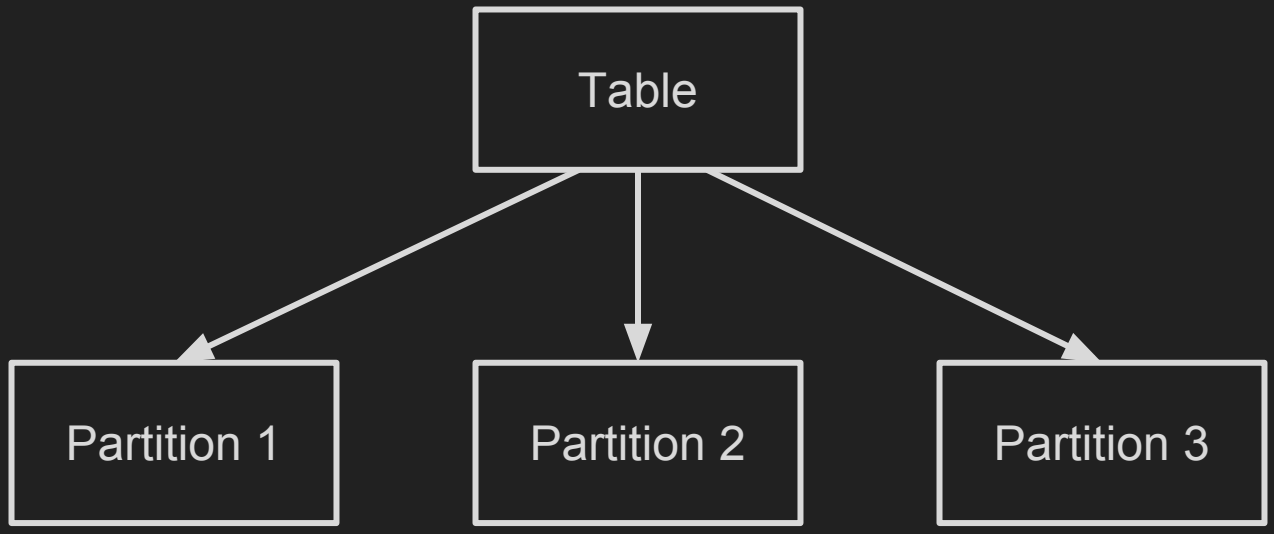
Ildar Musin
i.musin@postgrespro.ru

Dmitry Ivanov
d.ivanov@postgrespro.ru

pg_pathman

partitioning extension for PostgreSQL

Partitioning



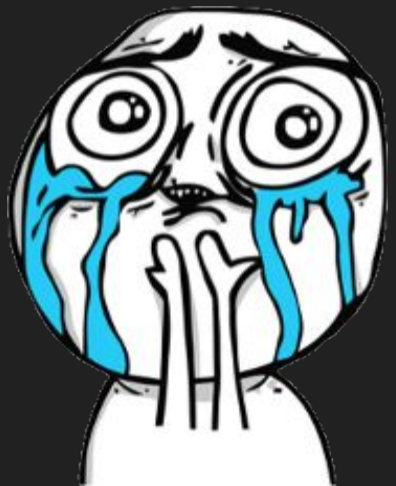
Partitioning advantages

1. Effective buffer cache usage in case when most frequently used data located in a small subset of partitions
2. Sequential access to a single (or few) partition instead of random access to the whole large table
3. Bulk operations
4. “Cold” data can be moved to a slower/cheaper data storage
5. Separate partition maintenance

Partitioning in PostgreSQL 9.6 and below

- Inheritance
- Check constraints
- Constraint exclusion mechanism (exhaustive search)
- **LIST** and **RANGE** partitioning
- Trigger-based tuple routing (slow)
- No executor optimization
- No partition-specific **DDL** syntax

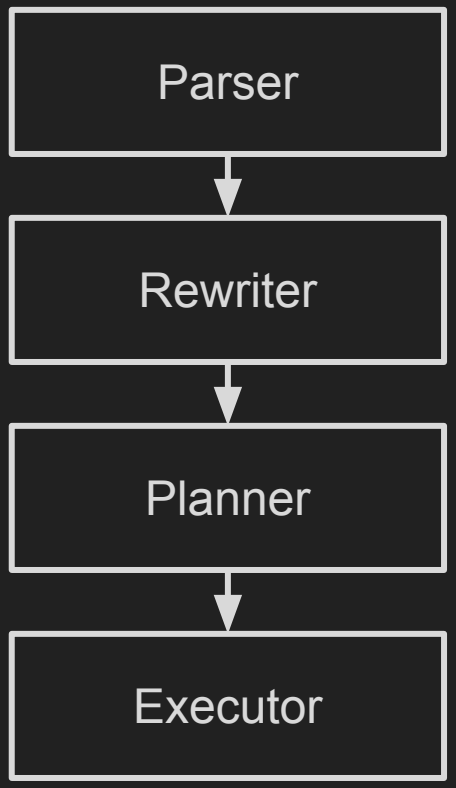
pg_pathman



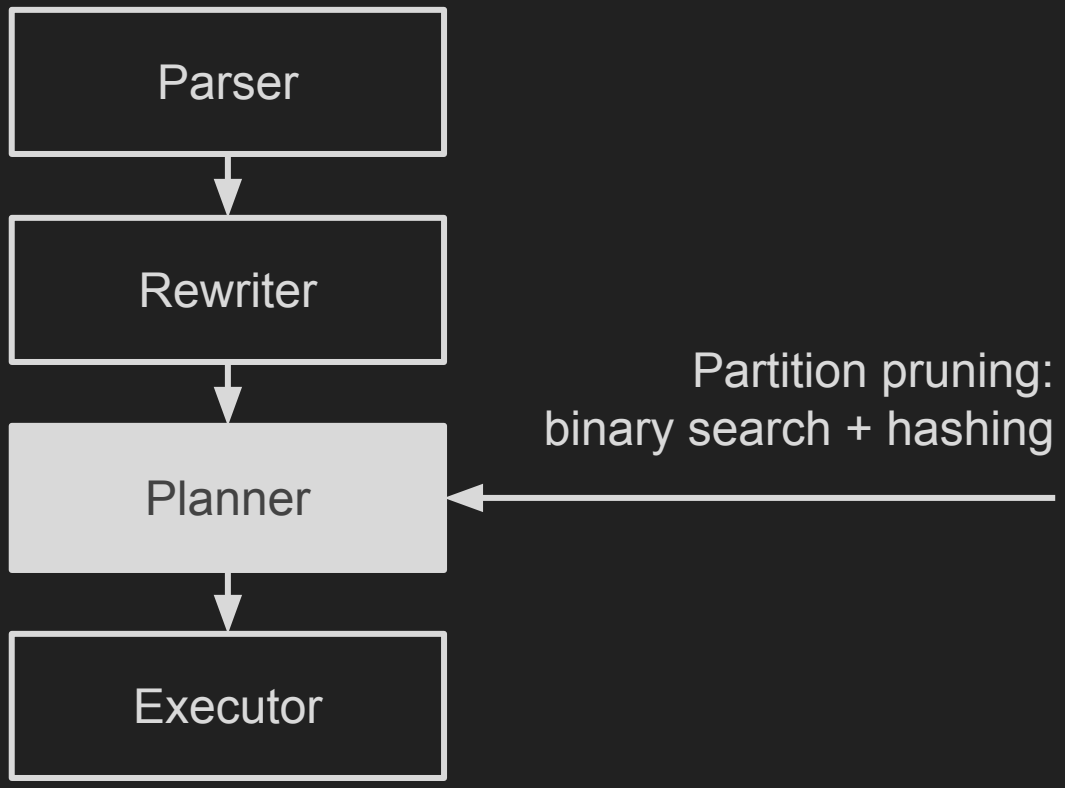
pg_pathman's features

- optimized planning
- execution optimizations (RuntimeAppend)
- **HASH** & **RANGE** partitioning
- custom node for INSERTs
- auto partition creation + user-defined callbacks
- concurrent data migration
- foreign tables (FDW) support
- rich and convenient **API**

How does it work?



How does it work?



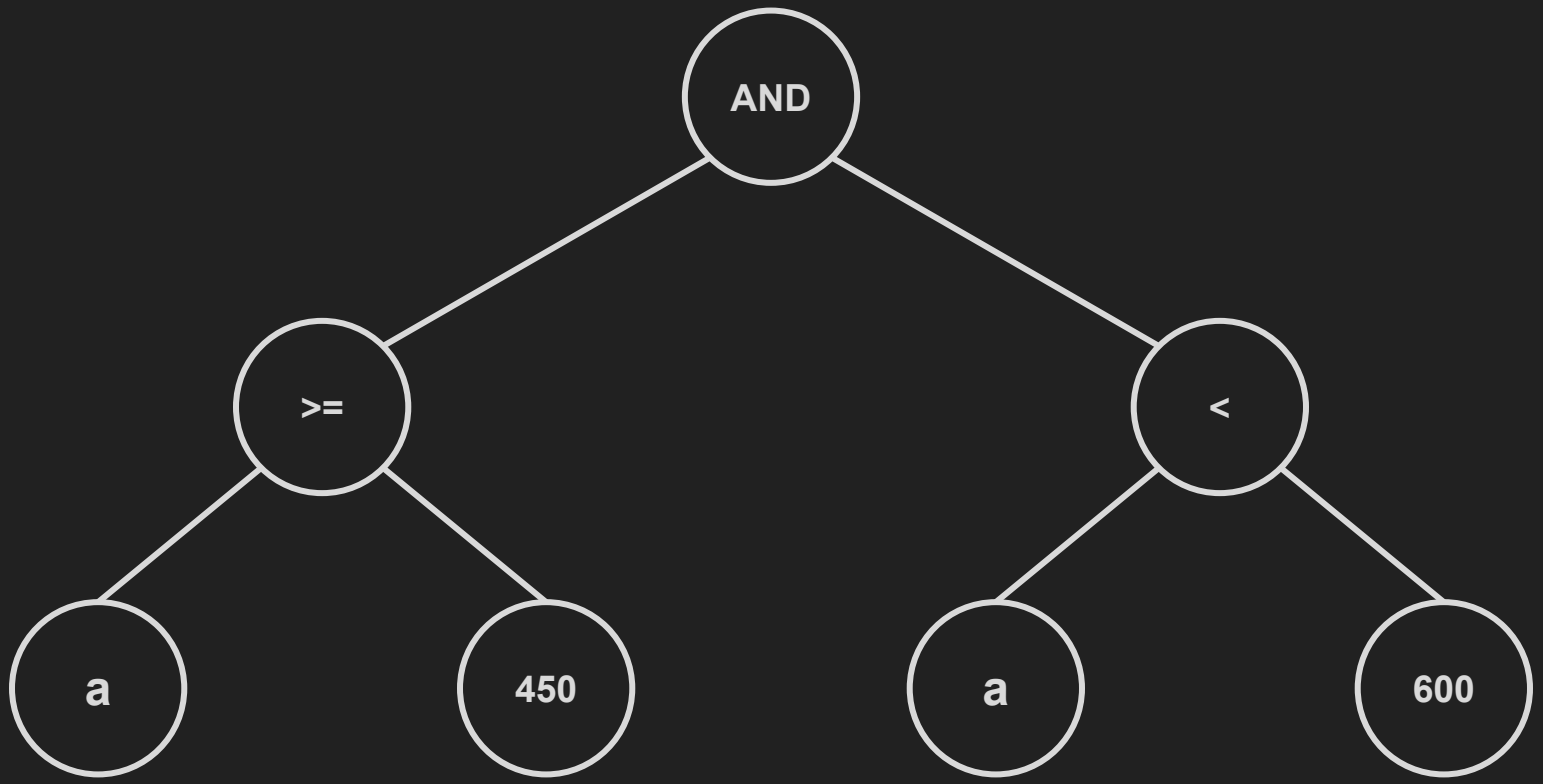
How does it work?

Query analysis

```
SELECT * FROM my_table
WHERE a >= 450 and a < 600
```

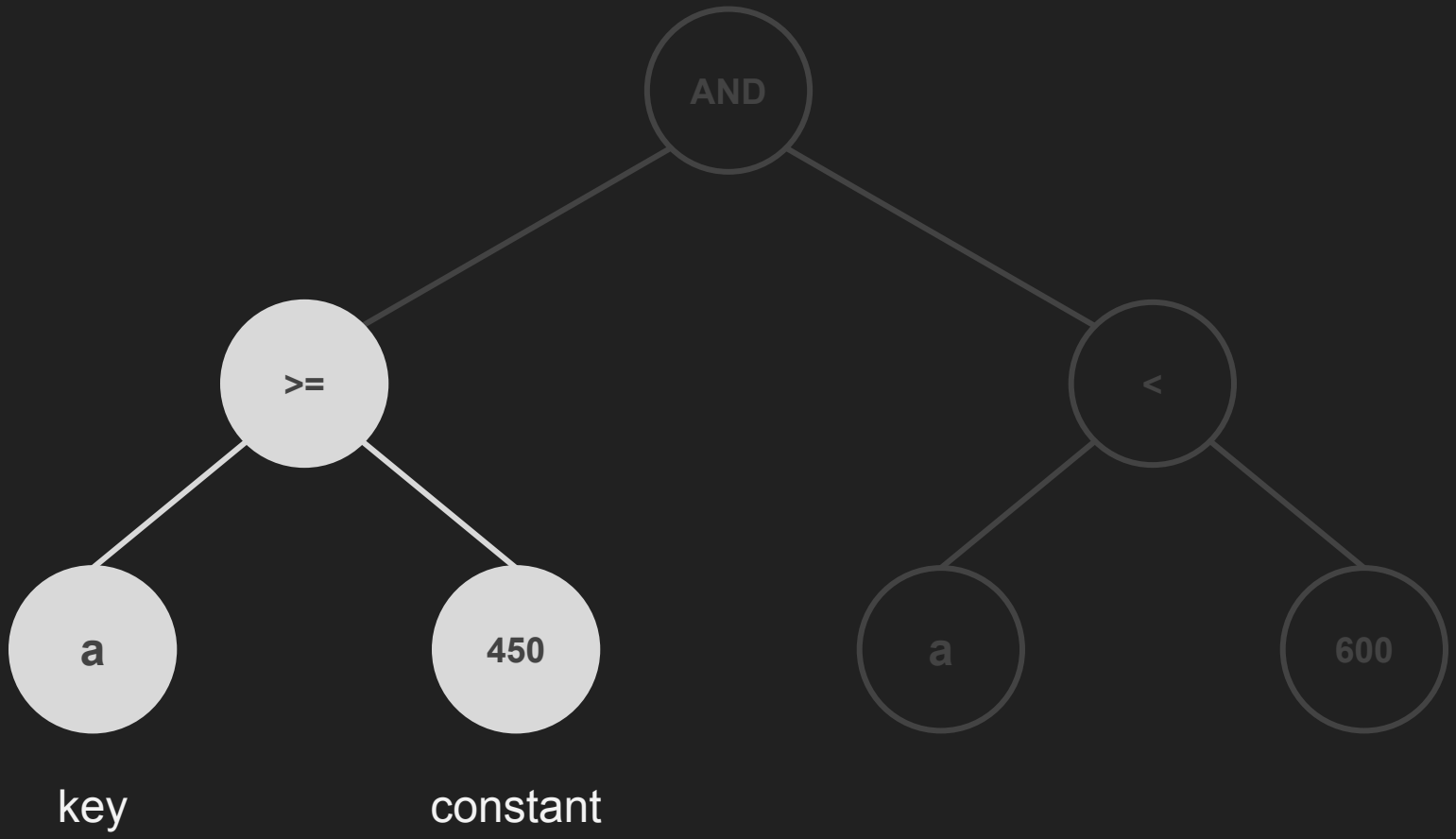
How does it work?

Query analysis



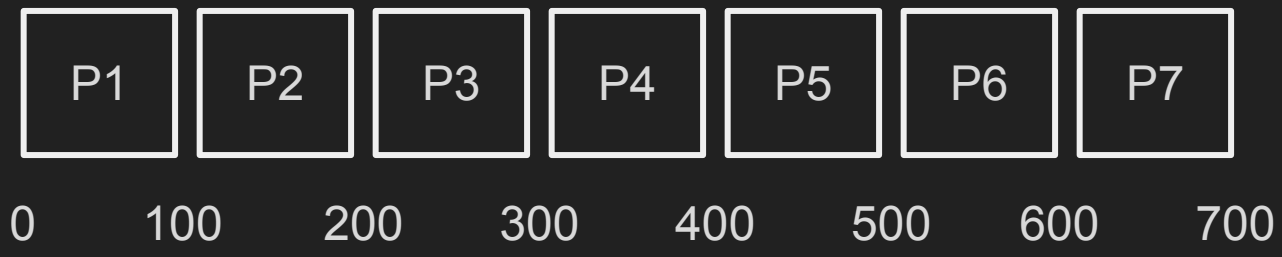
How does it work?

Query analysis



How does it work?

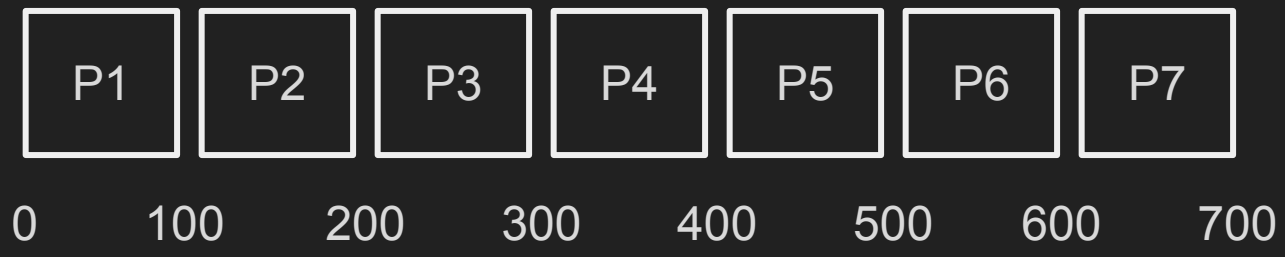
Partitions cache



How does it work?

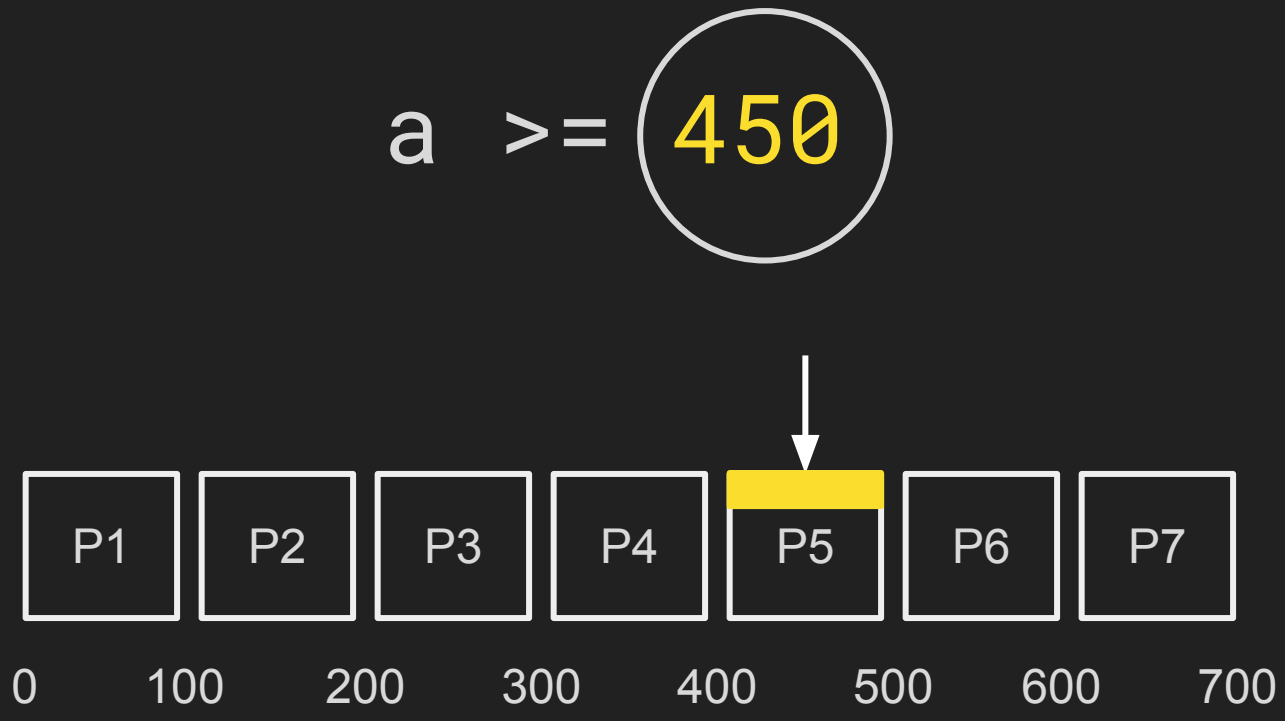
Partition pruning

```
a >= 450
```



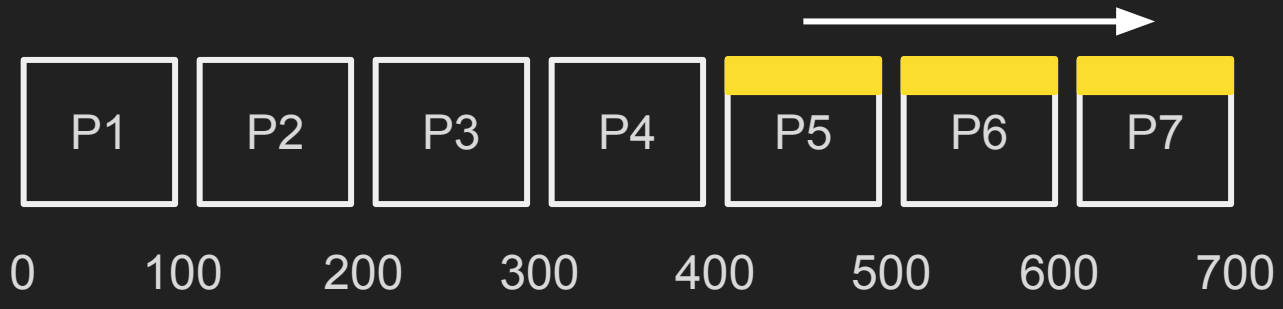
How does it work?

Partition pruning



How does it work?

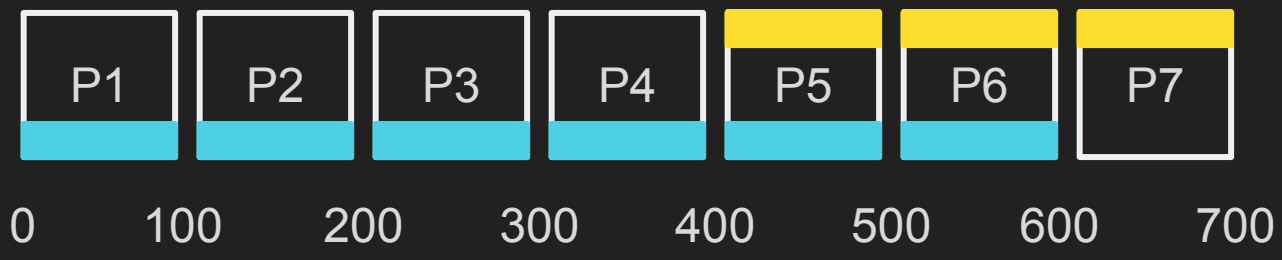
Partition pruning



How does it work?

Partition pruning

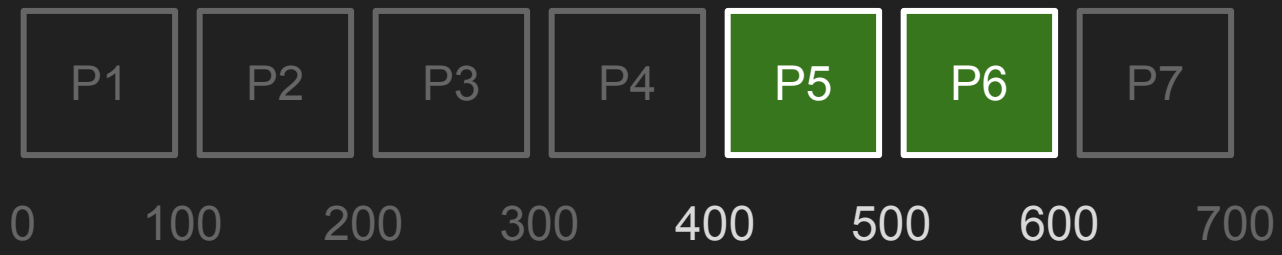
```
a >= 450 and a < 600
```



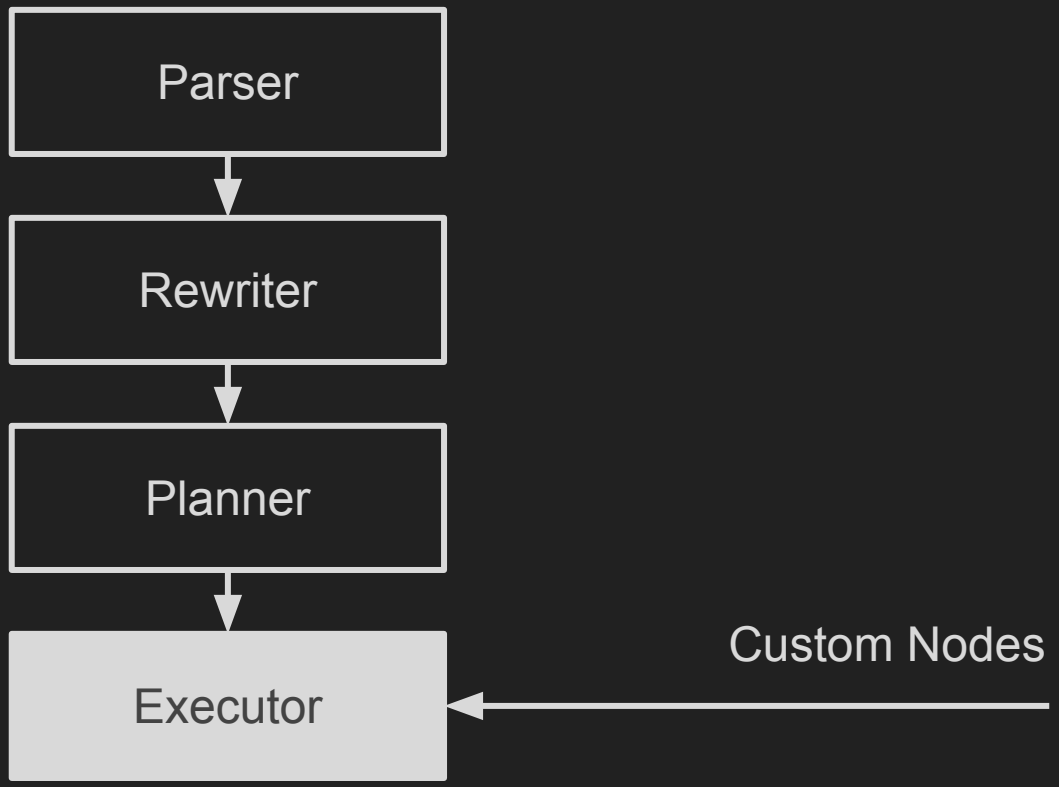
How does it work?

Partition pruning

```
a >= 450 and a < 600
```



How does it work?



How does it work?

Parameterized queries

```
SELECT * FROM my_table
WHERE a = ?
```

How does it work?

Parameterized queries

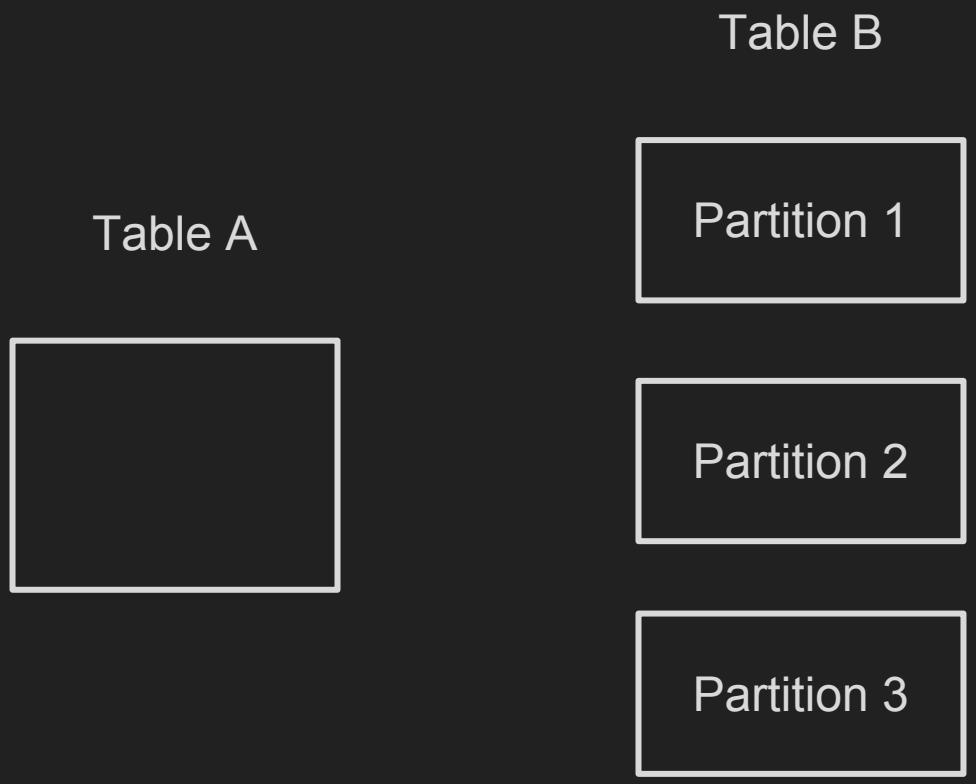
```
SELECT * FROM my_table
WHERE a = ?
```

Parameterized queries:

- prepared statements
- nested loops
- subqueries

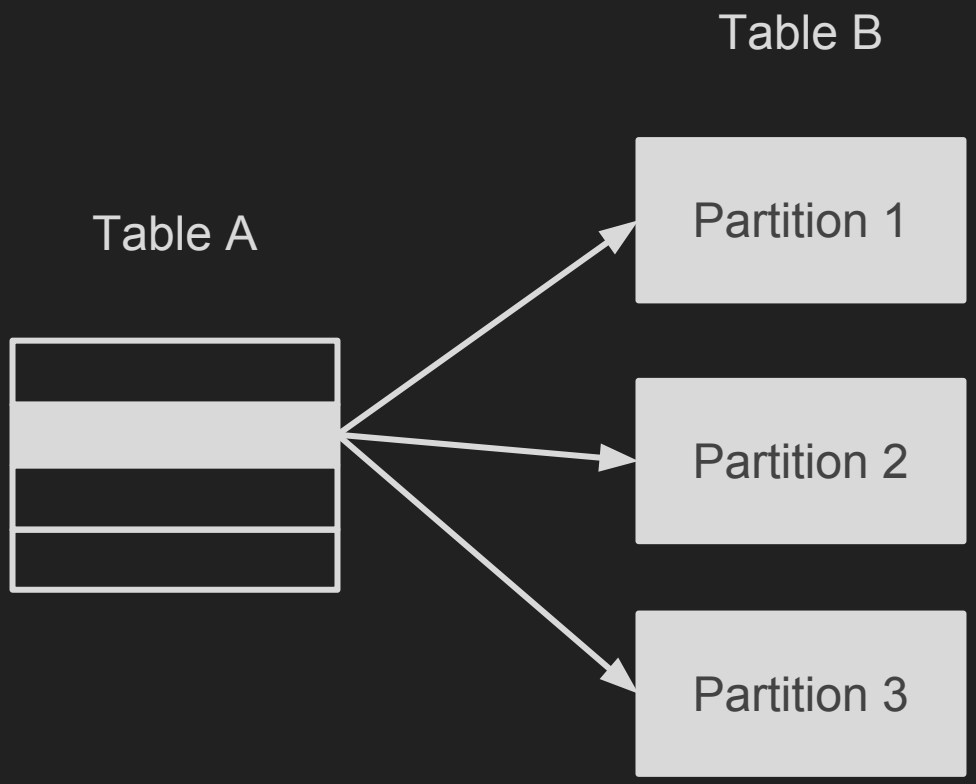
Execution example

Nested loop



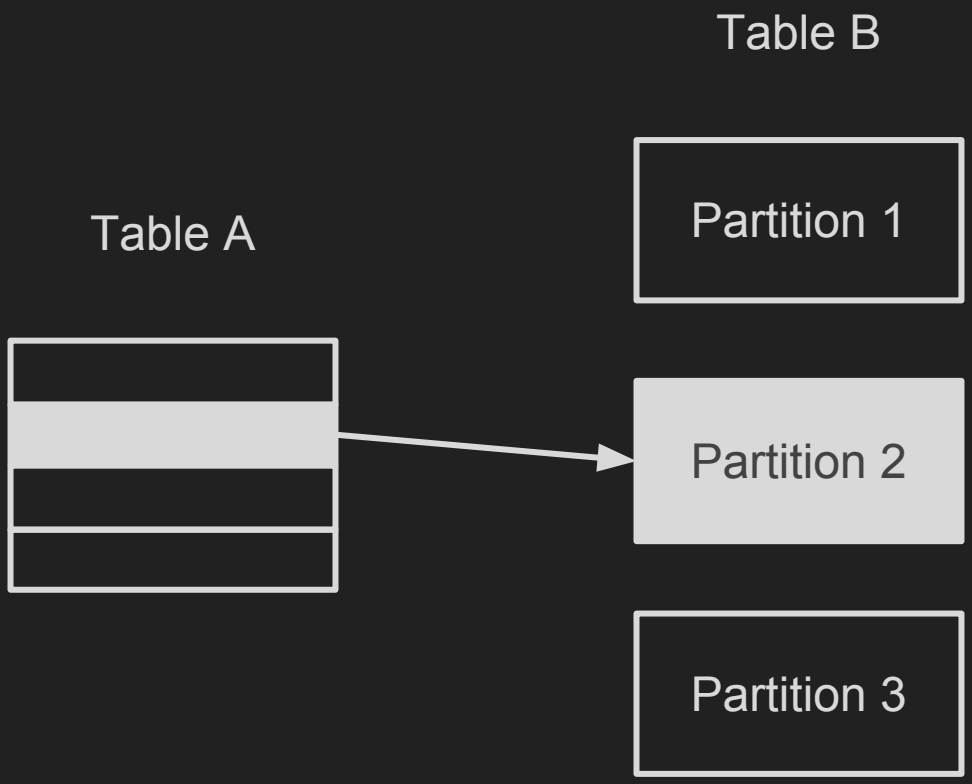
Execution example

Nested loop



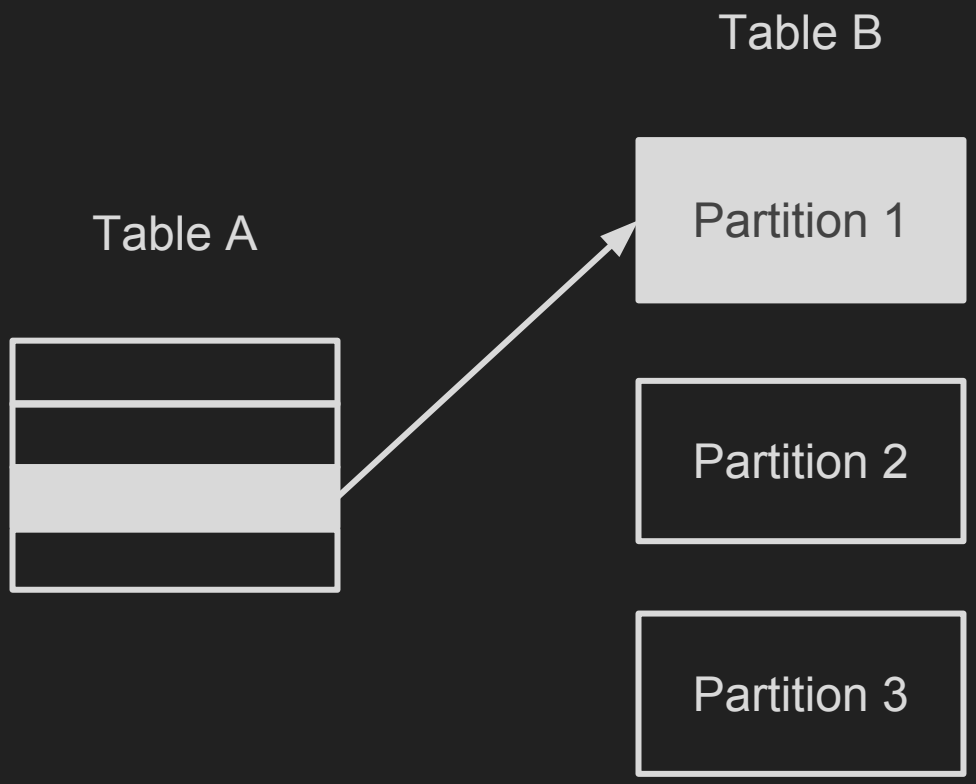
Execution example

Nested loop with pg_pathman



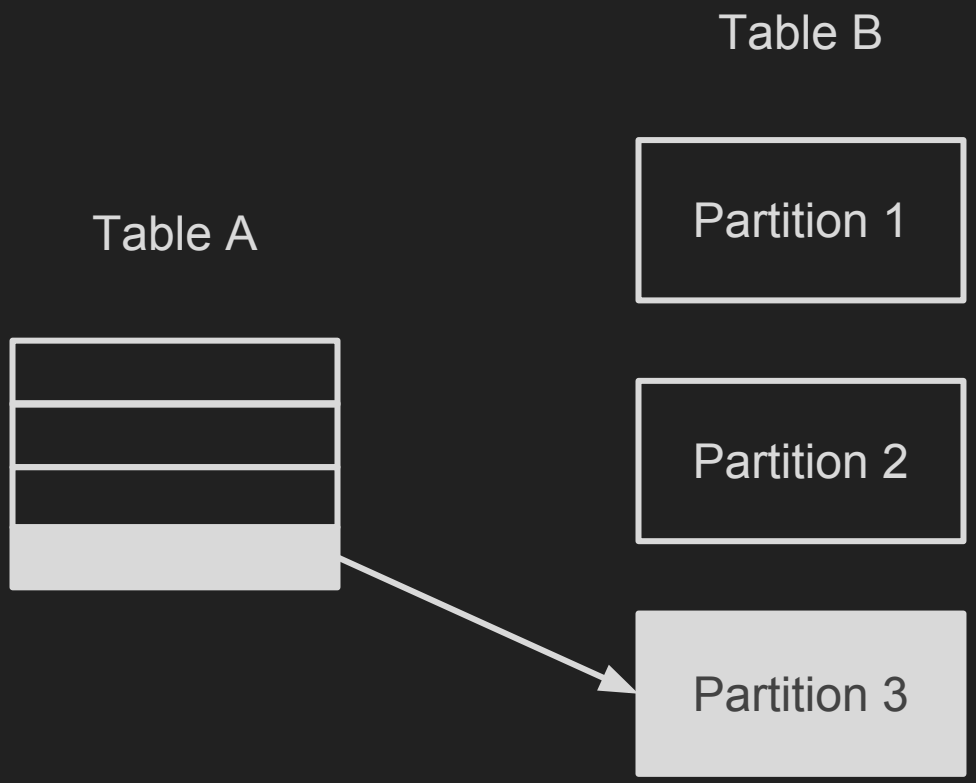
Execution example

Nested loop with pg_pathman



Execution example

Nested loop with pg_pathman



pg_pathman's API

```
SELECT create_hash_partitions(
    'abc',                  -- relation name
    'id',                   -- partitioning key
     3                      -- number of partitions
);

SELECT create_range_partitions(
    'abc',                  -- relation name
    'dt',                   -- partitioning key
    '2016-01-01'::date,     -- start value
    '1 month'::interval     -- interval
);
```

pg_pathman's API

Partition management

```
SELECT append_range_partition('abc');
SELECT prepend_range_partition('abc');
SELECT add_range_partition('abc',
                           '2016-05-01'::date,
                           '2016-06-01'::date);

SELECT attach_range_partition('abc', 'some_table',
                              '2016-06-01'::date,
                              '2016-07-01'::date);

SELECT merge_range_partitions('abc_1', 'abc_2');
SELECT split_range_partition('abc_1', '2016-02-01'::date);

SELECT detach_range_partition('some_table');
SELECT replace_hash_partition('abc_0', 'some_table');
SELECT drop_range_partition('abc_0', true);
SELECT drop_partitions('parent_table', false);
```

pg_pathman's API

Partition management

```
SELECT parent, partition, range_min, range_max,
       pg_size_pretty(pg_relation_size(partition)) AS size
FROM pathman_partition_list;
```

```
   parent   |  partition   | range_min | range_max | size
------------+--------------+-----------+-----------+------------
 test_hash  | test_hash_0  |           |           | 96 kB
 test_hash  | test_hash_1  |           |           | 88 kB
 test_hash  | test_hash_2  |           |           | 96 kB
 test_hash  | test_hash_3  |           |           | 96 kB
 test_range | test_range_1 | 1         | 2001      | 72 kB
 test_range | test_range_2 | 2001      | 4001      | 72 kB
 test_range | test_range_3 | 4001      | 6001      | 72 kB
 test_range | test_range_4 | 6001      | 8001      | 72 kB
 test_range | test_range_5 | 8001      |           | 8192 bytes
```

pg_pathman's API

Partition management

```
SELECT drop_range_partition(partition), range_min, range_max
FROM pathman_partition_list
WHERE parent = 'test_range'::regclass AND
      COALESCE(range_max::float, '+inf'::float) > 6000;
```

```
 drop_range_partition | range_min | range_max
----------------------+-----------+-----------
 test_range_3         | 4001      | 6001
 test_range_4         | 6001      | 8001
 test_range_5         | 8001      |
```

pg_pathman's API

Data migration

```
SELECT partition_table_concurrently('test');
SELECT stop_concurrent_part_task('test');
```

```
SELECT * FROM pathman_concurrent_part_tasks;

 userid | pid  | dbid  | relid | processed | status
--------+------+-------+-------+-----------+---------
 user   | 7367 | 16384 | test  |    472000 | working
(1 row)
```

pg_pathman's API

Additional parameters

```
SELECT set_interval      ('abc', '2 months');
SELECT set_enable_parent ('abc', true);
SELECT set_auto          ('abc', true);
SELECT set_init_callback ('abc', 'my_func(jsonb)');
```

pg_pathman's API

User-defined callbacks (partition creation)

```
CREATE FUNCTION my_callback(args JSONB)
```

Example #1: custom names for partitions

```
CREATE OR REPLACE FUNCTION my_callback(params jsonb)
RETURNS VOID AS
$$
DECLARE
    range_min    timestamp;
    new_relname  text;
BEGIN
    range_min := (params->>'range_min')::timestamp;

    -- generate new name for partition based
    -- on its parent name and lower bound
    new_relname := format('%s_%s',
                          params->>'parent',
                          to_char(range_min, 'YYYY_MM'));

    -- rename partition
    EXECUTE format('ALTER TABLE %s.%s RENAME TO %s',
                   params->>'partition_schema',
                   params->>'partition',
                   new_relname);
END
$$ LANGUAGE plpgsql;
```

Example #2: data rotation

```
CREATE OR REPLACE FUNCTION rotation_callback(params jsonb)
RETURNS VOID AS
$$
DECLARE
    relation  regclass;
BEGIN
    -- drop "old" partitions
    FOR relation IN (SELECT partition FROM pathman_partition_list
                     WHERE parent = (params->>'parent')::regclass
                     ORDER BY range_min::timestamp DESC
                     OFFSET 10)
    LOOP
        RAISE NOTICE 'dropping partition %', relation;
        SELECT drop_range_partition(relation);
    END LOOP;
END
$$ LANGUAGE plpgsql;
```

PartitionFilter

BEFORE:

```
EXPLAIN (COSTS OFF)
INSERT INTO partitioned_table
SELECT generate_series(1, 10), random();
               QUERY PLAN
----------------------------------------
 Insert on partitioned_table
   ->  Subquery Scan on "*SELECT*"
         -> Result
```

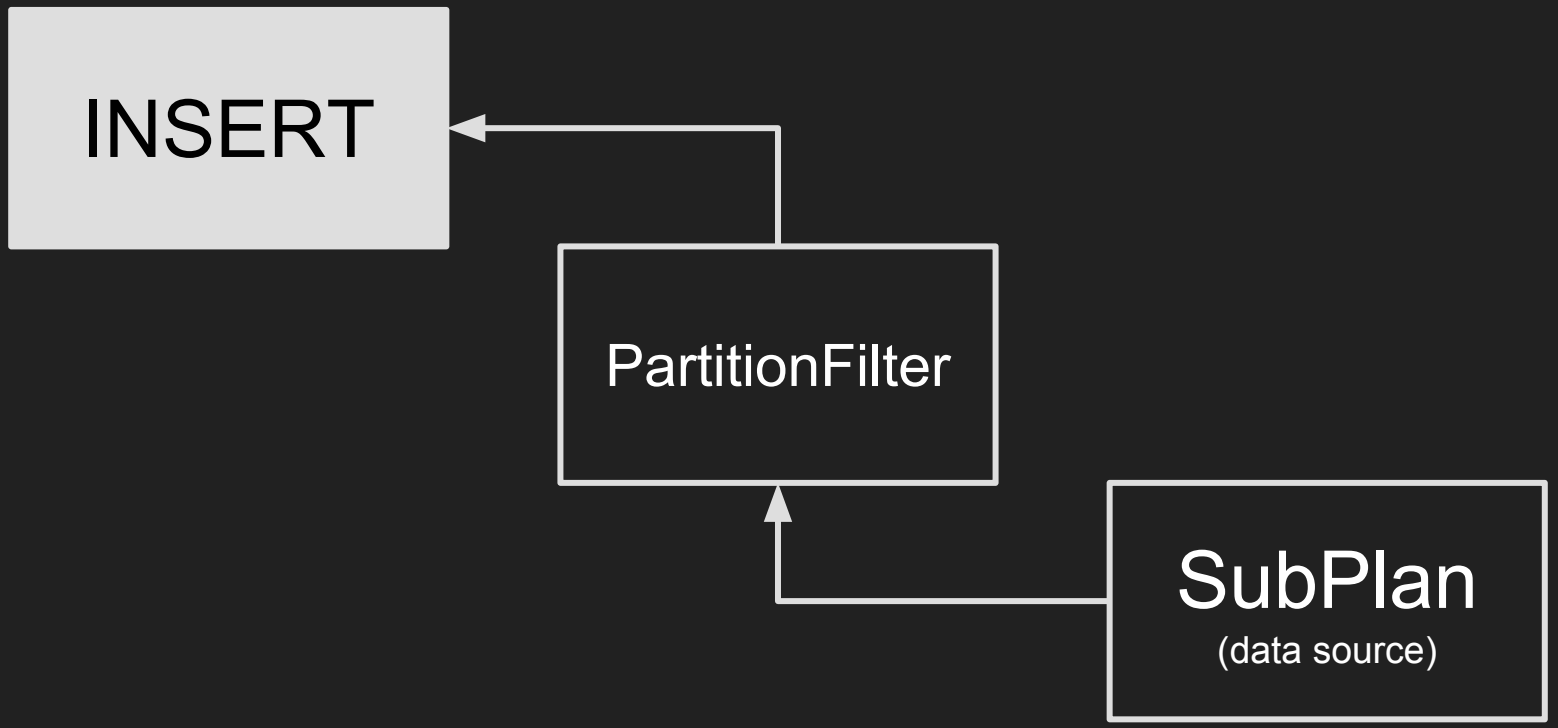
AFTER:

```
EXPLAIN (COSTS OFF)
INSERT INTO partitioned_table
SELECT generate_series(1, 10), random();
               QUERY PLAN
----------------------------------------
 Insert on partitioned_table
   ->  Custom Scan (PartitionFilter)
         ->  Subquery Scan on "*SELECT*"
               -> Result
```

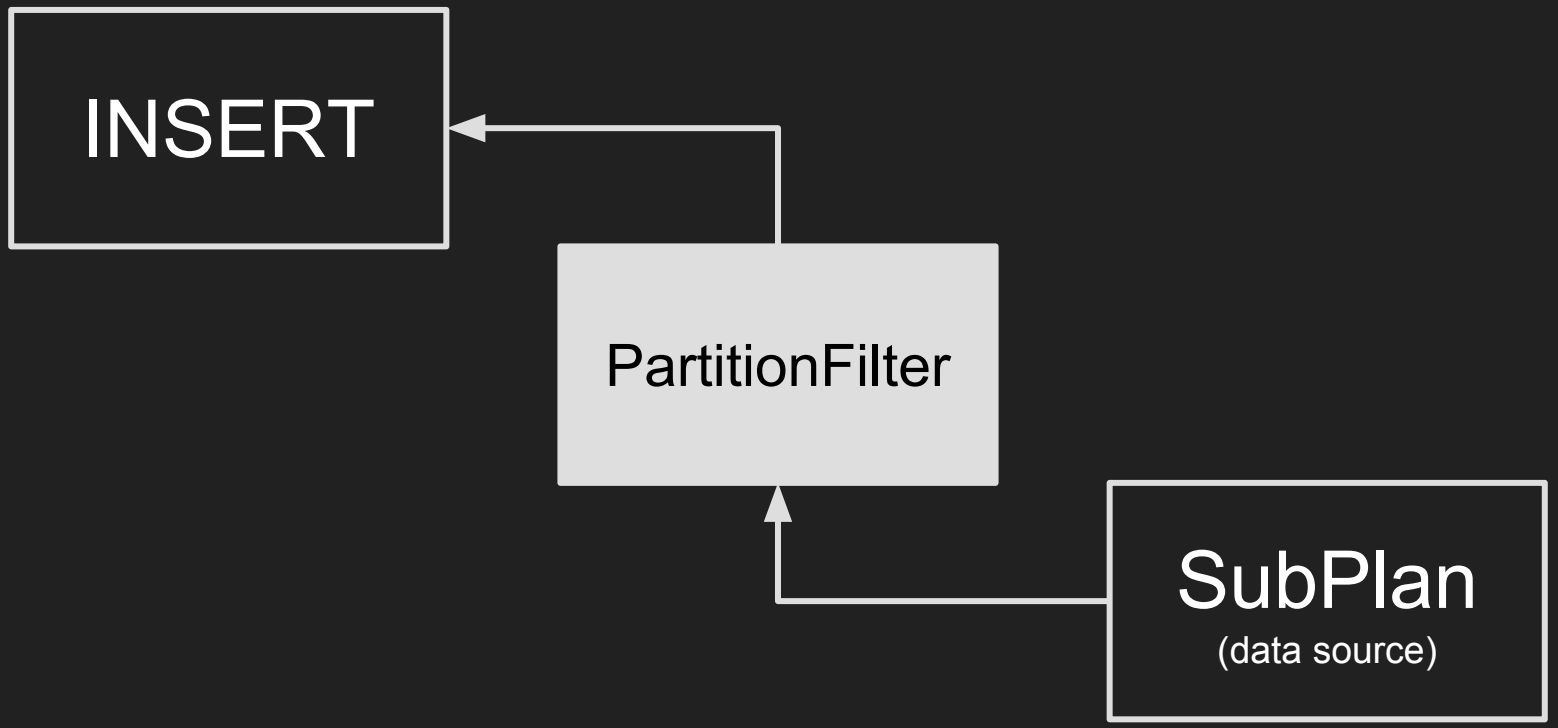
PartitionFilter

- Fast and efficient INSERTs
- Supports RETURNING *
- Shows total amount of inserted rows
- Supports BEFORE / AFTER / FOR EACH ROW triggers

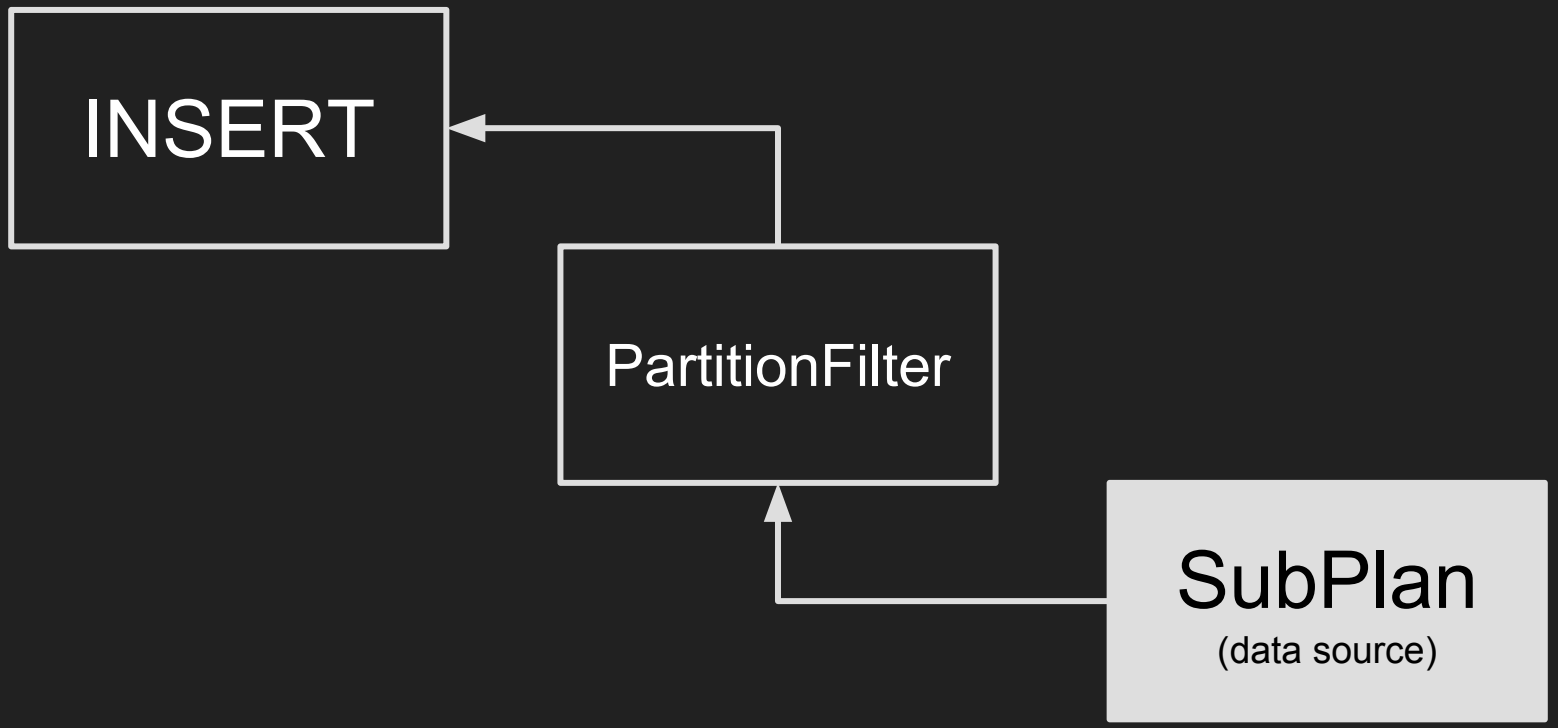
PartitionFilter



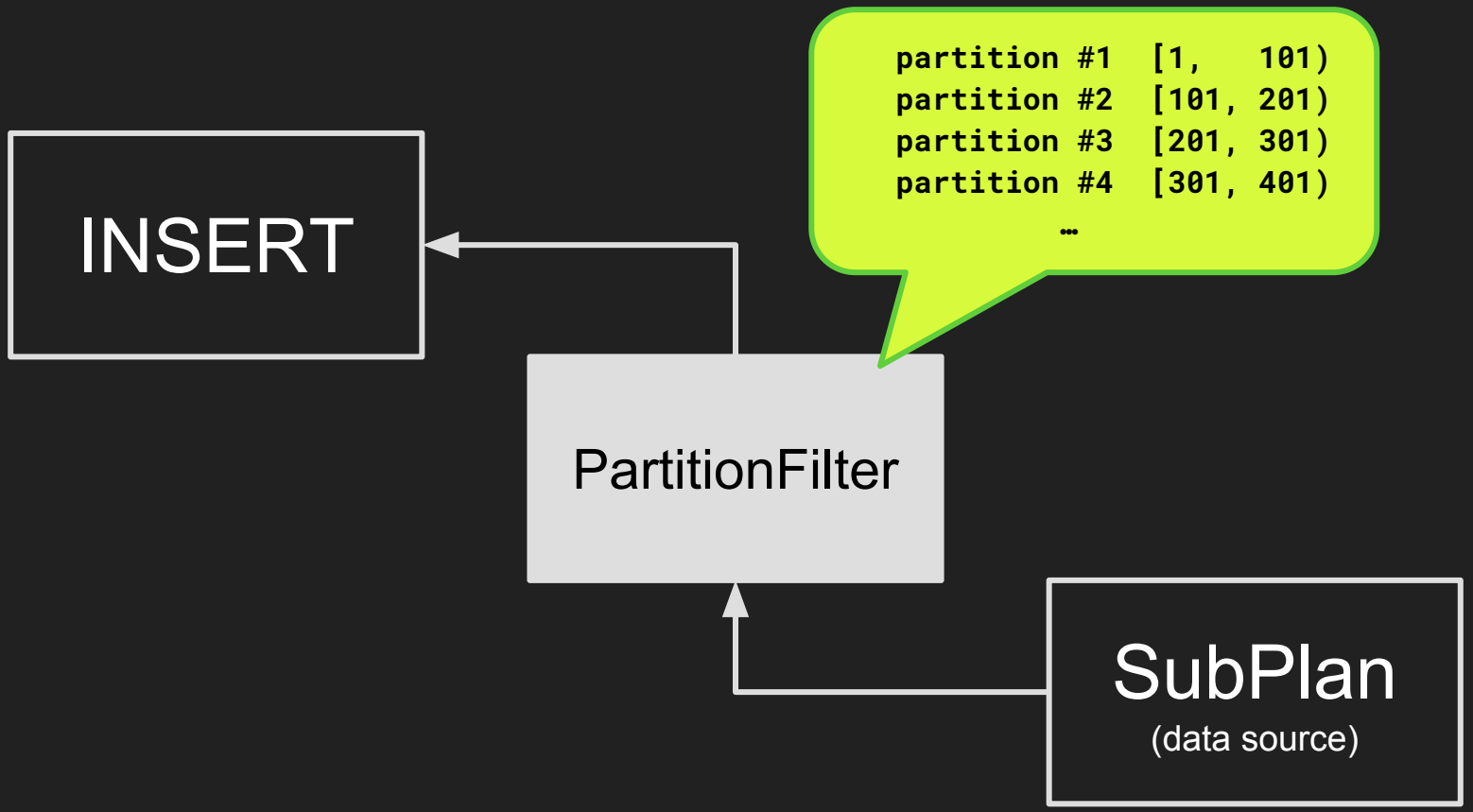
PartitionFilter



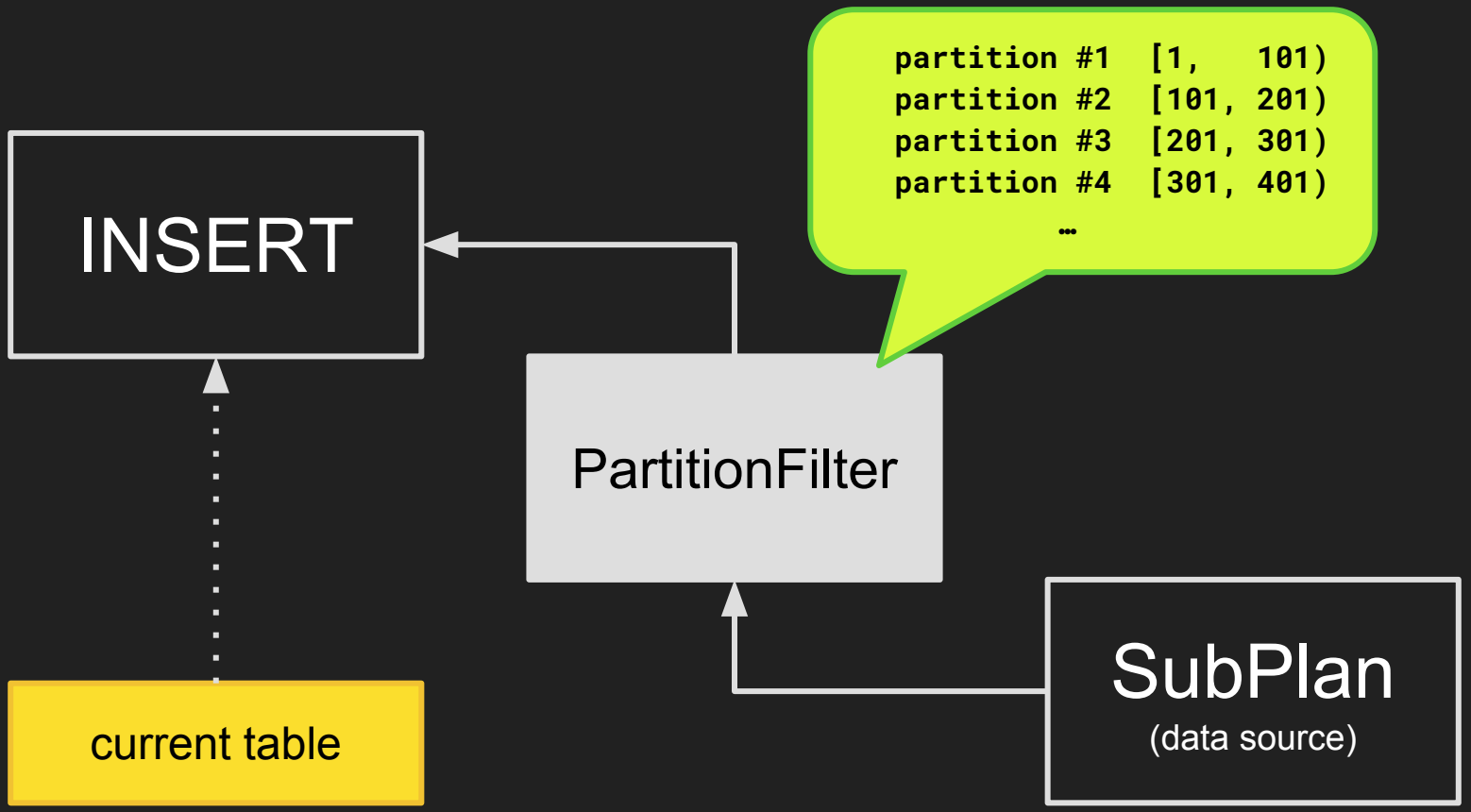
PartitionFilter



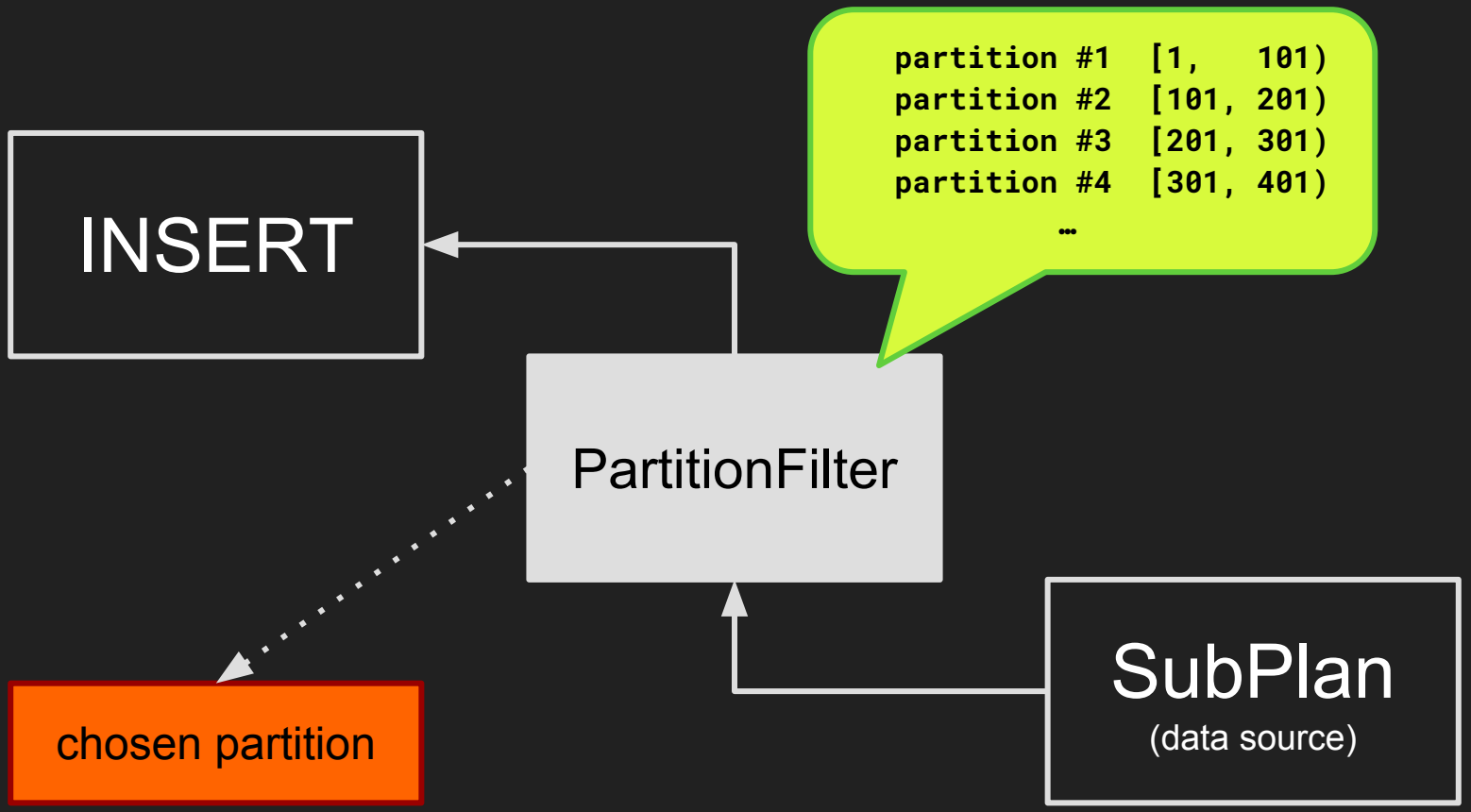
PartitionFilter



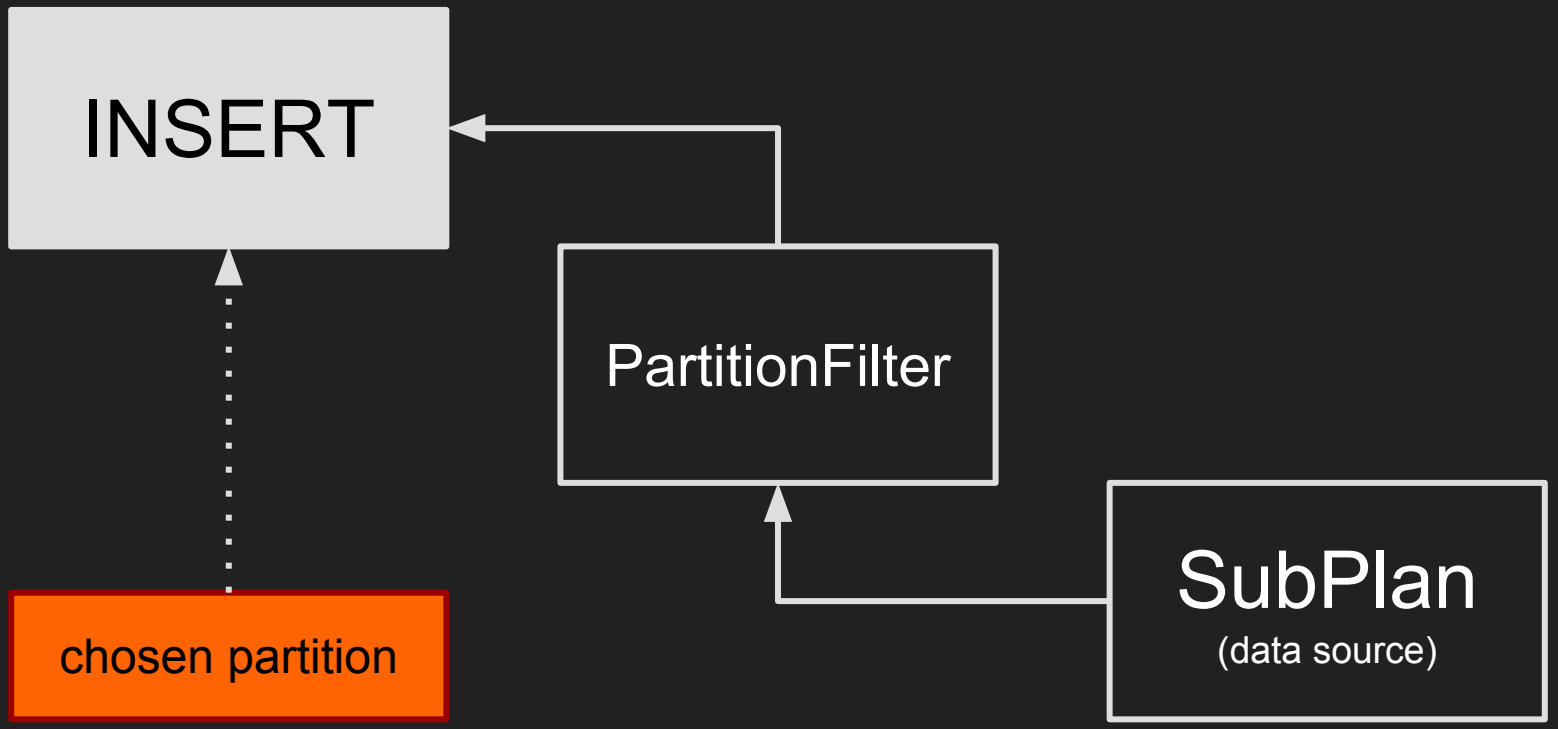
PartitionFilter



PartitionFilter



PartitionFilter

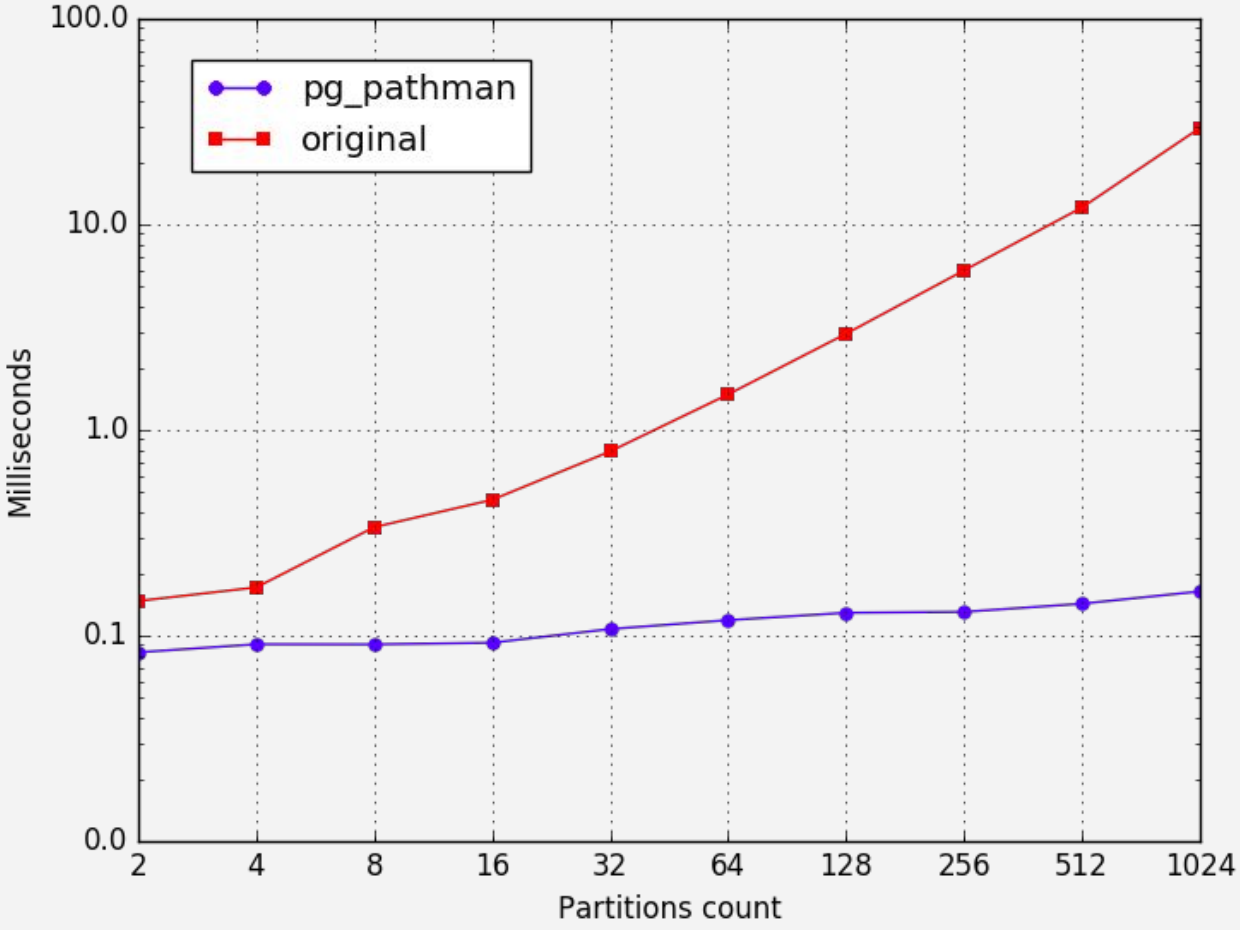


```
INSERT INTO journal (dt, level, msg)
VALUES ('2016-12-31', random(), 'test')
RETURNING *;

  id     |          dt           | level | msg
---------+-----------------------+-------+------
 1051202 | 2016-12-31 00:00:00   |     0 | test
(1 row)

INSERT 0 1
```

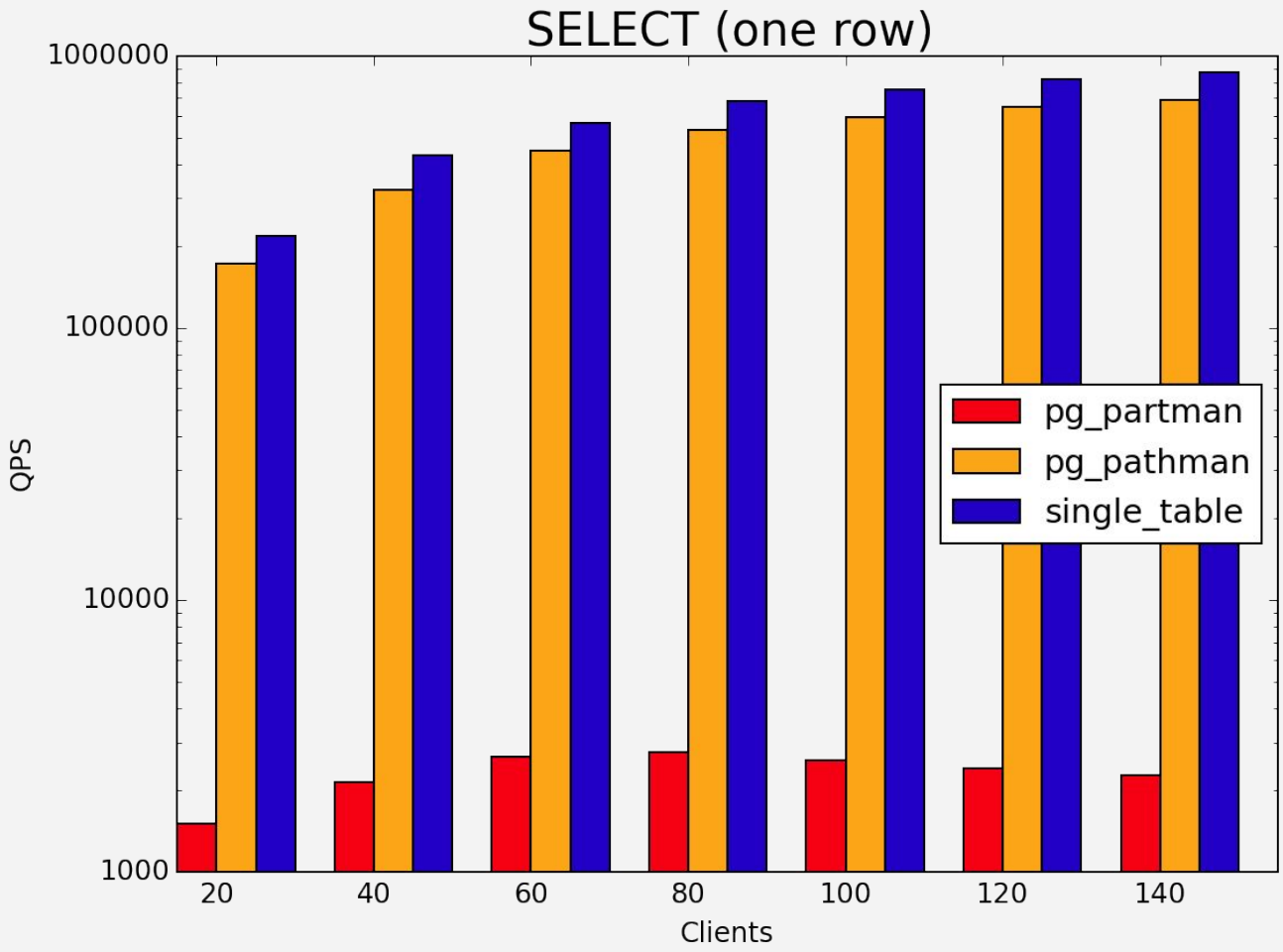
Benchmarks
planning time



Benchmarks

366
partitions

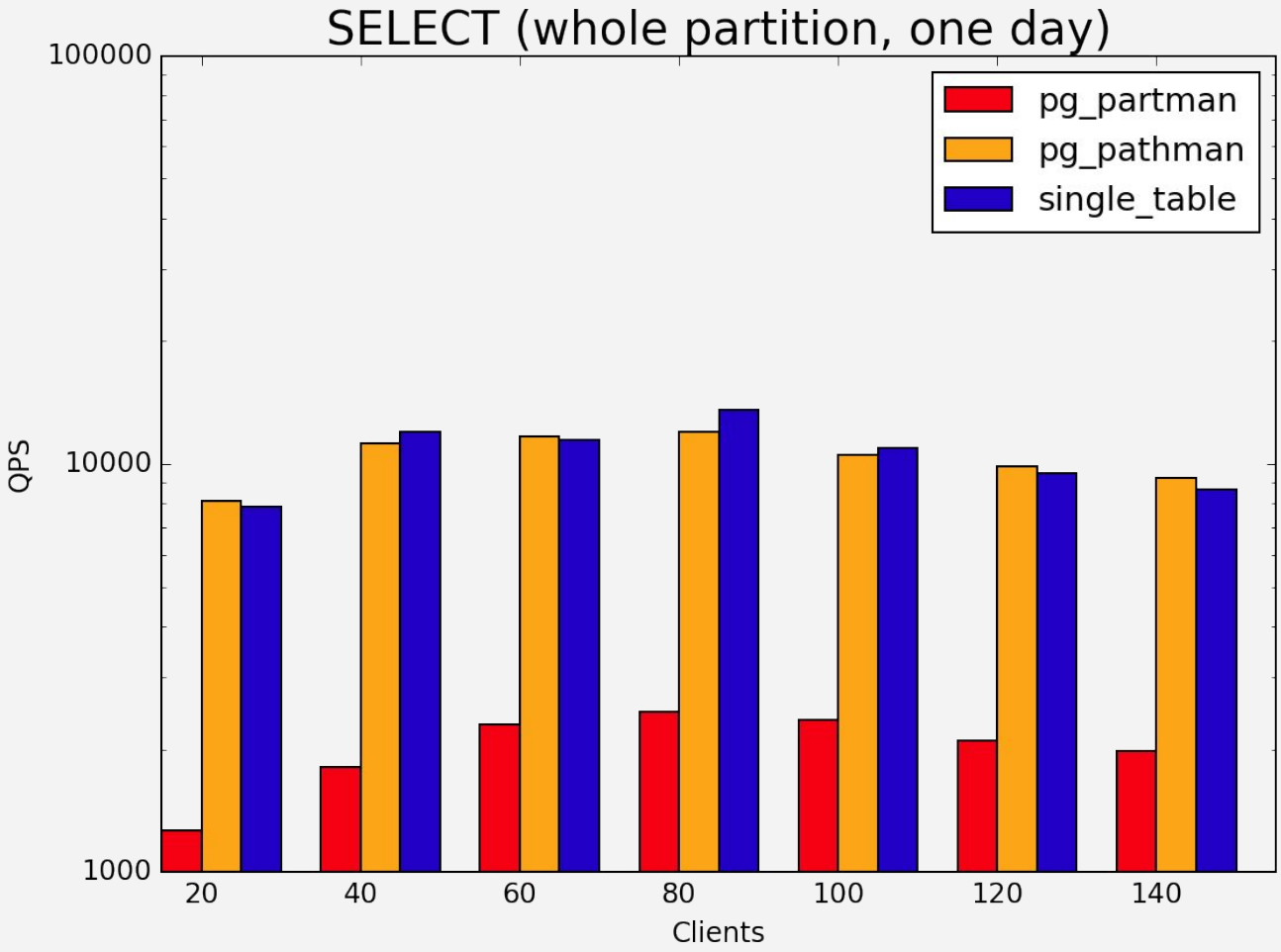
1KK
rows



Benchmarks

366
partitions

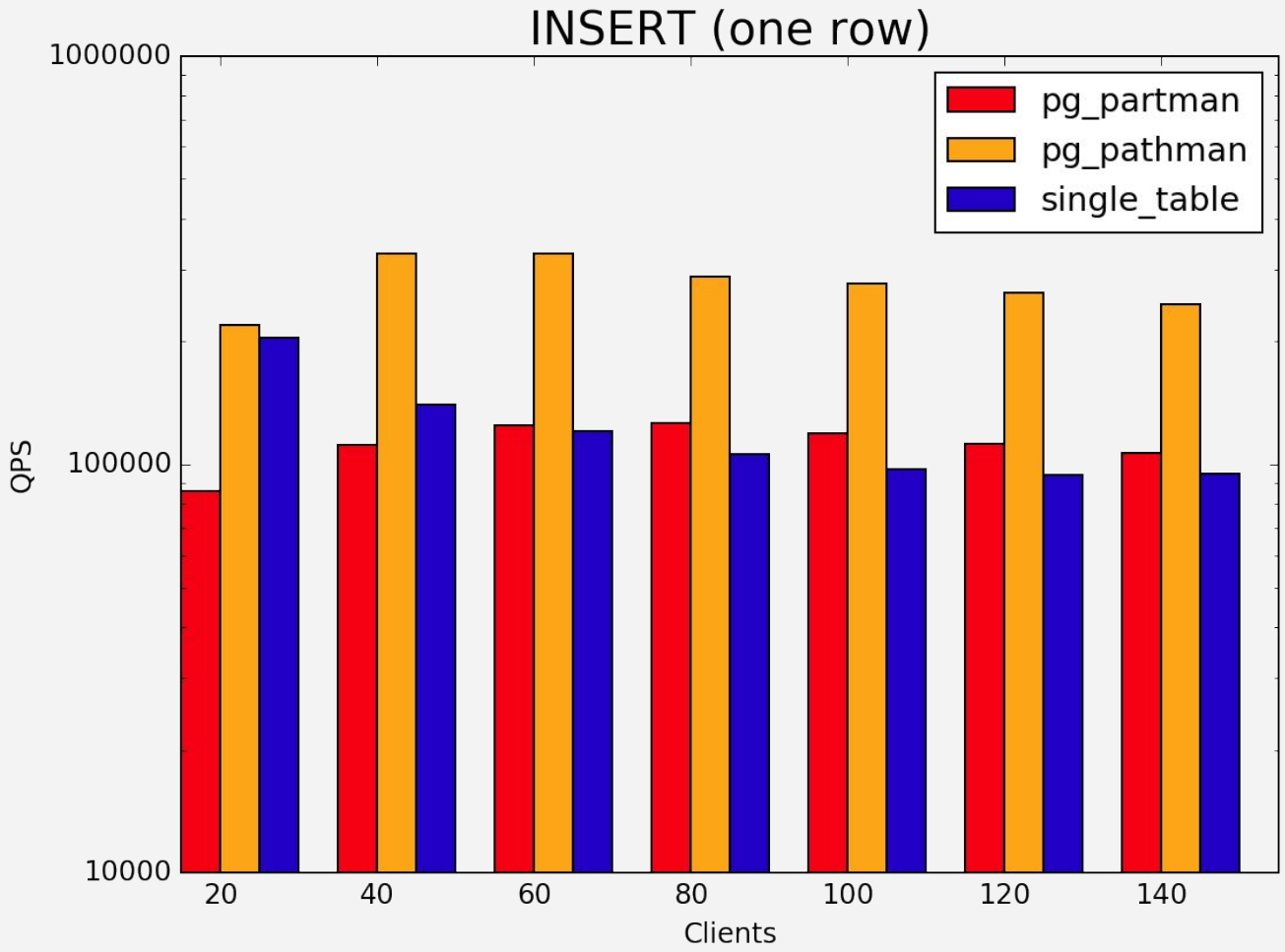
1KK
rows



Benchmarks

366
partitions

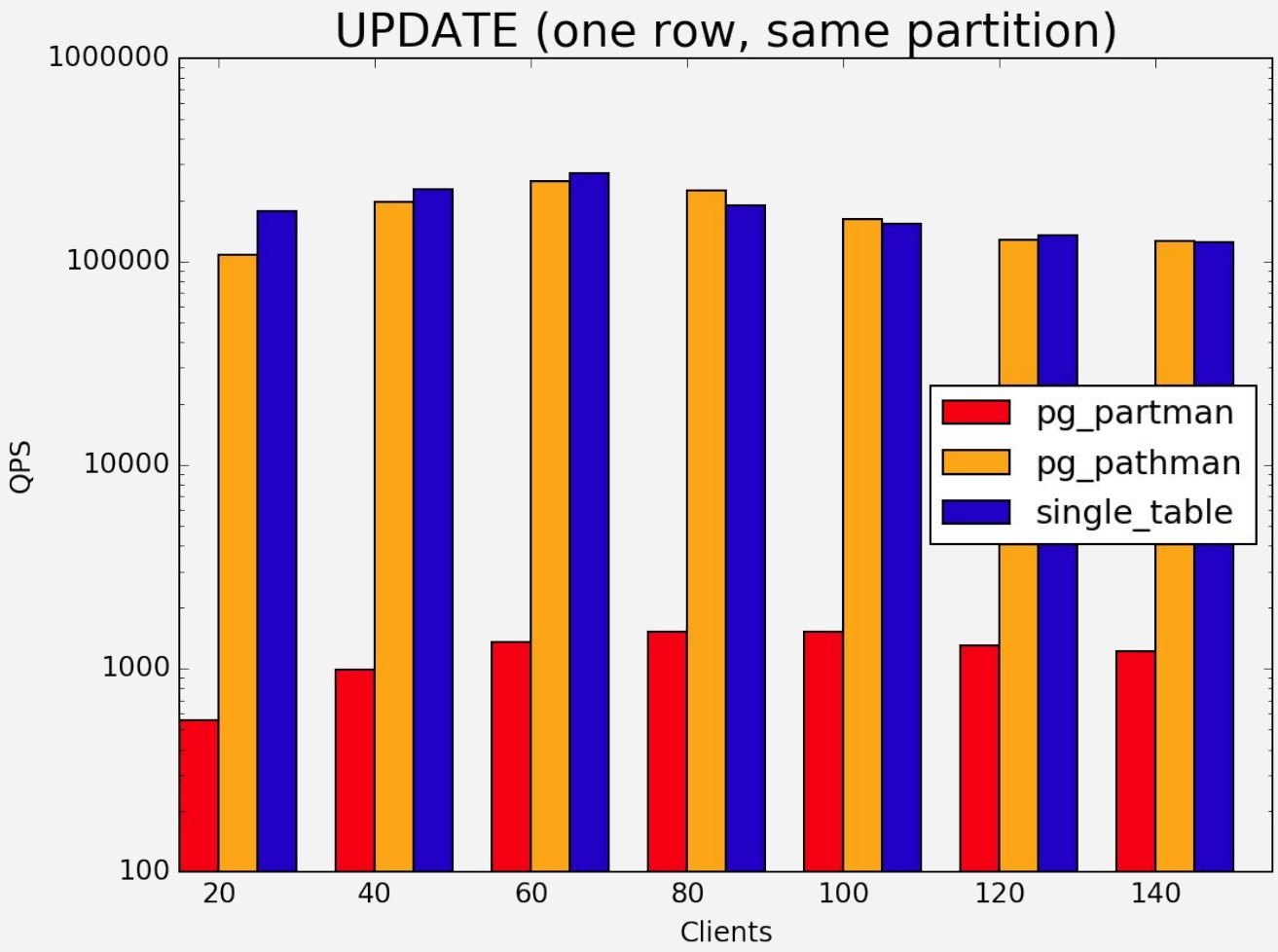
1KK
rows



Benchmarks

366
partitions

1KK
rows



Roadmap

- expression-based partitioning key
- multi level partitioning
- custom nodes for UPDATE / DELETE
- **LIST** partitioning
- more JOIN optimizations

PostgresPro EE

Oracle-like partitioning syntax in PostgresPro EE

```
CREATE TABLE clients (
    id     SERIAL PRIMARY KEY,
    name   TEXT,
    email  TEXT)
PARTITION BY HASH (id)
(
    PARTITION clients_1 TABLESPACE ts1,
    PARTITION clients_2 TABLESPACE ts2,
    PARTITION clients_3 TABLESPACE ts3
);
```

```
CREATE TABLE log (
    ts     TIMESTAMP NOT NULL,
    level  INTEGER,
    msg    TEXT)
PARTITION BY RANGE (ts) INTERVAL ('1 month')
(
    PARTITION log_archive VALUES LESS THAN ('2017-01-01'),
    PARTITION log_1       VALUES LESS THAN ('2017-02-01'),
    PARTITION log_2       VALUES LESS THAN ('2017-03-01')
);
```

Oracle-like partitioning syntax in PostgresPro EE

```
ALTER TABLE log ADD  PARTITION log_3 VALUES LESS THAN ('2017-04-01');
ALTER TABLE log DROP PARTITION log_3;

ALTER TABLE log MERGE PARTITIONS log_1, log_2 INTO PARTITION log_1;
ALTER TABLE log SPLIT PARTITION log_1 AT ('2017-02-01') INTO (
    PARTITION log_1_left,
    PARTITION log_1_right
);

ALTER TABLE log RENAME PARTITION log_3 TO log_march;
ALTER TABLE log MOVE PARTITION log_archive TABLESPACE archive_ts;

ALTER TABLE log SET INTERVAL ('2 months');
```

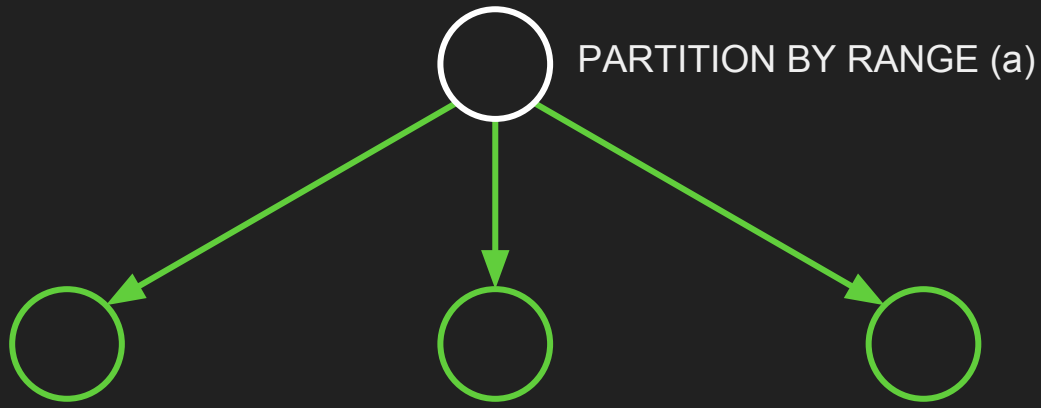
Partitioning syntax in PostgresPro EE

```
ALTER TABLE clients PARTITION BY HASH (id)
PARTITIONS (3)
CONCURRENTLY; -- optional
```

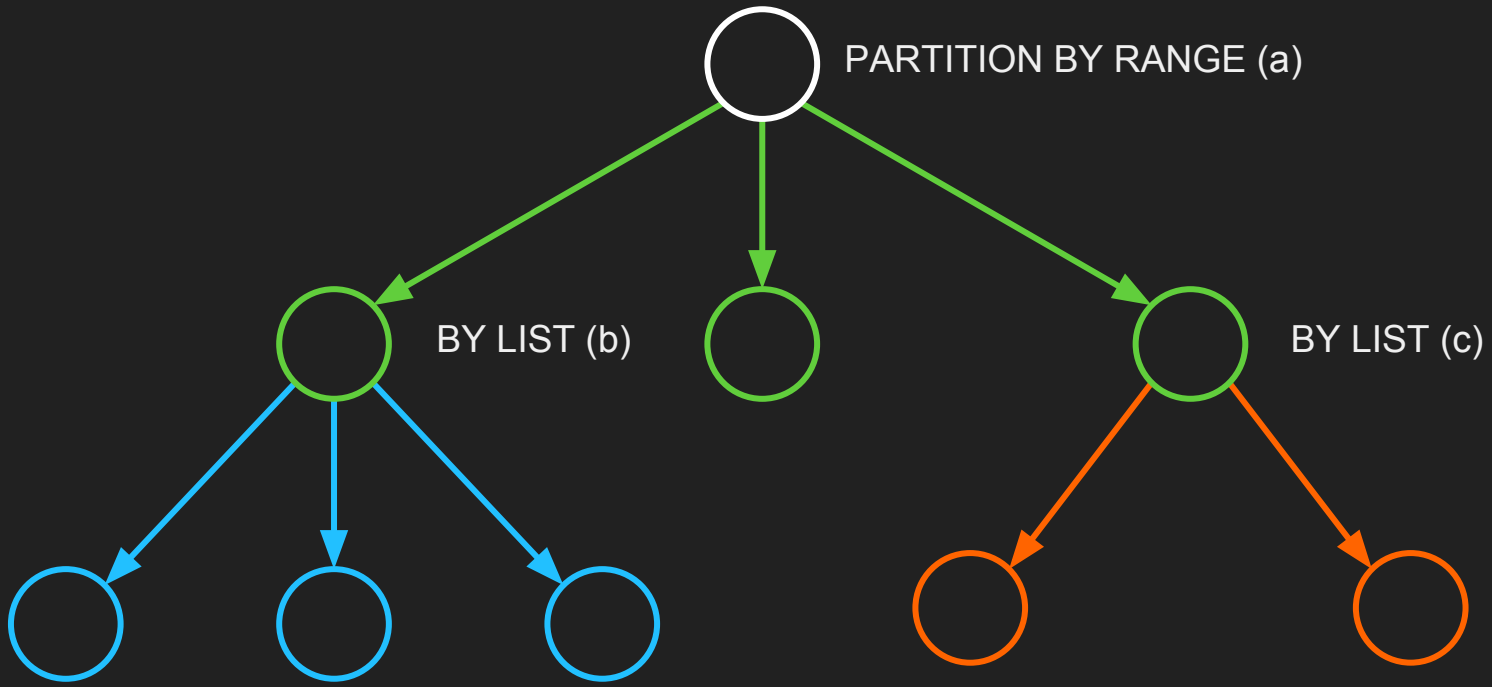
```
ALTER TABLE log PARTITION BY RANGE (ts)
START FROM ('2017-01-01')
INTERVAL ('1 month')
CONCURRENTLY; -- optional
```

PostgreSQL 10

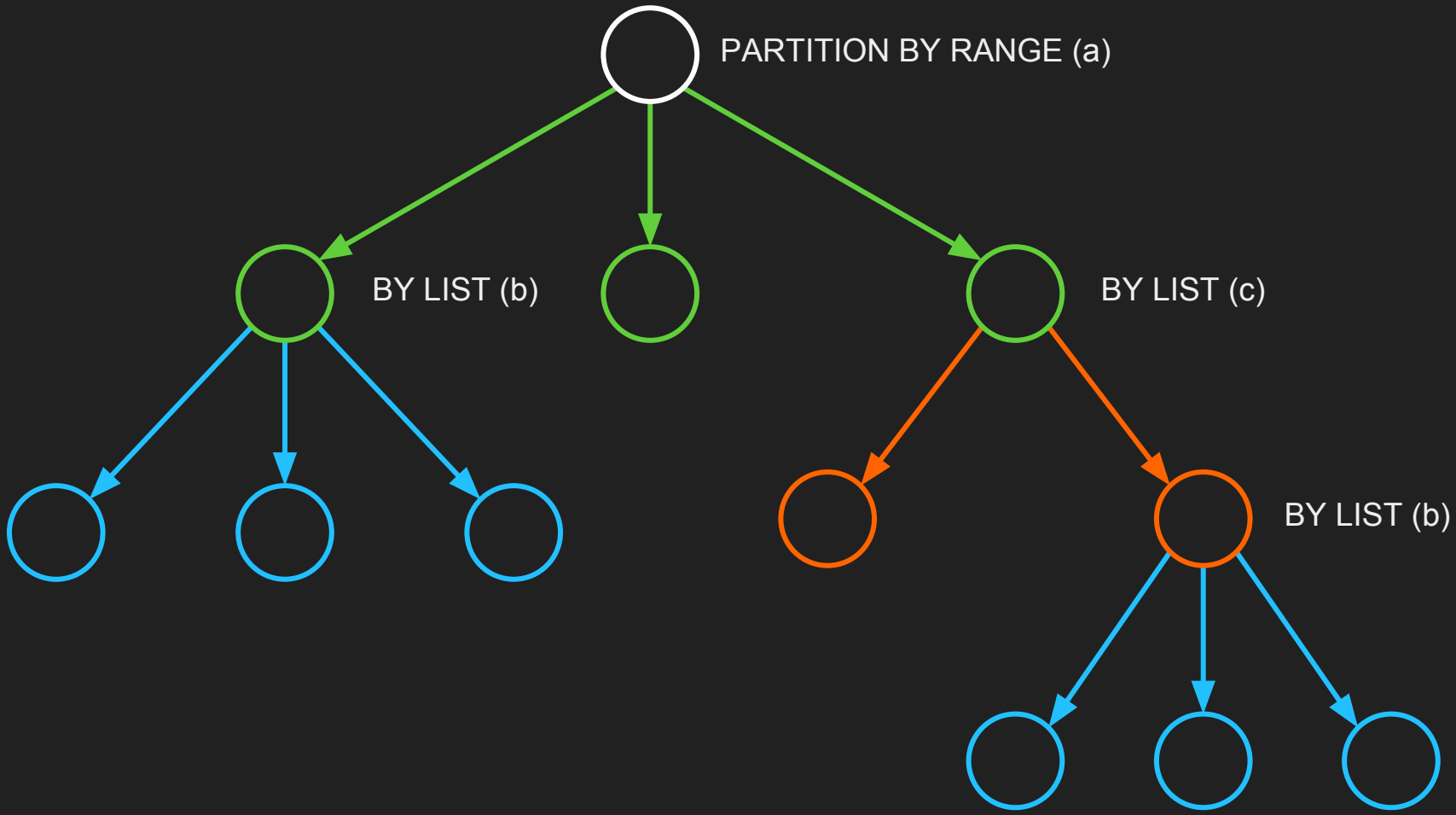
Declarative partitioning in PostgreSQL 10



Declarative partitioning in PostgreSQL 10



Declarative partitioning in PostgreSQL 10



Declarative partitioning in PostgreSQL 10

```
CREATE TABLE log (
    ts     TIMESTAMP,
    level  INTEGER,
    code   INTEGER,
    msg    TEXT)
PARTITION BY RANGE (ts);
```

Declarative partitioning in PostgreSQL 10

```
CREATE TABLE log (
    ts     TIMESTAMP,
    level  INTEGER,
    code   INTEGER,
    msg    TEXT)
PARTITION BY RANGE (ts);

CREATE TABLE log_17_01 PARTITION OF log
FOR VALUES FROM ('2017-01-01') TO ('2017-02-01');
```

Declarative partitioning in PostgreSQL 10

```
CREATE TABLE log (
    ts     TIMESTAMP,
    level  INTEGER,
    code   INTEGER,
    msg    TEXT)
PARTITION BY RANGE (ts);

CREATE TABLE log_17_01 PARTITION OF log
FOR VALUES FROM ('2017-01-01') TO ('2017-02-01');

CREATE TABLE log_17_02 PARTITION OF log
FOR VALUES FROM ('2017-02-01') TO ('2017-03-01')
PARTITION BY LIST (code);
```

Declarative partitioning in PostgreSQL 10

```
CREATE TABLE log (
    ts     TIMESTAMP,
    level  INTEGER,
    code   INTEGER,
    msg    TEXT)
PARTITION BY RANGE (ts);

CREATE TABLE log_17_01 PARTITION OF log
FOR VALUES FROM ('2017-01-01') TO ('2017-02-01');

CREATE TABLE log_17_02 PARTITION OF log
FOR VALUES FROM ('2017-02-01') TO ('2017-03-01')
PARTITION BY LIST (code);

CREATE TABLE log_17_02_errors   PARTITION OF log_17_02 FOR VALUES IN (1);
CREATE TABLE log_17_02_warnings PARTITION OF log_17_02 FOR VALUES IN (2);
```

Declarative partitioning in PostgreSQL 10

PROS

- supportable kinds of partitioning: **RANGE**, **LIST** (**HASH** is still on commit fest)
- multilevel partitioning
- tuple routing on INSERT without triggers
- partitioning by expression & composite key

CONS

- partition pruning through constraint exclusion mechanism
- INSERTs lock whole partition tree
- lack of global indexes ⇒ no UNIQUE, PRIMARY KEY and FOREIGN KEY
- no runtime optimization
- no easy way to partition existing tables

Thank you for your attention!

github.com/postgrespro/pg_pathman

Ildar Musin i.musin@postgrespro.ru
Dmitry Ivanov d.ivanov@postgrespro.ru