AUTO PLAN TUNING USING FEEDBACK LOOP

PGConf.EU 2018

Tatsuro Yamada
NTT Open Source Software Center

Who I am?

Innovative R&D by NTT

- **Tatsuro Yamada**
  - From Tokyo, Japan

- **Work**
  - for NTT Open Source Software Center
  - Database consulting for NTT Group companies
  - Oracle_fdw committer
  - Organizers of PGConf.Asia

- **Interest**
  - Listening to Bossa-nova and Jazz samba
  - Skiing, Craft beer
  - Plan tuning

NTT

Agenda

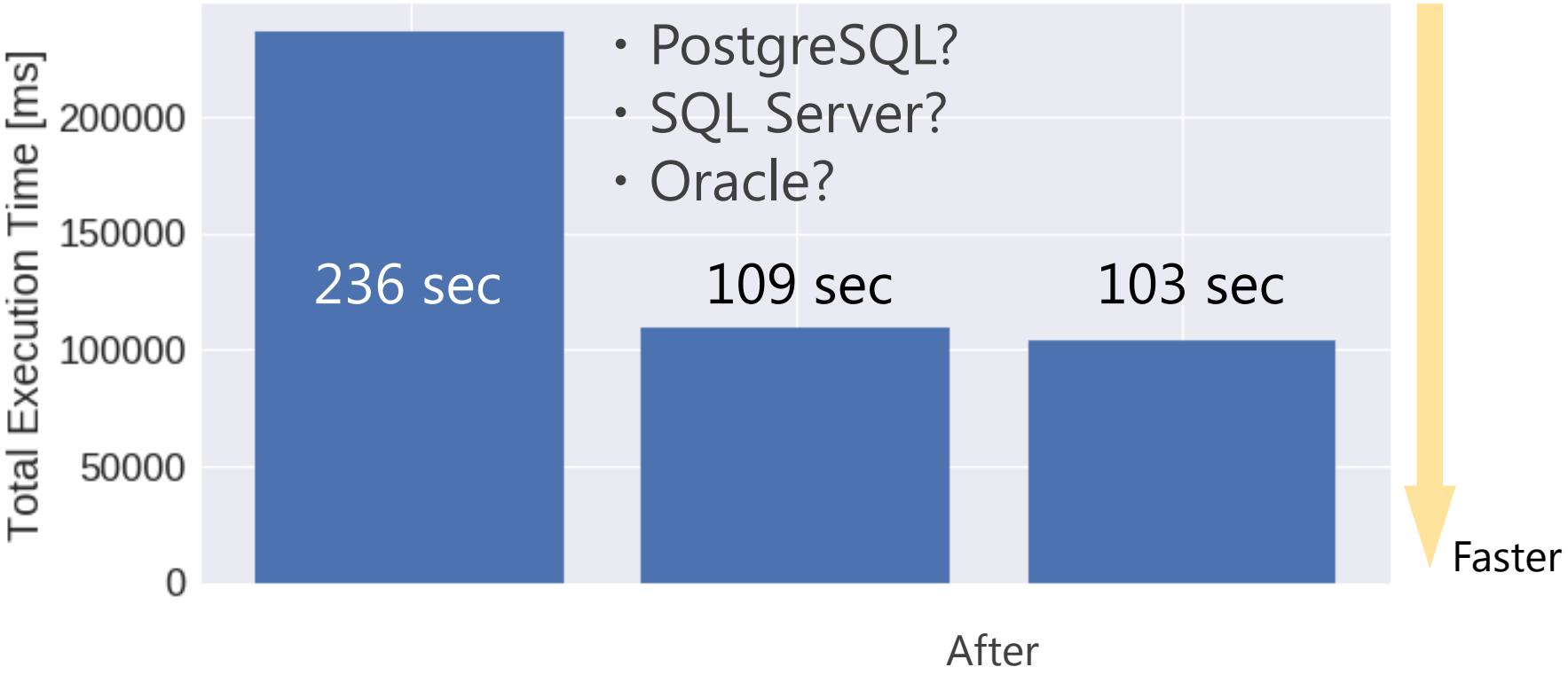
1. Background of plan tuning
2. Mechanism of pg_plan_advsr
3. Verification of effectiveness using Join order benchmark
4. Thoughts about the future PostgreSQL
5. Conclusion

Background of plan tuning

japan by www.travelbusy.com under CC BY 2.0

Question

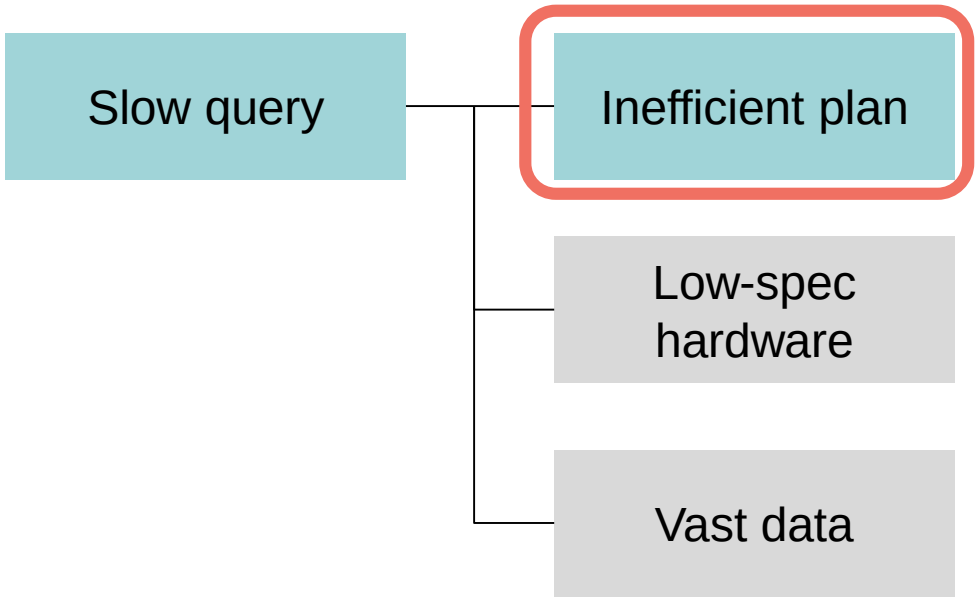
- **What are these graphs?**



These are before and after plan tuning in PostgreSQL

Why do we need plan tuning?

- **Because**



What does plan inefficiency mean?

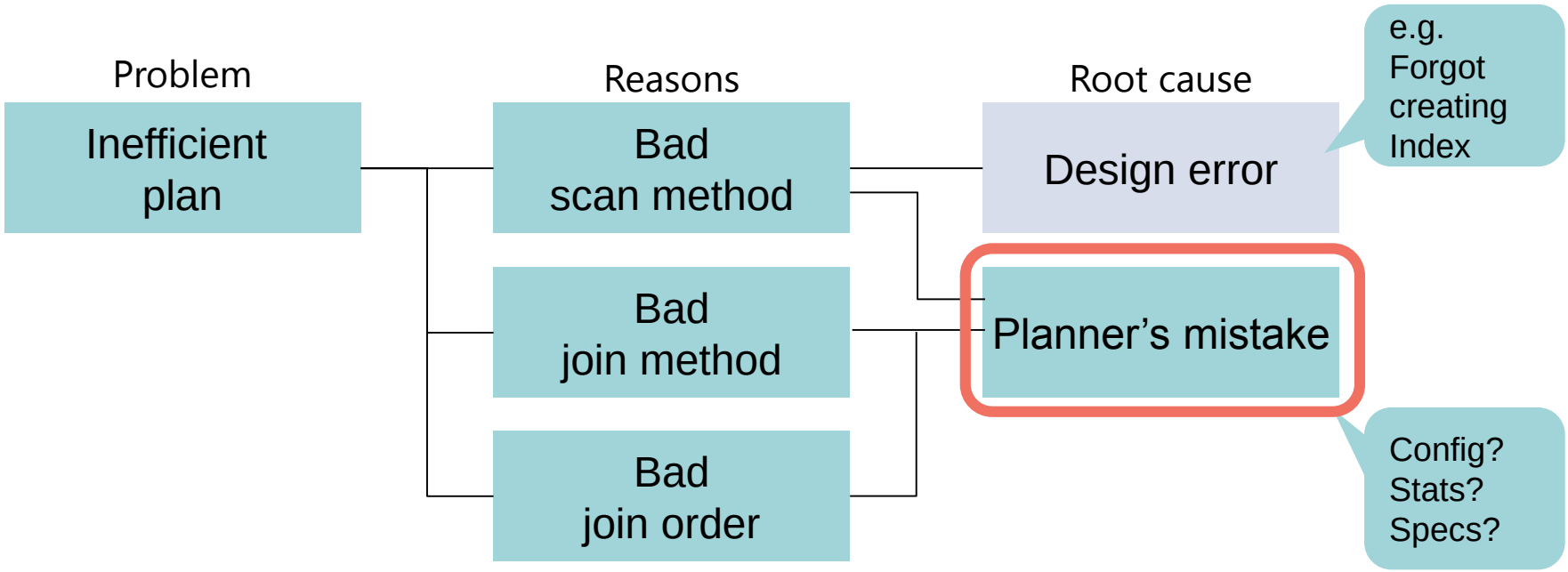
What is an inefficient plan?

- **An Inefficient plan is one that's not appropriate plan for the data.**

- As a result, following things could occur:
  - Long execution time
  - Resources being consuming wastefully
  - Processing not finished within the batch window

Where is the cause of an inefficient plan?

Where is the cause of an inefficient plan?



Planner takes mistakes sometimes.
What kind of mistakes?

Revisit Planner behavior

Innovative R&D by NTT

- The planner "guesses" a plan using cost base optimization.

- With a simple notion of "cost", each plan node's cost can be calculated by the following formula [1].

$$\mathrm{Cost} = c_{\mathrm{seq}} n_{\mathrm{seq}} + c_{\mathrm{rand}} n_{\mathrm{rand}} + c_{\mathrm{tup}} n_{\mathrm{tup}} + \cdots$$

Seq. I/O *Random I/O* *CPU cost per tuple*

$$= \mathbb{C} \cdot \mathbb{N}$$

$\mathbb{C}$ Cost of single operation

e.g.
- seq_page_cost
- random_page_cost

$\mathbb{N}$ Estimated number of each operation

- Cardinality estimation with statistics

- If "C" or "N" is wrong, the cost estimate is wrong. That may lead to an inefficient plan.

[1] cited from: "Beyond EXPLAIN: Query Optimization From Theory To Code" by YutoHayamizu, RyojiKawamichi PGCon 2016

Well known examples of planner mistakes

Innovative R&D by NTT

- **Cardinality estimation error**
  (aka Row count estimation error)
  - Over estimate
    - Estimated Rows = 5000 but Actual Rows = 1
  - Under estimate
    - Estimated Rows = 1 but Actual Rows = 50000

- The Ideally, there should be no cardinality estimation error.

- Such mistakes tend to select an inefficient plan.

DBA and user have to do various tunings.

See: https://blog.pgaddict.com/posts/common-issues-with-planner-statistics

NTT

Conventional plan tuning methods

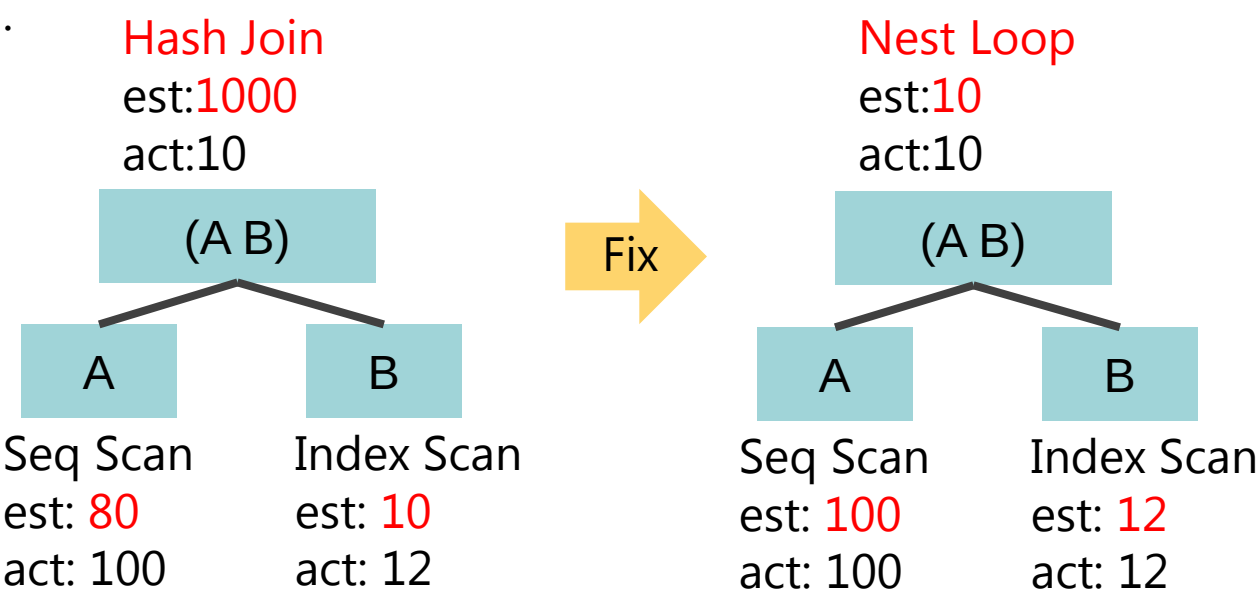
- **Improve accuracy of estimated rows**
  - Change the acquisition timing and sampling amount for Stats
  - Use extended statistics
  - Use pg_dbms_stats
- **Modify scan, join methods and join order**
  - Index tuning
  - Use GUC parameter
  - Rewrite queries
  - Use Optimizer Hint

EXPLAIN ANALYZE command for the tunings

Idea for getting more efficient plan

- **If we can correct cardinality estimations directly, we can be got more efficient plan?!**
  - Because, we know Actual rows by Explain Anayze command.

e.g.

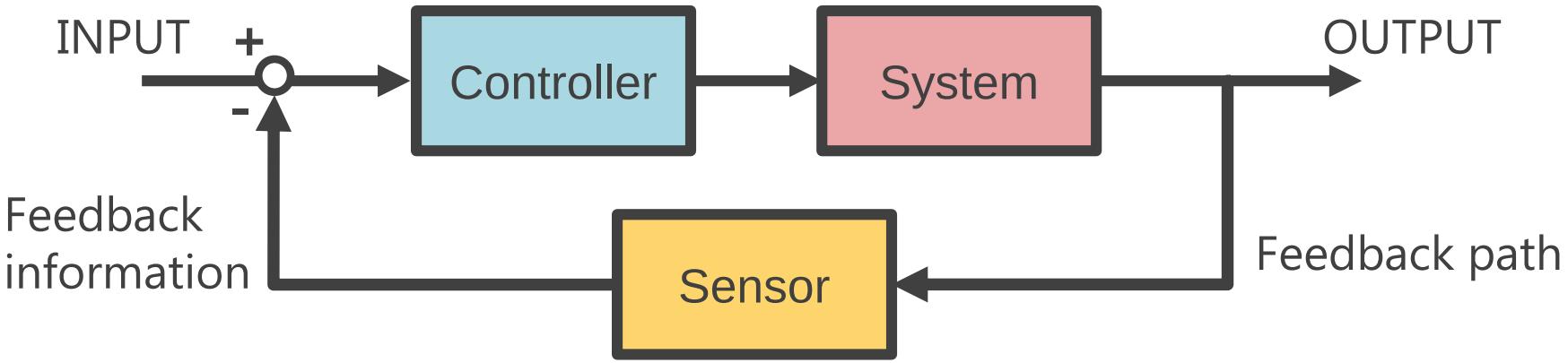


The idea is simple, but implementation is hard because there is no interface.

Mechanism of pg_plan_advsr

Control engineering: Feedback Loop

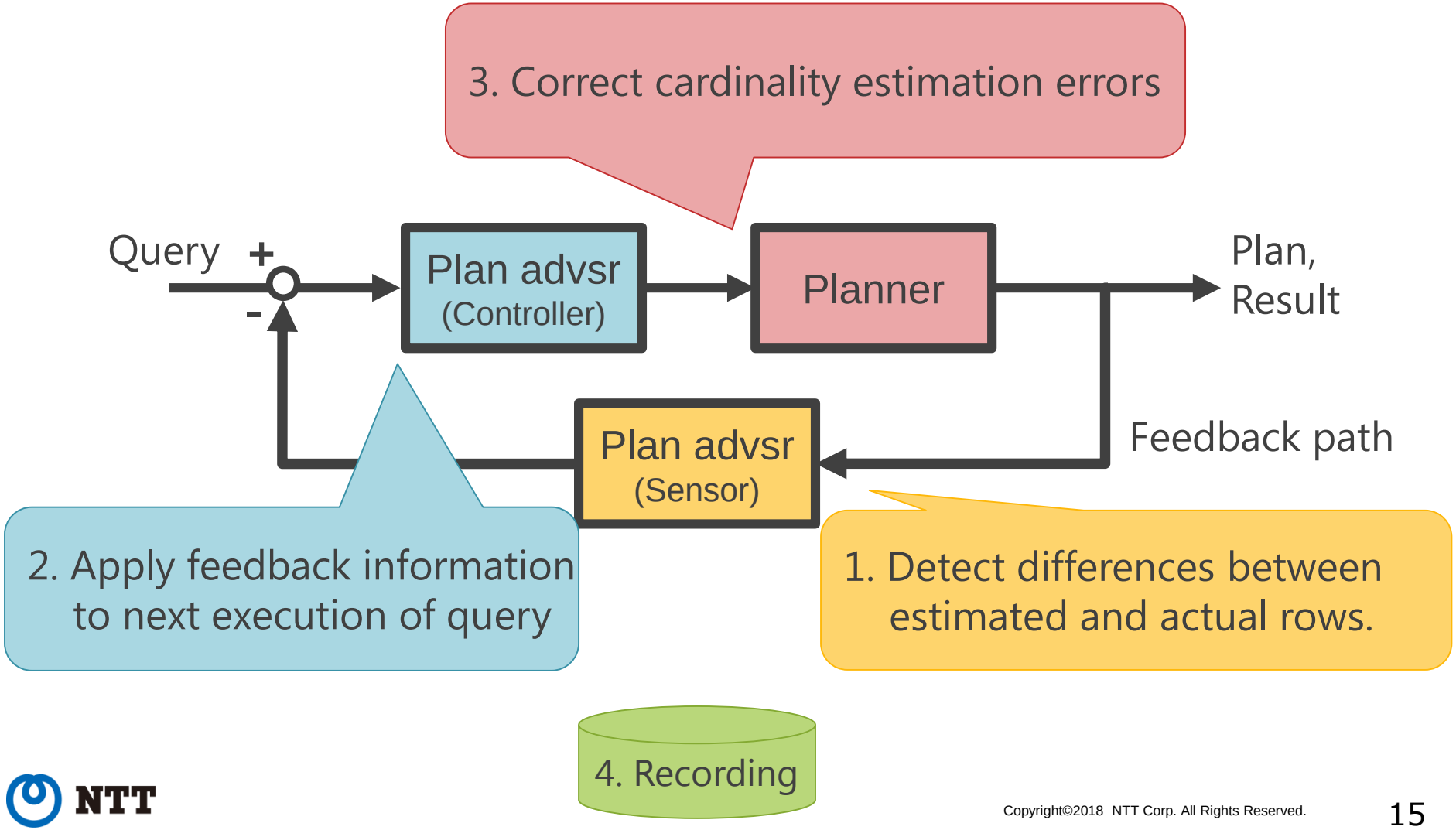
- **This is used to achieve the desired output.**
- **Theory**: Feedback is a mechanism that compares the output target value with the actual output value and automatically controls the output value and the target value to be equal.



How does this extension use feedback loop?

https://en.wikipedia.org/wiki/Control_theory
https://en.wikipedia.org/wiki/Control_engineering

Rough concept of pg_plan_advsr using feedback



Overview of pg_plan_advsr

Innovative R&D by NTT

- **The extension has four processes:**

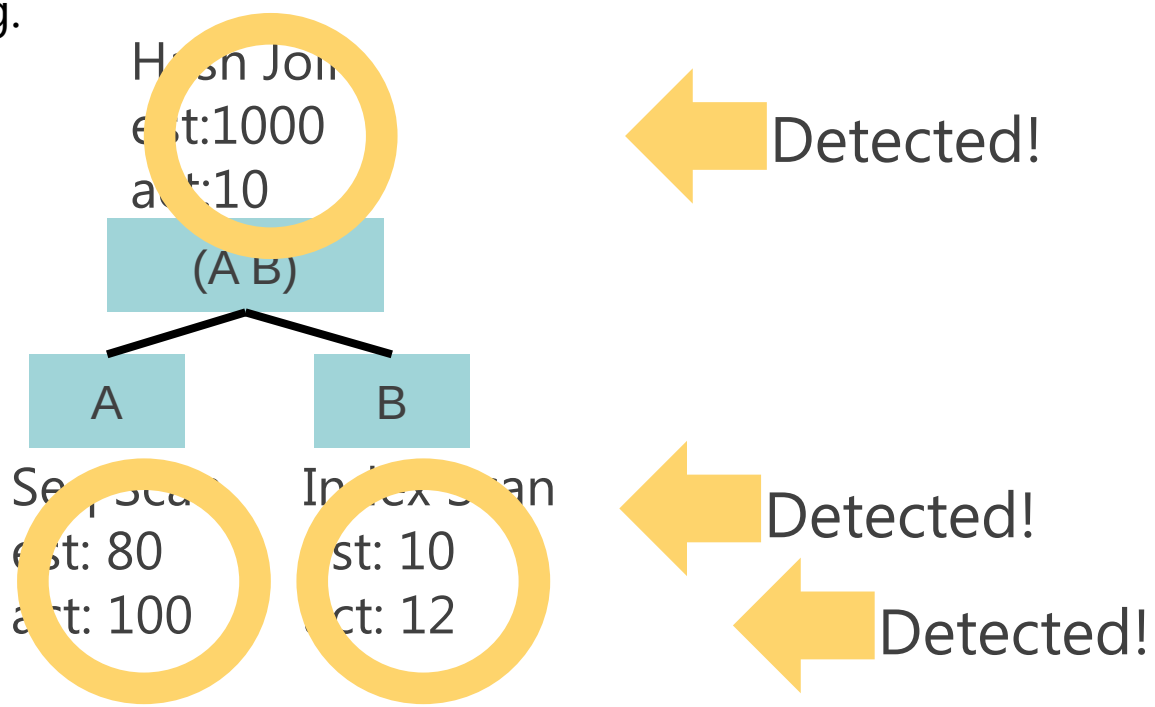
1. Detect differences between estimated and actual rows by inspecting plan tree
2. Apply feedback information to next execute of query
3. Correct cardinality estimation errors
4. Record Plan history

NTT

1. Detect differences by inspecting plan tree

- The extension searches in plan nodes recursively, and detect differences between estimated and actual rows.

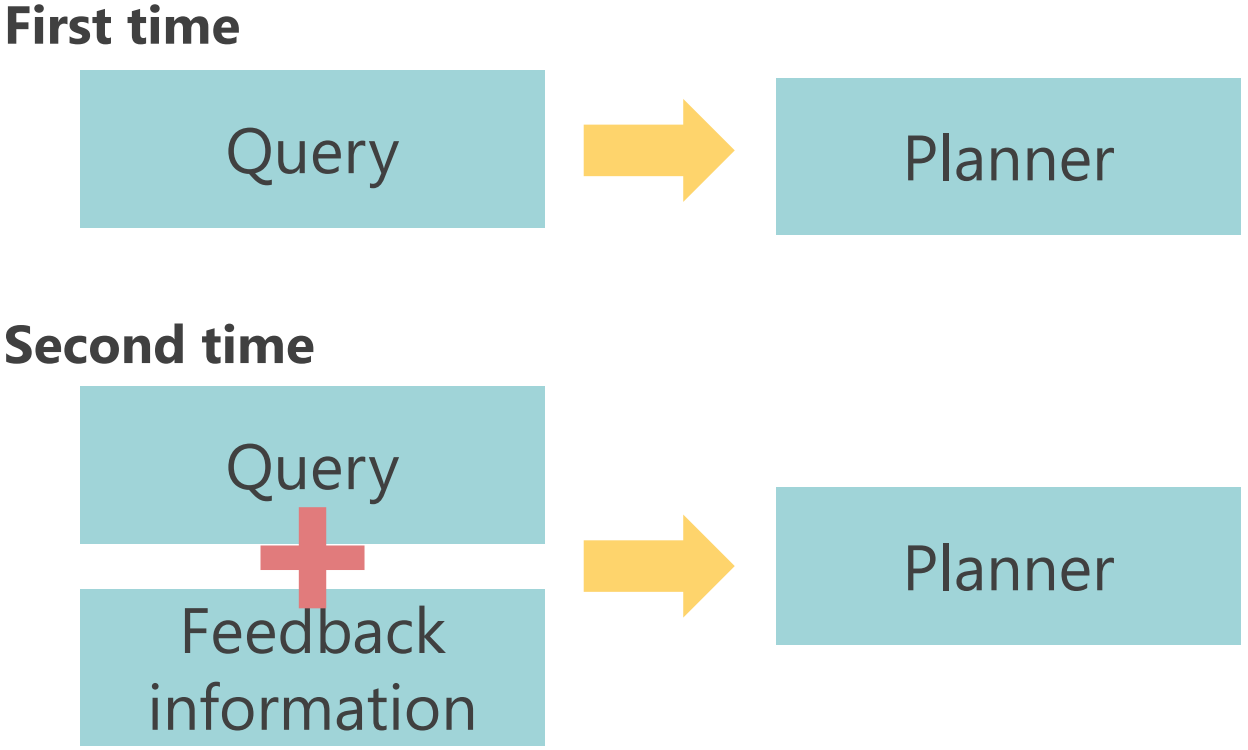
e.g.



- Feedback information is made in this phase.

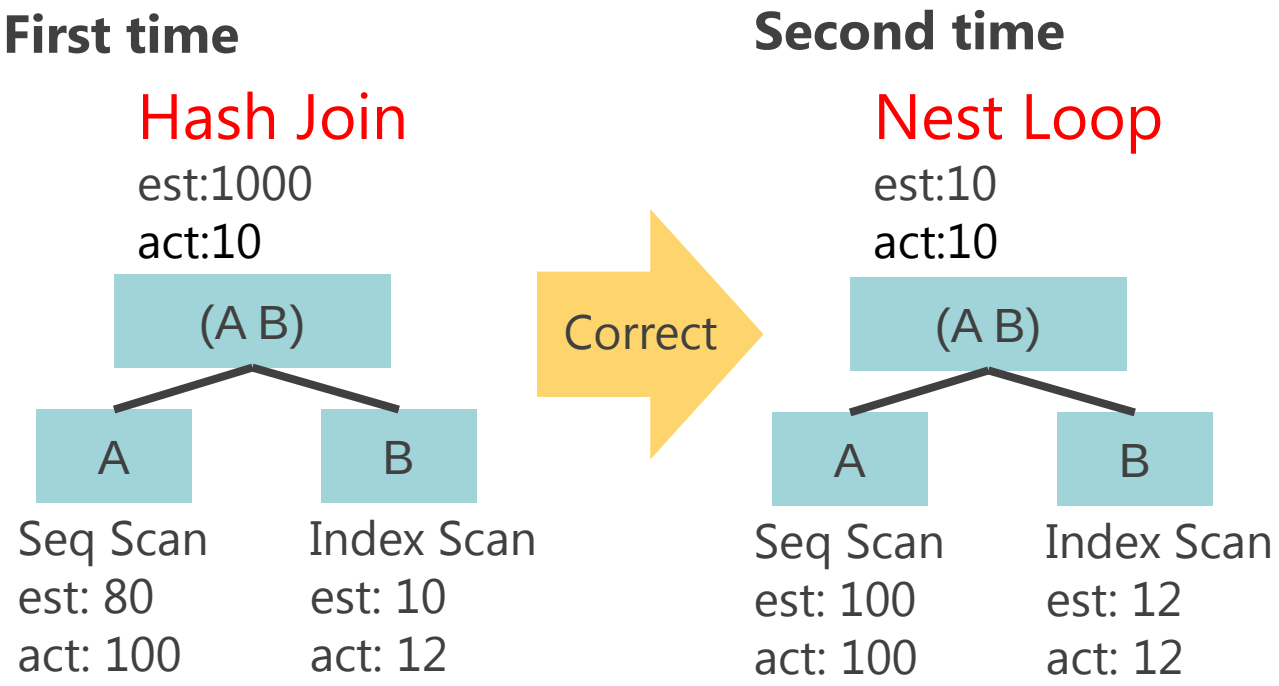
2. Apply feedback info to next query

- The extension applies feedback information to next execution of query to correct estimation error.



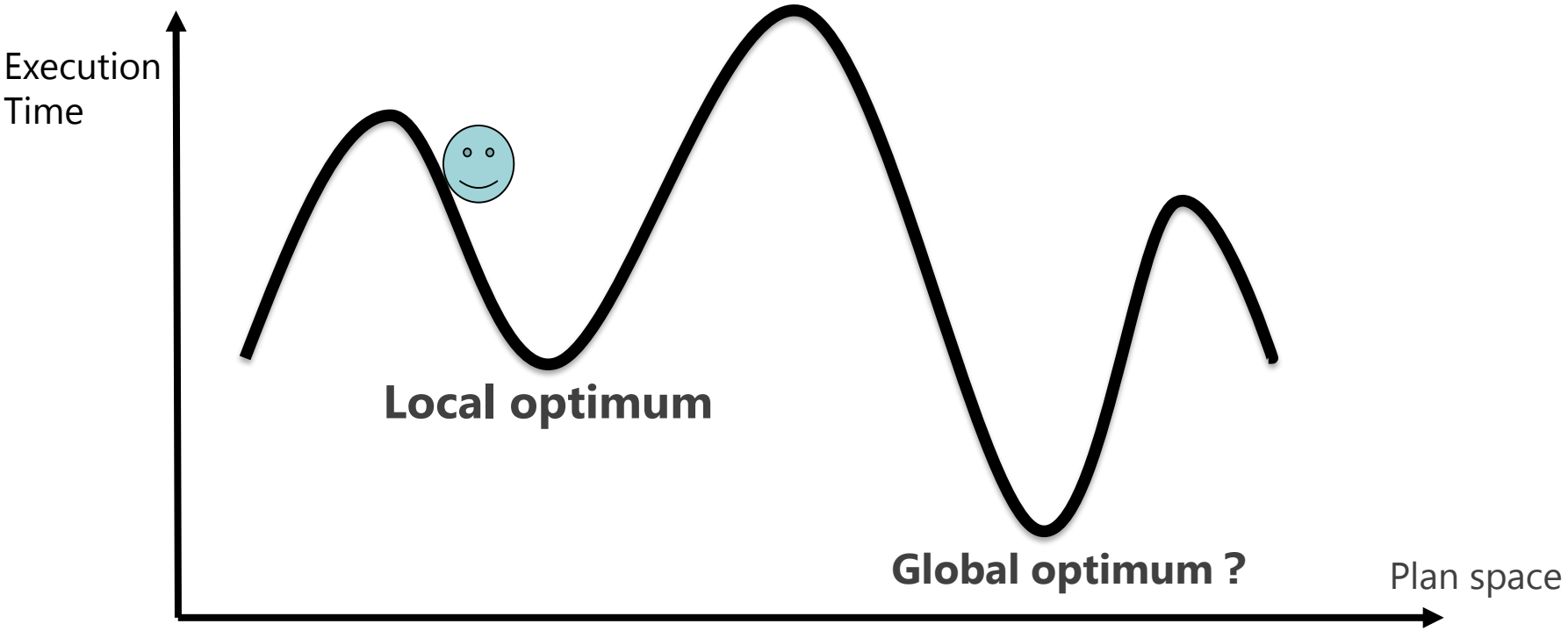
3. Correct cardinality estimation errors

- By using feedback information, planner will likely select an efficient plan because costs are accurate.



Local search

- pg_plan_advsr does local search to find an efficient plan.



- In the search process, the plan may temporarily get worse.

4. Record plan history

- There's a possibility that plan will not converge by feedback, and sometimes get worse during tuning.

- Therefore, the extension records plan history so as to allow investigation and also pick the most efficient plan from the history.

e.g. plan history table

Iterations	Query	Plan	Execution time
0	Query 1	PLAN A	100 ms
1	Query 1	PLAN B	200 ms
2	Query 1	PLAN C	80 ms
3	…	…	…

Implementation of pg_plan_advsr

1. Detecting differences to create feedback information
   - **ExecutorEnd_hook**
   - **New walker of queryDesc**

2. Applying feedback information to next execute of query
   and
3. Correcting cardinality estimation errors
   - **pg_hint_plan**'s Hint table feature

4. Record Plan history
   - **ExecutorEnd_hook**
   - **Create Plan_history table** to store all information
   - **pg_store_plans** is also used to store plan texts

When and How to use?

- **When?**
  - This extension is assumed to be used in the plan tuning process within system development.
- **How to use?**
  - Use explain analyze command until no estimation error.

Verification of effectiveness using Join order benchmark

Preconditions of measurement

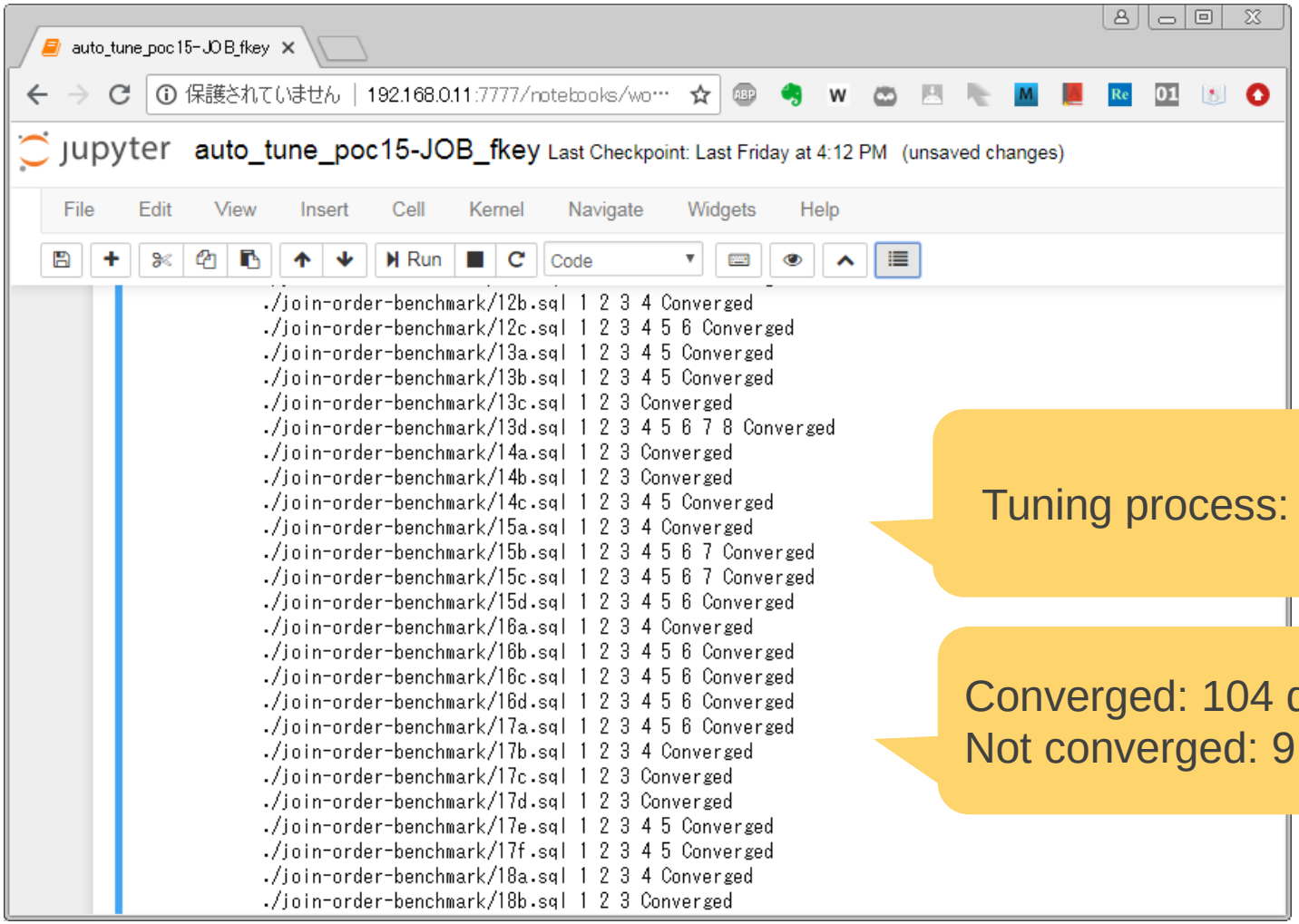
Innovative R&D by NTT

- **PG 10.4**
- **Data is stable and on memory** (using pg_prewarm)
- **Iterations for tuning:** Maximum 64 times per Query
  - (Iterations are completed when estimated rows equal actual rows)
- **Benchmark**: Join order benchmark (113 queries)
- **Parameters**
  - random_page_cost 2
  - shared_buffers 2GB
  - work_mem 16MB
  - default_statistics_target 100
  - geqo_threshold 18
  - max_worker_processes 8
  - max_parallel_workers_per_gather 0
  - max_parallel_workers 0
  - shared_preload_libraries = pg_hint_plan, pg_plan_advsr, pg_store_plans

NTT

My environment

- Jupyter notebook as a frontend

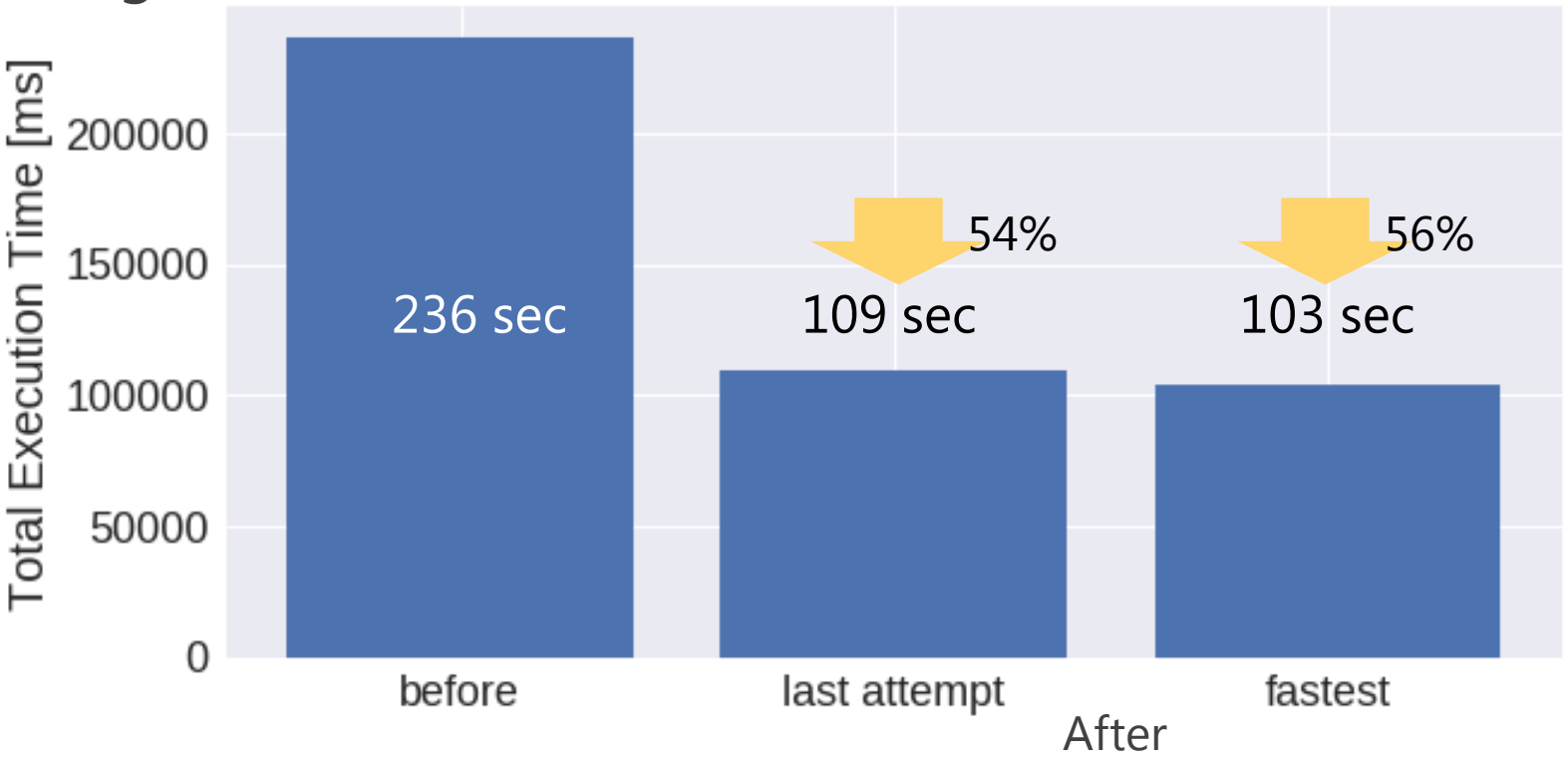


Tuning process: 3000 sec

Converged: 104 queries
Not converged: 9 queries

Results

- Query execution time is **reduced by 50%** of the original (236sec -> 103sec)



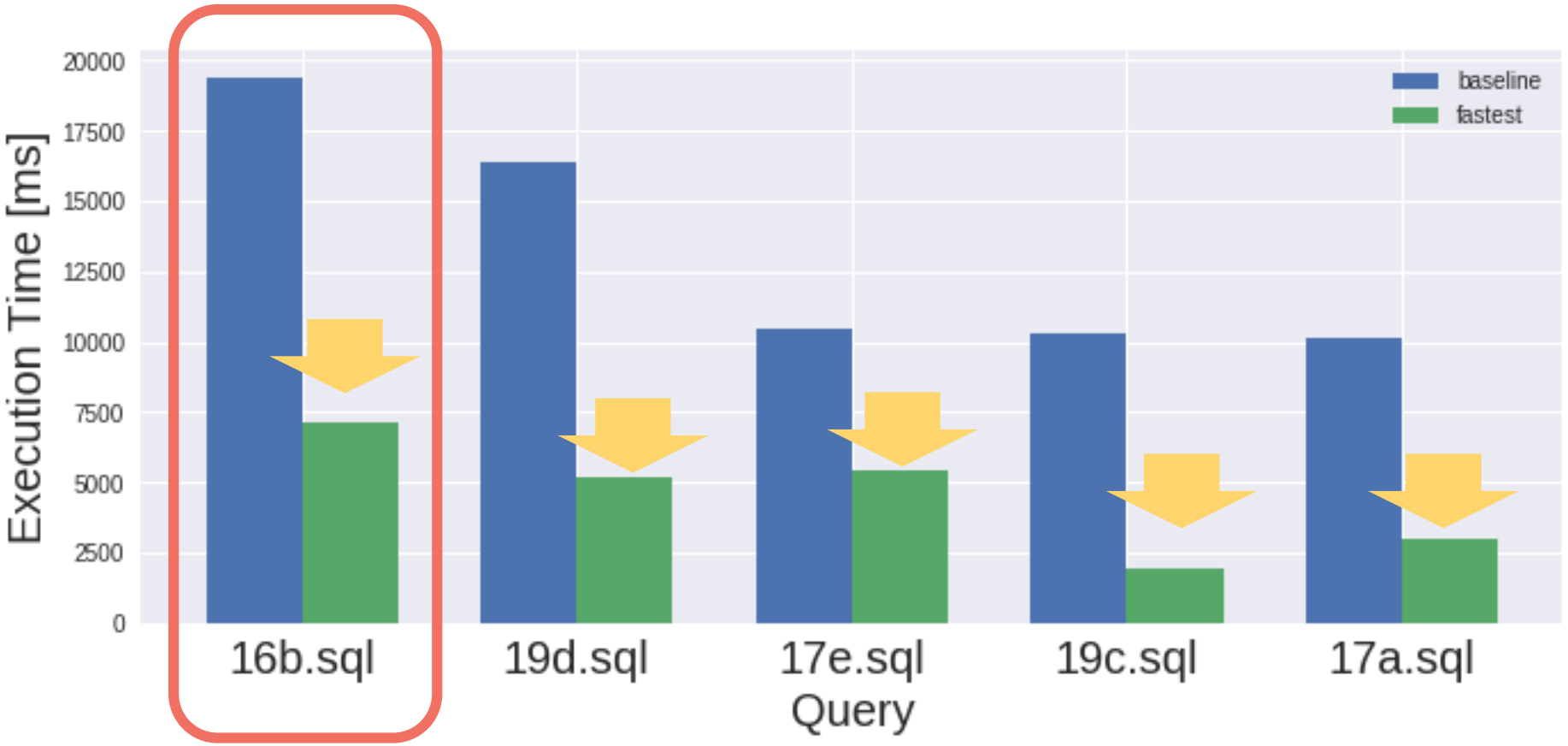
- **What's happened?**

Difference between baseline and fastest

- Almost all queries execution time are decreasing.
  Especially, middle part of the graph.

Top 5 long execution times

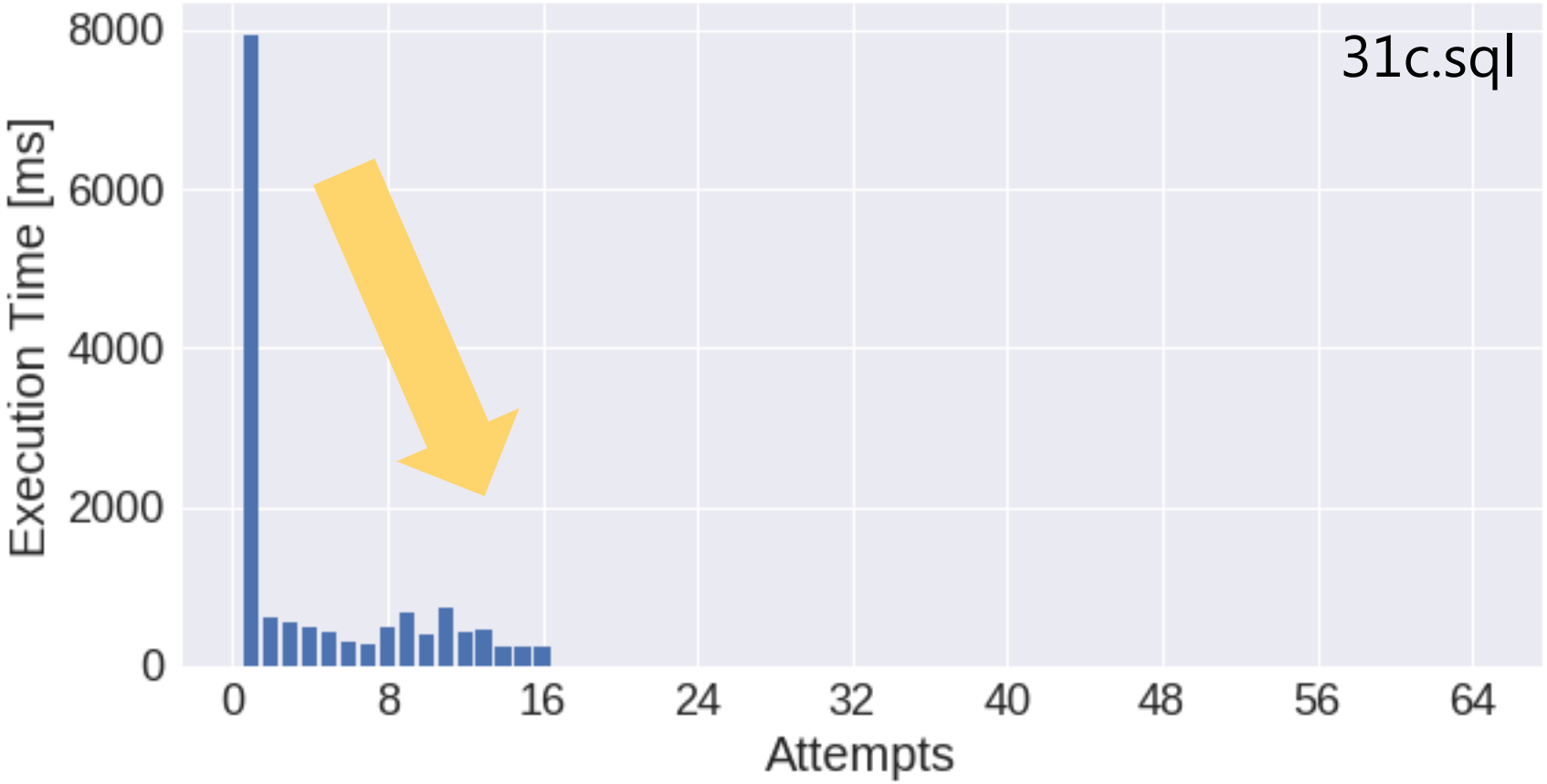
- **Top 5 long execution times were halved.**



63% down (19s -> 7s)

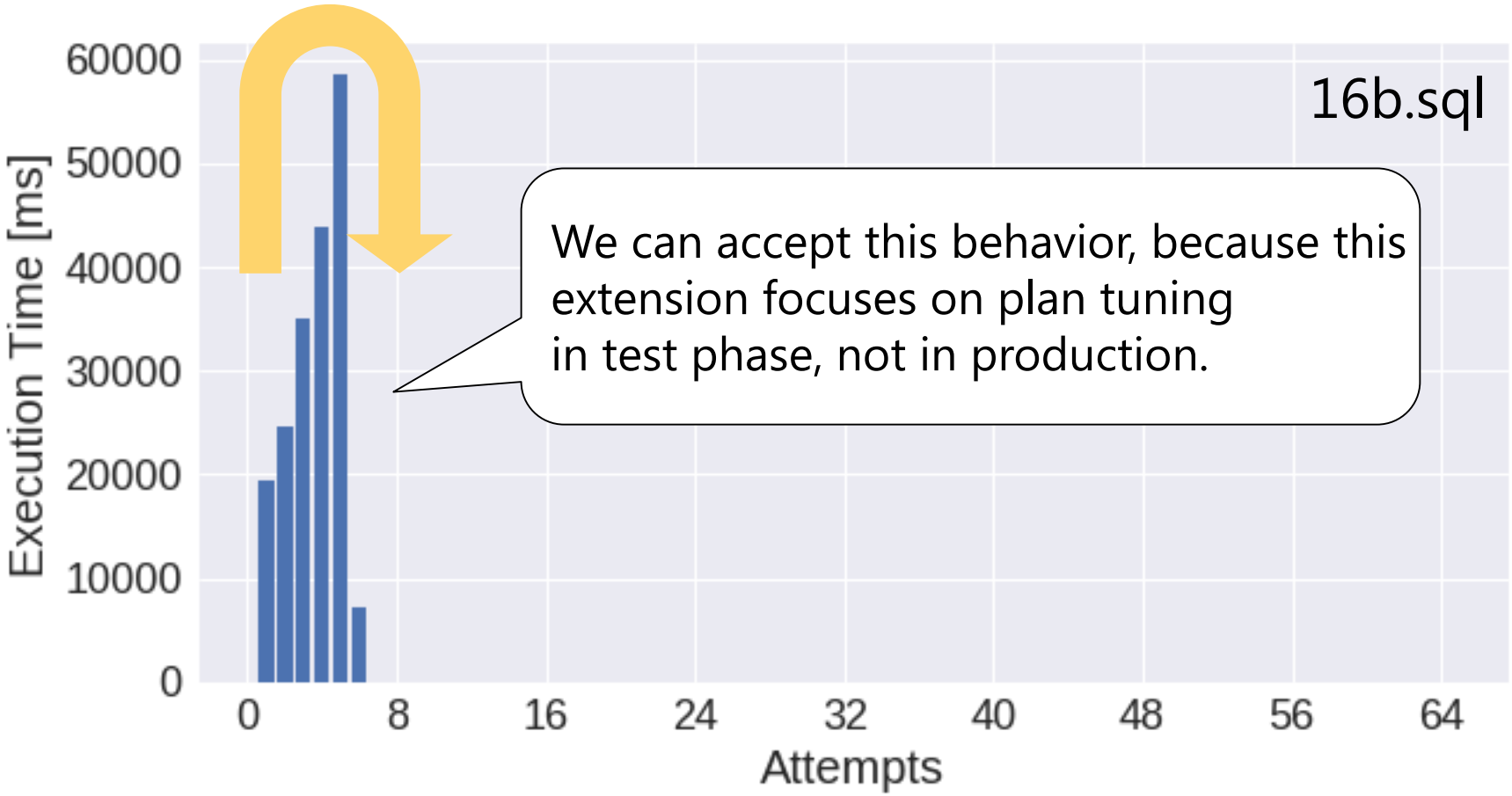
Expected behavior of execution times

- Ideally, it would be nice to get the following behavior for all queries



Execution Time history of query 16b (Q16b)

- However, I got a good plan after several bad plans.



An Example (Query 16b of join order benchmark)

Innovative R&D by NTT

- **Q16b includes 7 joins and 2 aggregate functions.**

```
SELECT MIN(an.name) AS cool_actor_pseudonym,
       MIN(t.title) AS series_named_after_char
FROM aka_name AS an,
     cast_info AS ci,
     company_name AS cn,
     keyword AS k,
     movie_companies AS mc,
     movie_keyword AS mk,
     name AS n,
     title AS t
WHERE cn.country_code ='[us]'
  AND k.keyword ='character-name-in-title'
  AND an.person_id = n.id
  AND n.id = ci.person_id
  AND ci.movie_id = t.id
  AND t.id = mk.movie_id
  AND mk.keyword_id = k.id
  AND t.id = mc.movie_id
  AND mc.company_id = cn.id
  AND an.person_id = ci.person_id
  AND ci.movie_id = mc.movie_id
  AND ci.movie_id = mk.movie_id
  AND mc.movie_id = mk.movie_id;
```

NTT

Plan history of Q16b

- The history table has all information of plan tuning.

(ms)

Times	Queryid	Plan_id	Execution_time	Total_diffs
0	…	**671501202**	**19396.89**	**9670384**
1	…	3725435884	24567.85	9504160
2	…	3151720077	35021.45	13242801
3	…	150735307	43750.84	17546662
4	…	1918733225	58548.67	23380179
5	…	**1368113010**	**7145.19**	**0**

Rows_hint	Scan_hint	Join_hint	Lead_hint
…	…	…	…

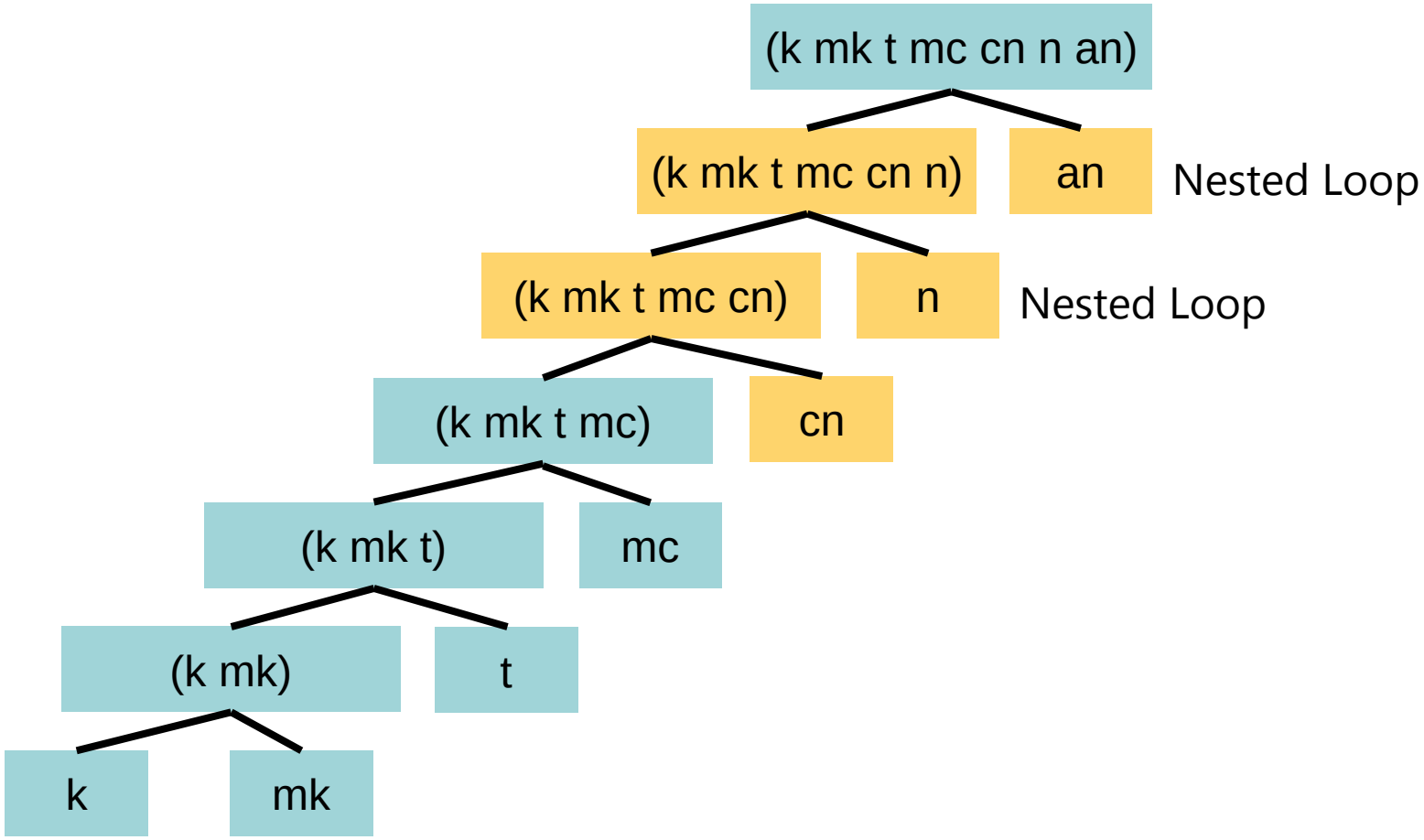
- You can see the Plan id had changed until the last time.

Feedback control has worked!

"Total_diffs" is the sum of row count estimation errors in the joins

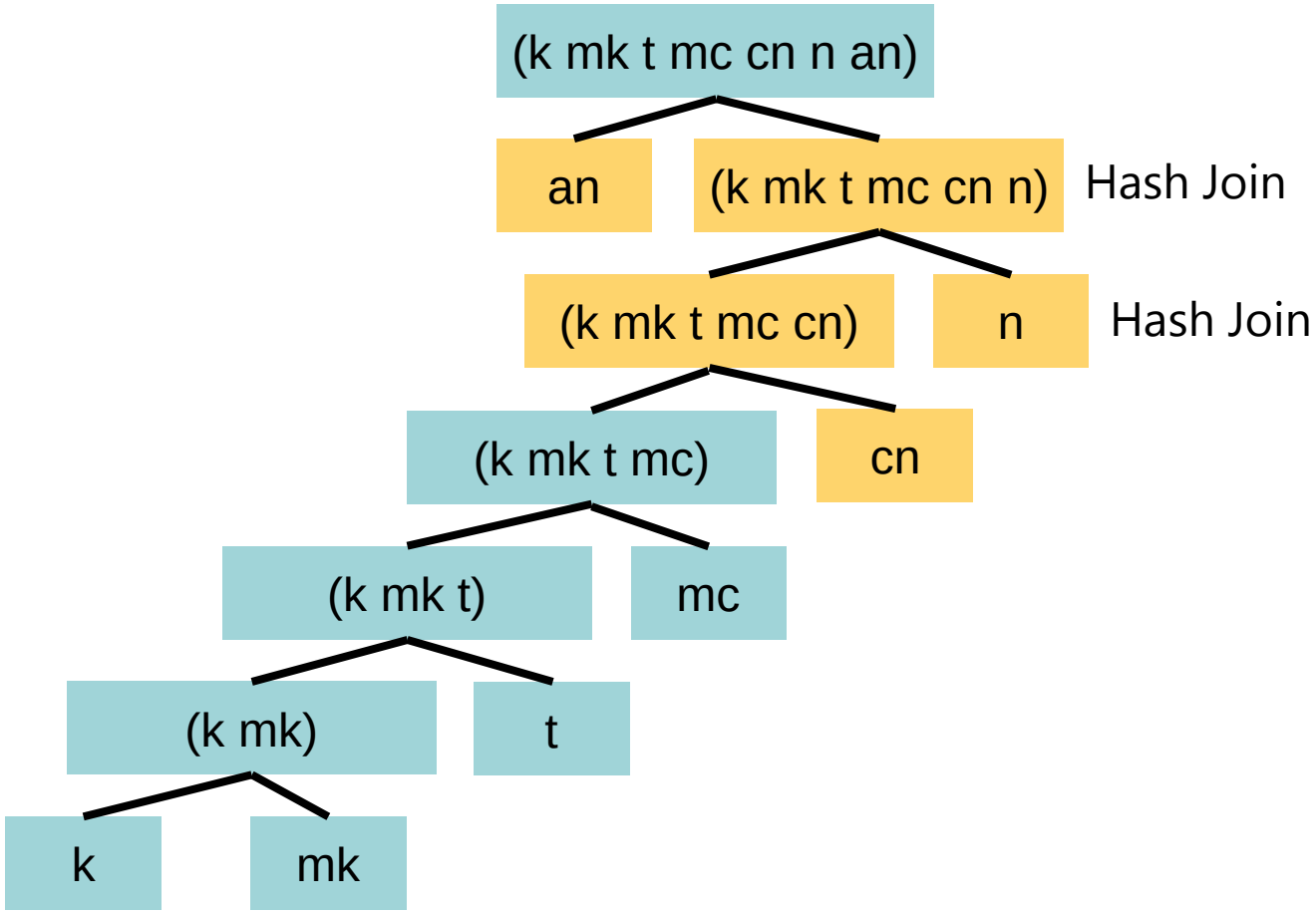
Plan shapes of Q16b

- **Before**



Plan shapes of Q16b

- **After**



Plan shapes of Q16b

Innovative R&D by NTT

Before

```
Aggregate  (cost=4645.64..4645.65 rows=1 width=64) (actual time=20608.515..20608.515 rows=1 loops=1)
  ->  Nested Loop  (cost=7.51..4630.64 rows=2999 width=33) (actual time=5.001..19980.694 rows=3710592 loops=1)
        Join Filter: (n.id = an.person_id)
        ->  Nested Loop  (cost=7.09..4021.98 rows=1253 width=25) (actual time=4.989..12126.936 rows=2832555 loops=1)
              ->  Nested Loop  (cost=6.66..3456.63 rows=1253 width=21) (actual time=4.974..4445.613 rows=2832555 loops=1)
                    Join Filter: (t.id = ci.movie_id)
                    ->  Nested Loop  (cost=6.09..3333.83 rows=66 width=29) (actual time=4.768..886.401 rows=68316 loops=1)
                          ->  Nested Loop  (cost=5.67..3251.47 rows=186 width=33) (actual time=4.738..431.818 rows=148552 loops=1)
                                ->  Parallel Nested Loop  (cost=5.24..3231.47 rows=34 width=25) (actual time=4.724..225.903 rows=41840 loops=1)
                                      ->  Nested Loop  (cost=4.81..3215.61 rows=34 width=4) (actual time=4.714..65.081 rows=41840 loops=1)
                                            ->  Seq Scan on keyword k  (cost=0.00..2626.12 rows=1 width=4) (actual time=0.445..10.156 rows=1 loops=1)
                                                  Filter: (keyword = 'character-name-in-title'::text)
                                                  Rows Removed by Filter: 134169
                                            ->  Bitmap Heap Scan on movie_keyword mk  (cost=4.81..586.42 rows=307 width=8) (actual time=4.267..47.872 rows=41840 loops=1)
                                                  Recheck Cond: (keyword_id = k.id)
                                                  Heap Blocks: exact=11541
                                                  ->  Bitmap Index Scan using keyword_id_movie_keyword  (cost=0.00..4.74 rows=307 width=0) (actual time=2.816..2.816 rows=41840 loops=1)
                                                        Index Cond: (keyword_id = k.id)
```

After

```
Aggregate  (cost=569619.76..569619.77 rows=1 width=64) (actual time=7724.114..7724.114 rows=1 loops=1)
  ->  Hash Join  (cost=488969.61..551066.80 rows=3710592 width=33) (actual time=6148.042..7284.667 rows=3710592 loops=1)
        Hash Cond: (an.person_id = n.id)
        ->  Seq Scan on aka_name an  (cost=0.00..20409.43 rows=901343 width=20) (actual time=0.006..125.215 rows=901343 loops=1)
        ->  Hash  (cost=434198.68..434198.68 rows=2832555 width=25) (actual time=6147.564..6147.564 rows=2832555 loops=1)
              Buckets: 262144  Batches: 16  Memory Usage: 12256kB
              ->  Hash Join  (cost=172645.20..434198.68 rows=2832555 width=25) (actual time=1244.367..5588.425 rows=2832555 loops=1)
                    Hash Cond: (ci.person_id = n.id)
                    ->  Nested Loop  (cost=6986.59..183304.07 rows=2832555 width=21) (actual time=57.009..2836.849 rows=2832555 loops=1)
                          Join Filter: (t.id = ci.movie_id)
                          ->  Hash Join  (cost=6986.02..56192.08 rows=68316 width=29) (actual time=56.826..536.766 rows=68316 loops=1)
                                Hash Cond: (mc.company_id = cn.id)
                                ->  Nested Loop  (cost=5.67..47336.27 rows=148552 width=33) (actual time=5.417..443.411 rows=148552 loops=1)
                                      ->  Nested Loop  (cost=5.24..22733.15 rows=41840 width=25) (actual time=5.382..251.833 rows=41840 loops=1)
                                            ->  Nested Loop  (cost=4.81..3215.61 rows=41840 width=4) (actual time=5.366..70.316 rows=41840 loops=1)
                                                  ->  Seq Scan on keyword k  (cost=0.00..2626.12 rows=1 width=4) (actual time=0.556..10.051 rows=1 loops=1)
                                                        Filter: (keyword = 'character-name-in-title'::text)
                                                        Rows Removed by Filter: 134169
```

-> Nested Loop (rows= 2999) (actual rows=3710592)
  -> Nested Loop (rows=1253) (actual rows=2832555)
  -> Index Scan using ... an (rows=2) (actual rows=1)

-> Hash Join (rows=3710592) (actual rows=3710592)
  -> Seq Scan on ... an (rows=901343) (actual rows=901343)
  -> Hash (rows=2832555) (actual rows=2832555)

Fixed

- By correcting estimated rows on lower nodes, join method and join order on the top node got fixed automatically.

Plan history of Q16b

Innovative R&D by NTT

- Various hints are stored in the history table.

Times	Queryid	Plan_id	Execution_time	Total_diffs
0	…	671501202	19396.89	9670384
1	…	3725435884	24567.85	9504160
2	…	3151720077	35021.45	13242801
3	…	150735307	43750.84	17546662
4	…	1918733225	58548.67	23380179
5	…	1368113010	7145.19	0

Rows_hint	Scan_hint	Join_hint	Lead_hint
…	…	…	…

NTT

Brief explanation of pg_hint_plan's hints

- **Samples**: scan method, join method, join order, and row correction hint.

Hint	Effect
INDEXSCAN(A) …	Forces "index scan" on table "A"
NESTLOOP(A B) …	Forces "nested loop" to the join consists of the table "A" and "B".
LEADING((A B) C)	Forces "join order" as specified. Firstly A join B, after that (A B) join C.
ROWS(A B #10)	Corrects row number of a result of the join consists of table "A" and "B".

A, B and C are Table or Alias

Scan method

Join method

Join order

Row correction

See: https://github.com/ossc-db/pg_hint_plan/blob/master/doc/hin

Rows hint of Q16b

Innovative R&D by NTT

- Rows Hints allow to override estimated rows on joins.

Times	Plan id	Rows_hint	Scan_hint	Join_hint	Lead_hint
…	…	…	…	…	…

Rows hint of Q16b

- We can check feedback information to check ROWS hints.

Times	Rows_hint	Total_diffs
0	ROWS(an ci cn k mc mk n t #3710592) …	9670384
1	…	9504160
2	…	13242801
3	…	17546662
4	ROWS(ci k mc mk n t #7796926) …	23380179
5	**Nothing!!**	0

Scan, Join and Leading hints of Q16b

Innovative R&D by NTT

- pg_plan_advsr also generated and stored these hints.

Times	Plan id	Rows_hint	Scan_hint	Join_hint	Lead_hint
…	…	…	…	…	…

- **Why these hints are stored?**
- **Because**
  - These **hints can expresses a plan structure**.
  - By using these hints, **you can reproduce the plan at a certain point**, anytime.

NTT

Scan hints of Q16b

Scan_hint	Join_hint	Lead_hint
…	…	…

Innovative R&D by NTT

- Several Index scans replaced with Seq scans.

times	Scan_hint
0	INDEXSCAN(cn) INDEXONLYSCAN(n) INDEXSCAN(an) …
1	…
2	…
3	…
4	SEQSCAN(cn) SEQSCAN(n) SEQSCAN(an) …

NTT

Join hints of Q16b

Scan_hint	Join_hint	Lead_hint
…	…	…

Innovative R&D by NTT

- Some Join methods were changed.

times	Join_hint
0	NESTLOOP(an ci cn k mc mk n t) NESTLOOP(ci cn k mc mk n t) …
1	…
2	…
3	…
4	…
5	HASHJOIN(an ci cn k mc mk n t) HASHJOIN(ci cn k mc mk n t) …

Leading hints of Q16b

Scan_hint	Join_hint	Lead_hint
…	…	…

Innovative R&D by NTT

- We can see table "an" moved to left-most side.

times	Lead_hint
0	LEADING(((((((k mk)t)mc)cn)ci)n)an))
1	…
2	…
3	…
4	…
5	LEADING((an ((((((k mk)t)mc)cn)ci)n)))

We can understand plan changes easily by using Scan, Join and Leading hints.

NTT

Reverification of plan tuning effect

- I rechecked the plan tuning effect by using baseline and fastest plans.
  - **Operations**
    1. Add fastest hints (Scan, Join method and Join order) to fastest.sql.
    2. Shutdown PG and clear OS cache
    3. Prewarm whole tables
    4. Run "psql –f baseline.sql and fastest.sql" three times.
  - **Result**

(ms)

Types	first	second	third
baseline.sql	252.93	233.57	233.87
fastest.sql	138.58	118.40	118.38

Reduced 48% on average, Good!!

Limitations and Future Work

- **Limitations**
  - Only correct join estimates
    - This is a limitation of pg_hint_plan
    - If the baserel can also be corrected, the convergence becomes faster
  - Initplan and subplan are not supported
  - Does not support concurrent execution

  …

- **Future work**
  - Remove above limitations
  - Improve correcting error rows mechanism to reduce iterations

Status of pg_plan_advsr

- POC phase
- Will share the extention on Github in this year.

NTT

Summary

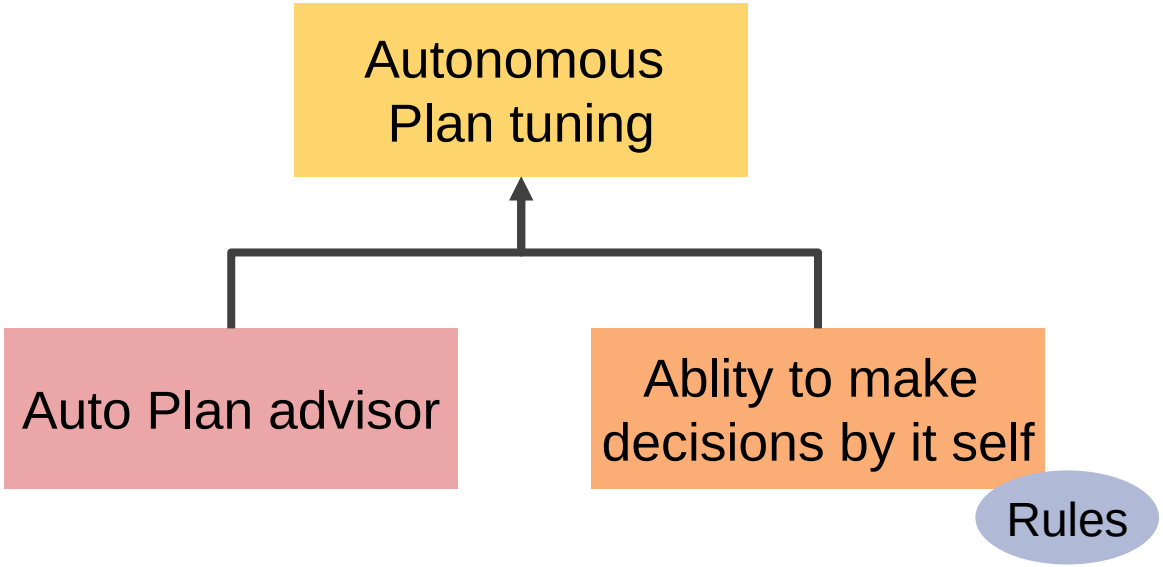
- pg_plan_advsr was able to improve the plan by reducing row count estimation error.
- Tuning process may temporarily result in plan worse than the initial plan, but this is tuning after all.
- In the measurement result, pg_plan_advsr was able to reduce about 50% in execution time.
- It is possible to utilize execution information for auto plan tuning.

Thoughts about the future PostgreSQL

Autonomous Database

- **Autonomous Features**
  - Upgrade
  - Tuning
  - Maintenance task

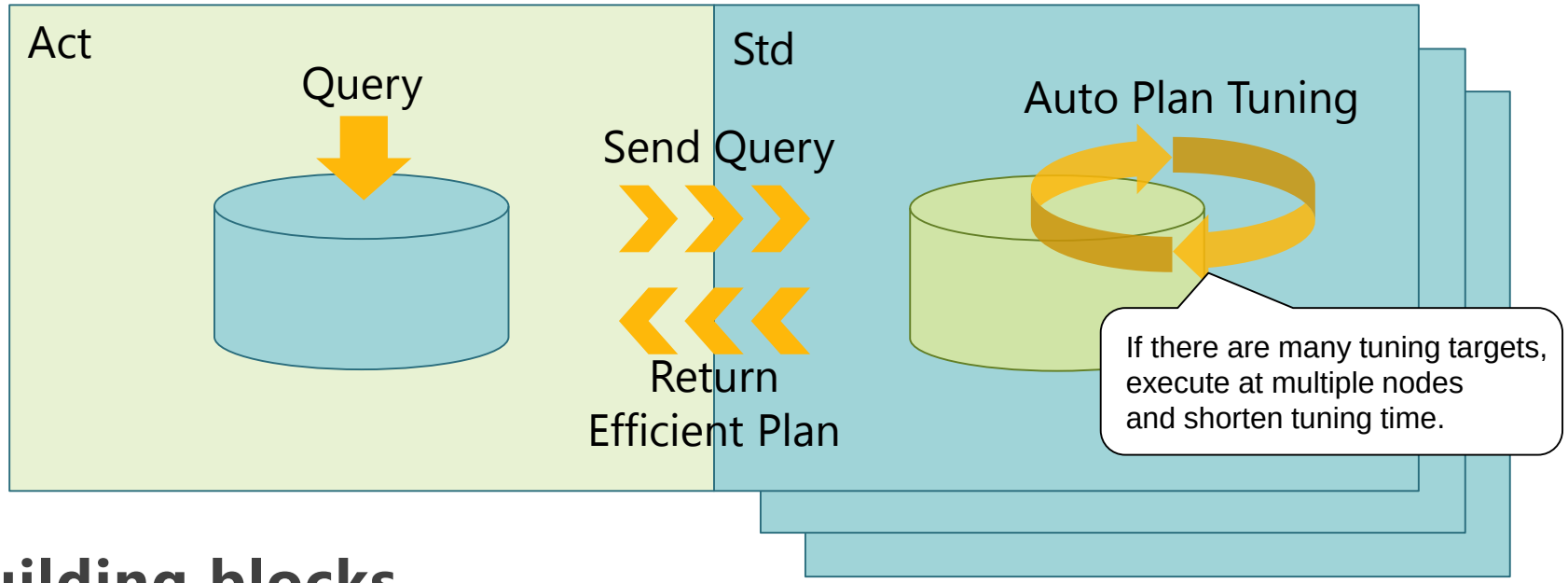
- **Idea**



Ideas for autonomous databases using replicationed PostgreSQL

Innovative R&D by NTT

- **Rough concept**



- **Building blocks**
  - Provide easy access to plan tree or explain analyze result: New hook
  - Interfaces to adjust estimated rows on base/join relations: New API
  - Avoid query cancellation on standby side: Improvement
  - Store all plans like a pg_stat_statement: Improvement

NTT

Conclusion

- In this talk, I have shared my experience of trying to get more efficient plans for complex queries using my POC extension, and also my thoughts about future PostgreSQL.

- I hope that I was able to prove that PostgreSQL can be improved to get more efficient plan using feedback loop.

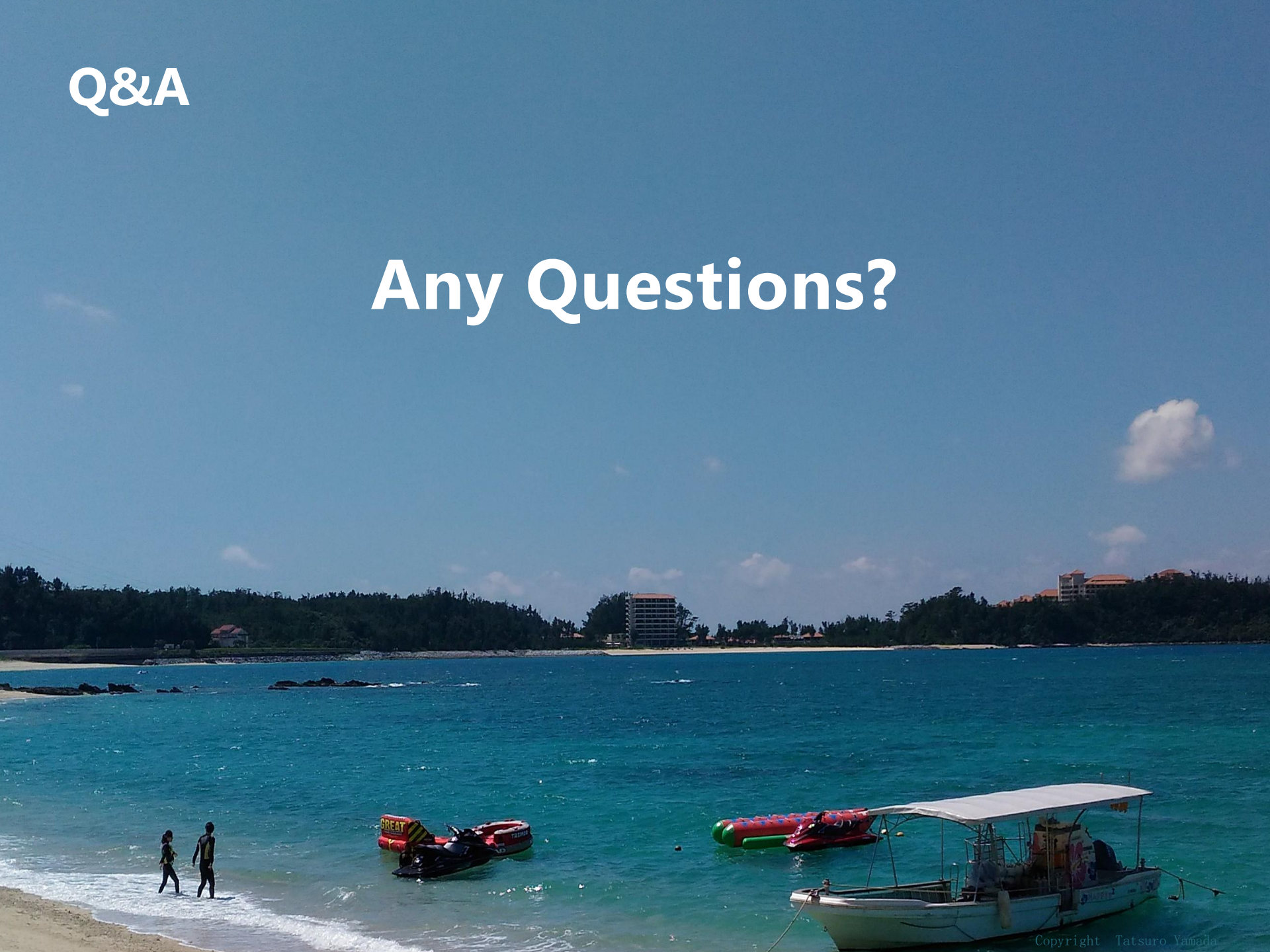
- I believe the improvement is a key challenge to reach future PostgreSQL.

Thank you!

Plan for better Plan

yamada.tatsuro@lab.ntt.co.jp
yamatattsu@gmail.com

Q&A



Appendix

- Join order benchmark
- Other examples of verification effect
- Extensions from NTT OSS Center
- References

Join order benchmark

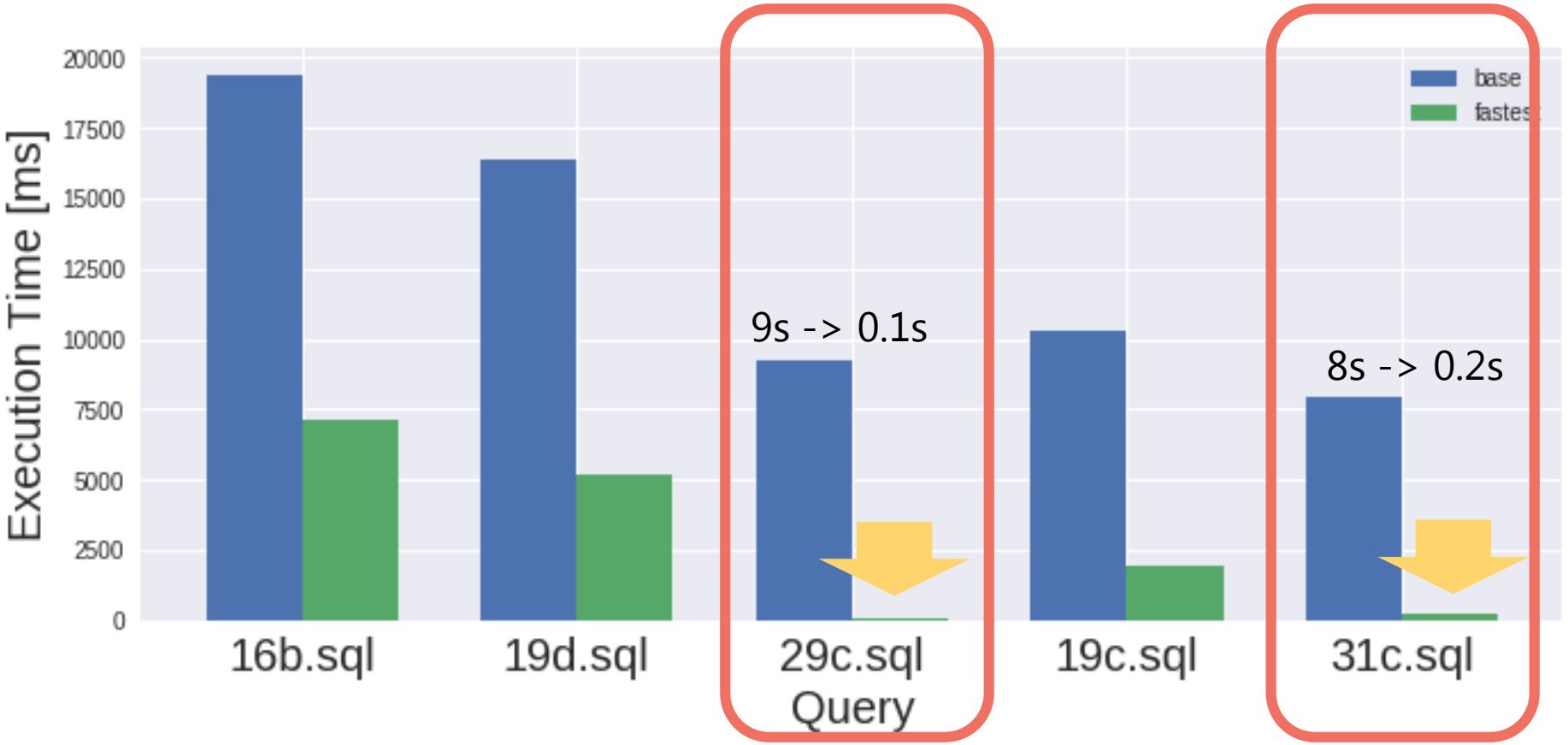
- **GitHub**
  - https://github.com/gregrahn/join-order-benchmark

<Examples>

Top 5 highly effective query

Innovative R&D by NTT

- **Query 29c and 31c ???**



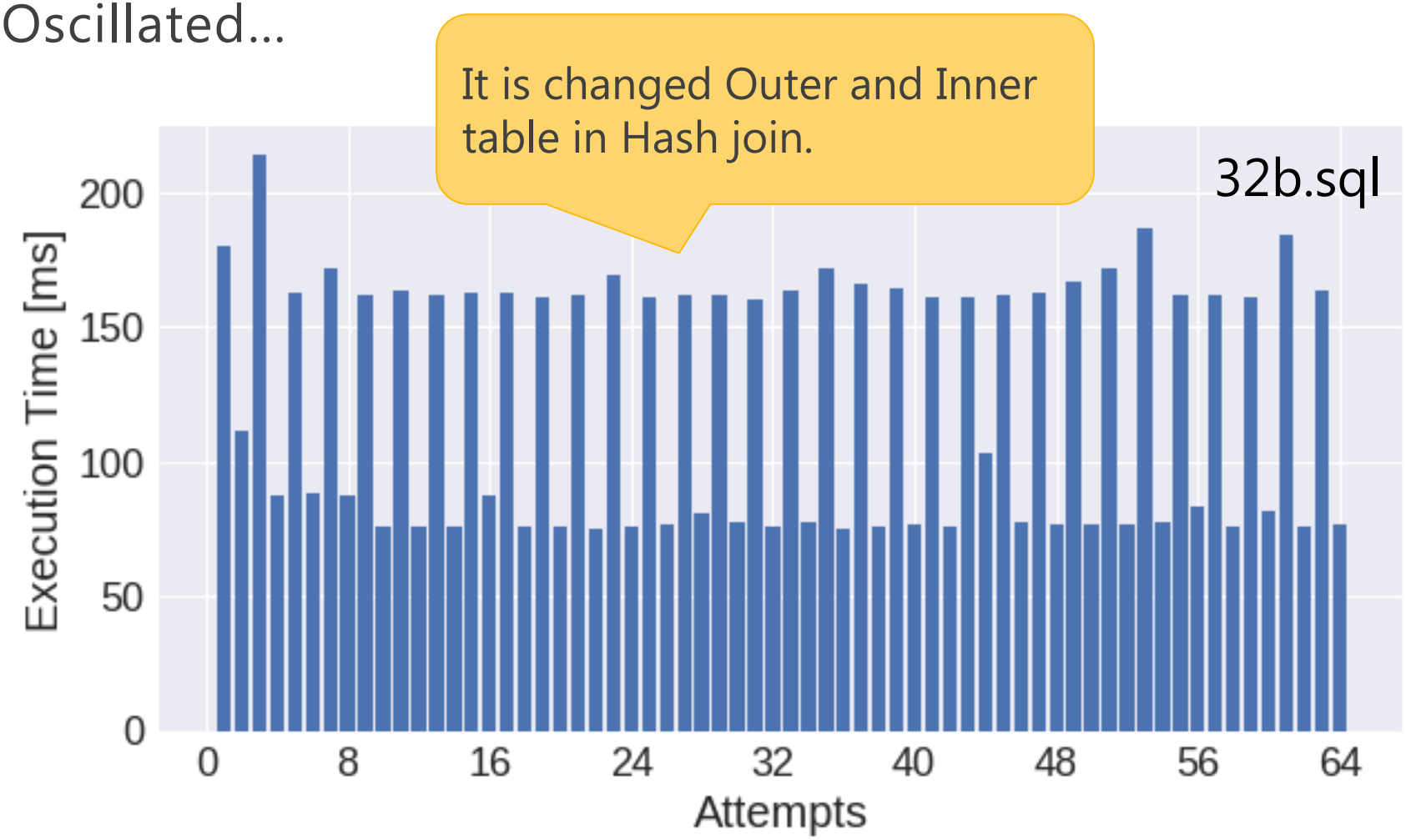
NTT

<Examples>

Sample: Not converged query (Q32b)

Innovative R&D by NTT

- Oscillated…



Extensions from NTT OSS Center

- **GitHub**
  - https://github.com/ossc-db
    - pg_hint_plan
    - pg_store_plans
    - pg_dbms_stats
    - pg_reorg
    - pg_rman
    - pg_bulkload
    - dblink_plus
    - Syncdb
    - db_syntax_diff
- **SourceForge**
  - https://sourceforge.net/projects/pgstatsinfo/
    - pg_statsinfo
    - pg_stats_reporter

References

- **"How Good Are Query Optimizers, Really?"**

  by Viktor Leis, Andrey Gubichev, Atans Mirchev, Peter Boncz, Alfons Kemper, Thomas Neumann
  PVLDB Volume 9, No. 3, 2015
  http://www.vldb.org/pvldb/vol9/p204-leis.pdf

- **"Query optimization through the looking glass, and what we found running the Join Order Benchmark"**

  by Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, Thomas Neumann

  https://db.in.tum.de/~leis/papers/lookingglass.pdf

- **My session at PGCon 2016**
  - https://www.pgcon.org/2016/schedule/attachments/422_A%20Challenge%20of%20Huge%20Billing%20System%20Migration_20160520.pdf

References

- **Beyond EXPLAIN: Query Optimization From Theory To Code**
  - by YutoHayamizu, RyojiKawamichi
  - https://www.pgcon.org/2016/schedule/attachments/433_PGCON2016_beyond_explain.pdf
- **AQO**
  - by Oleg Ivanov
  - https://github.com/postgrespro/aqo