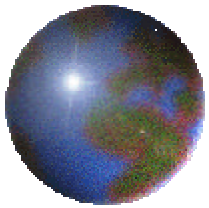
DBA World Tour 2001



Tuning When You Can't Touch The Code…

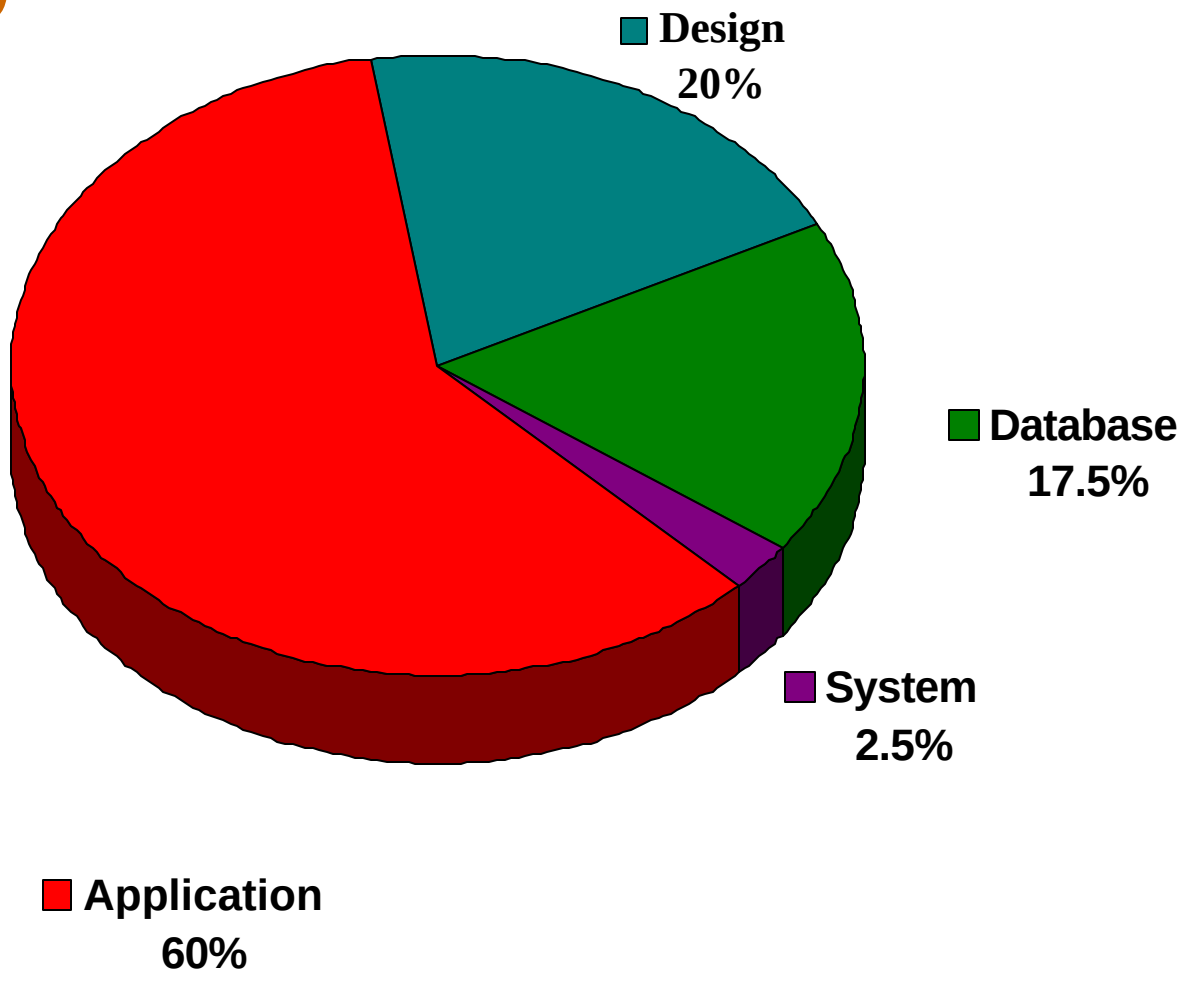
Michael R. Ault

Senior Seminar Leader

DBAGroup LLC DBA World Tour 2001

HTTP://WWW.DBAGROUP.NET/

Tuning Overview



But We Can't Touch The Code!

- Many application vendors forbid customers to touch code
  - SAP
  - Oracle Financials
  - Seibel
  - PeopleSoft
  - Baen

What Can Be Done?

Oracle Version:	**7.3.x**	**8.0.x**	**8.1.x**
Optimize Internals	X	X	X
Optimizer Modes	X	X	X
Add Resources	X	X	X
Tune Tables and Indexes	X	X	X
Parallel Query	X	X	X
Better Indexes		X	X
Index Only Tables		X	X
Partitioning		X	X
New INI features		X	X
Subpartitioning			X
Outlines			X
Resource Groups			X

Optimizing Oracle Internals

- Database Buffer Tuning
- Database Writer Tuning
- Shared Pool Tuning
- Checkpoints
- Redo Logs
- Rollback Segments
- Sort Area Size

Database Buffer Tuning

- DB_BLOCK_SIZE
- DB_BLOCK_BUFFERS
- Hit Ratios
- X$BH
- Buffer Busy Waits and V$WAITSTAT

Database Buffer Tuning

- Hit ratio:

```
CURSOR get_stat(stat IN VARCHAR2) IS
SELECT name,value FROM v$sysstat WHERE name
  = stat;

'db block gets‘,'consistent gets‘, 'physical
  reads‘,’direct reads’
```

Database Buffer Tuning

```
h_ratio := (1- (p_reads – d_reads)
  /(db_gets + con_gets)))*100;

---OR---

CURSOR get_hratio IS
  SELECT name, (1-(physical_reads/
  (db_block_gets+consistent_gets)))*100
  H_RATIO
  FROM v$buffer_pool_statistics;
```

Database Buffer Tuning

From: “ORACLE Performance Tuning Tips & Techniques”, Richard Niemiec, Oracle Press

X$BH:

```
SELECT DECODE(state, 0, ‘FREE’,
        1, DECODE(lrba_seq,0, ‘AVAILABLE’, ‘BEING USED’),
        3, ‘BEING USED’, state) “BLOCK STATUS”, COUNT(*)
FROM x$bh
GROUP BY
    decode(state,0,’FREE’,1,decode(lrba_seq,0,’AVAILABLE’,
    ’BEING USED’),3,’BEING USED’,state);
```

If 10-25% buffers free after 2 hours of use, good.

Database Buffer Tuning

Buffer Busy Waits and V$WAITSTATE

```
SELECT
   class,"COUNT"
   FROM
   v$waitstat
   WHERE
   class = ‘data block’;
```

Large number of data block waits may indicate a need for more db_block_buffers

Database Writer Tuning

- Use V$SYSSTAT to calculate average length of the dirty write queue, values larger than 100 show need for more DB_BLOCK_BUFFERS or DB_WRITERS or increase the size of DB_BLOCK_WRITE_BATCH

Database Writer Tuning

From “Oracle Performance Tuning” , Mark Gurry and Peter Corrigan, O’Reilly Press:

```
SELECT
DECODE (name, ‘summed dirty write queue length’, value)/
DECODE (name, ‘write requests’, value) “Write Request Length”
FROM v$sysstat
WHERE name IN ( ‘summed dirty queue length’, ‘write requests’) and value>0;
```

Database Writer Tuning

In Oracle 7 you could change:

- DB_WRITERS (2 x #disks)
- DB_BLOCK_BUFFERS
- _DB_BLOCK_WRITE_BATCH
- _DB_BLOCK_MAX_SCAN_CNT
- DISK_ASYNC_IO

Database Writer Tuning

In Oracle 8.0:

- DBWR_IO_SLAVES (2 x #disks)
- DB_BLOCK_BUFFERS
- _DB_BLOCK_WRITE_BATCH
- _DB_BLOCK_MAX_SCAN_CNT
- DISK_ASYNC_IO

Database Writer Tuning

In Oracle8i:

- DB_WRITER_PROCESSES (2 x #disks)
- DBWR_IO_SLAVES
- DB_BLOCK_LRU_LATCHES
- DB_BLOCK_MAX_DIRTY_TARGET
- DB_BLOCK_BUFFERS
- _DB_BLOCK_WRITE_BATCH
- _DB_BLOCK_MAX_SCAN_CNT
- DISK_ASYNC_IO
- Many more “_” parameters

Database Writer Tuning

- Also look at buffer waits

From Gurry and Corrigan:

SELECT name, value FROM v$sysstat
WHERE name='free buffer waits';

Shared Pool Tuning

- Look at not just how full shared pool is:
  - Review code usage – are areas reused?
  - If mixed mode (reused and garbage) use flushing
  - If data dictionary shows contention, increase pool size
  - Don't allow more than 5000 SQL areas
  - Use V$SQLAREA to determine reuse
  - If COUNT on V$SQLAREA takes a long time, flush!

Shared Pool Tuning

```
create or replace view sql_garbage as
select
  b.username users,
  sum(a.sharable_mem+a.persistent_mem) Garbage,
  to_number(null) good
from
   sys.v_$sqlarea a,
   dba_users b
where
  (a.parsing_user_id = b.user_id and a.executions<=1)
group by b.username
```

Shared Pool Tuning

```
union
select distinct
  b.username users,
  to_number(null) garbage,
  sum(c.sharable_mem+c.persistent_mem) Good
from
   dba_users b,
   sys.v_$sqlarea c
where
  (b.user_id=c.parsing_user_id and c.executions>1)
group by b.username;
```

Shared Pool Tuning

```
select 1 nopr,
a.users users,
to_char(a.garbage,'9,999,999,999') garbage,
to_char(b.good,'9,999,999,999') good,
to_char((b.good/(b.good+a.garbage))*100,'9,999,999.9
  99') good_percent
from sql_garbage a, sql_garbage b
where a.users=b.users
and a.garbage is not null and b.good is not null
union
select 2 nopr,
'-------------' users,'--------------' garbage,'----
  ----------' good,
'-------------' good_percent from dual
```

Shared Pool Tuning

```
union
select 3 nopr,
to_char(count(a.users)) users,
to_char(sum(a.garbage),'9,999,999,999') garbage,
to_char(sum(b.good),'9,999,999,999') good,
to_char(((sum(b.good)/(sum(b.good)+sum(a.garbage)
  ))*100),'9,999,999.999') good_percent
from sql_garbage a, sql_garbage b
where a.users=b.users
and a.garbage is not null and b.good is not null
order by 1,3 desc
/
```

Shared Pool Tuning

```
break on report
compute sum of areas on report
select
  b.username,
  count(1) areas
from
   sys.v_$sqlarea a,
   dba_users b
where
  (a.parsing_user_id = b.user_id)
group by username
```

Shared Pool Tuning

- Use DBMS_SHARED_POOL to pin large or frequently used PL/SQL, cursors, sequences, triggers, packages and procedures
- Determine usage from V$SQLAREA

Tuning Checkpoints

- Start the checkpoint process in Oracle 7
- Use the CHECKPOINT_PROCESS parameter (automatic post Oracle8)
- Set LOG_BUFFER_SIZE properly
- Set LOG_SMALL_ENTRY_MAX_SIZE (Gone in 8i)
- Set LOG_SIMULTANEOUS_COPIES (Gone to "_" in 8i)
- Set LOG_CHECKPOINT_TIMEOUT
- Set LOG_CHECKPOINT_INTERVAL
- Set LOG_ENTRY_PREBUILD_THRESHOLD(Gone to "_" in 8.0, gone in 8i)
- FAST_START_IO_TARGET (8i only)
- DB_BLOCK_CHECKPOINT_BATCH (Gone in 8i)

Tuning Redo Logs

- Actually tune LGWR process to optimize log writes
- LGWR writes when log buffers 1/3 full, or on COMMIT
- Too small results in frequent inefficient IO, too large results in too long a write
- Dependent on transaction size

Tuning Redo Logs

To Determine average transaction size as far as redo buffer writes:

$$\frac{\text{(redo size + redo wastage)}}{\text{Redo writes}}$$

From V$SYSSTAT.

Tuning Redo Logs

- If you size log buffers smaller than this then you will have problems
- If many times this size problems
- Size at near this size.

Tuning Redo Logs

Size actual redo logs such that they switch every thirty minutes, or based on the amount of data you can afford to lose (loss of the active redo log results in loss of its data.)

Tuning Rollback Segments

- Size INITIAL = NEXT = Size of average transaction
- Size OPTIMAL = Average size of large transaction rounded up to next value of NEXT
- Determine these values using V$ROLLSTAT, V$ROLLNAME and DBA_ROLLBACK_SEGS

Sort Area Size

- Sort area size should be set to the size of the average sort being performed.
- This is found empirically
- Set to half megabyte, then scale up to reduce Sorts(disk)
- Set temporary tablespace such that INITIAL=NEXT = Sort area size + 1 db_block_size

Optimizer Modes

- RULE or CHOOSE
- CHOOSE will also be used if:
- HINTS used
- Mode set to FIRST_ROWS or ALL_ROWS
- New features will not be used if set to RULE
- Must use frequent ANALYZE with CHOOSE
- Use histograms with skewed data

Rule Based Optimizer

- Works according to a predefined set of rules that dictates which access path is best
- RBO has been with Oracle since its early days
- RBO is still supported in version 8, but users are advised to migrate toward cost based optimization (i.e., on new projects or implementations)

Who Should Use RBO?

- Sites that have developed Oracle applications using Versions 6 and 7 in the rule based environment

- Sites that have tuned their applications for best performance under RBO should remain with RBO and gradually migrate toward CBO

Cost Based Optimizer

- Uses statistical knowledge of data volumes and data distribution, processing assumptions and access patterns to determine the appropriate access path to data
- Introduces the concept of a “hint”. Hints can force the usage of specific access paths
- Optimizes queries for best throughput, unless instructed otherwise

Hints… Explained

- Once a hint (other than “RULE”) is specified, the ENTIRE statement works in the cost based optimization mode

- This is true even if the init.ora parameter called “OPTIMIZER_MODE” is set to “CHOOSE” and statistics are not present at the object level

Best Throughput

- By default, the cost based optimizer generates an access path that reads all rows that fit the query criteria in the shortest time

- The “FIRST_ROWS” hint can be specified to change the optimizer’s goal to select an access path that fetches the first set of qualified rows in the shortest time

Adding Resources

- If all possible tuning has been accomplished then add resources
- Memory will give the most tuning benefit
- Multiple CPUs will help if you use parallel query
- Parallel query, multiple processes (DBWR, LGWR, etc) not much help with single CPU
- Use proper striping (RAID1/0 is usually best performer, RAID5 most dependable)

RAID Levels and Recommendations

RAID	Type of Raid	Control	Database	Redo Log	Archive
0	Striping	Avoid	OK	Avoid	Avoid
1	Shadowing	Best	OK	Best	Best
0+1	Striping and Shadowing	OK	Best	Avoid	Avoid
3	Striping with static parity	OK	OK	Avoid	Avoid
5	Striping w/rotating parity	OK	Best(1)	Avoid	Avoid

1 – Only if RAID 0+1 not available

Tuning Tables and Indexes

- Tables need to be properly sized and spread on the disk array
- Indexes need to be properly sized, ordered and spread on disk array
- Index order is driven by column order in the query until 8i
- In 8i enable query rewrite

Rebuild Tables When…

- Extents exceed ?
- Too many chained blocks
- Not enough free lists
- Initrans set to low

Rebuild Indexes When…

- Index has too many levels
- Index is too broad
- Index clustering factor too high

Assumptions Used by CBO

- Uniform data distribution (*when histograms are not present at the column level*)
- Assumptions on the amount of data to be retrieved when using bind variables
- Pessimistic predictions in the following areas:
  - A multi-user system
  - A low buffer hit ratio

Assumptions on Bind Variables

- The CBO makes the following assumptions about selectivity (for the case of an equal predicate if no statistics exist or for all other predicates even if statistics exist):
  - Default selectivity for all predicates besides "=' is 0.05 (5%)
  - Default selectivity for equal ("=") is 0.01 (1%)

Filtering Factor

- The filtering factor (FF) of a predicate is a number that indicates what percentage of data will be retrieved after applying this predicate
- To calculate the filtering factor of a predicate you must know the number of distinct values on the column as well as its maximum and minimum values
- Filtering is based upon the type of operator and the distribution of values
- Unless specifically informed, the CBO assumes uniform distribution (i.e., no Histograms)

Calculating Filtering Factors

*NOTE: both **n** and **m** are constants.*

- "C1 = n"
  - FF = 1 / Number of distinct values
- "C1 > n"
  - FF = (Max(C1) - n) / (Max(C1) - Min(C1)
- "C1 < n"
  - FF = (n - Min(C1)) / (Max(C1) - Min(C1)
- "C1 between n and m"
  - FF = (m - n) / (Max(C1) - Min(C1)

Example I:

Access Path Determination for a Single Table:

- Assume we have the following SQL statement:

  Select * from EMP where EMPNO > 1000

- There is an index on EMP(EMPNO)

- The CBO has to choose either to use the index or to perform a full table scan

Example I:

Statistical Information:

- Table Information:
  - Number of rows: 100,000
  - Number of blocks: 2,000
  - DB_FILE_MULTIBLOCK_READ_COUNT: 8
- Index Information:
  - Number of leaf blocks: 300
  - Clustering factor: 20,000
- Column Information for EMPNO:
  - Number of distinct values: 100,000
  - Minimum value: 1
  - Maximum value: 100,000

A Full Table Scan...

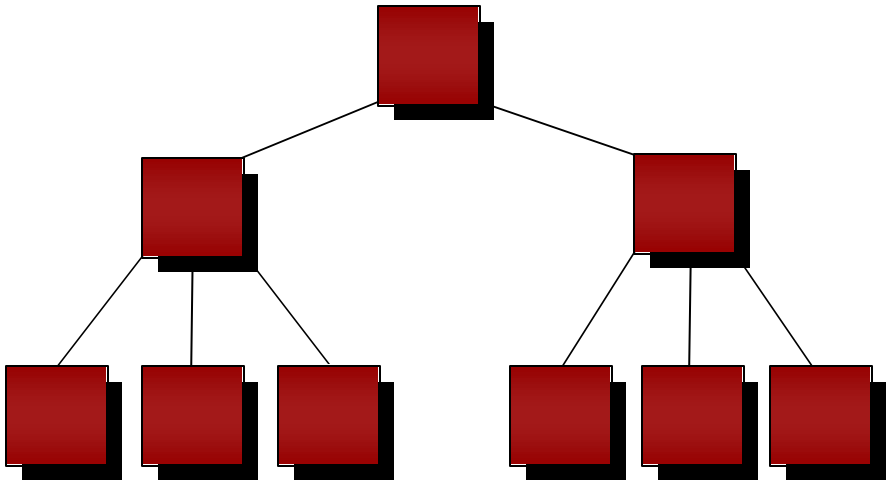
- Will read all table blocks (2,000)
- Each block will be read once
- Each I/O operation will read 8 blocks
- The total number of I/O operations will be **2,000 / 8 = 250**
- CPU activity in this case will be relatively small

B-Tree Index (a.k.a., Regular Index)

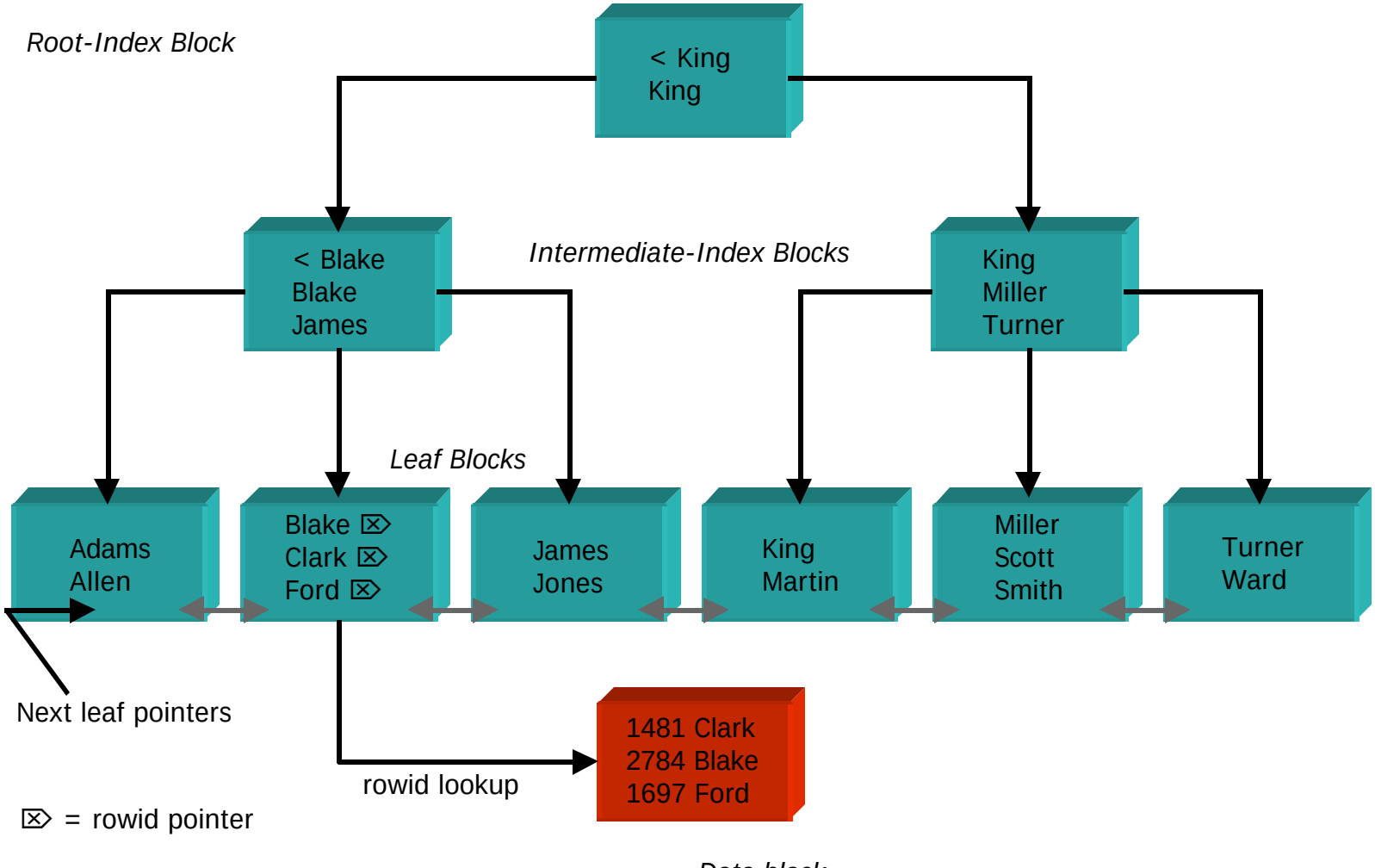
- Best used for data with a high degree of cardinality (*i.e., many distinct values or value combinations*)
- Most common type of index used in Oracle (and other) DBMSs
- Used for performance enhancement, or to enforce uniqueness
- Reduces I/O by providing a faster means by which to retrieve data. * That is the GOAL !!

*B*Tree (*Balanced Tree*) Index*

A Balanced tree is defined as a tree structure in which any path from the root page to any leaf page will traverse the same number of levels



Internal Structure of a B-Tree Index



*B*Tree Index Performance Considerations*

- Since Oracle performs dynamic maintenance on Indexes, mass data load performance can be increased by dropping the index prior to the load and then recreating it afterwards.
- Avoid creating B*Tree indexes on low cardinality columns.
- Deleted space is not reclaimed automatically. Reorganize regularly to reclaim “white space”
- Look at B*Tree Reverse indexes for time series and sequential increment index columns.
- Index Only Access Path

Index Range Scan – Scanning the Index

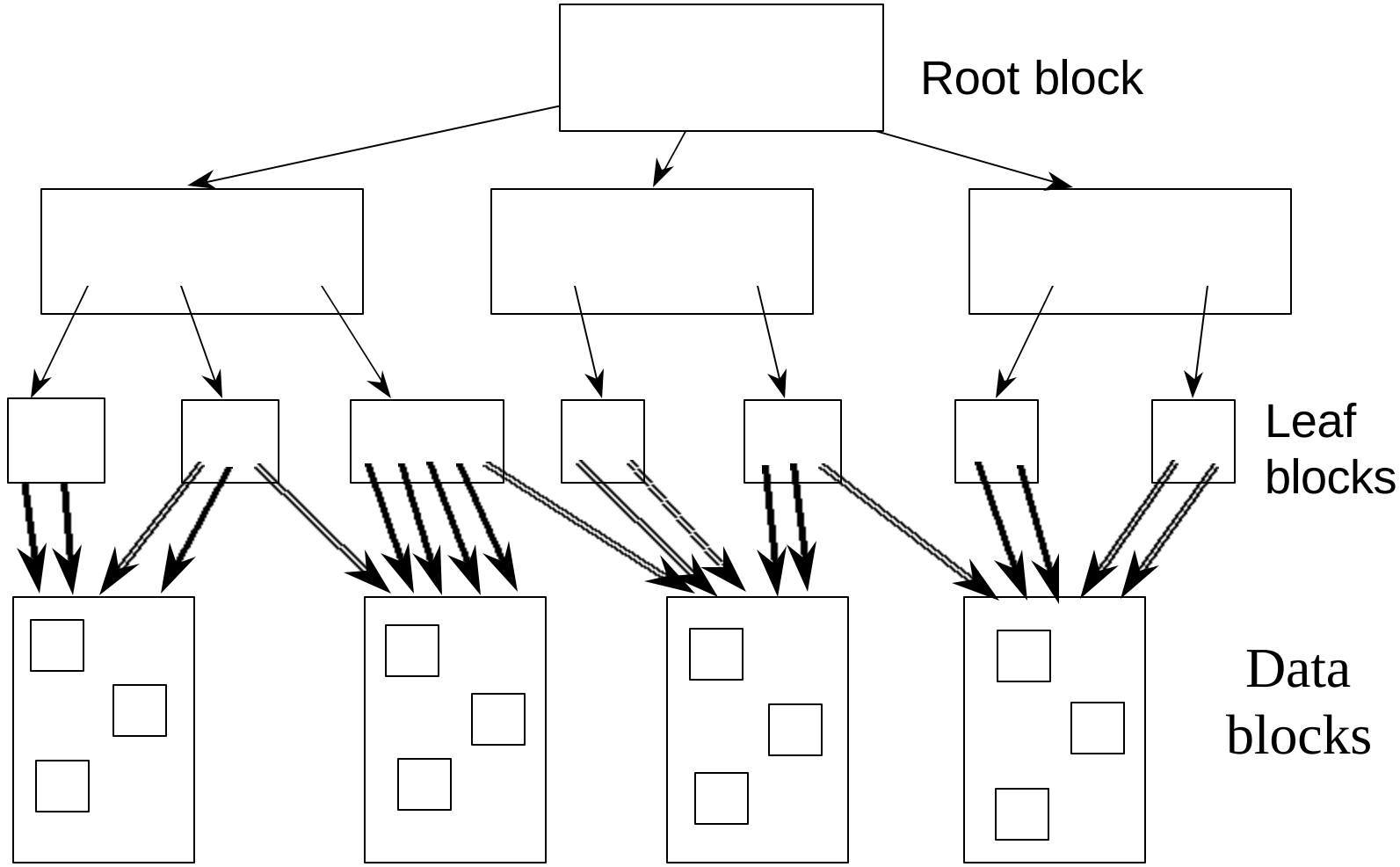
- We have to calculate what percentage of data will be read
- Assuming uniform distribution:

  (100,000 - 1,000) / (100,000 - 1) = 99% *of the data will be read*
- This means that 99% of the leaf blocks will be read,
  ∴ **99% * 300 = 297** I/O operations only for accessing the index
- For each index entry the relevant data block in the table must be read also

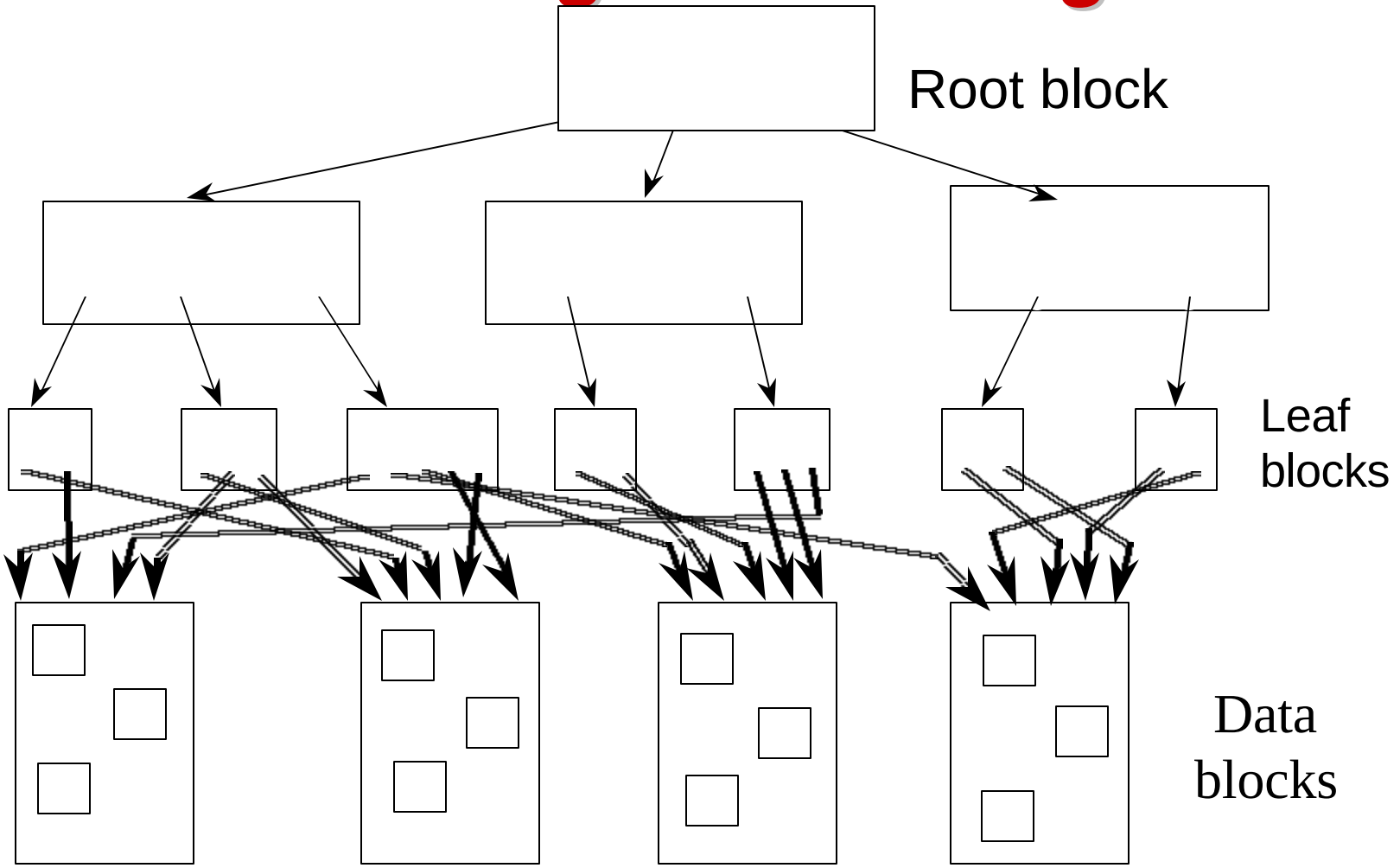
Index Range Scan - Clustering Factor...

- The index clustering factor (CF) determines the price of accessing data via the rowids retrieved from the index
- The CF tells you how many data blocks are read in a full index scan. You can determine the actual number of data blocks read by multiplying the CF by the percentage of data to be read
- The CF can range between the number of blocks containing data to the number of rows in the table

Index With a Low Clustering Factor...



Index With a High Clustering Factor...



Index Range Scan: Scanning the Table

- The clustering factor of the index is **20,000**
- That will bring the number of blocks read to **99% * 20,000 = 19,800**
- The total number of index and data I/O operations is **19,800 + 297 = 20,097**
- Since the number of I/O operations performed by the index scan is higher than the number of I/O operations performed by the full table scan, the full table scan will be selected by the optimizer

Example II:

Same Query, Different Filtering…

- If the query is …
  Select * from EMP where EMPNO > 99000;
- The selectivity will be
  (100,000 - 99,000) / (100,000 - 1) = 1%.
- The number of I/O operations via the index will be:
  1% * 300 + 1% * 20,000 = 203
- This number of I/O operations is lower than the number of I/O operations performed by the full table scan; therefore, the index will be selected

Adjusting the Cost in Oracle 8

- In Oracle 8i the total cost of the index is adjusted using the following formula:

  Adjusted Cost = Cost * OPTIMIZER_INDEX_COST_ADJ / 100

- This adjustment bypasses the Oracle assumption of a low buffer hit ratio used in the data access calculation

Previous Example With New Parameter

- If you assume that most of the data will be placed in the buffer cache and remain there, you can set **OPTIMIZER_INDEX_COST_ADJ** to a value less than 100, reflecting the percentage of time data will be found in the buffer
- ∴ If you set the **OPTIMIZER_INDEX_COST_ADJ** to 10, the price of accessing any index is 10% of the previously calculated price, presuming that data will be found in the buffer cache 90% of the time on average

*Select * from EMP where EMPNO > 1000;*

- Previous price was:
  - **297** I/O operations for the index access
  - **19,800** I/O operations for the table access
- **OPTIMIZER_INDEX_COST_ADJ=10** will cause:
  - No change in the index access price
  - Table access price to become: **19,800*10/100=1980**
  - Total price of **1980+297=2277**
- Index was not used and is still not used

*Select * from EMP where EMPNO > 99000;*

- Previous price was:
  - **3** I/O operations for the index access
  - **200** I/O operations for the table access
- **OPTIMIZER_INDEX_COST_ADJ=10** will cause:
  - No change in the index access price
  - Table access price to become: **200*10/100=20**
  - Total price of **3+20=23**
- Index was used and is still used

*Select * from EMP where EMPNO > 90000*

- Original price (without adjusting) is:
  - **30 (10%*300)** I/O operations for the index access
  - **2000 (10%*20000)** I/O operations for the table access
- **OPTIMIZER_INDEX_COST_ADJ=10** will cause:
  - No change in the index access price
  - Table access price to become: **2000*10/100=200**
  - Total price of **30+200=230**
- Index was not used before because the price was over 2000 compared with a price of 250 but is used after setting this parameter because the price dropped to 230

But, *caveat emptor…!*

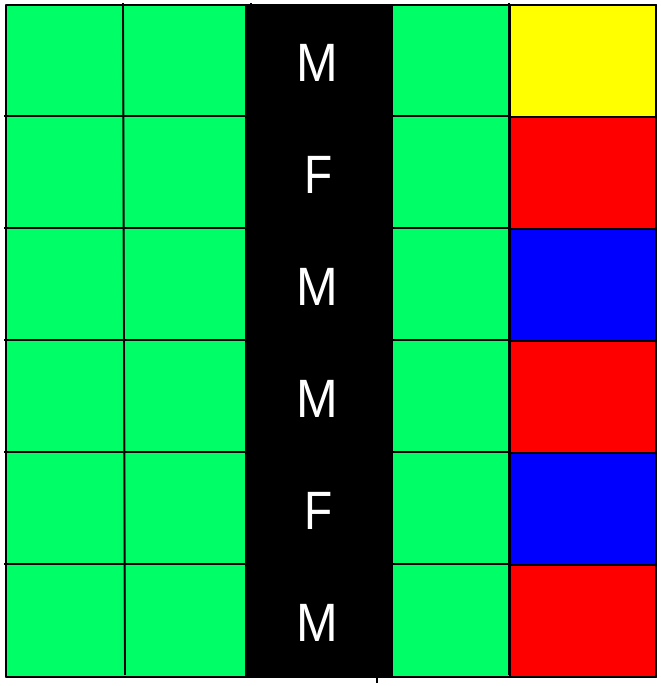
- It is very tempting to explicitly set a low value for OPTIMIZER_INDEX_COST_ADJ and force index usage in all cases
- *However*, you must be sure that your buffer cache can support data remaining in the buffer so that the assumption will be correct
- You must also be sure that the buffer cache remains in memory and is not paged out (*i.e., make sure you have adequate physical memory!*)

Bitmap Indexes

- New in Oracle 7.3.3. Popularity has increased with Oracle8.
- Efficient index for low cardinality data, e.g. {Male; Female}, {Yes; No}
- Compressed storage.
- May co-exist with B-tree indexes on the same table and/or in the same query
- Fast mathematical processing of multi-key filters.
- Very popular for Data Warehouse applications.

Bitmap Indexes - Example

- Create BITMAP indexes ..... Gender & Color



Yellow: **<1,0,0,0,0,0>**

Red: **<0,1,0,1,0,1>**

Blue: **<0,0,1,0,1,0>**

Male: **<1,0,1,1,0,1>**

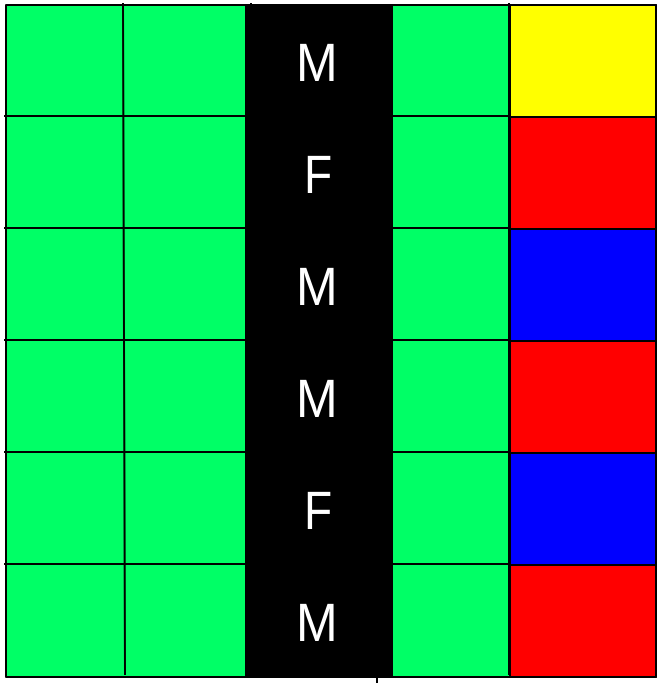
Female: **<0,1,0,0,1,0>**

0 - value not present for the row

1 - value exists for the row.

Bitmap Indexes - Query Resolution

- **...WHERE Gender = 'M' and Color = 'RED';**



Red: **<0,1,0,1,0,1>**

Male: **<1,0,1,1,0,1>**

Results:

Rows 4 & 6 match the criteria

When Are Bitmap Indexes Useful?

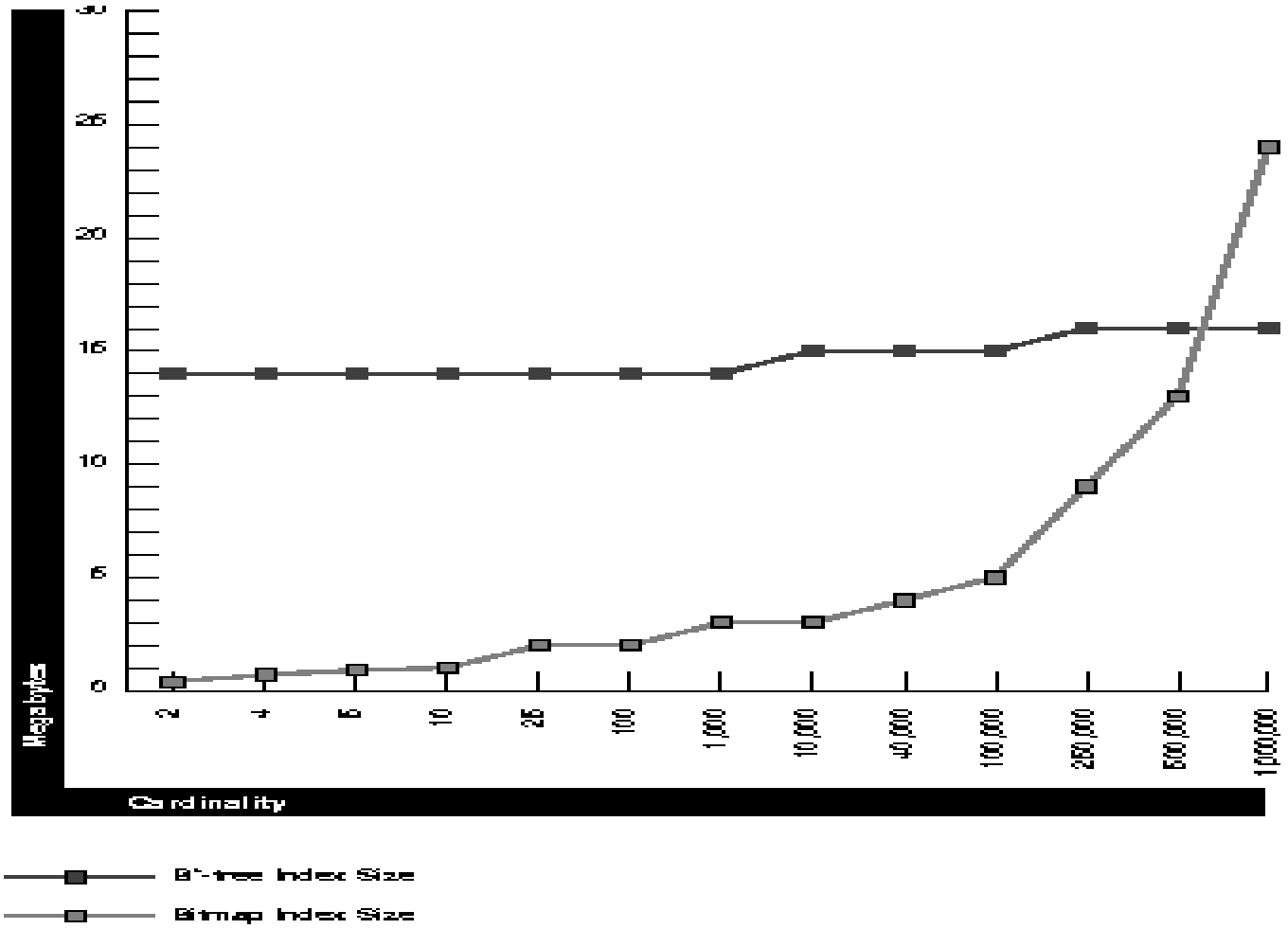
- The WHERE clause contains multiple predicates on low-cardinality columns
- The individual predicate values on these low-cardinality columns return a large number of rows
- The tables being queried have very high row counts

When Are Bitmap Indexes Not Useful?

- High cardinality columns
- When row level locking is required - locking on the bitmap index is done at the bitmap category level.
- High volume OLTP systems
- A full table scan is often more efficient if only a *single* bitmap value is filtered

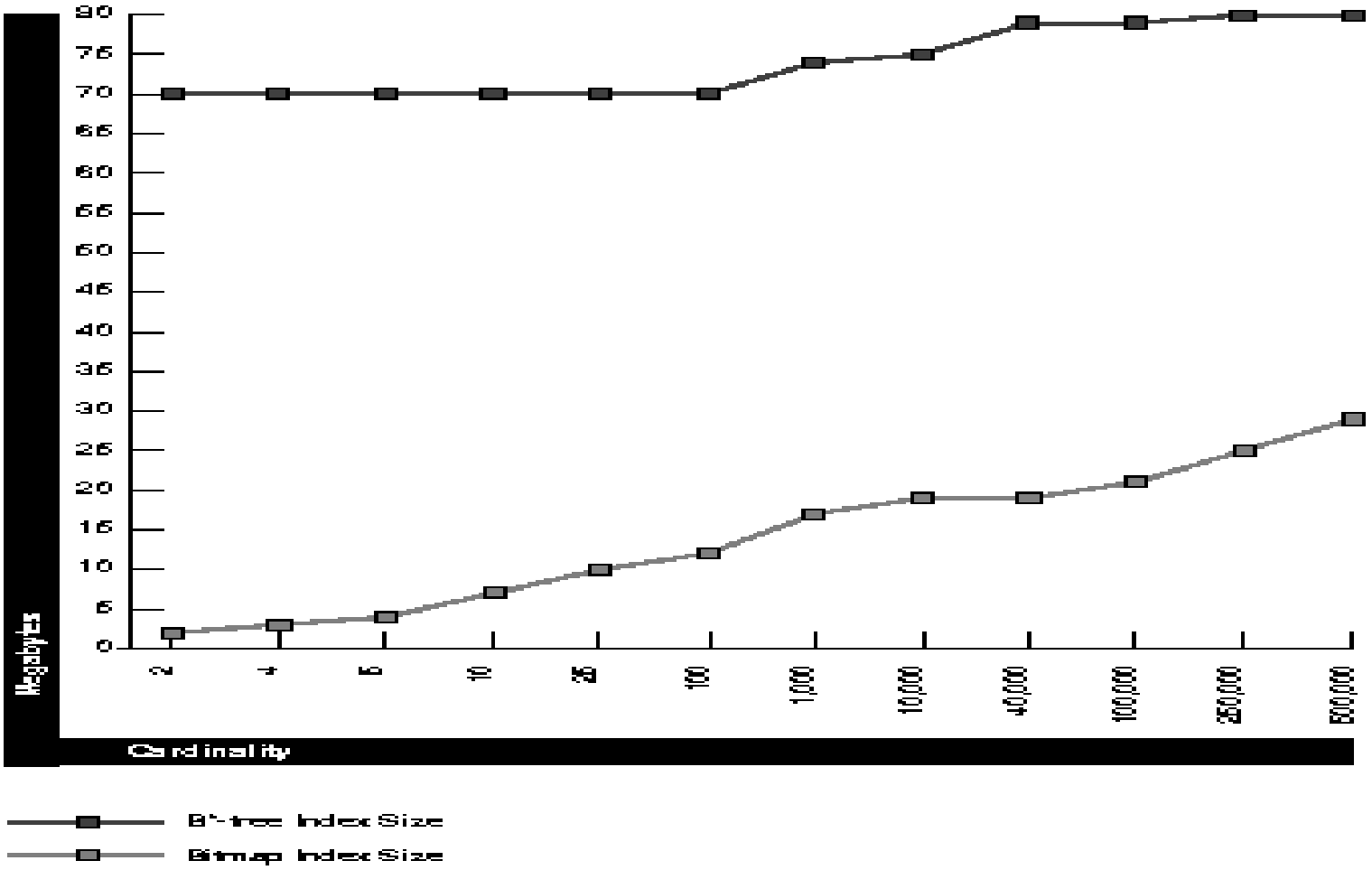
Bitmap vs. B*Tree Storage…1

Assuming 1,000,000 row table



Bitmap vs. B*Tree Storage…2

Assuming 5,000,000 row table



Initialization Parameters for Bitmap Indexing

✦CREATE_BITMAP_AREA_SIZE

Determines the amount of memory allocated for bitmap creation. Default is 8MB. If cardinality is small, this value can be reduced significantly.

✦BITMAP_MERGE_AREA_SIZE

Amount of memory to use for merging bitmap strings. Default value is 1MB. Larger value can improve performance since the bitmap segments must be pre-sorted before being merged into a single bitmap.

Bitmap Index Performance Considerations

- Larger block sizes can improve the storing and retrieving of bitmap information.
- To compress storage further, use the NOT NULL constraint on bitmap index columns.
- Nulls do exist in bitmap indexes, therefore they can be used to support IS NULL and IS NOT NULL conditions.
- ALTER TABLE command that modifies a bitmap index column may *invalidate* the index structure.
- Bitmap Indexes are not considered by RBO.

Function Based Indexes

- New in 8i
- Allow use oif functions such as UPPER, LOWER, DECODE in indexes
- Where clause function use must match index function use
- Must use CBO and have query rewrite enabled
- Index and base table must be ANALYZED
- Any FUNCTION or METHOD used by a FBI must be DETERMINISTIC

Reversed Key Index

- New in 8.0
- Used for high volume inserts to prevent index tilting
- Can’t be used with index range scan
- Will have large clustering factor

Index Organized Tables

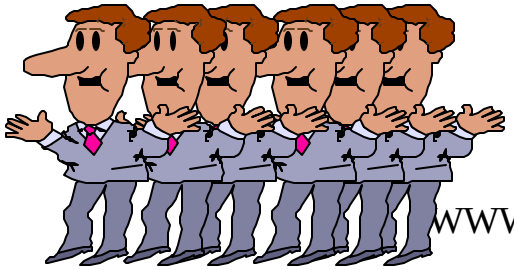
- New to Oracle 8 (although the concept has existed *operationally* for some time)
- Index is a regular B*tree structure, but rather than having a ROWID stored in the leaf blocks, the leaf blocks store the actual data.
- Index is built on the primary key of the table, thus it is suitable for applications that access the data by primary key
- Full index scan returns all rows in primary key order and is not affected by the clustering factor (i.e., fast full scans)

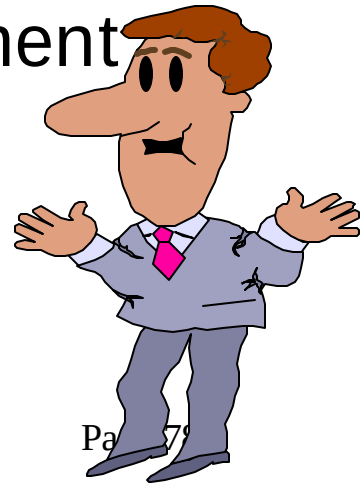
Advantages of IOTs

- Access to data via equality and range scan predicates is faster than in traditional model (less I/O intensive)
- DML is also accelerated because there are no data segments to be maintained along with the index
- Very well suited to OLAP/DWH/VLDB environments
- *However*, because there are no ROWIDs in an IOT, it cannot be used with distributed transactions or advanced replication

What is partitioning?

- The decomposition of large objects into smaller, more manageable pieces
- Physical partitioning introduced in Oracle8, although logical partitioning (via partitioned views) has been around for some time...
- Physical partitions have individual, and potentially different, storage attributes
- Each partition is stored in a separate segment and (optionally) in a separate tablespace





Partitioning Support in Oracle

- Oracle supports partitioned tables and indexes.
- An example of a DDL that creates a partitioned table:

```
CREATE TABLE sales (acct_no         NUMBER(5),
                    acct_name       CHAR(30),
                    amount_of_sale NUMBER(6),
                    week_no         INTEGER)
PARTITION BY RANGE (week_no)
(PARTITION sales1 VALUES LESS THAN (4) TABLESPACE ts0,
 PARTITION sales2 VALUES LESS THAN (8) TABLESPACE ts1,
 . . .
 PARTITION sales13 VALUES LESS THAN (52) TABLESPACE
ts12)
```

Partition "Pruning"

- The major performance advantage of partitioning lies in the Oracle optimizer's ability to eliminate (or "prune") unnecessary partitions from consideration
- Pruning is done either at parse time for statements with constants or at execution time for statements with bind variables
- Once the partition range has been established, the optimizer can strip (reduce) redundant predicates from the WHERE clause, if all rows in the partition are qualified, and accelerate query evaluation
- In most cases, pruning increases both performance and availability

Advantages of Partitioning

- Many database/application environments and situations can benefit from partitioning, including:
  - VLDBs
  - Tight maintenance schedules
  - High availability environments
  - OLAP/DSS environments
  - High I/O throughput requirements
  - Flexible disk striping for enhanced availability or performance
  - Partition transparency

Partitioned Indexes

- Also new to Oracle8
- Partitioning involves the *physical* separation of table or index data into multiple pieces or partitions, all of which share a common *logical* structure
- Very powerful for VLDB, OLAP, and large-scale OLTP environments - especially time-series data
- *Partition independence* also potentially reduces maintenance time and down time resulting from system failure recovery

Partitioned Indexes (cont.)

- An index can be range partitioned *unless*:
  - It is a cluster index
  - It is defined on a clustered table

- Oracle supports three types of partitioned indexes:
  - Local Prefixed
  - Local Non-Prefixed
  - Global Prefixed

Local Partitioned Indexes

- A local index is defined as an index that is *equi-partitioned* with the underlying base table, i.e., all keys in a given index partition refer only to rows stored in the single related table partition

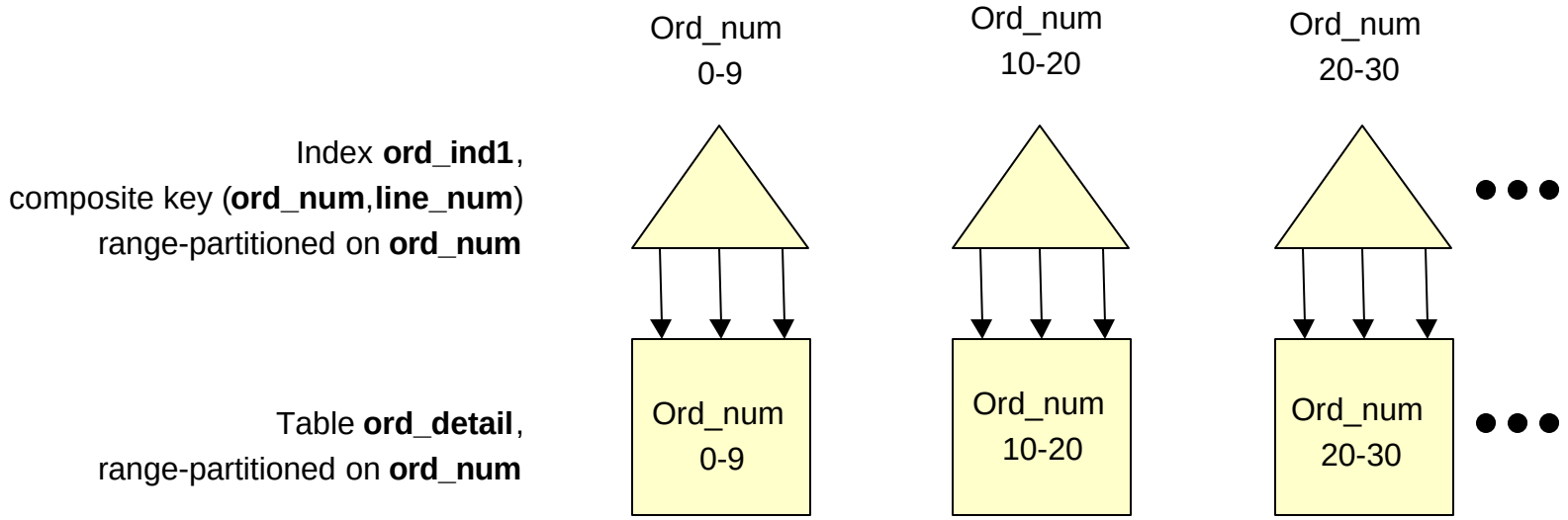
- Local index partitions are automatically maintained as table partitions are inserted, dropped or split

Global Partitioned Indexes

- A global partitioned index is defined as an index in which the keys in a given index partition may refer to rows in *more than one* underlying table partition
- Generally not equi-partitioned with the table
- Global partitioned indexes offer better performance via index partition “pruning”
- In Oracle, global indexes *must* be prefixed

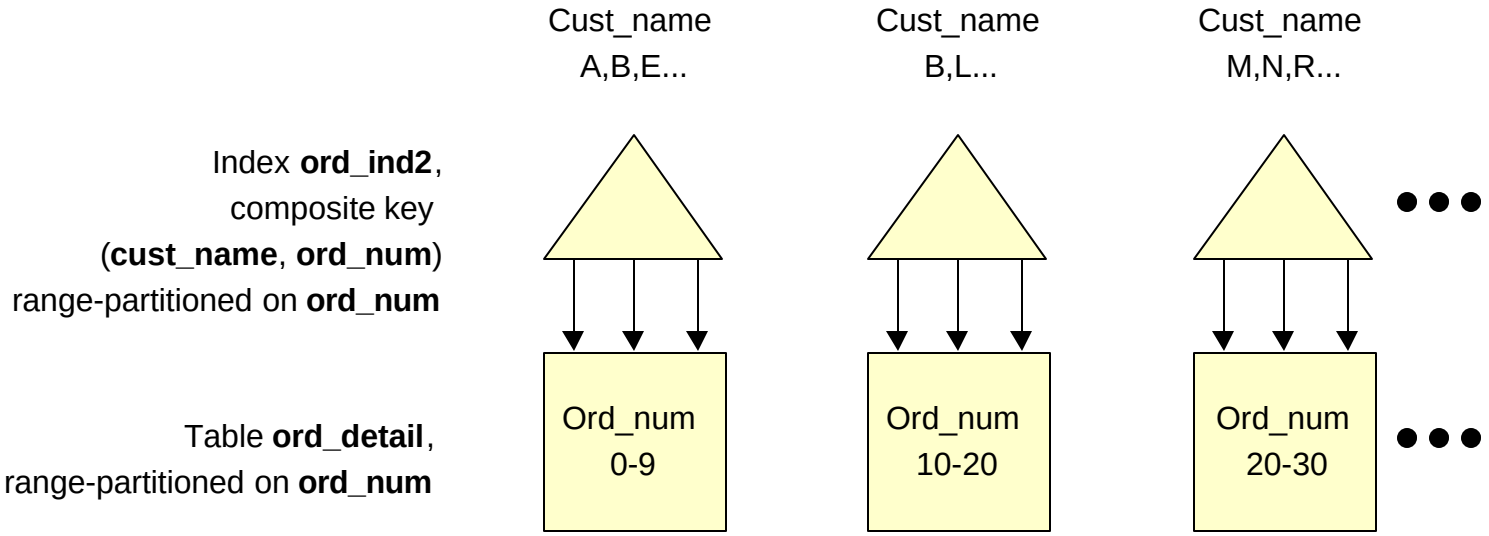
Local Prefixed Indexes

- A partitioned index is said to be *local prefixed* if it is partitioned based on the value of the *left-most* column(s) in the key



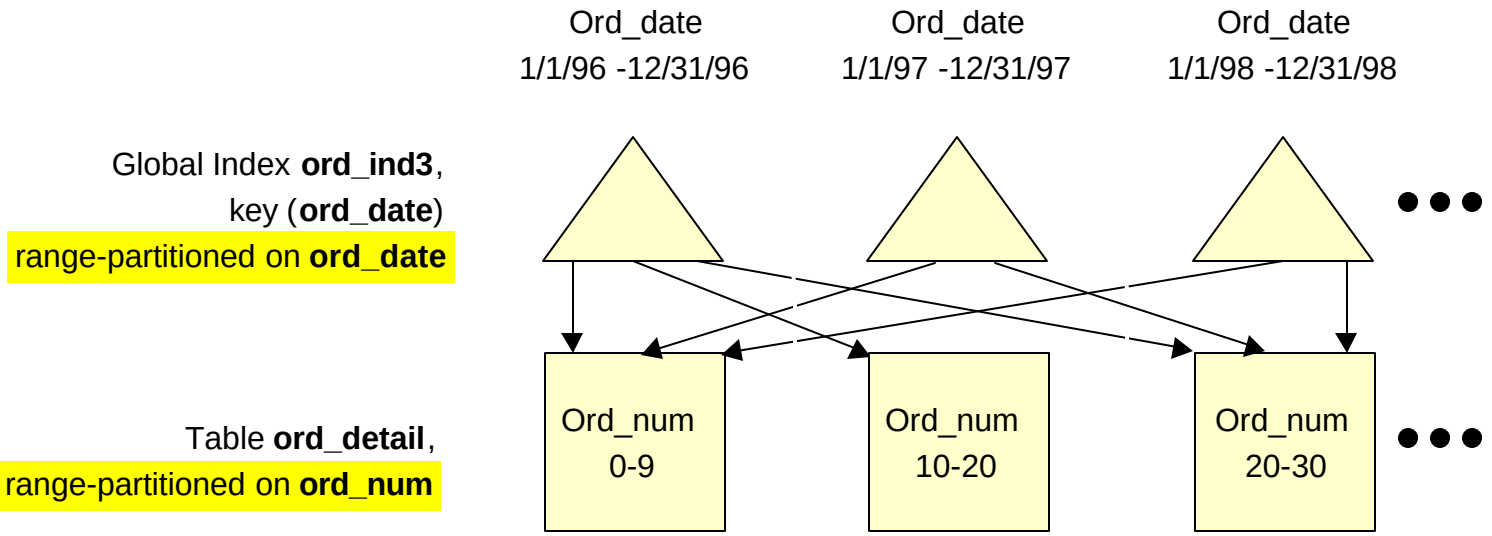
Local Non-Prefixed Indexes

- A partitioned index is said to be *local non-prefixed* if it is partitioned based on the value of any column(s) *other than* the left-most index column(s)



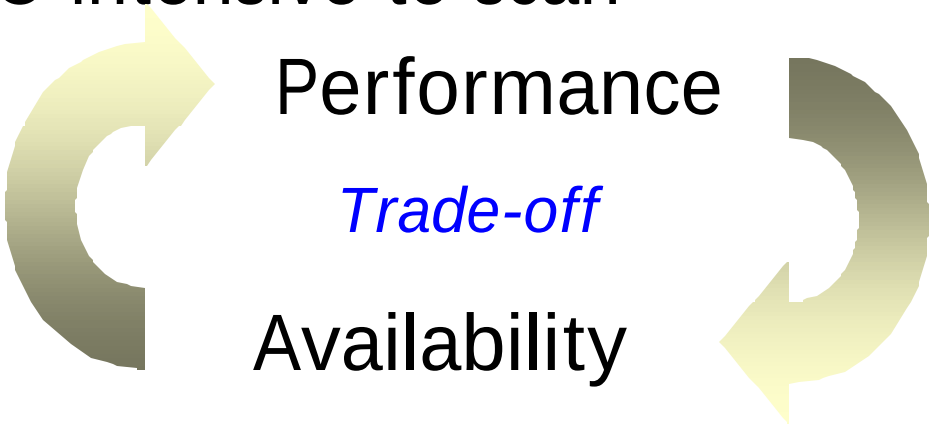
Global Prefixed Indexes

- A partitioned index is said to be *global prefixed* if it is partitioned based on the value of the *left-most* column(s) in the key, which *differs* from the table partition key



Local vs. Global Partitioned Indexes

- Global indexes can offer performance benefits as a result of partition pruning but they can potentially reduce availability by preventing partition-level maintenance operations
- Conversely, local indexes improve maintainability and availability but can be more I/O intensive to scan



Parallel Query

- Was first offered in late version 7
- Allows maximal usage of multiple CPUs
- Best if table and indexes are partitioned
- Need to set at database, table or index level
- Can be controlled at very fine level in Oracle8i

Parallel Query

- In Oracle7
  - PARALLEL_MAX_SERVERS
  - PARALLEL_MIN_SERVERS
  - PARALLEL_DEFAULT_MAX_SCANS
  - PARALLEL_DEFAULT_SCANSIZE
  - PARALLEL_MIN_PERCENT
- In Oracle8
  - OPTIMIZER_PERCENT_PARALLEL
  - PARALLEL_SERVER_IDLE_TIME
  - 2 UNDOCUMENTED PARAMETERS

Parallel Query

- In Oracle8i
  - PARALLEL_ADAPTIVE_MULTI_USER
  - PARALLEL_AUTOMATIC_TUNING
  - PARALLEL_BROADCAST_ENABLED
  - PARALLEL_EXECUTION_MESSAGE_SIZE
  - PARALLEL_THREADS_PER_CPU
  - 11 UNDOCUMENTED PARAMETERS

Multiple Buffer Pools

- You can use the **KEEP** buffer pool to maintain objects in the buffer cache
- You can use the **RECYCLED** buffer pool to prevent objects from consuming unnecessary space in the buffer cache
- Once an object is assigned to a buffer pool, all blocks of that object are placed in the buffer pool
- Different buffer pools can be assigned for different partitions of an object

Assigning Different Buffer Pools to the Partitions of a Table

- The newest partitions of a table will be placed in the KEEP buffer cache to ensure that the blocks remain in memory
- The size of the KEEP buffer pool will be defined to be big enough to hold all partition blocks in memory
- This will reduce the number of I/O operations performed to access the partitions blocks
- The oldest partitions, which are almost never used, will be placed in the RECYCLED buffer cache
- The syntax to assign the table partitions to a specific buffer pool is ... Storage (... buffer_pool keep ...)

Outlines

- Outlines are generally used to prevent changes in the database environment from affecting the performance characteristics of the application
- Outlines enable you to force execution plans by storing the old behavior in the database by use of “stealth hints”
- An Outline consists of a set of hints that is similar to the optimizer’s generated execution plan of the SQL statement
- Outlines can be enabled at the system level or the session level by setting parameter USE_STORED_OUTLINES

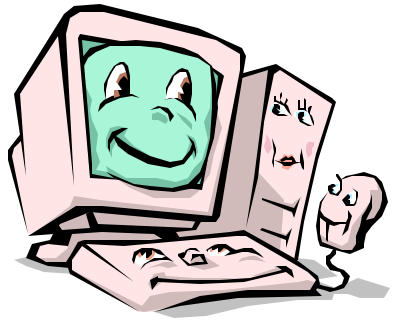
Using Outlines to Change Execution Plans of a Statement

- Drop the index you do not want to use. The statement now moves to the new index
- Create an outline for the statement by using the “create outline for select ...” statement
- This adds the /*+ index(....) */ hint to the outline
- Recreate the index; other statements need it
- ALTER SYSTEM SET USE_STORED_OUTLINES=TRUE command must be used to force the system to use outlines, also set USE_STORED_OUTLINES initialization parameter

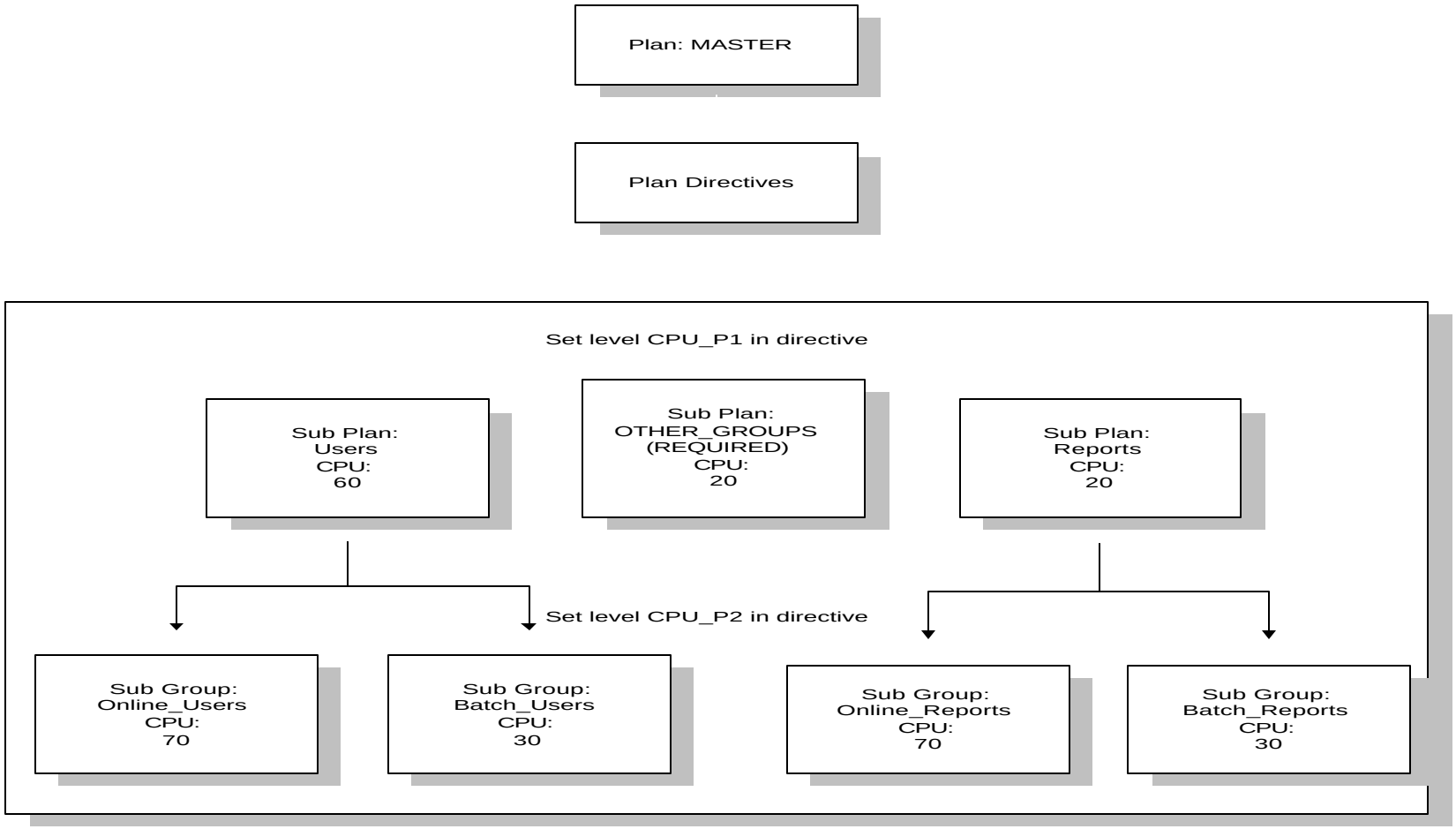
Resource Groups

Managing CPU Utilization in Oracle8i

- A database can have many users
- The users level of importance will vary
- CEO should get data faster than clerk
- RESOURCE PLANS and GROUPS allow this in ORACLE8i



Resource Groups



Resource Groups

Package Call	Step
dbms_resource_manager_privs.grant_system_privilege	Grant Privileges to Plan Administrator
dbms_resource_manager.create_pending_area();	Create Pending Area
Manually	Create Plan Map

Resource Groups

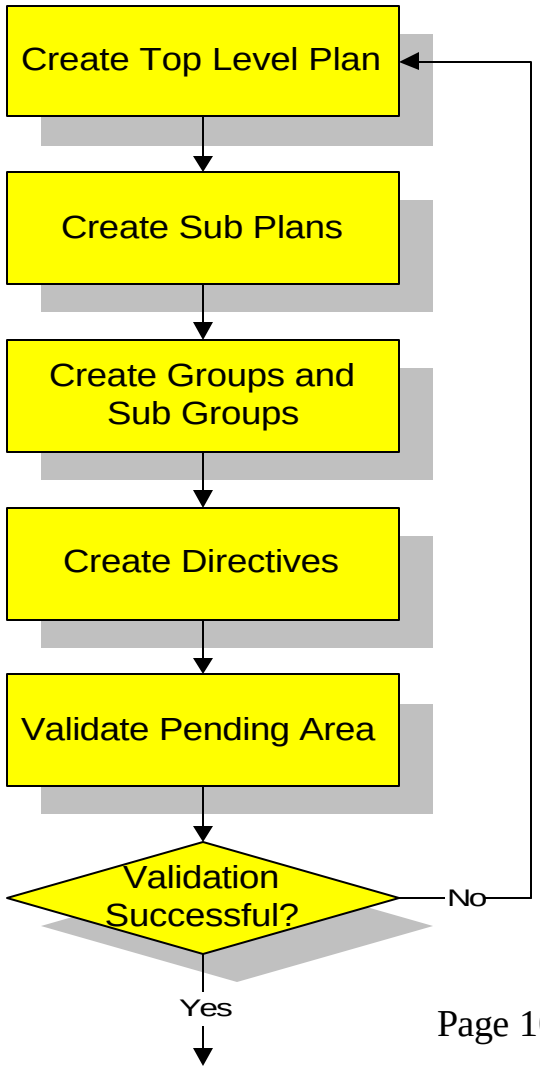
dbms_resource_manager.create_plan

dbms_resource_manager.create_plan

dbms_resource_manager.create_consumer_group

dbms_resource_manager.create_plan_directive

dbms_resource_manager.validate_pending_area



Resource Groups

dbms_resource_manager.submit_pending_area

Submit Pending Area

dbms_resource_manager_privs.grant_switch_consumer_group
dbms_resource_manager.set_initial_consumer_group
dbms_resource_manager.switch_consumer_group_for_user
dbms_resource_manager.switch_consumer_group_for_sess

Grant Groups to Users

Resource Groups

Example Script For Resource Group Creation

```
set echo on
spool test_resource_plan.doc
-- Grant system privilege to plan administrator
EXEC

dbms_resource_manager_privs.grant_system_privileg
e( 'SYSTEM','ADMINISTER_RESOURCE_MANAGER',TRUE);
--connect to plan administrator
CONNECT system/system_test@ortest1.world
-- Create Plan Pending Area
EXEC
dbms_resource_manager.create_pending_area();
```

Resource Groups

```
-- Create plan
EXECUTE dbms_resource_manager.create_plan('MASTER',
   'Example Resource Plan','EMPHASIS');
EXECUTE dbms_resource_manager.create_plan('USERS',
   'Example Resource Sub Plan','EMPHASIS');
EXECUTE dbms_resource_manager.create_plan('REPORTS',
   'Example Resource Sub Plan','EMPHASIS');
--Create tiers of groups in plan
EXECUTE dbms_resource_manager.create_consumer_group(
   'ONLINE_USERS','3rd level group','ROUND-ROBIN');
EXECUTE dbms_resource_manager.create_consumer_group(
   'BATCH_USERS','3rd level group','ROUND-ROBIN');
EXECUTE dbms_resource_manager.create_consumer_group(
   'ONLINE_REPORTS','2rd level group','ROUND-ROBIN');
EXECUTE dbms_resource_manager.create_consumer_group(
   'BATCH_REPORTS','2rd level group','ROUND-ROBIN');
```

Resource Groups

```
-- Create plan directives
EXECUTE dbms_resource_manager.create_plan_directive(
   'MASTER', 'USERS',0,60,0,0,0,0,0,0,NULL);
EXECUTE dbms_resource_manager.create_plan_directive(
   'MASTER', 'REPORTS',0,20,0,0,0,0,0,0,NULL);
EXECUTE dbms_resource_manager.create_plan_directive(
   'MASTER', 'OTHER_GROUPS',0,20,0,0,0,0,0,0,NULL);
EXECUTE dbms_resource_manager.create_plan_directive(
   'USERS',  'ONLINE_USERS',0,0,70,0,0,0,0,0,NULL);
EXECUTE dbms_resource_manager.create_plan_directive(
   'USERS',  'BATCH_USERS',0,0,30,0,0,0,0,0,NULL);
EXECUTE dbms_resource_manager.create_plan_directive(
   'REPORTS', 'ONLINE_REPORTS',0,0,70,0,0,0,0,0,NULL);
EXECUTE dbms_resource_manager.create_plan_directive(
   'REPORTS', 'BATCH_REPORTS', 0,0,30,0,0,0,0,0,NULL);
```

Resource Groups

```
-- Verify Plan
EXECUTE
  dbms_resource_manager.validate_pending_area;
-- Submit Plan
EXECUTE
  dbms_resource_manager.submit_pending_area;
spool off
set echo off
```

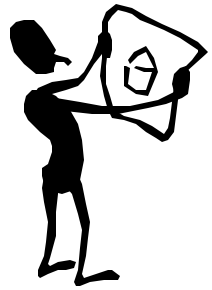
Resource Groups

- **DBMS_RESOURCE_MANAGER Package**
  - CREATE_PLAN
  - UPDATE_PLAN
  - DELETE_PLAN
  - DELETE_PLAN_CASCADE
  - CREATE_CONSUMER_GROUP
  - UPDATE_CONSUMER_GROUP
  - DELETE_CONSUMER_GROUP

Resource Groups

- **DBMS_RESOURCE_MANAGER Package**
  - CREATE_PENDING_AREA
  - VALIDATE_PENDING_AREA
  - CLEAR_PENDING_AREA
  - SUBMIT_PENDING_AREA
  - SET_INITIAL_CONSUMER_AREA
  - SWITCH_CONSUMER_GROUP_FOR_SESS
  - SWITCH_CONSUMER_GROUP_FOR_USER

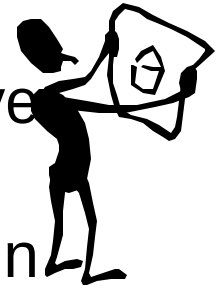
Resource Groups



- **To Be Valid A PLAN Must:**
  - No plan schema may contain any loops
  - All plans and consumer groups referred to by plan directives must exist
  - All plans must have plan directives that refer to either plans or consumer groups
  - All percentages in any given level must not add up to greater than 100 for the emphasis resource allocation method.
  - No plan may be deleted that is currently being used as a top plan by an active instance
  - For Oracle8i, the plan directive parameter, parallel_degree_limit_p1, may only appear in plan directives that refer to consumer groups (i.e., not at subplans)

Resource Groups

- There cannot be more than 32 plan directives coming from any given plan (i.e., no plan can have more than 32 children).
- There cannot be more than 32 consumer groups in any active plan schema.
- Plans and consumer groups use the same namespace; therefore, no plan can have the same name as any consumer group.
- There must be a plan directive for OTHER_GROUPS somewhere in any active plan schema.This ensures that a session not covered by the currently active plan is allocated resources as specified by the OTHER_GROUPS directive.



Resource Groups

- **DBMS_RESOURCE_MANAGER_PRIVS Package**
  - GRANT_SYSTEM_PRIVILEGE
  - REVOKE_SYSTEM_PRIVILEGE
  - GRANT_SWITCH_CONSUMER_GROUP
  - REVOKE_SWITCH_CONSUMER_GROUP

Resource Groups

- **Usage Notes For GRANT RESOURCE Privileges:**
- If you grant permission to switch to a particular consumer group to a user, then that user can immediately switch their current consumer group to the new group
- If you grant permission to switch to a particular consumer group to a role, then any users who have been granted that role and have enabled that role can immediately switch their current consumer group to the new group
- If you grant permission to switch to a particular consumer group to PUBLIC, then any user can switch to that group.

Resource Groups

- If the grant_option parameter is TRUE, then users granted switch privilege for the consumer group may also grant switch privileges for that consumer group to others

- In order to set the initial consumer group of a user, you must grant the switch privilege for that group to the user

Resource Groups

- **Usage Notes for REVOKE RESOURCE Privileges**:
- If you revoke a user's switch privilege for a particular consumer group, then any subsequent attempts by that user to switch to that consumer group will fail.
- If you revoke the initial consumer group from a user, then that user will automatically be part of the DEFAULT_CONSUMER_GROUP (OTHERS) consumer group when logging in.

Resource Groups

- If you revoke the switch privilege for a consumer group from a role, then any users who only had switch privilege for the consumer group via that role will not be subsequently able to switch to that consumer group.
- If you revoke the switch privilege for a consumer group from PUBLIC, then any users who could previously only use the consumer group via PUBLIC will not be subsequently able to switch to that consumer group.

Summary

- There are many options for tuning, even when you can't touch the code
- If you begin with a default installation the performance gains can be substantial
- Proper memory, sort area, shared pool and process tuning is critical
- Proper indexing is also critical and can lead to substantial performance gains
- New indexes, partitioning and parallel query also boost performance