\Orchestrating a brighter world NEC

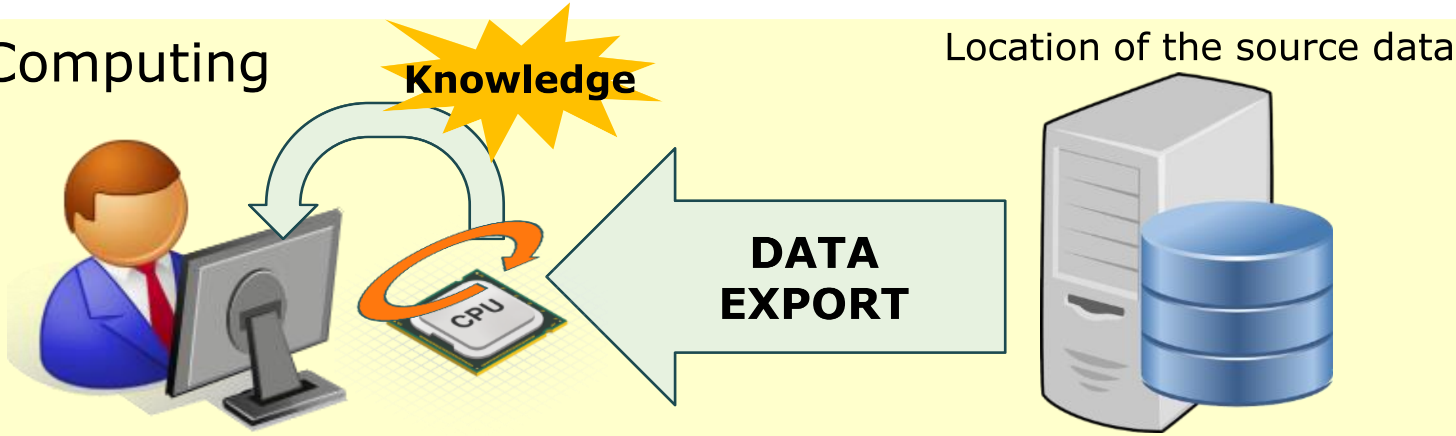
In-place computing on PostgreSQL

~SQL as a shortcut of GPGPU~

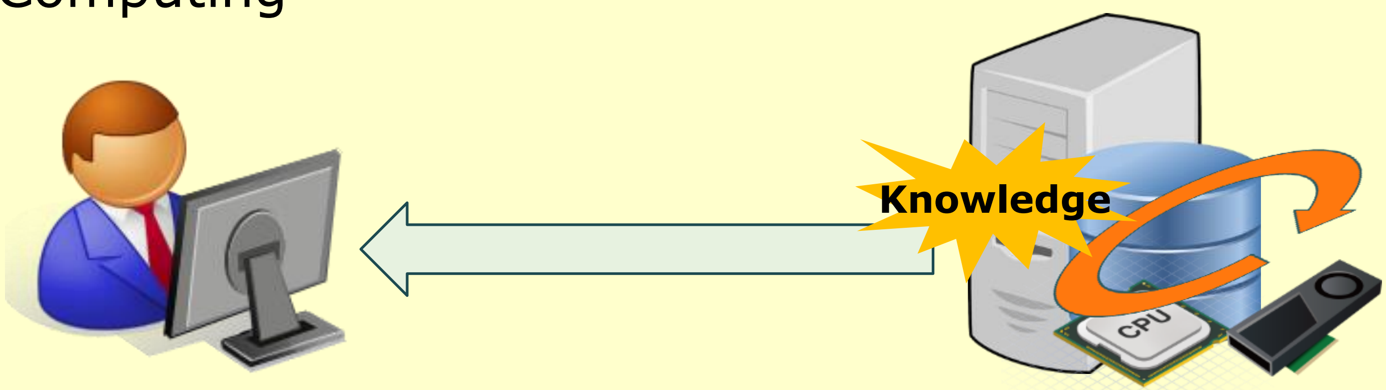
NEC Business Creation Division
The PG-Strom Project
KaiGai Kohei <kaigai@ak.jp.nec.com>

Where is better location to compute?

Local Computing



In-place Computing



about In-place Computing

Advantage

- Less amount of data transfer
- Utilization of server grade hardware capability
- Analytics towards the latest dataset

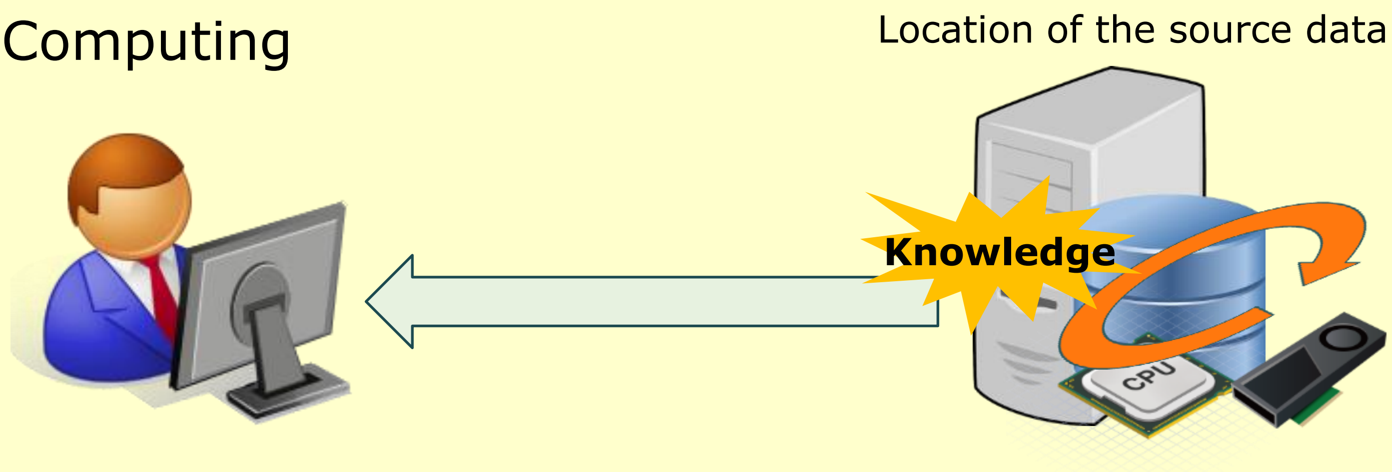
Disadvantage

- Server needs to have sufficient computing resource.

→distributed system is also an option, like Hadoop

- Server software needs to be designed for data processing.

In-place Computing

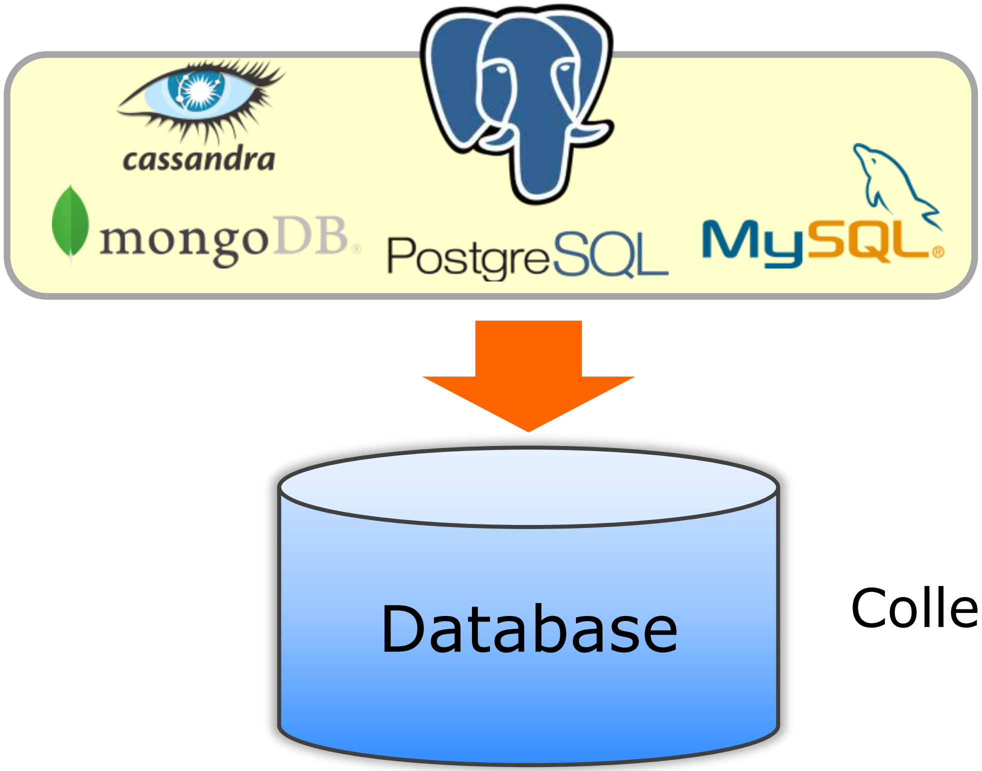


Who Am I?

- name: KaiGai Kohei
- mission: Development of GPU acceleration feature (PG-Strom) for PostgreSQL, and its business related stuff
- company: NEC
- background:
  - about 10 years experiences in PostgreSQL developer community
  - various contributions to the PostgreSQL core:
    - Security-Enhanced PostgreSQL (sepgsql), Security Barrier View, Writable FDW, Remote Join, Custom Scan/Join interface, ...etc...
  - Joined to GPU/CUDA development since 2012

DBMS ≠ Database, as literal

DBMS = Database Management System

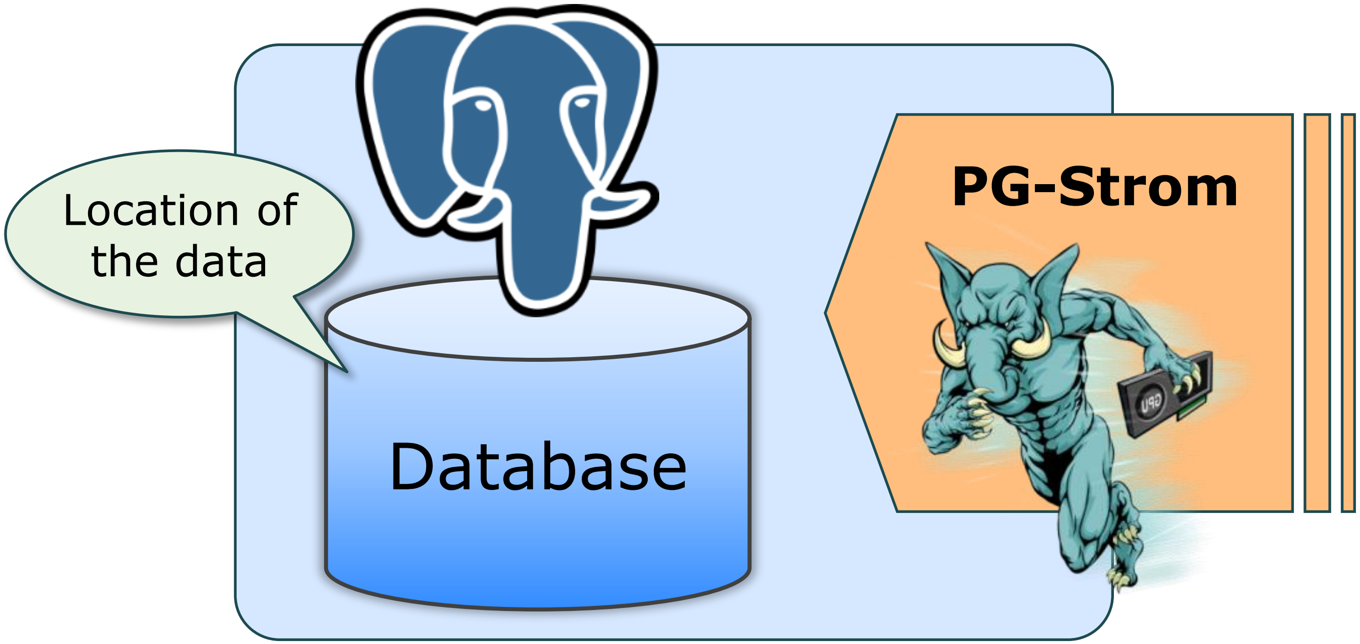


Software to “manage” database

Collection of dataset

- Optimized to store/reference fraction of the dataset scattered on the storage system (e.g indexing, ...)
- Not designed for massive calculation as primary purpose

Dogma of data processing close to the data location

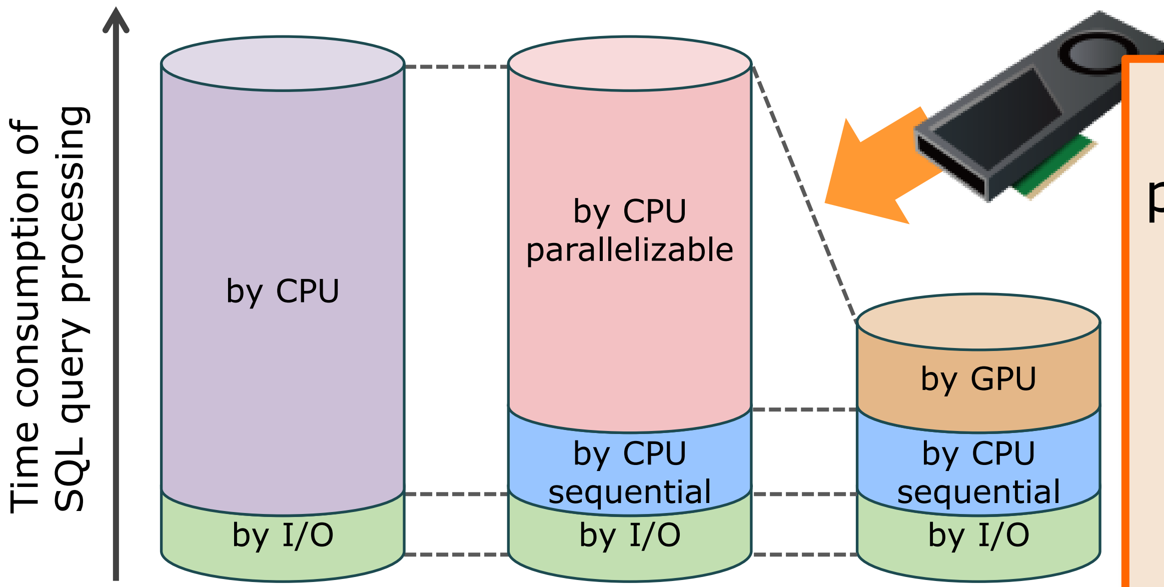


Database Management + Processing Capability

- DBMS is the best software platform for data processing, <u>if we could add reasonable computing capability</u>.

→ **PG-Strom** is an extension of PostgreSQL, to adds data processing capability using GPU

Overview of PG-Strom



GPU will take
parallelizable workloads,

CPU focuses on
what only CPU can do

By an extension
of PostgreSQL

Two core ideas

- GPU Native Code Generation on the fly
- Fully utilization of PostgreSQL infrastructure

Advantages

- CPU intensive SQL: OLAP, Reporting, Batch, Calculation, ...
- Continuity of application and user’s skill
- Inexpensive solution using OSS + GPU

Core ideas (1/2) – Native GPU code generation on the fly

```
QUERY:  SELECT cat, count(*), avg(x) FROM t0
        WHERE x between y and y + 20.0 GROUP BY cat;
```

E.g) Mathematical formula in SQL into CUDA code on the fly

```
        :
STATIC_FUNCTION(bool)
gpupreagg_qual_eval(kern_context *kcxt,
                    kern_data_store *kds,
                    size_t kds_index)
{
  pg_float8_t KPARAM_1 = pg_float8_param(kcxt,1);
  pg_float8_t KVAR_3 = pg_float8_vref(kds,kcxt,2,kds_index);
  pg_float8_t KVAR_4 = pg_float8_vref(kds,kcxt,3,kds_index);

  return EVAL((pgfn_float8ge(kcxt, KVAR_3, KVAR_4) &&
               pgfn_float8le(kcxt, KVAR_3,
                 pgfn_float8pl(kcxt, KVAR_4, KPARAM_1))));
}       :
```

Reference to input data

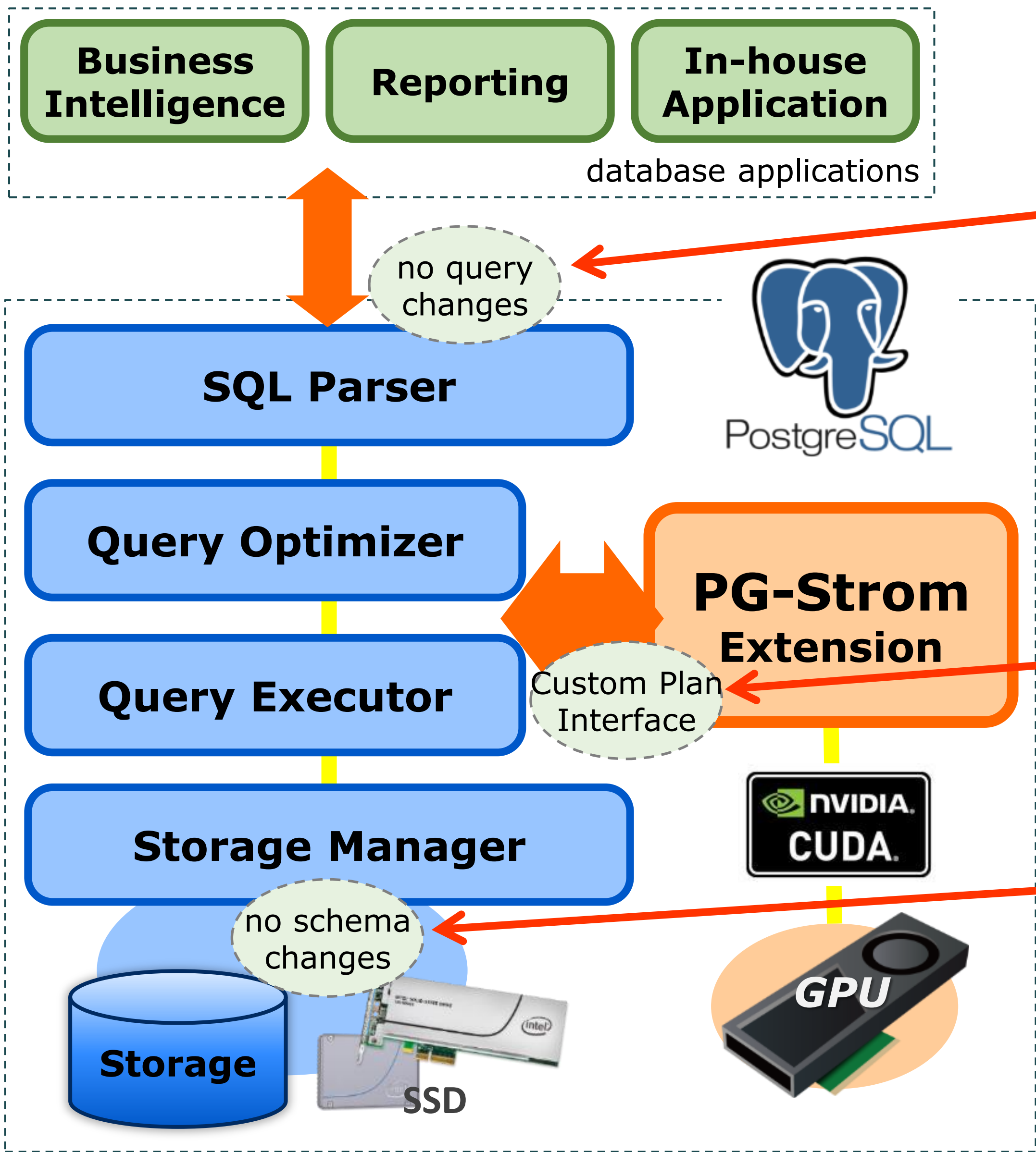
SQL expression in CUDA source code

Just-in-time Compile

Run-time Compiler (nvrtc)

Parallel Execution

Core ideas (2/2) – Fully Utilization of PostgreSQL Infrastructure



No change of SQL syntax

SQL parser break down the given query string into internal parse-tree structure. PG-Strom references only internal tree and don't required any special syntax enhancement.

No patch of PostgreSQL

Custom plan interface allows to inject alternative query execution paths, then executes them if estimated cost is reasonable then built-in implementation.
PG-Strom can be installed on the PostgreSQL now we're operating.

No change of Data Schema

PG-Strom never requires special storage format more than what PostgreSQL currently has.
It eliminates necessity of application modification and extra administrations.

Supported Features

①**GpuProjection**:
Query with calculations

```
SELECT c.category,
       max((x.p1 – y.p1)^2 + ... + (x.p10 – y.p10)^2) dist
  FROM sample_data x
  JOIN sample_data y ON x.id < y.id
  JOIN category c ON x.cat_id = c.cat_id
WHERE x.date BETWEEN '2010-01-01'::date AND NOW()::date
   AND x.quality > 0.8
GROUP BY c.category;
```

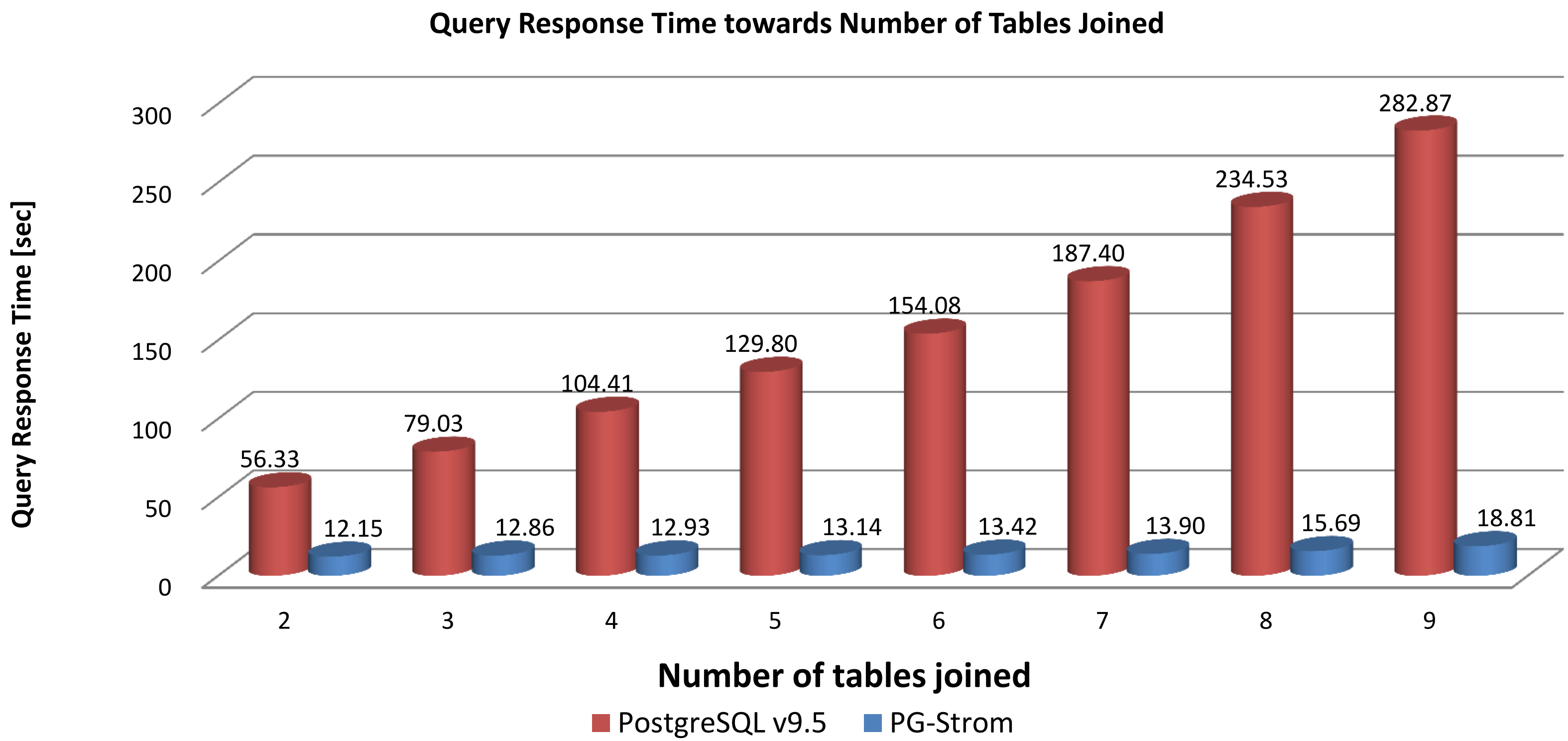
②**GpuJoin**:
N-way parallel Join

④:**GpuPreAgg:**
Aggregation/GROUP BY

③**GpuScan**:
Scan with WHERE clause

- PG-Strom suggest GPU accelerated plan if above workloads.
- Optimizer will choose cheaper execution plan if any.

Benchmark (1/2) – JOIN + GROUP BY on microbenchmark



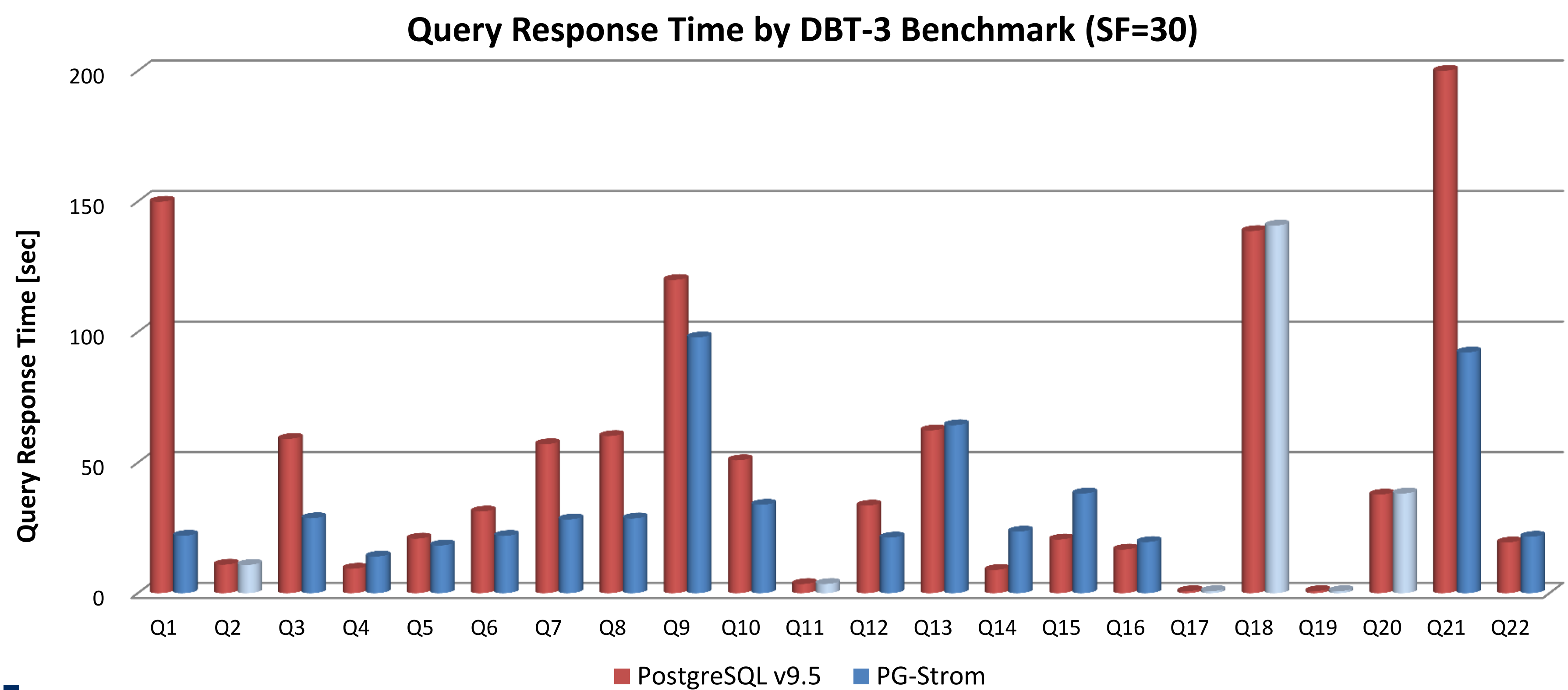
- QUERY:

```
SELECT cat, AVG(x) FROM t0 NATURAL JOIN t1 [, ...] GROUP BY cat;
```

✓ t0: 100M rows, t1~t10: 100K rows for each, all the data was preloaded.

- Environment:
  - PostgreSQL v9.5b + PG-Strom (4-Feb), CUDA 7.5 + CentOS 7(x86_64)
  - CPU: Xeon E5-2670v3, RAM: 384GB, GPU: NVIDIA GTX980 (2048cores)

Benchmark (2/2) – DBT-3, OLAP Benchmark



about DBT-3 benchmark:

- Clone of TPC-H benchmark, an open source edition.
- Simulate various reporting queries on large retail industry.

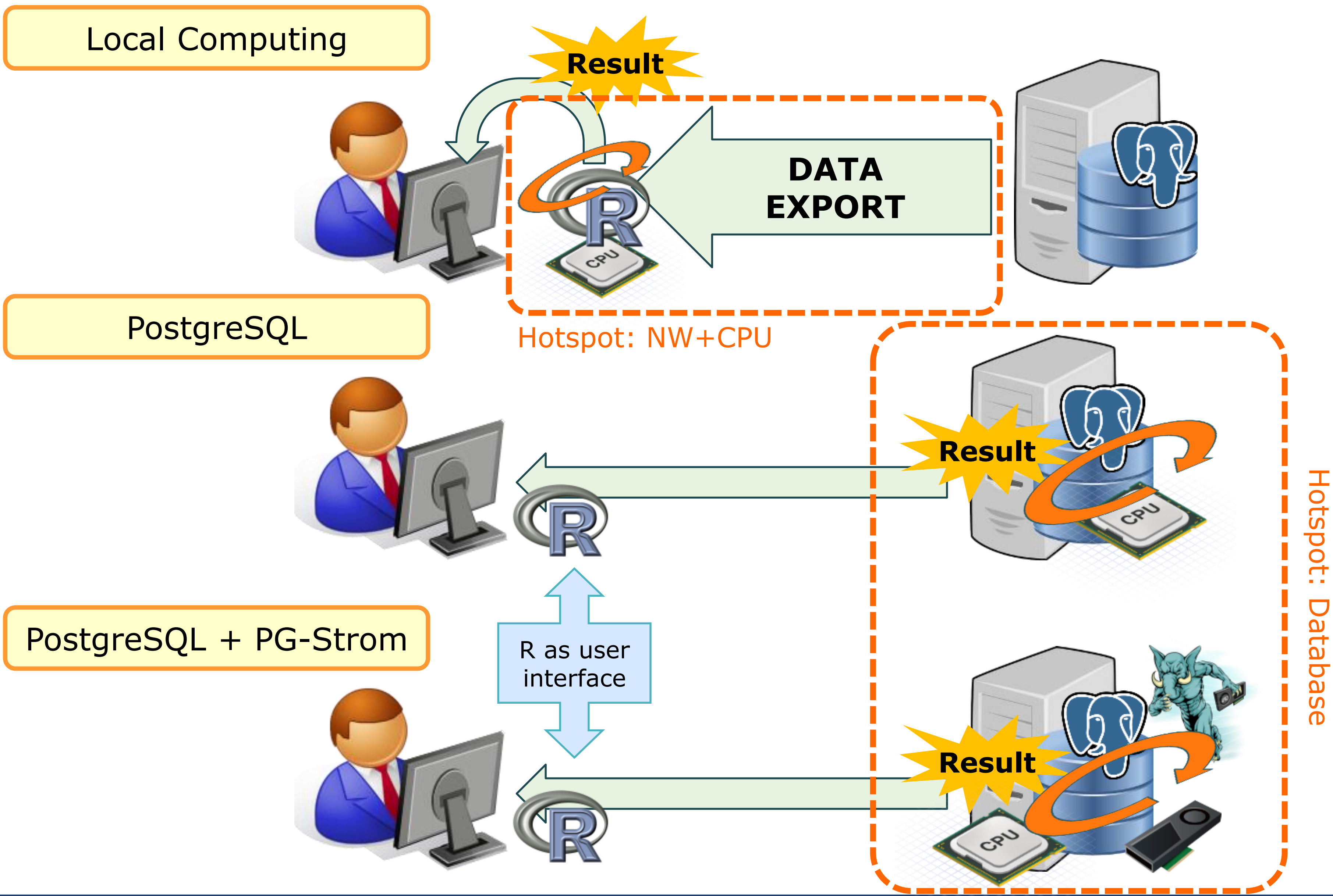
Note:

- Same test environment with last page
- Light-blue-colored bar involves no GPU execution due to query optimization
- Q21 in PostgreSQL didn't finish 2Hr, so I tuned the parameter by hand

Workloads characteristics

	OLTP	OLAP	In-place Computing
I/O	Large (read+write hybrid)	Small (mostly read)	Tiny (mostly read)
CPU	Small, sequential	Large, parallelizable	Large, parallelizable
Typical SQL	index accesses, UPDATE, INSERT,...	JOIN, GROUP BY, SORTING, ...	Numerical Calculations, ...
Concurrency	Massive	Small	Tiny
KPI	Latency	Throughput	Latency + Throughput
For PG-Strom	Not help	Originally Designed for	New Challenge

Test Scenario of In-place computing

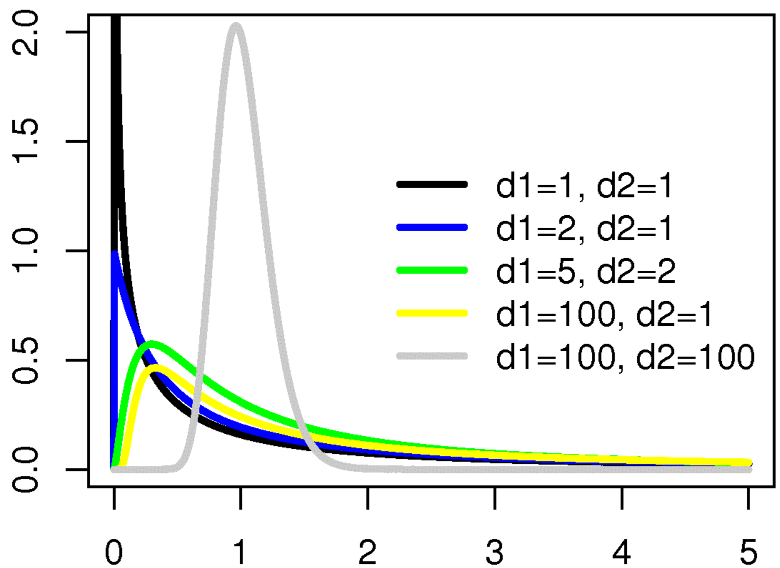


First Challenge – F-Test

F-Test

- A method for analysis of variance
- Checks the ratio of sample variance of two groups
- If same variance, its ratio shall follow F-distribution at statistically reasonable area.

➔ Sample variances, # of items are needed.



F-distribution

Data Set

- Heterogeneity Activity Recognition Data Set
  - http://archive.ics.uci.edu/ml/datasets/Heterogeneity+Activity+Recognition
- records of accelerometer and gyroscope of mobile device
- includes anonymized user-id, model of device, user's act
- 43.9M rows, 3.3GB in total

Code for F-Test

By Local R-script

```
# Obtain dataset from database
conn <- dbConnect(PostgreSQL())
sql <- "SELECT model, x, y, z
          FROM phones_accelerometer"
r <- dbGetQuery(conn, sql)
dbDisconnect(conn)

# Pickup ‘s3mini’ items
cond <- r[['model']] == 's3mini’
s3mini <- r[cond,]

# Pickup ‘nexus4’ items
cond <- r[['model']] == ‘nexus4’
nexus4 <- r[cond,]

# Run F-test on two variances
result <- var.test(s3mini[['x']]+
                   s3mini[['y']]+
                   s3mini[['z']],
                   nexus4[['x']]+
                   nexus4[['y']]+
                   nexus4[['z']])
```

By SQL

```
# Send query to get variances and
# number of items
conn <- dbConnect(PostgreSQL())
sql <- “SELECT count(*),
               variance(x+y+z) var
          FROM phones_accelerometer
         GROUP BY model
        HAVING model = ‘nexus4’ OR
               model = ‘s3mini’”
r <- dbGetQuery(conn, sql)
dbDisconnect(conn)

# Fetch the result
lbound <- qf(0.025, v[1,'count']-1,
                    v[2,'count']-1)
ubound <- qf(0.975, v[1,'count']-1,
                    v[2,'count']-1)
fv <- v[1, ’var’] / v[2, ’var’]
# Set result
result <- c(lbound, fv, ubound)
```

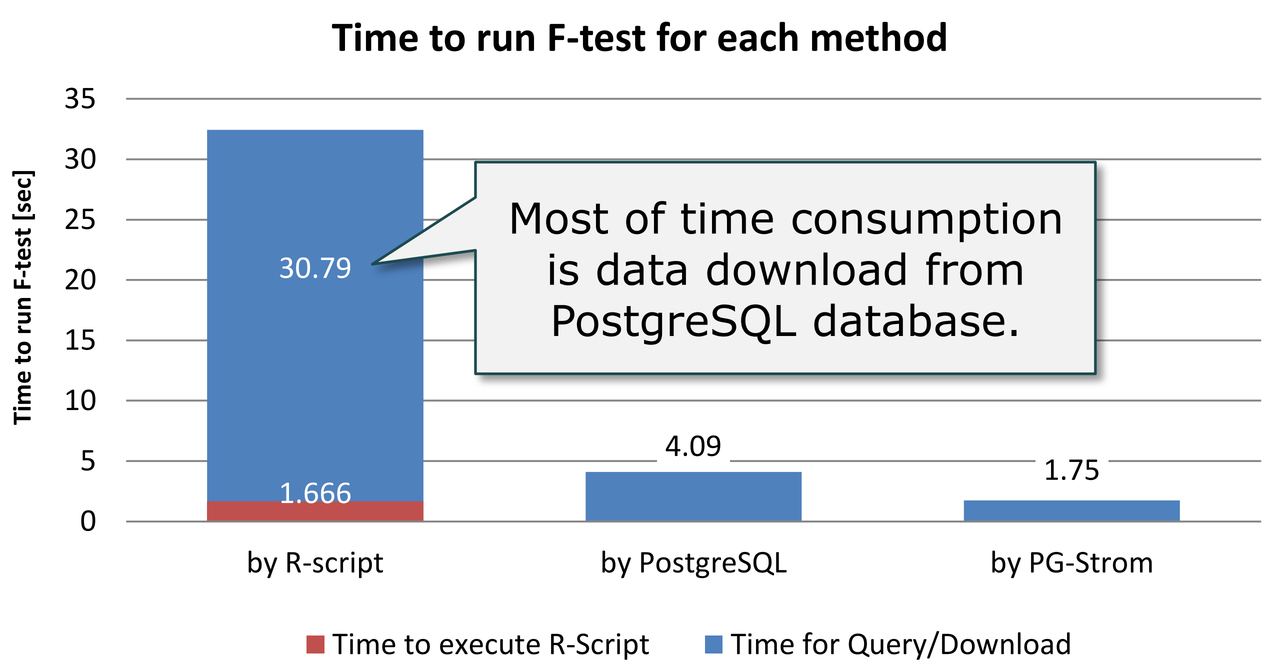
Test results

Difference of accelerometer between 'nexus4' and 's3mini'

```
> pgsql_ftest(0.05)
[1] 0.9978336 1.0018144 1.0021688
```

$F_{lower} = 0.9978336, \quad F = 1.0018144, \quad F_{upper} = 1.0021688$

$F_{lower} \leq F \leq F_{upper}$ ➔ These two group have likely same variances.



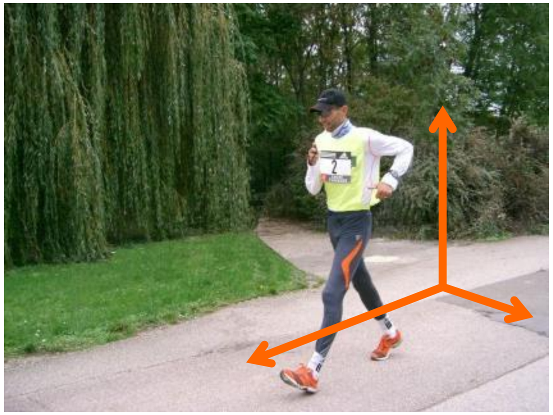
Adjustment of the test hypothesis

No difference in models, How about people's act?

```
   SELECT gt, count(*), variance(x+y+z) var
     FROM phones_accelerometer
    GROUP BY gt
   HAVING gt = 'walk' OR gt = 'bike'
> pgsql_ftest(0.05)
[1] 0.9979628 2.1744231 1.0020435
```

$F_{lower} = 0.9979628, \quad F = 2.1744231, \quad F_{upper} = 1.0020435$

$F_{upper} \ll F$ ➔ These two groups likely have different variances.



Adjusted Query and GPU code on the fly (1/2)

```
mobile=# EXPLAIN SELECT gt, count(*), variance(x+y+z) var
     FROM phones_accelerometer
    GROUP BY gt
   HAVING gt = 'walk' or gt = 'bike';
                            QUERY PLAN
------------------------------------------------------------------
 HashAggregate  (cost=255006.21..255006.25 rows=3 width=30)
   Group Key: gt
  -> Custom Scan (GpuPreAgg) on phones_accelerometer
                (cost=10063.55..215505.32 rows=255 width=56)
     Reduction: Local + Global
     GPU Projection: index, arrival_time, creation_time, x, y, z,
user_id, model, device, gt
     GPU Filter: ((gt = 'walk'::text) OR (gt = 'bike'::text))
     Kernel Source: /opt/..<snip>../pgsql_tmp_strom_118438.5.gpu
(6 rows)
```

Adjusted Query and GPU code on the fly (2/2)

```
% cat /opt/pgsql/kaigai/base/pgsql_tmp/pgsql_tmp_strom_118438.5.gpu
    :
    :
STATIC_FUNCTION(bool)
gpupreagg_qual_eval(kern_context *kcxt,
                    kern_data_store *kds,
                    size_t kds_index)
{
  pg_text_t KPARAM_1 = pg_text_param(kcxt,1);
  pg_text_t KPARAM_2 = pg_text_param(kcxt,2);
  pg_text_t KVAR_10 = pg_text_vref(kds,kcxt,9,kds_index);

  return EVAL((pgfn_texteq(kcxt, KVAR_10, KPARAM_1) ||
               pgfn_texteq(kcxt, KVAR_10, KPARAM_2)));
}
    :
    :
```

constant values:
'walk' and 'bike'

column 'gt'
in this record

Equivalent to the condition in SQL:

`gt = 'walk' or gt = 'bike'`

Next Challenge

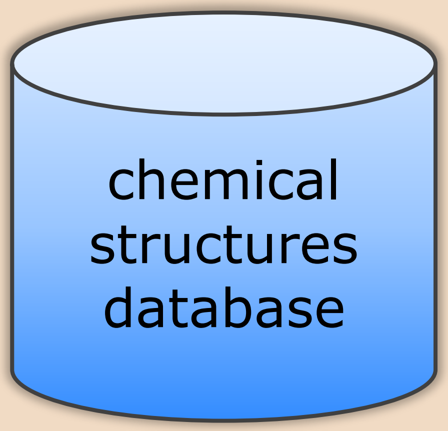
×

Information Technology

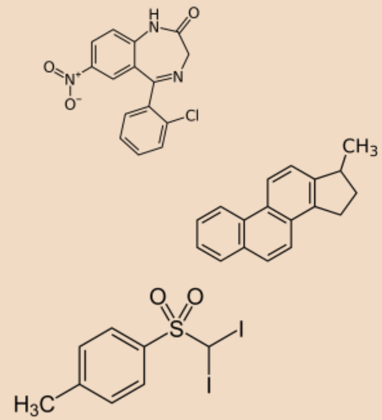
Drug Discovery

Background (1/2) – Diversified Chemical Structures Library

world of dry

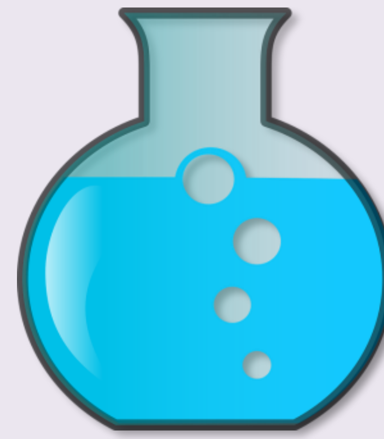


More than
million items



Pick up
small samples

world of wet



Investigation
at the Lab

To be avoided...

- Taking experiment on multiple but similar chemical components

Requirement

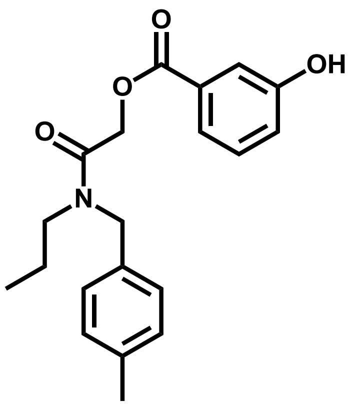
- Pick up small number of items from the database
- Each chemical structure are “different” as possible as we can

Background (2/2) – Chemical structures representation

Molecular Quantum Numbers (MQN)

A vector of 42 descriptor which represents characteristics of chemical structures

2D structure



List of the descriptors

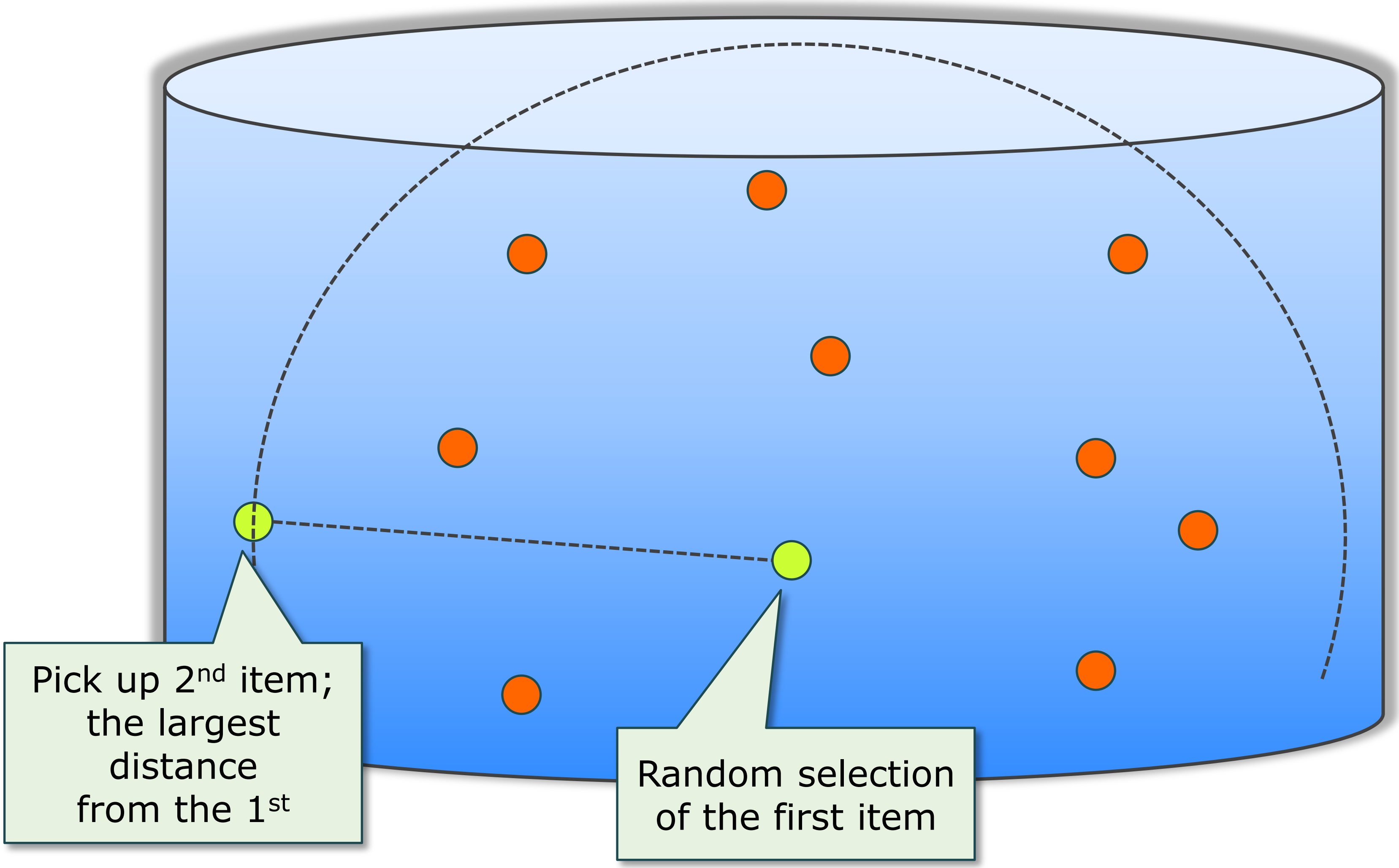
1. c (carbon)
2. f (fluorine)
3. cl (chlorine)
4. br (bromine)
5. i (iodine)
6. s (sulfur)
7. p (phosphorous)
8. an (acyclic nitrogen
9. cn (cyclic nitrogen)
10. ao (acyclic oxygen)
11. co (cyclic oxygen)
12. hac (heavy atoms)
13. asb (acyclic single bonds)
14. adb (acyclic double bonds)
15. atb (acyclic triple bonds)
16. csb (cyclic single bonds)
17. cdb (cyclic double bonds)
18. ctb (cyclic triple bonds)
19. rbc (rotatable bonds)
20. hbam (H-bond acceptor sites)
21. hba (H-bond acceptor atoms)
22. hbdm (H-bond donor sites)
23. hbd (H-bond donor atoms)
24. negc (negative charges)
25. posc (positive charges)
26. asv (acyclic single valent nodes)
27. adv (acyclic divalent nodes)
28. atv (acyclic trivalent nodes)
29. aqv (acyclic tetravalent nodes)
30. cdv (cyclic divalent nodes)
31. ctv (cyclic trivalent nodes)
32. cqv (cyclic tetravalent nodes)
33. r3 (3-membered rings)
34. r4 (4-membered rings)
35. r5 (5-membered rings)
36. r6 (6-membered rings)
37. r7 (7-membered rings)
38. r8 (8-membered rings)
39. r9 (9-membered rings)
40. rg10 (10-membered rings)
41. afrc (nodes shared by 2 rings)
42. bfrc (edges shared by 2 rings)

A vector of 42 descriptors

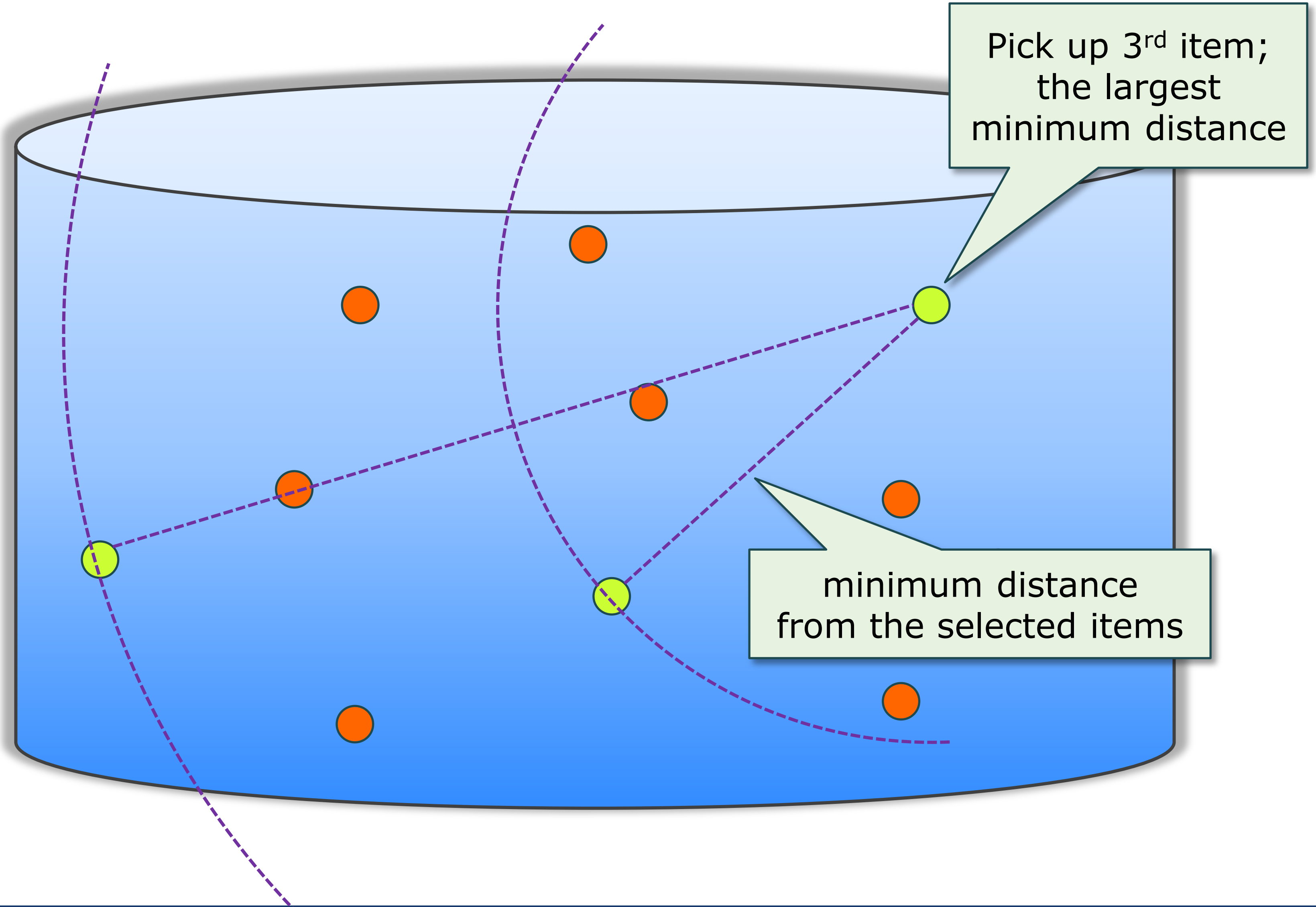
(20, 0, 0, 0, 0, 0, 0, 1, 0, 4, 0, 0, 12, 2, 0, 6, 6, 0, 9, 9, 5, 1, 1, 0, 0, 5, 5, 3, 0, 8, 4, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0)

Quote: J.Chem.Inf.Model, 2013, 53, 509–518

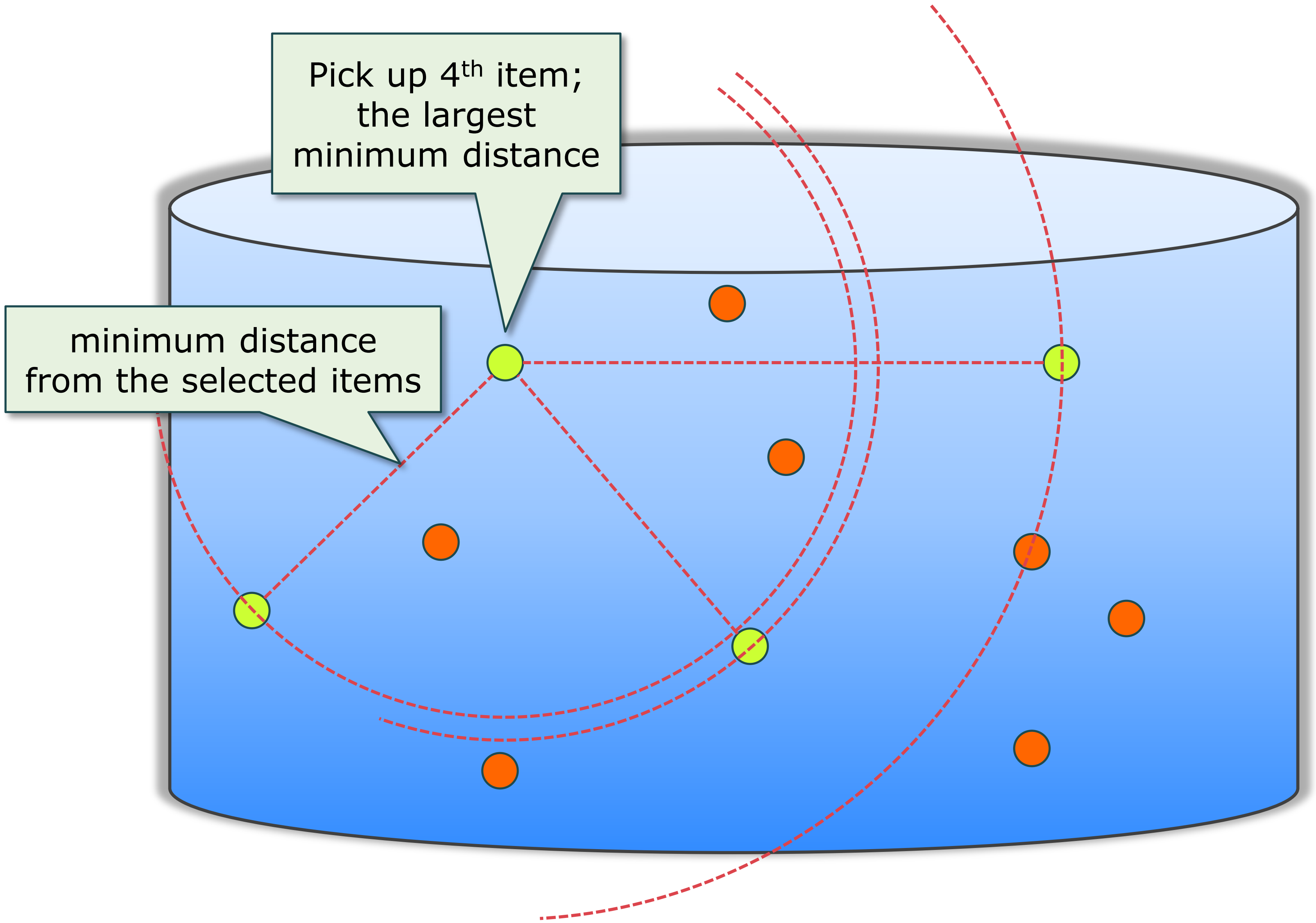
Workload – MAX-MIN Method



Workload – MAX-MIN Method



Workload – MAX-MIN Method

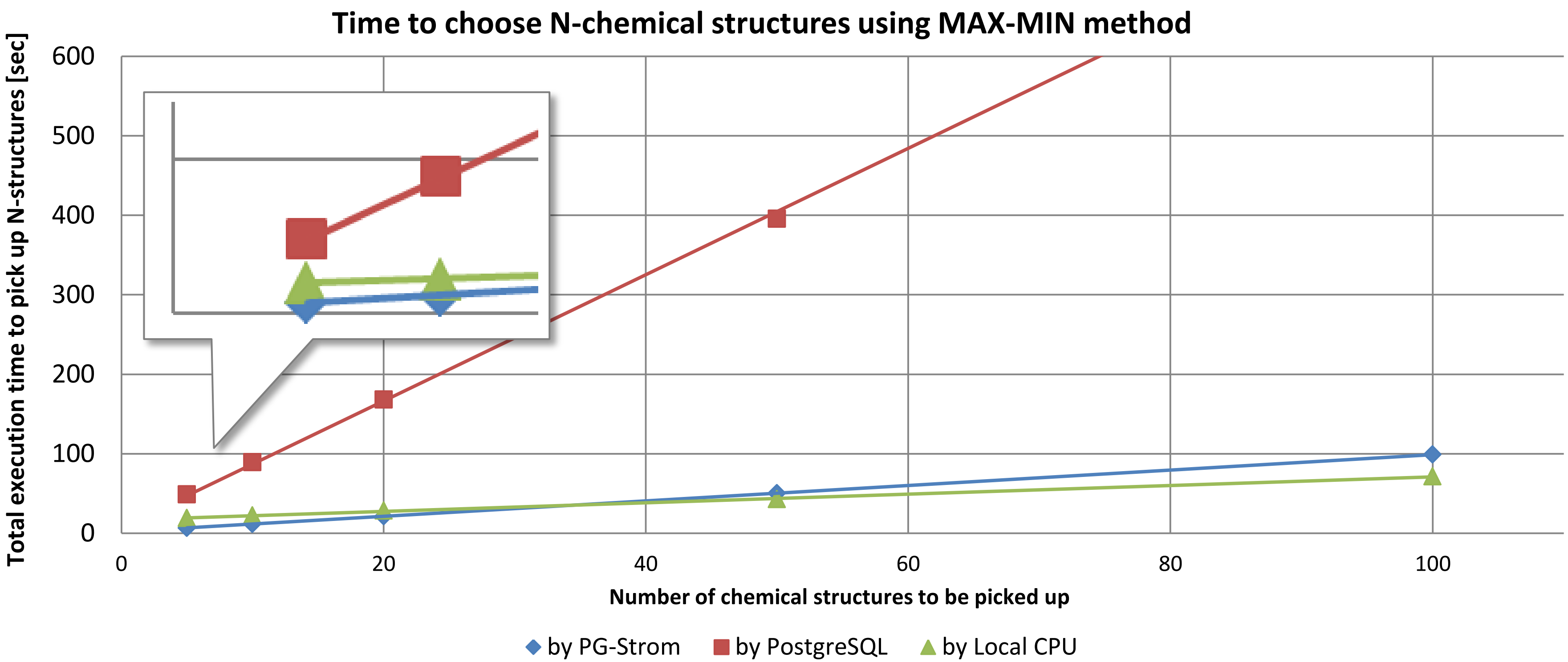


Reference) Query for MAX-MIN method

```
WITH next_item AS (
  INSERT INTO pg_temp.subset_table (
    SELECT r.*
      FROM mqn r,
           (SELECT id FROM pg_temp.dist_table
                     ORDER BY dist DESC LIMIT 1) d
     WHERE r.id = d.id)
 RETURNING *
)
SELECT r.id, LEAST(d.dist, sqrt((r.c1 - n.c1)^2 +
                                (r.c2 - n.c2)^2 +
                                      :
                                (r.c41 – n.c41)^2 +
                                (r.c42 - n.c42)^2)) dist
  INTO pg_temp.dist_table_new
  FROM pg_temp.dist_table d,
       next_item n, mqn r
 WHERE r.id = d.id
```

Hot point of the SQL query

Benchmark Results – MAX-MIN Method



Summary

- 10M rows, 211MB in total
- workload characteristics: $W_{I/O} \ll W_{CPU}$
- If no GPU support, calculation in RDBMS cannot be an option.
- PG-Strom recorded similar performance with download + R-script.

Conclusion

Advantages

- New usage of RDBMS, by integration of GPU capability.
- No more CSV dump to process statistical data.
  (also, analytics towards the latest data)
- SQL can be a way to alternate GPU programming for parallel-data processing.
- Inexpensive solution due to OSS + GPU

Future challenges

- Packaging for R, to run major analytic algorithm on database.
- More performance, than similar to local R-script.
  - Ideas) CPU+GPU hybrid parallel, SSD to GPU direct, columnar support
- Procedural Language support

Project Roadmap

2016

2017

2018

PostgreSQL 9.5

Custom-Plan Interface
:

PostgreSQL 9.6

CPU Parallel Query Execution
Combined Aggregation
Upper Path Optimization
:

PostgreSQL 9.7

Native Columnar Storage
Native Partitioning Support
:

PG-Strom 1.0

The first release that support primitive data types and workloads.

PG-Strom 1.1

Multi-backend concurrency improvement
Extra operators & functions support

PG-Strom 2.0

CPU+GPU Hybrid Parallel
SSD-to-GPU Direct Support
More graceful query optimization
Enterprise grade SW quality

PG-Strom 3.0(?)

Columnar storage support
3rd party extensions support
Self defined CUDA function

Resources

Source Code

- https://github.com/pg-strom/devel

Documentations

- http://strom.kaigai.gr.jp/manual.html

Questions

- Issue tracker:
  - https://github.com/pg-strom/devel/issues
- Direct contact:
  - e-mail: kaigai@ak.jp.nec.com
  - twitter: @kkaigai

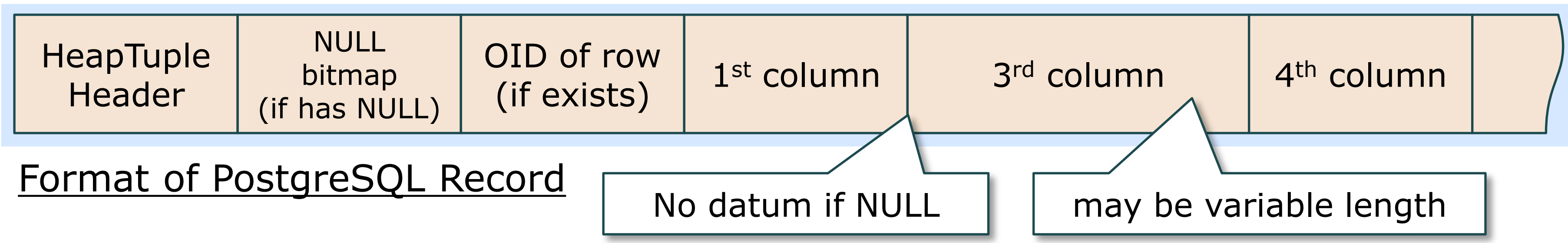
THANK YOU!

The PG-Strom Development Team

Backup: What we could learn from the workload

Challenge (1) – Cost to access datum in record

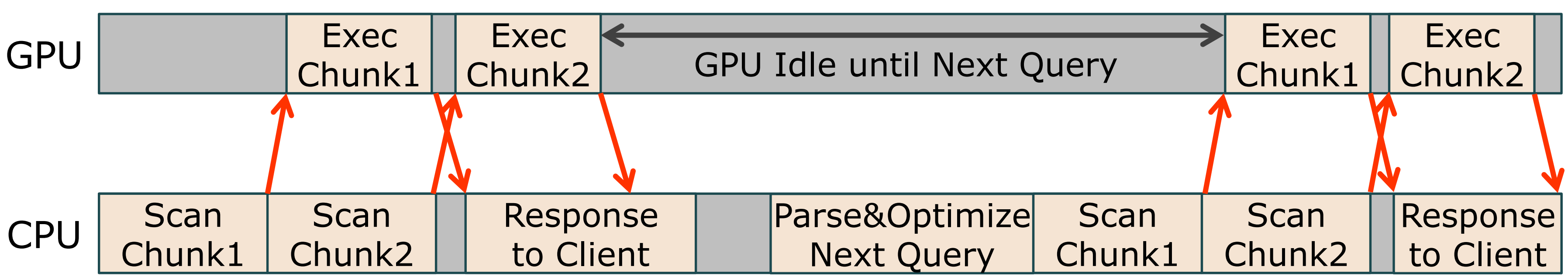
- Symptom: row-format is worst data structure for GPU
- Solution: columnar-format; planned for PostgreSQL v9.7



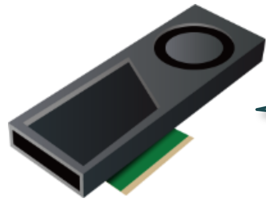
Format of PostgreSQL Record

Challenge (2) – Less Concurrent Multiprocessing Gain

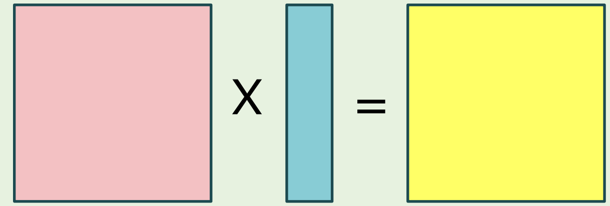
- Symptom: Each iteration is blocked by the client script
- Solution: Implement entire iteration with SQL/SQL function



Backup: Be more aggressive!



This SQL function kicks CUDA kernel internally




```
SELECT kmeans_plus(matrix, 10)
  FROM (SELECT matrix(c1, c2, ..., c42)
          FROM chemical_master
         WHERE <condition for scan>
      GROUP BY <condition for grouping>;
```

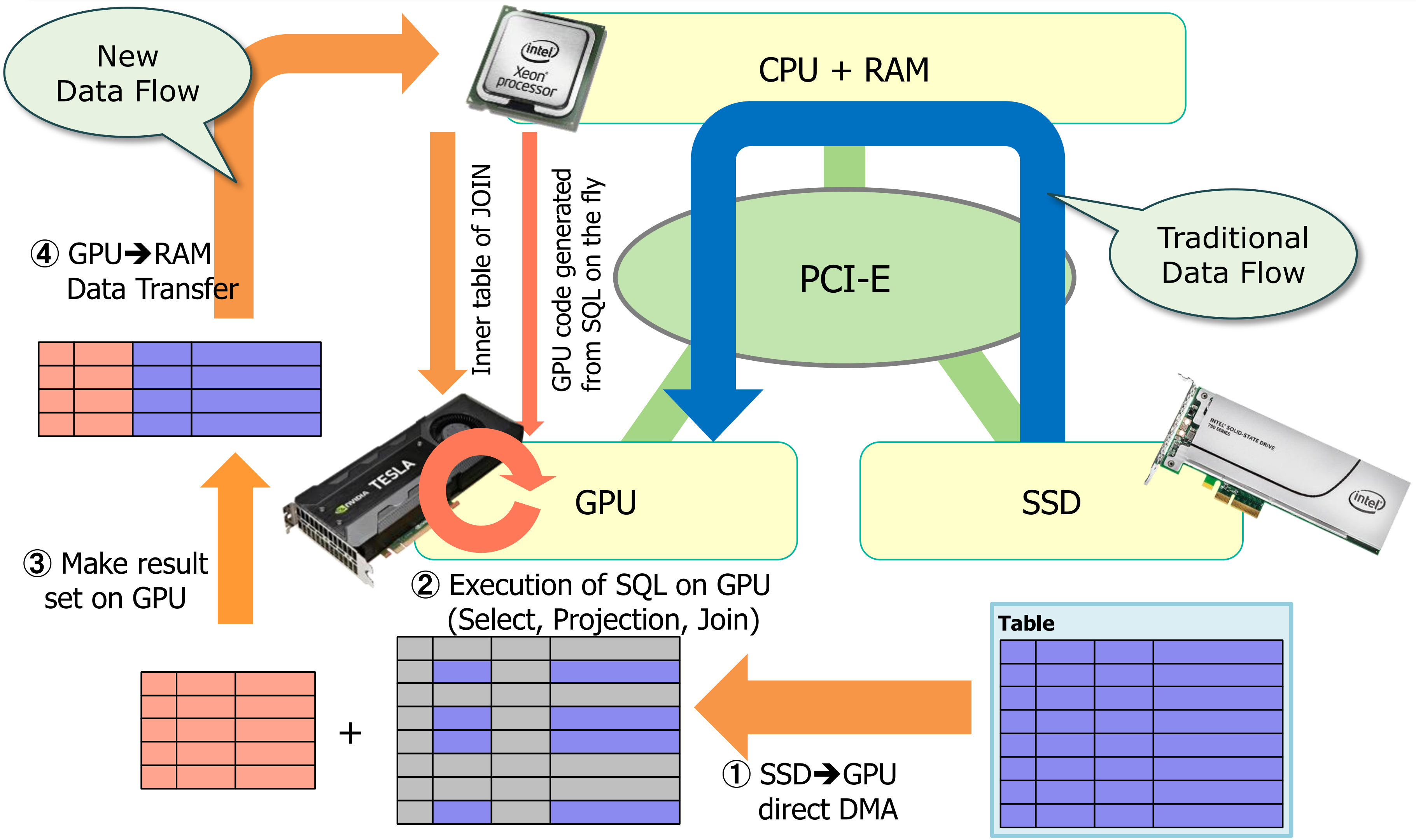
Transform millions of rows into high-density & GPU suitable 2D-array.

Challenge (3) – Self-defined CUDA function on SQL

- Symptom – iteration by script and row-format make slowdown
- Solution – special tune by capability of self-defined CUDA function, with simple 2D-array structure (matrix).

Backup: SSD-to-GPU under SQL execution

Ultimate "Near-Data Computing" than CPU/RAM



\Orchestrating a brighter world

NEC brings together and integrates technology and expertise to create
the ICT-enabled society of tomorrow.
We collaborate closely with partners and customers around the world,
orchestrating each project to ensure all its parts are fine-turned to local needs.

Every day, our innovative solutions for society contribute to
greater safety, security, efficiency and equality,
and enable people to live brighter lives.