LINUXCON containercon CLOUDOPEN

CHINA 2017

THINK OPEN

Flowchain: A Case Study on Building a **Blockchain** for the **IoT**

Jollen, *Devify Inc.*

jollen@flowchain.io
https://flowchain.io

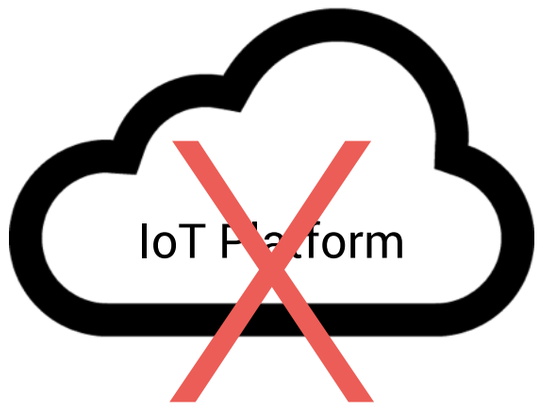
LF ASIA, LLC

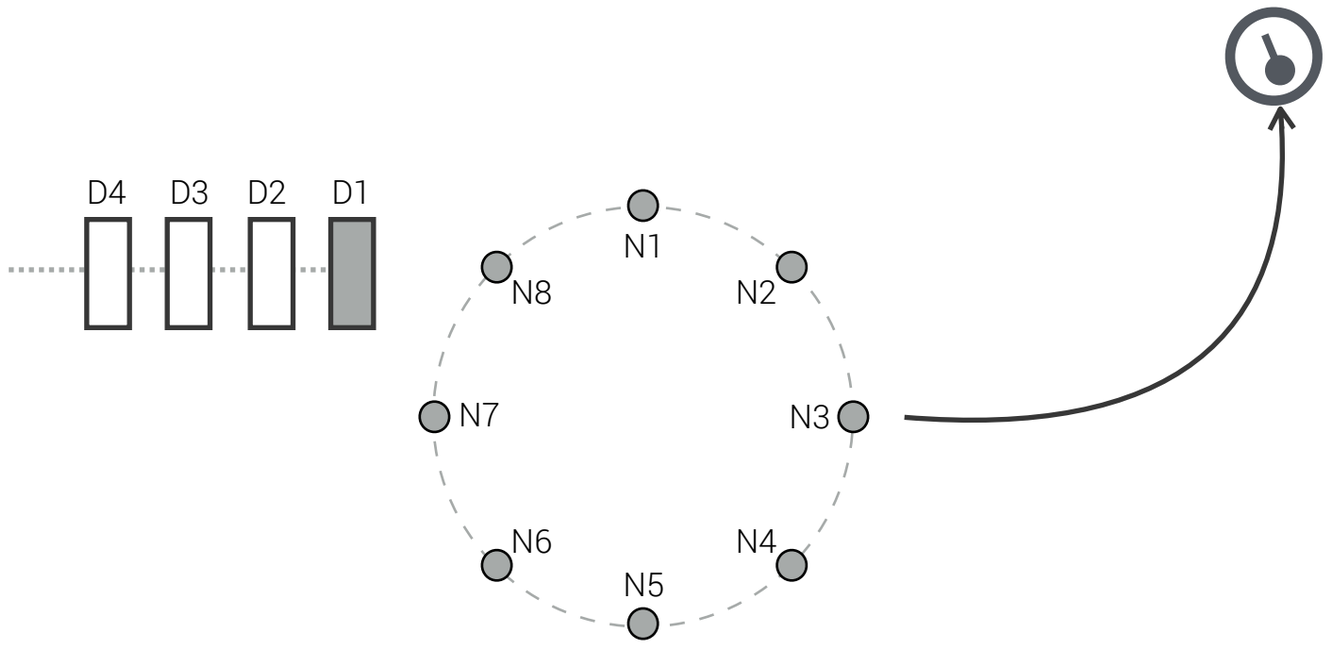
Agenda

- Motivation
- Architecture Overview
- Handling Heterogeneous Hardware
- Device Interoperability
- Distributed Ledger for the IoT
- Consensus System
- Use the Flowchain SDK

Motivation and Purpose

- Attempt to use the decentralized IoT model to ensure data privacy and security





Flowchain

Goals and Assumption

- **Decentralized IoT**
  - Centralized IoT model uses an IoT platform that acts as “hub” to control the data exchange between devices.
  - The IoT needs a decentralized model for exchanging data without any centralized party.
- **The blockchain technology is (maybe) a solution**
  - How does the blockchain technology help facilitate such challenges ?
  - Recently, the blockchain for the IoT has considered an

Agenda

- Motivation
- Architecture Overview
- Handling Heterogeneous Hardware
- Device Interoperability
- Distributed Ledger for the IoT
- Consensus System
- Use the Flowchain SDK

Blockchain IoT Purposes

Major IoT platforms are based on centralized model. The platform acts as broker or hub to control the interactions between devices.

Devices need to exchange data between themselves autonomously. This leads to decentralized IoT platforms.

Preliminary

- A blockchain is only one type of data structure considered to be a distributed ledger
- A distributed ledger technology (DLT) provides a new data structure varies from the blockchain for various purposes

Blockchain Data Structure

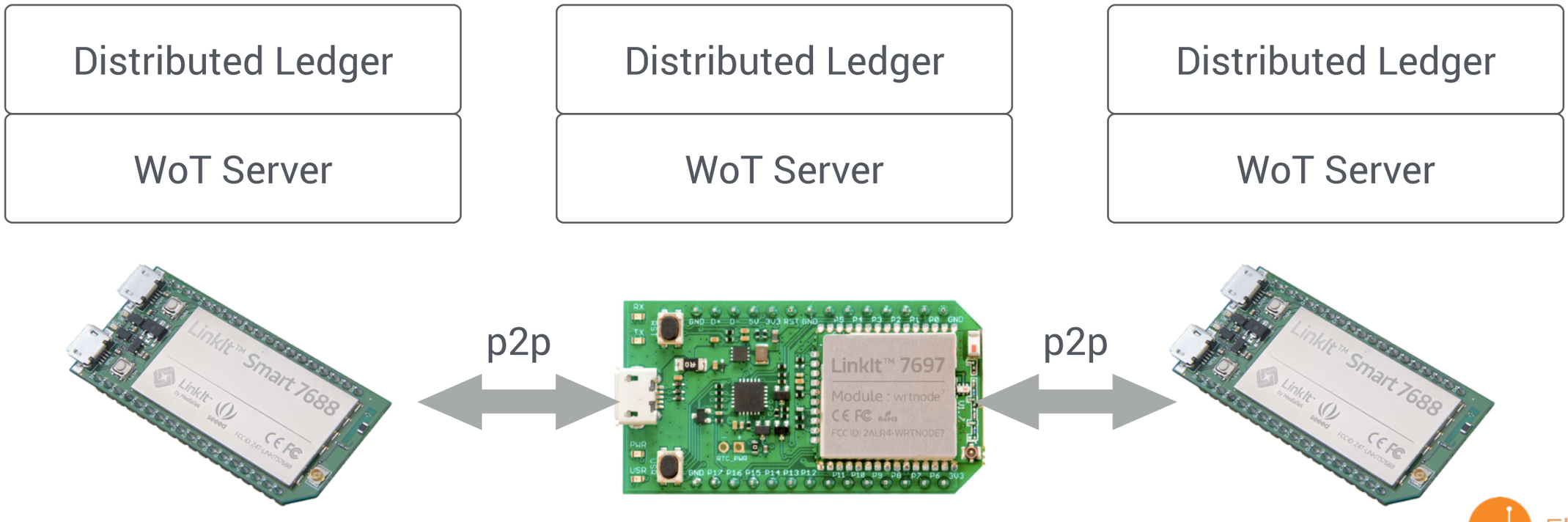
- Ordered and back-linked list
- Stored in a flat file or database
- Each block is identified by a hash
- Double SHA-256 hash function

DLT Essentials

- A peer-to-peer network
- Consensus algorithms

Background

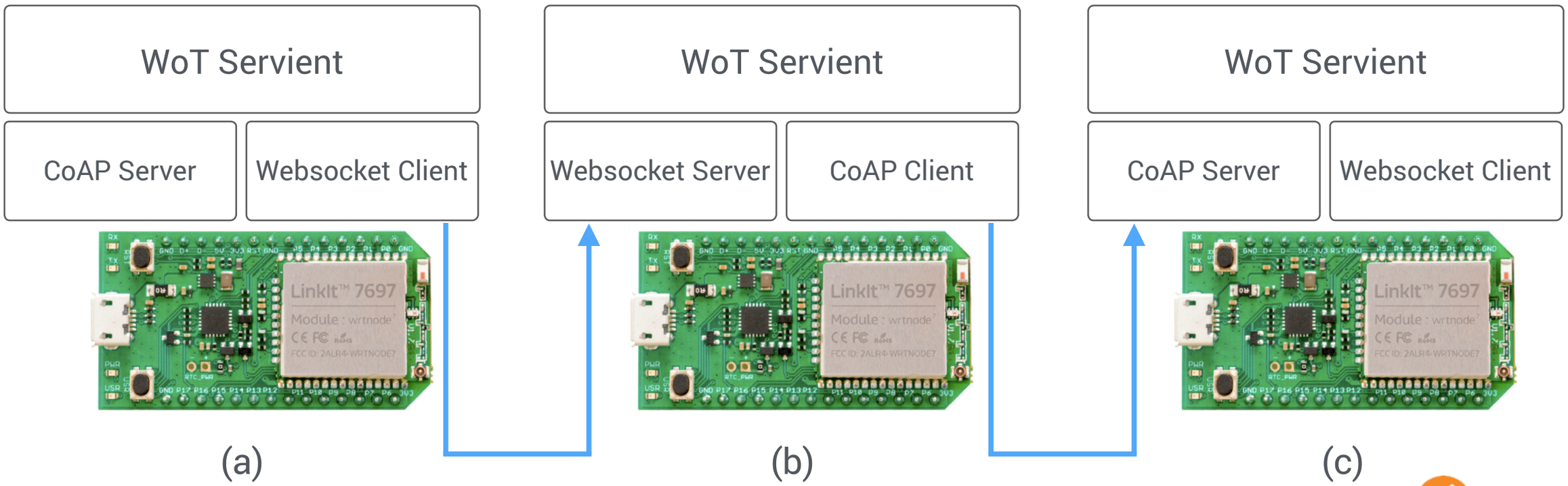
- A study of a distributed ledger designed for the IoT devices
- A Web of Things server to represent the device as a Virtual Thing



Flowchain

The Servient

- Composing the client and server in every single device
- Adopt the “broker service” architecture



Flowchain

Goal and Purpose

- **Decentralized IoT**
  - Centralized IoT model uses an IoT platform that acts as “hub” to control the data exchange between devices.
  - The IoT needs a decentralized IoT platform for exchanging data without any centralized party.
- **IoT Blockchain**
  - The Blockchain technology helps facilitate such challenges.

Architecture Design

Flowchain Software Framework

Distributed Ledger Layer

Flowcoin
Peer-to-peer trusted computing

Data Store
Distributed block store

Broker Server Layer

Chord
Distributed hash table

Virtual Block
Difficulty management

Miner
Proof-of-stake

Web of Things Layer
Interoperable and decentralized IoT model

Architecture Design

- **Distributed Ledger Layer**
  - Usually known as the “Blockchain”
  - Provides a distributed data store that shares transactional data across all IoT devices
- **Broker Server Layer**
  - Provides a helper library to create the IoT application server and establishes the peer-to-peer IoT networking
- **Web of Things (WoT) Layer**
  - Adopts the W3C’s WoT ontology that represents the physical IoT device as a virtual object

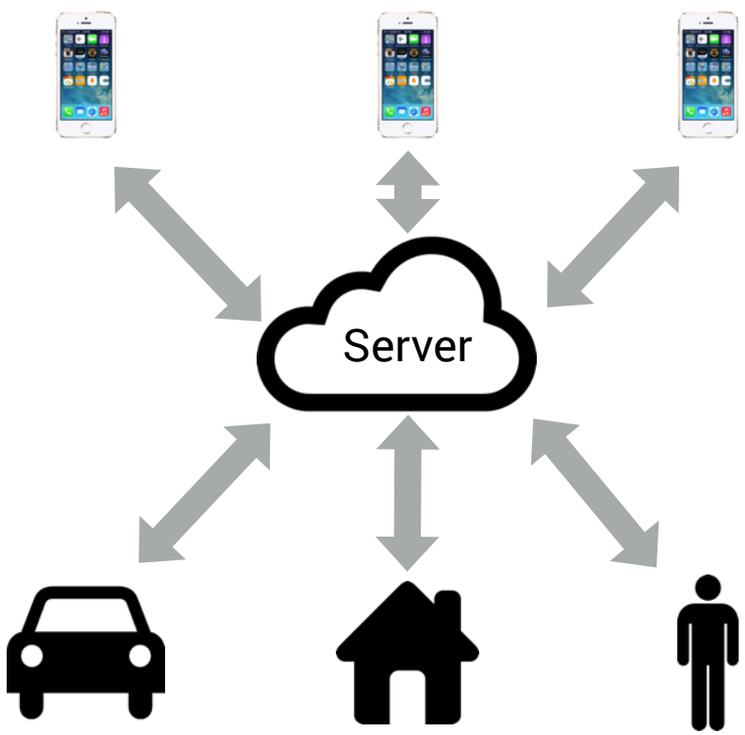
Philosophy

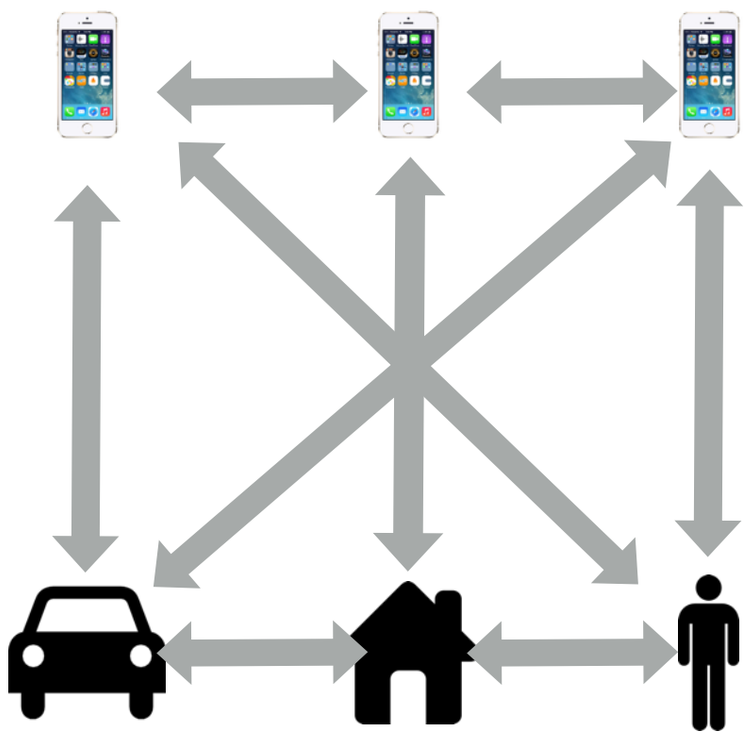
- Open source
- Open standards
- Web technologies
  - REST, JSON, JSON-LD, WebSocket, and etc
- **All in JavaScript**
  - The Flowchain software framework is a 100% JavaScript implementation

Agenda

- Motivation
- Architecture Overview
- Handling Heterogeneous Hardware
- Device Interoperability
- Distributed Ledger for the IoT
- Consensus System
- Use the Flowchain SDK

Decentralized Model





Development Milestone

- **Systems :** Hardware, FreeRTOS, JavaScript engine, MCU, Application Processor and etc.
- **P2P :** Development of the peer-to-peer network for IoT devices
- **The Ledger :** Development of the blockchain data structure for time-series data (the “virtual blocks”)
- **The Semantic Web+:** Development of the time-series database (TSDB), profiles (the ontology content patterns) …

Microprocessor System

The Ledger

RPC & DHT

Thing (WoT)

WebSocket / CoAP

OpenWRT (Linux)

Ledger Level

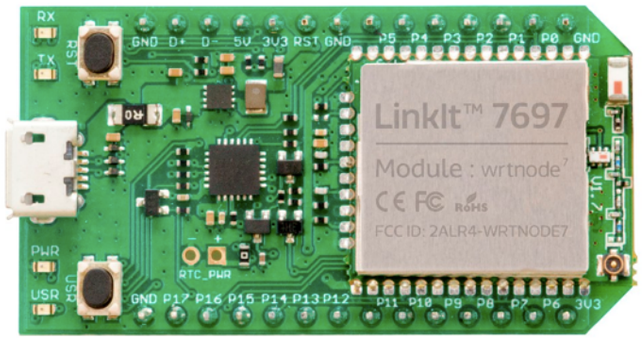
P2P Level

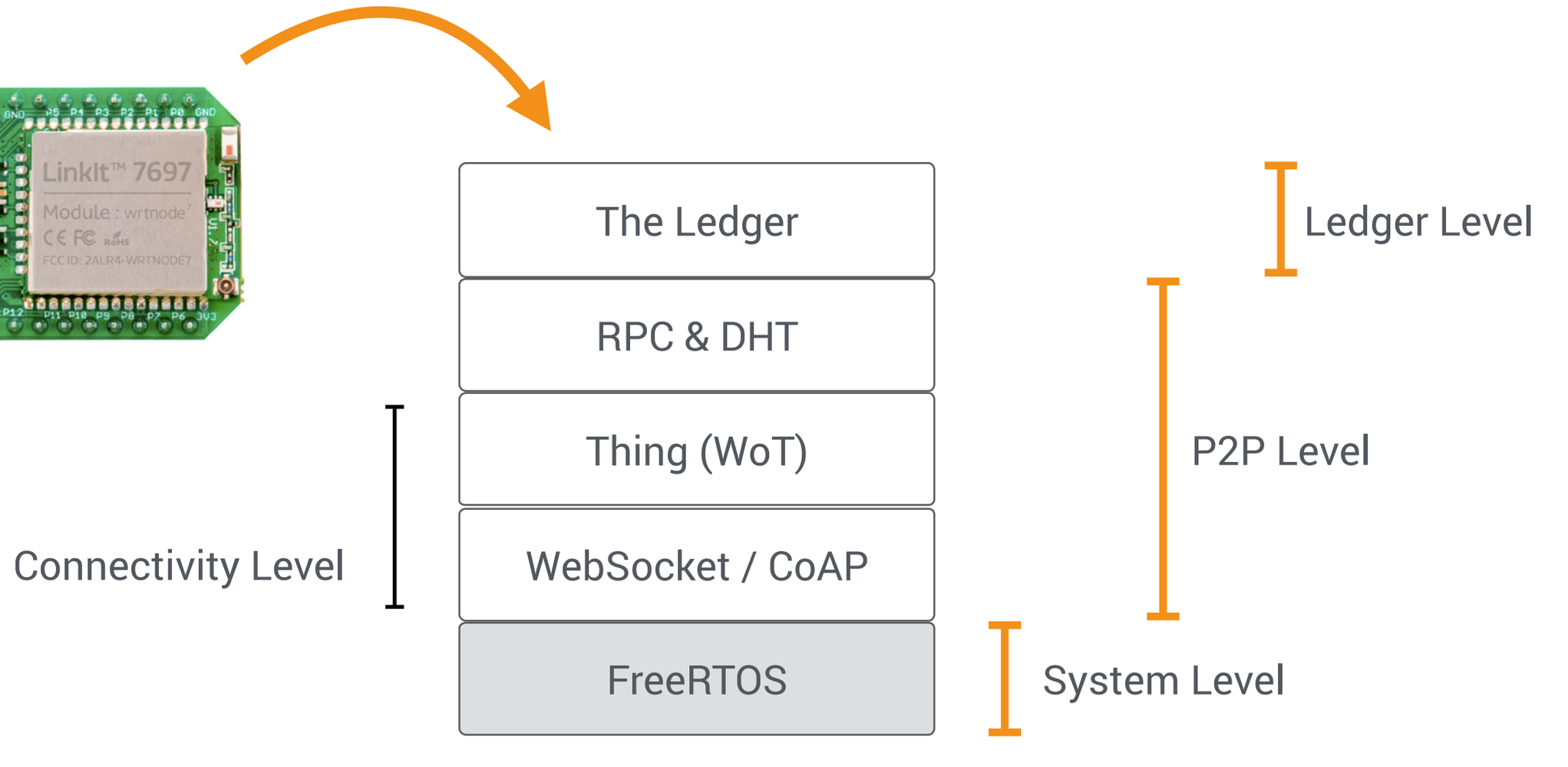
Connectivity Level

System Level

Flowchain

Microcontroller System





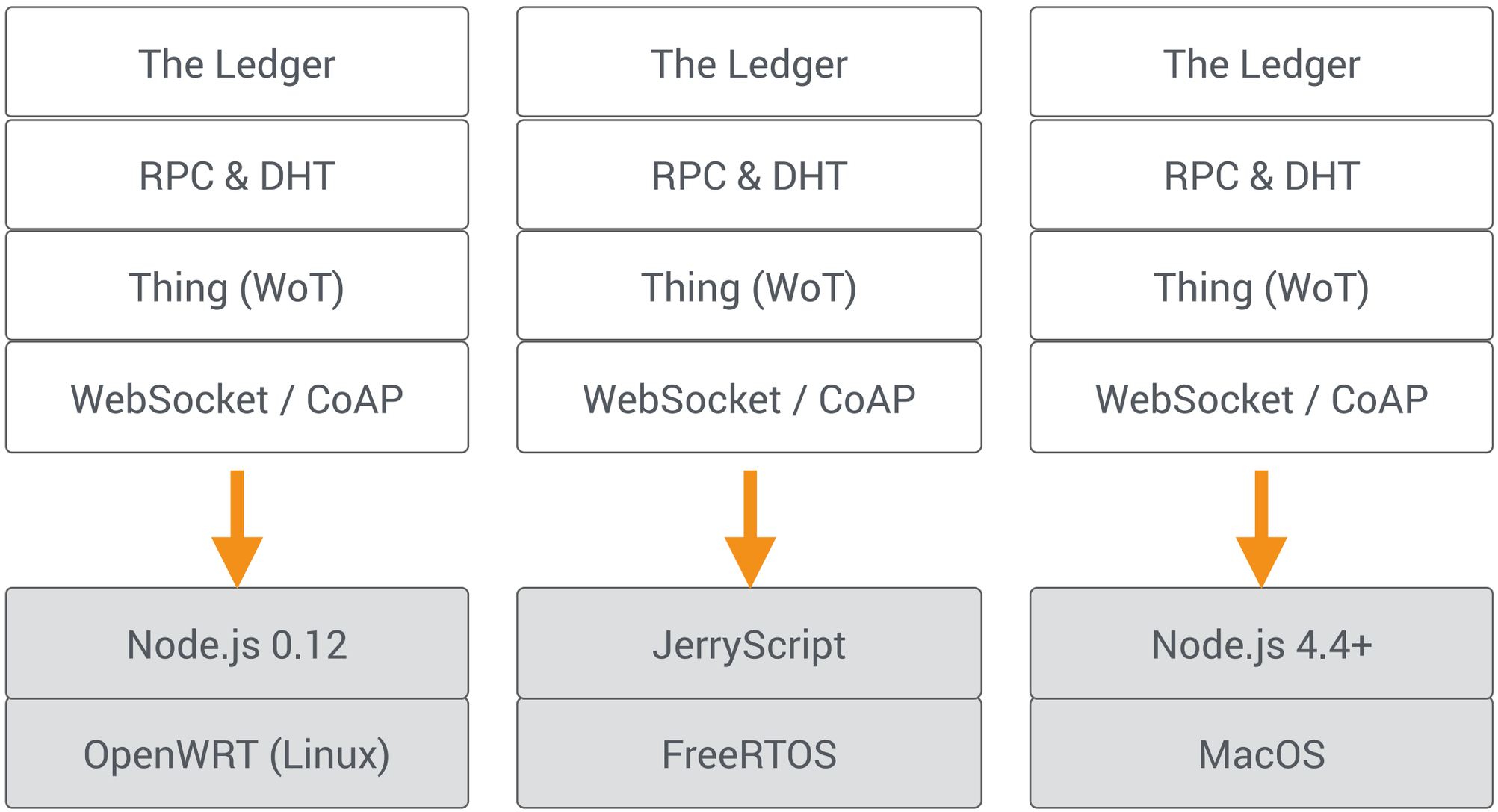
Flowchain

Heterogeneous Hardware

LinkIt Smart 7688	LinkIt 7697	Laptop
The Ledger	The Ledger	The Ledger
RPC & DHT	RPC & DHT	RPC & DHT
Thing (WoT)	Thing (WoT)	Thing (WoT)
WebSocket / CoAP	WebSocket / CoAP	WebSocket / CoAP
OpenWRT (Linux)	FreeRTOS	MacOS
MIPS Processor	ARM Cortex-M4	Intel Core 2
580MHz 128MB DDR2 32MB Flash	192MHz 352KB RAM 4MB Flash	1.4GHz 2GB DDR3 64GB SSD

Flowchain

JavaScript Runtime



Why use JavaScript ?

Flowchain

The Blockchain for the IoT Application

The Ledger	JavaScript
RPC & DHT	JavaScript
Thing (WoT)	JavaScript
WebSocket / CoAP	JavaScript

Devify

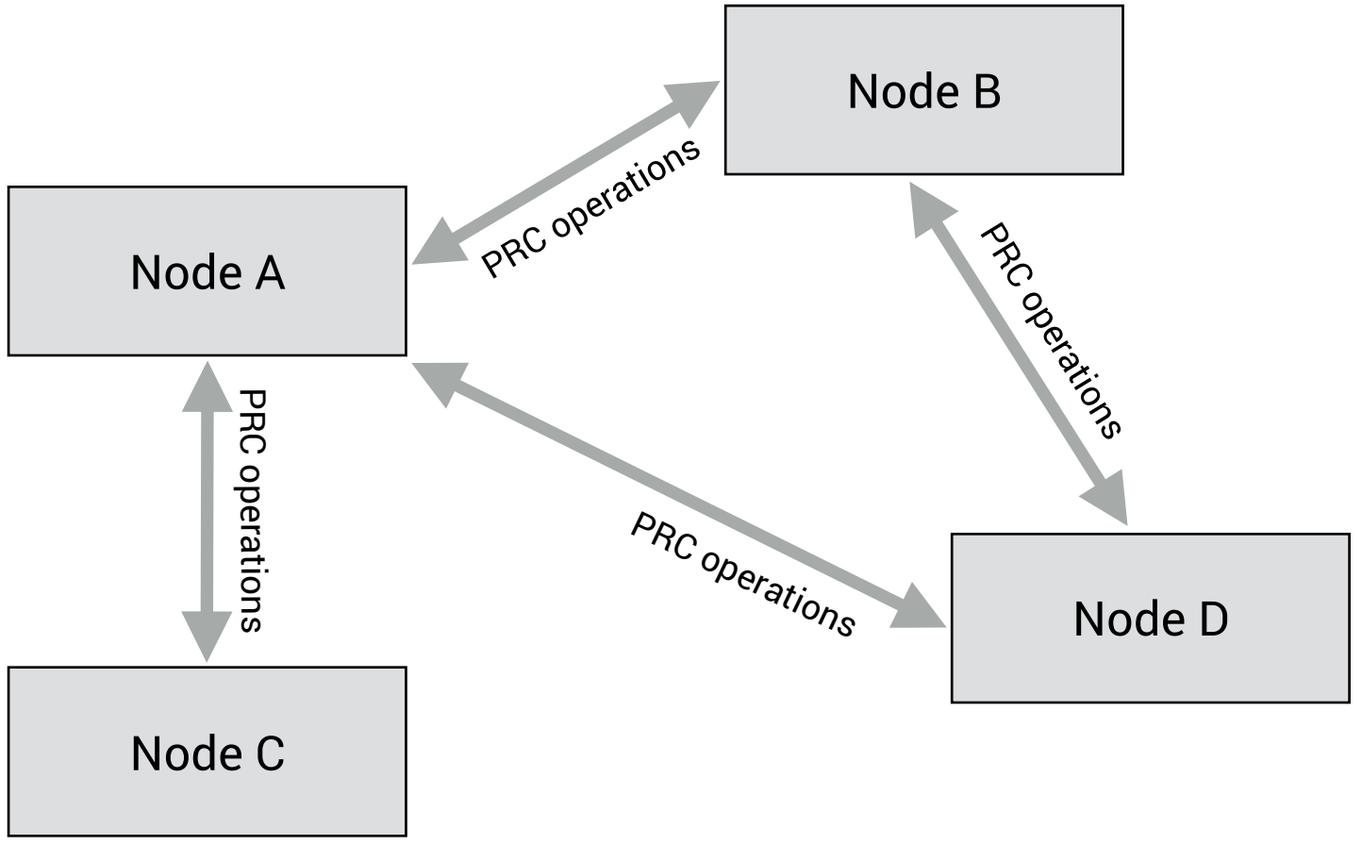
The Generic Software Framework

Agenda

- Motivation
- Architecture Overview
- Handling Heterogeneous Hardware
- Device Interoperability
- Distributed Ledger for the IoT
- Consensus System
- Use the Flowchain SDK

Overview

- RPC for the inter-device communications
- Over the Websocket and CoAP open standards

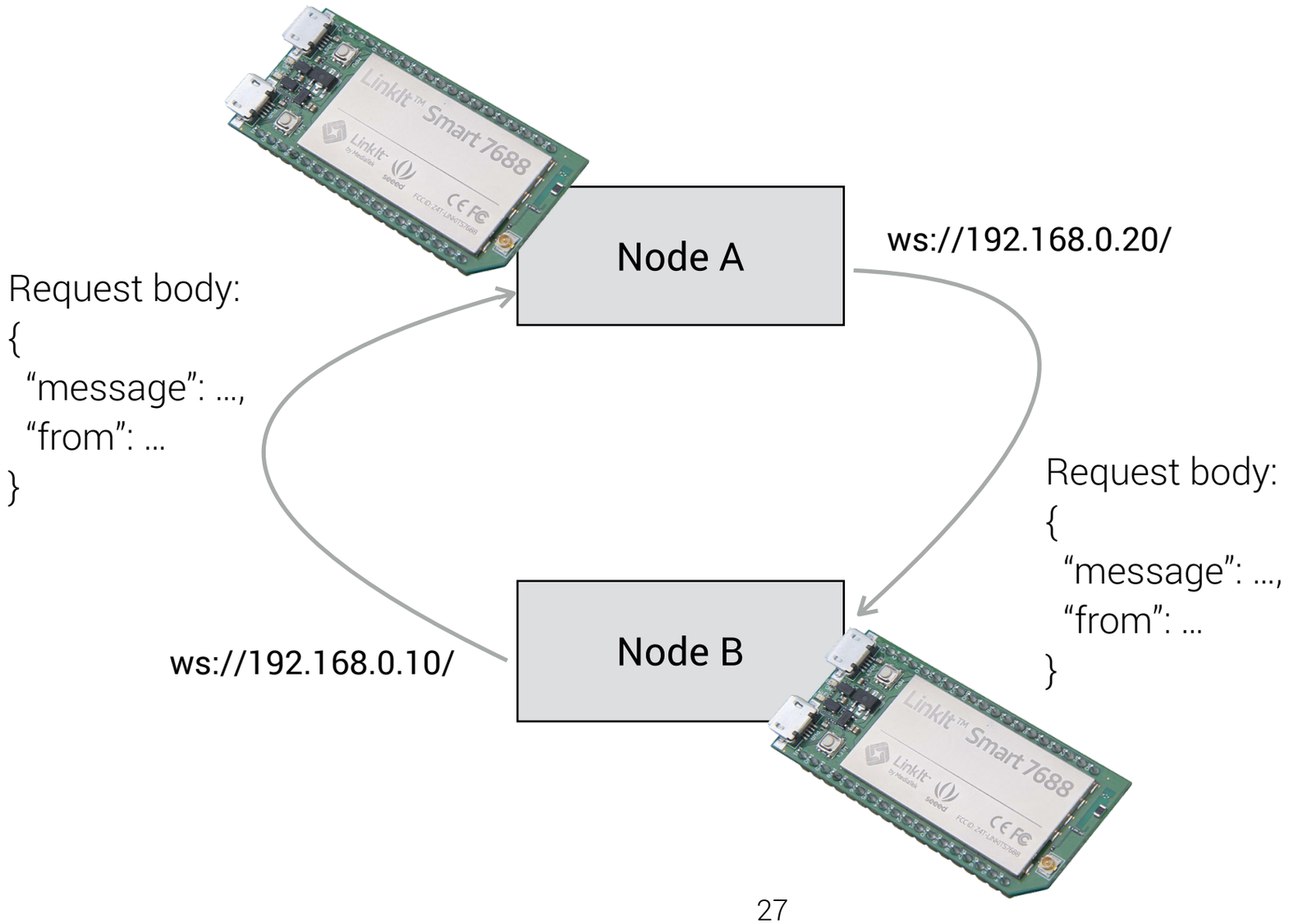


Interoperability of the IoT

- **The Interoperable IoT devices**
  - Device communications via REST-style RPC, and
  - A peer-to-peer network (p2p)
  - Transactional data is transferred in the p2p network by the REST-style RPC
    - The distributed ledger stores the transactional data in the JSON-LD format

The REST-style RPC

- RPC operations over **REST APIs**



```
/**
 * Send a RPC message.
 *
 * @param {Object} Destination
 * @param {Object} Data payload
 * @api private
 */
var send = function(to, packet) {
  var uri = util.format('ws://%s:%s/node/%s/receive', to.address, to.port, packet.message.id);
  var host = util.format('ws://%s:%s', to.address, to.port);
  var payload = {
    message: packet.message,
    from: packet.from
  };
  var connection = connections[host] || null; // Connection cache

  if (connection) {
    if (connection.connected) connection.sendUTF(JSON.stringify(payload));
    else delete connections[host];
    return 0;
  }

  var client = new WebSocketClient();

  client.on('connect', function(connection) {
    if (connection.connected) {
      connection.sendUTF(JSON.stringify(payload));
      connections[host] = connection;
    } else {
      delete connections[host];
    }
  });

  client.connect(uri, '');
};
```

RPC Operations

- RPC operations are built upon the Chord protocols
  - RPCMessage.NOTIFY_STABILIZE
  - RPCMessage.NOTIFY_PREDECESSOR
  - RPCMessage.NOTIFY_SUCCESSOR
  - RPCMessage.NOTIFY_JOIN
  - RPCMessage.FIND_SUCCESSOR
  - RPCMessage.FOUND_SUCCESSOR
  - RPCMessage.CHECK_PREDECESSOR
  - RPCMessage.CHECK_SUCESSOR
  - RPCMessage.CHECK_TTL
  - RPCMessage.MESSAG

Chord Algorithm Extension

- Handling Churn
- The continuous activities of node join and leave

```
// It is called periodically.
// n asks the successor
// about its predecessor.
n.stabilize()
    x = successor.predecessor;
    if (x is in (n, successor))
        successor = x;
        successor.notify(n);

// n' thinks it might be our predecessor, and
// n notify n' about its alive.
n.notify(n')
    if (predecessor is nil or n' is in (predecessor, n))
        predecessor = n';
    n'.notify_ttl();

// n updates the successor's TTL.
n.notify_ttl()
    n.successor_ttl = MAX_TTL;
```

Implement RPC Operations

- RPC operations are built upon the Chord protocols

```
// called periodically. n asks the successor
// about its predecessor, verifies if n's immediate
// successor is consistent, and tells the successor about n
n.stabilize()
  x = successor.predecessor;
  if (x∈(n, successor))
    successor = x;
  successor.notify(n);
```

Source: https://en.wikipedia.org/wiki/Chord_(peer-to-peer)

```
switch (message.type) {
    case RPCMessage.NOTIFY_STABILIZE:

        if (this.predecessor === null) this.predecessor = from;

        if (ChordUtils.isInRange(this.predecessor.id, from.id, this.id)) {
            message.type = Chord.NOTIFY_PREDECESSOR;
            return this.send(this.predecessor, message, from);
         }

         message.type = Chord.NOTIFY_SUCCESSOR;
         this.send(from, message, this);

         break;
   ...
}
```

Agenda

- Motivation
- Architecture Overview
- Handling Heterogeneous Hardware
- Device Interoperability
- Distributed Ledger for the IoT
- Consensus System
- Use the Flowchain SDK

Preliminary

- A blockchain is only one type of data structure considered to be a distributed ledger
- A distributed ledger technology (DLT) provides a new data structure varies from the blockchain for various purposes

DLT Essentials

- A peer-to-peer network+
- Consensus algorithms

Blockchain Data Structure

- Ordered and back-linked list
- Stored in a flat file or database
- Each block is identified by a hash
- Double SHA-256 hash function

Blockchain for the IoT

	Bitcoin Blockchain	Flowchain DLT
Datastructure	Ordered and back-linked list	V
Datastore	A flat file or database	Block Store
Block Identify	Hash	Hash
Consensus	POW	Mining-based POS
Data Transaction	Unverified pool	Unverified chunk data

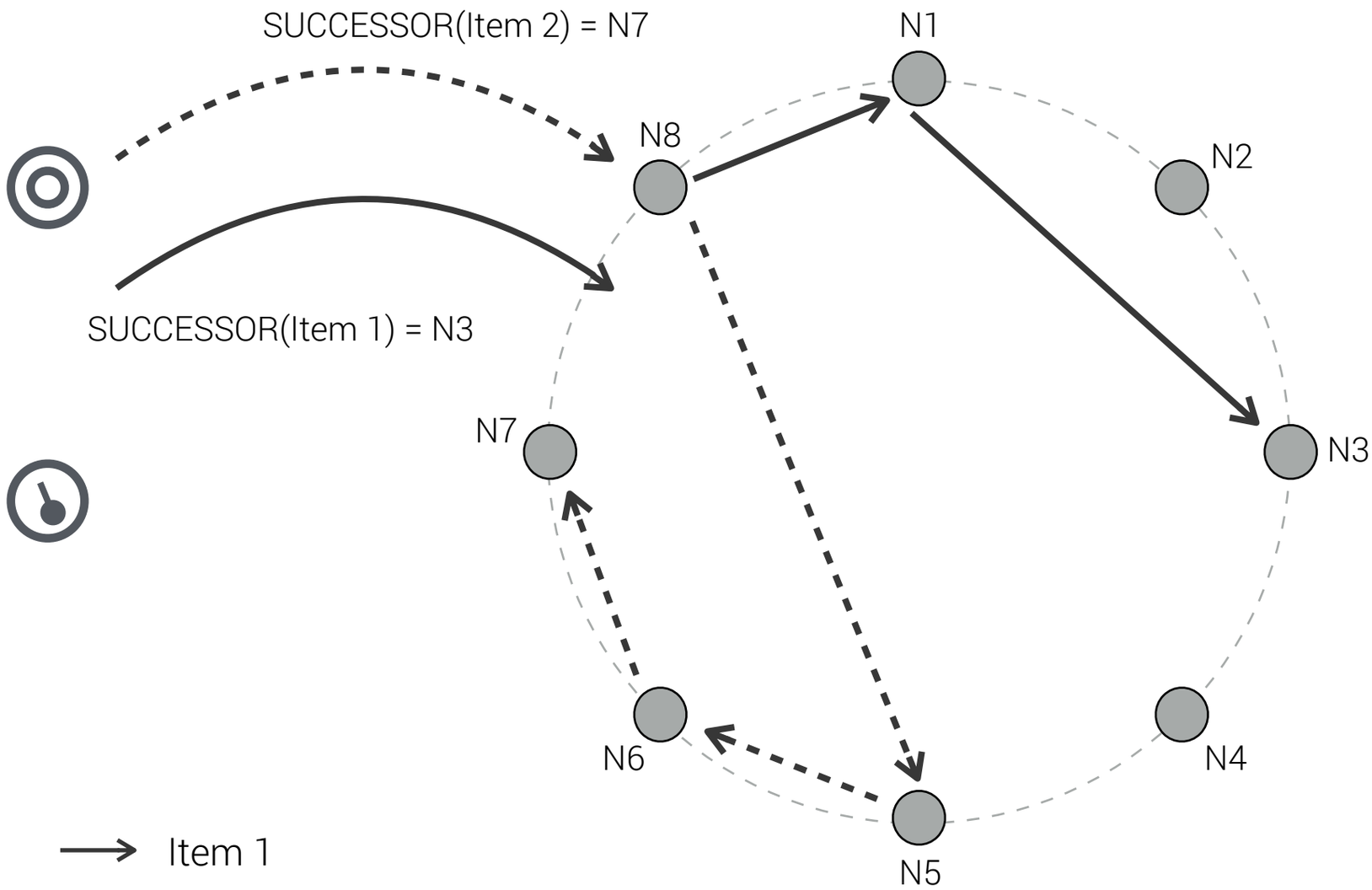
Flowchain

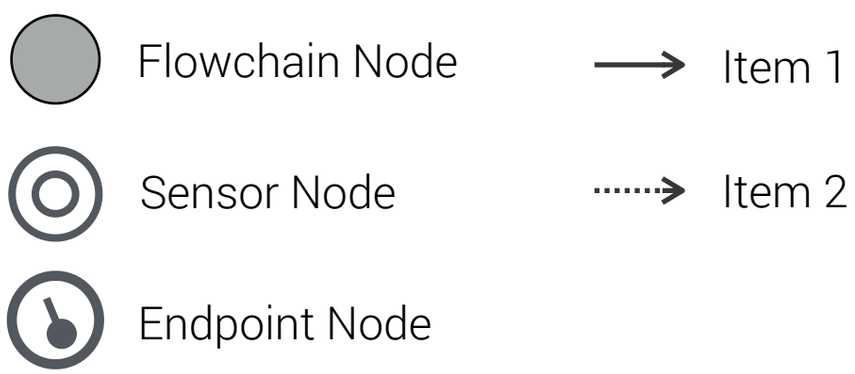
Data Transaction

- Use the Chord algorithm to organized the IoT devices as a “Ring” topology
- Each device maintains a “finger-table” (aka the DHT)
- **The data transaction process**
  - Step 1: Generate the key of the data by SHA-256
  - Step 2: Search the successor node of the key in the DHT
  - Step 3: Send data to the successor node by the RPC operation
  - Step 4: The successor node processes the data transaction

Peer-to-Peer Networking

- RPC for the IoT devices
- Over the Websocket and CoAP open standards





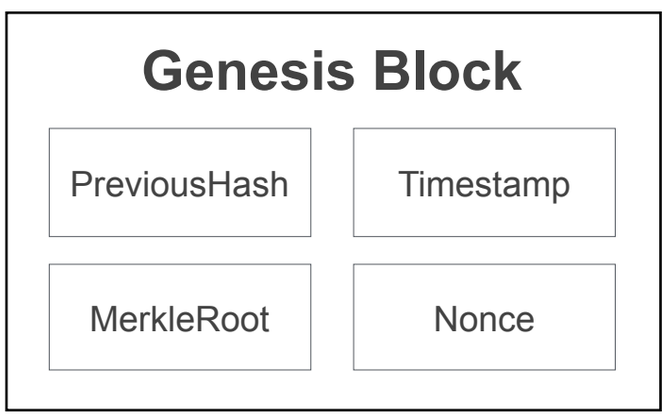
Flowchain

Flowchain IoT Ideas

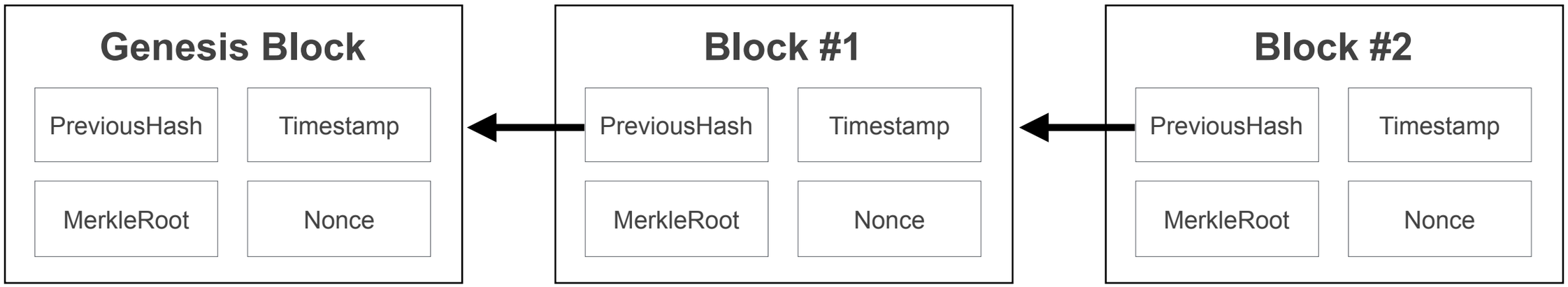
- Avoid “competition” and “mining fork” exception
- Propose and use “virtual blocks” concepts
  - Aka the “virtual mining”
- Fork the “genesis block” and find blocks (mining) at the “branches”

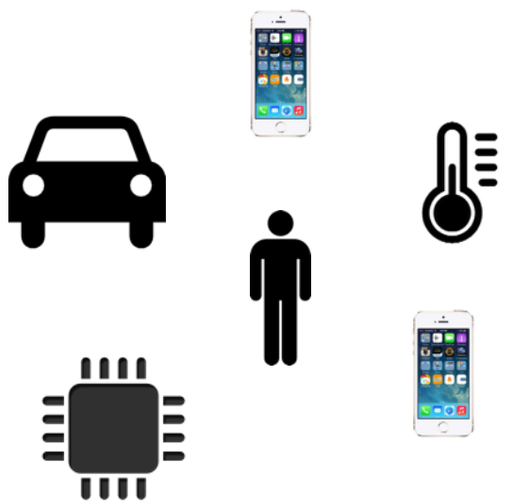
Establish Genesis Block


```
function Block(block) {
    if (typeof block === 'undefined') {
        block = {};
    }

    this.hash = block.hash || '';
    this.previousHash = block.previousHash || '';
    this.timestamp = block.timestamp || new Date();
    this.merkleRoot = block.merkleRoot ||
'0000000000000000000000000000000000000000000000000000000000000000';
    this.difficulty = block.difficulty ||
'00FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF';
    this.nonce = block.nonce || 0;
    this.no = block.no < 0 ? 0 : block.no;
}
```

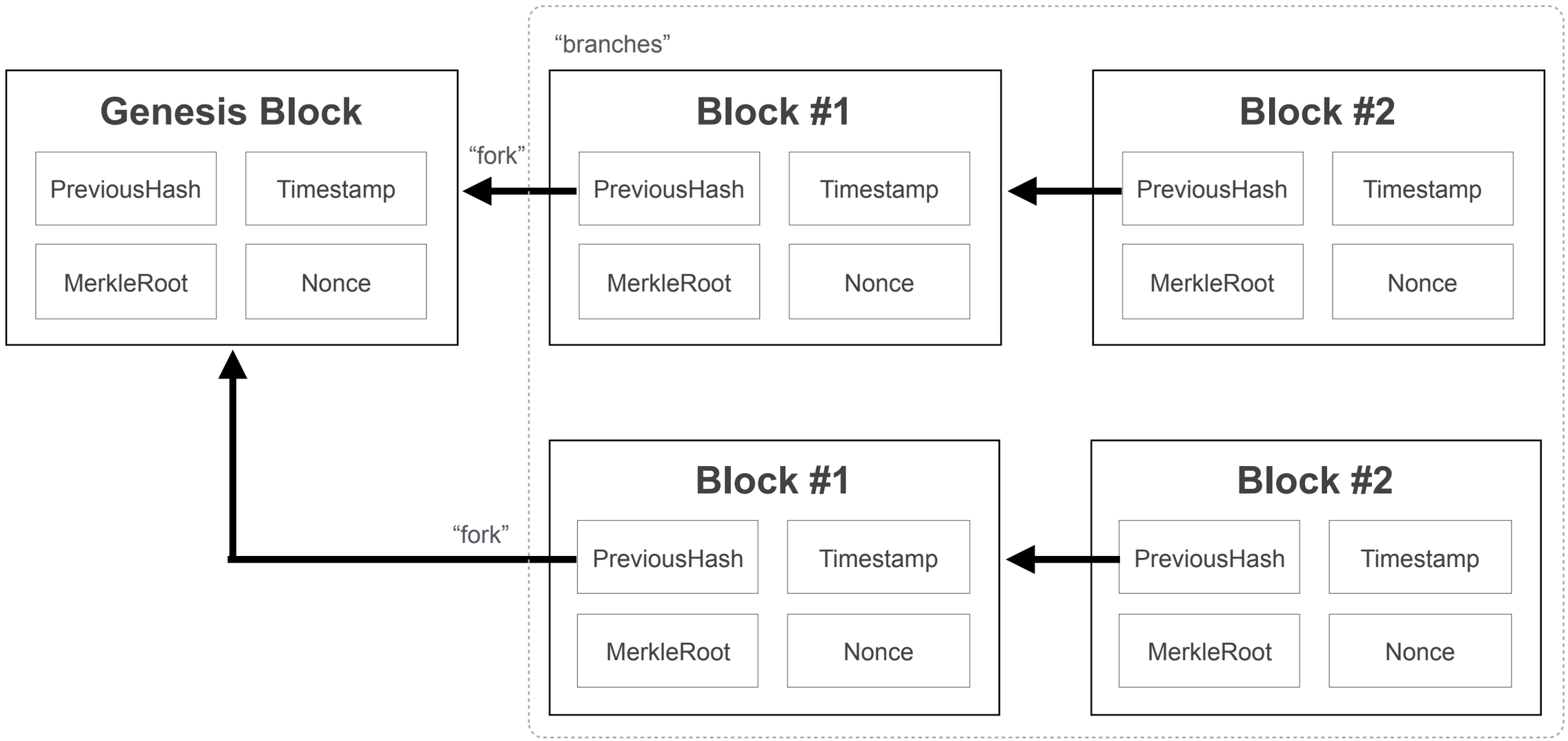


Bitcoin Blockchain Data





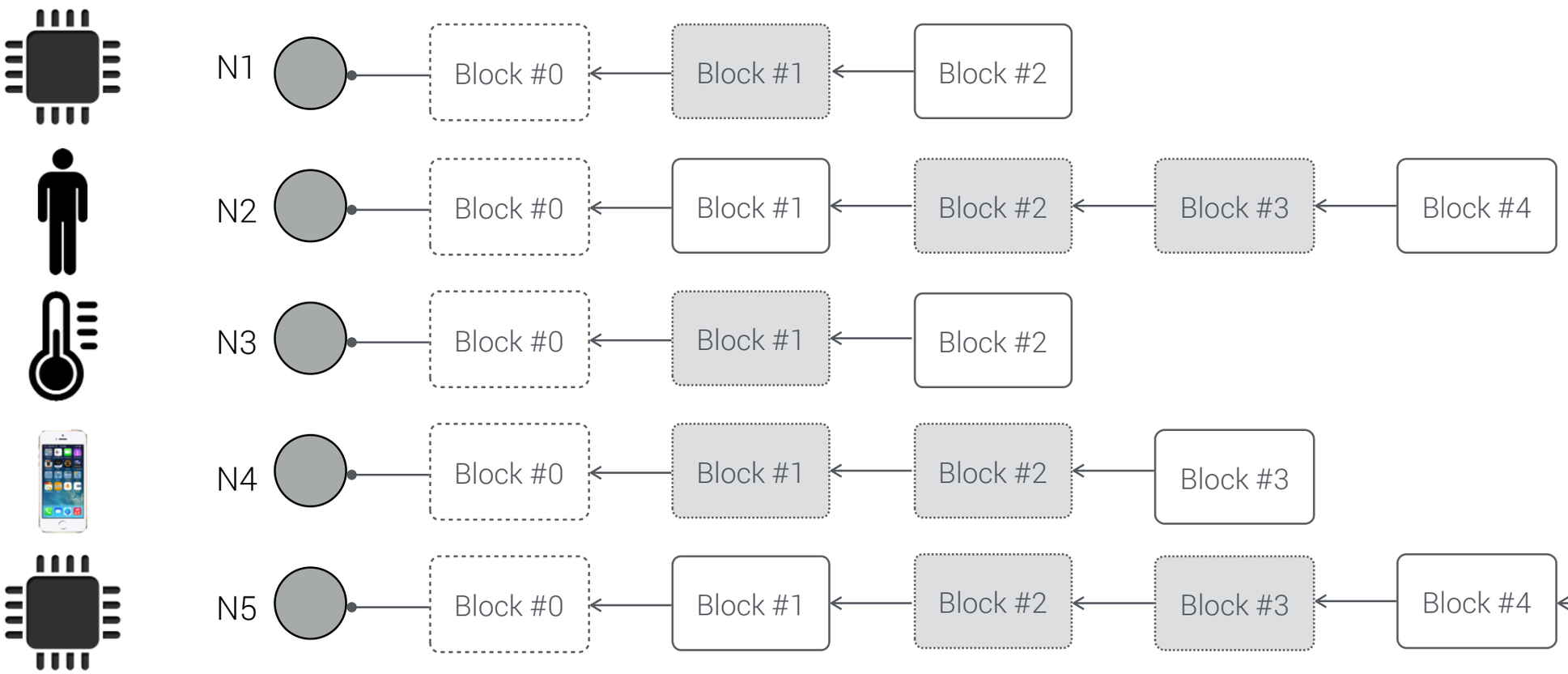
Flowchain Virtual Blocks



Real-Time Transaction

- Avoid “competition” and “mining fork” exception
  - Flowchain mining is nonblocking
- Propose and use “virtual blocks” concepts
  - Aka the POS “virtual mining”
  - Fork the “genesis block” and find blocks (mining) at the “branches”
  - Branches can be merged by hash join algorithm

Flowchain Virtual Blocks



Flowchain

Use the Virtual Blocks

LINUXCON containercon CLOUDOPEN CHINA 2017

```
/*
 * Flowchain virtual block algorithm (the pseudo code)
 */

Node.on(‘message’, function(key, value) {
  // Get a valid block of the device’s blockchain
  N = GetOneValidBlock(chains)

  // Put key-value pair in block “N”
  PutToBlock( N, { key: value } );
});
```


Flowchain

“Virtual Blocks”

- Five IoT devices are labeled N1 to N5, and each device is a “node” in a peer-to-peer network
- All nodes are mining blocks that use the same genesis block
- In other words, each node creates a new “branch” for mining; thus, there is no blockchain “fork”
- Every block in each branch is called a Virtual Block
- Virtual Blocks can be labeled as valid or invalid
- Only valid blocks are available to record transactions

Why “Virtual Blocks” ?

- Flowchain initially creates branches for each node when nodes mine their Virtual Blocks
- Estimate the block “forks” exception during the mining process
- Flowchain can act in a real-time manner

Data Transaction

- Use the Chord algorithm to organized the IoT devices as a “Ring” topology
- Each device maintains a “finger-table” (aka the DHT)
- **The data transaction process**
  - Step 1: Generate the key of the data - $\mathbf{H}_{DATA}$
  - Step 2: Search the successor node of the key in the DHT - SUCCESSOR($\mathbf{H}_{DATA}$)
  - Step 3: Send data to the successor node by the RPC operation
  - Step 4: The successor node processes the data transaction

Generating Transaction ID

- Use SHA256, SHA1, and Double SHA256
- The **H**DATA hash key is generated by the Chord algorithm

$$\mathbf{H}_{BLOCK} = \mathbf{SHA256(}\ BlockNo + timestamp + nonce\ \mathbf{)}$$

$$\mathbf{H}_{DATA} = \mathbf{SHA1(}\ data + timestamp + ramdom\ \mathbf{)}$$

$$\mathbf{H}_{txID} = \mathbf{SHA256(\ SHA256(\ H}_{BLOCK} + \mathbf{H}_{DATA}\ \mathbf{)\ )}$$

successor(HDATA) [1]

- Handling Churn
- the continuous activities of node join and leave

```
// ask node n to find the successor of HDATA
n.successor(HDATA)
//It is a half closed interval.
 if (id ∈ (n, successor] )
   return successor;
 else
   // forward the query around the circle
   n0 = closest_preceding_node(id);
   return n0.find_successor(id);
```

[1]: https://en.wikipedia.org/wiki/Chord_(peer-to-peer)

Simulating Data Transaction

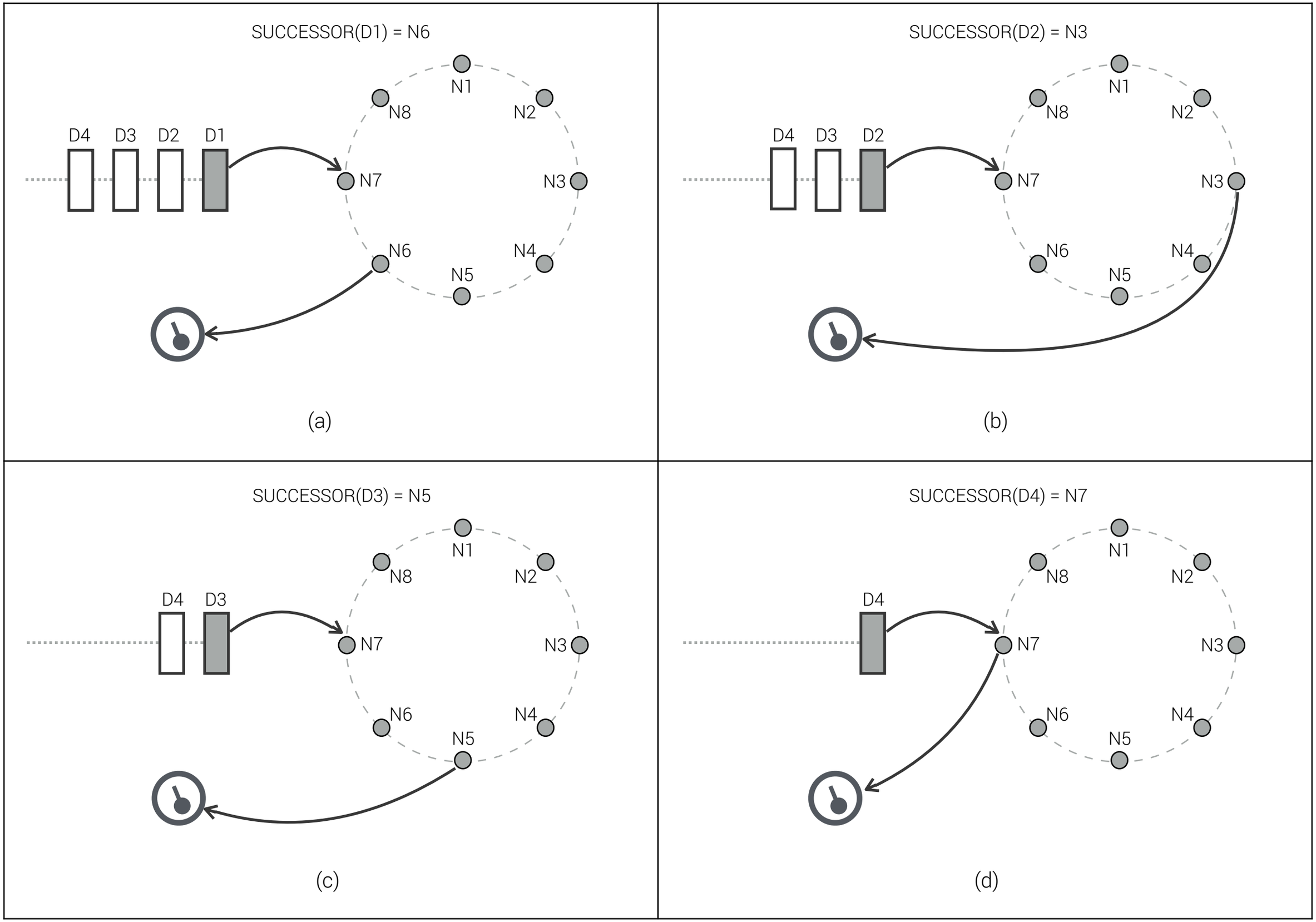
- The successor node is represented as N’
- N' receives the chunk data, and combines the valid block ID and the data key to generate a transaction ID
- N’ signs the transaction with its private key embedded in the hardware
- N’ creates a record that comprises the transaction ID and the chunk data and stores the record in a valid block

Linked Data for Transactions


```
N.put(data) {
  key = hashDataKey(data);

  // Send chunk data to N' over the Chord ring.
  send( SUCESSOR(key), { payload: data } );
}

N'.PutToBlock(block, doc) {
  db = DatabaseAdapter.getDatabase();

  txID = SHA256( SHA256( block.id + doc.key ) );

  tx = new Transaction( doc.value );
  tx.sign( privateKey );

  record = {
    "@context": "http://flowchain.io/ledger-context.jsonld",
    "txID": txID,
    "tx": tx
  };

  db.put( record );
}
```

Vaild Block Lookup

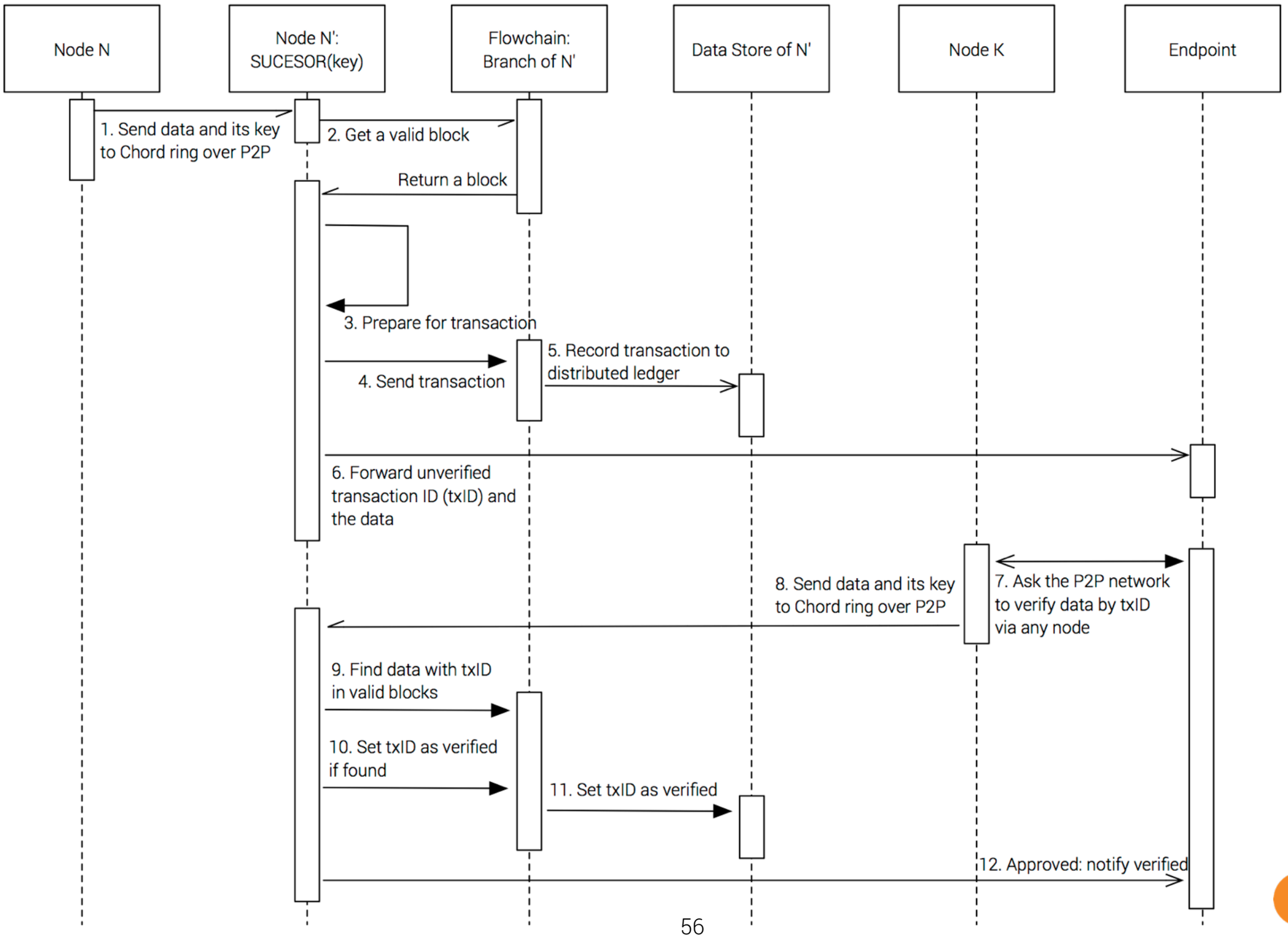

```
N’.lookupBlock(doc) {
  db = DatabaseAdapter.getDatabase();

  var blocks = db.queryValidBlocks();

  blocks.forEach(function(block) {
    if (this.PutToBlock(block, doc) === true)
      return;
  });
}
```


SUCCESSOR(D1) = N6
D4
D3
D2
D1
N1
N2
N3
N4
N5
N6
N7
N8
(a)
SUCCESSOR(D2) = N3
D4
D3
D2
N1
N2
N3
N4
N5
N6
N7
N8
(b)
SUCCESSOR(D3) = N5
D4
D3
N1
N2
N3
N4
N5
N6
N7
N8
(c)
SUCCESSOR(D4) = N7
D4
N1
N2
N3
N4
N5
N6
N7
N8
(d)
Flowchain Node
Endpoint Node
Flowchain

Ontology: Auditing Transactions

- The DLT can be “auditable”
- Use case example
  - The “administrator” approve the data transaction
  - The “monitor process” performs a “branch merge “operation that all “virtual blocks” merge imperceptibly into a single blockchain

The Auditing Pattern



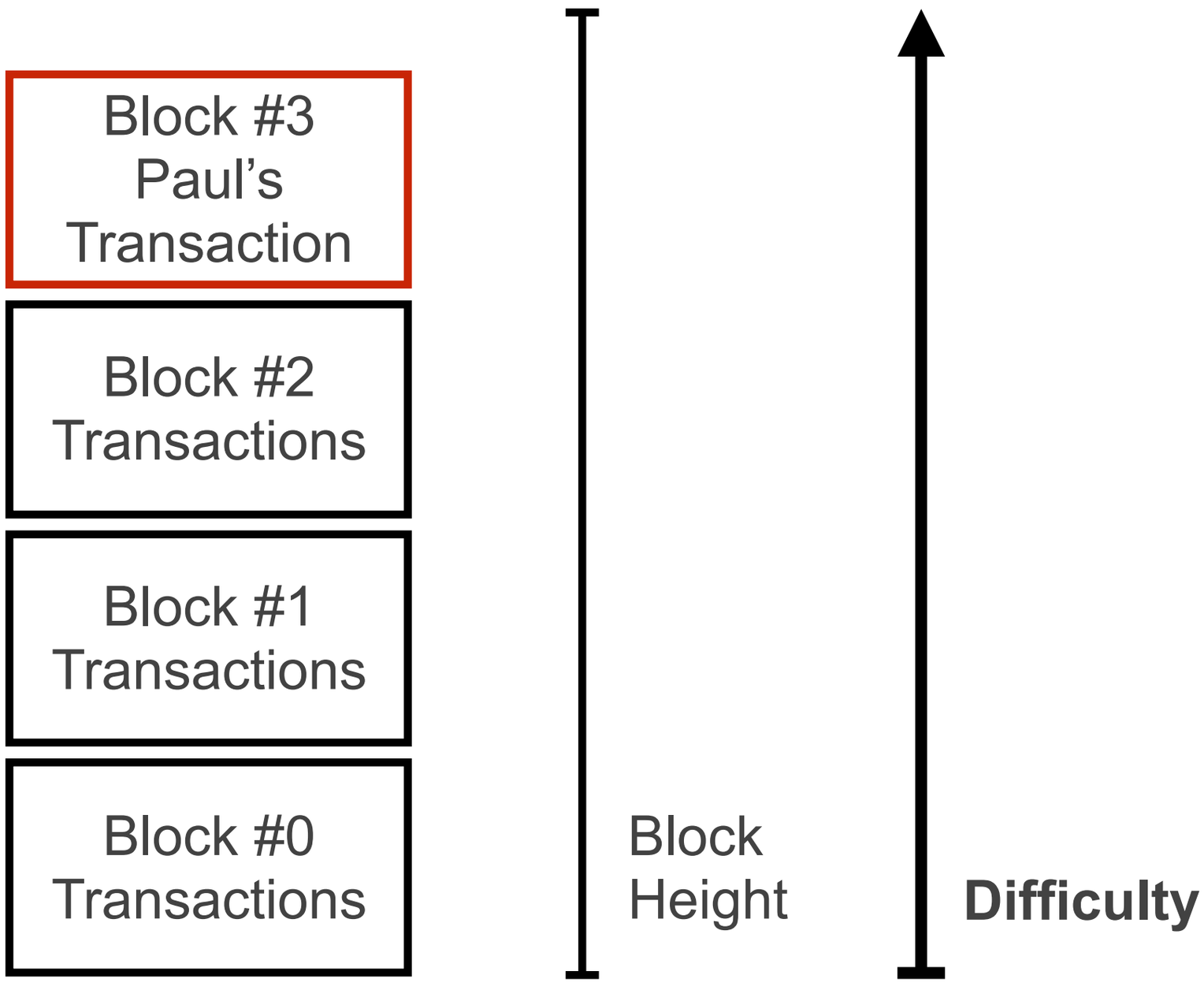
Agenda

- Motivation
- Architecture Overview
- Handling Heterogeneous Hardware
- Device Interoperability
- Distributed Ledger for the IoT
- Consensus System
- Use the Flowchain SDK

Understanding Block Time

- The “time” to find a valid block
  - Often know as the “mining”, and
  - Difficulty control
- Blockchain-based consensus system
  - Aka mining-based consensus system

Understanding “Difficulty”



“Difficulty Control”

- The mining-based PoS system will use the “difficulty” concept derived from the PoW system
- Flowchain is the mining-based PoS system
- Used in the consensus mechanism

Consensus Mechanisms

- Proof-of-Work (PoW)
- Proof-of-Stake (PoS)
- Proof-of-[Resource, Existence, Reliability, and etc]
- The “hybrid”
- Practical Byzantine Fault Tolerance (PBFT)
- Paxos / The Part-Time Parliament
- Delegate Proof-of-Stake (DPoS)

Consensus for the IoT

- A mining-based proof-of-stake
  - Minimal resource requirement
  - Device reliability
  - etc

Flowchain Consensus System

- The miner is timed in a fixed number calculation per second in which the Flowchain can comprise a proof-of-stake mechanism
- Flowchain implements a mining-based proof-of-stake consensus system
- Use the probability density function to ensure a cost-effective difficulty control system

Technical Challenges

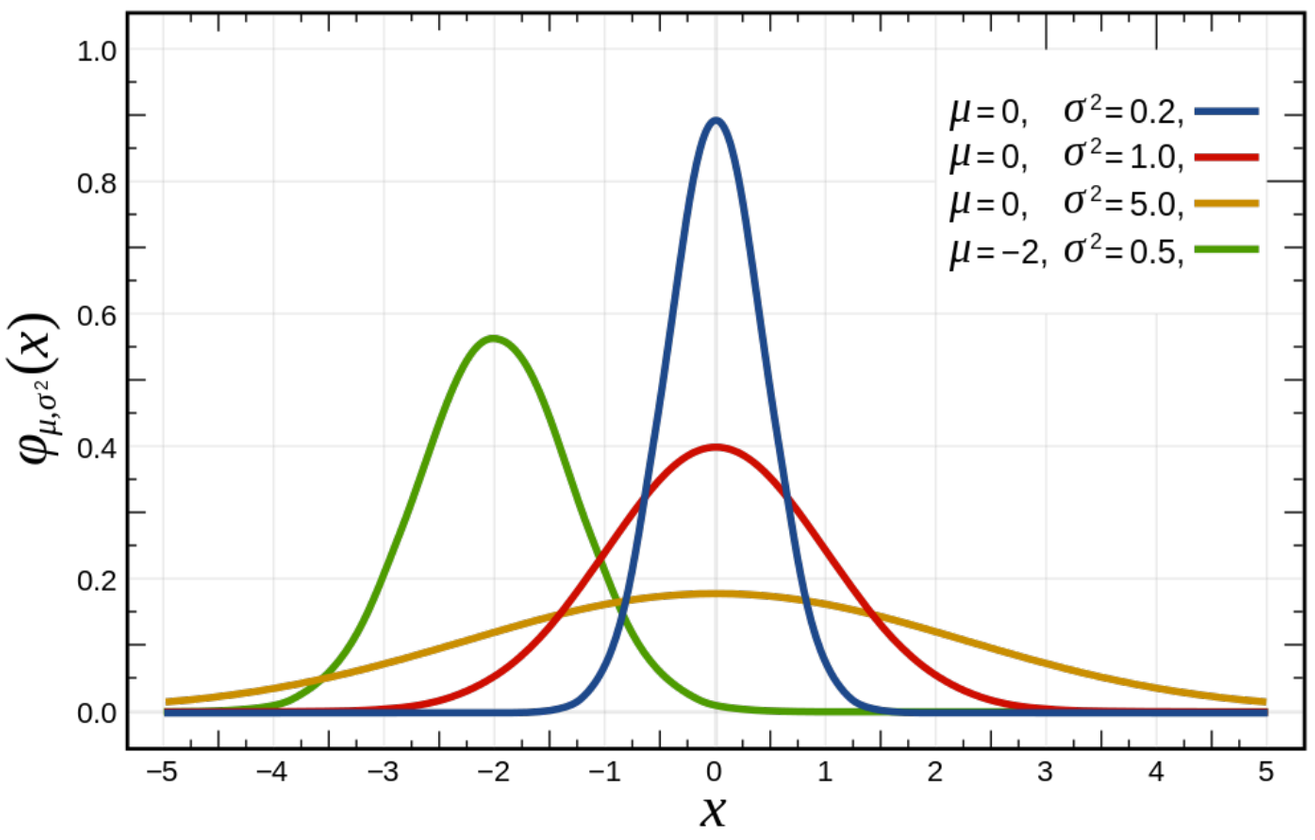
- Resource-constrained devices
  - Limited computation power
  - Limited resource (memory and etc)
- Difficulty control for a mining-based PoS on the IoT

Hashing Rate

- Problem-solving
  - Brute-force
  - Memory-hard functions
  - Probability distribution

Probability Density

- Probability density to control the mining difficulty
  - Normal
  - Poisson



Source: https://en.wikipedia.org/wiki/Normal_distribution (License: Public Domain)

Flowchain Difficulty Control


```
var gaussian = require('gaussian');

// the mean (μ) of the distribution
var mean = 0;

//the variance (σ^2) of the distribution
var variance = 0.2;

var distribution = gaussian(mean, variance);

function Difficulty(x) {
    if (!x) x = Math.random();

    x = x - variance;

    var probability = distribution.pdf(x);

    return fixDifficulty(probability);
}
```

Flowchain PoS for the IoT


```
var Miner = require('./VirtualMiner');      // Import flowchain miner

  // Create a new miner
  this.miner = new Miner();

  miner.setPreviousBlock(block);

  setInterval(function() {
      miner.generateHash();

      // A success hash is generated
      if (miner.isSuccess()) {
          var block = miner.getNewBlock();

          // Successful mined and save the new block
          self.blockchain.push(block);

          miner.setPreviousBlock(block);
      } else {
          var block = miner.getMiningBlock();
    }
}, 50);
```

Permissioned DLT

- The p2p network authorizes nodes intending to join the network that only authorized nodes can operate as a Flowchain node
- Flowchain ensures the interoperability between different ledgers within the same IoT peer-to-peer network
- Flowchain DLT transactional protocols has been built upon the Chord algorithm and protocol

Agenda

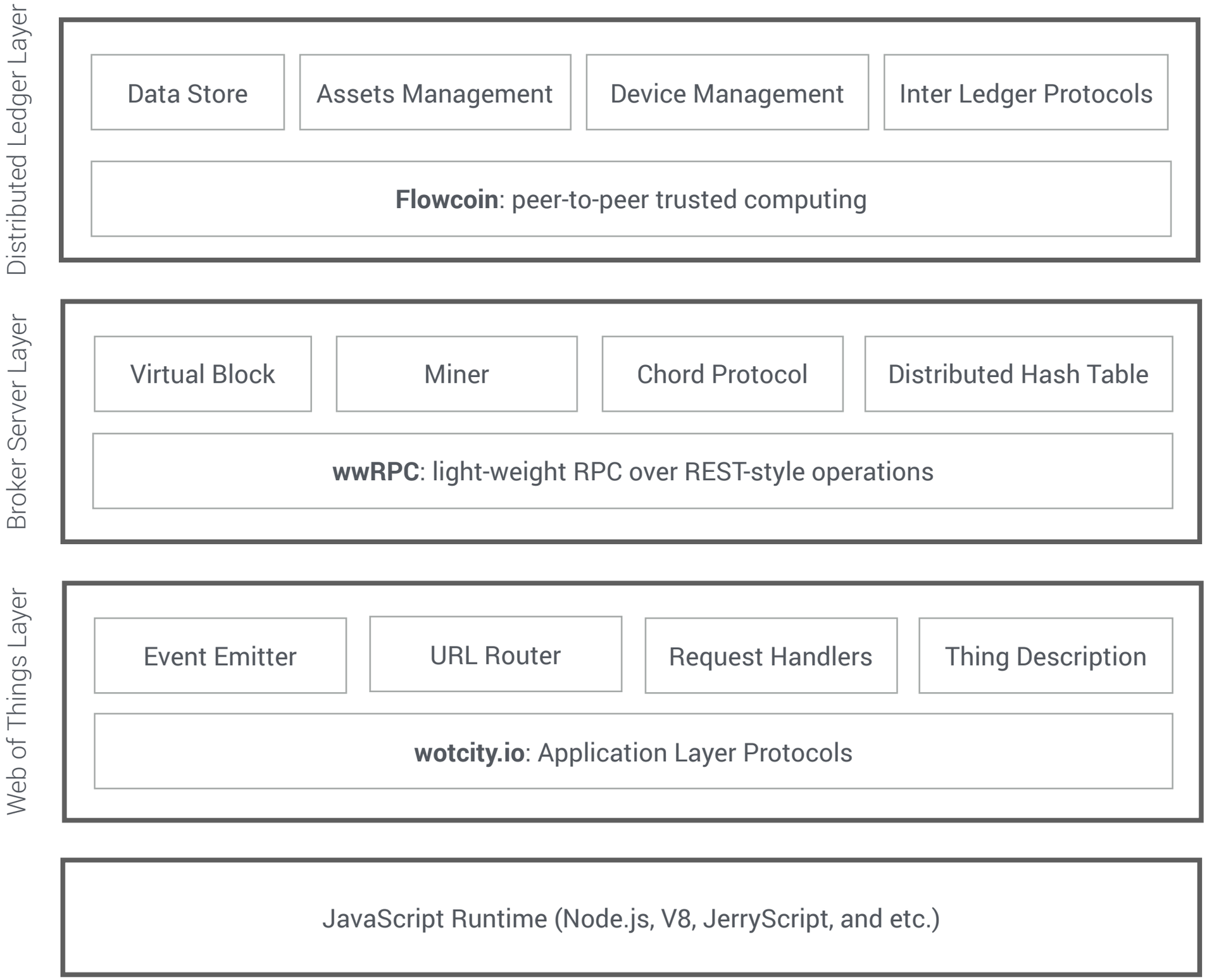
- Motivation
- Architecture Overview
- Handling Heterogeneous Hardware
- Device Interoperability
- Distributed Ledger for the IoT
- Consensus System
- Use the Flowchain SDK

Flowchain Open Source

- A distributed ledger for the IoT
- A programming framework
- Software architecture designed from the ground up
- Visit https://flowchain.io

Flowchain

Flowchain Software Framework



Flowchain

Architecture Design

- **Distributed Ledger Layer**
  - Usually known as the “Blockchain”
  - Provides a distributed data store that shares transactional data across all IoT devices
- **Broker Server Layer**
  - Provides a helper library to create the IoT application server and establishes the peer-to-peer IoT networking
- **Web of Things (WoT) Layer**
  - Adopts the W3C’s WoT ontology that represents the physical IoT device as a virtual object

TSDB with the Semantic

- A Linked Data document to support time series database (TSDB) via the semantic web technology
- the TSDB can use the semantic object in ```@context``` as the NoSQL schema design to index transaction records

Object ID	Timestamp	Transaction ID (offset 0-31)	Transaction Data
		+0	+32
001	1492041600	c20c44b5ba7a34e7ddd7ec8cbd2203d3	tx1
002	1492041600	21b15ca036d6c144fc14b6b7fb201290	tx2
003	1492041600	905a862315e2c58fcb8038792be3951b	tx3
004	1492041610	685433e9d3fa916fdca73ebe8efd878a	tx4
005	1492041610	7b46c90b7ac12d3ad47c4cb7645c5741	tx5
006	1492041650	7d2c0a88166f5b267613ea51f455adf7	tx6
007	1492041660	fb2cc23fe3f078aea367123a48799dd5	tx7

Flowchain Open Source

- A distributed ledger for the IoT
- A programming framework
- Software architecture designed from the ground up
- Visit https://flowchain.io

Flowchain

Start an IoT Node


```
// Import the broker server middleware
var server = require('./server');

// Utils
var crypto = require('crypto');

// Database
var Database = require('./database');
var db = new Database('picodb');

// Start the IoT application server and start to find blocks
server.start({
    onstart: onstart,
    onmessage: onmessage,
    onquery: onquery,
    ondata: ondata,
    join: {
        address: '10.186.110.91',
        port: '8000'
    }
});
```

Store Transactions


```
// Use DatabaseAdapter to select a data store
var Database = require('./database');
var db = new Database(‘picodb');

var onmessage = function(req, res) {
    var payload = req.payload;
    var block = req.block;
    var node = req.node;

    // Key of the data
    var key = message.id;

    // Data
    var tx = message.data;

    // Block hash as the secret and data key as the context
    var hash = crypto.createHmac('sha256', block.hash)
                     .update( key )
                     .digest(‘hex');

    db.put(hash, tx, function (err) {
        if (err)
            return console.log('Ooops! onmessage =', err)
    });
}
```

Flowchain Mission

IoT Devices themselves in a decentralized IoT platforms can have a new model to exchange data.

[device democracy] Blockchain IoT technology provides such new model for secured and trusted data exchange that keep trusted records of all exchanged data between devices.

Future Work

- Open issues, contributions, and review comments through the open source development methodology
- Time-series database for the distributed ledger
- Benchmark tools
- User Guide and Developer Docs
- Developing Ontology Patterns
- Blockchain IoT Kit in Q4 2016 along with a fine-grained software framework

LINUXCON containercon CLOUDOPEN

CHINA 2017

THINK OPEN

THANK YOU!

WeChat: jollentw
Email: jollen@flowchain.io

LF ASIA, LLC