Improving Postgres' Efficiency

Andres Freund
PostgreSQL Developer & Committer

Email: andres@anarazel.de
Email: andres.freund@enterprisedb.com
Twitter: @AndresFreundTec

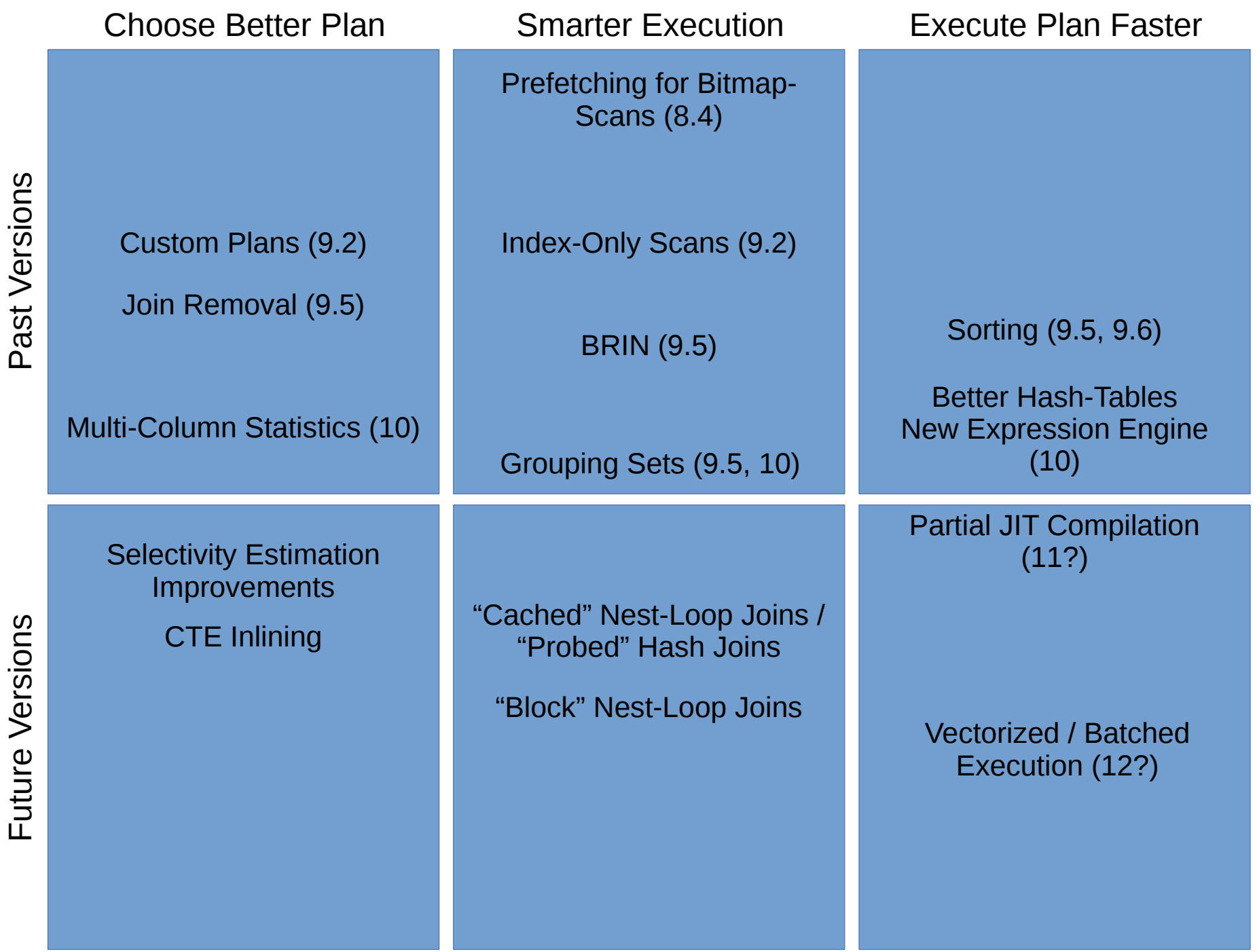
EnterpriseDB

anarazel.de/talks/pgopen-sf-2017-09-07/efficiency.pdf

EDB ENTERPRISEDB

Efficiency

Analytics



EDB ENTERPRISEDB

The Past

Choose A Better Plan: Multi-Column Statistics

- Selectivity Estimation
- Two Column Selectivity: likelihood(a) * likelihood(b)
- Correlation / Dependencies:
  - WHERE city = 'San Francisco' AND zipcode = '94102'
  - city = 'San Francisco' => 864k / 301M => 1/348
  - zipcode = '94102' => 31k / 301M => 1/9654
  - Result: 301M * (1/348 * 1/9654) => ~90
- Fix in PostgreSQL 10+:
  CREATE STATISTICS zips_and_cities
  ON (zipcode, city)
  FROM us_citizens;

Smarter Execution: Index-Only Scan

- ```
  SELECT l_shipdate, count(*)
  FROM lineitem
  WHERE l_shipdate BETWEEN '...' AND '...'
  GROUP BY l_shipdate
  ORDER BY count(*)
  LIMIT 1;
  CREATE INDEX "i_l_shipdate" ON lineitem(l_shipdate);
  ```
- Bitmap-Scan: 109900.866 ms, 1099789 buffers accessed
- Index-Only-Scan: 1317.177 ms, 38470 buffers accessed
- Requires: Regular (auto-)vacuum, index over all columns
- PostgreSQL 11: “covering indexes”
  ```

Smarter Execution: Prefetching:

- SELECT l_shipdate, SUM(quantity),
  FROM lineitem
  WHERE l_shipdate BETWEEN '...' AND '...'
  GROUP BY l_shipdate
  ORDER BY sum(quantity) DESC
  LIMIT 1;
  CREATE INDEX "i_l_shipdate" ON lineitem(l_shipdate);
- Good on rotational disks, awesome on SSDs
- No-Prefetching: SET effective_io_concurrency = 0;
  Buffers: shared hit=3 read=1119735
  Time: 30032.860 ms
  Average-IO: ~260 MB/Sec
  Utilization: ~60%
- Prefetching: SET effective_io_concurrency = 512;
  Buffers: shared hit=3 read=1119735
  Time: 17688.256 ms
  Average-IO: ~525 MB/Sec
  Utilization: 100.00%

Faster Execution:
New Hash-Table & Expression Engine

```
SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) AS sum_qty,
    sum(l_extendedprice) AS sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    avg(l_quantity) AS avg_qty,
    avg(l_extendedprice) AS avg_price,
    avg(l_discount) AS avg_disc,
    count(*) AS count_order
FROM lineitem
WHERE l_shipdate <= date '1998-12-01' - interval '74 days'
GROUP BY l_returnflag, l_linestatus
ORDER BY l_returnflag, l_linestatus;
```

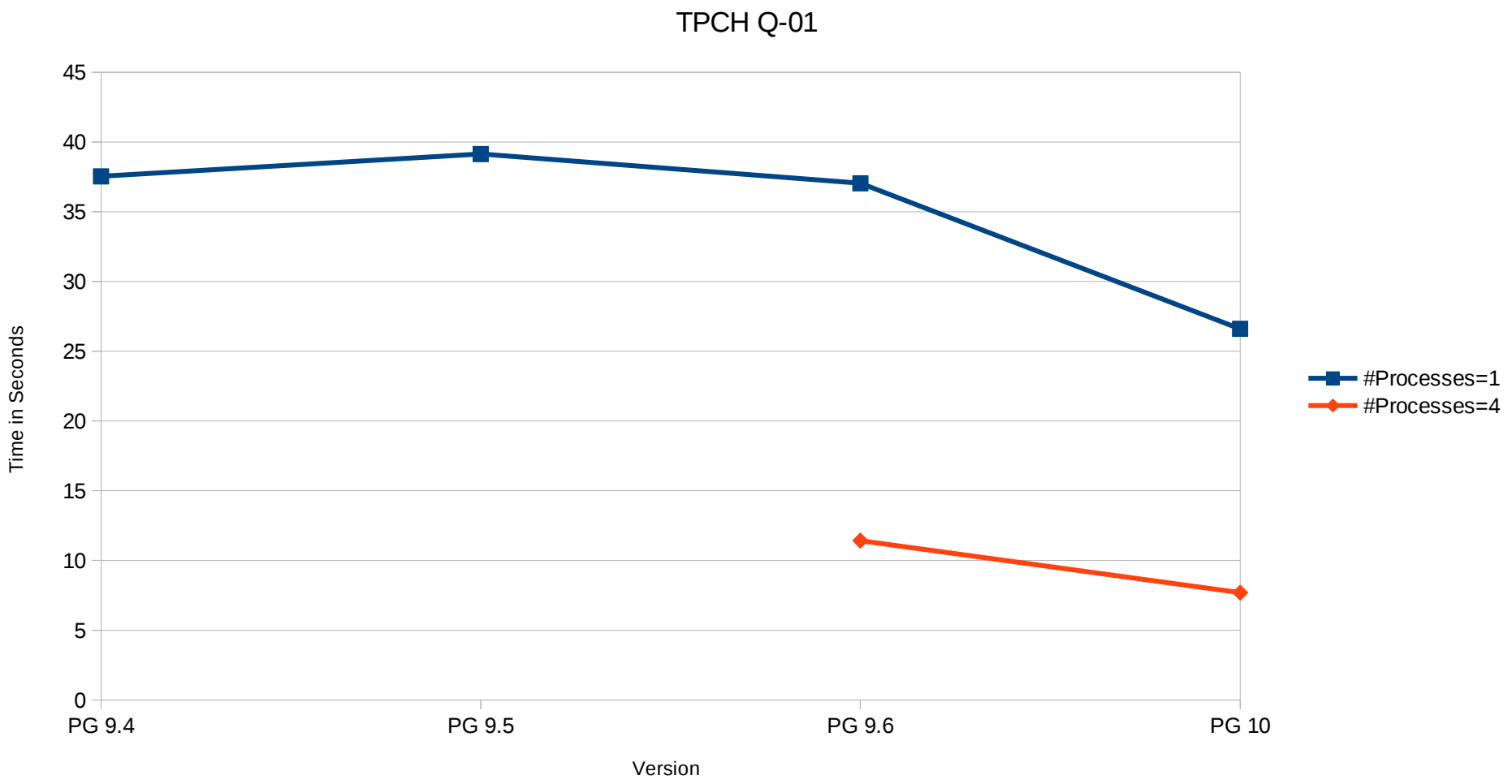
EDB ENTERPRISEDB

Faster Execution:
New Hash-Table & Expression Engine

```
Sort  (cost=4313533.34..4313533.36 rows=6 width=68)
  Sort Key: l_returnflag, l_linestatus
  ->  HashAggregate  (cost=4313533.16..4313533.26 rows=6 width=68)
        Group Key: l_returnflag, l_linestatus
        Output: …, sum(l_quantity), sum(l_extendedprice), sum(…), ...
        ->  Seq Scan on lineitem  (cost=0.00..1936427.80 rows=59427634 width=36)
              Filter: (l_shipdate <= '1998-09-18 00:00:00'::timestamp without time zone)
```

EDB ENTERPRISEDB

Faster Execution:
New Hash-Table & Expression Engine



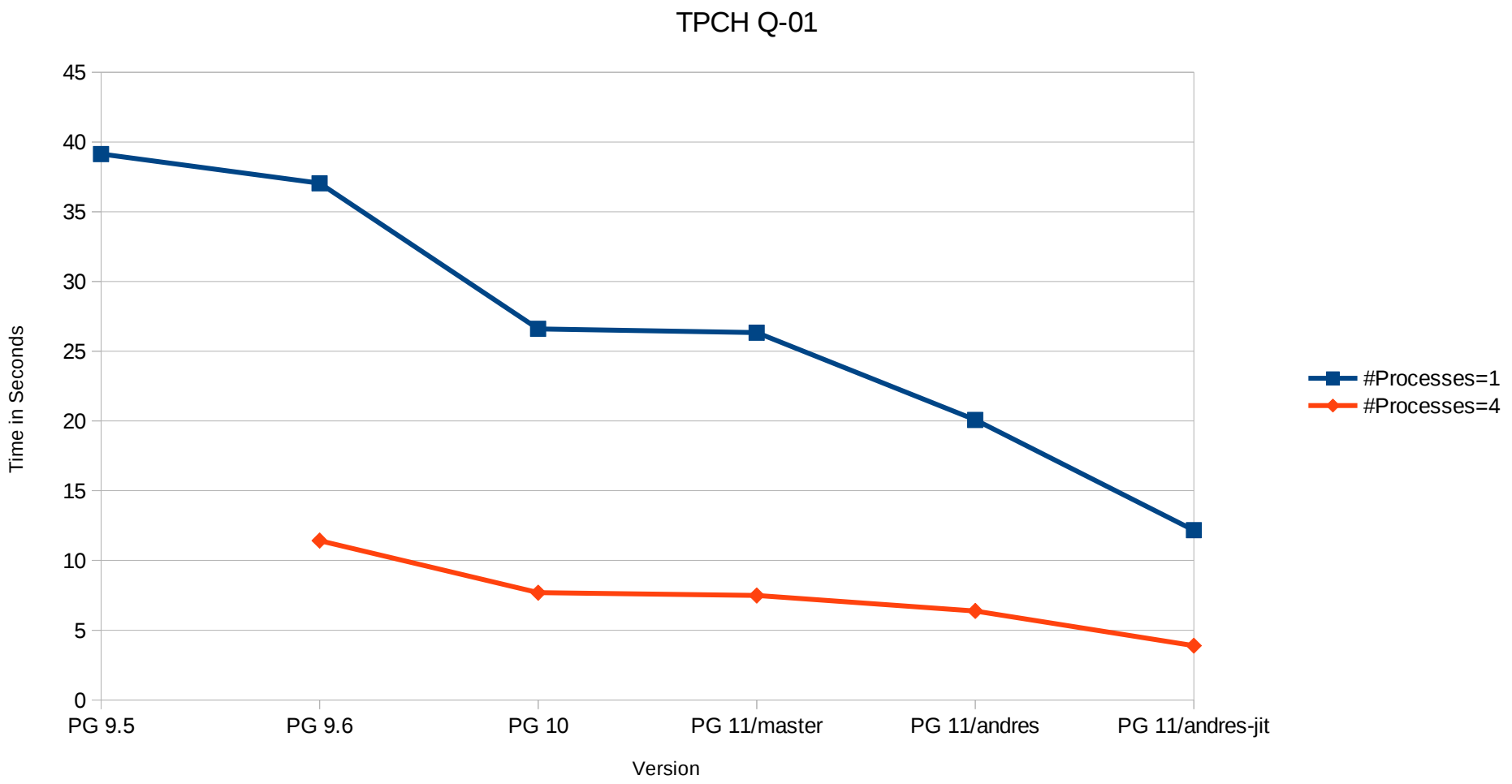
EDB ENTERPRISEDB

The Future

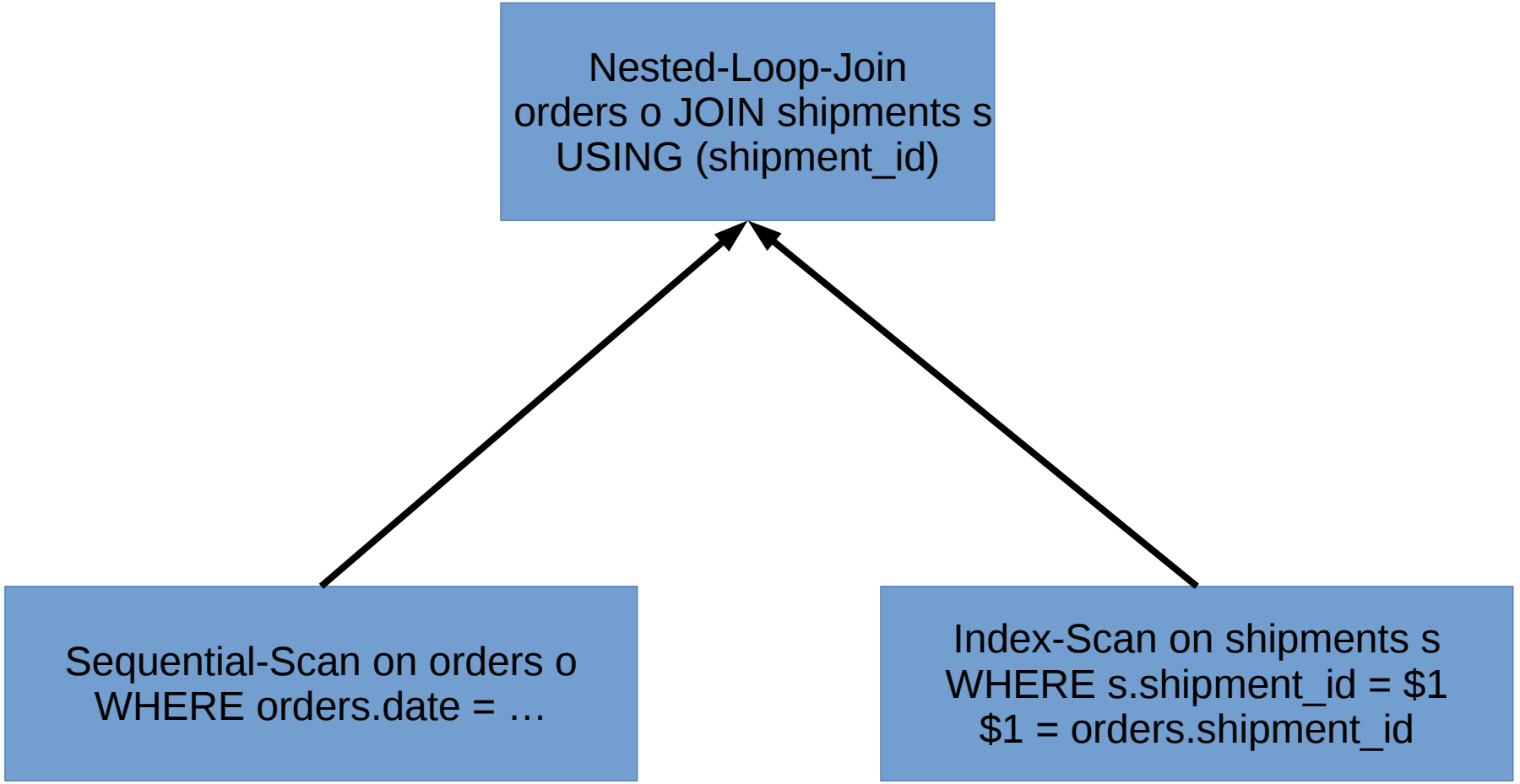
Faster Execution:
JIT Compilation

```
Sort  (cost=4313533.34..4313533.36 rows=6 width=68)
  Sort Key: l_returnflag, l_linestatus
  ->  HashAggregate  (cost=4313533.16..4313533.26 rows=6 width=68)
        Group Key: l_returnflag, l_linestatus
        Output: …, sum(l_quantity), sum(l_extendedprice), sum(…), ...
        ->  Seq Scan on lineitem  (cost=0.00..1936427.80 rows=59427634 width=36)
              Filter: (l_shipdate <= '1998-09-18 00:00:00'::timestamp without time zone)
```

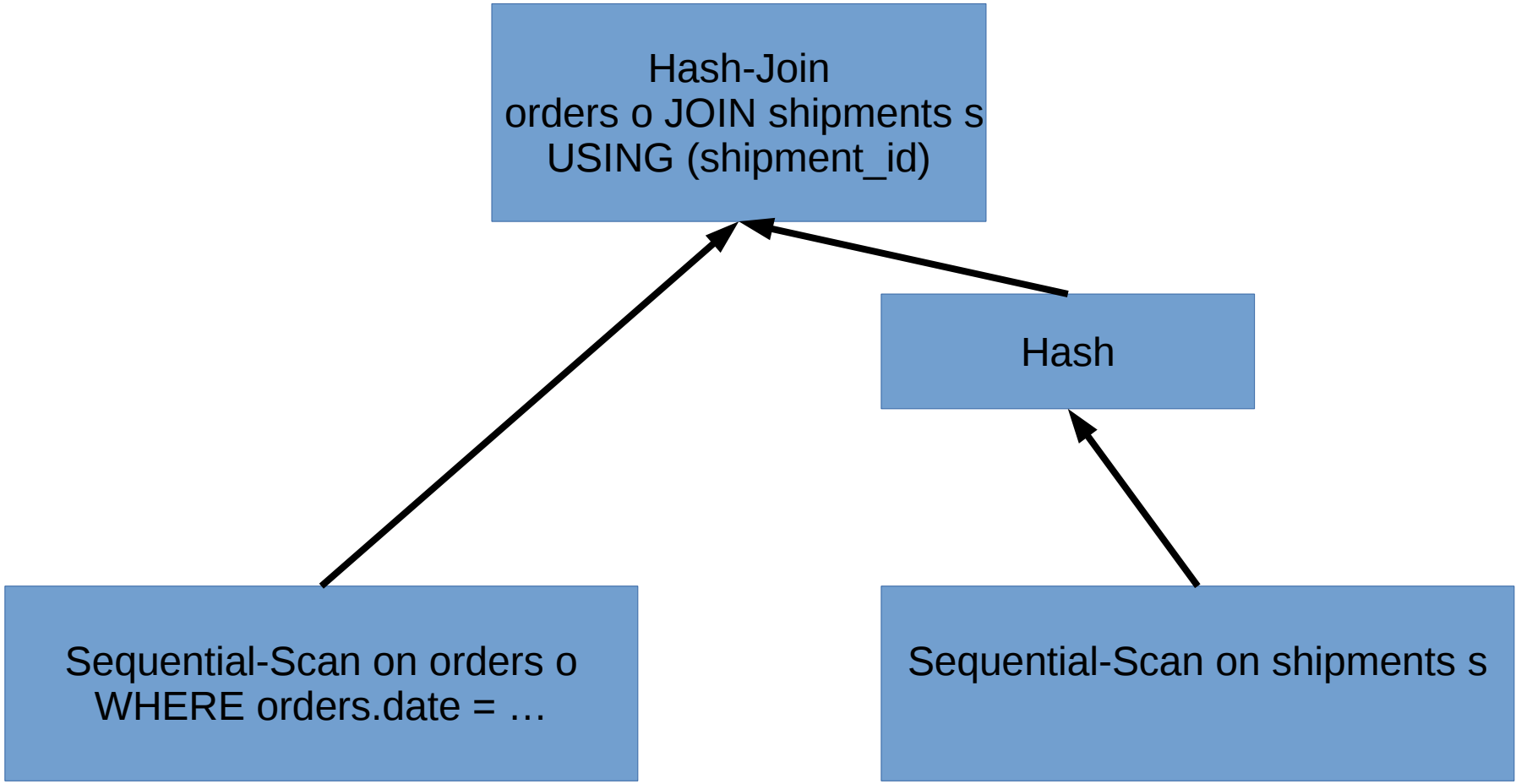
Faster Execution: JIT Compilation



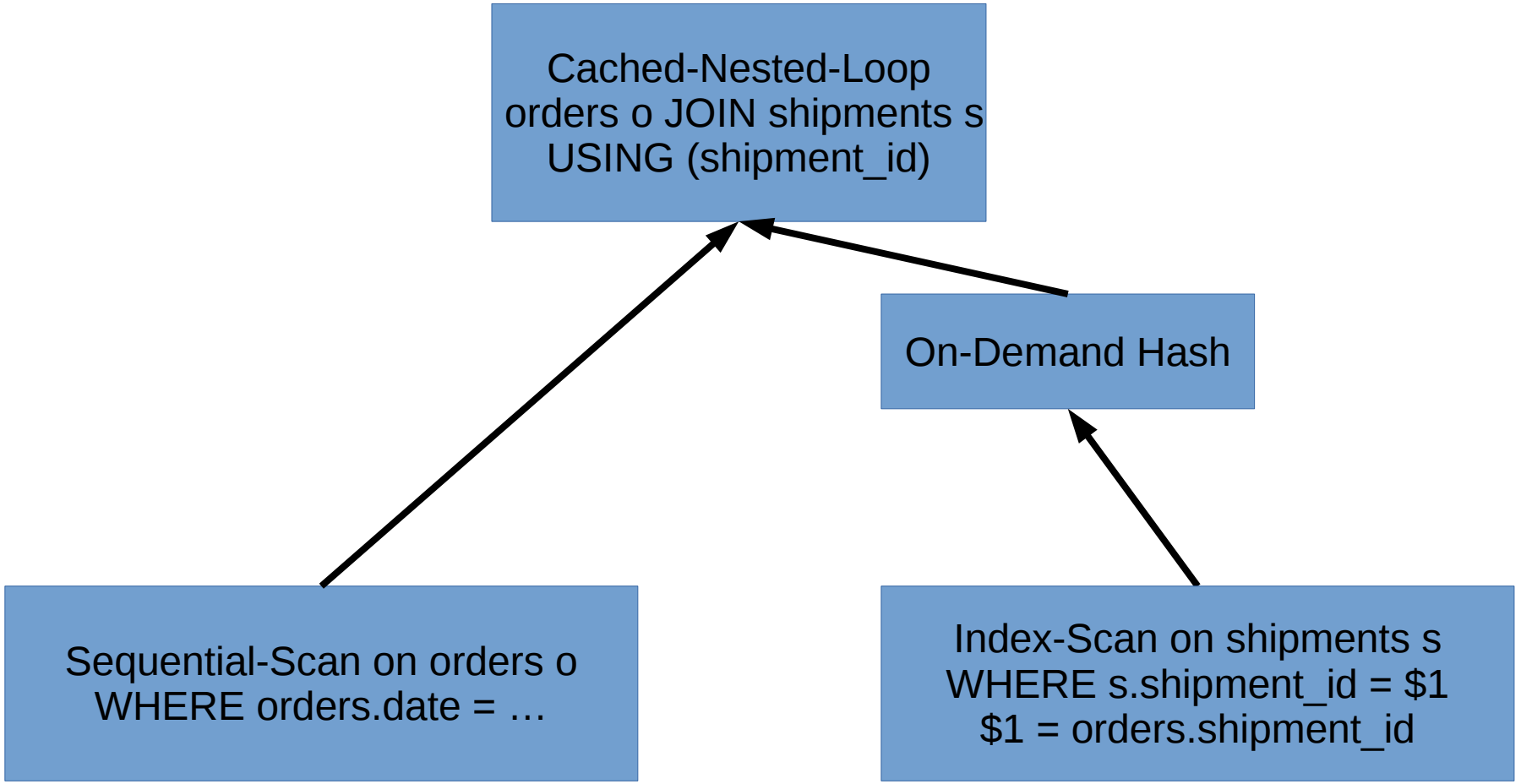
Smarter Execution: “Cached” Nested-Loop Joins



Smarter Execution: “Cached” Nested-Loop Joins



Smarter Execution: “Cached” Nested-Loop Joins



Better Planning: CTE Inlining / Barrier

```
WITH per_day AS (
    SELECT l_shipdate, count(*)
    FROM lineitem
    GROUP BY l_shipdate
)
  SELECT *
  FROM per_day
  WHERE l_shipdate = '1998-01-01'
UNION ALL
  SELECT *
  FROM per_day
  WHERE l_shipdate = '1998-01-02';
```

EDB ENTERPRISEDB

Better Planning: CTE Inlining / Barrier

- Non-Inlined: 5694 ms, fully cached
- Inlined: 5.495 ms, fully cached

Improving Postgres' Efficiency

Andres Freund
PostgreSQL Developer & Committer

Email: andres@anarazel.de
Email: andres.freund@enterprisedb.com
Twitter: @AndresFreundTec

EnterpriseDB

anarazel.de/talks/pgopen-sf-2017-09-07/efficiency.pdf

EDB ENTERPRISEDB