Empowered by Innovation NEC

GPGPU Accelerates PostgreSQL

NEC OSS Promotion Center
The PG-Strom Project
KaiGai Kohei <kaigai@ak.jp.nec.com>
(Tw: @kkaigai)

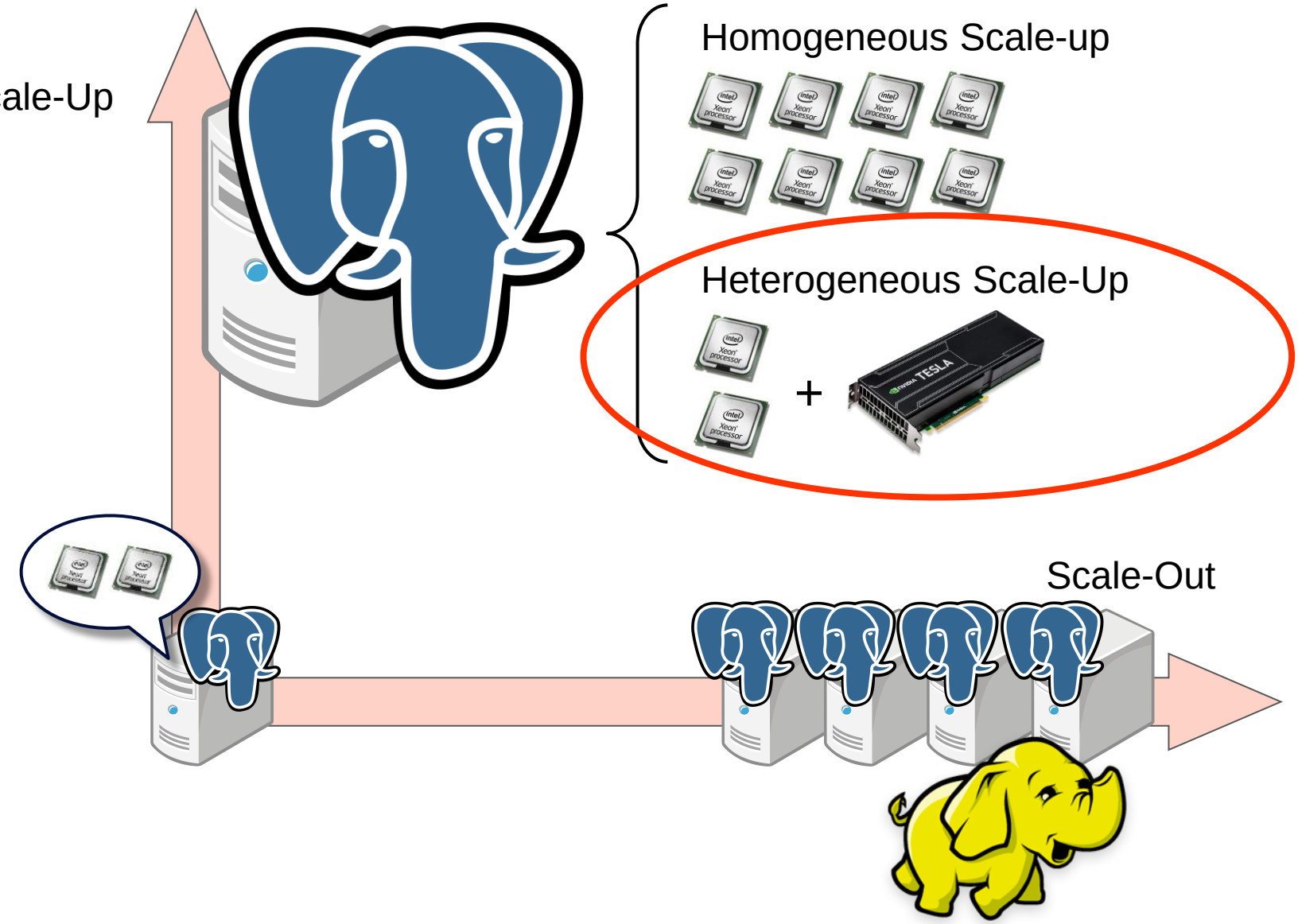
Self Introduction

- Name: KaiGai Kohei
- Company: NEC OSS Promotion Center
- Like: Processor with many cores
- Dislike: Processor with little cores
- Background:
  HPC ➔ OSS/Linux ➔ SAP ➔ GPU/PostgreSQL
- Tw: @kkaigai

My Jobs

- SELinux (2004~)
  - Lockless AVC、JFFS2 XATTR, ...
- PostgreSQL (2006~)
  - SE-PostgreSQL, Security Barrier View, Writable FDW, ...
- PG-Strom (2012~)

Approach for Performance Improvement



Characteristics of GPU (Graphic Processor Unit)

SOURCE: CUDA C Programming Guide (v6.5)

	GPU	CPU
Model	Nvidia Tesla K20X	Intel Xeon E5-2670 v3
Architecture	Kepler	Haswell
Launch	Nov-2012	Sep-2014
# of transistors	7.1billion	3.84billion
# of cores	2688 (simple)	12 (functional)
Core clock	732MHz	2.6GHz, up to 3.5GHz
Peak Flops (single precision)	3.95TFLOPS	998.4GFLOPS (with AVX2)
DRAM size	6GB, GDDR5	768GB/socket, DDR4
Memory band	250GB/s	68GB/s
Power consumption	235W	135W
Price	$3,000	$2,094

Characteristics

- Larger percentage of ALUs on chip
- Relatively smaller percentage of cache and control logic

➔ Advantages to simple calculation in parallel, but not complicated logic

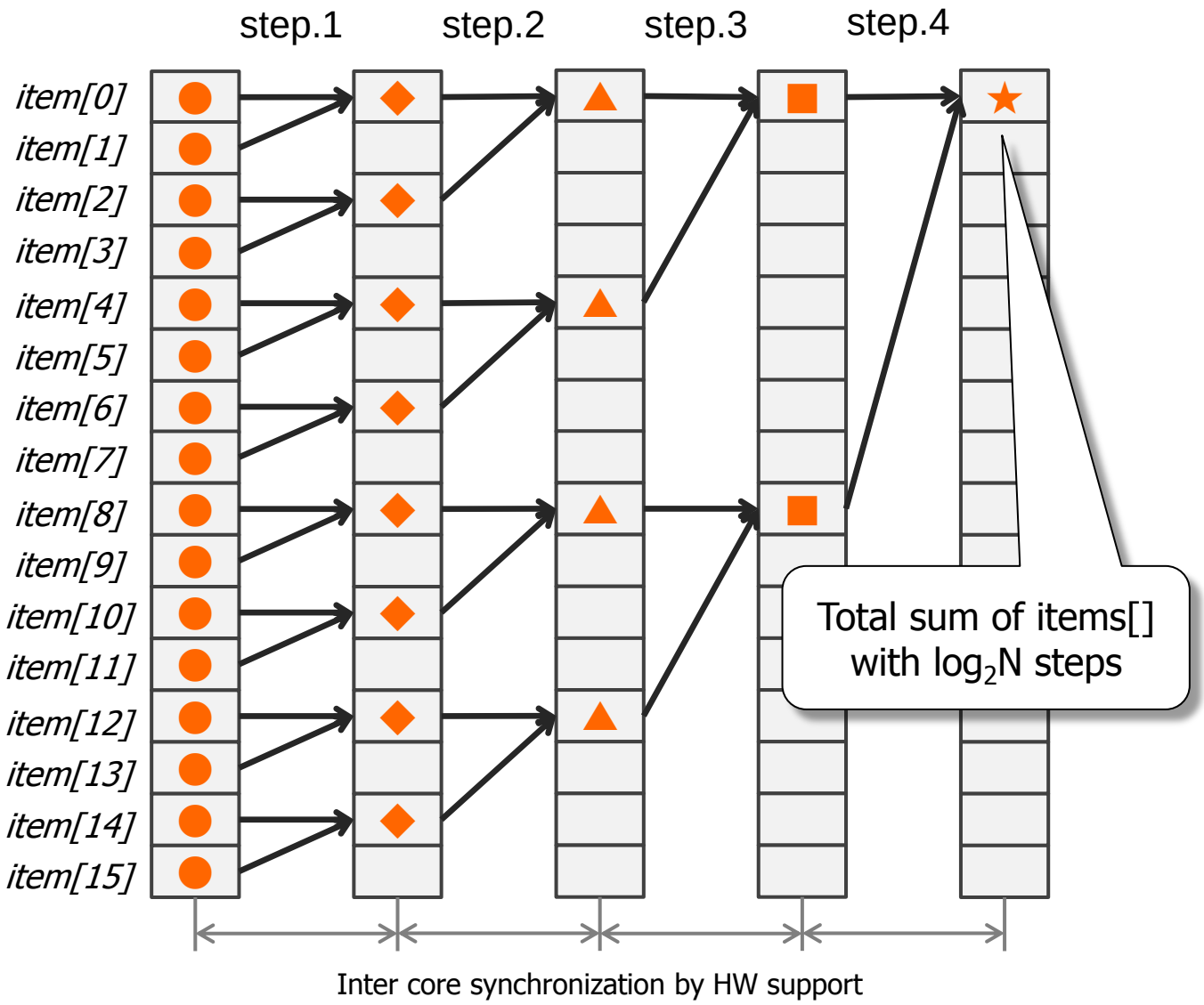
- Much higher number of cores per price
  - GTX750Ti (640core) with $150

How GPU works – Example of reduction algorithm

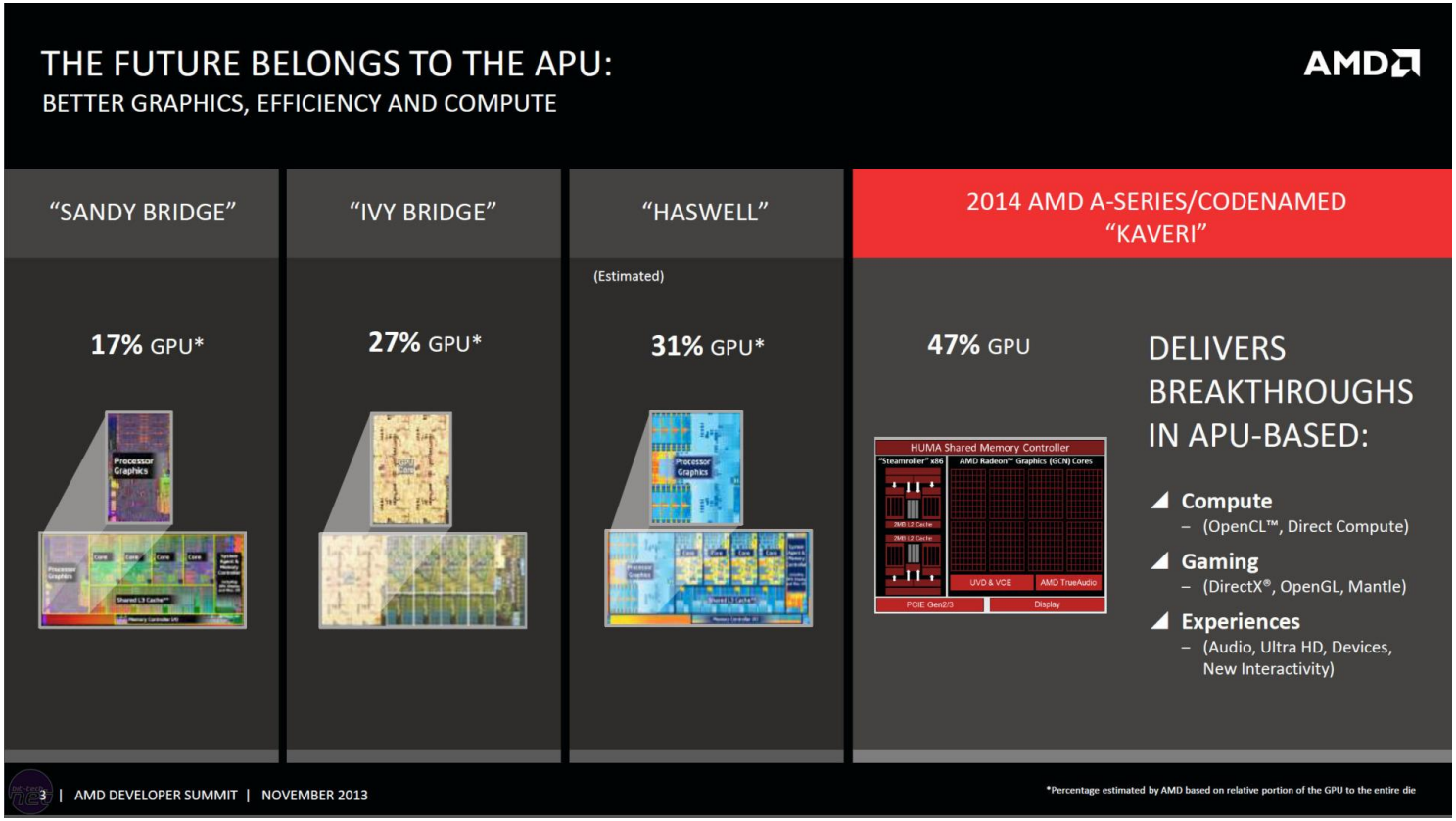
Computing
the sum of array:

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

with N-cores of GPU



Semiconductor Trend (1/2) – Towards Heterogeneous

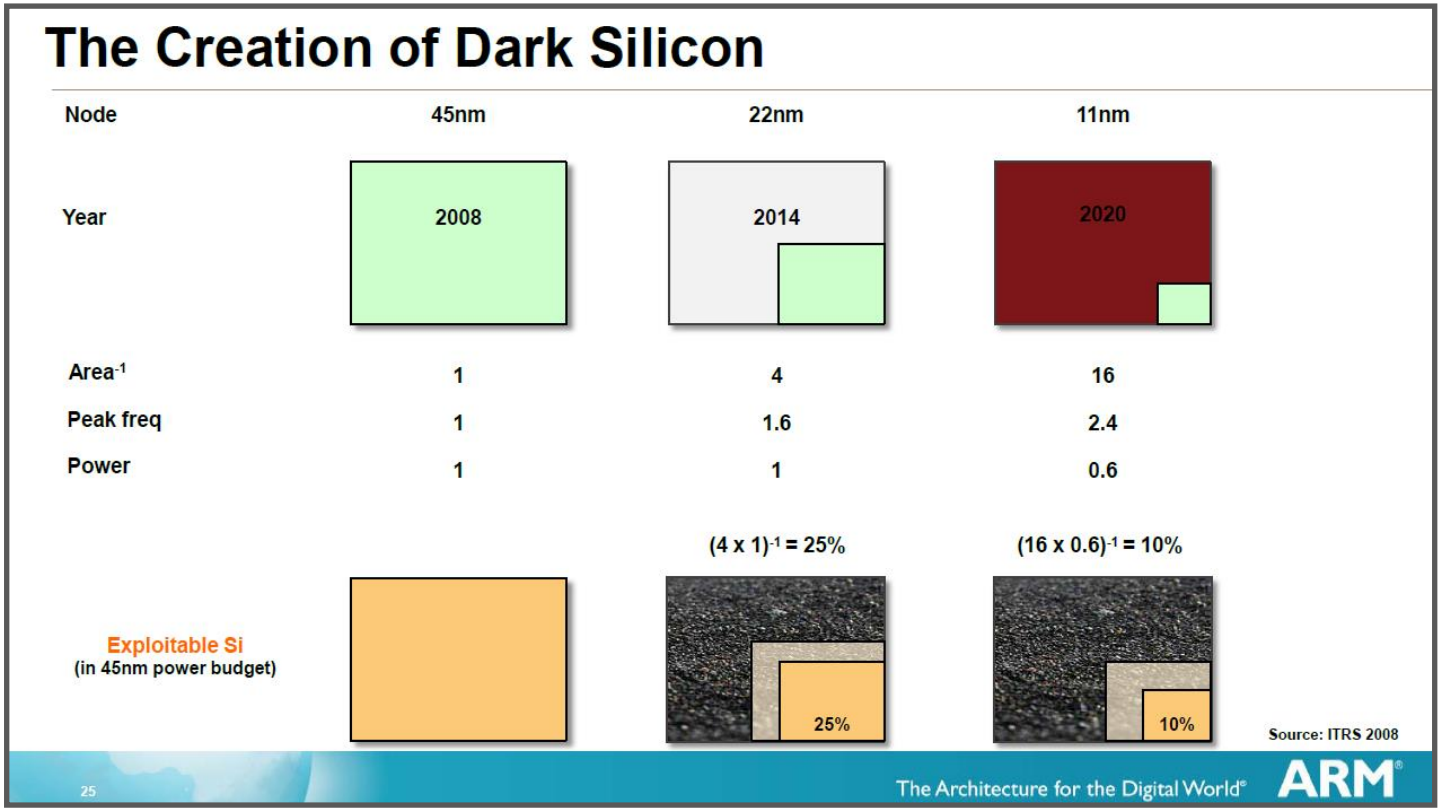


SOURCE: THE HEART OF AMD INNOVATION, Lisa Su, at AMD Developer Summit 2013

- Moves to CPU/GPU integrated architecture from multicore CPU
- Free lunch for SW by HW improvement will finish soon

➔ No utilization of semiconductor capability, unless SW is not designed with conscious of HW characteristics.

Semiconductor Trend (2/2) – Dark Silicon Problem



SOURCE: Compute Power with Energy-Efficiency, Jem Davies, at AMD Fusion Developer Summit 2011

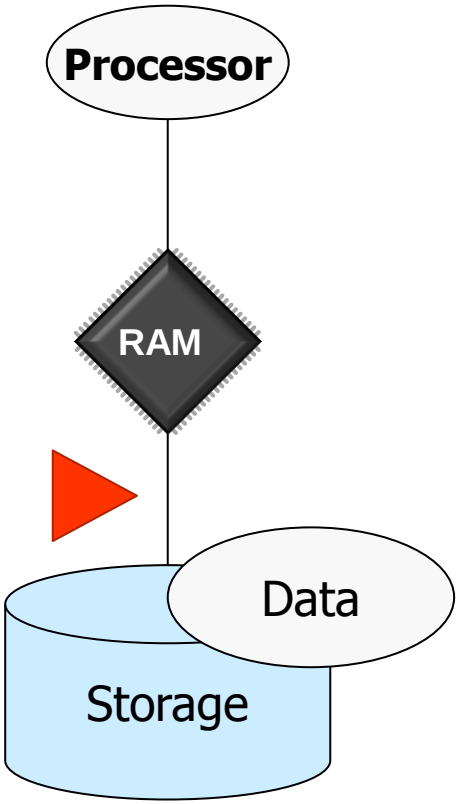
Background of CPU/GPU integrated architecture

- Increase of transistor density > Reduction of power consumption
- Unable to supply power for all the logic because of chip cooling

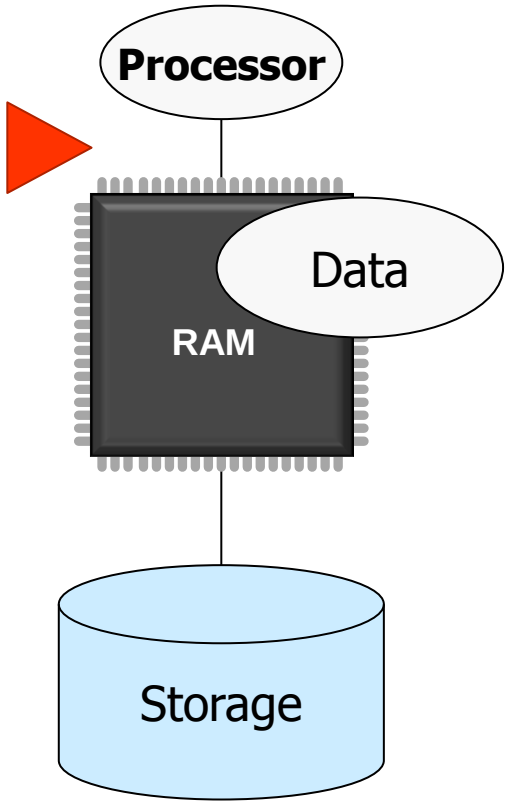
➔ A chip has multiple logics with different features, to save the peak power consumption.

RDBMS and its bottleneck (1/2)

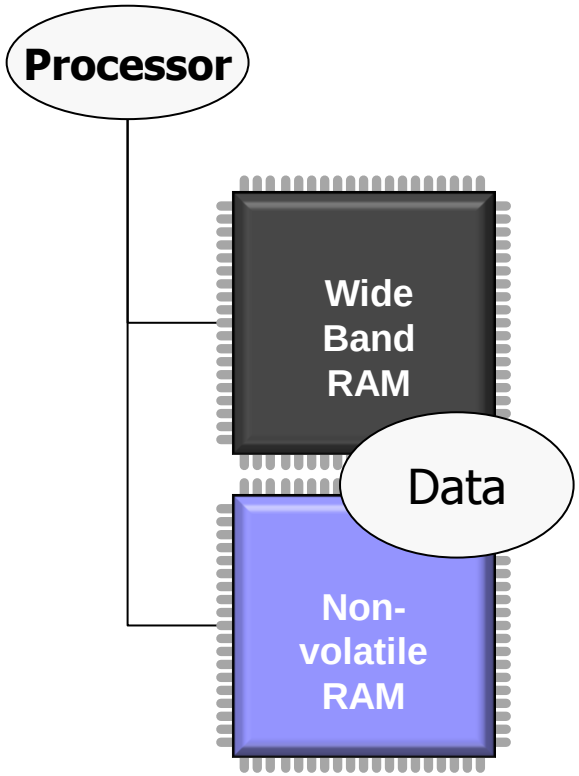
Data Size > RAM



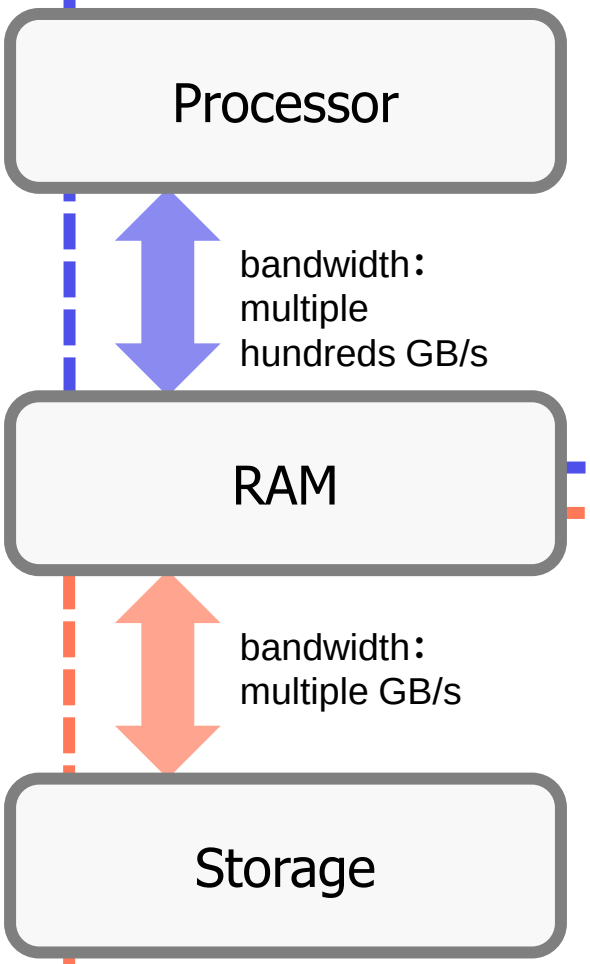
Data Size < RAM



In the future?



RDBMS and its bottleneck (2/2)



World of current memory bottleneck

Join, Aggregation, Sort, Projection, ...

[strategy]

- burstable access pattern
- parallel algorithm

World of traditional disk-i/o bottleneck

SeqScan, IndexScan, ...

[strategy]

- reduction of i/o (size, count)
- distribution of disk (RAID)

PG-Strom

What is PG-Strom

- An extension designed for PostgreSQL
- Off-loads a part of SQL workloads to GPU for parallel/rapid execution
- Three workloads are supported: Full-Scan, Hash-Join, Aggregate

(At the moment of Nov-2014, beta version)

Concept

- Automatic GPU native code generation from SQL query, by JIT compile
- Asynchronous execution with CPU/GPU co-operation.

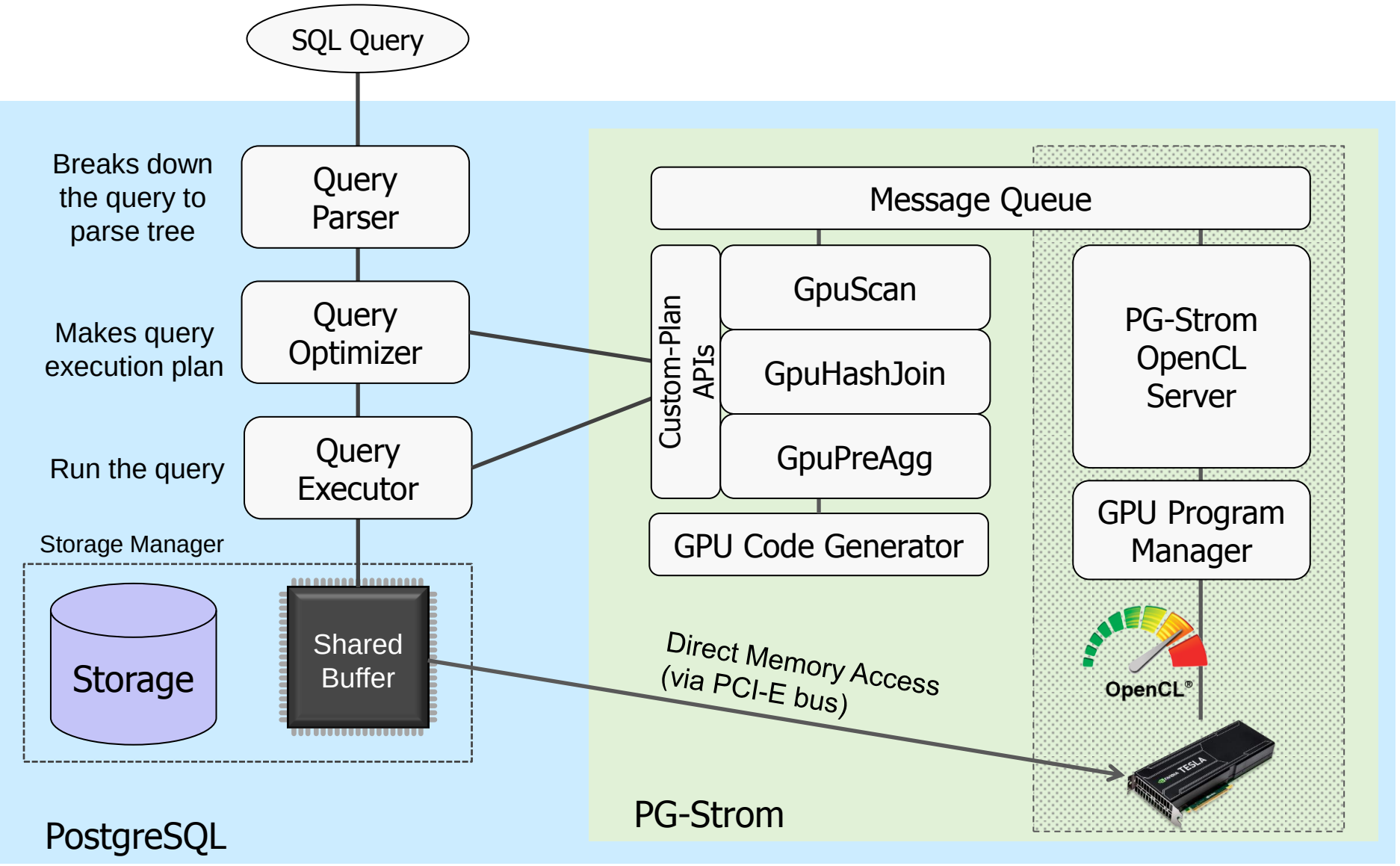
Advantage

- Works fully transparently from users
  - It allows to use peripheral software of PostgreSQL, including SQL syntax, backup, HA or drivers.
- Performance improvement with GPU+OSS; very low cost

Attention

- Right now, in-memory data store is assumed

Architecture of PG-Strom (1/2)



Architecture of PG-Strom (2/2)

Elemental technology① OpenCL

- Parallel computing framework on heterogeneous processors
- Can use for CPU parallel, not only GPU of NVIDIA/AMD
- Includes run-time compiler in the language specification

Elemental technology② Custom-Scan Interface

- Feature to implement scan/join by extensions, as if it is built-in logic for SQL processing in PostgreSQL.
- A part of functionalities got merged to v9.5, discussions are in-progress for full functionalities.

Elemental technology③ Row oriented data structure

- Shared buffer of PostgreSQL as DMA source
- Data format translation between row and column, even though column format is optimal for GPU performance.

Elemental technology① OpenCL (1/2)

Characteristics of OpenCL

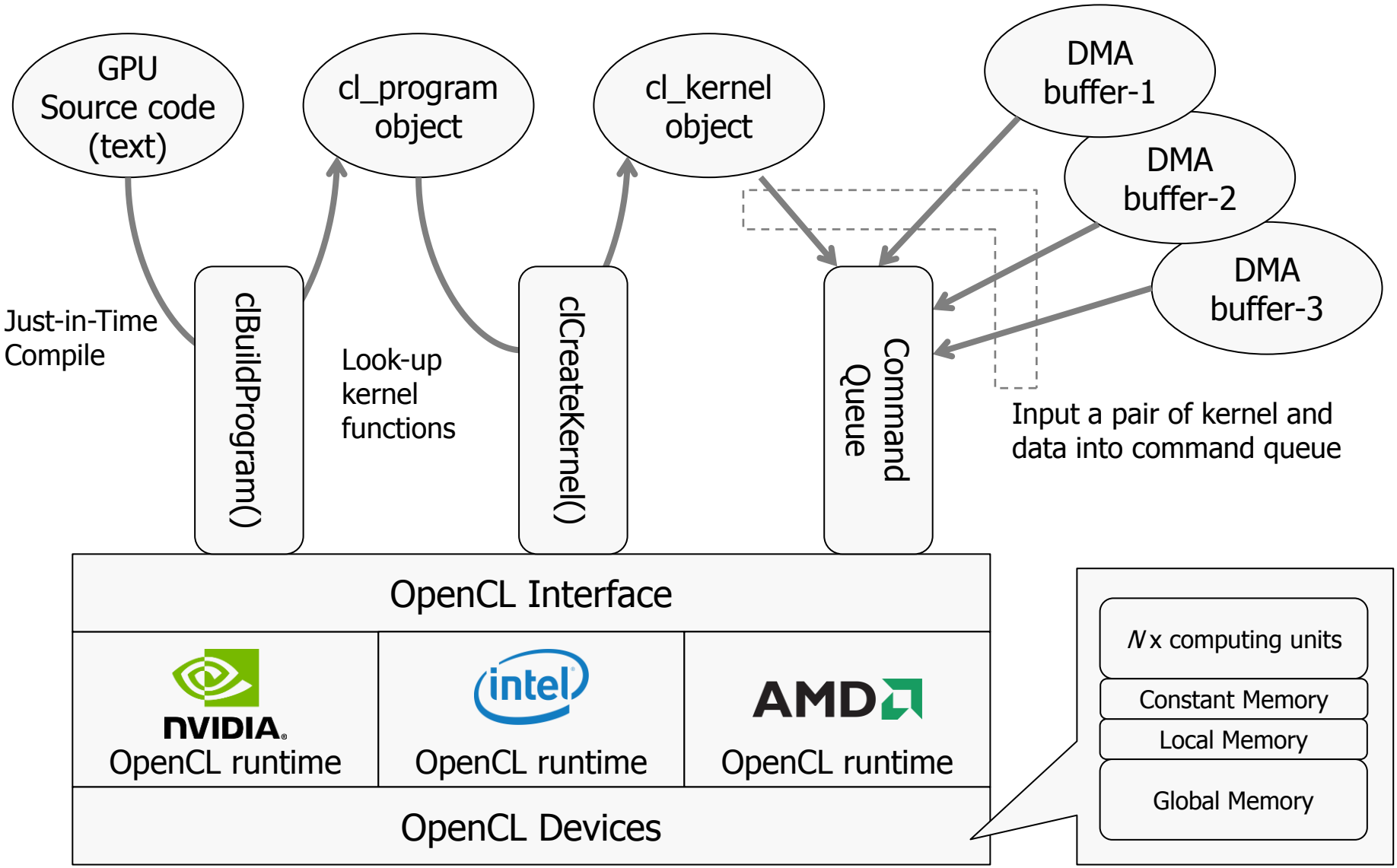
- Use abstracted "OpenCL device" for CPU/MIC, not only GPUs
  - Even though it follows characteristics of GPUs below....
  - ✓Three types of memory layer (global, local, constant)
  - ✓Concept of workgroup; synchronous execution inter-threads
- Just-in-time compile from source code like C-language to the platform specific native code

Comparison with CUDA

	CUDA	OpenCL
Advantage	• Detailed optimization • Latest feature of NVIDIA GPU • Driver stability	• Multiplatform support • Built-in JIT compiler • CPU parallel support
Issues	• Unavailable on AMD, Intel	• Driver stability

➔ We don't need to develop JIT compile functionality by ourselves, and want more people to try, so adopted OpenCL at this moment.

Elemental technology① OpenCL (2/2)

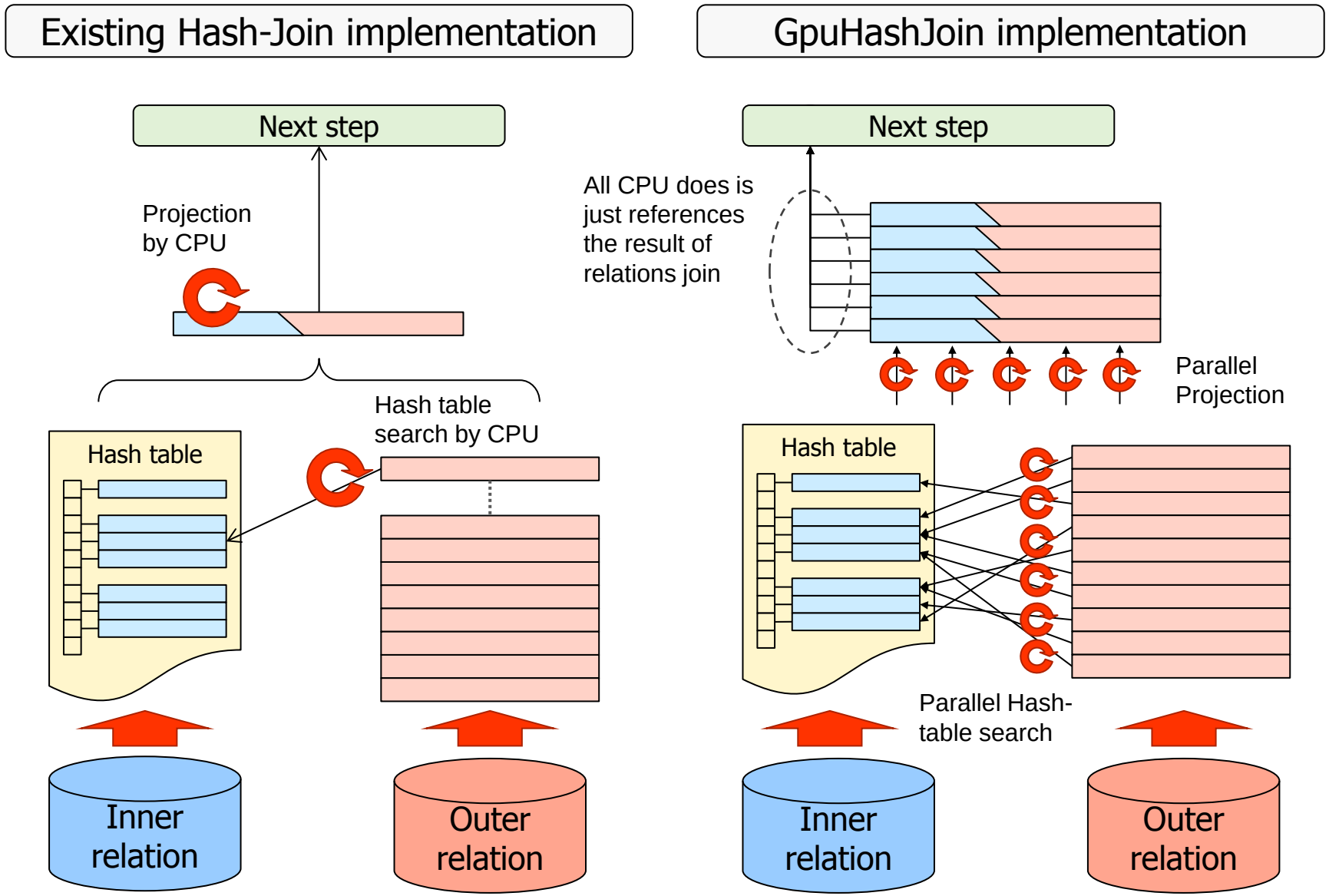


Automatic GPU native code generation

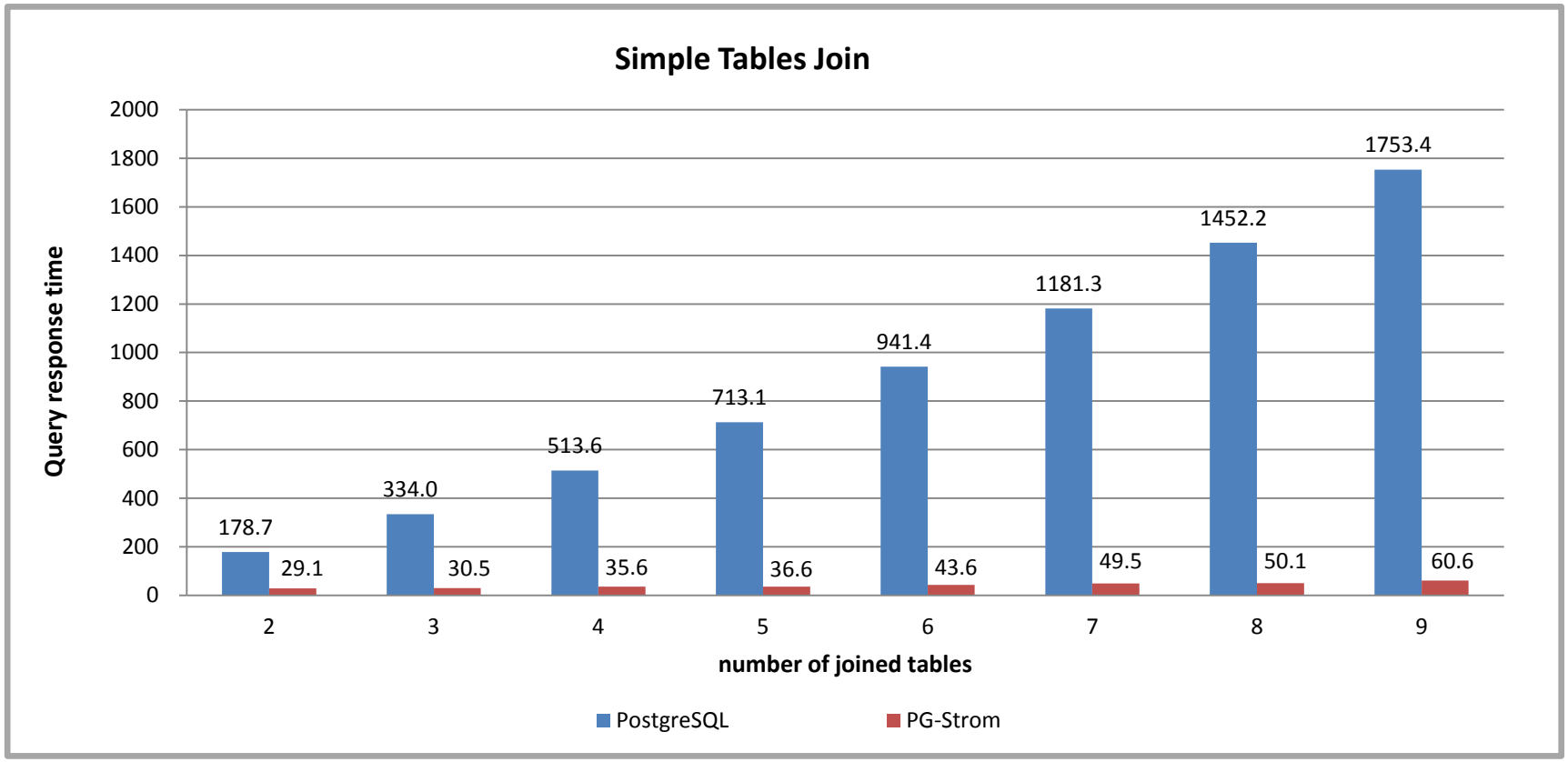
```
postgres=# SET pg_strom.show_device_kernel = on;
SET
postgres=# EXPLAIN (verbose, costs off) SELECT cat, avg(x) from t0 WHERE x < y GROUP BY cat;
                                        QUERY PLAN
-------------------------------------------------------------------------------------------
 HashAggregate
   Output: cat, pgstrom.avg(pgstrom.nrows(x IS NOT NULL), pgstrom.psum(x))
   Group Key: t0.cat
   ->  Custom (GpuPreAgg)
         Output: NULL::integer, cat, NULL::integer, NULL::integer, NULL::integer, NULL::integer,
                 NULL::double precision, NULL::double precision, NULL::text,
                 pgstrom.nrows(x IS NOT NULL), pgstrom.psum(x)
         Bulkload: On
         Kernel Source: #include "opencl_common.h"
 #include "opencl_gpupreagg.h"
 #include "opencl_textlib.h"
         :  <...snip...>
 static bool
 gpupreagg_qual_eval(__private cl_int *errcode,
                     __global kern_parambuf *kparams,
                     __global kern_data_store *kds,
                     __global kern_data_store *ktoast,
                     size_t kds_index)
 {
   pg_float8_t KVAR_7 = pg_float8_vref(kds,ktoast,errcode,6,kds_index);
   pg_float8_t KVAR_8 = pg_float8_vref(kds,ktoast,errcode,7,kds_index);

   return EVAL(pgfn_float8lt(errcode, KVAR_7, KVAR_8));
 }
```

How PG-Strom processes SQL workloads① – Hash-Join



Benchmark (1/2) – Simple Tables Join

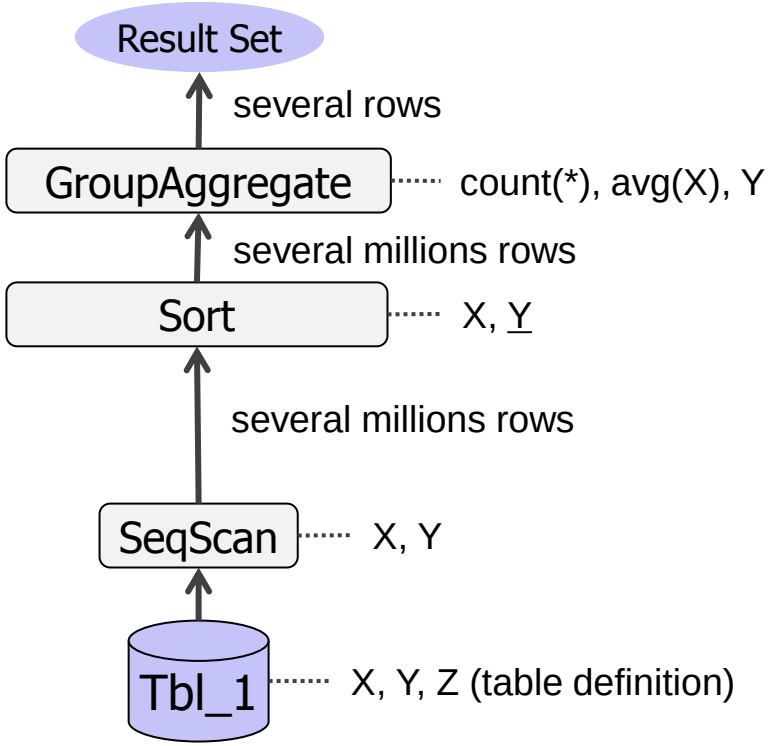


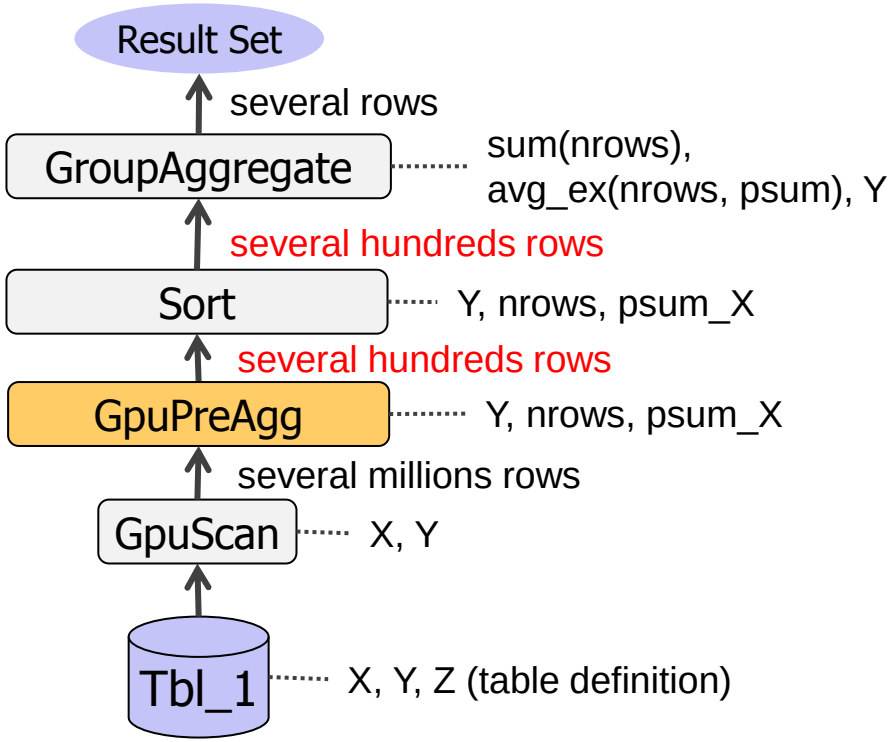
[condition of measurement]

- INNER JOIN of 200M rows x 100K rows x 10K rows ... with increasing number of tables
- Query in use: `SELECT * FROM t0 natural join t1 [natural join t2 ...];`
- All the tables are preliminary loaded
- HW: Express5800 HR120b-1, CPU: Xeon E5-2640, RAM: 256GB, GPU: NVIDIA GTX980

How PG-Strom processes SQL workloads② – Aggregate

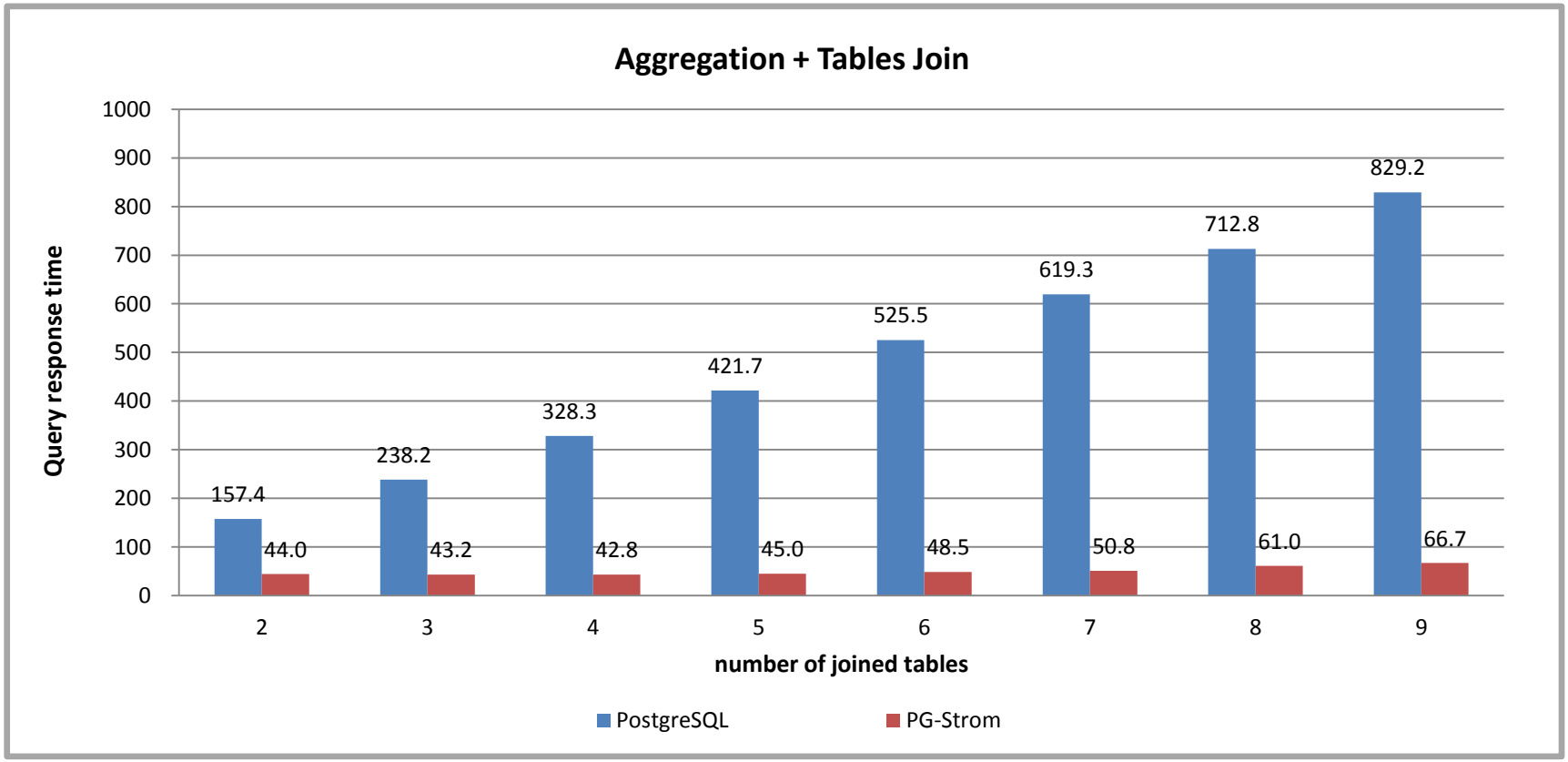
SELECT count(*), AVG(X), Y FROM Tbl_1 GROUP BY Y;





- GpuPreAgg, prior to Aggregate/Sort, reduces num of rows to be processed by CPU
- Not easy to apply aggregate on all the data at once because of GPU's RAM size, GPU has advantage to make "partial aggregate" for each million rows.
- Key performance factor is reducing the job of CPU

Benchmark (2/2) – Aggregation + Tables Join



[condition of measurement]

- Aggregate and INNER JOIN of 200M rows x 100K rows x 10K rows ... with increasing number of tables
- Query in use:

```
SELECT cat, AVG(x) FROM t0 natural join t1 [natural join t2 ...] GROUP BY CAT;
```

- Other conditions are same with the previous measurement

As an aside...

Let’s back to the development history prior to the remaining elemental technology

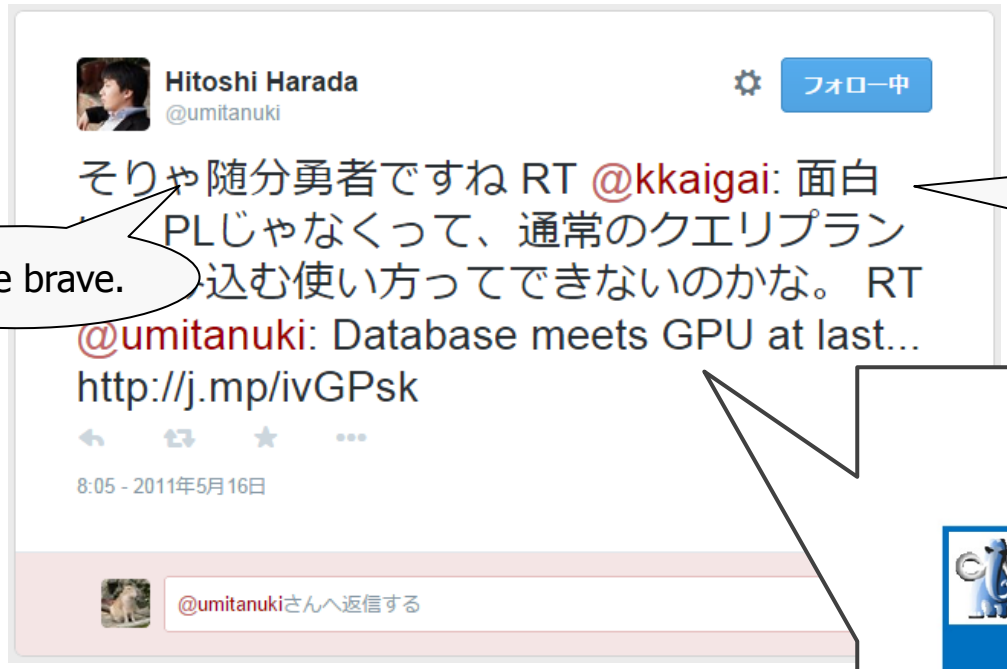
Development History of PG-Strom

2011	Feb	KaiGai moved to Germany
	May	PGconf 2011
2012	Jan	The first prototype
	May	Tried to use pseudo code
	Aug	Proposition of background worker and writable FDW
2013	Jul	NEC admit PG-Strom development
	Nov	Proposition of CustomScan API
2014	Feb	Moved to OpenCL from CUDA
	Jun	GpuScan, GpuHashJoin and GpuSort were implemented
	Sep	Drop GpuSort, instead of GpuPreAgg
	Nov	working beta-1 gets available

First prototype announced

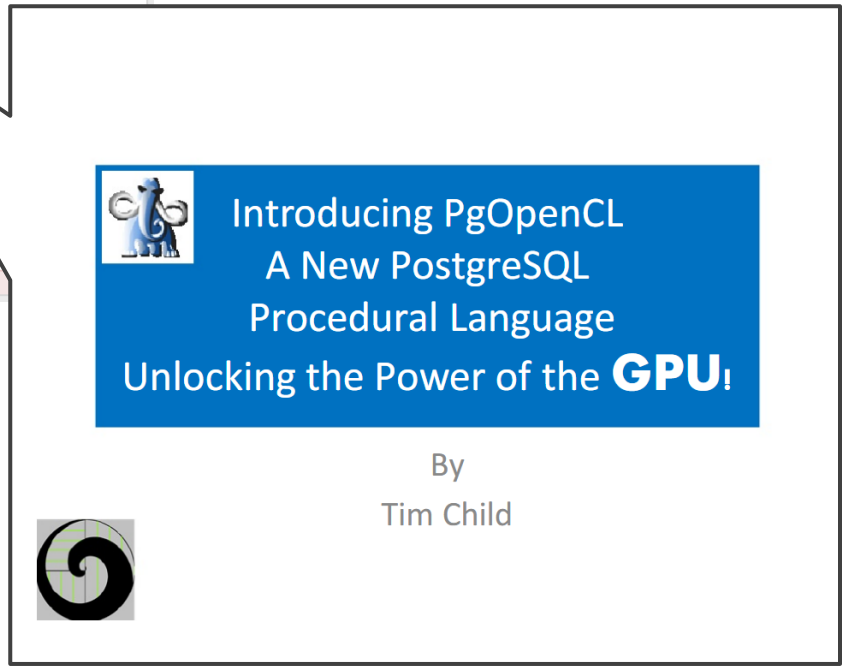
"Strom" comes from Germany term; where I lived in at that time

Inspiration – May-2011



You're brave.

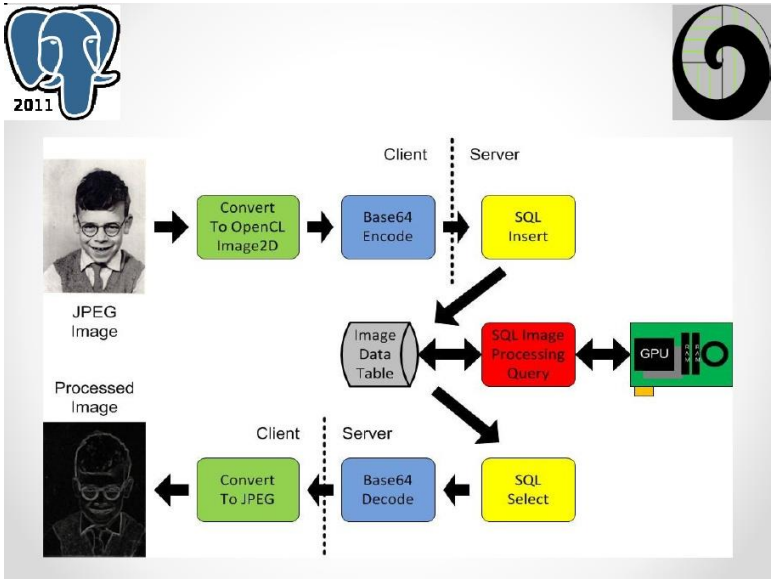
Unavailable to run on regular query, not only PL functions?



http://ja.scribd.com/doc/44661593/PostgreSQL-OpenCL-Procedural-Language

PGconf 2011 @ Ottawa, Canada

Inspired by PgOpencl Project

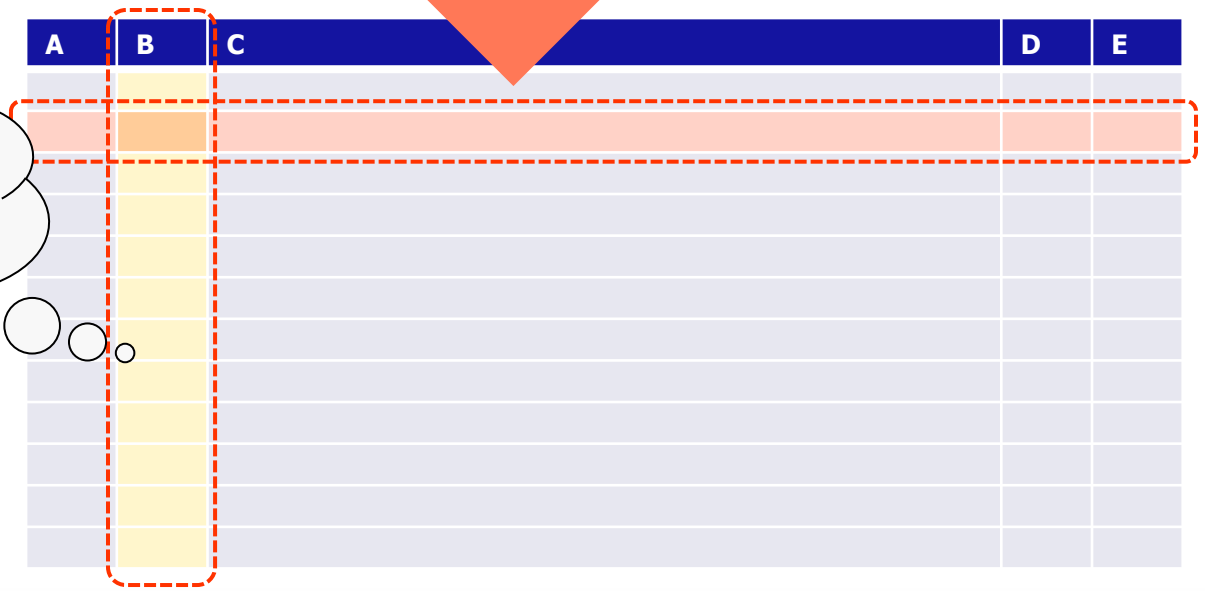


Parallel Image Searching Using PostgreSQL PgOpenCL @ PGconf 2011
Tim Child (3DMashUp)

summary:

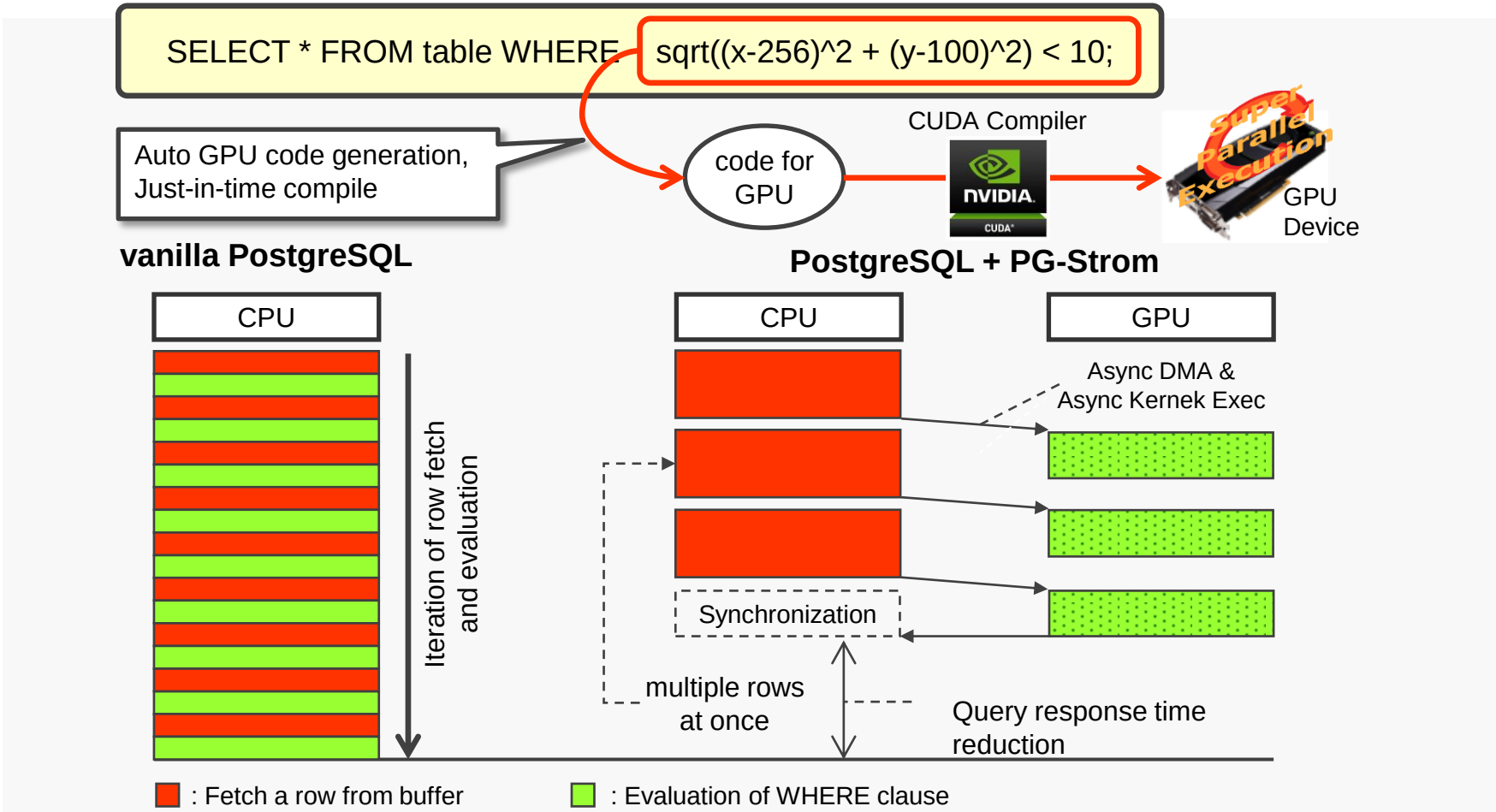
GPU works well towards large BLOB, like image data.

Even small width of values, GPU will work well if we have transposition.



Made a prototype – Christmas vacation in 2011

- CUDA based (➔ Now, OpenCL)
- FDW (Foreign Data Wrapper) based (➔ Now, CustomScan API)
- Column oriented data structure (➔ Now, row-oriented data)

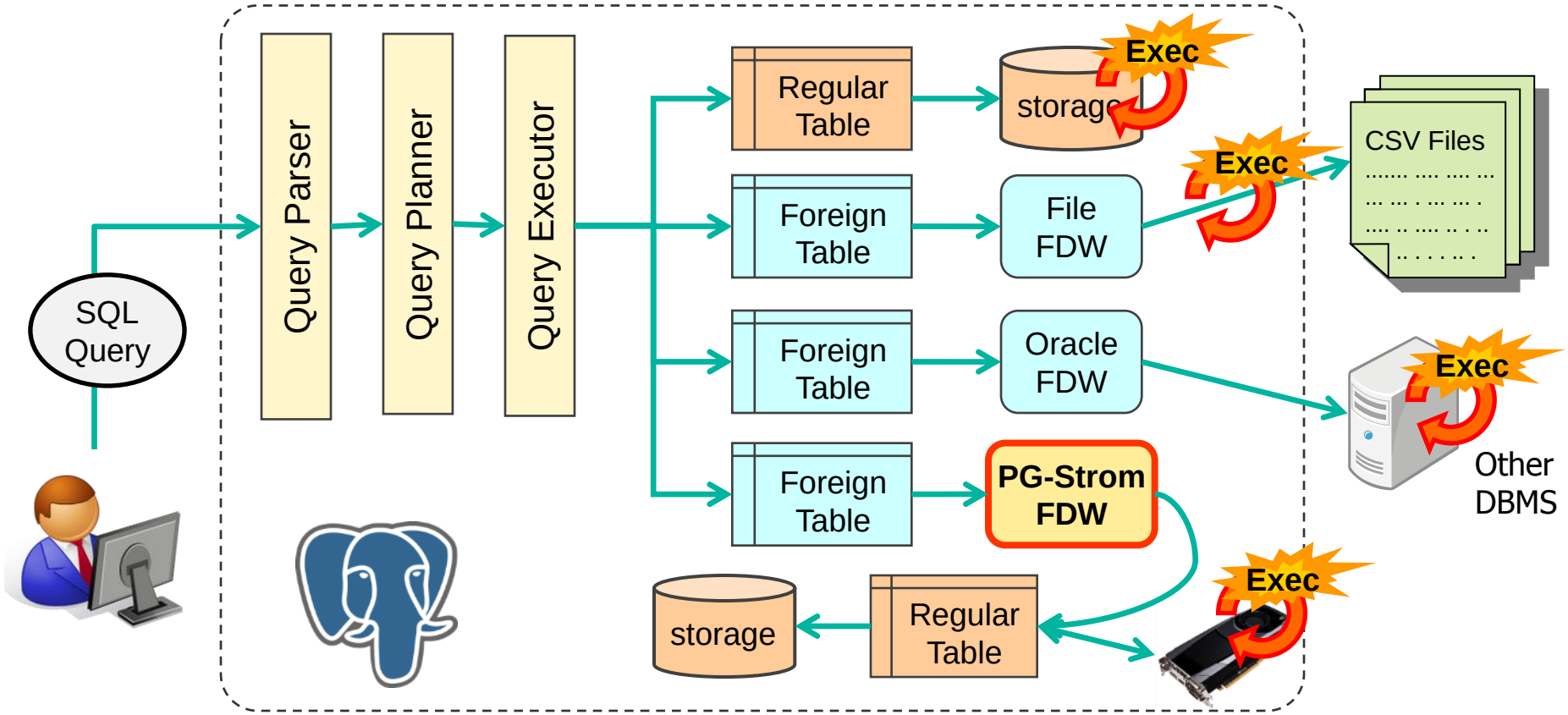


What is FDW (Foreign Data Wrapper)

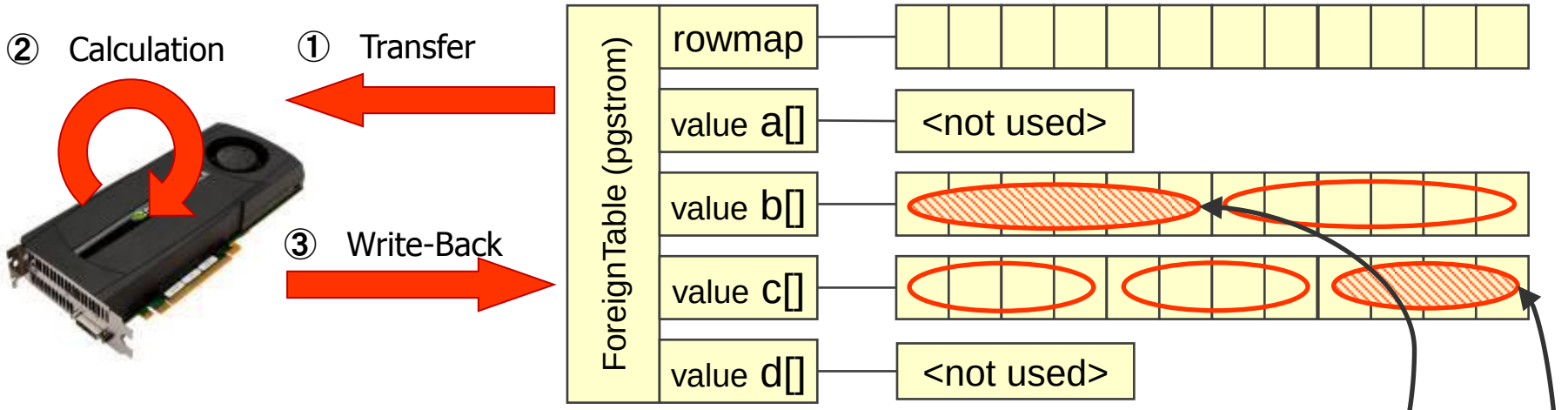
- A set of APIs to show external data source as like a table of PostgreSQL
- FDW driver generates rows on demand when foreign tables are referenced.

➔ Extension can have arbitrary data structure and logic to process the data.

➔ It is also allowed to scan internal data with GPU acceleration!



Implementation at that time



Problem

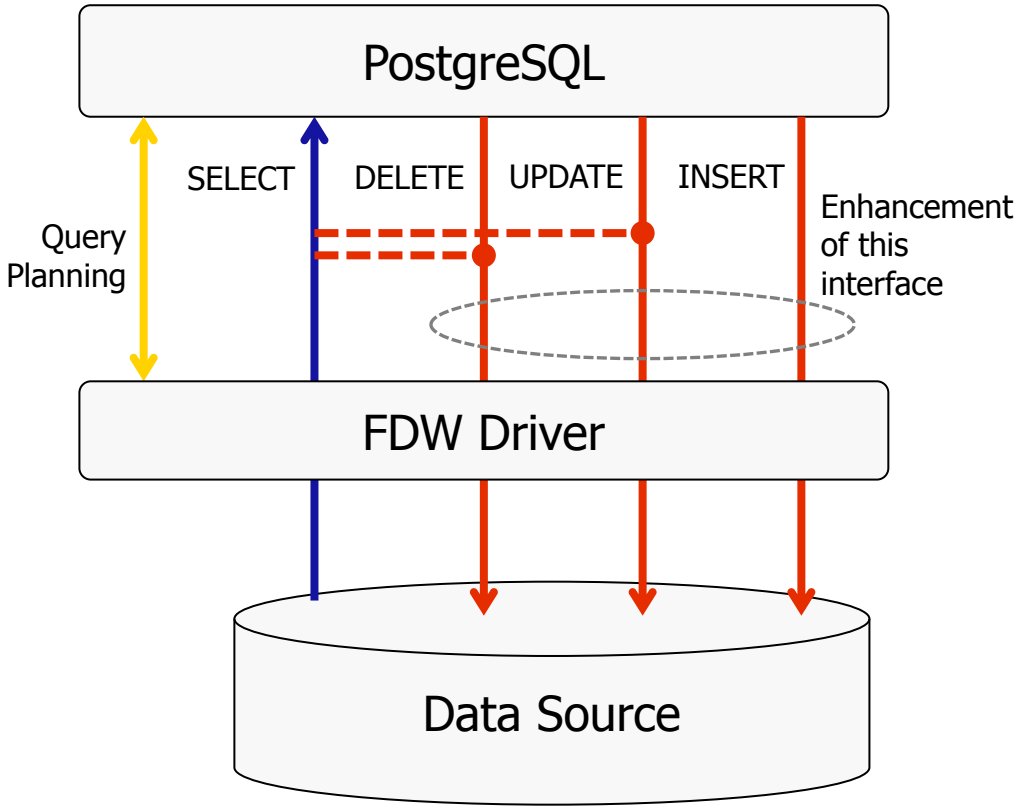
- Only foreign tables can be accelerated with GPUs.
- Only full table-scan can be supported.
- Foreign tables were read-only at that time.
- Slow-down of 1st time query because of device initialization.

➔summary: Not easy to use

PostgreSQL Enhancement (1/2) – Writable FDW

- Foreign tables had supported read access only (~v9.2)
  - Data load onto shadow tables was supported using special function

➔Enhancement of APIs for INSERT/UPDATE/DELETE on foreign tables



PostgreSQL Enhancement (2/2) – Background Worker

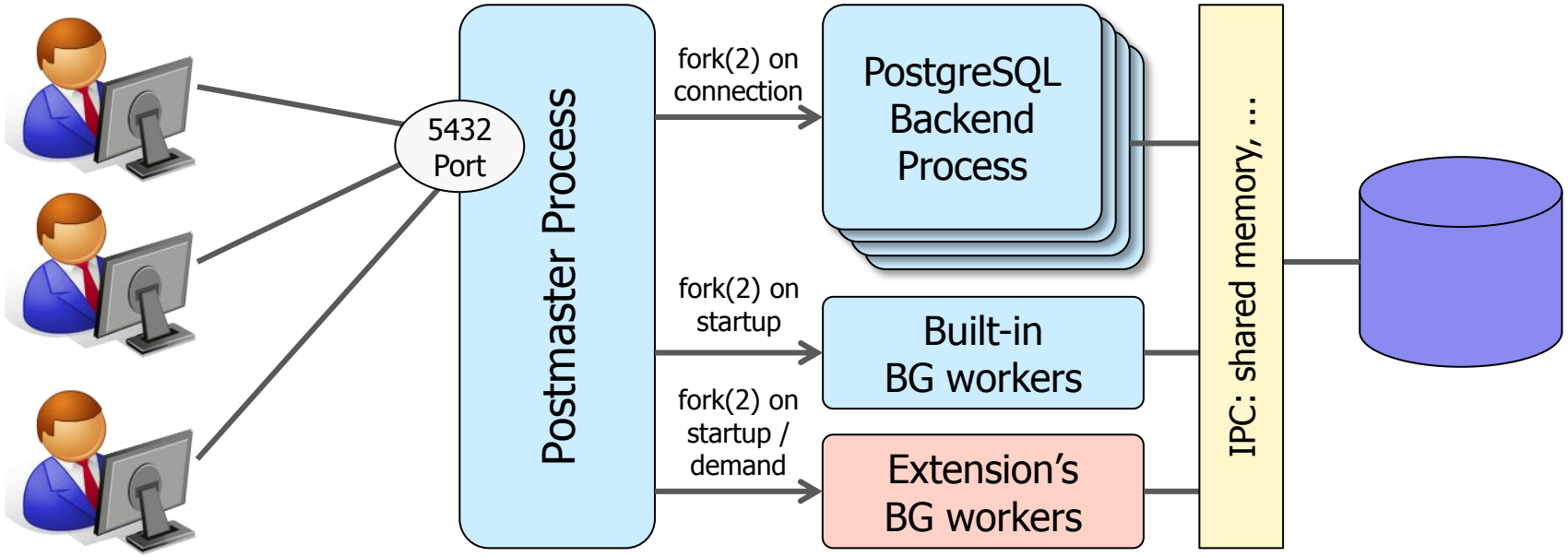
Slowdown on first time of each query

- Time for JIT compile of GPU code ➔ caching the built binaries
- Time for initialization of GPU devices ➔ initialization once by worker process

➔Good side effect: it works with run-time that doesn't support concurrent usage.

Background Worker

- An interface allows to launch background processes managed by extensions
- Dynamic registration gets supported on v9.4 also.



Design Pivot

Is the FDW really appropriate framework for PG-Strom?

- It originated from a feature to show external data as like a table.

[benefit]

- It is already supported on PostgreSQL

[Issue]

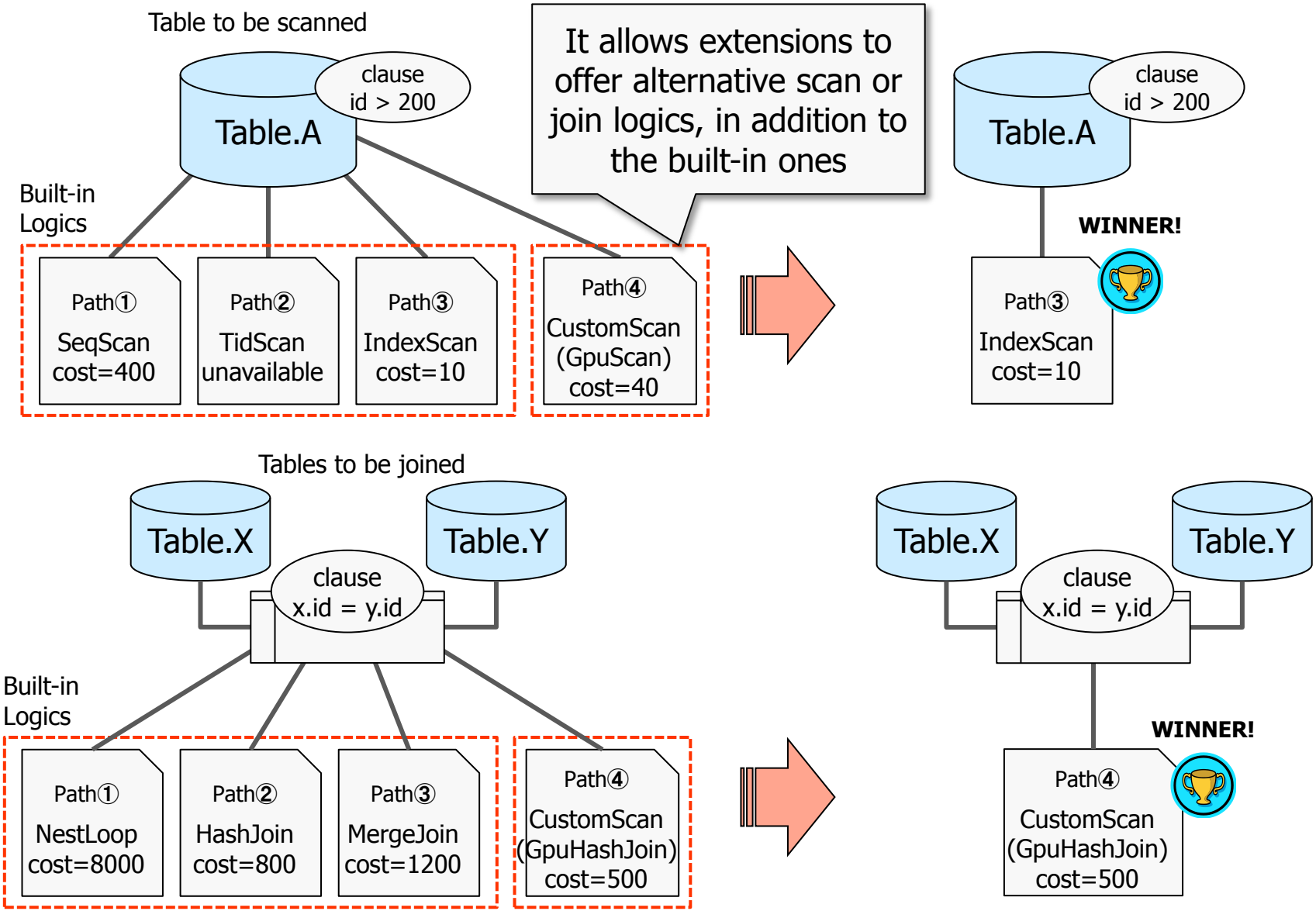
- Less transparency for users / applications
- More efficient execution path than GPU; like index-scan if available
- Workloads expects for full-table scan; like tables join

CUDA or OpenCL?

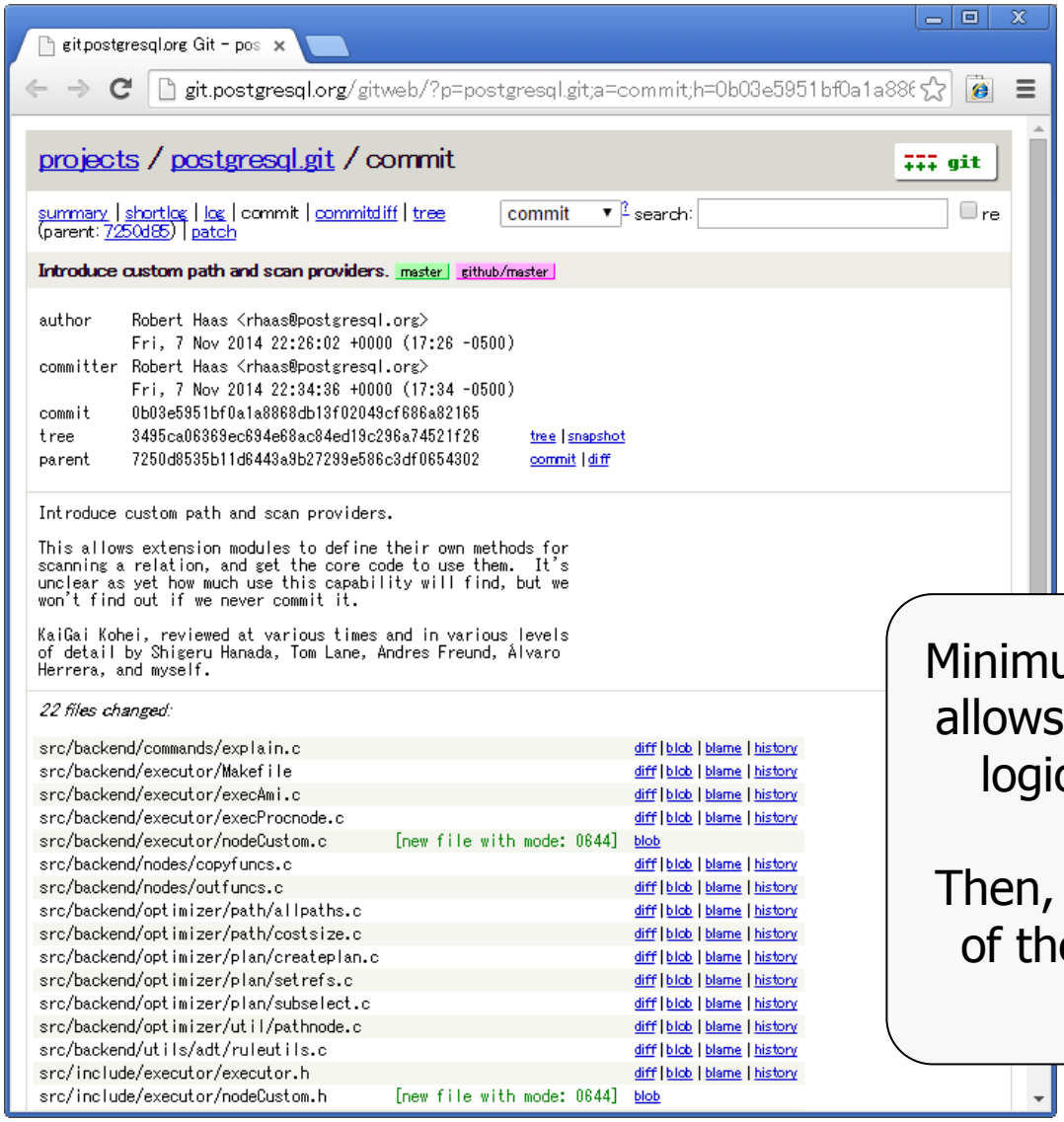
	CUDA	OpenCL
Good	• Fine grained optimization • Latest feature of NVIDIA GPU • Reliability of OpenCL drivers	• Multiplatform support • Built-in JIT compiler • CPU parallel support
Bad	• No support on AMD, Intel	• Reliability of OpenCL drivers

➔ First priority is many trials by wider range of people

Elemental Technology② – Custom-Scan Interface



Yes!! Custom-Scan Interface got merged to v9.5



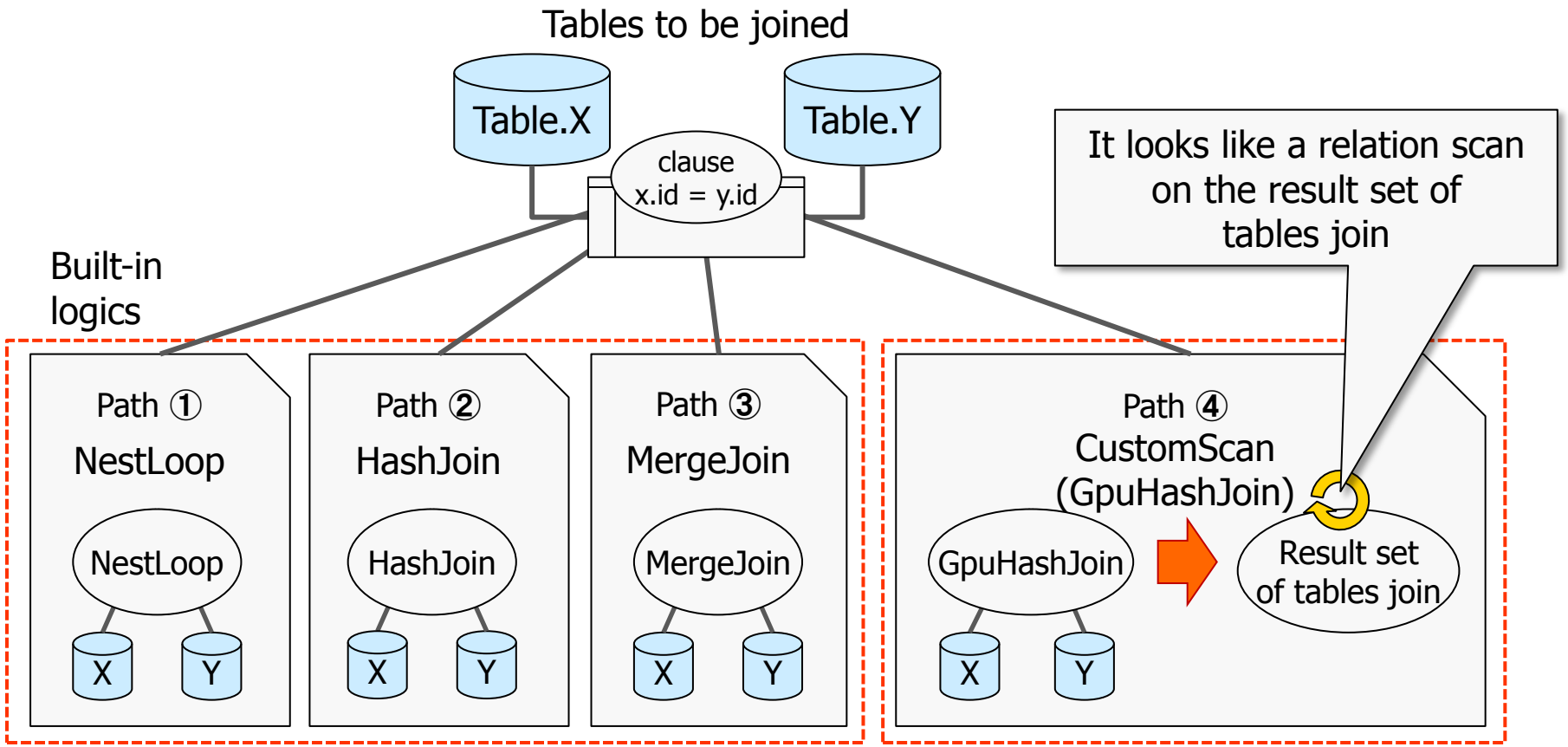
Minimum consensus is, an interface that allows extensions to implement custom logic to replace built-in scan logics.

↓

Then, we will discuss the enhancement of the interface for tables join on the developer's community.

Enhancement of Custom-Scan Interface

Join replacement by foreign-/custom-scan



- It replaces a node to be join by foreign-/custom-scan.
- Example: A FDW that scan on the result set of remote join.

Abuse of Custom-Scan Interface

```
postgres=# EXPLAIN SELECT * FROM t0 NATURAL JOIN t1;
                                  QUERY PLAN
------------------------------------------------------------------------------
 Custom (GpuHashJoin)  (cost=2234.00..468730.50 rows=19907950 width=106)
   hash clause 1: (t0.aid = t1.aid)
   Bulkload: On
   ->  Custom (GpuScan) on t0  (cost=500.00..267167.00 rows=20000024 width=73)
   ->  Custom (MultiHash)  (cost=734.00..734.00 rows=40000 width=37)
         hash keys: aid
         Buckets: 46000  Batches: 1  Memory Usage: 99.97%
         ->  Seq Scan on t1  (cost=0.00..734.00 rows=40000 width=37)
 Planning time: 0.220 ms
(9 rows)
```

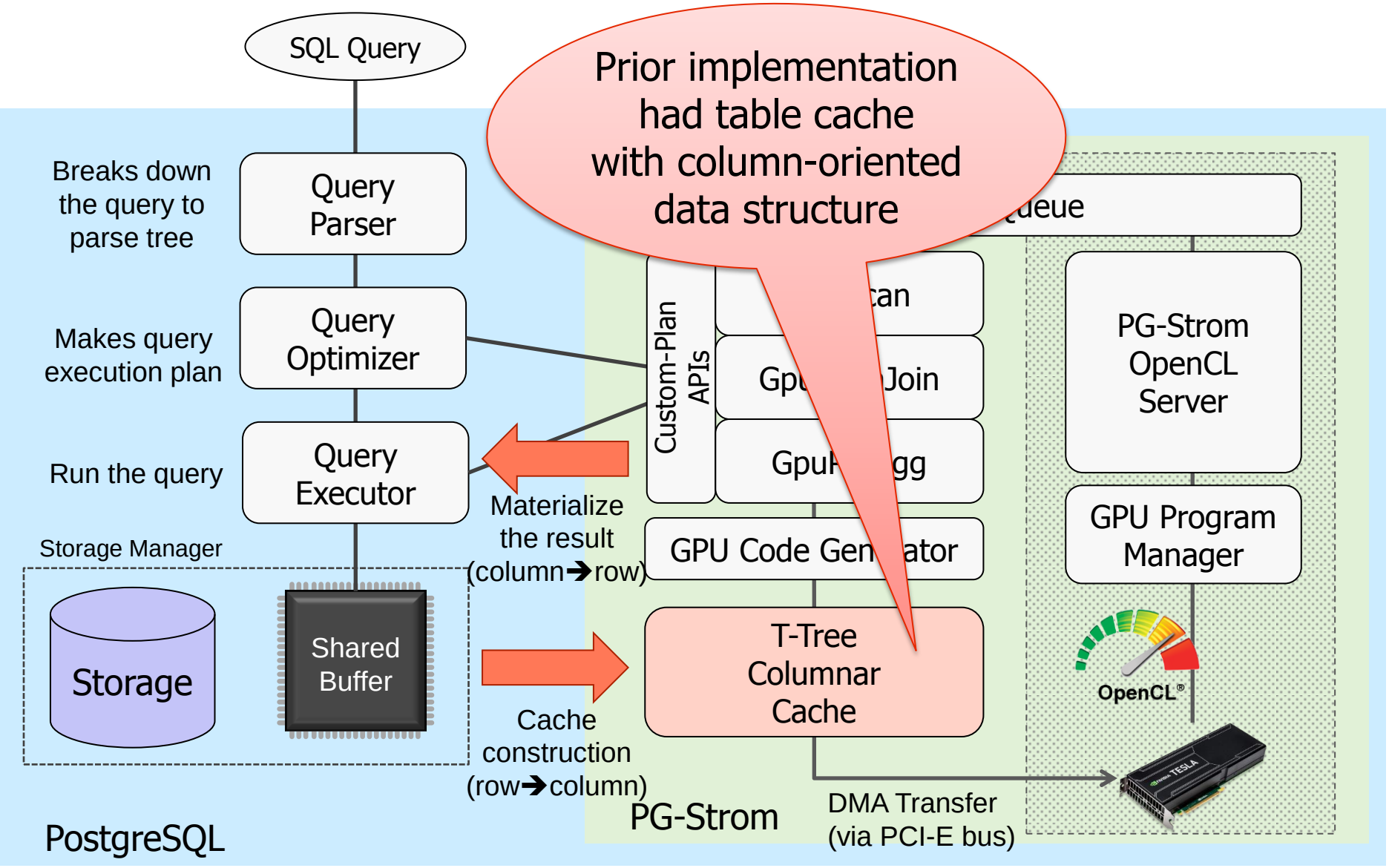
Usual interface contract

- `GpuScan/SeqScan` fetches rows, then passed to upper node and more ...
- TupleTableSlot is used to exchange record row by row manner.

Beyond the interface contract

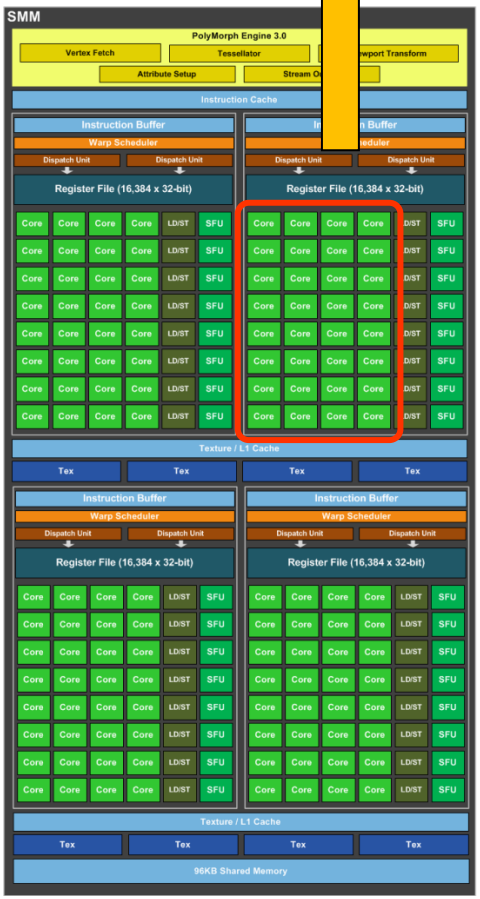
- In case when both of parent and child nodes are managed by same extension, nobody prohibit to exchange chunk of data with its own data structure.
- "Bulkload: On" ➔ multiple (usually, 10K~100K) rows at once

Element Technology③ – Row oriented data structure

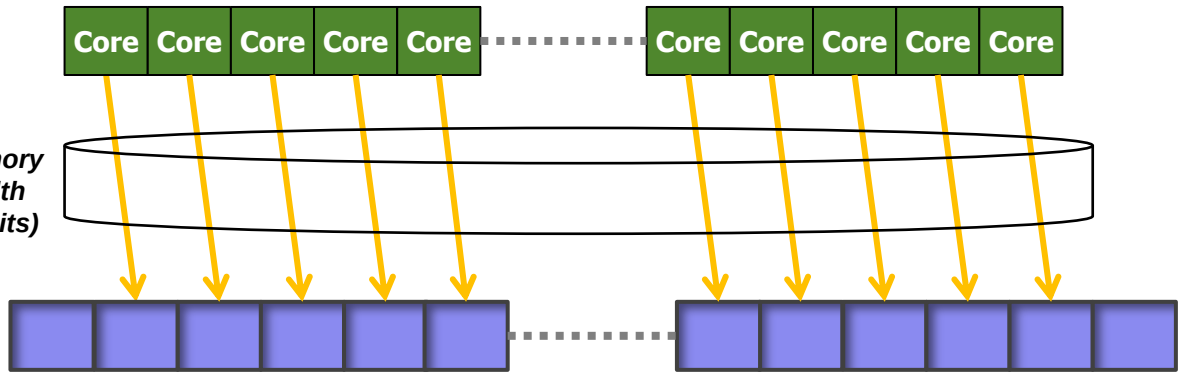


Why GPU well fit column-oriented data structure

WARP:
A set of processor cores that share instruction pointer. It is usually 32.

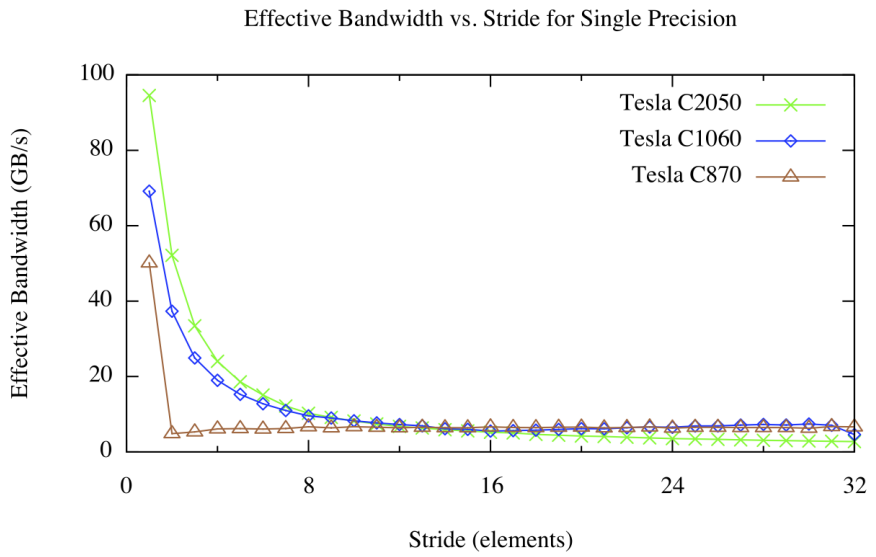


coalesced memory access



Wide Memory Bandwidth (256-384bits)

Global Memory (DRAM)



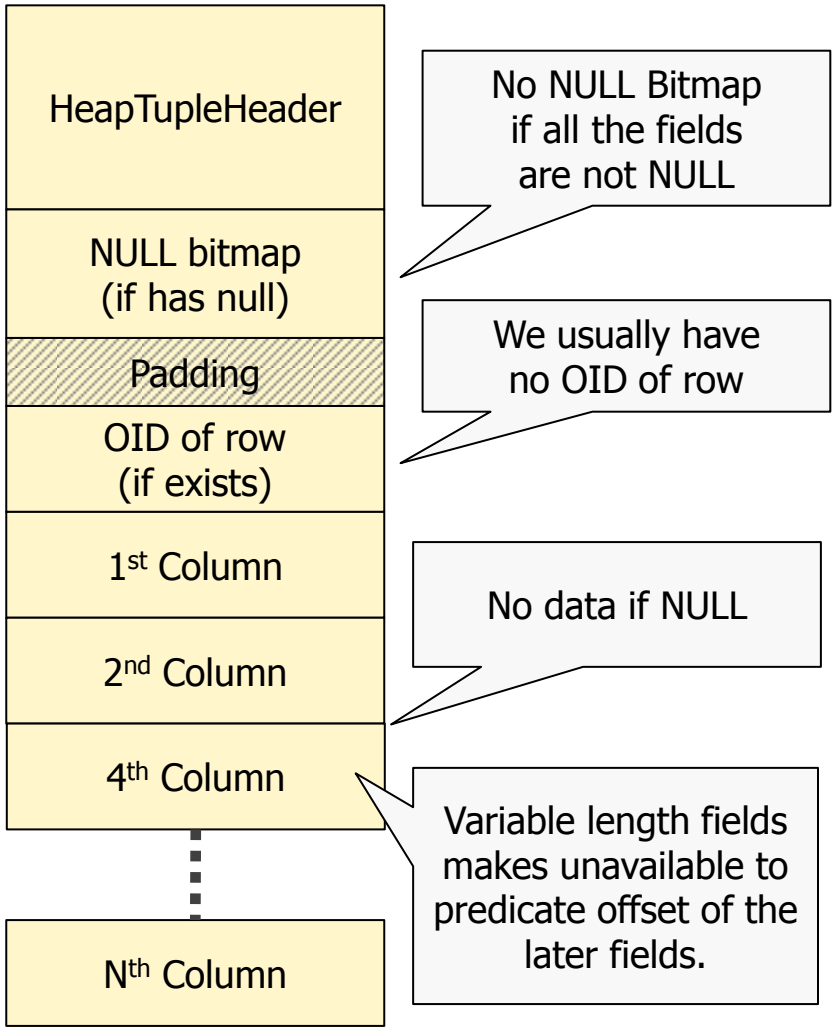
SOURCE: Maxwell: The Most Advanced CUDA GPU Ever Made

SOURCE: How to Access Global Memory Efficiently in CUDA C/C++ Kernels

Format of PostgreSQL tuples

```
struct HeapTupleHeaderData
{
  union
  {
    HeapTupleFields t_heap;
    DatumTupleFields t_datum;
  }   t_choice;
  /* current TID of this or newer tuple */
  ItemPointerData t_ctid;
  /* number of attributes + various flags */
  uint16      t_infomask2;
  /* various flag bits, see below */
  uint16      t_infomask;
  /* sizeof header incl. bitmap, padding */
  uint8       t_hoff;
  /* ^ - 23 bytes - ^ */
  /* bitmap of NULLs -- VARIABLE LENGTH */
  bits8       t_bits[1];
  /* MORE DATA FOLLOWS AT END OF STRUCT */
};
```

The worst data structure for GPU processor

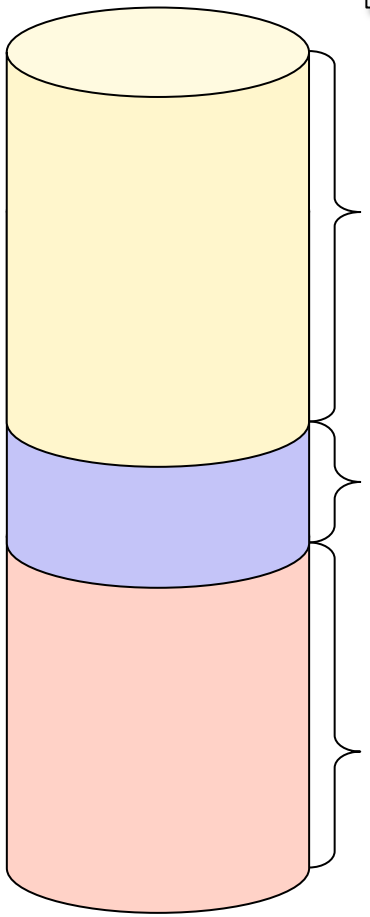


Nightmare of columnar cache (1/2)

```
postgres=# explain (analyze, costs off)
           select * from t0 natural join t1 natural join t2;
                                QUERY PLAN
 ------------------------------------------------------------------------------
Custom (GpuHashJoin) (actual time=54.005..9635.134 rows=20000000 loops=1)
   hash clause 1: (t0.aid = t1.aid)
   hash clause 2: (t0.bid = t2.bid)
   number of requests: 144
   total time to load: 584.67ms
   total time to materialize: 7245.14ms  ← 70% of total execution time!!
   average time in send-mq: 37us
   average time in recv-mq: 0us
   max time to build kernel: 1us
   DMA send: 5197.80MB/sec, len: 2166.30MB, time: 416.77ms, count: 470
   DMA recv: 5139.62MB/sec, len: 287.99MB, time: 56.03ms, count: 144
   kernel exec: total: 441.71ms, avg: 3067us, count: 144
   ->  Custom (GpuScan) on t0 (actual time=4.011..584.533 rows=20000000 loops=1)
   ->  Custom (MultiHash) (actual time=31.102..31.102 rows=40000 loops=1)
         hash keys: aid
         ->  Seq Scan on t1 (actual time=0.007..5.062 rows=40000 loops=1)
         ->  Custom (MultiHash) (actual time=17.839..17.839 rows=40000 loops=1)
               hash keys: bid
               ->  Seq Scan on t2 (actual time=0.019..6.794 rows=40000 loops=1)
Execution time: 10525.754 ms
```

Nightmare of columnar cache (2/2)

Breakdown of execution time



Optimization in GPU also makes additional data-format translation cost (not ignorable) on the interface of PostgreSQL

Translation from table (row-format) to the columnar-cache (only at once)

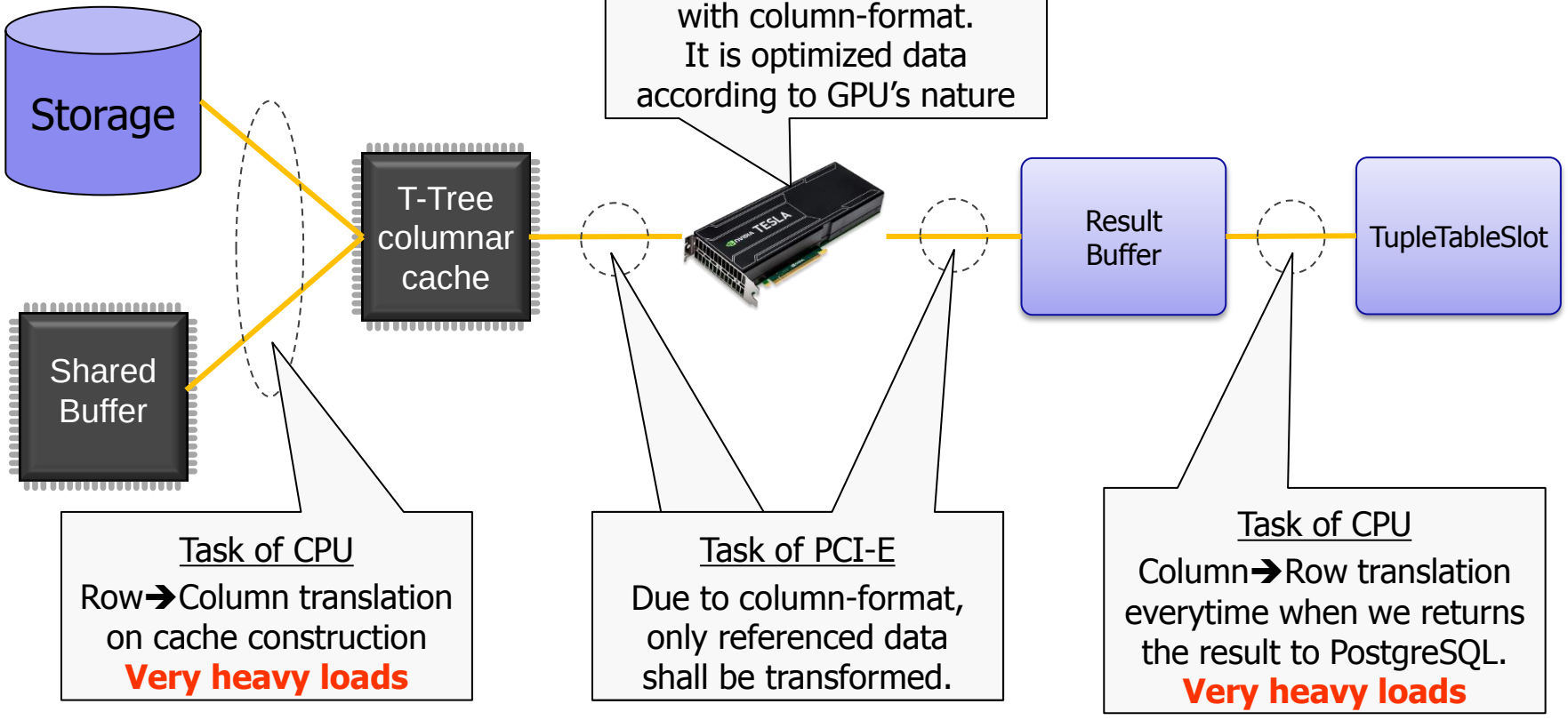
[Hash-Join] Search the relevant records on inner/outer relations.

Translation from PG-Strom internal (column-format) to TupleTableSlot (row-format) for data exchange in PostgreSQL

A significant point in GPU acceleration (1/2)

- CPU is rare resource, so should put less workload on CPUs unlike GPUs
- We need to pay attention on access pattern of memory for CPU's job

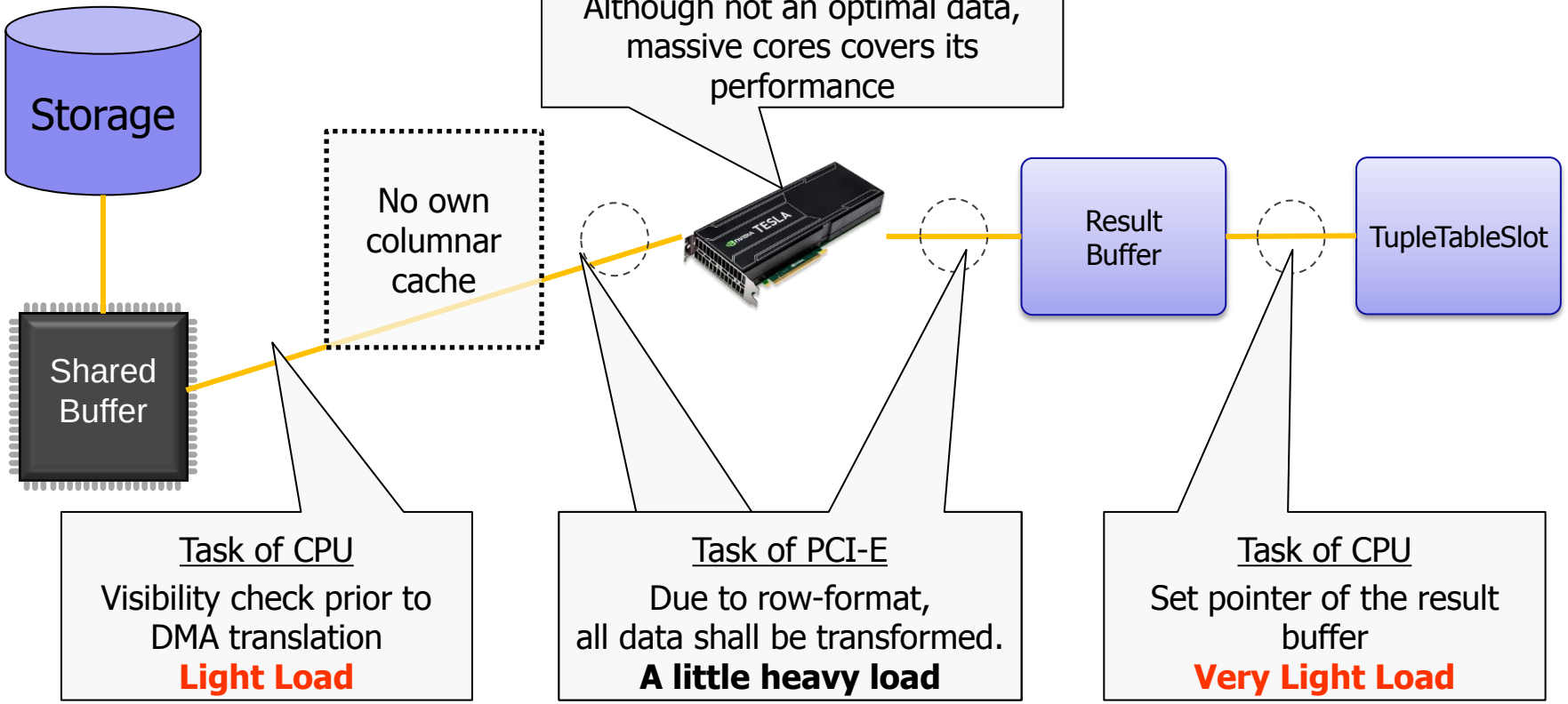
In case of column-format



A significant point in GPU acceleration (2/2)

- CPU is rare resource, so should put less workload on CPUs unlike GPUs
- We need to pay attention on access pattern of memory for CPU's job

In case of row-format



Integration with shared-buffer of PostgreSQL

```
postgres=# explain (analyze, costs off)
           select * from t0 natural join t1 natural join t2;
                                    QUERY PLAN
-----------------------------------------------------------------
 Custom (GpuHashJoin) (actual time=111.085..4286.562 rows=20000000 loops=1)
   hash clause 1: (t0.aid = t1.aid)
   hash clause 2: (t0.bid = t2.bid)
   number of requests: 145
   total time for inner load: 29.80ms
   total time for outer load: 812.50ms
   total time to materialize: 1527.95ms  ← Reduction dramatically
   average time in send-mq: 61us
   average time in recv-mq: 0us
   max time to build kernel: 1us
   DMA send: 5198.84MB/sec, len: 2811.40MB, time: 540.77ms, count: 619
   DMA recv: 3769.44MB/sec, len: 2182.02MB, time: 578.87ms, count: 290
   proj kernel exec: total: 264.47ms, avg: 1823us, count: 145
   main kernel exec: total: 622.83ms, avg: 4295us, count: 145
   ->  Custom (GpuScan) on t0 (actual time=5.736..812.255 rows=20000000 loops=1)
   ->  Custom (MultiHash) (actual time=29.766..29.767 rows=80000 loops=1)
         hash keys: aid
         ->  Seq Scan on t1 (actual time=0.005..5.742 rows=40000 loops=1)
         ->  Custom (MultiHash) (actual time=16.552..16.552 rows=40000 loops=1)
               hash keys: bid
               ->  Seq Scan on t2 (actual time=0.022..7.330 rows=40000 loops=1)
 Execution time: 5161.017 ms  ← Much better response time
```

Performance degrading little bit

PG-Strom's current

Supported logics

- GpuScan ... Full-table scan with qualifiers
- GpuHashJoin ... Hash-Join with GPUs
- GpuPreAgg ... Preprocess of aggregation with GPUs

Supported data-types

- Integer (smallint, integer, bigint)
- Floating-point (real, float)
- String (text, varchar(n), char(n))
- Date and time (date, time, timestamp)
- NUMERIC

Supported functions

- arithmetic operators for above data-types
- comparison operators for above data-types
- mathematical functions on floating-points
- Aggregate: MIN, MAX, SUM, AVG, STD, VAR, CORR

PG-Strom's future

Logics will be supported

- Sort
- Aggregate Push-down
- ... anything else?

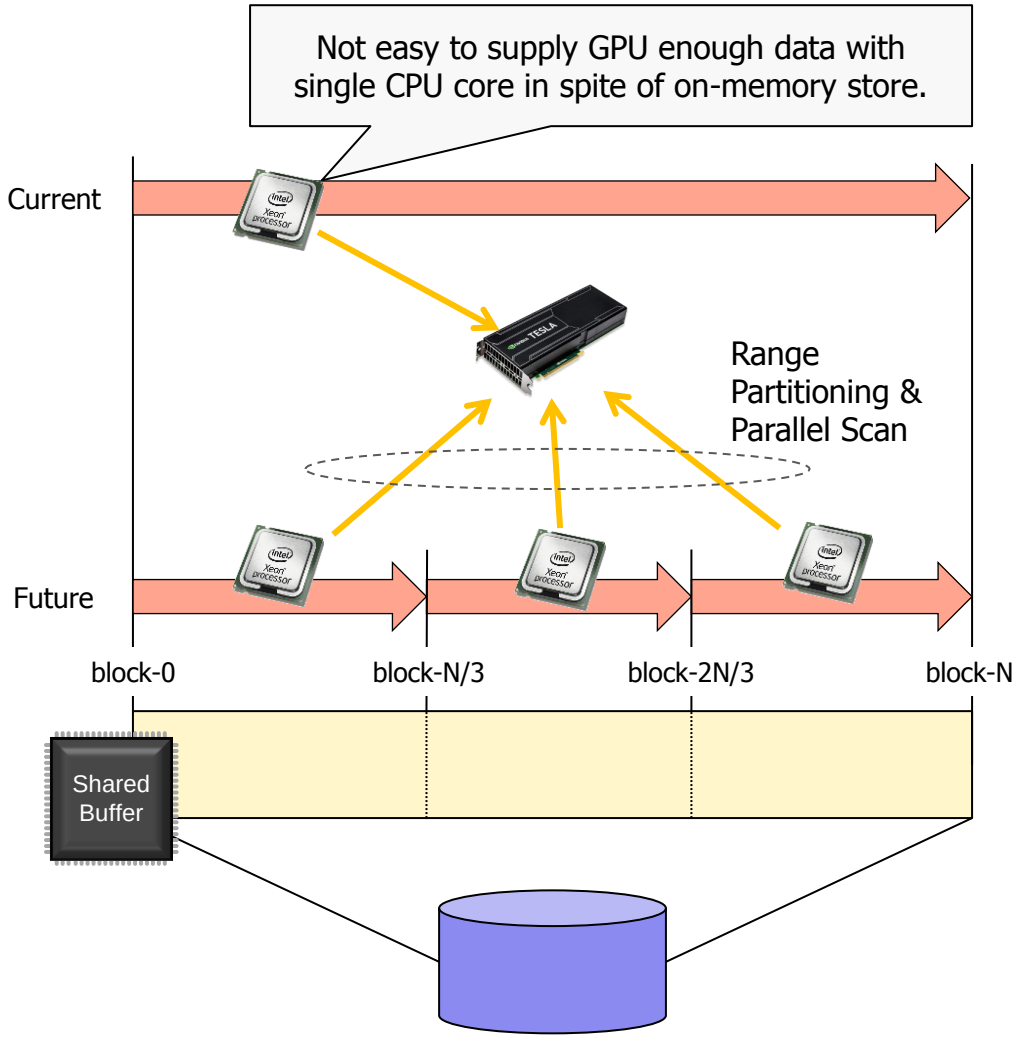
Data types will be supported

- ... anything else?

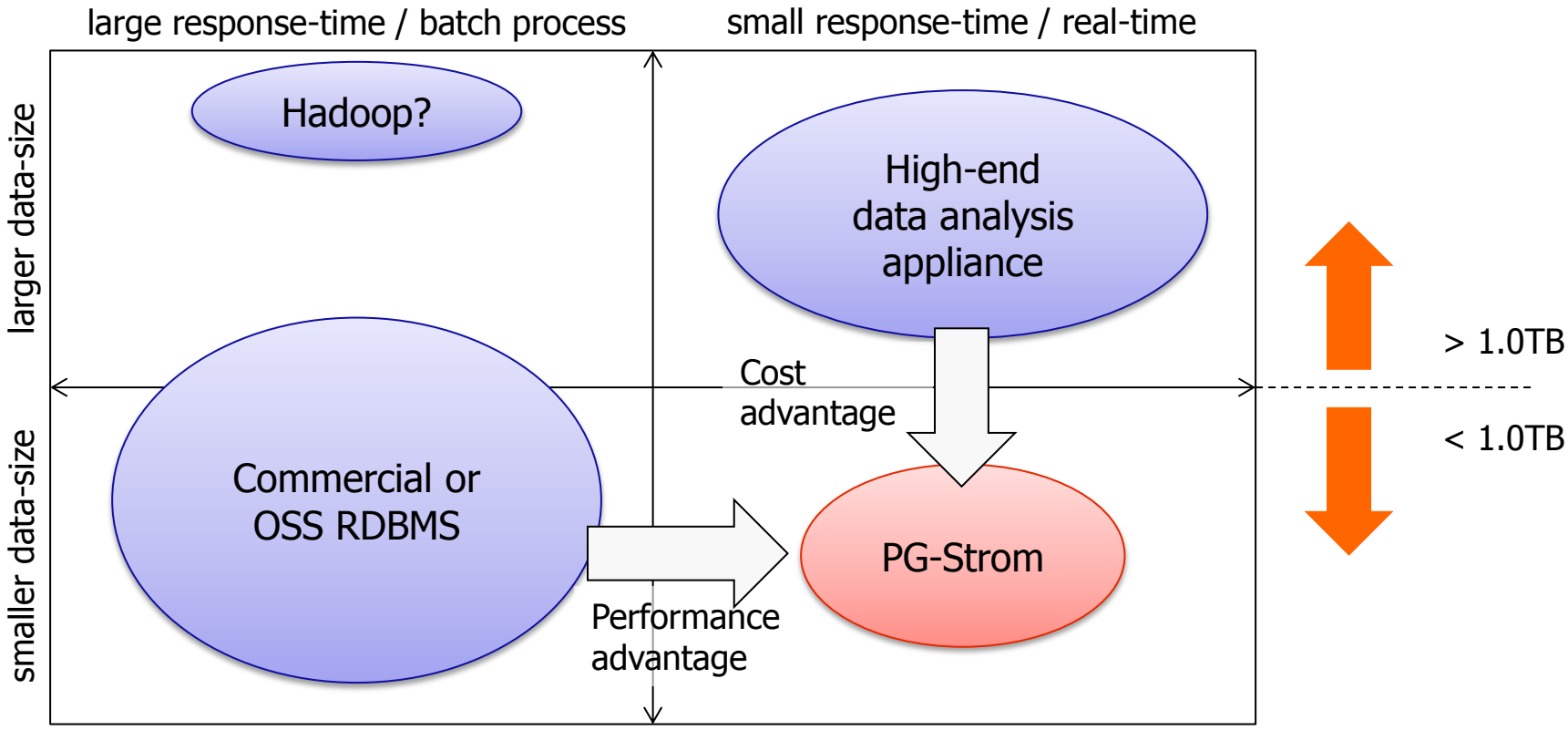
Functions will be supported

- LIKE, Regular Expression?
- Date/Time extraction?
- PostGIS?
- Geometrics?
- User defined functions? (like pgOpenCL)

Parallel Scan

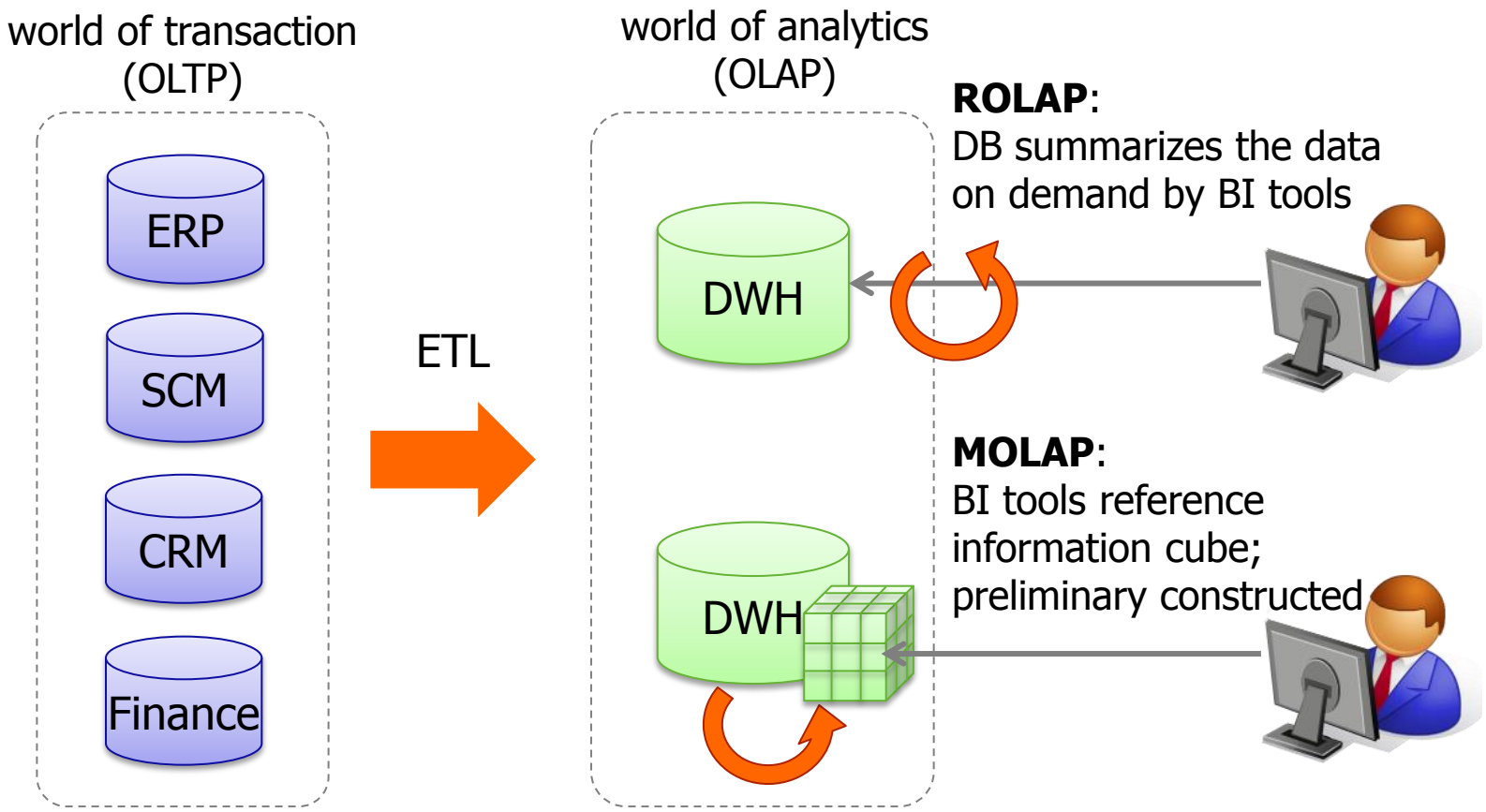


Our target segment



- 1st target application: BI tools are expected
- Up-to 1.0TB data: SMB company or branches of large company
  - Because it needs to keep the data in-memory
- GPU and PostgreSQL: both of them are inexpensive.

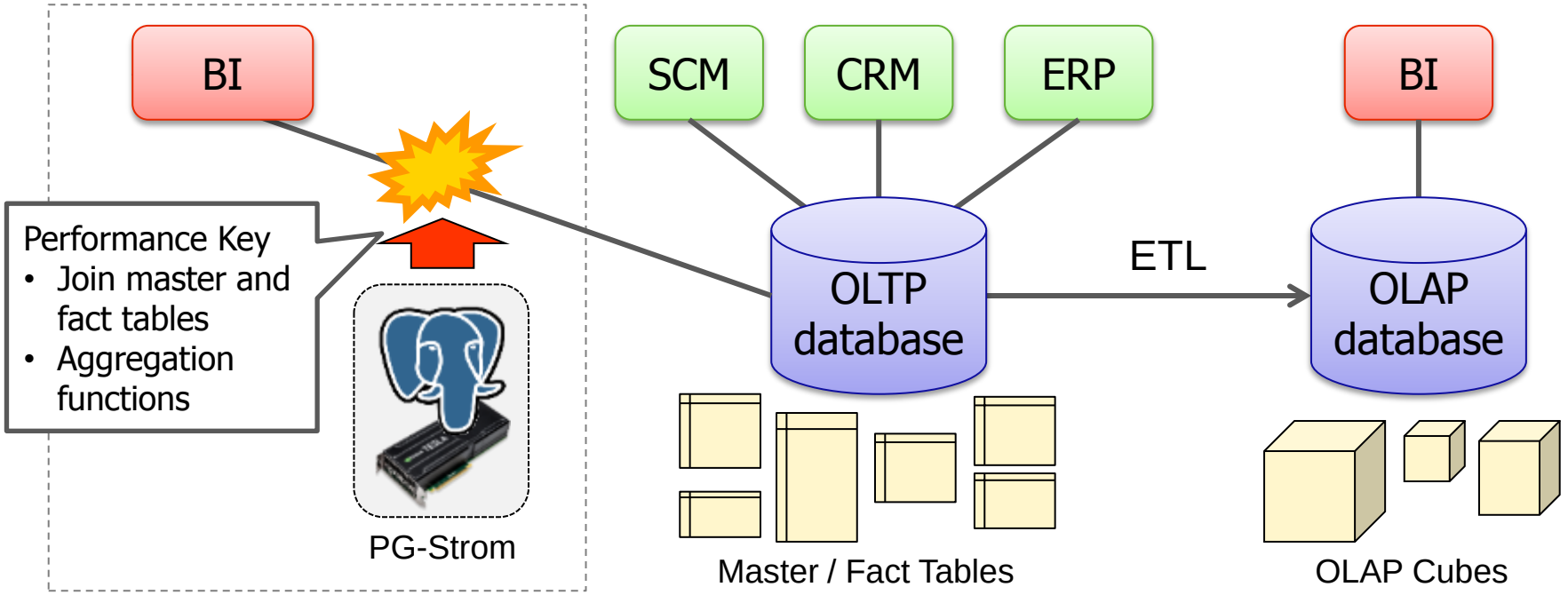
(Ref) ROLAP and MOLAP



Expected PG-Strom usage

- Backend RDBMS for ROLAP / acceleration of ad-hoc queries
- Cube construction for MOLAP / acceleration of batch processing

Expected Usage (1/2) – OLTP/OLAP Integration



- Why separated OLTP/OLAP system
  - Integration of multiple data source
  - Optimization for analytic workloads
- How PG-Strom will improve...
  - Parallel execution of Join / Aggregate; being key of performance
  - Elimination or reduction of OLAP / ETL, thus smaller amount of TCO

Expected Usage (2/2) – Let's investigate with us

- Which region PG-Strom can work for?
- Which workloads PG-Strom shall fit?
- What workloads you concern about?

➔ PG-Strom Project want to lean from the field.

How to use (1/3) – Installation Prerequisites

- OS: Linux (RHEL 6.x was validated)
- PostgreSQL 9.5devel (with Custom-Plan Interface)
- PG-Strom Module
- OpenCL Driver (like nVIDIA run-time)

Minimum configuration of PG-Strom

```
shared_preload_libraries = '$libdir/pg_strom'
shared_buffers = <enough size to load whole of the database>
```

Turn on/off PG-Strom at run-time

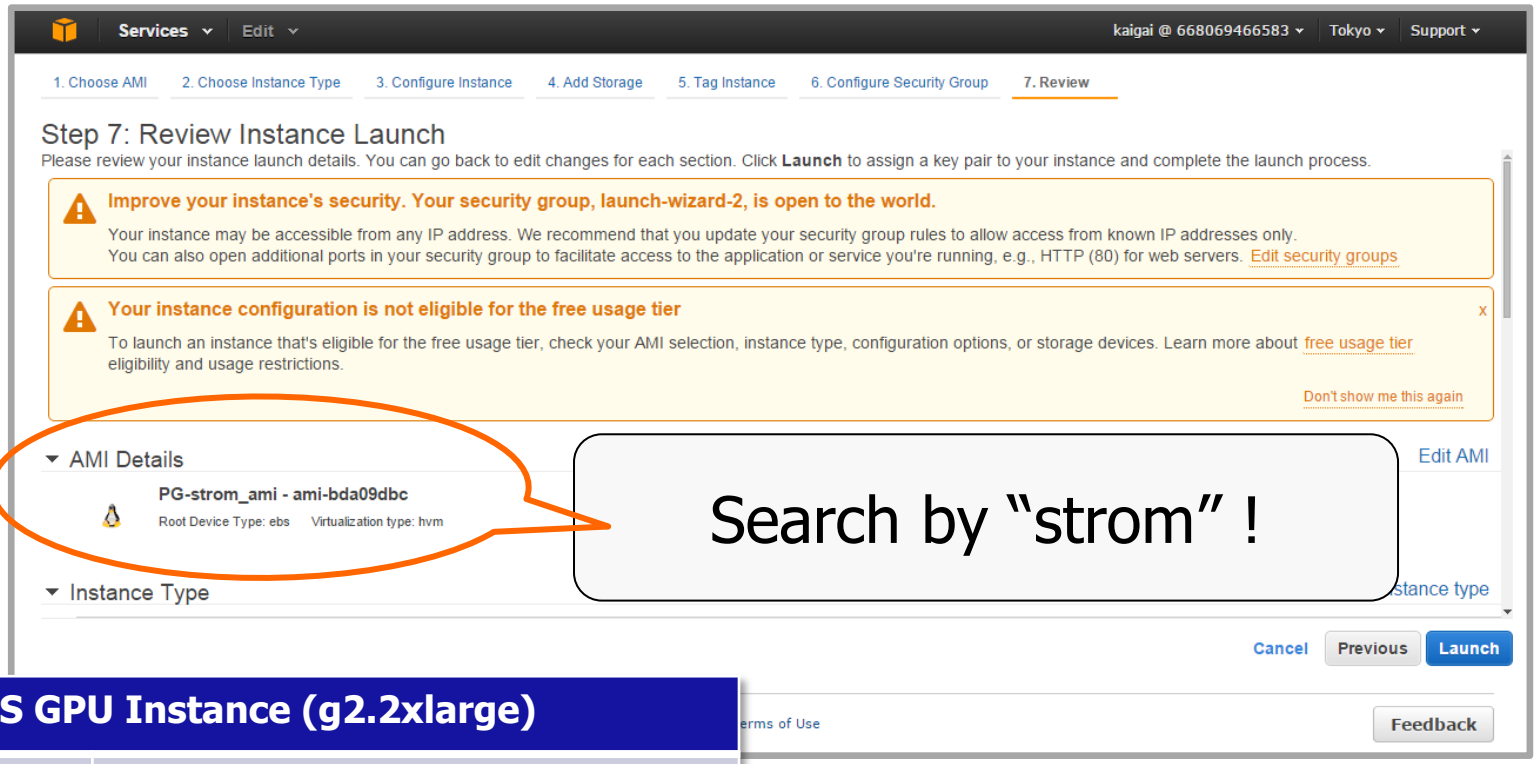
```
postgres=# SET pg_strom.enabled = on;
SET
```

How to use (2/3) – Build, Install and Starup

```
[kaigai@saba ~]$ git clone https://github.com/pg-strom/devel.git pg_strom
[kaigai@saba ~]$ cd pg_strom
[kaigai@saba pg_strom]$ make && make install
[kaigai@saba pg_strom]$ vi $PGDATA/postgresql.conf
```

```
[kaigai@saba ~]$ pg_ctl start
server starting
[kaigai@saba ~]$ LOG:  registering background worker "PG-Strom OpenCL Server"
LOG:  starting background worker process "PG-Strom OpenCL Server"
LOG:  database system was shut down at 2014-11-09 17:45:51 JST
LOG:  autovacuum launcher started
LOG:  database system is ready to accept connections
LOG:  PG-Strom: [0] OpenCL Platform: NVIDIA CUDA
LOG:  PG-Strom: (0:0) Device GeForce GTX 980 (1253MHz x 16units, 4095MB)
LOG:  PG-Strom: (0:1) Device GeForce GTX 750 Ti (1110MHz x 5units, 2047MB)
LOG:  PG-Strom: [1] OpenCL Platform: Intel(R) OpenCL
LOG:  PG-Strom: Platform "NVIDIA CUDA (OpenCL 1.1 CUDA 6.5.19)" was installed
LOG:  PG-Strom: Device "GeForce GTX 980" was installed
LOG:  PG-Strom: shmem 0x7f447f6b8000-0x7f46f06b7fff was mapped (len: 10000MB)
LOG:  PG-Strom: buffer 0x7f34592795c0-0x7f44592795bf was mapped (len: 65536MB)
LOG:  Starting PG-Strom OpenCL Server
LOG:  PG-Strom: 24 of server threads are up
```

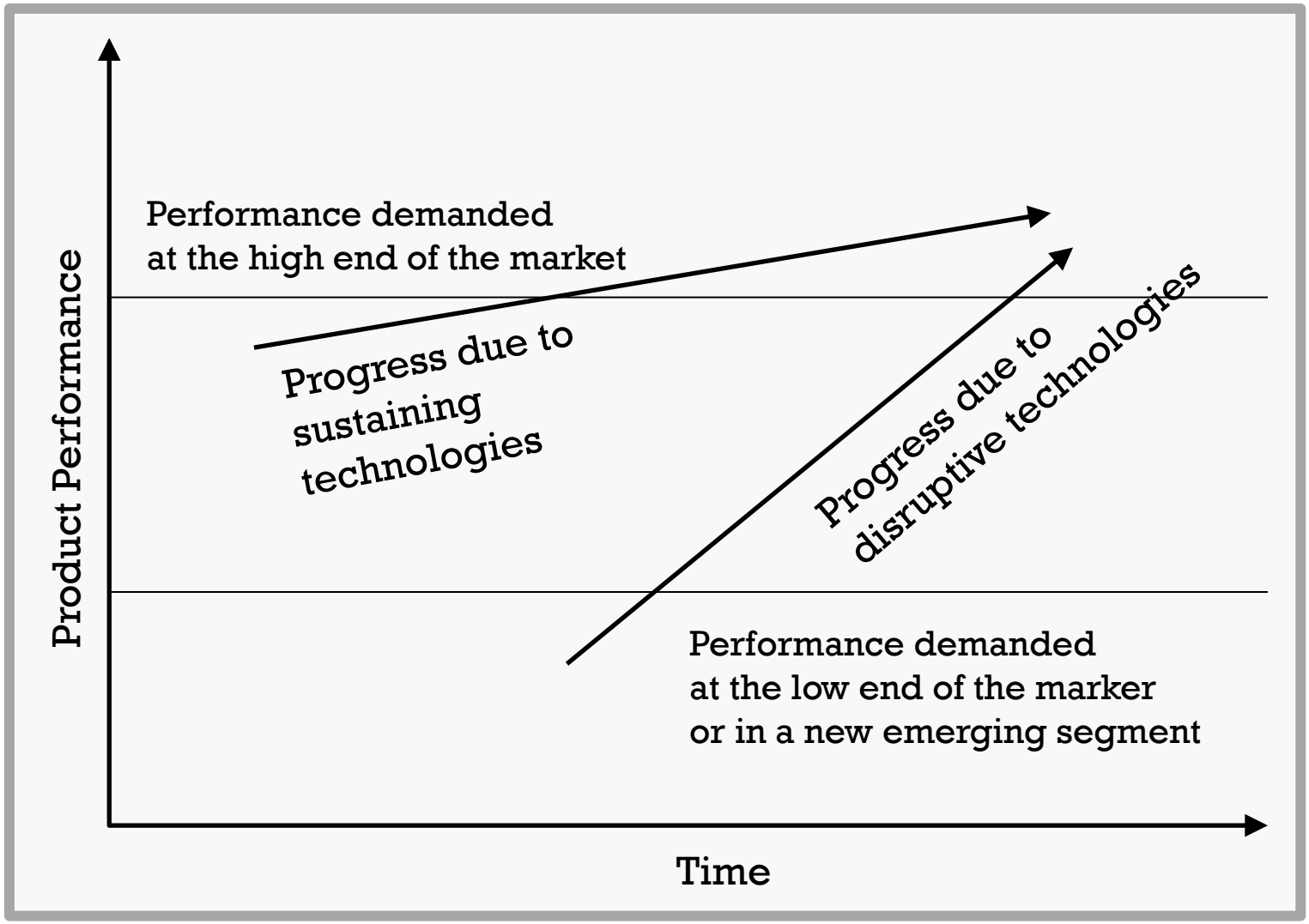
How to use (3/3) – Deployment on AWS



AWS GPU Instance (g2.2xlarge)	
CPU	Xeon E5-2670 (8 xCPU)
RAM	15GB
GPU	NVIDIA GRID K2 (1536core)
Storage	60GB of SSD
Price	$0.898/hour, $646.56/mon (*) Price for on-demand instance on Tokyo region at Nov-2014

Future of PG-Strom

Dilemma of Innovation

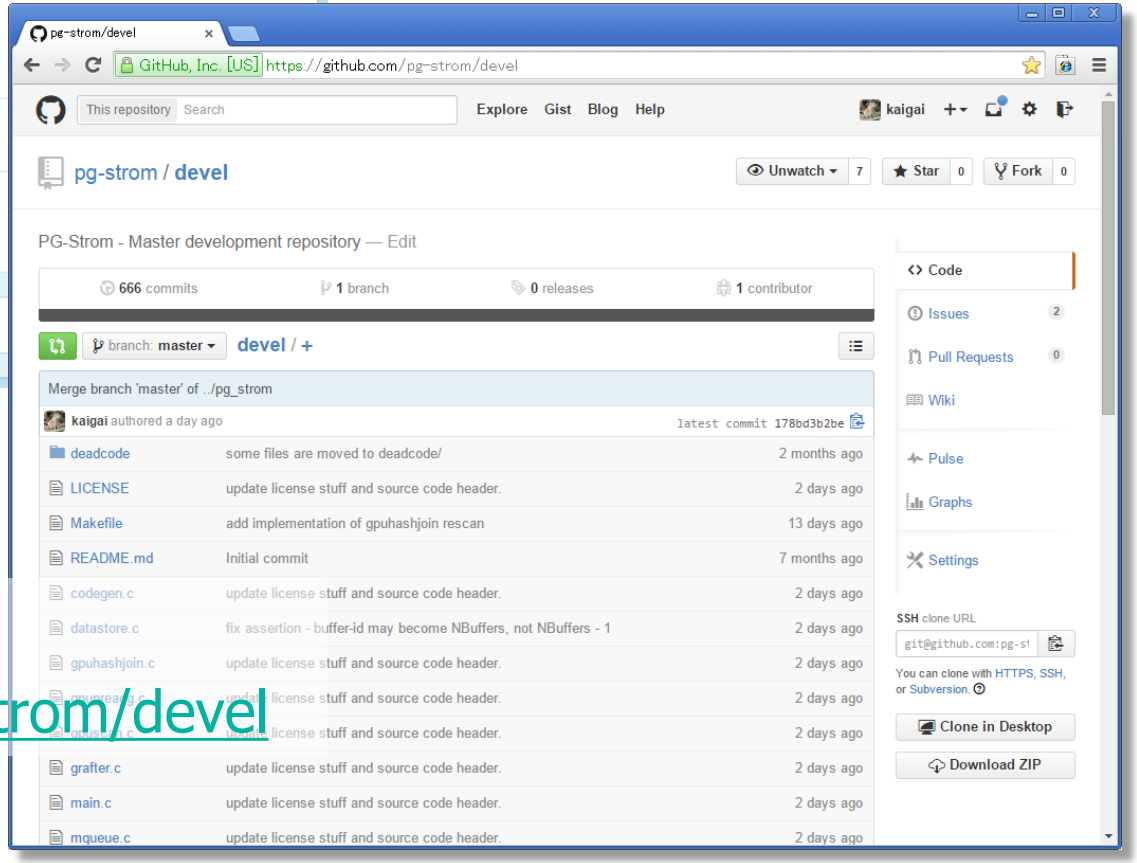


SOURCE: **The Innovator's Dilemma, Clayton M. Christensen**

Move forward with community

(Additional comment after the conference)

Oops, I forgot to say
in the conference!
PG-Strom is open source.



check it out!

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

Orchestrating a brighter world

世界の想いを、未来へつなげる。

未来に向かい、人が生きる、豊かに生きるために欠かせないもの。
それは「安全」「安心」「効率」「公平」という価値が実現された社会です。

NECは、ネットワーク技術とコンピューティング技術をあわせ持つ
類のないインテグレーターとしてリーダーシップを発揮し、
卓越した技術とさまざまな知見やアイデアを融合することで、
世界の国々や地域の人々と協奏しながら、
明るく希望に満ちた暮らしと社会を実現し、未来につなげていきます。

Empowered by Innovation