Performance improvements in PostgreSQL 9.5 and 9.6

5432meet.us 2016, June 29, Milan

Tomas Vondra
tomas.vondra@2ndquadrant.com

2ndQuadrant
Professional PostgreSQL

http://www.slideshare.net/fuzzycz/postgresql-performance-improvements-in-95-and-96

2ndQuadrant Professional PostgreSQL

PostgreSQL 9.5, 9.6, ...

- many improvements
  - many of them related to performance
  - many quite large

- release notes are good overview, but ...
  - many changes not mentioned explicitly
  - often difficult to get an idea of the impact

- many talks about new features in general
  - this talk is about changes affecting performance

What we'll look at?

- PostgreSQL 9.5 & 9.6
- only “main” improvements
  - complete “features” (multiple commits)
  - try to showcase the impact
  - no particular order
- dozens of additional optimizations
  - see release notes for the full list

PostgreSQL 9.5

2ndQuadrant
Professional PostgreSQL

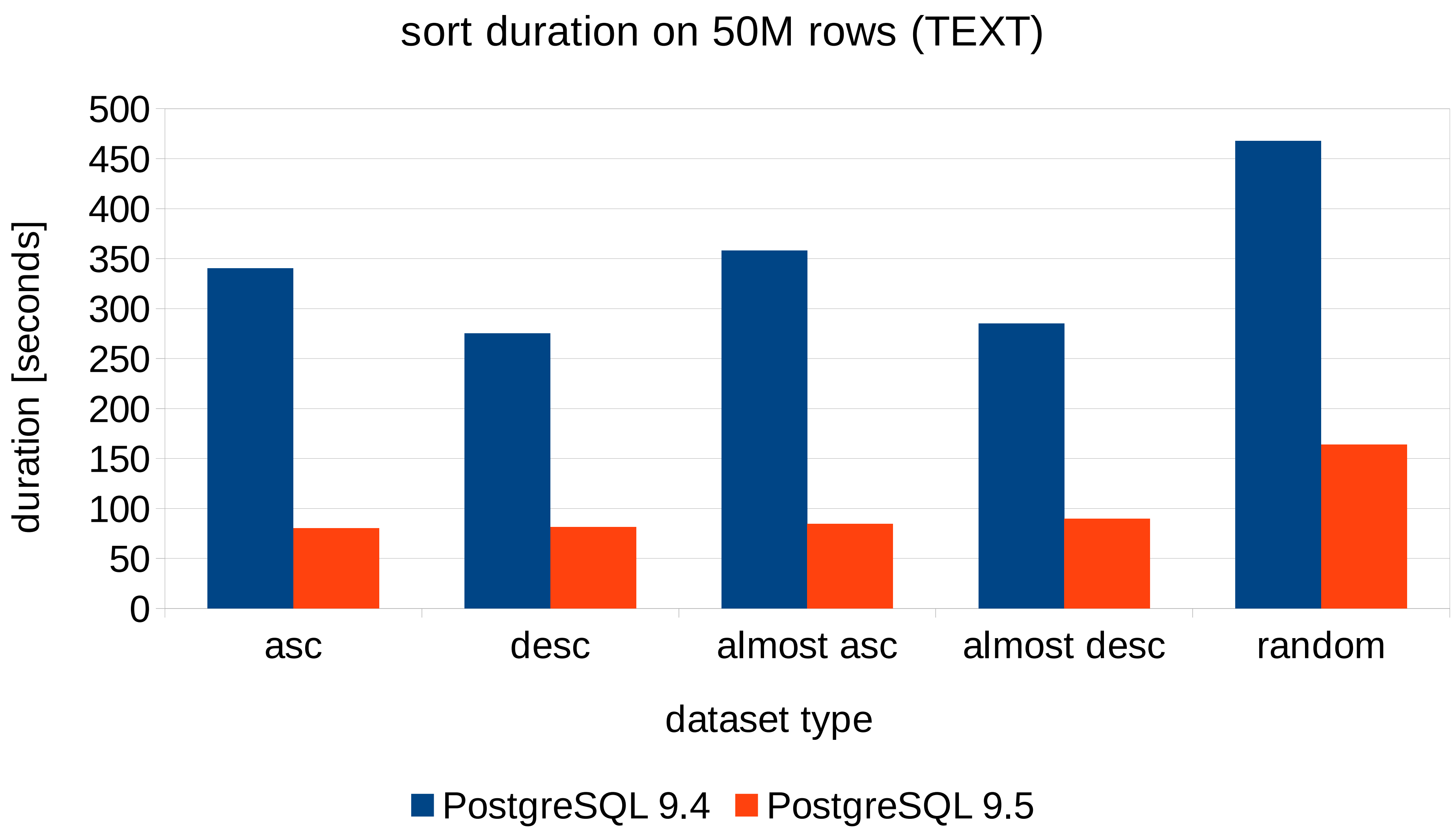
Sorting

- allow sorting by in-lined, non-SQL-callable functions
  - reduces per-call overhead
- use abbreviated keys for faster sorting (strxfrm)
  - VARCHAR, TEXT, NUMERIC
  - does not apply to CHAR values!
- places using “Sort Support” benefits from this
  - CREATE INDEX, REINDEX, CLUSTER
  - ORDER BY (when not evaluated using an index)

Sorting

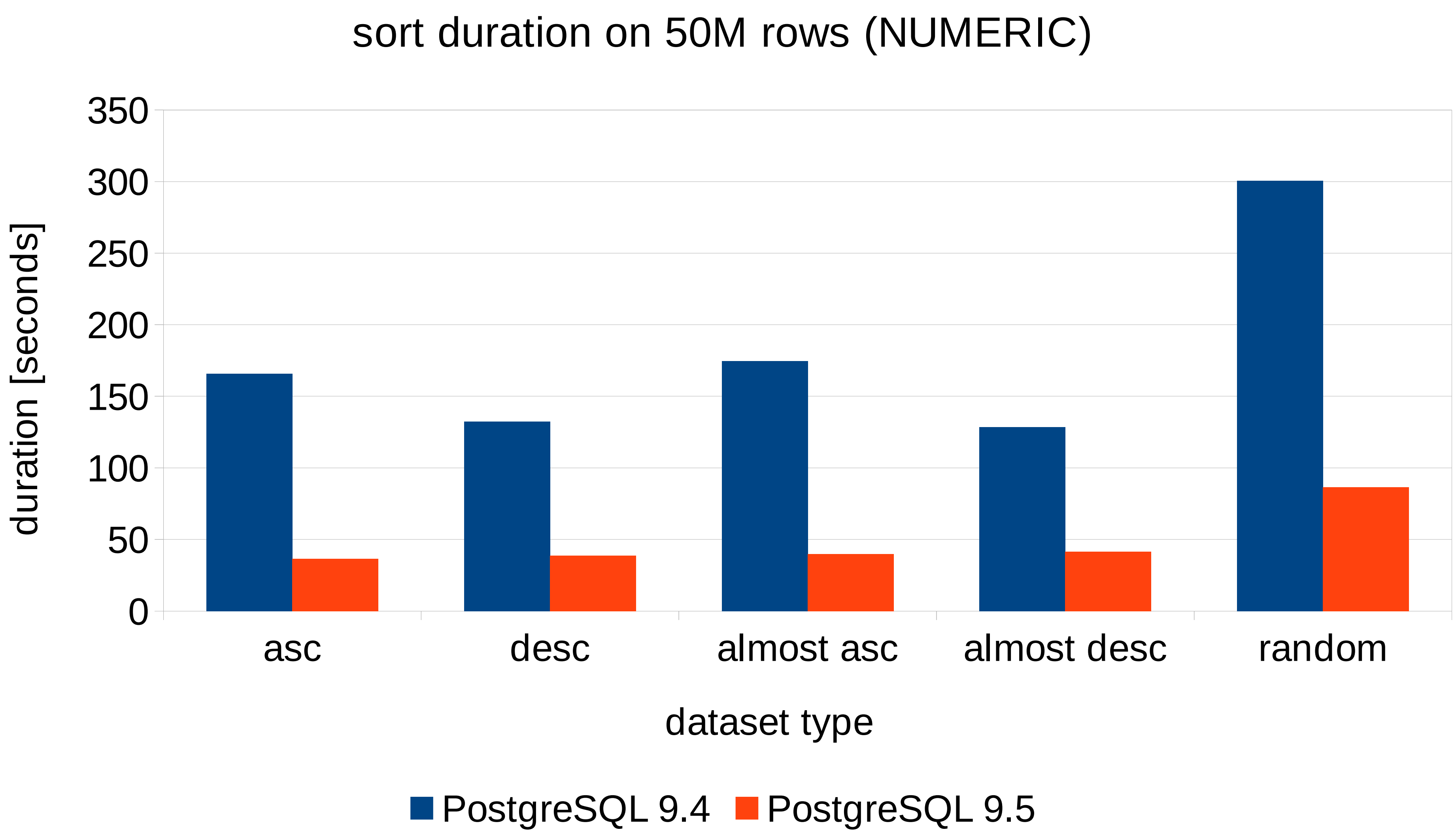
```
-- randomly sorted table
CREATE TABLE test_text_random AS
SELECT md5(i::text) AS val
  FROM generate_series(1, 50.000.000) s(i);

-- correctly sorted table
CREATE TABLE test_text_asc AS
SELECT * from test_text_random ORDER BY 1;

-- test query
SELECT COUNT(1) FROM (
   SELECT * FROM test_text_random ORDER BY 1
) foo;
```

Sorting improvements in PostgreSQL 9.5



2ndQuadrant
Professional PostgreSQL

Sorting improvements in PostgreSQL 9.5

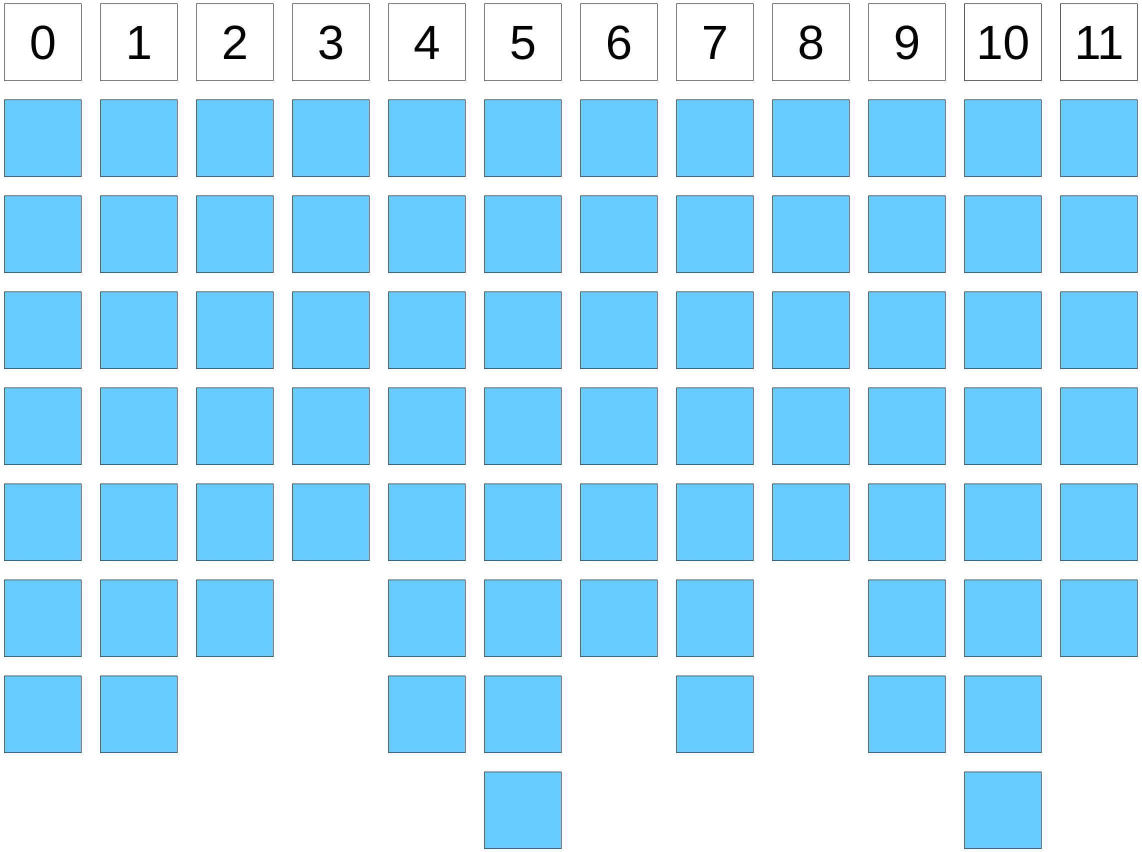


2ndQuadrant
Professional PostgreSQL

Hash Joins

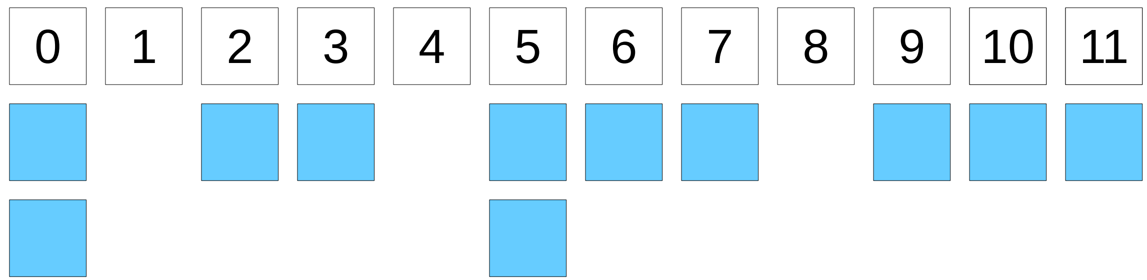
- reduce palloc overhead
  - dense packing of tuples (trivial local allocator, same life-span)
  - significant reduction of overhead (both space and time)
- reduce NTUP_PER_BUCKET to 1 (from 10)
  - goal is less that 1 tuple per bucket (on average)
  - significant speedup of lookups
- dynamically resize the hash table
  - handle under-estimates gracefully
  - otherwise easily 100s of tuples per bucket (linked list)

Hash Joins



2ndQuadrant
Professional PostgreSQL

Hash Joins

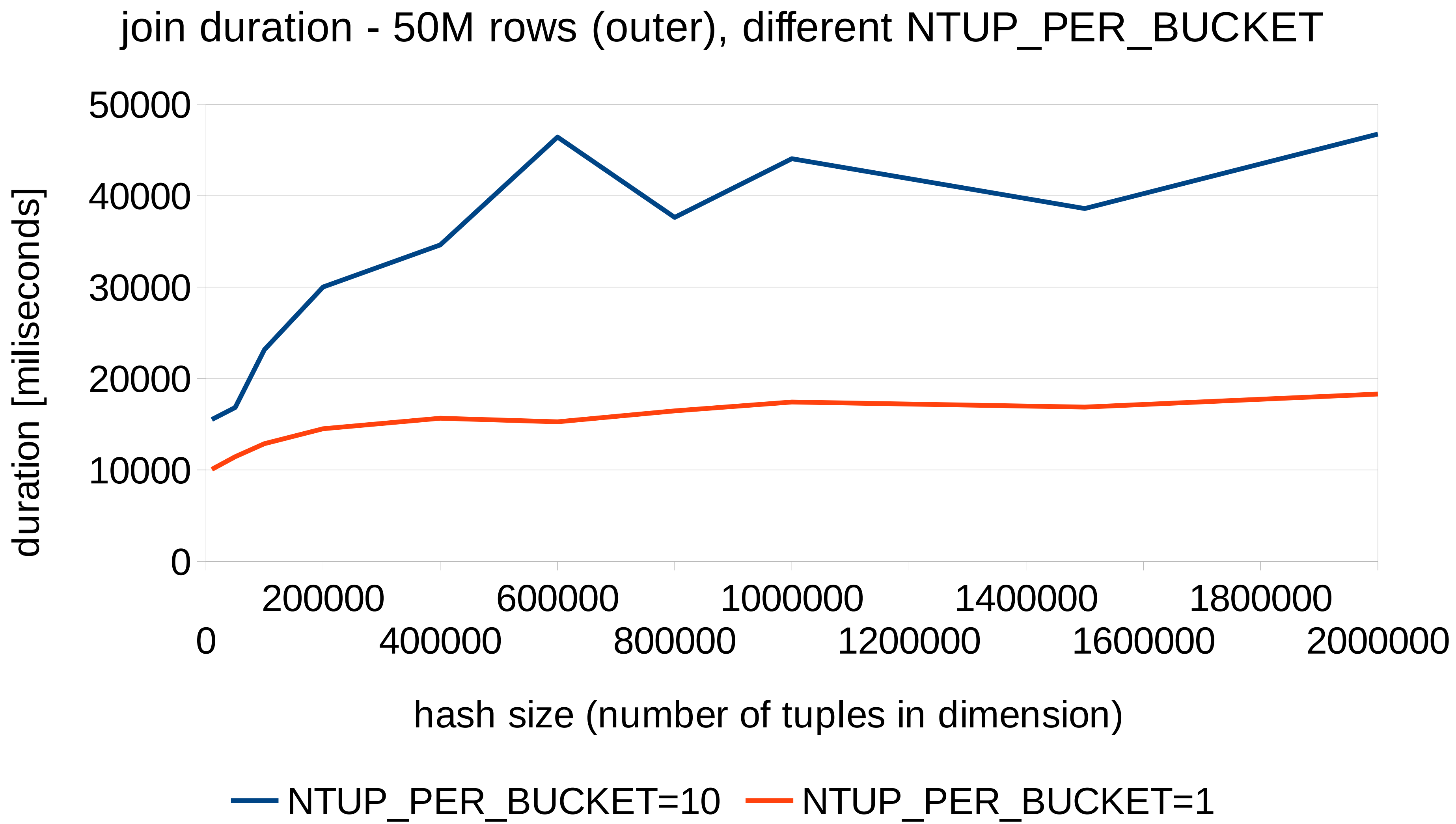


Hash Joins

```
-- dimension table (small one, will be hashed)
CREATE TABLE test_dim AS
SELECT (i-1) AS id, md5(i::text) AS val
  FROM generate_series(1, 100.000) s(i);

-- fact table (large one)
CREATE TABLE test_fact AS
SELECT mod(i, 100.000) AS dim_id, md5(i::text) AS val
  FROM generate_series(1, 50.000.000) s(i);

-- example query (join of the two tables)
SELECT count(*) FROM test_fact
                JOIN test_dim ON (dim_id = id);
```

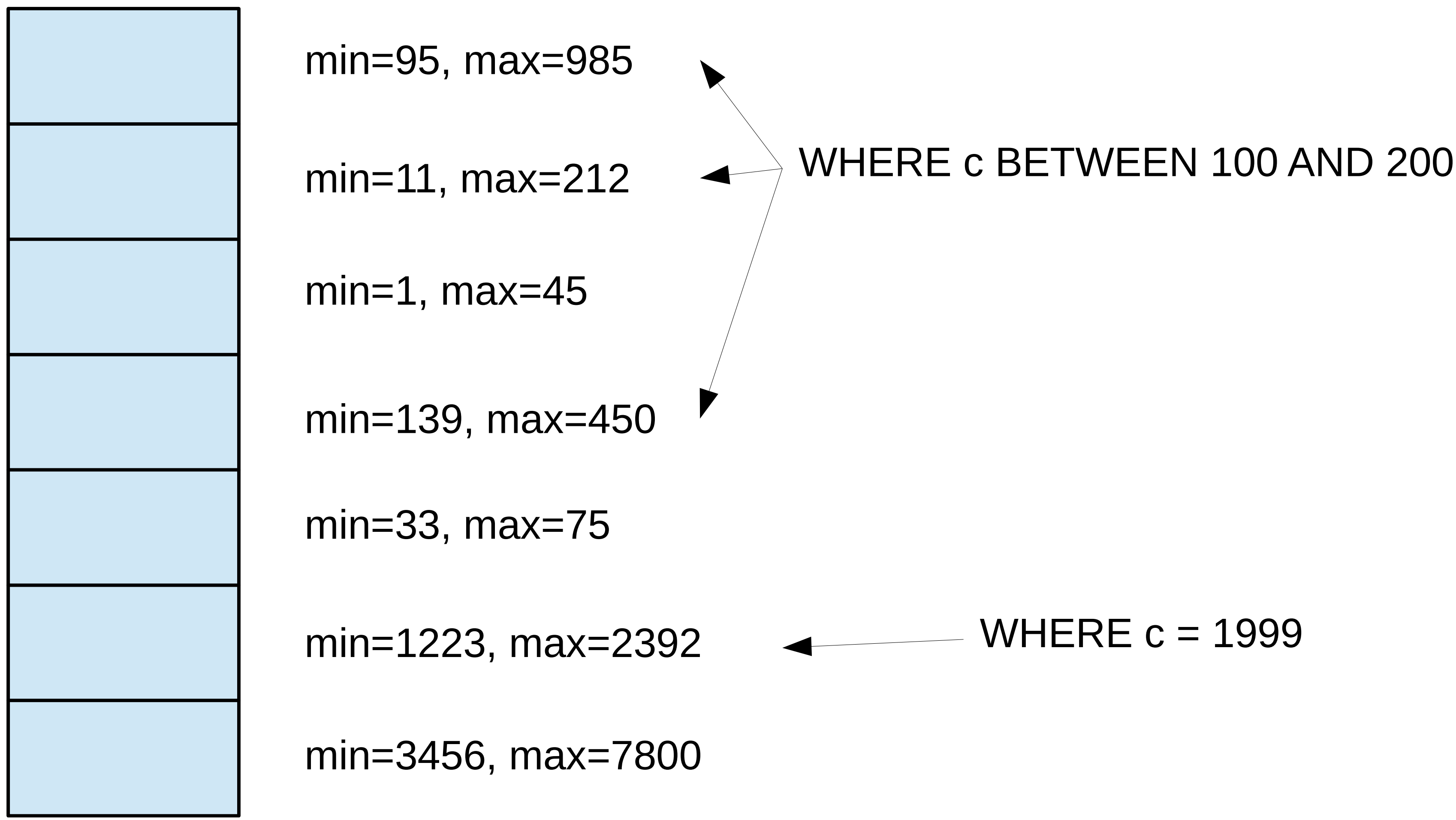
2ndQuadrant
Professional PostgreSQL

PostgreSQL 9.5 Hash Join Improvements



2ndQuadrant
Professional PostgreSQL

BRIN Indexes



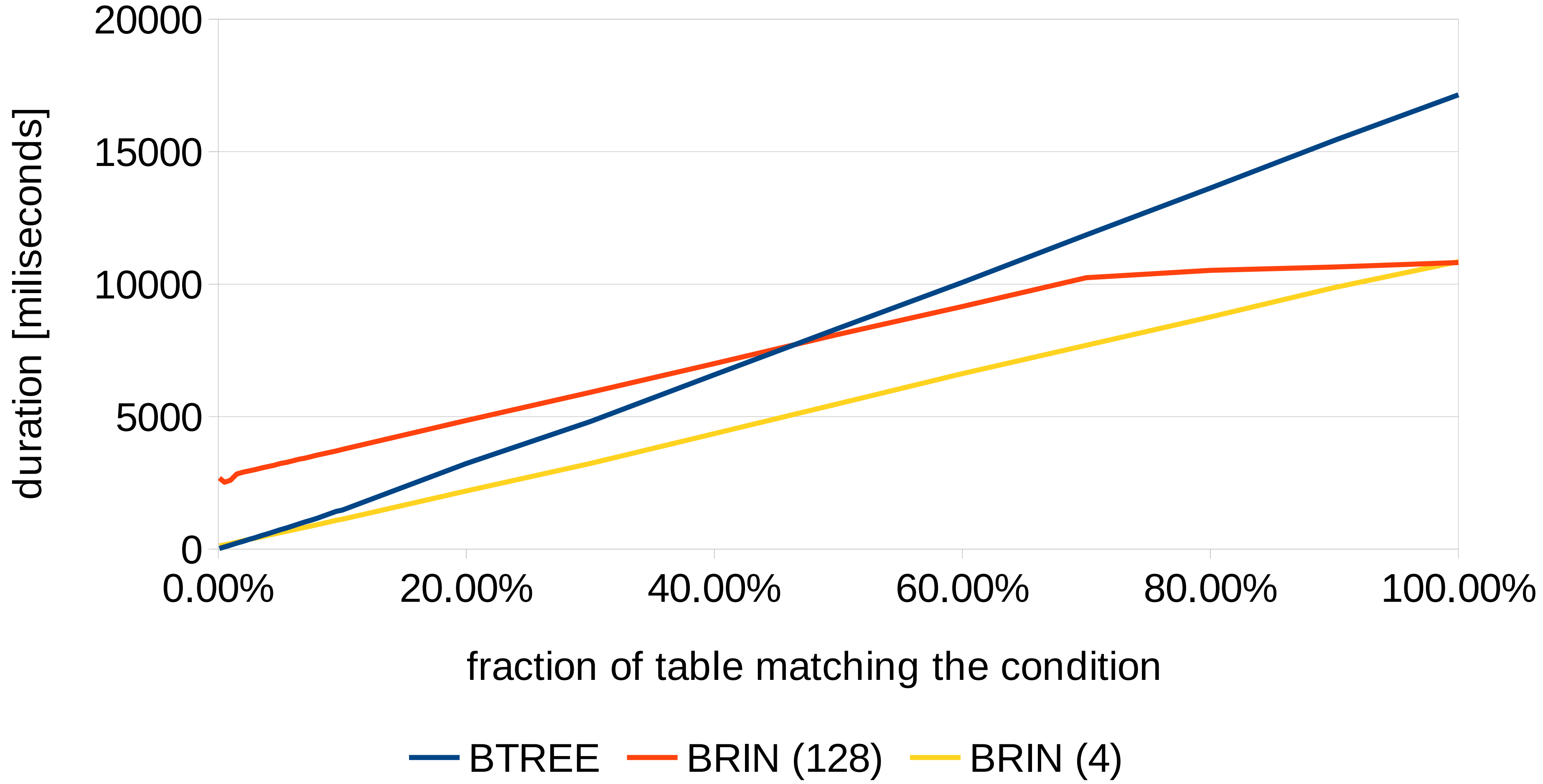
2ndQuadrant
Professional PostgreSQL

BRIN Indexes

```
-- table with 100M rows
CREATE TABLE test_bitmap AS
  SELECT mod(i, 100.000) AS val
    FROM generate_series(1, 100.000.000) s(i);
CREATE INDEX test_btree_idx ON test_bitmap(val);
CREATE INDEX test_brin_idx ON test_bitmap USING brin(val);

-- benchmark (enforce bitmap index scan)
SET enable_seqscan = off;
SET enable_indexscan = off;

SELECT COUNT(*) FROM test_bitmap WHERE val <= $1;
```

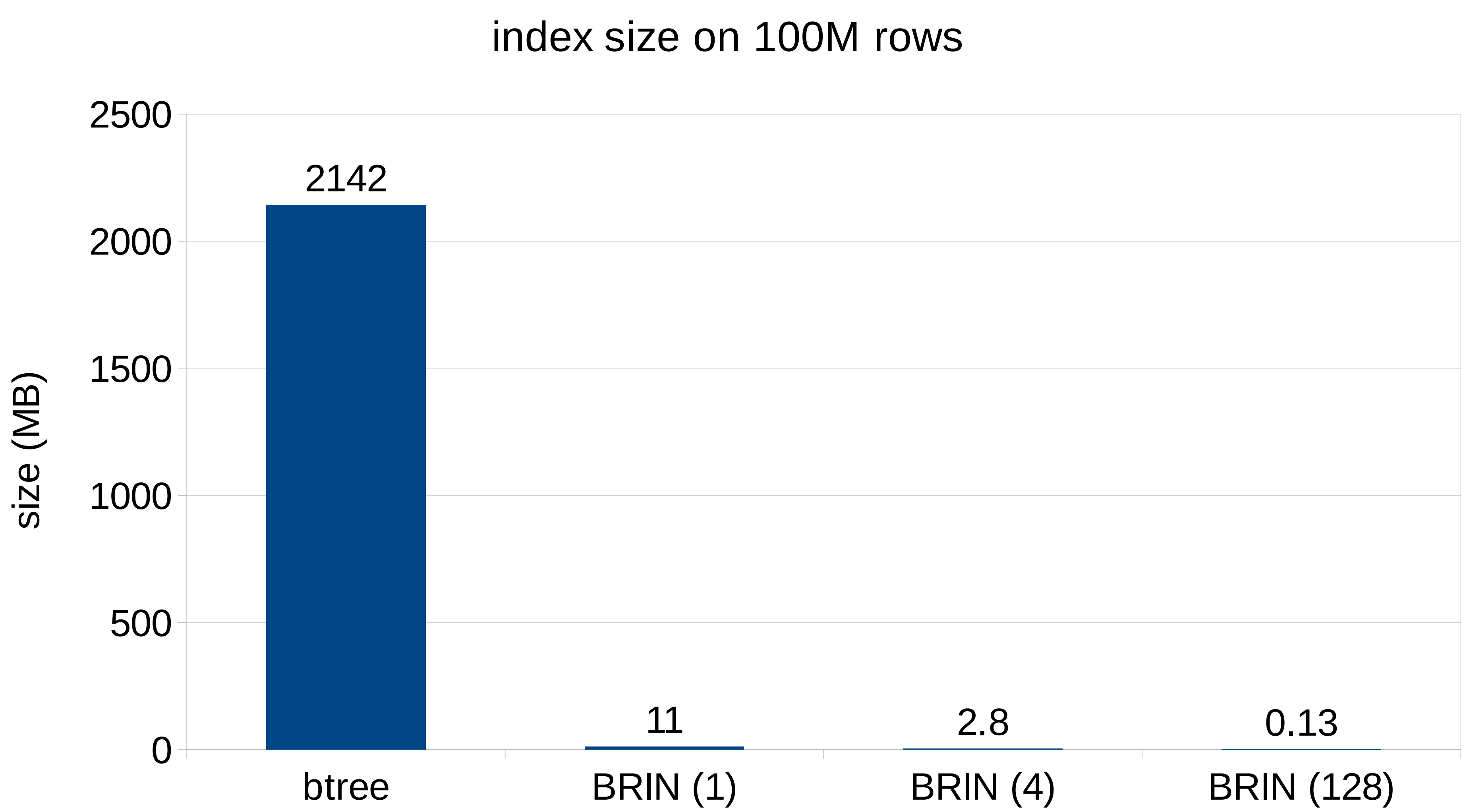
BRIN vs. BTREE

Bitmap Index Scan on 100M rows (sorted)



2ndQuadrant
Professional PostgreSQL

BRIN vs. BTREE



2ndQuadrant
Professional PostgreSQL

Other Index Improvements

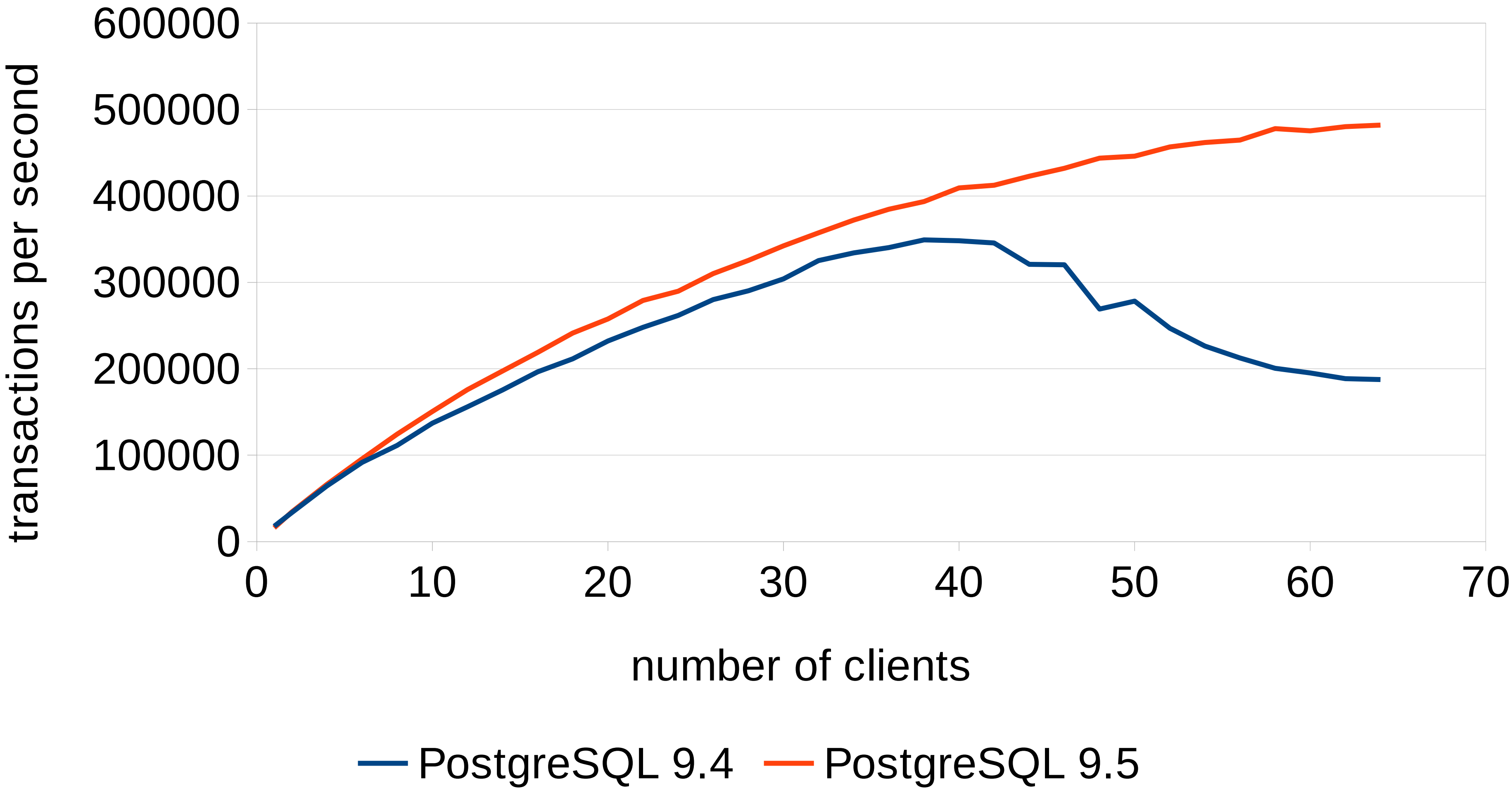
- CREATE INDEX
  - avoid copying index tuples when building an index (palloc overhead)
- Index-only scans with GiST
  - support to range type, inet GiST opclass and btree_gist
- Bitmap Index Scan
  - in some cases up to 50% spent in `tbm_add_tuples`
  - cache the last accessed page in `tbm_add_tuples`

Other Improvements

- locking and shared_buffers scalability
  - reduce overhead, make it more concurrent
  - large (multi-socket) systems
  - reduce lock strength for some DDL commands
- CRC optimizations (--data-checksums)
  - use SSE when available, various optimizations
  - significantly improved throughput (GB/s)
- planner optimizations
  - make the planning / execution smarter
- PL/pgSQL improvements

read-only scalability improvements in 9.5

pgbench -S -M prepared -j $N -c $N



2ndQuadrant
Professional PostgreSQL

PostgreSQL 9.6

Parallel Query

- until now, each query limited to 1 core
- 9.6 parallelizes some operations
  - sequential scan, aggregation, joins (NL + hash)
  - limited to read-only queries
  - setup overhead, efficient on large tables
- in the future
  - utility commands (CREATE INDEX, VACUUM, …)
  - additional operations (Sort, …)
  - improving supported ones (sharing hashtable in hashjoins)

2ndQuadrant
Professional PostgreSQL

Parallel Query

```
-- table with 1 billion rows (~80GB on disk)
CREATE TABLE f AS
       SELECT MOD(i,100000) AS id, MD5(i::text) AS h, random() AS amount
         FROM generate_series(1,1000000000) s(i);

EXPLAIN SELECT SUM(amount) FROM f JOIN d USING (id);

                              QUERY PLAN
----------------------------------------------------------------------
 Aggregate  (cost=35598980.00..35598980.01 rows=1 width=8)
   ->  Hash Join  (cost=3185.00..33098980.00 rows=1000000000 width=8)
         Hash Cond: (f.id = d.id)
         ->  Seq Scan on f  (cost=0.00..19345795.00 rows=1000000000 ...)
         ->  Hash  (cost=1935.00..1935.00 rows=100000 width=4)
               ->  Seq Scan on d  (cost=0.00..1935.00 rows=100000 ...)
(6 rows)
```

2ndQuadrant
Professional PostgreSQL

Parallel Query

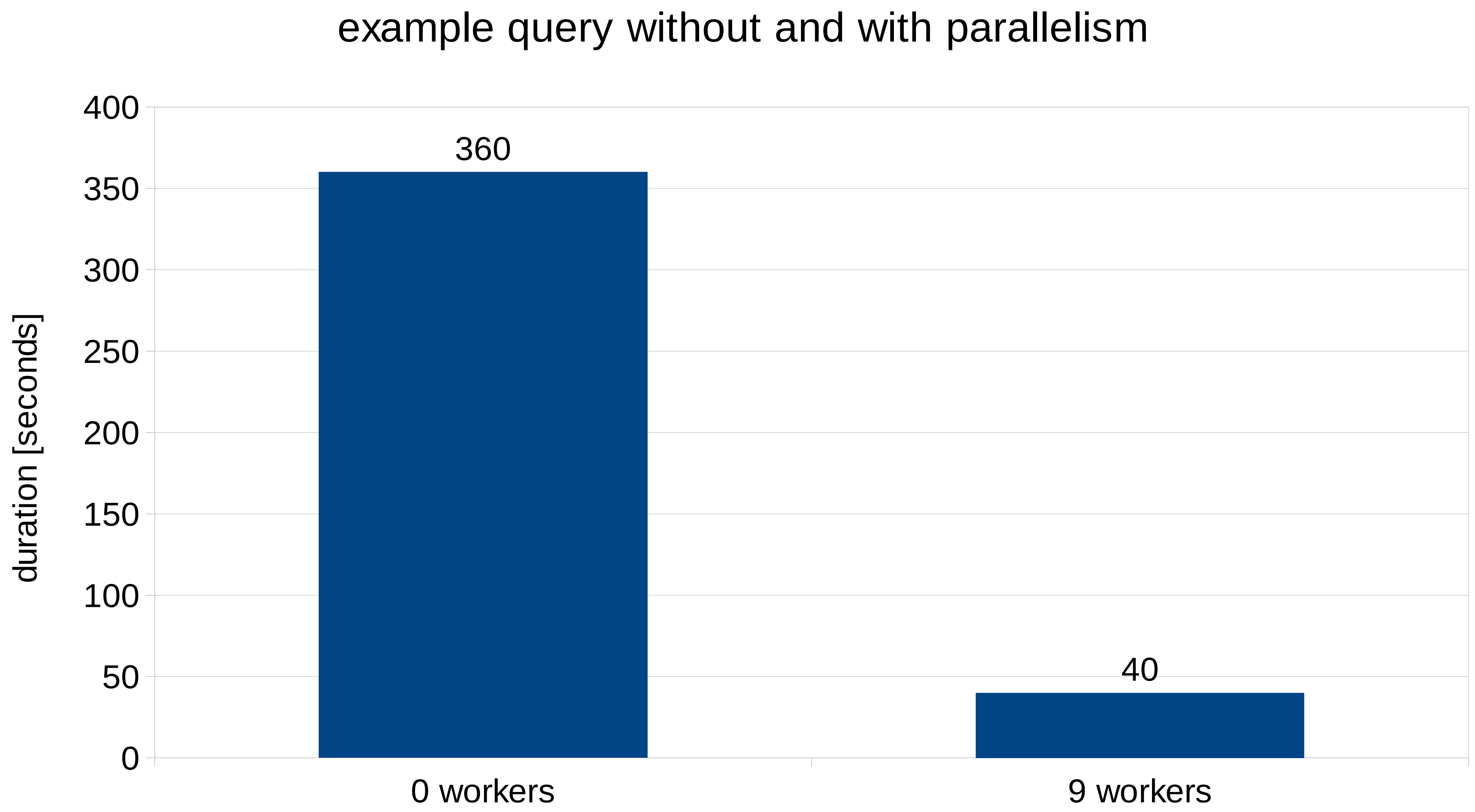
```
SET max_parallel_workers_per_gather = 32;

EXPLAIN SELECT SUM(amount) FROM f JOIN d USING (id);

                                   QUERY PLAN
---------------------------------------------------------------------------------
 Finalize Aggregate  (cost=14488869.82..14488869.83 rows=1 width=8)
   ->  Gather  (cost=14488868.89..14488869.80 rows=9 width=8)
         Workers Planned: 9
         ->  Partial Aggregate  (cost=14487868.89..14487868.90 rows=1 width=8)
               ->  Hash Join  (cost=3185.00..11987868.89 rows=1000000000 width=8)
                     Hash Cond: (f.id = d.id)
                     ->  Parallel Seq Scan on f  (cost=0.00..10456906.11 ...)
                     ->  Hash  (cost=1935.00..1935.00 rows=100000 width=4)
                           ->  Seq Scan on d  (cost=0.00..1935.00 rows=100000 ...)
(9 rows)
```

top

```
  PID   VIRT  RES   SHR S %CPU %MEM  COMMAND
19018  32.8g 441m 427m R  100  0.2  postgres: sekondquad test [local] SELECT
20134  32.8g  80m  74m R  100  0.0  postgres: bgworker: parallel worker for PID 19018
20135  32.8g  80m  74m R  100  0.0  postgres: bgworker: parallel worker for PID 19018
20136  32.8g  80m  74m R  100  0.0  postgres: bgworker: parallel worker for PID 19018
20140  32.8g  80m  74m R  100  0.0  postgres: bgworker: parallel worker for PID 19018
20141  32.8g  80m  74m R  100  0.0  postgres: bgworker: parallel worker for PID 19018
20142  32.8g  80m  74m R  100  0.0  postgres: bgworker: parallel worker for PID 19018
20137  32.8g  80m  74m R   99  0.0  postgres: bgworker: parallel worker for PID 19018
20138  32.8g  80m  74m R   99  0.0  postgres: bgworker: parallel worker for PID 19018
20139  32.8g  80m  74m R   99  0.0  postgres: bgworker: parallel worker for PID 19018
   16      0    0    0 S    0  0.0  [watchdog/2]
  281      0    0    0 S    0  0.0  [khugepaged]
....
```

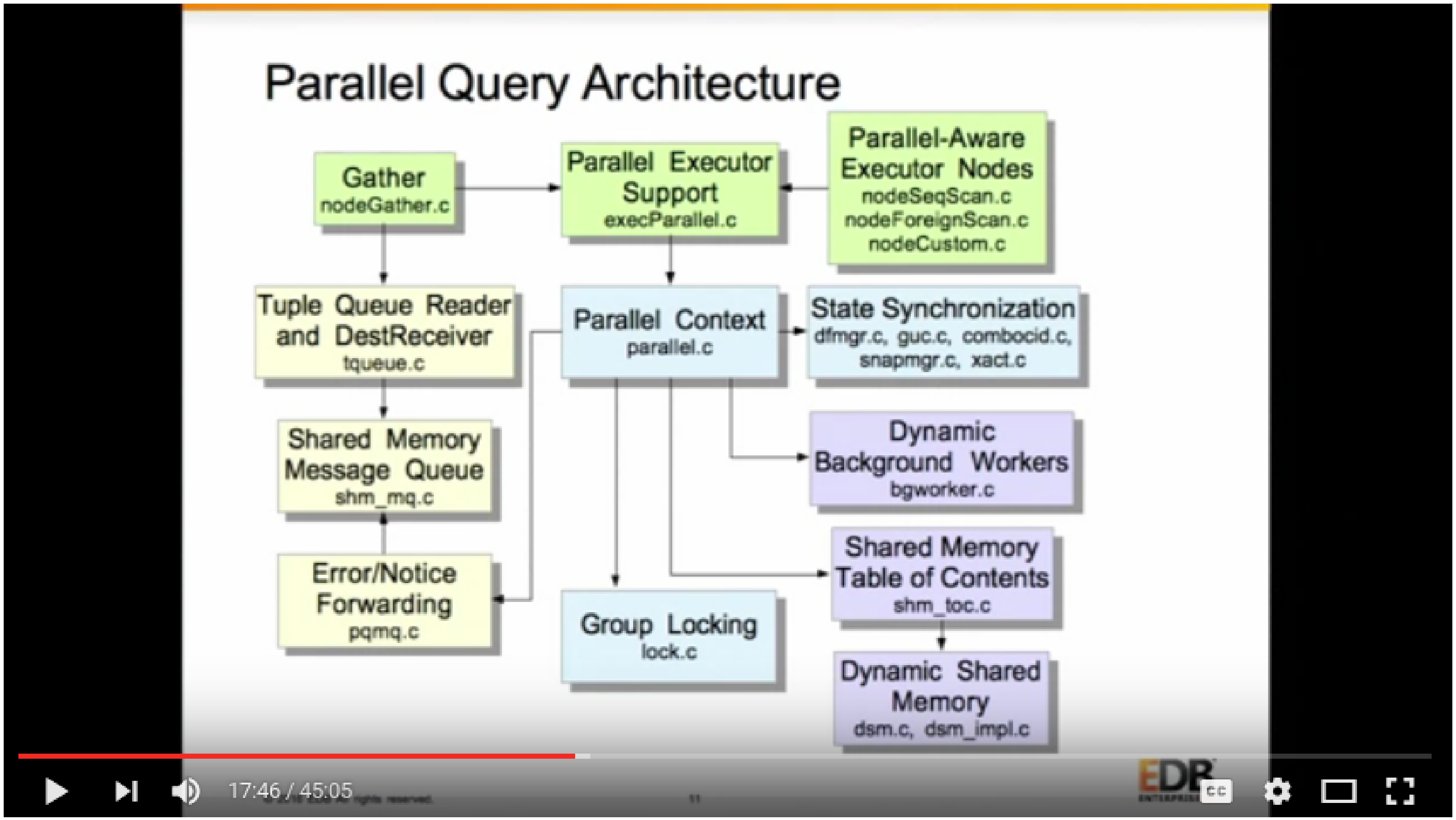
2ndQuadrant
Professional PostgreSQL

speedup with parallel query



2ndQuadrant
Professional PostgreSQL

Parallel Query Has Arrived!



https://www.youtube.com/watch?v=ysHZ1PDnH-s

2ndQuadrant
Professional PostgreSQL

Aggregate functions

- some aggregates use the same state
  - AVG, SUM, …
  - we’re keeping it separate and updating it twice
  - but only the final function is actually different
- so …

Share transition state between different aggregates when possible.

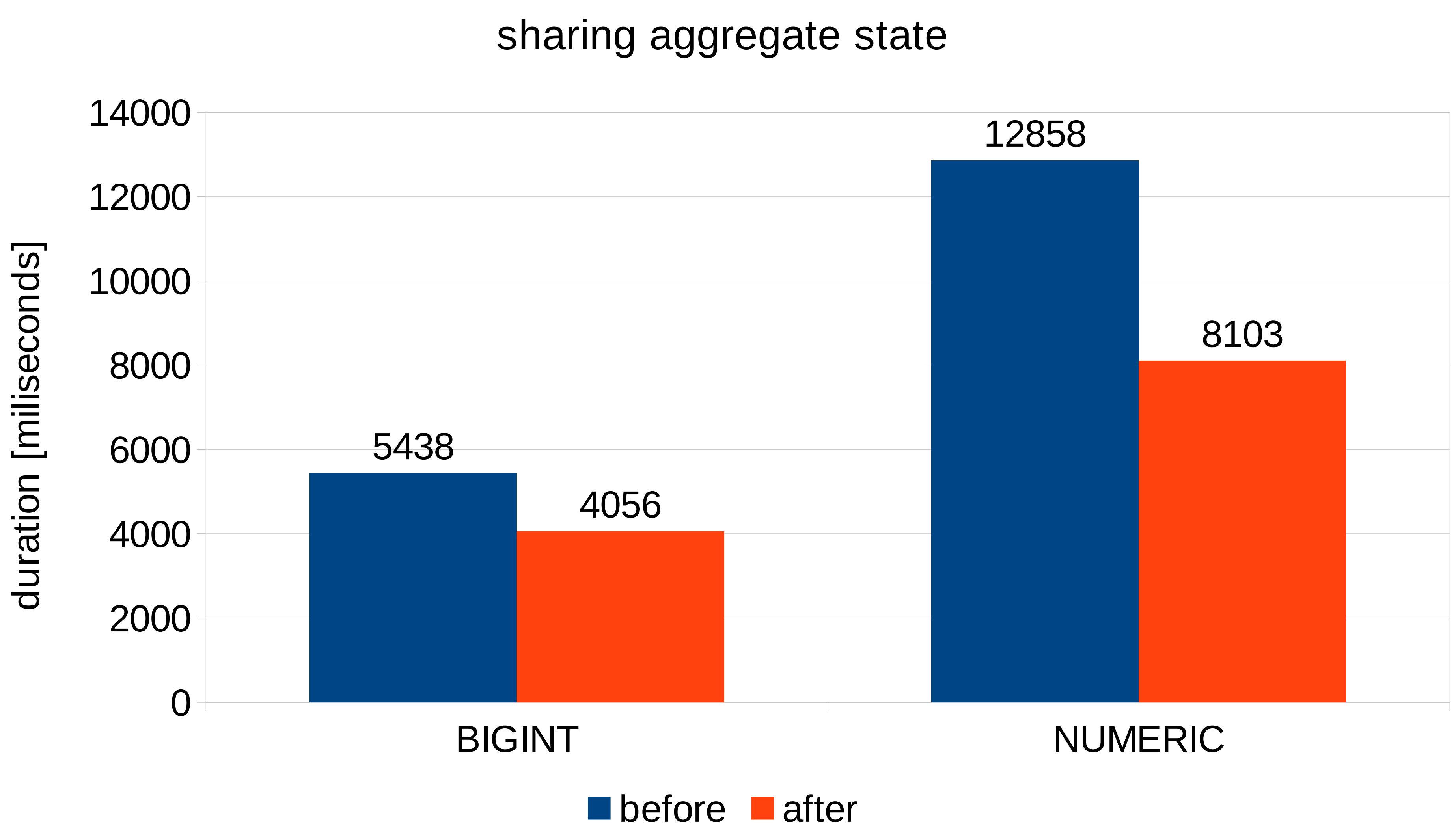
2ndQuadrant
Professional PostgreSQL

Aggregate functions

```
-- table with 50M rows
CREATE TABLE test_aggregates AS
SELECT i AS a
  FROM generate_series(1, 50.000.000) s(i);

-- compute both SUM and AVG on a column
SELECT SUM(a), AVG(a) FROM test_aggregates;
```

2ndQuadrant
Professional PostgreSQL

Aggregate functions



2ndQuadrant
Professional PostgreSQL

Checkpoints

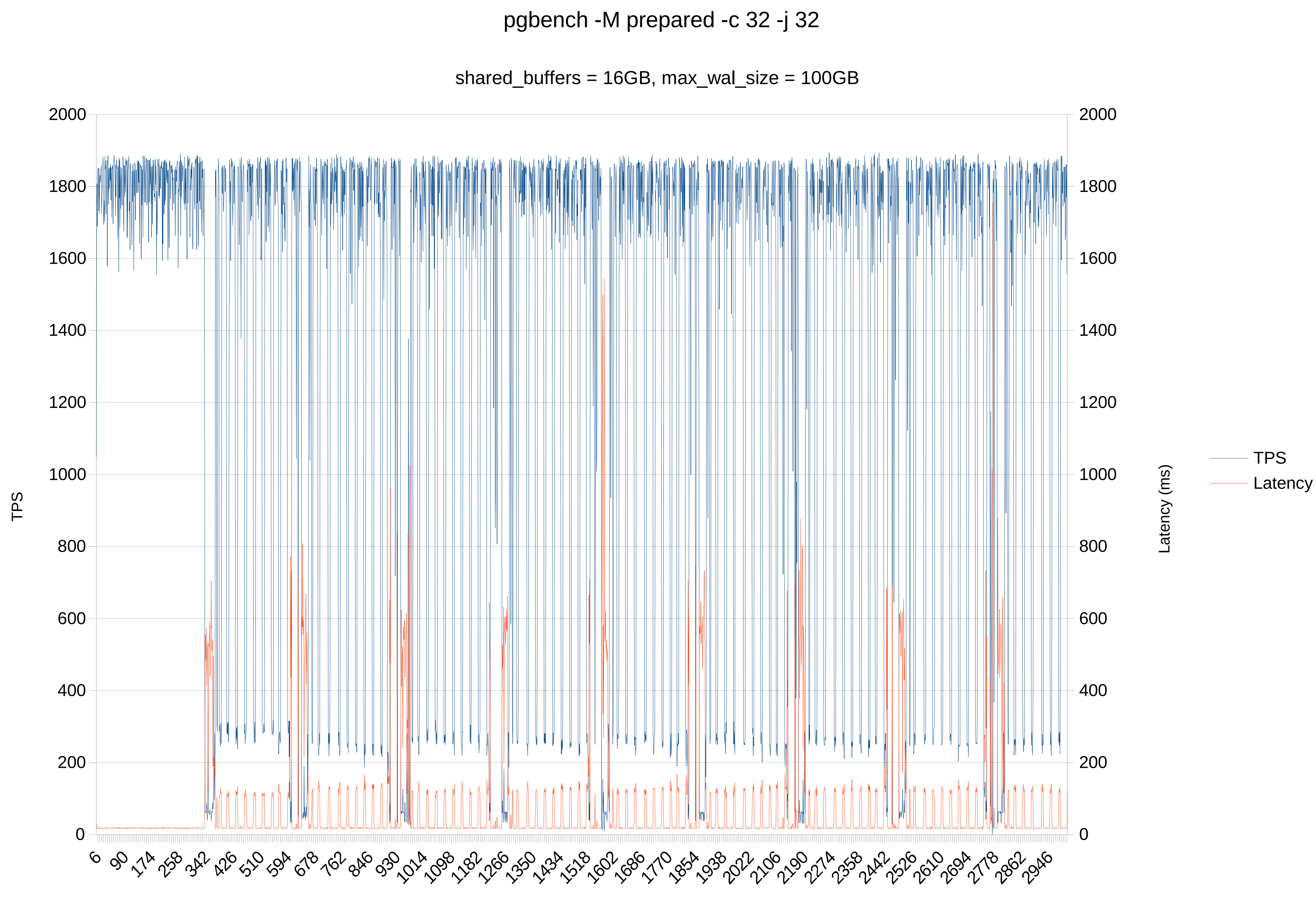
- we need to write dirty buffers to disk regularly
  - data written to page cache (no O_DIRECT)
  - kernel responsible for actual write out
- until now, we simply walked shared buffers
  - random order of buffers, causing random I/O
  - 9.6 sorts the buffers first, to get sequential order
- until now, we only only did fsync at the end
  - a lot of dirty data in page cache, latency spikes
  - 9.6 allows continuous flushing (disabled by default)

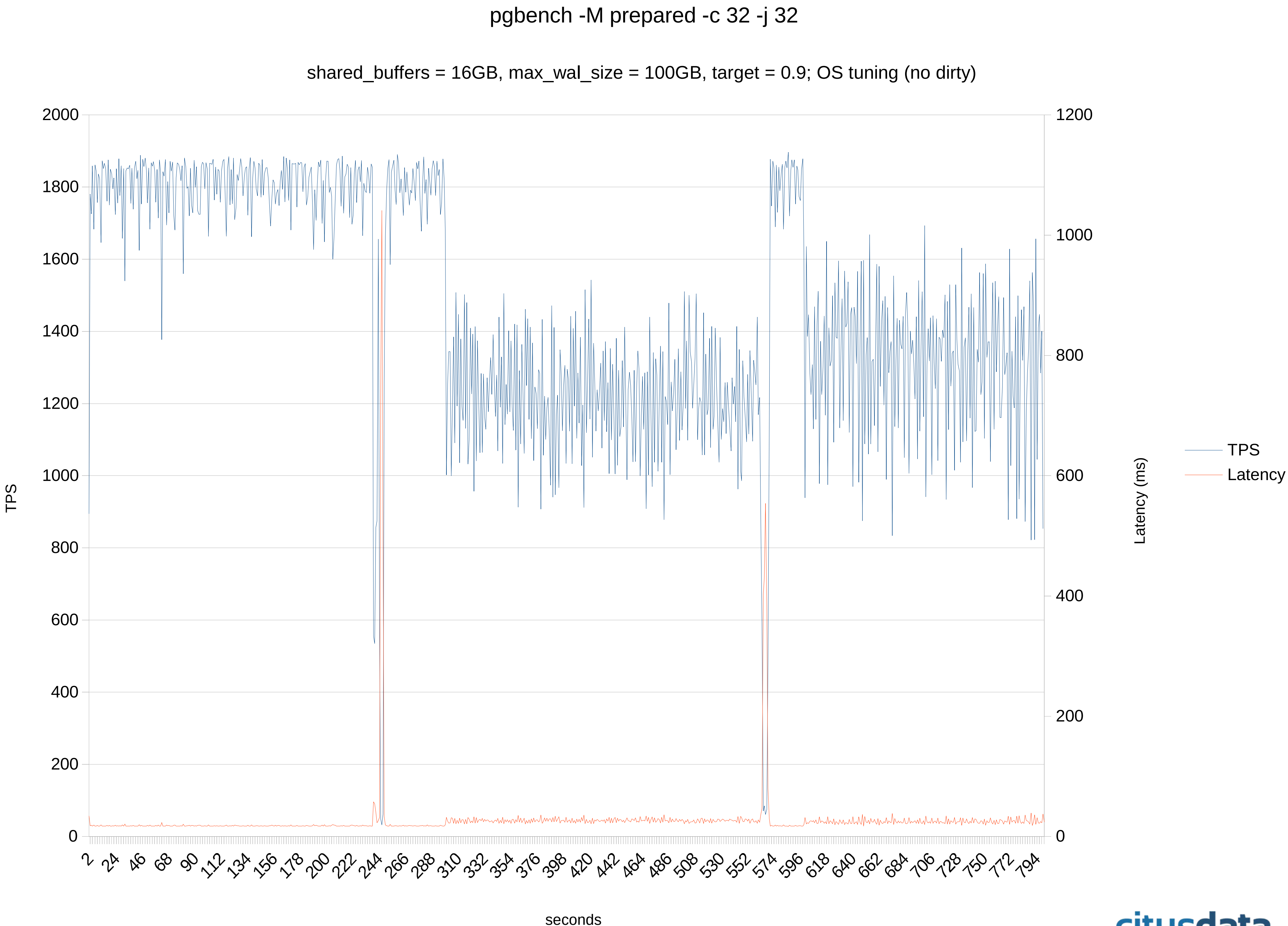
Improving Postgres' Buffer Manager

Andres Freund
PostgreSQL Developer & Committer
Citus Data – citusdata.com - @citusdata

http://anarazel.de/talks/fosdem-2016-01-31/io.pdf

citusdata


pgbench -M prepared -c 32 -j 32
shared_buffers = 16GB, max_wal_size = 100GB
TPS
Latency (ms)
seconds
TPS
Latency
0
200
400
600
800
1000
1200
1400
1600
1800
2000
6
90
174
258
342
426
510
594
678
762
846
930
1014
1098
1182
1266
1350
1434
1518
1602
1686
1770
1854
1938
2022
2106
2190
2274
2358
2442
2526
2610
2694
2778
2862
2946

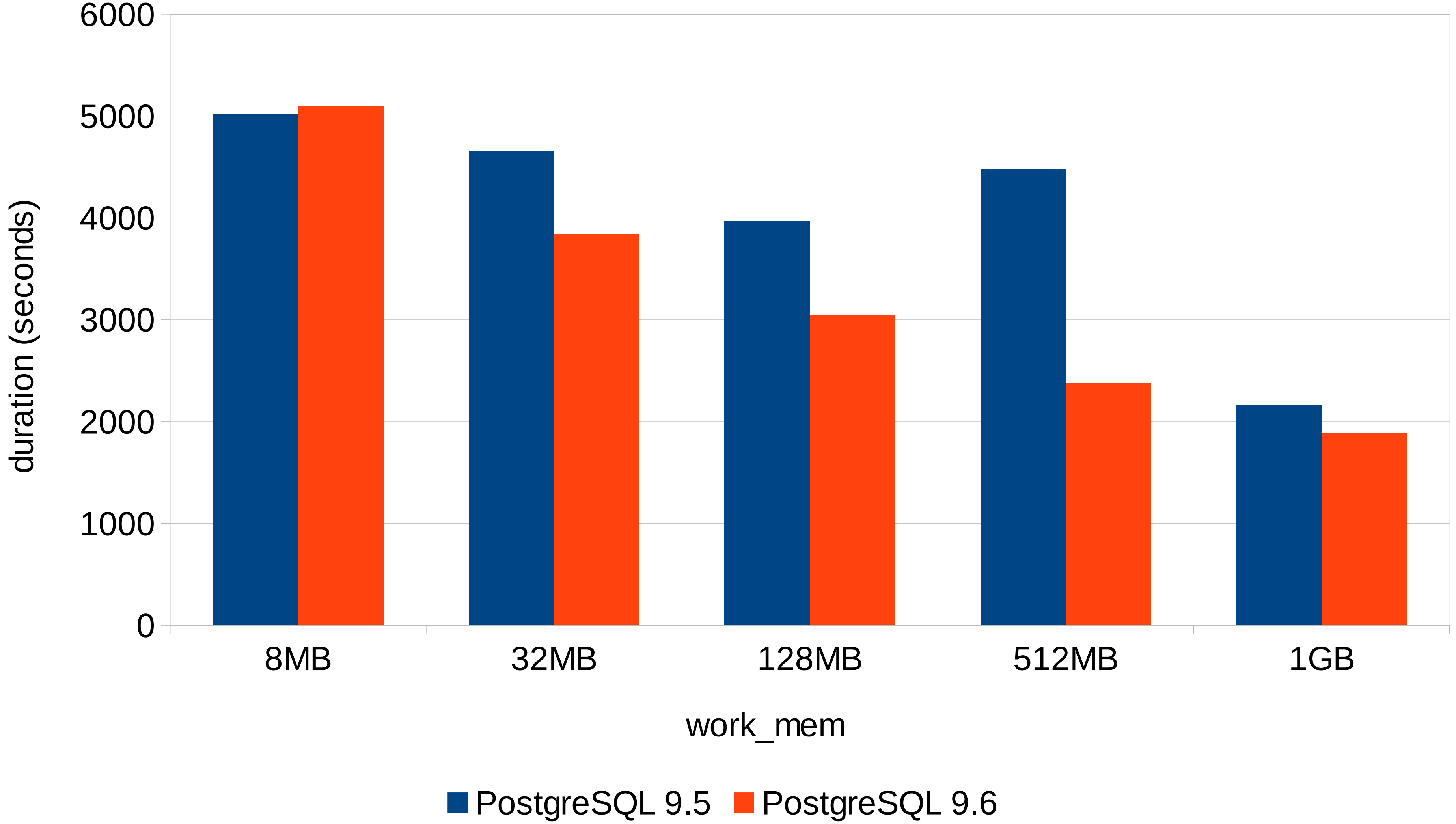

pgbench -M prepared -c 32 -j 32
shared_buffers = 16GB, max_wal_size = 100GB, target = 0.9; OS tuning (no dirty)
TPS
Latency (ms)
seconds
TPS
Latency
0
200
400
600
800
1000
1200
1400
1600
1800
2000
0
200
400
600
800
1000
1200
2
24
46
68
90
112
134
156
178
200
222
244
266
288
310
332
354
376
398
420
442
464
486
508
530
552
574
596
618
640
662
684
706
728
750
772
794

citusdata

Sort (again)

- abbreviated keys extended to
  - additional data types: uuid, bytea, char(n)
  - ordered set aggregates
- use quicksort (instead of replacement selection) for “external sort” case
- … and many other optimizations

Sort performance in 9.5 / 9.6



2ndQuadrant
Professional PostgreSQL

Freezing

- XIDs are 64-bit, but we only store the low 32 bits
  - need to do “freeze” every ~2 billion transactions
  - that means reading all the data (even unmodified parts)
  - problem on large databases (time consuming)
  - users often postpone until it’s too late (outage)
- PostgreSQL 9.6 introduces “freeze map”
  - similar to “visibility map” (and stored in the same file)
  - “all rows on page are frozen” - we can skip this 8kB page

2ndQuadrant
Professional PostgreSQL

Future

- extending parallel query (additional operations)
- declarative partitioning (smart joins, …)
- columnar features
  - vectorized execution, compression, …
  - do more with the same amount of resources
- improving planner
  - correlation statistics, optimizations (unijoins)

Questions?

2ndQuadrant
Professional PostgreSQL