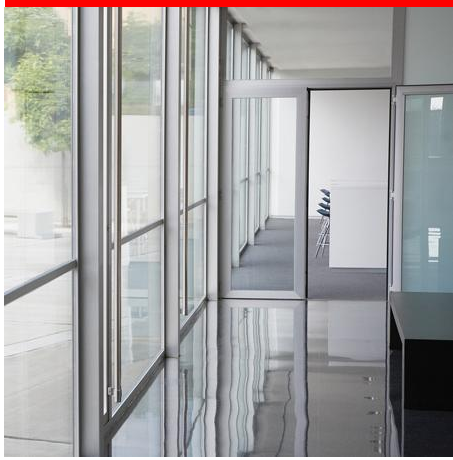

ORACLE®

ORACLE®

Best Practices for Extreme Performance with Data Warehousing on Oracle Database

Rekha Balwada
Principal Product Manager

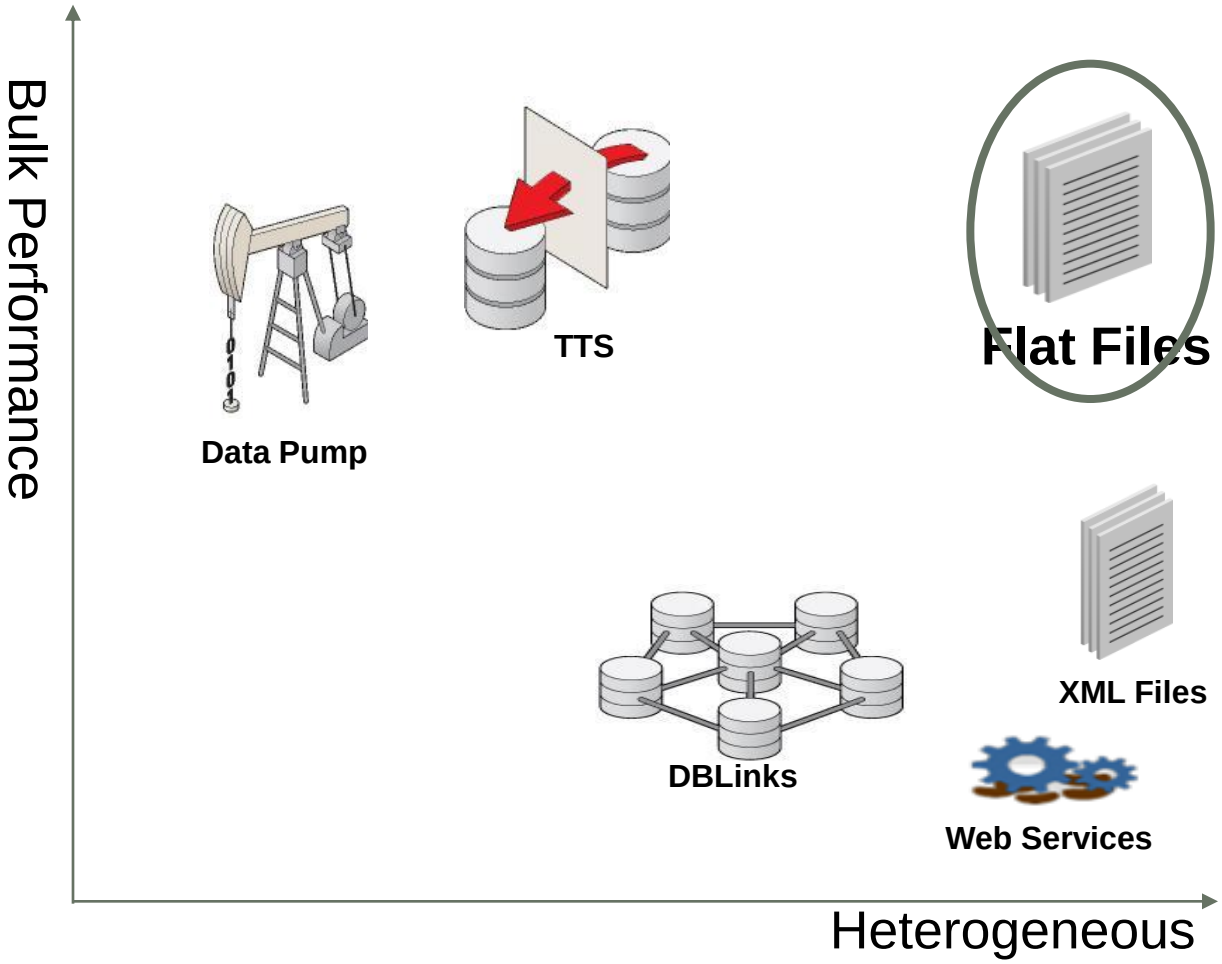
Agenda



- Data Loading
- Partitioning
- Parallel
- Workload Management on a Data Warehouse

Data Loading

Oracle offers many data loading Options. Which of the following options should you choice to achieve a high performance loads?



Data Loading Best Practices

- **External Tables**
  - Allows flat file to be accessed via SQL PL/SQL as if it was a table
  - Enables complex data transformations & data cleansing to occur “on the fly”
- **Pre-processing**
  - Ability to specify a program that the access driver will execute to read the data
    - Specify gunzip to decompress a .gzip file “on the fly” while its being accessed
  - Size versus speed trade-off
- **Direct Path in parallel**
  - Bypasses buffer cache and writes data directly to disk via multi-block async IO
  - Use parallel to speed up load
  - Remember to use `Alter session enable parallel DML`
- **Range Partitioning**
  - Enables partition exchange loads
- **Data Compression**

Why External tables and not SQL Loader?

- Full usage of SQL capabilities directly on the data
- Automatic use of parallel capabilities (just like a table)
- No need to stage the data again
- Better allocation of space when storing data
  - High watermark brokering
  - AUTOALLOCATE tablespace will trim extents after the load
- Interesting capabilities like
  - The usage of data pump
  - The usage of pre-processing

Preparing the raw data

- Typically, Oracle automatically divides up the files to be loaded in 10MB granules
  - True for position-able and seek-able files
  - Exceptions are compressed files, data read from a pipe or a tape
- If no granules can be created then parallelism-per-file
  - Manual granules (multiple input files) must be used
  - The number of files determines the maximum DOP
- General rules of thumb
  - If using multiple files keep them similar in size
  - If files sizes vary significantly, list them largest to smallest in the external table definition

Pre-Processing in an External Table

- New functionality in 11.1.0.7 and 10.2.0.5
- Allows flat files to be processed automatically during load
  - Decompression of large file zipped files
- Pre-processing doesn't support automatic granulation
  - Need to supply multiple data files - number of files will determine DOP
- Need to GRANT READ, EXECUTE privileges directories

```
CREATE TABLE sales_external (…)
    ORGANIZATION EXTERNAL
    (   TYPE ORACLE_LOADER
    DEFAULT DIRECTORY data_dir1
        ACCESS PARAMETERS
    (RECORDS DELIMITED BY NEWLINE
     PREPROCESSOR exec_dir: 'zcat'
          FIELDS TERMINATED BY '|'
       )
      LOCATION (…)
          );
```

Direct Path Load

- Data is written directly to the database storage using multiple blocks per I/O request using asynchronous writes
- A **CTAS** command always uses direct path
- An Insert As Select (**IAS**) needs an APPEND hint to go direct
  - Don't forget to COMMIT after loading

```
INSERT /*+ APPEND */
INTO sales PARTITION(p2)
SELECT *
FROM ext_tab_for_sales_data;
COMMIT;
```

- Only one direct path operation can occur on an object
  - By specifying a specific partition name in the table you can do multiple concurrent direct path loads into a table

Parallel Load

- Ensure direct path loads go parallel
  - Specify parallel degree either with hint or on both tables
  - Enable parallelism by issuing alter session command
- CTAS will go parallel automatically when DOP is specified
  - Remember that the DOP for creation is also the DOP for further query processing
- IAS will not – it needs parallel DML to be enabled

```
ALTER SESSION ENABLE PARALLEL DML;
```

Data Loading in Parallel into Partitioned tables

- Use external tables and a direct path loading technique
  - Allows for real parallel DML statements with full SQL capabilities
- Minimum of 4GB of memory per CPU core
  - Ensures each parallel process can get enough memory
- Use AutoAllocate tablespaces (No Uniform extents)
  - Prevent “holes” appearing in your data set
- Set large initial and next extents (8MB)
  - Prevent the partitioned table from having to many extents which impacts scan rates
- Use multiple tablespaces for partitioned tables
  - Prevent parallel processes from contending for single file header

Best Practices for Data Warehousing

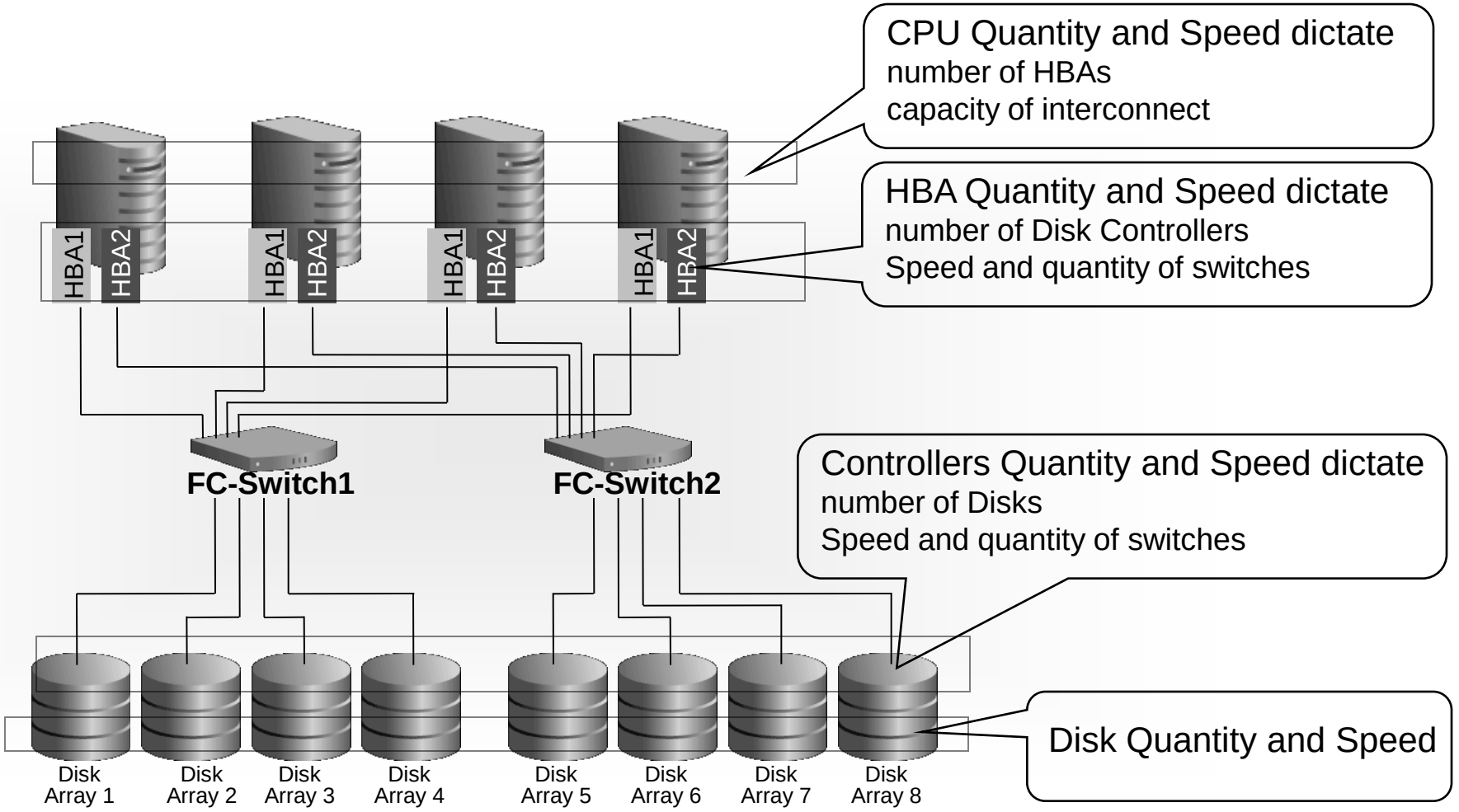
3 Ps - Power, Partitioning, Parallelism

- **Power** – A Balanced Hardware Configuration
  - Weakest link defines the throughput
- **Partition** larger tables or fact tables
  - Facilitates data load, data elimination and join performance
  - Enables easier Information Lifecycle Management
- **Parallel** Execution should be used
  - Instead of one process doing all the work multiple processes working concurrently on smaller units
  - Parallel degree should be power of 2

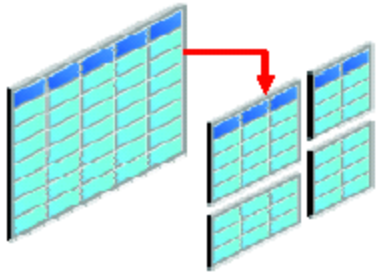
Goal is to minimize the amount of data accessed and use the most efficient joins

Balanced Configuration

"The weakest link" defines the throughput

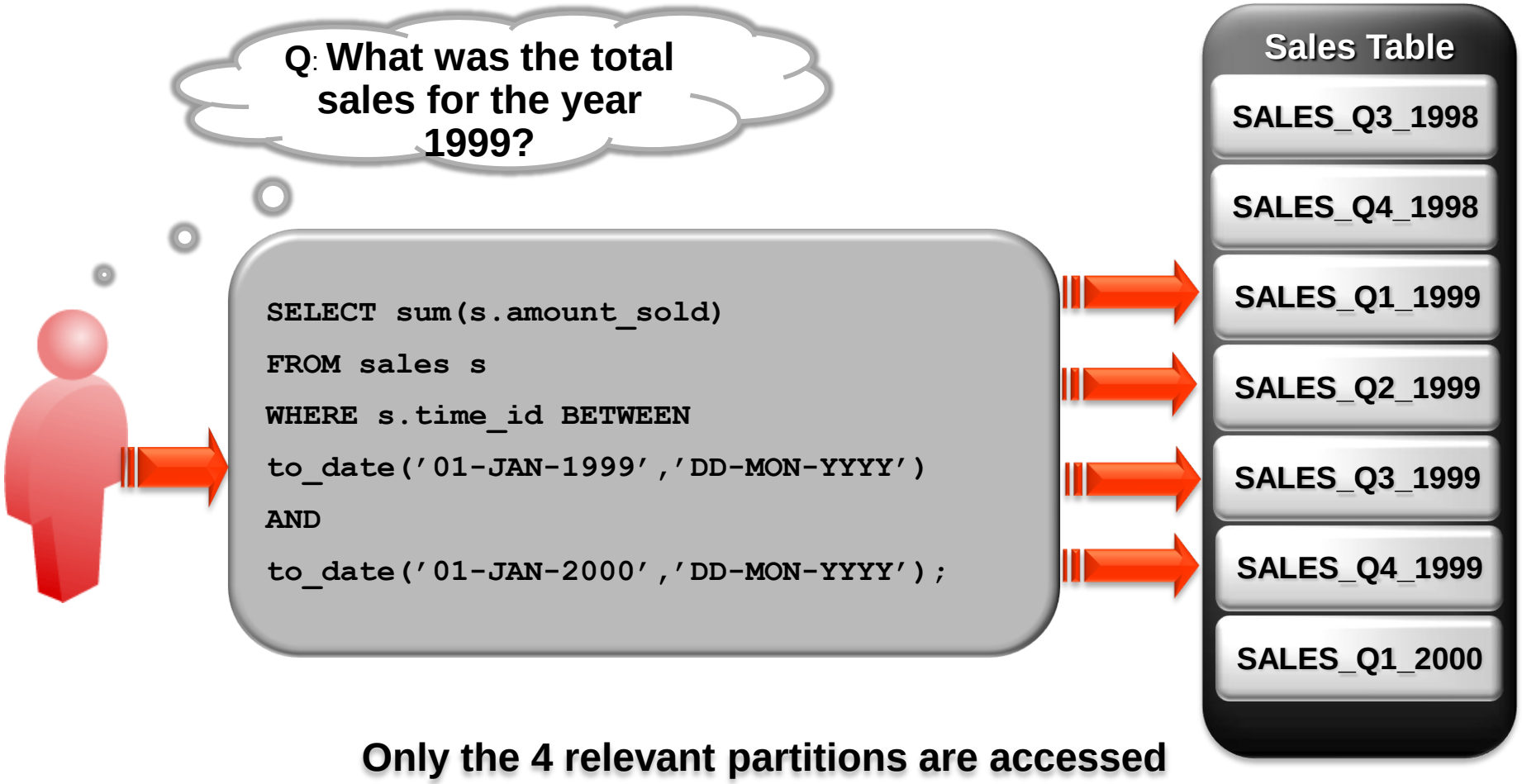


Partitioning



- First level partitioning
  - Goal to enable partitioning pruning and simplify data management
  - Most typical Range or interval partitioning on date column
  - How do you decide partitioning strategy?
    - What range of data do the queries touch - a quarter, a year?
    - Consider the data loading frequency
      - Is an incremental load required?
      - How much data is involved, a day, a week, a month?
- Second level of partitioning
  - Goal allow for multi-level pruning and improve join performance
  - Most typical hash or list
  - How do you decide partitioning strategy?
    - Select the dimension queried most frequently on the fact table OR
    - Pick the common join column

Partition Pruning



Only the 4 relevant partitions are accessed

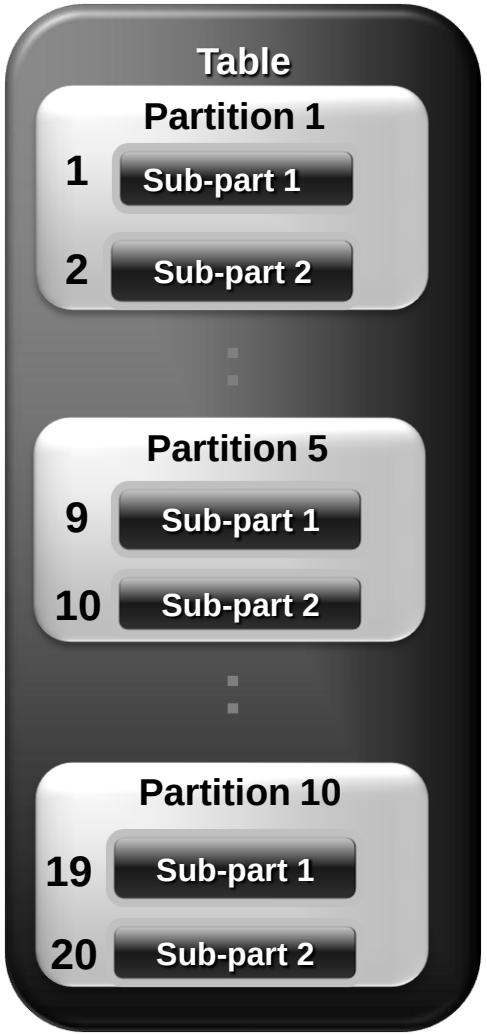
Monitoring Partition Pruning
Static Pruning

- Sample plan

```
 select sum(amount_sold) FROM sh.sales s, sh.times t WHERE s.time_id =
t.time_id AND s.time_id between TO_DATE('01-JAN-1999','DD-MON-YYYY')
and TO_DATE('01-JAN-2000','DD-MON-YYYY')

Plan hash value: 2025449199

-------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |       |       |       |     3 (100)|          |       |       |
|   1 |  SORT AGGREGATE            |       |     1 |    12 |            |          |       |       |
|   2 |   PARTITION RANGE ITERATOR|       |   313 |  3756 |     3   (0)| 00:00:01 |     9 |    13 |
|*  3 |    TABLE ACCESS FULL       | SALES |   313 |  3756 |     3   (0)| 00:00:01 |     9 |    13 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("S"."TIME_ID"<=TO_DATE(' 2000-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))

22 rows selected.
```

Numbering of Partitions

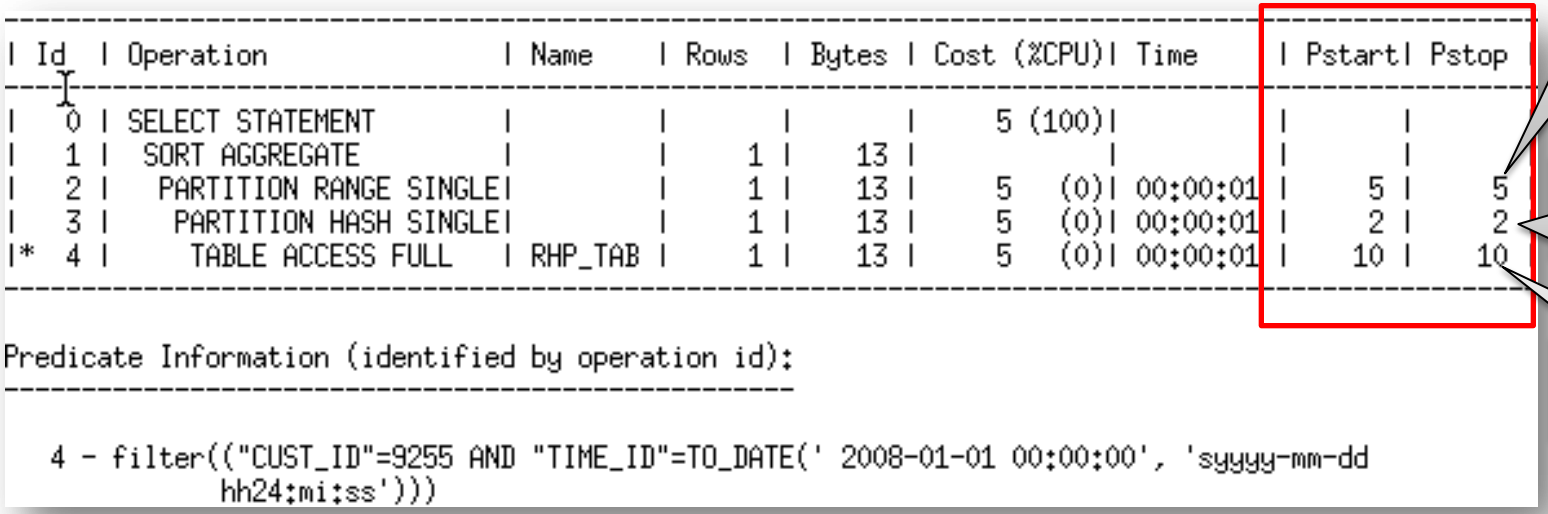
- An execution plan show partition numbers for static pruning
  - Partition numbers used can be relative and/or absolute



25

Monitoring Partition Pruning
Static Pruning

- Simple Query : SELECT COUNT(*)
  FROM RHP_TAB
  WHERE CUST_ID = 9255
  AND TIME_ID = '2008-01-01';
- Why do we see so many numbers in the Pstart / Pstop columns for such a simple query?

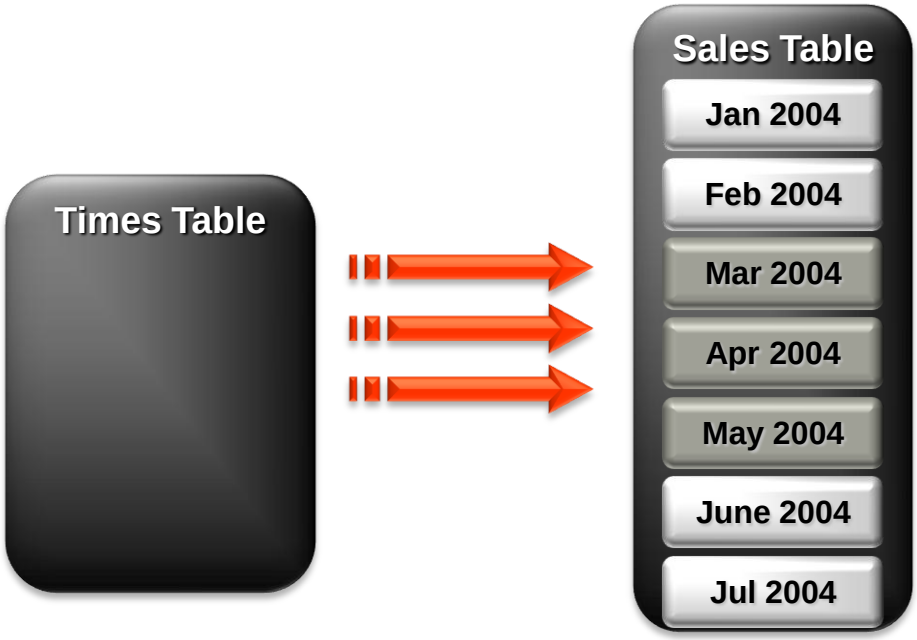


```
| Id  | Operation                | Name    | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
|   0 | SELECT STATEMENT         |         |       |       |     5 (100)|          |       |       |
|   1 |  SORT AGGREGATE          |         |     1 |    13 |            |          |       |       |
|   2 |   PARTITION RANGE SINGLE |         |     1 |    13 |     5   (0)| 00:00:01 |     5 |     5 |
|   3 |    PARTITION HASH SINGLE |         |     1 |    13 |     5   (0)| 00:00:01 |     2 |     2 |
|*  4 |     TABLE ACCESS FULL    | RHP_TAB |     1 |    13 |     5   (0)| 00:00:01 |    10 |    10 |

Predicate Information (identified by operation id):

   4 - filter(("CUST_ID"=9255 AND "TIME_ID"=TO_DATE(' 2008-01-01 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss')))
```

Monitoring Partition Pruning
Dynamic Partition Pruning

- Advanced Pruning mechanism for complex queries
- Recursive statement evaluates the relevant partitions at runtime
  - Look for the word ‘KEY’ in PSTART/PSTOP columns in the Plan

```
SELECT sum(amount_sold)
FROM sales s, times t
WHERE t.time_id = s.time_id
AND   t.calendar_month_desc IN
      (‘MAR-04’,‘APR-04’,‘MAY-04’);
```



Monitoring Partition Pruning

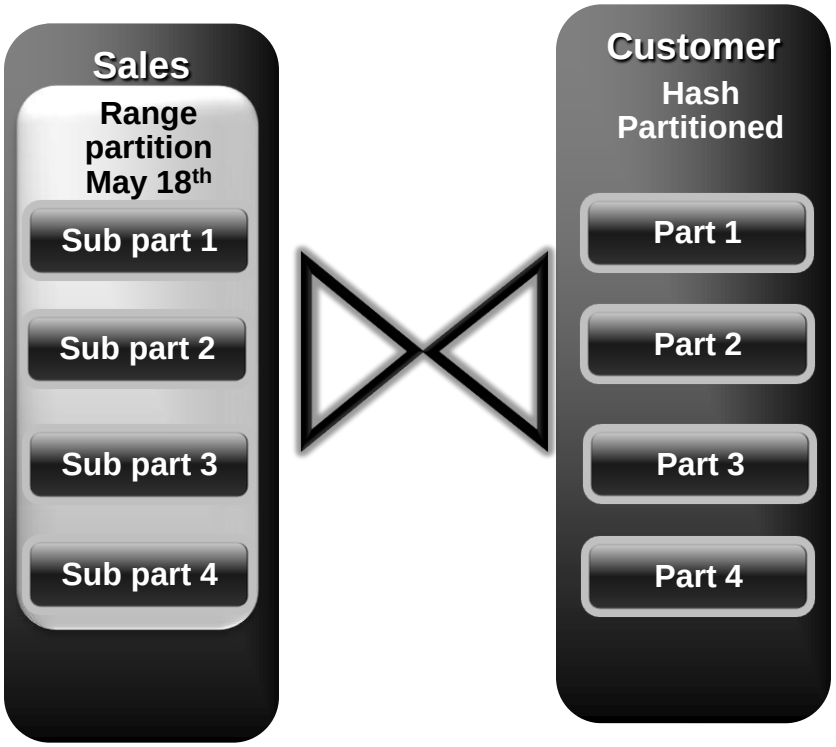
Dynamic Partition Pruning

- Sample plan

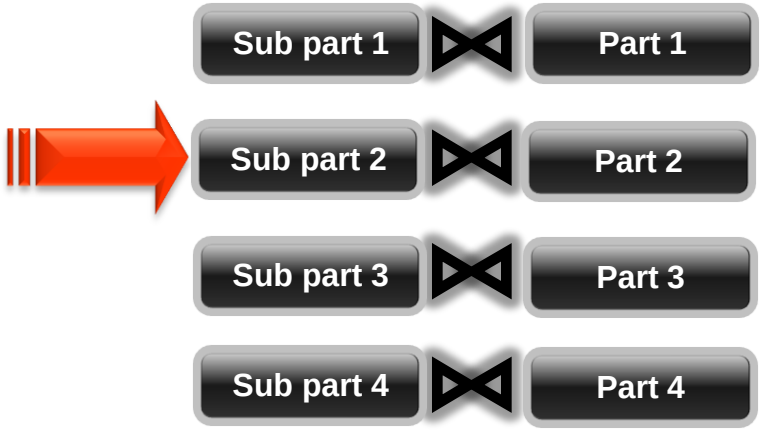
```
select sum(amount_sold) from sales s, times t where t.time_id =
s.time_id and t.calendar_month_desc in
('MAR-2004','APR-2004','MAY-2004')

Plan hash value: 1350851517

-------------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name  | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |       |       |       |    13 (100)|          |       |       |
|   1 |  SORT AGGREGATE               |       |     1 |    28 |            |          |       |       |
|   2 |   NESTED LOOPS                |       |     2 |    56 |    13   (0)| 00:00:01 |       |       |
|*  3 |    TABLE ACCESS FULL          | TIMES |     2 |    32 |    13   (8)| 00:00:01 |       |       |
|   4 |    PARTITION RANGE ITERATOR   |       |     2 |    24 |     0   (0)|          |   KEY |   KEY |
|*  5 |     TABLE ACCESS FULL         | SALES |     2 |    24 |     0   (0)|          |   KEY |   KEY |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter(("T"."CALENDAR_MONTH_DESC"='APR-2004' OR "T"."CALENDAR_MONTH_DESC"='MAR-2004'
              OR "T"."CALENDAR_MONTH_DESC"='MAY-2004'))
   5 - filter("T"."TIME_ID"="S"."TIME_ID")


26 rows selected.
```

Partition Wise join

```
SELECT sum(amount_sold)
FROM sales s, customer c
WHERE  s.cust_id=c.cust_id;
```



Both tables have the same degree of parallelism and are partitioned the same way on the join column (cust_id)



A large join is divided into multiple smaller joins, each joins a pair of partitions in parallel

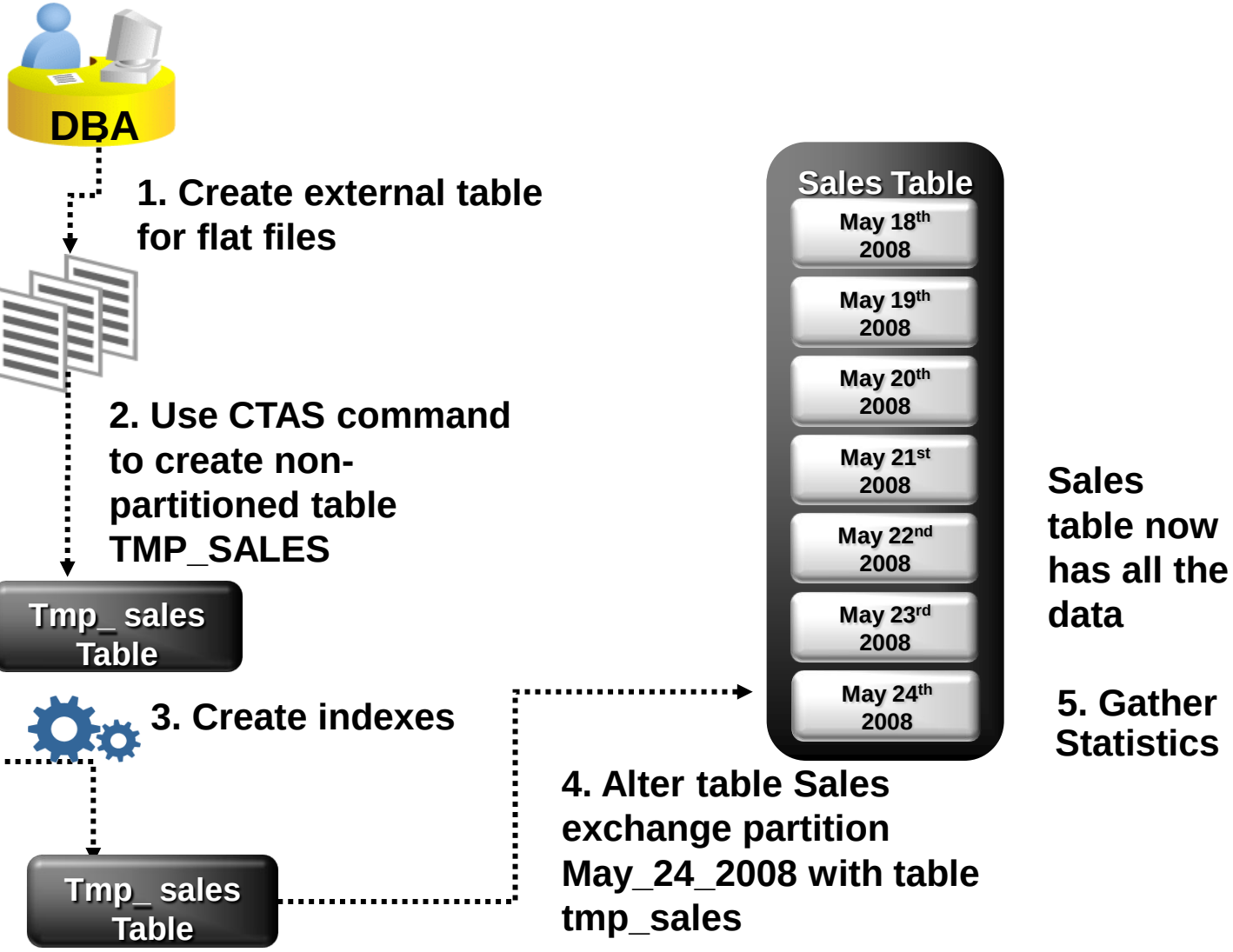
Monitoring of partition-wise join

Partition Hash All above the join method

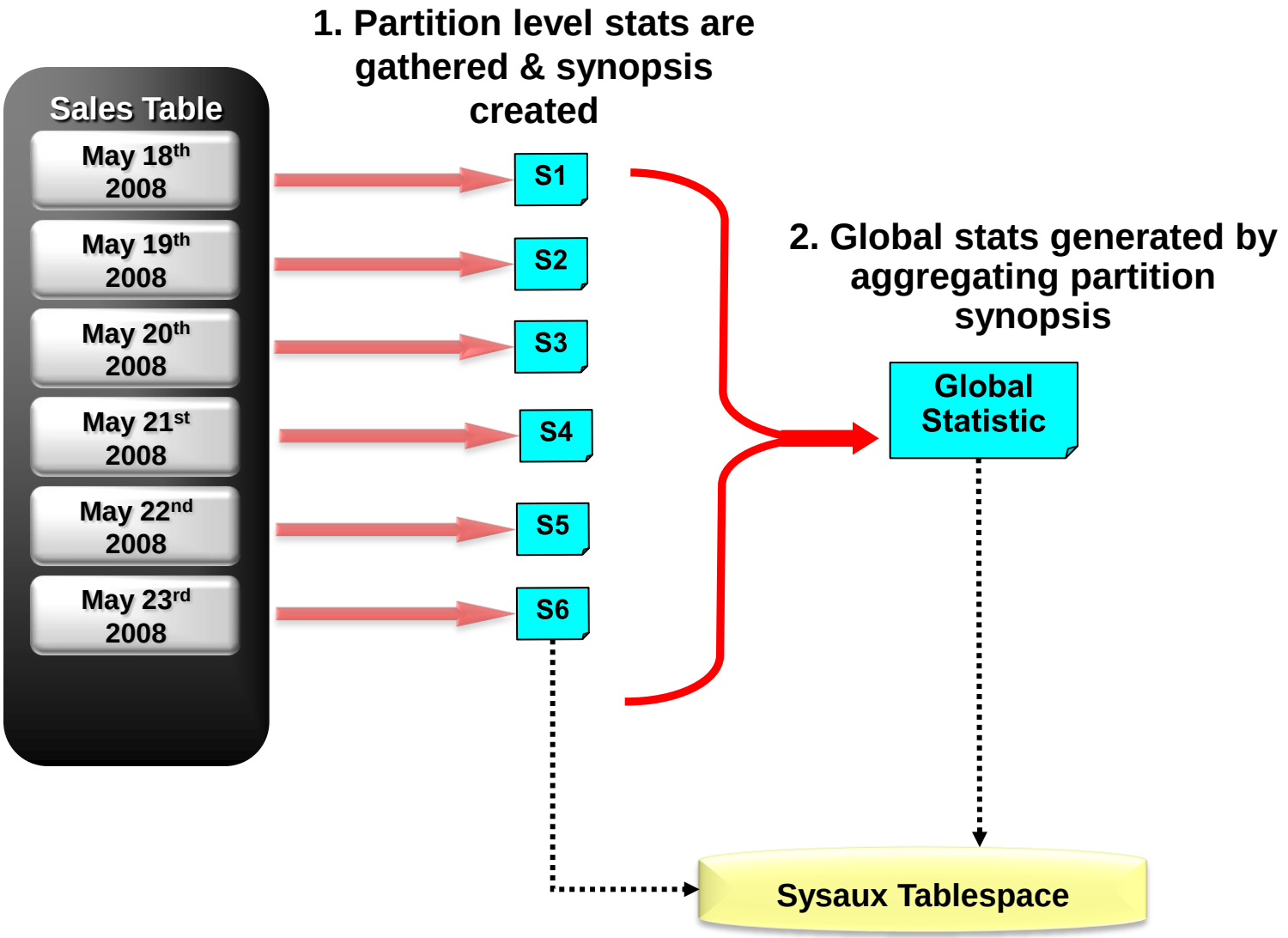
Indicates it's a partition-wise join

```
-------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name       | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |            |       |       |   914 (100)|          |       |       |
|   1 |  SORT AGGREGATE          |            |     1 |    39 |            |          |       |       |
|   2 |   PARTITION HASH ALL     |            |   988K|    36M|   914   (3)| 00:00:11 |     1 |     4 |
|*  3 |    HASH JOIN             |            |   988K|    36M|   914   (3)| 00:00:11 |       |       |
|   4 |     TABLE ACCESS FULL    | CUSTOMERS2 | 48431 |   614K|   416   (1)| 00:00:05 |     1 |     4 |
|   5 |     PARTITION RANGE ALL  |            |   988K|    24M|   491   (3)| 00:00:06 |     1 |    29 |
|   6 |      TABLE ACCESS FULL   | SALES2     |   988K|    24M|   491   (3)| 00:00:06 |     1 |   116 |
-------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("S"."CUST_ID"="C"."CUST_ID")
```

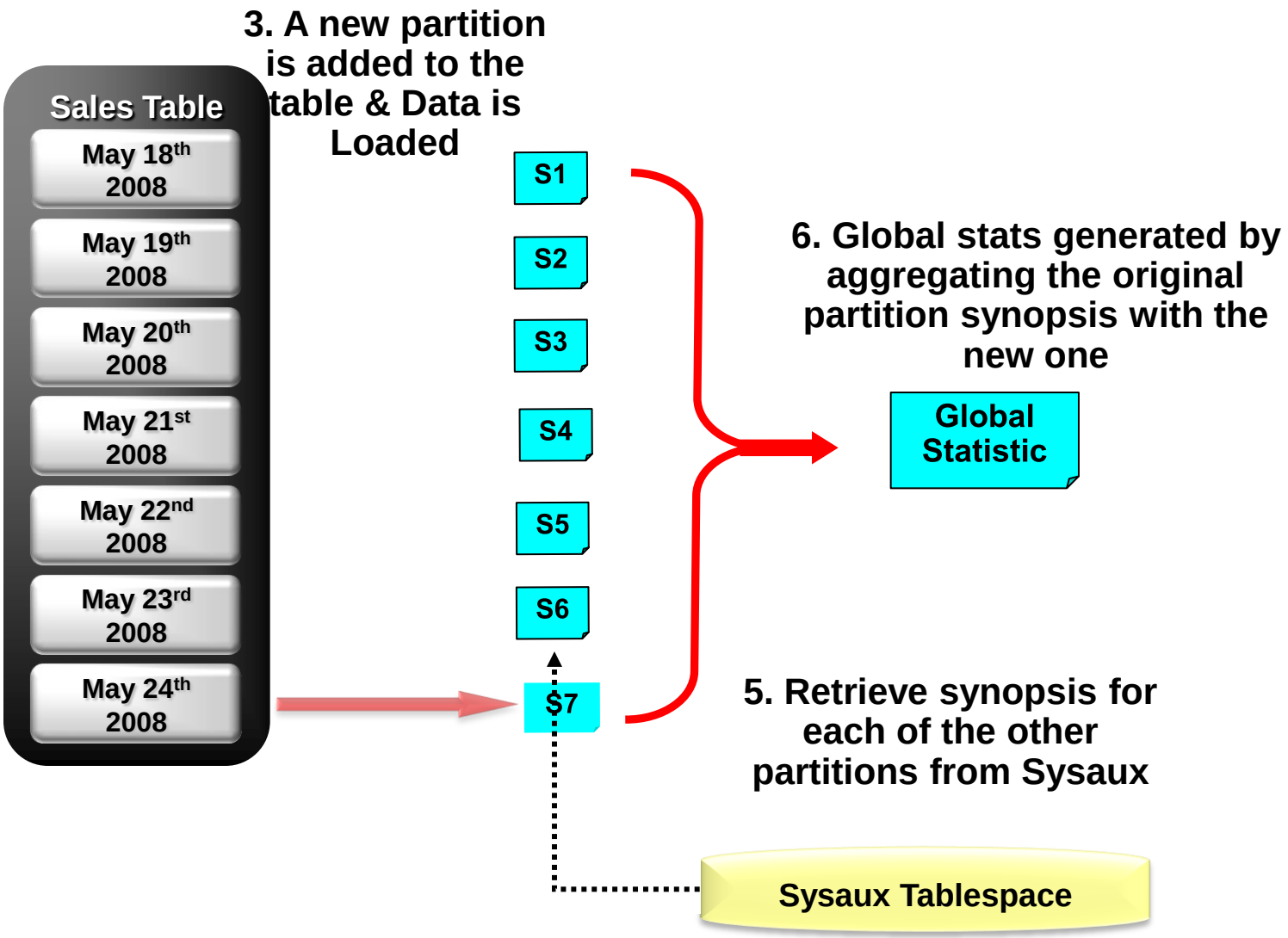
Partition Exchange Loading



Incremental Global Statistics



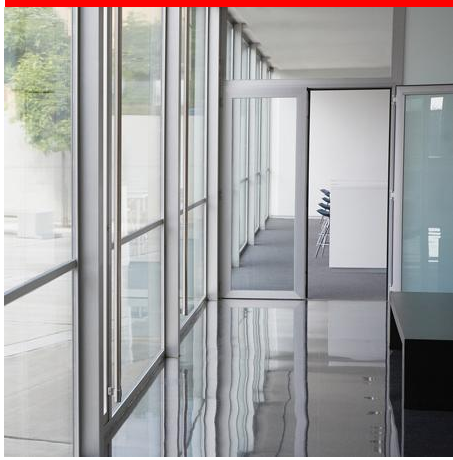
Incremental Global Statistics Cont'd



Things to keep in mind when using partition

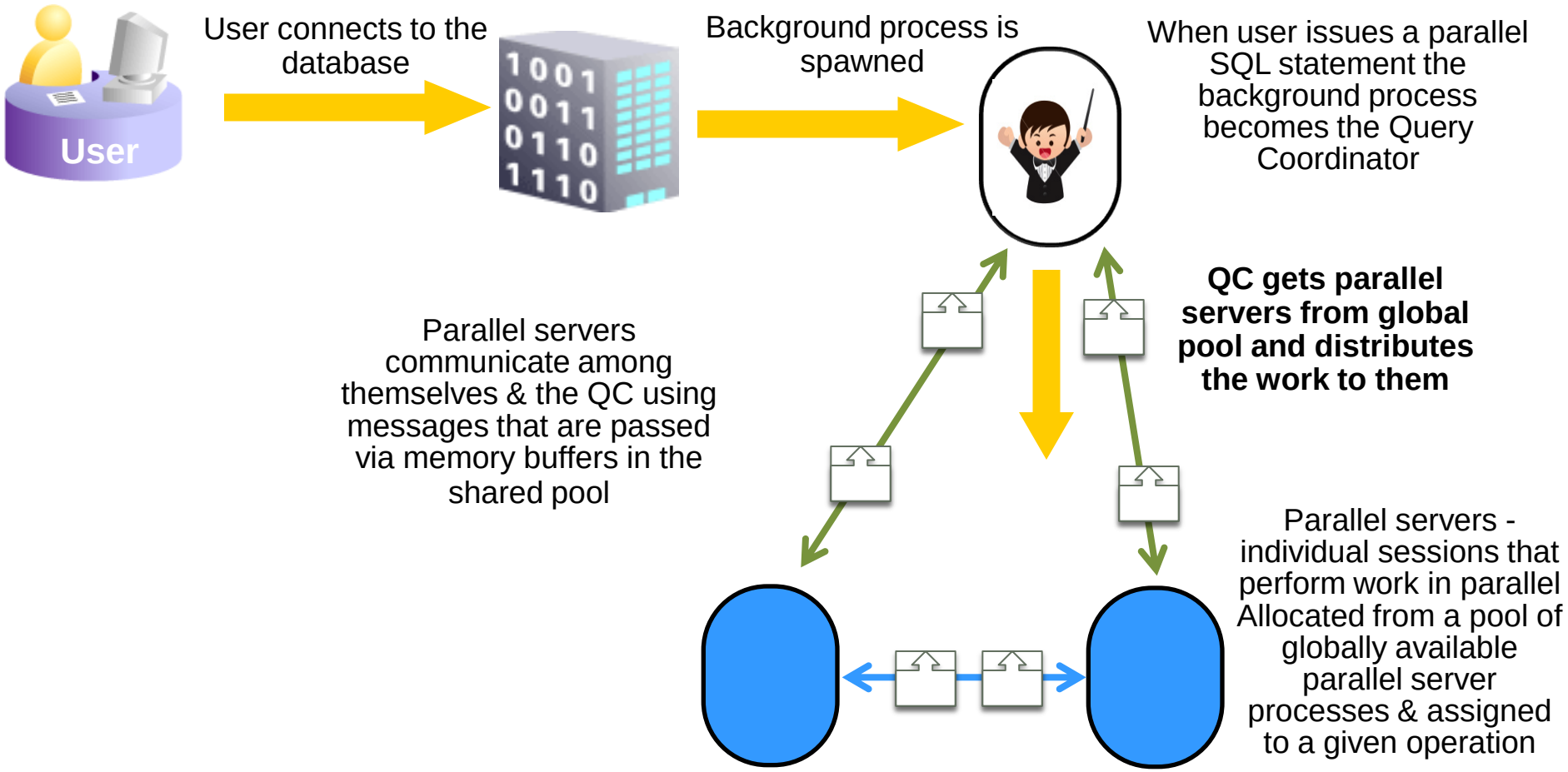
- Partition pruning on hash partitions only works for equality or in-list where clause predicates
- Partition pruning on multi-column hash partitioning only works if there is a predicate on all columns used
- To get a partition-wise join when using parallel query make sure the DOP is equal to or a multiple of the number of partitions
- If you load data into a new partition every day and users immediately start querying it, copy the statistics from the previous partition until you have time to gather stats (DBMS_STATS.COPY_TABLE_STATS)

Agenda



- Data Warehousing Reference Architecture
- The three Ps of Data Warehousing
  - Power
  - Partitioning
  - Parallel
- Workload Management on a Data Warehouse

How Parallel Execution works



Monitoring Parallel Execution

```
SELECT  c.cust_last_name, s.time_id, s.amount_sold
FROM sales s, customers c
WHERE s.cust_id = c.cust_id;
```

Query Coordinator

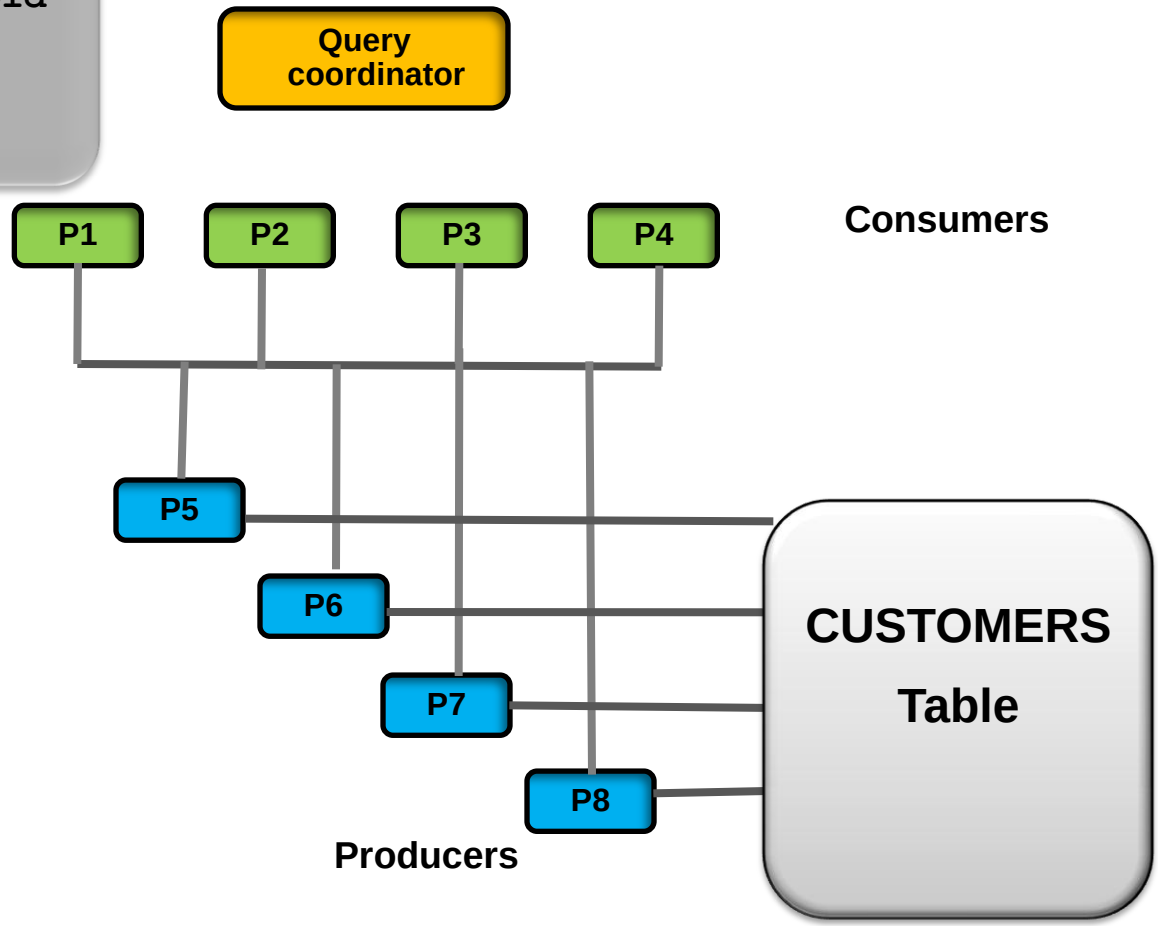
Id	Operation	Name	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT			311 (100)	
1	PX COORDINATOR				
2	PX SEND QC (RANDOM)	:TQ10002	1049K	31M 311 (2)	00:00:04
* 3	HASH JOIN BUFFERED		1049K	31M 311 (2)	00:00:04
4	PX RECEIVE		55500	704K 112 (0)	00:00:02
5	PX SEND HASH	:TQ10000	55500	704K 112 (0)	00:00:02
6	PX BLOCK ITERATOR		55500	704K 112 (0)	00:00:02
* 7	TABLE ACCESS FULL	CUSTOMERS	55500	704K 112 (0)	00:00:02
8	PX RECEIVE		1049K	18M 196 (2)	00:00:03
9	PX SEND HASH	:TQ10001	1049K	18M 196 (2)	00:00:03
10	PX BLOCK ITERATOR		1049K	18M 196 (2)	00:00:03
* 11	TABLE ACCESS FULL	SALES		18M 196 (2)	00:00:03

Parallel Servers

do majority of the work

How Parallel Execution works

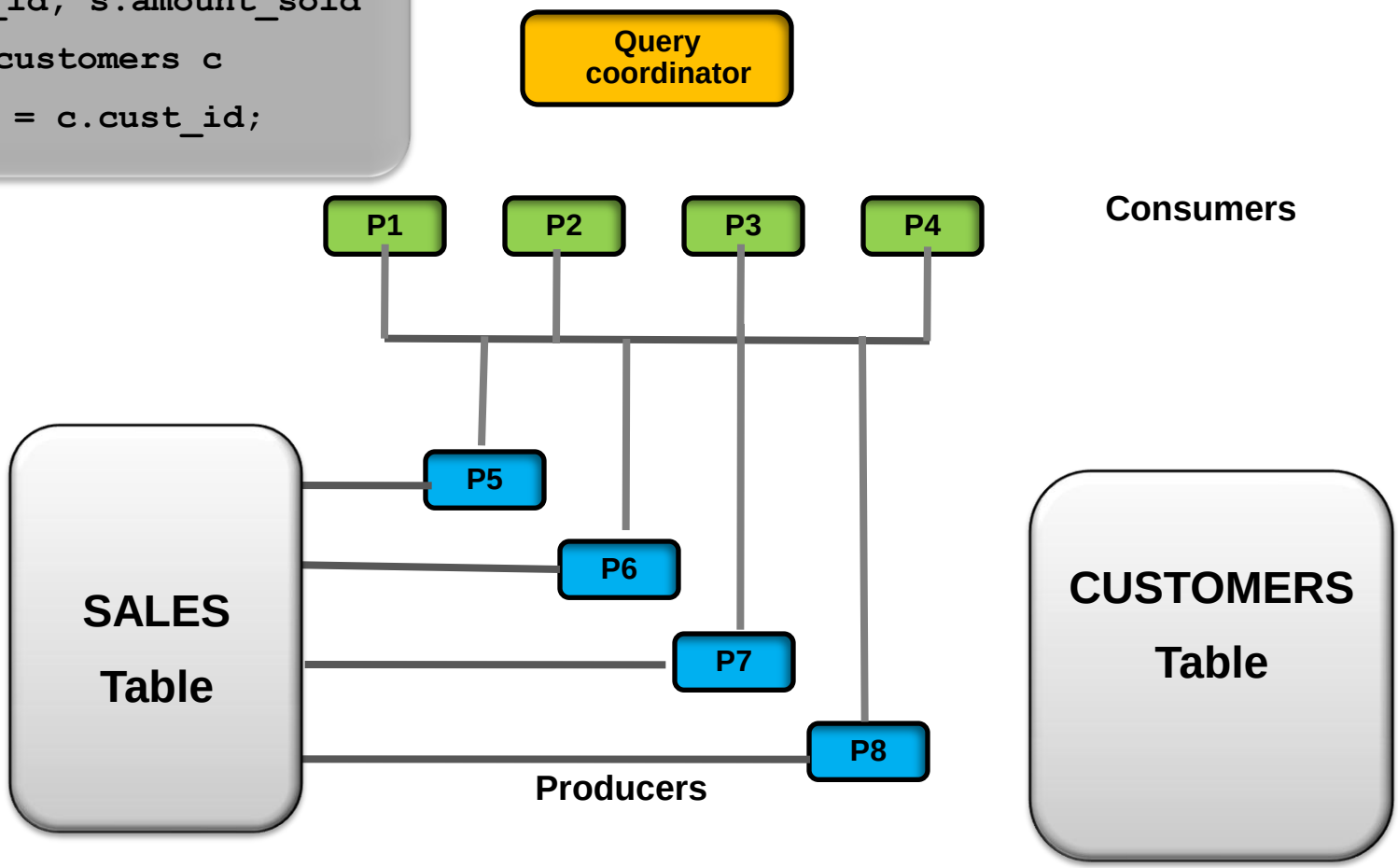
```
SELECT c.cust_last_name,
       s.time_id, s.amount_sold
FROM  sales s, customers c
WHERE s.cust_id = c.cust_id;
```



Hash join always begins with a scan of the smaller table. In this case that's is the customer table. The 4 producers scan the customer table and send the resulting rows to the consumers

How Parallel Execution works

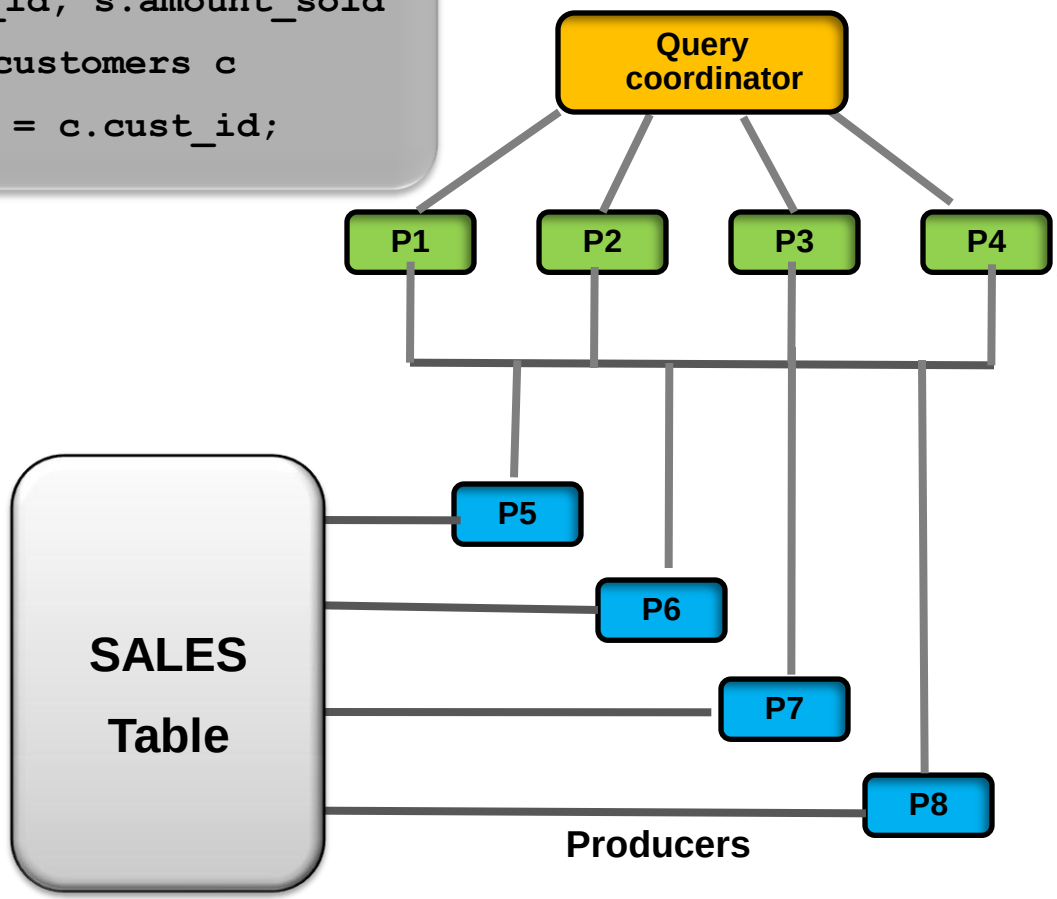
```
SELECT c.cust_last_name,
       s.time_id, s.amount_sold
FROM  sales s, customers c
WHERE s.cust_id = c.cust_id;
```



Once the 4 producers finish scanning the customer table, they start to scan the Sales table and send the resulting rows to the consumers

How Parallel Execution works

```
SELECT c.cust_last_name,
       s.time_id, s.amount_sold
FROM  sales s, customers c
WHERE s.cust_id = c.cust_id;
```



Consumers

Once the consumers receive the rows from the sales table they begin to do the join. Once completed they return the results to the QC

Monitoring Parallel Execution

```
SELECT  c.cust_last_name, s.time_id, s.amount_sold
FROM sales s, customers c
WHERE s.cust_id = c.cust_id;
```

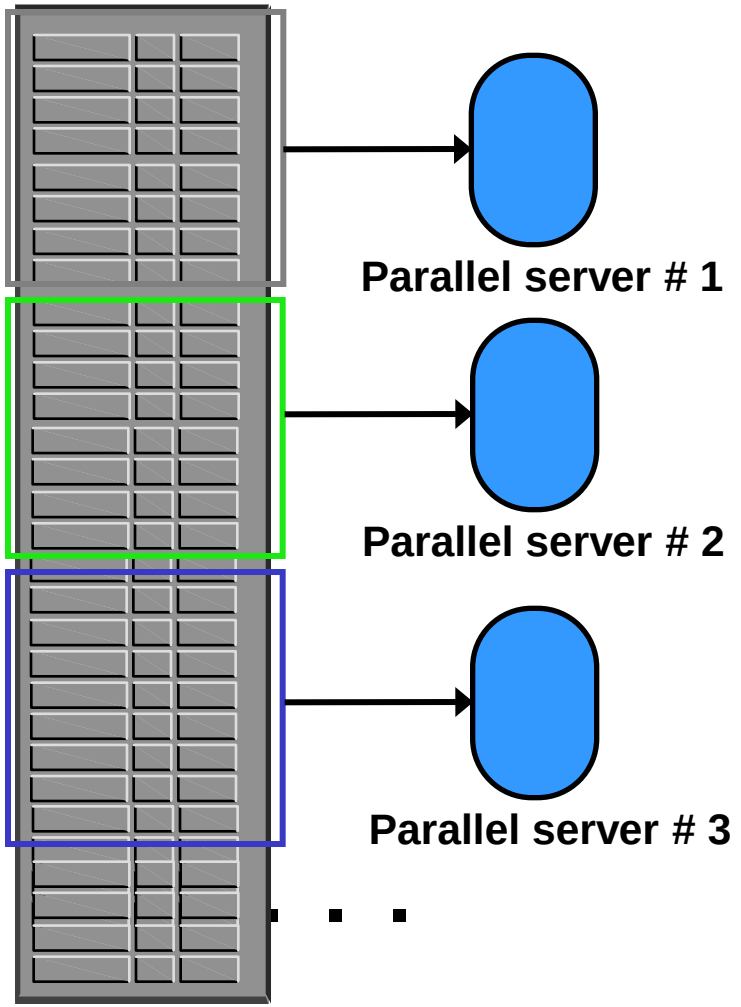
Consumers

Query Coordinator

	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT				311 (100)	
1	PX COORDINATOR					
2	PX SEND QC (RANDOM)	:TQ10002	1049K	31M	311 (2)	00:00:04
* 3	HASH JOIN BUFFERED		1049K	31M	311 (2)	00:00:04
4	PX RECEIVE		55500	704K	112 (0)	00:00:02
5	PX SEND HASH	:TQ10000	55500	704K	112 (0)	00:00:02
6	PX BLOCK ITERATOR		55500	704K	112 (0)	00:00:02
* 7	TABLE ACCESS FULL	CUSTOMERS	55500	704K	112 (0)	00:00:02
8	PX RECEIVE		1049K	18M	196 (2)	00:00:03
9	PX SEND HASH	:TQ10001	1049K	18M	196 (2)	00:00:03
10	PX BLOCK ITERATOR		1049K	18M	196 (2)	00:00:03
* 11	TABLE ACCESS FULL	SALES	1049K	18M	196 (2)	00:00:03

Producers

Oracle Parallel Query: Scanning a Table



- Data is divided into Granules
  - Block range or partition
- Each Parallel Server is assigned one or more Granules
- No two Parallel Servers ever contend for the same Granule
- Granules are assigned so that the load is balanced across all Parallel Servers
- Dynamic Granules chosen by the optimizer
  - Granule decision is visible in execution plan

Best Practices for using Parallel Execution

Current Issues

- Difficult to determine ideal DOP for each table without manual tuning
- One DOP does not fit all queries touching an object
- Not enough PX server processes can result in statement running serial
- Too many PX server processes can thrash the system
- Only uses IO resources

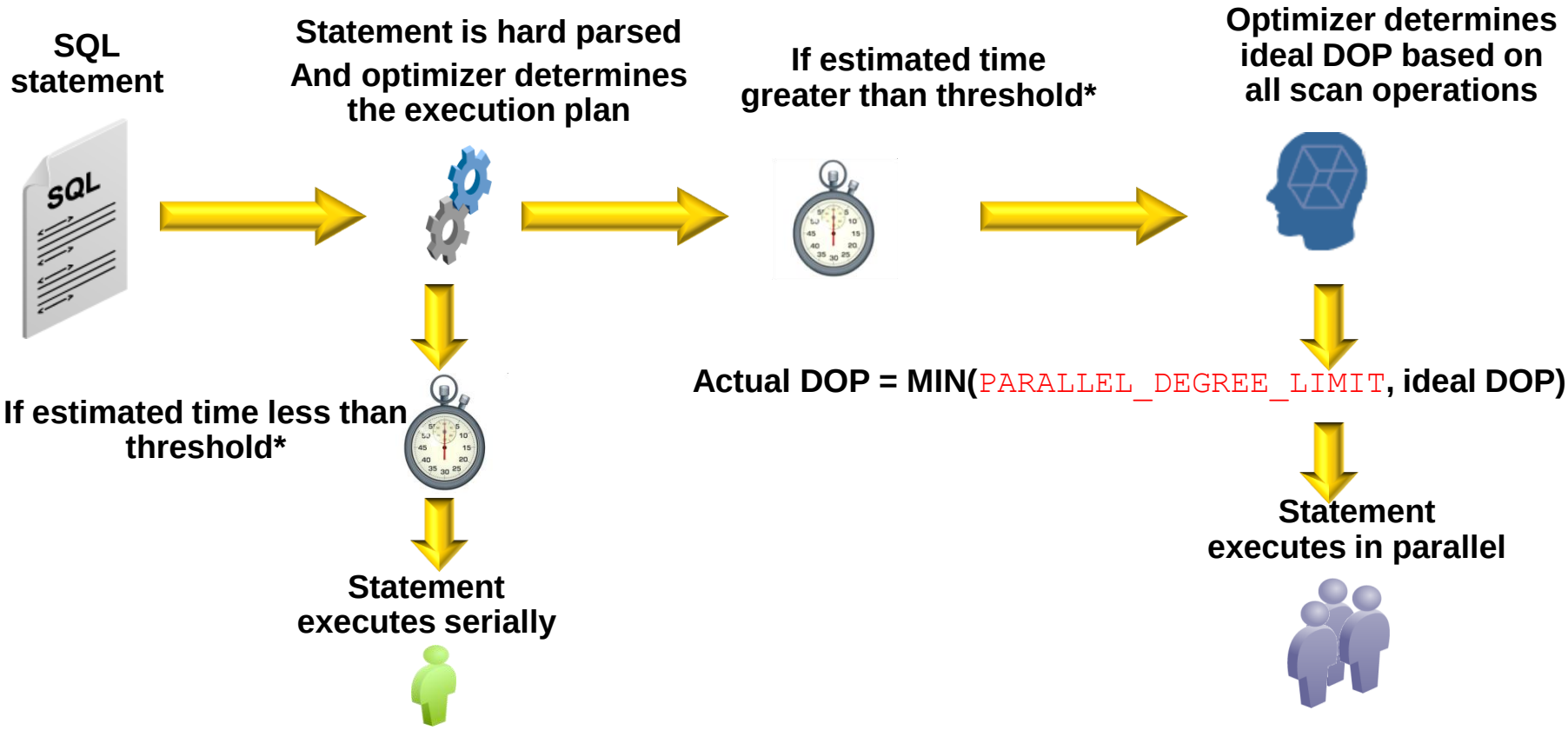
Solution

- Oracle automatically decides if a statement
  - Executes in parallel or not and what DOP it will use
  - Can execute immediately or will be queued
  - Will take advantage of aggregated cluster memory or not

Auto Degree of Parallelism

Enhancement addressing:

- Difficult to determine ideal DOP for each table without manual tuning
- One DOP does not fit all queries touching an object

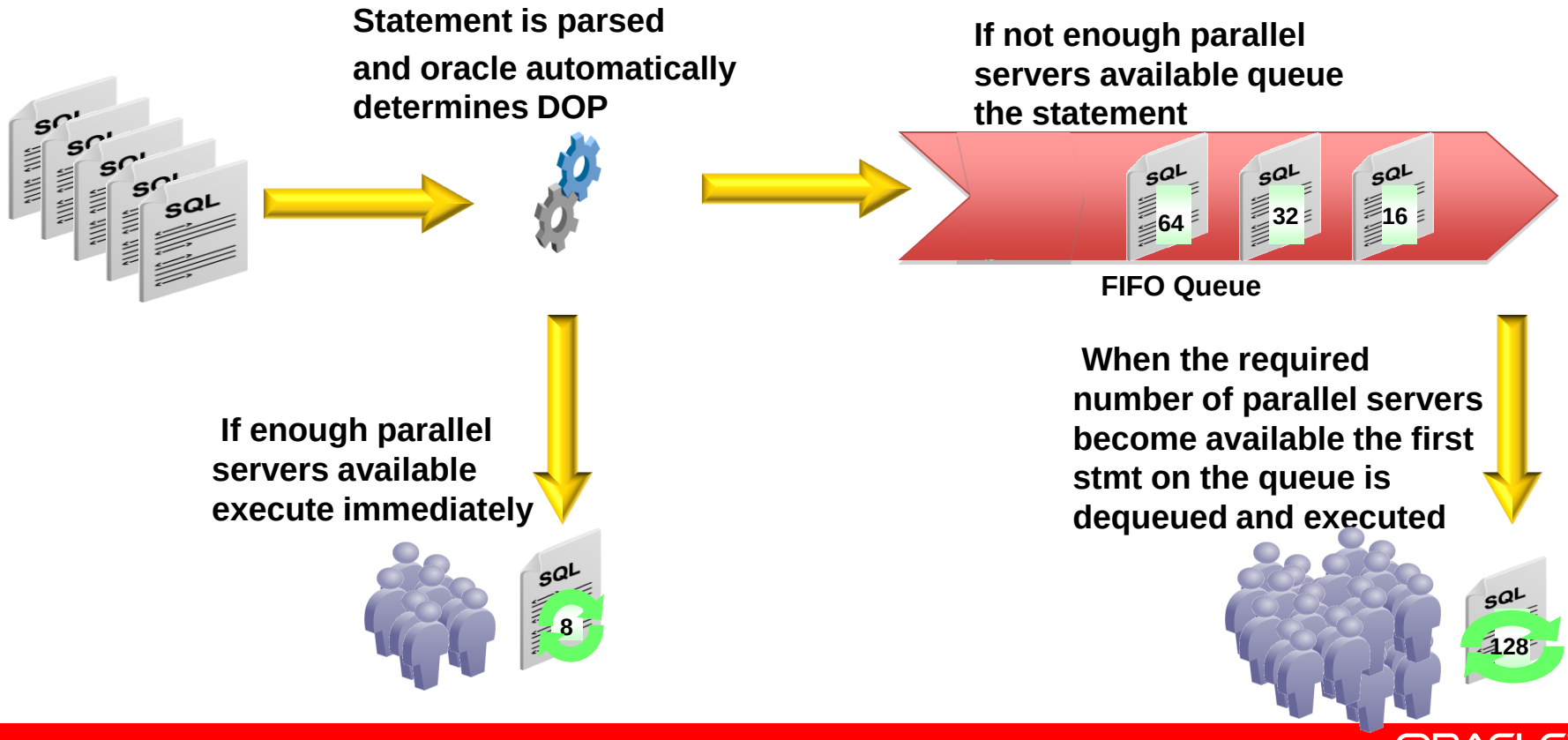


* Threshold set in parallel_min_time_threshold (default = 10s)

Parallel Statement Queuing

Enhancement addressing:

- Not enough PX server processes can result in statement running serial
- Too many PX server processes can thrash the system



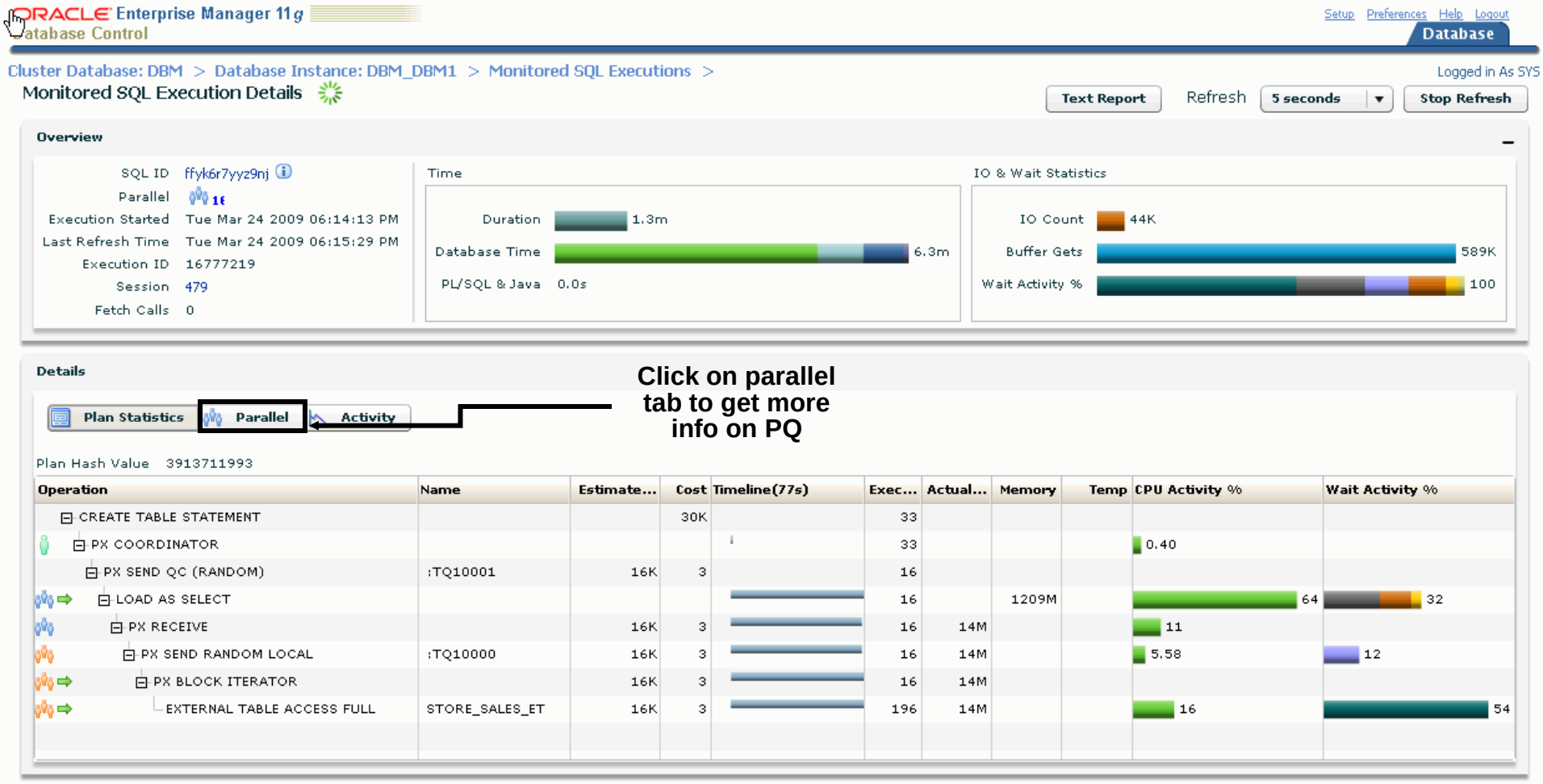
NOTE: Parallel_Servers_Target new parameter controls number of active PX processes before statement queuing kicks in

Identifying Granules of Parallelism during Scans in the Plan

```
 Id  | Operation                | Name     | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |    TQ  |IN-OUT| PQ Distrib
-----------------------------------------------------------------------------------------------------------------------
   0 | SELECT STATEMENT         |          |    17 |   153 |   565 (100)| 00:00:07 |       |       |        |      |
   1 |  PX COORDINATOR          |          |       |       |            |          |       |       |        |      |
   2 |   PX SEND QC (RANDOM)    | :TQ10001 |    17 |   153 |   565 (100)| 00:00:07 |       |       |  Q1,01 | P->S | QC (RAND)
   3 |    HASH GROUP BY         |          |    17 |   153 |   565 (100)| 00:00:07 |       |       |  Q1,01 | PCWP |
   4 |     PX RECEIVE           |          |    17 |   153 |   565 (100)| 00:00:07 |       |       |  Q1,01 | PCWP |
   5 |      PX SEND HASH        | :TQ10000 |    17 |   153 |   565 (100)| 00:00:07 |       |       |  Q1,00 | P->P | HASH
   6 |       HASH GROUP BY      |          |    17 |   153 |   565 (100)| 00:00:07 |       |       |  Q1,00 | PCWP |
   7 |        PX BLOCK ITERATOR |          |   10M |    85M|    60  (97)| 00:00:01 |     1 |    16 |  Q1,00 | PCWC |
*  8 |         TABLE ACCESS FULL| SALES    |   10M |    85M|    60  (97)| 00:00:01 |     1 |    16 |  Q1,00 | PCWP |
-----------------------------------------------------------------------------------------------------------------------

redicate Information (identified by operation id):
---------------------------------------------------

  8 - filter("CUST_ID"<=22810 AND "CUST_ID">=22300)
```

```
 Id  | Operation                          | Name       | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |    TQ  |IN-OUT| PQ Distrib
-----------------------------------------------------------------------------------------------------------------------------------
   0 | SELECT STATEMENT                   |            |    17 |   153 |     2  (50)| 00:00:01 |       |       |        |      |
   1 |  PX COORDINATOR                    |            |       |       |            |          |       |       |        |      |
   2 |   PX SEND QC (RANDOM)              | :TQ10001   |    17 |   153 |     2  (50)| 00:00:01 |       |       |  Q1,01 | P->S | QC (RAND)
   3 |    HASH GROUP BY                   |            |    17 |   153 |     2  (50)| 00:00:01 |       |       |  Q1,01 | PCWP |
   4 |     PX RECEIVE                     |            |    26 |   234 |     1   (0)| 00:00:01 |       |       |  Q1,01 | PCWP |
   5 |      PX SEND HASH                  | :TQ10000   |    26 |   234 |     1   (0)| 00:00:01 |       |       |  Q1,00 | P->P | HASH
   6 |       PX PARTITION RANGE ALL       |            |    26 |   234 |     1   (0)| 00:00:01 |     1 |    16 |  Q1,00 | PCWC |
   7 |        TABLE ACCESS BY LOCAL INDEX ROWID| SALES      |    26 |   234 |     1   (0)| 00:00:01 |     1 |    16 |  Q1,00 | PCWP |
*  8 |         INDEX RANGE SCAN           | SALES_CUST |    26 |       |     0   (0)| 00:00:01 |     1 |    16 |  Q1,00 | PCWP |
-----------------------------------------------------------------------------------------------------------------------------------

redicate Information (identified by operation id):
---------------------------------------------------

  8 - access("CUST_ID">=22300 AND "CUST_ID"<=22810)
```

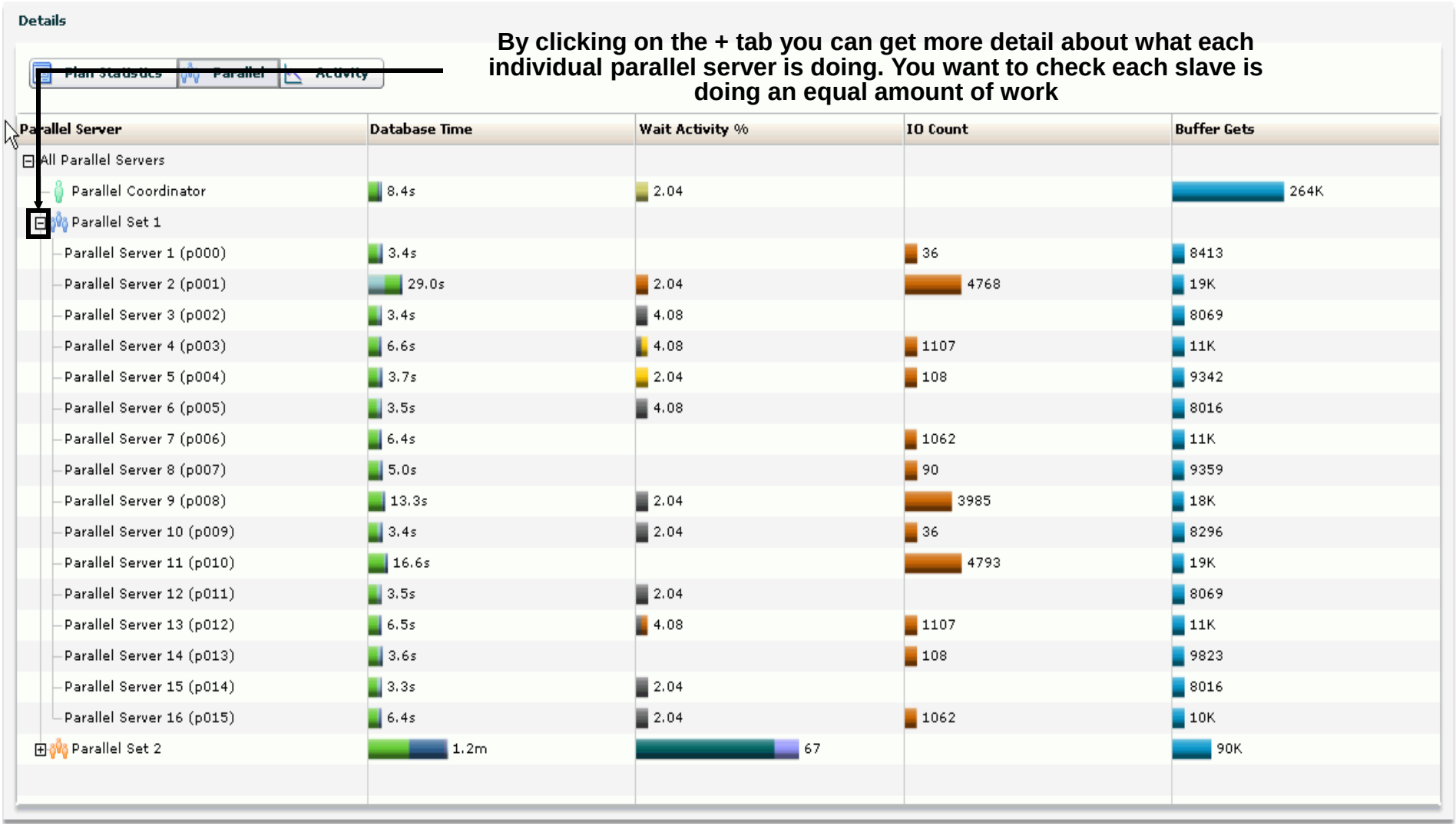
SQL Monitoring screens



Click on parallel tab to get more info on PQ

The green arrow indicates which line in the execution plan is currently being worked on

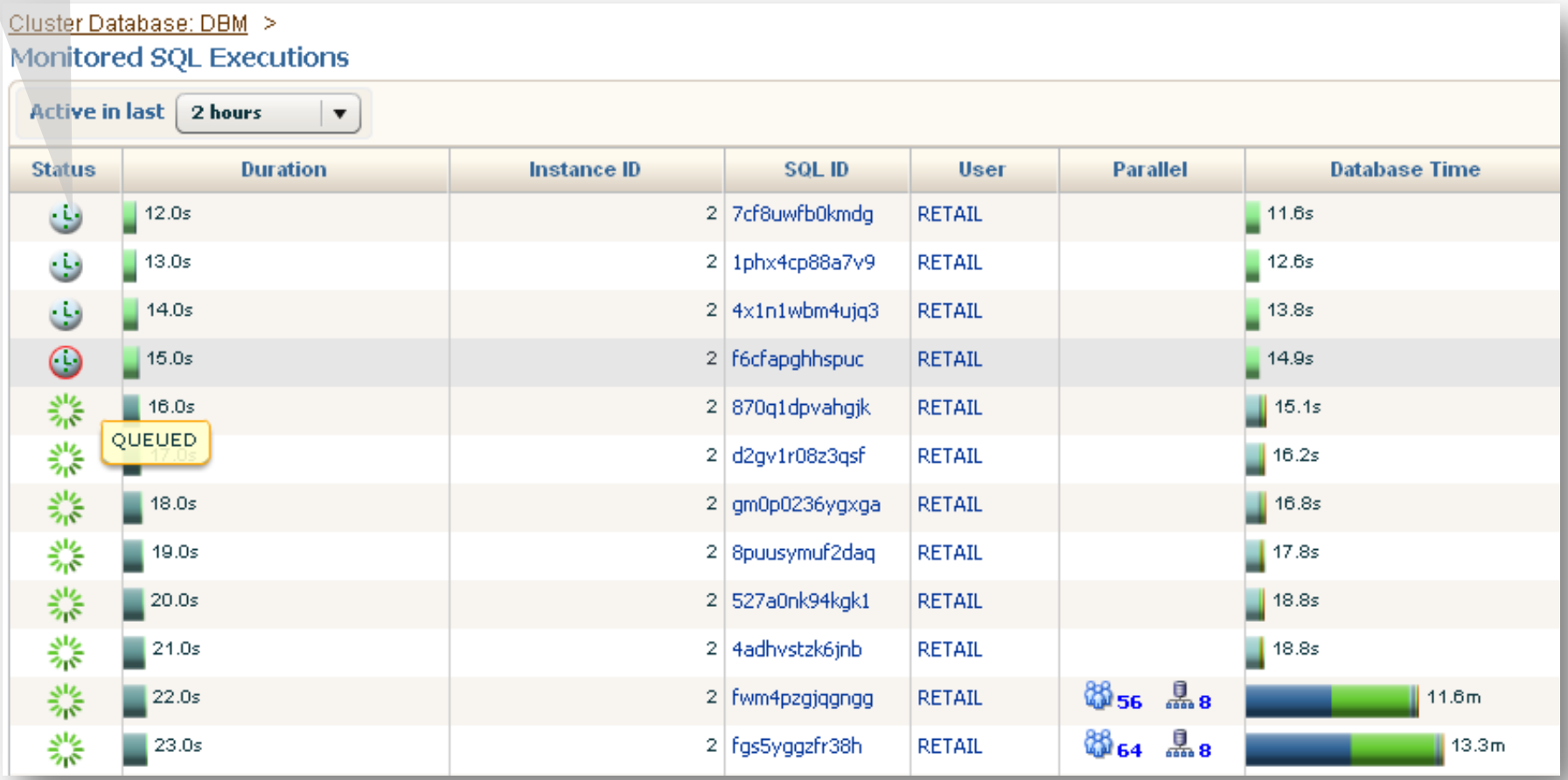
SQL Monitoring Screens



Parallel Server	Database Time	Wait Activity %	IO Count	Buffer Gets
All Parallel Servers				
Parallel Coordinator	8.4s	2.04		264K
Parallel Set 1				
Parallel Server 1 (p000)	3.4s		36	8413
Parallel Server 2 (p001)	29.0s	2.04	4768	19K
Parallel Server 3 (p002)	3.4s	4.08		8069
Parallel Server 4 (p003)	6.6s	4.08	1107	11K
Parallel Server 5 (p004)	3.7s	2.04	108	9342
Parallel Server 6 (p005)	3.5s	4.08		8016
Parallel Server 7 (p006)	6.4s		1062	11K
Parallel Server 8 (p007)	5.0s		90	9359
Parallel Server 9 (p008)	13.3s	2.04	3985	18K
Parallel Server 10 (p009)	3.4s	2.04	36	8296
Parallel Server 11 (p010)	16.6s		4793	19K
Parallel Server 12 (p011)	3.5s	2.04		8069
Parallel Server 13 (p012)	6.5s	4.08	1107	11K
Parallel Server 14 (p013)	3.6s		108	9823
Parallel Server 15 (p014)	3.3s	2.04		8016
Parallel Server 16 (p015)	6.4s	2.04	1062	10K
Parallel Set 2	1.2m	67		90K

Simple Example of Queuing

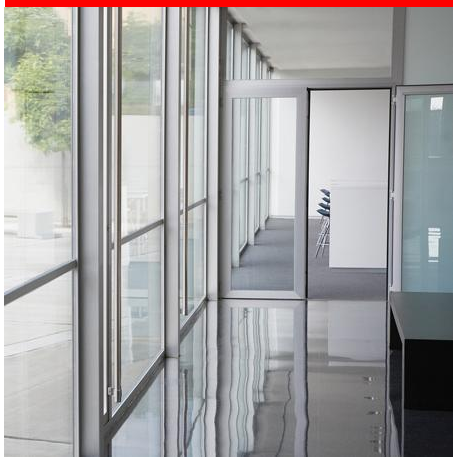
Queued stmts are indicated by the clock



Status	Duration	Instance ID	SQL ID	User	Parallel	Database Time
	12.0s	2	7cf8uwfb0kmdg	RETAIL		11.6s
	13.0s	2	1phx4cp88a7v9	RETAIL		12.6s
	14.0s	2	4x1n1wbm4ujq3	RETAIL		13.8s
	15.0s	2	f6cfapghhspuc	RETAIL		14.9s
	16.0s	2	870q1dpvahgjk	RETAIL		15.1s
	17.0s	2	d2gv1r08z3qsf	RETAIL		16.2s
	18.0s	2	gm0p0236ygxga	RETAIL		16.8s
	19.0s	2	8puusymuf2daq	RETAIL		17.8s
	20.0s	2	527a0nk94kgk1	RETAIL		18.8s
	21.0s	2	4adhvstzk6jnb	RETAIL		18.8s
	22.0s	2	fwm4pzgjqgngg	RETAIL	56 8	11.6m
	23.0s	2	fgs5yggzfr38h	RETAIL	64 8	13.3m

8 Statements run before queuing kicks in

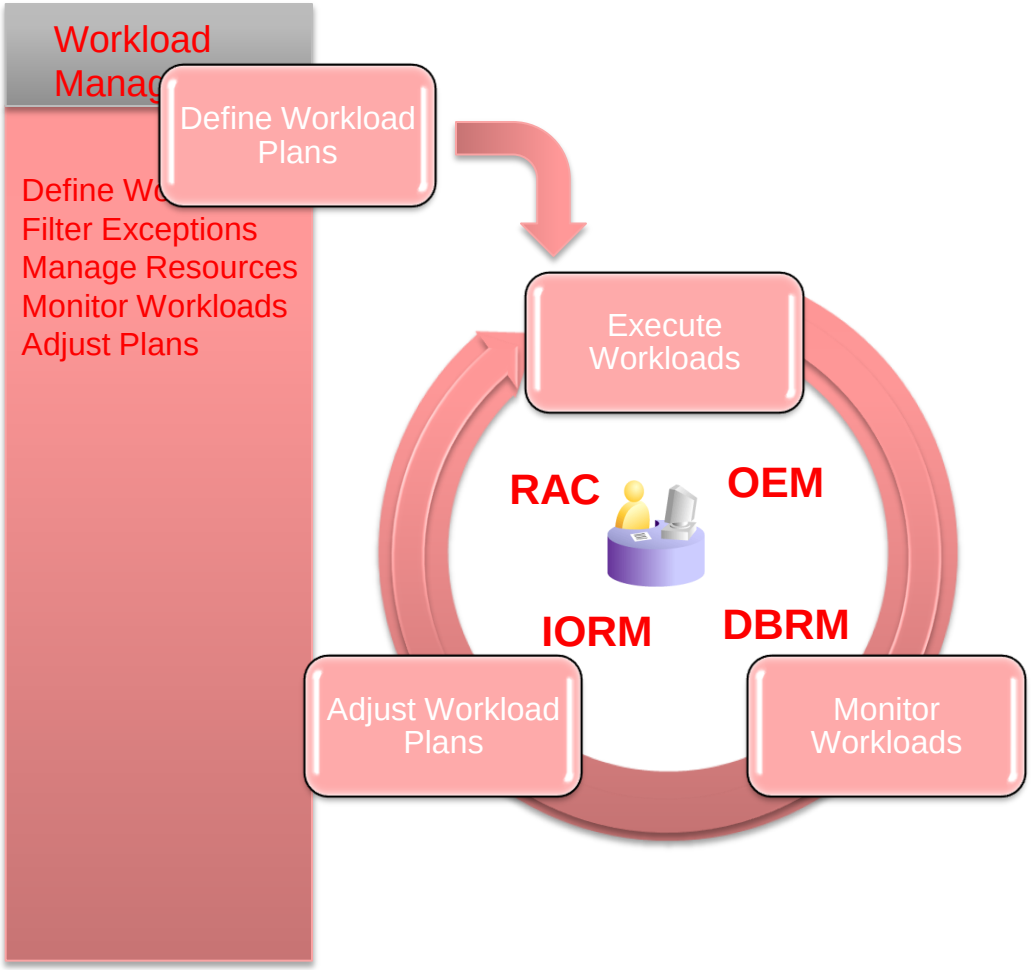
Agenda



- Data Warehousing Reference Architecture
- The three Ps of Data Warehousing
  - Power
  - Partitioning
  - Parallel
- Workload Management on a Data Warehouse

Workload Management for DW

Setting Up a Workload Management System



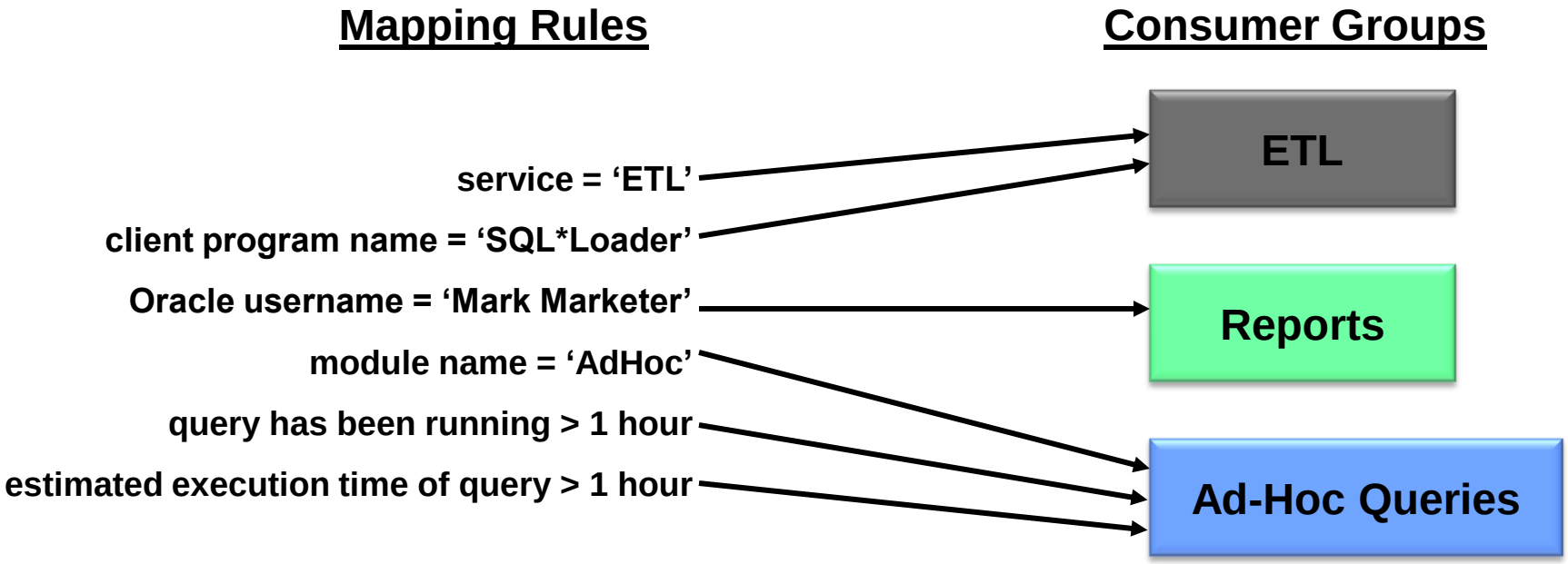
Workload Management for DW
Oracle Database Resource Manager

- Traditionally you were told to use Resource Management if data warehouse is CPU bound
  - Protects critical tasks from interference from non-critical tasks
  - Allows CPU to be utilized according to a specific ratio
  - Prevents thrashing and system instability that can occur with excessive CPU loads
- But did you know Resource Manager can
  - Control DOP for each group of users
  - Control the number of concurrent queries for each group
  - Prevent runaway queries

Setting up Resource Manager

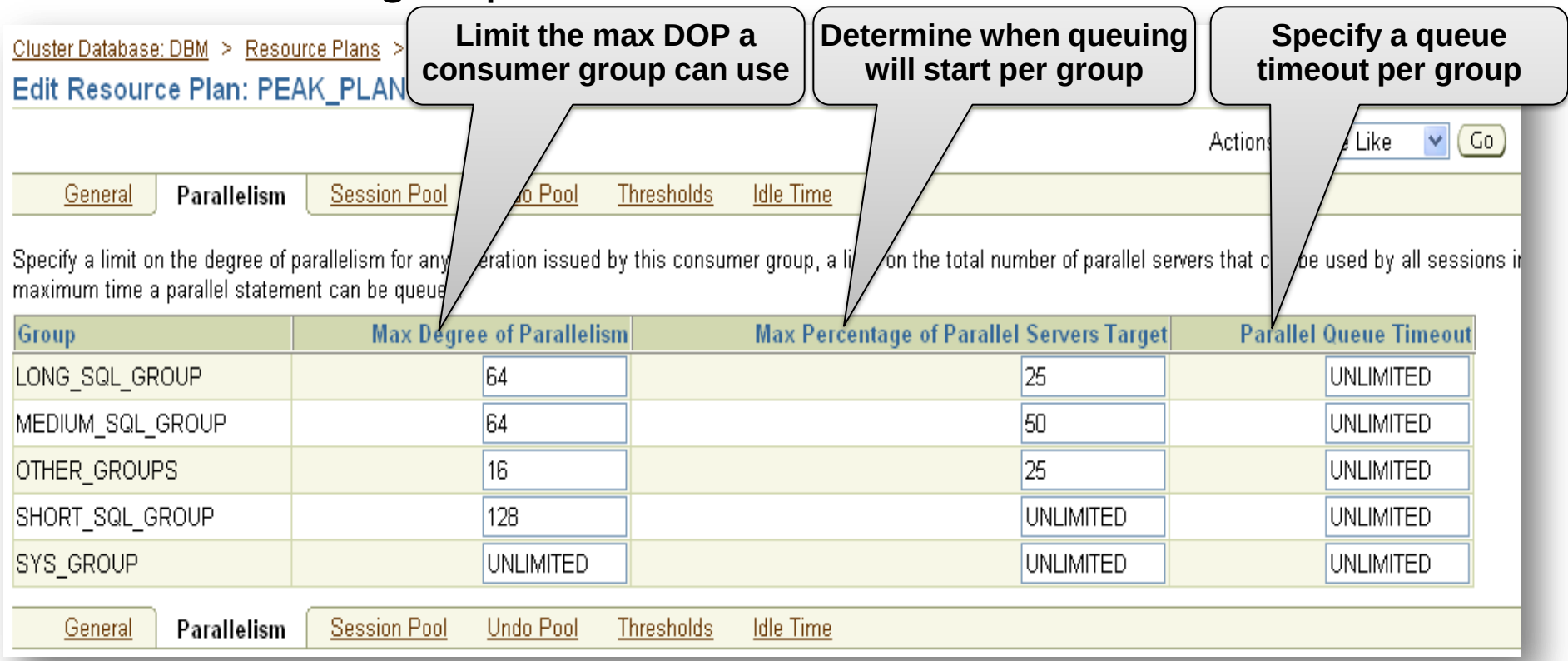
Step 1 Creating Consumer Groups

- Create Consumer Groups for each type of workload, e.g.
  - ETL consumer group
  - Reports consumer group
  - Ad-Hoc Queries consumer group
- Create rules to dynamically map sessions to consumer groups, based on session attributes



Setting up Resource Manager
Step 3 Configure Parallel Execution

- Manage parallel execution resources and priorities by consumer groups

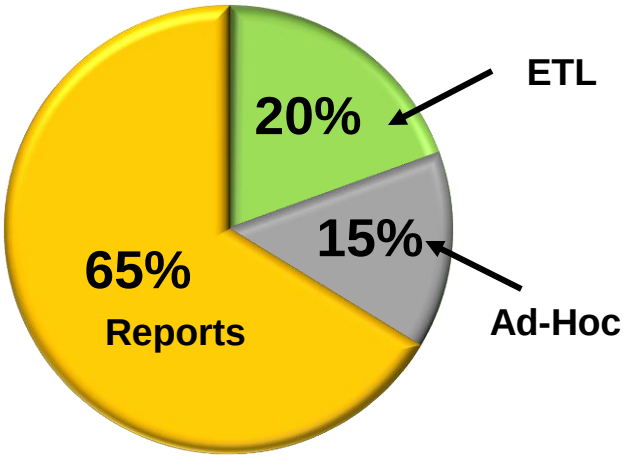


NOTE: Parallel_Servers_Target new parameter controls number of active PX processes before statement queuing kicks in

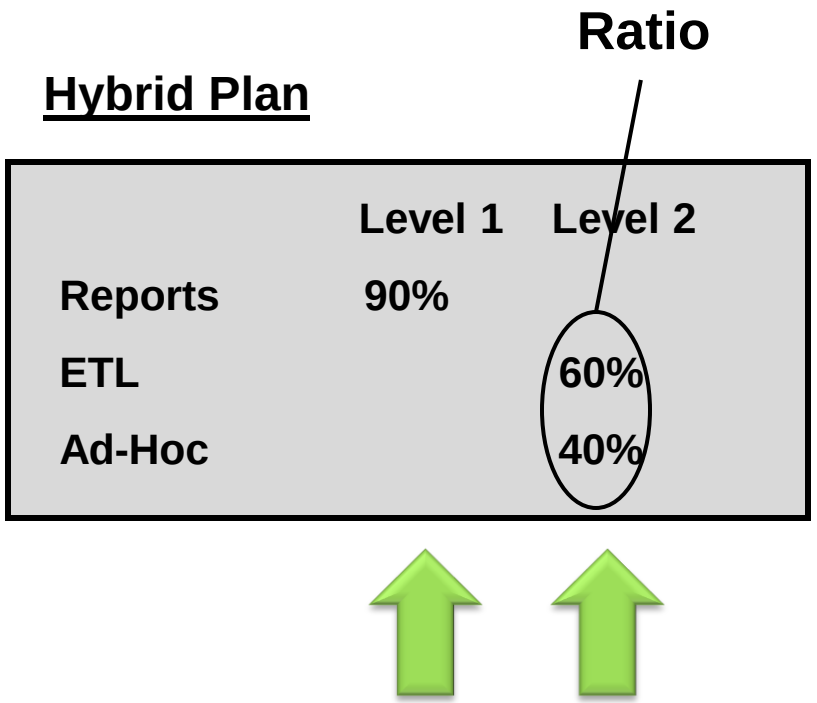
Setting up Resource Manager

Step 2 Creating Resource Plans

Ratio-Based Plan



Hybrid Plan



Priority-Based Plan

Priority 1: Reports
Priority 2: ETL
Priority 3: Ad-Hoc

Specify a priority for each group

Workload Monitoring

Resource Allocations

Group/Subplan	Max Utilization Limit	Percentage
OTHER_GROUPS		
RD_LIMIT_ACTIVE		40
RD_LIMIT_DOP		
RD_UNLIMITED		60

Directive Values

Group	Max Degree of Parallelism	Max Percentage of Parallel Servers Target	Parallel Queue Timeout	Max Number of Active Sessions	Activation Queue Timeout (sec)	Max Undo Space (KB)	Max Estimated Execution Time (sec)	Max Idle Time (sec)	Max Idle Time if Blocking Another Session (sec)
OTHER_GROUPS	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED
RD_LIMIT_ACTIVE	48	30	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED
RD_LIMIT_DOP	8	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED
RD_UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED

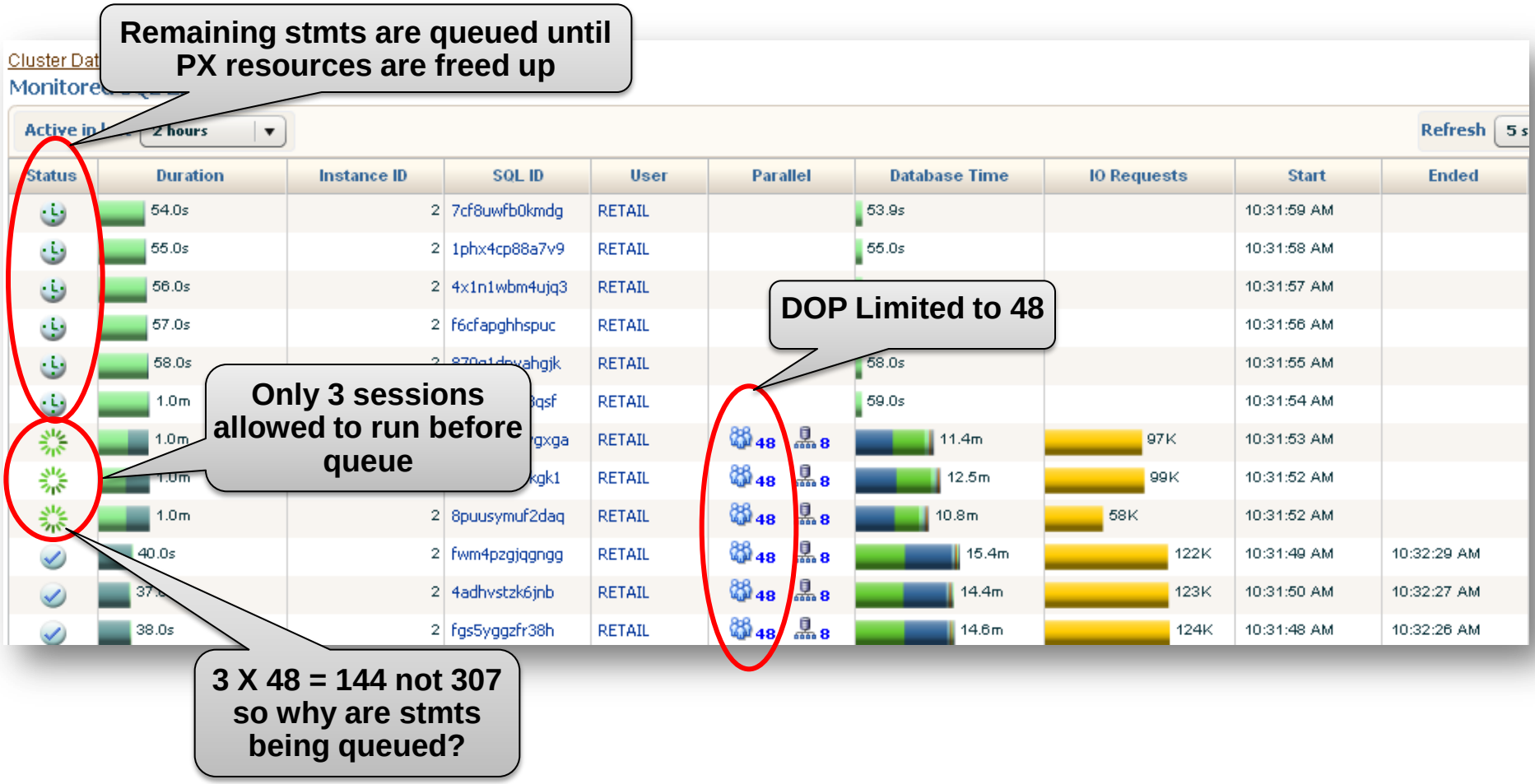
Thresholds

Group	Execution Time Limit (Sec)	I/O Limit (MB)	I/O Request Limit (Requests)	Action	Revert after call?	Use estimate?
OTHER_GROUPS	UNLIMITED	UNLIMITED	UNLIMITED		☐	☐
RD_LIMIT_ACTIVE	UNLIMITED	UNLIMITED	UNLIMITED		☐	☐
RD_LIMIT_DOP	UNLIMITED	UNLIMITED	UNLIMITED		☐	☐
RD_UNLIMITED	UNLIMITED	UNLIMITED	UNLIMITED		☐	☐

Looking at user Retail who is a member of RD_LIMIT_ACTIVE group They have a max DOP of 48, 30% of parallel_server_target & no timeout specified

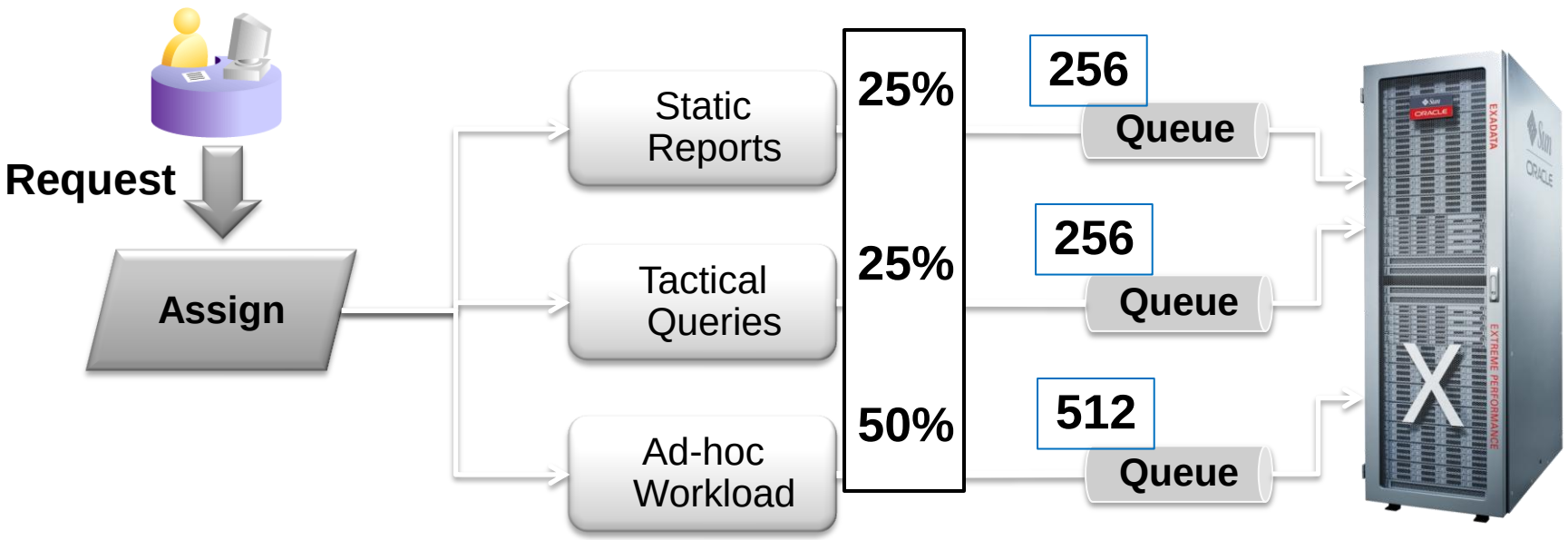
parallel_server_target =1024 => 30% = 307 allowed before queuing

Workload Monitoring



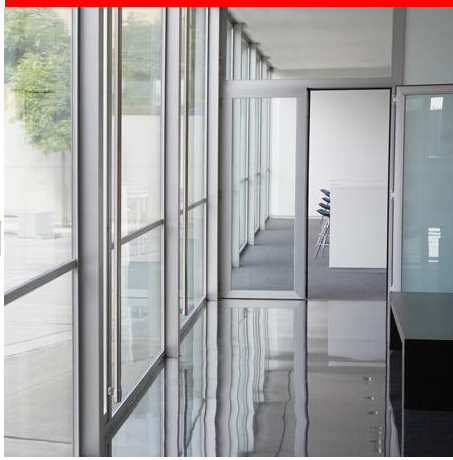
Statements require two sets of PX process for producers & consumers

Resource Manager - Statement Queuing



- Queuing is embedded with DBRM
- One queue per consumer group

Summary



- Implement the three Ps of Data Warehousing
  - Power – balanced hardware configuration
    - Make sure the system can deliver your SLA
  - Partitioning – Performance, Manageability, ILM
    - Make sure partition pruning and partition-wise joins occur
  - Parallel – Maximize the number of processes working
    - Make sure the system is not flooded using DOP limits & queuing
- Workload Management on a Data Warehouse
  - Use Database Resource Manager
    - Control maximum DOP each user can have
    - Control when statements should begin queue
    - Control what happens to “run away” queries

Q & A

The preceding is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.
The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.