Blockchain

Shuai Wang

香 港 科 技 大 學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

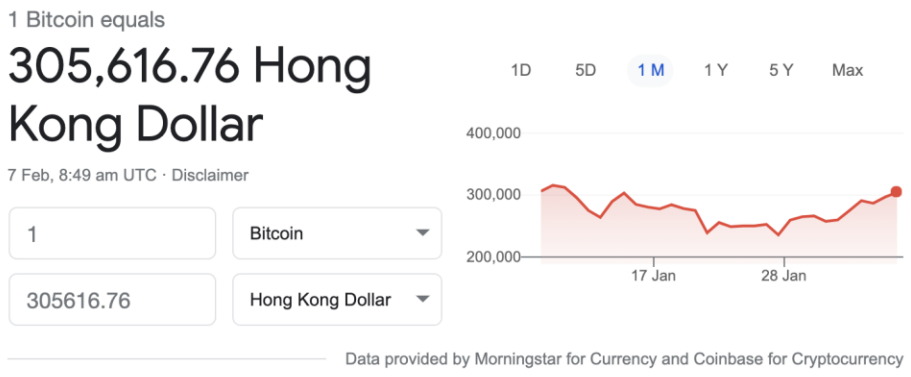
Digital Currency

- Digital currency is not new, but blockchain/Bitcoin generates quite a buzz.

On May 22, 2010, now known as **Bitcoin Pizza** Day, Laszlo Hanyecz agreed to pay 10,000 **Bitcoins** for two delivered Papa John's **pizzas**. Organized on bitcointalk forum, the Florida man reached out for help. Jun 25, 2019

www.investopedia.com › News › Markets News

Bitcoin Pizza Day: Celebrating the $80 Million Pizza Order



Preliminaries: Ledgers

- ***Ledger*** is a book of financial accounts
- Suppose Alice, Bob, Charlie, Trudy play weekly poker game online
- They all insert ledger entries such as,

  "Bob owes Alice $10", "Charlie owes Trudy $30", "Trudy owes Alice $25", and so on
- Once a month, they meet and settle up
- Any possible problems here?

Signed Ledger Entries

- How to prevent Trudy from inserting, say, “Bob owes Trudy $1M” ?
- Let’s require **digital *signatures***
  - For ledger entry to be valid, Bob must sign “Bob owes Alice $10”, Trudy must sign “Trudy owes Alice $25”, and so on …
- Then we know ledger entries are valid
  - That is, the payer agrees to pay

Signed Ledger

- Ledger now looks like
  - $[\text{Bob owes Alice } \$10]_{\text{Bob}}$
  - $[\text{Charlie owes Trudy } \$30]_{\text{Charlie}}$
  - $[\text{Trudy owes Alice } \$25]_{\text{Trudy}}$
  - and so on …
- And we know ledger entries are valid
- But, still some problems here…

Signed Ledger in Detail

- As an aside, note that signatures on previous slide really look like
  - $(M_1,[h(M_1)]_{Bob})$, where M_1="Bob owes Alice $10"
  - $(M_2,[h(M_2)]_{Charlie})$, M_2="Charlie owes Trudy $30"
  - $(M_3,[h(M_3)]_{Trudy})$, M_3="Trudy owes Alice $25"
  - And so on …
- We'll use the shorthand on previous slide

Ledger Duplication

- Still, nothing to prevent Trudy from duplicating a line…
  - $[\text{Bob owes Alice \$10}]_{\text{Bob}}$
  - $[\text{Charlie owes Trudy \$30}]_{\text{Charlie}}$
  - $[\text{Trudy owes Alice \$25}]_{\text{Trudy}}$
  - $[\text{Charlie owes Trudy \$30}]_{\text{Charlie}}$
- Signatures are still all valid
- How to prevent this attack?

Unique Ledger Entries

- Include unique transaction numbers
  - [1, Bob owes Alice \$10]$_{Bob}$
  - [2, Charlie owes Trudy \$30]$_{Charlie}$
  - [3, Trudy owes Alice \$25]$_{Trudy}$
  - And so on…
- Why does this help?
- We will never have an exact duplicate
  - So any duplicate is invalid

Ledger Prepayment

- How to be sure participants pay up?
- Can start with Alice, Bob, Charlie, and Trudy all putting money into the pot
- And don’t allow any transaction that would result in negative balance
- Transaction must still be signed and …
- … now, nobody can “overdraw” account

Ledger Prepayment Example

- Ledger example…
  - Alice has $100 // Alice's initial stake
  - Bob has $100 // Bob's intial stake
  - Charlie has $100 // Charlie's initial stake
  - Trudy has $100 // Trudy's initial stake
  - $[\text{Bob owes Alice } \$10]_{\text{Bob}}$ // **valid**
  - $[\text{Charlie owes Trudy } \$30]_{\text{Charlie}}$ // **valid**
  - $[\text{Trudy owes Alice } \$25]_{\text{Trudy}}$ // **valid**
  - $[\text{Trudy owes Bob } \$120]_{\text{Trudy}}$ // **invalid**

Ledger Prepayment

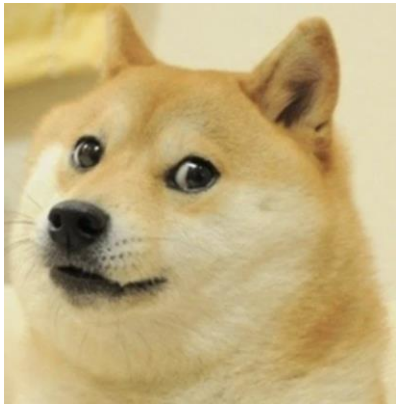
- Note that we must know the ***entire*** transaction history
  - So that we can know current balances
  - Then we can be sure a given transaction does not cause user to be overdrawn
- Hmm… what property we want to enforce on this "entire transaction history"?
  - Confidentiality?
    - Private blockchain (trusted hardware)
    - Zero-knowledge (zk) blockchain
  - Integrity?
  - Availability?

Eternal Ledger?

- Alice, Bob, Charlie, and Trudy could continue to settle accounts each month
- But, as the ledger currently stands, settling accounts is not necessary!
- We know the current balances, and no risk of anyone being “overdrawn”
- So, could play poker for months, years, or forever, without settling accounts

Ledger as Currency

- This ledger can act as its own currency!
  - let's just use symbol Đ
- Transactions ***within*** ledger are all in terms of Đ currency protocol
- Anyone can exchanged ledger currency (i.e., Đ) for USD or HKD or …
  - But, such exchanges occur ***outside*** the ledger currency protocol



Ledger Currency

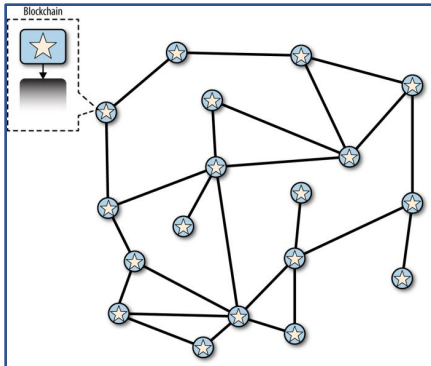
- For example, Alice could pay Bob $10 in real world dollars for, say, Đ 5 of currency in the ledger system
- Comparable to exchanging, say, USD for HKD
- But, ledger is a history of transactions within the ledger currency system
- In fact, ***the ledger is the currency***
  - This is ***one*** key insight for cryptocurrency

Distributed Ledger

- The ledger is the currency
  - So who is in charge of the ledger?
  - A govt? The UN? A bank? An individual?
- We don't trust them, so let's make ***everybody*** in charge of the ledger
  - Anybody can have copy of ledger, anyone can add entries
  - Protocol without a central authority
- What problem(s) do you foresee?

Distributed Ledger

1. Transactions must be signed
2. Nobody can be overdrawn
3. Transactions broadcast to everybody
   - How to have a consistent view of this distributed ledger?
   - Multiple ledgers exist at any time
     - Ledger contains more "works" eventually win
     - Cheaters will have a really hard time to cheat!
     - And no incentive because "cheating" must first pay huge amount of "works", then why bother to cheat?



Preliminaries: Work

- How to measure (digital) ***work*** ?
- Our unit of work will be 1 hash
- Suppose that we have a hash function h(x) that generates an N-bit output
- Then randomly chosen input generates one of 2^N equally likely outputs
  - For any input R, have, $0 \leq h(R) < 2^N$
  - Different R yield uncorrelated hashes

Pricing via processing or combatting junk mail. Cynthia Dwork, Moni Naor. 1992

Hashing to Prove Work

- Suppose we have a 16-bit hash function
- For any R, we have $0 \leq h(R) < 65{,}536$
- If we want R so that $h(R) < 64$, then how many R values do we need to hash?
- Since

$$h(R) = y = (y_{15}y_{14}y_{13}y_{12}y_{11}y_{10}y_9y_8y_7y_6y_5y_4y_3y_2y_1y_0)$$

we want output like (10 leading 0s)…

$$h(R) = y = (0000000000y_5y_4y_3y_2y_1y_0)$$

Work and Hashing

- For 16-bit hash, how many hashes until
  $h(R) = y = (0000000000y_5y_4y_3y_2y_1y_0)$?
- For random R, we have a 1/2 chance that
  $y = (0y_{14}y_{13}y_{12}y_{11}y_{10}y_9y_8y_7y_6y_5y_4y_3y_2y_1y_0)$
- And 1/4 chance that
  $y = (00y_{13}y_{12}y_{11}y_{10}y_9y_8y_7y_6y_5y_4y_3y_2y_1y_0)$
- And 1/8 chance that
  $y = (000y_{12}y_{11}y_{10}y_9y_8y_7y_6y_5y_4y_3y_2y_1y_0)$
- And so on…

Work and Hashing

- For 16-bit hash, if someone gives us an R such that h(R) < 64
  - Then expected number of hashes computed is 2^{10} ("expected" means average case)
  - That is, they have done 1,000 units of work
- We use hashing to show work was done

Work and Hashing

- It's hard to compute the "R" for the hash we want.
- ***Note***: We can easily verify that the expected amount of work was done
  - Only requires a single hash
  - No matter how much work to find R
- Furthermore, we can adjust parameter so more work (or less) is required
  - Instead of requiring 10 leading zeros, we require 9, then the work required is largely reduced.

Distributed Ledger and Work

- Every ledger will have some amount of work associated with it
- Ledger with most work “wins”
  - That is, everyone accepts ledger that has the most work put into it

❑ Recall, work is measured in hashes

❑ So, more hashes is “more better”

  ❑ Better computers have higher “hash rates"

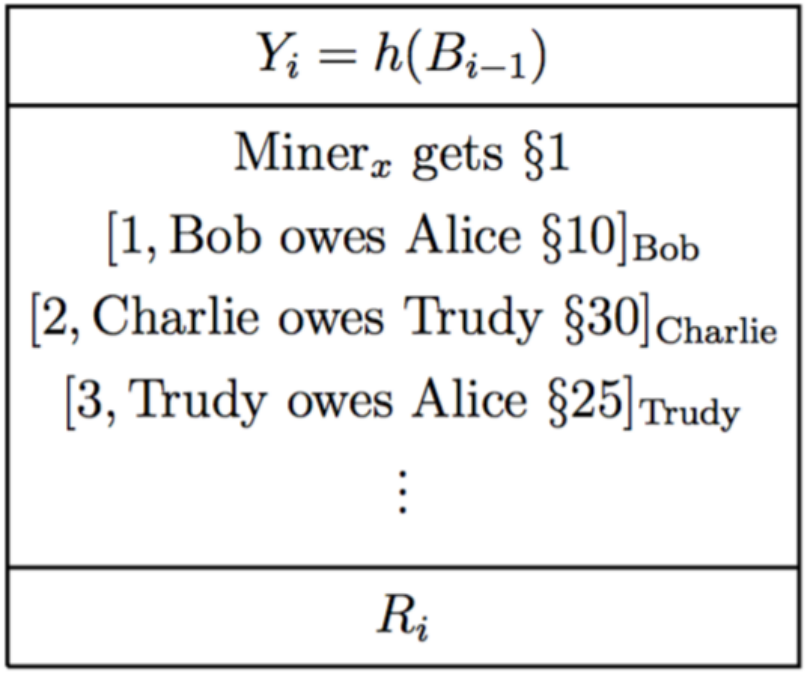
Global transaction ledger

Sender	Receiver	Bitcoin amount	
Alice	Bob	0.31	Block 1
Carol	Bob	1.21	
Alice	Bob	0.4	
Bob	Alice	0.532	Block 2
Carol	Bob	0.01	Block 4
Bob	Carol	0.01	
...	...	...	
...	...	...	

- Everyone agrees on the same ledger
  - Update each other on new transactions
- Divided into blocks, 1 block every 10 minutes

One Block

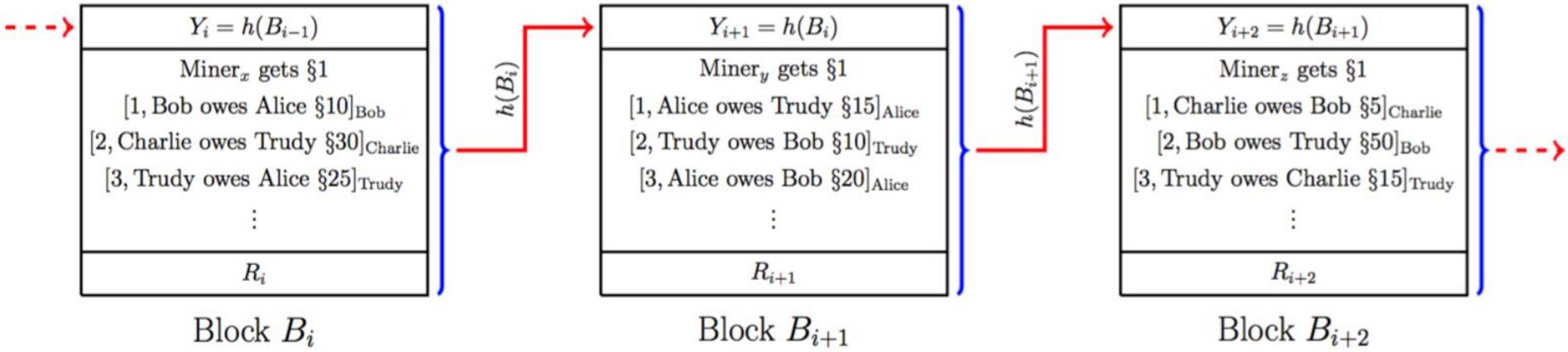
- Block B_i looks like…



Block B_i

Blockchain

- Blockchain looks like…



https://www.blockchain.com/explorer?view=btc

Chain & New Block

- Don't want to revalidate each block, want to order blocks, and so on
- We'll ***chain*** blocks together
  - Put hash of previous block in header of current block
- When can we (miner) add a new block?
  - When we find a R so that $h(Y,B,R) < 2^n$ (Proof of Work)
  - Where Y is hash of previous block
  - B is the new block that is going to be added to the chain
  - This is the so-called mining!

Mining?

- Anyone can create a new block
- But lots of work to find a valid hash
- So what is the incentive to do work?
- “Free” money!
  - Get (new) money for doing work, say, Đ 1
  - Transaction fee to include each transaction

Mining

- Free money, so miners are in a race to find hashes that yield valid blocks
- The more computing power (hash rate) a miner has, the better chance to win race
- Once a valid hash is found, miner sends the block out to everybody
- Again, easy to verify hash is correct
- A huge waste (?) of electricity money.

Mining

- Why is “mining” called mining ?
  - Really, just finding a valid block hash
- Miner is doing work, and creating new money that did not previously exist
  - In a sense, this is comparable to mining gold or silver (for example)
- This may be the most misunderstood (?) part of cryptocurrency protocols

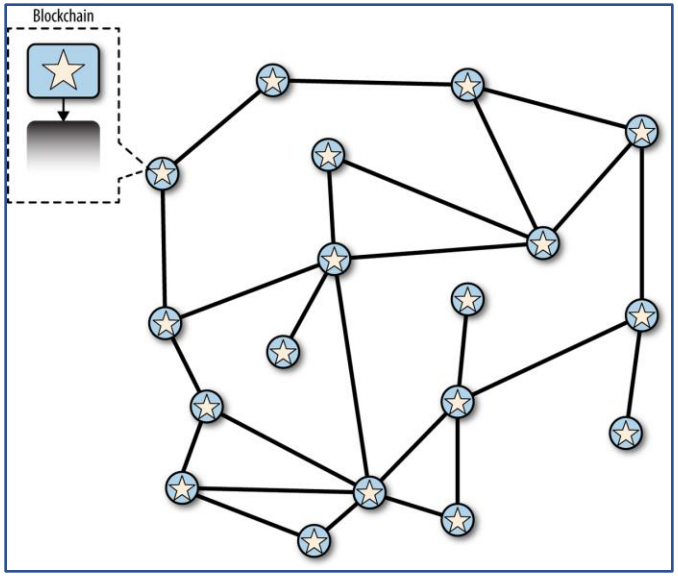
Non-Miners

- Users do not have to be miners
- Non-miner just wants blockchain
  - Needed to know how many Đ others have
- Also, non-miner sends out transactions for others to make blocks (and mine)
  - Need to pay transaction fee.
- User might see conflicting blockchains
  - What to do in such cases???
  - We will see…
- More work is “more better”!

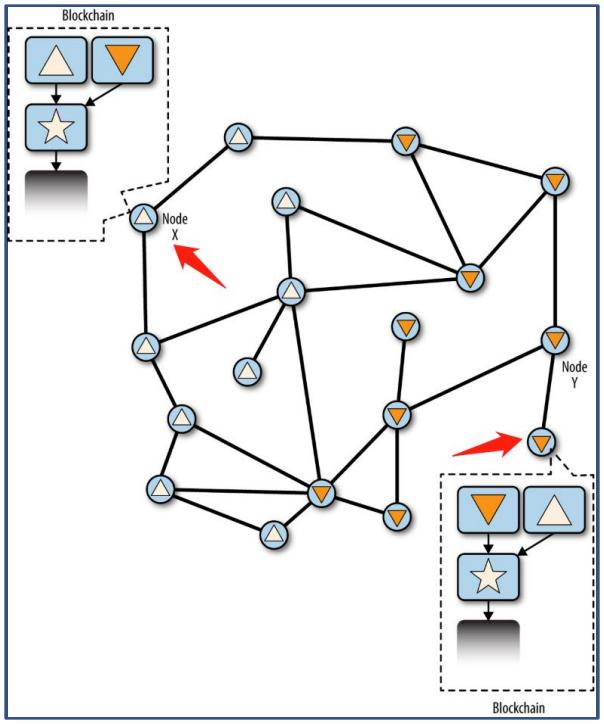
More Work

- If conflicting blockchains, how to know which represents more work?
- Each block is a fixed amount of work
  - In terms of expected number of hashes
- So, longer block chain is more work
- Thus, ***longer*** block chain always wins
  - If it's a tie, wait until one is longer

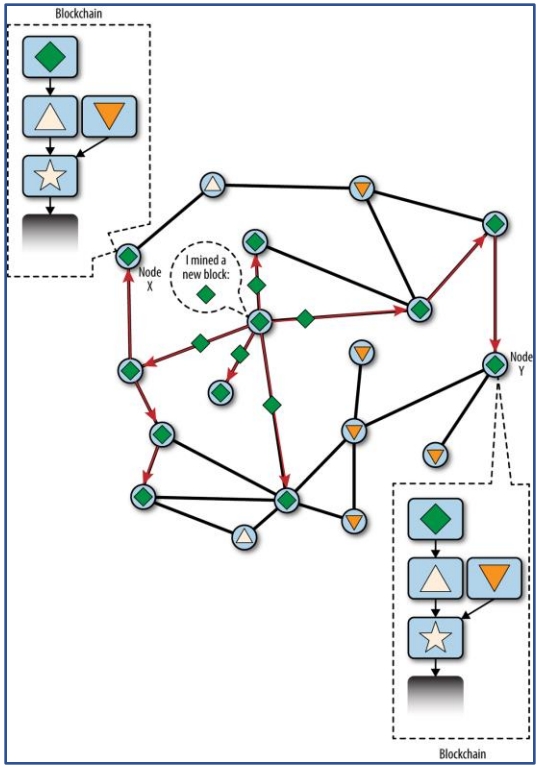
Blokchain Ties and Forks



(1) A blockchain



(2) Temporarily hold two chains of equal length



(3) Discard the shorter chain

As a “angry” miner whose block got discarded, what to do?

Attack Scenario (Double Spending)

- Suppose Trudy makes a block B that includes transaction
  - $[\text{Trudy pays Alice §100}]_{\text{Trudy}}$
  - Trudy sends B to Alice ***only***, nobody else
- **Q**: Why would Trudy do this?
- **A**: So she can spend that §100 again
  - Trudy likes **double spending**!
  - It's free money!

Double Spending

- For Trudy's double spending attack to work, she must compute valid hash
  - That is, find R, so that $h(Y,B,R) < 2^n$
- And send chain with block B to Alice
- But, nobody else knows about B, or the chain that contains it
  - All other miners working on other chains
  - Those other chains can (and will) grow
  - Trudy is in ongoing race with ***all miners***
  - Note that once Trudy's chain left behind, his attack will be discarded so nothing can be obtained by Trudy.

Double Spending Attack

- Alice will reject Trudy's chain once a longer chain appears
- Again, Trudy would need most of computing power in the network to win (51% to be ahead of others)
  - Trudy needs to win a lot!
  - Will explain "51%" later

Blockchain

- From users perspective…
  - Transaction in last (latest) block might not be entirely trustworthy
  - Possibility of double spending attack
  - But, the more blocks that follow, the more certain that a transaction is valid
- Just wait until a few more blocks are added before accepting a transaction
  - The common practice is that you need to wait for 2~3 blocks to really “confirm” this transaction.
  - But I never know if any chain can go three blocks ahead of the main chain…. Using quantum computer??

51% Attack

$$B_2 \rightarrow B_3 \rightarrow B_4$$

What if the miner change $B_2 \rightarrow B_2'$ when B_4 is the most recent block?

→ The miner must provide R'_2, R'_3, R'_4

→ Meanwhile, all miners are generating R_5….

→ If they generate R_5 before the malicious miner finishes, then the miner has to generate R_5 as well…

→ In general, for the malicious miner to catch up in this race, it should have sufficient computing power to win against ALL the remaining "benign" miners:

$$(3 + n) : n$$

when n is getting large, then it's slightly over 50% (i.e., 51%)

Summary of Protocol

1. New transactions broadcast
2. Miners collect transactions into blocks
3. Miners race to find valid block hash
4. When miner finds hash, broadcast it
5. Block accepted if all transactions signed, no overdraft, & block hash valid
6. New block extends blockchain
   - Miners use hash of new block in next block

Bitcoin

- Reward
  - Originally 50 per block → Now 12.5 bitcoins (will keep halving every few years)
  - 16/21 million have been mined
  - On average, 1 proof (= 1 block) will be found every 10 minutes (how can we ensure this?)

Bitcoin

- Transaction fees
  - Miners need to be motivated to include transactions
  - Solves another problem: as reward lowers and difficulty rises, miners need another source of income
- Transaction delay
  - Need to wait for blocks (several to be safe)
  - Each block is 10 minutes

One More Thing…

- Bitcoin makes a lot of millionaires?
- But more people got bankruptcy.

The smartest thing is to stay away from it?
Or at least don't do "leverage" (or so called "margin call")

币圈一天，人间一年 → Doing Bitcoin investment one day can get you lost your annual income.