PostgreSQLは最新ハードウェアでどこまでやれるのか？

～GPUとNVMEで実現する超高速ログデータ処理基盤～

HeteroDB,Inc
KaiGai Kohei <kaigai@heterodb.com>

HeteroDB社について

ヘテロジニアスコンピューティング技術を
データベース領域に適用し、
誰もが使いやすく、安価で高速なデータ解析基盤を提供する。

会社概要

商号	ヘテロDB株式会社
創業	2017年7月4日
拠点	品川区西大井1-1-2-201
事業内容	高速データベース製品の販売 GPU＆DB領域の技術コンサルティング

NVIDIA INCEPTION PROGRAM

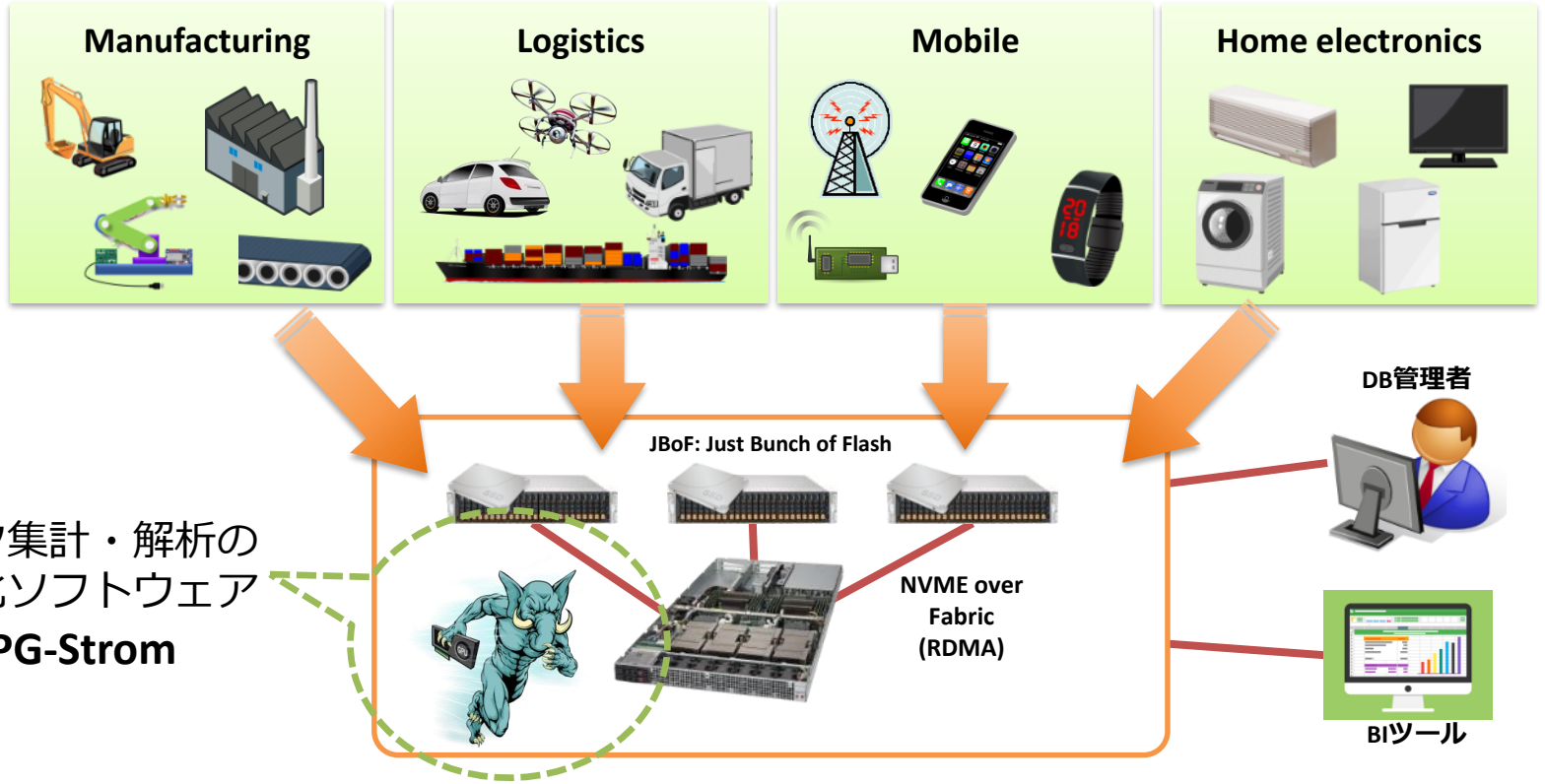
IPA 情報処理推進機構

代表者プロフィール

- 海外　浩平（KaiGai Kohei）
- OSS開発者コミュニティにおいて、PostgreSQLやLinux kernelの開発に15年以上従事。主にセキュリティ・FDW等の分野でアップストリームへの貢献。
- 2007年　IPA未踏ソフト事業において“天才プログラマー”認定
- 2017年　GPU Technology ConferenceでTop-5 Posters Finalistに選定

私たちの目指すもの – IoT/M2M向けログデータ処理基盤

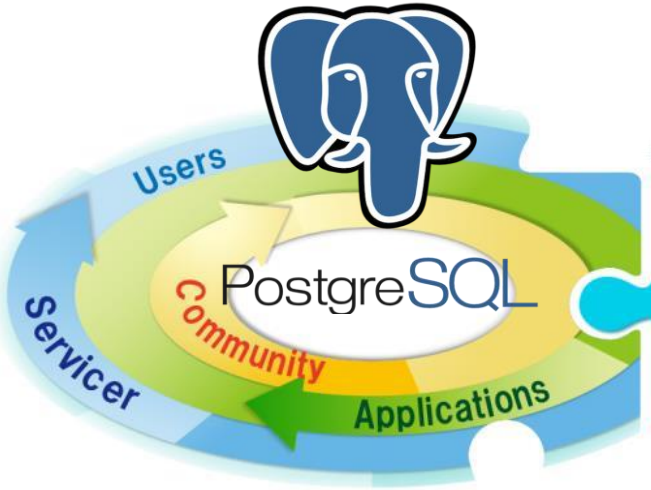
増え続けるログデータの管理、集計から解析・機械学習まで対応したデータ管理基盤

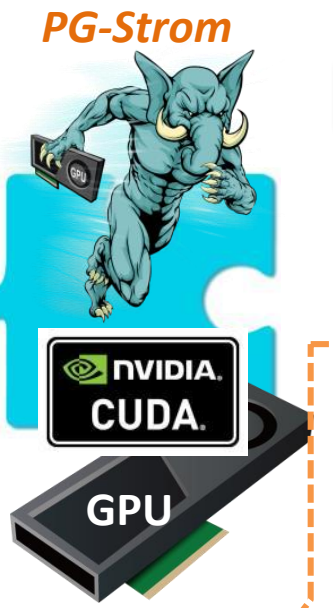


- 処理ユニットの拡張により、**シングルノードで～数十TB**のデータにも対応。
  - ➔ システム全体の大幅なダウンサイジングが可能に（ラック ➔ 2Uサイズ）
- 蓄積した**生ログデータのまま**、H/W限界に近い速度で集計する事が可能。
- **使い慣れた**PostgreSQLのSQL構文、周辺アプリケーションを適用可能。

中核技術 – PG-Strom

PG-Strom: GPUの持つ数百～数千コアと広帯域メモリを利用して、SQLワークロードを高速化するPostgreSQL向け拡張モジュール





RDBに求められる諸機能
- ✓ 可用性／クラスタリング
- ✓ バックアップ／運用管理
- ✓ トランザクション
- ✓ ＢＩ・可視化

➔ PostgreSQLのものをAs-Isで利用

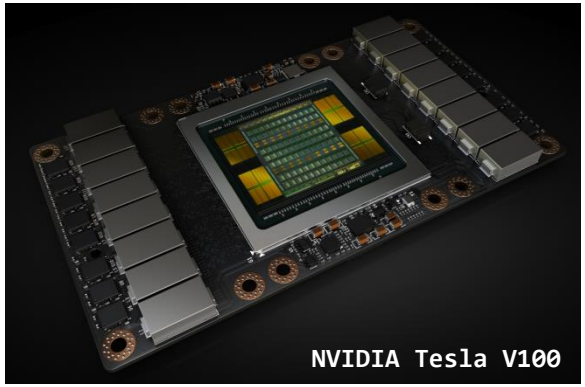
機械学習・統計解析

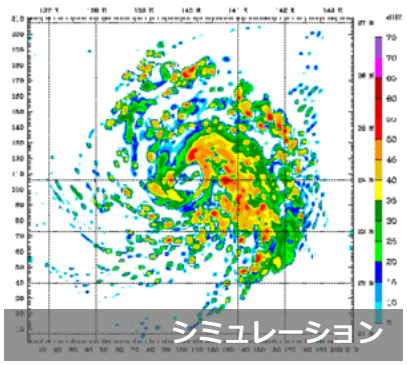
大量データの集計・解析

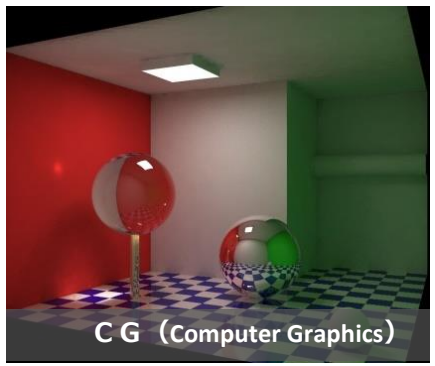
大量のデータを高速にストレージから読み出す

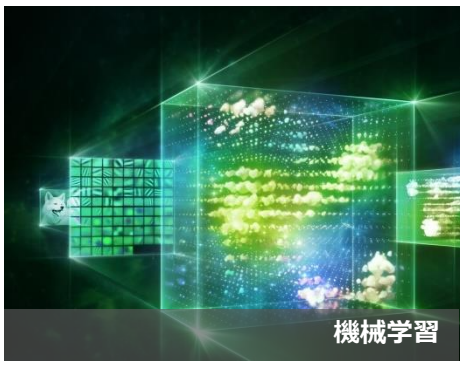
GPUとはどんなプロセッサなのか？

主にHPC分野で実績があり、機械学習用途で爆発的に普及







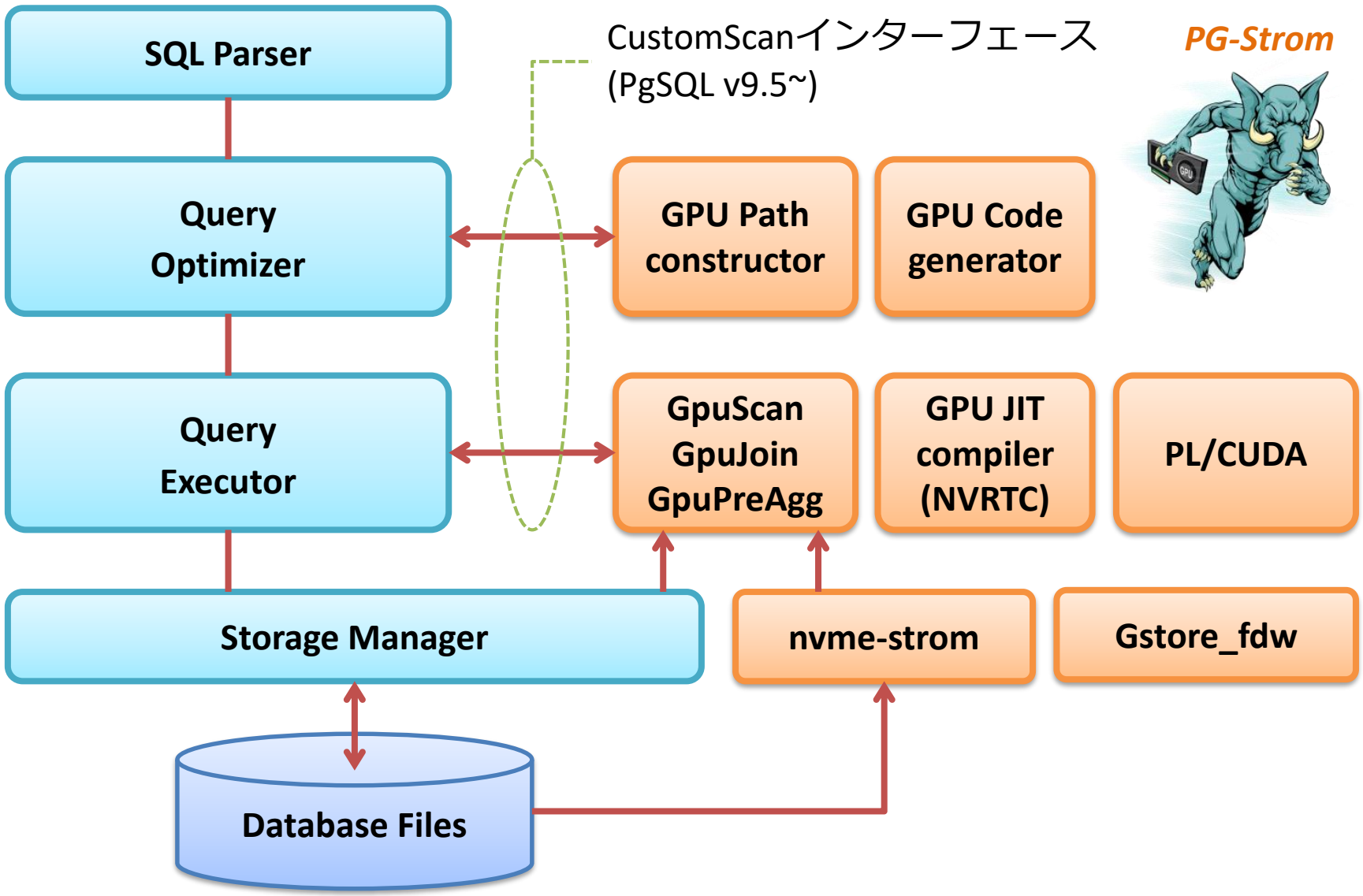


“**計算アクセラレータ**”のGPUで、どうやってI/Oを高速化するのか？

PostgreSQLはどうGPUを活用するのか？

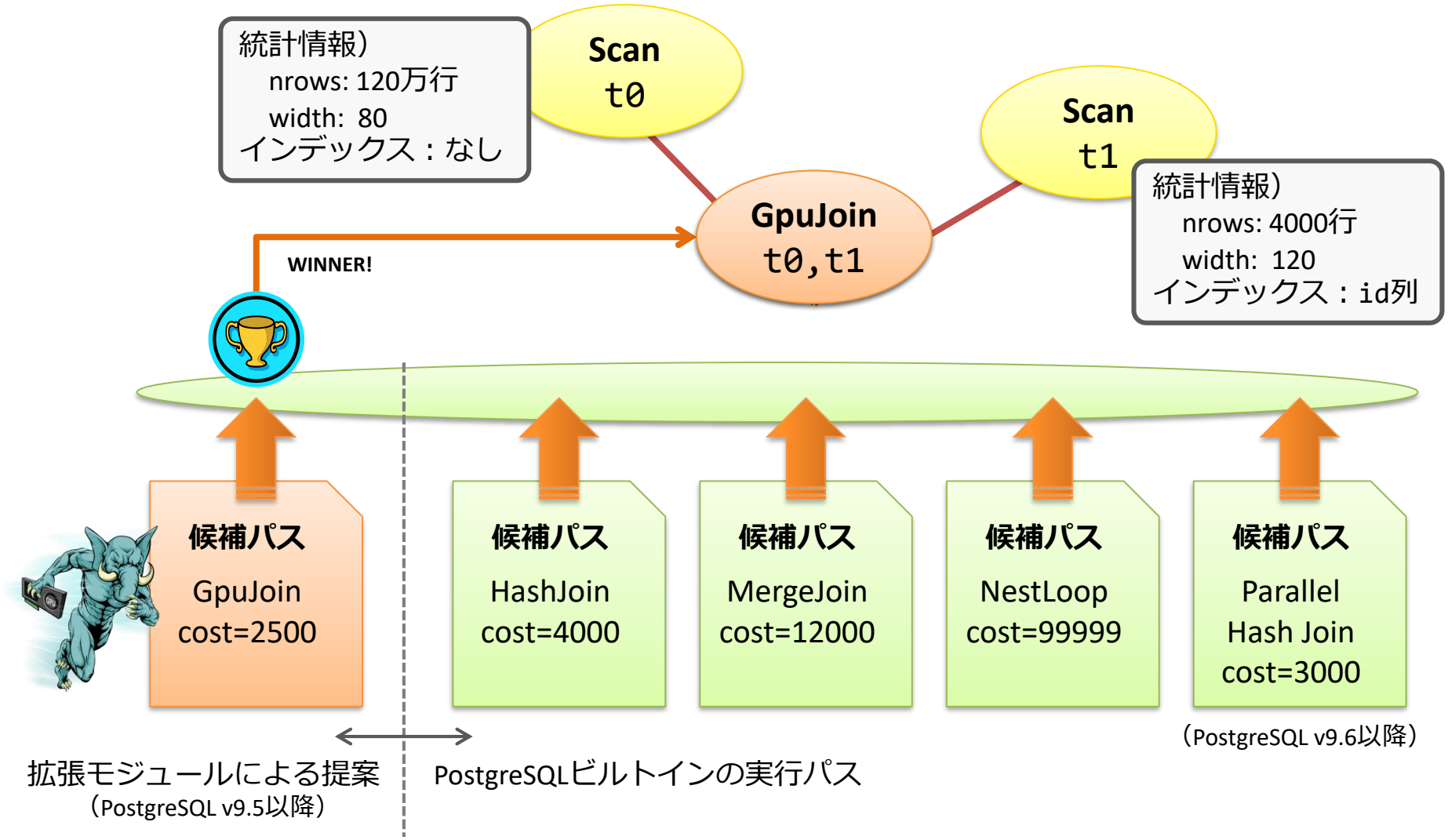
～PG-Stromのアーキテクチャ～

PostgreSQL ⇔ PG-Strom間の相互作用



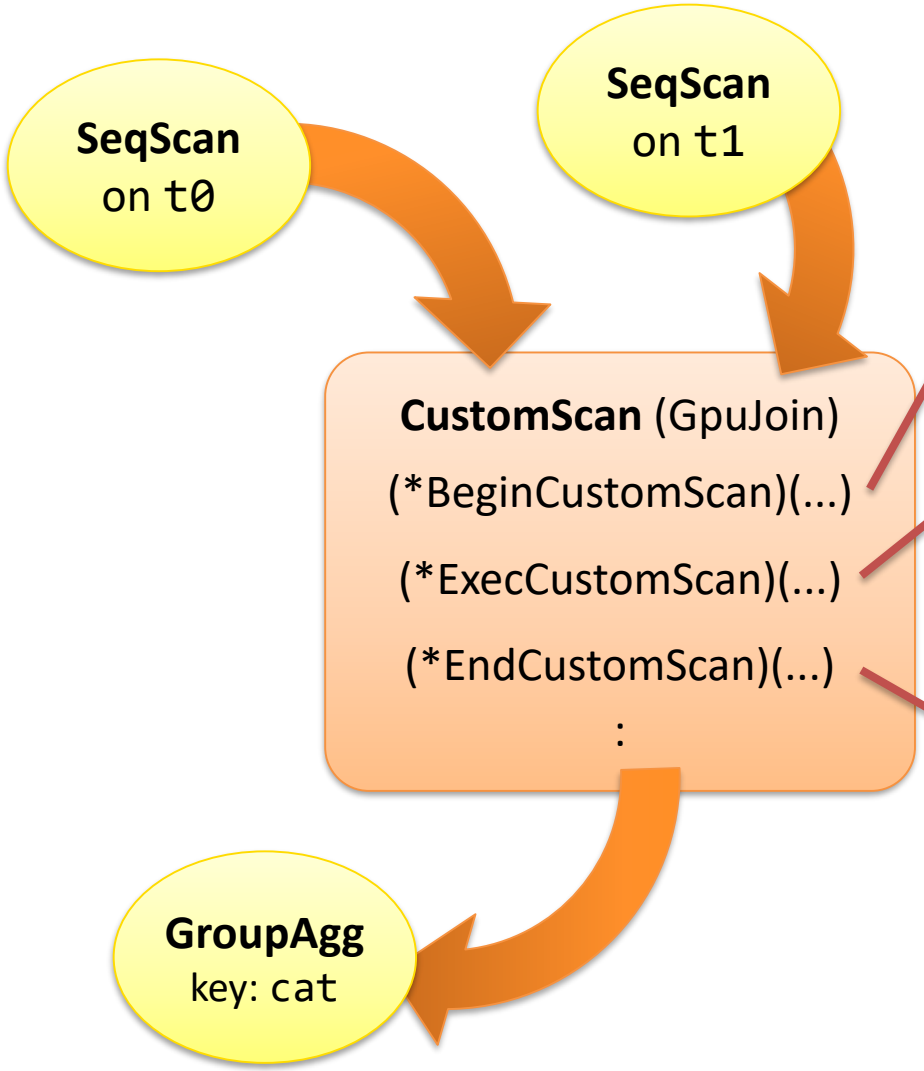
PostgreSQLはどのように実行計画を作るか

複数の処理アルゴリズムを競わせ、“コスト値”で評価



CustomScanによる介入

同じ結果を返しさえすれば、手段は問わない。



ExecInitGpuJoin(...)

- GPUコンテキストの初期化
- 自動生成したGPUプログラムの実行時コンパイル開始

ExecGpuJoin(...)

- 配下のt0,t1からデータを読み出し、DMAバッファにセット
- GPUへ非同期タスクをキック
- 実行が完了した非同期タスクを取り出し、結果をGroupAggへ渡す。

ExecEndGpuJoin(...)

- 非同期タスクの終了待ち（あれば）
- GPUリソースの解放

SQLからGPUコードを自動生成（WHERE句の例）

```
QUERY:  SELECT cat, count(*), avg(x) FROM t0
        WHERE x between y and y + 20.0 GROUP BY cat;
```

例) 条件句中の数値演算式を
CUDA命令列にオンデマンドで変換

```
        :
STATIC_FUNCTION(bool)
gpupreagg_qual_eval(kern_context *kcxt,
                    kern_data_store *kds,
                    size_t kds_index)
{                                   Reference to input data
  pg_float8_t KPARAM_1 = pg_float8_param(kcxt,1);
  pg_float8_t KVAR_3 = pg_float8_vref(kds,kcxt,2,kds_index);
  pg_float8_t KVAR_4 = pg_float8_vref(kds,kcxt,3,kds_index);

  return EVAL((pgfn_float8ge(kcxt, KVAR_3, KVAR_4) &&
               pgfn_float8le(kcxt, KVAR_3,
                 pgfn_float8pl(kcxt, KVAR_4, KPARAM_1))));
}       :
                        SQL expression in CUDA source code
```

Run-time
コンパイラ

Parallel
Execution

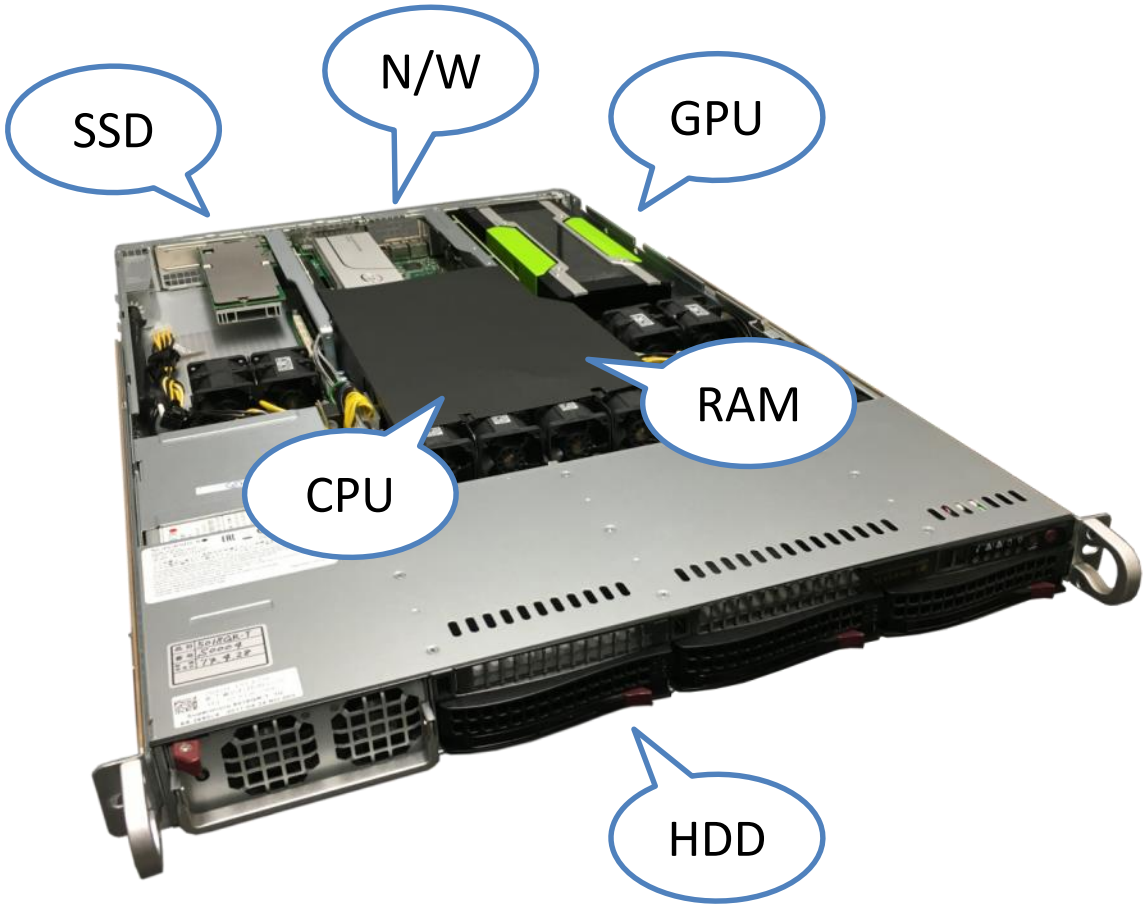
実行計画を確認する

```
postgres=# EXPLAIN ANALYZE
           SELECT cat,count(*),sum(ax) FROM tbl NATURAL JOIN t1 WHERE cid % 100 < 50 GROUP BY cat;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------
 GroupAggregate  (cost=203498.81..203501.80 rows=26 width=20)
                 (actual time=1511.622..1511.632 rows=26 loops=1)
   Group Key: tbl.cat
   ->  Sort  (cost=203498.81..203499.26 rows=182 width=20)
             (actual time=1511.612..1511.613 rows=26 loops=1)
         Sort Key: tbl.cat
         Sort Method: quicksort  Memory: 27kB
         ->  Custom Scan (GpuPreAgg)  (cost=203489.25..203491.98 rows=182 width=20)
                                      (actual time=1511.554..1511.562 rows=26 loops=1)
               Reduction: Local
               Combined GpuJoin: enabled
               ->  Custom Scan (GpuJoin) on tbl  (cost=13455.86..220069.26 rows=1797115 width=12)
                                                 (never executed)
                     Outer Scan: tbl  (cost=12729.55..264113.41 rows=6665208 width=8)
                                      (actual time=50.726..1101.414 rows=19995540 loops=1)
                     Outer Scan Filter: ((cid % 100) < 50)
                     Rows Removed by Outer Scan Filter: 10047462
                     Depth 1: GpuHashJoin  (plan nrows: 6665208...1797115,
                                            actual nrows: 9948078...2473997)
                              HashKeys: tbl.aid
                              JoinQuals: (tbl.aid = t1.aid)
                              KDS-Hash (size plan: 11.54MB, exec: 7125.12KB)
                     ->  Seq Scan on t1  (cost=0.00..2031.00 rows=100000 width=12)
                                         (actual time=0.016..15.407 rows=100000 loops=1)
 Planning Time: 0.721 ms
 Execution Time: 1595.815 ms
(19 rows)
```

GPUの役割を再定義する

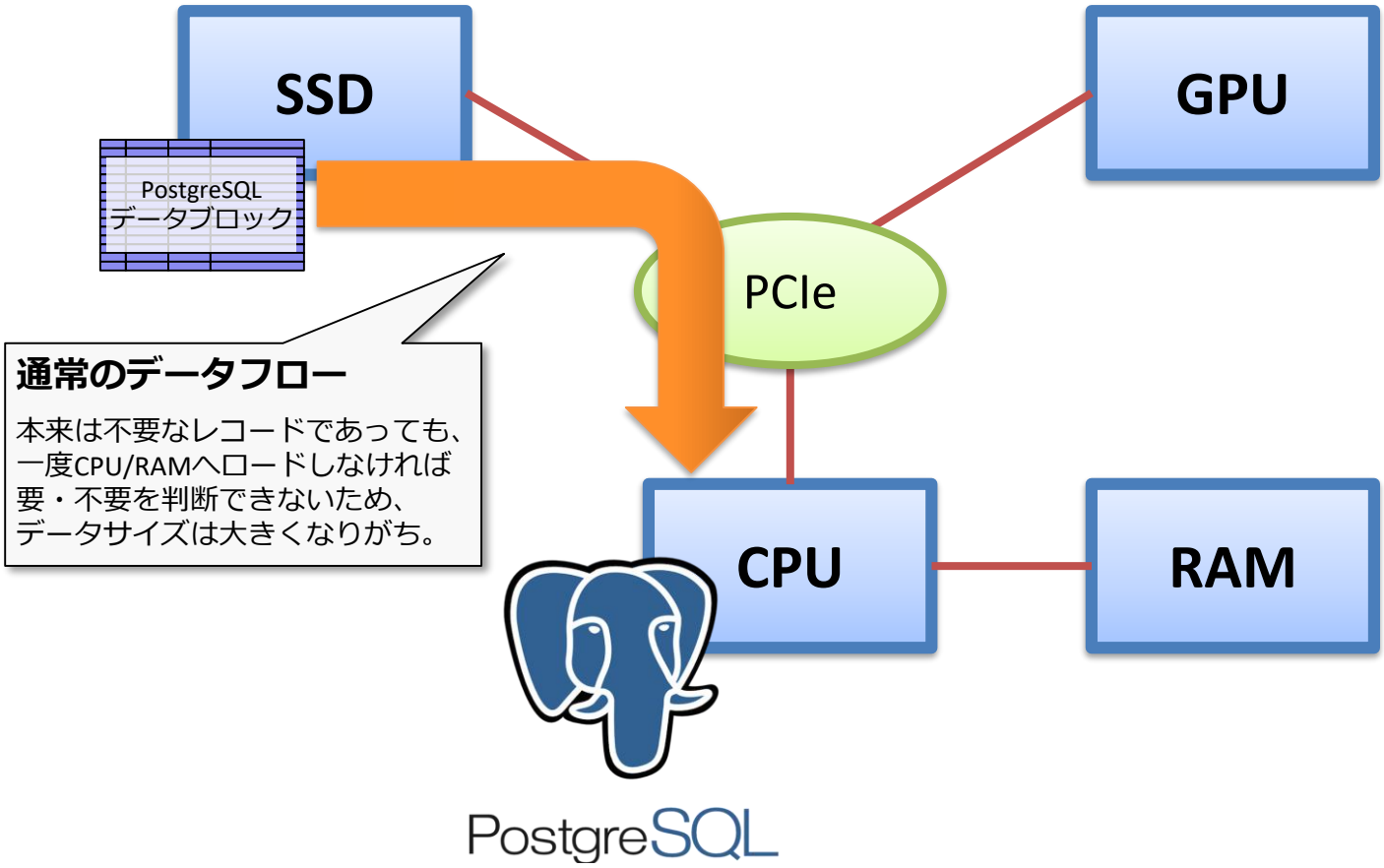
～どのようにI/Oを高速化するのか～

一般的な x86_64 サーバの構成



大量データを処理する時のデータの流れ

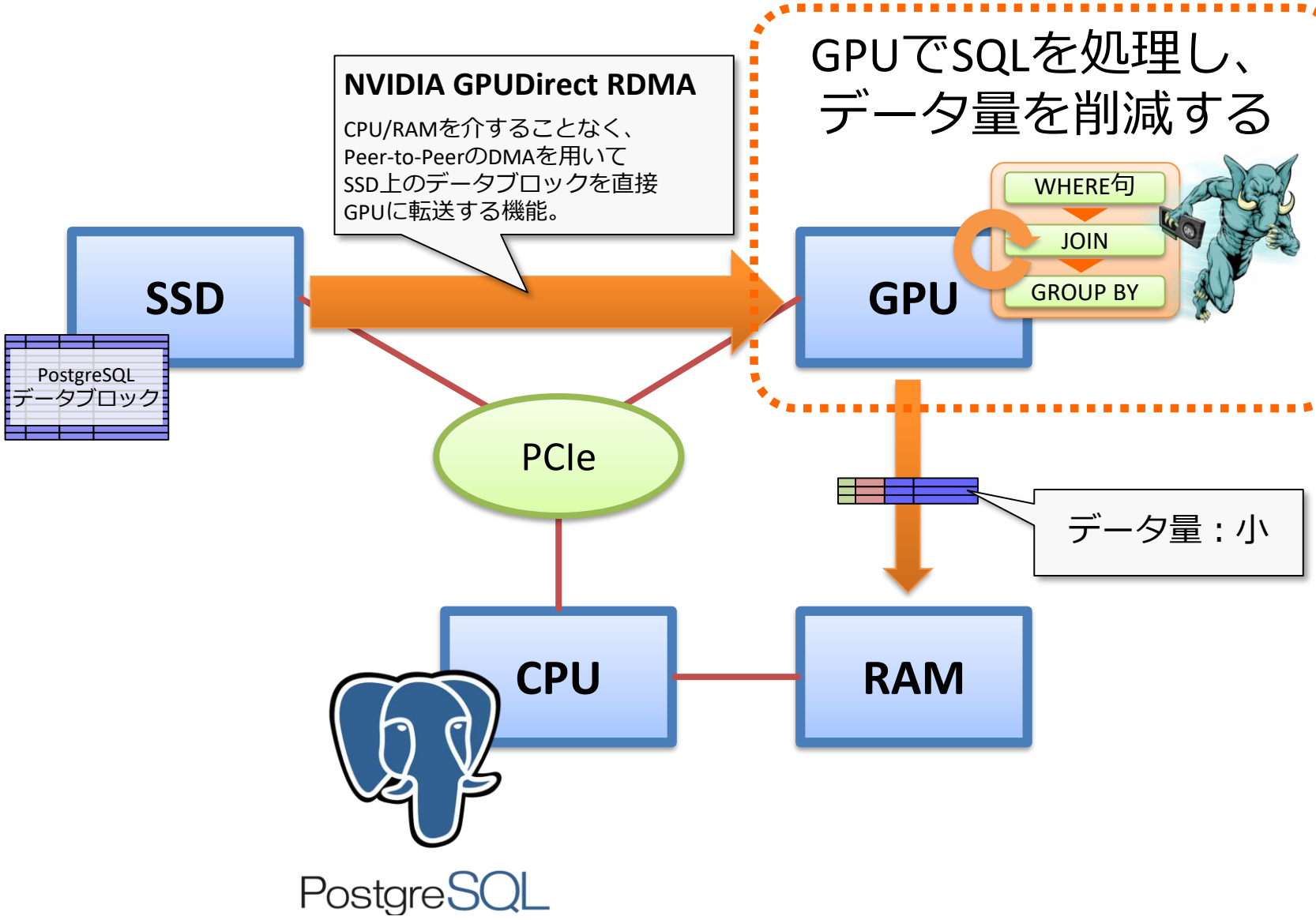
一度CPU/RAMへロードしなければ、
“ゴミデータ” であってもその要否を判断できない。



通常のデータフロー

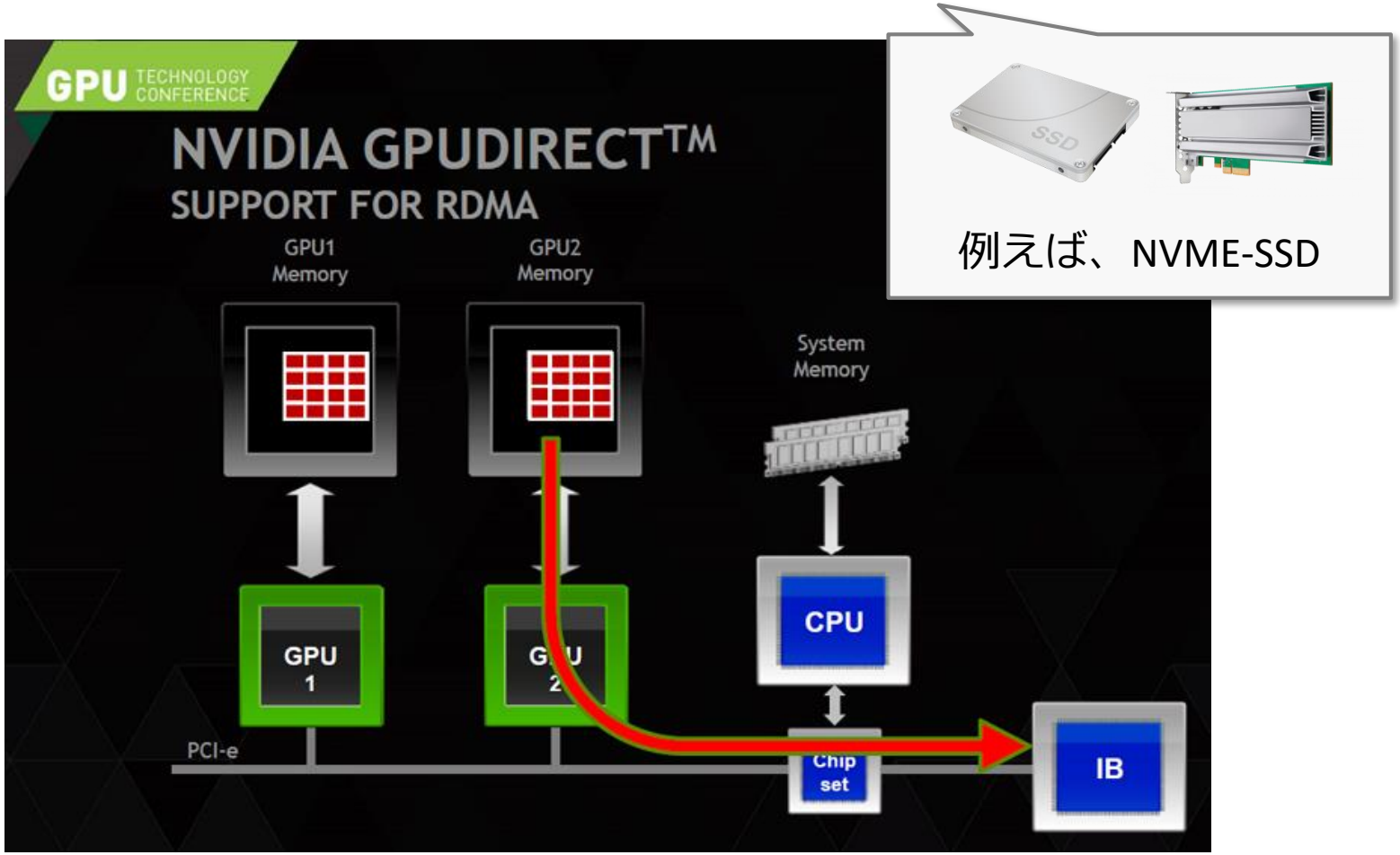
本来は不要なレコードであっても、
一度CPU/RAMへロードしなければ
要・不要を判断できないため、
データサイズは大きくなりがち。

中核機能：SSD-to-GPUダイレクトSQL

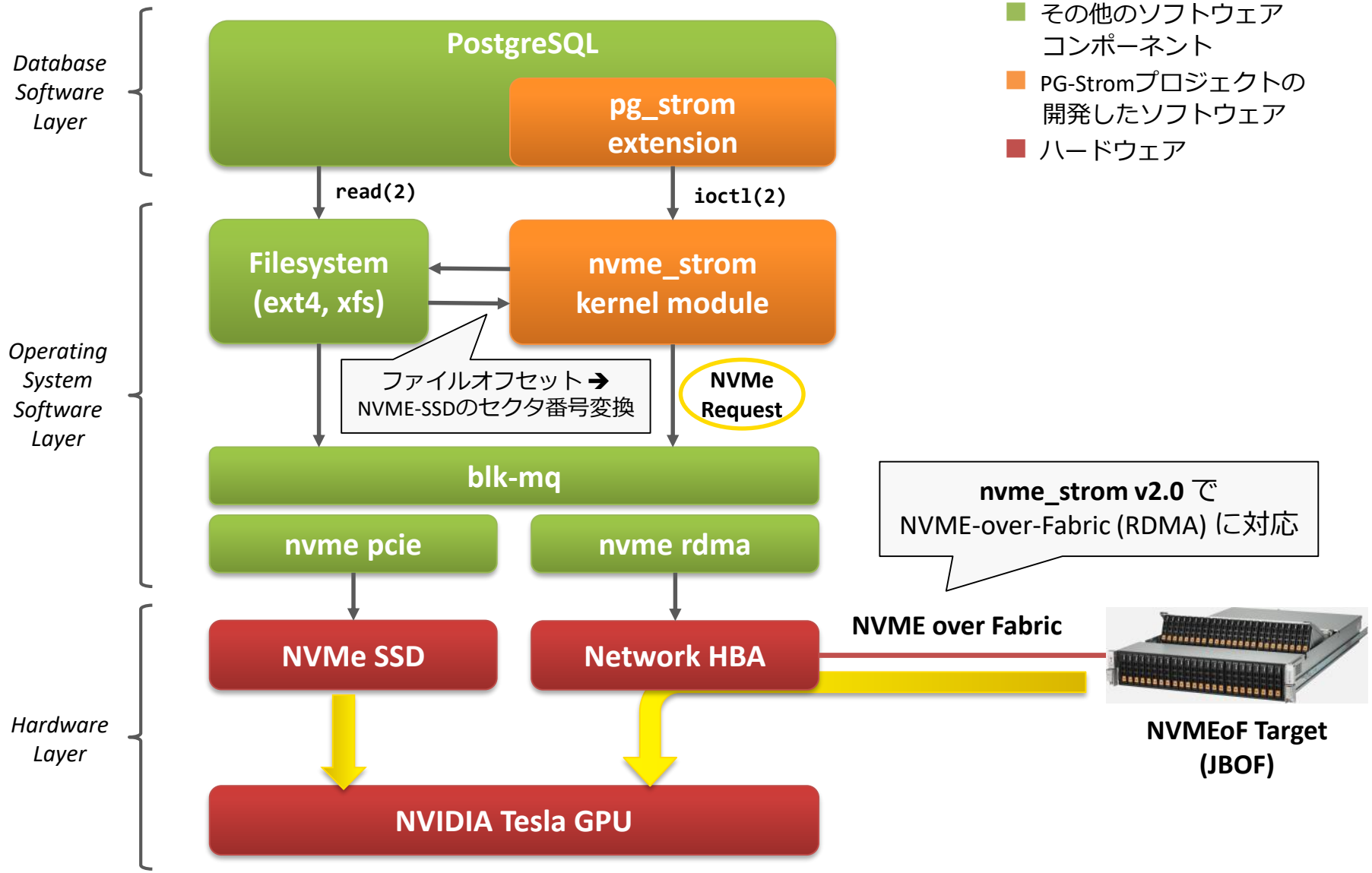


要素技術：GPUDirect RDMA

- CPUを介さず、GPU⇔他のPCIeデバイス間のP2Pデータ転送を行う機能
  - 元々は、マルチノード環境でInfinibandを介したGPU間のデータ転送効率化技術
  - Linux kernelドライバを作成する事で、**他の PCIe デバイス**にも対応可能



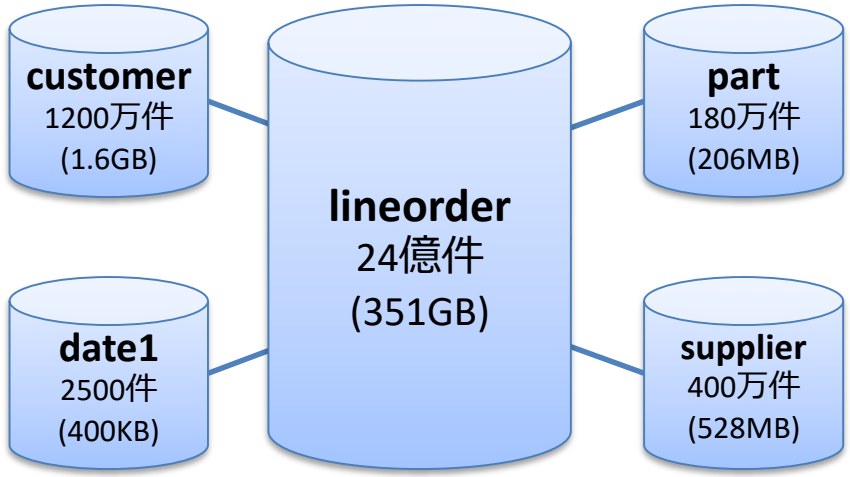
SSD-to-GPU Direct SQL：ソフトウェアスタック



ベンチマーク（1/2） – システム構成とワークロード

典型的なスタースキーマに対し、JOIN,GROUP BYを含む集計クエリを実行

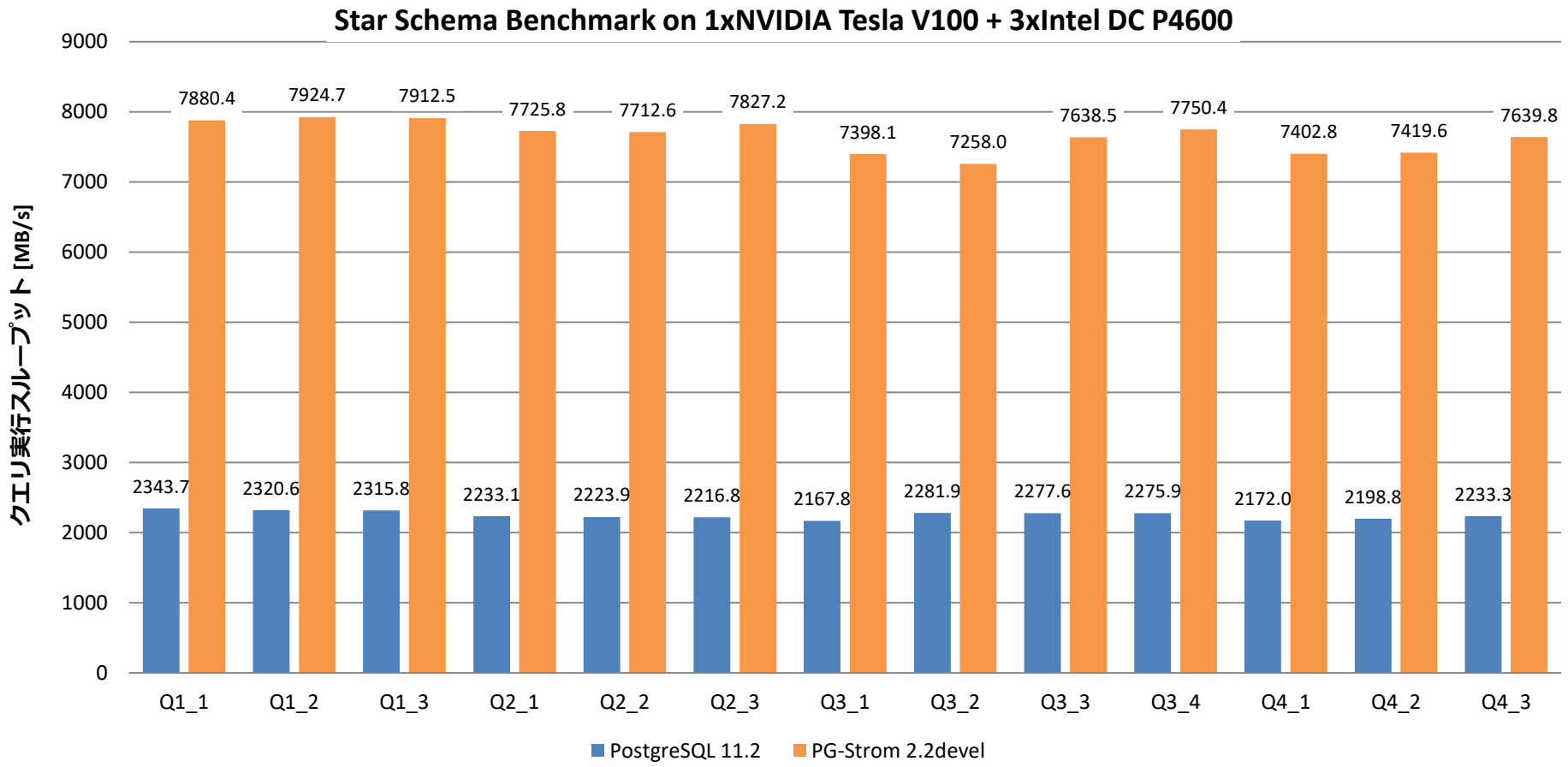
Supermicro SYS-1019GP-TT	
CPU	Xeon Gold 6126T (2.6GHz, 12C) x1
RAM	192GB (32GB DDR4-2666 x 6)
GPU	NVIDIA Tesla V100 (5120C, 16GB) x1
SSD	Intel SSD DC P4600 (HHHL; 2.0TB) x3 （md-raid0によるストライピング構成）
HDD	2.0TB(SATA; 72krpm) x6
Network	10Gb Ethernet 2ports
OS	Red Hat Enterprise Linux 7.6 CUDA 10.1 + NVIDIA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



■ クエリ例（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;
```

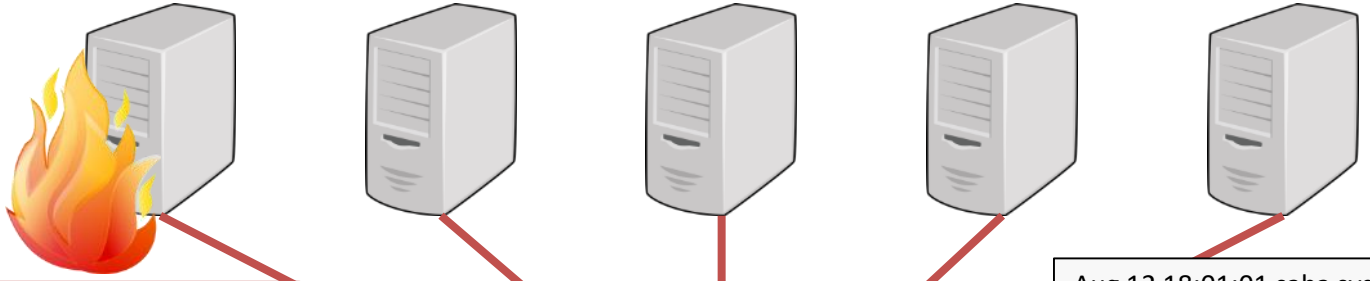
ベンチマーク（2/2） – 測定結果



- クエリ実行スループット = 353GB (DBサイズ) ÷ クエリ応答時間
- PostgreSQLの2.2～2.3GB/sに対して、SSD-to-GPUダイレクトSQLの効果により7.2～7.9GB/s（3倍強）の処理性能を発揮

➔ 従来はDWH専用機や小規模クラスタを用いていた水準の能力を１台のマシンで。

適用例：ログ集積・解析システム



セキュリティ事故

Aug 12 18:01:01 saba systemd: Started Session 21060 of user root.

GPU+NVME-SSD搭載
データベースサーバ

従来システム40分の
クエリを30秒で実行

被害状況把握

影響範囲特定

責任者への報告

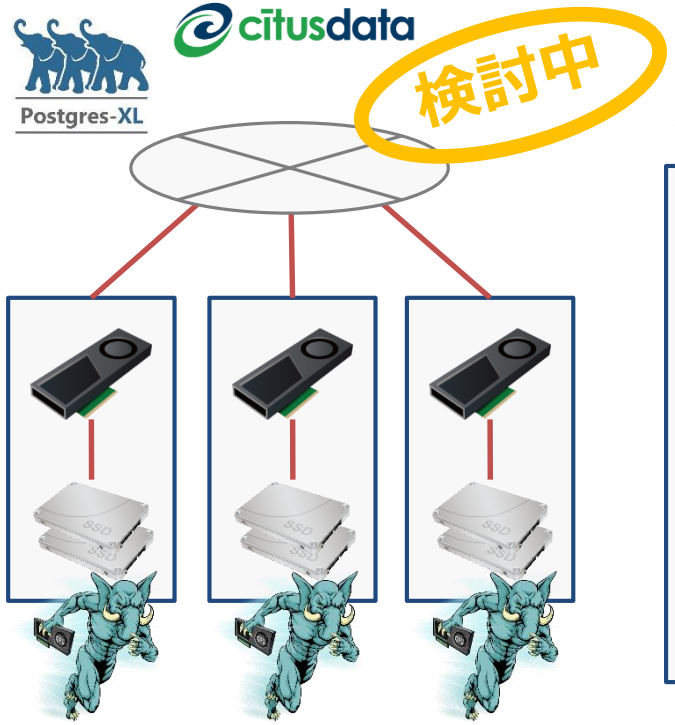
SQLによる検索

- **高速**なトライ＆エラー
- **使い慣れた**インターフェース
- **行データのまま**で高速な検索

More faster, beyond the limitation

更なる高速化に向けたアプローチ

スケールアウト

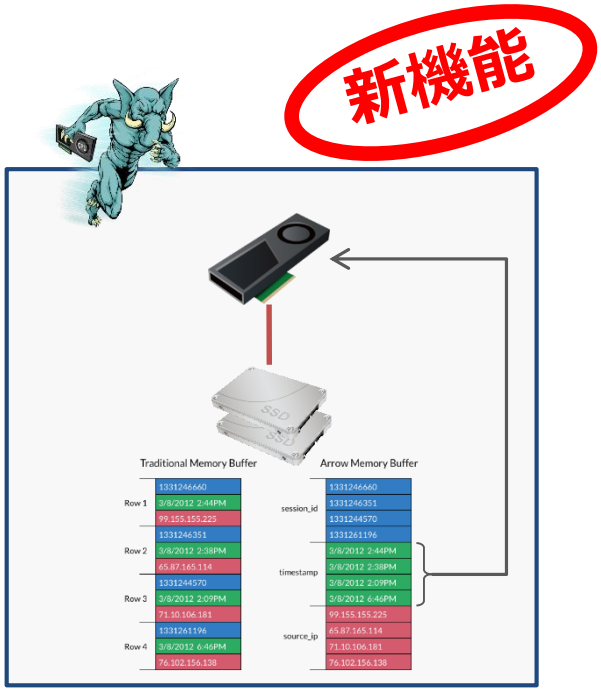


複数台のPostgreSQLノードで処理を分散させる。
各ノードにおける処理はシングルノードと同様に、GPUとSSDによる高速化が可能。

スケールアップ

GPUやSSDの台数を増やし、一台あたり並列処理能力を増強する。
スケールアウトと異なり、コーディネーターを必要としない。

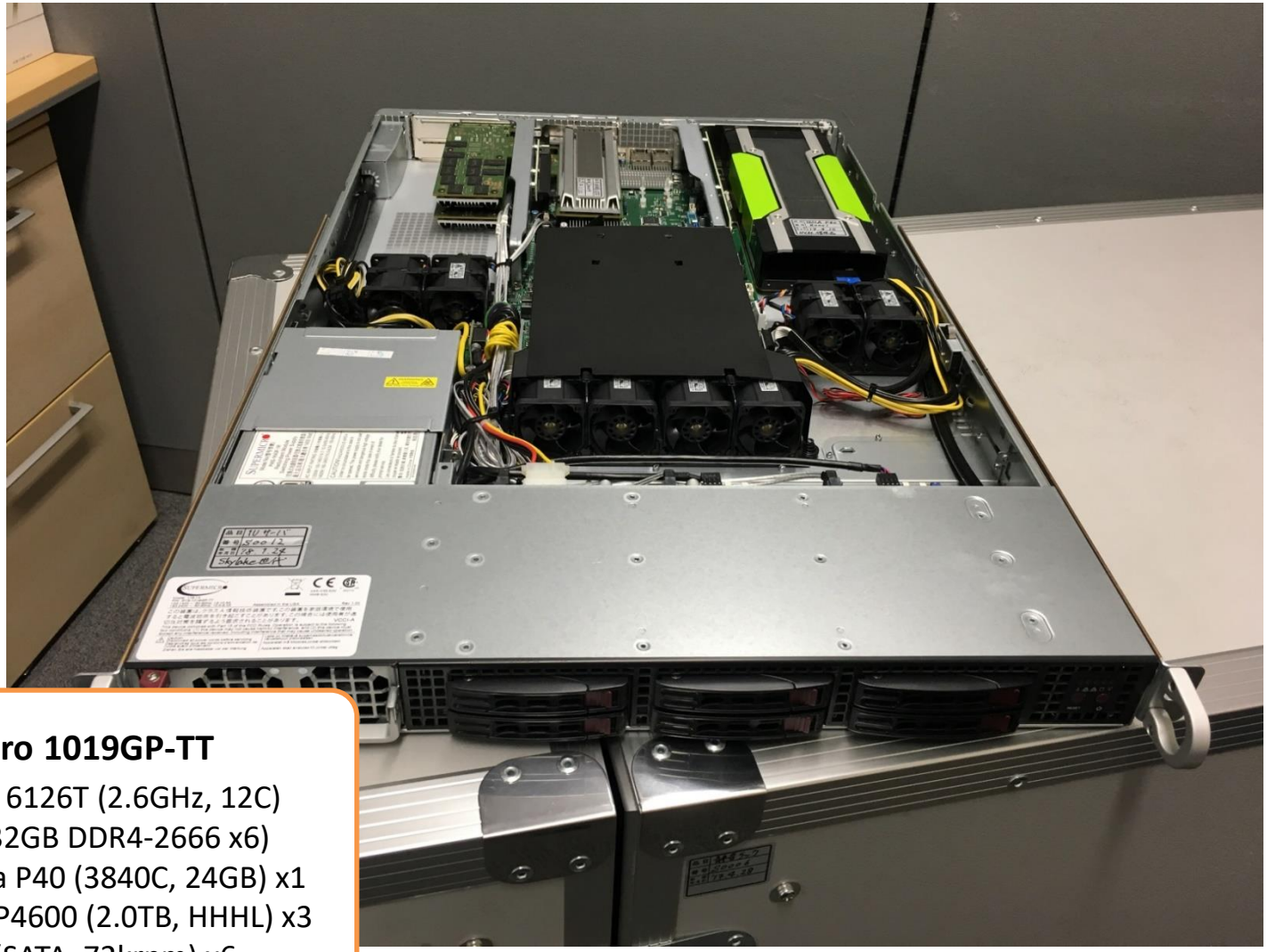
列指向データ



ストレージ上のデータ形式を集計・解析系処理に適した列形式に変更し、SSDから読み出すべきI/O量を削減する。

スケールアップによる性能強化

ハードウェア構成に関する検討（1/3）

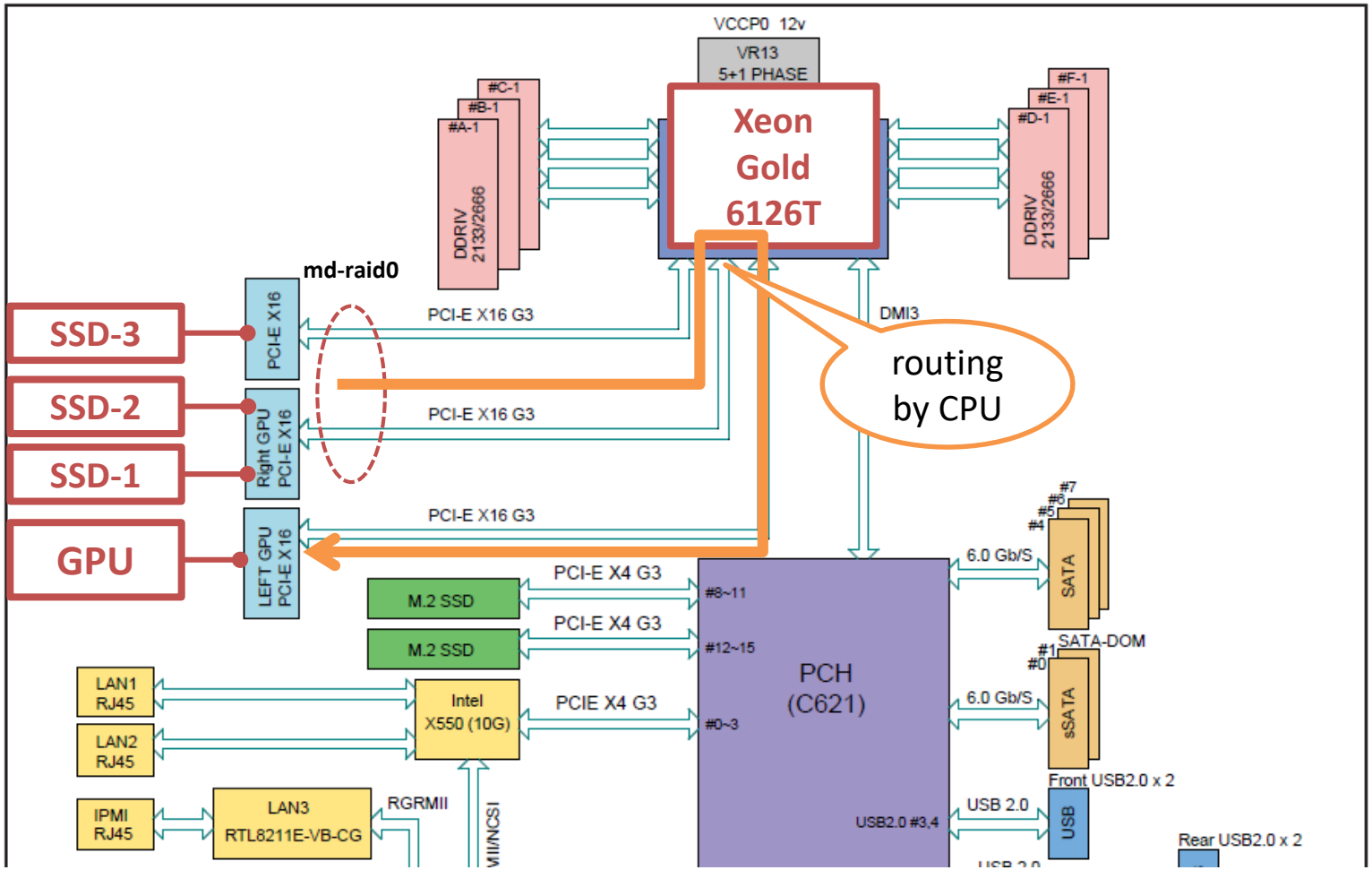


Supermicro 1019GP-TT

CPU: Xeon Gold 6126T (2.6GHz, 12C)
RAM: 192GB (32GB DDR4-2666 x6)
GPU: NVIDIA Tesla P40 (3840C, 24GB) x1
SSD: Intel SSD DC P4600 (2.0TB, HHHL) x3
HDD: 2.0TB (SATA, 72krpm) x6
N/W: 10Gb ethernet x2ports

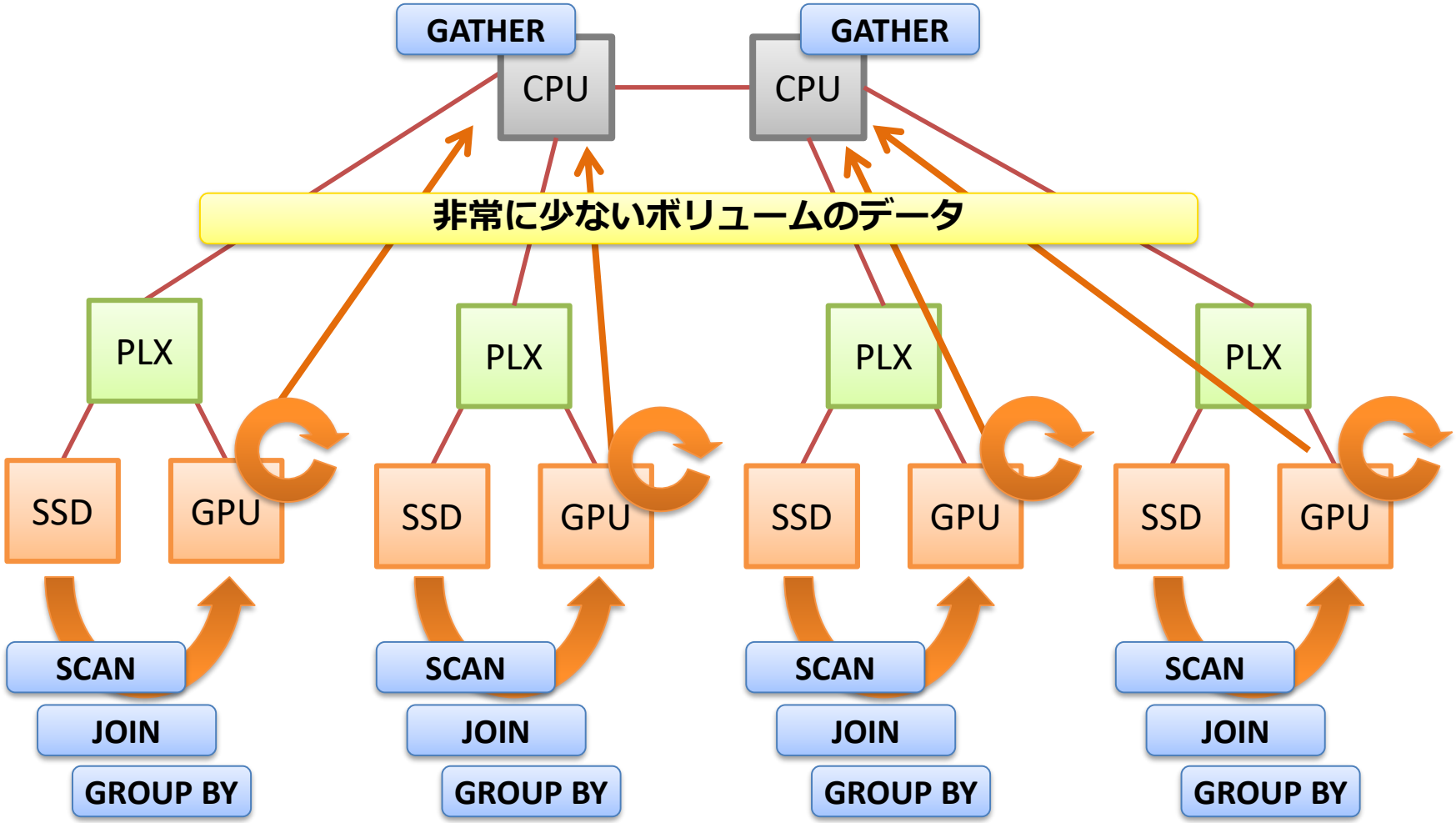
ハードウェア構成に関する検討（2/3）

PCIeバスを制御するCPUがデータ転送のボトルネックになる



ハードウェア構成に関する検討（3/3）

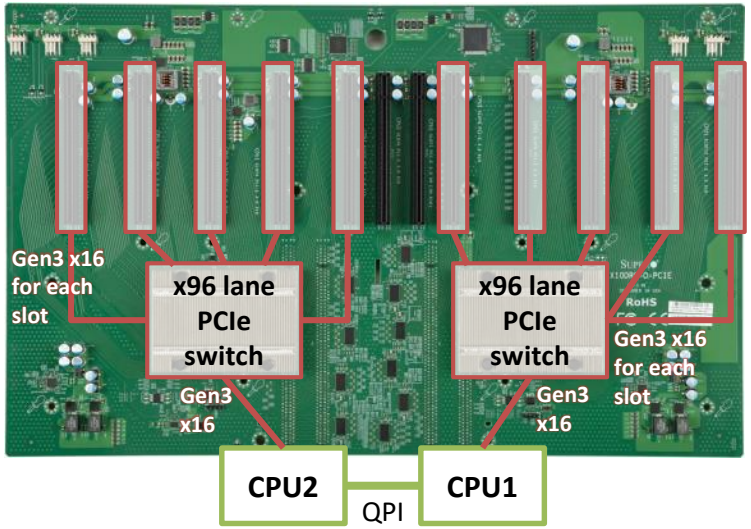
PCIeスイッチがI/Oをバイパスする事でCPUの負荷を軽減できる



マルチGPU・パーティション対応（1/3）

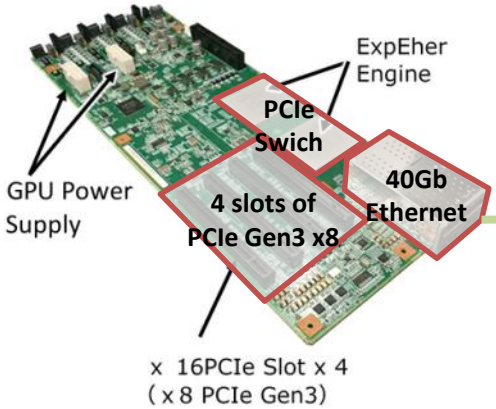
HPCサーバ – デバイス間RDMAに最適化したサーバ製品が販売されている

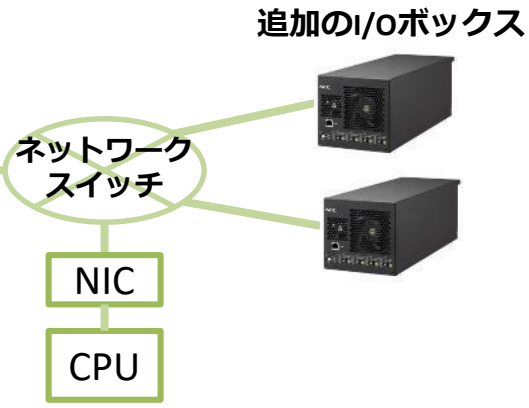
Supermicro社
SYS-4029TRT2



I/O拡張ボックス – 本来はデバイスの増設を目的とした製品だが…。

NEC ExpEther 40G
(4slots版)





マルチGPU・パーティション対応（2/3）

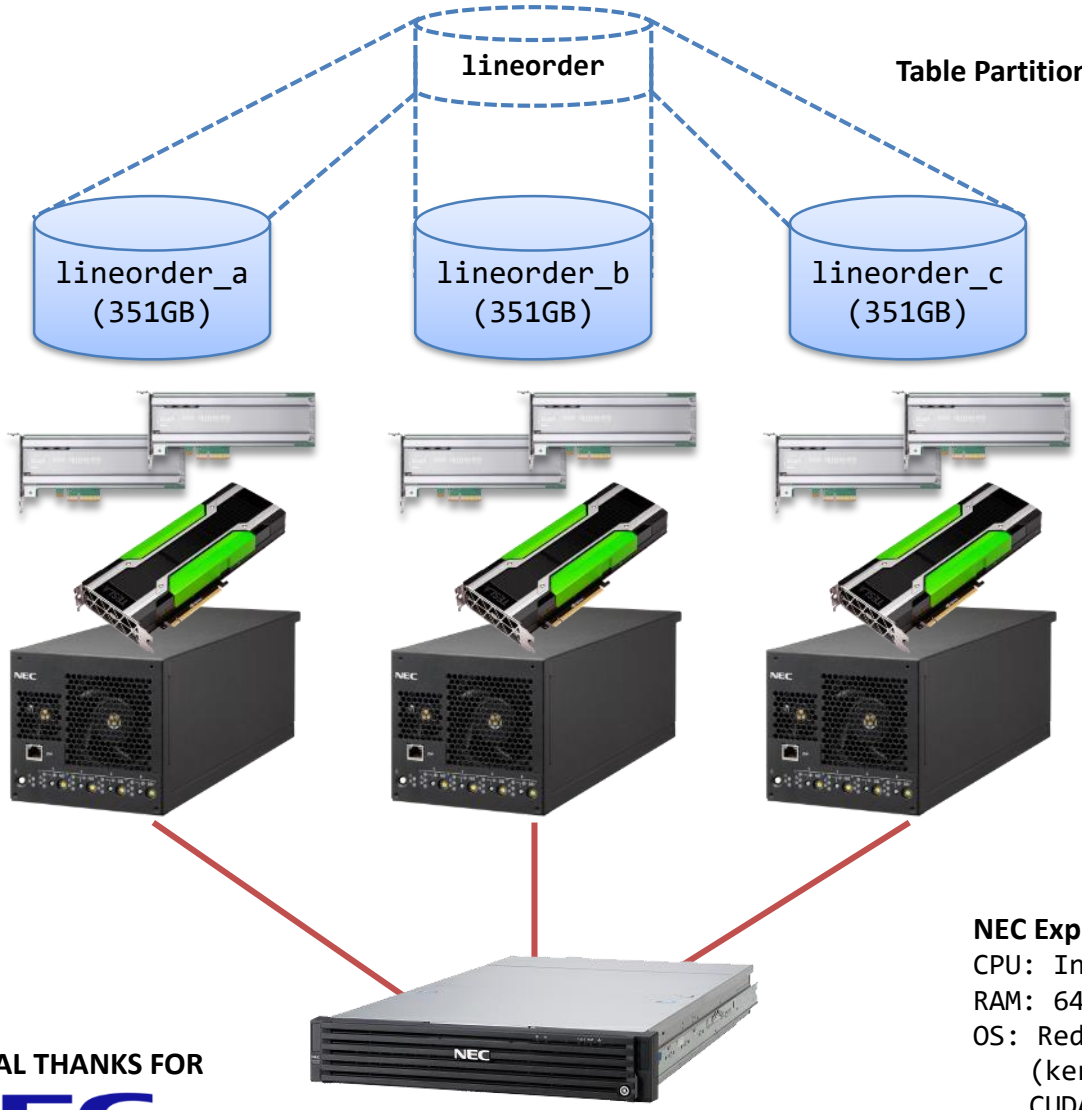


Table Partition by Hash

Intel DC P4600 (2.0TB; HHHL)
SeqRead: 3200MB/s
SeqWrite: 1575MB/s
I/F: PCIe 3.0 x4

NVIDIA Tesla P40
# of cores: 3840 (1.3GHz)
Device RAM: 24GB (347GB/s, GDDR5)
CC: 6.1 (Pascal, GP104)
I/F: PCIe 3.0 x16

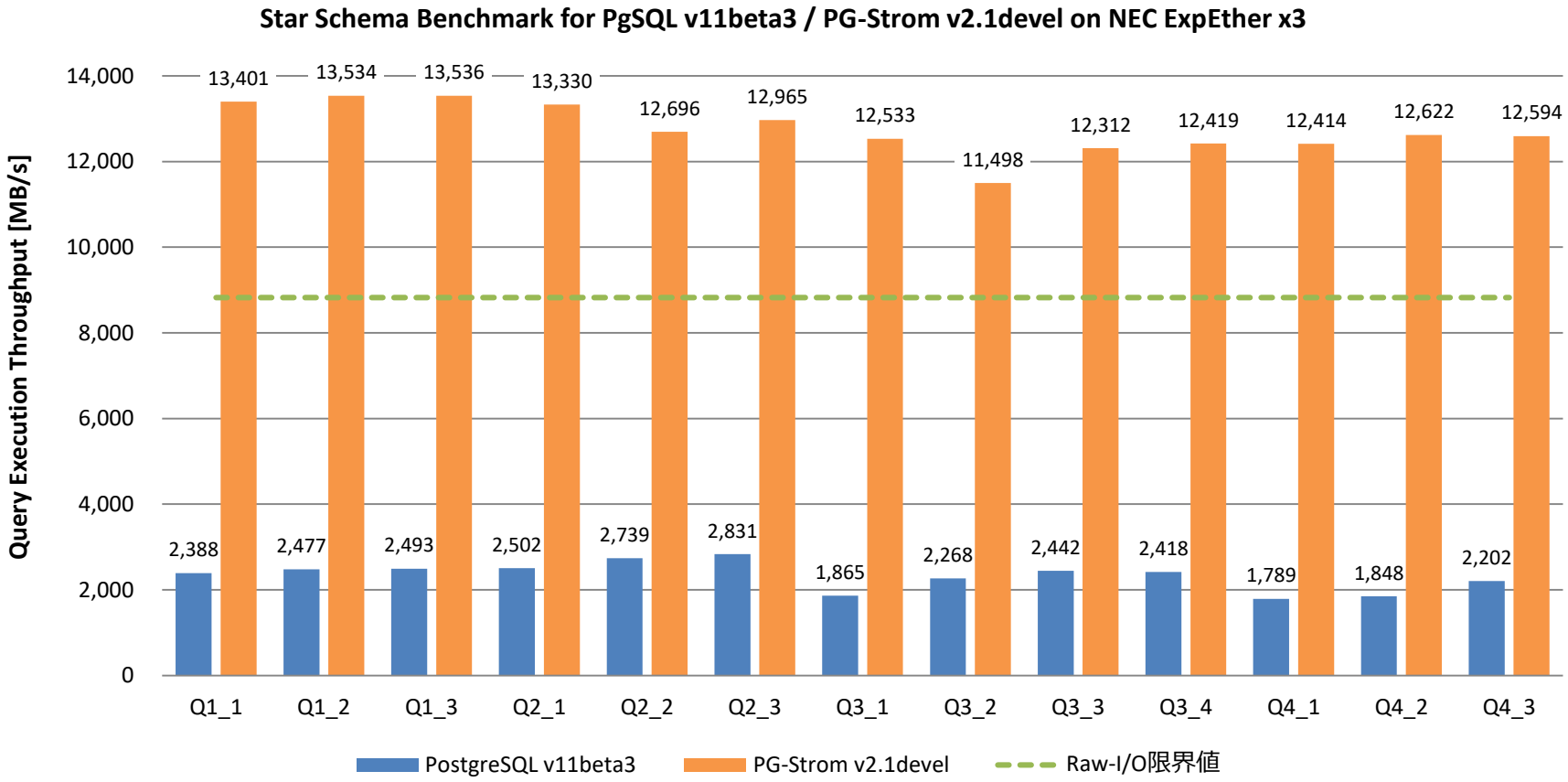
NEC ExpEther (40Gb; 4slots版)
I/F: PCIe 3.0 x8 (x16幅) x4スロット
+ internal PCIe switch
N/W: 40Gb-ethernet

NEC Express5800/R120g-2m
CPU: Intel Xeon E5-2603 v4 (6C, 1.7GHz)
RAM: 64GB
OS: Red Hat Enterprise Linux 7
(kernel: 3.10.0-862.9.1.el7.x86_64)
CUDA-9.2.148 + driver 396.44
DB: PostgreSQL 11beta3 + PG-Strom v2.1devel

SPECIAL THANKS FOR
NEC

マルチGPU・パーティション対応（3/3）- ベンチマーク

I/O拡張ボックス3台構成で、 max 13.5GB/s のクエリ処理速度を実現

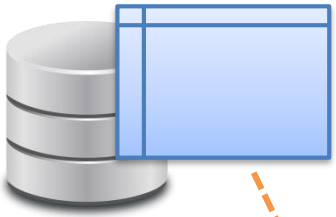


- I/O拡張ボックス毎に351GB、計1055GBのデータベースを構築し、13種類のSSBMクエリを実行
- SQL実行を伴わない SSD→Host への Raw-I/O データ転送は 9GB/s 弱が限界。
  つまり、Raw-I/Oのストレージ読出しよりも、SQLを高速に実行できている。

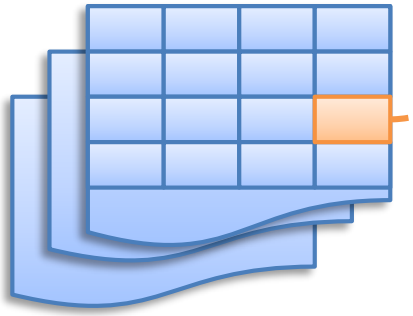
列指向データによる性能強化

データ構造を読み解く（1/2）- PostgreSQL（行データ）

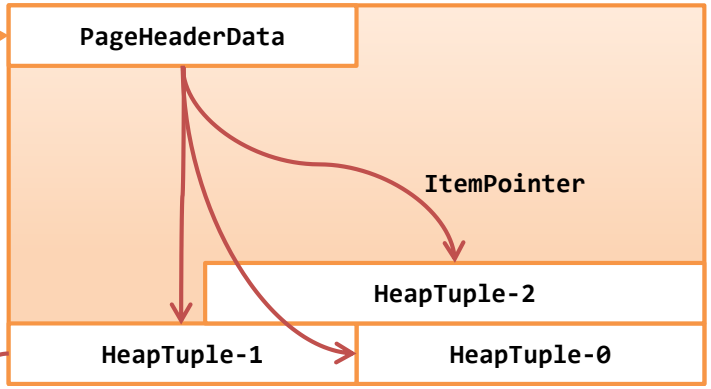
テーブル



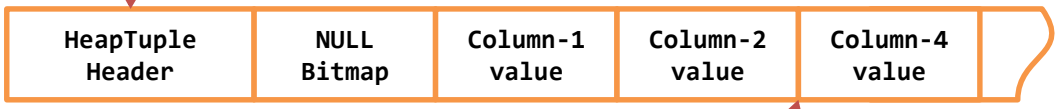
セグメント
1GB単位に分割
されたファイル



ブロック
通常8KBの固定長領域で、
PostgreSQLにおけるI/O単位



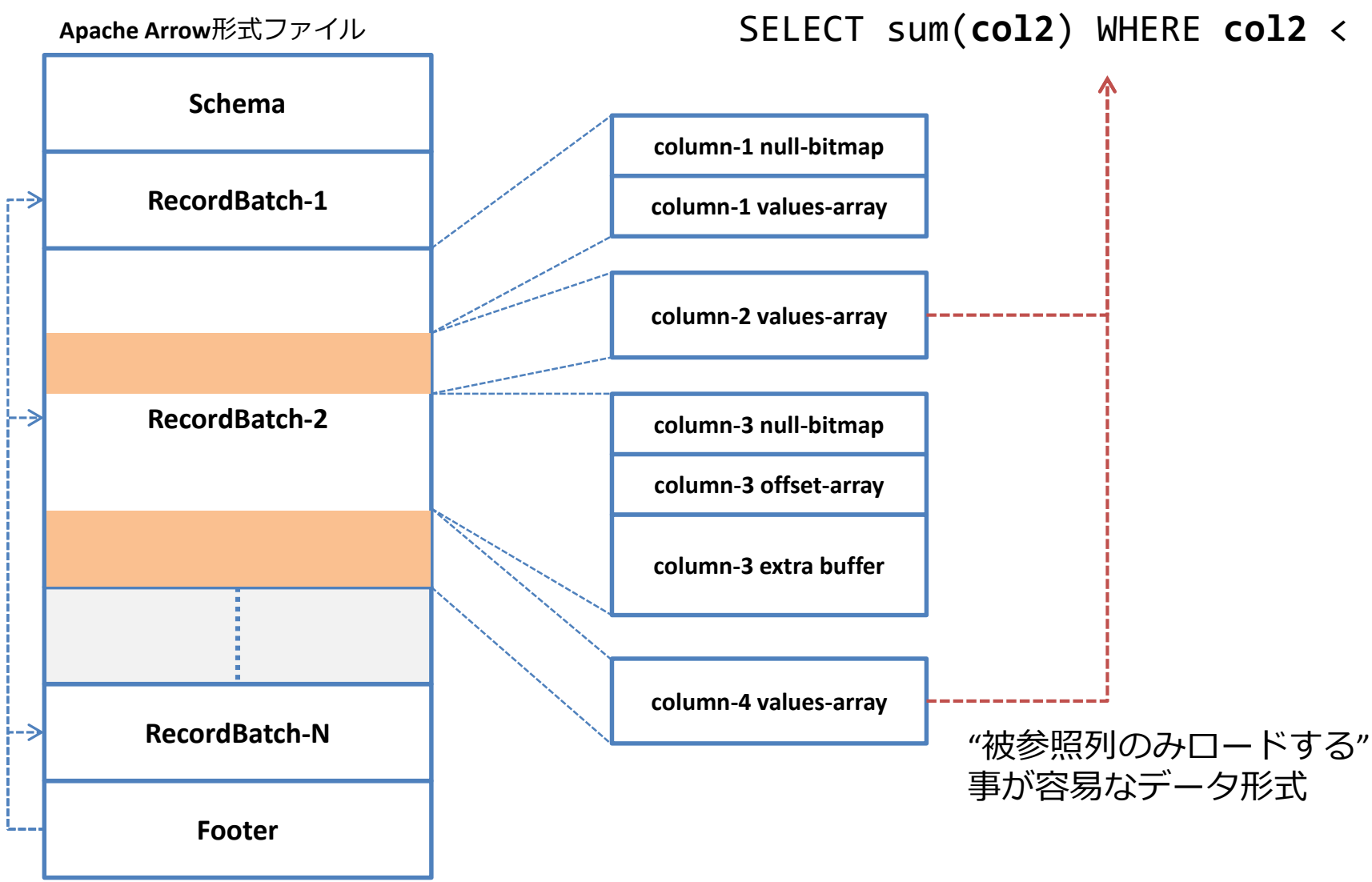
タプル
PostgreSQLにおけるストレージ上の行



- MVCC可視性制御情報
- 各種のフラグ情報
- 含まれる列の個数
- オブジェクトID（≦PG11）

NULL値や可変長データがあると、
常に一定の場所にデータが置かれる
わけではない。

データ構造を読み解く（2/2）- Apache Arrow（列データ）

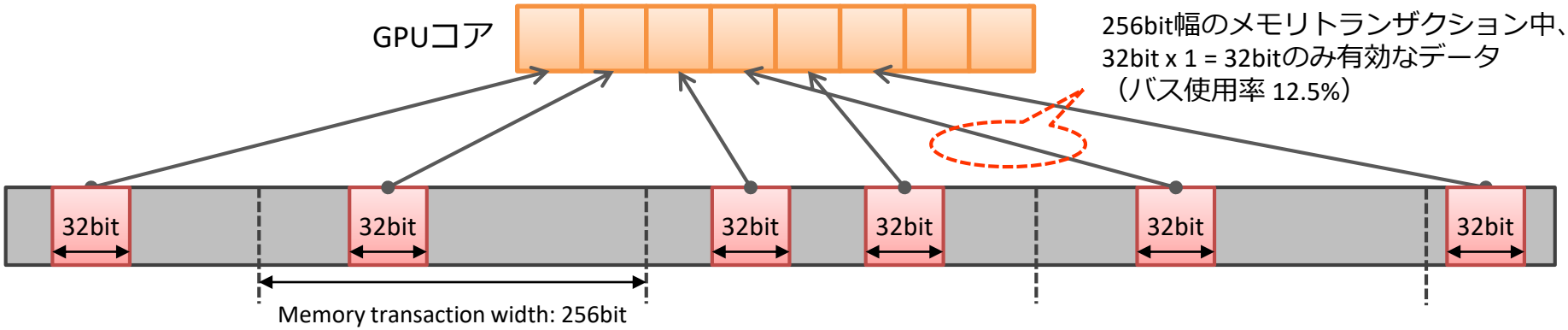


《補足》なぜ列指向データがGPUに適するのか？

列データ構造は、GPUのメモリバス使用率を最大化する

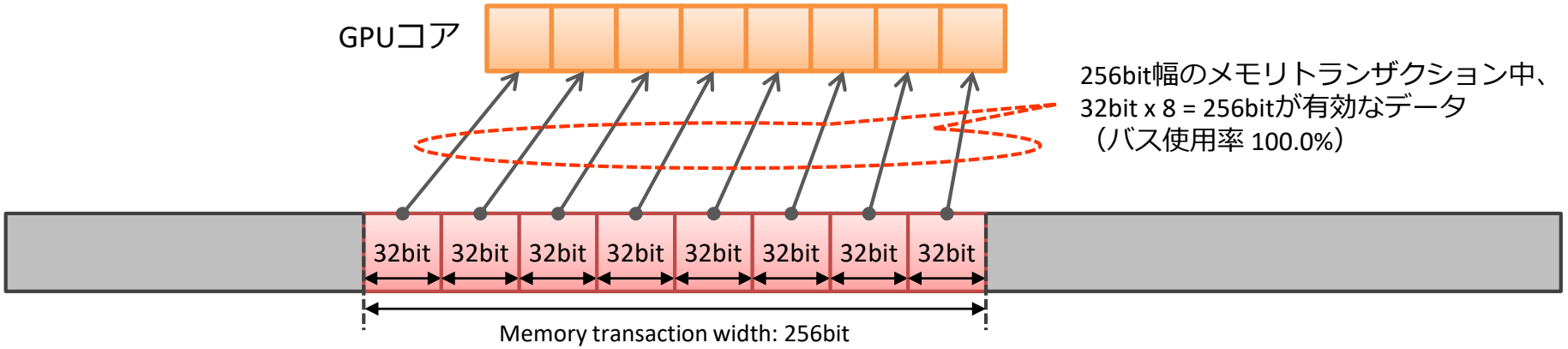
行データ形式 – 不連続なデータアクセス (*random memory access*)

➔ メモリトランザクションの回数が増え、データバスの使用率が低下



列データ形式 – 隣接領域に対するデータアクセス (*coalesced memory access*)

➔ 最小限のメモリトランザクション回数、データバスの使用率を最大化

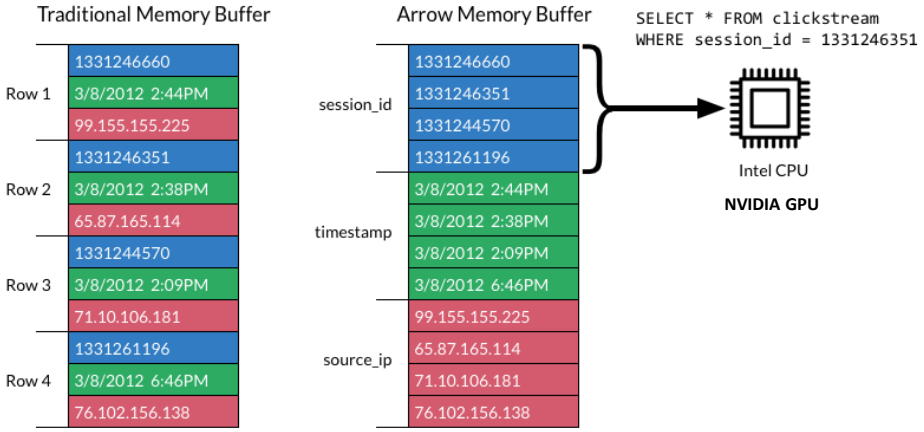


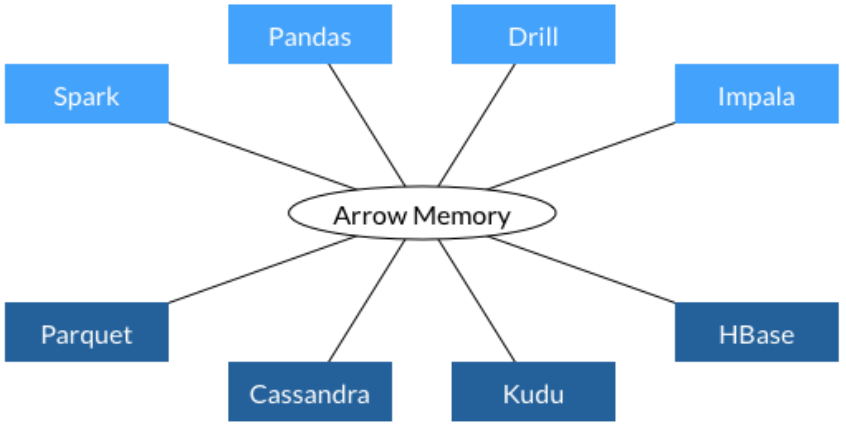
Apache Arrowとは（1/2）

特徴

- 列指向で分析用途向けに設計された**データ形式**
- アプリケーションによらず、**共通のデータ交換形式**として利用可能
- 整数、実数、日付時刻、文字列など基本的なデータ型を定義

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





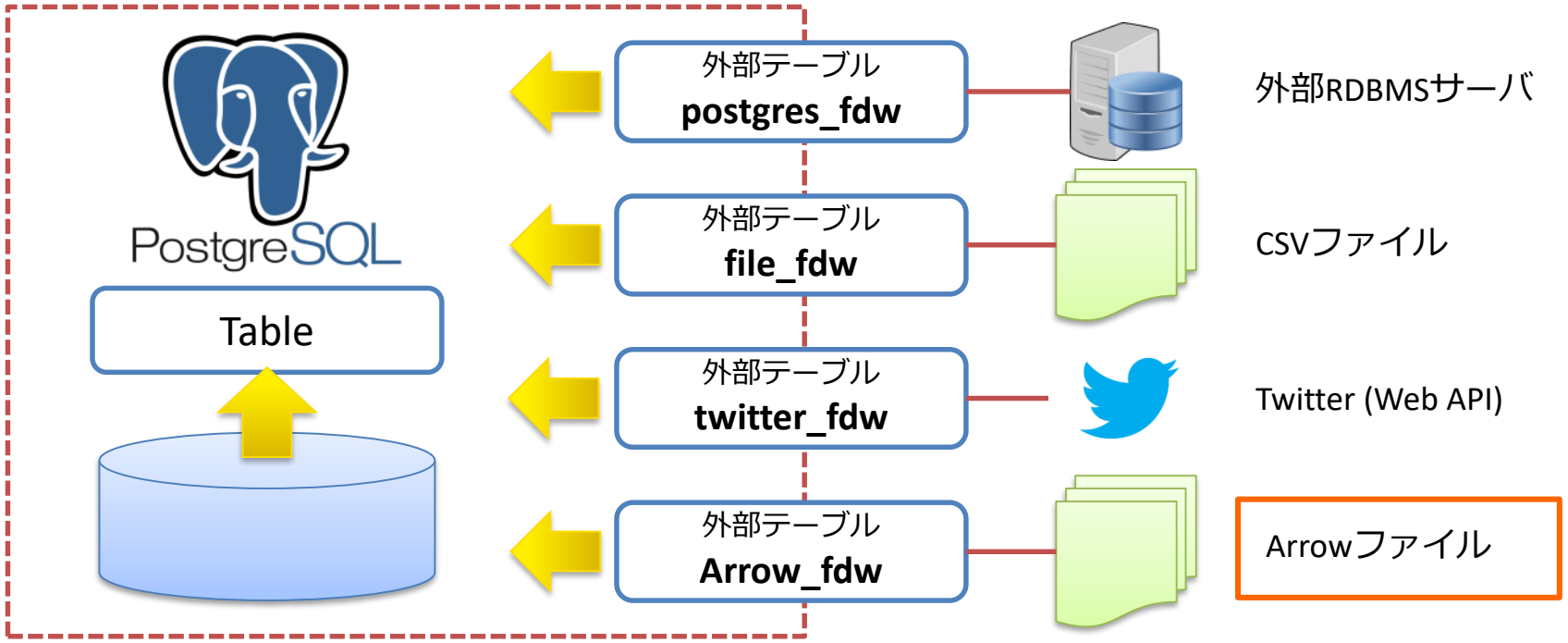
Apache Arrowとは（2/2） – データ型の対応

Apache Arrow⇔PostgreSQL間でデータ型の相互変換は可能

Apache Arrowデータ型	PostgreSQLデータ型	備考
Int	int2, int4, int8	
FloatingPoint	float2, float4, float8	float2はPG-Stromによる独自拡張
Binary	bytea	
Utf8	text	
Bool	bool	
Decimal	numeric	
Date	date	unitsz = Day に補正
Time	time	unitsz = MicroSecondに補正
Timestamp	timestamp	unitsz = MicroSecondに補正
Interval	interval	
List	配列型（array）	PostgreSQLでは行ごとに異なる次元数を指定可
Struct	複合型（composite）	
Union	------	
FixedSizeBinary	char(n)	
FixedSizeList	array型?	
Map	------	

FDW (Foreign Data Wrapper) 機能とは

外部テーブル（Foreign Table） – PostgreSQL管理外のデータソースを、
あたかもテーブルであるかのように取り扱うための機能

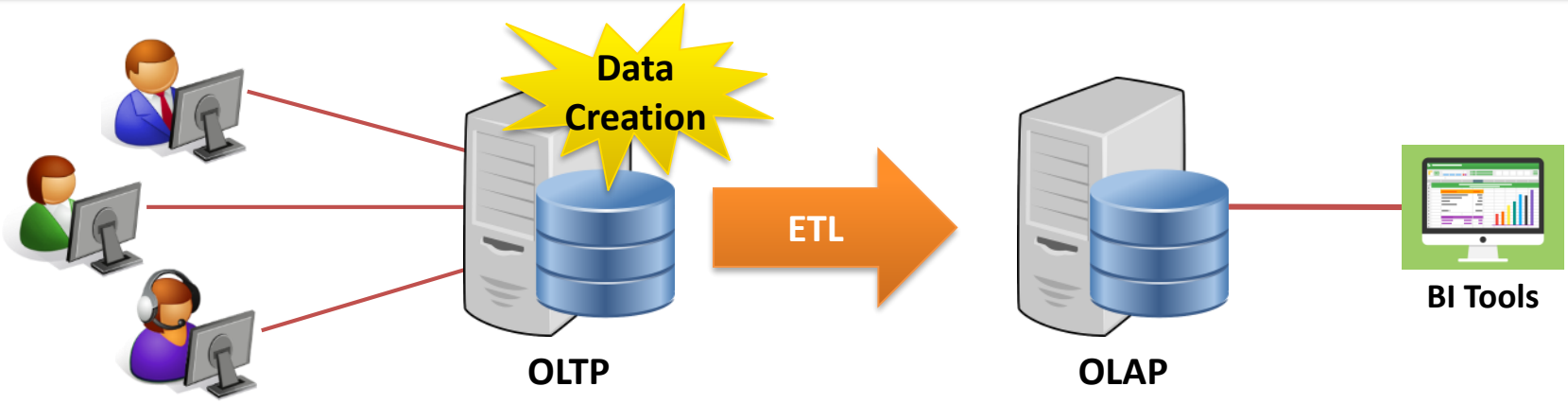


- FDWモジュールは、外部データ⇔PostgreSQLの相互変換に責任を持つ。
- Arrow_Fdwの場合、ファイルシステム上の Arrow 形式ファイルを Foreign Table という形で参照できるようにマップする。

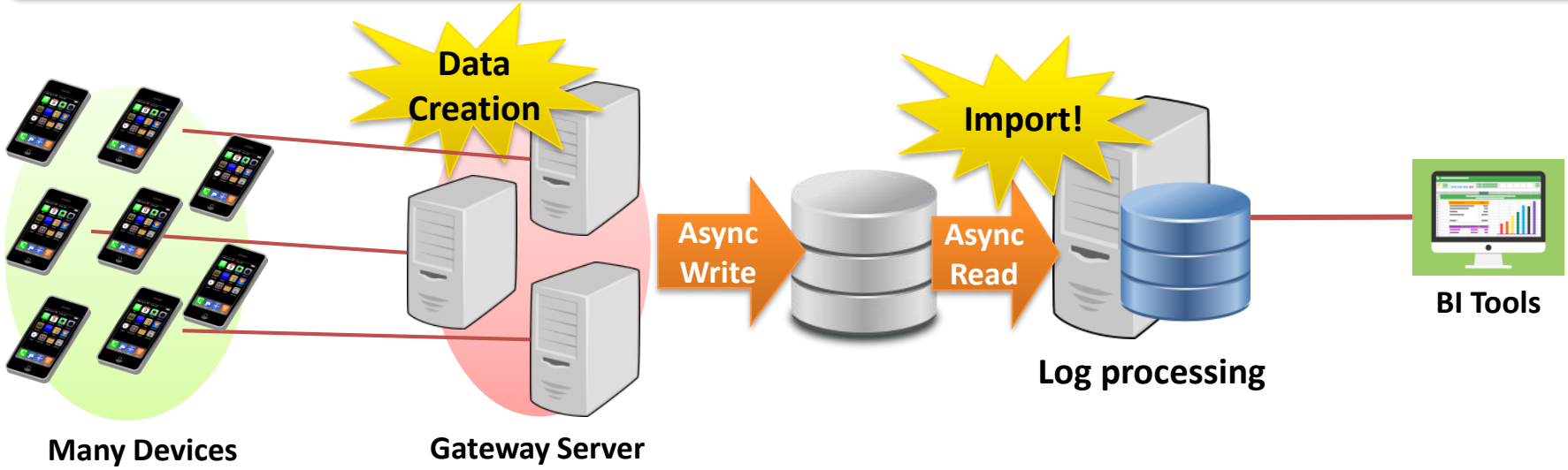
➔ 改めてデータをDBシステムにインポートする必要がない。

《補足》ログデータはどこで生成されるのか？

伝統的なOLTP&OLAPシステム – データはDBシステムの**内側**で生成される

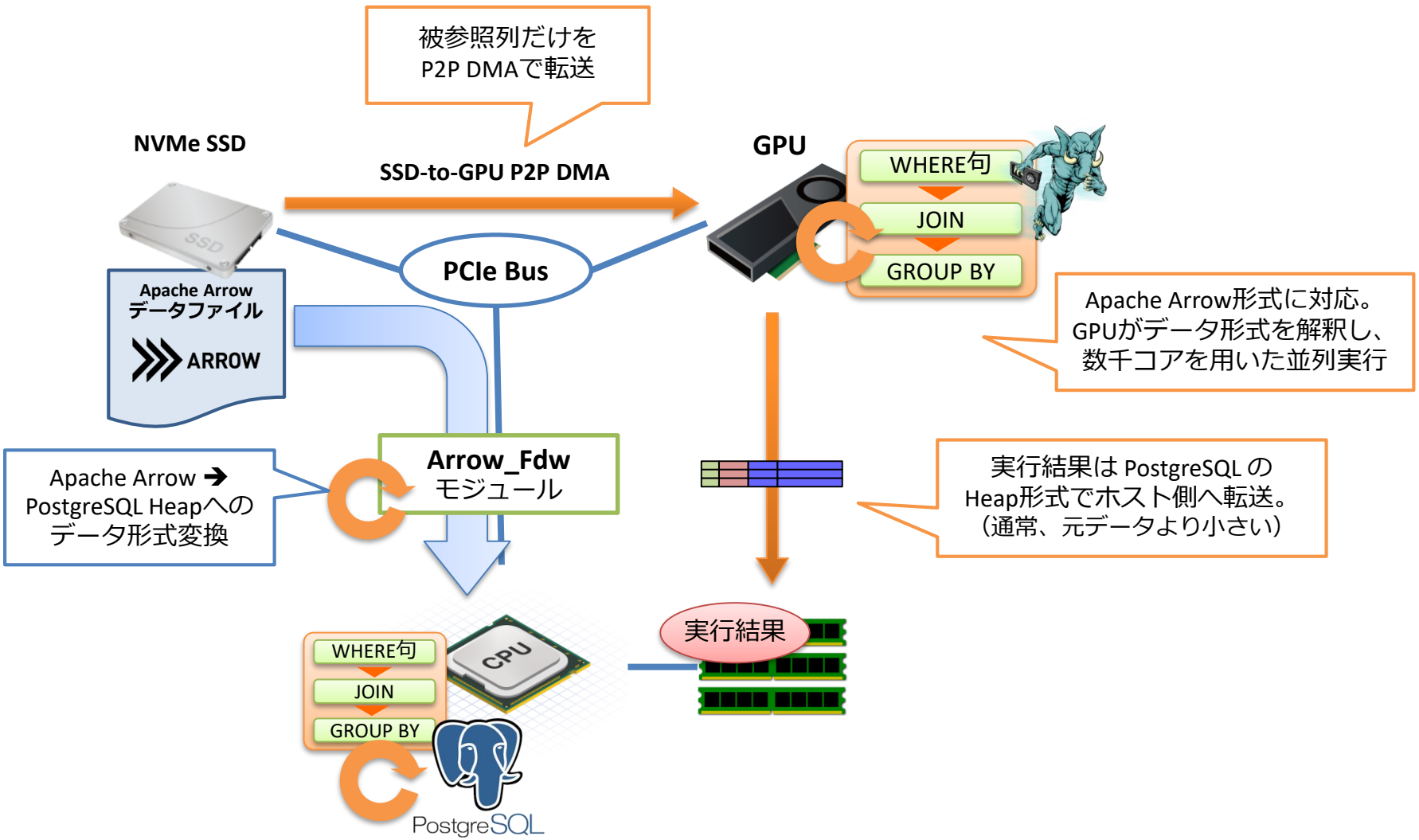


IoT/M2M時代 – データはDBシステムの**外側**で生成される

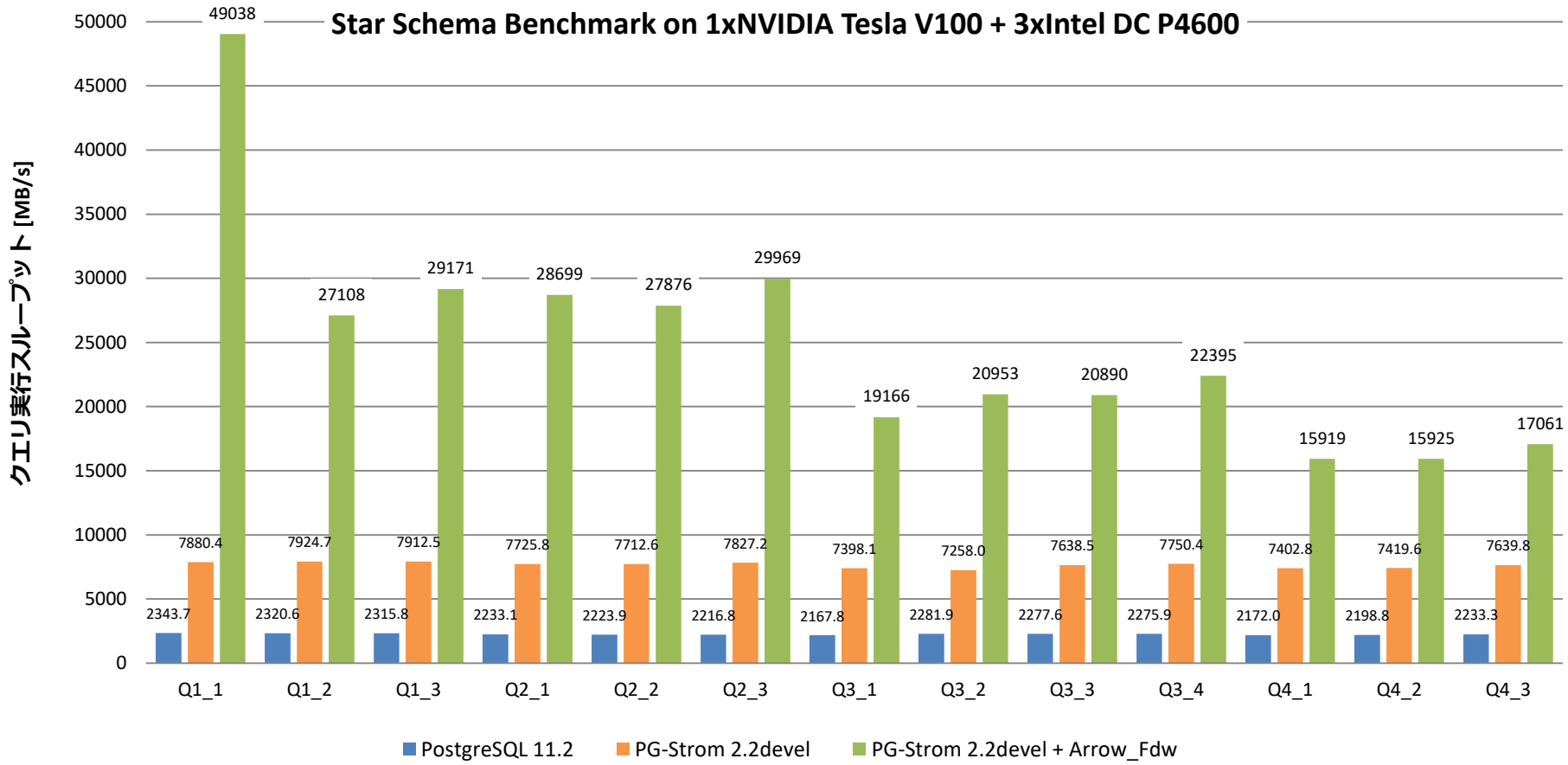


SSD-to-GPUダイレクトSQL on Arrow_Fdw

“被参照列だけ”をGPUに転送する点を除き、同一のインフラを使用している。



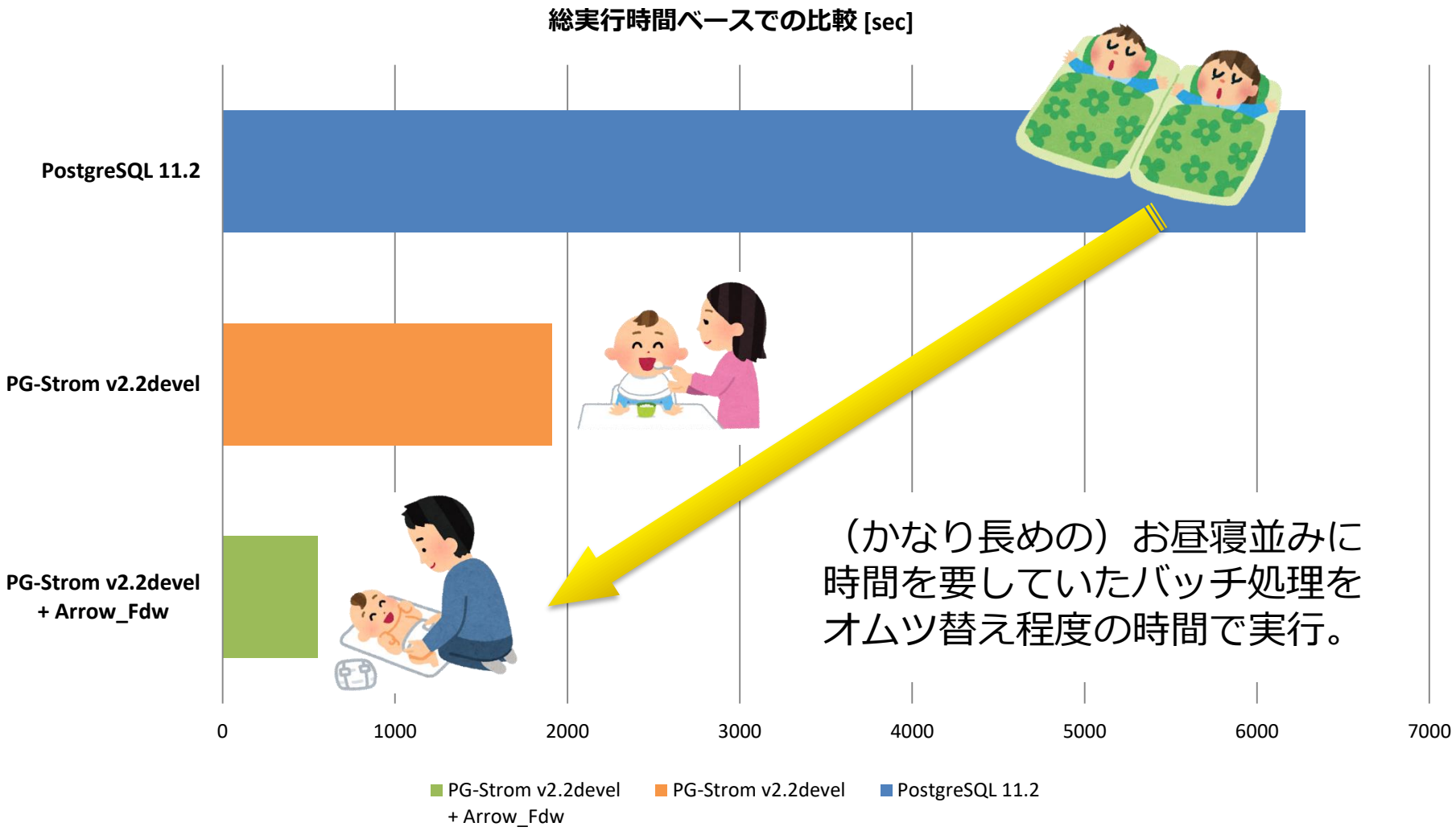
ベンチマーク結果（1/2）



- クエリ実行スループット = 353GB(DB) or 310GB(ファイル) ÷ クエリ応答時間
- PostgreSQLの2.2～2.3GB/sに対して、SSD-to-GPUダイレクトと列データの効果により、15GB/s～49GB/s相当のクエリ実行スループットを実現。

➔ シングルノードで数十TB相当のデータ処理も現実的なスコープに達してきた。

ベンチマーク結果（2/2） – 総実行時間の比較



※ ベンチマーク結果を採取するために、SSBMの13本のクエリを各々3回ずつ実行した。
上記グラフは、その総実行時間を実行エンジンごとに集計したもの。

ベンチマーク結果の検証

データ転送の性能は変わらないが、予め不要な列を読み飛ばせる事による高速化

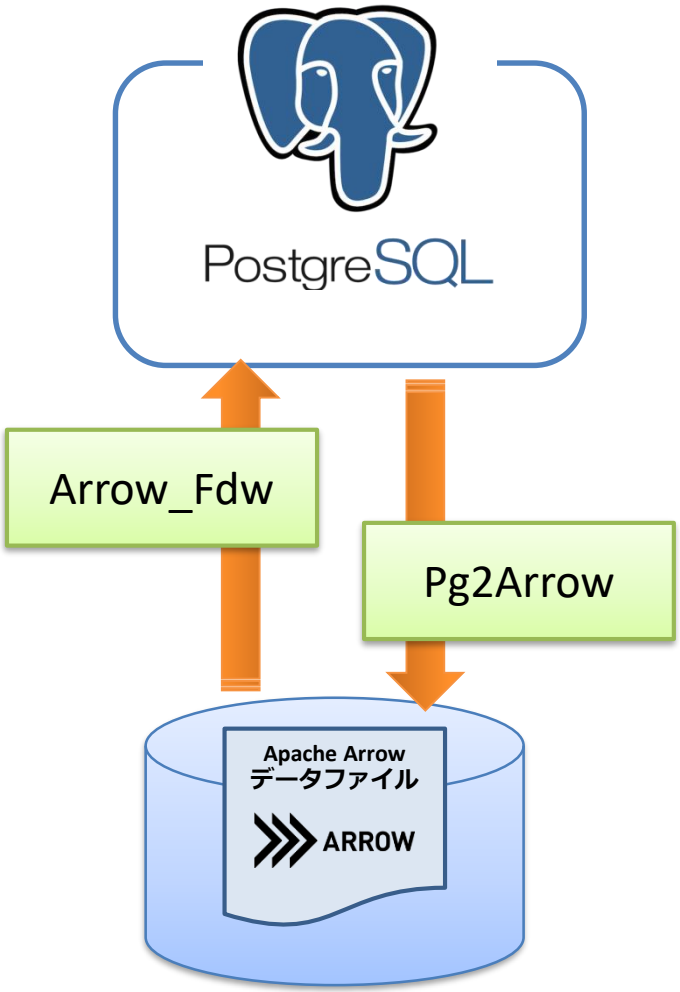
```
 Foreign table "public.flineorder"
       Column        |     Type      | Size
---------------------+---------------+---------------------------------
 lo_orderkey         | numeric       | 35.86GB
 lo_linenumber       | integer       |  8.96GB
 lo_custkey          | numeric       | 35.86GB
 lo_partkey          | integer       |  8.96GB
 lo_suppkey          | numeric       | 35.86GB   <-- ★Q2_1で参照
 lo_orderdate        | integer       |  8.96GB   <-- ★Q2_1で参照
 lo_orderpriority    | character(15) | 33.61GB   <-- ★Q2_1で参照
 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   <-- ★Q2_1で参照
 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
```

- 実際に読み出しているデータの合計は310GB中96.4GBのみ（31.08%）
- Q2_1の実行時間は 11.06s なので、96.4GB / 11.06s = 8.7GB/s

➔ Intel DC P4600 x3 の読出し速度としては概ね妥当なパフォーマンス。

Apache Arrowファイルの作成（1/2）

Pg2Arrowコマンドで、SQL出力結果をArrow形式で保存できる



- ✓ 実行すべきSQLとArrow形式データを出力するファイルを指定すればOK

```
$./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.
```

Apache Arrowファイルの作成（2/2） – Pg2arrow使用例

```
$ ./pg2arrow -d postgres -c “SELECT * FROM hogehoge LIMIT 1000“
                         -o /tmp/hogehoge1000.arrow

$ python
>>> import pyarrow as pa
>>> X = pa.RecordBatchFileReader("/tmp/hogehoge1000.arrow").read_all()
>>> X.schema
id: int32
a: int64
b: double
c: struct<x: int32, y: double, z: decimal(30, 11), memo: string>
  child 0, x: int32
  child 1, y: double
  child 2, z: decimal(30, 11)
  child 3, memo: string
d: string
e: double
ymd: date32[day]
```

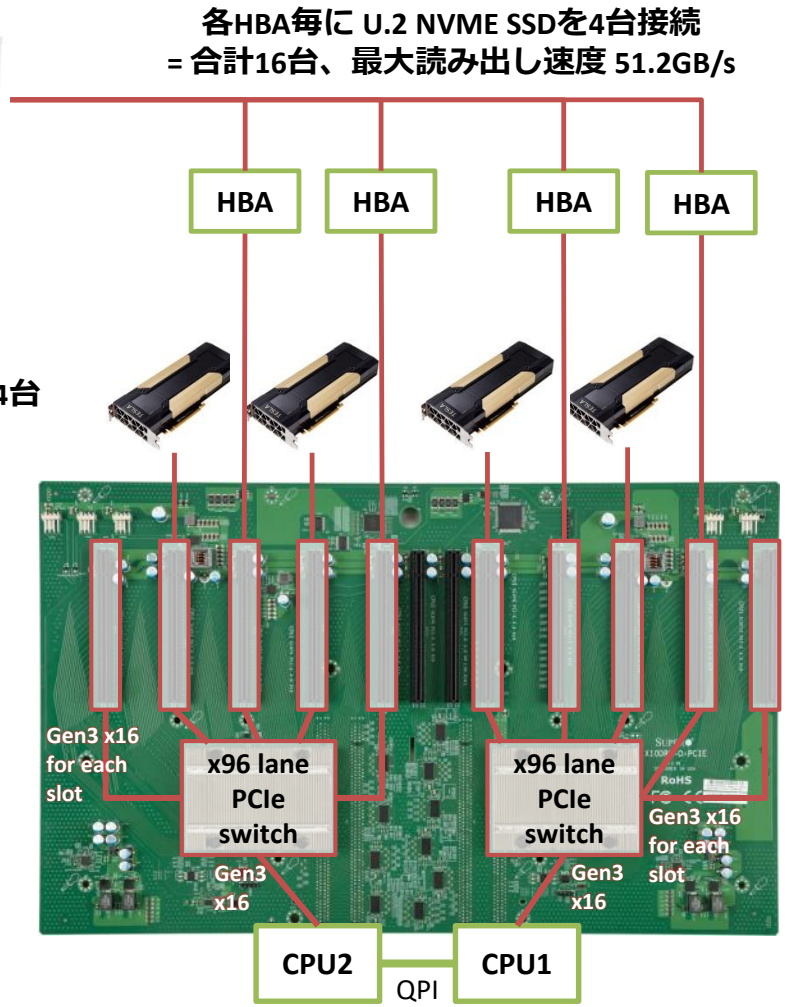
今後のロードマップ

作ってみたいシステム構成（1/2）

スケールアップ技術と列データ処理技術を組合せ、実効処理速度 **100GB/s** を目指す。

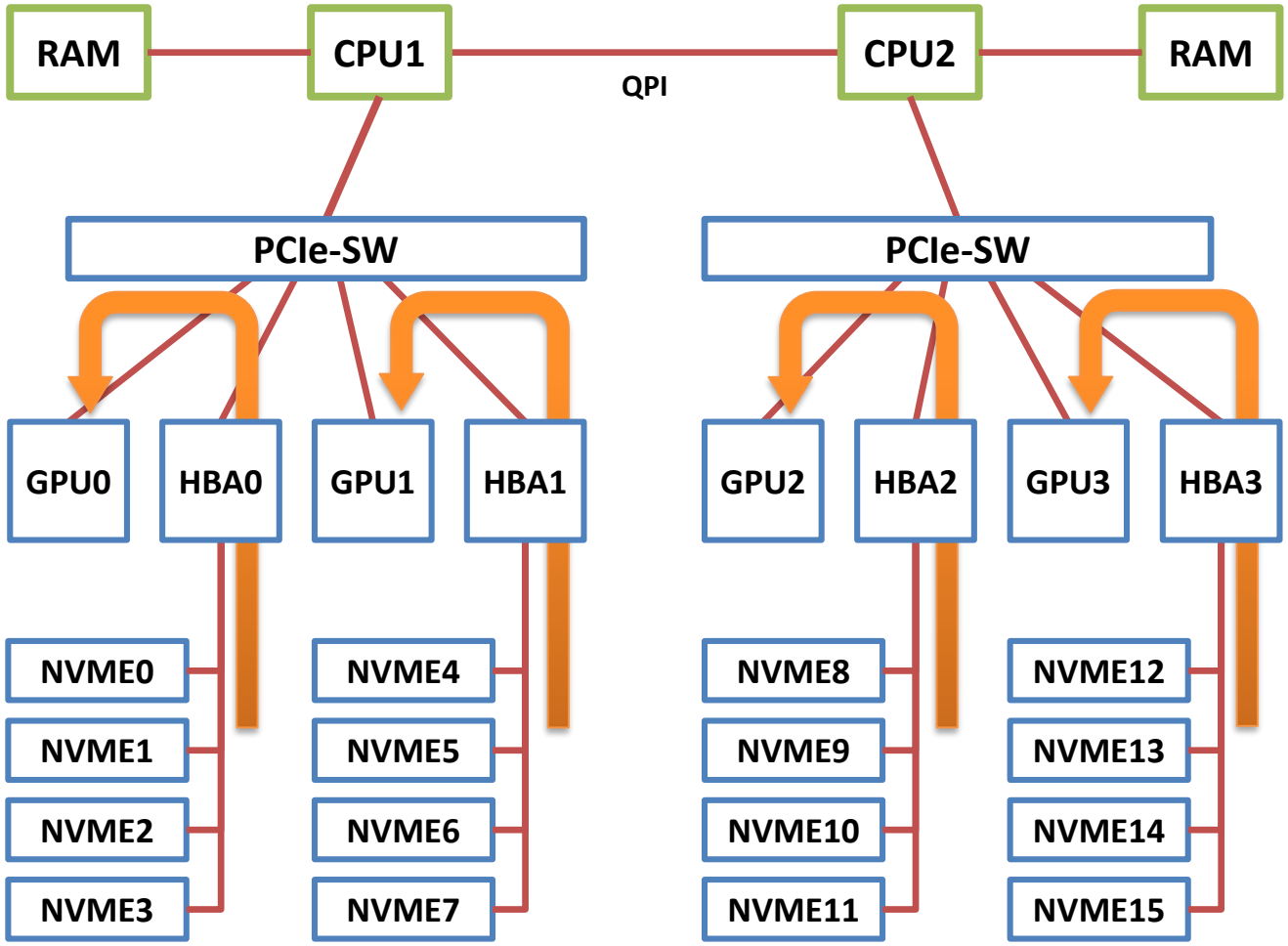
U.2 NVME JBOF製品

Supermicro社
SYS-4029TRT2



作ってみたいシステム構成（2/2）

スケールアップ技術と列データ処理技術を組合せ、実効処理速度 **100GB/s** を目指す。



GPU+4xSSDユニット
あたり**8.0〜10GB/s**の
物理データ転送性能

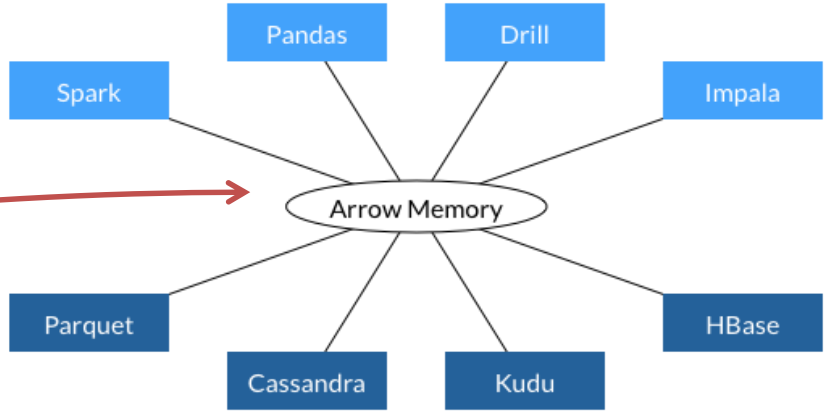
列データ構造により
ユニット毎**25〜30GB/s**の
実効データ転送性能

４ユニットの並列動作で、
100〜120GB/sの
実効データ処理能力

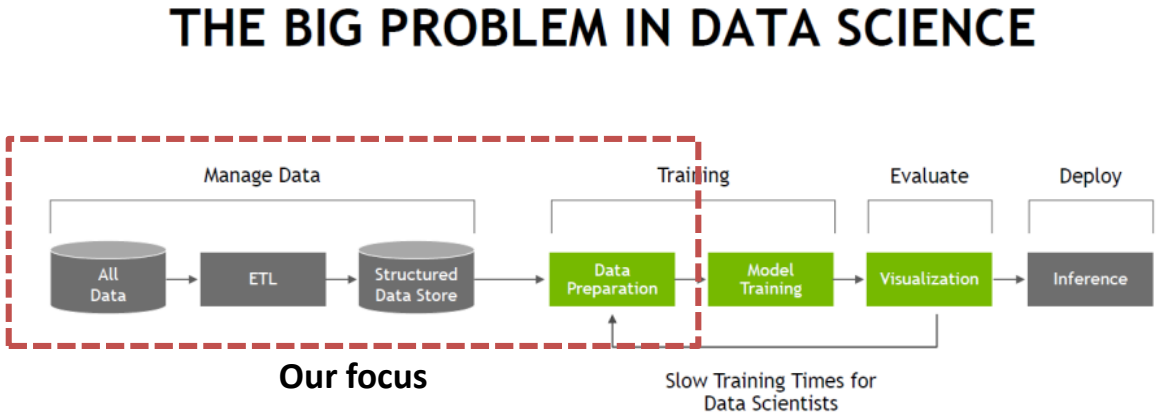
〜100TB程度の
ログデータを
シングルノードで
処理可能に

Apache Arrow / NVIDIA RAPIDSを介した機械学習エンジンとの統合

NVIDIA RAPIDS:
GPU上のデータフレームを統計解析・機械学習のソリューション間で交換するためのインフラストラクチャ。
データ形式にはApache Arrowを採用している。



Why PG-Strom is unique:
NVIDIA自身も、GPUがI/O中心のワークロードに適用可能だとは考えていない節があります。
しかし、SSD-to-GPU Direct SQL技術はGPUの適用領域をインメモリサイズから、ストレージサイズへと拡大し、データ処理のライフサイクル全体をカバーする事が可能となるのです。



まとめ

PG-Stromの中核機能

- SQLからGPUプログラムを自動生成し、数千コアでSQLワークロードを並列処理。
- SSD-to-GPUダイレクトSQLにより、I/OもGPUで高速化できる時代になってきた。

更なる高速化に向けて

- PCIeスイッチ：CPUをバイパスしてSSD⇔GPU間のデータ転送を可能にする。
  - ✓ テーブルパーティションとの併用でデータ転送経路を最適化
- Apache Arrow：列指向データ構造で、外部アプリとのデータ交換も可能に。
  - ✓ 被参照列だけを読み出す事で、I/O負荷を減らし実効性能を向上させる。
  - ✓ DBシステムへのインポートが不要であり、前処理時間を短縮可能

今後のロードマップ

- HPC向けハードウェアを活用する事で、実効処理性能 100GB/s を目指す。
- NVIDIA RAPIDSに対応する事で、同一システム上で大量データの集計・解析処理から、機械学習処理までをカバーする。

➔ IoT/M2M系ワークロードで、総データ容量 100TB 以下の領域をターゲットにソリューションの開発を進める。

関連リソース

本日の発表資料

- https://www.slideshare.net/kaigai/20190418pgstromonarrowfdw/

PG-Stromドキュメント

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

ソースコード

- https://github.com/heterodb/pg-strom
- https://github.com/heterodb/pg2arrow

コンタクト

- e-mail: kaigai@heterodb.com
- twitter: @kkaigai


HeteroDB