New York

Metro Area Oracle User Group

Date: Tuesday, October 2, 2007

Time: 10:30 PM – 11:30 PM

Venue: New Yorker Hotel, Gramercy Park

New York, NY

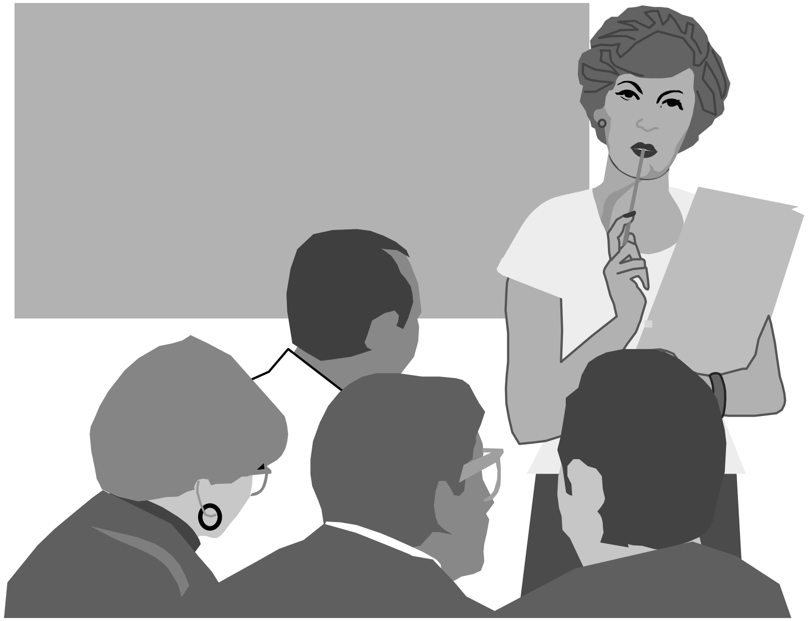
Oracle Database 11g: Top Features for DBAs

Daniel T. Liu
Principal Solution Architect

The following 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 remain at the sole discretion of Oracle.

Agenda

- **Oracle 11g Database Overview**
- **Top New Features**
- **Summary**
- **Q & A**



Overview

Oracle Database

30 Years of Sustained Innovation

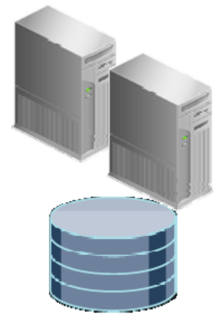
Database Vault
Transparent Data Encryption
Grid Computing
Self Managing Database
XML Database
Oracle Data Guard
Real Application Clusters
Flashback Query
Virtual Private Database
Built in Java VM
Partitioning Support
Built in Messaging
Object Relational Support
Multimedia Support
Data Warehousing Optimizations
Parallel Operations
Distributed SQL & Transaction Support
Cluster and MPP Support
Multi-version Read Consistency
Client/Server Support
Platform Portability
Commercial SQL Implementation

1977

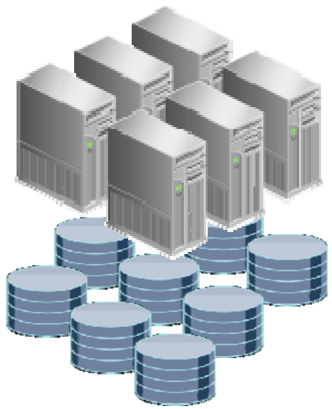
2007

Enterprise Grid Computing

SMP Dominance



RAC Clusters for Availability



Grids of low cost hardware and storage

ORACLE 8i INTERNET

ORACLE 9i INTERNET

ORACLE DATABASE 10g

Database Background Processes

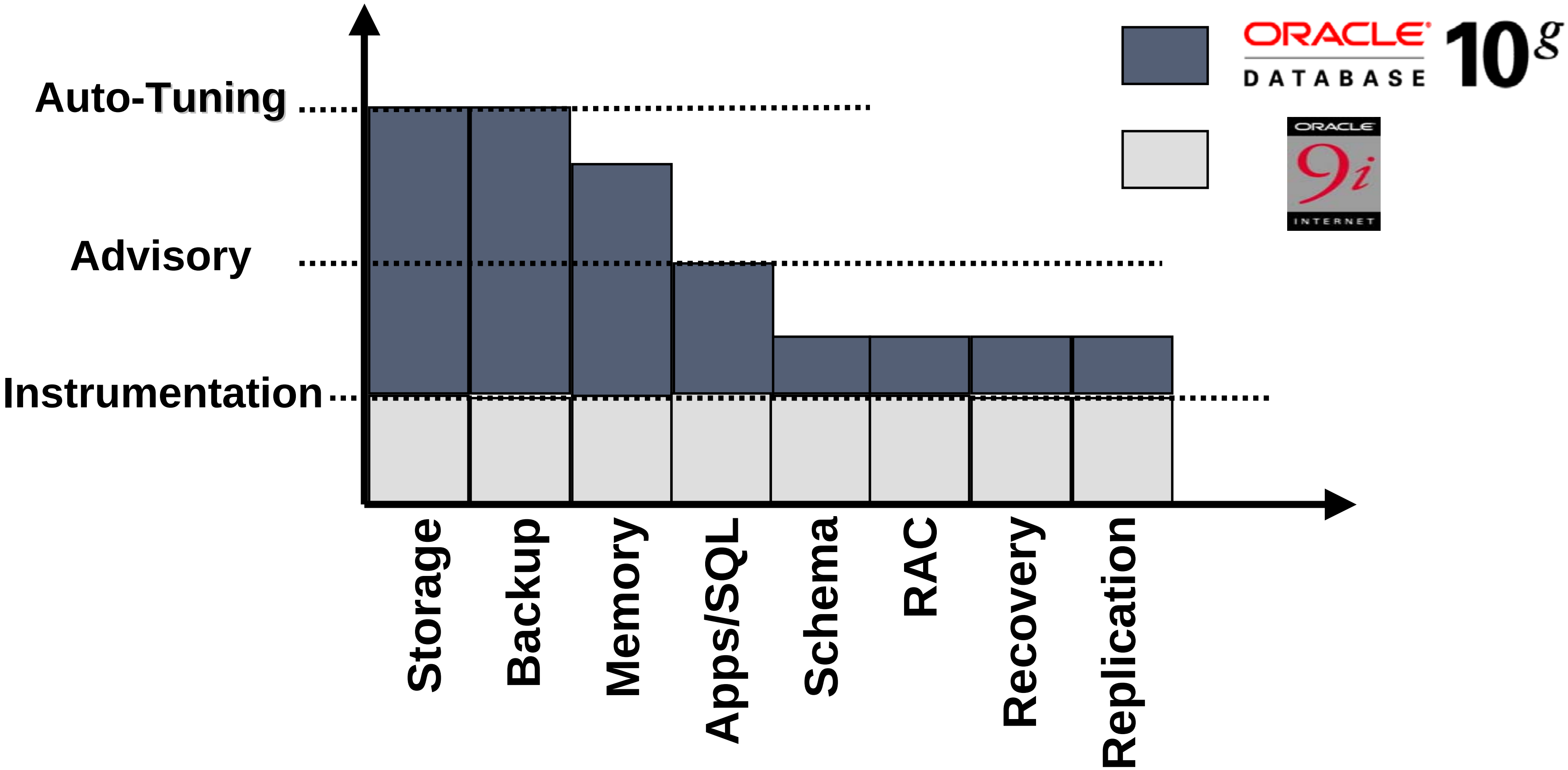
- In Version 6 we had 5 Basic Background Processes
  - PMON
  - SMON
  - DBWR
  - LGWR
  - ARCH
- In Oracle 11g we have > 50 background processes!

Binary Sizes

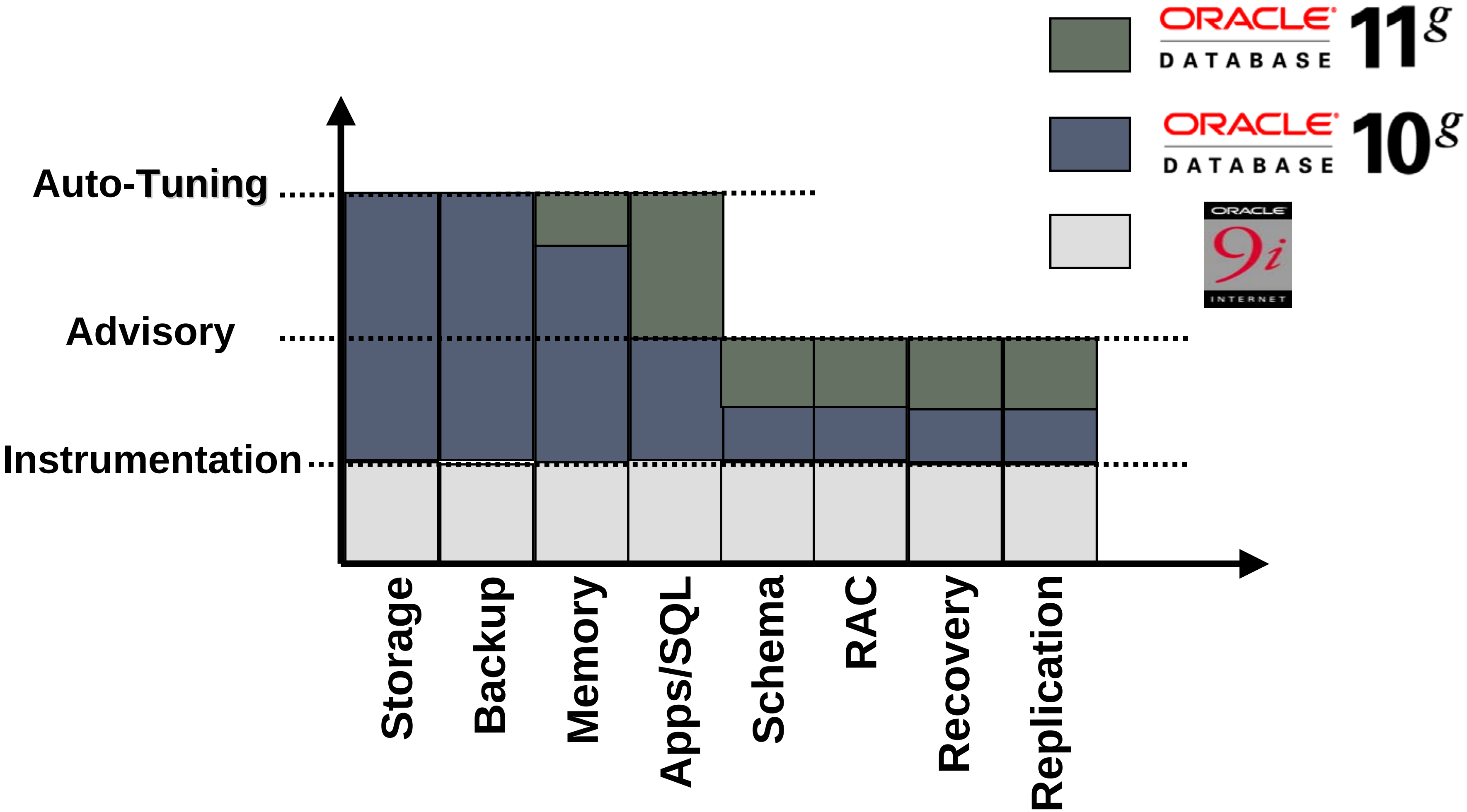
Binary Size

- Oracle Executable Binary Sizes For 32-bit Linux
  - 8.1.7.4 26M
  - 9.0.1 37M
  - 9.2.0.4 49M
  - 10.2.0.3 91M
  - 11.1.0.4 124M

Self Managing Database

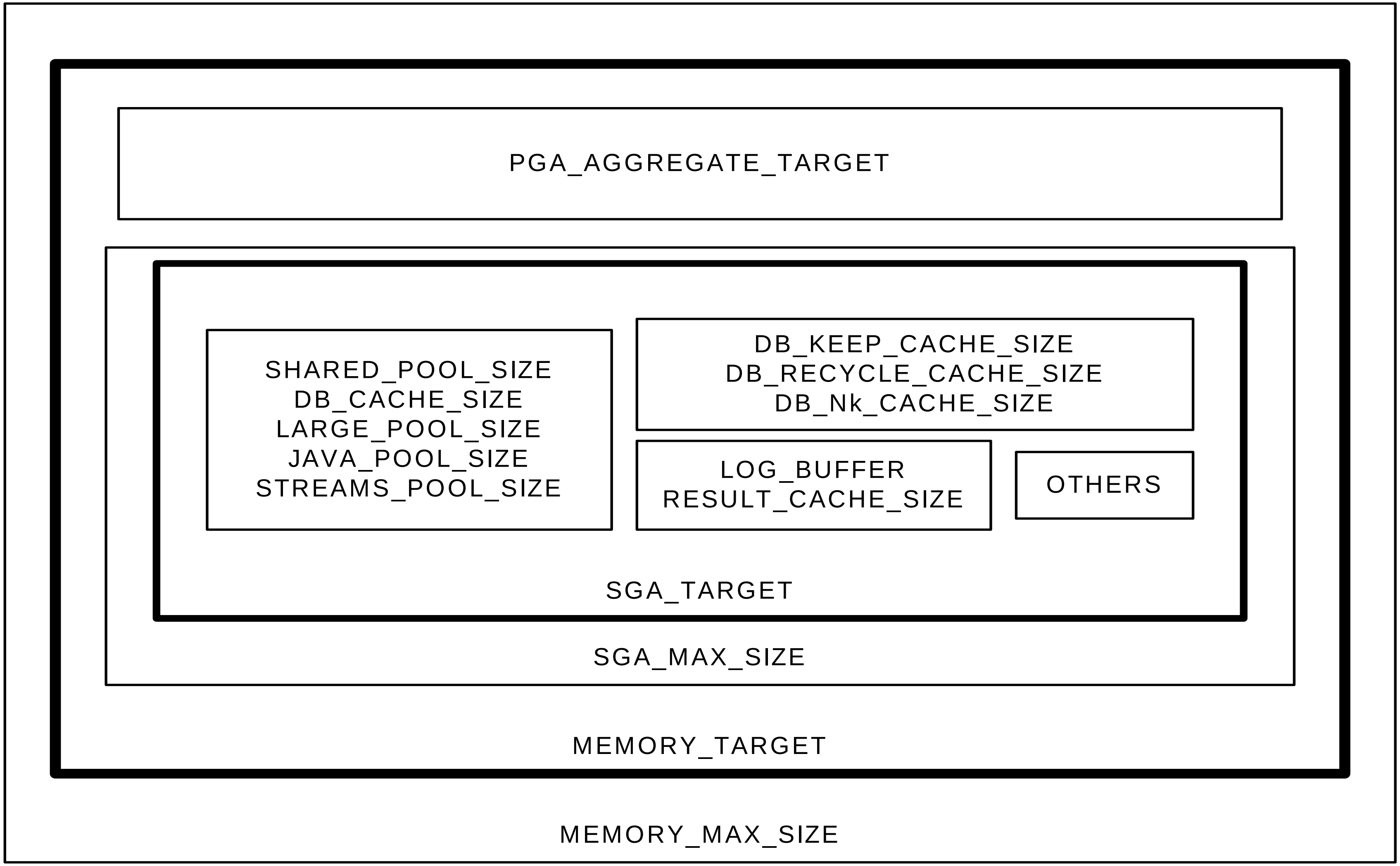


Self Managing Database



Automatic Memory Management

Automatic Memory Management



Automatic Diagnostic Repository

Automatic Diagnostic Repository

- Automatic Diagnostic Repository (ADR) is a file-based repository for database diagnostic data such as traces, incident dumps and packages, the alert log, Health Monitor reports, core dumps, and more
- The default location for all trace information is defined by DIAGNOSTIC_DIST

ADR Repository File Structure

- In Oracle Database 11g, the directory structure as follow:

```
/u01/app/oracle/diag/rdbms/DatabaseName/
  InstanceName/trace
               /alert
               /cdump
               /incident
               /incpkg
               /hm
               /metadata
               /(others)
```

ADR Repository File Structure

- The **TRACE** directory contains text alert log and background/foreground process trace files.
- The **ALERT** directory contains an XML version of the alert log.
- The **CDUMP** directory contains all core dump files.
- The **INCIDENT** directory contains multiple subdirectories, where each subdirectory is named for a particular incident, and where each contains dumps pertaining only to that incident.
- The **INCPKG** directory contains a collection of metadata to be ready to upload the data to Oracle Support Services.
- The **HM** directory contains the checker run reports generated by the Health Monitor.
- The **METADATA** directory contains important files for the repository itself.

Invisible Index

Invisible Index

- An invisible index is an index that is ignored by the optimizer unless you explicitly set the OPTIMIZER_USE_INVISIBLE_INDEXES initialization parameter to TRUE at the session or system level. The default value for this parameter is FALSE.
- Making an index invisible is an alternative to making it unusable or dropping it. Using invisible indexes, you can do the following:
  - Test the removal of an index before dropping it.
  - Use temporary index structures for certain operations or modules of an application without affecting the overall application.

Invisible Index

- Here are a few examples:

```
SQL> alter index emp_id_idx invisible;
SQL> alter index emp_id_idx visible;
SQL> create index emp_id_idx on emp
  (emp_id) invisible;
```

SQL Query Result Cache

Data Warehouse Workload

- Analyze data across large data sets
  - reporting
  - forecasting – trend analysis
  - data mining
- Use parallel execution for good performance
- Result
  - very IO intensive workload – direct reads from disk
  - memory is less important
    - mostly execution memory

Data Warehouse Query Example

```
select p.prod_category
,      sum(s.amount_sold) revenue
from products p
,    sales    s
where s.prod_id = p.prod_id
and   s.time_id
  between to_date('01-JAN-2006','dd-MON-yyyy')
  and     to_date('31-DEC-2006','dd-MON-yyyy')
group by rollup (p.prod_category)
```

- accesses very many rows
- returns few rows

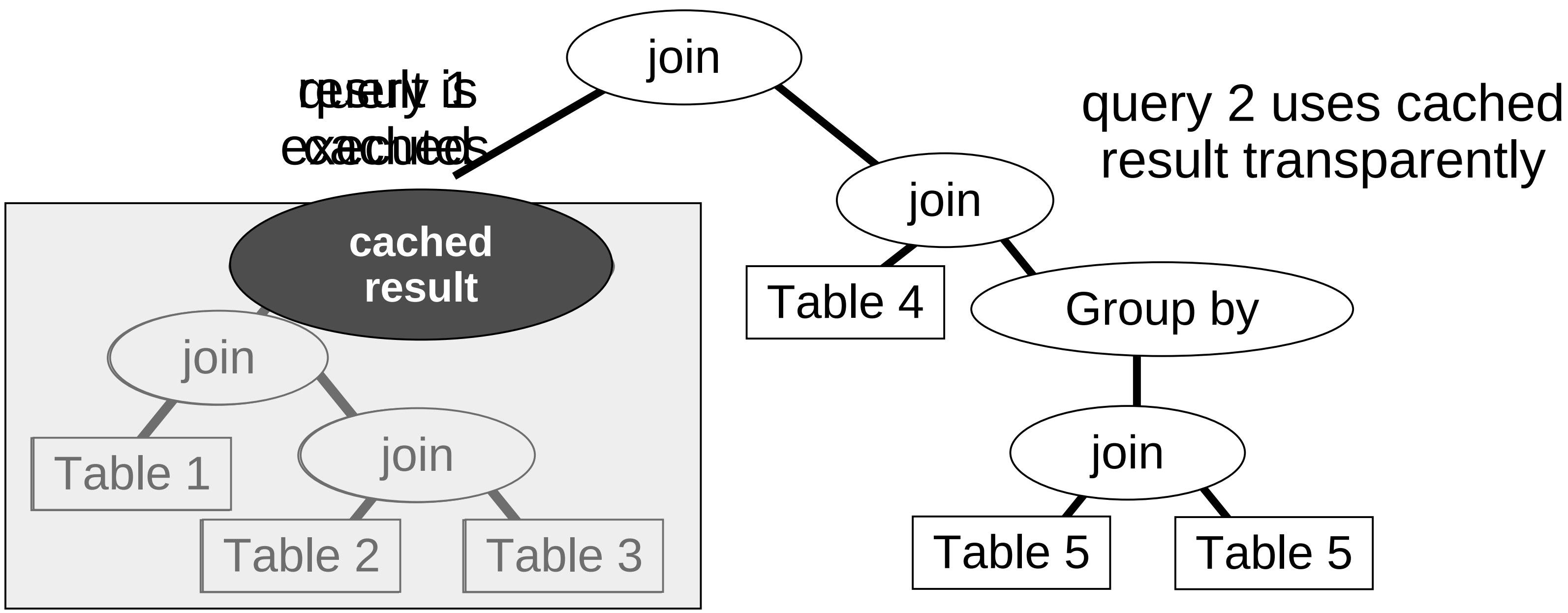
Data Warehouse Configuration Sizing

- Critical success factors
  - IO throughput
    - number of physical disks
    - number of channels to disks
  - CPU power
- Everything else follows
  - Storage capacity (500GB – 1TB common)
    - use surplus for high availability and ILM
  - Memory capacity (4GB/CPU is “standard”)
    - use surplus for... **RESULT CACHE**

ORACLE

SQL Query Result Cache Benefits

- Caches results of queries, query blocks, or pl/sql function calls
- Read consistency is enforced
  - DML/DDL against dependent database objects invalidates cache
- Bind variables parameterize cached result with variable values



SQL Query Result Cache Enabling

- result_cache_mode initialization parameter
  - MANUAL, use hints to populate and use
  - FORCE, queries will use cache without hint
- result_cache_max_size initialization parameter
  - default is dependent on other memory settings

    (0.25% of memory_target or 0.5% of sga_target or 1% of shared_pool_size)
  - 0 disables result cache
  - never >75% of shared pool (built-in restriction)
- /*+ RESULT_CACHE */ hint in queries

SQL Query Result Cache Example

- Use RESULT_CACHE hint

```
select /*+ RESULT_CACHE */ p.prod_category
,       sum(s.amount_sold) revenue
from products p
,    sales    s
where s.prod_id = p.prod_id
and   s.time_id
  between to_date('01-JAN-2006','dd-MON-yyyy')
  and     to_date('31-DEC-2006','dd-MON-yyyy')
group by rollup (p.prod_category)
```

SQL Query Result Cache Example

- Execution plan fragment

```
-----------------------------------------------------------------
| Id  | Operation                     | Name                      |
-----------------------------------------------------------------
|   0 | SELECT STATEMENT              |                           |
|   1 |  RESULT CACHE                 | fz6cm4jbpcwh48wcyk60m7qypu |
|   2 |   SORT GROUP BY ROLLUP        |                           |
|*  3 |    HASH JOIN                  |                           |
|   4 |     PARTITION RANGE ITERATOR|                           |
|*  5 |      TABLE ACCESS FULL        | SALES                     |
|   6 |     VIEW                      | index$_join$_001          |
|*  7 |      HASH JOIN                |                           |
|   8 |       INDEX FAST FULL SCAN    | PRODUCTS_PK               |
|   9 |       INDEX FAST FULL SCAN    | PRODUCTS_PROD_CAT_IX      |
-----------------------------------------------------------------
```

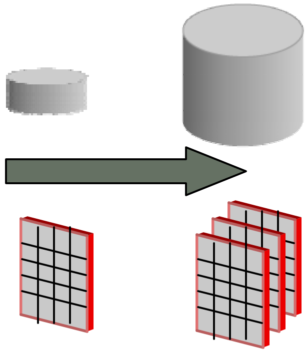
SQL Query Result Cache Opportunity

- Depends... based on
  - query repetitiveness
  - query execution times
  - DML activity (cache invalidation frequency)
- Remember data warehouse workload
  - query may run 30 minutes
  - query may return 5 rows
  - query served from result cache would take split second

Oracle Database 11g – Flashback Data Archive

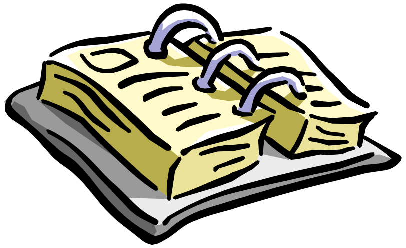
Data History and Retention

- Data retention and change control requirements are growing
  - Regulatory oversight and Compliance
    - Sarbanes-Oxley, HIPAA, Basel-II, Internal Audit
  - Business needs
    - Extract “temporal” dimension of data
    - Understand past behavior and manage customer relationships profitably
- Failure to maintain appropriate history & retention is expensive
  - Legal risks
  - Loss of Reputation
- Current approaches to manage historical data are inefficient and often ineffective



Data History and Retention - Requirements

- Historical data needs to be secure and tamper proof
  - Unauthorized users should not be able to access historical data
  - No one should be able to update historical data
- Easily accessible from existing applications
  - Seamless access
  - Should not require special interfaces or application changes
- Minimal performance overhead
- Optimal Storage footprint
  - Historical data volume can easily grow into hundreds of terabytes
- Easy to set up historical data capture and configure retention policies



Managing Data History – Current Approaches

- Application or mid-tier level
  - Combines business logic and archive policies
  - Increases complexity
  - No centralized management
  - Data integrity issues if underlying data is updated directly
- Database level
  - Enabled using Triggers
  - Significant performance and maintenance overhead
- External or Third-party
  - Mine redo logs
  - History stored in separate database
  - Cannot seamlessly query OLTP and history data
- None of the above approaches meet all customer requirements
  - Customers are therefore forced to make significant compromises

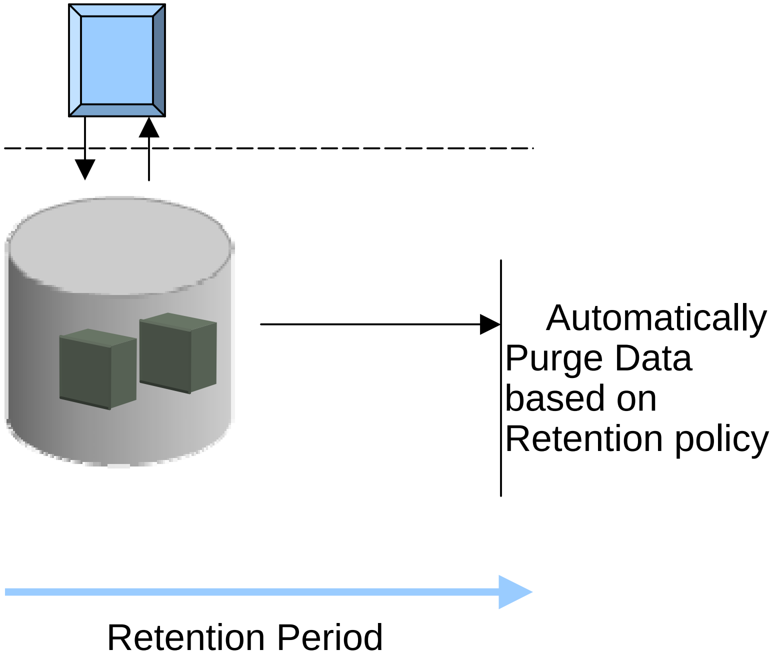
Introducing Flashback Data Archive

- New feature in Oracle Database 11g
- Transparently tracks historical changes to all Oracle data in a highly secure and efficient manner
  - Historical data is stored in the database and can be retained for as long as you want
  - Special kernel optimizations to minimize performance overhead of capturing historical data
  - Historical data is stored in compressed form to minimize storage requirements
  - Automatically prevents end users from changing historical data
- Seamless access to archived historical data
  - Using “AS OF” SQL construct

```
select * from product_information AS OF TIMESTAMP
'02-MAY-05 12.00 AM‘ where product_id = 3060
```

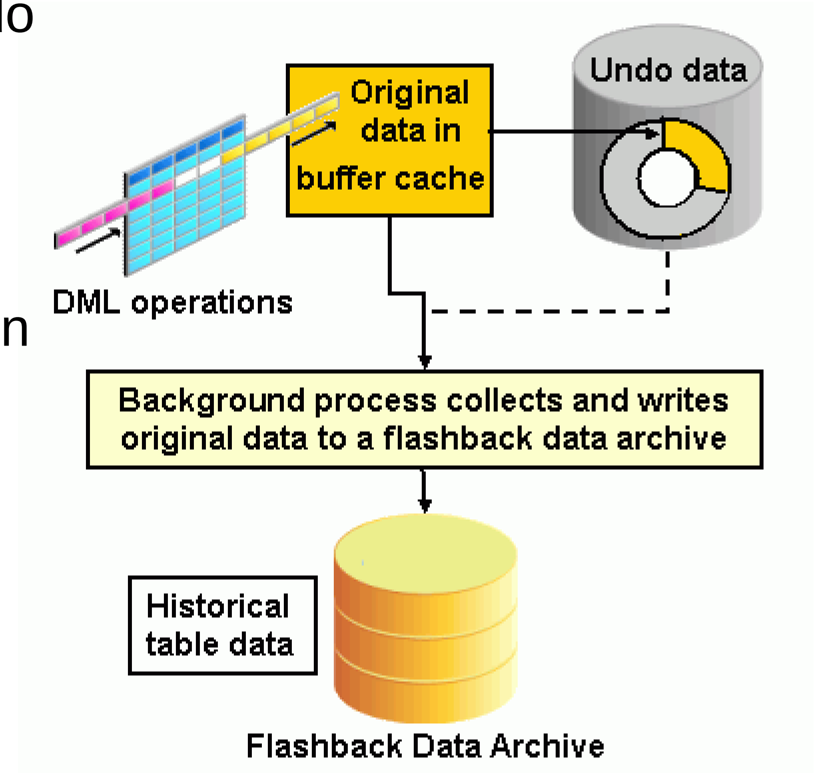
Introducing Flashback Data Archive

- Extremely easy to set up
  - Enable history capture in minutes!
- Completely transparent to applications
- Centralized and automatic management
  - Policy-based
  - Multiple tables can share same Retention and Purge policies
  - Automatic purge of aged history



How Does Flashback Data Archive Work?

- Primary source for history is the undo data
- History is stored in automatically created history tables inside the archive
- Transactions and its undo records on tracked tables marked for archival
  - Undo records not recycled until history is archived
- History is captured asynchronously by new background process (fbda)
  - Default capture interval is 5 minutes
  - Capture interval is self-tuned based on system activities
  - Process tries to maximize undo data reads from buffer cache for better performance
  - INSERTs do not generate history records



Flashback Data Archive And DDLs

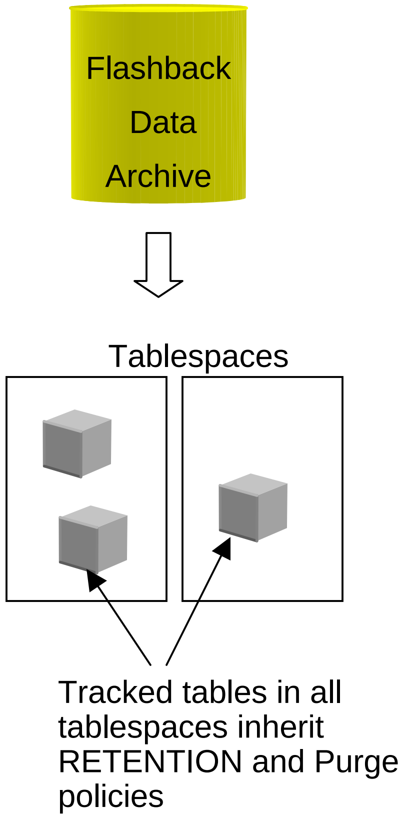
- Possible to add columns to tracked tables
- Automatically disallows any other DDL that invalidates history
  - Dropping and truncating a tables
  - Dropping or modifying a column
- Must disable archiving before performing any major changes
  - Disabling archiving discards already collected history
- Flashback Data Archive guarantees historical data capture and maintenance
  - Any operations that invalidates history or prevents historical capture will be disallowed

ORACLE

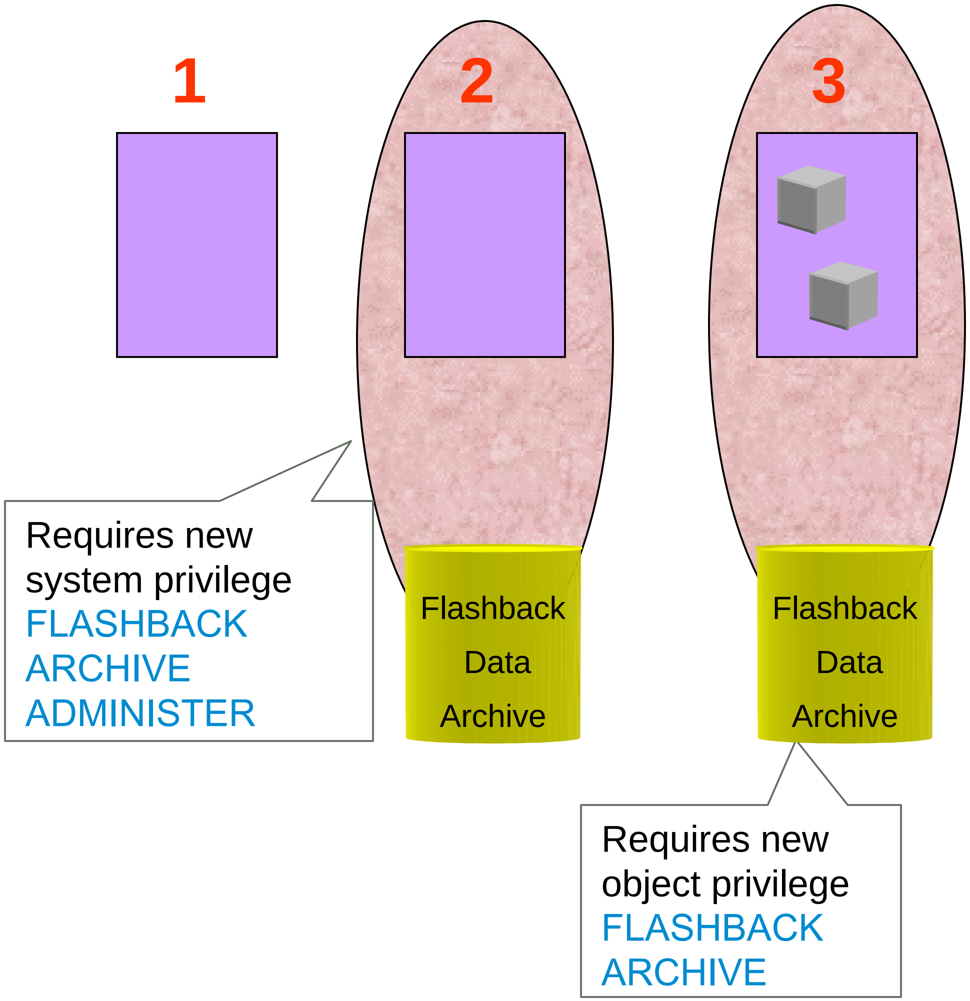
Historical Data Storage

- A new database object, flashback data archive, is a logical container for storing historical information
- Consists of one or more tablespaces
  - ‘QUOTA’ determines max amount of space a flashback data archive can use in each tablespace (default is Unlimited)
- Specify duration for retaining historical changes using ‘RETENTION’ parameter
- Tracks history for one or more tables
  - Tables should share the archiving characterstics
- Automatically purges aged-out historical data based on retention policy
- Create as many flashback data archives as needed
  - Group related tables by desired retention period

    - HIPAA requires all health transactions be maintained for 6 years



Creating Flashback Data Archive & Enable History Tracking



1. Create tablespace (ASSM is required)
2. Create a flashback data archive
   - Set the retention period

   CREATE FLASHBACK ARCHIVE fda1

   TABLESPACE tbs1

   RETENTION 5 YEAR;
3. Enable archiving on desired tables

   ALTER TABLE EMPLOYEES FLASHBACK ARCHIVE fda1;

Managing Flashback Data Archive

- Static data dictionary views
  - *_FLASHBACK_ARCHIVE - Displays information about Flashback Data Archives.
  - *_FLASHBACK_ARCHIVE_TS - Displays tablespaces of Flashback Data Archives.
  - *_FLASHBACK_ARCHIVE_TABLES - Displays information about tables that are enabled for flashback archiving.
- Alerts generated when flashback data archive is 90% full
- Automatically purges historical data after expiration of specified retention period
- Supports ad-hoc purge by administrators (privileged operation)
  - ALTER FLASHBACK ARCHIVE fla1 PURGE BEFORE TIMESTAMP (SYSTIMESTAMP - INTERVAL '1' DAY);

Managing Flashback Data Archive

- SYS_FBA_HIST_* - Internal History Table
  - Replica of tracked table with additional timestamp columns
  - Partitioned for faster performance
  - No modifications allowed to internal partitions
  - Compression reduces disk space required
  - No out-of-box indexes
  - Support for copying primary key indexes from tracked table in later releases (TBD)
- Applications don't need to access internal tables directly
  - Use 'AS OF' to seamlessly query history

Oracle Database 11g

Advanced Compression Option

ORACLE

Challenges

- Explosion in data volume managed by Enterprises
  - Government regulations (Sarbanes-Oxley, HIPPA, etc)
  - User generated content (Web 2.0)
- IT managers must support larger volumes of data with limited technology budgets
  - Need to optimize storage consumption
  - Also maintain acceptable application performance
- Intelligent and efficient compression technology can help address these challenges

Introducing Advanced Compression Option

- Oracle Database 11g introduces a comprehensive set of compression capabilities
  - Structured/Relational data compression
  - Unstructured data compression
  - Compression for backup data
  - Network transport compression
- Reduces resource requirements and costs
  - Storage System
  - Network Bandwidth
  - Memory Usage

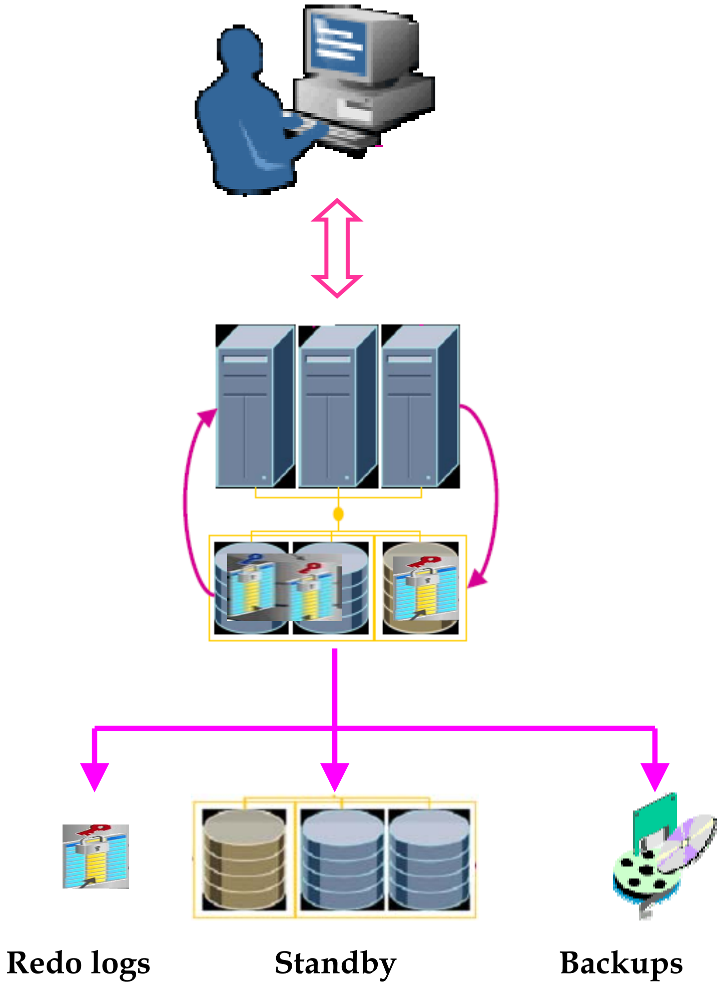


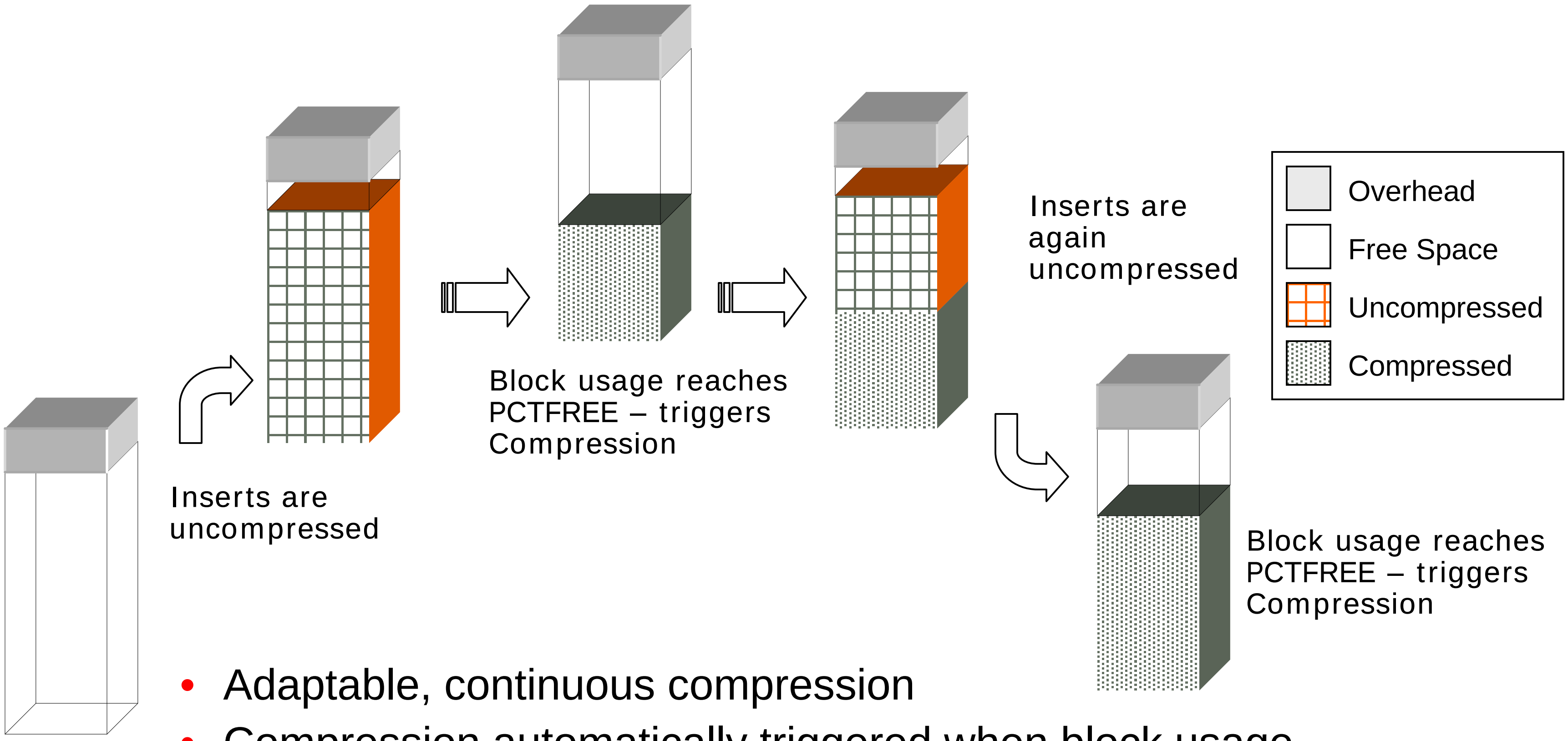
Table Compression

- Introduced in Oracle9i Release 2
  - Supports compression during bulk load operations (Direct Load, CTAS)
  - Data modified using conventional DML not compressed
- Optimized compression algorithm for relational data
- Improved performance for queries accessing large amounts of data
  - Fewer IOs
  - Buffer Cache efficiency
- Data is compressed at the database block level
  - Each block contains own compression metadata – improves IO efficiency
  - Local symbol table dynamically adapts to data changes
- Compression can be specified at either the table or partition levels
- Completely transparent to applications
- Noticeable impact on write performance

OLTP Table Compression

- Oracle Database 11g extends compression for OLTP data
  - Support for conventional DML Operations
    (INSERT, UPDATE, DELETE)
- New algorithm significantly reduces write overhead
  - Batched compression ensures no impact for most OLTP transactions
- No impact on reads
  - Reads may actually see improved performance due to fewer IOs and enhanced memory efficiency

OLTP Table Compression



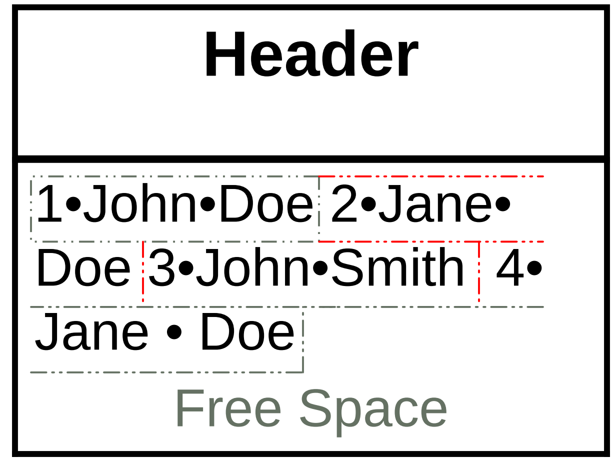
- Adaptable, continuous compression
- Compression automatically triggered when block usage reaches PCTFREE
- Compression eliminates holes created due to deletions and maximizes contiguous free space in block

OLTP Table Compression

Employee Table

ID	FIRST_NAME	LAST_NAME
1	John	Doe
2	Jane	Doe
3	John	Smith
4	Jane	Doe

Initially Uncompressed Block




```
INSERT INTO EMPLOYEE
    VALUES (5, ‘Jack’, ‘Smith’);
COMMIT;
```

OLTP Table Compression

Employee Table

ID	FIRST_NAME	LAST_NAME
1	John	Doe
2	Jane	Doe
3	John	Smith
4	Jane	Doe
5	Jack	S[illegible]

Compressed Block

Header

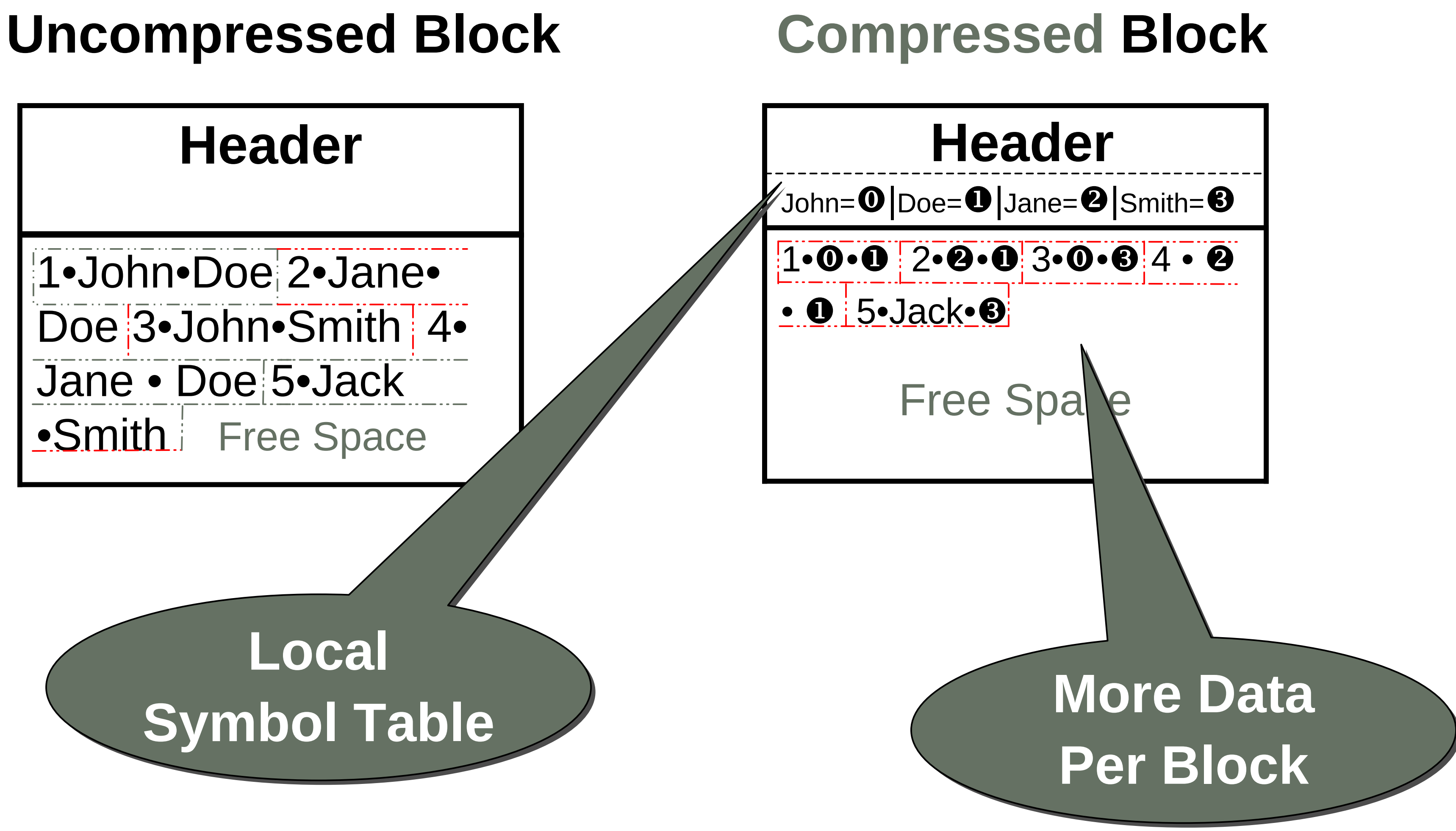
John=⓿|Doe=❶|Jane=❷|Smith=❸

1•⓿•❶ 2•❷•❶ 3•⓿•❸ 4 • ❷ • ❶ 5•Jack•❸

Free Space

Local Symbol Table

OLTP Table Compression



Using OLTP Table Compression

- Requires database compatibility level at 11.1 or greater
- New Syntax extends the ‘COMPRESS’ keyword
  - COMPRESS [FOR {ALL | DIRECT_LOAD} OPERATIONS]
  - DIRECT_LOAD (*DEFAULT)*
    - Refers to Bulk load operations from 10g and prior releases
  - ALL
    - OLTP + Direct loads
- Enable compression for new tables

```
CREATE TABLE t1 COMPRESS FOR ALL OPERATIONS
```

- Enable only direct load compression on existing table

```
ALTER TABLE t2 COMPRESS
```

  - Only new rows are compressed, existing rows are uncompressed

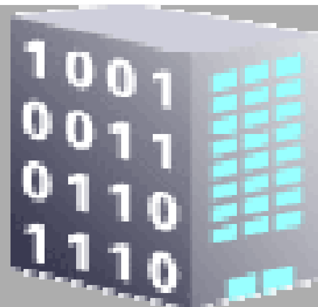
ORACLE

Data Pump Compression

- Metadata compression available since Oracle Database 10g
- Oracle Database 11g extends compression to table data during exports
  - No need to decompress before import
- Single step compression of both data and metadata
  - Compressed data directly hits disk resulting in reduced disk space requirements
  - 75% reduction in dump file size on export of sample OE and SH schemas
- Compression factor comparable to GNU gzip utility
- Application transparent
  - Complete Data Pump functionality available on compressed files

Backup data and Network transport Compression

- Fast RMAN Compression
  - Compresses the backup set contents before writing them to disk or tape
  - No extra decompression steps are required during recovery when you use RMAN compression.
  - High performance, industry standard compression algorithm
    - 40% faster backup compression versus Oracle Database 10g
  - Suitable for fast, incremental daily backups
  - Reduces network usage
- Data Guard Network Compression
  - Compression of redo traffic over the network
  - Improves redo transport performance
    - Gap resolution is up to 2x faster

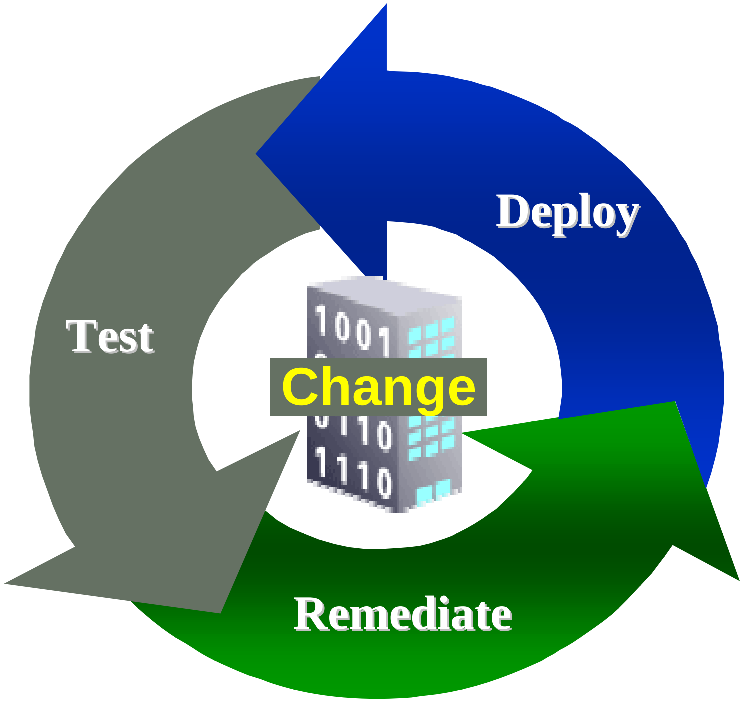


ere>

Oracle Database 11g
Real Application Testing

Real Application Testing

- Value
  - Rapid technology adoption
  - Higher testing quality
- Business Benefit
  - Lower cost
  - Lower risk



Solution for the Agile Business

Database Replay

The Need for Database Replay

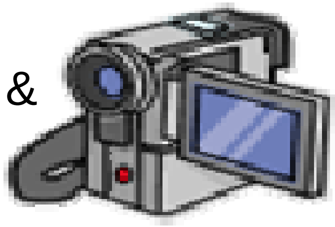
- Businesses want to adopt new technology that adds value
- Extensive testing and validation is expensive in time and cost
- Despite expensive testing success rate low
  - Many issues go undetected
  - System availability and performance negatively impacted
- Cause of low success rate
  - Current tools provide inadequate testing
    - Simulate synthetic workload instead of replaying actual production workload
    - Provide partial workflow coverage

Database Replay makes real-world testing possible

Database Replay

- Replay actual production database workload in test environment
- Identify, analyze and fix potential instabilities before making changes to production

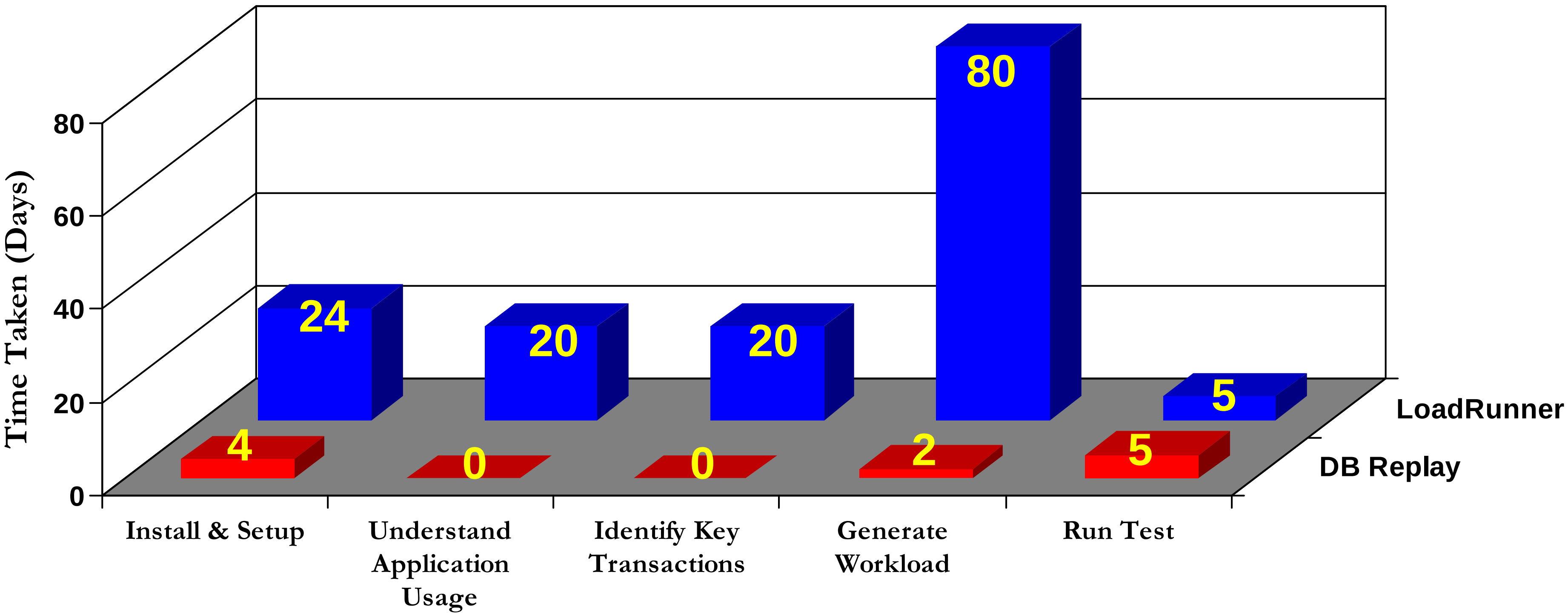
- Capture Workload in Production
  - Capture full production workload with real load, timing & concurrency characteristics
  - Move the captured workload to test system



- Replay Workload in Test
  - Make the desired changes in test system
  - Replay workload with full production characteristics
  - Honor commit ordering
- Analyze & Report
  - Errors
  - Data divergence
  - Performance divergence

Analysis & Reporting

Comparison of LoadRunner & DB Replay Testing e-Business Suite



Total Testing Time
DB Replay: 2 weeks
LoadRunner: 30 weeks

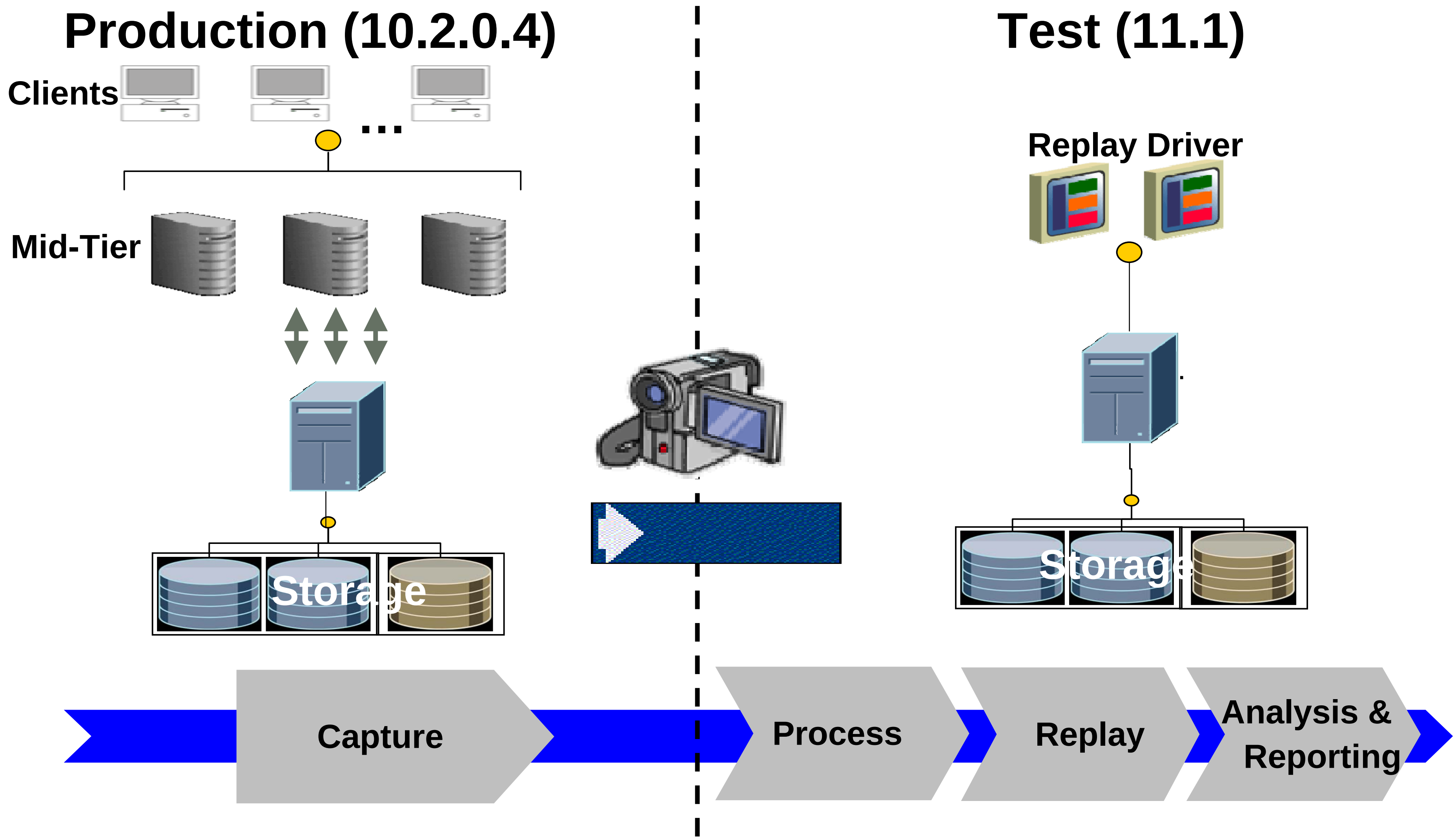
Why DB Replay?

150 Days

From:	To:
Artificial workloads	Production workloads
Partial workflows	Complete workflows
Months of development	Days of development
Manual intensive	Automated
High risk	Low risk

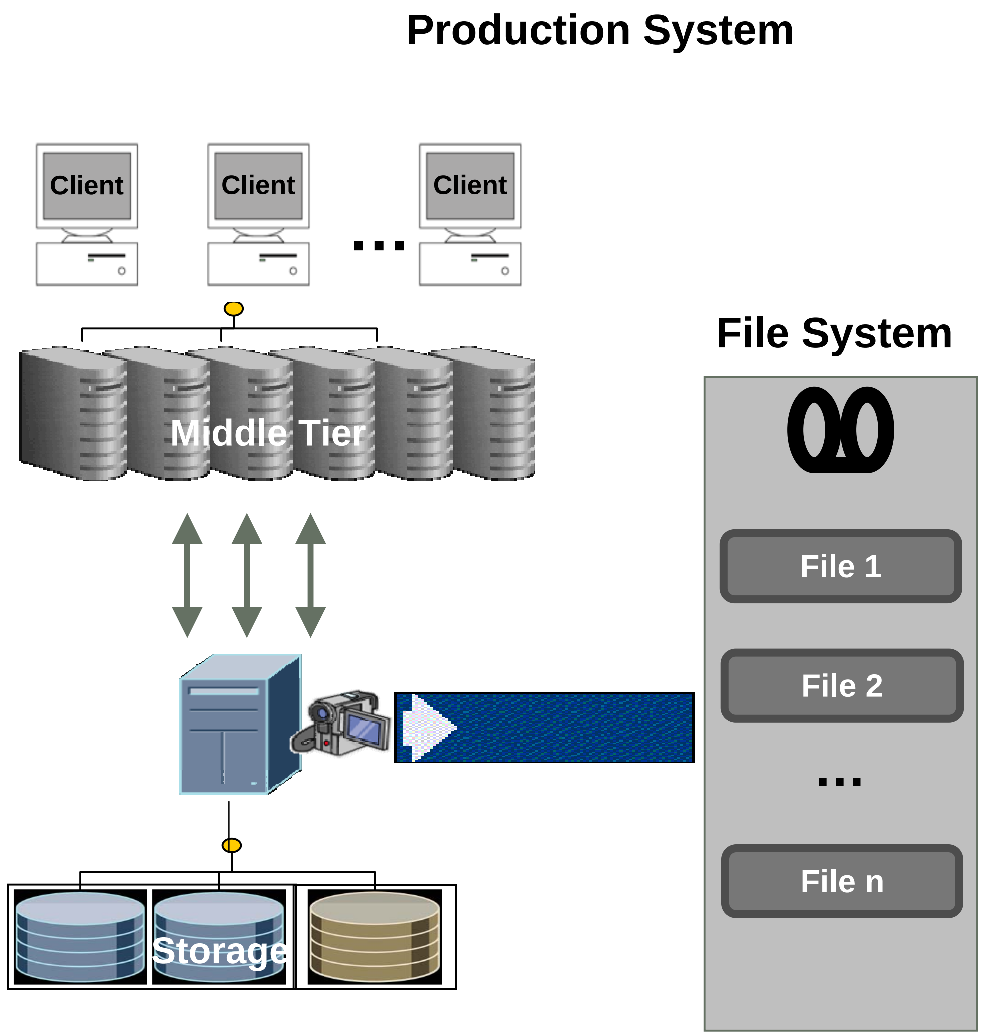
10 Days

Database Replay Workflow



Step 1: Workload Capture

- All external client requests captured in binary files
- System background, internal activity excluded
- Minimal performance overhead for capture
- For RAC, shared and local file system supported
- Specify interesting time period for capture, e.g., peak workload, month-end processing, etc.
- Can capture on 10.2.0.4 and replay on 11g

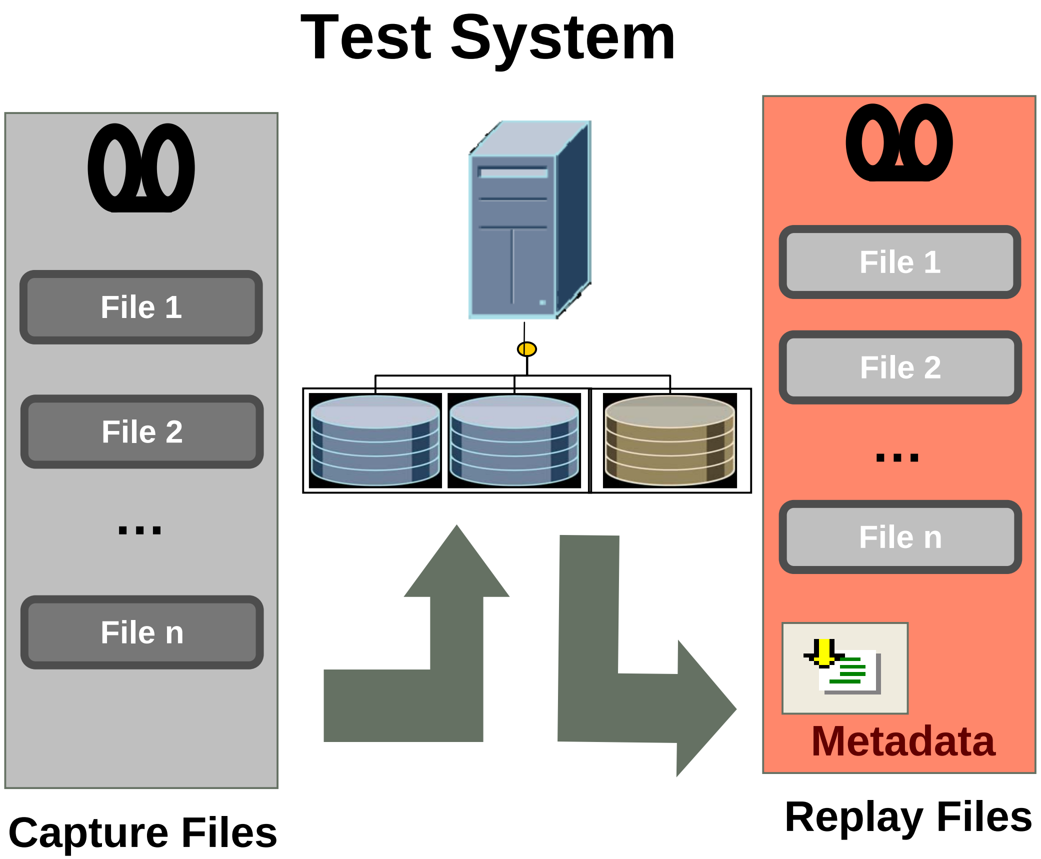


Capture Options

- Workload can be filtered to customize what is captured
  - Filter Types
    - Inclusion Filters: Specifies which sessions should be captured
    - Exclusion Filters: Specifies which sessions should NOT be captured
  - Filter Attributes: Workload capture can be filtered using any of the following session attributes
    - User
    - Program
    - Module
    - Action
    - Service
    - Session ID
- Workload capture can be run on-demand or scheduled to run at later time

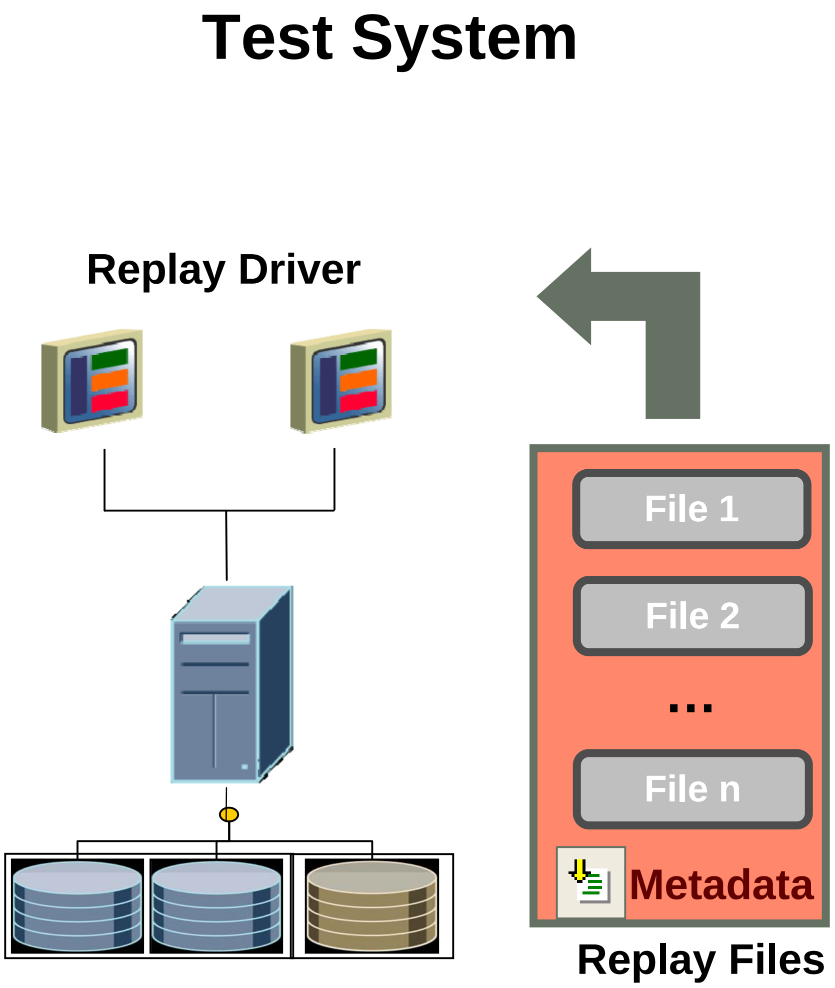
Step 2: Process Workload Files

- Setup test system
  - Test DB is at same point in time as before production capture
  - Use RMAN to physically restore production db from backup
  - Use Snapshot standby
  - Use imp/exp, Data Pump, etc.
- Processing transforms captured data into replayable format
- Once processed, workload can be replayed many times
- For RAC copy all capture files to single location for processing



Step 3: Replay Workload

- Replays workload preserving timing, concurrency and dependencies of the capture system
- Replay Driver is a special client program that consumes processed workload and sends requests to the replay system
- Replay Driver consists of one or more clients. For workloads with high concurrency, it may be necessary to start multiple clients to drive workload



Replay Options

- Synchronized Replay (Default)
  - Workload is replayed in full synchronized mode
  - Exact same concurrency and timing as production workload
  - Transaction commit order is honored
  - Ensures minimal data divergence
- Unsynchronized Replay
  - Workload can be replayed in unsynchronized mode
  - Useful for load/stress testing
  - High data divergence
  - Three (3) parameters provided to control degree of synchronization
    - Think time synchronization
    - Connect (logon) time synchronization
    - Commit order synchronization

Analysis & Reporting

- Comprehensive reports are provided for analysis purposes
- There (3) types of divergences are reported
  - Data Divergence: Number of rows returned by each call are compared and divergences reported
  - Error Divergence: For each call error divergence is reported
    - New: Error encountered during replay not seen during capture
    - Not Found: Error encountered during capture not seen during replay
    - Mutated: Different error produced in replay than during capture
  - Performance Divergence
    - Capture and Replay Report: Provides high-level performance information
    - ADDM Report: Provides in-depth performance analysis
    - AWR, ASH Report: Facilitates comparative or skew analysis

Database Replay Summary Report

Database Instance: database > Database Replay > Logged in As SYS

View Workload Replay: REPLAY3

OK

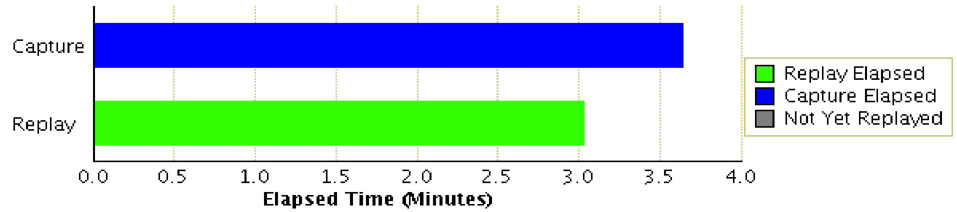
Status **Completed**

Summary

Replay Name	**REPLAY3**	Capture Name	**CAPTURE3**
Directory Object	**GP1**	Duration (hh:mm:ss)	**00:03:02**
Database Name	**V12**	Prepare Time	**Apr 13, 2007 4:46:46 PM PDT**
DBID	**1623065897**	Start Time	**Apr 13, 2007 4:48:32 PM PDT**
Replay Error Code	**N/A**	End Time	**Apr 13, 2007 4:51:34 PM PDT**
Replay Error Message	**None**		

Workload Profile | Detailed Comparison | Connections | Replay Parameters | Report

Elapsed Time Comparison



Divergence

	Calls	Percentage of Total
Capture Errors Not in Replay	0	0.00
Capture Errors Mutated in Replay	0	0.00
Capture Errors Not Replayed	0	0.00
Replay Errors Not in Capture	0	0.00
DML Rows Modified	0	0.00
Query Rows Fetched	0	0.00

Assessing the Replay

The Elapsed Time Comparison chart shows how much time the replayed workload has taken to accomplish the same amount of work as captured.

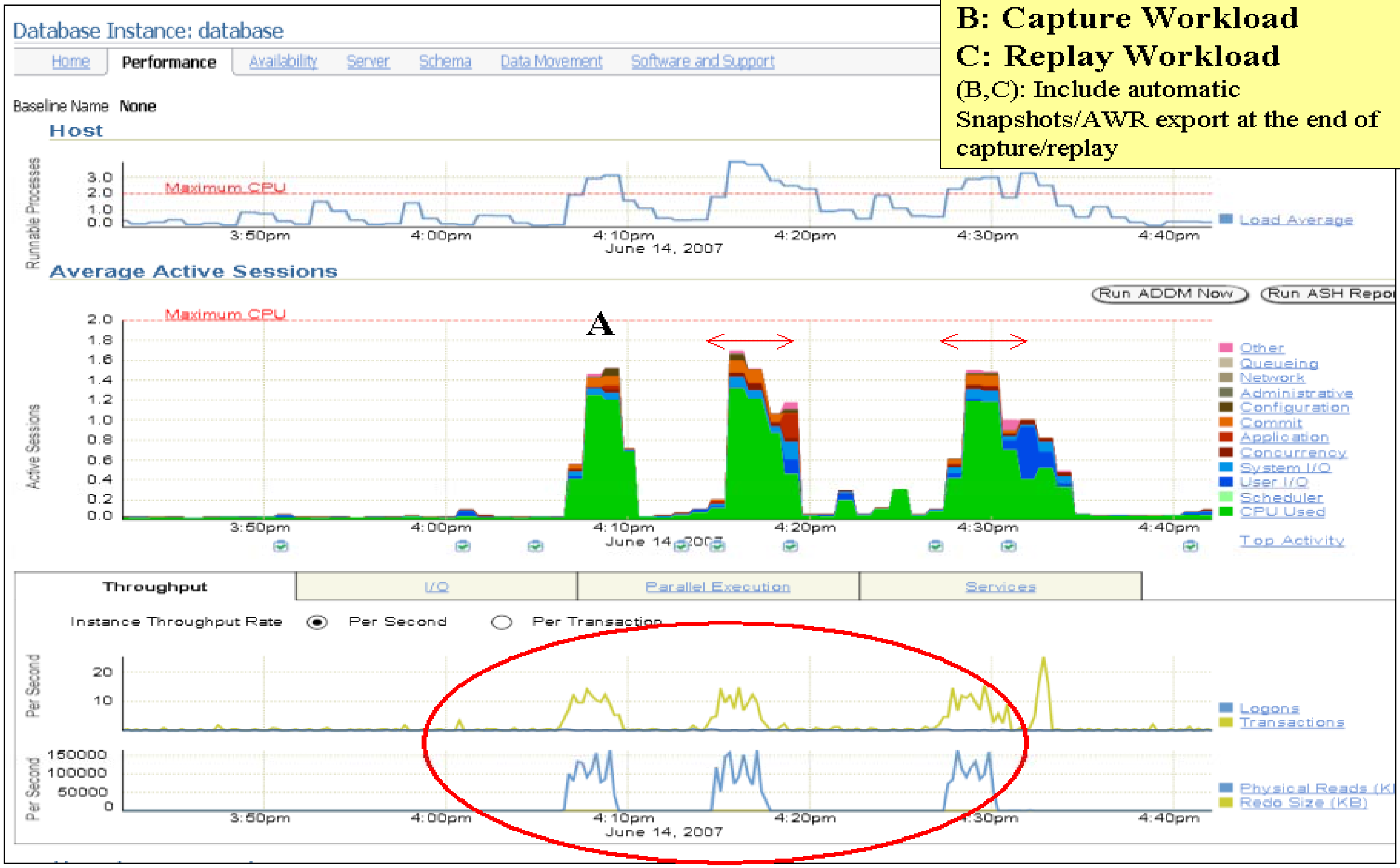
When the replay bar is shorter than the capture bar then the replay environment is processing the workload faster than the capture environment.

The divergence table gives information about both the data and error discrepancies between the replay and capture environments, which can be used as a measure of the replay quality.

View Workload Replay Report

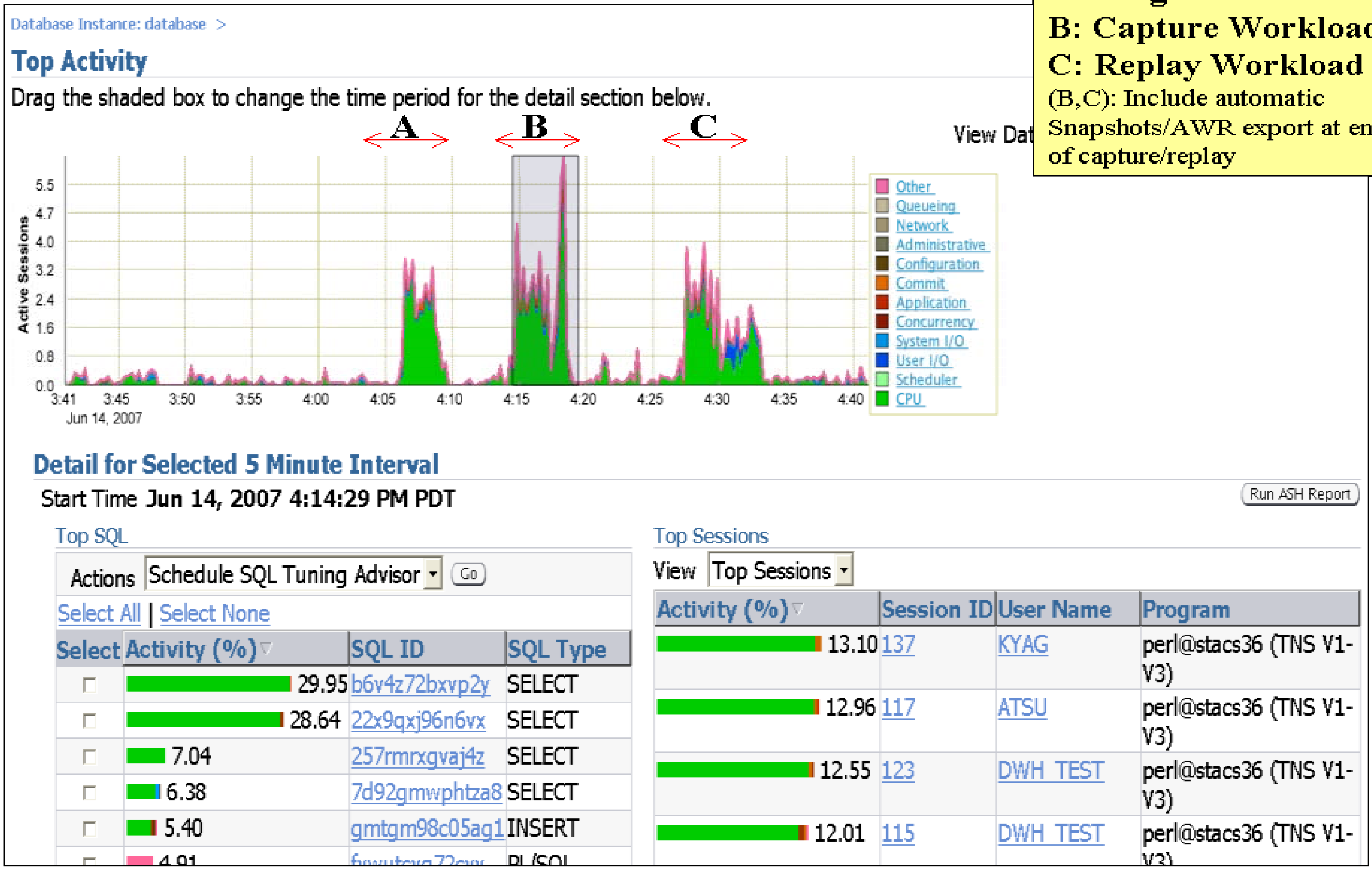
Performance Page – Database Replay

A: Original Workload
B: Capture Workload
C: Replay Workload
(B,C): Include automatic Snapshots/AWR export at the end of capture/replay



Top Activity Page: Database Replay

A: Original Workload
B: Capture Workload
C: Replay Workload
(B,C): Include automatic Snapshots/AWR export at end of capture/replay



Best Practices

- Capture
  - Provide adequate disk space for captured workload (binary files)
  - Database restart (Optional): Recommended to minimize divergence
  - For RAC, use shared file system
- Test System Setup
  - Ensure data in test is identical to production as of capture start time to minimize data divergence during replay
  - Use RMAN backup/restore or Snapshot Standby to setup test system
  - For performance analysis test system capacity should be similar to production
  - Reset system clock to same time as production if application logic involves SYSDATE usage
- Process Workload
  - Processing workload has performance overhead and can possibly take a long time
  - Process workload on test system instead of production
- Replay
  - Use Client Calibration Advisor to identify number of replay clients needed to replay workload properly

SQL Performance Analyzer (SPA)

The Need for SQL Performance Analyzer (SPA)

- Businesses want systems that are performant and meet SLA’s
- SQL performance regressions are #1 cause of poor system performance
- Solution for proactively detecting ***all*** SQL regressions resulting from changes not available
- DBA’s use ineffective and time-consuming manual scripts to identify problems

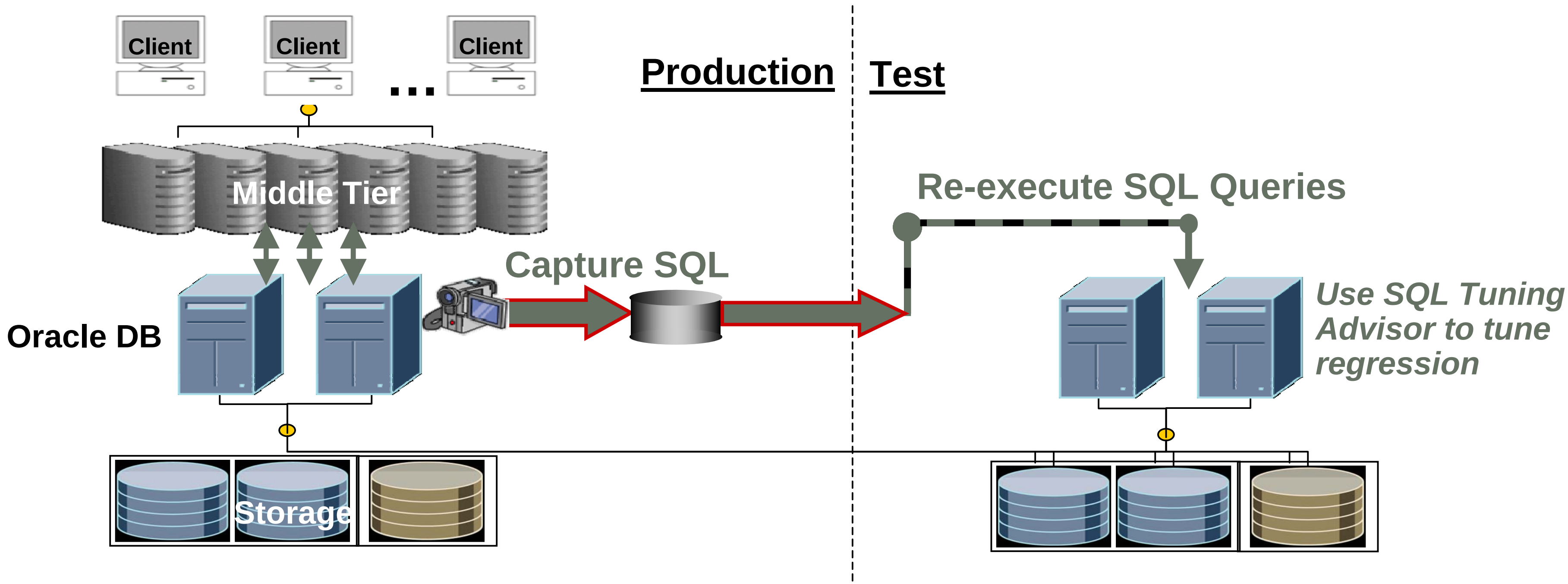
SPA identifies all changes in SQL performance before impacting users

Why SQL Performance Analyzer?

From:	To:
Manual workload creation	Automated workload capture
Synthetic workload	Production workload
Months of manual analysis	Automated analysis in minutes
Partial workload	Complete workload
High risk	Low risk

SQL Performance Analyzer

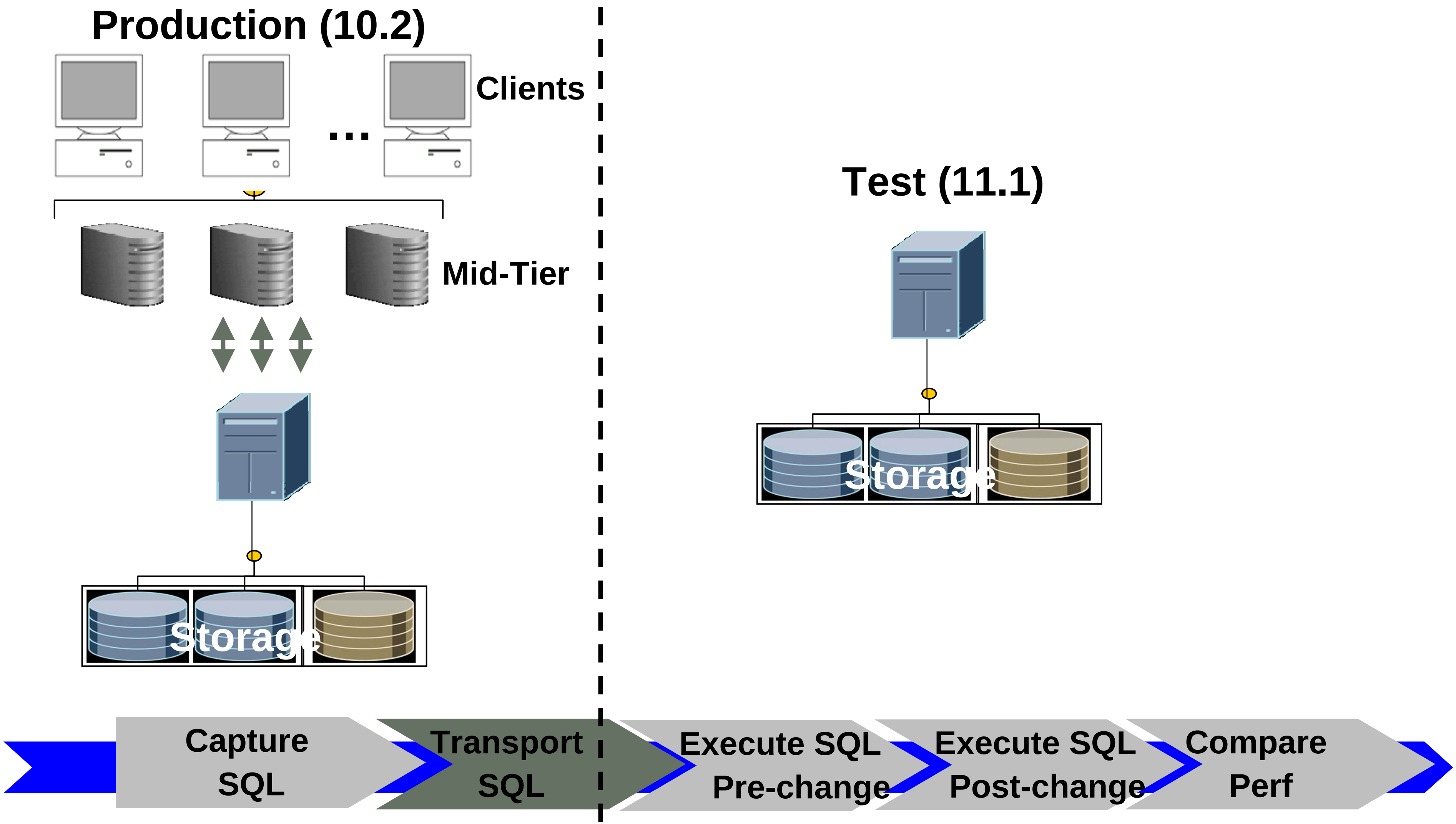
- Test impact of change on SQL query performance
- Capture SQL workload in production including statistics & bind variables
- Re-execute SQL queries in test environment
- Analyze performance changes – improvements and regressions



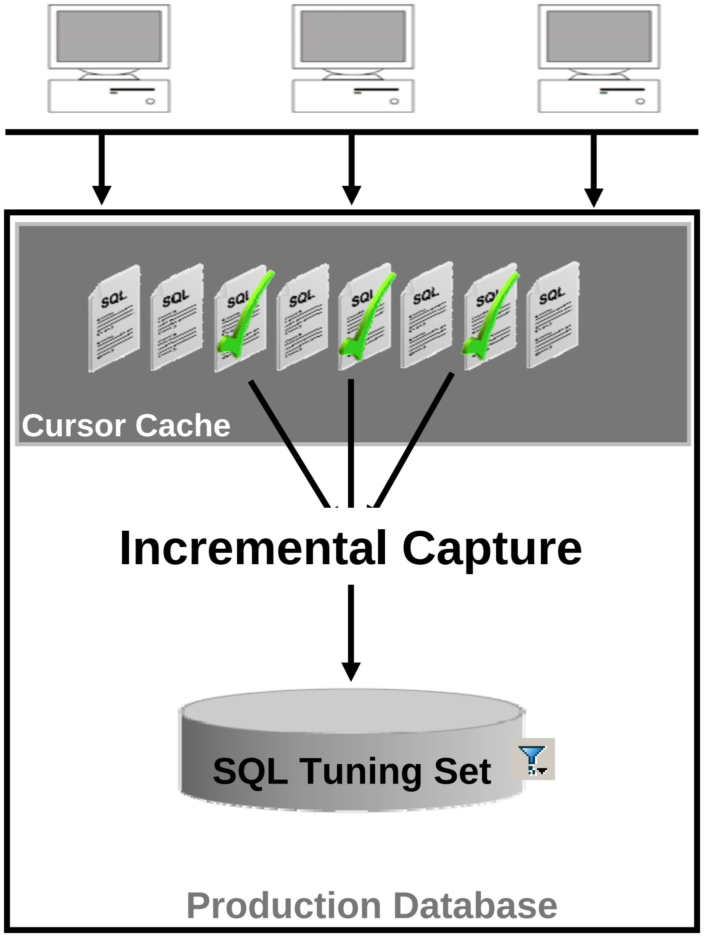
SPA Benefits

- Enables identification of SQL performance regressions ***before*** end-users can be impacted
- SPA can help with any change that impacts SQL execution plan
  - DB upgrades
  - Optimizer statistics refresh
  - New indexes, Materialized Views, Partitions, etc.
- Automates SQL performance tracking of hundreds of thousands of SQL statements – impossible to do manually
- Captures SQL workload with low overhead
- Integrated with SQL Tuning Advisor and SQL Plan Baselines for regression remediation

SQL Performance Analyzer Workflow

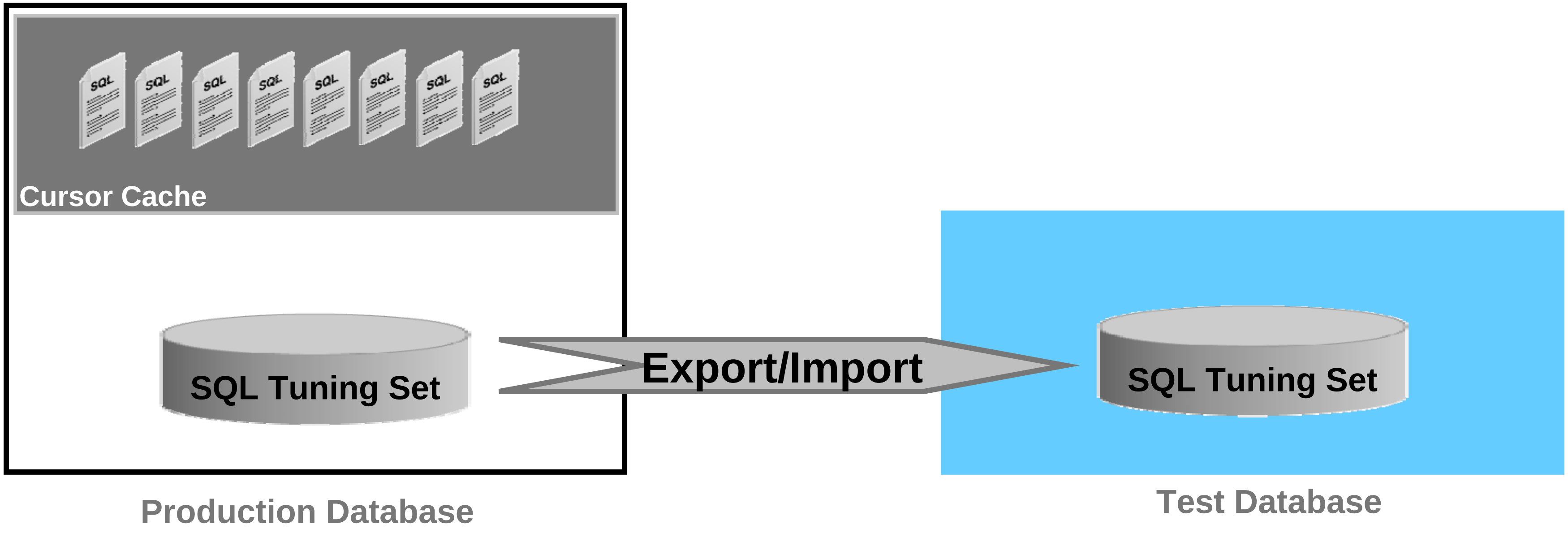


Step 1: Capture SQL Workload



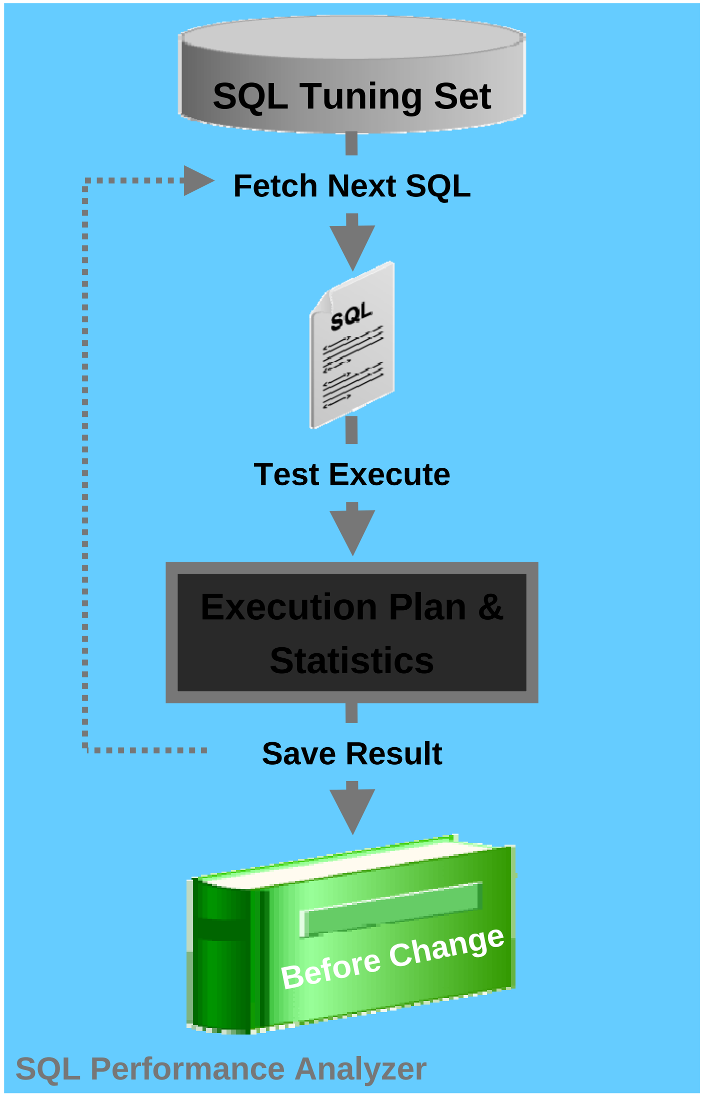
- SQL Tuning Set (STS) used to store SQL workload
- STS includes:
  - SQL Text
  - Bind variables
  - Execution plans
  - Execution statistics
- Incremental capture used to populate STS from cursor cache over a time period
- SQL tuning set's filtering and ranking capabilities filters out undesirable SQL
- SQL workload captured in 10.2.0.1 and higher can be used for SPA tasks in 11g

Step 2: Move SQL Workload to Test System



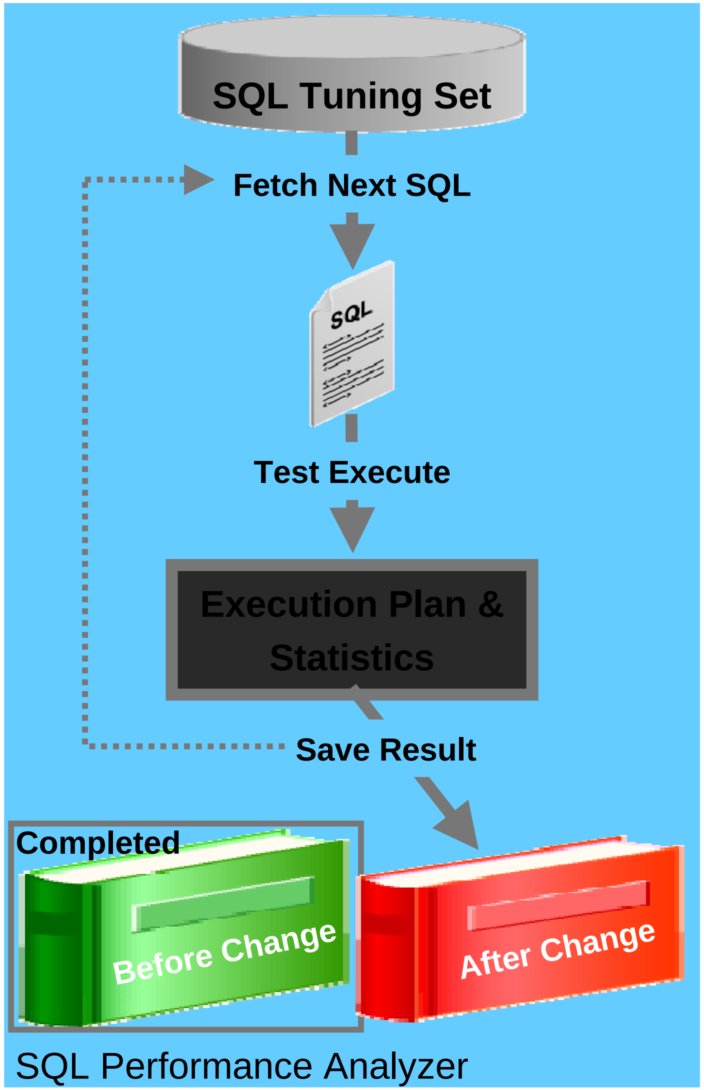
- Copy SQL tuning set to staging table ("pack")
- Transport staging table to test system (datapump, db link, etc.)
- Copy SQL tuning set from staging table ("unpack")

Step 3: Execute SQL Before Making Change



- Establishes SQL workload performance baseline
- SQL execution plan and statistics captured
- SQL executed serially (no concurrency)
- Each SQL executed only once
- DDL/DML skipped
- Option to do Explain Plan only analysis

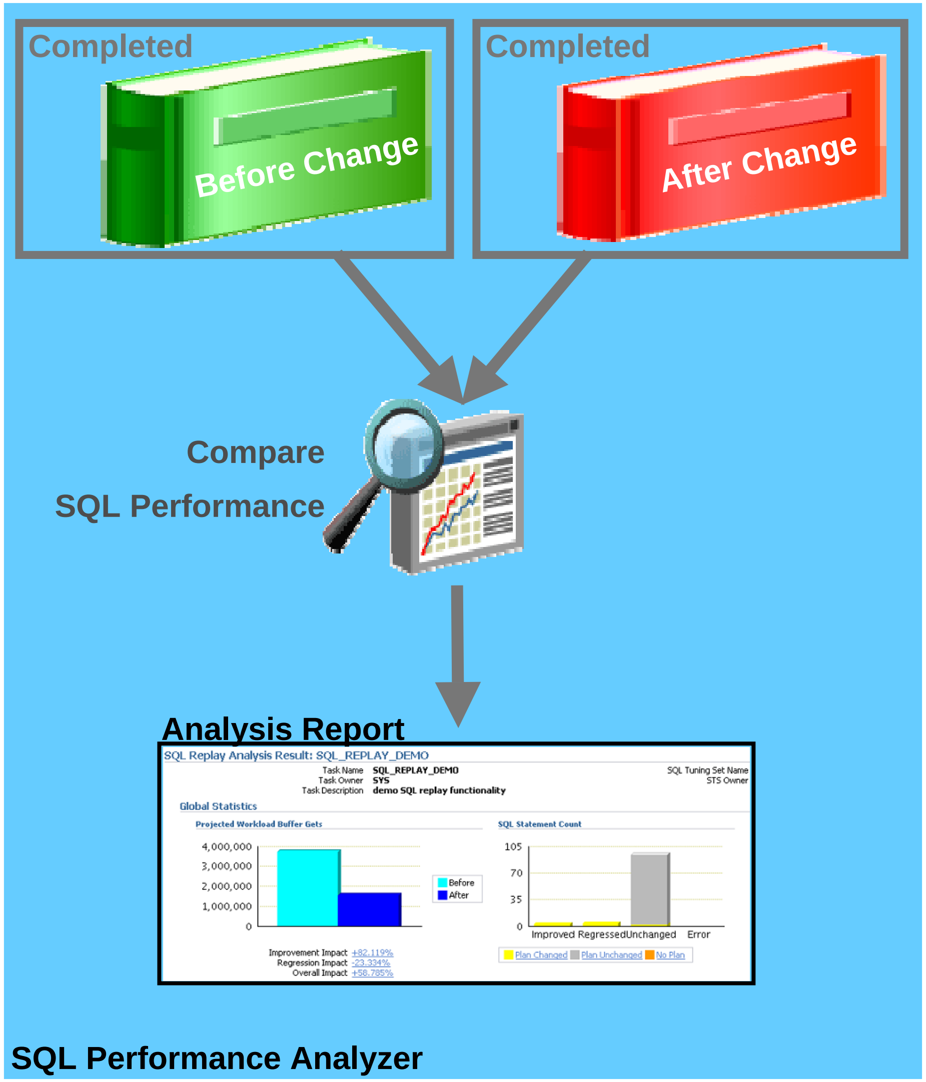
Step 4: Execute SQL After Making Change



- **Manually implement the planned change**
  - Database upgrade, patches
  - Optimizer statistics refresh
  - Schema changes
  - Database parameter changes
  - Tuning actions, e.g., SQL Profile creation

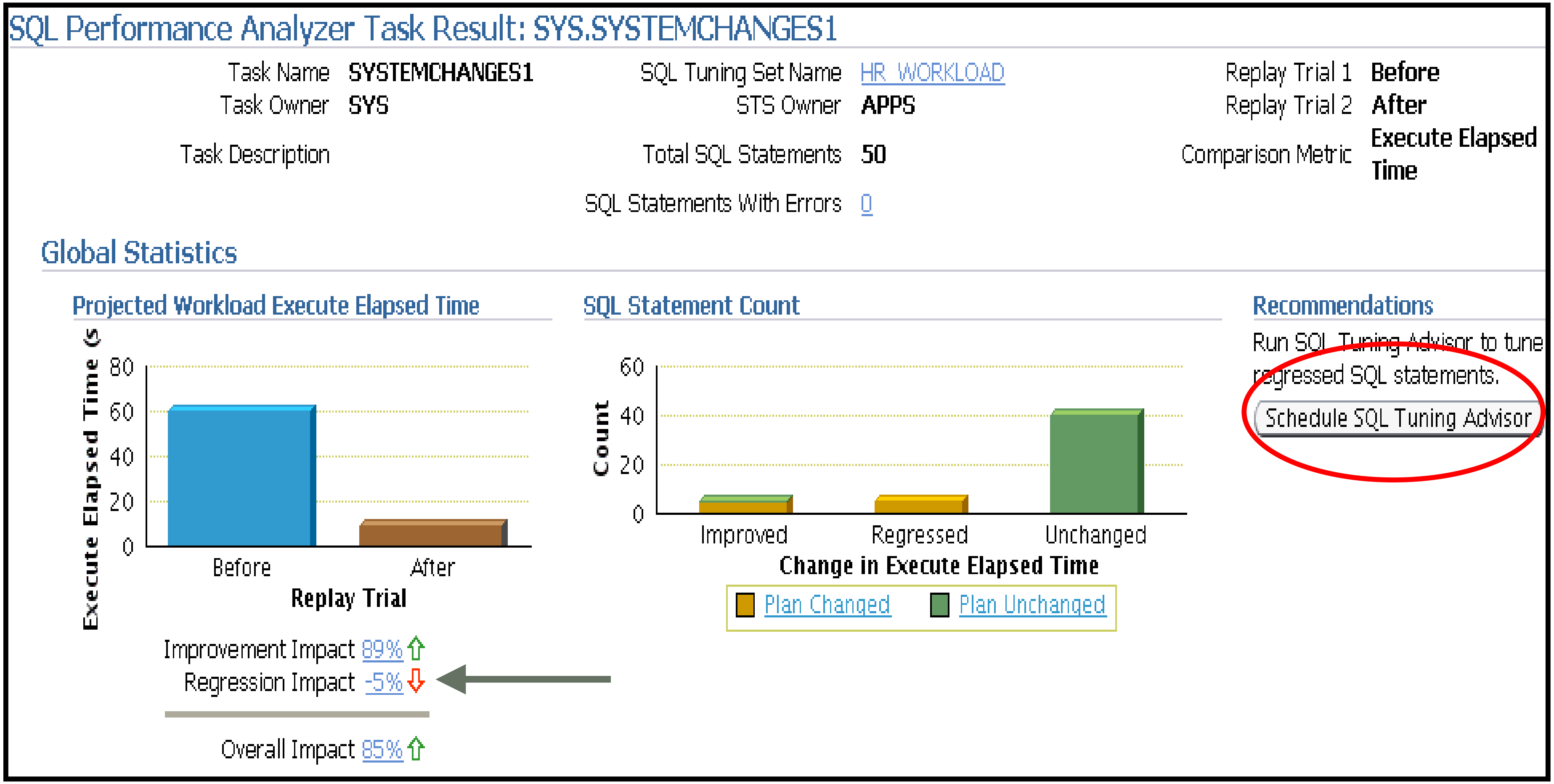
- **Re-execute SQL after change**
  - Gathers new SQL execution plans and statistics

Step 5: Compare & Analyze Performance



- Compare performance using different metrics, e.g.,
  - Elapsed Time
  - CPU Time
  - Optimizer Cost
  - Buffer Gets
- SPA Report shows impact of change for each SQL
  - Improved SQL
  - Regressed SQL
  - Unchanged SQL
- Fix regressed SQL using SQL Tuning Advisor or SQL Plan Baselines

SPA Report


SQL Performance Analyzer Task Result: SYS.SYSTEMCHANGES1
Task Name SYSTEMCHANGES1
Task Owner SYS
Task Description
SQL Tuning Set Name HR_WORKLOAD
STS Owner APPS
Total SQL Statements 50
SQL Statements With Errors 0
Replay Trial 1 Before
Replay Trial 2 After
Comparison Metric Execute Elapsed Time
Global Statistics
Projected Workload Execute Elapsed Time
Execute Elapsed Time (s
80
60
40
20
0
Before
After
Replay Trial
Improvement Impact 89%
Regression Impact -5%
Overall Impact 85%
SQL Statement Count
Count
60
40
20
0
Improved
Regressed
Unchanged
Change in Execute Elapsed Time
Plan Changed
Plan Unchanged
Recommendations
Run SQL Tuning Advisor to tune regressed SQL statements.
Schedule SQL Tuning Advisor

SPA Report – Regressed SQL Statements

Regressed Plan Changed SQL Statements

SQL ID	Executions	Net Impact on Workload (%)	Execute Elapsed Time		Net Impact on SQL (%)	% of Workload	
			Before_change	After_change		Before_change	After_change
2wtgxbjz6u2by	1	-4.390	2.004	4.697	-134.380	3.270	49.640
fbp9za0hqk2km	1	-0.020	0.000	0.015	-1,500.000	0.000	0.160
654xs8xs5wp42	1	-0.020	0.000	0.013	-1,300.000	0.000	0.140

SQL Details: 2wtgxbjz6u2by

Parsing Schema **APPS** Execution Frequency **1**

Schedule SQL Tuning Advisor

SQL Text

Single Execution Statistics

Execution Statistic Name	Net Impact on Workload (%)	Execution Statistic Collected		Net Impact on SQL (%)	% of Workload	
		Before_change	After_change		Before_change	After_change
Elapsed Time	-4.760	2.009	4.932	-145.500	3.270	50.740
Parse Time	-741.940	0.005	0.235	-4,600.000	16.130	90.730
Execute Elapsed Time	-4.390	2.004	4.697	-134.380	3.270	49.640
Execute CPU Time	-4.450	2.003	4.697	-134.500	3.310	49.650
Buffer Gets	-23.660	218,481.000	1,102,254.000	-404.510	5.850	70.970
Optimizer Cost	15.340	10,408.000	760.000	92.700	16.550	1.410
Disk Reads	0.000	0.000	0.000	0.000	0.000	0.000
Direct Writes	0.000	0.000	0.000	0.000	0.000	0.000
Rows Processed	0.000	135.000	135.000	0.000	0.000	0.000

Problem Findings

The performance of this SQL has regressed.

Symptom Findings

The structure of the SQL execution plan has changed.

Information Findings

Plan Comparison

Before_change

Plan Hash Value **1511732424**

Expand All | Collapse All

Operation	Line ID	Object
SELECT STATEMENT	0	
HASH	1	
NESTED LOOPS	2	

After_change

Plan Hash Value **3517011486**

Expand All | Collapse All

Operation	Line ID	Object
SELECT STATEMENT	0	
HASH	1	
NESTED LOOPS	2	

ORACLE

Best Practices

- Workload Capture
  - Use incremental STS capture
  - Peak or representative workloads can be more accurately captured with this mechanism
- Test system
  - Run SPA on test system instead of production as analysis can be resource intensive
  - Ensure test system has similar configuration and comparable optimizer statistics as production
- Performance comparison
  - Use several different metrics, e.g., elapsed time, CPU time, etc., to compare pre- and post-change performance for reliable results
- Regression remediation
  - Use SQL Tuning Advisor and SQL Plan Baselines to fix regressions

Oracle Partitioning in Oracle Database 11g

Oracle Partitioning

Ten Years of Development

	Core functionality	Performance	Manageability
Oracle8	Range partitioning Global range indexes	"Static" partition pruning	Basic maintenance operations: add, drop, exchange
Oracle8*i*	Hash and composite range-hash partitioning	Partition-wise joins "Dynamic" pruning	Merge operation
Oracle9*i*	List partitioning		Global index maintenance
Oracle9*i* R2	Composite range-list partitioning	Fast partition split	
Oracle10*g*	Global hash indexes		Local Index maintenance
Oracle10*g* R2	1M partitions per table	"Multi-dimensional" pruning	Fast drop table
Oracle Database 11*g*	More composite choices REF Partitioning Virtual Column Partitioning		Interval Partitioning Partition Advisor

Oracle Partitioning Enhancements

- **Complete the basic partitioning strategies**
  defines HOW data is going to be partitioned
  - new composite partitioning methods

- **Introduce partitioning extensions**
  defines WHAT controls the data placement
  - enhance the manageability and automation
  - virtual column based partitioning
  - REF partitioning
  - interval partitioning
  - partition advisor

Composite Partitioning in Oracle Database 11g

Extended Composite Partitioning Strategies

- Concept of composite partitioning
  - Data is partitioned along two dimensions (A,B)
  - A distinct value pair for the two dimensions uniquely determines the target partitioning
- Composite partitioning is complementary to multi-column range partitioning
- Extensions in Oracle Database 11g ..

New 11g Strategy	Use Case
List – Range	Geography -Time
Range - Range	ShipDate - OrderDate
List - Hash	Geography - OrderID
List - List	Geography - Product

Composite Partitioning - Concept

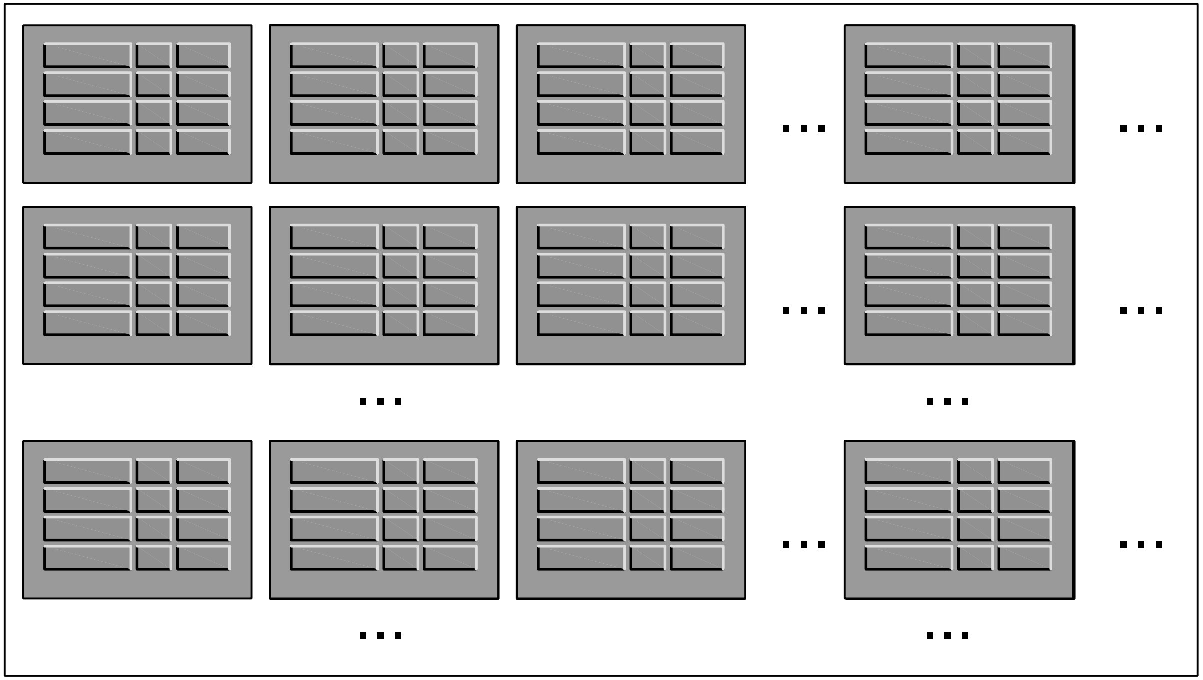
Table SALES
RANGE(order_date)-RANGE(ship_date)

ship_date

Jan 2006

Feb 2006

May 2006

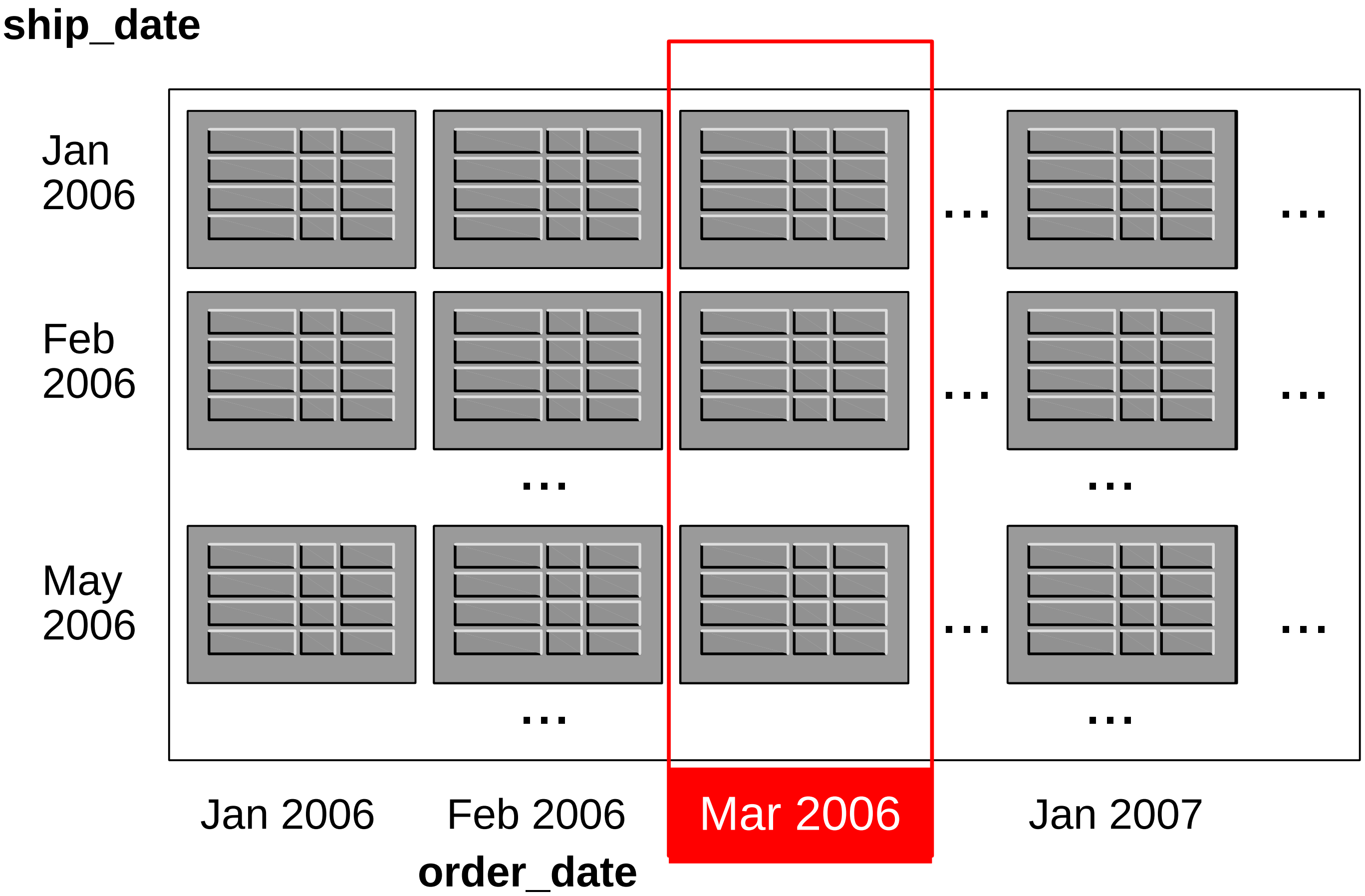


Jan 2006 | Feb 2006 | Mar 2006 | Jan 2007

order_date

Composite Partitioning - Concept

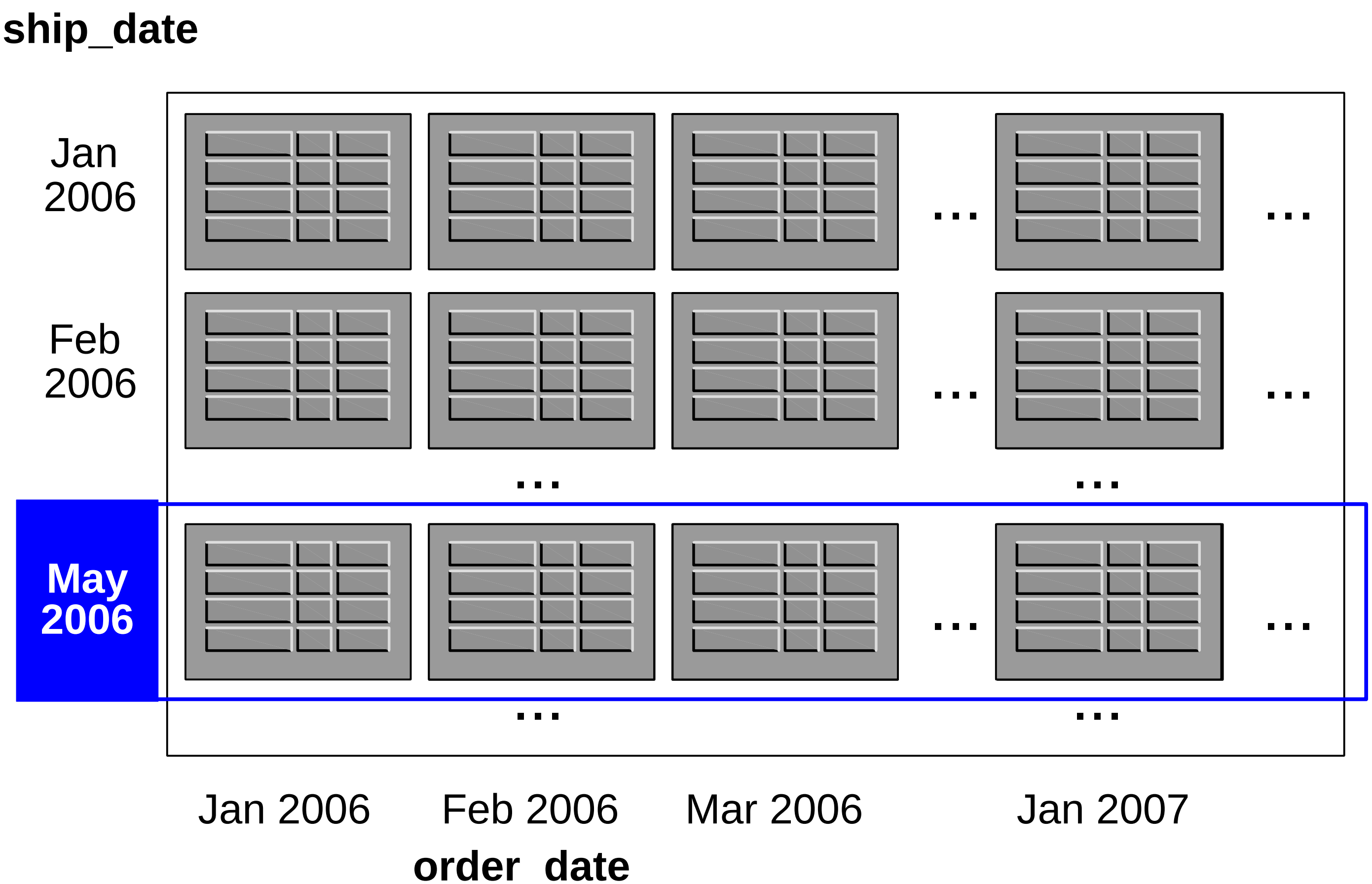
Table SALES
RANGE(order_date)-RANGE(ship_date)



- **All records with order_date in March 2006**

Composite Partitioning - Concept

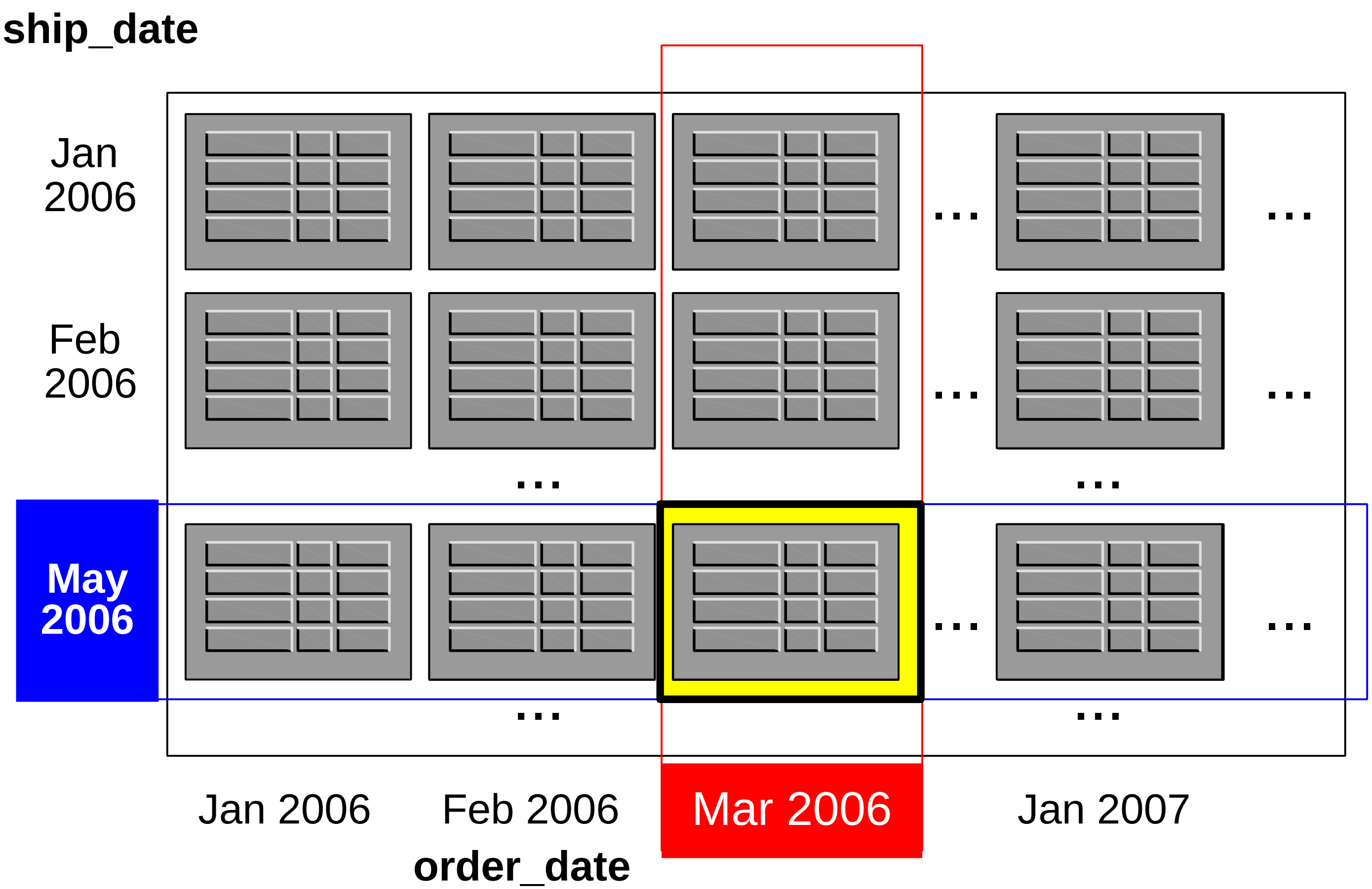
Table SALES
RANGE(order_date)-RANGE(ship_date)



- **All records with ship_date in May 2006**

Composite Partitioning - Concept

Table SALES
RANGE(order_date)-RANGE(ship_date)



- **All records with order_date in March 2006 AND ship_date in May 2006**

Virtual Column based Partitioning

Virtual Columns

Business Problem

- Extended Schema attributes are fully derived and dependent on existing common data
- Redundant storage or extended view definitions are solving this problem today
  - requires additional maintenance and creates overhead

Solution

- Oracle Database 11g introduces virtual columns
  - purely virtual, meta-data only
- Treated as real columns except no DML
  - can have statistics
  - eligible as partitioning key
- Enhanced performance and manageability

Virtual Columns - Example

- Base table with all attributes ...

```
CREATE TABLE accounts
(acc_no       number(10)    not null,
 acc_name     varchar2(50) not null, ...
```

12500	Adams
12507	Blake
12666	King
12875	Smith

Virtual Columns - Example

- Base table with all attributes ...
  - ... is extended with the virtual (derived) column

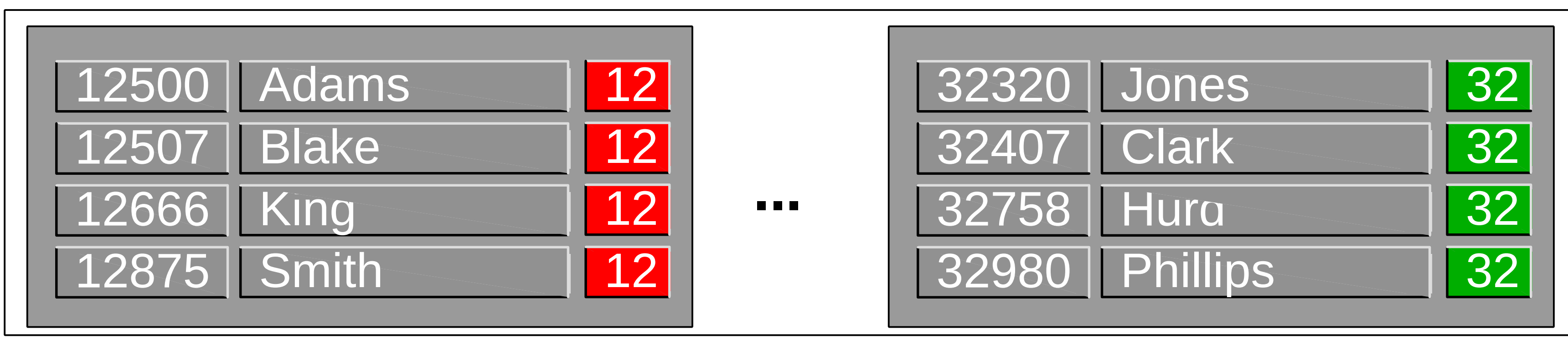
```
CREATE TABLE accounts
(acc_no      number(10)   not null,
 acc_name    varchar2(50) not null, ...
 acc_branch number(2)     generated always as
  (to_number(substr(to_char(acc_no),1,2)))
```

12500	Adams	12
12507	Blake	12
12666	King	12
12875	Smith	12

Virtual Columns - Example

- Base table with all attributes ...
  - ... is extended with the virtual (derived) column
  - ... and the virtual column is used as partitioning key

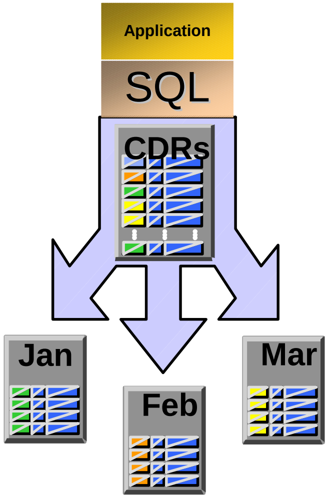
```
CREATE TABLE accounts
(acc_no      number(10)   not null,
 acc_name    varchar2(50) not null, ...
 acc_branch number(2)    generated always as
                  (to_number(substr(to_char(acc_no),1,2)))
partition by list (acc_branch) ...
```



Interval Partitioning

Interval Partitioning

- Partitioning is key-enabling functionality for managing large volumes of data
  - one logical object for application transparency
  - multiple physical segments for administration



but

- Physical segmentation requires additional data management overhead
  - new partitions must be created on-time for new data

Automate the partition management

Interval Partitioning

- Interval Partitioning
  - extension to range partitioning
  - full automation for equi-sized range partitions
- Partitions are created as metadata information only
  - start partition is made persistent
- Segments are allocated as soon as new data arrives
  - no need to create new partitions
  - local indexes are created and maintained as well

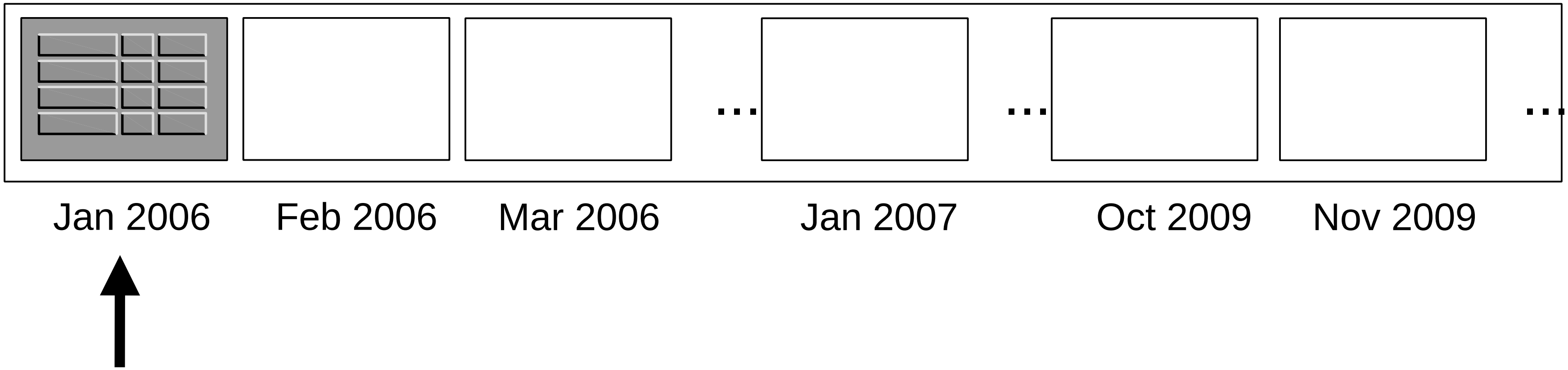
No need for any partition management

Interval Partitioning

How it works

```
CREATE TABLE sales (order_date DATE, ...)
PARTITON BY RANGE (order_date)
INTERVAL(NUMTOYMINTERVAL(1,'month')
(PARTITION p_first VALUES LESS THAN ('01-JAN-2006');
```

Table SALES



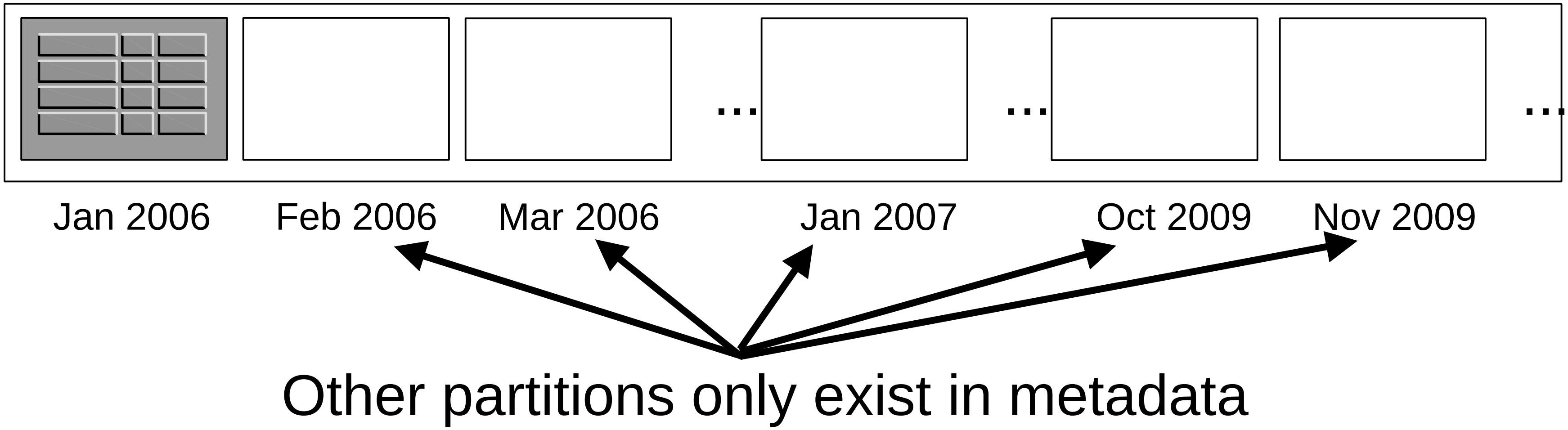
First segment is created

Interval Partitioning

How it works

```
CREATE TABLE sales (order_date DATE, ...)
PARTITON BY RANGE (order_date)
INTERVAL(NUMTOYMINTERVAL(1,'month')
(PARTITION p_first VALUES LESS THAN ('01-JAN-2006');
```

Table SALES

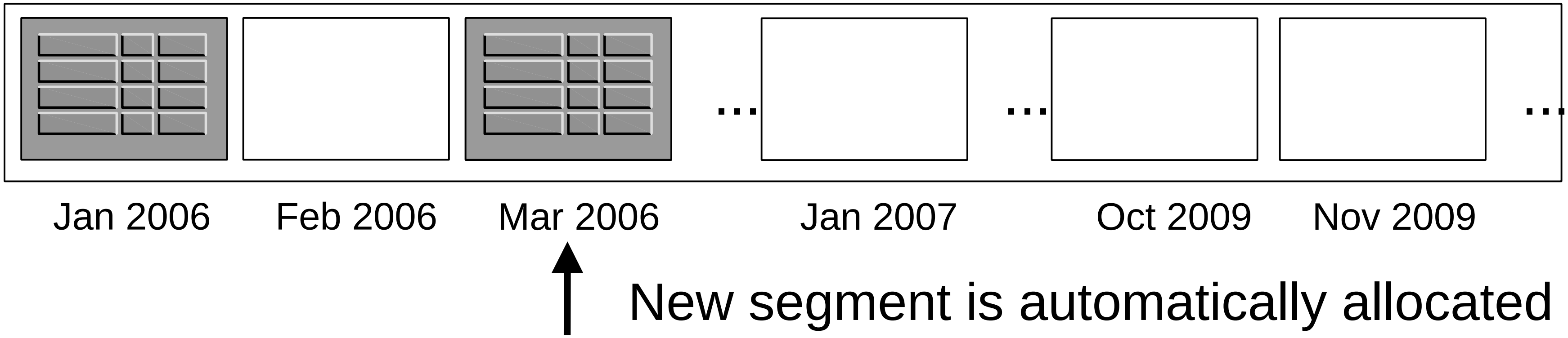


Interval Partitioning

How it works

```
CREATE TABLE sales (order_date DATE, ...)
PARTITON BY RANGE (order_date)
INTERVAL(NUMTOYMINTERVAL(1,'month')
(PARTITION p_first VALUES LESS THAN ('01-JAN-2006');
```

Table SALES



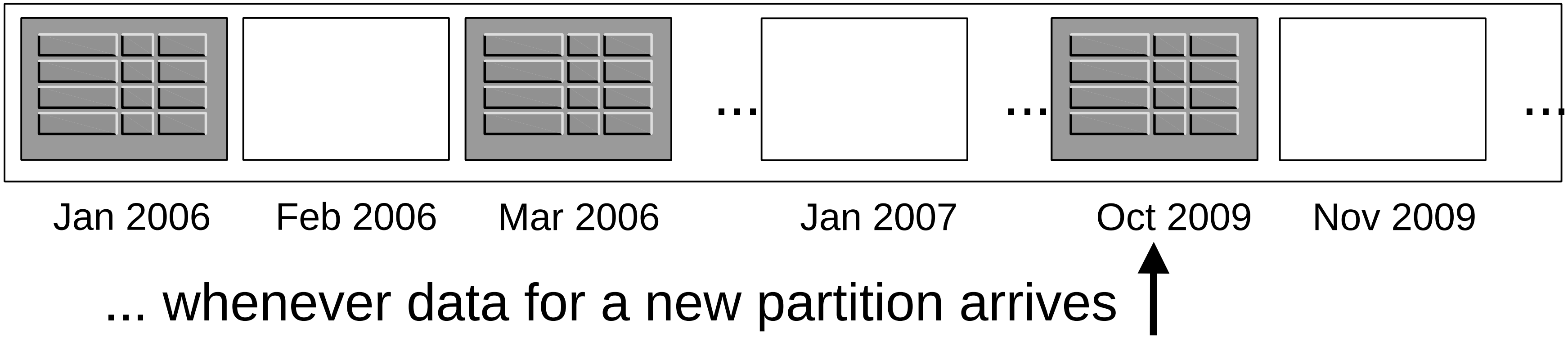

```
INSERT INTO sales (order_date DATE, ...)
VALUES ('04-MAR-2006',...);
```

Interval Partitioning

How it works

```
CREATE TABLE sales (order_date DATE, ...)
PARTITON BY RANGE (order_date)
INTERVAL(NUMTOYMINTERVAL(1,'month')
(PARTITION p_first VALUES LESS THAN ('01-JAN-2006');
```

Table SALES



... whenever data for a new partition arrives

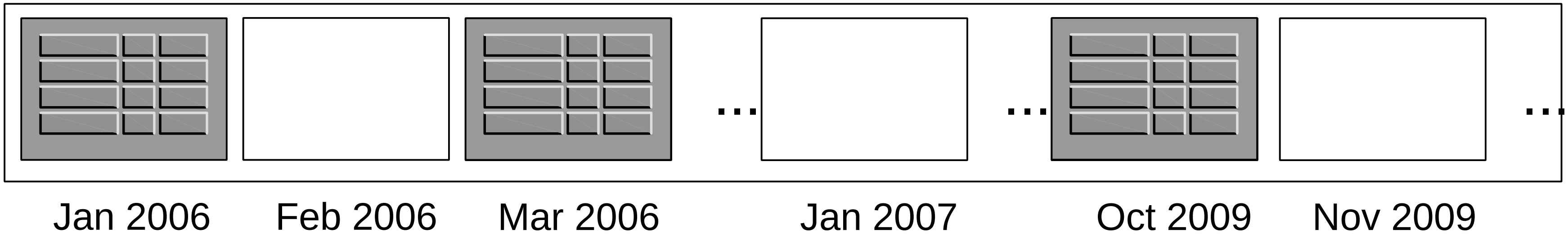
```
INSERT INTO sales (order_date DATE, ...)
VALUES ('17-OCT-2009',...);
```

Interval Partitioning

How it works

- Interval partitioned table can have classical range and automated interval section
  - Automated new partition management plus full partition maintenance capabilities: ***"Best of both worlds"***

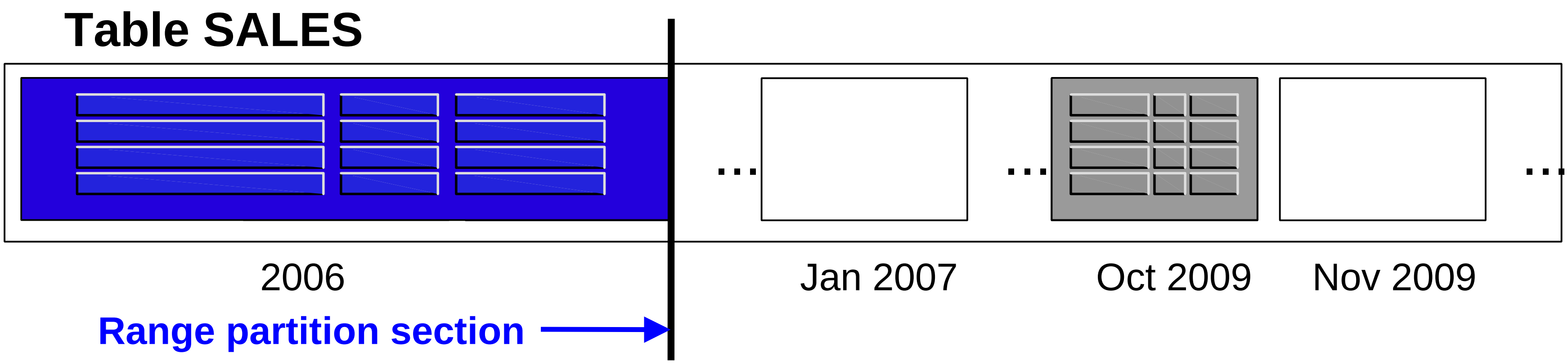
Table SALES



Interval Partitioning

How it works

- Interval partitioned table can have classical range and automated interval section
  - Automated new partition management plus full partition maintenance capabilities: ***"Best of both worlds"***



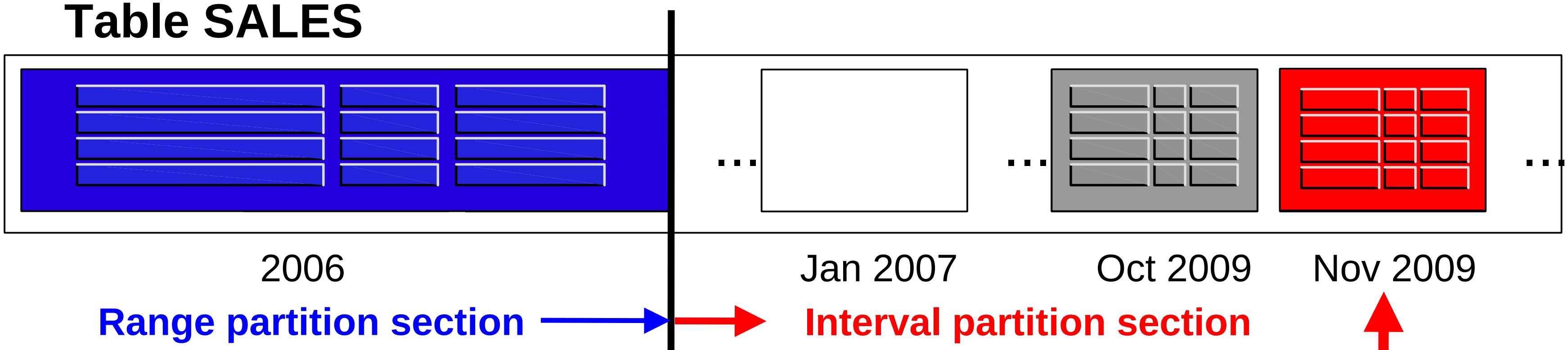
MERGE and move old partitions for ILM

Interval Partitioning

How it works

- Interval partitioned table can have classical range and automated interval section
  - Automated new partition management plus full partition maintenance capabilities: ***"Best of both worlds"***

Table SALES



MERGE and move old partitions for ILM

Insert new data
- Automatic segment creation

```
INSERT INTO sales (order_date DATE, ...)
VALUES ('13-NOV-2009',...);
```

REF Partitioning

REF Partitioning

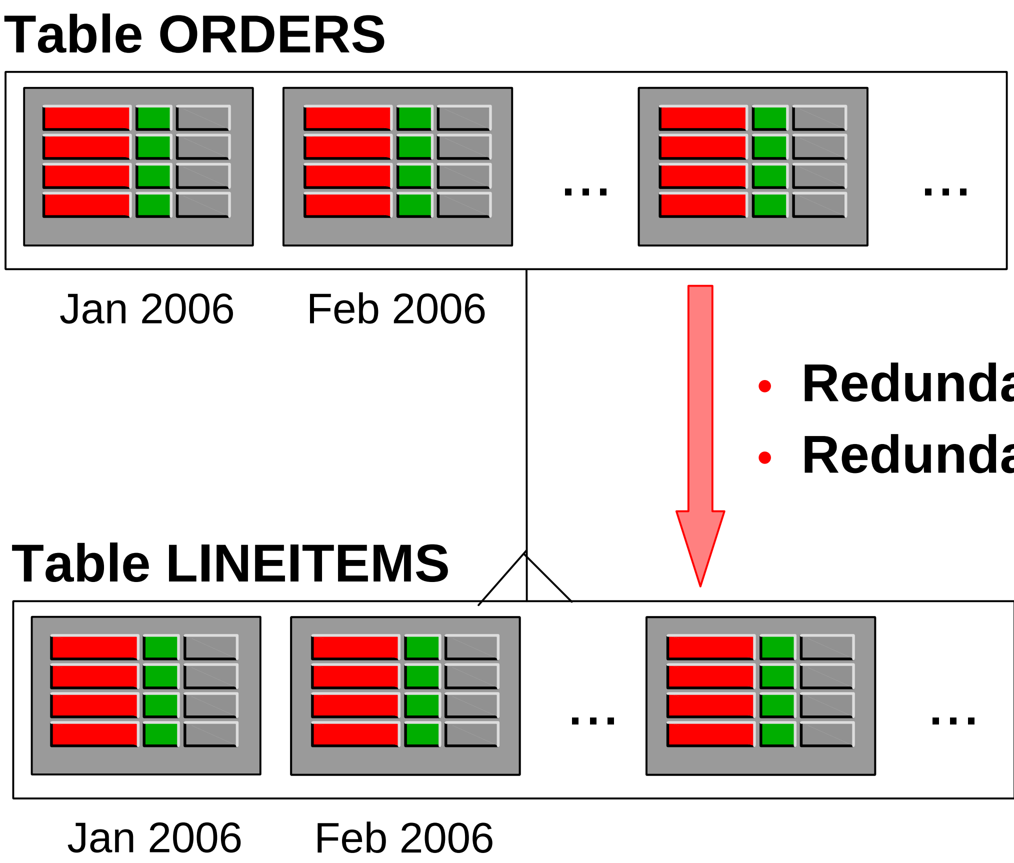
Business Problem

- Related tables benefit from same partitioning strategy
  - e.g. order – lineitem
- Redundant storage of the same information solves this problem
  - data overhead
  - maintenance overhead

Solution

- Oracle Database 11g introduces REF Partitioning
  - child table inherits the partitioning strategy of parent table through PK-FK relationship
  - intuitive modelling
- Enhanced Performance and Manageability

Before REF Partitioning



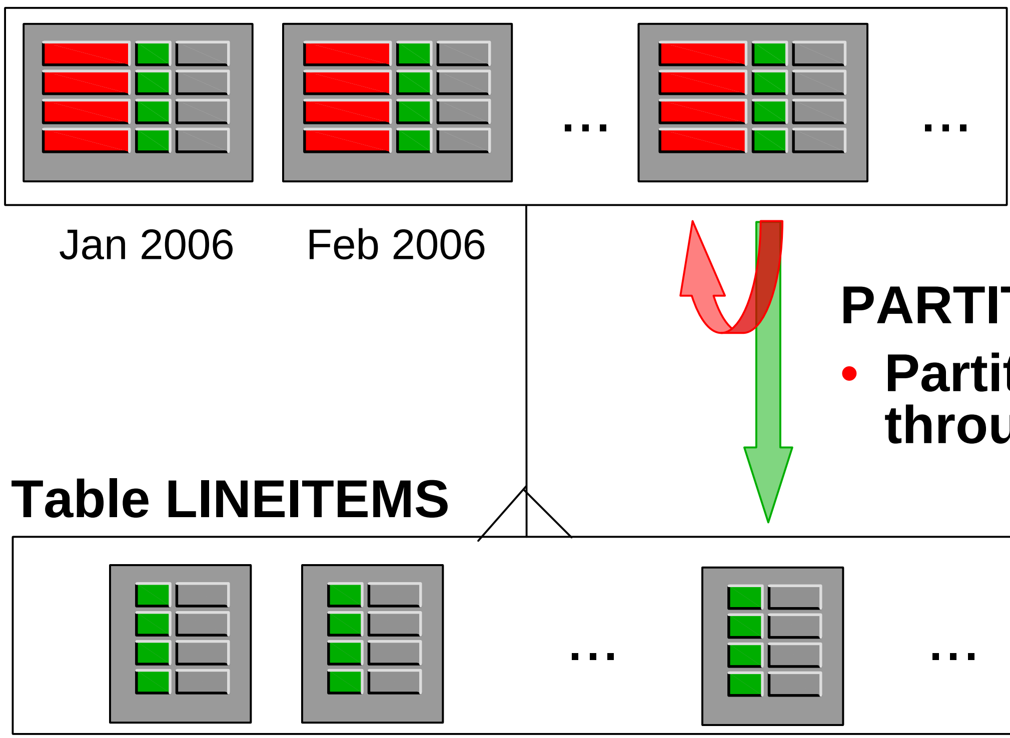
- RANGE(order_date)
- Primary key order_id

- Redundant storage of order_date
- Redundant maintenance

- RANGE(order_date)
- Foreign key order_id

REF Partitioning

Table ORDERS



- **RANGE(order_date)**
- **Primary key order_id**

PARTITION BY REFERENCE

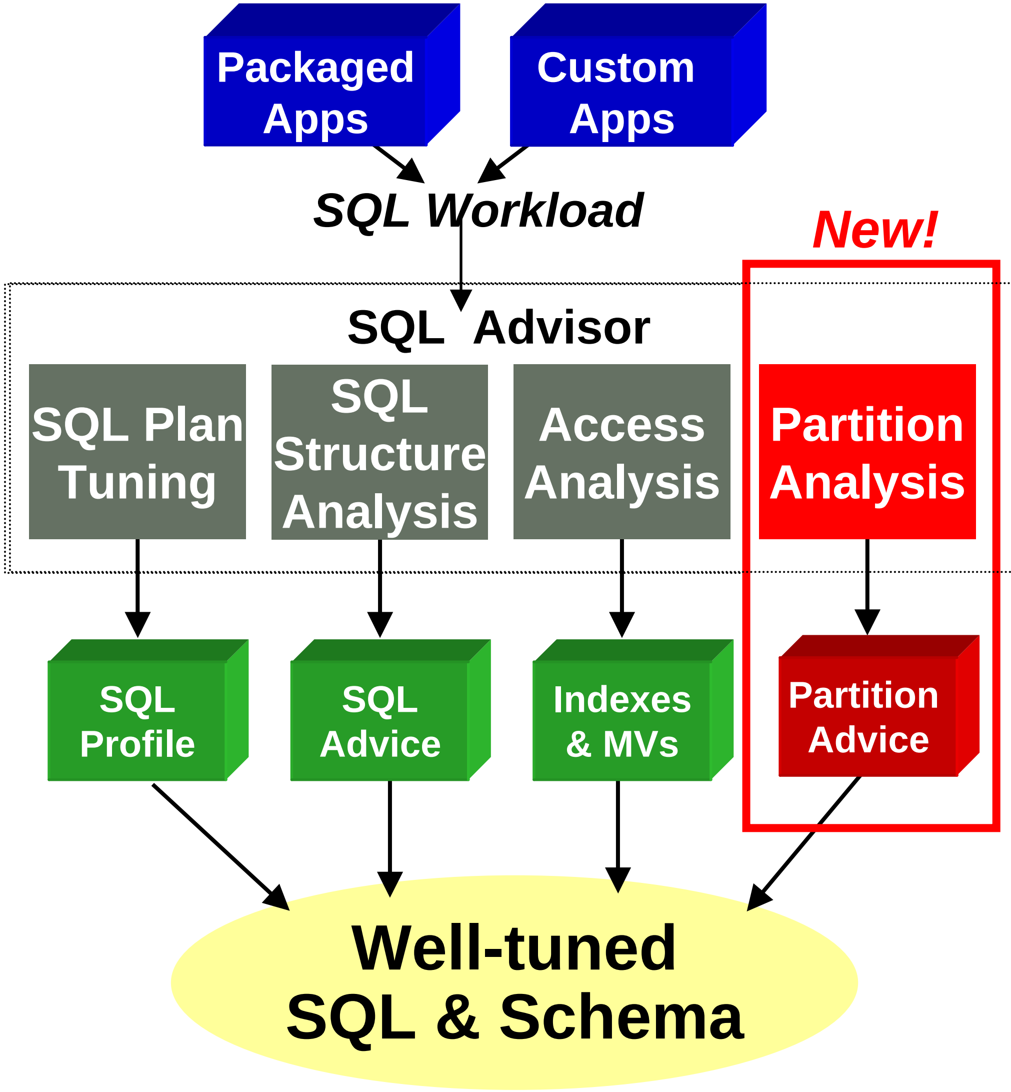
- **Partitioning key inherited through PK-FK relationship**

Table LINEITEMS

- **RANGE(order_date)**
- **Foreign key order_id**

Jan 2006 Feb 2006

Partitioning Advisor



- Considers entire query workload to improve query performance
- Advises on partitioning methods
  - Range (equal-interval), range key and interval
  - Hash, hash key
- Integrated, non-conflicting advice with Indexes, MVs

Q&A

Thanks For Coming !!

Daniel Liu Contact Information

Phone: (714) 376-8416

Email: daniel.liu@oracle.com

Email: daniel_t_liu@yahoo.com

Company Web Site:

http://www.oracle.com