BLOCKCHAIN PSEUDO_BITCOIN_1

指導老師：林宗男 教授
助教： 陳秉珏
r08942078@ntu.edu.tw

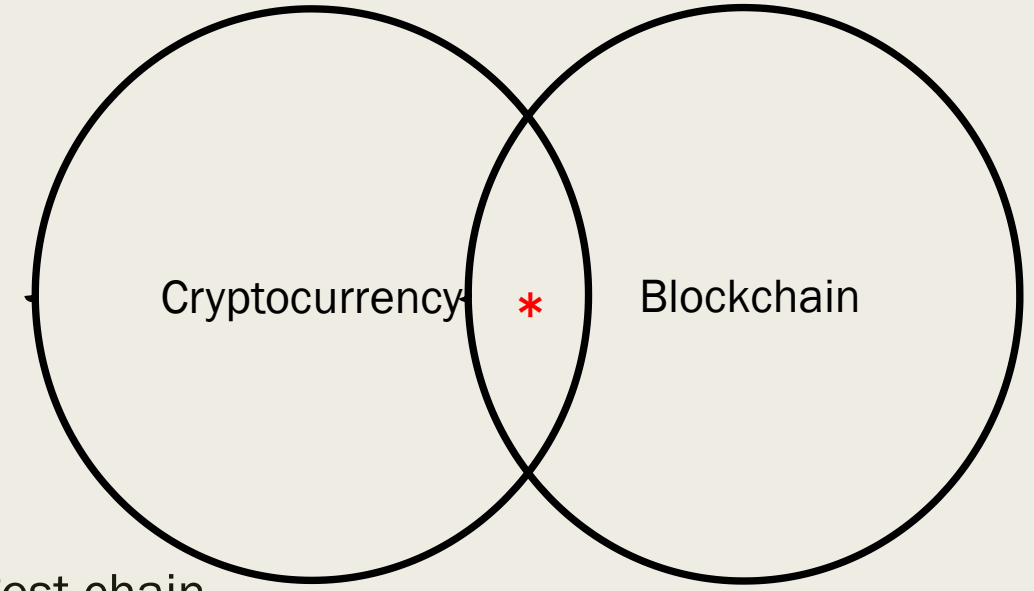
Schedule (5 weeks)

1. 09/18	Blockchain101	
2. 10/02	Pseudo-Bitcoin (part 1)	Announcement of HW1
3. 10/16	P-Bitcoin (part 2), Smart Contract (part 1)	Announcement of HW2-1
4. 10/30	Smart Contract (part 2)	Announcement of HW2-2
5. 11/13	Smart Contract (part 3) & Demo your HWs	

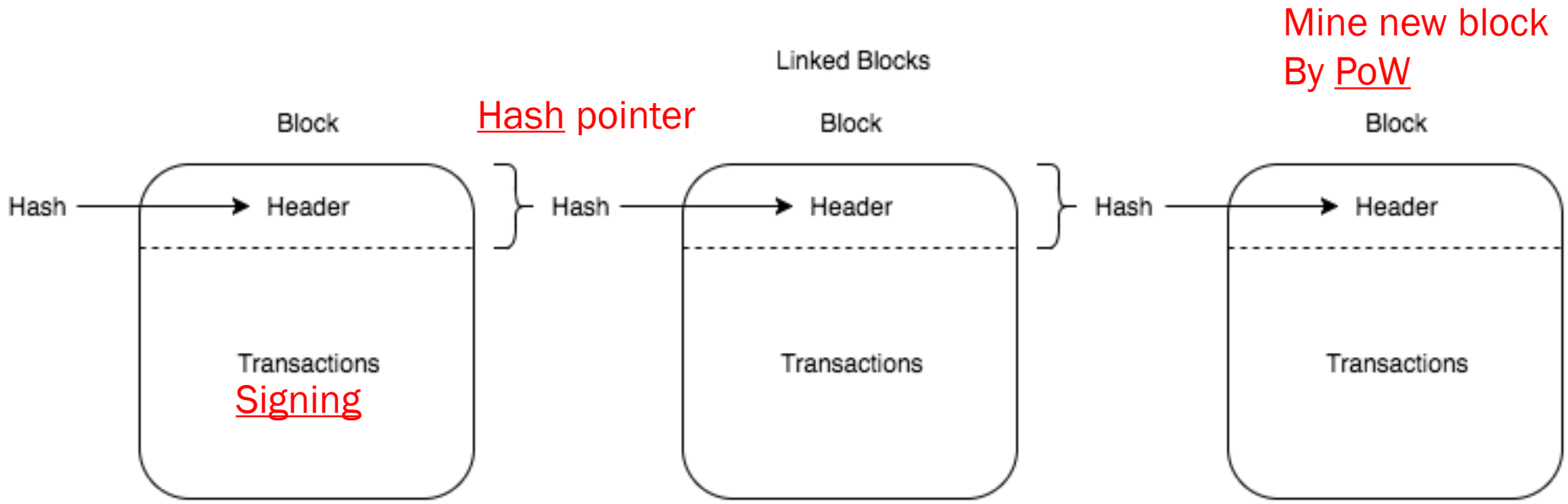
REVIEW

Key points, “bonus” ans.

Review key points

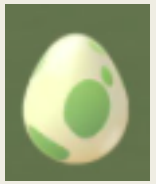


- Blockchain: distributed timestamp server
- Bitcoin(*): distributed ledger for cryptocurrency
- Final consistency: miner preferred to mine the longest chain



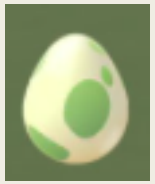
Prev. Bonus

E-1 Bitcoin在2009/1/3的創世區塊中紀錄下了什麼訊息？(Hint1: 一則新聞)



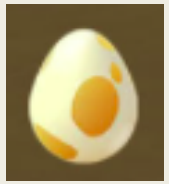
4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b

E-2 岳昕依法規申請北大公開教授性侵女學生事件而遭到迫害，這名教授的名字是？



0x2d6a7b0f6adeff38423d4c62cd8b6ccb708ddad85da5d3d06756ad4d8a04a6a2

M-1 在最多允許n個作惡節點的情況下，能確保達成一致的拜占庭系統節點至少需m個。試求n與m的關係式。



H-1 對Bitcoin來說，若整個系統每秒能運算r次雜湊函數計算新的nonce（其他運算時間可省略），時間t內挖到有效區塊的機率為何？（Hint: 已知當$\theta \ll 1$時，$\log(1-\theta) \approx -\theta$。）

M-2 承H-1，若對於礦工i來說，每秒能運行r_i次運算，由他挖出新區塊的機率為何？

(一般來說會認定Hash(.)的輸出是完全隨機)

HW1

Deadline: (GMT+8) 2019/10/30 23:59

Pseudo Bitcoin

＊ : Basic blockchain
＊ : Cryptocurrency

- Please Use **Python3**

#	Requirement	Description	Score
1	Prototype	Block(10%), Blockchain(10%), Proof-of-Work(20%)	40%
2	Persistence	Database(20%), Client(20%)	40%
3	$\text{Transaction}_{\text{basic}}$	UTXO(5%) or Account model(2%)	5%
4	Address	Sign & Verify(5%)	5%
5	$\text{Transaction}_{\text{advanced}}$	Mining reward(2%), Merkle tree(8%)	10%
6	Network	P2P(10%) or Server-Client(7%)	10%
	Other features	Proof of ???, Special design	5%

- Deadline: 10/30 23:59　　Demo: 11/13

(^: If not follow the rule)

HW1 Submission Rules

- 0. Upload on time (^ score*=0.7 per week)
- 1. Name your project directory: b05901xxxBTC (^ score-=2)
- 2. A Readme file in your project directory:
- 2.1 **Prerequisites** (^ score-=5)
- 2.1 **How to use** your pseudo bitcoin (^ score-=5)
- 2.2 The **functionalities** you've implemented (^ score-=5)
- 3. Compress your project directory to .zip (^ score-=5)

0 RESOURCES

Useful resources

Importance: ★ ★ ★ ★ ★ ★ ★ ★ ★

- *Javascript:* *A blockchain in 200 lines of code*
- *C++:* *tko22/simple-blockchain*
- *Python:* *yummybian/blockchain-py* ← *Clear, Suggestion for you!*
- *Python:* *https://zwindr.blogspot.com/2018/05/python.html?m=1*
- *Python, Go:* *liuchengxu/blockchain-tutorial* ← *Chinese tutorial (if you feel confused…)*
- *Go:* *Jeiwan/blockchain_go* ← *TA's main resource, good articles*

Feel free to refer to these resources, but it's important to understand scripts by yourself.

1_1 PROTOTYPE

A timestamp server by PoW

E-1 已知Bits在bitcoin中代表挖礦的target T， 若hash的輸出是64bits number, 則隨機取一個nonce， 挖到礦的機率為何？（用T表示）

Hint: 可參考p.20 對Bits的進一步解釋

Inside One Bitcoin Block

- Magic number: 0xD9B4BEF9
- Blocksize
- Height
- Blockheader:
  - *1. Version*
  - *2. HashPrevBlock*
  - *3. HashMerkleRoot (in next class)*
  - *4. Time*
  - *5. Bits*
  - *6. Nonce*
  - *(7. Hash)*
- (Transaction counter)
- Transactions

One page of the ledger

For convenience, we put block.header and block.transaction all together!

It’s time to code

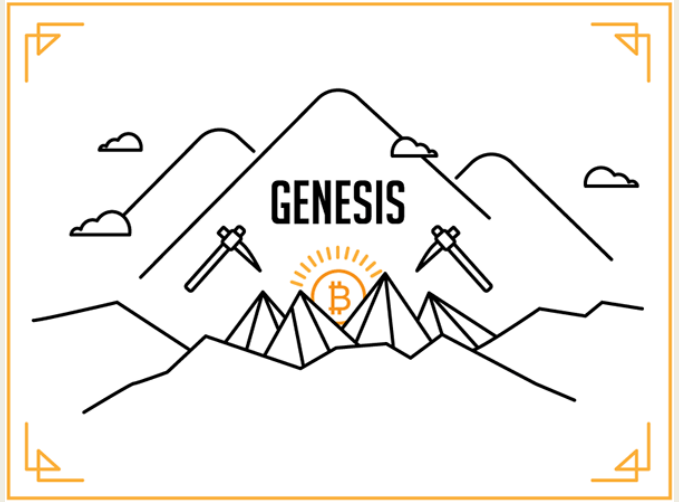
```
// Block keeps block headers
type Block struct {
    Height        int64
    PrevBlockHash []byte
    Time          int64
    Bits          int64
    Nonce         int64
    Transactions  []byte
    Hash          []byte
}
```

- “struct” is like “class” in oop language (but it’s not the same concept !)

- In python, it might look like...
- class block(object):
- def __init__(self, transaction, ...):
- self.transaction = ...
- self.time = ... # Use system time
- self.bits = ... # Difficulty: a constant
- ...

Build a Blockchain from Genesis Block

- Blockchain:
- $block_1$ – $block_2$ – $block_3$ – $block_4$ - … - $block_i$ - …



- Genesis block: $block_1$, first block in blockchain
- Write a function to initialize the blockchain

- Genesis block doesn’t contain prev. block, you may set an empty value here

It’s time to code

- Write functions

- newBlock(...): to initialize a new block
- newGenesisBlock(): return a genesis block (next page)

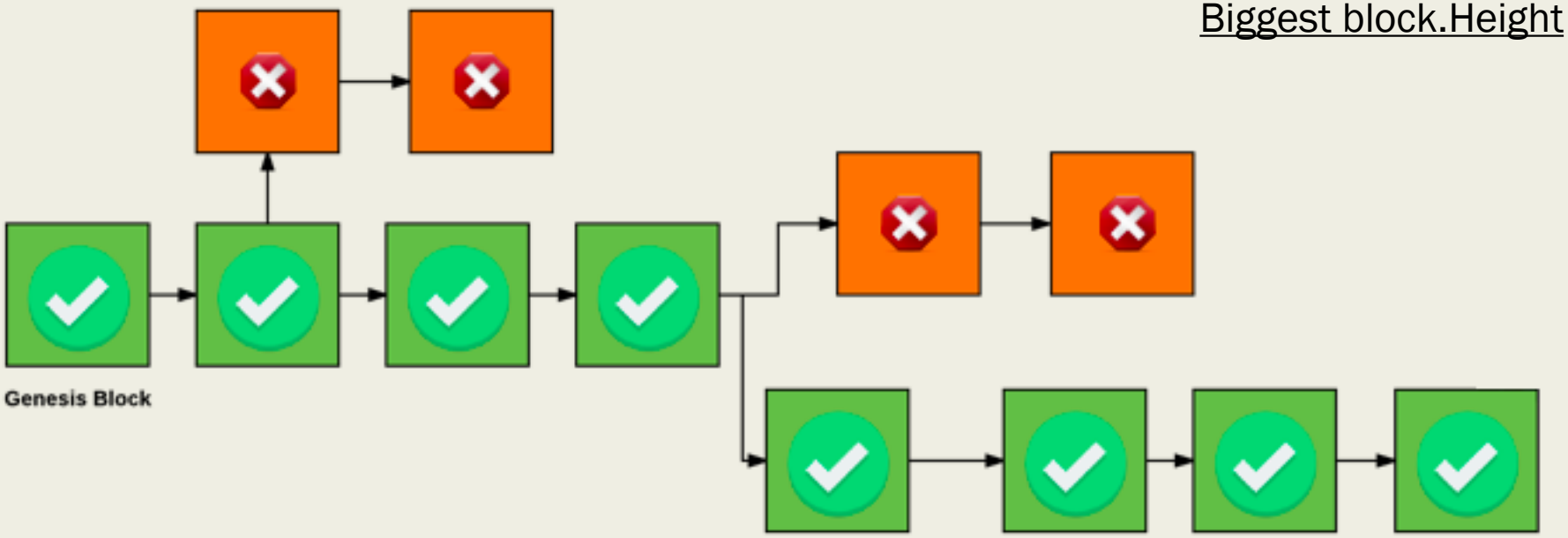
- pow is the consensus algorithm to mine new block

```go
// NewBlock creates and returns Block
func NewBlock(transaction string, prevBlockHash []byte, prevHeight int64) *Block {
    block := &Block{
        prevHeight + 1,
        prevBlockHash,
        time.Now().Unix(),
        20,
        0,
        []byte(transaction),
        []byte{},
    }
    block.SetHash()
    return block
}

// NewGenesisBlock creates and returns genesis Block
func NewGenesisBlock() *Block {
    return NewBlock("創世區塊：\n歡迎來到網多實驗室", []byte{}, 0)
}
```

Other Blocks

- Append new blocks -> addBlock()
- Remember to add prevHash
- Always mining the longest chain (Now we just mine after the latest block)

Biggest block.Height



It’s time to code

- You need to write a blockchain object
- And there is a function add_block() inside
- In python, it might look like...
- class blockchain(object):
- def __init__(self):
- ...
- self._blocks = ... # a list here
- def add_block(self, data):
- ...
- self._blocks.append(...)

```go
// Blockchain keeps a sequence of Blocks
type Blockchain struct {
    blocks []*Block
}

// AddBlock saves provided data as a block in the blockchain
func (bc *Blockchain) AddBlock(transactions string) {
    prevBlock := bc.blocks[len(bc.blocks)-1]
    newBlock := NewBlock(transactions, prevBlock.Hash, int64(len(bc.blocks)))
    bc.blocks = append(bc.blocks, newBlock)
}

// NewBlockchain creates a new Blockchain with genesis Block
func NewBlockchain() *Blockchain {
    return &Blockchain{[]*Block{NewGenesisBlock()}}
}
```

H-1 若Proof of Work使用hash(nonce) ≤ M/D， 最可能會遭受哪種攻擊？請說明。

A real “chain” by “Proof of Work” (Bitcoin)

- hash(nonce) ≤ M/D
- Better: hash(nonce) → hash(Block)
- (hash(·) ∈ [0, M], D∈ [1, M])

- Resistant to these possible attacks:
  - *Double spending*
  - *Mining on multiple chains*
  - *Grinding*
  - *……*

It’s time to code

- We need to modify newblock() first

- Why?
- Hash value of block would change with different nonce.

```go
// NewBlock creates and returns Block
func NewBlock(transaction string, prevBlockHash []byte, prevHeight int64) *Block {
    block := &Block{
        prevHeight + 1,
        prevBlockHash,
        time.Now().Unix(),
        20,
        0,
        []byte(transaction),
        []byte{},
    }
    block.SetHash()
    return block
}

// NewGenesisBlock creates and returns genesis Block
func NewGenesisBlock() *Block {
    return NewBlock("創世區塊：\n歡迎來到網多實驗室", []byte{}, 0)
}
```

```go
pow := NewProofOfWork(block)
nonce, hash := pow.Run()

block.Hash = hash[:]
block.Nonce = nonce
```

Proof of Work as a struct (or object)

- Remember Block?
- Now we could use its “Bits” as our PoW target bit
- If Bits = 11, below are some valid hash(Block):

```
001d649481a51be47f65c06b456c25074f7e9778a478363cc93fb83c4b122699  // 256-bits
0008175784966l0ef5d79ddeaf52fa08103b2b91ead1e2ef84fb8d47db270d6b  // 256-bits
```

```
// Block keeps block headers
type Block struct {
    Height        int64
    PrevBlockHash []byte
    Time          int64
    Bits          int64
    Nonce         int64
    Transactions  []byte
    Hash          []byte
}
```

- Now we need 3 functions in PoW(object):
  - *pow.prepareData(nonce): return data* *// call it when we try new nonce*
  - *pow.Run(): return nonce, hash* *// compare hash(block) and our target*
  - *pow.Validate(): return true/false* *// check whether pow satisfy the inequality or not*

It’s time to code

- Here just a brief explanation about pow.Run()

```
for nonce < maxNonce {
    data := pow.prepareData(nonce)

    hash = sha256.Sum256(data)
    fmt.Printf("\r%x", hash)
    hashInt.SetBytes(hash[:])

    if hashInt.Cmp(pow.target) == -1 {
        break
    } else {
        nonce++
    }
}
fmt.Print("\n\n")

return nonce, hash[:]
```

M-1 左側範例中，萬一nonce增為maxNonce（溢位），回傳的結果會是合法的？一個礦工此時會怎麼做？

- In python, you may write:

- class pow(object):
- def __init__(self, block):
- self._block = block
- self._target = 1 << (256 – block.bits)
- def prepare_data(self, nonce):
- ...
- def run(self):
- ...
- def validate(self):
- ...

```
type ProofOfWork struct {
    block  *Block
    target *big.Int
}
```

1_2 PERSISTENCE

Database & Client

Database (Bitcoin core version)

It's free for you to design your own database

- In real Bitcoin, there are two buckets:
  - *blocks*
  - *chainstate*

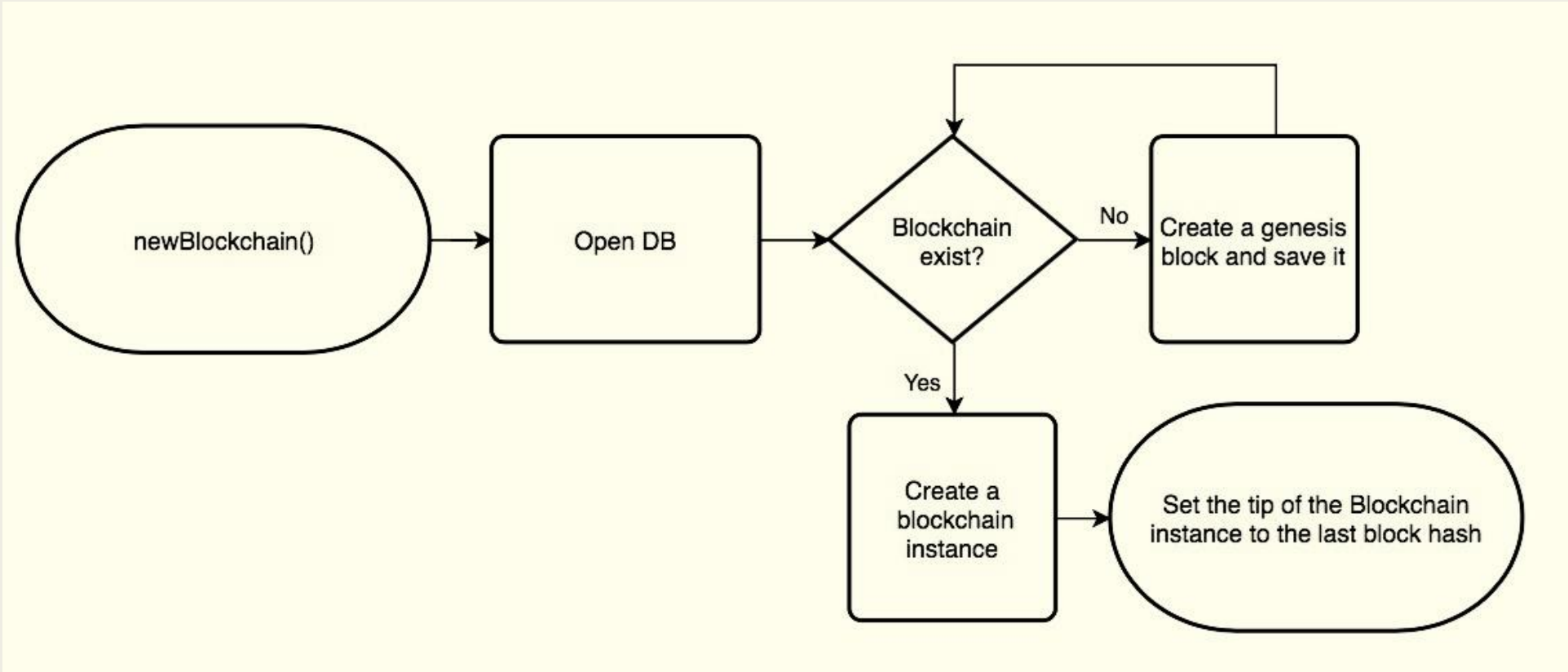
- It stores different blocks in different file
  - *In bucket of blocks:*
- $block_1$
- $block_2$
- …

M-2 如果比特幣礦工從其他節點獲取了兩個不同的blockchain DB，此時應該怎麼做才能保證利益最大化，應在兩支區塊鏈同時挖礦或是放棄其中一支？試說明。

Database (TA's implementation)

It's free for you to design your own database

- Suggestion: Key-Value pair database
- Remember to encode/decode your data
-



CLI

- Requirements:
- ./pseudoBitcoin addblock –transaction {“blablabla”}
- ./pseudoBitcoin printchain
- ./pseudoBitcoin printblock –height { height }

Remember to do Error handling!

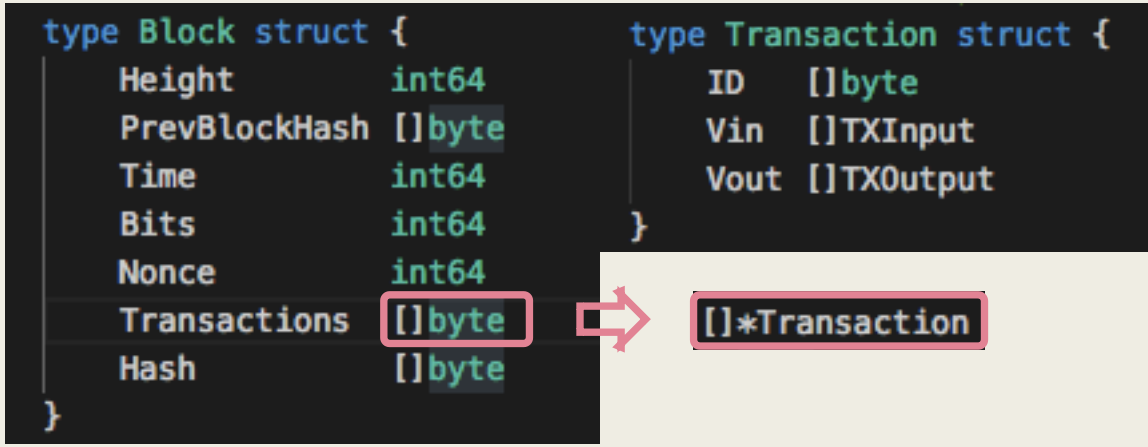
1_3

BASIC_TRANSACTION

Transaction model

Difference between basic blockchain

- Design for usage in Cryptocurrency



- Issues:
  - *How a transaction look like in this system?*
  - *How to spend money?*
  - *How to store so many transactions inside one block?*
  - *How to prevent double spending attack?*
  - *…*

It’s time to code

- Modify “block” first:
- If you use python,
- class block(object):
- def __init__(self, data, ...):
- ...
- def hashTX(self):
- ...
- return hash(self.transactions)
- And remember to modify pow.prepare_data(...) using block.hashTX(...)

```
func NewBlock(transactions string, prevBlockHash []byte, prevHeight int64) *Block {
    block := &Block{
        prevHeight + 1,
        prevBlockHash,
        time.Now().Unix(),
        18,
        0,
        []byte(transactions),
        []byte{},
    }
    pow := NewProofOfWork(block)
    nonce, hash := pow.Run()

    block.Hash = hash[:]
    block.Nonce = nonce
    return block
}
func (b *Block) HashTransactions() []byte
```

[]*Transaction

transactions,

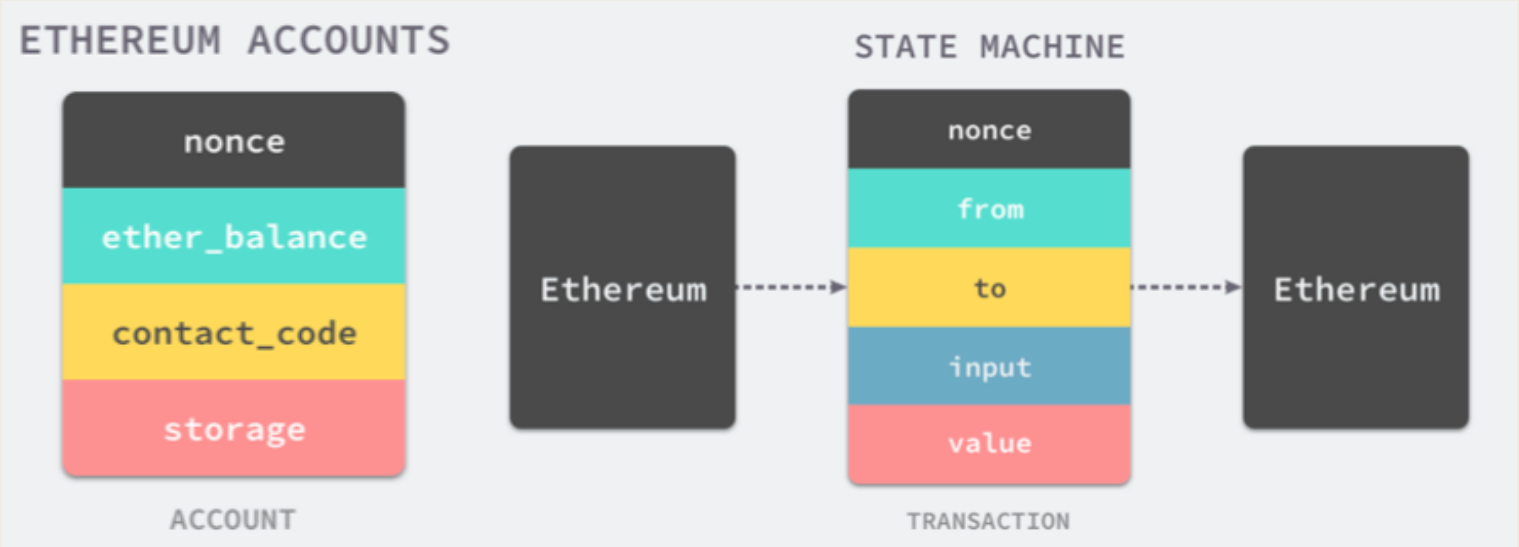
python -> block.hashTX()

Implementation in Account model

(Easy to understand, but little discussion in this class)

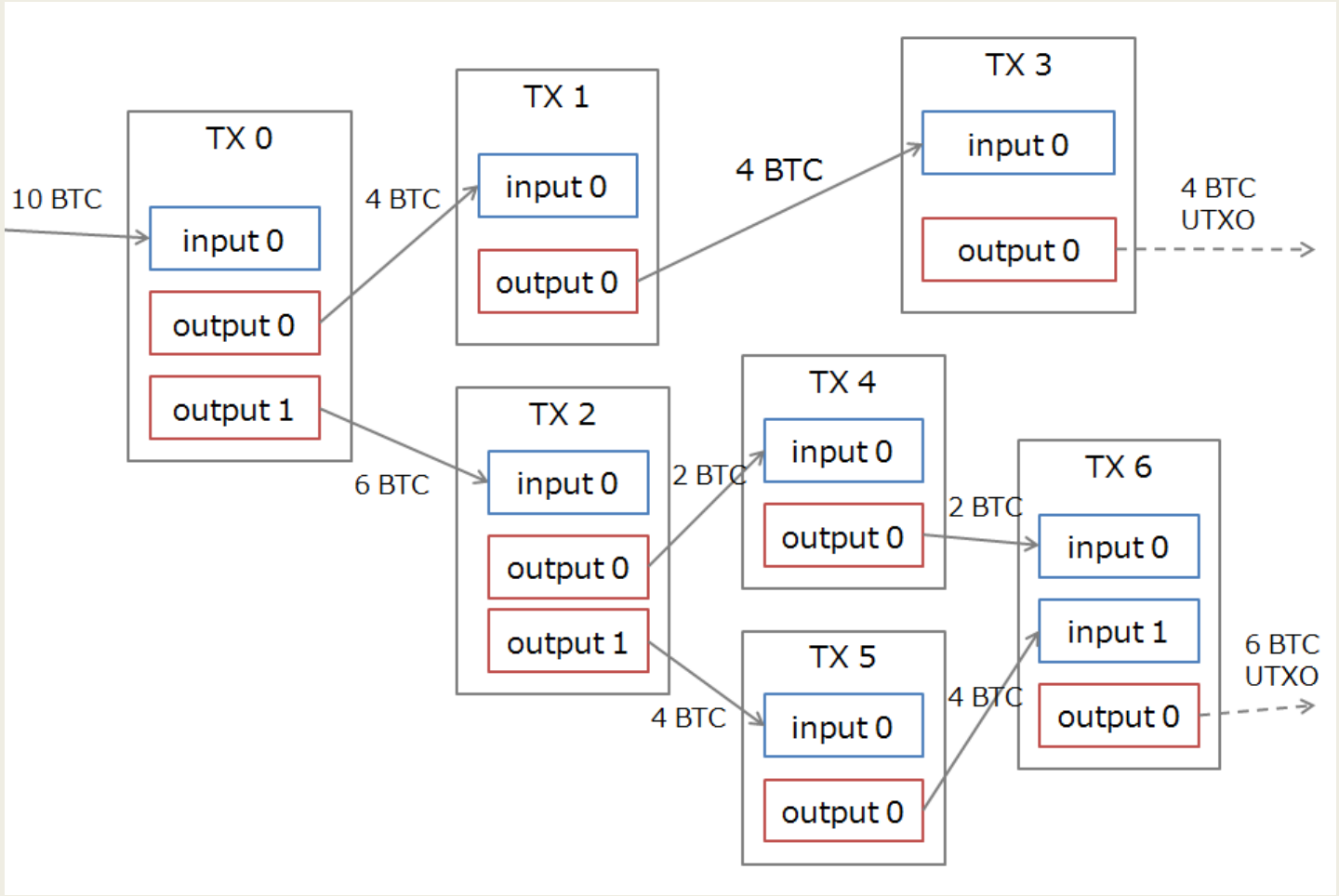
- Account address
- Account balance

- Ex. Ethereum

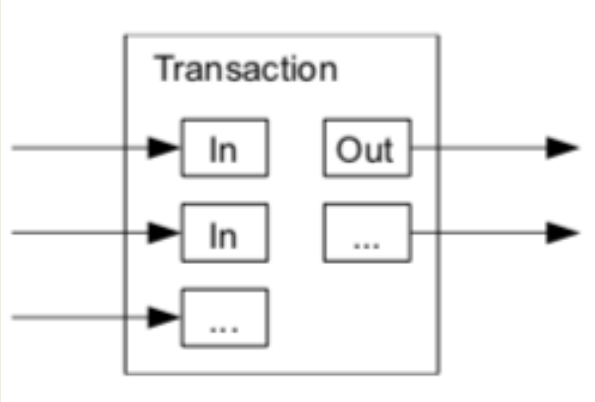


M-3 在加密貨幣中，常見有Account Model、UTXO Model和混合模型，哪一種模型比較支援帳號同時數筆平行交易？哪一種模型比較能夠快速取得帳戶餘額？試說明（可以舉例）。

Implementation in UTXO model



- In Bitcoin...



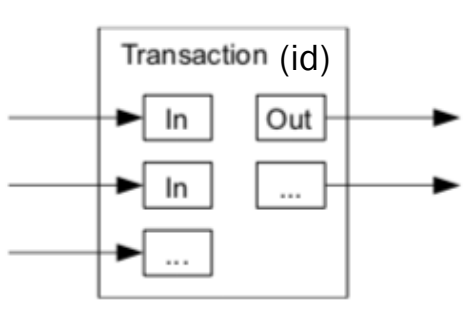
It's time to code

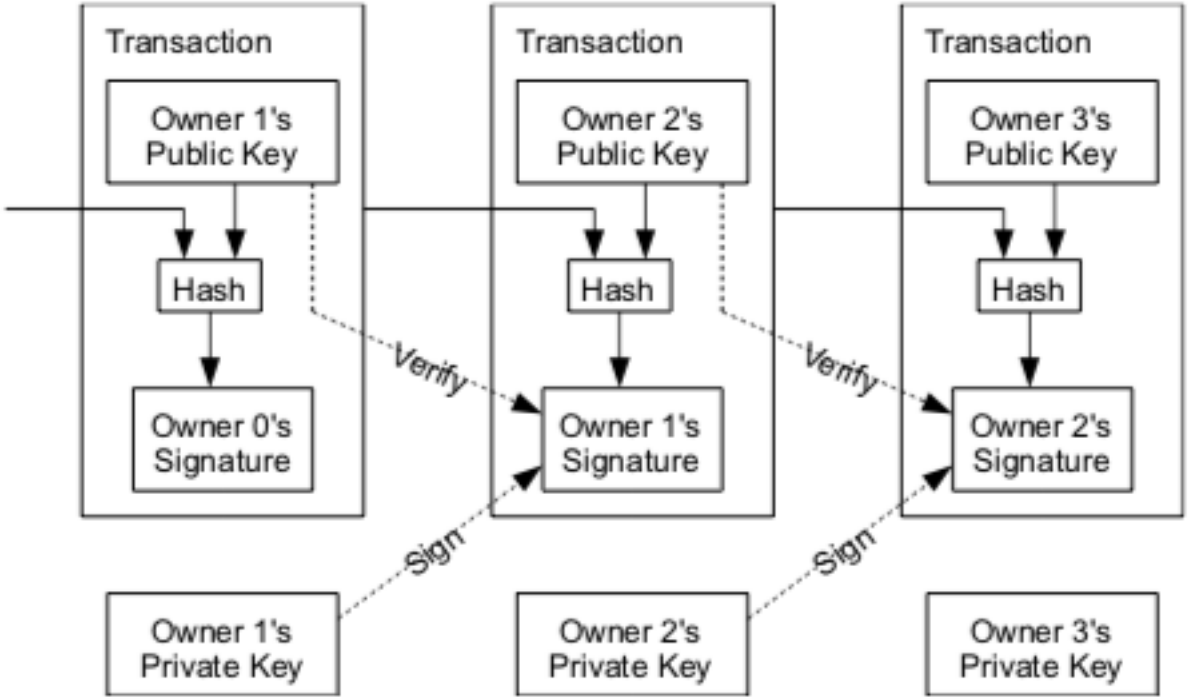
- Now let's implement "transaction"

```
type TXInput struct {
    Txid      []byte
    Vout      int
    ScriptSig string
}

type Transaction struct {
    ID   []byte
    Vin  []TXInput
    Vout []TXOutput
}

type TXOutput struct {
    Value        int
    ScriptPubKey string
}
```

TXOutput
1. Store unspent money
2. Couldn't be cut into pieces

- We will implement "address" in next class, here "ScriptSig" & "ScriptPubKey" are just user-defined data.





Coinbase transaction

- Aka. Generation transaction
- 【Coinbase transaction】 The first transaction in a block. Always created by a miner, it includes a single ”coinbase”.
- 【Coinbase】: A special field used as the sole input for coinbase transactions. The coinbase allows claiming the block reward and provides up to 100 bytes for arbitrary data.

- Subsidy(reward) = 50 * 0.5 ^ int(Height/21000)
  - *But in this HW, you can just set subsidy as a constant*

It’s time to code

```go
func NewCoinbaseTransaction(to, data string) *Transaction {
    if data == "" {
        data = fmt.Sprintf("Reward to '%s'", to)
    }

    txin := TXInput{[]byte{}, -1, data}
    txout := TXOutput{subsidy, to}
    tx := Transaction{nil, []TXInput{txin}, []TXOutput{txout}}
    tx.SetID()

    return &tx
}
```

- tx.SetID() assign the hash value of Tx to Tx.ID

It’s time to code

- Steps:
  - *Check whether the money is enough or not*
  - *Build a list of inputs*
  - *Build a list of outputs, and if sum of all inputs > sum of all outputs, append a new outputs (make change)*
  - *Create a transaction*
  - *Return this transaction*
- Add this function output to the new block when you find a PoW answer.

```go
func NewUTXOTransaction(from, to string, amount int, bc *Blockchain) *Transaction {
    var inputs []TXInput
    var outputs []TXOutput

    acc, validOutputs := bc.FindSpendableOutputs(from, amount)

    if acc < amount {
        log.Panic("ERROR: Not enough funds")
    }
    // Build a list of inputs
    for txid, outs := range validOutputs {
        txID, err := hex.DecodeString(txid)
        for _, out := range outs {
            input := TXInput{txID, out, from}
            inputs = append(inputs, input)
        }
    }
    // Build a list of outputs
    outputs = append(outputs, TXOutput{amount, to})
    if acc > amount {
        outputs = append(outputs, TXOutput{acc - amount, from}) // a change
    }

    tx := Transaction{nil, inputs, outputs}
    tx.SetID()

    return &tx
}
```

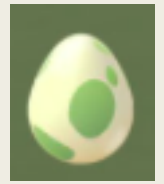
CLI

- Requirements:
- ./pseudoBitcoin createblockchain –address { YOUR_NAME }
  - *If call createblockchain(), it should create a blockchain with a coinbase TX (reward to {address}) inside the genesis block.*
- ./pseudoBitcoin getbalance -address { ADDRESS }
  - *Use function blockchain.FindUTXO().*
- ./pseudoBitcoin send –from { $NAME_1$ } –to { $NAME_2$ } –amount { HOW_MUCH }
  - *Append new block containing one UTXO transaction inside.*
- ./pseudoBitcoin printchain
- ./pseudoBitcoin printblock –height { HEIGHT }

BONUS

Bonus

E-1 已知Bits在bitcoin中代表挖礦的target T，若hash的輸出是64bits number，則隨機取一個nonce，挖到礦的機率為何？（用T表示）

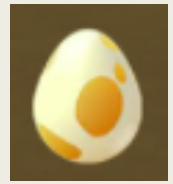


Hint: 可參考p.20 對Bits的進一步解釋

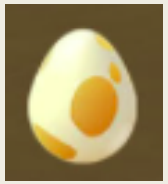
H-1 若Proof of Work使用hash(nonce) ≤ M/D，最可能會遭受哪種攻擊？請說明。

H-2 作業一中，Bits可設為一個常數，若希望像比特幣一樣平均每10分挖出一個block，應如何設計？

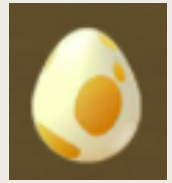
M-1 左側範例中，萬一nonce增為maxNonce（溢位），回傳的結果會是合法的？一個礦工此時會怎麼做？



M-2 如果比特幣礦工從其他節點獲取了兩個不同的blockchain DB，此時應該怎麼做才能保證利益最大化，應在兩支區塊鏈同時挖礦或是放棄其中一支？試說明。



M-3 在加密貨幣中，常見有Account Model、UTXO Model和混合模型，哪一種模型比較支援帳號同時數筆平行交易？哪一種模型比較能夠快速取得帳戶餘額？試說明（可以舉例）。



E-2 所以在比特幣裡，是先有雞（TxInput）或先有蛋（TxOutput）？