Unleashing PostgreSQL Performance

Andres Freund

PostgreSQL Developer & Committer

Email: andres@anarazel.de
Email: andres.freund@enterprisedb.com
Twitter: @AndresFreundTec
anarazel.de/talks/2018-06-06-pgvision-performance/performance.pdf

PostgreSQL [Adoption] Bottlenecks

```
andres@alap4:~$ psql tpch_1
psql (11beta1)
Type "help" for help.

tpch_1[10841][1]=# quit
andres@alap4:~$
```

Auto-Failover in PostgreSQL
-
How?

Bloat: zheap

Parallelism

Partitioning

EDB POSTGRES

Efficiency

Analytics / Transactional Workloads

New Hash-Table & Expression Evaluation Engine Rewrite

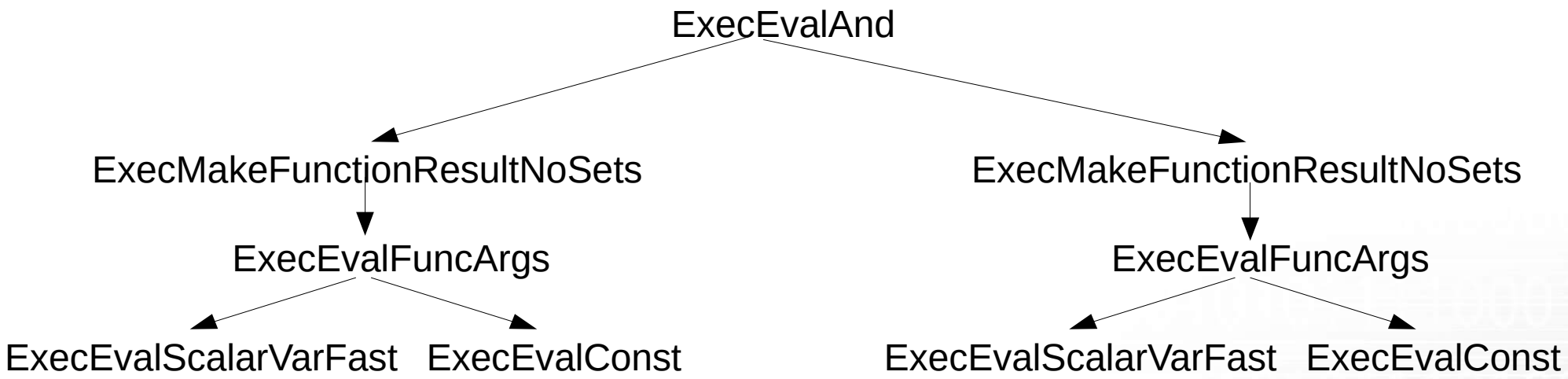
```
SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) AS sum_qty,
    sum(l_extendedprice) AS sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) AS sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) AS sum_charge,
    avg(l_quantity) AS avg_qty,
    avg(l_extendedprice) 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;
```

New Hash-Table & Expression Evaluation Engine Rewrite

```
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)
```

< v10 Expression Evaluation Engine

WHERE a.col < 10 AND a.another = 3

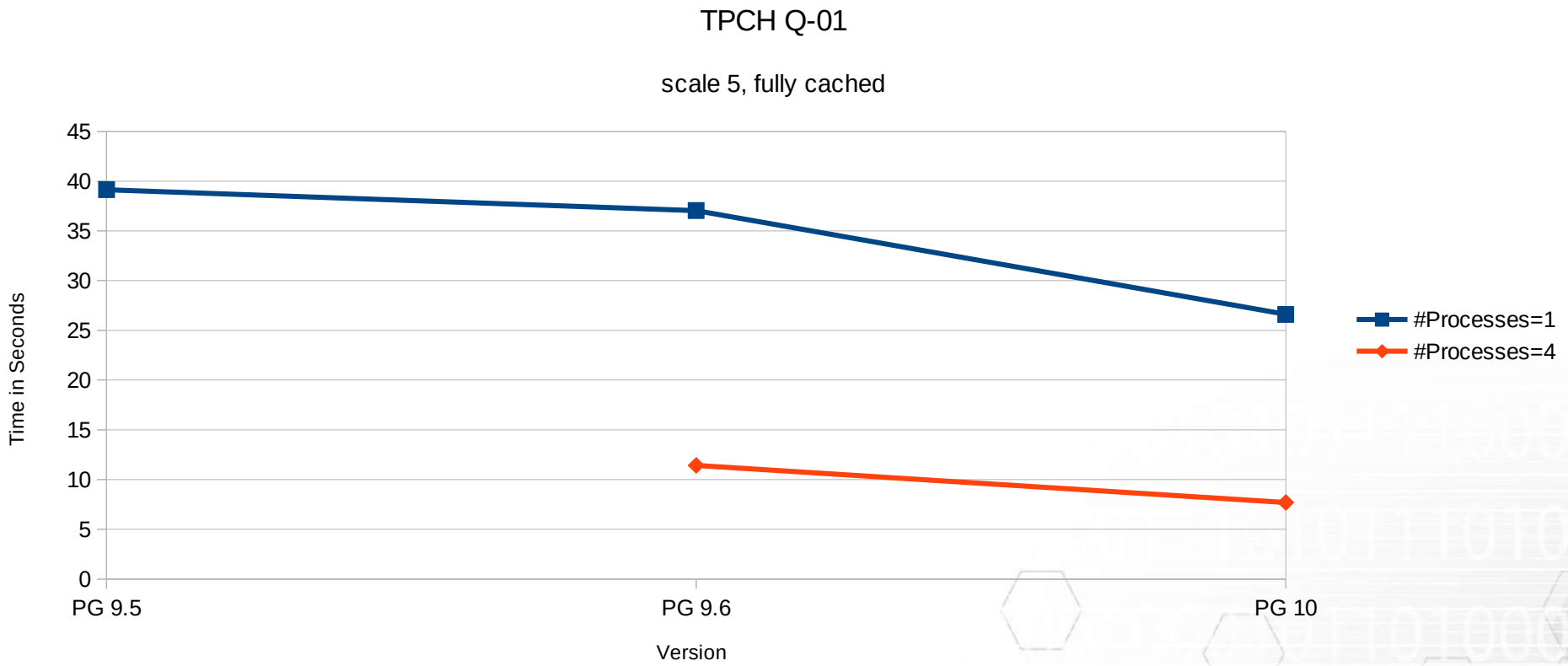


EDB POSTGRES

v10+ Expression Evaluation Engine

- WHERE a.col < 10 AND a.another = 3
  - EEOP_SCAN_FETCHSOME (deform necessary cols)
  - EEOP_SCAN_VAR (a.col)
  - EEOP_CONST (10)
  - EEOP_FUNCEXPR_STRICT (int4lt)
  - EEOP_BOOL_AND_STEP_FIRST
  - EEOP_SCAN_VAR (a.another)
  - EEOP_CONST (3)
  - EEOP_FUNCEXPR_STRICT (int4eq)
  - EEOP_BOOL_AND_STEP_LAST (AND)
- direct threaded
- lots of indirect jumps

v10+ Expression Evaluation Engine



Just-in-Time Compilation

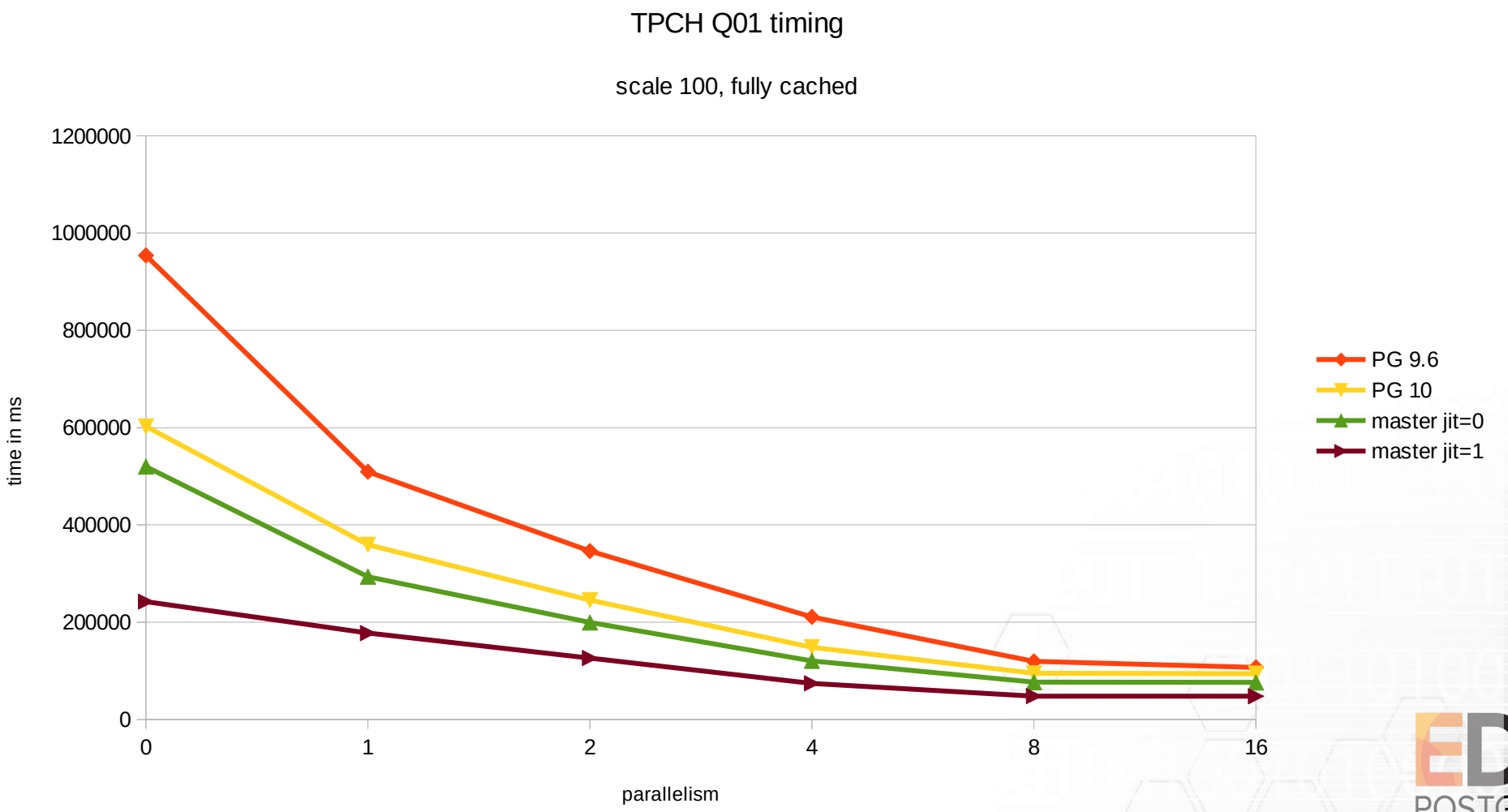
- Interpreted execution → compiled execution
  - removes interpretation overhead
- Make variables constant
  - removes branches, indirect jumps, reduces size
- Inline called functions
- PG 11: Optionally uses LLVM

Just-in-Time Compilation in v11

```
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 POSTGRES

Just-in-Time Compilation in v11



EDB POSTGRES

JIT - Good Cases / Bad Cases

- OLTP → bad, short query → bad
- OLAP → good, long query → good
- IO bound → not necessarily good
- CPU bound → likely good
- lots of aggregates → good
- wide tables (i.e. lots of columns) → good

JIT – Planning

- Naive!
- Disabled / Enabled: jit = on / off (currently enabled by default)
- `SELECT pg_jit_available();`
- Perform JIT if `query_cost > jit_above_cost`
  - `default: 100000`
- Optimize if `query_cost > jit_optimize_above_cost`
  - `default: 500000`
- Inline if `query_cost > jit_inline_above_cost`
  - `default: 500000`
- `-1` disables corresponding feature
- Whole query decision

JIT – Planning

```
Sort  (cost=4311583.16..4311583.18 rows=6 width=68) (actual time=[]18270.844 rows=4 loops=1)
  Sort Key: l_returnflag, l_linestatus
  Sort Method: quicksort  Memory: 25kB
  ->  HashAggregate  (...)
        Group Key: l_returnflag, l_linestatus
        ->  Seq Scan on lineitem  (...)
              Filter: (l_shipdate <= '1998-09-18 00:00:00'::timestamp without time zone)
              Rows Removed by Filter: 571965
Planning Time: 1.956 ms
JIT:
  Functions: 9
  Generation Time: 4.087 ms
  Inlining: true
  Inlining Time: 62.990 ms
  Optimization: true
  Optimization Time: 121.307 ms
  Emission Time: 102.416 ms
Execution Time: 18291.397 ms
```

EDB POSTGRES

JIT – how to test

- Debian:
  - apt.postgresql.org has v11 beta 1 packages
  - https://wiki.postgresql.org/wiki/Apt/FAQ#I_want_to_try_the_beta_version_of_the_next_PostgreSQL_release
  - Install postgresql-11
- RHEL:
  - yum.postgresql.org has v11 beta 1 packages
  - https://yum.postgresql.org/repopackages.php#pg11
  - install postgresql11-server postgresql11-llvmjit
    - depends on EPEL

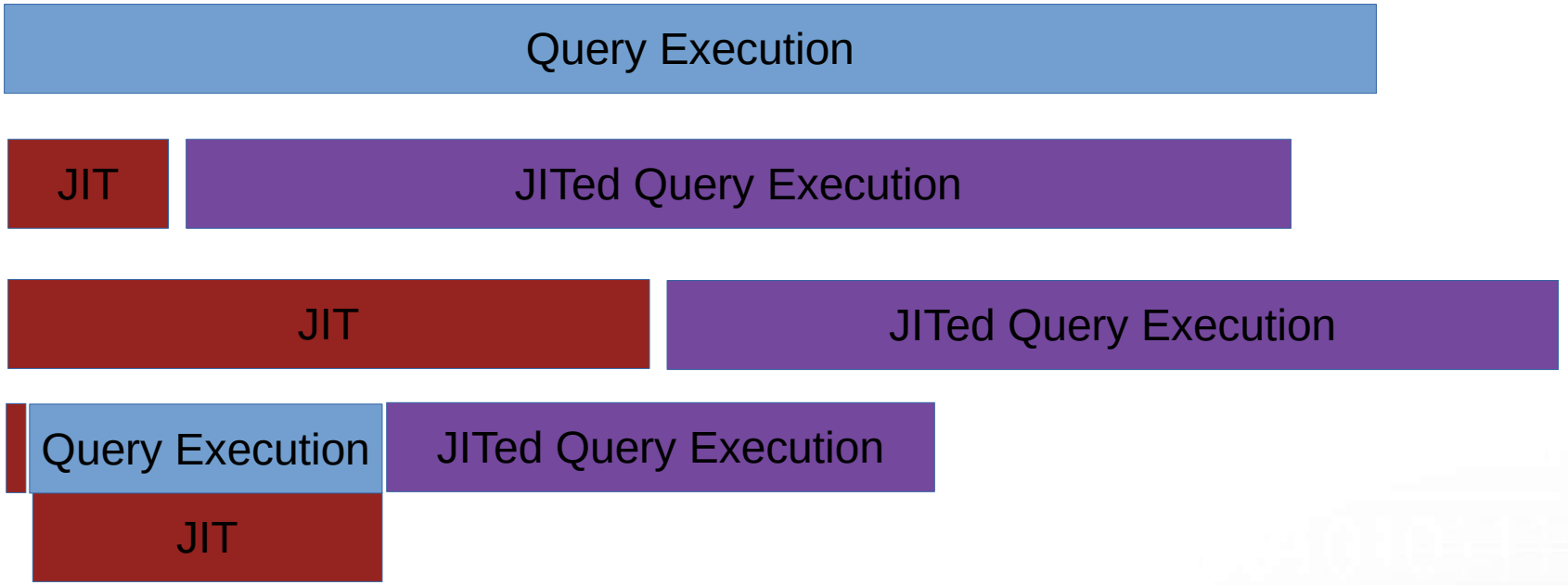
JIT Improvements: Code Generation

- Expressions refer to per-query allocated memory
  - generated code references memory locations
  - **lots** of superflous memory reads/writes for arguments, optimizer can't eliminate in most cases
    - massively reduces benefits of inlining
  - optimizer can't optimize away memory lots of memory references
  - FIX: separate permanent and per eval memory
- Expression step results refer to persistent memory
  - move to temporary memory
- Function Call Interface references persistent memory
  - FIX: pass FunctionCallInfoData and FmgrInfo separately to functions
    - remove FunctionCallInfoData->flinfo
    - move context, resultinfo, fncollation to FmgrInfo
    - move isnull field to separate argument? Return struct?
  - Non JITed expression evaluation will benefit too

JIT Improvements: Caching

- Optimizer overhead significant
  - TPCH Q01: unopt, noinline: time to optimize: 0.002s, emit: 0.036s
  - TPCH Q01: time to inline: 0.080s, optimize: 0.163s, emit 0.082s
- references to memory locations prevent caching (i.e. expression codegen has to be optimized first)
- Introduce per-backend LRU cache of functions keyed by hash of emitted LRU (plus comparator)
  - What to use as cache key?
    - IR? - requires generating it
    - Expression Trees?
    - Prepared Statement?
- Shared / Non-Shared / Persistent?
- relatively easy task, once pointers removed

JIT Improvements: Incremental JITing



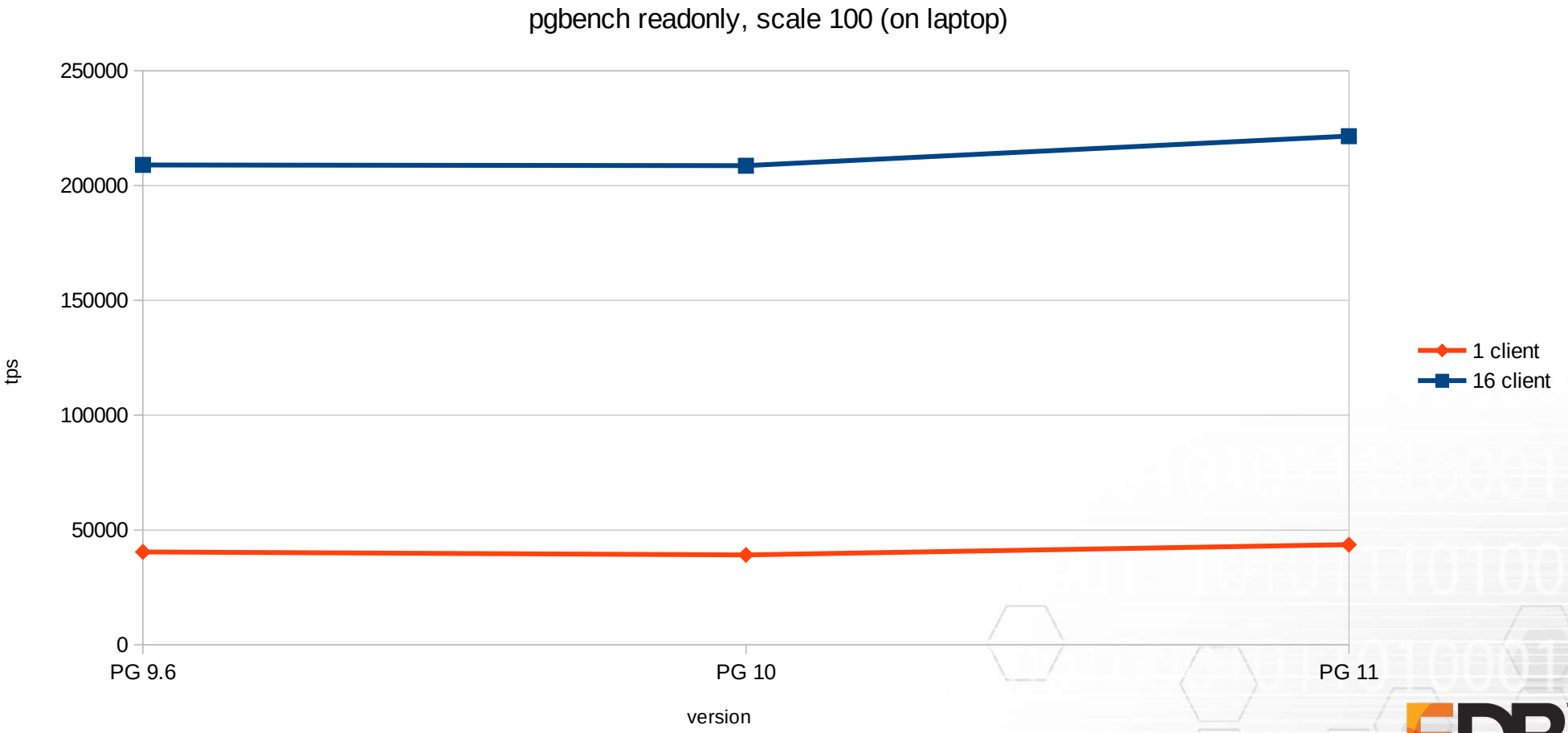
JIT Improvements: Planning

- Whole Query decision too coarse
  - use estimates about total number of each function evaluation?
- JIT more aggressively when using prepared statements?
  - after repeated executions?

Future things to JIT

- Executor control flow
  - hard, but lots of other benefits (asynchronous execution, non-JITed will be faster, less memory)
- COPY parsing, input / output function invocation
  - easy – medium
- Aggregate & Hashjoin hash computation
  - easy
- in-memory tuplesort
  - including tuple deforming (from MinimalTuple)
  - easy

OLTP Workloads



OLTP Workloads: Bottlenecks in v11

- 97.68% PostgresMain
  - 32.09% exec_execute_message (inlined)
    - 95.13% PortalRun
      - 92.08% PortalRunSelect
        - 98.38% standard_ExecutorRun
          + 98.23% ExecutePlan (inlined)
          + 0.86% printtup_startup

  - 31.90% exec_bind_message (inlined)
    - 53.38% PortalStart
      - 86.96% standard_ExecutorStart
        + 96.43% InitPlan (inlined)
        + 3.29% CreateExecutorState
    + 10.56% GetCachedPlan
    + 8.14% start_xact_command (inlined)

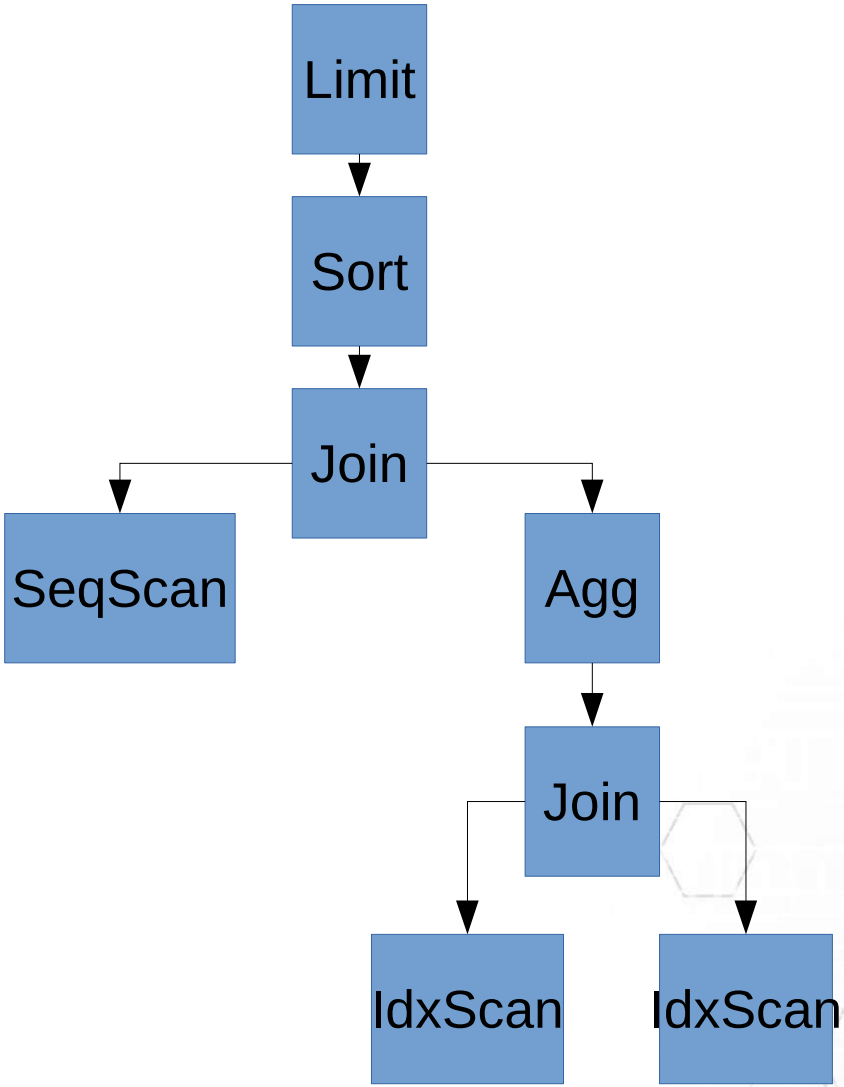
  + 13.06% finish_xact_command
  + 10.31% ReadCommand (inlined)

EDB POSTGRES

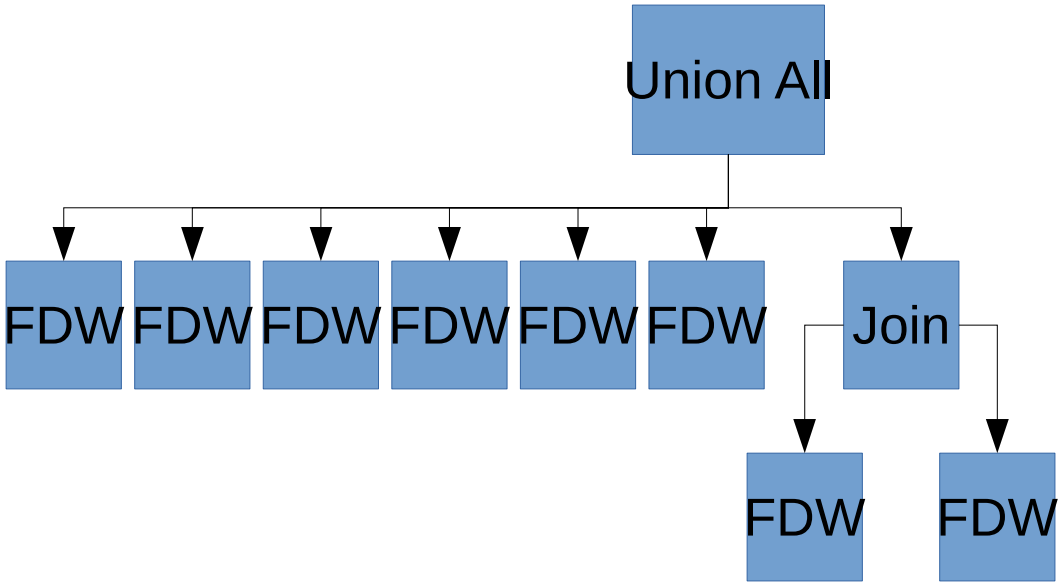
OLTP Workloads: Bottlenecks in v11

- Too much work done every query execution
  - move to planner, helping prepared statements
  - reduce duplicated work
- Execution uses too much memory / too many small allocations
  - rework executor data structures
  - batch all allocations into one single memory allocation

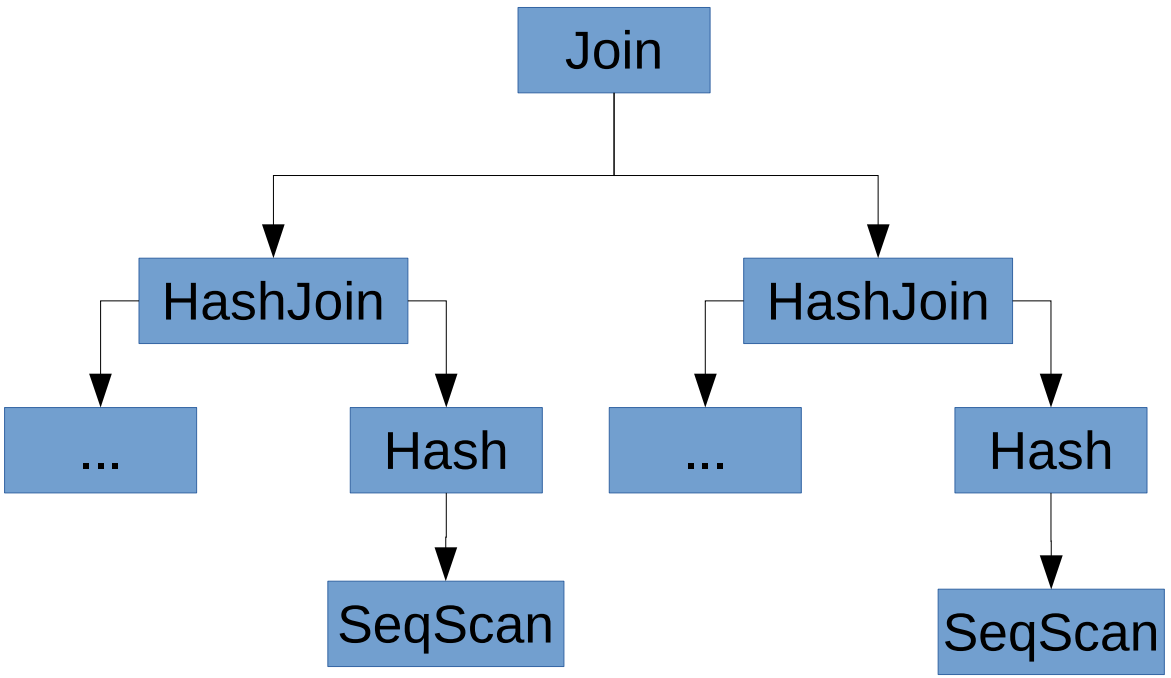
Current Executor



Async Execution



Async IO



JIT

```
static pg_attribute_always_inline
TupleTableSlot *
ExecHashJoinImpl(PlanState *pstate, bool
parallel)
{
...
      for (;;)
      {
          /* lotsa code */
...

                    slot =
ExecProcNode(outerNode);
...
```

Linearized Programs

```
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;
```

0: init_sort
1: seqscan_first
2: seqscan [j empty 5] > s0
3: qual [j fail 2] < scan s0
4: hashagg_tuple [j 2] < s0
5: drain_hashagg [j empty 7] > s1
6: sort_tuple [j 5] < s1
7: sort
8: drain_sort [j empty 10] > s2
9: return < s2 [next 8]
10: done

EDB ENTERPRISEDB