2ndQuadrant®
PostgreSQL

Table Partitioning in PostgreSQL

PgDU Sydney 15th November 2019

David Rowley

About me

PostgreSQL developer @ 2ndQuadrant
PostgreSQL major contributor and committer
Based in Christchurch, New Zealand

Worked on:

- Improving query planner
- Partitioning improvements
- Partial aggregate infrastructure
- Parallel aggregates
- Generally making stuff go faster

Contains Benchmark Results!

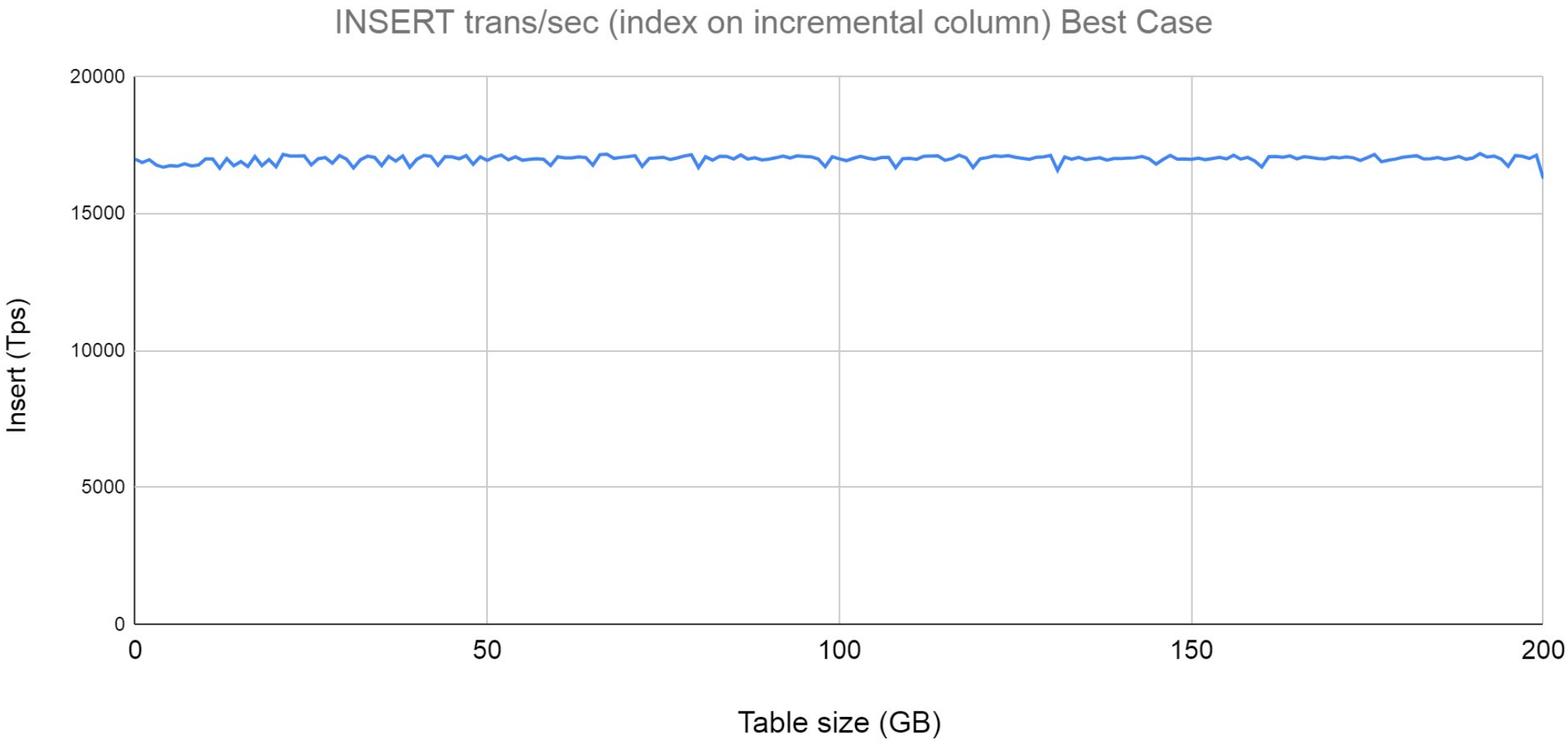
Agenda

- Big table problems
- How using a partitioned the table may help
- Short history of partitioning
- Introduction to partitioning
- Evolution of partitioning
- Application transparency
- Best practices
- The future of partitioning

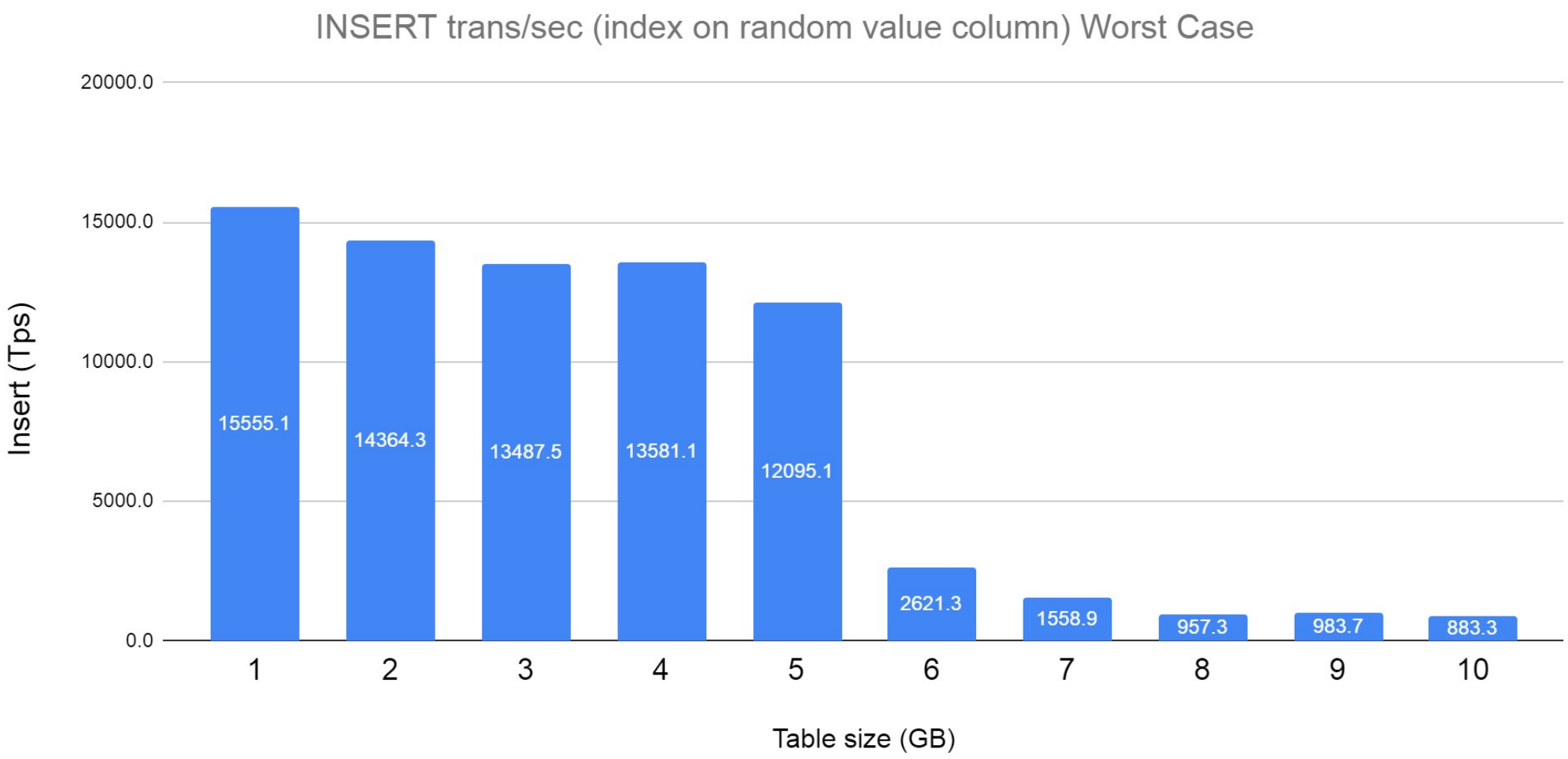
Problems with big tables

- Maximum table size is 32 TB (really 2^32-1 pages)
- Large indexes can become slow to access/update
- No control over data locality
- Slow operations, e.g vacuum.

Problems with big tables



Problems with big tables



Introducing Partitioned Tables

- Declarative partitioning added in v10
- Table Inheritance (DIY partitioning) existed since 8.2
- Allows large tables to be divide horizontally
- Which partition is determined by value of partition key
- Indexes are partitioned too
- Partitioned tables do not store any data
- Data stored in partitions
- Partitions are just tables that are attached to a partitioned table. (Some restrictions apply)

Introducing Partitioned Tables

```
CREATE TABLE tname ( <columns> )
      PARTITION BY LIST ( <col> );

CREATE TABLE tname ( <columns> )
      PARTITION BY RANGE ( <col> , [col2]);

CREATE TABLE tname ( <columns> )
      PARTITION BY HASH ( <col> , [col2]);
```

Example Partitioned Table

```
CREATE TABLE people (
   person_id BIGSERIAL,
   firstname VARCHAR(64) NOT NULL,
   state VARCHAR(3) NOT NULL
) PARTITION BY LIST (state);
```

Creating Partitions (new table)

```
CREATE TABLE people_nsw PARTITION OF
   people FOR VALUES IN ('NSW');

CREATE TABLE people_qld PARTITION OF
   people FOR VALUES IN ('QLD');

CREATE TABLE people_nt_tas PARTITION OF
   people FOR VALUES IN ('NT','TAS');
```

Creating Partitions (existing table)

```
CREATE TABLE people_wa (
   person_id BIGSERIAL,
   firstname VARCHAR(64) NOT NULL,
   state VARCHAR(3) NOT NULL
);
ALTER TABLE people ATTACH PARTITION
people_wa FOR VALUES IN ('WA');
```

Removing Partitions

```
-- Completely remove table
DROP TABLE people_wa;
```

- Or

```
-- Make partition a normal table again
ALTER TABLE people DETACH PARTITION
people_wa;
```

Partition Restrictions

- Table in ATTACH PARTITION must have the same columns/types
- No additional column(s) can exist
- Partitioning constraints cannot overlap

Querying a Partitioned Table

```
# EXPLAIN (COSTS OFF) SELECT * FROM people;
            QUERY PLAN
-----------------------------------
 Append
   ->  Seq Scan on people_nsw
   ->  Seq Scan on people_nt_tas
   ->  Seq Scan on people_qld
(4 rows)
```

Partition Pruning

```
# EXPLAIN (COSTS OFF) SELECT * FROM people
    WHERE state = 'NSW';
                 QUERY PLAN
-------------------------------------------
 Seq Scan on people_nsw
   Filter: ((state)::text = 'NSW'::text)
(2 rows)
```

PG11 would have had an Append node above the Seq Scan

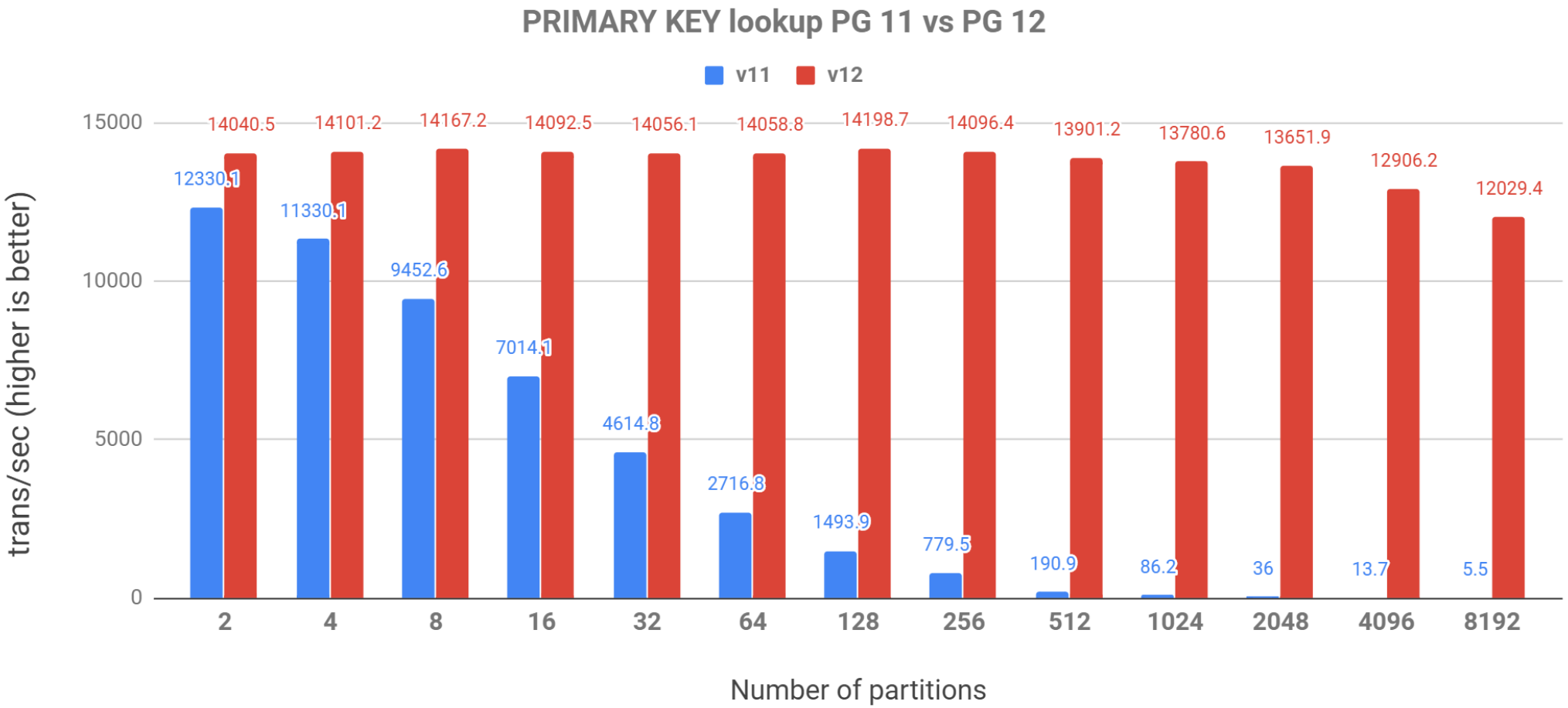
Partition Pruning

```
# EXPLAIN (COSTS OFF) SELECT * FROM people
   WHERE state IN('NSW','QLD');
                          QUERY PLAN
-----------------------------------------------------------
 Append
   ->  Seq Scan on people_nsw
         Filter: ((state)::text = ANY ('{NSW,QLD}'::text[]))
   ->  Seq Scan on people_qld
         Filter: ((state)::text = ANY ('{NSW,QLD}'::text[]))
(5 rows)
```

Partition Pruning Performance

- Many changes done here since v10
- v10
  - checks each partition one by one (slow)
- v11
  - adds new pruning algorithm. Identifies matching partitions quickly
  - load partition metadata -> prune
- v12
  - prune -> load partition metadata

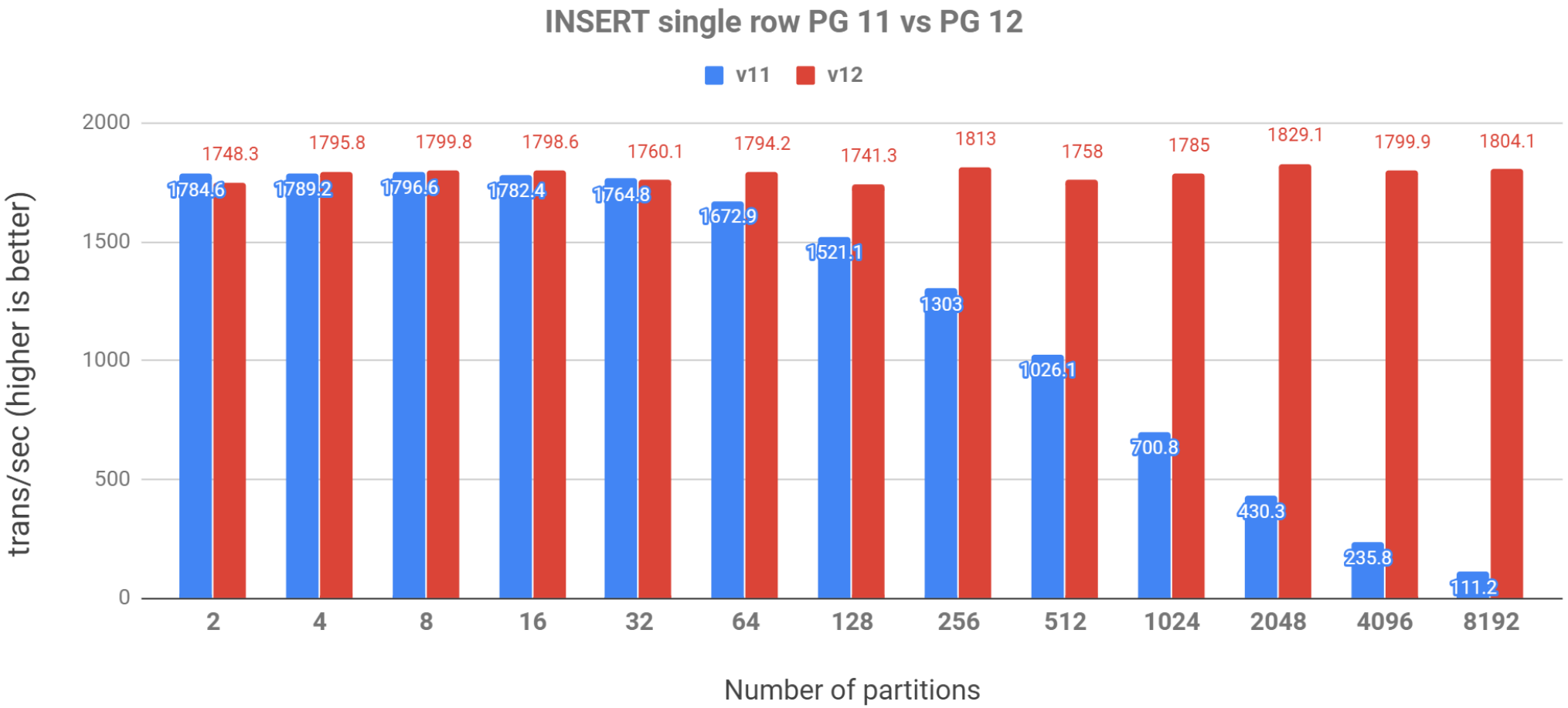
Partition Pruning Performance



INSERTing data

```
# INSERT INTO people (firstname, state) VALUES('Jane', 'NSW');
INSERT 0 1

# INSERT INTO people (firstname, state) VALUES('Scott', 'ACT');
ERROR:  no partition of relation "people" found for row
DETAIL:  Partition key of the failing row contains (state) = (ACT).

# INSERT INTO people_qld (firstname, state) VALUES('Sally', 'QLD');
INSERT 0 1

# INSERT INTO people_qld (firstname, state) VALUES('Mark', 'NSW');
ERROR:  new row for relation "people_qld" violates partition constraint
DETAIL:  Failing row contains (6, Mark, NSW).

# SELECT tableoid::regclass, * FROM people;
  tableoid  | person_id | firstname | state
------------+-----------+-----------+-------
 people_nsw |         4 | Jane      | NSW
 people_qld |         5 | Sally     | QLD
(2 rows)
```

INSERT Performance (single row)



Application Transparency (as of PG 12)

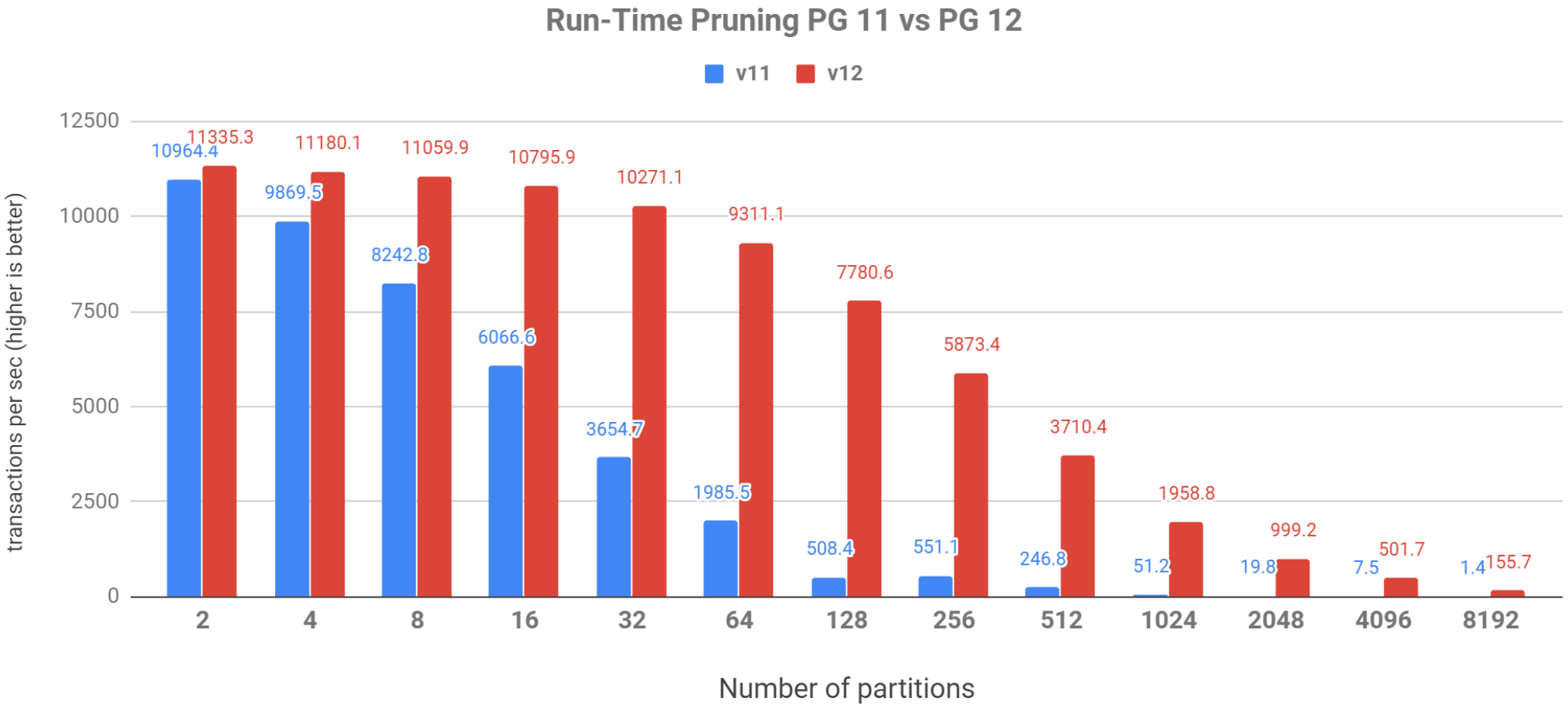
- Can I just use these? Will my application care?
  - PRIMARY KEY / UNIQUE constraints must contain partition key
  - Exclusion constraints not supported yet at partitioned table level. (must define at partition level)
  - UPDATEs to partition key column may result in serialization failures (application can retry on SQLCODE 40001)
  - COPY FREEZE not yet supported.
  - BEFORE ROW triggers cannot be defined on partitioned table (must define at partition level)

Run-time Partition Pruning

- Prunes using values which are unknown during planning.

```
# EXPLAIN (COSTS OFF) SELECT * FROM measurement
       WHERE logdate >= NOW();
                  QUERY PLAN
--------------------------------------
 Append
   Subplans Removed: 2
   ->  Seq Scan on measurement_y2019m11
         Filter: (logdate >= now())
(4 rows)
```

Run-time Partition Pruning Performance



Partition-wise Aggregation

- Exploits the fact that rows can only be stored in a single partition
- Reduces the number of rows to make their way up the plan tree
- Is disabled by default!

```
# enable partition-wise aggregates!
enable_partitionwise_aggregate = on
```

Feature	v9.6	v10	v11	v12
Declarative partitioning	✗	✓	✓	✓
Auto Tuple Routing – INSERT	✗	✓	✓	✓
Auto Tuple Routing – UPDATE	✗	✗	✓	✓
Optimizer Partition Elimination	⚠ [0]	⚠ [0]	✓ [1]	✓
Executor Partition Elimination	✗	✗	✓	✓
Foreign keys	✗	✗	⚠ [2]	✓
Primary Key / Unique Constraints	✗	✗	✓	✓
Default Partitions	✗	✗	✓	✓
ATTACH PARTITION does not block DML	✗	✗	✗	✓
AFTER EACH ROW TRIGGERS	✗	✗	✓	✓
Parallel Append	✗	✗	✓	✓
Foreign Partitions	✗	✗	✓	✓
Partition-wise Aggregates	✗	✗	✓	✓
Partition-wise Joins	✗	✗	✓	✓ [3]

[0] Using constraint exclusion (slow) [1] exists but planning still slow when many partitions are pruned. [2] On partitioned table only. Can't reference a partitioned table. [3] May be improved to work when partitioning bounds are not identical in v13 or beyond.

Best Practises

- Choose your partition strategy and key wisely. Consider:
  - Quick removal of old data
  - Data locality
  - WHERE clauses
- Ensure you have control over the number of partitions which need to be created
- Test to make sure it fixes your performance problems

Trip Hazards (as of v12)

- Large numbers of partitions can make some operations slow, particularly UPDATE/DELETE
- Locking overhead of quick to execute queries when many partitions are run-time pruned
- Memory usage when too many tables are accessed from each backend.

The future?

- More performance improvements!
  - Improve locking?
  - Make UPDATE/DELETE planning faster
- Add additional query planner smarts to take advantage of partitioning!
  - Improvements for time-series data
  - Take more advantage of partition order

Questions?