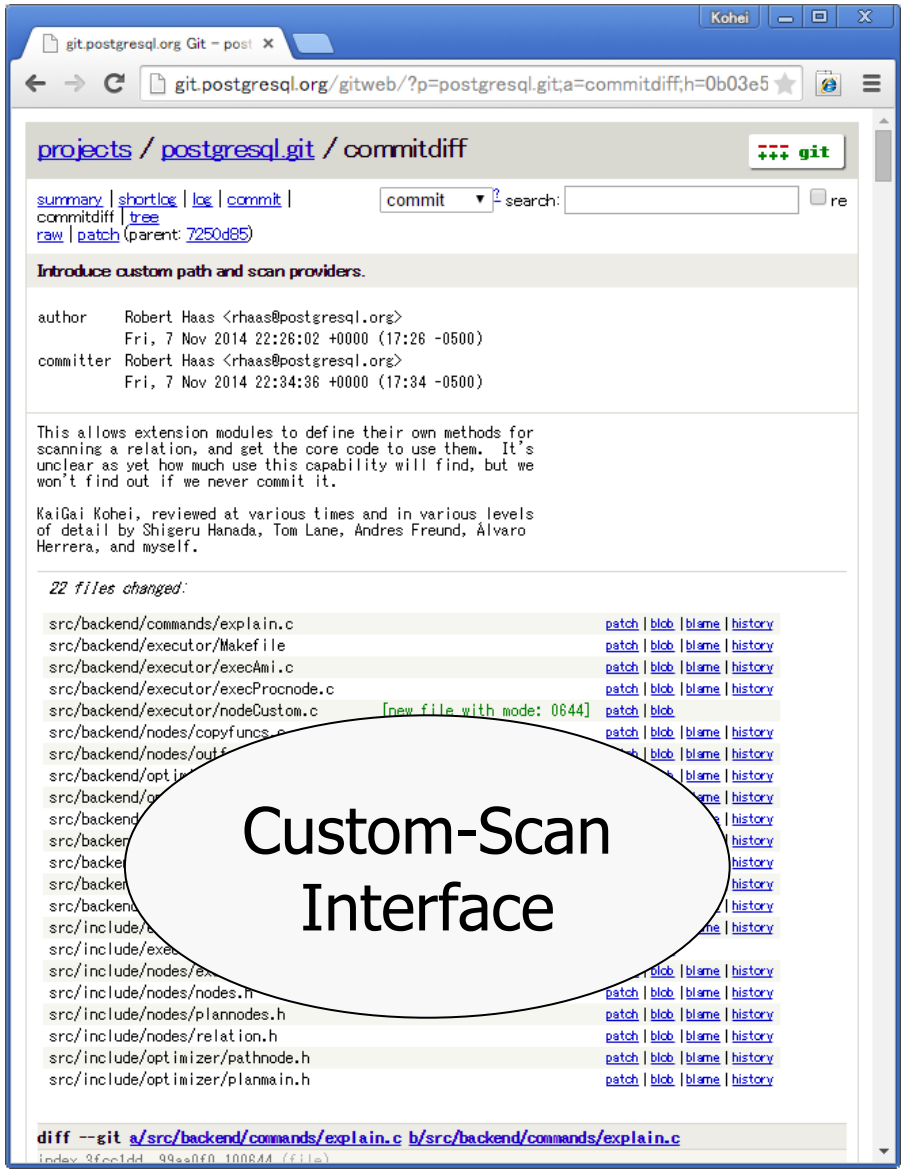
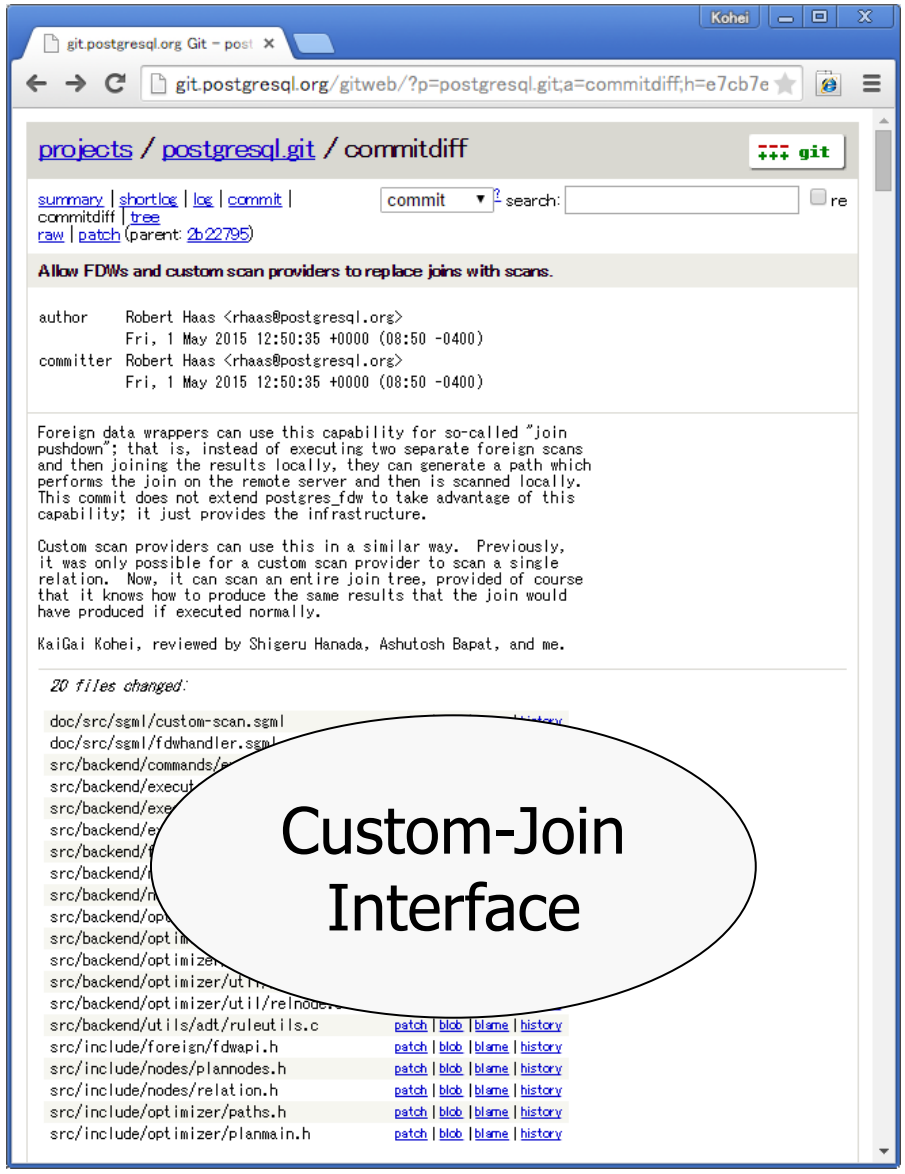
v9.5の新機能
Custom Scan/Join Interface

KaiGai Kohei <kaigai@kaigai.gr.jp>

v9.5 New Feature





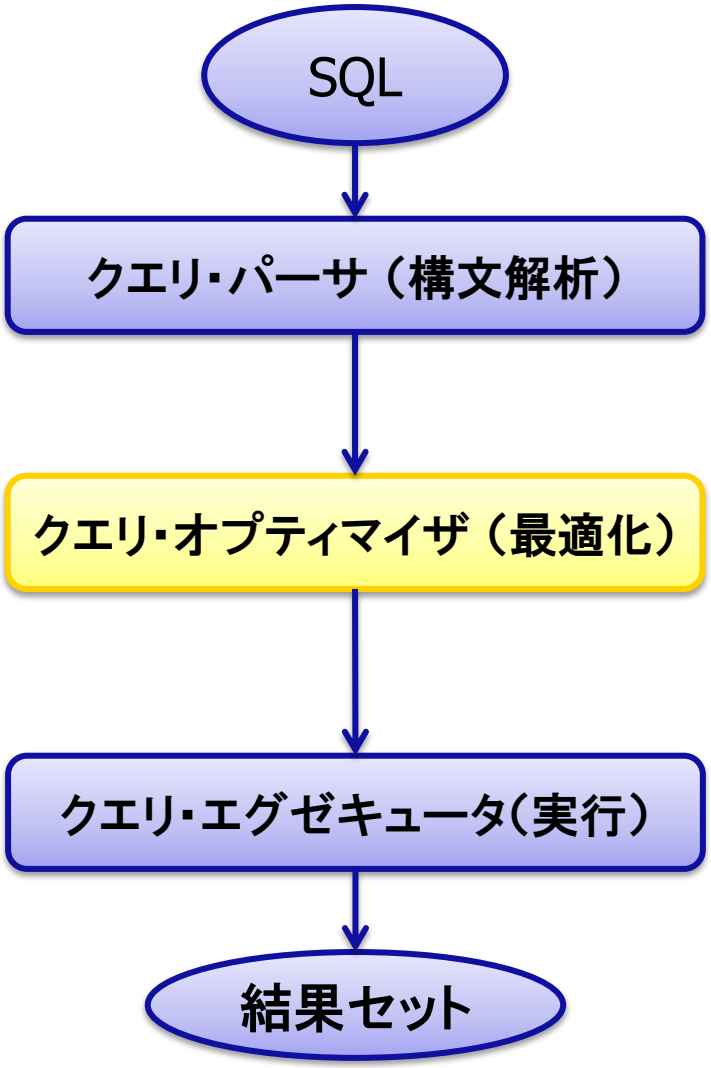
Custom Scan/Join Interfaceが実現する事

- クエリ実行計画の一部を、拡張モジュール実装の "何か" で置き換える
- 何を置き換える事ができるのか？
  - Scan ... set_rel_pathlist_hook で候補パス追加
  - Join ... set_join_pathlist_hook で候補パス追加
  - Others ... planner_hookを使ってインジェクションすれば何でもアリ。
    - ➔ 局所最適にならないよう、注意は必要だが…。
- 応用例
  - GPUアクセラレーションによる全体的なRDBMS性能の底上げ。
    - ➔ PG-Strom (オタワで発表するんでヨロ)
  - 特定のテーブル同士のJoinをMaterialized Viewスキャンに透過的に置換え。
  - 特定のテーブルのうち、参照頻度の高い列だけを列指向形式でキャッシュ
  - Scan/Joinの中間結果から統計情報を採取。値によるヒストグラム作成。
  - ……など

クエリ実行計画

```
postgres=# EXPLAIN SELECT cat, AVG(x)
    FROM t0 NATURAL JOIN t1 RIGHT JOIN t2 ON t0.aid=t2.bid OR t0.bid=t2.bid
    WHERE atext like '%abc%' GROUP BY cat;
                                QUERY PLAN
-------------------------------------------------------------------------------
 HashAggregate  (cost=979149.25..979149.58 rows=26 width=12)
 Group Key: t0.cat
 -> Nested Loop  (cost=834.05..979139.33 rows=1985 width=12)
    Join Filter: ((t0.aid = t2.bid) OR (t0.bid = t2.bid))
    -> Seq Scan on t2  (cost=0.00..734.00 rows=40000 width=4)
    -> Materialize  (cost=834.05..281207.82 rows=996 width=20)
       -> Hash Join  (cost=834.05..281202.84 rows=996 width=20)
          Hash Cond: (t0.aid = t1.aid)
          -> Seq Scan on t0  (cost=0.00..242858.60 rows=10000060 width=20)
          -> Hash  (cost=834.00..834.00 rows=4 width=4)
             -> Seq Scan on t1  (cost=0.00..834.00 rows=4 width=4)
                Filter: (atext ~~ '%abc%'::text)
(12 rows)
```

PostgreSQL クエリ処理の流れ (1/2)

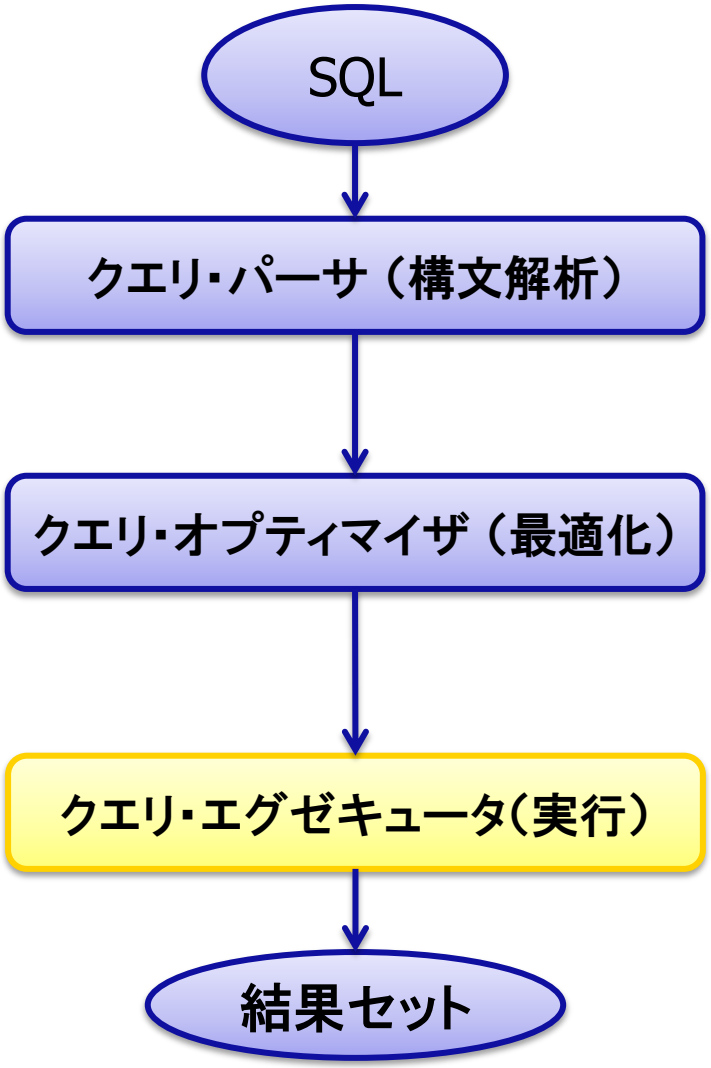


- リレーションに対する候補パスを列挙
  - Base Relation
  - Join Relation
- プラナーが最も実行コストの小さなパスを選択。

➔全てのテーブルを単一のJoin-Treeに収めるまで結合順を試行錯誤

- 実行パスを元に、実行計画を作成

PostgreSQL クエリ処理の流れ (2/2)



ExecInitNode

- 実行ノードの初期化

ExecProcNode

- 一行を生成して TupleTableSlot を返す

ExecEndNode

- 実行ノードの終了処理

その他

- ExecReScan
- ExecMarkPos (optional)
- ExecRestrPos (optional)

Custom Scan/Joinを使うには (1/3)

候補パスを追加する

```
typedef void (*set_rel_pathlist_hook_type) (PlannerInfo *root,
                                            RelOptInfo *rel,
                                            Index rti,
                                            RangeTblEntry *rte);
extern PGDLLIMPORT set_rel_pathlist_hook_type set_rel_pathlist_hook;

typedef void (*set_join_pathlist_hook_type) (PlannerInfo *root,
                                             RelOptInfo *joinrel,
                                             RelOptInfo *outerrel,
                                             RelOptInfo *innerrel,
                                             JoinType jointype,
                                             JoinPathExtraData *extra);
extern PGDLLIMPORT set_join_pathlist_hook_type set_join_pathlist_hook;
```

CustomPath 構造体

```
typedef struct CustomPath
{
    Path        path;                   // Path 共通部分
    uint32      flags;                  // CUSTOMPATH_* flags
    List       *custom_private;         // プライベート領域
    const CustomPathMethods *methods;   // コールバック関数
} CustomPath;
```

Custom Scan/Joinを使うには (2/3)

CustomPath ➔ CustomScan

- プラナーがCustomPathを選択

➔CustomScanノードを生成する必要がある

どういう事か？

- CustomPathから生成されるのは、CustomScanだけではない。(将来的に)
  - ✓CustomSort、CustomAgg、…
- プラン木はcopyObject()可能である事

➔GpuScanの例

```
/* Path information of GpuScan */
typedef struct {
    CustomPath  cpath;
    /* RestrictInfo run on host */
    List       *host_quals;
    /* RestrictInfo run on devices */
    List       *dev_quals;
} GpuScanPath;
```

CustomScan ➔ CustomScanState

- エグゼキュータの開始時に、CustomScanノードの内容に基づいてCustomScanStateを初期化。

➔GpuScanの例　CustomScanStateを継承

```
typedef struct {
    GpuTaskState    gts;

    BlockNumber     curr_blknum;
    BlockNumber     last_blknum;
    HeapTupleData   scan_tuple;
    List           *dev_quals;

    cl_uint         num_rechecked;
} GpuScanState;
```

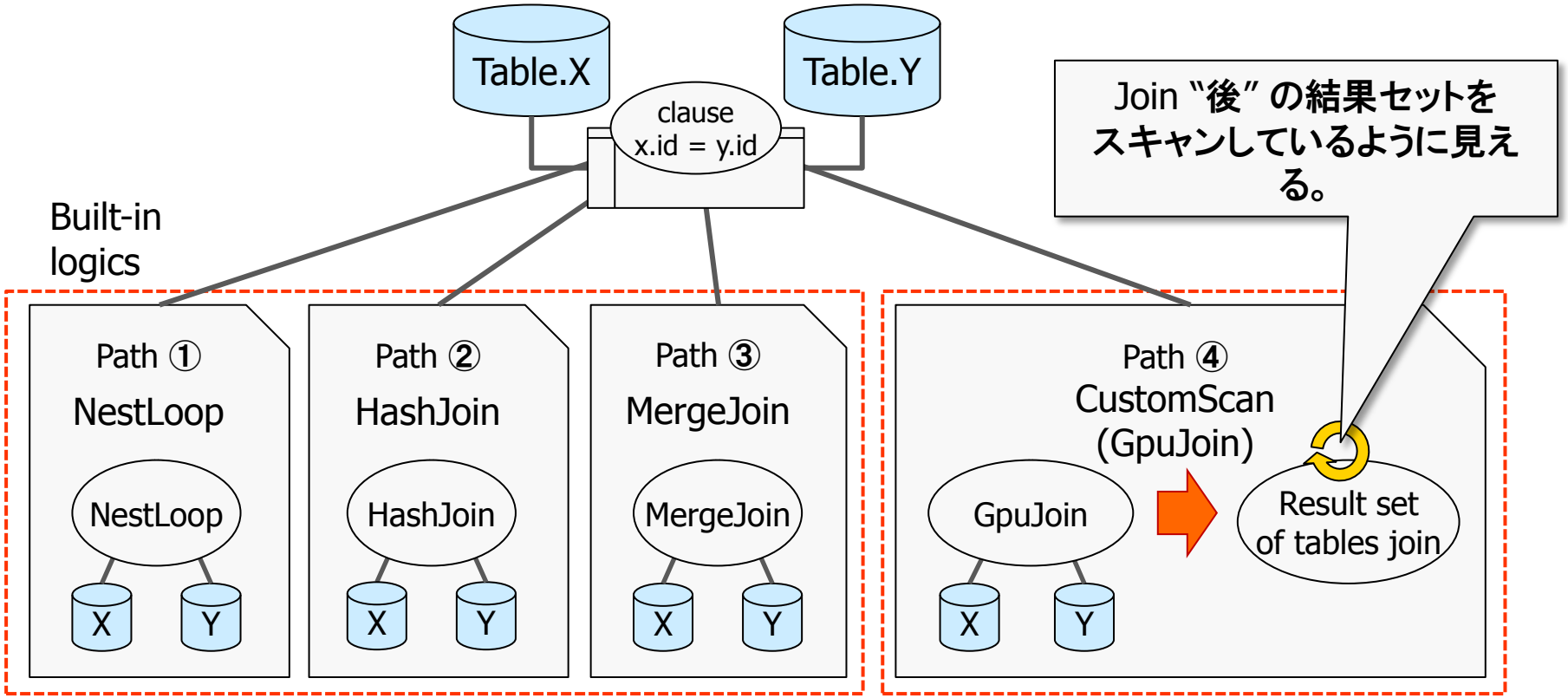
Custom Scan/Joinを使うには (3/3)

Executorコールバックを各自実装する

```
typedef struct CustomExecMethods
{
    const char *CustomName;
    /* Executor methods: mark/restore are optional, the rest are required */
    void        (*BeginCustomScan) (struct CustomScanState *node,
                                    EState *estate,
                                    int eflags);
    TupleTableSlot *(*ExecCustomScan) (struct CustomScanState *node);
    void        (*EndCustomScan) (struct CustomScanState *node);
    void        (*ReScanCustomScan) (struct CustomScanState *node);
    void        (*MarkPosCustomScan) (struct CustomScanState *node);
    void        (*RestrPosCustomScan) (struct CustomScanState *node);
    /* Optional: print additional information in EXPLAIN */
    void        (*ExplainCustomScan) (struct CustomScanState *node,
                                      List *ancestors,
                                      struct ExplainState *es);
} CustomExecMethods;
```

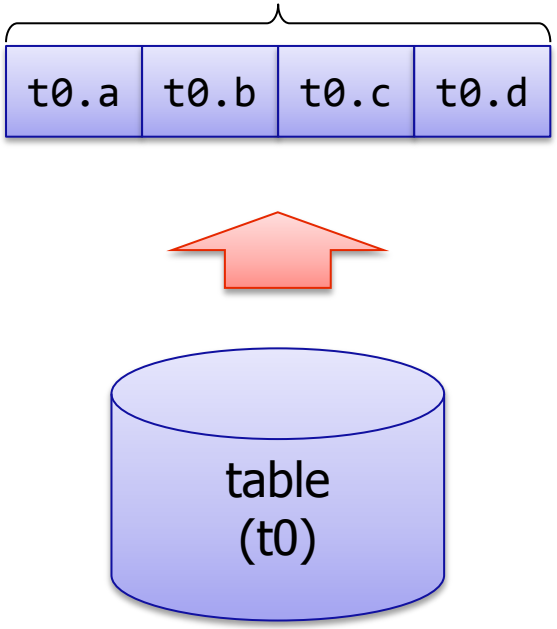
CustomJoinはどう見えるか？

CustomScanによる Join の置換え



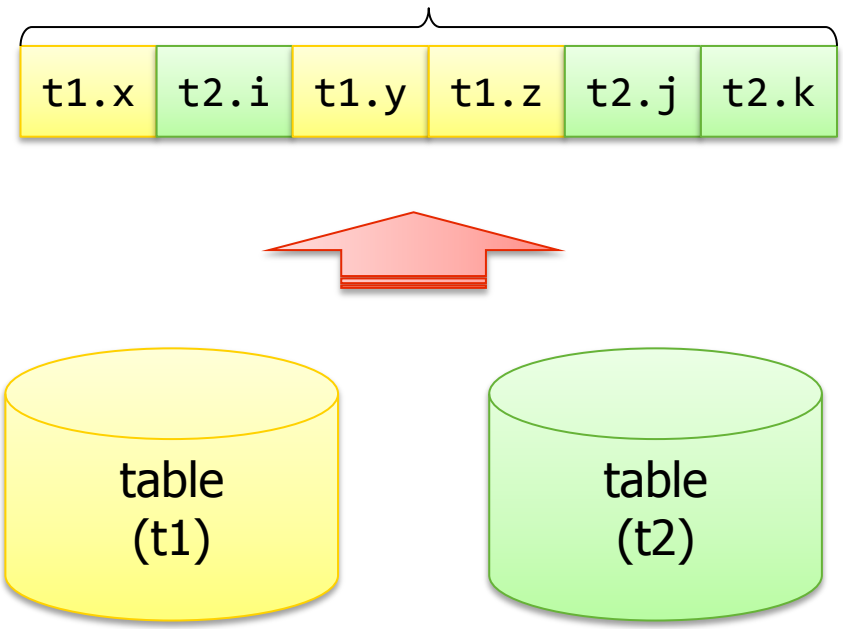
結果セットのレコード型

結果セットのレコード型は、
常にテーブル定義と一致



単純なスキャン

結果セットのレコード型は、
時と場合によって異なる

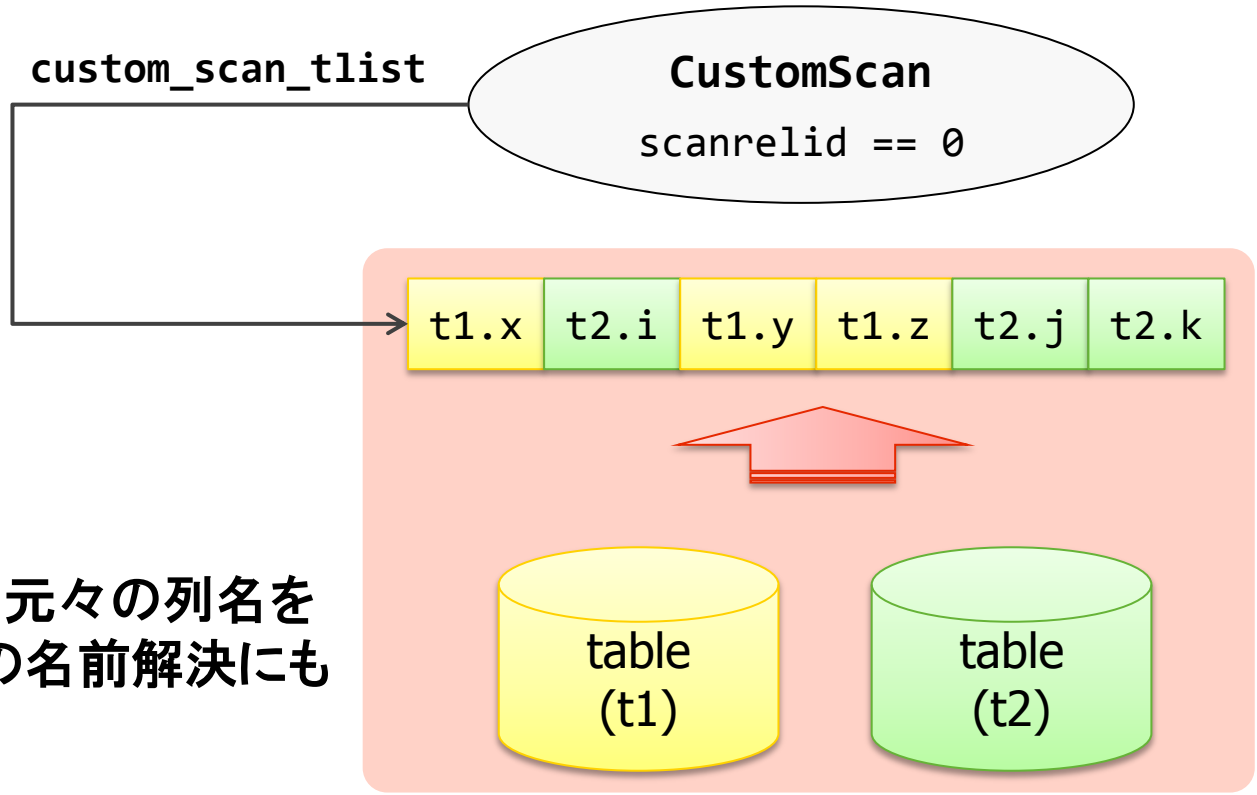


複数のテーブルから成るジョイン

scanrelid == 0 と custom_scan_tlist

scanrelid == 0

- CustomScanが特定の実テーブルスキャンでない事を示す Magic Number
- custom_scan_tlist で指定されたレコード型をスキャンすると見なす。

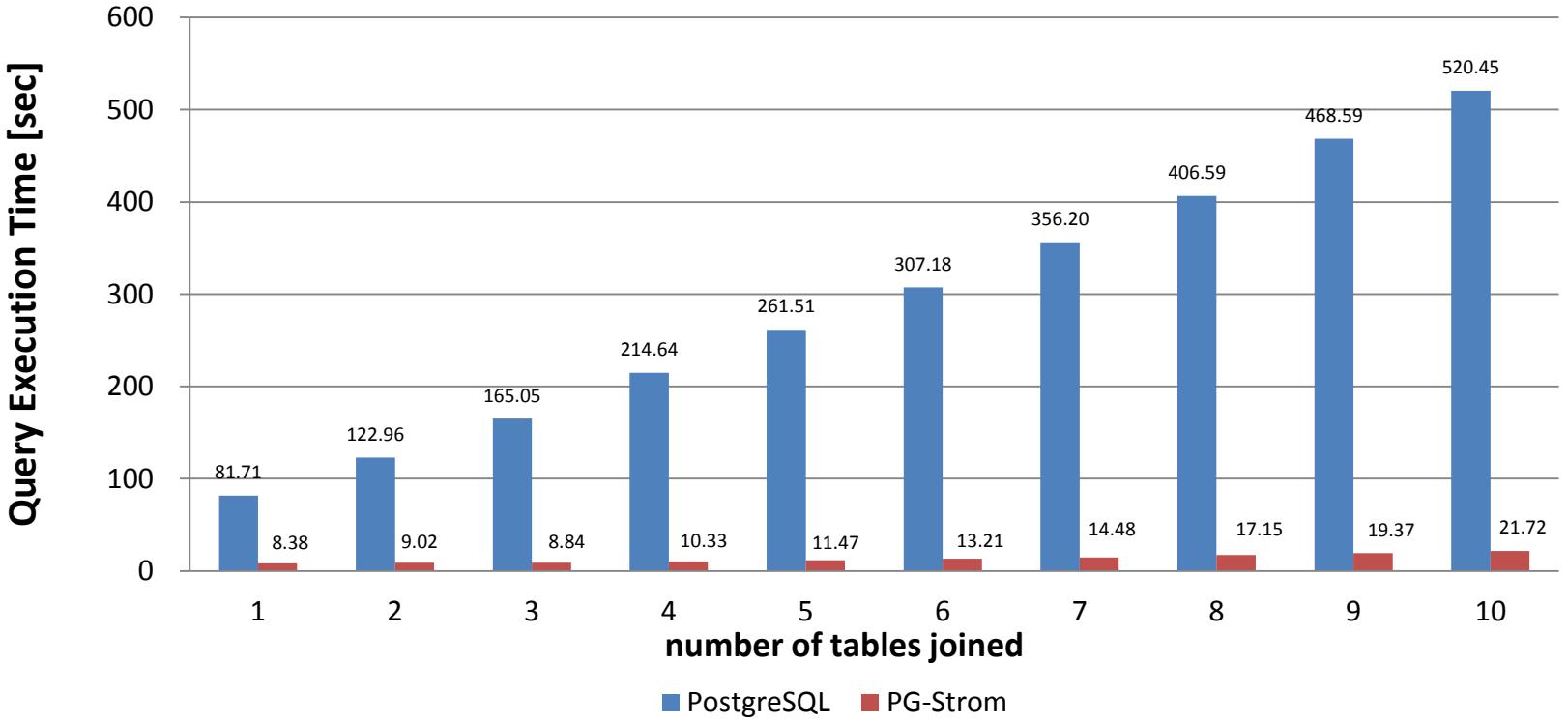


- EXPLAINで、元々の列名を表示する際の名前解決にも使用される。

CustomScanを含むクエリ実行計画

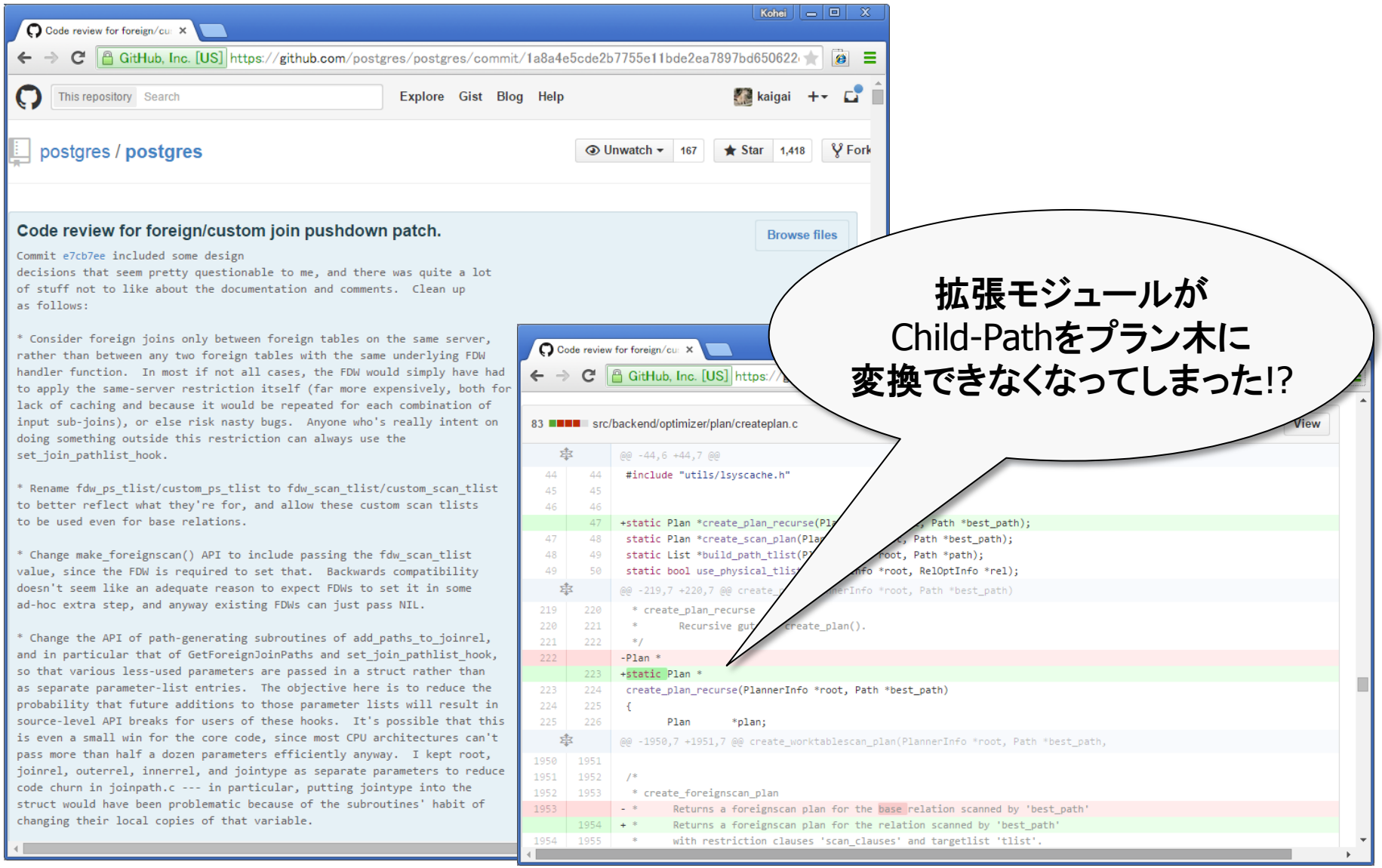
```
postgres=# EXPLAIN SELECT cat, AVG(x)
    FROM t0 NATURAL JOIN t1 RIGHT JOIN t2 ON t0.aid=t2.bid OR t0.bid=t2.bid
    WHERE atext like '%abc%' GROUP BY cat;
                                    QUERY PLAN
-------------------------------------------------------------------------------
 HashAggregate  (cost=348362.37..348362.70 rows=26 width=12)
  Group Key: t0.cat
  -> Custom Scan (GpuPreAgg) (cost=2134.00..348362.24 rows=26 width=52)
     Bulkload: On
     Reduction: Local + Global
     -> Custom Scan (GpuJoin)  (cost=1134.00..347362.19 rows=1985 width=12)
        Bulkload: On
        Depth 1: Logic: GpuHashJoin, HashKeys: (aid), JoinQual: (aid = aid)
        Depth 2: Logic: GpuNestLoop, JoinQual: ((aid = bid) OR (bid = bid))
        -> Custom Scan (GpuScan) on t0 (cost=1000.00..143858.00 rows=10000060 ...
        -> Custom Scan (MultiRels) (cost=834.05..343937.39 rows=4 width=4)
           Hash keys: aid
           nBatches: 1,  Buckets: 1024, Buffer Usage: 2.33%
           -> Seq Scan on t1  (cost=0.00..834.00 rows=4 width=4)
              Filter: (atext ~~ '%abc%'::text)
           -> Custom Scan (MultiRels) (cost=1134.00..347362.19 rows=40000 width=4)
              nBatches: 1, Buffer Usage: 97.64%
              ->  Seq Scan on t2  (cost=0.00..734.00 rows=40000 width=4)
```

CustomにJoinを実装してみた結果



- `SELECT cat, AVG(x) FROM t0 NATURAL JOIN t1 [, ...] GROUP BY cat;` をテーブル数を変えながら実行してクエリ応答時間を測定。
- `t0`:1億行、`t1~t10`:それぞれ10万行を含む。全データはバッファにロード済み。
- PostgreSQL v9.5devel + PG-Strom 5/26開発版、on CUDA7 (x86_64)
- CPU: Xeon E5-2640, RAM: 256GB, GPU: NVIDIA GTX980, OS: RedHat EL7

v9.5に向けての Open Issue



もふっ、もふもふ。