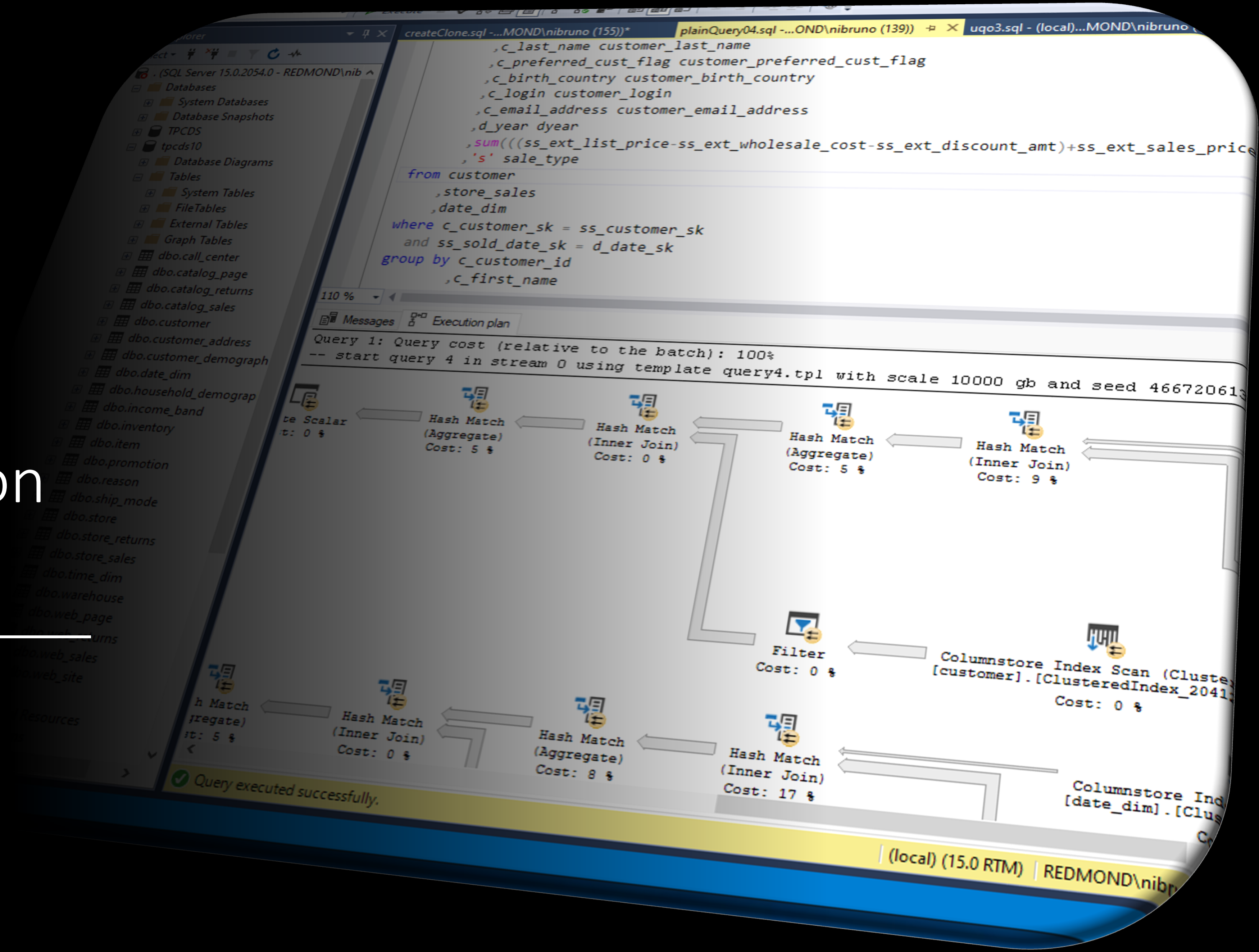
The Cascades Framework for Query Optimization at Microsoft

Nico Bruno

Cesar Galindo-Legaria



```
       ,c_last_name customer_last_name
       ,c_preferred_cust_flag customer_preferred_cust_flag
       ,c_birth_country customer_birth_country
       ,c_login customer_login
       ,c_email_address customer_email_address
       ,d_year dyear
       ,sum(((ss_ext_list_price-ss_ext_wholesale_cost-ss_ext_discount_amt)+ss_ext_sales_pric
       ,'s' sale_type
 from customer
     ,store_sales
     ,date_dim
 where c_customer_sk = ss_customer_sk
   and ss_sold_date_sk = d_date_sk
 group by c_customer_id
         ,c_first_name
```

```
Query 1: Query cost (relative to the batch): 100%
-- start query 4 in stream 0 using template query4.tpl with scale 10000 gb and seed 46672061
```

Query executed successfully.

The Cascades Framework

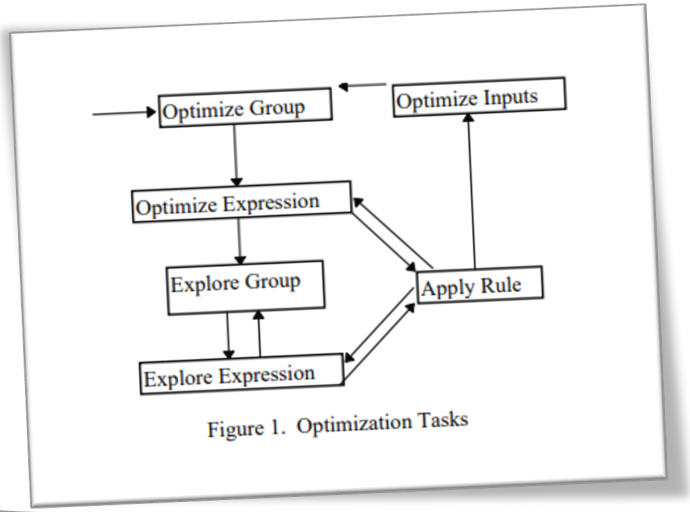
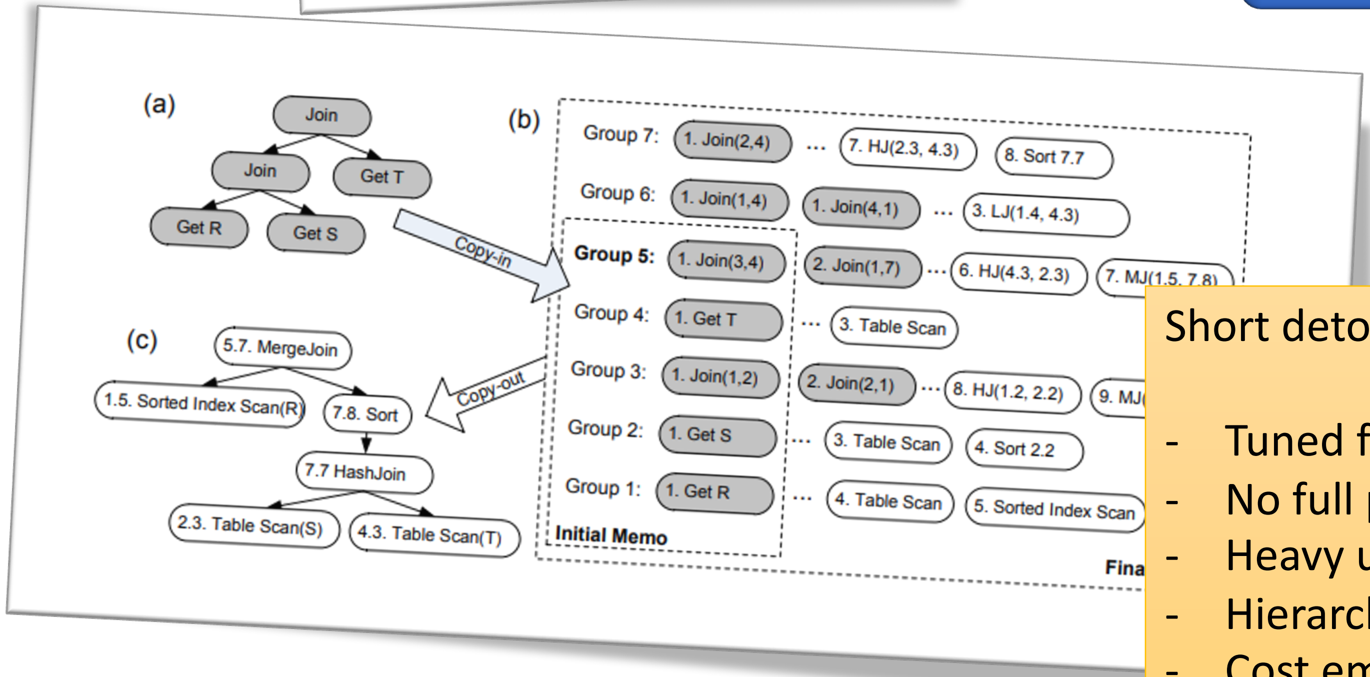


Figure 1. Optimization Tasks



Algebraic representation of logical/physical query fragments

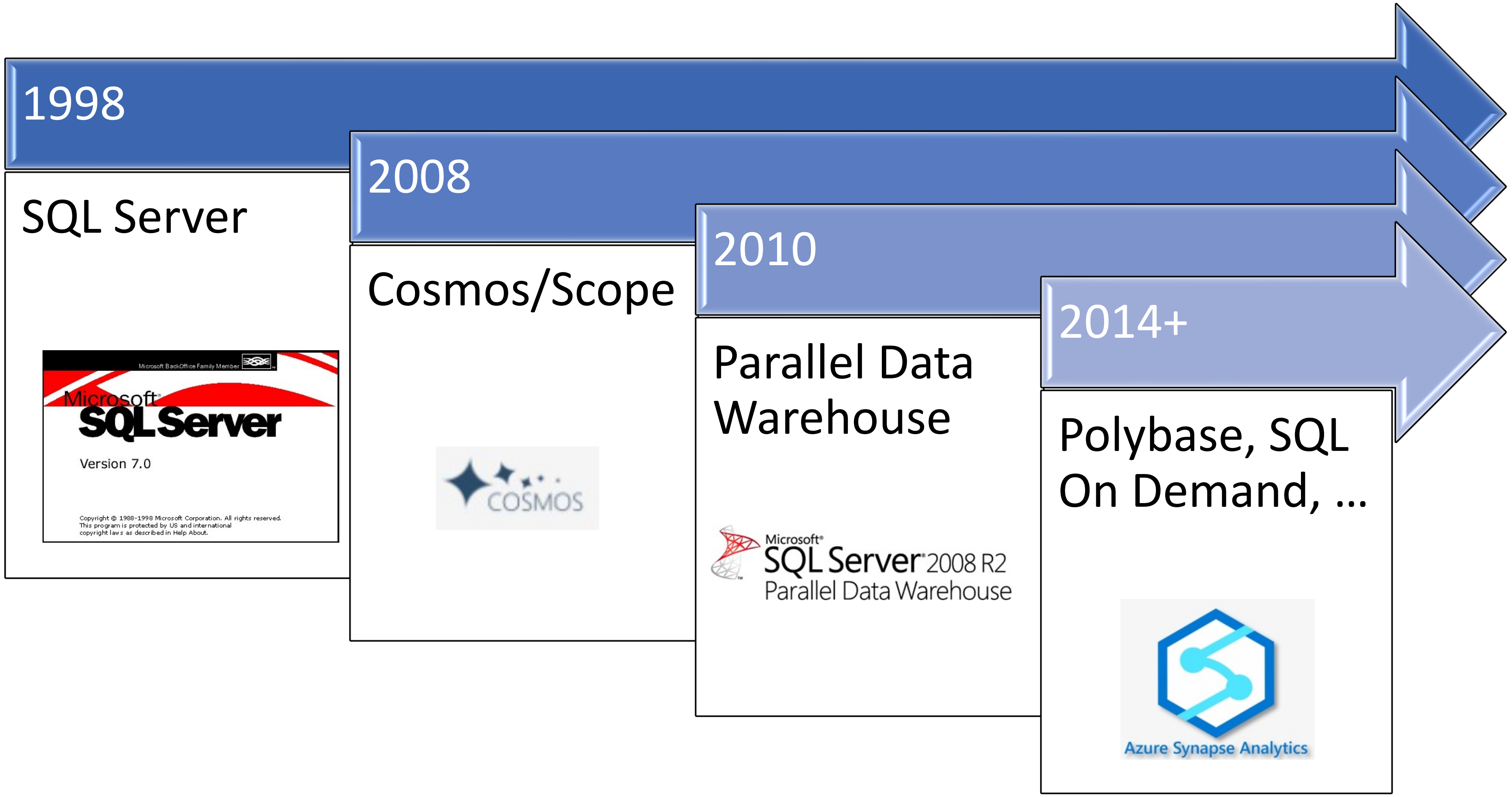
Composable rules (exploration, implementation, enforcers)

Integrated and uniform optimization

Short detour: CascadEGGs?

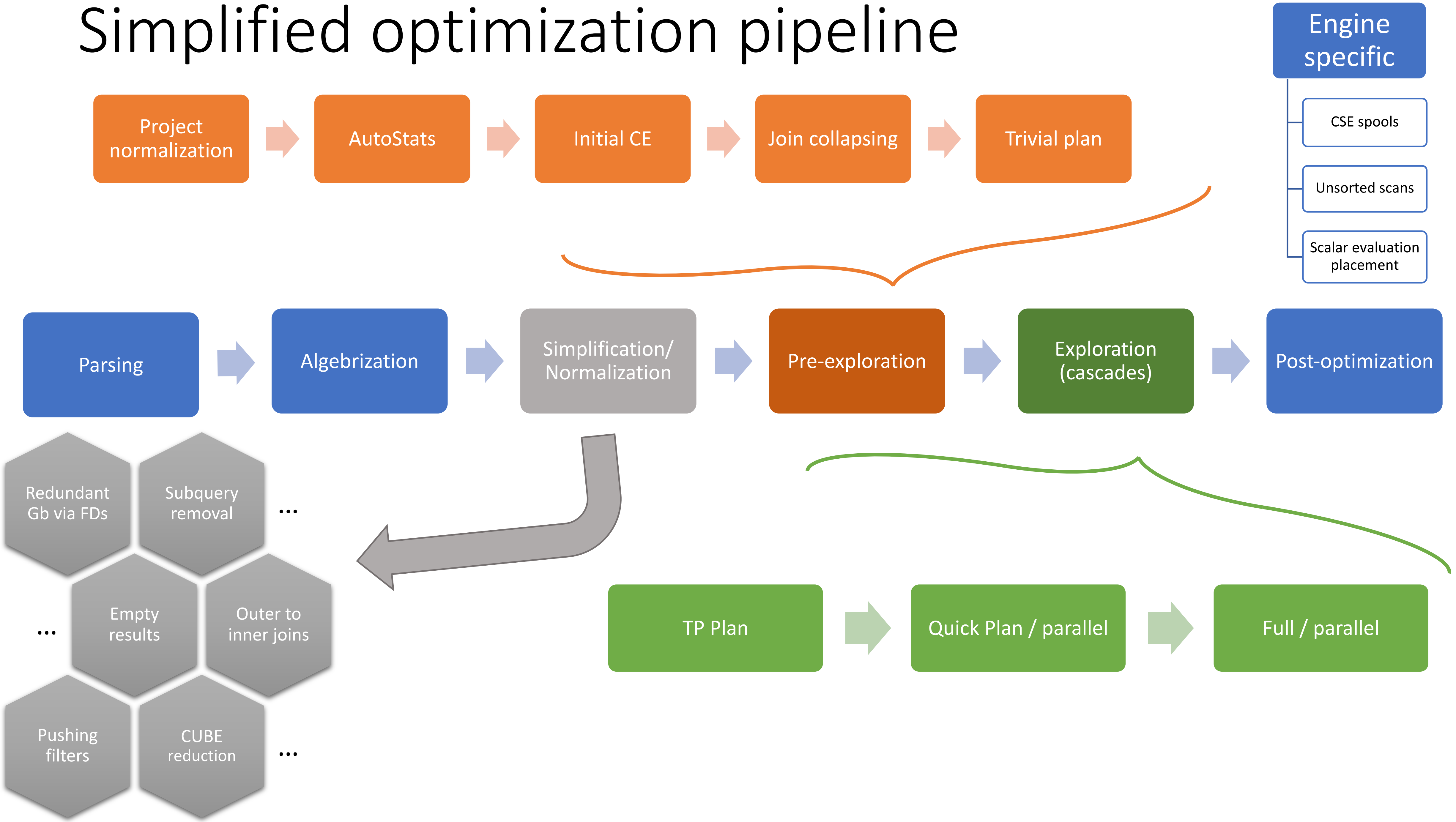
- Tuned for a specific domain (rule priorities, no cycles, etc.)
- No full propagation of congruences
- Heavy use of properties (derived & required)
- Hierarchical traversal of MEMO for rule binding and application
- Cost embedded in search (properties/pruning)

Cascades at Microsoft



Sql Server (2001) Scope (VLDBJ'12), PDW (Sigmod'12), Polybase (Sigmod'13)

Simplified optimization pipeline



Rules & Properties

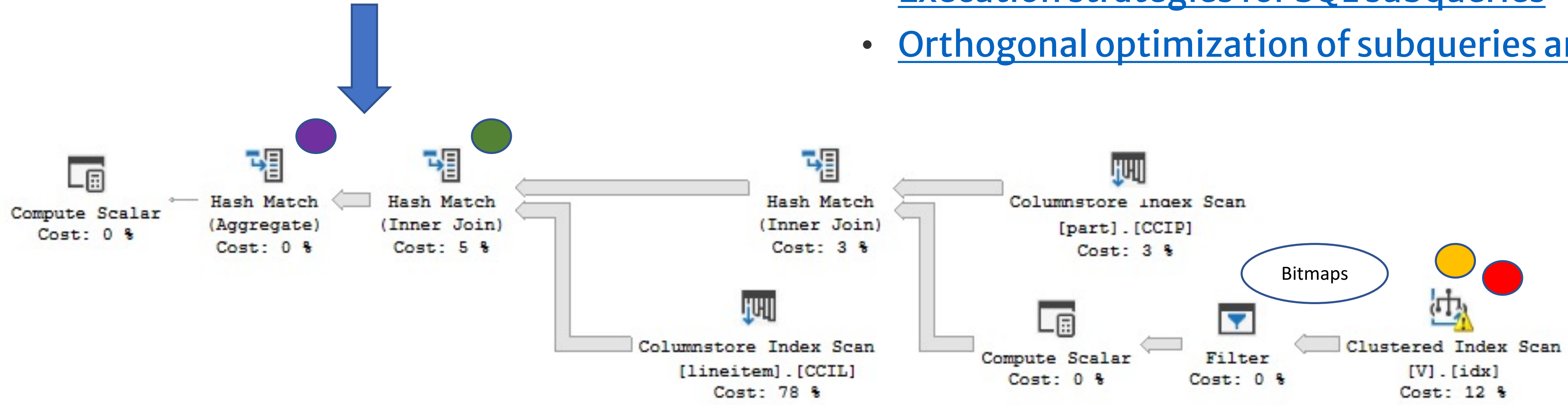
```
select sum(l_extendedprice) / 7.0 as avg_yearly
from lineitem join part on p_partkey = l_partkey
where
    p_brand = 'Brand#12'
    and l_quantity < (select 0.2 * avg(l_quantity)
                      from lineitem
                      where l_partkey = p_partkey)
```

```
create view V with schemabinding as
select l_partkey, sum(l_quantity) sc, count_big(*) cb
from dbo.lineitem
group by l_partkey
```

400+ rules

- 🟢 Join reordering
- Outer joins
- 🟡 Subqueries
- 🟣 Aggregation
- Union
- Stars and snowflakes
- Join elimination
- Empty table simplification
- 🔴 Materialized views
- Index plans
- Large IN lists
- Update plans
- Halloween protection
- Partitioned tables
- Parallelism
- Remote queries
- ...

- Execution strategies for SQL subqueries
- Orthogonal optimization of subqueries and aggregation



Statistics

Taxonomy

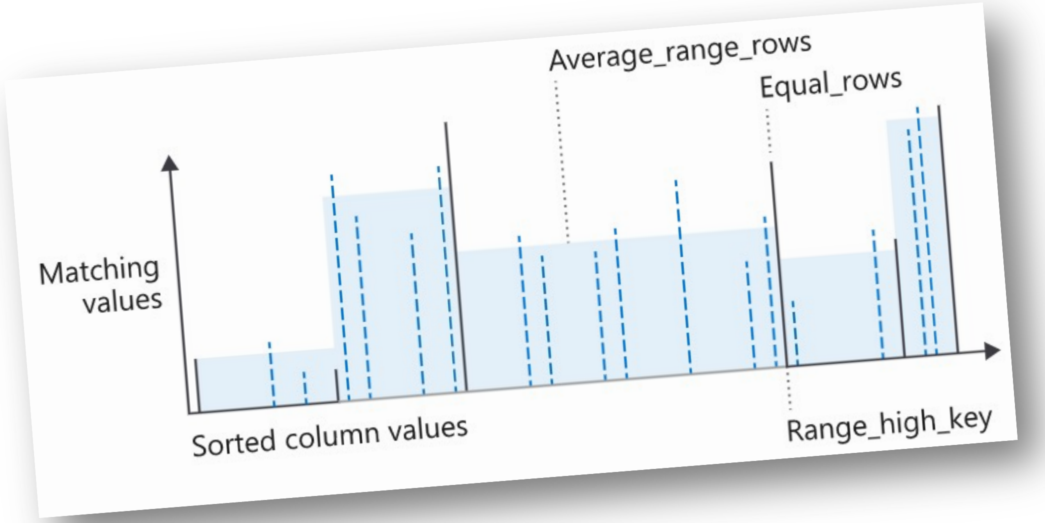
- Single-column ‘MaxDiff’ histograms
- Multi-column density information
- Average column lengths
- Tries
- HLL / Heavy Hitter sketches (DW / Partitioned tables)
- Skew (Cosmos)

Data sources

- Base tables (including computed columns)
- Filtered indexes
- Materialized views

Create / Update mechanics

- Creation: manual, implicit, automatic
- Update: manual, automatic with mod counts
- Block-level sampling (optional cross-validation)



Cardinality Estimation

Algebra of histograms

- Propagation of statistics through operators
- From arithmetic (`WHERE a+2>5`) to aggregation (`HAVING SUM(a)>10`)

How do we estimate cardinality values?

- `WHERE a=10 AND b=12 ->` H(a)&H(b)? MCD(a, b)? H(a | b=12)?
  Depends on skew, correlation, etc.
- QOE (Quality Of Estimation) to rank alternatives
- Rewritten cardinality estimation framework
  - Holistic *calculators* for estimating query fragments
  - Model assumptions overridable via hints (e.g., `ASSUME_JOIN_PREDICATE_DEPENDS_ON_FILTERS`)

Other topics

- Autoparameterization and parameter sniffing
- CE feedback / learned cardinalities
- Memory grants are based on cardinality estimation

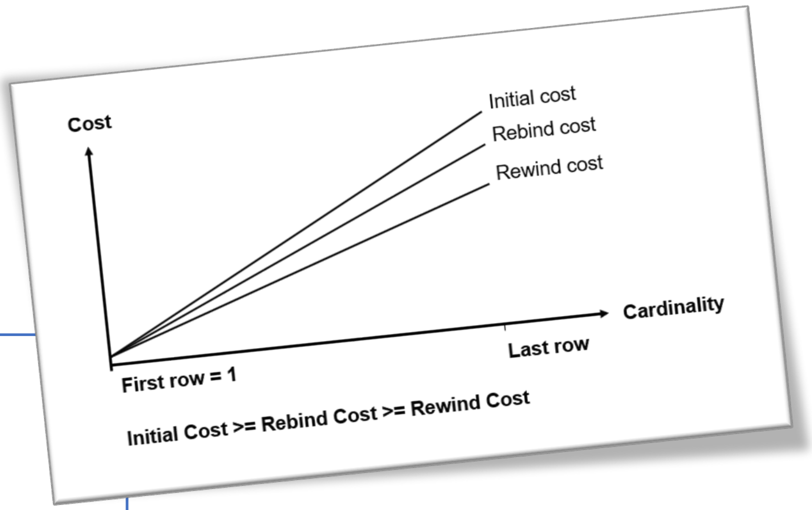
Costing

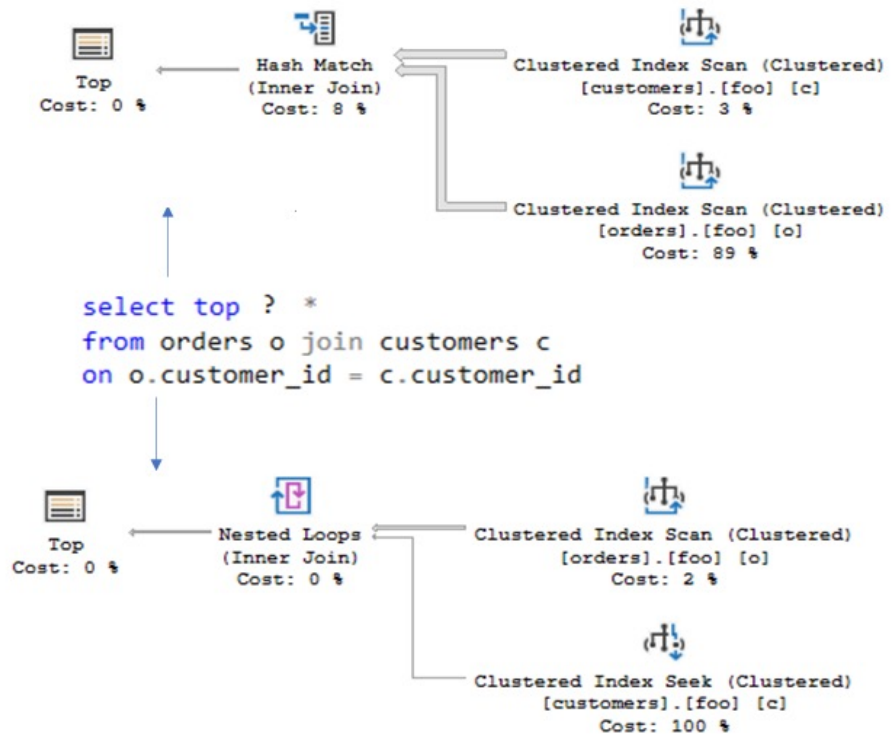
Bottom-up calculation…

- CPU (e.g., filters) and I/O (e.g., spilling aggs)
- Information: CE, DV, outliers, row sizes, DOP, memory, sorted-ness, etc.
- 3 cost lines: Initial / rewind / rebind

… with top-down context

- Row goals
- Bitmap filters
- Estimated rewinds/rebinds



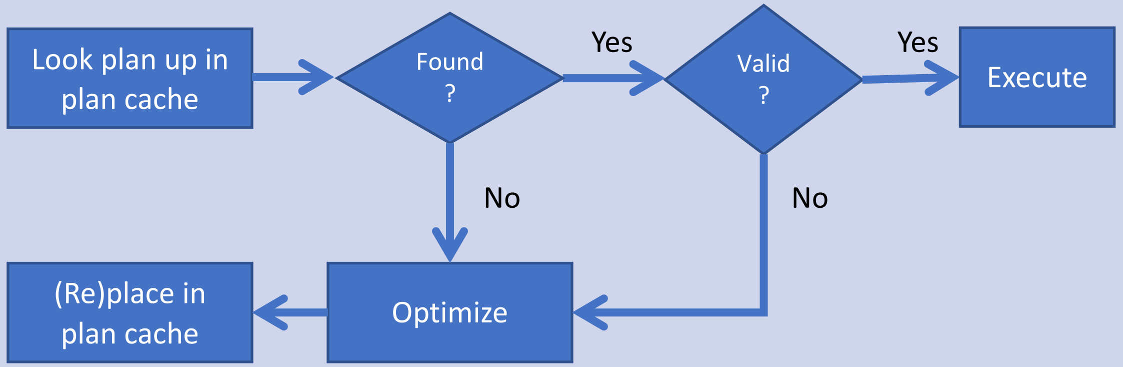


```
select top ? *
from orders o join customers c
on o.customer_id = c.customer_id
```

Optimization Performance

Many design choices in Cascades assume full plan space search!

Plan caches



Trivial plans

Optimization stages and timeouts

Subset of transformations enabled in each stage

Best cost at the end of a stage determines next steps

Memo seeding (e.g., Nary-join heuristic reordering)

Various approaches to gradual optimization (e.g., temperature-based)

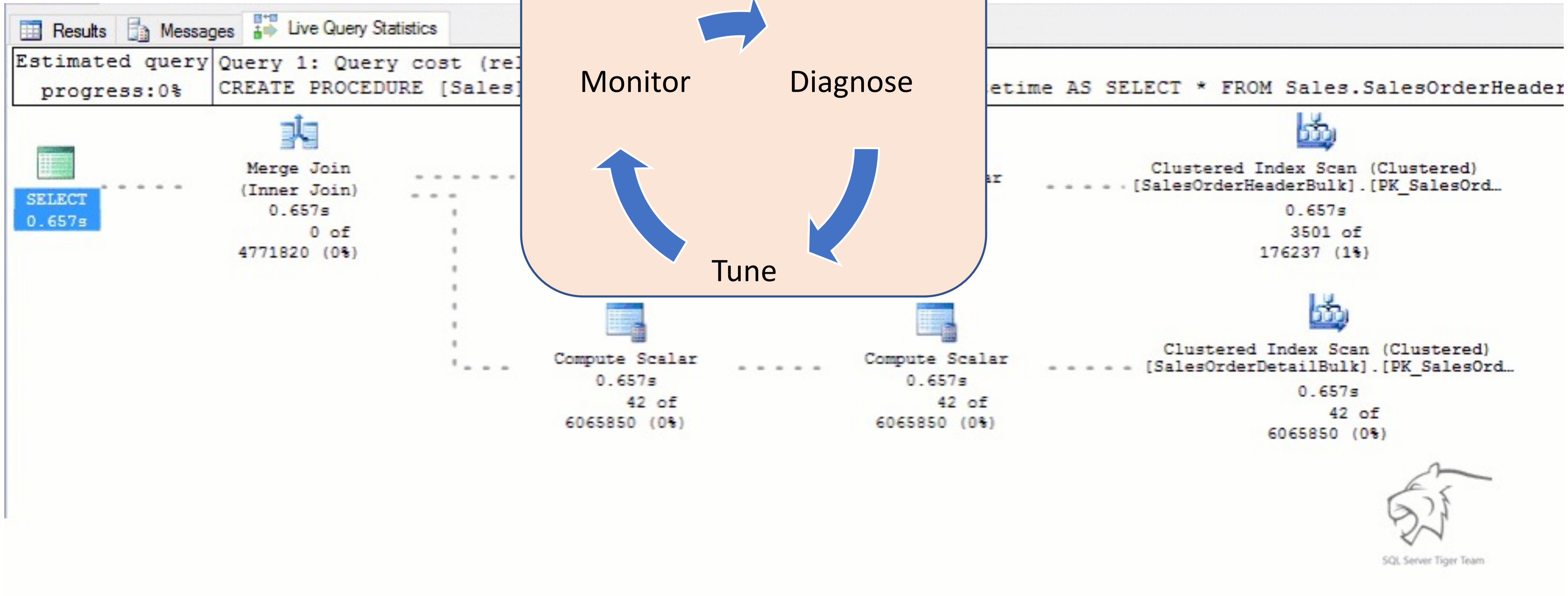
Supportability

Inputs to QO

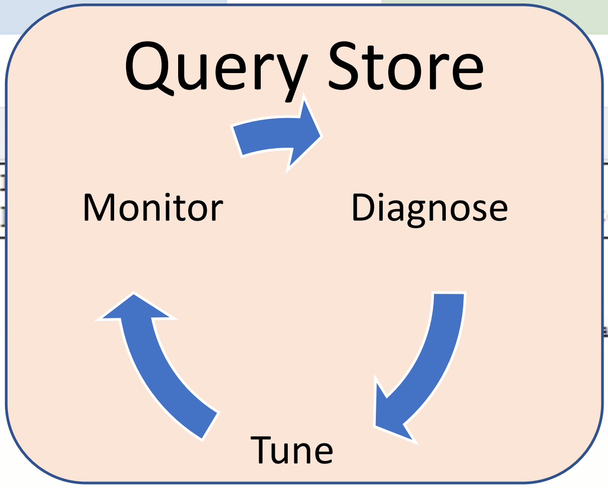
- Table, join, query-level hints
- Plan hints: *Find* plan in the Memo!
- Plan guides: Transparently hint queries

Outputs from QO

- Graphical showplan
- Execution traces, including live plans
- DMVs (e.g., dm_exec_query_optimizer_info)



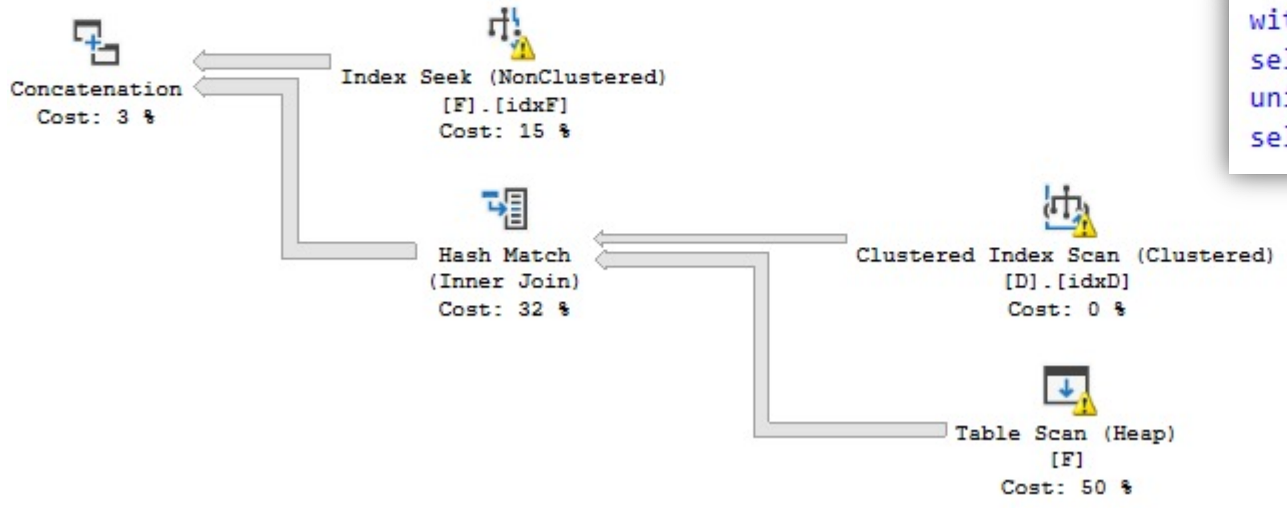
```
Results   Messages   Live Query Statistics

Estimated query   Query 1: Query cost (rel
  progress:0%     CREATE PROCEDURE [Sales]          etime AS SELECT * FROM Sales.SalesOrderHeader

SELECT     Merge Join                                        Clustered Index Scan (Clustered)
0.657s     (Inner Join)                                 ar   [SalesOrderHeaderBulk].[PK_SalesOrd…
           0.657s                                            0.657s
           0 of                                              3501 of
           4771820 (0%)                                      176237 (1%)

                       Compute Scalar      Compute Scalar    Clustered Index Scan (Clustered)
                       0.657s              0.657s            [SalesOrderDetailBulk].[PK_SalesOrd…
                       42 of               42 of             0.657s
                       6065850 (0%)        6065850 (0%)      42 of
                                                             6065850 (0%)

                                                             SQL Server Tiger Team
```

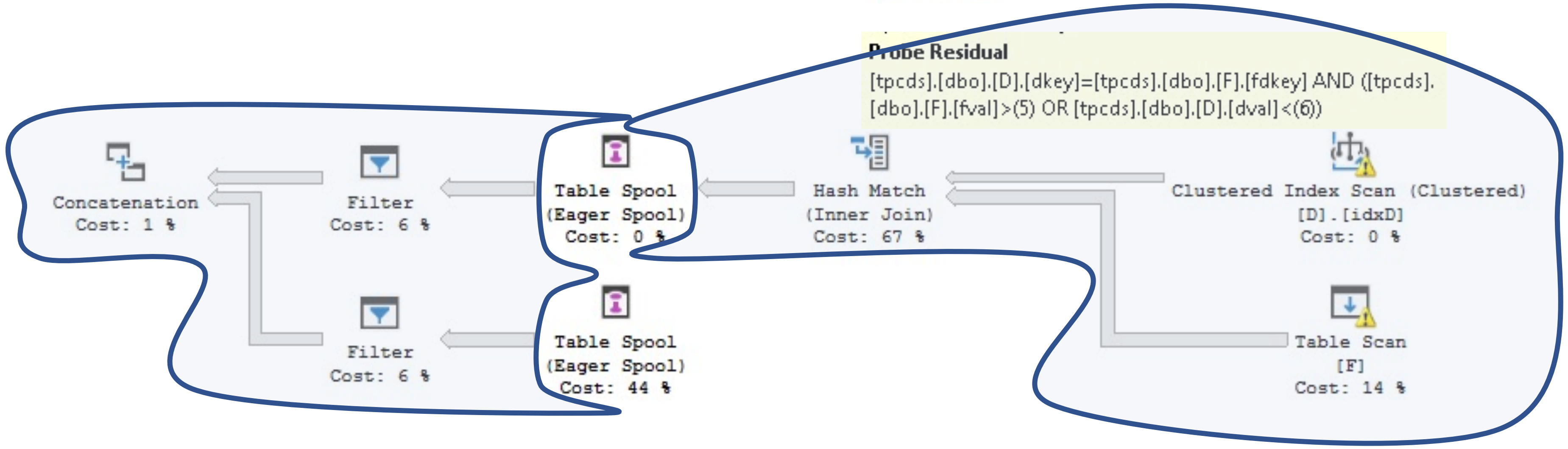


Common subexpressions

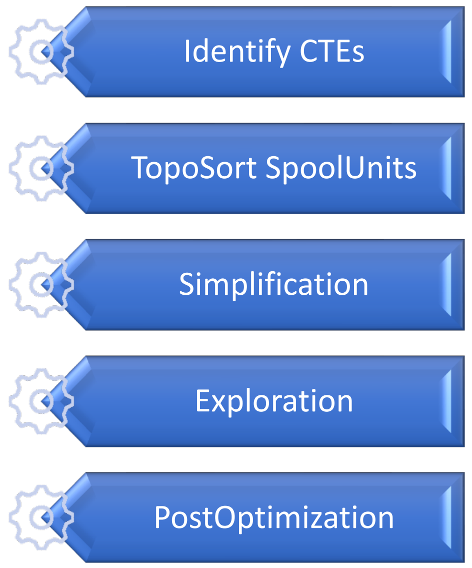
```
create table D(dkey int not null, dval int)
create unique clustered index idxD on D(dkey)

create table F(fkey int, fval int, fdkey int not null references D(dkey))
create index idxF on F(fval)
```

```
with cte (fval, dval) as (select fval, dval from F join D on fdkey = dkey)
select fval from cte where fval > 5
union all
select dval from cte where dval < 6
```





Probe Residual
[tpcds].[dbo].[D].[dkey]=[tpcds].[dbo].[F].[fdkey] AND ([tpcds].[dbo].[F].[fval]>(5) OR [tpcds].[dbo].[D].[dval]<(6))



QO Testing

Testing SQL Server's Query Optimizer: Challenges, Techniques and Experiences, DEBull'08

Dimensions

- Correctness: do we get the same results (modulo non-determinism)?
- Performance: do we get the same performance?
- QO scorecards: other metrics (QO memory, CE errors, plan sizes, etc.)

Data and query sources

- **Massive** test collateral
- Large-scale stochastic testing / Fuzzying
- Benchmarks and performance baselines (microbenchmarks)
- Customer playbacks (e.g., Cosmos Playback)
- MinRepros (Minimizing database repros using language grammars)

Sources of truth

- Different DB system (e.g., Cosmos vs SqlServer)
- Different DB release (e.g., SqlServer'19 vs SqlServer'17)
- Same DB! (Counting, enumerating, and sampling of execution plans in a cost-based QO)

QO as a Component

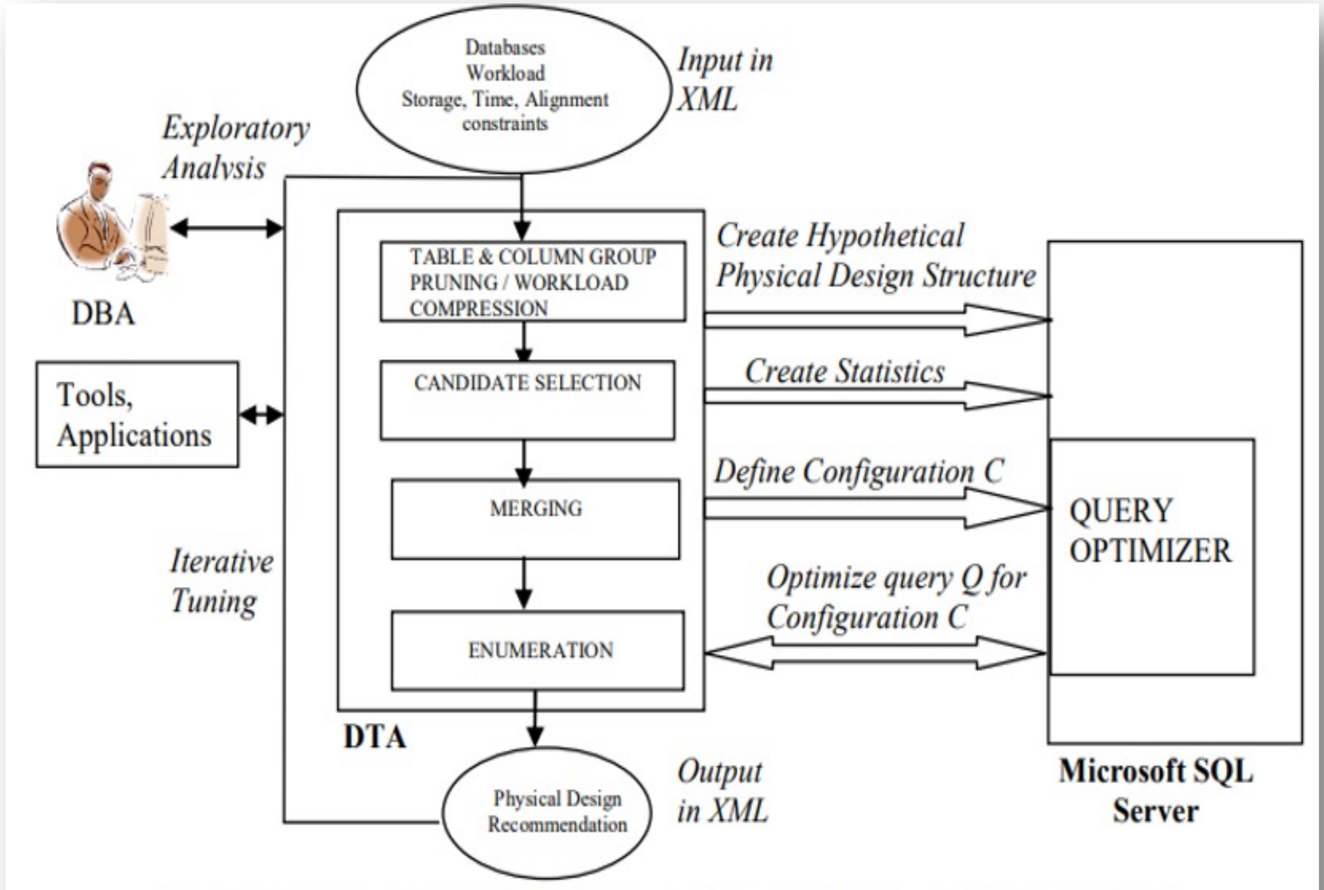


Figure 1: Overview of the Database Engine Tuning Advisor (DTA).

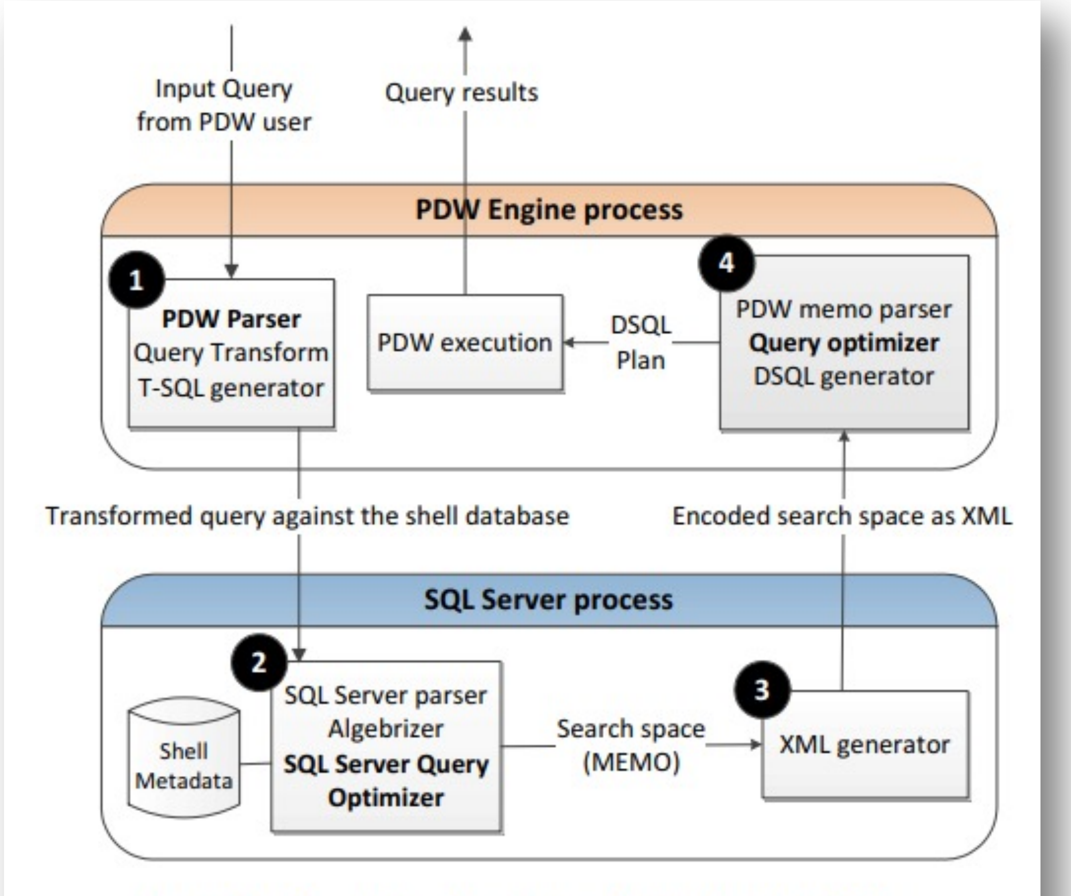


Figure 2: Overview of query optimization in PDW.

AutoAdmin "what-if" index analysis utility
SIGMOD Record'98

Query optimization in Microsoft SQL Server PDW
SIGMOD'12

Questions?