Learning to Hack on Postgres Planner

Melanie Plageman

Goals

- Provide a tangible, trivial example of adding a fix to PostgreSQL planner
- Start a discussion on specifying where to add new optimizations to PostgreSQL planner

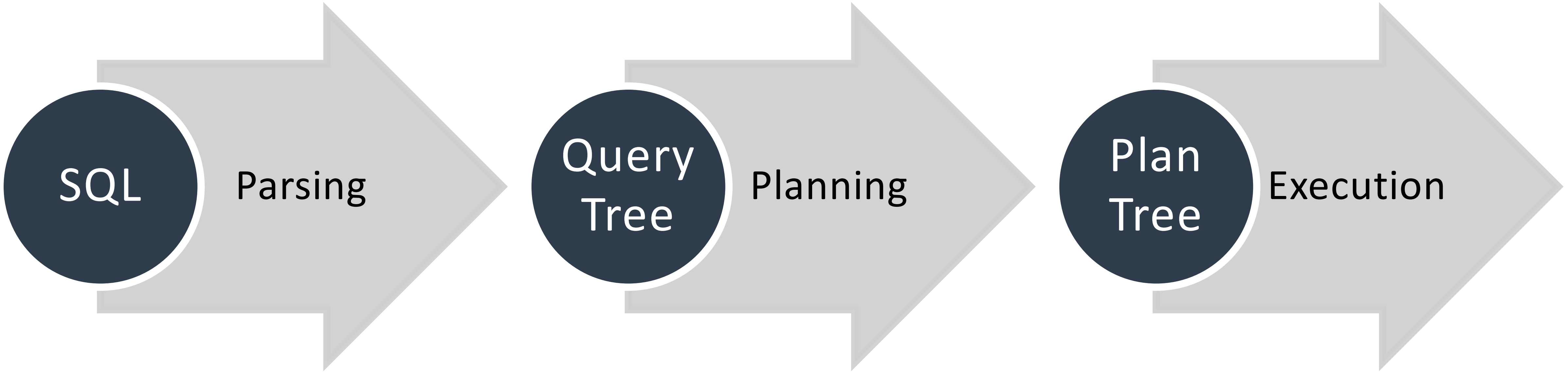
Table of Contents

Query Planning

SQL statement to plan tree

```
# SELECT a FROM foo;
```

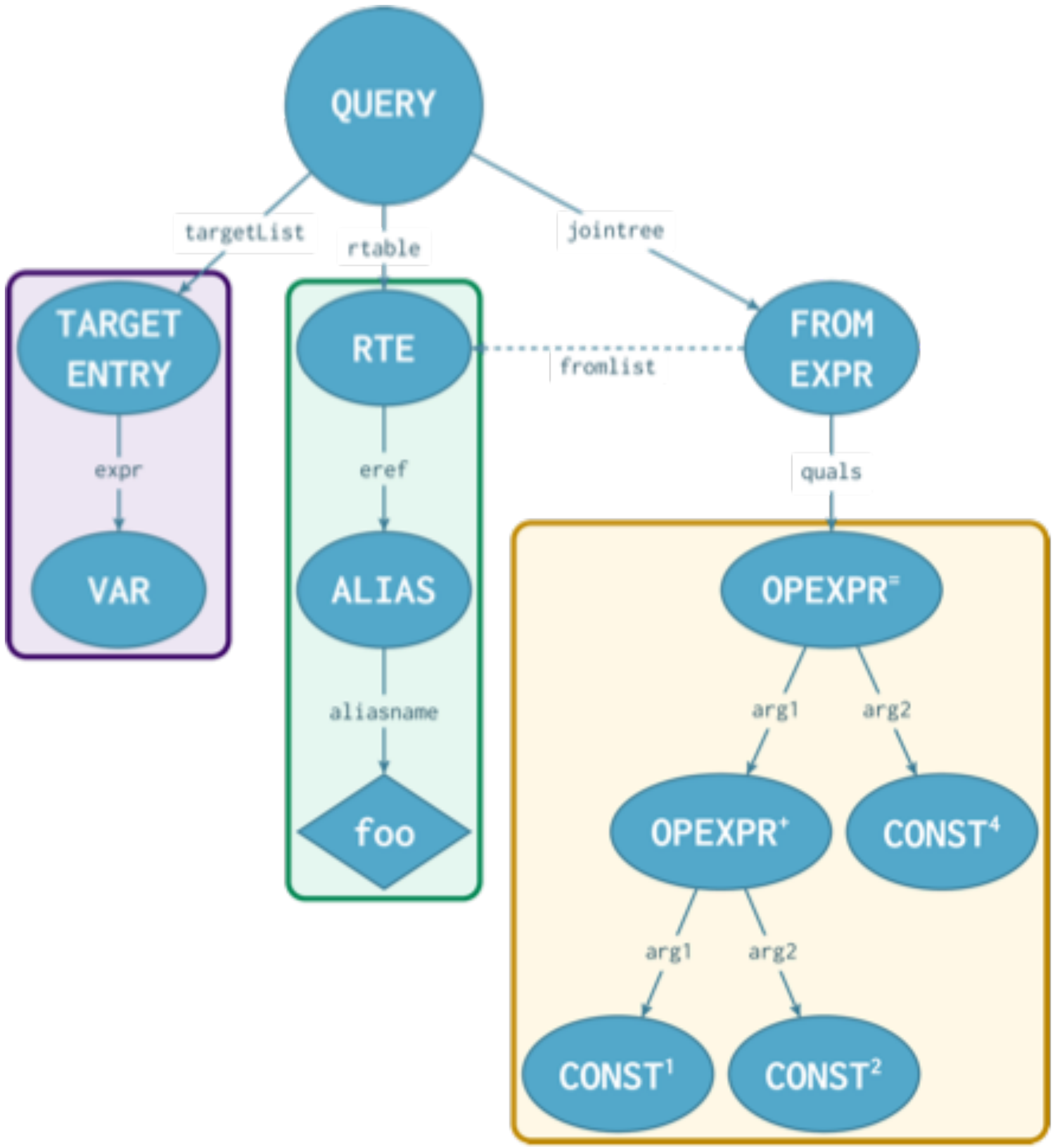
```
 a
---
 1
 2
 4
(3 rows)
```


SQL
Parsing
Query Tree
Planning
Plan Tree
Execution

Query Tree

```
# SET debug_print_parse
  TO on;

# SELECT a
  FROM foo
  WHERE 1 + 2 = 4;
```

```
{QUERY
:rtable (
   {RTE
   :eref
      {ALIAS
      :aliasname foo
      :colnames ("a")
:jointree
   {FROMEXPR
   :quals
      {OPEXPR
      :args (
         {OPEXPR
         :args (
            {CONST
            :constvalue 4 [ 1 … ]
            {CONST
            :constvalue 4 [ 2 … ]
         {CONST
         :constvalue 4 [ 4 … ]
:targetList (
   {TARGETENTRY
   :expr
      {VAR
   :resname a
```



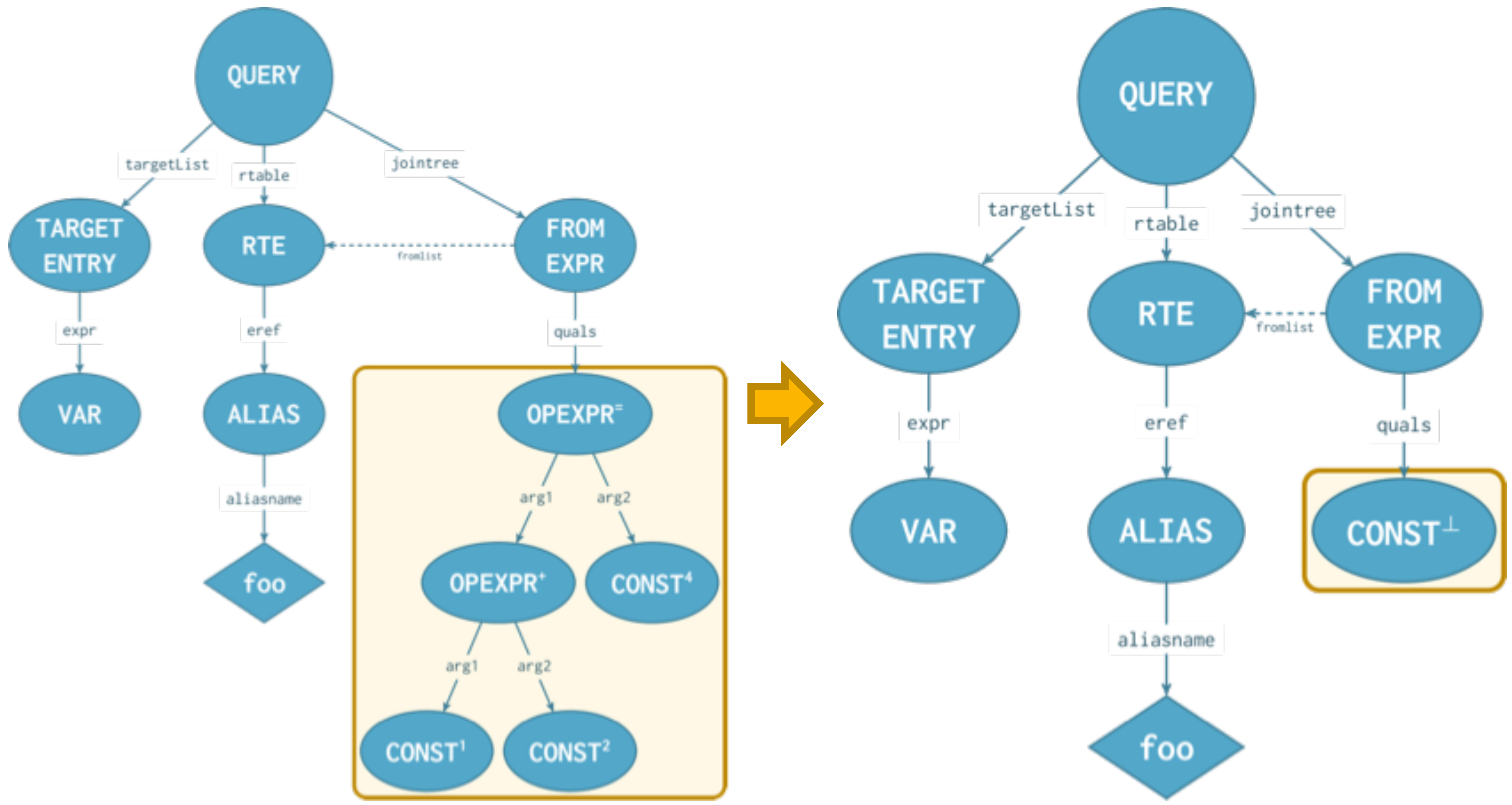
```
{QUERY
:rtable (
   {RTE
   :eref
      {ALIAS
      :aliasname foo
      :colnames ("a")
:jointree
   {FROMEXPR
   :quals
      {OPEXPR
      :args (
         {OPEXPR
         :args (
            {CONST
            :constvalue 4 [ 1 … ]
            {CONST
            :constvalue 4 [ 2 … ]
         {CONST
         :constvalue 4 [ 4 … ]
:targetList (
   {TARGETENTRY
   :expr
      {VAR
   :resname a
```

Semantic Optimization

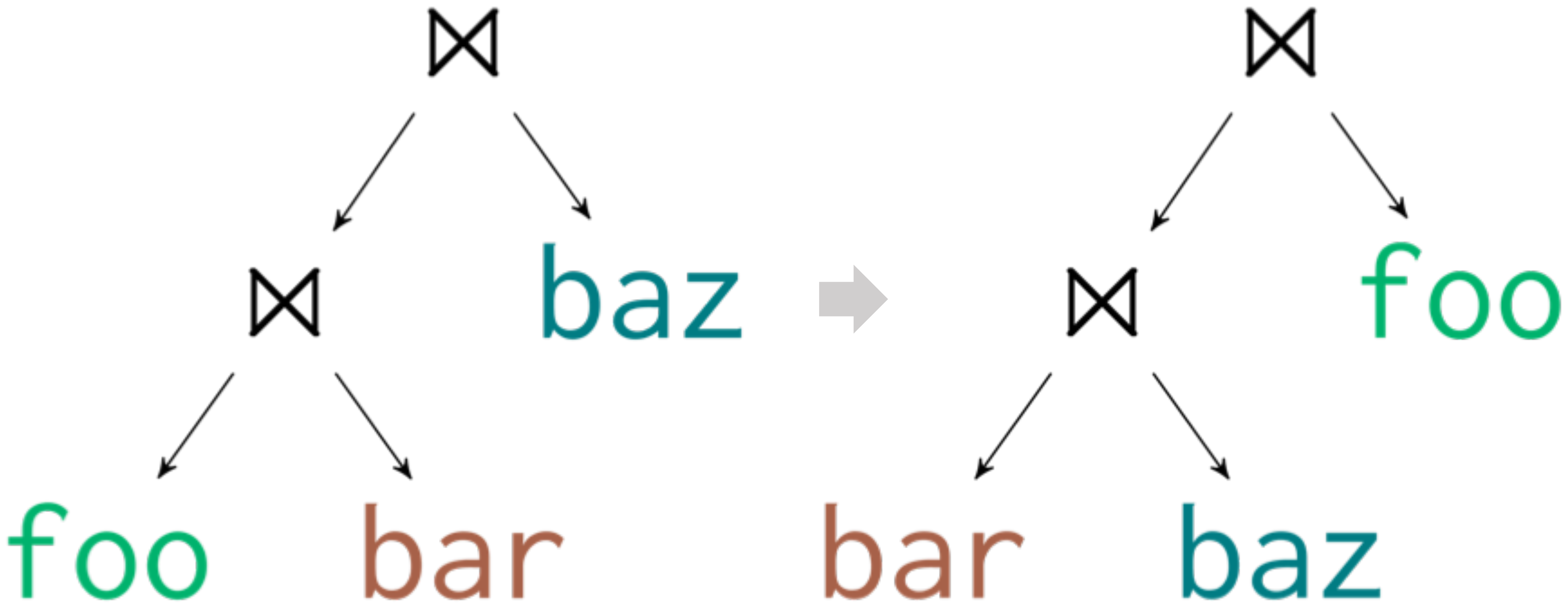
```
# SELECT a FROM foo WHERE 1 + 2 = 4;

                        1 + 2 = 4
                          ⇓
                        FALSE

# SELECT a FROM foo WHERE FALSE;
```

```
SELECT a FROM foo WHERE 1 + 2 = 4;
```

```
SELECT a FROM foo WHERE FALSE;
```



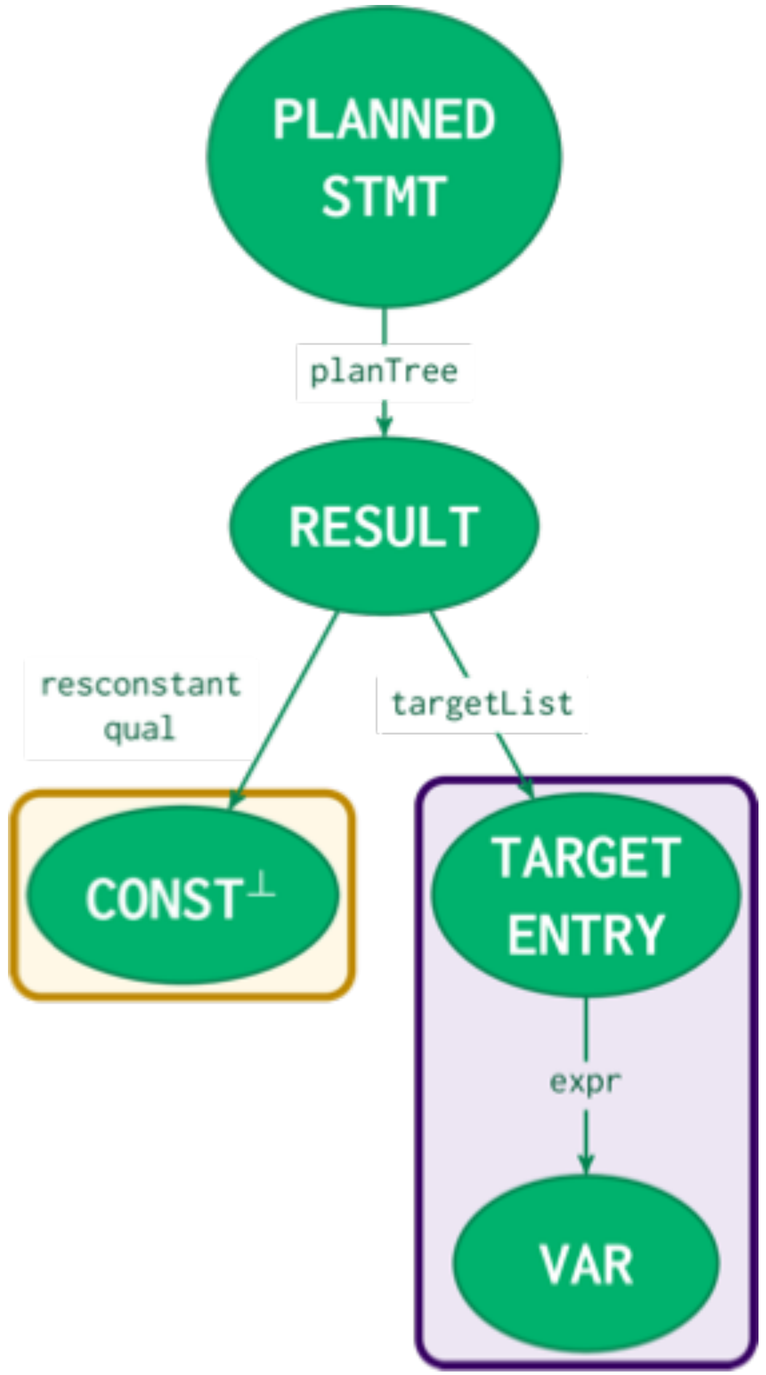
Cost-based Optimization



Plan Tree

```
# SET debug_print_plan
  TO on;

# SELECT a
  FROM foo
  WHERE 1 + 2 = 4;
```

```
{PLANNEDSTMT
:planTree
   {RESULT
   :targetlist (
      {TARGETENTRY
      :expr
         {VAR
      :resname a
   :resconstantqual (
      {CONST
      :constvalue 1 [ 0 … ]
```



```
{PLANNEDSTMT
:planTree
   {RESULT
   :targetlist (
      {TARGETENTRY
      :expr
         {VAR
      :resname a
   :resconstantqual (
      {CONST
      :constvalue 1 [ 0 … ]
```

Guidelines for New Optimizations

① Does it always retain semantic correctness?

$$A \mathbin{⟕} (B \bowtie C)$$

$$\neq$$

$$(A \mathbin{⟕} B) \bowtie C$$

An example from `src/backend/optimizer/README`

② Does it inhibit downstream optimizations?

Optimization Order Matters

An optimization for one query can be a regression for another

Planning steps have expectations for the query tree

② Does it inhibit downstream optimizations?

Optimization Order Matters

```
SELECT * FROM A, B, C
WHERE a IN (
  SELECT b FROM B WHERE b = 5
) AND a = c
  AND c = 7;
```

```
c = 7
c = a ⇒ a = 7
{ a, c, 7 } =

b = 5
{ b, 5 } =
```

② Does it inhibit downstream optimizations?

Optimization Order Matters

```
c = 7
c = a ⇒ a = 7
{ a, c, 7 } =

b = 5
{ b, 5 } =
```

1. Pullup

2. Pre-process

```
c = 7
c = a ⇒ a = 7
b = 5

a = b ⇒ a = 5,
      ⇒ c = 5

{ a, c, 7, b, 5 } =
```

② Does it inhibit downstream optimizations?

Optimization Order Matters

```
SELECT * FROM A, B, C
WHERE a IN (
  SELECT b FROM B WHERE b = 5
) AND a = c
  AND c = 7;
```

```
     QUERY PLAN
---------------------
Result
  One-Time Filter: false
```

② Does it inhibit downstream optimizations?

Order matters

An optimization for one query can be a regression for another

Planning steps have expectations for the query tree

② Does it inhibit downstream optimizations?

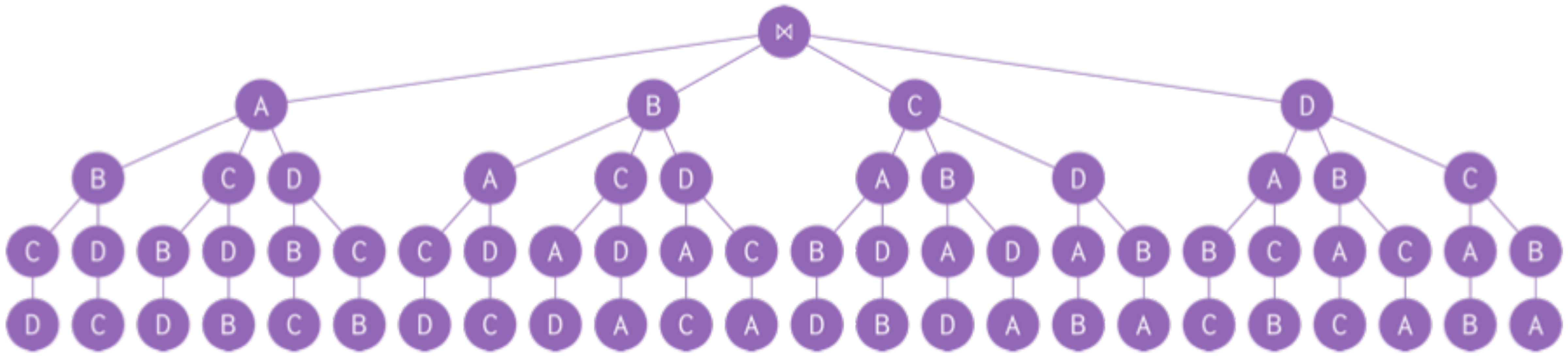
Order matters

An optimization for one query can be a regression for another

Planning steps have expectations for the query tree

③ Is the improvement in execution time worth the cost in planning time?

No in the case of exhaustive join order = $O(n!)$



④ Is the complexity cost commensurate with the performance benefit?

- **Narrow use cases**
- Optimizations for obscure features
- New APIs without reuse potential

Case Study

Adding a planner improvement

Table "public.foo"

Column	Type
a	integer

Table "public.bar"

Column	Type
b	integer

```
# SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
```

NULL ≈ Unknown

`p`	`q`	`p OR q`	`p AND q`	`p = q`

NULL ≈ Unknown

p	q	p OR q	p AND q	p = q
TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE

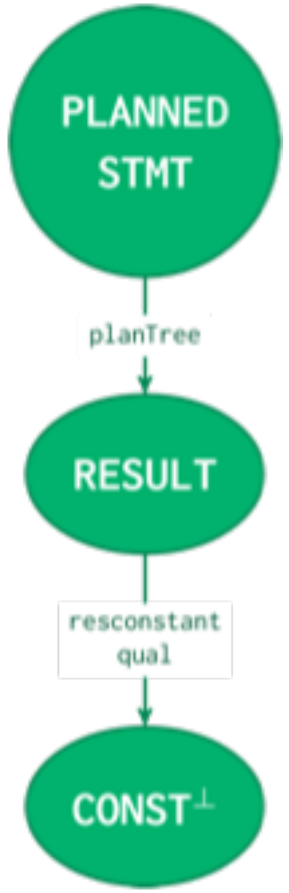
NULL ≈ Unknown

p	q	p OR q	p AND q	p = q
TRUE	TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	TRUE	FALSE	FALSE
FALSE	FALSE	FALSE	FALSE	TRUE
TRUE	NULL	TRUE	NULL	NULL
FALSE	NULL	NULL	FALSE	NULL
NULL	NULL	NULL	NULL	NULL

EXPLAIN Output?

```
# EXPLAIN SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
```

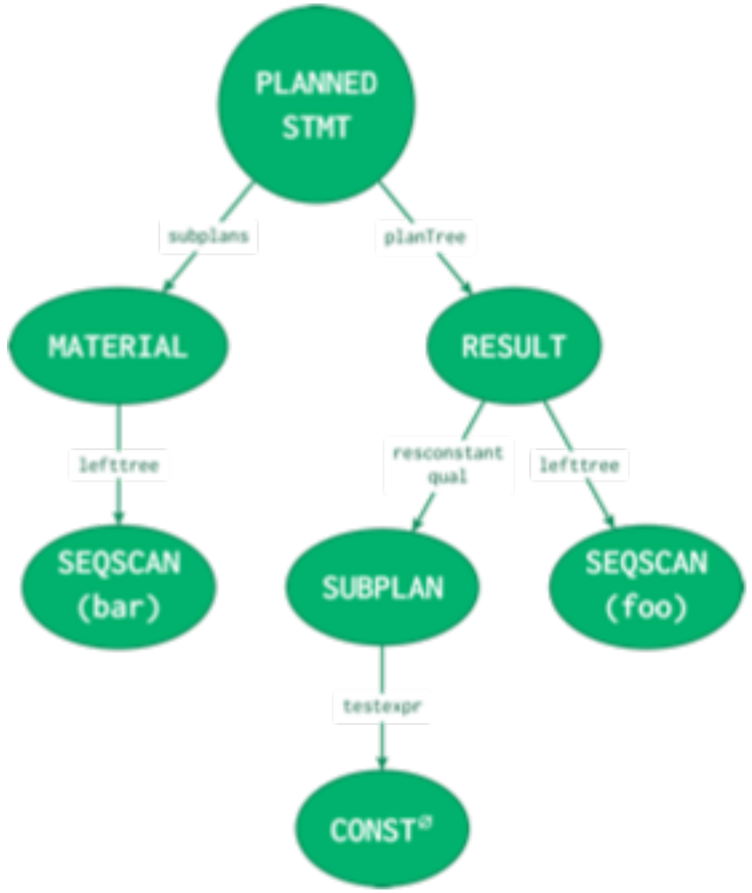
```
          QUERY PLAN
-------------------------------
Result
  One-Time Filter: false
```



EXPLAIN Output!

```
# EXPLAIN SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
```

```
          QUERY PLAN
──────────────────────────
Result
  One-Time Filter: (SubPlan 1)
  → Seq Scan on foo
  SubPlan 1
  → Materialize
    → Seq Scan on bar
```



Target Transformation

1. **Characterize the query**
2. Find analogues
3. Identify transformations

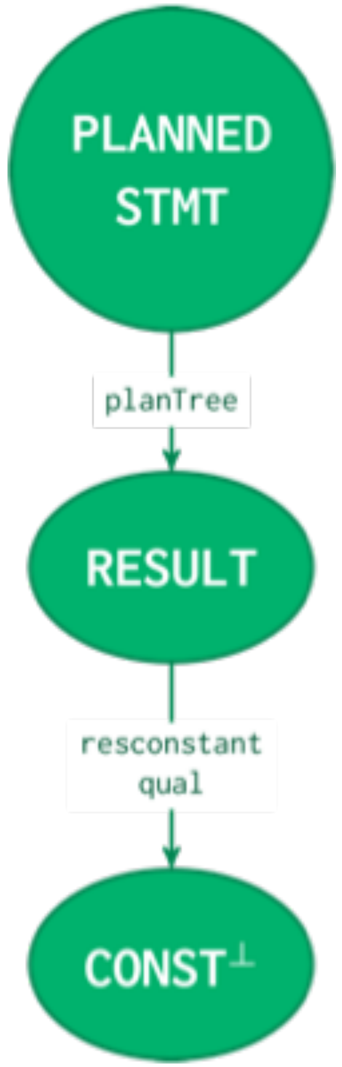
Provably UNTRUE `quals`

```
# SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);

                         NULL = ANY(SELECT b FROM bar)
                            ⇓
                         UNTRUE

# SELECT a FROM foo WHERE UNTRUE;
```

Target Transformation

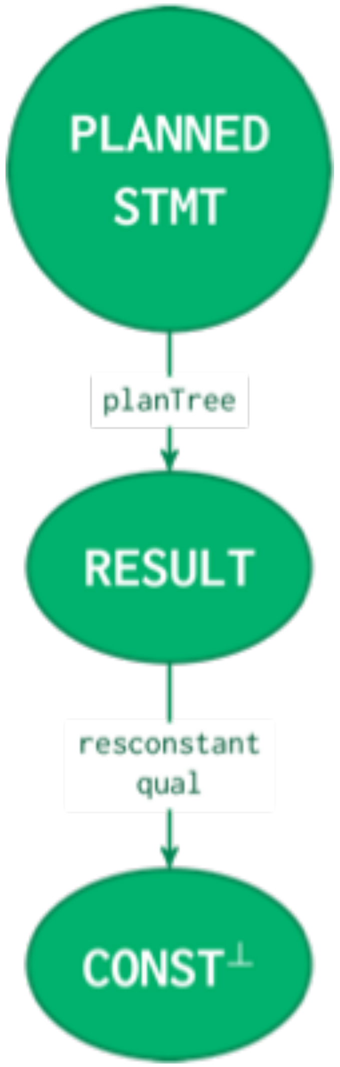
1. Characterize the query
2. **Find analogues**
3. Identify transformations

```
# EXPLAIN SELECT a FROM foo WHERE FALSE;

          QUERY PLAN
──────────────────────────
Result
  One-Time Filter: false
```

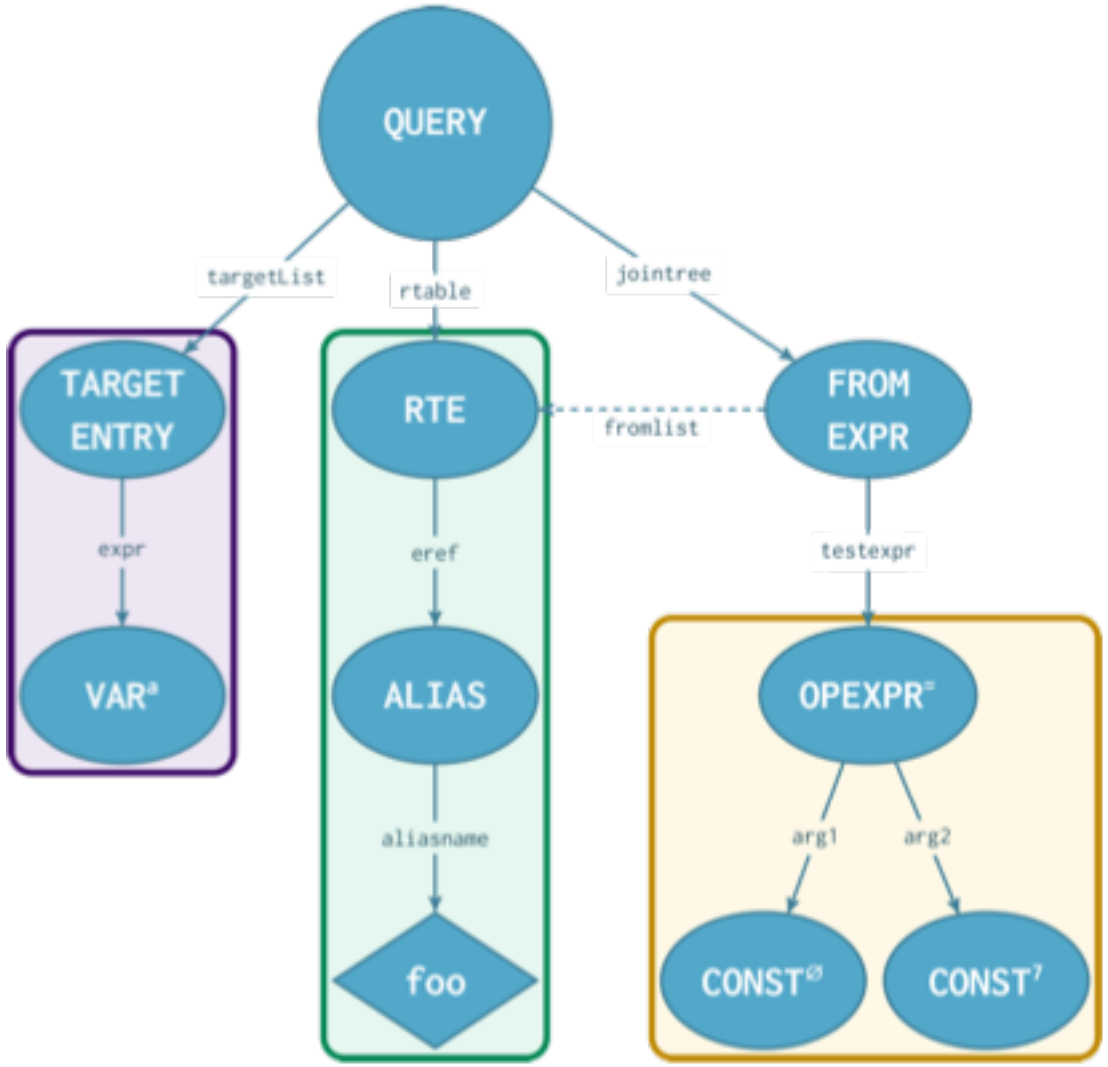


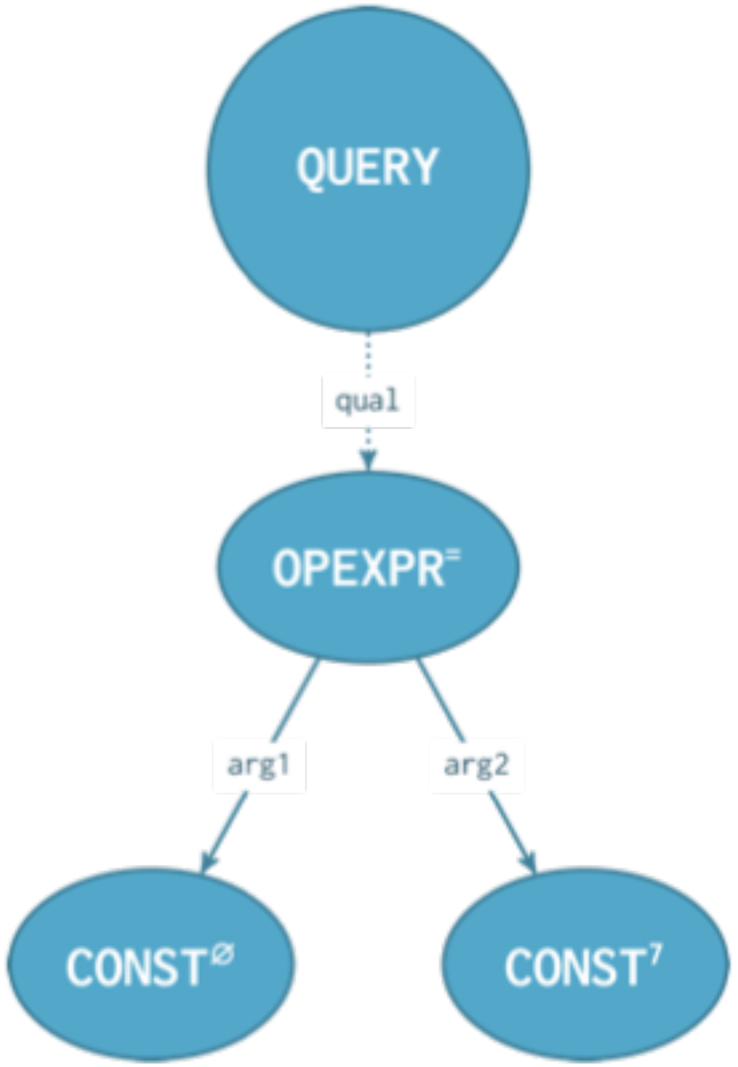
```
# EXPLAIN SELECT a FROM foo WHERE NULL = 7;
```

```
         QUERY PLAN
─────────────────────────
Result
  One-Time Filter: false
```



A Note on Notation

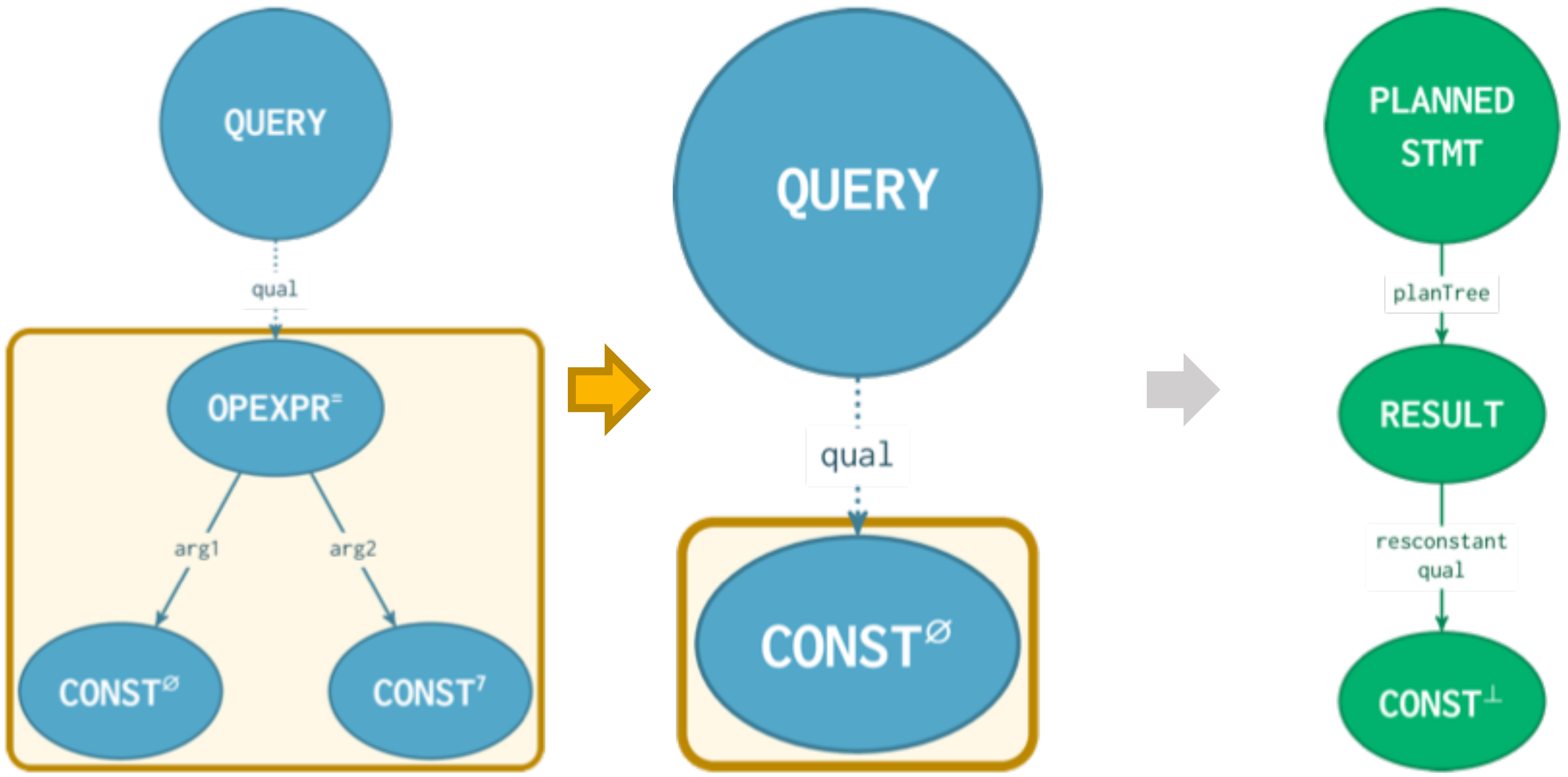




Target Transformation

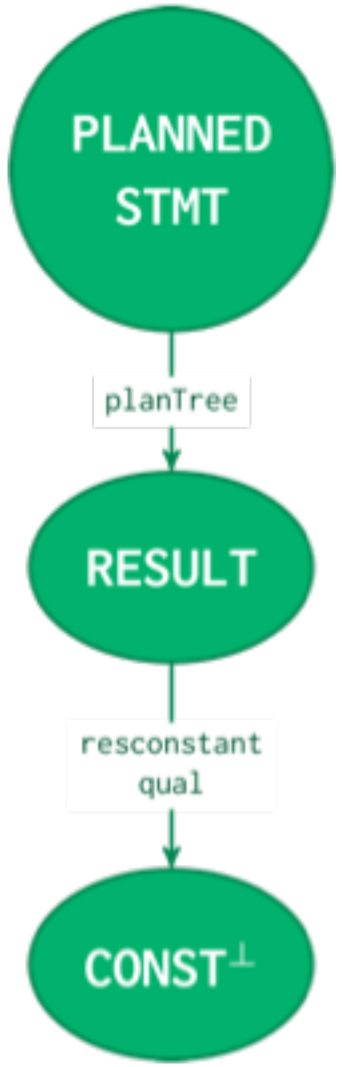
1. Characterize the query
2. Find analogues
3. **Identify transformations**

`# SELECT a FROM foo WHERE NULL = 7;`

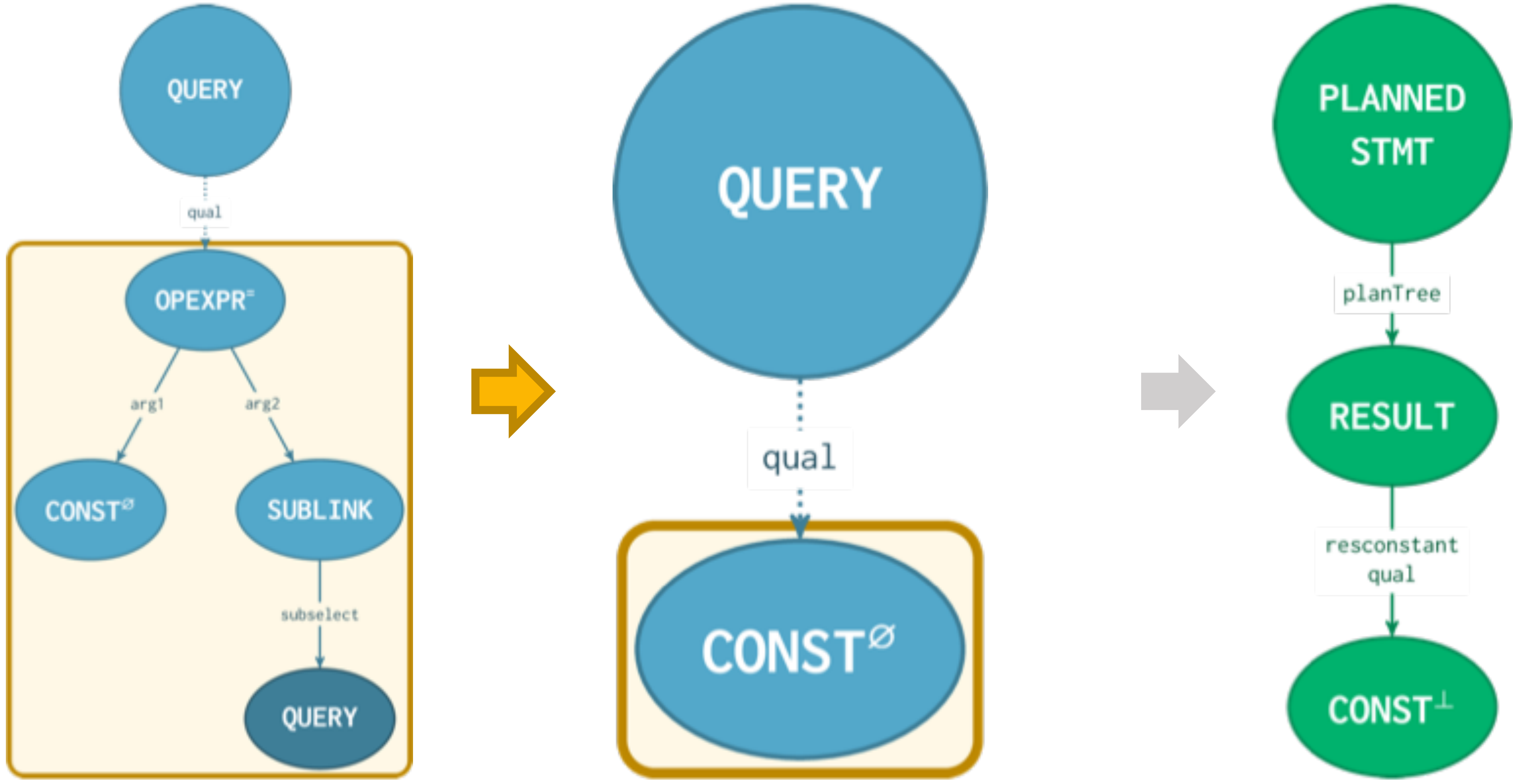


```
# EXPLAIN SELECT a FROM foo WHERE NULL = (SELECT b FROM bar);
```

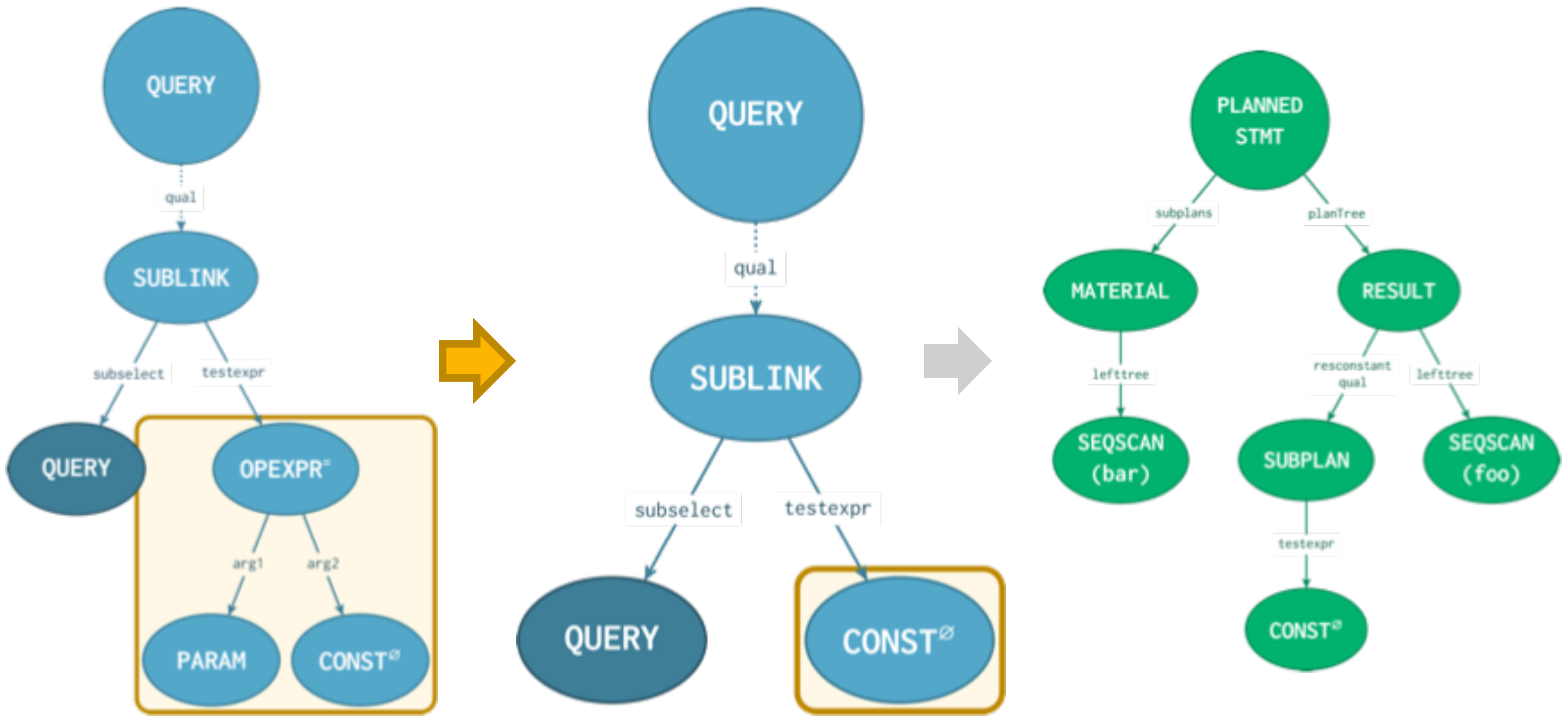
```
          QUERY PLAN
─────────────────────────────
Result
  One-Time Filter: false
```



```
# SELECT a FROM foo WHERE NULL = (SELECT b FROM bar);
```

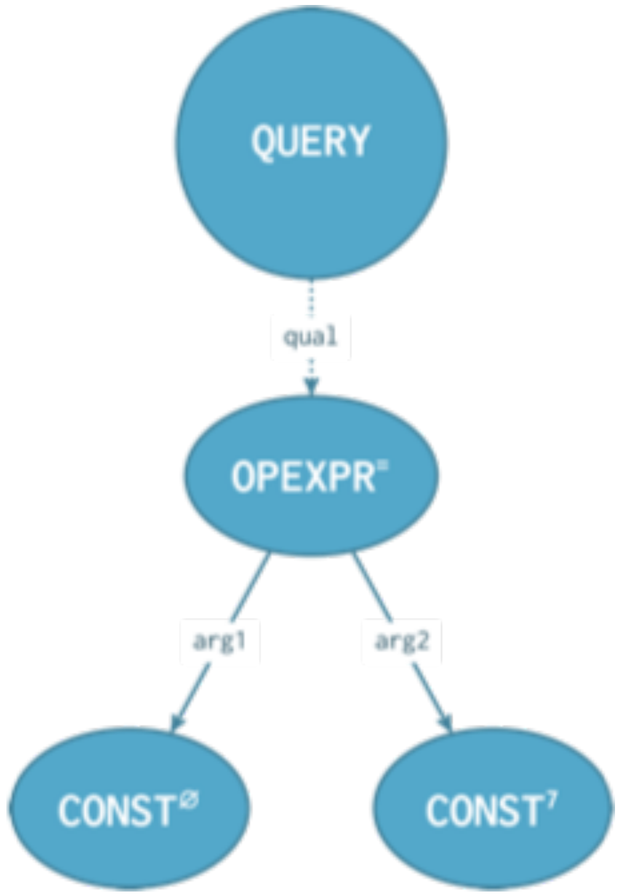


```
# SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
```

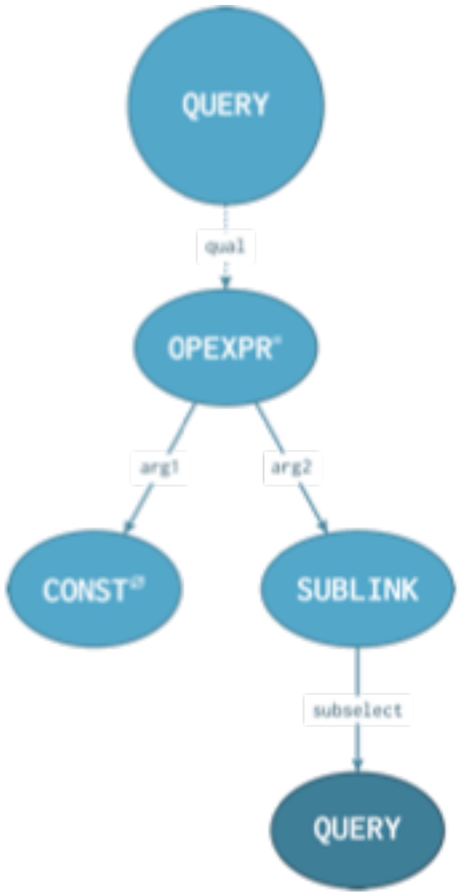


# SELECT a FROM foo WHERE …

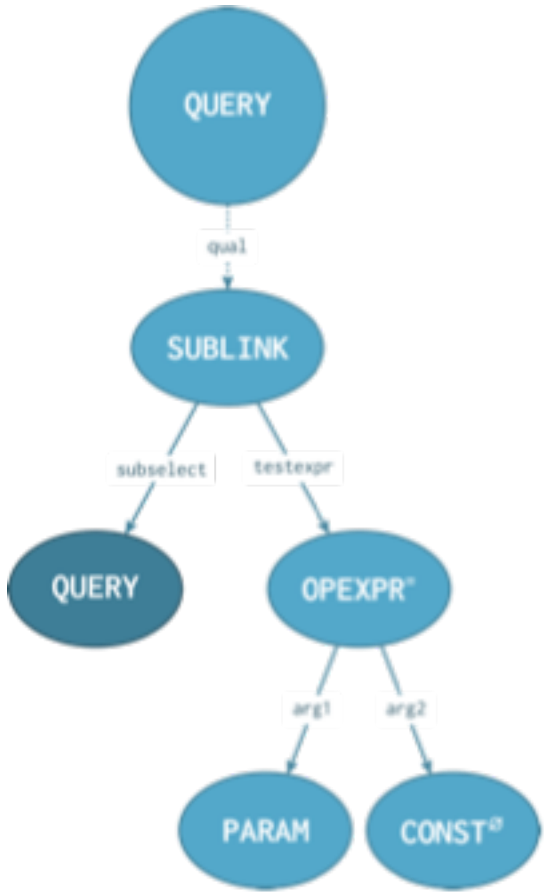
NULL = 7



NULL = (SELECT b FROM bar)



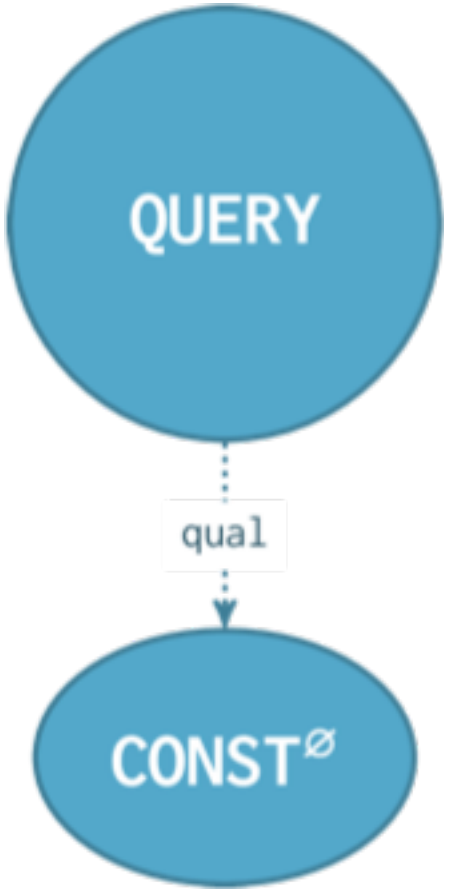
NULL = ANY(SELECT b FROM bar)



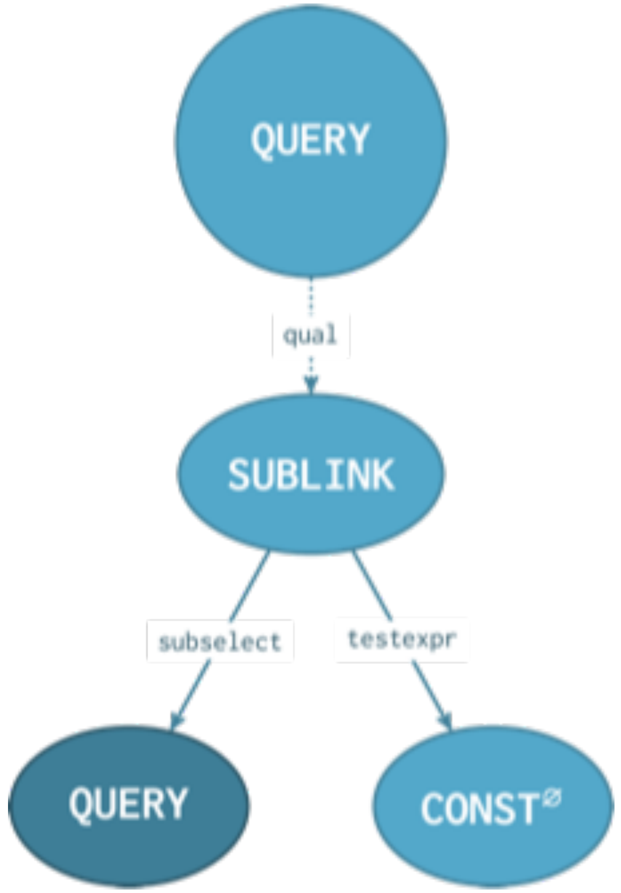
# SELECT a FROM foo WHERE …

NULL = 7

NULL = (SELECT b FROM bar)

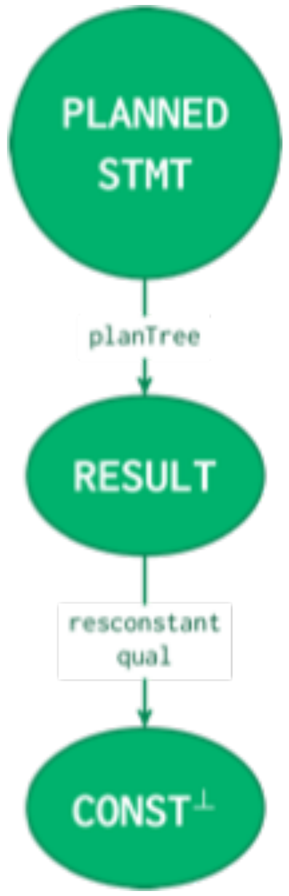


NULL = ANY(SELECT b FROM bar)

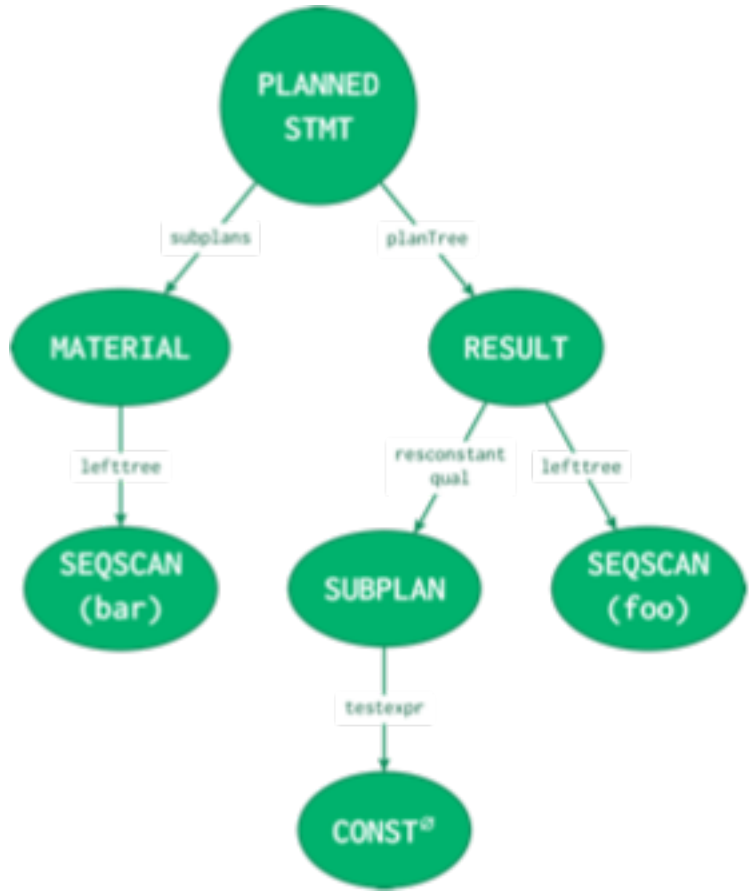


# EXPLAIN SELECT a FROM foo WHERE …

```
NULL = 7
NULL = (SELECT b FROM bar)
```



```
NULL = ANY(SELECT b FROM bar)
```



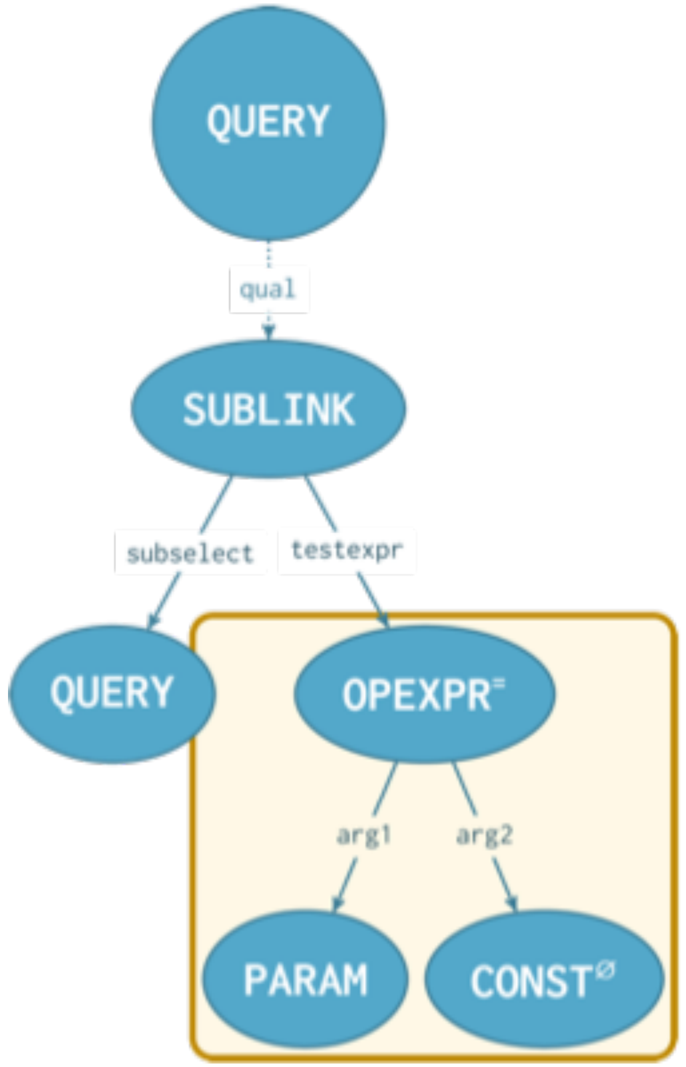
Two 💡s

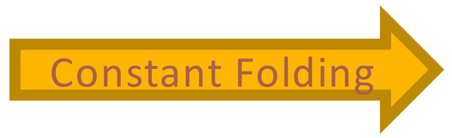
Constant Folding

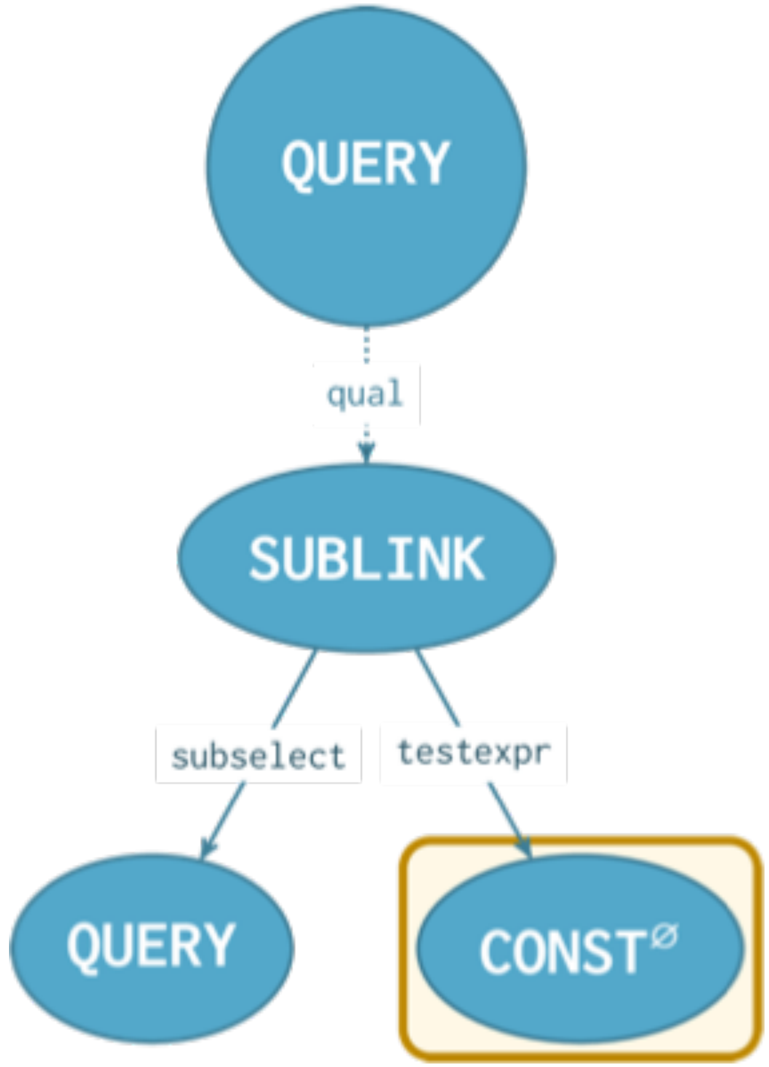
ANY Sublink Pullup

```
SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
```

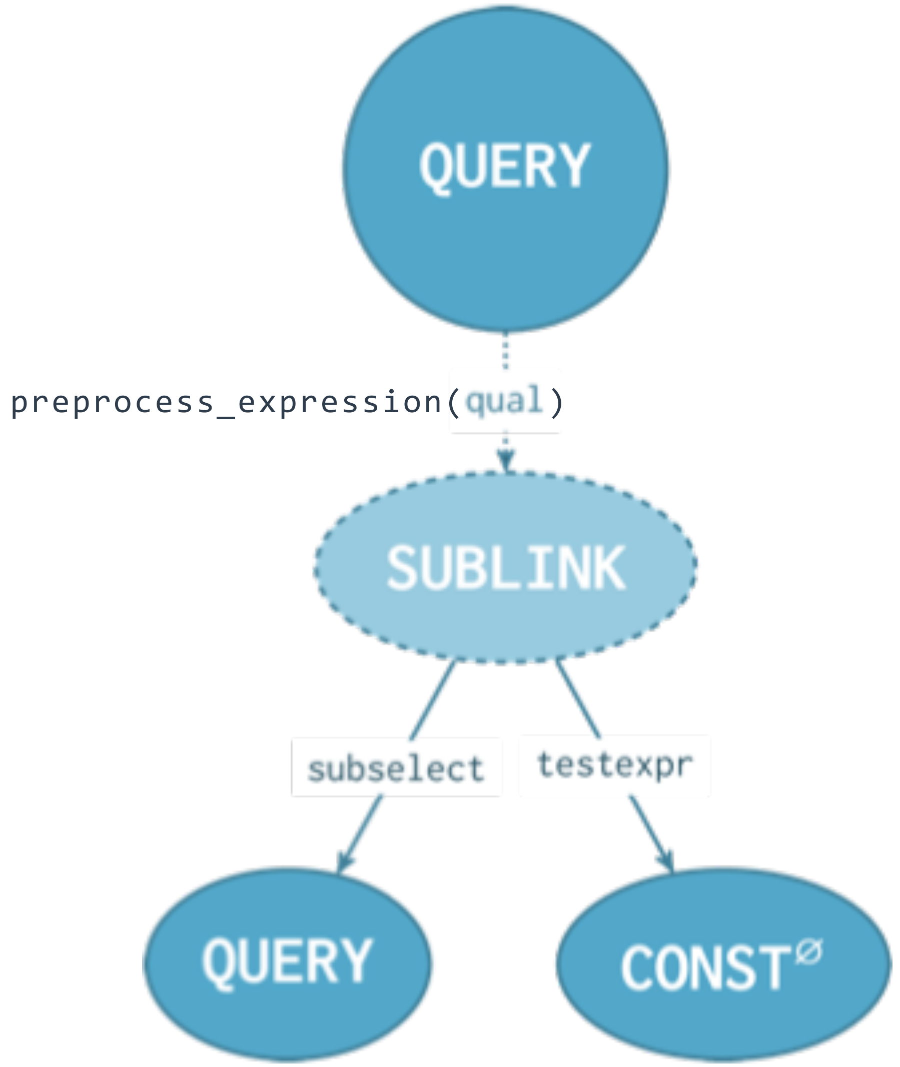
Current Pre-processed Query Tree

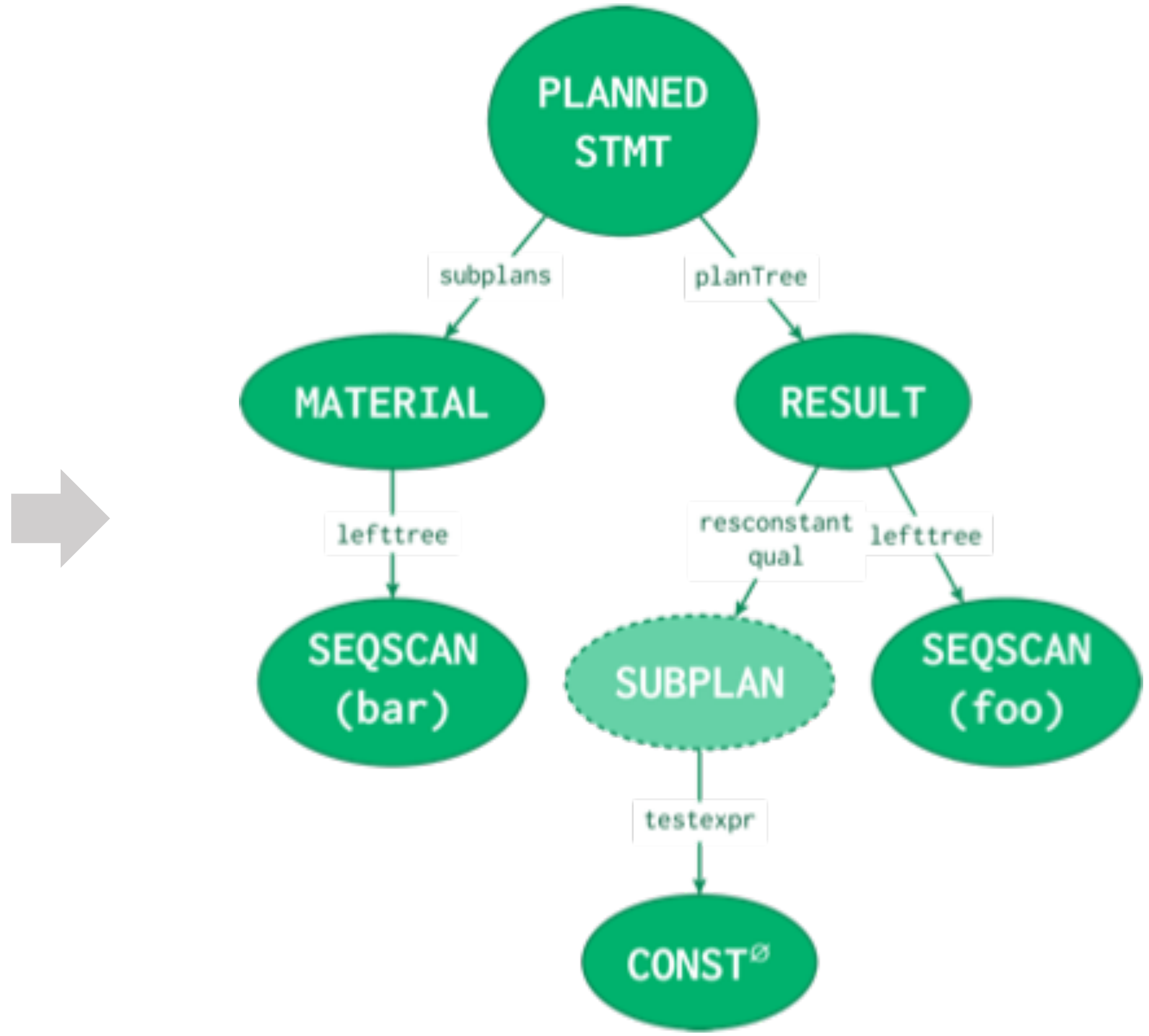






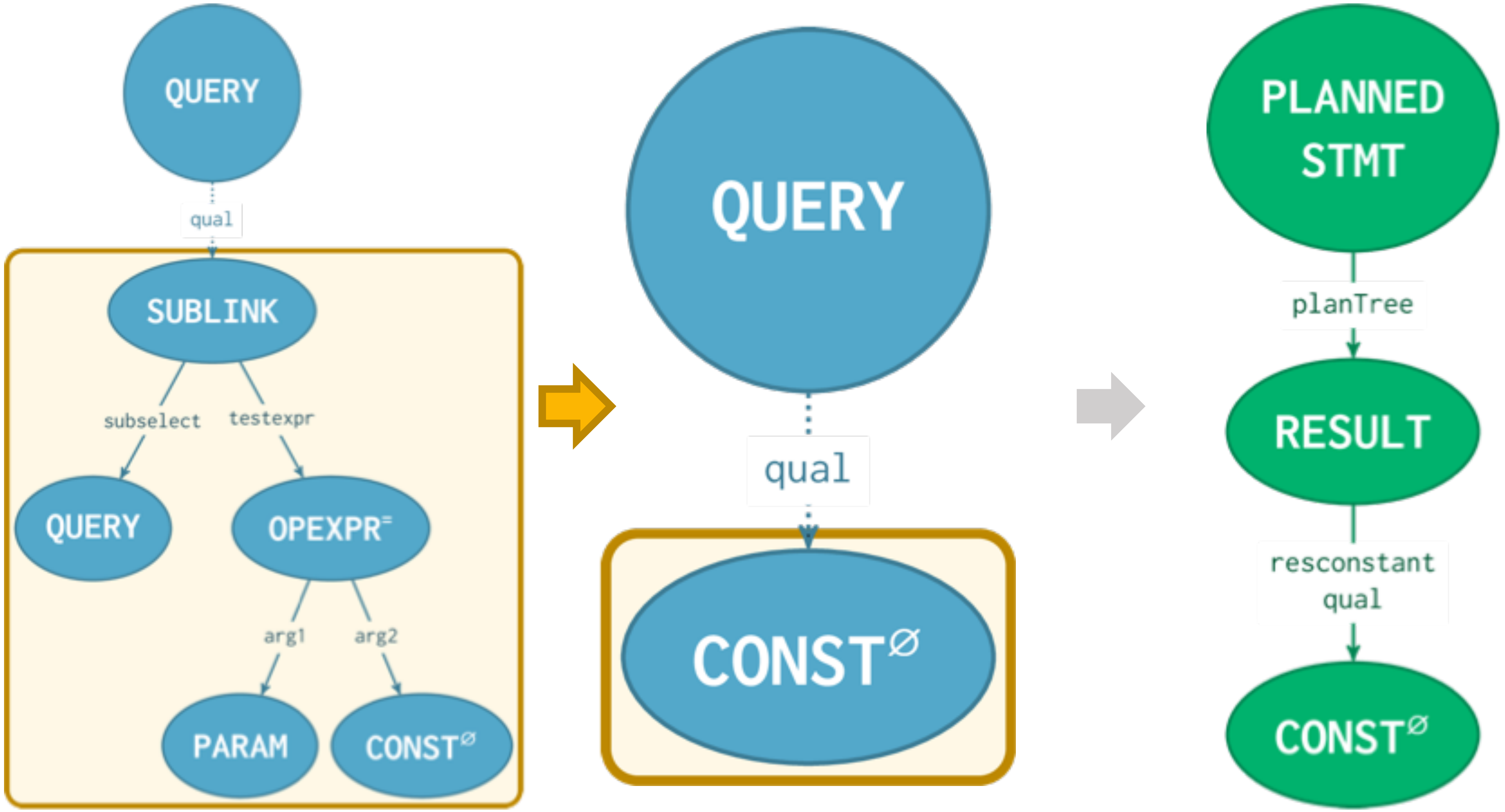
```
# SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
```





```
# SELECT a FROM foo WHERE NULL
  = ANY(SELECT b FROM bar);
```

```
# SELECT a FROM foo WHERE NULL;
```



~~Rule $\boxed{1}$~~

This is semantically incorrect in one case

NULL Semantics

Meet ANY semantics

`NULL` $\overset{?}{=}$ `ANY(SELECT b FROM bar)`

Does any b in bar equal an unknown?

```
# SELECT NULL = ANY(SELECT b FROM bar);
```

Does any b in bar equal an unknown?

Does any b in bar equal an unknown?

```
# SELECT NULL = ANY(SELECT
b FROM bar);

 ?column?
──────────


(1 row)
```

Does any b in bar equal an unknown?

```
# SELECT NULL = ANY(SELECT
b FROM bar);

 ?column?
──────────

(1 row)
```

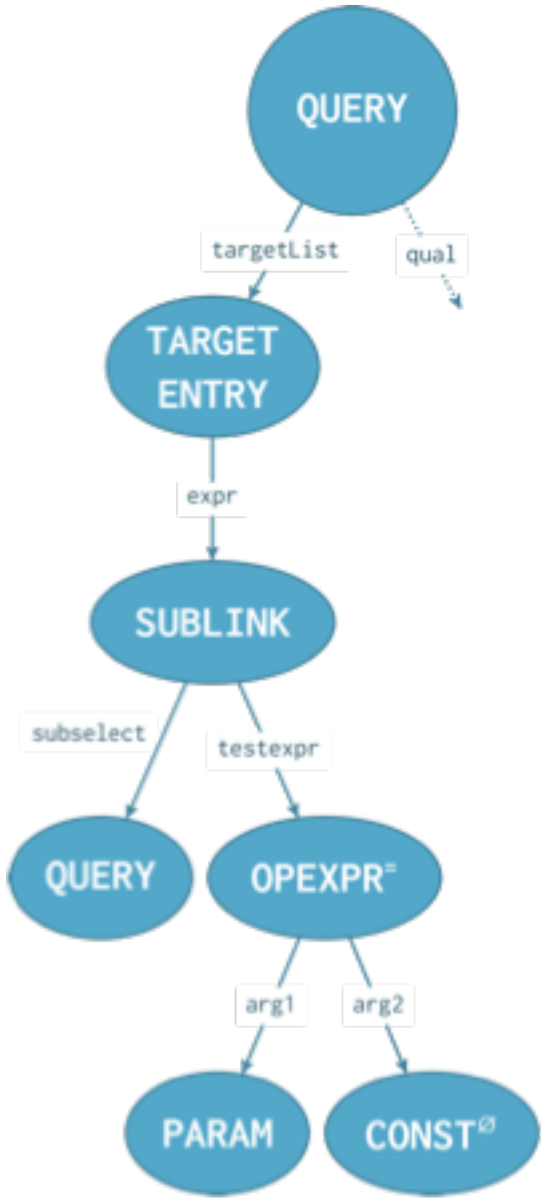
```
# TRUNCATE bar;

# SELECT NULL = ANY(SELECT
b FROM bar);

 ?column?
──────────
 f
(1 row)
```

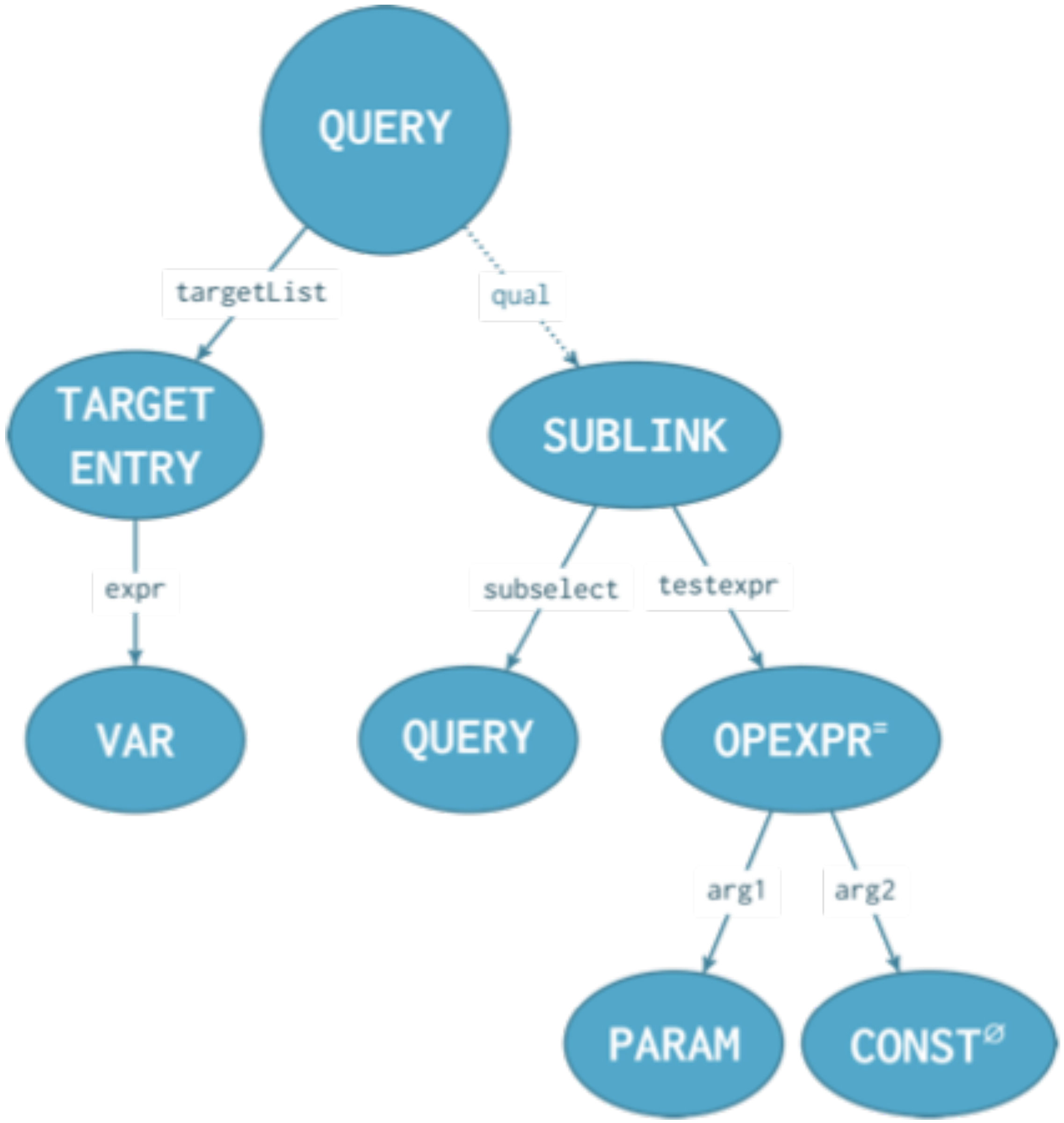
```
# SELECT a FROM foo
  WHERE NULL = ANY(
    SELECT b FROM bar
  );

 a
───

(0 rows)
```

```
# TRUNCATE bar;

# SELECT a FROM foo
  WHERE NULL = ANY(
    SELECT b FROM bar
  );

 a
───

(0 rows)
```

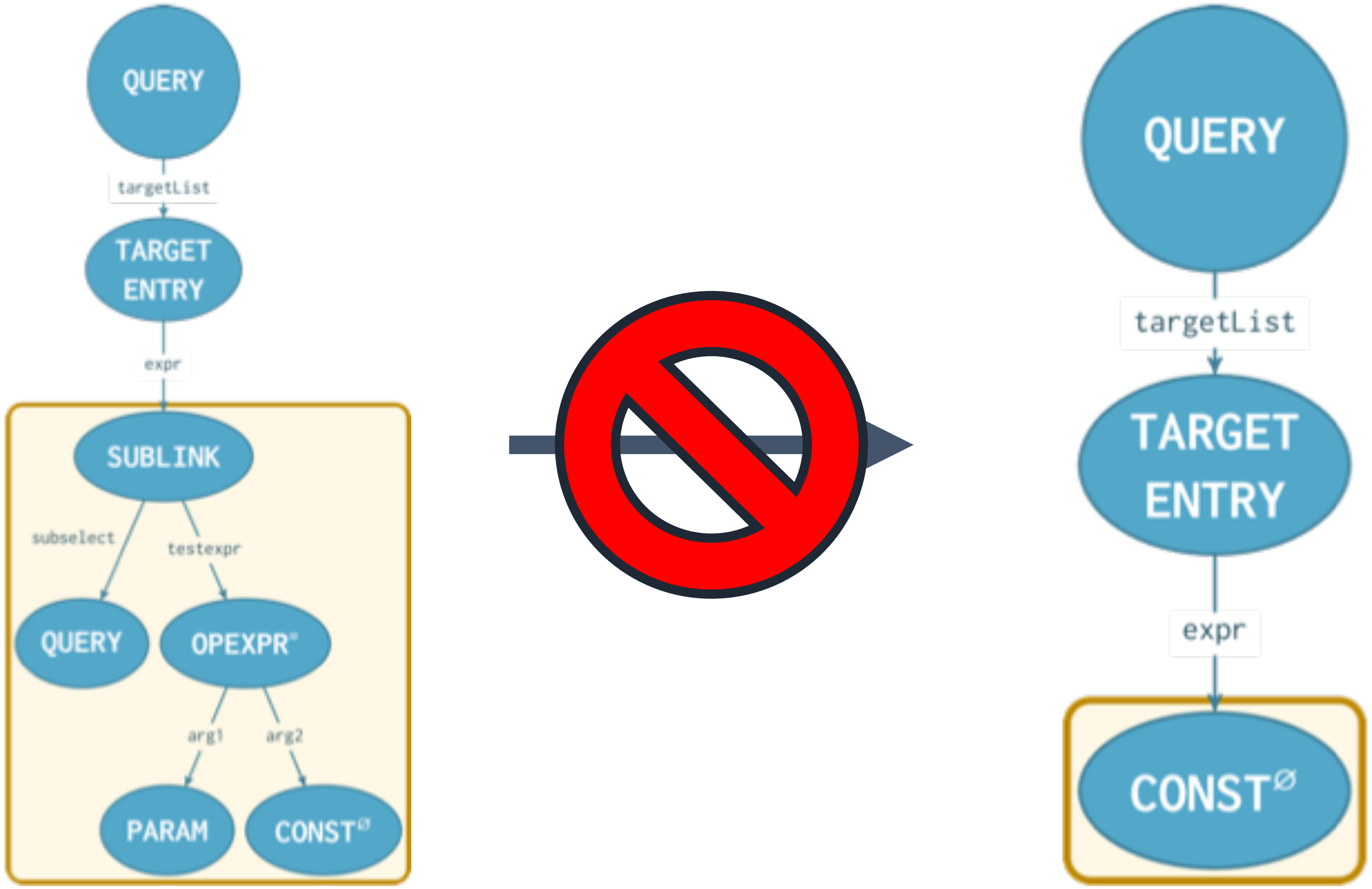
```
# SELECT NULL = ANY(SELECT b FROM bar);
```

```
# SELECT a FROM foo WHERE
  NULL = ANY(SELECT b FROM bar);
```





FALSE if bar is an empty table and NULL otherwise



What could we do instead?

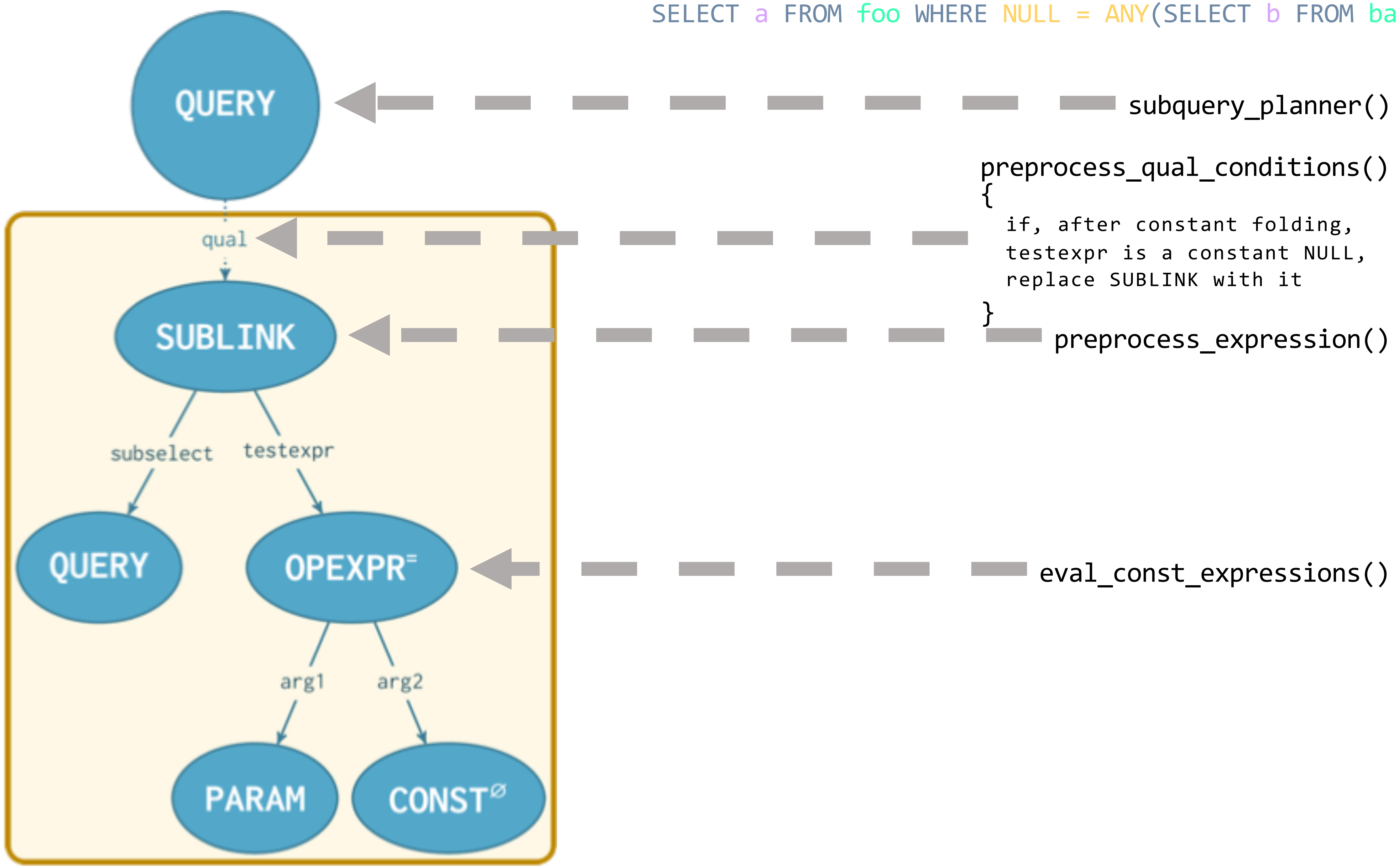
Two 💡s

Constant Folding <u>only in the `qual`</u>

ANY Sublink Pullup

```
# SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
```

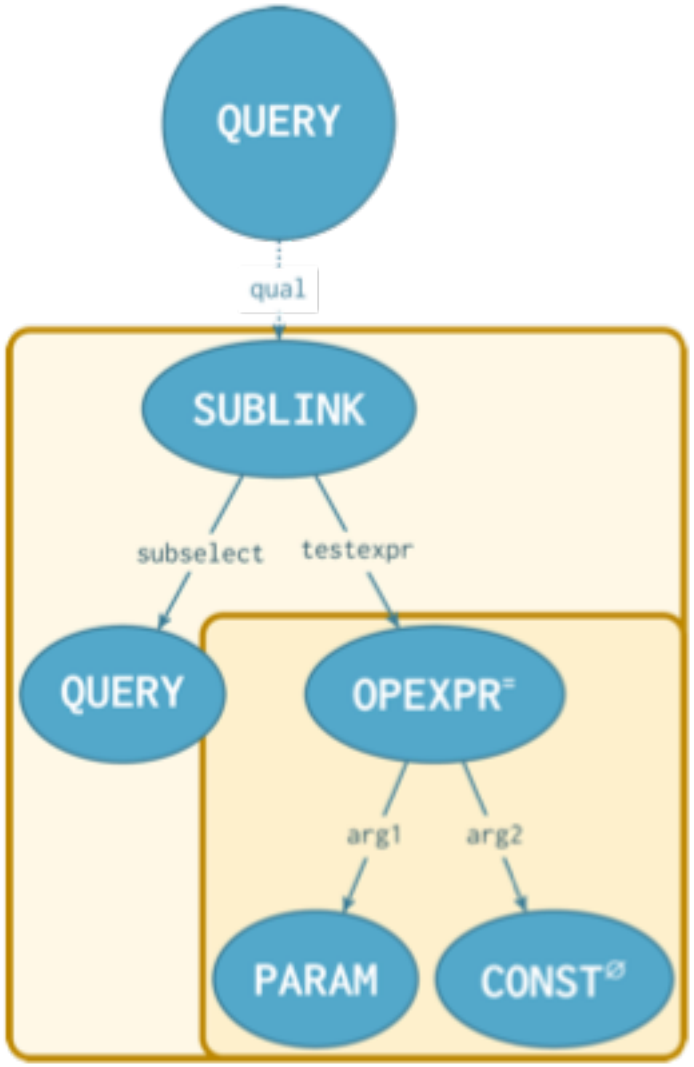
```
SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
```

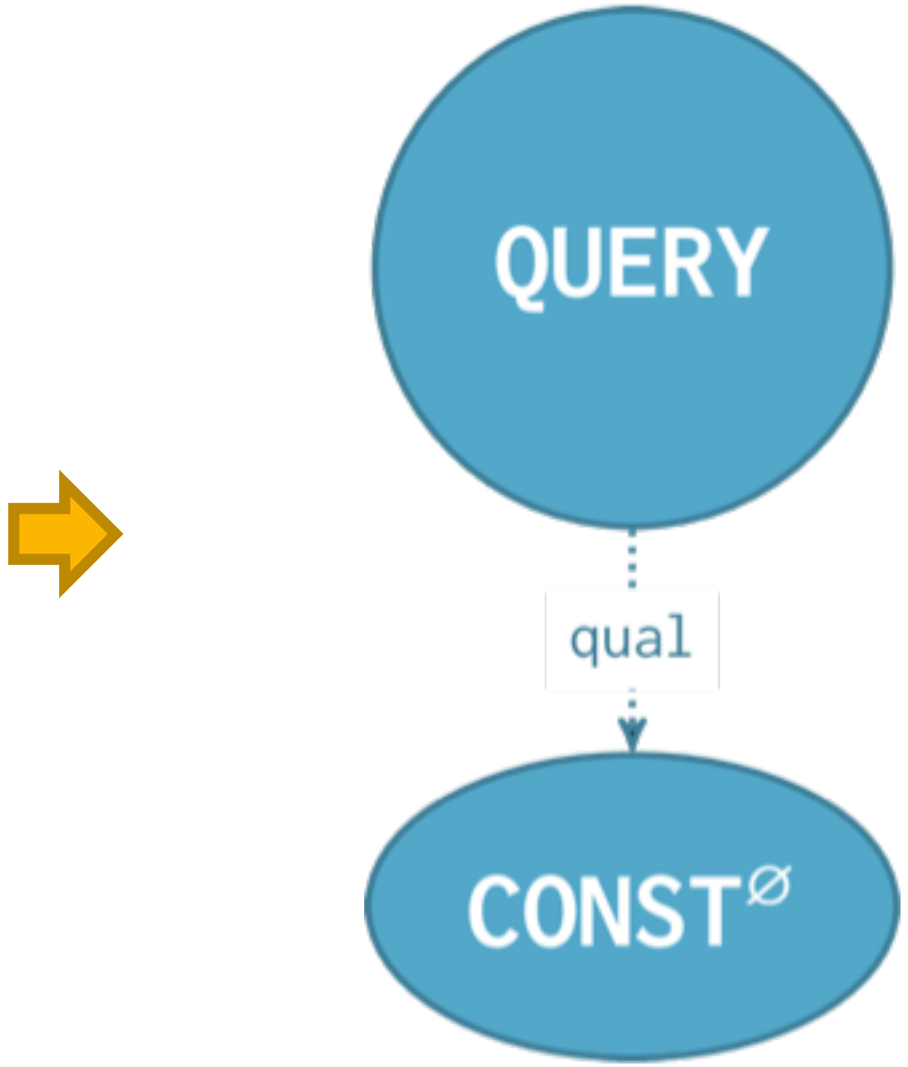


```
preprocess_qual_conditions()
{
  if, after constant folding,
  testexpr is a constant NULL,
  replace SUBLINK with it
}
```

```
SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
```

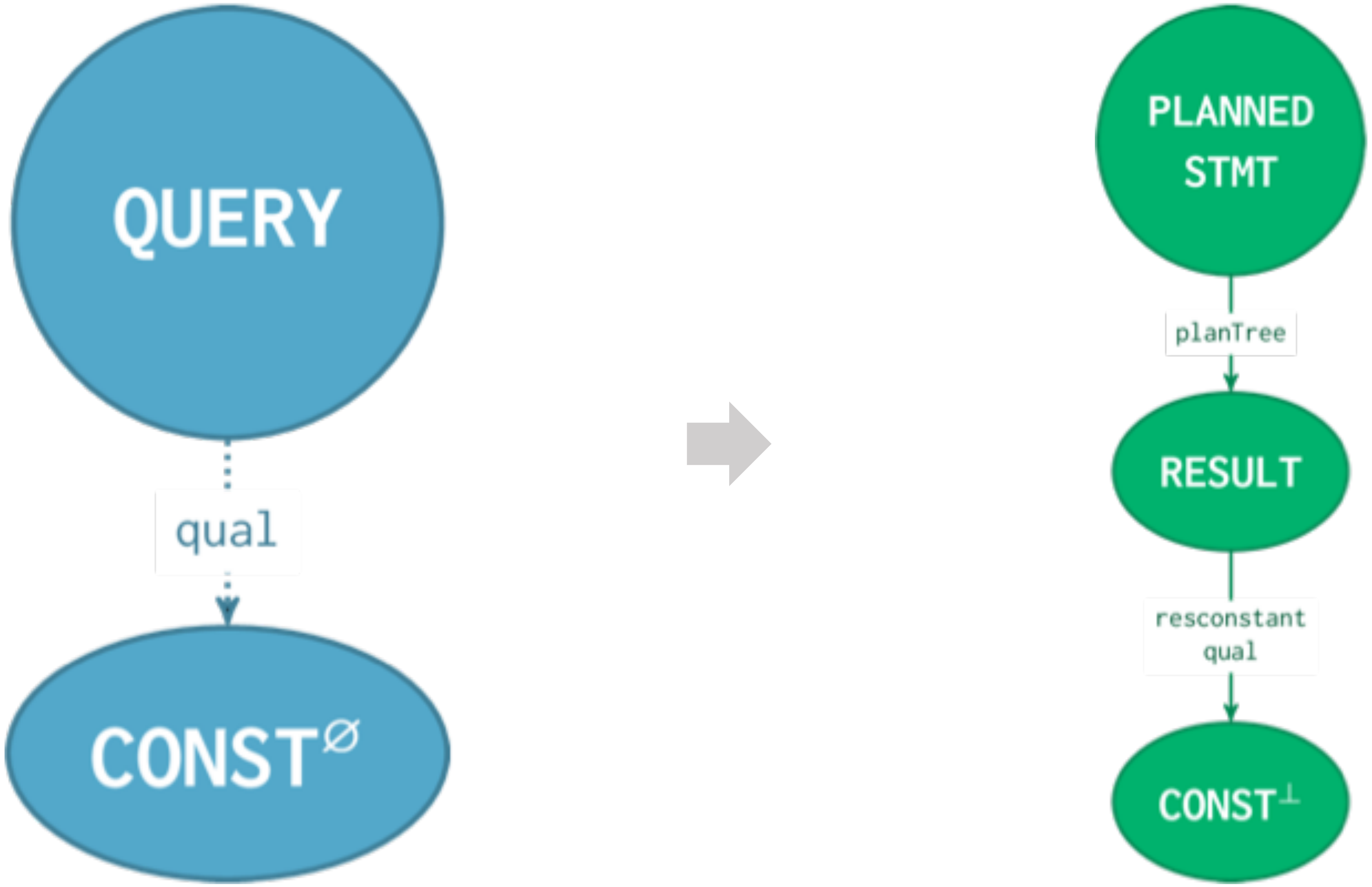
Replace **ANY SUBLINK** when pre-processing **qual**s





```
SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
```

Patched Planning



Patched Plan

```
# EXPLAIN SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);

                    QUERY PLAN
-------------------------------------------------------
Result (cost=… rows=0 width=…)
  One-Time Filter: false
```

~~Rule $\boxed{4}$~~

A very narrow case

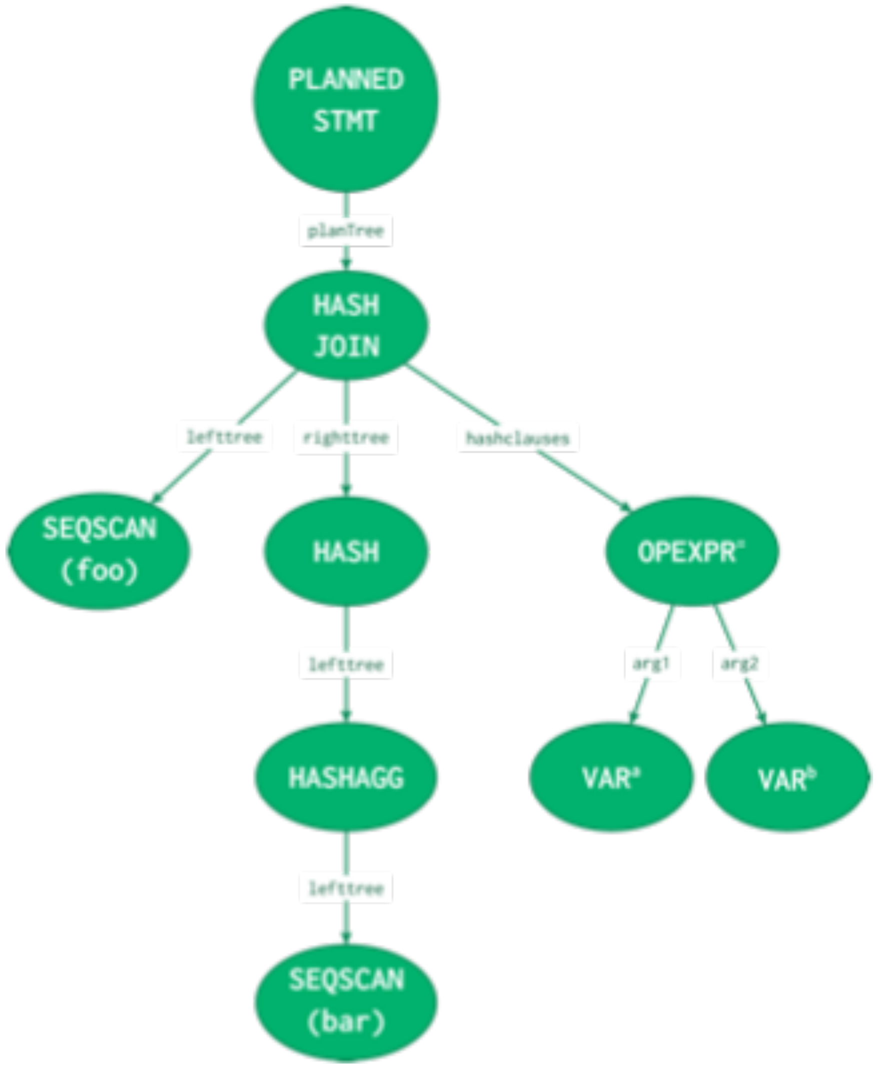
Two 💡s

Constant Folding

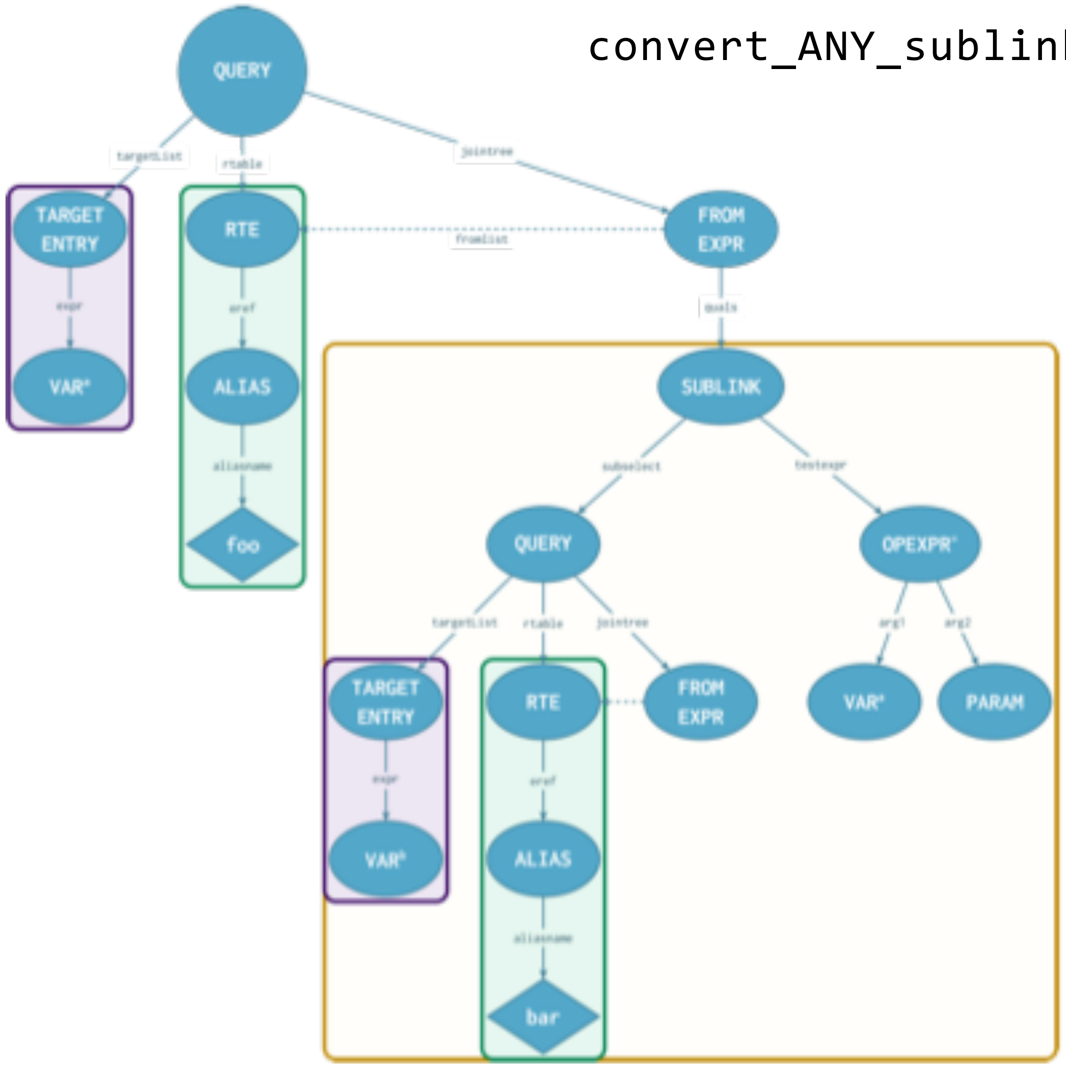
ANY Sublink Pullup

```
# SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
# EXPLAIN SELECT a FROM foo WHERE a = ANY(SELECT b FROM bar);
```

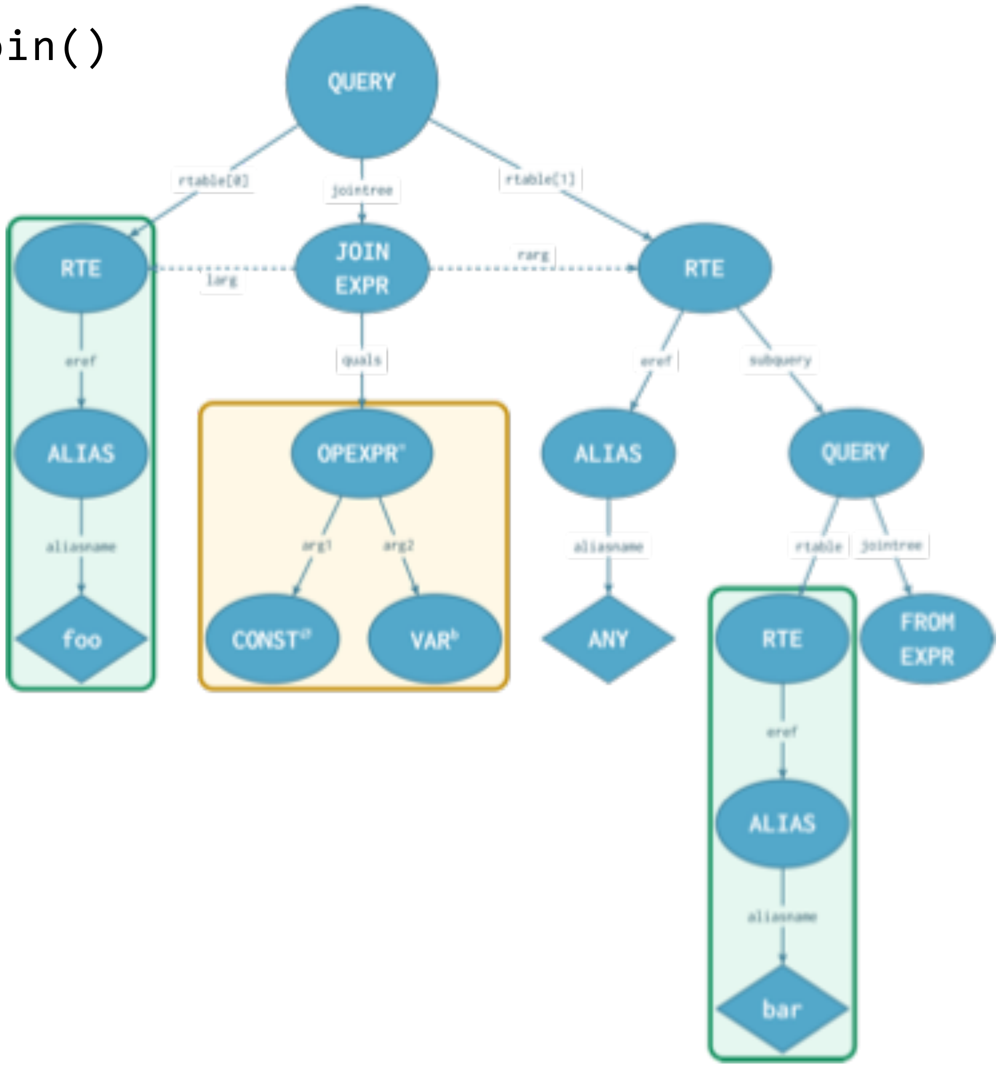
```
           QUERY PLAN
─────────────────────────────
Hash Join
  Hash Cond: (foo.a = bar.b)
  → Seq Scan on foo
  → Hash
    → HashAggregate
        Group Key: bar.b
        → Seq Scan on bar
```



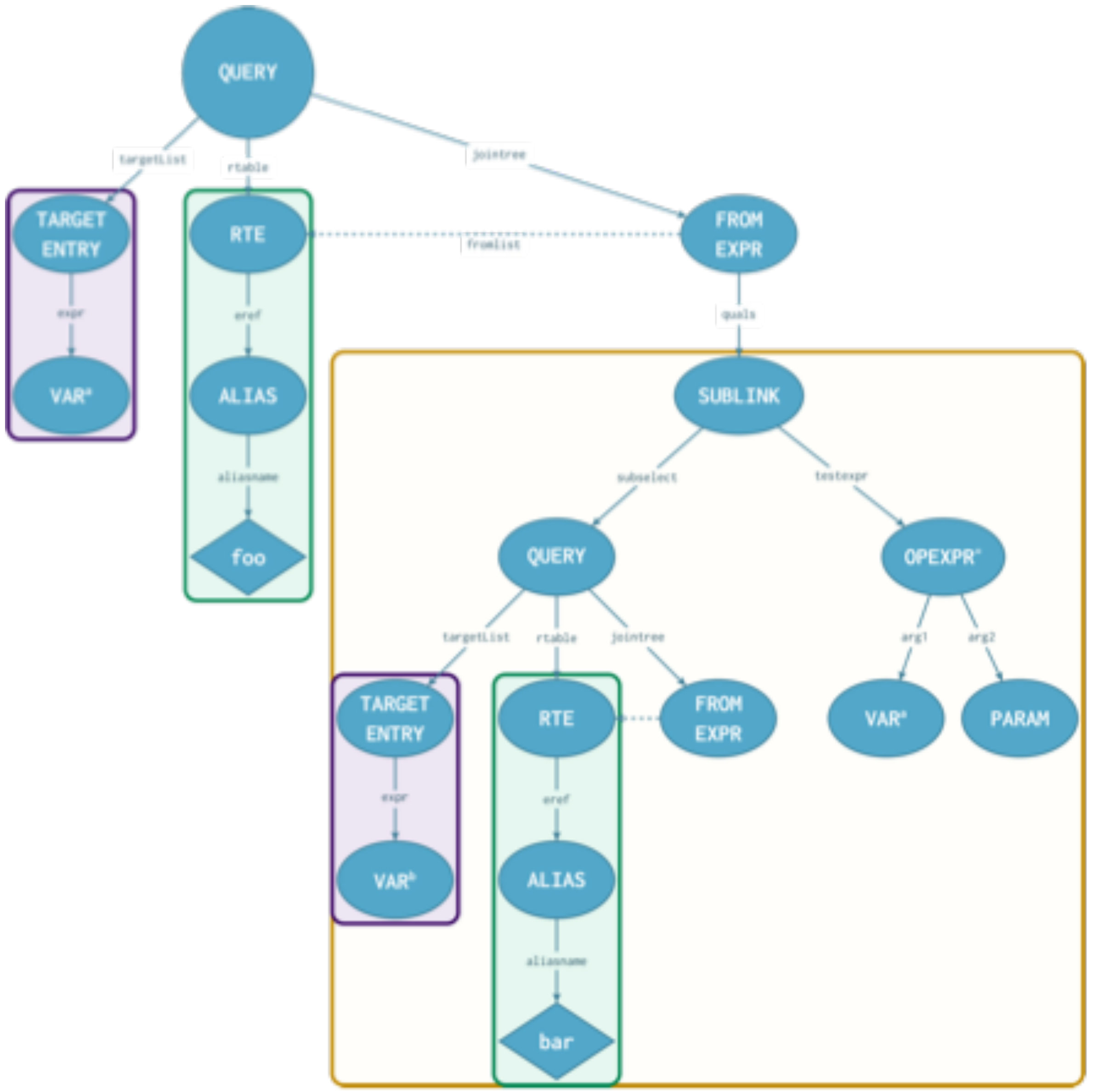
```
# SELECT a FROM foo WHERE a = ANY(SELECT b FROM bar);
```



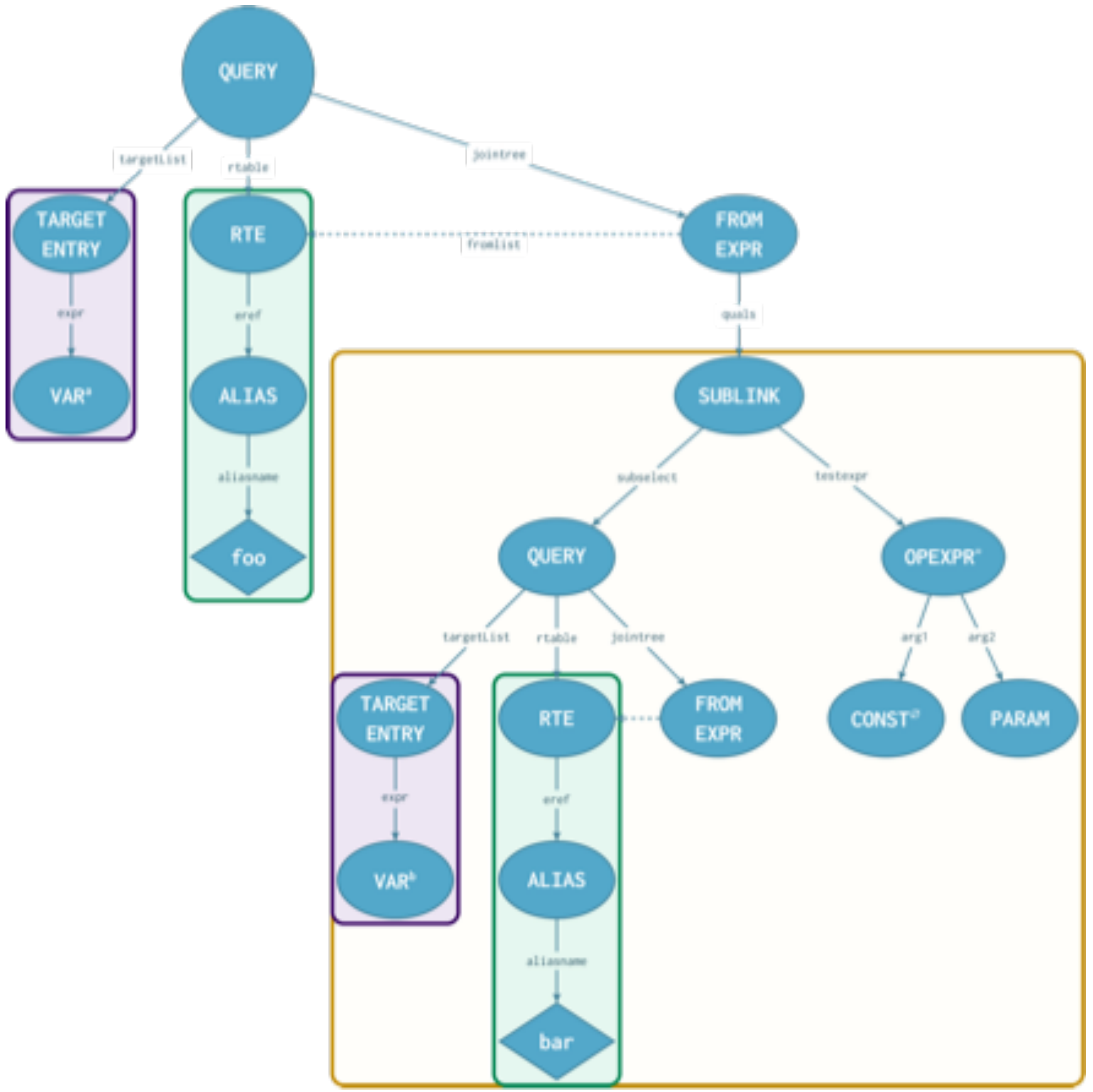
convert_ANY_sublink_to_join()



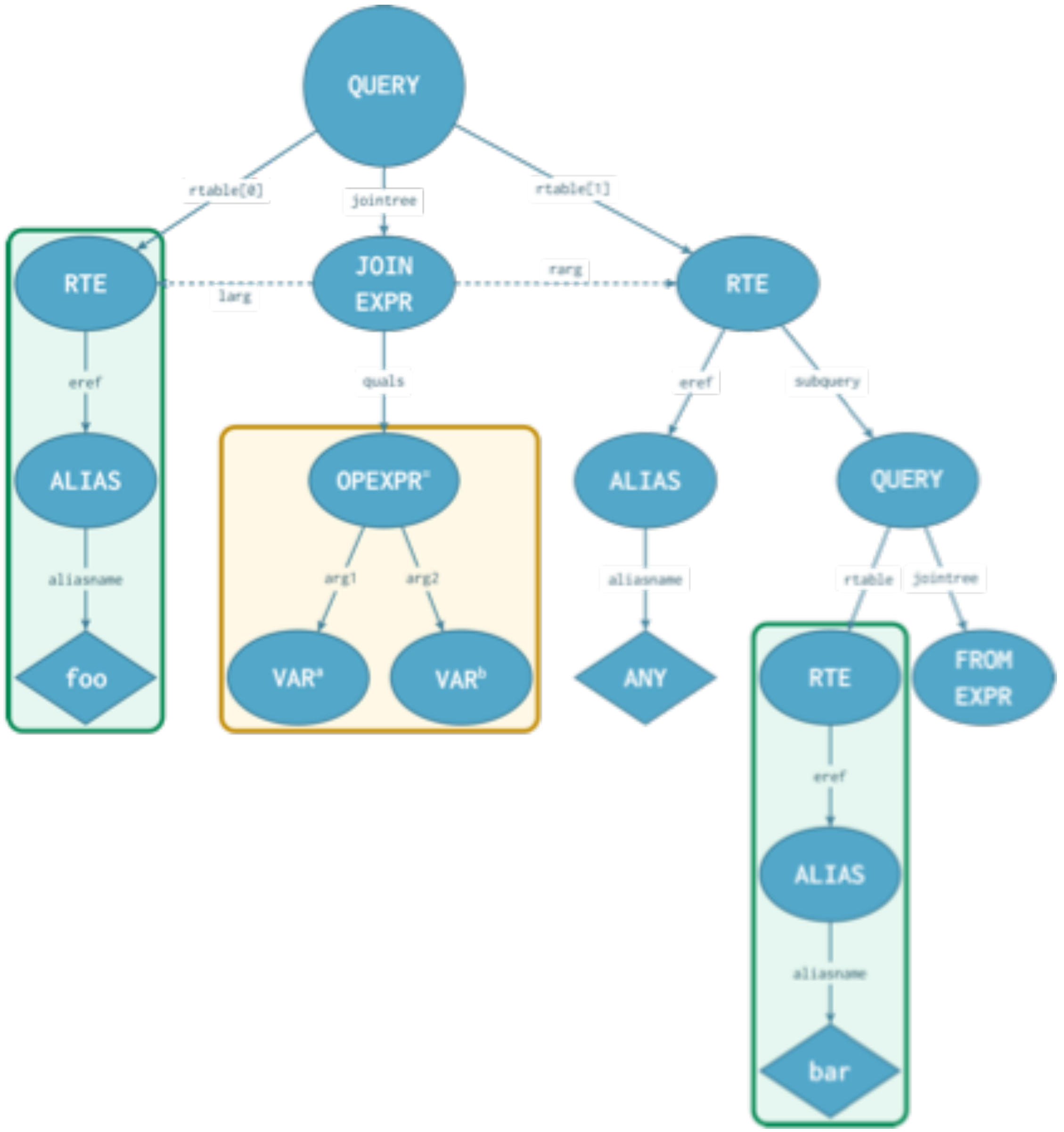
```
# … a = ANY(SELECT b FROM bar);
```

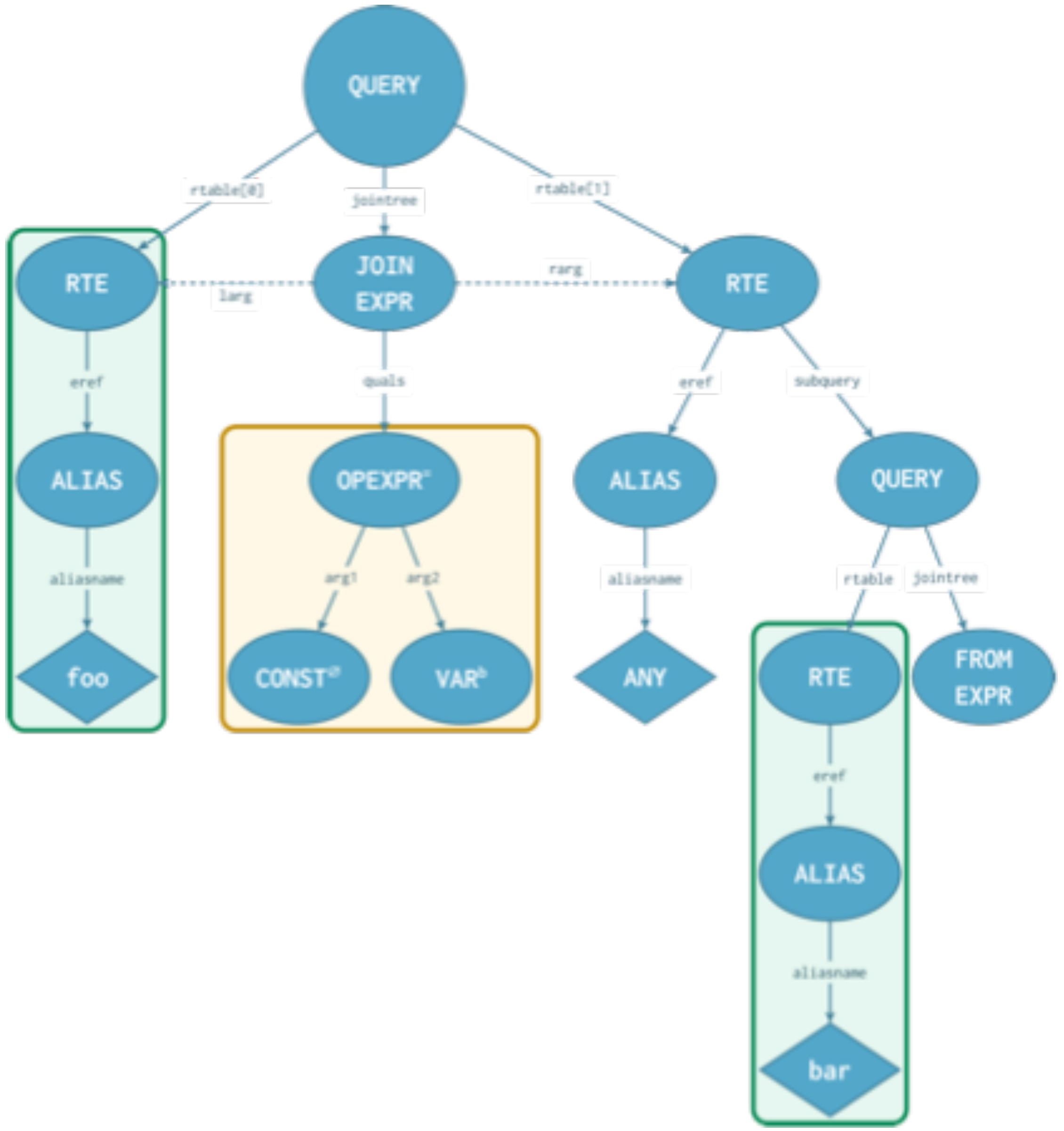


```
# … NULL = ANY(SELECT b FROM bar);
```

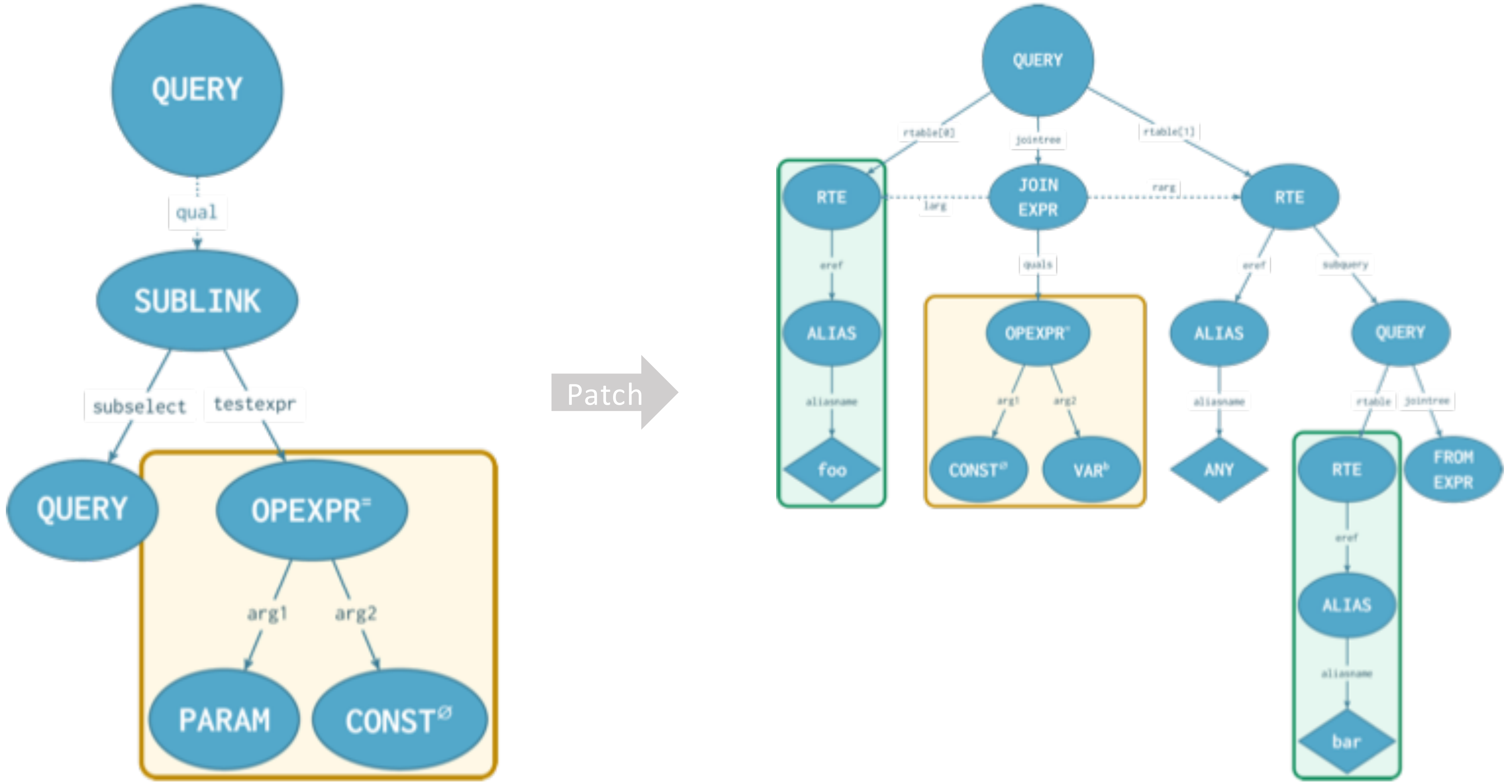


```
# … a = ANY(SELECT b FROM bar);        # … NULL = ANY(SELECT b FROM bar);
```



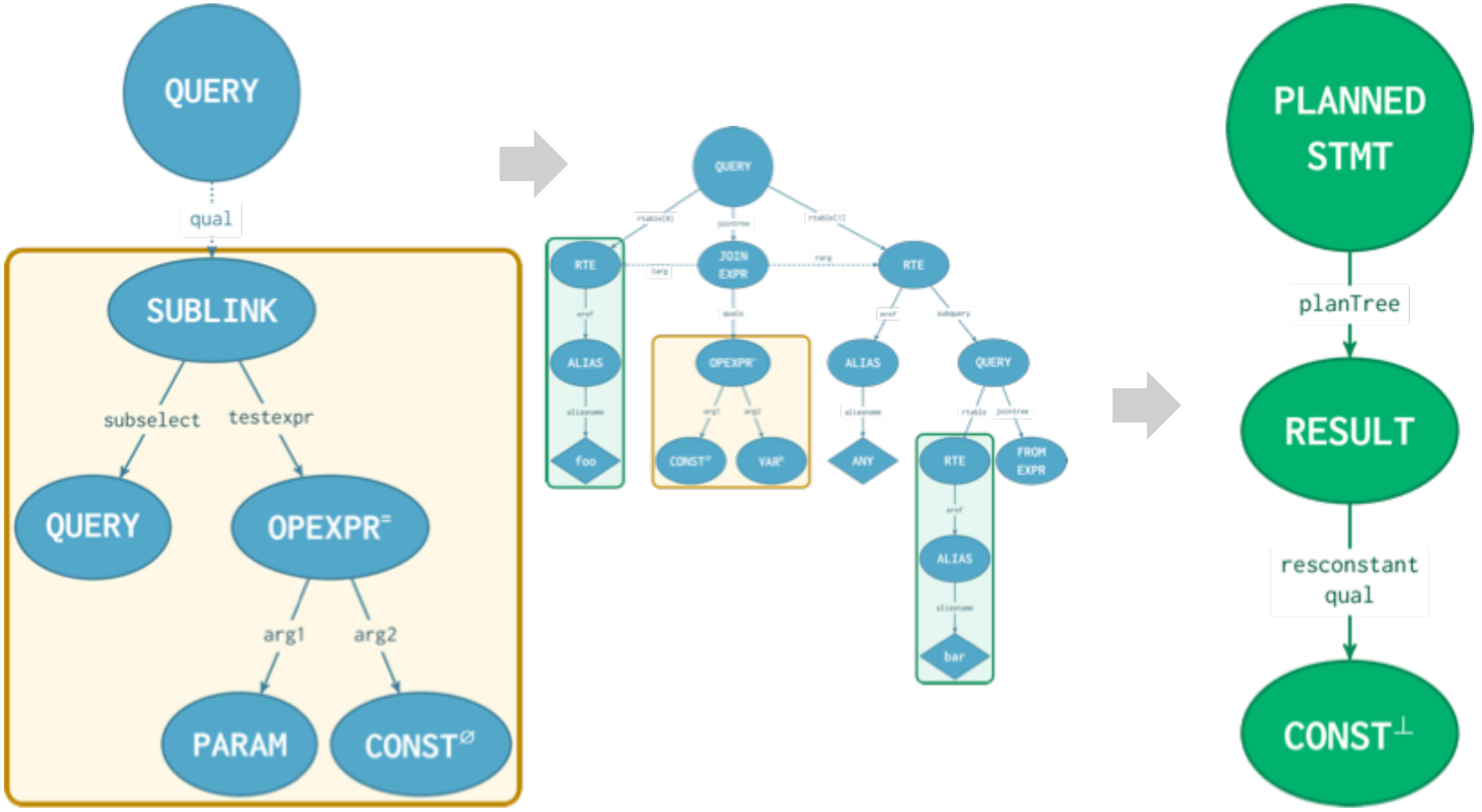


```
# SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);
```



```
# SELECT a FROM foo WHERE NULL
  = ANY(SELECT b FROM bar);
```

```
# SELECT a FROM foo JOIN bar
  WHERE NULL = deduped(b);
```



```
# EXPLAIN SELECT a FROM foo WHERE 7 = ANY(SELECT b FROM bar WHERE b = 5);
```

Current

```
                 QUERY PLAN
─────────────────────────────────────
Result
  One-Time Filter: (hashed SubPlan 1)
  → Seq Scan on foo
  SubPlan 1
  → Seq Scan on bar
       Filter: (b = 7)
```

Patched

```
                 QUERY PLAN
─────────────────────────────────────
Result
  One-Time Filter: false
```

```
# EXPLAIN SELECT a FROM foo WHERE 7 = ANY(SELECT b FROM bar);
```

Current

```
                  QUERY PLAN
─────────────────────────────────────────
Result
  One-Time Filter: (hashed SubPlan 1)
  → Seq Scan on foo
  SubPlan 1
  → Seq Scan on bar
```

Patched

```
                  QUERY PLAN
─────────────────────────────────────────
Nested Loop Semi Join
  → Seq Scan on foo
  → Materialize
    → Seq Scan on bar
        Filter: (7 = b)
```

~~Rules $\boxed{2}$, $\boxed{4}$~~

Produces worse plans when the join isn’t eliminated

A very narrow case

Guidelines for New Optimizations

① Does it always retain semantic correctness?
② Does it inhibit downstream optimizations?
③ Is the improvement in execution time worth the cost in planning time?
④ Is the complexity cost commensurate with the performance benefit?

Some Rejected 💡s

- Use stats
- Execute the subquery

🎤 Discussion 🎤

When is it okay to …?

- Do a catalog lookup
- Do partial execution
- Mutate the plan tree
- Save a reference to parent query

Guidelines … Others?

① Does it always retain semantic correctness?

② Does it inhibit downstream optimizations?

③ Is the improvement in execution time worth the cost in planning time?

④ Is the complexity cost commensurate with the performance benefit?

(Re)sources

- Uncommitted planner patches and discussion (browse old commitfests) https://commitfest.postgresql.org/
- Planner hacking presentations
  - Tom Lane PGCon 2011 Hacking the Query Planner https://www.pgcon.org/2011/schedule/attachments/188_Planner%20talk.pdf
  - Robert Haas (CTRL-F ‘planner’) https://sites.google.com/site/robertmhaas/presentations/2010-2012
- `src/backend/optimizer/README`

github.com/melanieplageman/

↳ /debugging_planner	Slides and Glossary
↳ /postgres/tree/	Code
↳ /const_folding_sublink_wrong	Constant Folding
↳ /qual_scoped_const_folding_sublink	Constant Folding only in the qual
↳ /const_ANY_sublink_pullup	ANY Sublink Pullup

Acknowledgements

Jesse Zhang – Queries and content assistance

Kaiting Chen—TikZ diagram designer