# Zero-Knowledge Proof Schemes for Lightweight IoT Devices in Permissioned Blockchain Networks

NGUYEN Viet Hoa

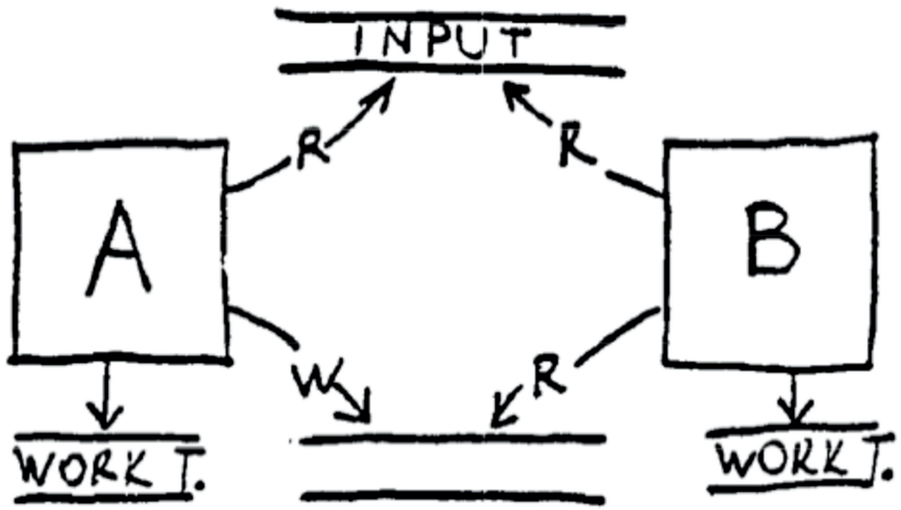
Feb. 12 | 2025

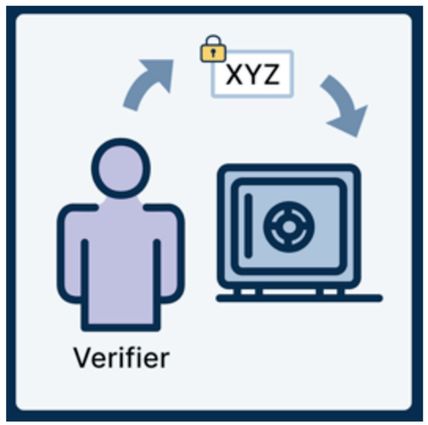
# CONTENT

# What is Zero Knowledge Proof (ZKP)

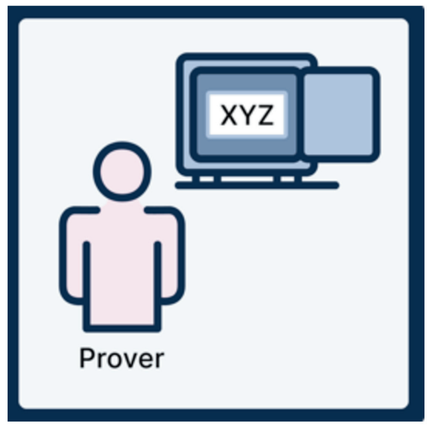
Zero-knowledge in cryptography first appeared in the 1985 paper "***The knowledge complexity of interactive proof systems [GMR85]***" by pioneers Shafi Goldwasser et al:

**"A zero-knowledge protocol is a method by which one party (the prover) can prove to another party (the verifier) that something is true, without revealing any information apart from the fact that this specific statement is true."**

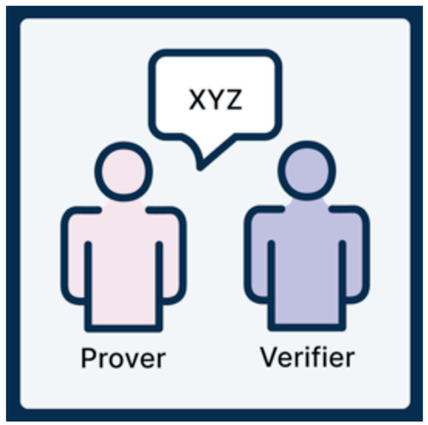


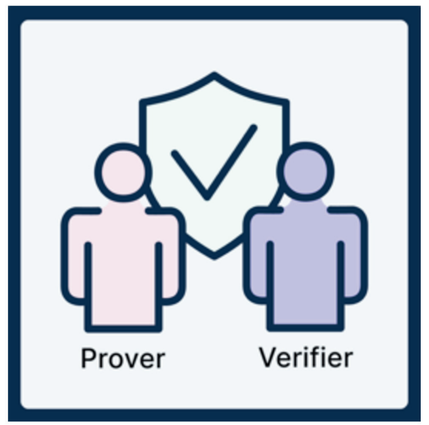

1. **Verifier** writes a secret message and put it in a locked safe




2. **Prover**, who fulfils the requirements, has knowledge of the combination code and opens the locked safe




3. Prover returns the secret message to Verifier




4. Verifier is convinced that the prover really knows the combination code and can therefore be trusted

# What is Zero Knowledge Proof (ZKP)

## What must ZKP satisfy?

1. C**ompleteness:** A true statement convinces an honest verifier.

If prices match, both parties learn this correctly.

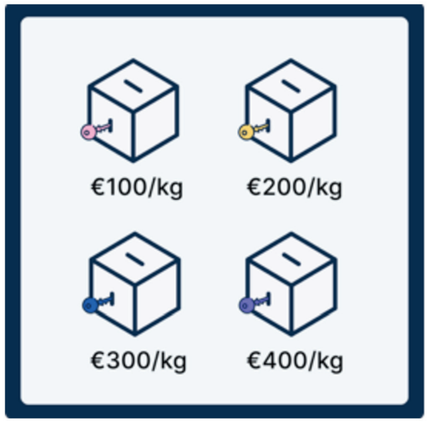
2. **Soundness:** A false statement can't fool an honest verifier.

If prices are different, no one can lie about it.

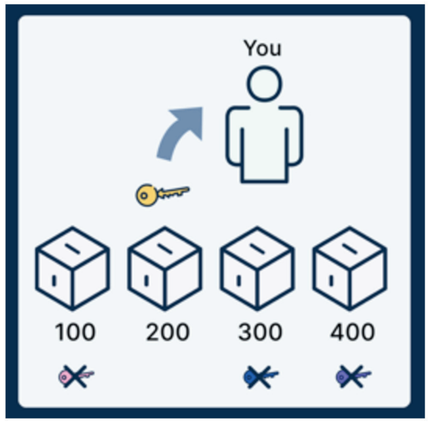
3. **Zero-Knowledge:** The verifier learns nothing beyond the truth of the statement.

They only find out if prices match, nothing more.

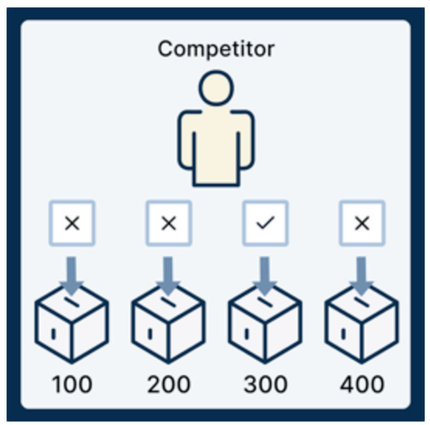

1. You and a competitor want to compare prices without revealing them.



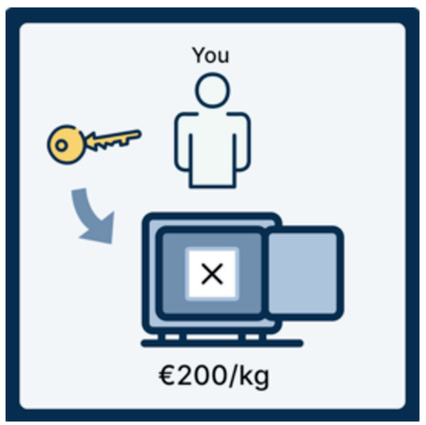

2. Four lockboxes (100, 200, 300, 400) are set up in a private room



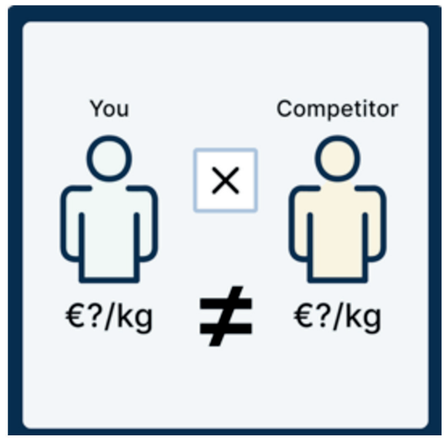

3. You enter first, keep the key for your price (200), and destroy the others.




4. Your competitor enters, marking ✅ in their price box (300) and placing ❌ in the rest




5. You come back, open box 200, and see a ❌ this means the prices are different.




6. Your competitor sees the ❌ in your box and also knows the prices differ

# What is Zero Knowledge Proof (ZKP)

## ZKP Architecture

Prover/Verifier

Statement

Circuit of Protocol

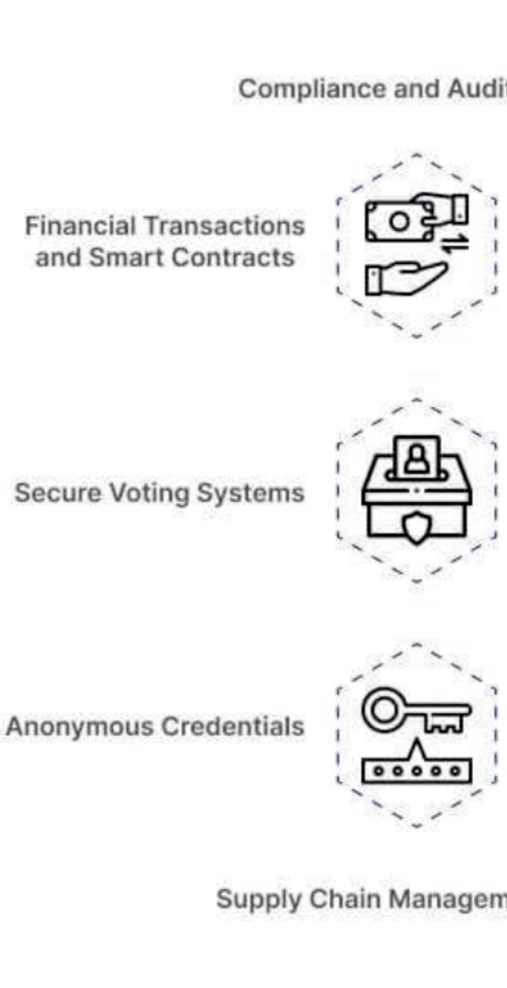
Common Reference String (CRS)

Proof Gen/Verify

Security

Scalability & Effeciency

Integration & Adoption

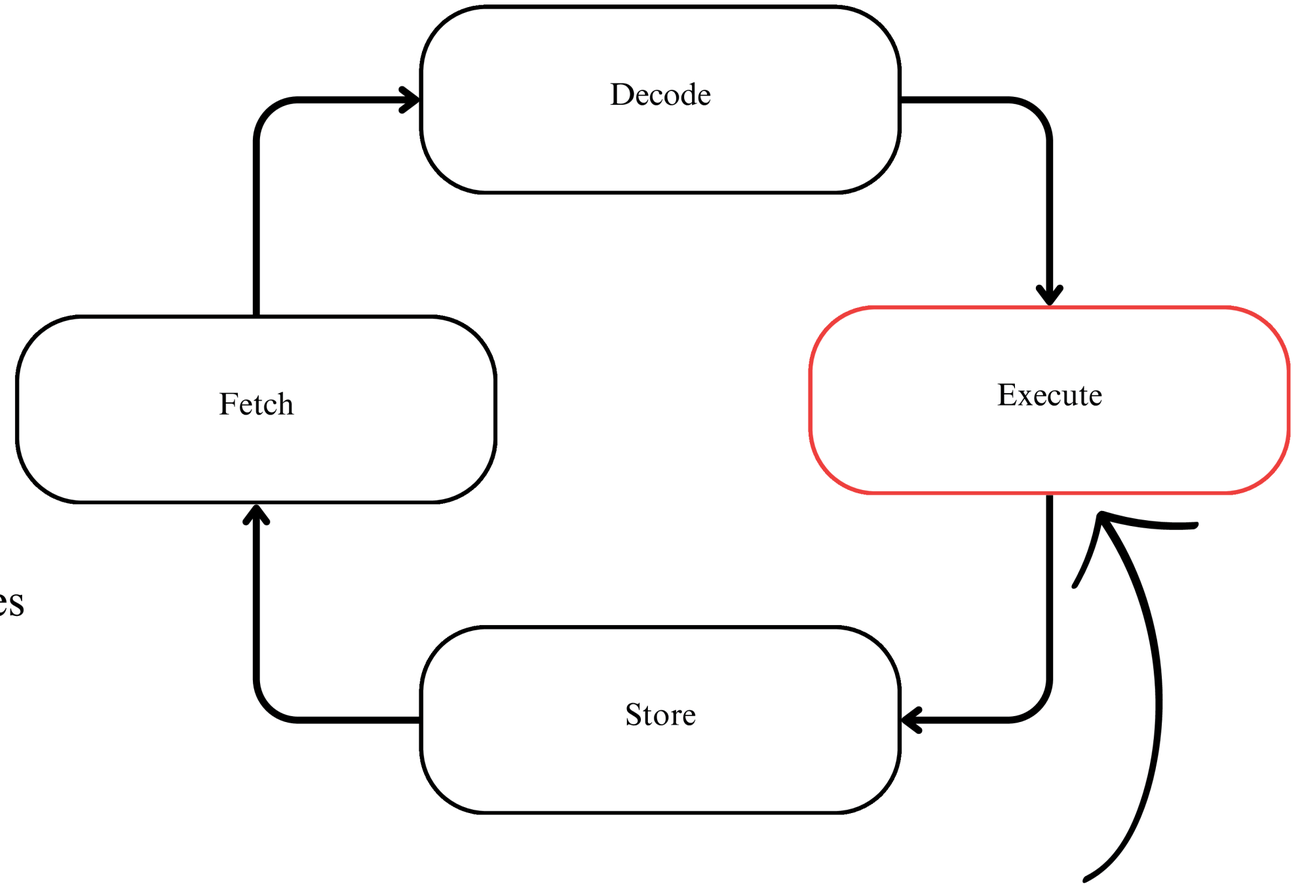


# Lookup arguments: what are they?

In a standard Virtual Machine (VM), the CPU executes each instruction through the following steps:

1. **Fetch**: The CPU retrieves an instruction from memory (RAM or cache).
2. **Decode**: The CPU identifies the opcode (the operation type) and operands (data involved in the operation).
3. **Execute**: The CPU processes the operation based on the opcode and operands.
4. **Writeback**: The CPU stores the result in registers or writes it back to memory.

* The (3) Execute step is typically the bottleneck, as the CPU which can be resource-intensive since it must perform actual computations




**In my proposed ZKP scheme, this step is replaced by using Lookup Argument**

[1] A. Arun, S. Setty, và J. Thaler, "Jolt: SNARKs for Virtual Machines via Lookups,"

## Lookup arguments: what are they?

- **Unindexed lookup argument:**
  - Lets **P** commit to a vector a $F^{\mathbf{m}}$, and prove that every entry of a resides in a pre-determined table **t** $F^{\mathbf{N}}$.
  - For every entry $a_i$ there is an index $b_i$ such that $a_i = t[b_i]$.

**Example 1:** A blockchain smart contract arithmetic operations:

**Lookup table t**

| Operation ID | Computation | Precomputed Result |
|---|---|---|
| 0x01 | ADD(10, 15) | 25 |
| 0x02 | MUL(20, 3) | 60 |
| 0x03 | EXP(5, 3) mod 23 | 10 |
| 0x04 | MODINV(7, 17) | 5 |

**Smart Contract Transactions (Vector a)**

| Transaction ID | Operation | Claimed Result |
|---|---|---|
| Tx1 | ADD(10, 15) | 25 |
| Tx2 | MUL(20, 3) | 60 |
| Tx3 | EXP(5, 3) mod 23 | 10 |

**Verifying**

| Transaction ($a_i$) | Index ($b_i$) in Lookup Table tt | Verification |
|---|---|---|
| ADD(10, 15) = 25 | t[0x01] | ✅ Valid |
| MUL(20, 3) = 60 | t[0x02] | ✅ Valid |
| EXP(5, 3) mod 23 = 10 | t[0x03] | ✅ Valid |
| MODINV(7, 17) = 6 | Tx4 | ❌ Rejected |

# Lookup arguments: what are they?

- **Indexed lookup argument:**
  - Lets **P** commit to vectors a, b  $F^{\wedge}\mathbf{m}$, and prove that $a_i = t[b_i]$ for all i.
  - We call a the vector of lookup values and b the indices.

**Example 2:** Ensuring correct account balance reads in a smart contract

**Lookup table t**

| Index $b_i$ (Account ID) | Stored Balance (Blockchain Memory $b_i$) |
|---|---|
| 0xA1 | 500 Bitcoin |
| 0xB2 | 200 Bitcoin |
| 0xC3 | 800 Bitcoin |
| 0xD4 | 350 Bitcoin |

**Smart Contract Transactions (Vector a)**

| Transaction ID | Queried Account (Index $b_i$) | Claimed Balance ($a_i$) |
|---|---|---|
| Tx1 | 0xA1 | 500 Bitcoin |
| Tx2 | 0xB2 | 200 Bitcoin |
| Tx3 | 0xC3 | 800 Bitcoin |

**Verifying**

| Queried Balance ($a_i$) | Index ($b_i$) (Account ID) | Blockchain Memory | Verified |
|---|---|---|---|
| 500 Bitcoin | 0xA1 | 500 Bitcoin | ✅ Valid |
| 200 Bitcoin | 0xB2 | 200 Bitcoin | ✅ Valid |
| 800 Bitcoin | 0xC3 | 800 Bitcoin | ✅ Valid |
| 400 Bitcoin | 0xD4 | 350 Bitcoin | ❌ Rejected |
| 350 Bitcoin | 0xD5 | none | ❌ Rejected |

## Lookup arguments: what are they?

- **Unindexed lookups** are proofs of a subset relationship (i.e., batch set-membership proofs).
    - a specifies a subset of t.

- **Indexed lookups** are reads into a read-only memory.
    - t is the memory, and $a_i = t[b_i]$ is a read of memory cell $b_i$.

| Aspect | Unindexed Lookup | Indexed Lookup |
|---|---|---|
| Checks if values exist in a set | ✅ Yes | ❌ No |
| Checks exact position in the set | ❌ No | ✅ Yes |
| Example 1: Valid transaction types | Ensures all transactions in a block are allowed. | ❌ Not applicable. |
| Example 2: Account balance verification | ❌ Not applicable. | Ensures each balance lookup matches the correct index. |

## Lasso+Jolt: what are they?

**Lasso: new family of (indexed) lookup arguments.**

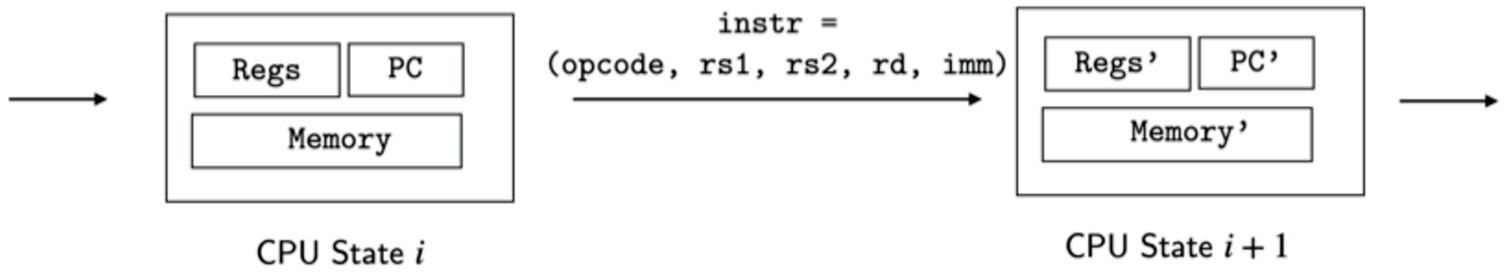
- **P** is an order of magnitude faster than in prior works.
  - Addresses key bottleneck for **P**: commitment costs.
- **P** commits to fewer field elements, and **all of them are small.**
- No commitment to **t** needed for many tables.
- Support for gigantic tables (decomposable, or LDE-structured).
  - **P** commitment costs: $O(c(m + N^{(1/c)}))$ field elements.

**Jolt: new zkVM technique.**

- Much lower commitment costs for **P** than prior works.
- Primitive instructions are implemented via one lookup into the entire evaluation table of the instruction.

## Jolt: A new front-end paradigm

- Say P claims to have run a computer program for ***m*** steps.
  - Say the program is written in the assembly language for a VM.
  - Popular VM's targeted: RISC-V, Ethereum Virtual Machine (EVM)
- Today, front-ends produce a circuit that, for each step of the computation:
  - 1. Figures out what instruction to execute at that step.
  - 2. Executes that instruction.
- Lasso lets one replace Step 2 with a single lookup.
  - For each instruction, the table stores the **entire evaluation table of the instruction.**
  - If instruction f operates on two 64-bit inputs, the table stores f(x, y) for every pair of 64-bit inputs (x, y).
    - This table has size $2^{128}$.
    - Jolt shows that all RISC-V instructions are decomposable.



# Jolt in a picture

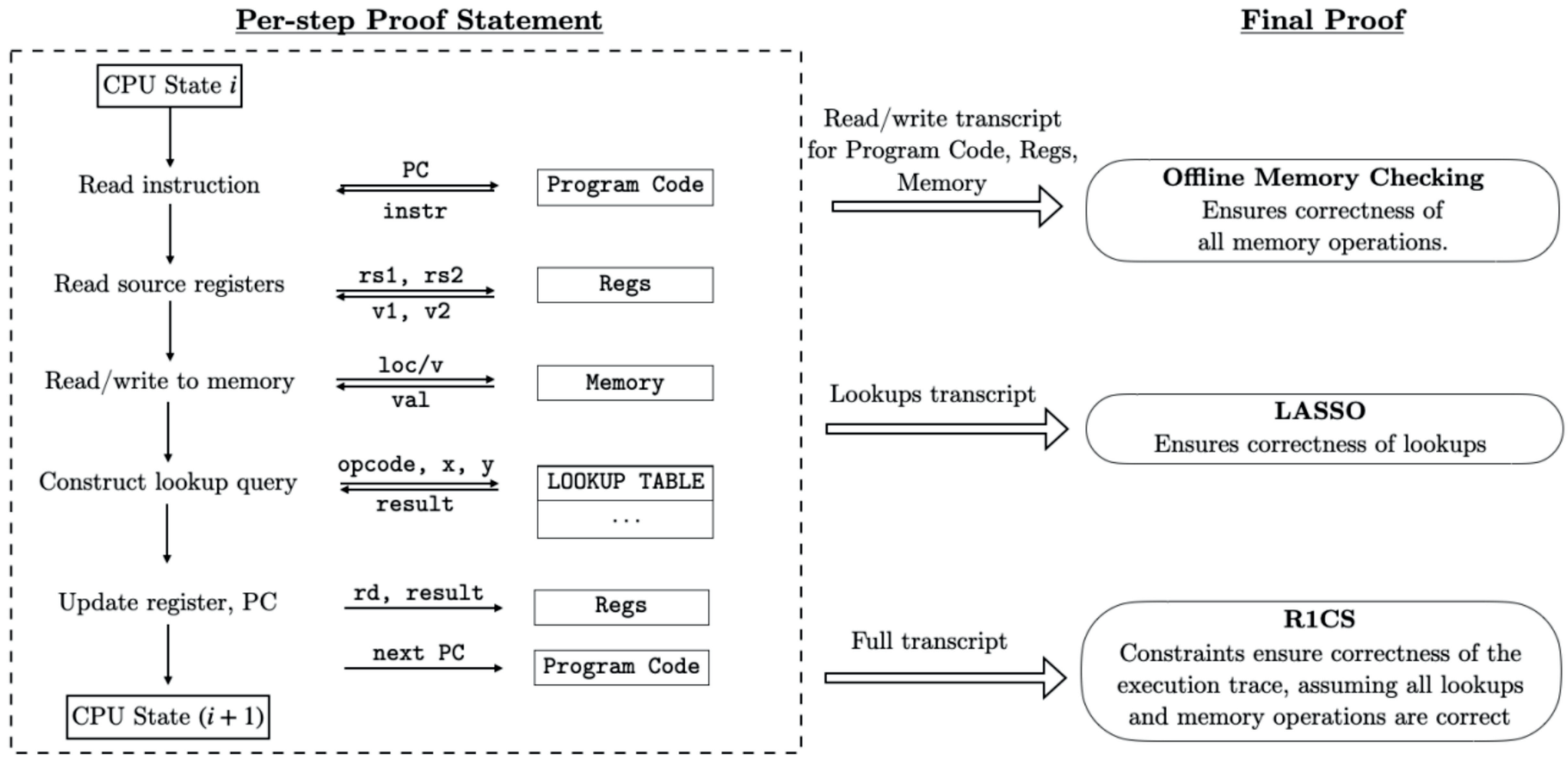


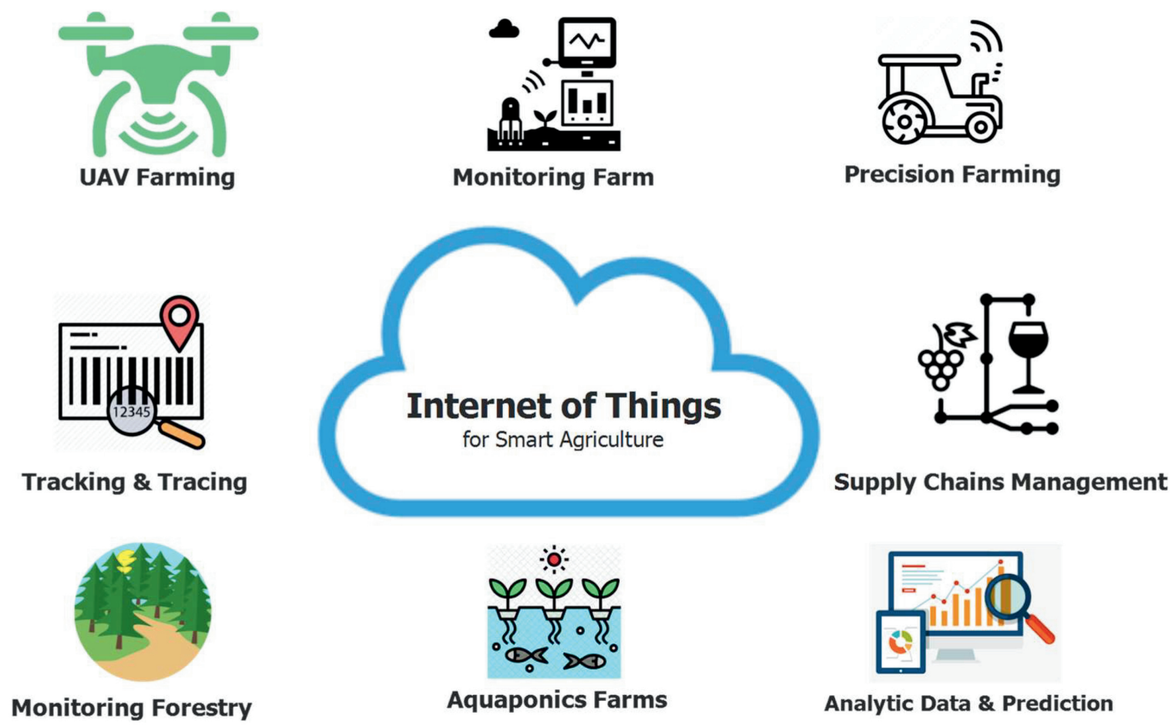

Figure 2: Proving the correctness of CPU execution using offline memory checking (Section 3.2) and lookups (Section 3.1).

[1] A. Arun, S. Setty, và J. Thaler, "Jolt: SNARKs for Virtual Machines via Lookups,"

# IoT and its problems

**Current situation of IoT devices:**

- **29 billion** IoT devices by 2030 (Ericsson).
- Key applications: Industrial IoT (IIoT), smart agriculture, healthcare, transportation, smart homes...
- Growth rate: IoT is expanding at 15-20% annually, surround by 5G, AI-driven automation, and cost-effective sensors.




**Characteristics of Lightweight IoT:**

- Limited computational power  Cannot run heavy cryptographic algorithms like RSA or ECC.
- Battery-powered or energy-harvesting  Requires ultra-low power consumption.
- Intermittent connectivity  Devices don't need to be online 24/7 but periodically transmit data (e.g., every few minutes or hours).
- Low data transmission rates  Optimized for efficiency, often not operating in high-bandwidth networks.

## IoT and its problems

**Background**, therefore:

- Lightweight IoT demands strong security, but devices face strict energy constraints.
- ZKP + Blockchain can enhance data security while preserving privacy

**Main Problem:**

- ZKP algorithms (even Jolt) are too computationally intensive for lightweight IoT devices.

**Research Objective:**

- Benchmark Jolt in lightweight IoT environment
- Develop Jolt-based ZKP model for IoT in blockchain network.

(either fine-tuning Jolt or create a new Jolt-based model)

- Compare ZKP schemes on IoT hardware energy use.

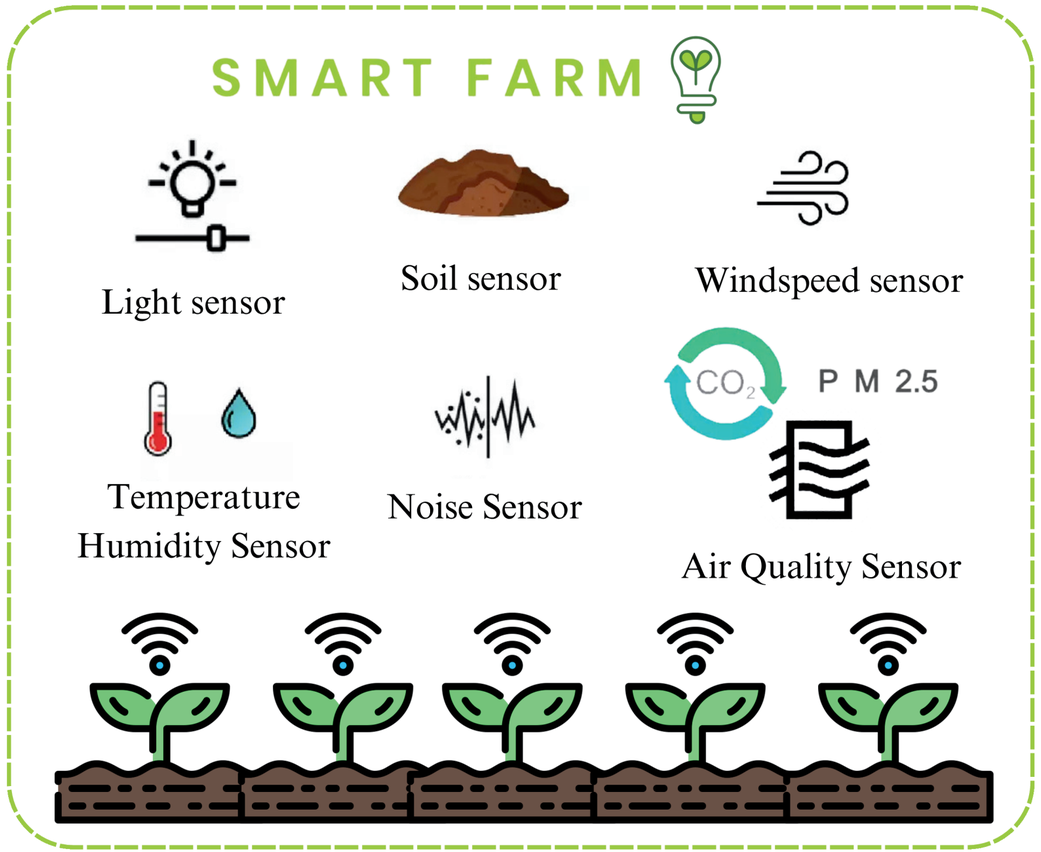
# Proposed Solution: Bad Case

**Pros:**

- Sensor only verify and send data  less power needed
- Cheap and Simple
- Offloading node computes everything

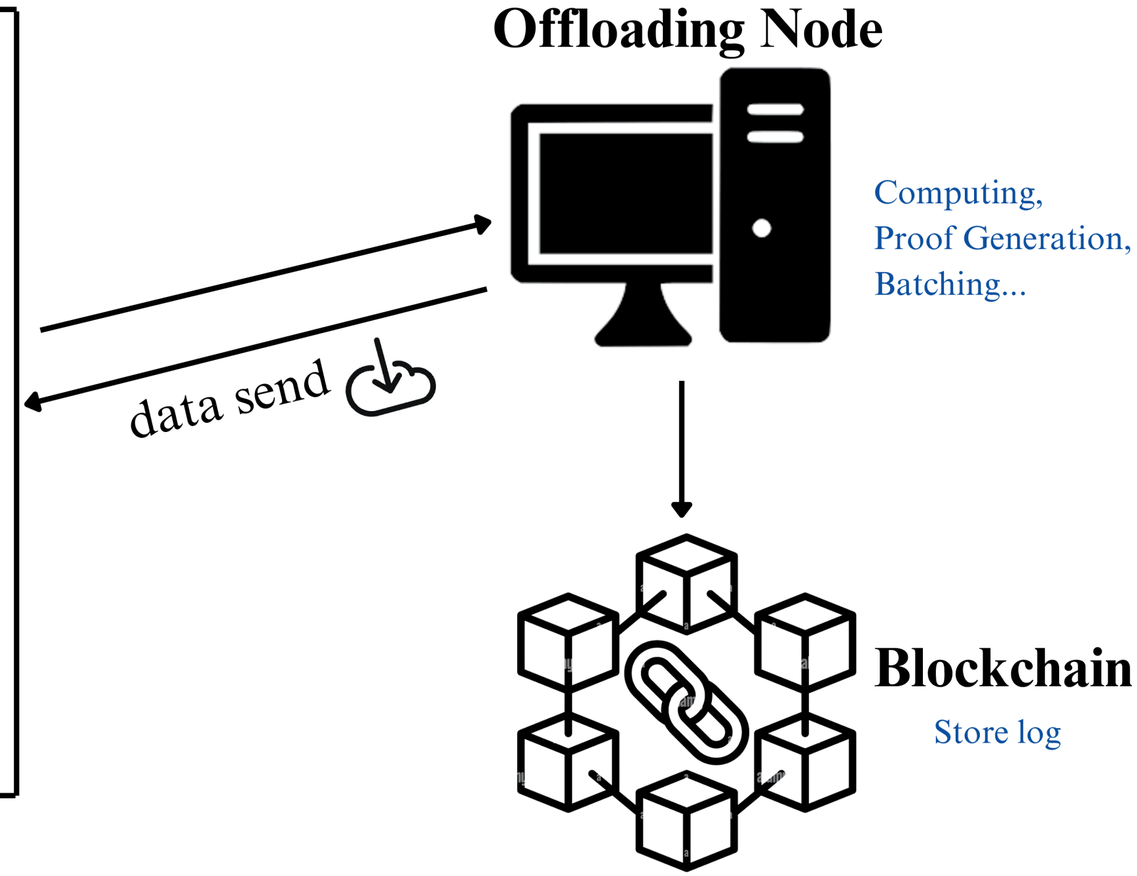
**Cons:**

- If the offloading node fail, the whole system breaks
- Big proof size, slow processing
- Data transmission expense is high




**IoT sensors**

Only act as Verifier and Sending data



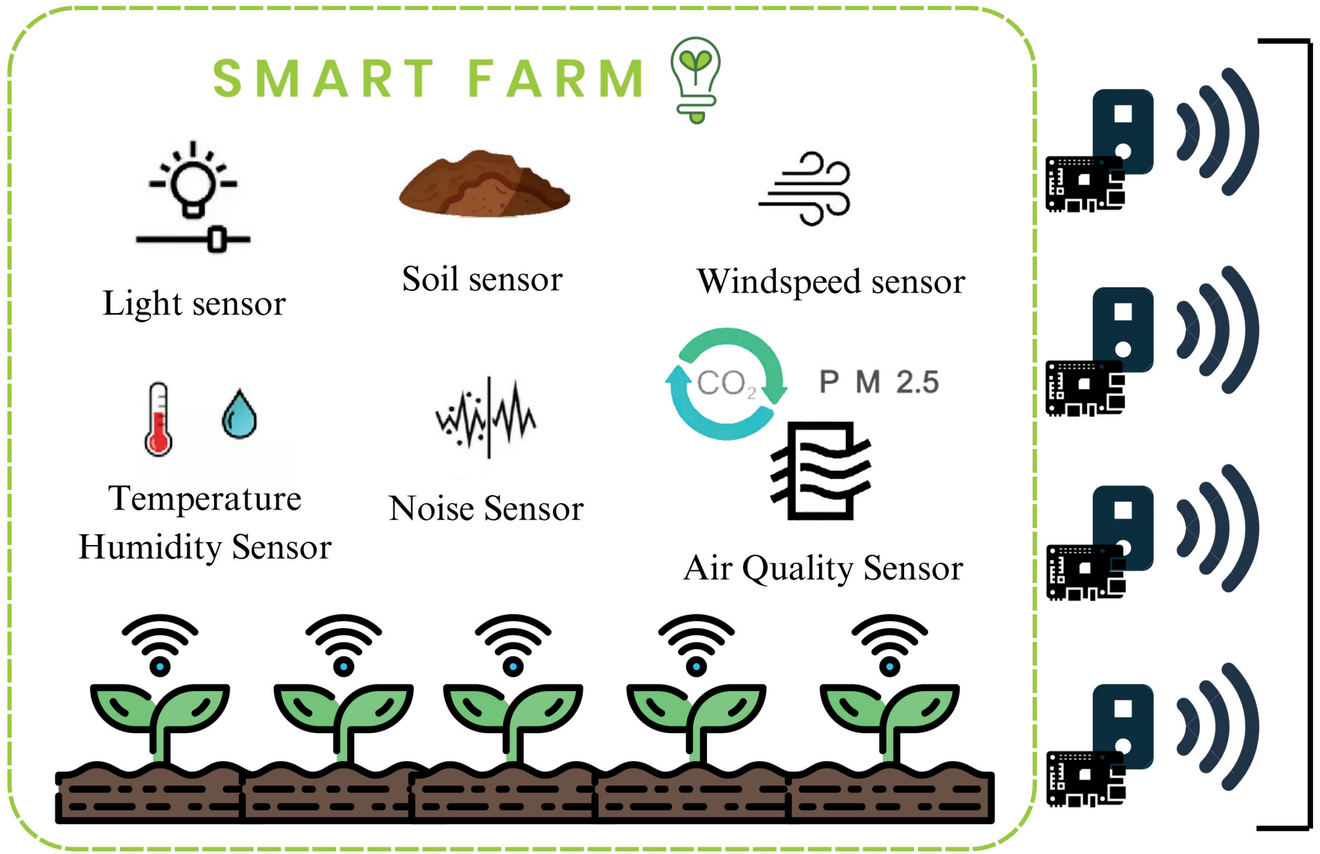
# Proposed Solution: Ideal Case

**Pros:**

- Proofs are generated on Raspberry Pi, reducing data transfer.
- Faster processing, no need to offload raw data
- Less reliance on expensive offloading nodes.

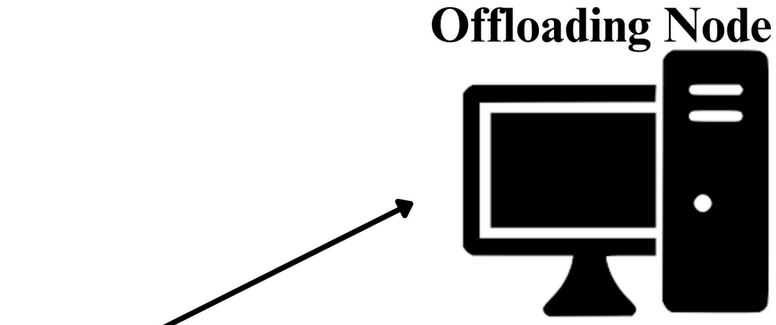
**Cons:**

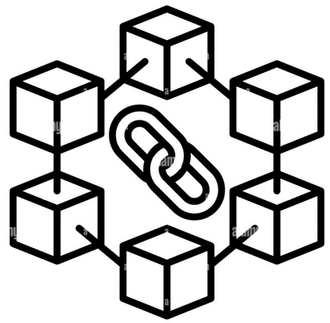
- Requires capable hardware and maintenance.




**RaspberryPi**

Send data & Compute Proof Generating



**Blockchain**

Store log

# Proposed Solution: Ideal Case

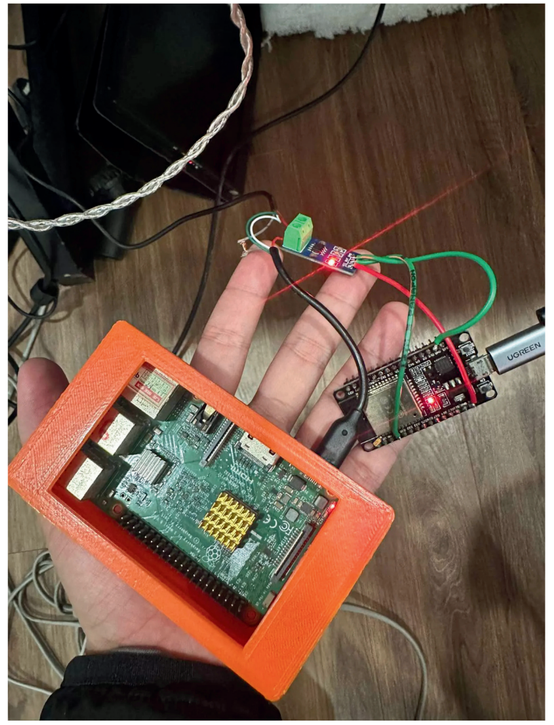
## Current Progress:

- Already tested using Raspberry Pi 3 Model B+ (ARMv7) for data collection, proof gen using Jolt:

```
   Compiling jolt-iot v0.1.0 (/home/ethan/projects/zk-iot/jolt-iot)
    Finished `release` profile [optimized + debuginfo] target(s) in 1m 02s
     Running `target/release/jolt-iot`
Trace length: 305
Environment healthy: true
Proof valid: true
Proof generation time: 1.8985547s
Proof size: 10304 bytes
```

- First, we compiled and successfully ran a simple Jolt circuit on an x86_64 machine.
- However, since it is impossible to compile Jolt directly on Raspbian OS (Raspberry Pi), we pre-built the project on an x86_64 machine (forcing the target for ARMv7) and then copied the binary to the RasPi 3.
- Unfortunately, the RasPi 3 could not run the program (too weak)

**Now need alternative plans.**

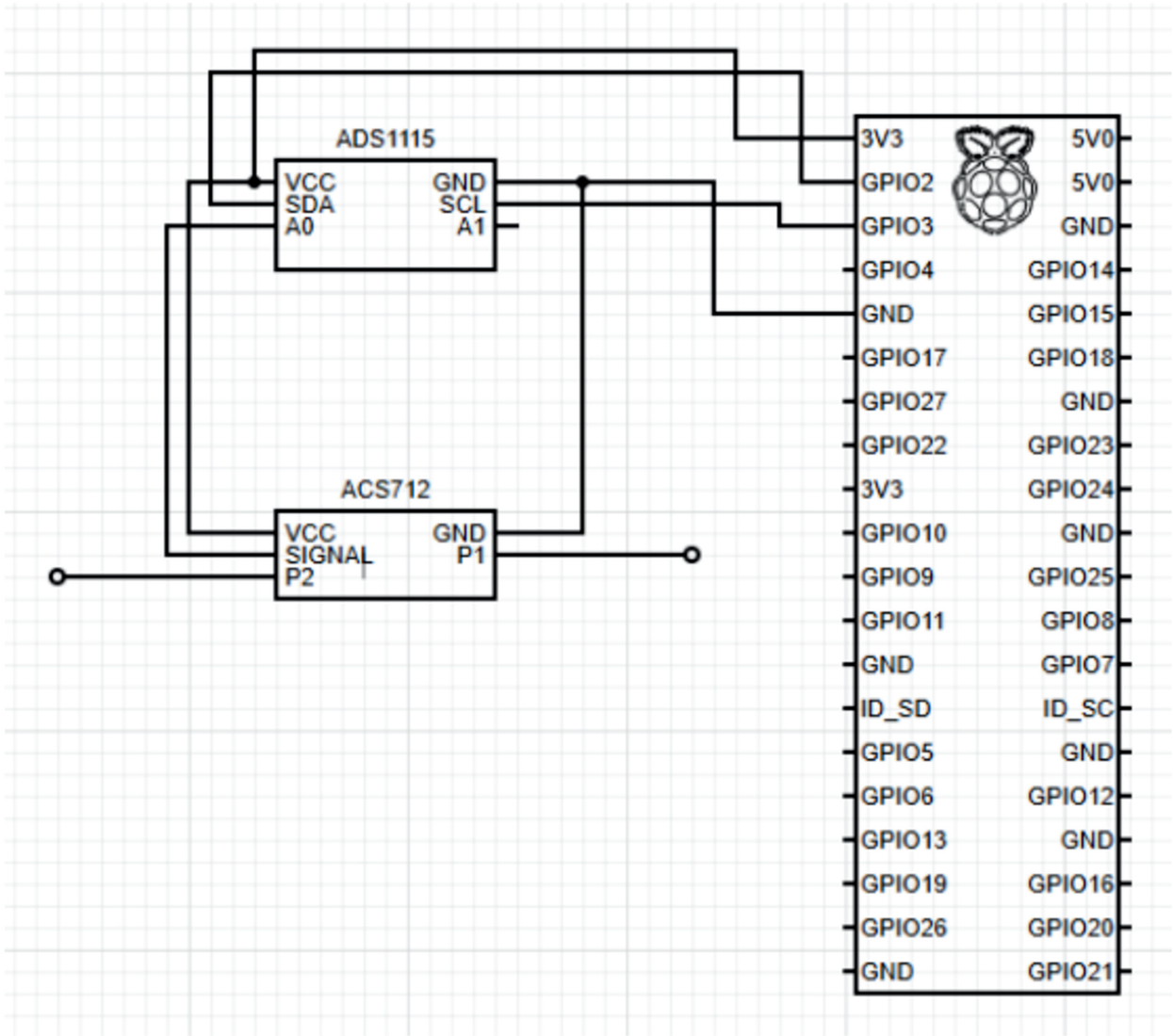


# Proposed Solution: Ideal Case

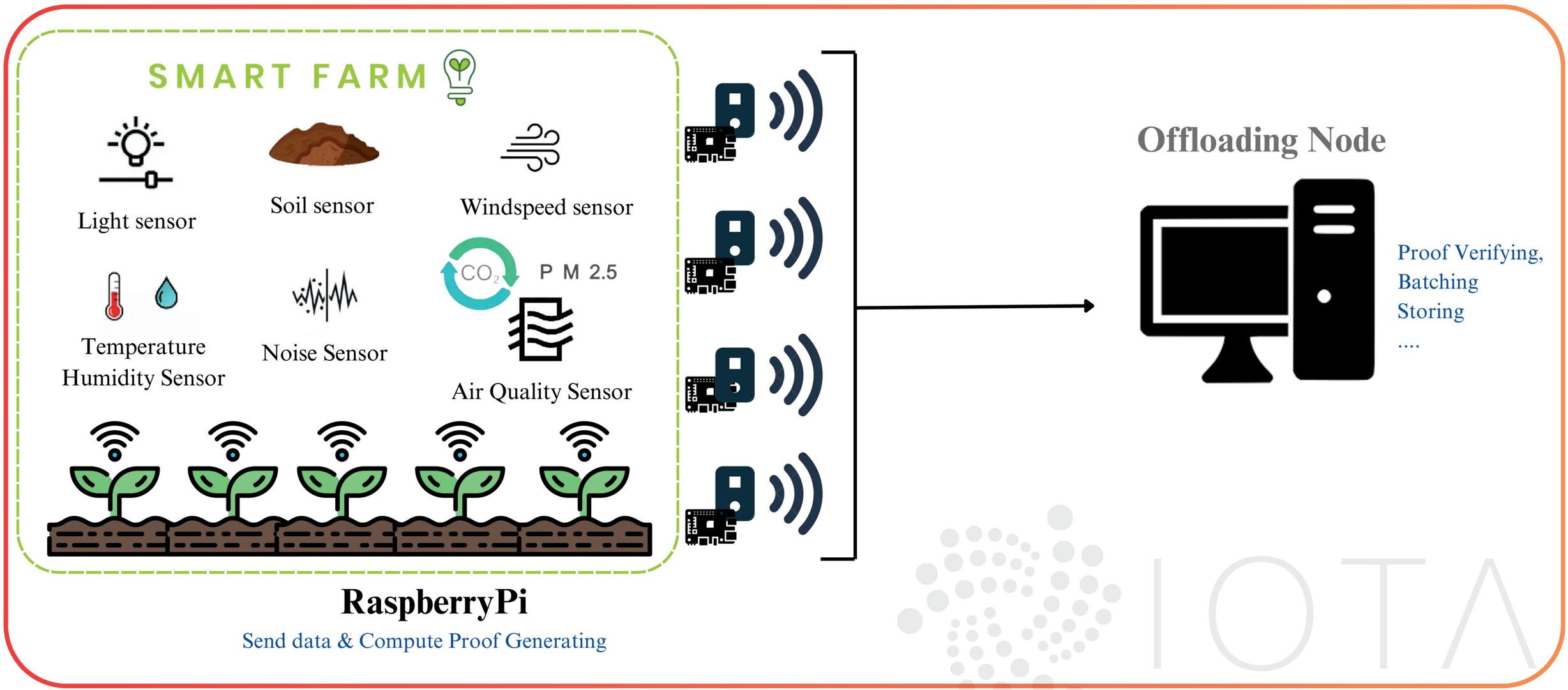
**Self-Monitoring Power Consumption on the RasPi:**

- We use the ACS712 module to evaluate power consumption from the machine's power source.

- However, since the ACS712 outputs an analog signal, which the RasPi does not support, we need to add an ADS1115 module to convert the analog signal to I2C communication for compatibility.



# Proposed Solution: Future Idea




**Local IOTA *Tangle Network* for IoT Transaction**

# THANK YOU.