DBToaster

Higher-Order Delta Processing for Dynamic, Frequently Fresh Views

Yanif Ahmad — Johns Hopkins

Oliver Kennedy — ~~EPFL~~ University at Buffalo

Christoph Koch — EPFL

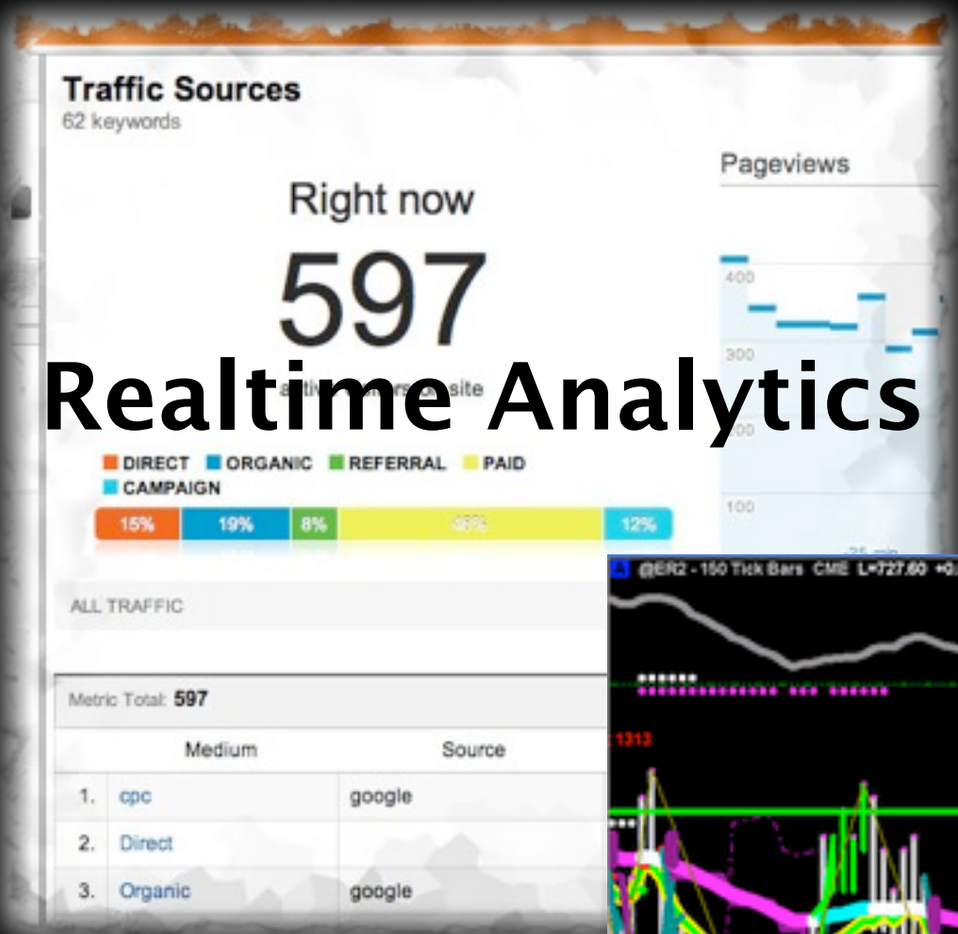
Milos Nikolic — EPFL

Realtime Monitoring Programs…

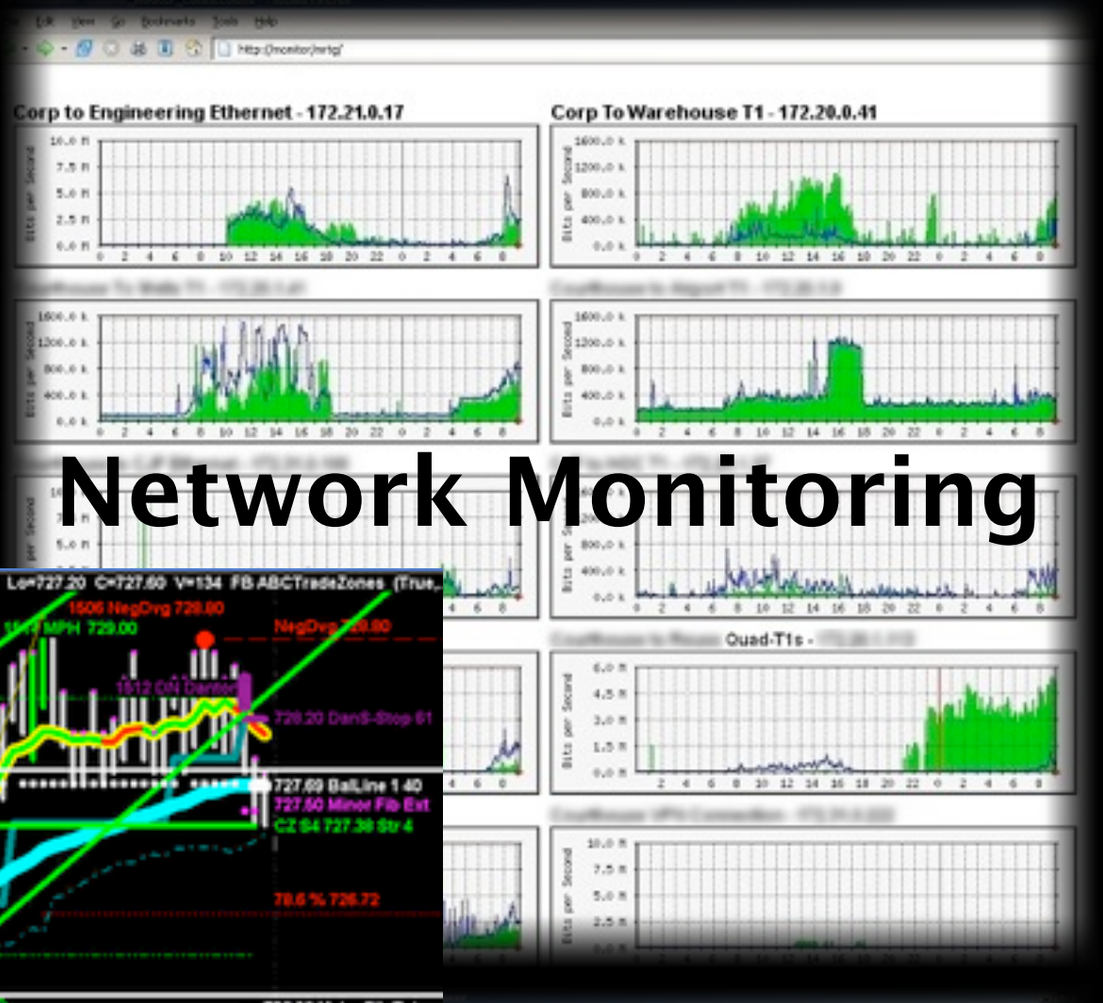
…Monitor The State of the World

…React to Conditions in that State

Realtime Monitoring Programs are Everywhere

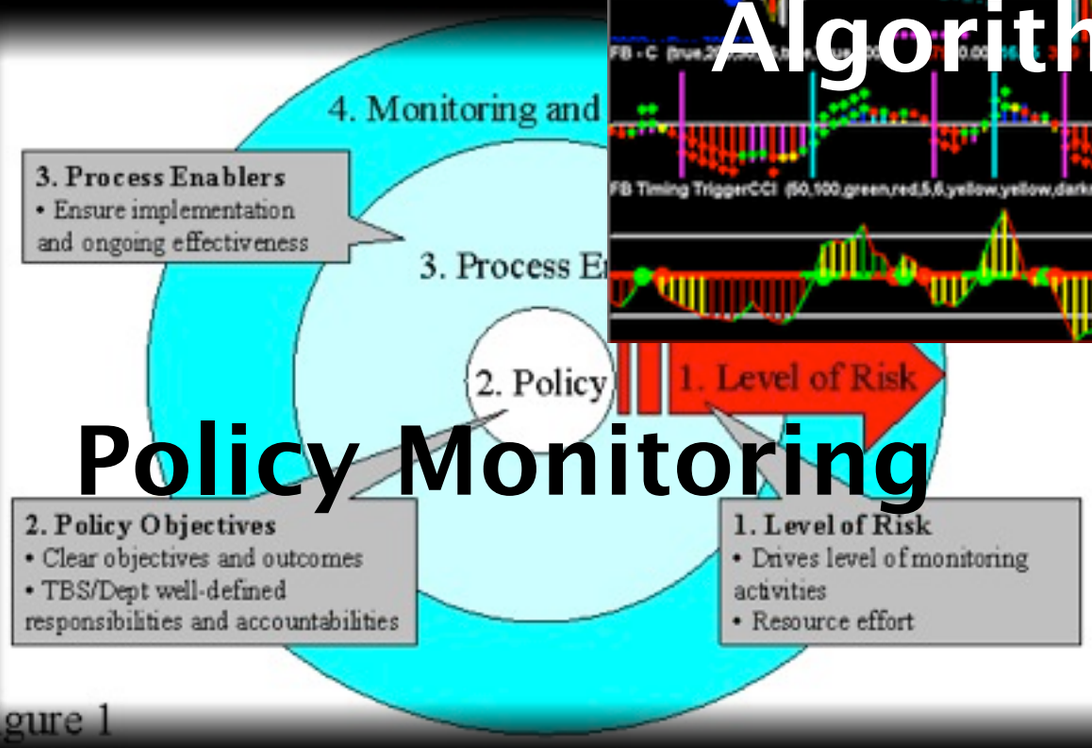


Realtime Analytics

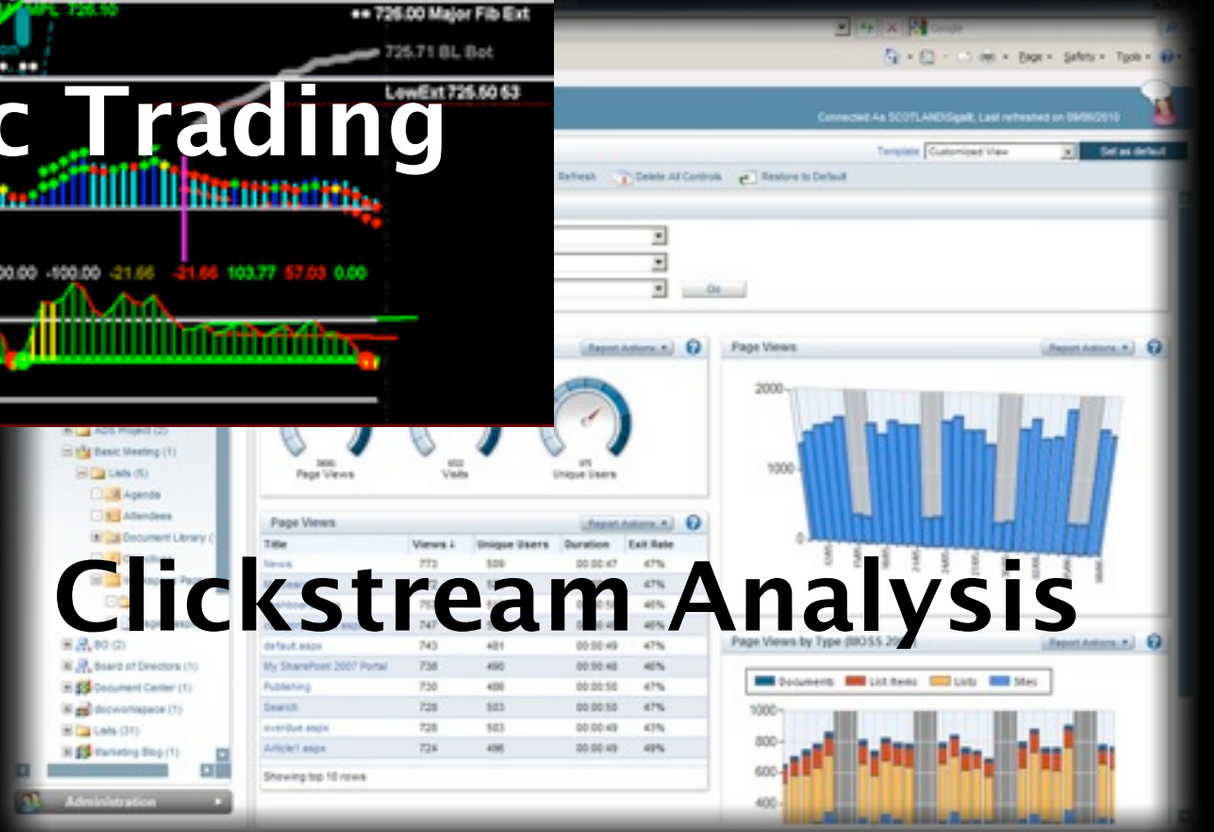


Network Monitoring

Algorithmic Trading

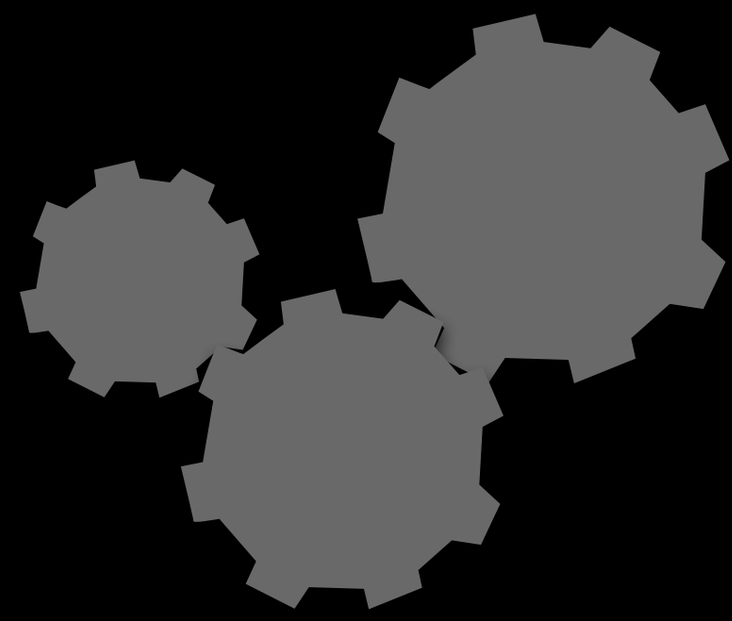


Policy Monitoring

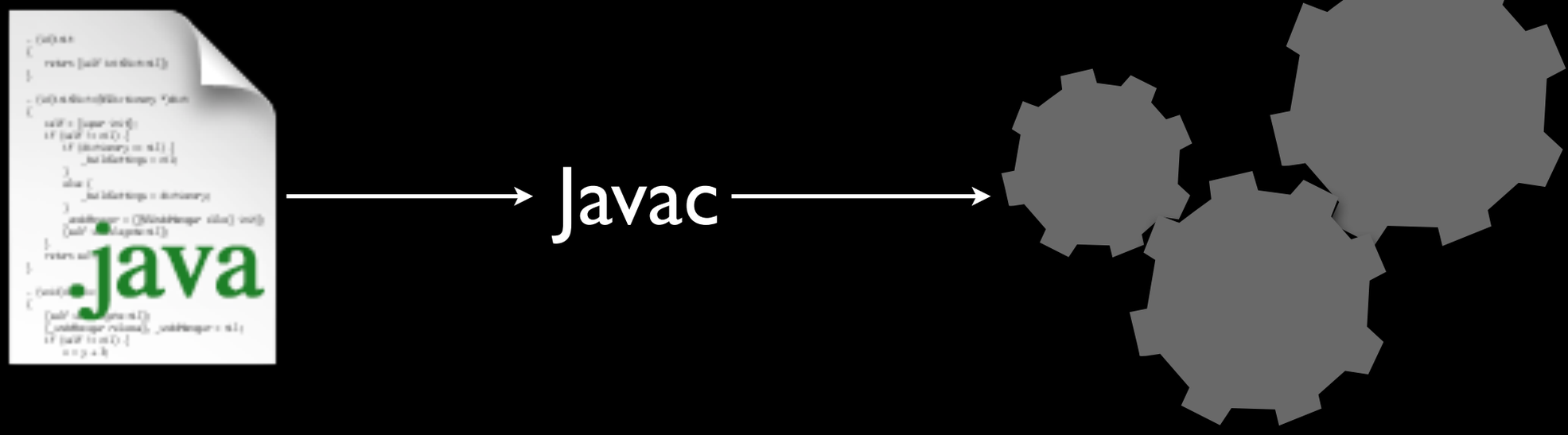


Clickstream Analysis

Monitoring Programs

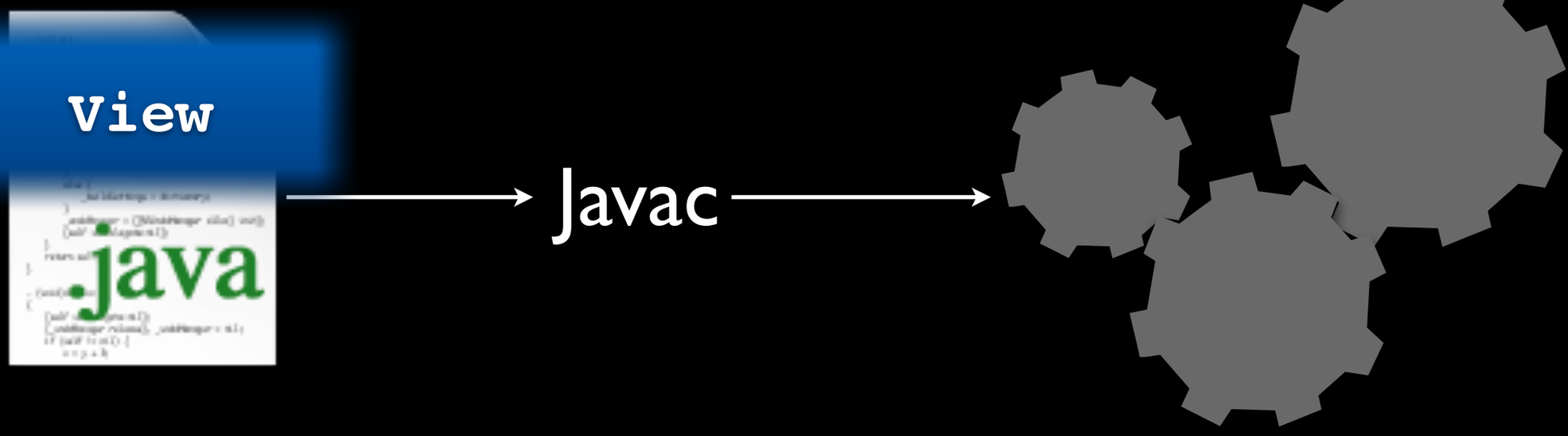


Monitoring Programs



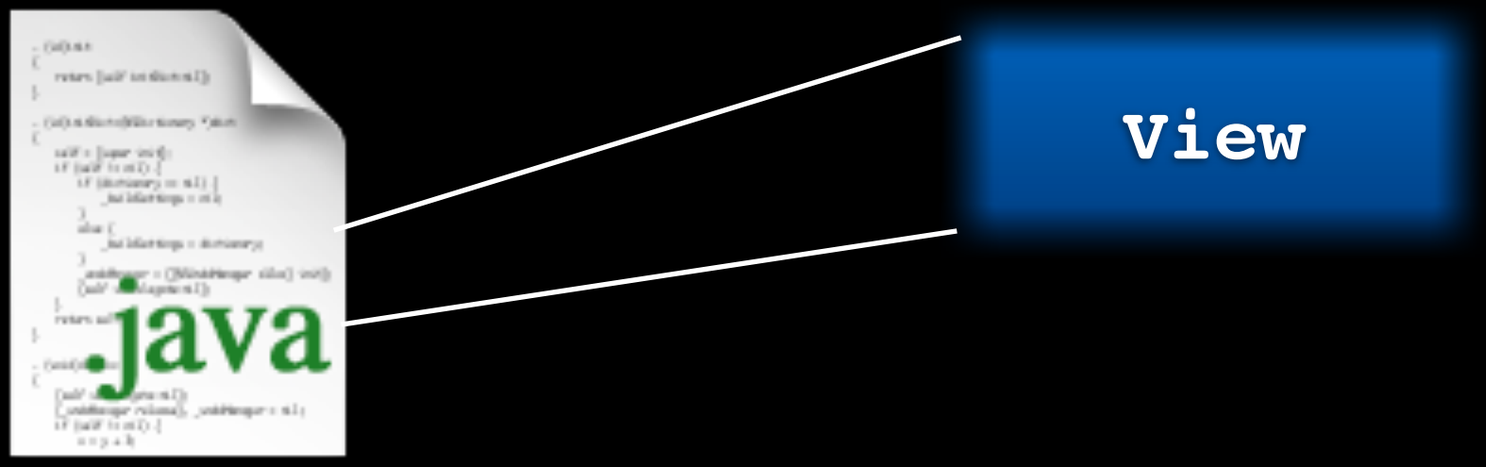
Problem: People write monitoring programs by hand

Monitoring Programs



Problem: People write monitoring programs by hand

Monitoring Programs



Monitoring Programs

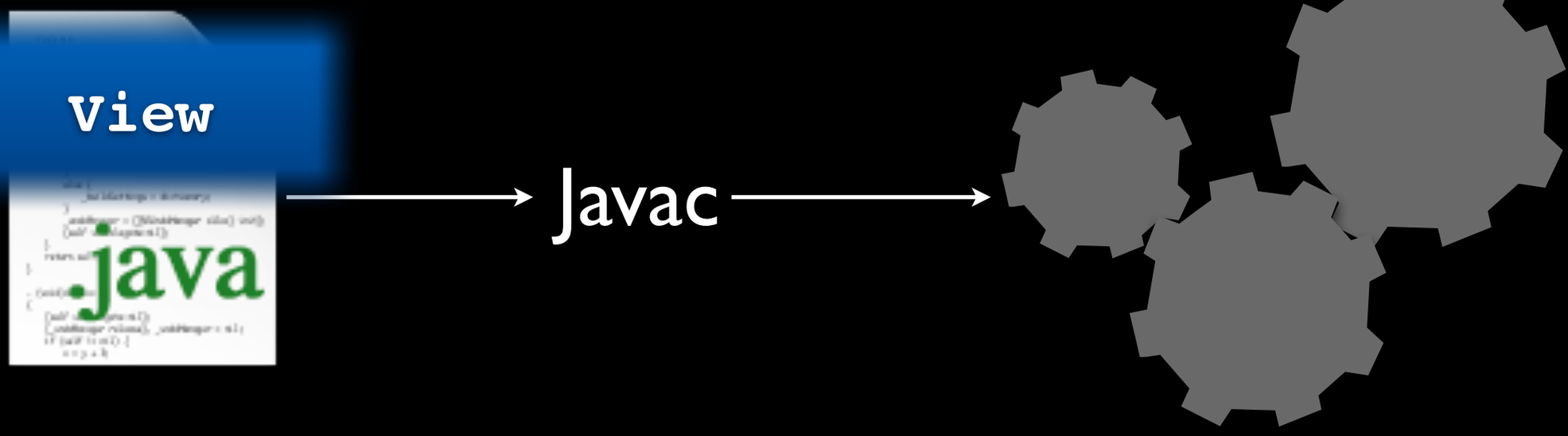
View

- An Aggregate Representation of the State of the World
- Maintained in Realtime as the State of the World Changes
- Needs to React to Changes In the World Quickly

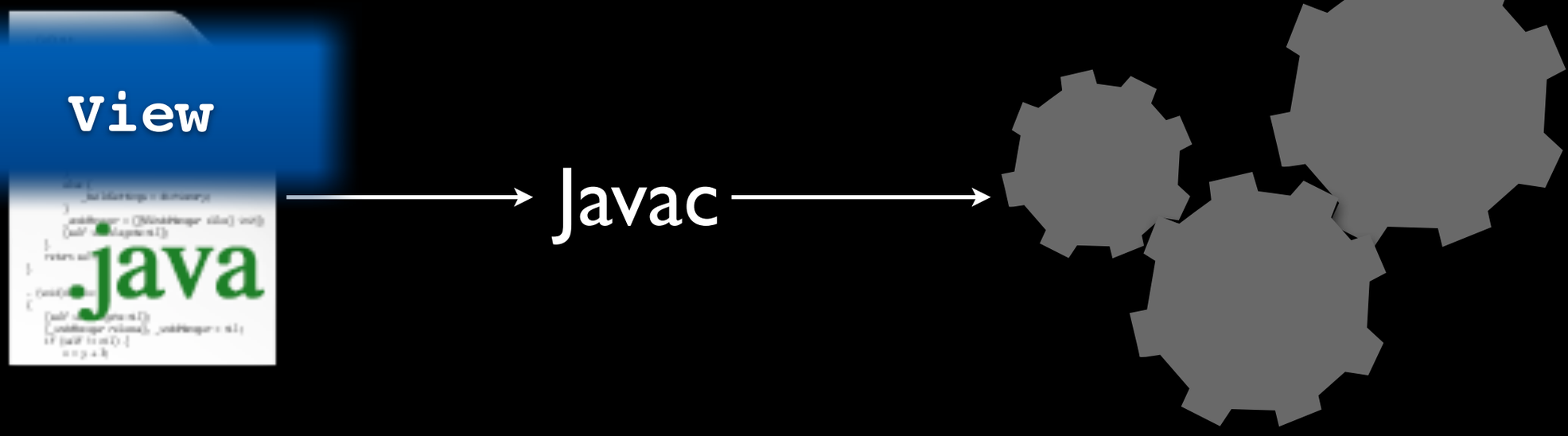
Not just Views

Frequently Fresh Views

Monitoring Programs

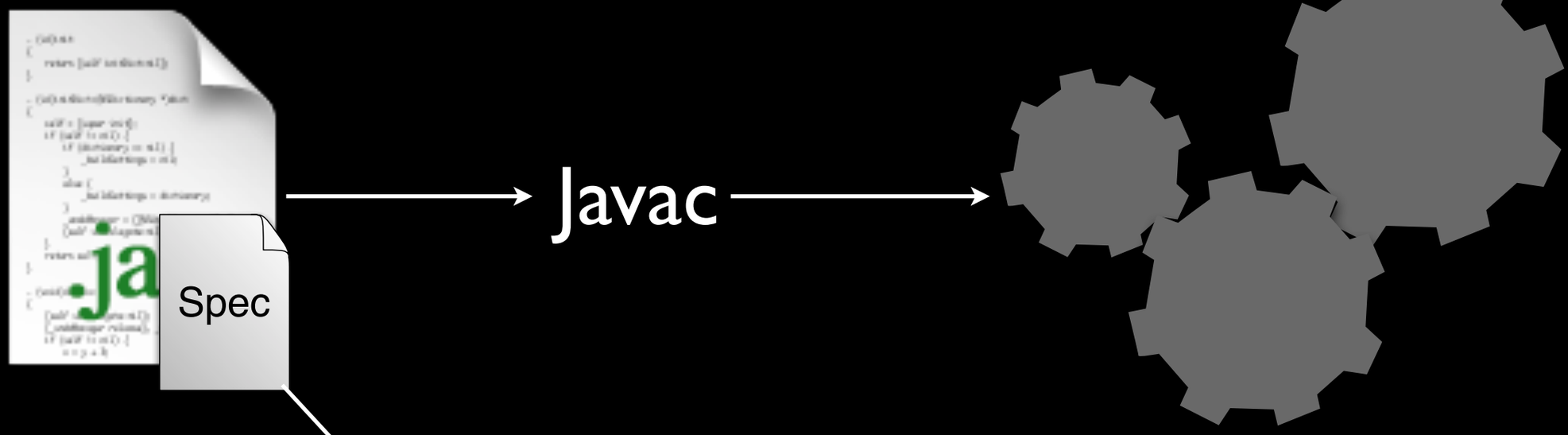


Monitoring Programs



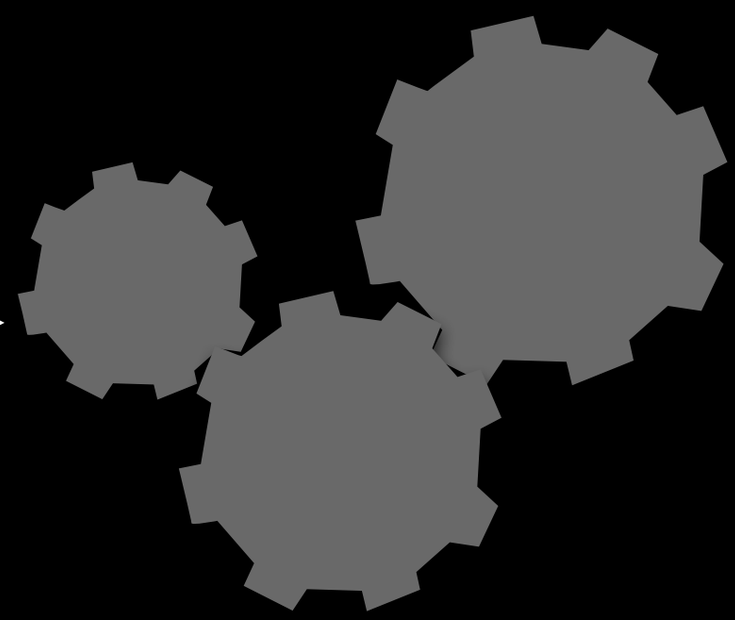
(The Current State of the Art)

Monitoring Programs



Monitoring Programs

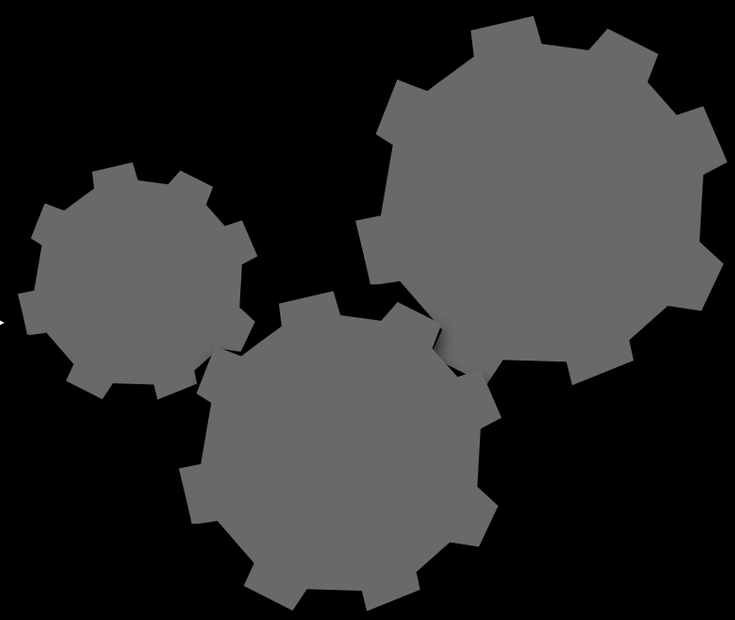
Javac



Spec

Monitoring Programs

Javac



Spec

View

The DBToaster Compiler

The Viewlet Transform

The Viewlet Transform

Use Auxiliary Views to Speed Up View Maintenance

The Viewlet Transform

Use Auxiliary Views to Speed Up View Maintenance

The Delta of a Query Can Be Materialized!

The Viewlet Transform

```
SELECT SUM(R.A * S.C)
FROM   R, S
WHERE  R.B = S.B
```

A Simple 2-Way Join Aggregate

The Viewlet Transform

```
q[] := SELECT SUM(R.A * S.C)
       FROM   R, S
       WHERE  R.B = S.B
```

A Simple 2-Way Join Aggregate

The Viewlet Transform

```
ON +R(∂A, ∂B):
 q[] +=  SELECT SUM(∂A * S.C)
         FROM   S
         WHERE  ∂B = S.B
```

Materialize and Incrementally Maintain The Query

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * ( SELECT SUM(S.C)
                FROM   S
                WHERE  ∂B = S.B )
```

Optimize

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * ( SELECT SUM(S.C)
                FROM   S
                WHERE  ∂B = S.B )
```

Optimize

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * ( SELECT S.B, SUM(S.C)
                FROM   S
                GROUP BY S.B )[∂B]
```

Optimize

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * mR[∂B]
```

```
mR[B] := SELECT S.B, SUM(S.C)
         FROM   S
         GROUP BY S.B
```

Extract and Materialize The Delta View

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * mR[∂B]
```

A Hash Map (indexed by S.B)

```
mR[B] := SELECT S.B, SUM(S.C)
         FROM   S
         GROUP BY S.B
```

Extract and Materialize The Delta View

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * mR[∂B]
```

```
ON +S(∂B, ∂C):
  mR[B] += SELECT ∂B,SUM(∂C)
```

Incrementally Maintain The Delta View

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * mR[∂B]
```

```
ON +S(∂B, ∂C):
  mR[∂B] += ∂C
```

Optimize

The Viewlet Transform

```
ON +R(∂A, ∂B):
  q[] += ∂A * mR[∂B]
  mS[∂B] += ∂A
```

```
ON +S(∂B, ∂C):
  mR[∂B] += ∂C
  q[] += ∂C * mS[∂B]
```

Repeat for the Other Deltas of the Query

The Viewlet Transform

The Viewlet Transform

- Take the Deltas
  - Optimize and Materialize Them
    - Take the Deltas
      - Optimize and Materialize Them
        - ...

The Viewlet Transform

- Take the Deltas
  - Optimize and Materialize Them
- Take the Deltas
  - Optimize and Materialize Them
    - ...

Performance

(and how we got there)

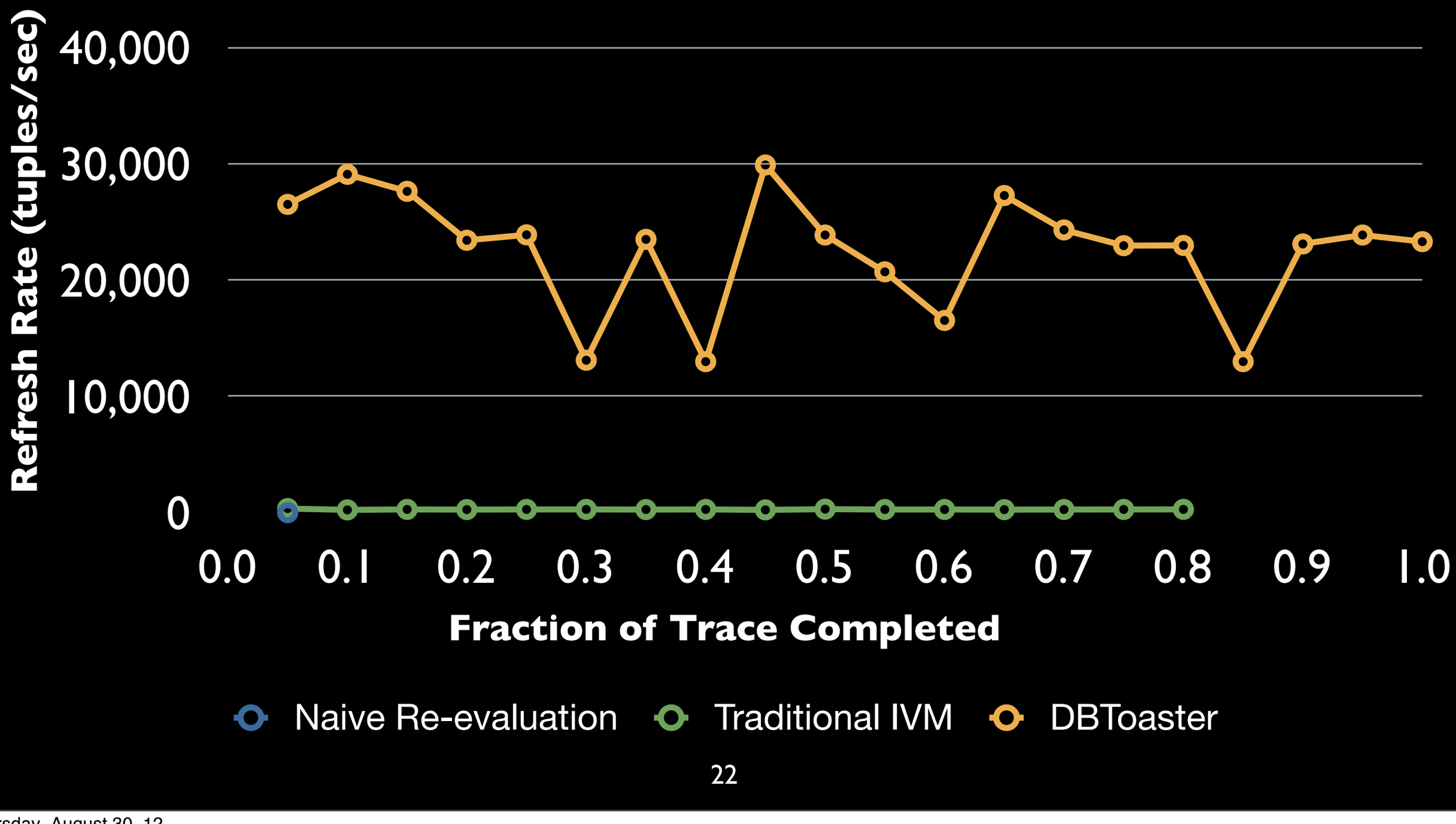
- TPC-H Workload
  - Simulated Realtime Data Warehouse
  - Update Stream Derived from TPC-H Gen

- Financial Benchmark
  - 24 hr Trace for an Actively Traded Stock.

TPCH: Q3

Refresh Rate

(3-Way Join)



TPCH: Q3

TPCH: Q3

Customer

CustID

Lineitem

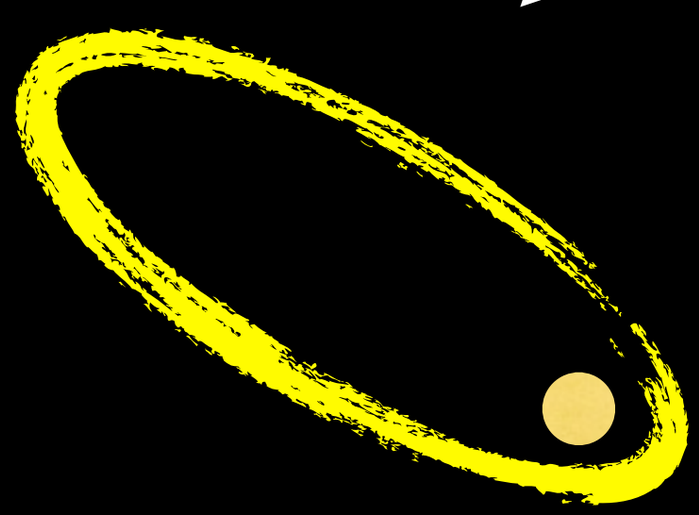
OrderID

The Δ for Orders

TPCH: Q3

Materialize Each Separately

Customer



CustID

Lineitem

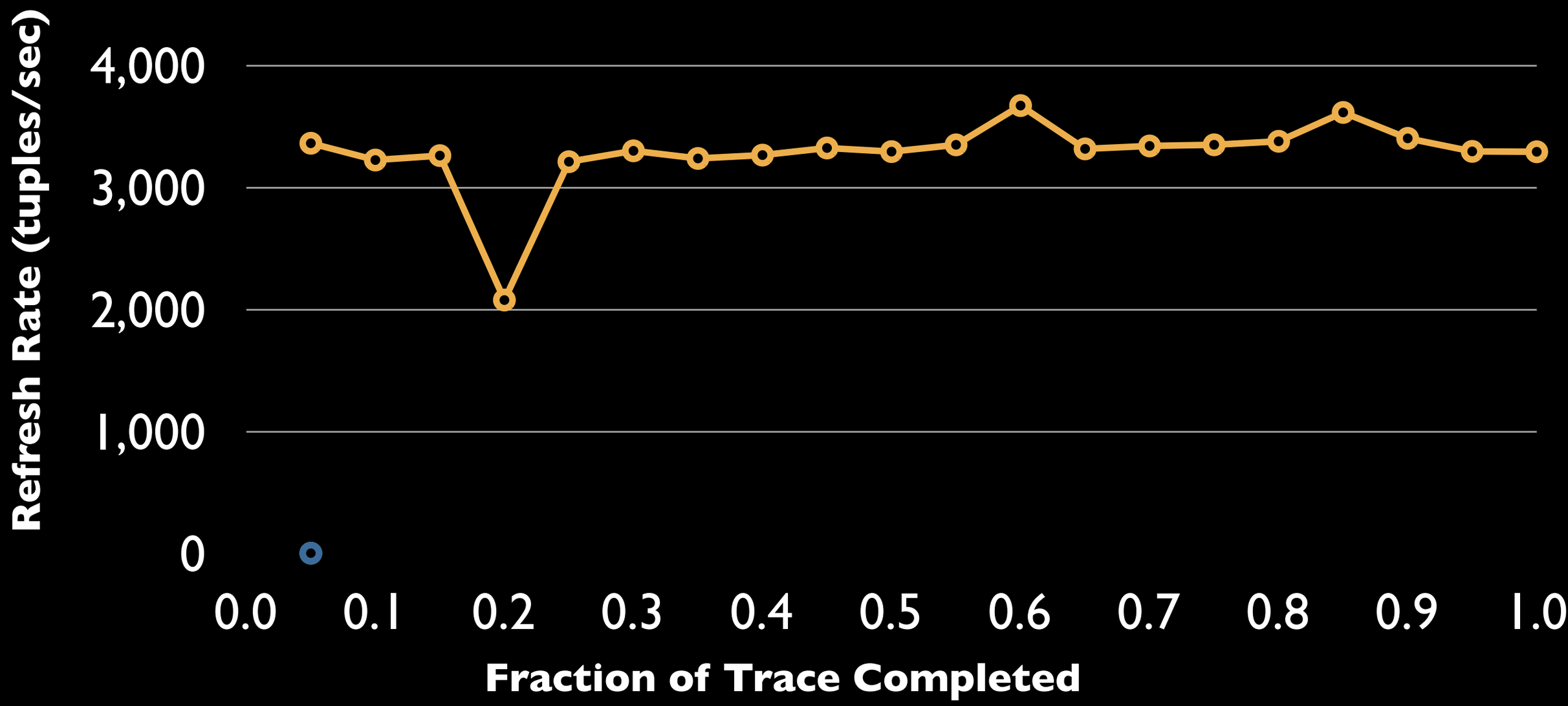
OrderID

The Δ for Orders

Financial: VWAP

Refresh Rate

(Self-join with Inequalitities)



Financial:VWAP

```
ON +BIDS(..., ∂price, ...)
    q[] += SELECT ...
           FROM BIDS b2
           WHERE ∂price > b2.price
```

Financial:VWAP

```
ON +BIDS(..., ∂price, ...)
    q[] += mB[∂price]
```

```
mB[∂price] := SELECT ...
              FROM BIDS b2
              WHERE ∂price > b2.price
```

Option 1: Create a Cache (best for VWAP)

Financial:VWAP

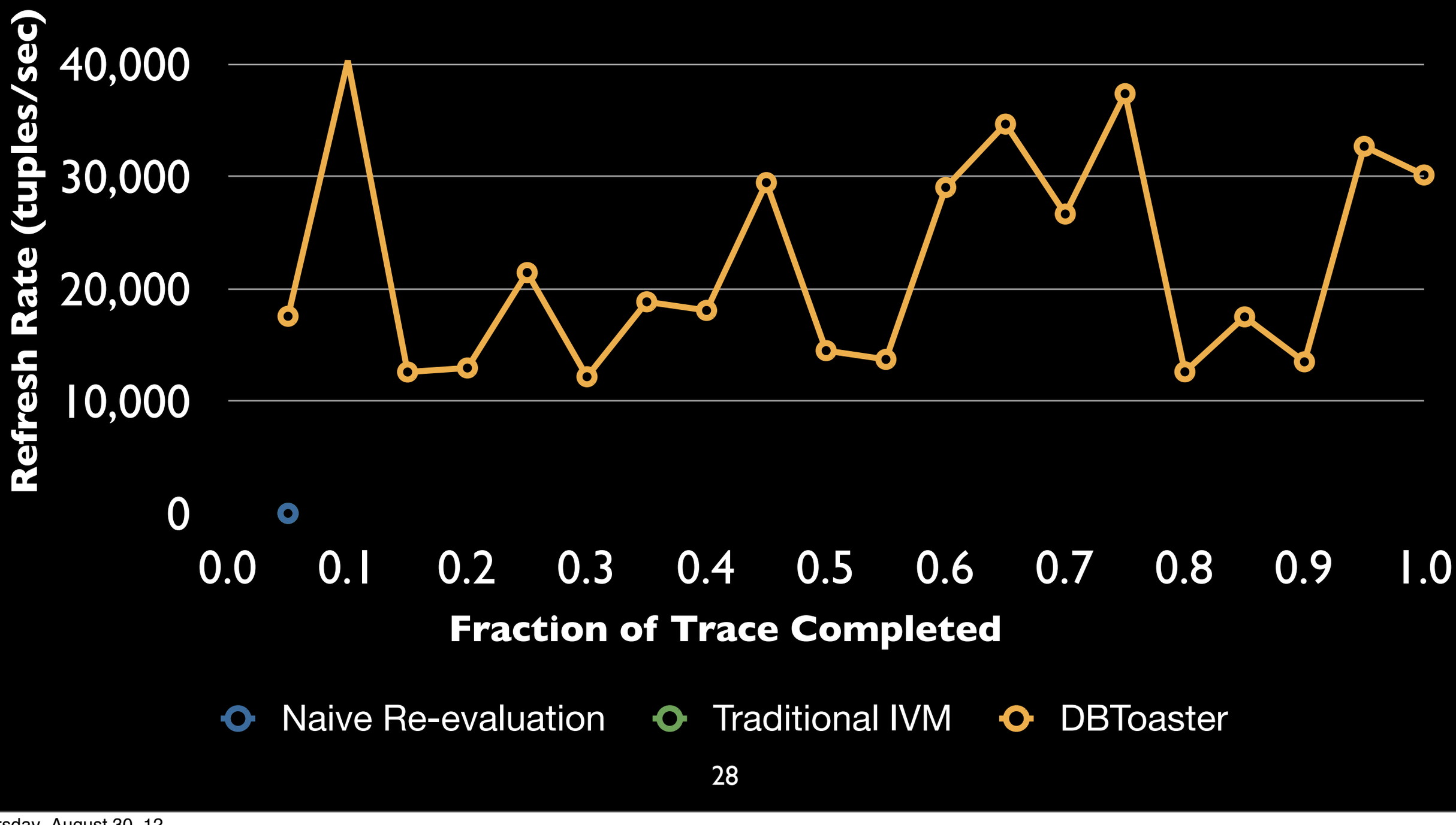
```
ON +BIDS(..., ∂price, ...)
    q[] += SELECT ...
           FROM  mB[]
           WHERE ∂price > b2.price
mB[] := SELECT ...
        FROM BIDS b2
```

Option 2: Defer Conditions Over Unsafe Variables

Financial: Pricespread

Refresh Rate

(Cross Product 'variance' Computation)



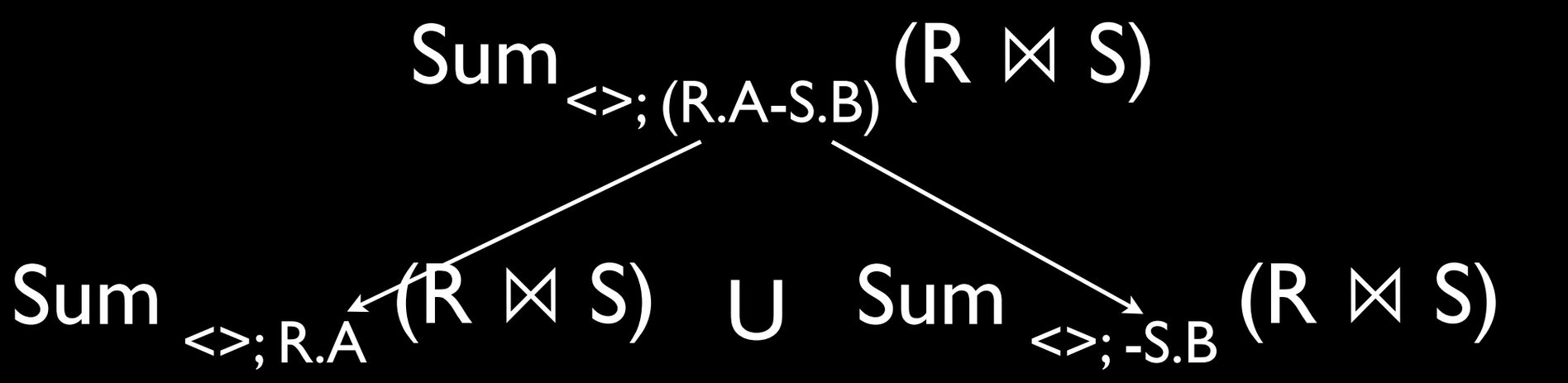
Financial: Pricespread

- 2-way Cross-Product with Nested Aggregates
  - IVM can't do better than Repeated re-evaluation.
- DBToaster wins on Data Representation Trickery!

Financial: Pricespread

$$\mathrm{Sum}_{<>;\,(R.A-S.B)}\,(R \bowtie S)$$

Financial: Pricespread



Financial: Pricespread

$$\mathrm{Sum}_{<>;\,(R.A-S.B)}\,(R \bowtie S)$$

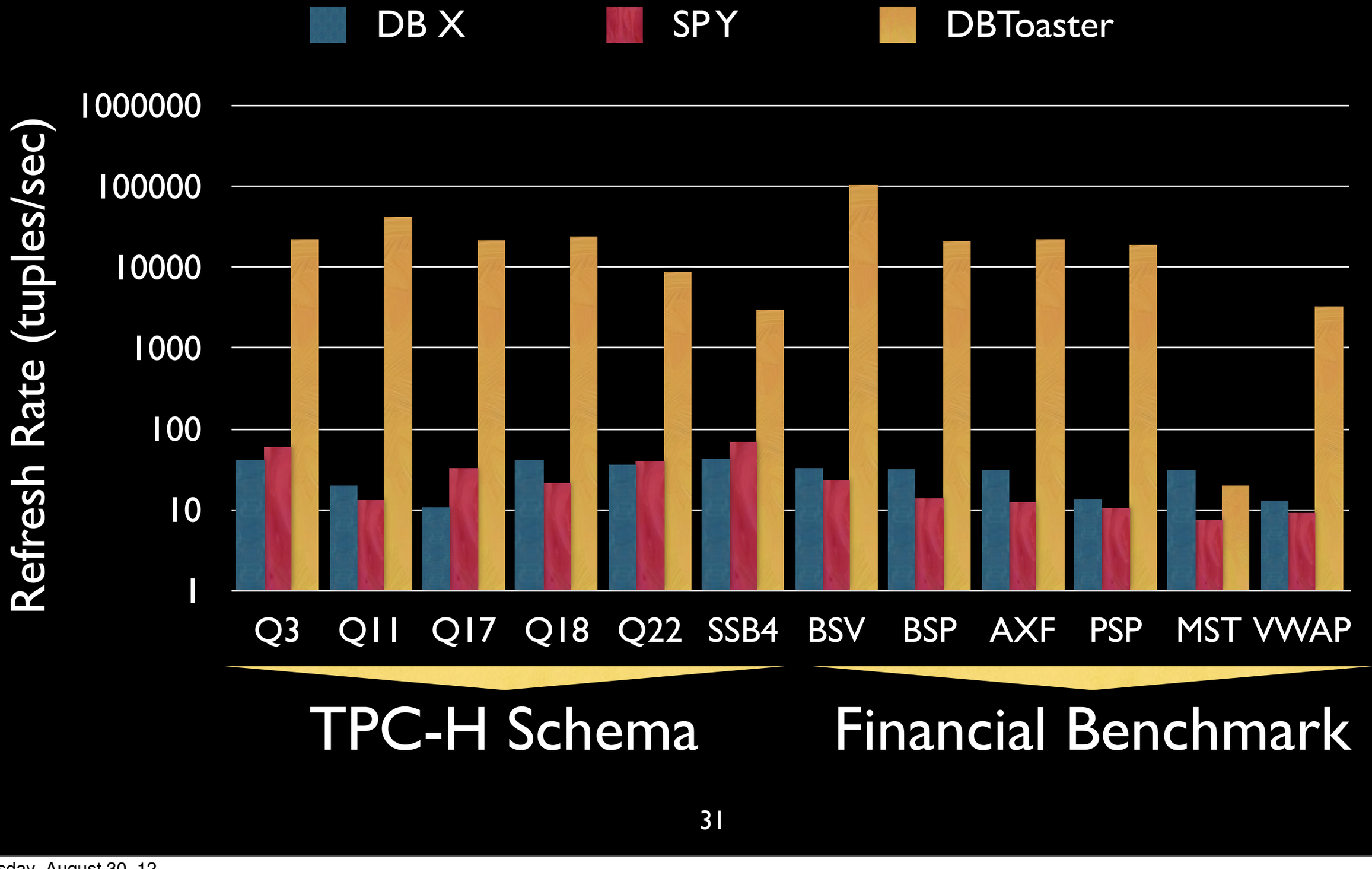
$$\mathrm{Sum}_{<>;\,R.A}\,(R \bowtie S) \;\cup\; \mathrm{Sum}_{<>;\,-S.B}\,(R \bowtie S)$$

$$\mathrm{Sum}_{<>;\,R.A}(R) \bowtie \mathrm{Sum}_{<>;\,1}(S)$$

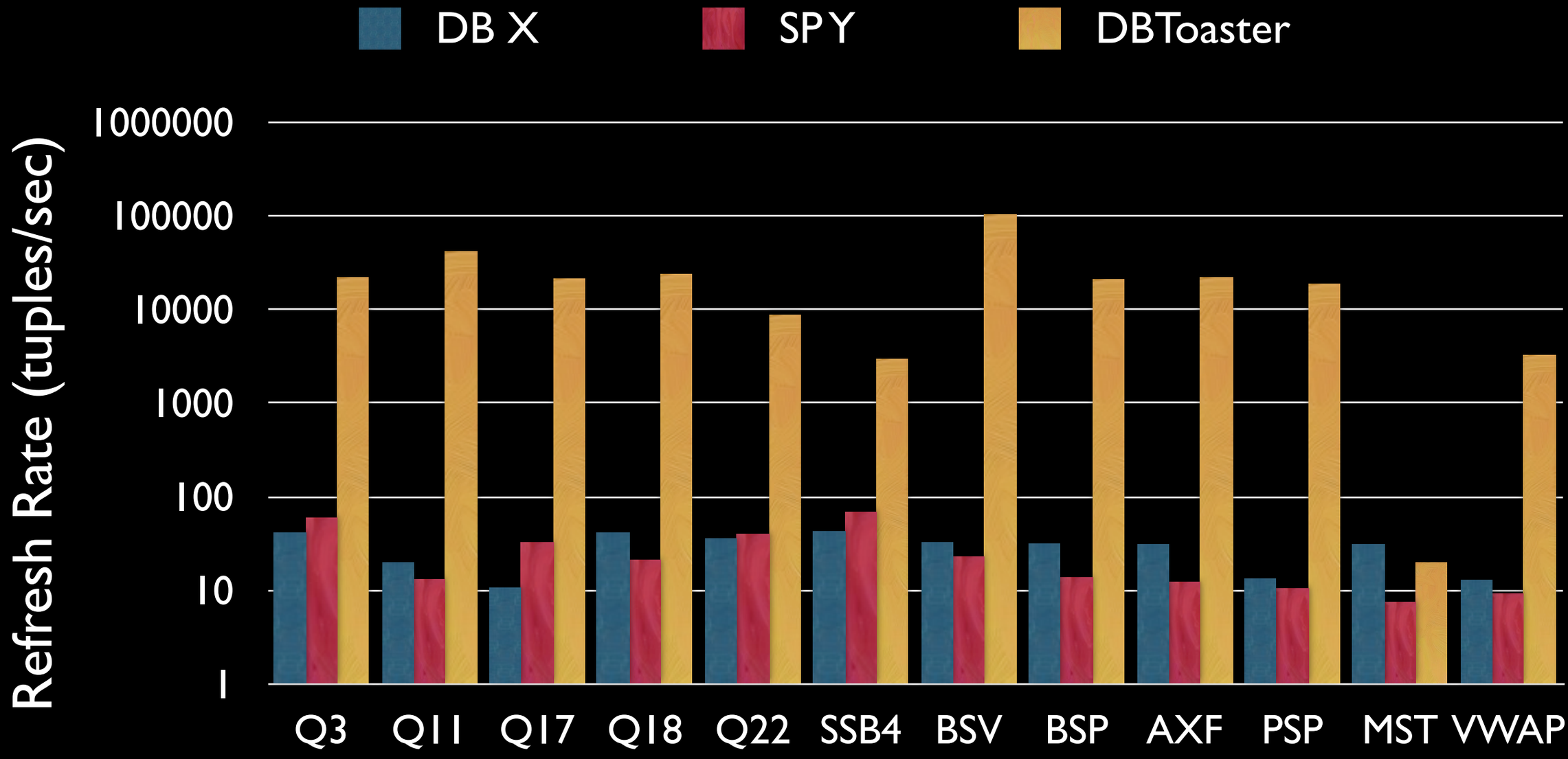
$$\cup$$

$$\mathrm{Sum}_{<>;\,1}(R) \bowtie \mathrm{Sum}_{<>;\,S.B}(S)$$

DBToaster vs Commercial Engines



DBToaster vs Commercial Engines



DBToaster is consistently 3 OOM better!

Limitations of Commercial Systems

- OLTP IVM is not designed for aggregating Low-Latency/Single-Tuple Updates.
- OLTP IVM doesn't support our full query workload.
- Stream Processors are not designed for rapidly changing long-lived data.

Limitations of Commercial Systems

- OLTP IVM is not designed for aggregating Low-Latency/Single-Tuple Updates.
- OLTP IVM doesn't support our full query workload.
- Stream Processors are not designed for rapidly changing long-lived data.

DBToaster opens entirely new application domains!

Conclusions

- The Viewlet Transform generates auxiliary views that make incremental maintenance fast.
- Materializing only part of an auxilliary view can sometimes be faster.
- DBToaster is commonly 3 OoM faster than Commercial Systems.

Conclusions

- The Viewlet Transform generates auxiliary views that make incremental maintenance fast.
- Materializing only part of an auxilliary view can sometimes be faster.
- DBToaster is commonly 3 OoM faster than Commercial Systems.

Download Now: http://www.dbtoaster.org

Thanks!

Oliver Kennedy

Yanif Ahmad

Christoph Koch

Milos Nikolic

Daniel Lupei

Amir Shaikhha

Andres Nötzli

Download Now: http://www.dbtoaster.org