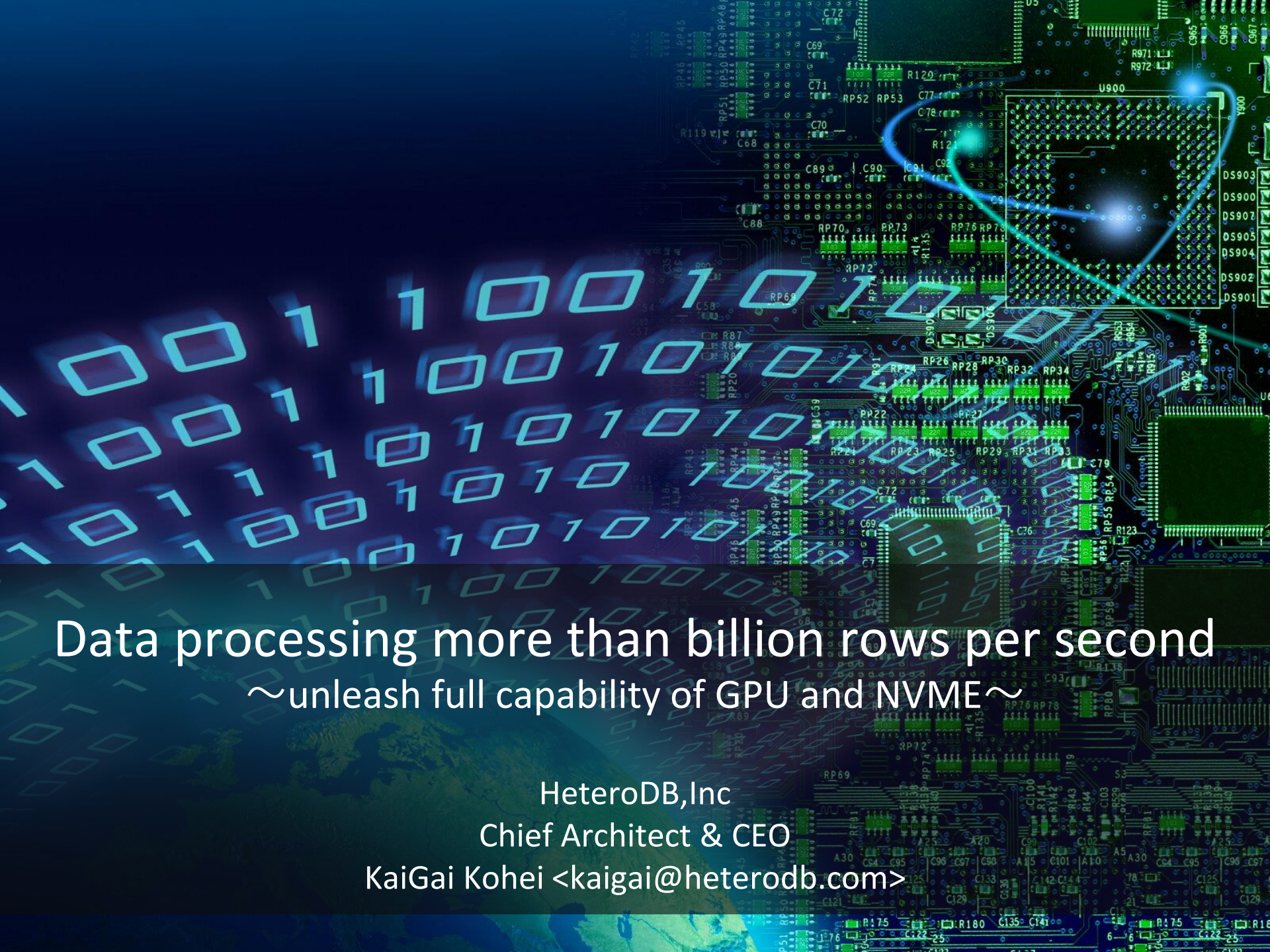


Data processing more than billion rows per second

～unleash full capability of GPU and NVME～

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

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

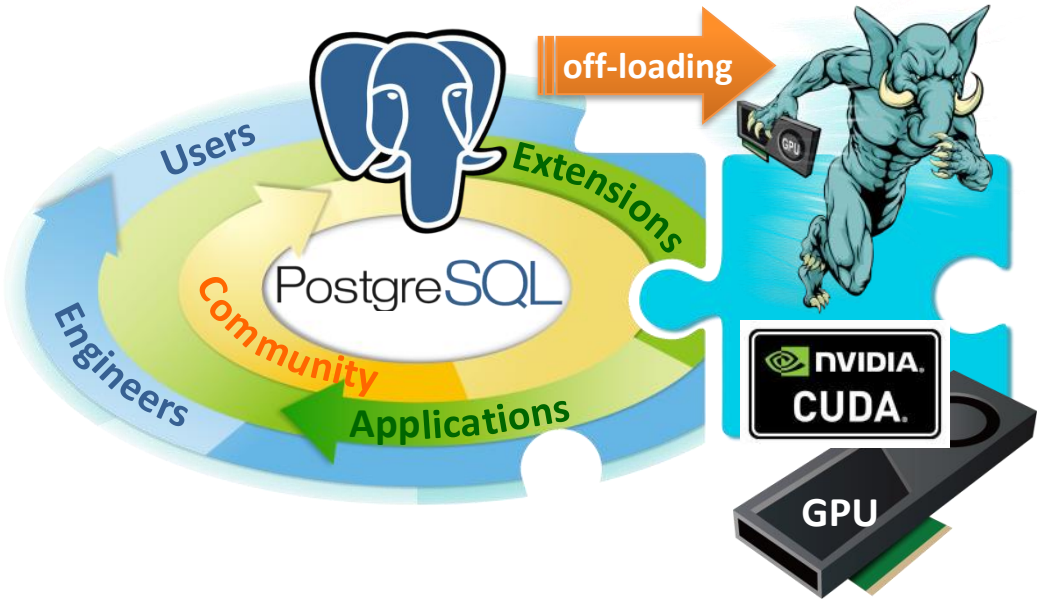
- Established: 4th-Jul-2017
- Location: Shinagawa, Tokyo, Japan
- Businesses:
  - ✓ Sales & development of high-performance data-processing software on top of heterogeneous architecture.
  - ✓ Technology consulting service on GPU&DB area.

NVIDIA INCEPTION PROGRAM

PG-Strom

What is PG-Strom?

PG-Strom is an extension of PostgreSQL for terabytes scale data-processing and in-database analytics, by utilization of GPU and NVME-SSD.



for IoT/Big-Data

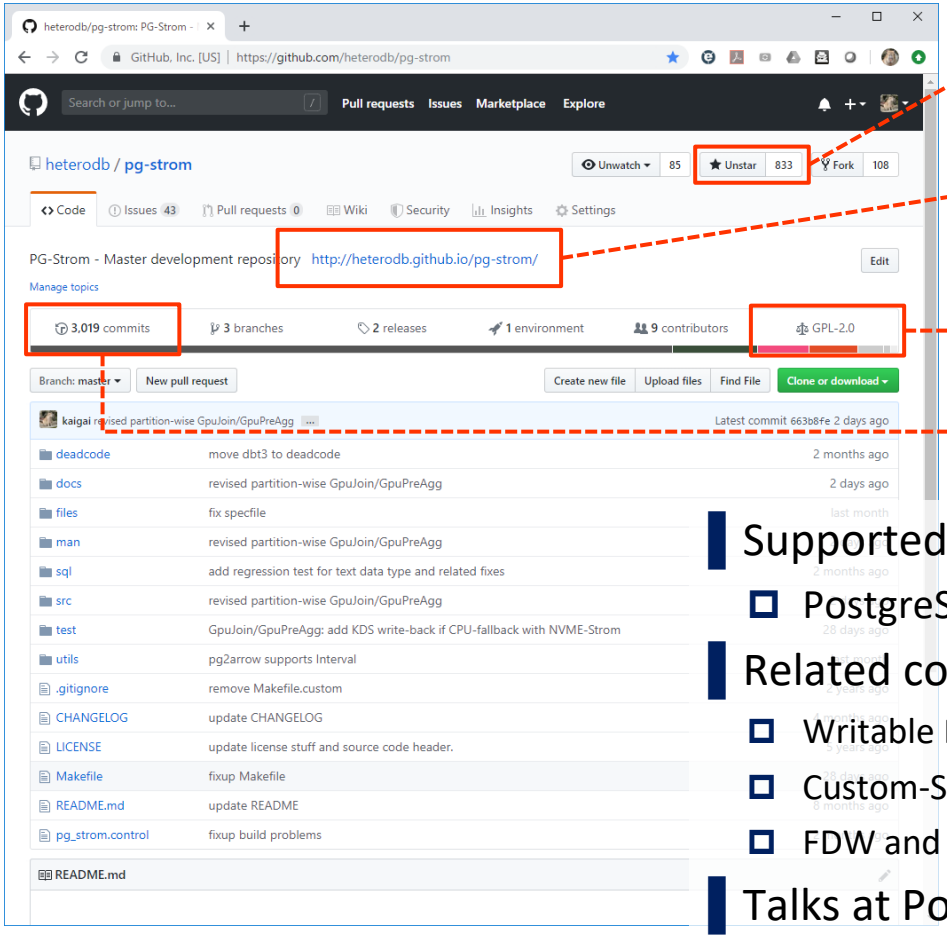
- SSD-to-GPU Direct SQL
- Columnar Store (Arrow_Fdw)
- Asymmetric Partition-wise JOIN/GROUP BY
- BRIN-Index Support
- NVME-over-Fabric Support
- PostGIS + GiST Support (WIP)

for ML/Analytics

- GPU Memory Store(WIP)
- Procedural Language for GPU native code.
- IPC on GPU device memory over cuPy data frame.

- 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 v12, v11, and v10. WIP for v13 support

Related contributions to PostgreSQL

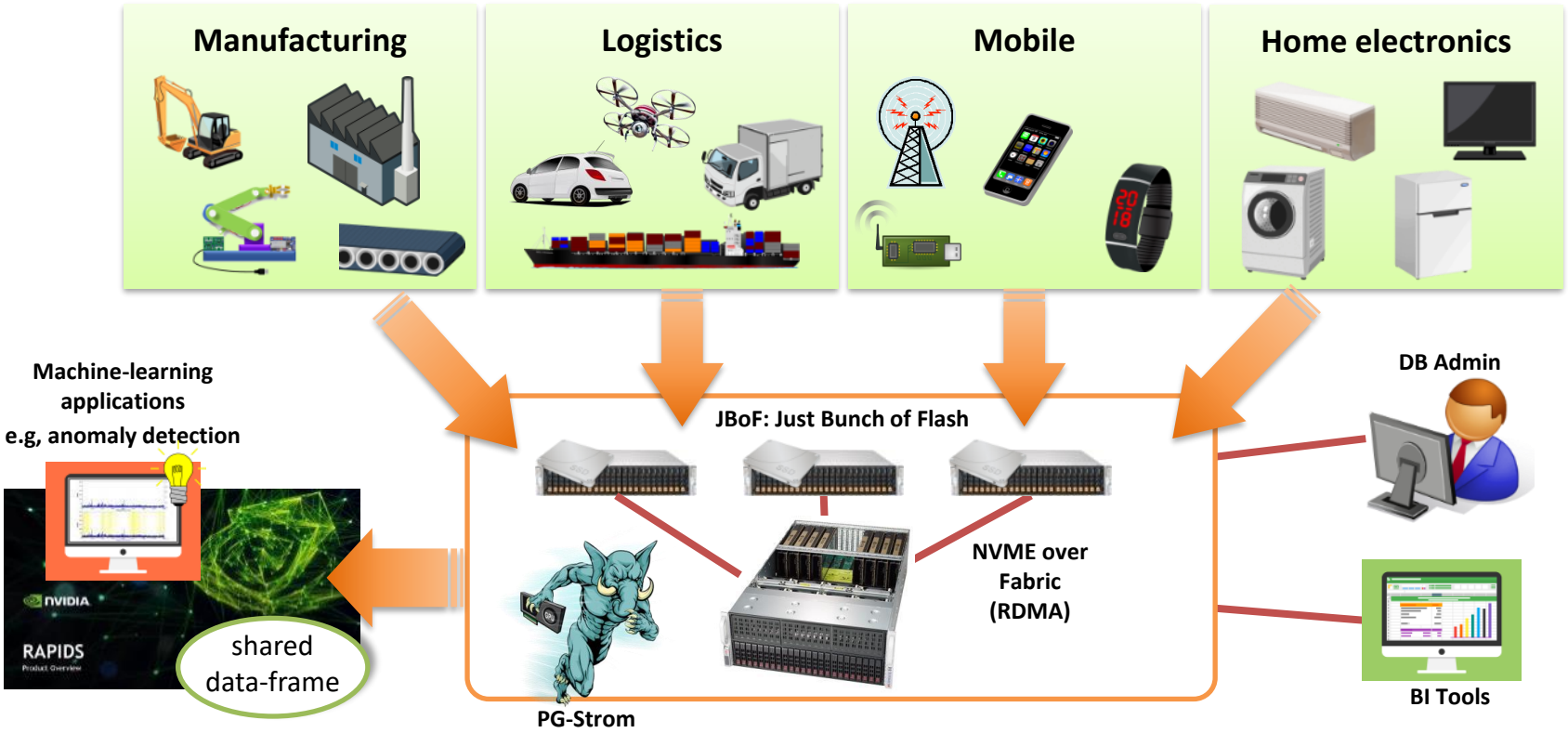
- Writable FDW support (9.3)
- Custom-Scan interface (9.5)
- FDW and Custom-Scan JOIN pushdown (9.5)

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)

Target Workloads?

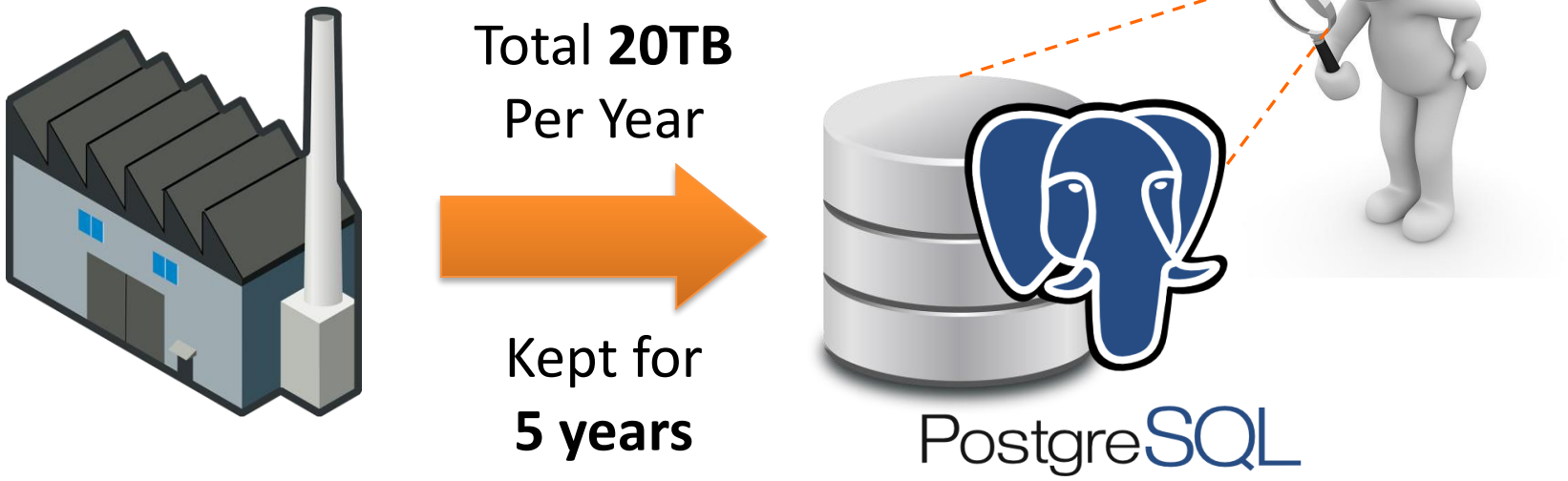
Target: IoT/M2M Log-data processing



- 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.

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

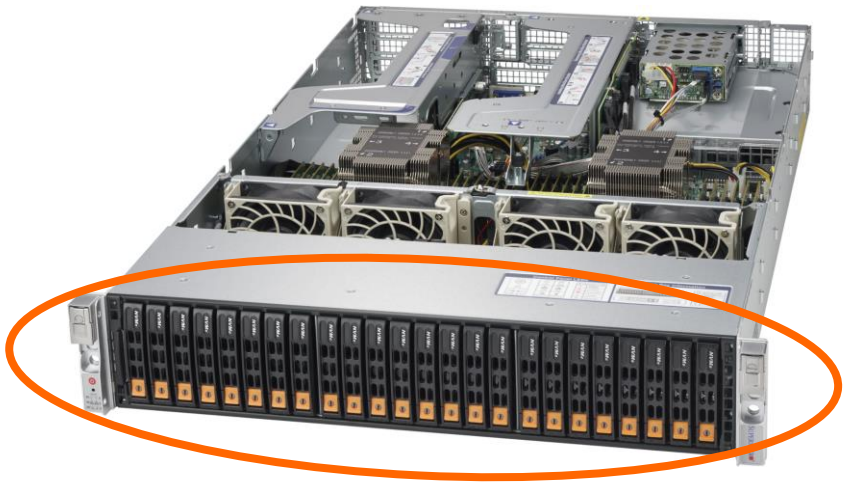
Case: A semiconductor factory managed up to 100TB with PostgreSQL



- 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

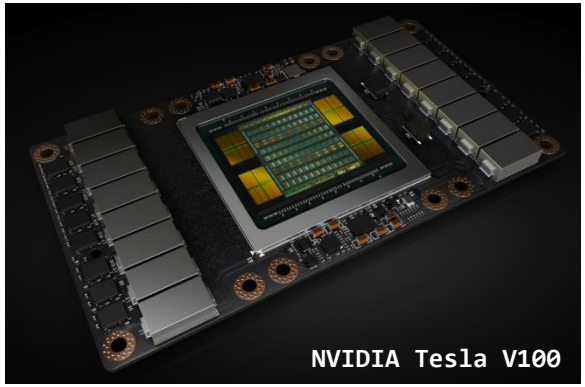
Efficient Storage	• **SSD-to-GPU Direct SQL** 
Efficient Data Structure	• **Arrow_Fdw** 
Efficient Data Deployment	• **Table Partitioning** 

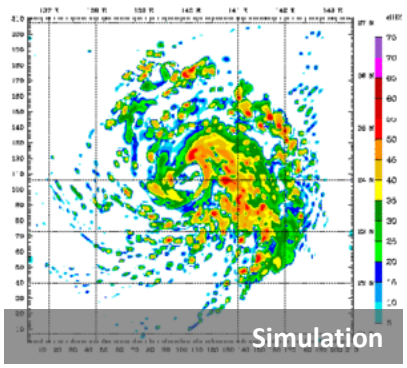
Efficient Storage

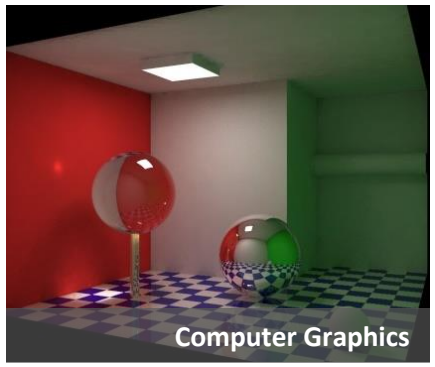
PG-Strom: SSD-to-GPU Direct SQL

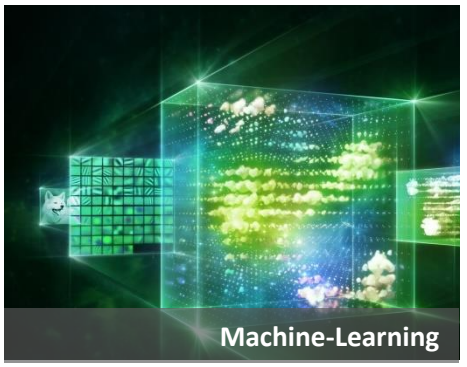
GPU's characteristics - mostly as a computing accelerator

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



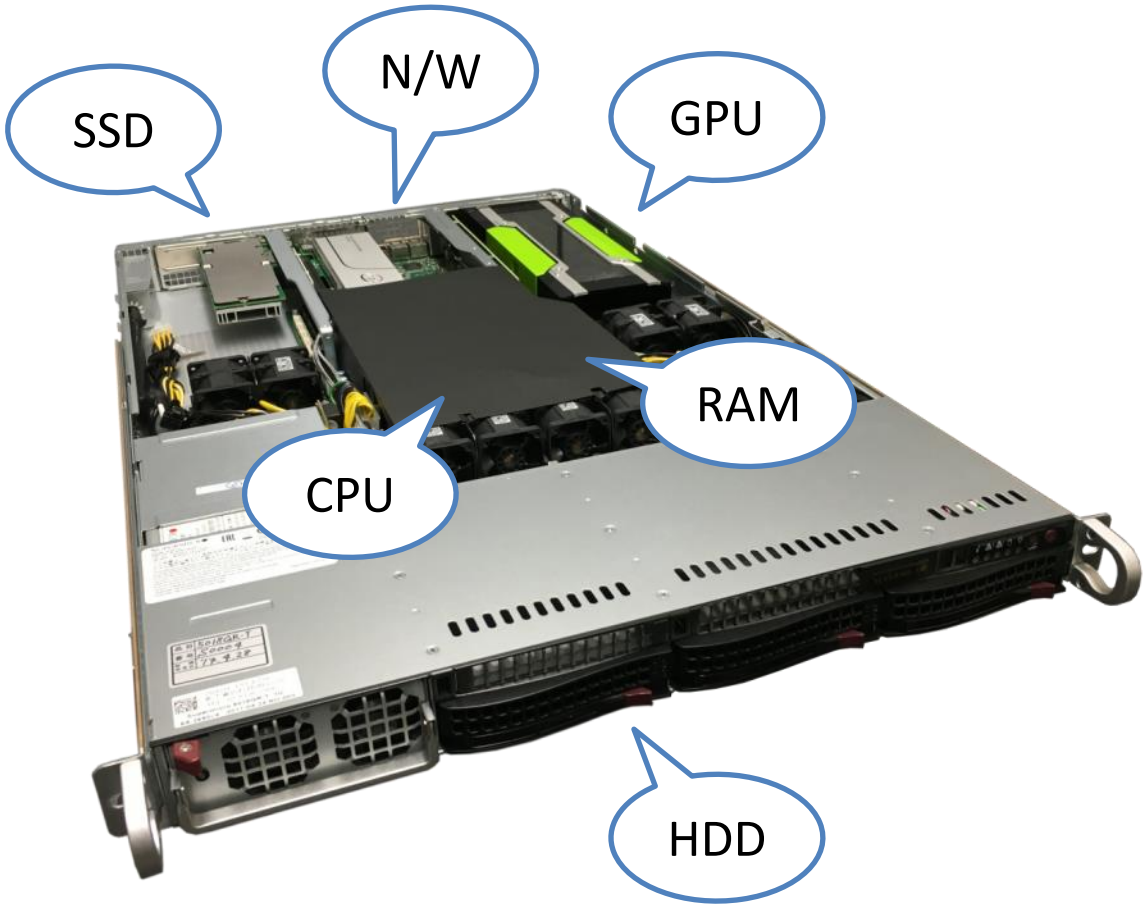






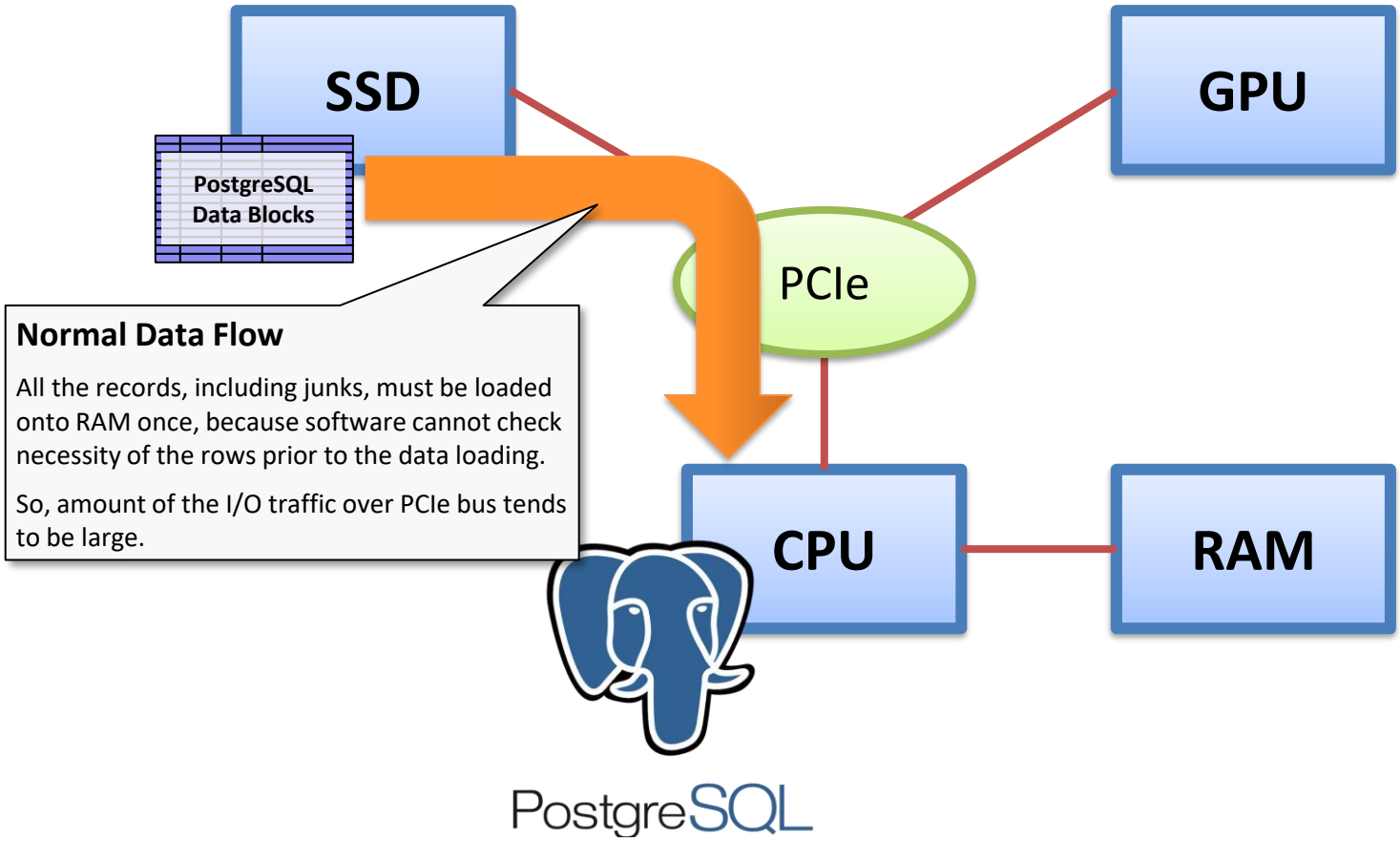
How I/O workloads are accelerated by GPU that is a **computing accelerator**?

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.

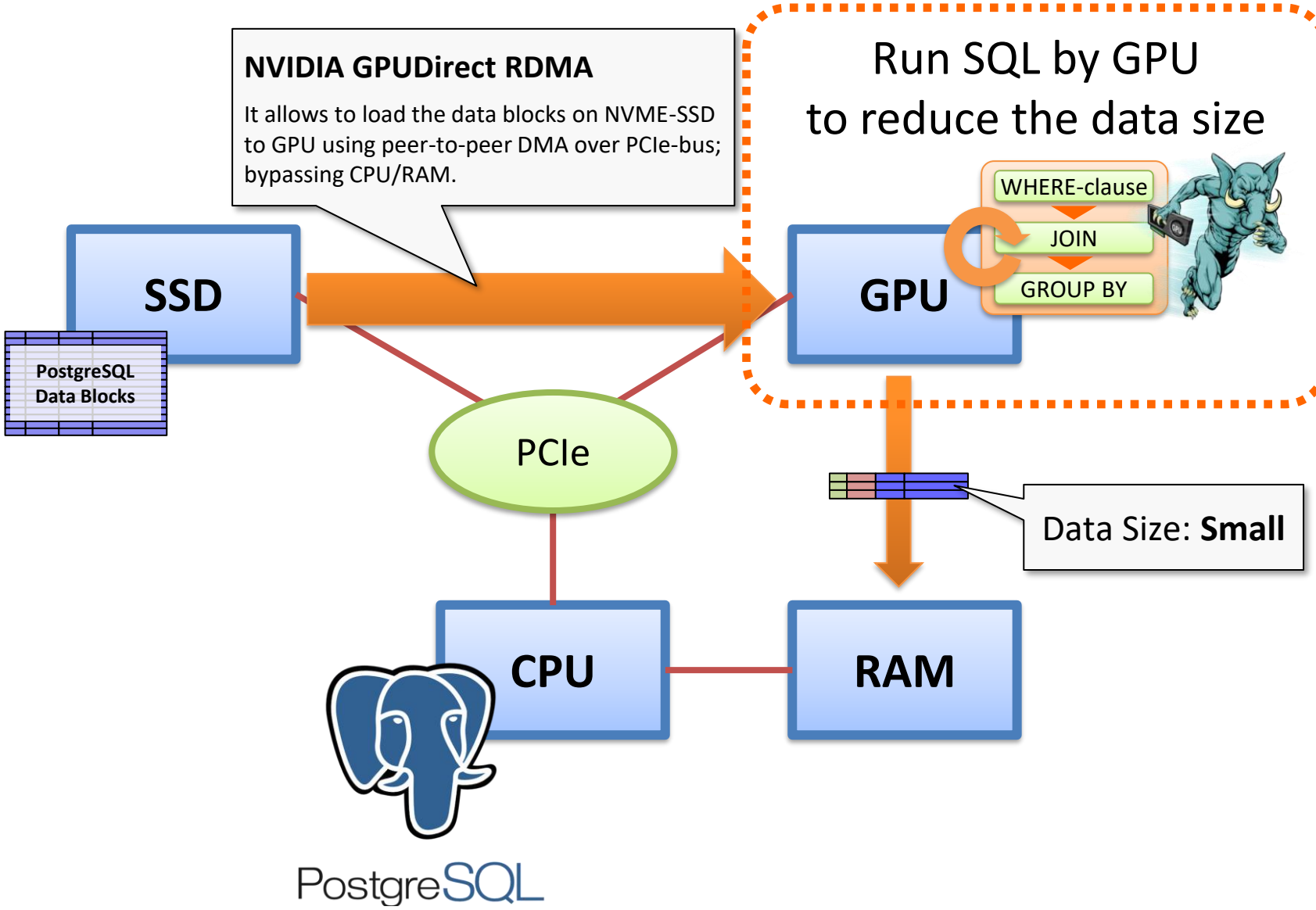


Normal Data Flow

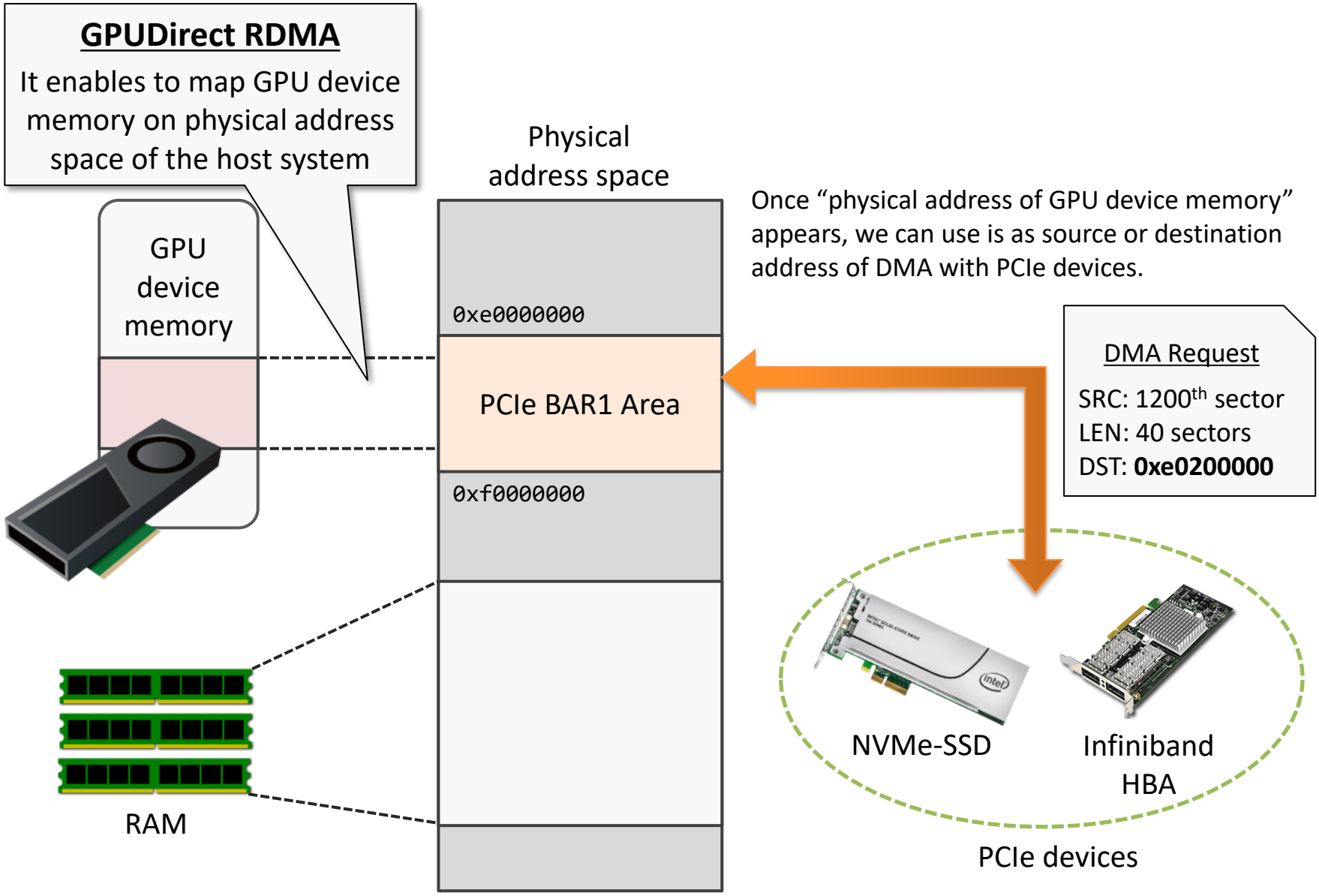
All the records, including junks, must be loaded onto RAM once, because software cannot check necessity of the rows prior to the data loading.

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

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



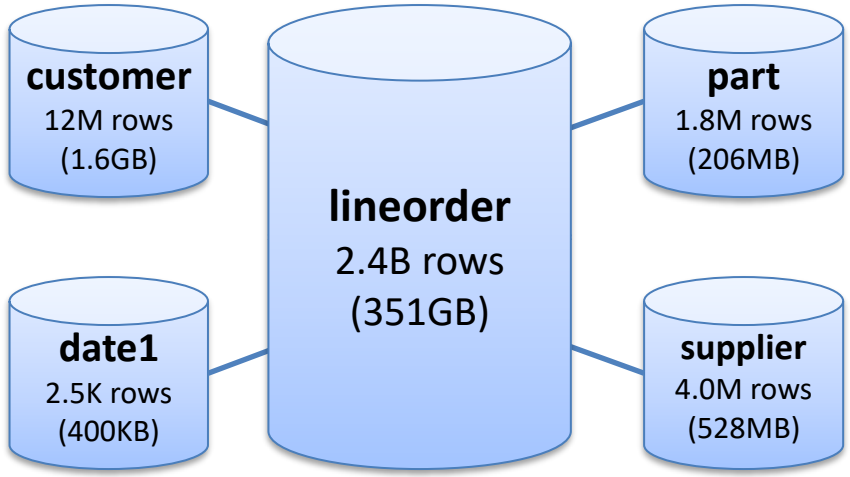
Background - GPUDirect RDMA by NVIDIA



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

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

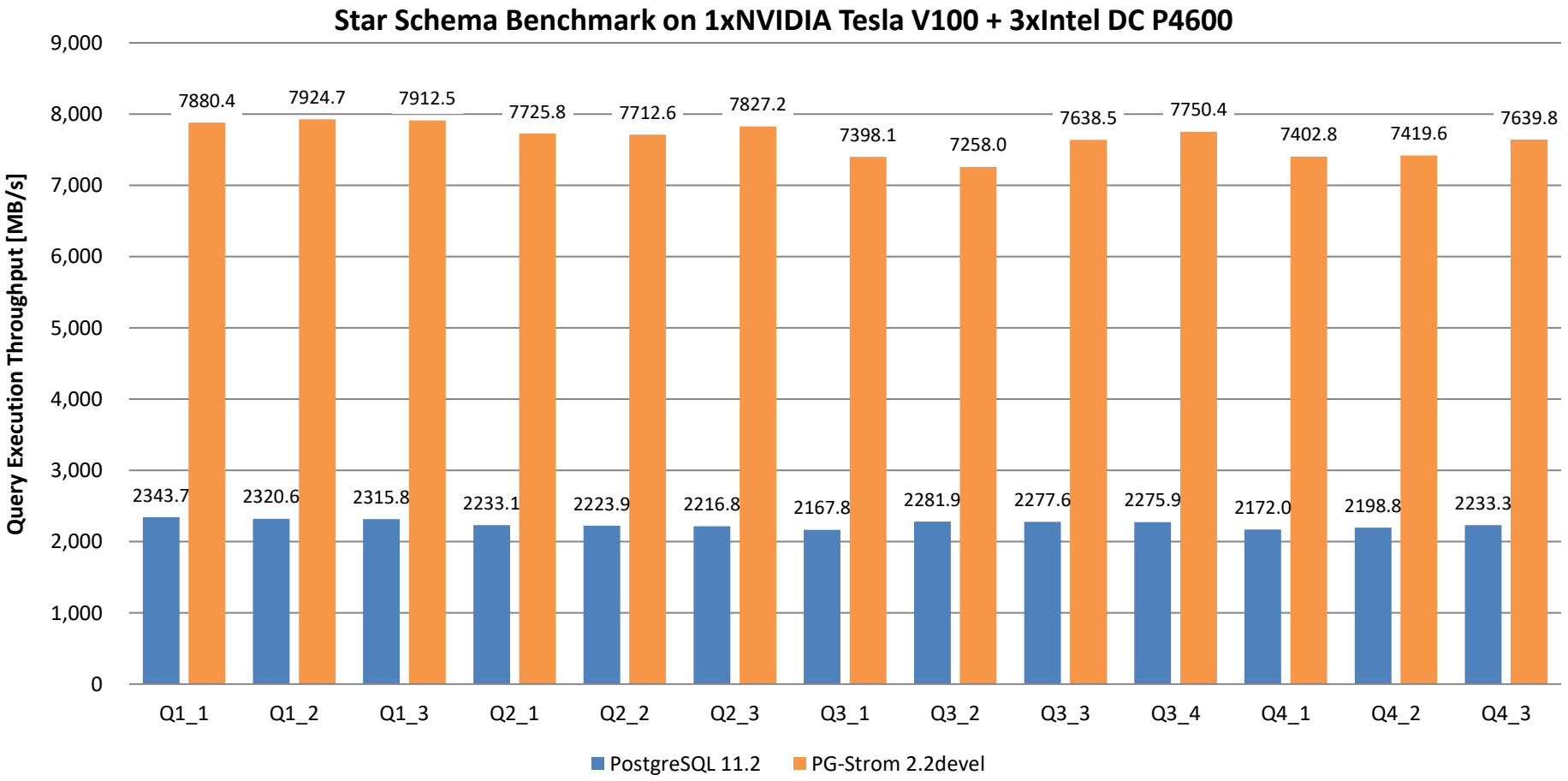
Supermicro SYS-1019GP-TT	
CPU	Xeon Gold 6126T (2.6GHz, 12C) x1
RAM	192GB (32GB DDR4-2666 x 6)
GPU	NVIDIA Tesla V100 (5120C, 16GB) x1
SSD	Intel SSD DC P4600 (HHHL; 2.0TB) x3 (striping configuration by md-raid0)
HDD	2.0TB(SATA; 72krpm) x6
Network	10Gb Ethernet 2ports
OS	Red Hat Enterprise Linux 7.6 CUDA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



■ Query Example (Q2_3)

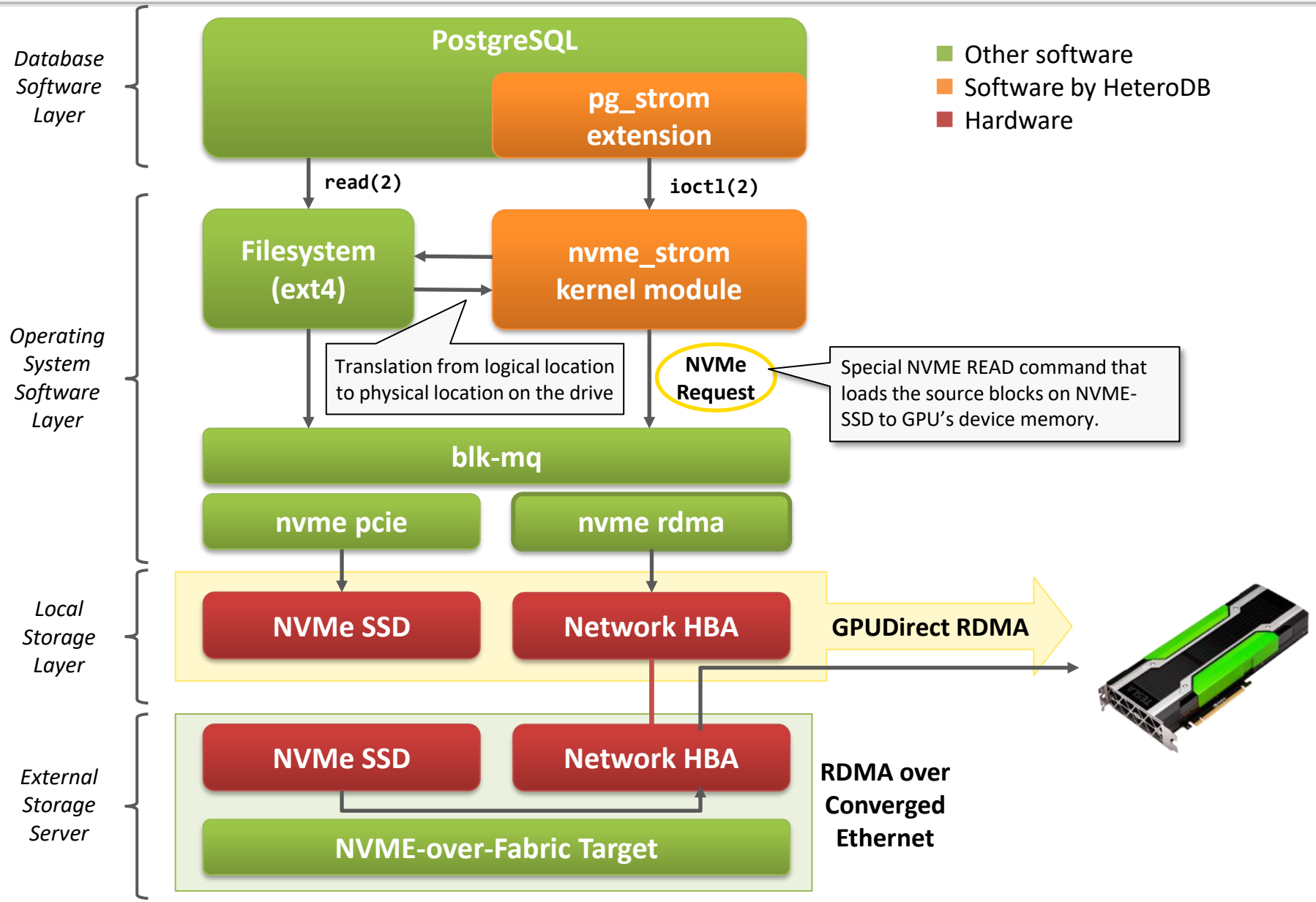
```
SELECT sum(lo_revenue), d_year, p_brand1
  FROM lineorder, date1, part, supplier
 WHERE lo_orderdate = d_datekey
   AND lo_partkey = p_partkey
   AND lo_suppkey = s_suppkey
   AND p_category = 'MFGR#12‘
   AND s_region = 'AMERICA‘
 GROUP BY d_year, p_brand1
 ORDER BY d_year, p_brand1;
```

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 Strucute

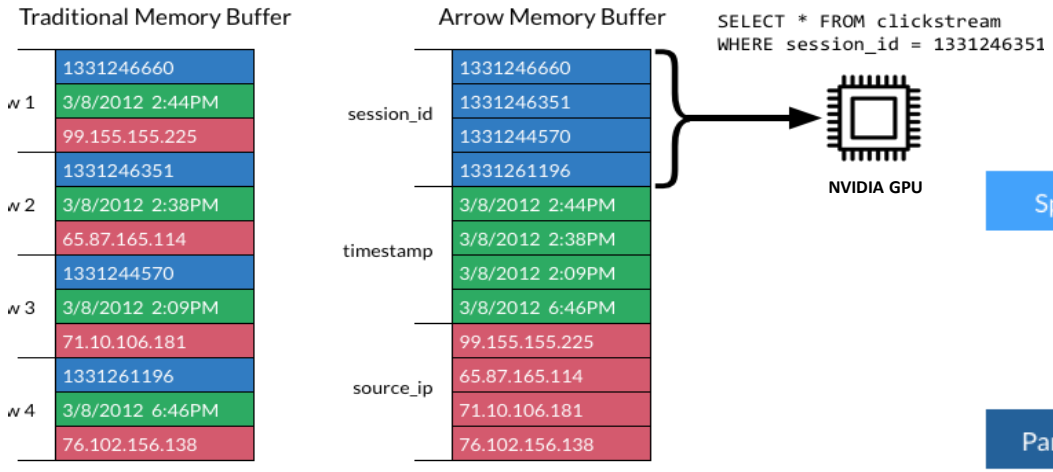
PG-Strom: Arrow_Fdw (columnar store)

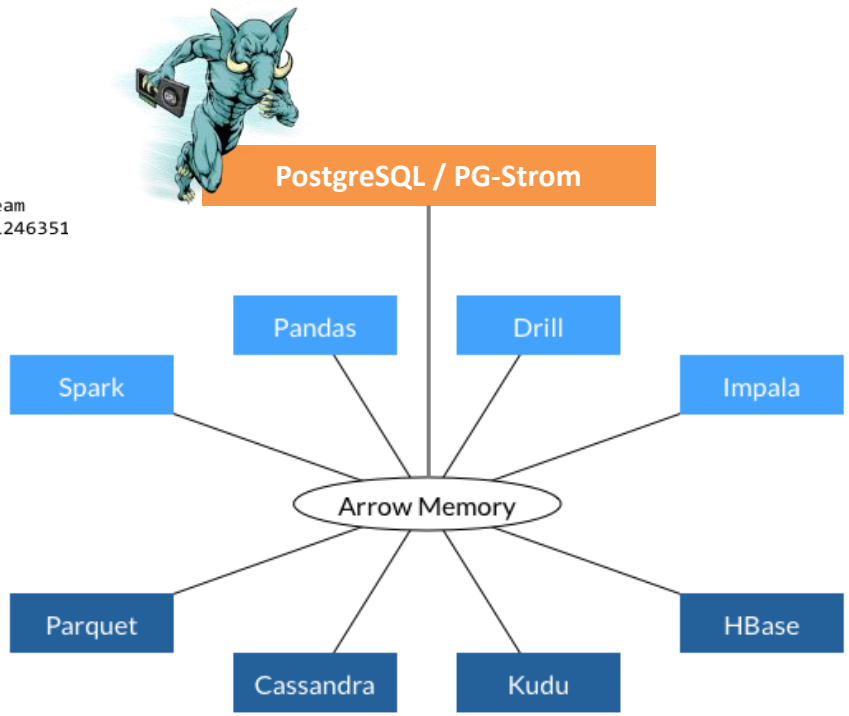
Background: Apache Arrow (1/2)

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





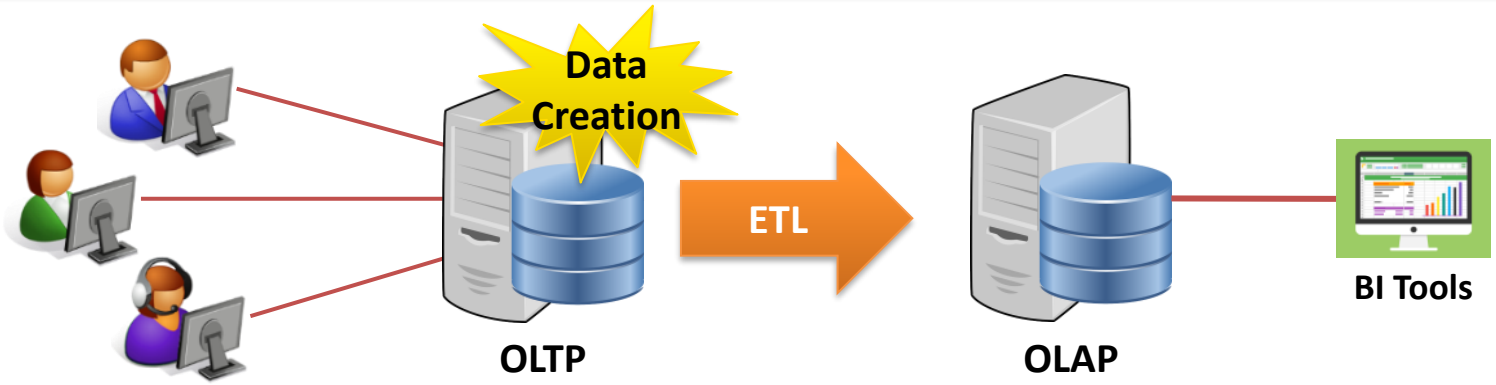
Background: Apache Arrow (2/2)

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

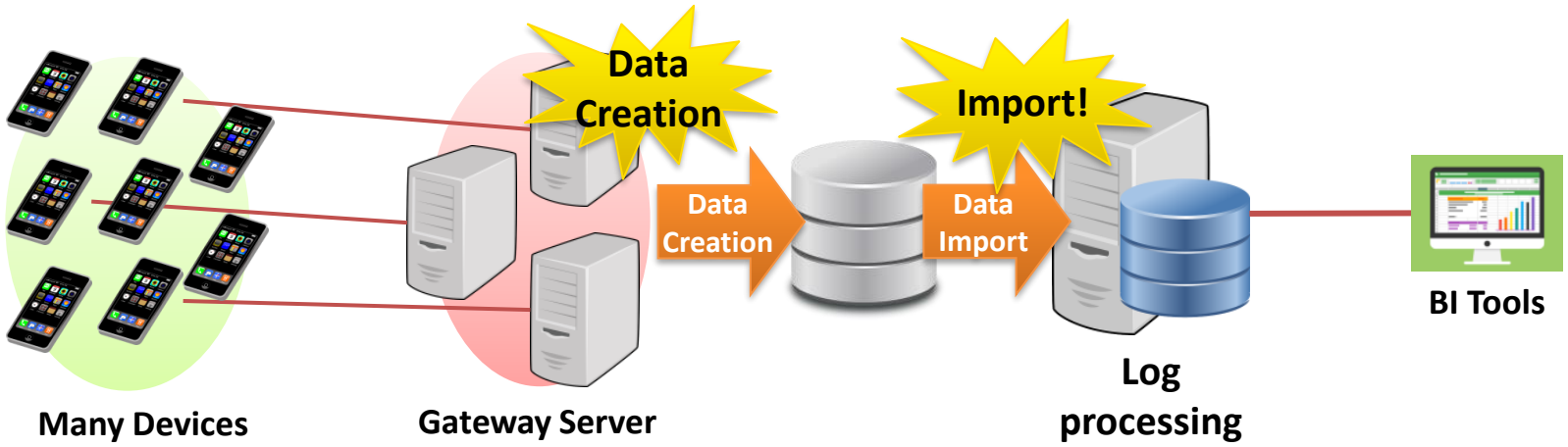
Apache Arrow Data Types	PostgreSQL Data Types	extra description
Int	int2, int4, int8	
FloatingPoint	float2, float4, float8	float2 is an enhancement by PG-Strom
Binary	bytea	
Utf8	text	
Bool	bool	
Decimal	numeric	Decimal = fixed-point 128bit value
Date	date	adjusted to unitsz = Day
Time	time	adjusted 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

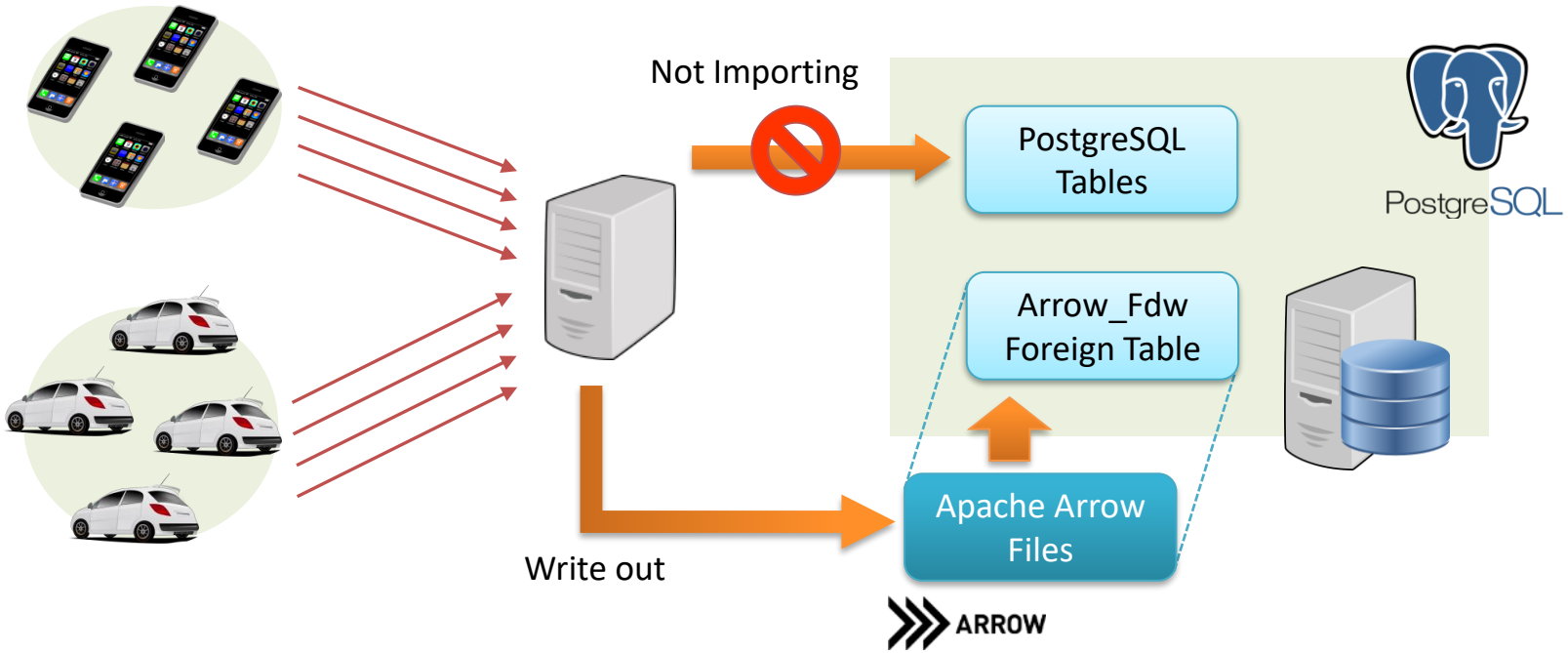


IoT/M2M use case – Data is generated **outside** of database system



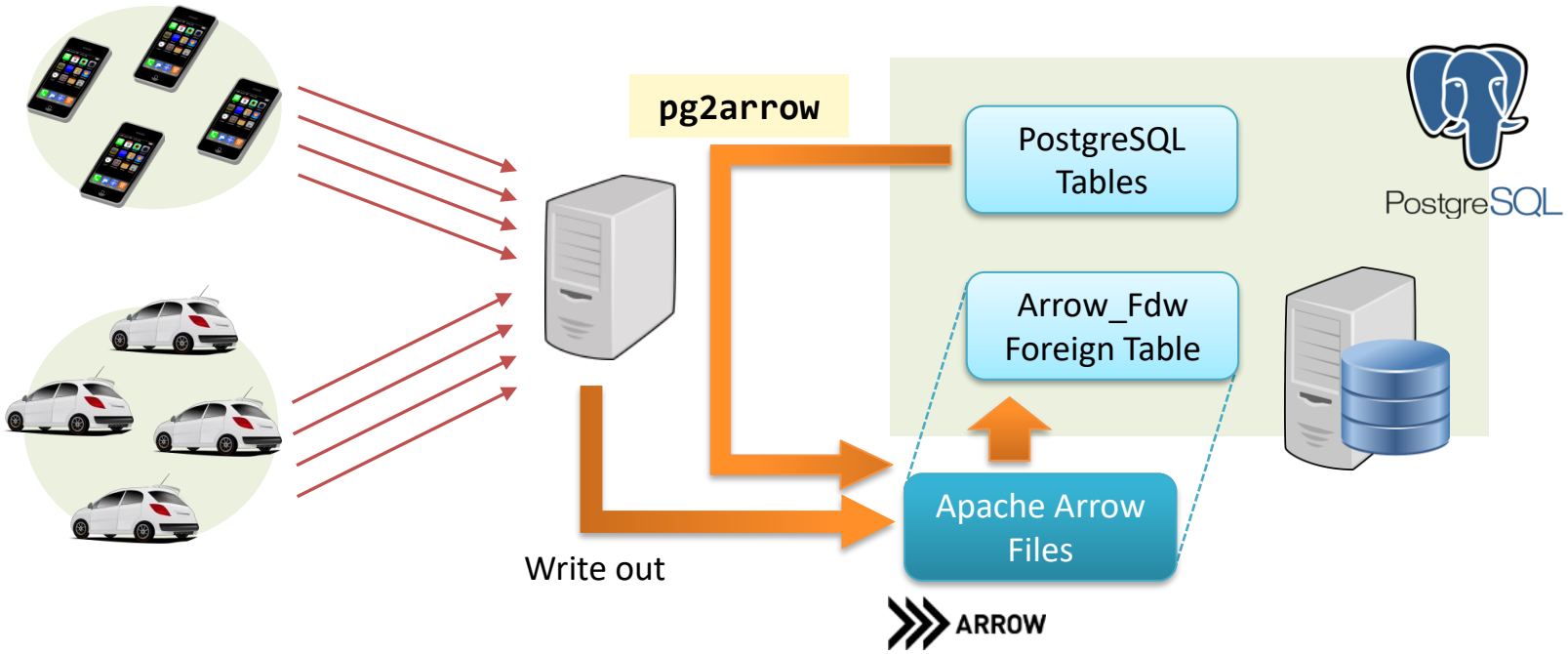
Data Importing becomes a heavy time-consuming operations for big-data processing.

Arrow_Fdw - maps Apache Arrow files as a foreign table



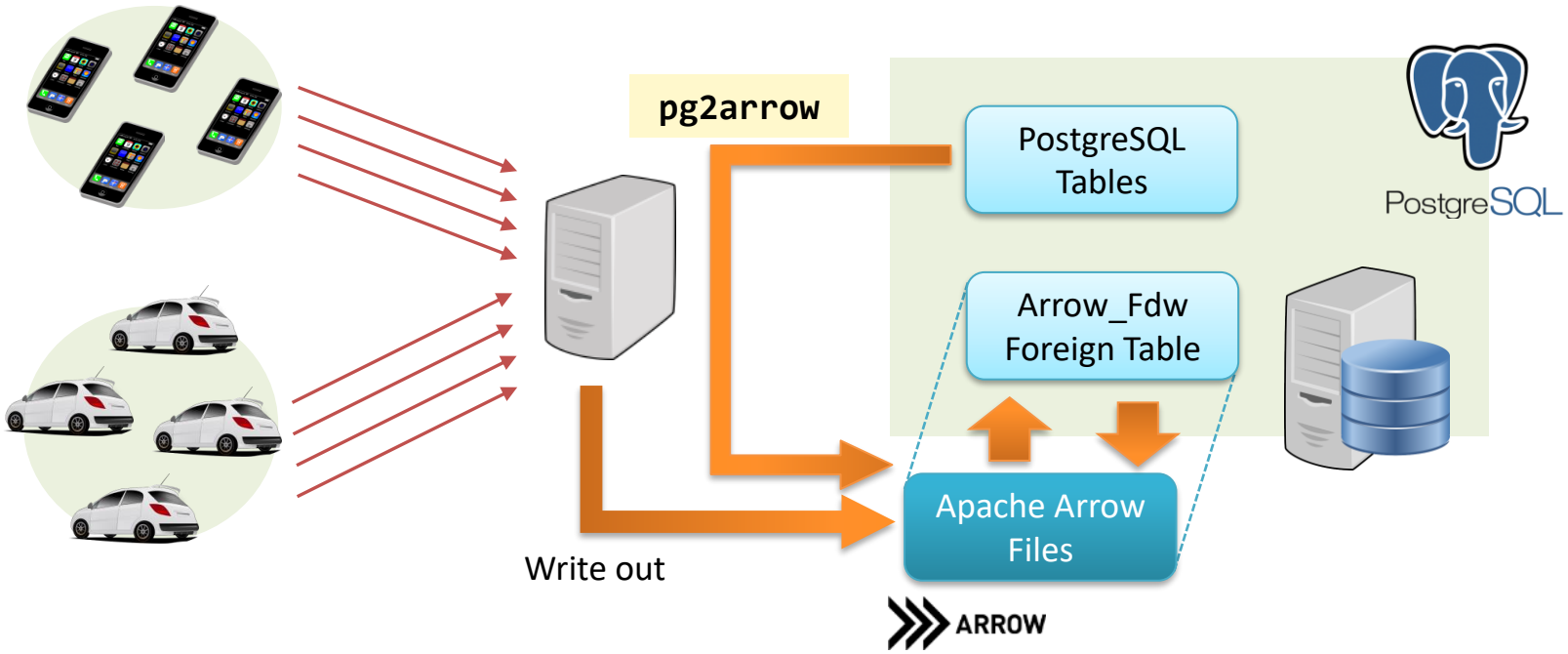
- Arrow_Fdw enables to map Apache Arrow files as a PostgreSQL foreign table.
- Not Importing, so Apache Arrow files are immediately visible.

Arrow_Fdw - maps Apache Arrow files as a foreign table



- Arrow_Fdw enables to map Apache Arrow files as a PostgreSQL foreign table.
- Not Importing, so Apache Arrow files are immediately visible.
- **`pg2arrow`** generates Apache Arrow file from query results of PostgreSQL.

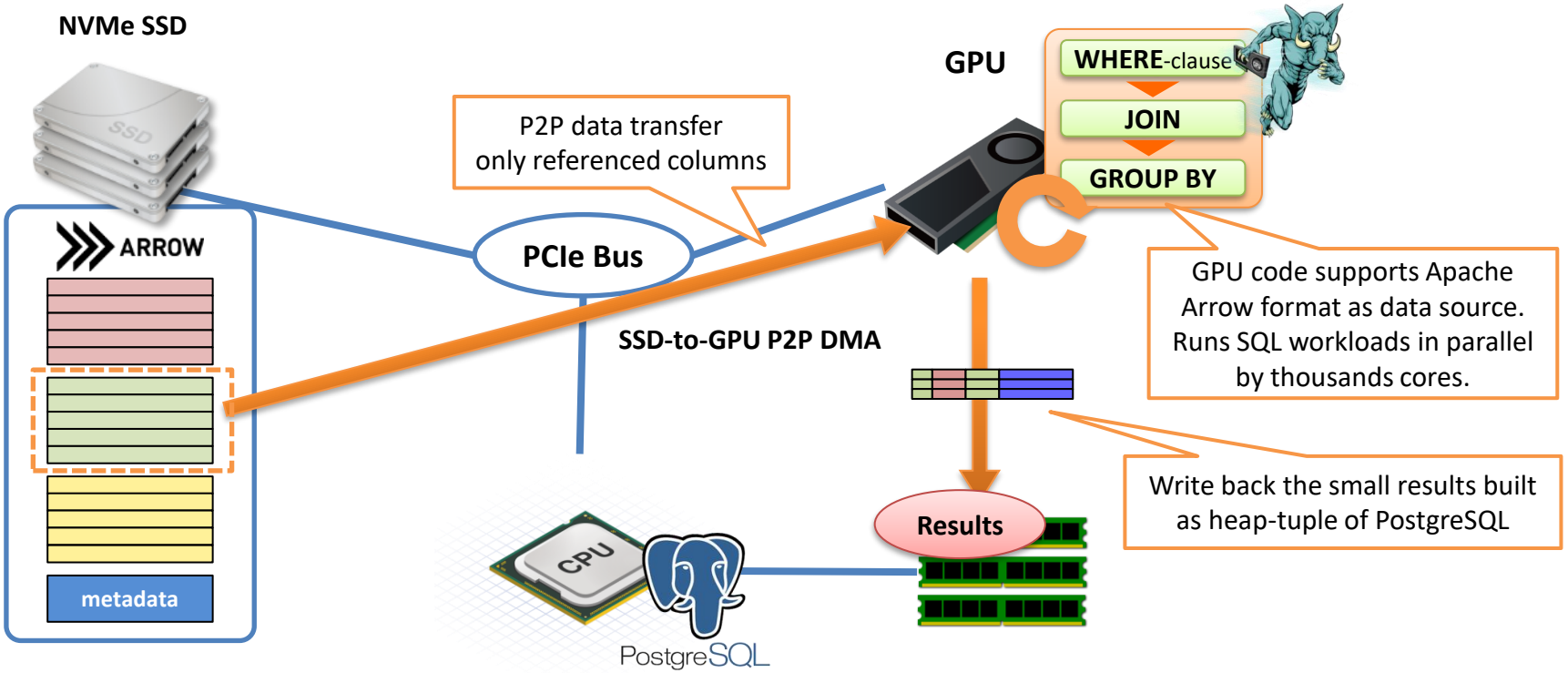
Arrow_Fdw - maps Apache Arrow files as a foreign table



- Arrow_Fdw enables to map Apache Arrow files as a PostgreSQL foreign table.
- Not Importing, so Apache Arrow files are immediately visible.
- **pg2arrow** generates Apache Arrow file from query results of PostgreSQL.
- Arrow_Fdw can be writable, but only batched-INSERT is supported.

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

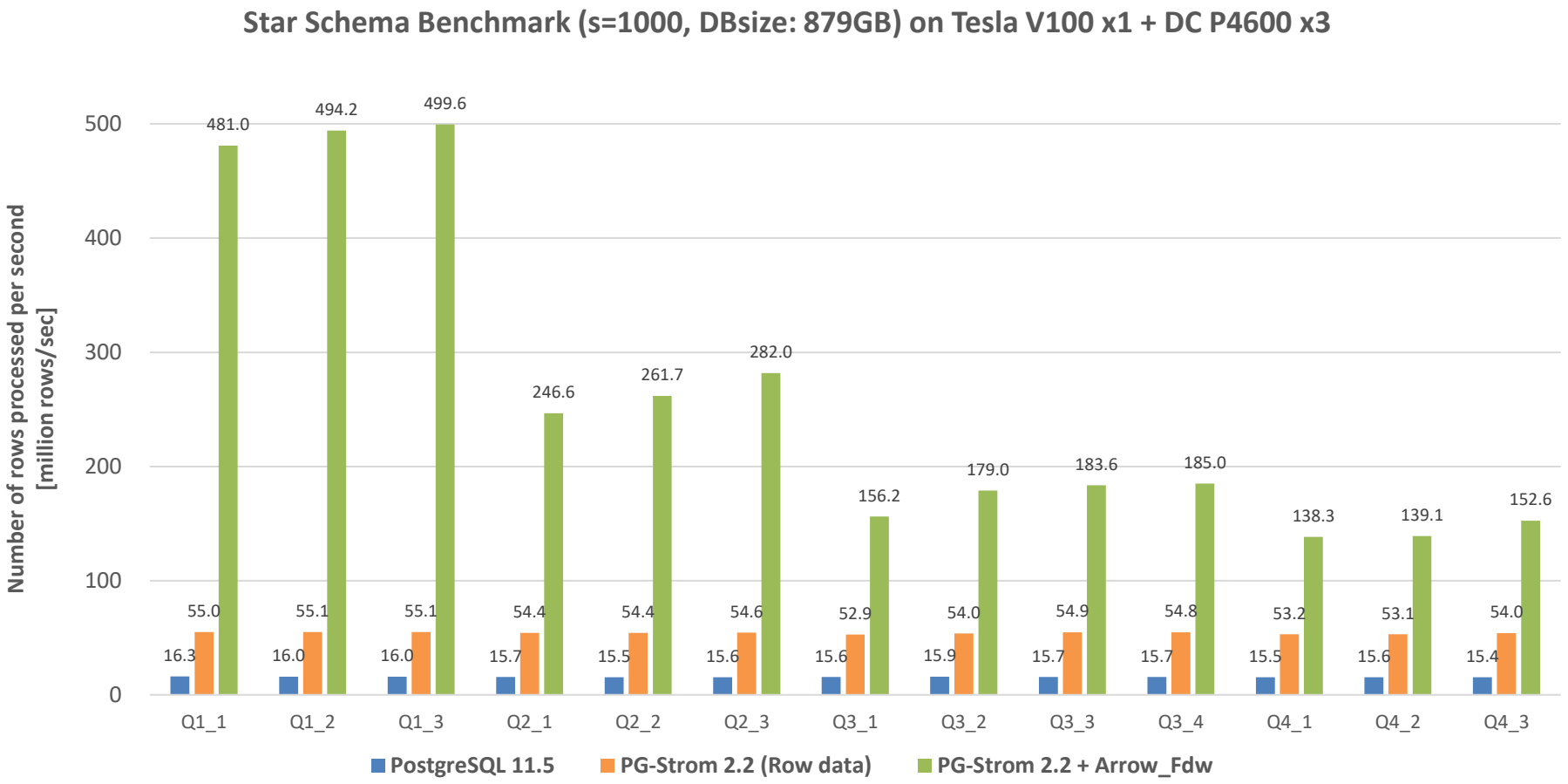
It transfers **ONLY Referenced Columns** over SSD-to-GPU Direct SQL mechanism



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 = (6.0B rows) / (Query Response Time [sec])
- Combined usage of SSD-to-GPU Direct SQL and Columnar-store pulled out the extreme performance; 130-500M rows per second
- Server configuration is identical to what we show on p.17. (1U, 1CPU, 1GPU, 3SSD)

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

Almost equivalent physical data transfer, but no need to load unreferenced columns

```
 Foreign table "public.flineorder"
       Column        |     Type      | Size
---------------------+---------------+--------------------------------
 lo_orderkey         | numeric       | 89.42GB
 lo_linenumber       | integer       | 22.37GB
 lo_custkey          | numeric       | 89.42GB
 lo_partkey          | integer       | 22.37GB   <-- ★Referenced by Q2_1
 lo_suppkey          | numeric       | 89.42GB   <-- ★Referenced by Q2_1
 lo_orderdate        | integer       | 22.37GB   <-- ★Referenced by Q2_1
 lo_orderpriority    | character(15) | 83.82GB
 lo_shippriority     | character(1)  |   5.7GB
 lo_quantity         | integer       | 22.37GB
 lo_extendedprice    | integer       | 22.37GB
 lo_ordertotalprice  | integer       | 22.37GB
 lo_discount         | integer       | 22.37GB
 lo_revenue          | integer       | 22.37GB   <-- ★Referenced by Q2_1
 lo_supplycost       | integer       | 22.37GB
 lo_tax              | integer       | 22.37GB
 lo_commit_date      | character(8)  | 44.71GB
 lo_shipmode         | character(10) | 55.88GB
FDW options: (file '/opt/nvme/lineorder_s401.arrow')  ... file size = 310GB
```

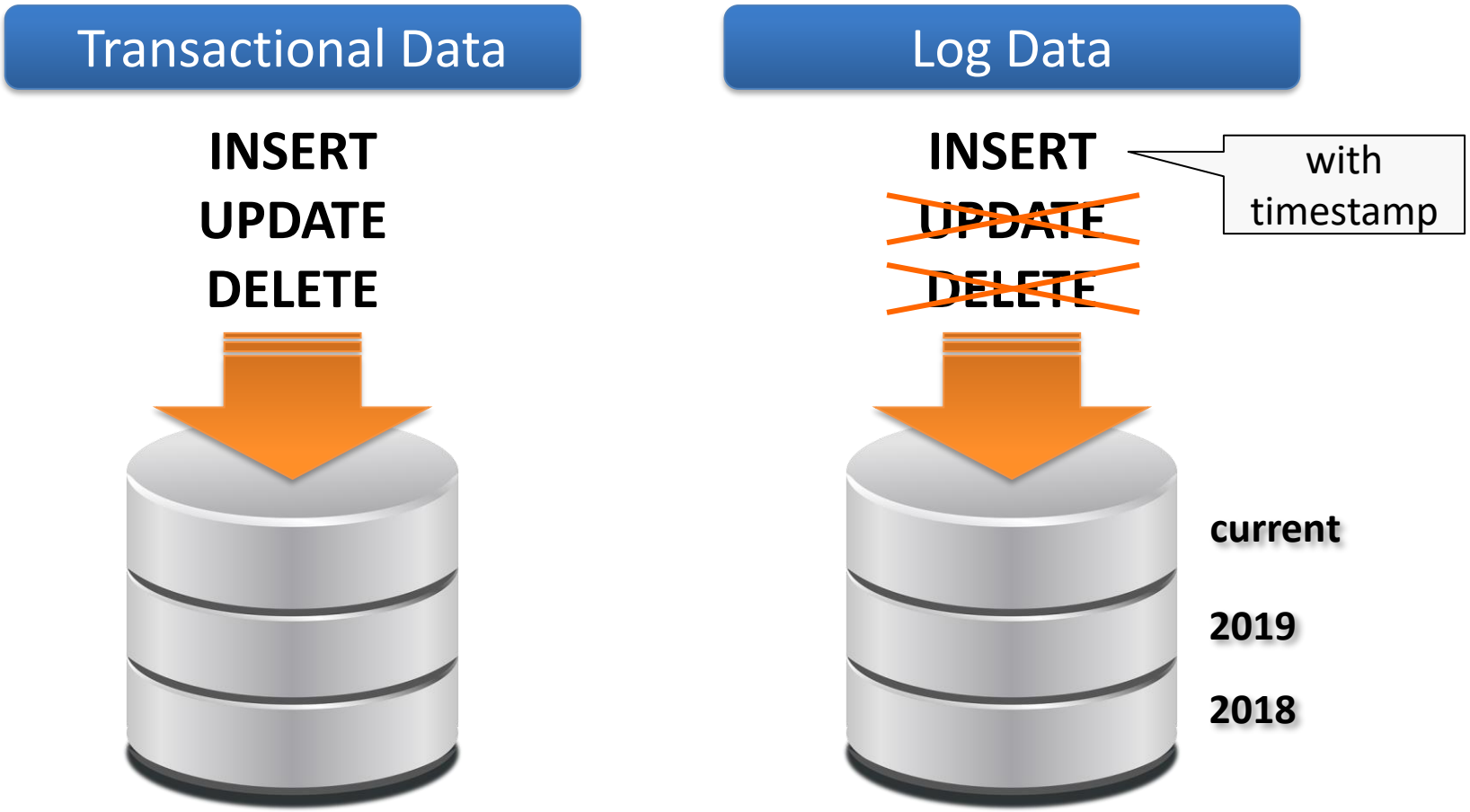
- Q2_1 actuall reads only 157GB of 681GB (23.0%) from the NVME-SSD.
- Execution time of Q2_1 is 24.3s, so 157GB / 24.3s = 6.46GB/s

➔ Reasonable performance for 3x Intel DC P4600 on single CPU configuration.

Efficient Data Deployment

Table Partitioning

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

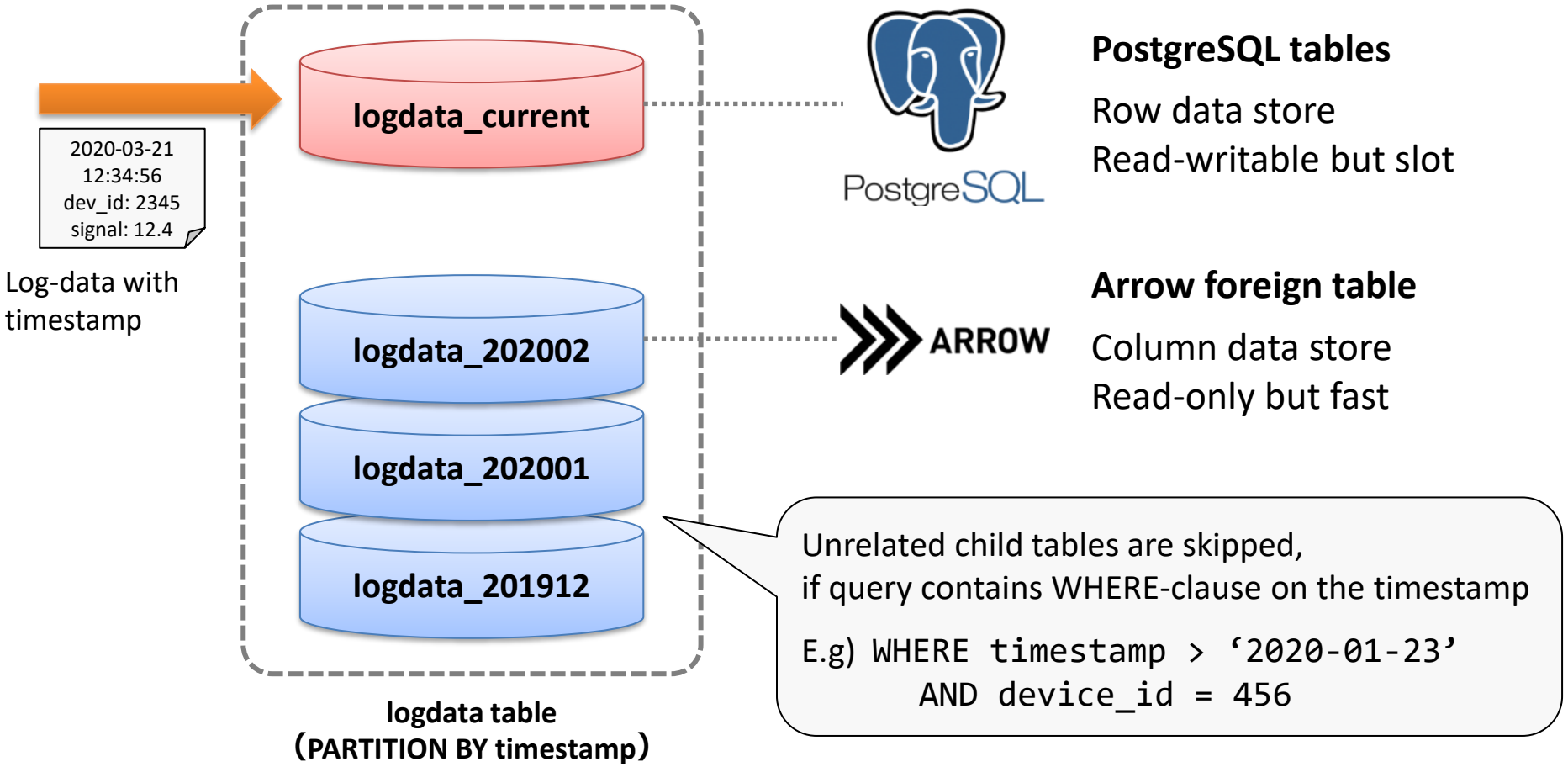


- ✓ MVCC visibility check is (relatively) not significant.
- ✓ Rows with old timestamp will never inserted.

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

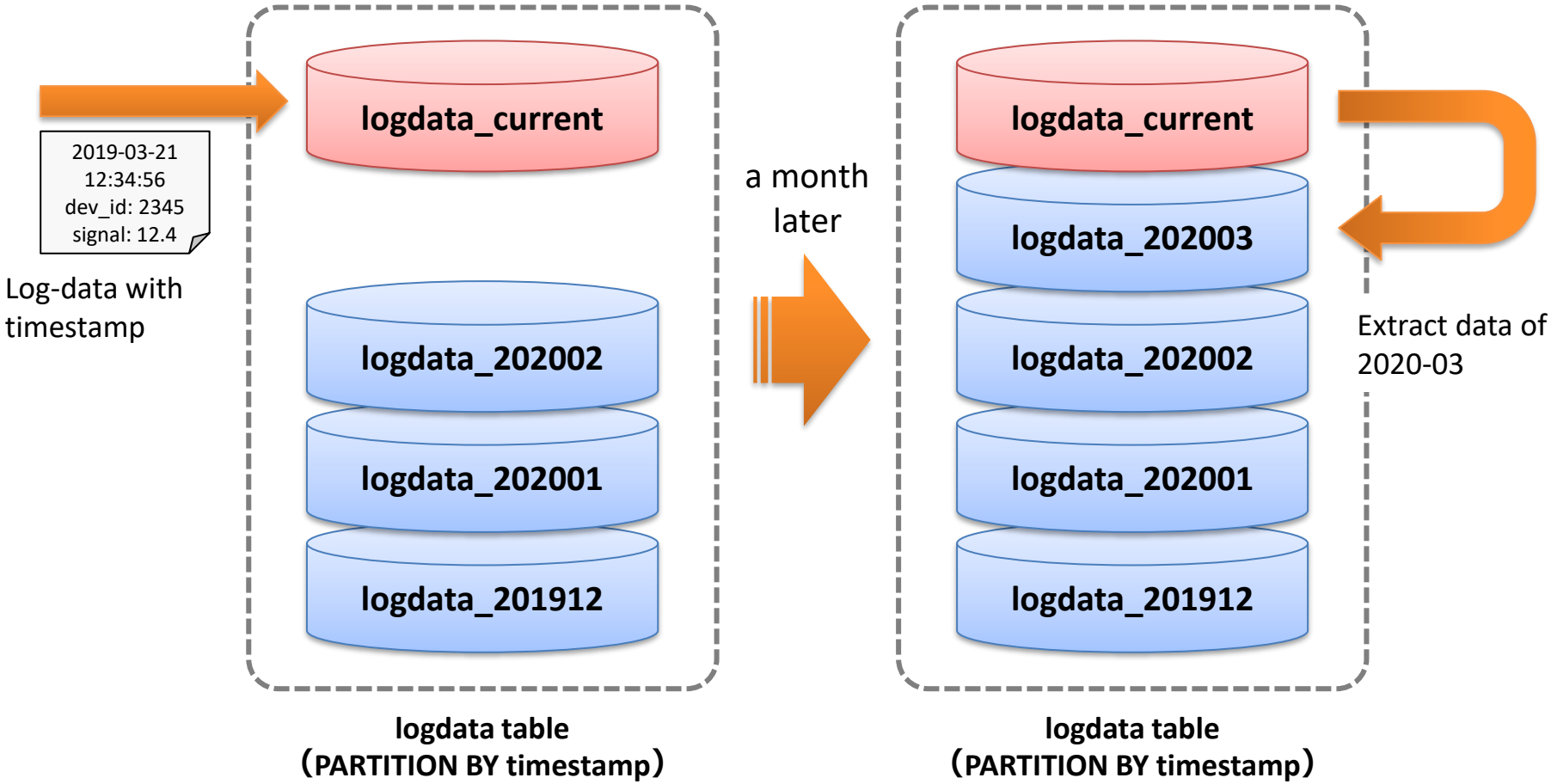
➔ Old data can be moved to Arrow foreign table for more efficient I/O



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

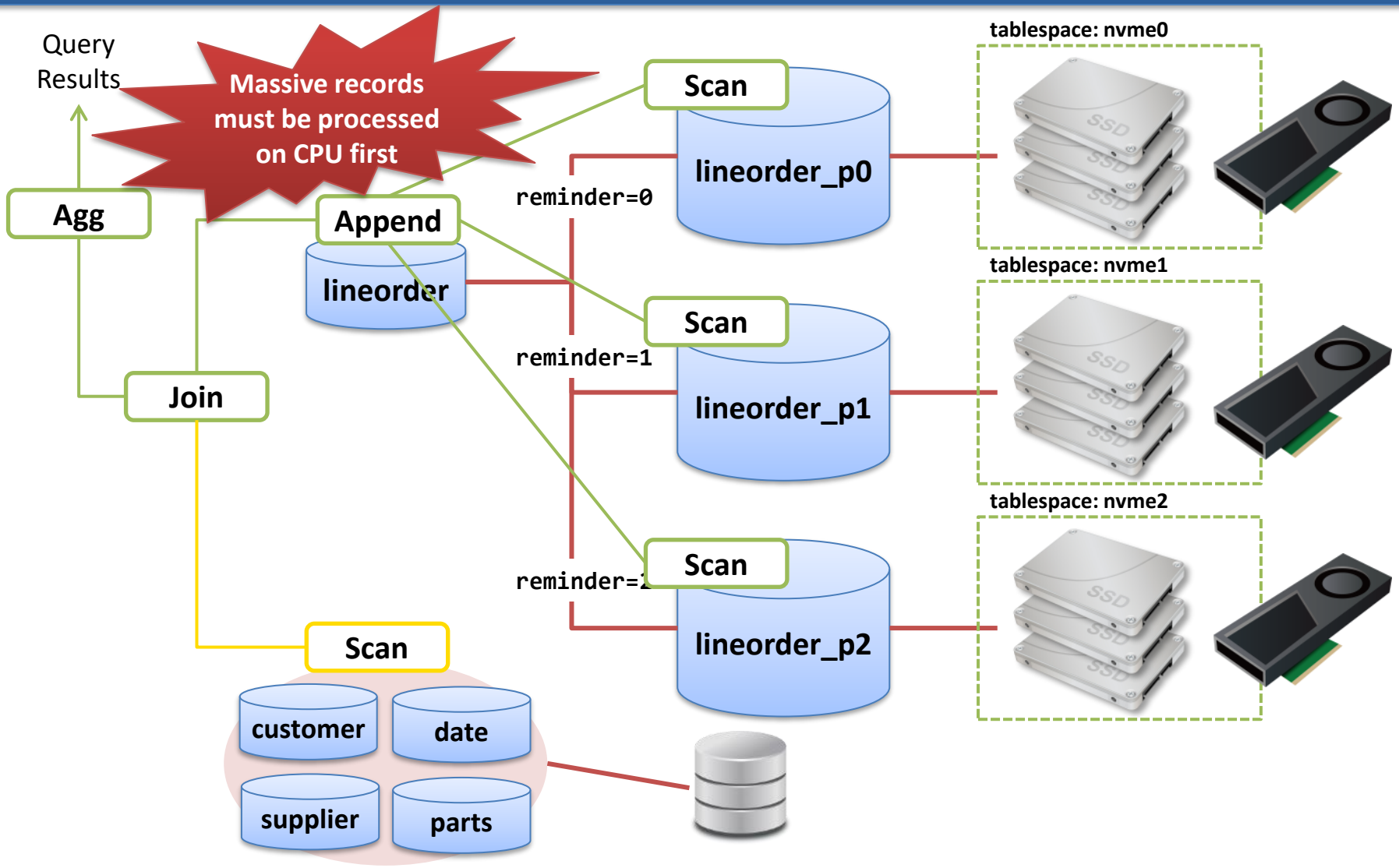
- Mixture of PostgreSQL table and Arrow foreign table in partition declaration
- Log-data should have timestamp, and never updated

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



Asymmetric Partition-wise JOIN (1/4)

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

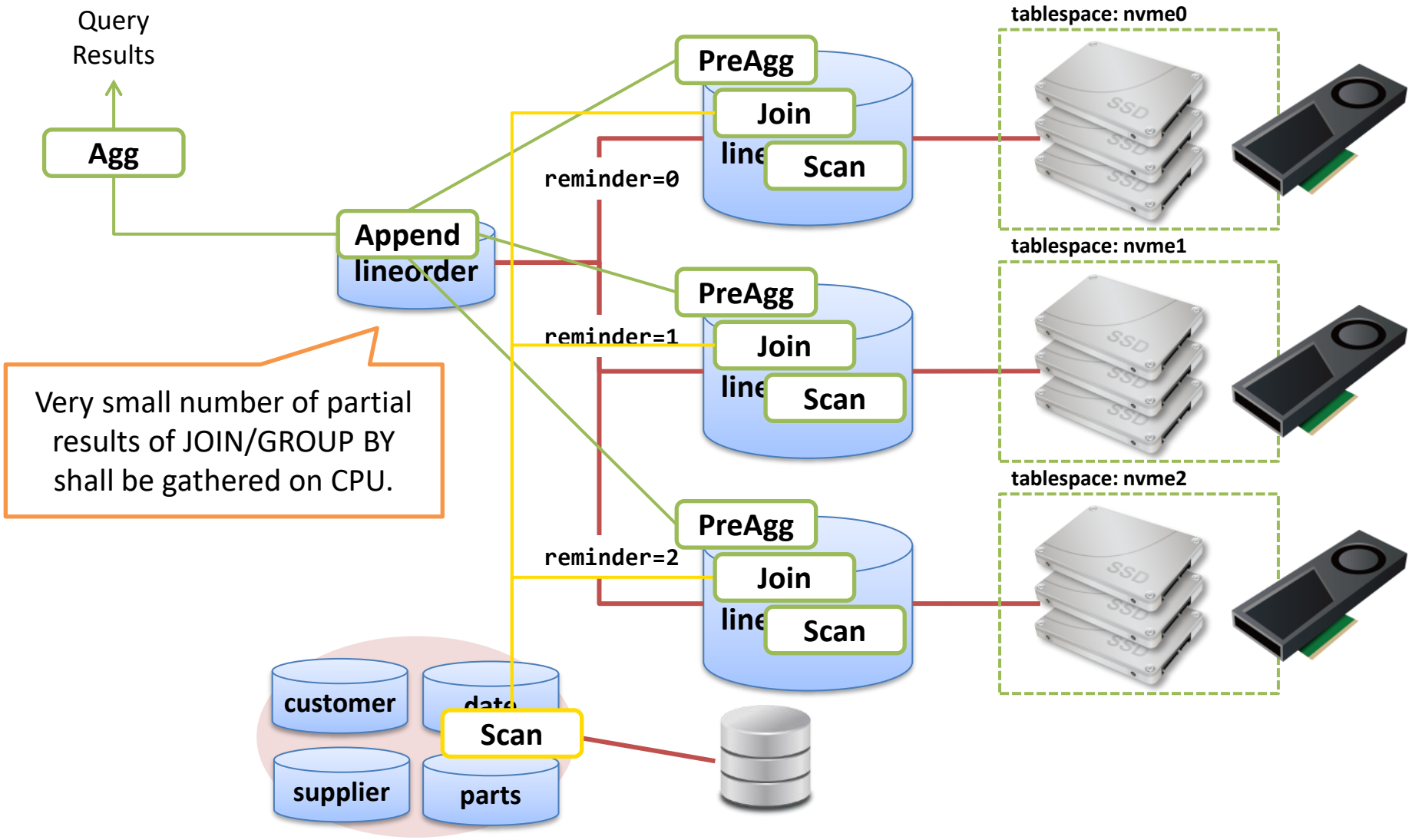


Asymmetric Partition-wise JOIN (2/4)

```
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 (3/4)

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



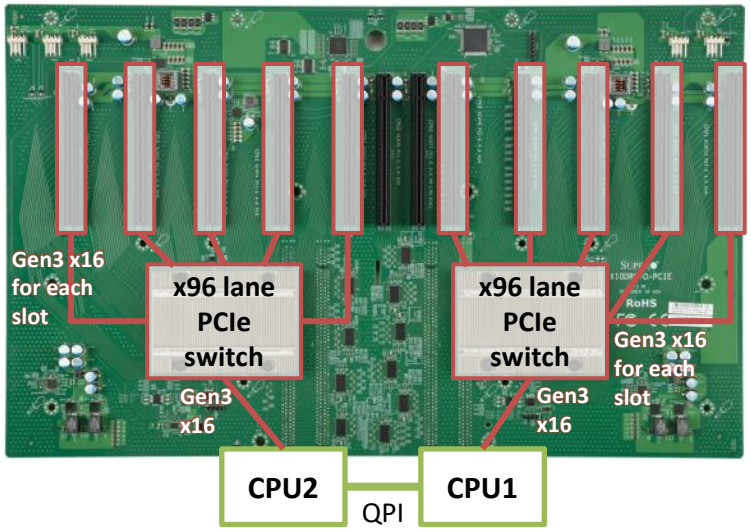
Asymmetric Partition-wise JOIN (4/4)

```
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)
```

PCIe-bus level optimization (1/3)

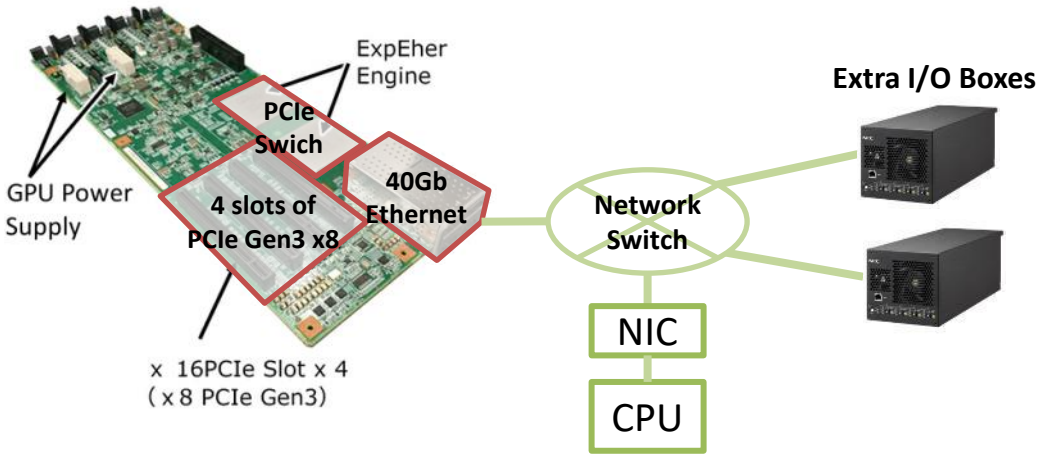
HPC Server – optimization for GPUDirect RDMA

Supermicro SYS-4029TRT2



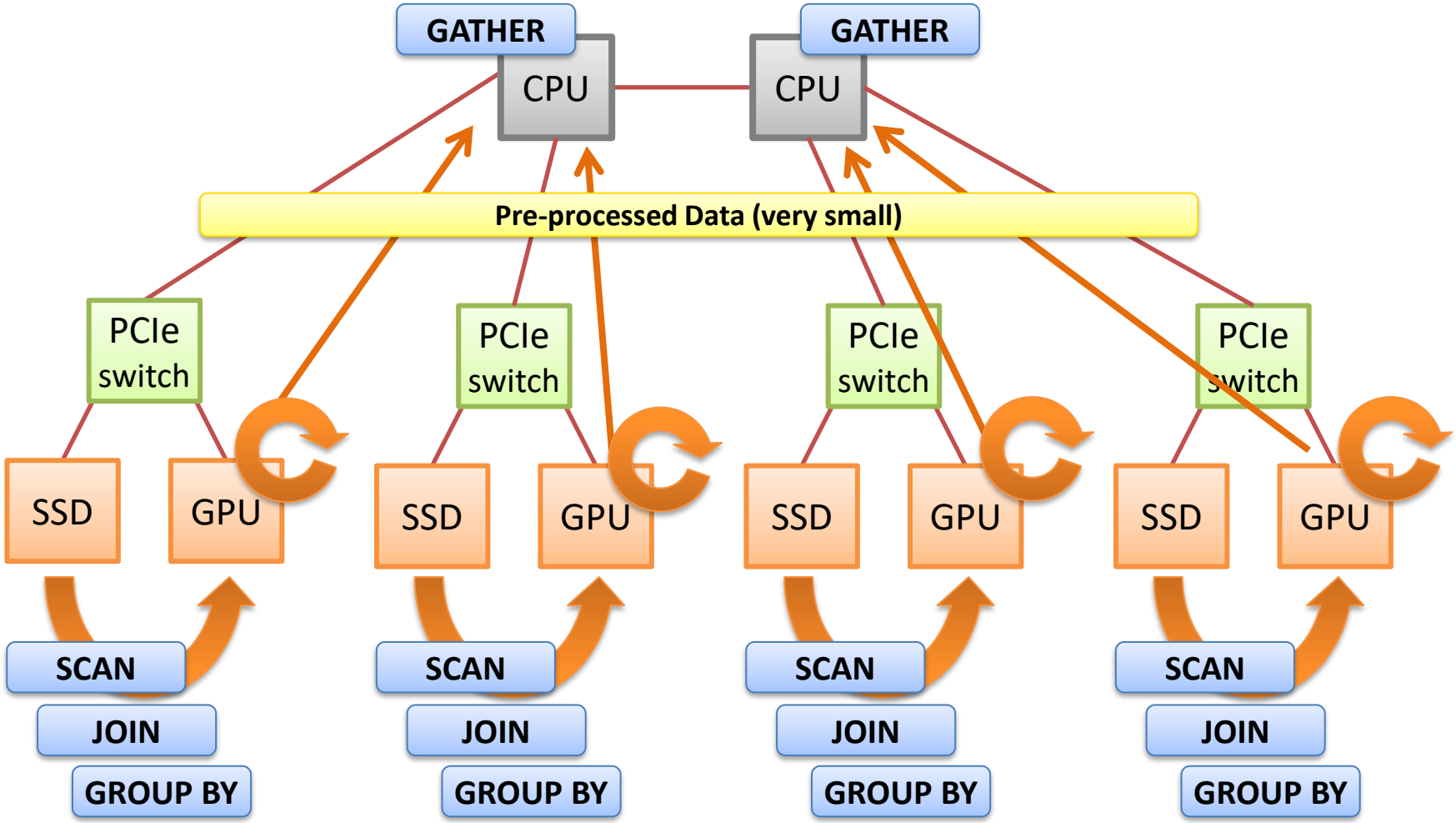
I/O Expansion Box

NEC ExpEther 40G (4slots edition)



PCIe-bus level optimization (2/3)

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



PCIe-bus level optimization (3/3)

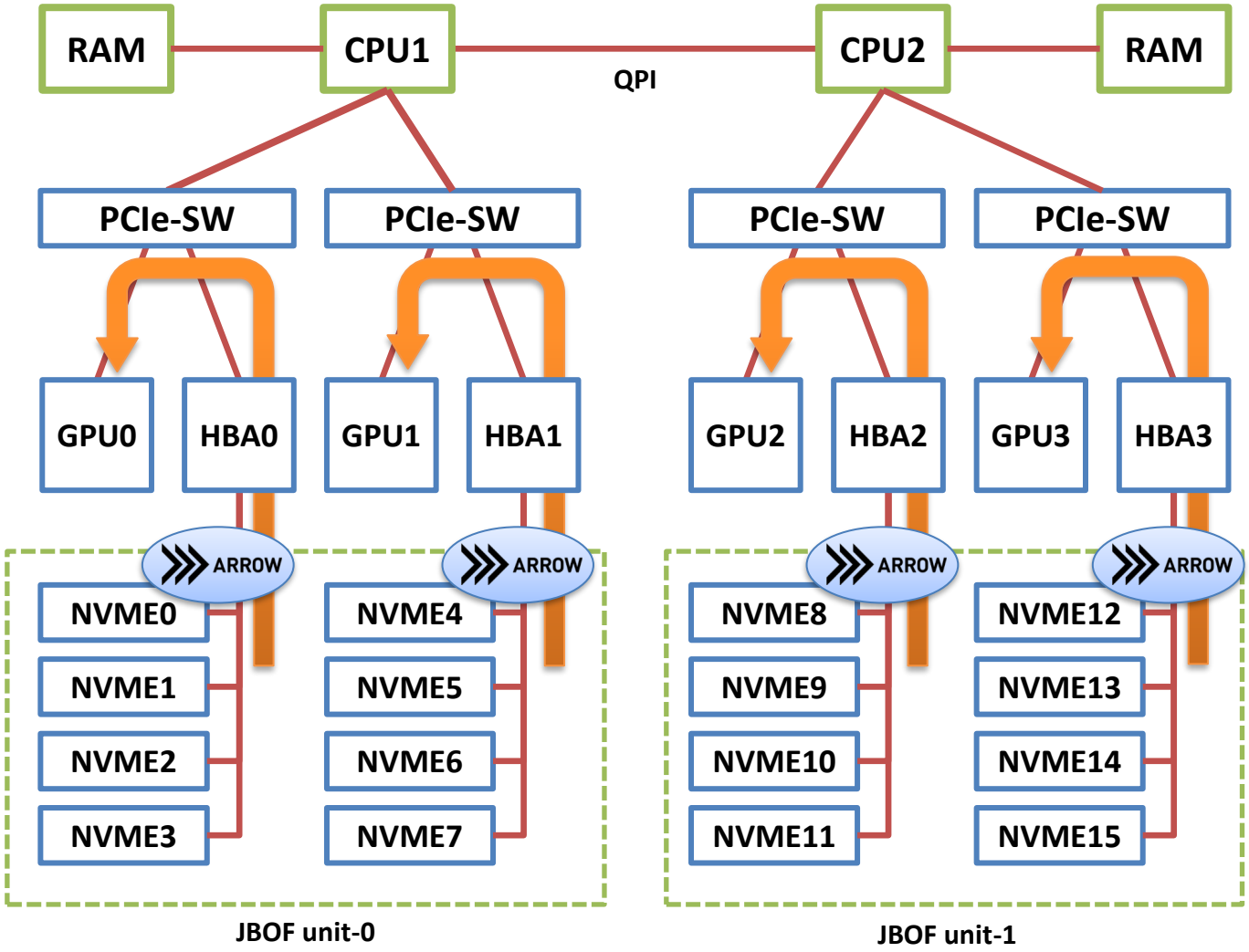
```
$ pg_ctl restart
    :
LOG:    -  PCIe[0000:80]
LOG:        -  PCIe(0000:80:02.0)
LOG:            -  PCIe(0000:83:00.0)
LOG:                -  PCIe(0000:84:00.0)
LOG:                    -  PCIe(0000:85:00.0) nvme0 (INTEL SSDPEDKE020T7)
LOG:                -  PCIe(0000:84:01.0)
LOG:                    -  PCIe(0000:86:00.0) GPU0 (Tesla P40)
LOG:                -  PCIe(0000:84:02.0)
LOG:                    -  PCIe(0000:87:00.0) nvme1 (INTEL SSDPEDKE020T7)
LOG:        -  PCIe(0000:80:03.0)
LOG:            -  PCIe(0000:c0:00.0)
LOG:                -  PCIe(0000:c1:00.0)
LOG:                    -  PCIe(0000:c2:00.0) nvme2 (INTEL SSDPEDKE020T7)
LOG:                -  PCIe(0000:c1:01.0)
LOG:                    -  PCIe(0000:c3:00.0) GPU1 (Tesla P40)
LOG:                -  PCIe(0000:c1:02.0)
LOG:                    -  PCIe(0000:c4:00.0) nvme3 (INTEL SSDPEDKE020T7)
LOG:        -  PCIe(0000:80:03.2)
LOG:            -  PCIe(0000:e0:00.0)
LOG:                -  PCIe(0000:e1:00.0)
LOG:                    -  PCIe(0000:e2:00.0) nvme4 (INTEL SSDPEDKE020T7)
LOG:                -  PCIe(0000:e1:01.0)
LOG:                    -  PCIe(0000:e3:00.0) GPU2 (Tesla P40)
LOG:                -  PCIe(0000:e1:02.0)
LOG:                    -  PCIe(0000:e4:00.0) nvme5 (INTEL SSDPEDKE020T7)
LOG:  GPU<->SSD Distance Matrix
LOG:              GPU0     GPU1     GPU2
LOG:       nvme0  (   3)       7        7
LOG:       nvme5      7       7   (   3)
LOG:       nvme4      7       7   (   3)
LOG:       nvme2      7   (   3)       7
LOG:       nvme1  (   3)       7        7
LOG:       nvme3      7   (   3)       7
```

PG-Strom chooses the closest GPU for the tables to be scanned, according to the PCI-E device distance.

Large-scale Benchmark

Large-scale Benchmark (1/4)

All of SSD-to-GPU Direct SQL, Apache Arrow and PCI-E bus optimization are used



8.0-10GB/s of physical data transfer capability per GPU+4xSSD unit

25-30GB/s of effective data transfer capability by columnar data structure per unit

100-120GB/s of effective data transfer capability by 4-units' parallel

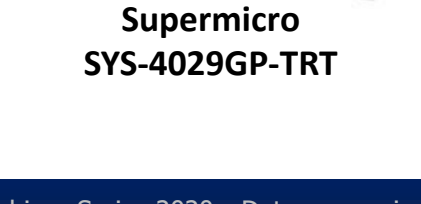
Capable to process **100TB Log data** on a single-node in reasonable time

Large-scale Benchmark (2/4)

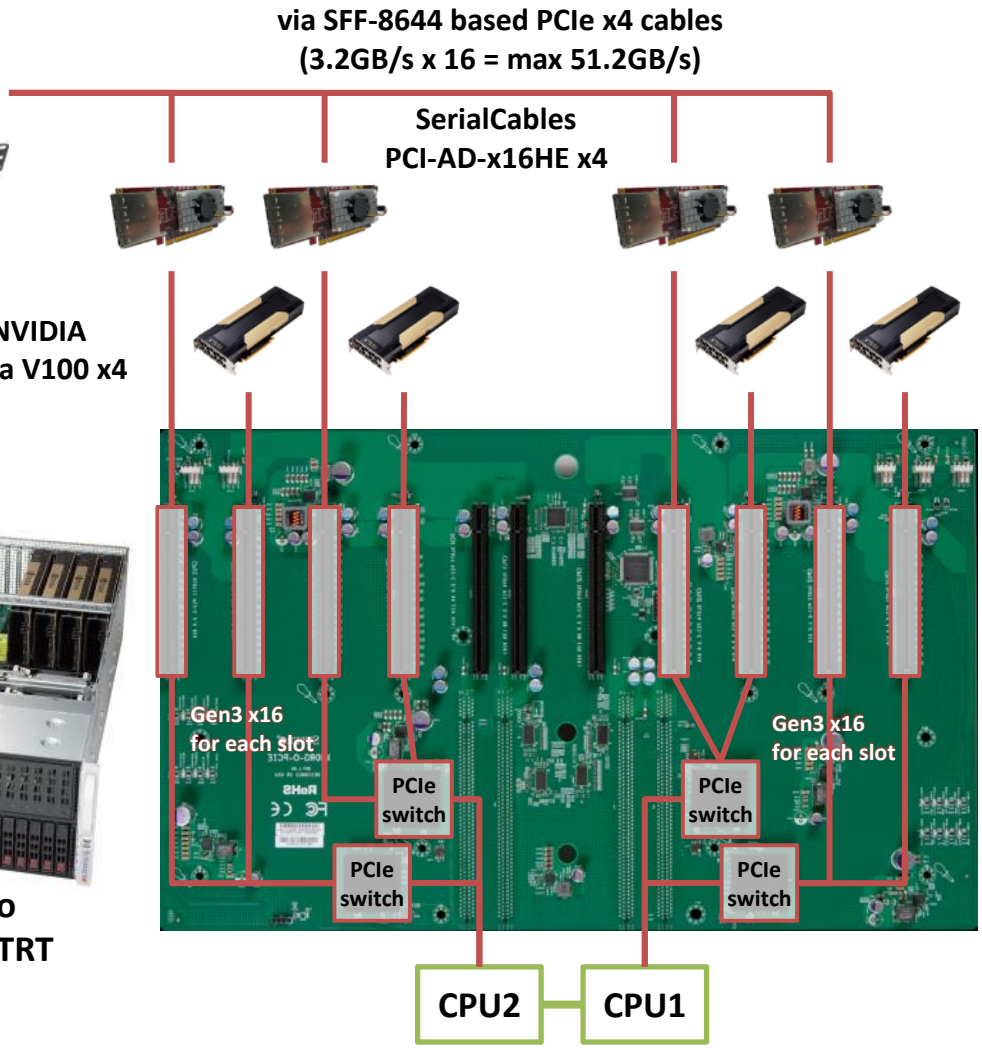
Build HPC 4U Server + 4 of GPU + 16 of NVME-SSD configuration

Intel SSD
DC P4510 (1.0TB) x16

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



Supermicro
SYS-4029GP-TRT



SPECIAL THANKS

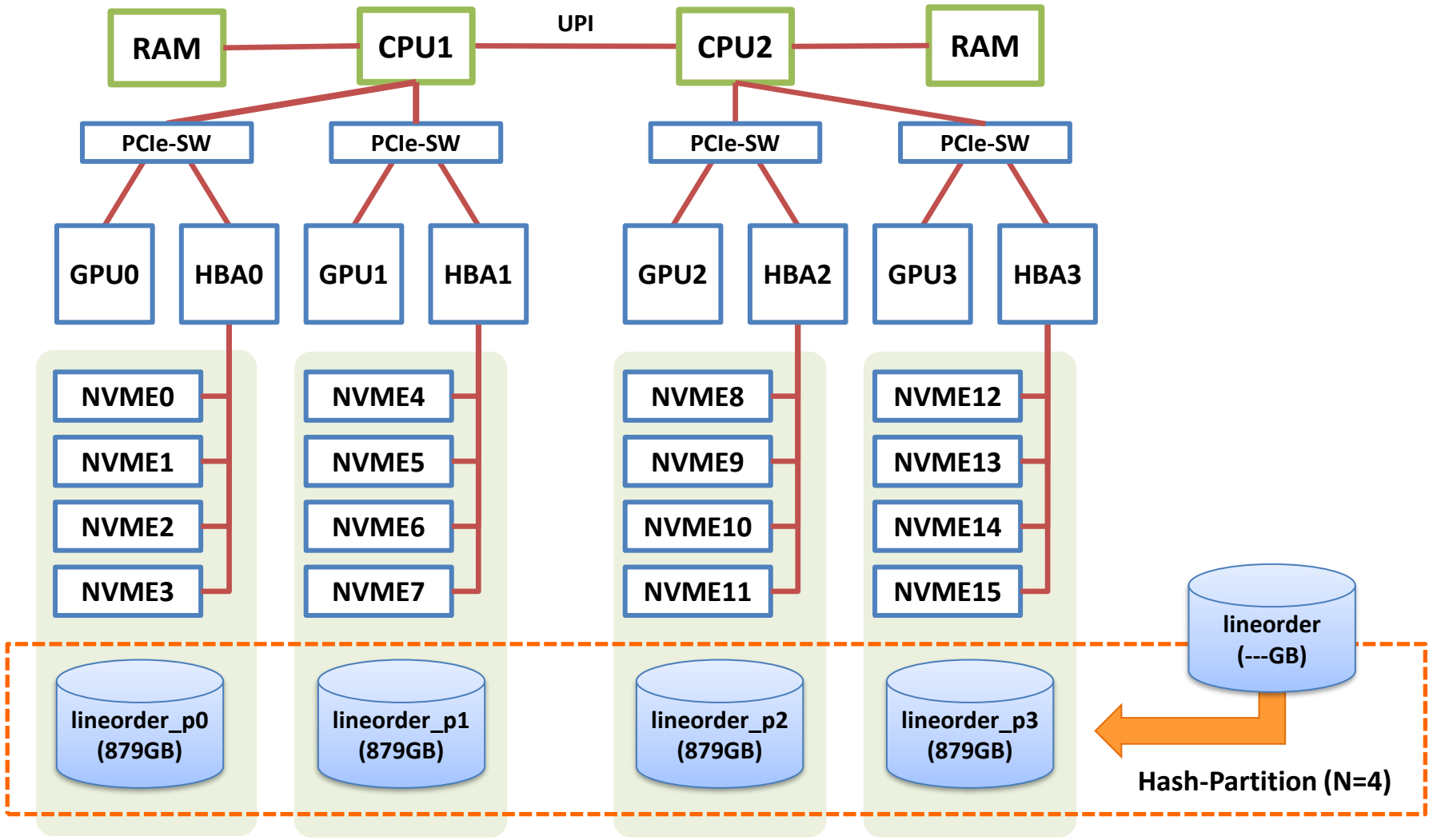
NVIDIA
MKI 三井情報株式会社 MITSUI KNOWLEDGE INDUSTRY
intel 中小企業庁

Large-scale Benchmark (3/4)

Build HPC 4U Server + 4 of GPU + 16 of NVME-SSD configuration

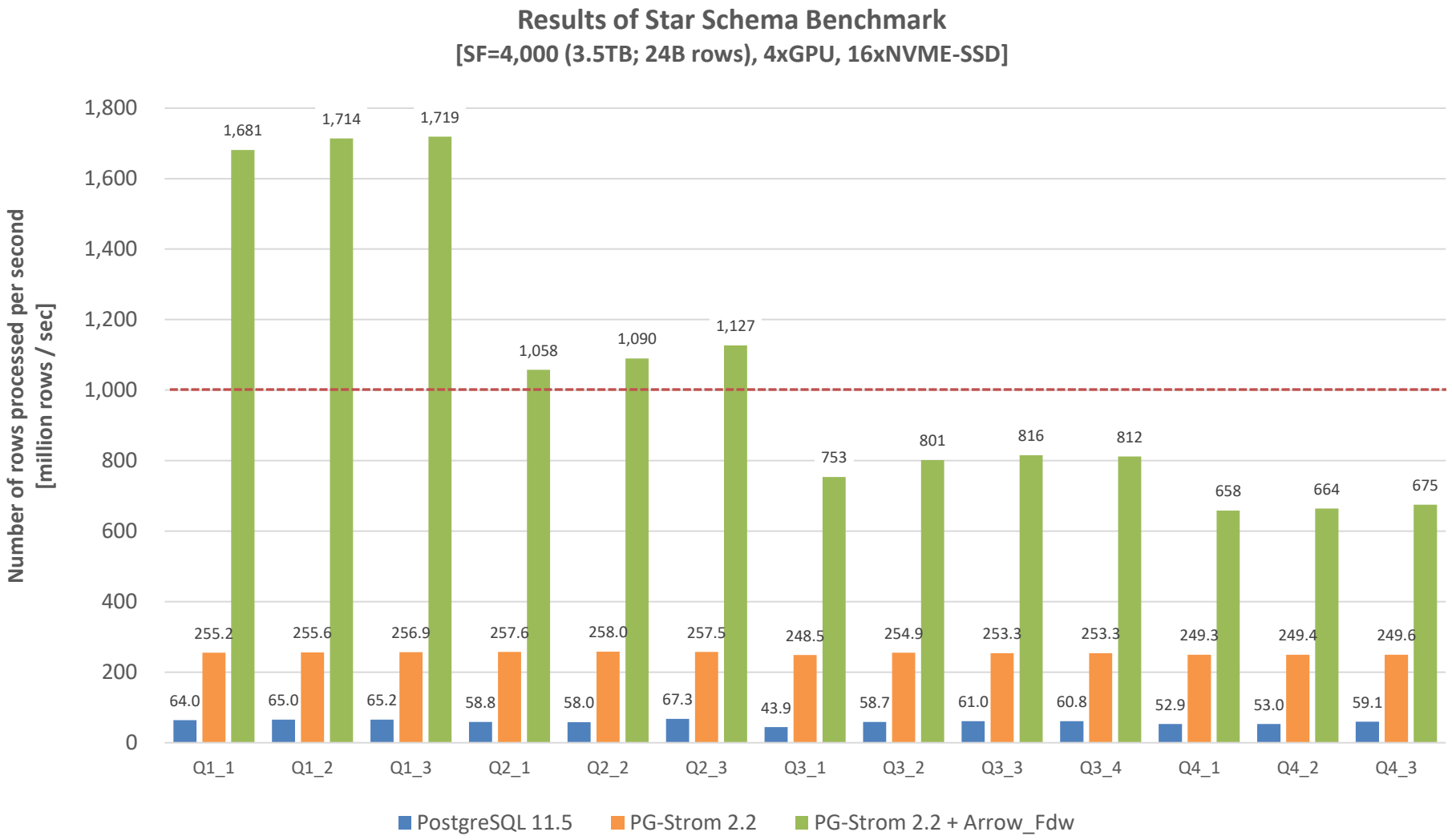
Large-scale Benchmark (4/4)

Distributed 3.5TB test data into 4 partitions; 879GB for each



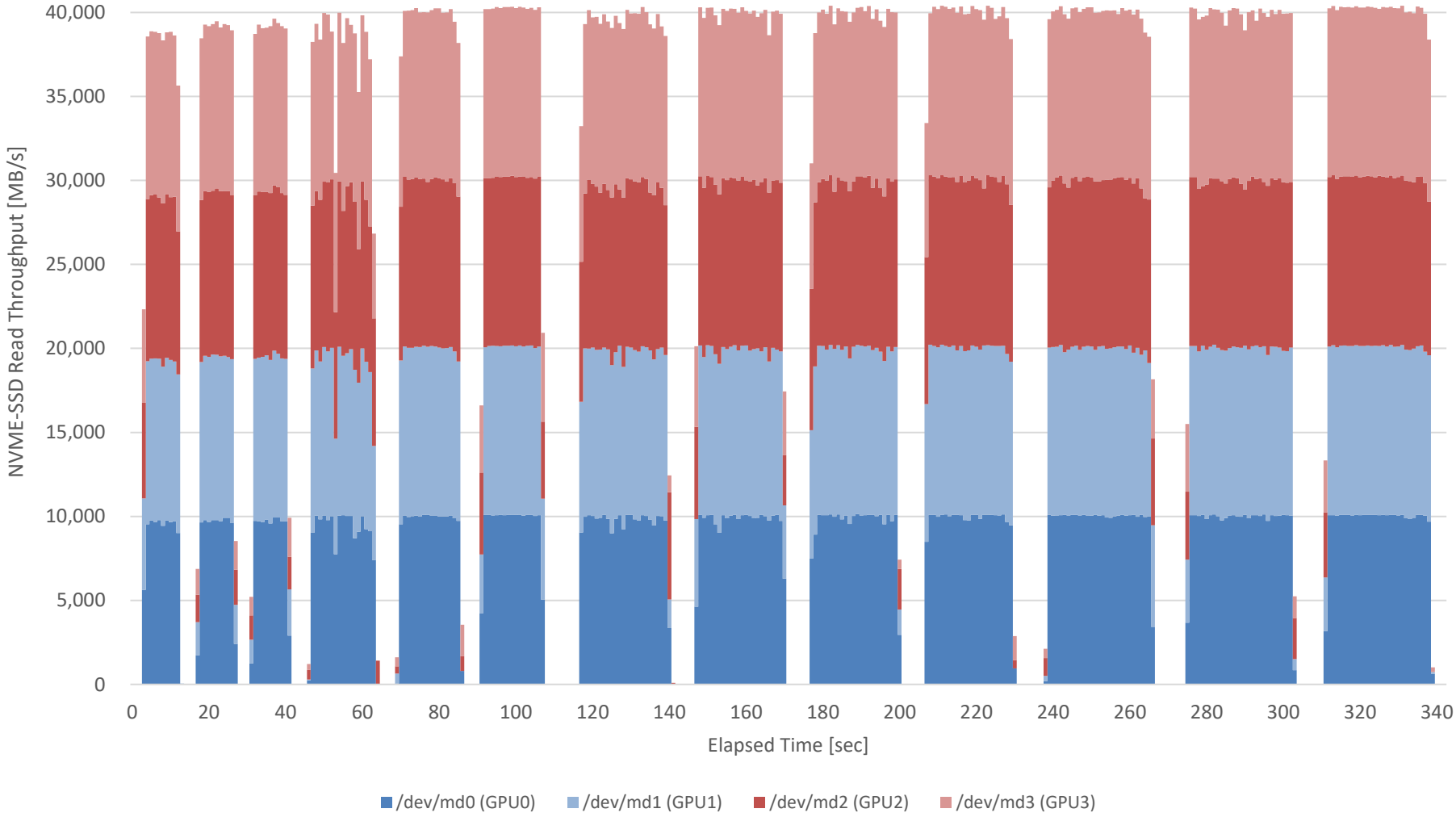
Benchmark Results (1/2)

Billion rows per second at a single-node PostgreSQL!!



Benchmark Results (2/2)

40GB/s Read from NVME-SSDs during SQL execution (95% of H/W limit)



Future Direction

NVIDIA GPUDirect Storage (cuFile API) support

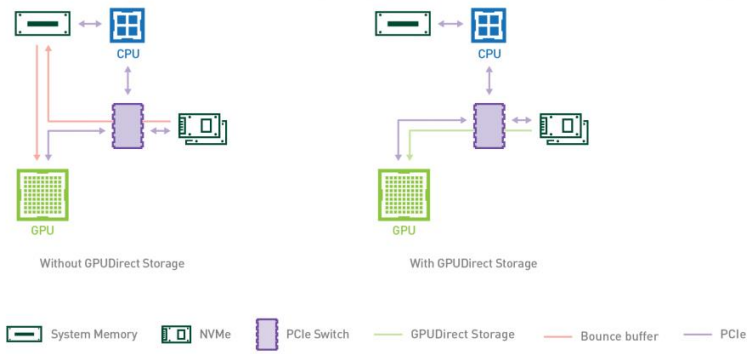
WIP

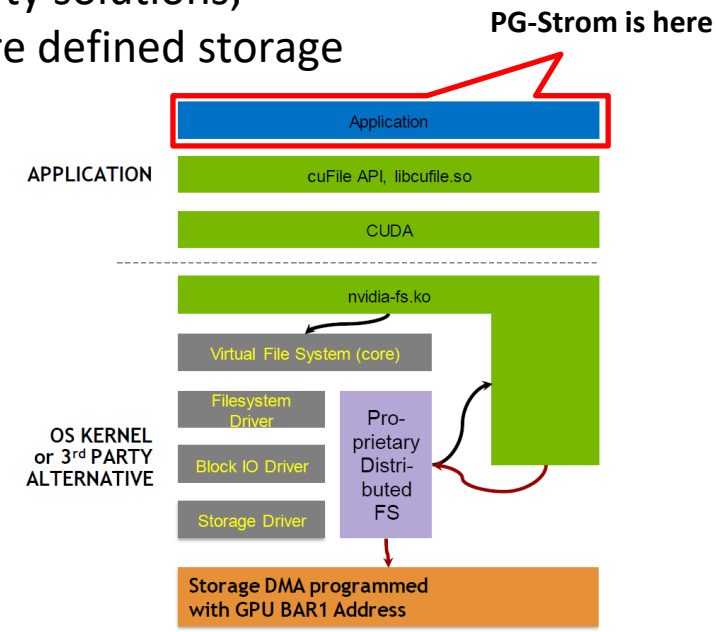
NVIDIA GPUDirect Storage?

- API & Driver stack for direct read from NVME to GPU
  - ✓ Features are almost equivalend to NVME-Strom
  - ✓ Ubuntu 18.04 & RHEL8/CentOS8 shall be released at the initial release
- NVIDIA will officially release the software in 2020.

Why beneficial?

- Linux kernel driver that is tested/evaluated by NVIDIA's QA process.
- Involvement of broader ecosystem with third-party solutions;
  like distributed-filesystem, block storage, software defined storage
- Broader OS support; Ubuntu 18.04.





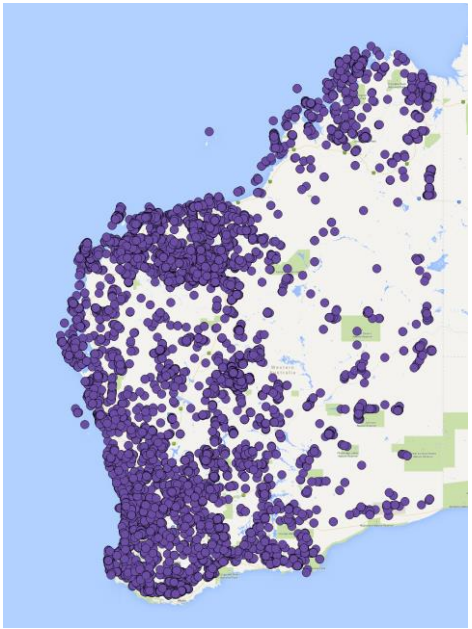
Ref: **GPUDIRECT STORAGE: A DIRECT GPU-STORAGE DATA PATH**
https://on-demand.gputechconf.com/supercomputing/2019/pdf/sc1922-gpudirect-storage-transfer-data-directly-to-gpu-memory-alleviating-io-bottlenecks.pdf

PostGIS Functions & Operator support (1/2)

WIP

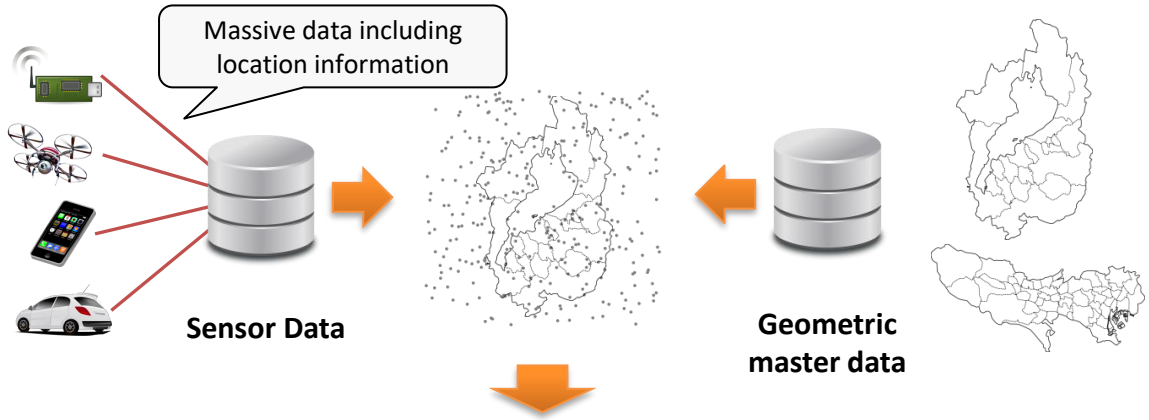
PostGIS

- An extension of handle geographic objects and relational operations on them.
  - ✓ Geographic objects: Point, LineString, Polygon, and etc ....
  - ✓ Geographic relational operations: contains, crosses, touch, distance, and etc...
- Wise design for fast search
  - ✓ Bounding-box, GiST-index, CPU Parallel
- v1.0 is released at 2005, then its development has continued over 15 years.



© GAIA RESOURCES

【IoT/M2Mデータに対しての想定利用シーン】



PostGIS Functions & Operator support (2/2)

WIP

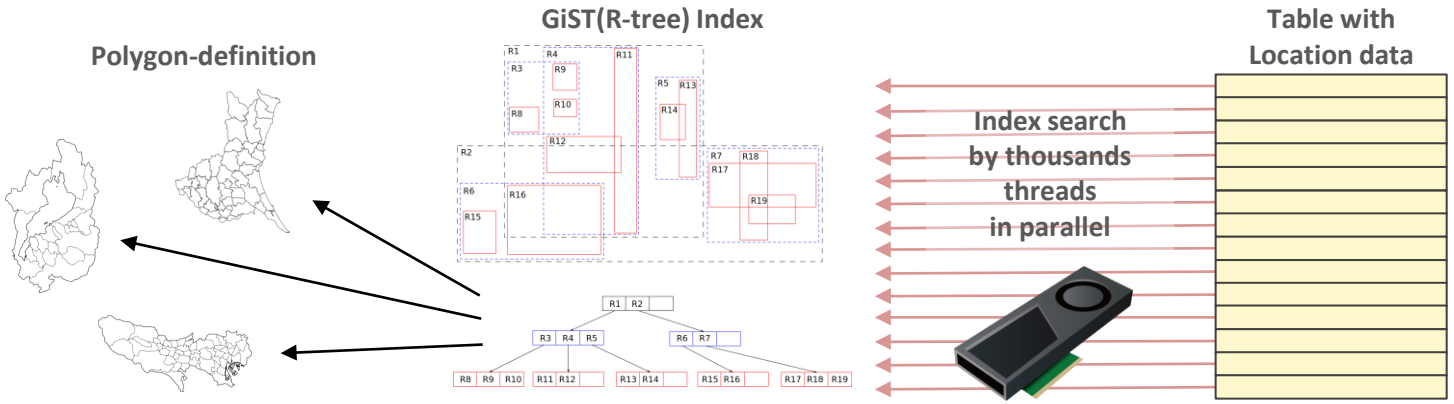
Current status

- `geometry st_makepoint(float8,float8,...)`
- `float8 st_distance(geometry,geometry)`
- `bool st_dwithin(geometry,geometry,float8)`
- `bool st_contains(geometry,geometry)`
- `bool st_crosses(geometry,geometry)`

Next: GiST (R-tree) Index support

- Index-based nested-loop up to million polygon x billion points.
- Runs index-search by GPU's thousands threads in parallel.

➔ Because of GiST internal structure, GPU-parallel is likely efficient.



Resources

Repository

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

Document

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

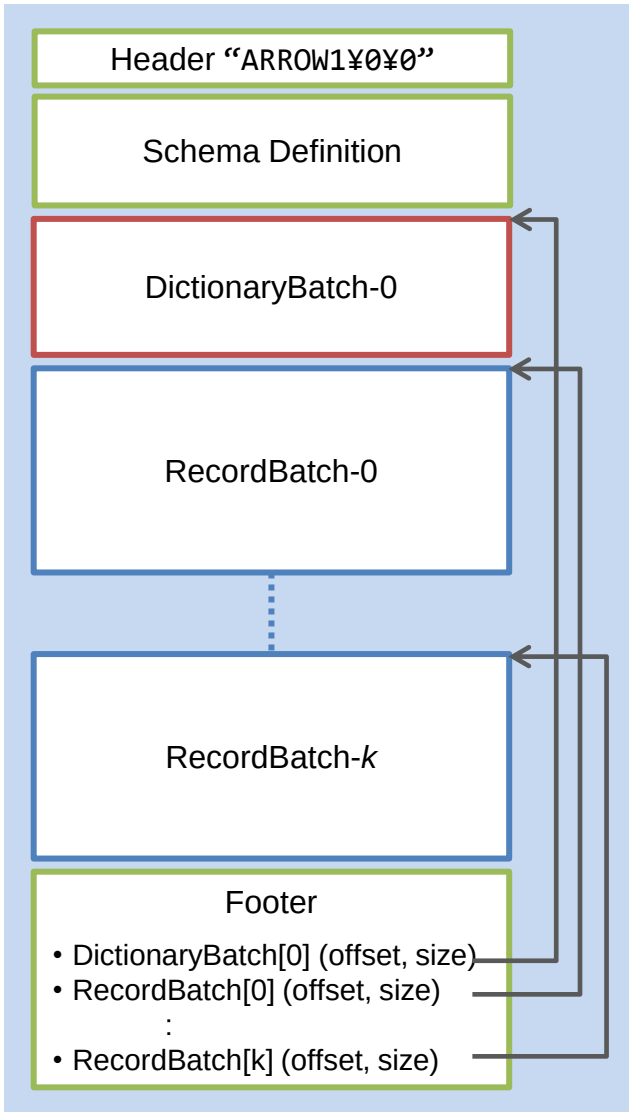
Contact

- kaigai@heterodb.com
- Tw: @kkaigai / @heterodb


HeteroDB

Backup Slides

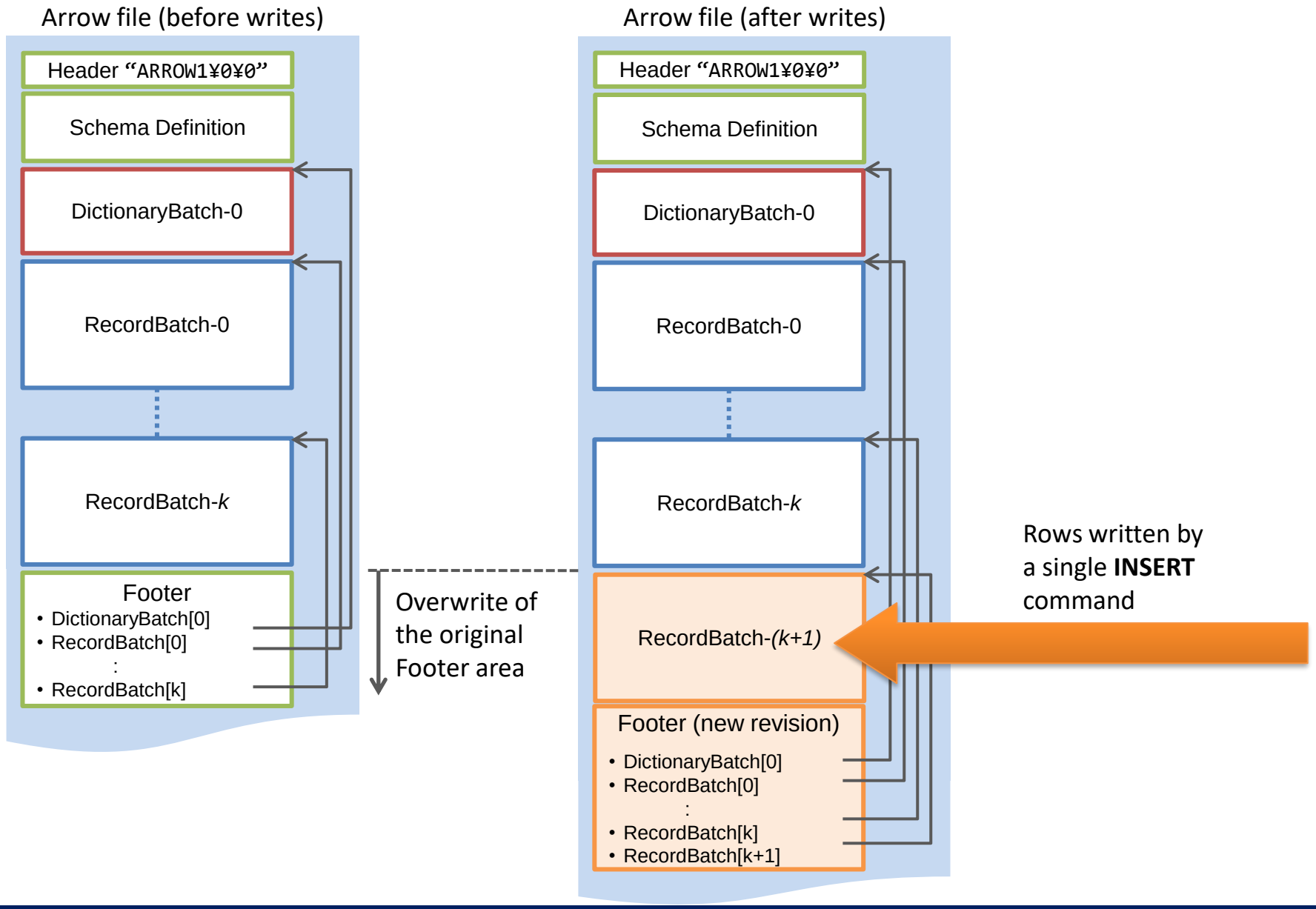
Apache Arrow Internal (1/3)



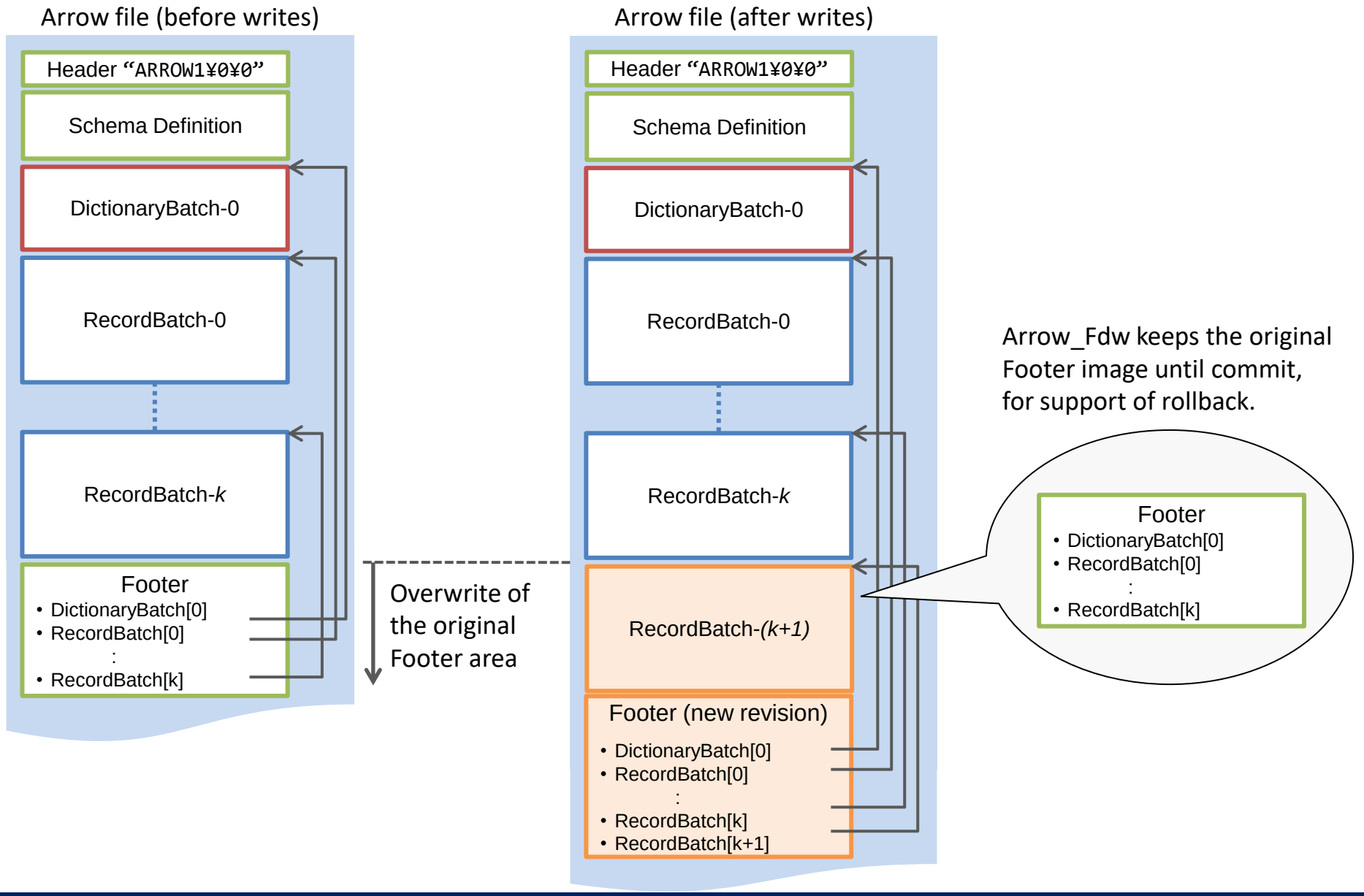
Apache Arrow Internal

- Header
  - Fixed byte string: “ARROW1¥0¥0”
- Schema Definition
  - # of columns and columns definitions.
  - Data type/precision, column name, column number, ...
- Record Batch
  - Internal data block that contains a certain # of rows.
  - E.g, if N=1M rows with (Int32, Float64), this record-batch begins from 1M of Int32 array, then 1M of Float64 array follows.
- Dictionary Batch
  - Optional area for dictionary compression.
  - E.g) 1 = “Tokyo”, 2 = “Osaka”, 3 = “Kyoto”
    - ➔ Int32 representation for frequently appearing words
- Footer
  - Metadata - offset/length of RecordBatches and DictionaryBatches

Apache Arrow Internal (2/3) - writable Arrow_Fdw

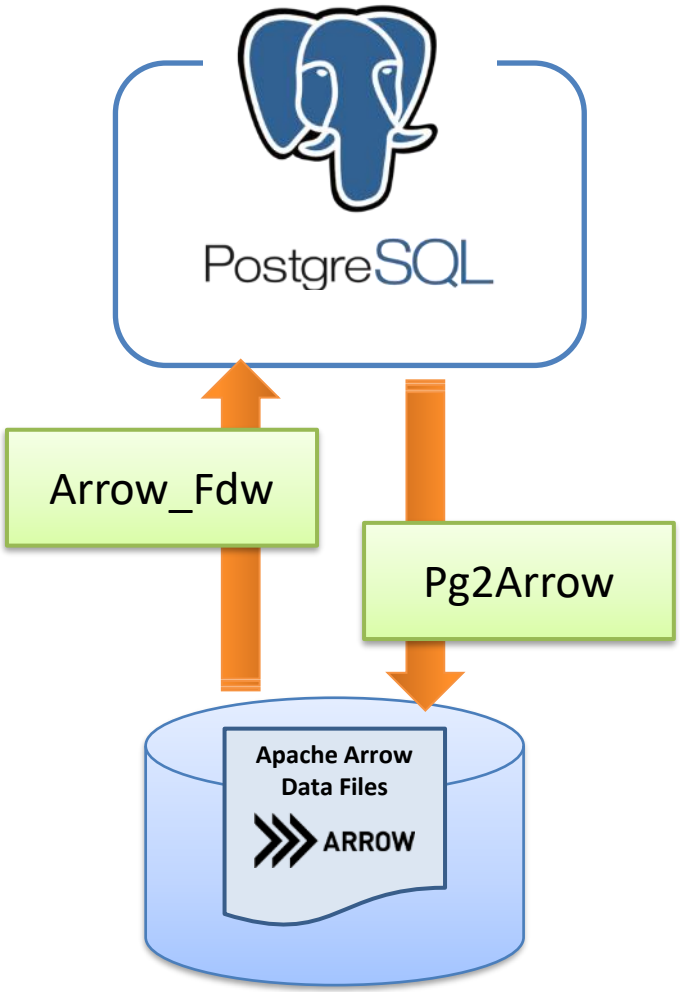


Apache Arrow Internal (3/3) - writable Arrow_Fdw



Pg2Arrow / MySQL2Arrow

Pg2Arrow / MySQL2Arrow enables to dump SQL results as Arrow file




```
$ pg2arrow --help
Usage:
  pg2arrow [OPTION] [database] [username]

General options:
  -d, --dbname=DBNAME   Database name to connect to
  -c, --command=COMMAND SQL command to run
  -t, --table=TABLENAME Table name to be dumped
      (-c and -t are exclusive, either of them must be given)
  -o, --output=FILENAME result file in Apache Arrow format
      --append=FILENAME result Apache Arrow file to be appended
      (--output and --append are exclusive. If neither of them
       are given, it creates a temporary file.)

Arrow format options:
  -s, --segment-size=SIZE size of record batch for each

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

Other options:
      --dump=FILENAME  dump information of arrow file
      --progress       shows progress of the job
      --set=NAME:VALUE config option to set before SQL execution
      --help           shows this message
```

✓ `mysql2arrow` also has almost equivalent capability