Empowered by Innovation NEC

GPGPUがPostgreSQLを加速する

～"超"メニーコアによる検索高速化～

NEC OSS推進センター
The PG-Strom Project
海外 浩平 <kaigai@ak.jp.nec.com>
(Tw: @kkaigai)

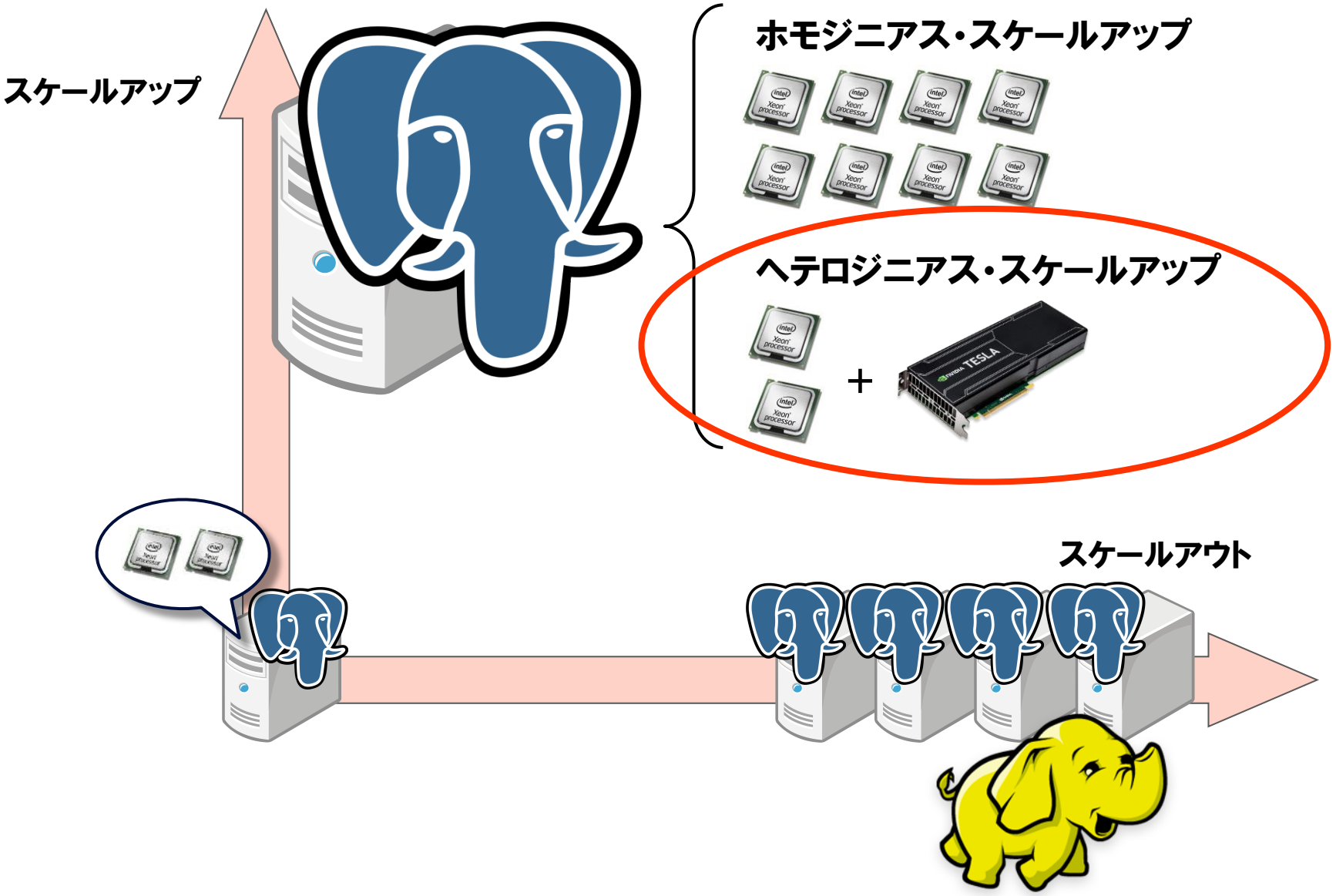
自己紹介

- 名前: 海外 浩平
- 所属: NEC OSS推進センター
- 好きなもの: コアの多いプロセッサ
- 嫌いなもの: コアの少ないプロセッサ
- 経歴:
  - HPC ➔ OSS/Linux ➔ SAP ➔ GPU/PostgreSQL
- Tw: @kkaigai

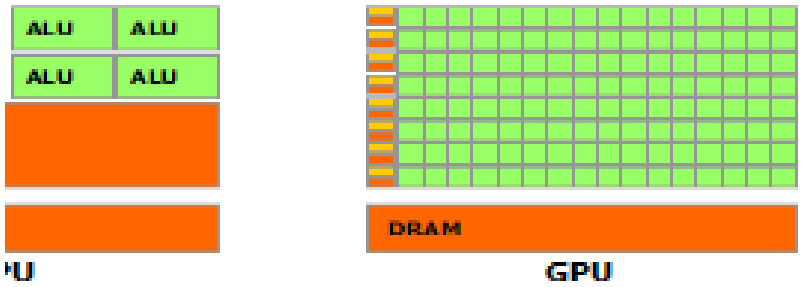
主な仕事

- SELinux周り諸々 (2004~)
  - Lockless AVC、JFFS2 XATTRなど
- PostgreSQL周り諸々 (2008~)
  - SE-PostgreSQL、Security Barrier View、Writable FDWなど
- PG-Strom (2012~)

処理性能向上のアプローチ



GPU (Graphic Processor Unit) の特徴



SOURCE: CUDA C Programming Guide (v6.5)

	GPU	CPU
Model	Nvidia Tesla K20X	Intel Xeon E5-2690 v3
Architecture	Kepler	Haswell
Launch	Nov-2012	Sep-2014
# of transistors	7.1billion	3.84billion
# of cores	2688 (simple)	12 (functional)
Core clock	732MHz	2.6GHz, up to 3.5GHz
Peak Flops (single precision)	3.95TFLOPS	998.4GFLOPS (AVX2使用)
DRAM size	6GB, GDDR5	768GB/socket, DDR4
Memory band	250GB/s	68GB/s
Power consumption	235W	135W
Price	$3,000	$2,094

GPUの特徴

- チップに占める演算ユニットの比率が高い
- キャッシュ・制御ロジックの比率が小さい

➔ 単純な演算の並列処理に向く。
複雑なロジックの処理は苦手。

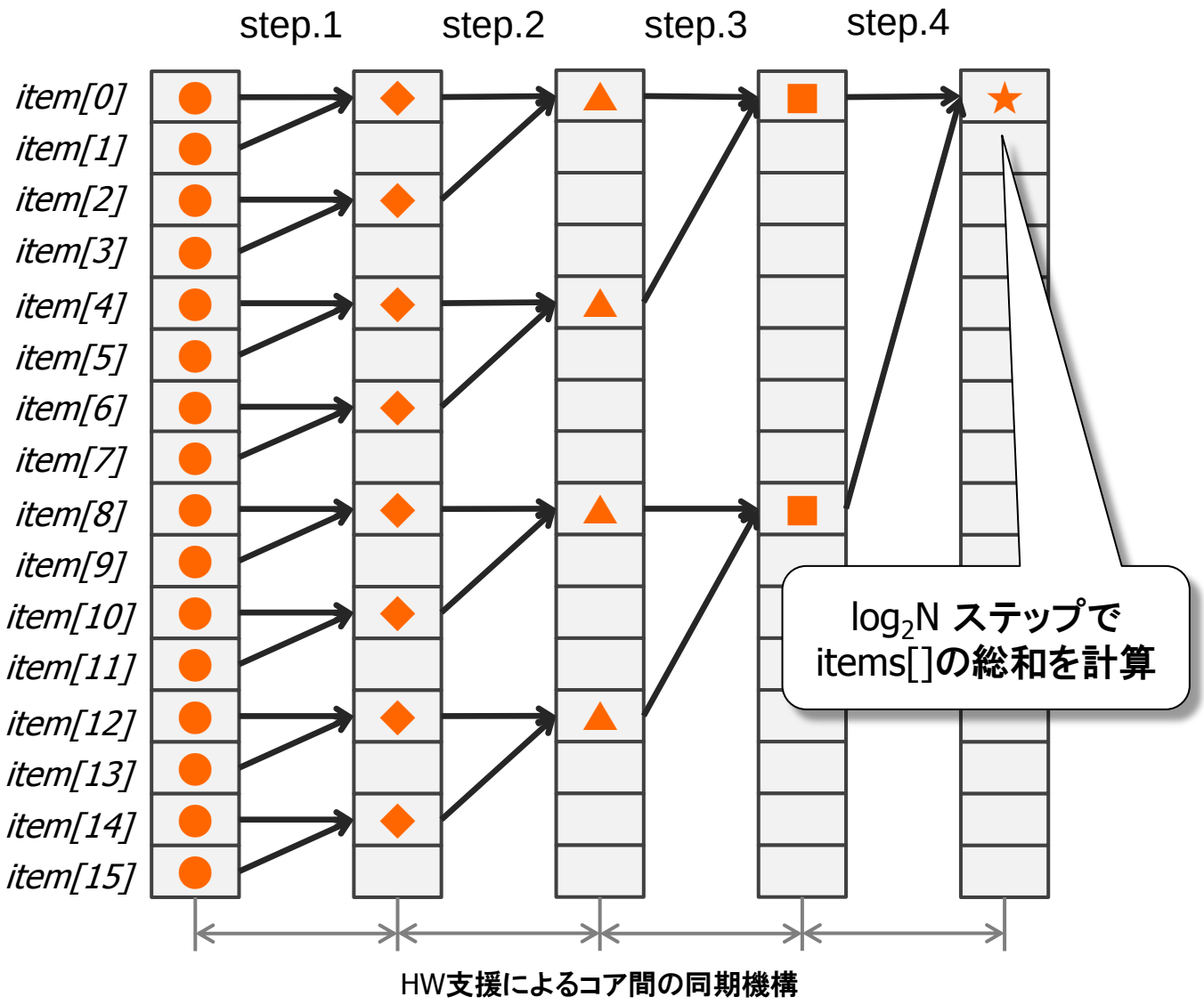
- 値段の割にコア数が多い
  - GTX750Ti(640core)で 1万5千円くらい

GPU活用による計算の例 – 縮約アルゴリズム

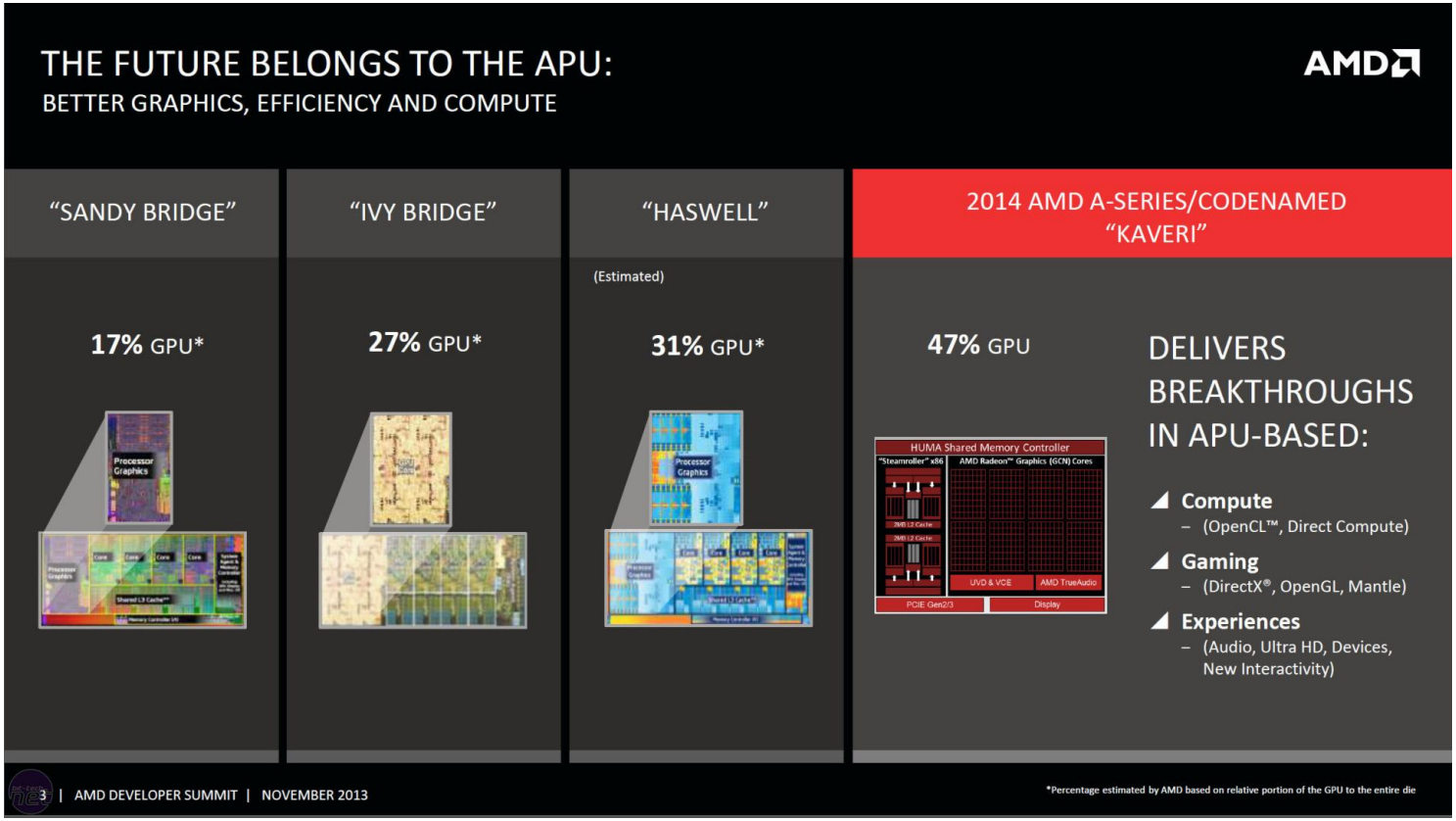
N個のGPUコアによる

$\Sigma_{i=0...N-1}$item[i]

配列総和の計算



半導体技術のトレンド (1/2) – ヘテロジニアス化

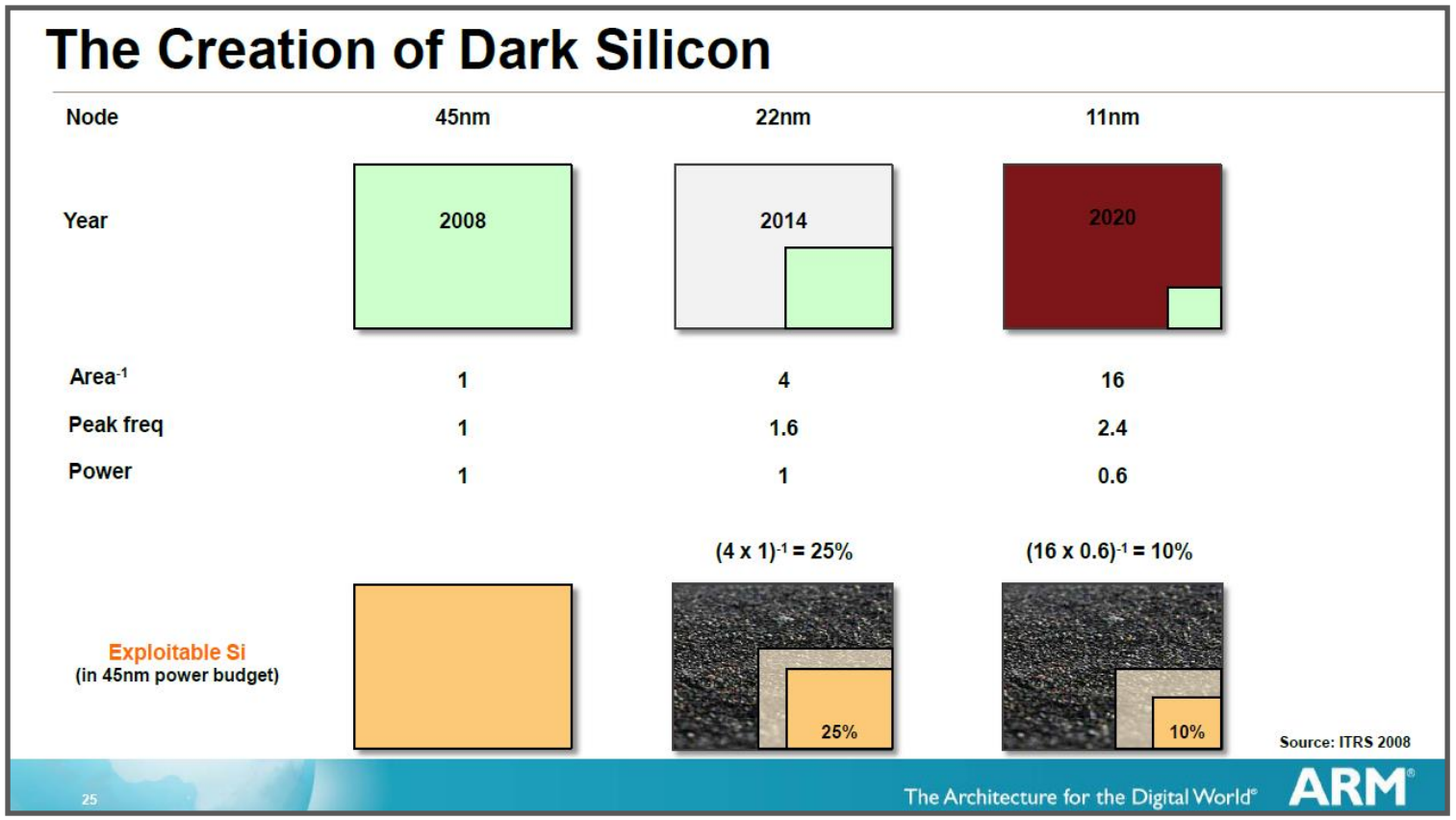


SOURCE: THE HEART OF AMD INNOVATION, Lisa Su, at AMD Developer Summit 2013

- マルチコアCPUから、CPU/GPU統合型アーキテクチャへの移行
- HW性能向上からSWがフリーランチを享受できた時代は終わりつつある。

➔ HW特性を意識してSWを設計しないと、半導体の能力を引き出せない。

半導体技術のトレンド (2/2) – ダークシリコン問題



SOURCE: Compute Power with Energy-Efficiency, Jem Davies, at AMD Fusion Developer Summit 2011

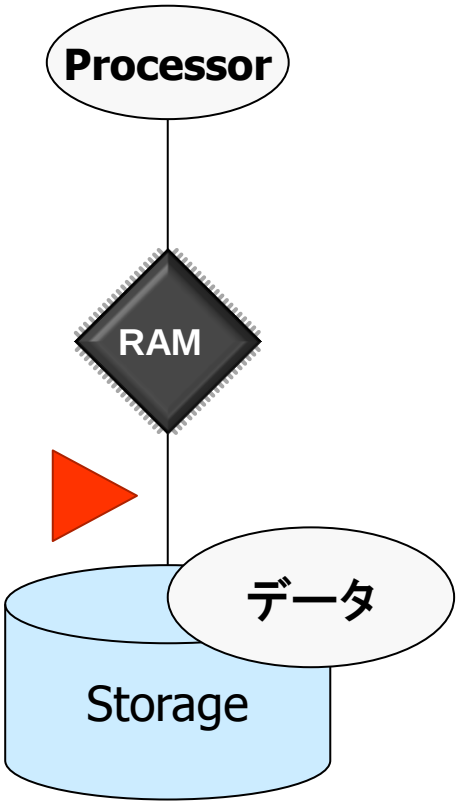
CPU/GPU統合型アーキテクチャの背景

- トランジスタ集積度向上のスピード > トランジスタあたり消費電力削減のスピード
- 排熱が追いつかないので、全ての回路に同時に電力供給するワケにはいかない。

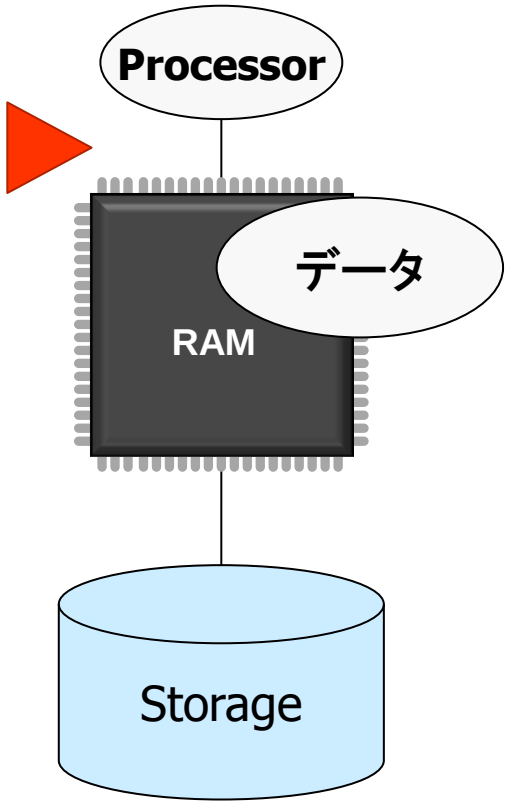
➔ 同じチップ上に特性の異なる回路を実装。ピーク電力消費を抑える。

RDBMSとボトルネックを考える (1/2)

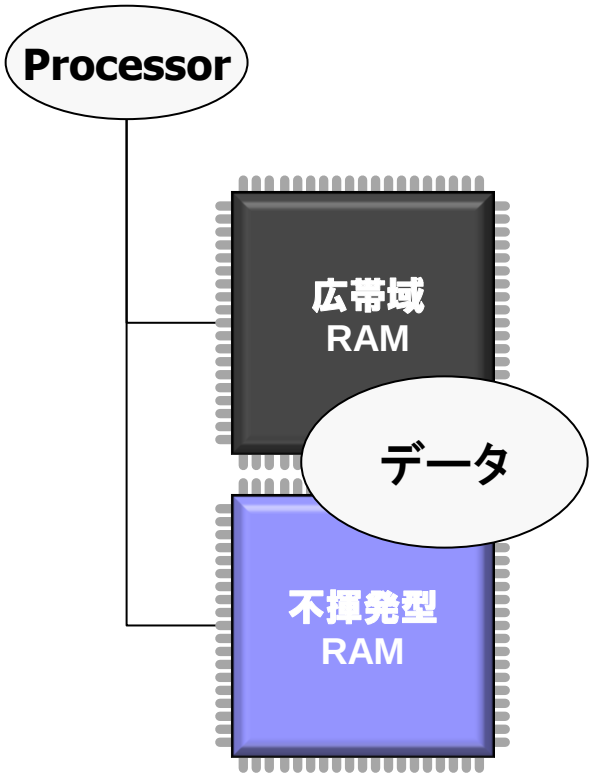
データサイズ > RAM容量



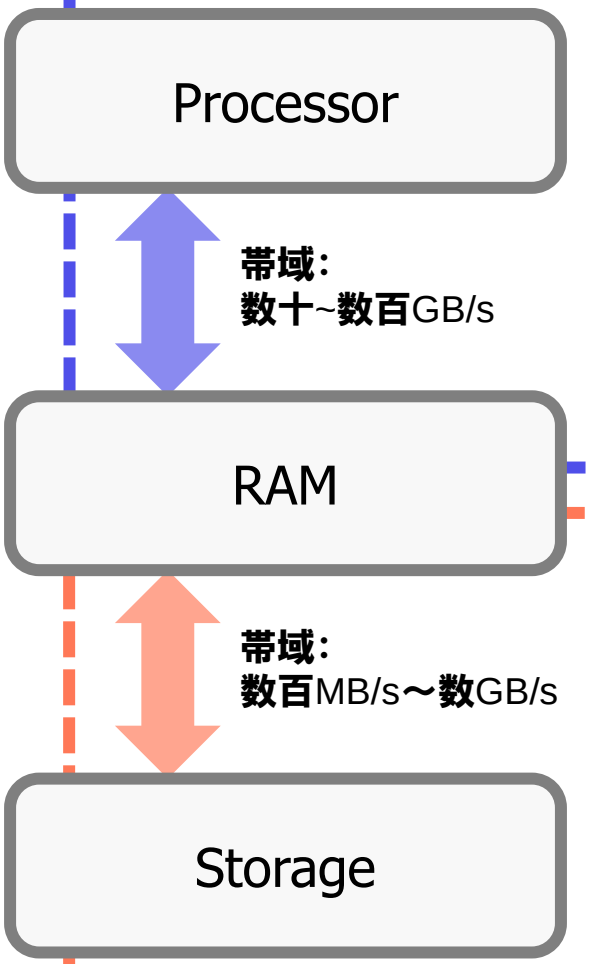
データサイズ < RAM容量



将来は？



RDBMSとボトルネックを考える (2/2)



現代のメモリボトルネックの世界

Join、Aggregation、Sort、Projectionなど

戦略

- バーストしやすいアクセスパターン
- 並列計算アルゴリズムの適用

伝統的なディスクI/Oボトルネックの世界

SeqScan、IndexScanなど

戦略

- I/Oの量 (サイズ、回数) を減らす
- ディスク分散 (RAID)

PG-Stromのアプローチ

PG-Stromとは

- PostgreSQL用の拡張モジュール
- SQL処理ワークロードの一部を**GPUで並列実行**し、高速化を実現。
- Full-Scan、Hash-Join、Aggregate の3種類のワークロードに対応

(2014年11月時点のβ版での対応状況)

コンセプト

- SQL構文からGPU用のネイティブ命令を動的に生成。JITコンパイル。
- CPU/GPU協調による非同期並列実行

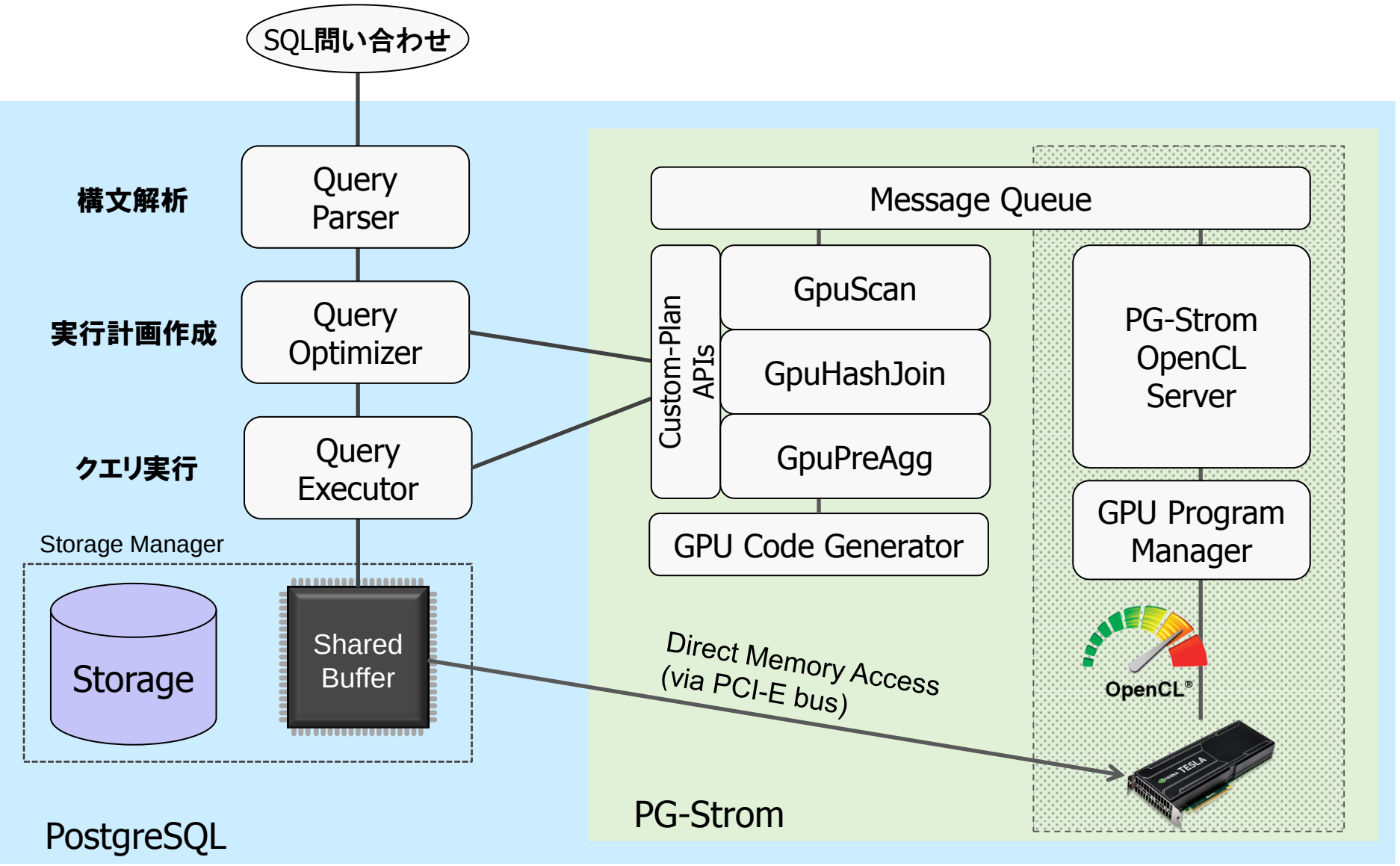
利点

- 利用者からは完全に透過的に動作。
  - SQL構文や周辺ツール、ドライバを含めPostgreSQL向けのソフトウェア資産を利用可
- GPU+OSSの組み合わせにより、低コストでの性能改善が計れる。

注意点

- ただし、現時点ではインメモリ・データストアが前提。

PG-Stromのアーキテクチャ (1/2)



PG-Stromのアーキテクチャ (2/2)

OpenCL

- ヘテロジニアス計算環境を用いた並列計算フレームワーク
- NVIDIAやAMDのGPUだけでなく、CPU並列にも適用可能
- 言語仕様にランタイムコンパイラを含む。

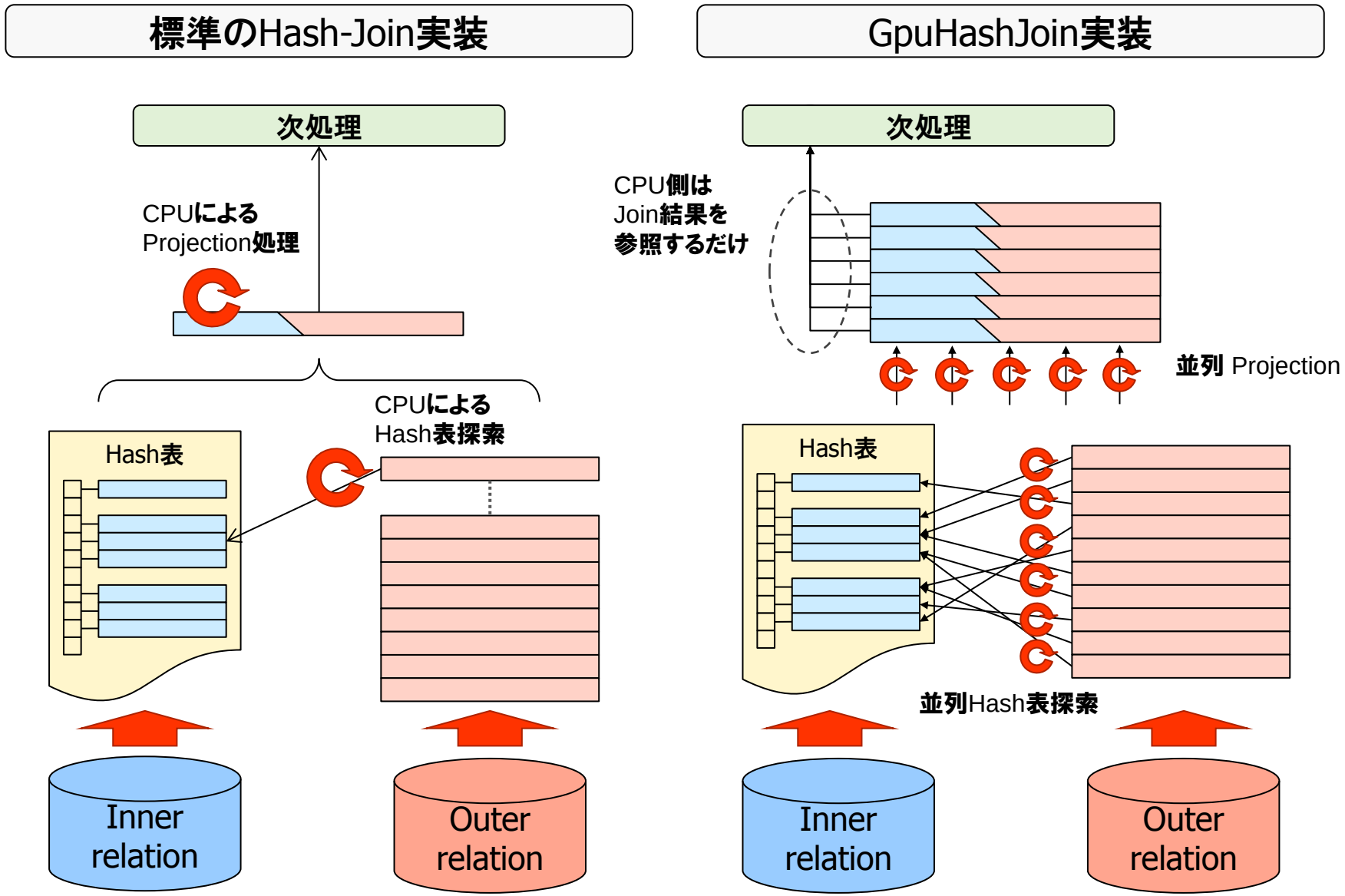
行指向データ構造

- PostgreSQLの共有バッファをDMA転送元として利用する。
- GPUの性能を引き出すには列指向データが最適ではあるものの、行⇔列変換が非常にコスト高で本末転倒に。

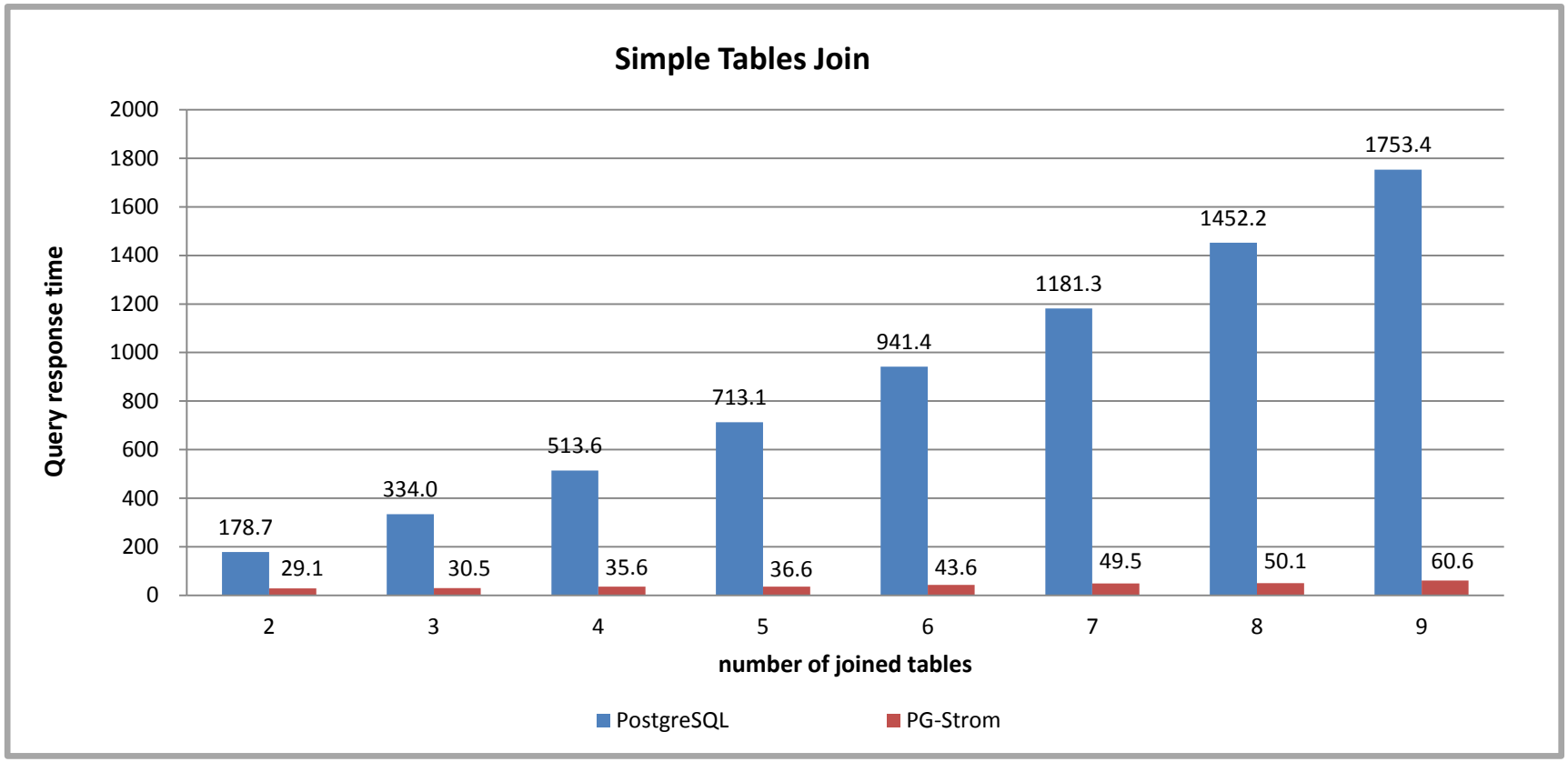
Custom-Plan APIs

- あたかもPostgreSQLのSQL処理ロジックであるかのように、拡張モジュールがスキャンやジョインを実装するための機能。
- PostgreSQL v9.5に向けて提案し、現在、開発者コミュニティにおいて議論が進められている。

PG-Stromの処理イメージ – Hash-Joinのケース



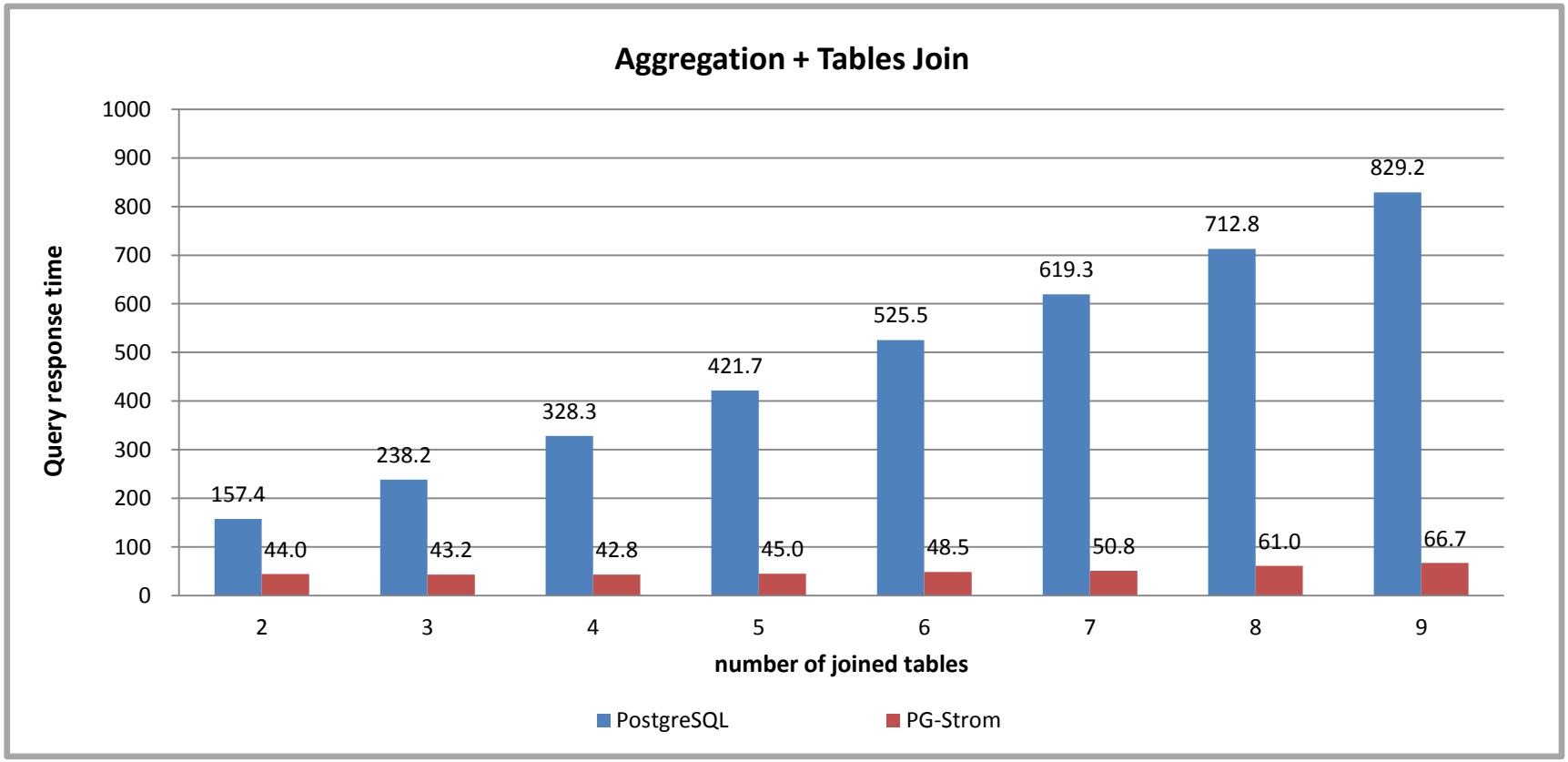
ベンチマーク (1/2) – Simple Tables Join



[測定条件]

- 2億件 x 10万件 x 10万件 ... のINNER JOINをテーブル数を変化させながら実行
- 使用したクエリ: `SELECT * FROM t0 natural join t1 [natural join t2 ...];`
- 全てのテーブルは事前にバッファにロード済み
- HW: Express5800 HR120b-1, CPU: Xeon E5-2640, RAM: 256GB, GPU: NVIDIA GTX980

ベンチマーク (2/2) – Aggregation + Tables Join



[測定条件]

- 2億件 x 10万件 x 10万件 ... のINNER JOINをテーブル数を変化させながら実行
- 使用したクエリ:

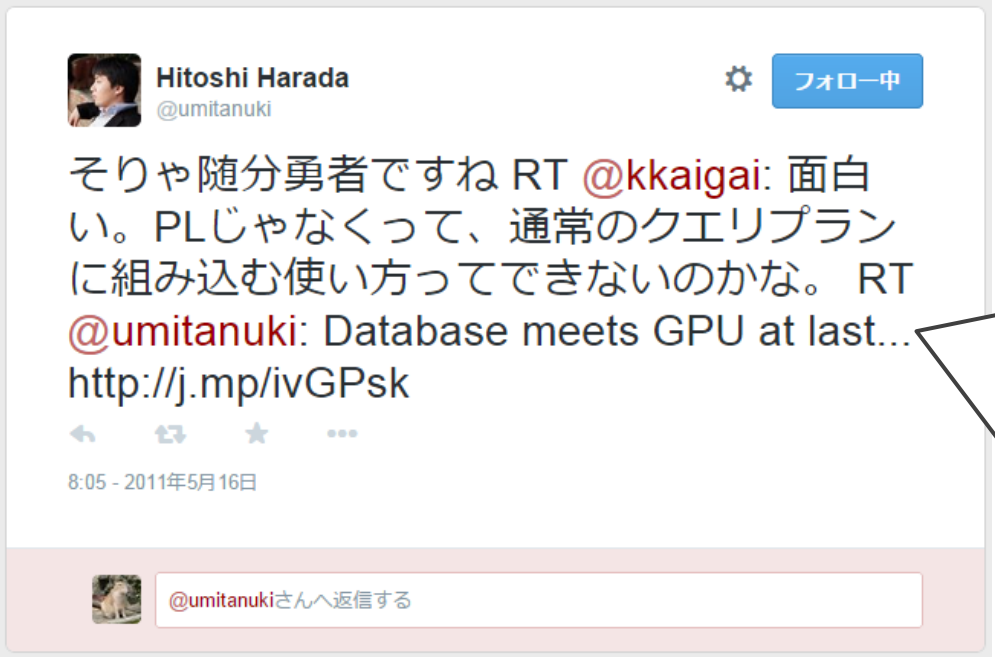
```
SELECT cat, AVG(x) FROM t0 natural join t1 [natural join t2 ...] GROUP BY CAT;
```

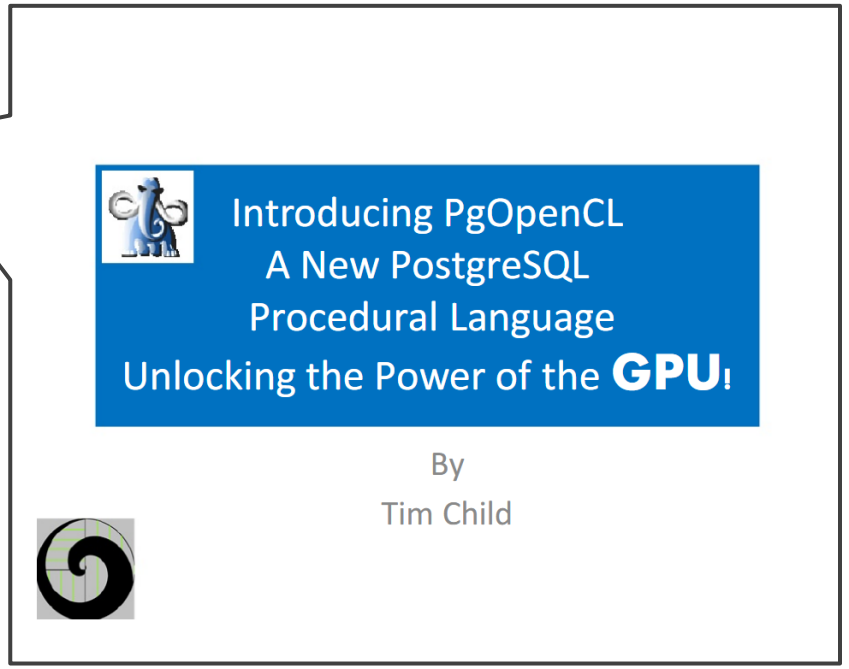
- 他の測定条件は前試験と同一

PG-Strom開発の歴史

2011	2月 海外、ドイツへ赴任 5月 PGconf 2011
2012	1月 最初のプロトタイプを実装 5月 疑似コード実行のアプローチ 8月 Background Worker & Writable FDWの提案
2013	7月 PG-Stromの開発がお仕事になる 11月 CustomScan Interfaceの提案
2014	2月 CUDAからOpenCLへの移行 6月 GpuScan、GpuSort、 GpuHashJoinを実装 9月 GpuSortを破棄、GpuPreAggを実装 10月 列指向キャッシュを破棄 11月 β版を公開

着想 – 2011年5月頃

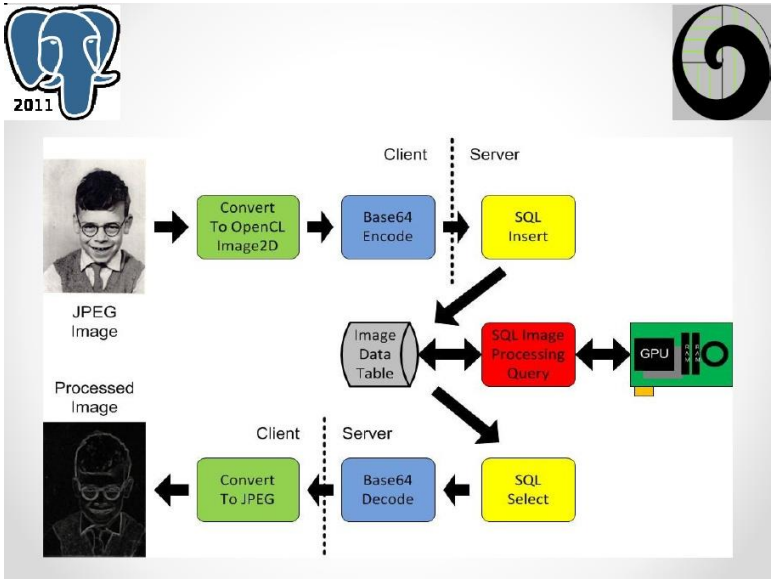




http://ja.scribd.com/doc/44661593/PostgreSQL-OpenCL-Procedural-Language

PGconf 2011 @ Ottawa, Canada

PgOpenclプロジェクトからの着想



Parallel Image Searching Using PostgreSQL PgOpenCL @ PGconf 2011
Tim Child (3DMashUp)

要約:

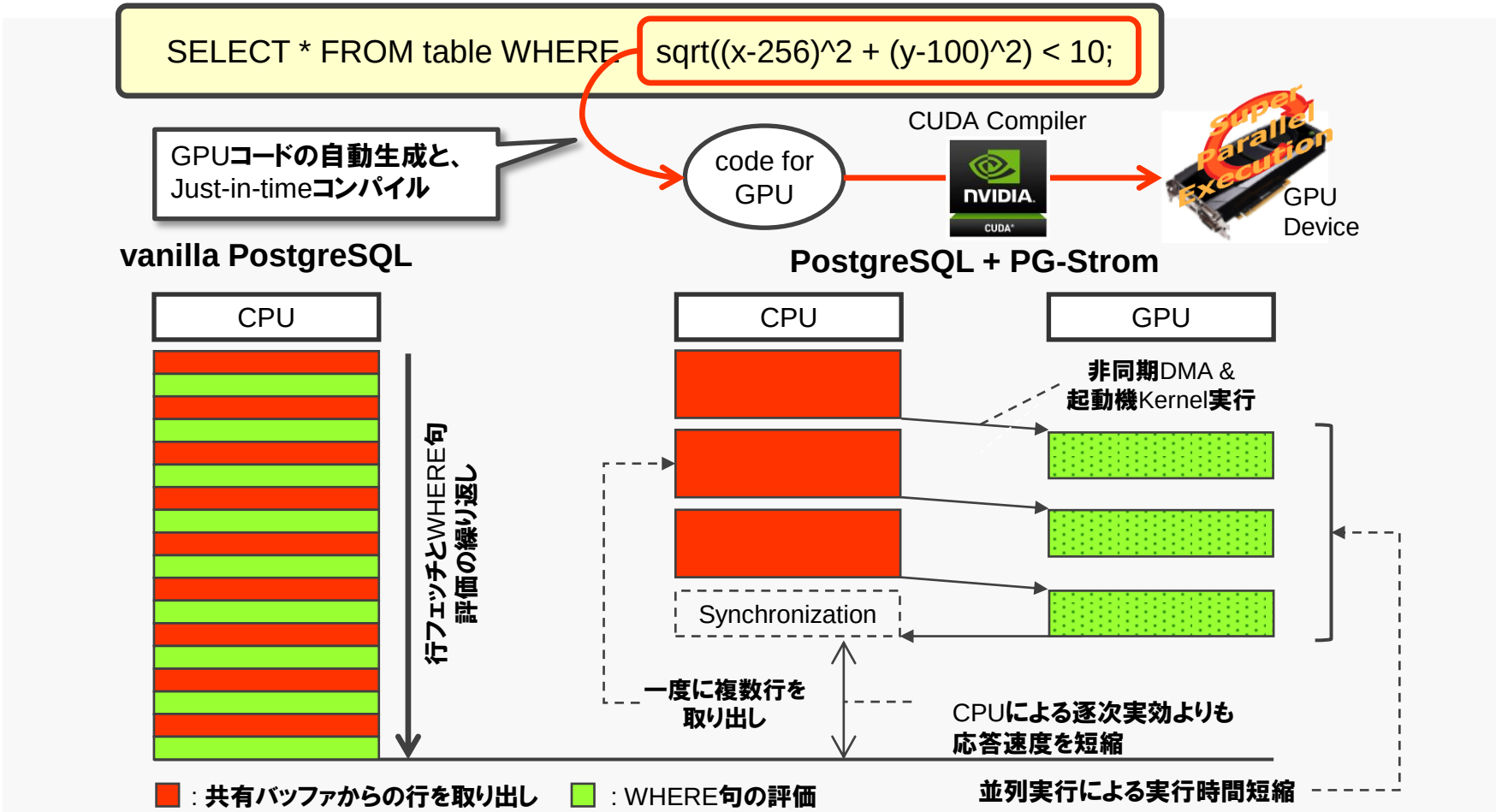
画像データみたく長大なBLOBが入っている時に、そいつをGPUで並列処理させてやるとイイよ！

幅の小さいデータでも、大量の行を並べれば長大なBLOBと同じくGPU並列処理が効くのでは？

A列	B列	C列	D列	E列

実際に作ってみた – 2011年末のクリスマス休暇

- CUDAを利用 (➔ 現在はOpenCL)
- FDW (Foreign Data Wrapper) を利用 (➔ 現在はCustomPlan APIs)
- 列指向データ構造 (➔ 現在は行指向データ構造)

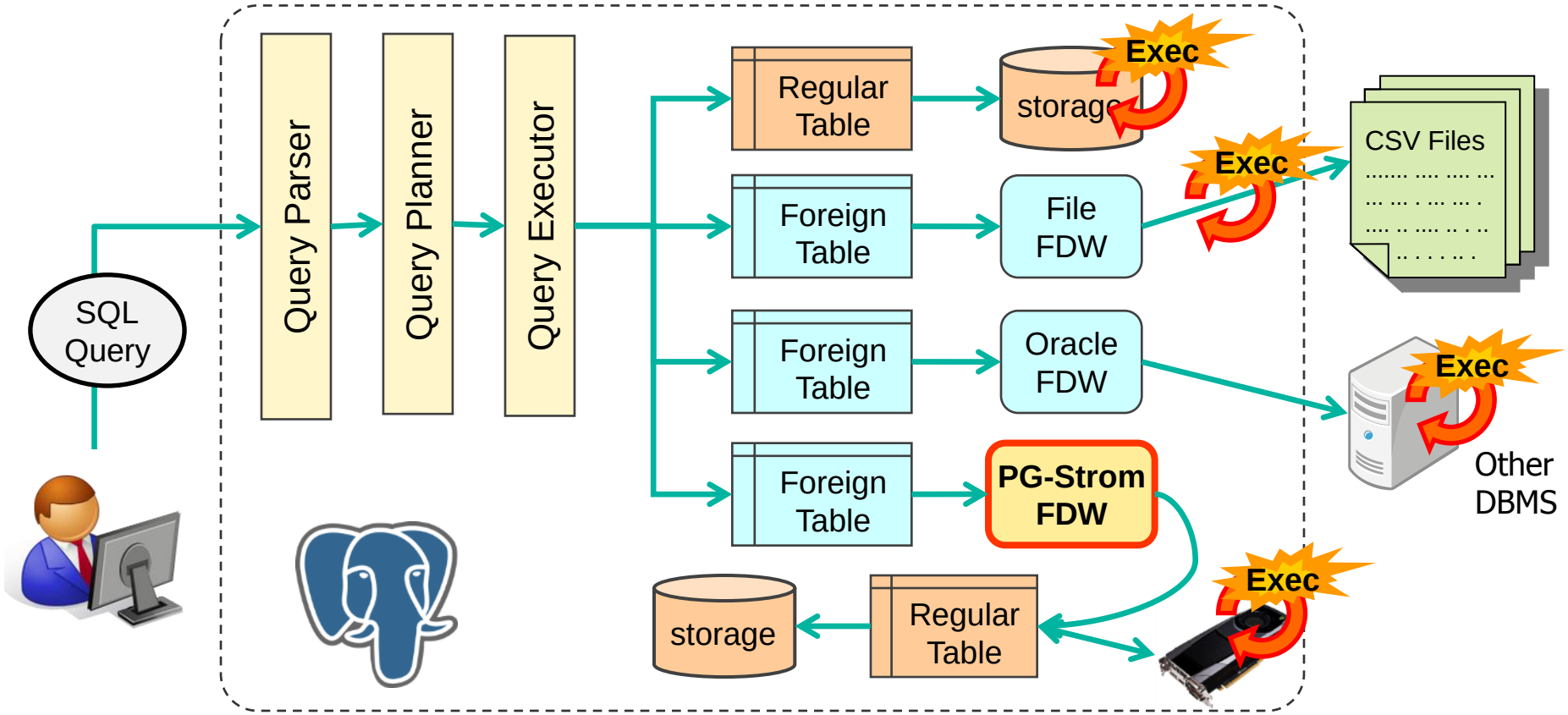


FDW (Foreign Data Wrapper) とは

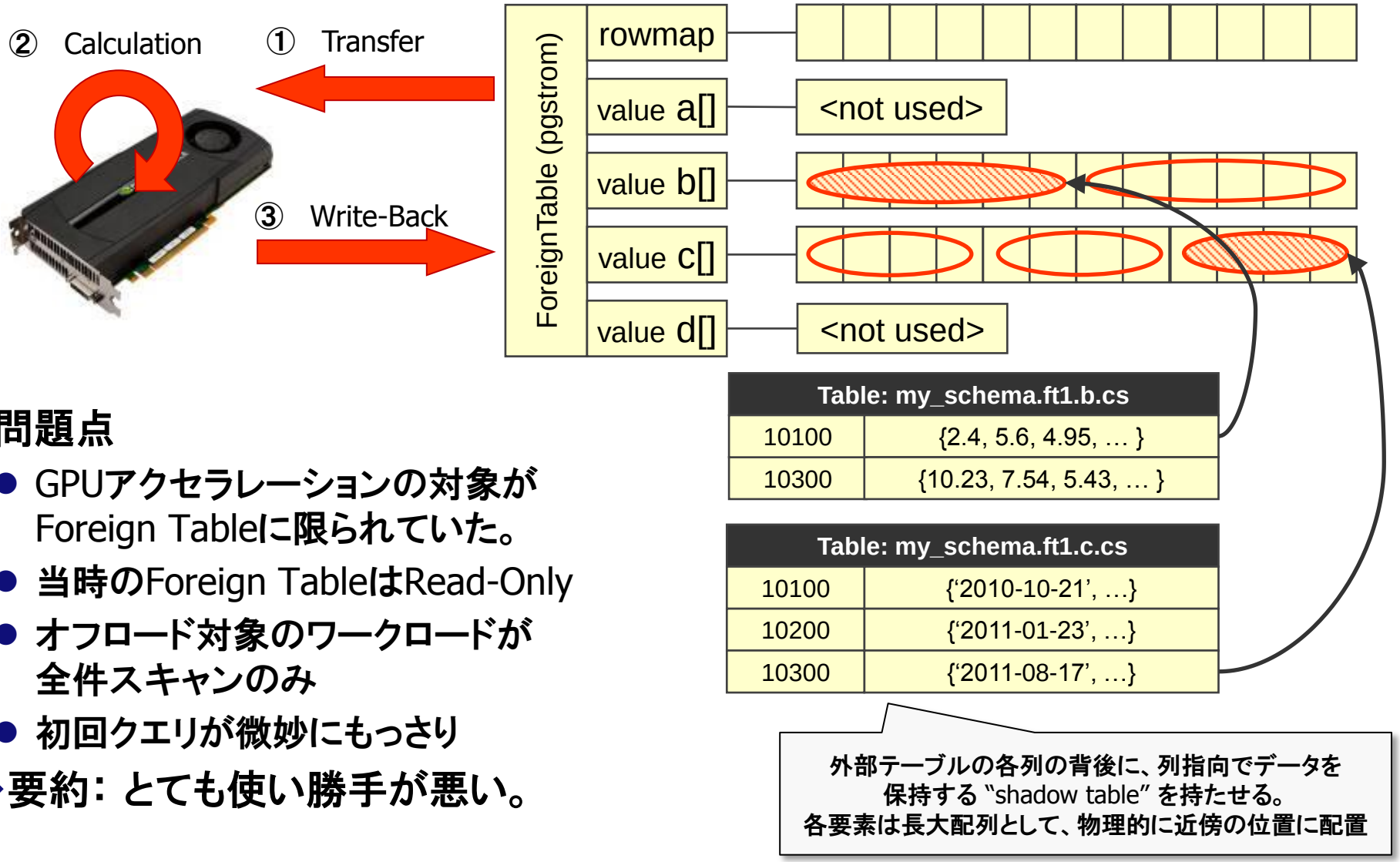
- 元々は外部データソースをPostgreSQLの表として見せるためのAPI群
- “外部テーブル” へのアクセスは、ドライバがPostgreSQLの行を生成する

➔ データ構造を自由に規定できる

➔ 処理ロジックを自由に規定できる



当時の実装



問題点

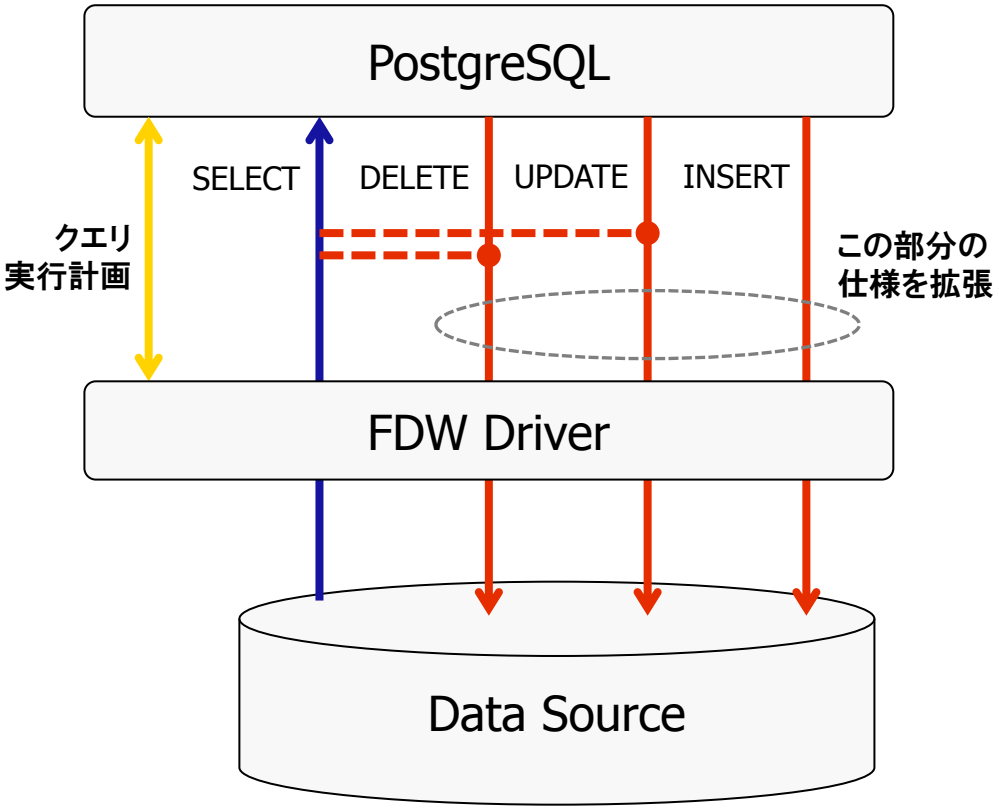
- GPUアクセラレーションの対象がForeign Tableに限られていた。
- 当時のForeign TableはRead-Only
- オフロード対象のワークロードが全件スキャンのみ
- 初回クエリが微妙にもっさり

➔要約：とても使い勝手が悪い。

PostgreSQLの強化 (1/4) – Writable FDW

- FDWの仕様上、Read-Onlyアクセスのみ (~v9.2)
  - 専用関数で shadow table へとデータをロードする必要があった。

➔API仕様を拡張して、外部テーブルへのINSERT・UPDATE・DELETEを可能に




PostgreSQL
The world's most advanced open source database.

Home | About | Download | Documentation | Community | Developers | Support | Your account

- Community
- Contributors
- Mailing Lists
  - Subscribe
  - User lists
  - Developer lists
    - pgsql-cluster-hackers
    - pgsql-committers
    - pgsql-hackers
    - pgsql-rrreviewers
    - pgsql-translators
    - pgsql-www
  - Regional lists
  - Associations
  - User groups
  - Project lists
  - Inactive lists
- IRC
- Local User Groups
- Featured Users
- International Sites
- Propaganda
- Resources
- Weekly News

Re: [v9.3] writable foreign tables

From: Tom Lane <tgl(at)sss(dot)pgh(dot)pa(dot)us>
To: Craig Ringer <craig(at)2ndquadrant(dot)com>
Cc: Kohei KaiGai <kaigai(at)kaigai(dot)gr(dot)jp>, Daniel Farina <daniel(at)heroku(dot)com>, pgsql-hackers <pgsql-hackers(at)postgresql(dot)org>
Subject: Re: [v9.3] writable foreign tables
Date: 2013-03-03 15:17:26
Message-ID: 15550.1362323846@sss.pgh.pa.us (view raw or flat)
Thread: 2013-03-03 15:17:26 from Tom Lane <tgl(at)sss(dot)pgh(dot)pa(dot)us>
Lists: pgsql-hackers

Craig Ringer <craig(at)2ndquadrant(dot)com> writes:
> On 02/08/2013 01:03 AM, Kohei KaiGai wrote:
>> The attached patch adds Daniel's reworks on make_modifytable
>> invocation, and add a short comment on add_base_rels_to_query(). Rest
>> of portion has not been changed from the previous version.

> How's this looking for 9.3? On-list discussion seems to have been
> positive but inconclusive and time's running out. Do you think this can
> be turned into a production-worthy feature in the next week or two?

I think it needs major changes. The portion against
contrib/postgres_fdw fails to apply at all, of course, but that's my
fault for having hacked so much on postgres_fdw before committing it.
More generally, I don't much like the approach to ctid-substitute
columns --- I think hacking on the rel's tupledesc like that is
guaranteed to break things all over the place. The assorted ugly
kluges that are already in the patch because of it are just scratching
the surface, and there may well be consequences that are flat out
unfixable. Probably the resjunk-columns mechanism would offer a better
solution.

I had hoped to spend several days on this and perhaps get it into
committable shape, because I think this is a pretty significant feature
that will take FDWs over the line from curiosity to useful tool.
However, I've been hoping that for nigh two weeks now and not actually
had any cycles to spend on it ...

regards, tom lane

In response to

- Re: [v9.3] writable foreign tables at 2013-03-03 14:15:57 from Craig Ringer

Responses

- Re: [v9.3] writable foreign tables at 2013-03-04 02:51:06 from Craig Ringer
- Re: [v9.3] writable foreign tables at 2013-03-04 09:05:17 from Kohei KaiGai

PostgreSQLの強化 (2/4) – Background Worker

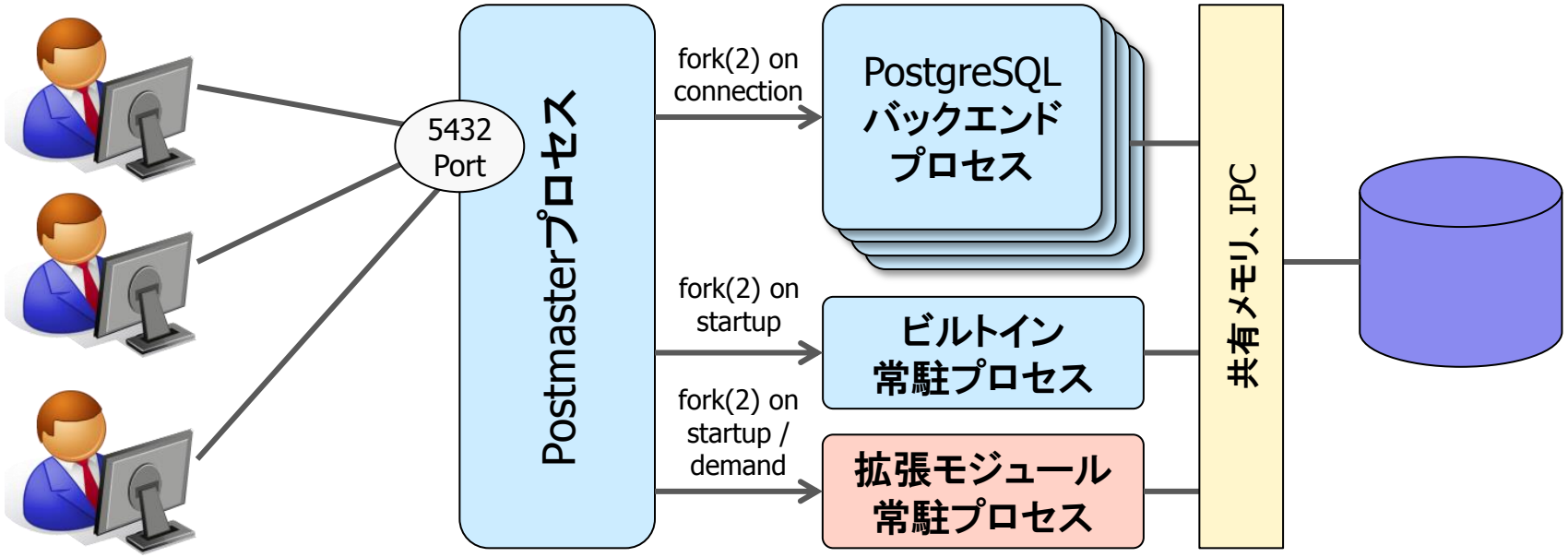
初回クエリが微妙にもっさり

- GPUコードのコンパイルに要する時間 ➔ ビルド結果をキャッシュする
- GPUデバイスの初期化に要する時間 ➔ 常駐プロセスが一度だけ初期化する

➔副産物：複数プロセスの同時利用で “device busy” を返すドライバでも実装可能に

Background Worker

- 拡張モジュールが管理する常駐プロセスを登録可能とするAPI
- v9.4で動的登録も可能に。今後のパラレルクエリ実装の基盤となる機能



設計転換

はたして FDW は適切なフレームワークか？

- 元々、外部のデータソースをテーブルとして見せるための枠組み

[利点]

- 既にPostgreSQLで標準機能化済み

[課題]

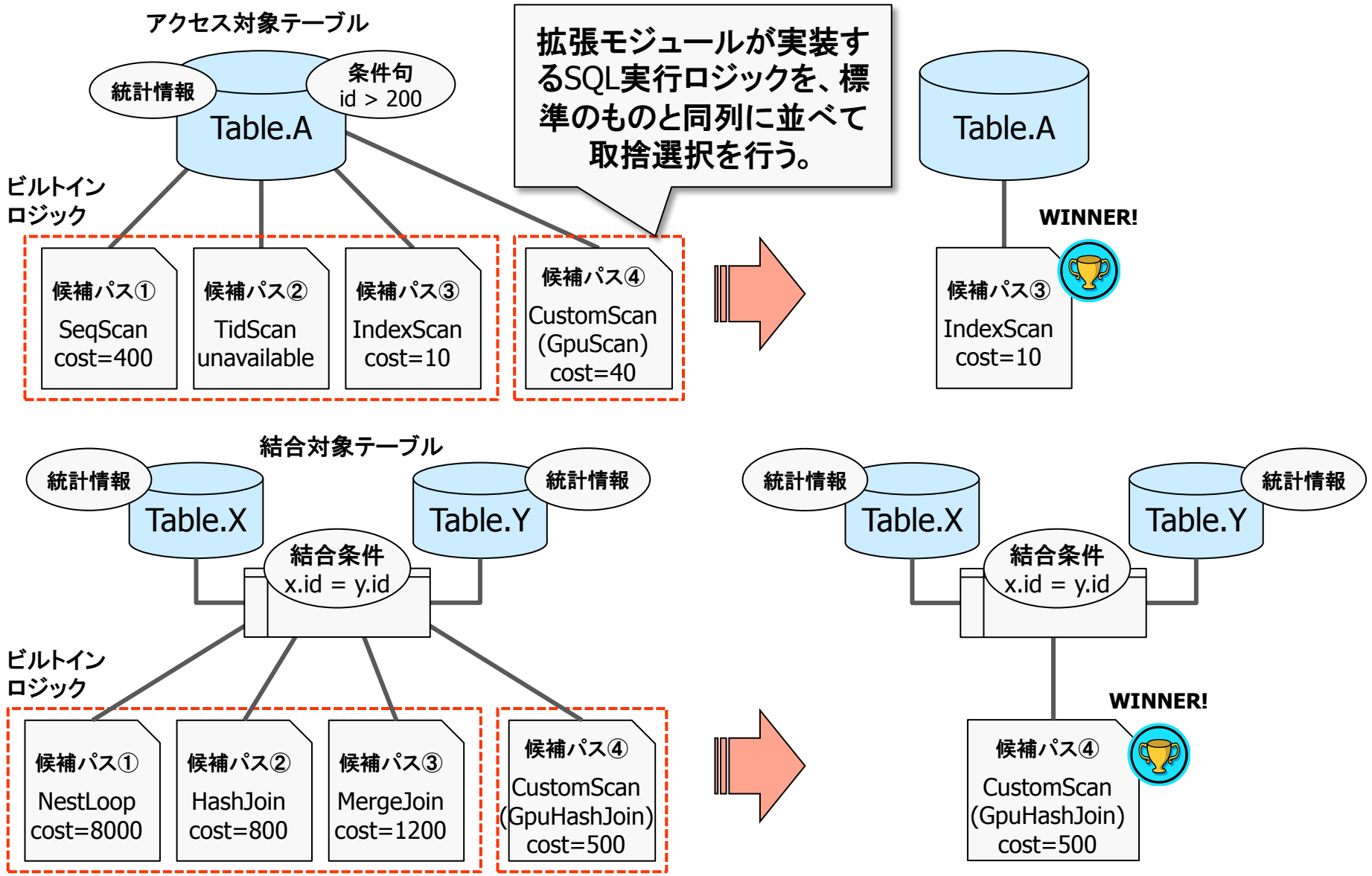
- 利用者、アプリケーションに対する透過性
- GPUを使わない方が実行性能が良いケースへの対応
- 全件スキャン以外のワークロードへの対応

CUDA か OpenCL か？

	CUDA	OpenCL
利点	• 細かな最適化が可能 • NVIDIA GPUの最新機能 • ドライバの実績・安定性	• マルチプラットフォームへの対応 • JITコンパイラが言語仕様に含まれる • CPU並列へも対応が可能
課題	• AMD, Intel環境に対応できない	• ドライバの実績・安定性

➔ 先ずは幅広く使ってもらう事を優先し、OpenCLへ移行

PostgreSQLの強化 (3/4) – Custom-Plan APIs



PostgreSQLの強化 (4/4) – Custom-Plan APIs

Custom-Plan APIs のポイント

- 拡張モジュールの実装するScan/Joinのロジックであっても、標準実装のものと同列にコストベースで取捨選択
- 外部テーブルだけでなく、通常のテーブルに対しても独自実装の処理ロジックを適用可能。

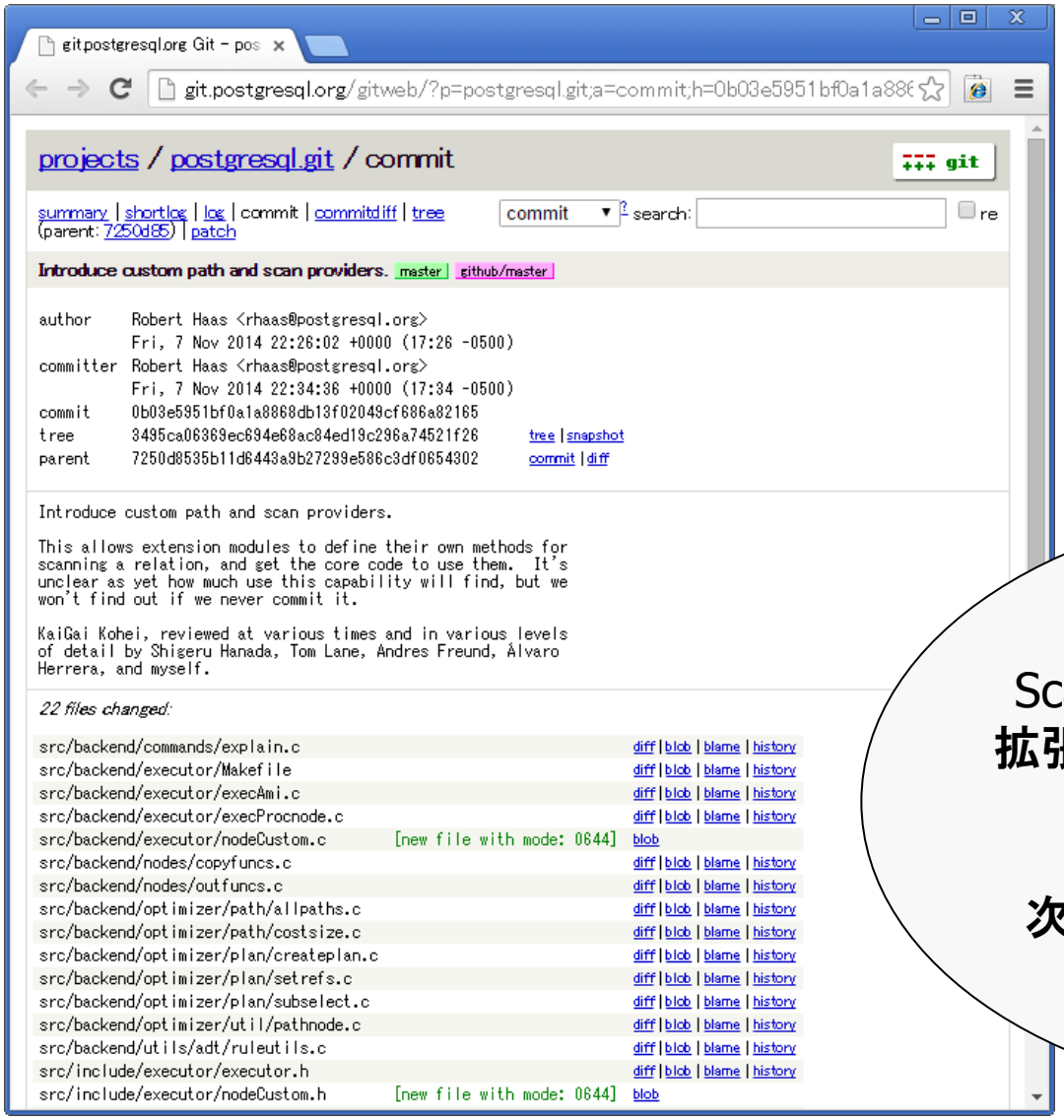
FDWベースの実装に比べたアドバンテージ

- 利用者、アプリケーションに対する透過性
- Index-Scanが有効なケースでGpuScanを無理強いしない
- Joinワークロードへの対応

PostgreSQL v9.5 で標準機能化

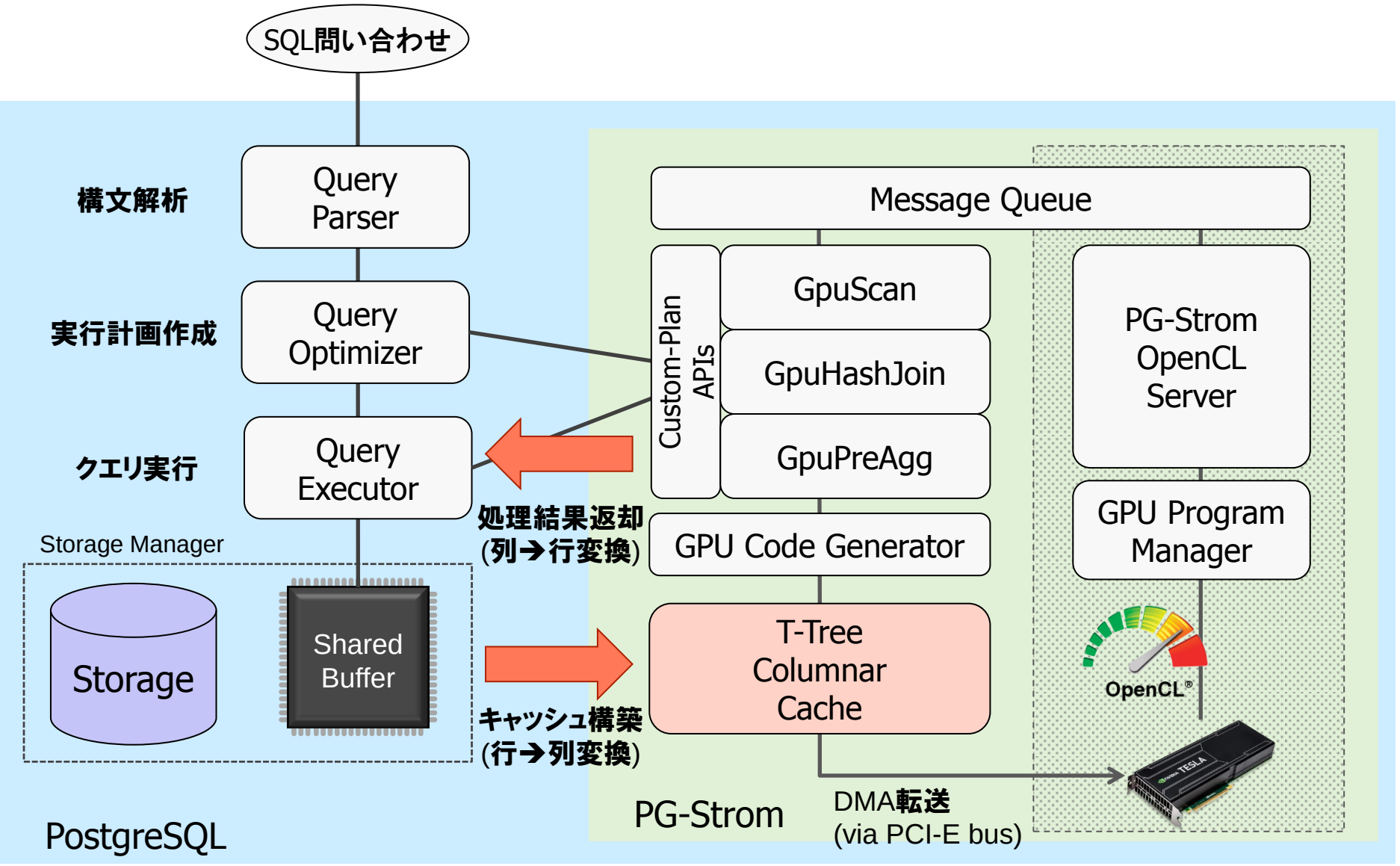
- 11/8(土) にマージされたばかり。

(参考) Custom Plan Interface, ready for v9.5



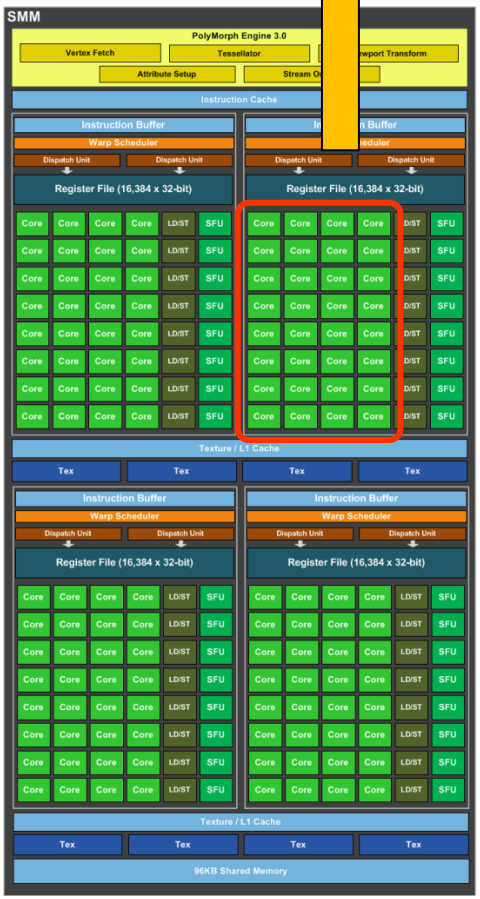
まず、コンセンサスである
Scan部分のカスタムロジックを
拡張モジュールで実装できるよう
インターフェースを定義。
↓
次にJoinのカスタムロジックを
開発者コミュニティで議論

Custom Plan APIs ベースの実装



なぜGPUと列指向データの相性が良いか

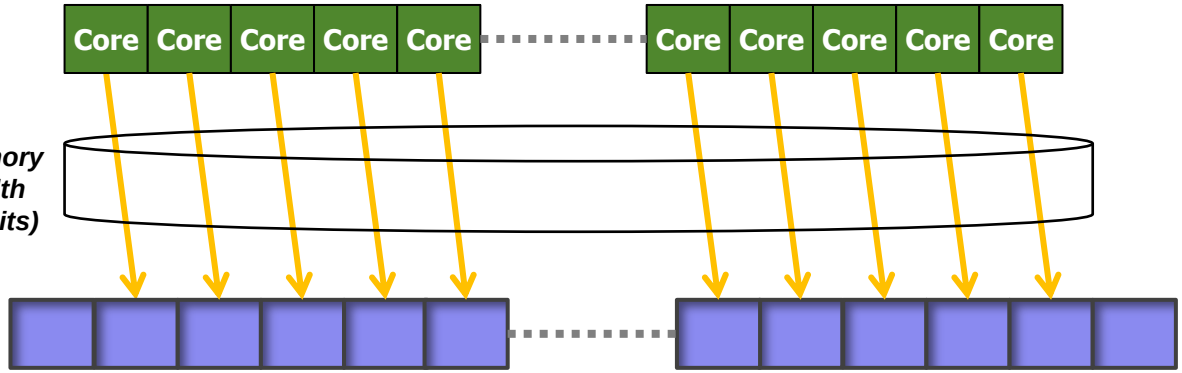
WARP:
命令ポインタを共有する
GPUの命令処理単位。
32個単位である事が多い。



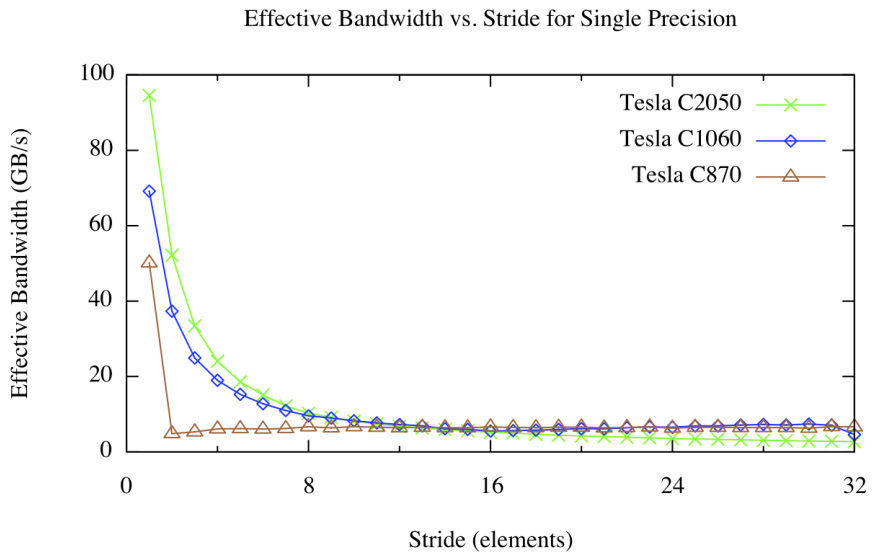
SOURCE: Maxwell: The Most Advanced CUDA GPU Ever Made

coalesced memory access

Wide Memory Bandwidth (256-384bits)



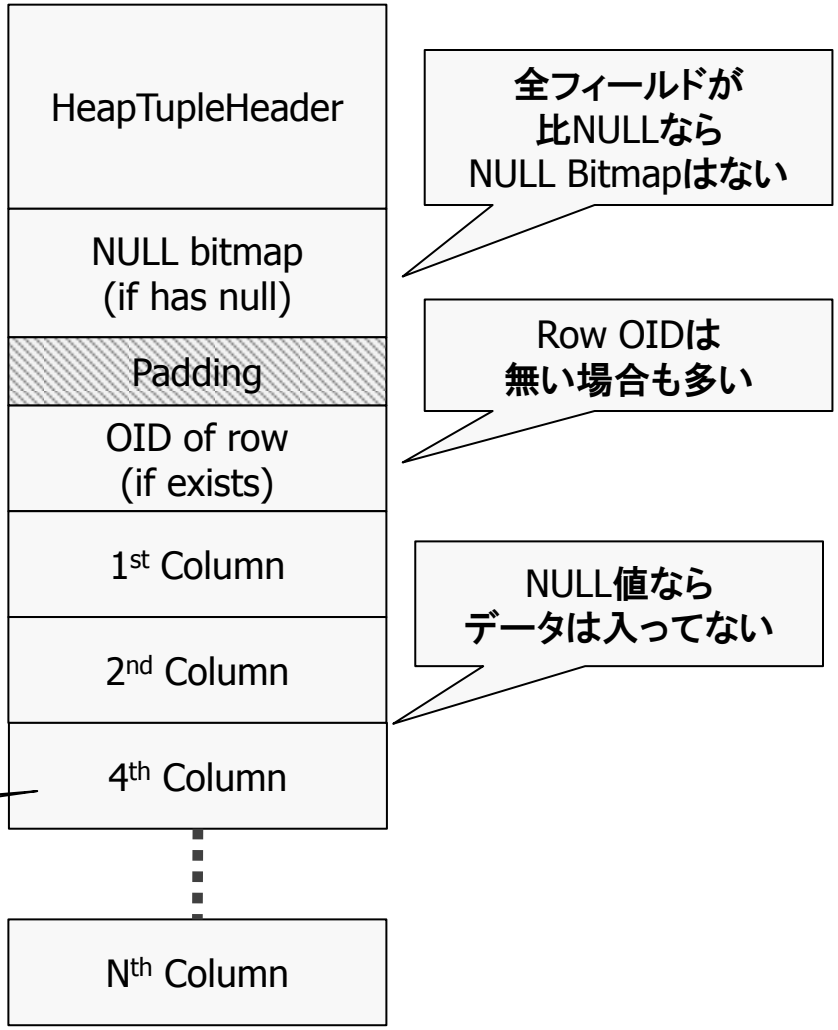
Global Memory (DRAM)



SOURCE: How to Access Global Memory Efficiently in CUDA C/C++ Kernels

PostgreSQLのタプル形式

```
struct HeapTupleHeaderData
{
  union
  {
    HeapTupleFields t_heap;
    DatumTupleFields t_datum;
  }   t_choice;
  /* current TID of this or newer tuple */
  ItemPointerData t_ctid;
  /* number of attributes + various flags */
  uint16      t_infomask2;
  /* various flag bits, see below */
  uint16      t_infomask;
  /* sizeof header incl. bitmap, padding */
  uint8       t_hoff;
  /* ^ - 23 bytes - ^ */
  /* bitmap of NULLs -- VARIABLE LENGTH */
  bits8       t_bits[1];
  /* MORE DATA FOLLOWS AT END OF STRUCT */
};
```



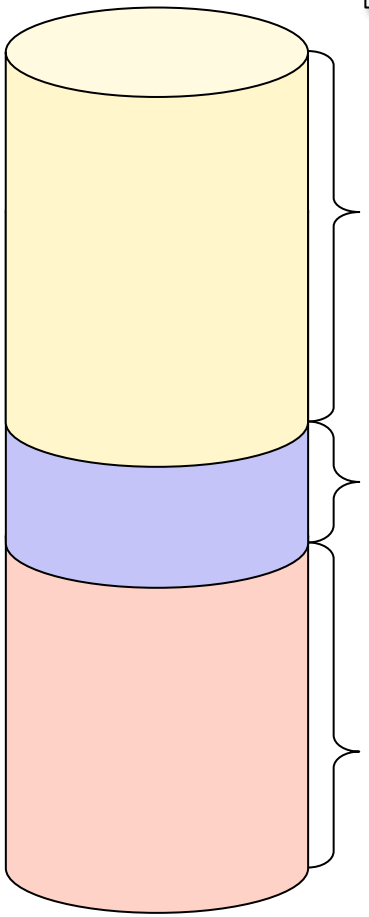
列指向キャッシュの悪夢 (1/2)

```
postgres=# explain (analyze, costs off)
           select * from t0 natural join t1 natural join t2;
                                QUERY PLAN
 ------------------------------------------------------------------------------
Custom (GpuHashJoin) (actual time=54.005..9635.134 rows=20000000 loops=1)
   hash clause 1: (t0.aid = t1.aid)
   hash clause 2: (t0.bid = t2.bid)
   number of requests: 144
   total time to load: 584.67ms
   total time to materialize: 7245.14ms  <-- 全体の70%!!
   average time in send-mq: 37us
   average time in recv-mq: 0us
   max time to build kernel: 1us
   DMA send: 5197.80MB/sec, len: 2166.30MB, time: 416.77ms, count: 470
   DMA recv: 5139.62MB/sec, len: 287.99MB, time: 56.03ms, count: 144
   kernel exec: total: 441.71ms, avg: 3067us, count: 144
   ->  Custom (GpuScan) on t0 (actual time=4.011..584.533 rows=20000000 loops=1)
   ->  Custom (MultiHash) (actual time=31.102..31.102 rows=40000 loops=1)
         hash keys: aid
         ->  Seq Scan on t1 (actual time=0.007..5.062 rows=40000 loops=1)
         ->  Custom (MultiHash) (actual time=17.839..17.839 rows=40000 loops=1)
               hash keys: bid
               ->  Seq Scan on t2 (actual time=0.019..6.794 rows=40000 loops=1)
Execution time: 10525.754 ms
```

列指向キャッシュの悪夢 (2/2)

GPU内のロジックを最適化するために、足回り (= PostgreSQLとのインターフェース部分) で無視できない処理コストが発生している。

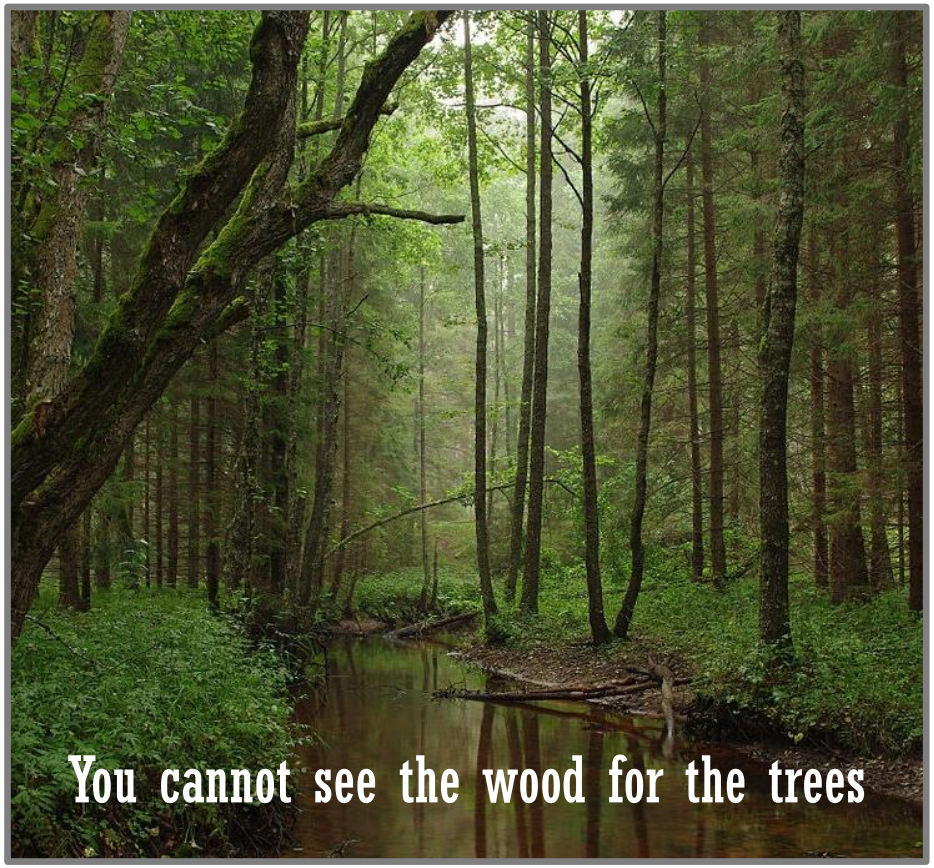
処理時間の内訳



テーブル (行指向データ)
⇩
列指向キャッシュへの変換
（最初の一回だけ）

[Hash-Join処理]
テーブル間で対応する
レコードを探索する処理

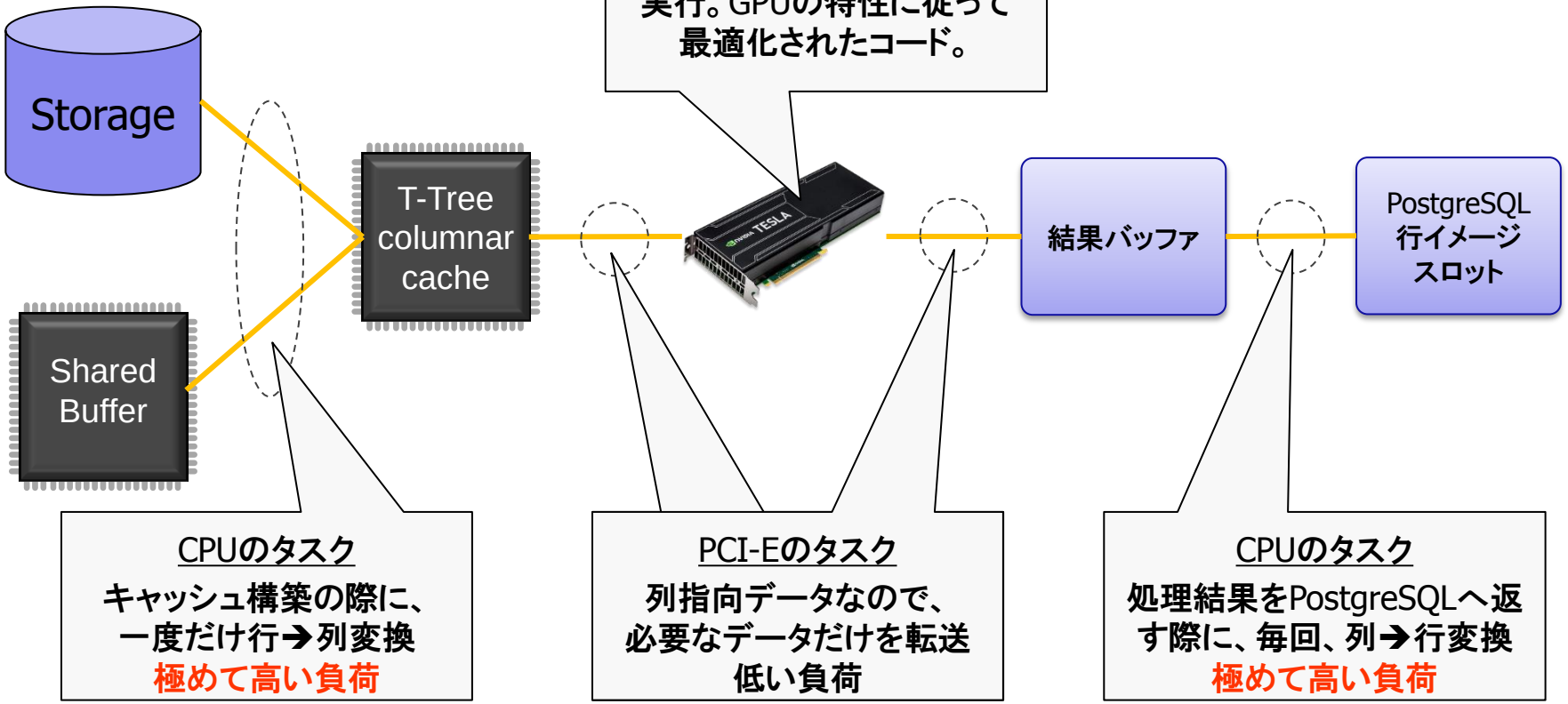
PG-Strom内部形式
(列指向データ)
⇩
PostgreSQLの
データ受渡し形式
(行指向データ)



GPUアクセラレーションのポイント (1/2)

- CPUは希少リソース。いかにCPUに仕事をさせないかがポイント。
- CPUに仕事をさせる場合、メモリ性能を発揮しやすいパターンを意識する。

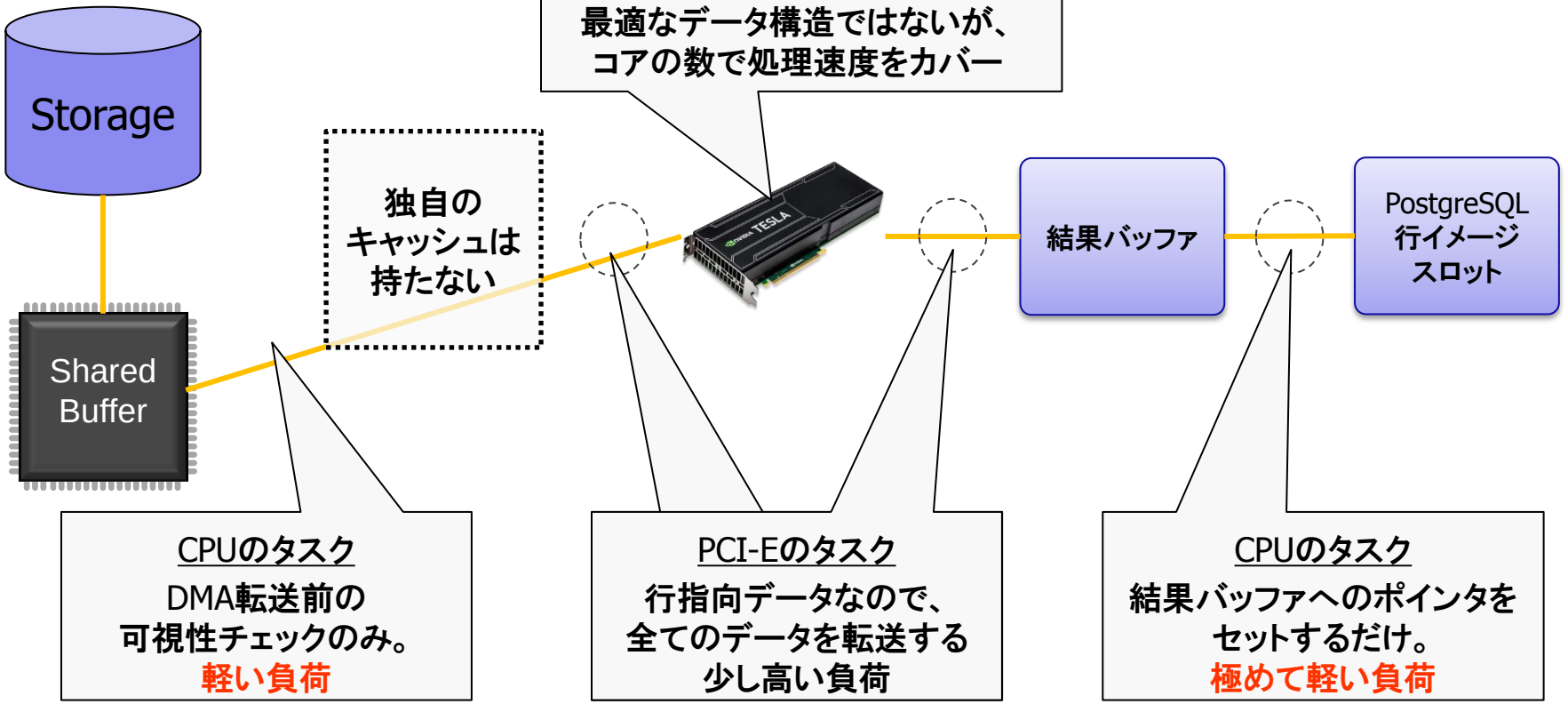
列指向データの利用時



GPUアクセラレーションのポイント (2/2)

- CPUは希少リソース。いかにCPUに仕事をさせないかがポイント。
- CPUに仕事をさせる場合、メモリ性能を発揮しやすいパターンを意識する。

行指向データの利用時



PostgreSQLのshared_bufferと統合した結果

```
postgres=# explain (analyze, costs off)
           select * from t0 natural join t1 natural join t2;
                                    QUERY PLAN
-----------------------------------------------------------------
 Custom (GpuHashJoin) (actual time=111.085..4286.562 rows=20000000 loops=1)
   hash clause 1: (t0.aid = t1.aid)
   hash clause 2: (t0.bid = t2.bid)
   number of requests: 145
   total time for inner load: 29.80ms
   total time for outer load: 812.50ms
   total time to materialize: 1527.95ms  <-- 大幅に削減
   average time in send-mq: 61us
   average time in recv-mq: 0us
   max time to build kernel: 1us
   DMA send: 5198.84MB/sec, len: 2811.40MB, time: 540.77ms, count: 619
   DMA recv: 3769.44MB/sec, len: 2182.02MB, time: 578.87ms, count: 290
   proj kernel exec: total: 264.47ms, avg: 1823us, count: 145
   main kernel exec: total: 622.83ms, avg: 4295us, count: 145
   ->  Custom (GpuScan) on t0 (actual time=5.736..812.255 rows=20000000 loops=1)
   ->  Custom (MultiHash) (actual time=29.766..29.767 rows=80000 loops=1)
         hash keys: aid
         ->  Seq Scan on t1 (actual time=0.005..5.742 rows=40000 loops=1)
         ->  Custom (MultiHash) (actual time=16.552..16.552 rows=40000 loops=1)
               hash keys: bid
               ->  Seq Scan on t2 (actual time=0.022..7.330 rows=40000 loops=1)
 Execution time: 5161.017 ms  <-- 応答性能は大幅改善
```

やや、性能劣化

PG-Stromの現在と今後 (1/2) – PG-Stromなう

対応しているロジック

- GpuScan … 条件句付き全件スキャンのGPU処理
- GpuHashJoin … Hash-JoinのGPU実装
- GpuPreAgg … GPUによる集約関数の前処理

対応しているデータ型

- 整数型 (smallint, integer, bigint)
- 浮動小数点型 (real, float)
- 文字列型 (text, varchar(n), char(n))
- NUMERIC型 (開発中; 昨日動くようになった)

対応している関数

- 各データ型の四則演算オペレータ
- 各データ型の大小比較オペレータ
- 浮動小数点型の数値計算関数
- 集約演算: MIN, MAX, SUM, AVG, 標準偏差, 分散, 共分散

PG-Stromの現在と今後 (2/2) – PG-Stromういる

対応するロジック

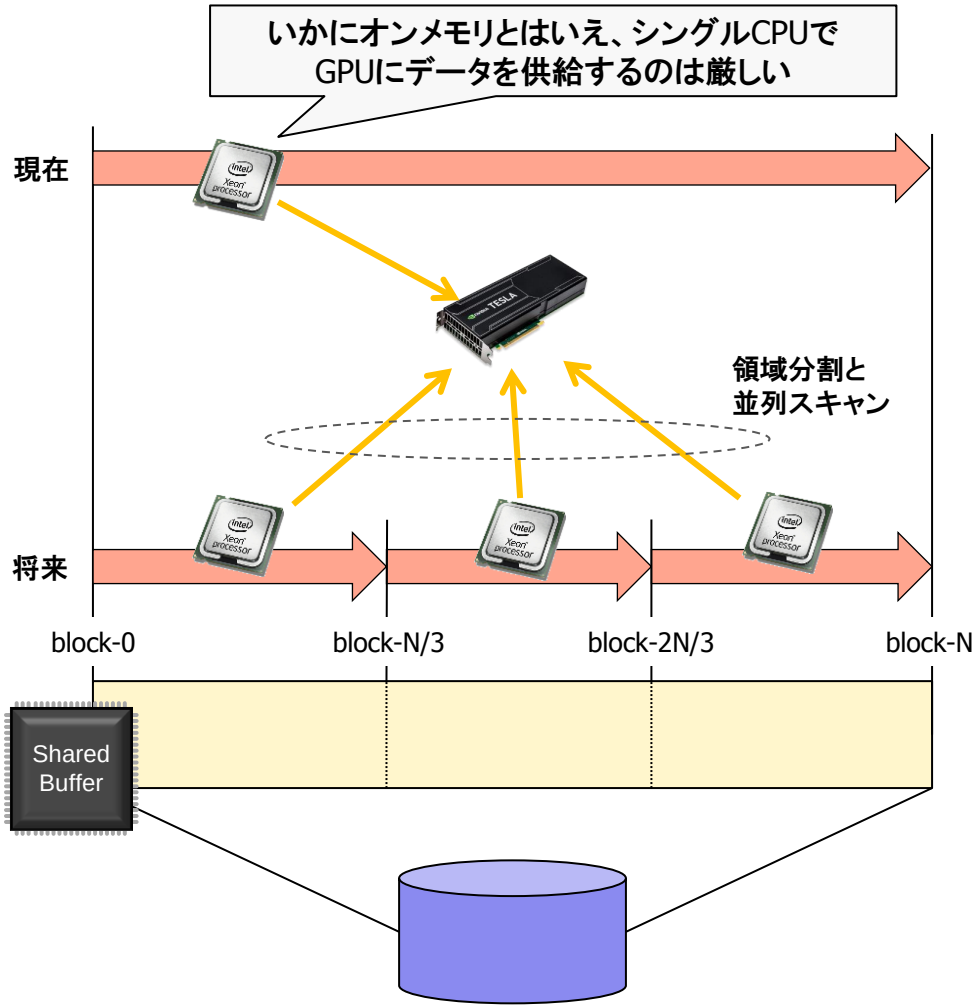
- Outer Join
- Sort
- … その他？

対応するデータ型？

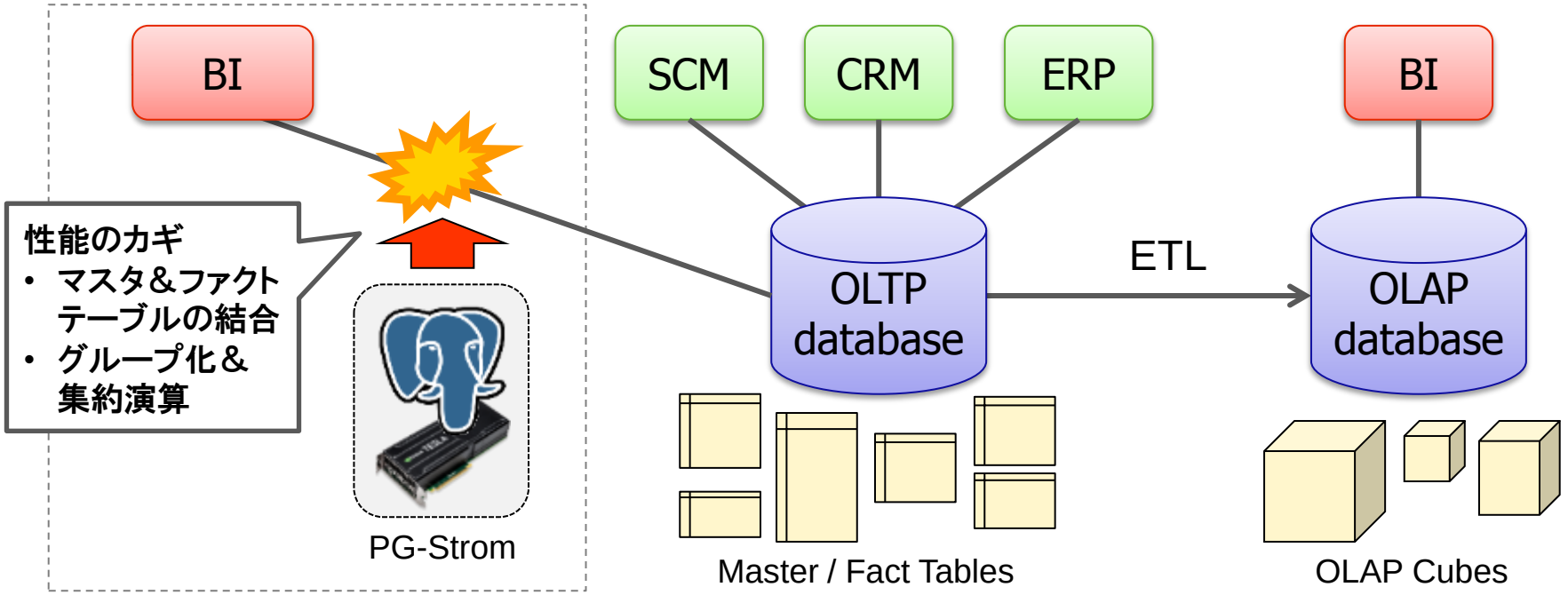
対応する関数？

- LIKE句、正規表現？
- 日付/時刻関数
- PostGIS関数？？
- 幾何関数？
- ユーザ定義関数？

足回りの強化



想定利用シーン (1/3) – OLTP/OLAP統合



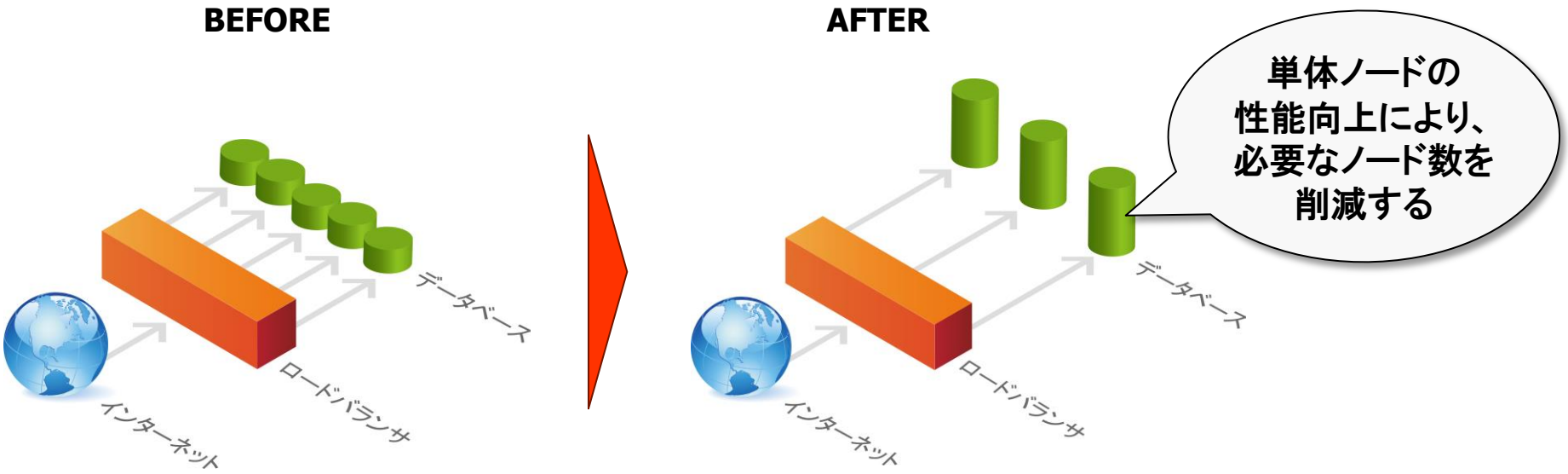
OLTP/OLAPデータベースを分ける理由

- 複数のデータソースを統合する
- 参照系ワークロードへの最適化

PG-Stromの適用により…

- 性能のカギとなるJoinやAggregateを並列処理、リアルタイム化
- OLAPとETLが不要とする事で、システムコストを圧縮

想定利用シーン (2/3) – Webバックエンド



従来の課題

- アクセス集中に備えてWebのバックエンドDBを分散。DBへの参照系負荷を抑える。
- シングルマスタ・レプリケーション構成で、DBノード数が増加。運用コスト増に繋がる。
- 運用中にDBスキーマが変わる事も。適切なインデックスを設定できない事も。

PG-Stromの適用により…

- 単体ノードの処理性能が向上する事で、必要なDBノード数を抑える事ができる。
- インデックスの効きにくい問合せに対しても、コア数の”力技で” 対処する事ができる。

想定利用シーン (3/3) – 我々と一緒に考えませんか？

- どういった領域に適用可能だろうか？
- どういった用途で使えるだろうか？
- どういったワークロードに困っているだろうか？

➔ PG-Stromプロジェクトはフィールドから学びたいと考えています

How to use (1/3) – インストールに必要なのは

- OS: Linux (RHEL 6.x で動作確認)
- PostgreSQL 9.5devel (with Custom-Plan Interface)
- PG-Strom 拡張モジュール
- OpenCL ドライバ (NVIDIAランタイムなど)

PG-Stromを使用するために最低限必要な設定

```
shared_preload_libraries = '$libdir/pg_strom'
shared_buffers = <DBサイズと同程度>
```

実行時に PG-Strom の有効/無効を切り替える

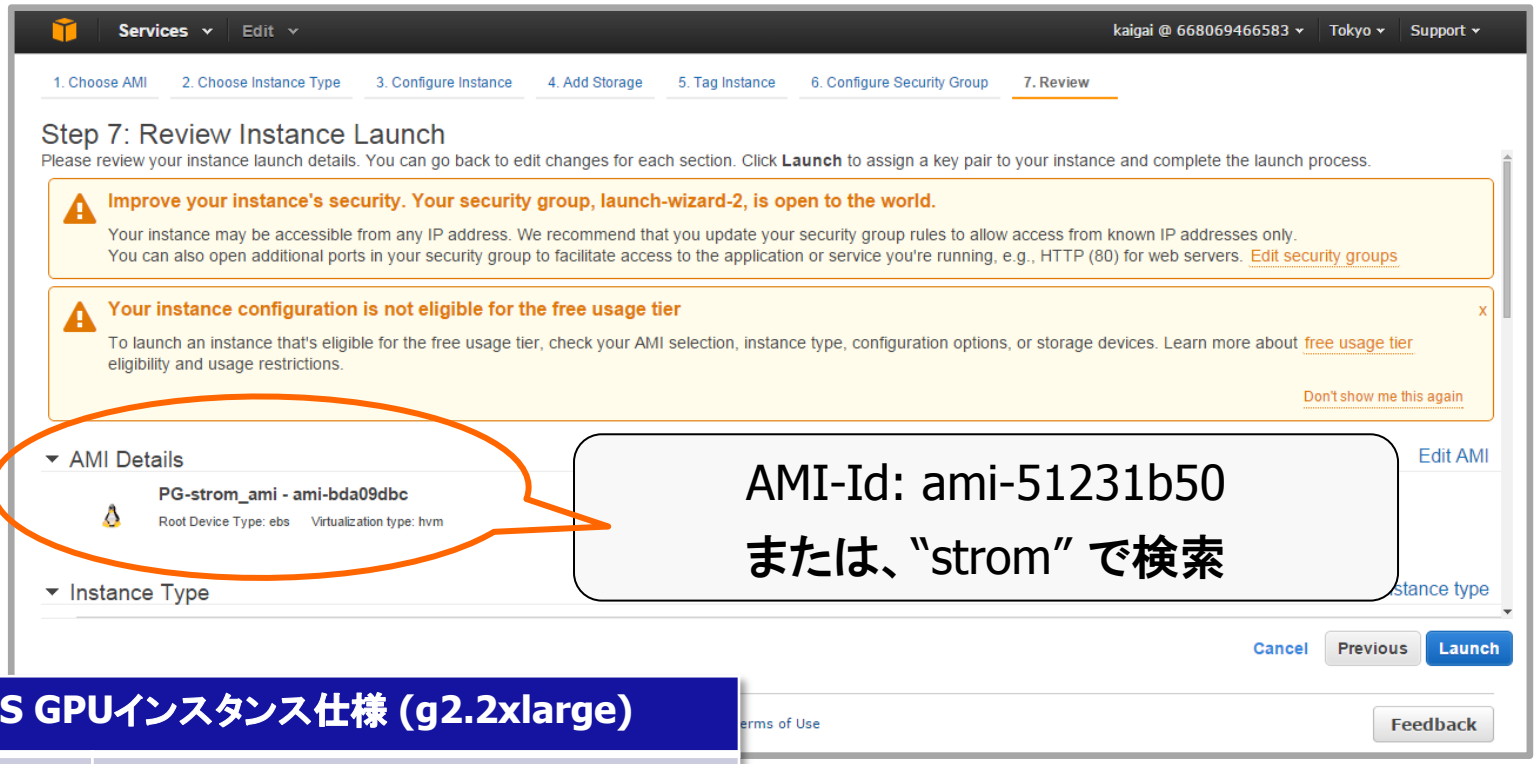
```
postgres=# SET pg_strom.enabled = on;
SET
```

How to use (2/3) – ビルド、インストール、起動

```
[kaigai@saba ~]$ git clone https://github.com/pg-strom/devel.git pg_strom
[kaigai@saba ~]$ cd pg_strom
[kaigai@saba pg_strom]$ make && make install
[kaigai@saba pg_strom]$ vi $PGDATA/postgresql.conf
```

```
[kaigai@saba ~]$ pg_ctl start
server starting
[kaigai@saba ~]$ LOG:  registering background worker "PG-Strom OpenCL Server"
LOG:  starting background worker process "PG-Strom OpenCL Server"
LOG:  database system was shut down at 2014-11-09 17:45:51 JST
LOG:  autovacuum launcher started
LOG:  database system is ready to accept connections
LOG:  PG-Strom: [0] OpenCL Platform: NVIDIA CUDA
LOG:  PG-Strom: (0:0) Device GeForce GTX 980 (1253MHz x 16units, 4095MB)
LOG:  PG-Strom: (0:1) Device GeForce GTX 750 Ti (1110MHz x 5units, 2047MB)
LOG:  PG-Strom: [1] OpenCL Platform: Intel(R) OpenCL
LOG:  PG-Strom: Platform "NVIDIA CUDA (OpenCL 1.1 CUDA 6.5.19)" was installed
LOG:  PG-Strom: Device "GeForce GTX 980" was installed
LOG:  PG-Strom: shmem 0x7f447f6b8000-0x7f46f06b7fff was mapped (len: 10000MB)
LOG:  PG-Strom: buffer 0x7f34592795c0-0x7f44592795bf was mapped (len: 65536MB)
LOG:  Starting PG-Strom OpenCL Server
LOG:  PG-Strom: 24 of server threads are up
```

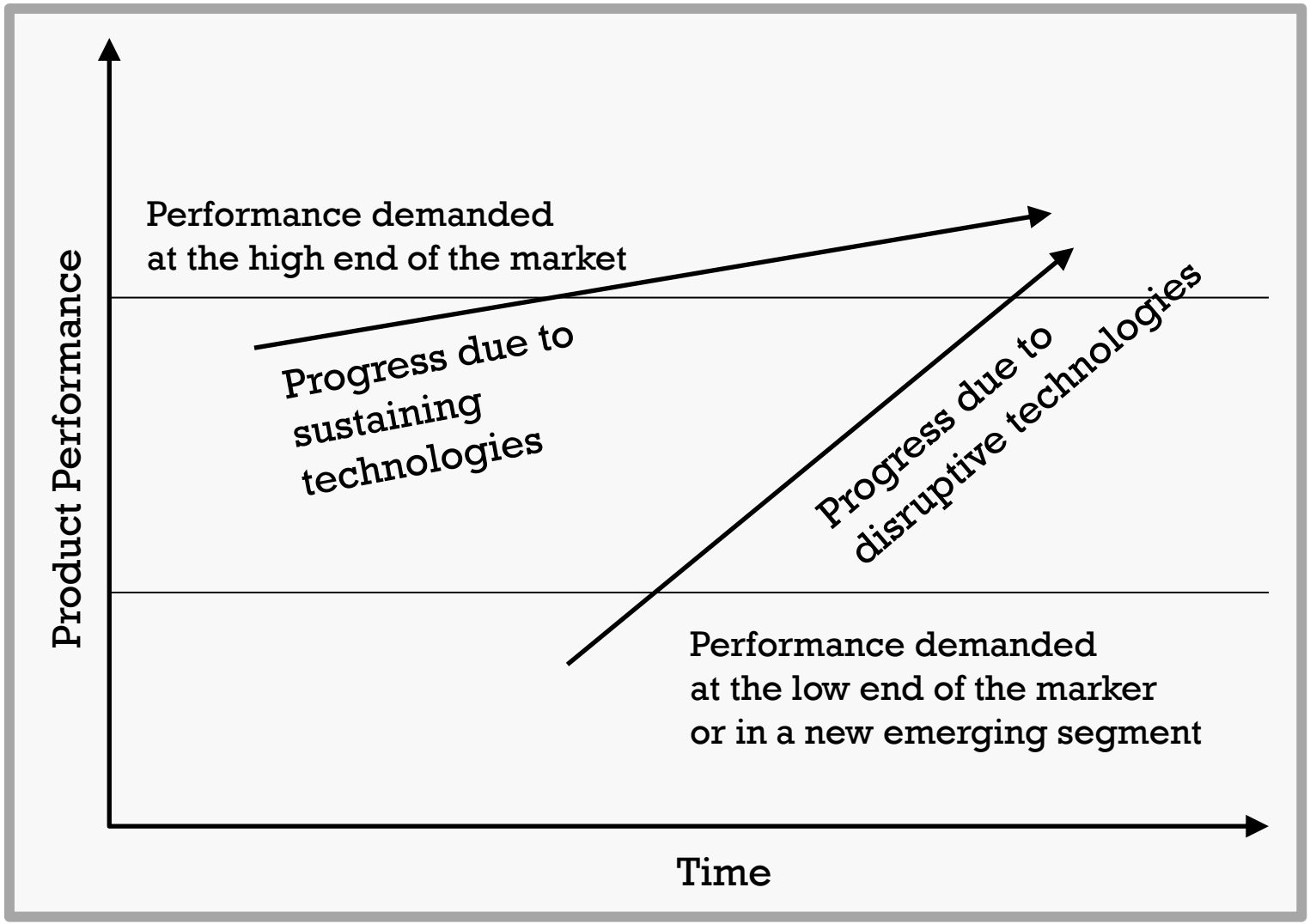
How to use (3/3) – AWSで楽々デプロイ



AWS GPUインスタンス仕様 (g2.2xlarge)	
CPU	Xeon E5-2670 (8 xCPU)
RAM	15GB
GPU	NVIDIA GRID K2 (1536core)
Storage	60GB of SSD
Price	$0.898/hour、$646.56/mon (*) 2014年11月8日現在の東京リージョン オンデマンドインスタンス価格

PG-Stromの今後

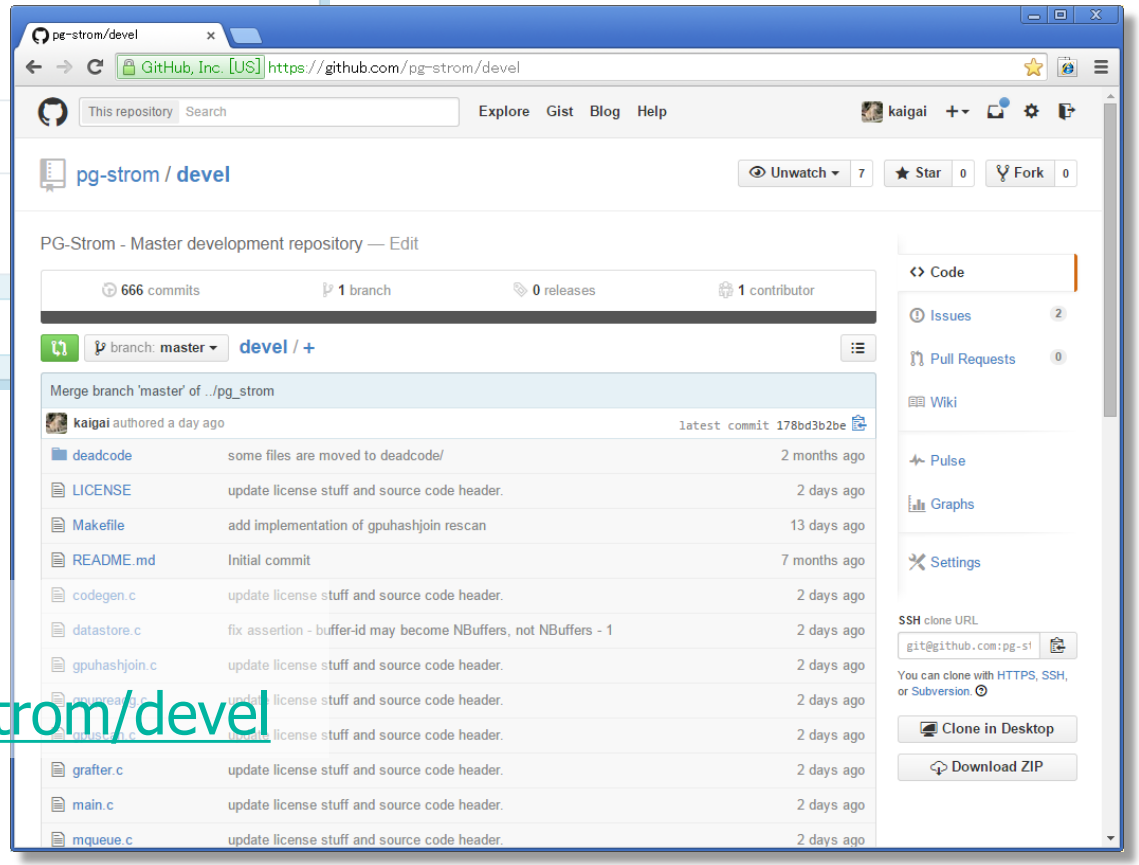
イノベーションのジレンマ



SOURCE: The Innovator's Dilemma, Clayton M. Christensen

コミュニティと共に進む

（発表後の補足）



check it out!

https://github.com/pg-strom/devel

Orchestrating a brighter world

世界の想いを、未来へつなげる。

未来に向かい、人が生きる、豊かに生きるために欠かせないもの。
それは「安全」「安心」「効率」「公平」という価値が実現された社会です。

NECは、ネットワーク技術とコンピューティング技術をあわせ持つ
類のないインテグレーターとしてリーダーシップを発揮し、
卓越した技術とさまざまな知見やアイデアを融合することで、
世界の国々や地域の人々と協奏しながら、
明るく希望に満ちた暮らしと社会を実現し、未来につなげていきます。

Empowered by Innovation