Parallel Execution Plans

Jonathan Lewis

jonathanlewis.wordpress.com

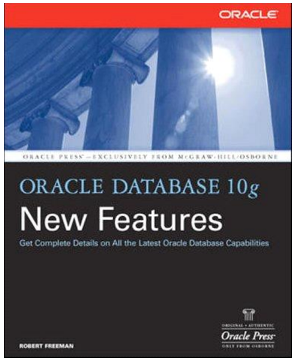
www.jlcomp.demon.co.uk

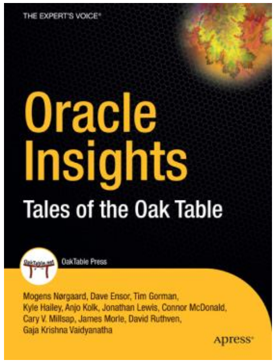
My History

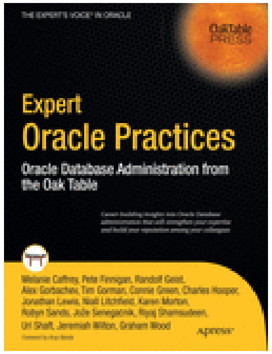
Independent Consultant

33+ years in IT
28+ using Oracle (5.1a on MSDOS 3.3

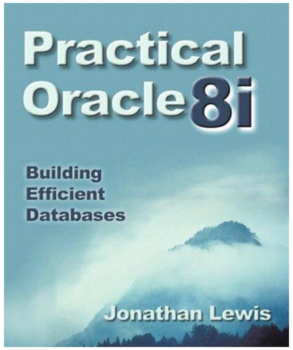
Strategy, Design, Review,
Briefings, Educational,
Trouble-shooting

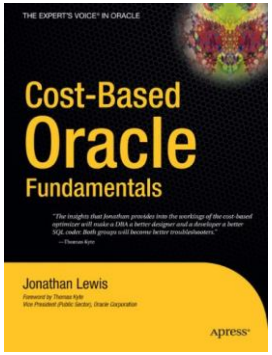

ORACLE
ORACLE DATABASE 10g
New Features
Get Complete Details on All the Latest Oracle Database Capabilities
Oracle Press
ROBERT FREEMAN

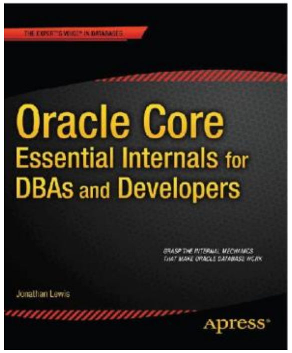




Oracle author of the year 2006
Select Editor's choice 2007
UKOUG Inspiring Presenter 2011
ODTUG 2012 Best Presenter (d/b)
UKOUG Inspiring Presenter 2012
UKOUG Lifetime Award (IPA) 2013
Member of the Oak Table Network
Oracle *ACE Director*
O1 visa for USA







Many slides have a foot-note. This is just two lines summarizing the highlights of the slide so that you have a reference when reading the handouts at a later date.

Topics

- What are we looking for in a plan
  - Order of operation (row source generation)
  - Resource usage
  - Early elimination of data
- What tools can we use
  - dbms_xplan
  - v$pq_tqstat
  - Extended sql_trace, or equivalent
  - v$sql_monitor - if licensed (diagnostic + performance)

Terminology

QC: "Query coordinator" - the process controlling the query (and passing data to the front end)

PX Server: single process used in parallel query

a.k.a Parallel server, Parallel Query Slave, PQ slave, PX slave

Slave Set: A set of PX Servers performing one operation of an execution plan - commonly a single query will use two sets of PX servers

DOP: "degree of parallelism" - number of slaves in ***each*** slave set involved in a full parallel execution plan

Table Queue: Logical communication channel between two sets of slaves, or from a slave set to the QC

a.k.a Virtual table

DFO: "data flow operation" - the set of actions that moves data through a single table queue

DFO tree: Set of DFOs moving data from its source to the QC

Big Problem

SMALL PRINT

Sample Data (a)

```
create table t1 as
select
        rownum                id,
        to_char(rownum)       small_vc,
        rpad('x',100)         padding
from    all_objects
where   rownum <= 70;

alter table t1 add constraint t1_pk primary key(id);

begin
   dbms_stats.gather_table_stats(
        user,
        't1',
        method_opt => 'for all columns size 1'
   );
end;
```

Repeat for matching t2 and t3

Sample Data (b)

```
create table t4 as
select
        t1.id                  id1,
        t2.id                  id2,
        t3.id                  id3,
        rpad(rownum,10)        small_vc,
        rpad('x',100)          padding
from
        t1, t2, t3             -- 343,000 rows
;

begin
   dbms_stats.gather_table_stats(
        user,
        't4',
        method_opt => 'for all columns size 1'
   );
end;
```

Sample Query (serial)

```
select
        /*+ gather_plan_statistics */
        count(t1.small_vc),    count(t2.small_vc),
        count(t3.small_vc),    count(t4.small_vc)
from
        t4,
        t1,
        t2,
        t3
where
        t1.id = t4.id1
and     t2.id = t4.id2
and     t3.id = t4.id3
and     t1.small_vc in (1,2,3)                    -- type mismatch !!!
and     t2.small_vc in (1,2,3,4)
and     t3.small_vc in (1,2,3,4,5)
;
```

Sample Query (serial plan)

select * from table(dbms_xplan.display_cursor(null,null,***'allstats last'***));

```
| Id  | Operation                   | Name | Starts | E-Rows | A-Rows |
--------------------------------------------------------------------
|   0 | SELECT STATEMENT            |      |      1 |        |      1 |
|   1 |  SORT AGGREGATE             |      |      1 |      1 |      1 |
|*  2 |   HASH JOIN                 |      |      1 |     56 |     60 |
|*  3 |    TABLE ACCESS FULL        | T3   |      1 |      5 |      5 |
|*  4 |    HASH JOIN                |      |      1 |    810 |    840 |
|*  5 |     TABLE ACCESS FULL       | T2   |      1 |      4 |      4 |
|*  6 |     HASH JOIN               |      |      1 |  14491 |  14700 |
|*  7 |      TABLE ACCESS FULL| T1   |      1 |      3 |      3 |
|   8 |      TABLE ACCESS FULL| T4   |      1 |    343K|    343K|
--------------------------------------------------------------------
```

```
 leading(t4 t1 t2 t3)
 use_hash(t1) swap_join_inputs(t1)
 use_hash(t2) swap_join_inputs(t2)
 use_hash(t3) swap_join_inputs(t3)
```

We can read the plan by "first child - recursive descent". The order of action is: scan and hash t3, scan and hash t2, scan and hash t1, scan t4 and probe x3

Sample Query (serial trace)

```
alter system flush buffer_cache;
alter session set events '10046 trace name context forever, level 8';
```

Tablescan table t3

```
WAIT #: nam='db file sequential read' ela= 2207 f#=7 b#=640 bs=1 obj#=235626
WAIT #: nam='db file scattered read'  ela=  570 f#=7 b#=641 bs=2 obj#=235626
```

Tablescan table t2

```
WAIT #: nam='db file sequential read' ela=  458 f#=7 b#=384 bs=1 obj#=235624
WAIT #: nam='db file scattered read'  ela=  387 f#=7 b#=385 bs=2 obj#=235624
```

Tablescan table t1

```
WAIT #: nam='db file sequential read' ela=  524 f#=7 b#=128 bs=1 obj#=235622
WAIT #: nam='db file scattered read'  ela=  477 f#=7 b#=129 bs=2 obj#=235622
```

Tablescan table t4 (direct)

```
WAIT #: nam='db file sequential read' ela=  502 f#=7 b#=896 bs=  1 obj#=235628
WAIT #: nam='direct path read'        ela= 1658 f= 7 fd=897 bc=127 obj#=235628

WAIT #140096457765816: nam='direct path read'
```

As a little check for order of operation, the 10046 trace file (flushing the buffer cache before doing the test) can show us the physical read waits.

Going Parallel *(hash/hash)*

```
select
        /*+
                gather_plan_statistics
                leading(t4 t1 t2 t3)
                parallel(t4,2) full(t4) parallel(t1,2) full(t1)
                parallel(t2,2) full(t2) parallel(t3,2) full(t3)
--
                use_hash(t1) swap_join_inputs(t1)
                pq_distribute(t1 hash hash)
--
                use_hash(t2) swap_join_inputs(t2)
                pq_distribute(t2 hash hash)
--
                use_hash(t3) swap_join_inputs(t3)
                pq_distribute(t3 hash hash)
        */
        count(t1.small_vc),     count(t2.small_vc),
        count(t3.small_vc),     count(t4.small_vc)
from    ...
```

parallel/full - force a parallel full tablescan. use_hash/swap_join_inputs - do a hash join with swap; when ***tN*** is the "next" table in the join ***hash distribute*** both inputs

Going Parallel *(broadcast)*

```
select
        /*+
               gather_plan_statistics
               leading(t4 t1 t2 t3)
               parallel(t4,2) full(t4) parallel(t1,2) full(t1)
               parallel(t2,2) full(t2) parallel(t3,2) full(t3)
--
               use_hash(t1) swap_join_inputs(t1)
               pq_distribute(t1 none broadcast)
--
               use_hash(t2) swap_join_inputs(t2)
               pq_distribute(t2 none broadcast)
--
               use_hash(t3) swap_join_inputs(t3)
               pq_distribute(t3 none broadcast)
        */
        count(t1.small_vc),   count(t2.small_vc),
        count(t3.small_vc),   count(t4.small_vc)
from    ...
```

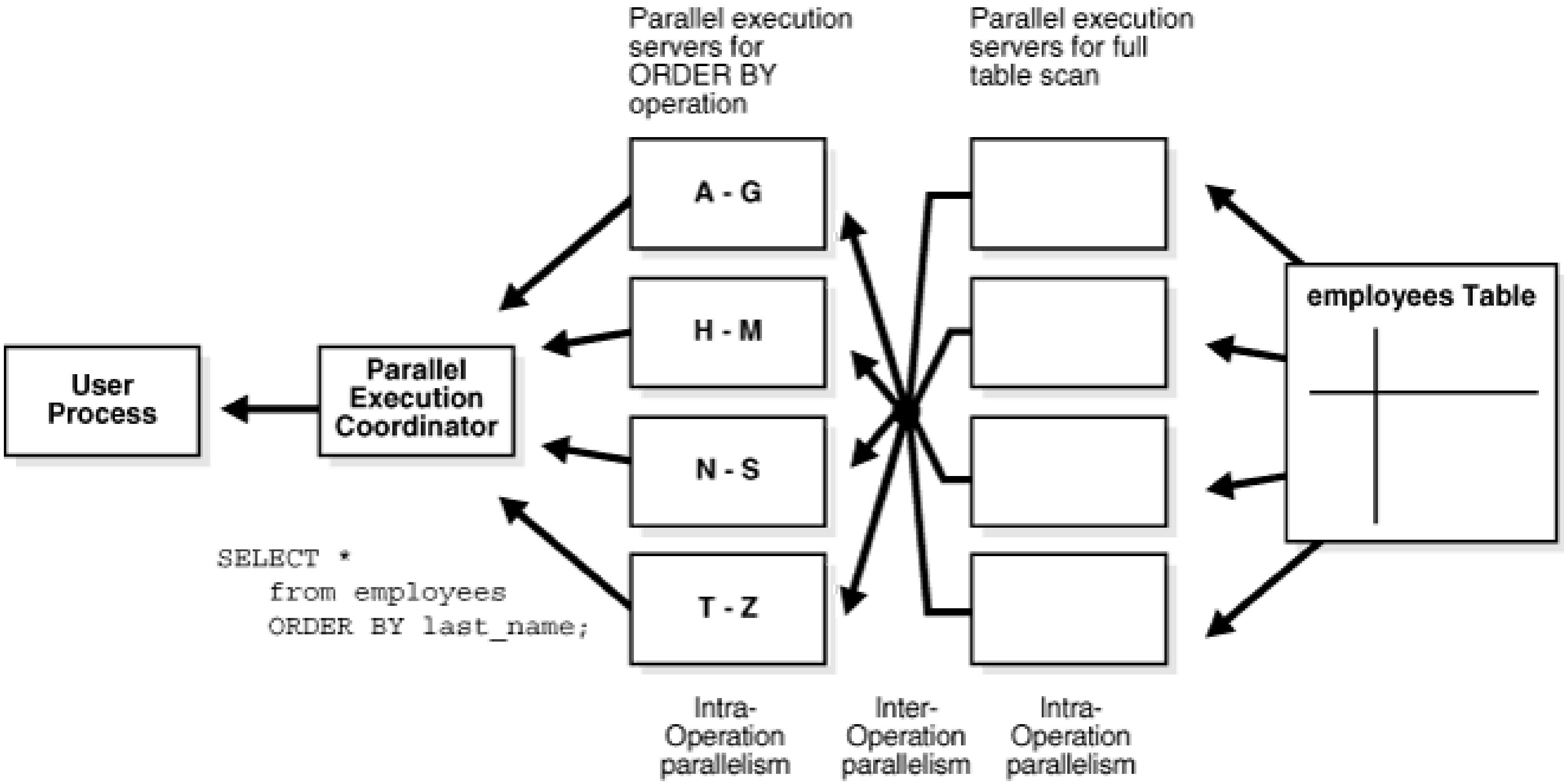
parallel/full - force a parallel full tablescan. use_hash/swap_join_inputs - do a hash join with swap; when ***tN*** is the next table in the join ***broadcast*** it

Execution plan (broadcast)

```
| Id | Operation                    | Name     | Rows  |   TQ   |IN-OUT| PQ Distrib |
| 0  | SELECT STATEMENT             |          |       |        |      |            |
| 1  |  SORT AGGREGATE              |          |     1 |        |      |            |
| 2  |   PX COORDINATOR             |          |       |        |      |            |
| 3  |    PX SEND QC (RANDOM)       | :TQ10003 |     1 |  Q1,03 | P->S | QC (RAND)  |
| 4  |     SORT AGGREGATE           |          |     1 |  Q1,03 | PCWP |            |
| 5  |      HASH JOIN               |          |    56 |  Q1,03 | PCWP |            |
| 6  |       PX RECEIVE             |          |     5 |  Q1,03 | PCWP |            |
| 7  |        PX SEND BROADCAST     | :TQ10000 |     5 |  Q1,00 | P->P | BROADCAST  |
| 8  |         PX BLOCK ITERATOR    |          |     5 |  Q1,00 | PCWC |            |
| 9  |          TABLE ACCESS FULL   | T3       |     5 |  Q1,00 | PCWP |            |
| 10 |       HASH JOIN              |          |   810 |  Q1,03 | PCWP |            |
| 11 |        PX RECEIVE            |          |     4 |  Q1,03 | PCWP |            |
| 12 |         PX SEND BROADCAST    | :TQ10001 |     4 |  Q1,01 | P->P | BROADCAST  |
| 13 |          PX BLOCK ITERATOR   |          |     4 |  Q1,01 | PCWC |            |
| 14 |           TABLE ACCESS FULL  | T2       |     4 |  Q1,01 | PCWP |            |
| 15 |        HASH JOIN             |          | 14491 |  Q1,03 | PCWP |            |
| 16 |         PX RECEIVE           |          |     3 |  Q1,03 | PCWP |            |
| 17 |          PX SEND BROADCAST   | :TQ10002 |     3 |  Q1,02 | P->P | BROADCAST  |
| 18 |           PX BLOCK ITERATOR  |          |     3 |  Q1,02 | PCWC |            |
| 19 |            TABLE ACCESS FULL| T1       |     3 |  Q1,02 | PCWP |            |
| 20 |         PX BLOCK ITERATOR    |          |  343K|  Q1,03 | PCWC |            |
| 21 |          TABLE ACCESS FULL   | T4       |  343K|  Q1,03 | PCWP |            |
```

We now have 22 lines instead of 9 but, between all the send/receive operations we can still see the shape of the four table hash join with the original join order.

Parallel Images

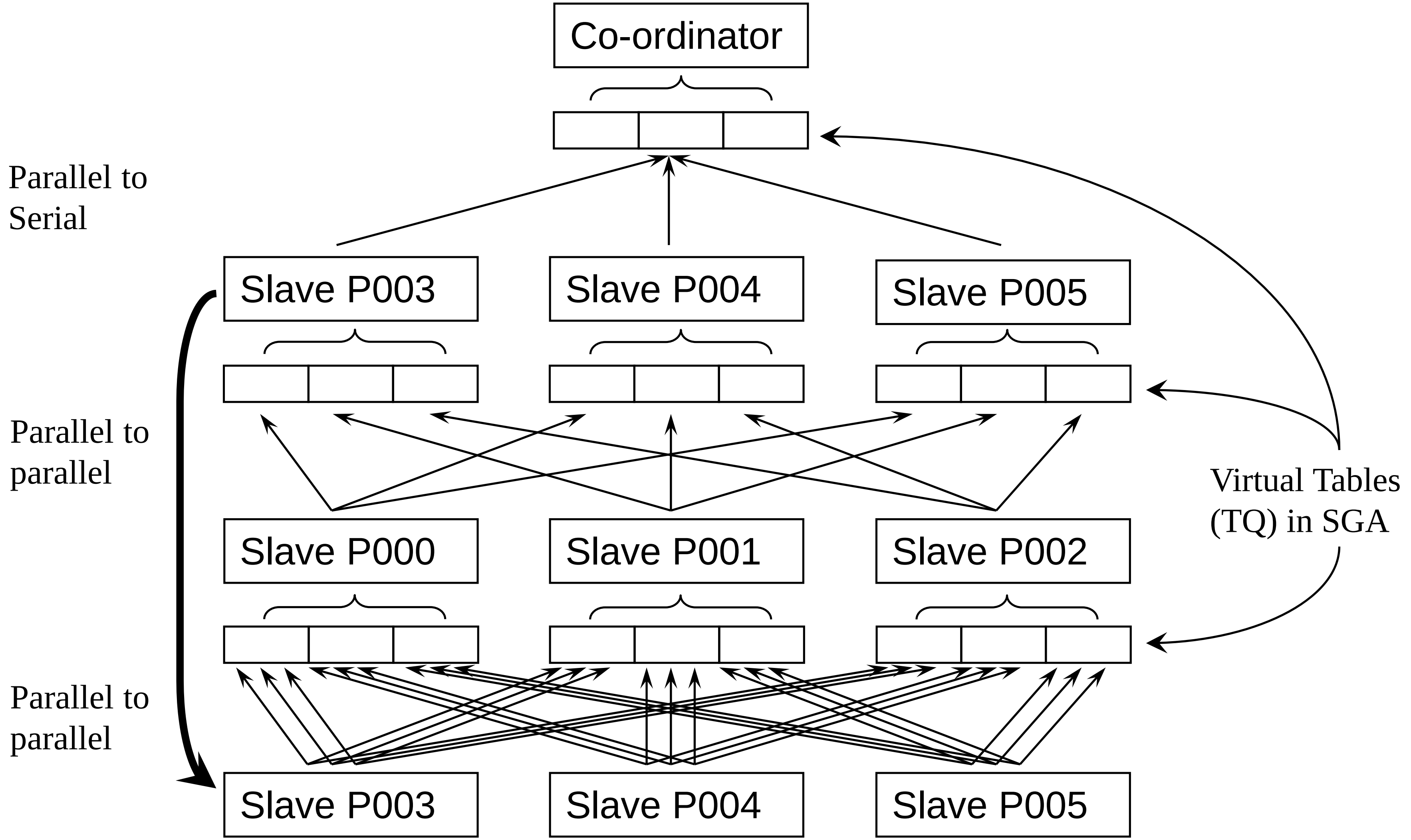

Parallel execution servers for ORDER BY operation
Parallel execution servers for full table scan
A - G
H - M
N - S
T - Z
employees Table
User Process
Parallel Execution Coordinator
Intra-Operation parallelism
Inter-Operation parallelism
Intra-Operation parallelism

```
SELECT *
    from employees
    ORDER BY last_name;
```

Oracle® Database VLDB and Partitioning Guide Ch. 8

Conveniently this simple example shows just two send/receive pairs: from slave set 1 to slave set 2, then from slave set to the query coordinator. Real-life is more complex

Parallel Execution - visual



Complex queries may need many layers of parallel execution - but Oracle limits a query to two sets of parallel execution slaves, and this has interesting consequences

Execution plan (broadcast)

```
| Id  | Operation                   | Name     | Rows  |    TQ  |IN-OUT| PQ Distrib |
|   0 | SELECT STATEMENT            |          |       |        |      |            |
|   1 |  SORT AGGREGATE             |          |     1 |        |      |            |
|   2 |   PX COORDINATOR            |          |       |        |      |            |
|   3 |    PX SEND QC (RANDOM)      | :TQ10003 |     1 |  Q1,03 | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE          |          |     1 |  Q1,03 | PCWP |            |
|   5 |      HASH JOIN              |          |    56 |  Q1,03 | PCWP |            |
|   6 |       PX RECEIVE            |          |     5 |  Q1,03 | PCWP |            |
|   7 |        PX SEND BROADCAST    | :TQ10000 |     5 |  Q1,00 | P->P | BROADCAST  |
|   8 |         PX BLOCK ITERATOR   |          |     5 |  Q1,00 | PCWC |            |
|   9 |          TABLE ACCESS FULL  | T3       |     5 |  Q1,00 | PCWP |            |
|  10 |       HASH JOIN             |          |   810 |  Q1,03 | PCWP |            |
|  11 |        PX RECEIVE           |          |     4 |  Q1,03 | PCWP |            |
|  12 |         PX SEND BROADCAST   | :TQ10001 |     4 |  Q1,01 | P->P | BROADCAST  |
|  13 |          PX BLOCK ITERATOR  |          |     4 |  Q1,01 | PCWC |            |
|  14 |           TABLE ACCESS FULL | T2       |     4 |  Q1,01 | PCWP |            |
|  15 |        HASH JOIN            |          | 14491 |  Q1,03 | PCWP |            |
|  16 |         PX RECEIVE          |          |     3 |  Q1,03 | PCWP |            |
|  17 |          PX SEND BROADCAST  | :TQ10002 |     3 |  Q1,02 | P->P | BROADCAST  |
|  18 |           PX BLOCK ITERATOR |          |     3 |  Q1,02 | PCWC |            |
|  19 |            TABLE ACCESS FULL| T1       |     3 |  Q1,02 | PCWP |            |
|  20 |         PX BLOCK ITERATOR   |          |   343K|  Q1,03 | PCWC |            |
|  21 |          TABLE ACCESS FULL  | T4       |   343K|  Q1,03 | PCWP |            |
```

For parallel queries we have to follow the "virtual tables", known as ***"table queues"***.
The order of operation follows the sequence of generating TQs (Name column.)

Execution plan (broadcast)

```
| Id | Operation                   | Name     | Rows   |   TQ  |IN-OUT| PQ Distrib |

|  6 |       PX RECEIVE            |          |      5 | Q1,03 | PCWP |            |
|  7 |        PX SEND BROADCAST    | :TQ10000 |      5 | Q1,00 | P->P | BROADCAST  |
|  8 |         PX BLOCK ITERATOR   |          |      5 | Q1,00 | PCWC |            |
|  9 |          TABLE ACCESS FULL  | T3       |      5 | Q1,00 | PCWP |            |

| 11 |        PX RECEIVE           |          |      4 | Q1,03 | PCWP |            |
| 12 |         PX SEND BROADCAST   | :TQ10001 |      4 | Q1,01 | P->P | BROADCAST  |
| 13 |          PX BLOCK ITERATOR  |          |      4 | Q1,01 | PCWC |            |
| 14 |           TABLE ACCESS FULL | T2       |      4 | Q1,01 | PCWP |            |

| 16 |         PX RECEIVE          |          |      3 | Q1,03 | PCWP |            |
| 17 |          PX SEND BROADCAST  | :TQ10002 |      3 | Q1,02 | P->P | BROADCAST  |
| 18 |           PX BLOCK ITERATOR |          |      3 | Q1,02 | PCWC |            |
| 19 |            TABLE ACCESS FULL| T1       |      3 | Q1,02 | PCWP |            |
```

In this case the order of operation matches the serial plan. All three tables are inputs to the same (TQ10003) virtual table - so a single slave set must be receiving them.

Execution plan (broadcast)

```
| Id | Operation                    | Name     | Rows  |    TQ  |IN-OUT| PQ Distrib |
|  0 | SELECT STATEMENT             |          |       |        |      |            |
|  1 |  SORT AGGREGATE              |          |     1 |        |      |            |
|  2 |   PX COORDINATOR             |          |       |        |      |            |
|  3 |    PX SEND QC (RANDOM)       | :TQ10003 |     1 |  Q1,03 | P->S | QC (RAND)  |
|  4 |     SORT AGGREGATE           |          |     1 |  Q1,03 | PCWP |            |
|  5 |      HASH JOIN               |          |    56 |  Q1,03 | PCWP |            |
|  6 |       PX RECEIVE             |          |     5 |  Q1,03 | PCWP |            |




| 10 |       HASH JOIN              |          |   810 |  Q1,03 | PCWP |            |
| 11 |        PX RECEIVE            |          |     4 |  Q1,03 | PCWP |            |




| 15 |        HASH JOIN             |          | 14491 |  Q1,03 | PCWP |            |
| 16 |         PX RECEIVE           |          |     3 |  Q1,03 | PCWP |            |




| 20 |         PX BLOCK ITERATOR    |          |   343K|  Q1,03 | PCWC |            |
| 21 |          TABLE ACCESS FULL   | T4       |   343K|  Q1,03 | PCWP |            |
```

Think of the ***name*** column as the virtual table being used as a pipeline, then the ***TQ*** column tells you how a set of slaves finds data for that virtual table.

PQ TQ stats (v$pq_tqstat)

```
select
        dfo_number,
        tq_id,
        server_type,   -- producer/consumer/ranger
        instance,      -- for RAC
        process,       -- pNNN
        num_rows
from
        v$pq_tqstat
order by
        dfo_number,
        tq_id,
        server_type desc,
        instance,
        process
;
```

TQ10003 in the plan can be aligned with dfo_number = 1, tq_id = 3 (We will ignore the significance of DFO's (Data Flow Operation trees) at present.

PQ Stats (broadcast)

```
DFO NUMBER   TQ ID SERVER TYPE   INSTANCE PROCESS     NUM ROWS
         1       0 Producer (send)      1 P002              10
                                        1 P003               0       t3 scan
                   Consumer (receive)   1 P000               5
                                        1 P001               5       t3 hash

                 1 Producer             1 P002               8
                                        1 P003               0       t2 scan
                   Consumer             1 P000               4
                                        1 P001               4       t2 hash

                 2 Producer             1 P002               6
                                        1 P003               0       t1 scan
                   Consumer             1 P000               3
                                        1 P001               3       t1 hash

                 3 Producer             1 P000               1       t4 scan,
                                        1 P001               1       probe & ct
                   Consumer             1 QC                 2
```

group by and ***order by*** result in rows where the QC operates as server_type = Ranger

Running parallel 2: we see three consecutive jobs as p002 & p003 broadcast N x 2 rows to p000 & p001; then p000 & p001 produce the result of all three joins.

Trace files (broadcast)

In the previous slide slaves p000 and p001 scanned table t4 - so what do their trace files say about the work done - the estimate was to generate 343,000 rows before joining

```
| 20 |         PX BLOCK ITERATOR   |           |   343K|  Q1,03 | PCWC |          |
| 21 |          TABLE ACCESS FULL  | T4        |   343K|  Q1,03 | PCWP |          |
```

P000

```
STAT #N id=20 cnt=40 pid=15 pos=2 obj=0 op='PX BLOCK ITERATOR(card=343000)'
STAT #N id=21 cnt=40 pid=20 pos=1 obj=235635 op='TABLE ACCESS FULL T4(card=343000)'
```

P001

```
STAT #N id=20 cnt=20 pid=15 pos=2 obj=0 op='PX BLOCK ITERATOR(card=343000)'
STAT #N id=21 cnt=20 pid=20 pos=1 obj=235635 op='TABLE ACCESS FULL T4(card=343000)'
```

This shows a total of 60 rows returned from the table scan of t4 ***before*** the first join.

This is the effect of ***Bloom*** filtering.

On the Exadata database machine the Bloom filters can be sent to the storage server

The storage server can use storage indexes and smart scans to minimise disk and network load

A Bloom filter changes a join into a predicate. It eliminates (most of the) data that you don't want, allows all the data you do want - but may return some unwanted data

Parallel display_cursor()

```
|Id |Operation                   | Name    |E-Rows |   TQ  |IN-OUT| PQ Dist  |A-Rows|
|  0|SELECT STATEMENT            |         |       |       |      |          |     1|
|  1| SORT AGGREGATE             |         |     1 |       |      |          |     1|
|  2|  PX COORDINATOR            |         |       |       |      |          |     2|
|  3|   PX SEND QC (RANDOM)      | :TQ10003|     1 | Q1,03 | P->S | QC (RAND)|     0|
|  4|    SORT AGGREGATE          |         |     1 | Q1,03 | PCWP |          |     2|
|* 5|     HASH JOIN              |         |    56 | Q1,03 | PCWP |          |    60|
|  6|      PX RECEIVE            |         |     5 | Q1,03 | PCWP |          |    10|
|  7|       PX SEND BROADCAST    | :TQ10000|     5 | Q1,00 | P->P | BROADCAST|     0|
|  8|        PX BLOCK ITERATOR   |         |     5 | Q1,00 | PCWC |          |     5|
|* 9|         TABLE ACCESS FULL  | T3      |     5 | Q1,00 | PCWP |          |     5|
|*10|      HASH JOIN             |         |   810 | Q1,03 | PCWP |          |    60|
| 11|       PX RECEIVE           |         |     4 | Q1,03 | PCWP |          |     8|
| 12|        PX SEND BROADCAST   | :TQ10001|     4 | Q1,01 | P->P | BROADCAST|     0|
| 13|         PX BLOCK ITERATOR  |         |     4 | Q1,01 | PCWC |          |     4|
|*14|          TABLE ACCESS FULL | T2      |     4 | Q1,01 | PCWP |          |     4|
|*15|       HASH JOIN            |         | 14491 | Q1,03 | PCWP |          |    60|
| 16|        PX RECEIVE          |         |     3 | Q1,03 | PCWP |          |     6|
| 17|         PX SEND BROADCAST  | :TQ10002|     3 | Q1,02 | P->P | BROADCAST|     0|
| 18|          PX BLOCK ITERATOR |         |     3 | Q1,02 | PCWC |          |     3|
|*19|           TABLE ACCESS FULL| T1      |     3 | Q1,02 | PCWP |          |     3|
| 20|        PX BLOCK ITERATOR   |         |   343K| Q1,03 | PCWC |          |    60|
|*21|         TABLE ACCESS FULL  | T4      |   343K| Q1,03 | PCWP |          |    60|
```

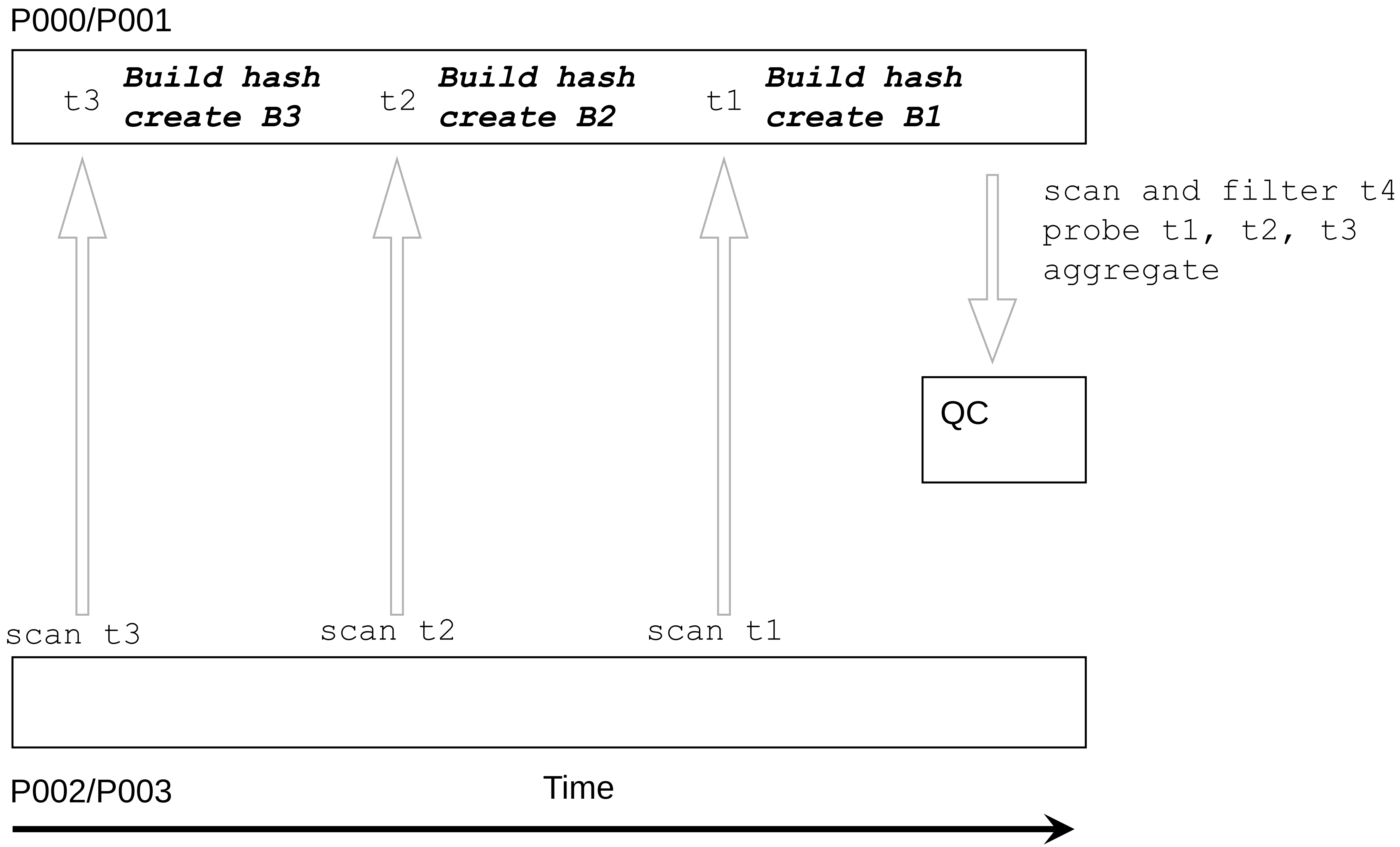
After running the query just once I've used the format option ***'allstats parallel'*** without ***"last"*** to get this output (and then I've deleted several columns).

Execution plan (broadcast)

```
Predicate Information (identified by operation id):
   5 - access("T3"."ID"="T4"."ID3")
   9 - access(:Z>=:Z AND :Z<=:Z)           -- check rowid ranges
       filter((   TO_NUMBER("T3"."SMALL_VC")=1 OR TO_NUMBER("T3"."SMALL_VC")=2
               OR TO_NUMBER("T3"."SMALL_VC")=3 OR TO_NUMBER("T3"."SMALL_VC")=4
               OR TO_NUMBER("T3"."SMALL_VC")=5))
  10 - access("T2"."ID"="T4"."ID2")
  14 - access(:Z>=:Z AND :Z<=:Z)
       filter((   TO_NUMBER("T2"."SMALL_VC")=1 OR TO_NUMBER("T2"."SMALL_VC")=2
               OR TO_NUMBER("T2"."SMALL_VC")=3 OR TO_NUMBER("T2"."SMALL_VC")=4))
  15 - access("T1"."ID"="T4"."ID1")
  19 - access(:Z>=:Z AND :Z<=:Z)
       filter((TO_NUMBER("T1"."SMALL_VC")=1 OR TO_NUMBER("T1"."SMALL_VC")=2 OR
              TO_NUMBER("T1"."SMALL_VC")=3))
  21 - access(:Z>=:Z AND :Z<=:Z)
       filter(SYS_OP_BLOOM_FILTER_LIST(
                 SYS_OP_BLOOM_FILTER(:BF0000,"T4"."ID1"),
                 SYS_OP_BLOOM_FILTER(:BF0000,"T4"."ID2"),
                 SYS_OP_BLOOM_FILTER(:BF0000,"T4"."ID3")
       ))        -- Blooom filters from all three dimensions used during tablescan
```

In 11g the predicate section shows that three Bloom filters were used during the ***t4*** tablescan. They identify which column is filtered on, but the BF numbering is odd.

Graphic (broadcast)



P000/P001

t3 ***Build hash create B3***

t2 ***Build hash create B2***

t1 ***Build hash create B1***

scan and filter t4
probe t1, t2, t3
aggregate

QC

scan t3

scan t2

scan t1

P002/P003

Time

12c Broadcast plan

```
| Id | Operation                    | Name     | Rows  | Bytes | Cost |   TQ  |IN-OUT| PQ Distrib|
|  0 | SELECT STATEMENT             |          |       |       |  351 |       |      |           |
|  1 |  SORT AGGREGATE              |          |     1 |    38 |      |       |      |           |
|  2 |   PX COORDINATOR             |          |       |       |      |       |      |           |
|  3 |    PX SEND QC (RANDOM)       | :TQ10000 |     1 |    38 |      | Q1,00 | P->S | QC (RAND) |
|  4 |     SORT AGGREGATE           |          |     1 |    38 |      | Q1,00 | PCWP |           |
|* 5 |      HASH JOIN               |          |    56 |  2128 |  351 | Q1,00 | PCWP |           |
|  6 |       JOIN FILTER CREATE     | :BF0000  |     5 |    30 |    2 | Q1,00 | PCWP |           |
|* 7 |        TABLE ACCESS FULL     | T3       |     5 |    30 |    2 | Q1,00 | PCWP |           |
|* 8 |       HASH JOIN              |          |   810 | 25920 |  349 | Q1,00 | PCWP |           |
|  9 |        JOIN FILTER CREATE    | :BF0001  |     4 |    24 |    2 | Q1,00 | PCWP |           |
|*10 |         TABLE ACCESS FULL    | T2       |     4 |    24 |    2 | Q1,00 | PCWP |           |
|*11 |        HASH JOIN             |          | 14491 |   367K|  347 | Q1,00 | PCWP |           |
| 12 |         JOIN FILTER CREATE   | :BF0002  |     3 |    18 |    2 | Q1,00 | PCWP |           |
|*13 |          TABLE ACCESS FULL   | T1       |     3 |    18 |    2 | Q1,00 | PCWP |           |
| 14 |         JOIN FILTER USE      | :BF0000  |   343K|  6699K|  345 | Q1,00 | PCWP |           |
| 15 |          JOIN FILTER USE     | :BF0001  |   343K|  6699K|  345 | Q1,00 | PCWP |           |
| 16 |           JOIN FILTER USE    | :BF0002  |   343K|  6699K|  345 | Q1,00 | PCWP |           |
| 17 |            PX BLOCK ITERATOR |          |   343K|  6699K|  345 | Q1,00 | PCWC |           |
|*18 |             TABLE ACCESS FULL| T4       |   343K|  6699K|  345 | Q1,00 | PCWP |           |
```

There are many differences in the 12c, which changes dramatically, caching scans, avoiding many table queues, and showing us where the filters are created and used.

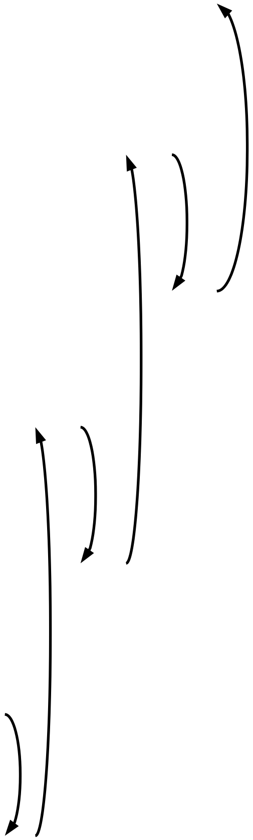
Execution plan *(hash / hash)*

```
| Id  | Operation                          | Name     | Rows  | Time     |    TQ  |IN-OUT| PQ Distrib |
|   0 | SELECT STATEMENT                   |          |       |          |        |      |            |
|   1 |  SORT AGGREGATE                    |          |     1 |          |        |      |            |
|   2 |   PX COORDINATOR                   |          |       |          |        |      |            |
|   3 |    PX SEND QC (RANDOM)             | :TQ10006 |     1 |          |  Q1,06 | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE                 |          |     1 |          |  Q1,06 | PCWP |            |
|*  5 |      HASH JOIN                     |          |    56 | 00:00:29 |  Q1,06 | PCWP |            |
|   6 |       JOIN FILTER CREATE           | :BF0000  |     5 | 00:00:01 |  Q1,06 | PCWP |            |
|   7 |        PX RECEIVE                  |          |     5 | 00:00:01 |  Q1,06 | PCWP |            |
|   8 |         PX SEND HASH               | :TQ10004 |     5 | 00:00:01 |  Q1,04 | P->P | HASH       |
|   9 |          PX BLOCK ITERATOR         |          |     5 | 00:00:01 |  Q1,04 | PCWC |            |
|* 10 |           TABLE ACCESS FULL        | T3       |     5 | 00:00:01 |  Q1,04 | PCWP |            |
|  11 |       PX RECEIVE                   |          |   810 | 00:00:29 |  Q1,06 | PCWP |            |
|  12 |        PX SEND HASH                | :TQ10005 |   810 | 00:00:29 |  Q1,05 | P->P | HASH       |
|  13 |         JOIN FILTER USE            | :BF0000  |   810 | 00:00:29 |  Q1,05 | PCWP |            |
|* 14 |          HASH JOIN BUFFERED        |          |   810 | 00:00:29 |  Q1,05 | PCWP |            |
|  15 |           JOIN FILTER CREATE       | :BF0001  |     4 | 00:00:01 |  Q1,05 | PCWP |            |
|  16 |            PX RECEIVE              |          |     4 | 00:00:01 |  Q1,05 | PCWP |            |
|  17 |             PX SEND HASH           | :TQ10002 |     4 | 00:00:01 |  Q1,02 | P->P | HASH       |
|  18 |              PX BLOCK ITERATOR     |          |     4 | 00:00:01 |  Q1,02 | PCWC |            |
|* 19 |               TABLE ACCESS FULL    | T2       |     4 | 00:00:01 |  Q1,02 | PCWP |            |
|  20 |           PX RECEIVE               |          | 14491 | 00:00:29 |  Q1,05 | PCWP |            |
|  21 |            PX SEND HASH            | :TQ10003 | 14491 | 00:00:29 |  Q1,03 | P->P | HASH       |
|  22 |             JOIN FILTER USE        | :BF0001  | 14491 | 00:00:29 |  Q1,03 | PCWP |            |
|* 23 |              HASH JOIN BUFFERED    |          | 14491 | 00:00:29 |  Q1,03 | PCWP |            |
|  24 |               JOIN FILTER CREATE   | :BF0002  |     3 | 00:00:01 |  Q1,03 | PCWP |            |
|  25 |                PX RECEIVE          |          |     3 | 00:00:01 |  Q1,03 | PCWP |            |
|  26 |                 PX SEND HASH       | :TQ10000 |     3 | 00:00:01 |  Q1,00 | P->P | HASH       |
|  27 |                  PX BLOCK ITERATOR |          |     3 | 00:00:01 |  Q1,00 | PCWC |            |
|* 28 |                   TABLE ACCESS FULL| T1       |     3 | 00:00:01 |  Q1,00 | PCWP |            |
|  29 |               PX RECEIVE           |          |   343K| 00:00:29 |  Q1,03 | PCWP |            |
|  30 |                PX SEND HASH        | :TQ10001 |   343K| 00:00:29 |  Q1,01 | P->P | HASH       |
|  31 |                 JOIN FILTER USE    | :BF0002  |   343K| 00:00:29 |  Q1,01 | PCWP |            |
|  32 |                  PX BLOCK ITERATOR |          |   343K| 00:00:29 |  Q1,01 | PCWC |            |
|* 33 |                   TABLE ACCESS FULL| T4       |   343K| 00:00:29 |  Q1,01 | PCWP |            |
```

We've now gone from 22 lines to 34 lines, but we can still see the shape and order of the original four table hash join. (***Hash Join Buffered*** is a threat!)

Execution plan (hash / hash)

```
| Id  | Operation                          | Name     | Rows  | Time     |    TQ  |IN-OUT| PQ Distrib |
|   0 | SELECT STATEMENT                   |          |       |          |        |      |            |
|   1 |  SORT AGGREGATE                    |          |     1 |          |        |      |            |
|   2 |   PX COORDINATOR                   |          |       |          |        |      |            |
|   3 |    PX SEND QC (RANDOM)             | :TQ10006 |     1 |          |  Q1,06 | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE                 |          |     1 |          |  Q1,06 | PCWP |            |
|*  5 |      HASH JOIN                     |          |    56 | 00:00:29 |  Q1,06 | PCWP |            |
|   6 |       JOIN FILTER CREATE           | :BF0000  |     5 | 00:00:01 |  Q1,06 | PCWP |            |
|   7 |        PX RECEIVE                  |          |     5 | 00:00:01 |  Q1,06 | PCWP |            |
|   8 |         PX SEND HASH               | :TQ10004 |     5 | 00:00:01 |  Q1,04 | P->P | HASH       |
|   9 |          PX BLOCK ITERATOR         |          |     5 | 00:00:01 |  Q1,04 | PCWC |            |
|* 10 |           TABLE ACCESS FULL        | T3       |     5 | 00:00:01 |  Q1,04 | PCWP |            |
|  11 |       PX RECEIVE                   |          |   810 | 00:00:29 |  Q1,06 | PCWP |            |
|  12 |        PX SEND HASH                | :TQ10005 |   810 | 00:00:29 |  Q1,05 | P->P | HASH       |
|  13 |         JOIN FILTER USE            | :BF0000  |   810 | 00:00:29 |  Q1,05 | PCWP |            |
|* 14 |          HASH JOIN BUFFERED        |          |   810 | 00:00:29 |  Q1,05 | PCWP |            |
|  15 |           JOIN FILTER CREATE       | :BF0001  |     4 | 00:00:01 |  Q1,05 | PCWP |            |
|  16 |            PX RECEIVE              |          |     4 | 00:00:01 |  Q1,05 | PCWP |            |
|  17 |             PX SEND HASH           | :TQ10002 |     4 | 00:00:01 |  Q1,02 | P->P | HASH       |
|  18 |              PX BLOCK ITERATOR     |          |     4 | 00:00:01 |  Q1,02 | PCWC |            |
|* 19 |               TABLE ACCESS FULL    | T2       |     4 | 00:00:01 |  Q1,02 | PCWP |            |
|  20 |           PX RECEIVE               |          | 14491 | 00:00:29 |  Q1,05 | PCWP |            |
|  21 |            PX SEND HASH            | :TQ10003 | 14491 | 00:00:29 |  Q1,03 | P->P | HASH       |
|  22 |             JOIN FILTER USE        | :BF0001  | 14491 | 00:00:29 |  Q1,03 | PCWP |            |
|* 23 |              HASH JOIN BUFFERED    |          | 14491 | 00:00:29 |  Q1,03 | PCWP |            |
|  24 |               JOIN FILTER CREATE   | :BF0002  |     3 | 00:00:01 |  Q1,03 | PCWP |            |
|  25 |                PX RECEIVE          |          |     3 | 00:00:01 |  Q1,03 | PCWP |            |
|  26 |                 PX SEND HASH       | :TQ10000 |     3 | 00:00:01 |  Q1,00 | P->P | HASH       |
|  27 |                  PX BLOCK ITERATOR |          |     3 | 00:00:01 |  Q1,00 | PCWC |            |
|* 28 |                   TABLE ACCESS FULL| T1       |     3 | 00:00:01 |  Q1,00 | PCWP |            |
|  29 |               PX RECEIVE           |          |   343K| 00:00:29 |  Q1,03 | PCWP |            |
|  30 |                PX SEND HASH        | :TQ10001 |   343K| 00:00:29 |  Q1,01 | P->P | HASH       |
|  31 |                 JOIN FILTER USE    | :BF0002  |   343K| 00:00:29 |  Q1,01 | PCWP |            |
|  32 |                  PX BLOCK ITERATOR |          |   343K| 00:00:29 |  Q1,01 | PCWC |            |
|* 33 |                   TABLE ACCESS FULL| T4       |   343K| 00:00:29 |  Q1,01 | PCWP |            |
```



We now have ***seven*** table queues to follow. Notice how they don't follow a simple consecutive pattern up the plan, though.

Execution plan (hash / hash)

```
| Id  | Operation                           | Name       | Rows  | Time      |    TQ  |IN-OUT| PQ Distrib |
|   0 | SELECT STATEMENT                    |            |       |           |        |      |            |
|   1 |  SORT AGGREGATE                     |            |     1 |           |        |      |            |
|   2 |   PX COORDINATOR                    |            |       |           |        |      |            |
|   3 |    PX SEND QC (RANDOM)              | :TQ10006 |     1 |           |  Q1,06 | P->S | QC (RAND)  |
|   4 |     SORT AGGREGATE                  |            |     1 |           |  Q1,06 | PCWP |            |
|*  5 |      HASH JOIN                      |            |    56 |  00:00:29 |  Q1,06 | PCWP |            |
|   6 |       JOIN FILTER CREATE            | :BF0000  |     5 |  00:00:01 |  Q1,06 | PCWP |            |
|   7 |        PX RECEIVE                   |            |     5 |  00:00:01 |  Q1,06 | PCWP |            |
|   8 |         PX SEND HASH                | :TQ10004 |     5 |  00:00:01 |  Q1,04 | P->P | HASH       |
|   9 |          PX BLOCK ITERATOR          |            |     5 |  00:00:01 |  Q1,04 | PCWC |            |
|* 10 |           TABLE ACCESS FULL         | T3         |     5 |  00:00:01 |  Q1,04 | PCWP |            |
|  11 |       PX RECEIVE                    |            |   810 |  00:00:29 |  Q1,06 | PCWP |            |
|  12 |        PX SEND HASH                 | :TQ10005 |   810 |  00:00:29 |  Q1,05 | P->P | HASH       |
|  13 |         JOIN FILTER USE             | :BF0000  |   810 |  00:00:29 |  Q1,05 | PCWP |            |
|* 14 |          HASH JOIN BUFFERED         |            |   810 |  00:00:29 |  Q1,05 | PCWP |            |
|  15 |           JOIN FILTER CREATE        | :BF0001  |     4 |  00:00:01 |  Q1,05 | PCWP |            |
|  16 |            PX RECEIVE               |            |     4 |  00:00:01 |  Q1,05 | PCWP |            |
|  17 |             PX SEND HASH            | :TQ10002 |     4 |  00:00:01 |  Q1,02 | P->P | HASH       |
|  18 |              PX BLOCK ITERATOR      |            |     4 |  00:00:01 |  Q1,02 | PCWC |            |
|* 19 |               TABLE ACCESS FULL     | T2         |     4 |  00:00:01 |  Q1,02 | PCWP |            |
|  20 |           PX RECEIVE                |            | 14491 |  00:00:29 |  Q1,05 | PCWP |            |
|  21 |            PX SEND HASH             | :TQ10003 | 14491 |  00:00:29 |  Q1,03 | P->P | HASH       |
|  22 |             JOIN FILTER USE         | :BF0001  | 14491 |  00:00:29 |  Q1,03 | PCWP |            |
|* 23 |              HASH JOIN BUFFERED     |            | 14491 |  00:00:29 |  Q1,03 | PCWP |            |
|  24 |               JOIN FILTER CREATE    | :BF0002  |     3 |  00:00:01 |  Q1,03 | PCWP |            |
|  25 |                PX RECEIVE           |            |     3 |  00:00:01 |  Q1,03 | PCWP |            |
|  26 |                 PX SEND HASH        | :TQ10000 |     3 |  00:00:01 |  Q1,00 | P->P | HASH       |
|  27 |                  PX BLOCK ITERATOR |            |     3 |  00:00:01 |  Q1,00 | PCWC |            |
|* 28 |                   TABLE ACCESS FULL| T1         |     3 |  00:00:01 |  Q1,00 | PCWP |            |
|  29 |               PX RECEIVE            |            |   343K|  00:00:29 |  Q1,03 | PCWP |            |
|  30 |                PX SEND HASH         | :TQ10001 |   343K|  00:00:29 |  Q1,01 | P->P | HASH       |
|  31 |                 JOIN FILTER USE     | :BF0002  |   343K|  00:00:29 |  Q1,01 | PCWP |            |
|  32 |                  PX BLOCK ITERATOR |            |   343K|  00:00:29 |  Q1,01 | PCWC |            |
|* 33 |                   TABLE ACCESS FULL| T4         |   343K|  00:00:29 |  Q1,01 | PCWP |            |
```

We create the filter ***after*** we receive the build table (T1)

We use it ***before*** we send the probe table

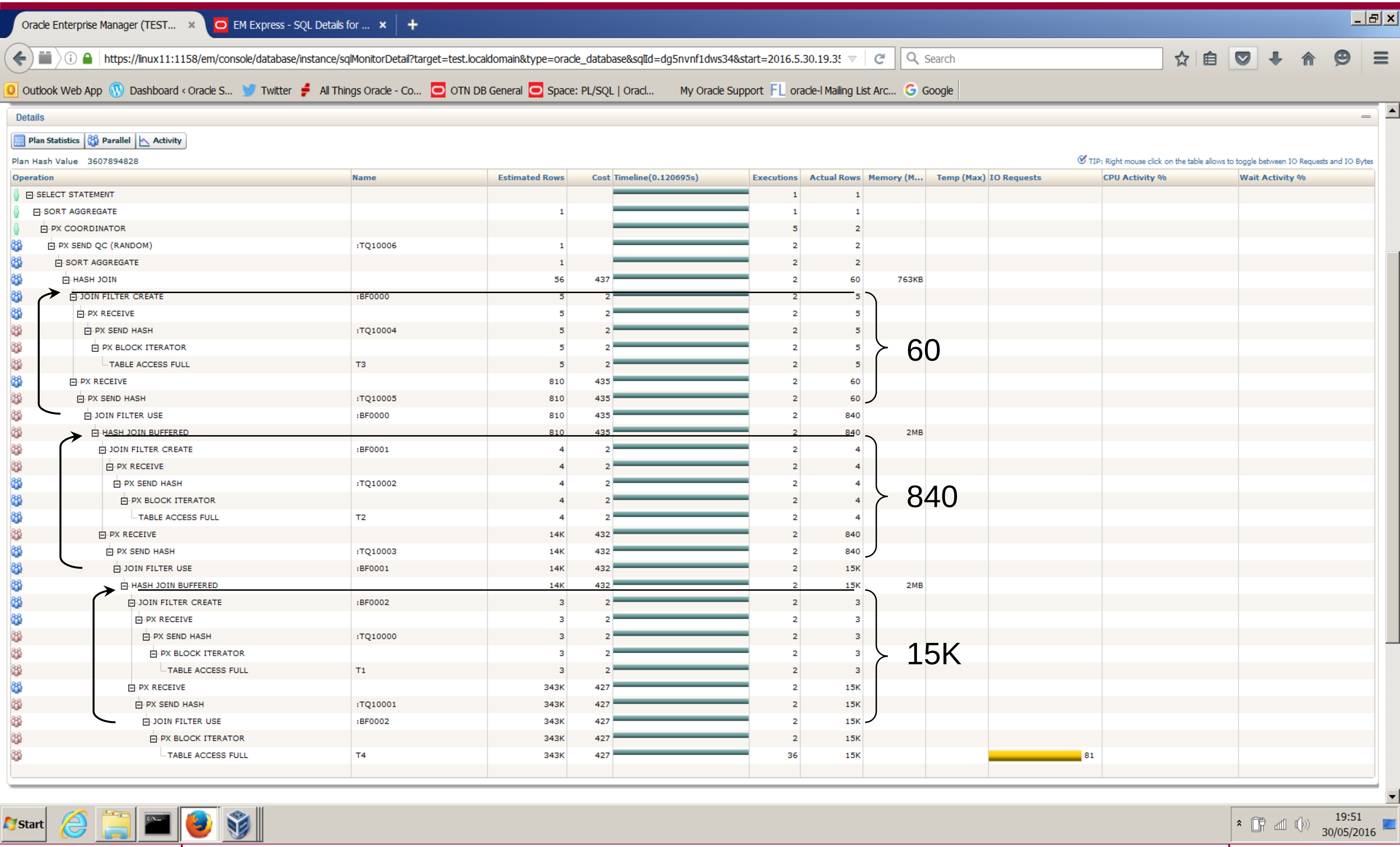
The plan includes several pairs of lines showing the creation and use of Bloom filters (We have to ignore the BF numbers as they don't agree with the order of creation).

Execution plan (hash / hash)

```
Predicate Information (identified by operation id):
   5 - access("T3"."ID"="T4"."ID3")
  10 - access(:Z>=:Z AND :Z<=:Z)
       filter((TO_NUMBER("T3"."SMALL_VC")=1 OR TO_NUMBER("T3"."SMALL_VC")=2 OR
              TO_NUMBER("T3"."SMALL_VC")=3))
  14 - access("T2"."ID"="T4"."ID2")
  19 - access(:Z>=:Z AND :Z<=:Z)
       filter((TO_NUMBER("T2"."SMALL_VC")=1 OR TO_NUMBER("T2"."SMALL_VC")=2 OR
              TO_NUMBER("T2"."SMALL_VC")=3))
  23 - access("T1"."ID"="T4"."ID1")
  28 - access(:Z>=:Z AND :Z<=:Z)
       filter((TO_NUMBER("T1"."SMALL_VC")=1 OR TO_NUMBER("T1"."SMALL_VC")=2 OR
              TO_NUMBER("T1"."SMALL_VC")=3))
  33 - access(:Z>=:Z AND :Z<=:Z)
       filter(SYS_OP_BLOOM_FILTER(:BF0000,"T4"."ID1"))
```

Although the plan says we created and used three Bloom filters the predicate section only reports using one of them. We need to check execution stats.

OEM monitor (11g)


Oracle Enterprise Manager (TEST...
EM Express - SQL Details for ...
https://linux11:1158/em/console/database/instance/sqlMonitorDetail?target=test.localdomain&type=oracle_database&sqlId=dg5nvnf1dws34&start=2016.5.30.19.3!
Search
Outlook Web App
Dashboard ‹ Oracle S...
Twitter
All Things Oracle - Co...
OTN DB General
Space: PL/SQL | Oracl...
My Oracle Support
oracle-l Mailing List Arc...
Google
Details
Plan Statistics
Parallel
Activity
TIP: Right mouse click on the table allows to toggle between IO Requests and IO Bytes
Start
19:51
30/05/2016

Plan Hash Value 3607894828

Operation	Name	Estimated Rows	Cost	Timeline(0.120695s)	Executions	Actual Rows	Memory (M...	Temp (Max)	IO Requests	CPU Activity %	Wait Activity %
SELECT STATEMENT					1	1					
SORT AGGREGATE		1			1	1					
PX COORDINATOR					5	2					
PX SEND QC (RANDOM)	:TQ10006	1			2	2					
SORT AGGREGATE		1			2	2					
HASH JOIN		56	437		2	60	763KB				
JOIN FILTER CREATE	:BF0000	5	2		2	5					
PX RECEIVE		5	2		2	5					
PX SEND HASH	:TQ10004	5	2		2	5					
PX BLOCK ITERATOR		5	2		2	5					
TABLE ACCESS FULL	T3	5	2		2	5					
PX RECEIVE		810	435		2	60					
PX SEND HASH	:TQ10005	810	435		2	60					
JOIN FILTER USE	:BF0000	810	435		2	840					
HASH JOIN BUFFERED		810	435		2	840	2MB				
JOIN FILTER CREATE	:BF0001	4	2		2	4					
PX RECEIVE		4	2		2	4					
PX SEND HASH	:TQ10002	4	2		2	4					
PX BLOCK ITERATOR		4	2		2	4					
TABLE ACCESS FULL	T2	4	2		2	4					
PX RECEIVE		14K	432		2	840					
PX SEND HASH	:TQ10003	14K	432		2	840					
JOIN FILTER USE	:BF0001	14K	432		2	15K					
HASH JOIN BUFFERED		14K	432		2	15K	2MB				
JOIN FILTER CREATE	:BF0002	3	2		2	3					
PX RECEIVE		3	2		2	3					
PX SEND HASH	:TQ10000	3	2		2	3					
PX BLOCK ITERATOR		3	2		2	3					
TABLE ACCESS FULL	T1	3	2		2	3					
PX RECEIVE		343K	427		2	15K					
PX SEND HASH	:TQ10001	343K	427		2	15K					
JOIN FILTER USE	:BF0002	343K	427		2	15K					
PX BLOCK ITERATOR		343K	427		2	15K					
TABLE ACCESS FULL	T4	343K	427		36	15K			81		

60

840

15K

Compare the actual rows with estimates and you can see the "actual rows" figures drop by N/70 ***before*** each join as each Bloom filter is used.

PQ Stats (hash / hash)

```
DFO_NUMBER  TQ_ID SERVER_TYPE  INSTANCE PROCESS   NUM_ROWS
         1      0 Producer            1 P002             3  scan t1
                                      1 P003             0  pass to 0/1 to build
                  Consumer            1 P000             2
                                      1 P001             1  return filter (b1)

                1 Producer            1 P002          7297  scan t4 filter (b1)
                                      1 P003          7405  Pass to 0/1
                  Consumer            1 P000          9801           buffer
                                      1 P001          4901           buffer

                2 Producer            1 P000             4  scan t2
                                      1 P001             0  pass to 2/3 to build
                  Consumer            1 P002             3
                                      1 P003             1  return filter (b2)
```

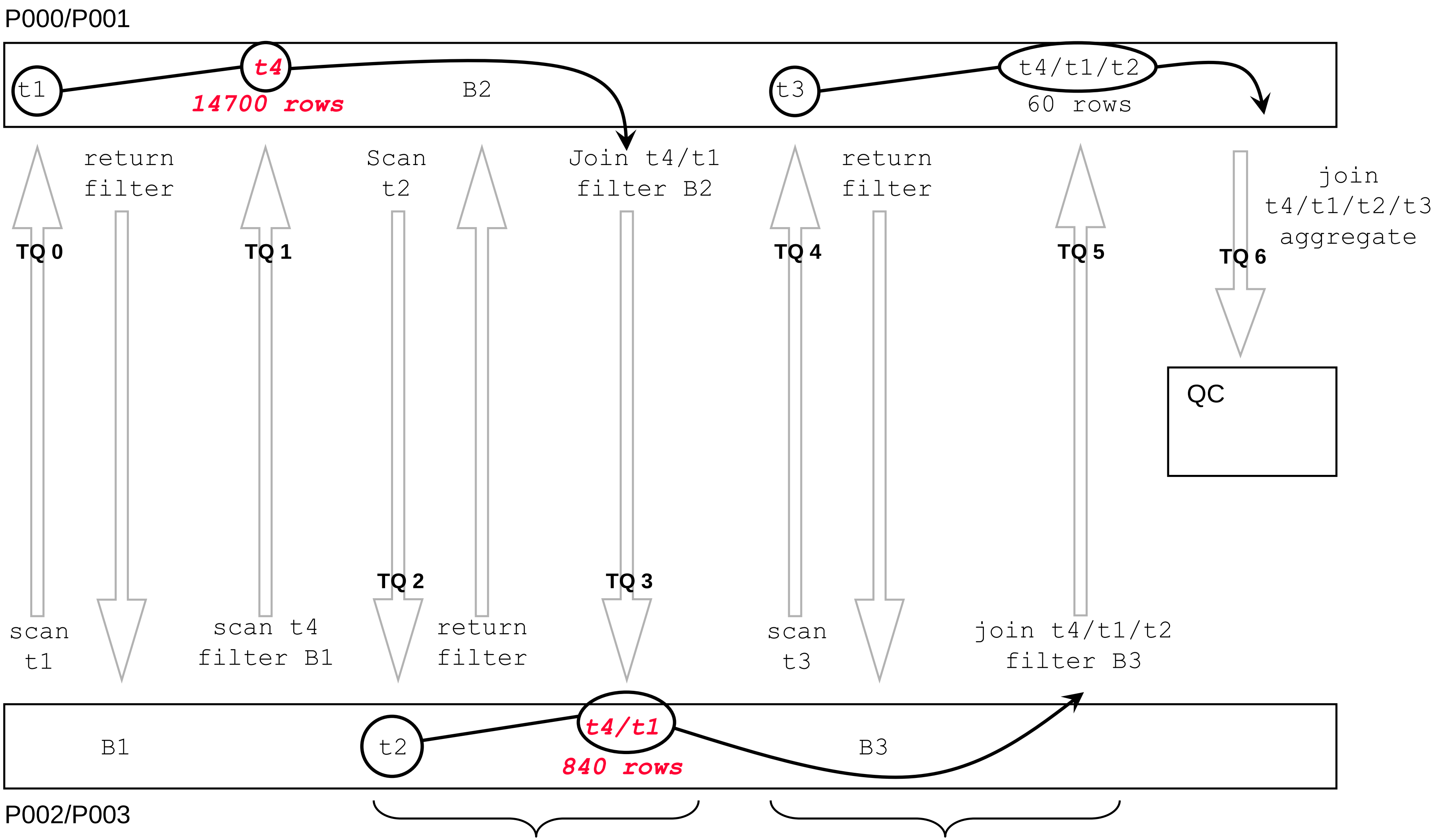
The first two steps of the TQ stats show slave set 2 scanning t1 then scanning t4 with a Bloom filter - but slave set 1 doesn't join the two rowsources straight away.

PQ Stats (hash / hash)

```
DFO_NUMBER  TQ_ID SERVER_TYPE  INSTANCE PROCESS   NUM_ROWS
                3 Producer            1 P000           560   t1/t4 filter (b2)
                                      1 P001           282   Pass to 2/3
                  Consumer            1 P002           632           buffer
                                      1 P003           210           buffer

                4 Producer            1 P002             5   scan t3
                                      1 P003             0   pass to 0/1 to build
                  Consumer            1 P000             4
                                      1 P001             1   return filter (b3)

                5 Producer            1 P002            45   t2/(t1/t4) filter (b3)
                                      1 P003            15
                  Consumer            1 P000            48   Pass to 0/1
                                      1 P001            12

                6 Producer            1 P000             1   join t3/(t2/(t1/t4)
                                      1 P001             1   and aggregate results
                  Consumer            1 QC               2
```

After sending t2 to slave set 2, slave set 1 joins t1 and t4 and sends the result to slave set 2 - but slave set 2 doesn't join these two rowsources straight away.

Graphic (hash / hash)



Buffering - we don't join then buffer, we just buffer the incoming probe data.
Note the recurring groups of 3 - scan X, get filter X, join previous and use filter X.

Observations

- Follow the TQxxyyyy name order - within DFO tree
  - "Name" = :TQxxyyy and "TQ" = Qxx,yyyy
- Hash Join ***Buffered*** may spill the "large table" to disc
  - Use lots of memory and broadcast
- Bloom filters "hide" (in 11g)
  - Look at v$pq_tqstat, 10046, OEM Monitor (v$sql_monitor)
- Bloom filter numbering is "wrong"
  - (The same is true of DFO trees)
- Keep an eye on v$pq_tqstat for uneven distribution
  - But it has many limitations. SQL Monitor is far better if licensed