Diffences between Bitcoin and Ethereum—

⇒ Ethereum: Ethereum is a blockchain-based distributed platform which uses Ether as its cryptocurrency.

Bitcoin: It is the most famous cryptocurrency at this point of time. There is a significant price difference between bitcoin and ether; but ethereum serves as the 2nd most popular platform regarding blockchain-based transactions.

Basic Difference: Bitcoin uses scripting language which supports basic operations such as duplication of top most element in the stack, apply of hash function on the top most element, checking equality or equivaleeeness between two top elements in the stack etc. But it does not support other features which are supported by other programming languages. For example it can not support loop, recursion. So bitcoin scripting language can be compared to a calculator as it bitcoin only supports financial transactions and checks their validity using scripting language.

If bitcoin is considered as a calculator then Ethereum can be considered as a smartphone as it not only supports financial transactions but also provides program executions. It is world's programmable blockchain. whatever other programming languages provide, all these features are provided by Ethreum based programming languages too one such language is Solidity language which is a Turing Complete language. Any programs can be written in this languages and whatever

program will be developed using this programming languages that are to be executed on the blockchain are called smart contracts. So Ethereum supports smart contracts which enables much more blockchain based application other than cryptocurrency transaction.

Other differences:
- Bitcoin runs on SHA-256 hash algorithm, whereas Ethereum makes use of Ethash proof-of-work algorithm.
- Creation of new block in bitcoin ledger takes 10 mins but in Ethereum it takes 12 to 15 seconds.
- Bitcoin blocks supports 1MB of storage each, in case of ethereum there is no storage limit for blocks.

2) Smart Contracts: A smart contract is a computer program executed in a secure blockchain environment that directly controls different kinds of digital assets. In 1994, Nick Szabo, identified the application of decentralized ledger for smart contracts. Smart contracts can be thought of as accounts controlled by code rather than user.

How it works:

Suppose an institution or organization wants to incorporate smart contract in their system. Then all parties related with the contract can be termed as stakeholders. All stakeholders will be treated as external user and they will have account. There are two types of accounts
- Externally owned account (EOA)
- Contracts accounts (CA)

o EOA are controlled by the private key of the stakeholder and it contains an address, a nonce value [how many transactions it has takenpart of], some balance. It does not contain any code. Here address is calculated by public key and private key.

o In case of CA, it is associated with smart contract code consisting of variables, class, methods etc. CA contains address which is created using sender's (contract creater; which can be any stakeholder) address and nonce value. To compute the address Recursive length prefix encoding is done on address and nonce value and then KEC hash is applied and right 160 bits are taken as address. Here nonce is how many contract creation transactions are issued from this account. It also contains storage hash and code hash. Code hash is hash of the code of contract. Storage hash contains hash pointer to the storage which has values of the variables that are used in the code. This contract code is immutable.

o All actions on the Ethereum blockchain are initiated by EOAs.

o Every time a contract account receives a transaction, its code is executed as instructed by the input parameters.

o The code is executed by Ethereum Virtual machine (EVM) on each node for verification, which is deterministic.

There can be two types of transaction -

1) EOA to EOA or CA

2) Contract creation transaction.

Each transaction has many parameters, such as "nonce" "gas price", gasLimit, to, "value", v, r, s", "init or data".

There are four cases of messages—

1) EOA to EOA

2) CA to EOA

3) EOA to CA

4) CA to CA

Hence, the contracts has ability to send message as well as external actors and using this mechanism of transaction transfer and message transfer ~~between~~ among Externally owned accounts and Contract accounts smart contracts are executed on blockchain platform.

GAS in Ethereum: When a contract is executed as a result of being triggered by a message or transaction, every instruction is executed on every node of the network as part of their verification process. This has a cost, for every operation there is a specified cost. Gas is the name for this cost. It is the execution fee that senders of transactions need to pay for every operation made on a ethereum blockchain. Gas is purchased in exchange of ether from the miners that execute the code written in the smart contract.

Primary role: The ethereum protocol charges a fee per computational step that is executed in a contract on transaction to prevent deliberate attacks and abuse of the ethereum network. Every transaction is required to include a gas limit and a fee that it is willing to pay per gas.

- If the total amount of gas used by the computational steps initiated by the transaction, is less than or equal to the gas limit, then the transaction is processed.
- If the total gas exceeds the gas limit, then all changes are reverted, except that the transaction is still valid and miners can collect their fees.
- All excess gas not used by a transaction execution is refunded to the sender as Ether.

Transaction costs are based on 2 factors — ① gasUsed i.e. total consumed gas ② gasPrice i.e. price of 1 unit of gas specified by the transaction.

Total cost = gasUsed * gasPrice

3) In the given problem, three transactions are given out of which two are valid and they are already in blockchain and one transaction is new. We have to check validity of this new transaction w.r.to previous two transactions.

Two valid transactions — T_1, T_2

New transaction — T_3

As we know each transaction consists three parts namely—

1) Metadata [information related to version, no. of input, no. of output, size etc]

2) Inputs [contains hash of previous transaction, index of the previous transaction's output that's being claimed, signature of the claimer]

3) Outputs [Each output has two fields, value that is being transferred and a script containing hash of public key of the receipient]

Inputs and outputs both forms an array and contains scripts.

To validate a transaction (new), we combine the new transaction's input script and the earlier transaction's output script. Bitcoin scripting language check validity using stack-based execution. So to check validity of T_3 first we have consider the transaction given—

Metadata — "hash": "h3", // hash value of transaction T_3 (unique)
"ver": 1, // software version 1
"vin_sz": 2, // no. of inputs = 2
"vout_sz": 1, // no. of outputput = 1
lock_time": 0,
"size": 604, // size = 604.

Inputs: "in": {

```
{
"prev_out": {
"hash": "h1",
"n": 0
}; "scriptSig": "s2 P2"
},
{
"prev_out": {
"hash": "h2",
"n": 0
}; "scriptSig": "s2 P2"
}
}
```

refers to 1st output of T_1

refers to 1st output of T_2

Here, it should be mentioned that in the given question hashes are denoted by h_i, public key as P_i and private key as s_i.

In transaction T_3, for inputs we have two inputs.
For first input we should refer to h1 i.e., the hash pointer to Transaction T_1.
The index of T_1's output that is being claimed here is 0'th output and the first input is signed using the private key s2. Hence, the claimer for this previous Transaction (T_1)'s output has public key as P_2 and private key as s2. Suppose he/she is Y(det).

The first output of T_1 is:

```
"out": [
  {
    "value": "10.12",
    "scriptPubkey": " .... <hash of P2> .... "
  }
]
```

as we know T_1 is valid so by this output script Y (public key = P_2, private key = S_2) is receiving a value of 10.12.

For the second input of transaction T_3, we should refer to h_2 which is the hash pointer of transaction T_2 and output index that should refers here is 1st output of T_2. This input is signed using ~~public key~~ private key S_2. ~~Let~~ ~~has~~ So it is also being signed by Y. So again Y is the claimer here and s/he claims that 1st output value of Transaction T_2 belongs to him/her.

First output of transaction T2:

```
"out":[
        {
          "value":"5.00",
          "scriptPubKey":"... <hash of P2> ..."
        }
]
```

T_2 is a valid transaction, so we need not check its validity but if we want to check we can see that someone (let Z) who has public key as p_3 and private key as s_3 transfers value of 5.00 to Y. Z is receiving some value in transaction T_1's output 1 (second output), that is 5.25, and spending 5.00 from it to Y. Z has 0.25 remaining.

So after seeing T_1 and T_2 we can see that Y has 10.12 from T_1 and 5.00 from T_2, So Y has 15.12. Y is refering to both this output in input of T_3 and signing it with its private key S_2.

Now in the output of T_3 we can see that,

```
"out": [
        { "value": "15.00",
          "scriptPubkey": "..... <hash of P4> ...."
        }
      ]
```

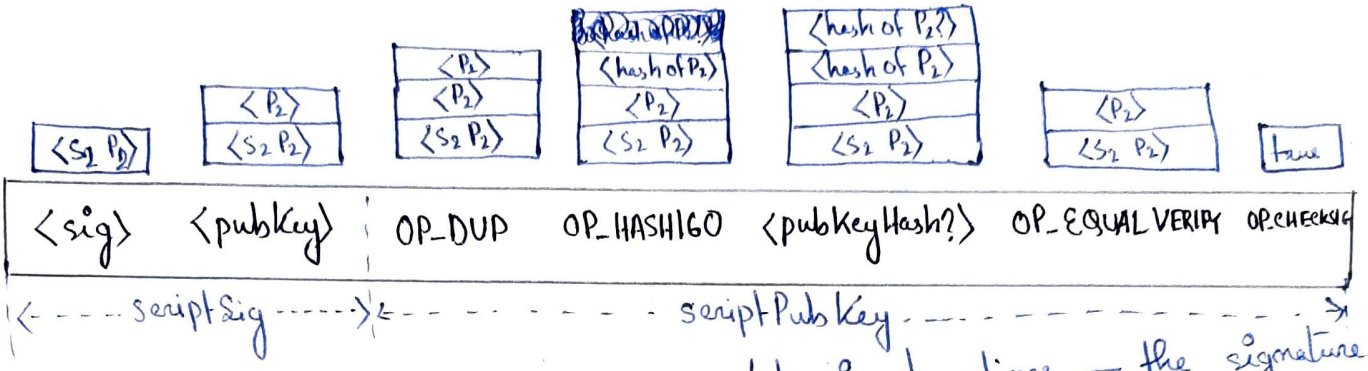
In the output the value 15.00 is sent to someone who has P_4 as its public key that means the value 15.00 is transferred to the person address which is hash of P_4. We saw that Y has value 15.12 so Y can send a value = 15.00. This transaction is hence valid. The scripting language does not check validity in this way. It uses stack based execution. To perform that first a transaction -based ledger is drawn for simplicity. Suppose S has public key as P_4 and X has private key as S_1 and public key as P_1 -

T_1	Input: $h_0[0]$ (Suppose X had this money already) Output: 10.12 → Y 5.15 → Z Signed(X)
T_2	Input: $h_1[1]$ Output: 5.00 → Y 0.15 → Z Signed(Z)
T_3	Input: $h_1[0]$, $h_2[0]$ Output: 15.00 → S 0.12 → Y Signed(Y)

A transaction-based ledger very close to Bitcoin.

Stack based, validity checking of T_3.

Execution of Bitcoin Script



Here, 1st two instructions are data instructions — the signature (here $\langle S_2 P_2 \rangle$) and the public key used to verify that signature which is specified in the scriptSig component of the input part of transaction T_3. As they are data instruction, they are pushed onto the stack.

Now in the scriptPubkey part that is the output script of T_2, 1st we have OP_DUP which pushes a copy of top element into the stack top. [scriptPubkey of T_1 can also be used to validate]

The next instruction is OP_HASH160 which pops the top value, compute its cryptographic hash and pushes the result onto top of the stack.

Then we are going to push one more data onto the stack. This data was specified by the sender of the transaction T_2. It is the hash value of a public key that was specified; the corresponding private key must be used to generate the signature to redeem these coins. At this point, there are two values at the top of the stack.

There is the hash of the public key, as specified by the sender and the hash of the public key

that was used by the recipient when trying to claim the coins. OP_EQUALVERIFY checks these two values at the top. If they are not equal, error occurs, & script stops. Else if both of them are equal, the instruction consumes these values. Now stack has a signature and the public key.

The public key is already checked so we have to check validity of the signature. OP_CHECKSIG pops both the signature and public key and does signature verification. It then returns TRUE, hence T_3 is a valid Transaction as in the same way we can check validity for 1st input as well as output.

4) Here, it is asked that if we use public key instead of hash of public key as address then what modifications to be made —

We know the bitcoin address is only a hash, so the sender can't provide a full public key in scriptPubkey component. When redeeming coins that have been sent to a Bitcoin address, the recipient provides both the signature and the public key. The script verifies that the provided public key does hash to the hash in scriptPubkey, and then it also checks the signature against the public key.

If the sender provides full public key instead of hash in scriptPubkey then there is chance of alteration of public key value. But if this is the situation then the receipient has to compare its public key with the public key present in scriptPubkey instead of hash value comparison.

So, output will be of the form

"out": [

{ "value": x

"scriptPubkey": " OP_DUP <P_K> OP_EQUALVERIFY OP_CHECKSIG"

}

]

Now, recipient will provide its public key to script and execution for validation will be as follows

Stack	Script	Description
Empty	<sig> <pubkey> OP_DUP <Pubkey> OP_EQUALVERIFY OP_CHECKSIG	scriptSig and ScriptPubkey are combined.
<sig> <pubkey>	OP_DUP<Pubkey> OP_EQUALVERIFY OP_CHECKSIG	Constants are added to stack
<sig> <pubkey> <pubkey>	<pubkey> OP_EQUALVERIFY OP_CHECKSIG	Top stack item is duplicated
<sig> <pubkey><pubkey> <pubkey>	OP_EQUALVERIFY OP_CHECKSIG	Constant added
<sig> <pubkey>	OP_CHECKSIG	Equality is checked between the top two stack items
true	Empty	Signature is checked for top two stack items.