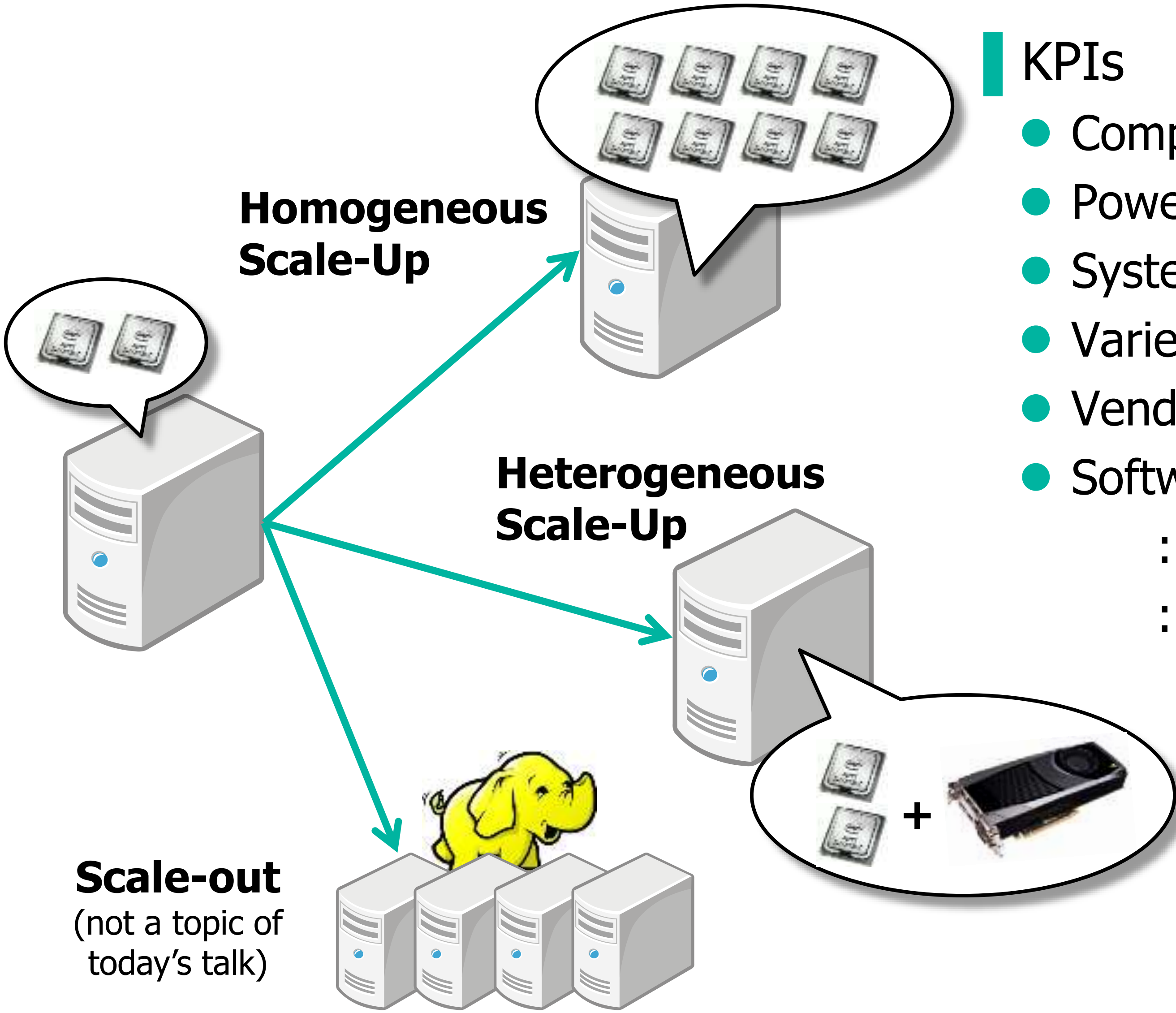
Empowered by Innovation NEC

PG-Strom
GPU Accelerated Asynchronous Query Execution Module

NEC Europe, Ltd
SAP Global Competence Center
KaiGai Kohei <kohei.kaigai@emea.nec.com>

Homogeneous vs Heterogeneous Computing



KPIs

- Computing Performance
- Power Consumption
- System Cost
- Variety of Applications
- Vendor Support
- Software Development

:
:

Characteristics of GPU (1/2)

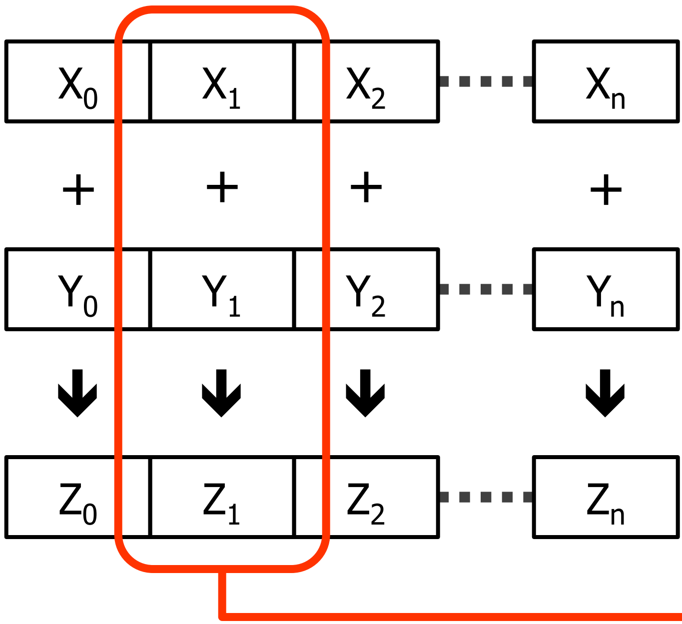
	Nvidia Kepler	AMD GCN	Intel SandyBridge
Model	GTX 680 (*) (Q1/2012)	FirePro S9000 (Q3/2012)	Xeon E5-2690 (Q1/2012)
Number of Transistors	3.54billion	4.3billion	2.26billion
Number of Cores	**1536** Simple	**1792** Simple	**16** Functional
Core clock	1006MHz	925MHz	2.9GHz
Peak FLOPS	3.01Tflops	3.23TFlops	185.6GFlops
Memory Size / TYPE	2GB, GDDR5	6GB, GDDR5	up to 768GB, DDR3
Memory Bandwidth	~192GB/s	~264GB/s	~51.2GB/s
Power Consumption	~195W	~225W	~135W

(*) Nvidia shall release high-end model (Kepler K20) at Q4/2012

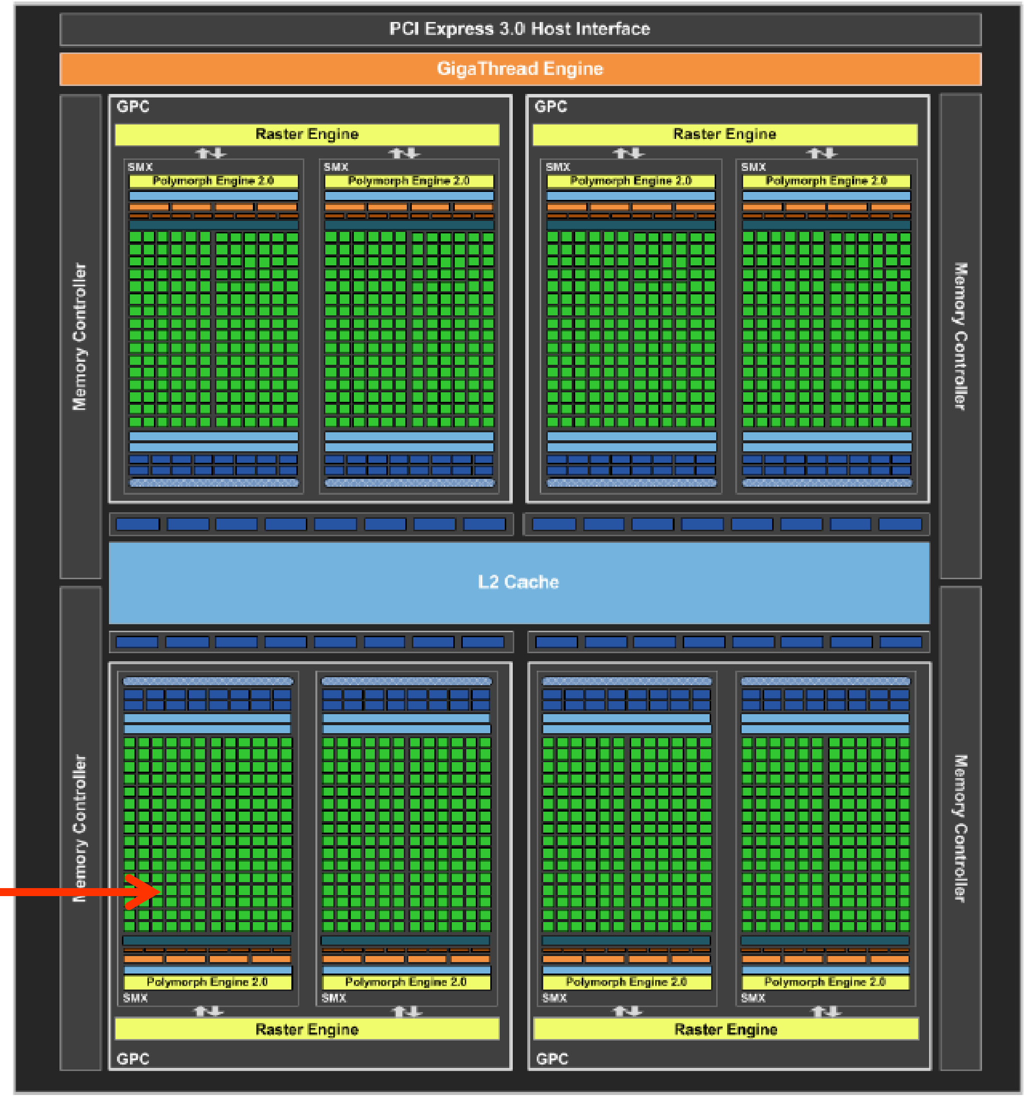
Characteristics of GPU (2/2)

Example)

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



Assign a particular “core”



Nvidia’s GeForce GTX 680 Block Diagram (1536 Cuda cores)

Programming with GPU (1/2)

Example) Parallel Execution of "$\texttt{sqrt}(X_i\text{^}2 + Y_i\text{^}2) < Z_i$"

GPU Code

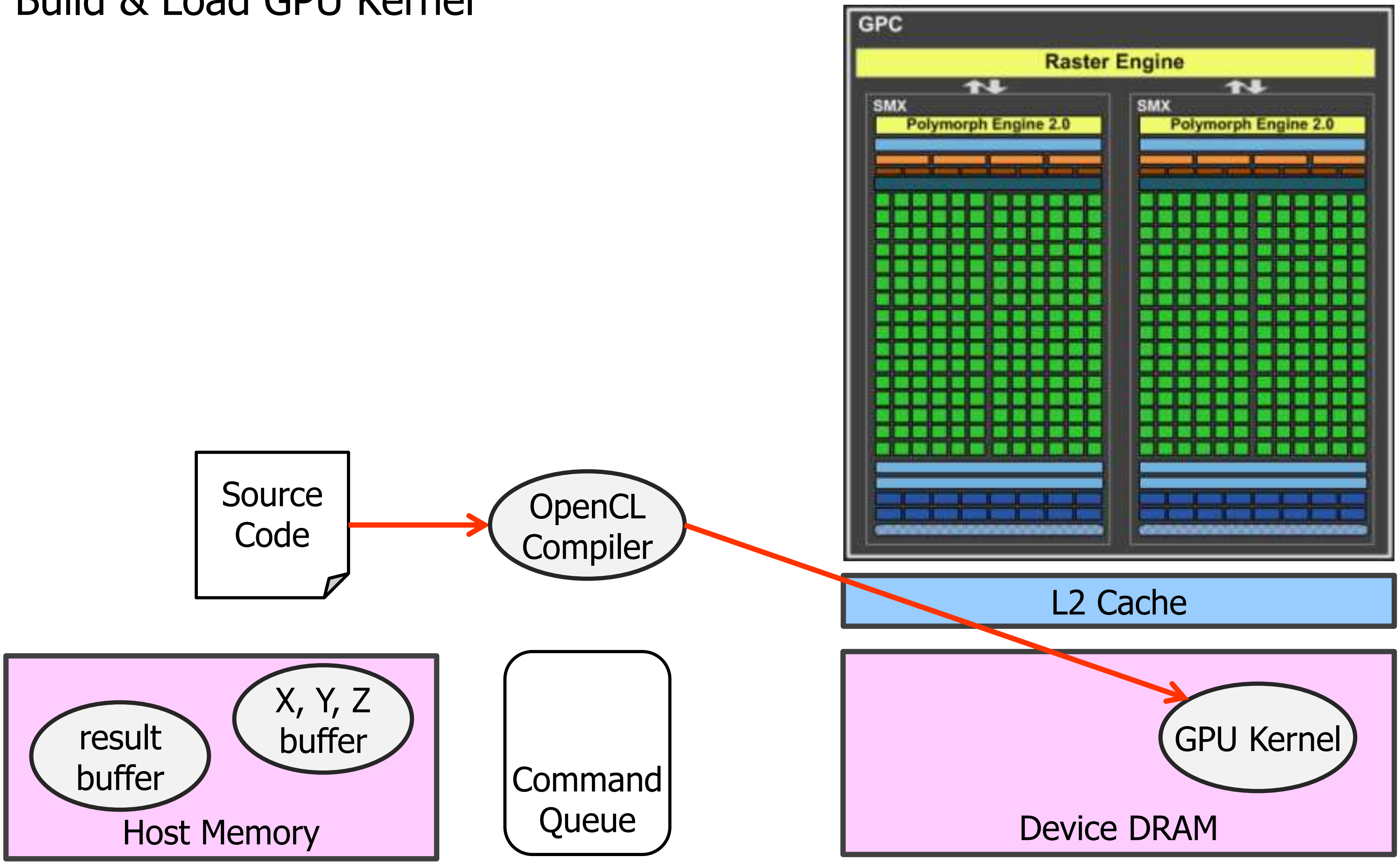
```
__kernel void
sample_func(bool result[], float x[], float y[], float z[]) {
    int i = get_global_id(0);

    result[i] = (bool)(sqrt(x[i]^2 + y[i]^2) < z[i]);
}
```

Host Code

```
#define N   (1<<20)
size_t g_itemsz = N / 1024;
size_t l_itemsz = 1024;

/* Acquire device memory and data transfer (host -> device) */
X = clCreateBuffer(cxt, CL_MEM_READ_WRITE, sizeof(float)*N, NULL, &r);

clEnqueueWriteBuffer(cmdq, X, CL_TRUE, sizeof(float)*N, ...);
/* Set argument of the kernel code */
clSetKernelArg(kernel, 1, sizeof(cl_mem), (void *)&X);

/* Invoke device kernel */
clEnqueueNDRangeKernel(cmdq, kernel, 1, &g_itemsz, &l_itemsz, ...);
```

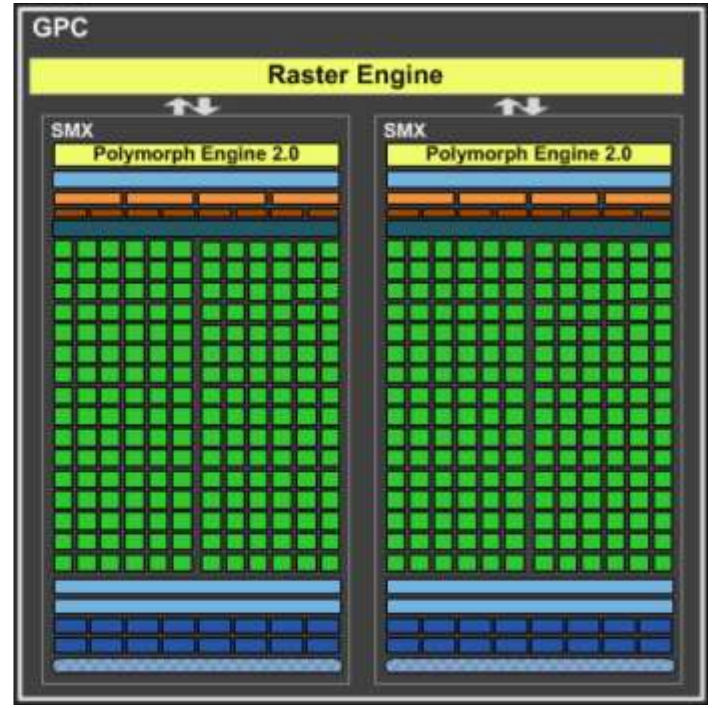
Programming with GPU (2/2)

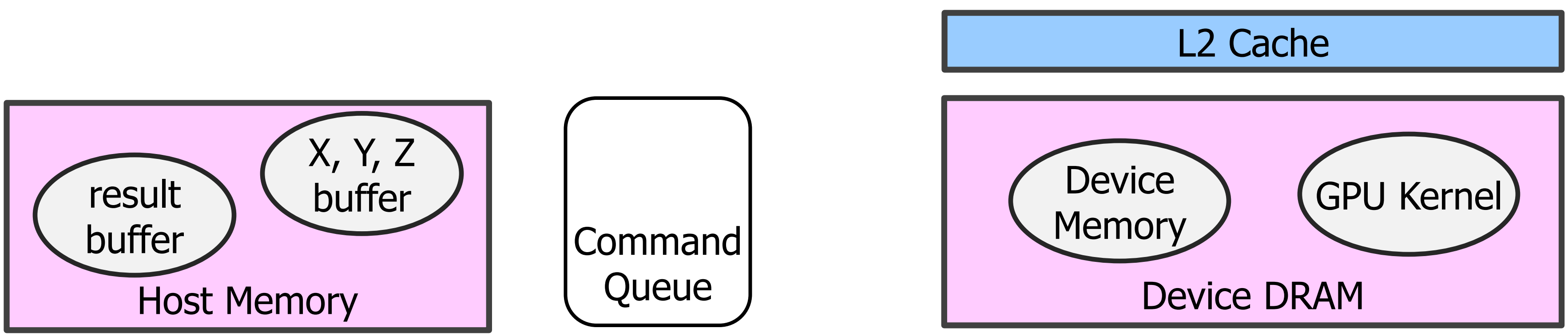
1. Build & Load GPU Kernel



Programming with GPU (2/2)

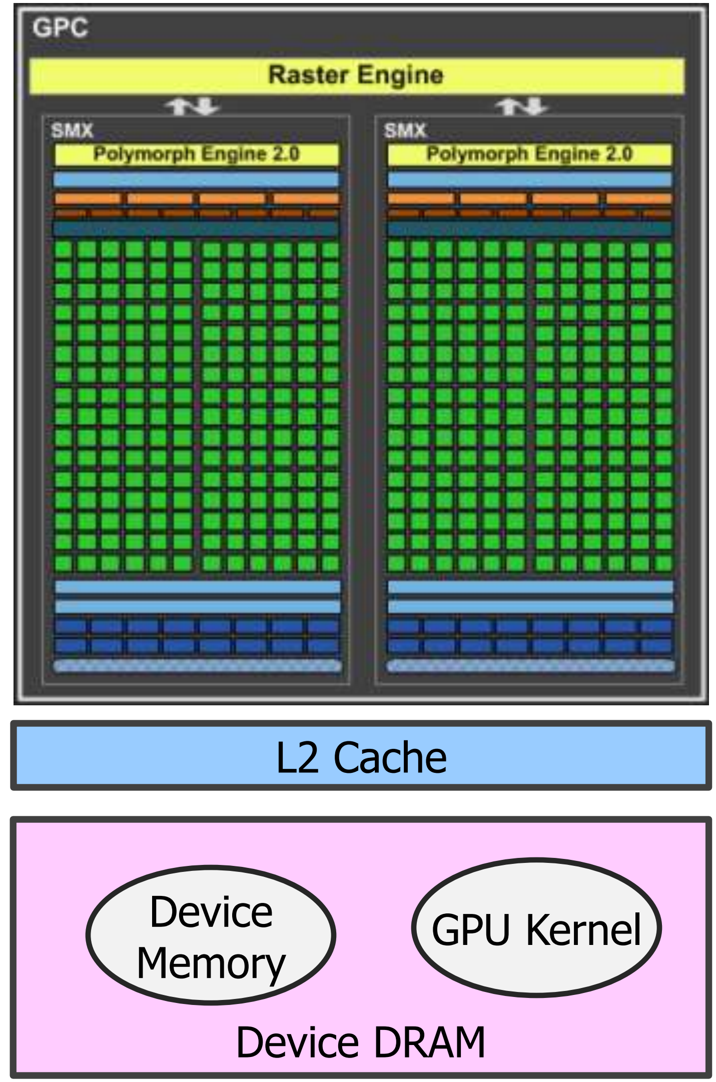
1. Build & Load GPU Kernel
2. Allocate Device Memory

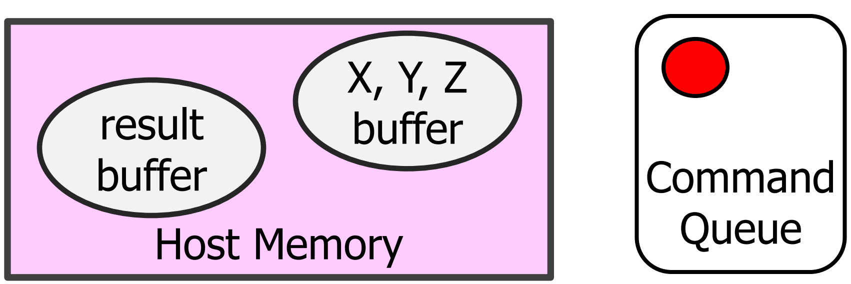




Programming with GPU (2/2)

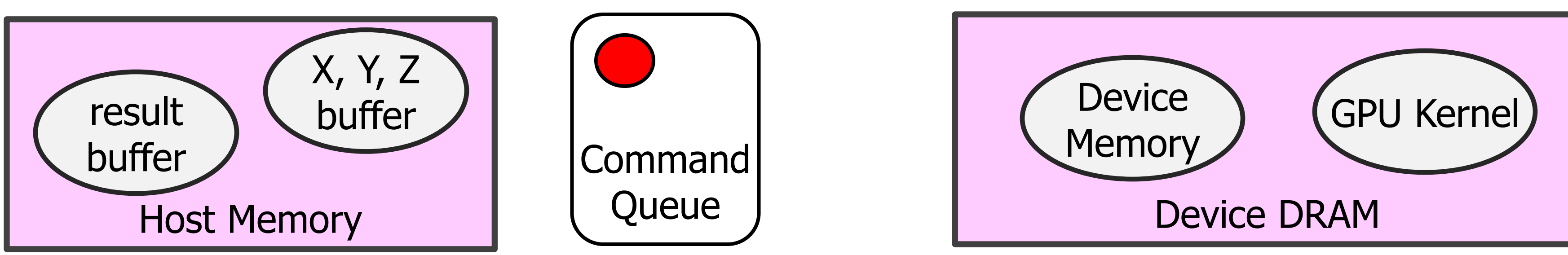
1. Build & Load GPU Kernel
2. Allocate Device Memory
3. Enqueue DMA Transfer (host ➔ device)





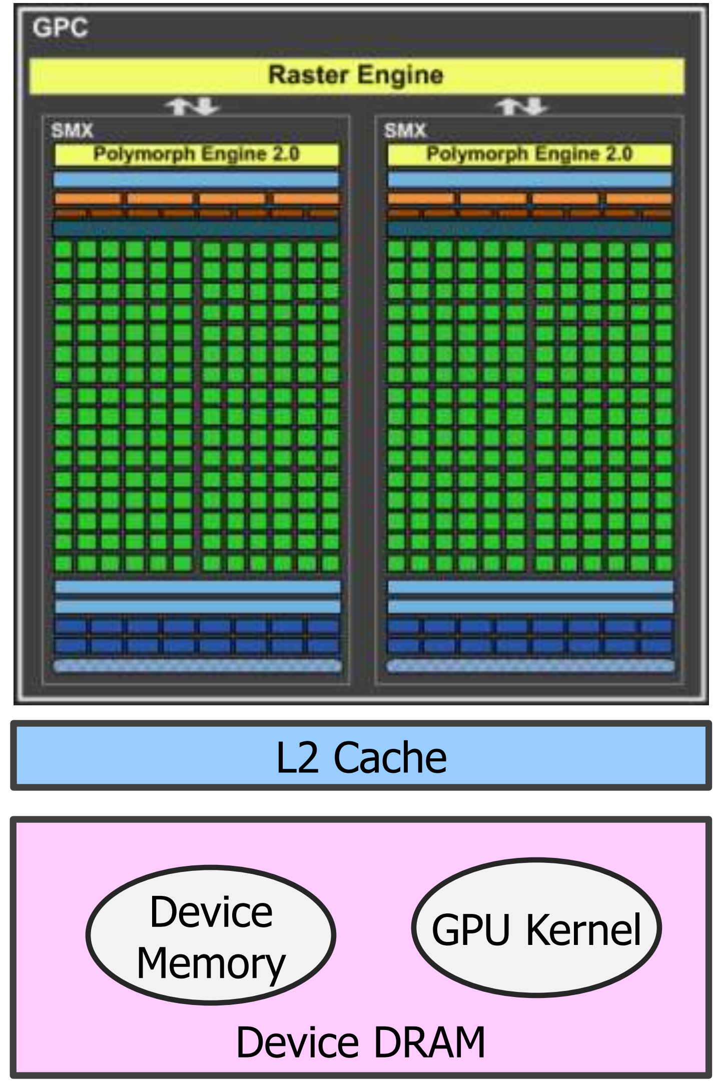
Programming with GPU (2/2)

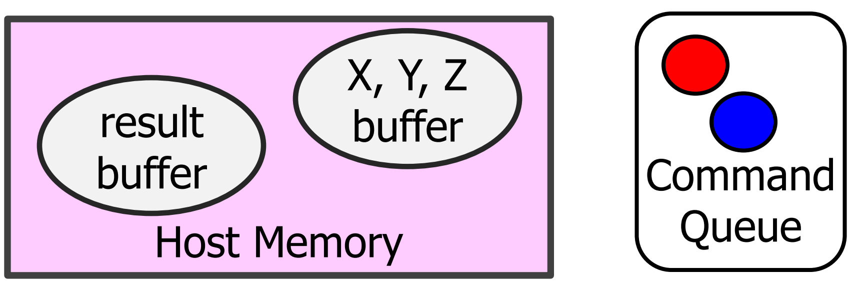
1. Build & Load GPU Kernel
2. Allocate Device Memory
3. Enqueue DMA Transfer (host ➔ device)
4. Setup Kernel Arguments



Programming with GPU (2/2)

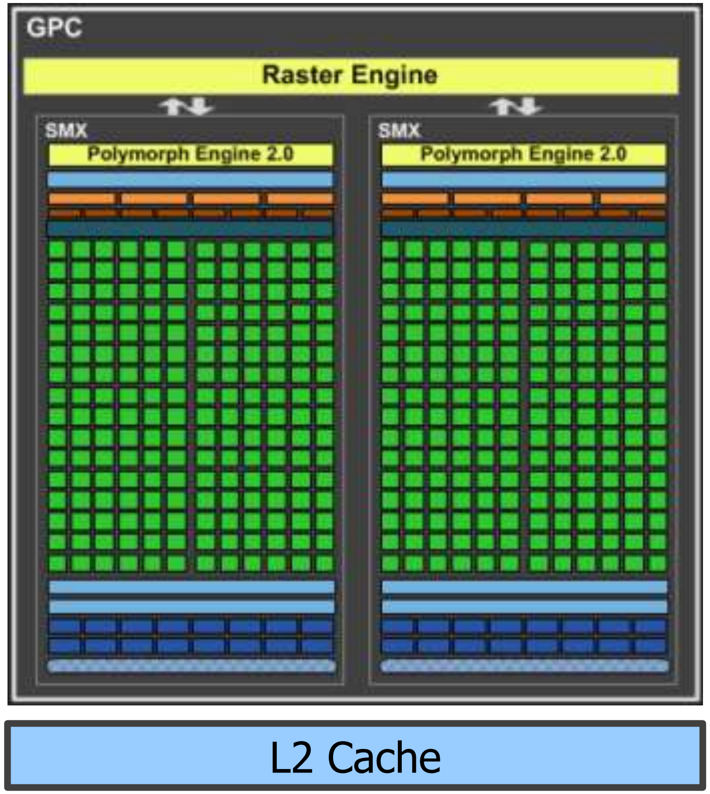
1. Build & Load GPU Kernel
2. Allocate Device Memory
3. Enqueue DMA Transfer (host ➔ device)
4. Setup Kernel Arguments
5. Enqueue Execution of GPU Kernel

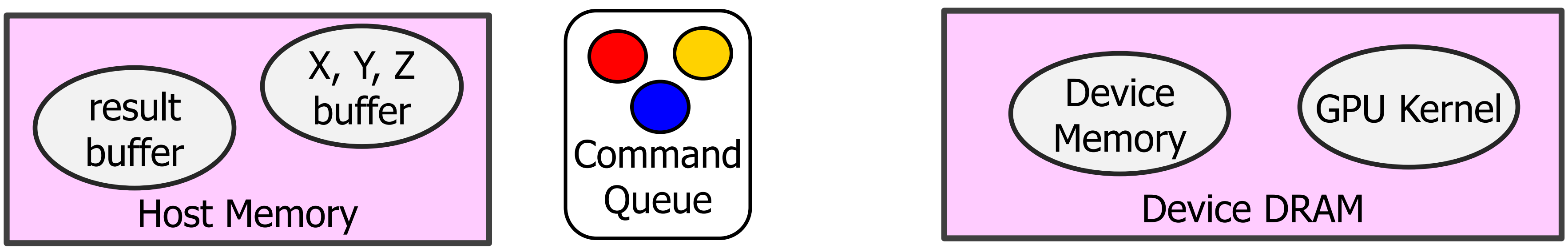




Programming with GPU (2/2)

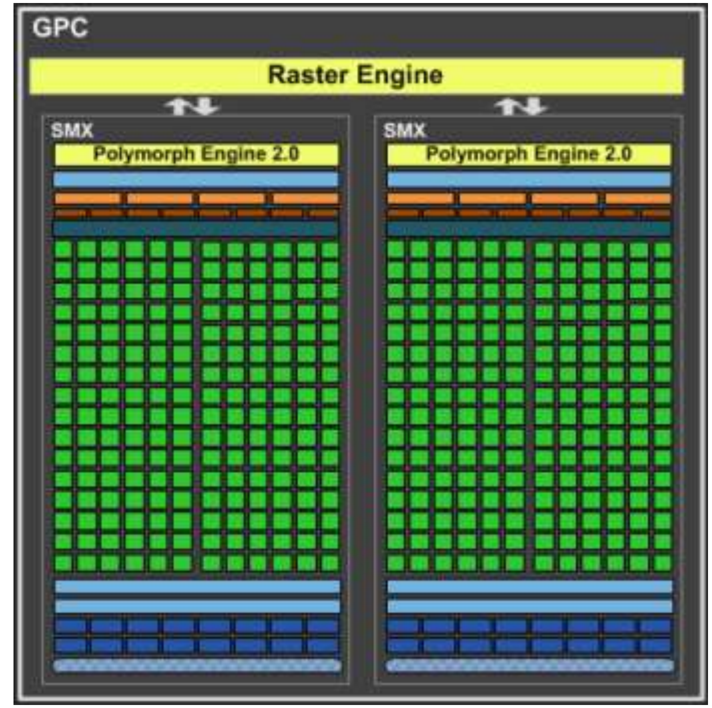
1. Build & Load GPU Kernel
2. Allocate Device Memory
3. Enqueue DMA Transfer (host ➔ device)
4. Setup Kernel Arguments
5. Enqueue Execution of GPU Kernel
6. Enqueue DMA Transfer (device ➔ host)

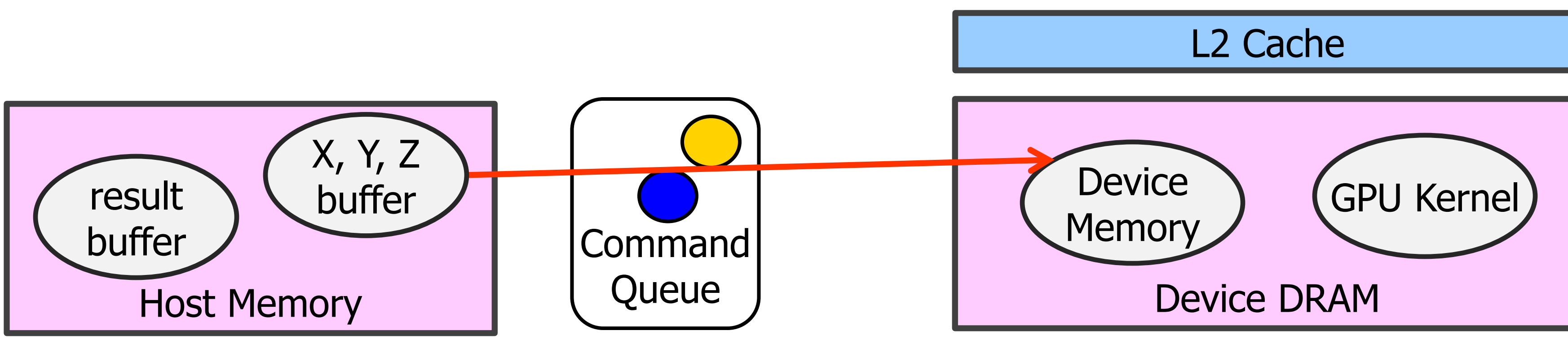




Programming with GPU (2/2)

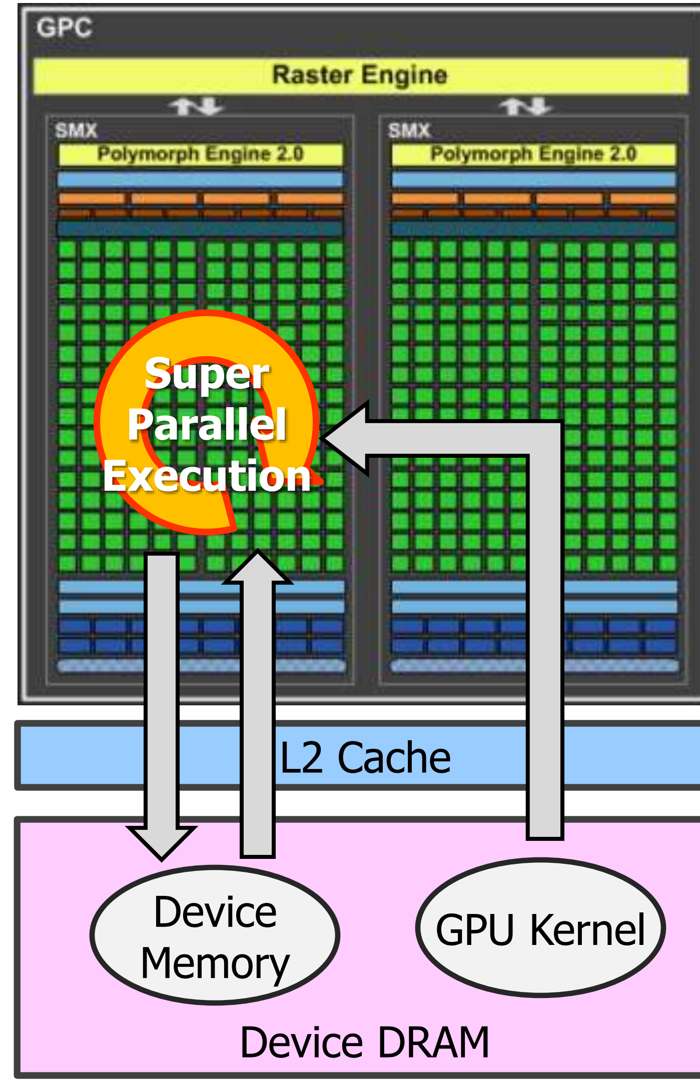
1. Build & Load GPU Kernel
2. Allocate Device Memory
3. Enqueue DMA Transfer (host ➔ device)
4. Setup Kernel Arguments
5. Enqueue Execution of GPU Kernel
6. Enqueue DMA Transfer (device ➔ host)
7. Synchronize the command queue
   - DMA Transfer (host ➔ device)

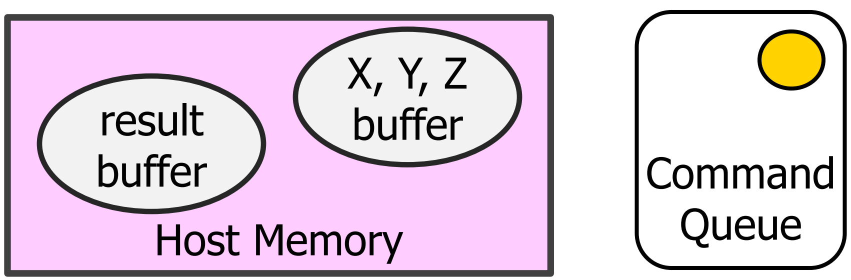




Programming with GPU (2/2)

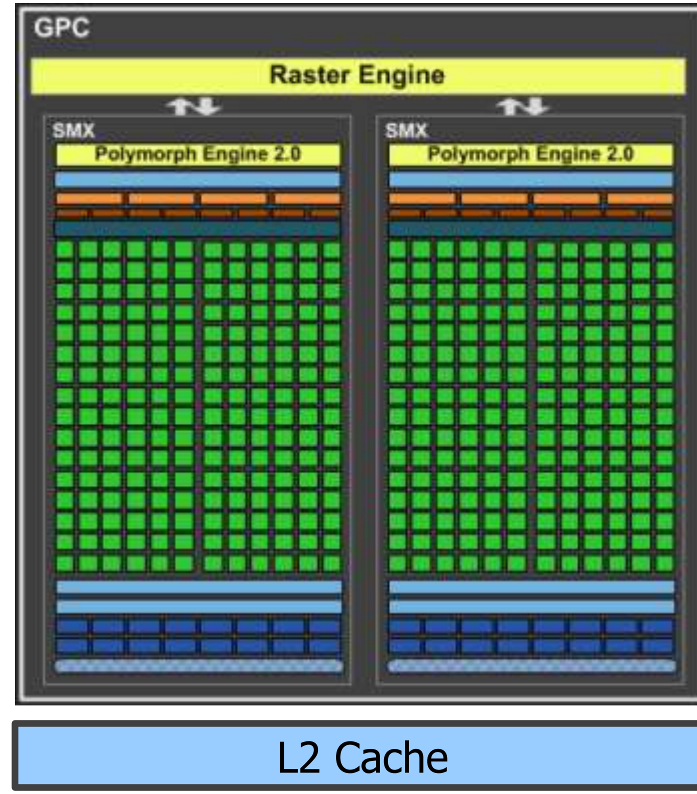
1. Build & Load GPU Kernel
2. Allocate Device Memory
3. Enqueue DMA Transfer (host → device)
4. Setup Kernel Arguments
5. Enqueue Execution of GPU Kernel
6. Enqueue DMA Transfer (device → host)
7. Synchronize the command queue
   - DMA Transfer (host → device)
   - Execution of GPU Kernel

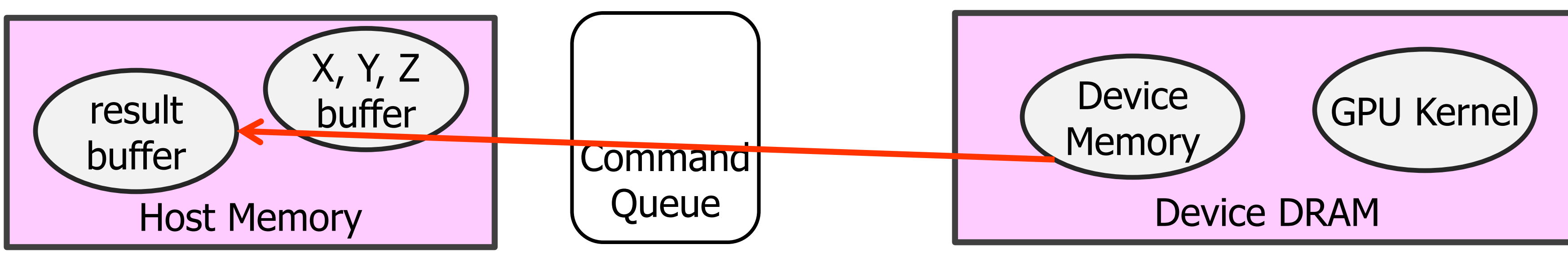




Programming with GPU (2/2)

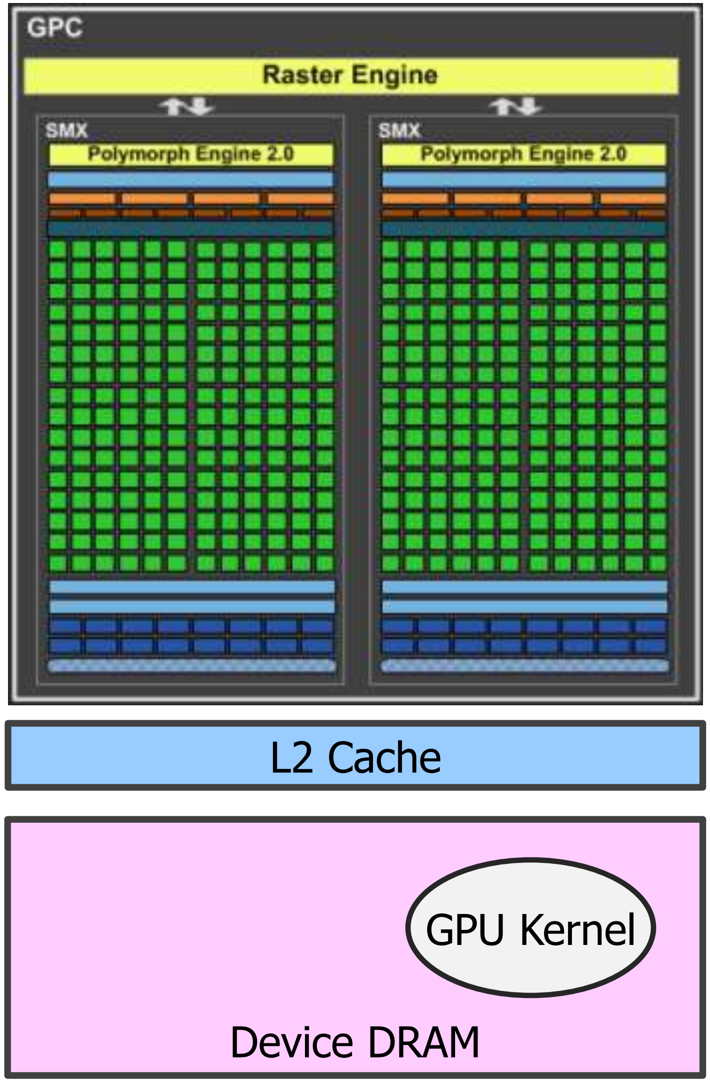
1. Build & Load GPU Kernel
2. Allocate Device Memory
3. Enqueue DMA Transfer (host ➔ device)
4. Setup Kernel Arguments
5. Enqueue Execution of GPU Kernel
6. Enqueue DMA Transfer (device ➔ host)
7. Synchronize the command queue
   - DMA Transfer (host ➔ device)
   - Execution of GPU Kernel
   - DMA Transfer (device ➔ host)
8. Release Device Memory

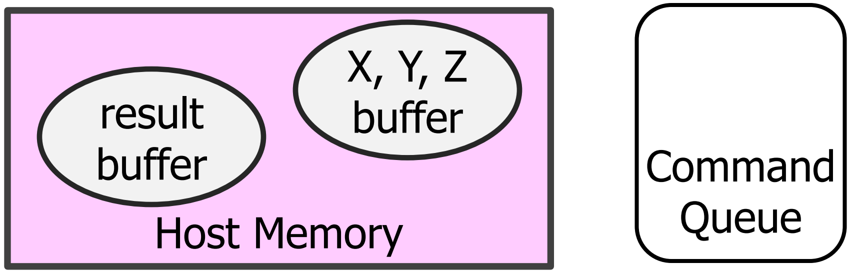




Programming with GPU (2/2)

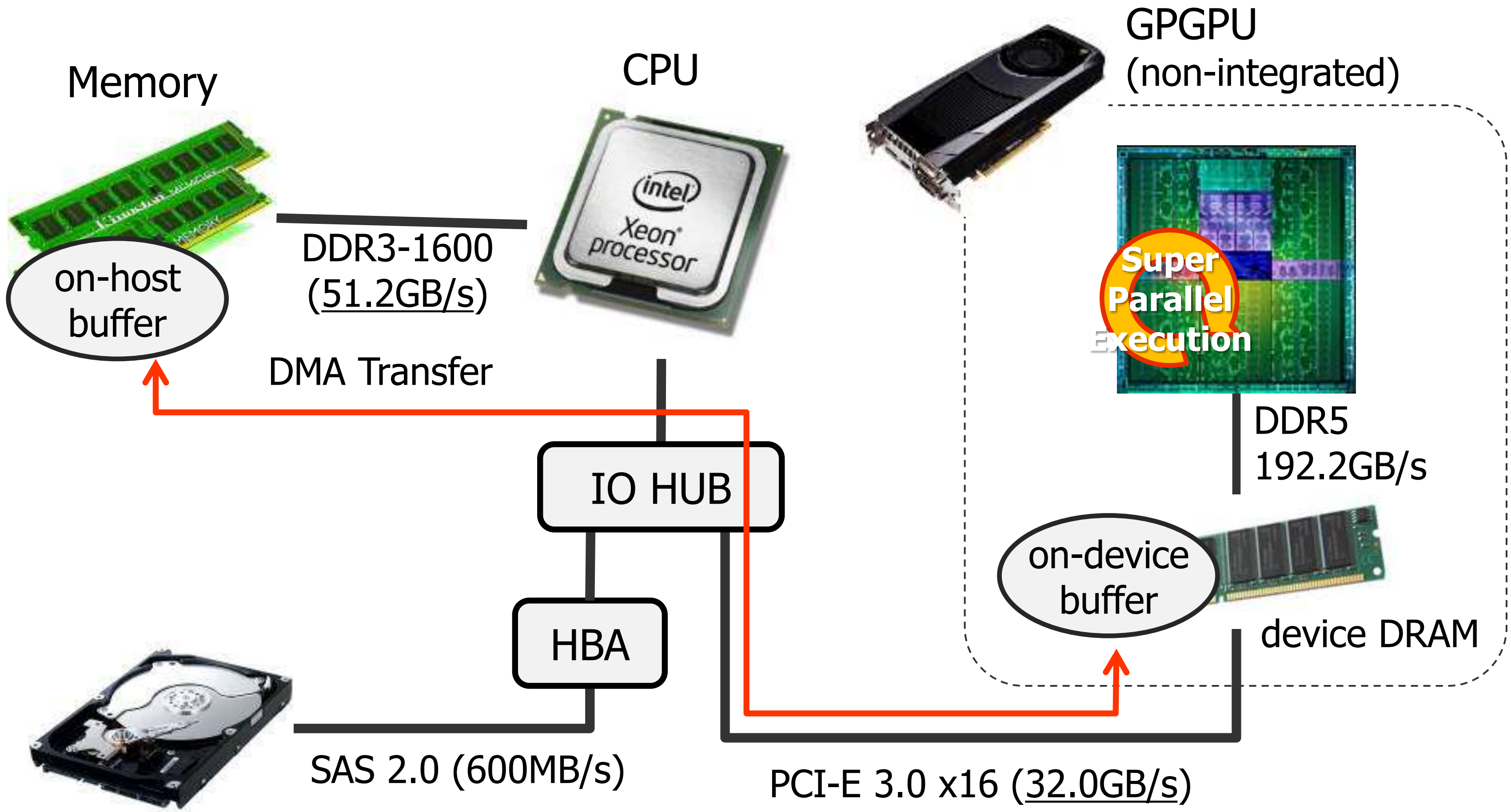
1. Build & Load GPU Kernel
2. Allocate Device Memory
3. Enqueue DMA Transfer (host ➔ device)
4. Setup Kernel Arguments
5. Enqueue Execution of GPU Kernel
6. Enqueue DMA Transfer (device ➔ host)
7. Synchronize the command queue
   - DMA Transfer (host ➔ device)
   - Execution of GPU Kernel
   - DMA Transfer (device ➔ host)
8. Release Device Memory





Basic idea to utilize GPU

- Simultaneous (Asynchronous) execution of CPU and GPU
- Minimization of data transfer between host and device



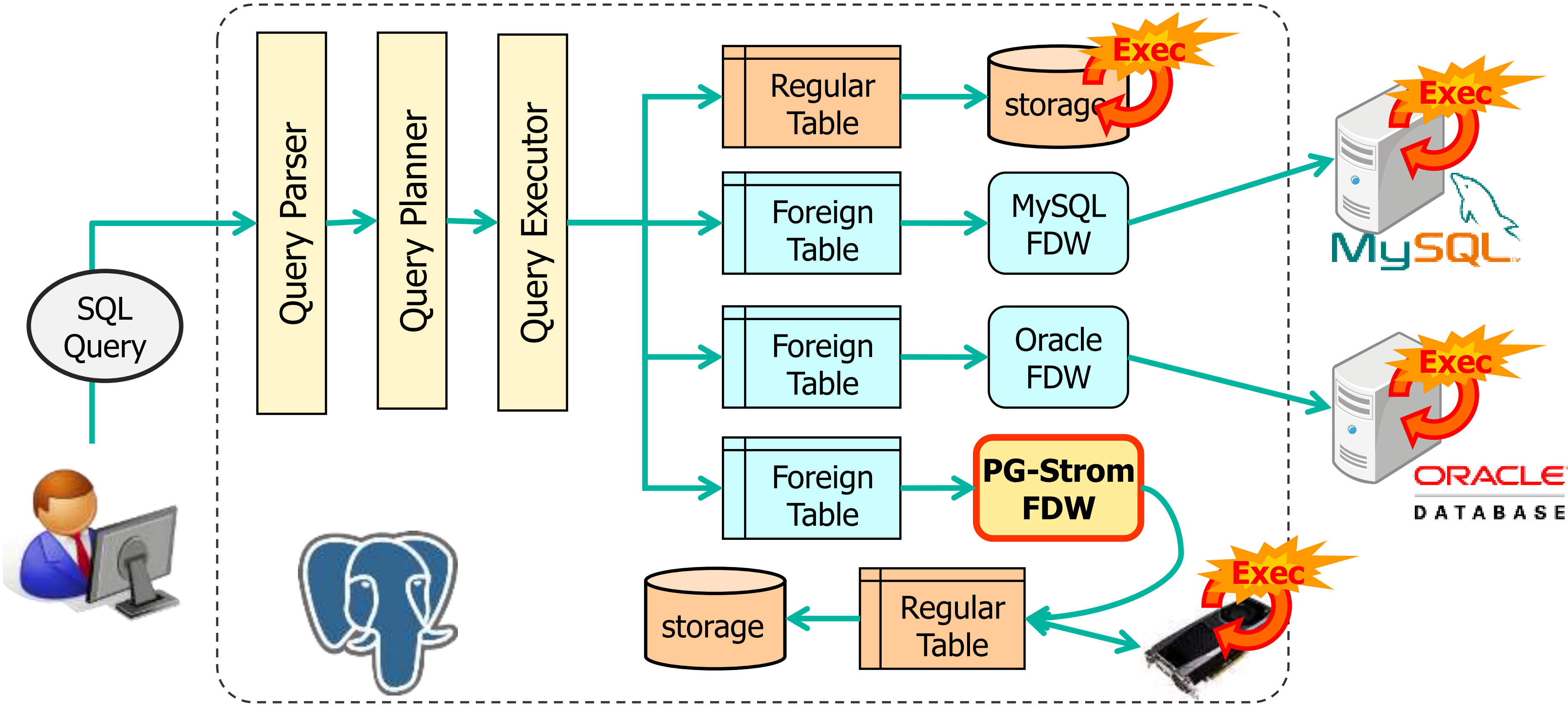
Back to the PostgreSQL world

Don’t I forget I’m talking
at PGconf.EU 2012?

Re-definition of SQL/MED

SQL/MED (Management of External Data)

- External data source performing as if regular tables
- Not only "management", but external computing resources also



Introduction of PG-Strom

PG-Strom is ...

- A FDW extension of PostgreSQL, released under the GPL v3.
  https://github.com/kaigai/pg_strom
- Not a stable module, please **don't** use in production system yet.
- Designed to utilize GPU devices for CPU off-load according to their characteristics.

Key features of PG-Strom

- Just-in-time pseudo code generation for GPU execution
- Column-oriented internal data structure
- Asynchronous query execution

→ Reduction of response-time dramatically!

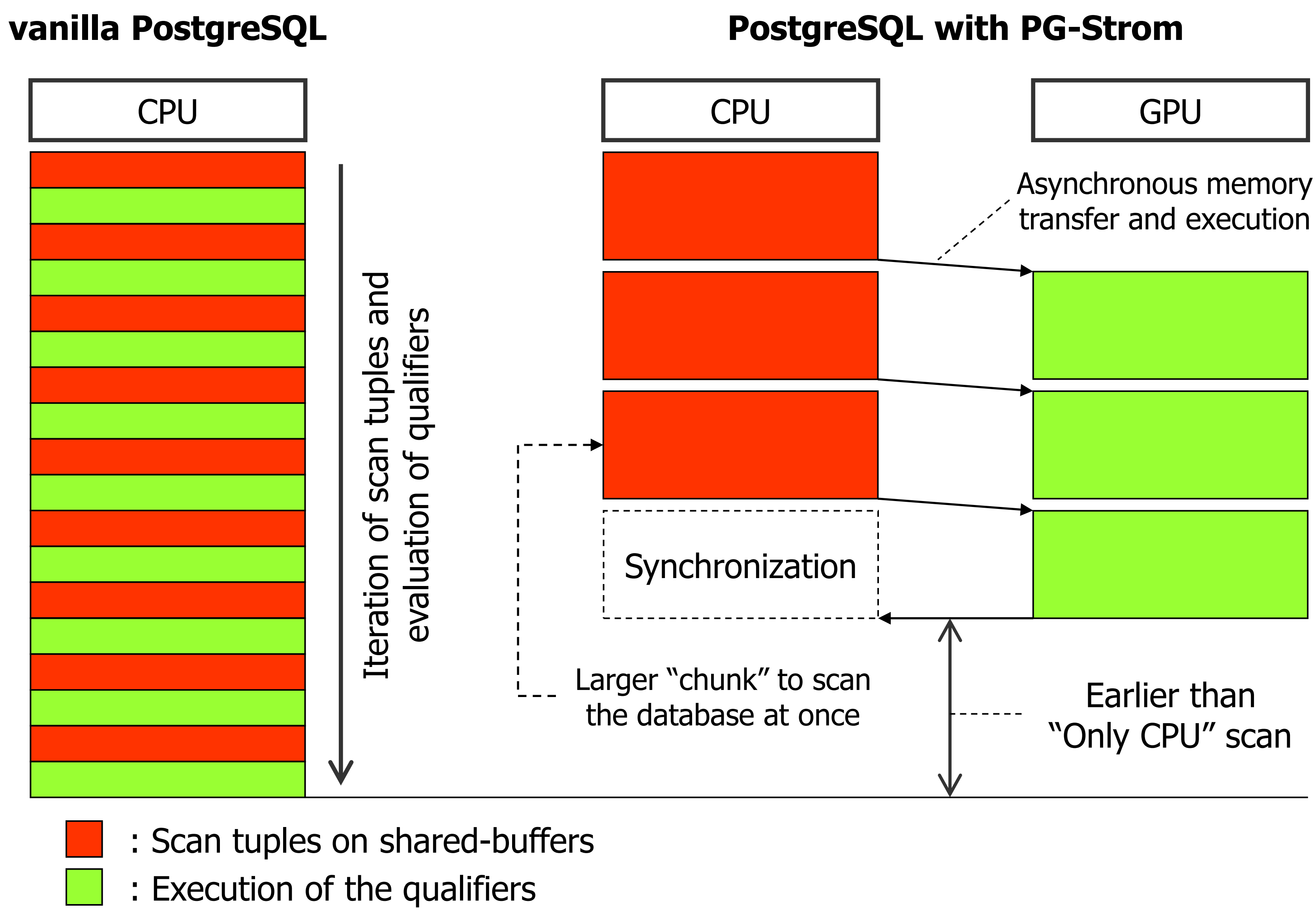
Asynchronous Execution using CPU/GPU (1/2)

CPU characteristics
- Complex Instruction, less parallelism
- Expensive & much power consumption per core
- I/O capability

GPU characteristics
- Simple Instruction, much parallelism
- Cheap & less power consumption per core
- Device memory access only (except for integrated GPU)

"Best Mix" strategy of PG-Stom
- CPU focus on I/O and control stuff.
- GPU focus on calculation stuff.

Asynchronous Execution using CPU/GPU (2/2)



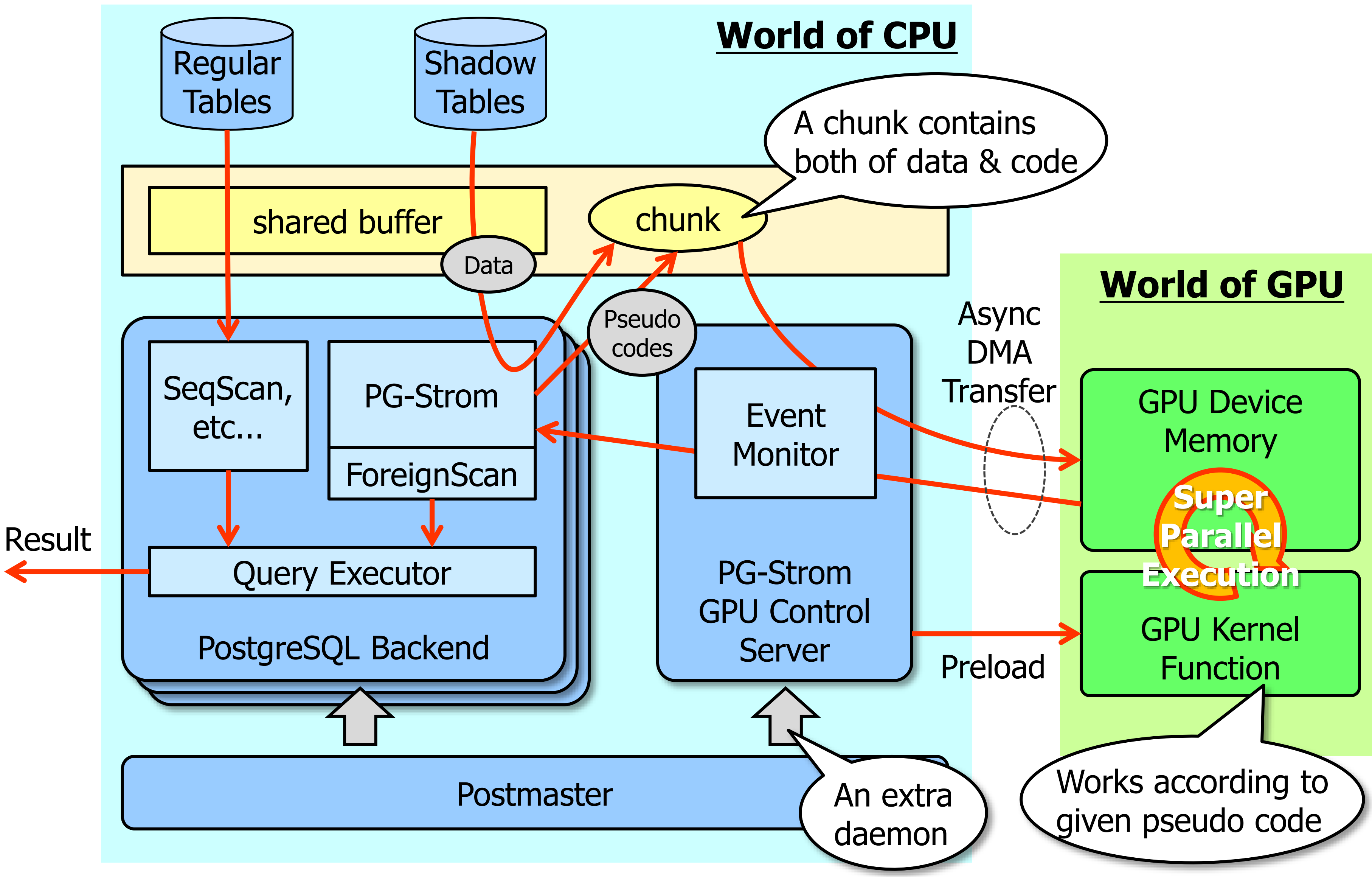
So what, How fast is it?

```
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

Architecture of PG-Strom



Pseudo code generation (1/2)

```
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
         :
```

Super Parallel Execution

GPU Kernel Function

Pseudo code generation (2/2)

Regularly, we should avoid branch operations on GPU code

```
result = 0;
if (condition)
{
  result = a + b;
}
else
{
  result = a - b;
}
return 2 * result;
```

```
__global__
void kernel_qual(const int commands[],...)
{
  const int  *cmd = commands;
    :
  while (*cmd != GPUCMD_TERMINAL_COMMAND)
  {
    switch (*cmd)
    {
      case GPUCMD_CONREF_INT4:
        regs[*(cmd+1)] = *(cmd + 2);
        cmd += 3;
        break;
      case GPUCMD_VARREF_INT4:
        VARREF_TEMPLATE(cmd, uint);
        break;
      case GPUCMD_OPER_INT4_PL:
        OPER_ADD_TEMPLATE(cmd, int);
        break;
          :
          :
```

Pseudo code generation (2/2)

Regularly, we should avoid branch operations on GPU code

```
result = 0;
if (condition)
{
  result = a + b;
}
else
{
  result = a - b;
}
return 2 * result;
```

```
__global__
void kernel_qual(const int commands[],...)
{
  const int  *cmd = commands;
     :
  while (*cmd != GPUCMD_TERMINAL_COMMAND)
  {
    switch (*cmd)
    {
      case GPUCMD_CONREF_INT4:
        regs[*(cmd+1)] = *(cmd + 2);
        cmd += 3;
        break;
      case GPUCMD_VARREF_INT4:
        VARREF_TEMPLATE(cmd, uint);
        break;
      case GPUCMD_OPER_INT4_PL:
        OPER_ADD_TEMPLATE(cmd, int);
        break;
          :
          :
```

Pseudo code generation (2/2)

Regularly, we should avoid branch operations on GPU code

```
result = 0;
if (condition)
{
  result = a + b;
}
else
{
  result = a - b;
}
return 2 * result;
```

```
__global__
void kernel_qual(const int commands[],...)
{
  const int  *cmd = commands;
      :
  while (*cmd != GPUCMD_TERMINAL_COMMAND)
  {
    switch (*cmd)
    {
      case GPUCMD_CONREF_INT4:
        regs[*(cmd+1)] = *(cmd + 2);
        cmd += 3;
        break;
      case GPUCMD_VARREF_INT4:
        VARREF_TEMPLATE(cmd, uint);
        break;
      case GPUCMD_OPER_INT4_PL:
        OPER_ADD_TEMPLATE(cmd, int);
        break;
            :
            :
```

Pseudo code generation (2/2)

Regularly, we should avoid branch operations on GPU code

```
result = 0;
if (condition)
{
   result = a + b;
}
else
{
   result = a - b;
}
return 2 * result;
```

```
__global__
void kernel_qual(const int commands[],...)
{
  const int  *cmd = commands;
     :
  while (*cmd != GPUCMD_TERMINAL_COMMAND)
  {
    switch (*cmd)
    {
      case GPUCMD_CONREF_INT4:
        regs[*(cmd+1)] = *(cmd + 2);
        cmd += 3;
        break;
      case GPUCMD_VARREF_INT4:
        VARREF_TEMPLATE(cmd, uint);
        break;
      case GPUCMD_OPER_INT4_PL:
        OPER_ADD_TEMPLATE(cmd, int);
        break;
          :
          :
```

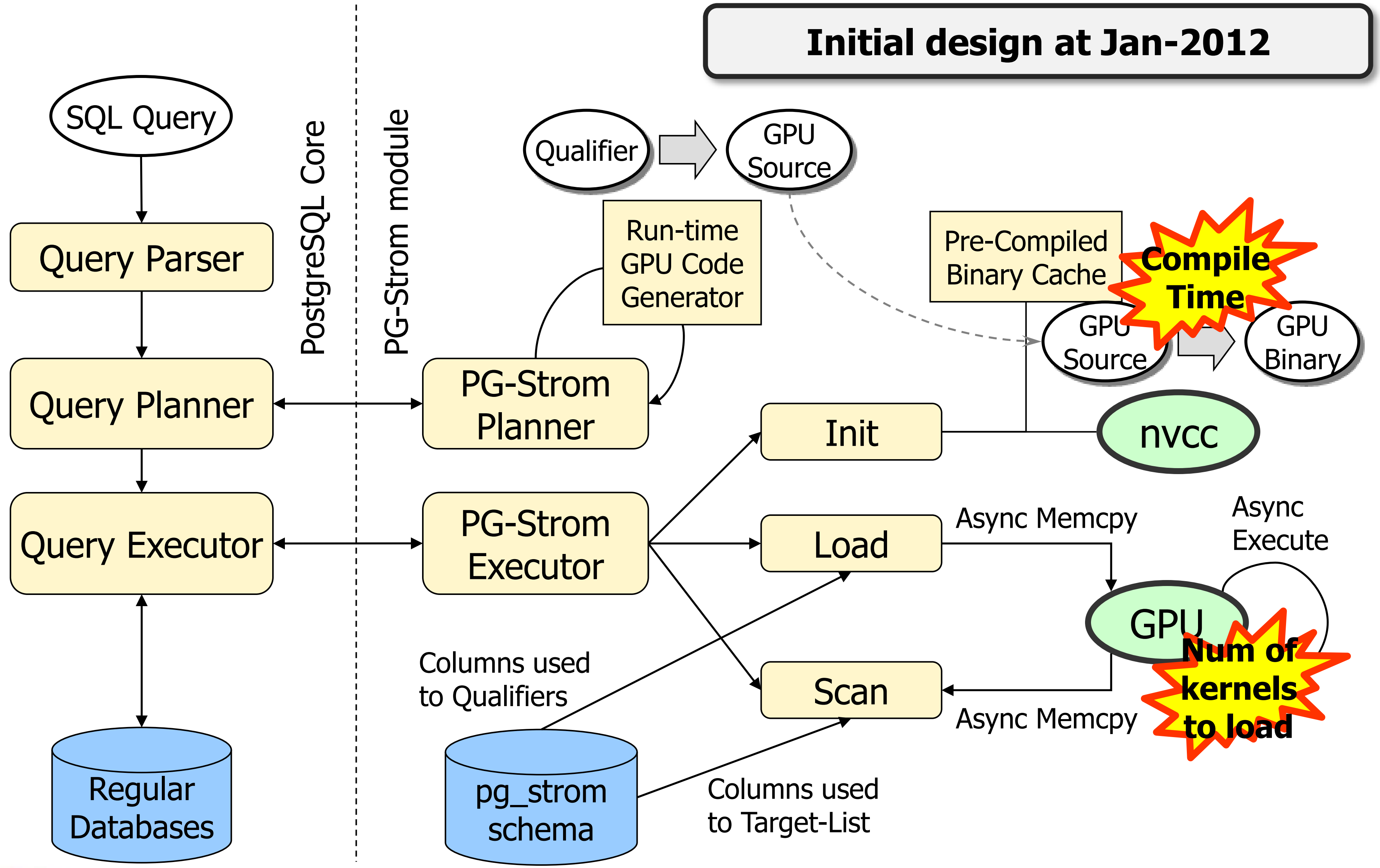
Pseudo code generation (2/2)

Regularly, we should avoid branch operations on GPU code

```
result = 0;
if (condition)
{
   result = a + b;
}
else
{
   result = a - b;
}
return 2 * result;
```

```
__global__
void kernel_qual(const int commands[],...)
{
  const int  *cmd = commands;
    :
  while (*cmd != GPUCMD_TERMINAL_COMMAND)
  {
    switch (*cmd)
    {
      case GPUCMD_CONREF_INT4:
        regs[*(cmd+1)] = *(cmd + 2);
        cmd += 3;
        break;
      case GPUCMD_VARREF_INT4:
        VARREF_TEMPLATE(cmd, uint);
        break;
      case GPUCMD_OPER_INT4_PL:
        OPER_ADD_TEMPLATE(cmd, int);
        break;
          :
          :
```

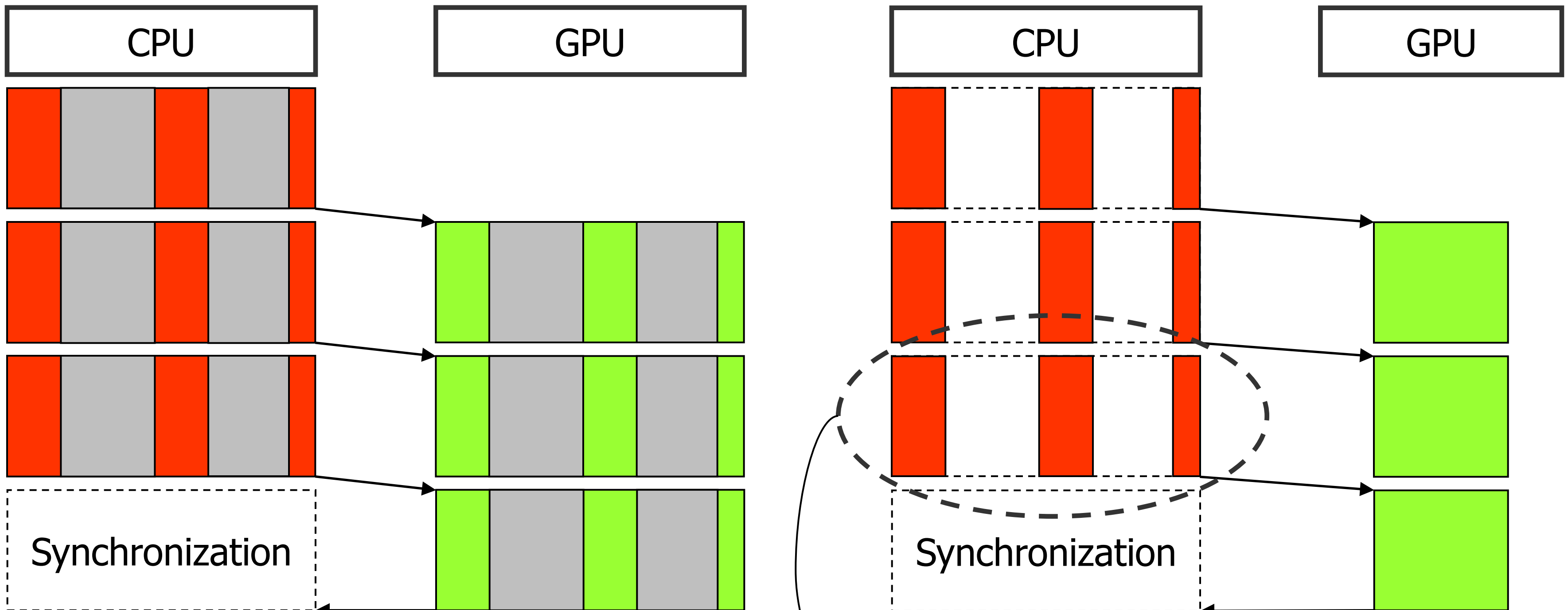
OT: Why "pseudo", not native code



Save the bandwidth of PCI-Express bus

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

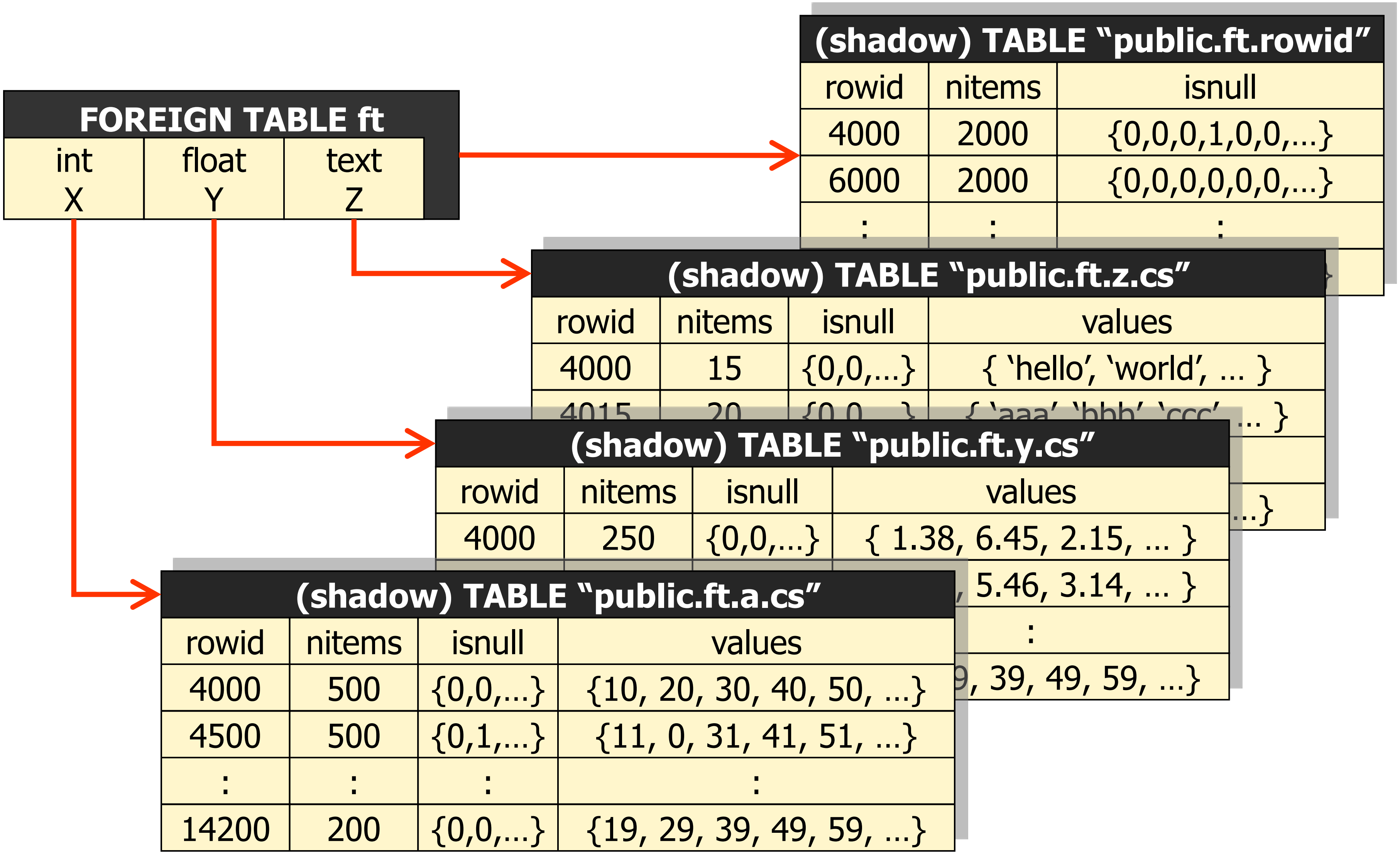
➔ No sense to fetch columns being not in use



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

Reduction of data-size to be transferred via PCI-E

Data density & Column-oriented structure (1/3)



Data density & Column-oriented 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)
```

Data density & Column-oriented structure (2/3)

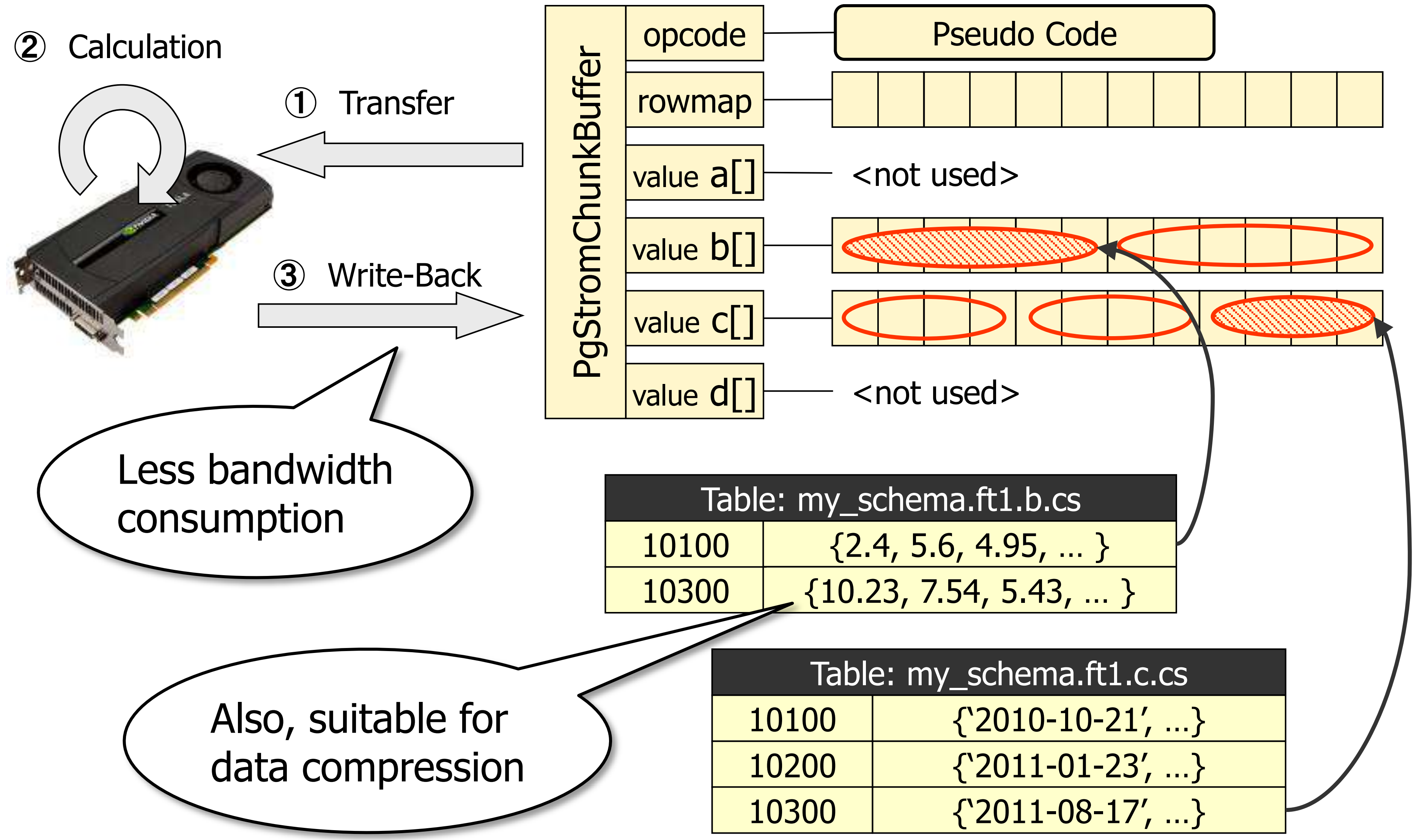


Table: my_schema.ft1.b.cs	
10100	{2.4, 5.6, 4.95, … }
10300	{10.23, 7.54, 5.43, … }

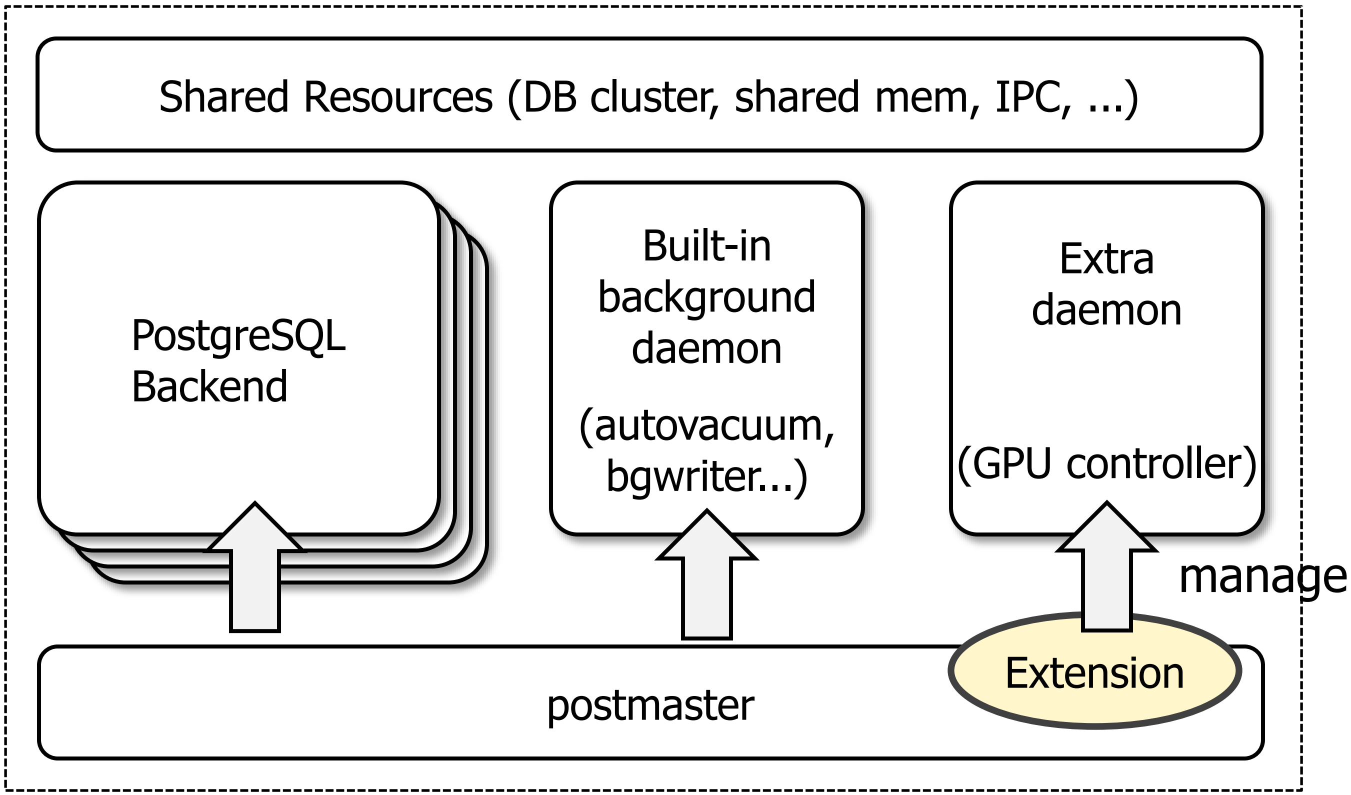
Table: my_schema.ft1.c.cs	
10100	{'2010-10-21', …}
10200	{'2011-01-23', …}
10300	{'2011-08-17', …}

Demonstration

Key features towards upcoming v9.3 (1/2)

Extra Daemon

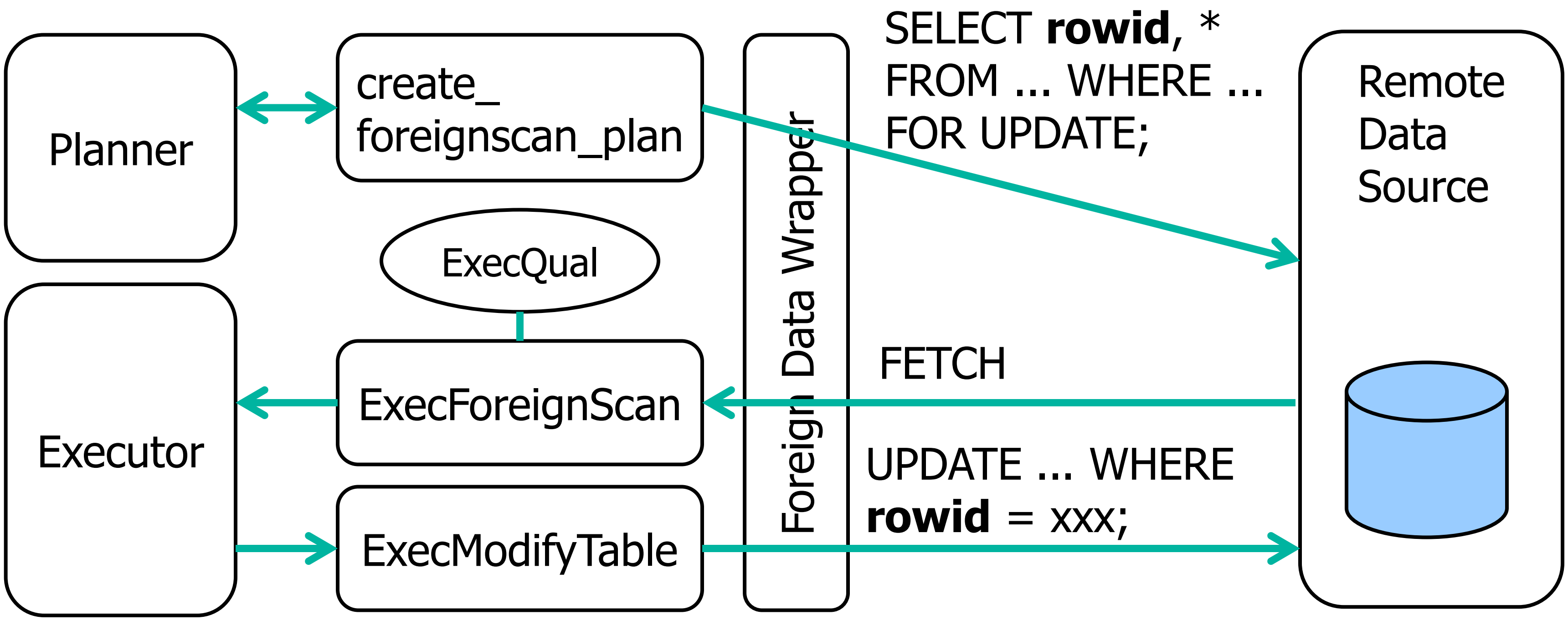
- It enables extension to manage background worker processes.
- Pre-requisites to implement PG-Strom's GPU control server
- Alvaro submitted this patch on CommitFest:Nov.



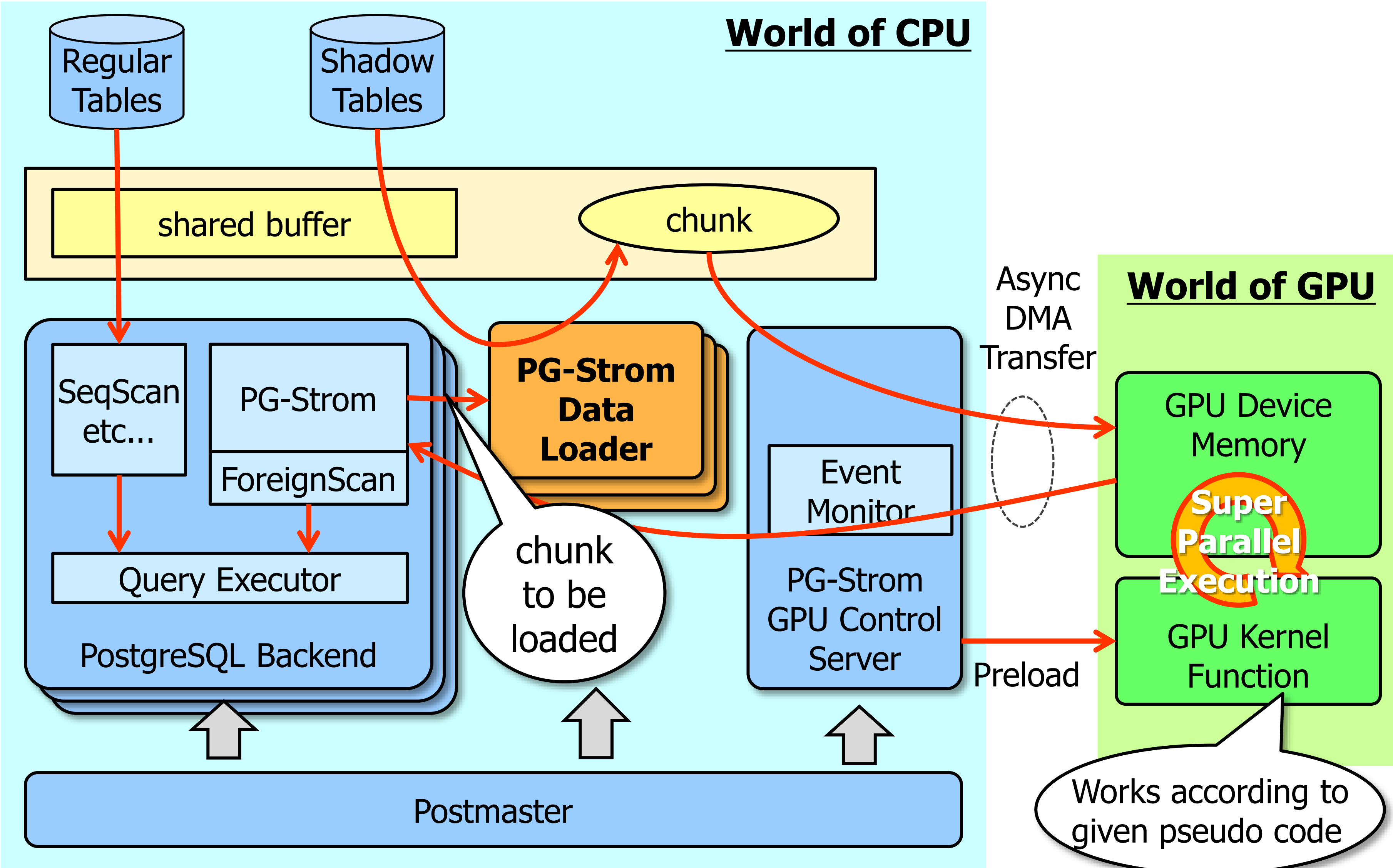
Key features towards upcoming v9.3 (2/2)

Writable Foreign Table

- It enables to use usual INSERT, UPDATE or DELETE to modify foreign tables managed by PG-Strom.
- KaiGai submitted a proof-of-concept patch to CommitFest:Sep.
- In-core postgresql_fdw is needed for working example.



More Rapidness (1/2) – Parallel Data Load



More Rapidness (2/2) – TargetList Push-down

SELECT ((a + b) * (c – d))^2 FROM ftbl;

↓

SELECT pseudo_col FROM ftbl;

a	b	c	d	pseudo_col
1	2	3	4	9
3	1	4	1	144
2	4	1	4	324
2	2	3	6	144
:	:	:	:	:

Computed during ForeignScan

- Pseudo column hold "computed" result, to be just referenced
- Performs as if extra columns exist in addition to table definition

We need you getting involved

- Project was launched from my personal curiousness,
- So, it is uncertain how does PG-Strom fit “real-life” workload.
- We definitely have to **find out** attractive usage of PG-Strom

Summary

- Characteristics of GPU device
  - Inflexible instructions, but much higher parallelism
  - Cheap & small power consumption per computing capability
- PG-Strom
  - Utilization of GPU device for CPU off-load and rapid response
  - Just-in-time pseudo code generation according to the given query
  - Column-oriented data structure for data density on PCI-Express bus
  - → In the result, dramatic shorter response time
- Upcoming development
  - Upstream
    - Extra daemons, Writable Foreign Tables
  - Extension
    - Move to OpenCL rather than CUDA
- **Your involvement can lead future evolution of PG-Strom**

Any Questions?

Thank you

ありがとうございました

THANK YOU

DĚKUJEME

DANKE

MERCI

GRAZIE

GRACIAS

Empowered by Innovation

NEC