Efficient Use of the Query Optimizer for Automated Physical Design

Stratos Papadomanolakis

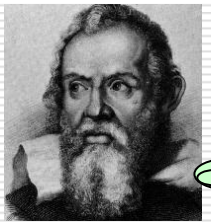
Debabrata Dash

Anastasia Ailamaki

ORACLE

Databases @ Carnegie Mellon

Database Physical Design



select clustID, count(*)
from Galaxies
group by clustID

ClustID		

Galaxies 10TB

Astronomy database

Design Algorithm

Automated database design tool

Cost Model

Partitioning

Materialized Views

Index Selection

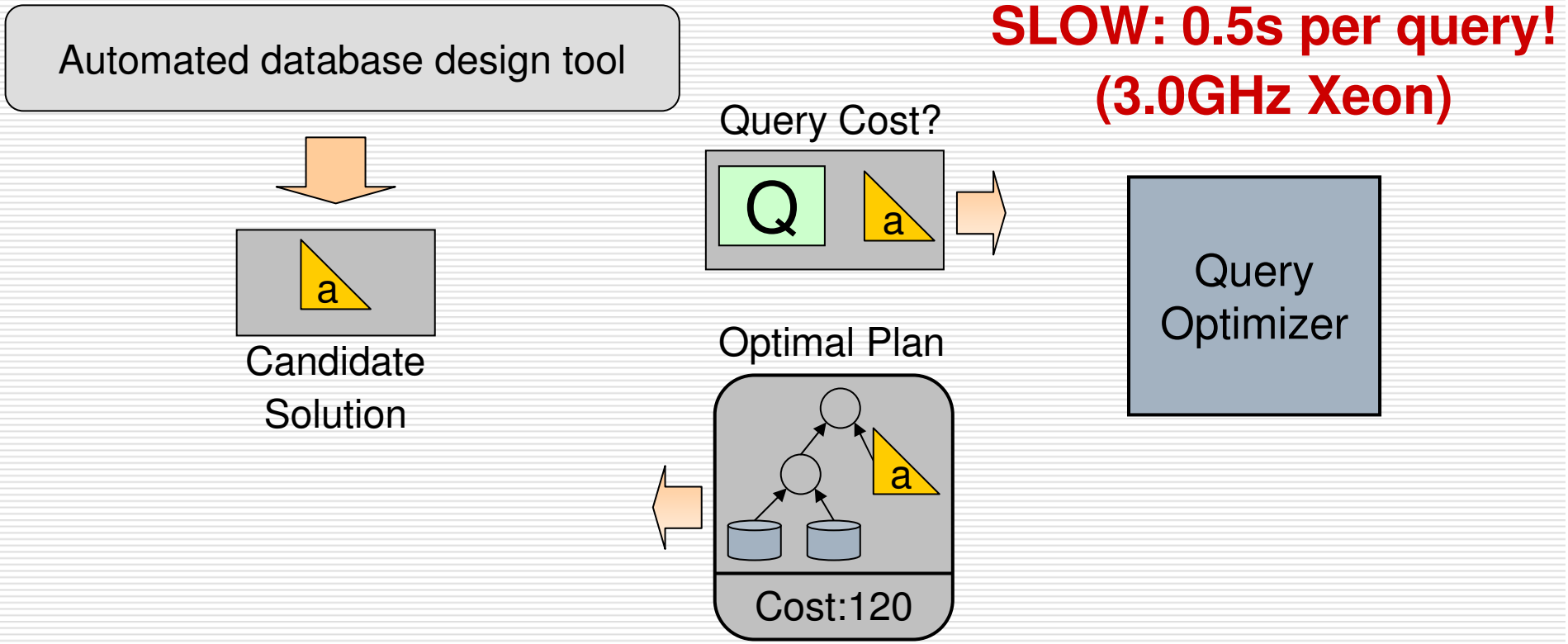
ClustID		

ClustID	Counts

ClustID		

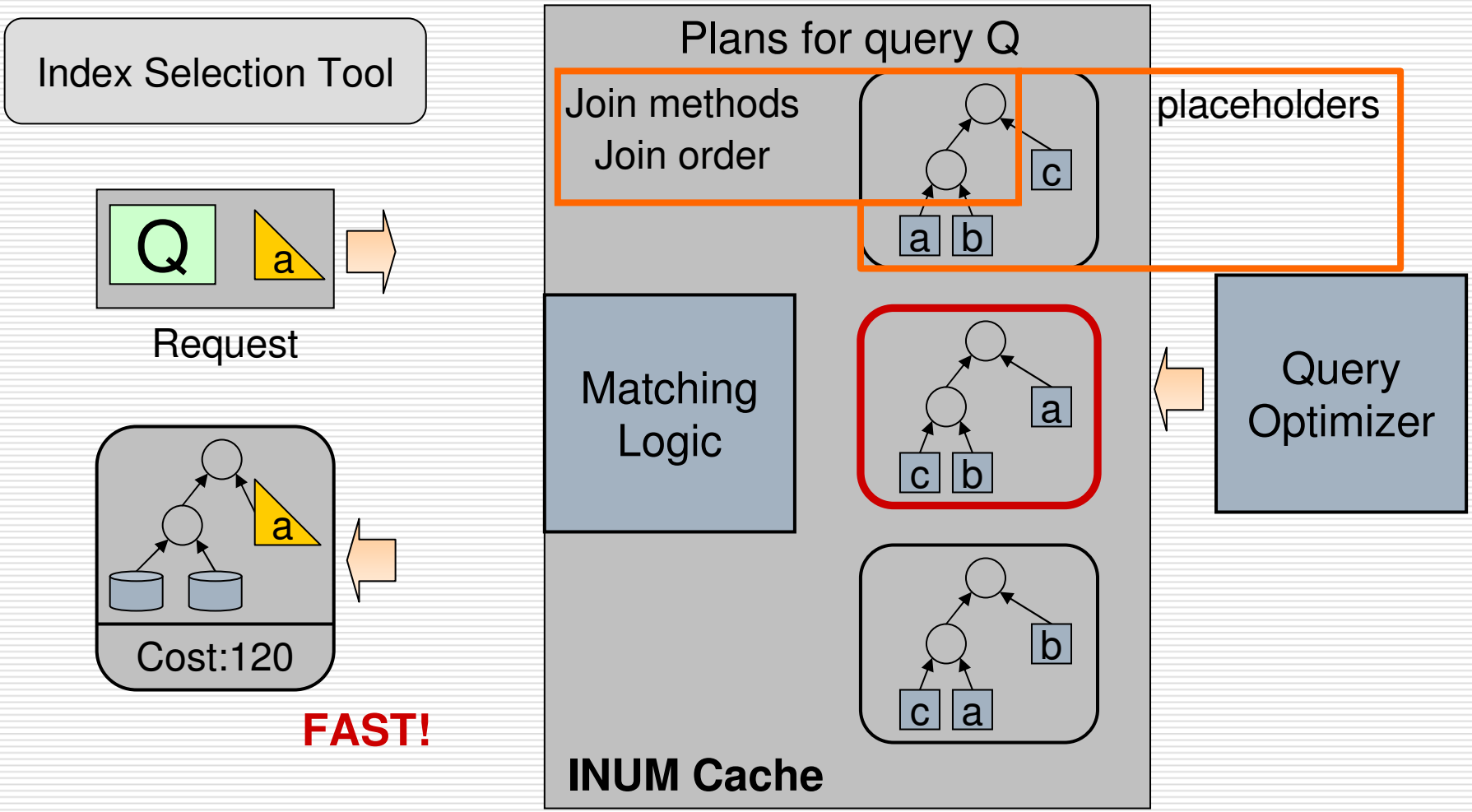
Cost model: Must be fast and accurate!

Dependence on the Query Optimizer



- BIG problem
  - Large workloads take hours
  - Limited search spaces
- BUT: We need the optimizer!

The INdex Usage Model (INUM)



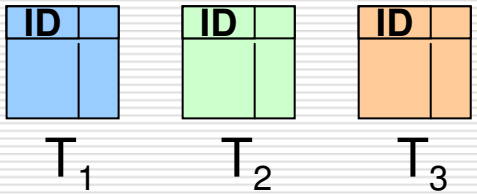
1000x faster than optimizer, same result!

Scalable algorithms, better solutions!

Outline

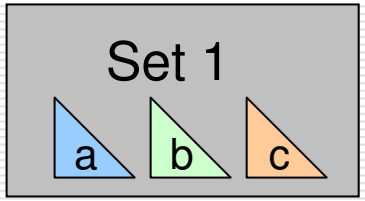
- Motivation
- A Simple Design Scenario
- The Index Usage Model
- Experimental Results
- Conclusion

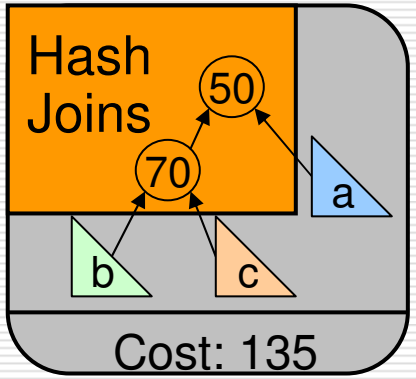
Simple example



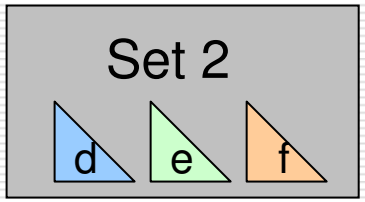
Q: Join T_1, T_2, T_3 on ID
(and other predicates)

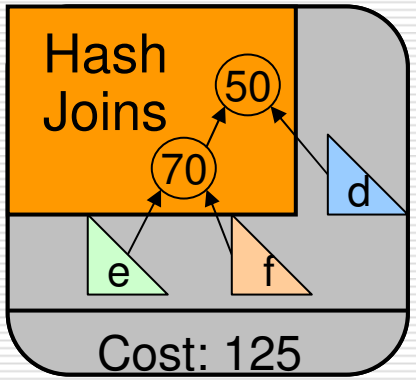
Index sets to evaluate
no ID columns





- No IDs
- No plan alternatives
- One call!

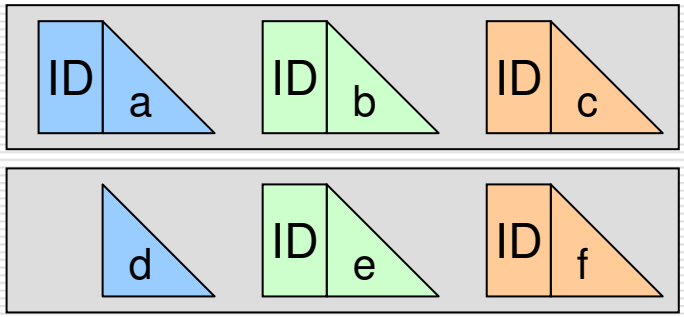




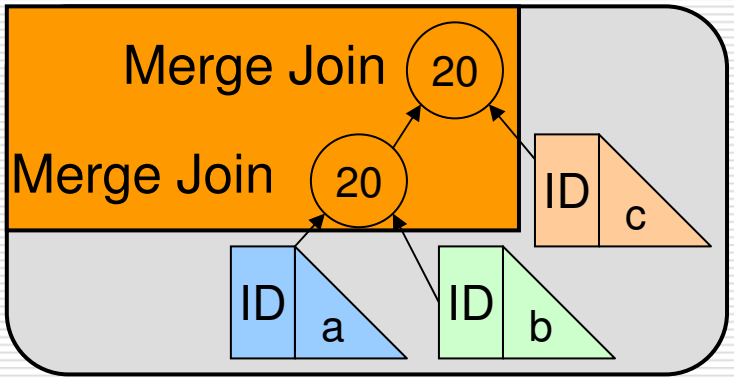
- Join columns
- Interesting orders

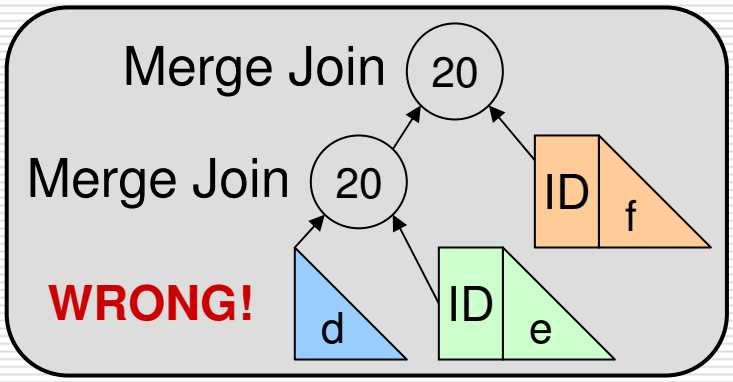
Challenge: Interesting orders

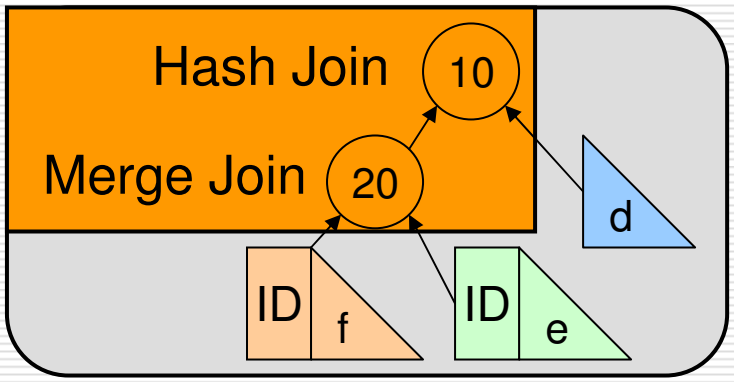
Q: Join T_1, T_2, T_3 on ID



No ID Col

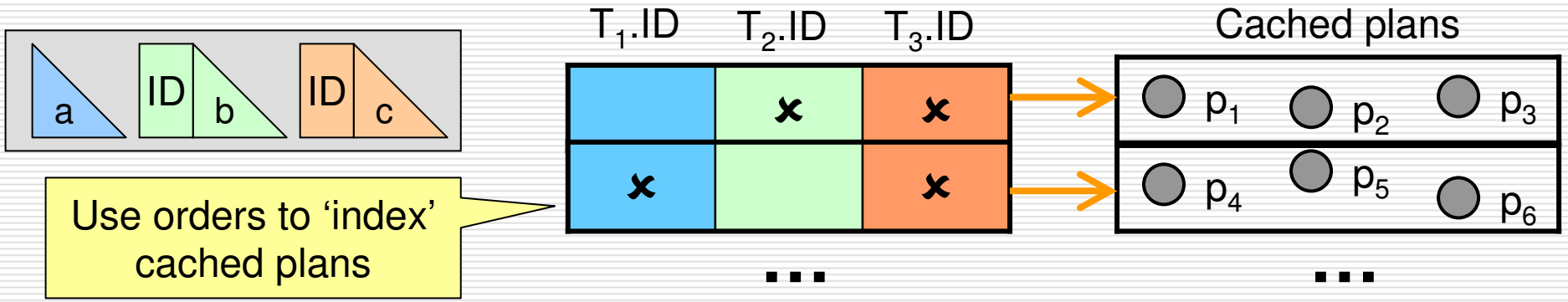






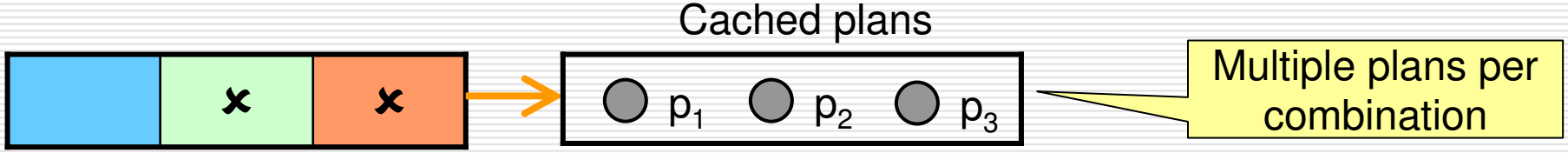
- Multiple plans!
- Plans to cache? When to use each plan?

Dealing with interesting orders



- Step 1: Merge & Hash join plans (MHJ plans)

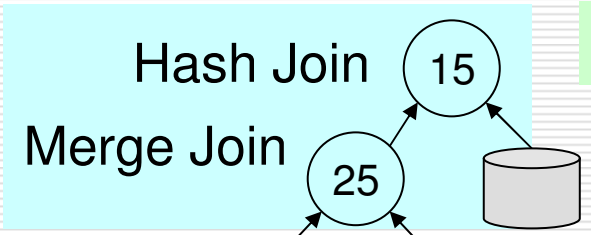
- Step 2: Nested loop join plans (NLJ plans)



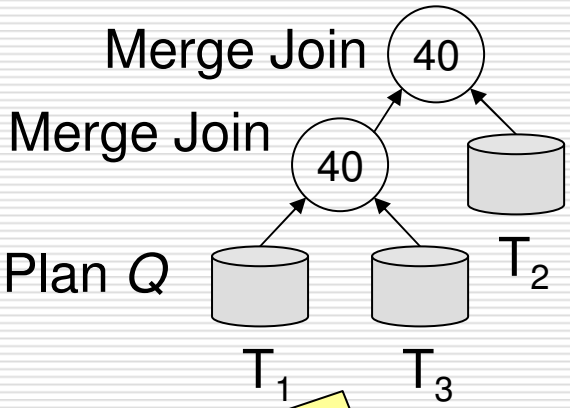
Outline

- Motivation
- A Simple Design Scenario
- The Index Usage Model
  - Merge & hash join plans (MHJ)
  - Nested loop join plans (NLJ)
- Experimental Results
- Conclusion

Merge & Hash joins (MHJ Plans)



Q: Join T_1, T_2, T_3 on ID

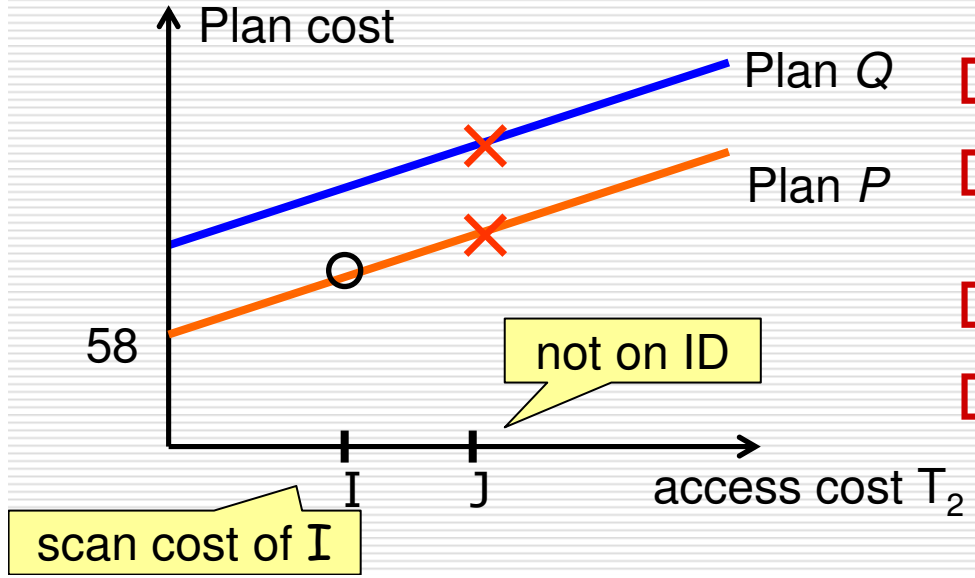


cost: 40 + T_1 (10) + T_2 (?) + T_3 (8)

internal cost

index access costs

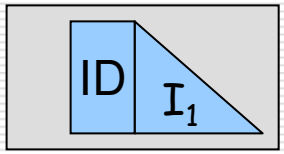
Structure of cost equation is the same!

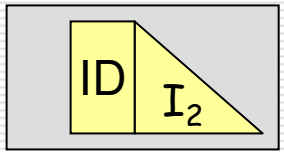


- Linearity, parallel surfaces
- Works for more dimensions
- Different interesting orders
- Need multiple graphs!

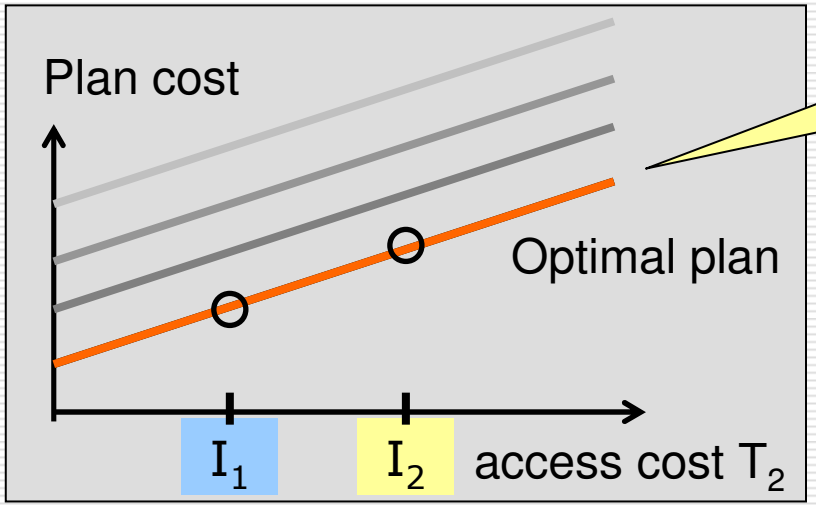
Exploiting linearity

Candidates for T_2



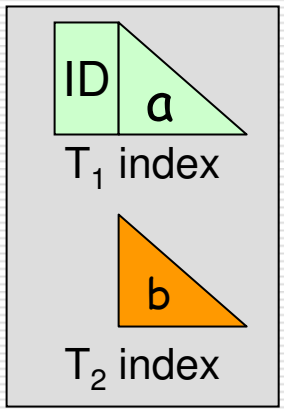


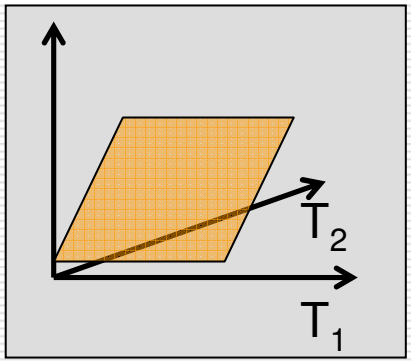
Q: Join T_1, T_2, T_3 on ID

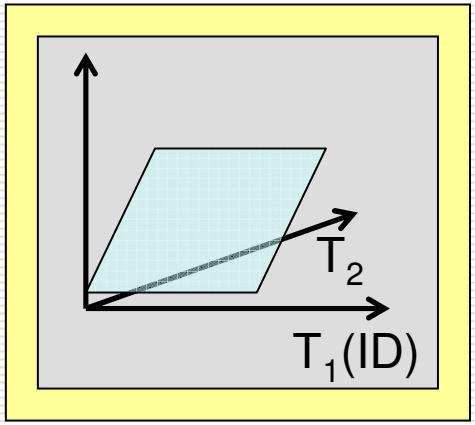


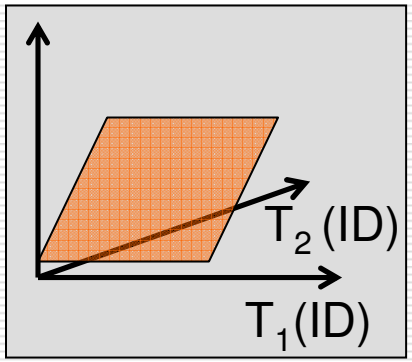
One optimizer call!

Candidates for T_1, T_2









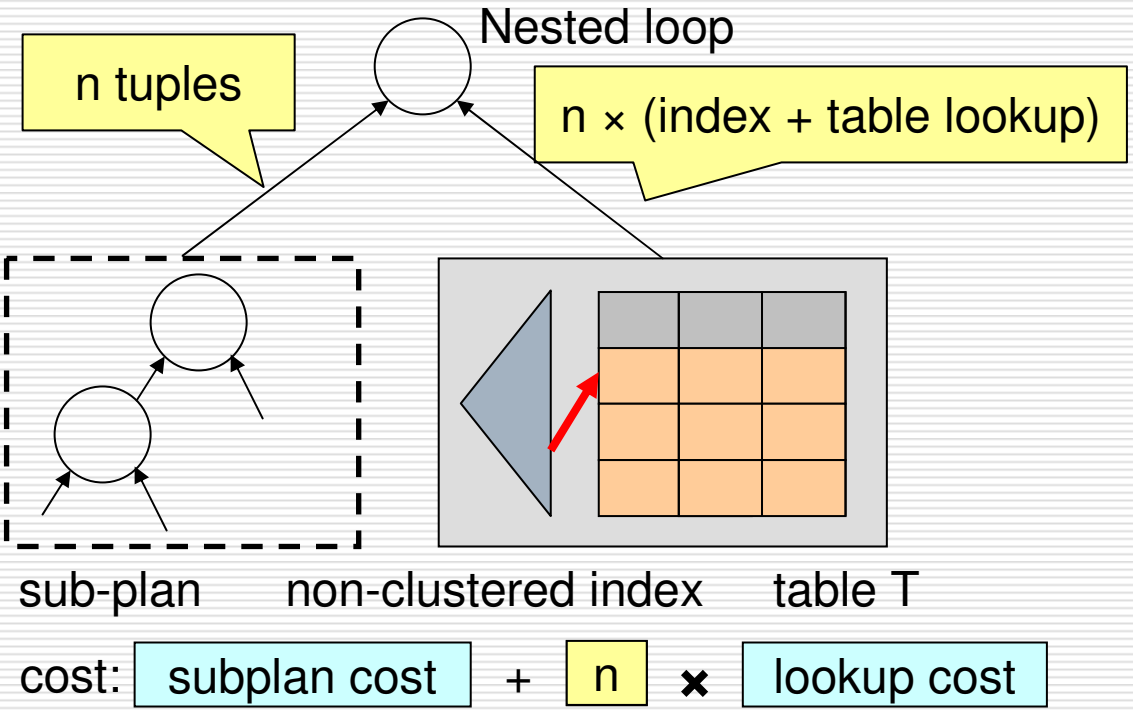
☐ One call per interesting order combination

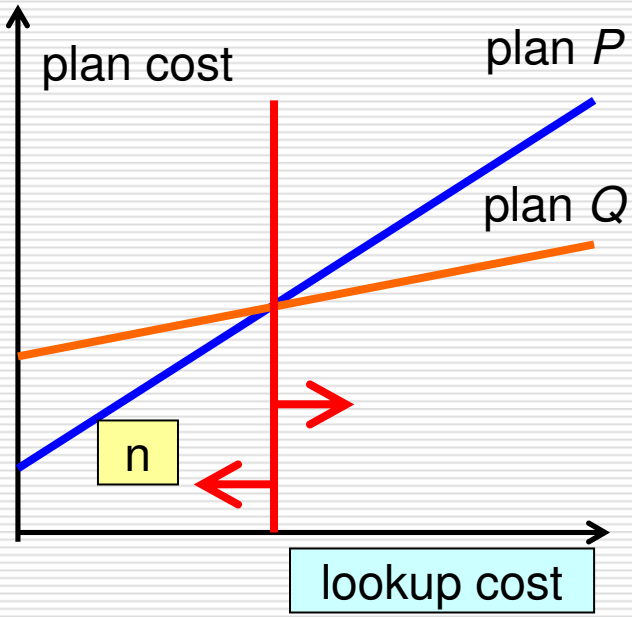
Outline

- Motivation
- A Simple Design Scenario
- The Index Usage Model
  - Merge & Hash join plans (MHJ plans)
  - Nested loop join plans (NLJ plans)
- Experimental Results
- Conclusion

Plans with nested loop joins

- Or, what if linearity does not hold?





- Different parameters
- Intersecting plan surfaces
- Must find and cache multiple plans

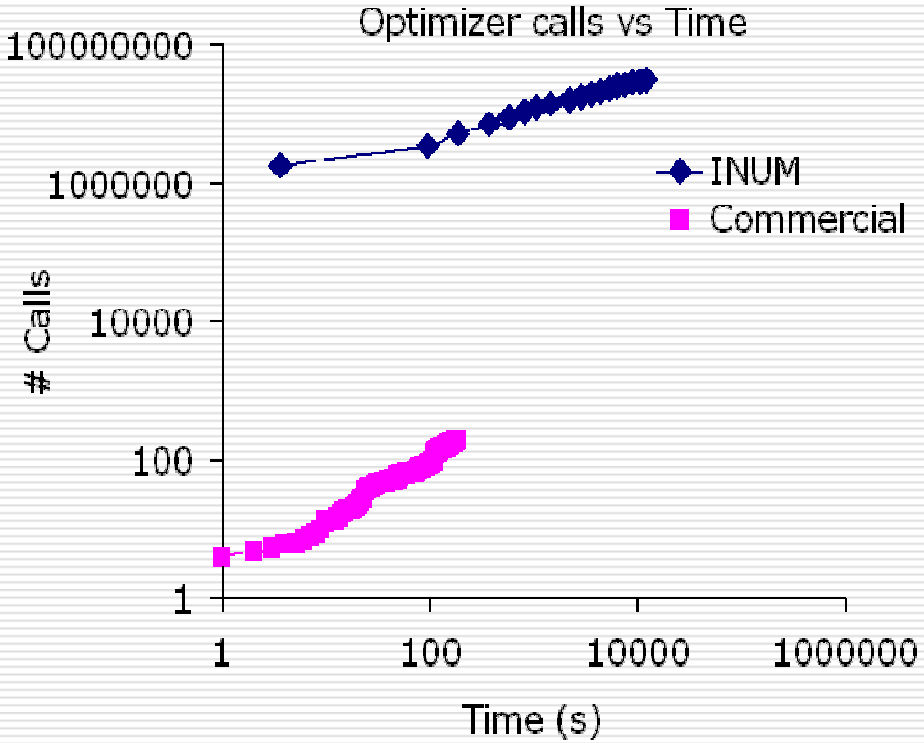
Outline

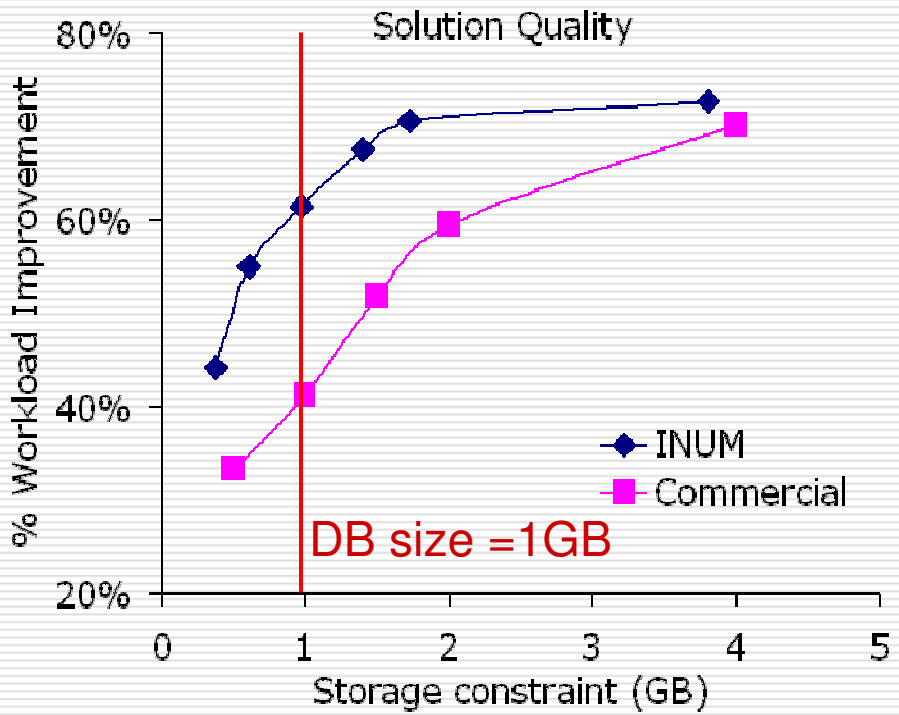
- Motivation
- A Simple Design Scenario
- The Index Usage Model
  - Dealing with multiple interesting orders
  - Nested loop joins
- Experimental Results
- Conclusion

Experimental Setup

- Implementation
  - INUM Cache interfacing to a commercial DBMS
  - eINUM: Simple index selection tool
- Performance test
  - 15 TPC-H queries (1GB), 100K "candidates"
  - IMPOSSIBLE with existing techniques
  - Compare to a commercial tool

Experimental results





31M "optimizer calls" in ~ 3.5 hrs

1000 times faster than optimizer

Time to "fill" INUM Cache: 21min

Found better solutions

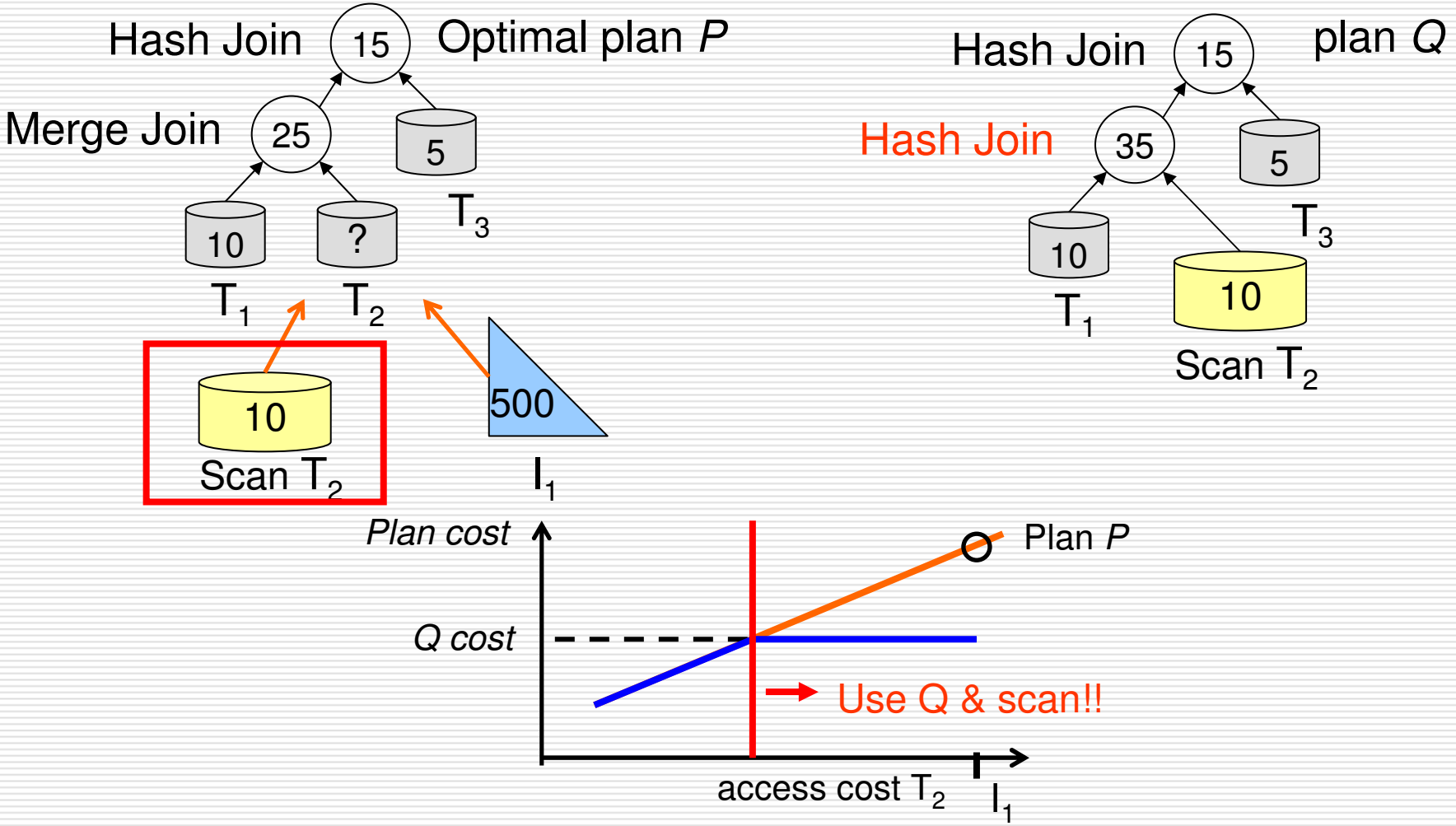
Indexes "missed" by commercial

Conclusion

- Query optimizer in automated DB design
  - Time overhead!
  - Need for additional pruning heuristics

- Index Usage Model
  - Reuse optimizer computation
  - Optimal plan without optimizer!
  - 1000x faster estimation, accurate results ✓
  - Fewer constraints ⇨ better quality ✓

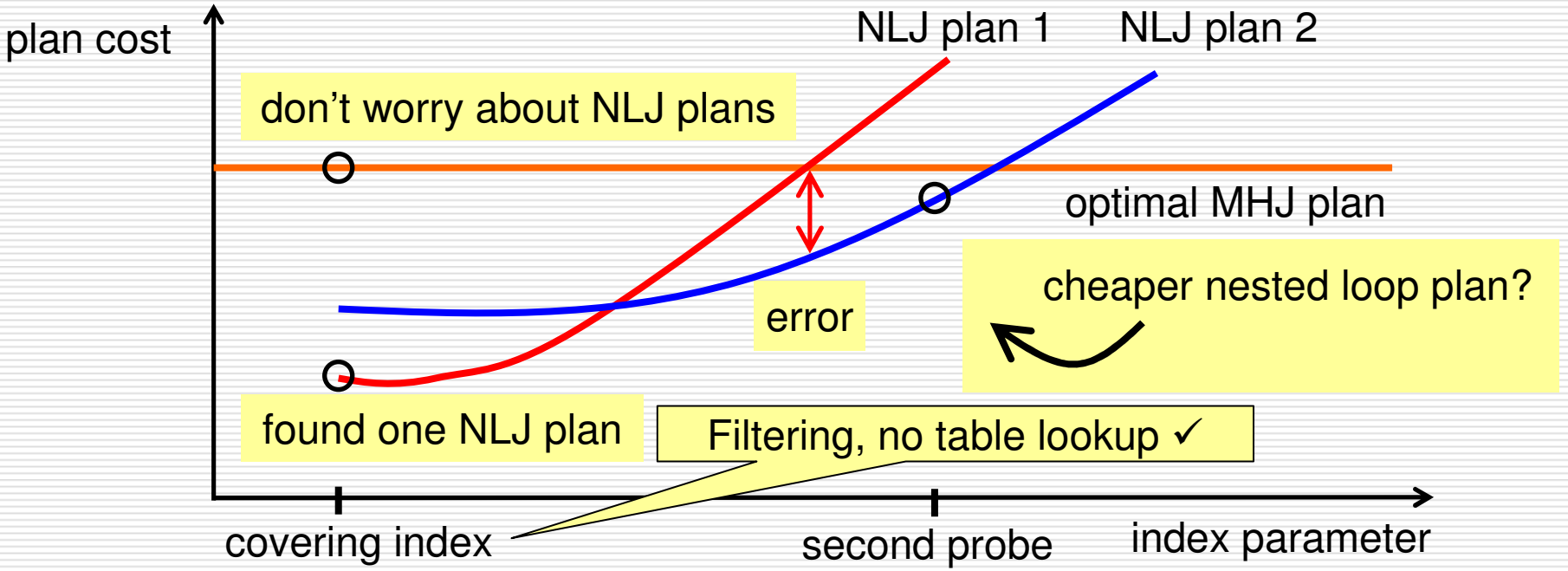
Accounting for table scans



- Table scans ⇨ Two-part cost surfaces

Dealing with non-linear cost models

- Approach 1: Analyze the cost model
  - Access to the optimizer?
- Approach 2: "Probe" the model



- Easy to find at least one NLJ plan!