BLOCKCHAIN - INTERNET OF THINGS (IoT)

Introduction to IoT and Blockchain

- The **Internet of Things (IoT)** refers to the network of interconnected physical devices (sensors, machines, etc.) that can collect and exchange data, enabling real-time monitoring and control of devices via the internet.

- Blockchain is a distributed ledger technology that records transactions in a secure, transparent, and immutable way.

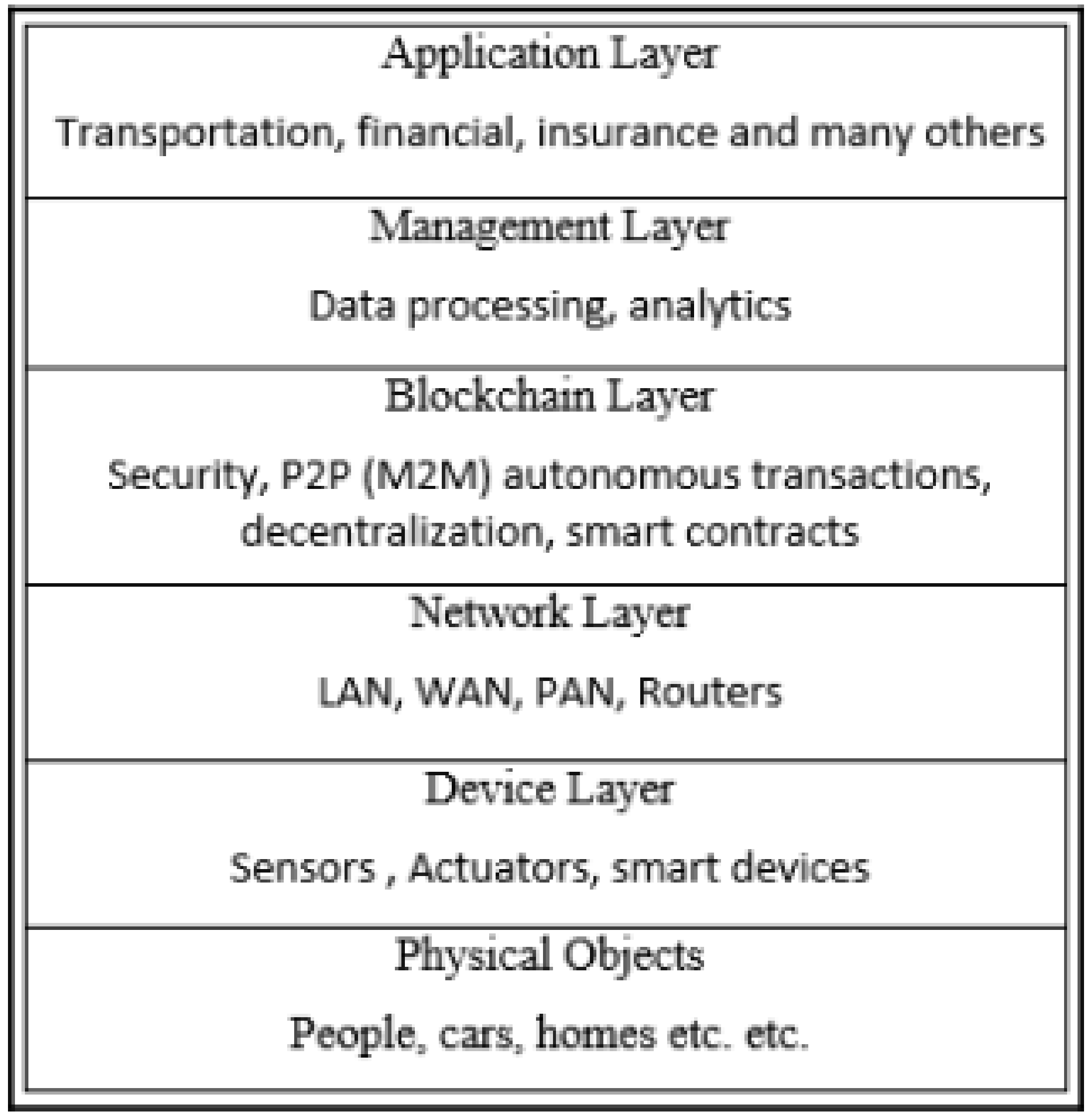
Four Main Functions:

- **Sensing** - Performed by sensors to monitor environments.
- **Reacting** - Carried out by actuators to control external environments.
- **Collecting** - Data collection via various sensors.
- **Communicating** - Done by chips for network connectivity.

  Components Involved: Sensors, actuators, chips, and network connectivity tools.

The Blockchain Based IoT Model:

- **Physical Objects Layer:** These physical objects generate data through sensors and smart devices.
- **Device Layer:** Contains IoT components like sensors, actuators, and smart devices that collect data from the physical environment.
- **Network Layer:** Responsible for transmitting data between IoT devices and the blockchain network. Ensures secure and reliable data transmission.
- **Blockchain Layer :** Integrates blockchain technology to enhance security, decentralization, and automation in IoT systems.
- **Management Layer:** Handles data processing and analytics for IoT devices. Uses AI, machine learning, and big data techniques to extract meaningful insights.
- **Application Layer:** Ensures that IoT-generated data and blockchain transactions support real-world applications.



Application Layer

Transportation, financial, insurance and many others

Management Layer

Data processing, analytics

Blockchain Layer

Security, P2P (M2M) autonomous transactions, decentralization, smart contracts

Network Layer

LAN, WAN, PAN, Routers

Device Layer

Sensors , Actuators, smart devices

Physical Objects

People, cars, homes etc. etc.

Blockchain-based IoT model

IoT Blockchain Experiment

Objective: Connect Raspberry Pi to Ethereum blockchain.

- **Hardware Used:** Raspberry Pi 3, LED, Resistors, Breadboard, Jumper wires.
- **Software Used:** Raspbian OS, Geth (Go Ethereum client), Web3.js.
- **Raspberry Pi :**A Raspberry Pi is a **low-cost**, credit-card-sized computer used for a wide range of projects. It is popular building **IoT applications** due to its **affordability** and **flexibility**. For this experiment, we use **Raspberry Pi 3 Model B.**
- **GPIO (General Purpose Input/Output)** pins allow for interaction with external devices like **sensors** and **actuators**, making it a powerful tool for experimentation.

1 Low-Cost Computer

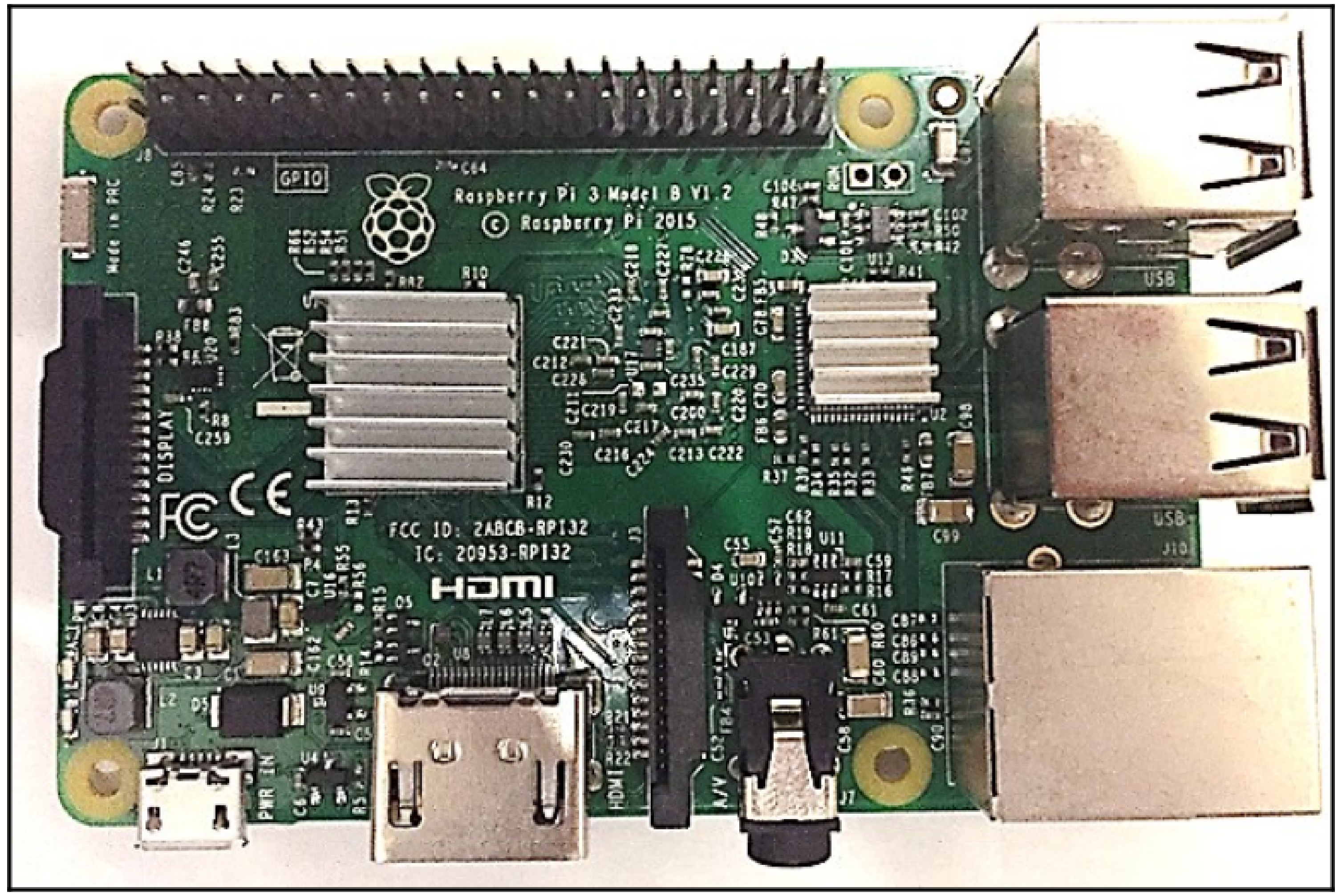
Affordable and versatile for various projects.

2 GPIO Pins

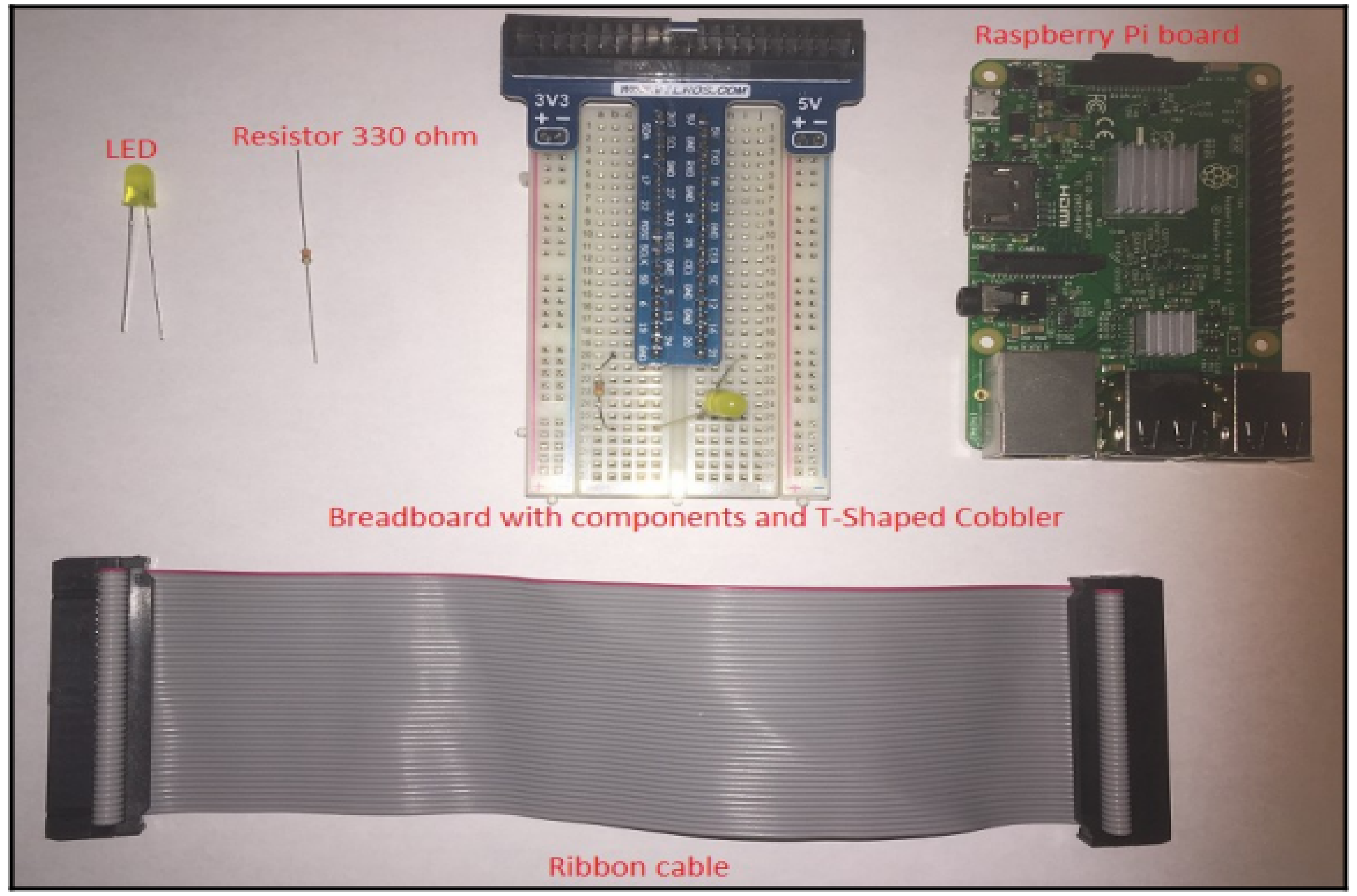
Enables interaction with external sensors and actuators.

3 Educational Tool

Popular for learning robotics, IoT, and blockchain.



Raspberry Pi Model B



Required components

Setting Up the Raspberry Pi:

- Install **NOOBS** (New Out of Box Software), a simple **operating system** installation manager for Raspberry Pi, providing a selection of OS options, including Raspbian.
- Install Raspbian OS using NOOBS: This can be downloaded and installed from the link : **https://www.raspberrypi.org/downloads/noobs/.**
- Confirm the architecture of the Raspberry Pi by running the command **uname -a** in the terminal. For this experiment, the architecture will typically be **ARMv7**. This helps **identify** the correct **version** of **Geth** to download.

1 Install NOOBS or Raspbian

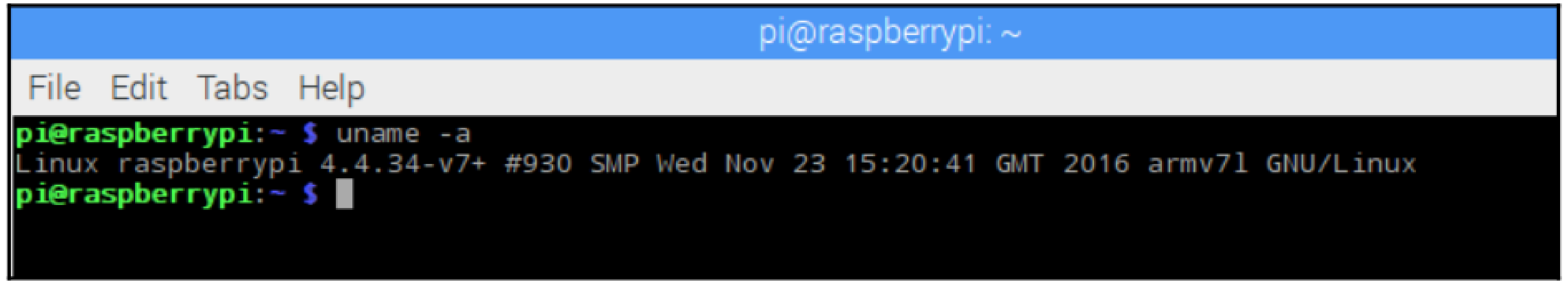
Choose an operating system for the Raspberry Pi.

2 Check Architecture

Confirm the architecture using **uname -a**.

3 Download Geth

Get the appropriate ARM binary for your Raspberry Pi.



```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 4.4.34-v7+ #930 SMP Wed Nov 23 15:20:41 GMT 2016 armv7l GNU/Linux
pi@raspberrypi:~ $ 
```

Raspberry Pi architecture

Installing Go Ethereum Client (Geth):

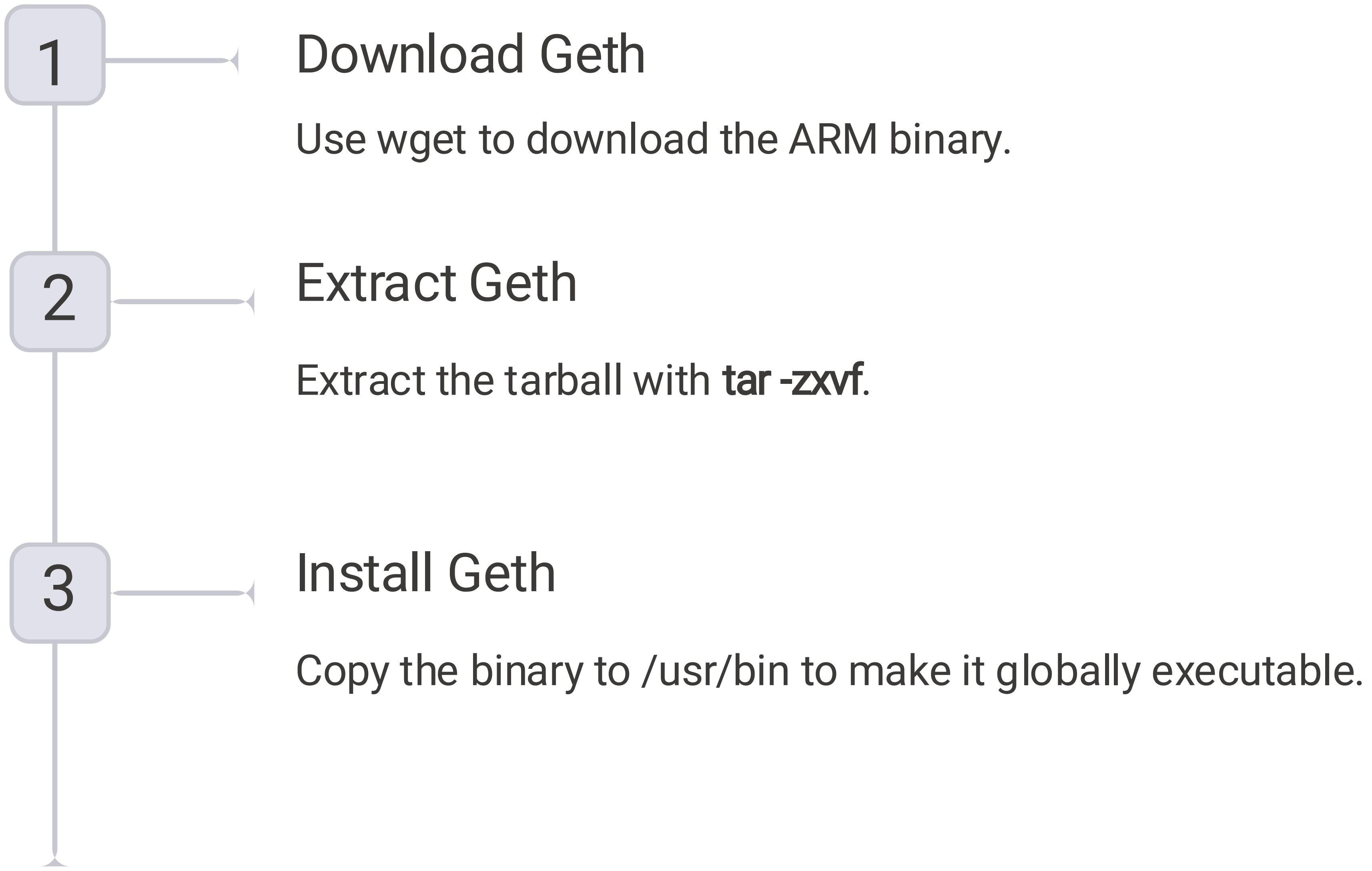
- Geth is a command-line client for **running Ethereum nodes**.

- Use **wget** to download the appropriate ARM binary for your Raspberry Pi. After downloading the Geth extract it with:

```
$ tar -zxvf geth-linux-arm7-1.5.6-2a609af5.tar.gz.
```

This creates a directory named : **geth-linux-arm7-1.5.6-2a609af5**.

- Move the Geth binary to /usr/bin so it can be used from anywhere: **sudo mv geth-linux-arm7-1.5.6-2a609af5/geth /usr/bin/geth.**



1 Download Geth

Use wget to download the ARM binary.

2 Extract Geth

Extract the tarball with **tar -zxvf**.

3 Install Geth

Copy the binary to /usr/bin to make it globally executable.

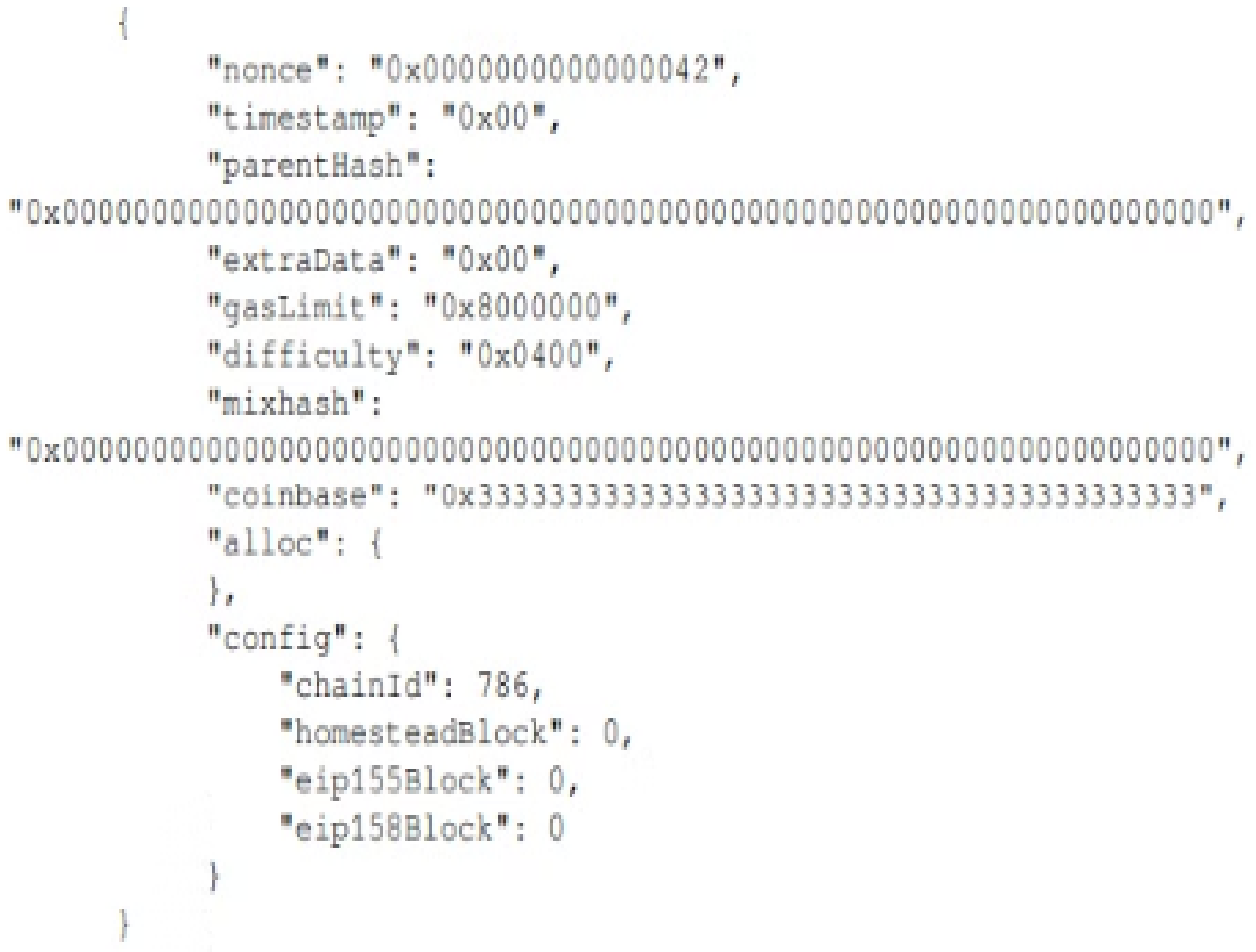
Genesis Block and Node Setup for Blockchain-IoT :

Genesis Block :

- The **Genesis Block** is the first block in a blockchain network.
- It serves as the foundation for all subsequent blocks.
- In an Ethereum-based private blockchain, a custom **genesis.json** file is required to initialize the network.

1. Creating a Genesis Block :

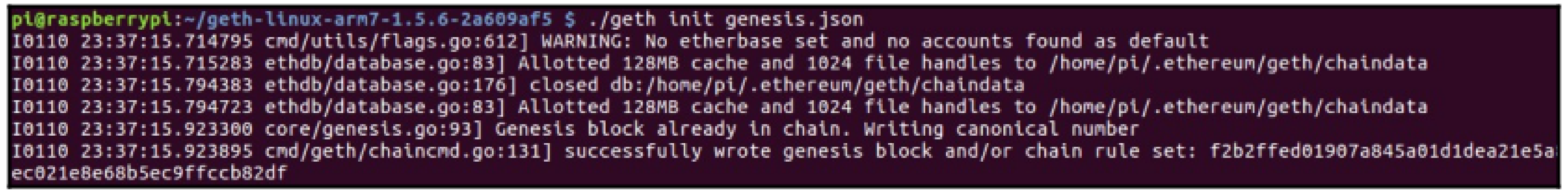
- A **genesis.json** file must be configured to define network-specific parameters such as difficulty, gas limit, and chain ID.



```
{
        "nonce": "0x0000000000000042",
        "timestamp": "0x00",
        "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
        "extraData": "0x00",
        "gasLimit": "0x8000000",
        "difficulty": "0x0400",
        "mixhash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
        "coinbase": "0x3333333333333333333333333333333333333333",
        "alloc": {
        },
        "config": {
            "chainId": 786,
            "homesteadBlock": 0,
            "eip155Block": 0,
            "eip158Block": 0
        }
}
```

Initializing the Genesis Block:

- Once the **genesis.json** file is created, it must be initialized on each node participating in the private Ethereum network.
- The following command used to initialize the genesis block on a Raspberry Pi or any other node is : **$ ./geth init genesis.json.**



```
pi@raspberrypi:~/geth-linux-arm7-1.5.6-2a609af5 $ ./geth init genesis.json
I0110 23:37:15.714795 cmd/utils/flags.go:612] WARNING: No etherbase set and no accounts found as default
I0110 23:37:15.715283 ethdb/database.go:83] Allotted 128MB cache and 1024 file handles to /home/pi/.ethereum/geth/chaindata
I0110 23:37:15.794383 ethdb/database.go:176] closed db:/home/pi/.ethereum/geth/chaindata
I0110 23:37:15.794723 ethdb/database.go:83] Allotted 128MB cache and 1024 file handles to /home/pi/.ethereum/geth/chaindata
I0110 23:37:15.923300 core/genesis.go:93] Genesis block already in chain. Writing canonical number
I0110 23:37:15.923895 cmd/geth/chaincmd.go:131] successfully wrote genesis block and/or chain rule set: f2b2ffed01907a845a01d1dea21e5a
ec021e8e68b5ec9ffccb82df
```

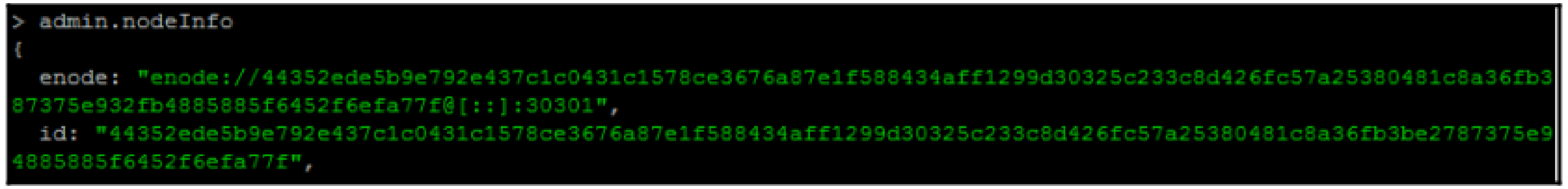
Initialize genesis file

2. Connecting Nodes in a Private Blockchain:

- After initializing the genesis block, nodes must connect to form a **private blockchain network**.

- **Adding Peers for Synchronization:** Nodes communicate through **static-nodes.json**, which stores the enode addresses of peer nodes.
- To retrieve a node's enode ID, run:

```
> admin.nodeInfo
```



```
> admin.nodeInfo
{
  enode: "enode://44352ede5b9e792e437c1c0431c1578ce3676a87e1f588434aff1299d30325c233c8d426fc57a25380481c8a36fb3
87375e932fb4885885f6452f6efa77f@[::]:30301",
  id: "44352ede5b9e792e437c1c0431c1578ce3676a87e1f588434aff1299d30325c233c8d426fc57a25380481c8a36fb3be2787375e9
4885885f6452f6efa77f",
```

- The **static-nodes.json** file should be updated with the enode information:

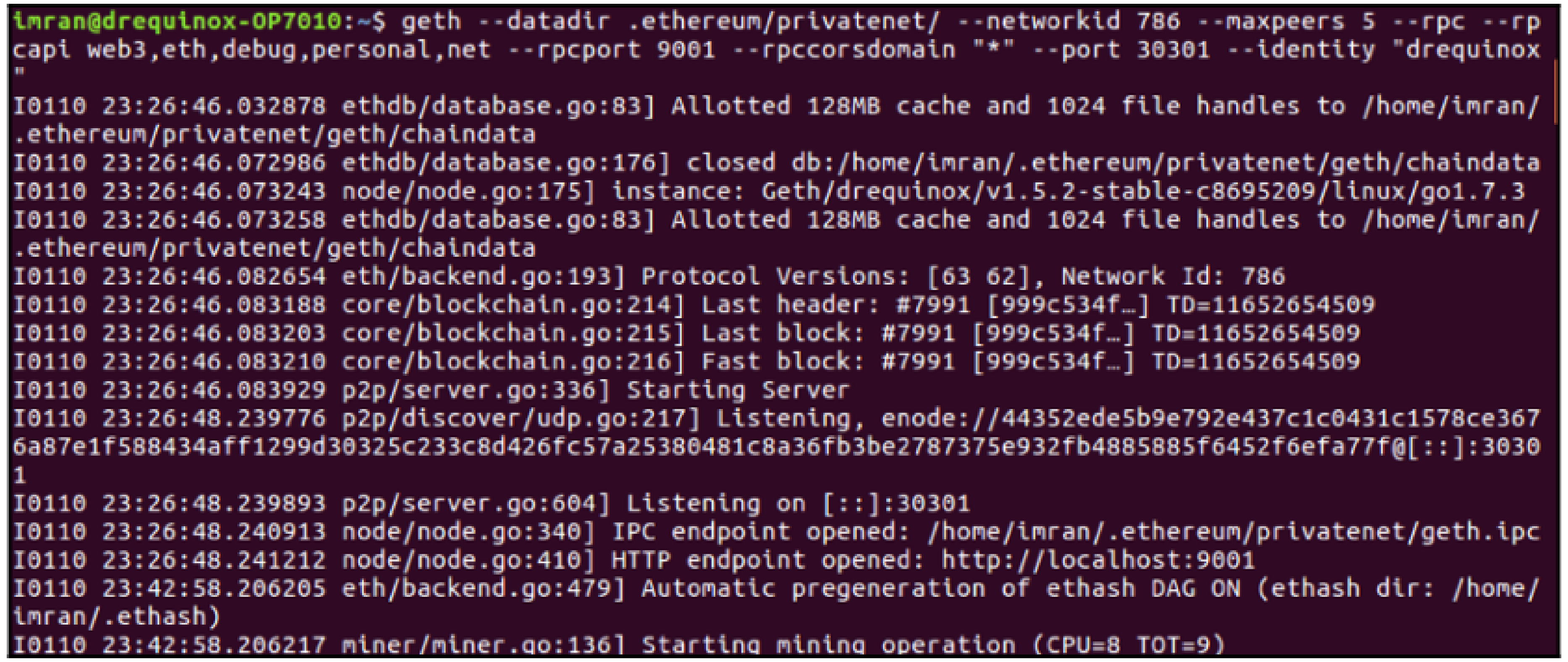
```json
[ "enode://<peer-node-enode-id>@<peer-node-ip>:30303" ]
```

3. First Node Setup :

The **first node** serves as the main blockchain participant. To start the first node, use:

```
$ geth --datadir .ethereum/privatenet/ --networkid 786 --maxpeers 5 --rpc \--rpcapi web3,eth,debug,personal,net –rpcport
 9001 --rpccorsdomain "*" \--port 30301 --identity "drequinox"
```

- **networkid:** Matches the network ID from **genesis.json**.
- **rpc & rpcapi:** Enables Remote Procedure Call (RPC) with necessary APIs.
- **identity:** Assigns a unique name to the node.
- Once the first node starts successfully, it should be **kept running** for other nodes to connect.



```
imran@drequinox-OP7010:~$ geth --datadir .ethereum/privatenet/ --networkid 786 --maxpeers 5 --rpc --rp
capi web3,eth,debug,personal,net --rpcport 9001 --rpccorsdomain "*" --port 30301 --identity "drequinox
"
I0110 23:26:46.032878 ethdb/database.go:83] Allotted 128MB cache and 1024 file handles to /home/imran/
.ethereum/privatenet/geth/chaindata
I0110 23:26:46.072986 ethdb/database.go:176] closed db:/home/imran/.ethereum/privatenet/geth/chaindata
I0110 23:26:46.073243 node/node.go:175] instance: Geth/drequinox/v1.5.2-stable-c8695209/linux/go1.7.3
I0110 23:26:46.073258 ethdb/database.go:83] Allotted 128MB cache and 1024 file handles to /home/imran/
.ethereum/privatenet/geth/chaindata
I0110 23:26:46.082654 eth/backend.go:193] Protocol Versions: [63 62], Network Id: 786
I0110 23:26:46.083188 core/blockchain.go:214] Last header: #7991 [999c534f…] TD=11652654509
I0110 23:26:46.083203 core/blockchain.go:215] Last block: #7991 [999c534f…] TD=11652654509
I0110 23:26:46.083210 core/blockchain.go:216] Fast block: #7991 [999c534f…] TD=11652654509
I0110 23:26:46.083929 p2p/server.go:336] Starting Server
I0110 23:26:48.239776 p2p/discover/udp.go:217] Listening, enode://44352ede5b9e792e437c1c0431c1578ce367
6a87e1f588434aff1299d30325c233c8d426fc57a25380481c8a36fb3be2787375e932fb4885885f6452f6efa77f@[::]:3030
1
I0110 23:26:48.239893 p2p/server.go:604] Listening on [::]:30301
I0110 23:26:48.240913 node/node.go:340] IPC endpoint opened: /home/imran/.ethereum/privatenet/geth.ipc
I0110 23:26:48.241212 node/node.go:410] HTTP endpoint opened: http://localhost:9001
I0110 23:42:58.206205 eth/backend.go:479] Automatic pregeneration of ethash DAG ON (ethash dir: /home/
imran/.ethash)
I0110 23:42:58.206217 miner/miner.go:136] Starting mining operation (CPU=8 TOT=9)
```

4. Raspberry Pi Node Setup :

To connect **Raspberry Pi to the blockchain network**, run:

```
$./geth --networkid 786 --maxpeers 5 --rpc --rpcapi \web3,eth,debug,personal,net --rpccorsdomain "*" --port 30302 -
-identity "raspberry"
```

When "**Block synchronization started**" appears in the output, the node has successfully connected to its peer.

Verifying Network Synchronization :

- Attach the **Geth console** to check connected peers:

$ geth attach > admin.peers

- To attach from the first node:

$ geth attach ipc:.ethereum/privatenet/geth.ipc

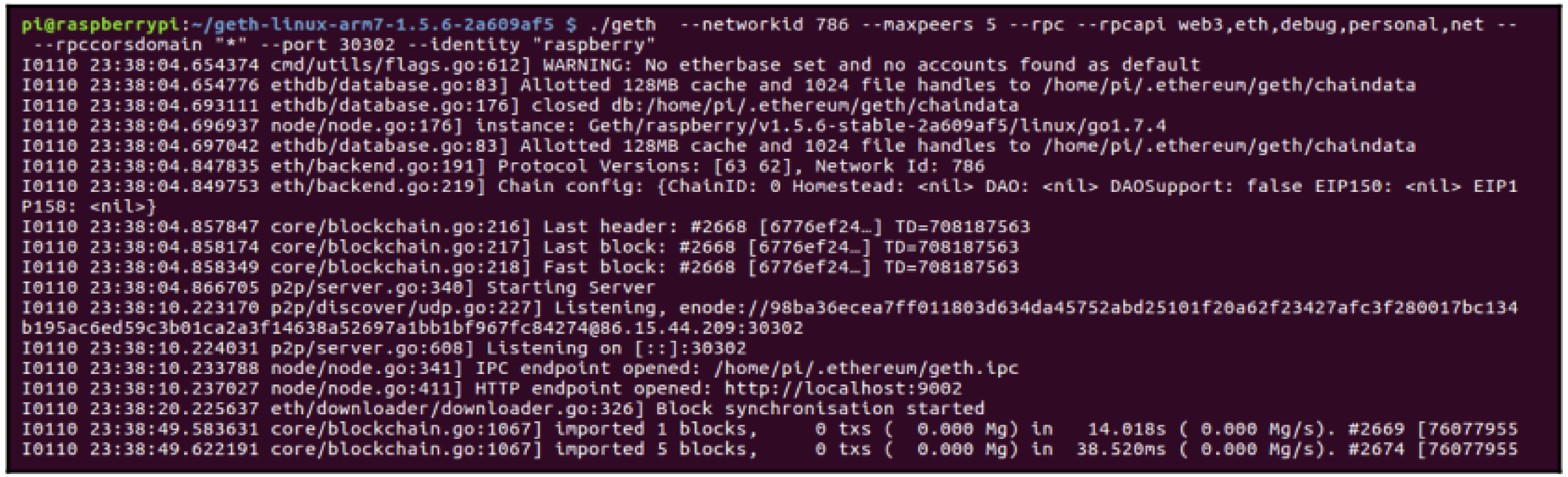
These steps confirm that **Raspberry Pi and the first node are properly connected**.

5. Installing Node.js and Dependencies:

For **smart contract execution and GPIO control**, **Node.js** and additional libraries are required.

Step 1: Install Node.js on Raspberry Pi:

$ curl -sL https://deb.nodesource.com/setup_7.x | sudo -E bash - $ sudo apt-get install nodejs



```
pi@raspberrypi:~/geth-linux-arm7-1.5.6-2a609af5 $ ./geth  --networkid 786 --maxpeers 5 --rpc --rpcapi web3,eth,debug,personal,net --
 --rpccorsdomain "*" --port 30302 --identity "raspberry"
I0110 23:38:04.654374 cmd/utils/flags.go:612] WARNING: No etherbase set and no accounts found as default
I0110 23:38:04.654776 ethdb/database.go:83] Allotted 128MB cache and 1024 file handles to /home/pi/.ethereum/geth/chaindata
I0110 23:38:04.693111 ethdb/database.go:176] closed db:/home/pi/.ethereum/geth/chaindata
I0110 23:38:04.696937 node/node.go:176] instance: Geth/raspberry/v1.5.6-stable-2a609af5/linux/go1.7.4
I0110 23:38:04.697042 ethdb/database.go:83] Allotted 128MB cache and 1024 file handles to /home/pi/.ethereum/geth/chaindata
I0110 23:38:04.847835 eth/backend.go:191] Protocol Versions: [63 62], Network Id: 786
I0110 23:38:04.849753 eth/backend.go:219] Chain config: {ChainID: 0 Homestead: <nil> DAO: <nil> DAOSupport: false EIP150: <nil> EIP1
P158: <nil>}
I0110 23:38:04.857847 core/blockchain.go:216] Last header: #2668 [6776ef24…] TD=708187563
I0110 23:38:04.858174 core/blockchain.go:217] Last block: #2668 [6776ef24…] TD=708187563
I0110 23:38:04.858349 core/blockchain.go:218] Fast block: #2668 [6776ef24…] TD=708187563
I0110 23:38:04.866705 p2p/server.go:340] Starting Server
I0110 23:38:10.223170 p2p/discover/udp.go:227] Listening, enode://98ba36ecea7ff011803d634da45752abd25101f20a62f23427afc3f280017bc134
b195ac6ed59c3b01ca2a3f14638a52697a1bb1bf967fc84274@86.15.44.209:30302
I0110 23:38:10.224031 p2p/server.go:608] Listening on [::]:30302
I0110 23:38:10.233788 node/node.go:341] IPC endpoint opened: /home/pi/.ethereum/geth.ipc
I0110 23:38:10.237027 node/node.go:411] HTTP endpoint opened: http://localhost:9002
I0110 23:38:20.225637 eth/downloader/downloader.go:326] Block synchronisation started
I0110 23:38:49.583631 core/blockchain.go:1067] imported 1 blocks,     0 txs (  0.000 Mg) in   14.018s ( 0.000 Mg/s). #2669 [76077955
I0110 23:38:49.622191 core/blockchain.go:1067] imported 5 blocks,     0 txs (  0.000 Mg) in  38.520ms ( 0.000 Mg/s). #2674 [76077955
```

geth on the Raspberry Pi.

Verify the installation:

```
$ node -v $ npm -v
```

Recommended versions: **Node.js v7.4.0** and **npm 4.0.5**.

Step 2: Install Web3.js for Blockchain Interaction

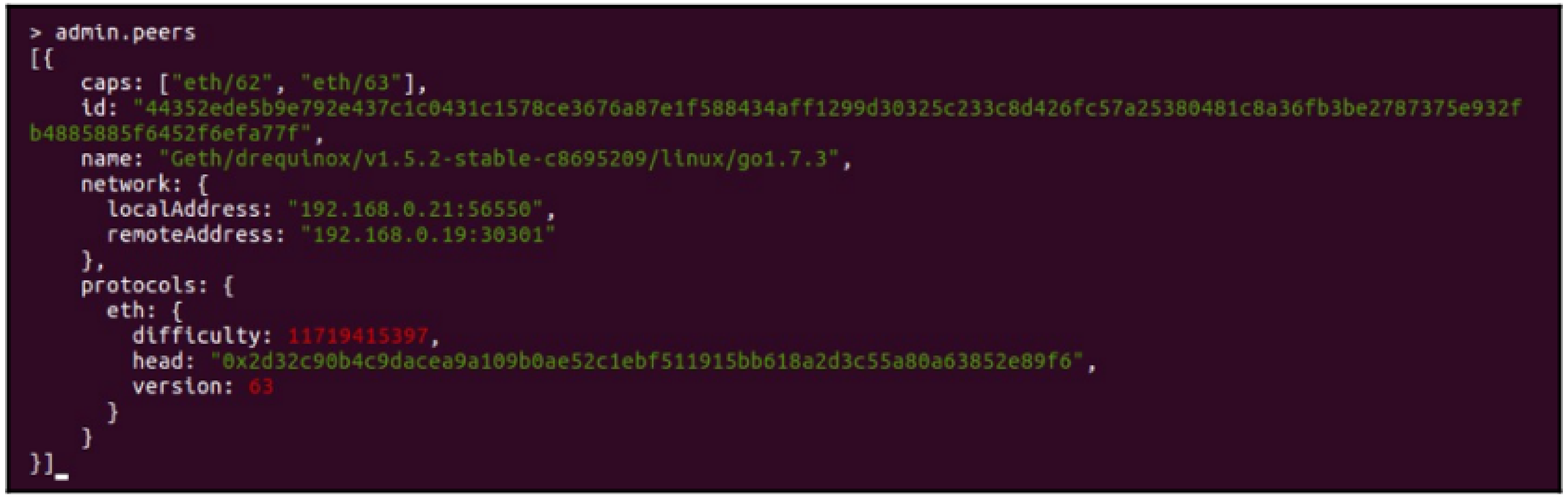
```
$ npm install web3@0.20.2
```

- Ensures compatibility with the Ethereum network.

Step 3: Install Onoff for GPIO Control

```
$ npm install onoff
```

- Enables Raspberry Pi to interact with connected IoT devices.

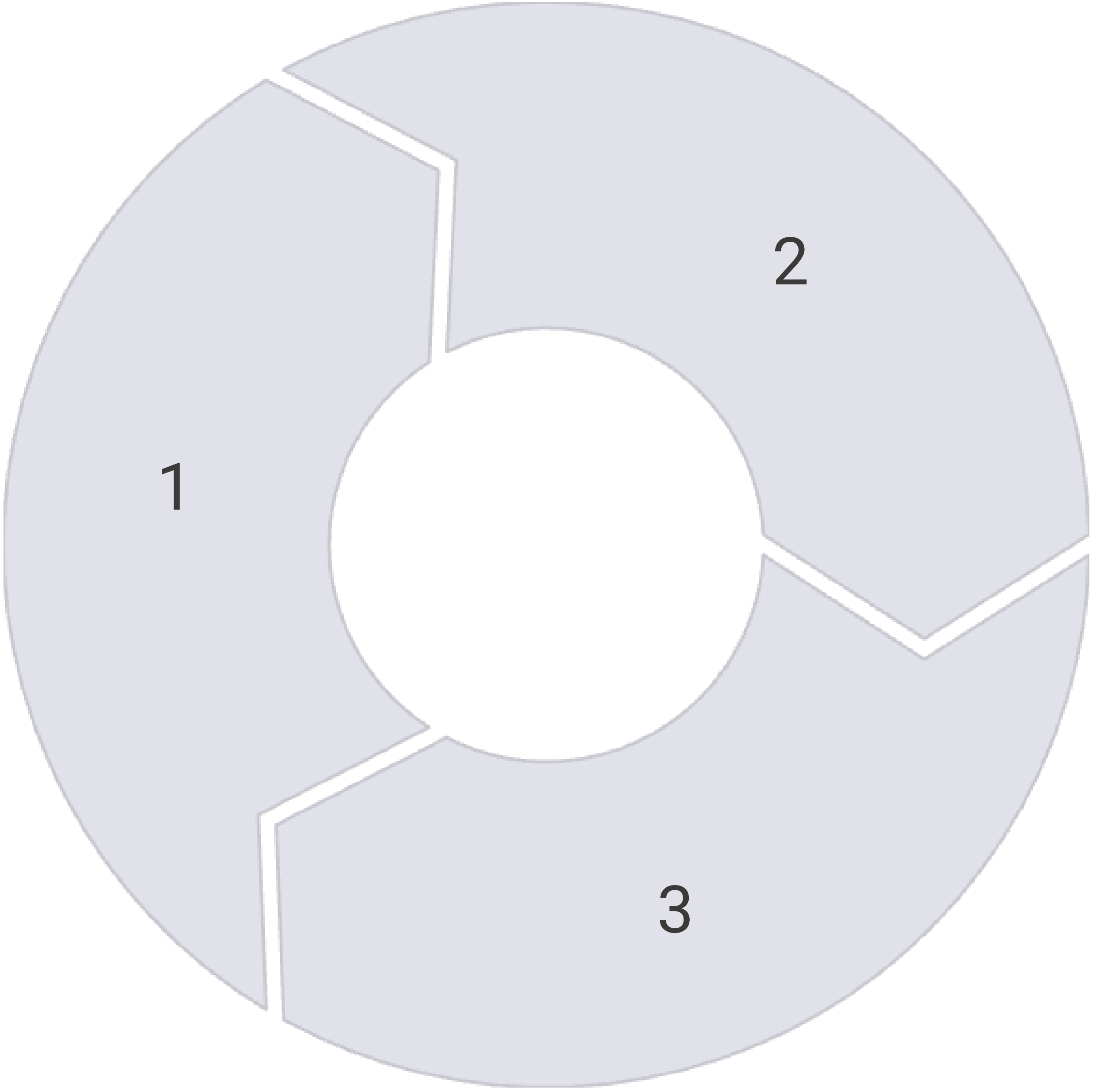


```
> admin.peers
[{
    caps: ["eth/62", "eth/63"],
    id: "44352ede5b9e792e437c1c0431c1578ce3676a87e1f588434aff1299d30325c233c8d426fc57a25380481c8a36fb3be2787375e932f
b4885885f6452f6efa77f",
    name: "Geth/drequinox/v1.5.2-stable-c8695209/linux/go1.7.3",
    network: {
      localAddress: "192.168.0.21:56550",
      remoteAddress: "192.168.0.19:30301"
    },
    protocols: {
      eth: {
        difficulty: 11719415397,
        head: "0x2d32c90b4c9dacea9a109b0ae52c1ebf511915bb618a2d3c55a80a63852e89f6",
        version: 63
      }
    }
}]_
```

geth console admin peers command running on Raspberry Pi

Final Output – LED Control

- The LED will turn on when the smart contract event is triggered, providing a visual indication of the IoT device being controlled by the blockchain event.
- The console will display **LED On** when the event is successfully triggered.



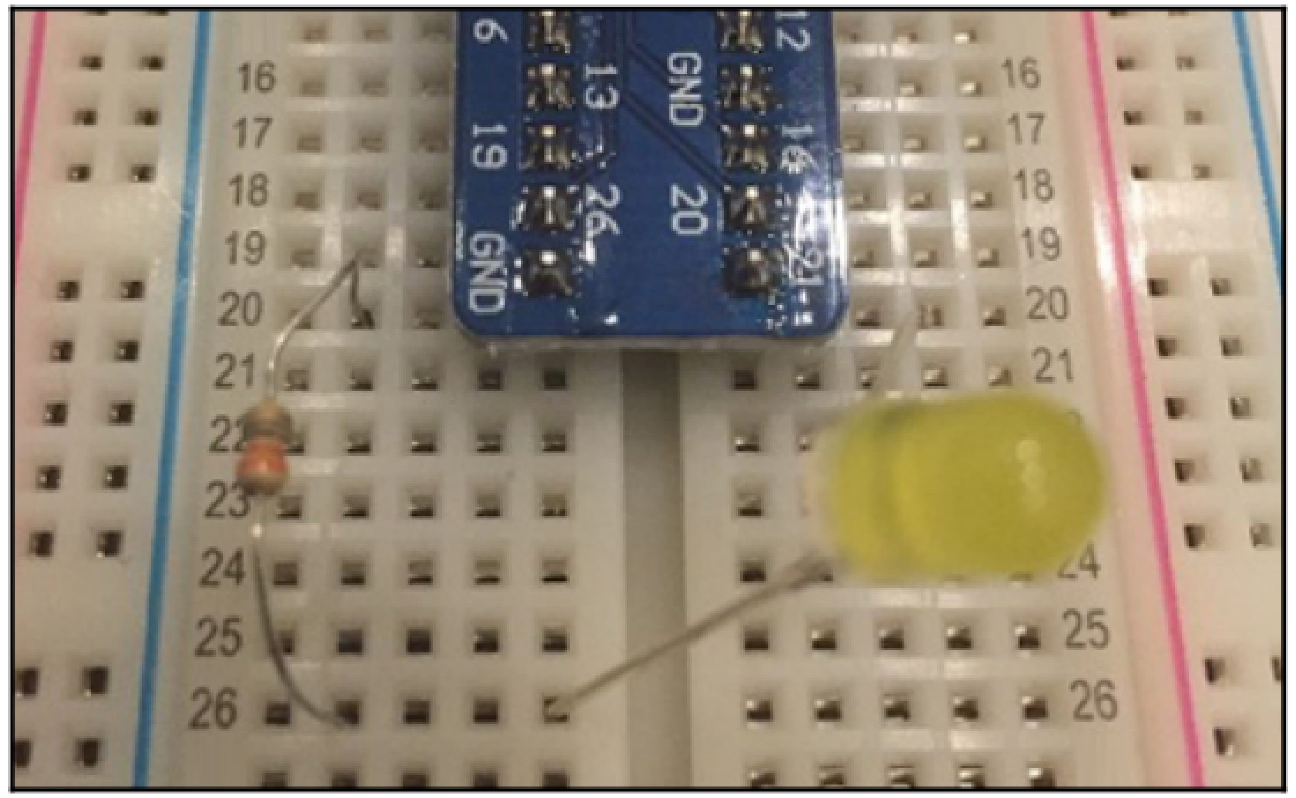
Smart Contract Event

LED Turns On

Console Output: LED On

Circuit Setup

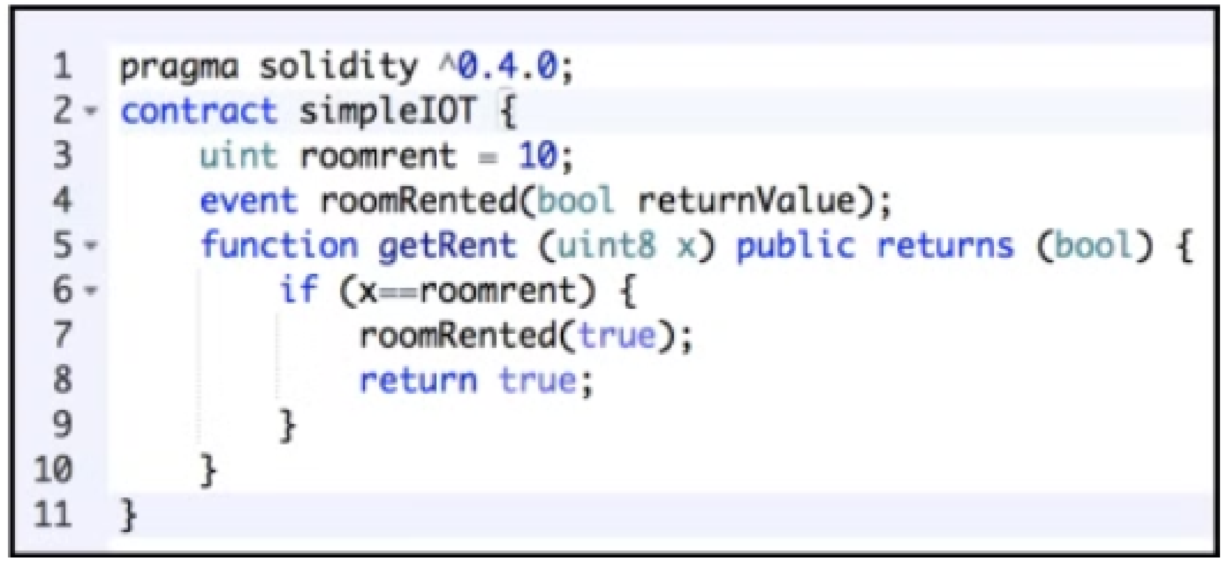
- The circuit setup involves connecting an **LED to the Raspberry Pi GPIO pins.**
- The positive leg (long leg) of the LED is connected to GPIO pin 21, while the negative leg (short leg) is connected to a resistor, which is then grounded (GND).
- A ribbon cable is used to interface with the GPIO connector on the Raspberry Pi, ensuring a stable connection for the IoT application.



Connections for components on the breadboard

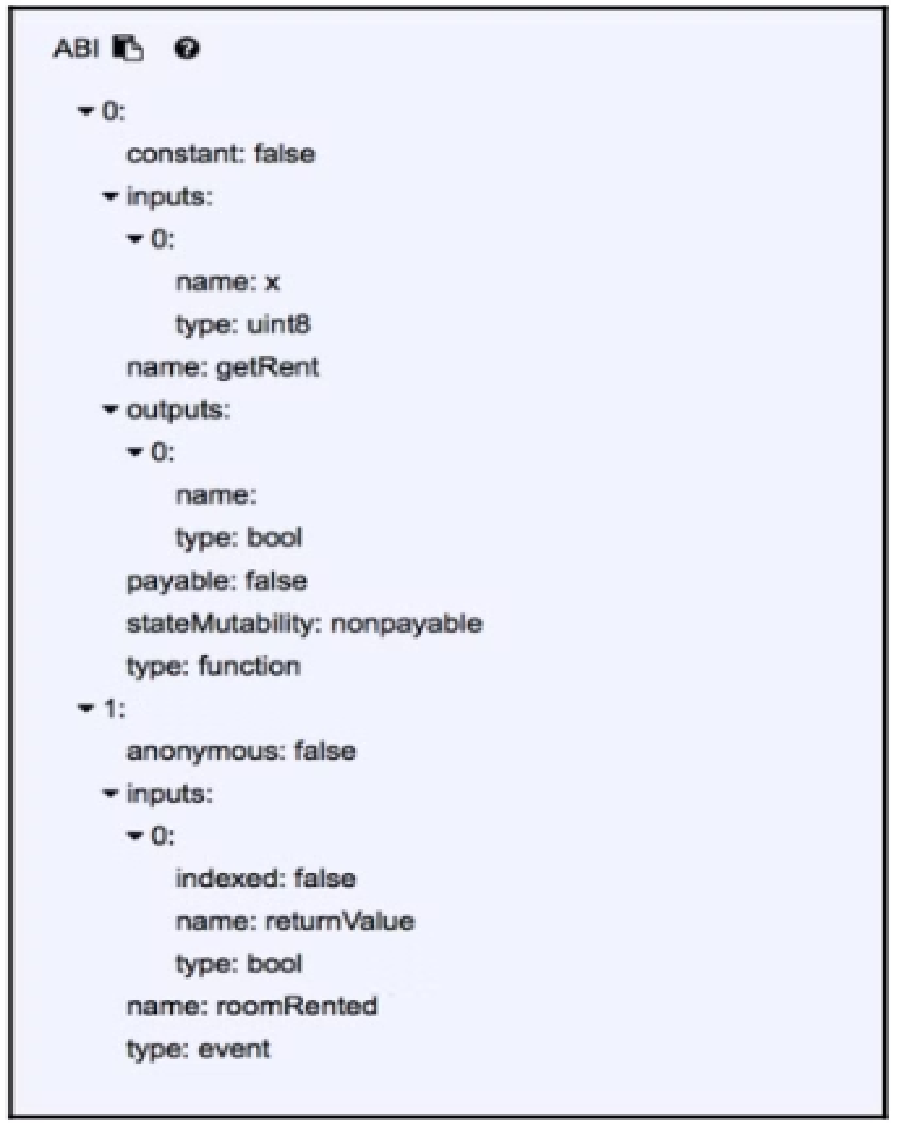
Smart Contract Development :

- A **Solidity-based smart contract** is developed to **control the LED**. The contract contains a function that takes an input value and **triggers** an event if the expected value matches the input.
- The smart contract source code is shown as follows :



```
pragma solidity ^0.4.0;
contract simpleIOT {
    uint roomrent = 10;
    event roomRented(bool returnValue);
    function getRent (uint8 x) public returns (bool) {
        if (x==roomrent) {
            roomRented(true);
            return true;
        }
    }
}
```

Application Binary Interface (ABI): ABI generated by the Remix IDE enables interaction with the deployed smart contract.



```
ABI
▾ 0:
    constant: false
  ▾ inputs:
    ▾ 0:
        name: x
        type: uint8
    name: getRent
  ▾ outputs:
    ▾ 0:
        name:
        type: bool
    payable: false
    stateMutability: nonpayable
    type: function
▾ 1:
    anonymous: false
  ▾ inputs:
    ▾ 0:
        indexed: false
        name: returnValue
        type: bool
    name: roomRented
    type: event
```

ABI from Remix IDE

Connecting Raspberry Pi to a Private Blockchain :

Two methods exist for the Raspberry Pi to interact with the private blockchain using Web3:

1. **Running a Local Geth Client** – The Raspberry Pi runs a local Geth client to maintain its ledger.

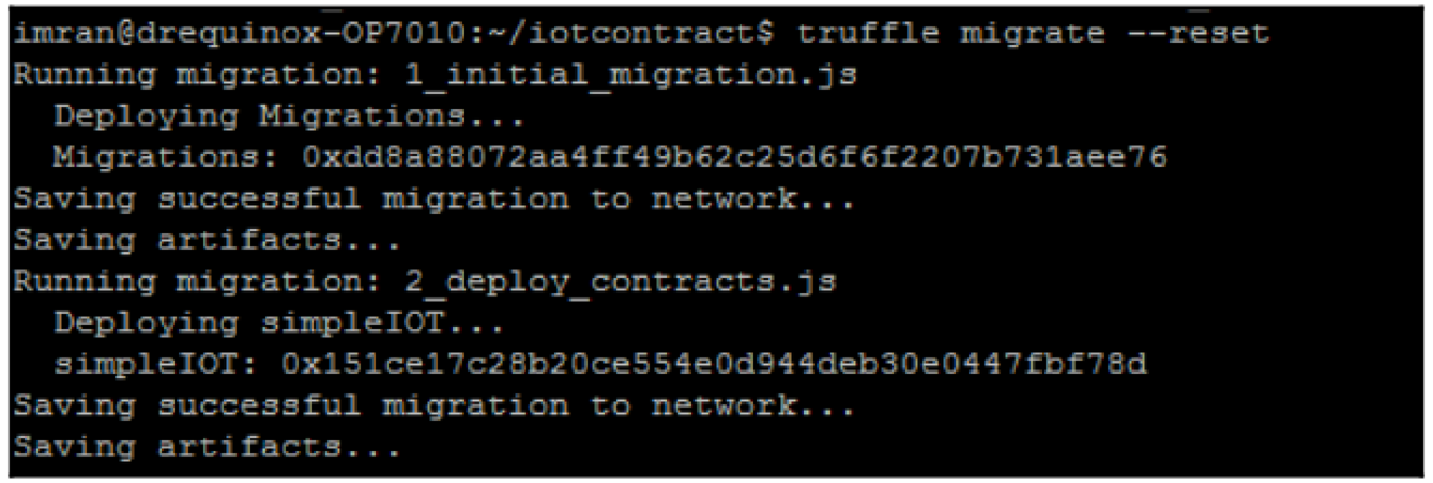
2. **Connecting to an External Node** – Due to resource constraints, the Pi connects to an external blockchain node via a Web3 provider over RPC.

Deploying the Smart Contract : The contract is deployed on a private Ethereum network using Truffle:

```
$ truffle migrate
```

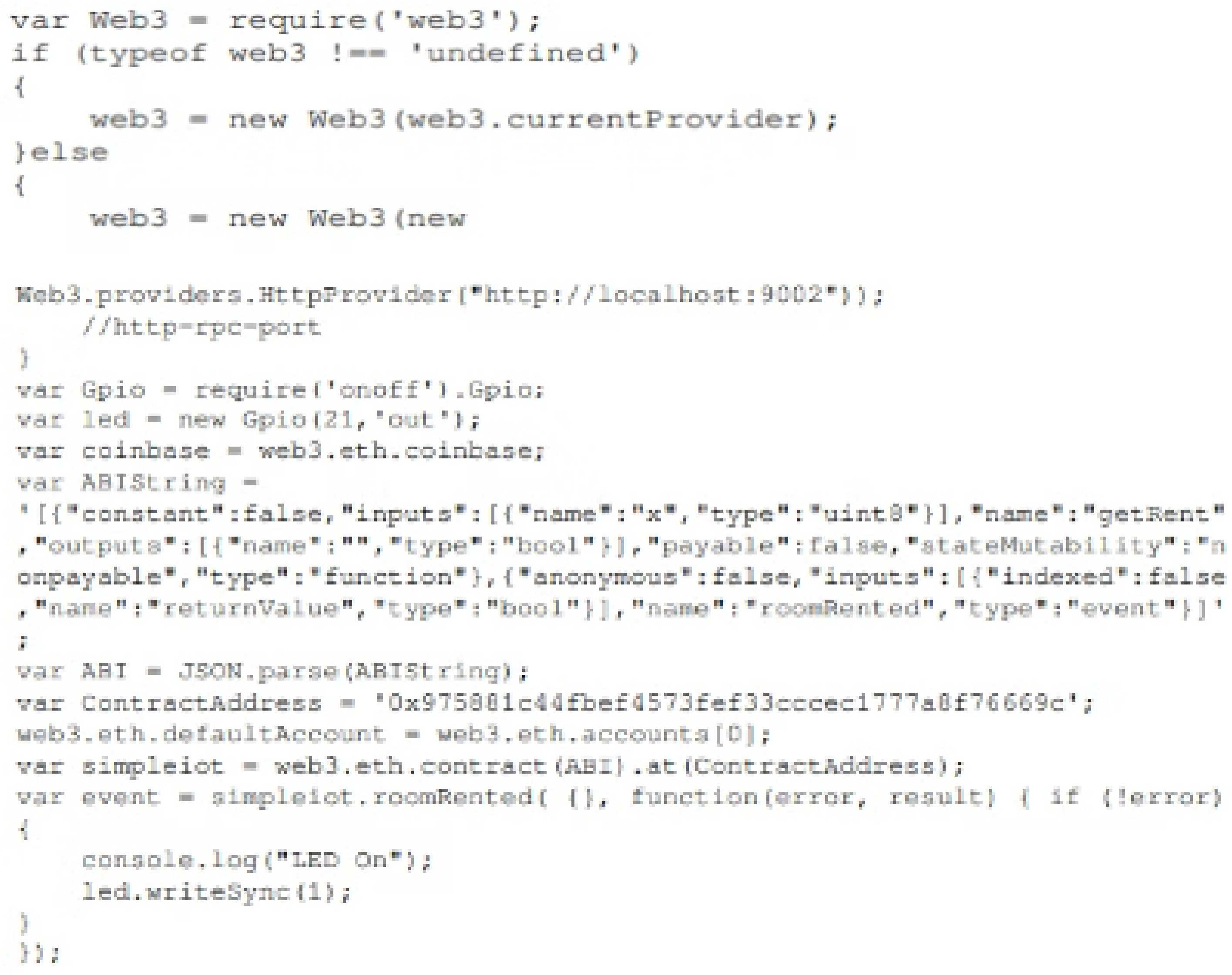
Once deployed, the contract's address must be updated in the JavaScript client.

JavaScript Client to Control IoT Device : A JavaScript program **listens for** **smart contract events** and triggers the LED using the Raspberry Pi GPIO library. The code is as follows:



```
imran@drequinox-OP7010:~/iotcontract$ truffle migrate --reset
Running migration: 1_initial_migration.js
  Deploying Migrations...
  Migrations: 0xdd8a88072aa4ff49b62c25d6f6f2207b731aee76
Saving successful migration to network...
Saving artifacts...
Running migration: 2_deploy_contracts.js
  Deploying simpleIOT...
  simpleIOT: 0x151ce17c28b20ce554e0d944deb30e0447fbf78d
Saving successful migration to network...
Saving artifacts...
```

Truffle deploy



```
var Web3 = require('web3');
if (typeof web3 !== 'undefined')
{
    web3 = new Web3(web3.currentProvider);
}else
{
    web3 = new Web3(new

Web3.providers.HttpProvider("http://localhost:9002"));
    //http-rpc-port
}
var Gpio = require('onoff').Gpio;
var led = new Gpio(21,'out');
var coinbase = web3.eth.coinbase;
var ABIString =
'[{"constant":false,"inputs":[{"name":"x","type":"uint8"}],"name":"getRent"
,"outputs":[{"name":"","type":"bool"}],"payable":false,"stateMutability":"n
onpayable","type":"function"},{"anonymous":false,"inputs":[{"indexed":false
,"name":"returnValue","type":"bool"}],"name":"roomRented","type":"event"}]'
;
var ABI = JSON.parse(ABIString);
var ContractAddress = '0x975881c44fbef4573fef33cccec1777a8f76669c';
web3.eth.defaultAccount = web3.eth.accounts[0];
var simpleiot = web3.eth.contract(ABI).at(ContractAddress);
var event = simpleiot.roomRented( {}, function(error, result) { if (!error)
{
    console.log("LED On");
    led.writeSync(1);
}
});
```

Running the Application :

The JavaScript client is executed using Node.js:

```
$ node index.js
```

Once running, the smart contract can be triggered via the Truffle console:

```
truffle(development)> getRent(10)
```

- If the transaction is successful, the event is emitted, and the Raspberry Pi turns on the LED.

Thank You!