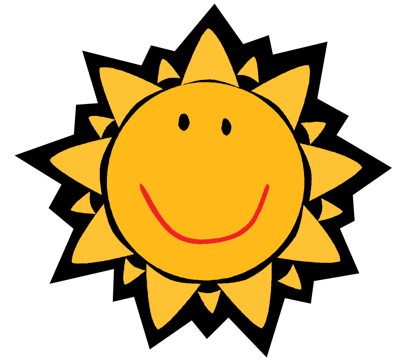
Query Optimization: Are We <u>There</u> Yet?

Guy Lohman
IBM Almaden Research Center (retired)

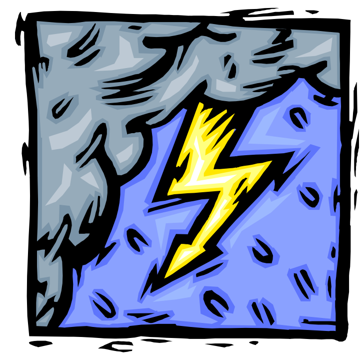
Guy_Lohman@alumni.pomona.edu

Outline for a Keynote Speech

1. Congratulations on Past Successes!



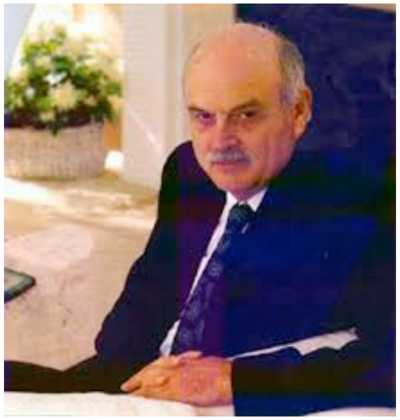
2. Storm on the Horizon!



3. Roadmap for Challenges Before Us

Query Optimization 101

- **What IS Query Optimization?**
  - A model of query performance for different **execution plans**
  - **Goal:** Pick the best-performing plan (more on this later!)



- **Why is it needed?**
  - SQL is **non-procedural** (specify what, not how)
  - **Execution Plan** is (roughly) implemented by **relational operators**:
    - ***SELECT*** — apply a single-table predicate
    - ***PROJECT*** — access columns
    - ***JOIN*** — apply a multi-table join predicate
  - **Relational operators** form an algebra (Thanks, Ted Codd!) that is
    - Commutative
    - Associative

➡ Operators may be re-ordered!

➡ **LOTS of possible plans** (roughly exponential in # of tables)!

  - No one access method (index vs. scan) or algorithm (join method) dominates
  - **Heuristics** reduce the space somewhat:
    - **Predicate push-down**: apply ***SELECT*** as soon as possible to eliminate unqualifying rows
    - **Cartesian product avoidance:** defer Cartesian products, i.e., joining tables with no join predicate

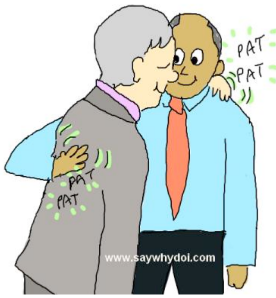
➡ Depends upon the shape of the query graph

- Refn: Ono & Lohman: Measuring the Complexity of Join Enumeration in Query Optimization. **VLDB 1990**: 314-325

Is Query Optimization Successful?

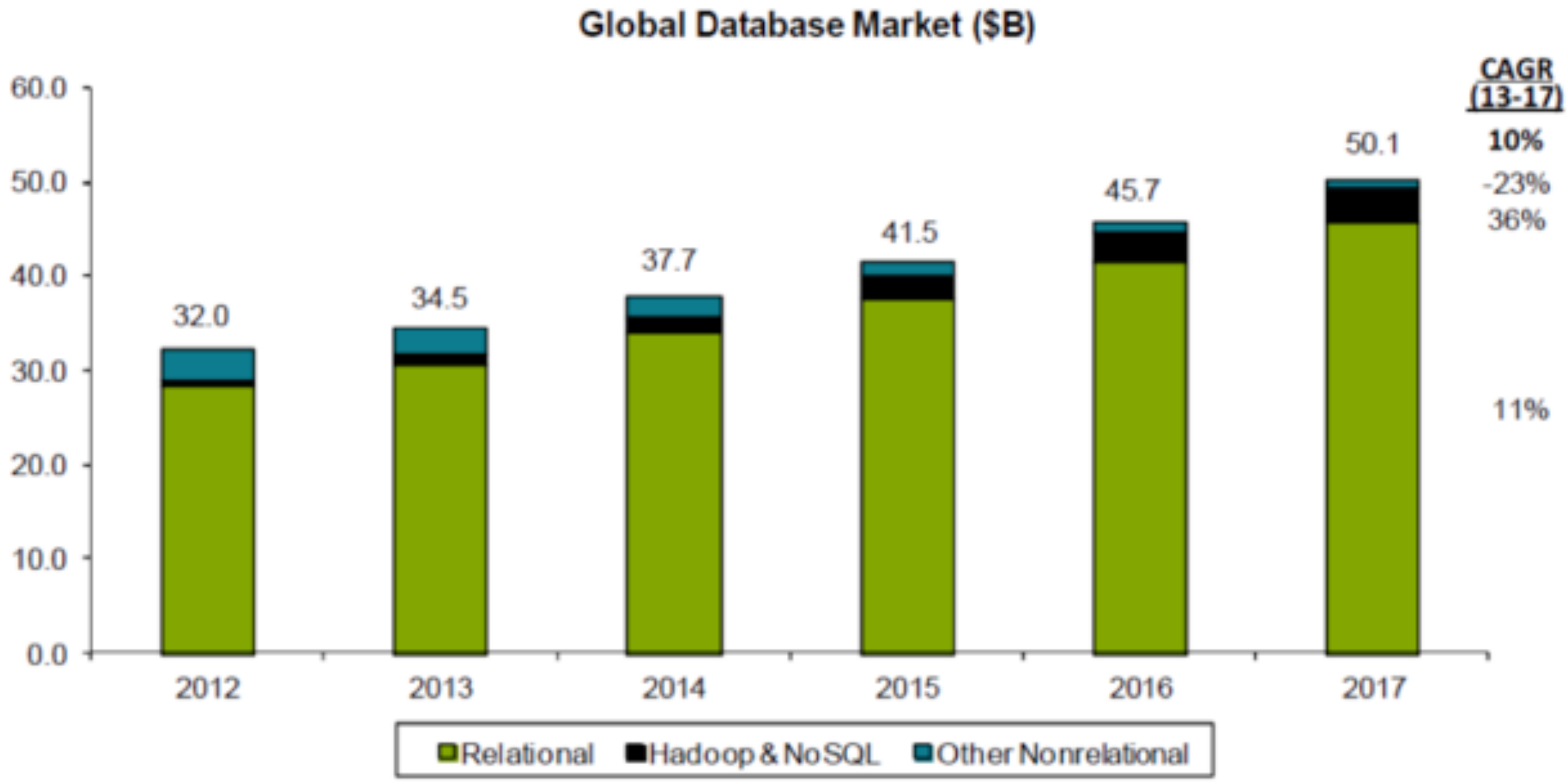
- **Relational Databases are a WHOPPING SUCCESS:**
  - **Relational database industry > $40B business in 2016!**
    - **Wouldn't have happened if optimization failed often enough**
  - **Optimizers get the "right plan" most of the time**
  - **SQL is (still)…**
    - **widely accepted for writing database apps**
    - **recognized as most successful declarative language**
    - **used by 95% of Spark users, too! (Refn.: Spark Summit 2017)**

- **Many of today's products derive from academia:**
  - **<u>Huge</u> literature: 858 hits for "query.optimiz" on DBLP**
  - **Endured & evolved as a "hot topic" for decades**
  - **Many of today's products derive from research prototypes**
    - **original, extensible, object-relational, distributed, parallel, …**



- **So give yourselves a big pat on the back!**

RDBMS is a YUGE Market!



Source: IDC, Bernstein analysis

But…LOTS of Problems Remain

- **Even a small percent of "bad plans"…**
  - **Contributes most to bad performance**
  - **Breeds mistrust of the optimizer**
  - **Fosters demands for "easy fixes" that hurt, don't help**
    - **"Hints" are an admission of failure!**
      - **Hint:** user specifies (portion of) a plan for a query
      - Tell optimizer how to do its job!
      - Harder for DBA to do right for more complex queries
      - Aren't robust to changes in design, statistics, or environment
    - **"Fudge factors" in optimizer are even more undermining**
      - **"Fudge factor":** Bias a cost in favor of (or against) a type of plan
      - Unknowingly affect other queries (that might have been fine before!)
      - Fudge factors beget more fudge factors, and yet more…

- **Bad plans are the "tip of the iceberg"**
  - **Indicate unseen problems lurking**
  - **Will bite you at the least opportune time (Murphy's Law)**

Scope

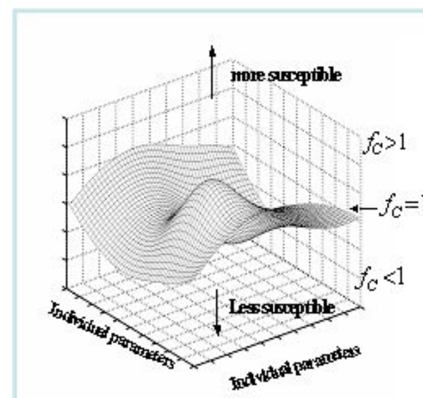
- **This talk covers…**
  - **Cardinality and Cost Models**
  - **Assumptions and their impact**
  - **Some ways to avoid the impact of failed assumptions**
  - **Research areas**
- **This talk does NOT cover…**
  - **Rule-based Query Rewrite transformations**
  - **Plan Enumeration strategies**
  - **Yet another histogramming technique**
- **Focus on things having the most impact!**
  - **Avoid "polishing the round ball"**
  - **Look for order-of-magnitude errors**

Thesis of This Talk:
Query Optimizers are Mathematical Models

- Optimizers model <u>performance</u> of each query plan that it considers
- <u>Assumptions</u> that underly the model must be
  - Carefully identified and understood, especially their potential impact
  - Minimized for robustness
  - Verified against application

- Strict comparison to <u>reality</u> is the <u>only</u> robust metric
  - Relativists and theoreticians need not apply!
- To be trustworthy, the model itself <u>must be validated against reality</u> for <u>all</u>…
  - Possible permutations of values in its parametric space, i.e., its inputs:
    - <u>All</u> database designs (normalization, indexes, partitioning,…), even bad designs!
    - <u>All</u> table characteristics (statistics, even values!)
    - <u>All</u> SQL queries, of arbitrary complexity (negation, disjuncts, 10-sigma constructs, subqueries, …)
  - Environments
    - Hardware
      - Not just the latest & greatest!
      - Any degree of parallelism, on any number of nodes
    - Competing workloads on same system

→ Requires an <u>INCREDIBLE amount</u> of testing!



- This Mission is <u>Impossible</u>, and <u>no one</u> has even <u>attempted</u> it

Our Most Egregious Assumptions

Simplifying assumptions that too often aren't true:

1. **Workload:** We can optimize each query independently
   i.e., only one query is running at a time
2. **Predicate values fixed:** Values of predicates are known & fixed
   i.e., no parameter markers or host variables
3. **Independence:** Selectivities of predicates are mutually independent
   i.e., univariate distributions of columns suffice
4. **Subsumption:** Joins are on domains, one of which subsumes the other
   i.e., Primary Key and Foreign Key
5. **Weighting:** Certain types of cost (I/Os, memory, CPU) dominate others, which can be ignored
6. **Additivity:** Estimated costs can be simply added together
   i.e., nothing ever happens in parallel, and even if they do…
   …they all start at the same time
7. **Non-relational data:** Relational data dominates
   i.e., we don't need no stinkin' hierarchies / repeating groups!
8. **Detail:** More detailed models are more accurate

Assumption #1: Workload Independence

- **Assumes: Each query can be optimized <u>once, by itself</u>**
- **Reality: Queries affected by concurrent workload**
- **Other queries and applications compete for resources, e.g.,**
  - **Memory & caches used for buffer pool, hash tables, …**
  - **CPU**
  - **Bandwidth**
- **These ~~may~~ will vary from run to run of a given query!**
- **<u>Cannot know</u> *a priori* what will be running concurrently!**
  - **"Your mileage may vary."**
- **Example: Buffer pool available to query determines disk I/Os in very complicated, non-linear way (e.g., table all fits vs. 1 page over)**

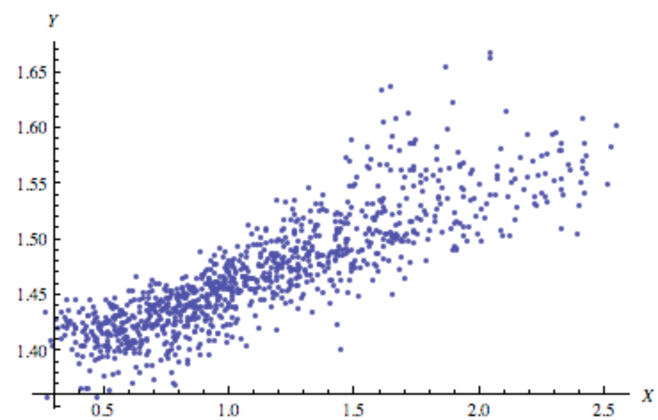
Assumption #2: Predicate Values Fixed

- **Assumes: Predicate constants are fixed & known at optimization time**
- **Reality: Applications love parameter markers & host variables, whose distribution of occurrence is unknown to the optimizer**
- **Related to Assumption #1: each execution occurrence may differ!**
- **Customer "war story":**
  - ***SubsystemID* added retroactively to all tables & predicate on it to all queries**
    - 6 possible values
    - But one value occurs 99% of the time in app (the original subsystem)
  - **What should the selectivity of "*SubsystemID* = ?" be?**
    - DB2 calculates selectivity as **1/distinct values, so 1/6 = .167**
    - **Reality: 0.99 if ? = 1; else < 0.01**

- **What's the best strategy for re-compiling?**
  - Every query execution? Expensive!
  - Just the first execution, assuming a "typical" value? Inaccurate!
  - Some compromise? What?

Assumption #3: Predicate Independence

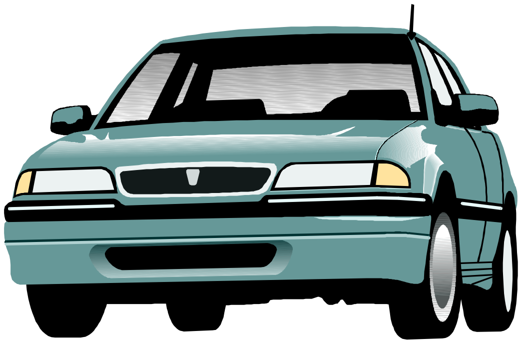
- **Assumes: Predicates are mutually independent**
- **Reality: Attributes can be correlated, even across tables!**
- **Originates from System R and Ingres Optimizers**
- **Assumes:** $f(c_1, c_2, c_3, \ldots, c_N) = f(c_1) * f(c_2) * f(c_3) * \ldots * f(c_N)$
- **Significantly simplifies:**
  - **Statistics collection —** do for each column independently (N distributions vs. 2^N)
  - **Selectivity estimation —** just multiply selectivities of conjuncts!
- **Problems:**
  - **Customers are unaware of**
    - existence of correlation among attributes
    - its impact on optimization, think "more is better"
  - **Result:**
    - Can significantly under-estimate cardinalities!
    - Incorrectly favors nested-loop joins — **disaster!**



- **Examples: "war stories" from real customer databases (next 3 slides):**
  1. **Honda Accords**
  2. **"More is better" predicates**
  3. **Cross-table predicates and intersections in star schemas**

Correlation Example #1: Honda Accord

- **Problem:**
  - Database for governmental car registration agency
  - **WHERE *Make* = *'Honda'* AND *Model* = *'Accord'***
  - Suppose, for ease of exposition, …
    - 10 *Makes* ==> selectivity(*Make*) = 1/10
    - 100 *Models* ==> selectivity(*Model*) = 1/100
  - So selectivity of both = 1/10 * 1/100 = 1/1000
  - **But only Honda makes an Accord model, by trademark law!**
  - We assumed the predicates were independent by multiplying their selectivities!
  - In fact, *Model* functionally determines *Make*
    (predicate on *Make* really adds no information)!
  - **Effect:** We under-estimated cardinality by an **order of magnitude**!



- **In general,**
  - Can be among any subset of (perhaps dozens of) predicates in the WHERE clause
  - How do we know which subset of predicates caused the error?
  - How do we generalize to all instances of *Make* and *Model*?
  - What happens if we repeat the same predicates, and optimizer doesn't remove them?

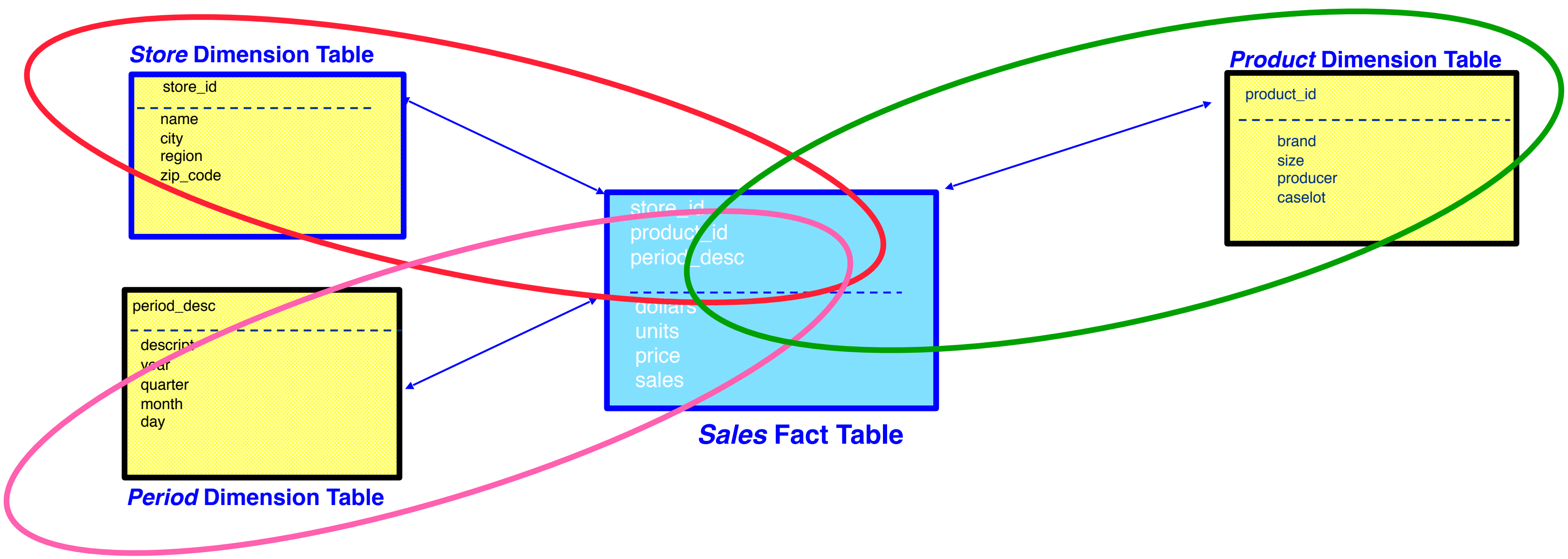
Correlation Example #2: “More is Better”

- **Context:**
  - Major U.S. Insurance Company
  - Complex query joining 10s of tables (EXPLAIN print-out was > 2 cm. thick)
  - 10M-row table ***AccountHolders*** had these predicates (using me as example):
    - ***NameLast = ‘Lohman’***
    - ***NameFirst = ‘Guy’***
    - ***NameMiddleInitial = ‘M’***
    - ***AddressStreet = ‘1114 Virgil Place’***
    - ***AddressCity = ‘San Jose’***
    - ***AddressState = ‘CA’***
    - ***AddressZip = ‘95120’***
    - ***SocSecNum = ‘123-45-6789’***

- **Adding <u>one predicate</u>** to query degraded performance from a **few seconds** to **> 1 hour**!!
- **Problem:** Can you figure out WHY?
- **Hints:**
  - Cardinality estimate for ***AccountHolders*** **decreased by 10^{-7}** in modified query
  - Added predicate: ***PersonID = ‘LOHMGM951206789’*** (concatenation of name, zip, & SSN)
- **Solution:**
  - **Predicate is <u>completely redundant</u> (correlated to others)!!**
  - Developer **thought it would help** this query, because there was an index on it
  - It <u>might</u> help the execution, if that index was picked, …
  - BUT it caused under-estimation of cardinality, changing join type from Hash to Nested-Loop

Correlation Example #3: Cross-Table Correlation

- **EXAMPLE Query to Star Schema:**
  - ***City = 'San Jose'***: 10s of millions of sales in all San Jose stores!
  - ***Month = 'December'***: 100s of millions of sales in December!
  - ***Brand = 'Levi Dockers'***: millions of Levi's Dockers!
- **TOGETHER:**
  - Probably only thousands of Levi Dockers sold in San Jose stores in December!!
  - But might be much higher if there was a promotion, or lower if competitor did
  - **"It depends!"**



-

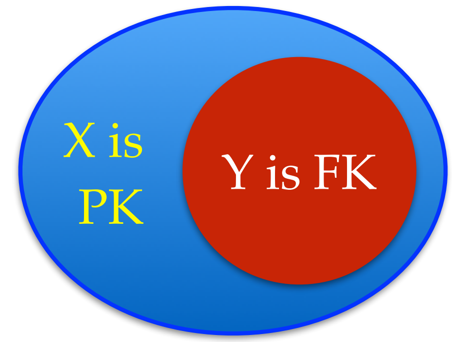
Assumption #4: Subsumption in Joins

- **Assumes: One domain in a join subsumes the other, i.e., PrimaryKey joined with ForeignKey**
- **Reality: It Depends!**
- **System R assumed this with following formula for join cardinality:**

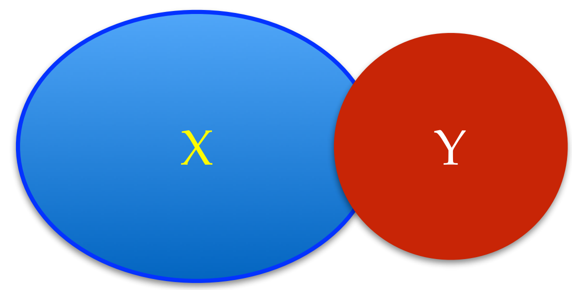
$$|T_1 \; join_{X=Y} \; T_2| = |T_1| * |T_2| \,/\, Max\{\, |X|, |Y| \,\}$$

where X is a column in T_1 and Y is a column in T_2

- **When X is a PrimaryKey and Y is a ForeignKey,**
  - *Domain (PK)* **<u>subsumes</u>** *Domain (FK)*, so …
  - $|X| > |Y|$ and $|X| = |T_1|$, so …
  - *JoinCard* $= |T_2|$ (the FK table, usually the bigger one)



- **Fortunately, most joins <u>are</u> on PK - FK!**
- **BUT … Not necessarily! Could be….**



- **Example:** Join **online logs** to **transactions table** on *Date* and *Timestamp* columns

Assumption #5: Weighting in Costs

- **Assumes: Certain resources dominate others**
- **Reality: It Depends!**
- **Early Optimizers (e.g., System R) assumed disk I/O dominated CPU**
  - The mysterious factor H = 1/3 (for CPU)
  - But some operations (e.g., SORT) use much more CPU than others (e.g., SCAN)
- **Better (e.g., in Starburst, DB2 LUW, and many others):**
  - Linear combination: **Cost** = w_1 * **I/O** + w_2 * **CPU** + w_3 * **Comm**
  - Weights $\mathbf{w_i}$
    - Convert unit-less counts (e.g., I/Os, instructions, message blocks) to time (msec.)
    - Must be determined by system automatically and empirically (measured mile)
  - I/Os further broken down into Random and Sequential I/Os
  - **BUT** still assumes these cost components are additive (no parallelism) …see next slide!
- **Need to modernize by adding:**
  - Cost of lock granularity (row vs. table) and duration
    - How weight this?
  - Multi-core parallelism
  - Cache awareness
  - Compression and de-compression costs
  - **Cloud metrics** (total resources, SLA penalties,…)
  - So much more…

Assumption #6: Additivity in Costs

- **Assumes: Costs can simply be added together, i.e.,**
  <u>Nothing</u> happens in parallel, and even if they did,
  They start at the <u>same time</u>
- **Reality: Lots of parallelism among I/O, CPU, & Network!**

- **Additivity benefits:**
  - Simplifies cost calculations
  - Avoids cost functions like ***Max { time$_1$, time$_2$, time$_3$ }***
  - <u>Required</u> by **Principle of Optimality** for Dynamic Programming
- **But …**
  - **Is it realistic? Maybe for queries run on AWS**
  - **What if overlap is partial?**
  - **Don't forget: other, unknown queries & apps run concurrently (violating Assumption #1)!**

Assumption #7: Relational Data Dominates

- **Assumes: Most data is simple, relational tables, i.e.,**

  <u>No</u> JSON, XML, etc., i.e.,

  <u>No:</u>
  - **objects**
  - **structures of hierarchies**
  - **arrays or repeating groups**
  - **navigational query constructs**


```
"firstName": "John",
"lastName": "Smith",
"age": 25,
"address": {
    "streetAddress": "21 2nd S
    "city": "New York",
    "state": "NY",
    "postalCode": 10021
},
"phoneNumbers": [
    {
        "type": "home",
```


- **Reality: Brave new world of non-tabular data (JSON, XML,…)!**

- **Benefits of assuming relational:**
  - Simplifies cardinality and cost calculations, run-time, statistics, etc.
  - Preserves commutativity and associativity of operations
  - Avoids adding non-relational operations that might interfere with reordering opns.

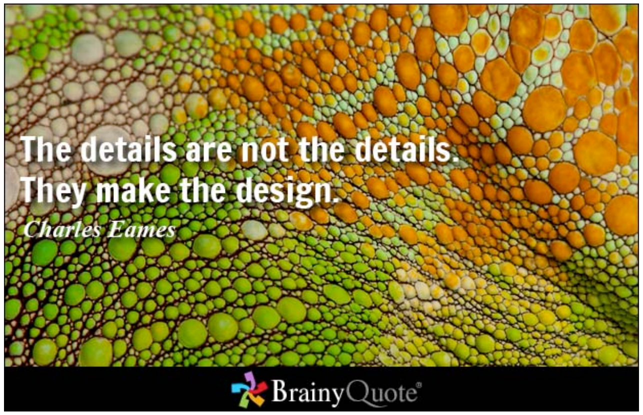
- **But …**
  - **Prohibits user-defined operations that might not be commutative or associative**
  - **Limits supported apps and platforms (Spark, Hadoop)**

Assumption #8: Detail Improves Model

- **Assumes: Increasing model detail improves accuracy**
- **Reality: More detail ⇒ more assumptions**
  ⇒ model more brittle!!!

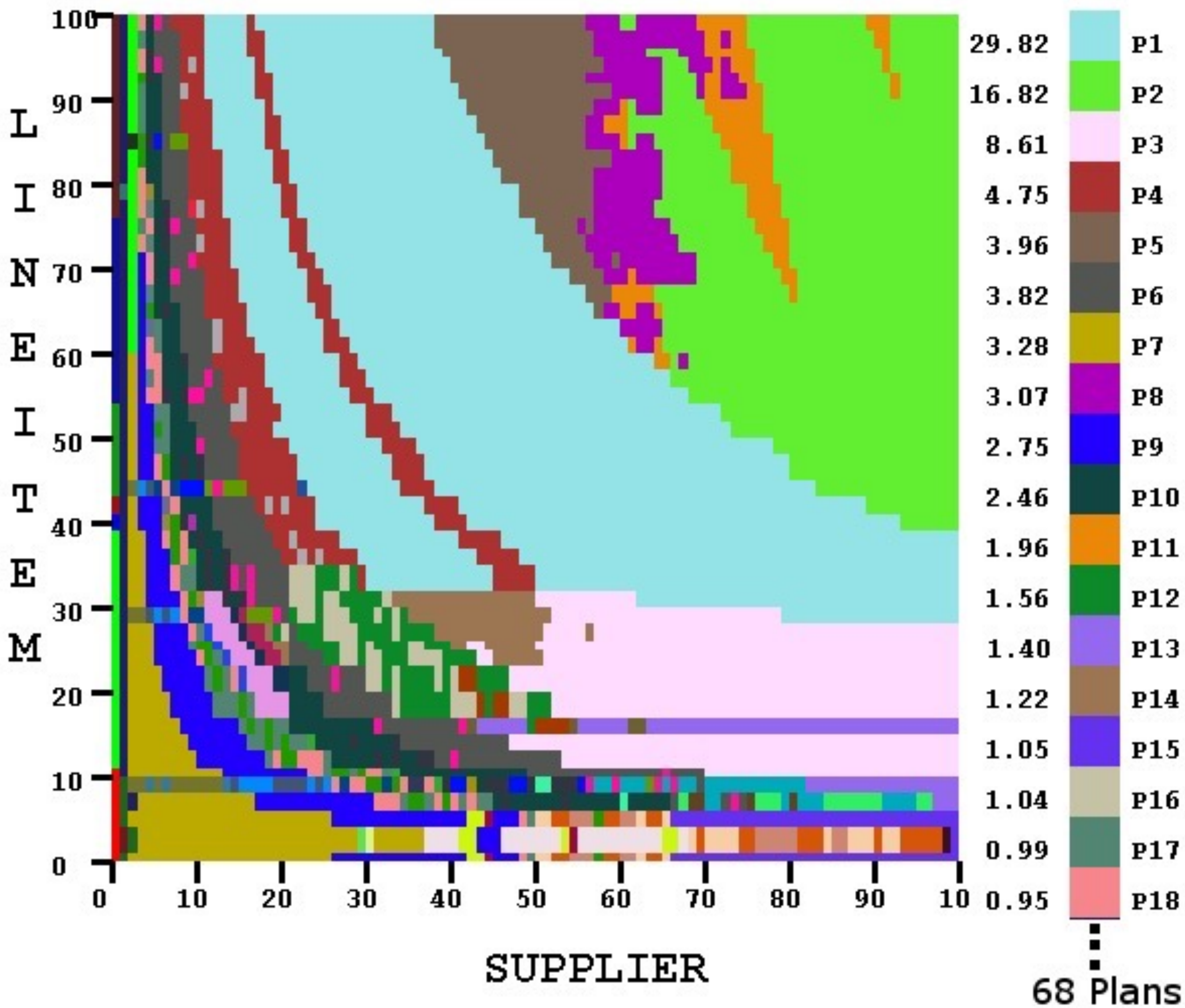
- **Our natural reaction to wrong plans is to embellish the model**
  - Specifically in the area where we went wrong
  - A more detailed model <u>must</u> be more accurate, right?
- **But, but, but … Additional details inevitably**
  - Contain additional assumptions
  - Require additional statistics
- **<u>Query Optimization Conundrum:</u>**

  More detailed optimizer models risk increased brittleness, because there are more places to go wrong.



- **Example: Adding a detailed model of multi-core parallelism adds assumptions about:**
  - Relative start times of parallel threads
  - Cache utilization
  - % of stalls
  - etc.

In Fact,
Richer Plan Repertoire can be Counter-Productive!



- Reference: N. Reddy and J. R. Haritsa. *Analyzing plan diagrams of database query optimizers.* **VLDB 2005**.

What's a Conscientious Optimizer Guru to Do?

- **Avoid Unvalidated Assumptions:**
  - Minimize the number (KISS!)
  - Explicitly validate that they hold — or don't! — for the application
  - Or at least be on the alert for their impact!
- **Exploit:**
  1. Improved statistics about correlations
  2. **Actuals** (e.g., observed cardinalities) or samples whenever possible
  3. Plans that **adapt** to learned information
  4. **Robust execution strategies**

- **Use <u>measurable reality</u> as metric (e.g., execution time), <u>not</u> relativism (fudge factors beget more fudge factors)**

- **<u>Thoroughly validate</u> our cardinality and cost models!**

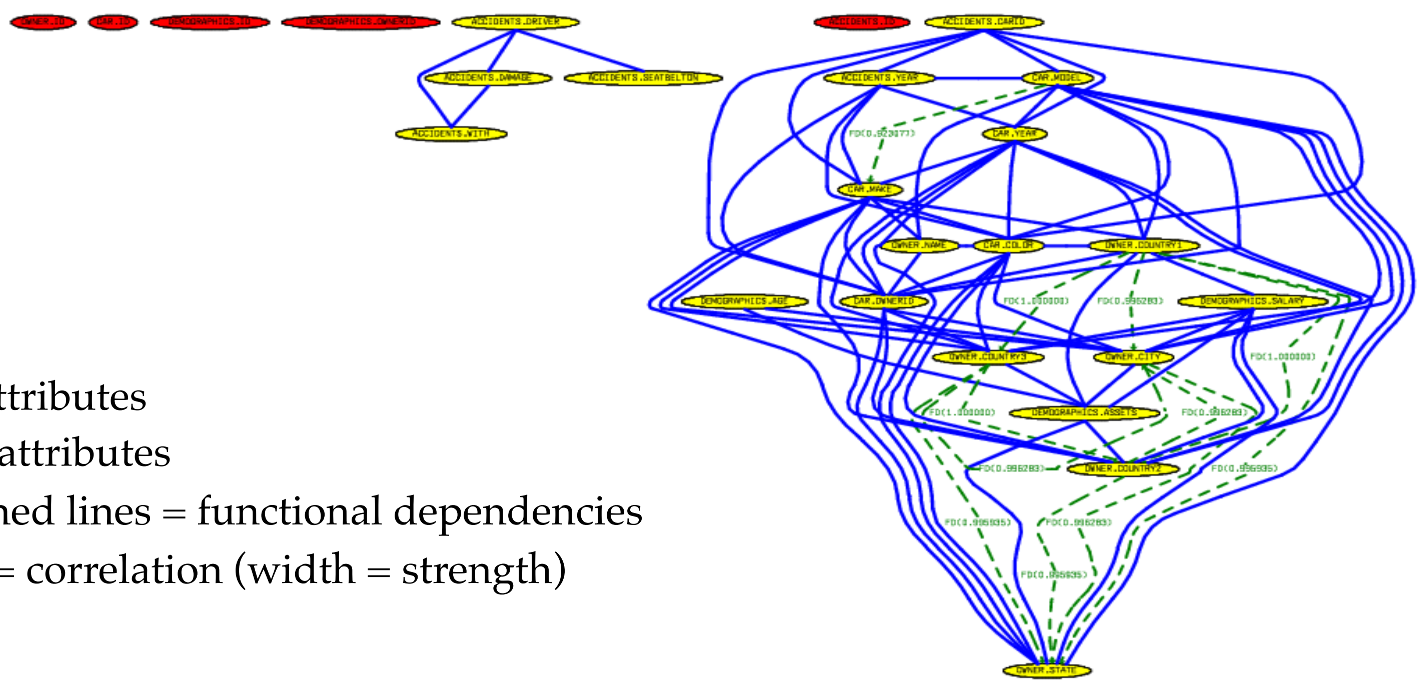
Solution #1: Proactively Finding Correlations using CORDS

(**COR**relation **D**etection by **S**ampling)

1. **Sample each column to determine**
   - Keys
   - Possible joins
2. **Sample pairs of columns to determine correlations**
   - Within a table
   - Across joinable tables
3. **Determine correlation of each pair**

- **Reference:**

  Ilyas, Markl, Haas, Brown, & Aboulnaga: *CORDS: Automatic Discovery of Correlations and Soft Functional Dependencies.* **SIGMOD 2004**, 647-658



Key:

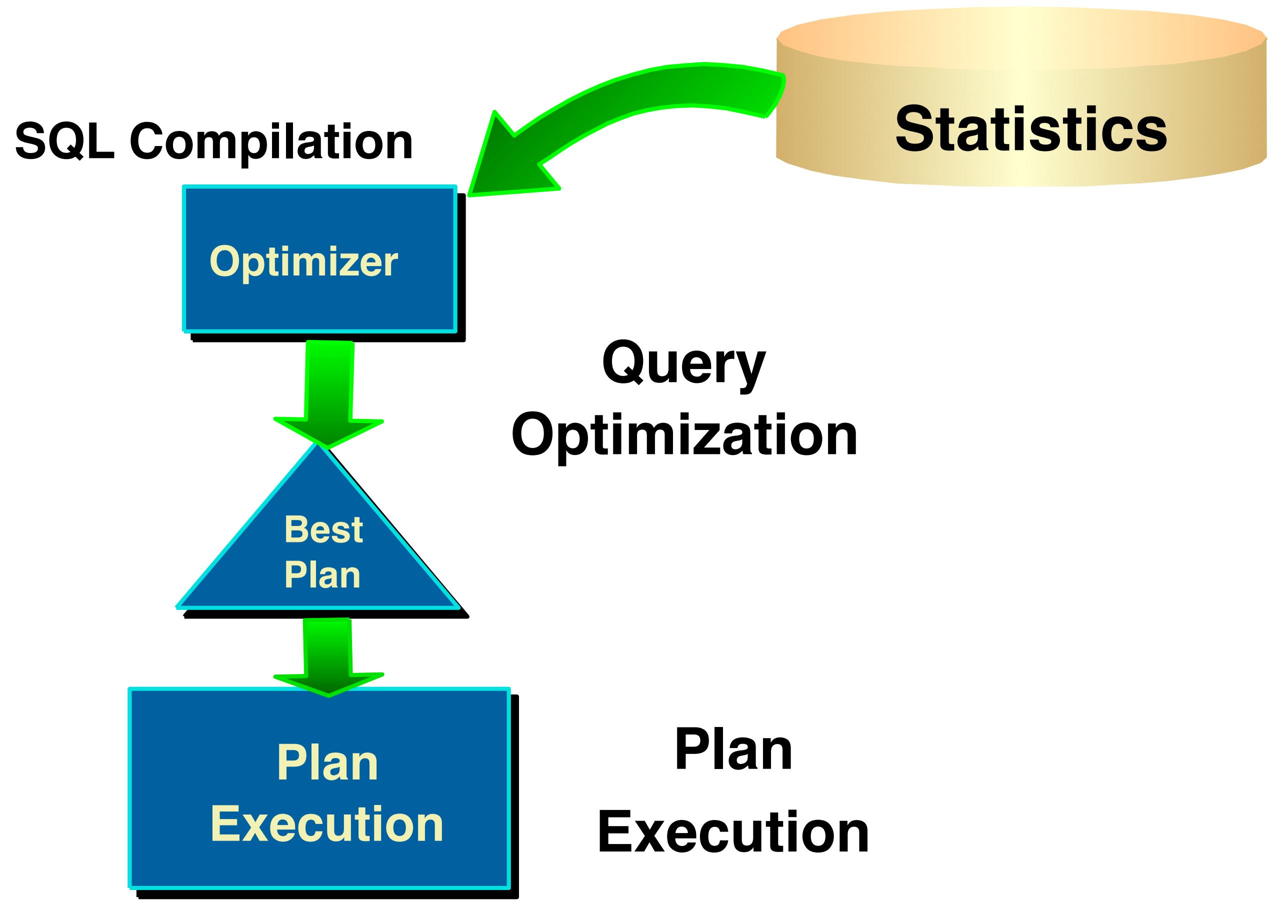
- Yellow = attributes
- Red = key attributes
- Green dashed lines = functional dependencies
- Blue lines = correlation (width = strength)

Solution #2: Learn from past mistakes!
The LEarning Optimizer(*LEO*)

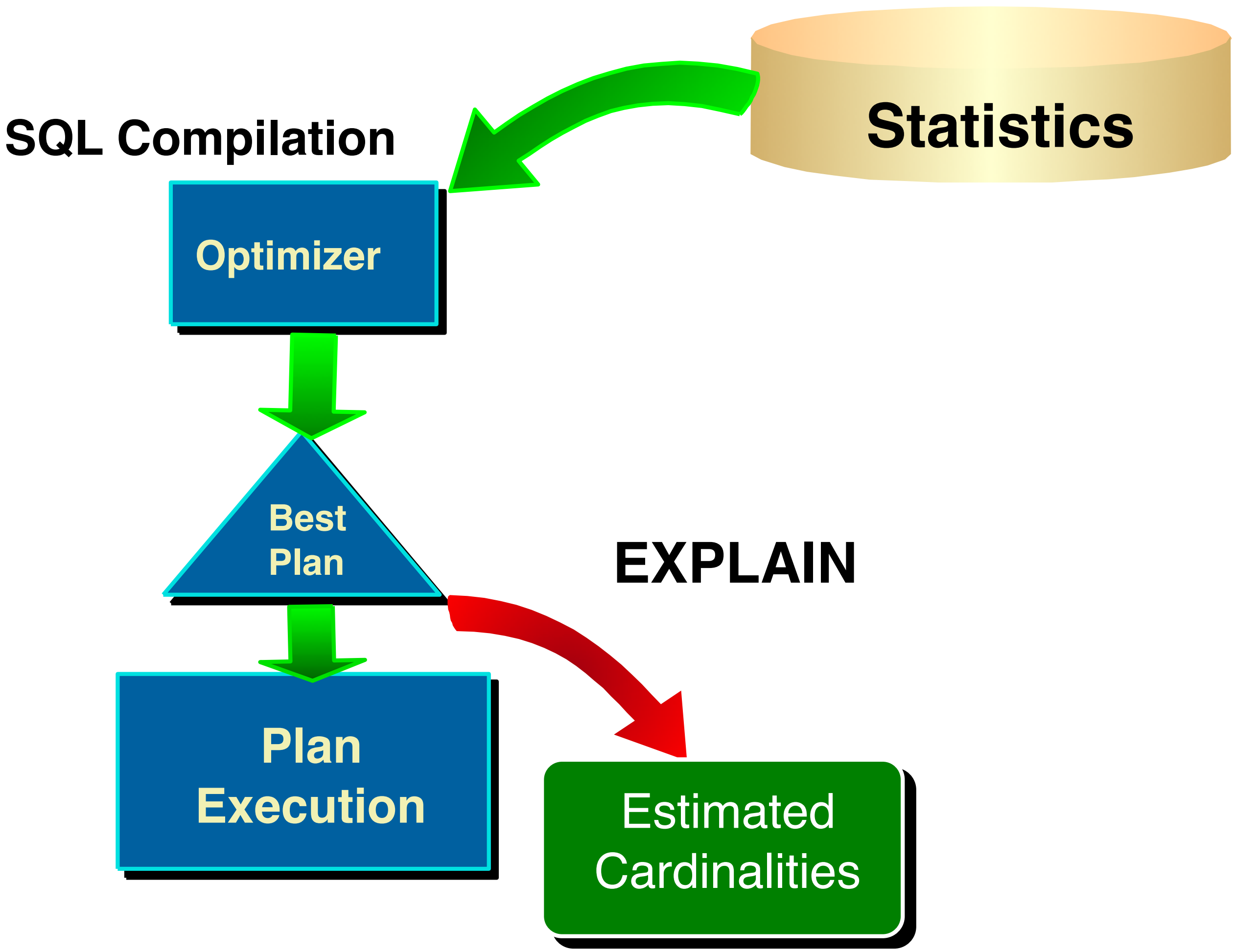
- Default is to collect statistics on individual columns
- *LEO* automatically determines **statistics profiles**
  - What statistics are needed for this workload?
  - **Column groups** to collect statistics on
    - Too many to collect all combinations of columns
    - Detects correlation between columns, e.g.
    - **WHERE *Make* = 'Honda' AND *Model* = 'Accord'**
    - By comparing actual cardinalities to optimizer's estimates
    - **True learning with feedback!**
- **Improves access plan selection in future queries**

Reference: Stillger, Lohman, Markl, Kandil: *LEO — DB2's LEarning Optimizer*, **VLDB 2001 (Rome, Sept. 2001),** 19-28

Traditional Query Optimization (without LEO)



EXPLAIN Gives Optimizer's Estimates



New: Capture Actual Number of Rows!

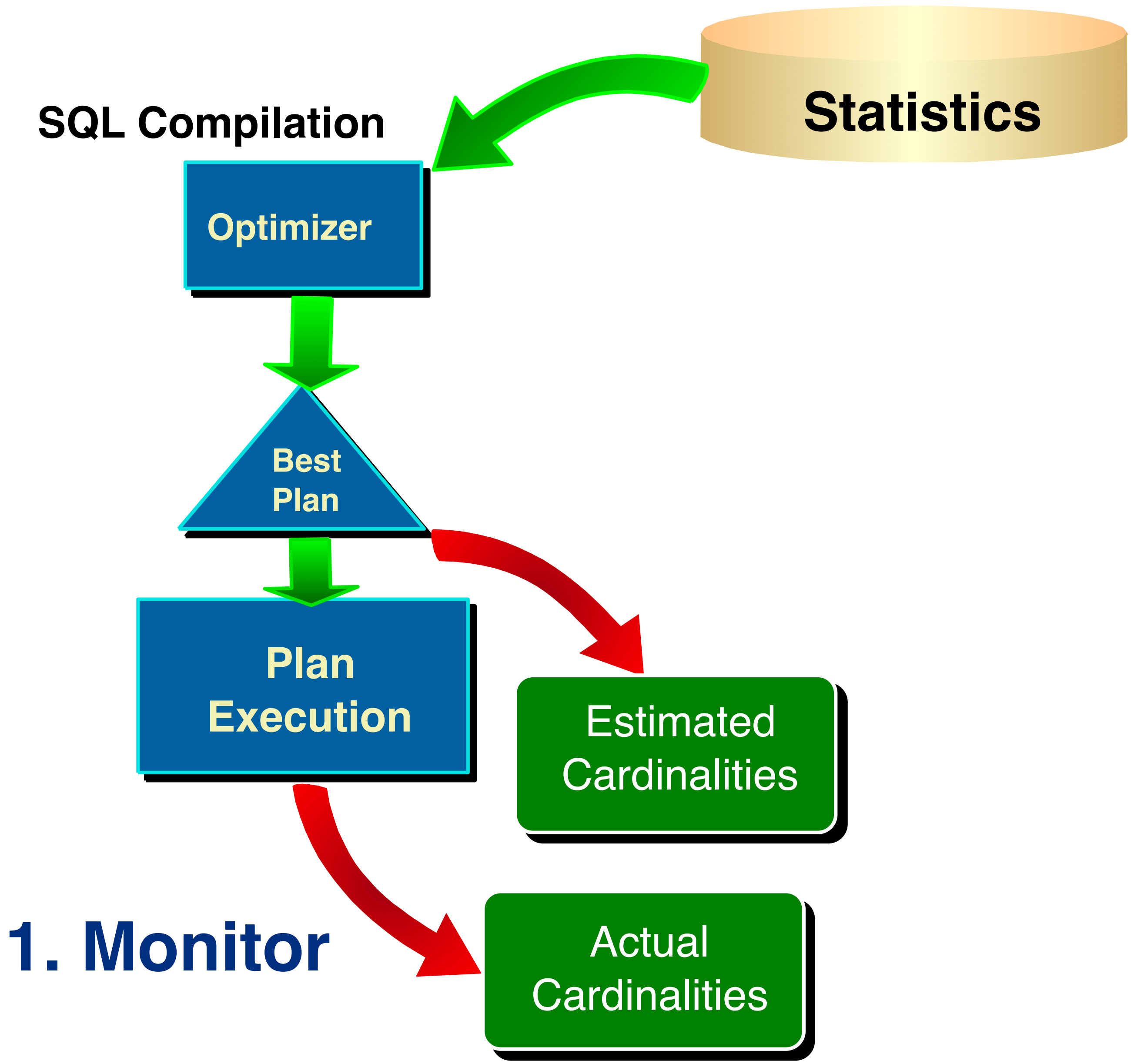
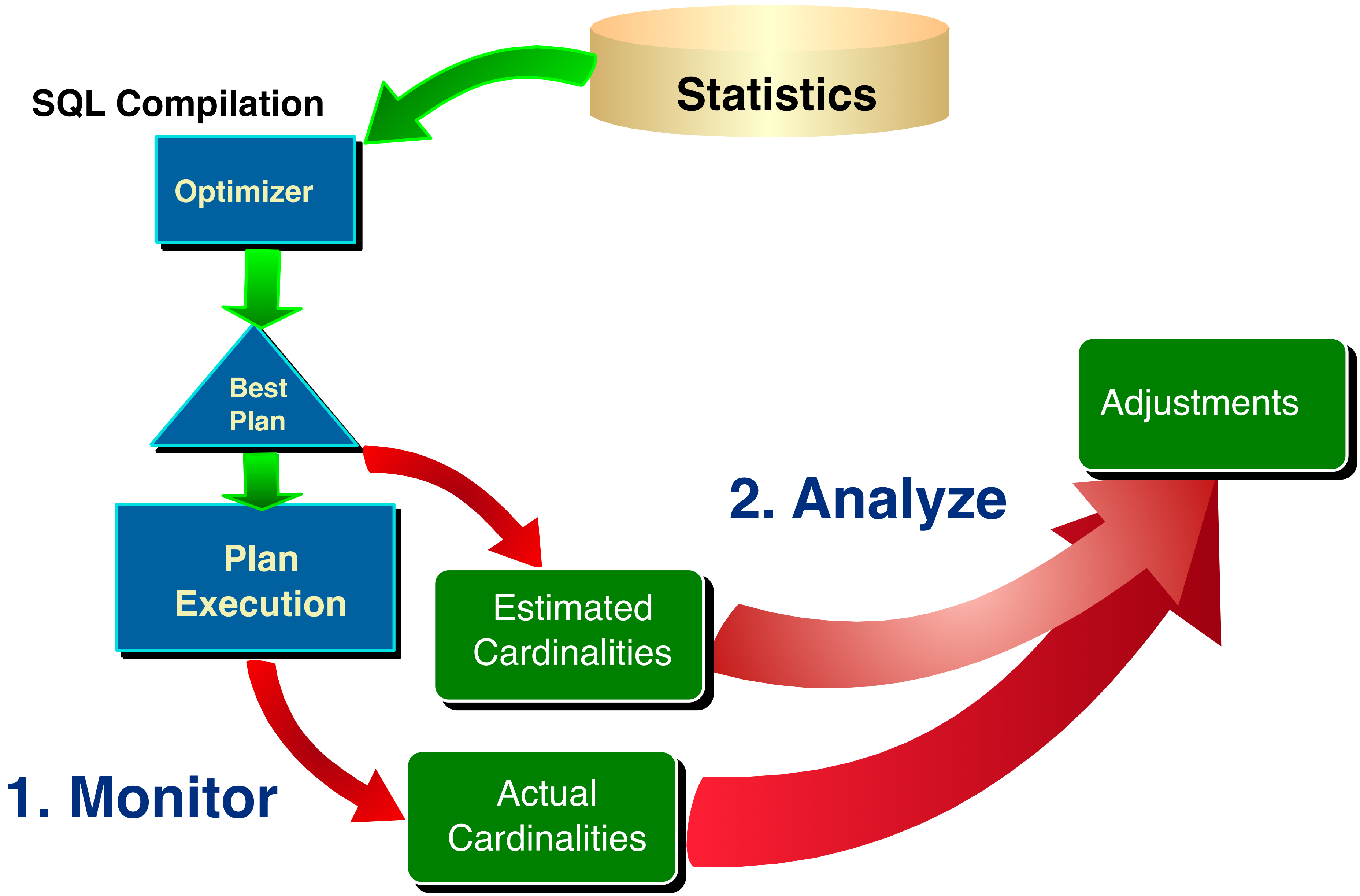
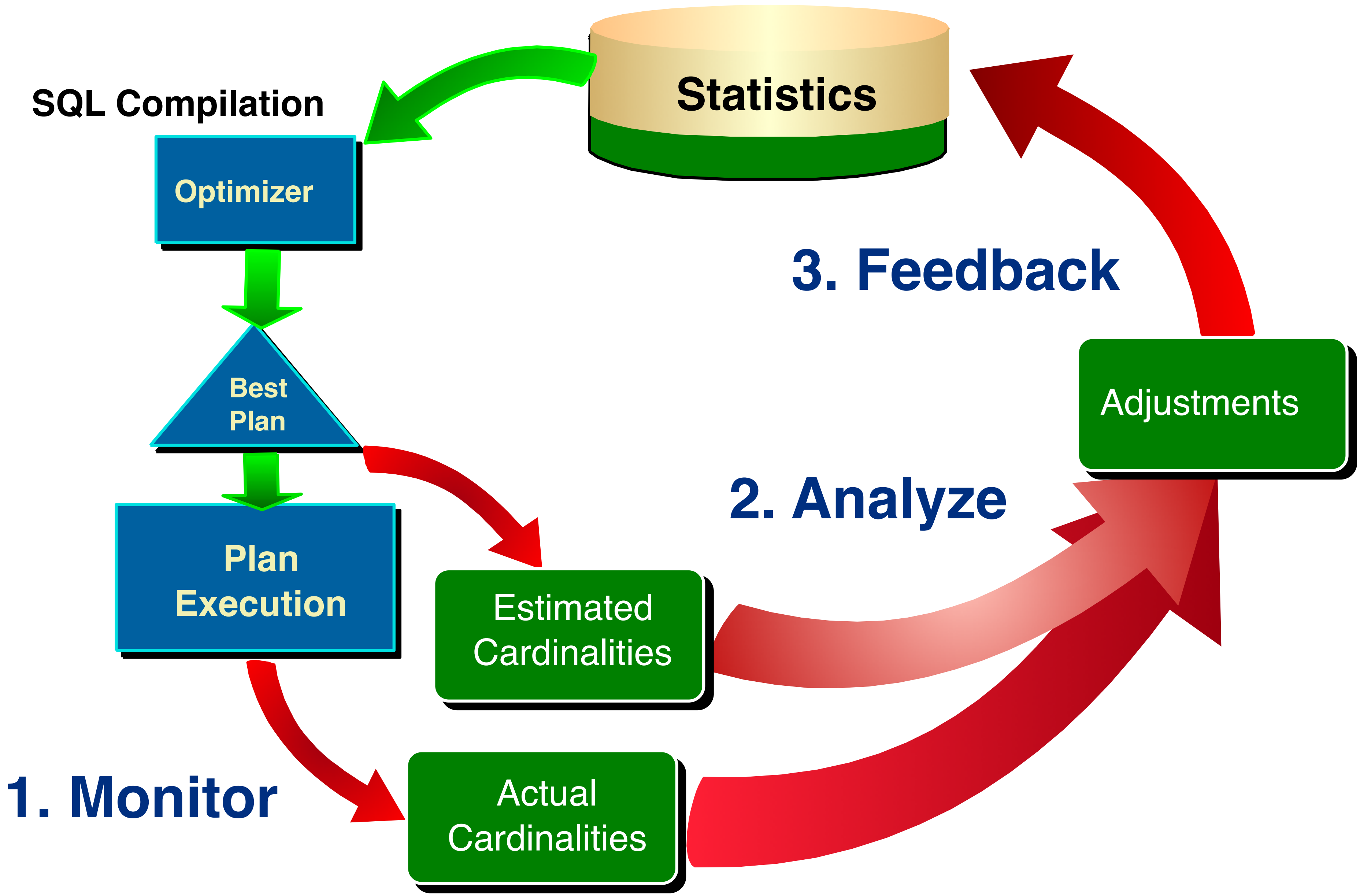


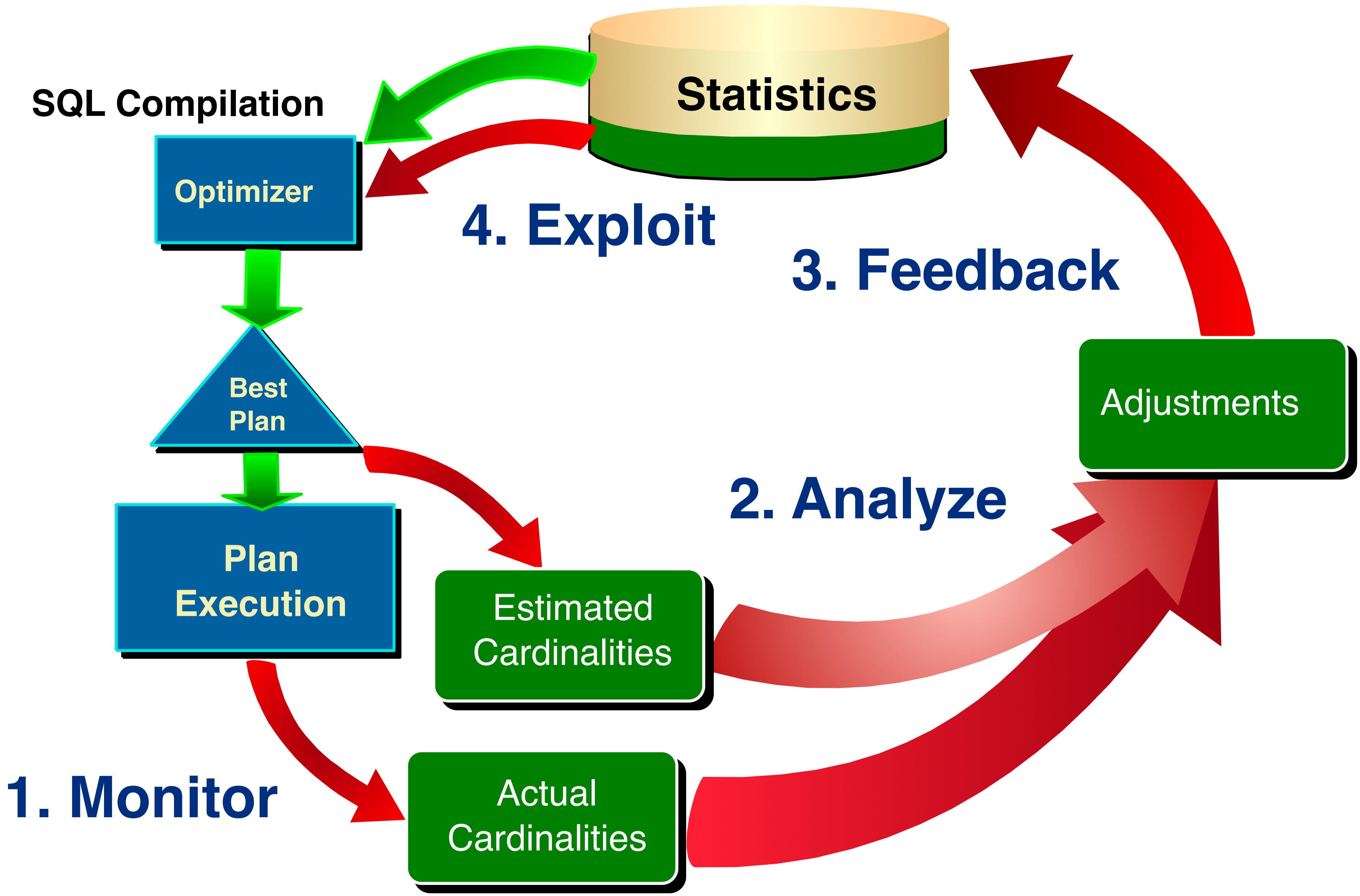
Figure Out Where the Differences Are



Augment Statistics with Adjustments

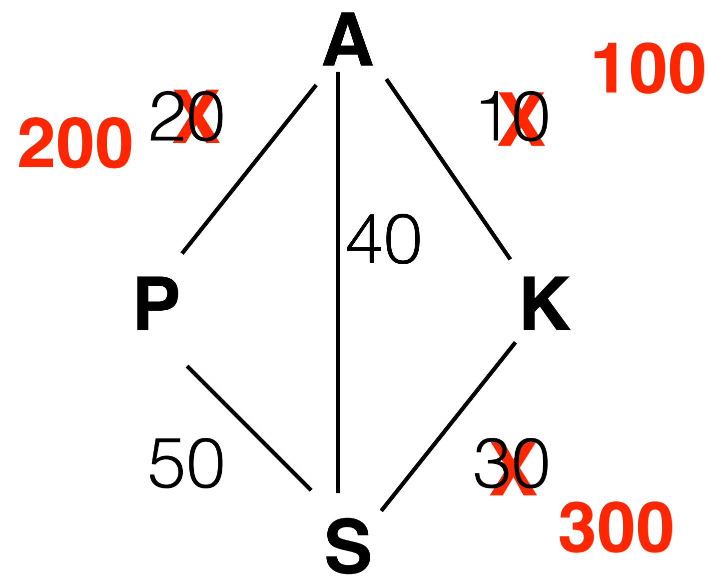


Exploit: Learning in Query Optimization!



A Danger with Actuals:
"Fleeing from Knowledge to Ignorance"

Query Graph:



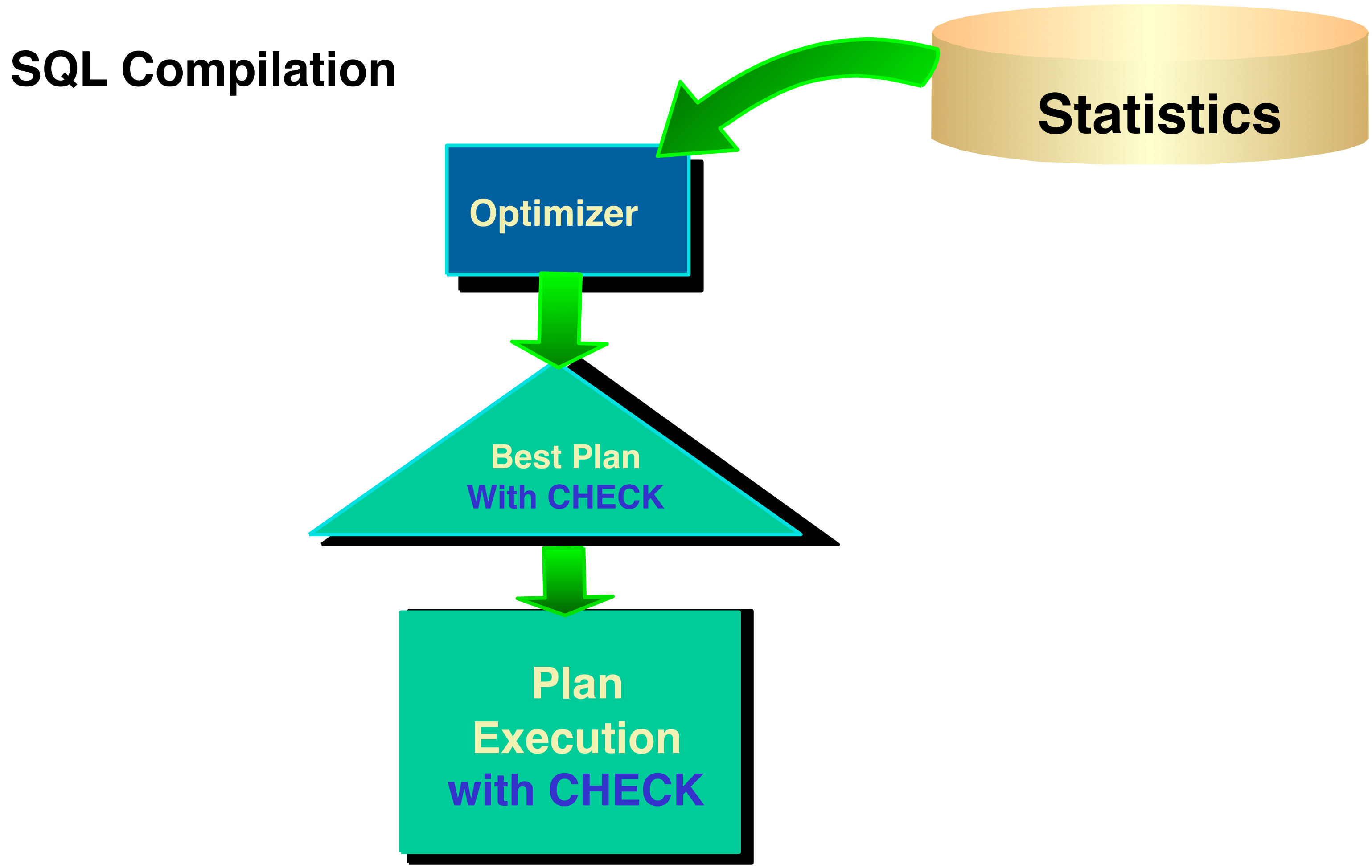
Plan 1: ((A * K) * P) * S

Plan 2: ((A * P) * (S * K)

Best Plan: ((A * S) * P) * K

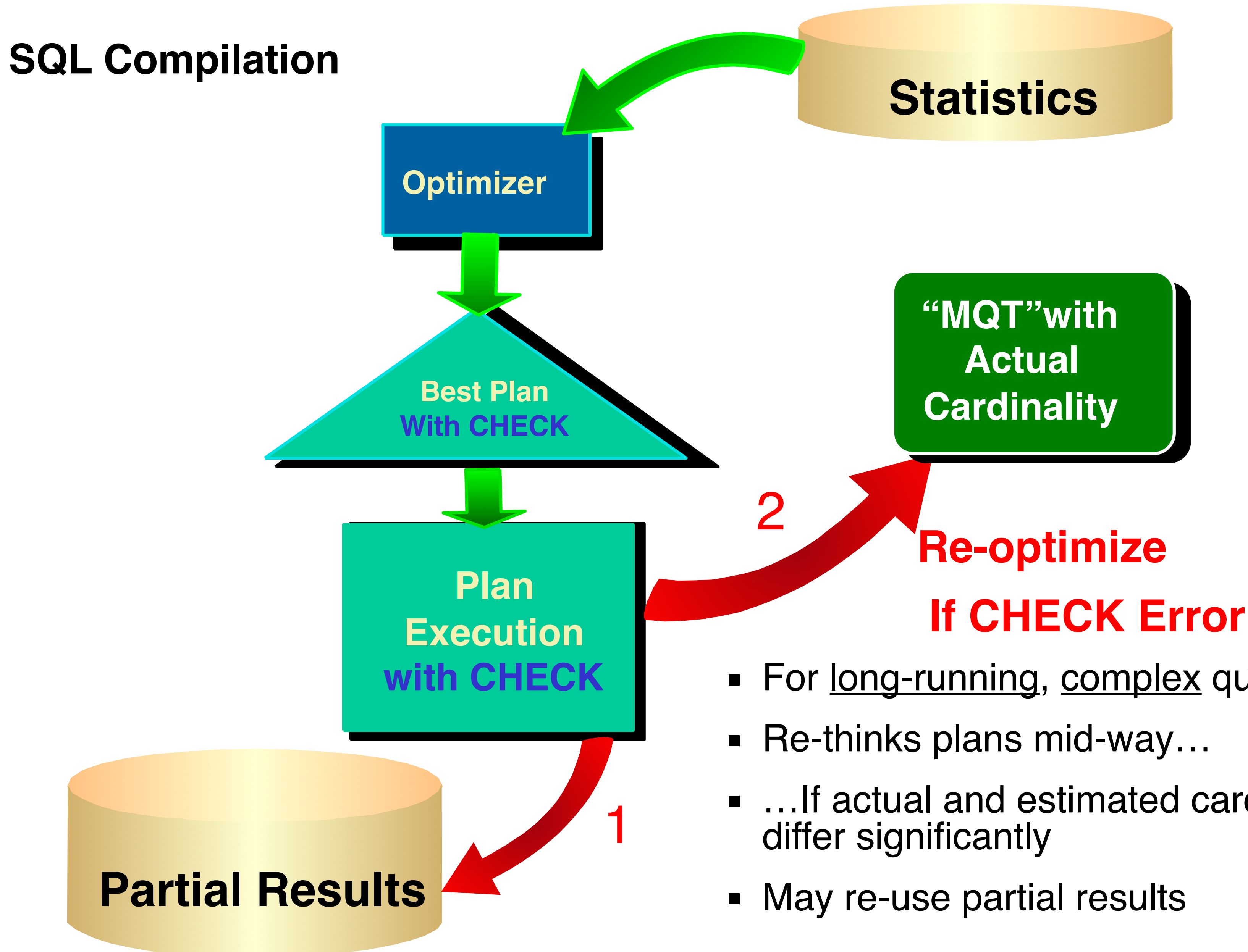
Reference:
Srivastava, Haas, Markl, Kutsch, Tran: *ISOMER: Consistent Histogram Construction Using Query Feedback.* **ICDE 2006**: 39

Solution #3: Why Wait Till the Query is Finished? Progressive OPtimization (POP)



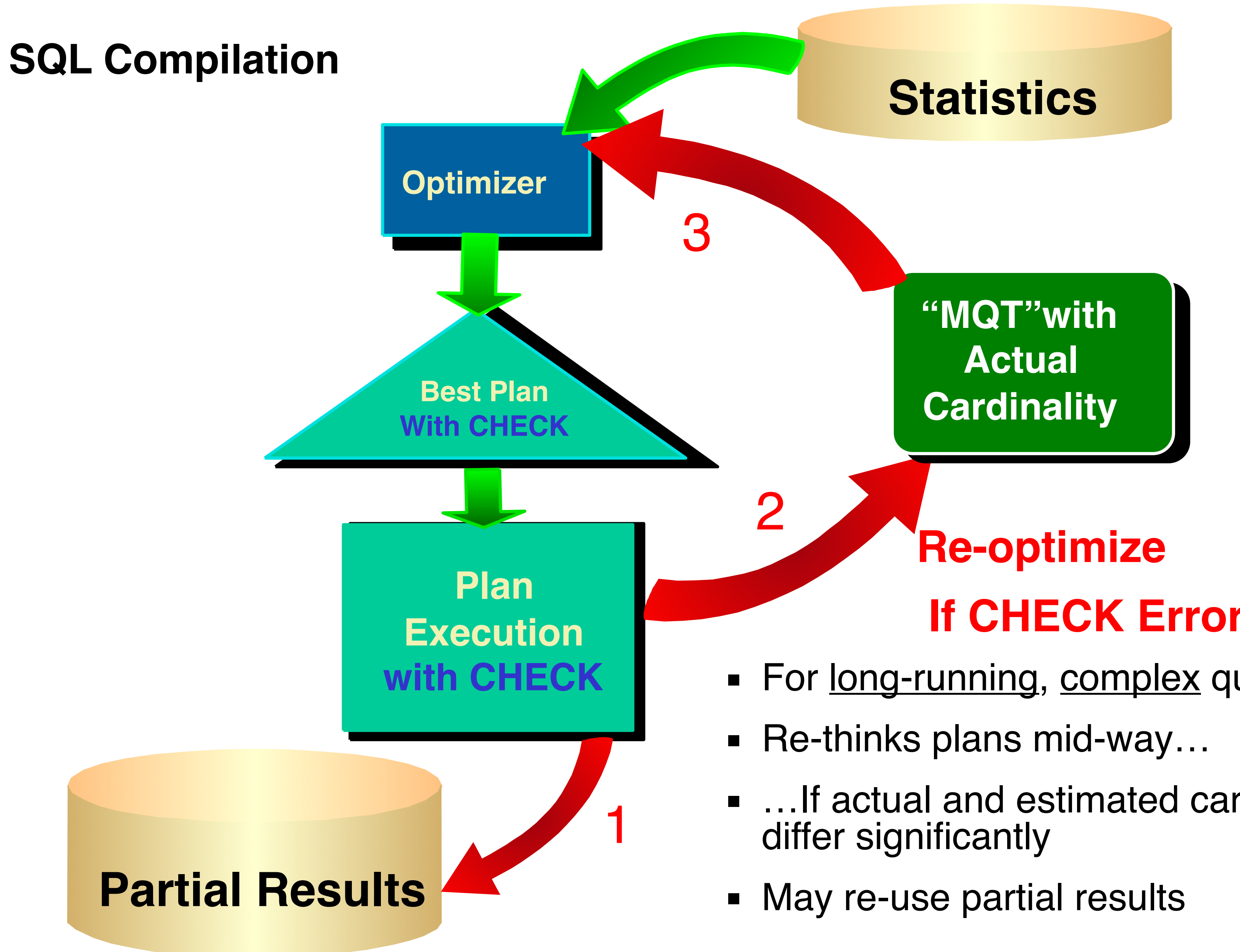
Reference: Markl, Raman, Simmen, Lohman, Pirahesh: *Robust Query Processing through Progressive Optimization*. **SIGMOD 2004:** 659-670

Save Partial Results and Actual Cardinality



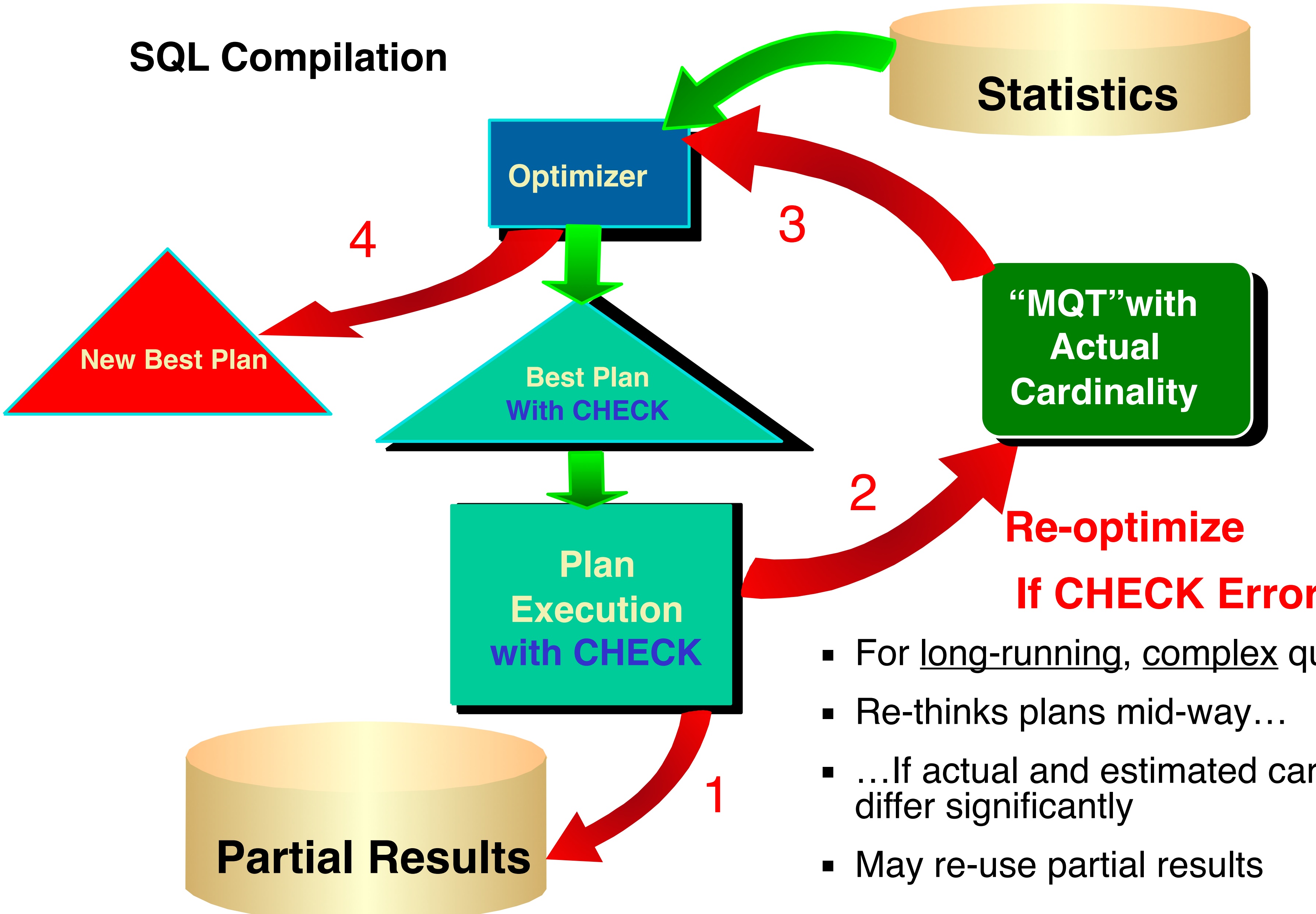
- For long-running, complex queries
- Re-thinks plans mid-way…
- …If actual and estimated cardinality differ significantly
- May re-use partial results

Re-optimize using actual cardinality



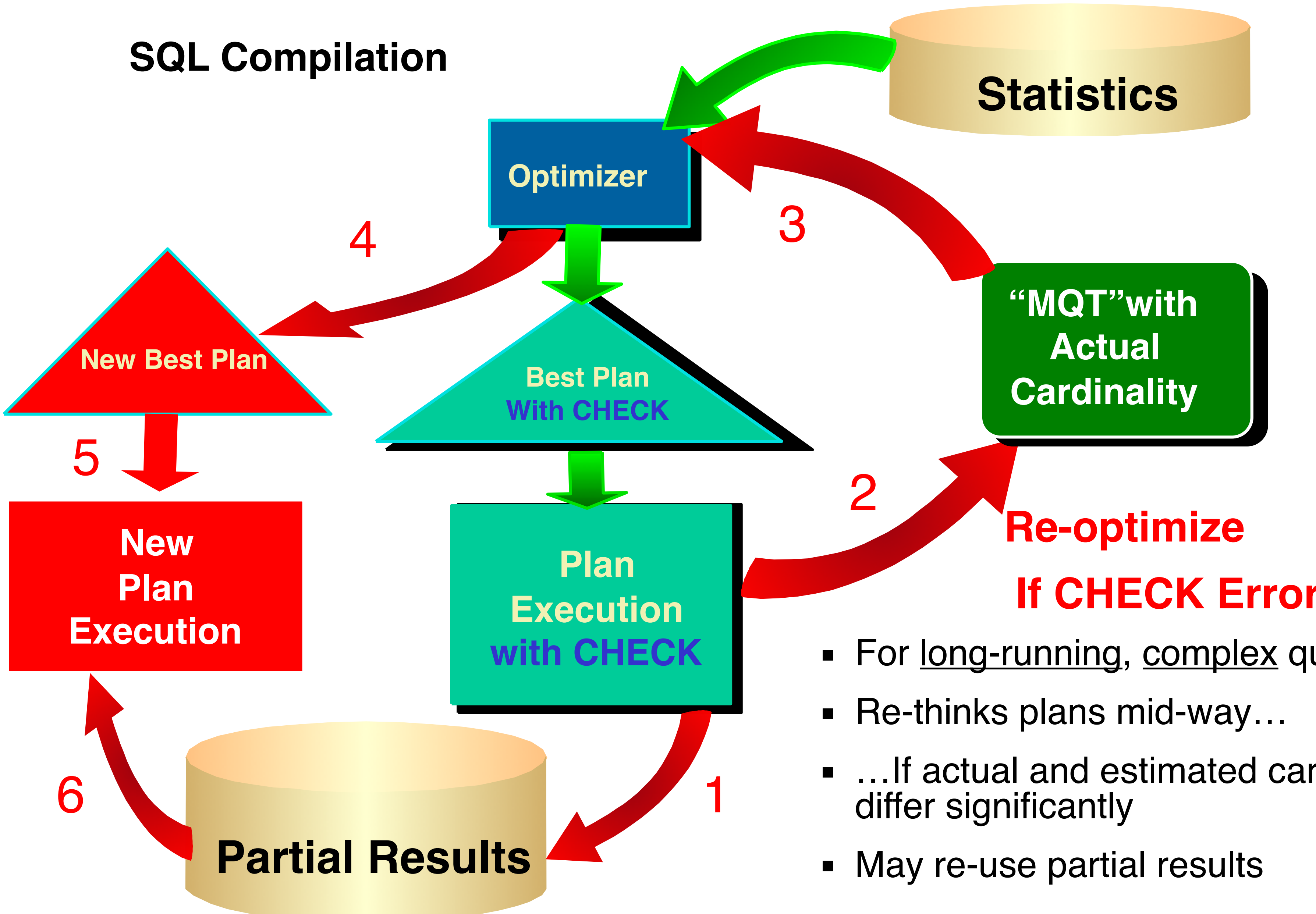
- For <u>long-running</u>, <u>complex</u> queries
- Re-thinks plans mid-way…
- …If actual and estimated cardinality differ significantly
- May re-use partial results

Create new best plan, using actuals



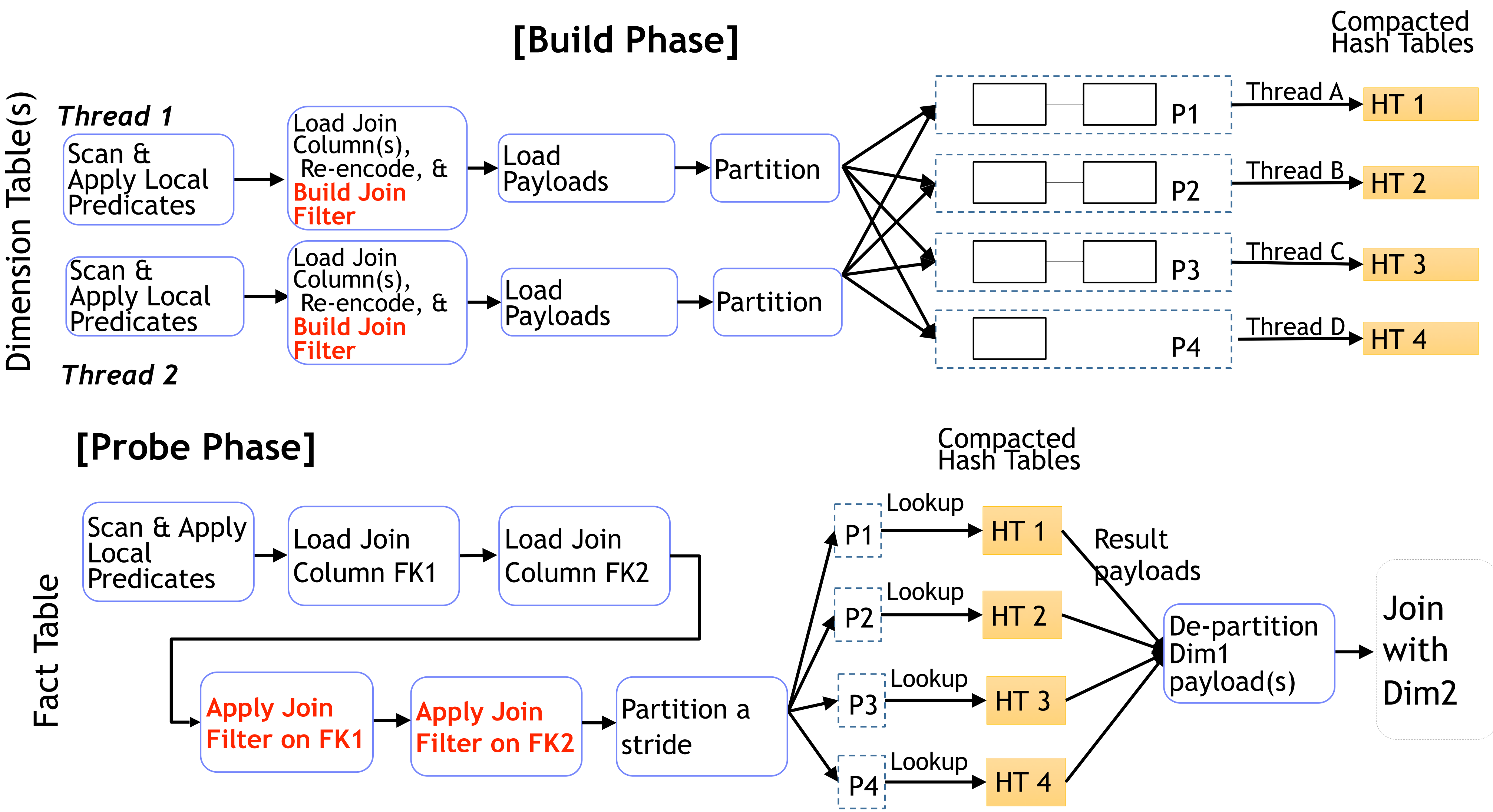
- For <u>long-running</u>, <u>complex</u> queries
- Re-thinks plans mid-way…
- …If actual and estimated cardinality differ significantly
- May re-use partial results

Execute new plan, optionally using earlier results



- For <u>long-running</u>, <u>complex</u> queries
- Re-thinks plans mid-way…
- …If actual and estimated cardinality differ significantly
- May re-use partial results

Solution #4: More Robust Execution Strategies: "Bloom Joins" in DB2 BLU



Refn: Barber, Lohman, Raman, Sidle, Lightstone, Schiefer: *In-memory BLU acceleration in IBM's DB2 and dashDB: Optimized for modern workloads and hardware architectures.* **ICDE 2015:** 1246-1252

Don't Forget…

The <u>Real</u> Goal of Query Optimization is…

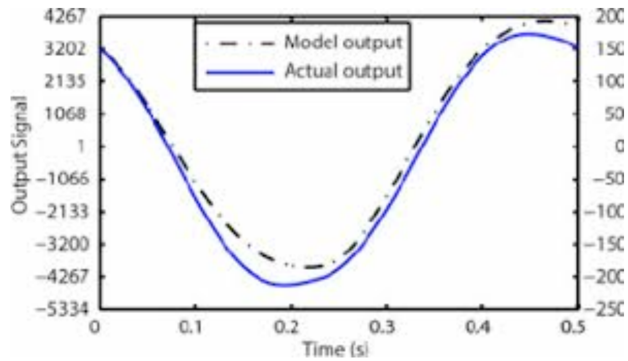
<u>NOT</u> to find the Very Best Plan,

but

to Avoid the Really Bad Plans

Validating an Optimizer's Model(s)

- *Any unvalidated model isn't worth the paper it's written on!*
- **Currently done by exception:**
  - **When customer complains about a "bad plan", or**
  - **When a test case breaks**
  - **BUT this severely limits the validation process to a few breakpoints**
- **Need to compare optimizer's choice (estimate) to actual best plan!**
- **Requires testing all permutations of:**
  - **Possible values in its parametric space**
    - All database designs (normalization, indexes, partitioning,…), even bad designs!
    - All table characteristics (statistics, values!)
    - All SQL queries, of arbitrary complexity
      - values of parameter markers
      - negation and disjuncts
      - inequality join predicates
      - complex SQL constructs (CASE statements, subqueries, 4-page queries, …)
      - rare SQL constructs & corner cases
  - **Environments**
    - **Hardware**
      - Not just the latest & greatest!
      - Any degree of parallelism, on any number of nodes
    - **Competing workloads on same system**
- *But this is a huge, daunting, probably impossible task!!*
- Gets harder the more complex the model
- Some modest attempts:
  - Mackert & Lohman, *R* Optimizer Validation and Performance Evaluation for Local Queries.* **SIGMOD 1986**: 84-95
  - Mackert & Lohman, *R* Optimizer Validation and Performance Evaluation for Distributed Queries.* **VLDB 1986**: 149-159
  - Leis, Gubichev, Mirchev, Boncz, Kemper, Neumann: *How Good Are Query Optimizers, Really?* **PVLDB 9(3) (2015)**: 204-215



Conclusions

- **Query optimization is generally very successful**
- **Optimizers are mathematical models**
  - **Assumptions underlying those models can cause <u>order-of-magnitude errors</u>**
    - Especially in cardinality, the "Achilles' Heel" of optimization
    - Need to minimize impact of assumptions
  - **Need to validate our optimizer models thoroughly**
    - Requires determining the right metric
    - Requires testing all permutations of data, statistics, & environment
- **Numerous problems still remain, but researchers are (generally) ignoring the important ones!**

What ARE the Important Ones?

Should focus on areas having the <u>most impact:</u>

- **Detect and correct correlations**, especially across joins
- **More realistically model** the increased…
  - **Dynamic aspects** from one execution of a query to the next
  - **Competition for resources** among concurrent queries & apps
  - **Parallelism** among resources
- **Automatically learn** from prior or current execution
- **Automatically adapt plans** to new information from
  - Prior executions
  - This execution
- Devise **robust execution strategies** that are less sensitive to estimation errors
- **Model modern environments**
  - Multi-core
  - Cloud metrics, including SLA penalties
  - Big Data applications
    - May not have basic statistics!
    - May have "foreign" optimizers

Ευχαριστώ!

Greek

धन्यवाद

Hindi

多謝

Traditional Chinese

ขอบคุณ

Thai

Спасибо

Russian

Gracias

Spanish

شكراً

Arabic

Thank You

English

Obrigado

Brazilian Portuguese

Grazie

Italian

多谢

Simplified Chinese

Danke

German

Merci

French

நன்றி

Tamil

ありがとうございました

Japanese

감사합니다

Korean