What's New in PostgreSQL 9.6, by PostgreSQL contributor

NTT OpenSource Software Center
Masahiko Sawada

@PGConf.ASIA 2016 (2 Dec)

Who am I?

- **Masahiko Sawada**
  - Twitter : @sawada_masahiko
- **PostgreSQL Contributor**
  - Freeze Map
  - Multiple Sync Replication
- **PostgreSQL Support**
- **Attended a developer meeting at PGCon 2016**

Photo by Oleg Bartunov

Agenda

1. What’s new in PostgreSQL 9.6
2. Towards to PostgreSQL 10
3. Conclusion

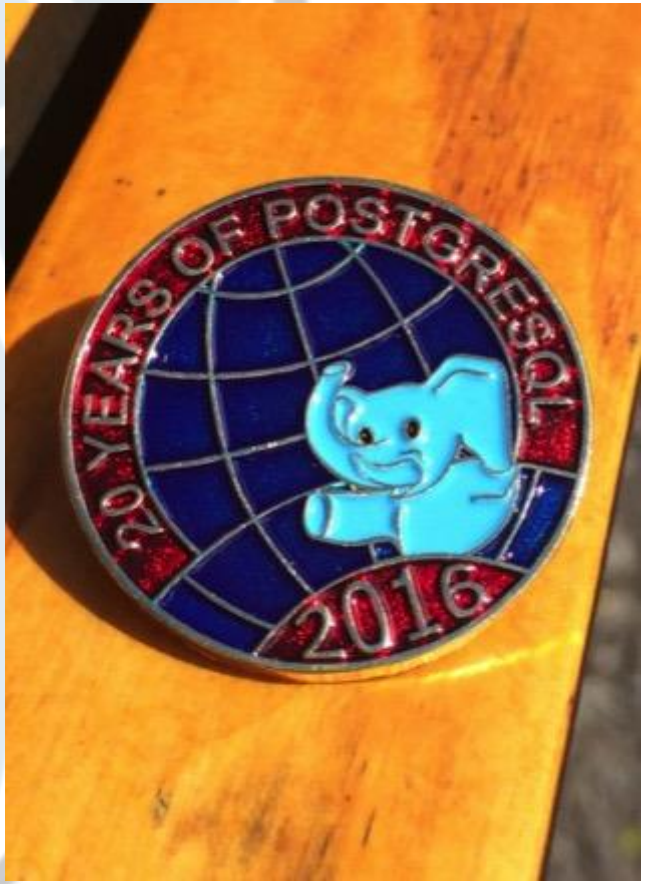
PostgreSQL

- **Open source Relational Database Management System.**

- **Great features.**
  - Window function.
  - Transactional DDL.
  - etc.

- **20th anniversary!**

- **Latest version is 9.6.1 (27th Oct).**
  - Version 9.1 is EOL.



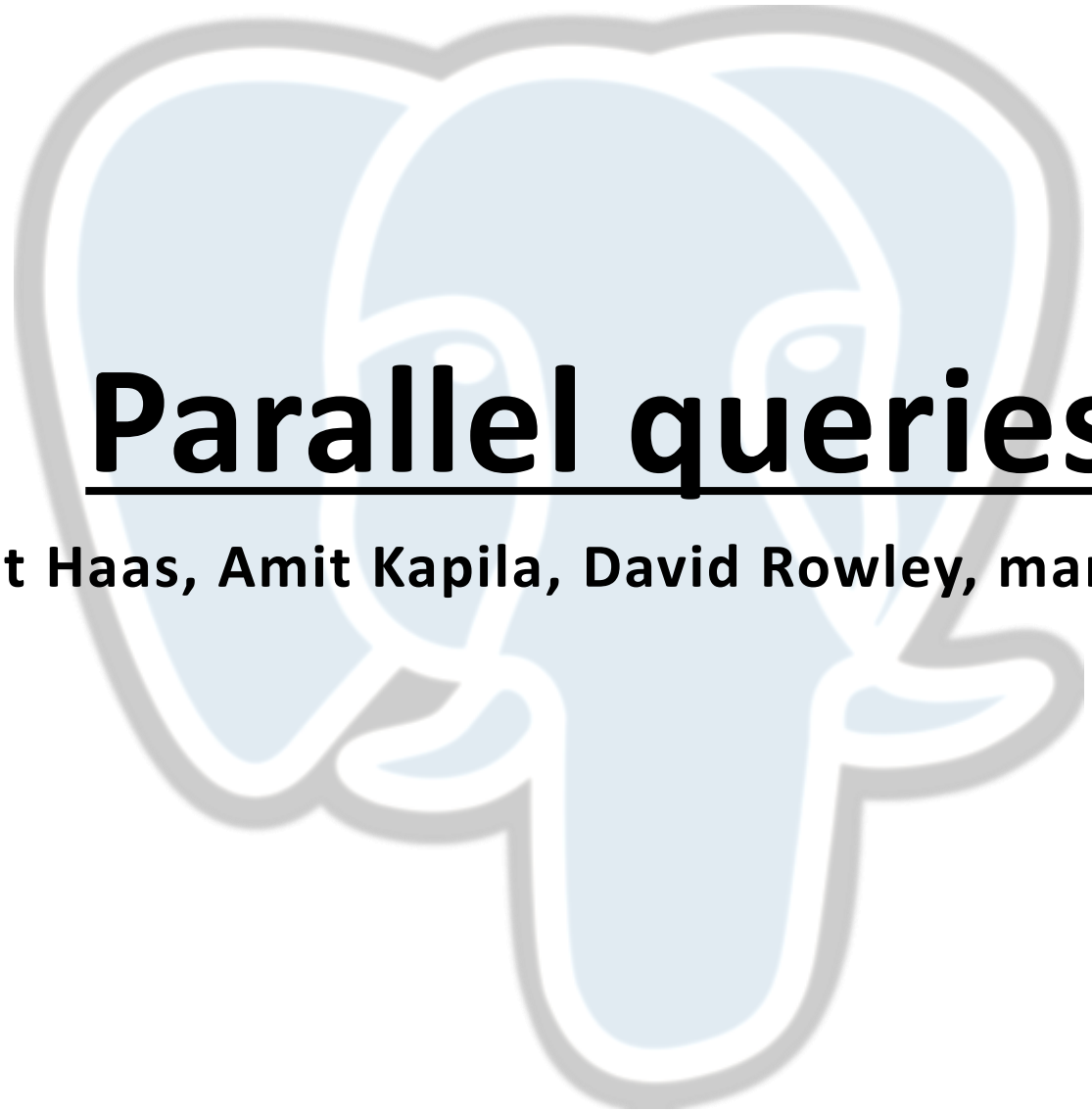
New Features

1. Parallel queries.
2. Avoid VACUUM on all-frozen page (Freeze Map).
3. Monitoring progress of VACUUM.
4. Phrase full text search.
5. Multiple synchronous replication.
6. synchronous_commit = ‘remote_apply’
7. postgrse_fdw support remote jois, sorts, UPDATEs and DELETEs.
8. Trigger Kernel write-back.
9. Better wait information in pg_stat_activity.
10. pg_blocking_pids().

What's new in PostgreSQL 9.6

bridging knowledge to health by paul bica

NTT

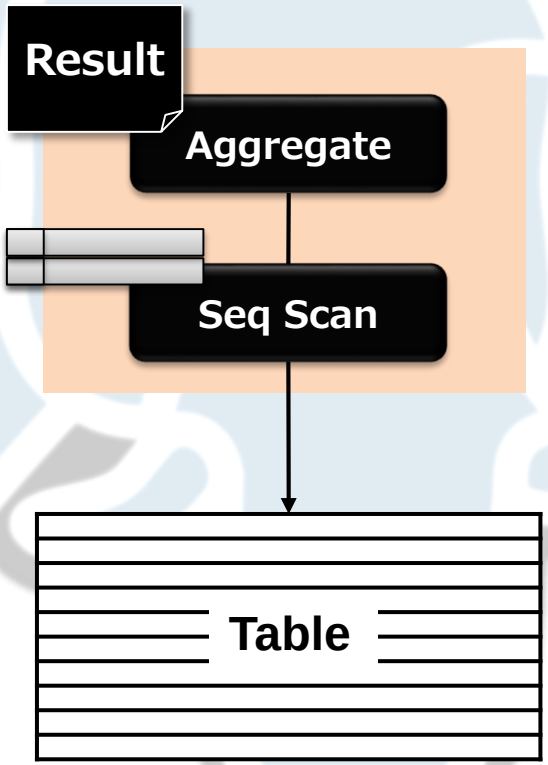


Parallel queries

(Robet Haas, Amit Kapila, David Rowley, many others)

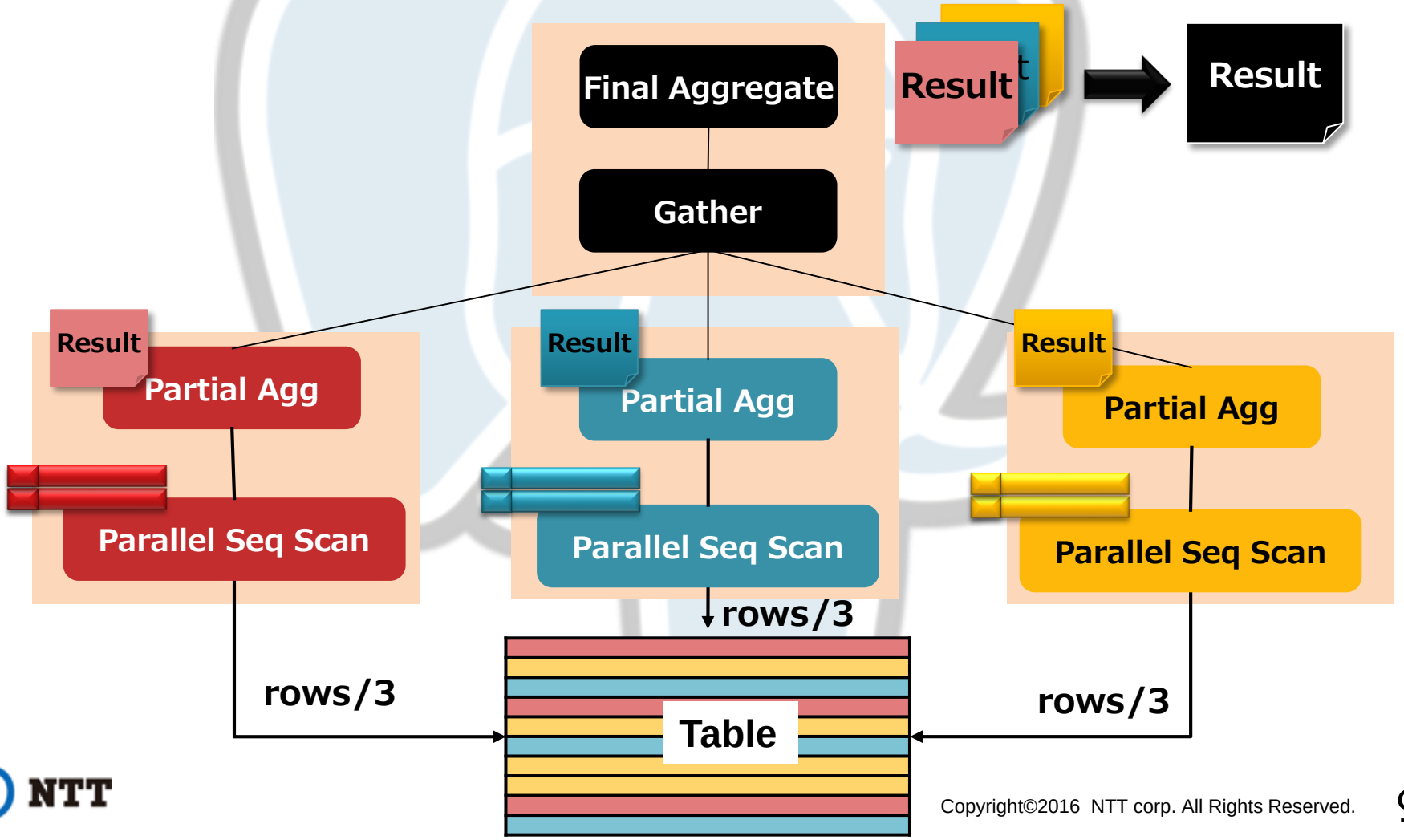
Perpendicularly aggregation

```
=# SELECT count(*) FROM test_table;
```



Parallel aggregation

```
=# SELECT count(*) FROM test_table;
```



Plans

Non Parallel

```
=# EXPLAIN (ANALYZE on, VERBOSE on, COSTS off, TIMING off) SELECT count(*) FROM lineitem;
                                    QUERY PLAN
-----------------------------------------------------------------------------------------
Aggregate (actual rows=1 loops=1)
   Output: count(*)
   ->  Seq Scan on public.lineitem (actual rows=29999795 loops=1)
         Output: l_orderkey, l_partkey, l_suppkey, l_linenumber, l_quantity, ...
```

Parallel

```
=# EXPLAIN (ANALYZE on, VERBOSE on, COSTS off, TIMING off) SELECT count(*) FROM lineitem;
                                    QUERY PLAN
-----------------------------------------------------------------------------------------
 Finalize Aggregate (actual rows=1 loops=1)
   Output: count(*)
   ->  Gather (actual rows=3 loops=1)
         Output: (PARTIAL count(*))
         Workers Planned: 2
         Workers Launched: 2
         ->  Partial Aggregate (actual rows=1 loops=3)
               Output: PARTIAL count(*)
               Worker 0: actual rows=1 loops=1
               Worker 1: actual rows=1 loops=1
               ->  Parallel Seq Scan on public.lineitem (actual rows = ...
                     Worker 0: actual rows=9838356 loops=1
                     Worker 1: actual rows=10019336 loops=1
```

As of 9.6 parallel queries

- **Support**
  - Sequential Scan
  - Aggregate (e.g. count, sum, avg)
  - Nested Loop Join
  - Hash Join

- **Not Support**
  - UPDATE, DELETE
  - Index Scan
  - Sorting
  - Merge Join
  - DDL
  - etc

Evaluation on a great machine

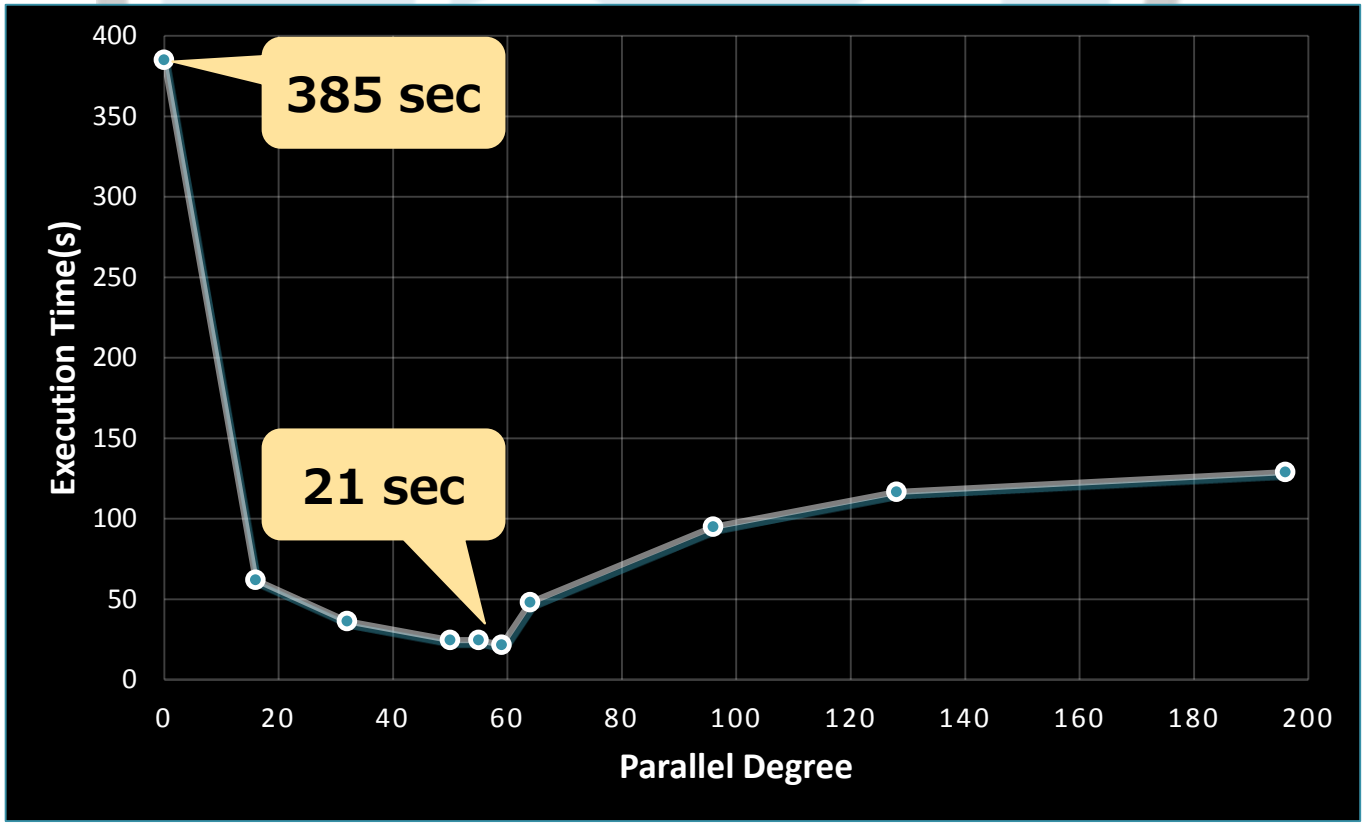
- **HPE ProLiant DL580 GEN9**
- **Intel Xeon E7-8890 v4 2.20GHz (4P/192 core (96-HT))**
- **2TB RAM**
- **Workload Accelerator(PCIe SSD)**
  - Read : 715,000 IOPS, 3.0GB/s
  - Write : 95,000 IOPS, 2.5GB/s

Supported by Hewlett Packard Enterprise

Thank you!

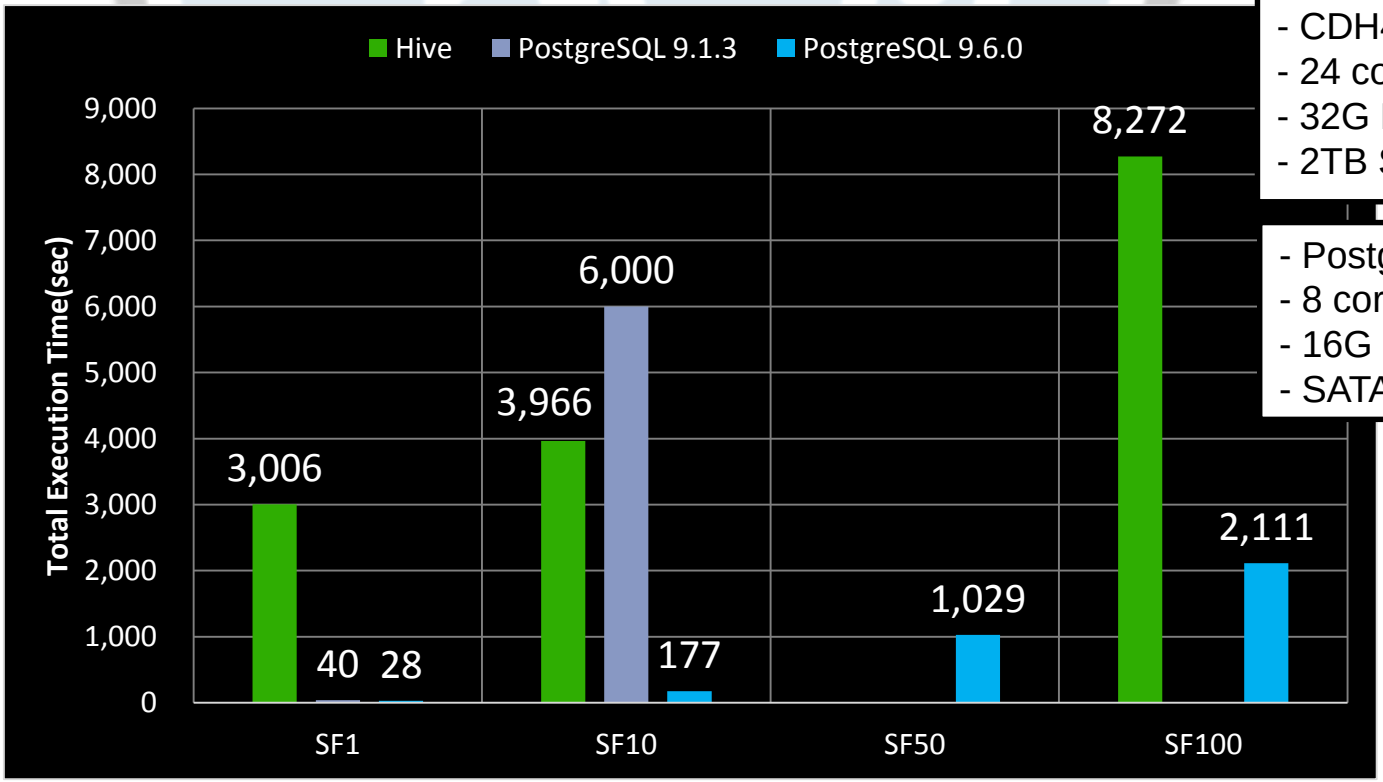
Parallel Query on 192 cores machine

- All table data(400GB) is on the shared buffer; no disk access.
- Simple aggregation, count(*).
- Parallel query makes aggregation **19x faster!!**



For your reference

- Comparing DBT-3 benchmark result with Hive (SF1 - SF100)
- Single PostgreSQL node, 192 parallel degree.
  - 7 of 22 queries timed out(30min) at SF500. shared_buffers = 800GB, work_mem = 3GB.
- 12 node Hadoop cluster (master 2, slave 10).
- Hive flushes data to disk whenever finished each job.
  - Now in 2016 we should use Hive on Tez.
- On the other hand, PostgreSQL has the all table data in shared buffer.



- CDH4.1.2
- 24 cores (12-HT)
- 32G RAM
- 2TB SATA

- PostgreSQL 9.1.3
- 8 cores (4-HT)
- 16G RAM
- SATA

Avoid VACUUM on all-frozen page (Freeze Map)

(Masahiko Sawada, Robert Haas, Andres Freund)

Freezing of database

- Necessary to prevent transaction ID(XID) wraparound failures.
- anti-wraparound VACUUM is invoked every 200 million transaction by default
- Previously it always scanned all table pages.
  - Could be performance degradation.

Avoid VACUUM on all-frozen page (Freeze Map)

- Keep track of which pages are completely frozen.
- Avoid VACUUM on all-frozen pages.
  - Very effective for mostly-read tables.

```
-- 9.5
=# VACUUM FREEZE large_table;
VACUUM
Time: 685363.793 ms

=# VACUUM FREEZE large_table;
VACUUM
Time: 711380.587 ms
```

```
-- 9.6
=# VACUUM FREEZE large_table;
VACUUM
Time: 703509.523 ms

=# VACUUM FREEZE large_table;
VACUUM
Time: 222.719 ms
```

Monitoring progress of VACUUM

(Amit Langote, Robert Haas, Vinayak Pokale, Rahila Syed)

Progress information of vacuum

- Introduce new system view pg_stat_progress_vacuum.
- Report progress of running VACUUM.
- Not supported for VACUUM FULL and CLUSTER.

```
=# ¥d pg_stat_progress_vacuum
       Column       |   Type   | Modifiers
--------------------+----------+-----------
 pid                | integer  |
 datid              | oid      |
 datname            | name     |
 relid              | oid      |
 phase              | text     |
 heap_blks_total    | bigint   |
 heap_blks_scanned  | bigint   |
 heap_blks_vacuumed | bigint   |
 index_vacuum_count | bigint   |
 max_dead_tuples    | bigint   |
 num_dead_tuples    | bigint   |
```

Monitoring progress of VACUUM

```
=# SELECT pid, datname, relname,
          now() - query_start as duration,
          ((heap_blks_scanned / heap_blks_total::numeric(10,2)) * 100) as
          percentage,
          p.phase,
          index_vacuum_count
   FROM pg_stat_progress_vacuum as p, pg_class as c
   WHERE p.relid = c.oid;

-[ RECORD 1 ]-------+-----------------
 pid                | 100026
 datname            | postgres
 relname            | pgbench_accounts
 duration           | 01:23:45.000000
 percentage         | 19.72
 phase              | scanning heap
 index_vacuum_count | 10
-[ RECORD 2 ]-------+-----------------
 pid                | 100027
 datname            | postgres
 relname            | very_large_table
 duration           | 02:35:12.123456
 percentage         | 95.12
 phase              | scanning heap
 index_vacuum_count | 300
```

- Table name
- Duration
- Progress (percentage)
- Phase
- Index vacuum count

Phrase full text search

(Teodor Sigaev, Oleg Bartunov, Dmitry Ivanov)

Search for Phrases

- Search for words positioned relative to other words.
- Added new tsquery operators ‘<->’ and ‘<N>’

- **‘index <-> scan’** means that ‘scan’ follows by ‘index’.
  - Match to ‘index scan’.
- **‘index <2> scan’** means that ‘index’ and ‘scan’ separated by at most 1 other word.
  - Match to ‘index only scan’.

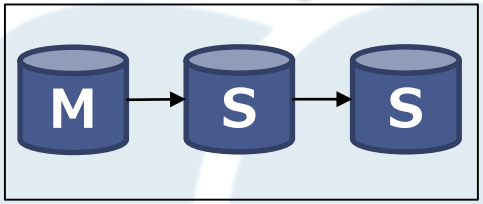
Search for ‘parallel <->/<N> execute’

```
=# SELECT title, body FROM pgdoc
   WHERE content @@ to_tsquery('parallel <-> execute') LIMIT 1;
-[ RECORD 1 ]-------------------------
 title    | Foreign Data Wrapper Callback Routines
 body     | A ForeignScan node can, optionally, support
          | parallel execution. …

=# SELECT title, body FROM pgdoc
   WHERE content @@ to_tsquery('parallel <2> execute') LIMIT 1;
-[ RECORD 1 ]-------------------------
 title    | When Can Parallel Query Be Used?
 body     | Even when parallel query is generated for …
          | impossible to execute that plan in parallel at execution …

=# SELECT title, body FROM pgdoc
   WHERE content @@ to_tsquery('parallel <3> execute') LIMIT 1;
-[ RECORD 1 ]-------------------------
 title    | How Parallel Query Works
 body     | Every background worker process which is …
          | parallel query will execute the portion of …
```

Multiple synchronous replication

(Masahiko Sawada, Beena Emerson, Michael Paquier, Fujii Masao, Kyotaro Horiguchi)

History of Replication feature



Asynchronous Replication

Cascade Replication

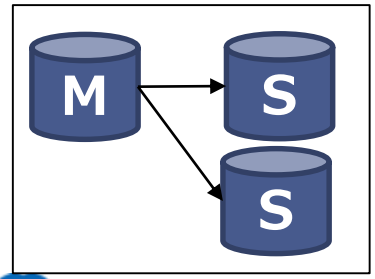
- Replication Slot
- Logical Decoding

9.0 (2010) | 9.1 (2011) | 9.2 (2012) | 9.3 (2013) | 9.4 (2014) | 9.5 (2015) | 9.6 (2016)

Synchronous Replication

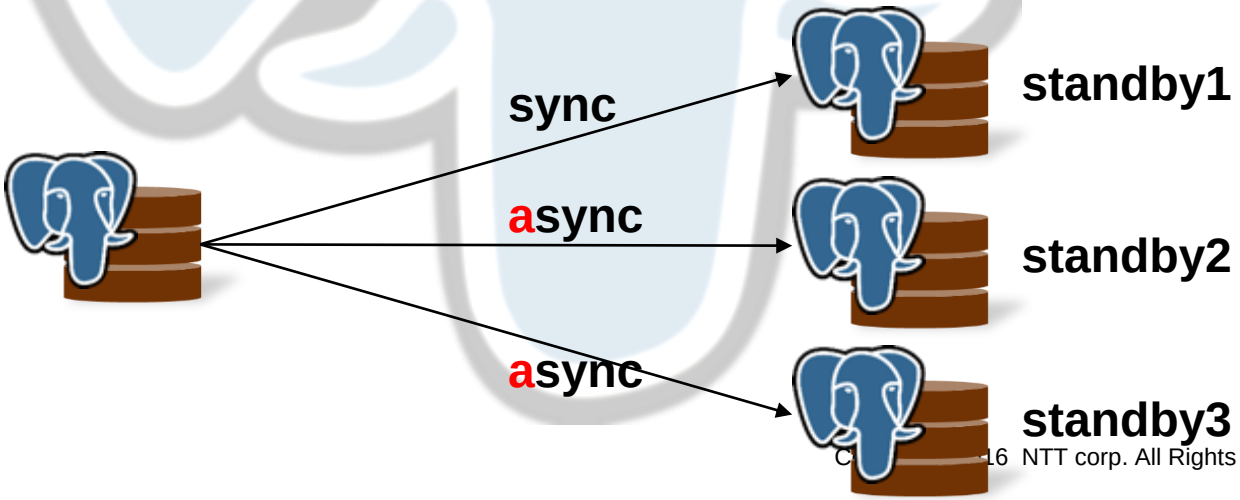
Fast Failover

- WAL Compression
- Fast Failback (pg_rewind)



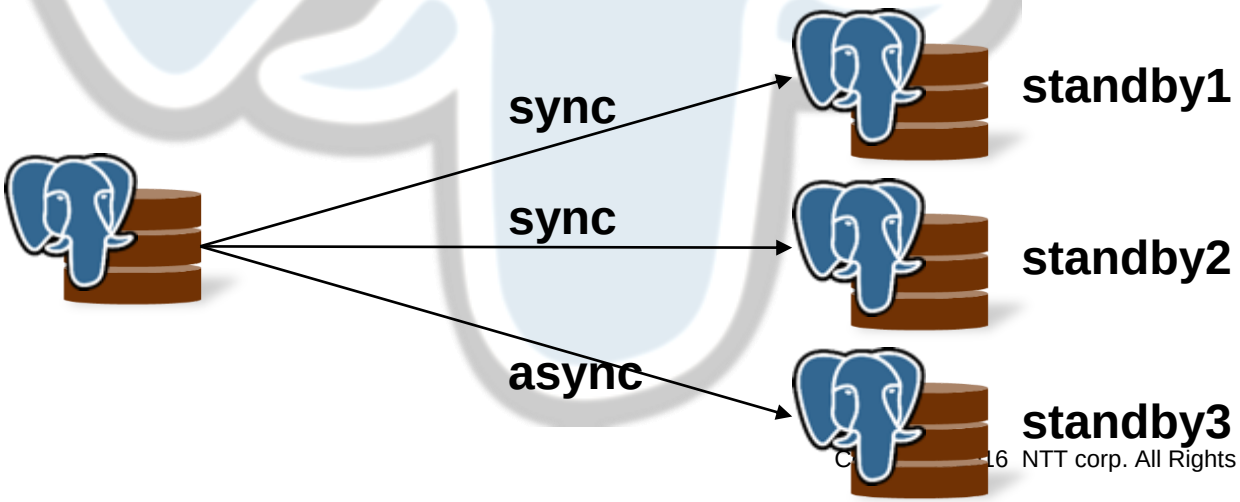
Synchronous Replication (~9.5)

```
=# SHOW synchronous_standby_names;
   synchronous_standby_names
-----------------------------
 standby1, standby2, standby3

=# SELECT application_name, sync_state, sync_priority
   FROM pg_stat_replication;
 application_name | sync_state | sync_priority
------------------+------------+---------------
 standby1         | sync       |             1
 standby2         | potential  |             2
 standby3         | potential  |             3
```



Multiple Synchronous Replication (9.6~)

```
=# SHOW synchronous_standby_names;
    synchronous_standby_names
---------------------------------
 2(standby1, standby2, standby3)

=# SELECT application_name, sync_state, sync_priority
   FROM pg_stat_replication;
 application_name | sync_state | sync_priority
------------------+------------+---------------
 standby1         | sync       |             1
 standby2         | sync       |             2
 standby3         | potential  |             3
```



New syntax of synchronous_standby_names

- When all three standbys are available,
  - **standby1** and **standby2** are synchronous standbys.
  - **standby3** is potential standby (async).
- After **standby1** crashed,
  - **standby3** becomes to synchronous standby.
  - **standby2** and **standby3** are synchronous standbys.

synchronous_commit = ‘remote_apply’

(Thomas Munro)

synchronous_commit = remote_apply

New!

synchronous_commit = [off | local | remote_write | on | remote_apply]

	On master server	On standby server		
	Flush WAL	Write WAL	Flush WAL	Apply WAL
off	Not Wait	Not Wait	Not Wait	Not Wait
local	Wait	Not Wait	Not Wait	Not Wait
remote_write	Wait	Wait	Not Wait	Not Wait
on	Wait	Wait	Wait	Not Wait
remote_apply	Wait	Wait	Wait	Wait

Read balancing with remote_apply

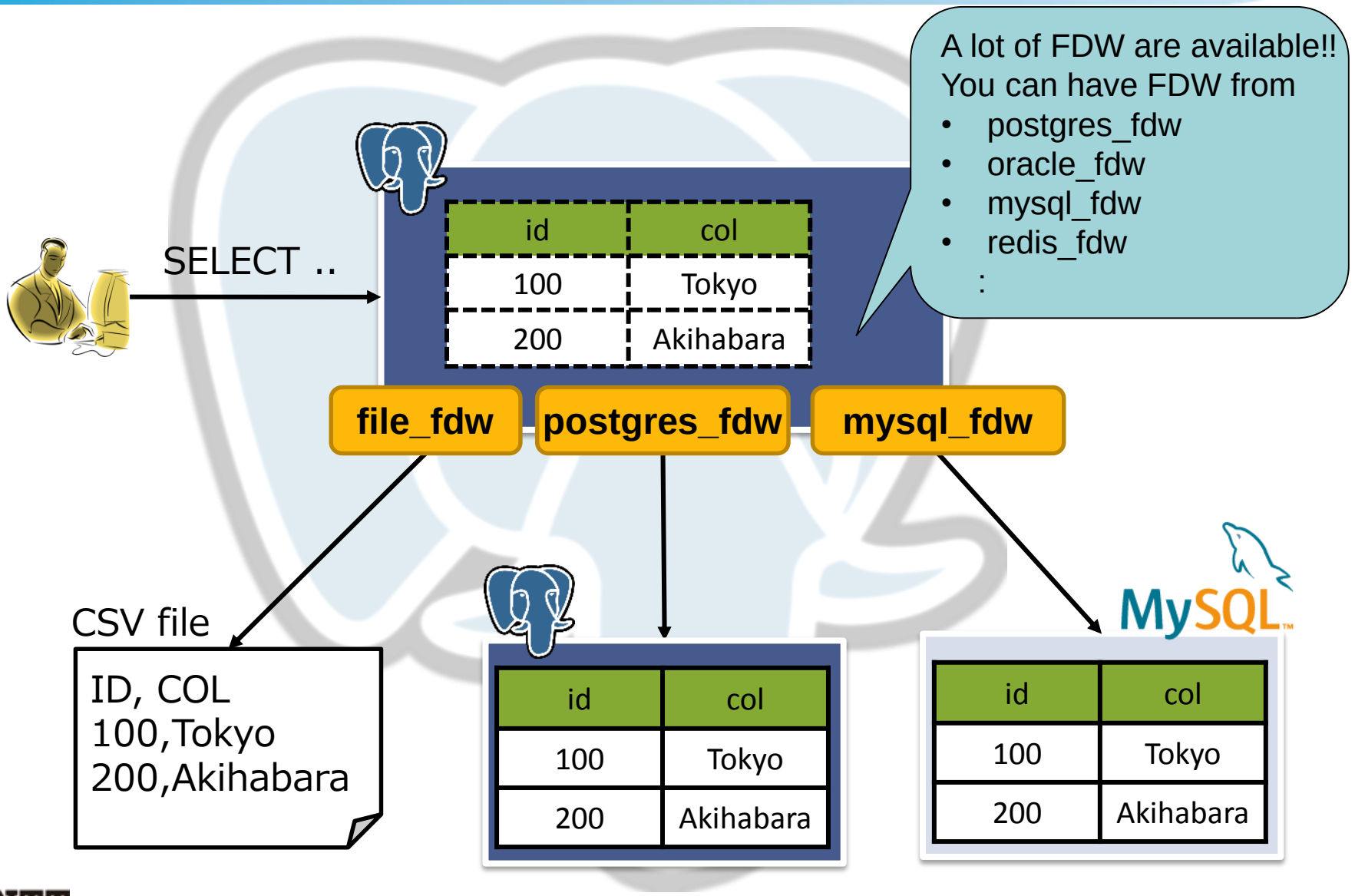
- With remote_apply and synchronous replication, committed data is visible on both the master server and the slave server.

- The client can always see updated data even on the slave server.

Postgres_fdw support remote joins, sorts, UPDATEs and DELETEs

(Etsuro Fujita, Shigeru Hanada, Ashutosh Bapat)

What's the FDW?



Improvement of FDW API and postgres_fdw

Operation	PostgreSQL 9.4	PostgreSQL 9.5	PostgreSQL 9.6
SELECT	-	Support foreign table inheritance	Support foreign table inheritance
WHERE clause	Push down	Push down	Push down
Aggregate	Local	Local	Local
Sort	Local	Local	Push down
Join	Local	Local	Push down
UPDATE, DELETE	Tuple based using CURSOR	Tuple based using CURSOR	Directly execution
INSERT	INSERT to remote server using Prepare/Execute	INSERT to remote server using Prepare/Execute	INSERT to remote server using Prepare/Execute

Sort push down

```
-- 9.5
=# EXPLAIN (VEROBSE on, COSTS off) SELECT * FROM f_table ORDER BY col DESC;
                       QUERY PLAN
-------------------------------------------------
 Sort
   Output: col
   Sort Key: f_table.col DESC
   ->  Foreign Scan on public.f_table
         Output: col
         Remote SQL: SELECT col FROM public.f_table
```

```
-- 9.6
=# EXPLAIN (VEROBSE on, COSTS off) SELECT * FROM f_table ORDER BY col DESC;
                                QUERY PLAN
---------------------------------------------------------------------------
 Foreign Scan on public.f_table
   Output: col
   Remote SQL: SELECT col FROM public.f_table ORDER BY col DESC NULLS FIRST
```

Join push down

```
-- 9.5
=# EXPLAIN (VERBOSE on, COSTS off) SELECT * FROM f_table a JOIN f_table2 b ON a.col = b.col LIMIT 10;
                                QUERY PLAN
---------------------------------------------------------------------------
 Limit
   Output: a.col, b.col
   ->  Nested Loop
         Output: a.col, b.col
         Join Filter: (a.col = b.col)
         ->  Foreign Scan on public.f_table a
               Output: a.col
               Remote SQL: SELECT col FROM public.f_table
         ->  Materialize
               Output: b.col
               ->  Foreign Scan on public.f_table2 b
                     Output: b.col
                     Remote SQL: SELECT col FROM public.f_table2
```

```
-- 9.6
=# EXPLAIN (VERBOSE on, COSTS off) SELECT * FROM f_table a JOIN f_table2 b ON a.col = b.col LIMIT 10;
                                       QUERY PLAN
-----------------------------------------------------------------------------------------------
 Limit
   Output: a.col, b.col
   ->  Foreign Scan
         Output: a.col, b.col
         Relations: (public.f_table a) INNER JOIN (public.f_table2 b)
         Remote SQL: SELECT r1.col, r2.col FROM (public.f_table r1 INNER JOIN public.f_table2 r2 ON
(((r1.col = r2.col))))
```

Trigger kernel writeback

(Fabien Coelho, Andres Freund)

Kernel write-back configurations

- PostgreSQL writes data to the kernel's disk cache.
- Write-back could be cause I/O storms
- Can be configured on global level
  - vm.dirty_background_ratio etc

Kernel write-back configurations

- New configure parameters
  - checkpoint_flush_after
    - 256kB by default
  - bgwriter_flush_after
    - 512kB by default
  - backend_flush_after
    - 0 by default
  - wal_writer_flush_after
    - 1MB by default
- Enable by default on Linux only

Better wait information in pg_stat_activity

(Amit Kapila, Ildus Kurbangaliev)

Details of wait information in pg_stat_activity

- **Tracking of wait event.**
- **Removed ‘waiting’ column**
- **Added ‘wait_event_type’ column**
  - The type of event for which the backend is waiting.
- **Added ‘wait_event’ column**
  - Wait event name.

```
=# SELECT pid, query, wait_event_type, wait_event
   FROM pg_stat_activity;
  pid  |       query          | wait_event_type |      wait_event
-------+----------------------+-----------------+--------------------
 12345 | SELECT * FROM hoge;  |                 |
 90000 | VACUUM FULL hoge;    | Lock            | relation
 10000 | SELECT * FROM bar;   | LWLockNamed     | XidGenLock
 20000 | UPDATE bar SET…;     | LWLockTranche   | replication_slot_io
 30000 | INSERT INTO …        | BufferPin       | BufferPin
(5 rows)
```

pg_blocking_pids() function

(Tom Lane)

pg_blocking_pids() function

- Returns an array of the PIDs that are blocking the session given PID.

```
=# SELECT pid, query, wait_event_type, wait_event
   FROM pg_stat_activity;
  pid  |       query         | wait_event_type | wait_event
-------+---------------------+-----------------+------------
 12345 | SELECT * FROM hoge; |                 |
 90000 | VACUUM FULL hoge;   | Lock            | relation
(2 rows)

=# SELECT pg_blocking_pids(90000);
 pg_blocking_pids
------------------
 {12345}
(1 row)
```

Towards to PostgreSQL 10

Future Energy by Floris Oosterveld

NTT

New versioning scheme

~ 9.6

9. 5. 0

Major ver. Minor ver.

9. 6. 0

9. 5. 1

10 ~

10. 0

Major ver. Minor ver.

11. 0

10. 1

PostgreSQL 10 Roadmap is available

- **A number of companies publish its roadmap for PostgreSQL 10.**
  - https://wiki.postgresql.org/wiki/PostgreSQL10_Roadmap

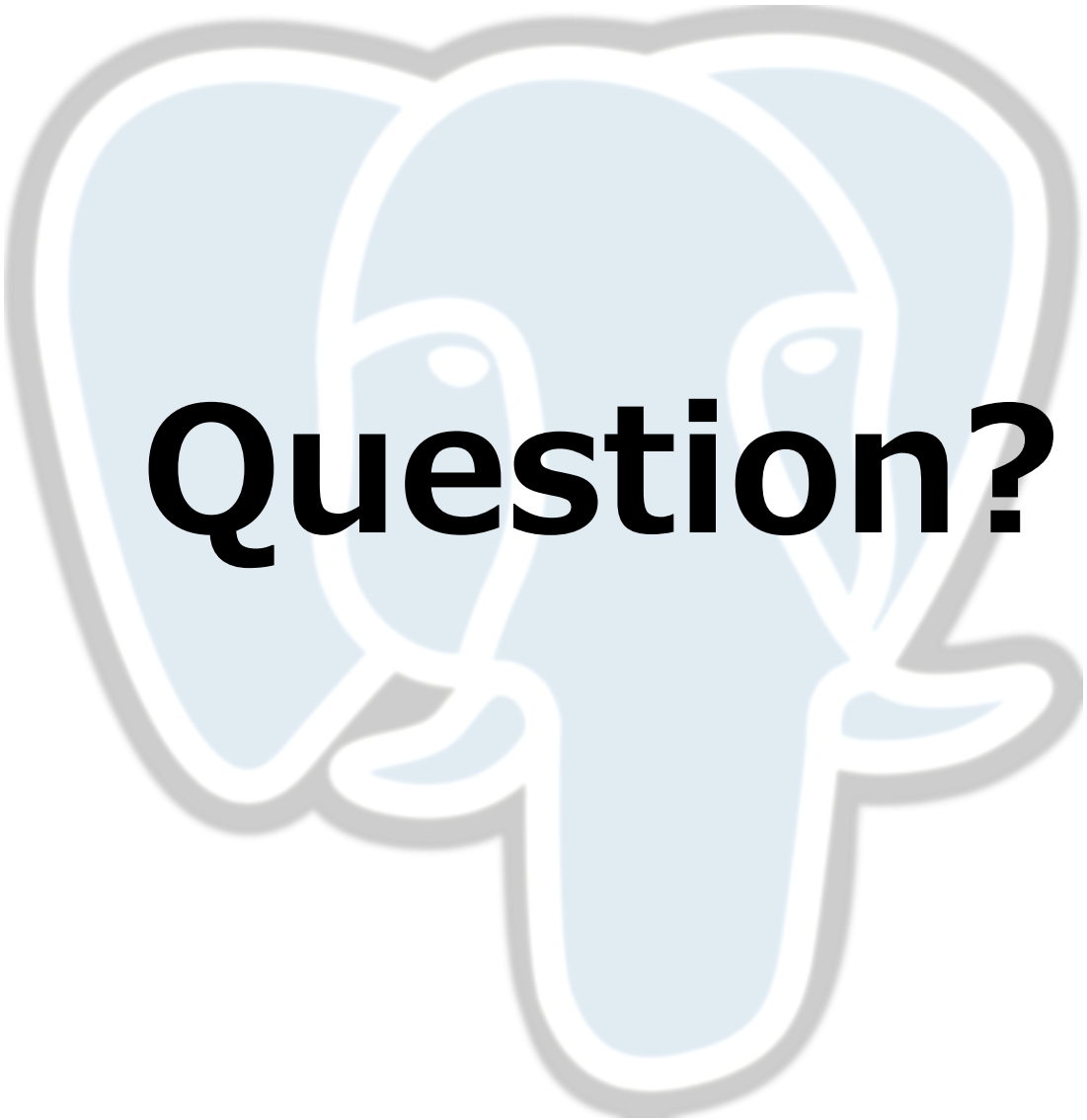
- **Feedback is very welcome!!**

Conclusion

NTT

Conclusion

- Over 200 new feature and improvement.
- Scale up with Parallel Query.
- Scale out with Synchronous Replication and postgres_fdw.
- Easier to use of very large database.



Question?