GPGPU Accelerates PostgreSQL

~Unlock the power of multi-thousand cores~

PGconf.EU 2015 Wien

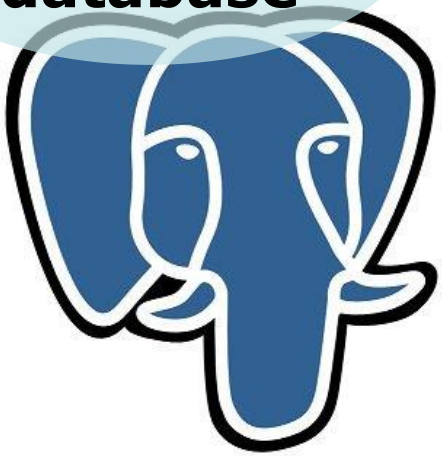
NEC Business Creation Division
The PG-Strom Project
KaiGai Kohei <kaigai@ak.jp.nec.com>

Today's topic

PG-Strom:

Glue of the both promising technologies

Very functional & well-used database



PostgreSQL

GPGPU

Very powerful computing capability

Self Introduction

- Name: KaiGai Kohei
- Works for: NEC
- Role:
  - Lead of the PG-Strom project
  - Contribution to PostgreSQL and other OSS projects
  - Making business opportunity using this technology

PG-Strom Project

- Mission: To deliver the power of semiconductor evolution for every people who want
- Launched at Jan-2012, as a personal development project
- Project is now funded by NEC
- Fully open source project

Agenda

1. **GPU characteristics and why?**
2. What intermediates PostgreSQL and GPU
3. How GPU processes SQL workloads
4. Performance results and analysis
5. Further development

Features of GPU (Graphic Processor Unit)

SOURCE: CUDA C Programming Guide

Characteristics

- Massive parallel cores
- Much higher DRAM bandwidth
- Better price / performance ratio

➔ advantages for massive but simple arithmetic operations

✓ Different manner to write software algorithm

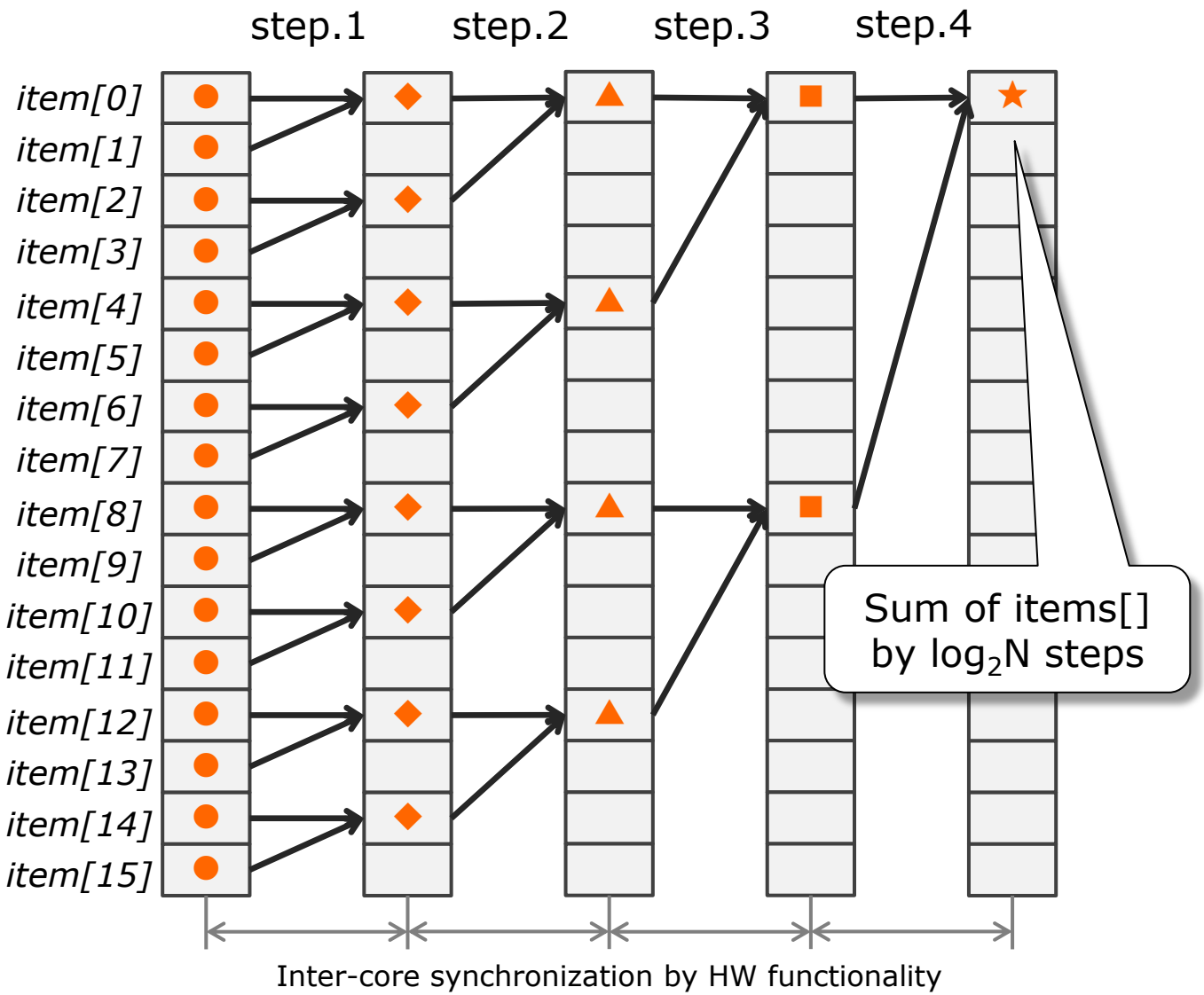
	GPU	CPU
Model	Nvidia GTX TITAN X	Intel Xeon E5-2690 v3
Architecture	Maxwell	Haswell
Launch	Mar-2015	Sep-2014
# of transistors	8.0billion	3.84billion
# of cores	**3072 (simple)**	12 (functional)
Core clock	1.0GHz	2.6GHz, up to 3.5GHz
Peak Flops (single precision)	6.6TFLOPS	998.4GFLOPS (with AVX2)
DRAM size	12GB, GDDR5 (384bits bus)	768GB/socket, DDR4
Memory band	**336.5GB/s**	68GB/s
Power consumption	250W	135W
Price	$999	$2,094

How GPU cores works

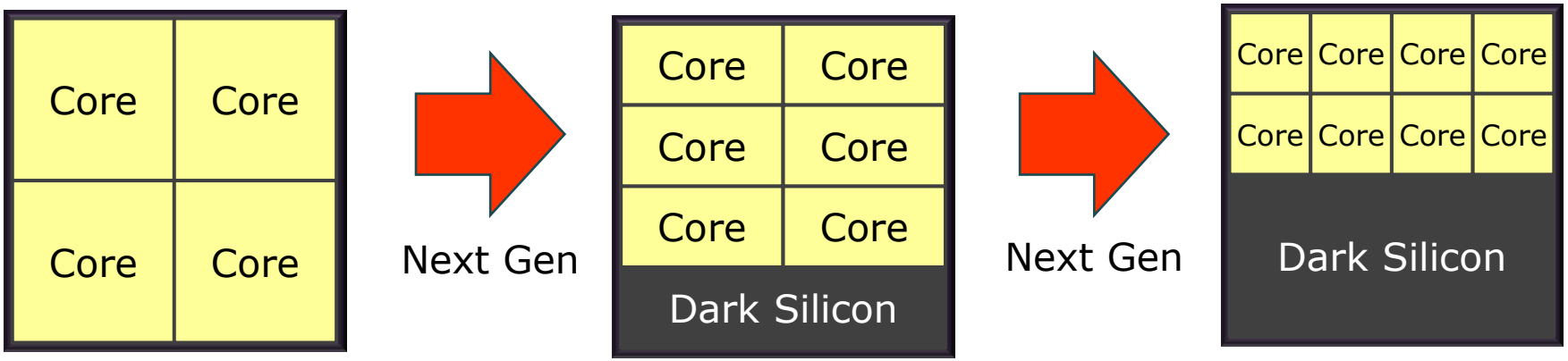
Calculation of

$$\sum_{i=0\ldots N-1} item[i]$$

with GPU cores

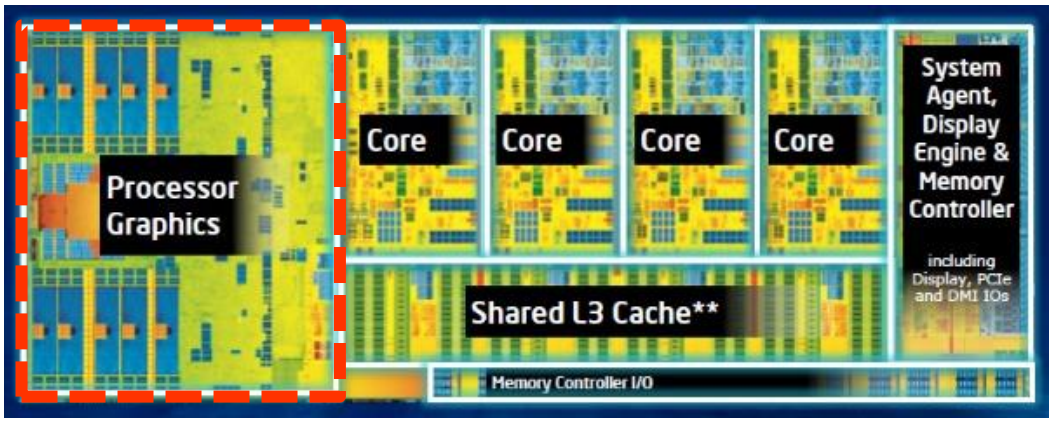


Dark Silicon and Semiconductor Trend



- Processor shrink also increases power leakage.
- Dark Silicon: area cannot power-on due to thermal problem.
- People thought: How to utilize these extra transistors?

➔Domain specific processors, like GPU



Intel Haswell on-chip layout (4core system)

Towards heterogeneous era...

Software must be designed to intermediate CPU and GPU

Agenda

1. GPU characteristics and why?
2. **What intermediates PostgreSQL and GPU**
3. How GPU processes SQL workloads
4. Performance results and analysis
5. Further development

What is PG-Strom

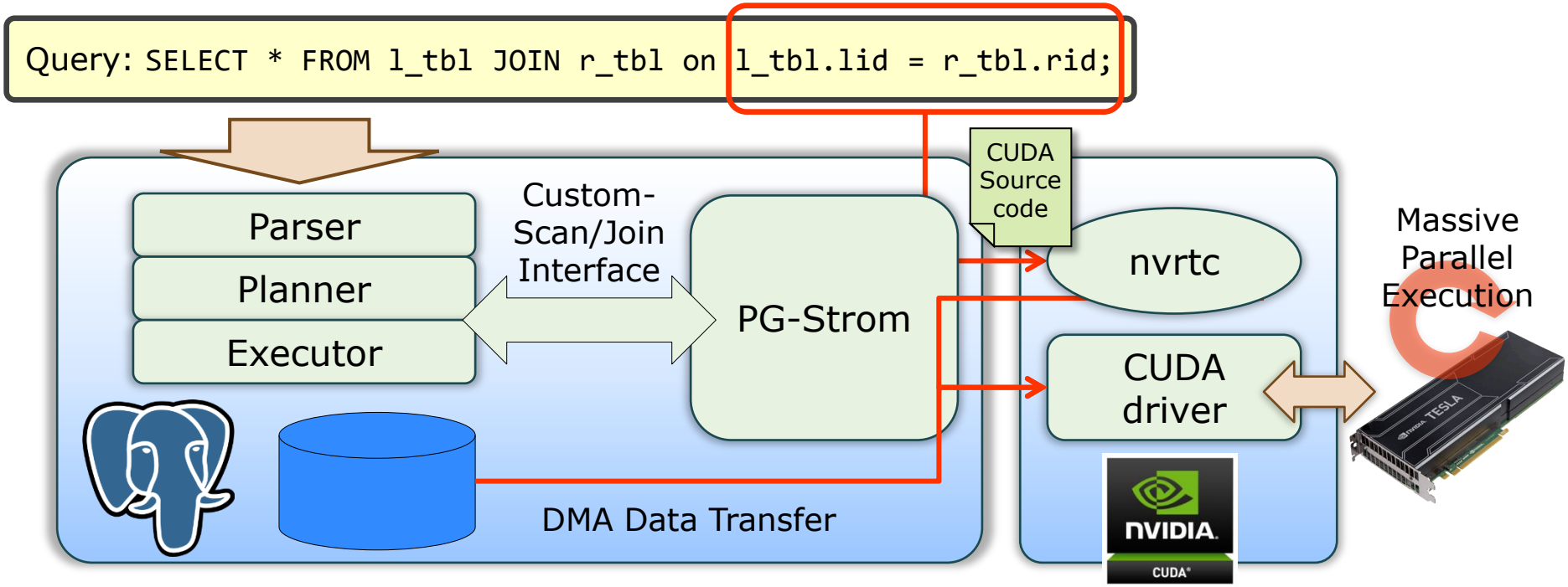
Core ideas

① GPU native code generation on the fly

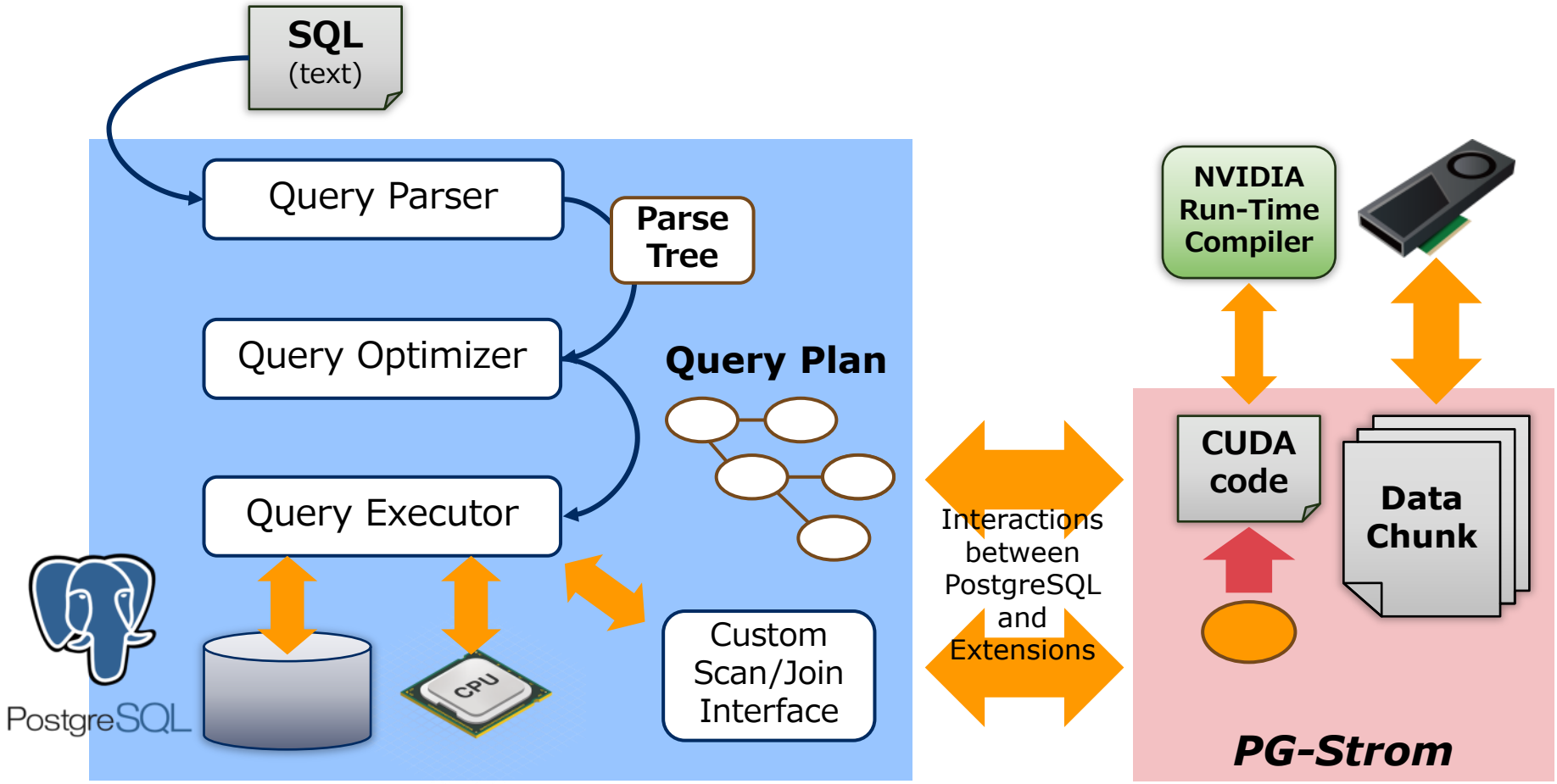
② Asynchronous massive parallel execution

Advantages

- Transparent acceleration with 100% query compatibility
- Commodity H/W and less system integration cost

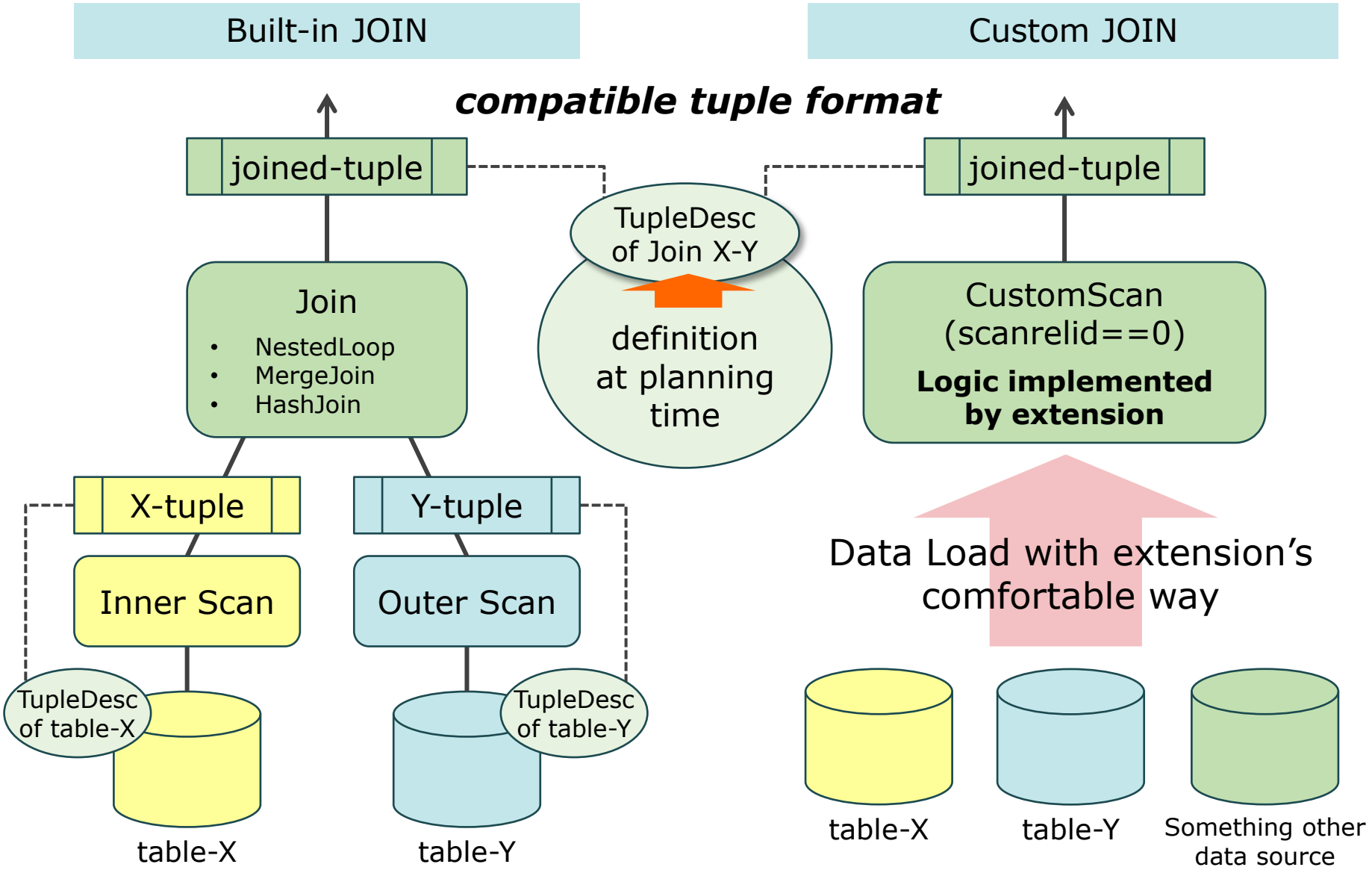


How query is processed



- Custom-Scan/Join interface (v9.5 new!)
- PG-Strom provides own scan/join logics using GPUs
- It generates same results, but different way

(OT) Dive into Custom-Join



SQL-to-GPU native code generation (1/3)

```
postgres=# EXPLAIN VERBOSE
      SELECT cat, count(*), avg(x) FROM t0
             WHERE x between y and y + 20.0 GROUP BY cat;
                                        QUERY PLAN
-------------------------------------------------------------------------------------
HashAggregate  (cost=167646.33..167646.36 rows=3 width=12)
   Output: cat, pgstrom.count((pgstrom.nrows())),
                pgstrom.avg((pgstrom.nrows((x IS NOT NULL))), (pgstrom.psum(x)))
   Group Key: t0.cat
   ->  Custom Scan (GpuPreAgg)  (cost=24150.22..163315.53 rows=258 width=48)
         Output: cat, pgstrom.nrows(), pgstrom.nrows((x IS NOT NULL)), pgstrom.psum(x)
         Bulkload: On (density: 100.00%)
         Reduction: Local + Global
         Device Filter: ((t0.x >= t0.y) AND (t0.x <= (t0.y + '20'::double precision)))
         Features: format: tuple-slot, bulkload: unsupported
         Kernel Source: /opt/pgsql/base/pgsql_tmp/pgsql_tmp_strom_24905.4.gpu
         ->  Custom Scan (BulkScan) on public.t0  (cost=21150.22..159312.94
                                                   rows=10000060 width=12)
               Output: id, cat, aid, bid, cid, did, eid, x, y, z
               Features: format: heap-tuple, bulkload: supported
(13 rows)
```

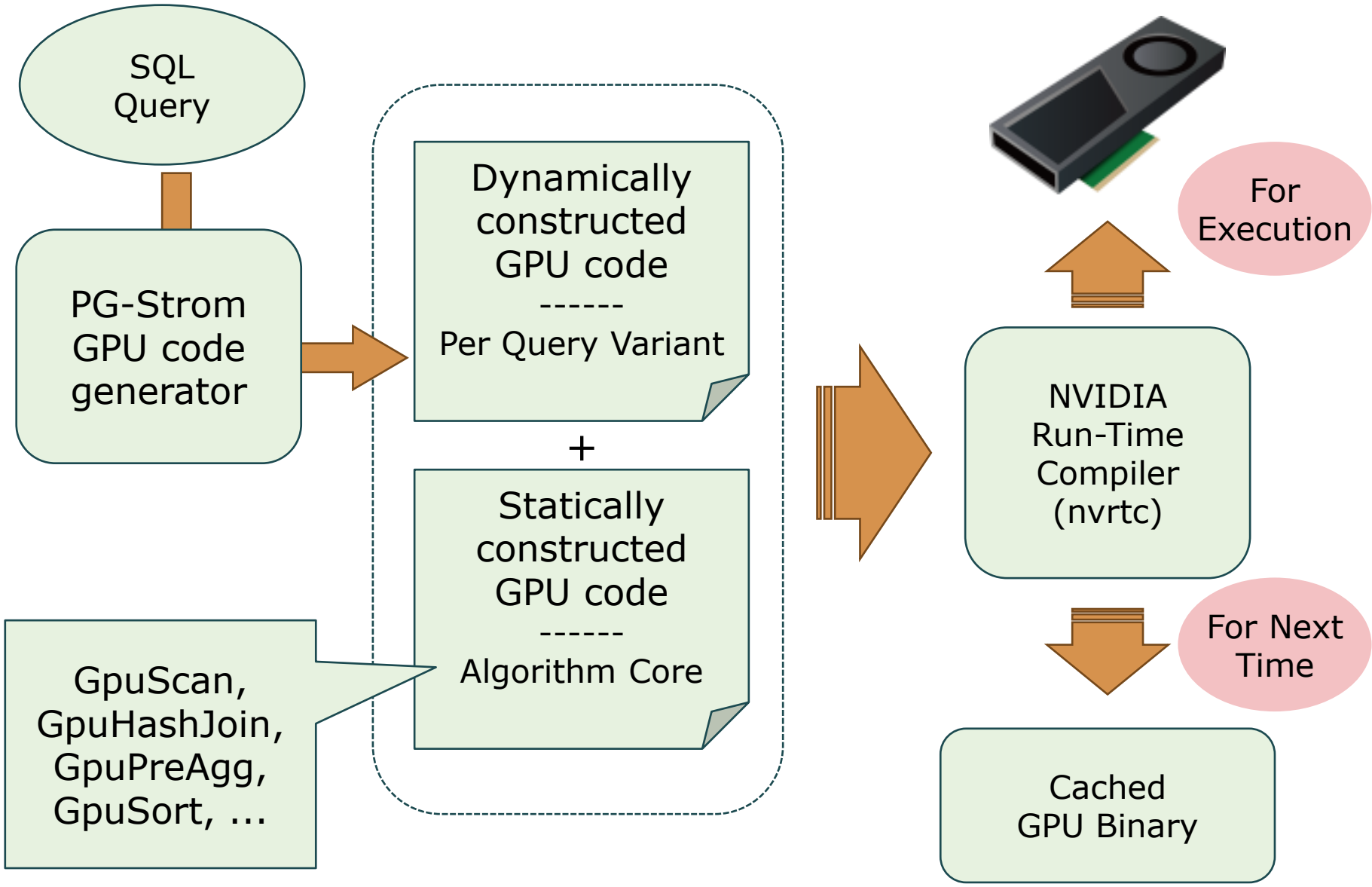
SQL-to-GPU native code generation (2/3)

```
$ less /opt/pgsql/base/pgsql_tmp/pgsql_tmp_strom_24905.4.gpu
                  :
STATIC_FUNCTION(bool)
gpupreagg_qual_eval(kern_context *kcxt,
                    kern_data_store *kds,
                    kern_data_store *ktoast,
                    size_t kds_index)
{
  pg_float8_t KPARAM_1 = pg_float8_param(kcxt,1);        /* constant 20.0 */
  pg_float8_t KVAR_8 = pg_float8_vref(kds,kcxt,7,kds_index);  /* var of X */
  pg_float8_t KVAR_9 = pg_float8_vref(kds,kcxt,8,kds_index);  /* var of Y */

  return EVAL(pgfn_boolop_and_2(kcxt,
              pgfn_float8ge(kcxt, KVAR_8, KVAR_9),
              pgfn_float8le(kcxt, KVAR_8,
                            pgfn_float8pl(kcxt, KVAR_9, KPARAM_1))));
}
                  :
```

↑ formula expression based on WHERE clause

SQL-to-GPU native code generation (3/3)



PG-Strom functionalities at Sep-2015

Logics

- GpuScan ... Simple loop extraction by GPU multithread
- GpuHashJoin ... GPU multithread based N-way hash-join
- GpuNestLoop ... GPU multithread based N-way nested-loop
- GpuPreAgg ... Row reduction prior to CPU aggregation
- GpuSort ... GPU bitonic + CPU merge, hybrid sorting

Data Types

- Numeric ... int2/4/8, float4/8, numeric
- Date and Time ... date, time(tz), timestamp(tz), interval
- Text ... simple matching only, at this moment
- others ... currency

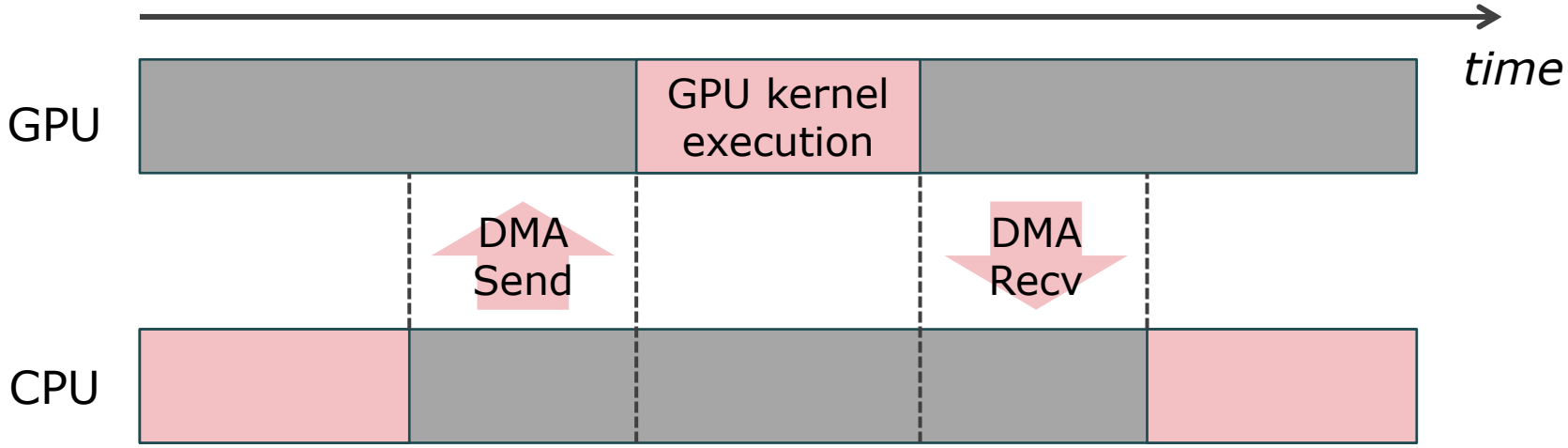
Functions

- Comparison operator ... <, <=, !=, =, >=, >
- Arithmetic operators ... +, -, *, /, %, ...
- Mathematical functions ... sqrt, log, exp, ...
- Aggregate functions ... min, max, sum, avg, stddev, ...

Agenda

1. GPU characteristics and why?
2. What intermediates PostgreSQL and GPU
3. **How GPU processes SQL workloads**
4. Performance results and analysis
5. Further development

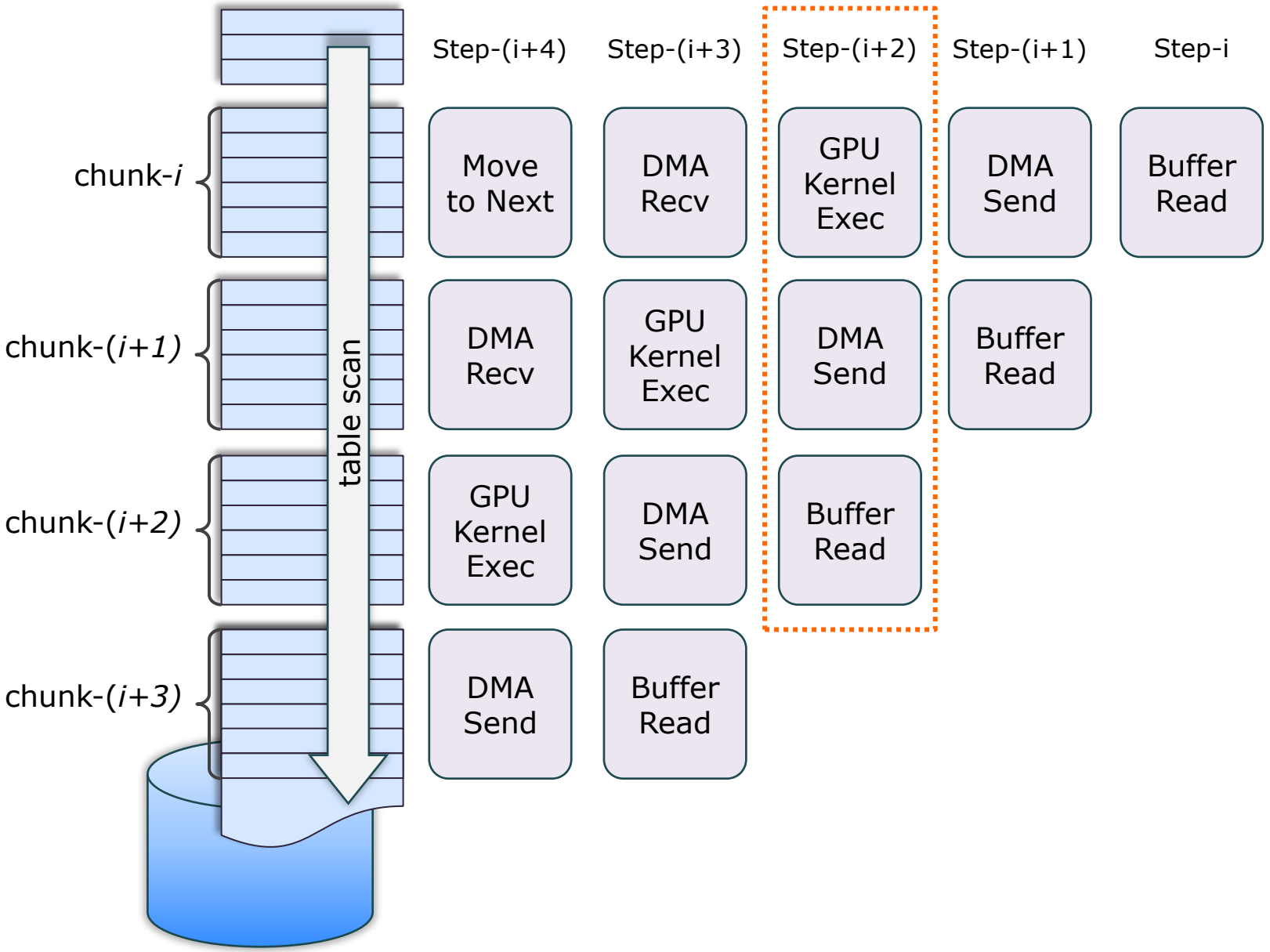
Asynchronous & Pipelining Execution (1/3)



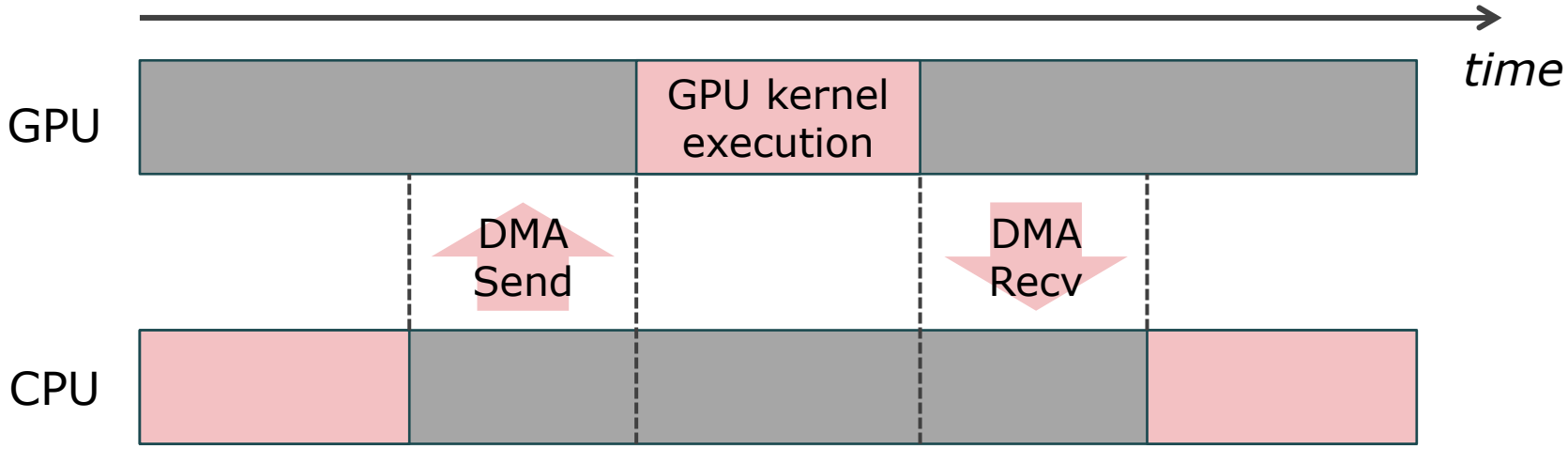
- Karma of discrete GPU device
  - We have to move the data to be processed.
  - Thus, people say data intensive tasks are not good for GPU.
  - Correct, but we can reduce the impact.

➔ Our solution: **Pipelining**

Asynchronous & Pipelining Execution (2/3)



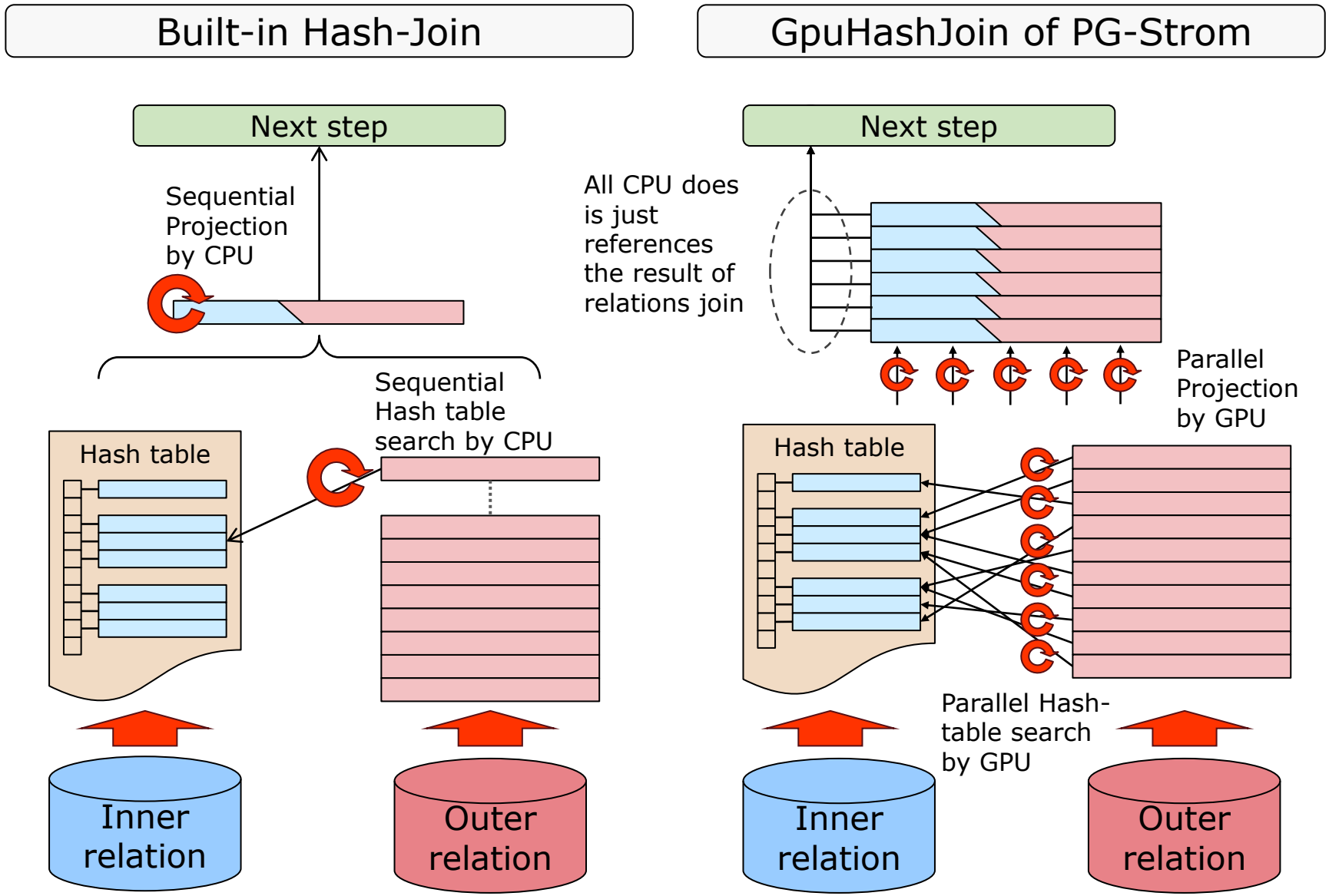
Asynchronous & Pipelining Execution (3/3)



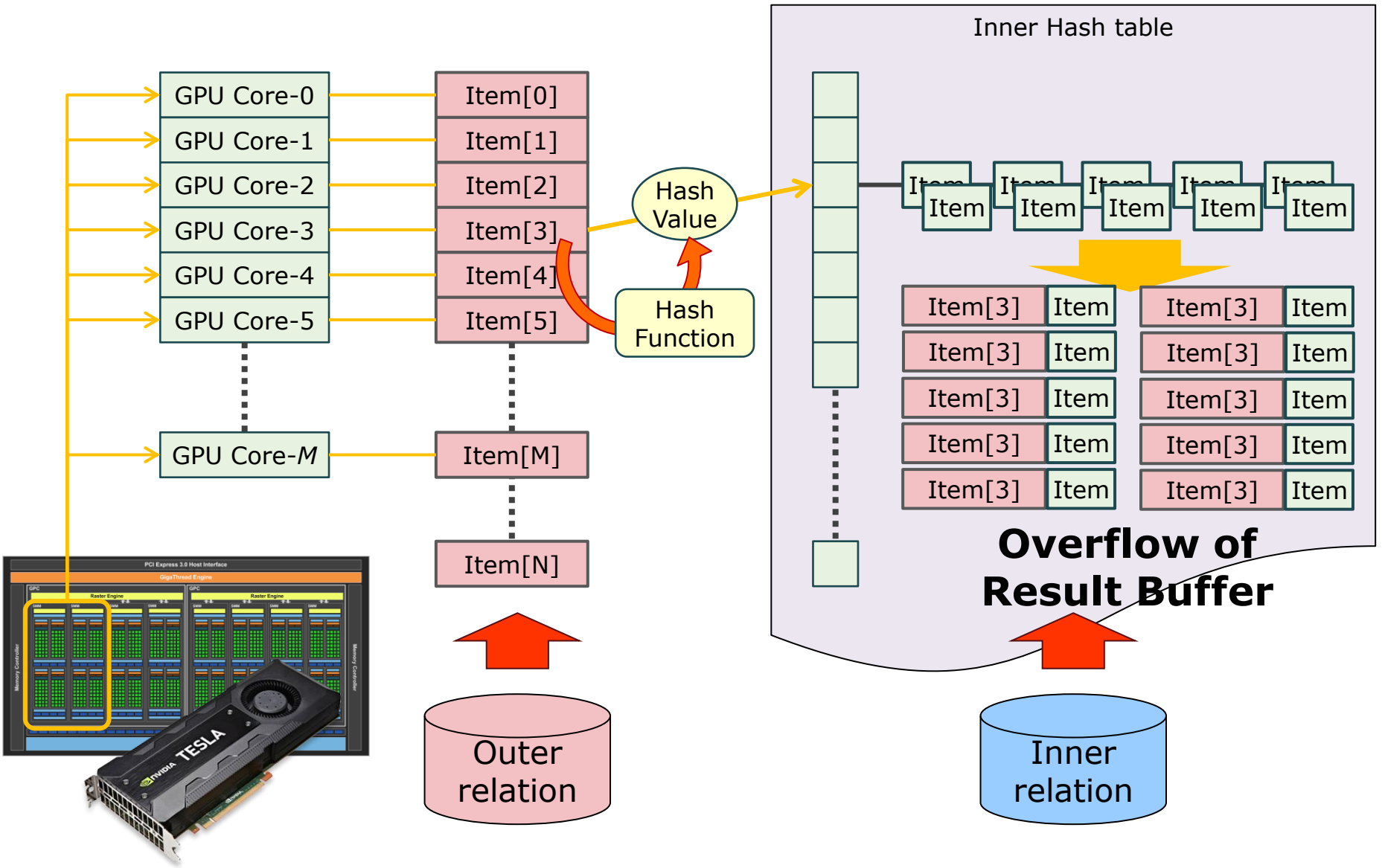
- Karma of discrete GPU device
  - We have to move the data to be processed.
  - Thus, people say data intensive tasks are not good for GPU.
  - Correct, but we can reduce the impact

  ➔ Our solution: **Pipelining**
- Downside of the pipelining
  - Too small workloads
  - Pipeline hazard

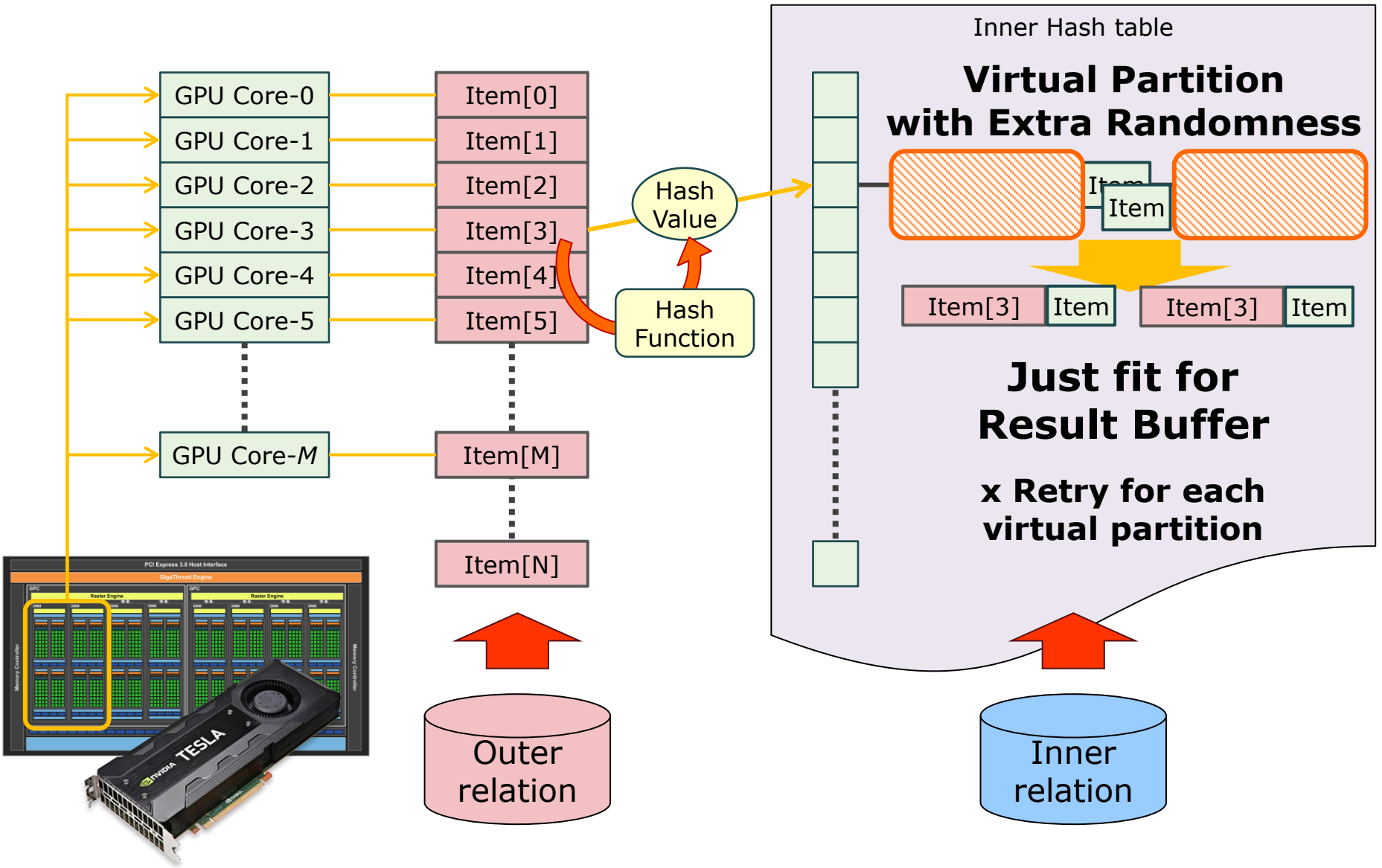
SQL Logic on GPU (1/3) – GpuHashJoin



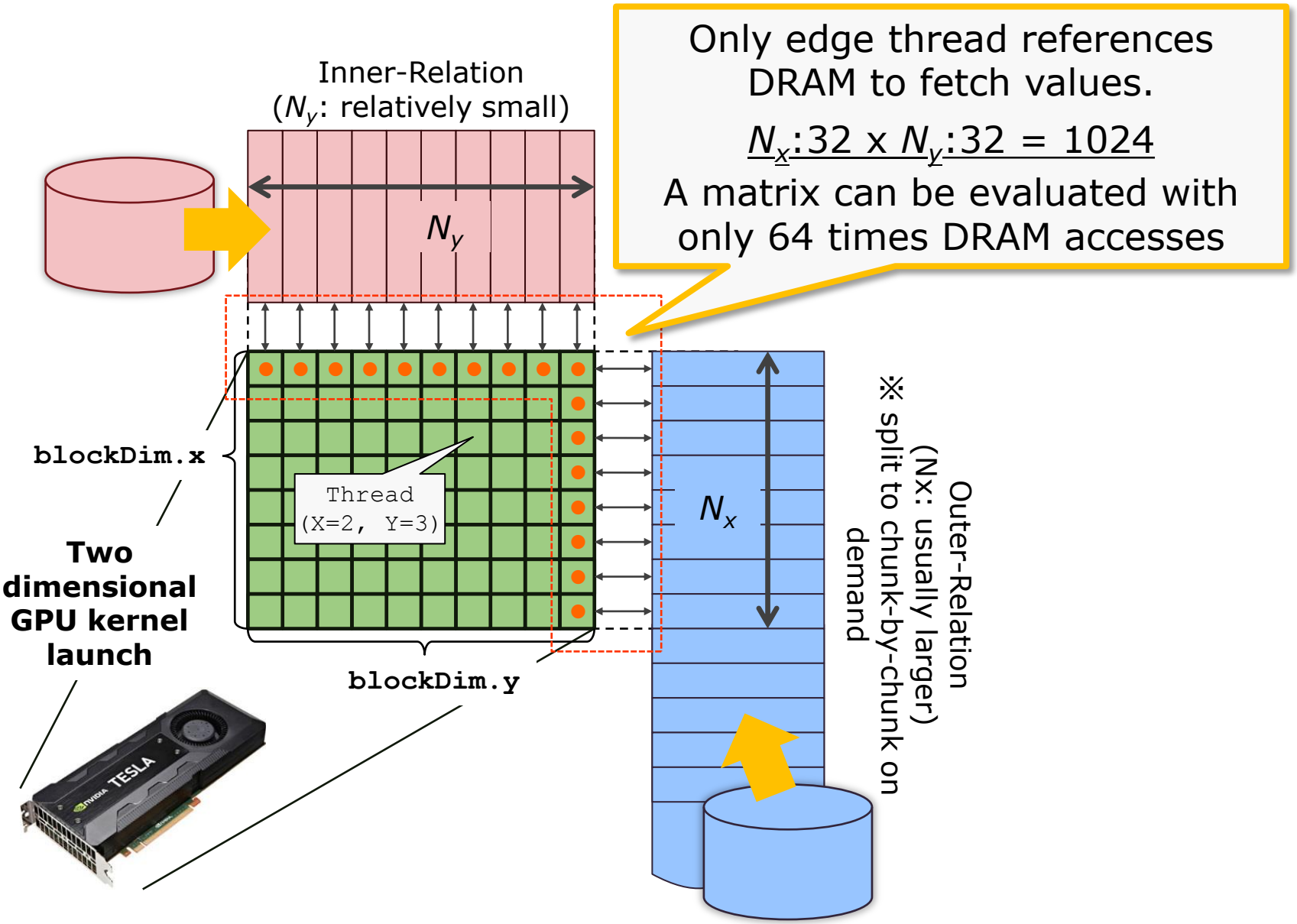
OT: Ununiformly Distributed Hash-Table



OT: Ununiformly Distributed Hash-Table

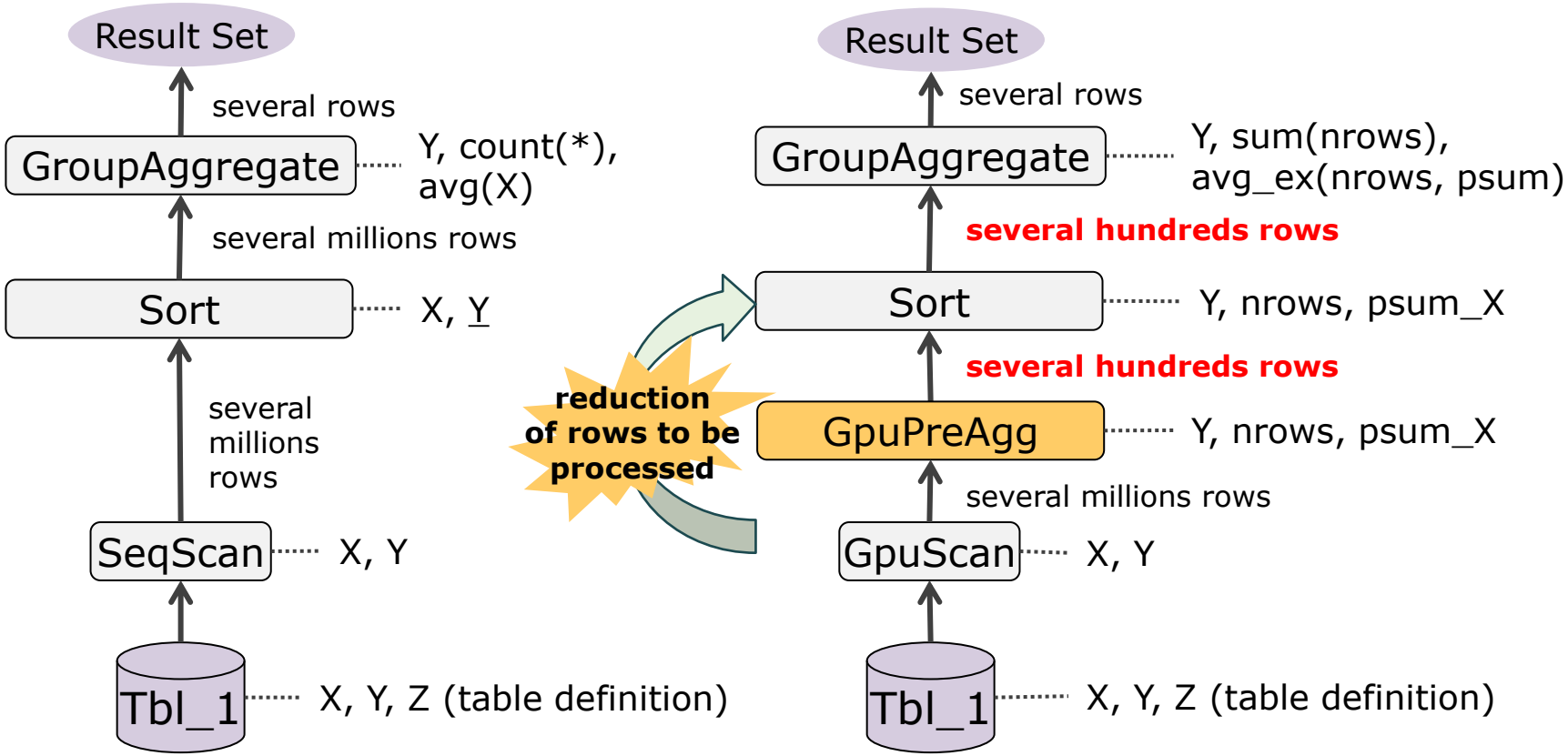


SQL Logic on GPU (2/3) – GpuNestLoop (unparametalized)



SQL Logic on GPU (3/3) – GpuPreAgg

SELECT count(*), AVG(X), Y FROM Tbl_1 GROUP BY Y;

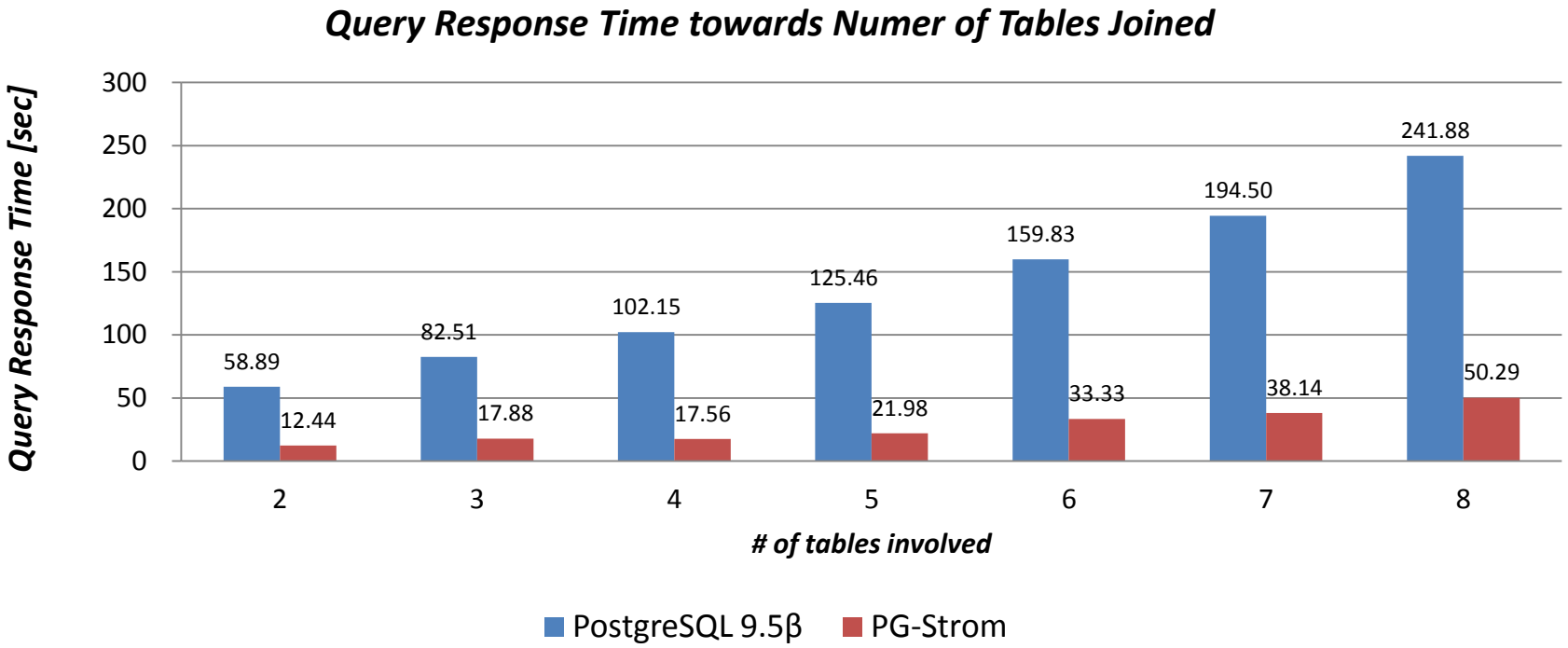


- GpuPreAgg, prior to Aggregate, reduces number of rows to be processed.
- Query rewrite to make equivalent results from intermediate aggregation.

Agenda

1. GPU characteristics and why?
2. What intermediates PostgreSQL and GPU
3. How GPU processes SQL workloads
4. **Performance results and analysis**
5. Further development

Microbenchmark (1/2) – total response time



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

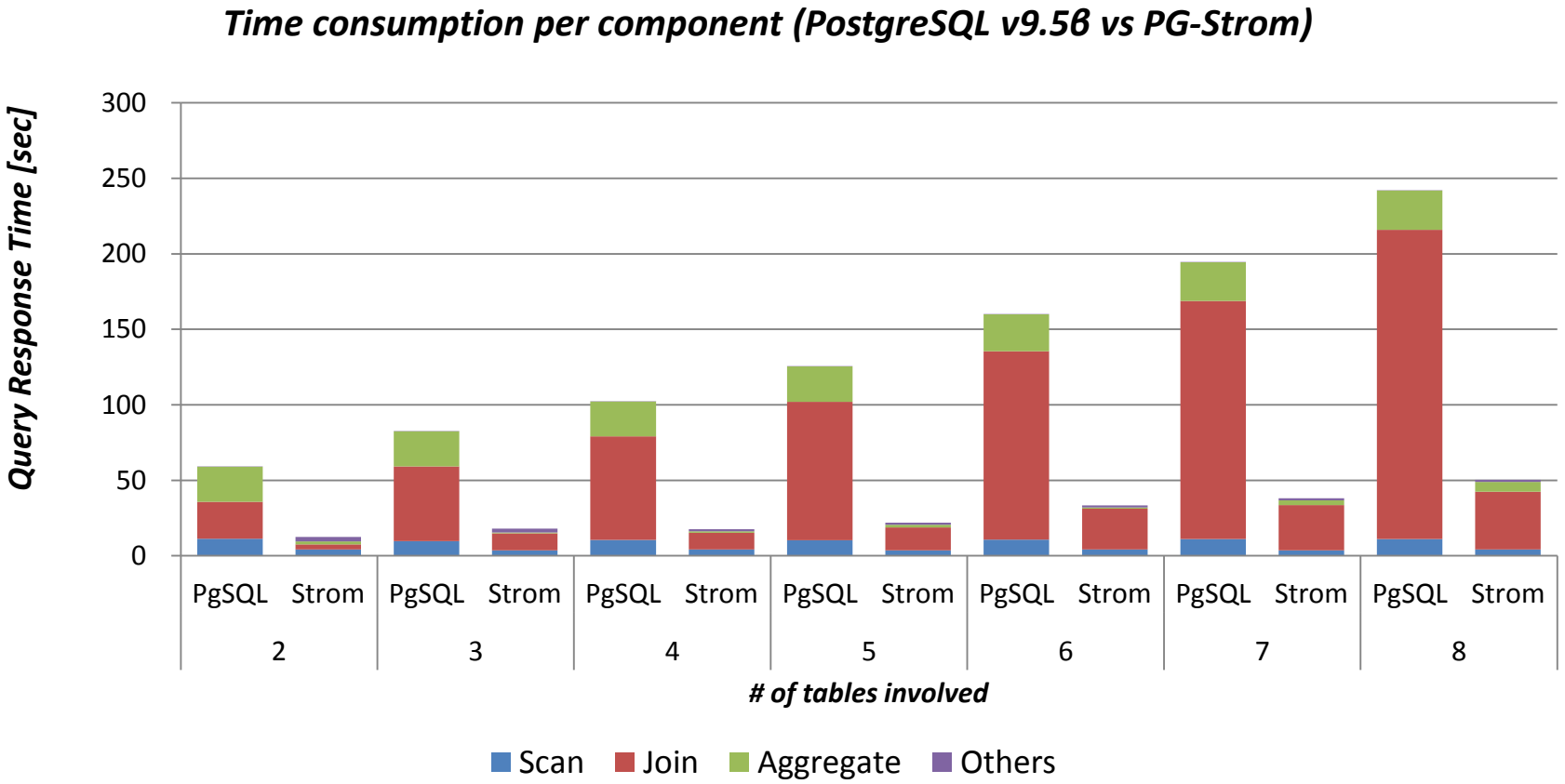
➔measurement of query response time with increasing of inner relations

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

Environment:

- PostgreSQL v9.5beta1 + PG-Strom (22-Oct), CUDA 7.0 + RHEL6.6 (x86_64)
- CPU: Xeon E5-2670v3, RAM: 384GB, GPU: NVIDIA TESLA K20c (2496cores)

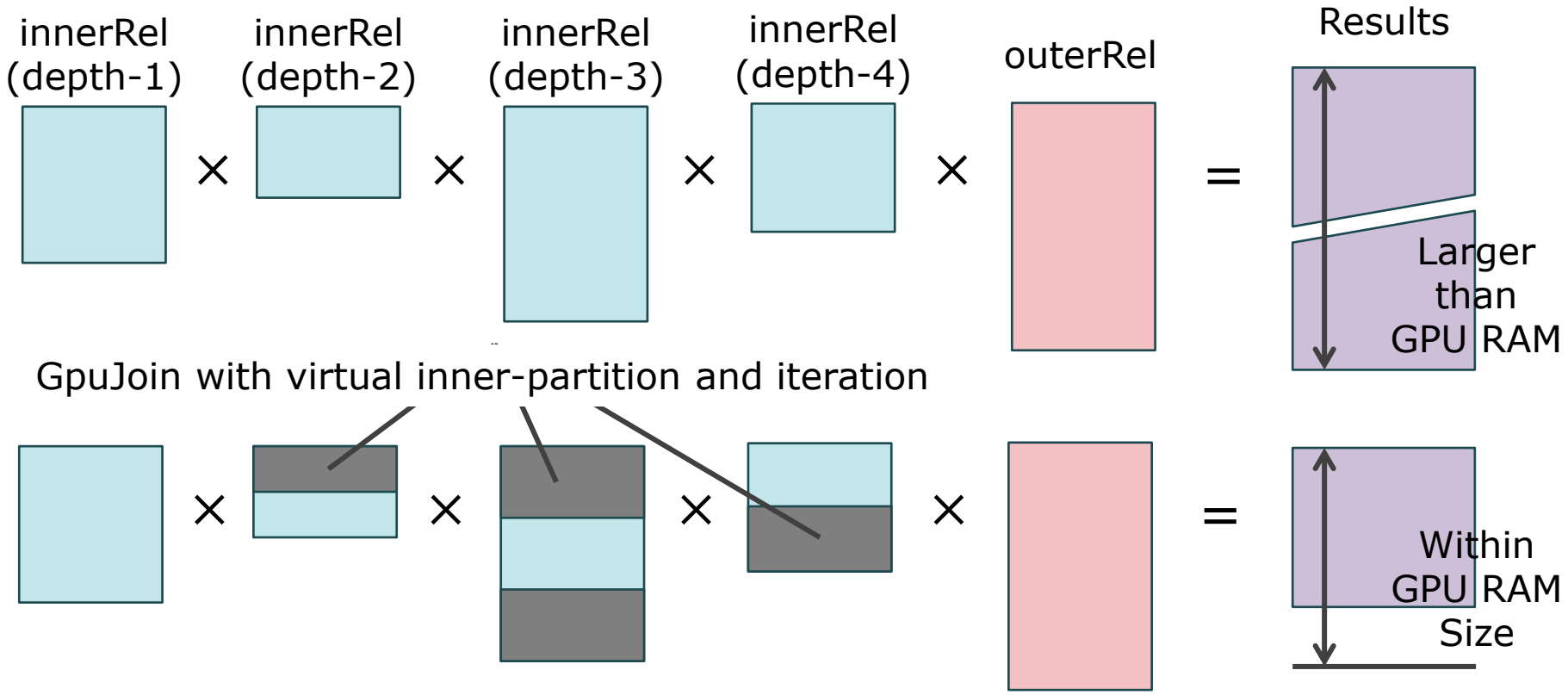
Microbenchmark (2/2) – Time consumption per component



- Good acceleration results towards JOIN and AGGREGATION
- Less acceleration ratio when N is larger than 6

  ➔By inaccurate problem size estimation, why?

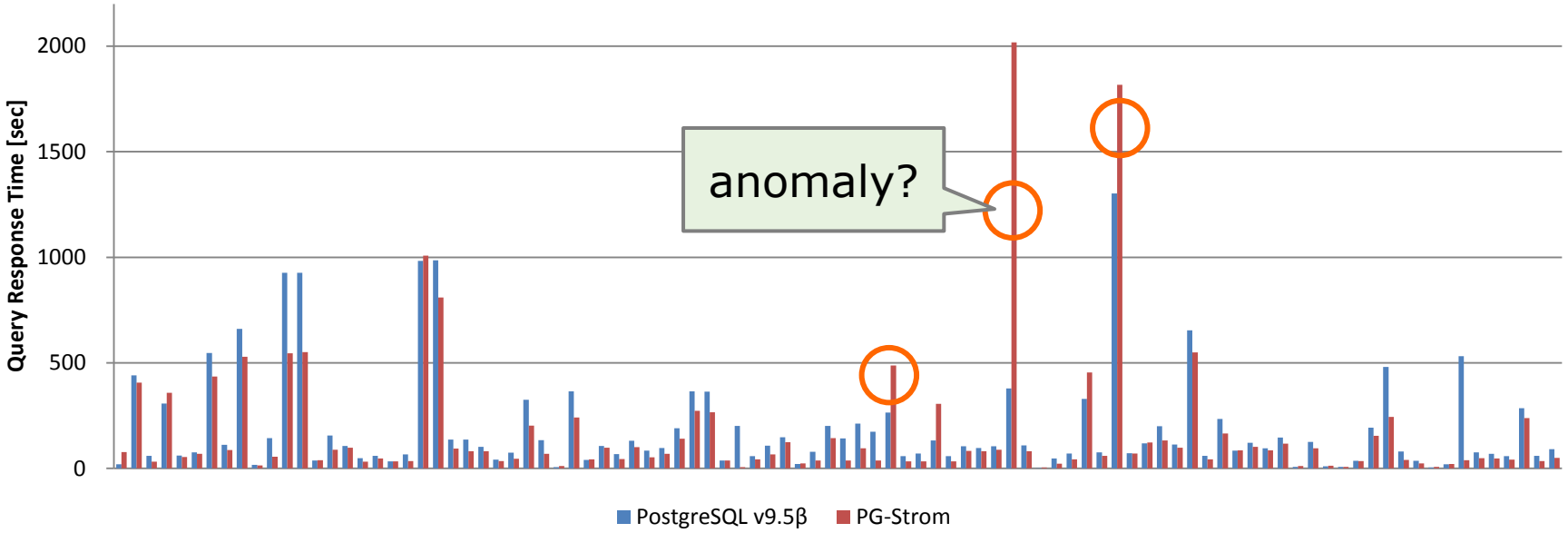
OT: Why accurate problem size estimation is important



GpuJoin results must be smaller than GPU RAM size.
Virtual inner-partition saves scale of the Join results.

- ➔ Better size estimation reduces number of retries.
- ➔ We can determine the problem size using **dynamic parallelism**
  downside: Supported by only Kepler or later generation.

TPC-DS Benchmark (1/2) – Total Results



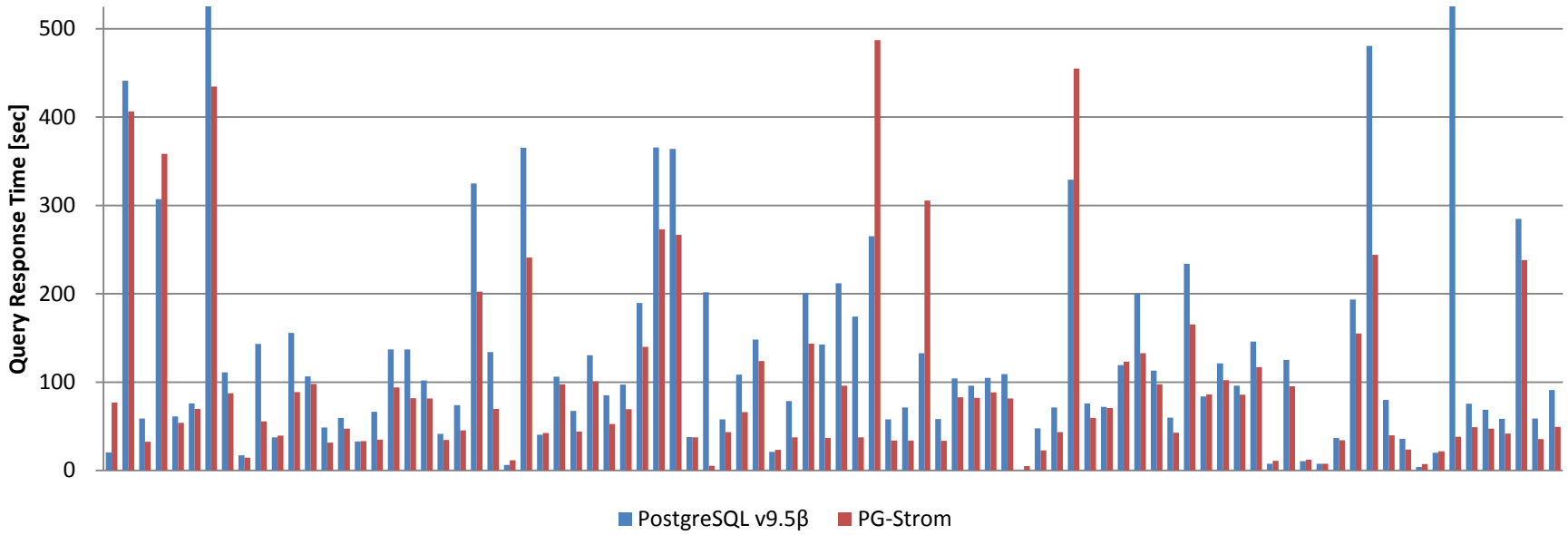
What is TPC-DS?

- Successor of TPC-H, defines 103 (99+4) kind of queries
- Simulation of decision support queries in retail industry

About this benchmark

- Run the series of TPC-DS queries sequentially.
- We could collect 96 results of 103.
- Environment:
  - PostgreSQL v9.5beta1 + PG-Strom (22-Oct), CUDA 7.0 + RHEL6.6 (x86_64)
  - CPU: Xeon E5-2670v3, RAM: 384GB, GPU: NVIDIA TESLA K20c (2496cores)

TPC-DS Benchmark (2/2) – Total Result



Summary

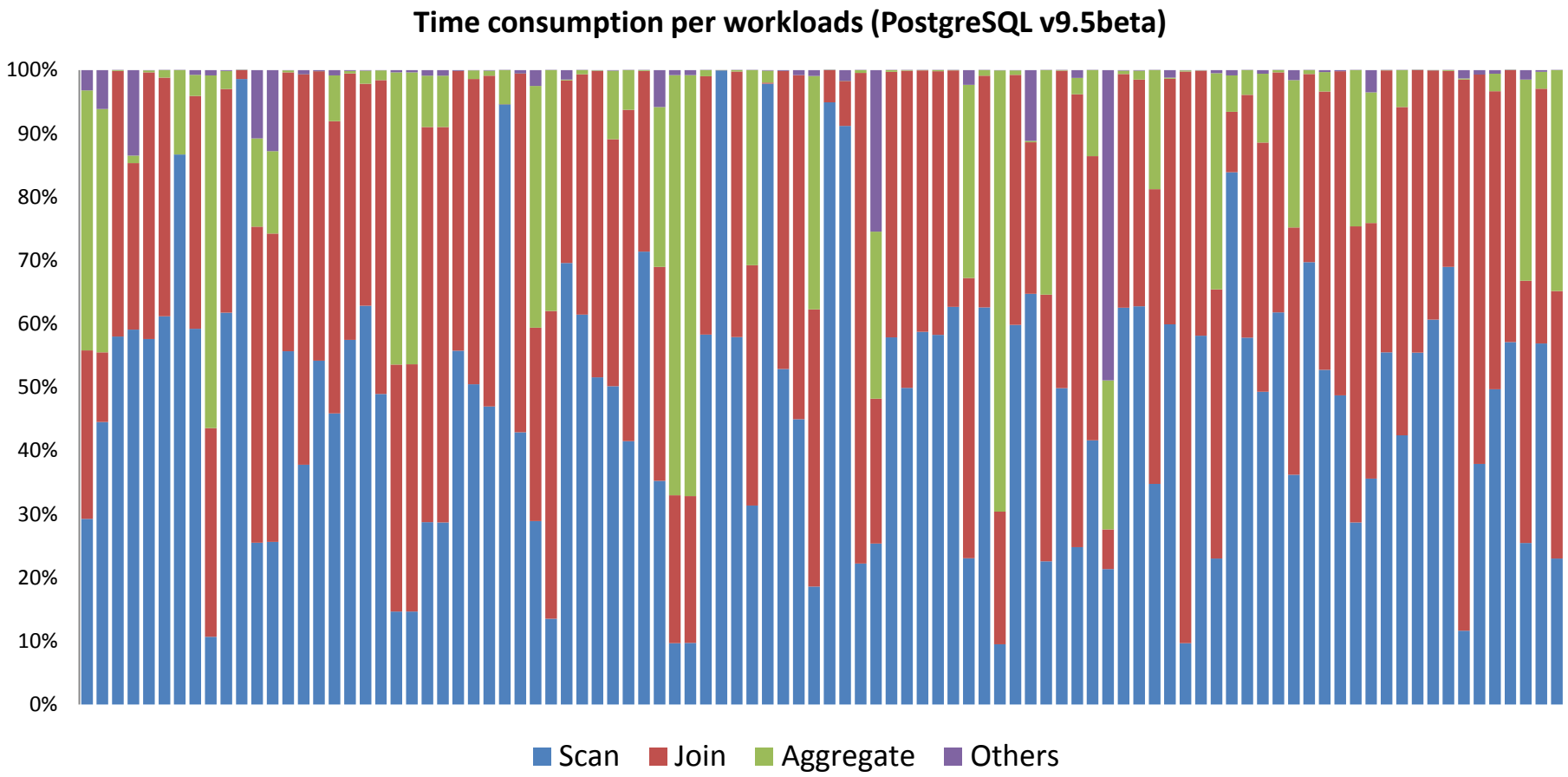
- PG-Strom wins to PostgreSQL: 76 of 96
- PostgreSQL wins to PG-Strom: 20 of 96

➔ shows advantages on most of simple queries

Challenges

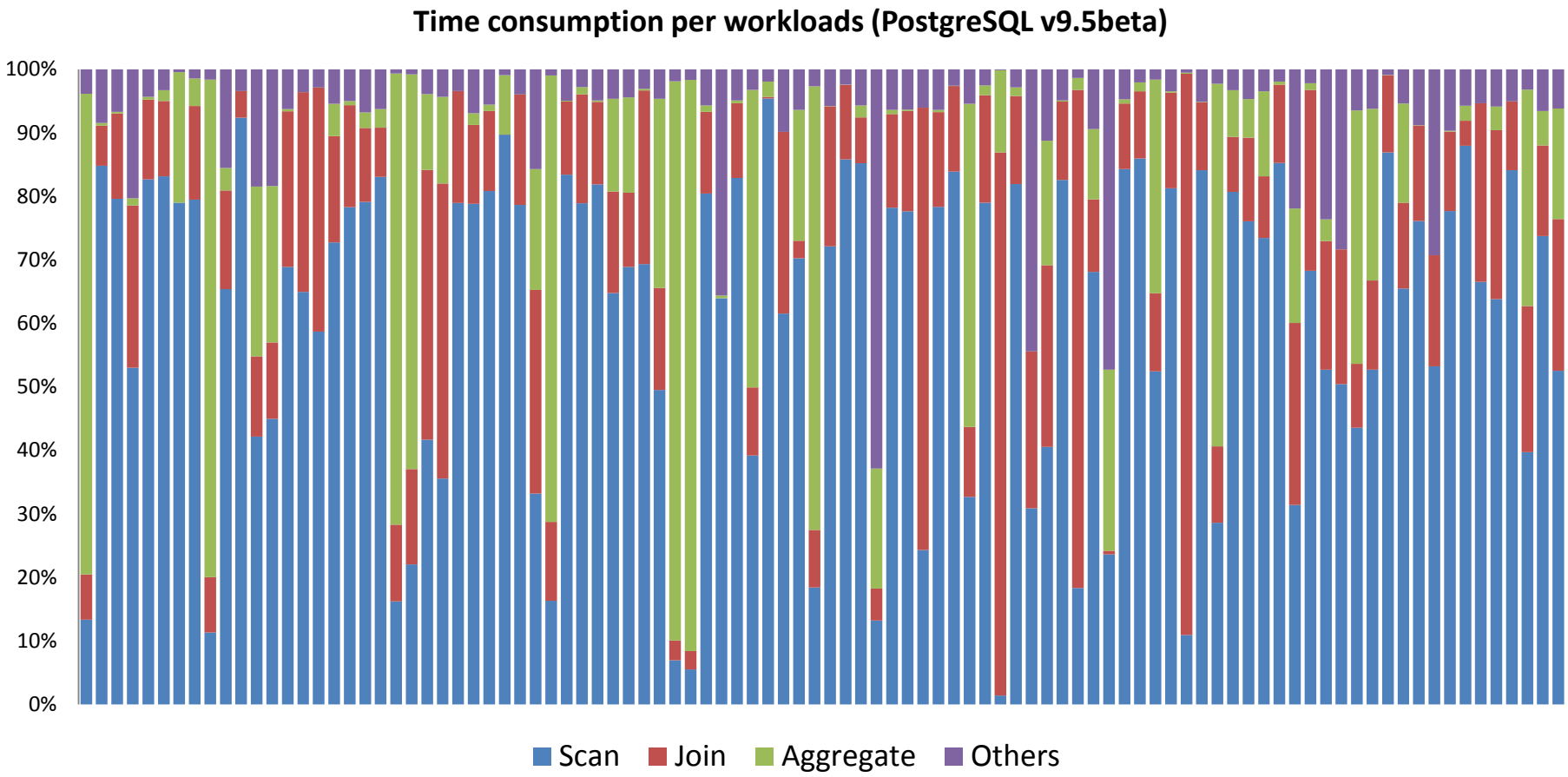
- Wise plan construction, to avoid GpuXXX node not to be here
- Functionality improvement for Aggregation and Scan

Time consumption per workloads (1/3) – PostgreSQL v9.5



- Scan, Join and Aggregation provides a clear explanation for the time consumption factor in TPC-DS workloads

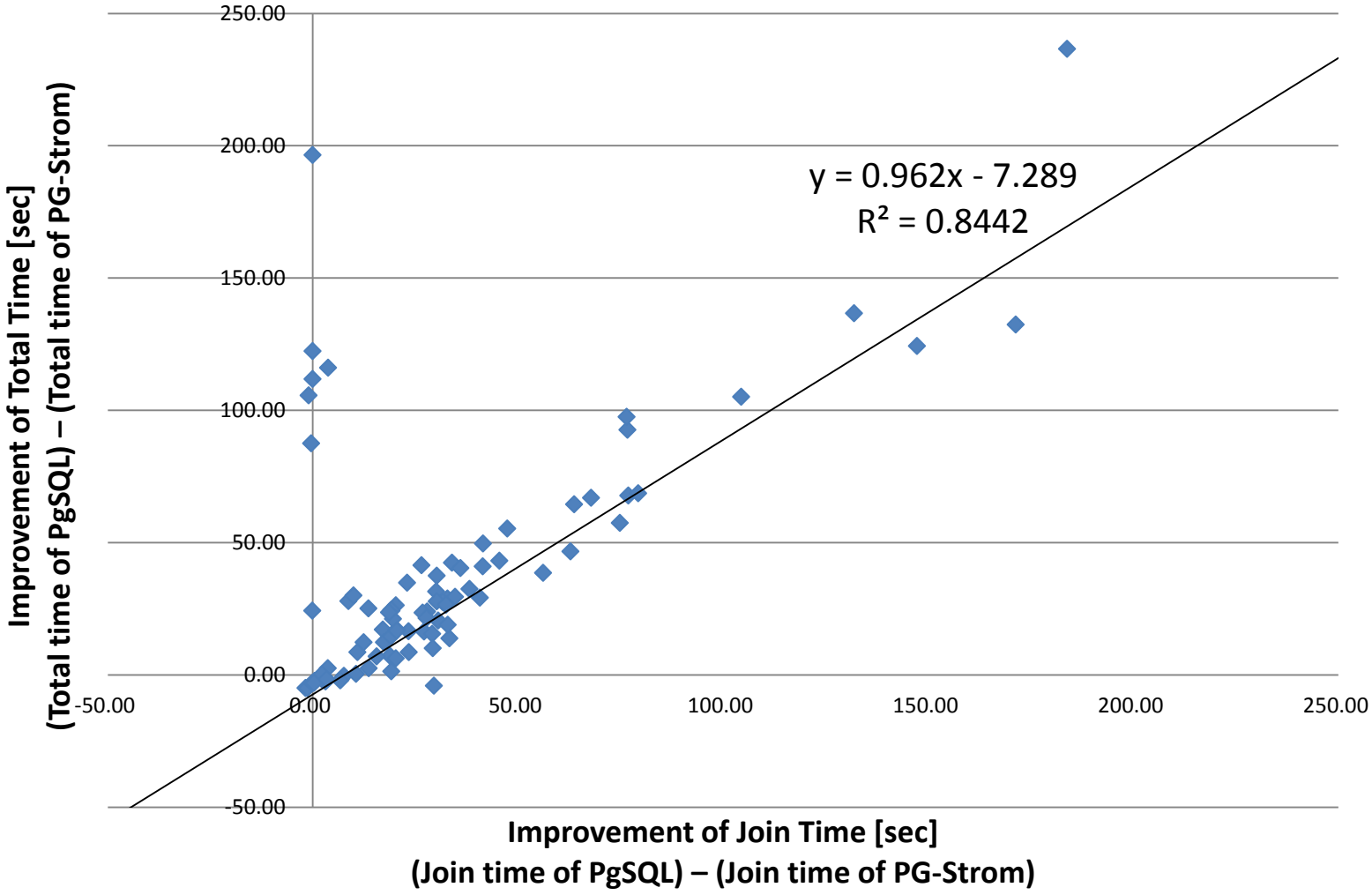
Time consumption per workloads (2/3) – PG-Strom



- PG-Strom dramatically reduced time for Joins.
- Time for Joins are: 37% ➔ 17%

Time consumption per workloads (3/3) – Wow! GpuJoin

GpuJoin can explain 84% of performance improvement



Agenda

1. GPU characteristics and why?
2. What intermediates PostgreSQL and GPU
3. How GPU processes SQL workloads
4. Performance results and analysis
5. **Further development**

What we learn from TPC-DS

GpuJoin may be a killer feature of PG-Strom

- ....unless problem size estimation is accurate.
- Planner needs to be wiser, not to inject GpuJoin towards the situation not suitable.

For Aggregation

- GpuPreAgg didn’t appear as we expected
- Anything GPU can help CPU aggregation?

For Scan

- Scan performance == data stream bandwidth of GPU input
- Single thread is a restriction for bandwidth of data supply

Features enhancement / improvement

Target-list calculation on GPU

- Pre-calculation of hash-value for HashAggregate
- Projection off-load if complicated mathematical expression

Custom-Scan under Gather node

- Allows multiple workers to provide data stream to GPUs

Partial Aggregation before Join

- Wider utilization of GpuPreAgg to reduce number of rows to be joined by CPU

Utilization of dynamic parallelism

- Wise approach towards result buffer overflow problem in Join

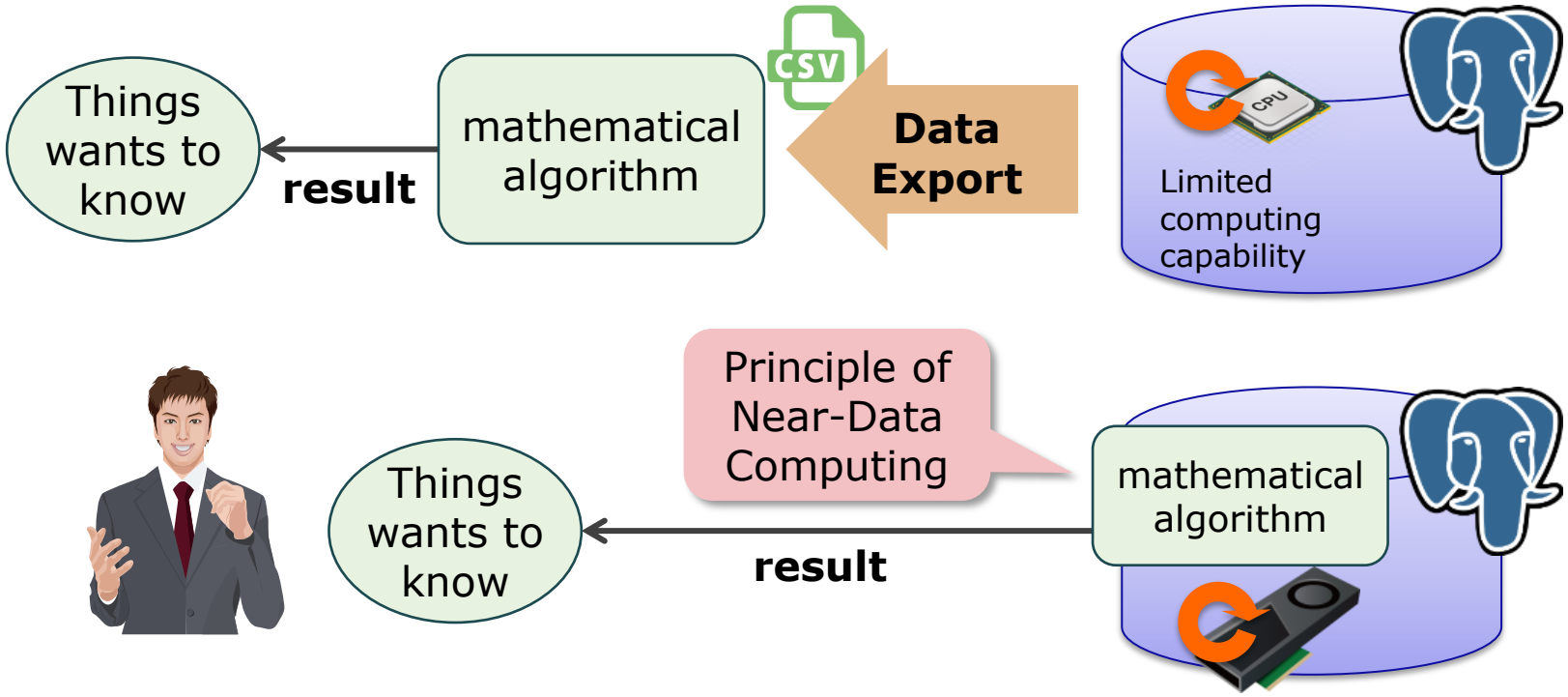
... and more...

Short term development

1. Software quality improvement
2. Corner case handling
3. Documentation
4. Target-List calculation in GPU
5. Software quality improvement

WIP Project – In-place Computing

Why export entire data to run mathematical algorithm?



- Runs complicated mathematical expression on GPUs, then CPU just references the result
  - Algorithms in SQL (even partially) automatically moves to GPUs
  - WIP: Data-Clustering algorithms, like Min-Max, K-means, ...

Future Direction of PG-Strom

Real-life workloads are the only compass for development

WE WANT YOU GET INVOLVED

PostgreSQL Users

- who concern about query execution performance on your workloads

Application Developers

- who are interested in trying yet another database acceleration methodology

PostgreSQL Developers

- who can bet the future of heterogeneous era

Resources

Github

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

Documentation

- Under construction.... sorry

Today's slides

- Uploaded soon, check #pgconfeu on Tw

Author's contact

- e-mail: kaigai@ak.jp.nec.com
- Tw: @kkaigai

Orchestrating a brighter world