The BU Secure Analytics Stack

A 5-year journey

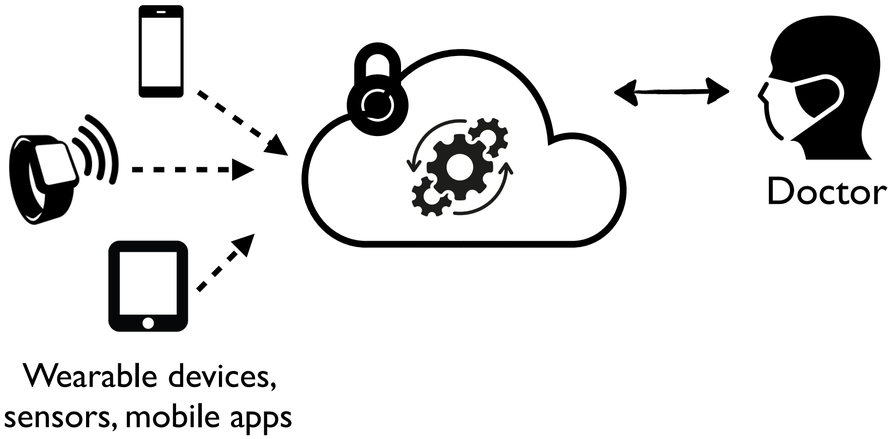
John Liagouris
liagos@bu.edu

BUSec

Real-world use cases

Digital health analytics

(BU Medical & Hariri Institute for Computing)

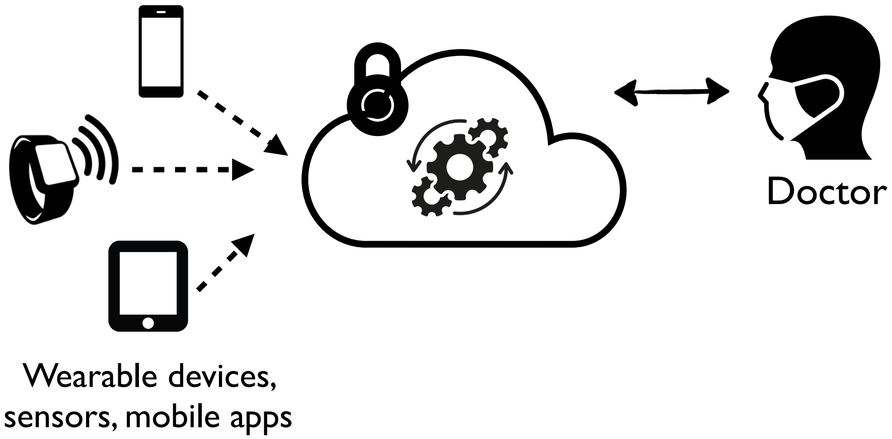


What is the % of non-diabetic patients whose glucose levels reach 180mg/dL while eating?

Real-world use cases

Digital health analytics

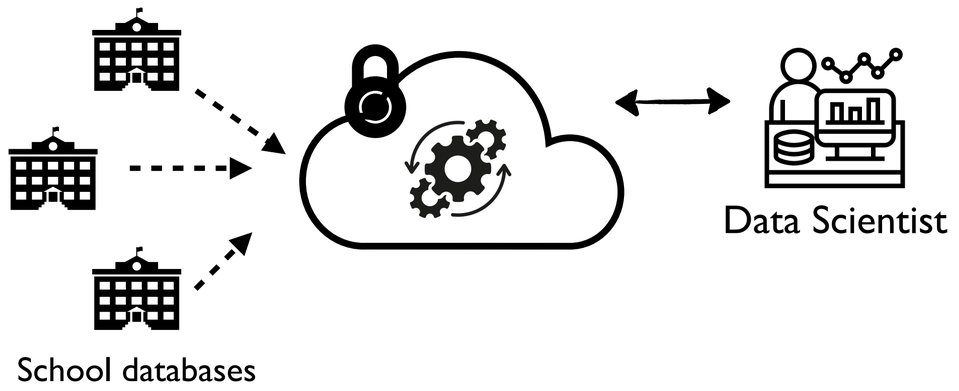
(BU Medical & Hariri Institute for Computing)



What is the % of non-diabetic patients whose glucose levels reach 180mg/dL while eating?

Predictive models in education

(BU Wheelock & McGovern Institute for Brain Research)

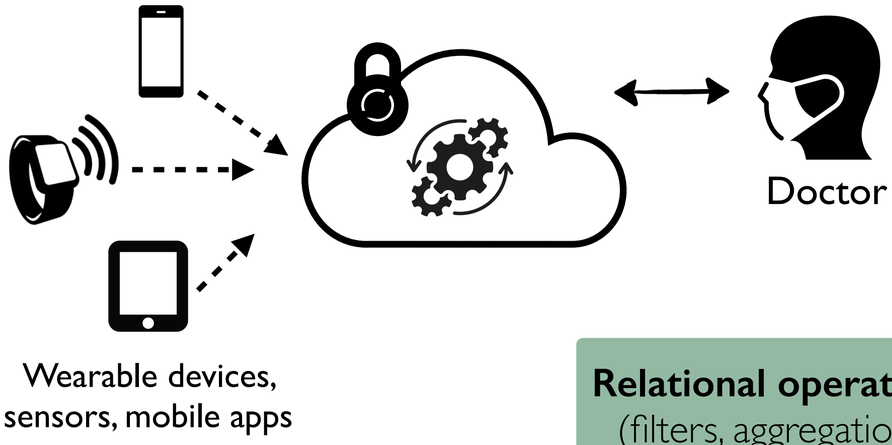


What is the expected student reading performance given past exam scores and family histories?

Data analysis pipelines are complex

Digital health analytics

(BU Medical & Hariri Institute for Computing)



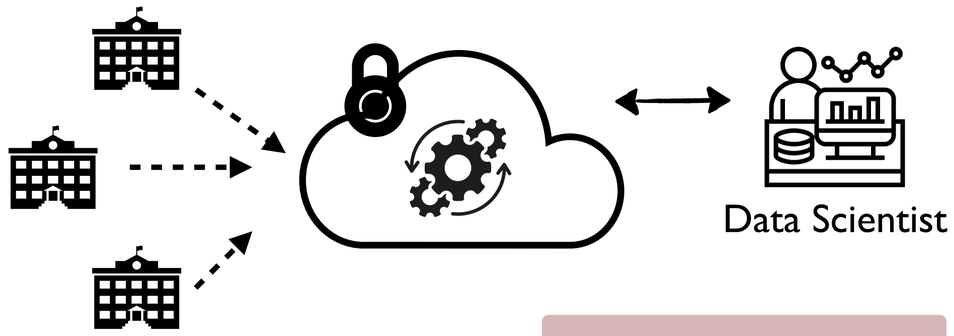
Relational operations
(filters, aggregations)

What is the % of non-diabetic patients whose glucose levels reach 180mg/dL while eating?

Time series operations
(threshold windows)

Predictive models in education

(BU Wheelock & McGovern Institute for Brain Research)

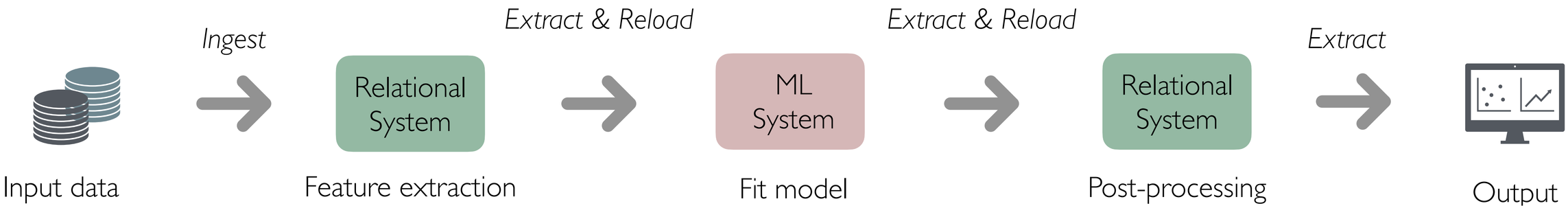


Machine learning
(logistic regression)

What is the expected student reading performance given past exam scores and family histories?

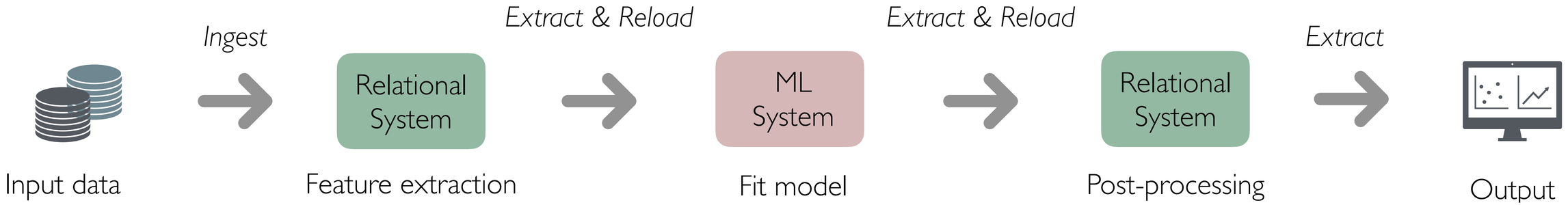
Relational operations
(joins on tabular data)

Lessons learned from the "big data" era



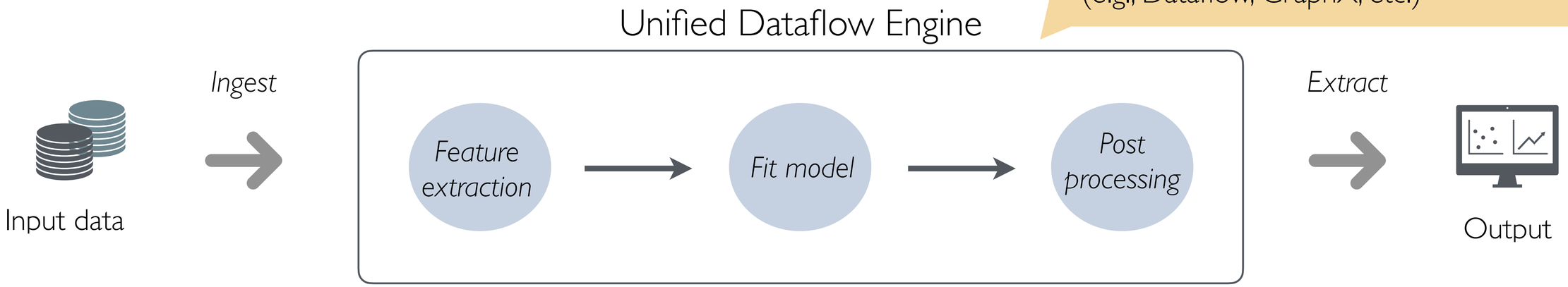
Specialized systems have **high maintenance and orchestration costs**

Lessons learned from the "big data" era

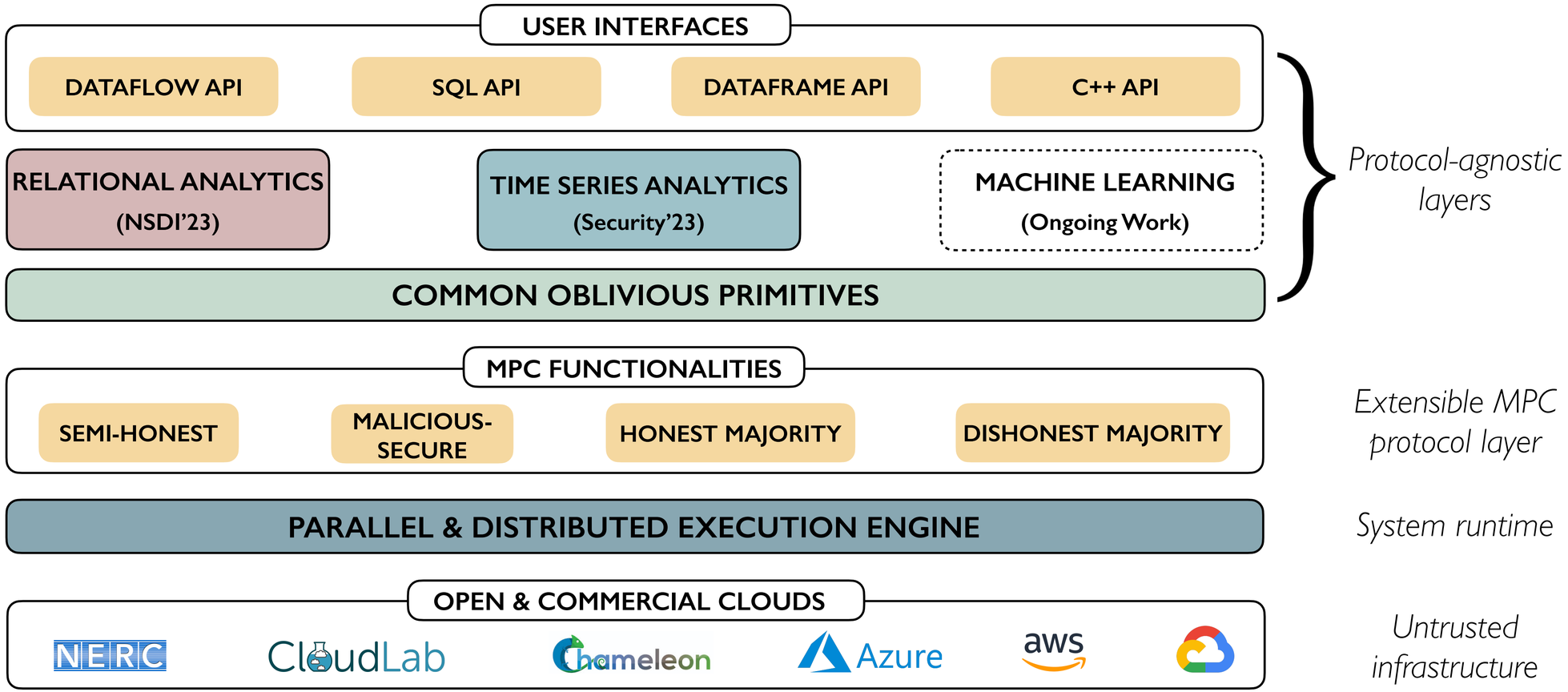


Specialized systems have **high maintenance and orchestration costs**

- **Common abstractions** (e.g., Spark RDDs)
- **High-level libraries and DSLs** on top (e.g., Dataflow, GraphX, etc.)



A unified analytics stack for MPC



Vision: Secure analytics in the cloud with MPC

+ Relational, time series, ML analytics
+ Full security guarantees of MPC (no leakage)
+ Practical performance

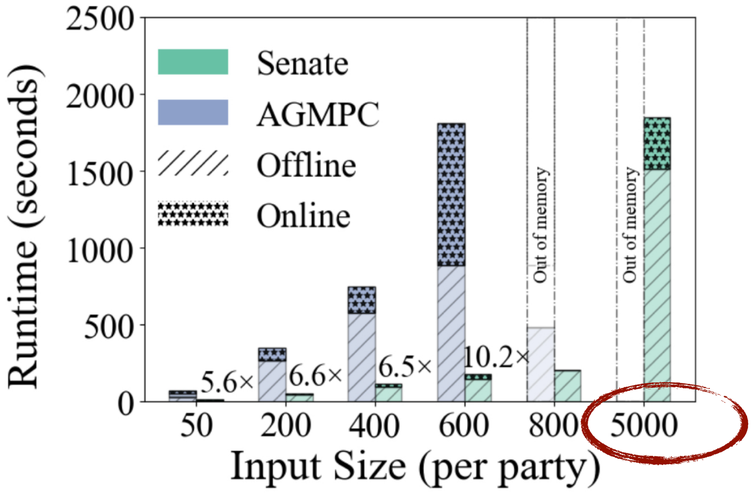
Overhead of MPC analytics

"Running the query entirely under MPC […] fails to scale beyond 3,000 total records…"

"The primary source of the slowdown arises from their join operators that have hundreds of input tuples…"

"Computing a function f on millions of client inputs […] could potentially take an astronomical amount of time in a full MPC."

Experimental Setup. We perform our experiments using r5.12xlarge Amazon EC2 instances in the Northern California region. Each instance offers 48 vCPUs and 384 GB of RAM, and was additionally provisioned with 20 GB of swap space, to account for transient spikes in memory requirements.



[1] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros. *Conclave: secure multi-party computation on big data*. EuroSys, 2019.

[2] J. Bater, G. Elliott, C. Eggen, S. Goel, A. N. Kho, and J. Rogers. *SMCQL: secure querying for federated databases*. PVLDB, 10(6):673–684, 2017.

[3] H. Corrigan-Gibbs and D. Boneh. *Prio: Private, Robust, and Scalable Computation of Aggregate Statistics*, NSDI, 2017.

[4] R. Poddar, S. Kalra, A. Yanai, R. Deng, RA. Popa and JM. Hellerstein. *Senate: A Maliciously-Secure MPC Platform for Collaborative Analytics*, Security, 2021.

Barriers to making MPC practical

Poor performance

*Best published results 5 years ago would perform **simple analytics** on very **small inputs** (100s-1000s records)*

Protocol dependence

*Analytics systems were built for **specific protocols and settings** (i.e., number of parties, data ownership constraints)*

Programming complexity

***Limited support for familiar APIs**; analysts need to have some basic understanding of MPC*

Tedious deployment

*Setting up computing parties and deploying the computation **requires cryptographic expertise***

The BU Secure Analytics Stack

Performance: Built entirely from scratch

- **Vectorized** and **data-parallel** by design
- Custom LAN/WAN communicator

Protocol independence: Configurable security

- Support for **multiple threat models**
- **Extensible** protocol layer

Ease of programming: Familiar APIs

- **Relational, time series,** and **ML** APIs
- **Protocol-agnostic** oblivious primitives

Ease of deployment: Cloud automation

- Designed for the **outsourced setting**
- Built-in support for **cloud infrastructure**

High-level APIs

USER INTERFACES

DATAFLOW API | SQL API | PANDAS-like API | C++ API | GRAPHICAL UI

Oblivious Operators

RELATIONAL LIBRARY

SELECT | JOIN | DISTINCT | GROUP BY | SEMI-JOIN | ...

TIME SERIES LIBRARY

TUMBLING WINDOW | SESSION WINDOW | SNAPSHOT | ...

MACHINE LEARNING (Ongoing)

REGRESSION | CLASSIFICATION | INFERENCE | ...

Oblivious Primitives

INTERMEDIATE VECTORIZED PRIMITIVES

COMPARISON | MULTIPLEXER | RCA | BUTTERFLY | PERM GEN | SORT | SHUFFLE | RELU | ...

Protocol Layer

IDEAL VECTORIZED FUNCTIONALITIES

Arithmetic sharing (numbers): ADD (+) | MUL (×) | DIV (/)

Boolean sharing (bitstrings): XOR (⊕) | AND (∧)

SEMI-HONEST SECURITY | MALICIOUS SECURITY | HONEST MAJORITY | DISHONEST MAJORITY

RANDOMNESS

BEAVER TRIPLES | ZERO SHARING PRG | AUTHENTICATION

COMMUNICATION

SEND | RECV | EXCHANGE | OBLIV. TRANSFER

PARALLEL & DISTRIBUTED RUNTIME

UNTRUSTED INFRASTRUCTURE

FABRIC | SPHERE RESEARCH INFRASTRUCTURE | CloudLab | NERC | aws | Azure

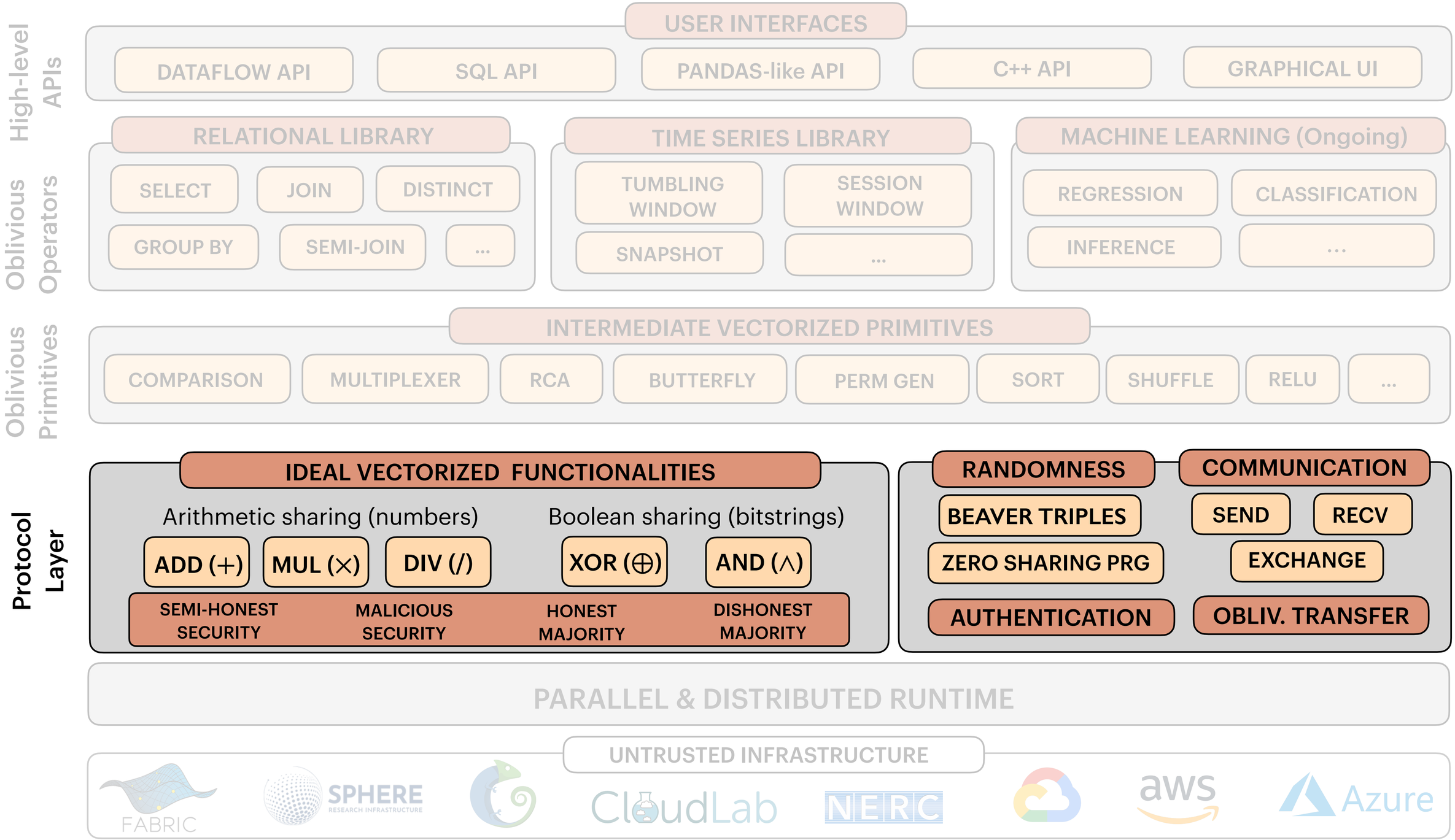
Data analyst APIs

Dataflow API

```
// Define time series schema
TS ts = get_shares ({"TIMESTAMP", "PID", "DIABETES", "GLUCOSE"});
// Compute the maximum glucose value for each window
TS res = ts.filter("DIABETES" == 0)
           .keyBy("PID")
           .threshold_window("TIMESTAMP", "GLUCOSE", 180)
           .aggregate("GLUCOSE", Aggregation::MAX);
```

PANDAS-like API

```
// Define data schema and allocate table with 10 rows
EncodedTable<int> Salaries("SALARIES", {"HOURLY_RATE", "HOURS", "TOTAL"}, 10);
...
// Compute total salary per employee and store it in column "TOTAL"
Salaries["TOTAL"] = Salaries["HOURLY_RATE"] * Salaries["HOURS"];
// Sort employee records from lower to higher salaries
Salaries.sort({"TOTAL"}, SortOrder::ASC);
```


High-level APIs
USER INTERFACES
DATAFLOW API
SQL API
PANDAS-like API
C++ API
GRAPHICAL UI
Oblivious Operators
RELATIONAL LIBRARY
SELECT
JOIN
DISTINCT
GROUP BY
SEMI-JOIN
...
TIME SERIES LIBRARY
TUMBLING WINDOW
SESSION WINDOW
SNAPSHOT
...
MACHINE LEARNING (Ongoing)
REGRESSION
CLASSIFICATION
INFERENCE
...
Oblivious Primitives
INTERMEDIATE VECTORIZED PRIMITIVES
COMPARISON
MULTIPLEXER
RCA
BUTTERFLY
PERM GEN
SORT
SHUFFLE
RELU
...
Protocol Layer
IDEAL VECTORIZED FUNCTIONALITIES
Arithmetic sharing (numbers)
ADD (+)
MUL (×)
DIV (/)
Boolean sharing (bitstrings)
XOR (⊕)
AND (∧)
SEMI-HONEST SECURITY
MALICIOUS SECURITY
HONEST MAJORITY
DISHONEST MAJORITY
RANDOMNESS
BEAVER TRIPLES
ZERO SHARING PRG
AUTHENTICATION
COMMUNICATION
SEND
RECV
EXCHANGE
OBLIV. TRANSFER
PARALLEL & DISTRIBUTED RUNTIME
UNTRUSTED INFRASTRUCTURE
FABRIC
SPHERE
RESEARCH INFRASTRUCTURE
CloudLab
NERC
aws
Azure

Protocol developer APIs

- Implement ideal functionalities (e.g., `+, ×, ⊕, ∧`)
- Define conversion operations (e.g., `a2b, b2a`)
- Select randomness source
- Set up party communication

Upper layers of the stack use functionalities as "black boxes"

```
/**
 * @brief Perform bitwise AND between two binary secret shared values.
 * Consumes one Beaver AND triple. Direct analog to multiplication.
 *
 * @param x binary shared input
 * @param y binary shared input
 * @param z binary shared output
 */
void and_b(const EVector &x, const EVector &y, EVector &z) {
    auto [a, b, c] = BTANDgen->getNext(x.size());

    auto A = open_shares_b(x ^ a);
    auto B = open_shares_b(y ^ b);

    z = (y & A) ^ (a & B) ^ c;
}
```

Only allow vector input/output

Overloaded operations on vectors

2PC AND

Configuring the threat model

```
using namespace BUSTACK::service::spdz;  // Will compile program against the SPDZ functionalities
...
// Define time series schema
TS ts = get_shares ({"TIMESTAMP", "PID", "DIABETES", "GLUCOSE"});
// Compute the maximum glucose value for each window
TS res = ts.filter("DIABETES" == 0)
            .keyBy("PID")
            .threshold_window("TIMESTAMP", "GLUCOSE", 180)
            .aggregate("GLUCOSE", Aggregation :: MAX);
```

High-level APIs

USER INTERFACES

DATAFLOW API | SQL API | PANDAS-like API | C++ API | GRAPHICAL UI

Oblivious Operators

RELATIONAL LIBRARY

SELECT | JOIN | DISTINCT | GROUP BY | SEMI-JOIN | ...

TIME SERIES LIBRARY

TUMBLING WINDOW | SESSION WINDOW | SNAPSHOT | ...

MACHINE LEARNING (Ongoing)

REGRESSION | CLASSIFICATION | INFERENCE | ...

Oblivious Primitives

INTERMEDIATE VECTORIZED PRIMITIVES

COMPARISON | MULTIPLEXER | RCA | BUTTERFLY | PERM GEN | SORT | SHUFFLE | RELU | ...

Protocol Layer

IDEAL VECTORIZED FUNCTIONALITIES

Arithmetic sharing (numbers): ADD (+) | MUL (×) | DIV (/)

Boolean sharing (bitstrings): XOR (⊕) | AND (∧)

SEMI-HONEST SECURITY | MALICIOUS SECURITY | HONEST MAJORITY | DISHONEST MAJORITY

RANDOMNESS

BEAVER TRIPLES | ZERO SHARING PRG | AUTHENTICATION

COMMUNICATION

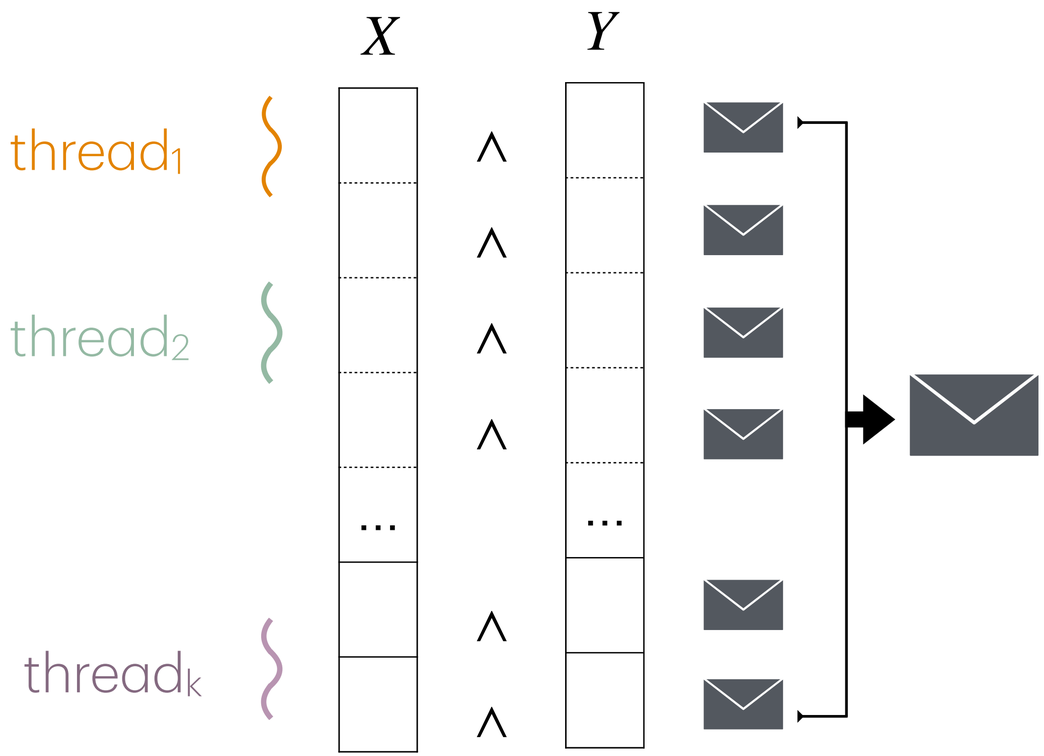
SEND | RECV | EXCHANGE | OBLIV. TRANSFER

PARALLEL & DISTRIBUTED RUNTIME

UNTRUSTED INFRASTRUCTURE

FABRIC | SPHERE RESEARCH INFRASTRUCTURE | CloudLab | NERC | aws | Azure

Vectorization enables **message batching** and **data parallelism**

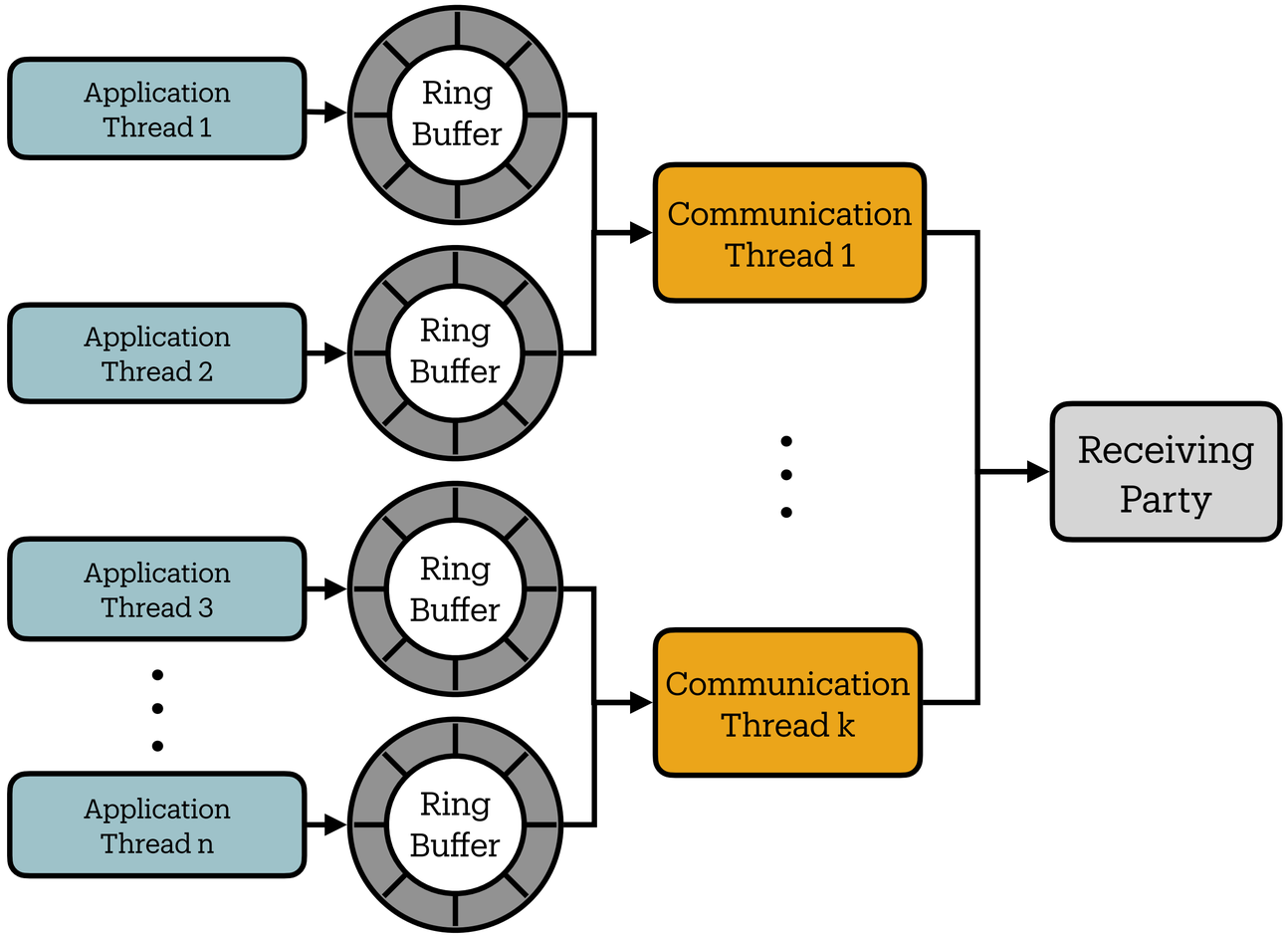


One communication round
for the entire vector

(independent of the number of elements)

Custom no-copy communicator

- **Dedicated communication threads** for send and receive
- **Lock-free ring buffers** between application and communication threads
- **Configurable** number of communication threads and **parallel TCP connections**
- WAN deployments benefit from more connections



High-level APIs

USER INTERFACES

SQL API | DATAFLOW API | PANDAS API | GRAHICAL UI | ML API

Oblivious Operators

RELATIONAL LIBRARY

SELECT | JOIN | DISTINCT | GROUP BY | SEMI-JOIN | ...

TIME SERIES LIBRARY

TUMBLING WINDOW | SESSION WINDOW | SNAPSHOT | ...

MACHINE LEARNING (Ongoing)

REGRESSION | CLASSIFICATION | INFERENCE | ...

Oblivious Primitives

INTERMEDIATE VECTORIZED PRIMITIVES

COMPARISON | MULTIPLEXER | RCA | BUTTERFLY | PERM GEN | SORT | SHUFFLE | RELU | ...

Protocol Layer

IDEAL VECTORIZED FUNCTIONALITIES

Arithmetic sharing (numbers): ADD (+) | MUL (×) | DIV (/)

Boolean sharing (strings): XOR (⊕) | AND (∧)

SEMI-HONEST SECURITY | MALICIOUS SECURITY | HONEST MAJORITY | DISHONEST MAJORITY

RANDOMNESS

BEAVER TRIPLES | CORRELATED RAND. | AUTHENTICATION

COMMUNICATION

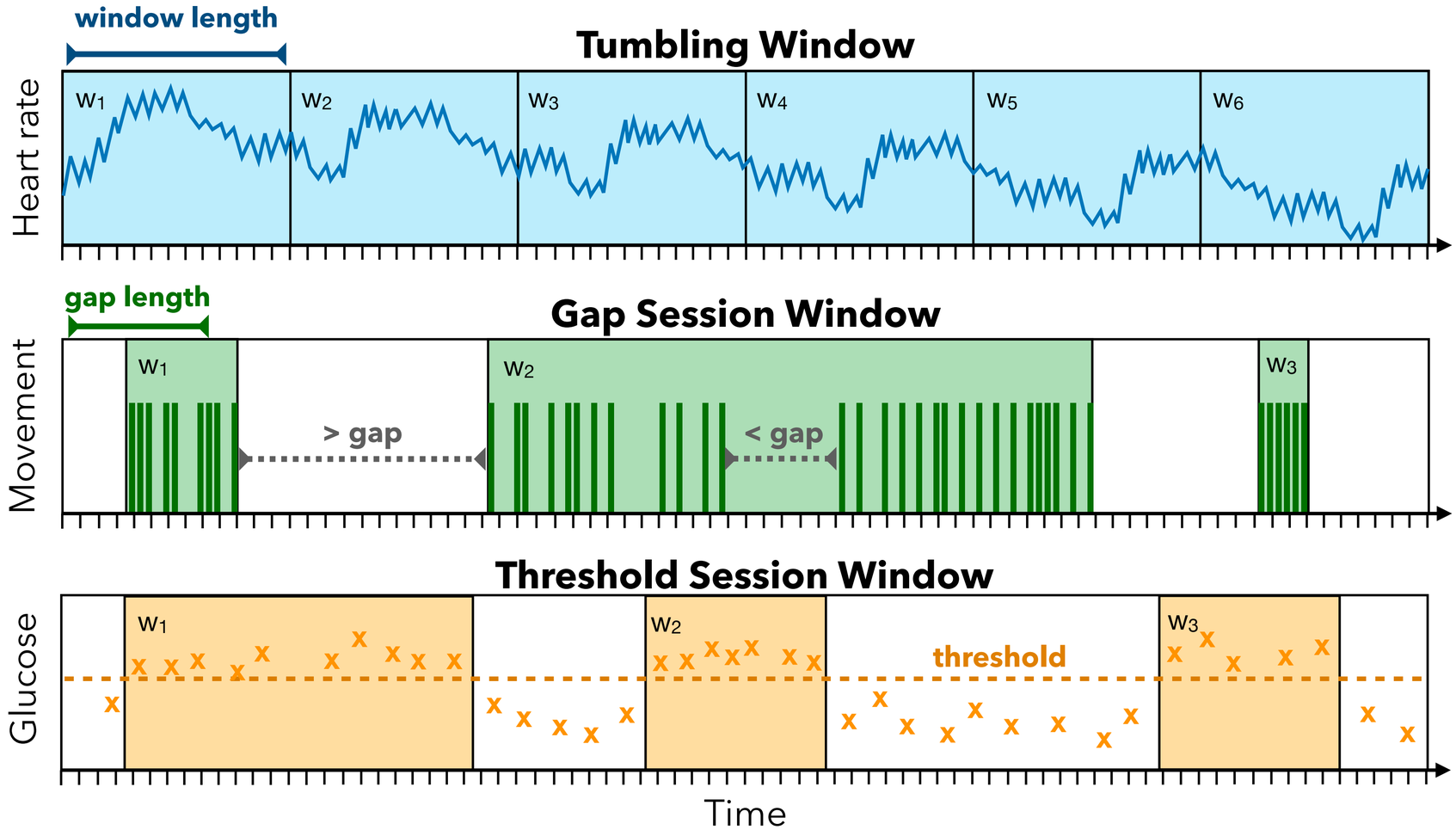
SEND | RECV | EXCHANGE | OBLIV. TRANSFER

PARALLEL & DISTRIBUTED RUNTIME

UNTRUSTED INFRASTRUCTURE

FABRIC | SPHERE RESEARCH INFRASTRUCTURE | CloudLab | NERC | aws | Azure

Secure time series analytics



M. Faisal, J. Zhang, J. Liagouris, V. Kalavri, M. Varia. *TVA: A multiparty computation system for secure and expressive time series analytics*. Security, 2023

Real-world applications with online constraints

Energy monitoring

- New data points every **10s**
- Dashboard updates every **5min**

"What is the total energy consumption collected by sensors of a large-scale grid every 5 minutes?"

(tumbling window)

Mobile health analytics

- New data points every **5min**
- Dashboard updates every **1hour**

"Given glucose measurements from a large patient cohort, what is the total number of insulin doses during a patient's eating period?"

(threshold window)

Cluster scheduling

"What is the number of tasks scheduled within the same job session per machine type in a large datacenter?"

(gap session window)

- 10K data points (scheduled tasks) every **1min**
- Dashboard updates every **10min**

[1] M. Jawurek, F. Kerschbaum, and G. Danezis. *Privacy technologies for smart grids - A survey of options*. MSR TR, 2012.

[2] A. Galderisi, L. Zammataro, E. Losiouk, G. Lanzola, et al. *Continuous glucose monitoring linked to an artificial intelligence risk index: early footprints of intraventricular hemorrhage inpreterm neonates*. Diabetes technology & therapeutics, 21(3):146–153, 2019.

[3] A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. *Large-scale cluster management at Google with Borg*. In EuroSys, 2015.

Real-world applications with online constraints

Energy monitoring

```
// Data schema
TS ts = get_shares({"TIMESTAMP",  "ENERGY_CONSUMPTION",
                    "TOTAL_CONSUMPTION"});

// Window aggregation
TS res = ts.tumbling_window("TIMESTAMP", 3600)
           .aggregate("ENERGY_CONSUMPTION",
            "TOTAL_CONSUMPTION",  Agg::SUM);
```

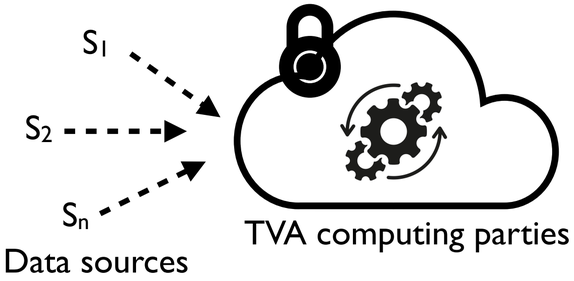
Mobile health analytics

```
// Data schema
TS ts = get_shares({"TIMESTAMP", "PATIENT_ID", "GLUCOSE",
                    "INSULIN", "TOTAL_INSULIN_EVENTS"});

// Window aggregation
TS res = ts.keyBy("PATIENT_ID")
           .threshold_window("TIMESTAMP", "GLUCOSE", 5)
           .aggregate("INSULIN", "TOTAL_INSULIN_EVENTS",
            Agg::COUNT);
```

Cluster scheduling

```
// Data schema
TS ts = get_shares({"TIMESTAMP",  "MACHINE_TYPE",
             "EVENT_TYPE", "TOTAL_TASKS_PER_SESSION"});

// Window aggregation
TS res = ts.filter("EVENT_TYPE" == 0)
           .keyBy("MACHINE_TYPE")
           .gap_window("TIMESTAMP",  10)
           .aggregate("*", "TOTAL_TASKS_PER_SESSION",
            Agg::COUNT);
```

TVA can support thousands of data sources in the online setting



	Maximum number of data sources (WAN)		
	Energy	Health	Cluster
Semi-honest (Araki et al.)	17400	174700	52400
Malicious-secure (Fantastic Four)	8700	87300	26210

TVA can sustain 87K+ patients sending glucose measurements every 5 minutes while providing malicious security

To guarantee online operation, the system must compute query results at a rate higher than that of data ingestion

Each TVA party uses one `r5.8xlarge` AWS instance (32 vCPUs, 256GB RAM) in a different AWS region

High-level APIs

USER INTERFACES

SQL API | DATAFLOW API | PANDAS API | GRAHICAL UI | ML API

Oblivious Operators

RELATIONAL LIBRARY

SELECT | JOIN | DISTINCT | GROUP BY | SEMI-JOIN | ...

TIME SERIES LIBRARY

TUMBLING WINDOW | SESSION WINDOW | SNAPSHOT | ...

MACHINE LEARNING (Ongoing)

REGRESSION | CLASSIFICATION | INFERENCE | ...

Oblivious Primitives

INTERMEDIATE VECTORIZED PRIMITIVES

COMPARISON | MULTIPLEXER | RCA | BUTTERFLY | PERM GEN | SORT | SHUFFLE | RELU | ...

Protocol Layer

IDEAL VECTORIZED FUNCTIONALITIES

Arithmetic sharing (numbers): ADD (+) | MUL (×) | DIV (/)

Boolean sharing (strings): XOR (⊕) | AND (∧)

SEMI-HONEST SECURITY | MALICIOUS SECURITY | HONEST MAJORITY | DISHONEST MAJORITY

RANDOMNESS

BEAVER TRIPLES | CORRELATED RAND. | AUTHENTICATION

COMMUNICATION

SEND | RECV | EXCHANGE | OBLIV. TRANSFER

PARALLEL & DISTRIBUTED RUNTIME

UNTRUSTED INFRASTRUCTURE

FABRIC | SPHERE RESEARCH INFRASTRUCTURE | CloudLab | NERC | aws | Azure

Secure relational analytics

```
select
        o_year,
        sum(case
                when nation = '[NATION]'
                then volume
                else 0
        end) / sum(volume) as mkt_share
from (
        select
                extract(year from o_orderdate) as o_year,
                l_extendedprice * (1-l_discount) as volume,
                n2.n_name as nation
        from
                part,
                supplier,
                lineitem,
                orders,
                customer,
                nation n1,
                nation n2,
                region
        where
                p_partkey = l_partkey
                and s_suppkey = l_suppkey
                and l_orderkey = o_orderkey
                and o_custkey = c_custkey
                and c_nationkey = n1.n_nationkey
                and n1.n_regionkey = r_regionkey
                and r_name = '[REGION]'
                and s_nationkey = n2.n_nationkey
                and o_orderdate between date '1995-01-01' and date '1996-12-31'
                and p_type = '[TYPE]'
        ) as all_nations
group by
        o_year
order by
        o_year;
```

Custom aggregation per group

Custom arithmetic expressions

8 input tables

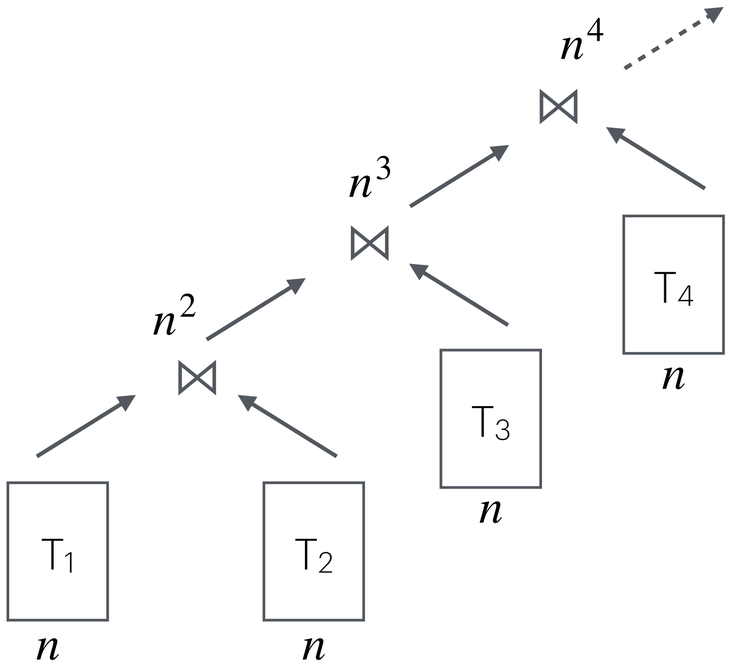
7 joins and 4 filters

1 group-by and 1 order-by on the join output

"The query determines how the market share of a given nation within a given region has changed over two years for a given part type"

chmark specification:
https://www.tpc.org/TPC_Documents_Current_Versions/pdf/TPC-H_v3.0.1.pdf

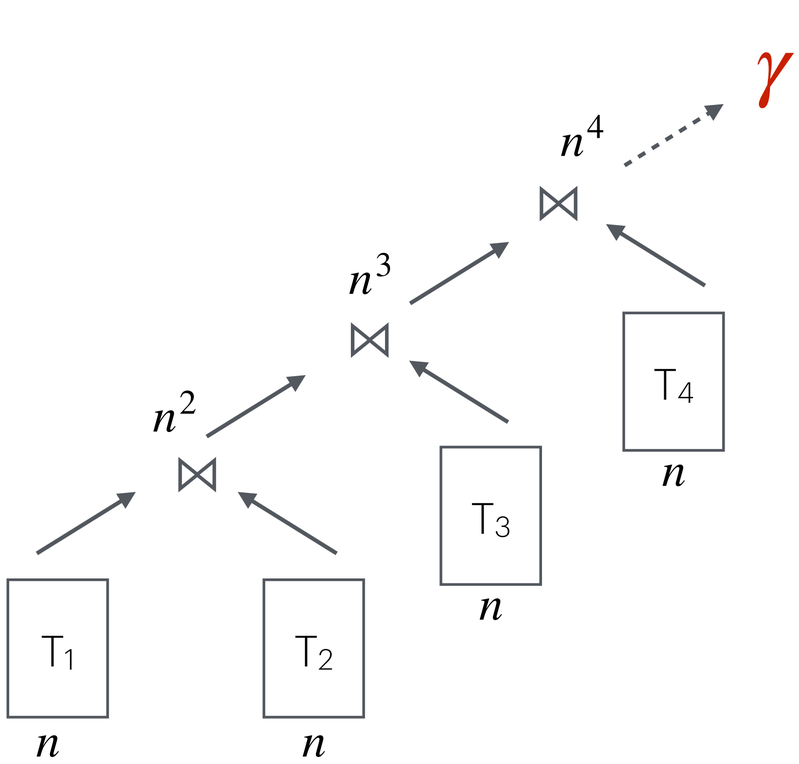
Core problem: the cascading effect



Joining k tables of n rows each would require $O(n^k)$ operations under MPC

- Oblivious operators produce **worst-case** outputs
- Potential **duplicates** in both join inputs require computing the **cartesian product**
- Duplicates are the **norm** — multiple data owners may contribute to the same tables
- A challenging open problem in oblivious analytics (MPC, TEEs, FHE, etc.)

Avoiding the cascading effect



Joining k tables of n rows each would require $O(n^k)$ operations under MPC

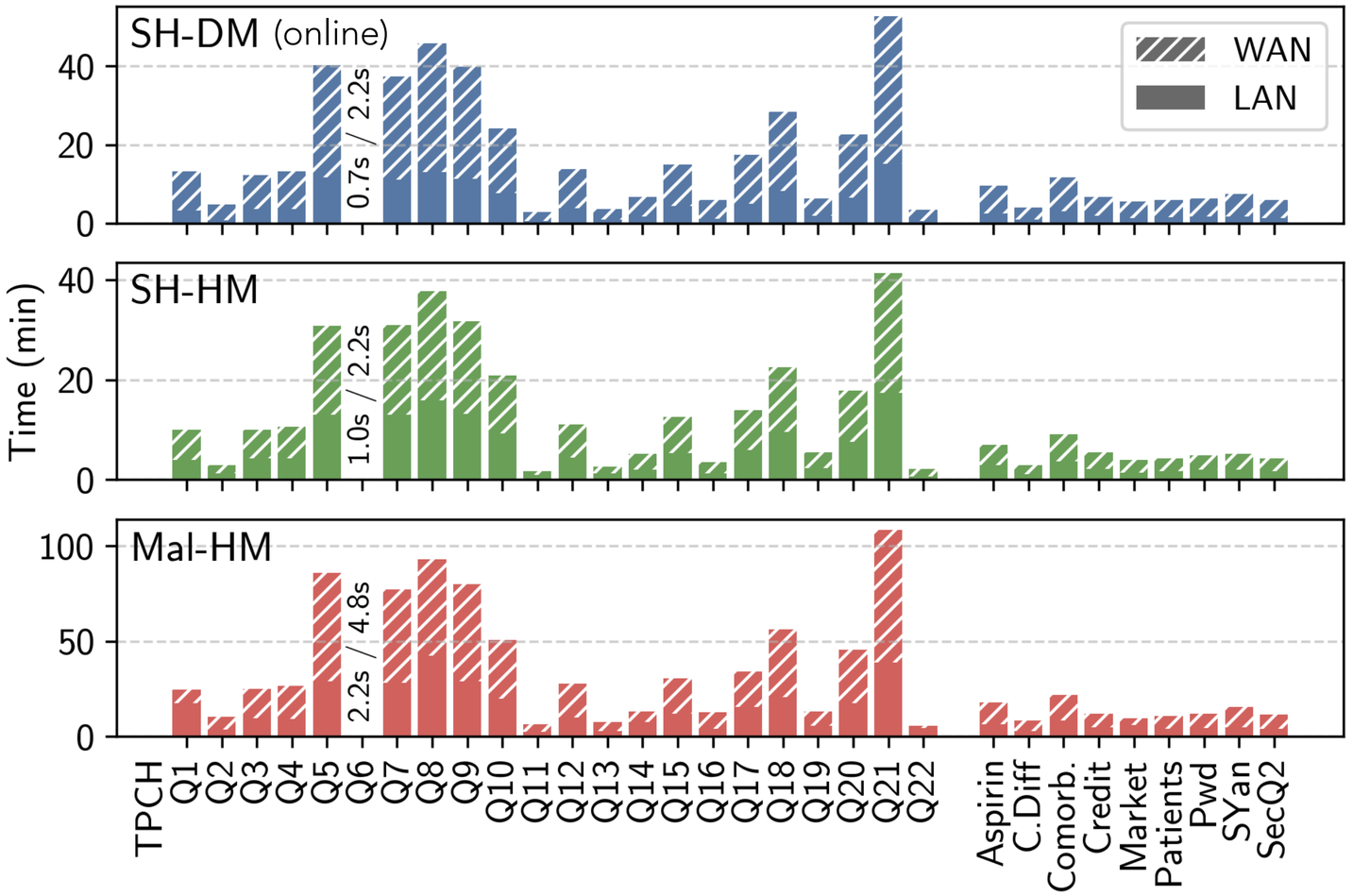
MPC queries typically return **aggregated** results

- The query result size is bounded by the size of the aggregated result (e.g., the number of groups)
- **Interesting fact**: this gives us an $O(n)$ size bound for a broad class of analytics, including all queries used in prior MPC works
- We leverage this bound to avoid computing the cartesian product and evaluate the entire query in $O(n \log n)$

(more at Eli's lightning talk)

Complex analytics under MPC are practical

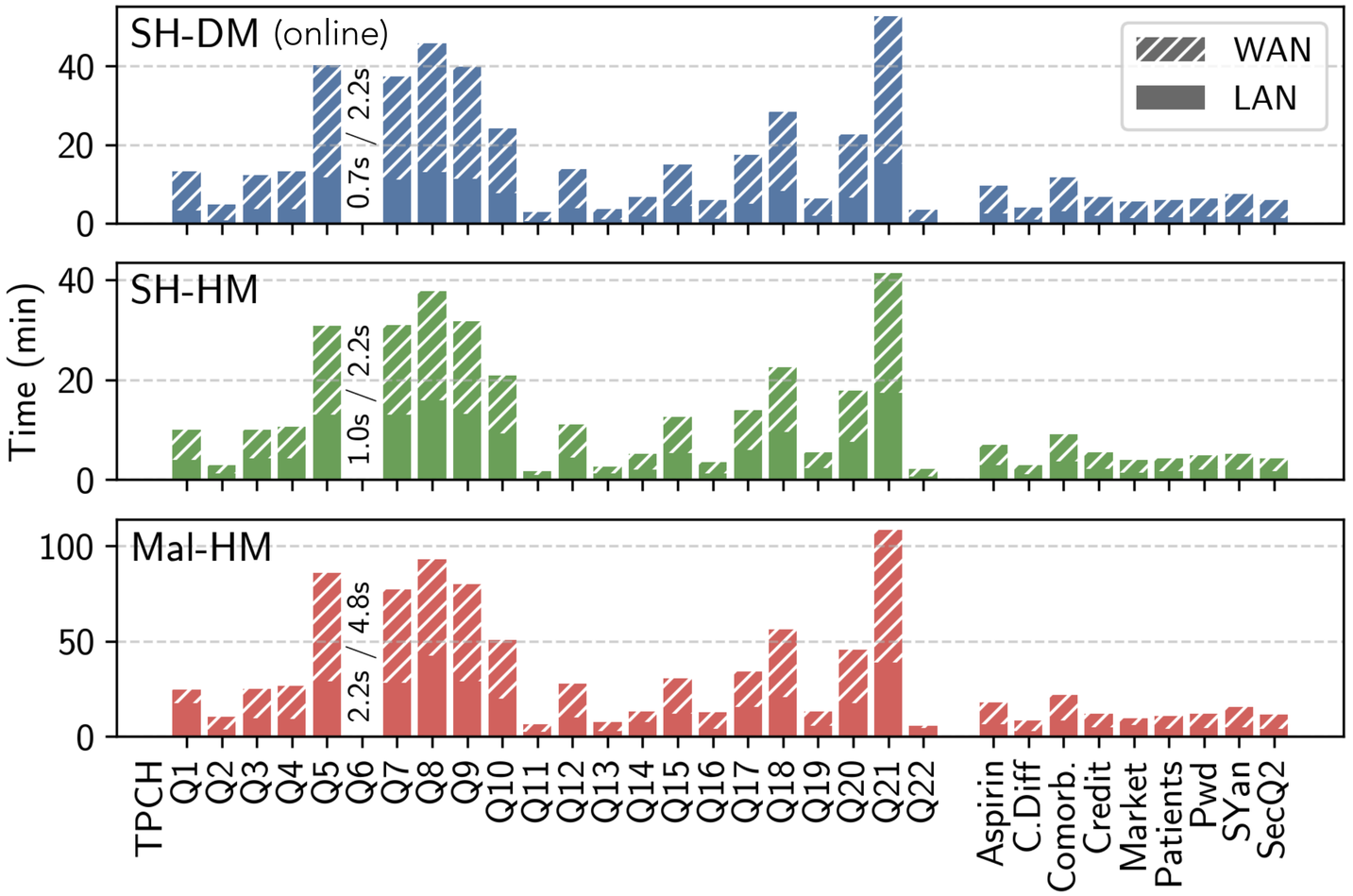
Complex analytics under MPC are practical



- **31** queries with **multiple joins** and **aggregations**, including all TPC-H queries at **SF1 and SF10** (up to 85M input records — 110M+ intermediate records)
- **3 protocols** with semi-honest and malicious security: Demmler et al. (SH-DM), Araki et al. (SH-HM), Dalskov et al. (Mal-HM)

Each computing party uses one `c7a.16xlarge` AWS instance (64 vCPUs, 128GB memory)

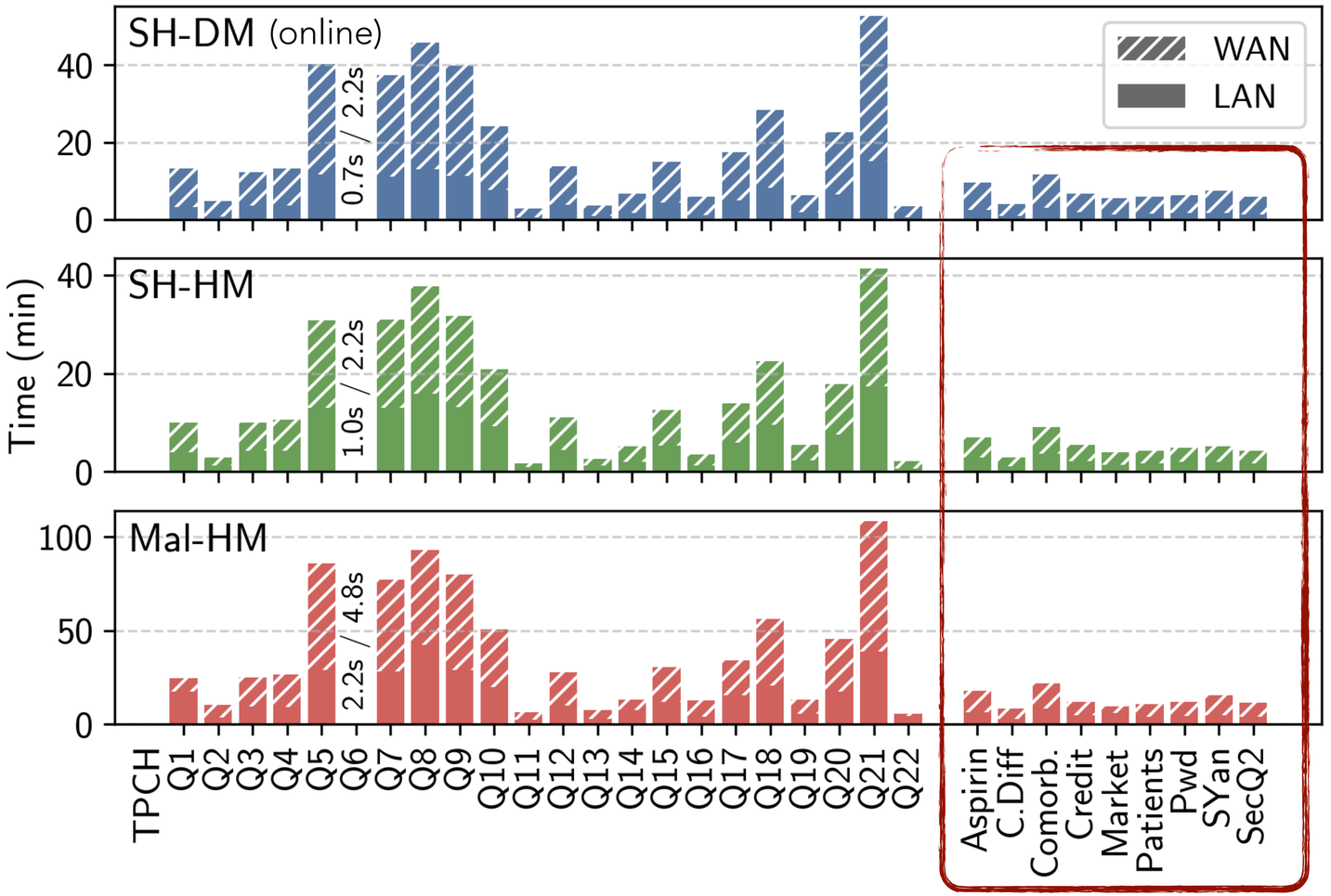
Complex analytics under MPC are practical



- **31** queries with **multiple joins** and **aggregations**, including all TPC-H queries at **SF1 and SF10** (up to 85M input records — 110M+ intermediate records)
- **3 protocols** with semi-honest and malicious security: Demmler et al. (SH-DM), Araki et al. (SH-HM), Dalskov et al. (Mal-HM)
- The most expensive query, **Q21** (12 oblivious sort calls) completes in **42min**, under **malicious** security

Each computing party uses one `c7a.16xlarge` AWS instance (64 vCPUs, 128GB memory)

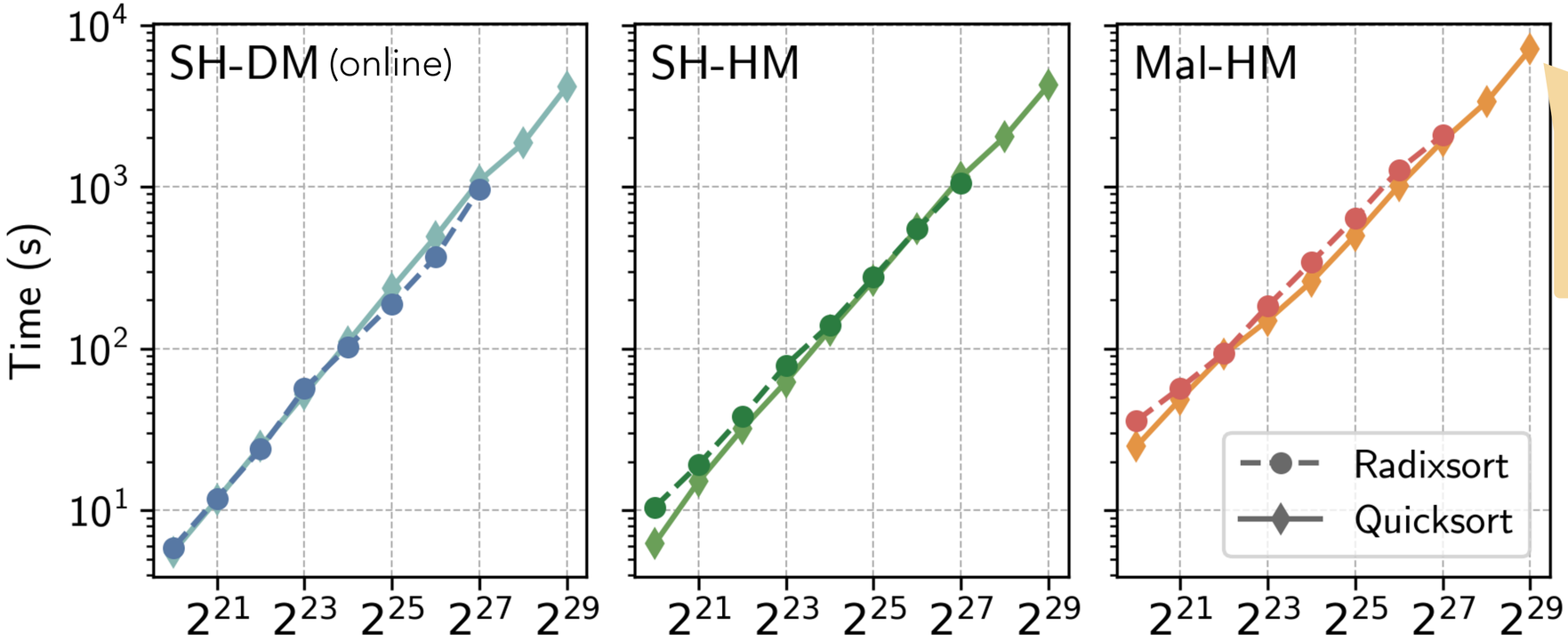
Complex analytics under MPC are practical



- **31** queries with **multiple joins** and **aggregations**, including all TPC-H queries at **SF1 and SF10** (up to 85M input records — 110M+ intermediate records)
- **3 protocols** with semi-honest and malicious security: Demmler et al. (SH-DM), Araki et al. (SH-HM), Dalskov et al. (Mal-HM)
- The most expensive query, **Q21** (12 oblivious sort calls) completes in **42min**, under **malicious** security
- All other queries from prior work in under **10min**

Each computing party uses one `c7a.16xlarge` AWS instance (64 vCPUs, 128GB memory)

Obliviously sorting half a billion elements

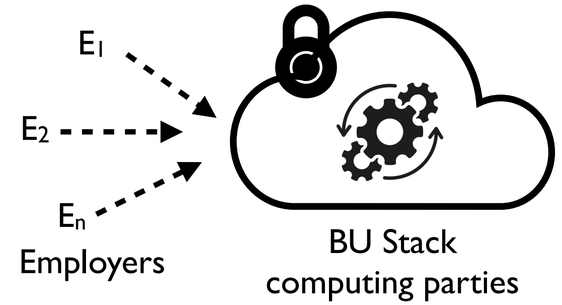


~2 hours for 512M elements with malicious security

LAN setting with one `c7a.16xlarge` AWS instance (64 vCPUs, 128GB memory) per party

The MA Workforce Data Report

Our latest use case

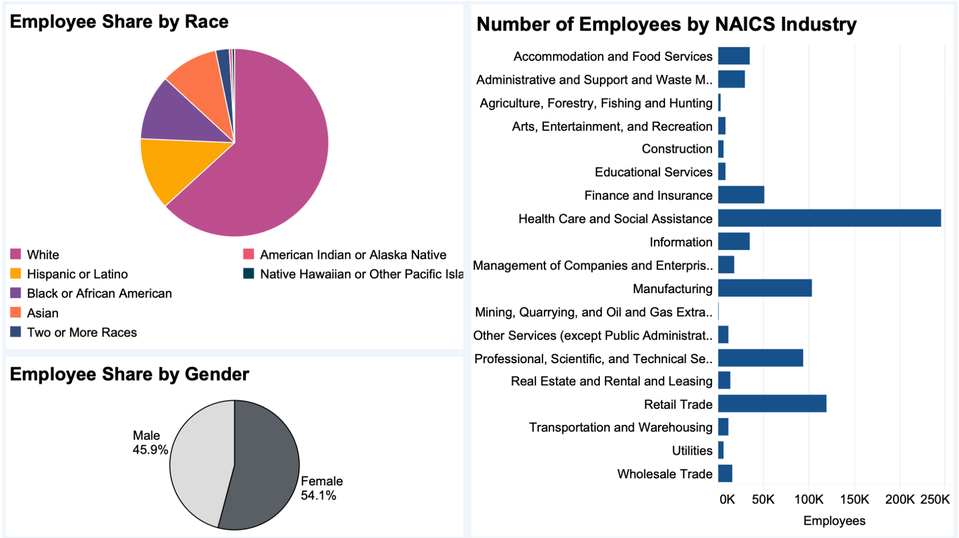


EEO form example

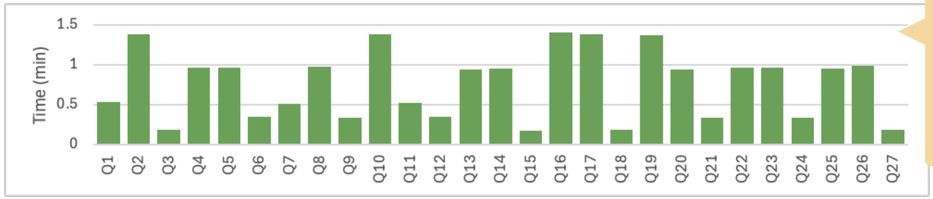
SECTION H – WORKFORCE DEMOGRAPHIC DATA

JOB CATEGORIES	Race/Ethnicity														Row Total
	Hispanic or Latino		Not Hispanic or Latino												
			Male						Female						
	Male	Female	White	Black or African American	Asian	Native Hawaiian or Other Pacific Islander	American Indian or Alaska Native	Two or More Races	White	Black or African American	Asian	Native Hawaiian or Other Pacific Islander	American Indian or Alaska Native	Two or More Races	
Executive/Senior Level Officials and Managers	0	0	8	0	1	0	0	0	6	1	1	0	0	1	18
First/Mid-Level Officials and Managers	9	22	128	12	18	0	0	5	306	46	30	0	1	6	583
Professionals	58	114	476	60	261	0	0	34	2294	192	406	4	4	64	3967
Technicians	42	76	183	96	66	0	1	15	522	225	162	1	3	54	1446
Sales Workers	0	1	1	0	0	0	0	0	6	1	1	0	0	0	10
Administrative Support Workers	21	130	53	56	23	0	1	10	353	379	77	2	4	37	1146
Craft Workers	8	0	29	15	2	0	0	1	0	0	0	0	0	0	55
Operatives	3	0	0	2	0	0	0	0	1	0	0	0	0	0	6
Laborers and Helpers	2	1	0	0	0	0	0	0	0	0	0	0	0	0	3
Service Workers	64	93	49	137	18	0	0	5	109	232	35	0	0	16	758
CURRENT 2023 REPORTING YEAR TOTAL	207	437	927	378	389	0	2	70	3597	1076	712	7	12	178	7992
PRIOR 2022 REPORTING YEAR TOTAL	70	136	203	116	67	0	0	10	1480	386	172	3	2	44	2689

(27 queries)



3,050 forms in 2025



All queries under **1.4min** in WAN with semi-honest security (Araki et al.)

https://www.mass.gov/info-details/2025-massachusetts-workforce-data-report-dashboard

Meet the team


Muhammad Faisal


Eli Baum


Sam Buxbaum


John Liagouris


Vasia Kalavri


Mayank Varia

BSc/MSc contributors:

Nitin Mathai, Vineet Raju, Iman Attia, Jerry Zhang, Pierre-Francois Wolf, Henry Wu, Lucas Ou, Bang Tran, Hasnain Abdul Rehman, Samyak Jain, Suli Hu, Yufeng Lin, Eric Chen, Derek Ewers, Ian Saucy, Xavier Ruiz, Jingyu Su

SECRECY (USENIX NSDI'23): https://www.usenix.org/system/files/nsdi23-liagouris.pdf

TVA (USENIX Security'23): https://www.usenix.org/system/files/usenixsecurity23-faisal.pdf

QueryShield (ACM SIGMOD Demos'24): https://dl.acm.org/doi/pdf/10.1145/3626246.3654749

with generous support from

BOSTON UNIVERSITY

NSF

BOSCH

The BU Secure Analytics Stack

A 5-year journey

John Liagouris
liagos@bu.edu

BUSec