

15-721 DATABASE SYSTEMS



Lecture #18 – Query Planning (Cost Models)

Andy Pavlo // Carnegie Mellon University // Spring 2016

TODAY'S AGENDA

Cost Models

Cost Estimation

Working with a large code base

COST-BASED QUERY PLANNING

Generate an estimate of the cost of executing a particular query plan for the current state of the database.

→ Estimates are only meaningful internally.

This is independent of the search strategies that we talked about last class.

COST MODEL COMPONENTS

Choice #1: Physical Costs

- Predict CPU cycles, I/O, cache misses, RAM consumption, pre-fetching, etc...
- Depends heavily on hardware.

Choice #2: Logical Costs

- Estimate result sizes per operator.
- Independent of the operator algorithm.
- Need estimations for operator result sizes.

Choice #3: Algorithmic Costs

- Complexity of the operator algorithm implementation.

DISK-BASED DBMS COST MODEL

The number of disk accesses will always dominate the execution time of a query.

- CPU costs are negligible.
- Can easily measure the cost per I/O.

This is easier to model if the DBMS has full control over buffer management.

- We will know the replacement strategy, pinning, and assume exclusive access to disk.

IN-MEMORY DBMS COST MODEL

No I/O costs, but now we have to account for CPU and memory access costs.

Memory cost is more difficult because the DBMS has no control cache management.

→ Unknown replacement strategy, no pinning, shared caches, non-uniform memory access.

The number of tuples processed per operator is a reasonable estimate for the CPU cost.

SMALLBASE COST MODEL

Two-phase model that automatically generates hardware costs from a logical model.

Phase #1: Identify Execution Primitives

- List of ops that the DBMS does when executing a query
- Example: evaluating predicate, index probe, sorting.

Phase #2: Microbenchmark

- On start-up, profile ops to compute CPU/memory costs
- These measurements are used in formulas that compute operator cost based on table size.



OBSERVATION

The number of tuples processed per operator depends on three factors:

- The access methods available per table
- The distribution of values in the database's attributes
- The predicates used in the query

Simple queries are easy to estimate.

More complex queries are not.

SELECTIVITY

The selectivity of an operator is the percentage of data accessed for a predicate.

→ Modeled as probability of whether a predicate on any given tuple will be satisfied.

The DBMS estimates selectivities using:

- Domain Constraints
- Min/Max Statistics
- Histograms

RESULT CARDINALITY

The number of tuples that will be generated per operator is computed from its selectivity multiplied by the number of tuples in its input.

RESULT CARDINALITY

Assumption #1: Uniform Data

→ The distribution of values (except for the heavy hitters) is the same.

Assumption #2: Independent Predicates

→ The predicates on attributes are independent

Assumption #3: Inclusion Principle

→ The domain of join keys overlap such that each key in the inner relation will also exist in the outer table.

CORRELATED ATTRIBUTES

Consider a database of automobiles:

→ # of Makes = 10, # of Models = 100

And the following query:

→ (make="Honda" **AND** model="Accord")

With the independence and uniformity assumptions, the selectivity is:

→ $1/10 \times 1/100 = 0.001$

But since only Honda makes Accords the real selectivity is $1/100 = 0.01$

COLUMN GROUP STATISTICS

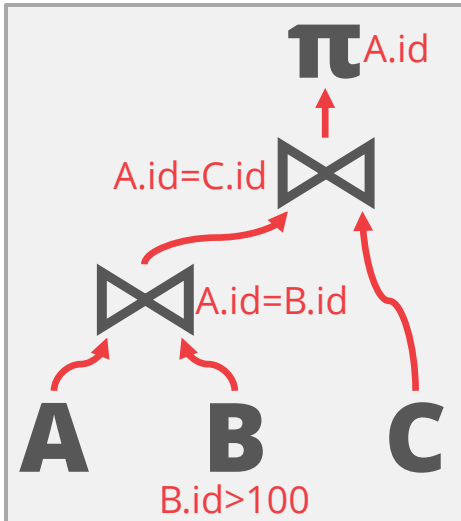
The DBMS can track statistics for groups of attributes together rather than just treating them all as independent variables.

- Only supported in commercial systems.
- Requires the DBA to declare manually.

ESTIMATION PROBLEM

```

SELECT A.id
FROM A, B, C
WHERE A.id = B.id
AND A.id = C.id
AND B.id > 100
  
```



Compute the cardinality of base tables

$$A \rightarrow |A|$$

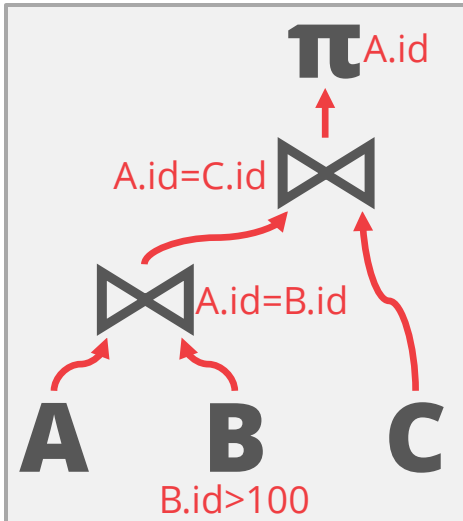
$$B.id > 100 \rightarrow sel(B.id > 100)$$

$$C \rightarrow |C|$$

ESTIMATION PROBLEM

```

SELECT A.id
FROM A, B, C
WHERE A.id = B.id
AND A.id = C.id
AND B.id > 100
  
```



Compute the cardinality of base tables

$$A \rightarrow |A|$$

$$B.id > 100 \rightarrow sel(B.id > 100)$$

$$C \rightarrow |C|$$

Compute the cardinality of join results

$$A \bowtie B = (|A| \cdot |B|) / \max(sel(A.id = B.id), sel(B.id > 100))$$

$$(A \bowtie B) \bowtie C = (|A| \cdot |B| \cdot |C|) / \max(sel(A.id = B.id), sel(B.id > 100), sel(A.id = C.id))$$

ESTIMATOR QUALITY

Evaluate the correctness of cardinality estimates generated by DBMS optimizers as the number of joins increases.

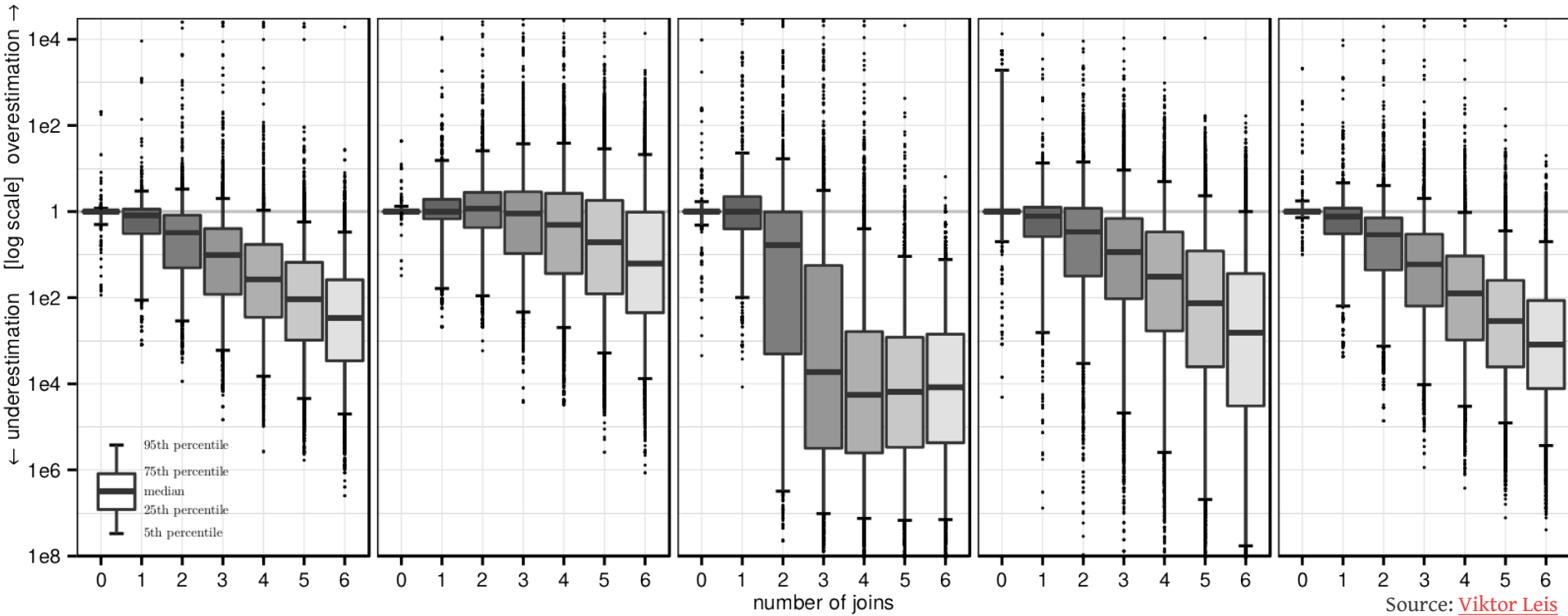
- Let each DBMS perform its stats collection.
- Extract measurements from query plan

Compared five DBMSs using 100k queries.



HOW GOOD ARE QUERY OPTIMIZERS, REALLY?
VLDB 2015

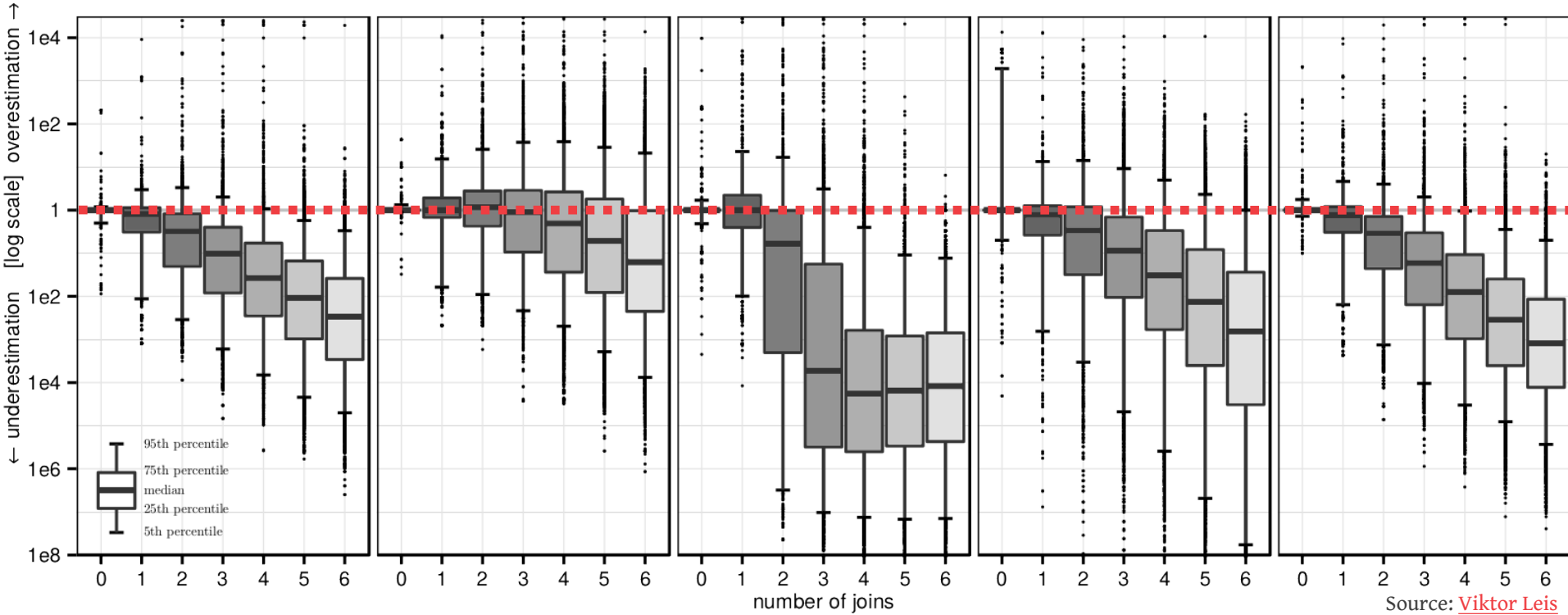
ESTIMATOR QUALITY



Source: [Viktor Leis](#)

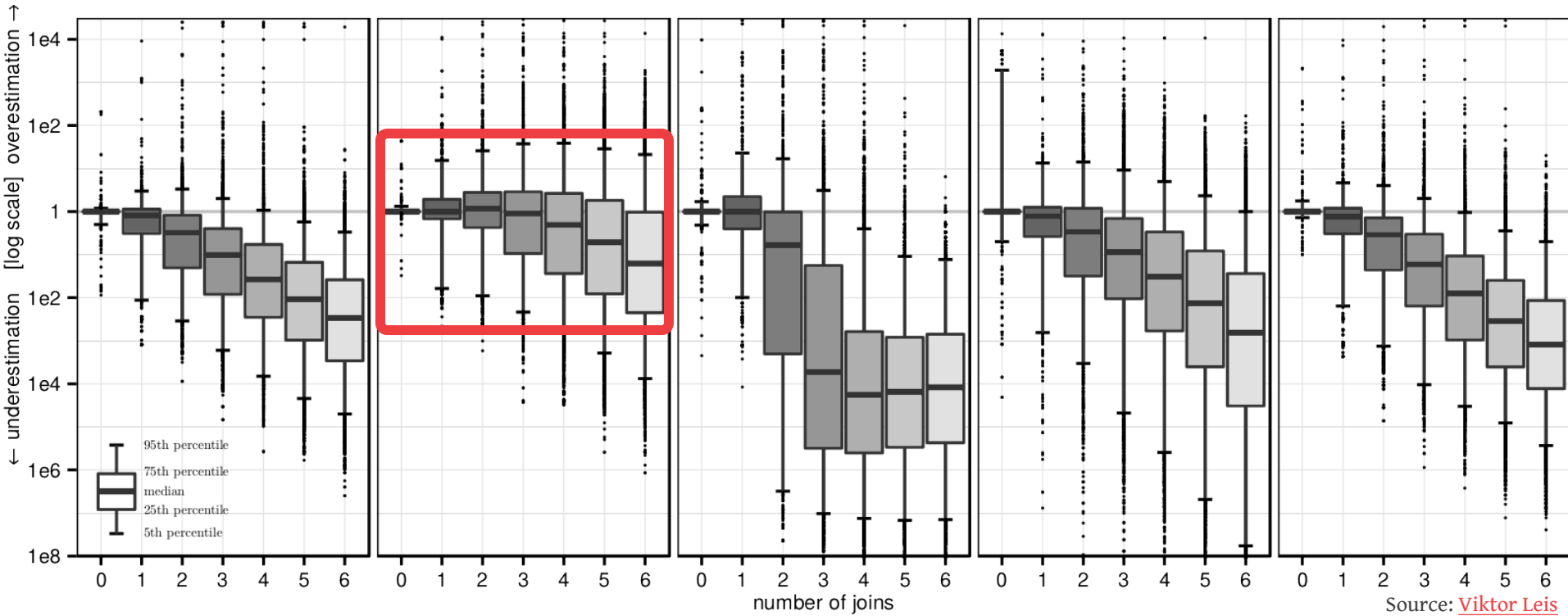
CMU 15-721 (Spring 2016)

ESTIMATOR QUALITY

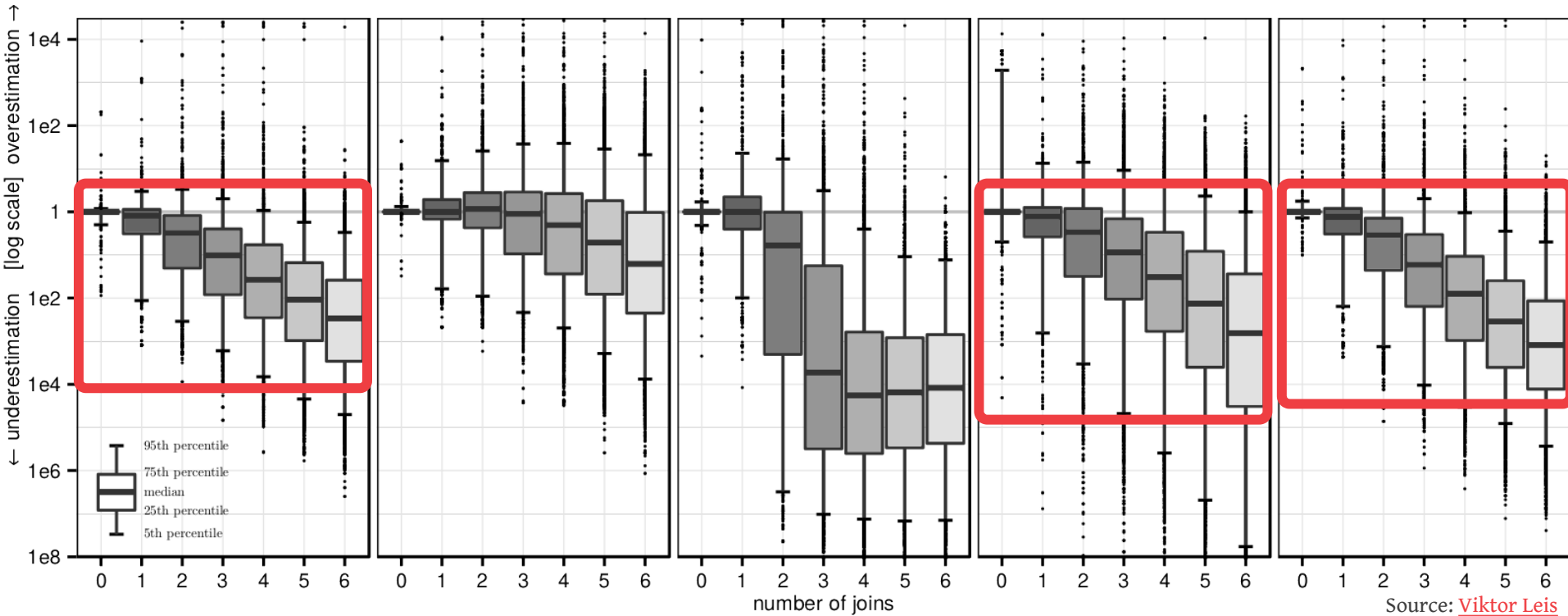
Source: [Viktor Leis](#)

CMU 15-721 (Spring 2016)

ESTIMATOR QUALITY



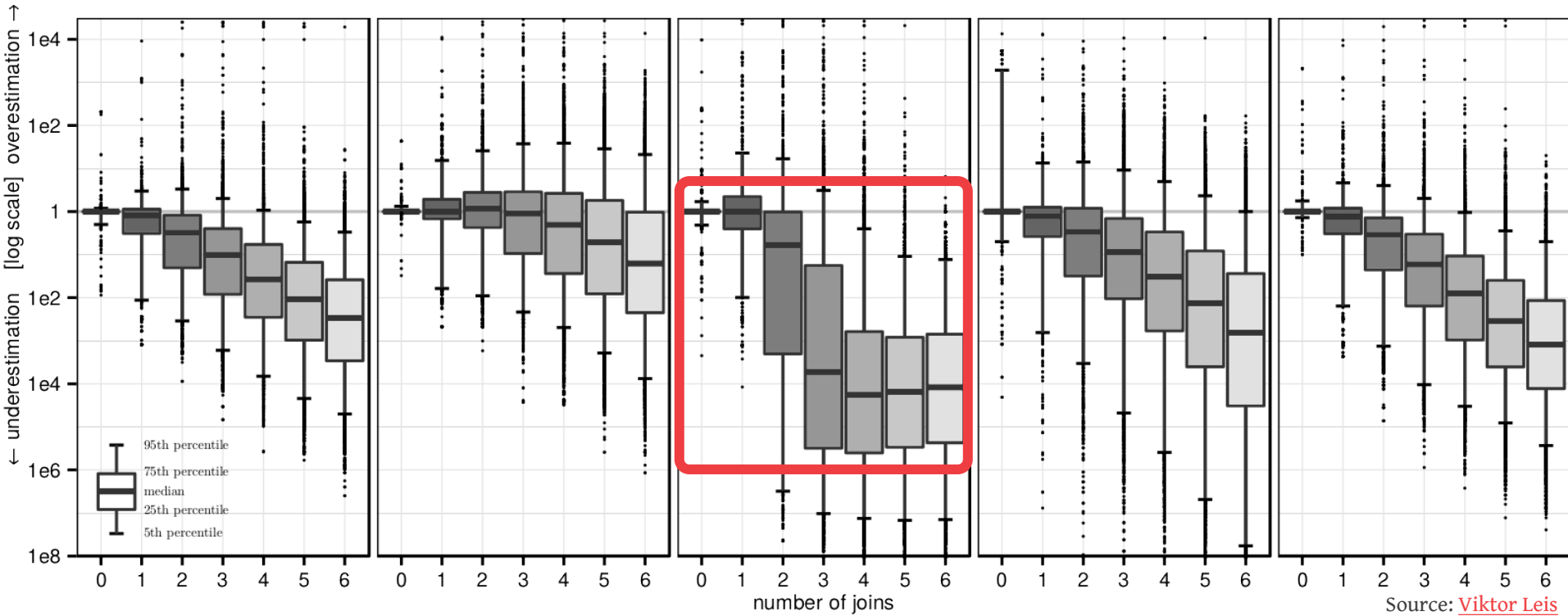
ESTIMATOR QUALITY



Source: [Viktor Leis](#)

CMU 15-721 (Spring 2016)

ESTIMATOR QUALITY



ESTIMATOR QUALITY

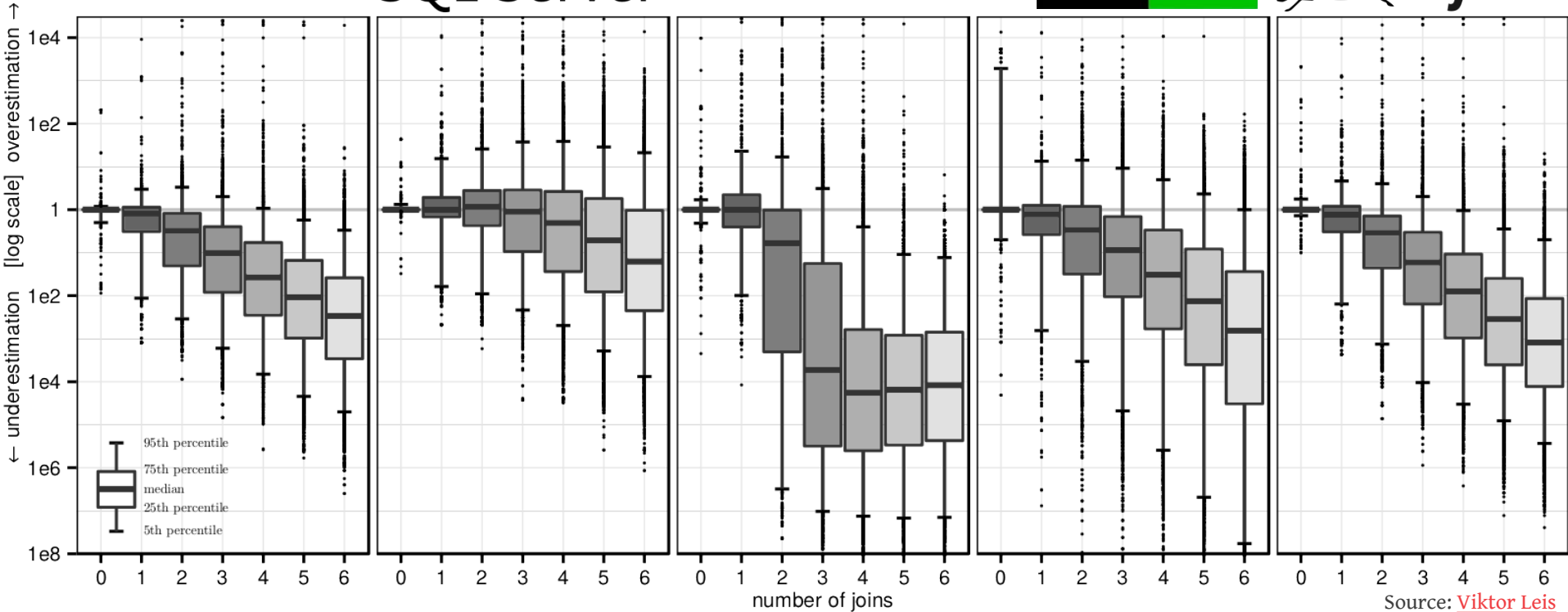


Microsoft®
SQL Server®

ORACLE®

IBM

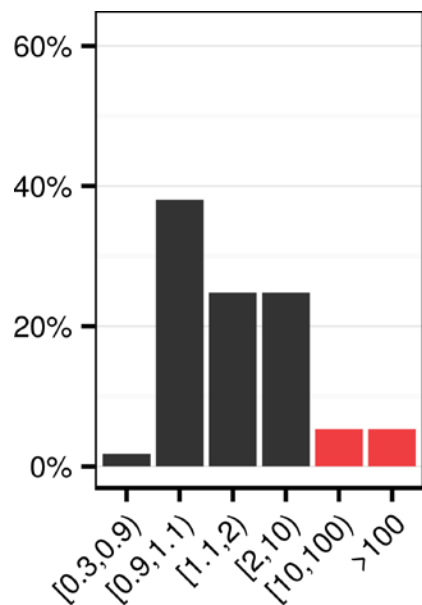
DB2



EXECUTION SLOWDOWN

Postgres 9.4 – JOB Workload

Default Planner

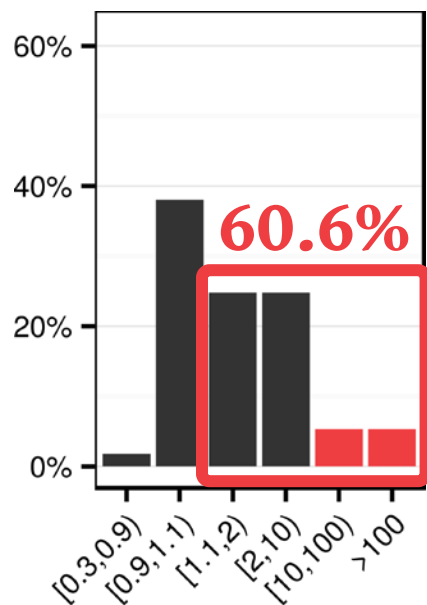


Slowdown compared to using true cardinalities

EXECUTION SLOWDOWN

Postgres 9.4 – JOB Workload

Default Planner



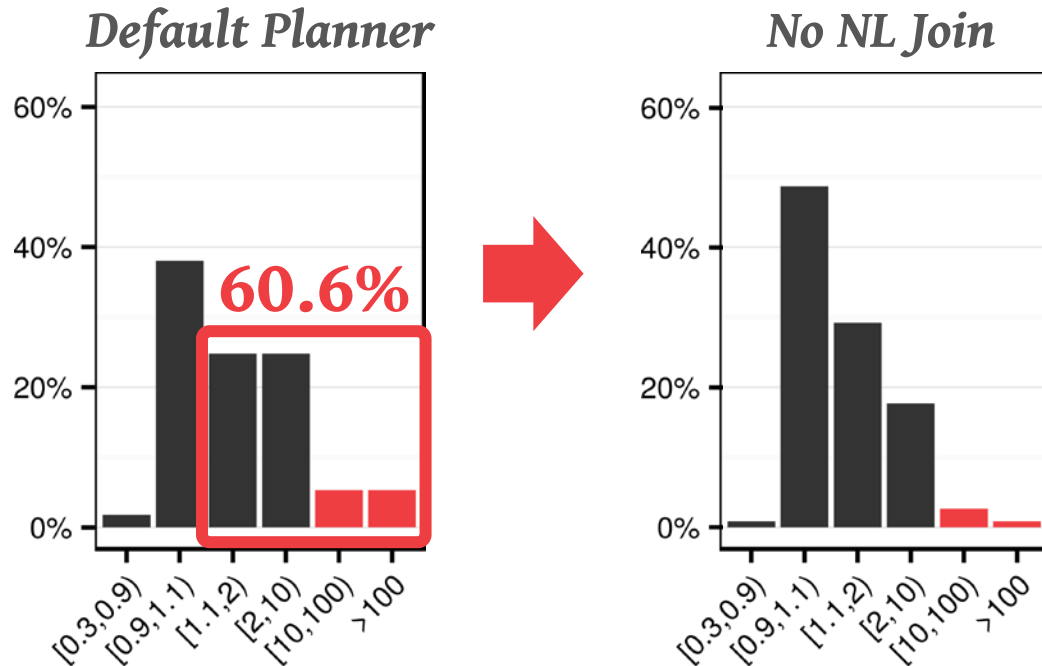
Slowdown compared to using true cardinalities

Source: [Viktor Leis](#)

CMU 15-721 (Spring 2016)

EXECUTION SLOWDOWN

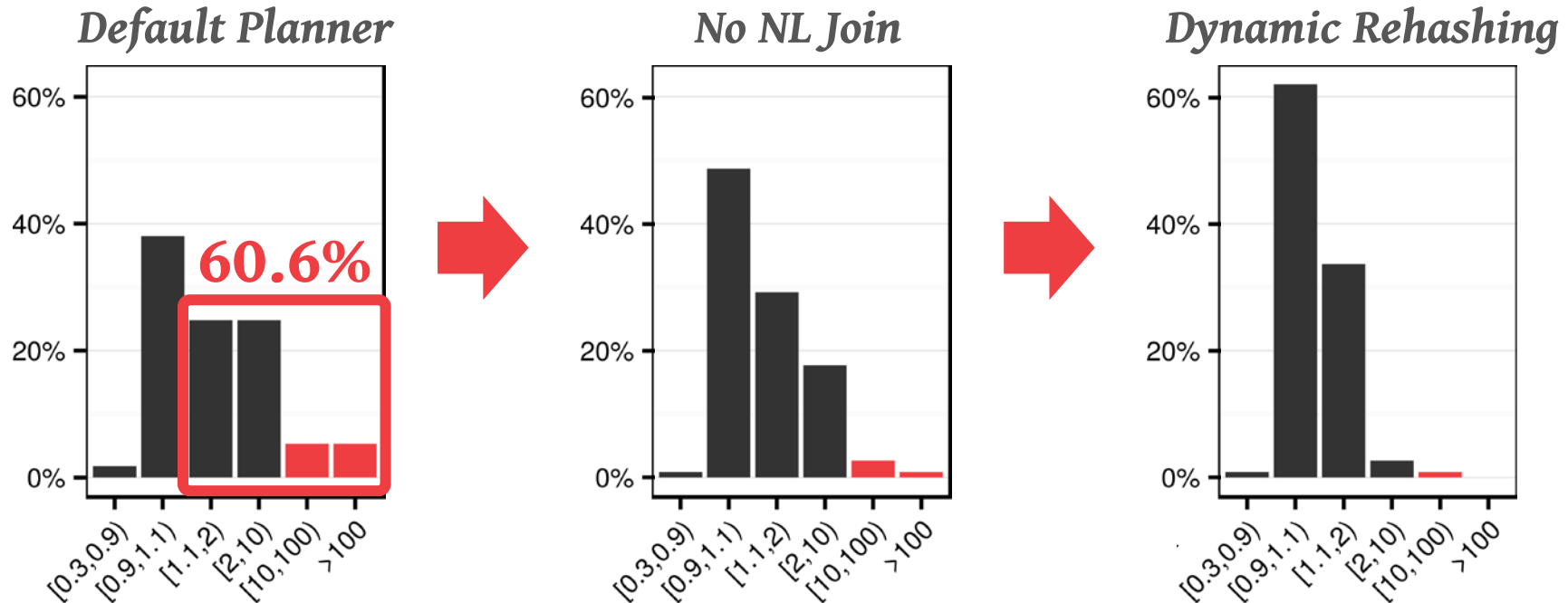
Postgres 9.4 – JOB Workload



Slowdown compared to using true cardinalities

EXECUTION SLOWDOWN

Postgres 9.4 – JOB Workload



Slowdown compared to using true cardinalities

LESSONS FROM THE GERMANS

Query opt is more important than a fast engine
→ Cost-based join ordering is necessary

Cardinality estimates are routinely wrong
→ Try to use operators that do not rely on estimates

Hash joins + seq scans are a robust exec model
→ The more indexes that are available, the more brittle the plans become (but also faster on average)

Working on accurate models is a waste of time
→ Better to improve cardinality estimation instead

PARTING THOUGHTS

Using number of tuples processed is a reasonable cost model for in-memory DBMSs.
→ But computing this is non-trivial.

If you are building a new DBMS, then using Volcano/Cascade planning + # of tuples cost model is the way to go.



ANDY'S TRILL
**LIFE LESSONS
FOR WORKING
ON CODE**

DISCLAIMER

I have worked on a few large projects in my lifetime (2 DBMSs, 1 distributed system).

I have also read a large amount of “enterprise” code for legal stuff over multiple years.

But I’m not claiming to be all knowledgeable in modern software engineering practices.

OBSERVATION

Most software development is never from scratch. You will be expected to be able to work with a large amount of code that you did not write.

Being able to independently work on a large code base is the #1 skill that companies tell me they are looking for in students they hire.

PASSIVE READING

Reading the code for the sake of reading code is (usually) a waste of time.

→ It's hard to internalize functionality if you don't have direction.

It's important to start working with the code right away to understand how it works.

TEST CASES

Adding or improving tests allows you to improve the reliability of the code base without the risk of breaking production code.
→ It forces you to understand code in a way that is not possible when just reading it.

Nobody will complain (hopefully) about adding new tests to the system.

REFACTORING

Find the general location of code that you want to work on and start cleaning it up.

- Add/edit comments
- Clean up messy code
- Break out repeated logic into separate functions.

Tread lightly though because you are changing code that you are not familiar with yet.

TOOLCHAINS & PROCESSES

Beyond working on the code, there will also be an established protocol for software development.

More established projects will have either training or comprehensive documentation.

→ If the documentation isn't available, then you can take the initiative and try to write it.

NEXT CLASS

Query Compilation