PG-Strom

GPU Accelerated Asynchronous Super-Parallel Query Execution

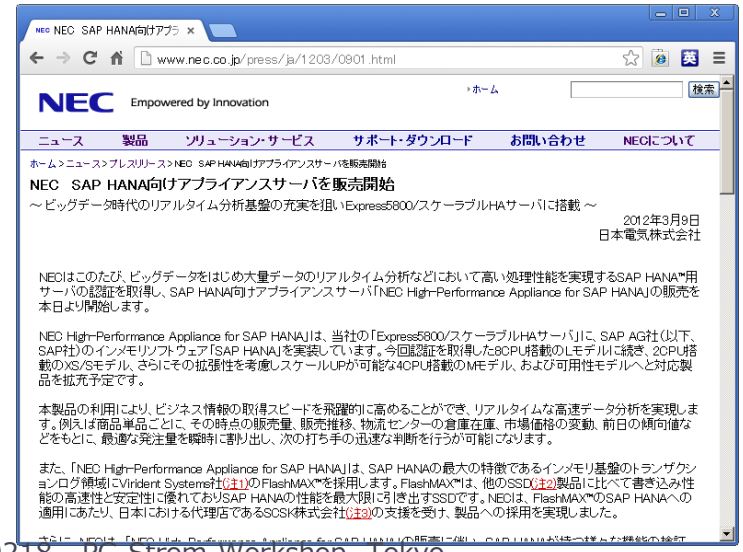
KaiGai Kohei <kaigai@kaigai.gr.jp>

(Tw: @kkaigai)

Self Introduction

- 名前　　海外 浩平 (KaiGai Kohei)
- 所属　　NEC Europe - SAP Global Competence Center
- 仕事　　OSS活用によるイノベーション創出 (20-30%)
  　　SAPとのアライアンス、PF製品の拡販 (70-80%)
  - SAPのIn-memory DB “SAP HANA” の認証作業とか
  - CLUSTERPROのSAP認定取得、拡販とか

特にコレの
割合がデカい



All everyone talks about BIG-DATA

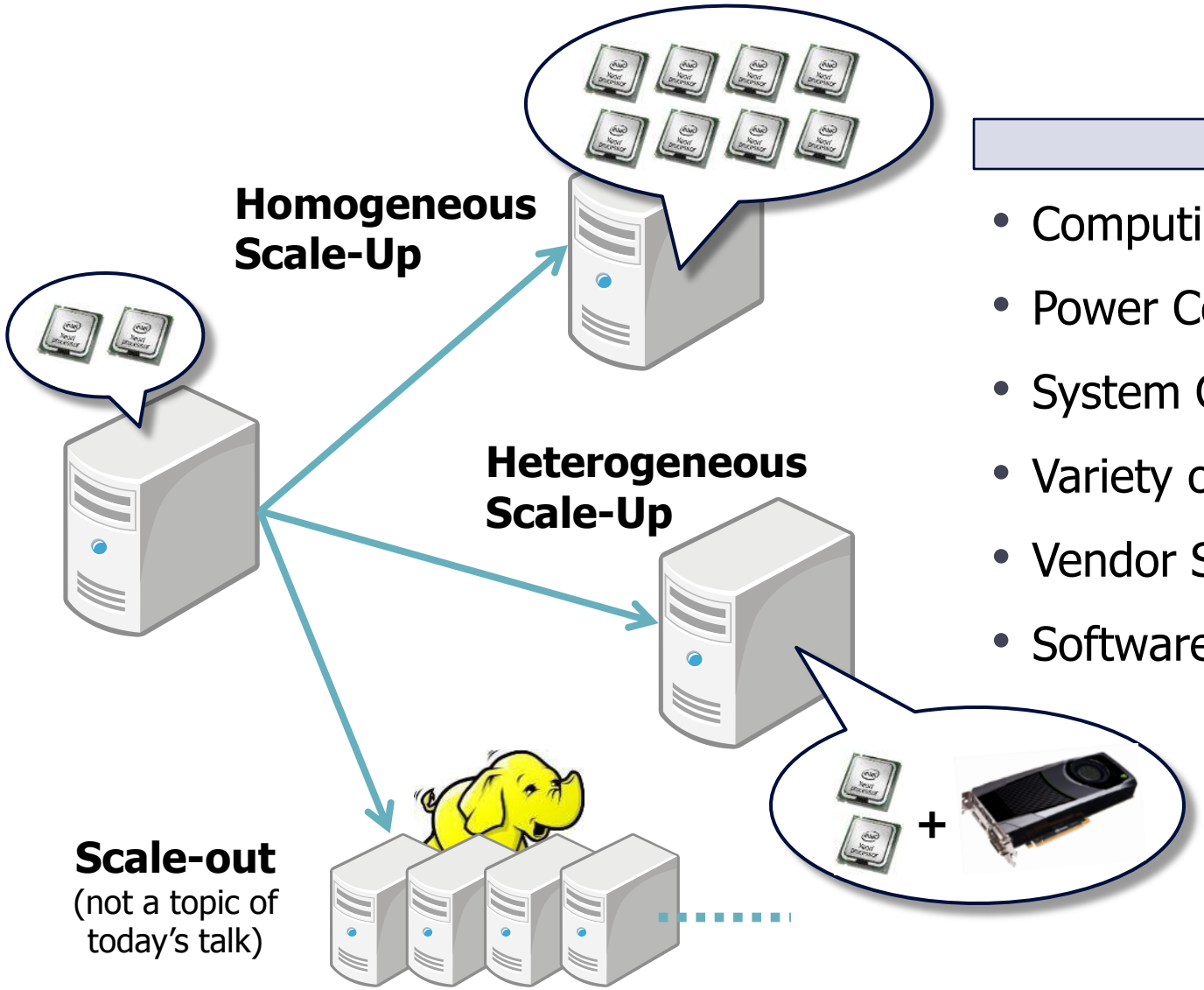
猫

杓子

熱い視線

Big-Data Database?

Homogeneous / Heterogeneous computing



KPIs

- Computing Performance
- Power Consumption
- System Cost (HW/SW)
- Variety of Applications
- Vendor Support
- Software Development

:

Design concept of PG-Strom

~~The world cheapest~~

The most Cost-Effective Big-Data Database

- Utilization of open source technology
- Utilization of commodity hardware
  - up-to ?? CPUs
  - up-to ??? GB RAM
  - up-to ??? Data size

  まだ、この辺をとやかく言える段階ではない
- Utilization of heterogeneous computing with GPU

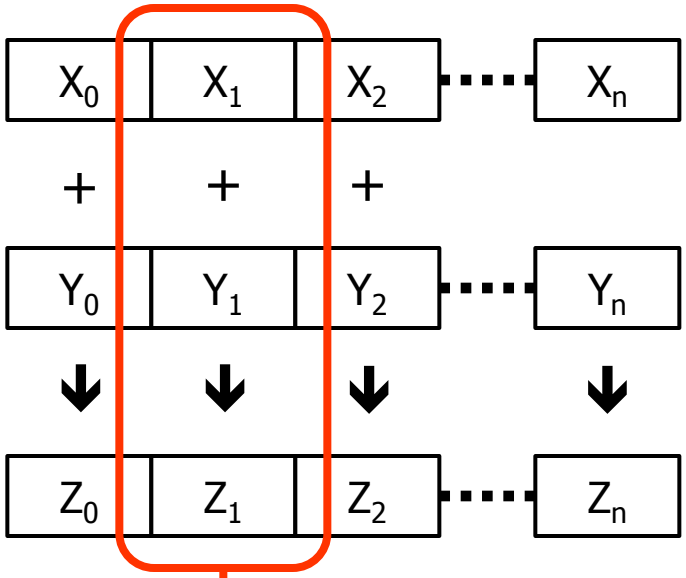
Characteristics of GPU (1/2)

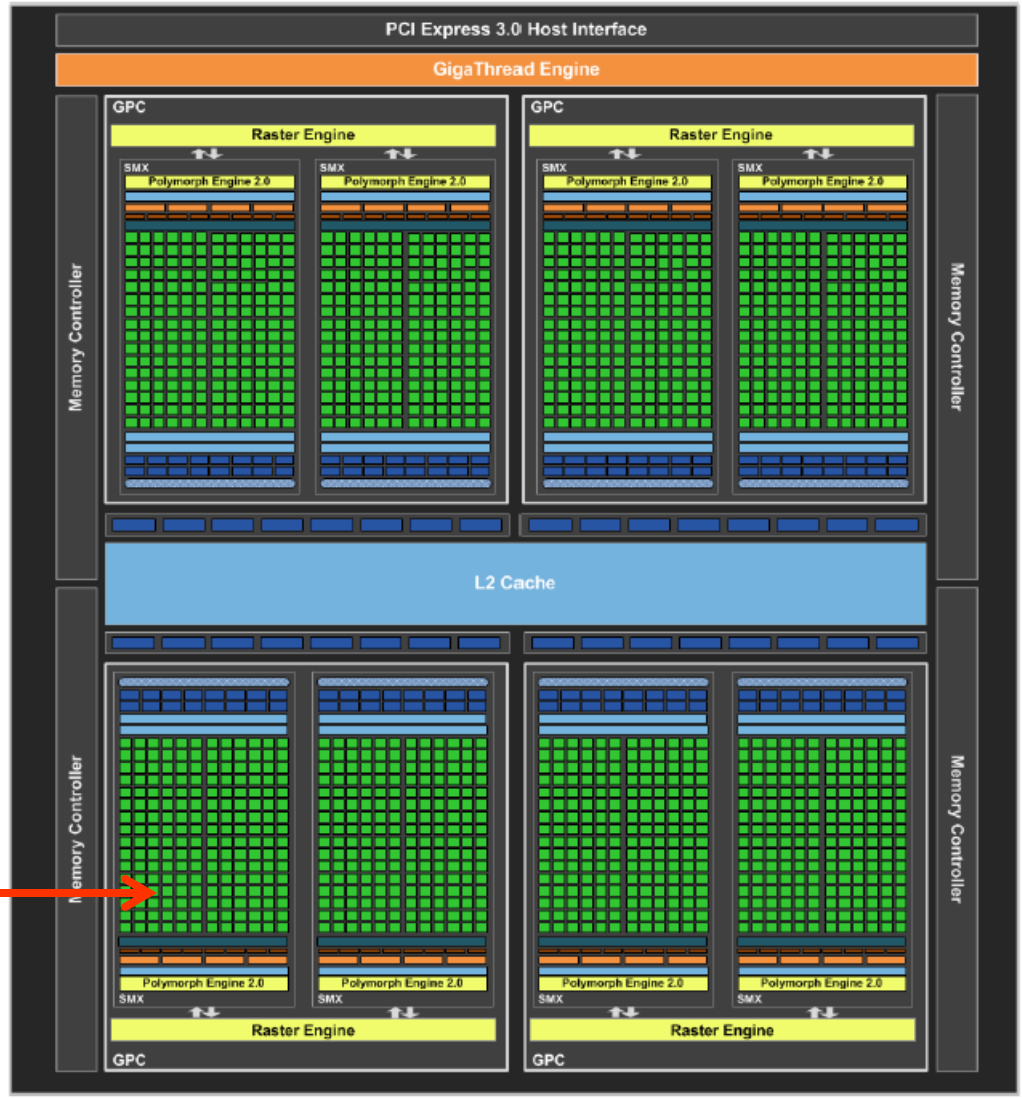
	Nvidia Kepler	AMD GCN	Intel SandyBridge
Model	Tesla K20X (Q4/2012)	FirePro S9000 (Q3/2012)	Xeon E5-2690 (Q1/2012)
Number of Transistors	7.1billion	4.3billion	2.26billion
Number of Cores	**2688** Simple	**1792** Simple	**16** Functional
Core clock	732MHz	925MHz	2.9GHz
Peak FLOPS	3.95Tflops	3.23TFlops	185.6GFlops
Memory Size / TYPE	6GB, GDDR5	6GB, GDDR5	384GB/socket, DDR3
Memory Bandwidth	~192GB/s	~264GB/s	~51.2GB/s
Power Consumption	~235W	~225W	~135W
Price	$3199?	$2499?	$2061

Characteristics of GPU (2/2)

Example)

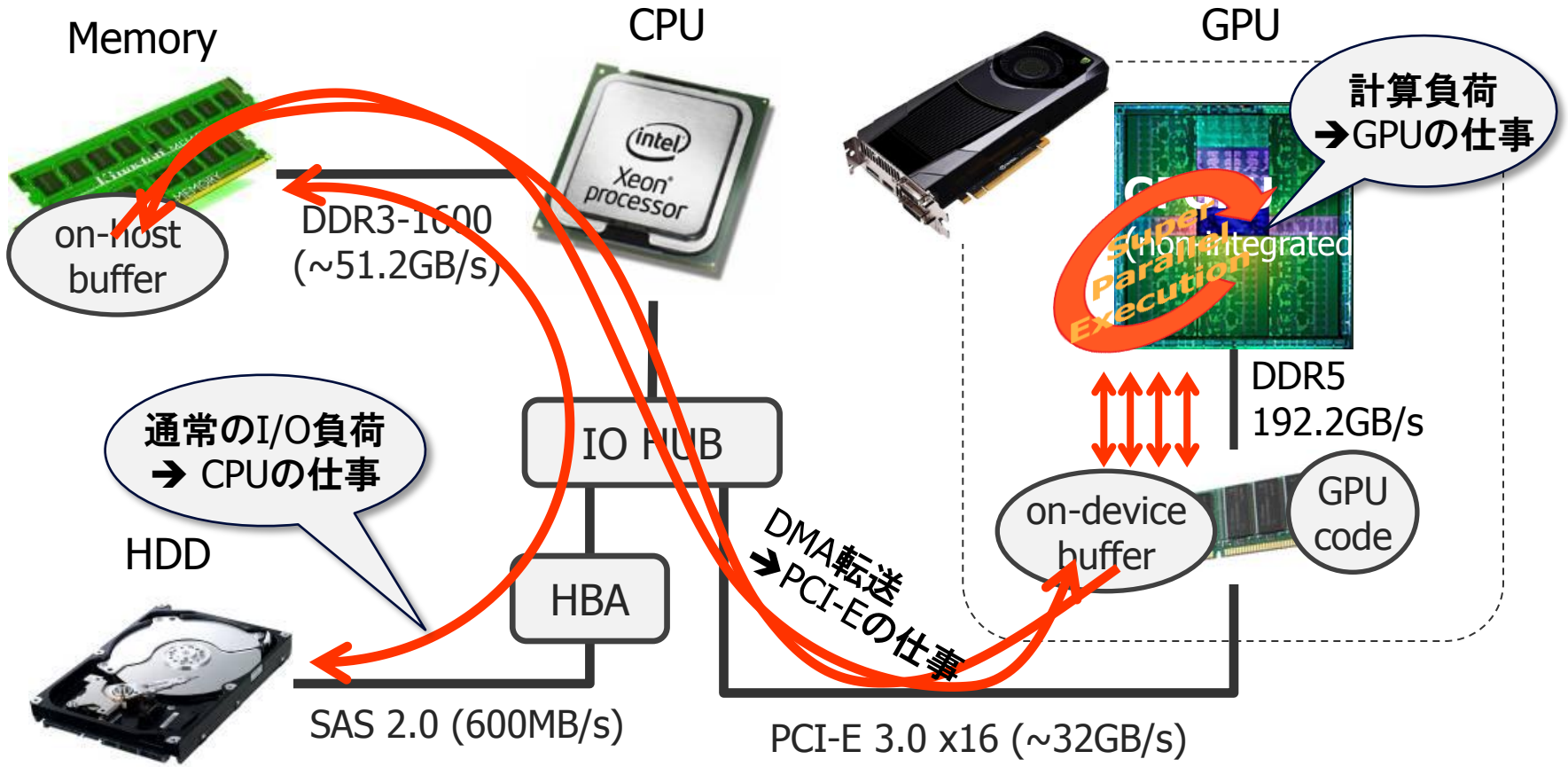
$$Z_i = X_i + Y_i \quad (0 <= i <= n)$$





Nvidia's GeForce GTX 680 Block Diagram

Play with GPU (1/3)



- Asynchronous Execution of CPU, GPU and PCI-E
- Minimization of data transfer between host and device

Play with GPU (2/3)

Host code example

```
void sqrt_float4(int n, float v[])
{
  /* Acquire device memory and data transfer (host -> device) */
  dev_v = clCreateBuffer(cxt, CL_MEM_READ_WRITE,
                         sizeof(float) * n, NULL, &rv);
  /* Enqueue data transfer host to device */
  clEnqueueWriteBuffer(cmdq, dev_x, CL_TRUE, 0, NULL,
                       v, 0, NULL, NULL);
  /* Set arguments of kernel code */
  clSetKernelArg(kernel, 0, sizeof(int), (void *)&n);
  clSetKernelArg(kernel, 1, sizeof(cl_mem), (void *)&dev_v);
  /* Enqueue invocation of device kernel */
  clEnqueueNDRangeKernel(cmdq, kernel, 1, NULL, &g_itemsz, &l_itemsz,
                         0, NULL, NULL);
  /* Enqueue data transfer device to host */
  clEnqueueReadBuffer(cmdq, dev_x, CL_TRUE, 0, NULL,
                      v, 0, NULL, NULL);
  /* Release device memory */
  clReleaseMemObject(dev_x)
}
```

Play with GPU (3/3)

Device code example

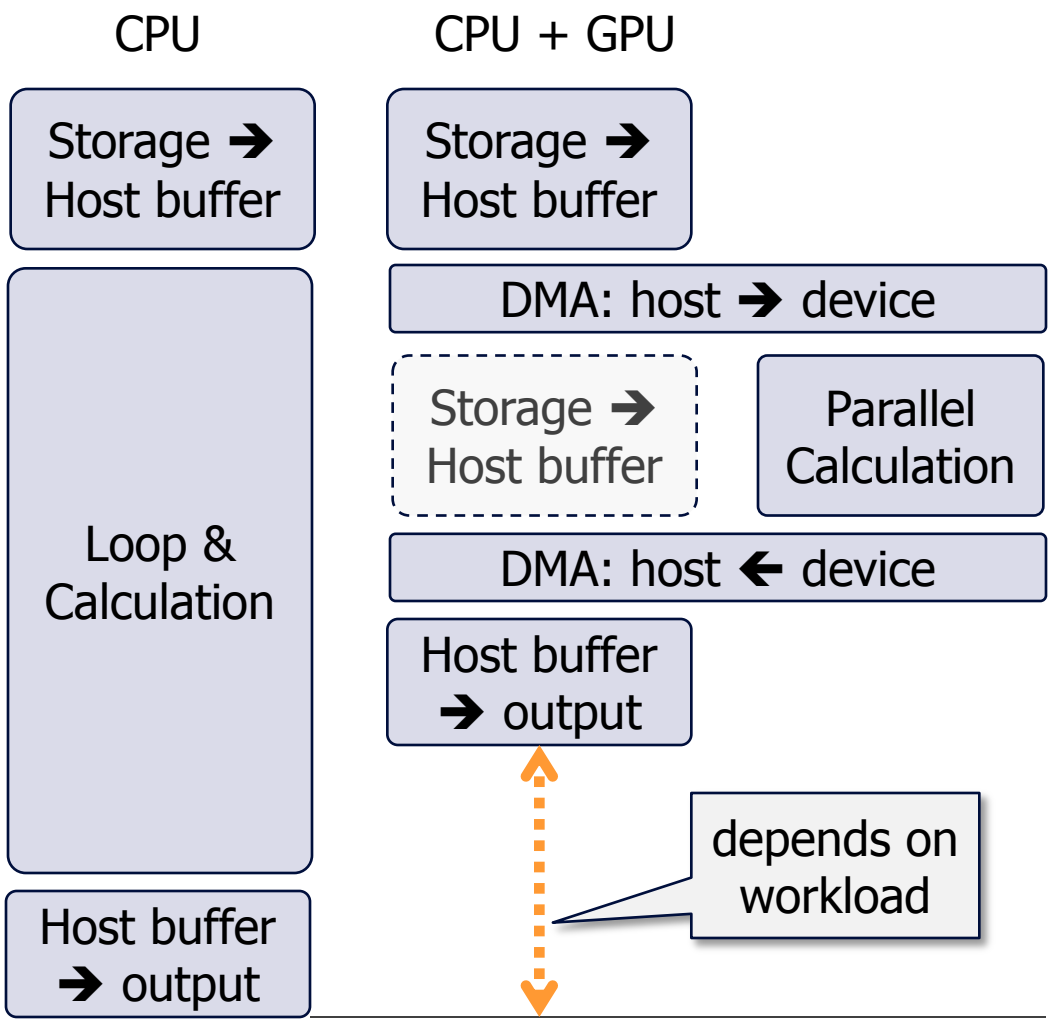
```
__kernel void dev_sqrt_float(int length, float x[])
{
  int i = get_global_id(0);

  if (i < length)
    x[i] = sqrt(x[i]);
}
```

Host code to load kernel

```
/* Load source code of the program */
program = clCreateProgramWithSource(cxt, 1,
                        (const char *)&kernel_source,
                        (const size_t *)&kernel_source_len, &rv);
/* Run-time build of the program */
rv = clBuildProgram(program, 1, &device_id, NULL, NULL, NULL);
/* Create a device kernel object */
kernel = clCreateKernel(program, “dev_sqrt_float”, &rv);
```

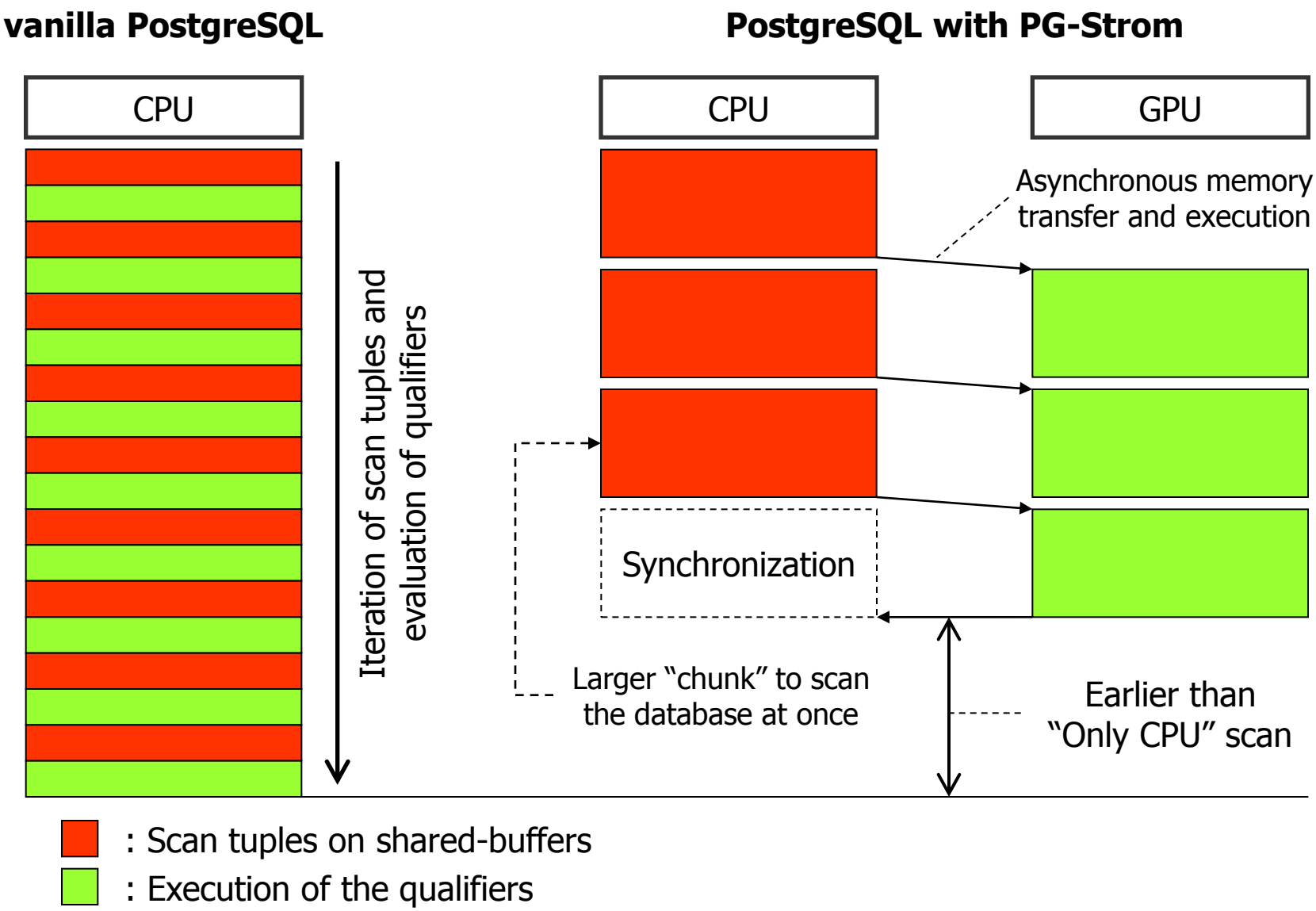
Comparison - CPU vs CPU + GPU



- Advantage
  - シンプルな演算の超並列処理
  - 非同期実行＆パイプライン処理
- Disadvantage
  - ホスト⇔デバイス間のDMA転送コスト
  - コードの複雑化

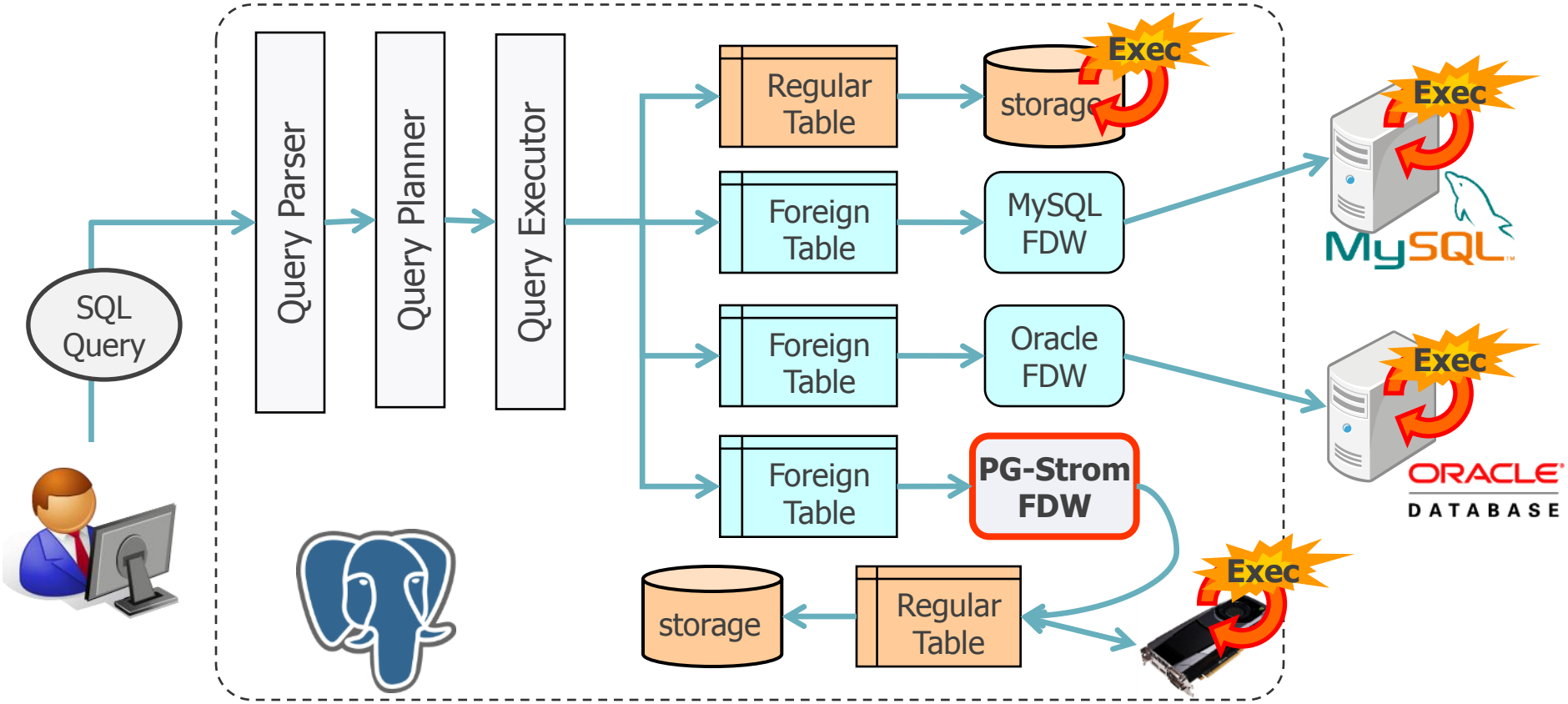
Architecture of PG-Strom

PG-Strom's Asynchronous Execution model

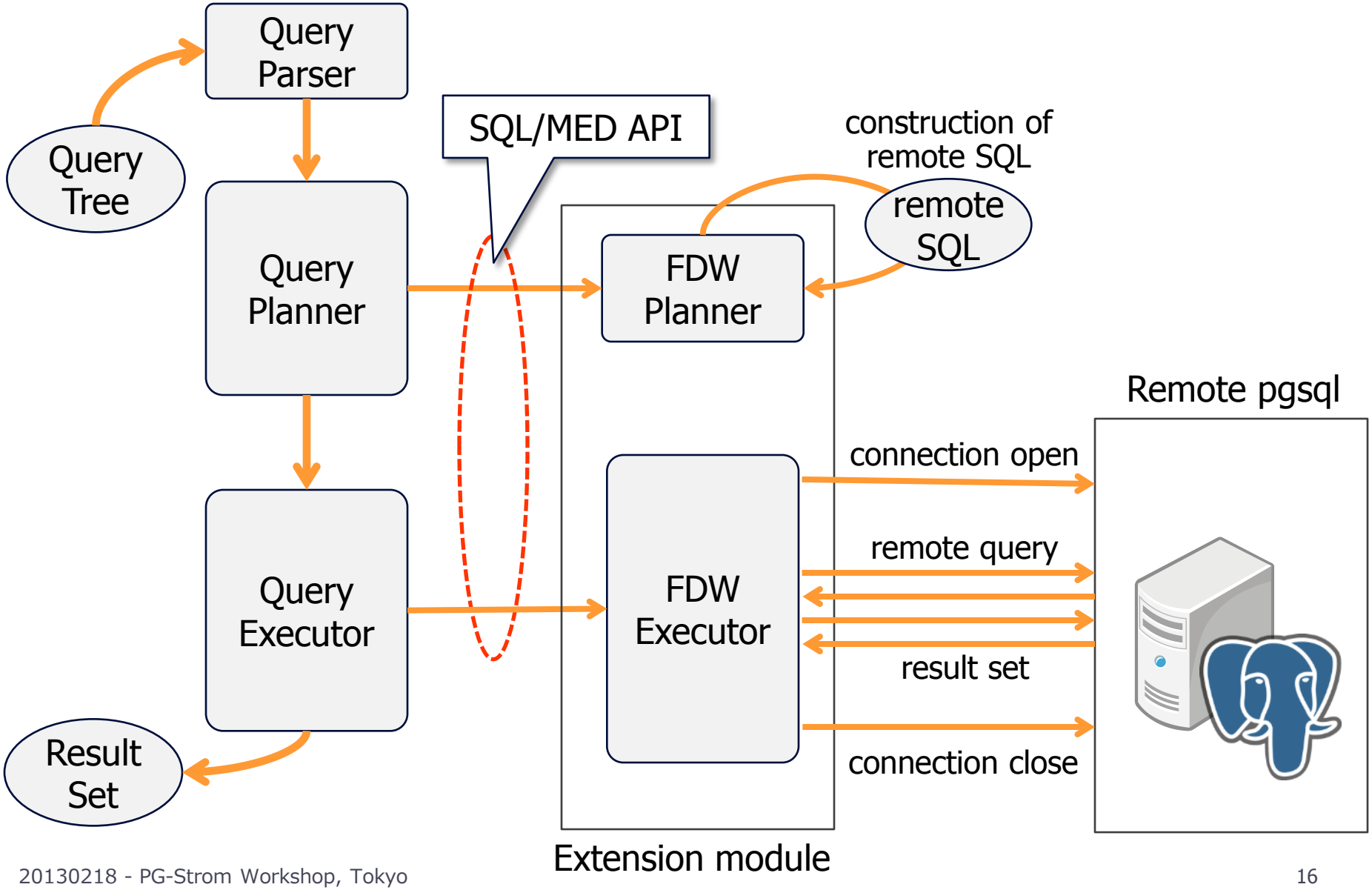


Re-definition of SQL/MED (1/2)

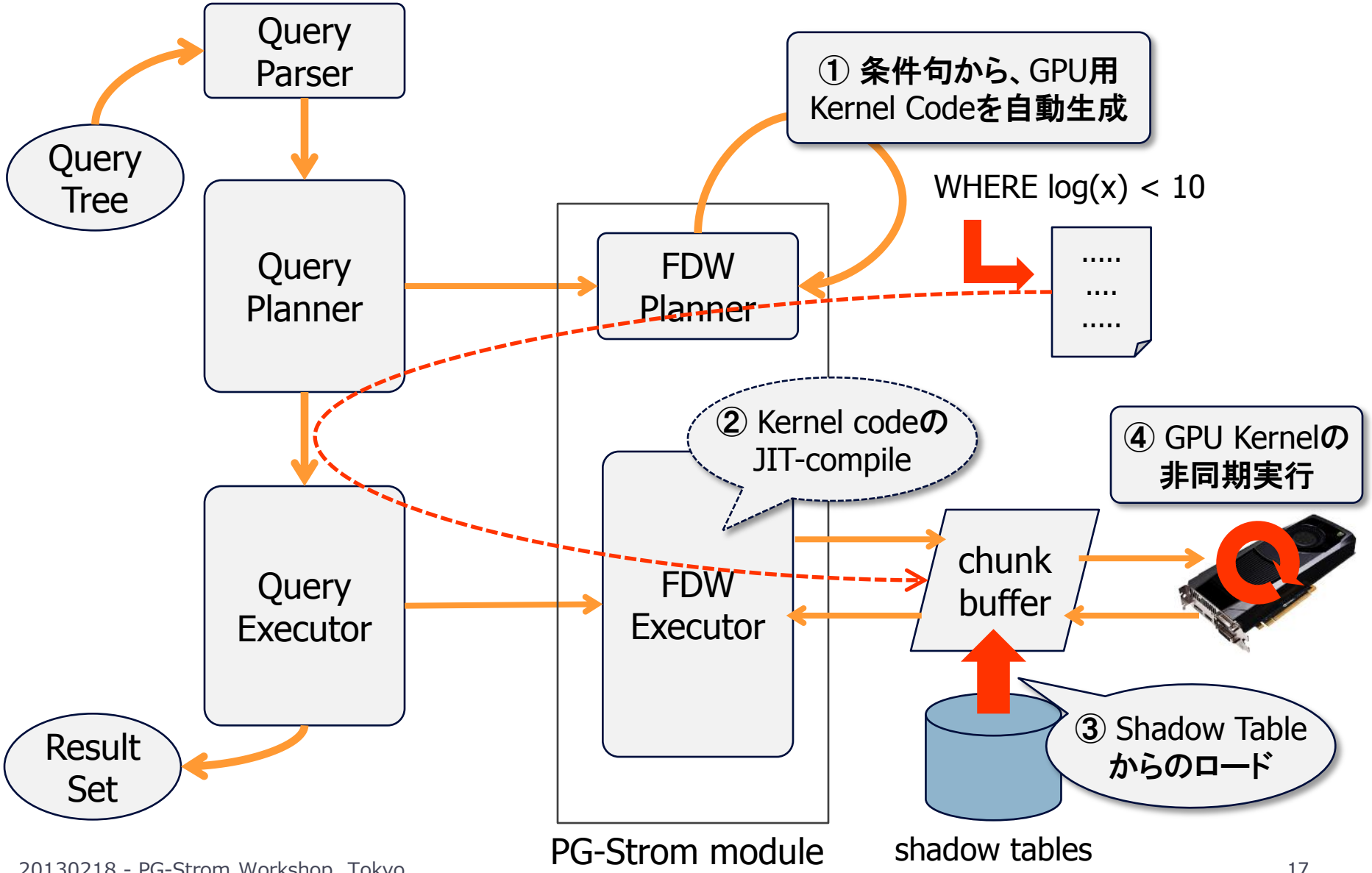
- SQL/MED (Management of External Data)
  - External data source performing as if regular tables
  - Not only "management", but external computing resources also



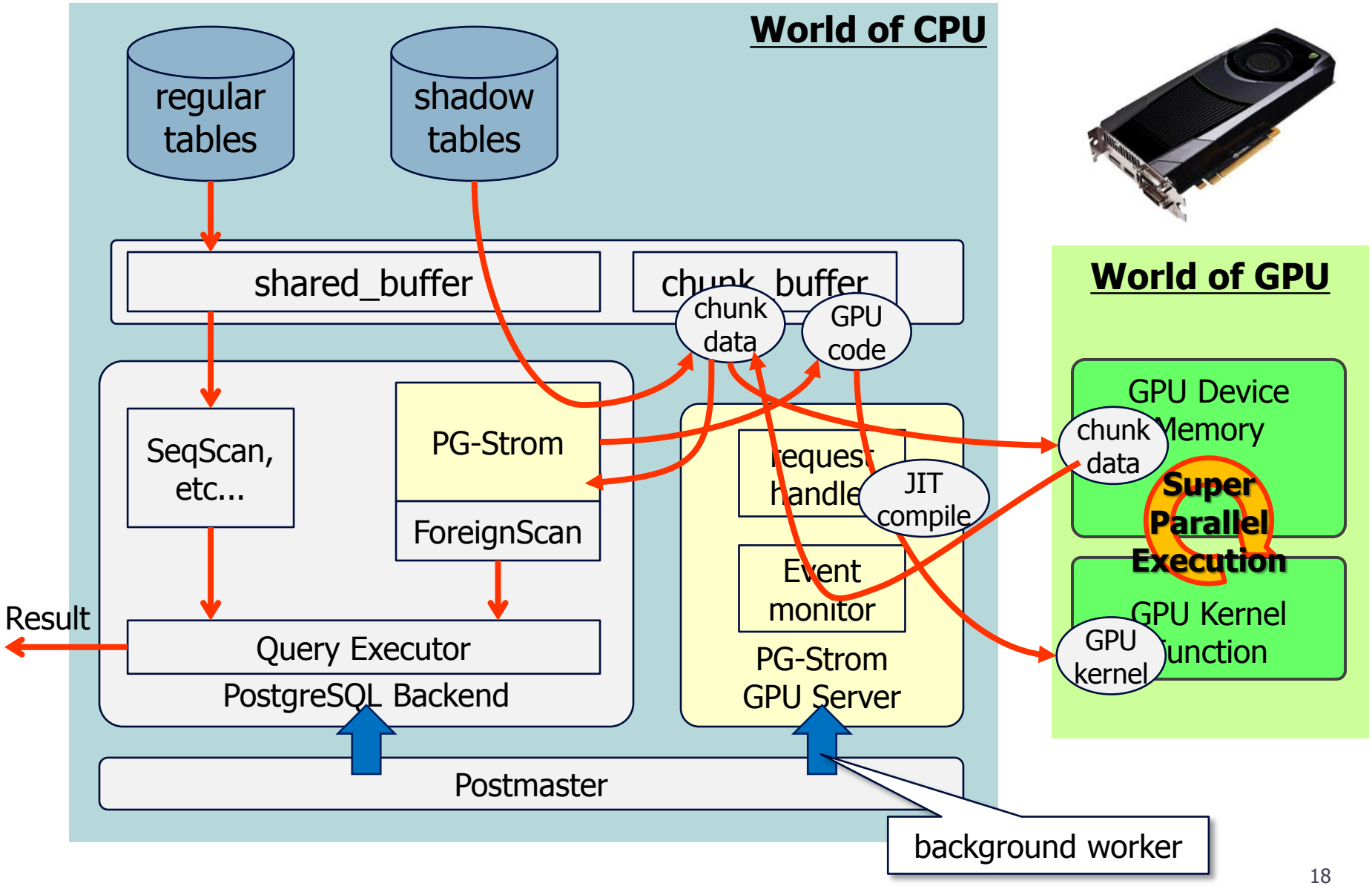
Re-definition of SQL/MED (2/2)



PG-Strom as SQL/MED driver



Overall architecture



So what, How fast is it?

```
postgres=# SELECT COUNT(*) FROM rtbl
           WHERE sqrt((x-256)^2 + (y-128)^2) < 40;
 count
--------
 100467
(1 row)
Time: 7668.684 ms

postgres=# SELECT COUNT(*) FROM ftbl
           WHERE sqrt((x-256)^2 + (y-128)^2) < 40;
 count
--------
 100467
(1 row)
Time: 857.298 ms
```

Accelerated!

- CPU: Xeon E5-2670 (2.60GHz), GPU: NVidia GeForce GT640, RAM: 384GB
- Both of regular **rtbl** and PG-Strom **ftbl** contain 20milion rows with same value

Key Technologies

- Automatic GPU code generation & JIT compile
- Column-oriented data structure
- Asynchronous Execution

Automatic "pseudo" code generation

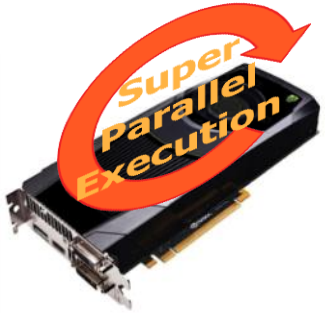
```
SELECT * FROM ftbl WHERE
c like '%xyz%' AND sqrt((x-256)^2+(y-100)^2) < 10;
```

contains unsupported operators / functions

Translation to pseudo code

```
xreg10 = $(ftbl.x)
xreg12 = 256.000000::double
xreg8 =  (xreg10 - xreg12)
xreg10 = 2.000000::double
xreg6 = pow(xreg8, xreg10)
xreg12 = $(ftbl.y)
xreg14 = 128.000000::double
        :
```

Pseudo-code based implementation will be replaced by native code and JIT-compile approach soon.

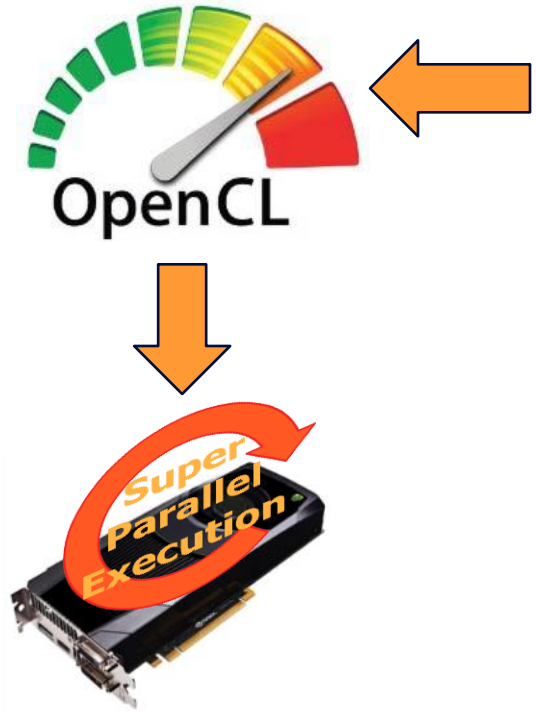


Automatic native code generation - WIP

SELECT * FROM ftbl WHERE

c like '%xyz%' AND sqrt((x-256)^2+(y-100)^2) < 10;

OpenCL run-time builds native GPU binary



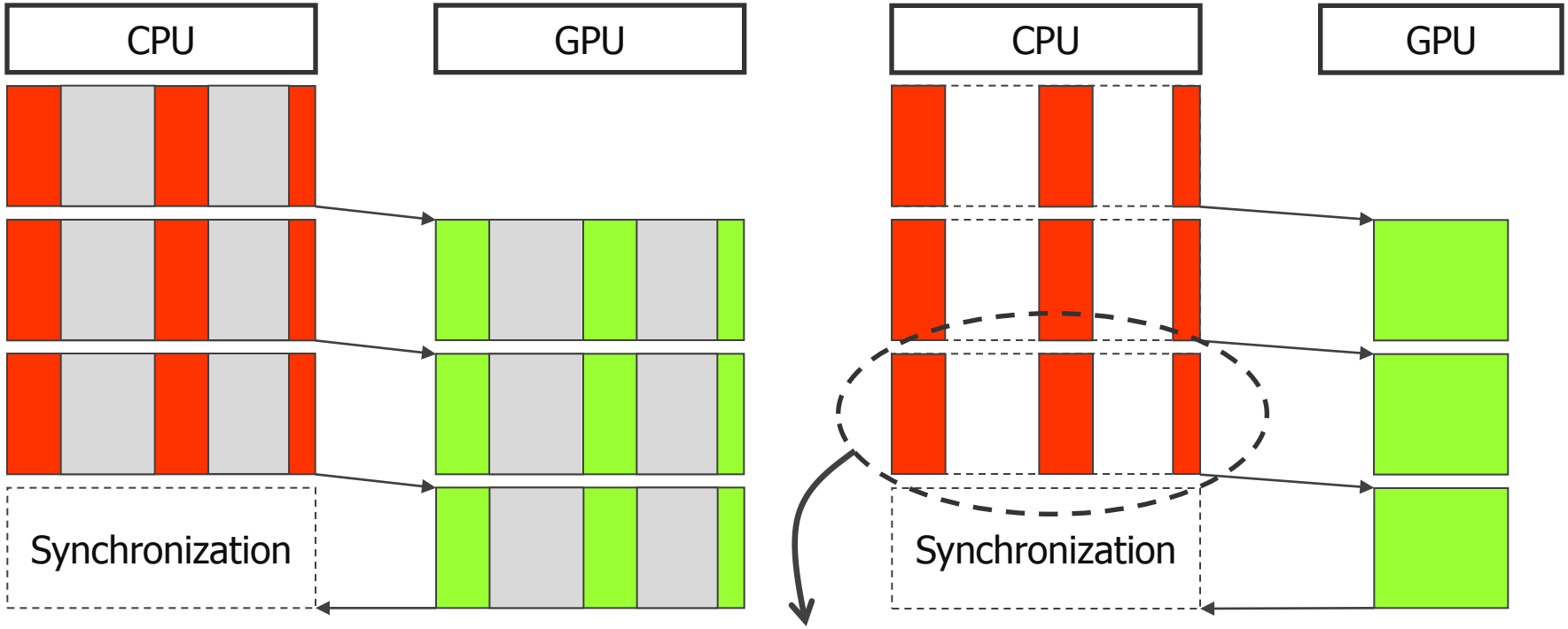

```
__kernel void
pgstrom_qual(int nitems, bool result[],
             float x[], float y[])
{
    int  index = get_global_id(0);

    if (sqrt(pow(x[i] - 256, 2) +
             pow(y[i] - 100, 2)) < 10)
        result[i] = true;
    else
        result[i] = false;
}
```

Save bandwidth & shared-buffer usage

E.g) SELECT name, tel, email, address FROM address_book
WHERE sqrt((**pos_x** – 24.5)^2 + (**pos_y** – 52.3)^2) < 10;

➔ No sense to fetch columns being not in use



: Scan tuples on the shared-buffers
: Execution of the qualifiers
: Columns being not used the qualifiers

✓ Save the bandwidth of PCI-E bus
✓ Save the shared-buffer usage

Column-oriented data structure (1/3)

FOREIGN TABLE ft

int X	float Y	text Z

(shadow) TABLE "public.ft.rowid"

rowid	nitems	isnull
4000	2000	{0,0,0,1,0,0,...}
6000	2000	{0,0,0,0,0,0,...}
:	:	:

(shadow) TABLE "public.ft.z.cs"

rowid	nitems	isnull	values
4000	15	{0,0,...}	{ 'hello', 'world', ... }
			, 'ccc', ... }
			, 'zzz', ...}

(shadow) TABLE "public.ft.y.cs"

rowid	nitems	isnull	values
4000	250	{0,0,...}	{ 1.38, 6.45, 2.15, ... }
4250	250	{0,1,...}	{ 4.32, 5.46, 3.14, ... }
			:
			39, 49, 59, ...}

(shadow) TABLE "public.ft.a.cs"

rowid	nitems	isnull	values
4000	500	{0,0,...}	{10, 20, 30, 40, 50, ...}
4500	500	{0,1,...}	{11, 0, 31, 41, 51, ...}
:	:	:	:
14200	200	{0,0,...}	{19, 29, 39, 49, 59, ...}

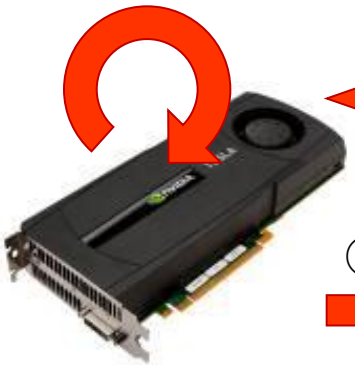
Column-oriented data structure (2/3)

```
postgres=# CREATE FOREIGN TABLE example
             (a int, b text) SERVER pg_strom;
CREATE FOREIGN TABLE

postgres=# SELECT * FROM pgstrom_shadow_relations;
  oid  |       relname        | relkind |  relsize
-------+----------------------+---------+-----------
 16446 | public.example.rowid | r       |         0
 16449 | public.example.idx   | i       |      8192
 16450 | public.example.a.cs  | r       |         0
 16453 | public.example.a.idx | i       |      8192
 16454 | public.example.b.cs  | r       |         0
 16457 | public.example.b.idx | i       |      8192
 16462 | public.example.seq   | S       |      8192
(9 rows)

postgres=# SELECT * FROM pg_strom."public.example.a.cs" ;
 rowid | nitems | isnull | values
-------+--------+--------+--------
(0 rows)
```

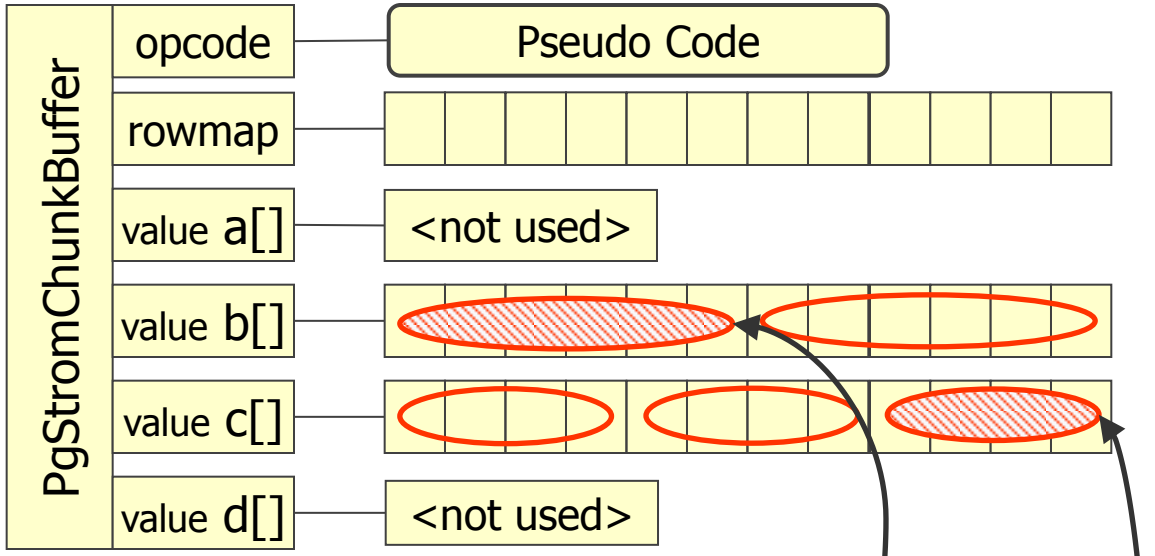
Column-oriented data structure (3/3)



② Calculation

① Transfer

③ Write-Back

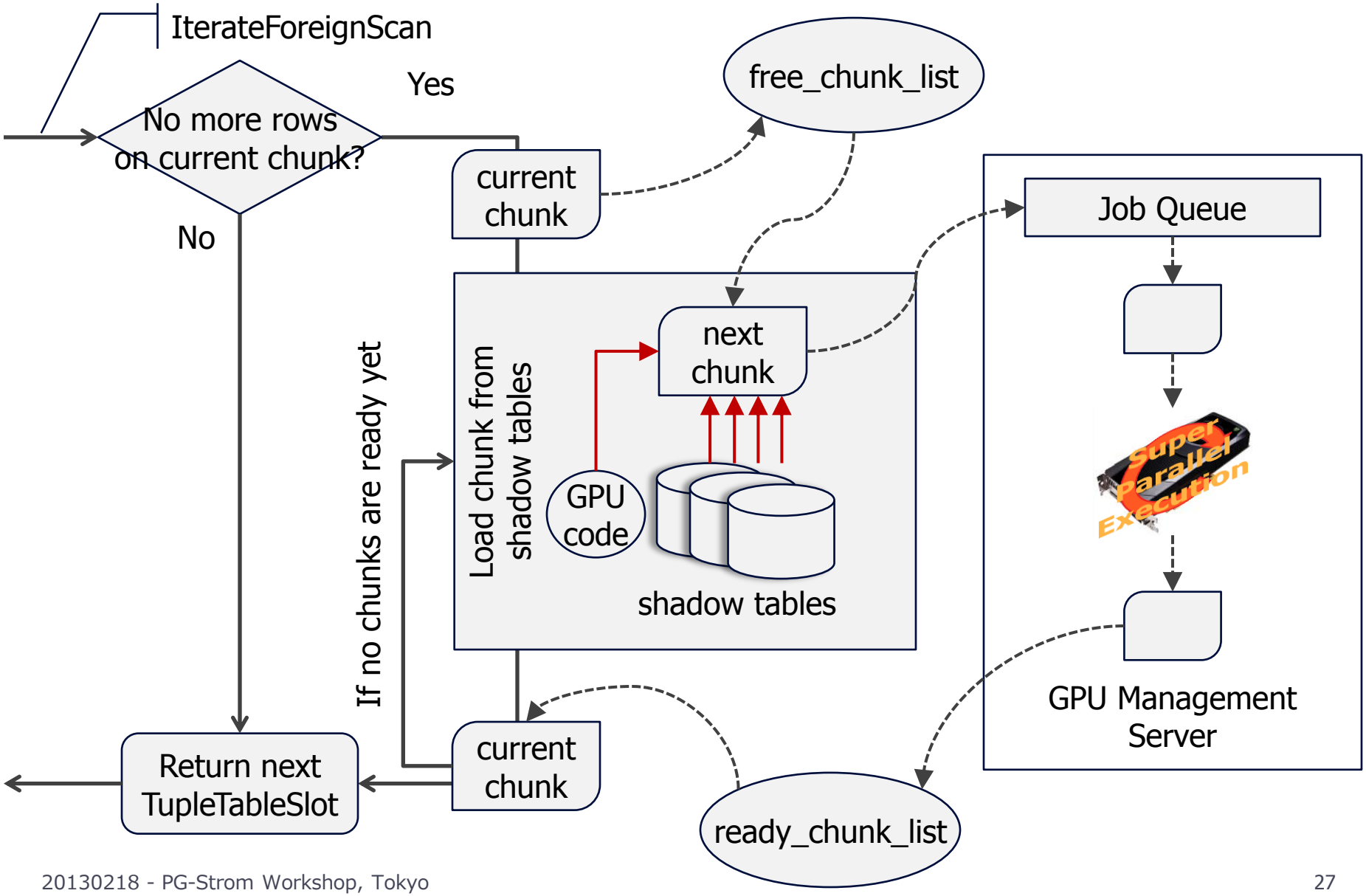


- Less bandwidth consumption of PCI-Express bus
- Less usage of buffer-cache
- Suitable for data compression

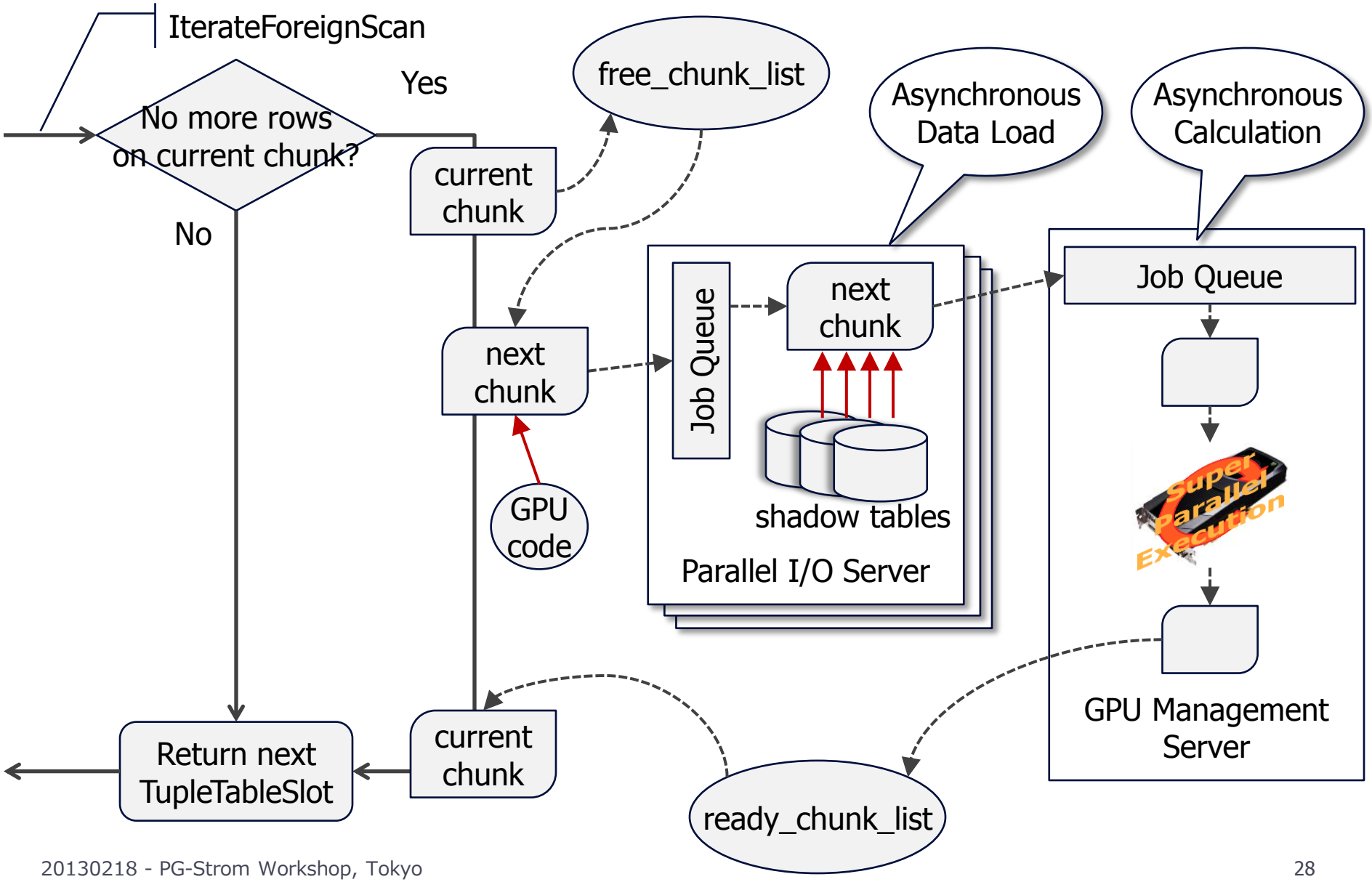
Table: my_schema.ft1.b.cs	
10100	{2.4, 5.6, 4.95, … }
10300	{10.23, 7.54, 5.43, … }

Table: my_schema.ft1.c.cs	
10100	{'2010-10-21', …}
10200	{'2011-01-23', …}
10300	{'2011-08-17', …}

Asynchronous Execution (1/2)

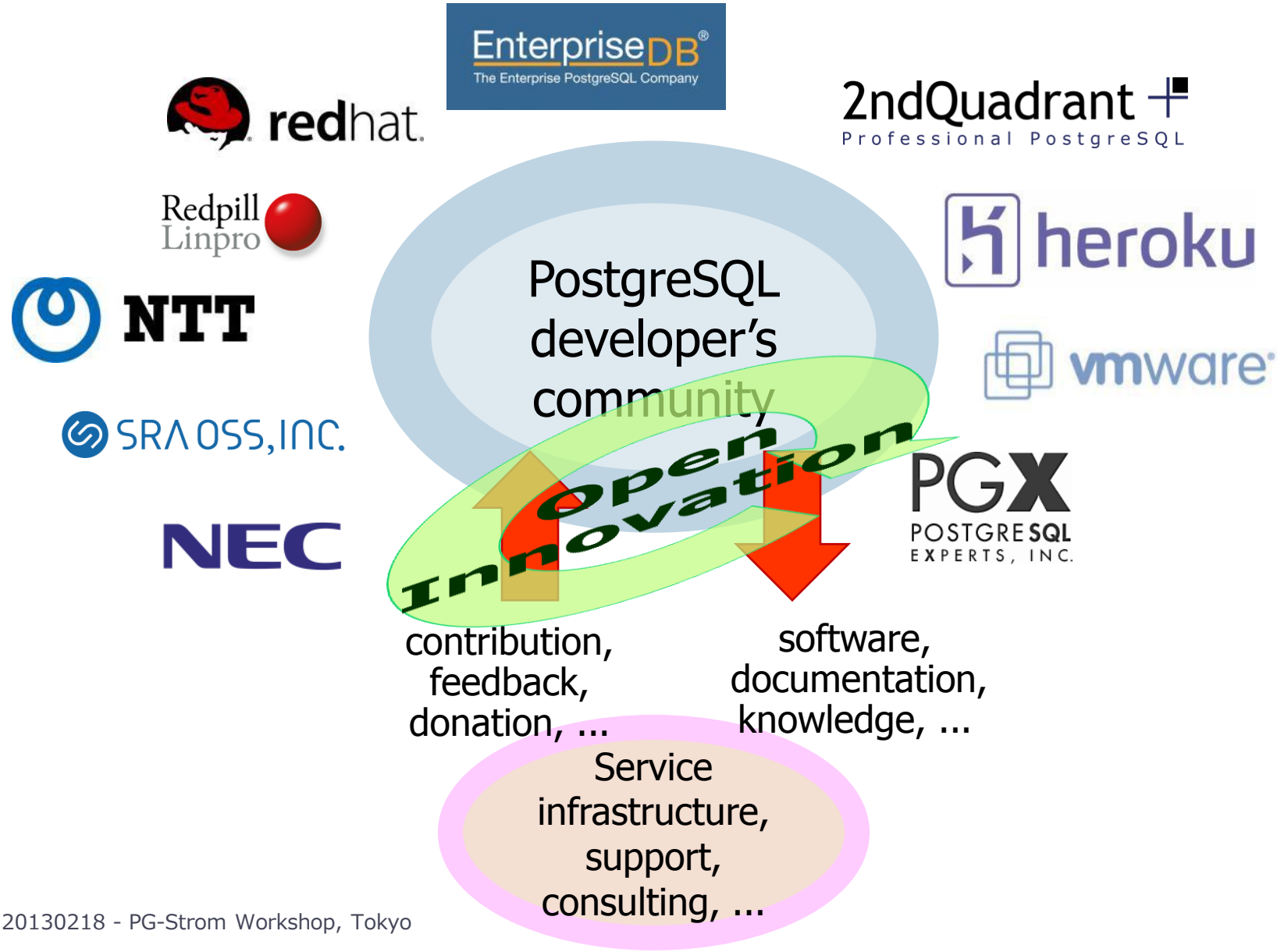


Asynchronous Execution (2/2) - in the future

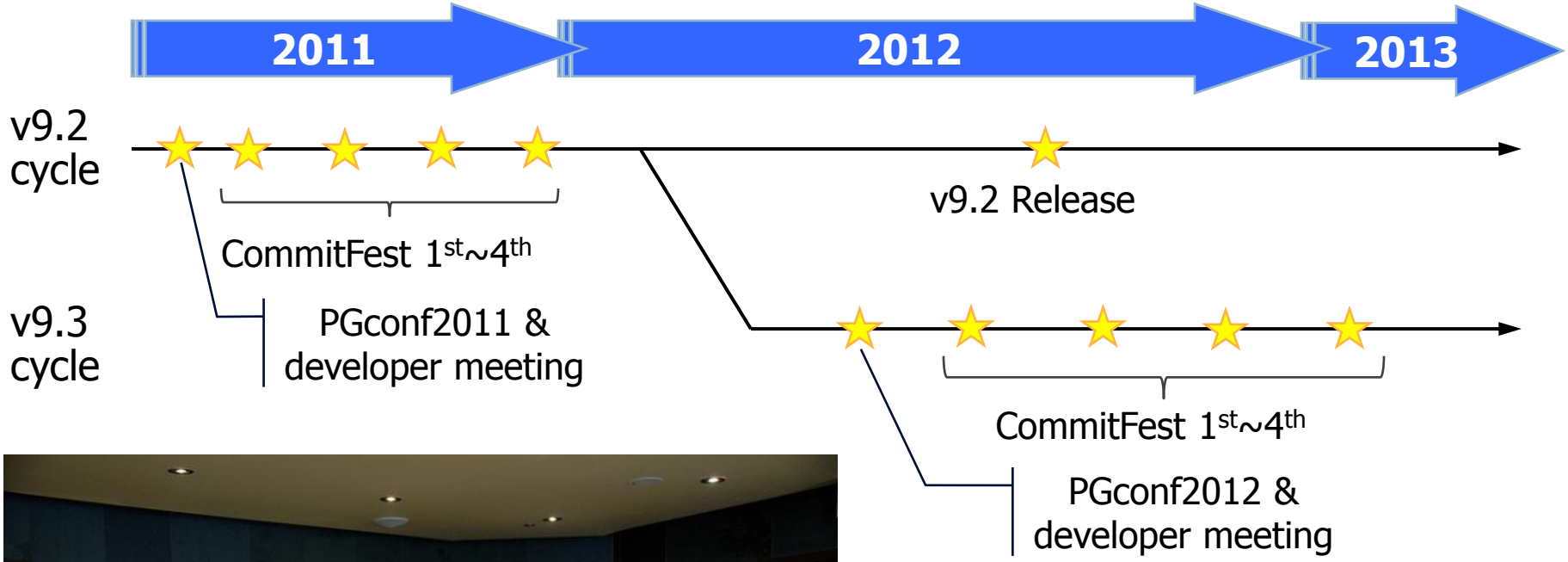


Eco-System in PostgreSQL Development

PostgreSQL developer's community



PostgreSQL development cycle



PostgreSQL developer meeting (17th-May-2012)

v9.3 development schedule

- 17th-May developer meeting
- 15th-Jun CommitFest:1st
- 15th-Sep CommitFest:2nd
- 15th-Nov CommitFest:3rd
- 15th-Jan CommitFest:4th

PostgreSQL CommitFest

PostgreSQL CommitFest

https://commitfest.postgresql.org/action/commitfest_view?id=17

Welcome, kaigai - Log Out - Home Page

PostgreSQL

CommitFest 2013-01 (In Progress)

New Patch - Activity Log - CommitFest Topics

The most recent three comments for each patch will be displayed below. To view all the comments for a particular patch, or to add a comment or make other changes, click on the patch name.

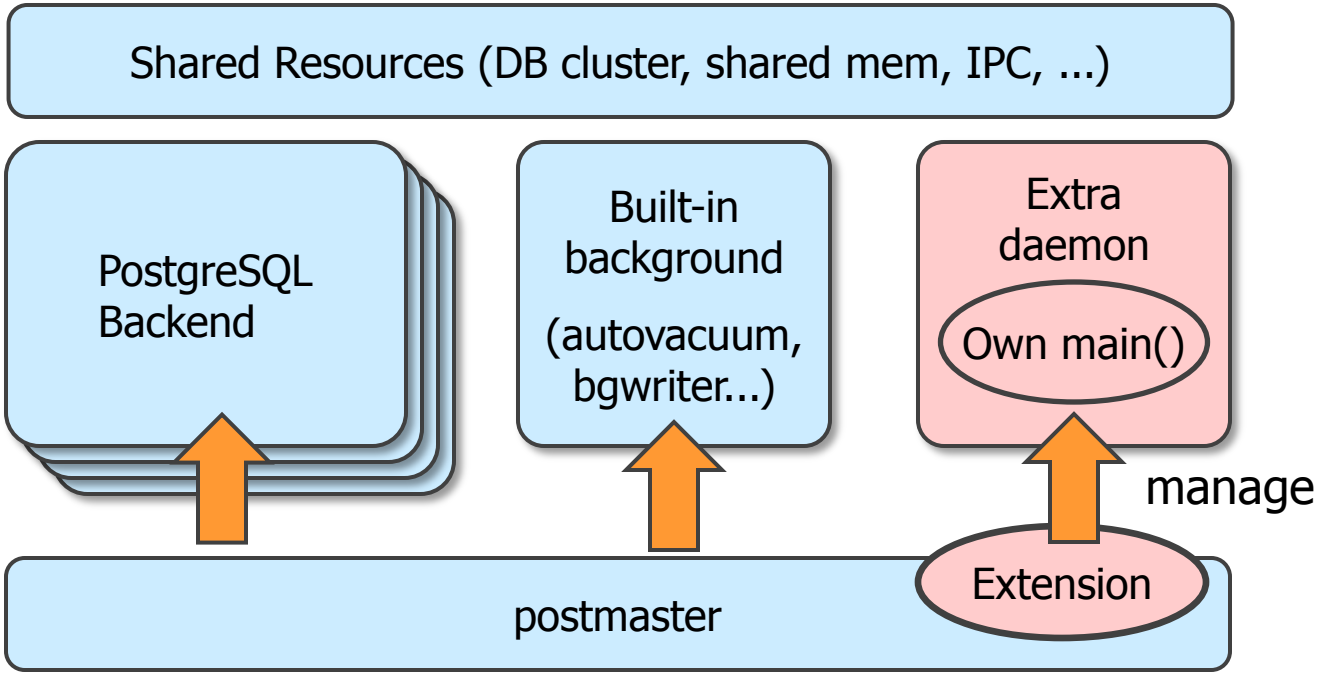
Status Summary. Needs Review: 28, Waiting on Author: 23, Ready for Committer: 10, Committed: 33, Returned with Feedback: 4, Rejected: 2. Total: 100.

Pending Patches

Patch Name	Status	Author	Reviewers	Last Activity
Server Features				
Timeout framework extension and lock_timeout (from 2012-11) Review by ringerc on 2013-01-29: Tested. Needs rebase, regress .out file not updated correctly. Attached document is tests run on patch during eval. Patch by zboszor on 2013-01-30: Rebased patch. Patch by zboszor on 2013-01-31: Update regression test output for the max_prepared_transactions=0 case.	Waiting on Author	Zoltán Böszörményi	Tom Lane, Hari Babu	2013-01-31
[PoC] Writable Foreign Tables Comment by ringerc on 2013-01-18: Tom intends to evaluate in next couple of days; check back 2013-01-24 Patch by kaigai on 2013-01-29: v11, rebased to the latest master. Patch by kaigai on 2013-02-07: v12, add optimization on make_modifytable invocation by Daniel Farina	Ready for Committer	KaiGai Kohei	Alexander Korotkov, Laurenz Albe	2013-02-07
Incorrect behaviour when using a GiST index on points (from 2012-09) Patch by nmisch on 2012-11-10: v3 Review by nmisch on 2012-11-10: Cosmetic points remain, but Ready for Committer	Ready for Committer	Alexander Korotkov	Noah Misch	2013-01-18

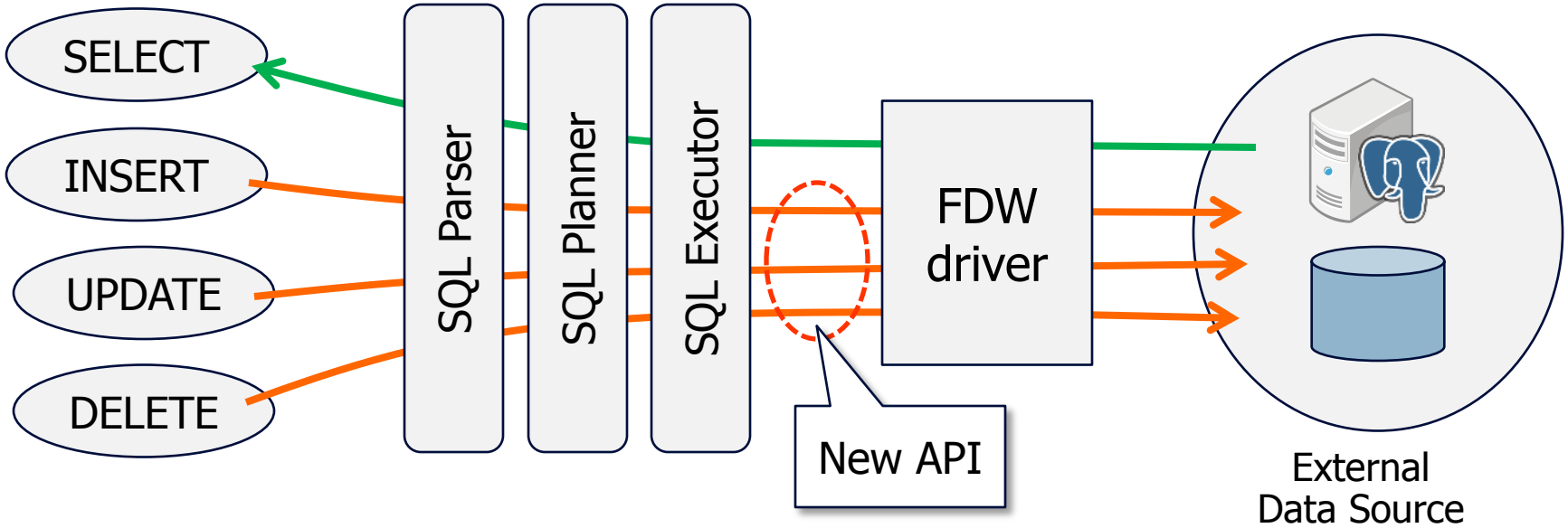
Key features towards upcoming v9.3 (1/3)

- Background Worker
  - It enables extensions to manage own background worker process
  - Pre-requisite of PG-Strom's GPU control server
  - KaiGai implemented 1st version, then Alvaro revised and committed



Key features towards upcoming v9.3 (2/3)

- Writable-FDW
  - It allows FDW-drivers to modify external data source via foreign-table.
  - Several new APIs shall be added
  - Helpful for PG-Strom to modify shadow-tables using standard DML
  - KaiGai submitted patch, then it is "ready-for-committer" status now



Key features towards upcoming v9.3 (3/3)

- Writable-FDW (Pseudo-column support)
  - It required to identify a particular remote-row to be written.
  - "rowid" shall be carried from scan-stage to modify-stage as a value of pseudo-column.
  - Pseudo-column concept is also available to push-down complex calculation into external computing resource.

```
SELECT X, Y, (X-Y)^2 from ftable;
        ↓
SELECT X, Y, Pcol_1 from ftable;
```

Just reference to the calculated result in the remote side

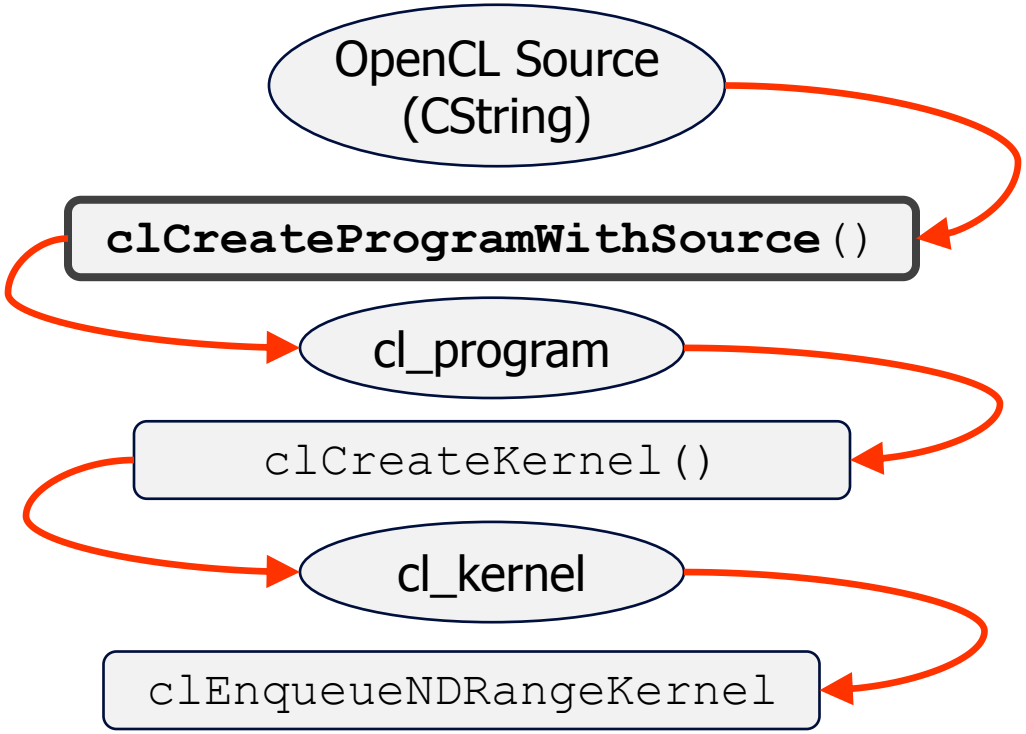
```
(SELECT X, Y, (X-Y)^2 AS Pcol_1
  from remote_data_source)
```

Remote Query

Direction of The Future Development

Move to OpenCL - WIP

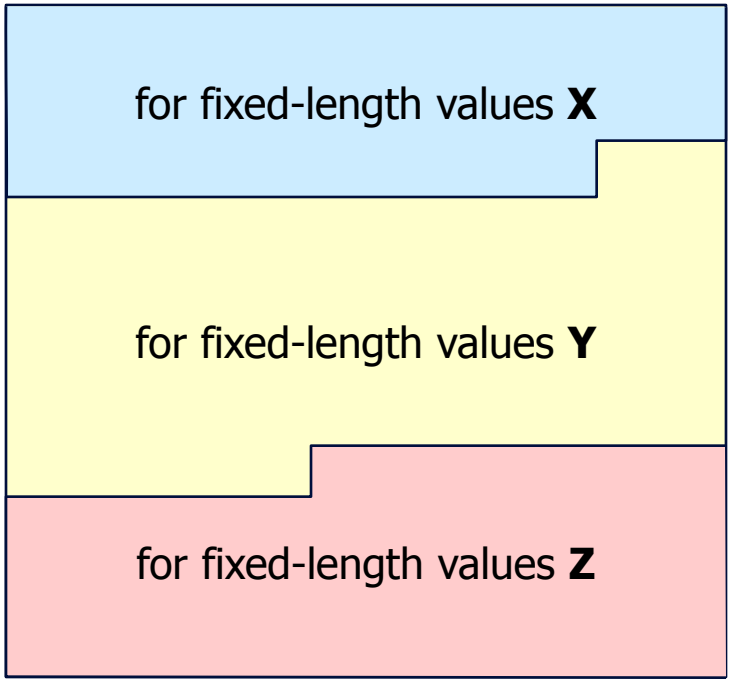
- OpenCL support, instead of CUDA
  - multiplatform support
  - built-in JIT compiler



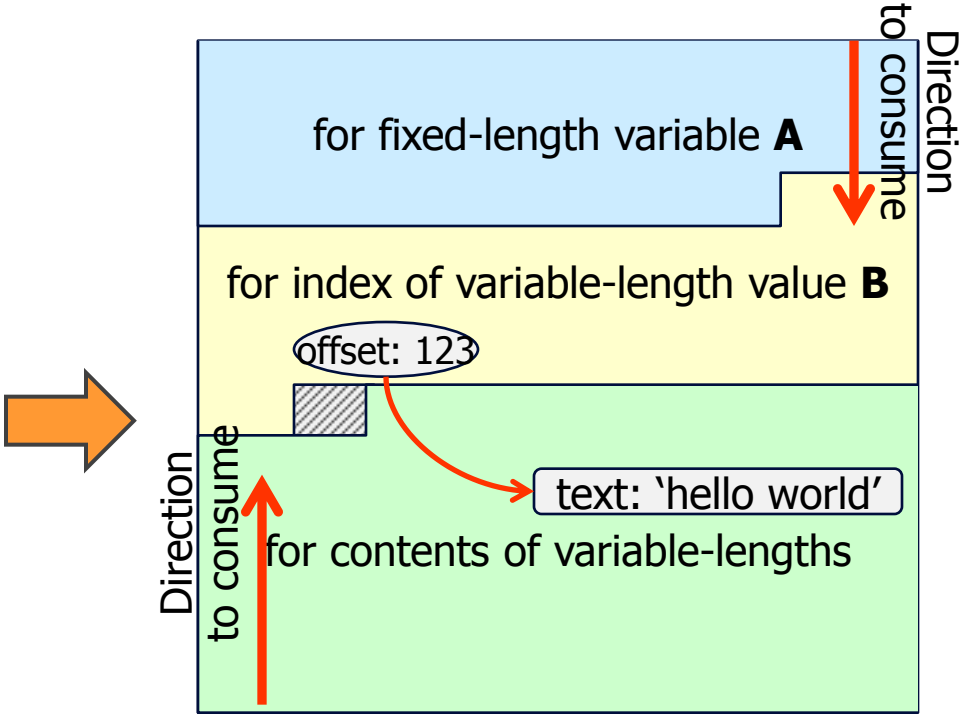
	OpenCL	NVIDIA
NVIDIA GeForce	○	○
Radeon Graphics AMD	○	×
Intel Xeon Phi	○	×

Variable Length Data Support - WIP

- Data layout on chunk-buffer is revising, to accept variable-length data.
- Older format assumed fixed-number of items per chunk.
- Newer format assumes fixed-size chunk; consumed from head/tail



Older chunk-buffer layout



Newer chunk-buffer layout

Procedural Language Support

- This idea allows users to describe complicated logic as procedural language to be executed on GPU.
- Expected usage: image processing, genome matching, ...

```
CREATE FUNCTION genome_similarity(text,text) RETURNS float AS
$$
    varlena *genome1 = ARG1;
    varlena *genome2 = ARG2;
        :
     <something complicated logic>
        :
    return similarity;
$$ LANGUAGE pg_strom;

SELECT id, label FROM genome_db
    WHERE genome_similarity(data, 'ATGCAGGT....') > 0.9;
```

You getting involved

I'd like to know ...

- How PG-Strom run on real-world dataset and workload
- How PG-Strom should get evolved
- Which region and problem will fit

↓

All the co-development / co-evaluation projects are always welcome!

Summary

- Characteristics of GPU & OpenCL
  - Inflexible instructions but much higher parallelism
  - Cheap and small power consumption per computing capability
- PG-Strom - towards most cost-effective database
  - Utilization of GPU to off-load CPU jobs
  - Automatic code generation and JIT compile
  - Asynchronous execution
  - Column-oriented data structure
- Upcoming development
  - Move to OpenCL rather then CUDA
  - Support for variable length values
  - Support for procedural language
- Your involvement will lead future evolution!

Source

- Source code
  - https://github.com/kaigai/pg_strom
- Wikipage
  - http://wiki.postgresql.org/wiki/PGStrom
    (need maintenance...)

Any Questions?