__/__/____

#19 Blockchain 01 Oct 2022

Today Agenda :- 1. CPU Vs GPU Vs ASIC
2) Memepool
3) orphaned Blocks
4) 51% Attack
5) UTXO's
6) Wallet
7) public Key Vs Address

CPU Vs GPU Vs ASIC
(It is technology to used mining)

- CPU :- Central Processing Unit
  - generate smaller crypto like shiba
- GPU :-
  - Pr → Graphic Processing Unit
    mining speed better than CPU

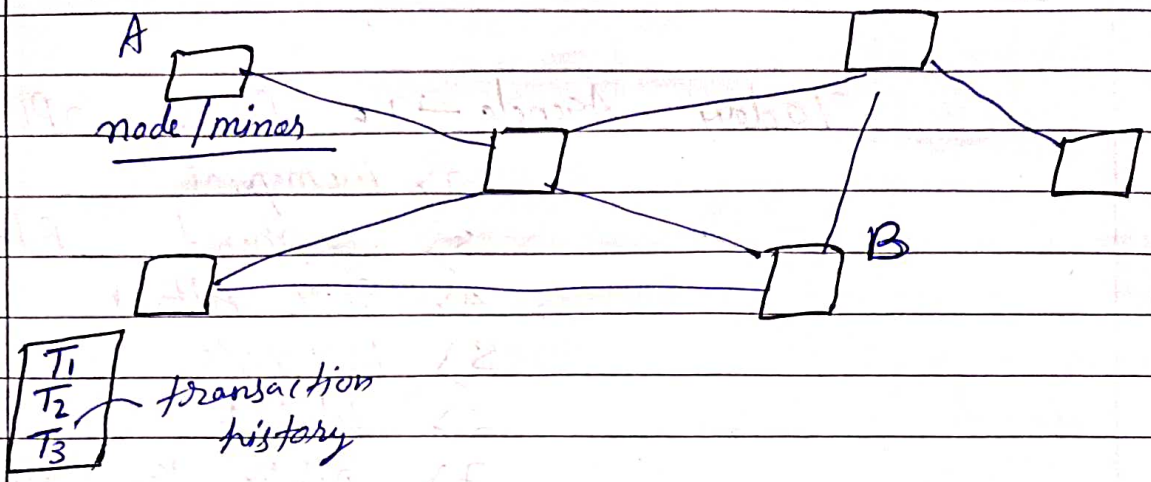
best ASIC → (Application Specific Integrated Circuit)

- It is only design only bitcoin mining.
- High performance, cost low.
-

(2)

It is memory we put transaction.

#Mempool (Every time you transaction data in mempool)



T_1 A send 0.5 bitcoin → B

T_2 C → A

* in IMB transaction 2700.

~~FIFO~~

- Every transaction will pay fee. different transaction free.
- Gas fee & transaction fee different.
- Pay fees sender.

Who pay fees?

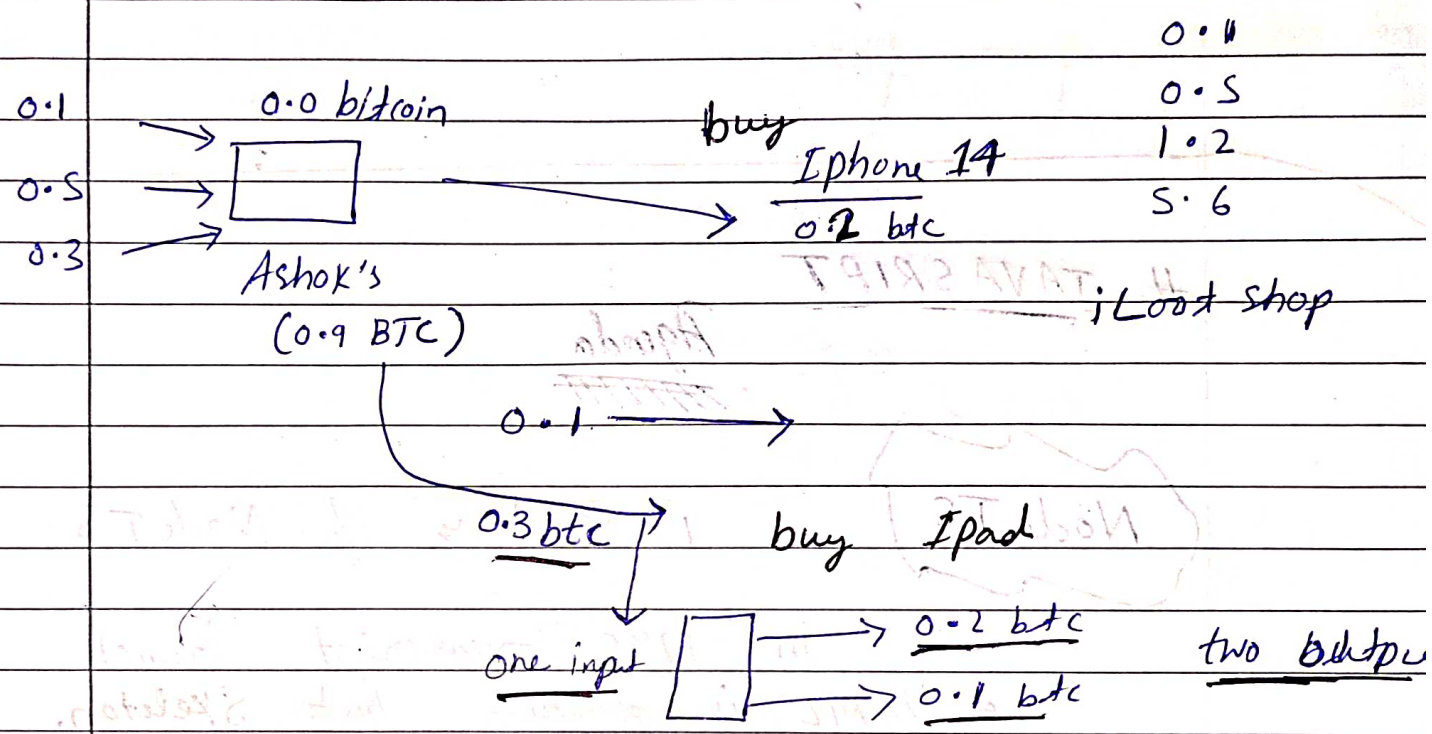
SEGWIT →

(3)

__/__/____

UTXO's – (Unspent Transaction Outputs)

The amount of digital currency remaining after crypto currency transaction is excuted.



Wallet → private key

Public key →

5 btc

Address no name → pseudo Address

- Every transaction create new pub key

④

Public key Address

Private key — Public key

Master Private key — Pr_1 — Pu_1

Pr_2 — Pu_2

Pr_3 — Pu_3

⇒ How to create bitcoin address.

Public key → hash → bitcoin Adress

JAVASRIPT

Agenda

NodeJS

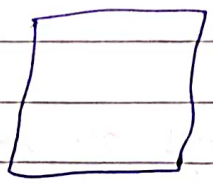
1) History of NodeJS

in 1995 javascript launch

- HTML is nothing but Skeleton.

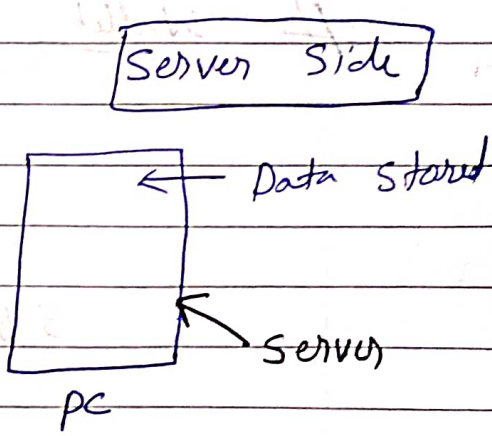
Javascript → DOM → Codding

Modulatiy



facebook.com
ineuron.com

user A



- Define domain then complete the show
- Client side

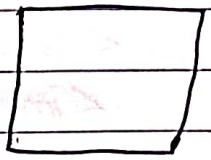
(5)

__/__/____

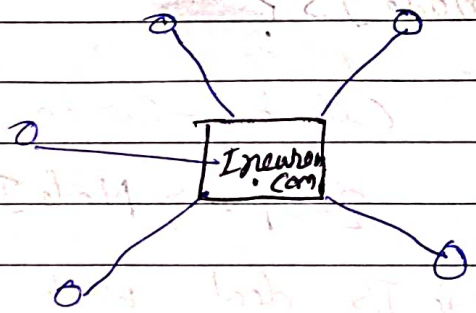
Modulatij

Modular programming allows many programmers to collaborate on same application. This code is stored across multiple files. Code is short, simple & easy understand.

high configuration



- It handle lot people.
- huge amount ram



- Proper management error.
- It need very fast

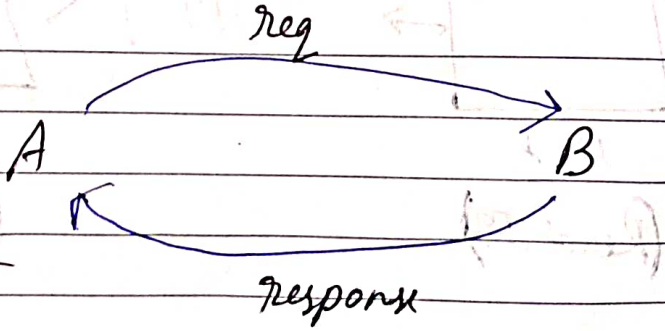
#1° Server

- High performance compute PC.

costly 1) On site server — costly

2) Cloud computing — cheap ✓

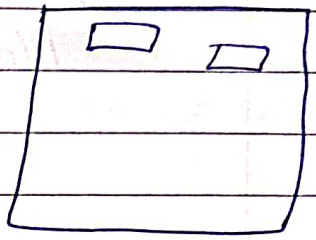
C



__/__/__

— **Browser Engine** : The browser engine works as bridge between the user interface and the rendering engine.

```
<html>
<P>
<Script>
<image>
```



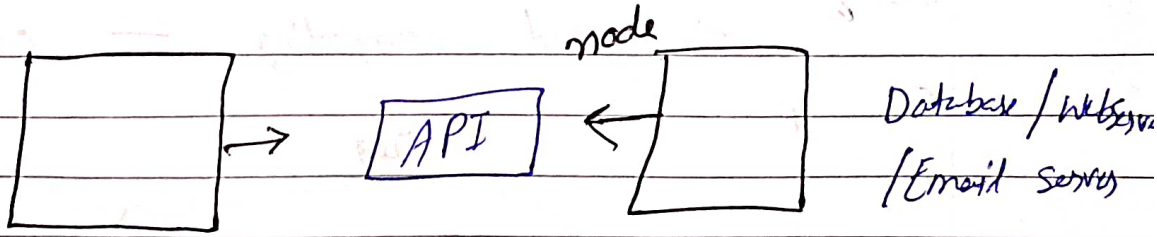
This is

2) Why nodeJS ?

Client - Server - DOM

2009 → NodeJS build

- nodeJS deal with server side functionality
- Frontend & backend
  - Frontend → What you see
  - backend → logic that you can see directly.



Client (ReactJS)

Server (NodeJS)

__/__/__

Node used

- node it is used for Data collection, Data Manupulation, data base access, fetching files from server.

example

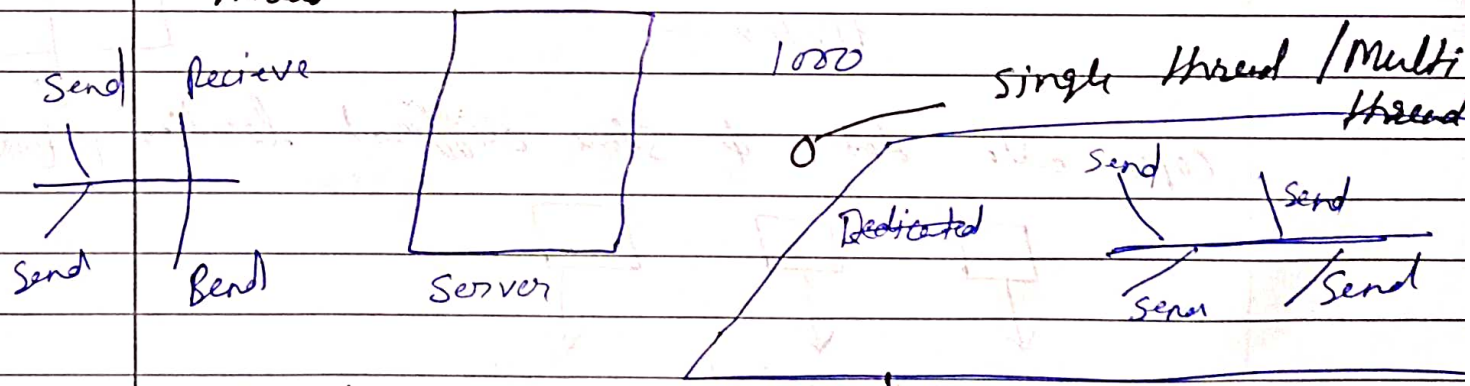
user1 personal address no

React used button, text form

3) properties of Node JS

- Single thread
- Event Driven
-

Thread



Single thread	Multi thread
1) Resource • Dedicated CPU • dedicated memory	• Multiple Resource

(8)

__/__/__

★ Web application 99.99% of the input/output operation

0.01% — CPU intensive

10 lakh → most of them

Cost → Multi thread.

Single thread

• larger cost → organization

• 3 lakh – 5 lakh

Node JS handle — 1–5 million users concurrent users.

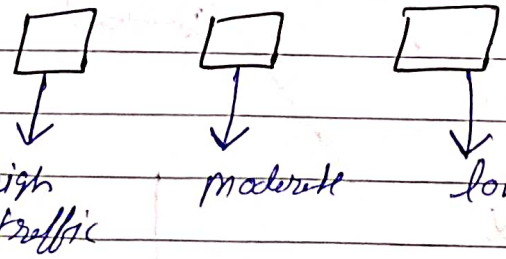
Traffic Part

How browsers work Search engine

Google — dedicated data world house across world

firefox —

copy all data & store different location petabytes



Web Server → Multiple process → own thread → dedicated resources.

require — import something

__/__/____

Node JS is non blockchain i/o. task.

- io task → time latency → time req to get response.

1 sec → 10^6 micro

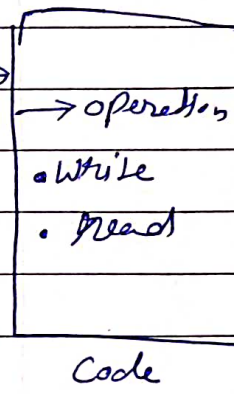
io task → which required 99-%.

programming

non blocking io

⇒ demo.txt

⇒ file.JS



File System

Whenever open a file

↓

asynchronously

- file system — fs module
- install node JS — fs module will be

```
const fs = require("fs");
```

Reading file in node JS

- fs Module
- reading
- Writing

fs.readWhile (name of the file, fallback data)

or "utf-8"

⇒ fs.readfile ("demo.txt", (err, data) ⇒ {

if (err) {

throw err.name;

}

else {

console.log (data.toString ());

or

(data)

}

});

binary hexadecimal ASCI UTF-8

What is utf-8

w binary pe format → plain english

process file

⇒ const fs = require ("fs");

fs.readfile ('demo.txt', utf-8, (err, data)

⇒ {

if (err) {

throw err.name;

}

console.log (data);

});

asynchronous nature ensure no blocking io

process.on ("uncaughtException", err ⇒ {

console.error ("there is some uncaught error that needs to be checked ${err}");

(11)

__/__/____

```
process.exit();
});
console.log("this part will be printed last");
```