

# Debugging the Postgres Planner

Melanie Plageman

# Goals & Agenda

- Demystify PostgreSQL planner
- Query optimization concepts
- Case study: adding a planner improvement

# Query Planning

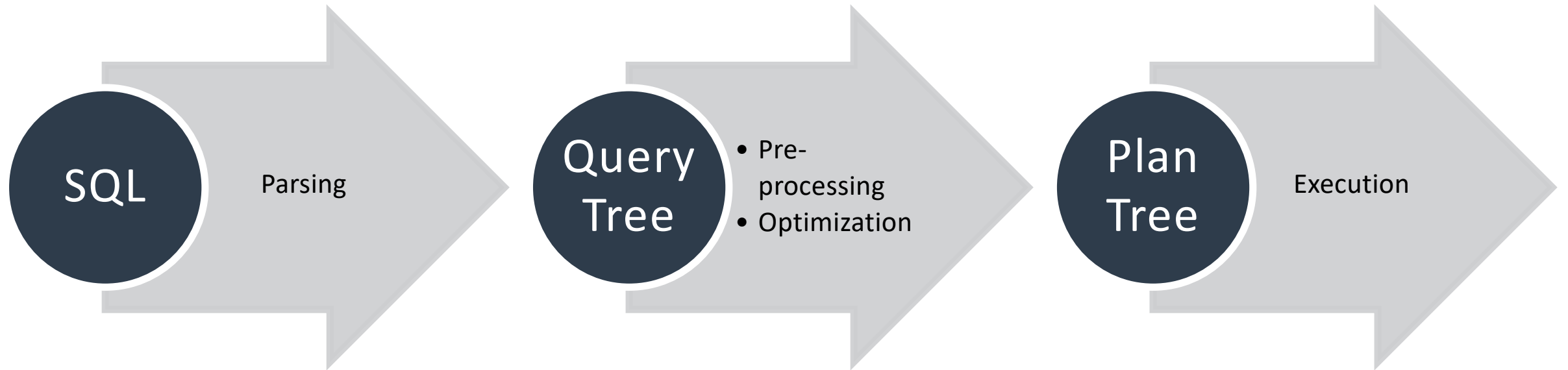
SQL statement to plan tree

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



a
1
2
4

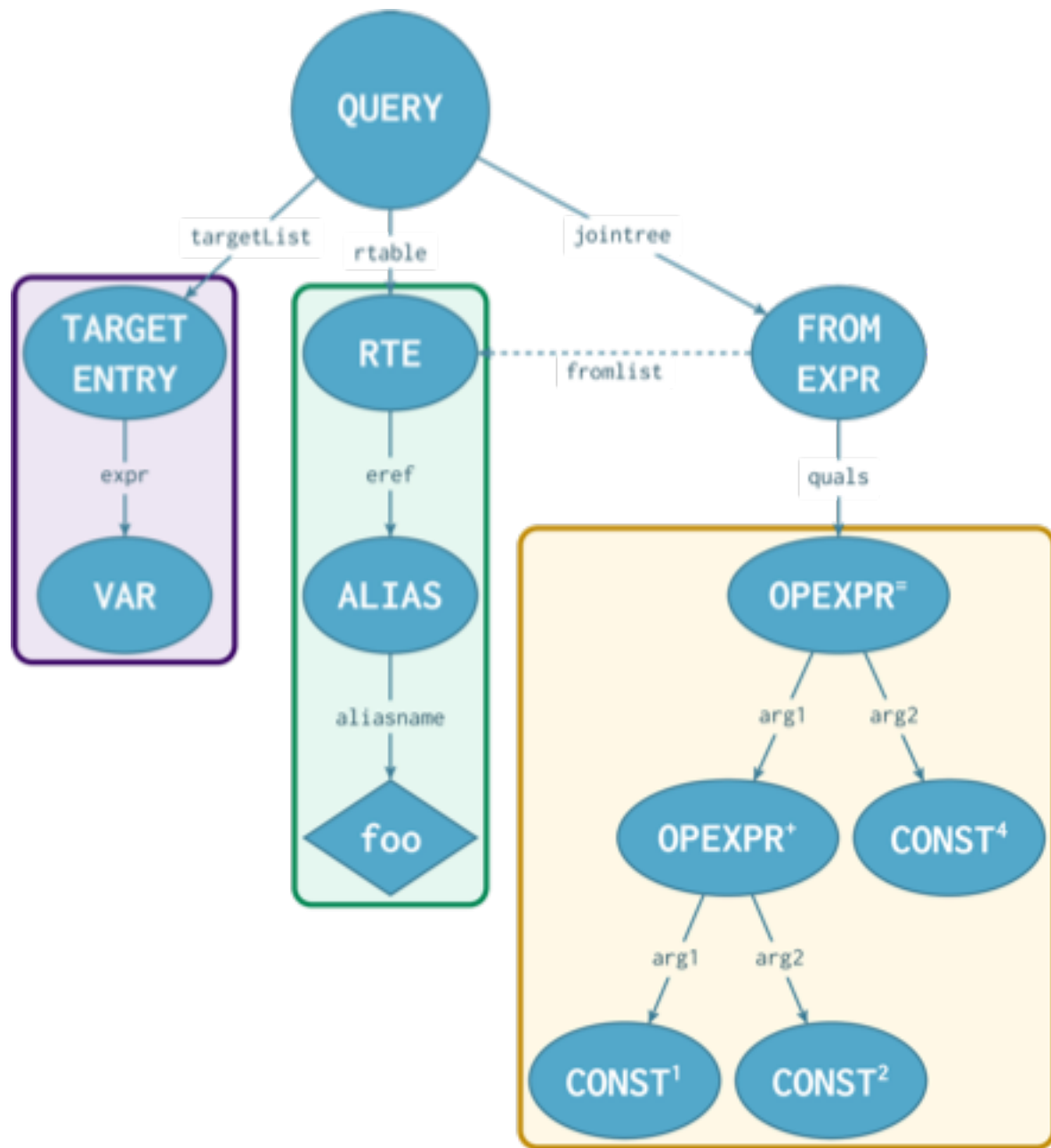
(3 rows)



# Parsing

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

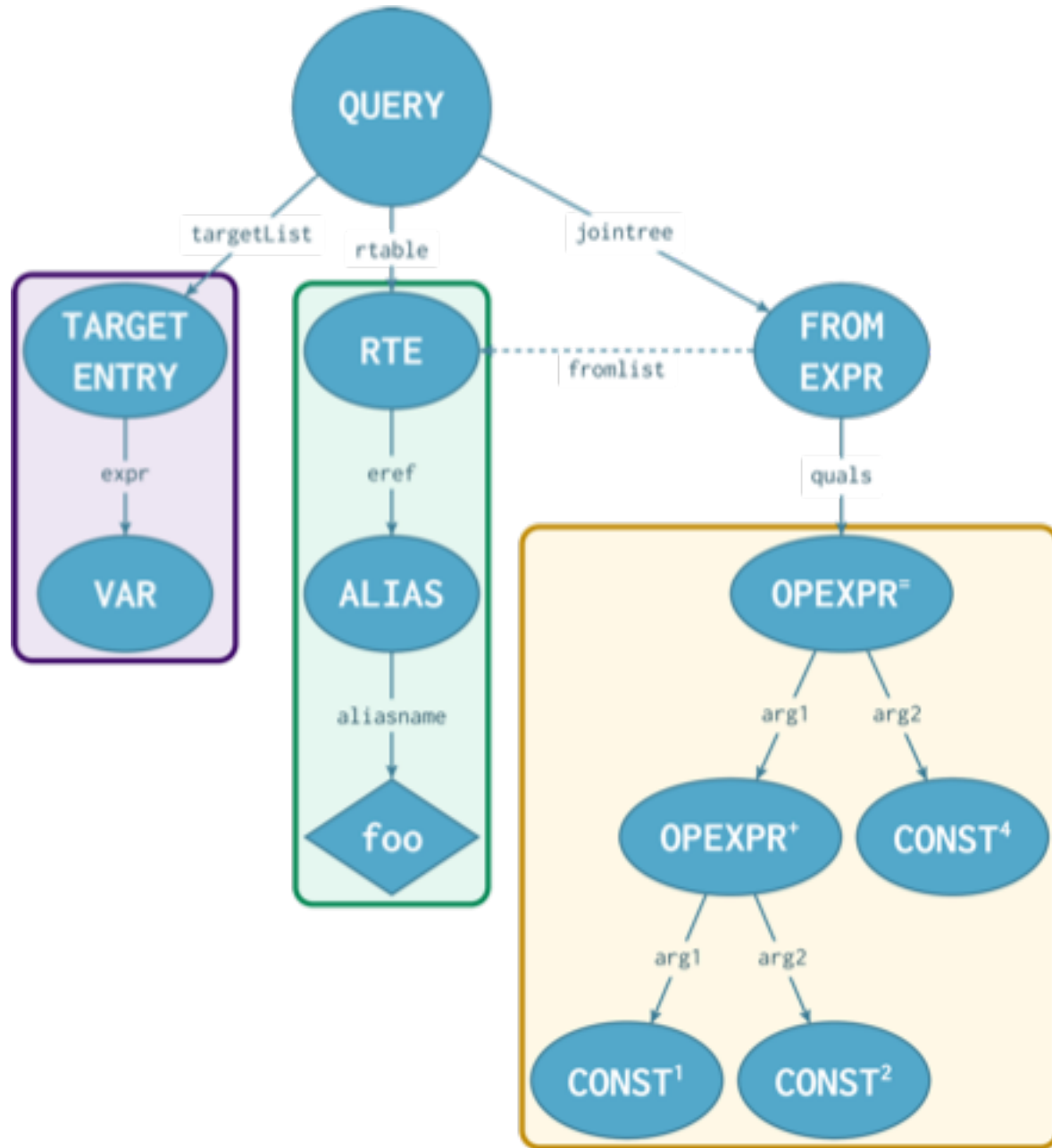
```
{QUERY
  :rtable (
    {RTE
      :eref
        {ALIAS
          :aliasname foo
          :colnames ("a")
        }
      :jointree
        {FROMEXPR
          :quals
            {OPEXPR
              :args (
                {OPEXPR
                  :args (
                    {CONST
                      :constvalue 4 [ 1 0 0 0 0 0 0 0 0 ]
                    }
                    {CONST
                      :constvalue 4 [ 2 0 0 0 0 0 0 0 0 ]
                    }
                    {CONST
                      :constvalue 4 [ 4 0 0 0 0 0 0 0 0 ]
                    }
                  )
                }
              :targetList (
                {TARGETENTRY
                  :expr
                    {VAR
                      :resname a
                    }
                }
              )
            }
          }
        }
      )
    }
  )
}
```



```

{QUERY
  :rtable (
    {RTE
      :eref
        {ALIAS
          :aliasname foo
          :colnames ("a")}
      :jointree
        {FROMEXPR
          :quals
            {OEXPR
              :args (
                {OEXPR
                  :args (
                    {CONST
                      :constvalue 4 [ 1 0 0 0 0 0 0 0 ]}
                    {CONST
                      :constvalue 4 [ 2 0 0 0 0 0 0 0 ]}
                    {CONST
                      :constvalue 4 [ 4 0 0 0 0 0 0 0 ]}
                  )
                )
            }
          :targetList (
            {TARGETENTRY
              :expr
                {VAR
                  :resname a}
            }
          )
        }
      )
    }
  )
}

```



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



Pre-processing

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

1 + 2 = 4

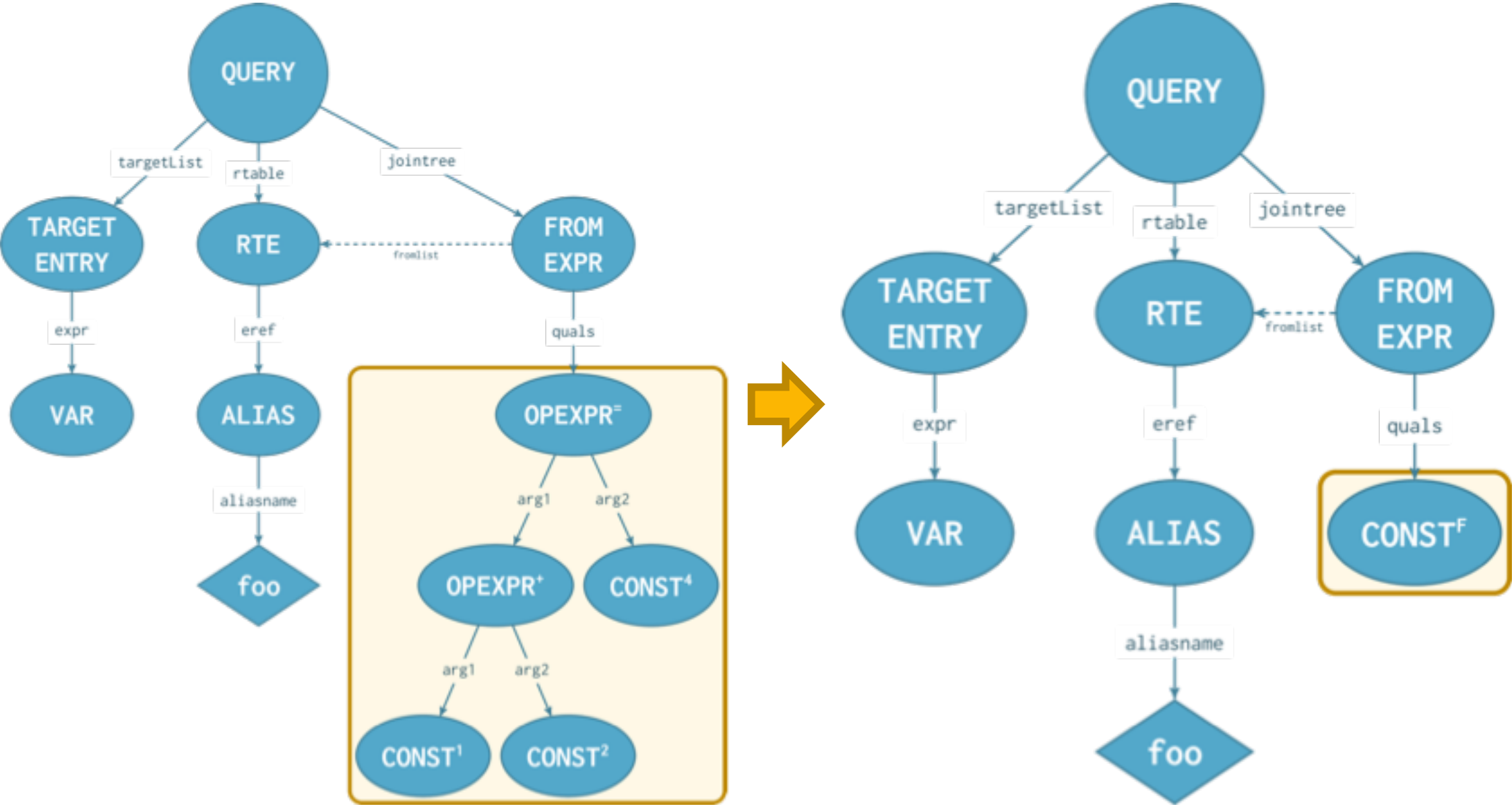


FALSE

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

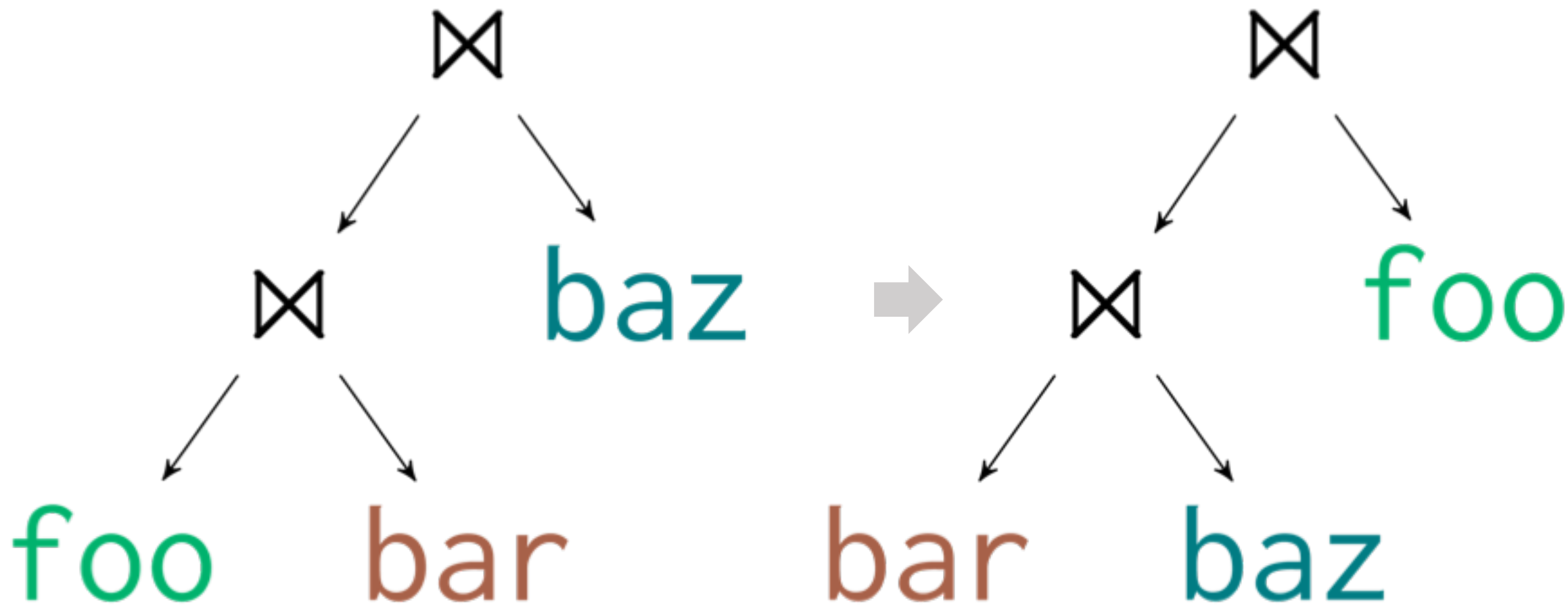
SELECT a FROM foo WHERE 1 + 2 = 4;

SELECT a FROM foo WHERE FALSE;

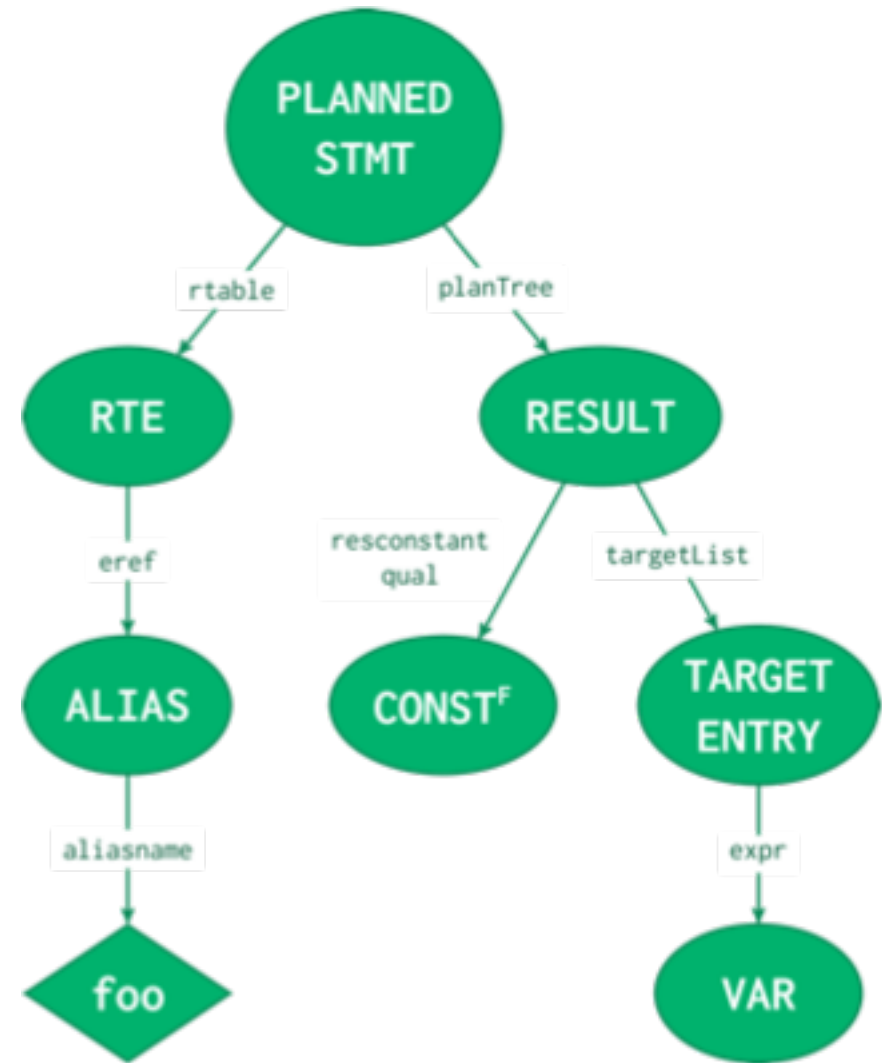
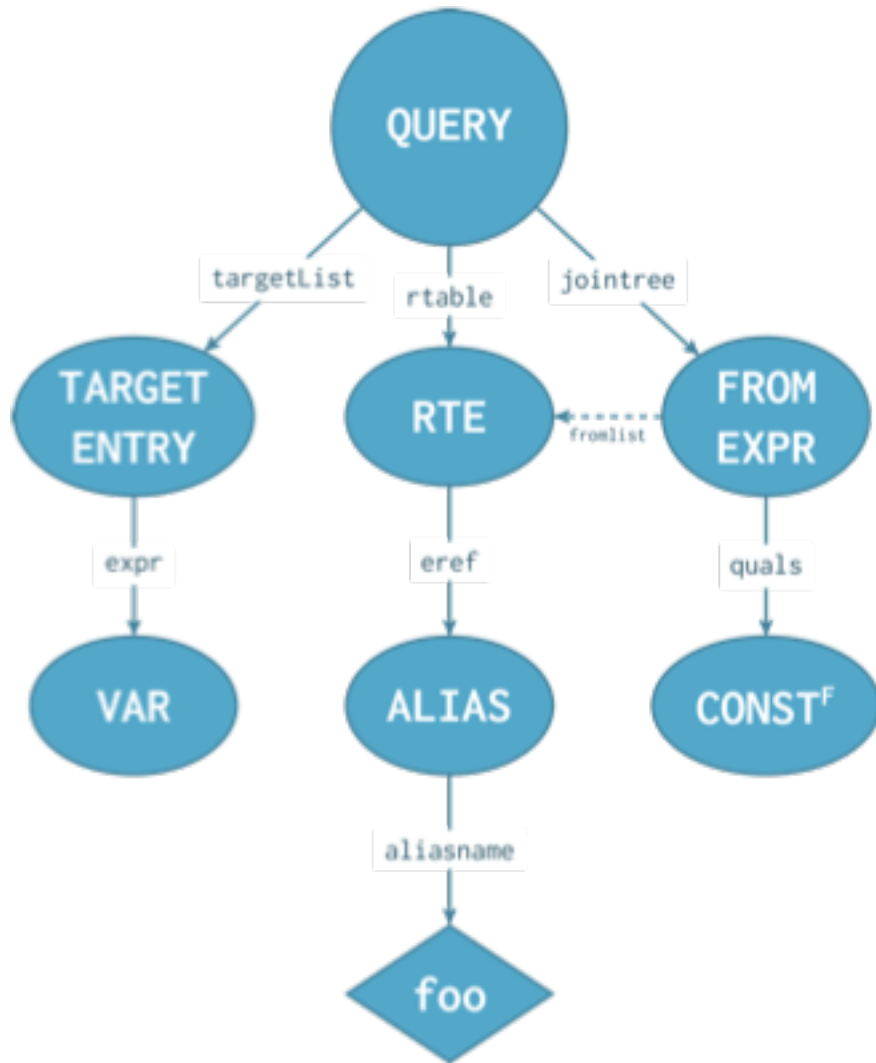


Optimization

# SELECT a FROM foo ⋈ bar ⋈ baz;



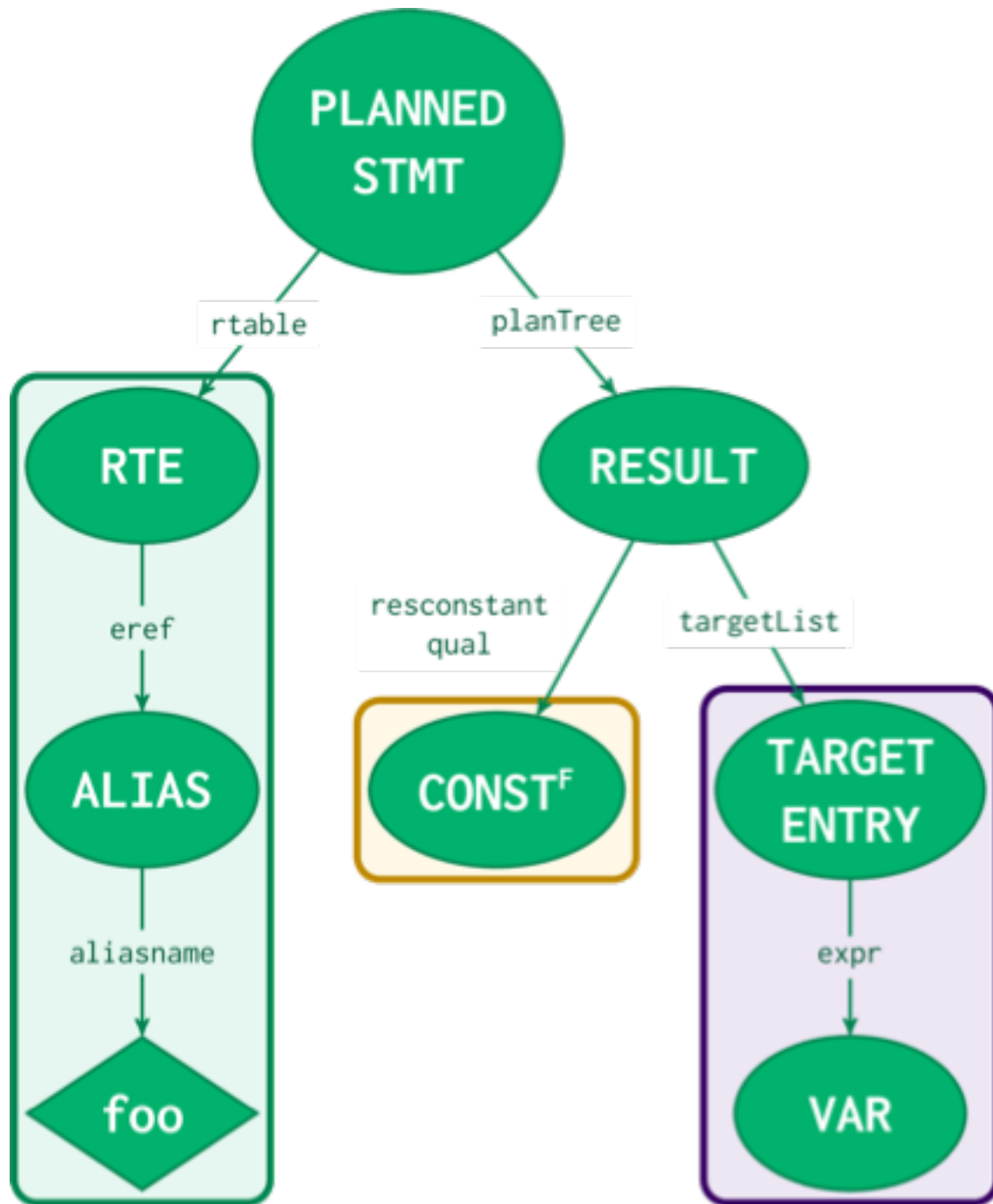
# SELECT **a** FROM **foo** WHERE **FALSE**;



# Plan Tree

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

```
{PLANNEDSTMT
  :planTree
    {RESULT
      :targetlist (
        {TARGETENTRY
          :expr
            {VAR
              :resname a
              :resconstantqual (
                {CONST
                  :constvalue 1 [ 0 0 0 0 0 0 0 0 ]
                }
              )
            }
          :rtable (
            {RTE
              :eref
                {ALIAS
                  :aliasname foo
                  :colnames ("a")
                }
              }
            )
          }
        }
      )
    }
  }
```



```

{PLANNEDSTMT
  :planTree
    {RESULT
      :targetlist (
        {TARGETENTRY
          :expr
            {VAR
              :resname a
            :resconstantqual (
              {CONST
                :constvalue 1 [ 0 0 0 0 0 0 0 0 ]
              :rtable (
                {RTE
                  :eref
                    {ALIAS
                      :aliasname foo
                      :colnames ("a")
                    }
                  }
                )
              }
            )
          }
        )
      }
    }
  }

```



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

## QUERY PLAN

---

```
Seq Scan on foo (cost=0.00..1.03 rows=3 width=4)
```

```
# EXPLAIN SELECT a FROM foo WHERE a = 4;
```

## QUERY PLAN

---

```
Seq Scan on foo (cost=0.00..1.04 rows=1 width=4)  
  Filter: (a = 4)
```

# Case Study

Adding a planner improvement

# Achieving Faster Execution

- Database tuning
- Change the query itself
- **Better plan**

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);
```

# EXPLAIN Output?

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

## QUERY PLAN

---

Result (cost=... rows=0 width=...)

One-Time Filter: (not true)

# EXPLAIN Output!

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

## QUERY PLAN

---

Result (cost=... rows=3 width=...)

One-Time Filter: (SubPlan 1)

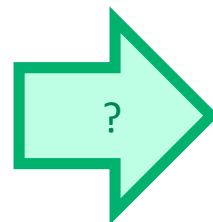
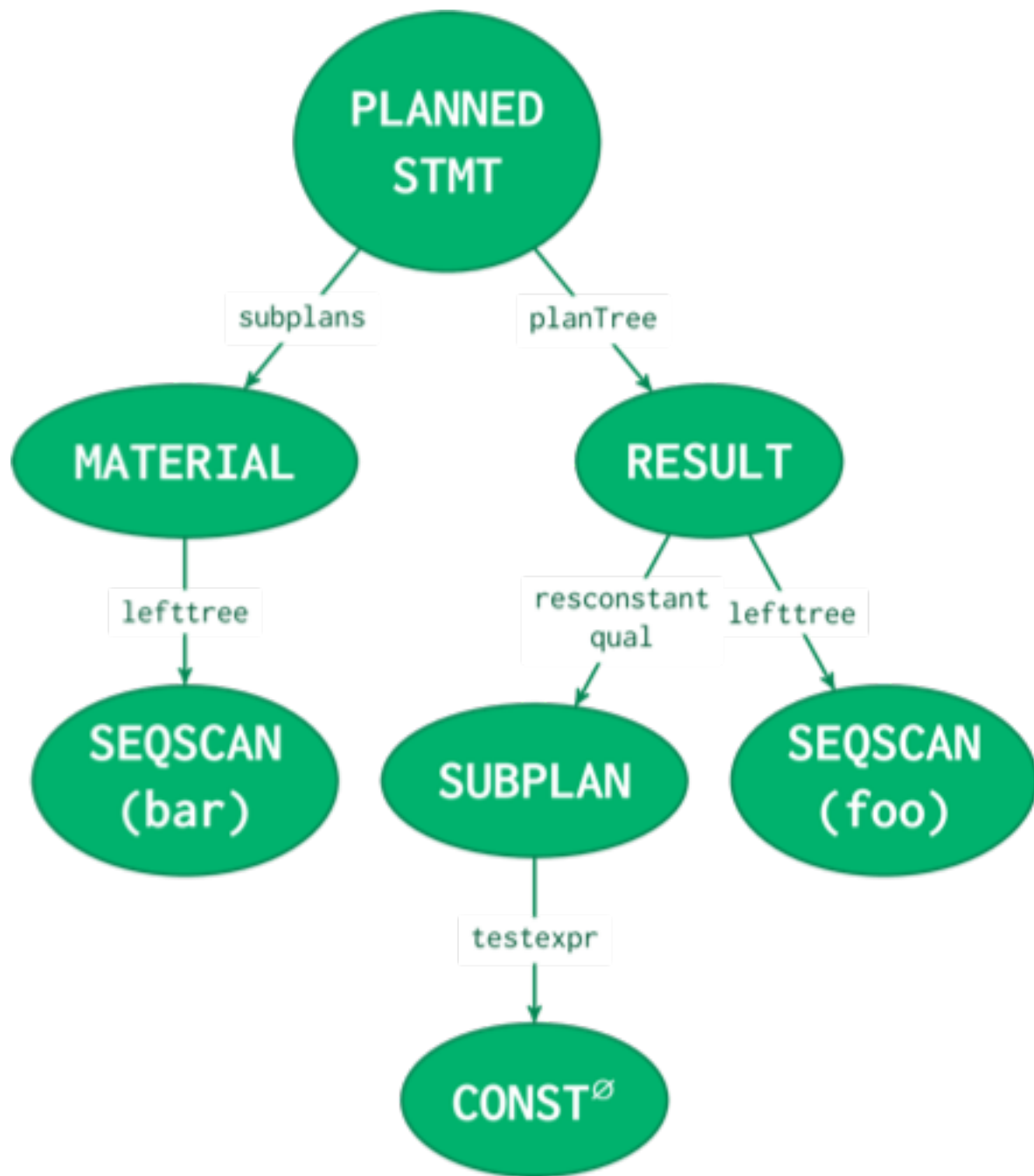
-> Seq Scan on foo (cost=... rows=3 width=...)

SubPlan 1

-> Materialize (cost=... rows=1000 width=...)

-> Seq Scan on bar (cost=... rows=1000 width=...)

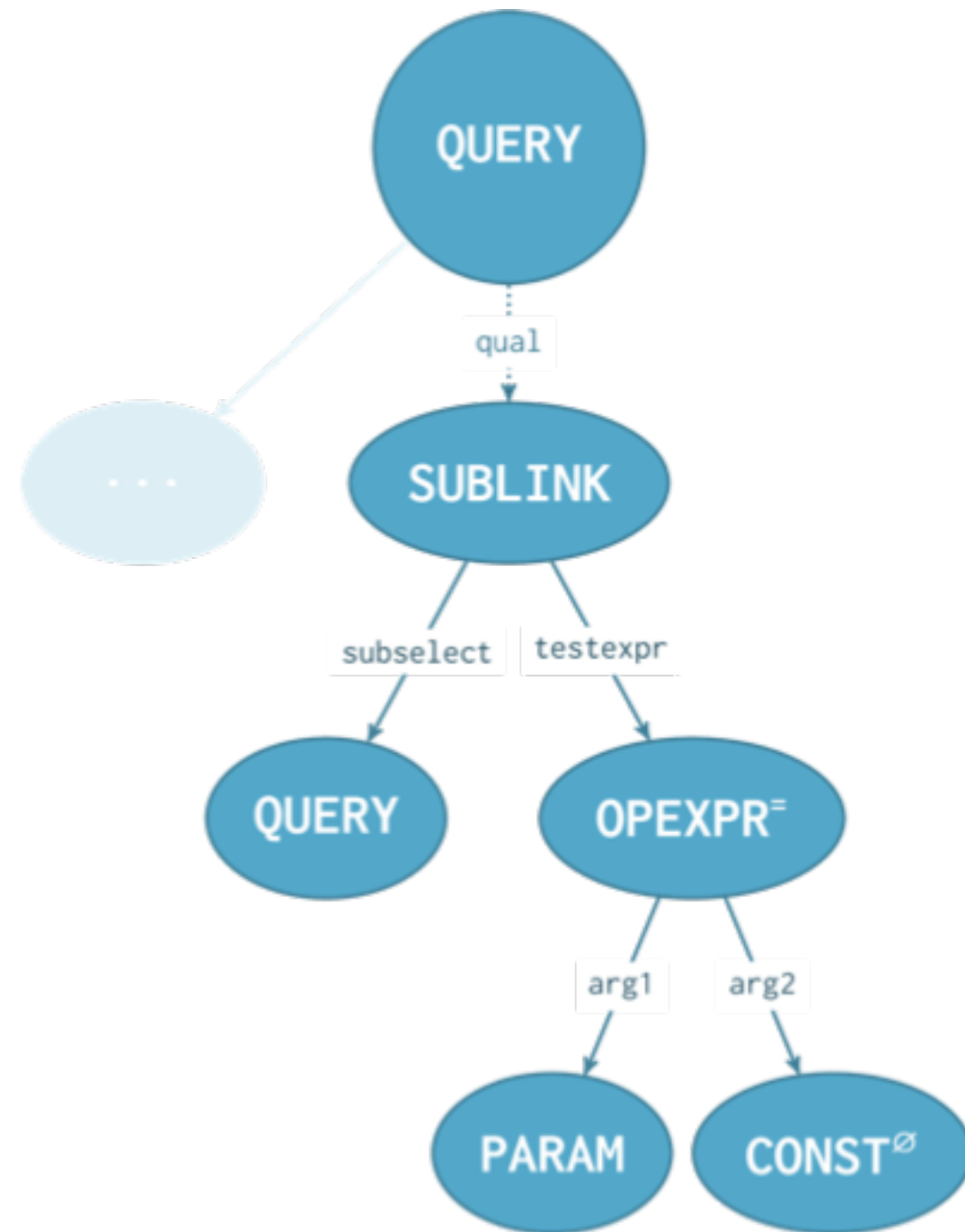




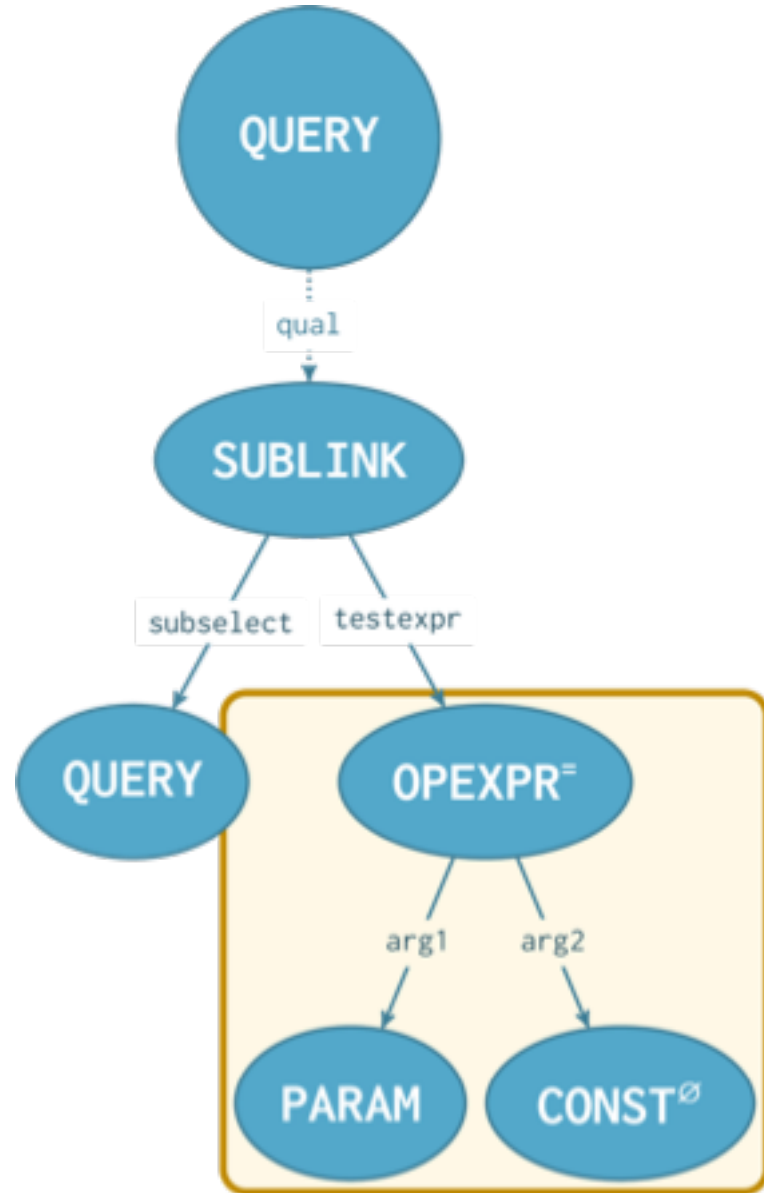
SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);

# Parsing

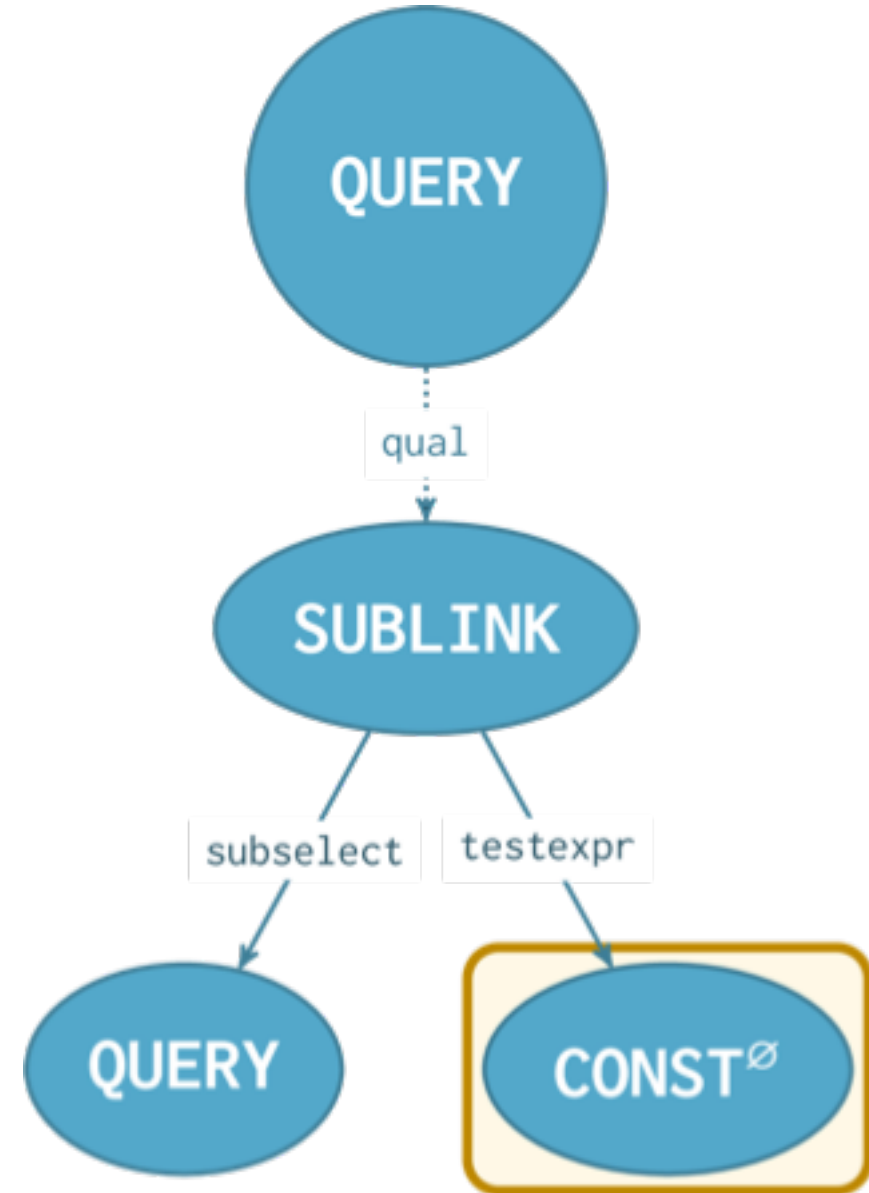
```
# SELECT a FROM foo
  WHERE NULL = ANY(
    SELECT b FROM bar
  );
```



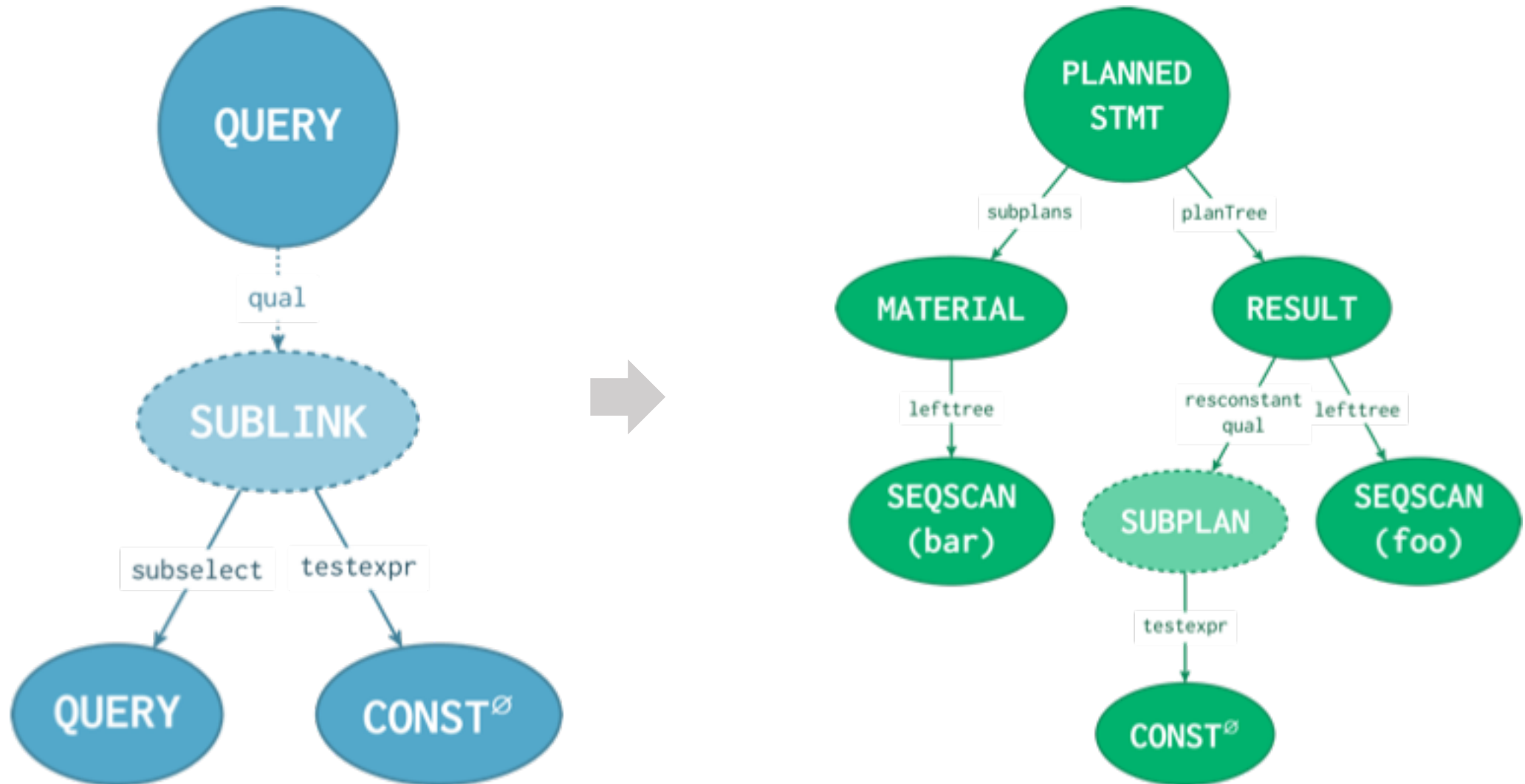
# SELECT a FROM foo WHERE  
NULL = ANY(SELECT b FROM bar);



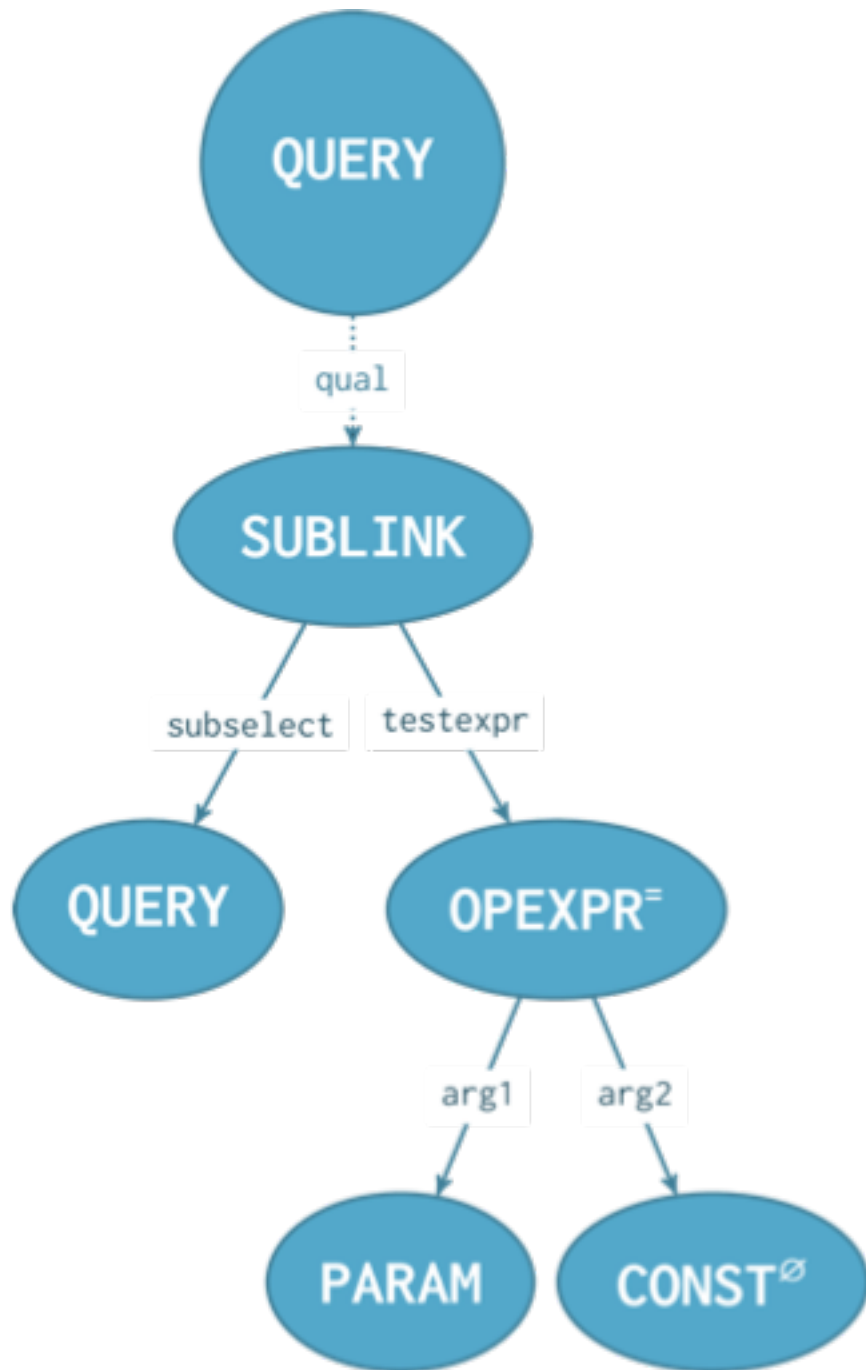
# SELECT a FROM foo WHERE  
(SELECT NULL FROM bar);



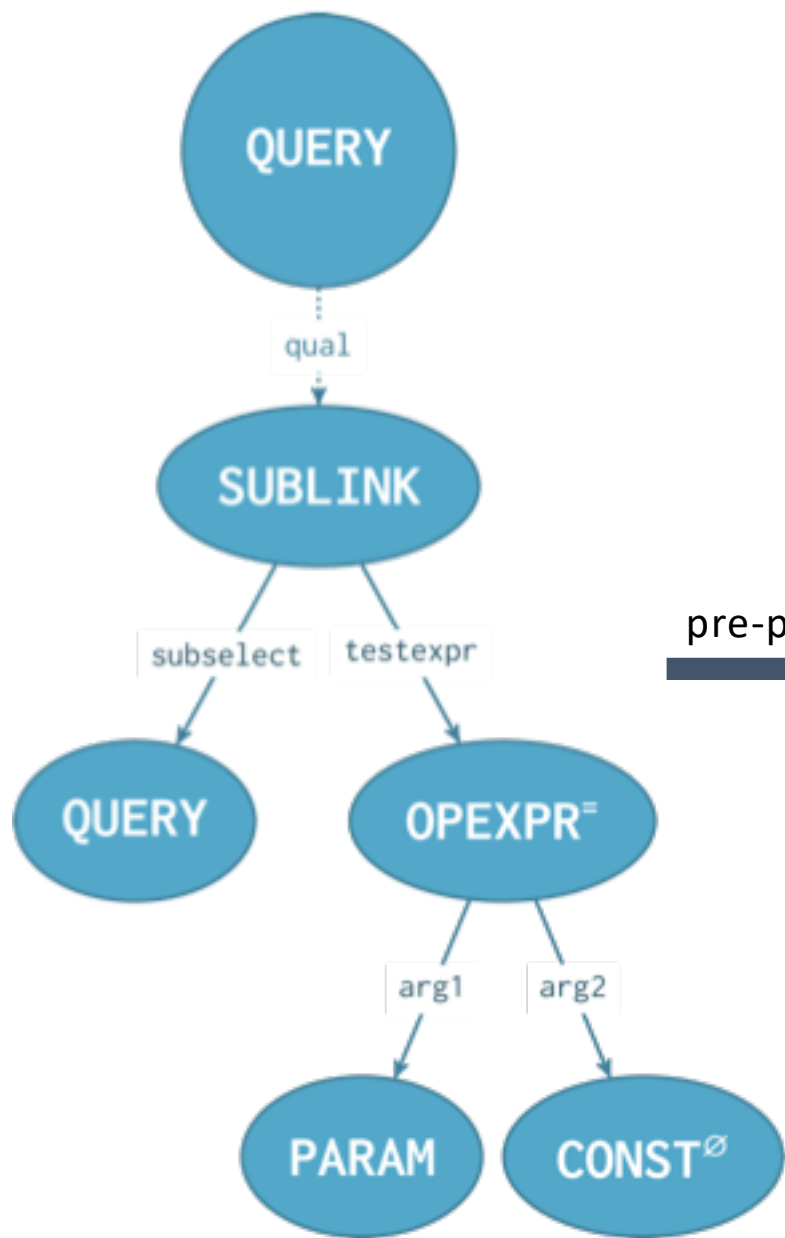
# SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);



Where should the change go?



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



pre-process



optimize

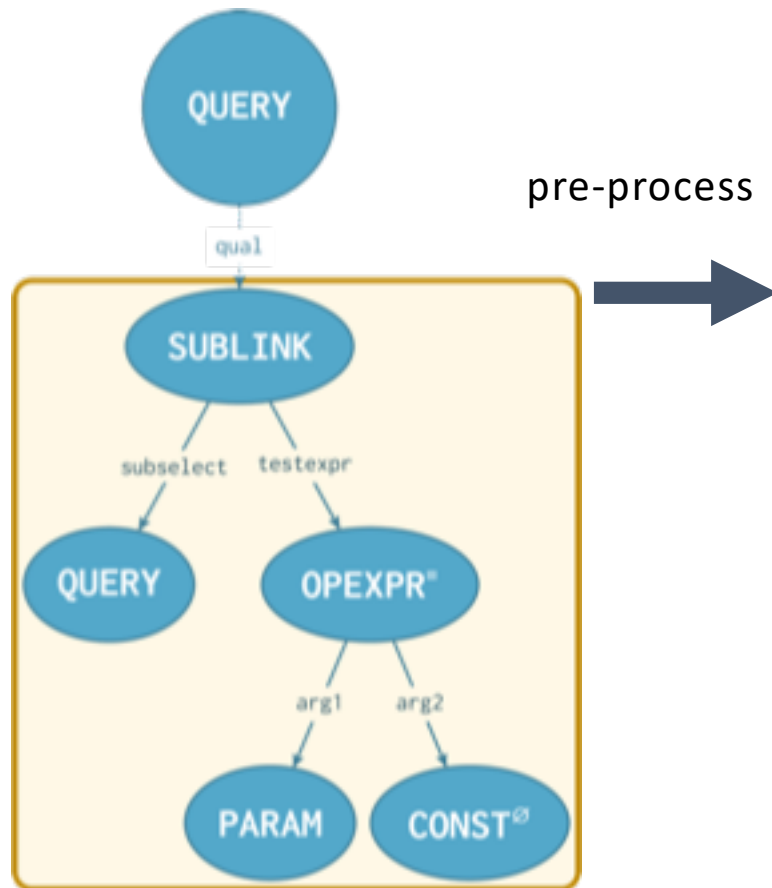


SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);

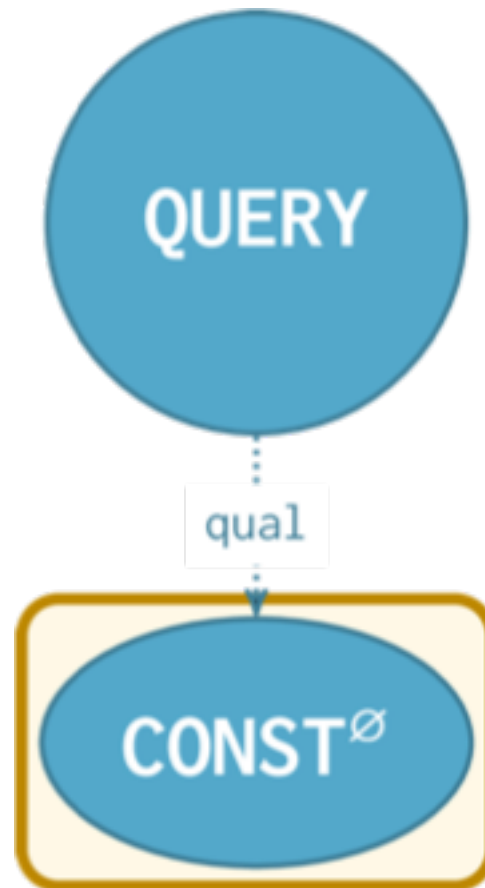
# Proposed pre-processing transformation

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

# SELECT a FROM foo WHERE NULL;



pre-process



optimize





Hacking Pre-processing

```
/*  
 * Invokes the planner for a subquery and recursively  
 * processes each sub-SELECT found in a plan  
 */
```

Plan \*

```
subquery_planner(PlannerGlobal *glob, Query *parse,  
                 PlannerInfo *parent_root,  
                 bool hasRecursion,  
                 double tuple_fraction,  
                 PlannerInfo **subroot,  
                 PlannerConfig *config)  
{  
    ...  
}
```

```
/*  
 * Do subquery_planner's preprocessing work for an  
 * expression, which can be a targetlist, a WHERE clause  
 * (including JOIN/ON conditions), a HAVING clause, etc.  
 */
```

```
static Node *  
preprocess_expression(PlannerInfo *root,  
                       Node *expr, int kind)  
{  
    ...  
    eval_const_expressions(...);  
    ...  
}
```

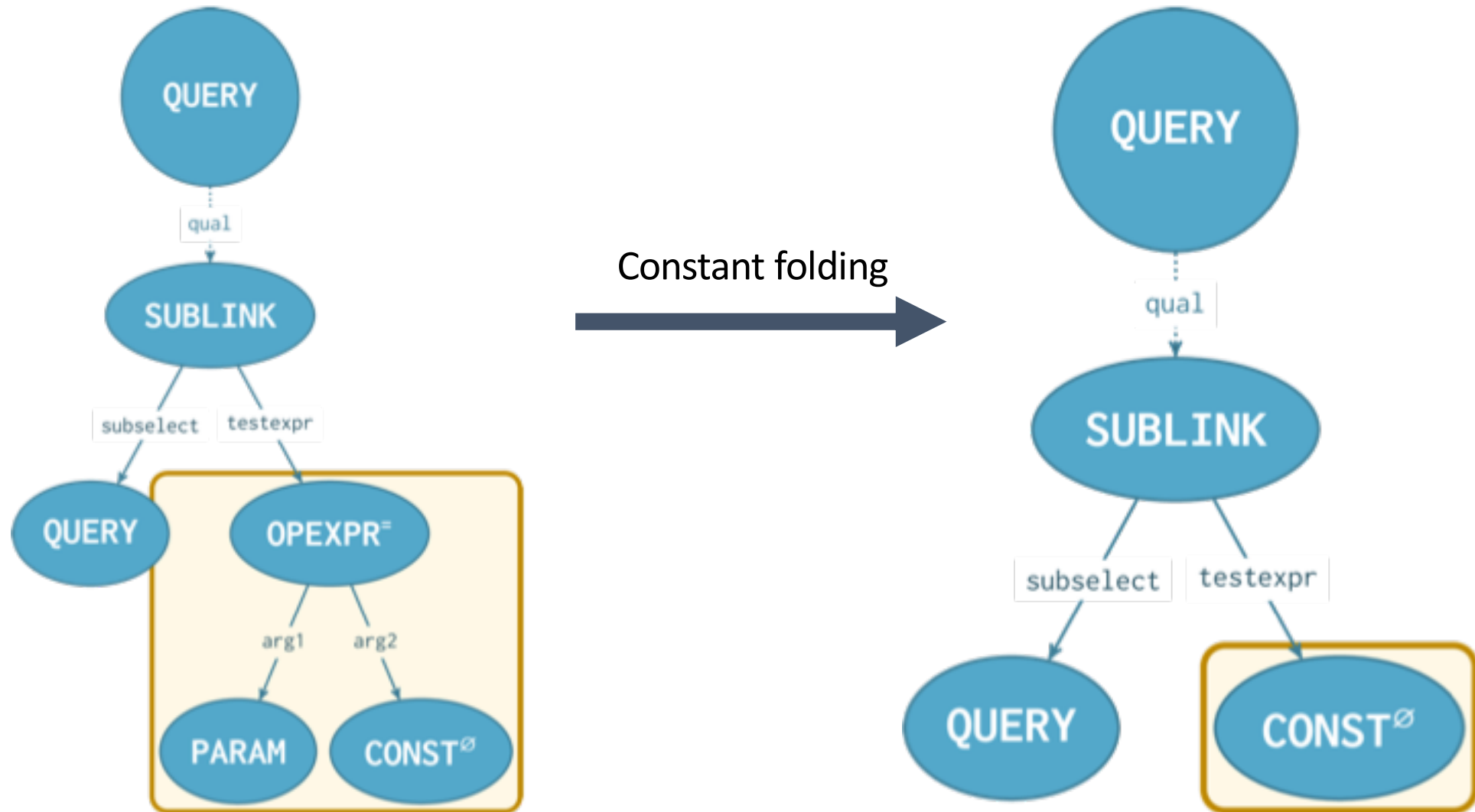
```
/*  
 * Reduce any recognizably constant subexpressions of  
 * the given expression tree, e.g.:  
 *     "2 + 2" => "4"  
 */
```

Node \*

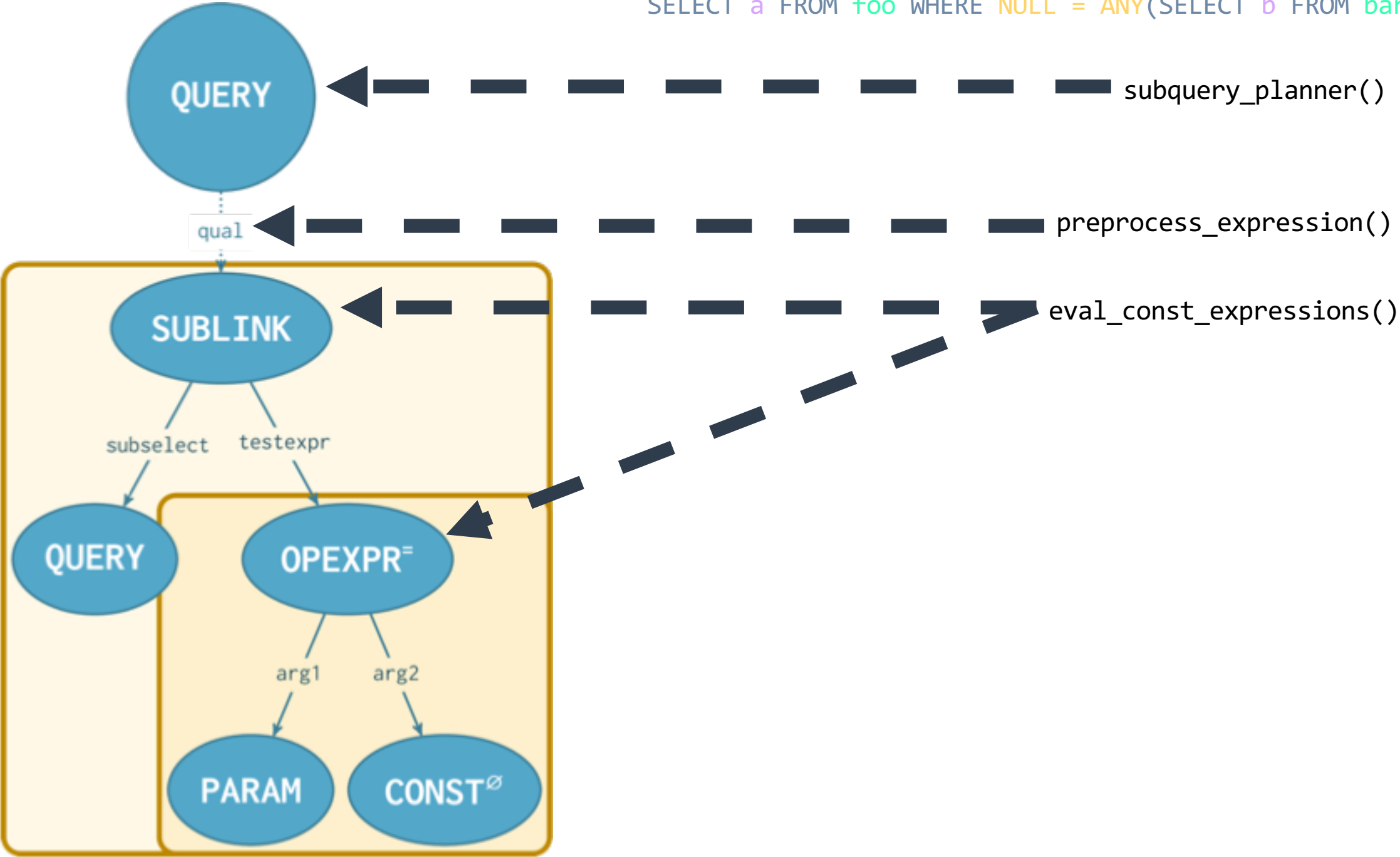
```
eval_const_expressions(PlannerInfo *root, Node *node)  
{  
    ...  
}
```

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

# Current pre-processing

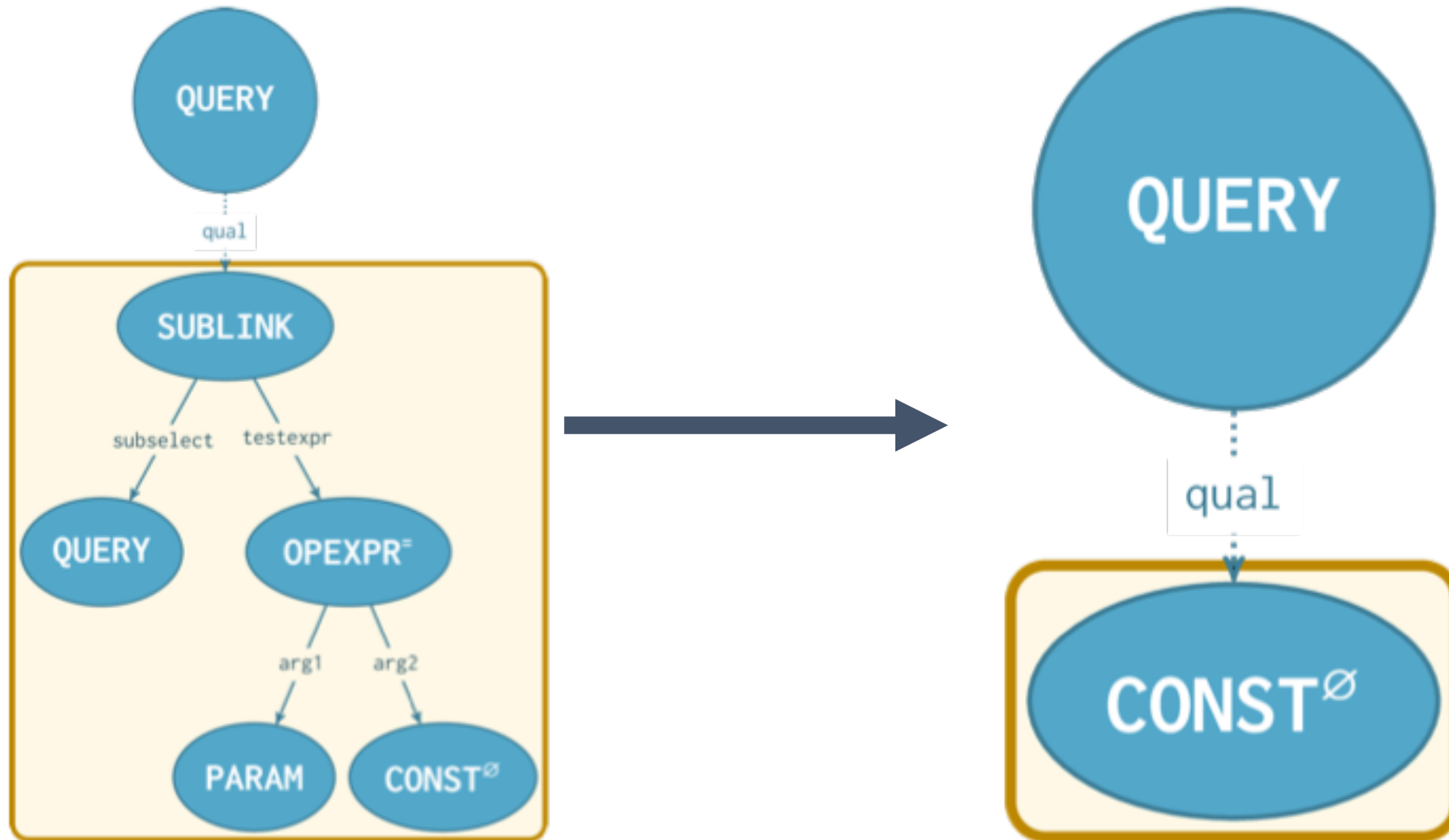


SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);



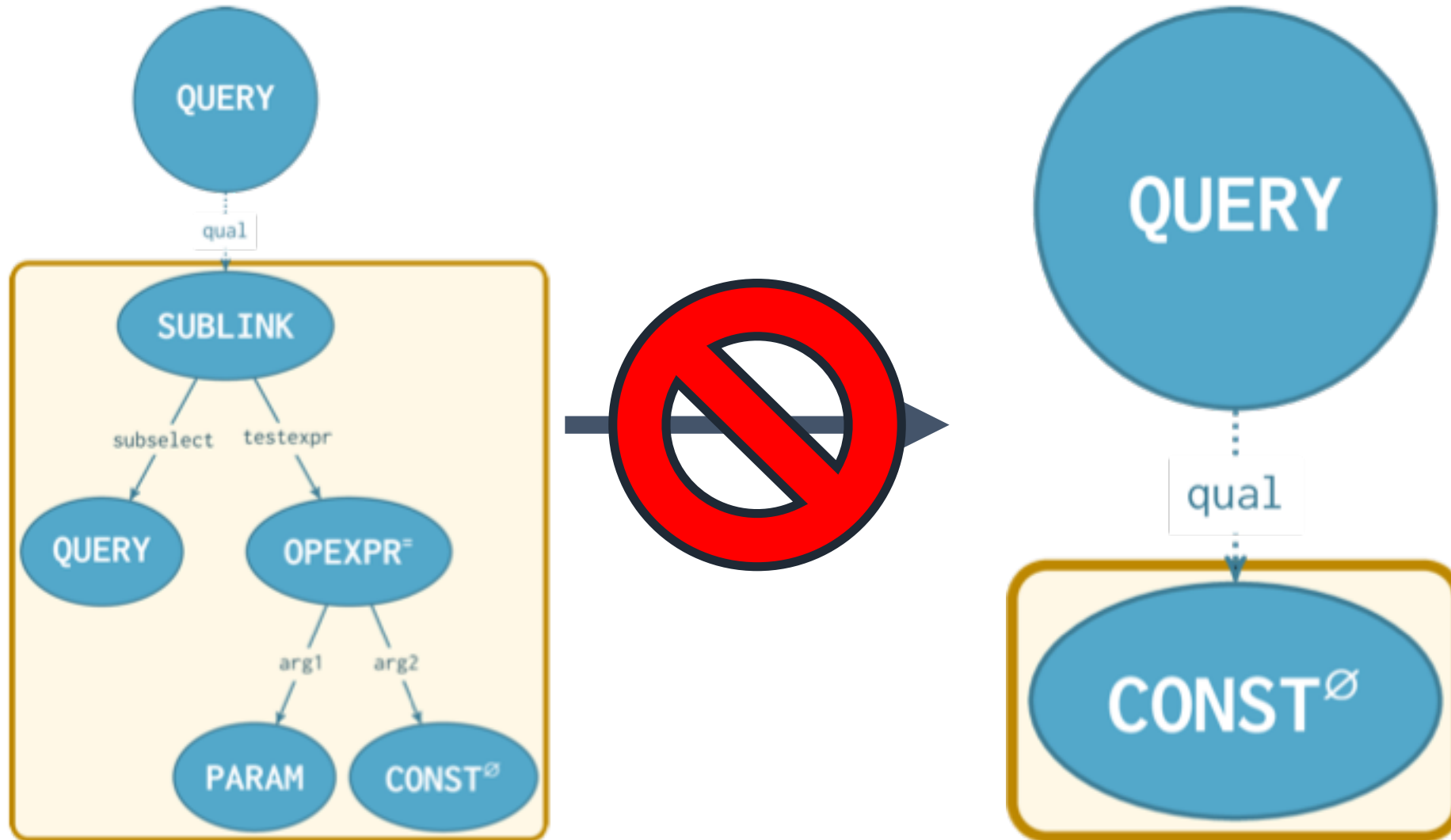
SELECT a FROM foo WHERE NULL = ANY(SELECT b FROM bar);

# Add **SUBLINK** case to constant folding walker



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

# Add **SUBLINK** case to constant folding walker





# NULL Semantics

# NULL $\approx$ Unknown

p

q

p OR q

p AND q

p = q

# NULL $\approx$ 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 $\approx$ 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

NULL  $\stackrel{?}{=}$  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

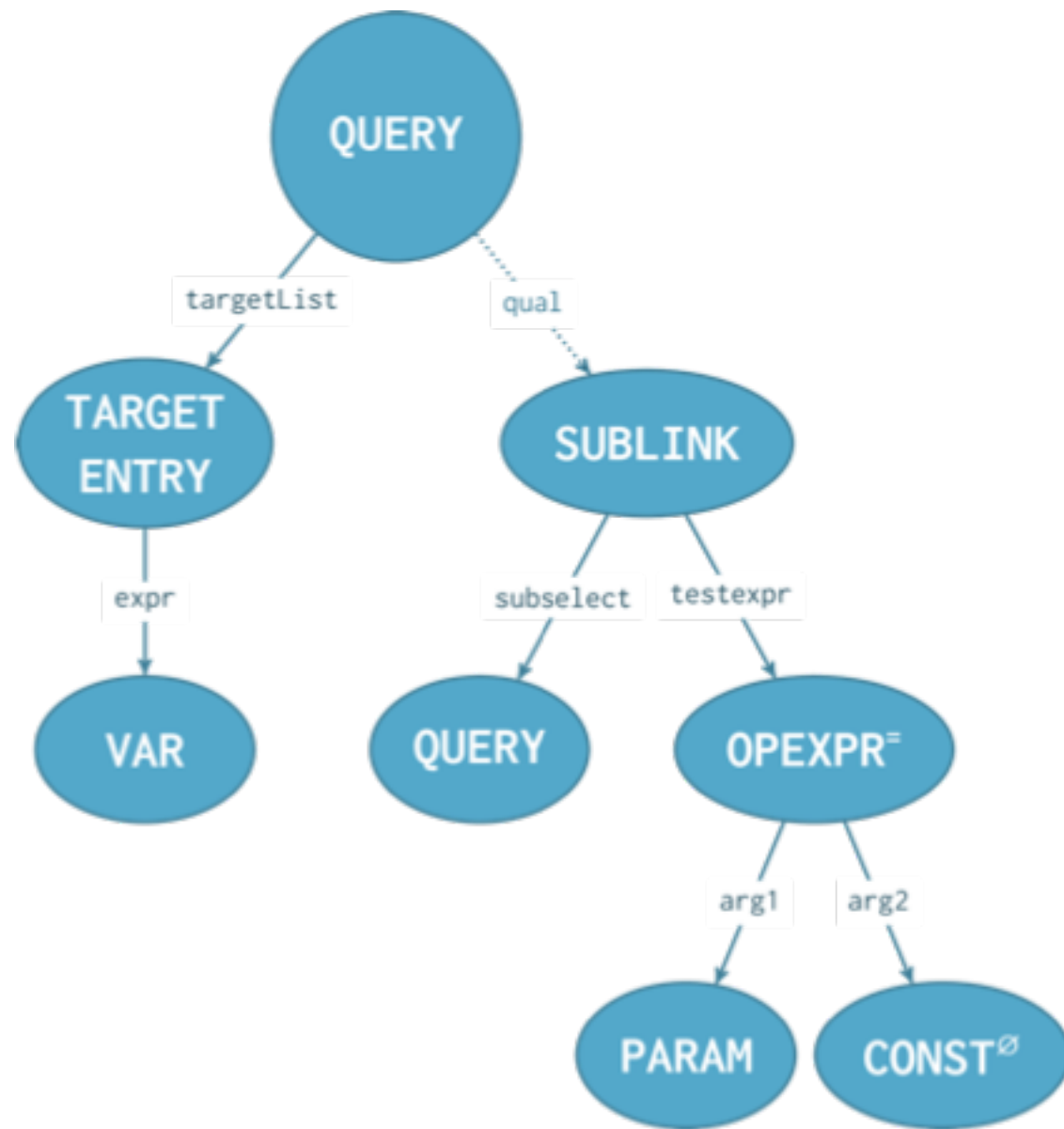
---

(0 rows)

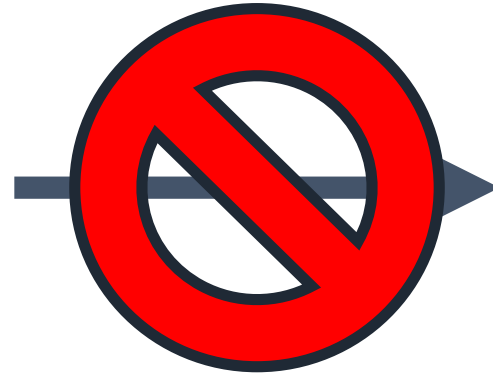
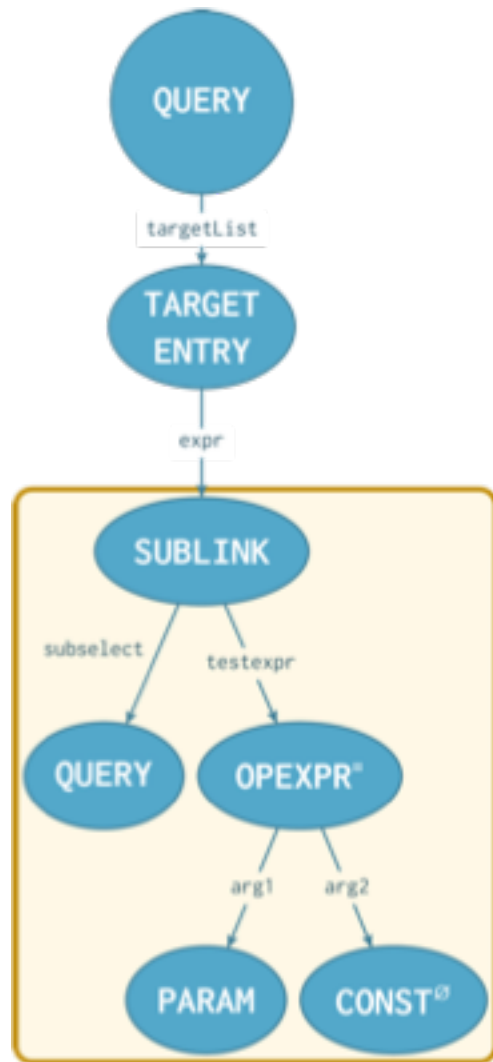
# SELECT NULL = ANY(SELECT b FROM bar);



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



FALSE if bar is an empty table and NULL otherwise

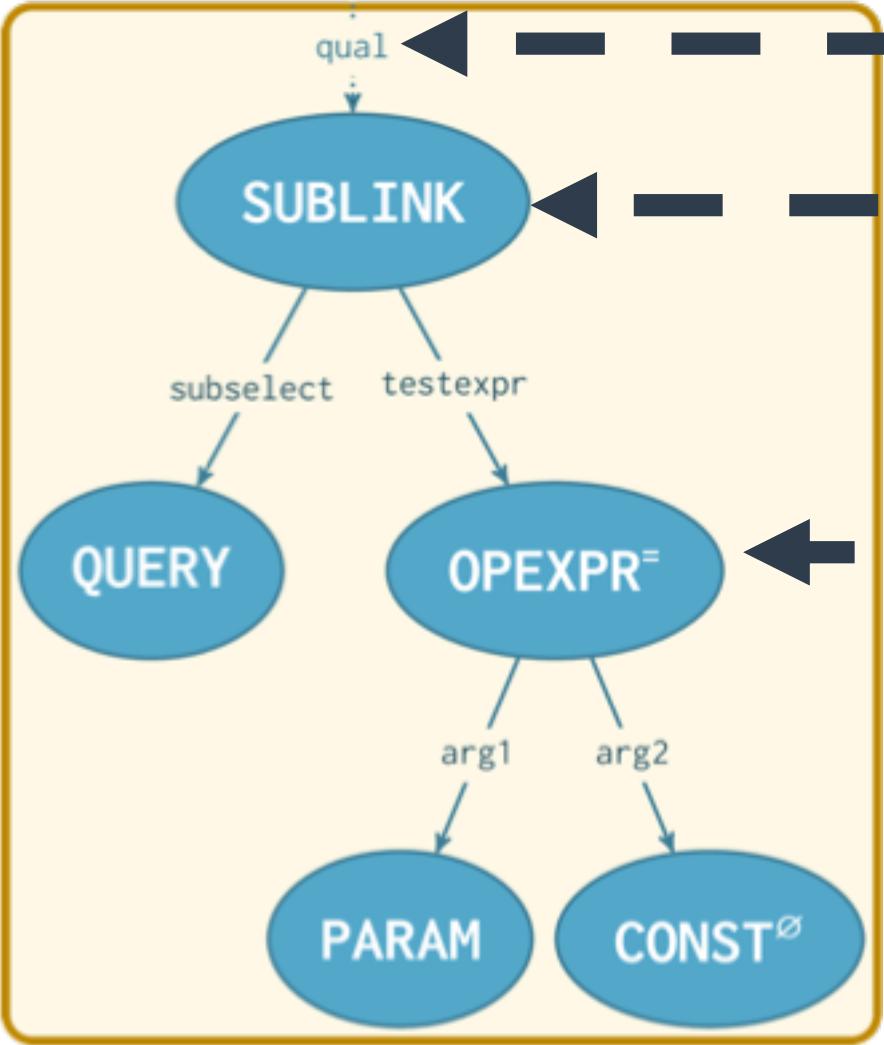


Where else can we put a fix?

```
/*  
 * Recursively scan the query's jointree and do  
 * subquery_planner()'s preprocessing work on each qual  
 * condition found therein.  
 */
```

```
static void  
preprocess_qual_conditions(PlannerInfo *root,  
                           Node *jtnode)  
{  
    ...  
    preprocess_expressions(...);  
    ...  
}
```

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



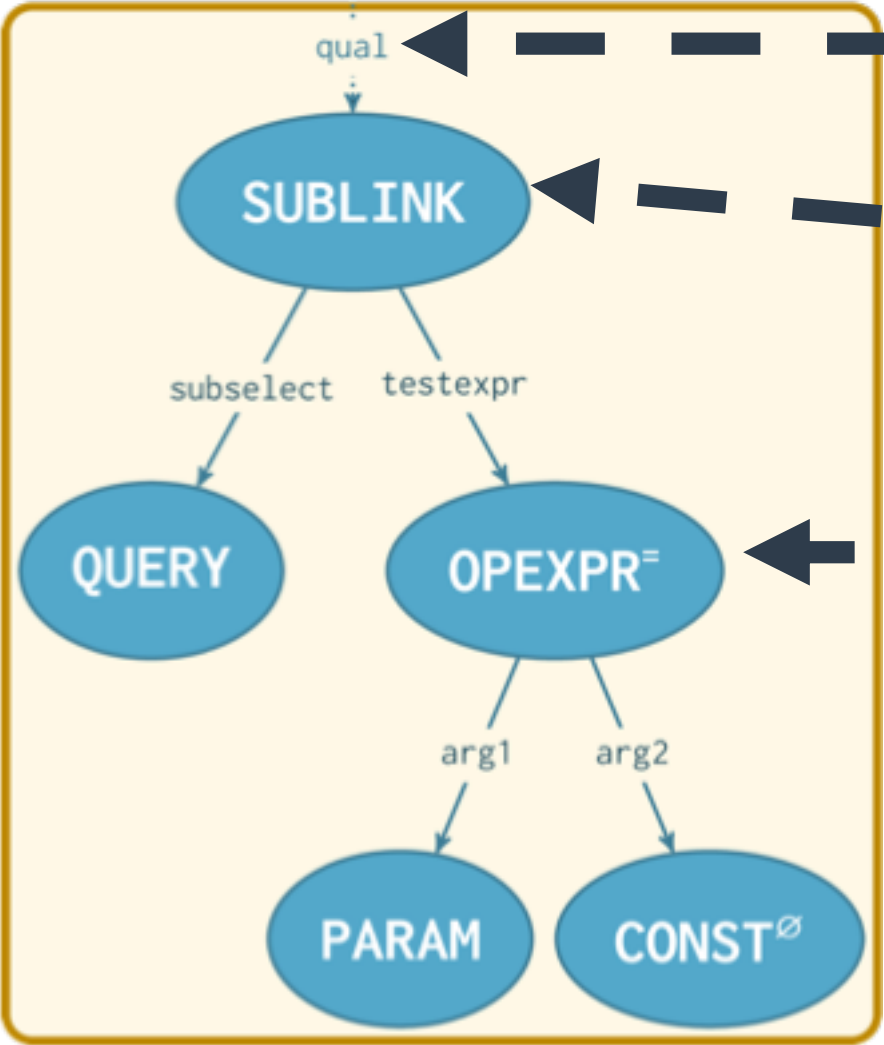
subquery\_planner()

preprocess\_qual\_conditions()

preprocess\_expression()

eval\_const\_expressions()

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



subquery\_planner()

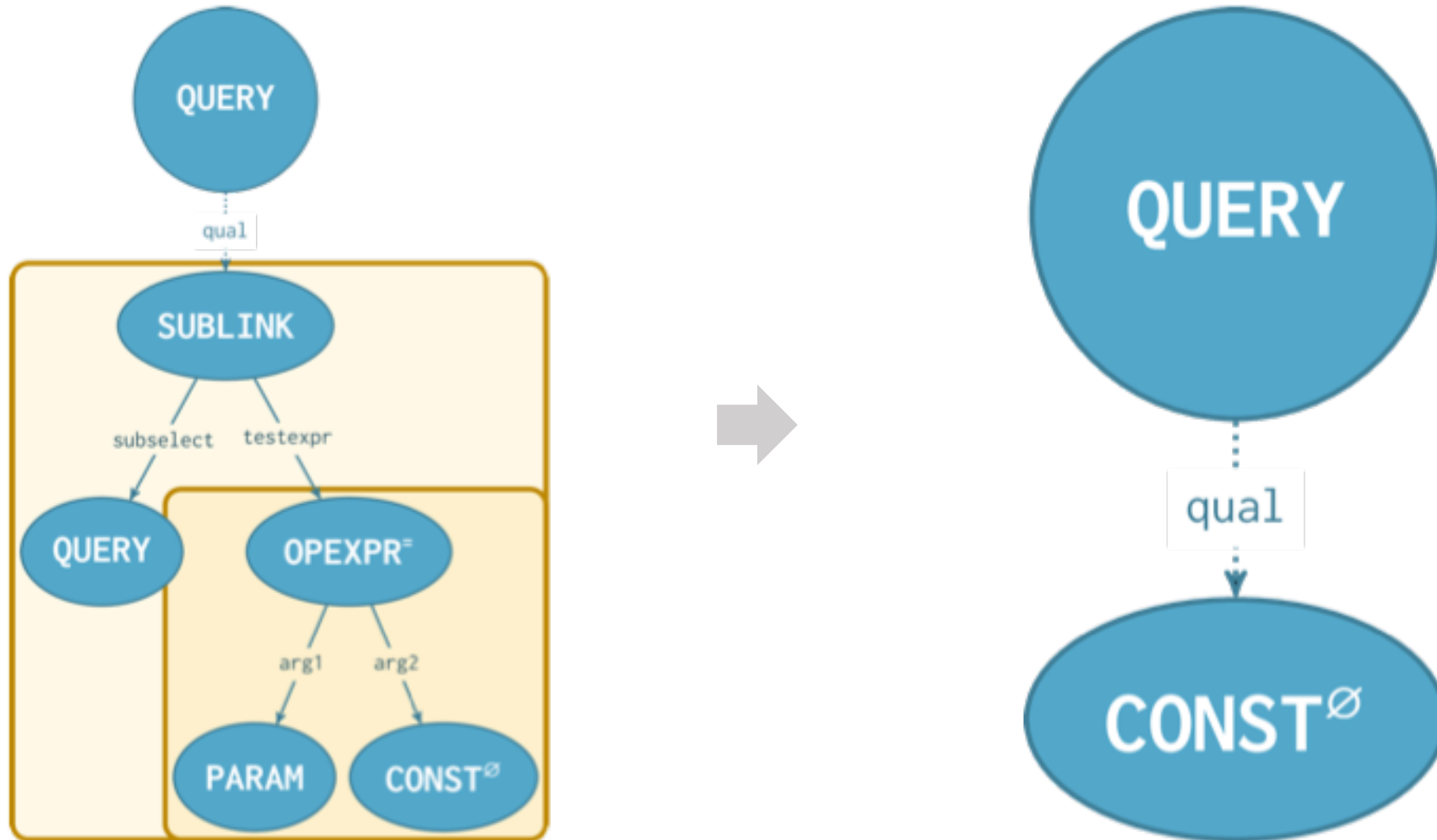
```
preprocess_qual_conditions()  
{  
  if, after constant folding,  
  testexpr is a constant NULL,  
  replace SUBLINK with constant  
  NULL  
}
```

preprocess\_expression()

eval\_const\_expressions()

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

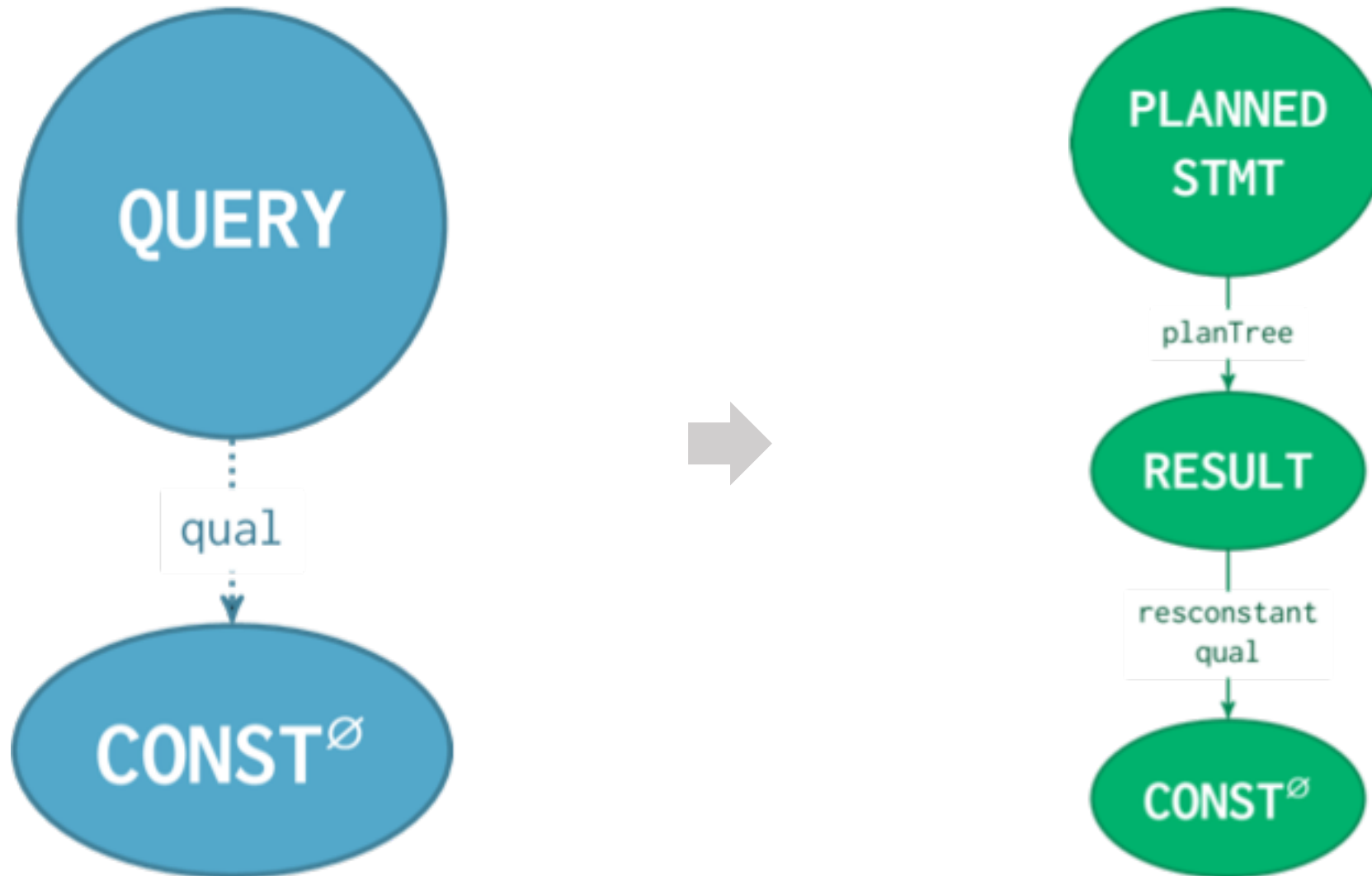
# Replace **SUBLINK** when pre-processing **quals**





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=0.00..0.00 rows=0 width=4)

One-Time Filter: false

[github.com/melanieplageman](https://github.com/melanieplageman)

**slides and glossary:** [github.com/melanieplageman/debugging\\_planner](https://github.com/melanieplageman/debugging_planner)

**code:** [github.com/melanieplageman/postgres/tree/experiment\\_null\\_in\\_subquery](https://github.com/melanieplageman/postgres/tree/experiment_null_in_subquery)