# The Key to Effective UDF Optimization: Before Inlining, First Perform Outlining

**Samuel Arch**, Yuchen Liu, Todd C. Mowry,
Jignesh M. Patel, Andrew Pavlo

Carnegie Mellon
Database Group

VLDB 2025 Runner-Up Best Paper Award

# UDFs are Popular!

```sql
SELECT customer_name, is_vip(customer_key)
  FROM customer;
```

SQL

# UDFs are Popular!

```
SELECT customer_name, is_vip(customer_key)

  FROM customer;
```

SQL

UDF

```
CREATE FUNCTION is_vip(ckey INTEGER)

RETURNS BOOLEAN DECLARE DECIMAL total; BOOLEAN vip;

total = (SELECT SUM(o_totalprice) FROM orders

          WHERE o_custkey = ckey);

IF (total > 1000000) THEN vip = True;

                     ELSE vip = False;

RETURN vip;
```

# UDFs are Popular!

```
SELECT customer_name, is_vip(customer_key)

  FROM customer;
```

SQL

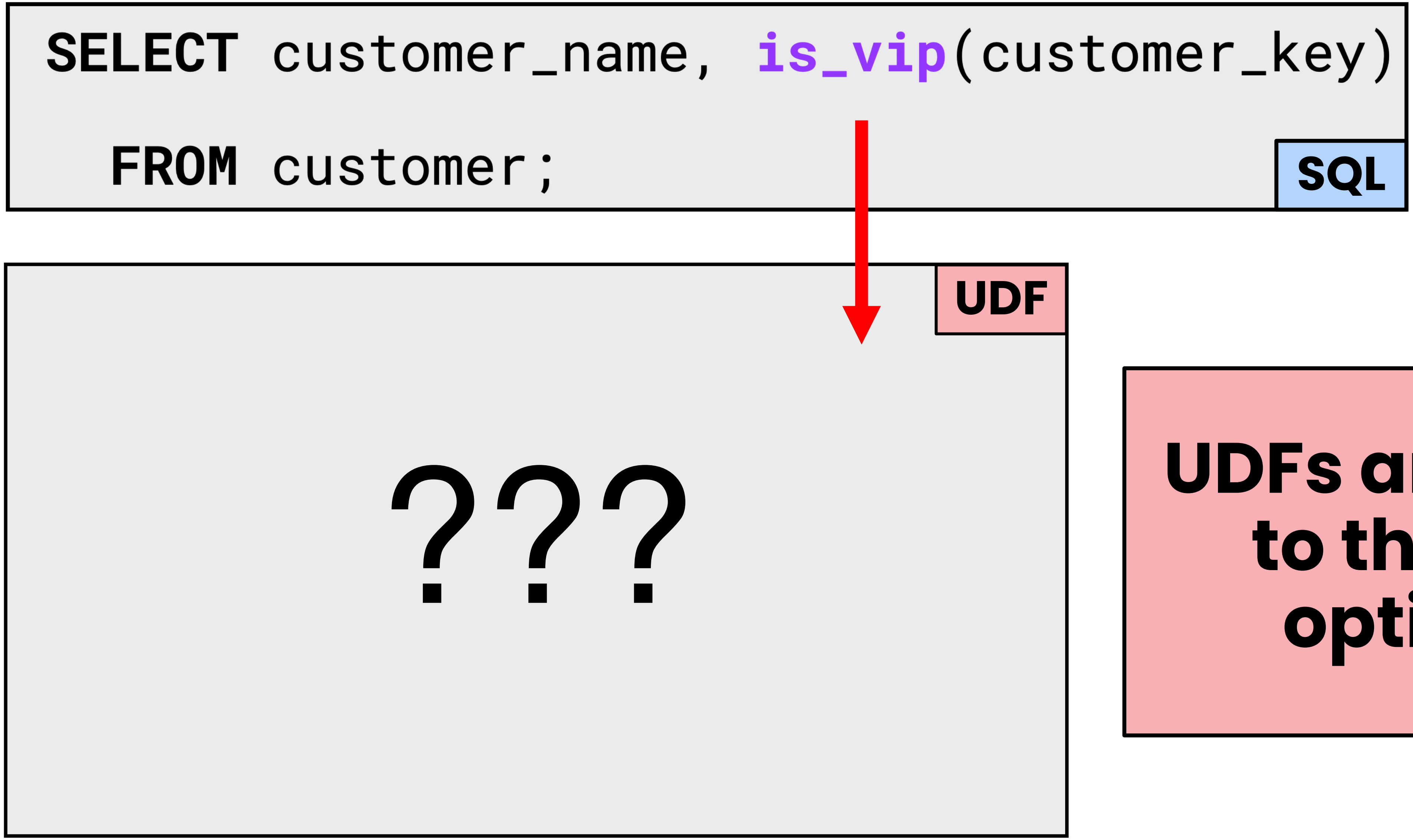
UDF

```
CREATE FUNCTION is_vip(ckey INTEGER)

RETURNS BOOLEAN DECLARE DECIMAL total; BOOLEAN vip;

total = (SELECT SUM(o_totalprice) FROM orders

          WHERE o_custkey = ckey);

IF (total > 1000000) THEN vip = True;

                     ELSE vip = False;

RETURN vip;
```

[VLDB 2018]

**Billions of queries per day invoke UDFs**

# UDF Optimization is Hard!



```
SELECT customer_name, is_vip(customer_key)
  FROM customer;
```

**UDFs are opaque to the query optimizer!**

# UDF Inlining (FROID)

UDF

**Hard to optimize**

✗

# UDF Inlining (FROID)

UDF

Hard to optimize

✗

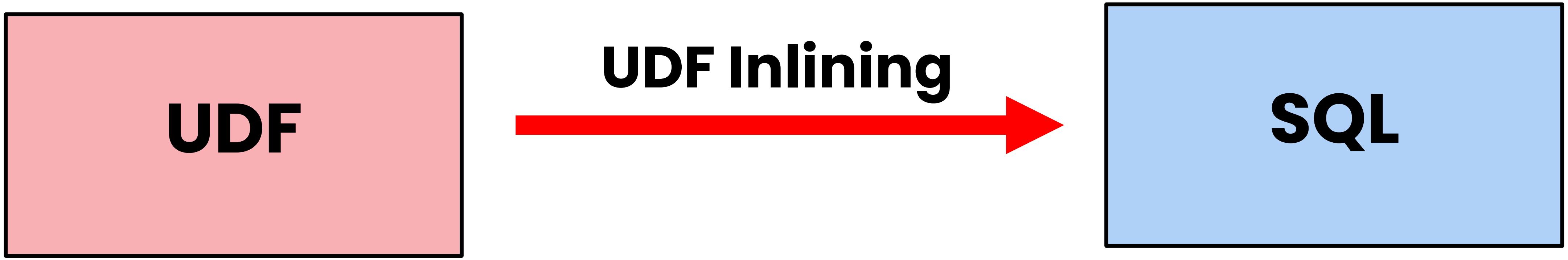
Let's translate UDFs to SQL

Microsoft

**VLDB 2018**

**Froid: Optimization of Imperative Programs in a Relational Database**

# UDF Inlining (FROID)



UDF

UDF Inlining

SQL

Hard to optimize

✗

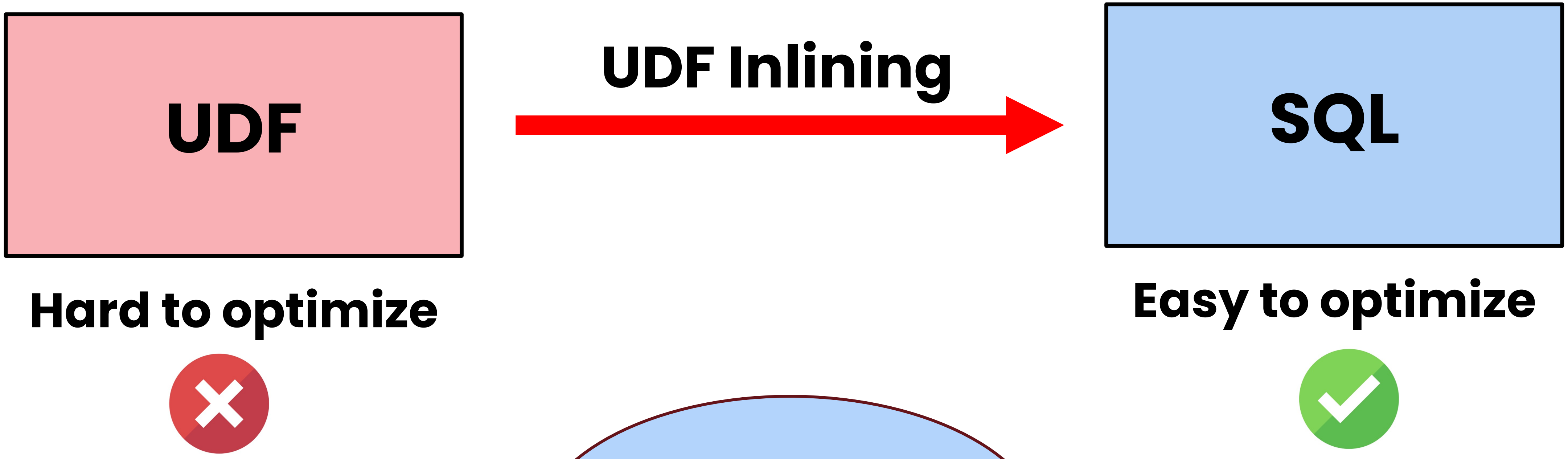
Let's translate UDFs to SQL

**VLDB 2018**

**Froid: Optimization of Imperative Programs in a Relational Database**

Microsoft

# UDF Inlining (FROID)



UDF

UDF Inlining

SQL

Hard to optimize ✗

Easy to optimize ✓

Let's translate UDFs to SQL

**<u>VLDB 2018</u>**

**Froid: Optimization of Imperative Programs in a Relational Database**

# Talk Overview

# Talk Overview

**Inlining leads to sub-optimal performance**

# Talk Overview

**Inlining leads to sub-optimal performance**

**Inlining entire UDFs creates complex queries**

# Talk Overview

**Inlining leads to sub-optimal performance**

**Inlining entire UDFs creates complex queries**

**Instead, we inline only the important pieces of a UDF**

# Talk Overview

**Inlining leads to sub-optimal performance**

**Inlining entire UDFs creates complex queries**

**Instead, we inline only the important pieces of a UDF**

**We achieve this through UDF outlining**

# Talk Overview

Inlining leads to sub-optimal performance

Inlining entire UDFs creates complex queries

Instead, we inline only the important pieces of a UDF

We achieve this through UDF outlining

Our approach outperforms inlining by more than 1000x

# UDF Inlining (FROID)

UDF

```
total = (SELECT ...);
IF (total > 1000000) THEN vip = True;
                          ELSE vip = False;
RETURN vip;
```

UDF Inlining

# UDF Inlining (FROID)

UDF

```
total = (SELECT ...);
IF (total > 1000000) THEN vip = True;
                     ELSE vip = False;
RETURN vip;
```

UDF Inlining

**Chain together translated UDF statements with LATERAL joins**

# UDF Inlining (FROID)

UDF

```
total = (SELECT ...);
IF (total > 1000000) THEN vip = True;
                          ELSE vip = False;
RETURN vip;
```

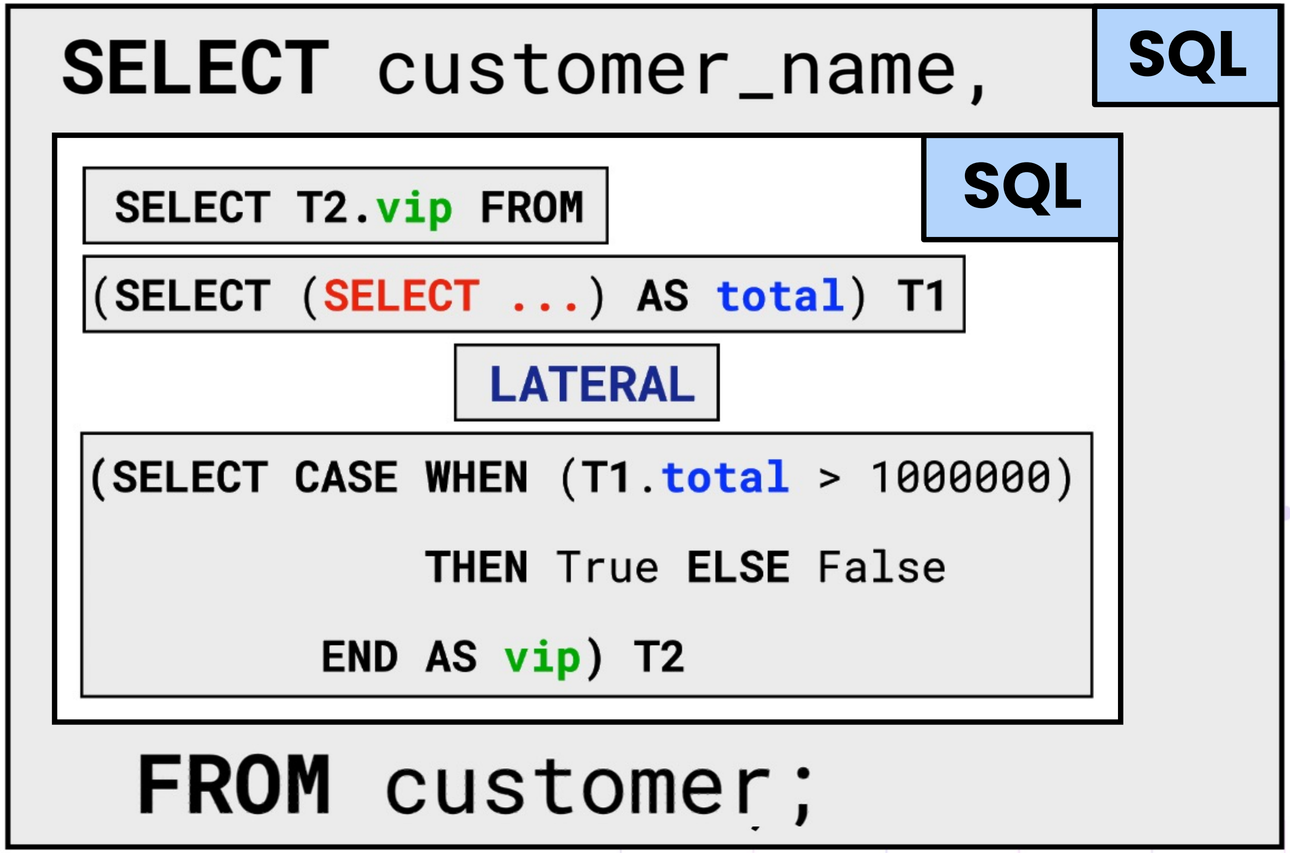
UDF Inlining

SQL

```
 SELECT T2.vip FROM
(SELECT (SELECT ...) AS total) T1
            LATERAL
(SELECT CASE WHEN (T1.total > 1000000)
             THEN True ELSE False
         END AS vip) T2
```

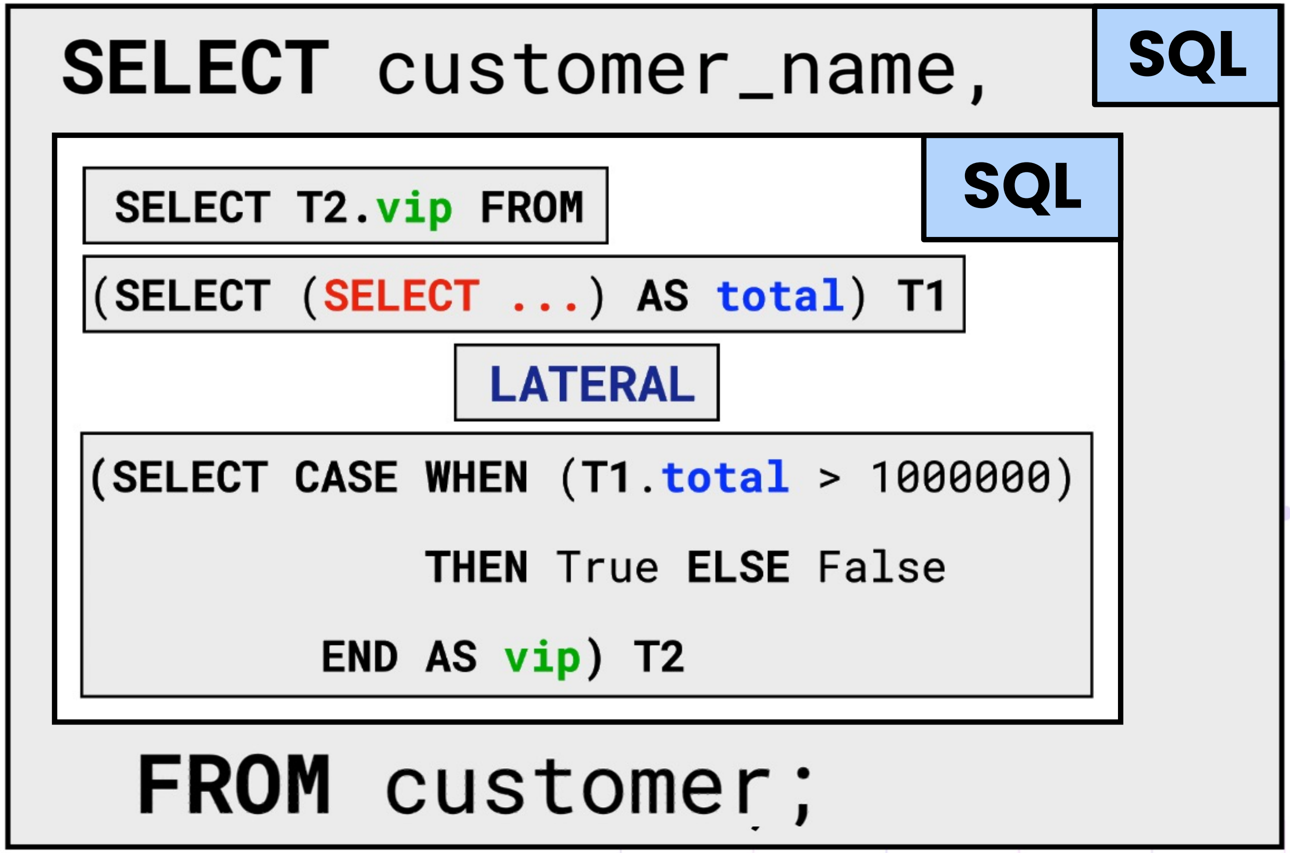
**Chain together translated UDF statements with LATERAL joins**

# The Problem with Inlining



```
SELECT customer_name,
  SELECT T2.vip FROM
  (SELECT (SELECT ...) AS total) T1
                LATERAL
  (SELECT CASE WHEN (T1.total > 1000000)
                THEN True ELSE False
          END AS vip) T2
    FROM customer;
```

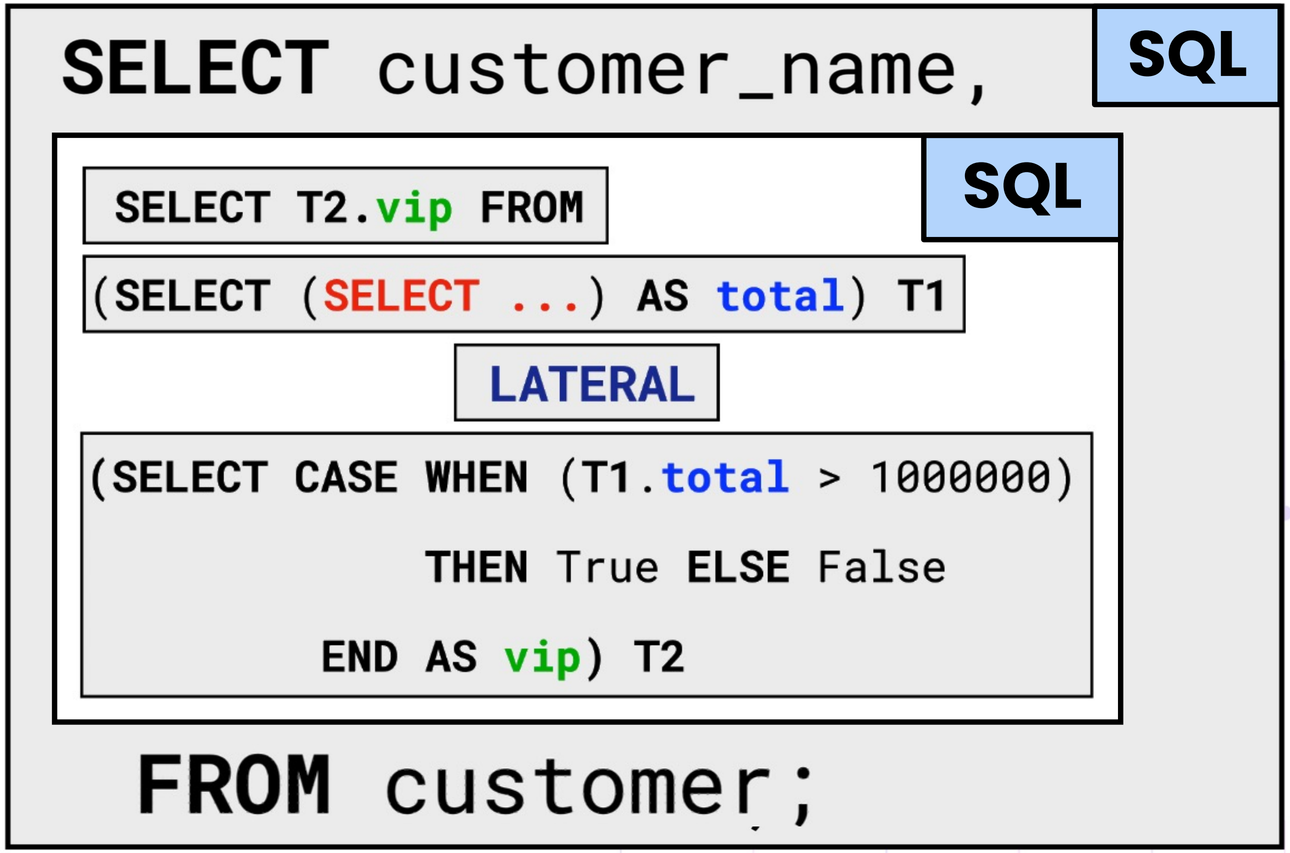
# The Problem with Inlining



```
SELECT customer_name,
  SELECT T2.vip FROM
  (SELECT (SELECT ...) AS total) T1
                LATERAL
  (SELECT CASE WHEN (T1.total > 1000000)
               THEN True ELSE False
           END AS vip) T2
  FROM customer;
```

**Complex subqueries!**

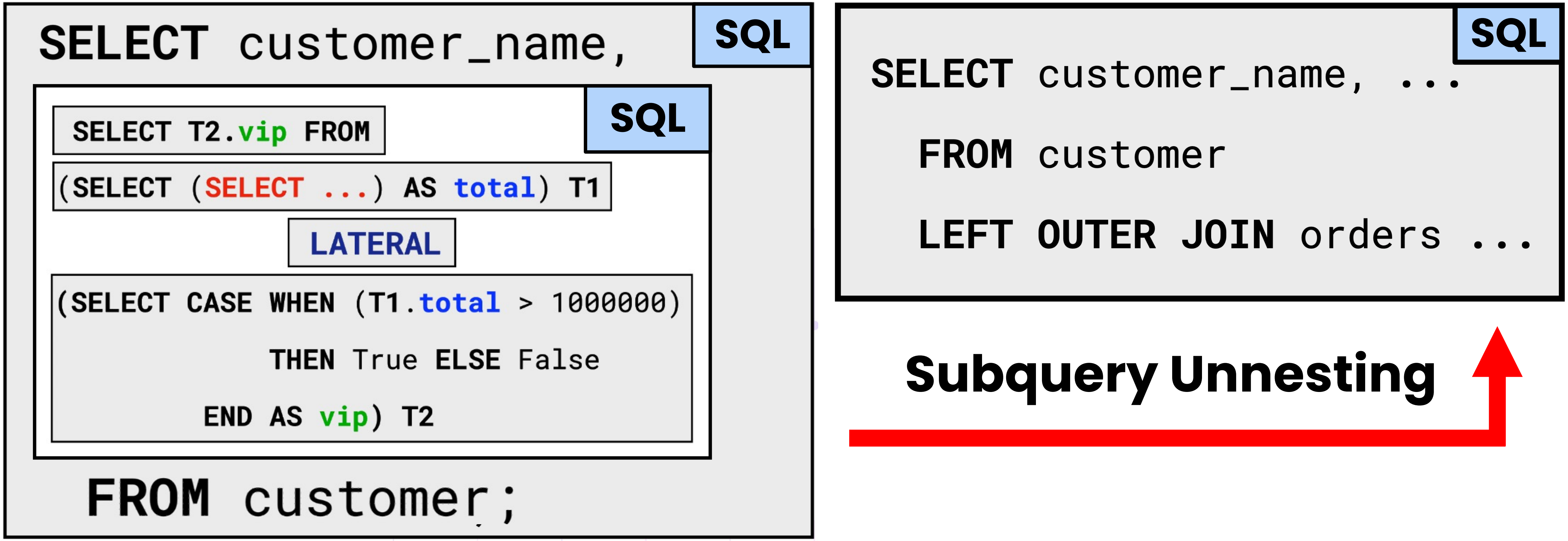
# The Problem with Inlining



```
SELECT customer_name,
  SELECT T2.vip FROM
  (SELECT (SELECT ...) AS total) T1
                LATERAL
  (SELECT CASE WHEN (T1.total > 1000000)
               THEN True ELSE False
           END AS vip) T2
  FROM customer;
```

**Complex subqueries!**

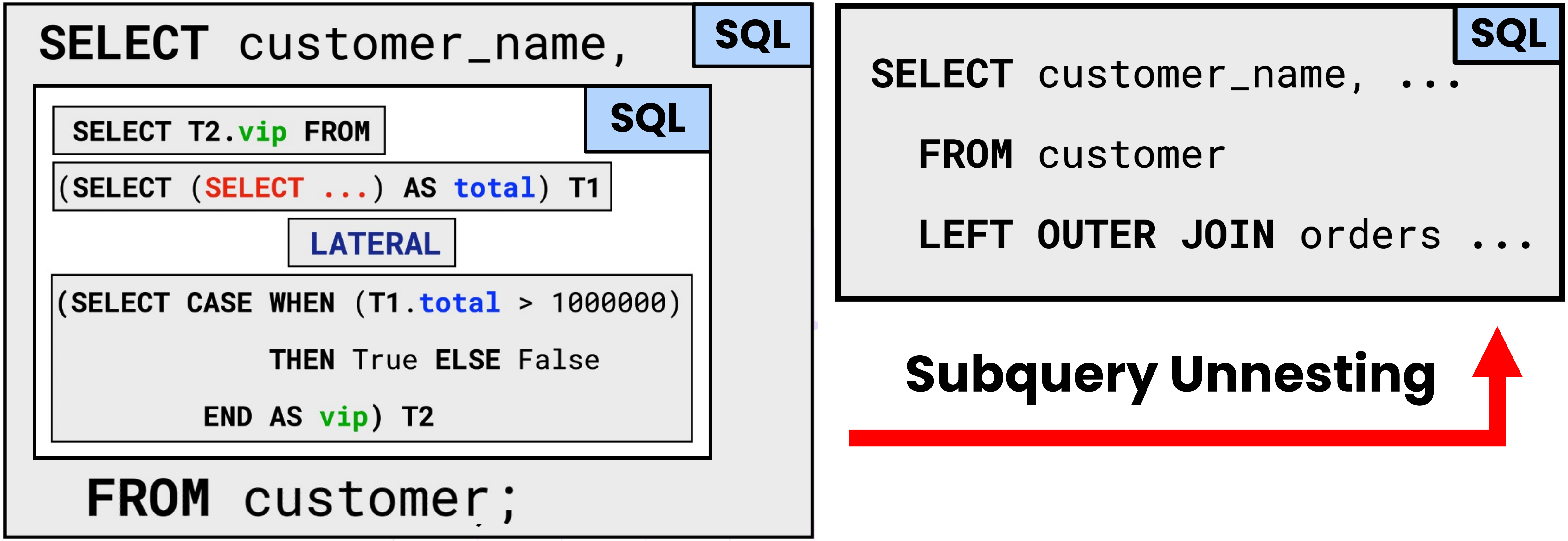
# The Problem with Inlining



```
SELECT customer_name,
  SELECT T2.vip FROM
  (SELECT (SELECT ...) AS total) T1
  LATERAL
  (SELECT CASE WHEN (T1.total > 1000000)
              THEN True ELSE False
          END AS vip) T2
  FROM customer;
```

```
SELECT customer_name, ...
  FROM customer
  LEFT OUTER JOIN orders ...
```

# The Problem with Inlining



```
SELECT customer_name,
    SELECT T2.vip FROM
    (SELECT (SELECT ...) AS total) T1
                LATERAL
    (SELECT CASE WHEN (T1.total > 1000000)
               THEN True ELSE False
          END AS vip) T2
  FROM customer;
```

```
SELECT customer_name, ...
  FROM customer
  LEFT OUTER JOIN orders ...
```

**Inlining generates complex subqueries that database systems can't unnest!**

# The Problem with Inlining

| | | ProcBench Queries | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Technique | Q1 | Q5 | Q6 | Q7 | Q9a | Q9b | Q12 | Q13 | Q15 | Q17 | Q18 | Q20a1 | Q20a2 | Q20b1 | Q20b2 |
| SQL Server | Inlining | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |

**Inlining: 4/15**

**Inlining generates complex subqueries that database systems can't unnest!**

# Why Inline the Entire UDF?

UDF

```
total = (SELECT ...);

IF (total > 1000000) THEN vip = True;

                     ELSE vip = False;

RETURN vip;
```

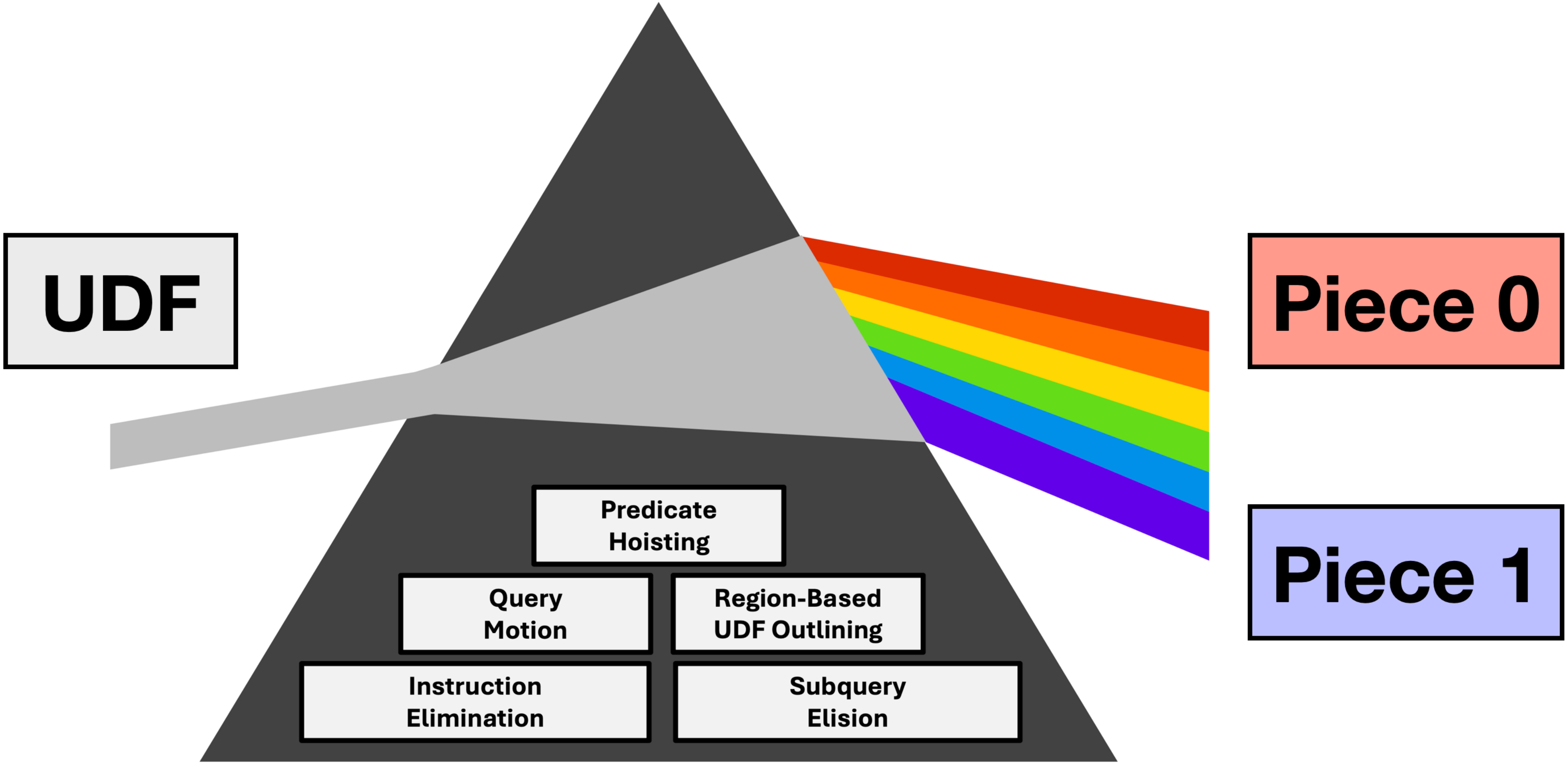
# Why Inline the Entire UDF?

UDF

```
total = (SELECT ...);

IF (total > 1000000) THEN vip = True;
                     ELSE vip = False;

RETURN vip;
```

**Can we "outline" irrelevant UDF code?**

# PRISM: A UDF Optimizing Compiler



# PRISM: A UDF Optimizing Compiler

**P**redicate Hoisting

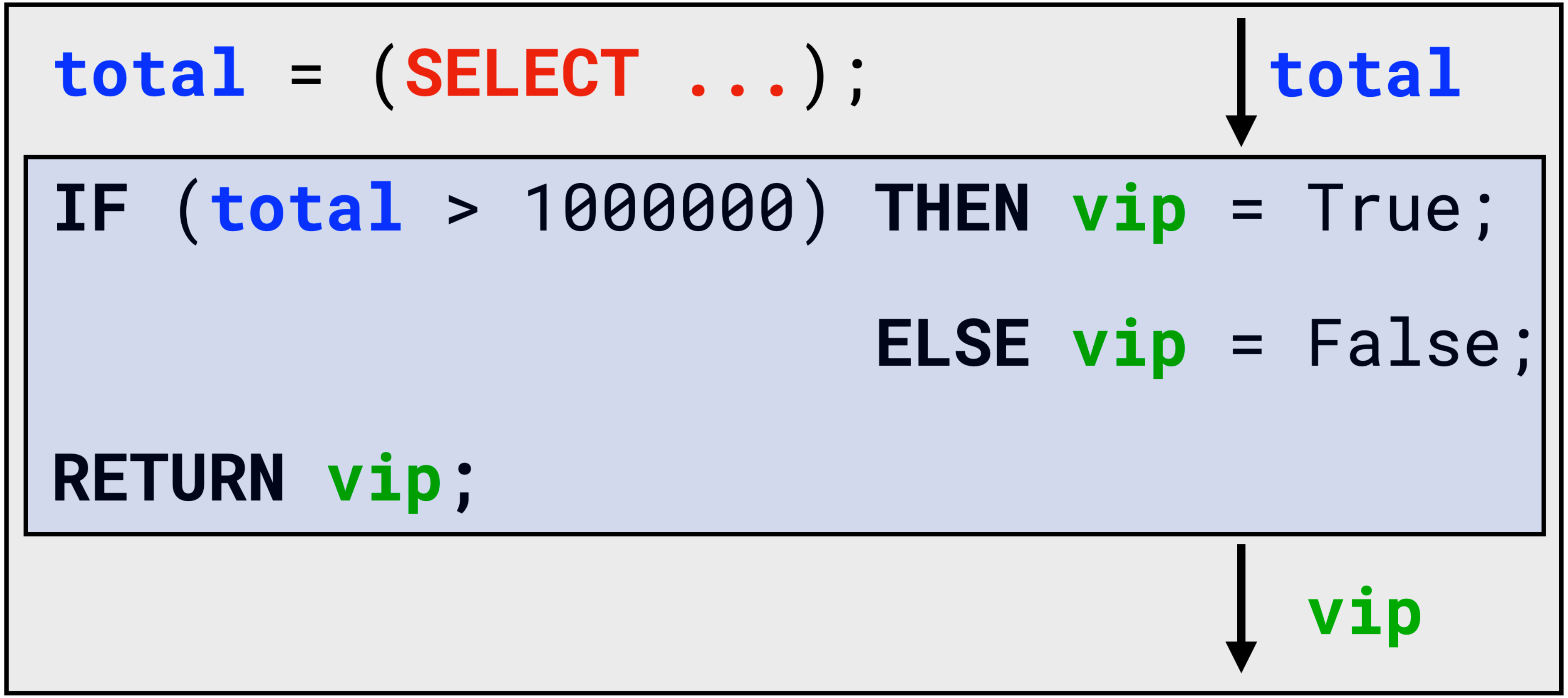
**R**egion-Based UDF Outlining

**I**nstruction Elimination

**S**ubquery Elision

Query **M**otion

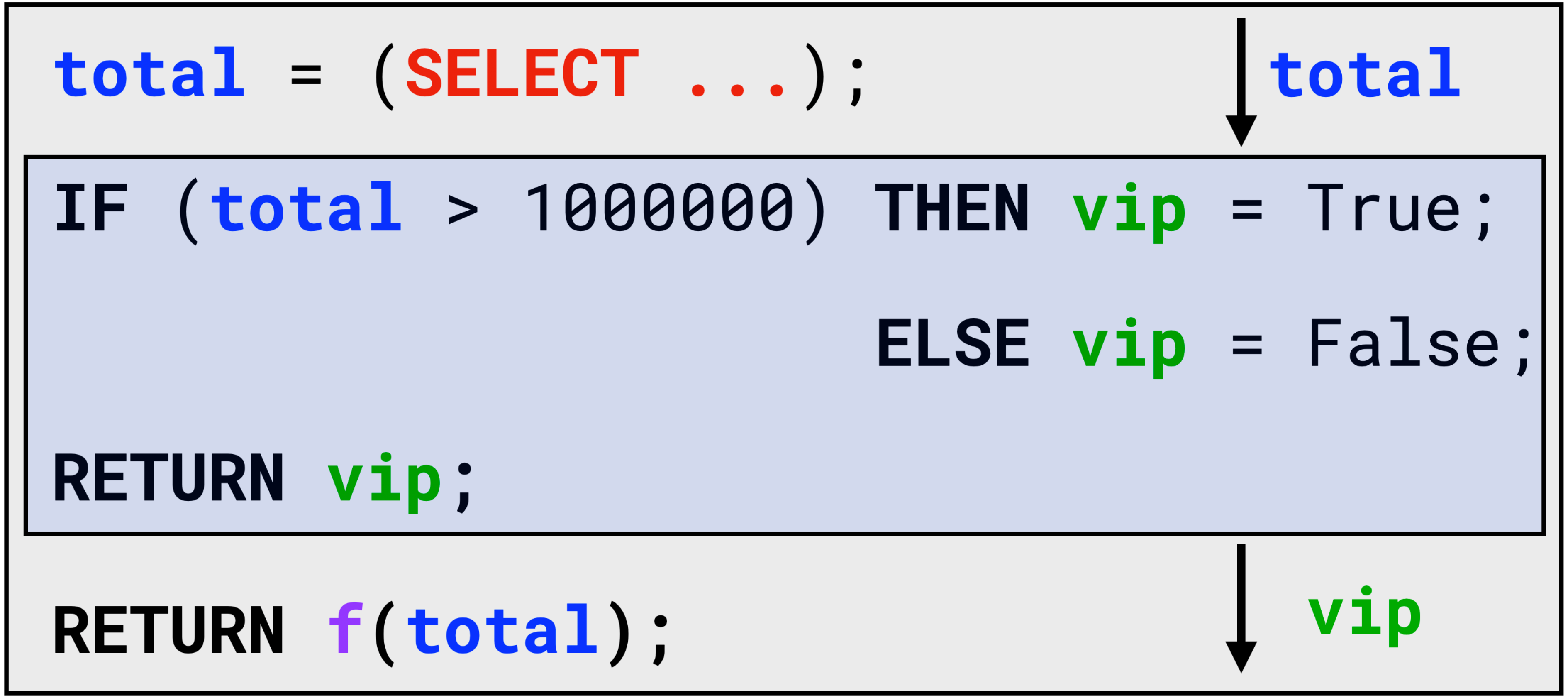
# Region-Based UDF Outlining



```
total = (SELECT ...);

IF (total > 1000000) THEN vip = True;

                      ELSE vip = False;

RETURN vip;
```

**Identify regions of irrelevant UDF code**

# Region-Based UDF Outlining



```
total = (SELECT ...);
IF (total > 1000000) THEN vip = True;
                     ELSE vip = False;
RETURN vip;
RETURN f(total);
```

**Outline regions into separate functions**

# Region-Based UDF Outlining

```
total = (SELECT ...);




RETURN f(total);
```

**Replace regions with function calls**

# Instruction Elimination

```
total = (SELECT ...);

RETURN f(total);
```

**Eliminate redundant instructions**

# Instruction Elimination

```
RETURN f(SELECT ...);
```

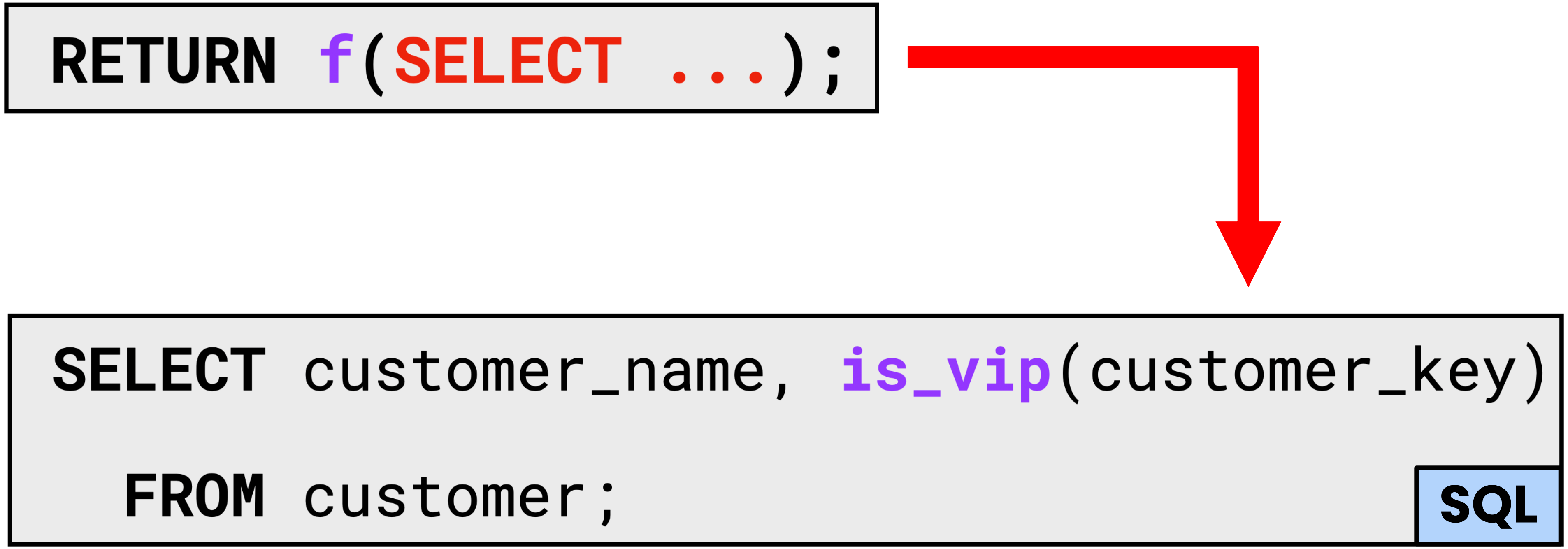
**Eliminate redundant instructions**

# Instruction Elimination

```
RETURN f(SELECT ...);
```

**Eliminate redundant instructions**

# Subquery Elision



```
RETURN f(SELECT ...);
```

```sql
SELECT customer_name, is_vip(customer_key)
  FROM customer;
```

**Directly inject the UDF's return value**

# Subquery Elision

```
RETURN f(SELECT ...);
```

```
SELECT customer_name, f(SELECT ...)
  FROM customer;
```

SQL

**Directly inject the UDF's return value**

# Subquery Elision

```sql
SELECT customer_name, f(SELECT ...)
  FROM customer;
```

**Directly inject the UDF's return value**

# Inlining vs PRISM

```
SELECT customer_name,                                SQL
                                              SQL
  SELECT T2.vip FROM
  (SELECT (SELECT ...) AS total) T1
                 LATERAL
  (SELECT CASE WHEN (T1.total > 1000000)
               THEN True ELSE False
       END AS vip) T2
  FROM customer;
```

```
SELECT customer_name, f(SELECT ...)
  FROM customer;                                      SQL
```

**PRISM generates simpler, faster queries**

# Evaluation

**Ran PRISM-optimized UDFs in DuckDB & SQL Server**

# Evaluation

Ran PRISM-optimized UDFs in DuckDB & SQL Server

Ran the ProcBench @ 10GB Scale Factor (TPC-DS + UDFs)

**VLDB 2021**

**Procedural Extensions of SQL: Understanding Their Usage in the Wild**

# Evaluation

Ran PRISM-optimized UDFs in DuckDB & SQL Server

Ran the ProcBench @ 10GB Scale Factor (TPC-DS + UDFs)

Intel Xeon Gold 5218R CPU, 192GB DDR4 RAM, 960GB NVMe SSD

Microsoft

**VLDB 2021**

**Procedural Extensions of SQL: Understanding Their Usage in the Wild**

# Evaluation

Ran PRISM-optimized UDFs in DuckDB & SQL Server

Ran the ProcBench @ 10GB Scale Factor (TPC-DS + UDFs)

Intel Xeon Gold 5218R CPU, 192GB DDR4 RAM, 960GB NVMe SSD

Built Column-Store Indexes on SQL Server

**VLDB 2021**

**Procedural Extensions of SQL: Understanding Their Usage in the Wild**

# Evaluation

Ran PRISM-optimized UDFs in DuckDB & SQL Server

Ran the ProcBench @ 10GB Scale Factor (TPC-DS + UDFs)

Intel Xeon Gold 5218R CPU, 192GB DDR4 RAM, 960GB NVMe SSD

Built Column-Store Indexes on SQL Server

2x Cold Runs and 5x Hot Runs

Microsoft

**VLDB 2021**

**Procedural Extensions of SQL: Understanding Their Usage in the Wild**

# Evaluation (Unnesting)

| | | ProcBench Queries | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Technique | Q1 | Q5 | Q6 | Q7 | Q9a | Q9b | Q12 | Q13 | Q15 | Q17 | Q18 | Q20a1 | Q20a2 | Q20b1 | Q20b2 |
| SQL Server | Inlining | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |

**Inlining: 4/15**

# Evaluation (Unnesting)

| | Technique | ProcBench Queries | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Q1 | Q5 | Q6 | Q7 | Q9a | Q9b | Q12 | Q13 | Q15 | Q17 | Q18 | Q20a1 | Q20a2 | Q20b1 | Q20b2 |
| SQL Server | Inlining | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| | PRISM | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ |

**Inlining: 4/15**

**PRISM: 12/15**

# Evaluation (Unnesting)

| | | ProcBench Queries | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Technique** | Q1 | Q5 | Q6 | Q7 | Q9a | Q9b | Q12 | Q13 | Q15 | Q17 | Q18 | Q20a1 | Q20a2 | Q20b1 | Q20b2 |
| SQL Server | **Inlining** | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ | ✗ | ✗ | ✗ | ✓ | ✓ | ✗ | ✗ |
| | **PRISM** | ✓ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ | ✓ | ✗ | ✓ | ✗ | ✓ | ✓ | ✓ | ✓ |
| DuckDB | **Inlining** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | **PRISM** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

**Inlining: 4/15**

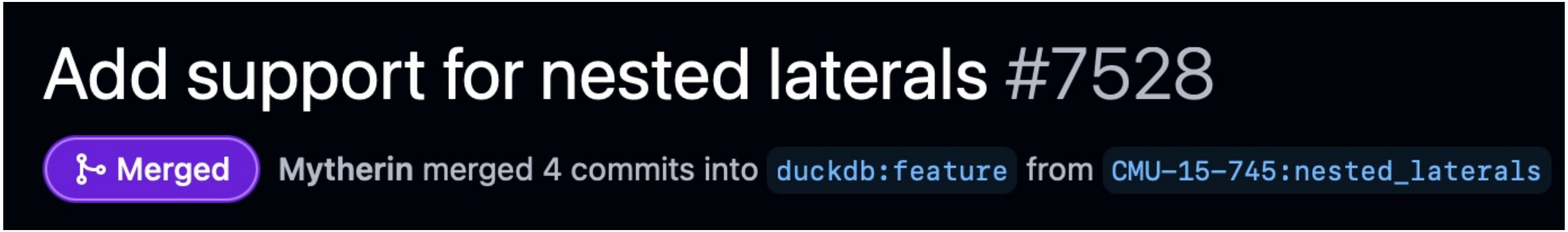
**PRISM: 12/15**

**DuckDB: 15/15**

**<u>BTW 2015</u>**

**Unnesting Arbitrary Queries**

# Evaluation (Unnesting)



Add support for nested laterals #7528

Merged Mytherin merged 4 commits into `duckdb:feature` from `CMU-15-745:nested_laterals`

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| DuckDB | Inlining | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | PRISM | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

**Inlining: 4/15**

**PRISM: 12/15**

**DuckDB: 15/15**

**<u>BTW 2015</u>**

**Unnesting Arbitrary Queries**

# Evaluation (Overall Speedup)

| | Average Speedup | Maximum Speedup |
|---|---|---|
| SQL Server | 298.7× | |
| DuckDB | | |

**Unnesting**

# Evaluation (Overall Speedup)

| | Average Speedup | Maximum Speedup |
|---|---|---|
| SQL Server | 298.7× | 2997.9× |
| DuckDB | | |

Unnesting

# Evaluation (Overall Speedup)

| | Average Speedup | Maximum Speedup |
|---|---|---|
| SQL Server | 298.7× | 2997.9× |
| DuckDB | 1.3× | |

Fewer LATERAL Joins

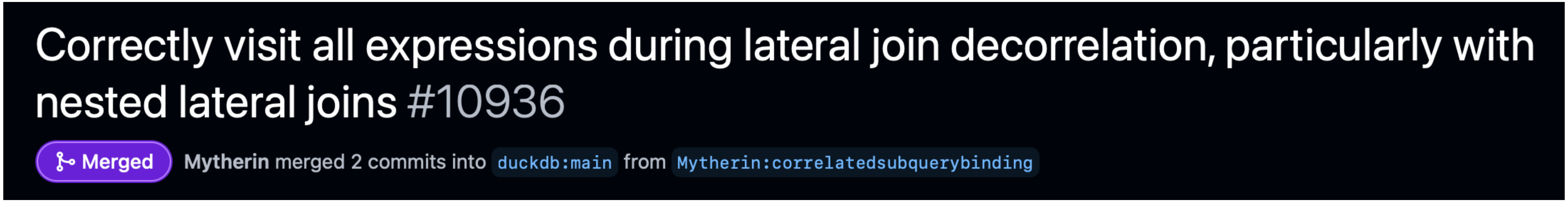
# Evaluation (Overall Speedup)

| | Average Speedup | Maximum Speedup |
|---|---|---|
| SQL Server | 298.7× | 2997.9× |
| DuckDB | 1.3× | 2270.2× |

Better Query Plan

# Evaluation (Overall Speedup)

| | Average Speedup | Maximum Speedup |
|---|---|---|
| SQL Server | 298.7× | 2997.9× |
| DuckDB | 1.3× | 2270.2× |



Correctly visit all expressions during lateral join decorrelation, particularly with nested lateral joins #10936

Mytherin merged 2 commits into `duckdb:main` from `Mytherin:correlatedsubquerybinding`

# Evaluation (Overall Speedup)

| | Average Speedup | Maximum Speedup |
|---|---|---|
| SQL Server | 298.7× | 2997.9× |
| DuckDB | 1.3× | 18.0× |

**Compiled Loop vs Recursive SQL**

# Conclusion

# Conclusion

**Inlining leads to sub-optimal performance**

# Conclusion

**Inlining leads to sub-optimal performance**

**Inlining entire UDFs creates complex queries**

# Conclusion

**Inlining leads to sub-optimal performance**

**Inlining entire UDFs creates complex queries**

**Instead, we inline only the important pieces of a UDF**

# Conclusion

Inlining leads to sub-optimal performance

Inlining entire UDFs creates complex queries

Instead, we inline only the important pieces of a UDF

We achieve this through UDF outlining with PRISM

# Conclusion

Inlining leads to sub-optimal performance

Inlining entire UDFs creates complex queries

Instead, we inline only the important pieces of a UDF

We achieve this through UDF outlining with PRISM

Our approach outperforms inlining by more than 1000x

# Future Work

**Python UDFs.**

samarch.xyz