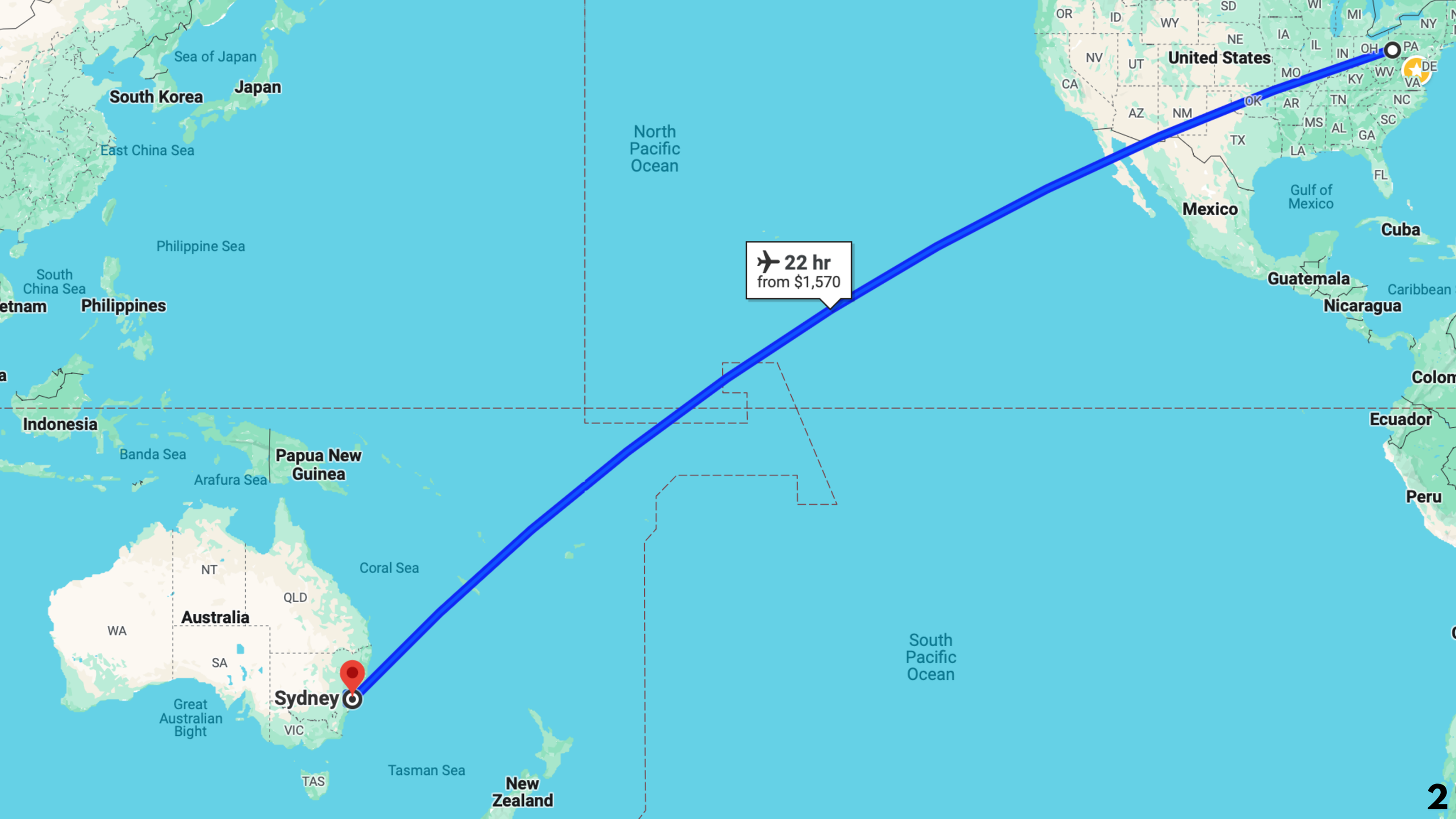
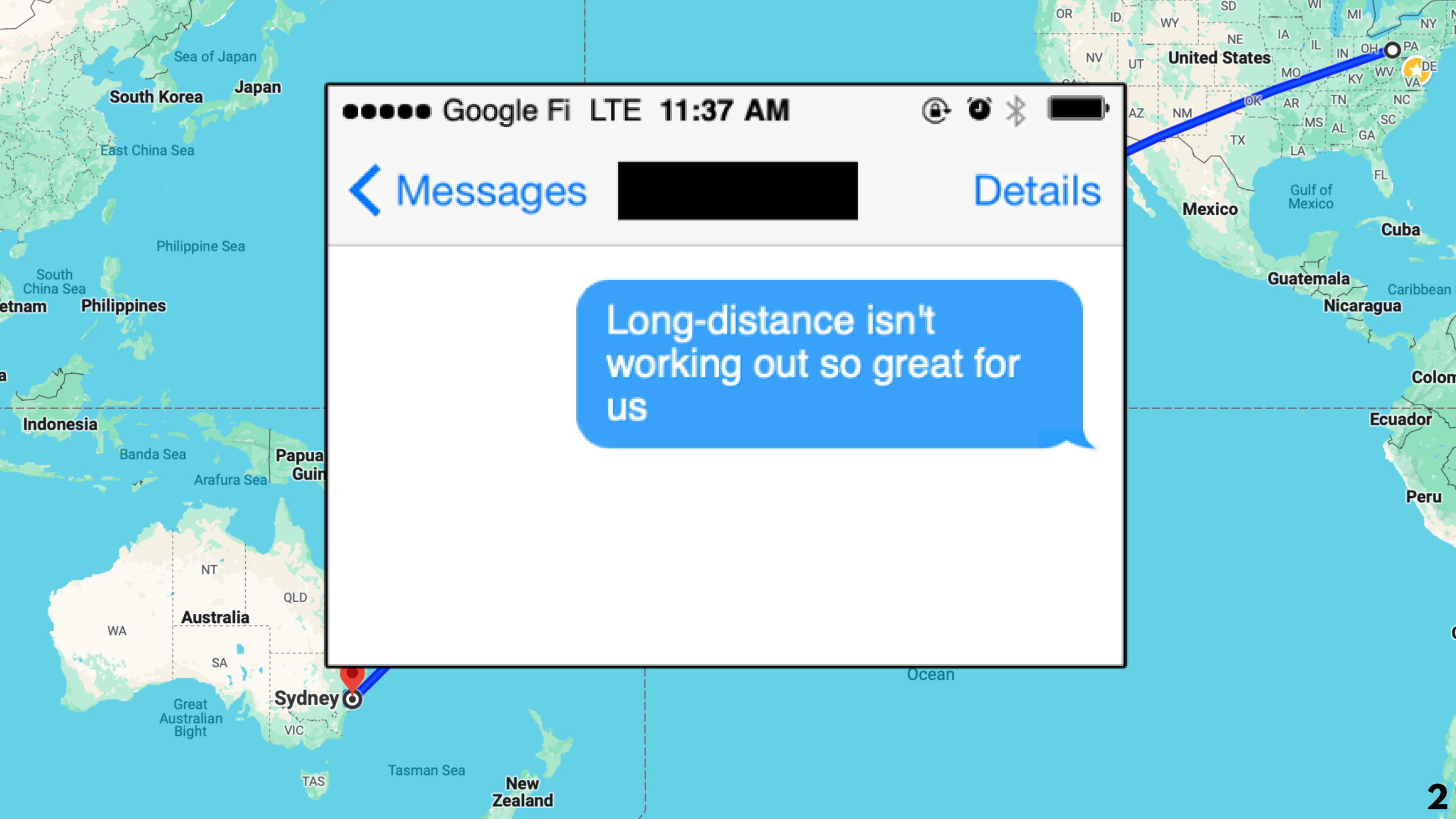
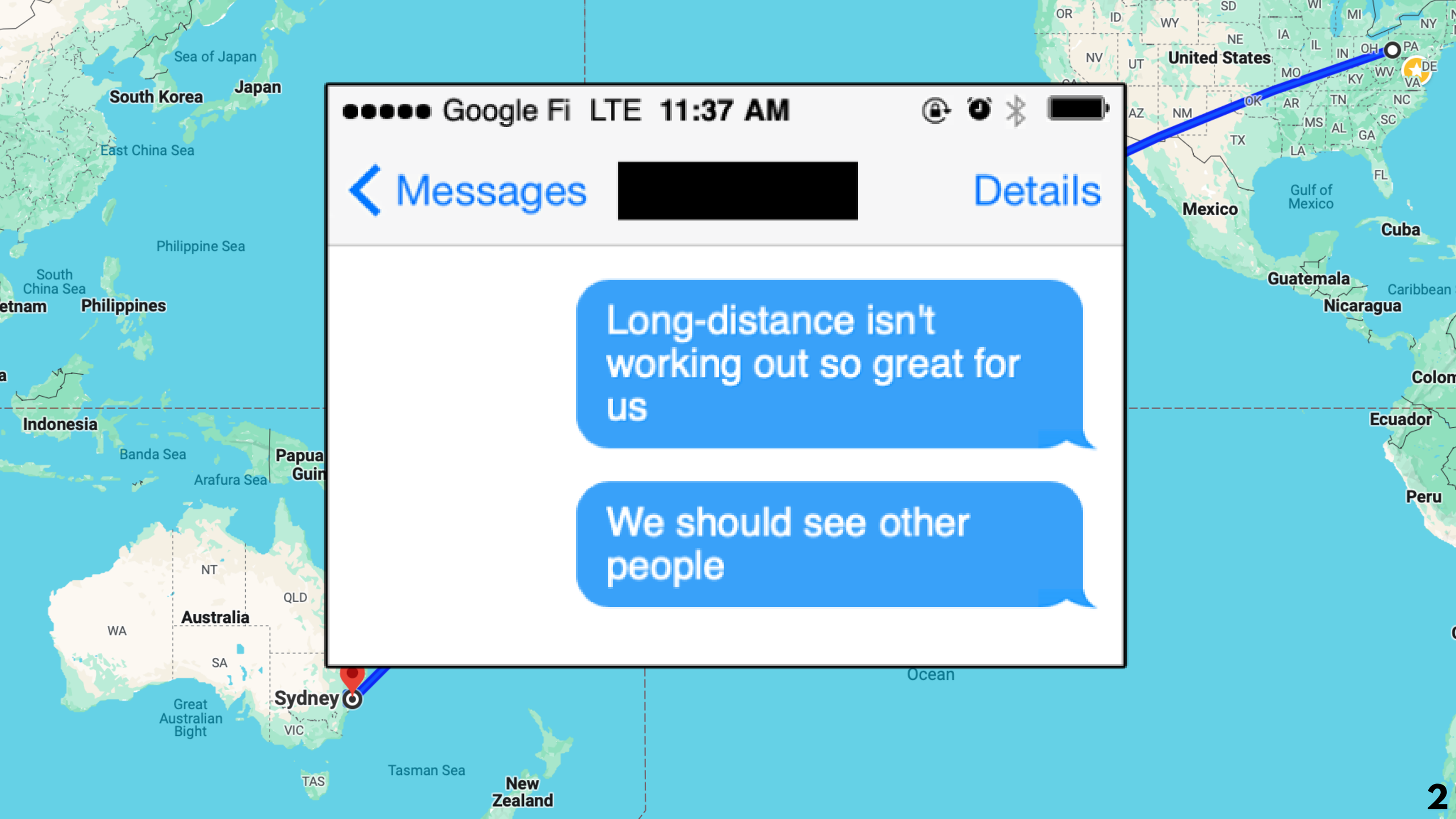
Dear User-Defined Functions,
Inlining isn't working out so great for us.
Let's try batching to make our relationship work.
Sincerely, SQL

Kai Franz, Sam Arch, Denis Hirn,
Torsten Grust, Todd Mowry, Andy Pavlo


22 hr
from $1,570
Sydney
Australia
United States
Mexico
Gulf of Mexico
Cuba
Guatemala
Nicaragua
Caribbean
Ecuador
Peru
Colom
Indonesia
Philippines
South China Sea
Philippine Sea
East China Sea
Sea of Japan
South Korea
Japan
North Pacific Ocean
South Pacific Ocean
Coral Sea
Tasman Sea
Banda Sea
Arafura Sea
Papua New Guinea
New Zealand
Great Australian Bight
NT
QLD
WA
SA
VIC
TAS
OR
ID
WY
SD
WI
MI
NY
NE
IA
IL
IN
OH
PA
DE
NV
UT
CA
AZ
NM
CO
MO
KY
WV
VA
OK
AR
TN
NC
MS
AL
SC
GA
TX
LA
FL


Google Fi LTE 11:37 AM
Messages
[redacted]
Details
Long-distance isn't working out so great for us
Sea of Japan
South Korea
Japan
East China Sea
Philippine Sea
South China Sea
Philippines
Indonesia
Banda Sea
Arafura Sea
Papua Guin
Australia
NT
QLD
WA
SA
VIC
TAS
Great Australian Bight
Sydney
Tasman Sea
New Zealand
Ocean
United States
Mexico
Gulf of Mexico
Cuba
Guatemala
Nicaragua
Caribbean
Ecuador
Peru
Colom
OR
ID
WY
SD
NE
IA
IL
MI
WI
NY
PA
OH
IN
NV
UT
CO
MO
KY
WV
VA
DE
AZ
NM
OK
AR
TN
NC
MS
AL
SC
GA
TX
LA
FL


Google Fi LTE 11:37 AM
Messages
[redacted]
Details
Long-distance isn't working out so great for us
We should see other people
Sea of Japan
South Korea
Japan
East China Sea
Philippine Sea
South China Sea
Philippines
Indonesia
Banda Sea
Arafura Sea
Papua Guin
NT
QLD
Australia
WA
SA
VIC
Great Australian Bight
Sydney
Tasman Sea
TAS
New Zealand
Ocean
United States
OR
ID
WY
SD
WI
MI
NY
NE
IA
IL
IN
OH
PA
NV
UT
MO
KY
WV
VA
DE
AZ
NM
OK
AR
TN
NC
MS
AL
SC
GA
TX
LA
FL
Gulf of Mexico
Mexico
Cuba
Guatemala
Caribbean
Nicaragua
Colom
Ecuador
Peru

Summary

Summary

UDF Inlining is #1 for perf

Summary

UDF Inlining is #1 for perf

We found major problems with it

Summary

UDF Inlining is #1 for perf

We found major problems with it

UDF Batching is another technique

Summary

UDF Inlining is #1 for perf

We found major problems with it

UDF Batching is another technique

We compared them on 4 DBMSs

Summary

UDF Inlining is #1 for perf

We found major problems with it

UDF Batching is another technique

We compared them on 4 DBMSs

A hybrid strategy gives the best perf

UDFs

```
SELECT c_name,
       service_level(c_custkey)
  FROM customer;
```

PL/SQL UDFs

```
SELECT c_name,
       service_level(c_custkey)
  FROM customer;
```

```
CREATE FUNCTION service_level(ckey int)
RETURNS char(10)
LANGUAGE plpgsql
AS
BEGIN
    DECLARE total float;
    DECLARE service_level char(10);

    SELECT total = SUM(o_total_price)
      FROM orders
     WHERE o_custkey = ckey;

    IF (total > 1000000) THEN
        service_level = 'Platinum';
    ELSE IF (total > 500000) THEN
        service_level = 'Gold';
    ELSE
        service_level = 'Regular';

    RETURN service_level;
END;
```

PL/SQL UDFs

```
SELECT c_name,
       service_level(c_custkey)
  FROM customer;
```

Code re-use ✓

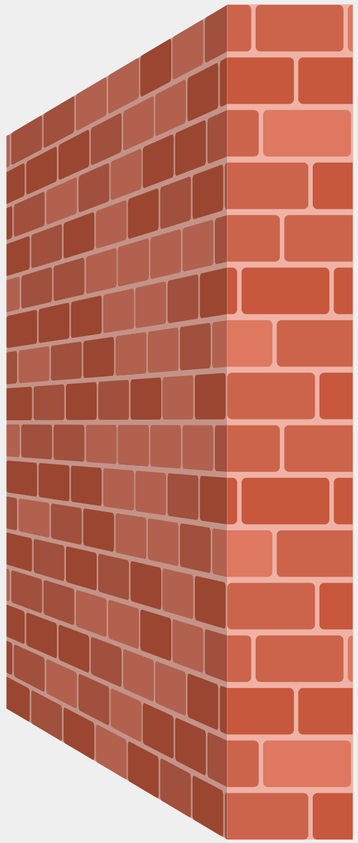
Intuitive ✓

Billions of daily queries

```
CREATE FUNCTION service_level(ckey int)
RETURNS char(10)
LANGUAGE plpgsql
AS
BEGIN
    DECLARE total float;
    DECLARE service_level char(10);

    SELECT total = SUM(o_total_price)
      FROM orders
     WHERE o_custkey = ckey;

    IF (total > 1000000) THEN
        service_level = 'Platinum';
    ELSE IF (total > 500000) THEN
        service_level = 'Gold';
    ELSE
        service_level = 'Regular';

    RETURN service_level;
END;
```

UDFs are optimization barriers

SQL



UDF

UDFs are slow!

```
SELECT c_name,
       service_level(c_custkey)
  FROM customer;
```

```
CREATE FUNCTION service_level(ckey int)
RETURNS char(10)
LANGUAGE plpgsql
AS
BEGIN
    DECLARE total float;
    DECLARE service_level char(10);

    SELECT total = SUM(o_total_price)
      FROM orders
     WHERE o_custkey = ckey;

    IF (total > 1000000) THEN
        service_level = 'Platinum';
    ELSE IF (total > 500000) THEN
        service_level = 'Gold';
    ELSE
        service_level = 'Regular';

    RETURN service_level;
END;
```

UDFs are slow!

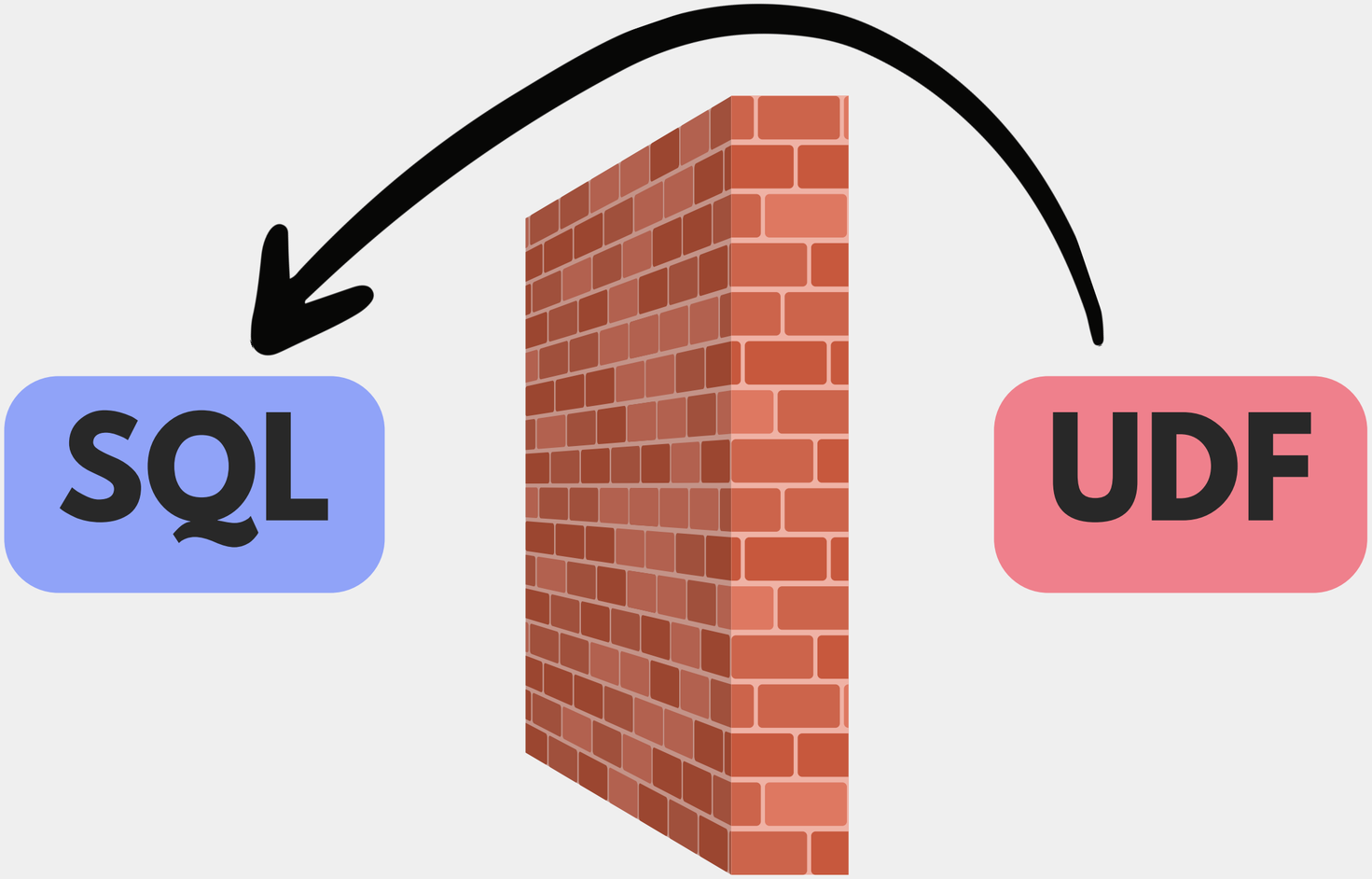
```
SELECT c_name,
       service_level(c_custkey)
  FROM customer;
```

Bad query plans!

Painfully slow!

```
CREATE FUNCTION service_level(ckey int)
RETURNS char(10)
LANGUAGE plpgsql
AS
BEGIN
    DECLARE total float;
    DECLARE service_level char(10);

    SELECT total = SUM(o_total_price)
      FROM orders
     WHERE o_custkey = ckey;

    IF (total > 1000000) THEN
        service_level = 'Platinum';
    ELSE IF (total > 500000) THEN
        service_level = 'Gold';
    ELSE
        service_level = 'Regular';

    RETURN service_level;
END;
```

UDF Inlining



UDF Inlining

Decorrelation of User Defined Function Invocations in Queries

Varun Simhadri [#1], Karthik Ramachandra [*2], Arun Chaitanya [#3], Ravindra Guravannavar [#4], S. Sudarshan [*5]

[#] *IIT Hyderabad, India* [6]

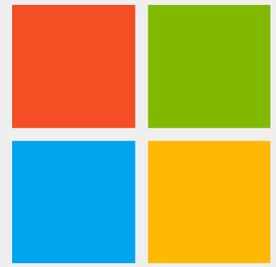
[1] varun.simhadri@netapp.com, [3] arun@worksap.co.jp, [4] ravig@acm.org

[*] *IIT Bombay, India*

[2] karthiksr@cse.iitb.ac.in, [5] sudarsha@cse.iitb.ac.in

IIT Bombay (2014)

FROID: UDF Inlining



Froid: Optimization of Imperative Programs in a Relational Database

Karthik Ramachandra
Microsoft Gray Systems Lab
karam@microsoft.com

Kwanghyun Park
Microsoft Gray Systems Lab
kwpark@microsoft.com

K. Venkatesh Emani*
IIT Bombay
venkateshek@cse.iitb.ac.in

Alan Halverson
Microsoft Gray Systems Lab
alanhal@microsoft.com

César Galindo-Legaria
Microsoft
cesarg@microsoft.com

Conor Cunningham
Microsoft
conorc@microsoft.com

Gray Systems Lab (2017)

FROID: UDF Inlining

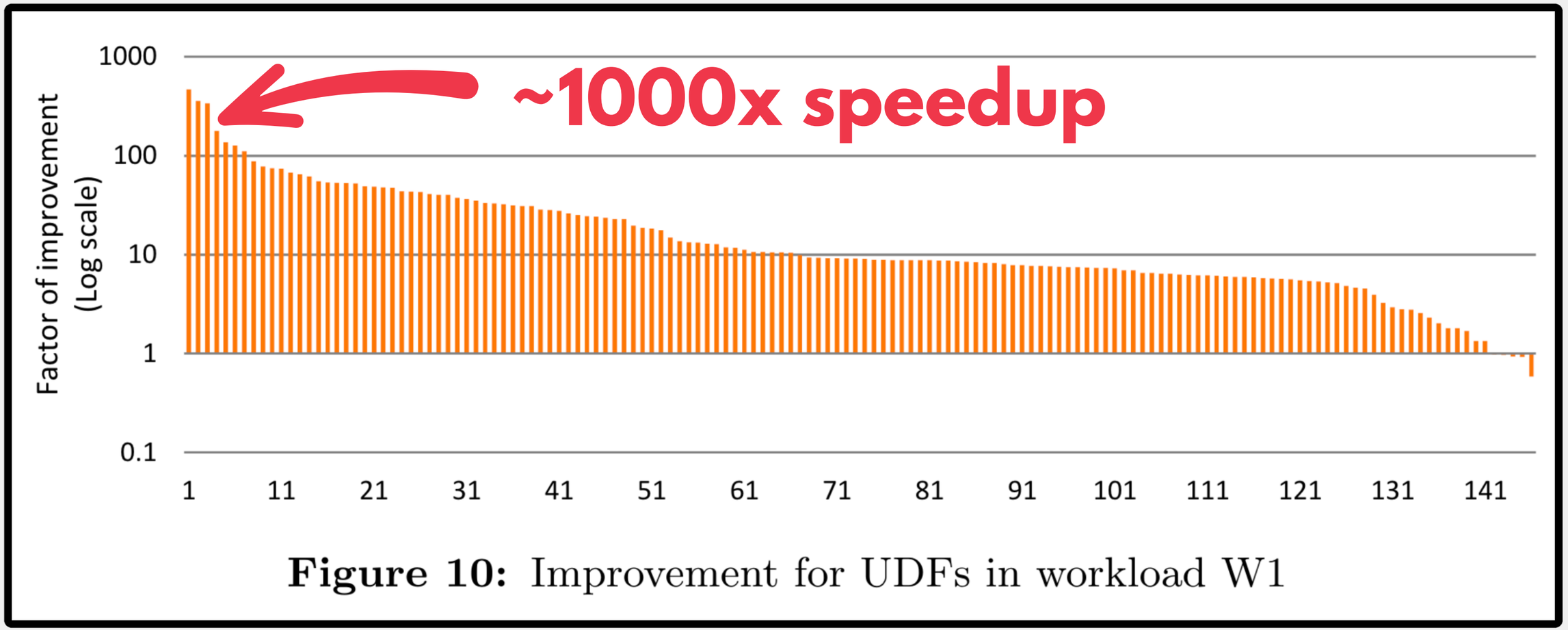


Figure 10: Improvement for UDFs in workload W1

UDF to SQL

UDF to SQL

`x = y`  `(SELECT y AS x) T1`

UDF to SQL

x = y

(SELECT y AS x) T1

x = (SELECT ...)

(SELECT (SELECT ...) AS x) T2

UDF to SQL

x = y

(SELECT y AS x) T1

x = (SELECT ...)

(SELECT (SELECT ...) AS x) T2

IF ... THEN ... ELSE ...

(CASE WHEN ... ELSE ...) T3

UDF to SQL

x = y

(SELECT y AS x) T1

LATERAL

x = (SELECT ...)

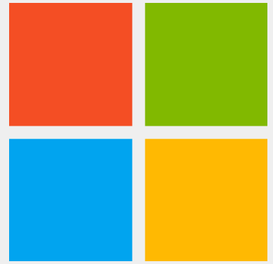
(SELECT (SELECT ...) AS x) T2

LATERAL

IF ... THEN ... ELSE ...

(CASE WHEN ... ELSE ...) T3

FROID (2017)



```
CREATE FUNCTION service_level(@ckey int)
RETURNS char(10) AS
BEGIN
    DECLARE @total float;
    DECLARE @level char(10);

    SELECT @total = SUM(o_totalprice) FROM orders WHERE o_custkey = @ckey;

    IF (@total > 1000000)
        SET @level = 'Platinum';
    ELSE IF (@total > 500000)
        SET @level = 'Gold';
    ELSE
        SET @level = 'Regular';

    RETURN @level;
END
```

```
SELECT (SELECT NULL AS @total) T1
```

```
SELECT (SELECT SUM(...) AS @total) T2
```

```
SELECT (SELECT
(CASE WHEN  T2.@total  > 1000000 THEN 'Platinum'
          WHEN  T2.@total  >  500000  THEN ‘Gold’
          ELSE ‘Regular’
END) AS @level) T3
```

```
SELECT (SELECT NULL AS @total) T1
```

LATERAL

```
SELECT (SELECT SUM(...) AS @total) T2
```

LATERAL

```
SELECT (SELECT
(CASE WHEN  T2.@total  > 1000000 THEN 'Platinum'
          WHEN  T2.@total  >  500000  THEN ‘Gold’
          ELSE ‘Regular’
END) AS @level) T3
```

```
SELECT c_name,
    service_level(c_custkey)
FROM customer;
```

```
SELECT c_name,
    service_level(c_custkey)
FROM customer;
```

Subquery

```
SELECT c_name,
(SELECT T3.@total FROM SELECT (SELECT NULL AS @total) T1,
LATERAL SELECT (SELECT SUM(...) AS @total) T2, LATERAL SELECT
(SELECT (CASE WHEN T2.@total > 1000000 THEN 'Platinum' WHEN
T2.@total > 500000 THEN ‘Gold’ ELSE ‘Regular’ END) AS @level) T3)
 FROM customer;
```

FROID (2017)

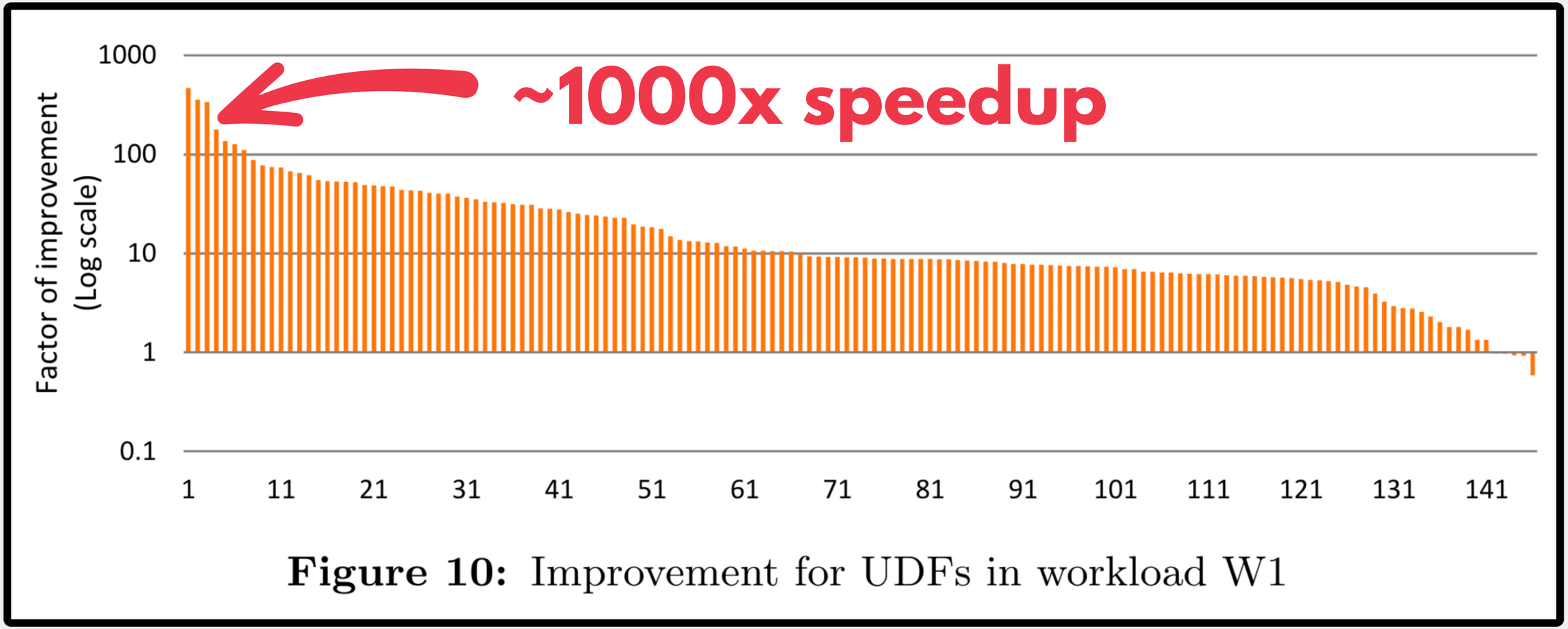


Figure 10: Improvement for UDFs in workload W1

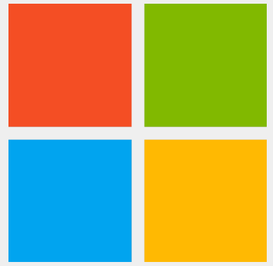
SQL ProcBench (2021)

Procedural Extensions of SQL:
Understanding their usage in the wild

Surabhi Gupta
Microsoft Research India
t-sugu@microsoft.com

Karthik Ramachandra
Microsoft Azure Data (SQL), India
karam@microsoft.com

SQL ProcBench (2021)



Procedural Extensions of SQL:
Understanding their usage in the wild

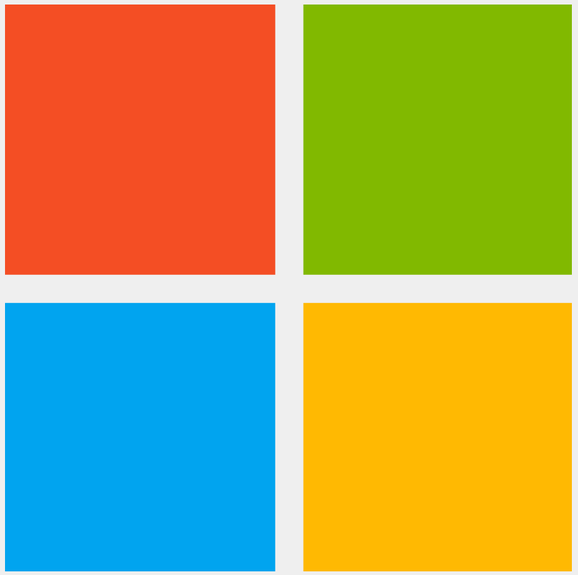
Surabhi Gupta
Microsoft Research India
t-sugu@microsoft.com

Karthik Ramachandra
Microsoft Azure Data (SQL), India
karam@microsoft.com

24 queries with UDFs ✓

Realistic ✓

ProcBench on SQL Server with FROID



SQL Server

```
SELECT c_name,
       service_level(c_custkey)
  FROM customer;
```

SLOW

vs.

```
SELECT c_name, CASE WHEN e.total > 1000000 THEN 'Platinum'
                    WHEN e.total > 500000  THEN 'Gold'
                                           ELSE 'Regular'
                     END
 FROM customer c
 LEFT OUTER JOIN (SELECT o_custkey, SUM(o_totalprice) AS total
                    FROM orders
                GROUP BY o_custkey) e
               ON c.c_custkey = e.o_custkey;
```

Fast

```
SELECT c_name,
    service_level(c_custkey)
FROM customer;
```

Subquery

```
SELECT c_name,
(SELECT T3.@total FROM SELECT (SELECT NULL AS @total) T1,
LATERAL SELECT (SELECT SUM(...) AS @total) T2, LATERAL SELECT
(SELECT (CASE WHEN T2.@total > 1000000 THEN 'Platinum' WHEN
T2.@total > 500000 THEN ‘Gold’ ELSE ‘Regular’ END) AS @level) T3)
 FROM customer;
```

Subquery Execution

(1) Evaluate subquery per row

Subquery Execution

(1) Evaluate subquery per row

```
SELECT (SELECT c_name
          FROM customer
         WHERE c_id = o_id)
  FROM orders;
```

Subquery Execution

(1) Evaluate subquery per row 

```
SELECT (SELECT c_name
          FROM customer
         WHERE c_id = o_id)
  FROM orders;
```

Subquery Execution

(1) Evaluate subquery per row 

(2) Replace subquery with join 

```
SELECT (SELECT c_name
          FROM customer
         WHERE c_id = o_id)
  FROM orders;
```

Subquery Execution

(1) Evaluate subquery per row

(2) Replace subquery with join

```
SELECT (SELECT c_name
          FROM customer
         WHERE c_id = o_id)
  FROM orders;
```

```
SELECT c_name
  FROM customer
  JOIN orders
ON c_id = o_id;
```

SQL Server Subqueries (2001)

Orthogonal Optimization of Subqueries and Aggregation

César A. Galindo-Legaria Milind M. Joshi

{cesarg,milindj}@microsoft.com
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052

SQL Server Subqueries (2001)

$$R\ \mathcal{A}^{\otimes}\ E = R \otimes_{\text{true}} E, \quad (1)$$

if no parameters in E resolved from R

$$R\ \mathcal{A}^{\otimes}\ (\sigma_p E) = R \otimes_p E, \quad (2)$$

if no parameters in E resolved from R

$$R\,\mathcal{A}^{\times}\ (\sigma_p E) = \sigma_p(R\,\mathcal{A}^{\times}\ E) \quad (3)$$

$$R\,\mathcal{A}^{\times}\ (\pi_v E) = \pi_{v\,\cup\,\text{columns}(R)}(R\,\mathcal{A}^{\times}\ E) \quad (4)$$

$$R\,\mathcal{A}^{\times}\ (E_1 \cup E_2) = (R\,\mathcal{A}^{\times}\ E_1) \cup (R\,\mathcal{A}^{\times}\ E_2) \quad (5)$$

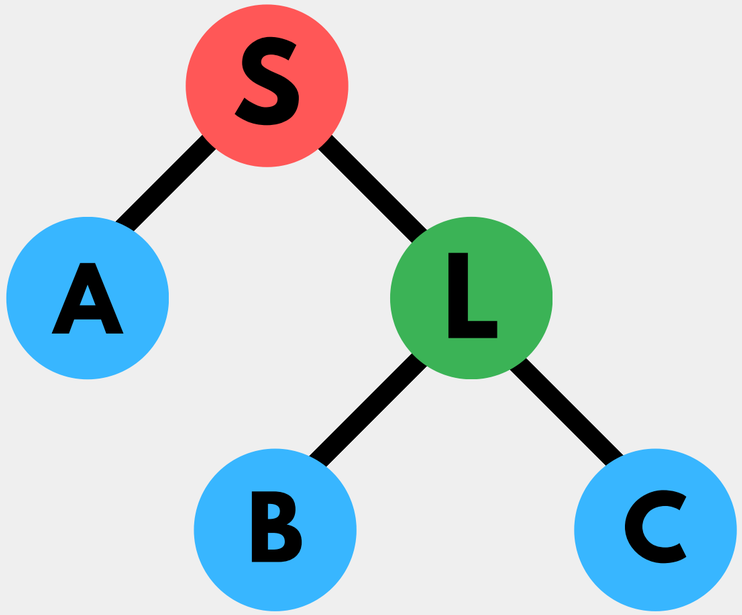
$$R\,\mathcal{A}^{\times}\ (E_1 - E_2) = (R\,\mathcal{A}^{\times}\ E_1) - (R\,\mathcal{A}^{\times}\ E_2) \quad (6)$$

$$R\,\mathcal{A}^{\times}\ (E_1 \times E_2) = (R\,\mathcal{A}^{\times}\ E_1) \bowtie_{R.key} (R\,\mathcal{A}^{\times}\ E_2) \quad (7)$$

$$R\,\mathcal{A}^{\times}\ (\mathcal{G}_{A,F} E) = \mathcal{G}_{A\,\cup\,\text{columns}(R),F}(R\,\mathcal{A}^{\times}\ E) \quad (8)$$

$$R\,\mathcal{A}^{\times}\ (\mathcal{G}^1_F E) = \mathcal{G}_{\text{columns}(R),F'}(R\,\mathcal{A}^{\text{LOJ}}\ E) \quad (9)$$

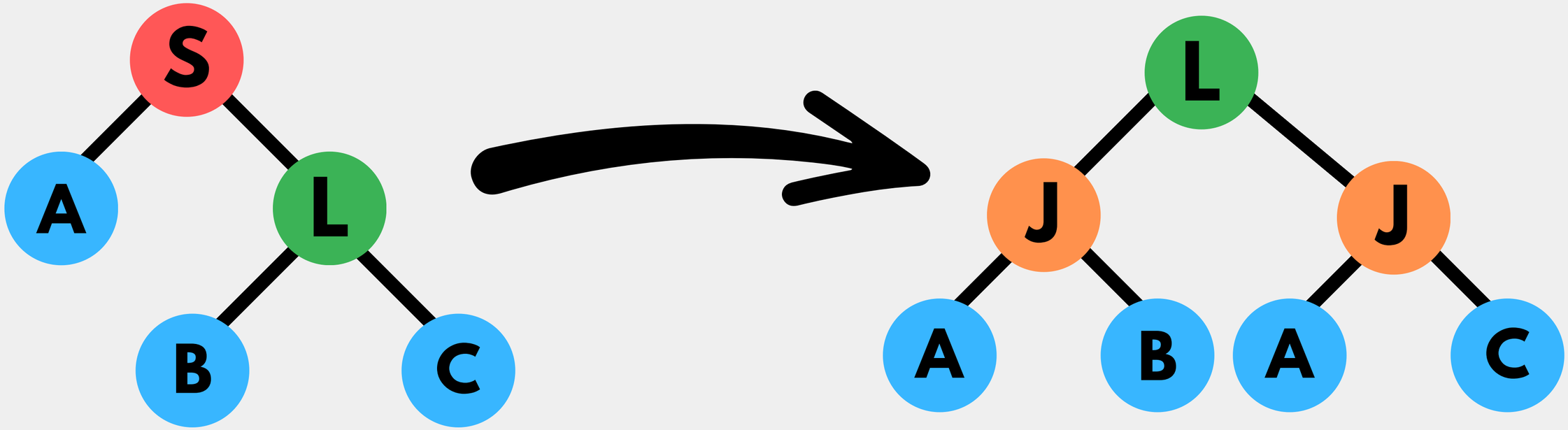
The Problem With LATERAL Joins



S = Subquery

L = LATERAL

The Problem With LATERAL Joins

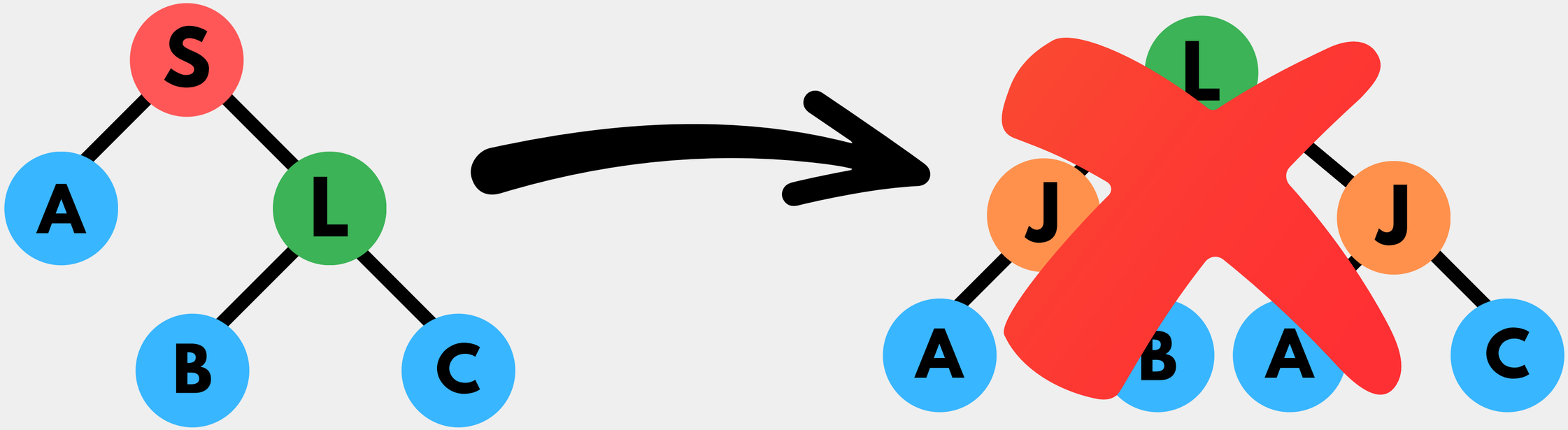


S = Subquery

L = LATERAL

J = Join

The Problem With LATERAL Joins



S = Subquery

L = LATERAL

J = Join

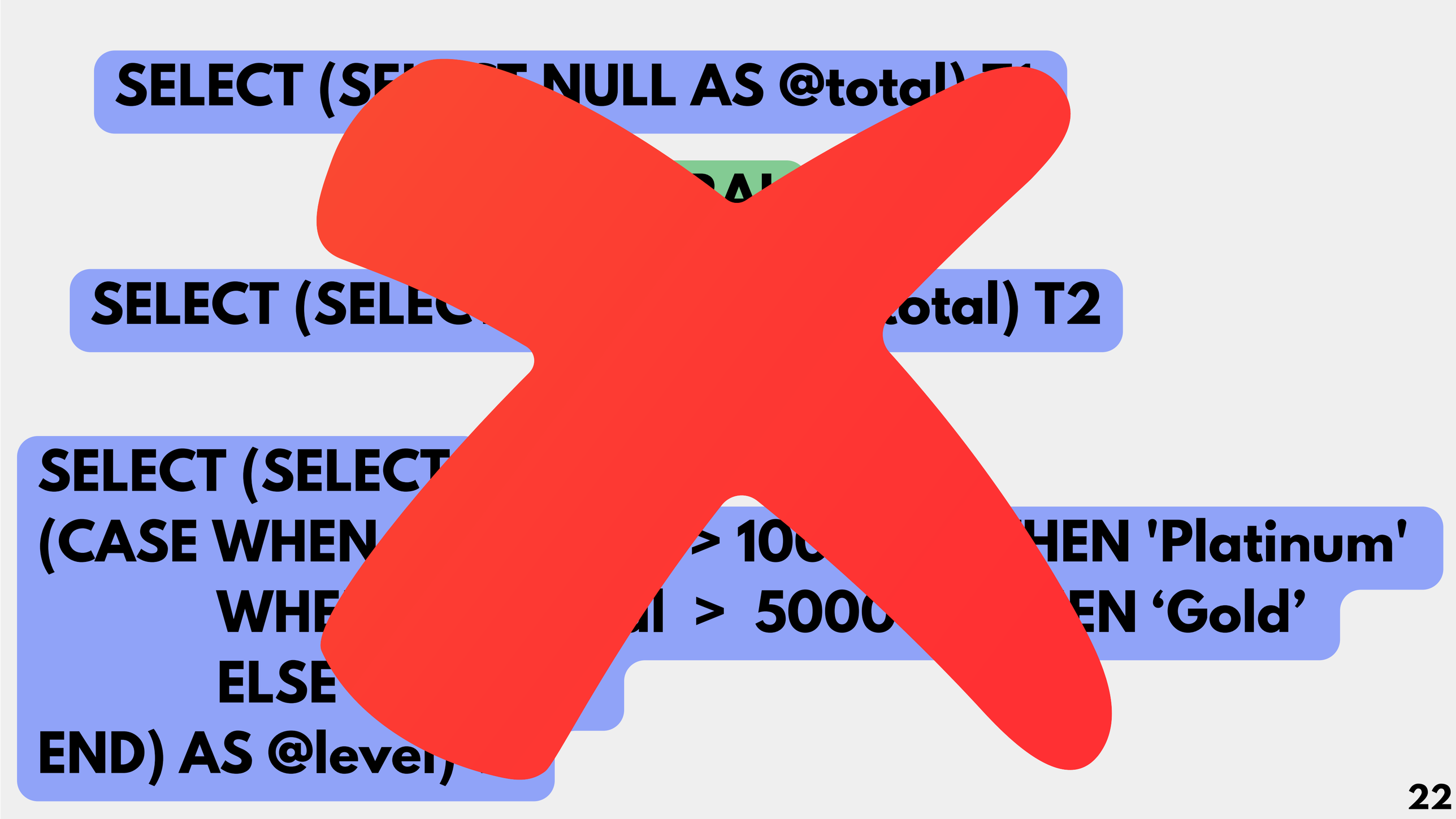
```
SELECT (SELECT NULL AS @total) T1
```

LATERAL

```
SELECT (SELECT SUM(...) AS @total) T2
```

LATERAL

```
SELECT (SELECT
(CASE WHEN  T2.@total  > 1000000 THEN 'Platinum'
          WHEN  T2.@total  >  500000  THEN ‘Gold’
          ELSE ‘Regular’
END) AS @level) T3
```


SELECT (SE NULL AS @total)
SELECT (SELEC otal) T2
SELECT (SELECT
(CASE WHEN > 10 HEN 'Platinum'
WHE l > 5000 EN 'Gold'
ELSE
END) AS @level

How do we replace all subqueries with joins?

Unnesting Arbitrary Queries (2015)

Unnesting Arbitrary Queries

Thomas Neumann and Alfons Kemper
Technische Universität München
Munich, Germany
neumann@in.tum.de, kemper@in.tum.de

Abstract: SQL-99 allows for nested subqueries at nearly all places within a query. From a user's point of view, nested queries can greatly simplify the formulation of complex queries. However, nested queries that are correlated with the outer queries frequently lead to dependent joins with nested loops evaluations and thus poor performance.

Existing systems therefore use a number of heuristics to *unnest* these queries, i.e., de-correlate them. These unnesting techniques can greatly speed up query processing, but are usually limited to certain classes of queries. To the best of our knowledge no existing system can de-correlate queries in the general case. We present a generic approach for unnesting arbitrary queries. As a result, the de-correlated queries allow for much simpler and much more efficient query evaluation.

Unnesting Arbitrary Queries (2015)

Unnesting Arbitrary Queries

Thomas Neumann and Alfons Kemper
Technische Universität München
Munich, Germany
neumann@in.tum.de, kemper@in.tum.de

Abstract: SQL-99 allows for nested subqueries at nearly all places within a query. From a user's point of view, nested queries can greatly simplify the formulation of complex queries. However, nested queries that are correlated with the outer queries frequently lead to dependent joins with nested loops evaluations and thus poor performance.

Existing systems therefore use a number of heuristics to *unnest* these queries, i.e., de-correlate them. These unnesting techniques can greatly speed up query processing, but are usually limited to certain classes of queries. To the best of our knowledge no existing system can de-correlate queries in the general case. We present a generic approach for unnesting arbitrary queries. As a result, the de-correlated queries allow for much simpler and much more efficient query evaluation.

Replace all subqueries with joins!

Which system?

(1) DBMS must support “Neumann-Style” unnesting

Which system?

(1) DBMS must support "Neumann-Style" unnesting

HyPer

UMBRA

DuckDB

Which system?

(1) DBMS must support "Neumann-Style" unnesting

HyPer

UMBRA

DuckDB

(2) DBMS must be open-source

Which system?

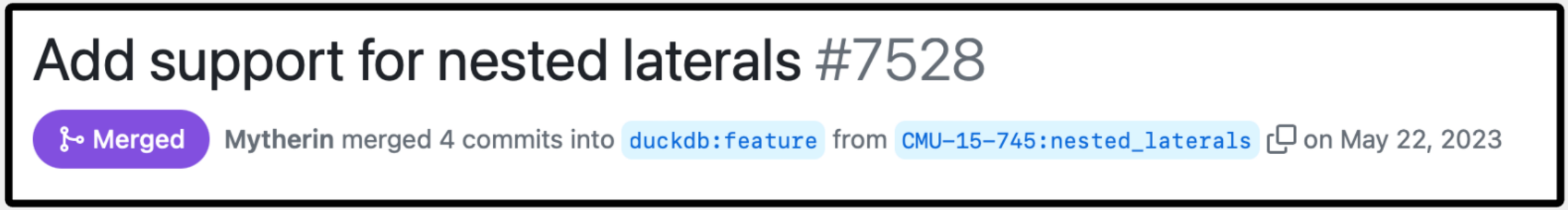
(1) DBMS must support "Neumann-Style" unnesting

(2) DBMS must be open-source

DuckDB

```
D SELECT (SELECT z FROM (SELECT x) t(y), LATERAL (SELECT y) t2(z)) FROM generate_series(1,5) t(x);
Error: Binder Error: Nested lateral joins or lateral joins in correlated subqueries are not (yet) supported
D
```

DuckDB



github.com/duckdb/duckdb/pull/7528

DuckDB

Add support for nested laterals #7528

Merged Mytherin merged 4 commits into `duckdb:feature` from `CMU-15-745:nested_laterals` on May 22, 2023

github.com/duckdb/duckdb/pull/7528

```
D SELECT (SELECT z FROM (SELECT x) t(y), LATERAL (SELECT y) t2(z)) FROM generate_series(1,5) t(x);
┌──────────────────────────────────────────────────────────────────────────┐
│ (SELECT z FROM (SELECT x) AS t(y) , (SELECT y) AS t2(z))                 │
│                                  int64                                   │
├──────────────────────────────────────────────────────────────────────────┤
│                                                                        1 │
│                                                                        2 │
│                                                                        3 │
│                                                                        4 │
│                                                                        5 │
└──────────────────────────────────────────────────────────────────────────┘
D
```

But what about other DBMSs?

But what about other DBMSs?

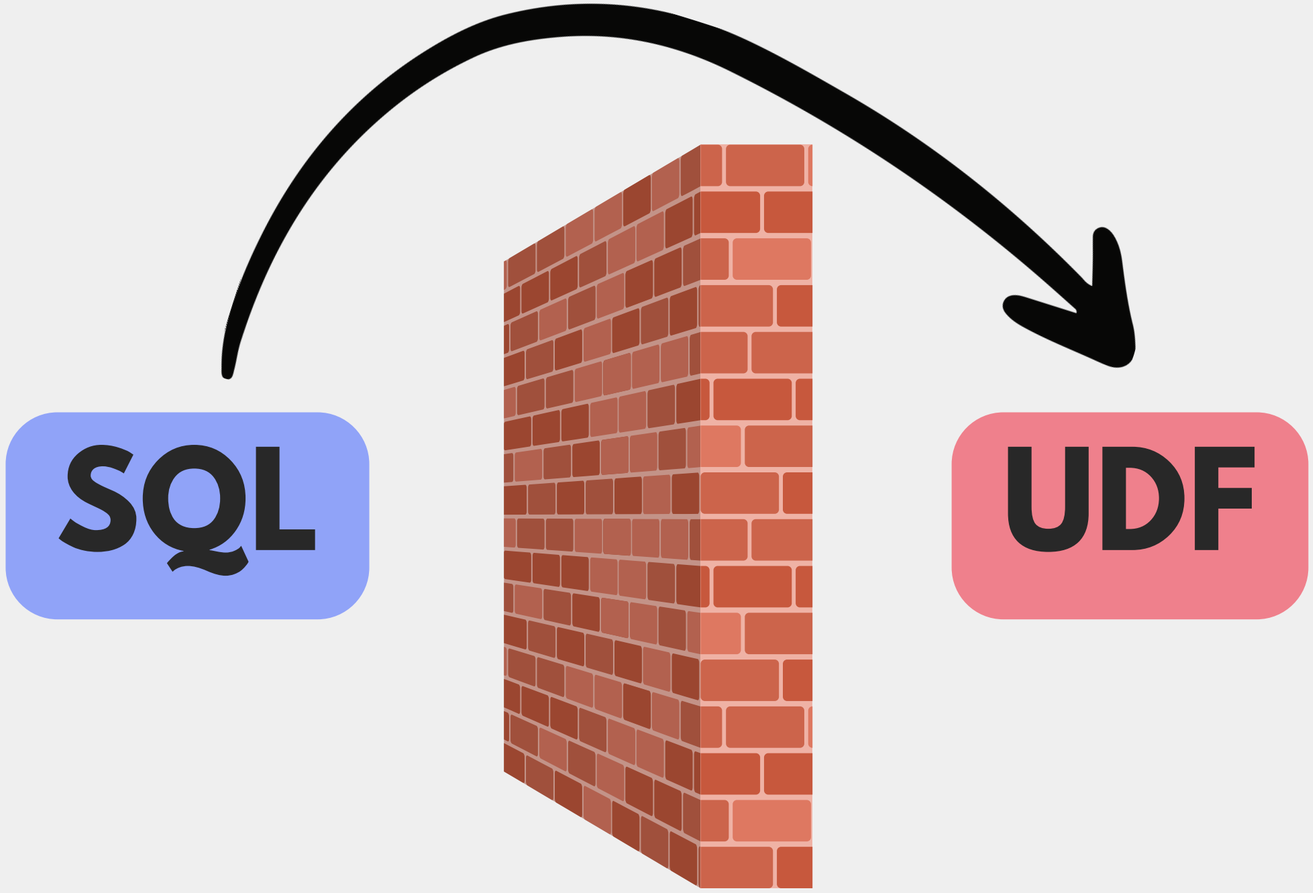
Inlining = LATERAL joins = Slow

But what about other DBMSs?

Inlining = LATERAL joins = Slow

Can we avoid LATERAL joins?

Batching



Batching

```
CREATE TEMPORARY TABLE temp (x, y);
```

```
INSERT INTO temp
SELECT (NULL, NULL)
  FROM input;
```

Batching

x = y

UPDATE temp
SET x = y

x = (SELECT ...)

UPDATE temp
SET x = (SELECT ...)

IF cond THEN x = a

UPDATE temp
SET x = a
WHERE cond

UDF Batching

```
CREATE TEMPORARY TABLE
temp (c_name, c_custkey,
@total, @level);
```

```
INSERT INTO temp
SELECT (c_name, c_custkey,
NULL, NULL)
  FROM customer;
```

```
UPDATE temp
SET @total = NULL;
```

```
UPDATE temp
SET @total =
(SELECT SUM(...));
```

```
UPDATE temp
SET @level =
(CASE WHEN  @total  > 1000000
            THEN 'Platinum'
          WHEN  @total  >  500000
            THEN ‘Gold’
              ELSE ‘Regular’ END)
```

```
SELECT c_name, @level
  FROM temp;
```

Batching vs Inlining

Batching

Copying overhead ✗

Many small queries ✓

Inlining

No copying overhead ✓

One complex query ✗

Who created batching?

Optimizing Procedural User-Defined Functions in Database Systems

Kai Franz

CMU Thesis (2023)

Who created batching?

Optimization of PL/pgSQL Translations Using Batching and Multiple Recursive References

Marcus Huber

Tubingen Thesis (2022)

Who created batching?

denis 7 months ago

Second thing, do you know about this work? http://www.vldb.org/pvldb/vol1/1453975.pdf

Who created batching?

Rewriting Procedures for Batched Bindings

Ravindra Guravannavar*
Indian Institute of Technology, Bombay
ravig@cse.iitb.ac.in

S Sudarshan
Indian Institute of Technology, Bombay
sudarsha@cse.iitb.ac.in

IIT Bombay (2008)

Which is better?
Inlining or batching?

Experimental Setup

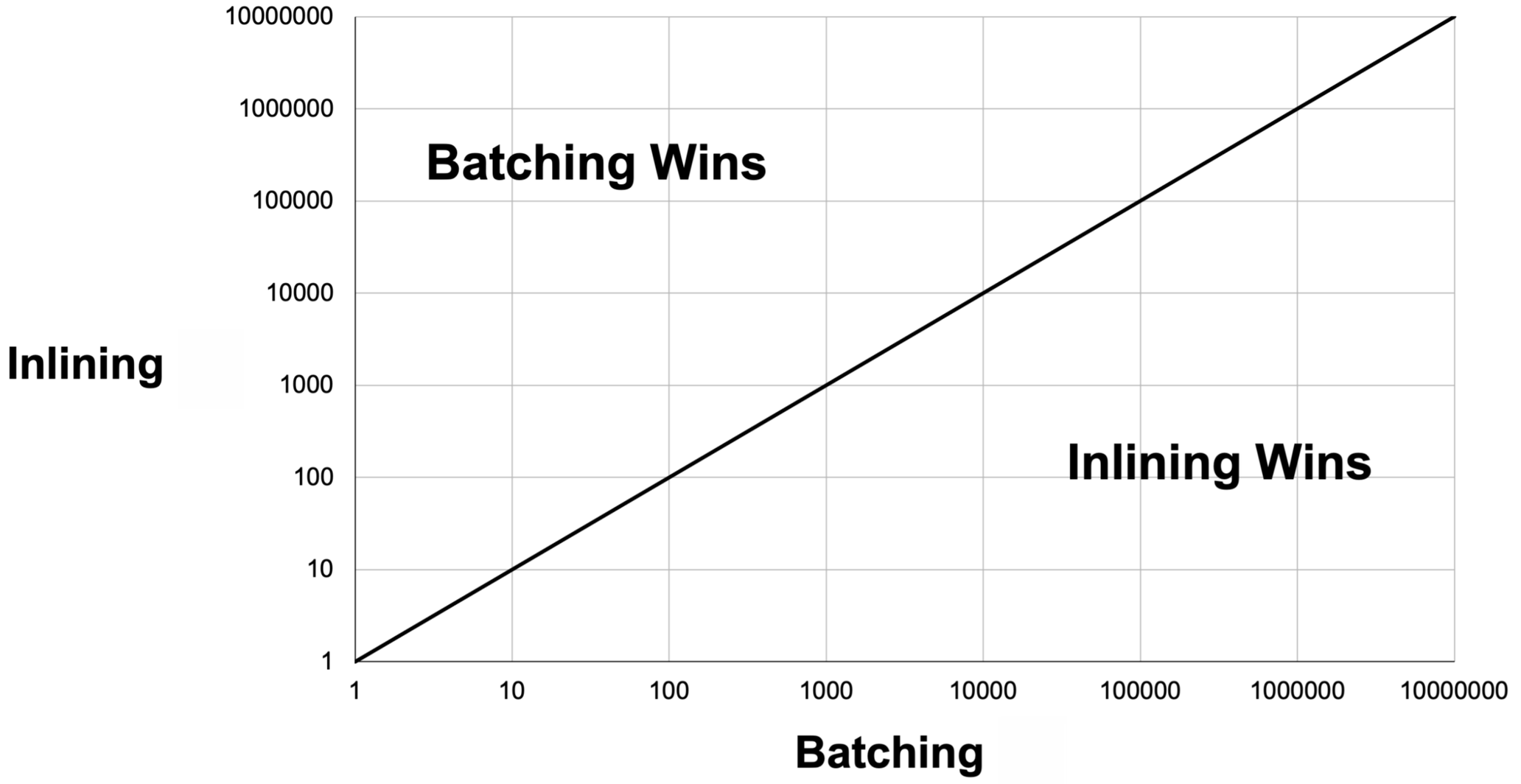
First batching vs inlining comparison!

UDFs from ProcBench, 1GB Scale

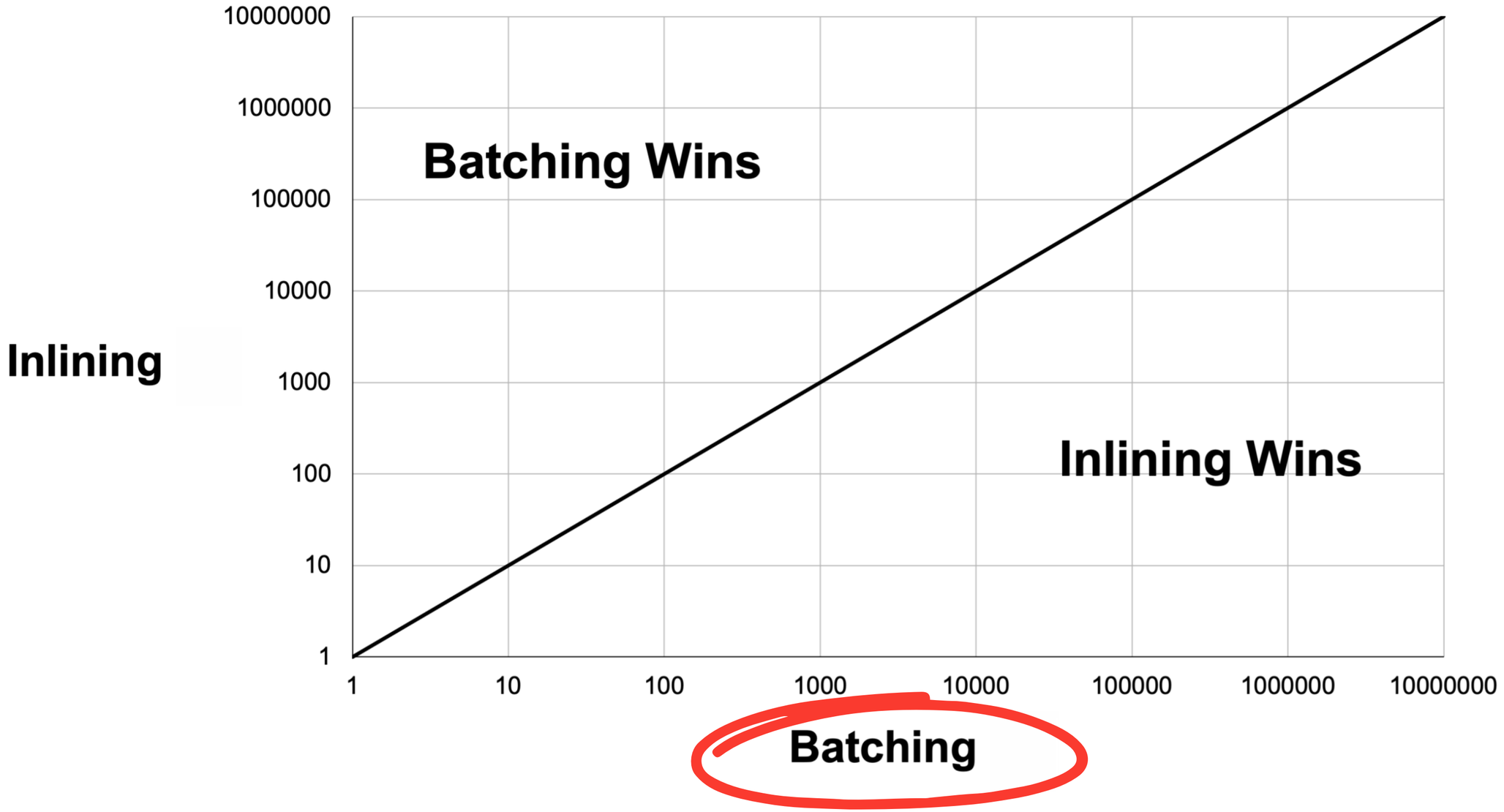
SQL Server, DuckDB, PostgreSQL, Oracle

We report relative speedup

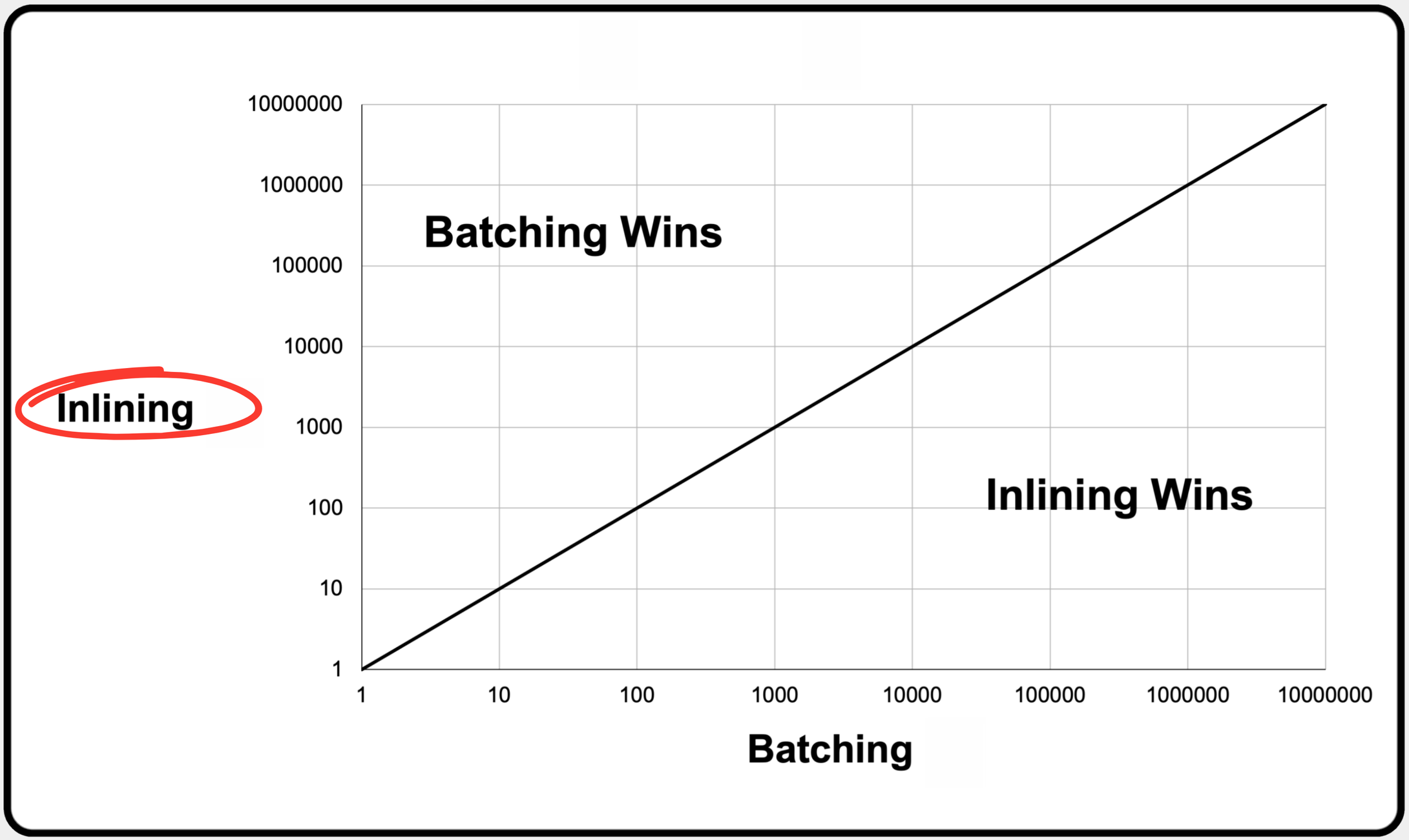
Experimental Results



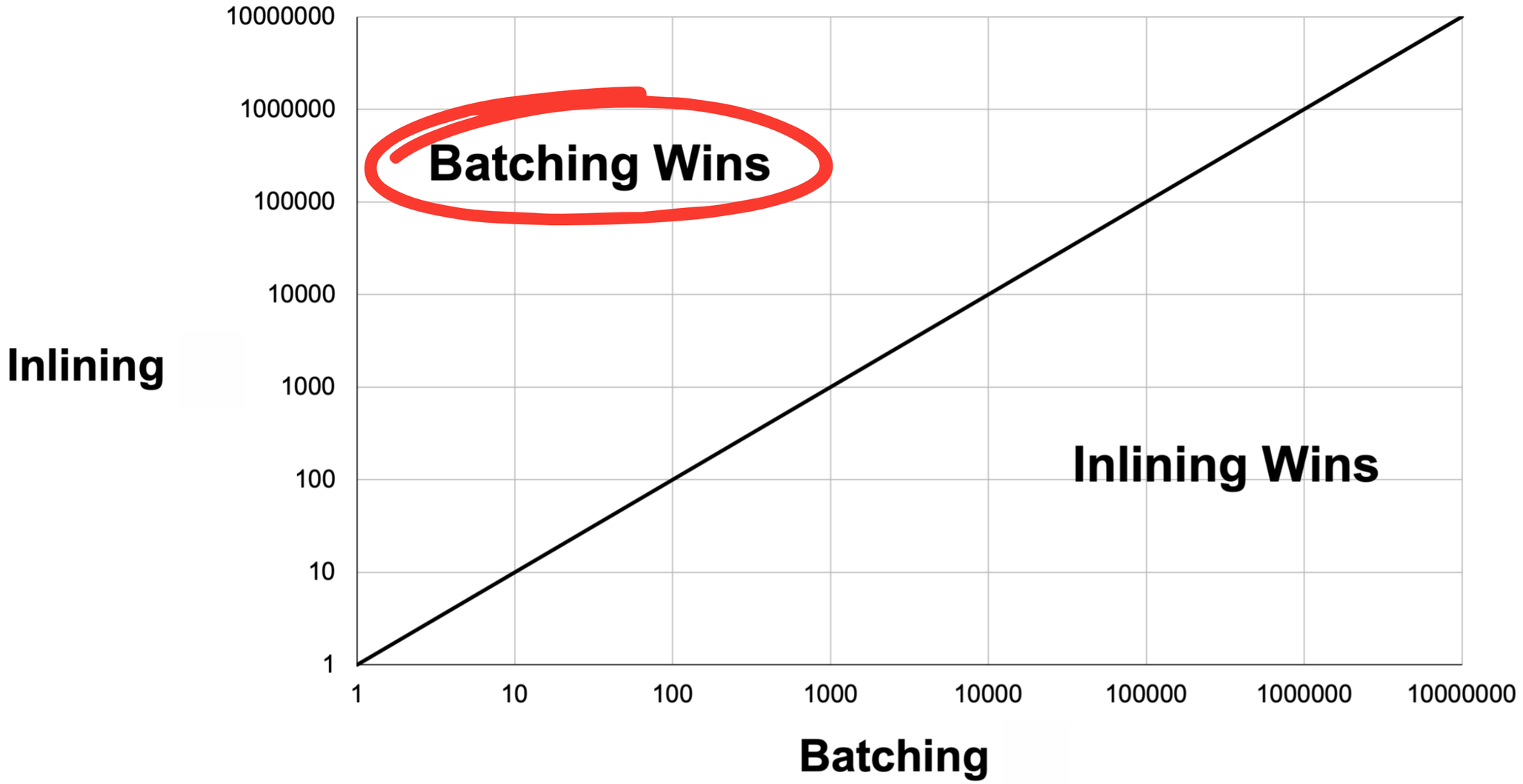
Experimental Results



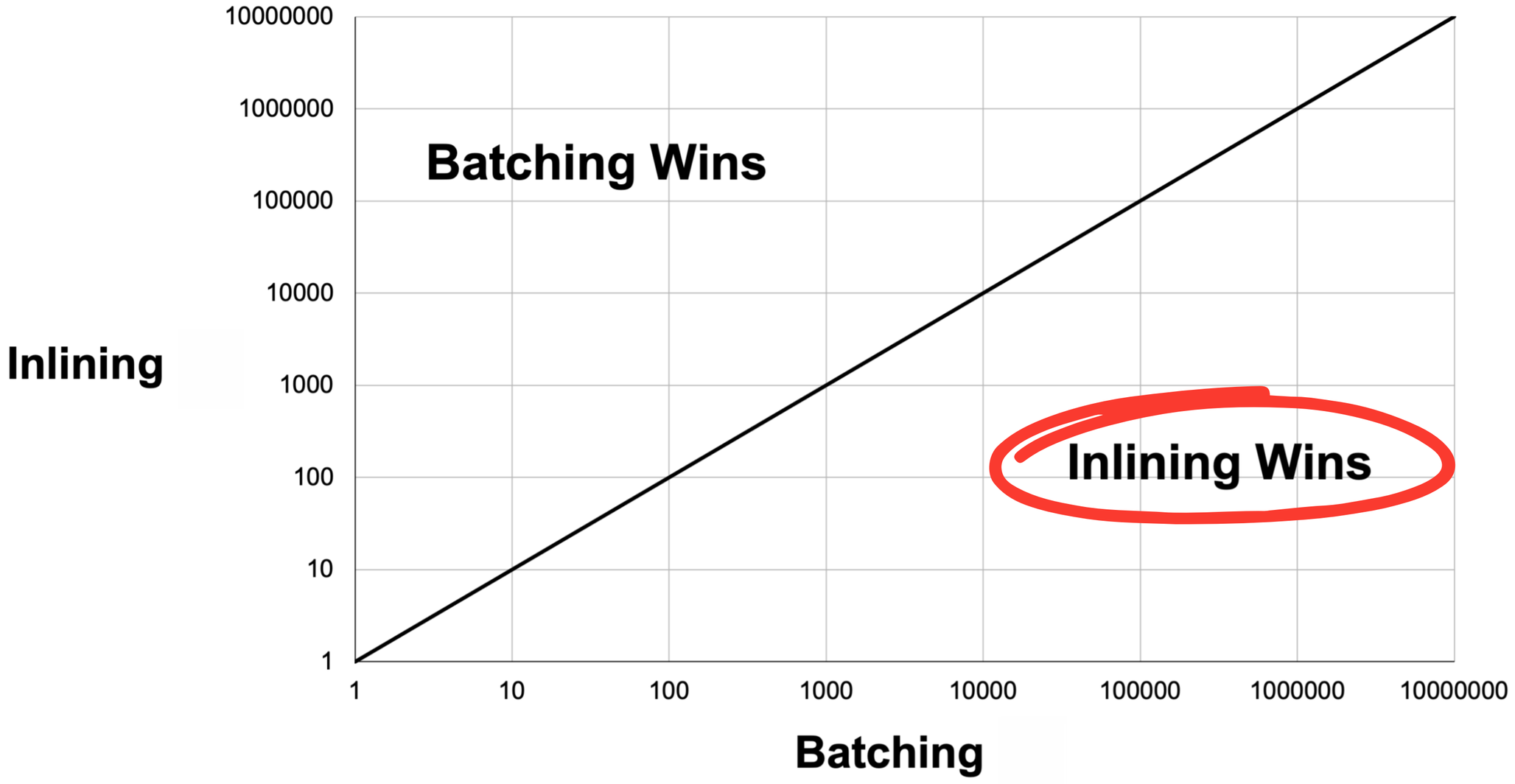
Experimental Results



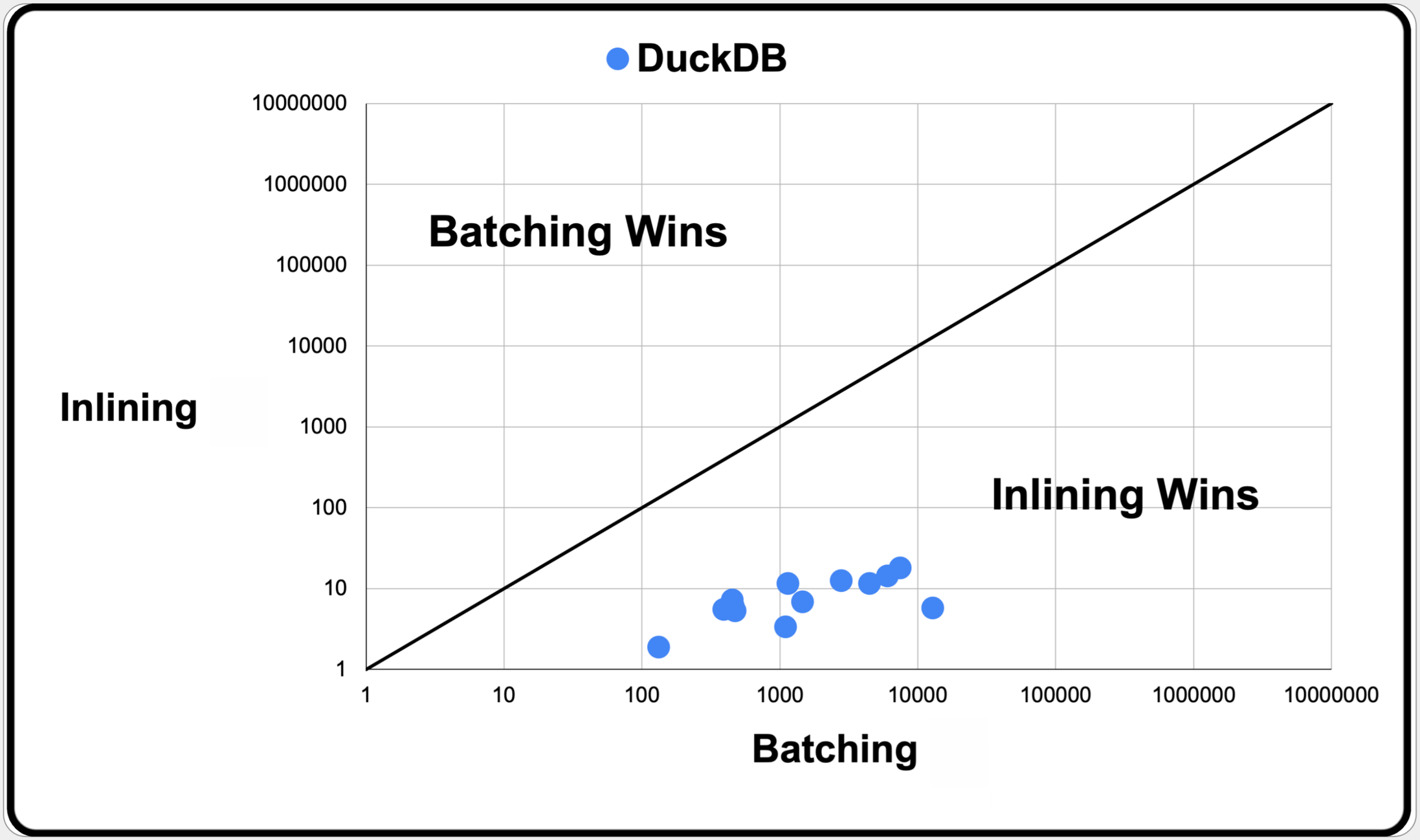
Experimental Results



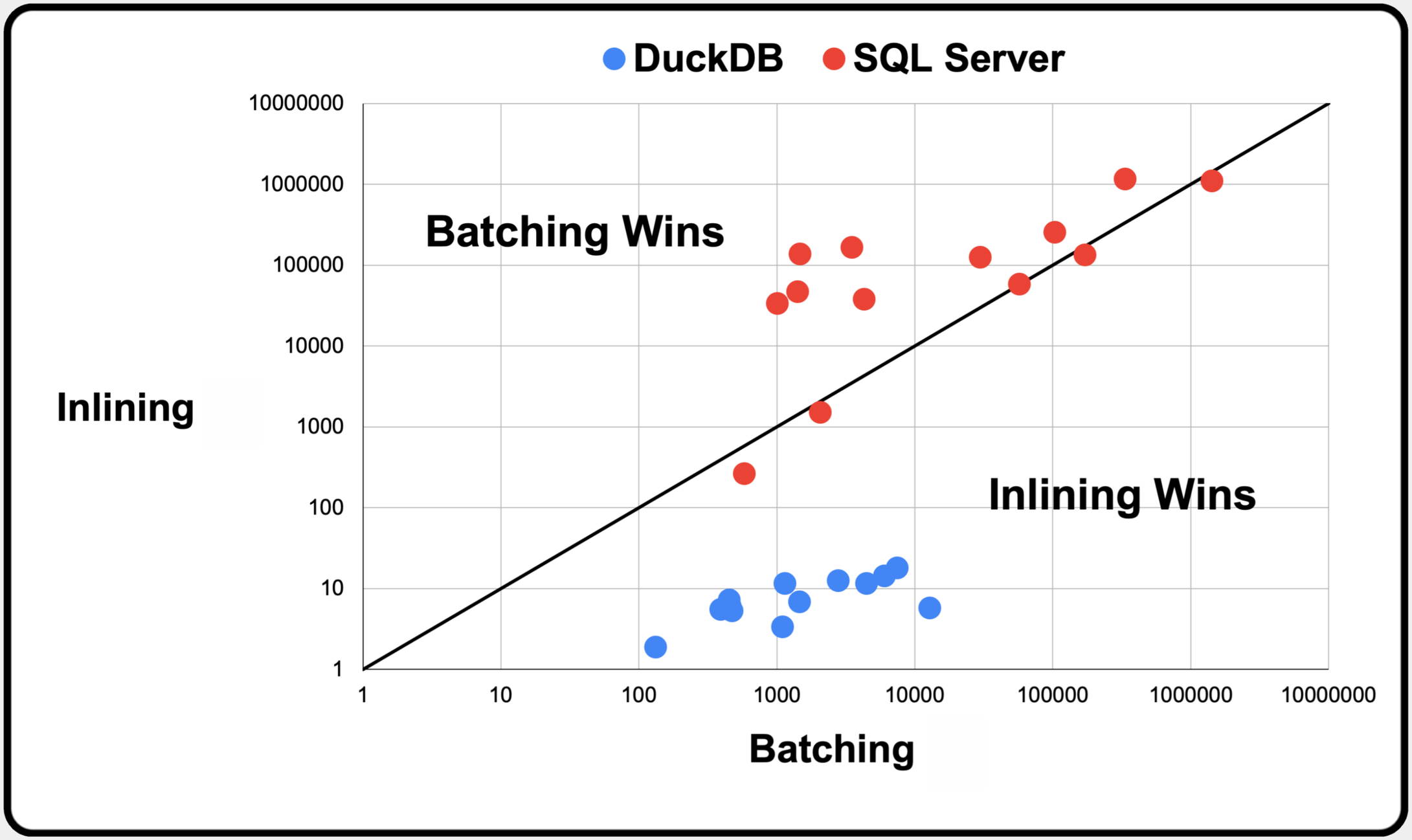
Experimental Results



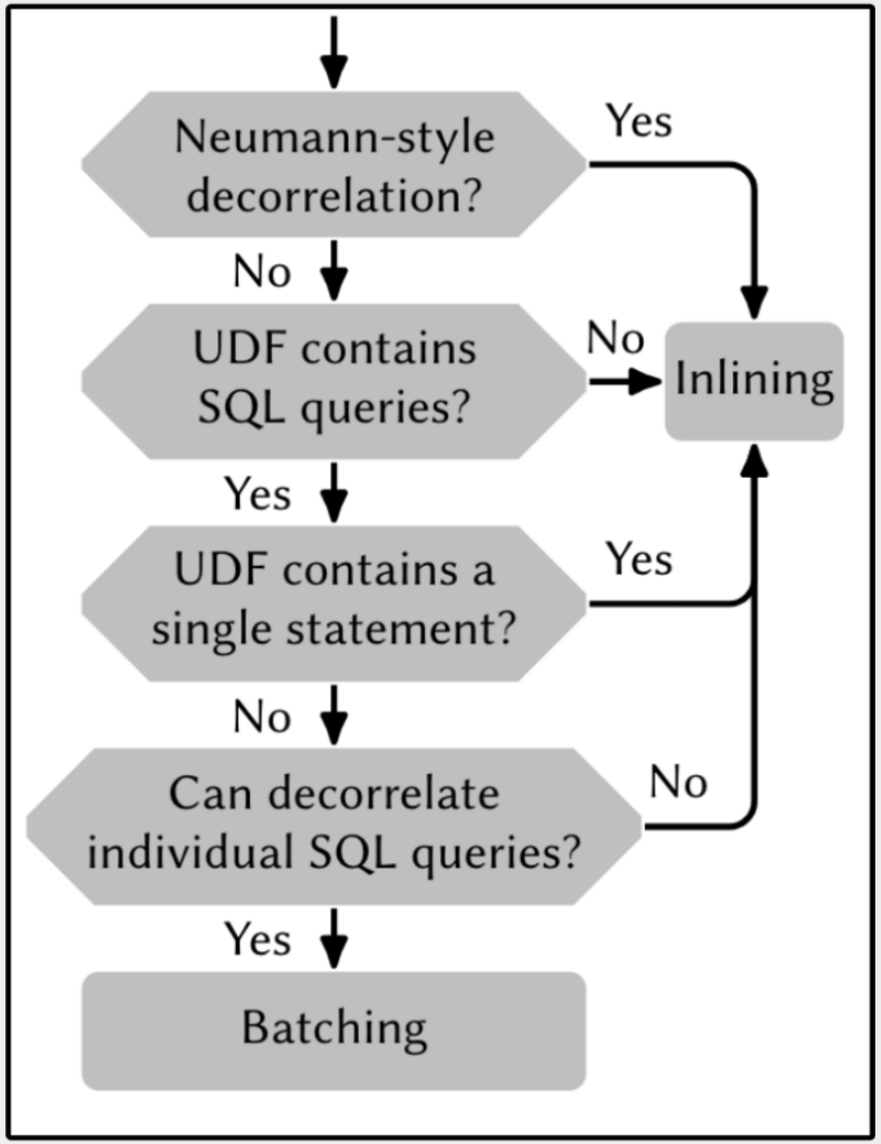
Experimental Results



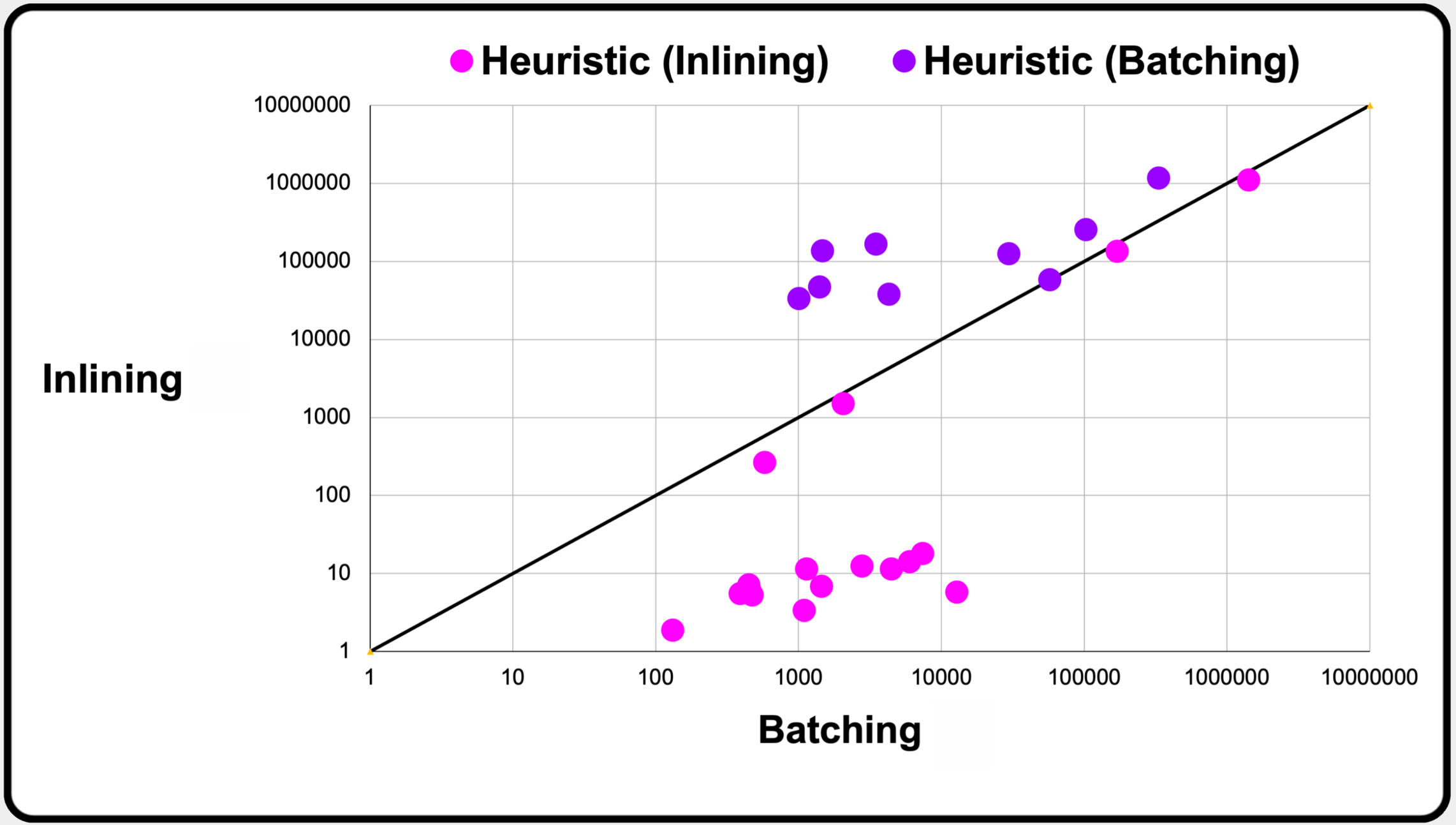
Experimental Results



Heuristic


Neumann-style decorrelation?
Yes
No
UDF contains SQL queries?
No
Inlining
Yes
UDF contains a single statement?
Yes
No
Can decorrelate individual SQL queries?
No
Yes
Batching

Experimental Results



Wrap Up

(1) UDF Inlining is not a silver bullet

Wrap Up

(1) UDF Inlining is not a silver bullet

(2) Subquery unnesting is crucial

Wrap Up

(1) UDF Inlining is not a silver bullet

(2) Subquery unnesting is crucial

(3) Batching works well

Wrap Up

(1) UDF Inlining is not a silver bullet

(2) Subquery unnesting is crucial

(3) Batching works well

(4) Hybrid is best

Future of UDFs

(1) LATERAL-free inlining?

Future of UDFs

(1) LATERAL-free inlining?

(2) Combine UDF inlining & compilation?

Future of UDFs

(1) LATERAL-free inlining?

(2) Combine UDF inlining & compilation?

(3) Inline Python UDFs?

New Girlfriend?

samarch.xyz

sarch@cs.cmu.edu

Experimental Setup

First batching vs inlining comparison!

UDFs from ProcBench, 1GB SF

Intel Xeon 5218R CPU, 192GB DDR4 RAM, 500GB NVMe SSD

SQL Server, DuckDB, PostgreSQL, Oracle

We report relative speedup