zkStream

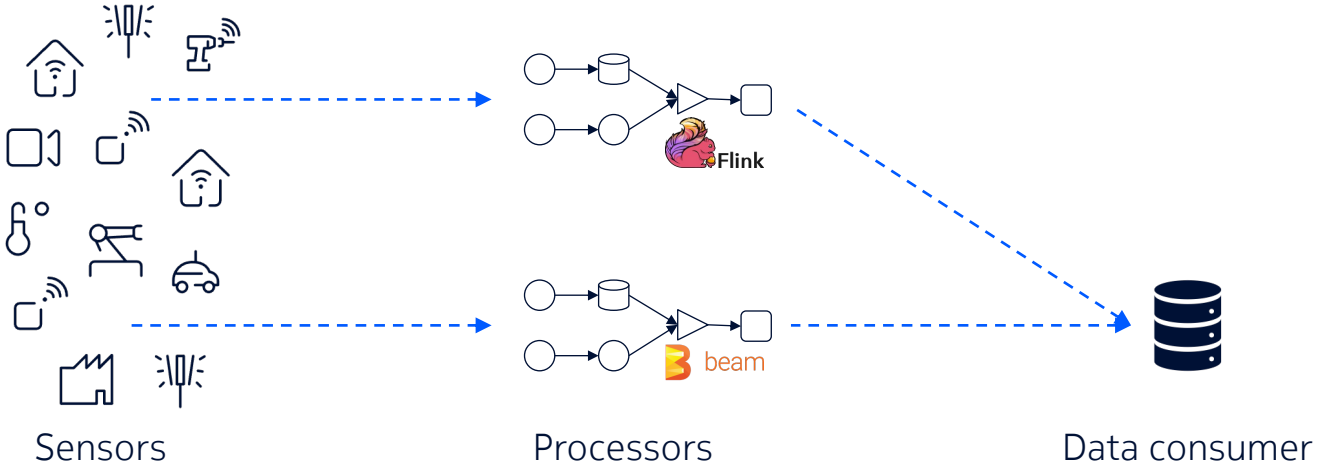
a Framework for Trustworthy Stream Processing

Middleware 2024

Janwillem Swalens, Lode Hoste,
Emad Heydari Beni, Lieven Trappeniers

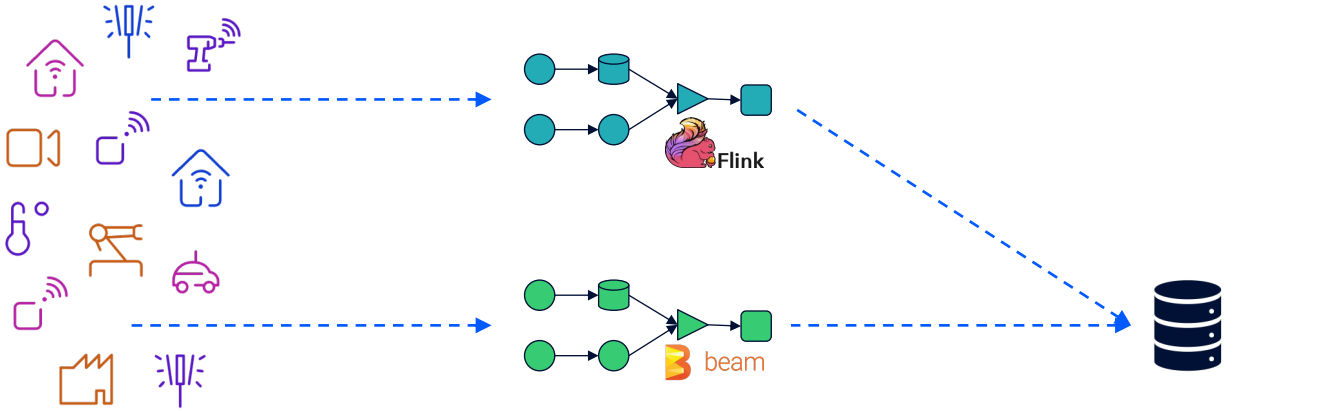
Distributed stream processing

leads to collaboration between many parties



Distributed stream processing

leads to collaboration between many parties



- Is each party computing honestly?
- Where is the data coming from? Where is it going?
- Can we prevent leaking confidential or privacy-sensitive data?

How can we automatically provide these trust guarantees?

NOKIA BELL LABS

We studied three use cases

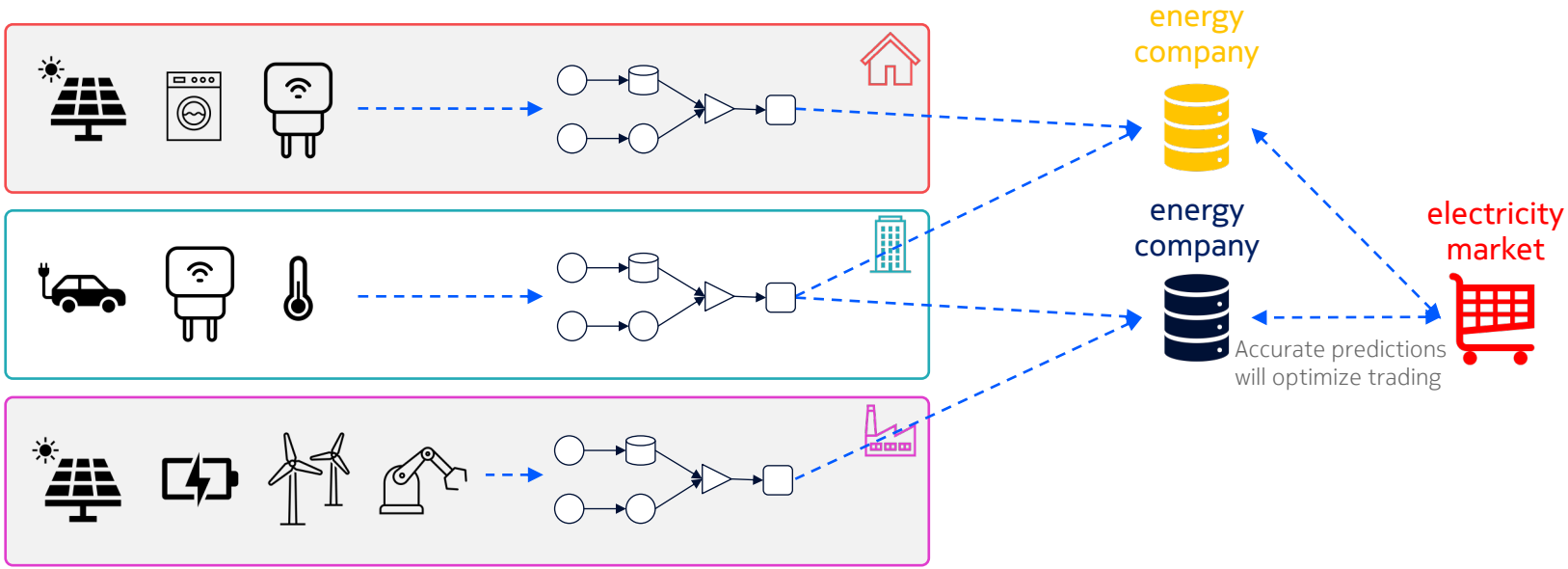
1. Electricity prediction (based on DEBS Challenge)
2. Energy orchestration (based on real use case)
3. NEXMark benchmark suite: 4 SQL-like queries for an online auctioning system

We studied three use cases

1. Electricity prediction (based on DEBS Challenge)
2. Energy orchestration (based on real use case)
3. NEXMark benchmark suite: 4 SQL-like queries for an online auctioning system

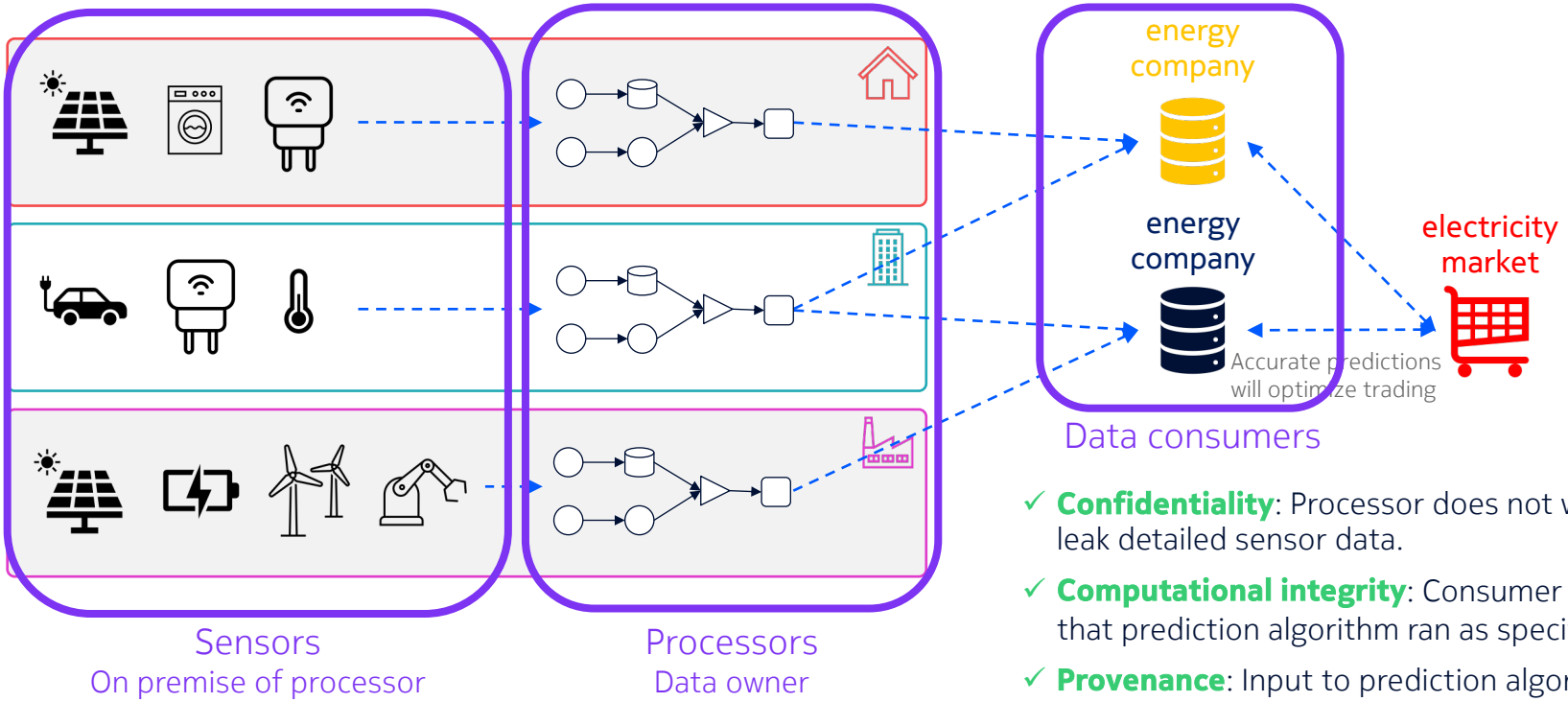
Example: electricity prediction

as a distributed stream processing system



Example: electricity prediction

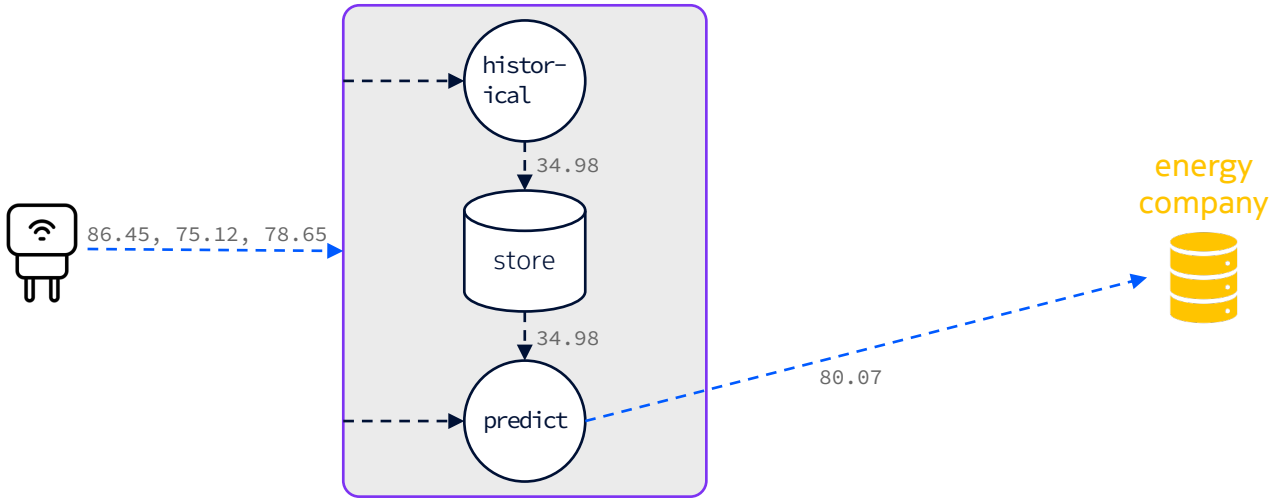
as a distributed stream processing system



- ✓ **Confidentiality**: Processor does not want to leak detailed sensor data.
- ✓ **Computational integrity**: Consumer knows that prediction algorithm ran as specified.
- ✓ **Provenance**: Input to prediction algorithm must come from trusted sensors.

Example: electricity prediction

Specification of DEBS 2014 Challenge

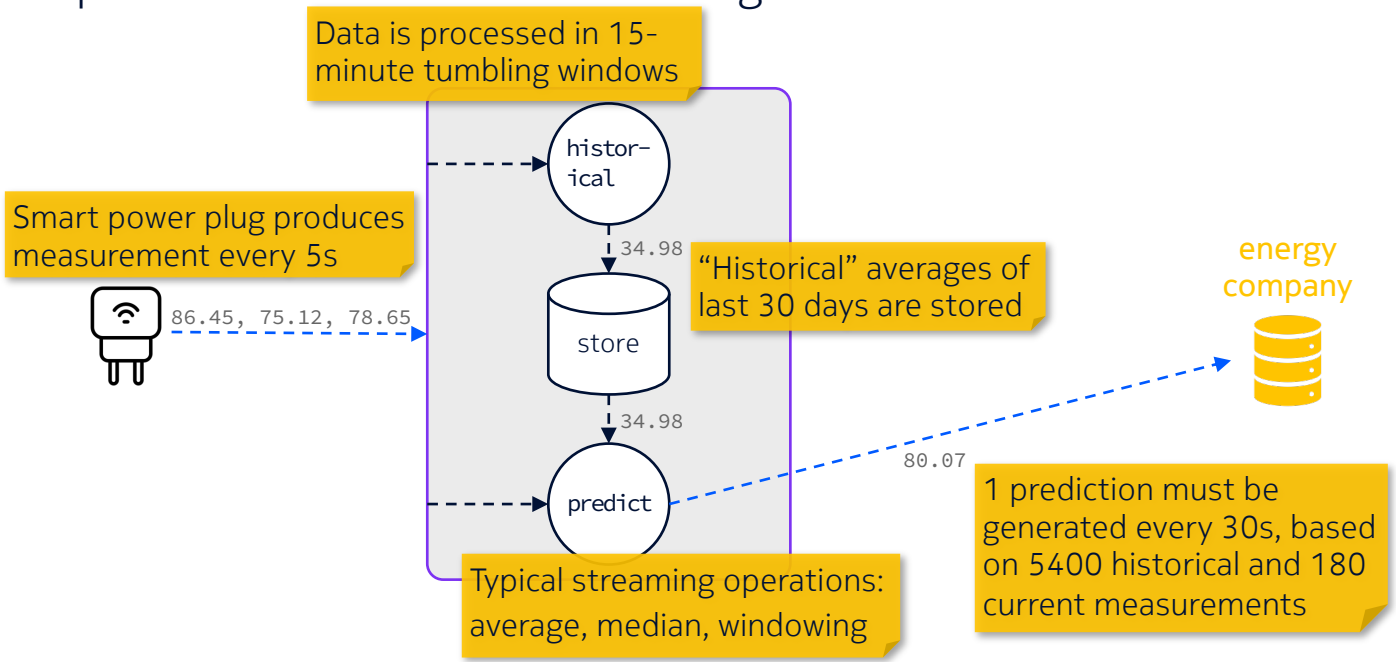


H. Ziekow and Z. Jerzak. “The DEBS 2014 Grand Challenge.” In *Proceedings of the 8th ACM International Conference on Distributed Event-based Systems, DEBS ’14.*

NOKIA BELL LABS

Example: electricity prediction

Specification of DEBS 2014 Challenge

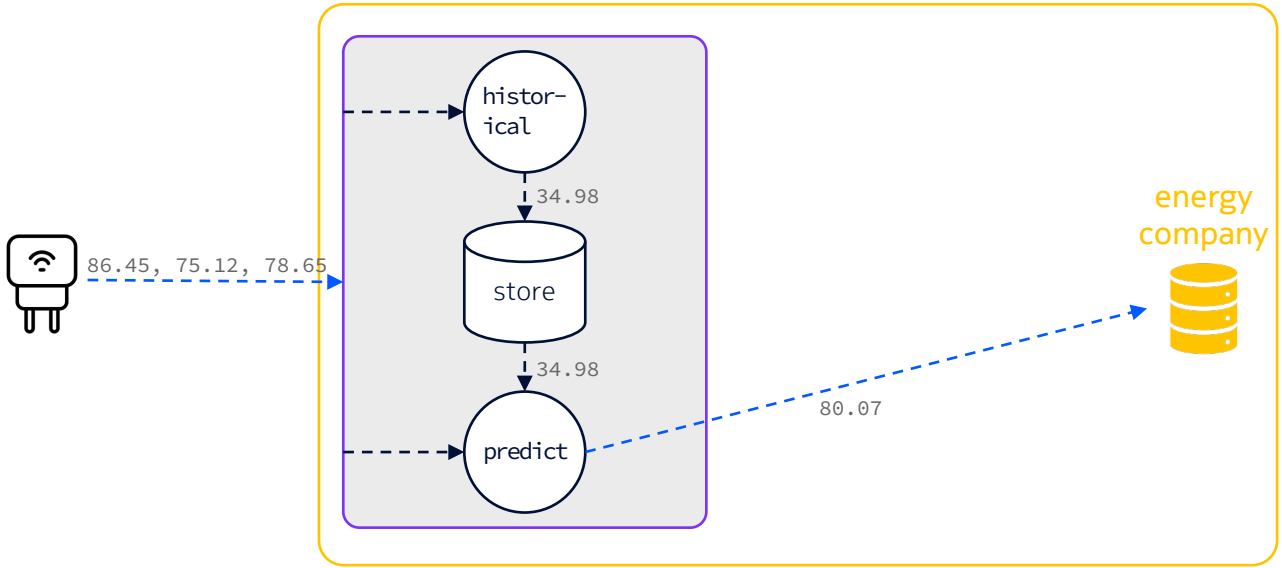


H. Ziekow and Z. Jerzak. "The DEBS 2014 Grand Challenge." In *Proceedings of the 8th ACM International Conference on Distributed Event-based Systems, DEBS '14.*

NOKIA BELL LABS

Typical streaming applications

process data in the cloud

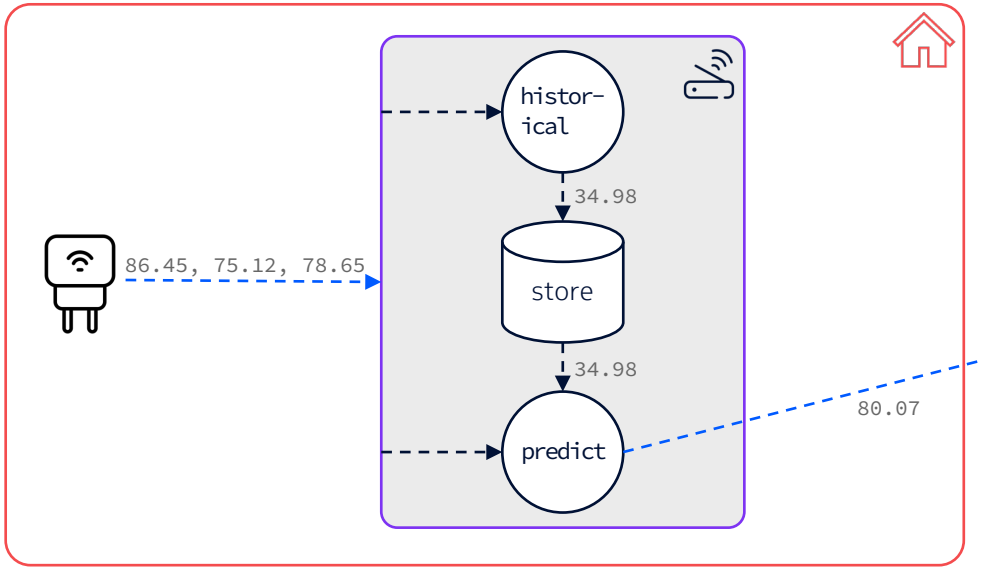


✗ All data is shared with energy company, but this breaks **confidentiality**.

H. Ziekow and Z. Jerzak. “The DEBS 2014 Grand Challenge.” In *Proceedings of the 8th ACM International Conference on Distributed Event-based Systems, DEBS ’14.*

NOKIA BELL LABS

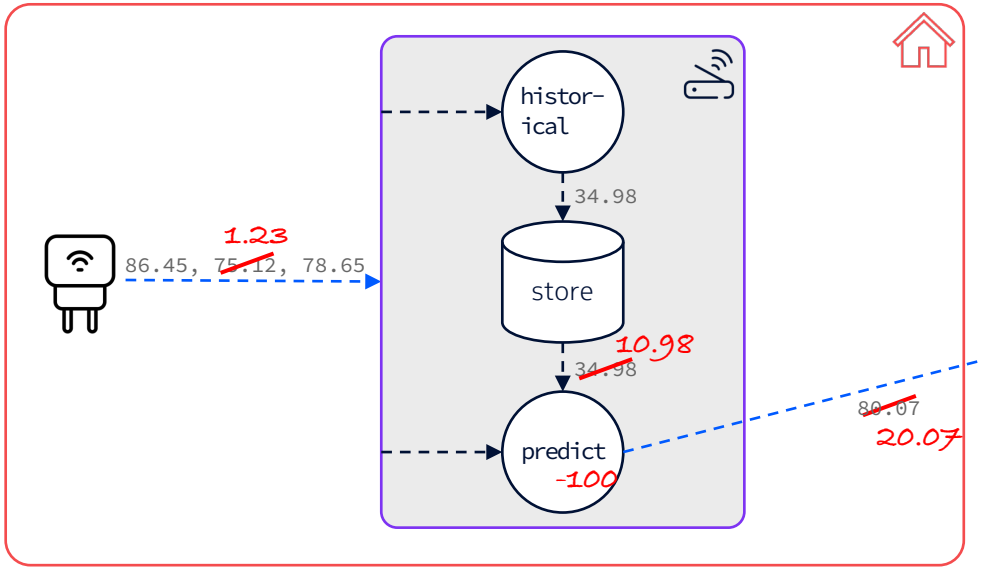
Alternative: process the data on the edge



✓ Only final prediction is shared:
confidentiality maintained.

Alternative: process the data on the edge

This is an **untrusted** setting!



✓ Only final prediction is shared: **confidentiality** maintained.

✗ Processor can tamper with inputs,

This breaks **computational integrity** and **provenance**.

NOKIA BELL LABS

zkStream: a framework for trustworthy stream processing

1. Signatures
2. Zero-Knowledge Proofs

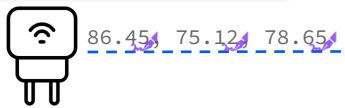
in a streaming architecture

NOKIA BELL LABS

zkStream: a framework for trustworthy stream processing

1. Signatures

We assume sensors contain a public–private key pair.



Sensor signs its output:

`σ = sign(hash(value), secret_key)`

To verify a sensor reading:

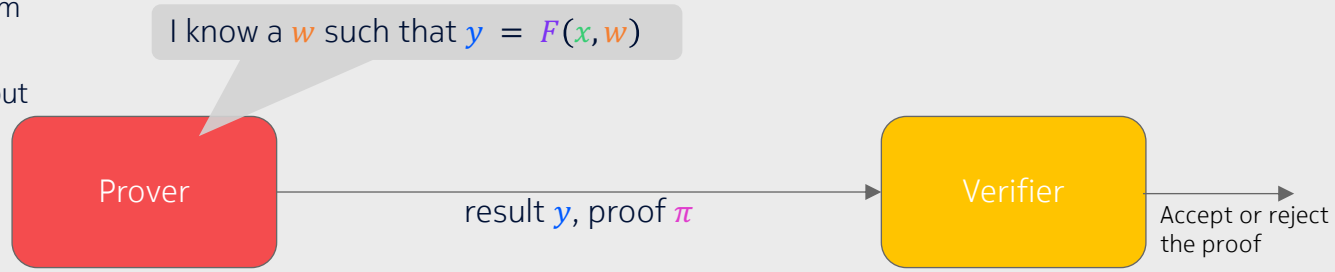
`verify(σ, hash(value), public_key)`

NOKIA BELL LABS

Zero-Knowledge Proofs to prove correct computation

A prover can convince a verifier that a statement is true,
without revealing anything besides the fact that the statement is true.

F = pre-agreed program
x = public input
w = private (secret) input
y = output



Goldwasser, Micali, and Rackoff (1985). "The knowledge complexity of interactive proof-systems." *Proceedings of the 17th Annual ACM Symposium on Theory of Computing.*

Bitansky, Canetti, Chiesa, Tromer (2012). "From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again." *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference.*

NOKIA BELL LABS

Zero-Knowledge Proofs to prove correct computation

Example: prove average

Program (ZoKrates)

```
const u32 N = 100;
def main(private u64[N] a) -> u64 {
  u64 mut sum = 0;
  for u32 i in 0..N {
    sum = sum + a[i];
  }
  return sum / cast(N);
}
```

compile

Set of constraints (R1CS)

```
((-1) * ~one + 1 * _1) * (1 * _4) == 1 * _3
(1 * ~one + (-1) * _3) * ((-1) * ~one + 1 * _1) == 0
(1 * ~one) * (1 * ~one) == 1 * _3
((-1) * ~one + 1 * _0) * (1 * _7) == 1 * _6
(1 * ~one + (-1) * _6) * ((-1) * ~one + 1 * _0) == 0
(1 * ~one) * (1 * ~one) == 1 * _6

…

(1 * ~one) * (1 * _6672) == 1 * ~out_0
```

6769 constraints

Witness

```
~out_0 = 50
~one = 1
_0 = 1
_1 = 0

…

_6864 = 1
_6865 = 0
```

generate prove

Proof


```
{"scheme": "g16",
 "curve": "bn128",
 "proof": {
   "a": ["0x1def914a6975111f9175dcfbcd536b64122c1f74ee8742ee894e03971240995f",
         "0x23e36195308359a848127ab17c9752ae934cea958b442049ee41bfcf060d8a63"],
   "b": [["0x0831e687bf60b170f76f8cfe8b4459c3712e1e51a266a0a5bcee4c5f0524dd46",
          "0x0caa6c13cf6535a6aa1cabbc7d848986df0602c67203bf7d95cc2e87e6cfa14d"],
         ["0x001d233eb8f4bb5307fd8ab77337f66f27bd9ef2057744d00d0bf6ad048348f9",
          "0x1aaada0ade26b6f5baeedbeaab0b711a2075c6853af8b49306b474c37fd0cdfd"]],
   "c": ["0x0ed1e3845fb7d175a60c3b58912727466b80d9c7859d26980c77ee81ab683c44",
         "0x2a619be66e620ba4985ae00a64ffcd0b796fb094391cf9cf58fd2ee1a55d322c"]},
 "inputs": ["0x0000000000000000000000000000000000000000000000000000000000000032"]}
```


verify ✓

- ✓ **Computational integrity**: proves that computation was performed correctly.
- ✓ **Confidentiality**: some inputs to the computation can be hidden from verifiers.

Eberhardt, Tai (2018). “Zokrates - Scalable Privacy-Preserving Off-Chain Computations.” In *28th IEEE International Symposium on Reliable Distributed Systems.*

Bitansky, Canetti, Chiesa, Tromer (2012). “From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again.” *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference.*

NOKIA BELL LABS

Zero-Knowledge Proofs to prove correct computation

Example: prove average

Program (ZoKrates)

```
const u32 N = 100;
def main(private u64[N] a) -> u64 {
  u64 mut sum = 0;
  for u32 i in 0..N {
    sum = sum + a[i];
  }
  return sum / cast(N);
}
```

compile

Set of constraints (R1CS)

```
((-1) * ~one + 1 * _1) * (1 * _4) == 1 * _3
(1 * ~one + (-1) * _3) * ((-1) * ~one + 1 * _1) == 0
(1 * ~one) * (1 * ~one) == 1 * _3
((-1) * ~one + 1 * _0) * (1 * _7) == 1 * _6
(1 * ~one + (-1) * _6) * ((-1) * ~one + 1 * _0) == 0
(1 * ~one) * (1 * ~one) == 1 * _6

…

(1 * ~one) * (1 * _6672) == 1 * ~out_0
```

6769 constraints

Witness

```
~out_0 = 50
~one = 1
_0 = 1
_1 = 0

…

_6864 = 1
_6865 = 0
```

generate prove

Proof


```
{"scheme": "g16",
 "curve": "bn128",
 "proof": {
   "a": ["0x1def914a6975111f9175dcfbcd536b64122c1f74ee8742ee894e03971240995f",
         "0x23e36195308359a848127ab17c9752ae934cea958b442049ee41bfcf060d8a63"],
   "b": [["0x0831e687bf60b170f76f8cfe8b4459c3712e1e51a266a0a5bcee4c5f0524dd46",
          "0x0caa6c13cf6535a6aa1cabbc7d848986df0602c67203bf7d95cc2e87e6cfa14d"],
         ["0x001d233eb8f4bb5307fd8ab77337f66f27bd9ef2057744d00d0bf6ad048348f9",
          "0x1aaada0ade26b6f5baeedbeaab0b711a2075c6853af8b49306b474c37fd0cdfd"]],
   "c": ["0x0ed1e3845fb7d175a60c3b5891272746 6b80d9c7859d26980c77ee81ab683c44",
         "0x2a619be66e620ba4985ae00a64ffcd0b796fb094391cf9cf58fd2ee1a55d322c"]},
 "inputs": ["0x0000000000000000000000000000000000000000000000000000000000000032"]}
```


verify ✓

- **Limited set of (efficient) operations**
  (e.g. floating points operations need to be emulated; both branches of `if` are evaluated)

E.g. average of 1000 numbers: ~1 sec
median of 1000 numbers: ~20 minutes (+ 200 GB RAM to generate proof)

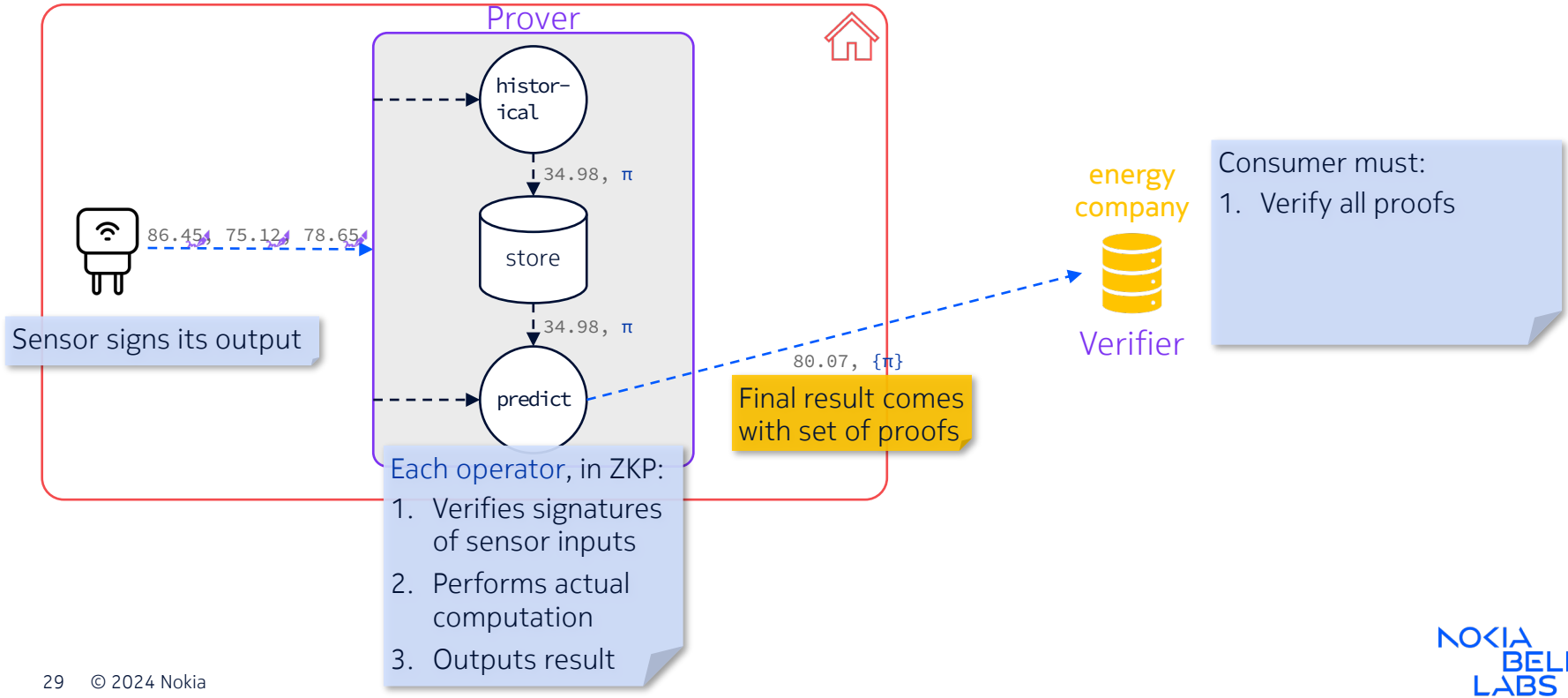
→ requires custom 'ZK-friendly' implementation

Eberhardt, Tai (2018). "Zokrates - Scalable Privacy-Preserving Off-Chain Computations." In *28th IEEE International Symposium on Reliable Distributed Systems*.

NOKIA BELL LABS

zkStream: a framework for trustworthy stream processing

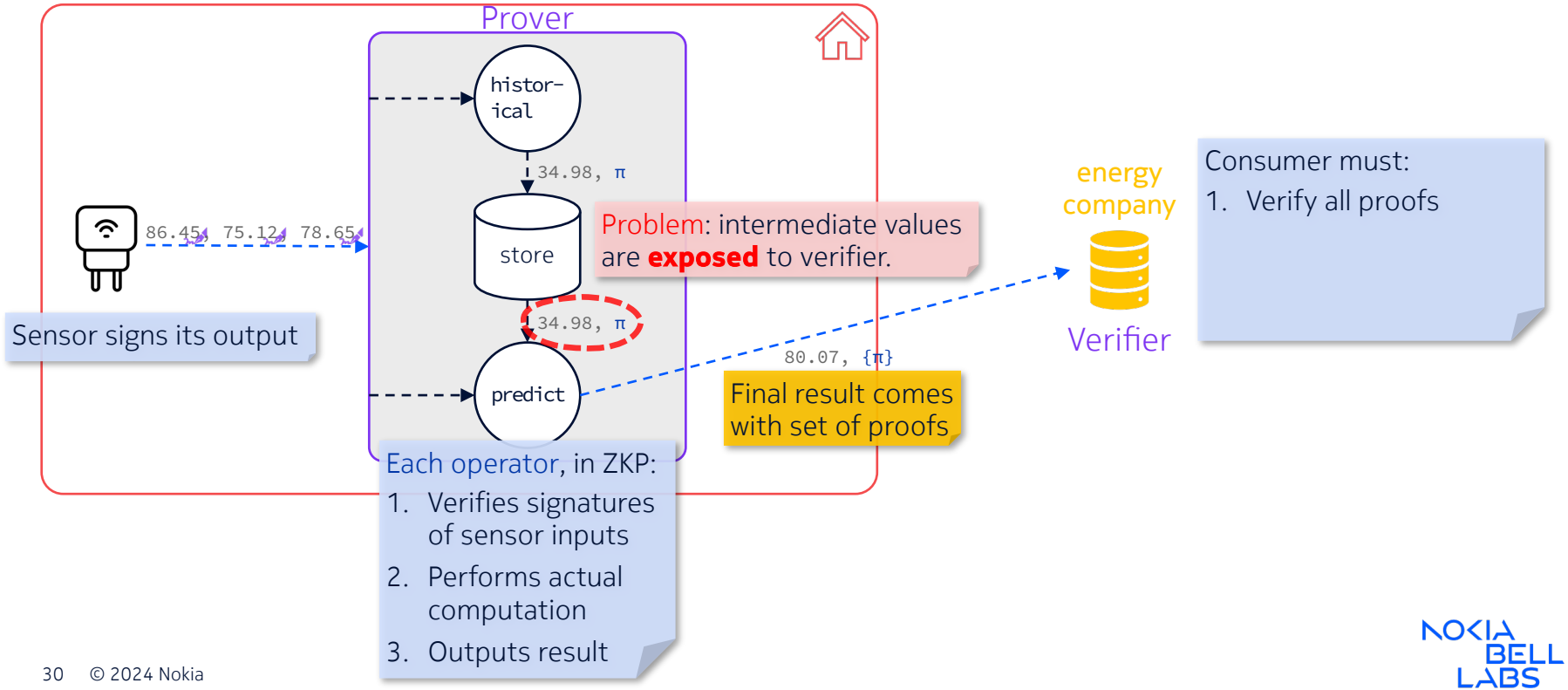
2. Zero-Knowledge Proofs



NOKIA BELL LABS

zkStream: a framework for trustworthy stream processing

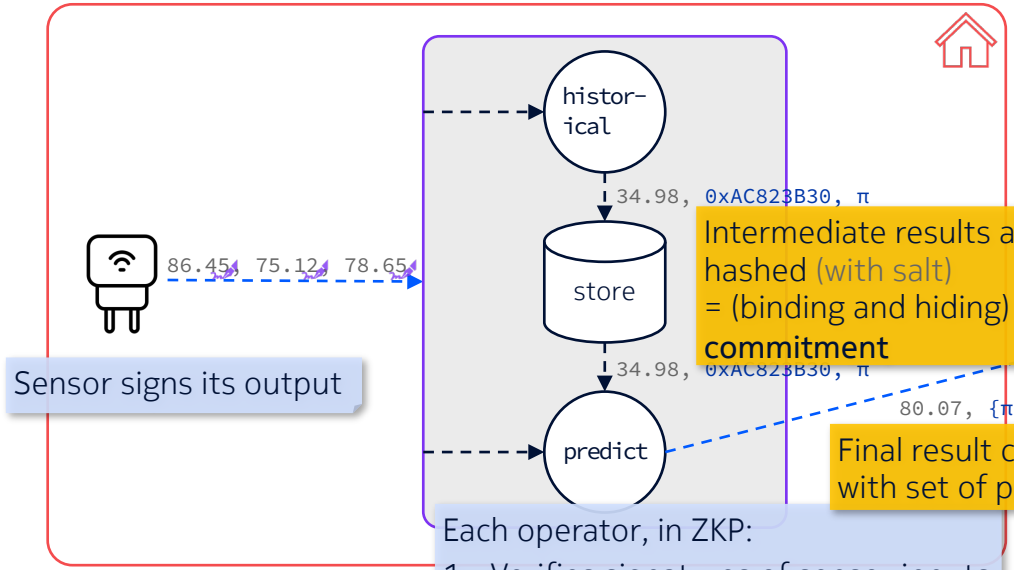
2. Zero-Knowledge Proofs



NOKIA BELL LABS

zkStream: a framework for trustworthy stream processing

3. Streaming architecture with "**confidential proof chaining**"



Consumer must:

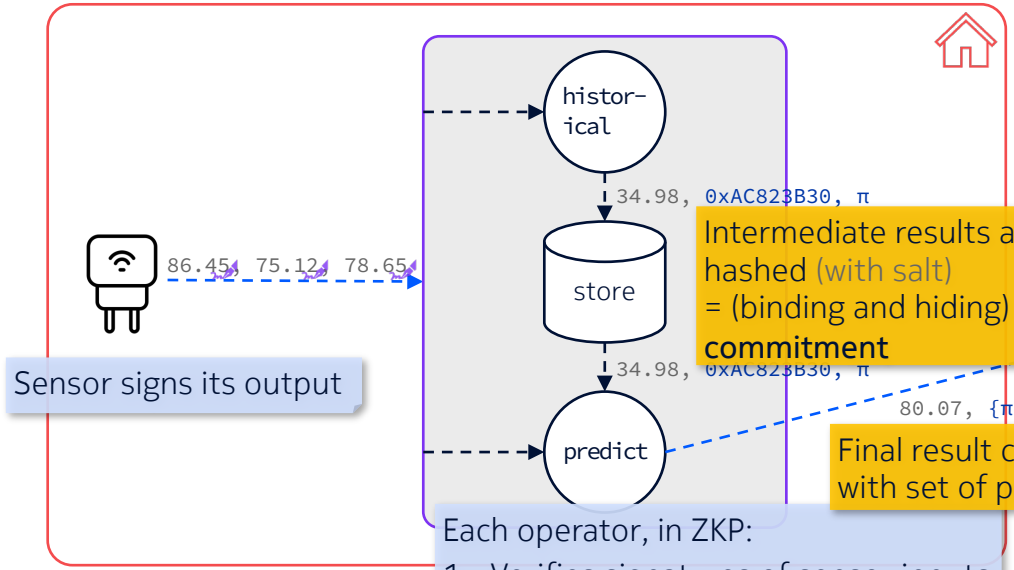
1. Verify all proofs
2. Verify commitments on intermediate values

- ✓ **Confidentiality**: Sensor inputs and intermediate values are kept private.
- ✓ **Computational integrity**: Proof attests that algorithm ran correctly.
- ✓ **Provenance**: Proof attests that signatures can be verified ⇒ came from a trusted sensor.

NOKIA BELL LABS

zkStream: a framework for trustworthy stream processing

3. Streaming architecture with “**confidential proof chaining**”



Consumer must:

1. Verify all proofs
2. Verify commitments on intermediate values

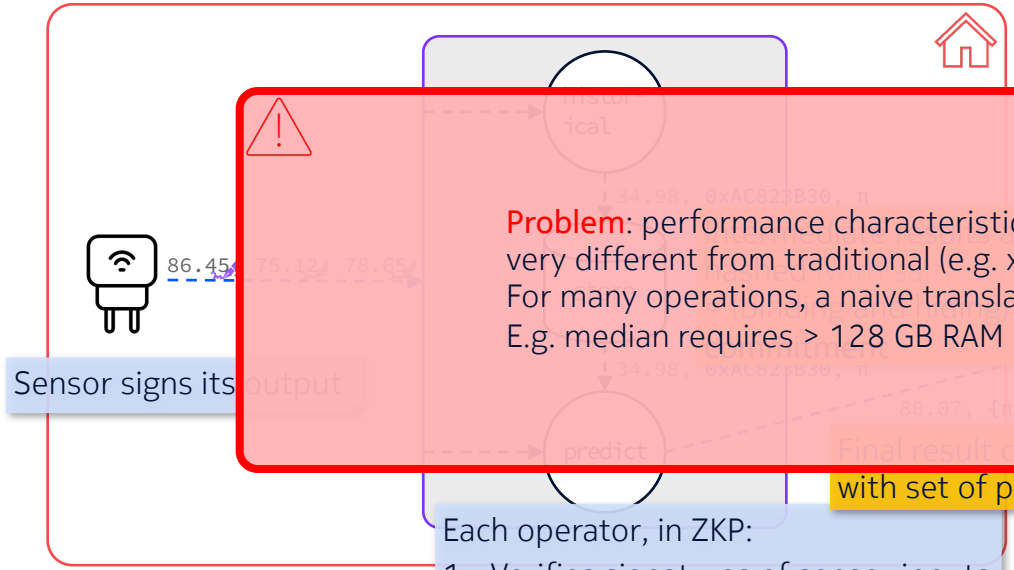
Enables typical stream processing features: **pipeline parallelism**, **caching**, **distribution**, and **stateful operators**.

⇒ For electricity prediction example, pre-compute and cache `historical`.

NOKIA BELL LABS

zkStream: a framework for trustworthy stream processing

3. Streaming architecture with “**confidential proof chaining**”



Problem: performance characteristics of ZKPs are very different from traditional (e.g. x86) architectures.
For many operations, a naive translation is **too expensive**.
E.g. median requires > 128 GB RAM

Consumer must:
1. Verify all proofs
2. Verify commitments on intermediate values

Enables typical stream processing features: **pipeline parallelism**, **caching**, **distribution**, and **stateful operators**.

⇒ For electricity prediction example, pre-compute and cache `historical`.

NOKIA BELL LABS

zkGadgets: ‘ZK-friendly’ aggregation operations

A toolkit for verifiable stream processing

A “gadget” is a small, specialized, and **efficient** proof **component** that can be **composed** and **re-used** as part of a larger proof. [Campanelli2019]

- ZK-friendly implementations of existing algorithms, e.g. matmul, SHA256 hash
- New ZK-friendly algorithms, e.g. Poseidon hash

We built a **toolkit of ZK-friendly aggregation operations for streaming**.

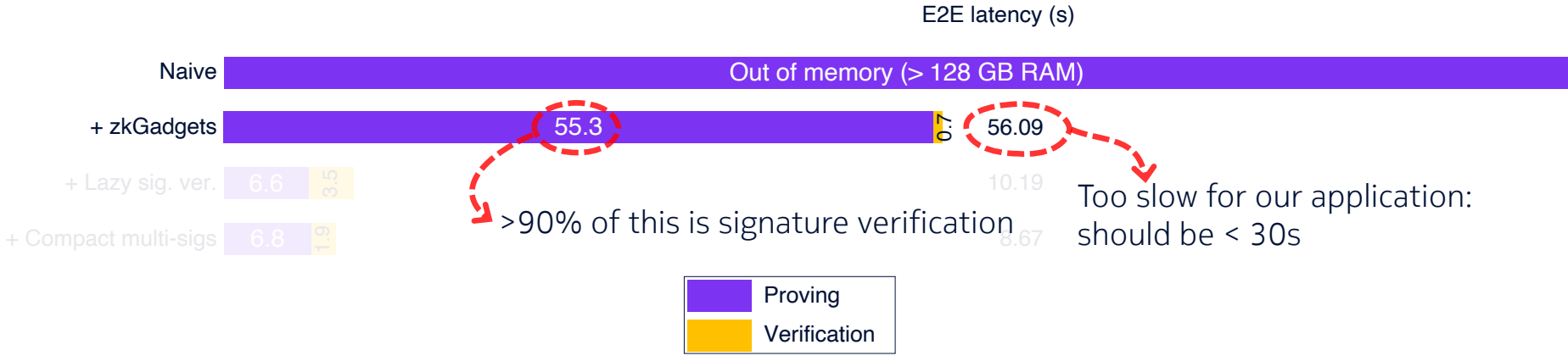
COUNT	MIN	TOP N	DENSE RANK	PERCENTILE	ANY
COUNT DISTINCT	AVERAGE	BOTTOM N	PERCENT RANK	FIRST	EVERY
COLLECT DISTINCT	VARIANCE	TOP DISTINCT N	CUME DIST	LAST	BITWISE AND
SUM	STD DEV	BOTTOM DISTINCT N	ROW NUMBER	LEAD	BITWISE OR
MAX	MEDIAN	RANK	NTILE	LAG	BITWISE XOR

https://github.com/Nokia-Bell-Labs/zkstream/tree/master/zkgadgets

Matteo Campanelli, Dario Fiore, and Anaïs Querol. 2019. *LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs*. In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (CCS ’19).

NOKIA BELL LABS

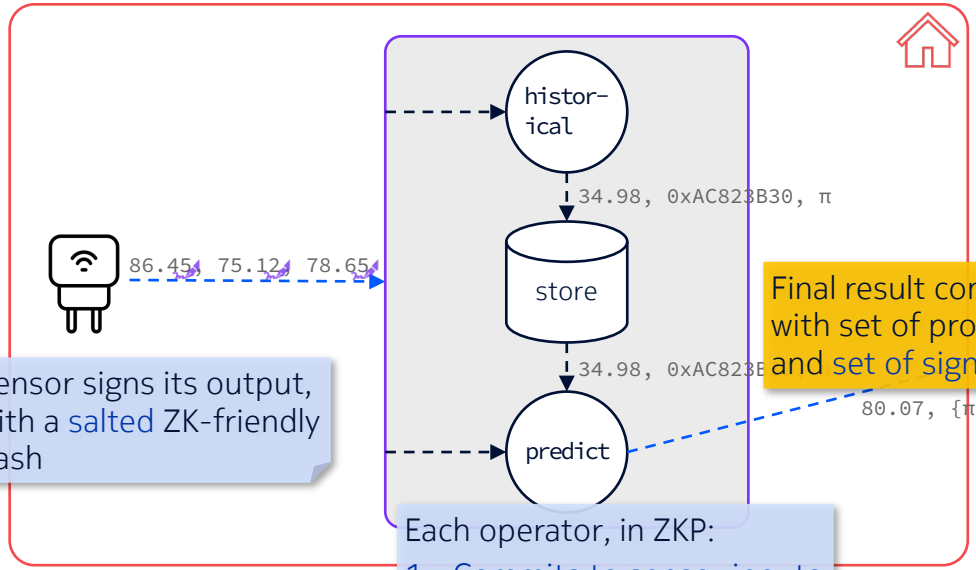
Performance analysis



NOKIA BELL LABS

Optimizing zkStream: 1. Lazy signature verification

Outsource signature verification to verifier



Sensor signs its output, with a salted ZK-friendly hash

Each operator, in ZKP:

1. Commits to sensor inputs
2. Commits to other inputs
3. Performs actual computation
4. Outputs result

Final result comes with set of proofs and set of signatures

energy company

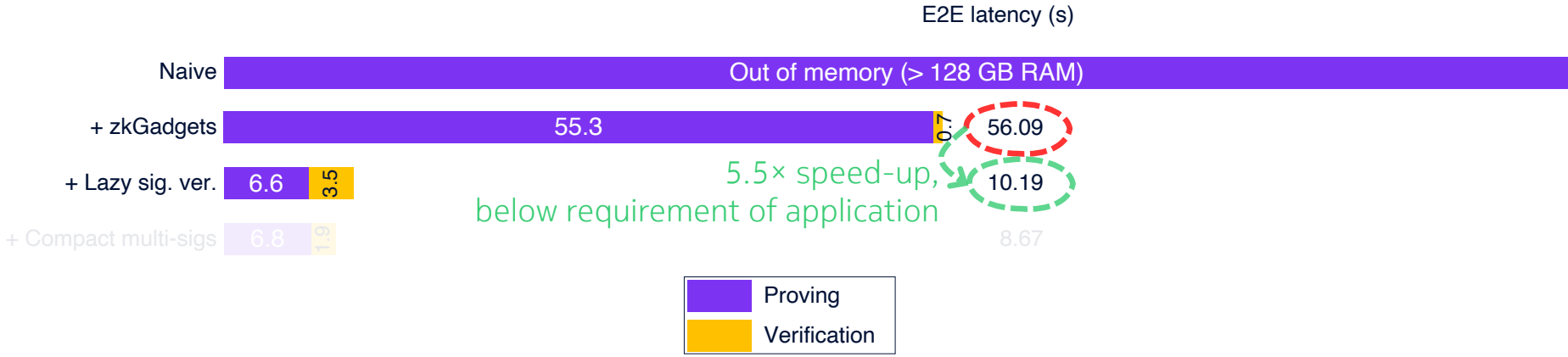
Consumer must:

1. Verify all proofs
2. Verify commitments on intermediate values
3. Verify signatures

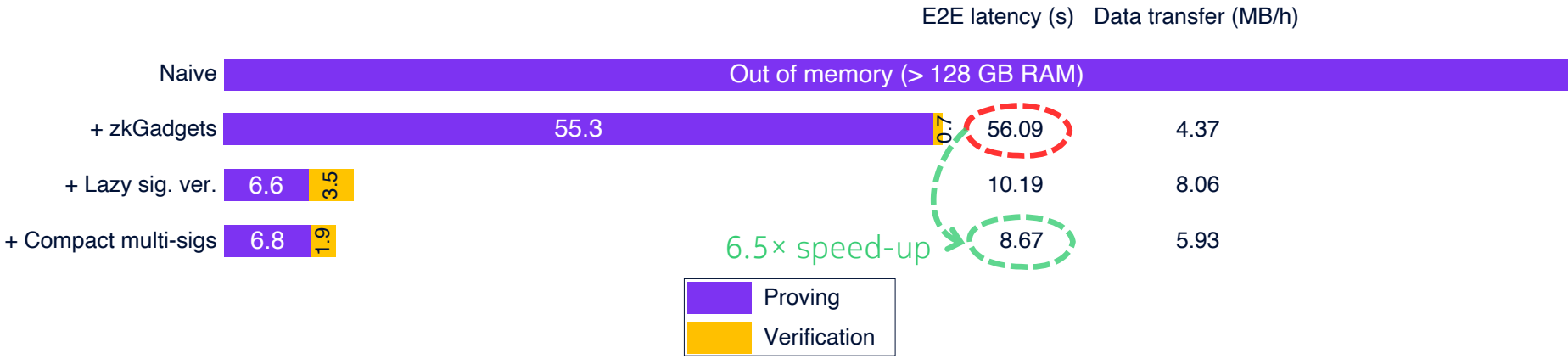
- ✓ **Confidentiality**: Input data not leaked due to hiding of salted hashes.
- ✓ **Computational integrity**: maintained.
- ✓ **Provenance**: maintained due to signatures + commitments.

NOKIA BELL LABS

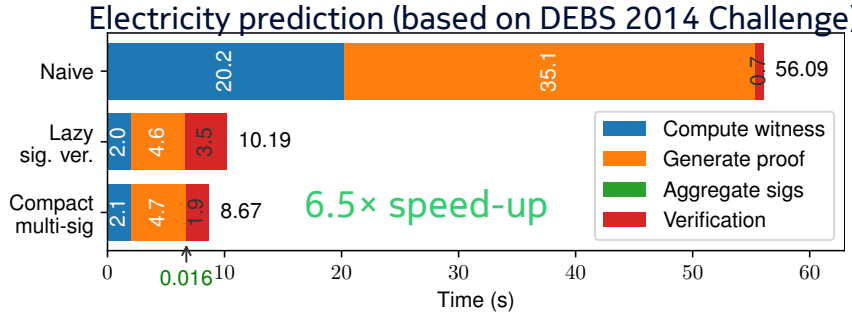
Optimizing zkStream: 1. Lazy signature verification

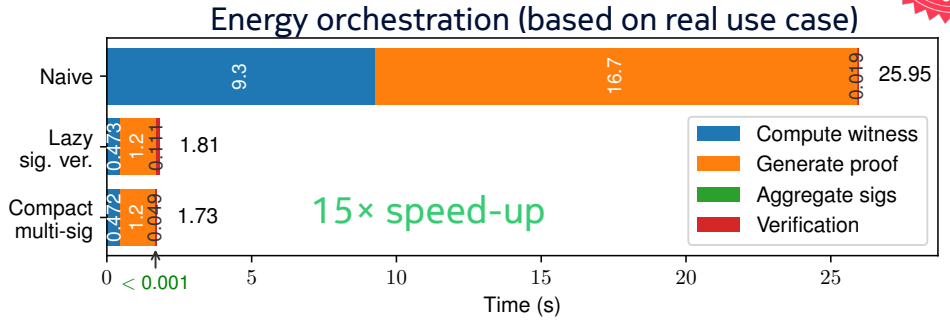


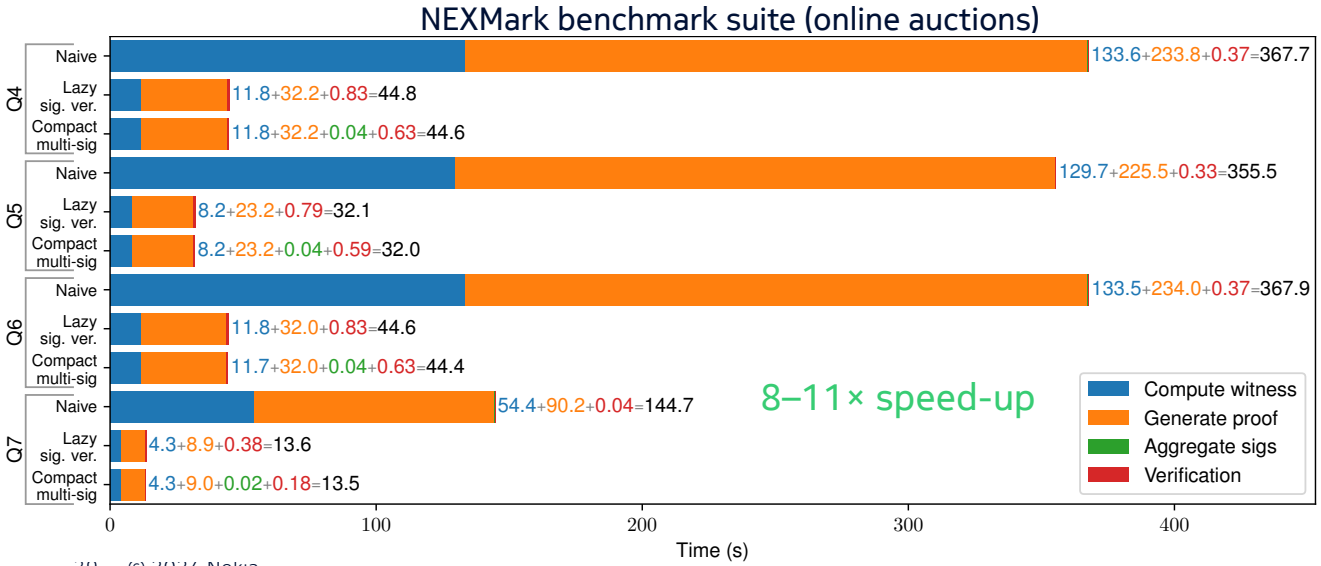
Optimizing zkStream: 2. Compact multi-signatures



Evaluation







Specs: 2× Intel Xeon Gold 5318Y CPU (2× 24 cores/48 threads, 2.10 GHz), 256 GB RAM. ZoKrates 0.8.7.

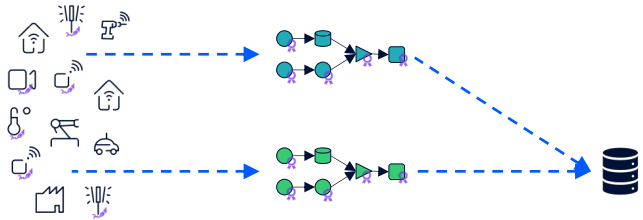
Open sourced at https://github.com/Nokia-Bell-Labs/zkstream

zkStream: a framework for trustworthy stream processing

A **trustworthy** stream processing framework, relying on two ingredients:

- **Signatures** on sensor inputs
- **Zero-Knowledge Proofs** for operator code

in an **architecture** that supports typical **streaming features** and with optimized **gadgets for aggregation operations**.



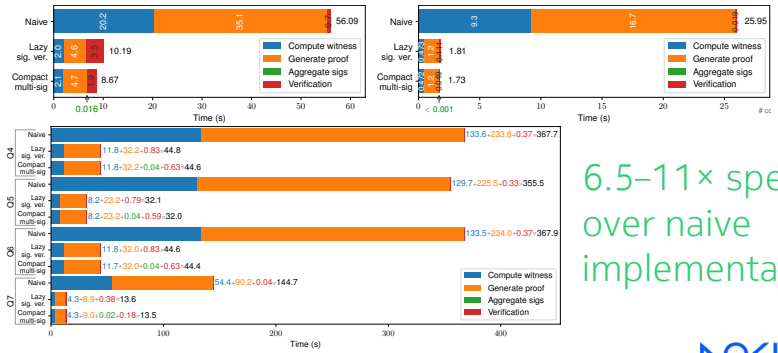
Optimizations:

1. **Lazy signature verification**
2. **Compact multi-signatures**

to make this feasible for practical applications.

Guarantees:

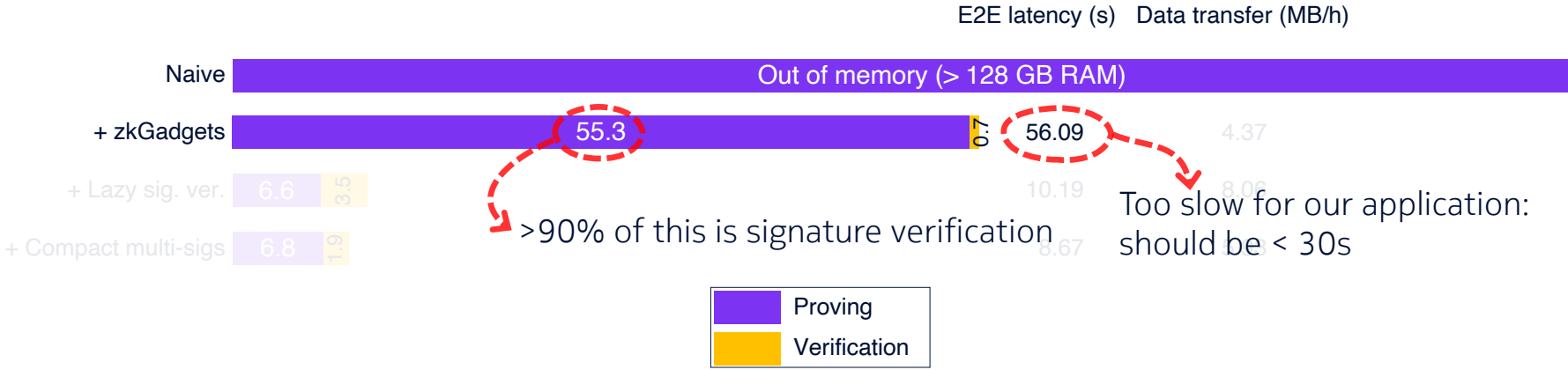
- ✓ **Confidentiality**: Data owner does not leak sensor data nor intermediate results.
- ✓ **Computational integrity**: Data consumer knows algorithm ran as specified.
- ✓ **Provenance**: Inputs are guaranteed to come from trusted sensors.



6.5–11× speed-up over naive implementation

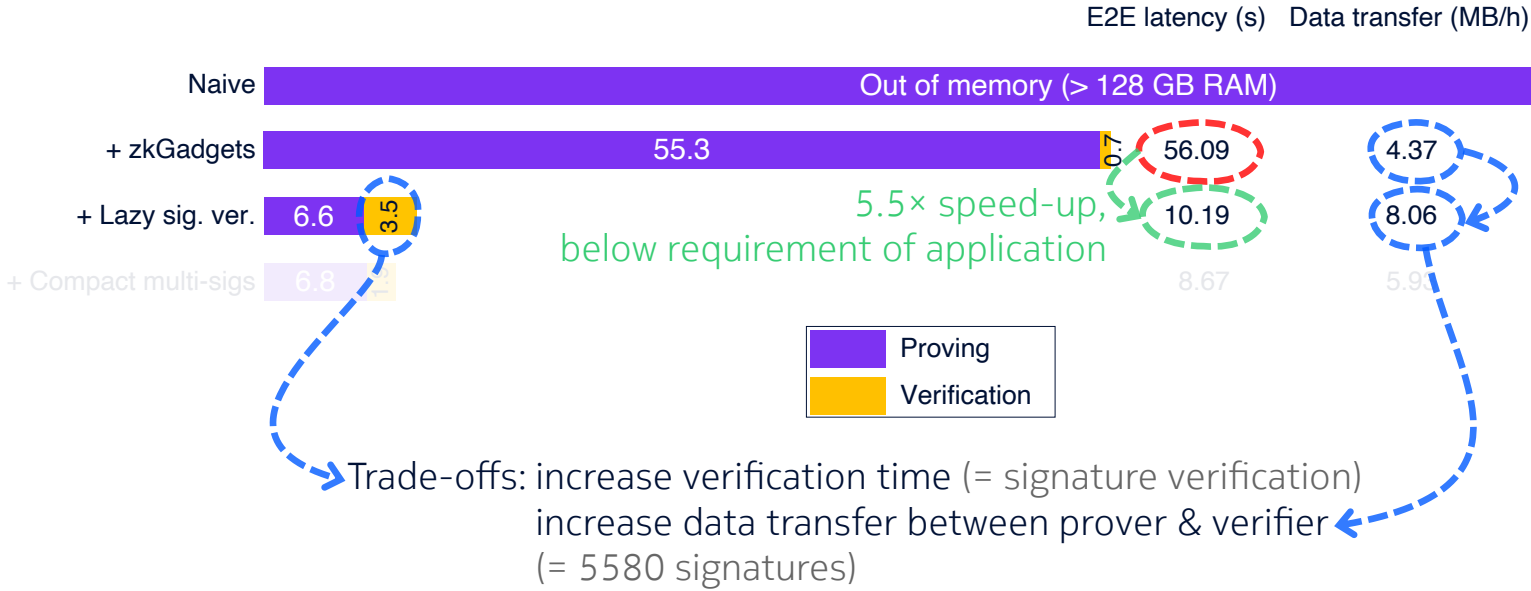
NOKIA
BELL
LABS

Performance analysis



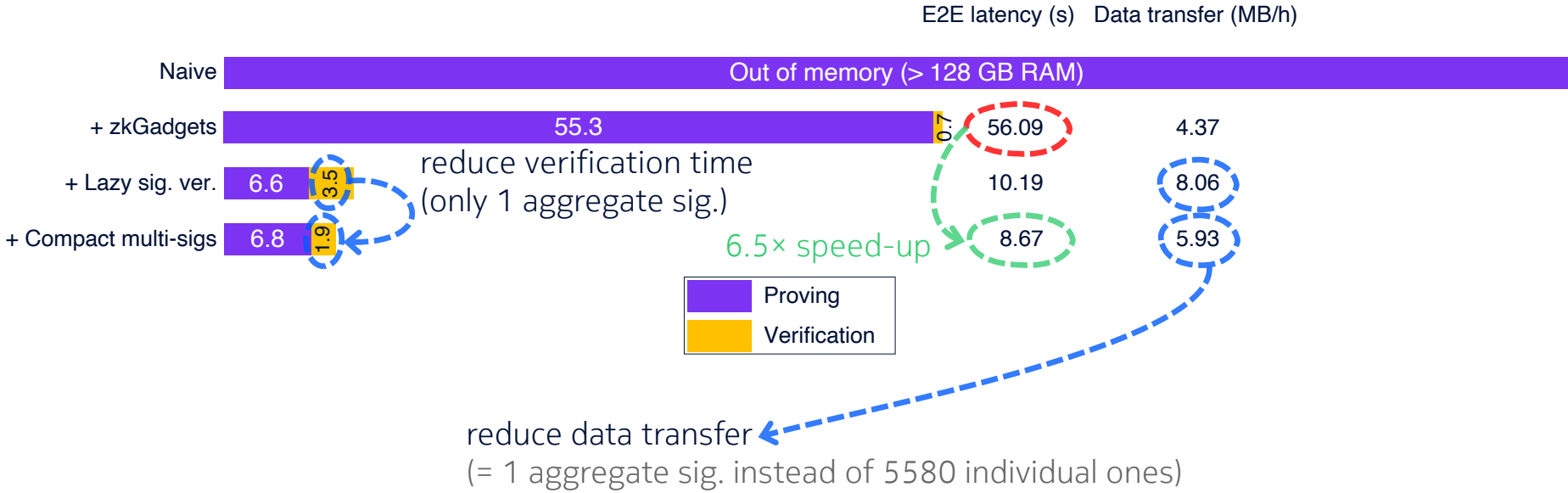
NOKIA BELL LABS

Optimizing zkStream: 1. Lazy signature verification

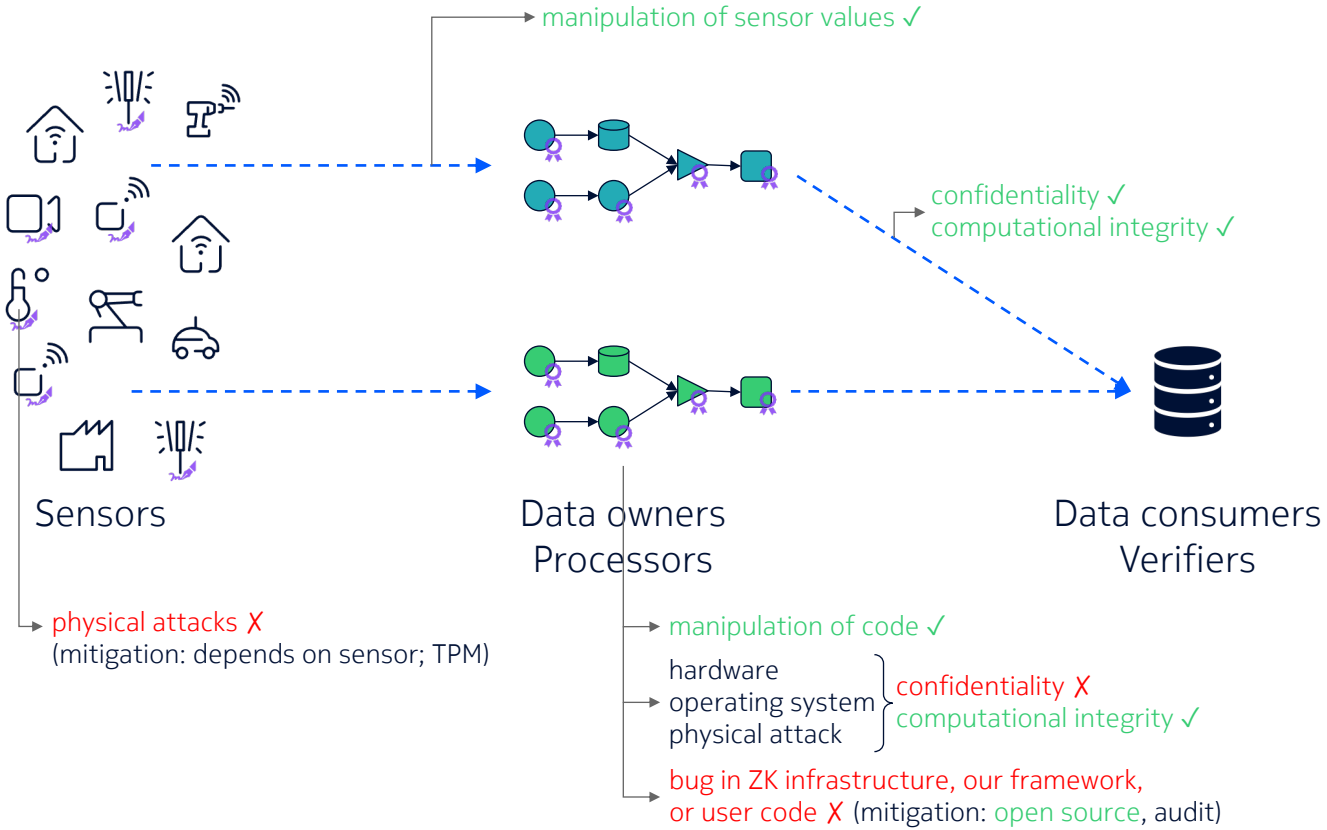


Optimizing zkStream: 2. Compact multi-signatures

💡 Signature verification outside ZKP
⇒ no need for 'ZK-friendly' signature scheme
⇒ use aggregate signature scheme (BLS)



Security analysis



Assumptions / Threat model

Sensors

- Limited processing power
- On premise of processor
- They function reliably
- They can contain secrets (i.e. signing keys), possibly protected using Trusted Platform Modules (TPM)

Processors

- Own the sensor data
- Want confidentiality
- Fully untrusted environment: tamper with data, act maliciously, physical attacks

(We assume a safe channel between processor and consumer, e.g. SSL)

Consumers

- Want correctness of output stream
- Should not learn raw sensor readings

Why not use TEEs?

Zero-Knowledge Proofs vs. Trusted Execution Environments

We are running in a fully untrusted environment, e.g. household, where the attacker has **physical access** to the system. (TEEs are often used in the cloud, in data centers with controlled physical access.)

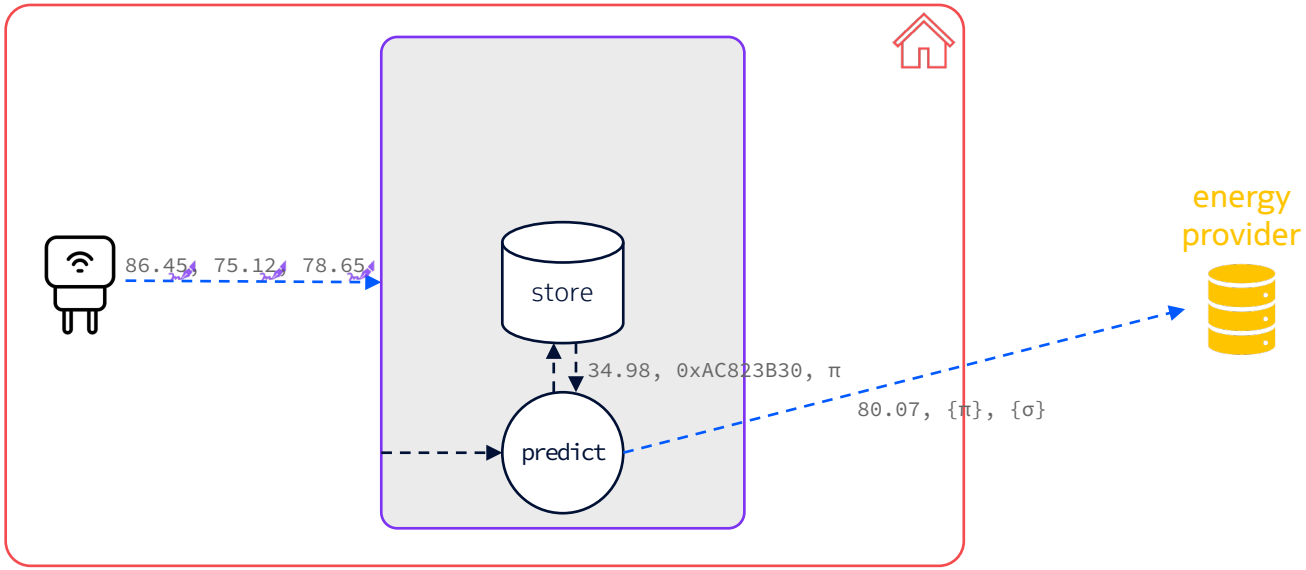
- TEEs have been subject to many **exploits**.
  → exacerbated in a **fully untrusted environment with physical access**.
- **Patches** require firmware updates and sometimes even hardware upgrades.
  → tricky in our context: would require **end user** to update firmware or even supplying and installing **new hardware**.

We aim to explore ZKPs as an alternative.

- ZKPs are **software-only** and based on cryptography, not requiring trust in HW. This is interesting.
- **TCB (Trusted Computing Base) is smaller** for ZKP. (But maybe less well tested?)

NOKIA BELL LABS

How to support stateful operators



We already support a store, in which hidden values can be stored.

To implement a stateful operator, you can store the state in the store and re-use it on the next invocation.

(Potentially, could use incremental/recursive proving systems like Nova to avoid increasing proof size.)

Copyright and confidentiality

NOKIA BELL LABS