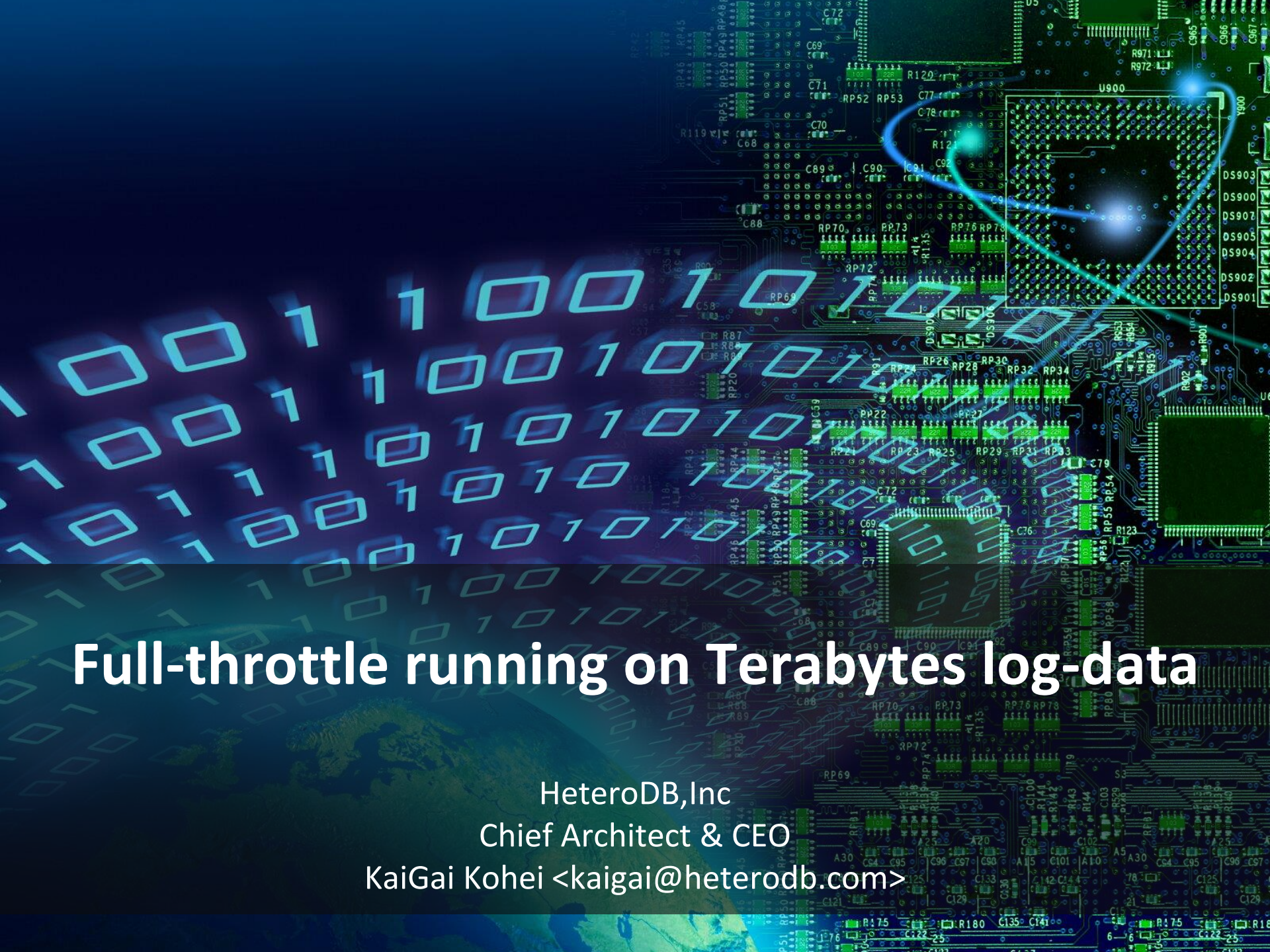


Full-throttle running on Terabytes log-data

HeteroDB,Inc
Chief Architect & CEO
KaiGai Kohei <kaigai@heterodb.com>

PGConf.Asia 2019

bali

Indonesia PostgreSQL User Group

EQUNIX BUSINESS SOLUTIONS

Japan PostgreSQL User Group

NAWA CITA PARIWISATA INDONESIA

UNIVERSITAS JEMBER

SRA OSS, INC

2ndQuadrant PostgreSQL

QISCUS

Self-Introduction

about myself

- KaiGai Kohei (海外浩平)
- Chief Architect & CEO of HeteroDB
- Contributor of PostgreSQL (2006-)
- Primary Developer of PG-Strom (2012-)
- Interested in: Big-data, GPU, NVME/PMEM, ...

about our company

HeteroDB

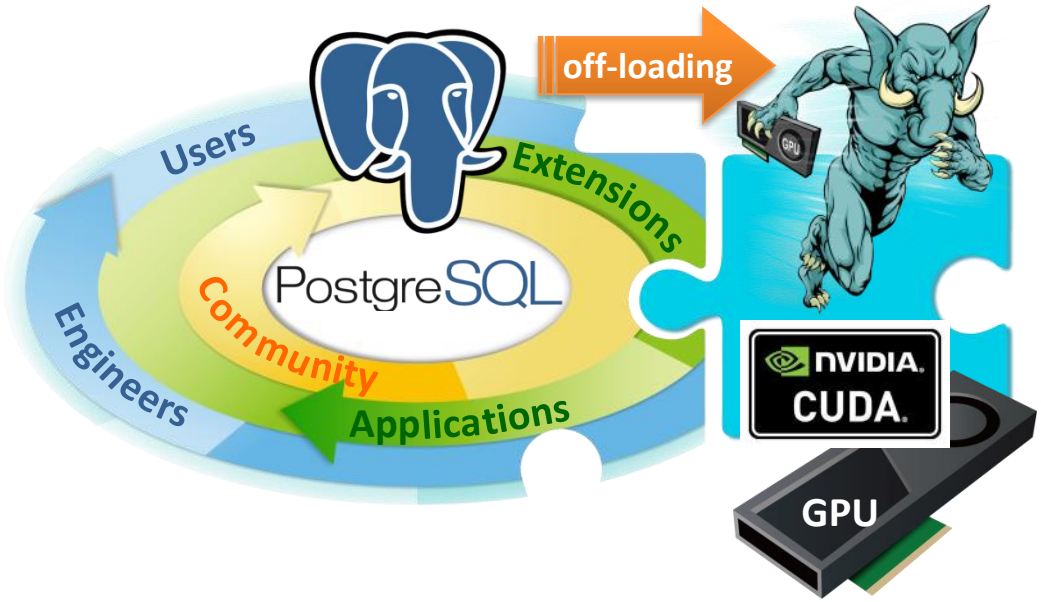
- Established: 4th-Jul-2017
- Location: Shinagawa, Tokyo, Japan
- Businesses:
  - ✓ Sales & development of high-performance data-processing software on top of heterogeneous architecture.
  - ✓ Technology consulting service on GPU&DB area.

NVIDIA INCEPTION PROGRAM

PG-Strom

What is PG-Strom?

PG-Strom is an extension of PostgreSQL for terabytes scale data-processing and inter-operation of AI/ML, by utilization of GPU and NVME-SSD.



for IoT/Big-Data

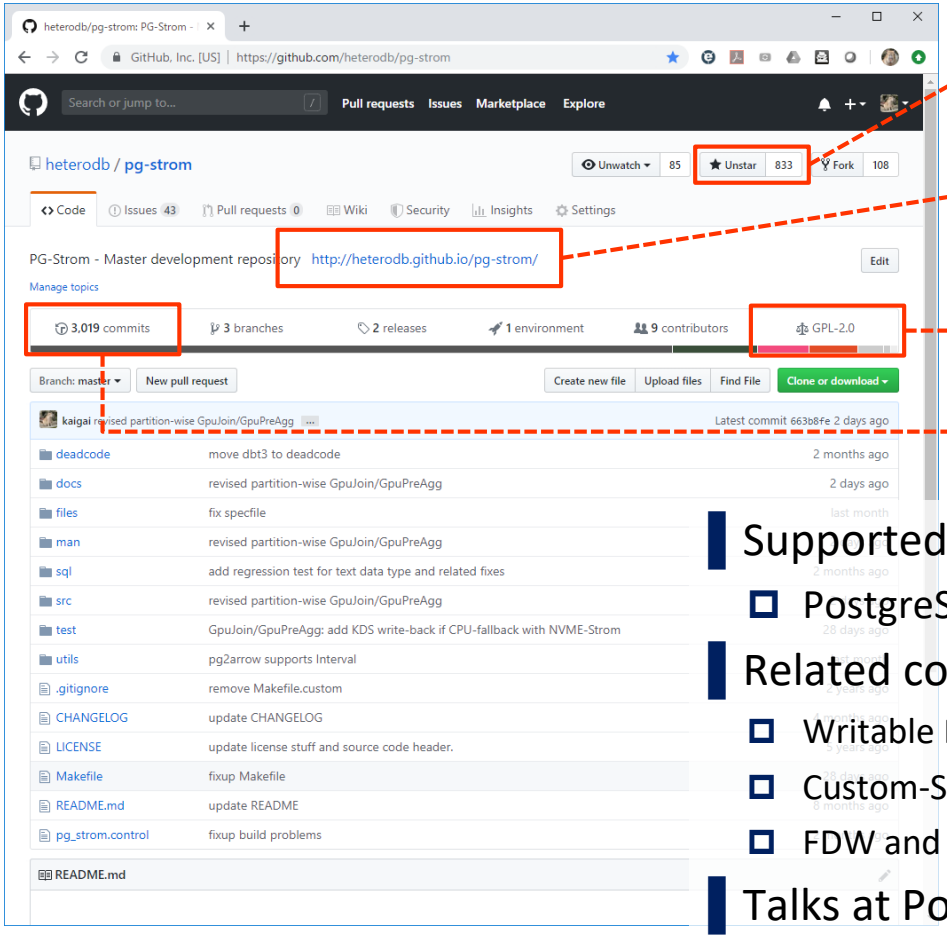
- SSD-to-GPU Direct SQL
- Columnar Store (Arrow_Fdw)
- Asymmetric Partition-wise JOIN/GROUP BY
- BRIN-Index Support
- NVME-over-Fabric Support

for ML/Analytics

- Procedural Language for GPU native code (PL/CUDA)
- NVIDIA RAPIDS data frame support (WIP)
- IPC on GPU device memory

- Transparent GPU acceleration for analytics and reporting workloads
- Binary code generation by JIT from SQL statements
- PCIe-bus level optimization by SSD-to-GPU Direct SQL technology
- Columnar-storage for efficient I/O and vector processors

PG-Strom as an open source project



Many stars ☺

Official documentation in English / Japanese
http://heterodb.github.io/pg-strom/

Distributed under GPL-v2.0

Has been developed since 2012

Supported PostgreSQL versions

- PostgreSQL v11.x, v10.x and v9.6.x

Related contributions to PostgreSQL

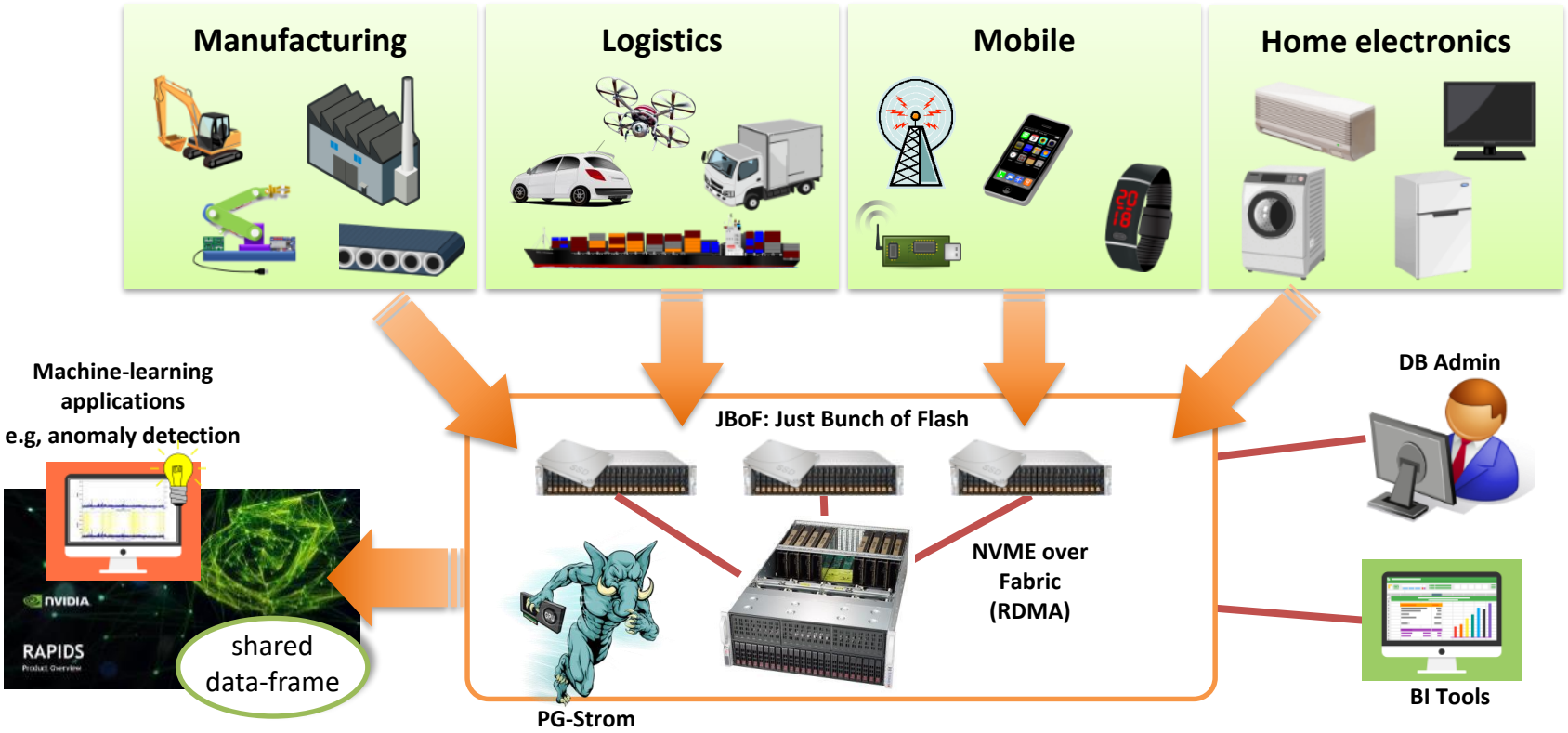
- Writable FDW support (9.3)
- Custom-Scan interface (9.5)
- FDW and Custom-Scan JOIN pushdown (9.5)

Talks at PostgreSQL community

- PGconf.EU 2012 (Prague), 2015 (Vienna), 2018 (Lisbon)
- PGconf.ASIA 2018 (Tokyo), 2019 (**Bali, Indonesia**)
- PGconf.SV 2016 (San Francisco)
- PGconf.China 2015 (Beijing)

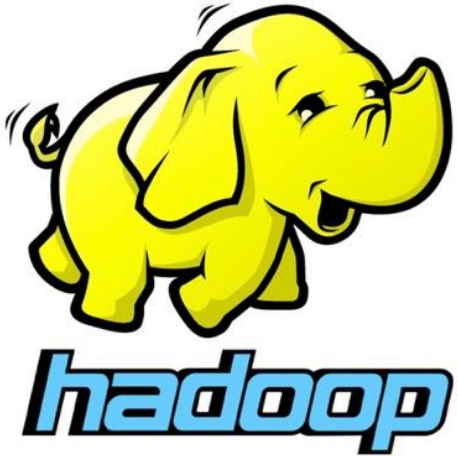
Terabytes scale log-data processing on PostgreSQL

System image of log-data platform



- Any kind of devices generate various form of log-data.
- People want/try to find out insight from the data.
- Data importing and processing must be rapid.
- System administration should be simple and easy.

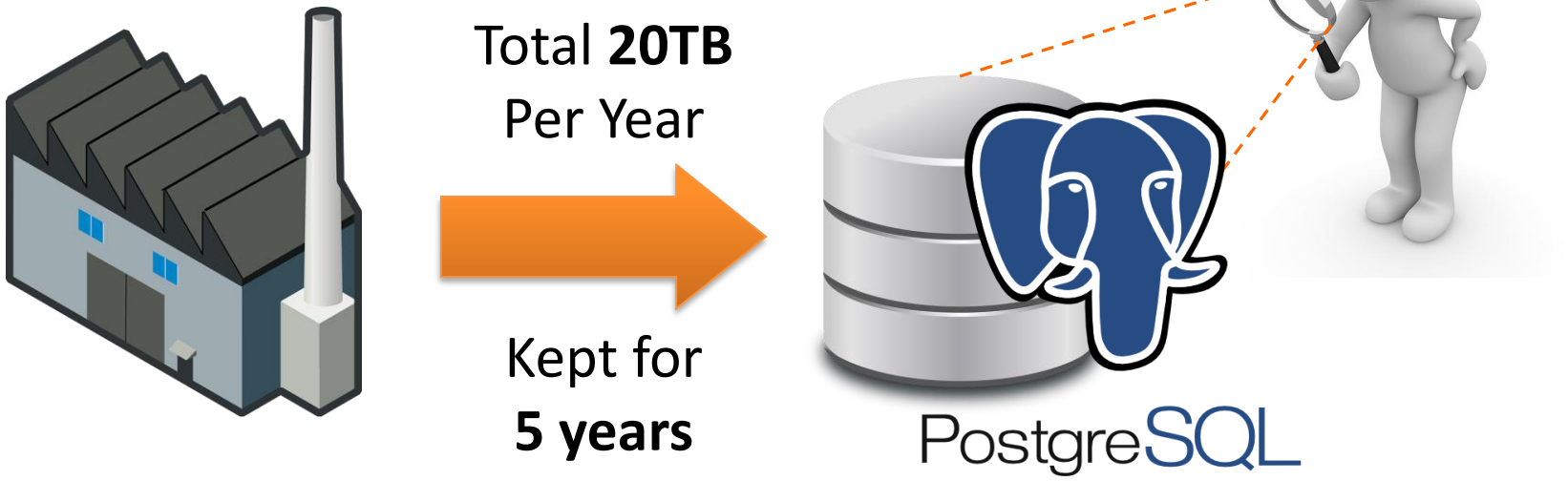
Why elephant is cobalt blue, not yellow.



- Here is PGconf.ASIA 2019 :-)
- Standalone system is much simpler than multi-node cluster system.
- Engineers are familiar with PostgreSQL for more than 10 years
- How many users actually have petabytes scale data?

How much data size we expect? (1/3)

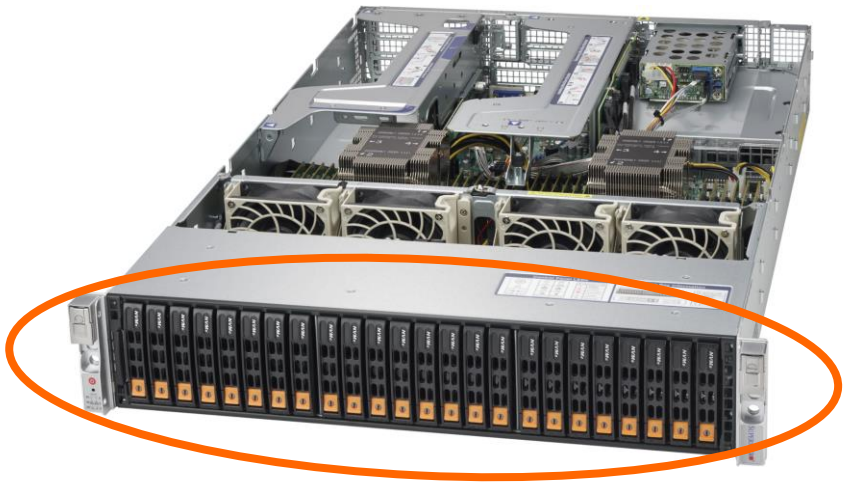
Case: A semiconductor factory managed up to 100TB with PostgreSQL



- This is a factory of an international top-brand company.
- Most of users don’t have bigger data than them.

How much data size we expect? (2/3)

Nowadays, 2U server is capable for 100TB capacity



8.0TB x 24 = 192TB

model	Supermicro 2029U-TN24R4T	Qty
CPU	Intel Xeon Gold 6226 (12C, 2.7GHz)	2
RAM	32GB RDIMM (DDR4-2933, ECC)	12
GPU	NVIDIA Tesla P40 (3840C, 24GB)	2
HDD	Seagate 1.0TB SATA (7.2krpm)	1
NVME	Intel DC P4510 (8.0TB, U.2)	24
N/W	built-in 10GBase-T	4

How much is it?

60,932USD

by thinkmate.com
(31st-Aug-2019)

How much data size we expect? (3/3)

On the other hands, it is not small enough.

Weapons to tackle terabytes scale data

Efficient Storage

- **SSD-to-GPU Direct SQL**
- Direct data transfer pulls out maximum performance of NVME for SQL workloads

Efficient Data Structure

- **Arrow_Fdw**
- Columnar store that is optimal for both of I/O throughput and vector processors

Partitioning

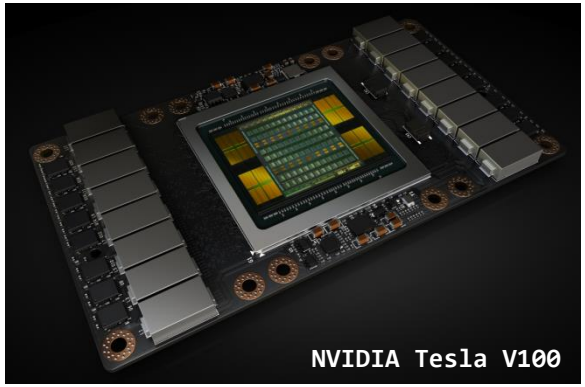
- **PostgreSQL Partitioning & Asymmetric Partition-wise JOIN**
- Prune the range obviously unreferenced

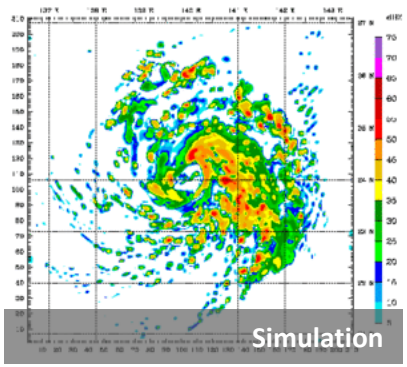
Efficient Storage

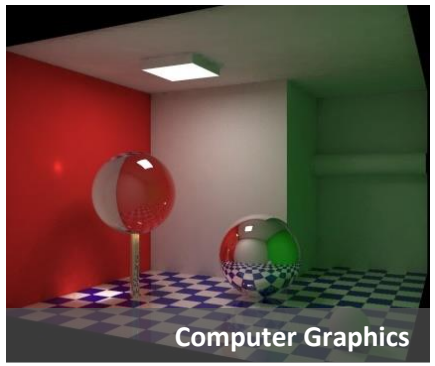
PG-Strom: SSD-to-GPU Direct SQL

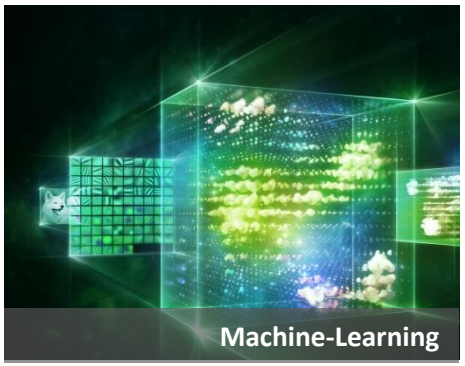
GPU's characteristics - mostly as a computing accelerator

Over 10years history in HPC, then massive popularization in Machine-Learning





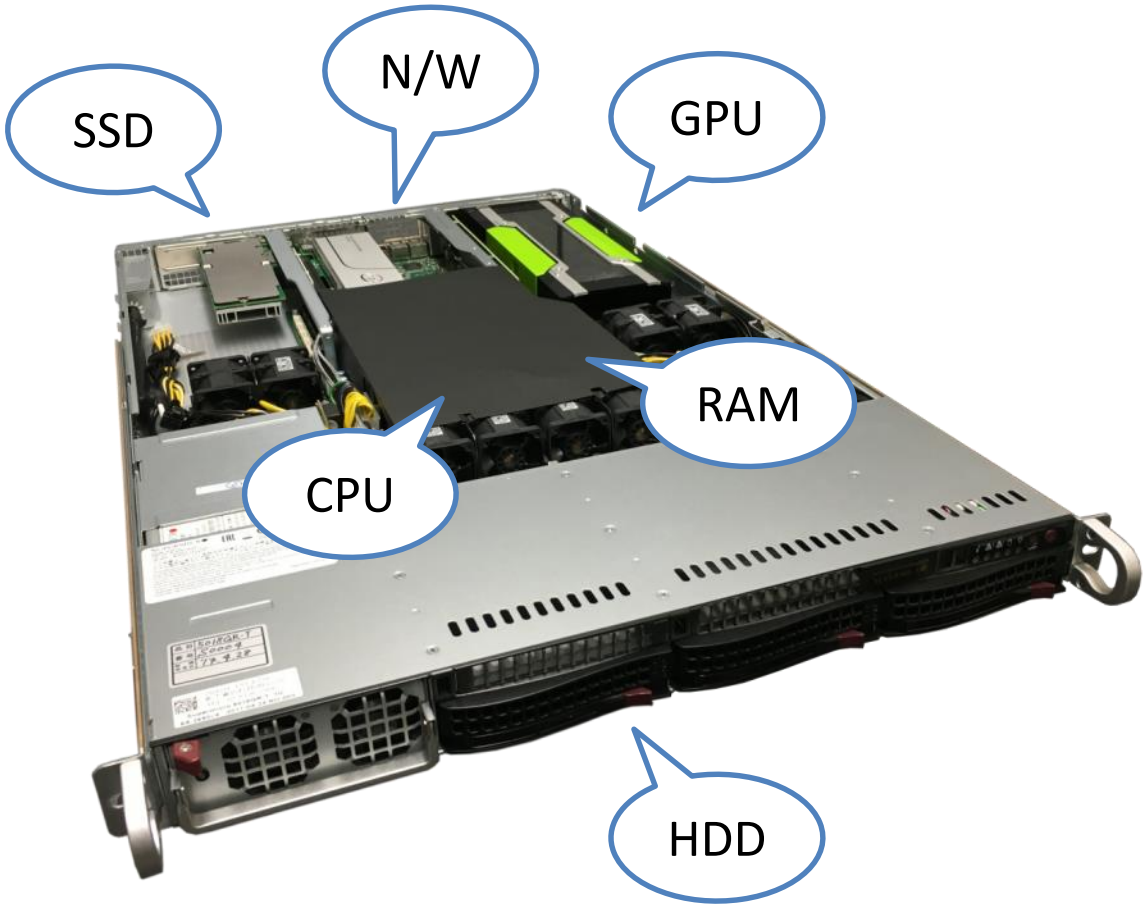




Today's Topic

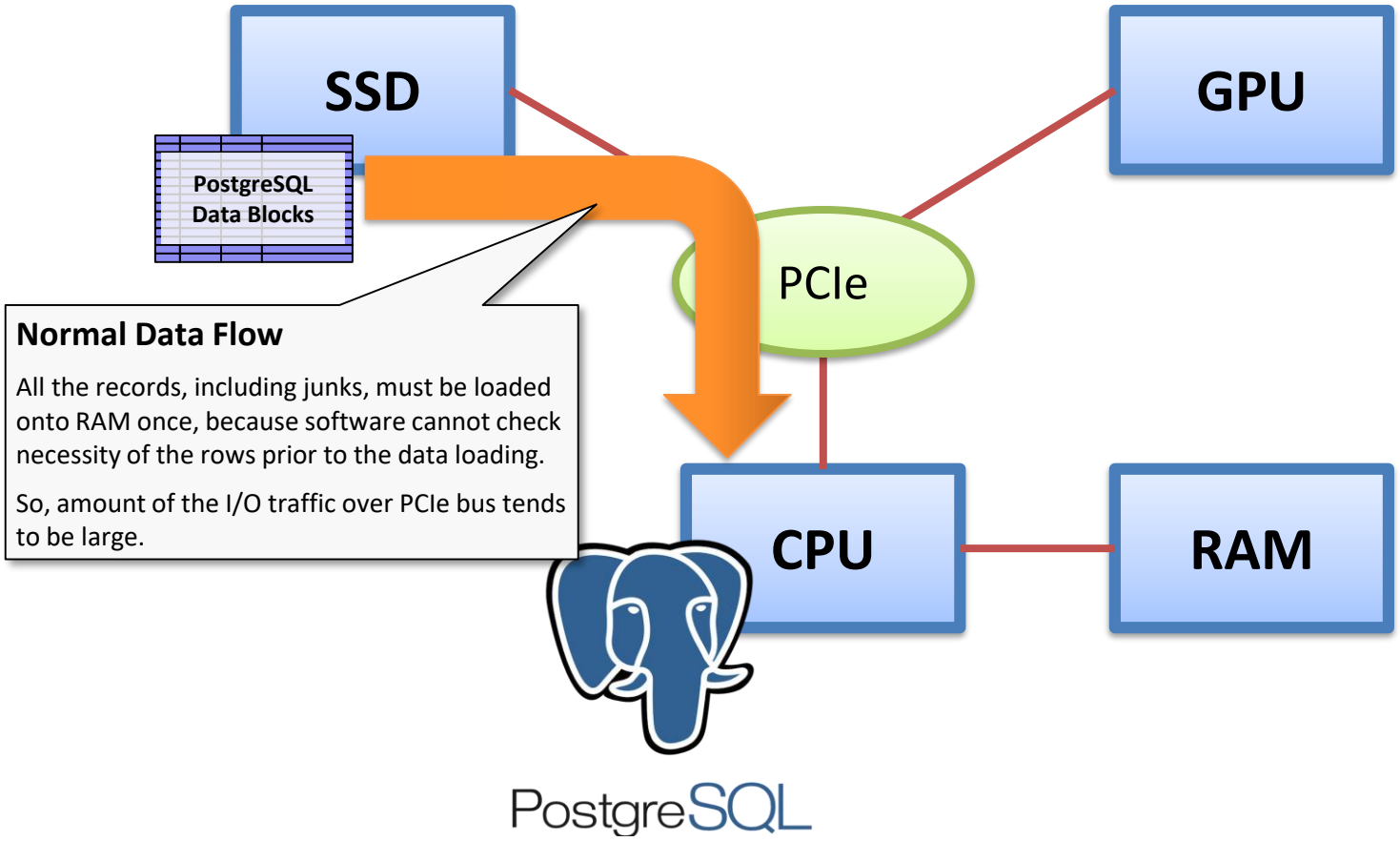
How I/O workloads are accelerated by GPU that is a **computing accelerator**?

A usual composition of x86_64 server



Data flow to process a massive amount of data

Unless records are not loaded to CPU/RAM once, over the PCIe bus, software cannot check its necessity even if they are “junk” records.



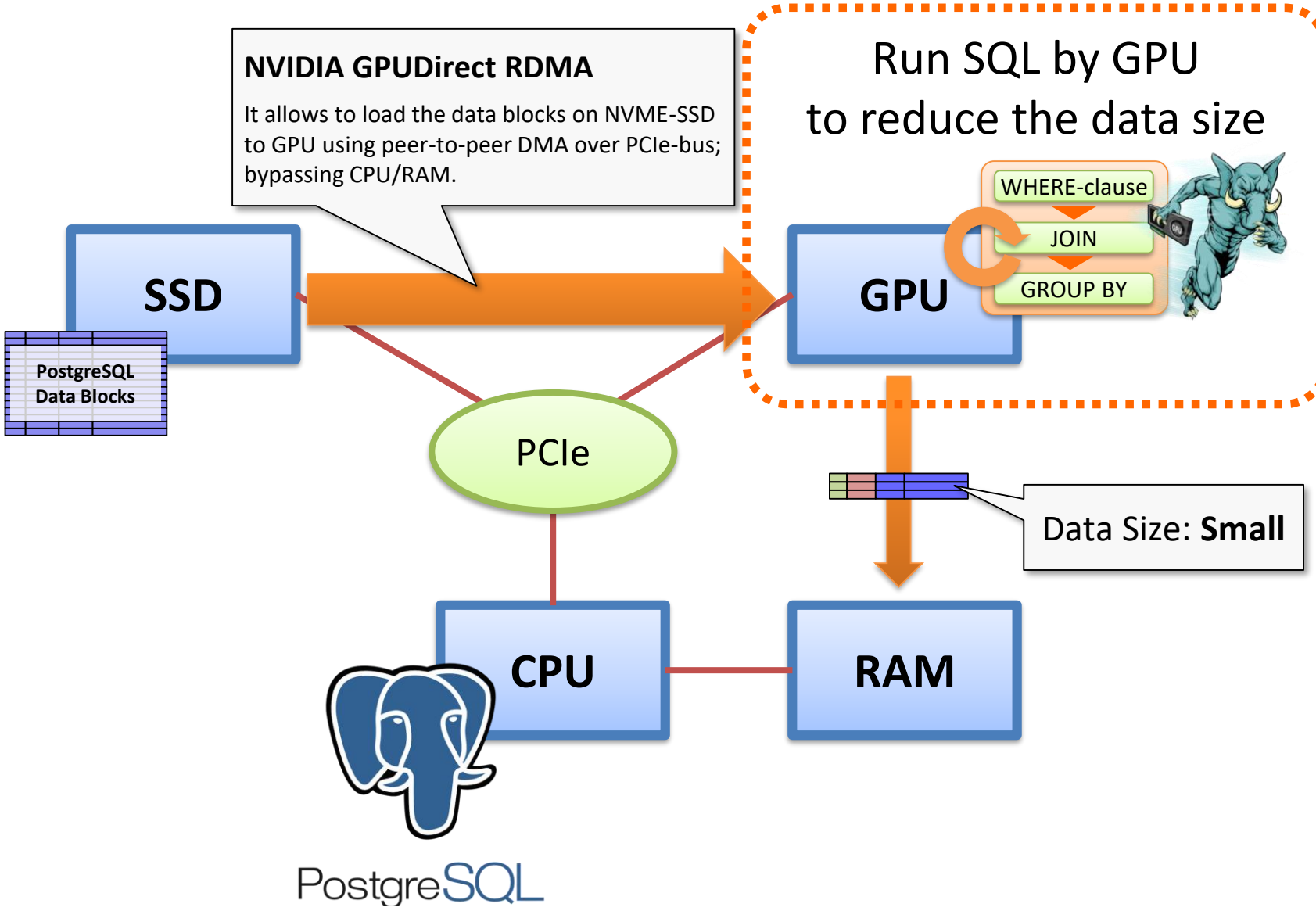
Normal Data Flow

All the records, including junks, must be loaded onto RAM once, because software cannot check necessity of the rows prior to the data loading.

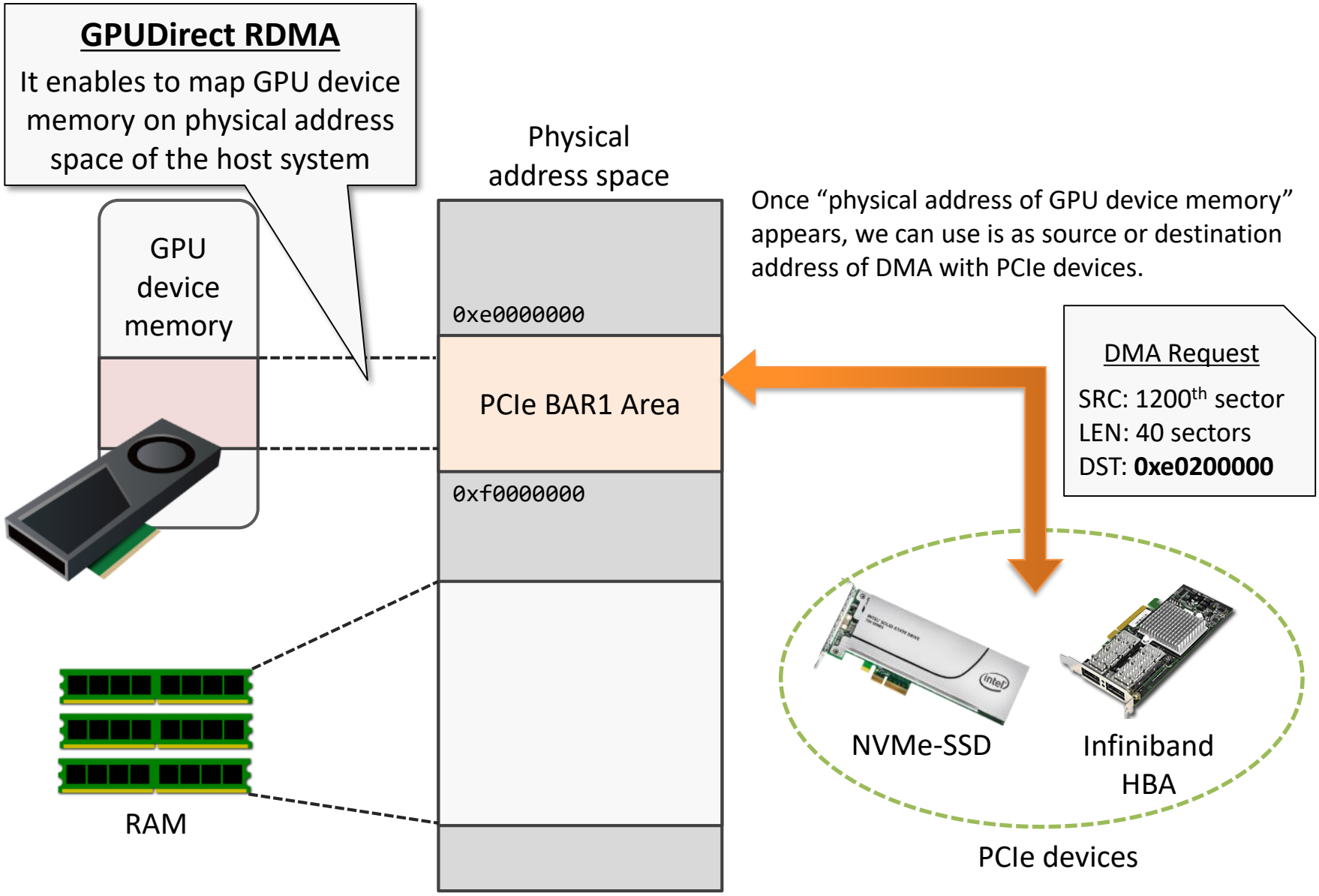
So, amount of the I/O traffic over PCIe bus tends to be large.

SSD-to-GPU Direct SQL (1/4) – Overview

v2.0



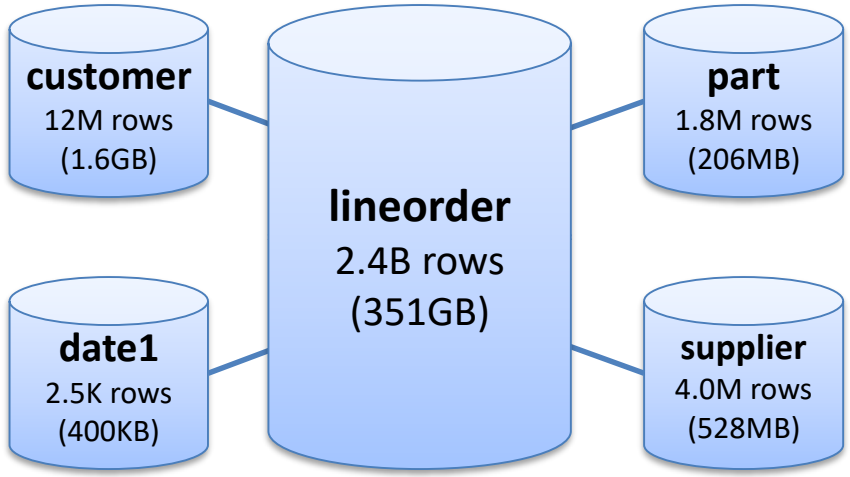
Background - GPUDirect RDMA by NVIDIA



SSD-to-GPU Direct SQL (2/4) – System configuration and workloads

Summarizing queries for typical Star-Schema structure on simple 1U server

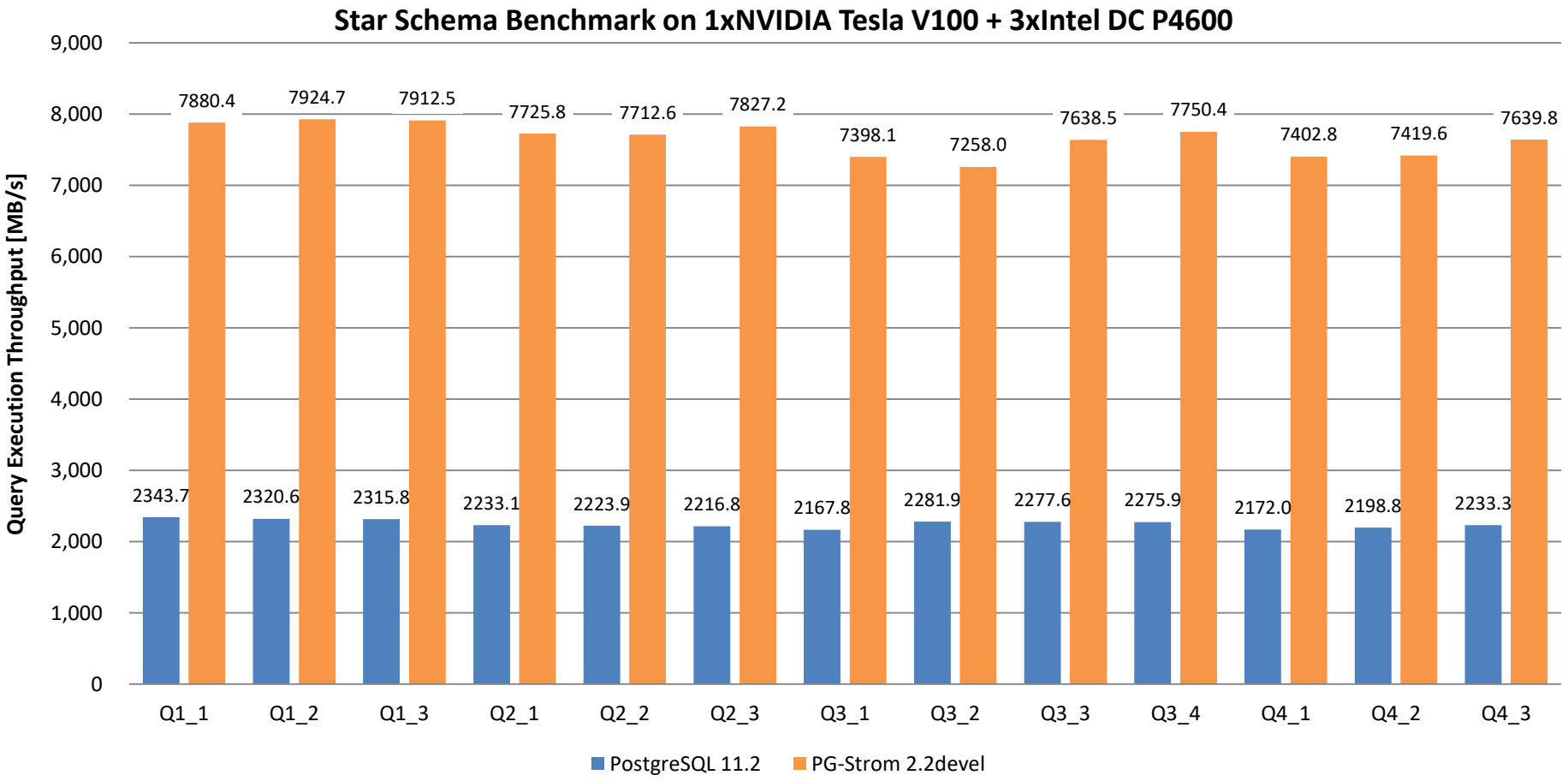
Supermicro SYS-1019GP-TT	
CPU	Xeon Gold 6126T (2.6GHz, 12C) x1
RAM	192GB (32GB DDR4-2666 x 6)
GPU	NVIDIA Tesla V100 (5120C, 16GB) x1
SSD	Intel SSD DC P4600 (HHHL; 2.0TB) x3 (striping configuration by md-raid0)
HDD	2.0TB(SATA; 72krpm) x6
Network	10Gb Ethernet 2ports
OS	Red Hat Enterprise Linux 7.6 CUDA 10.1 + NVIDIA Driver 418.40.04
DB	PostgreSQL v11.2 PG-Strom v2.2devel



■ Query Example (Q2_3)

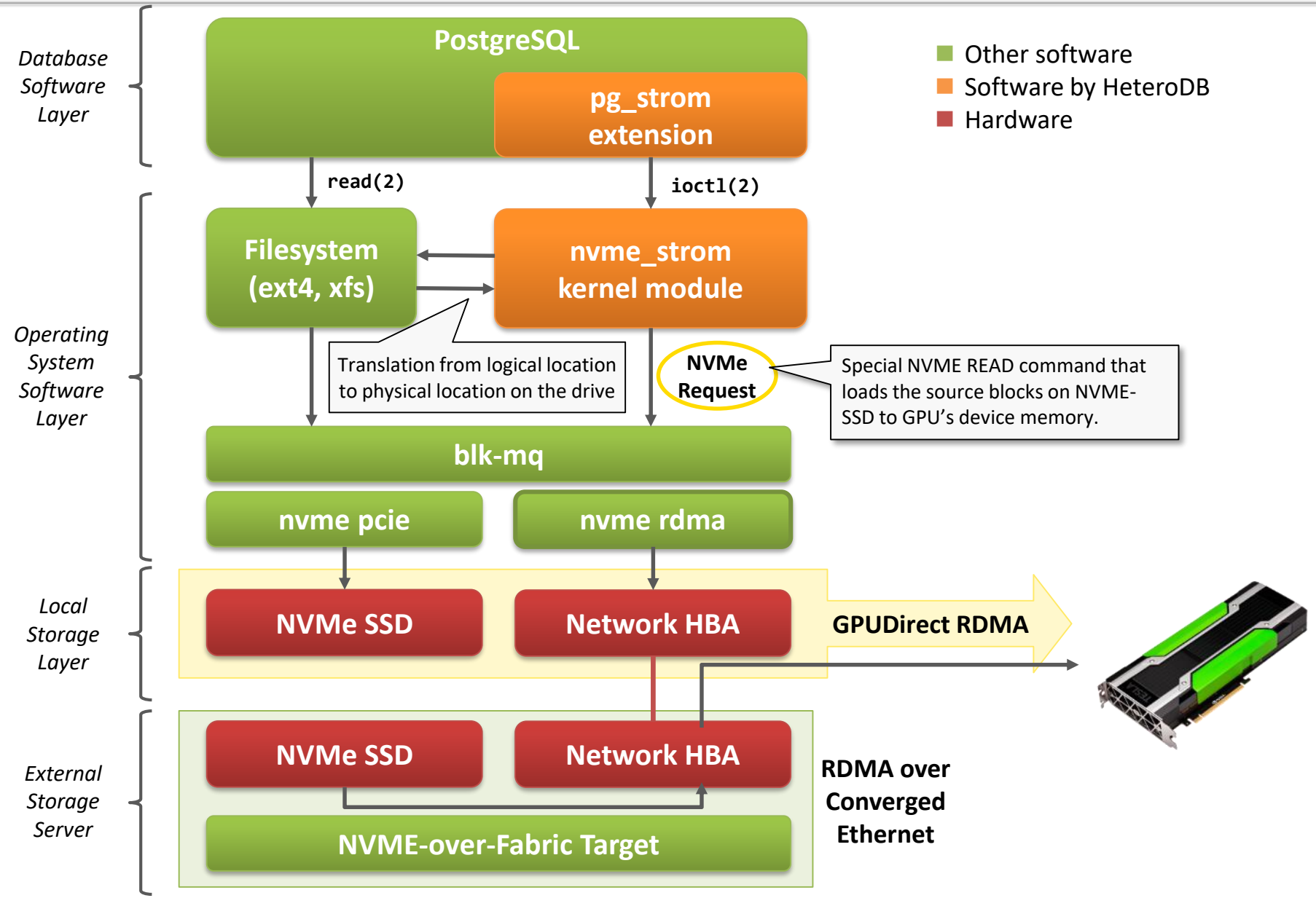
```
SELECT sum(lo_revenue), d_year, p_brand1
  FROM lineorder, date1, part, supplier
 WHERE lo_orderdate = d_datekey
   AND lo_partkey = p_partkey
   AND lo_suppkey = s_suppkey
   AND p_category = 'MFGR#12‘
   AND s_region = 'AMERICA‘
 GROUP BY d_year, p_brand1
 ORDER BY d_year, p_brand1;
```

SSD-to-GPU Direct SQL (3/4) – Benchmark results



- Query Execution Throughput = (353GB; DB-size) / (Query response time [sec])
- SSD-to-GPU Direct SQL runs the workloads close to the hardware limitation (8.5GB/s)
- about x3 times faster than filesystem and CPU based mechanism

SSD-to-GPU Direct SQL (4/4) – Software Stack



Efficient Data Structure

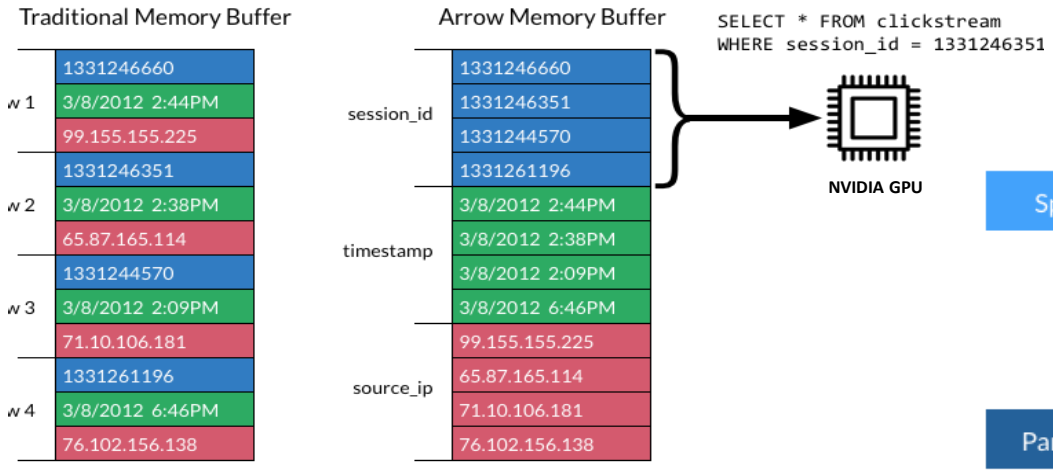
PG-Strom: Arrow_Fdw (Columnar Store)

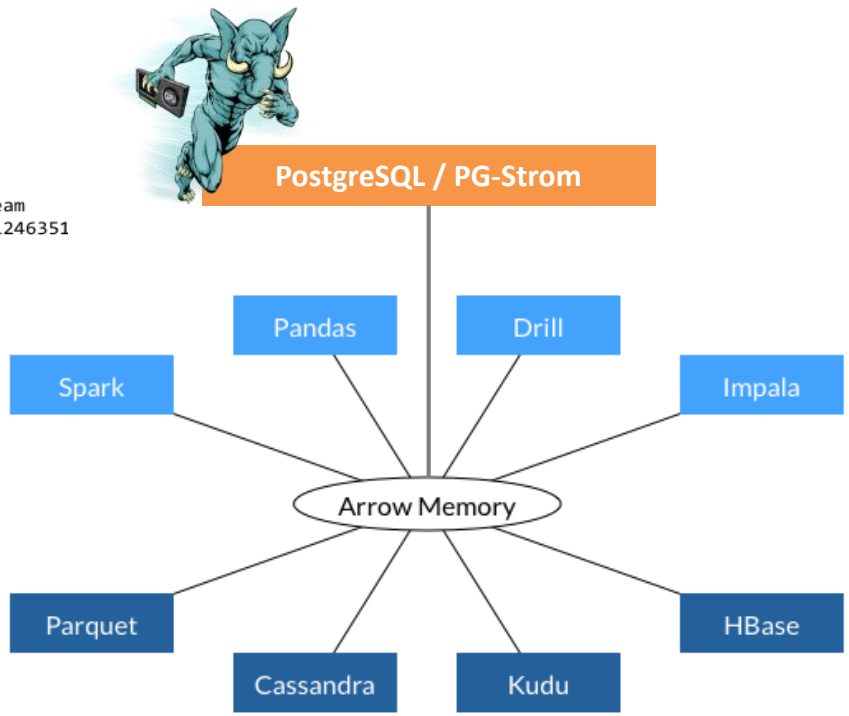
Background: Apache Arrow (1/2)

Characteristics

- **Column-oriented data format** designed for analytics workloads
- **A common data format** for inter-application exchange
- Various primitive data types like integer, floating-point, date/time and so on

	session_id	timestamp	source_ip
1	1331246660	3/8/2012 2:44PM	99.155.155.225
2	1331246351	3/8/2012 2:38PM	65.87.165.114
3	1331244570	3/8/2012 2:09PM	71.10.106.181
4	1331261196	3/8/2012 6:46PM	76.102.156.138





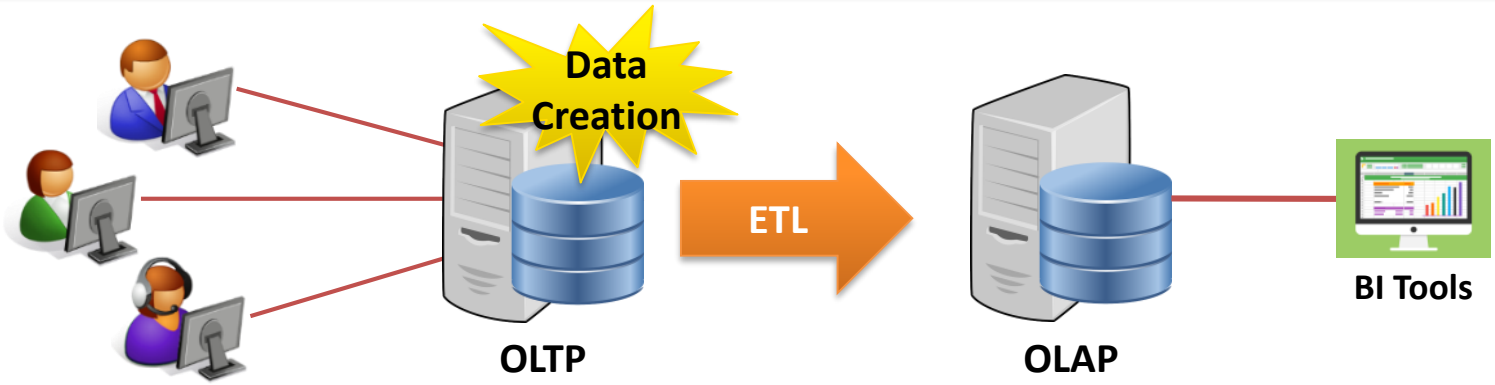
Background: Apache Arrow (2/2)

Most of data types are convertible between Apache Arrow and PostgreSQL.

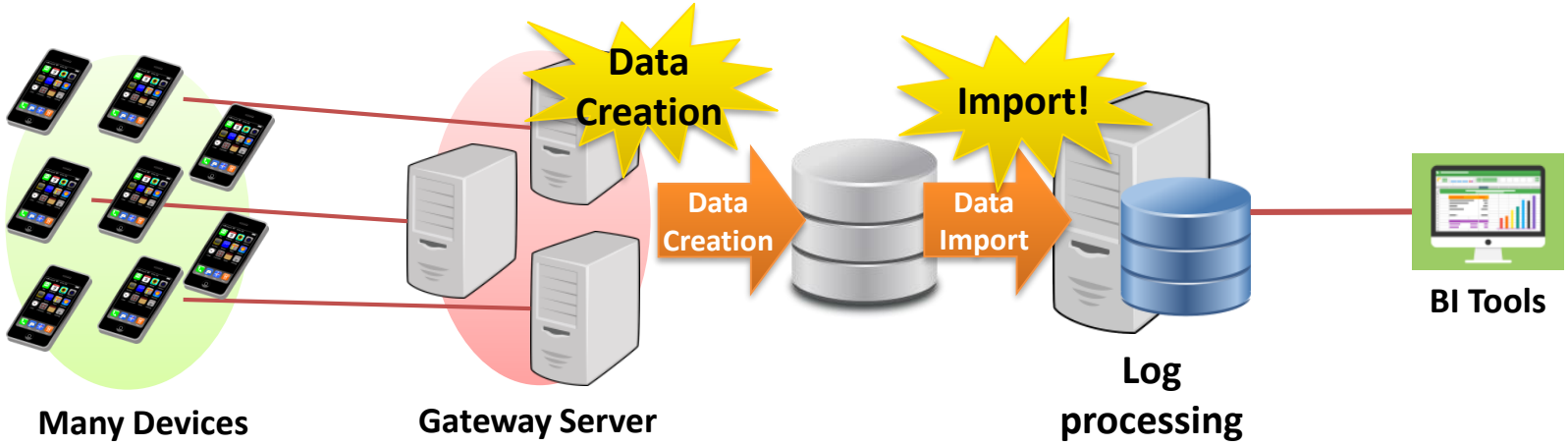
Apache Arrow Data Types	PostgreSQL Data Types	extra description
Int	int2, int4, int8	
FloatingPoint	float2, float4, float8	float2 is an enhancement of PG-Strom
Binary	bytea	
Utf8	text	
Bool	bool	
Decimal	numeric	
Date	date	adjusted to unitsz = Day
Time	time	adjusted to unitsz = MicroSecond
Timestamp	timestamp	adjusted to unitsz = MicroSecond
Interval	interval	
List	array types	Only 1-dimensional array is supportable
Struct	composite types	
Union	------	
FixedSizeBinary	char(n)	
FixedSizeList	------	
Map	------	

Log-data characteristics (1/2) – WHERE is it generated on?

Traditional OLTP&OLAP – Data is generated **inside** of database system



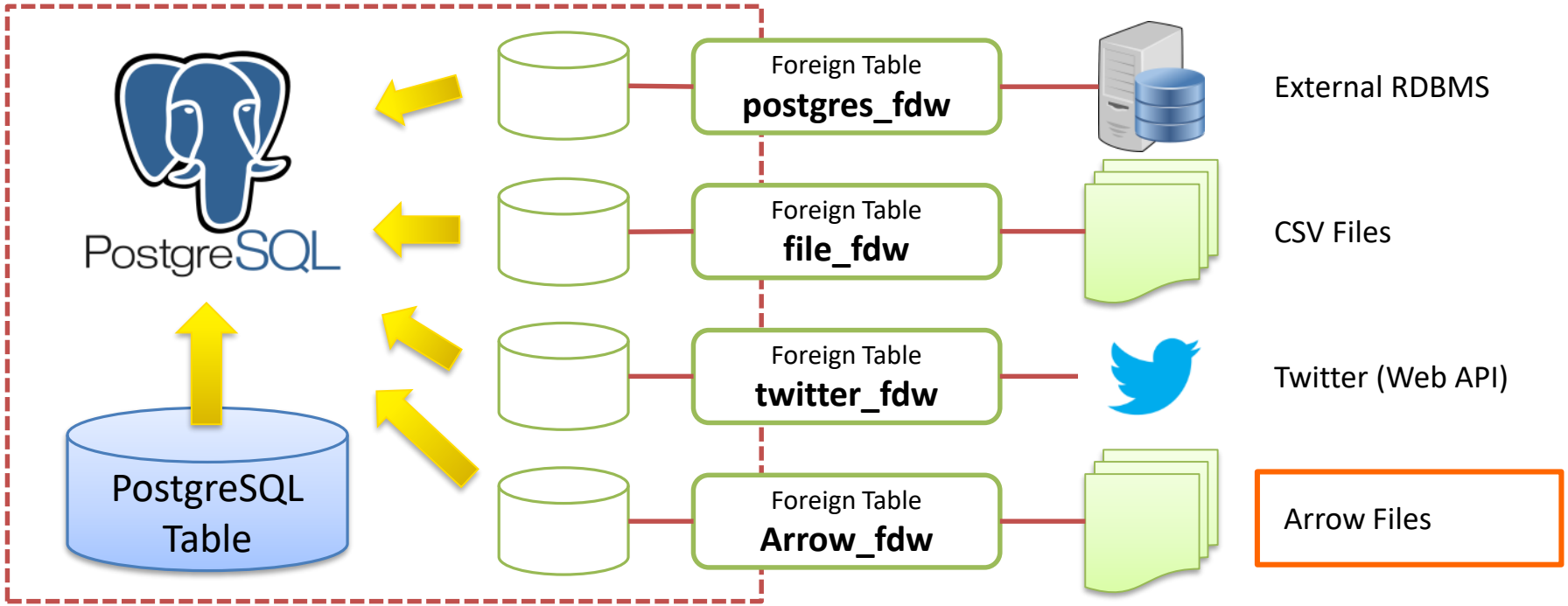
IoT/M2M use case – Data is generated **outside** of database system



Data Importing becomes a heavy time-consuming operations for big-data processing.

Background – FDW (Foreign Data Wrapper)

Foreign Table – it allows to read (and potentially write) external data source as like normal PostgreSQL tables, for SQL commands.

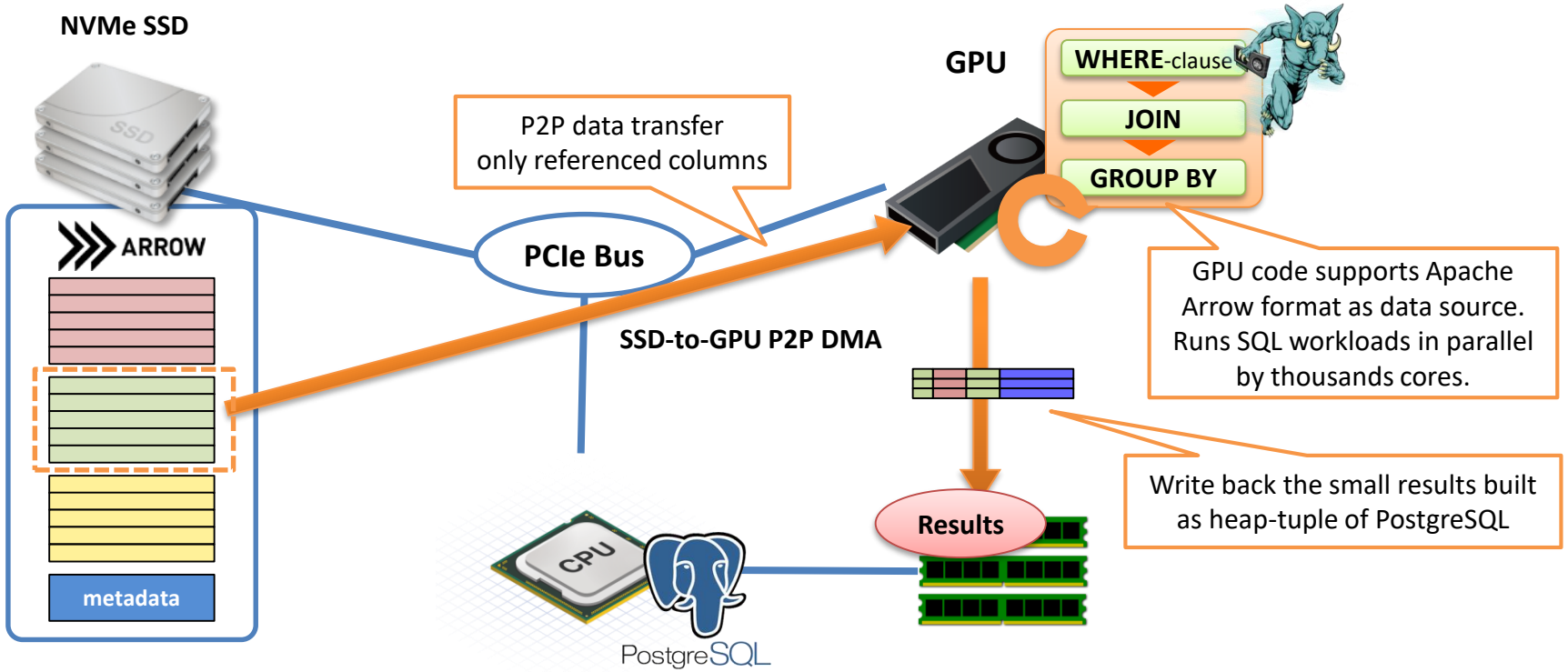


- FDW module is responsible to transform external data into PostgreSQL internal data.
- In case of Arrow_Fdw, it maps Arrow-files on the filesystem as foreign-table.

➔ Just mapping, so **no need to import the external data again**.

SSD-to-GPU Direct SQL on Arrow_Fdw (1/3)

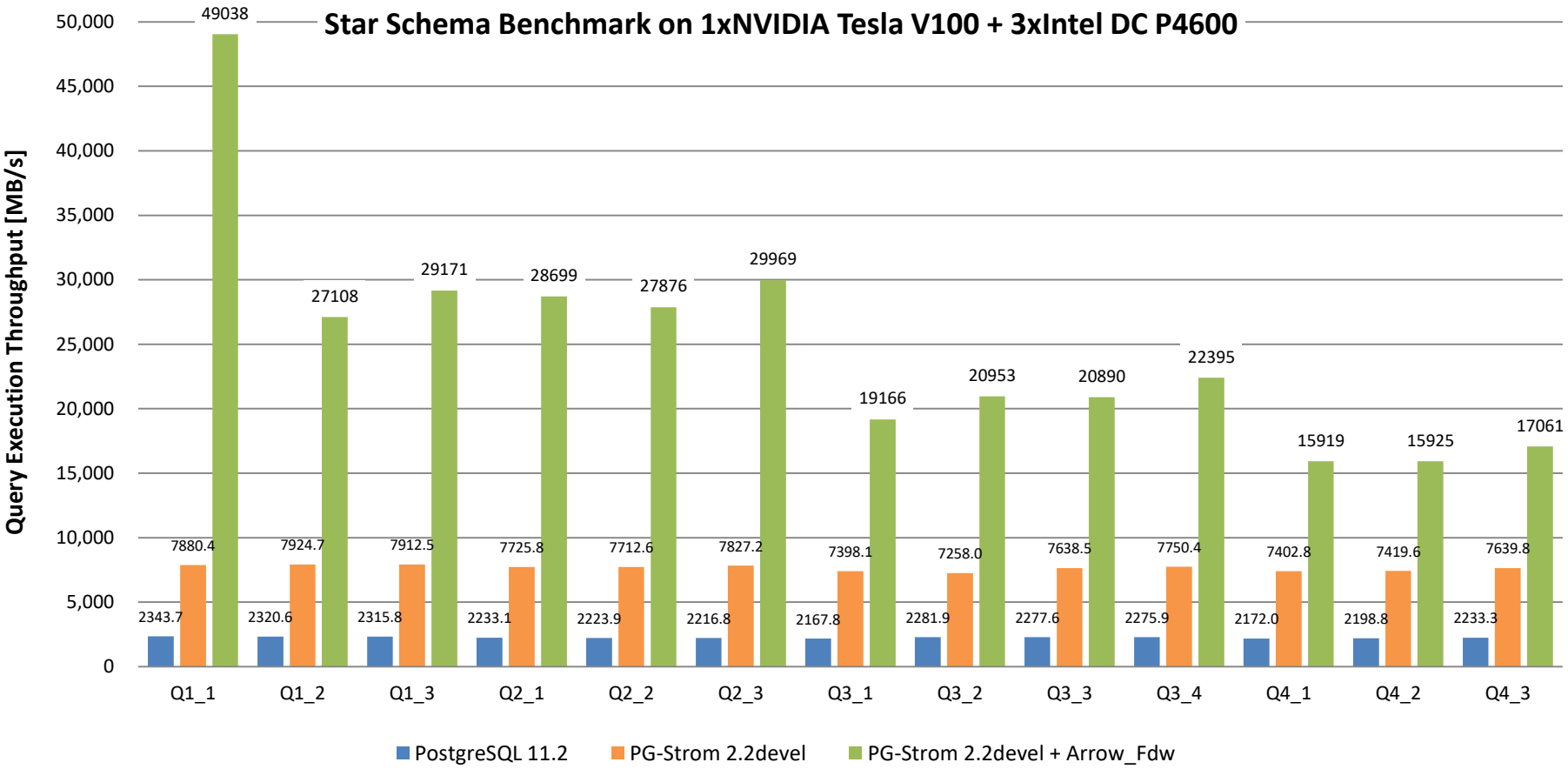
It transfers **ONLY Referenced Columns** over SSD-to-GPU Direct SQL mechanism



Why Apache Arrow is beneficial?

- Less amount of I/O to be loaded; only referenced columns
- Higher utilization of GPU core; by vector processing and wide memory bus
- Read-only structure; No MVCC checks are required on run-time

SSD-to-GPU Direct SQL on Arrow_Fdw (2/3) – Benchmark results



- Query Execution Throughput = (353GB; DB or 310GB; arrow) / (Query response time[s])
- Combined use of SSD-to-GPU Direct SQL and Columnar-store pulled out 15GB-49GB/s query execution throughput according to the number of referenced columns.
- See p.10 for the server configuration; just 1U-rackserver with 1 CPU+1 GPU+3 SSD

SSD-to-GPU Direct SQL on Arrow_Fdw (3/3) – Validation of the results

Almost equivalent raw data transfer, but no need to copy unreferenced columns.

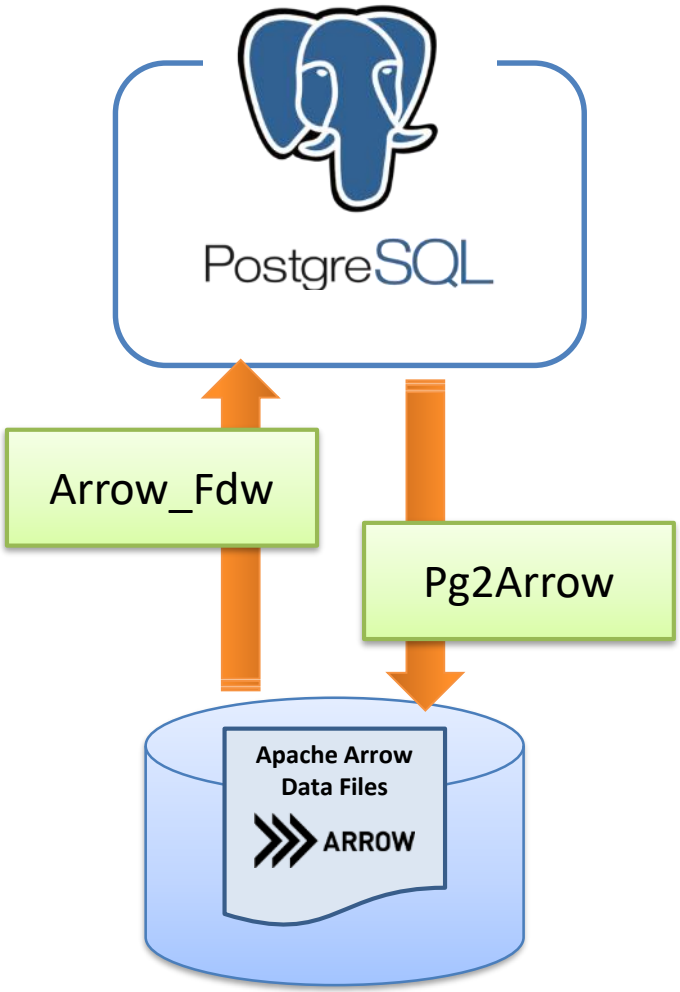
```
 Foreign table "public.flineorder"
       Column        |     Type      | Size
---------------------+---------------+---------------------------------
 lo_orderkey         | numeric       | 35.86GB
 lo_linenumber       | integer       |  8.96GB
 lo_custkey          | numeric       | 35.86GB
 lo_partkey          | integer       |  8.96GB
 lo_suppkey          | numeric       | 35.86GB   <-- ★Referenced by Q2_1
 lo_orderdate        | integer       |  8.96GB   <-- ★Referenced by Q2_1
 lo_orderpriority    | character(15) | 33.61GB   <-- ★Referenced by Q2_1
 lo_shippriority     | character(1)  |  2.23GB
 lo_quantity         | integer       |  8.96GB
 lo_extendedprice    | bigint        | 17.93GB
 lo_ordertotalprice  | bigint        | 17.93GB
 lo_discount         | integer       |  8.96GB
 lo_revenue          | bigint        | 17.93GB
 lo_supplycost       | bigint        | 17.93GB   <-- ★Referenced by Q2_1
 lo_tax              | integer       |  8.96GB
 lo_commit_date      | character(8)  | 17.93GB
 lo_shipmode         | character(10) | 22.41GB
FDW options: (file '/opt/nvme/lineorder_s401.arrow')  ... file size = 310GB
```

- Only 96.4GB of 310GB was actually read from NVME-SSD (31.08%)
- Execution time of Q2_1 is 11.06s, so 96.4GB / 11.06s = 8.7GB/s

➔ It is a reasonable performance for 3x Intel DC P4600 on single CPU configuration

Generation of Apache Arrow files

Pg2Arrow command saves SQL results in Arrow-format.

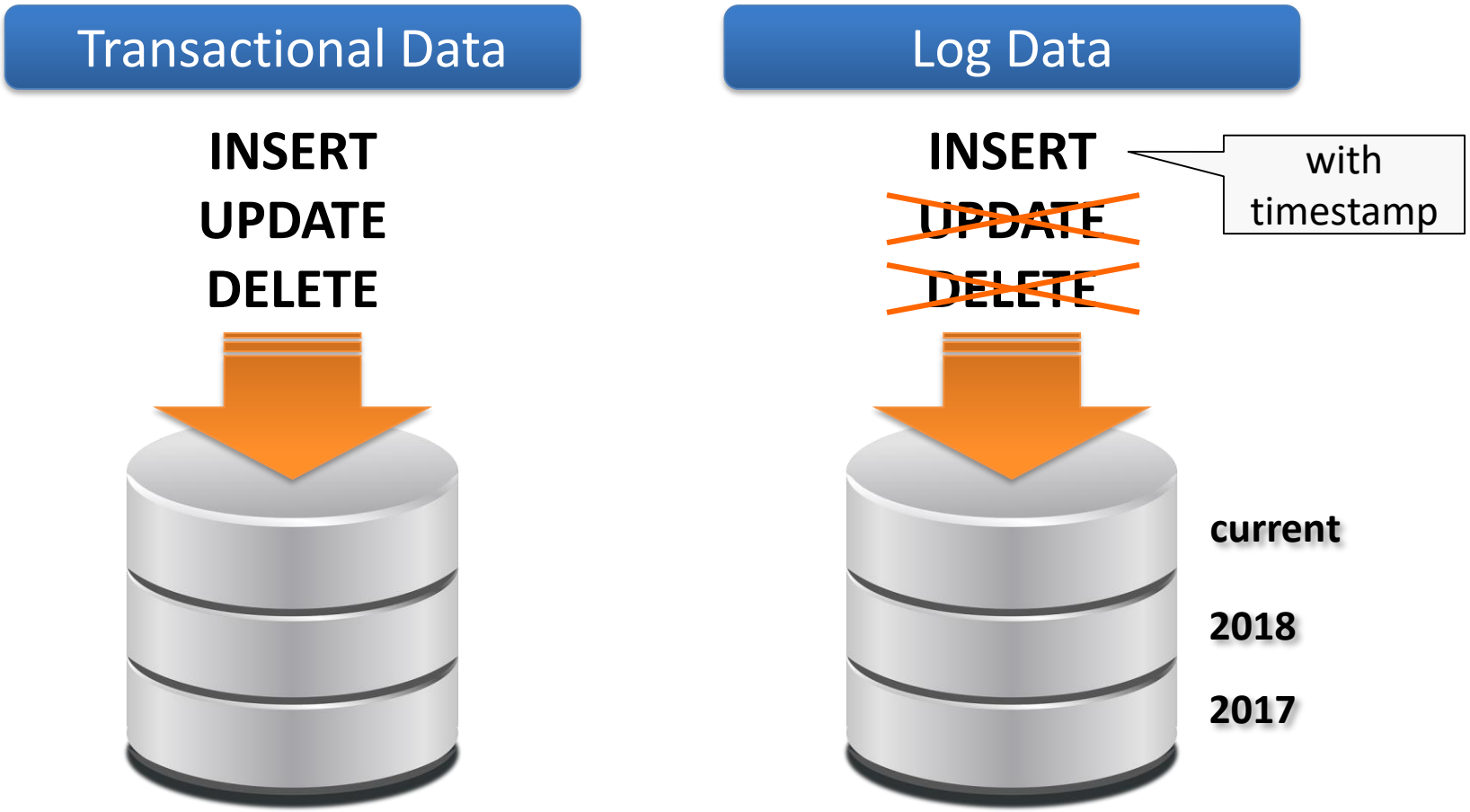


- ✓ Simply, specifies the SQL to run by "-c" and Arrow-file to be written by "-o" option.

```
$./pg2arrow -h
Usage:
  pg2arrow [OPTION]... [DBNAME [USERNAME]]
General options:
  -d, --dbname=DBNAME      database name to connect to
  -c, --command=COMMAND    SQL command to run
  -f, --file=FILENAME      SQL command from file
  -o, --output=FILENAME    result file in Apache Arrow format

Arrow format options:
  -s, --segment-size=SIZE size of record batch for each
      (default is 256MB)

Connection options:
  -h, --host=HOSTNAME      database server host
  -p, --port=PORT          database server port
  -U, --username=USERNAME database user name
  -w, --no-password        never prompt for password
  -W, --password           force password prompt

Debug options:
      --dump=FILENAME      dump information of arrow file
      --progress           shows progress of the job.
```

PostgreSQL Partitioning and PCIe-bus level optimization

Log-data characteristics (2/2) – INSERT-only

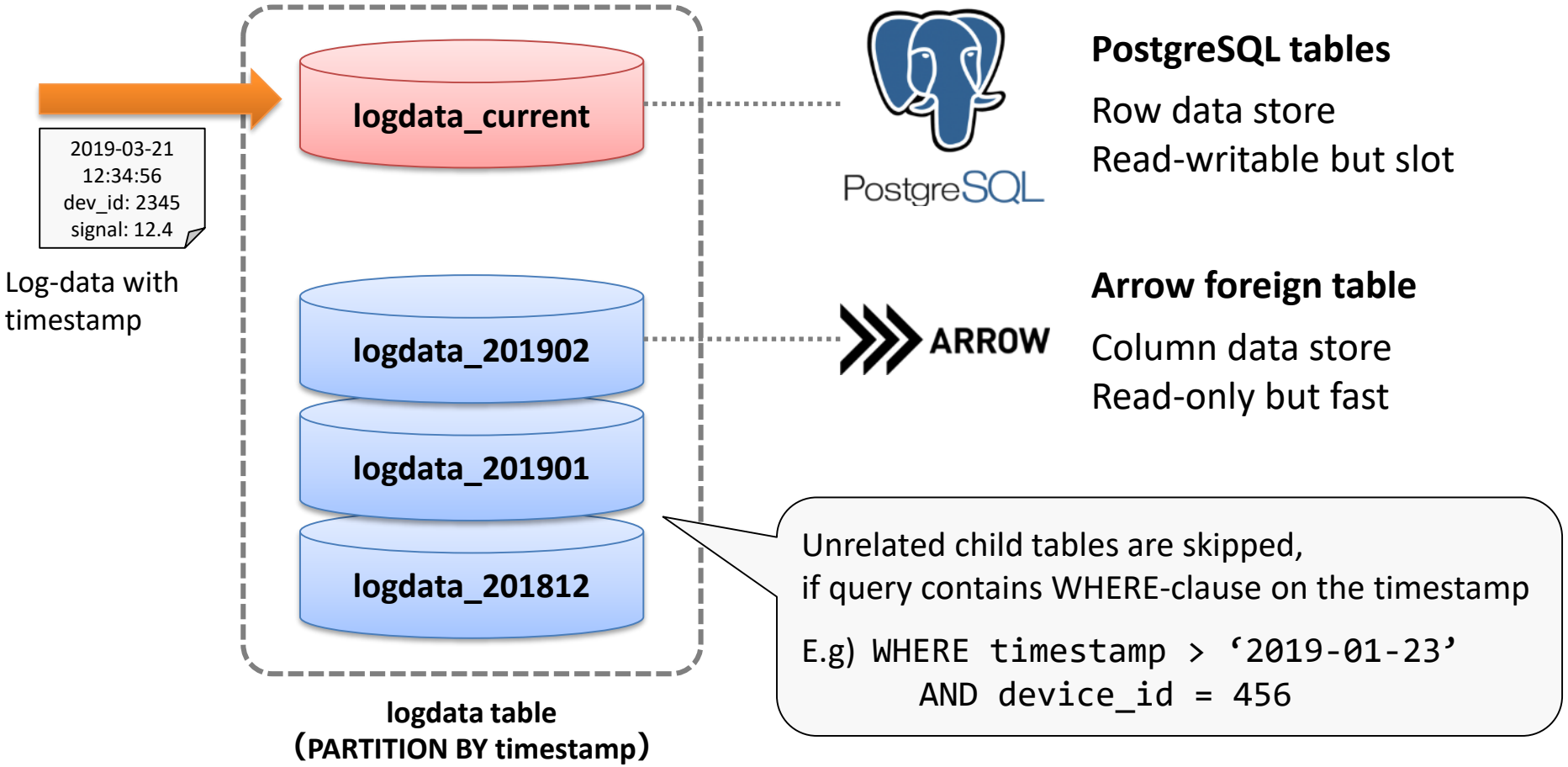


- ✓ MVCC visibility check is (relatively) not significant.
- ✓ Rows with old timestamp will never inserted.

Configuration of PostgreSQL partition for log-data (1/2)

- Mixture of PostgreSQL table and Arrow foreign table in partition declaration
- Log-data should have timestamp, and never updated

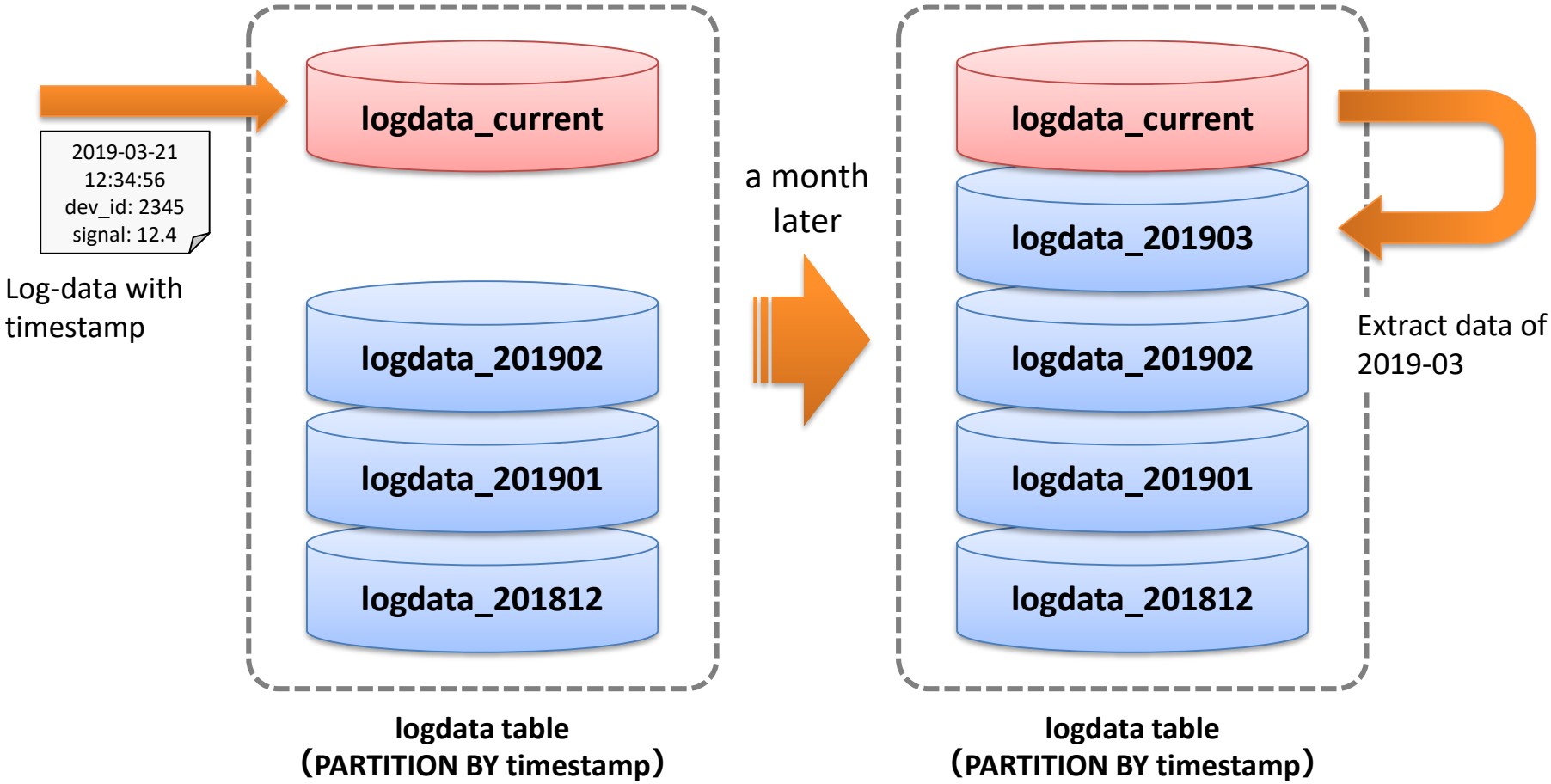
➔ Old data can be moved to Arrow foreign table for more efficient I/O



Configuration of PostgreSQL partition for log-data (2/2)

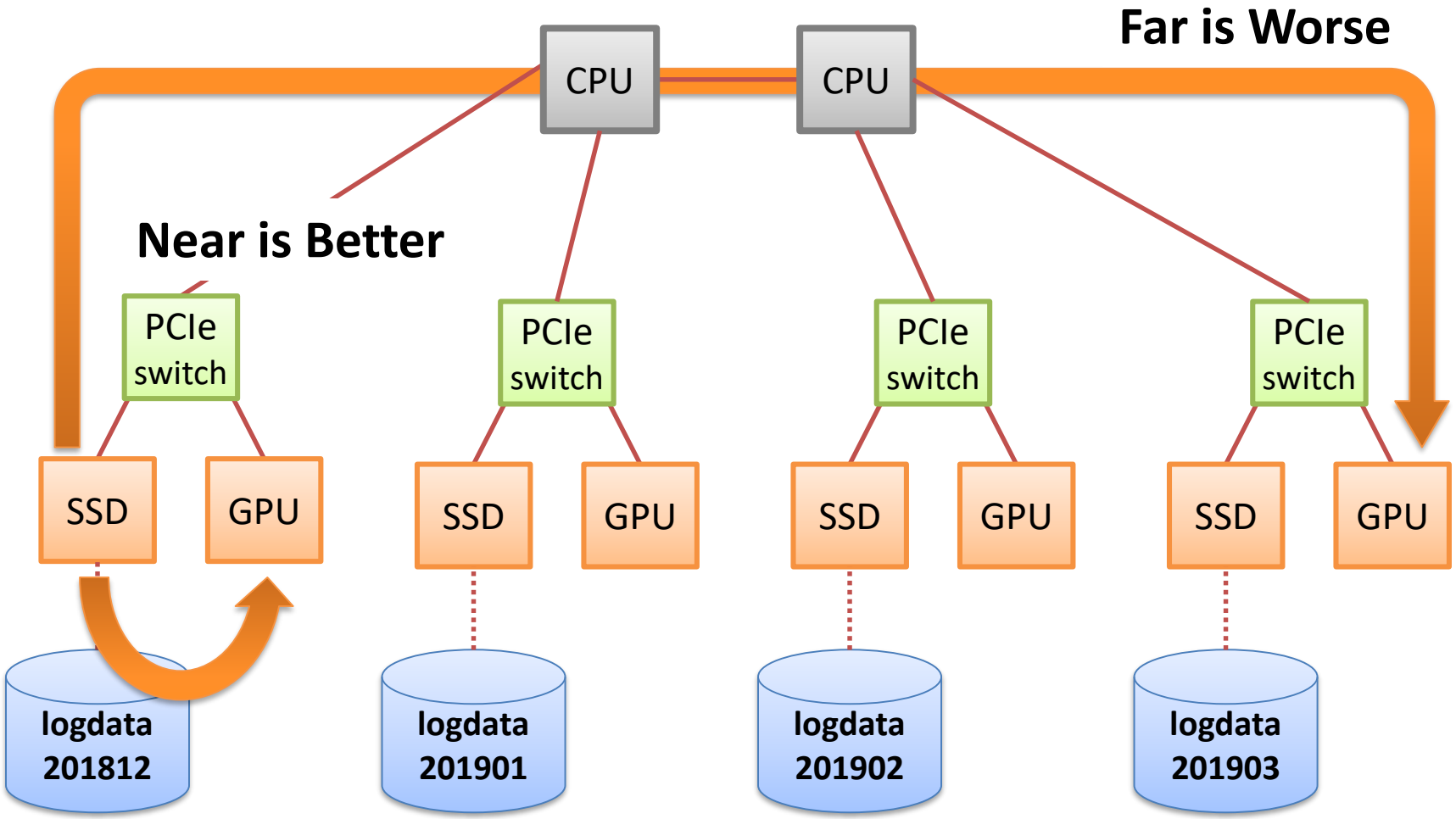
- Mixture of PostgreSQL table and Arrow foreign table in partition declaration
- Log-data should have timestamp, and never updated

➔ Old data can be moved to Arrow foreign table for more efficient I/O



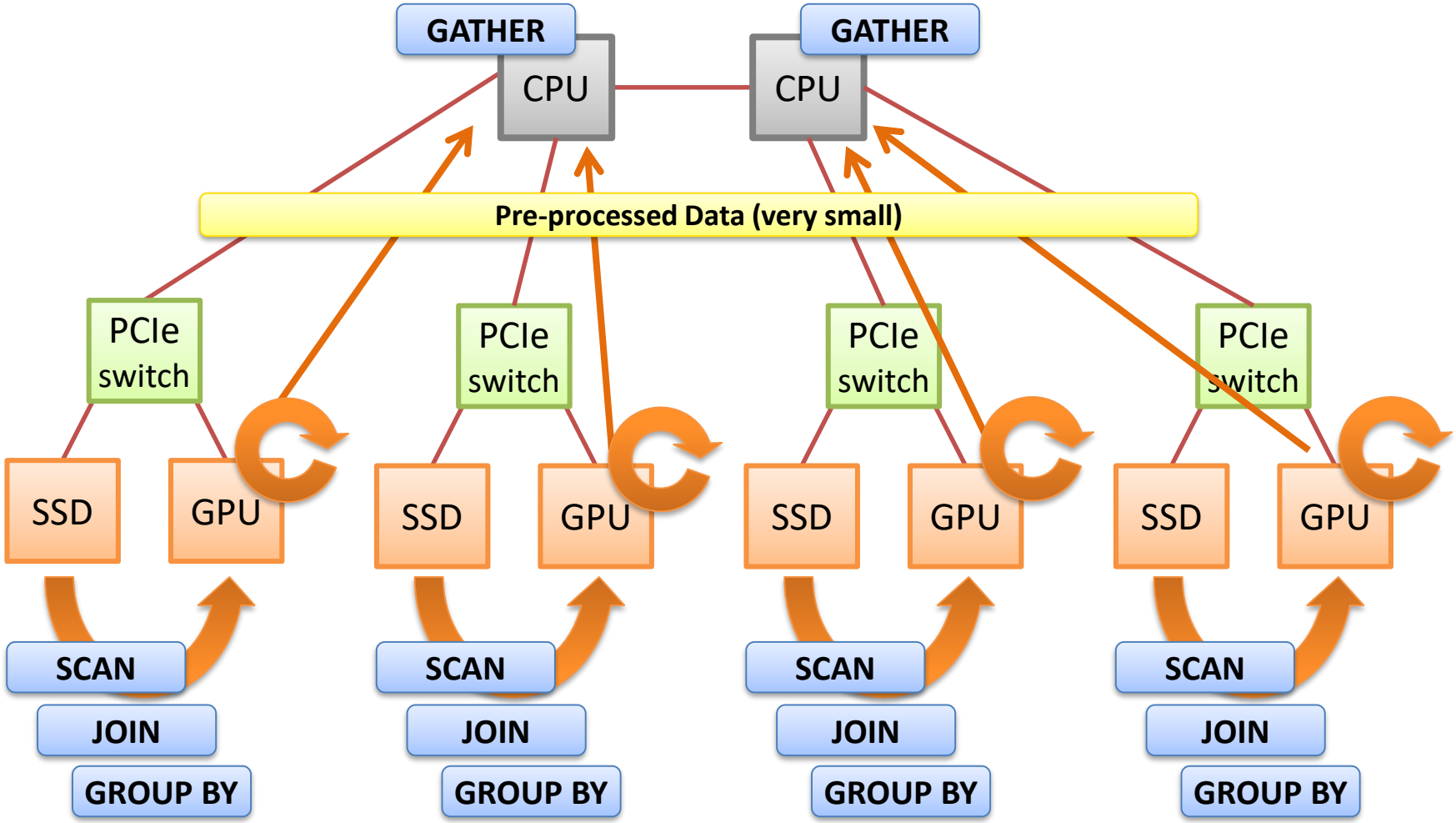
PCIe-bus level optimization (1/3)

Physical data distribution, and utilization of **closest** GPU



PCIe-bus level optimization (2/3)

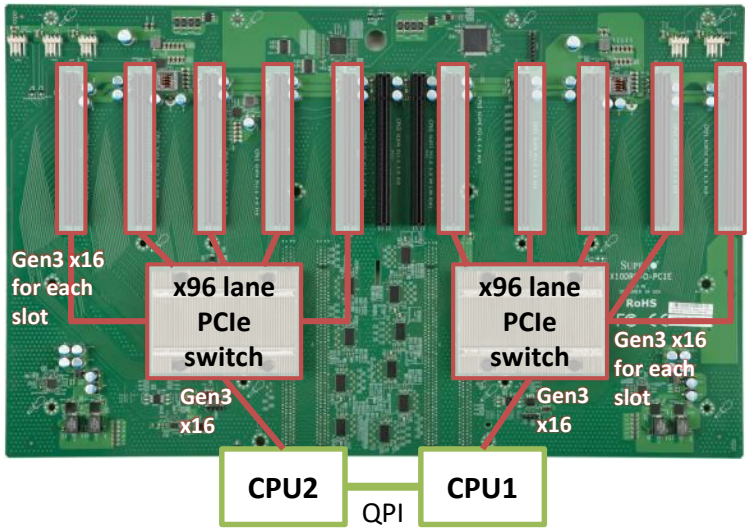
By P2P DMA over **PCIe-switch**, major data traffic bypass CPU



PCIe-bus level optimization (3/3)

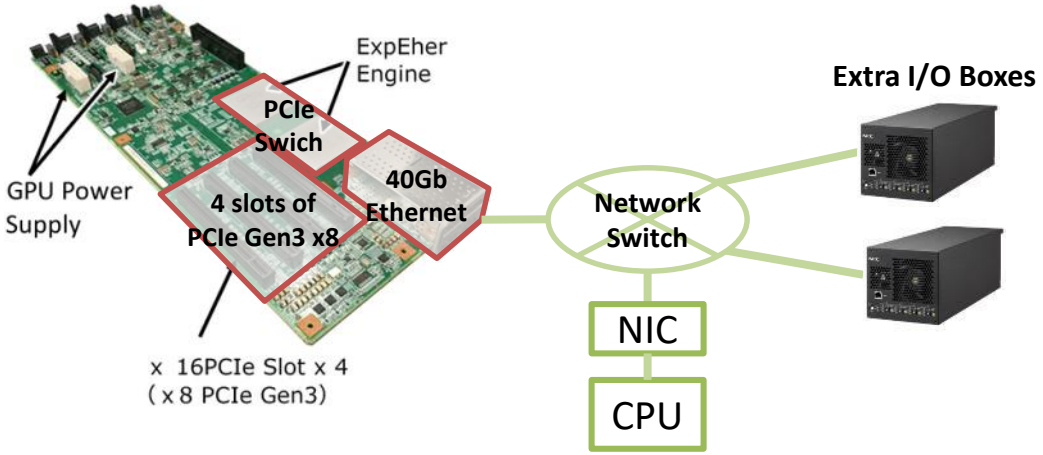
HPC Server – optimization for GPUDirect RDMA

Supermicro SYS-4029TRT2



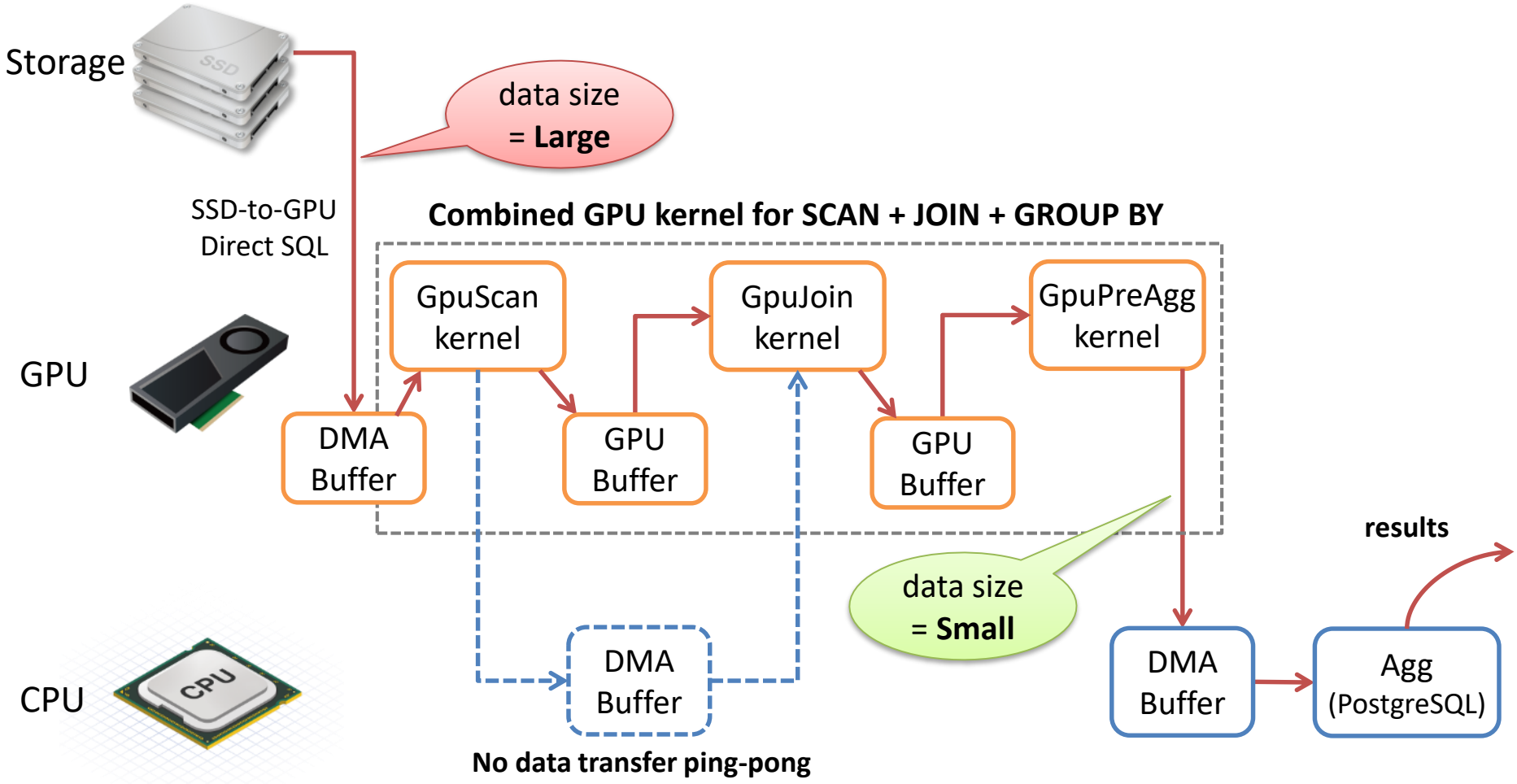
I/O Expansion Box

NEC ExpEther 40G (4slots edition)



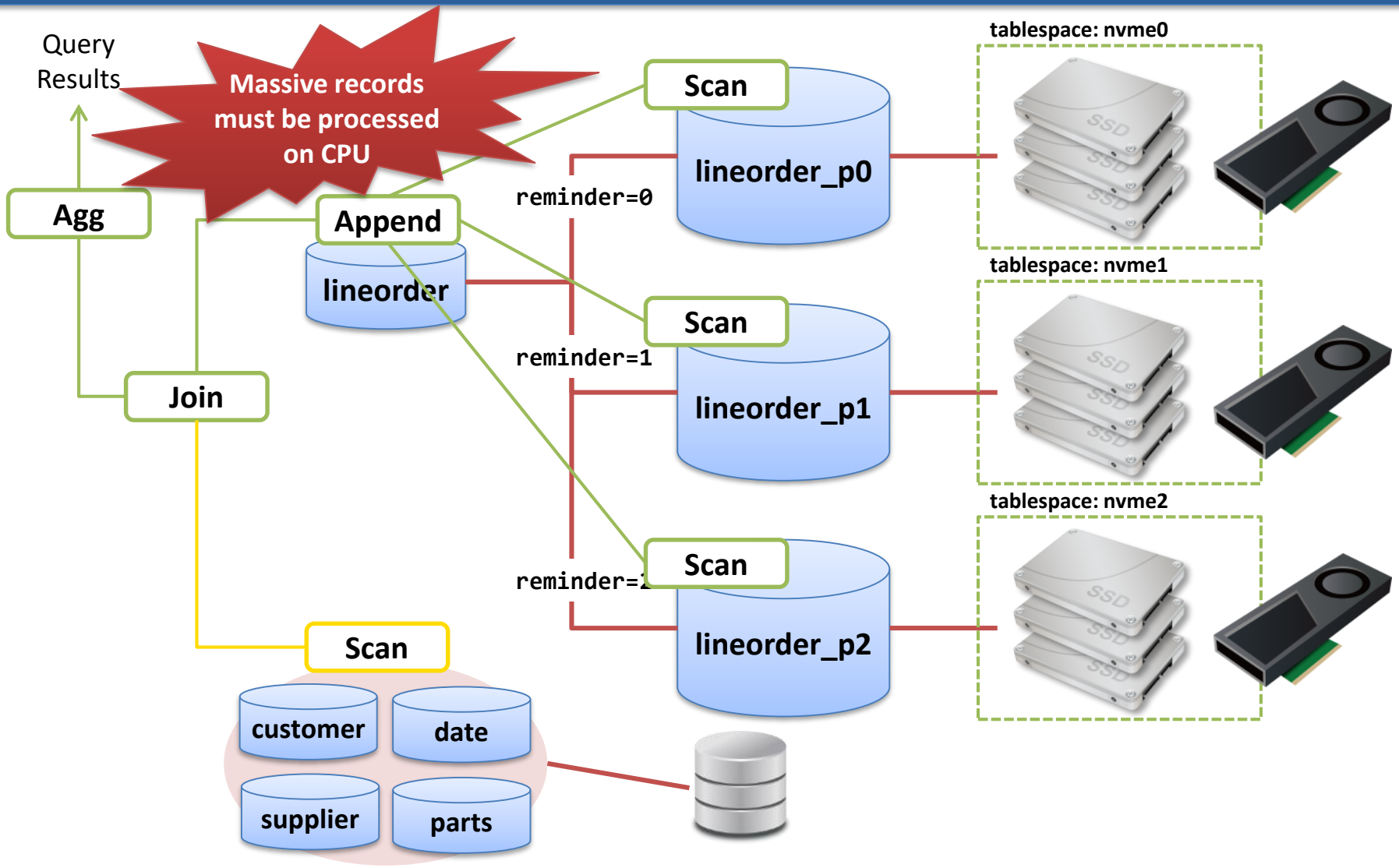
Special Optimization – Combined GPU Kernel

Reduction of “Data Ping-Pong” is a key of performance



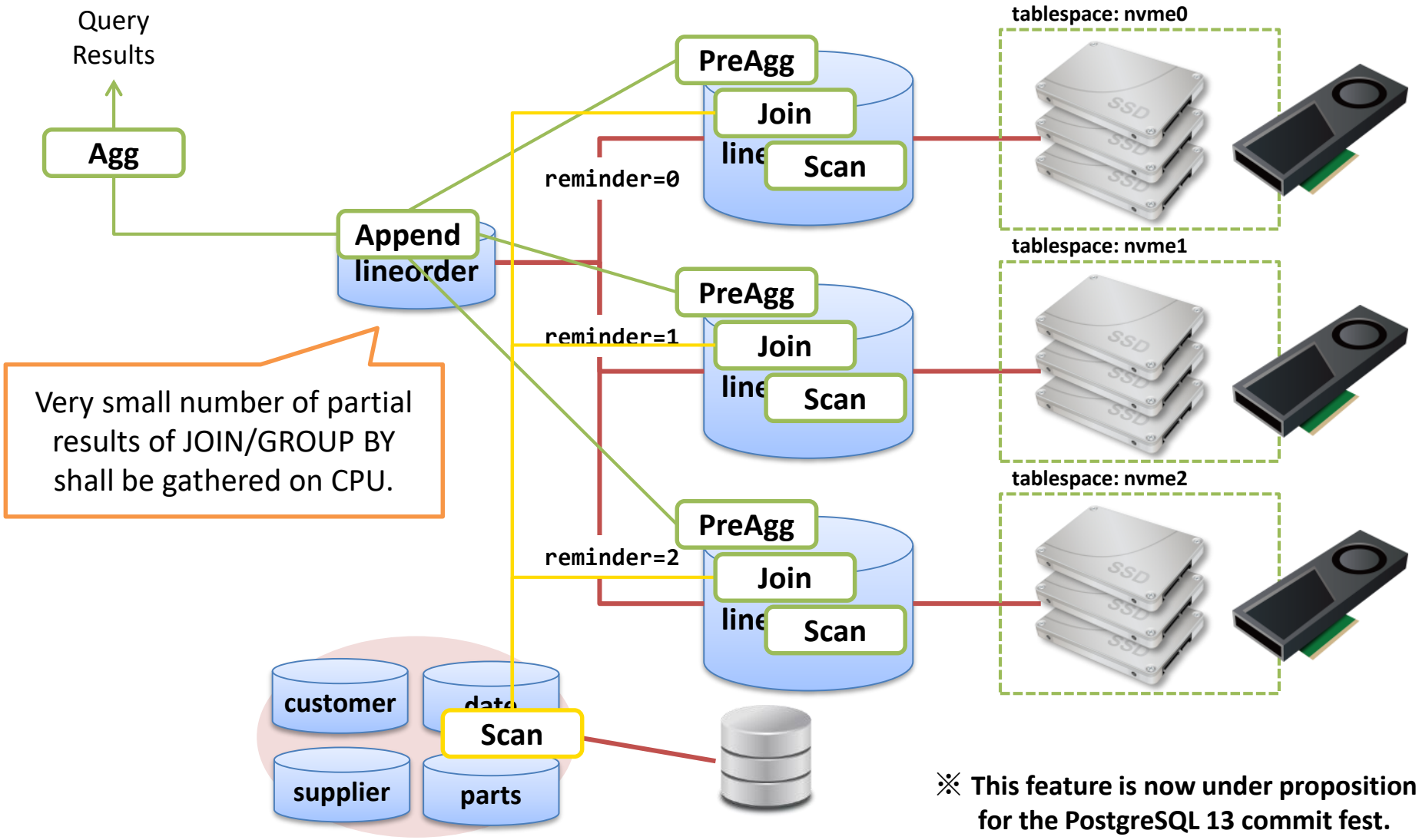
Asymmetric Partition-wise JOIN (1/5)

Records from partition-leafs must be backed to CPU and processed once!



Asymmetric Partition-wise JOIN (2/5)

Push down JOIN/GROUP BY, even if smaller half is not a partitioned table.



※ This feature is now under proposition for the PostgreSQL 13 commit fest.

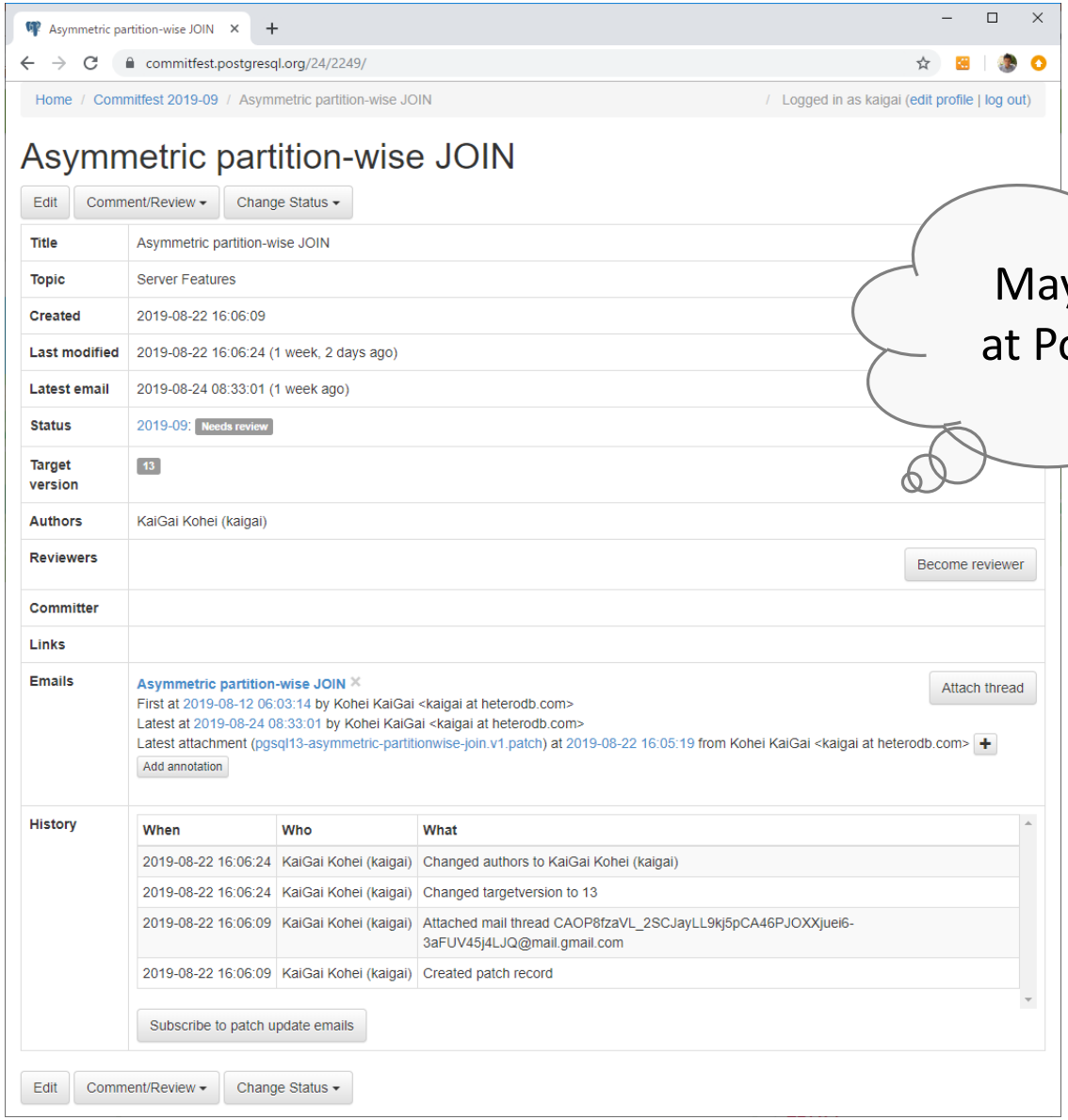
Asymmetric Partition-wise JOIN (3/5)

```
postgres=# explain select * from ptable p, t1 where p.a = t1.aid;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 Hash Join  (cost=2.12..24658.62 rows=49950 width=49)
   Hash Cond: (p.a = t1.aid)
   ->  Append  (cost=0.00..20407.00 rows=1000000 width=12)
         ->  Seq Scan on ptable_p0 p  (cost=0.00..5134.63 rows=333263 width=12)
         ->  Seq Scan on ptable_p1 p_1  (cost=0.00..5137.97 rows=333497 width=12)
         ->  Seq Scan on ptable_p2 p_2  (cost=0.00..5134.40 rows=333240 width=12)
   ->  Hash  (cost=1.50..1.50 rows=50 width=37)
         ->  Seq Scan on t1  (cost=0.00..1.50 rows=50 width=37)
(8 rows)
```

Asymmetric Partition-wise JOIN (4/5)

```
postgres=# set enable_partitionwise_join = on;
SET
postgres=# explain select * from ptable p, t1 where p.a = t1.aid;
                                      QUERY PLAN
-------------------------------------------------------------------------------
 Append  (cost=2.12..19912.62 rows=49950 width=49)
   ->  Hash Join  (cost=2.12..6552.96 rows=16647 width=49)
         Hash Cond: (p.a = t1.aid)
         ->  Seq Scan on ptable_p0 p  (cost=0.00..5134.63 rows=333263 width=12)
         ->  Hash  (cost=1.50..1.50 rows=50 width=37)
               ->  Seq Scan on t1  (cost=0.00..1.50 rows=50 width=37)
   ->  Hash Join  (cost=2.12..6557.29 rows=16658 width=49)
         Hash Cond: (p_1.a = t1.aid)
         ->  Seq Scan on ptable_p1 p_1  (cost=0.00..5137.97 rows=333497 width=12)
         ->  Hash  (cost=1.50..1.50 rows=50 width=37)
               ->  Seq Scan on t1  (cost=0.00..1.50 rows=50 width=37)
   ->  Hash Join  (cost=2.12..6552.62 rows=16645 width=49)
         Hash Cond: (p_2.a = t1.aid)
         ->  Seq Scan on ptable_p2 p_2  (cost=0.00..5134.40 rows=333240 width=12)
         ->  Hash  (cost=1.50..1.50 rows=50 width=37)
               ->  Seq Scan on t1  (cost=0.00..1.50 rows=50 width=37)
(16 rows)
```

Asymmetric Partition-wise JOIN (5/5)



May be available
at PostgreSQL v13

(Near) Future works & Conclusion

Benchmarks on HPC Server (1/2)

WIP

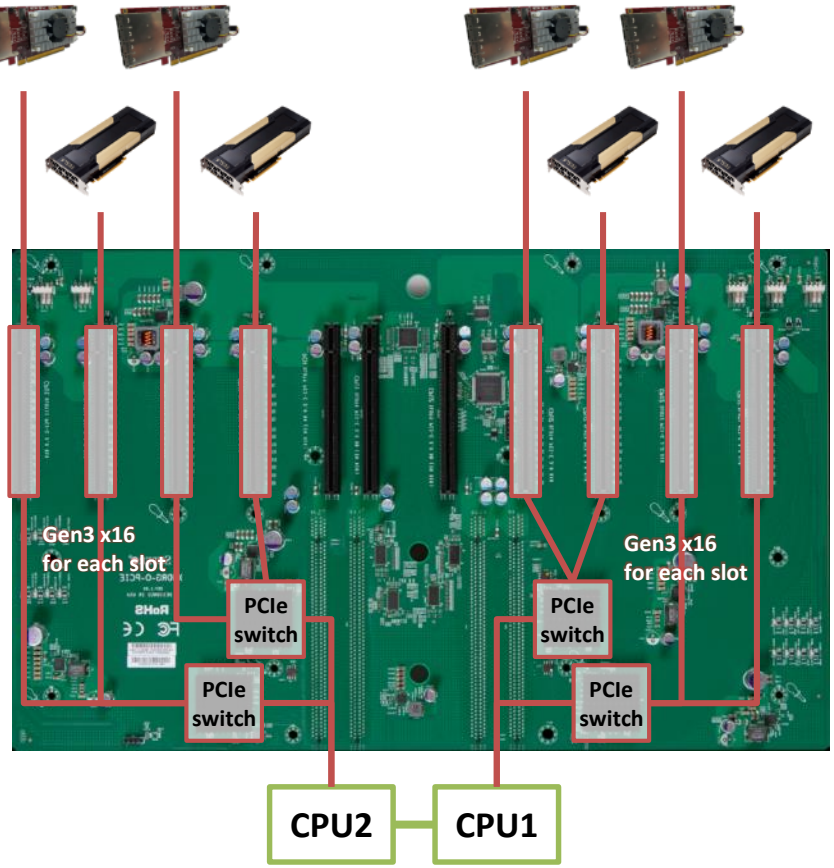
Intel SSD
DC P4510 (1.0TB) x16

SerialCables
dual PCI-ENC8G-08A
(U.2 NVME JBOF; 8slots) x2

via SFF-8644 based PCIe x4 cables
(3.2GB/s x 16 = max 51.2GB/s)

SerialCables
PCI-AD-x16HE x4

NVIDIA
Tesla V100 x4



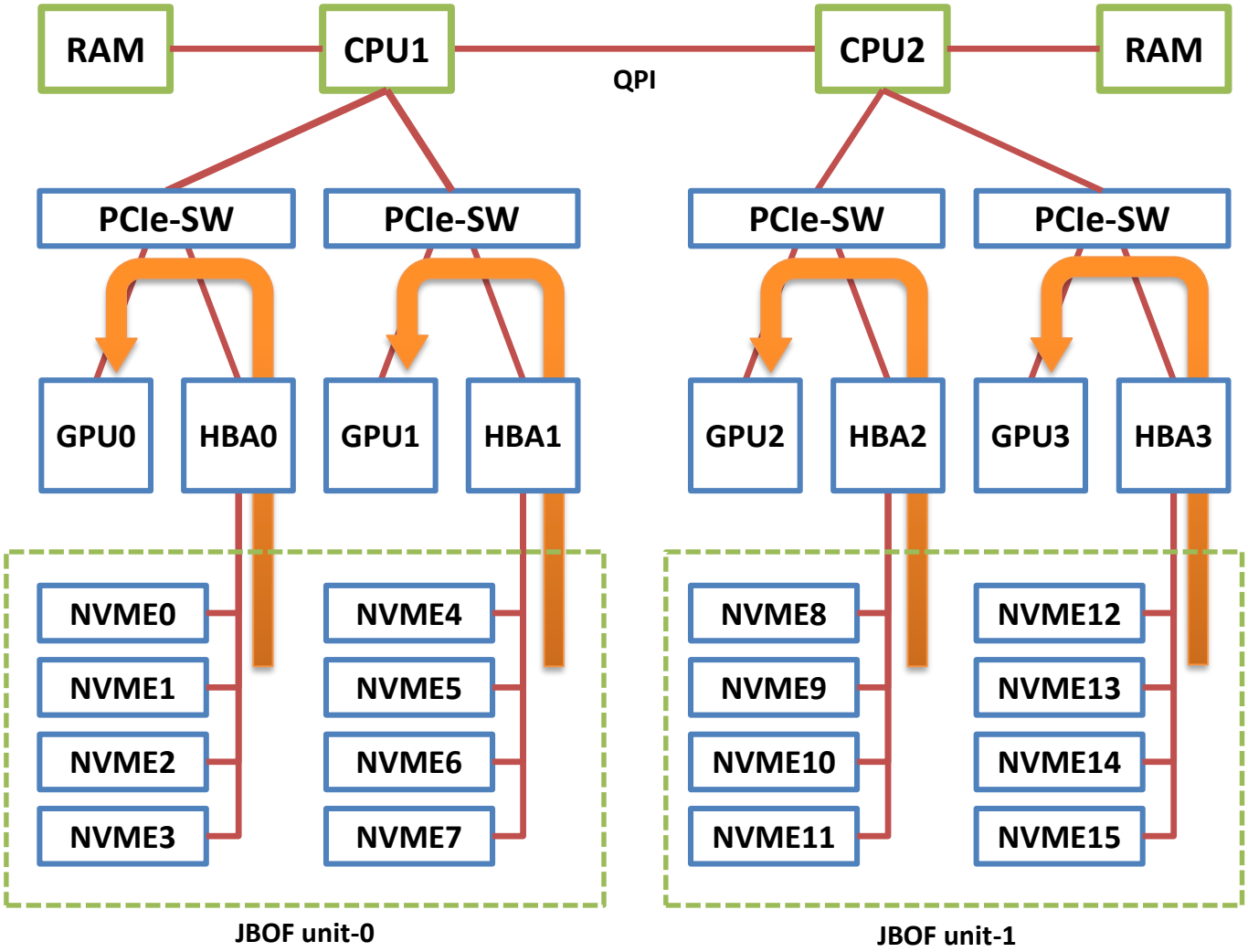
Now facilities are
under arrangement.
We shall have
benchmark project
at Sep/Oct-2019.

Supermicro
SYS-4029GP-TRT

Benchmarks on HPC Server (2/2)

WIP

Towards effective 100GB/s performance by PCIe bus optimization and columnar-storage



8.0 - 10GB/s physical data transfer performance per (GPU + 4 SSD) unit

By columnar storage, **25 - 30GB/s** effective data transfer per unit

By 4 units parallel execution, **100 - 120GB/s** effective data processing capability

Capable to process **up to 100TB grade** log-data in single node

Future works

PG-Strom development team welcomes your volunteership!

- Asymmetric Partition-wise JOIN (PostgreSQL v13)
- MVCC checks on GPU device
- Data-frame exchange over GPU device memory (NVIDIA RAPIDS)
- RHEL8 support (Linux kernel driver)
- DRAM access reductions on GpuJoin
- More reliable statistics framework
- Enlargement of regression test cases
- ……and so on

Conclusion

Full utilization of H/W, optimized data structure, and proper partitioning enable terabytes scale data-processing on a standalone PostgreSQL system.

Characteristics of Log-data

- INSERT-only
- Must be imported once
- Always have timestamp

Weapons we can use

- SSD-to-GPU Direct SQL
- Arrow_Fdw
- PostgreSQL Partitioning
- PCIe-bug level optimization

Why PostgreSQL?

- Standalone system is much simple and inexpensive than cluster.
- Engineers have been familiar with PostgreSQL over 10 years.

Resources

Repository

- https://github.com/heterodb/pg-strom
- https://heterodb.github.io/swdc/

Documents

- http://heterodb.github.io/pg-strom/

Contact

- kaigai@heterodb.com
- Tw: @kkaigai


HeteroDB