DBTOASTER

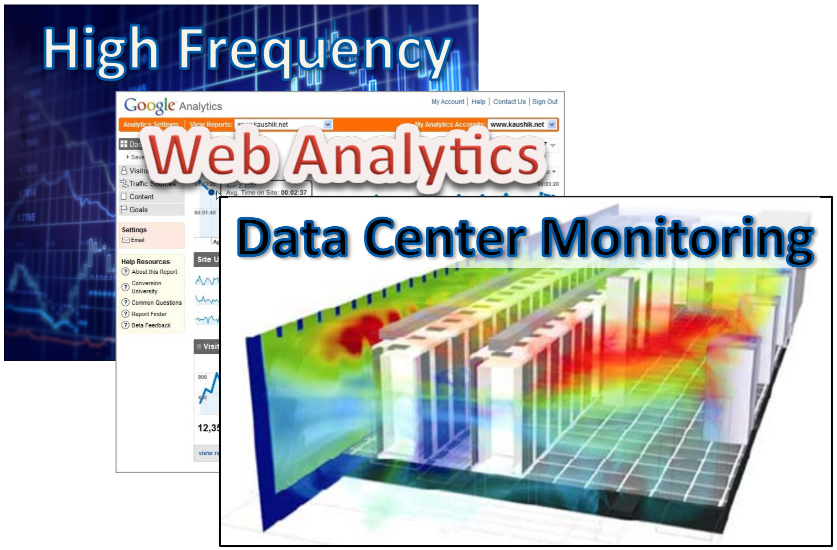
Do It Fast, Do It Incrementally

Christoph Koch, Yanif Ahmad, Oliver Kennedy,
Milos Nikolic, Andres Nötzli, Daniel Lupei, Amir Shaikhha

May 31st, 2013

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

What is this talk about?



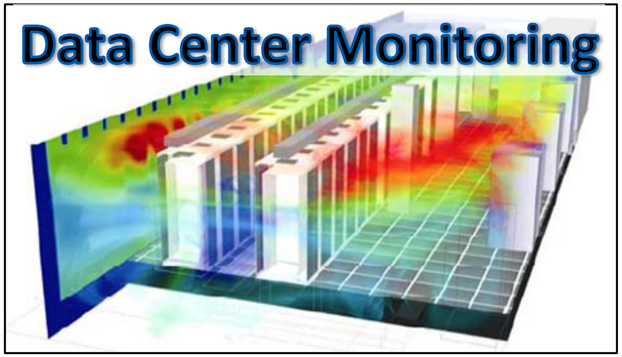
- Monitor state
- Views over current and historical data
- High update rates
- *Frequently* fresh views
- Customized engines

Declarative Specification (SQL Queries) → **DBToaster compiler** [Incremental Processing | Program Synthesis] → Runtime Engine

Outline

- Background and Motivation
- (Recursive) Incremental Processing
  - Compilation Example
- Experimental Results
- Next Directions

Update-Intensive Applications



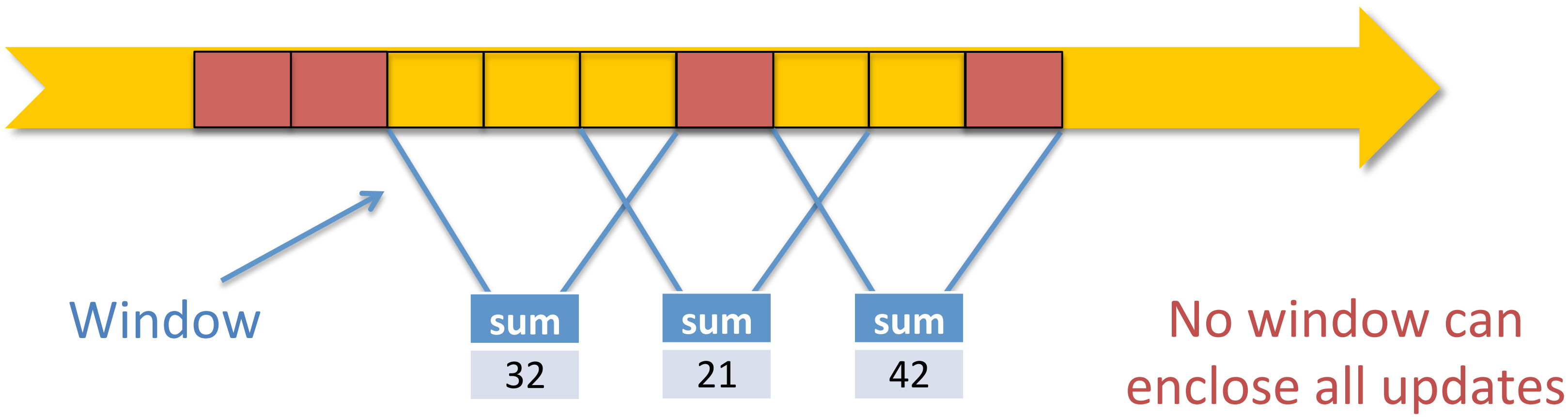
must sustain high update rates

Data Stream → Runtime Engine → Decision Support → Actions

Continuously arriving data
(e.g. buy/sell orders, sensor readings)

Continuously evaluated views
(e.g. over order books, active website users)

Data Stream Processing Systems

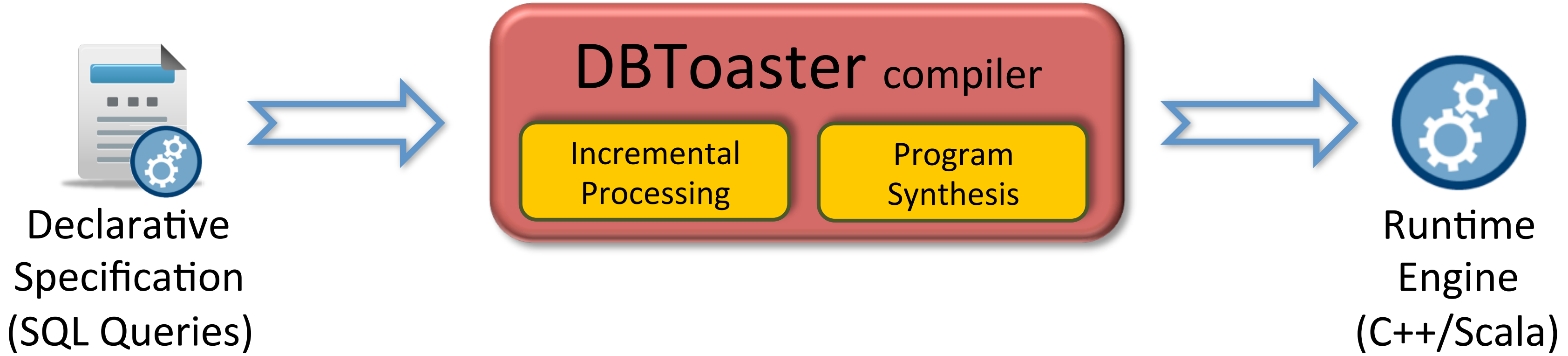


- Key architectural features
  - Continuous queries
  - Process queries over *windows* of input data
  - Assume append-only ordered inputs
- Problems:
  - Not designed for rapidly changing *long-lived* data
  - No "state-of-the-world" queries
  - No complex queries (e.g. nested aggregates)

Stream processing is unsuitable for update-intensive apps!

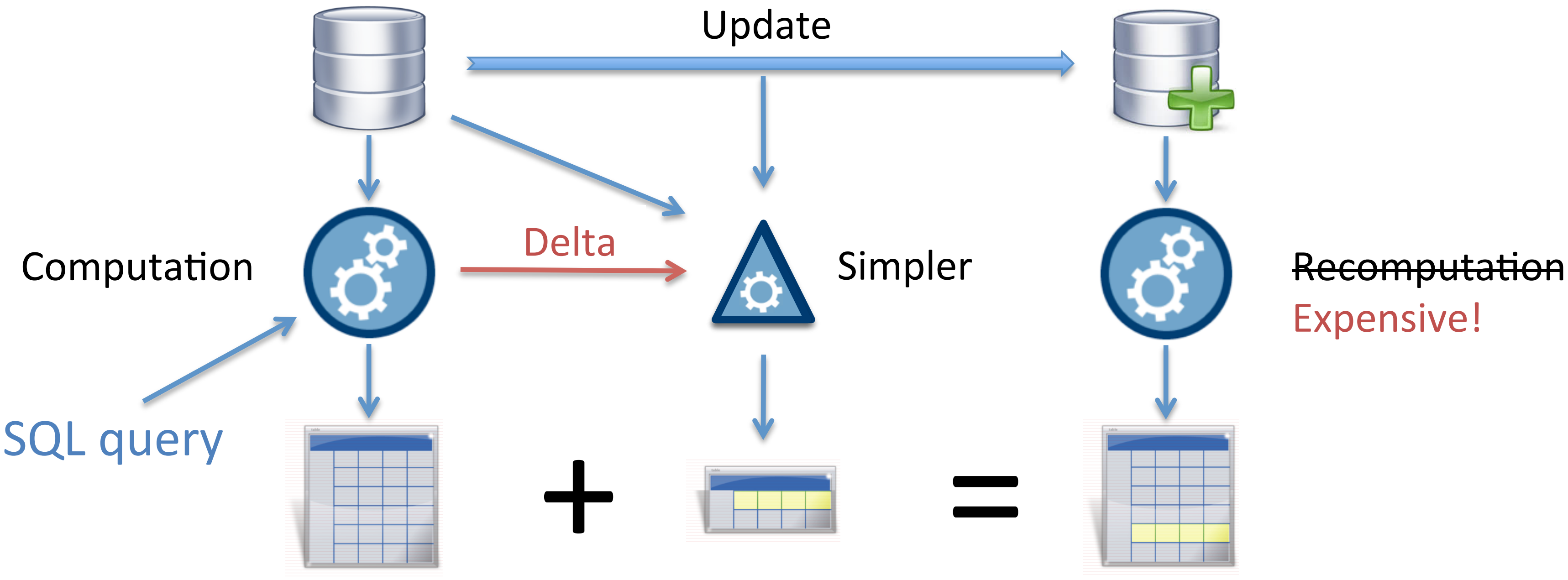
The DBToaster Project

- Automate the instantiation of special-purpose lightweight engines that are fast and scalable



- An aggressive query compilation technique
  - Turns queries into native code & eliminates all operators
  - The compiled engines incrementally maintain query results

Incremental Processing

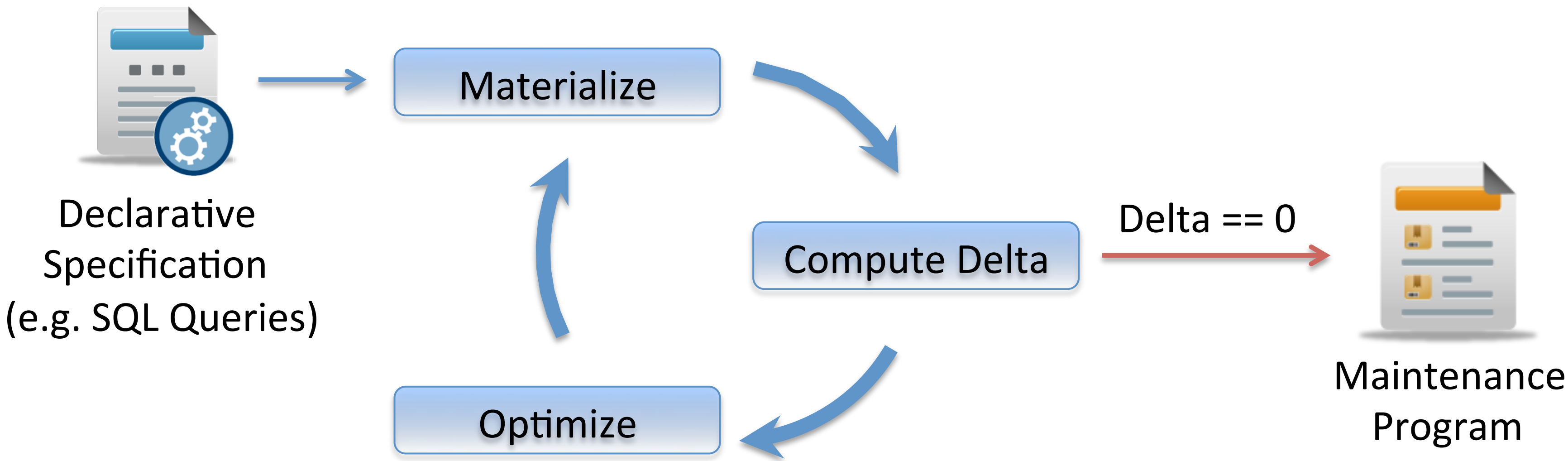


- Incremental View Maintenance in Databases
  - Implemented in major systems (Oracle, DB2, PostgreSQL, …)
  - Delta queries still evaluated using a classical query processing engine

DBToaster Compilation

Insight: Maintain query results recursively

- Compute deltas of deltas, deltas of deltas of deltas…



Compilation Example

R	A	B
	…	…

S	B	C
	…	…

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

A Simple 2-Way Join Aggregate

```
ON INSERT R(dA,dB) {

}
ON INSERT S(dB,dC) {

}
```

Maintenance Program

Compilation Example

R	A	B
	…	…

S	B	C
	…	…

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

1st step ⟼ Materialize

$q := SUM_{A*C;<>}(R \bowtie S)$

```
ON INSERT R(dA,dB) {

}

ON INSERT S(dB,dC) {

}
```

Compilation Example

R	A	B
	…	…
ΔR →	dA	dB

S	B	C
	…	…

```
q‘:= SELECT SUM(R.A * S.C)
     FROM   R + ΔR, S
     WHERE  R.B = S.B

q‘:= q +
     SELECT SUM(ΔR.A * S.C)
     FROM   ΔR, S
     WHERE  ΔR.B = S.B
```

$$q := SUM_{A*C;<>}(R \bowtie S)$$

```
ON INSERT R(dA,dB) {

}

ON INSERT S(dB,dC) {

}
```

2nd step ⟼ Compute Delta

* Slight abuse of SQL notation

Compilation Example

R	A	B
	…	…
ΔR →	dA	dB

S	B	C
	…	…

```
q := SUM_{A*C;<>}(R ⋈ S)

ON INSERT R(dA,dB) {
    q += ...
}

ON INSERT S(dB,dC) {

}
```

Incrementally maintain

```
q +=
     SELECT SUM(ΔR.A * S.C)
     FROM   ΔR, S
     WHERE  ΔR.B = S.B
```

2nd step ⟼ Compute Delta

Compilation Example

R	A	B
	…	…
ΔR →	dA	dB

S	B	C
	…	…

```
q +=
      SELECT SUM(ΔR.A * S.C)
      FROM   ΔR, S
      WHERE  ΔR.B = S.B
```

3rd step ⟼ Optimize

$q := SUM_{A*C;<>}(R \bowtie S)$

```
ON INSERT R(dA,dB) {
   q += ...

}

ON INSERT S(dB,dC) {


}
```

Compilation Example

R	A	B
	…	…
ΔR →	dA	dB

S	B	C
	…	…

No more join

```
q +=
    SELECT SUM(dA * S.C)
    FROM   S
    WHERE  dB = S.B
```

3rd step ⟼ Optimize

$q := SUM_{A*C;<>}(R \bowtie S)$

```
ON INSERT R(dA,dB) {
    q += ...
}

ON INSERT S(dB,dC) {

}
```

Compilation Example

R	A	B
	…	…
ΔR →	dA	dB

S	B	C
	…	…

```
q := SUM_{A*C;<>}(R ⋈ S)

ON INSERT R(dA,dB) {
    q += ...
}

ON INSERT S(dB,dC) {

}
```

Distributive law

```
q += dA *
     SELECT SUM(S.C)
     FROM   S
     WHERE  dB = S.B
```

3rd step ⟼ Optimize

Compilation Example

R	A	B
	…	…
ΔR →	dA	dB

S	B	C
	…	…

$q := SUM_{A*C;<>}(R \bowtie S)$

```
ON INSERT R(dA,dB) {
    q += ...

}

ON INSERT S(dB,dC) {


}
```

```
q += dA *
  ( SELECT SUM(S.C)
    FROM   S
    WHERE  dB = S.B )
```

3rd step ⟼ Optimize

Compilation Example

R	A	B
	…	…
ΔR →	dA	dB

S	B	C
	…	…

```
q := SUM_{A*C;<>}(R ⋈ S)

ON INSERT R(dA,dB) {
    q += ...

}

ON INSERT S(dB,dC) {


}
```

```
q += dA *
  ( SELECT S.B, SUM(S.C)
    FROM   S
    GROUP BY S.B          ) [dB]
```

3rd step ⟼ Optimize

Compilation Example

R	A	B
	…	…
ΔR →	dA	dB

S	B	C
	…	…

A Hash Map (indexed by S.B)
↓

```
q += dA * mR[dB]
mR[B] := SELECT S.B, SUM(S.C)
         FROM   S
         GROUP BY S.B
```

$q := SUM_{A*C;<>}(R \bowtie S)$

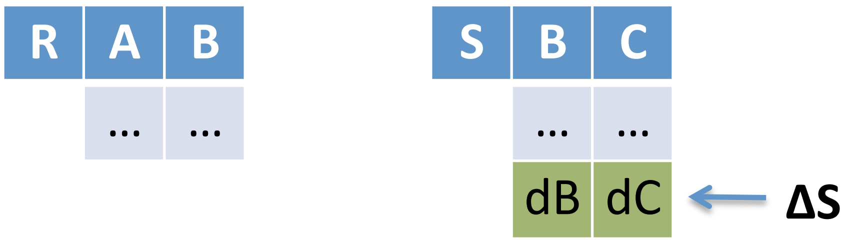
$mR[B] := SUM_{C,<B>}S$

```
ON INSERT R(dA,dB) {
    q += dA * mR[dB]
}

ON INSERT S(dB,dC) {
}
```

Materialize ⟼ Compute Delta ⟼ Optimize

Compilation Example




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

$q := SUM_{A*C;<>}(R \bowtie S)$

$mR[B] := SUM_{C,<B>}S$

```
ON INSERT R(dA,dB) {
    q += dA * mR[dB]

}

ON INSERT S(dB,dC) {


}
```

Materialize ⟼ Compute Delta ⟼ Optimize

Compilation Example

R	A	B
	…	…

S	B	C
	…	…
	dB	dC

← ΔS

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

mR[dB] += dC
```

$q := SUM_{A*C;\langle\rangle}(R \bowtie S)$

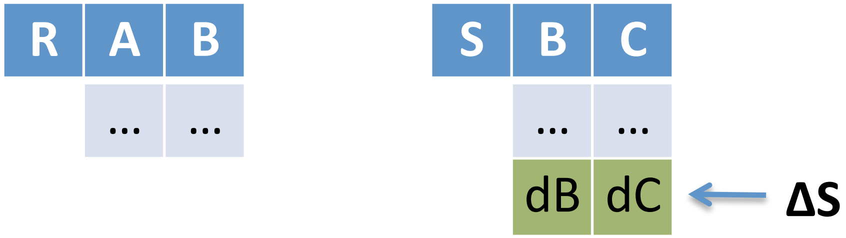
$mR[B] := SUM_{C,\langle B\rangle} S$

```
ON INSERT R(dA,dB) {
    q += dA * mR[dB]
}

ON INSERT S(dB,dC) {
    mR[dB] += dC
}
```

Materialize ⟼ Compute Delta ⟼ Optimize

Compilation Example




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

Minimal memory overhead!

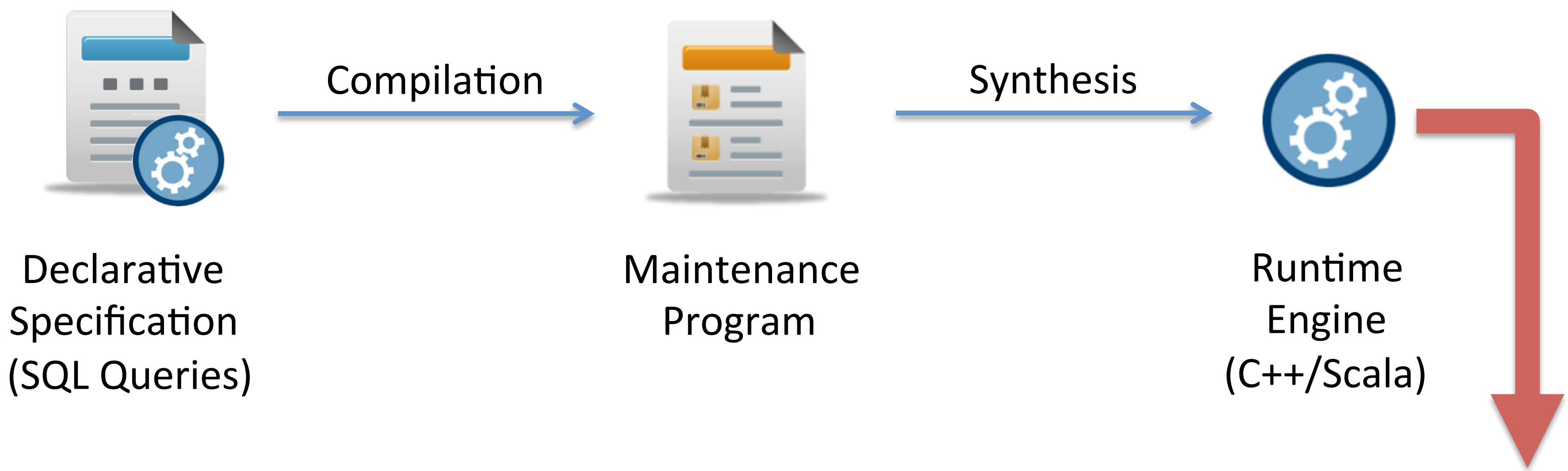
```
q := SUM_{A*C;<>}(R ⋈ S)
mR[B] := SUM_{C,<B>}S
mS[B] := SUM_{A,<B>}S

ON INSERT R(dA,dB) {
    q += dA * mR[dB]
    mS[dB] += dA
}

ON INSERT S(dB,dC) {
    mR[dB] += dC
    q += dC * mS[dB]
}
```

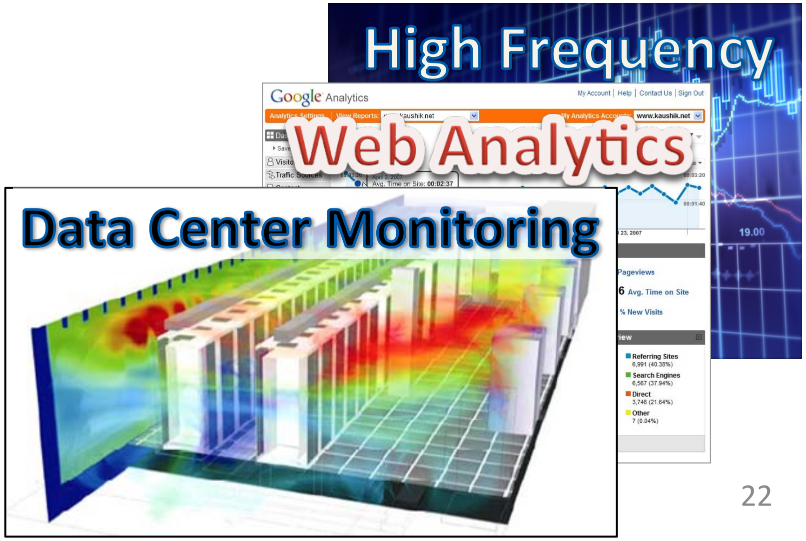
The triggers run in constant time!

DBToaster Workflow



Extremely easy to build runtimes!

Reduced development cost!

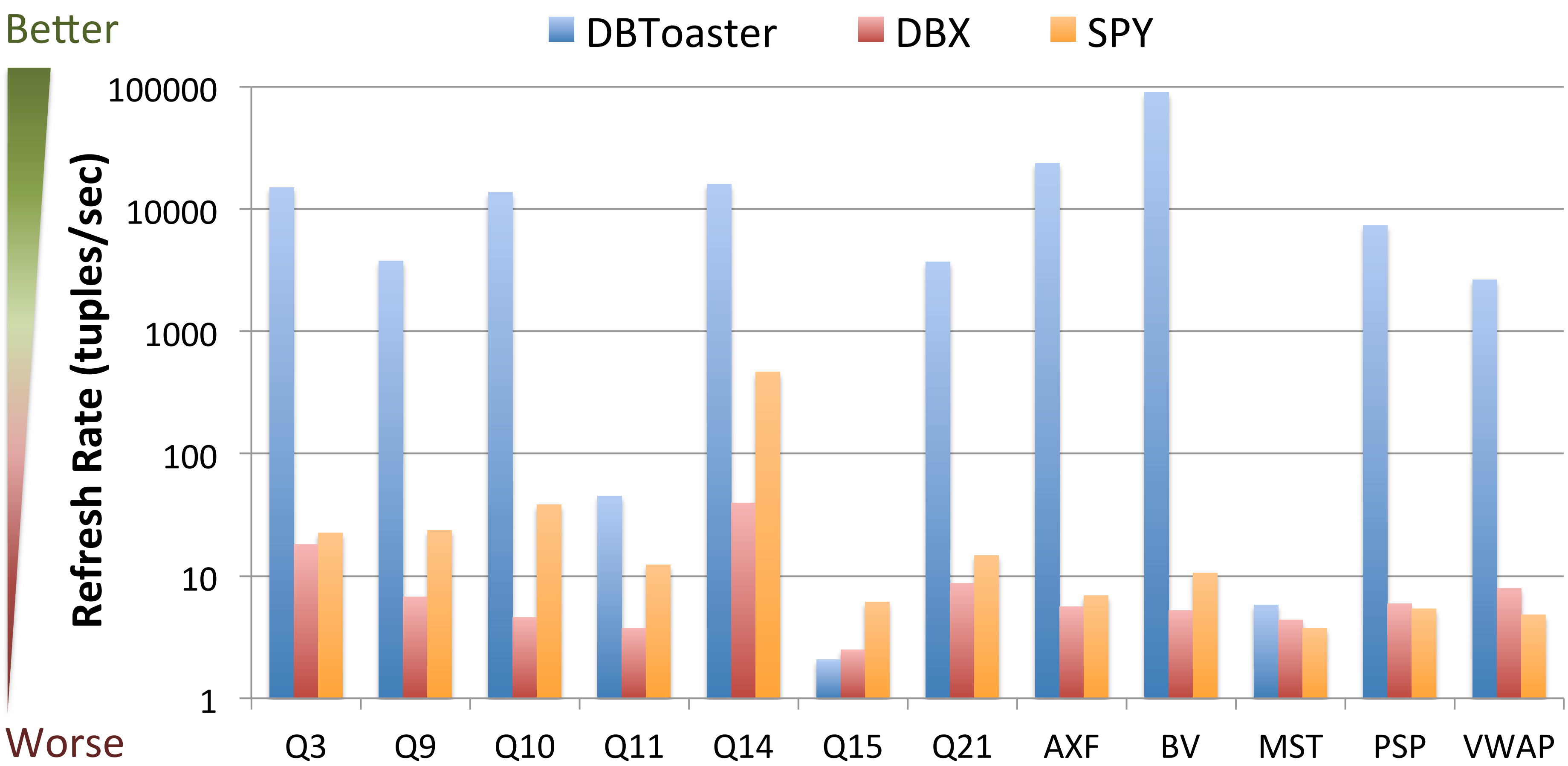


Experimental Setup

- TPC-H Workload
  - Simulated realtime data warehouse
  - Update stream derived from TPC-H Gen

- Financial Benchmark
  - 24hr trace for an actively traded stock

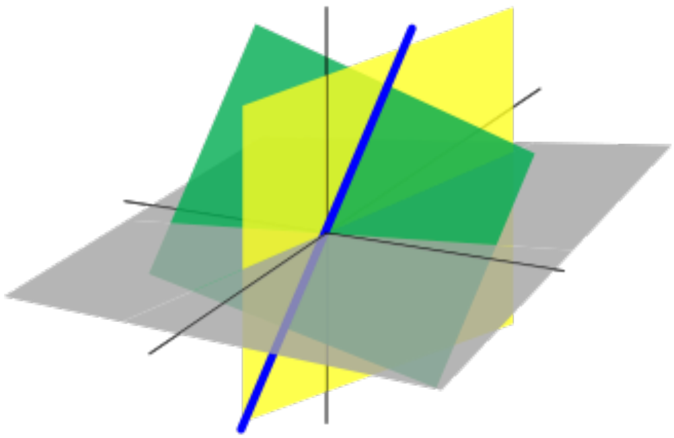
DBToaster vs Commercial Engines



DBToaster achieves up to 4 OOM speedup!

Incremental Linear Algebra

- Applications: Machine learning, big-data analytics
- Goal: Eliminate expensive operations (e.g. matrix multiplication)
- Challenges:
  - Global program optimization
  - New building blocks (A^T, A^{-1}, SVD, etc.)
- Domain-specific data representation
  - Array data model, dense vs. sparse matrices
  - Optimizing data layout, I/O sharing



DBTOASTER Ecosystem

- 4 years of research
- From SQL queries to runtime engines
  - Novel recursive compilation technique
  - Can handle nested aggregates
- Up to 4 OOM faster than commercial systems

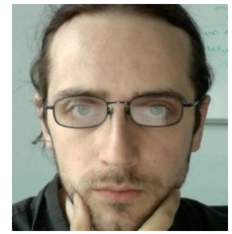
- DBToaster opens entirely new application domains!

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

Thanks!

Christoph Koch
(EPFL)

Yanif Ahmad
(JHU)



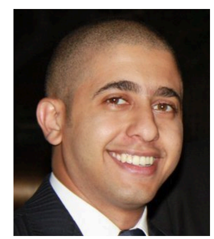
Oliver Kennedy
(UB)

Milos Nikolic
(EPFL)

Andres Nötzli
(EPFL)

Daniel Lupei
(EPFL)

Amir Shaikhha
(EPFL)



Mohammed
El Seidy
(EPFL)

Mohammad
Dashti
(EPFL)

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