

A Monte Carlo Approach to Managing Uncertain Data

Peter J. Haas

Kevin Beyer

Vuk Ercegovic

Rajasekar Krishnamurthy

Eirinaos Michaelis*

Bo Shekita

Shiv Vaithyanathan

IBM Almaden Research Ctr.

*UC Berkeley

Chris Jermaine

Ravi Jampani

Luis Perez

Mingxi Wu

Fei Xu

U. Florida Gainesville

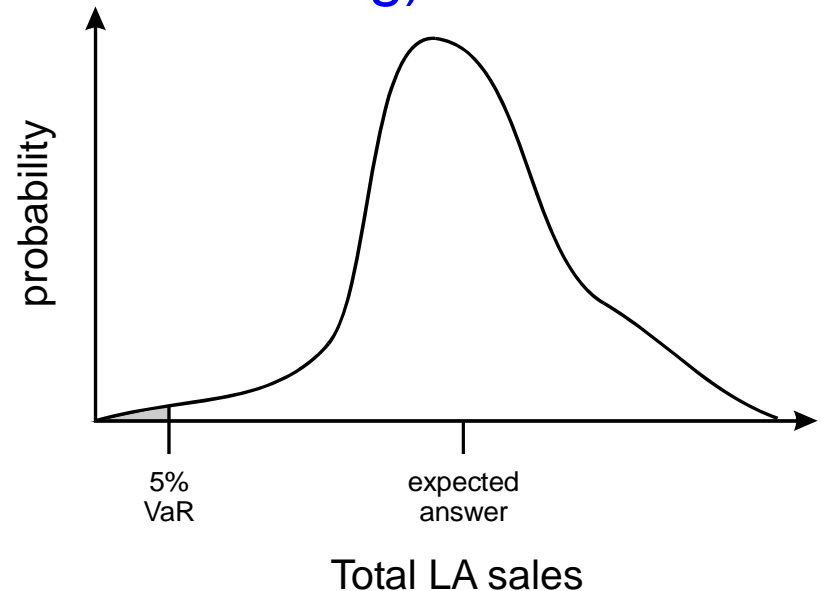
Outline

- Motivation via examples
- The MCDB system (SIGMOD 2008)
- Uncertainty in the Cloud
- An end-to-end scenario
- Future directions

Risk Due to Data Uncertainty

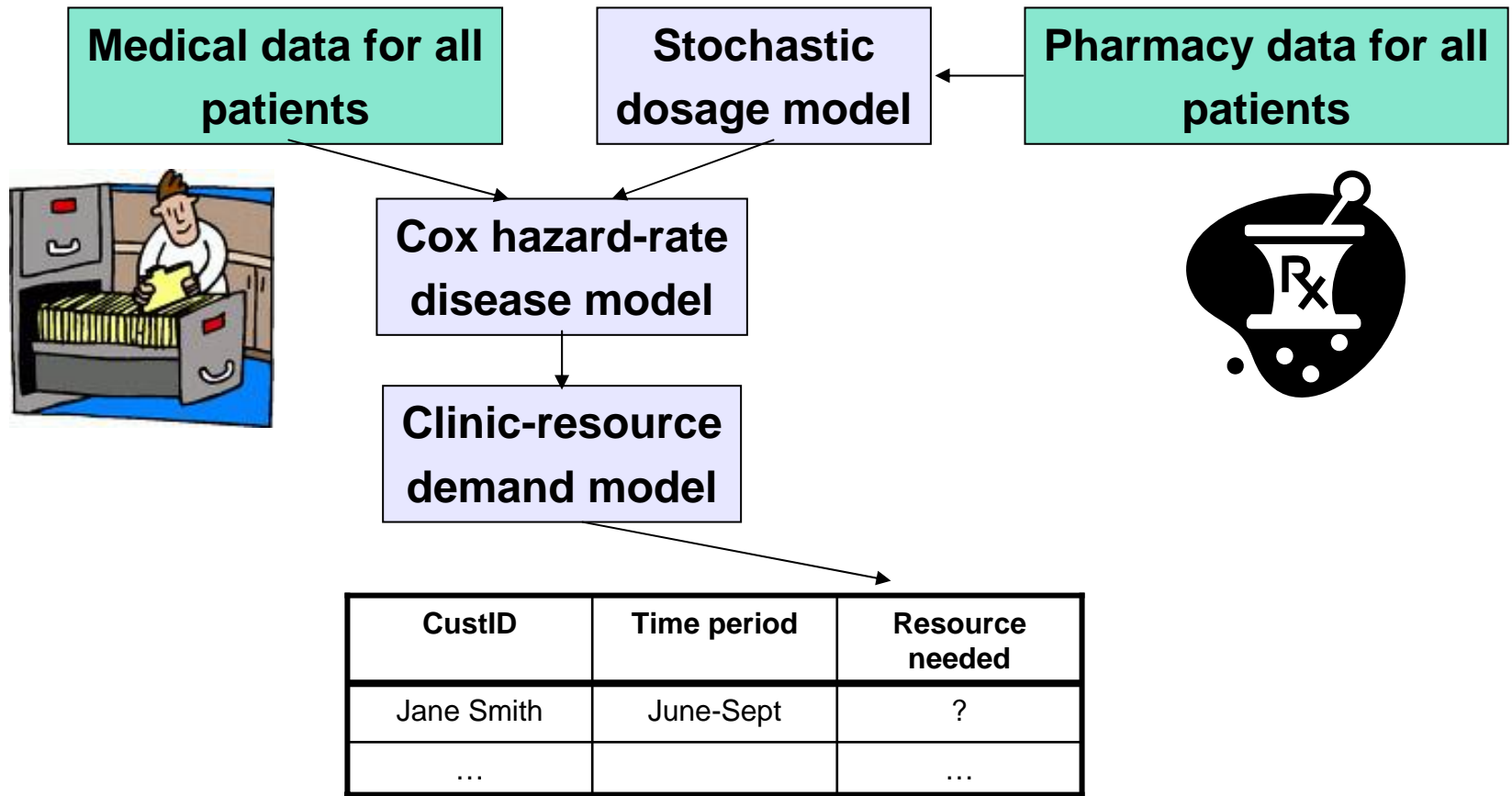
- Key uncertainty sources: info integration & extraction
- Ex: Value of assets (for financial reporting, compliance, business-process monitoring)

```
SELECT SUM (s.amount)
FROM SALES s, CUST c
WHERE s.ID = c.ID
      AND c.city = 'Los Angeles'
```



- Ex: ERP
 - # OS experts needed for help desk
 - Based on (uncertain) data from last year
 - Provide principled **safety factor**

Clinic-Capacity Risk



Logistics Under Uncertainty

- Retailer: ship from warehouses to outlets today or next week?
- Deterministic tables

ITEM_ID	QUANTITY
curtains	50
...	...

ITEM_ID	QUANTITY
curtains	20
...	...

ITEM_ID	Price
curtains	\$120
...	...

- Random tables

CUST_ID	ITEM_ID	QUANTITY
Smith	curtains	?
...

CUST_ID	ITEM_ID	QUANTITY
Smith	curtains	?
...

- Queries:

```
SELECT SUM (c.price * s.quantity)
FROM SALES_W_SHIP s,
CUR_PRICE c
WHERE c.ITEM_ID = s.ITEM_ID
```

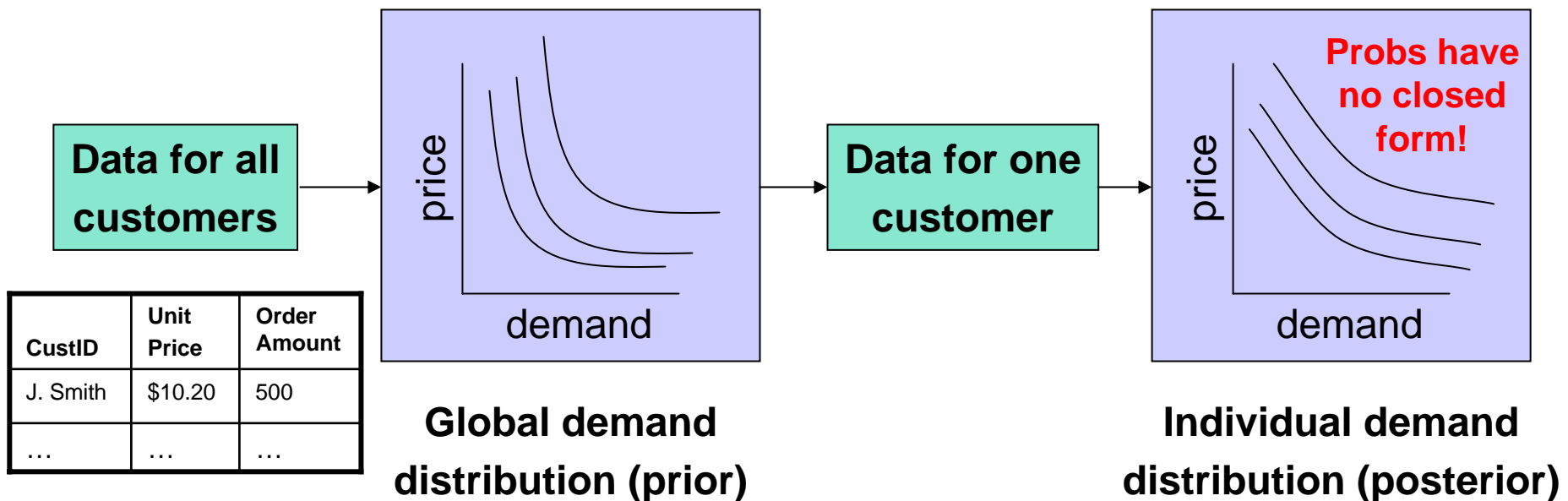
```
SELECT SUM (c.price * s.quantity)
FROM SALES_WO_SHIP s,
CUR_PRICE c
WHERE c.ITEM_ID = s.ITEM_ID
```

- Issues:
 - Complicated statistical models for purchase quantity (how to integrate in DB?)
 - Uncertainty (random tables) depend **dynamically** on **huge** number of parameters

Pricing: Individual Demand Curves

“A wide variety of business strategies leads one to analyze data at the most detailed level possible”

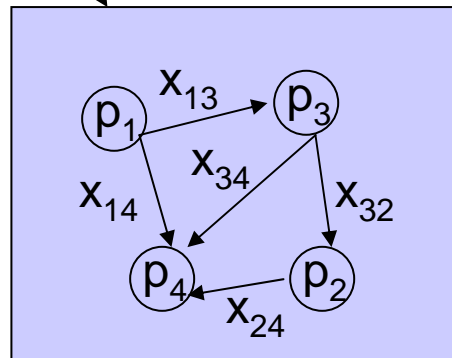
– Richard Winter, BI Network Report, April 2008



- Can analyze arbitrary dynamic customer segments when determining effect of price increase

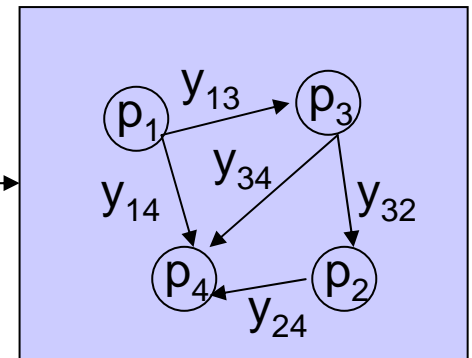
Individual Click Behavior (EBay)

Click data for all EBay customers



**Global Markov model
distribution (Dirichlet prior)**

Data for one customer



**Individual Markov model
distribution (posterior)**

- Can analyze arbitrary dynamic customer segments when determining effect of changing EBay pages

Portfolio Values

Customer

CustID	OptionID	NumShares
John Smith	23	50
...		...

EuroCallOptions

OptionID	InitVal	r	a	dt	StrikeP	Val
23	\$2.35	0.8	1.01	0.0001	\$4.00	?

Modified Black-Scholes model for European call option (simulation approximation):

$$V(t + \Delta t) = V(t) + rV(t)\Delta t + \left(a\sqrt{V(t)} \right) V(t)\sqrt{\Delta t}Z_j$$

$$\text{Value} = \max(V(t_{\text{final}}) - S, 0)$$

Sample from
Normal dist'n



```
SELECT SUM (c.NumShares * o.Val)
FROM Customer c, EuroCallOptions o
WHERE c.OptionID = o.OptionID
      AND c.CustID = 'John Smith'
```

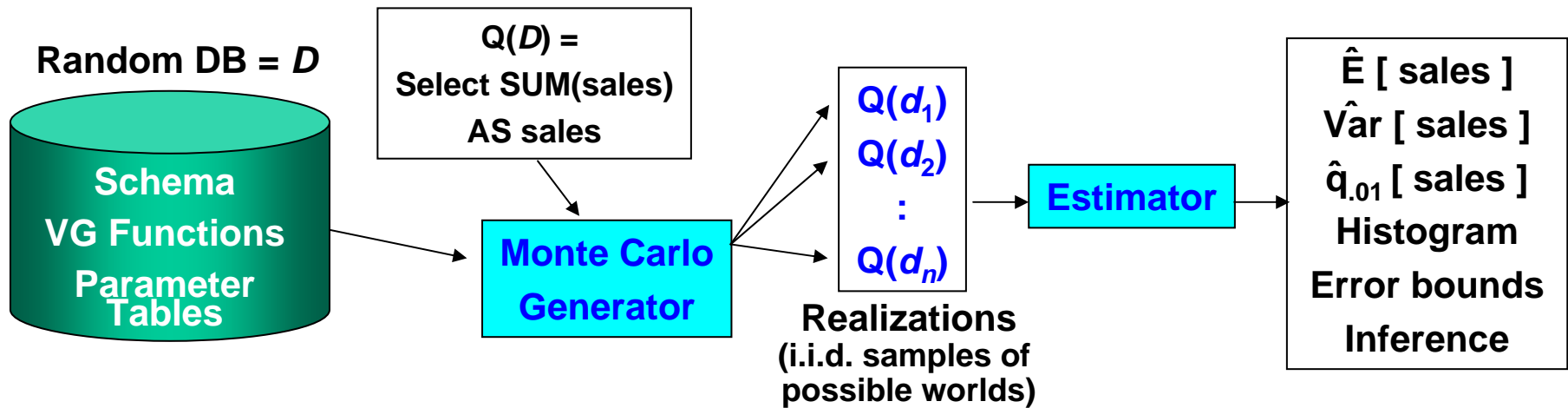
Motivation Summary

- Customer needs
 - Risk assessment
 - Decision-making under uncertainty
- Uncertainty models
 - Both **data**-based and **model**-based
 - Highly **heterogeneous** and complex
 - Often depend **dynamically** on **large #** of parameters
 - Correlation matrices for multivariate distributions
 - Customer purchase histories
 - Probabilities perpetually changing
- Queries
 - Complicated SQL aggregation queries
 - Subqueries, DISTINCT clauses, ...
- What-if and sensitivity analysis are **crucial**
 - Input probabilities are **not** precise

Outline

- Motivation via examples
- The MCDB system (SIGMOD 2008)
- Uncertainty in the Cloud
- An end-to-end scenario
- Future directions

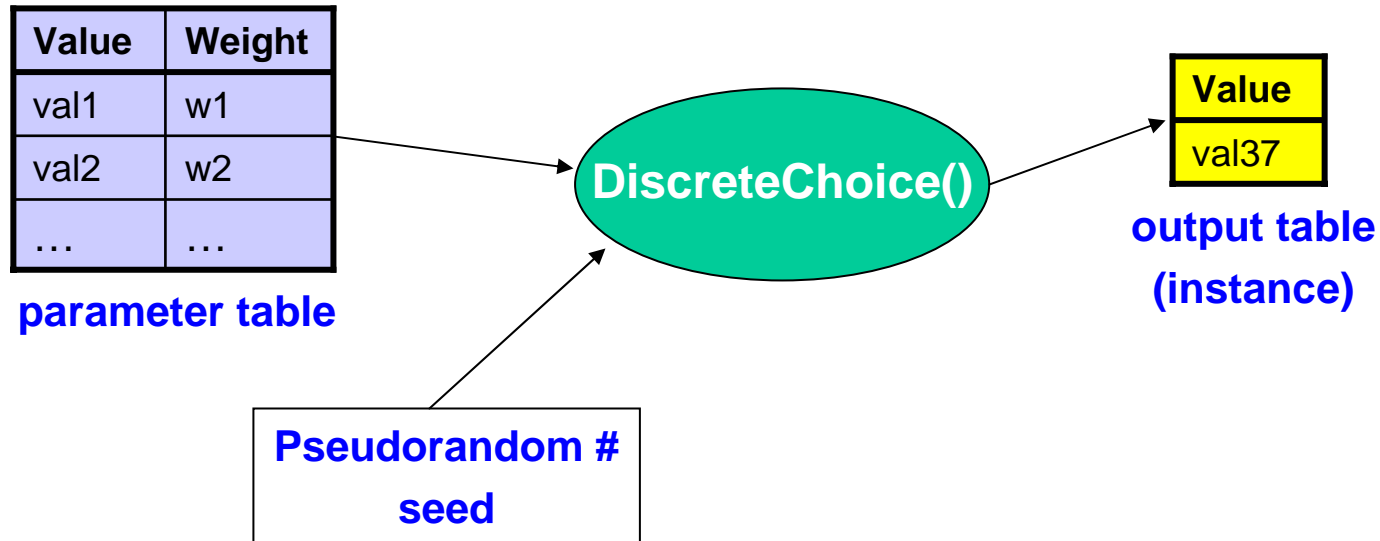
The MCDB System



- **Nice features of MCDB approach**

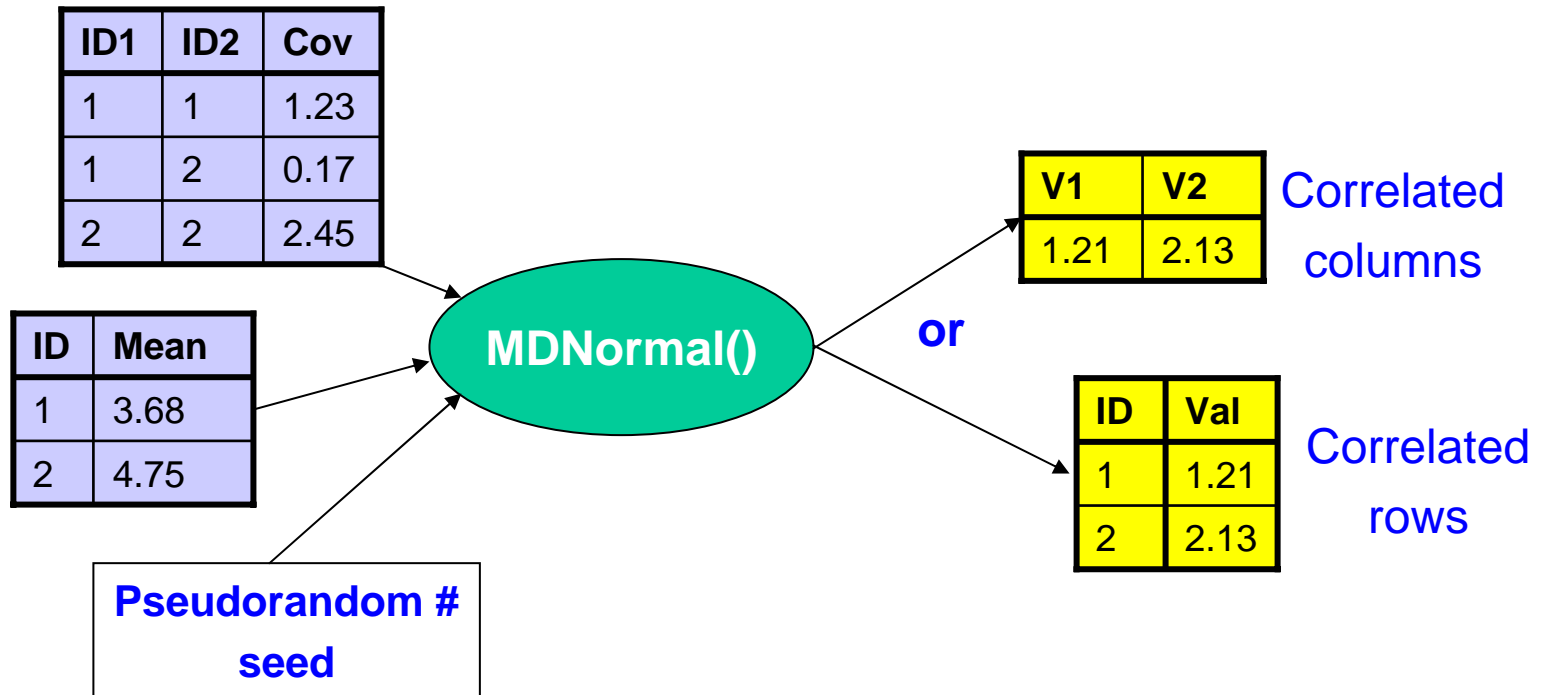
- Flexible and **extensible** uncertainty model
 - Wrapper around standard relational model
 - Can capture extended relational models (Trio, MayBMS, Mystiq,...)
 - Can capture arbitrarily complex correlations, continuous data
 - Can capture dynamic, highly parameterized distributions
 - Can bring complex stochastic models to data (no extraction needed)
- Can handle arbitrary SQL queries
- What-if analysis, sensitivity analysis, data updates are easy₁₁

VG Functions



- **Used to generate instances of values in random tables**
 - Library of standard functions: `DiscreteChoice`, `Normal`, `Poisson`, ...
 - Can define custom functions (similar to UDFs)
 - Parameter tables are standard relational tables (can index, etc.)
 - What-if and sensitivity analyses are easy
 - Run ordinary SQL queries to update parameter tables
 - Modify schema definitions

VG Functions and Correlation



Schema Syntax

```
CREATE TABLE RAND_CUST (CID, GENDER, MONEY, LIVES_IN) AS
FOR EACH d in CUST
WITH MONEY AS Gamma(
  (SELECT n.SHAPE FROM MONEY_SHAPE n WHERE n.CID = d.CID),
  (SELECT sc.SCALE FROM MONEY_SCALE sc WHERE sc.REGION =
    d.REGION),
  (SELECT SHIFT FROM MONEY_SHIFT)
)
WITH LIVES_IN AS DiscreteChoice (
  (SELECT c.NAME, c.PROB
    FROM CITIES c
    WHERE c.REGION = d.REGION)
)
SELECT d.CID, d.GENDER, m.VALUE, I.VALUE
FROM MONEY m, LIVES_IN I
```

Query Processing

- Naïve approach
 - Repeatedly instantiate DB and run query
 - Has **horrible** performance
- MCDB approach
 - Execute query plan **once**
 - Process **tuple bundles** instead of tuples
 - Keep bundles in **compressed** form whenever possible
 - Use pseudorandom **seeds** for compression
 - Apply selections early to compressed bundles

Tuple Bundles

- Array of N tuples with fixed schema
 - $N = \#$ Monte Carlo reps

(Jane, Smith, 20)

(Jane, Smith, 21)

--

(Jane, Smith, 21)

← Tuple bundle (4 reps)

isPresent

(Jane, Smith, (20,21,20,21), (T,T,F,T), Seed)

← Representation

(Jane, Smith, (T,T,F,T), Seed)

← Compressed representation

Operations on Tuple Bundles

- **Seed:**

(Jane, Smith, --, --) \Rightarrow
(Jane, Smith, --, --, Seed)

- **Split:**

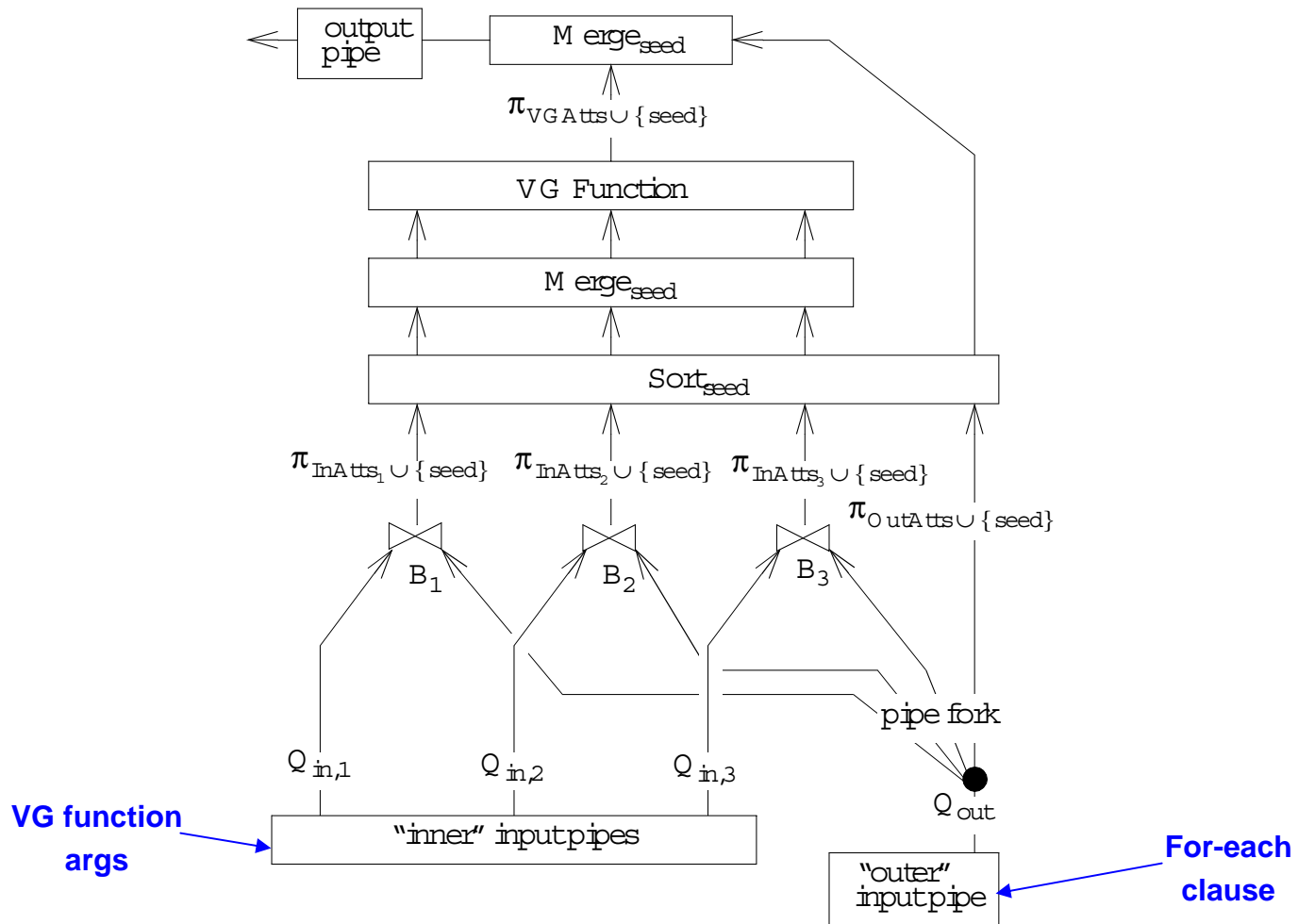
(Jane, Smith, (20,21,20,21), (T,T,T,T), Seed) \Rightarrow
(Jane, Smith, 20, (T,F,T,F), Seed),
(Jane, Smith, 21, (F,T,F,T), Seed)

- **Inference:**

(Jane, Smith, (20,21,20,21), (T,T,T,T), Seed) \Rightarrow
(Jane, Smith, 20, 0.5),
(Jane, Smith, 21, 0.5)

Also: Aggregate

Instantiate Operation



Standard Operations

- **Select** (FNAME = 'Jane' AND AGE = 20)

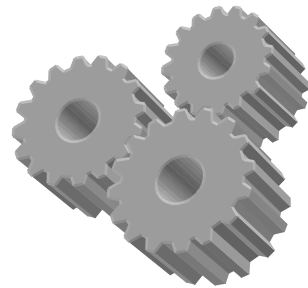
(Jane, Smith, (20,21,20,21), (F,T,T,T), Seed)
(John, Jones, (32,31,20,30), (T,T,F,T), Seed)
(Jane, Jones, (21,23,22,22), (T,T,T,T), Seed) ⇒
(Jane, Smith, (20,21,20,21), (F,F,T,F), Seed)

- **Join** (equijoin on Department #)

(Smith, (D1,D2,D2,D1), (F,T,T,T), Seed1)
(Jones, (D1,D2,D2,D2), (T,T,F,T), Seed2) ⇒
(Smith, D2, Jones, D2, (F,T,F,F), Seed1, Seed2)

Uses SPLIT
+
sort-merge

Estimation and Inference



MCDB inference operator

Distinct tuple value

Val1	Val2	Frac
3	4	0.324
...

Frac. replications where value appears

OutputTable

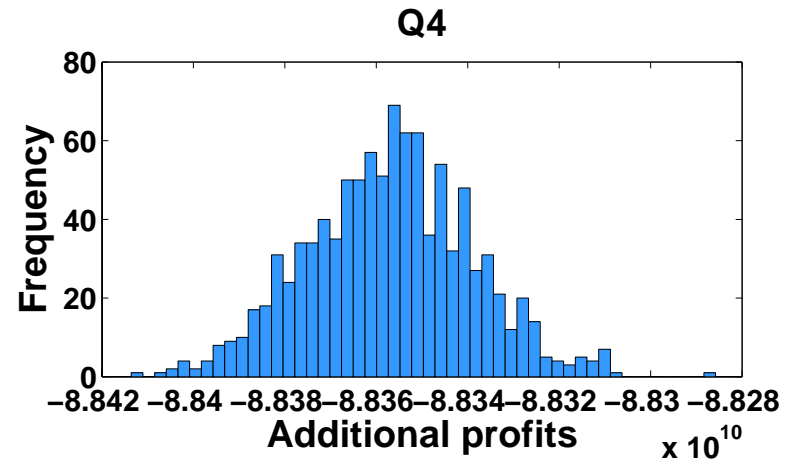
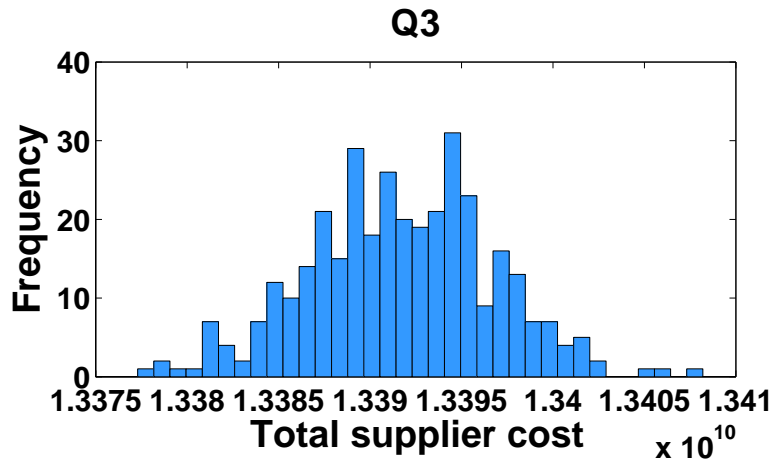
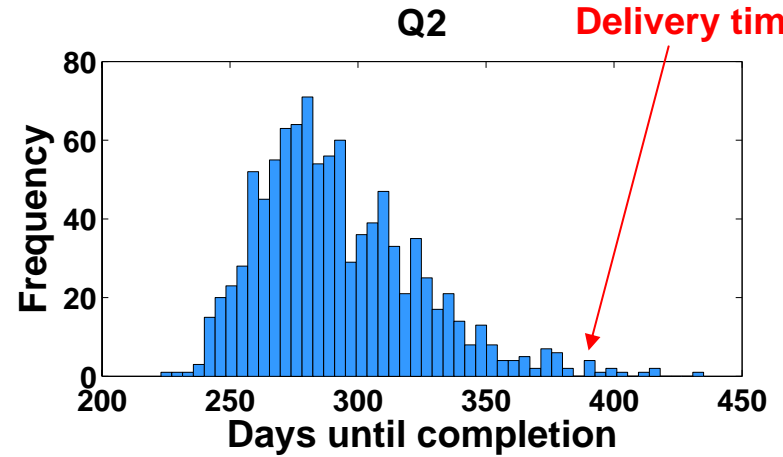
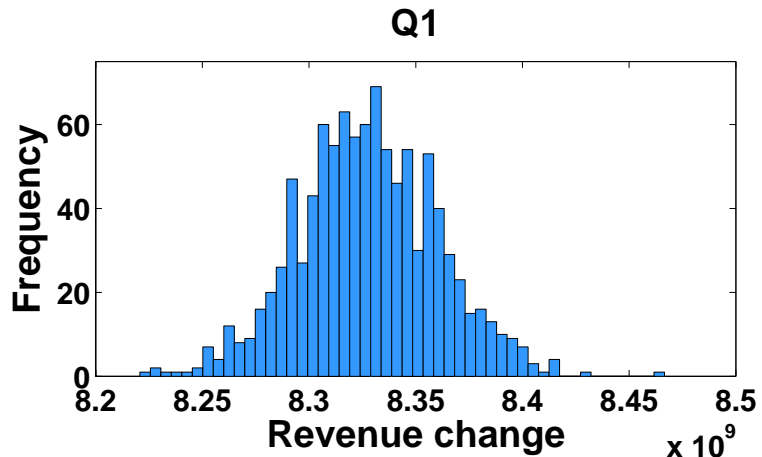
```
WITH Stats(Mu, Var) AS (  
  SELECT SUM(Val1*Frac),  
         SUM(Val*Val1*Frac)  
         - SUM(Val1*Frac)*SUM(Val1*Frac)  
  FROM OutputTable)  
SELECT Mu AS Mean, SQRT(Var) AS Stdev,  
       1.96*SQRT(Var)/SQRT(1000) AS CIHW  
FROM Stats
```

SQL queries

```
WITH CumDistFn(Val1, Cum, PrevCum) AS (  
  SELECT Val1,  
         SUM(Frac) OVER (ORDER BY Val1  
                          ROWS BETWEEN UNBOUNDED PRECEDING  
                          AND CURRENT ROW),  
         SUM(Frac) OVER (ORDER BY Val1  
                          ROWS BETWEEN UNBOUNDED PRECEDING  
                          AND 1 PRECEDING)  
  FROM OutputTable)  
SELECT Val FROM CumDistFn  
WHERE Cum >= 0.5 AND PrevCum < 0.5
```

Results 1 (1000 Reps*)

Long tail in
Delivery times



*Q3 histogram based on 350 reps

Results 2: Execution Times (Min)

Query	1 rep	10 reps	100 reps	1000 reps
Q1	25	25	25	28
Q2	36	35	36	36
Q3	37	42	87	222*
Q4	42	45	60	214

vs 25000,
36000

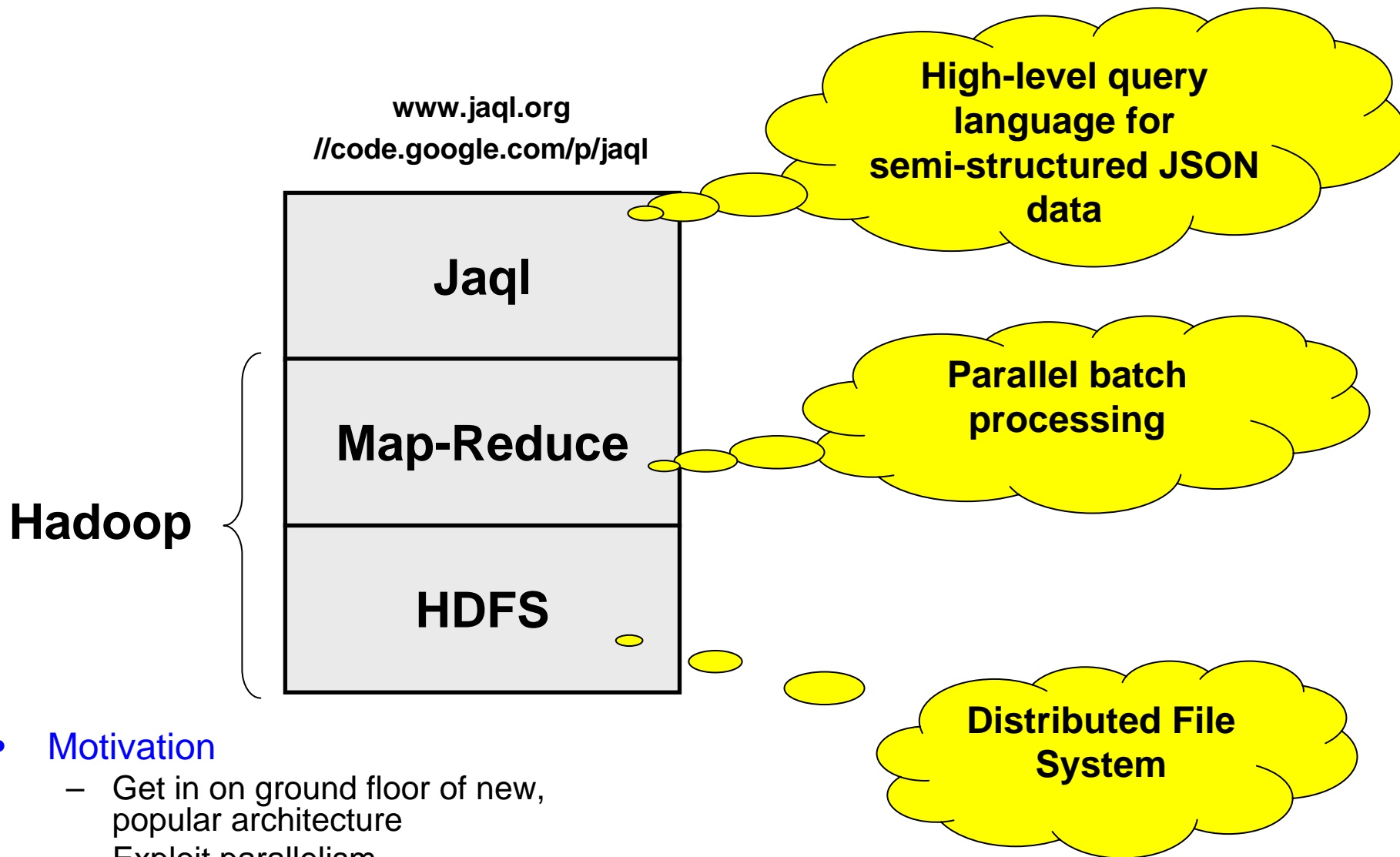
*Based on 350 reps

- Much faster than naïve method in all cases

Outline

- Motivation via examples
- The MCDB system (SIGMOD 2008)
- **Uncertainty in the Cloud**
- An end-to-end scenario
- Future directions

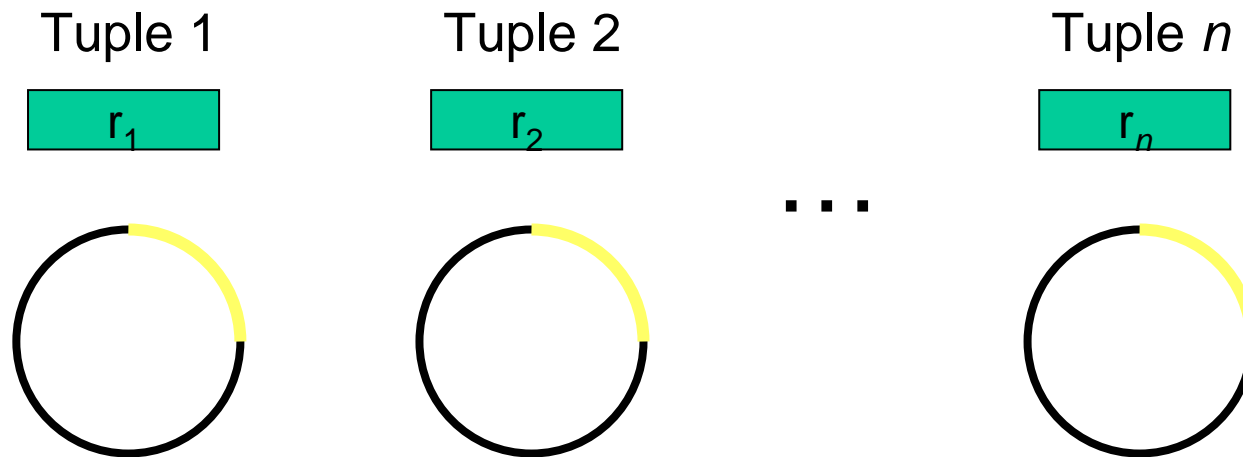
A Cloud-Computing Infrastructure



- **Motivation**

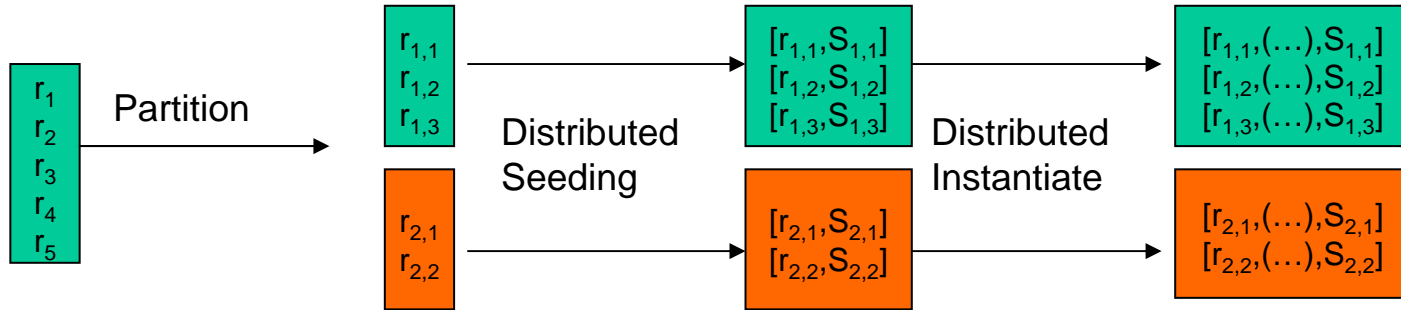
- Get in on ground floor of new, popular architecture
- Exploit parallelism
- Handle nested data

Issue 1: Distributed Seeding

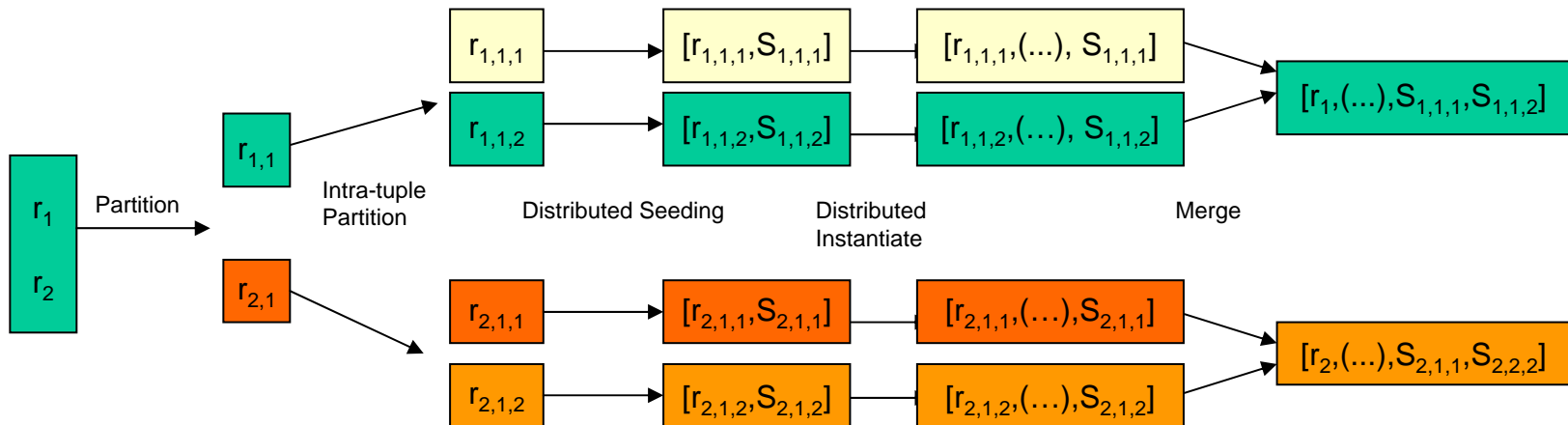


- Maximize parallelization (tuples on different processors)
- Must avoid overlapping seed sequences
- Minimize seed size stored in each tuple

Issue 2: (Inter/Intra)-Tuple Parallelism



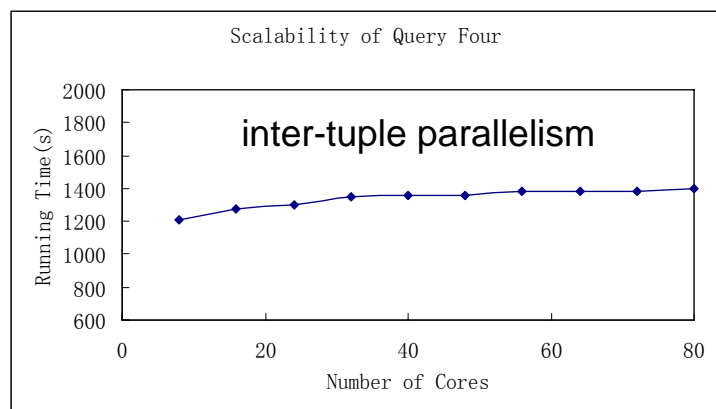
- Inter-tuple parallelism: corresponds directly to Map-Reduce's horizontally partitioned parallelism



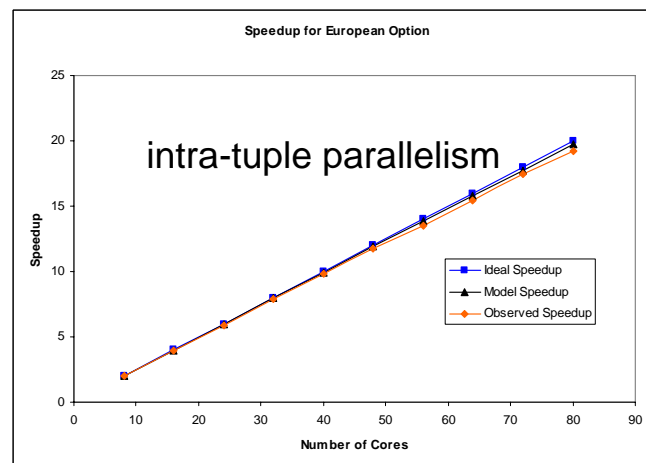
- Intra-tuple parallelism: good for few tuples + (expensive VG functions or many possible worlds)

Preliminary Results

- Implemented two nontrivial queries from MCDB paper
 - Jaql: Map-Reduce plan = original MCDB plan
 - Immediately can handle nested data with uncertain leaf values
 - Scalability with both inter-tuple and intra-tuple parallelism



data size proportional to # cores

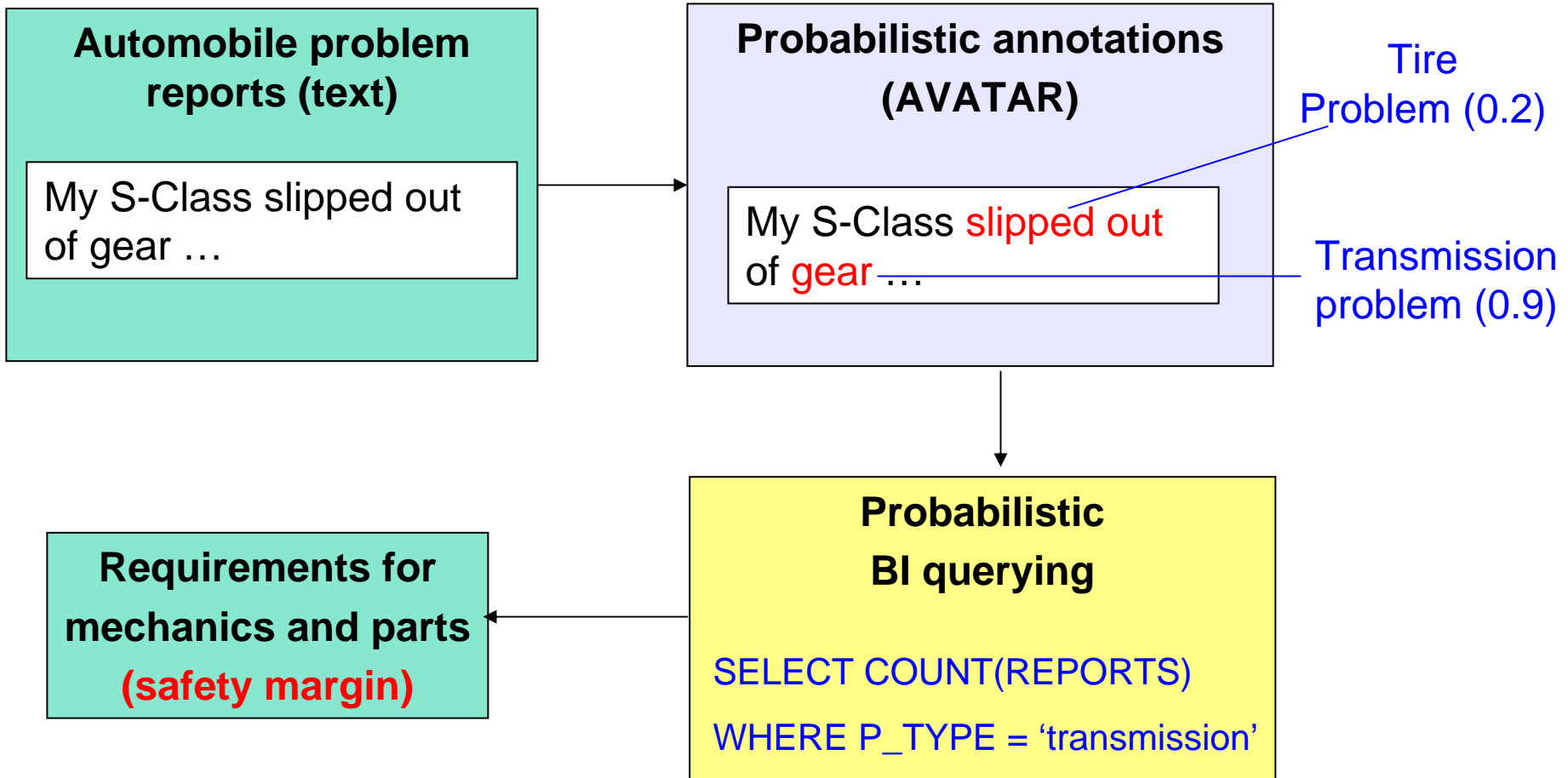


partitions proportional to # cores

Outline

- Motivation via examples
- The MCDB system (SIGMOD 2008)
- Uncertainty in the Cloud
- An end-to-end scenario
- Future directions

An End-to-End ERP Scenario



Future Directions

- **MCDB**
 - Better performance: query optimization, parallelization
 - Specification and provision of desired precision
 - Correlated tables
 - Better inference
- **Cloud Environment**
 - Query optimization
 - Sequential and/or adaptive simulation?
 - More general uncertainty model for unstructured data
- **General**
 - Extreme-quantile estimation (value-at-risk)
 - Other Monte Carlo technology (50 years of research)
 - Combine with exact methods?
 - Lineage management?

Conclusion

- Monte Carlo approach looks promising
 - Complements much of existing probDB research
 - Powerful functionality for real-world problems
 - Plausible performance, can exploit parallelism
- Much work to be done
 - Techniques inherit Monte Carlo weaknesses
 - Many possible improvements and extensions
 - Integrate with other probDB research?
- Best path to market? (How to make it real)
 - Customer needs? Killer apps?
 - Choice of technologies (relational, Cloud, ...)

Questions?

peterh@almaden.ibm.com

www.almaden.ibm.com/cs/people/peterh

Backup Slides

VG Function Implementation

- C++ class with four public methods
 - **Initialize**: set up data structures, seed RNG
 - **TakeParams**: read in “parameter vector”
 - **OutputVals**: return row of output table
 - Return NULL when done
 - **Finalize**: clean up

```
If newRep:  
    newRep = false  
    uniform = myRandGen()  
    probSum = i = 0  
    while (uniform >= probSum)  
        i++  
        probSum += L[i].wt / totWeight  
    return L[i].val  
Else  
    newRep = true  
    return NULL
```

**OutputVals method
For DiscreteChoice()**

Schema Syntax: Example 1

- Goal: generate random customer table
 - MONEY, LIVES_IN are uncertain attributes
 - MONEY has Gamma dist'n
 - shift, shape, scale parameters
 - Use DiscreteChoice for LIVES_IN value
 - Customers are mutually independent, given region
- Parameter table schemas
 - CUST (CID, GENDER, REGION)
 - CITIES (NAME, REGION, PROB)
 - Probabilities sum to 1 in each region
 - MONEY_SHIFT (SHIFT) ← 1 row, 1 column
 - MONEY_SCALE (REGION, SCALE)
 - MONEY_SHAPE (CID, SHAPE)

Normalized
storage

Schema Syntax: Example 2

- Suppose MONEY and LIVES_IN are correlated

```
CREATE TABLE RAND_CUST (CID, GENDER, MONEY, LIVES_IN) AS
FOR EACH d in CUST
WITH MLI AS MyJointDistribution(...)
SELECT d.CID, d.GENDER, MLI.V1, MLI.V2
FROM MLI
```

MLI has 1 row, 2 columns



Schema Syntax: Example 3

- Correlated sensors
 - Sensors in same “sensor group” are correlated (multivariate normal)
- Parameter table schemas
 - S_PARAMS (ID, LAT, LONG, GID)
 - MEANS (ID, MEAN)
 - COVARS (ID1, ID2, COV)

```
CREATE TABLE SENSORS (ID, LAT, LONG, TEMP) AS
FOR EACH g in (SELECT DISTINCT GID FROM S_PARAMS)
WITH TEMP AS MDNormal (
  (SELECT m.ID, m.MEAN FROM MEANS m S_PARAMS ss
   WHERE m.ID = ss.ID AND ss.GID = g.GID),
  (SELECT c.ID1, c.ID2, c.COV FROM COVARS c, S_PARAMS ss
   WHERE c.ID1 = ss.ID AND ss.GID = g.GID)
)
SELECT s.ID, s.LAT, s.LONG, t.VALUE
FROM S_PARAMS s, TEMP t WHERE s.ID = t.ID
```

Query Results

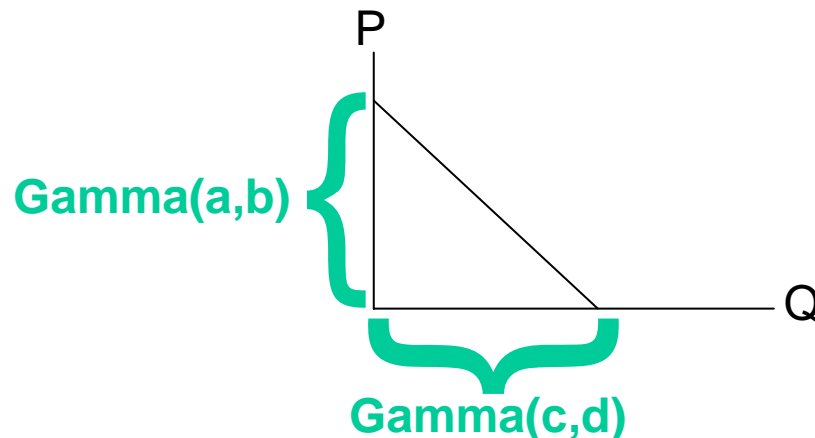
- User specifies number of replications
- Currently MCDB returns
 - List of all distinct tuples produced
 - For each distinct tuple, # of possible worlds where it appears
- Can compute
 - Appearance probabilities
 - Means, variances, quantiles
 - Histograms
- More work needed
 - Specifying uncertainty metrics + precision
 - Exploiting this information in query processing
 - Joint inclusion probabilities

Experimental Queries

- Q1: Next year's revenue gain from Japanese products
 - Assuming current trends hold
 - Each order duplicated Poisson # of times
 - Poisson mean = (this year)/(last year) for customer
- Q2: Order Delays
 - From placement to delivery
 - Fitted Gamma distribution for each delay type (for each part)
- Q3: What if we had used cheapest supplier?
 - TPC-H only has **current** prices
 - Prior prices generated by backwards random walk with drift
- Q4: Change in profits with 5% price increase
 - Bayesian model of customer demand
 - Based on **all** customers orders at current price

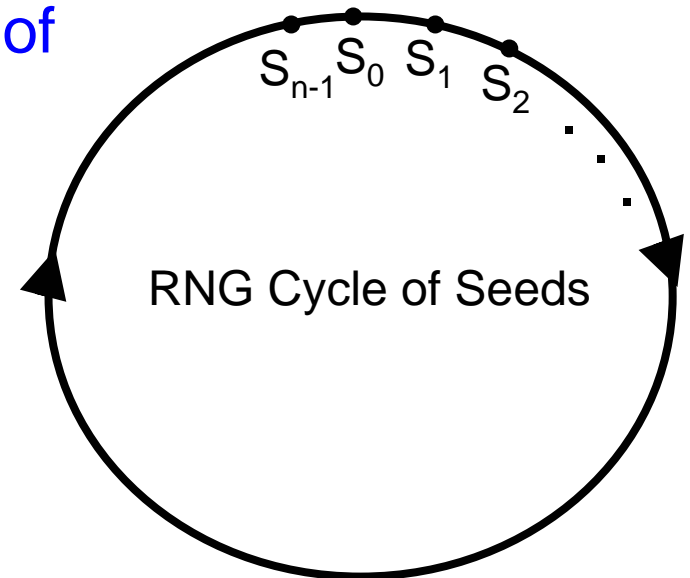
An Experimental Query

- Effect on profits of 5% price increase
 - Want more accuracy than usual aggregated demand functions
 - E.g, exploit detailed point-of-sale data
 - For each part
 - Fit “prior” demand-function distribution to all customers (MLE)
 - Determine “posterior” distribution for each cust. (Bayes Thm)
 - Generate random demand for each customer at new price
 - Use rejection algorithm to sample from posterior



Pseudorandom Number Generators (RNG)

- Needed by VG function
 - E.g., to generate random ages
- Produces a deterministic sequence of seeds that “look” random
- Typical RNG recurrence:
 - $S_{i+1} = M * S_i$
 - Seed S = vector of k unsigned integers
 - M is a matrix
- Keeping only initial seed, S_0 , is sufficient to regenerate sequence



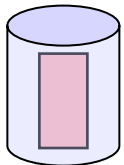
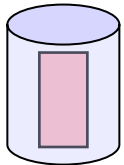
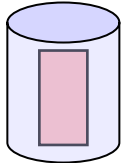
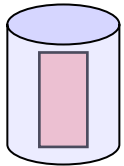
Jaql Overview

- What is Jaql?
 - Jaql: a JSON query language
 - JSON==Java Script Object Notation
 - Nested, self-describing data
 - Widely used in web applications
 - Easily integrated into most programming languages
- Why did we choose Jaql?
 - Designed for massive-scale cloud computing
 - Rewrites queries to use Map-Reduce
 - Partial aggregation is supported
 - Designed for extensibility
 - Read / write data from any source into JSON view of data
 - Easy to add new user-defined functions
 - VG function can easily be plugged in.
 - Exploit nested data model (JSON)

Map-Reduce Overview

Partitioned
Input File:

$[(K, V)]$



(K, V)



$[(K_m, V_m)]$

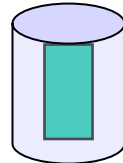
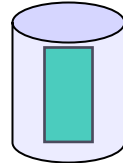


$(K_m, [V_m])$ $[V_r]$





Partitioned
Output File:

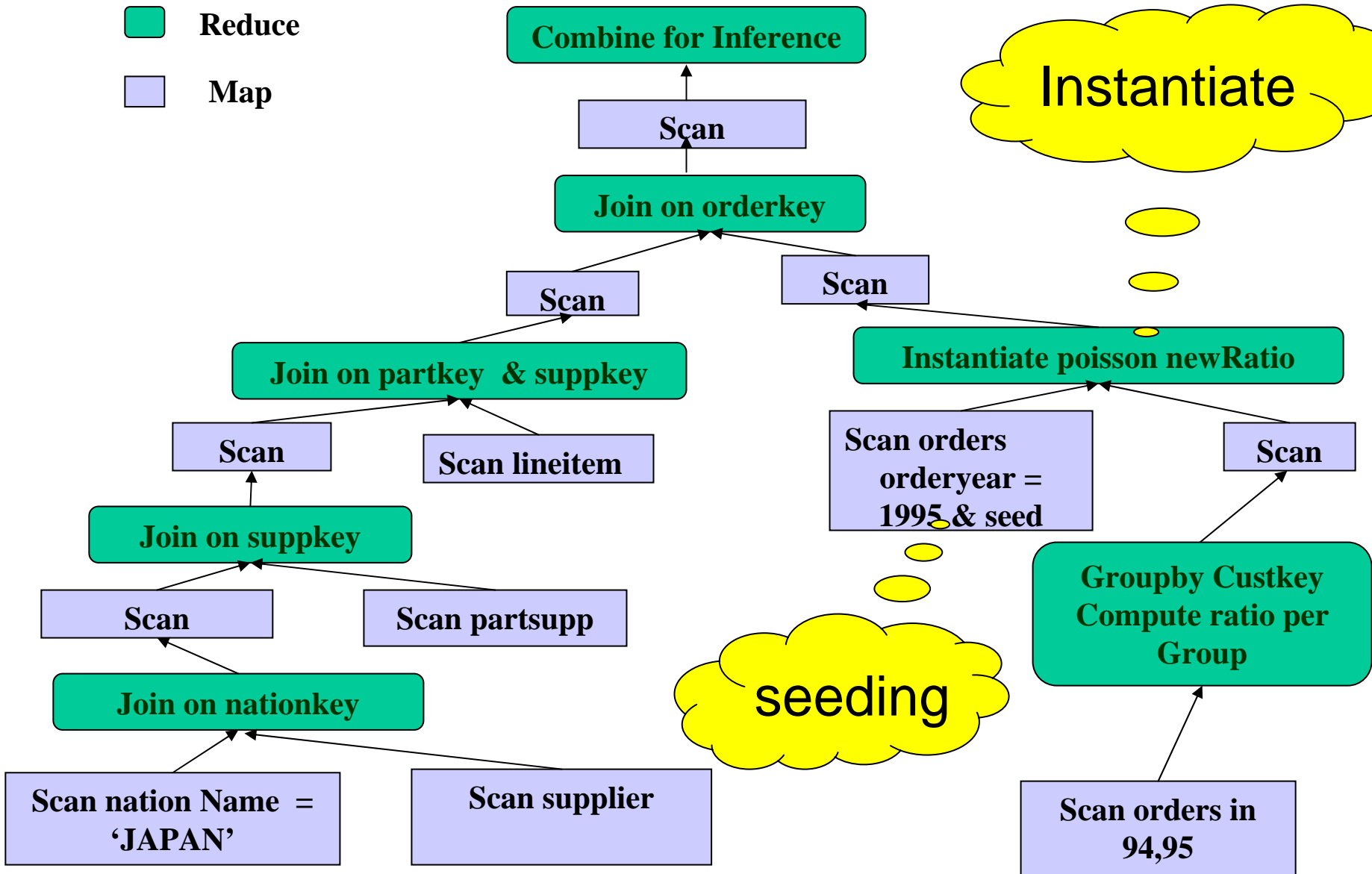
$[V_r]$



- Programmer focus:
 - Map: $(K, V) \rightarrow [(K_m, V_m)]$
 - Reduce: $(K_m, [V_m]) \rightarrow [V_r]$
- System provides:
 - Parallelism
 - Sorting
 - Synchronization
 - Fault tolerance

Example of a Query Plan

-  Reduce
-  Map



Non-relational Data

- TPC-H schema is used.
- Two different ways to nest data
 - Nest *lineitem* table under *orders* table
 - Nest *lineitem* table under *partsupp* table
- Modified version of Q4 from original MCDB paper
 - Jaql successfully runs through the query
 - In the first nested case, running time is slower
 - In the second nested case, running time is faster
- Only uncertain attributes in leaf node is supported