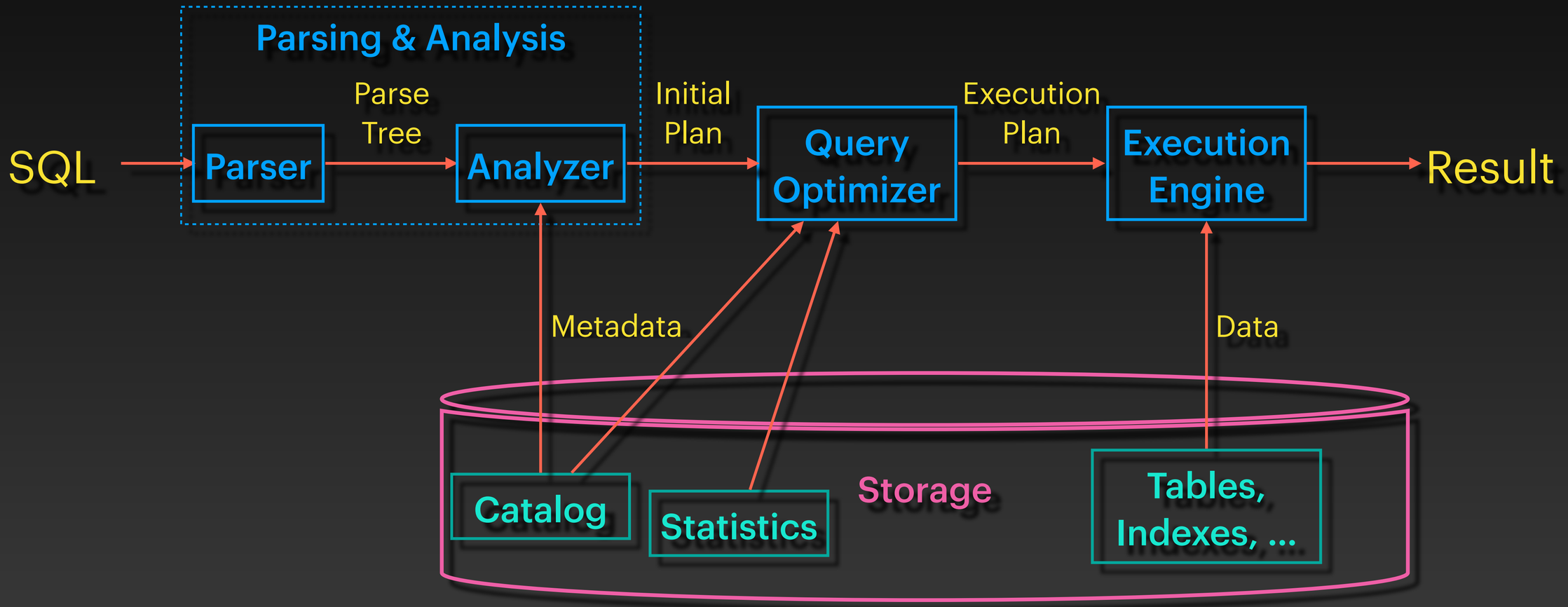
TECH
VAULT

SQL QUERY OPTIMIZER

AMR ELHELW

Query Engine



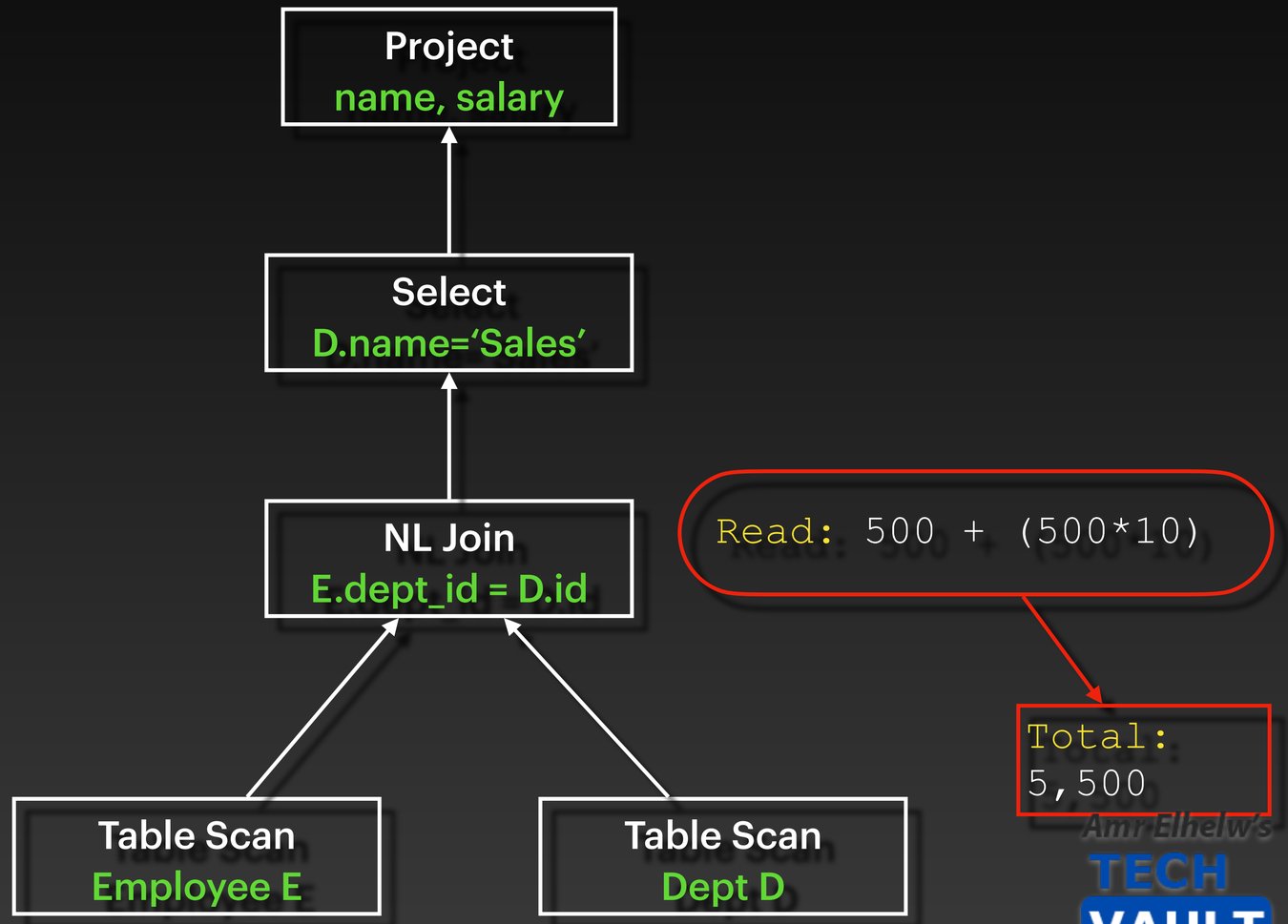
```
Employee (id, name, salary, dept_id)

5,000 rows, 500 pages
10 rows per page
```

Index on (dept_id)

```
Dept (id, name, mgr, location)

100 rows, 10 pages
10 rows per page
```

Index on (name)

```
SELECT name, salary
FROM Employee E
JOIN Dept D ON E.dept_id = D.id
WHERE D.name = 'Sales'
```



Switch Join Order

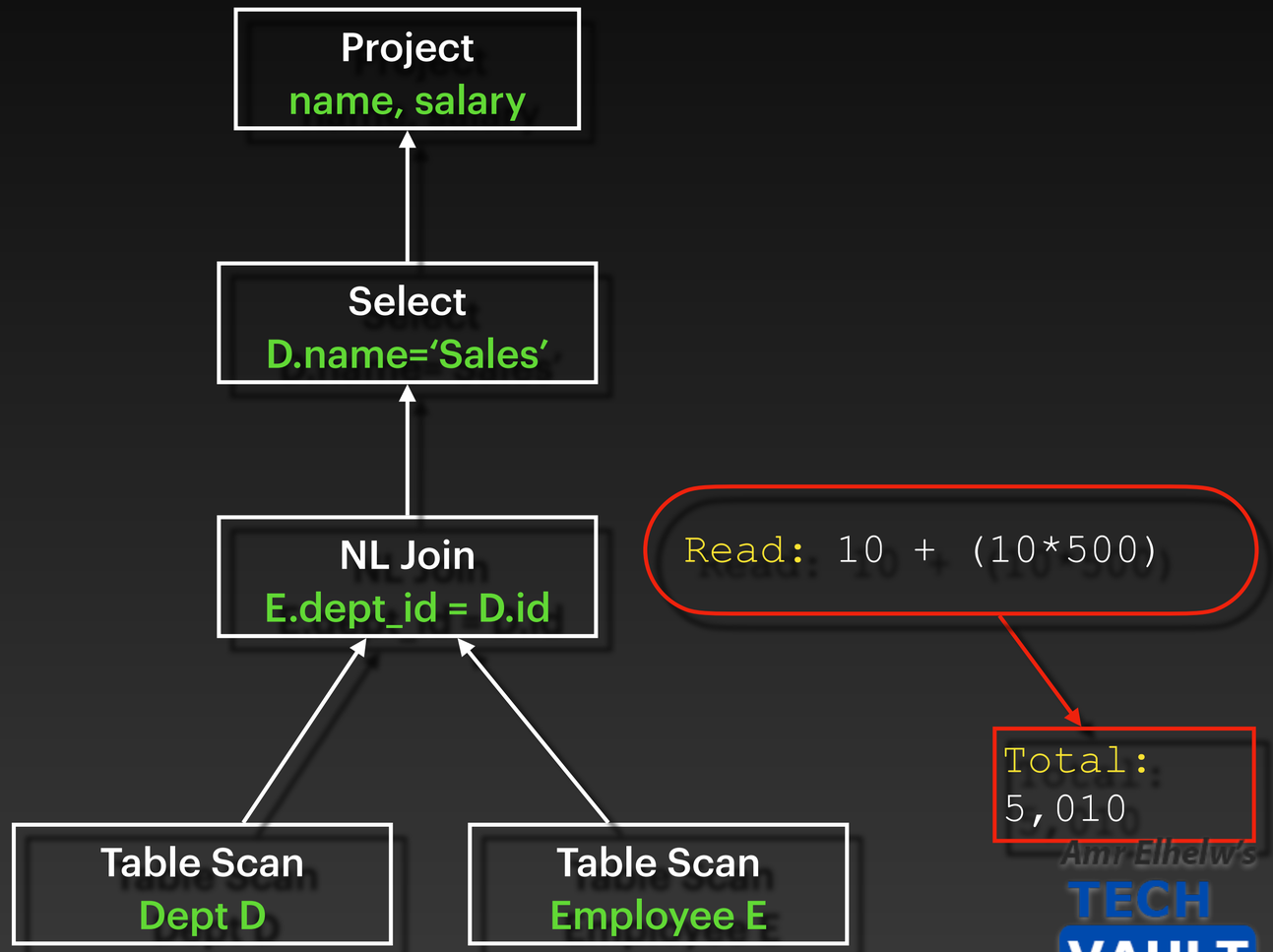
```
Employee (id, name, salary, dept_id)

5,000 rows, 500 pages
10 rows per page
```

Index on (dept_id)

```
Dept (id, name, mgr, location)

100 rows, 10 pages
10 rows per page
```

Index on (name)

```
SELECT name, salary
FROM Employee E
JOIN Dept D ON E.dept_id = D.id
WHERE D.name = 'Sales'
```



Push down selection

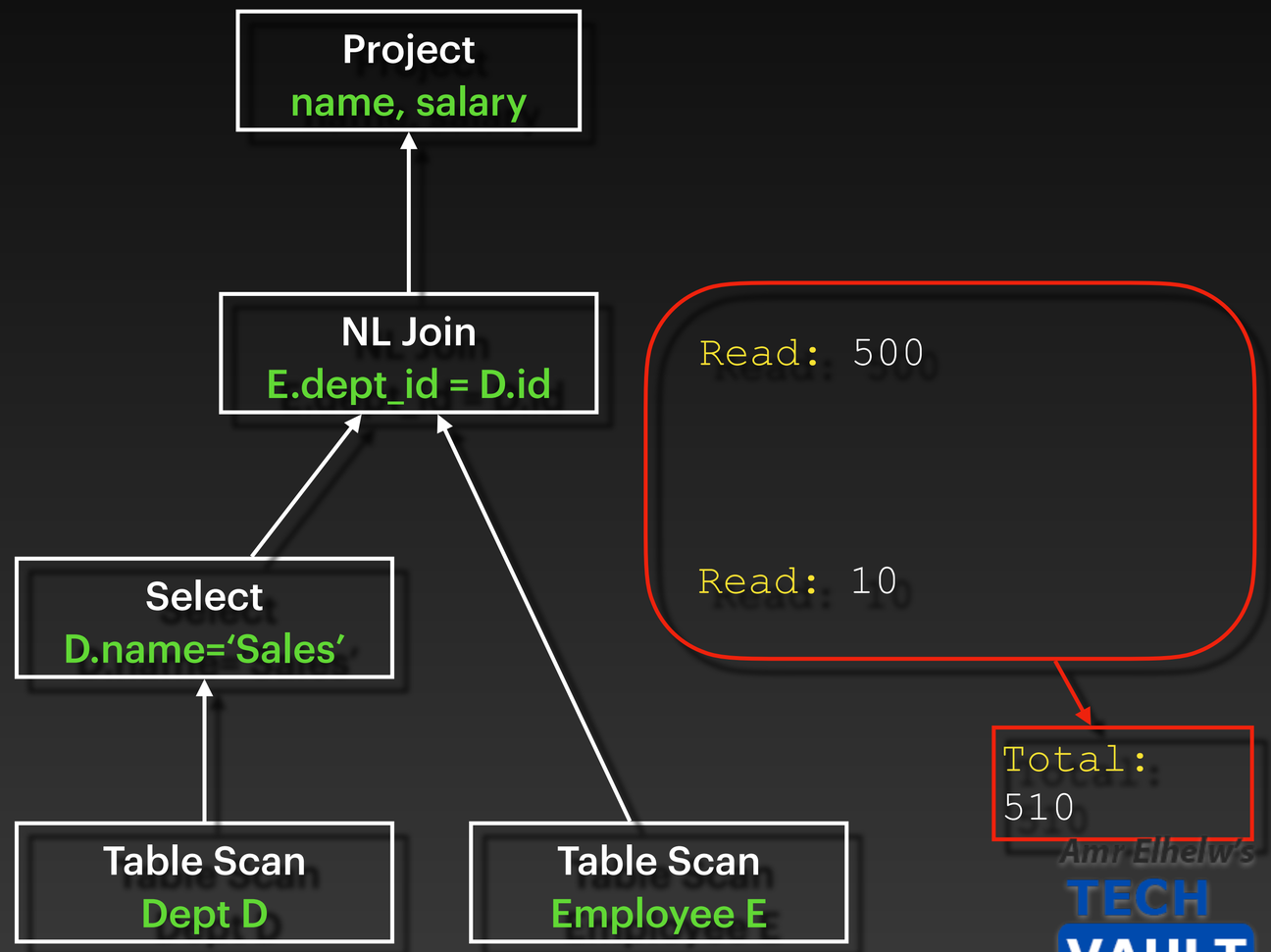
```
Employee (id, name, salary, dept_id)

5,000 rows, 500 pages
10 rows per page
```

Index on (dept_id)

```
Dept (id, name, mgr, location)

100 rows, 10 pages
10 rows per page
```

Index on (name)

```
SELECT name, salary
FROM Employee E
JOIN Dept D ON E.dept_id = D.id
WHERE D.name = 'Sales'
```



```
Employee (id, name, salary, dept_id)

5,000 rows, 500 pages
10 rows per page
```

Index on (dept_id)

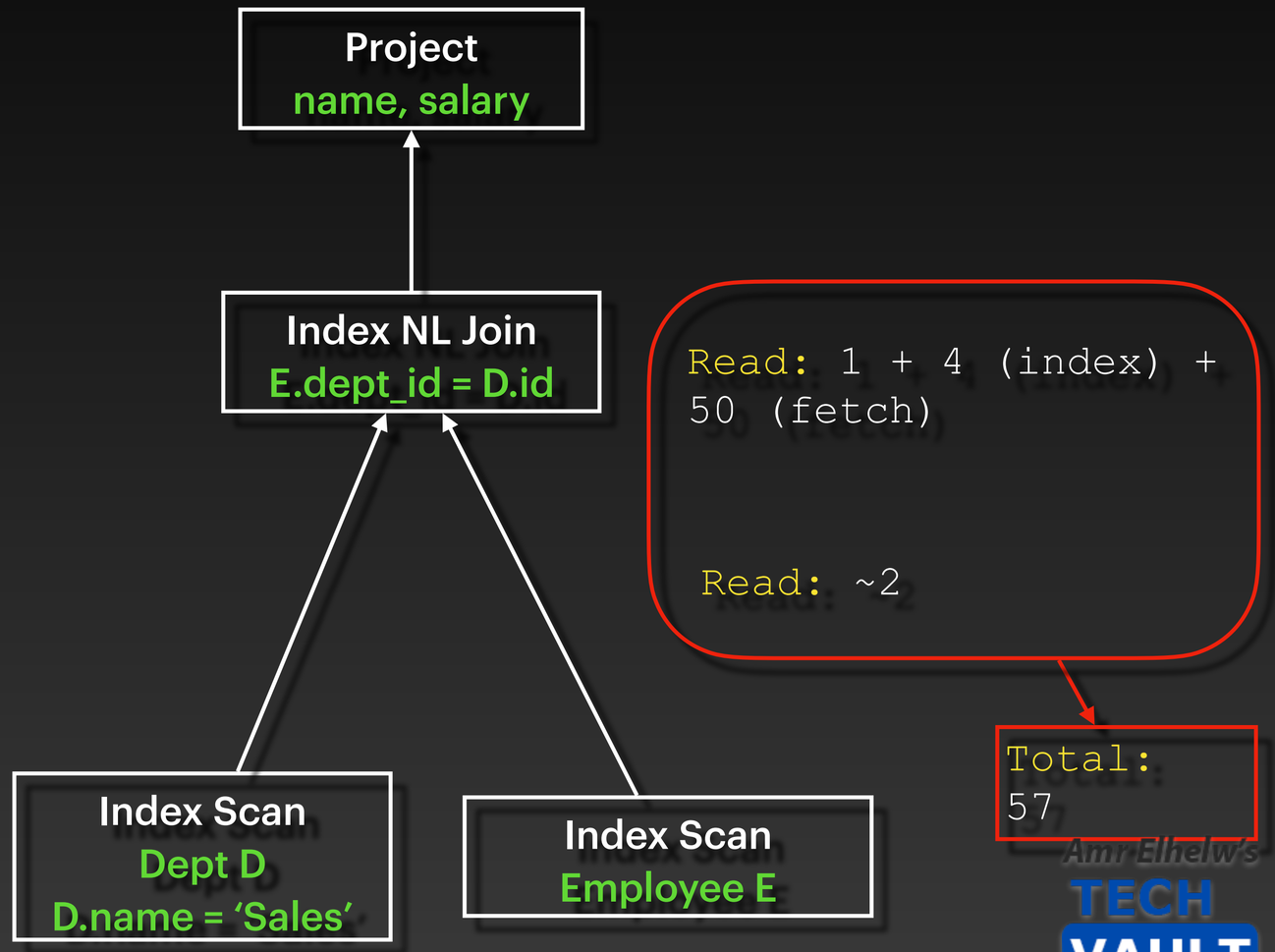
```
Dept (id, name, mgr, location)

100 rows, 10 pages
10 rows per page
```

Index on (name)

```
SELECT name, salary
FROM Employee E
JOIN Dept D ON E.dept_id = D.id
WHERE D.name = 'Sales'
```

Use index on name

& Index NL Join



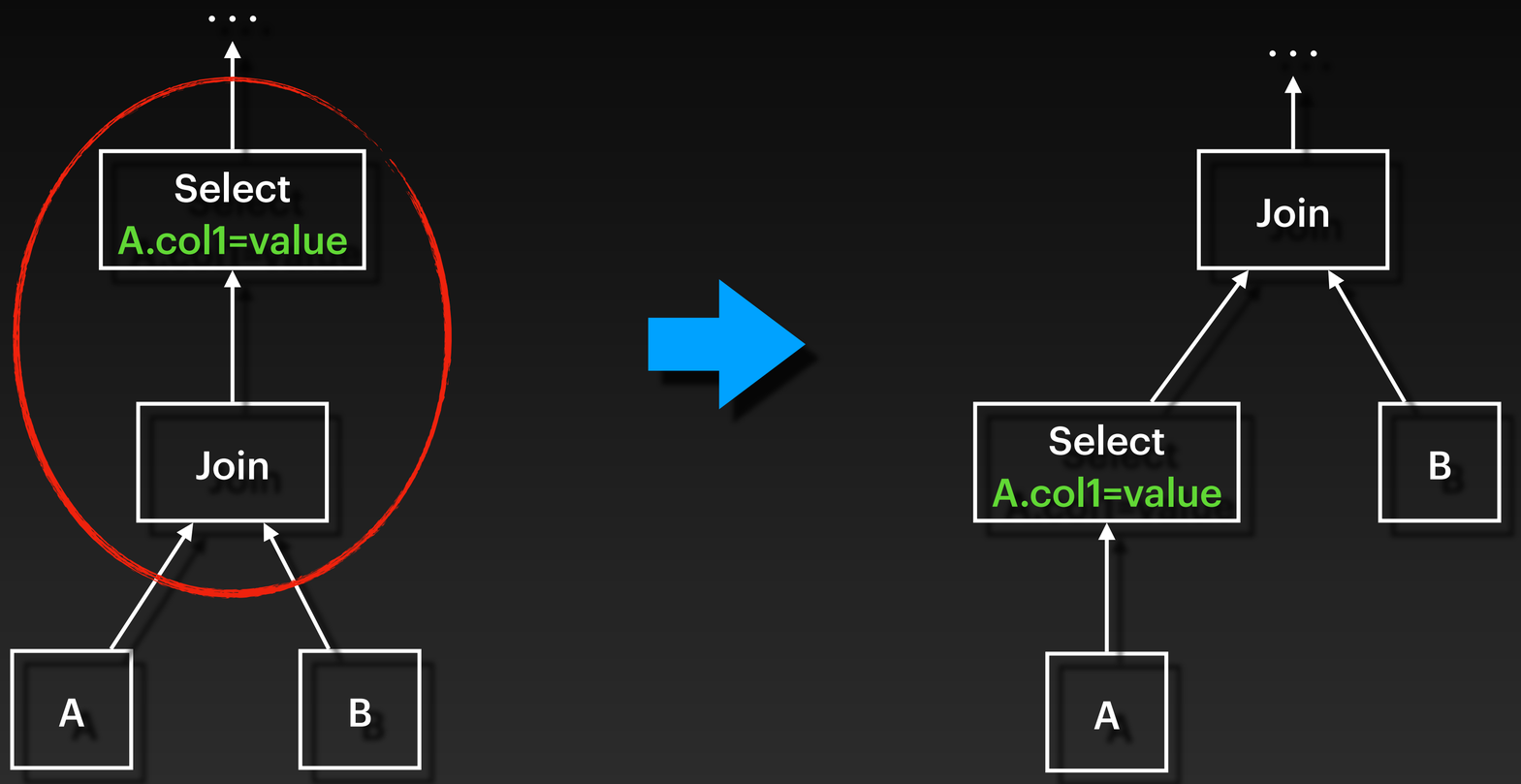
Logical Nodes

- An abstract operation - cannot be executed
- Corresponds to relational algebra
- E.g.: Join, Scan, Aggregation, etc.

Physical Nodes

- An concrete algorithm/implementation
- Can be executed, has a cost function
- E.g.: NL Join, Hash Join, Full scan, Index Scan, Sort- and Hash-based Aggregation, etc.

Filter pushdown



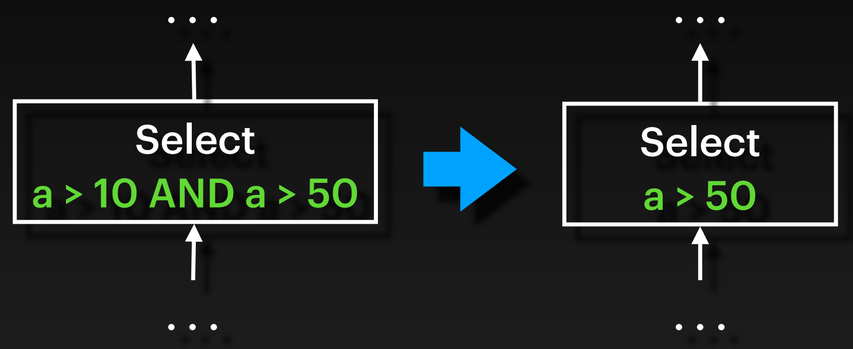
$$\ldots(\sigma_{A.col1=value}(A \bowtie B)) \longrightarrow \ldots(\sigma_{A.col1=value}(A) \bowtie B)$$

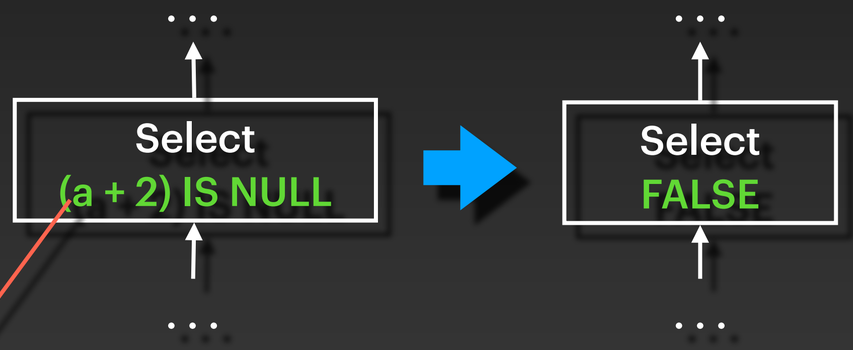
Rule-based Optimization

More relational algebra transformation rules:

https://www.postgresql.org/message-id/attachment/32513/EquivalenceRules.pdf

Expression Simplification

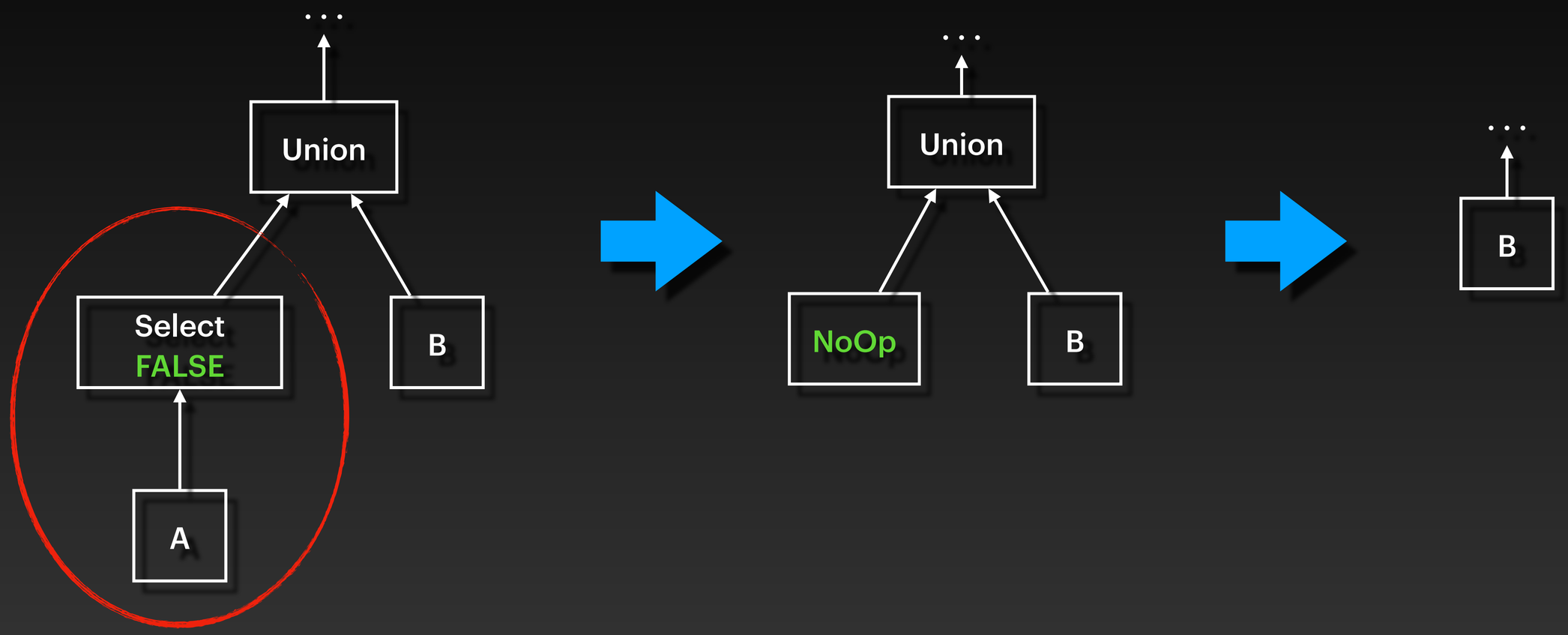




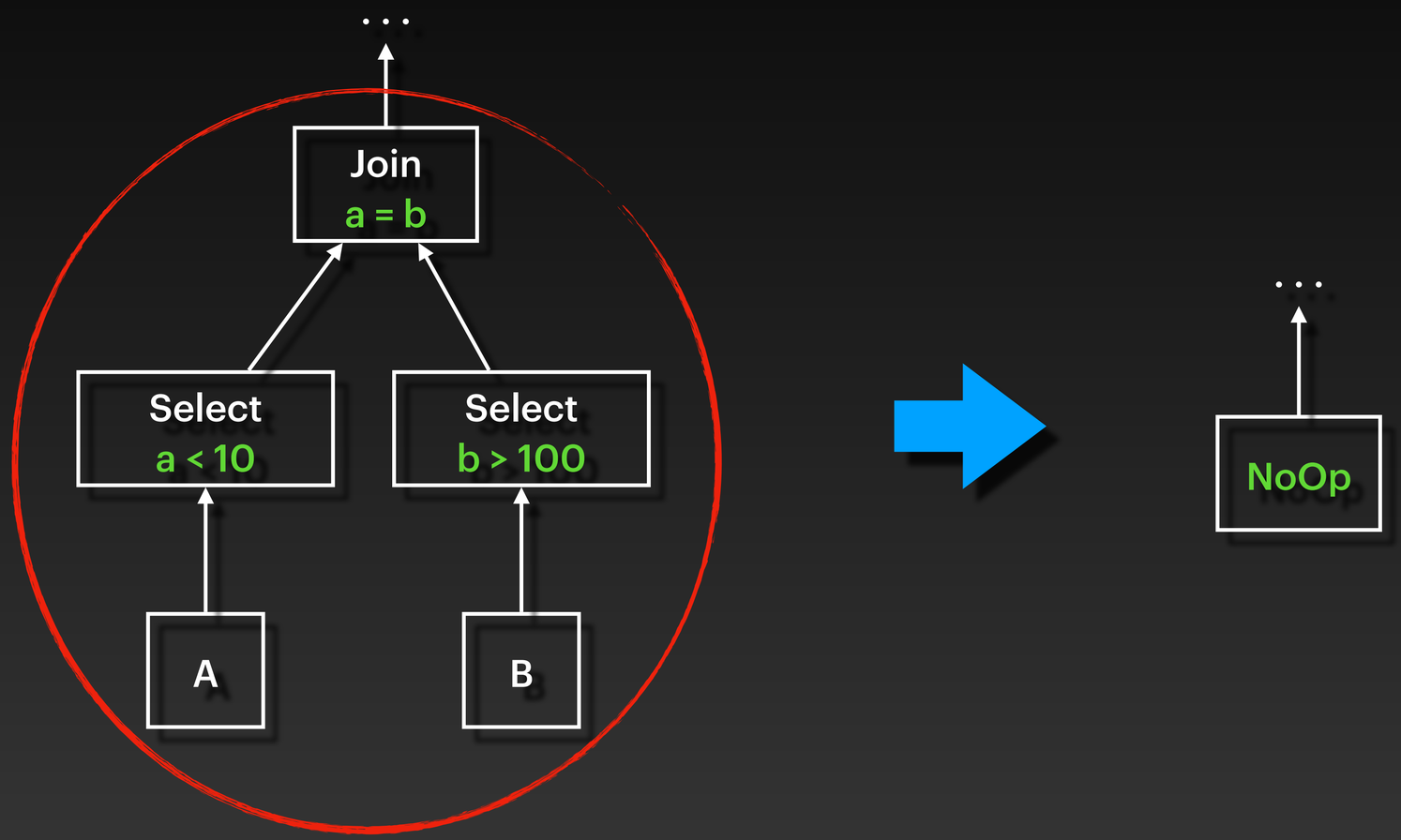
NOT
NULL

Amr Elhelw's
TECH
VAULT

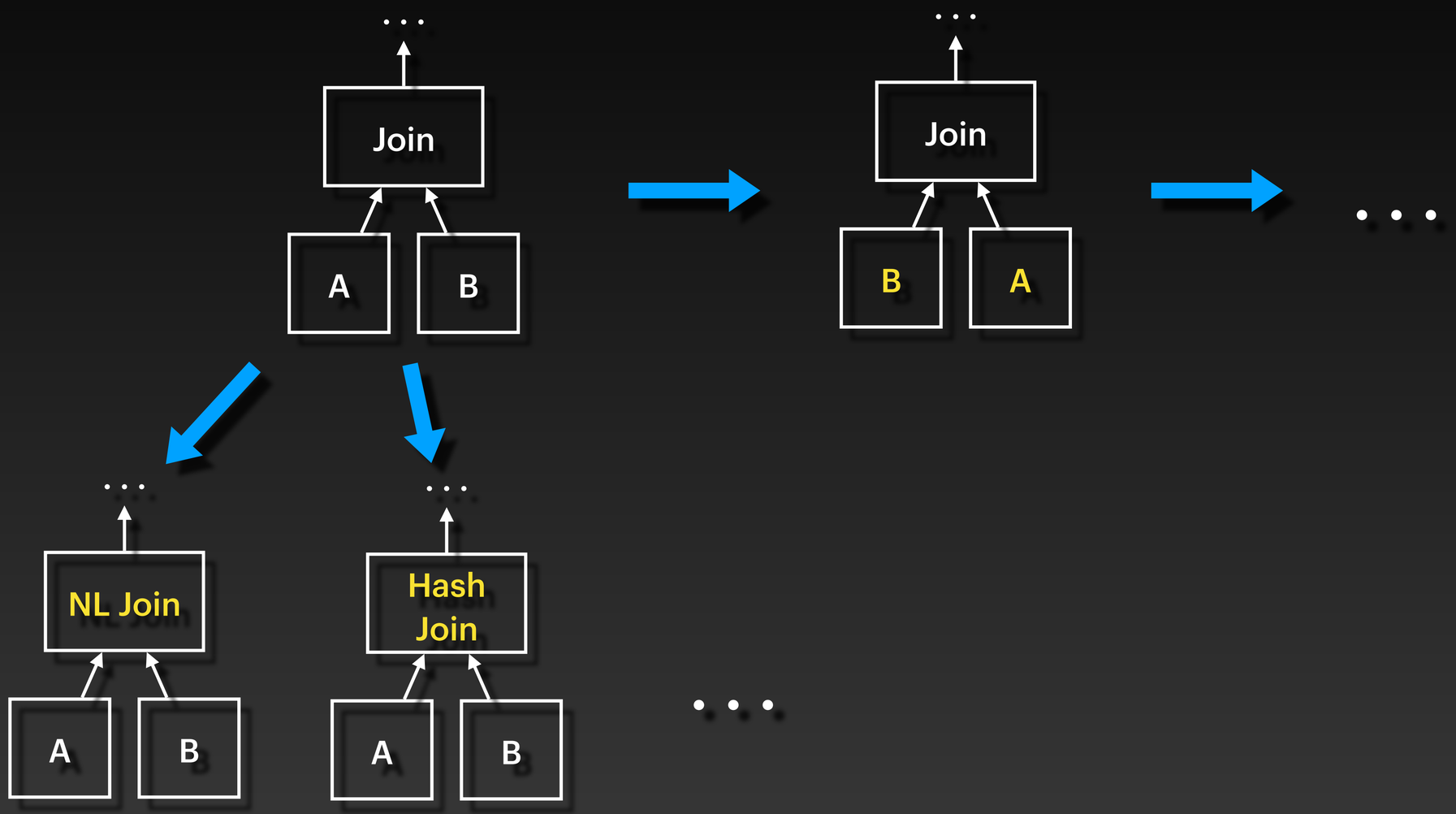
Empty Subtree Elimination



Empty Subtree Elimination



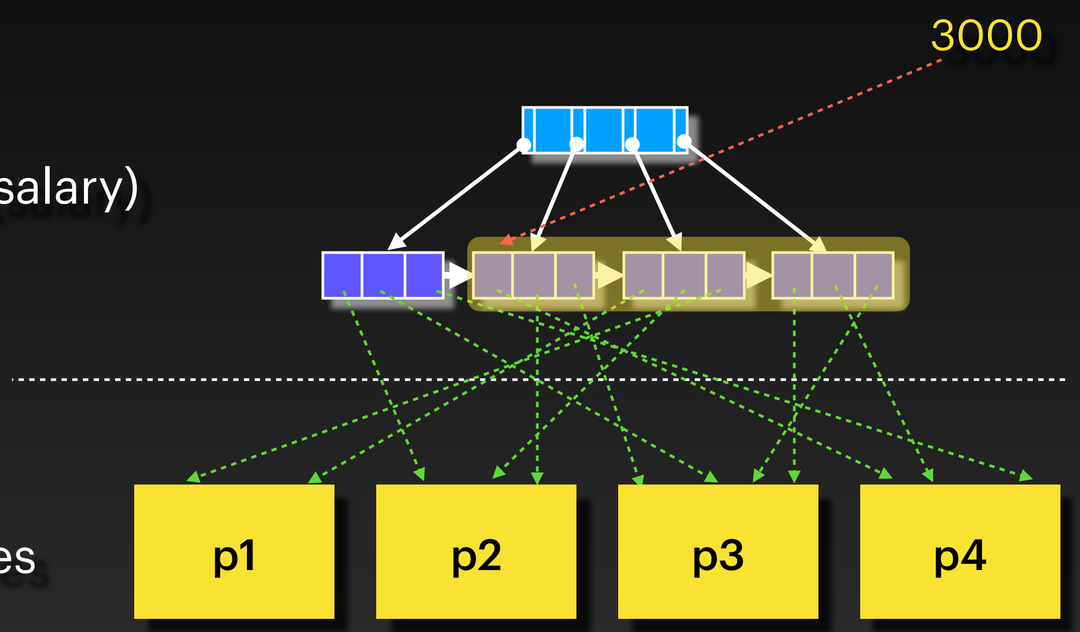
Transformations

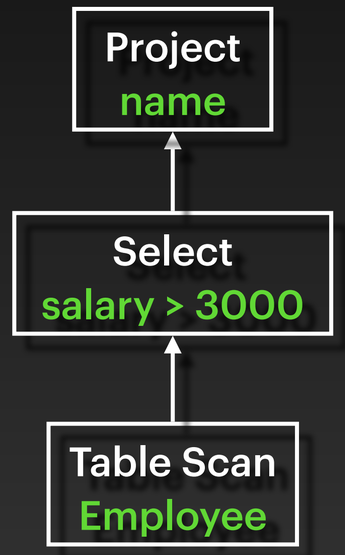


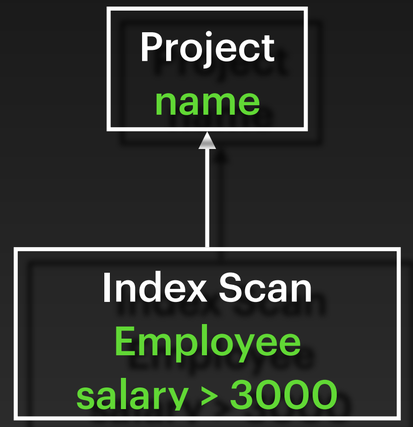
Cost-based Optimization

- Join Order
- Access Path Selection (Full scan, index scan, etc.)
- Join algorithms (NL, Hash, Merge...)
- Aggregation algorithms
- Sorting algorithms
- Etc.

```sql
SELECT name
FROM Employee E
WHERE salary > 3000
```







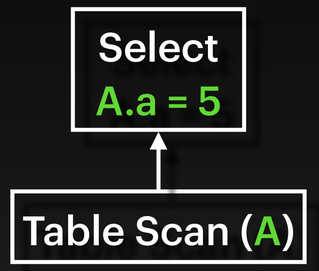
Amr Elhelw's TECH VAULT

Bottom-Up Optimization

- For the given query, enumerate and compare sub-plans with:
  - 1 table
  - 2 tables
  - 3 tables
  - etc.
- At each level find the best sub-plan, and use in higher levels

```
SELECT *
FROM A, B, C
WHERE A.x = B.x AND B.y = C.y
AND A.a = 5
```

Sub-expression	Best plan	Cost
(A)	Index Scan (A)	...
(B)		
(C)		
(A, B)		
(B, C)		
(A, C)		
(A, B, C)		



Cost = ...

Cost = ...

```
SELECT *
FROM A, B, C
WHERE A.x = B.x AND B.y = C.y
AND A.a = 5
```

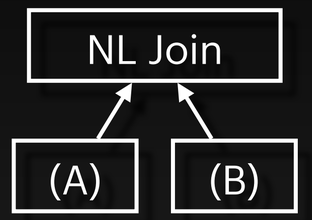
Sub-expression	Best plan	Cost
(A)	Index Scan (A)	...
(B)	Table Scan (B)	...
(C)	Table Scan (C)	...
(A, B)		
(B, C)		
(A, C)		
(A, B, C)		

Table Scan (B)

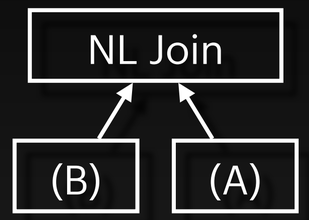
Cost = ...

```
SELECT *
FROM A, B, C
WHERE A.x = B.x AND B.y = C.y
AND A.a = 5
```

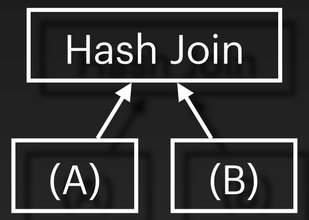
Sub-expression	Best plan	Cost
(A)	Index Scan (A)	...
(B)	Table Scan (B)	...
(C)	Table Scan (C)	...
(A, B)	(B) NL Join (A)	...
(B, C)		
(A, C)		
(A, B, C)		



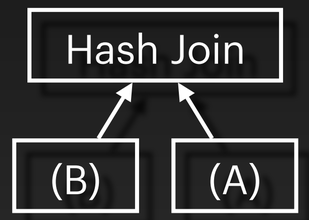
Cost = ...



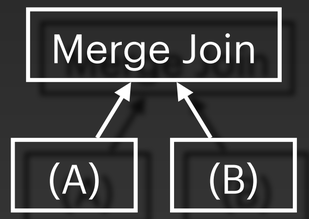
Cost = ...



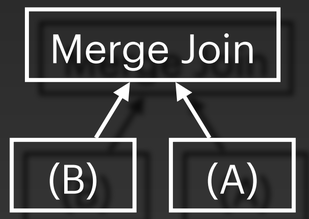
Cost = ...



Cost = ...



Cost = ...



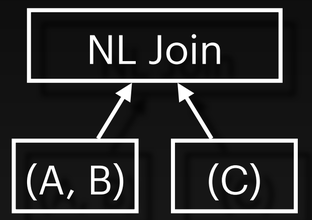
Cost = ...

```
SELECT *
FROM A, B, C
WHERE A.x = B.x AND B.y = C.y
AND A.a = 5
```

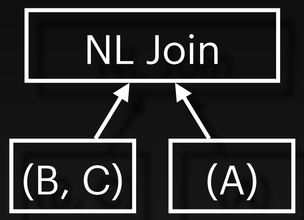
Sub-expression	Best plan	Cost
(A)	Index Scan (A)	...
(B)	Table Scan (B)	...
(C)	Table Scan (C)	...
(A, B)	(B) NL Join (A)	...
(B, C)	(B) Hash Join (C)	...
~~(A, C)~~		
(A, B, C)		

```
SELECT *
FROM A, B, C
WHERE A.x = B.x AND B.y = C.y
AND A.a = 5
```

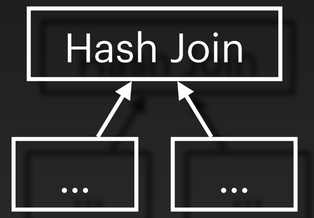
Sub-expression	Best plan	Cost
(A)	Index Scan (A)	...
(B)	Table Scan (B)	...
(C)	Table Scan (C)	...
(A, B)	(B) NL Join (A)	...
(B, C)	(B) Hash Join (C)	...
~~(A, C)~~		
(A, B, C)	(B, C) NL Join (A)	...



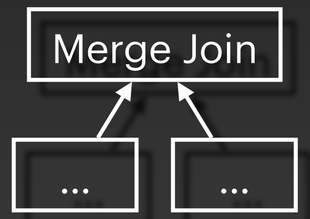
Cost = ...



Cost = ...



. . .

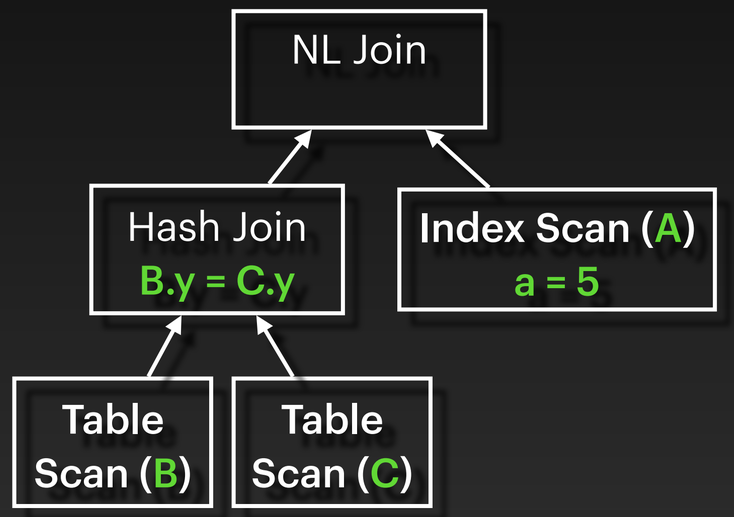


. . .

Amr Elhelw's TECH VAULT

```
SELECT *
FROM A, B, C
WHERE A.x = B.x AND B.y = C.y
AND A.a = 5
```

Sub-expression	Best plan	Cost
(A)	Index Scan (A)	...
(B)	Table Scan (B)	...
(C)	Table Scan (C)	...
(A, B)	(B) NL Join (A)	...
(B, C)	(B) Hash Join (C)	...
~~(A, C)~~		
(A, B, C)	(B, C) NL Join (A)	...



Bottom-Up Optimization

(+) Simple to implement

(-) Adding constraints about output column, sort order, etc. is not easy

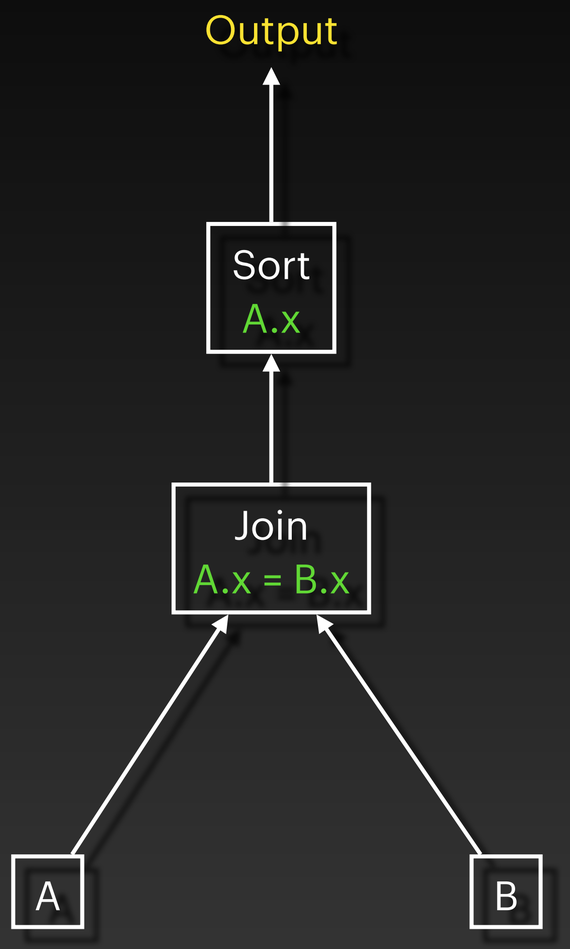
(-) Usually considers only left-deep trees

IBM System R, DB2, MySQL, Postgres, most open-source DBMSs

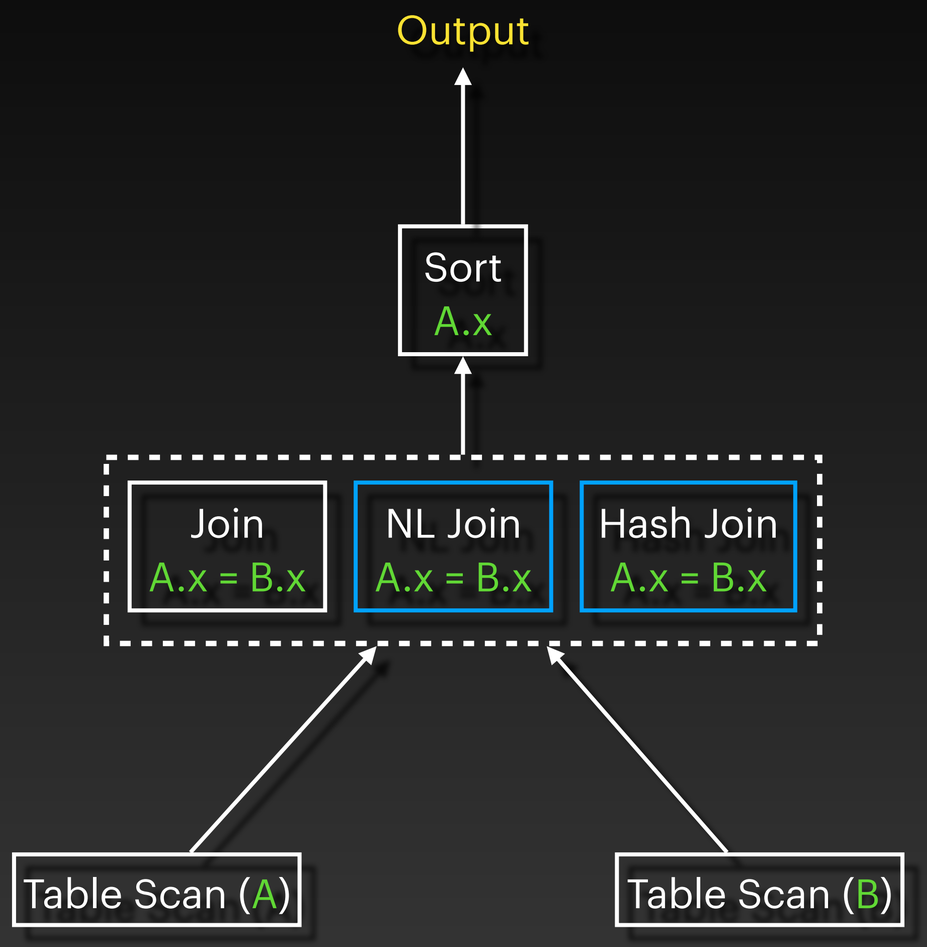
Top-down Optimization

- Start from logical plan (after applying heuristic rules)
- Apply more rules to generate other logical and physical alternatives
- Estimate cost and choose cheapest alternative

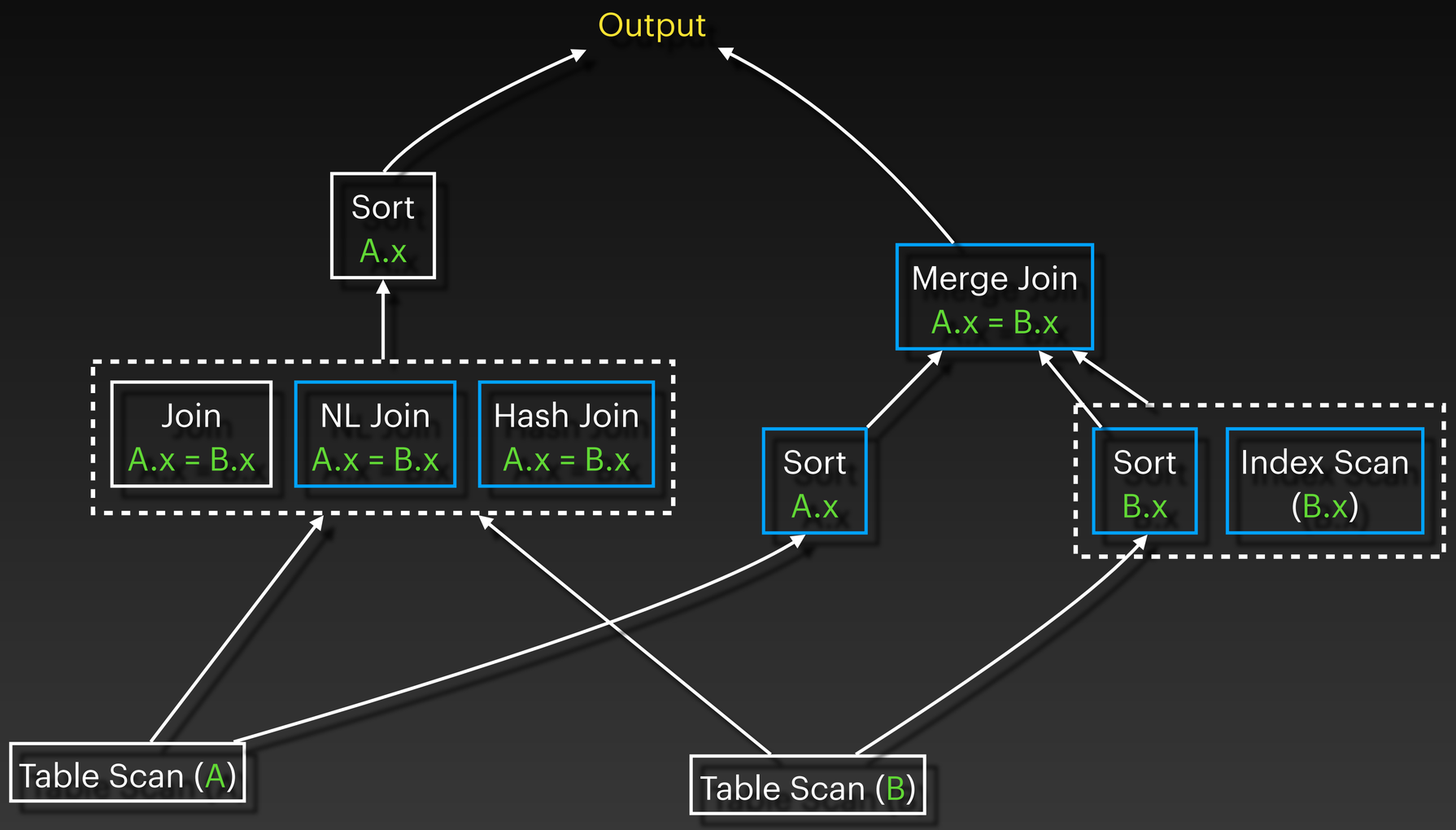
```
SELECT *
FROM A, B
WHERE A.x = B.x
ORDER BY A.x
```



```
SELECT *
FROM A, B
WHERE A.x = B.x
ORDER BY A.x
```



```
SELECT *
FROM A, B
WHERE A.x = B.x
ORDER BY A.x
```



Top-down Optimization

(+) Sort order, output columns, etc. are an essential part of the framework

(+) More plan alternatives

(-) Somewhat complex

Volcano/Cascades Framework

SQL Server, Greenplum DB