Greenplum for PostgreSQL hackers

Heikki Linnakangas / Pivotal

Greenplum

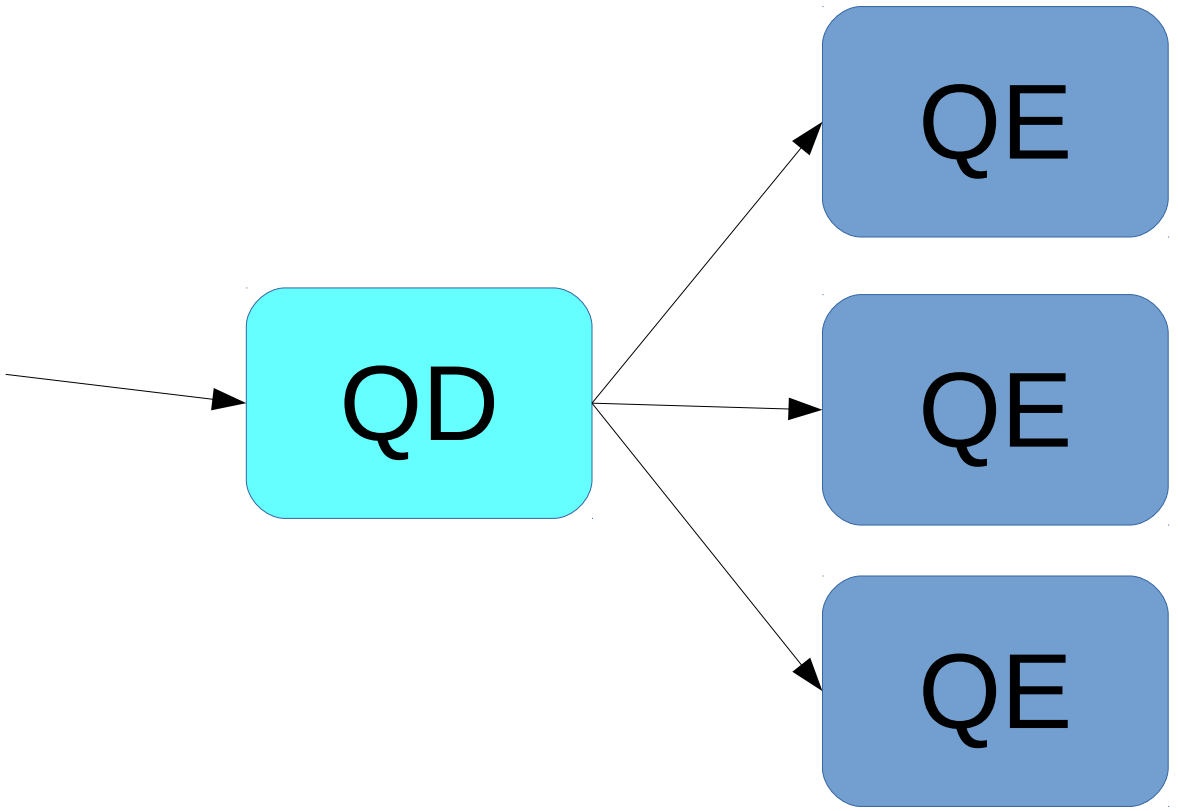
Greenplum is PostgreSQL

- with Massively Parallel Processing bolted on
- with lots other extra features and addons

Greenplum is an Open Source Project

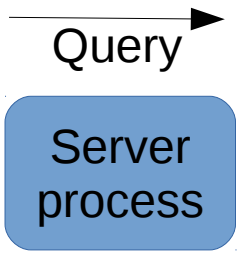
How MPP works?

Query Dispatching



QD = Query Dispatcher

QE = Query Executor



QD/QE

- Query Dispatcher (aka *master*)
  - Accept client connections
  - Manage distributed transactions
  - Create query plans
- Query Executor (aka *segment*)
  - Store the data
  - Execute slices of the query

Deployment

- One QD, as many QE nodes as needed
- Multiple QEs on one physical server
  - To utilize multiple CPU cores
- Gigabit ethernet
- Firewall

QD / QE roles

- Utility mode
  - `PGOPTIONS='-c gp_session_role=utility' psql postgres`
- Catalogs are the same in all nodes
  - OIDs of objects must match
  - `gpcheckcat` to verify

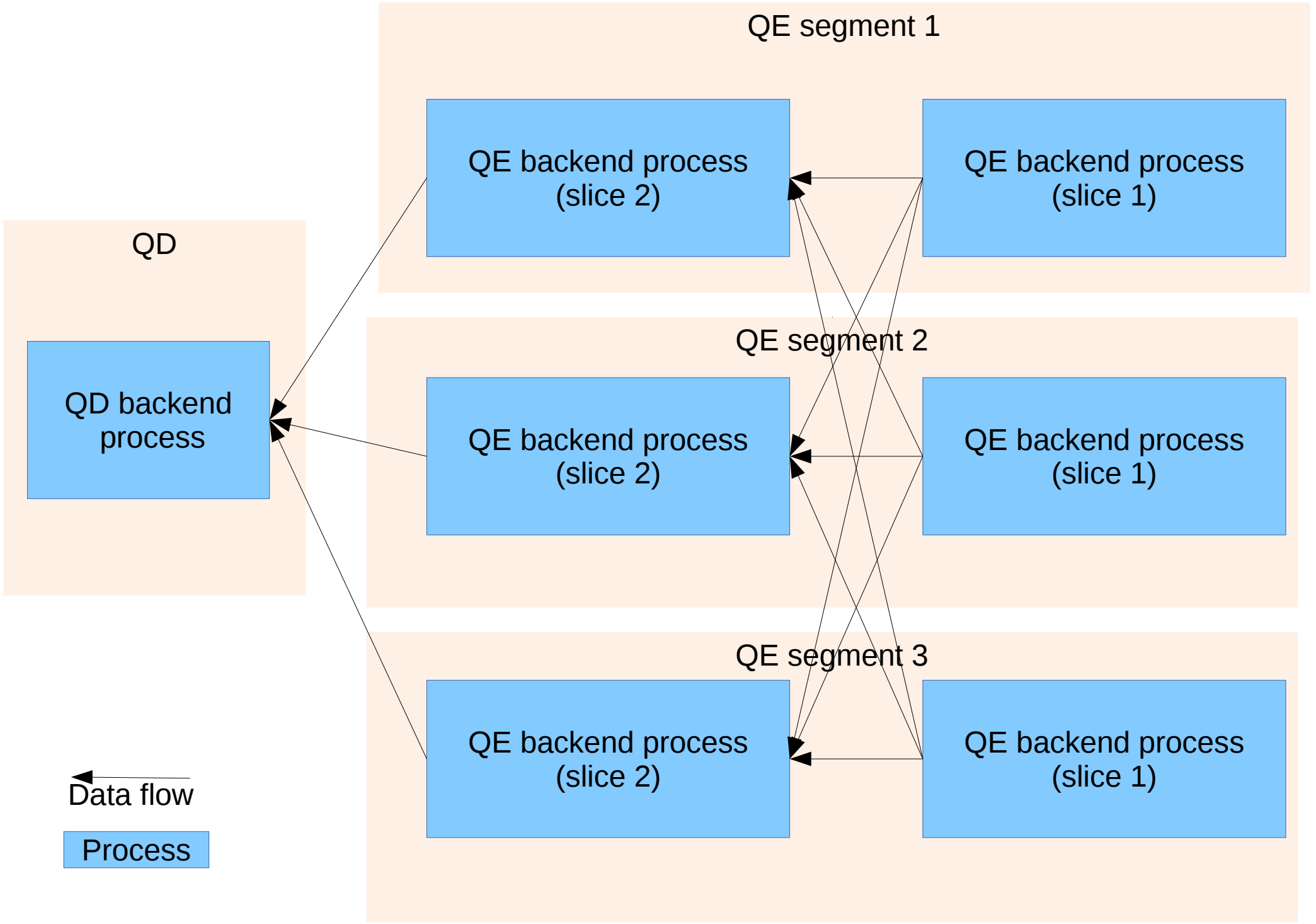
Distributed plans

```
postgres=# explain select * from foo;
                        QUERY PLAN
-------------------------------------------------
 Gather Motion 3:1  (slice1; segments: 3)
   ->  Seq Scan on foo
```

- Query plan is dispatched from QD to QE nodes
- Query results are returned from QEs via a Motion node

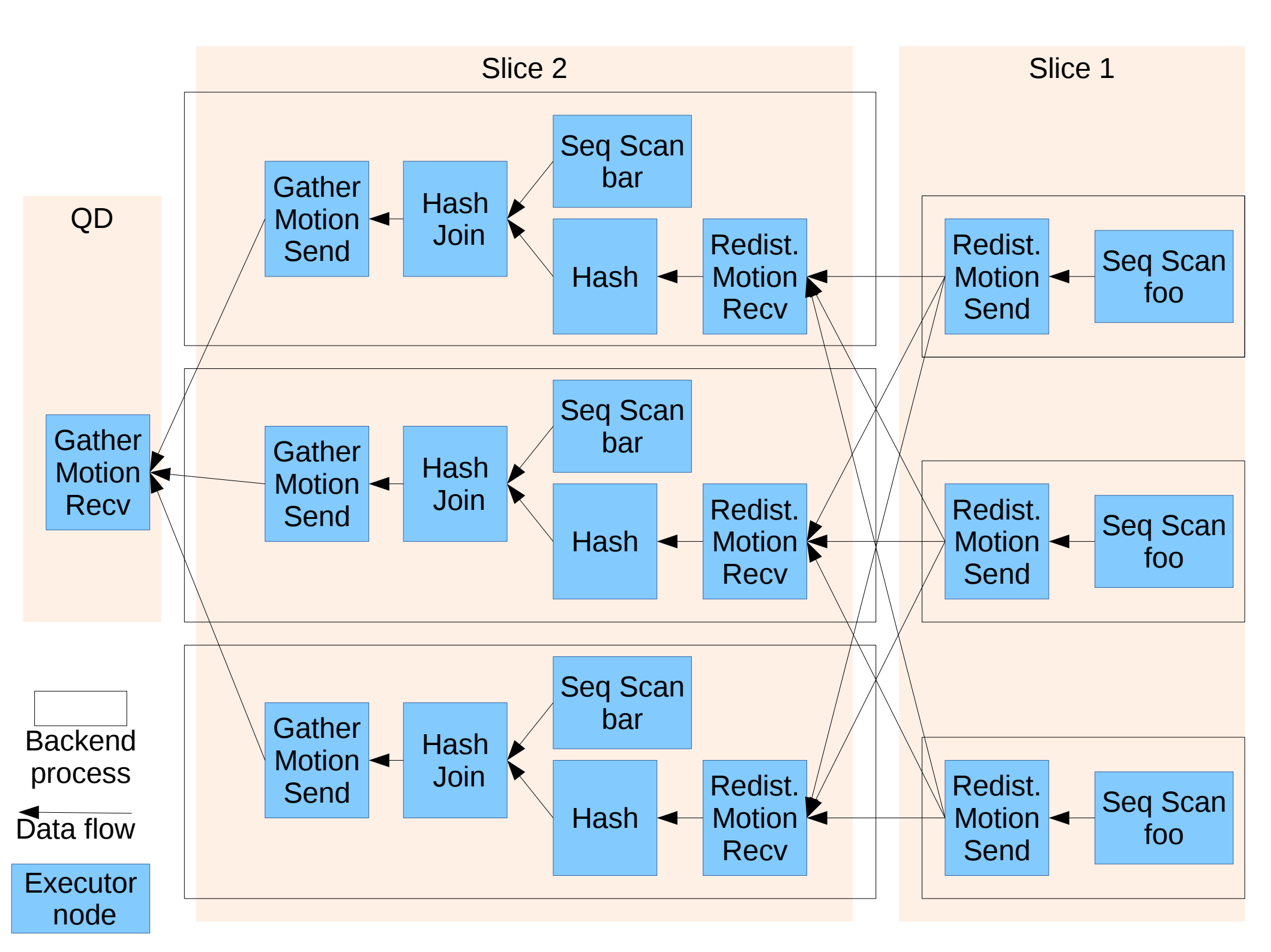
More complicated query

```
postgres=# explain select * from foo, bar where bar.c = foo.a;
                              QUERY PLAN
-------------------------------------------------------------------
 Gather Motion 3:1  (slice2; segments: 3)
   ->  Hash Join  (cost=2.44..5.65 rows=2 width=12)
         Hash Cond: bar.c = foo.a
         ->  Seq Scan on bar
         ->  Hash
               ->  Broadcast Motion 3:3  (slice1; segments: 3)
                     ->  Seq Scan on foo
```


QE segment 1
QE backend process (slice 2)
QE backend process (slice 1)
QD
QD backend process
QE segment 2
QE backend process (slice 2)
QE backend process (slice 1)
QE segment 3
QE backend process (slice 2)
QE backend process (slice 1)
Data flow
Process

Interconnect

- Data flows between Motion Sender and Receiver via the *Interconnect*
- UDP based protocol


Slice 2
Slice 1
QD
Gather Motion Recv
Gather Motion Send
Hash Join
Seq Scan bar
Hash
Redist. Motion Recv
Redist. Motion Send
Seq Scan foo
Backend process
Data flow
Executor node

Distributed planning

- Prepare a non-distributed plan
- Keep track of distribution keys at each step
- Add Motion nodes to make it distributed
- Different strategies to distribute aggregates

Data flows in one direction

- Nested loops with executor parameters cannot be dispatched across query slices
  - Need to be materialized
  - The planner heavily discourages Nested loops
  - Hash joins are preferred
- Subplans with parameters need to use seqscans and must be fully materialized
- “Skip-level” subplans not supported

Distributed transactions

- Every transaction uses two-phase commit
- QD acts as the transaction manager
- Distributed snapshots to ensure consistency across nodes

Mirroring

If one QE node goes down, no queries can be run:

```
postgres=# select * from foo;

ERROR:  failed to acquire resources on one or more
segments
DETAIL:  could not connect to server: Connection
refused
Is the server running on host "127.0.0.1" and
accepting
TCP/IP connections on port 40002?
 (seg2 127.0.0.1:40002)
```

Mirroring

- Each QE node has a mirror QE node as backup
- GPDB-specific “file replication” between the master and mirror QE node

- PostgreSQL WAL replication between master and mirror QD

Other non-MPP features

ORCA

- Alternative optimizer
- Written in C++
- Shared with Apache HAWQ
- https://github.com/greenplum-db/gporca
- Raw query is converted to XML, sent to ORCA, and converted back
- Slow startup time, better plans for some queries

Partitioning

```
CREATE TABLE partition_test
(
  col1 int,
  col2 decimal,
  col3 text
)
distributed by (col1)
partition by list(col2)
(
  partition part1 VALUES(1,2,3,4,5,6,7,8,9,10),
  partition part2 VALUES(11,12,13,14,15,16,17,18,19,20),
  partition part3 VALUES(21,22,23,24,25,26,27,28,29,30),
  partition part4 VALUES(31,32,33,34,35,36,37,38,39,40),
  default partition def
);
```

Append-optimized Tables

- Alternative to normal heap tables
  - CREATE TABLE … WITH (**appendonly=true**)
- Optimized for appending in large batches and sequential scans
- Compression
- Column- or row-oriented

Bitmap indexes

- More dense than B-tree for duplicates
- Good for columns with few distinct values

Show Me the Code!

$ ls -l

```
drwxr-xr-x  5 heikki heikki   4096 Jun 30 11:37 concourse
drwxr-xr-x  2 heikki heikki   4096 Jun 30 11:37 config
-rwxr-xr-x  1 heikki heikki 528544 Jun 30 11:37 configure
-rw-r--r--  1 heikki heikki  78636 Jun 30 11:37 configure.in
drwxr-xr-x 54 heikki heikki   4096 Jun 30 11:37 contrib
-rw-r--r--  1 heikki heikki   1547 Jun 30 11:37 COPYRIGHT
drwxr-xr-x  3 heikki heikki   4096 Jun 30 11:37 doc
-rw-r--r--  1 heikki heikki   8253 Jun 30 11:37 GNUmakefile.in
drwxr-xr-x  8 heikki heikki   4096 Jun 30 11:37 gpAux
drwxr-xr-x  4 heikki heikki   4096 Jun 30 11:37 gpdb-doc
drwxr-xr-x  7 heikki heikki   4096 Jun 30 11:37 gpMgmt
-rw-r--r--  1 heikki heikki  11358 Jun 30 11:37 LICENSE
-rw-r--r--  1 heikki heikki   1556 Jun 30 11:37 Makefile
-rw-r--r--  1 heikki heikki 113093 Jun 30 11:37 NOTICE
-rw-r--r--  1 heikki heikki   2051 Jun 30 11:37 README.amazon_linux
-rw-r--r--  1 heikki heikki   1514 Jun 30 11:37 README.debian
-rw-r--r--  1 heikki heikki  19812 Jun 30 11:37 README.md
-rw-r--r--  1 heikki heikki   1284 Jun 30 11:37 README.PostgreSQL
drwxr-xr-x 14 heikki heikki   4096 Jun 30 11:37 src
```

The Code

- On Github:

  https://github.com/greenplum-db/gpdb/

Code layout

Same as PostgreSQL:

- src/, contrib/, configure

Greenplum-specific:

- gpMgmt/
- gpAux/
- concourse/

Catalogs

- Extra catalog tables for GPDB functionality
  - pg_partition, pg_partitition_rule,
  - gp_distribution_policy
  - ...
- Extra columns in pg_proc, pg_class
  - a Perl script provides defaults for them at build time

pg_proc.h

```
DATA(insert OID = 2557 ( bool                                   PGNSP PGUID 12 1 0 f f t f i 1  16 "23"
_null_ _null_ _null_ int4_bool - _null_ _null_ ));
DESCR("convert int4 to boolean");
DATA(insert OID = 2558 ( int4                                   PGNSP PGUID 12 1 0 f f t f i 1  23 "16"
_null_ _null_ _null_ bool_int4 - _null_ _null_ ));
DESCR("convert boolean to int4");
DATA(insert OID = 2559 ( lastval                                PGNSP PGUID 12 1 0 f f t f v 0 20 "" _null_
_null_ _null_    lastval - _null_ _null_ ));
DESCR("current value from last used sequence");

/* start time function */
DATA(insert OID = 2560 (  pg_postmaster_start_time PGNSP PGUID 12 1 0 f f t f s 0 1184 "" _null_
_null_ _null_ pgsql_postmaster_start_time - _null_ _null_ ));
DESCR("postmaster start time");

/* new functions for Y-direction rtree opclasses */
DATA(insert OID = 2562 (  box_below                PGNSP PGUID 12 1 0 f f t f i 2 16 "603 603" _null_
_null_ _null_ box_below - _null_ _null_ ));
DESCR("is below");
DATA(insert OID = 2563 (  box_overbelow    PGNSP PGUID 12 1 0 f f t f i 2 16 "603 603" _null_ _null_
_null_ box_overbelow - _null_ _null_ ));
DESCR("overlaps or is below");
DATA(insert OID = 2564 (  box_overabove    PGNSP PGUID 12 1 0 f f t f i 2 16 "603 603" _null_ _null_
_null_ box_overabove - _null_ _null_ ));
DESCR("overlaps or is above");
```

pg_proc_gp.sql

```
-- All GPDB-added functions are here, instead of pg_proc.h. pg_proc.h should
-- kept as close as possible to the upstream version, to make merging easier.
--
-- This file is translated into DATA rows by catullus.pl. See
-- README.add_catalog_function for instructions on how to run it.

 CREATE FUNCTION float4_decum(_float8, float4) RETURNS _float8 LANGUAGE internal IMMUTABLE
STRICT AS 'float4_decum' WITH (OID=6024, DESCRIPTION="aggregate inverse transition
function");
 CREATE FUNCTION float4_avg_accum(bytea, float4) RETURNS bytea LANGUAGE internal IMMUTABLE
STRICT AS 'float4_avg_accum' WITH (OID=3106, DESCRIPTION="aggregate transition function");
 CREATE FUNCTION float4_avg_decum(bytea, float4) RETURNS bytea LANGUAGE internal IMMUTABLE
STRICT AS 'float4_avg_decum' WITH (OID=3107, DESCRIPTION="aggregate inverse transition
function");
 CREATE FUNCTION float8_decum(_float8, float8) RETURNS _float8 LANGUAGE internal IMMUTABLE
STRICT AS 'float8_decum' WITH (OID=6025, DESCRIPTION="aggregate inverse transition
function");
 CREATE FUNCTION float8_avg_accum(bytea, float8) RETURNS bytea LANGUAGE internal IMMUTABLE
STRICT AS 'float8_avg_accum' WITH (OID=3108, DESCRIPTION="aggregate transition function");
 CREATE FUNCTION float8_avg_decum(bytea, float8) RETURNS bytea LANGUAGE internal IMMUTABLE
STRICT AS 'float8_avg_decum' WITH (OID=3109, DESCRIPTION="aggregate inverse transition
function");
 CREATE FUNCTION btgpxlogloccmp(gpxlogloc, gpxlogloc) RETURNS int4 LANGUAGE internal
IMMUTABLE STRICT AS 'btgpxlogloccmp' WITH (OID=7081, DESCRIPTION="btree less-equal-greater");

-- MPP -- array_add
 CREATE FUNCTION array_add(_int4, _int4) RETURNS _int4 LANGUAGE internal IMMUTABLE STRICT AS
'array_int4_add' WITH (OID=6012, DESCRIPTION="itemwise add two integer arrays");
```

Documentation

- `doc/` contains the user manual in PostgreSQL
  - All but reference pages removed in Greenplum
- `gpdb-doc/` contains the Greenplum manuals

Compile!

```
./configure
make install
```

- Some features that are optional in PostgreSQL are mandatory in Greenplum to run the regression tests.
- Read the README.md for which flags to use

```
~/gpdb.master$ bin/initdb -D data
The files belonging to this database system will be owned by user "heikki".
This user must also own the server process.


The database cluster will be initialized with locale en_US.utf8.
The default database encoding has accordingly been set to UTF8.
The default text search configuration will be set to "english".

creating directory data ... ok

…
Success. You can now start the database server using:

    bin/postgres -D data
or
    bin/pg_ctl -D data -l logfile start


~/gpdb.master$ bin/postgres -D data
2017-06-30 08:50:05.440267 GMT,,,p16890,th-751686272,,,,0,,,seg-
1,,,,,"FATAL","22023","dbid (from -b option) is not specified or is invalid.
 This value must be >= 0, or >= -1 in utility mode.  The dbid value to pass
can be determined from this server's entry in the segment configuration; it
may be -1 if running in utility
mode.",,,,,,,,"PostmasterMain","postmaster.c",1174,
```

Creating a cluster

- `gpinitsystem` – runs initdb on each node
- `gpstart` / `gpstop` – runs `pg_ctl start/stop` on each node

- For quick testing and hacking on your laptop:

```
make cluster
```

Regression tests

- Regression test suite in src/test/regress
  - Core is the same as in PostgreSQL
  - Greatly extended to cover Greenplum-specific features
  - Needs a running cluster

```
make -C src/test/regress installcheck
```

gpdiff.pl

- In GPDB, rows can arrive from QE nodes in any order
- Post-processing step in installcheck masks out the row-order differences

gpdiff.pl directives

```
--start_ignore
DROP TABLE IF EXISTS T_a1 CASCADE;
DROP TABLE IF EXISTS T_b2 CASCADE;
DROP TABLE IF EXISTS T_random
CASCADE;
--end_ignore
```

More tests

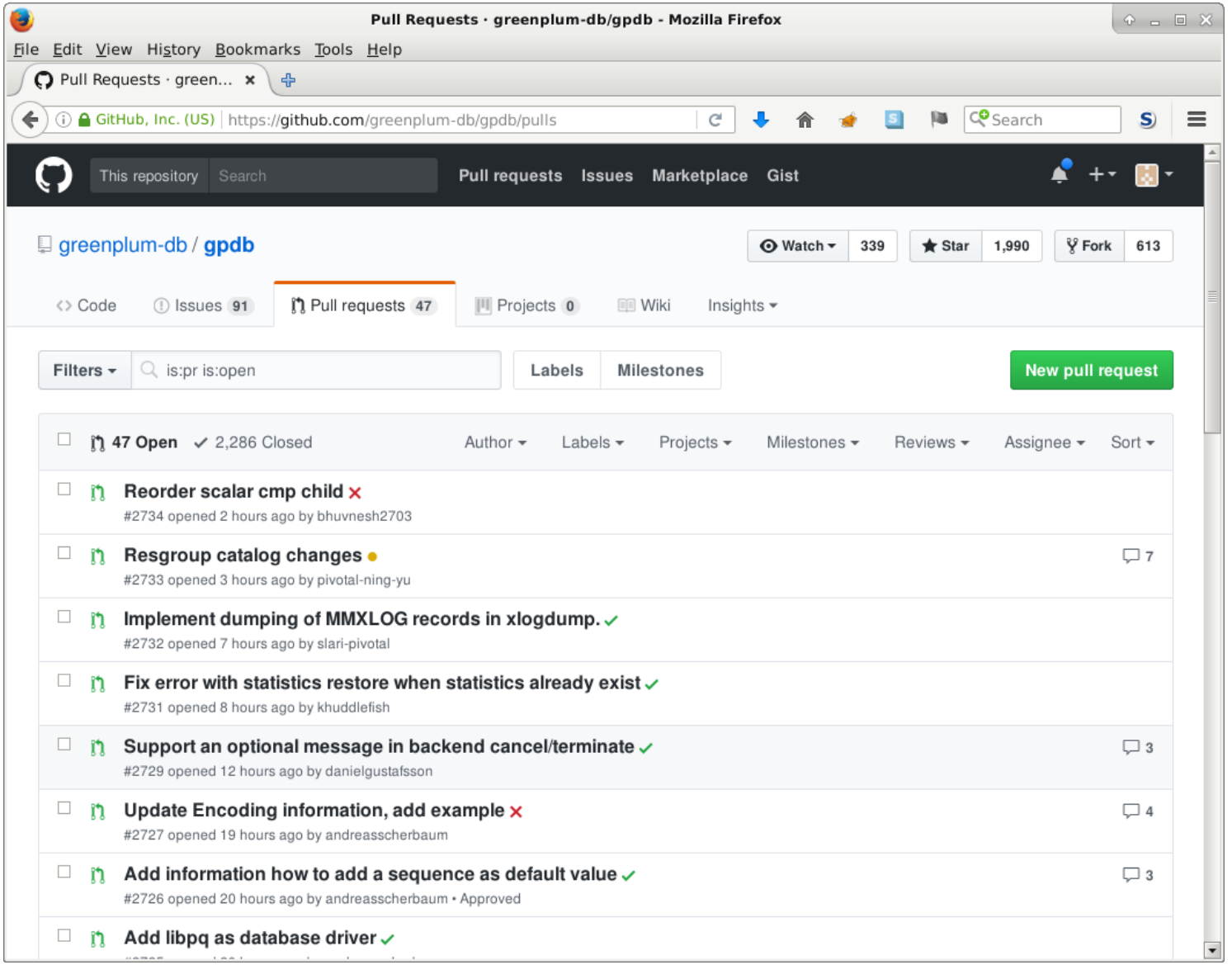
- src/test/
  - isolation/
  - isolation2/
  - tinc/
- Use “make installcheck-world” to run most of these

Yet more tests

- Pivotal runs a Concourse CI instance to run “make installcheck-world”
- And some additional tests that need special setup
- Scripts in concourse/ directory in the repository

Greenplum development on github

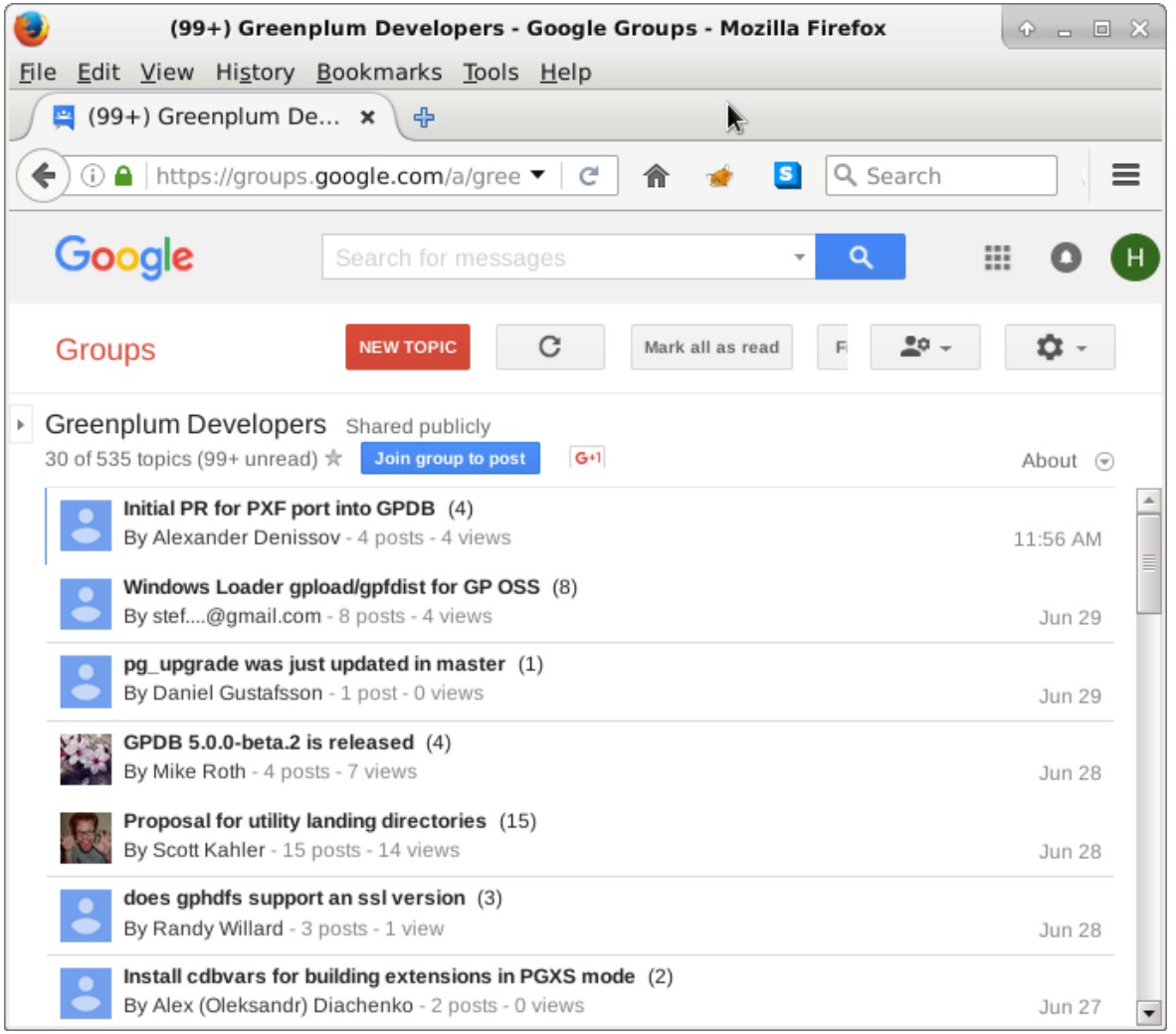
- PRs
- Issues



Greenplum mailing lists

Public mailing lists on Google Groups:

- gpdb-dev
- gpdb-users



greenplum.org

- News, events
- Links to the github project, mailing list, Concourse instance, and more

That’s all, folks!