How to Win a Hot Dog Eating Contest: Incremental View Maintenance with Batch Updates

Milos Nikolic, Mohammad Dashti, Christoph Koch
DATA lab, EPFL

SIGMOD, 28th June 2016

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

REALTIME APPLICATIONS

Web Analytics

Sensor Networks

Cloud Monitoring

EVENTS → RUNTIME ENGINE → DECISION SUPPORT → ACTIONS

Continuously arriving data

Continuously evaluated views

REALTIME SYSTEMS: REQUIREMENTS

LOW LATENCY PROCESSING

Incremental view maintenance

$Q(D + \Delta D) = Q(D) + \Delta Q(D, \Delta D)$

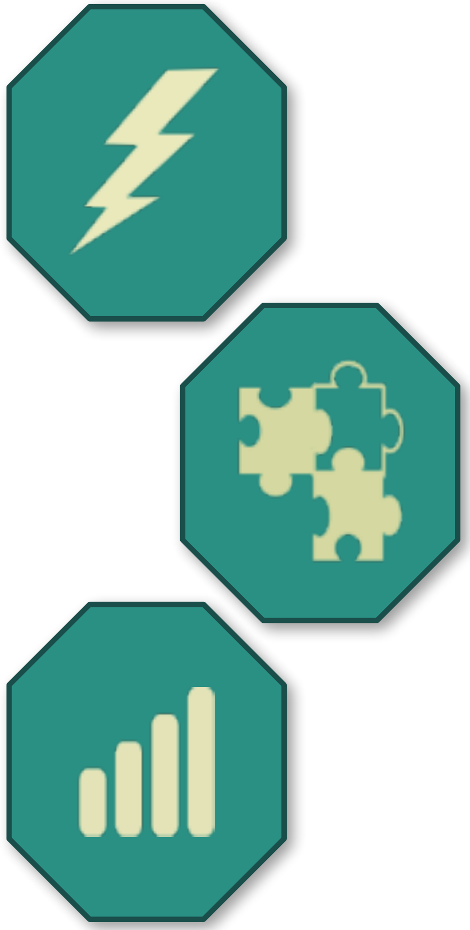
COMPLEX CONTINUOUS QUERIES

SQL queries (w/ nested aggregates)

No window semantics

SCALABLE PROCESSING

Synchronous execution model



IN THIS TALK

Q1: How does the size of update affect the performance of incremental computation?

Q2: (Idea) How to achieve efficient distributed incremental computation?

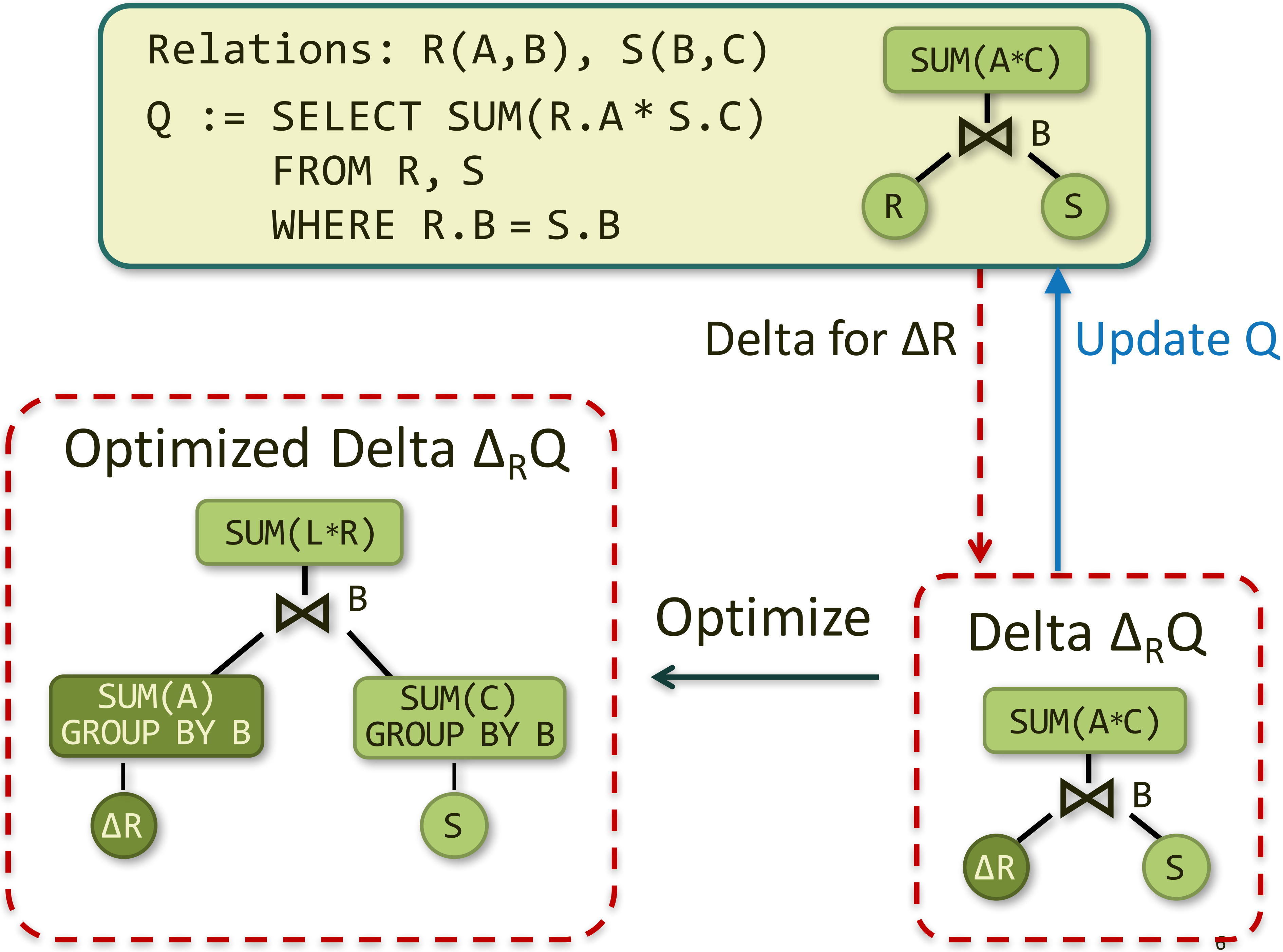
HIGH-PERFORMANCE INCREMENTAL COMPUTATION

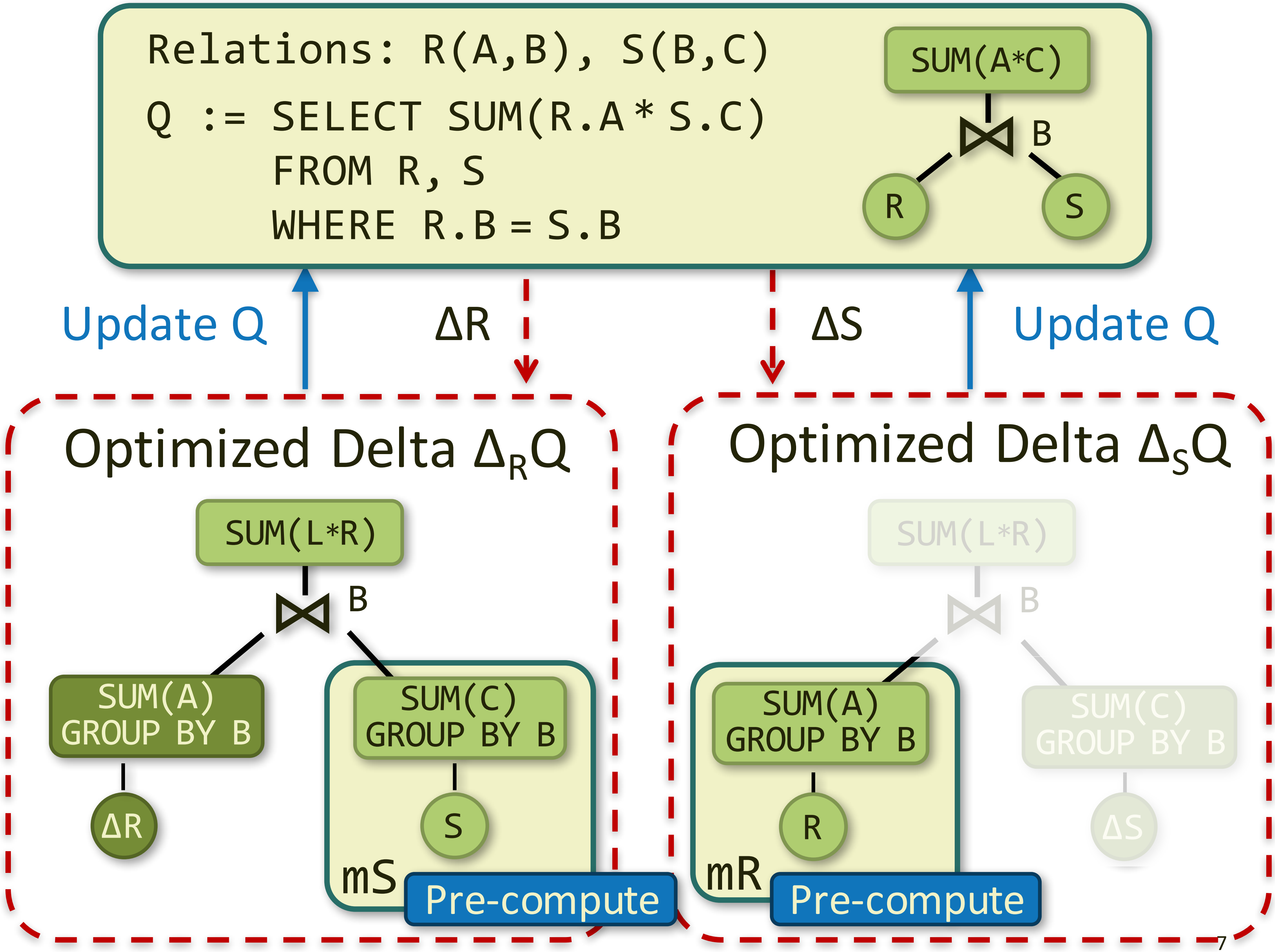
PROBLEM: DBMS & stream engines with classical IVM can have poor performance on fast, long-lived data

OUR APPROACH: Compilation of SQL queries into incremental engines

TOASTER = Recursive IVM + Code Generation (C++, Scala, Spark)

PERF: Million view refreshes/sec for single-tuple updates

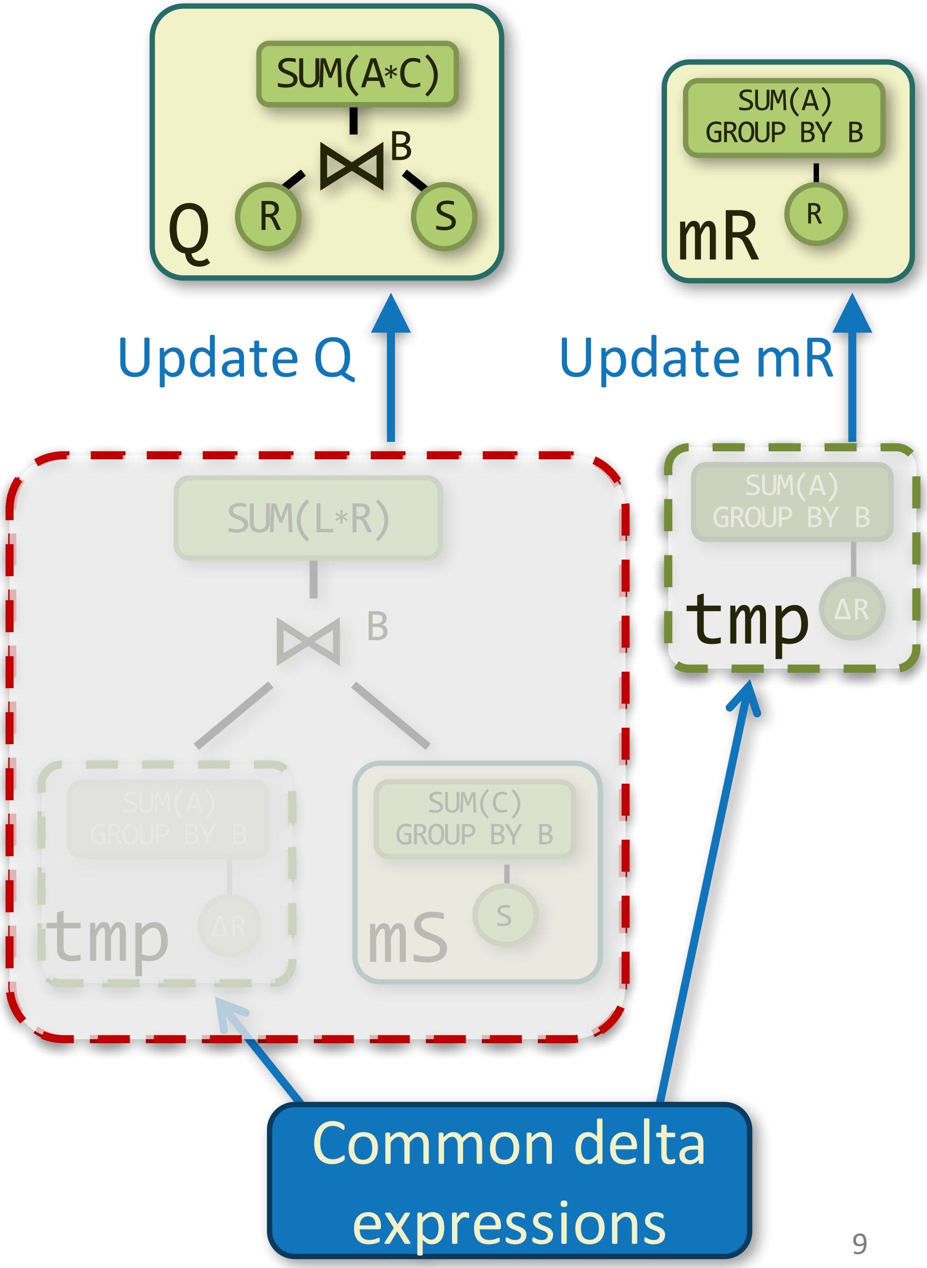

Relations: R(A,B), S(B,C)
Q := SELECT SUM(R.A * S.C)
FROM R, S
WHERE R.B = S.B
SUM(A*C)
B
R
S
Delta for ΔR
Update Q
Optimized Delta $\Delta_R Q$
SUM(L*R)
B
SUM(A)
GROUP BY B
SUM(C)
GROUP BY B
ΔR
S
Optimize
Delta $\Delta_R Q$
SUM(A*C)
B
ΔR
S


Relations: R(A,B), S(B,C)
Q := SELECT SUM(R.A * S.C)
FROM R, S
WHERE R.B = S.B
SUM(A*C)
B
R
S
Update Q
ΔR
ΔS
Update Q
Optimized Delta Δ_R Q
SUM(L*R)
B
SUM(A)
GROUP BY B
ΔR
SUM(C)
GROUP BY B
S
mS
Pre-compute
Optimized Delta Δ_S Q
SUM(L*R)
B
SUM(A)
GROUP BY B
R
mR
Pre-compute
SUM(C)
GROUP BY B
ΔS

BATCH TRIGGER

ON UPDATE R BY ΔR:

```
// Pre-aggregate batch
tmp[B] := SELECT B, SUM(A)
             FROM ΔR
             GROUP BY B

// Update Q
Q  += SELECT SUM(tmp.V*mS.V)
         FROM tmp, mS
         WHERE tmp.B = mS.B

// Update mR
mR[B] += SELECT * FROM tmp
```



BATCH TRIGGER

```
ON UPDATE R BY ΔR:

  // Pre-aggregate batch
  tmp[B] := SELECT B, SUM(A)
              FROM ΔR
              GROUP BY B

  // Update Q
  Q  += SELECT SUM(tmp.V*mS.V)
          FROM tmp, mS
          WHERE tmp.B = mS.B

  // Update mR
  mR[B] += SELECT * FROM tmp
```

BATCH C++

```
void onUpdateR(List<T> dR) {
  // Pre-aggregate batch
  HashMap<int,int> tmp;
  foreach (dA,dB) in dR
    tmp[dB] += dA;

  // Update Q (of type int)
  foreach (k,v) in tmp
    Q += v * mS[k];

  // Update mR
  foreach (k,v) in tmp
    mR[k] += v;
}
```

SINGLE-TUPLE C++

```
void onUpdateR(int dA, int dB) {
  Q += dA * mS[dB];
  mR[dB] += dA;
}
```

BASELINE

BATCH C++

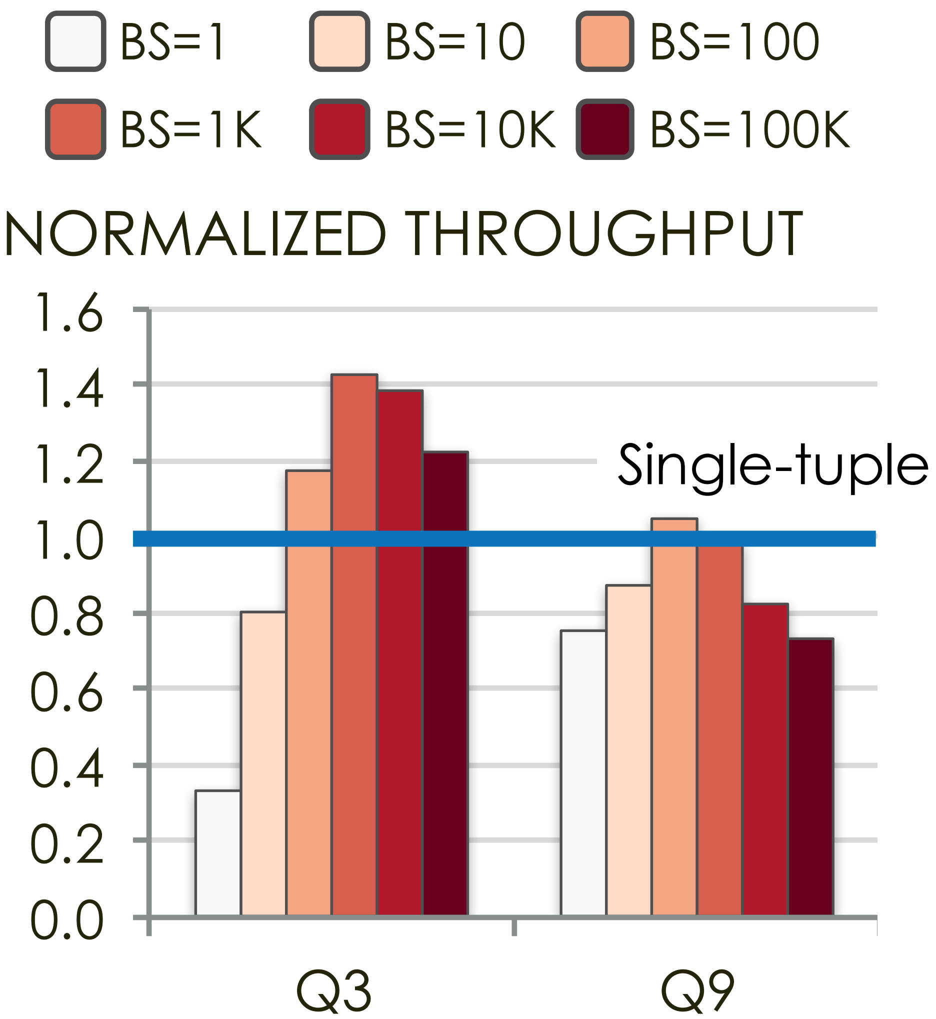
```
void onUpdateR(List<T> dR) {
  // Pre-aggregate batch
  HashMap<int,int> tmp;
  foreach (dA,dB) in dR
    tmp[dB] += dA;

  // Update Q (of type int)
  foreach (k,v) in tmp
    Q += v * mS[k];

  // Update mR
  foreach (k,v) in tmp
    mR[k] += v;
}
```

CODE SPECIALIZATION

- Primitive-type parameters
- No intermediate maps
- Loop elimination
- Partial evaluation, inlining

SINGLE-TUPLE VS. BATCH IVM

TPC-H, 10GB stream, batch size = 1…100,000, C++



MAIN RESULTS

1) Best performance w/ medium batch sizes (= *bite sizes*)

2) Single-tuple processing faster for 5 queries; 7 queries within 20% of best-batch performance

3) Batch pre-aggregation can enable cheaper maintenance

4) OOM faster than DBMS

DISTRIBUTED IVM

DESIGN CHOICE 1:

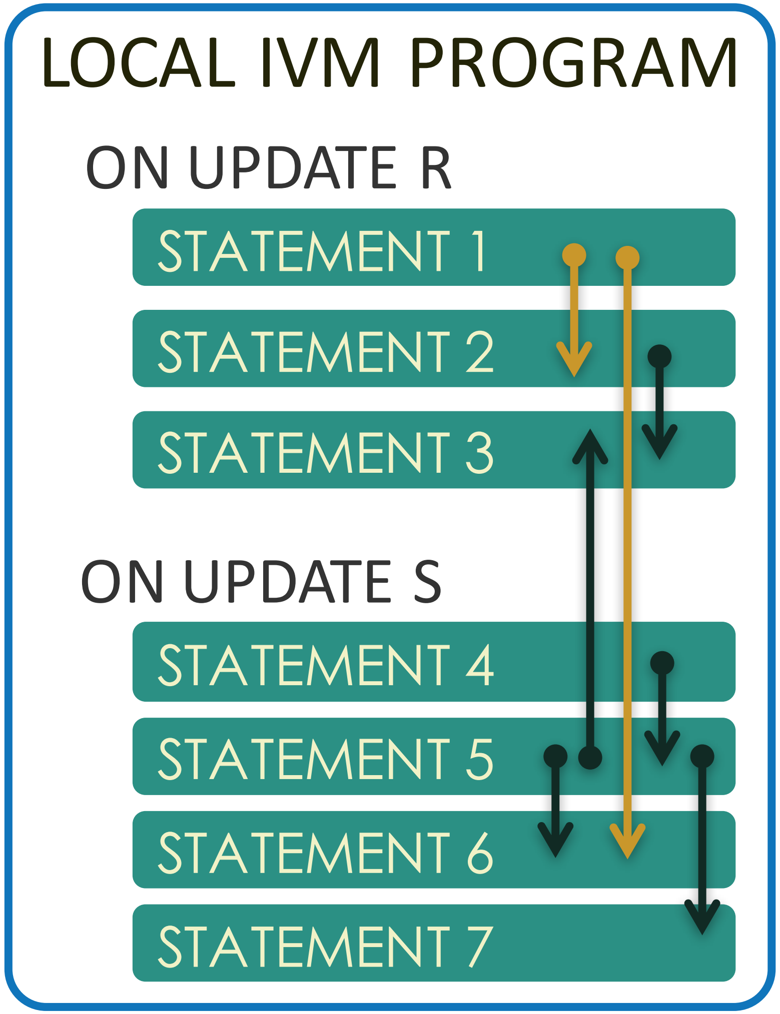
Local ➞ Distributed programs

CHALLENGE:

Dependencies among statements prevent arbitrary re-orderings

DESIGN CHOICE 2:

Synchronous execution model (on top of Spark)

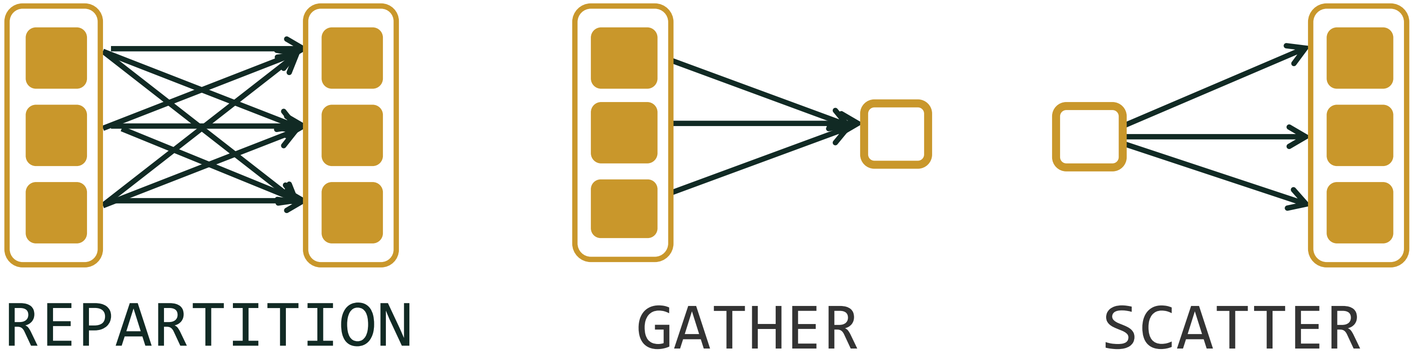


OUR APPROACH

LOCATION TAGS: LOCAL, PARTITIONED BY KEY, RANDOM

Annotate each node in query plan with location info

LOCATION TRANSFORMERS: Insert communication operations into query plan to preserve query semantics



HOLISTIC OPTIMIZATION: Minimize network cost

CONCLUSION

Much more in the paper:

- Single-tuple vs. batch incremental processing (single-tuple can be better!) + more experiments
- Distributed IVM (+ optimization framework)
- IVM of queries with nested aggregates
- Code and data-structure specialization

Download: http://www.dbtoaster.org