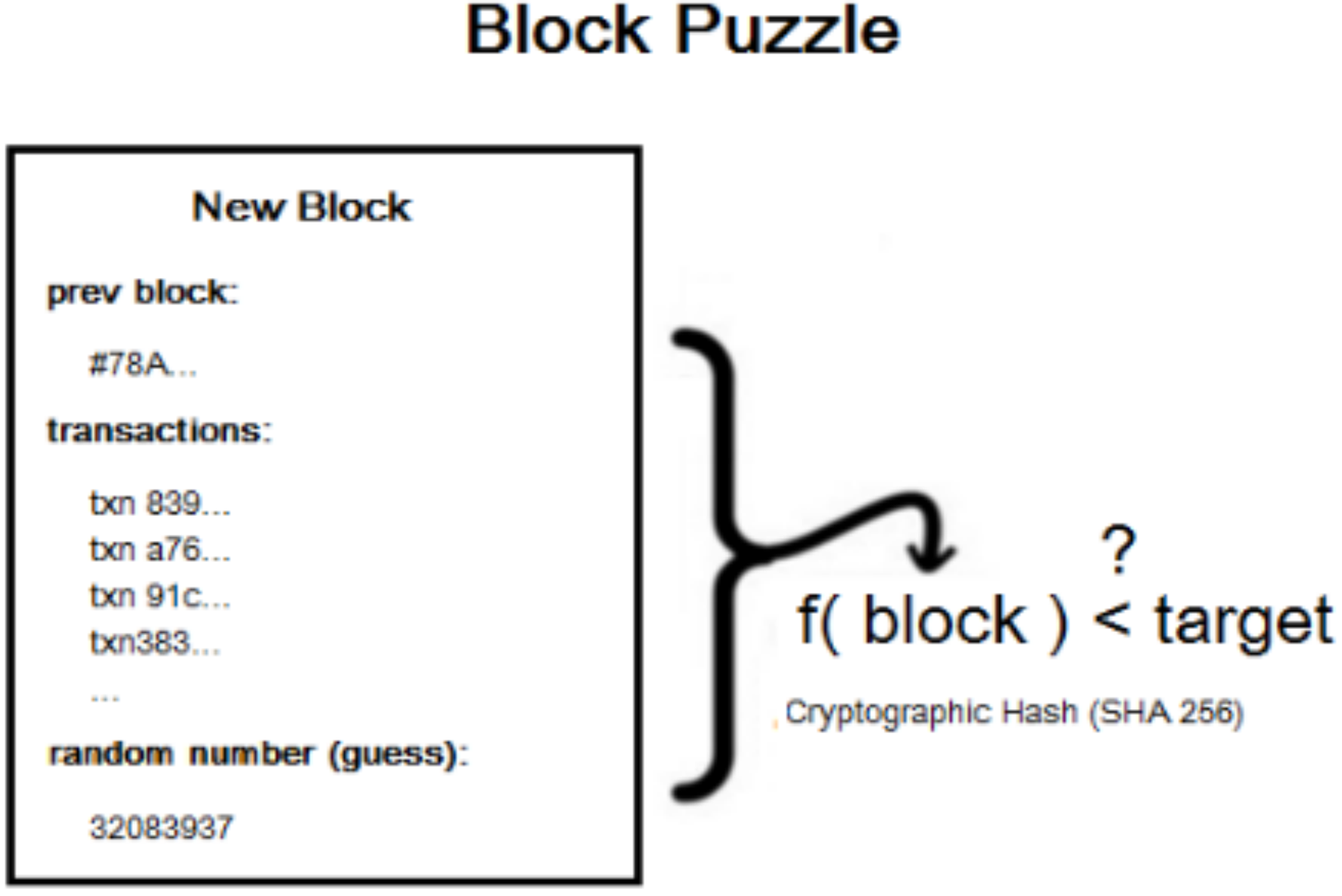
- 不停的變更區塊頭中的隨機數即nonce的數值，並對每次變更後的的區塊頭做雙重SHA256運算（即SHA256(SHA256(Block_Header))），將結果值與當前網絡的目標值做對比，如果小於目標值，則解題成功，工作量證明完成。

真正的工作量證明(挖礦運算過程)

- 挖礦的過程就是找到x使得
  SHA256(SHA256(version + prev_hash + merkle_root + ntime + nbits + x)) < TARGET

上式的x的範圍是0~2^32, TARGET可以根據當前難度求出的。除了x之外，你還可以嘗試改動merkle_root和ntime。由於hash的特性，找這樣一個x只能暴力搜索(**窮舉法**)。

pow 總結


Block Puzzle
New Block
prev block:
#78A...
transactions:
txn 839...
txn a76...
txn 91c...
txn383...
...
random number (guess):
32083937
?
f(block) < target
Cryptographic Hash (SHA 256)

難度值的調整

新难度值 = 旧难度值 * (过去2016个区块花费时长 / 20160 分钟)

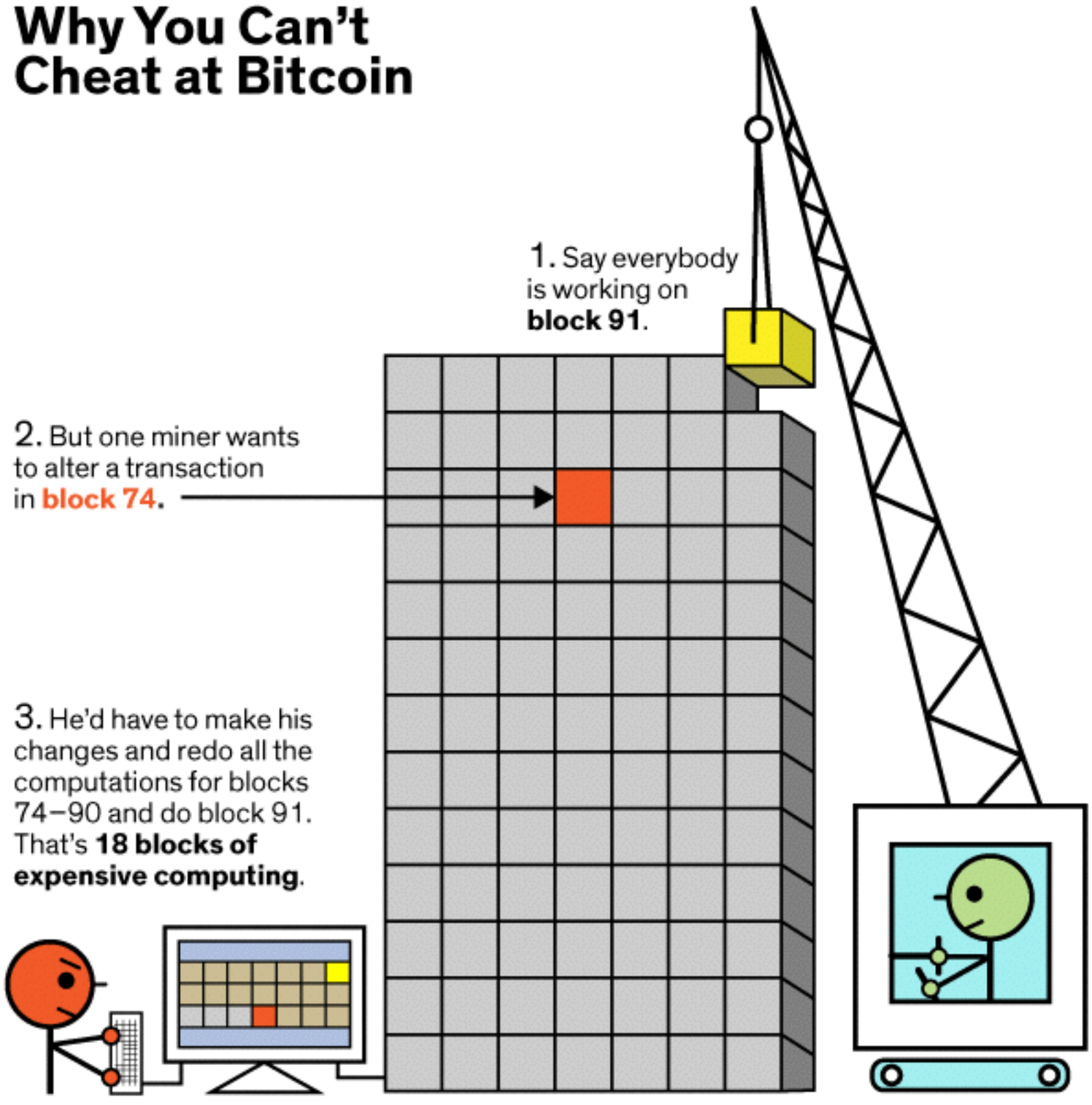
工作量证明的目标值

- 目标值 = 最大目标值 / 难度值
- 其中最大目标值为一个恒定值：
- 0x00000000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFF

Why You Can't Cheat at Bitcoin

1. Say everybody is working on **block 91**.

2. But one miner wants to alter a transaction in **block 74**.



3. He'd have to make his changes and redo all the computations for blocks 74–90 and do block 91. That's **18 blocks of expensive computing**.

4. What's worse, he'd have to do it all **before** everybody else in the Bitcoin network finished **just the one block (number 91)** that they're working on.

用Python模擬工作量證明

POW工作量證明的模擬

- 利用**哈希現金**的原理
- 給定一個隨機或固定的挑戰值 及一個目標門檻值
- 求解方必須拿著這個被指定的挑戰值 加上自己的嘗試值 串連起來，帶入哈希函數 得到一次的求解值
- 把這次的求解值與目標門檻值進行比對
- 是否小於目標門檻值？ (掃描SHA-256的哈希數由多少個0開頭)
- 若是的話，求解成功！ 若不是的話，重新更換一個嘗試值繼續試誤下去 (try and error) 或稱暴力搜尋法 或稱遍歷法 或稱試誤法

用Python模擬工作量證明

```
import string
import random
import hashlib

example_challenge = '9Kzs52jSfxjGJ54Sfjz5gZllls'

def generation(challenge=example_challenge,size=25):
    answer = ''.join(random.choice(string.ascii_lowercase +
                                   string.ascii_uppercase +
                                   string.digits) for x in range(size))
    attempt = challenge+answer
    return attempt, answer

shaHash = hashlib.sha256()

def testAttempt():
    attempt, answer = generation()
    shaHash.update(attempt)
    solution = shaHash.hexdigest()
    if solution.startswith('000'):
        print solution

for x in xrange(0,10000):
    testAttempt()
```

https://www.youtube.com/watch?v=K_Ac1Ko8-p8&list=PLQVvvaa0QuDebbCxrDPCux6SzC1RET4mF&t=757s&index=6

盧老師改寫的哈希現金程式碼

```
import string
import random
import hashlib
import datetime
shahash = hashlib.sha256()
challenge_example = 'dJf2LKs29Djkfdj3897jfdkjf323719'

def source_generation (challenge=challenge_example, size =30):
    answer = ''.join(random.choice(string.ascii_lowercase + string.ascii_uppercase +
string.digits) for x in range(size))
    attempt = challenge + answer
    return attempt, answer
```

```
def mining():
    attempt,answer = source_generation()
    shahash.update(attempt)
    solution = shahash.hexdigest()
    if solution.startswith('0000'):
        endtime = datetime.datetime.now()
        print solution
        print (endtime - starttime).seconds

starttime = datetime.datetime.now()

for x in range(0,1000000):
    mining()
```