ORACLE®

Performance Tuning in Oracle Spatial

(Its only a database!)

Siva and Tim

Agenda

- Tuning Methodology
- Top Ten Mistakes
- Database Options
  - Partitioning
  - Real Application Clusters
- Optimising Data Access

Agenda

- Tuning Methodology
- Top Ten Mistakes
- Database Options
  - Partitioning
  - Real Application Clusters
- Optimising Data Access

Tuning Methodology

- Performance Tuning is **NOT** and afterthought
- Application performance should be part of the design specification
  - Understand application design issues properly
  - Understand the way Oracle works
  - Understand good SQL coding practice
- Set SLA goals – make them reasonable
- Instrument Code
- Measure Baseline Performance Statistics
- Understand the Application Useage Patterns

Tuning Methodology

- The DBA may not understand the Data but they will understand the Database
- An Oracle Spatial Database works in exactly the same way as a “normal” database
- Utilise standard tools and techniques (ADDM and AWR)

Agenda

- Tuning Methodology
- Top Ten Mistakes
- Database Options
  - Partitioning
  - Real Application Clusters
- Optimising Data Access

Top Ten Mistakes

- Poor Connection Management
  - Common in 3 Tier Apps
- Poor Use of Cursors and the Shared Pool
  - Bind Variables – care with dynamic SQL
- Non Standard Initialisation Parameters
  - Incorrectly set, bad advice
- Deployment/Migration Errors
  - Missing object, incomplete stats, testing against poor data sets,
- Unreasonable Expectations
  - Unsuitable hardware, poor user expectation management

Top Ten Mistakes

- Excessive Full Tabl Scans
  - Poor application design, missing Indexes
- Excessive Recursive SQL
  - May be poor space management
- Bad SQL
  - Use ADDM to identify igh resource statements
- Poor I/O Design
- Redo Log Issues
  - Too small, too few

Agenda

- Tuning Methodology
- Top Ten Mistakes
- Database Options
  - Partitioning
  - Real Application Clusters
- Optimising Data Access

Oracle Partitioning

- Logical table decomposed to more than one physical table
- Transparent to applications
- Based on a partitioning key
  - Single Column
  - Multicolumn
- Partitioned index is one logical index
  - Decomposed to one physical index per partition

Creating a Partitioned Table

- Use a regular **`CREATE TABLE`** statement, adding a **`PARTITIONING`** clause:

```
CREATE TABLE partition_table
  in_date DATE, geom SDO_GEOMETRY)
PARTITION BY RANGE (in_date)
  (PARTITION Q1_2004 VALUES LESS THAN ('01-APR-2004'),
   PARTITION Q2_2004 VALUES LESS THAN ('01-JUL-2004'),
   PARTITION Q3_2004 VALUES LESS THAN ('01-OCT-2004'),
   PARTITION Q4_2004 VALUES LESS THAN ('01-JAN-2005')
  );
```

Row Placement in Partitions

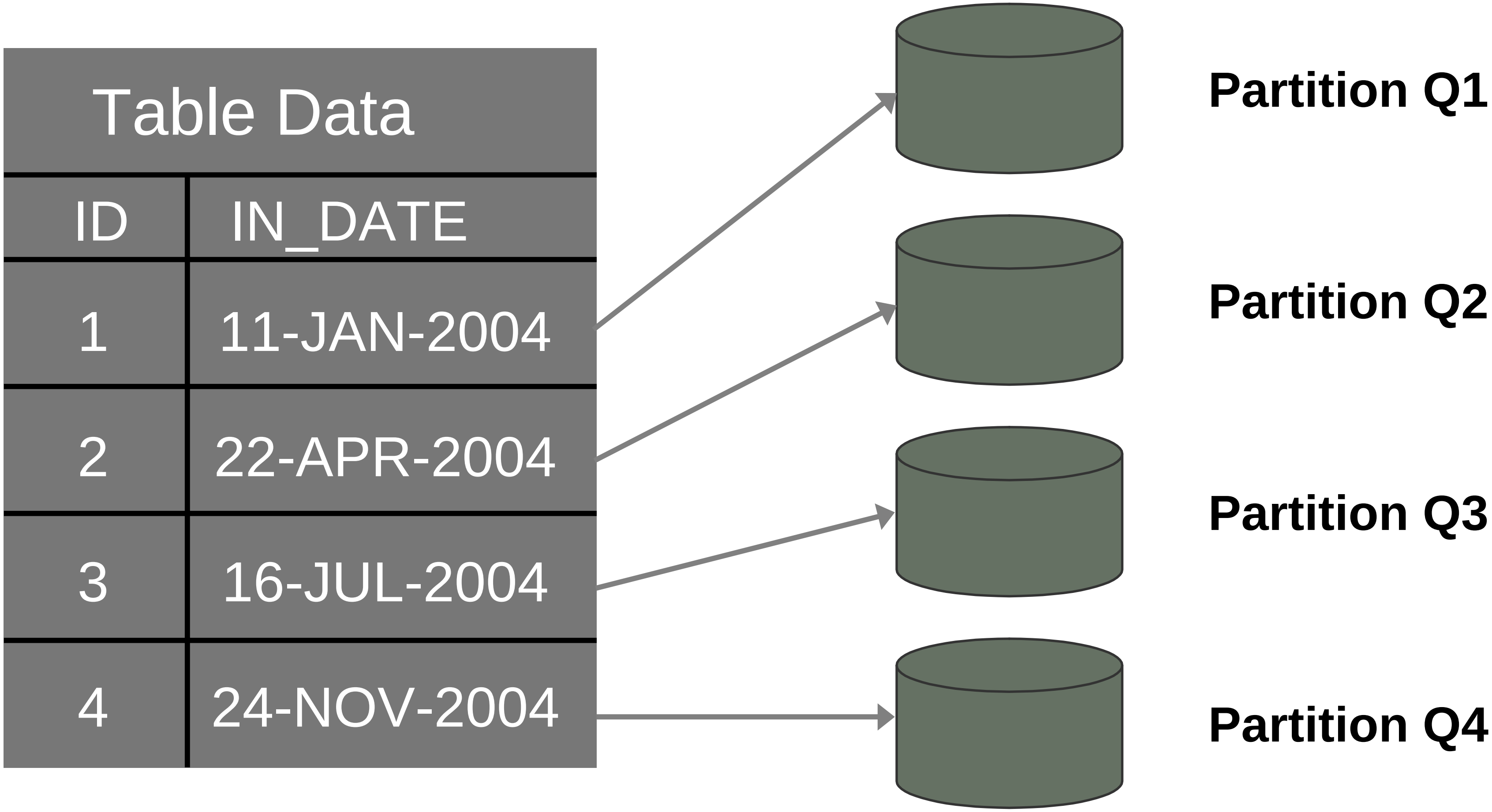


Table Data

ID	IN_DATE
1	11-JAN-2004
2	22-APR-2004
3	16-JUL-2004
4	24-NOV-2004

Partition Q1

Partition Q2

Partition Q3

Partition Q4

Why Partition Data?

- Maintainability
  - Partition-at-a-time load, backup, and restore
  - Load data, create and rebuild indexes in different tablespaces on a partition-by-partition basis
  - Exchange tables and their indexes with partitions and their indexes
- Performance and Scalability
  - Search multiple partitions in parallel
  - Spread I/O load for accessing partitions across multiple I/O channels/controllers
  - Store data physically close together that is likely to be accessed together
  - Utilize partition elimination through the Oracle optimizer

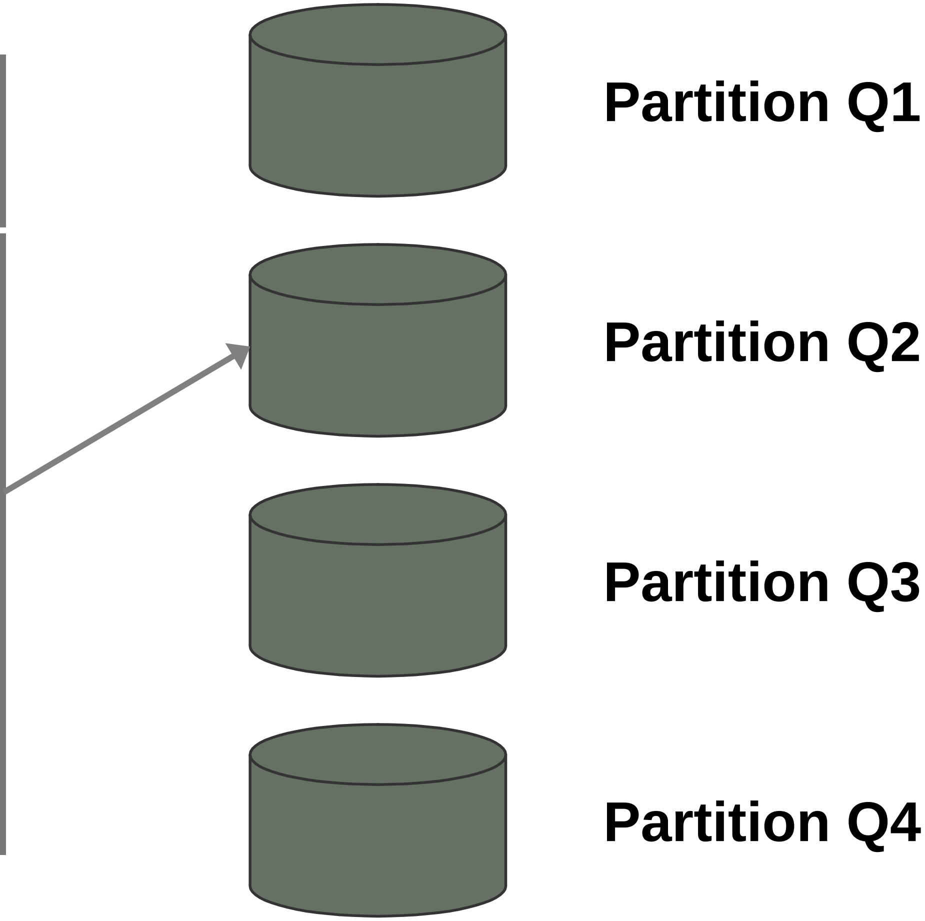
Partition Elimination

- Including the partition key in the **`WHERE`** clause allows the Oracle optimizer to eliminate from consideration all partitions that do not meet the partition criteria
  - Completely automatic
  - Reduces processing requirements
  - Reduces I/O requirements
- If the partition key is not included in the where clause all partitions are searched
  - Can be serial or parallel

Partition Elimination

Return all data from month of April

```
SELECT *

FROM partition_table

WHERE

 IN_DATE > '31-MAR-2004'

AND

 IN_DATE < '01-MAY-2004'
```



Partition Q1

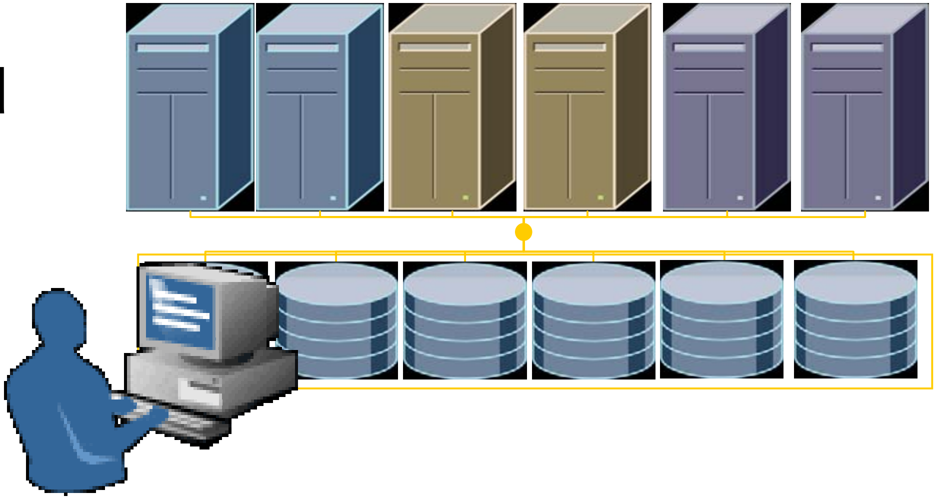
Partition Q2

Partition Q3

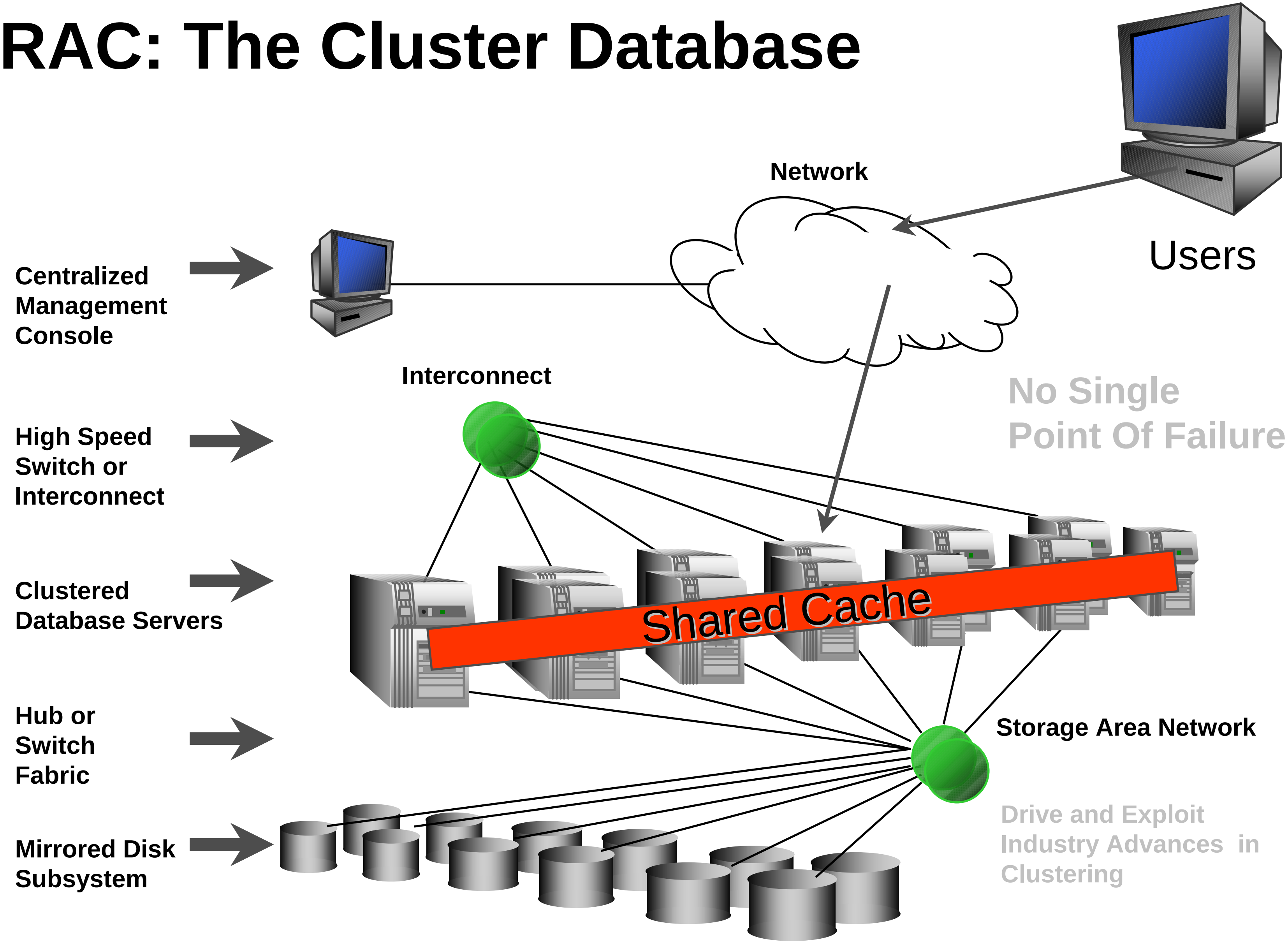
Partition Q4

Real Application Clusters

- Run All Your Applications
- Add Low Cost Capacity on Demand
- Scalable
- Highly Available
- Manageable
- Secure



RAC: The Cluster Database



Network

Users

Centralized Management Console

Interconnect

No Single Point Of Failure

High Speed Switch or Interconnect

Clustered Database Servers

Shared Cache

Hub or Switch Fabric

Storage Area Network

Drive and Exploit Industry Advances in Clustering

Mirrored Disk Subsystem

Agenda

- Tuning Methodology
- Top Ten Mistakes
- Database Options
  - Partitioning
  - Real Application Clusters
- Optimising Data Access

Spatial Partitioning
and
Spatial Partition Pruning

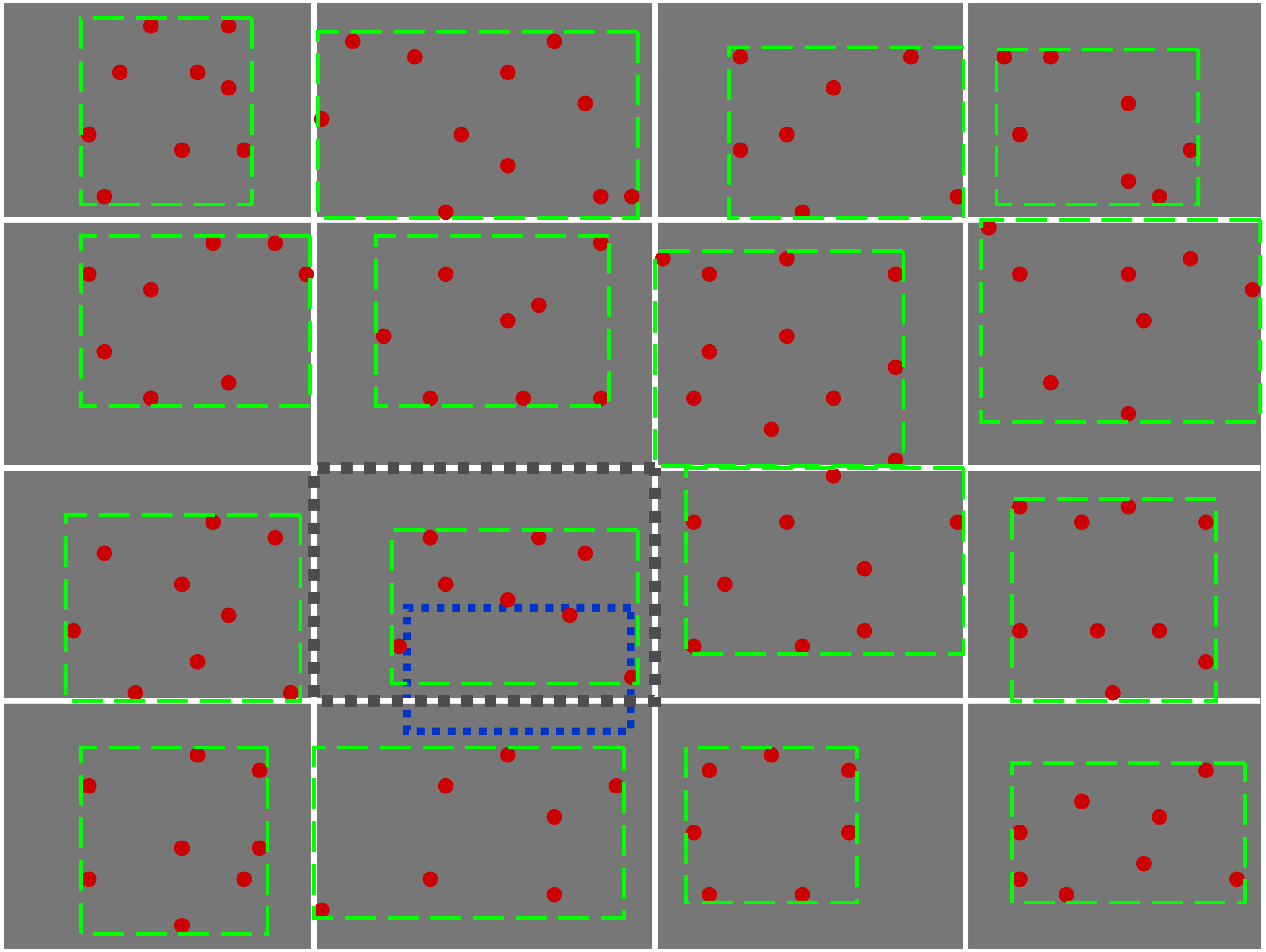
Spatial Partitioning

- Spatial data can be partitioned based on its location
  - Can partition on X (Longitude) or Y (Latitude)
  - Can partition using both X and Y
  - Can partition using a “traditional” partition key and location partitioning
- Oracle Spatial can use index information to do “Spatial Partition Pruning”
  - **New and unique to Oracle Spatial**

Spatial Partition Pruning

- What is Spatial partition pruning?
  - Similar to Oracle optimizer partition elimination
  - Included in Spatial index metadata is **SDO_ROOT_MBR**
    - The minimum bounding rectangle encompassing the spatial data in that indexed partition
  - At query time, Oracle Spatial compares the query window minimum bounding rectangle (MBR) with the partition's **SDO_ROOT_MBR**
  - If there is no overlap, then no further processing is done on that partition

Spatial Partition Pruning



Spatial Partition Pruning

- Used with Spatial operators:
  - **`SDO_WITHIN_DISTANCE`**, **`SDO_FILTER`**, and **`SDO_RELATE`**
- Completely transparent and automatic
- Requires no partition key
- Occurs at run time

Single Column Spatial Partitioning

Creating a Spatially Partitioned Table: Single Column Key

- Can partition on X (longitude) or Y (latitude)
  - Numeric column added to table to hold X or Y value

```
CREATE TABLE new_partition_table ( in_date DATE,
  geom SDO_GEOMETRY, x_value NUMBER)
PARTITION BY RANGE (x_value)
  (PARTITION P_LT_90W VALUES LESS THAN (-90),
   PARTITION P_LT_0 VALUES LESS THAN (0),
   PARTITION P_LT_90E VALUES LESS THAN (90),
   PARTITION P_MAX VALUES LESS THAN (180)
  );
```

Creating a Spatially Partitioned Table: Single Column Key

- Load X or Y values based on input geometry
  - Directly load from **`SDO_POINT`** field
  - Use **`SDO_GEOM.SDO_CENTROID`** function to load from a point that is the centroid of a polygon
    - Only valid for polygons
  - Use **`SDO_GEOM.SDO_POINTONSURFACE`** to load from point, line string, or polygon

Loading a Spatially Partitioned Table: Single Column Key

- Example:

```
INSERT INTO new_partition_table NOLOGGING
SELECT val.in_date,
       val.geom,
       val.pt.sdo_point.x
FROM (SELECT in_date,
             geom,
             SDO_GEOM.SDO_POINTONSURFACE(
                geom, 0.05) pt
      FROM partition_table) val;
```

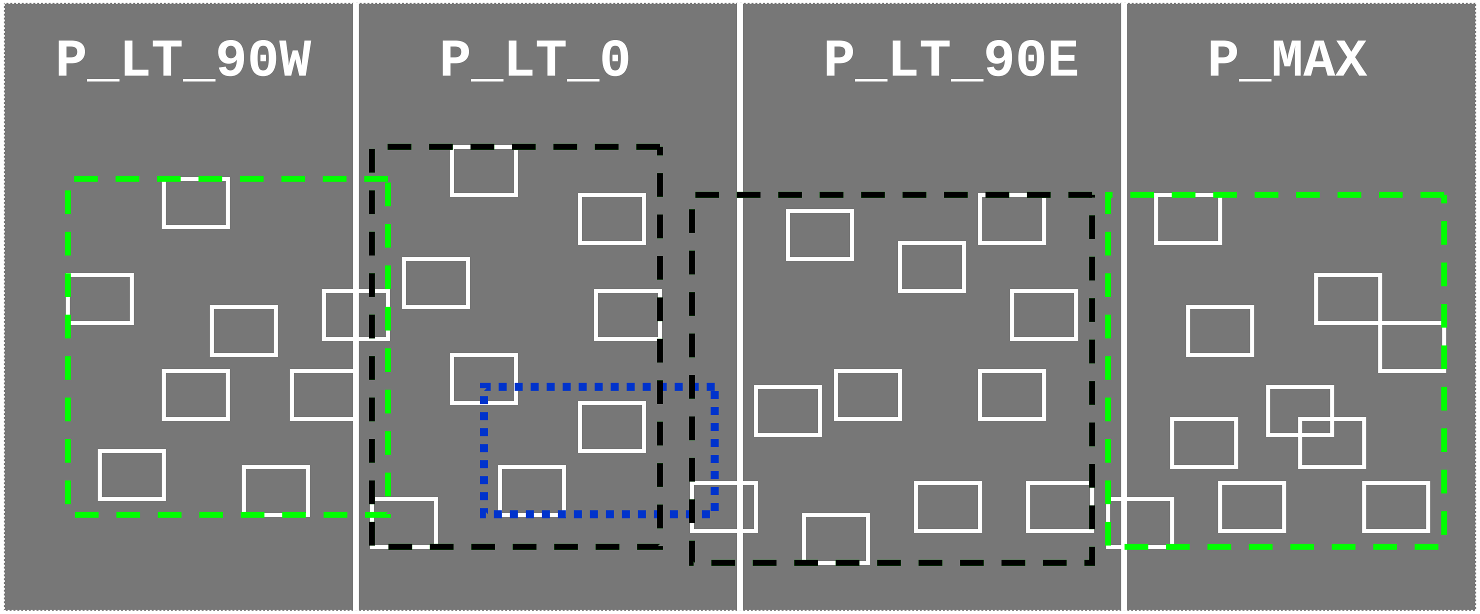
Loading a Spatially Partitioned Table: Single Column Key

- Based on the **`X_VALUE`**, data gets loaded as follows:

X_VALUE	Partition
-180 <= x < -90	P_LT_90W
-90 <= x < 0	P_LT_0
0 <= x < 90	P_LT_90E
90<= x <= 180	P_MAX

Spatially Partitioned Table and Index: Single Column Key

- Four partitions, based on the first point of each geometry
- Only partitioned by longitude



Spatially Partitioned Table and Index: Single Column Key

- Each partition’s spatial index stores the MBR around all of the geometries in that index
- Each index MBR may overlap other index MBRs
- The query window MBR determines which spatial partitioned indexes will be searched
- This partitioning scheme (in X) could be more effective by adding a Y value
- Do ***NOT*** use **`X_VALUE`** in the **`WHERE`** clause
  - Allowing the Oracle optimizer to eliminate partitions can lead to wrong results with non-point spatial data

Multicolumn
Spatial Partitioning

Creating a Spatially Partitioned Table: Multicolumn Key

- Can partition on X (longitude) and Y (latitude)
  - Numeric columns added to hold X and Y values
- Special rules for multicolumn range partitioned tables
  - The next key is only used for loading if there is an exact match on the previous key
    - All but the last key in a multicolumn partitioned table should be associated with a single value
  - Partition key is inclusive of key value

Creating a Spatially Partitioned Table: Multicolumn Key

```
CREATE TABLE multi_partition_table (
  in_date DATE,   geom    SDO_GEOMETRY,
  x_value NUMBER, y_value NUMBER)
PARTITION BY RANGE (X_VALUE, Y_VALUE)
  (
   PARTITION P_LT_90W_45S VALUES LESS THAN (1,-45),
   PARTITION P_LT_90W_0 VALUES LESS THAN   (1,0),
   PARTITION P_LT_90W_45N VALUES LESS THAN (1,45),
   PARTITION P_LT_90W_90N VALUES LESS THAN (1,MAXVALUE),
   PARTITION P_LT_0_45S VALUES LESS THAN   (2,-45),
   PARTITION P_LT_0_0 VALUES LESS THAN     (2,0),
   PARTITION P_LT_0_45N VALUES LESS THAN   (2,45),
   PARTITION P_LT_0_90N VALUES LESS THAN   (2,MAXVALUE),
```

Creating a Spatially Partitioned Table: Multicolumn Key

- Previous example continued

```
…
   PARTITION P_LT_90E_45S VALUES LESS THAN  (3,-45),
   PARTITION P_LT_90E_0 VALUES LESS THAN    (3,0),
   PARTITION P_LT_90E_45N VALUES LESS THAN  (3,45),
   PARTITION P_LT_90E_90N VALUES LESS THAN  (3,MAXVALUE),
   PARTITION P_LT_180E_45S VALUES LESS THAN (4,-45),
   PARTITION P_LT_180E_0 VALUES LESS THAN   (4,0),
   PARTITION P_LT_180E_45N VALUES LESS THAN (4,45),
   PARTITION P_LT_180E_90N VALUES LESS THAN (4,MAXVALUE)
  );
```

Loading a Spatially Partitioned Table: Multicolumn Key

- Example:

```
INSERT INTO multi_partition_table NOLOGGING
SELECT val.in_date,
       val.geom,
       CEIL(ABS(-180 - val.pt.sdo_point.x)/90),
       val.pt.sdo_point.y
FROM (SELECT in_date,
             geom,
             SDO_GEOM.SDO_POINTONSURFACE(
                geom, 0.05) pt
      FROM partition_table) val;
```

Loading a Spatially Partitioned Table: Multicolumn Key

- Based on the **X_VALUE** and the **Y_VALUE**, data gets loaded as follows:

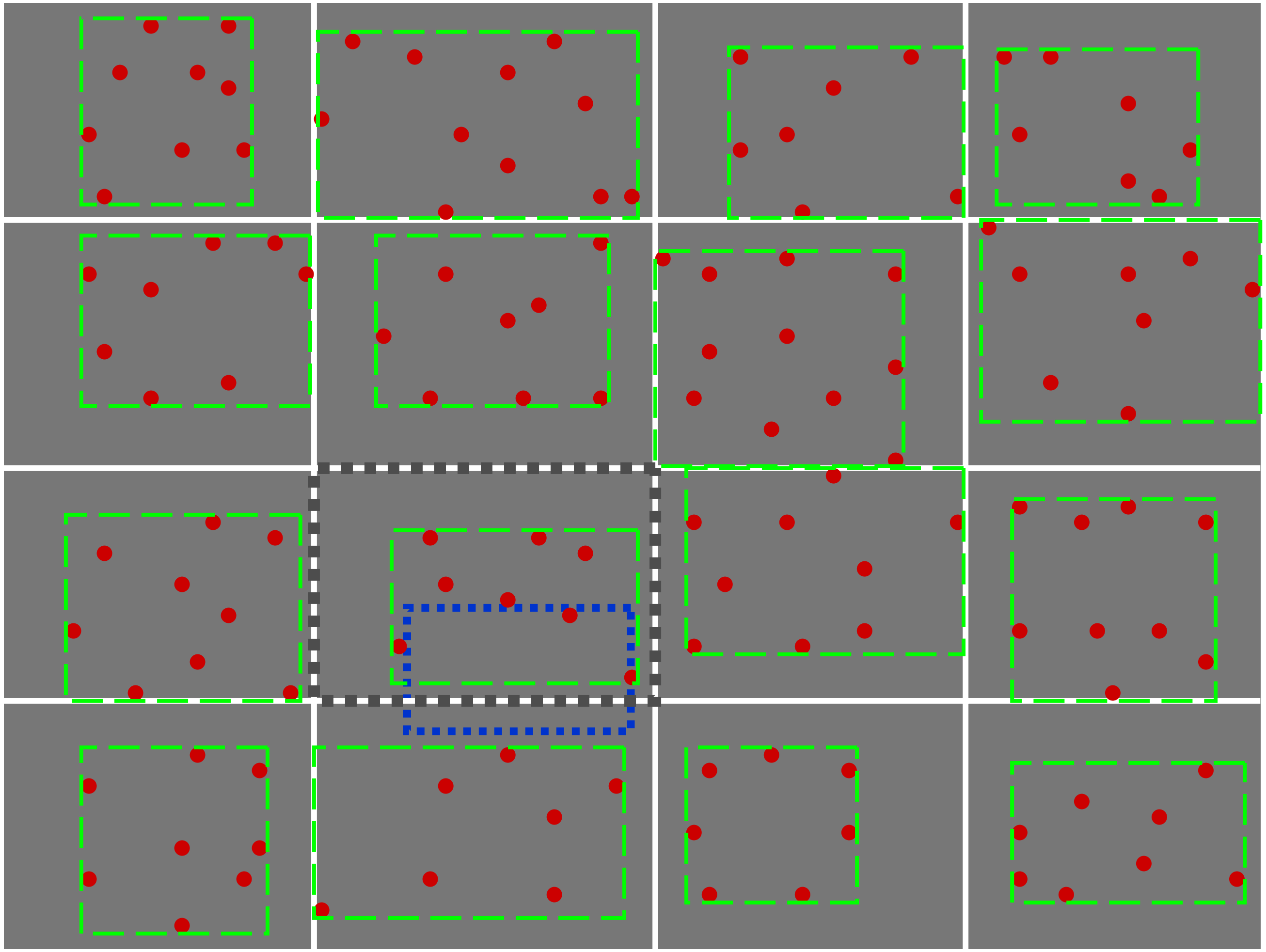
X_VALUE	Y_VALUE	Partition
1 (-180 < X <= -90)	-90 <= Y < -45	P_LT_90W_45S
1	-45 <= Y < 0	P_LT_90W_0
1	0 <= Y < 45	P_LT_90W_45N
1	45 <= Y <= 90	P_LT_90W_90N
2 (-90 < X <= 0)	-90 <= Y < -45	P_LT_0_45S
2	-45 <= Y < 0	P_LT_0_0
2	0 <= Y < 45	P_LT_0_45N
2	45 <= Y <= 90	P_LT_90W_90N

Loading a Spatially Partitioned Table: Multicolumn Key

- Based on the **`X_VALUE`** and the **`Y_VALUE`**, data gets loaded as follows (continued):

X_VALUE	Y_VALUE	Partition
3 (0 < X <= 90)-9	0 <= Y < -45	P_LT_90E_45S
3	-45 <= Y < 0	P_LT_90E_0
3	0 <= Y < 45	P_LT_90E_45N
3	45 <= Y <= 90	P_LT_90E_90N
4 (90 < x < =180)	-90 <= Y < -45	P_LT_180E_45S
4	-45 <= Y < 0	P_LT_180E_0
4	0 <= Y < 45	P_LT_180E_45N
4	45 <= Y <= 90	P_LT_180E_90N

Spatially Partitioned Table and Index: Multicolumn Key



Combining Nonspatial and Spatial Partitioning

- The nonspatial partition key is the leading key
  - Include it in the **`WHERE`** clause
  - The Oracle optimizer will use partition elimination to only search partitions associated with the partition key

  Then, after the Oracle optimizer reduces the search space:

  - Oracle Spatial partition pruning based on the query window MBR and the **`SDO_ROOT_MBR`** columns occurs only against the remaining partitions

Creating a Table Combining Nonspatial and Spatial Partitioning

- Start of create table statement

```
CREATE TABLE quarter_spatial_partition_table (
  in_date DATE,  quarter number
  geom SDO_GEOMETRY  x_value NUMBER, y_value number
PARTITION BY RANGE  QUARTER, X_VALUE, Y_VALUE
  (
   PARTITION Q1_P_LT_90W_45S VALUES LESS THAN (1,1,-45),
   PARTITION Q1_P_LT_90W_0 VALUES LESS THAN   (1,1,0),
   PARTITION Q1_P_LT_90W_45N VALUES LESS THAN (1,1,45),
   PARTITION Q1_P_LT_90W_90N VALUES LESS THAN (1,1,90),
   PARTITION Q1_P_LT_0_45S VALUES LESS THAN   (1,2,-45),
   PARTITION Q1_P_LT_0_0 VALUES LESS THAN     (1,2,0),
```

Creating a Table Combining Nonspatial and Spatial Partitioning

- End of create table statement

```
…
   PARTITION Q4_P_LT_0_45N VALUES LESS THAN (4,2,45),
   PARTITION Q4_P_LT_0_90N VALUES LESS THAN (4,2,90),
   PARTITION Q4_P_LT_90E_45S VALUES LESS THAN (4,3,-45),
   PARTITION Q4_P_LT_90E_0 VALUES LESS THAN (4,3,0),
   PARTITION Q4_P_LT_90E_45N VALUES LESS THAN (4,3,45),
   PARTITION Q4_P_LT_90E_90N VALUES LESS THAN (4,3,90),
   PARTITION Q4_P_LT_180E_45S VALUES LESS THAN (4,4,-45),
   PARTITION Q4_P_LT_180E_0 VALUES LESS THAN (4,4,0),
   PARTITION Q4_P_LT_180E_45N VALUES LESS THAN (4,4,45),
   PARTITION Q4_P_LT_180E_90N VALUES LESS THAN (4,4,90)
   );
```

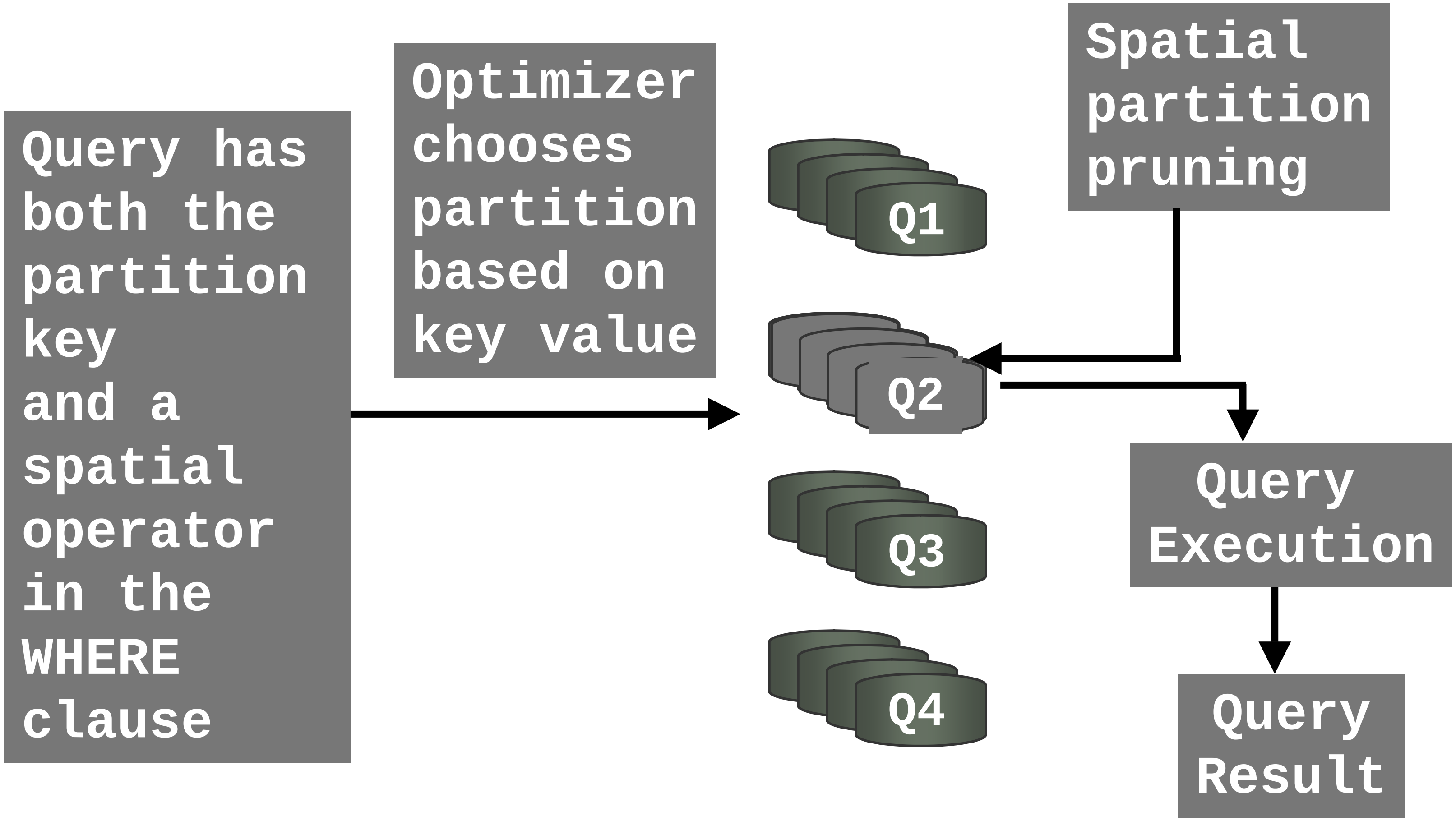
Combining Nonspatial and Spatial Partitioning

- In this example, each date range has to be associated with single value, as do all but the last partitioning key in the multicolumn range partitioning key
- Each date range (quarter) has 16 spatial partitions in it (4 quarters x 16 spatial partitions = 64 partitions)

Loading a Table Combining Nonspatial and Spatial Partitioning

```
INSERT INTO quarter_spatial_partition_table NOLOGGING
SELECT val.in_date,
       ceil (months_between (val.in_date,
               TO_DATE('31-DEC-2003'))/3),
       val.geom,
       ceil(abs(-180 -val.pt.sdo_point.x)/90),
       val.pt.sdo_point.y
FROM (SELECT in_date,
             geom,
             SDO_GEOM.SDO_POINTONSURFACE(
               geom, 0.05) pt
      FROM partition_table) val;
```

Combining Nonspatial and Spatial Partitioning



Query has both the partition key and a spatial operator in the WHERE clause

Optimizer chooses partition based on key value

Spatial partition pruning

Query Execution

Query Result

Performance Considerations of Spatial Partitioning

- Large Geometries (in area or length) that cross multiple partitions cause the **`SDO_ROOT_MBR`** to expand
  - Reduces effectiveness of spatial partitioning
  - If only a few, can create a separate partition
    - Use geometry length or area or MBR area as a criteria to determine if data is pushed off to special partition

Performance Considerations of Spatial Partitioning

- Overhead associated with checking each partition
  - A little bit over 1 millisecond/partition on 1.5 Ghz Itanium box*
  - If a very large number of partitions, overhead can be substantial
  - Working to reduce/eliminate in a future release

* Tested on Hewlett Packard Integrity RX4640 server with 4 1.5 Ghz Itanium processors running Red Hat Linux Advanced Server 3.0, with a Storageworks Enterprise Virtual Array 5000 Running VCS 3.010

Spatial Indexing

Spatial Indexing and Performance

- Create spatial indexes for the same reason other indexes are created in Oracle
  - They give us *FAST* access to data
- You can help Oracle do a better job at query time
  - Use the index parameter **`<LAYER_GTYPE>`**
  - Used for performance and geometry type checking
    - Very important for point data

Spatial Indexing and Performance

- Why is **`<LAYER_GTYPE>`** important?
  - Spatial index has tree structure of MBRs and ROWID pointers
  - What is the MBR of a point?
    - Spatial stores “degenerate” MBR information for point data
    - The degenerate MBR is the point itself
    - When doing relate processing, Oracle doesn’t have to join back to the table to get the point
    - It gets it directly from the index

Spatial Indexing and Performance

- Other optimizations
  - Spatial index compares MBRs
  - With relate processing, might not want to go directly to geometry to geometry comparisons (they are expensive)
  - May want to further compare simplified window with layer geometry MBRs
    - Compare convex hull of query window with layer geometry MBRs

Spatial Indexing and Performance

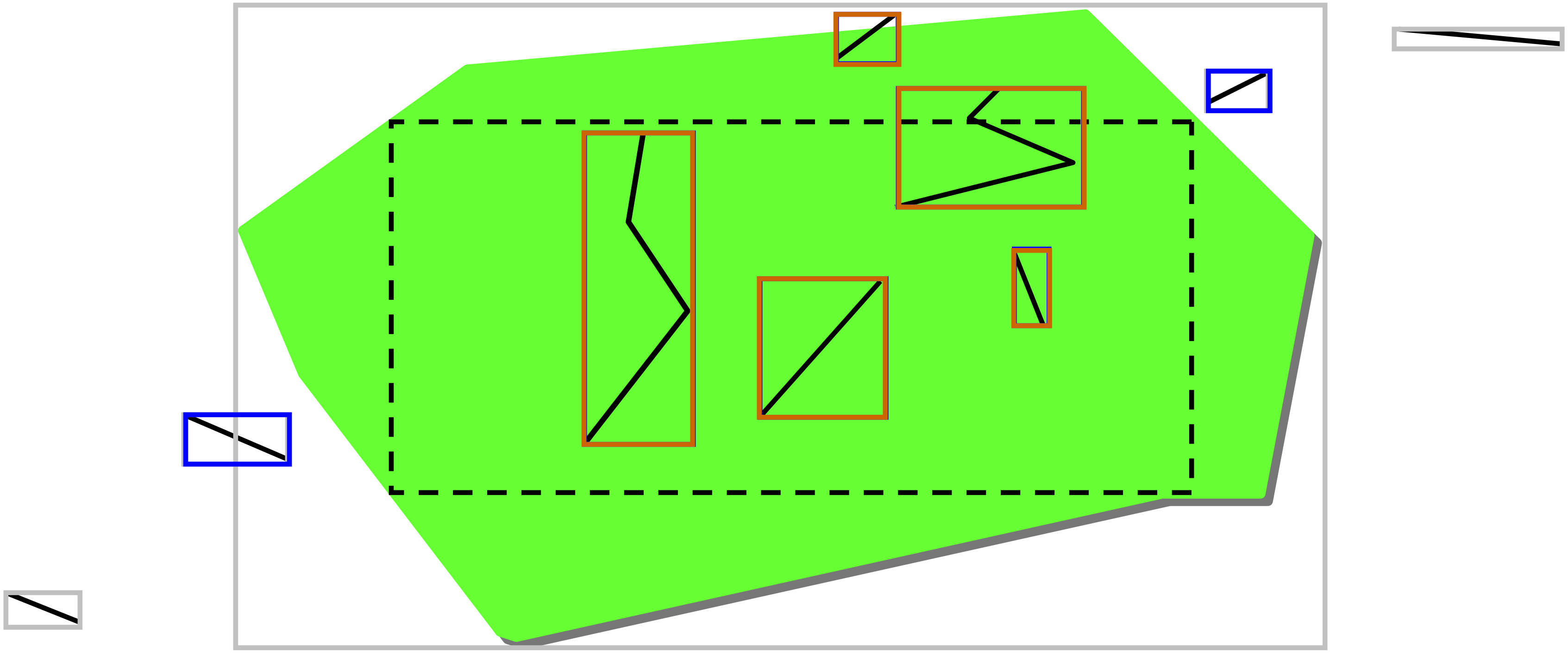
- May choose to tile query window geometry
  - Compare tiles and layer geometry MBRs
  - **<SDO_LEVEL>** parameter for **SDO_RELATE**
    - Number of times to divide the query window MBR
    - Compare interior tiles with geometry MBR
    - Default is 4, some have seen benefit with 5
      - Takes extra time to create extra level, and does more comparisons
    - Try increasing it if your window geometry area is much smaller than the MBR area

Spatial Indexing and Performance

- May choose to create interior MBR for a polygon query window
    - Interaction when layer geometry MBR is inside interior MBR
- May compare query window geometry with layer geometry MBR

Example Interior Optimization

- Which roads in the US have some interaction with this county?
  - Primary filter compares geometry approximations, so result is not exact
  - Interior optimizations are applied to candidate set
  - Geometry comparisons only where required



Spatial Indexing and Performance

- An example:
  - Customer has hundreds of thousands of rows of data
  - Mixed up all kinds of features (seismic, drill, geopolitical boundaries, etc.)
  - Their data model required this
  - Wants to use locator (no partitioning)
  - Sometimes their queries have to find a specific type of feature that has an interaction

Spatial Indexing and Performance

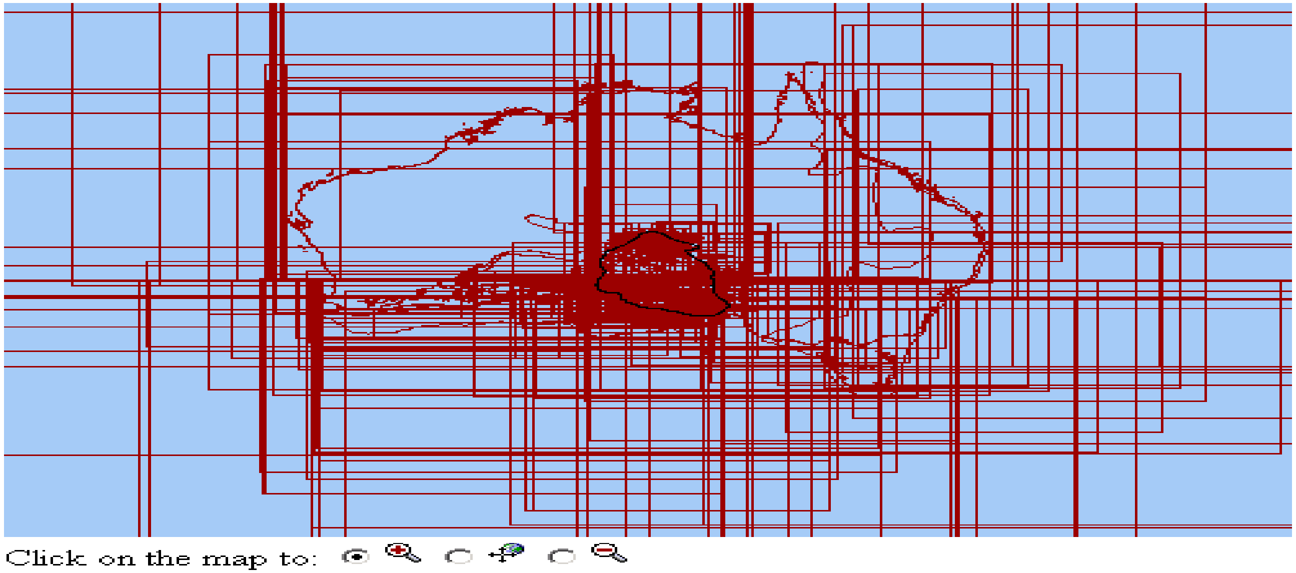
- Example query:

```
select /*+ ordered */ et.id
from entities en, entities et
where en.id = 1644321
  and et.entity_type = 'SEISMIC_AREA'
  and sdo_relate(et.geom, en.geom,
  'mask=ANYINTERACT querytype=WINDOW')='TRUE'
```

  - Takes about 70 seconds to return 2 rows
  - **`SDO_FILTER`** returns in 1 second

Spatial Indexing and Performance

- What is going on?
  - Use MapViewer for analysis
  - Just the filter (no other predicate)



Spatial Indexing and Performance

- The filter plus other predicate
- Very fast!
- Not too many rows

Spatial Indexing and Performance

- But the customer wants **`SDO_RELATE`**
  - They want it fast using locator!
  - These customers are very smart

```
select /*+ ordered */ et.id
from entities en, entities et
where en.id = 1644321
  and et.entity_type = 'SEISMIC_AREA'
  and sdo_filter(et.geom, en.geom)='TRUE'
  and sdo_geom.sdo_distance(
     et.geom,en.geom,0.5) = 0;
```

Spatial Indexing and Performance

- Whenever the query window comes from a table using an operator, make sure to use the /*+ ordered */ hint
  - Ensures the optimizer will generate a fast execution plan
- If you are joining tables, use a polygon layer as the driving table
  - Enables interior tile optimizations
  - Unless the other table is much smaller

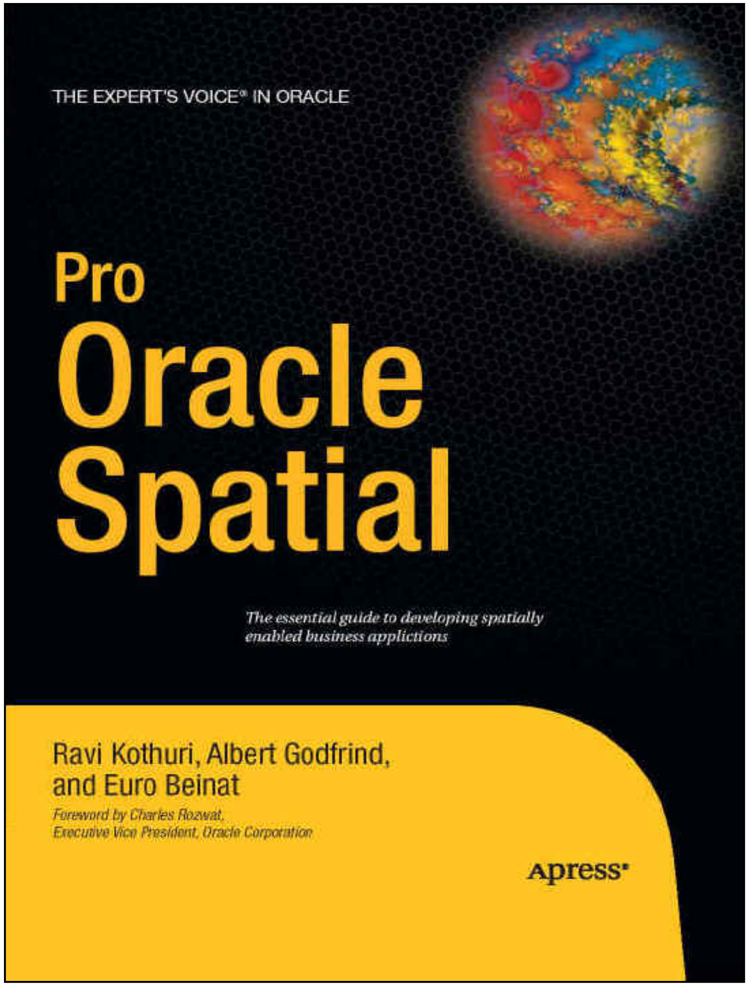
Spatial Indexing and Performance

- When moving spatial data and indexes
  - Use transportable tablespaces
  - New support in Oracle 10g
  - One important restriction:
    - Cannot transport indexes across endian platforms

Next Steps

- Related Web Sites For More Information
  - otn.oracle.com/products/spatial
- Oracle Training
  - Oracle Spatial 10*g*: Fundamentals
  - Oracle Spatial 10*g*: Advanced
- Discussion Forum on OTN
- Venues such as this (Spatial SIG)

Pro Oracle Spatial


THE EXPERT'S VOICE® IN ORACLE
Pro
Oracle
Spatial
The essential guide to developing spatially enabled business applictions
Ravi Kothuri, Albert Godfrind, and Euro Beinat
Foreword by Charles Rozwat, Executive Vice President, Oracle Corporation
Apress®

- “An Essential Guide” for developing Spatially-enabled Business Applications
- For Oracle Professionals (Developers, DBAs), Spatial/GIS Analysts, Consultants, …
- By Oracle Spatial Developers

  and Geodan, NL
- Covers Spatial technology in Oracle (Oracle Locator, Spatial, Mapviewer products)
- Includes Sample code, Tips,…
- ISBN: 1-59059-383-9
- http://www.apress.com/book/bookDisplay.html?bID=315

Q&A

ORACLE IS THE INFORMATION COMPANY