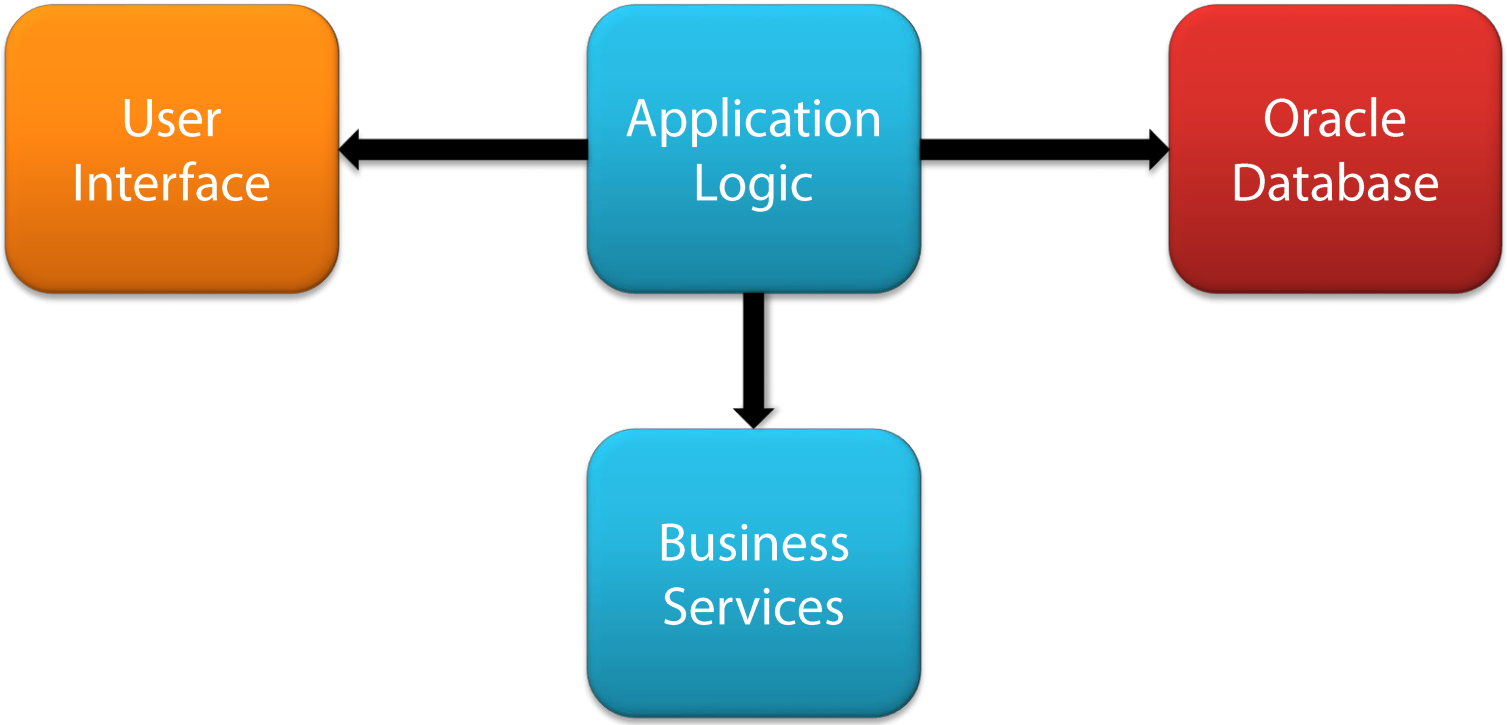
Oracle Performance Tuning for Developers

Why Performance Tuning Matters

David Berry
http://buildingbettersoftware.blogspot.com/

pluralsight
hardcore developer training

Typical Application Layers



Why Database Performance Matters

Responsive applications are more usable

Fast applications are more productive for users

Efficient applications use less hardware

Performance Tuning is Not Magic

"Oracle must be broken because my query ran fast last week!"

"I don't understand database performance. Its more art than science!"

"The database is just a black box. I don't know what happens over there!"

"Oracle performs poorly. I'm switching to SQL Server!"

Know What Instrument To Read

Oracle and Performance Data

Oracle is highly instrumented

Data available for almost every function

Learn what data is available

Enables an analytic approach to tuning

Who is Responsible for Performance Tuning?

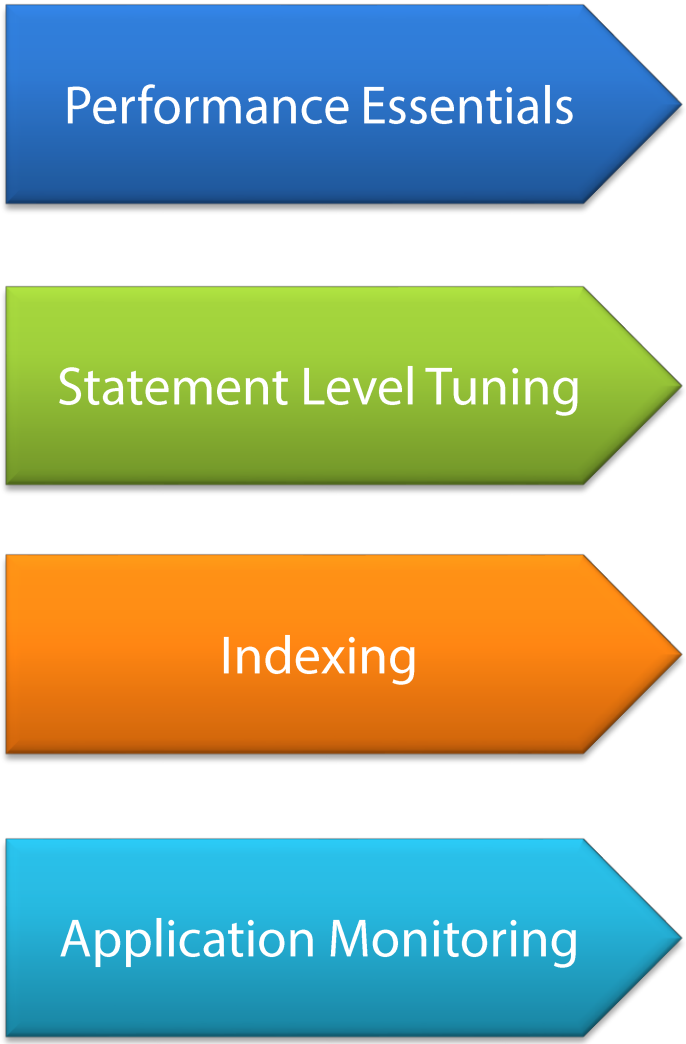
Application developers are knowledgeable about

- The composition of the data
- What the application is trying to do
- What the user's expectations are

Integrate performance tuning with development

- Design performance in from the start
- Benchmark components as they are finished

Course Overview



- Oracle architecture basics
- Performance metrics
- Connecting to Oracle

- Use of bind variables
- Execution plan evaluation

- What makes a good index
- What capabilities does Oracle provide

- What is happening real time in Oracle
- What statements should I target

Performance Tuning Approach

Understand the details of what Oracle is doing

- Why did Oracle make that decision
- What criteria did it evaluate

Don’t fly on autopilot

- Know how to interpret the data Oracle provides
- Understand how Oracle processes the SQL you write

Tools Used in This Course

Used In This Course

- Oracle SQL Developer
- Oracle SQL*Plus
- V$ dynamic performance views

Not Used In This Course

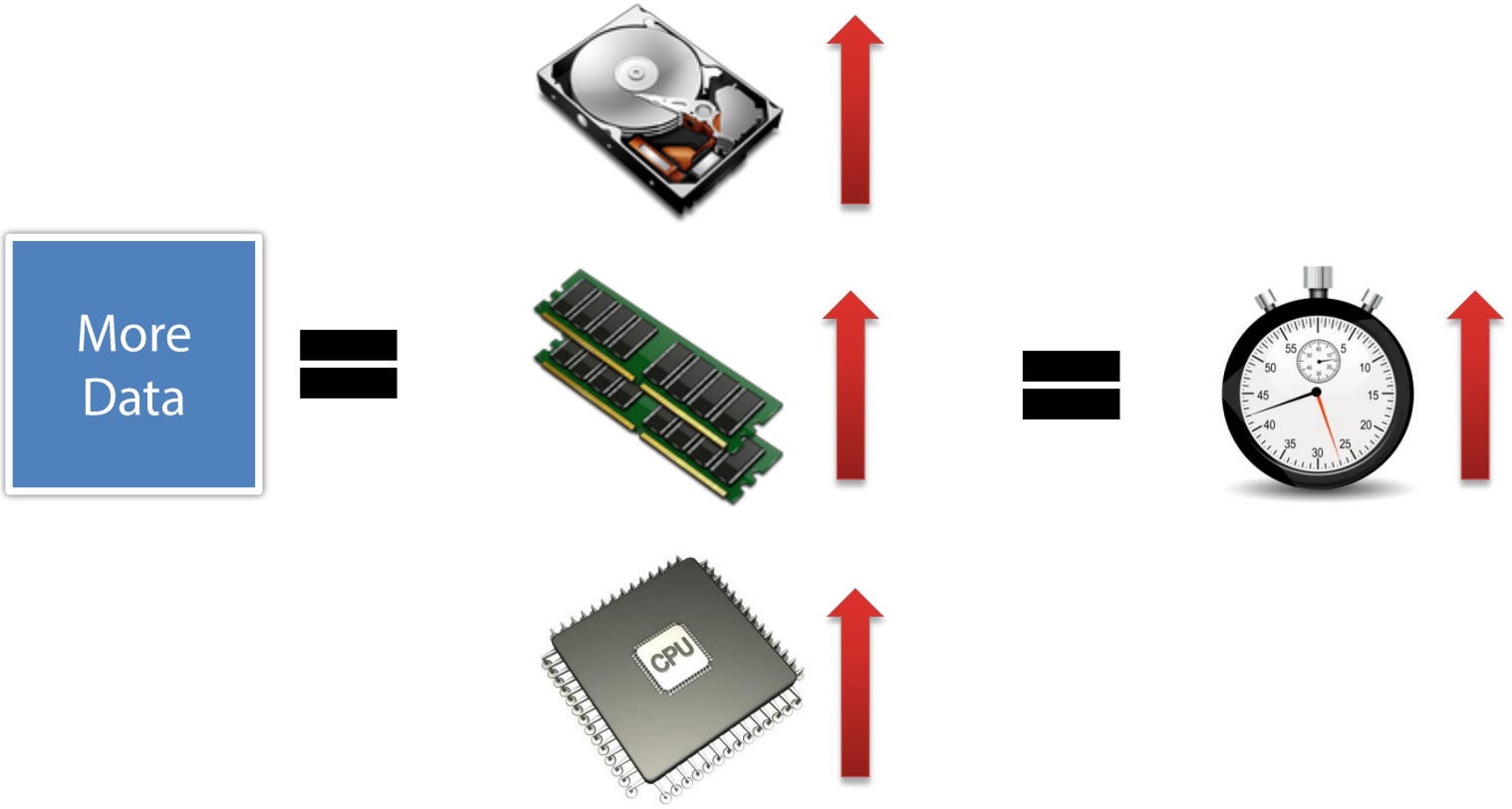
- SQL Tuning Advisor
- Automated Workload Repository
- Third party tools like TOAD

The Best Performance Tuning Tool

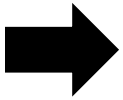
A knowledgeable professional like you

First Performance Principle

Focus on processing less data



Use Only The Resources You Need



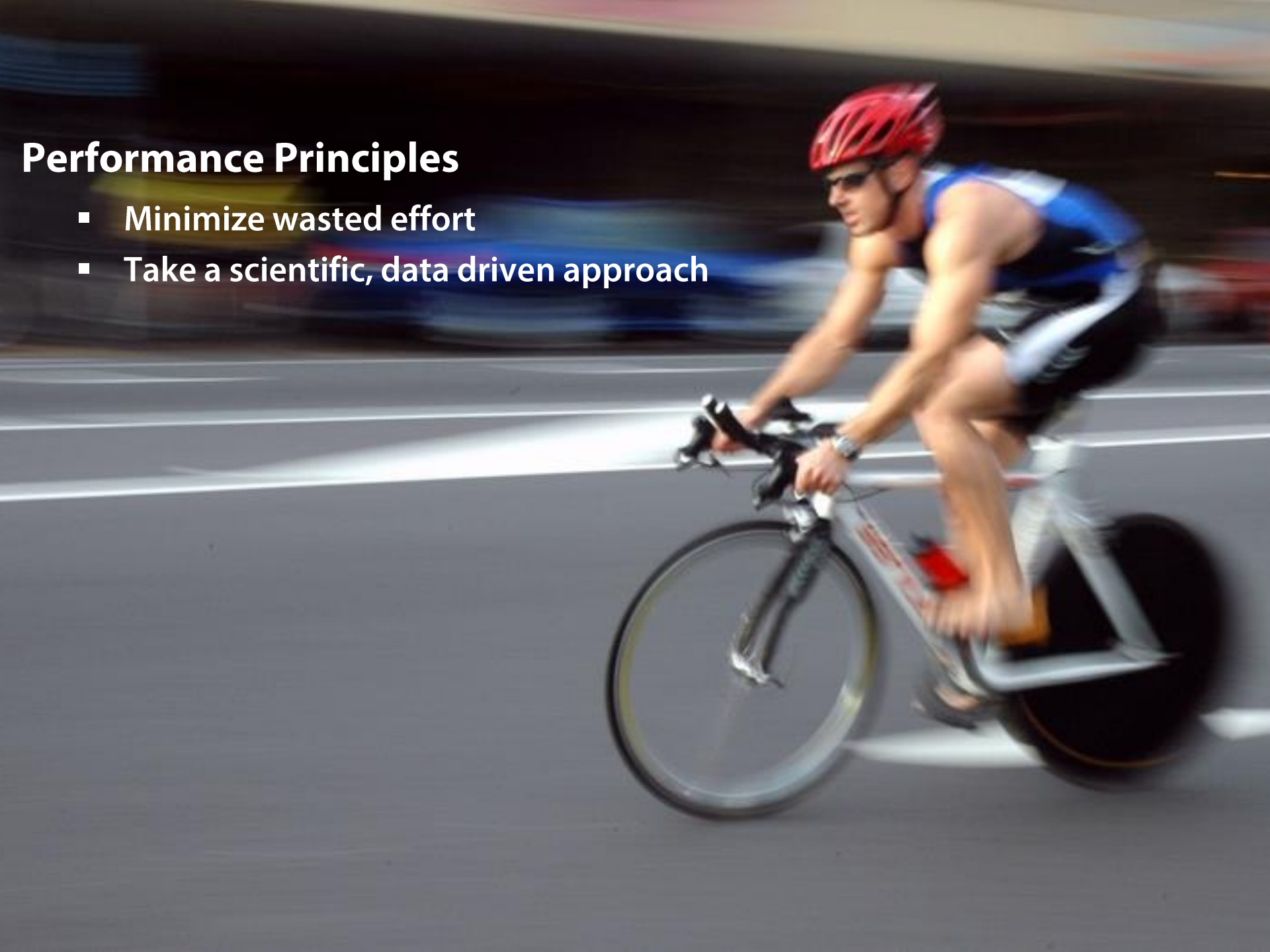
Base Your Actions on Data

Oracle provides a wealth of data

- Data indicates what the problem is
- Compare before and after data to understand a change

Have data drive the final decision

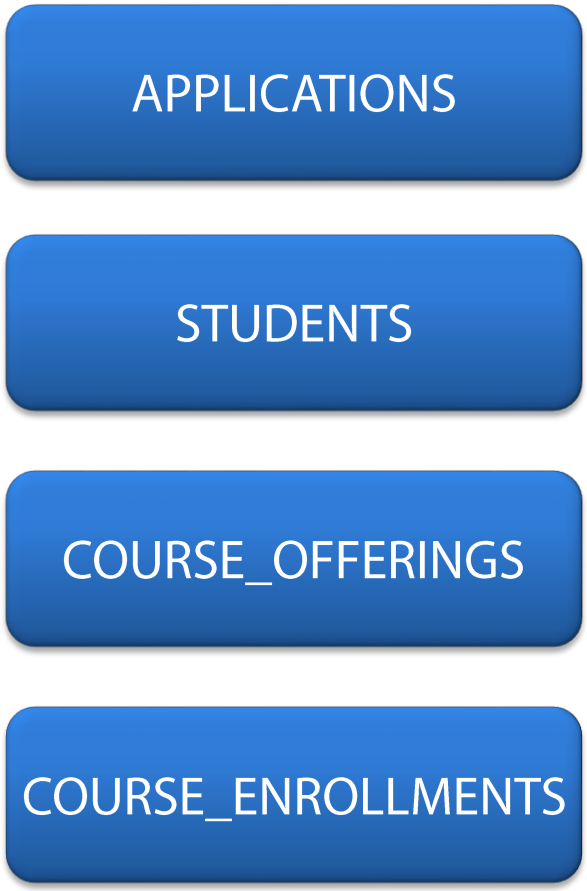
- Use cases may vary
- Database characteristics may vary
- Data reveals the best choice for your situation



Performance Principles

- Minimize wasted effort
- Take a scientific, data driven approach

Sample Database



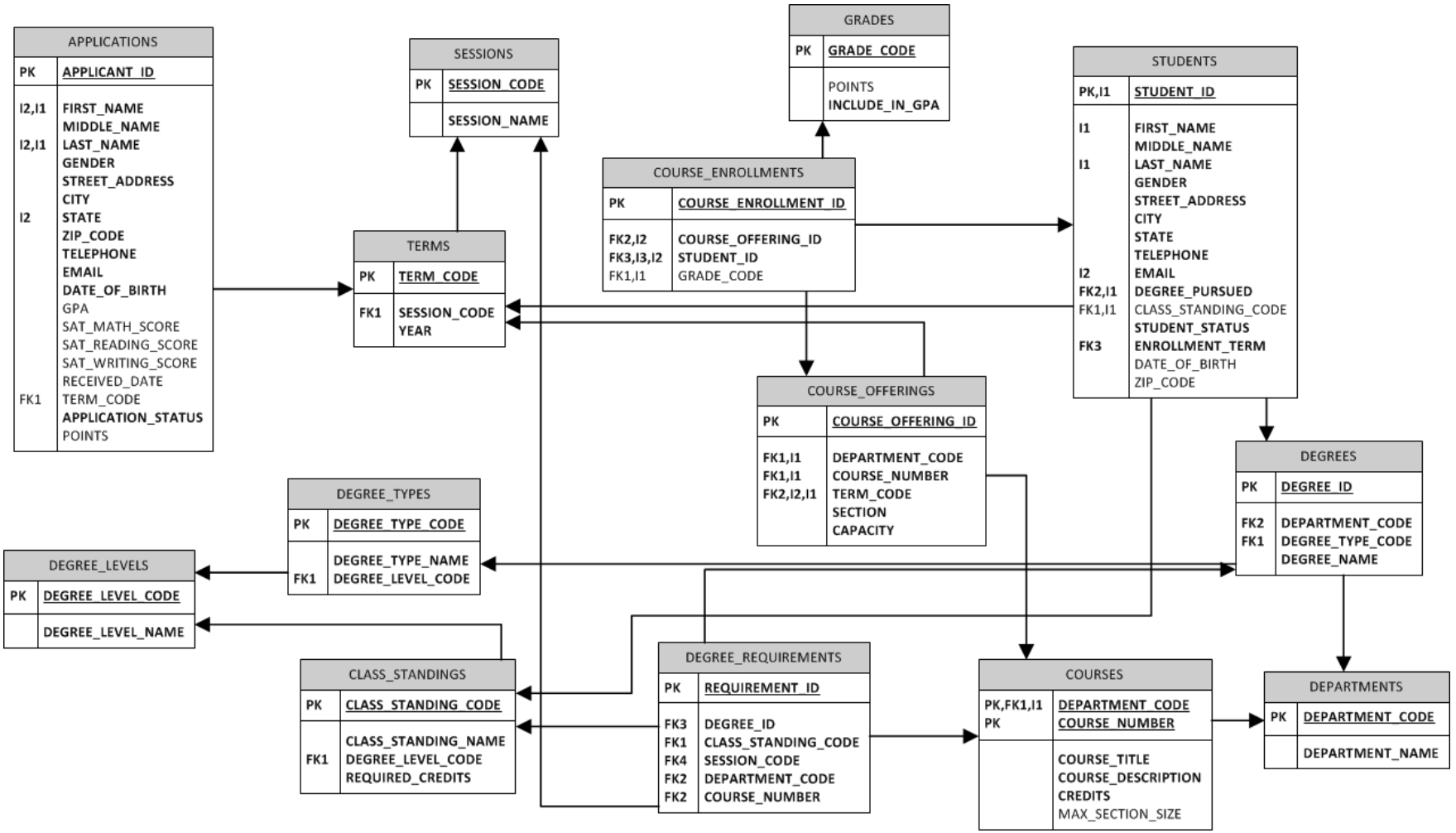
Students that applied to the university

Students enrolled in the university

Courses and sections offered each term

Course enrollment information for each student

ER Diagram



http://bit.ly/QCM126

Loading the Sample Data

1. **Create a schema (user) in Oracle to house the data (ex: student)**
2. **Create a directory on the machine running Oracle and put the input file in this directory**
3. **As a SYSDBA user in Oracle, run the following**
    1. CREATE OR REPLACE DIRECTORY import_dir AS <<your directory>>';
    2. GRANT READ, WRITE ON DIRECTORY import_dir TO <<your user name>>;
4. **Run the following command from the command prompt**

```
impdp <<your target user>> schemas=student
    remap_schema=student:<<your user name>>
    directory=import_dir
    dumpfile=student_data.dmp
    logfile=import.log
```

Oracle Architecture and Performance Basics

David Berry
http://buildingbettersoftware.blogspot.com/

Module Outline

Oracle Architecture and Memory Usage

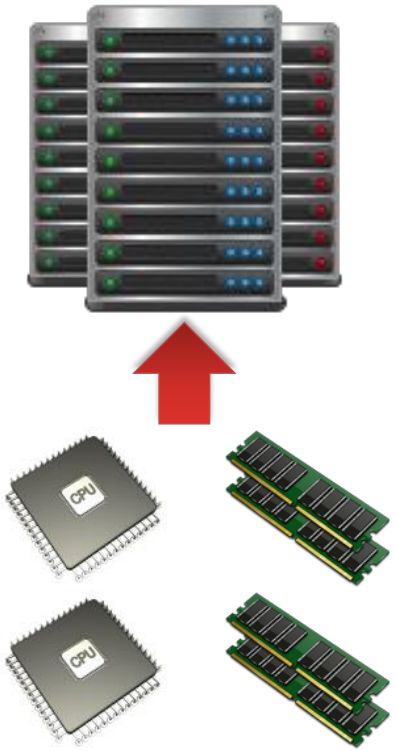
Performance Metrics

Performance Test Database and Concepts

Typical Line of Business Application Tiers

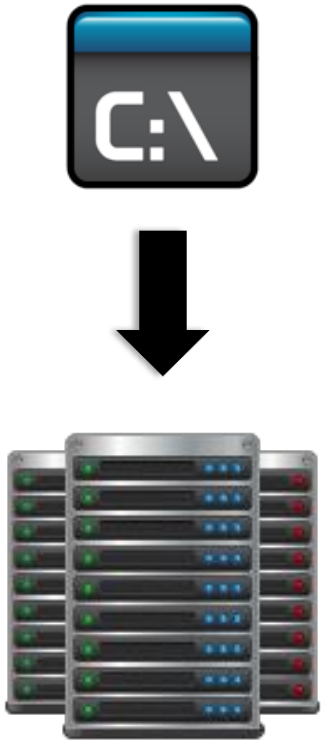
Web Servers/Application Servers

Database Server

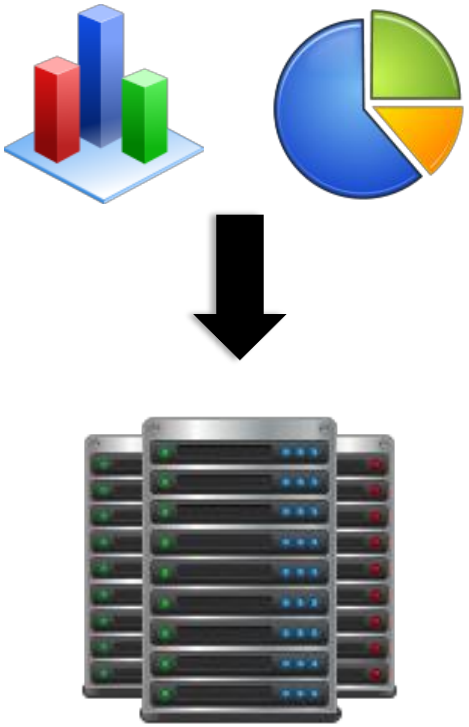


Other Application Contexts

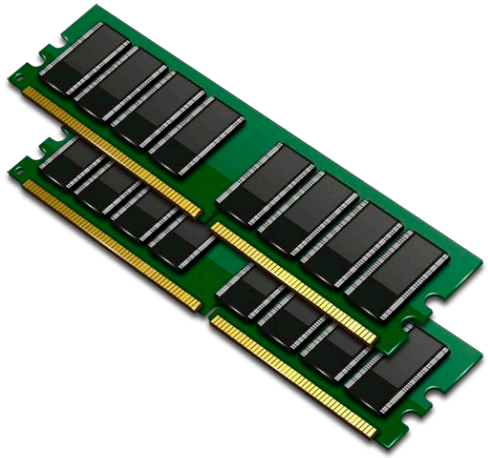
Batch Applications



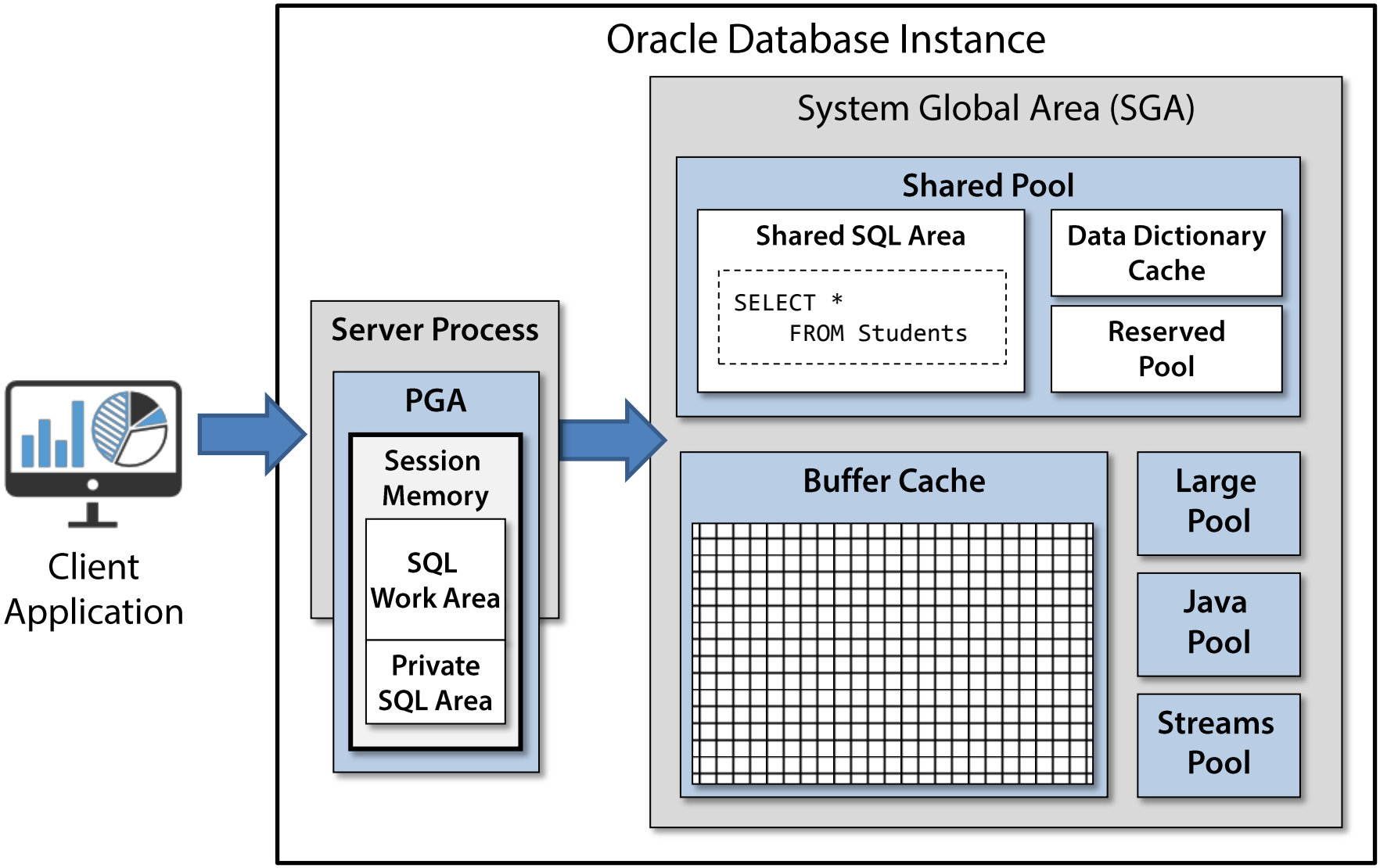
Business Intelligence Applications



Oracle Loves Memory!

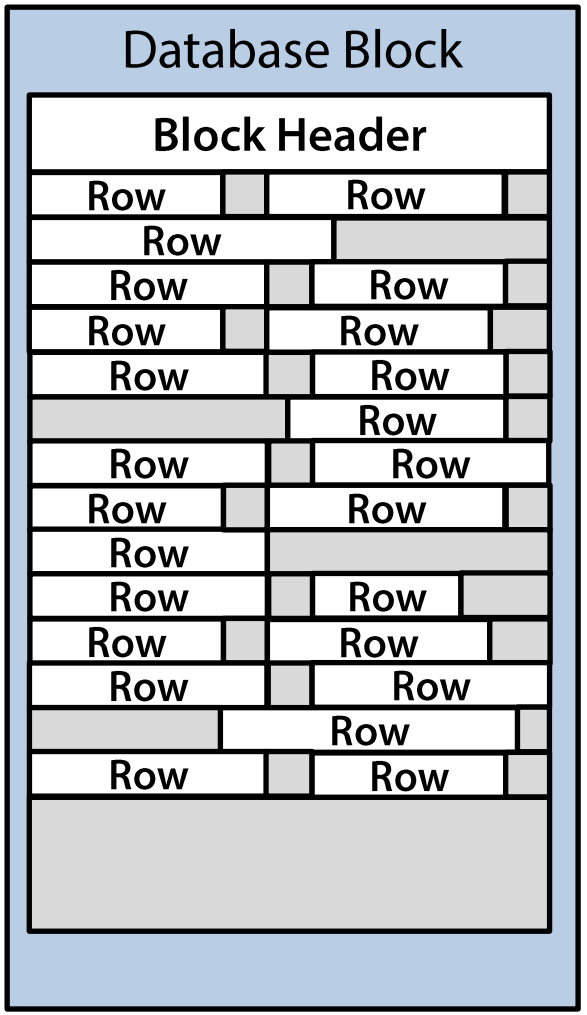
I  

Oracle Memory Architecture



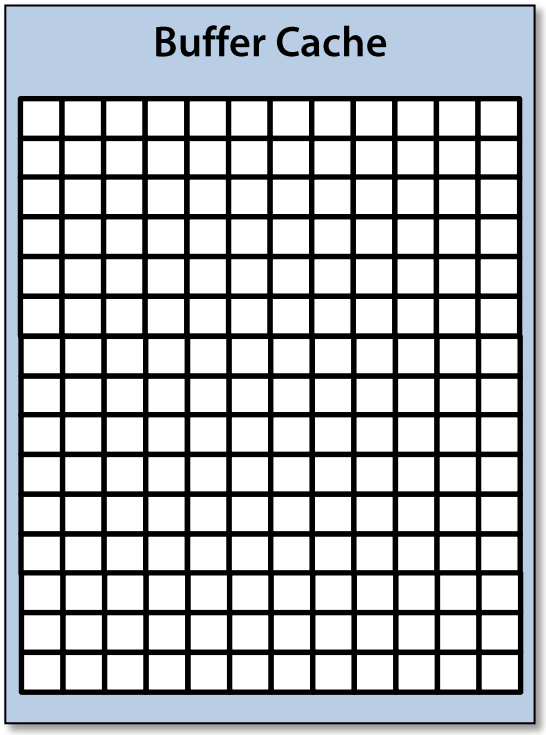
Database Block

- **Block sizes range from 2 KB to 32 KB**
  - 8 KB is typical
- **Block Header**
  - Block type information
  - Table information
  - Row directory
- **Row Data**
  - Some free space left at end of each row
  - Free space allows for insertion of new rows

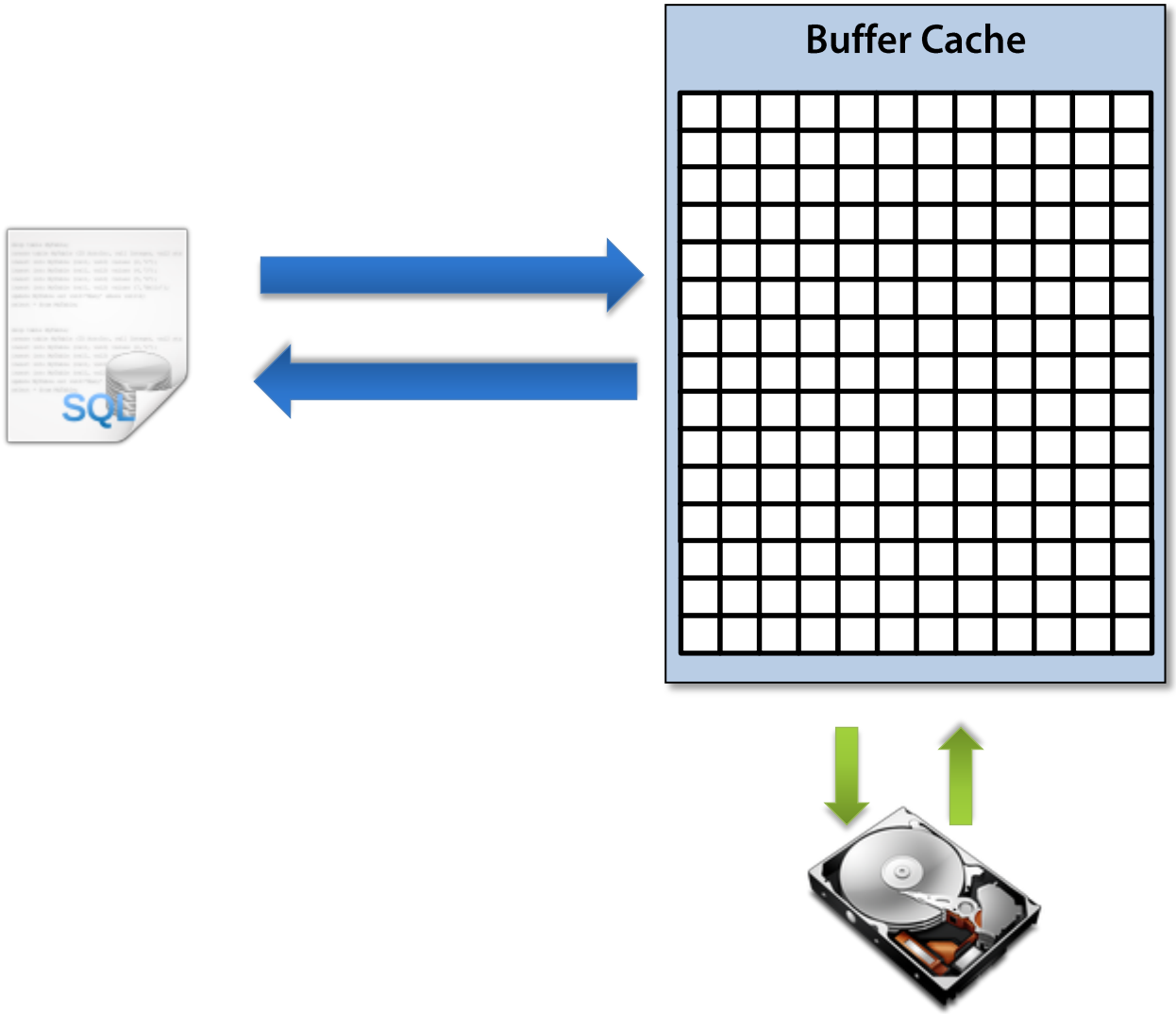


Buffer Cache

- All SQL statements need to access blocks of data
- Used to cache data blocks (both table and index) from disk
- Reading from memory much faster than disk
- Largest component of the SGA

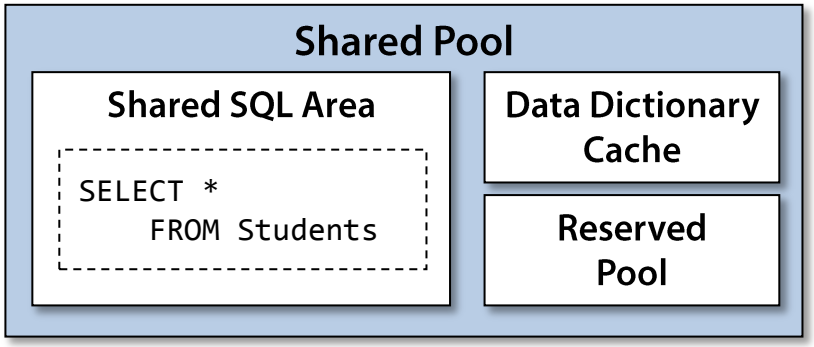


Reading Blocks Into the Buffer Cache



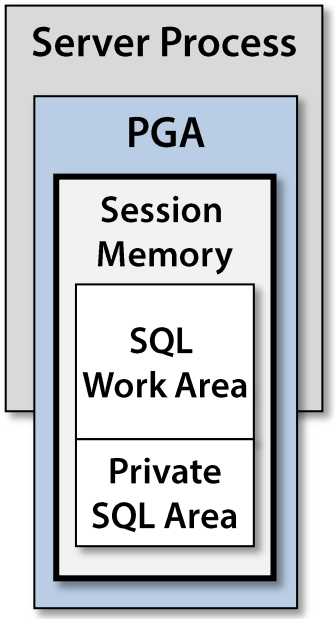
Shared Pool

- **Data Dictionary Cache**
  - Database object definitions
  - Object permissions
- **Shared SQL Area**
  - Cached SQL statements and execution plans
- **Reserved Pool**
  - Used to borrow memory for short term operations



Program Global Area

- **Memory assigned to a process when it connects to Oracle**
- **SQL Work Area**
  - Sorting data
  - Creating hash tables for hash joins
- **Private SQL Area**
  - Values of bind variables
  - Query execution state information



Oracle Architecture Takeaways

- **Caching is extensively used**
- **Write applications that take advantage of caching**
  - Prefer memory operations over disk operations
- **Oracle Concepts Guide**
  - PDF - http://bit.ly/1eSsBQf
  - HTML - http://bit.ly/1p9C4Wx

Performance Metrics Introduction

Goal: Write faster and more efficient data access code

What performance metrics should we look at?

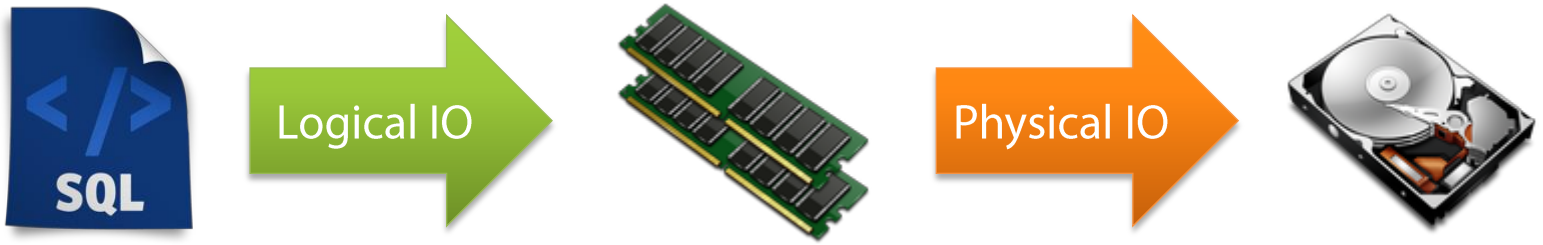
Elapsed Time

- How long did a statement take to execute
- What your user feels
- Treat as an end result
- Product of other metrics
- Impacted by external factors

Logical IO Operations

When a data block is in the buffer cache

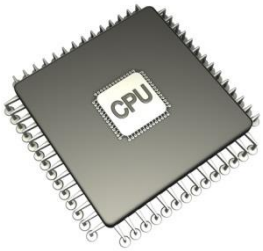
When a data block is not in the buffer cache



Why Logical IO Matters

Impacts Physical IO

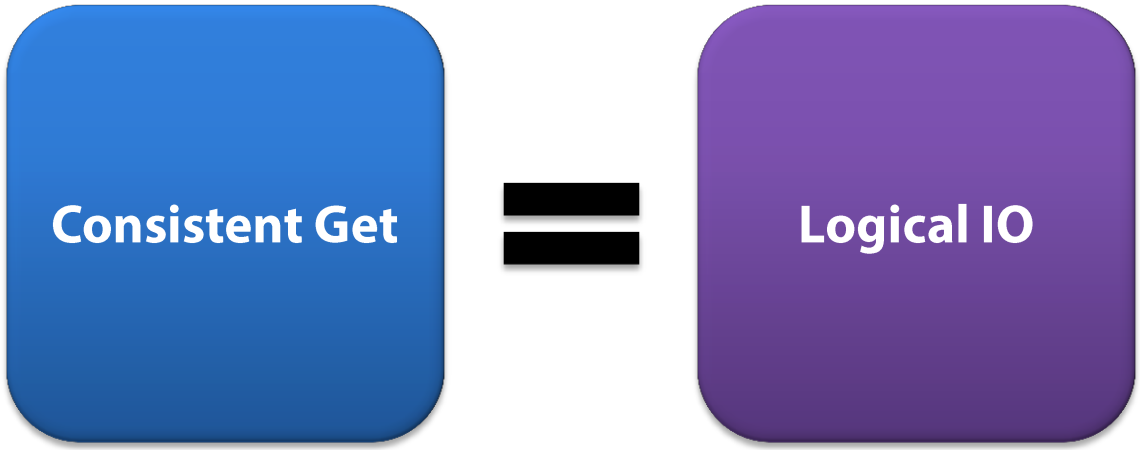
- **More likely need to read data from disk**
- **Higher probability of reading more blocks**



Impacts CPU

- **More data to filter, join and sort**
- **More data means more CPU**

Terminology Around Logical IO Operations



Measuring Logical IO Operations

Autotrace

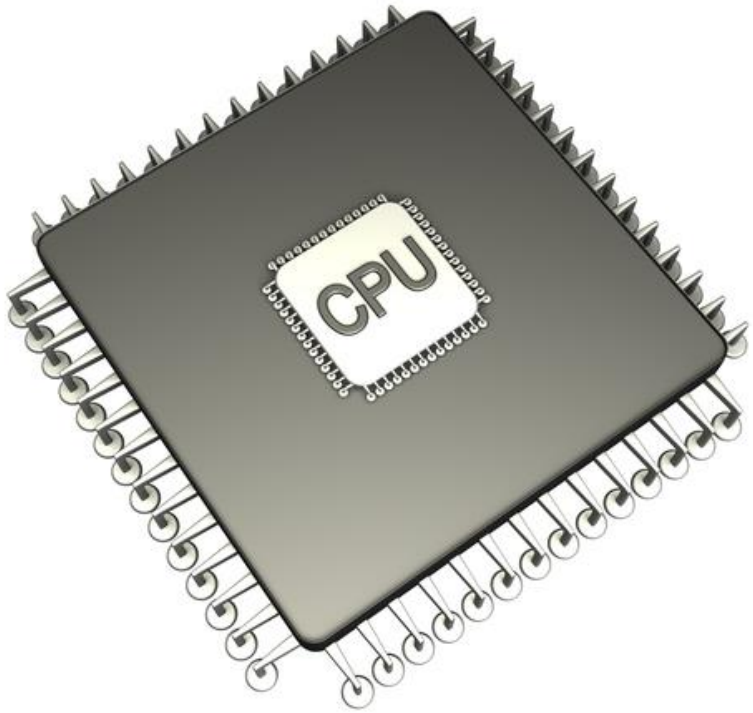
- Available in SQL*Plus and SQL Developer
- Develop performance baseline

V$ Views

- V$SqlStats summarizes all statements
- Identify which statements are most costly

CPU Time

- Oracle database servers are busy
- 100% CPU usage will lead to bottlenecks



CPU Usage in Oracle

Statement Parsing

- Available in SQL*Plus and SQL Developer
- Develop performance baseline

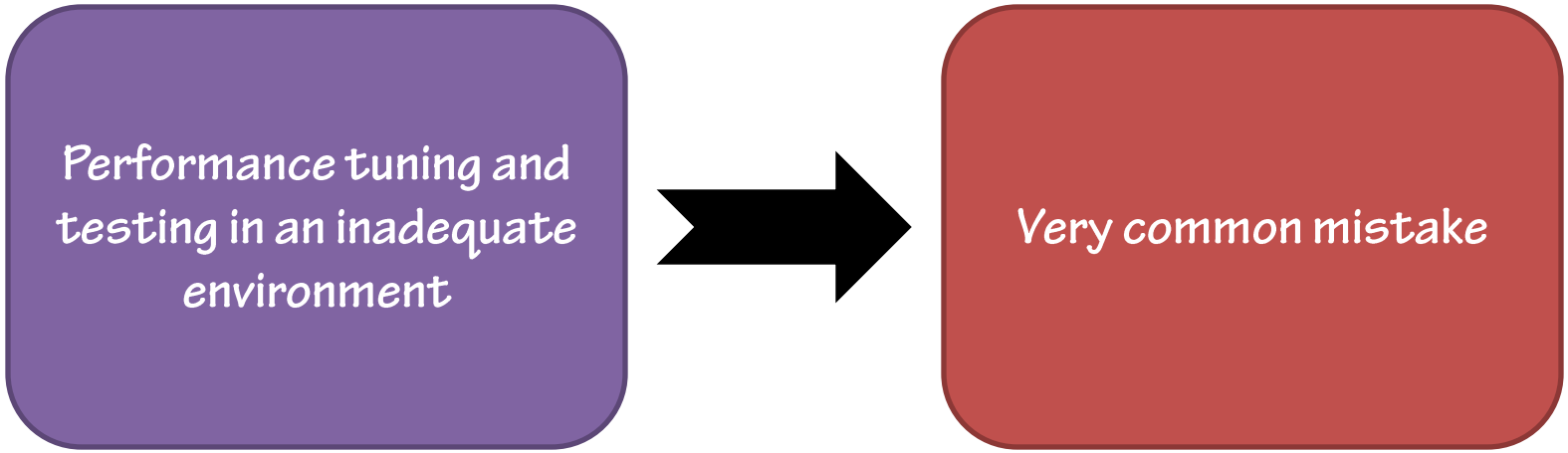
Statement Processing

- Filtering, joining and sorting data
- Driven by amount of data to process

Performance Metrics Summary

- Use metrics other than elapsed time to guide tuning efforts
- Logical IOs help to understand statement efficiency
- Think in terms of statement efficiency

Environments for Performance Tuning and Testing



Data Size and Test Regions

Metric	Development	Test	Production
Database Size			
SQL Statement Performance	Fast	Fast	Uh-oh!

What Happened

- Hardware is fast enough that every statement runs fast in a small database

- Oracle used a different execution plan to execute your statement between the regions

Imperative Programming Paradigm

- Define the sequence of steps a program should follow in order to accomplish a task
- Algorithm remains same for any input

Declarative Programming Paradigm

- Express what the outcome of the program should be without defining the algorithm to follow
- Different algorithms may be used for different input data

Implications

- **Have a test region that mirrors production**
  - Amount of data contained
  - Distribution of data
- **Differing environments will produce different answers**
- **Advantages of an effective test environment**
  - Results will translate to production
  - Evaluate performance solutions early
  - Ability to reproduce production performance issues

Building a Test Environment

Copy data from production

- Can use tools like Oracle Data Pump
- Known to match your distribution
- Be sure to scramble sensitive information

Data generation tools

- Eliminates information disclosure risks
- Generate any amount of data
- Be sure to get the distribution correct

Performance Test Environment Uses

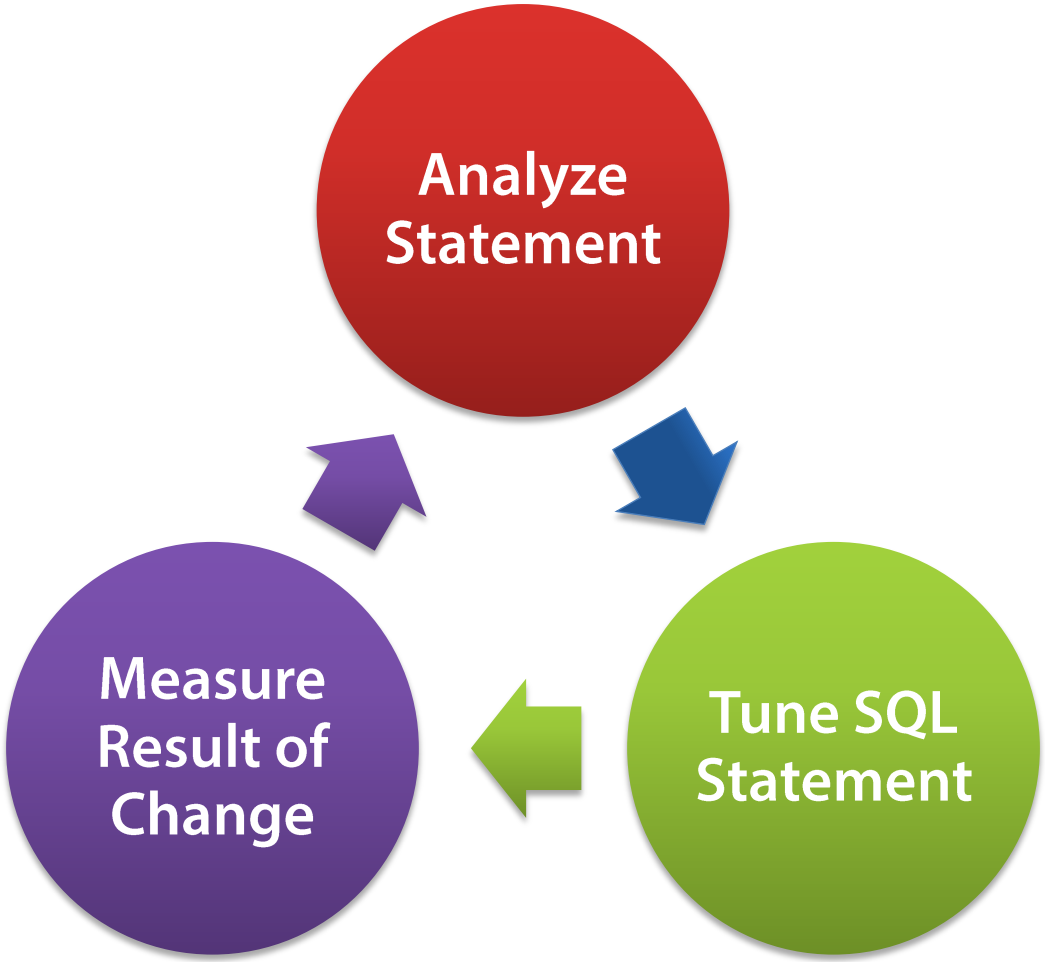
Analyze individual SQL statements

- Determine execution plan
- Develop performance baseline

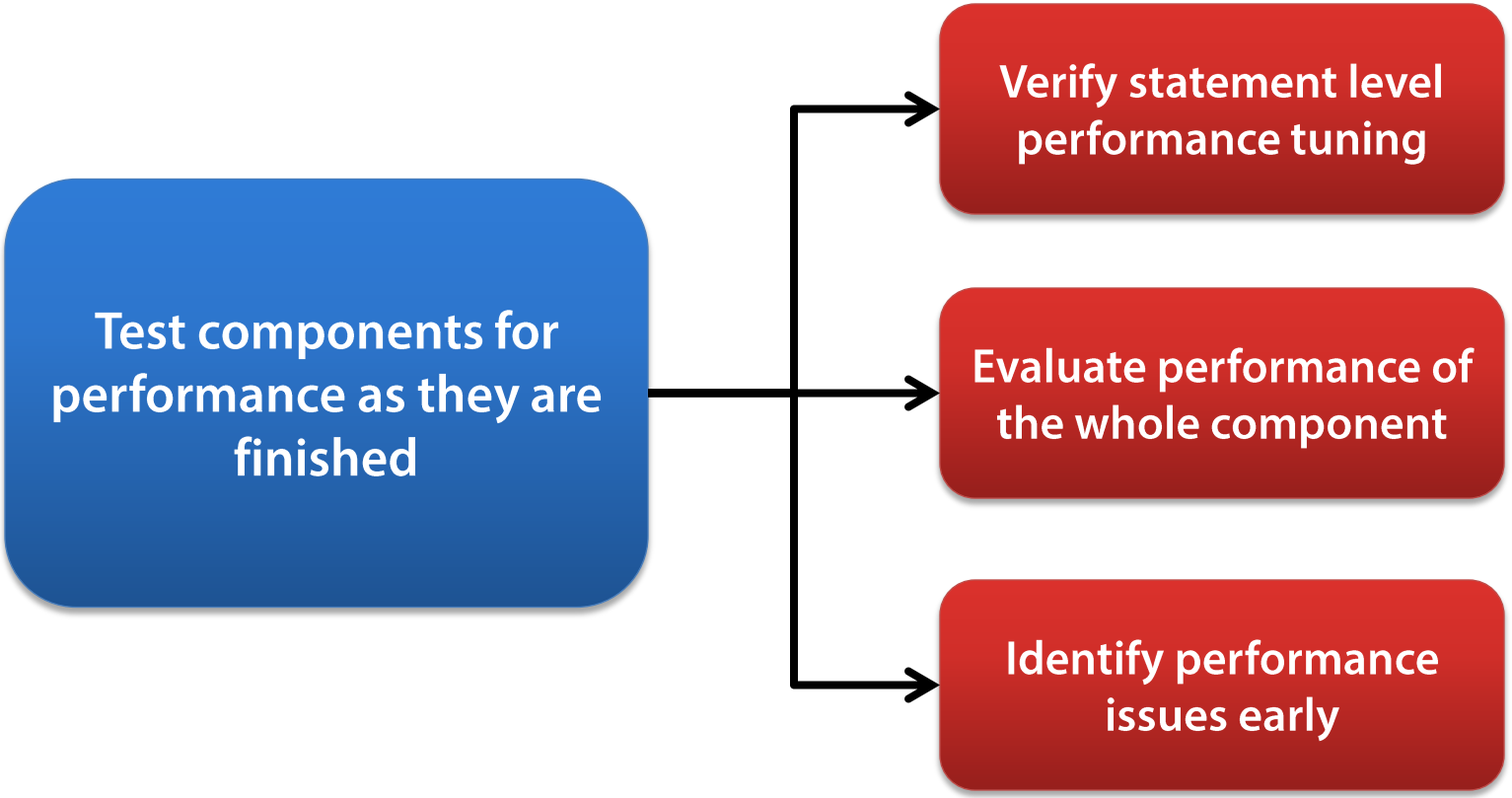
Tune statements as needed

- Rewrite parts of the statement
- Add/modify indexes
- Evaluate application design choices

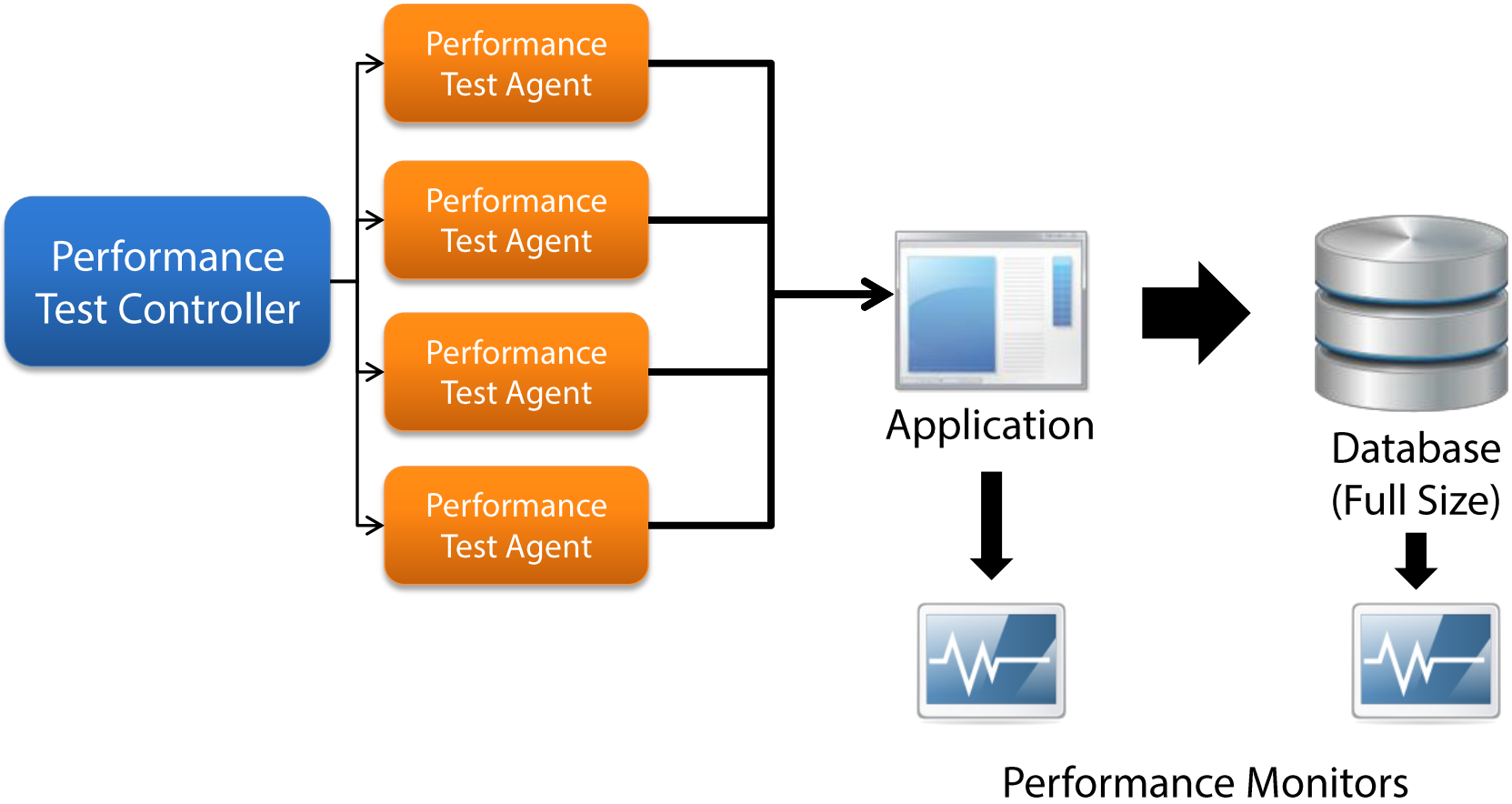
Performance Tuning Process



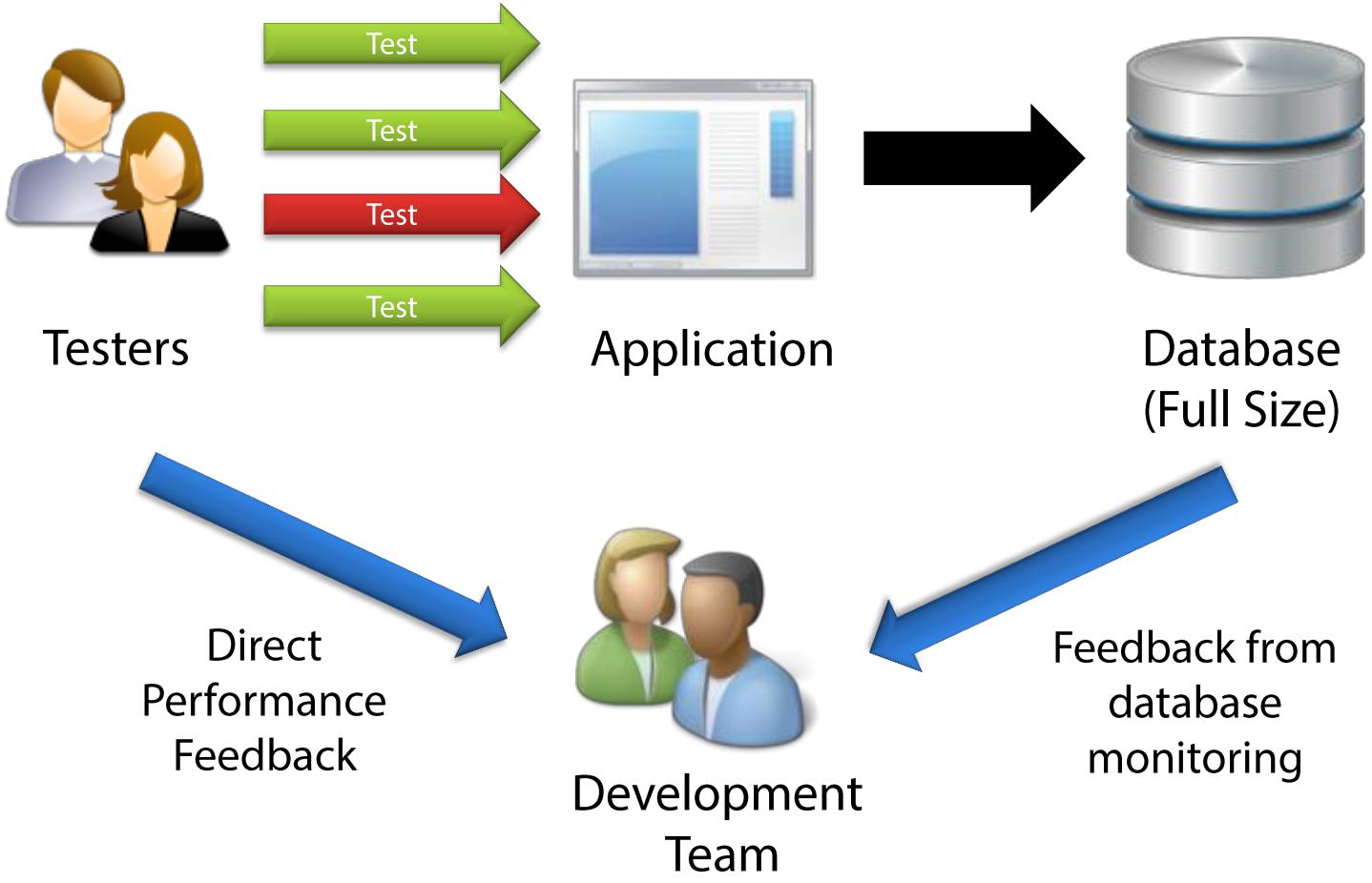
Component Level Performance Testing



Formal Performance Testing



Informal Performance Testing



Proactive vs Reactive Performance Tuning

Proactive tuning

- Conducted during application development
- Preventative in nature

Reactive tuning

- Troubleshooting a production performance issue
- Can be urgent or chronic in nature

Test all changes in a test environment first

- Minimize impact on production users
- Some indexes can be time consuming/resource intensive to build

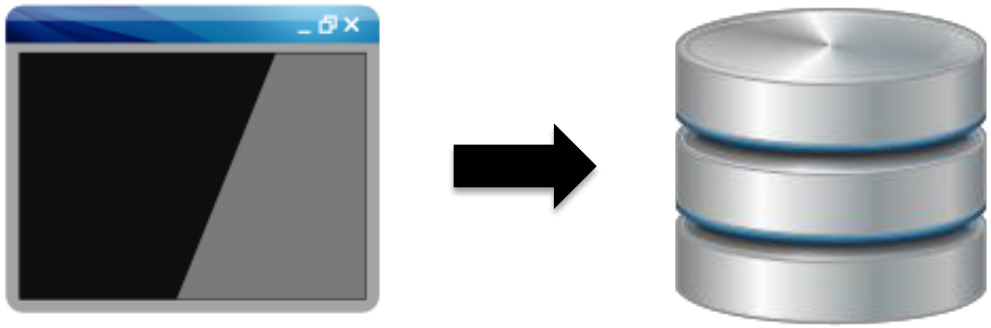
Performance Testing Environment Summary

- **Use a full size database for performance tuning and testing**
  - Analyze SQL statements during development
  - Test components as the are completed
- **Include performance testing early in the project**
  - Identify and attack risks early
- **Use a full size database during test phases**
- **Well performing apps are more usable**

Connections and Connection Pooling

David Berry
http://buildingbettersoftware.blogspot.com/

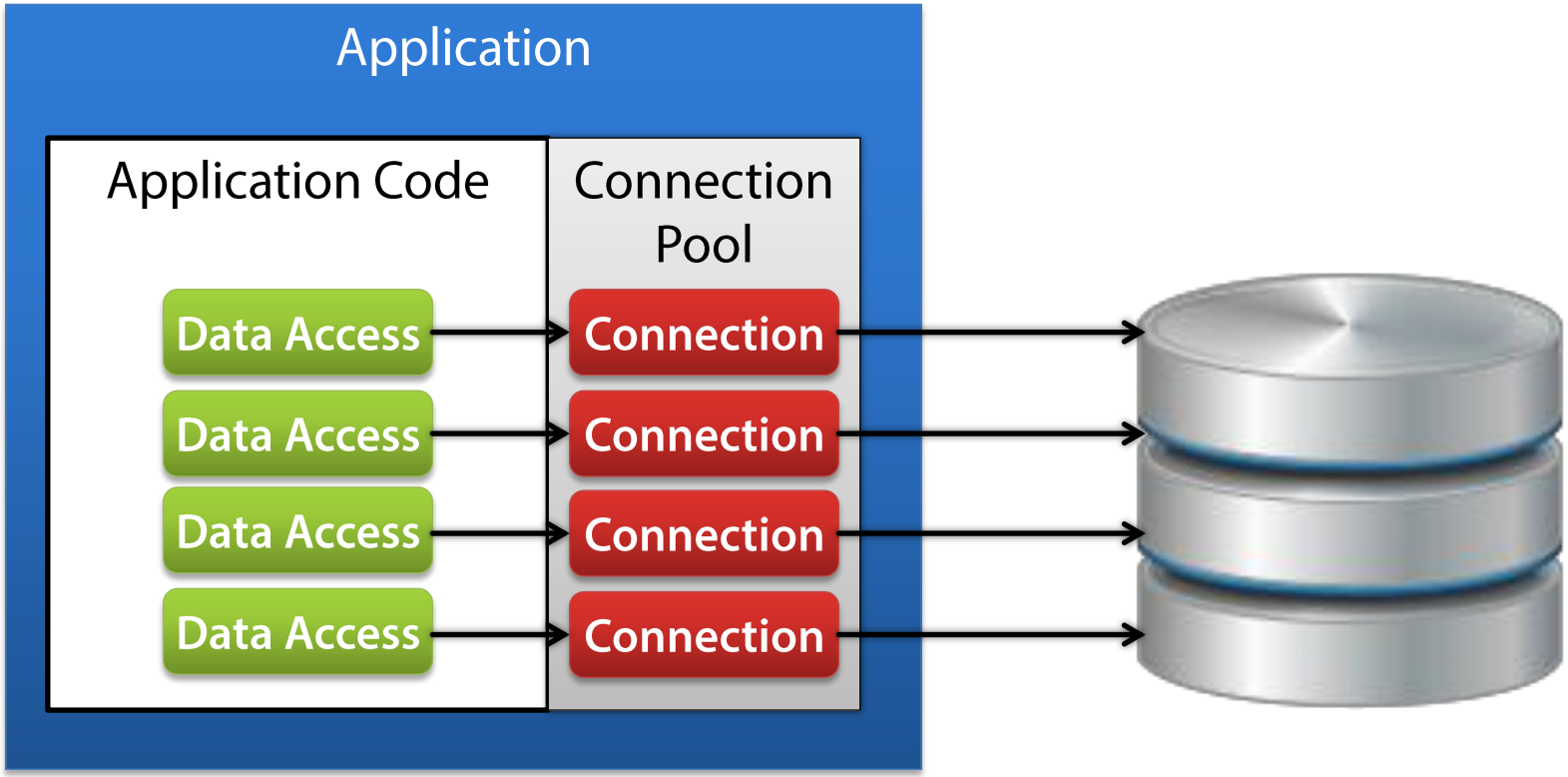
Connecting to Oracle



Your Application

Oracle

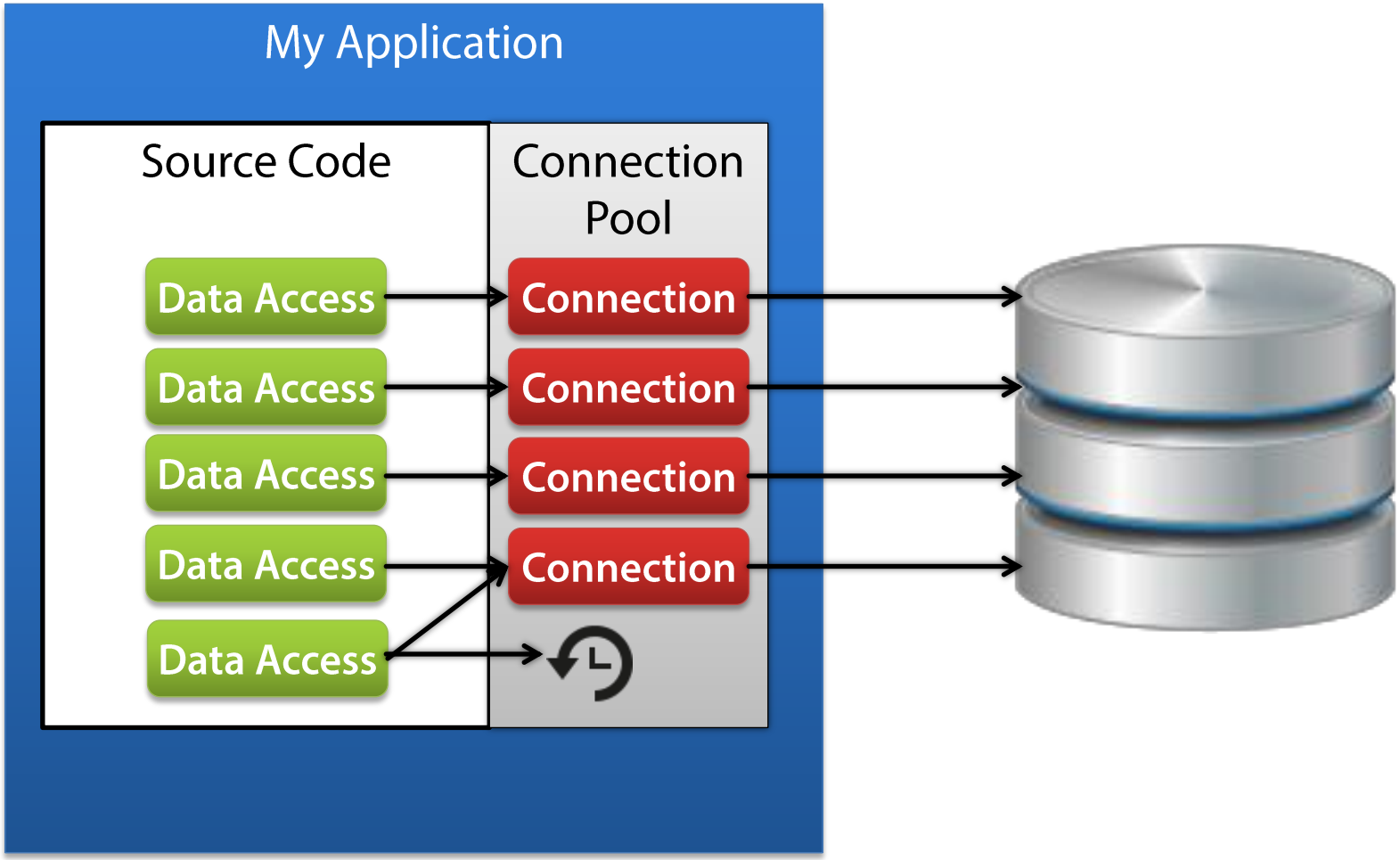
Connection Pooling Example



Connection Pool Mechanics

- A connection pool is established for each connection string
- Each connection pool has a minimum and maximum size
- More connections are created when all existing connections are in use
- Number of connections in the pool fluctuates depending on the need for connections

Connection Pool at Maximum Size



Using Connection Pooling in .NET

```
 OracleConnection con = null;
 OracleCommand cmd = null;
 try
 {
     con = new OracleConnection(connectionString);
     con.Open();
     // More code here …
}
catch (Exception ex)
{
    // Handle exception
}
finally
{
    if (con != null) con.Close();
}
```

ODP.NET Driver Connection Pool Parameters

Parameter	Description	Default Value
Pooling	Turns connection pooling on or off	true
Min Pool Size	Minimum size of the pool	1
Max Pool Size	Maximum size of the pool	100
Connection Timeout	Maximum time in seconds to wait for a free connection	15
Incr Pool Size	Number of connections to create when all connections are in use	5
Decr Pool Size	Number of connections to be closed when connections are unused	1

Connection Pooling in Java

- Oracle Universal Connection Pool library
- Download from http://bit.ly/1mzDlYc
- Java doc located at http://bit.ly/1fw1nM0

Connection Pool Initialization in Java

```
public static PoolDataSource InitializeConnectionPool(String poolName,
    String url, String username, String password)
{
    // Create a new PoolDataSource using the factory
    PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();

    //Set up the connection info
    pds.setConnectionPoolName(poolName);
    pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
    pds.setURL(url);
    pds.setUser(username);
    pds.setPassword(password);

    //Setting initial connection pool properties
    pds.setInitialPoolSize(5);
    pds.setMinPoolSize(1);
    pds.setMaxPoolSize(100);

    return pds;
}
```

Using the Connection Pool in Java

```
String poolName = "My Pool";
PoolDataSource pds = GetPoolDatasource(poolName);
Connection con = null;
try
{
    // Get a connection from the pool
    con = pds.GetConnection();

    // Do something interesting with the connection
}
catch (Exception ex)
{
    // Handle any exceptions
}
finally
{
    con.close();
}
```

Keep Data Access Code Focused

```
OracleConnection con = null;
OracleCommand cmd = null;
OracleDatereader = null;
try
{
    con = new OracleConnection(connectionString);
    con.Open();

    reader = cmd.ExecuteReader();
    while (reader.Read() )
    {
        // Map the result set
        // Do some business logic
        // Call a web service to look some things up
        // Perform some file access
        // Oh yeah, the data …
    }
}
```

Effects of Holding Connections Too Long

- **Treat database connections as a scarce resource**
  - Limited number available
  - Expensive to create new ones
- **If all existing connections are in use**
  - Next request will open an additional physical connection
  - Additional PGA memory required on the Oracle server
- **If the pool is at maximum size**
  - Threads will block waiting for connections to come available
  - Root cause is other threads holding connections too long

Using Multiple Connections

```
public List<Course> GetCourses(string termCode)
{
    using (OracleConnection con = new OracleConnection(conString))
    {
        con.Open();
        // Query Courses
        Department dept = this.QueryDepartment(departmentId);
    }
}
```

```
public Department QueryDepartment(string deptId)
{
    using (OracleConnection con = new OracleConnection(conString))
    {
        con.Open();
        // Query the department by ID
    }
}
```

Problems With Using Multiple Connections

- Need to open additional physical connections
- Higher probability of reaching max connection pool size

Reuse the Same Connection

```
public List<Course> GetCourses(string termCode)
{
    using (OracleConnection con = new OracleConnection(conString))
    {
        con.Open();
        // Query Courses
        Department dept = this.QueryDepartment(departmentId, con);
    }
}
```

```
public Department QueryDepartment(string deptId, OracleConnection con)
{
    using (OracleCommand cmd = new OracleCommand(con))
    {
        // Code Query the department by ID
    }
}
```

Connection Pooling Summary

- **Connection pooling saves time**
  - Faster to reuse existing pooled connections
  - Easy to use
- **Keep connections only as long as you need them**
  - Makes sure free connections are always available
- **Use one connection when possible**
  - Take advantage of multiple active result sets
  - Prevents unnecessary connections from being created
- **Keep minimum pool sizes small**
  - Allow idle connections to be closed
  - Save memory on the Oracle database server

Bind Variables

David Berry
http://buildingbettersoftware.blogspot.com/

Bind Variables Introduction

- **Using bind variables is very important for Oracle performance**
  - Often overlooked aspect but easy to implement
- **Parsing a SQL statement uses resources on the database server**
  - Using bind variables allows reuse of cached execution plans
  - Reuse of execution plans saves resources, especially CPU
- **Excessive parsing of SQL will cause contention in Oracle's shared memory structures**
  - This contention will limit the scalability of your applications and hinder their responsiveness
- **Bind variables also provide protection against SQL injection attacks**

Life of a SQL Statement

Parse Phase

Check SQL Syntax
Check Object Permissions
Query Rewrite Process

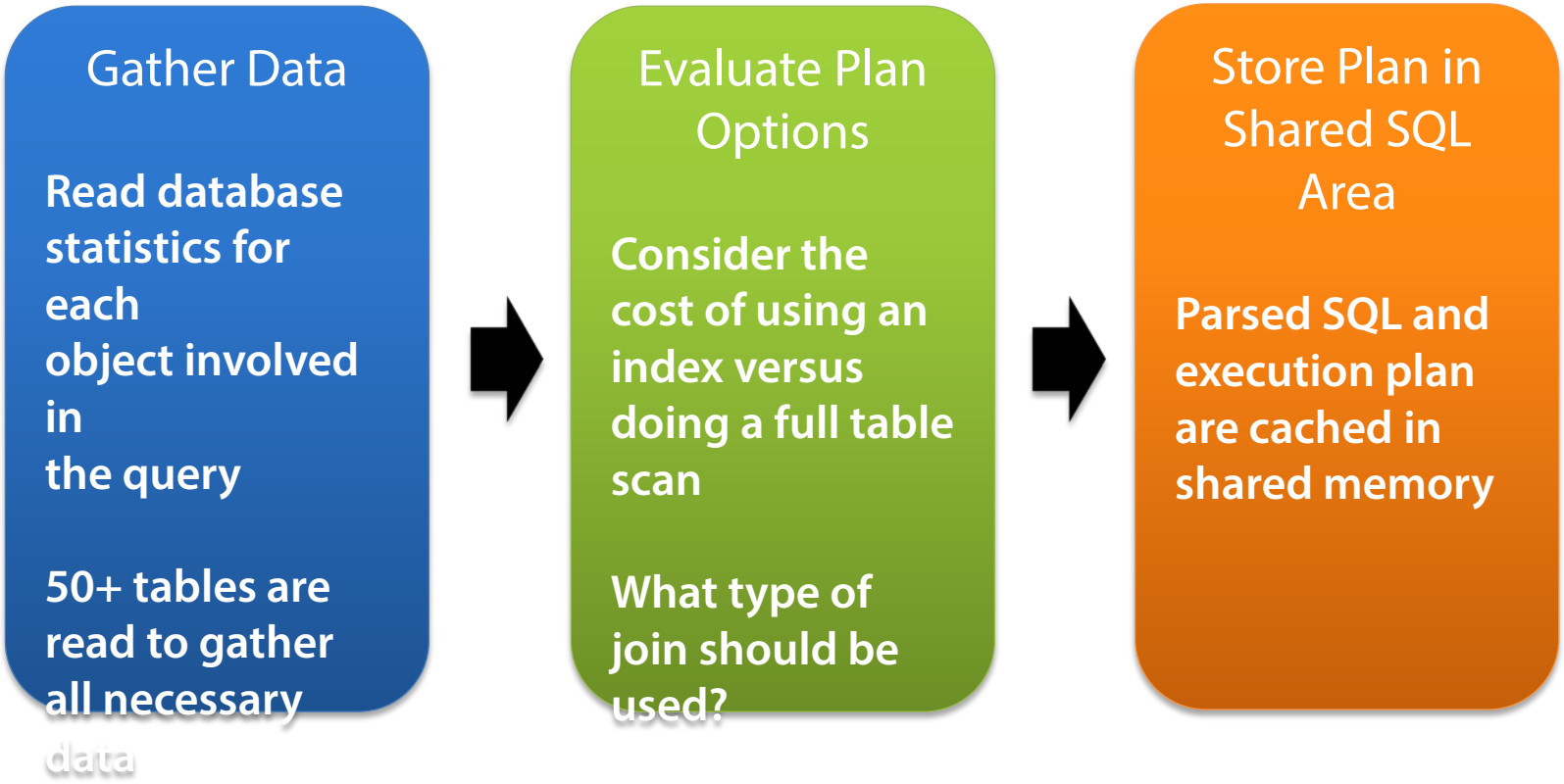
Query Optimization

Evaluate Statistics
Create Execution Plan

Execution Phase

Read Blocks
Filter Rows
Sort Data

Steps in Creating an Execution Plan



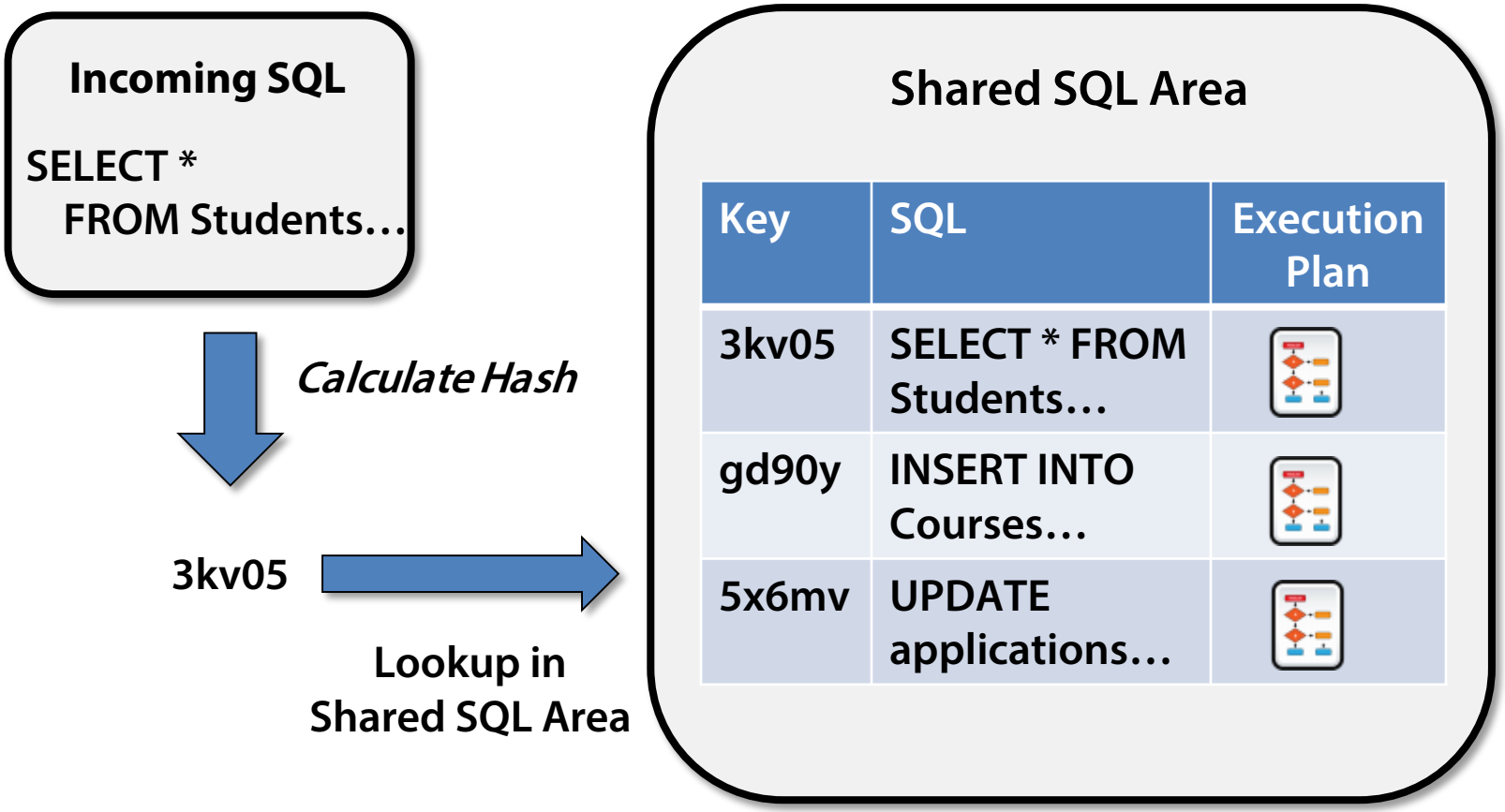
The Shared SQL Area

From the Oracle Optimizer

Shared SQL Area

Key	SQL	Execution Plan
3kv05	SELECT * FROM Students…	
gd90y	INSERT INTO Courses…	
5x6mv	UPDATE applications…	

Plan Lookup in the Shared SQL Area



Why Cache Execution Plans

- **Most applications contain relatively few unique SQL statements**
  - Even the largest applications usually contain only a few hundred distinct SQL statements
  - Each of these SQL statements is executed over and over, just with different parameters each time

- **Impacts of caching SQL statements and execution plans**
  - Resources on the database server are conserved because the same operation doesn't have to be performed over and over
  - Looking up a execution plan in the Shared SQL Area is faster than generating a plan from scratch

Soft Parses Versus Hard Parses

Soft Parse

- *Oracle can re-use an existing execution plan already in the library cache*
- *Consists of a lookup in the Shared SQL Area*
- *Preferable over a hard parse*

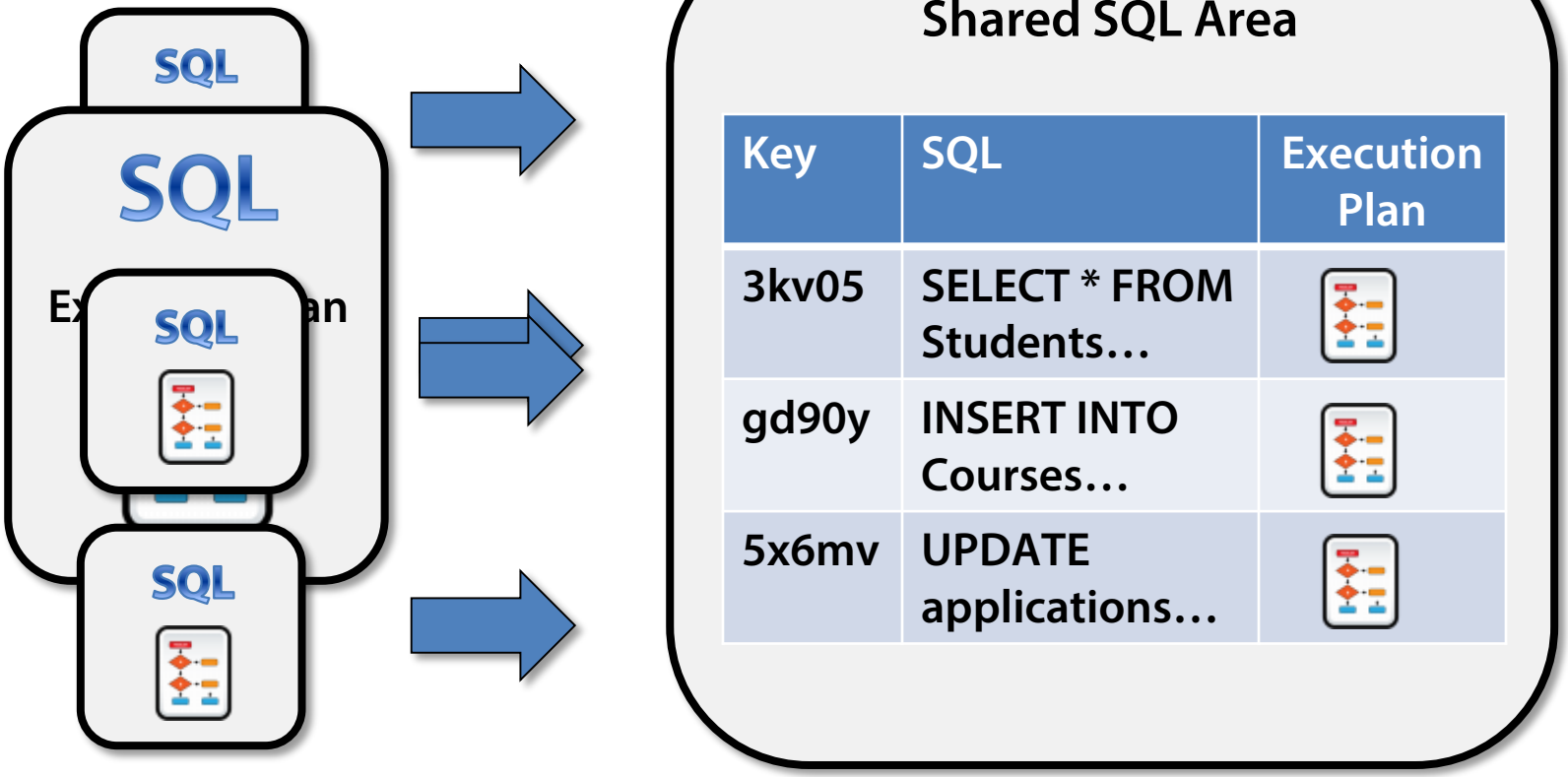
Hard Parse

- *Oracle generates a new execution plan*
- *All statements must be hard parsed on their first execution or after a period of time such that they are no longer in the Shared SQL Area*

Shared SQL Area Wrap Up

- **How Large is the Shared SQL Area**
  - Varies by system
  - Usually sufficient to hold several thousand statements and execution plans
- **The Shared SQL Area is shared throughout an Oracle instance**
  - Will contain SQL from all users and applications
  - One poorly behaved application can cause issues for other applications
- **Regenerating statistics on a database object will cause an execution plan to be flushed from the shared SQL area**
  - This is managed internally by Oracle
- **Contents can be flushed by a DBA**
  - Useful in test sometimes, not appropriate for production

Multiple Processes and the Shared SQL Area



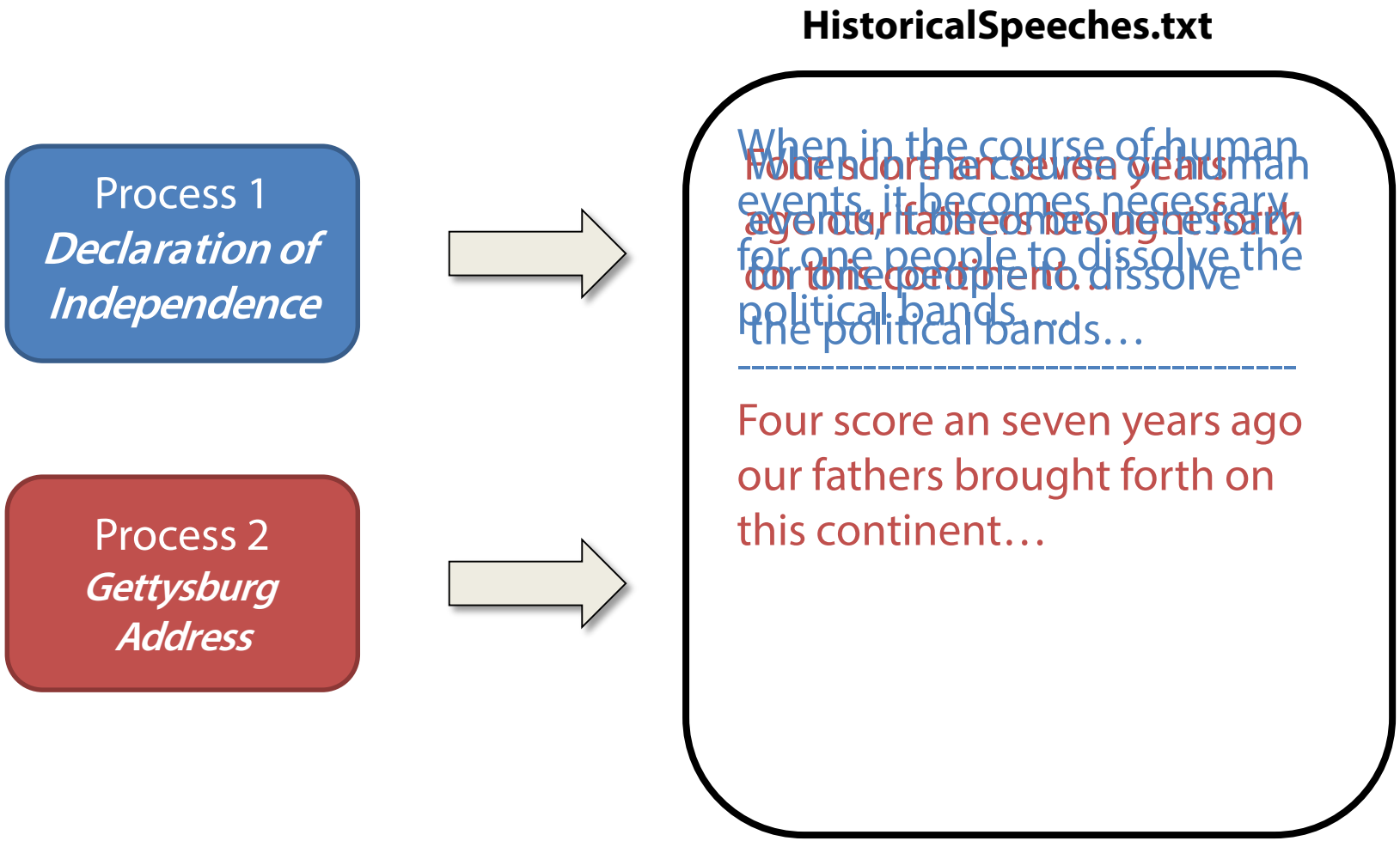
Accessing Shared Resources

- **Concept is the same as multi-threaded programming**
  - Only one thread can modify the object at a time
  - Other threads must wait before entering critical sections of code

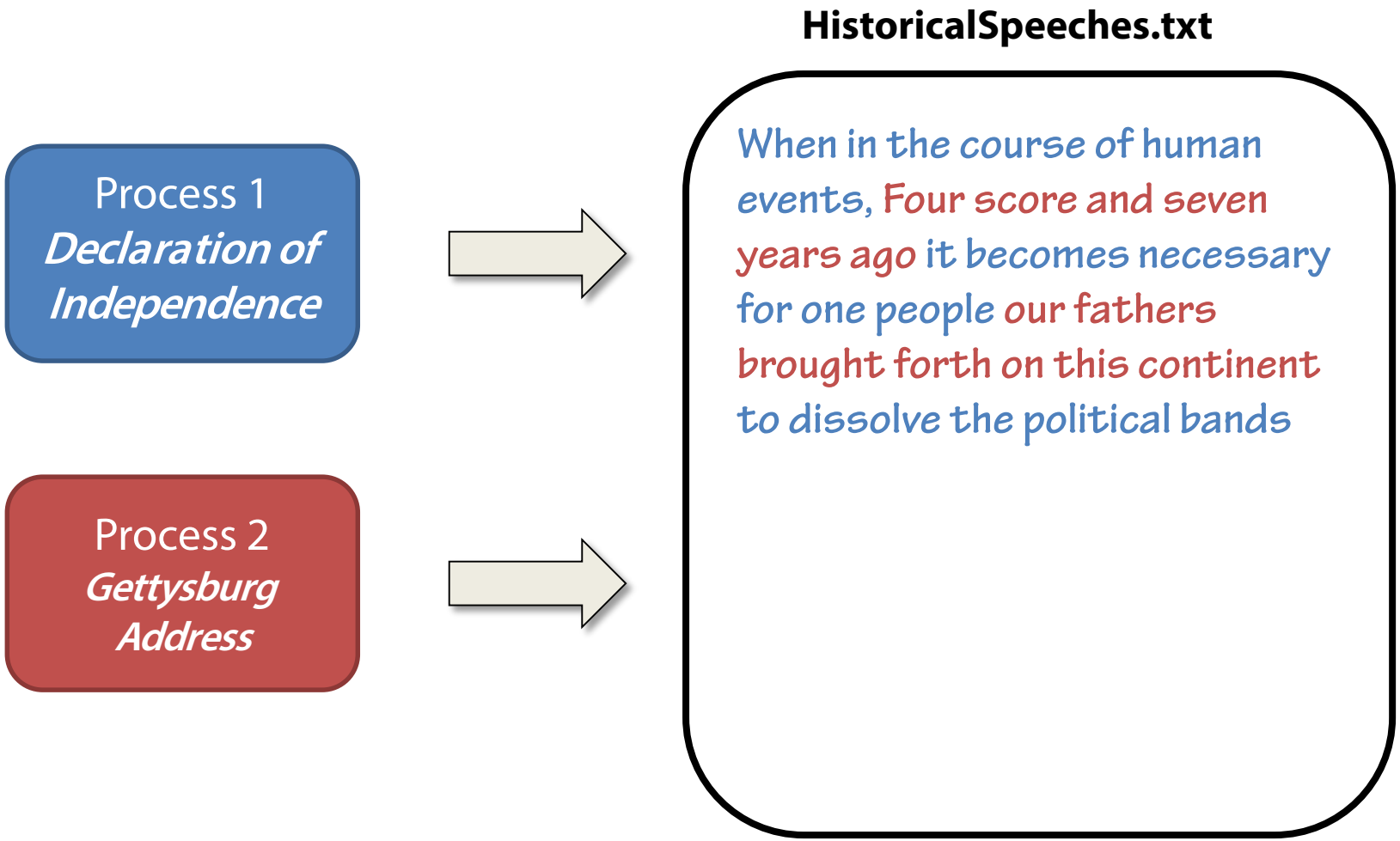
- **Latch – A serialization mechanism in Oracle used to synchronize access to shared resources**

- **Processes that are waiting to acquire the latch to access the shared SQL area are said to be experiencing latch waits**

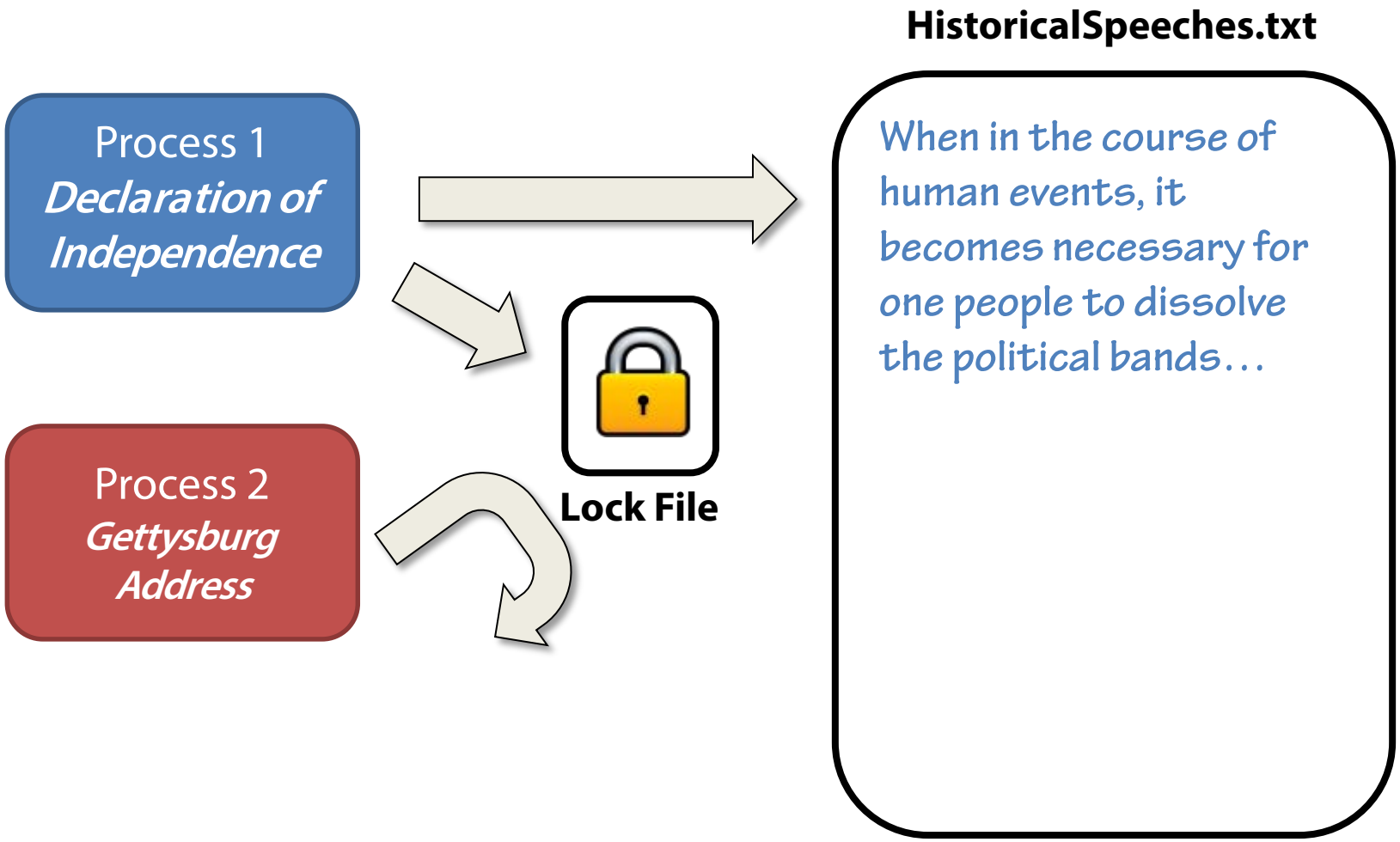
Multiple Processes Writing to a File



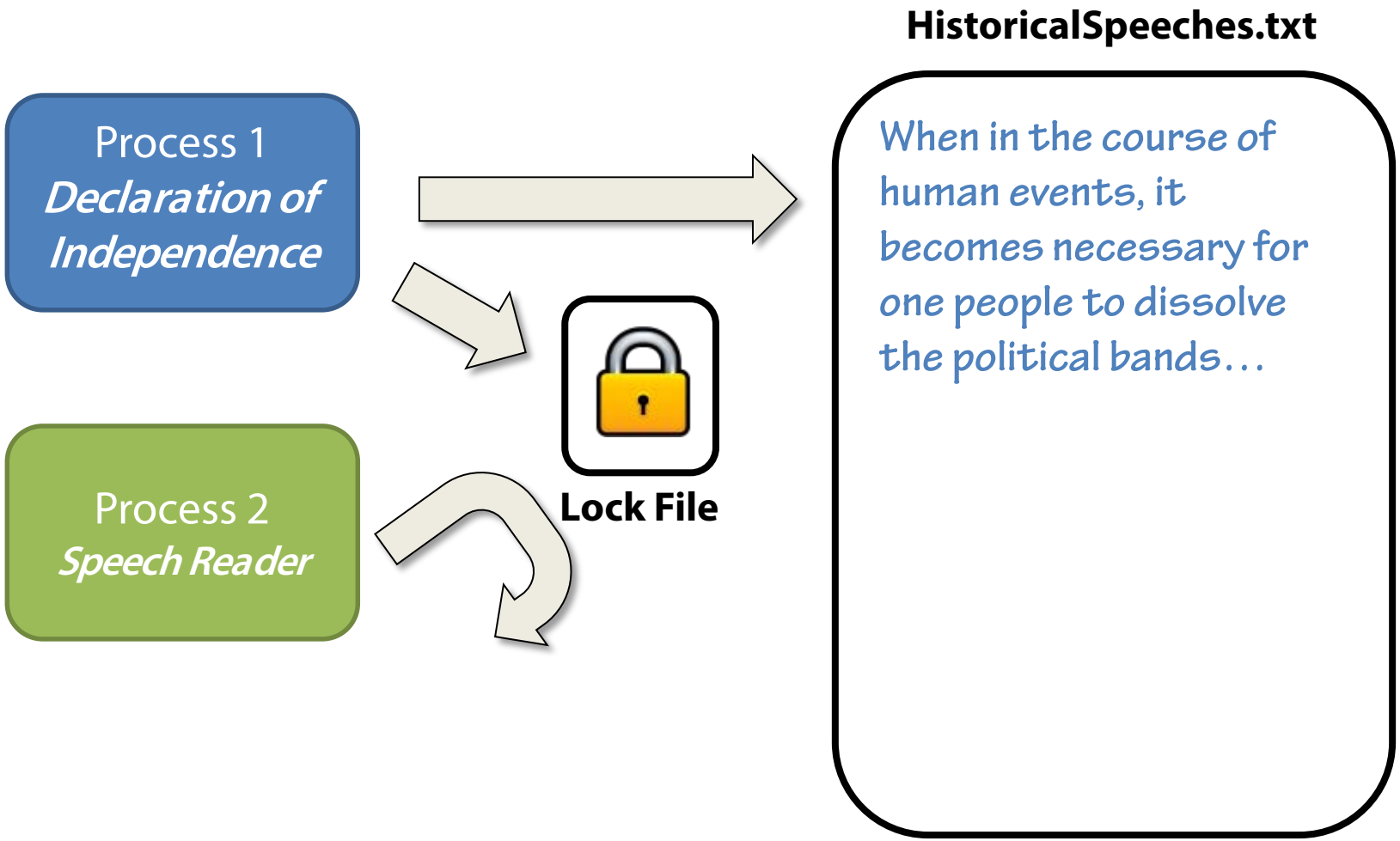
Data Corrupted by Multiple Concurrent Writes



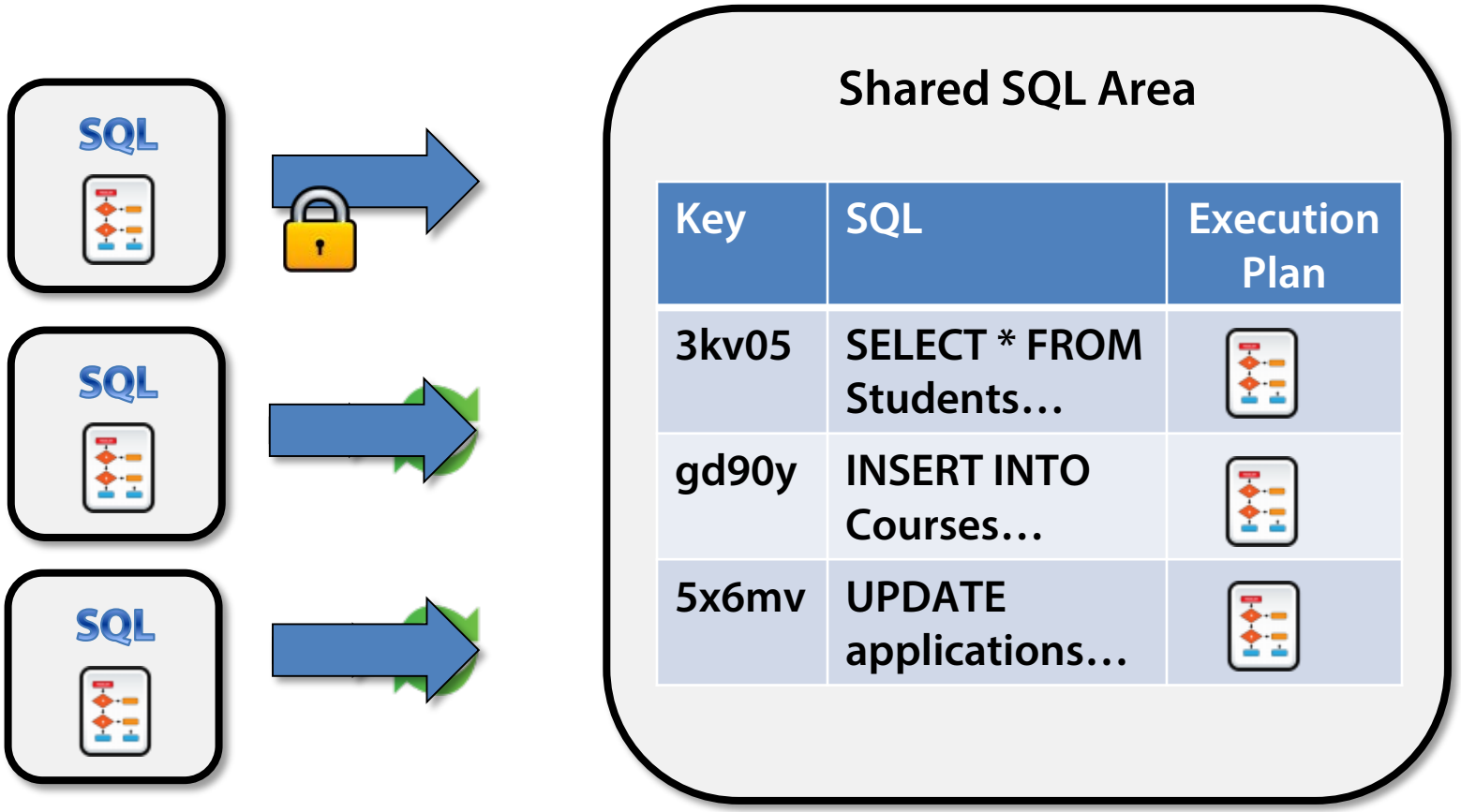
Introducing a Lock File



Reads and Writes Also Must Be Coordinated



Synchronizing Access to the Shared SQL Area



When a Process Cannot Acquire the Latch

Latch Spinning

Process will loop trying over and over to acquire the latch

Sleep

After a few thousand iterations, the process will sleep for a short duration

Latch Spinning

Upon waking, the process will resume looping trying to acquire the latch

Impacts of Shared SQL Area Contention

- **CPU is wasted as processes undergoing latch waits spin trying to acquire the latch**

- **Access to the Shared SQL Area will become a bottleneck**
  - The more statements that are hard parsing, the worse the bottleneck will be
  - Think of a multi-lane highway merging down to a single lane
  - Response times will be slower because each process has to wait for access to the Shared SQL Area

- **A system will only run as fast as its slowest component**
  - System resources may be available, but the bottleneck prevents them from being utilized.

Matching Statements

- **To use a cached execution plan, a SQL statements must exactly match a prior executed statement**

- **Statements have to match character by character**
  - Column order
  - Order of WHERE clause predicates
  - Table aliases
  - Spacing
  - Character case
  - Any specified literal values

- **If String.Equals() would return false, then the statements do not match**

Different Casing, Different Spacing, Different Query

```
SELECT
        department_code, course_number, course_title,
        credits, grade_code, points
    FROM v_student_grades
    WHERE student_id = 12345
        AND term_code = 'FA2013'
```

```
SELECT
        department_code, course_number, course_title,
        credits, grade_code, points
    FROM V_STUDENT_GRADES
    WHERE student_id = 12345
        AND term_code = 'FA2013'
```

Different

Different Column Order – Different Query

```
SELECT
        department_code, course_number, course_title,
        credits, grade_code, points
    FROM v_student_grades
    WHERE student_id = 12345
        AND term_code = 'FA2013'
```

```
SELECT
        course_title, department_code, course_number,
        credits, grade_code, points
    FROM V_STUDENT_GRADES
    WHERE student_id = 12345
        AND term_code = 'FA2013'
```

Different

SQL Statements Using Literal Values

```
SELECT
        department_code, course_number, course_title,
        credits, grade_code, points
    FROM v_student_grades
    WHERE student_id = 12345
        AND term_code = 'FA2013'
```

```
SELECT
        department_code, course_number, course_title,
        credits, grade_code, points
    FROM v_student_grades
    WHERE student_id = 67890
        AND term_code = 'FA2013'
```

Different

SQL Statements Using Literal Values

```
SELECT
        department_code, course_number, course_title,
        credits, grade_code, points
    FROM v_student_grades
    WHERE student_id = 12345
        AND term_code = 'FA2013'
```

```
SELECT
        department_code, course_number, course_title,
        credits, grade_code, points
    FROM v_student_grades
    WHERE student_id = 12345
        AND term_code = 'SP2013'
```

Different

Queries Using Bind Variables

```
SELECT
        department_code, course_number, course_title,
        credits, grade_code, points
    FROM v_student_grades
    WHERE student_id = :student_id
        AND term_code = :term_code
```

- **Bind variables in this statement**
  - :student_id and :term_code are bind variables
  - These serve as placeholders for the actual values, which will be supplied at execution time
  - In Oracle, bind variables always start with a colon (:) character
  - Actual values must be provided when the statement is executed

Using Bind Variables from Applications

- **Most ORM's use bind variables implicitly**
  - Great for new development, but most of us still need to support existing applications
  - It is not always possible to replace an existing data access layer with a new data access layer based on an ORM

- **Any application that does not use bind variables will impact every other application that uses the same Oracle database instance**
  - Poorly written applications consume Oracle resources that are then unavailable for well written applications

Results – Literal Values vs Bind Variables

Metric	Literal Values	Bind Variables
Elapsed Time	14.132 seconds	5.933 seconds
Database CPU	37.02 seconds	9.38 seconds
Parse Count	4068	5
Parse Time Elapsed	29.12 seconds	0.01 seconds
Parse Time CPU	26.95 seconds	0.04 seconds
Shared Pool Latch Gets	349129	2755
Shared Pool Latch Misses	14764	0
Shared Pool Latch Sleeps	78	0
Shared Pool Latch Spin Gets	14709	0

Takeaways From This Demonstration

- **Contention can occur even if only a few processes are hard parsing**
  - What would the impact be if this were 50 processes? 100? 500?
- **Applications that don't use bind variables consume an oversized amount of resources**
  - Amount of CPU used and contention caused by one application will impact all applications
- **Failure to use bind variables limits system throughput**
  - Literal values ~ 280 queries per second
  - Bind Variables ~ 667 queries per second

Statement Level Performance Tuning

David Berry
http://buildingbettersoftware.blogspot.com/

Tuning SQL Statements for Performance

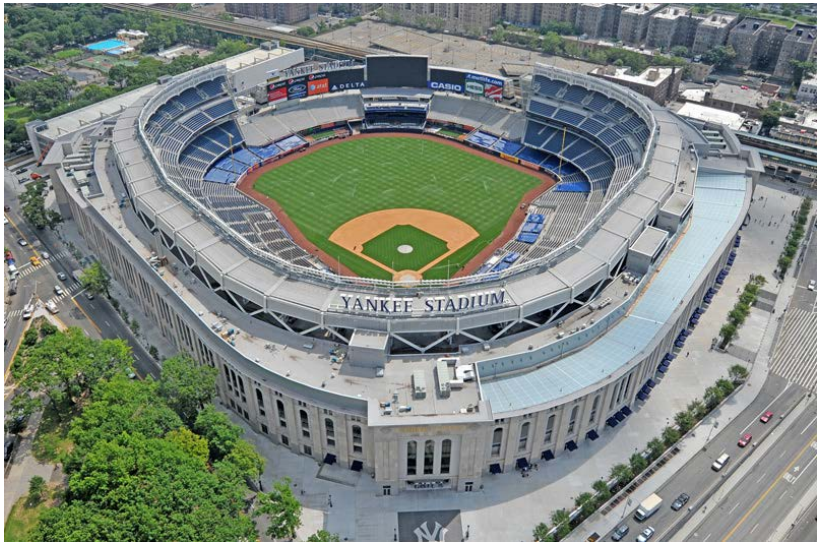
SQL Statement is running slow

Where do I start?

Look at the execution plan

Driving Directions

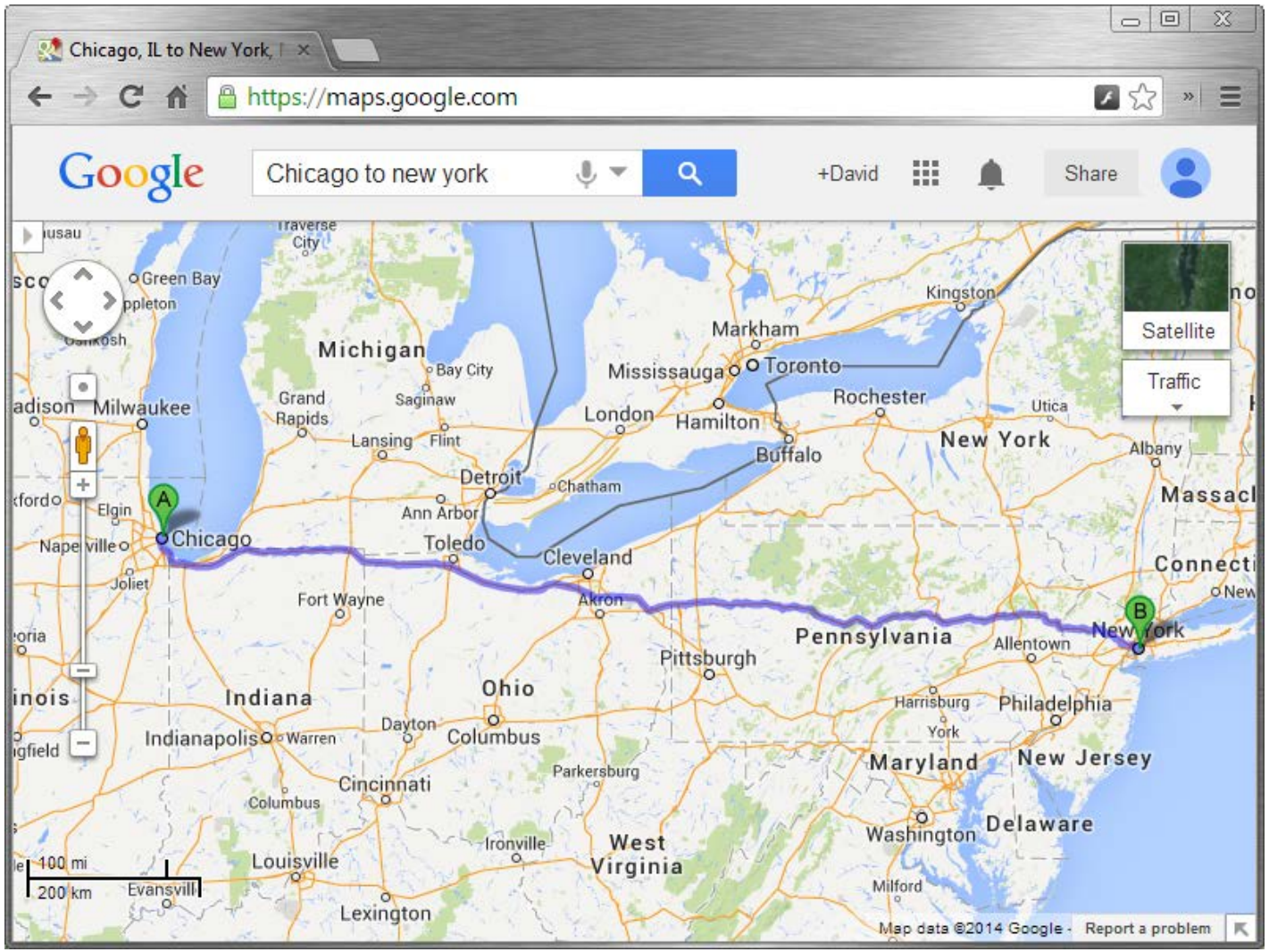
Chicago



New York

Photos:
http://en.wikipedia.org/wiki/File:Wrigley_field_720.jpg and

How Do I Go from Chicago to New York?



Similarities Abound

Oracle	Google Maps
This SQL statement needs to return this data	I want to drive from Chicago to New York
Evaluates table statistics	Evaluates street and traffic data
An execution plan of how to fulfill the SQL statement in the most efficient way	A route for how to drive from Chicago to New York in the shortest period of time

Execution Plan Defined

Execution Plan	The individual steps Oracle will execute for a statement
Each Step	A basic operation Oracle executes
Estimated Cost	The relative cost of this operation

Autotrace Capability

- **Available in SQL Developer and SQL Plus**
- **Will execute the SQL statement**
- **Provides details like:**
  - Logical and physical IO's
  - Number of sorts
  - Network packets

Logical IO vs Physical IO

Data Block in Buffer Cache

Buffer Cache

Data Block Not in Buffer Cache

Buffer Cache

Oracle Data Files

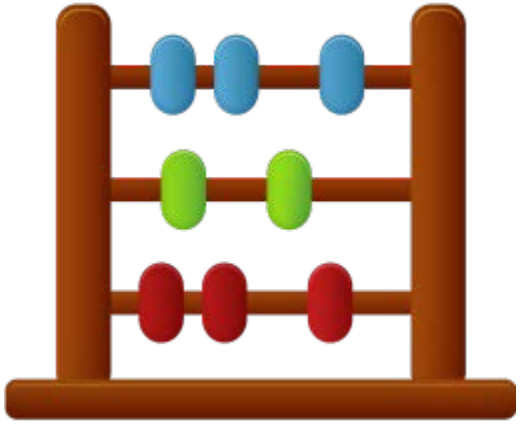
Logical IO Tuning Goals

Fewer logical IO operations means less data Oracle has to process

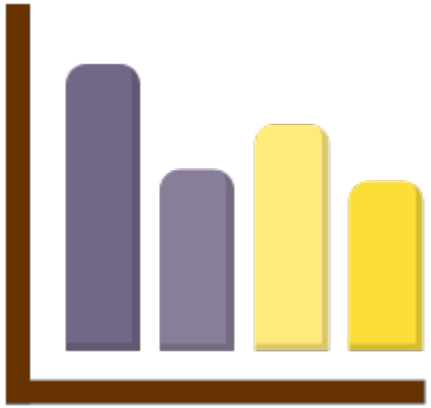
Reducing logical IO will in turn reduce physical IO

Data Size and Distribution

- **What operation is used depends on statistics for each object**

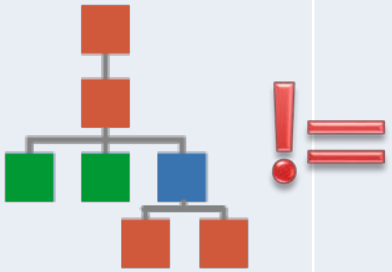
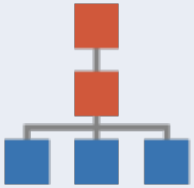


Data Size



Data Distribution

Production vs Test Environments

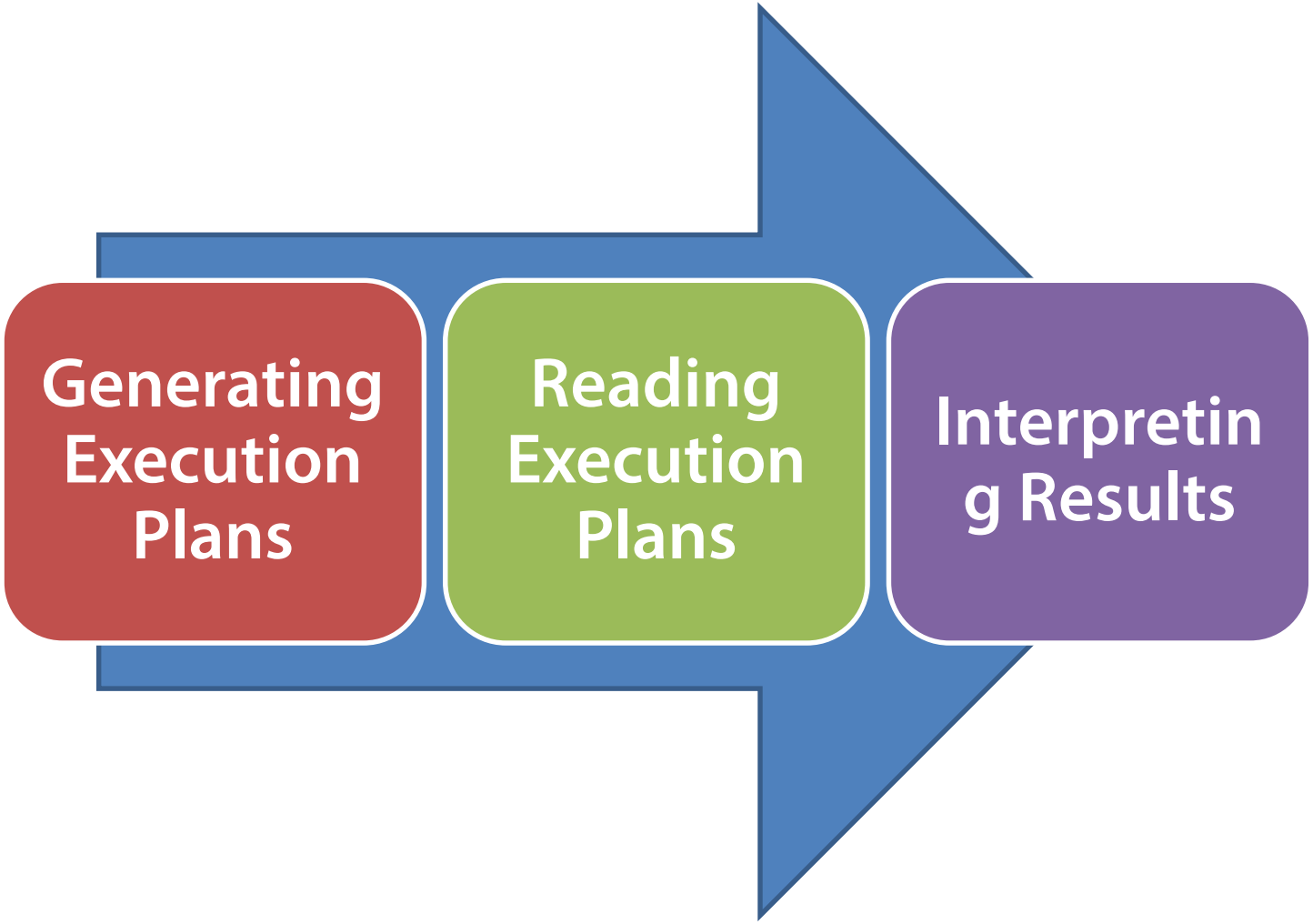
	Production	Test
Database Size		
Execution Plan		

Impacts of Database Size

Database size impacts what execution plan Oracle generates

Build a test database that accurately represents production data

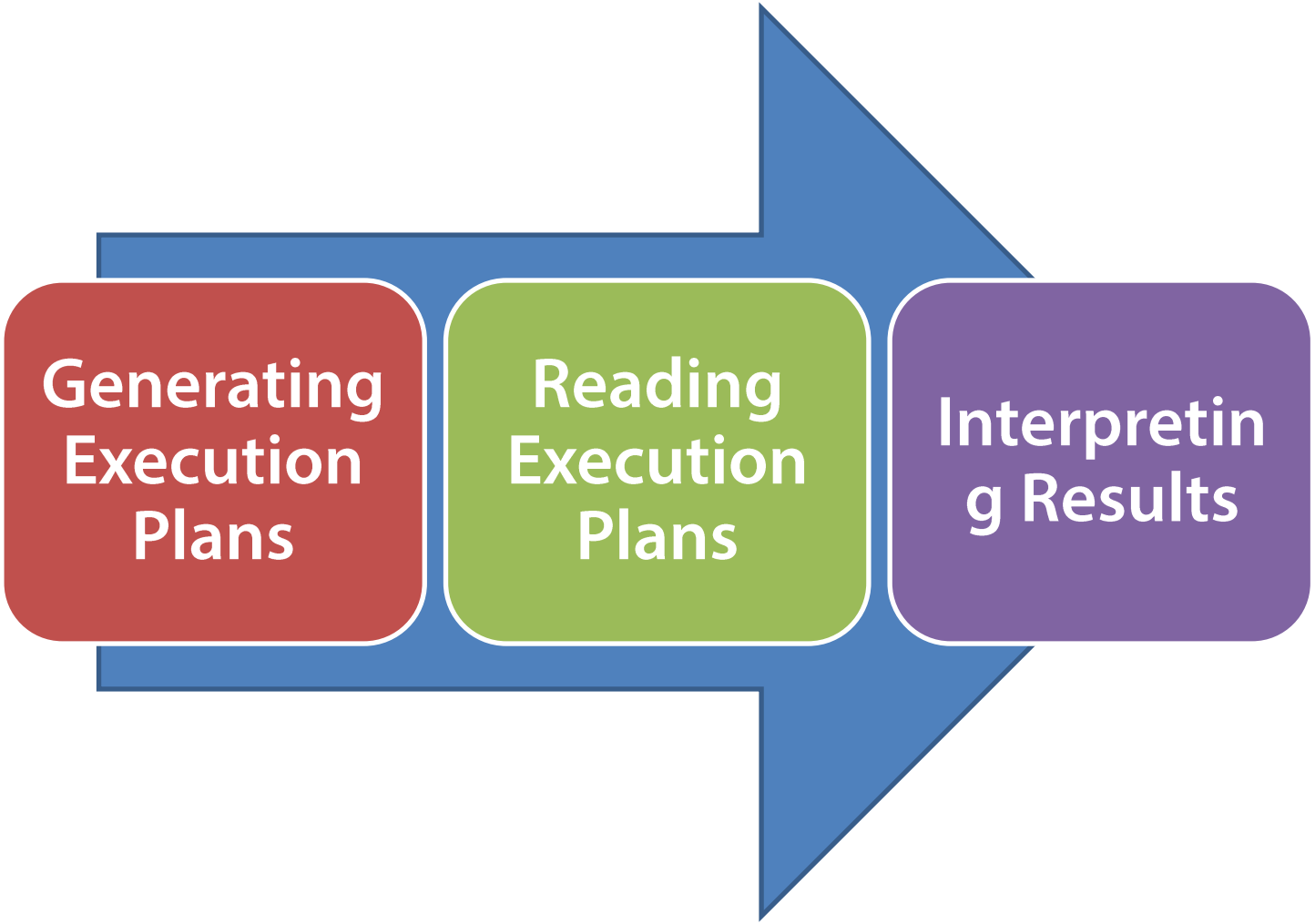
Execution Plan Summary



Execution Plans In Depth

David Berry
http://buildingbettersoftware.blogspot.com/

Execution Plan Summary



Execution Plan Operations

Table Operations

- Table Scan Full
- Table Access by Index Row ID

Index Operations

- Index Range Scan
- Index Unique Scan
- Index Fast Full Scan
- Index Full Scan

Join Operations

- Nested Loops Join
- Merge Join
- Hash Join

Table Access Operations

- One of the primary drivers of performance
- Try to retrieve and process the minimum amount of data needed

Table Scan Full

- Oracle will read and process all blocks for the table
- Generally the most expensive operation
- Equivalent to doing a sequential scan through an array

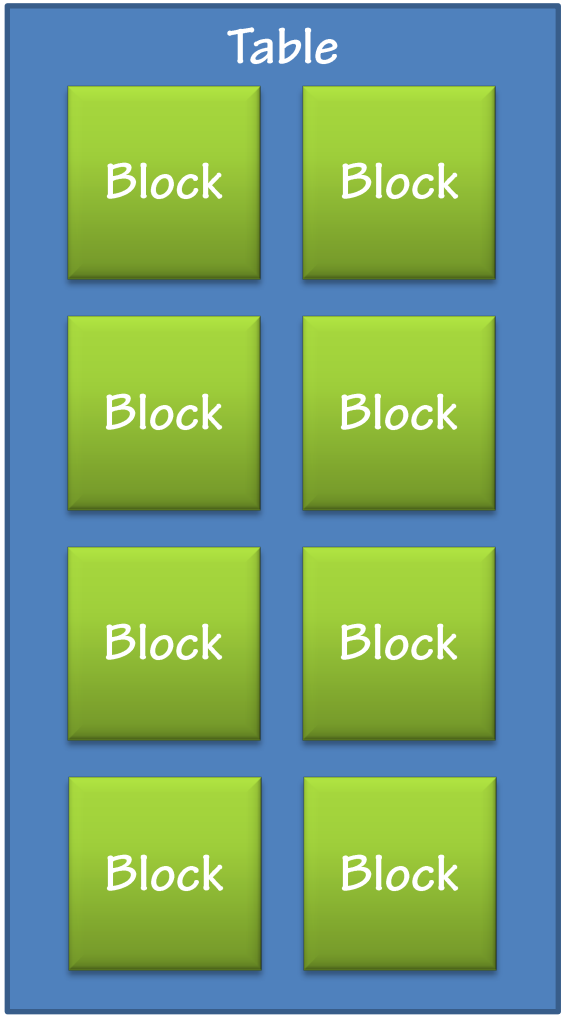
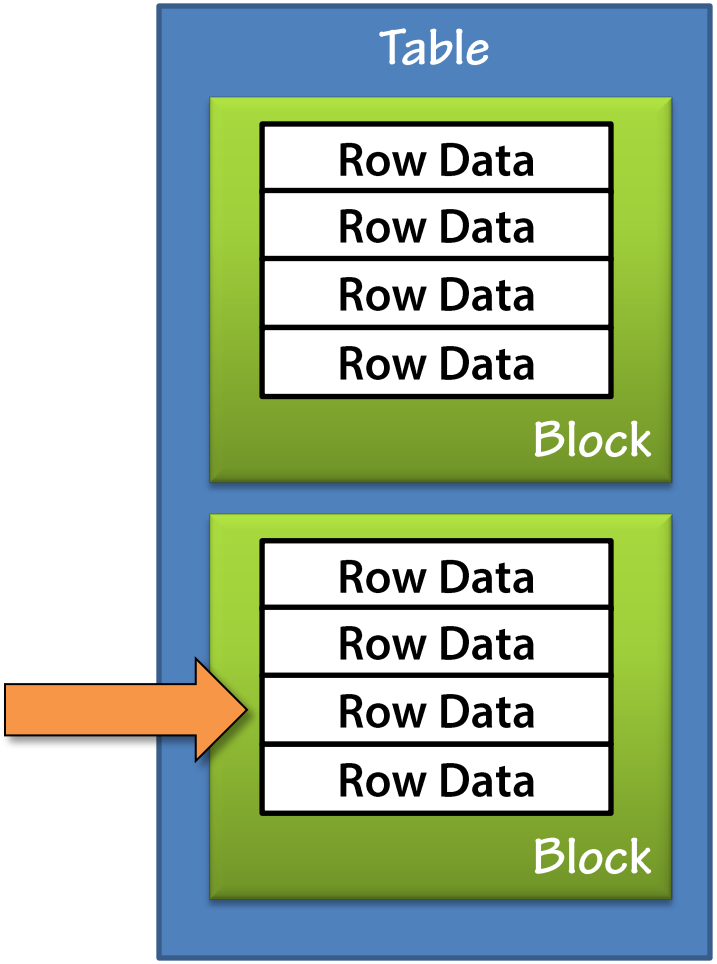


Table Scan Full – Small Tables

- Full table scans normal for smaller tables
- Often more efficient than using an index
- No need to investigate – Oracle is functioning correctly

Table Access by Index Row Id

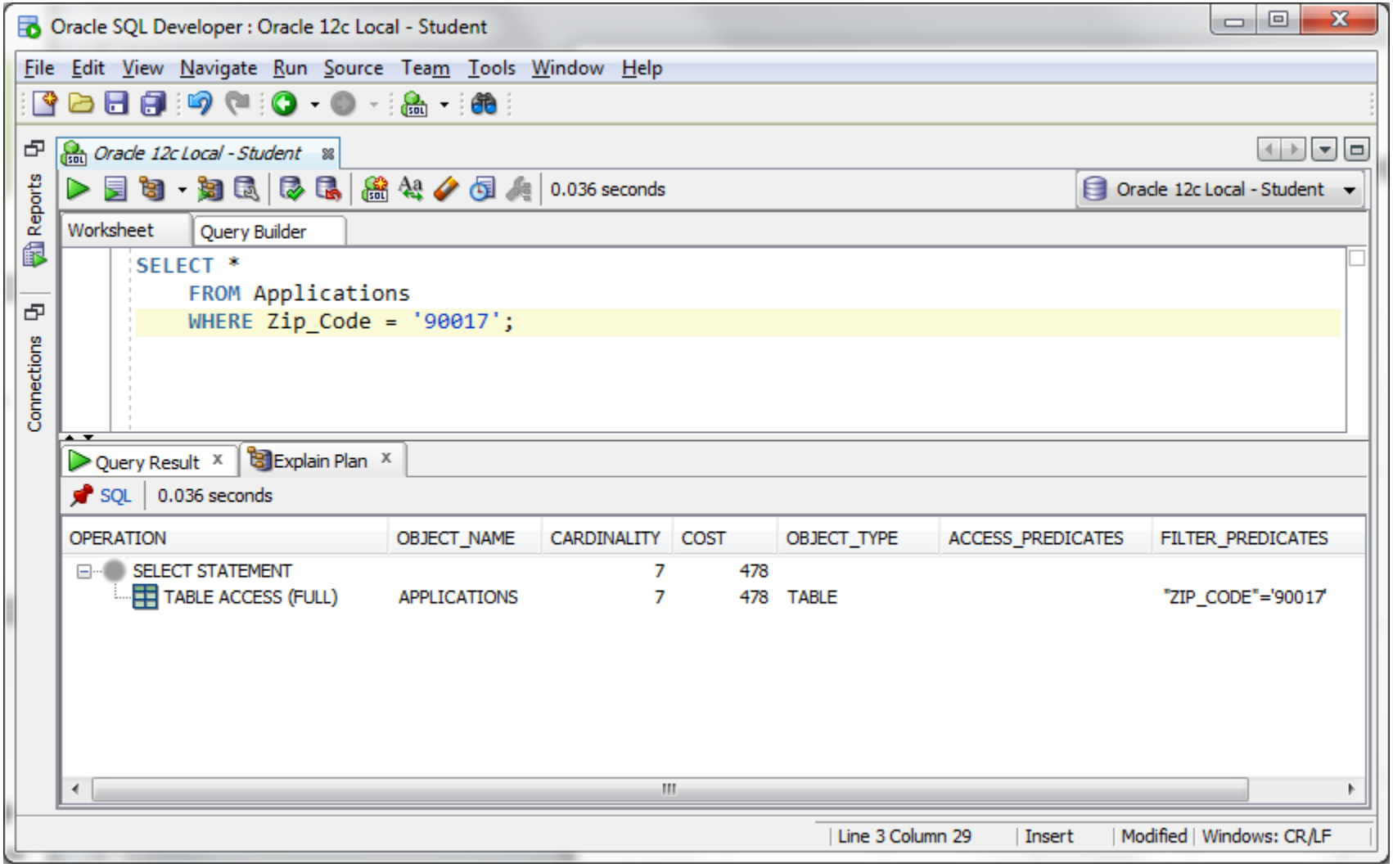
- Used when Oracle knows the ROWID of the row it is looking for
- Oracle directly accesses the block and then the row containing the needed data



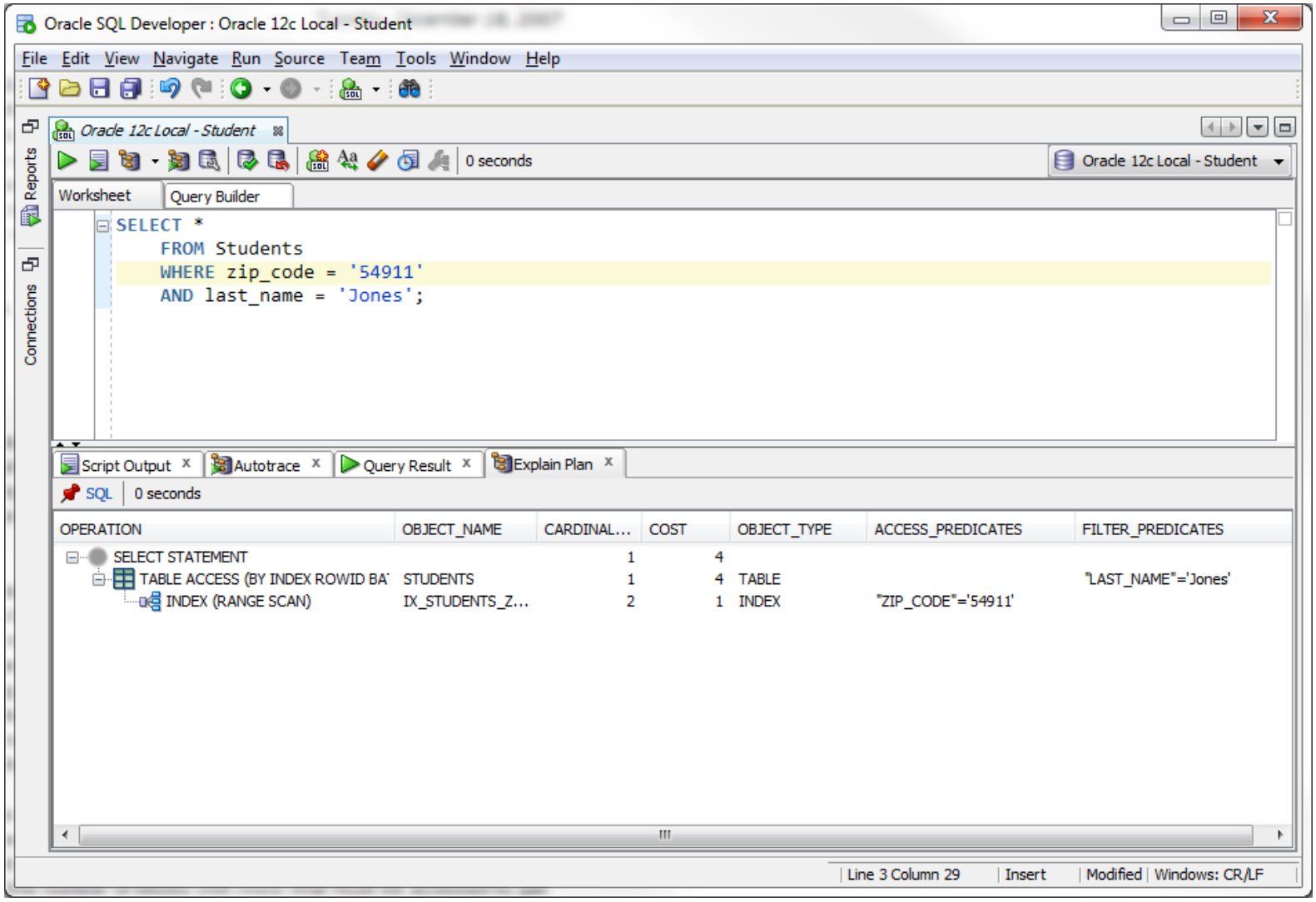
Filter Predicates

- Filter predicates are applied *after* the data is read
- Filter predicates on full table scans may indicate the need for an index

Filter Predicate

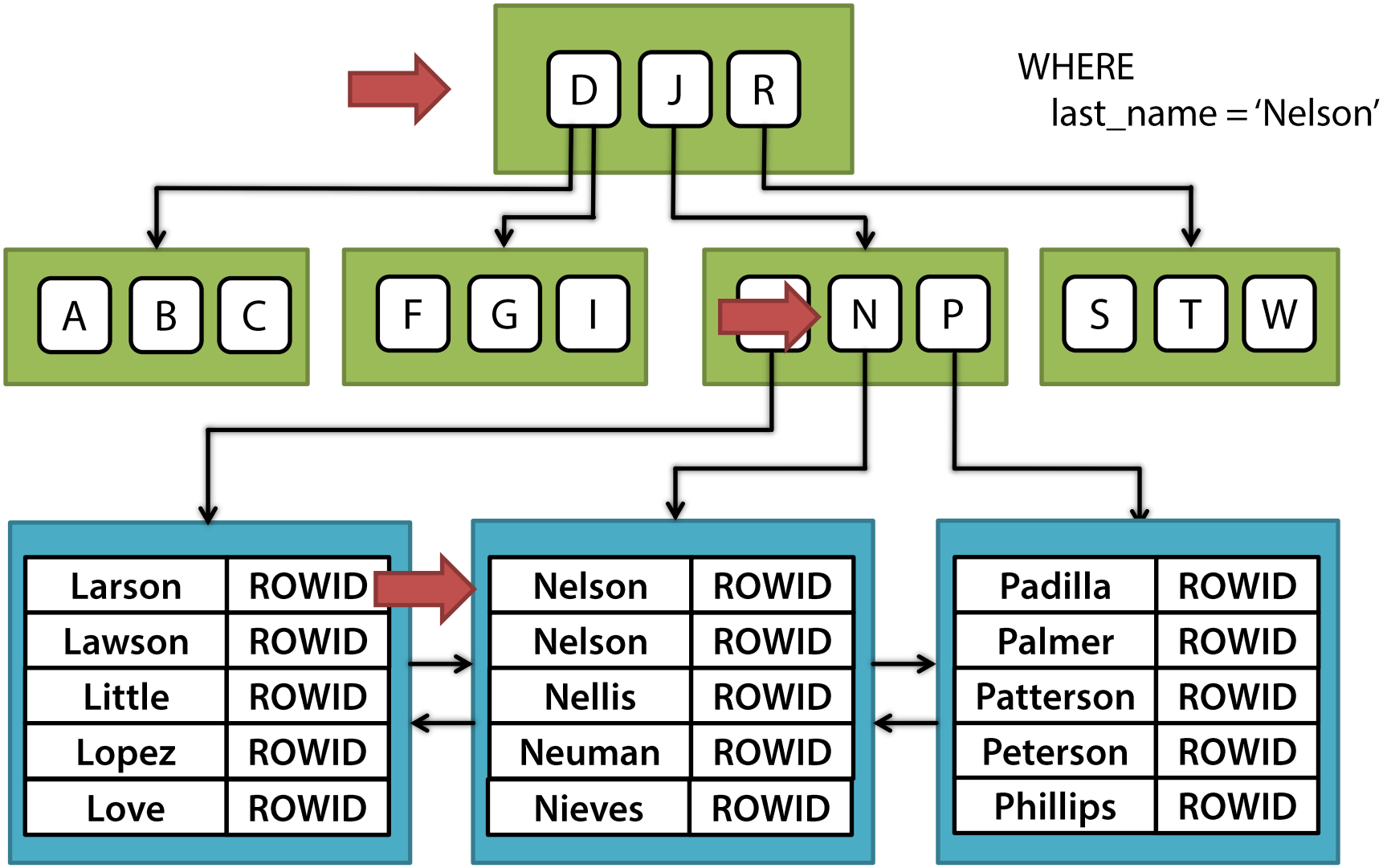


Filter Predicate – Table Access by Row Id

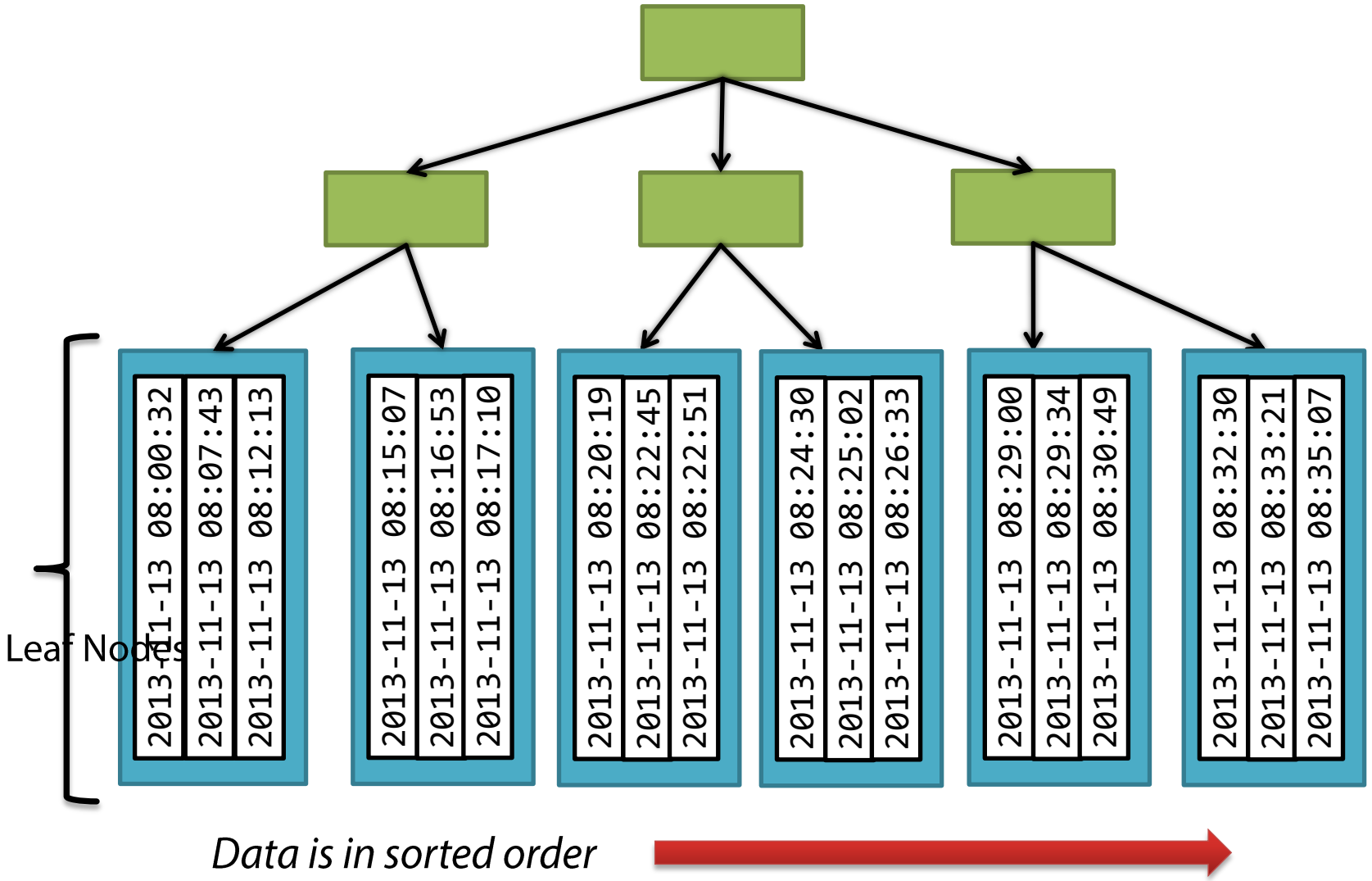


Index Lookup Operations

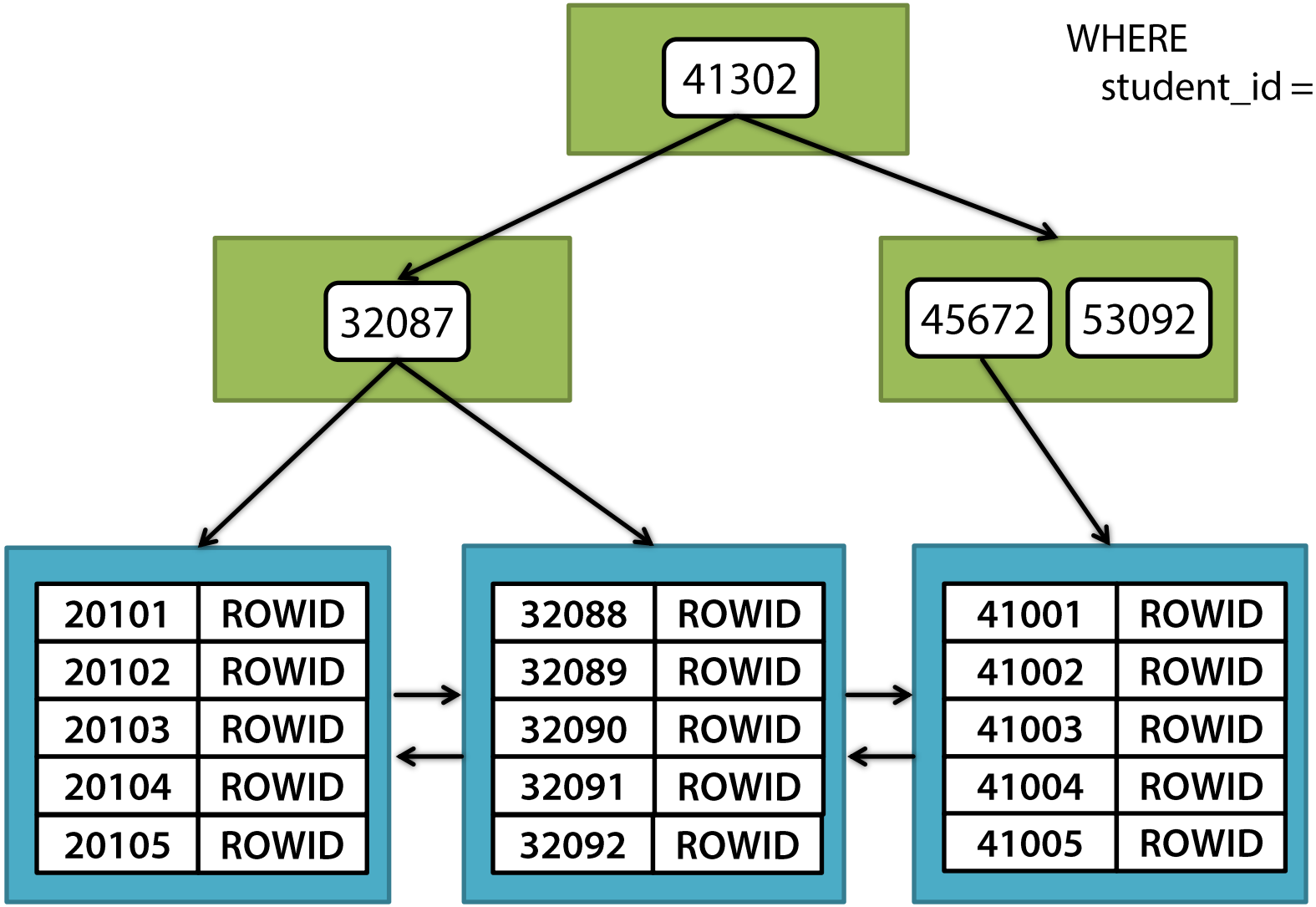
Index Range Scan



Range Scan Through Leaf Nodes

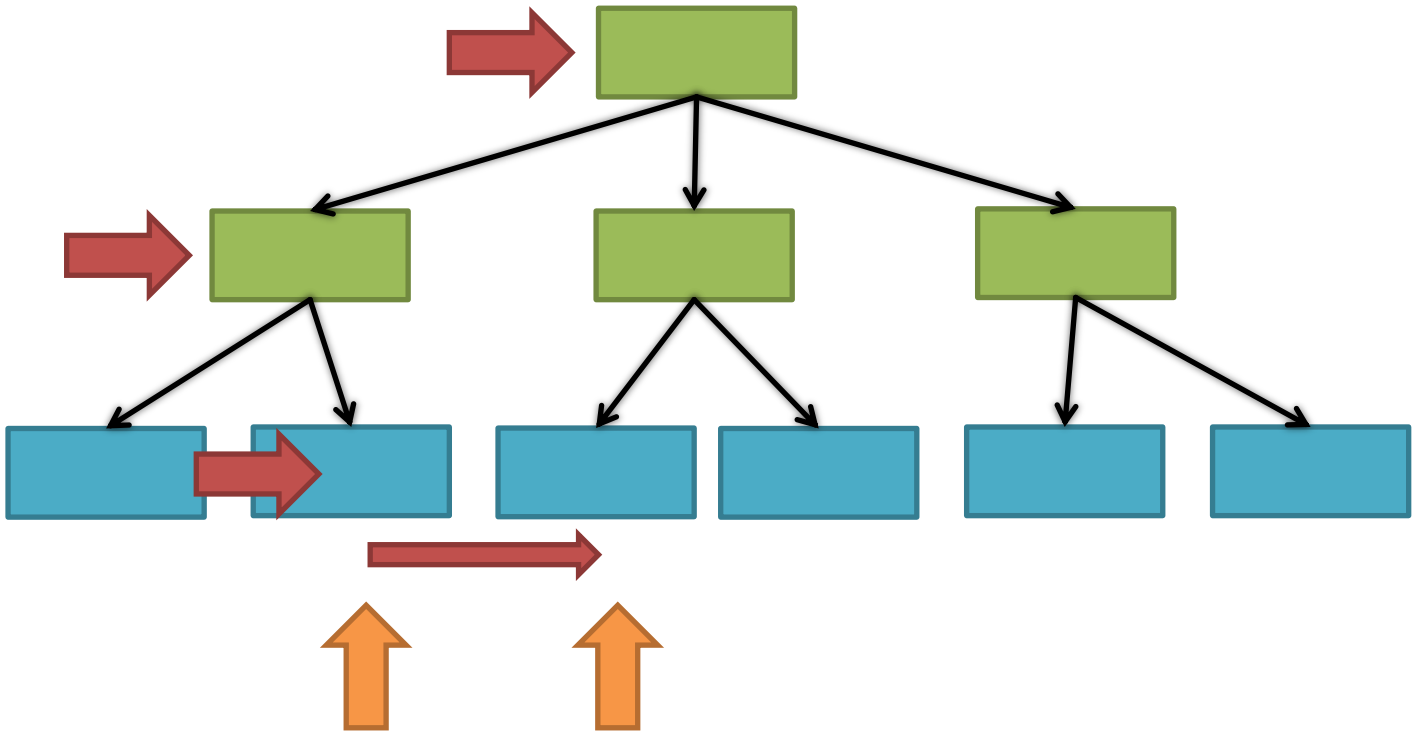


Index Unique Scan



WHERE
student_id = '32091'

Index Access Predicates



Defines start and stop point of index range scan operation

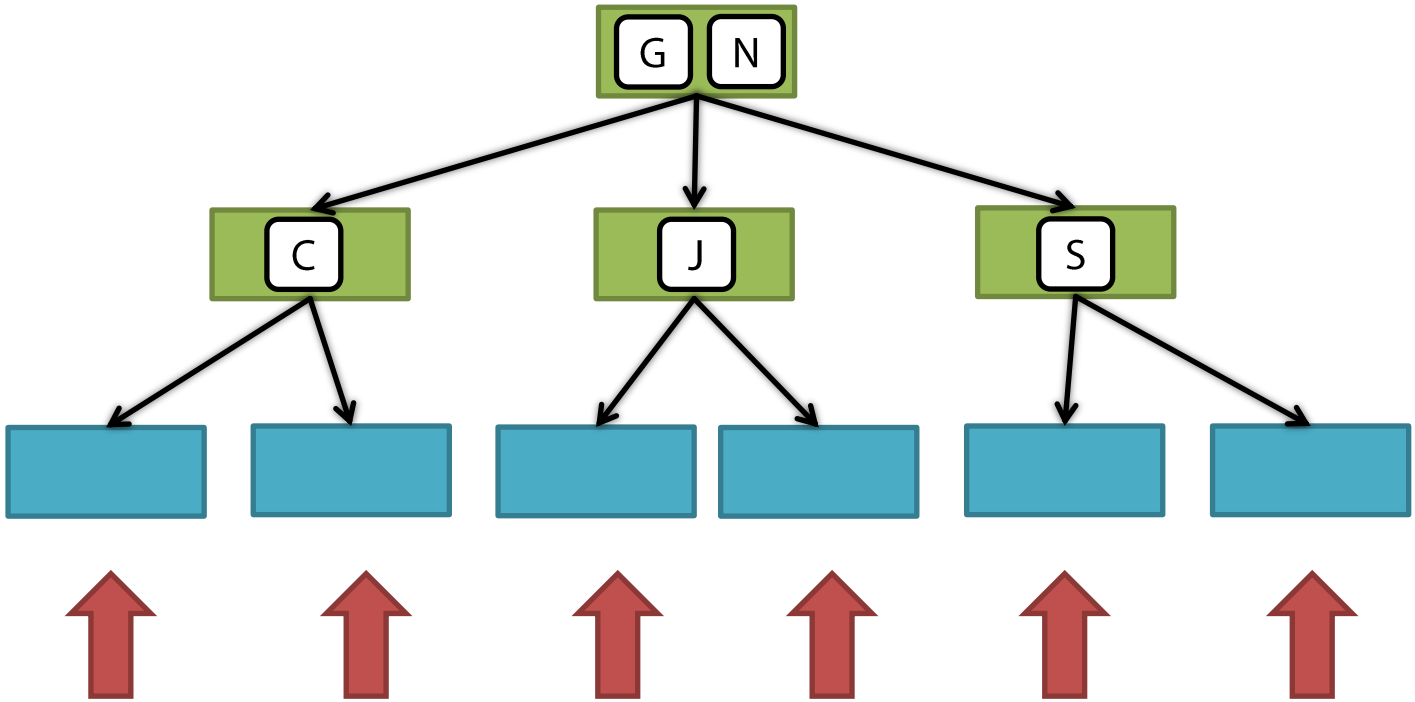
Index Access and Filter Predicates

- **Access Predicate**
  - Applied while reading the index
  - Limits amount of data read

- **Filter Predicate**
  - Applied after index blocks have been read
  - Do not reduce the amount of data read

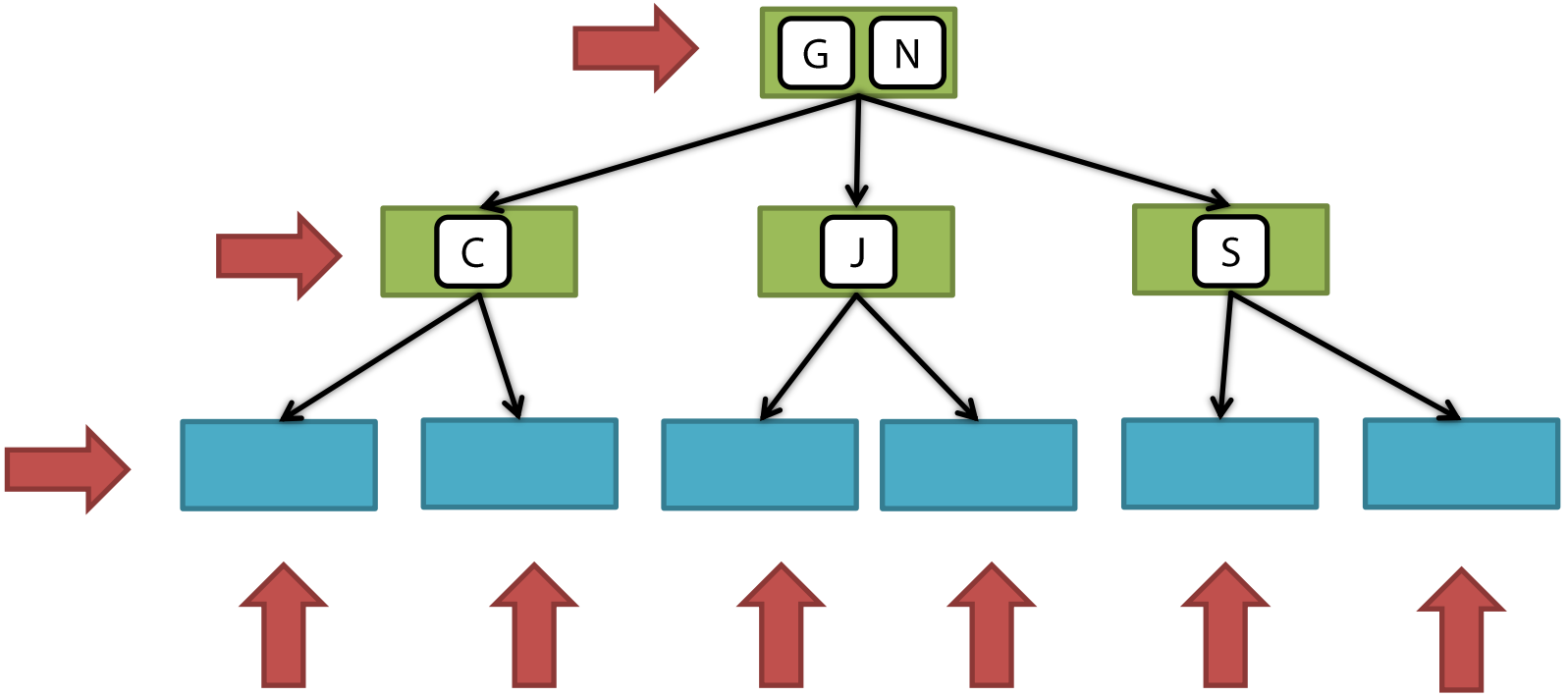
Index Full Scan Operations

Index Fast Full Scan



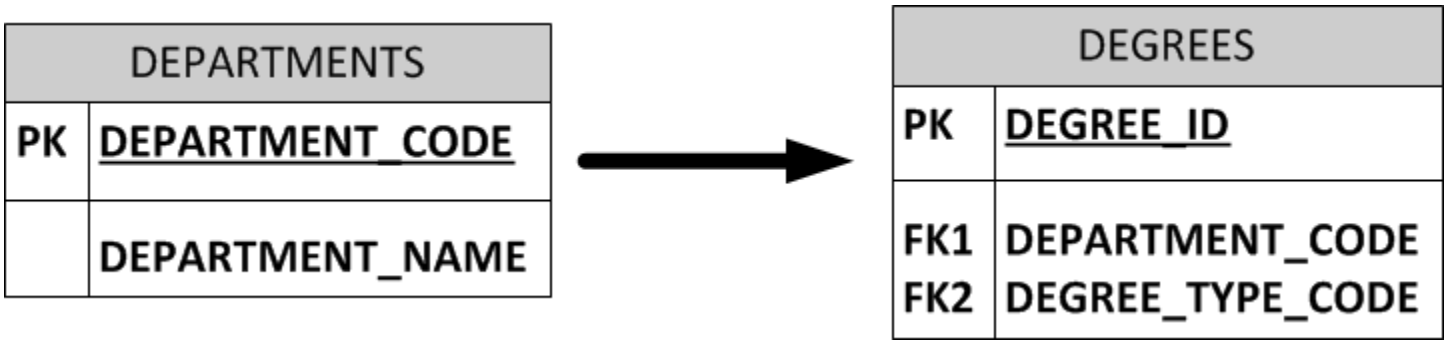
- Entire index is read as fast as possible
- Multi-block reads are used

Index Full Scan



- Index data is read in order
- Single block reads are used

Nested Loops Join



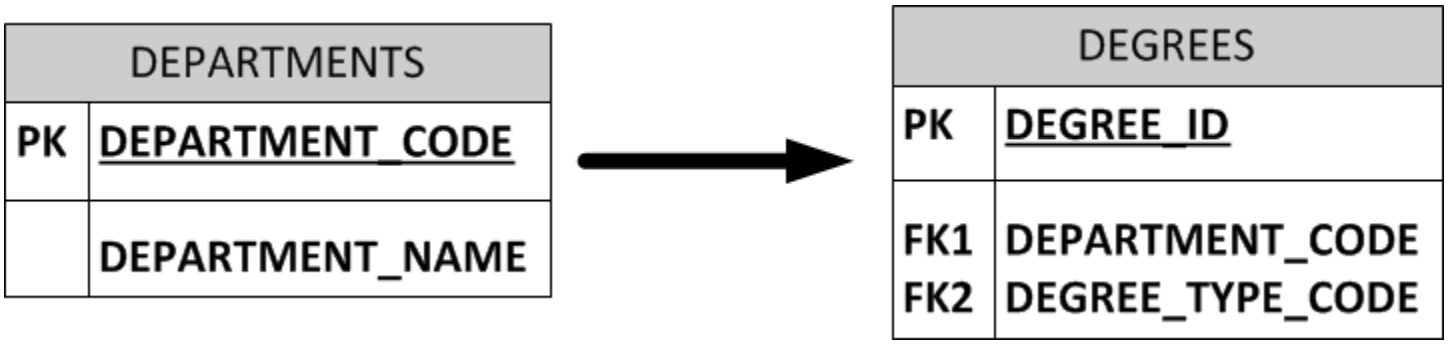
Driver Source
(External Source)

Inner Source

Nested Loops Join - Pseudocode

```
//RowSource externalRowSource;
//RowSource internalRowSource;

while (externalRowSource.Next() )
{
    Object externalItem = externalRowSource.CurrentItem();
    internalRowSource.ResetToBeginning();

    while (internalRowSource.Next() )
    {
        Object internalItem = internalRowSource.CurrentItem();
        if ( JoinCondition.Matches(internalItem, externalItem) )
        {
            // Add to output row source
        }
    }
}
```

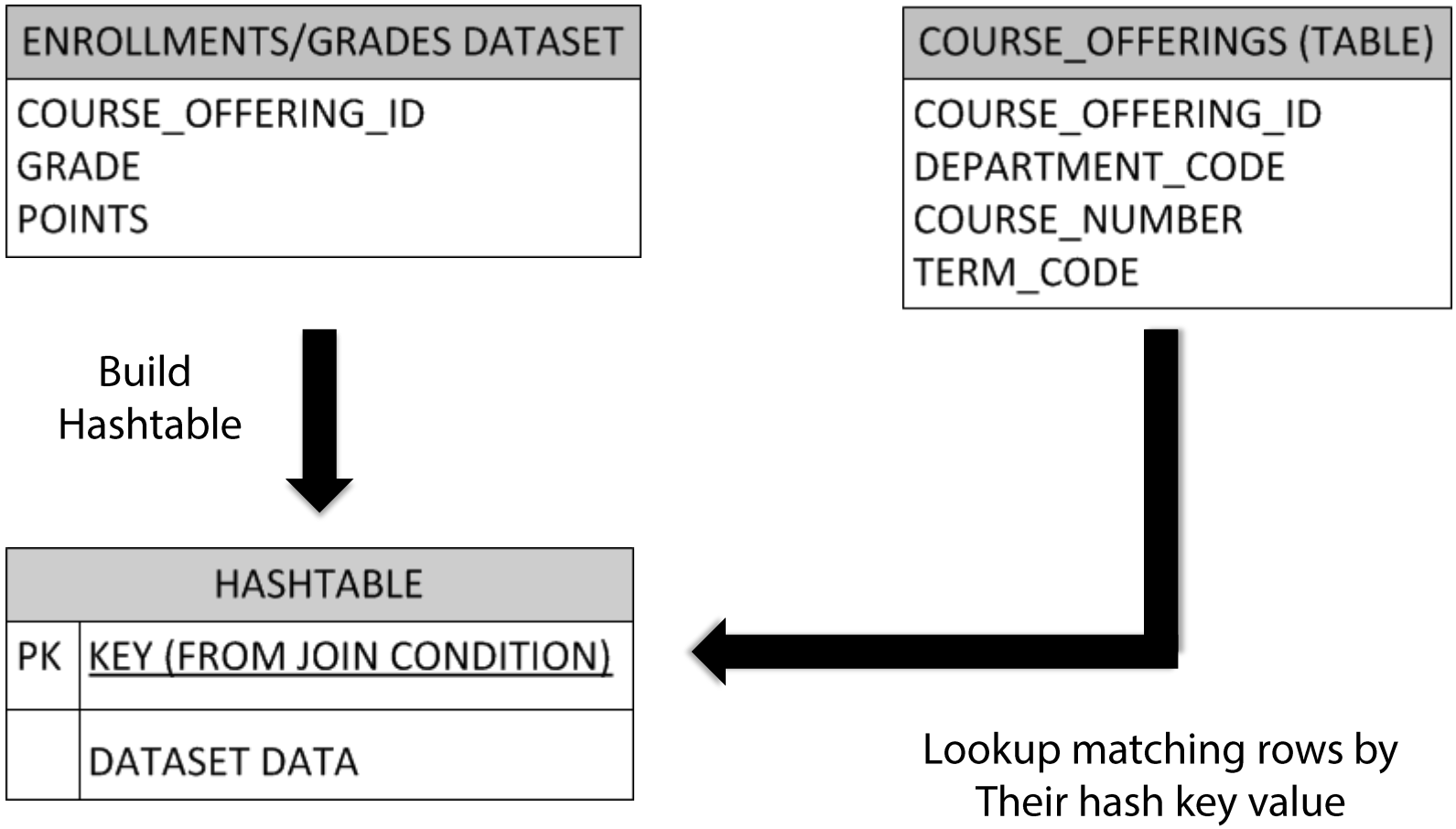
Nested Loops Join – Performance Considerations



Driver Source
(External Source)

Inner Source

Hash Join


ENROLLMENTS/GRADES DATASET
COURSE_OFFERING_ID
GRADE
POINTS
COURSE_OFFERINGS (TABLE)
COURSE_OFFERING_ID
DEPARTMENT_CODE
COURSE_NUMBER
TERM_CODE
Build
Hashtable
HASHTABLE
PK
KEY (FROM JOIN CONDITION)
DATASET DATA
Lookup matching rows by
Their hash key value

Hash Join vs Nested Loops Join

Nested Loops Join

- Up to 100 million comparisons

Hash Join

- Cost to build hashtable
- Cost to iterate through second row source

Merge Join

COURSE_NBR	COURSE_TITLE	DEPT_CODE
101	Physics I	PH
101	Chemistry I	CH
101	College Writing	LA
102	Physics II	PH
102	Chemistry II	CH
101	Intro Programming	CS

DEPT_CODE	DEPT_NAME
PH	Physics
CH	Chemistry
LA	Liberal Arts
CS	Computer Science

COURSE_NBR	COURSE_TITLE	DEPT_CODE
101	Chemistry I	CH
102	Chemistry II	CH
101	Intro Programming	CS
101	Physics I	PH
102	Physics II	PH
101	College Writing	LA

DEPT_CODE	DEPT_NAME
CH	Chemistry
CS	Computer Science
PH	Physics
LA	Liberal Arts

COURSE_NBR	COURSE_TITLE	DEPT_CODE	DEPT_NAME
101	Chemistry I	CH	Chemistry
102	Chemistry II	CH	Chemistry
101	Intro Programming	CS	Computer Science
101	Physics I	PH	Physics
102	Physics II	PH	Physics
101	College Writing	LA	Liberal Arts

Join Operations – Performance Questions

- Which table does Oracle use to drive the SQL statement
- Are the joined columns indexed in both tables
- Do I need to perform this join?

Statement Performance Tuning Summary

Application Context

- **Interactive Applications**
  - Many users
  - High volume, short duration queries

- **Batch Applications**
  - Few, long running queries
  - Must finish in specified window

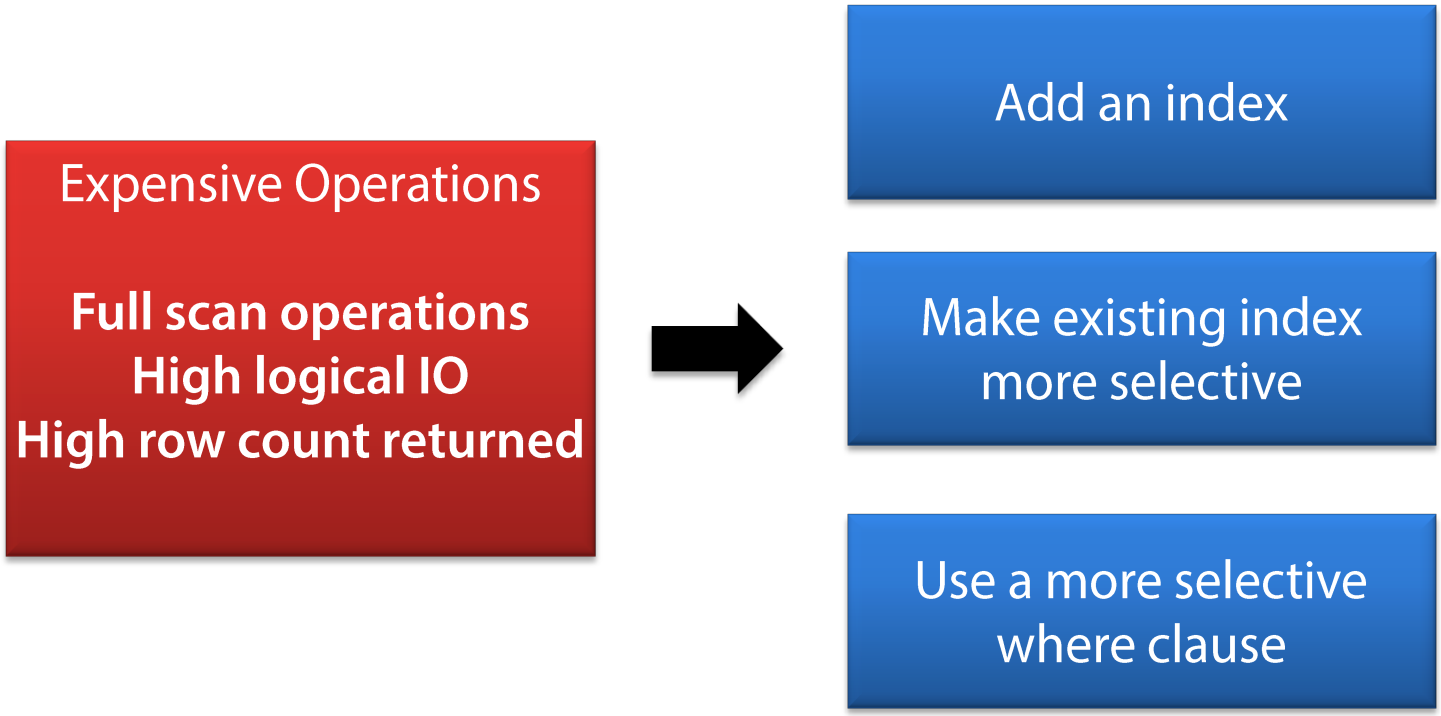
- **Business Intelligence Applications**
  - Fixed and ad-hoc queries
  - Queries process large volumes of data

Identifying Key Tables for Performance

- Identify the key tables in the application
  - Role the table plays
  - Relations to other key tables

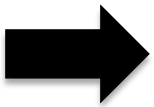
Statement Tuning Methodology

Red Flags in Execution Plans



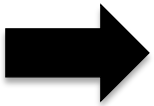
More Selective Where Clause

```
SELECT * FROM Customers
    WHERE Last_Name = 'Johnson'
```



~ 8100 results for
1 million row table

```
SELECT * FROM Customers
    WHERE Last_Name = 'Johnson'
        AND state_code = 'CO'
        AND First_Name LIKE 'G%'
```



25-50? results

Think About Your SQL Statement

- What is the purpose of my SQL statement?
- Does the user really need to see all this data?
- Single responsibility principle applies here too

Final Thoughts

- **Poor performing SQL statements**
  - Impact application usability
  - Consume excess resources in Oracle
  - May impact other SQL statements

- **Well tuned SQL statements**
  - Provide a responsive application
  - Allow greater application scalability

Indexing Essentials

David Berry
http://buildingbettersoftware.blogspot.com/

How Important Are Indexes

- Most important factor influencing database performance
- Poor indexing strategy equates to poor performance

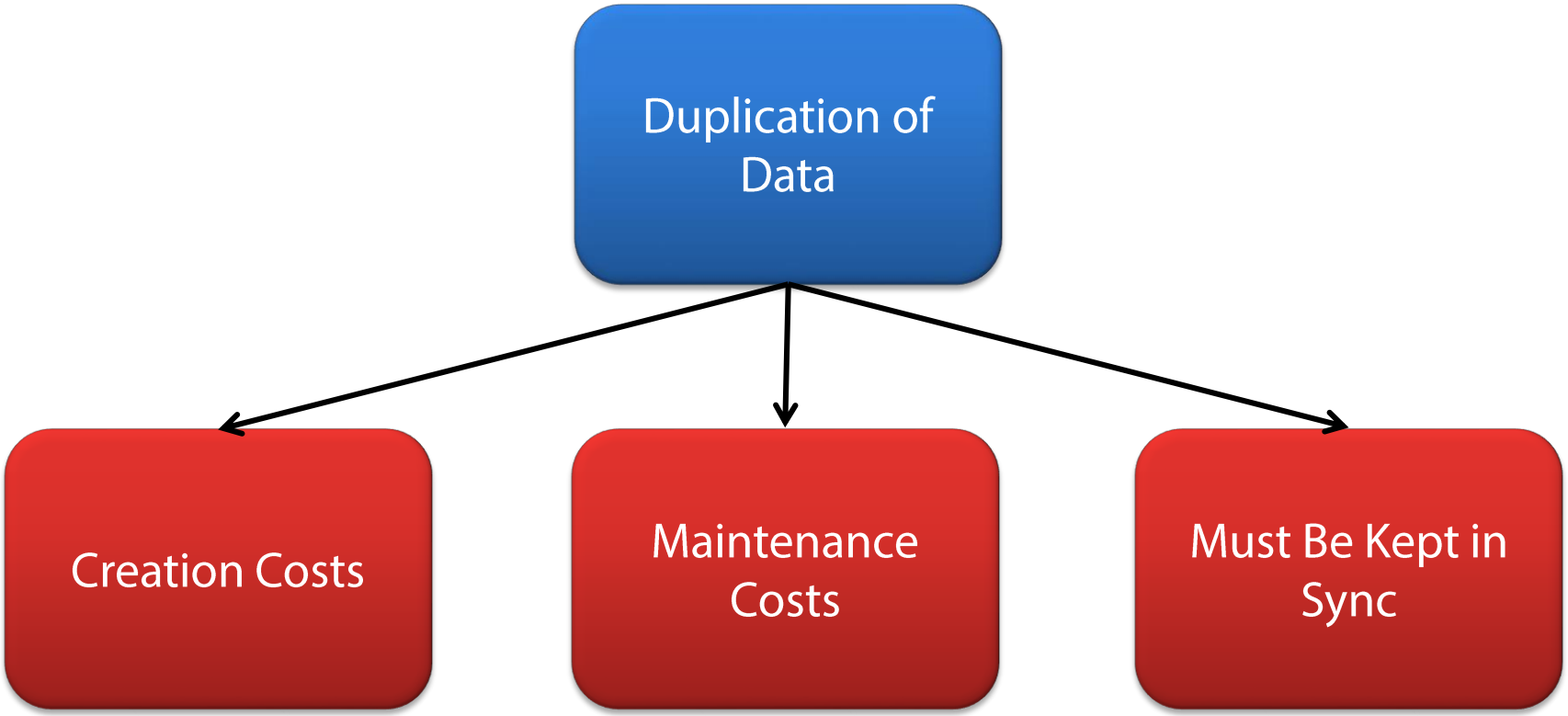
Why Do Indexes Matter?

How Data is Stored in a Table	Data Needed for One Statement	Resulting Data Access Operation
Randomly distributed	Small fraction	All data for a table must be read and processed

What is an Index?

Definition	Separate data structure designed to improve retrieval of rows
Index Key	The columns over which the index is built
Row ID	Points to exact location where the row is stored

Index Costs


Duplication of Data
Creation Costs
Maintenance Costs
Must Be Kept in Sync

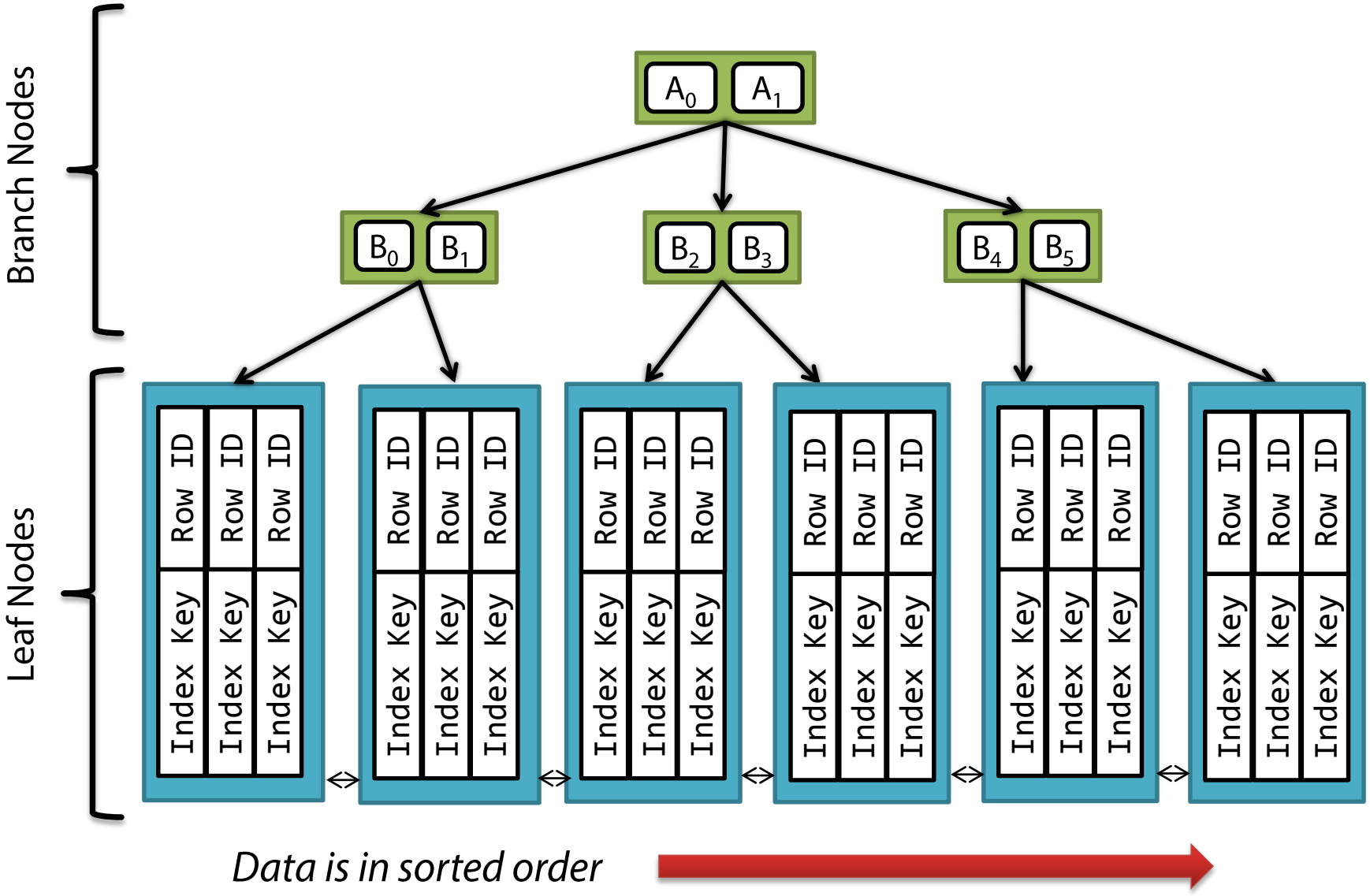
Module Outline

- Common index types
- Index column order
- Index selectivity

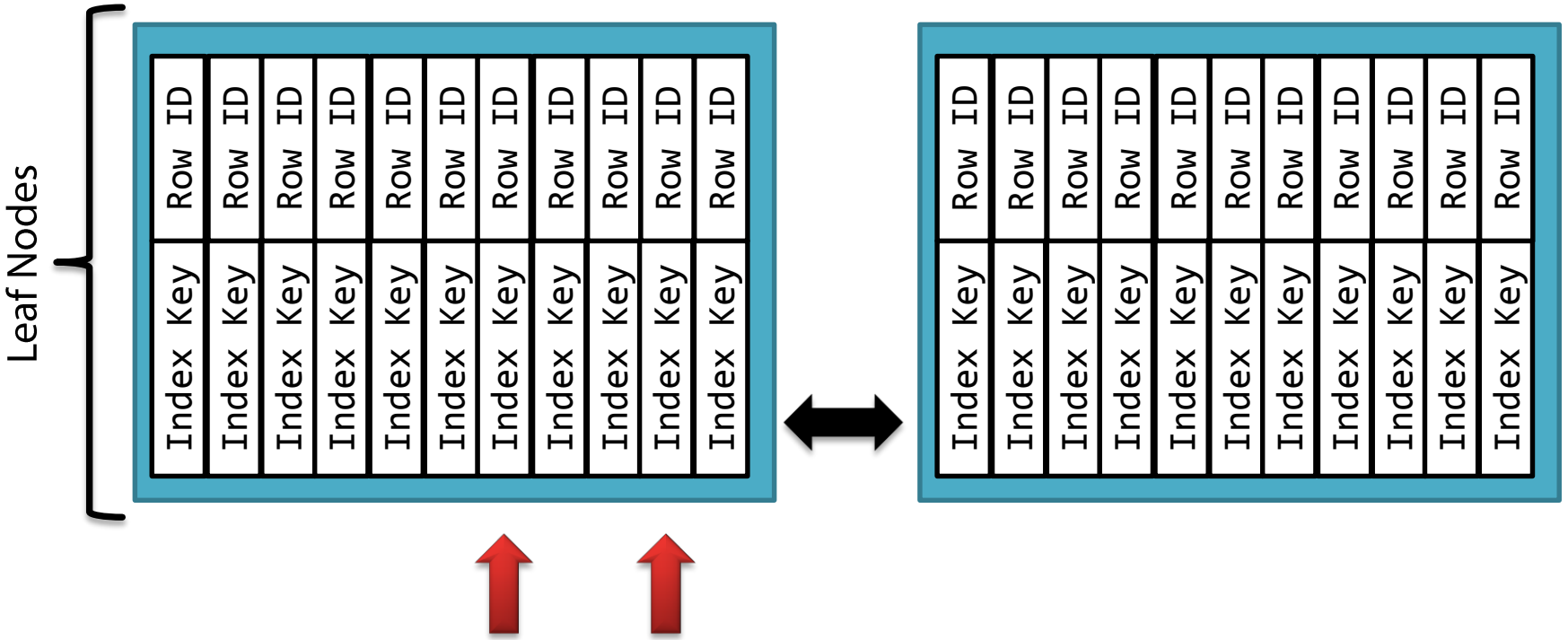
Standard (B-Tree) Index

- Most common type of index
- B stands for balanced

Standard (B-Tree) Index

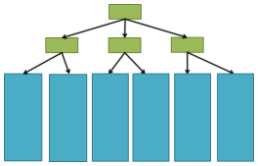


Reading Values From Leaf Nodes



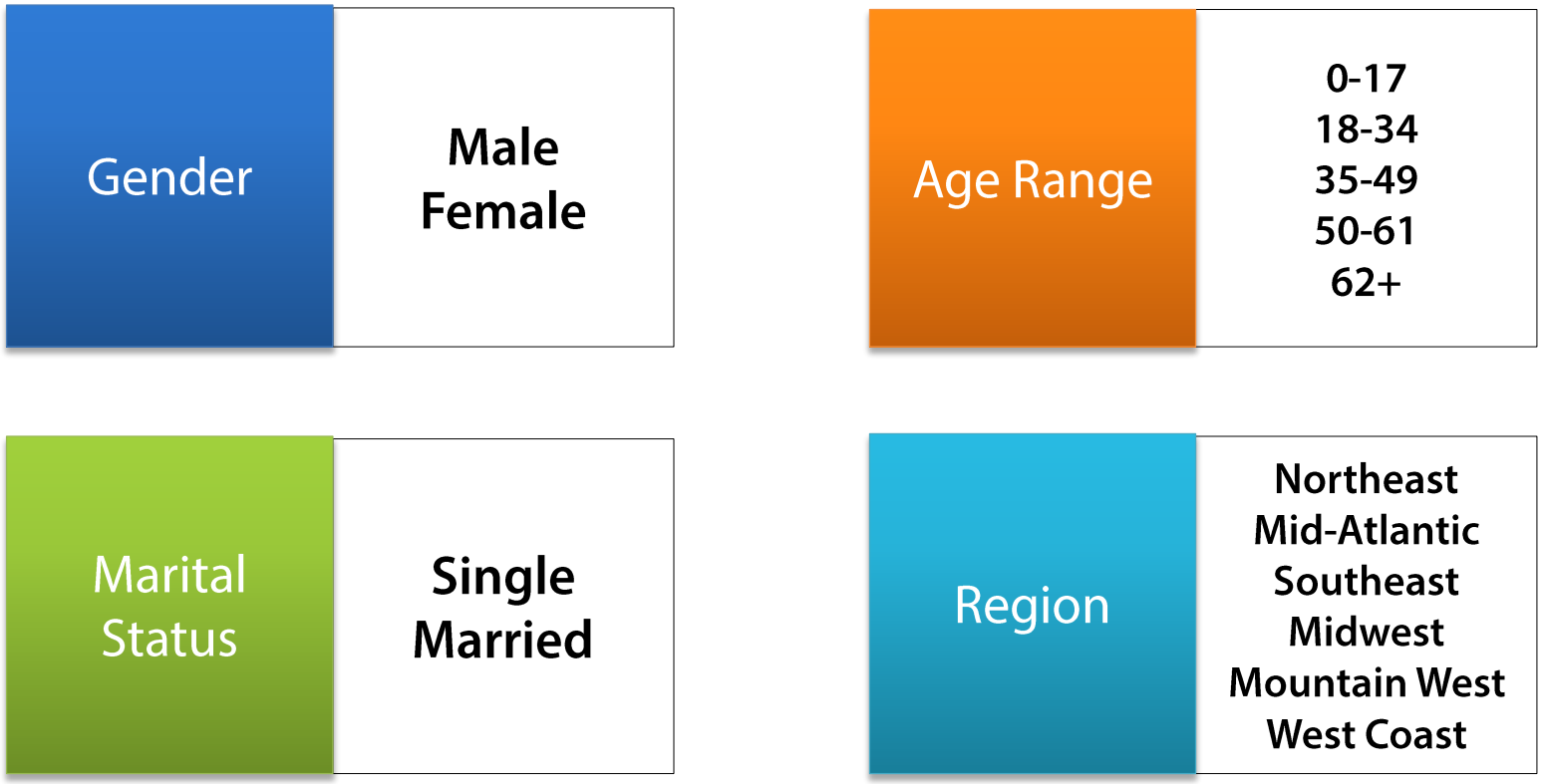
Data is in sorted order

Index Efficiency

	Index (B-Tree)	Table Scan
		
Search Time	O(log *n*)	O(*n*)
For 1,000,000 items	19	1,000,000

Bitmap Indexes

- Useful for columns with a low number of distinct values



Bitmap Indexes – Simple Example

Table Data

Student ID	Gender
1001	F
1002	F
1003	M
1004	F
1005	F
1006	M
1007	F
1008	M
1009	F
1010	M
1011	M
1012	F

Bitmap Index

Female	Male
1	0
1	0
0	1
1	0
1	0
0	1
1	0
0	1
1	0
0	1
0	1
1	0

Bitmap Indexes – Many Distinct Values

Table Data

Student ID	Age Range
1001	18-34
1002	18-34
1003	35-49
1004	35-49
1005	18-34
1006	62+
1007	50-61
1008	35-49
1009	35-49
1010	35-49
1011	35-49
1012	18-34

Bitmap Index

18-34	35-49	50-61	62+
1	0	0	0
1	0	0	0
0	1	0	0
0	1	0	0
1	0	0	0
0	0	0	1
0	0	1	0
0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0
1	0	0	0

Using Bitmap Indexes

Table Data

Student ID	Gender	Married	Age Range
1001	F	N	18-34
1002	F	Y	35-49
1003	M	Y	35-49
1004	F	Y	35-49
1005	F	Y	18-34
1006	M	Y	62+
1007	F	N	50-61
1008	M	Y	35-49
1009	F	Y	35-49
1010	M	N	35-49
1011	M	Y	35-49
1012	F	N	18-34

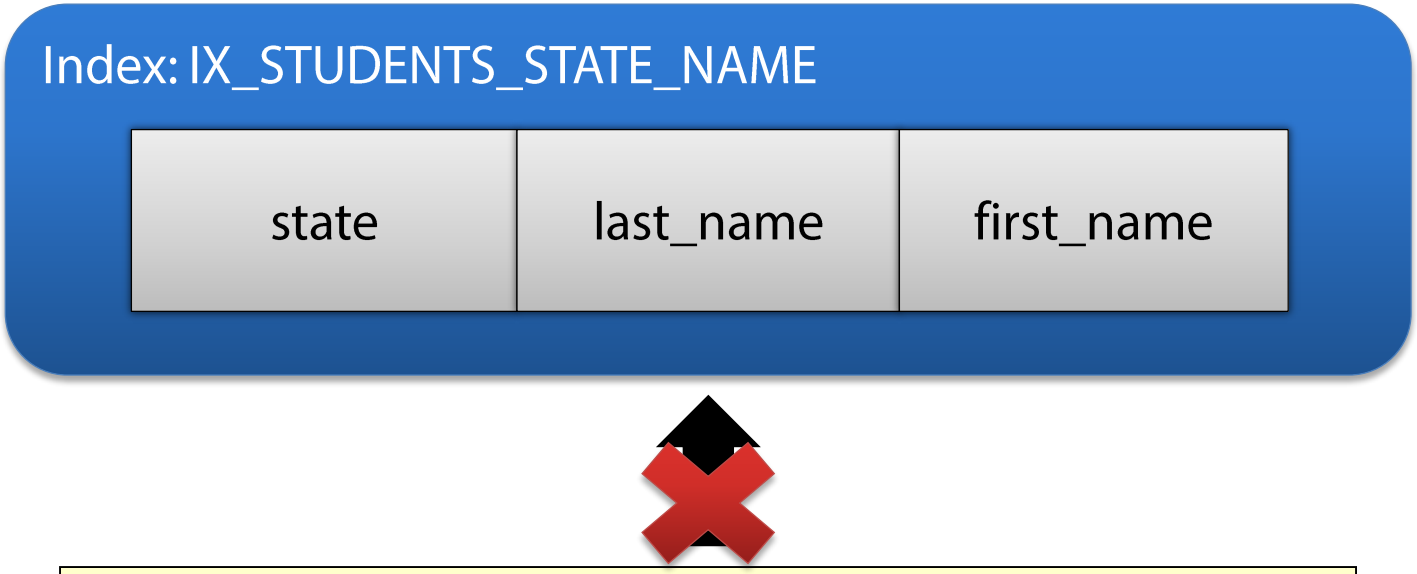
Bitmap Index

Female	Married	18-34	35-49	50-62	62+
1	0	1	0	0	0
1	1	1	0	0	0
0	1	0	1	0	0
1	1	0	1	0	0
1	1	1	0	0	0
0	1	0	0	0	1
1	0	0	0	1	0
0	1	0	1	0	0
1	1	0	1	0	0
0	0	0	1	0	0
0	1	0	1	0	0
1	0	1	0	0	0

Bitmap Index Usage

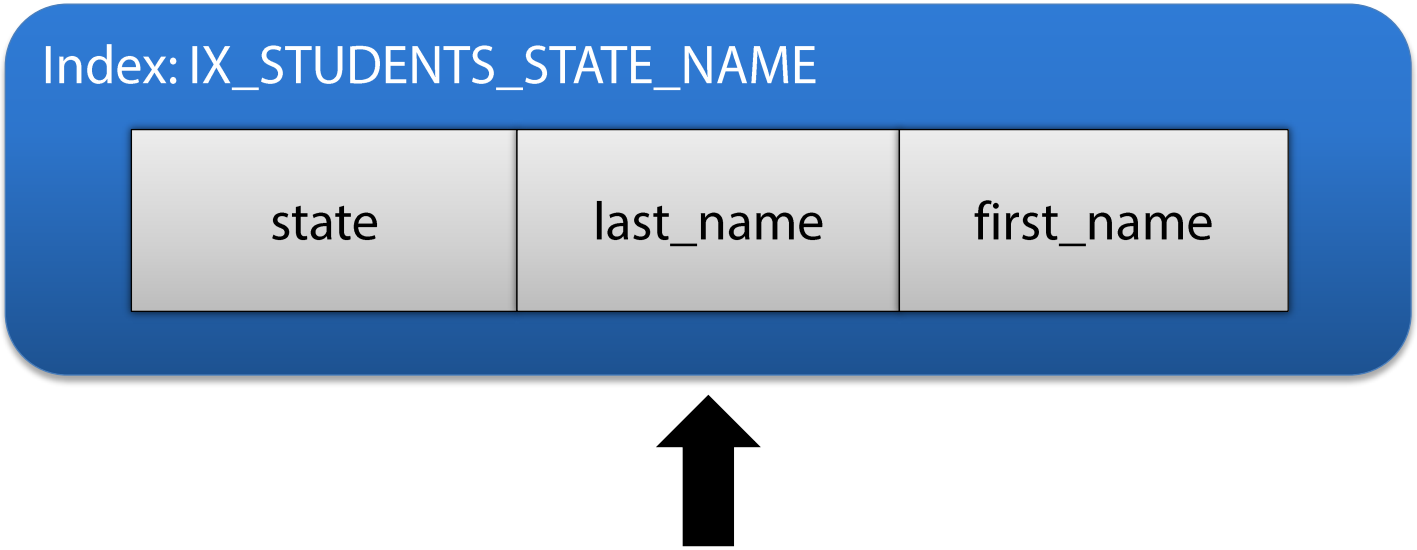
- Targeted for data warehouse applications
- Do not use in OLTP databases
- Only available in Oracle Enterprise Edition

Column Order Matters



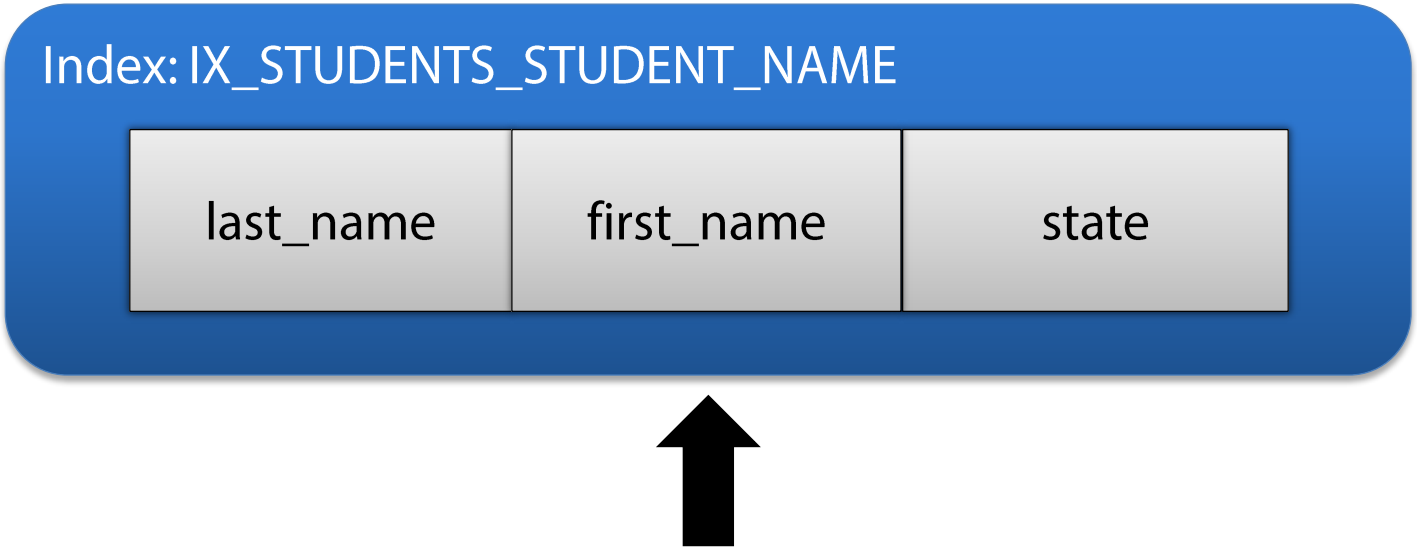

```
SELECT *
    FROM students
    WHERE last_name = 'Harris'
        AND first_name = 'Sam'
```

Add Column to Query



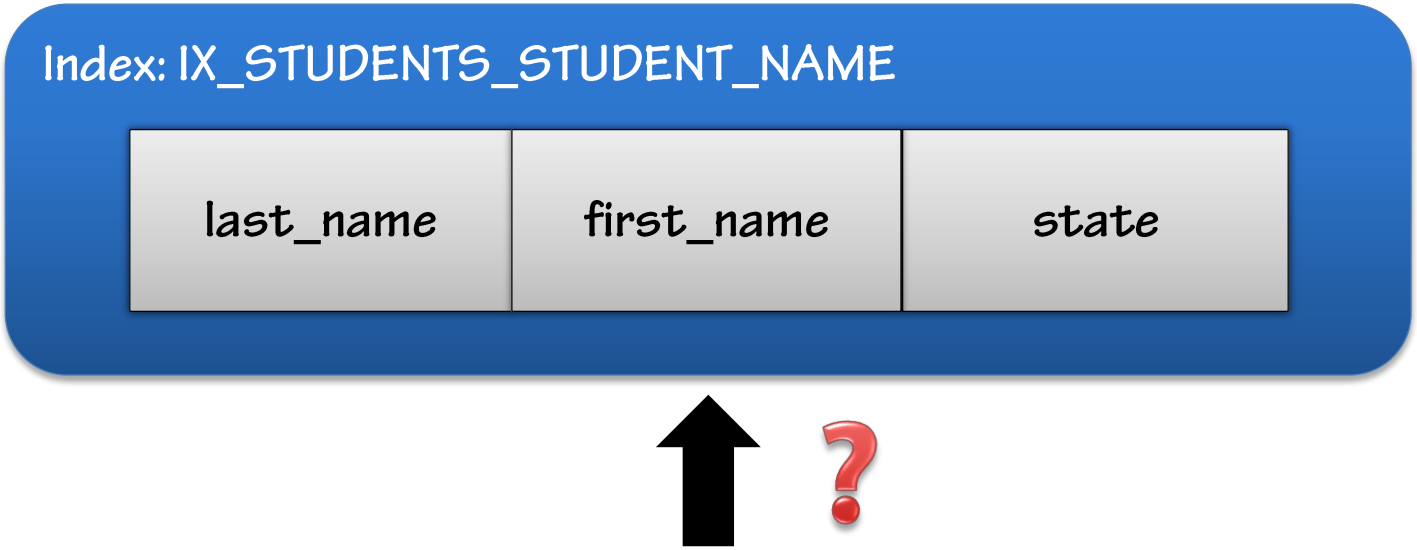

```
SELECT *
    FROM students
    WHERE last_name = 'Harris'
        AND first_name = 'Sam'
        AND state = 'CO'
```

Reordering the Index Columns




```
SELECT *
    FROM students
    WHERE last_name = 'Harris'
        AND first_name = 'Sam'
```

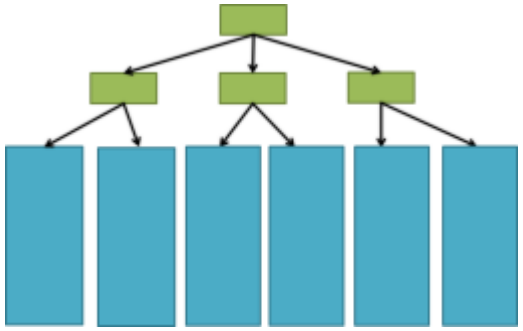
Reordering the Index Columns




```
SELECT *
    FROM students
    WHERE last_name = 'Harris'
        AND state = 'CO'
```

Using Non-Consecutive Index Columns

IX_STUDENTS_STUDENT_NAME
- last_name
- first_name
- state



Index access predicate
last_name = 'Harris'

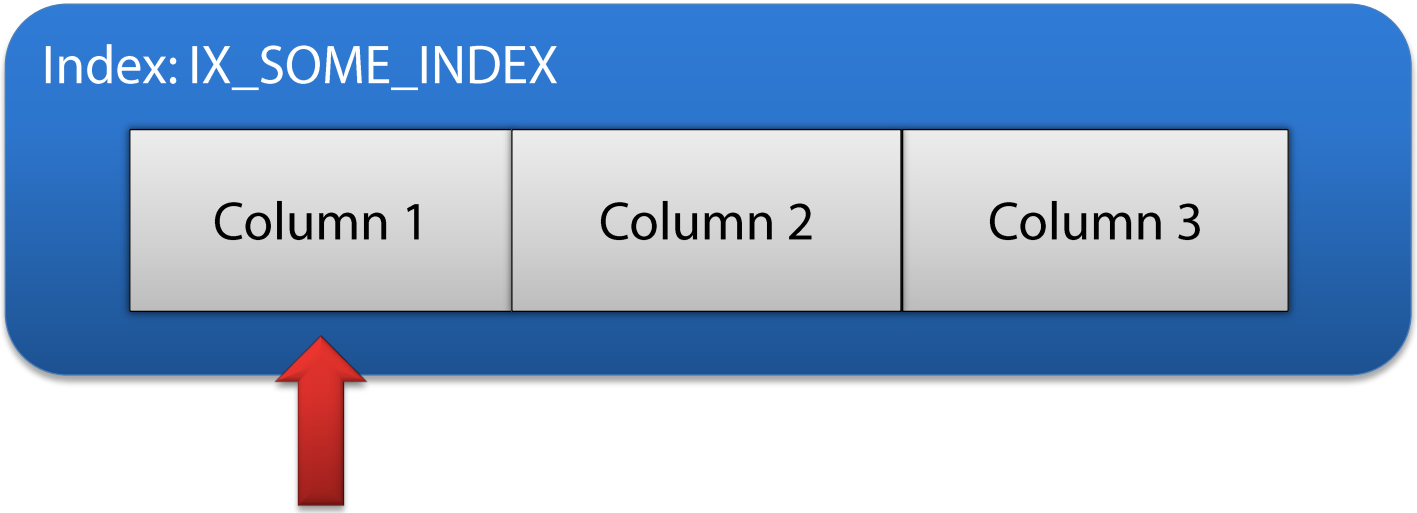
Index Keys

last_name	first_name	state
Harris	Al	WI
Harris	Amanda	IA
Harris	Ben	WA
Harris	Christie	CA
Harris	Fred	TX
Harris	George	CA
Harris	Jason	IL
Harris	Michelle	NY
Harris	Rachel	MI
Harris	Sam	CO
Harris	Tim	FL
Harris	Tom	CO

Index filter predicate
state = 'CO'

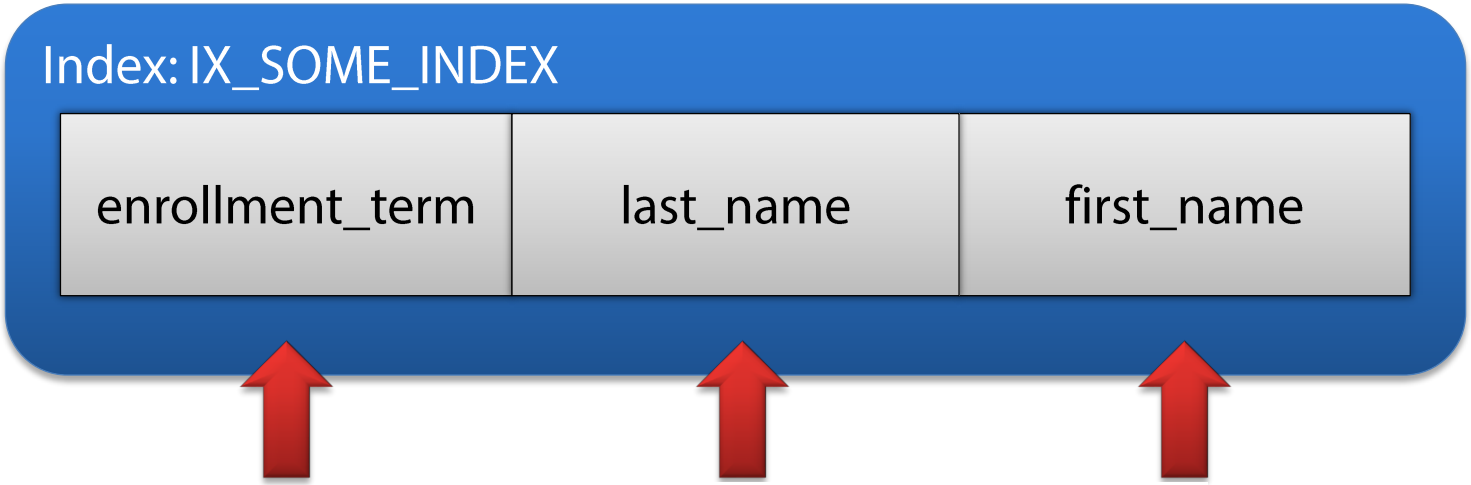
last_name	first_name	state
Harris	Sam	CO
Harris	Tom	CO

Index Skip Scan



Low cardinality column
(few distinct values)

Index Skip Scan



Probe with each unique value in column

Values in where clause are used

FA2008	Harris	Sam
FA2009	Harris	Sam
FA2011	Harris	Sam
FA2012	Harris	Sam
FA2013	Harris	Sam

Index Skip Scan

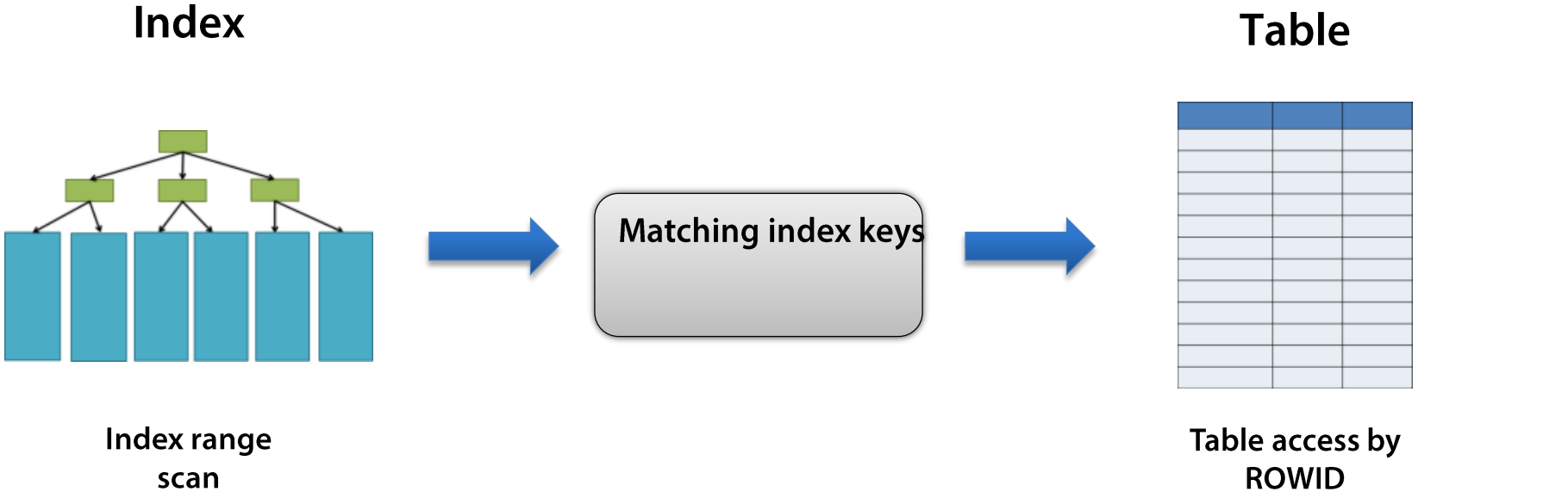
- **Advantages**
  - Can still use an index operation
  - Can reduce overall number of indexes
- **Disadvantages**
  - Not as efficient
  - Only applies in certain scenarios

Index Selectivity

Definition	Measure of how discerning the index is
Why it Matters	Selective indexes allow Oracle to more directly find just the data it needs
Keep in Mind	Favor indexes with a higher selectivity

Examples of Index Selectivity

Why Does Index Selectivity Matter



Highly Selective Index	Small number	Few table blocks read
Poor Selectivity Index	Large number	Most/all table blocks read

Formula for Index Selectivity

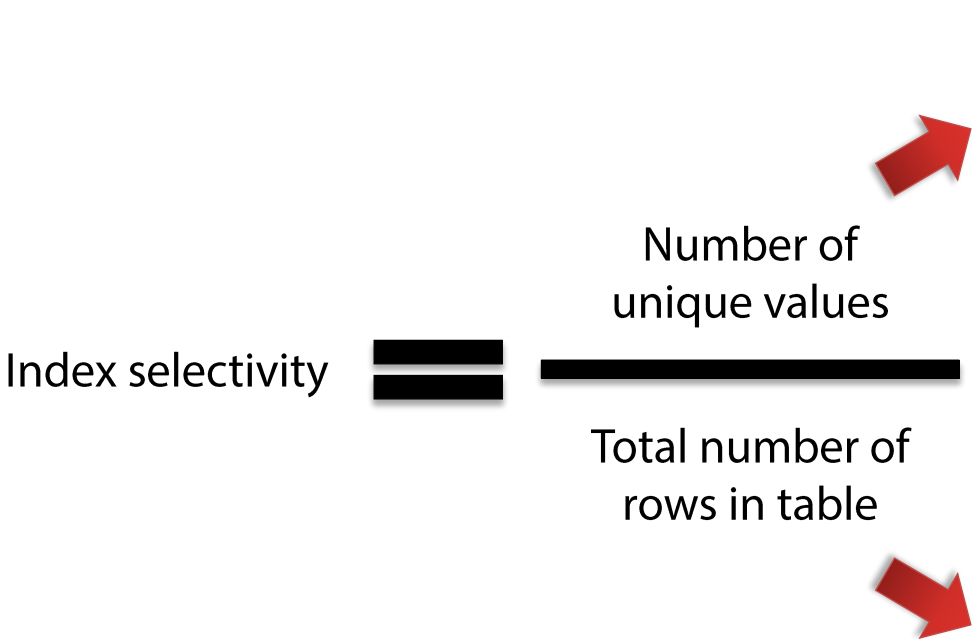
$$\text{Index selectivity} = \frac{\text{Number of unique values}}{\text{Total number of rows in table}}$$

Ideal selectivity = 1 (unique index)

$$\text{Expected rows per index key} = \frac{\text{Total number of rows in table}}{\text{Number of unique values}}$$

Lower numbers are better

Calculating Selectivity – Single Column Index




```
SELECT
    count(DISTINCT column_name)
    FROM table_name
```

```
SELECT count(1)
    FROM table_name
```

Index Selectivity – Using Database Statistics

Check when stats where last gathered on table

```
SELECT to_char(last_analyzed, 'YYYY-MM-DD hh24:mm')
    FROM all_tables
    WHERE owner = ‘<<schema name>>’
        AND table_name = ‘<<table name>>’
```

Gather statistics on table

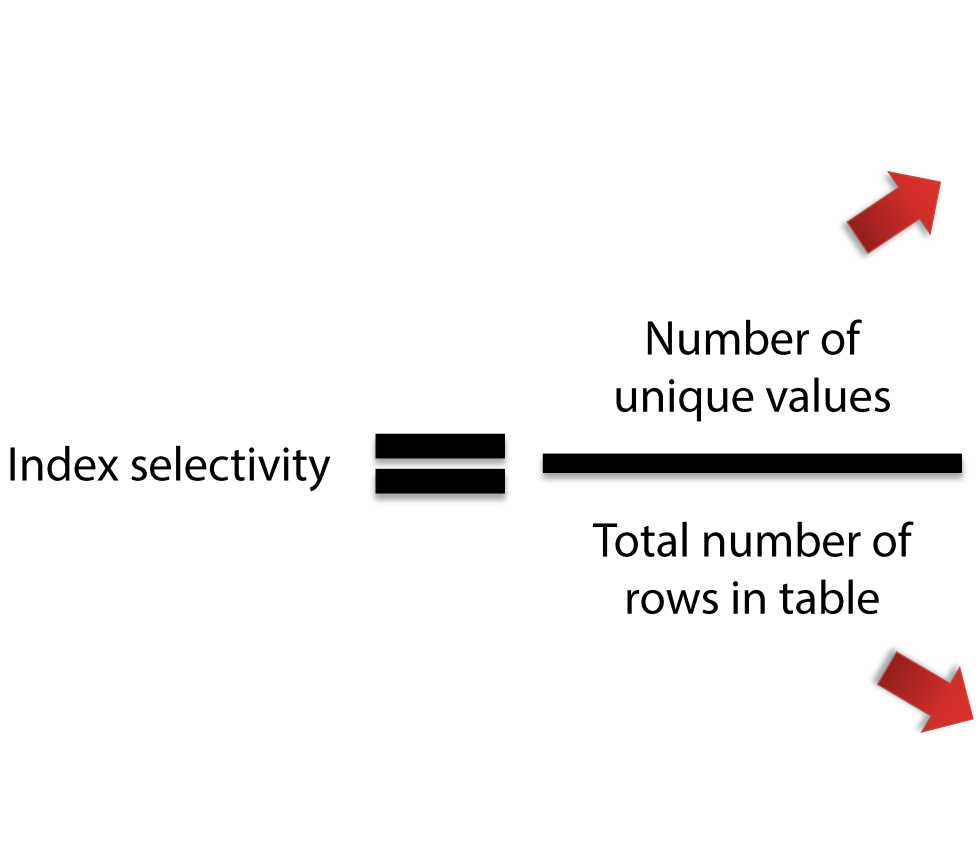
```
-- Compute all rows
DBMS_STATS.GATHER_TABLE_STATS(‘<<schema name>>',
    ‘<<table name>>‘);

-- Sample 25 percent of the rows
DBMS_STATS.GATHER_TABLE_STATS(‘<<schema name>>',
    ‘<<table name>>‘, estimate_percent => 25);
```

Getting Number of Distinct Values

```
SELECT column_name, num_distinct
    FROM all_tab_columns
    WHERE table_name = ‘<<table name>>'
    ORDER BY num_distinct DESC
```

COLUMN_NAME	NUM_DISTINCT
APPLICANT_ID	90000
TELEPHONE	90000
STREET_ADDRESS	88912
EMAIL	88832
LAST_NAME	14915
ZIP_CODE	12785
CITY	8301
FIRST_NAME	3769

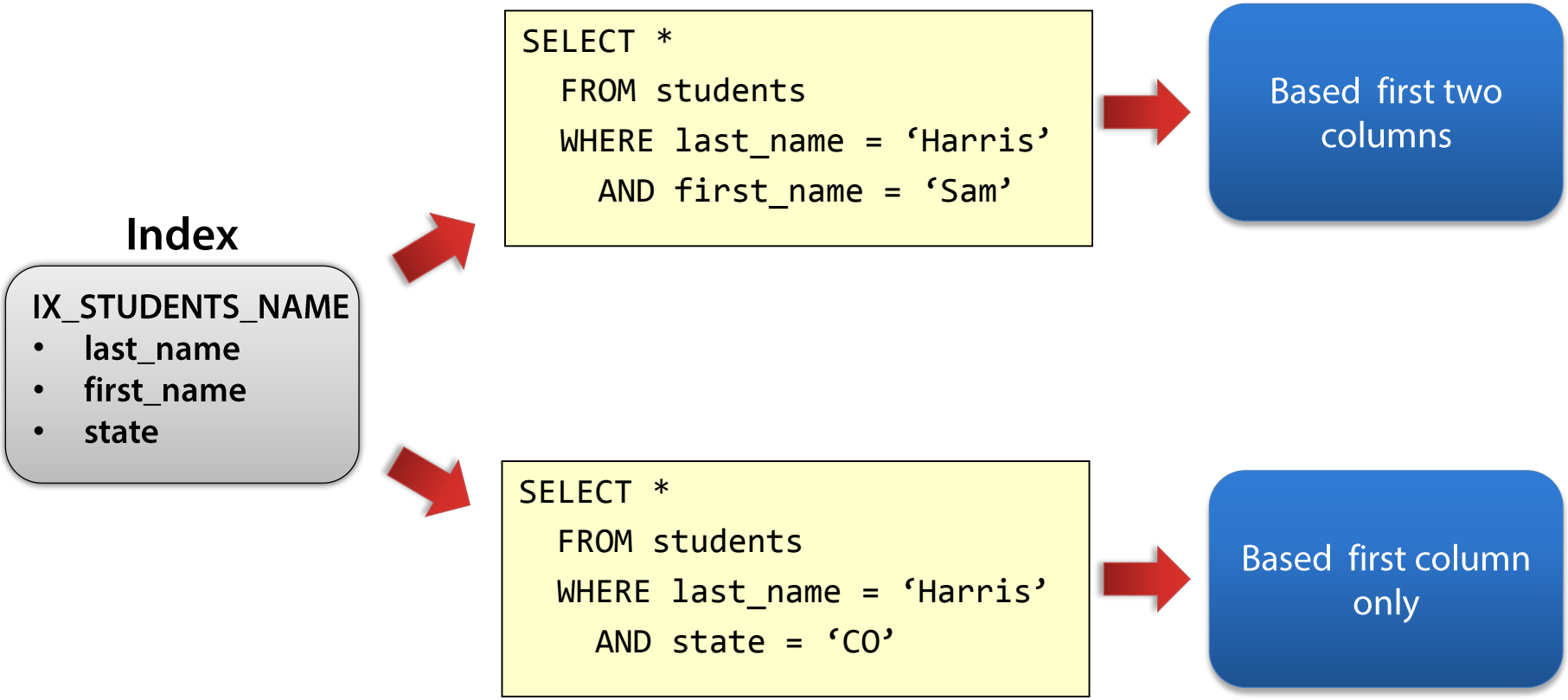
Calculating Selectivity – Multi Column Index




```
SELECT count(1) FROM
(
    SELECT DISTINCT
        column1, column2
    FROM table_name
)
```

```
SELECT count(1)
    FROM table_name
```

Selectivity Examples



Determining Index Column Order

- **Frequently used columns need to be at the front**
  - Assures Oracle can use the index
- **Pay attention to index selectivity**
  - Consider multiple columns to improve selectivity
- **Favor frequency of use over selectivity**

Advanced Indexing Techniques

David Berry
http://buildingbettersoftware.blogspot.com/

pluralsight
hardcore developer training

Module Overview

- Covering indexes
- Function based indexes
- Index compression
- Invisible indexes

Covering Indexes

Typical Index Operation

- Find matching ROW IDs in index
- Look up rows in table via ROW ID

Covering Index

- All required data is found in the index

Covering Index Benefits

Benefits	Drawbacks
• No need to lookup data in table • Save IO operations to read table	• Increase in index size • Cost to maintain index • Specific to one/few statements

Covering Index Summary

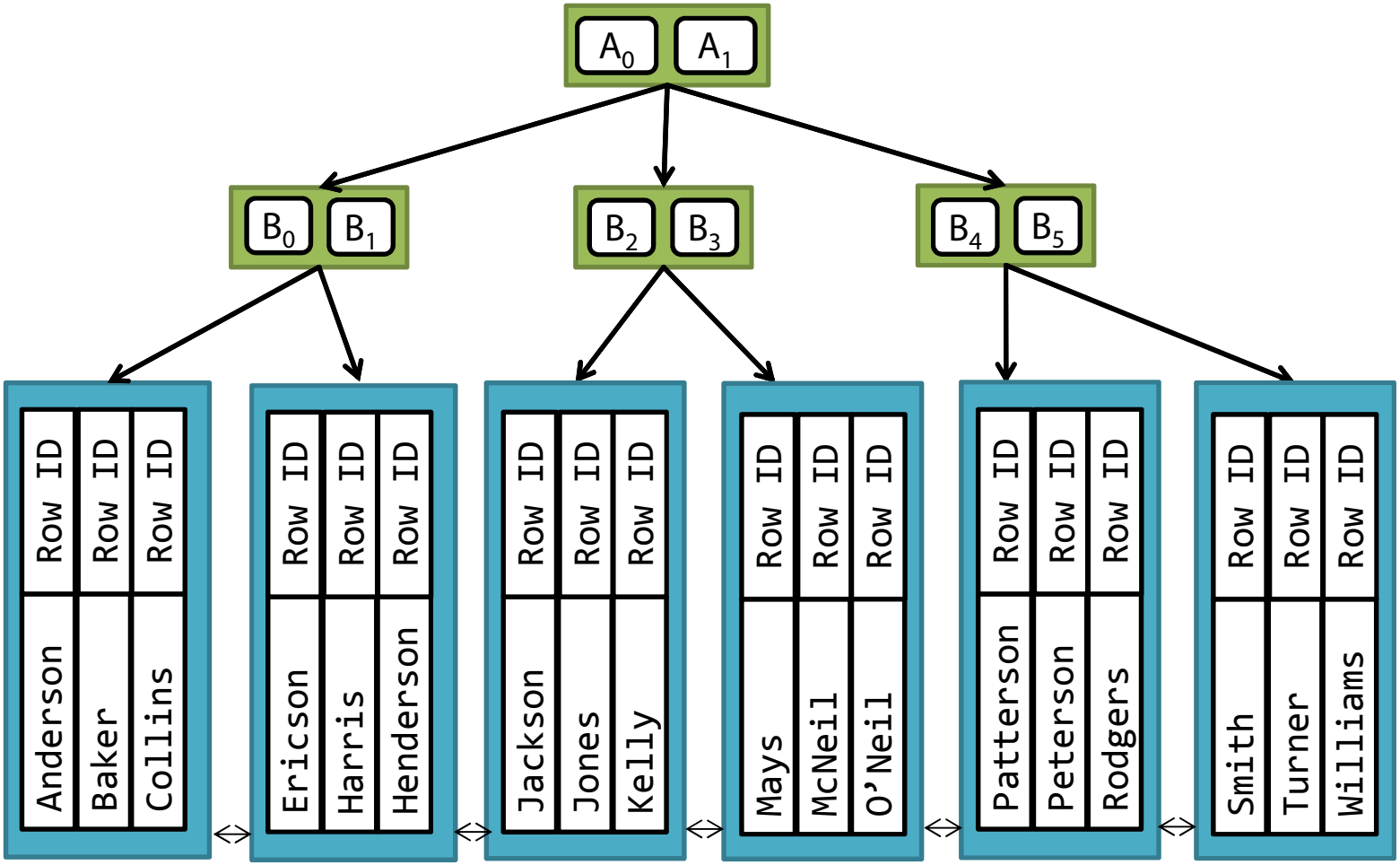
- **Consider for queries that closely match index they are using**
- **Not considered a silver bullet**
  - Adding columns adds overhead
  - Indexes are not meant to be mini-tables
- **Be sure to test**
  - Consider impacts to all statements, not just target query

Function Based Indexes

Index built over actual value in table

Index is built over a derived value

Standard Index Over Last Name Column



Case Sensitivity in Oracle

A != a

Comparisons in Oracle are case sensitive

Problematic for name searches

Oracle is Case Sensitive

```
SELECT *
    FROM students
    WHERE last_name = 'McNeil'
```

9 records

```
SELECT *
    FROM students
    WHERE last_name = 'mcneil'
```

0 records

student_id	last_name	first_name
101971	McNeil	Alan
94083	McNeil	Andrew
16027	McNeil	Desmond
26345	Mcneil	Dirk
67791	McNeil	James
60325	McNeil	Jason
13195	McNeil	Jonathan
28260	McNeil	Mary
34371	McNeil	Nicole
51820	McNeil	Stella
42938	Mcneil	William

11 records total

Function in the WHERE Clause

```
SELECT *
    FROM students
    WHERE UPPER(last_name) =
        UPPER('<<user supplied value>>')
```

FULL TABLE SCAN ALERT!

High amounts of IO and
CPU consumption ahead

Using a Function Based Index

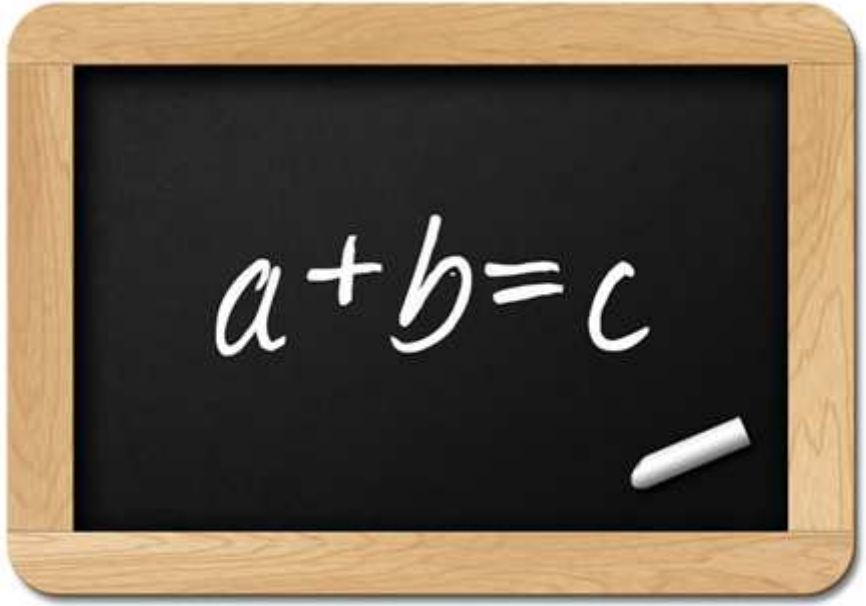
```
CREATE INDEX ix_students_last_name
    ON students ( UPPER(last_name) );
```

```
SELECT *
    FROM students
    WHERE UPPER(last_name) =
        UPPER('<<user supplied value>>')
```

Function Based Index Rules

- **Both built in functions and user defined functions can be used**
  - UPPER(), SOUNDEX(), SUBSTR() are common
- **Functions must be deterministic**
  - Always return the same value for a given input
- **Aggregate functions cannot be used**
  - COUNT(), MIN(), MAX(), AVG(), STDDEV()
  - Any analytic functions

Function Performance



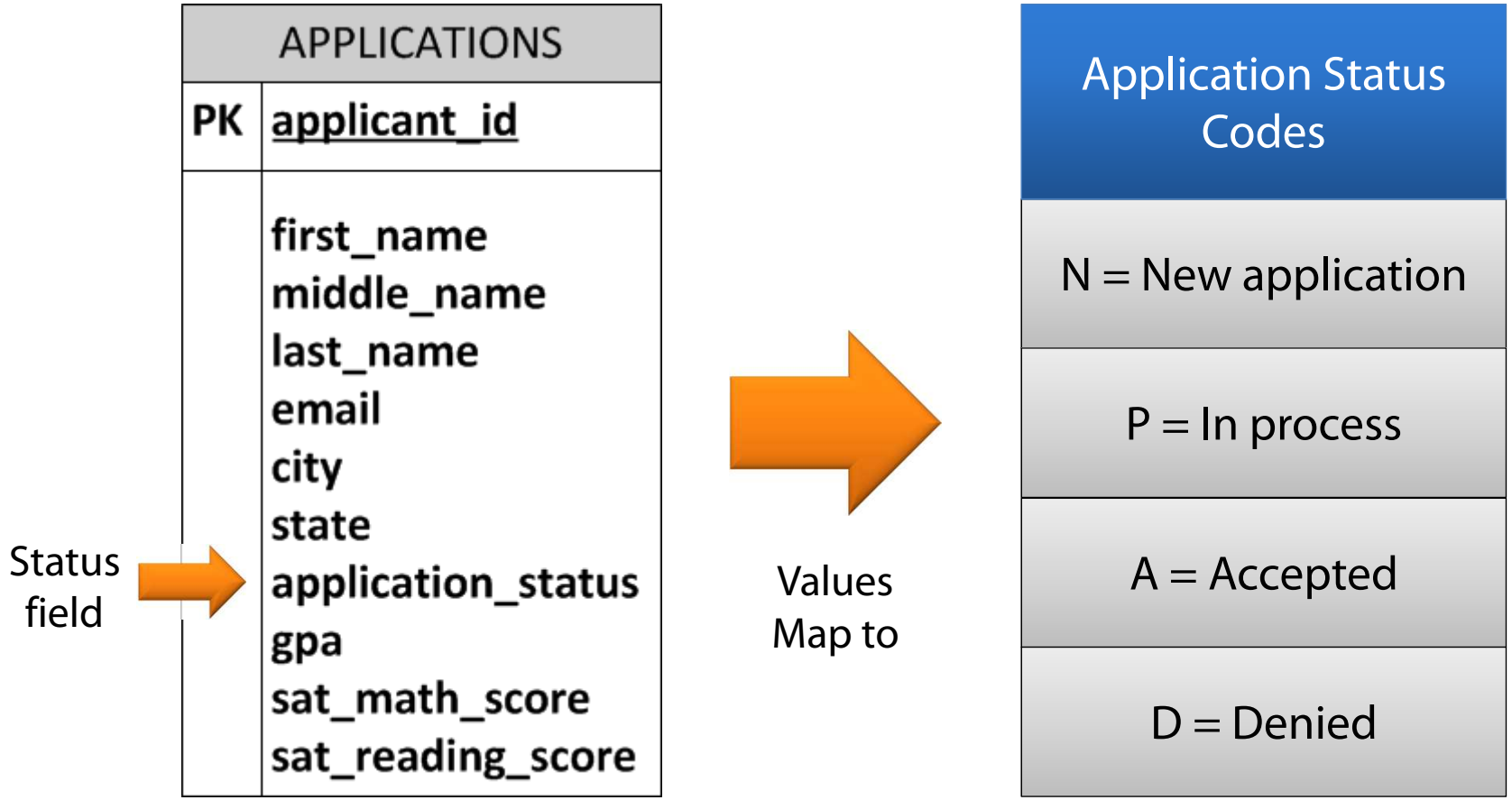
- Keep functions short and concise
- Slow functions will impact DML operations

Functions in a Composite Index

```
CREATE INDEX ix_students_name_state
    ON students
    (
        UPPER(last_name),
        UPPER(first_name),
        state
    );
```

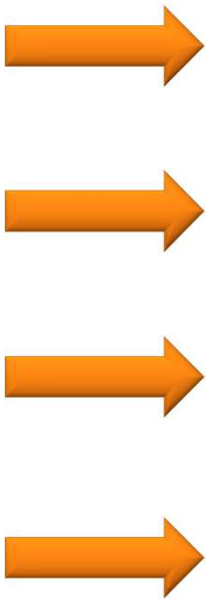
```
SELECT *
    FROM students
    WHERE
        UPPER(last_name) = UPPER('<<last name value>>')
        AND UPPER(first_name) = UPPER('<<first name value>>')
```

Indexing a Subset of Rows



Relative Occurrence of Status Codes

Application Status Codes	Relative Occurrences
N = New application	~200
P = In process	~50
A = Accepted	> 10,000
D = Denied	> 50,000



Indexing a Subset of Rows

- **Standard B-Tree Index**
  - All rows will be indexed
  - Large size on disk

- **Function Based Index**
  - Use a case statement to only index rows of interest
  - Values we want map to the existing column value
  - Values we don't want map to null

CREATE INDEX for Subset of Rows

```
CREATE INDEX fx_applications_app_status
    ON applications
        (
            CASE application_status
                WHEN 'N' THEN 'N'
                WHEN 'P' THEN 'P'
                ELSE NULL
             END
        )
```

Corresponding SQL Statement

```
SELECT *
    FROM applications
    WHERE
        (
            CASE application_status
                WHEN 'N' THEN 'N'
                WHEN 'P' THEN 'P'
                ELSE NULL
             END
        )
        IN ('N', 'P')
```

Using a User Defined Function

```
CREATE OR REPLACE FUNCTION map_application_status_value
        (application_status IN VARCHAR)
    RETURN VARCHAR
    DETERMINISTIC
    IS mapped_value VARCHAR2(1);
    BEGIN
        mapped_value :=
            CASE application_status
                WHEN 'N' THEN application_status
                WHEN 'P' THEN application_status
                ELSE NULL
            END;
        RETURN (mapped_value);
    END;
```

Using the User Defined Function

Create Index Statement

```
CREATE INDEX fx_applications_app_status
    ON applications
   (map_application_status_value(application_status) );
```

Using the Index in a SELECT statement

```
SELECT * FROM applications
    WHERE
        map_application_status_value(application_status)
        IN (‘N‘, ‘P’)
```

Index Compression

What is it?

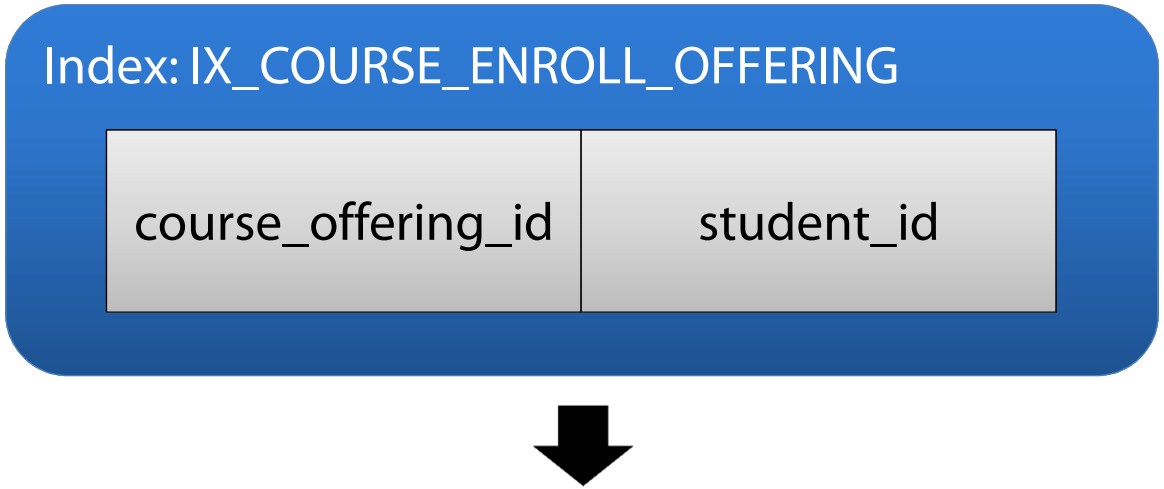
Repeated values at the front of the index are compressed into a single prefix value

How does this help?

Storage space for the index is reduced

IO to read the index is reduced

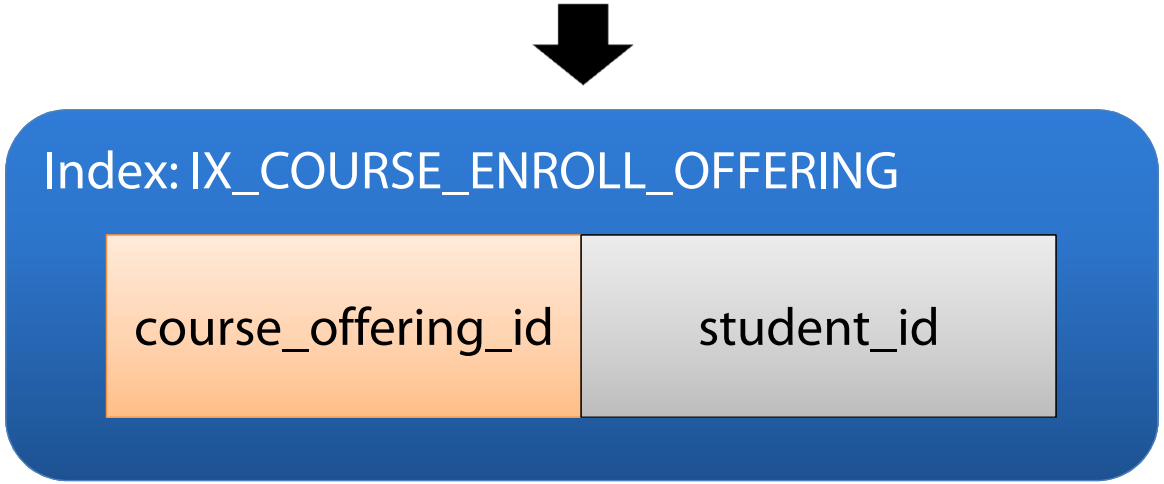
Index Compression Example



course_offering_id	student_id
25478	200211
25478	200215
25478	200256
25479	200273
25479	200291
25479	200294

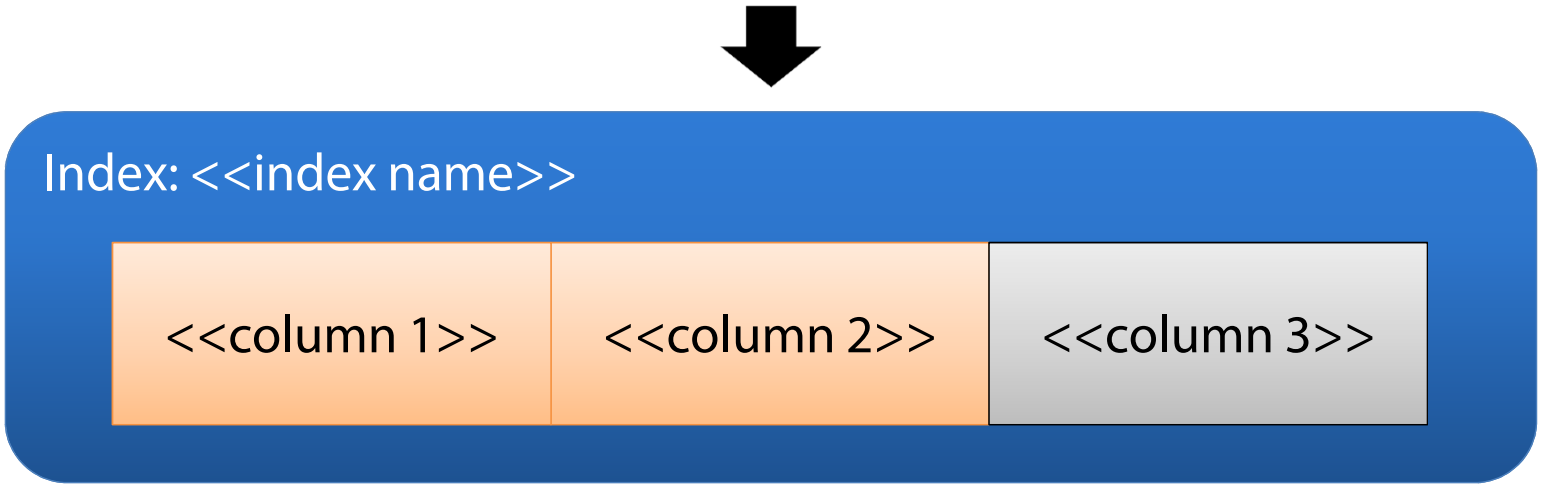
Index Compression Syntax

```
CREATE INDEX ix_course_enroll_offering
    ON course_enrollments
        (course_offering_id, student_id)
    COMPRESS 1;
```



Compressing Multiple Index Columns

```
CREATE INDEX <<index_name>>
    ON <<table name>>
        (<<column 1>>, <<column 2>>, <<column 3>>)
    COMPRESS 2;
```



Index Compression Strategy

Benefits	Drawbacks
• Disk space savings • Reduction in IO • Smaller buffer cache footprint • Can use in both OLTP and data warehouse environments	• Not applicable to all indexes • Increase in CPU

Index Compression Summary

- **Use with large number of repeated values in front of key**
  - Foreign key column indexes can be good candidates

- **Experiment with different key compression sizes**

Invisible Indexes

- Index exists but is not used by the Oracle Optimizer
- Can be used by specific sessions via a session parameter
- Index is still maintained by Oracle, just not used

Invisible Index Usage

Test New Index	• Test in dedicated session • Make available only when ready
Dropping an Index	• Soft drop the index • Quickly turn back on if need be

Commands for Invisible Indexes

Create an Invisible Index

```
CREATE INDEX <<index name>>
    ON <<table name>> (<<column 1>>, <<column 2>>, …)
    INVISIBLE;
```

Change Visibility of an Existing Index

```
ALTER INDEX <<index name>> INVISIBLE;

ALTER INDEX <<index name>> VISIBLE;
```

Using Invisible Indexes

Get Current Visibility of an Index

```
SELECT index_name, visibility
    FROM all_indexes
    WHERE index_name = '<<index name>>'
```

Use Invisible Indexes for a Session

```
ALTER SESSION SET optimizer_use_invisible_indexes=true;

ALTER SESSION SET optimizer_use_invisible_indexes=false;
```

Advanced Indexing Techniques Summary

Covering Index	When Oracle can get all the data it needs from the index
Function Based Index	Ability to build an index over a computed value
Index Compression	Compress repeated parts of the index key, saving disk space and IO
Invisible Indexes	Ability to hide an index from the optimizer

Application Indexing Practices

David Berry
http://buildingbettersoftware.blogspot.com/

pluralsight
hardcore developer training

Module Outline

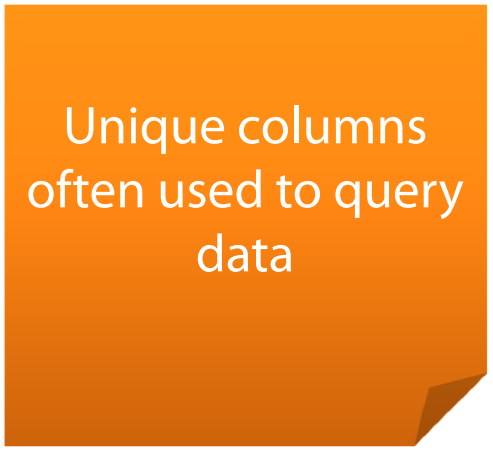
Primary Key Columns

Automatically indexed by Oracle

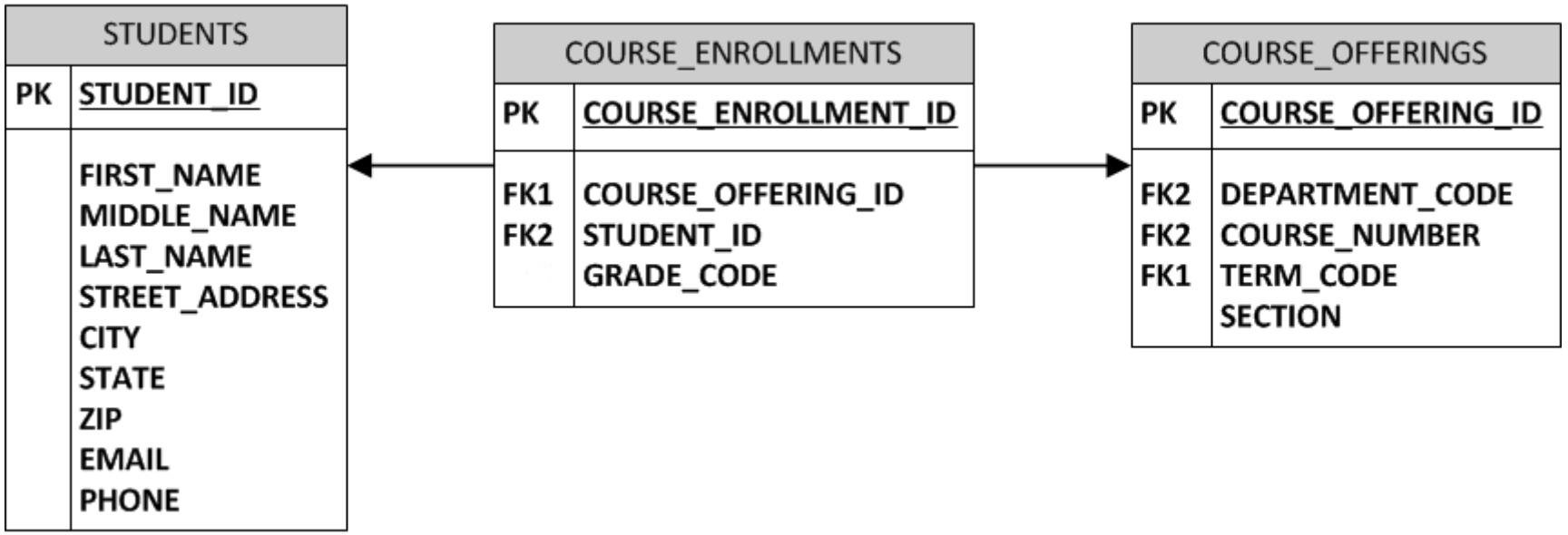
Consider giving a meaningful name

Define a primary key on every table

Unique Columns



Foreign Key Columns



Index for join performance

Queries often follow foreign key constraints

Columns Frequently Used in WHERE Clauses

Columns used to query and search data

Give attention to date columns

Consider adding columns to FK indexes

Identify clusters of columns used together

Indexing Strategy Wrap Up

- **Get execution plans for important SQL statements**
- **Important SQL statements**
  - Use largest tables
  - Most critical business functions
  - Most complex statements
- **Determine if indexes match statement needs**
  - Iterate until all statements perform well

Costs Associated With Indexes

If indexing columns improves performance, why don't we create an index for every column in the table?

A

Indexes improve query performance but degrade DML performance and use storage space

Storage Space of Indexes

- **Each index is a separate data structure**
  - Takes space on disk
  - Takes memory when loaded into buffer cache

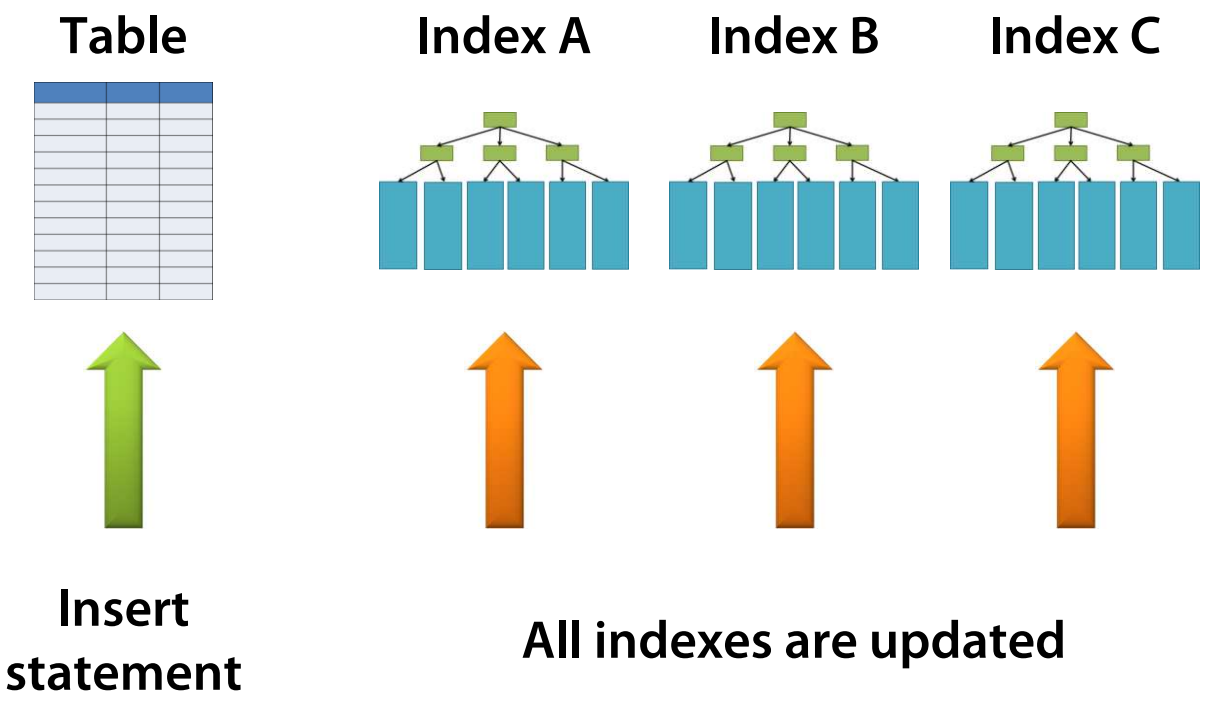
- **Query to find space used by index**

```
SELECT ix.table_name, ix.index_name, s.blocks, s.bytes
    FROM all_indexes ix
    INNER JOIN dba_segments s
        ON ix.index_name = s.segment_name
        AND ix.table_owner = s.owner
    WHERE ix.table_name = '<<table name>>'
```

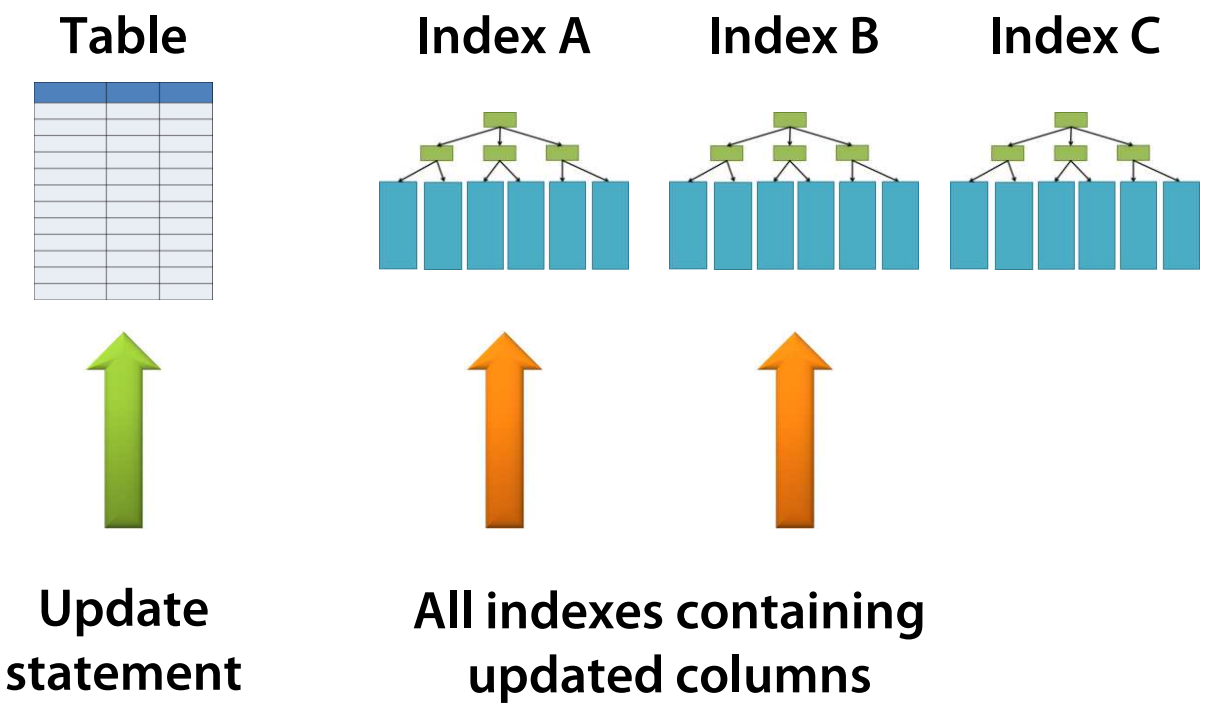
Why Does Storage Space Matter

- SAN space is expensive
- Storage is cheap, fast storage is not

Table Inserts



Updates to Table Row



Index Overhead Summary

- **Indexes must be kept in sync with table data**
- **Accepting some overhead is normal**
- **Problems occur when indexes aren't used**
  - Paying the cost of the index for no benefit

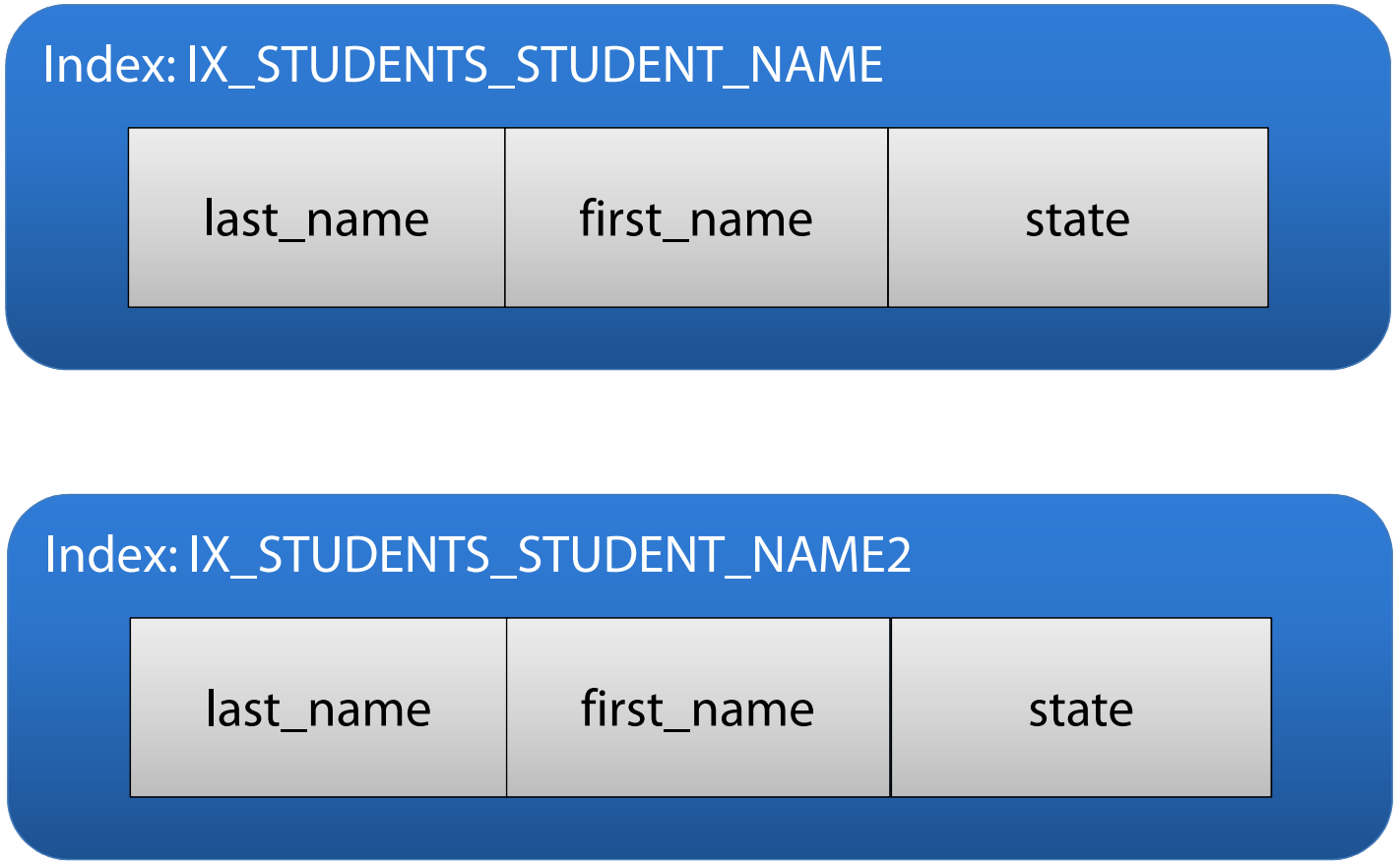
Demo – Index Overhead

APPLICATIONS_ONE_INDEX	APPLICATIONS_OVER_INDEXED
Primary key index	Primary key index
One additional index	Ten additional indexes
Two total indexes	Eleven total indexes

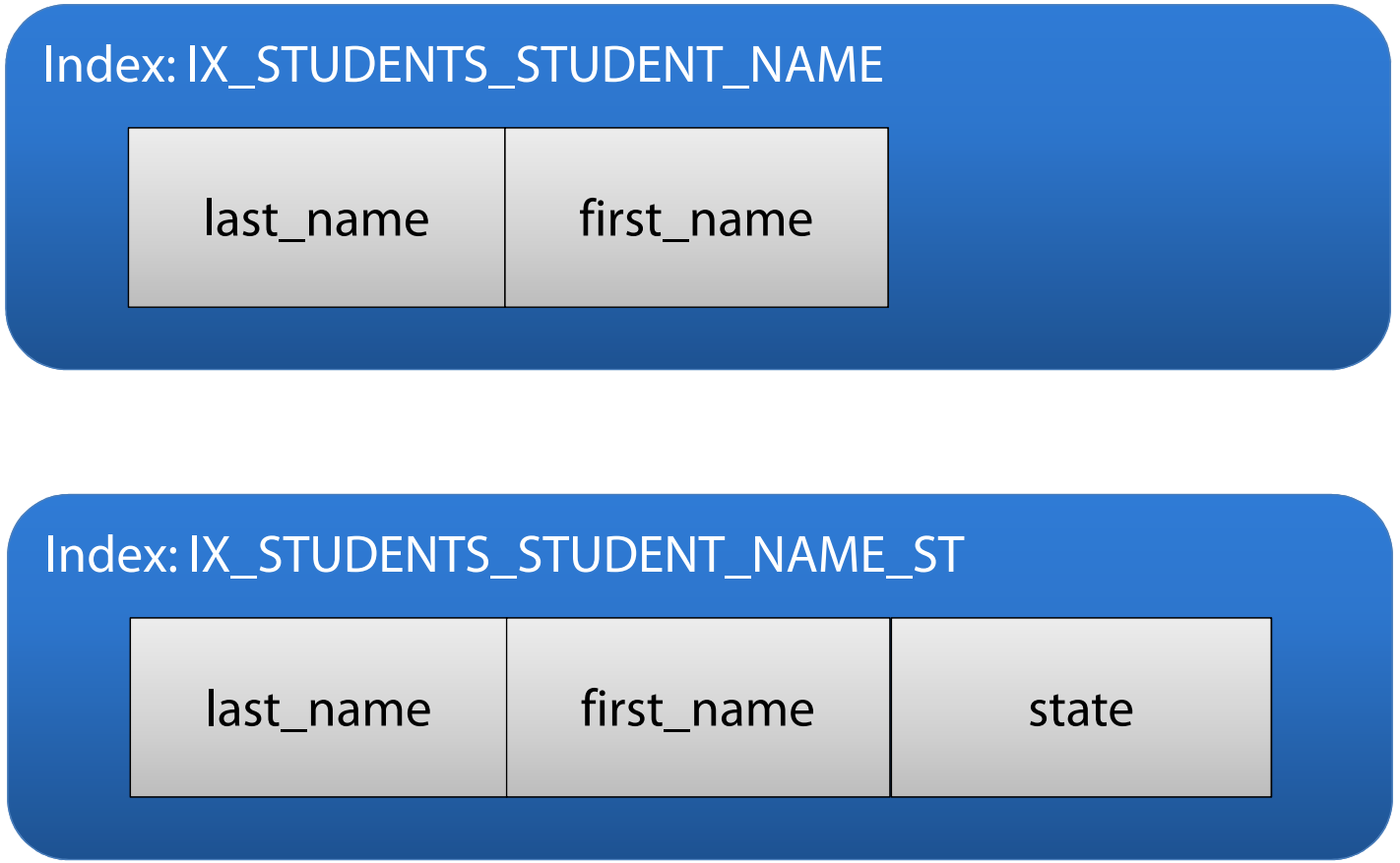
Index Overhead Demo Results

	One Index + Primary Key	Ten Indexes + Primary Key
Elapsed Time	55.871 sec	74.893 sec
CPU Time	12.21 sec	16.84 sec
Block Changes	421.010	1,365,728

Duplicate Indexes



Similar Indexes



Index Usage Flag

- Toggle index usage monitoring on/off

```
ALTER INDEX <<index name>> MONITORING USAGE;

ALTER INDEX <<index name>> NONMONITORING USAGE;
```

- Check for index usage

```
SELECT table_name, index_name,
        monitoring, usage,
        start_monitoring, end_monitoring
    FROM v$object_usage
    WHERE index_name = '<<index name>>';
```

Use V$Sql_plan View

```
WITH index_usage AS
(
    SELECT pl.sql_id, pl.object_owner, pl.object_name, pl.operation,
            pl.options, count(1) as use_count
        FROM v$sql_plan pl
        WHERE pl.operation = 'INDEX'
        GROUP BY pl.sql_id, pl.object_owner, pl.object_name, pl.operation,
    pl.options
)
SELECT
        ix.table_owner, ix.table_name, ix.index_name, iu.operation,
        iu.options, ss.sql_text, ss.executions
    FROM all_indexes ix
    LEFT OUTER JOIN index_usage iu
             ON ix.owner = iu.object_owner
             AND ix.index_name = iu.object_name
    LEFT OUTER JOIN v$sqlstats ss
        ON iu.sql_id = ss.sql_id
    WHERE ix.owner = '<<owner name>>'
    ORDER BY ix.table_name, ix.index_name;
```

Oracle AWR and Index Usage

- Separately licensed product
- DBA should be able to generate a report

Index Usage and Cost Summary

Indexes have a cost

- Disk storage space
- Overhead to maintain

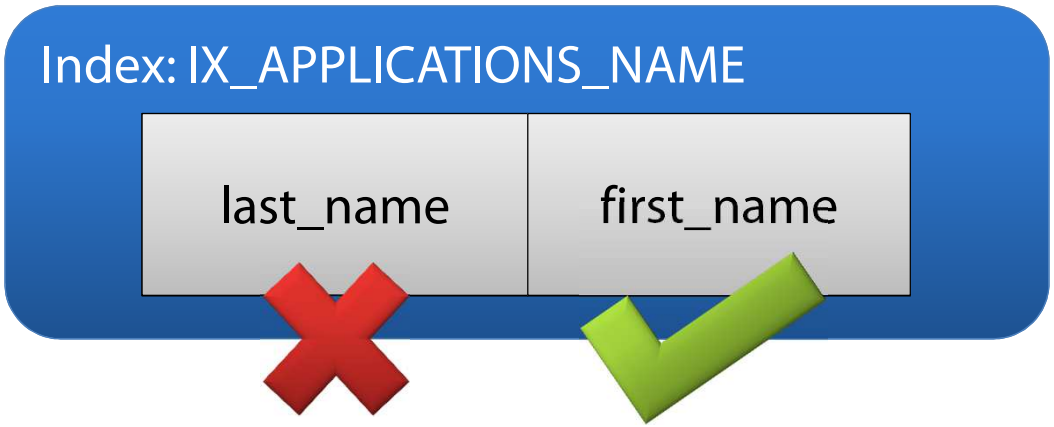
Evaluate the tradeoff

- Frequently used indexes are worth it
- Find unused indexes and consider dropping

Why Isn’t Oracle Using My Index?

Missing Leading Edge of Index?

```
-- Index Columns (last_name, first_name)
SELECT
    application_id, first_name, last_name, …
FROM applications
WHERE first_name = 'Jason'
```



Index will not be used

Leading Edge of Index is Required

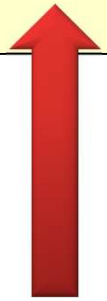
- Always include the first column in the WHERE clause

```
-- Index Columns (last_name, first_name)
SELECT
    application_id, first_name, last_name, …
FROM applications
WHERE first_name = ‘Jason’
    AND last_name = ‘Harris’
```

- Consider a different column order in the index

Index Not Selective Enough?

```
SELECT
    application_id, first_name, last_name, …
FROM applications
WHERE state = ‘CA’
```

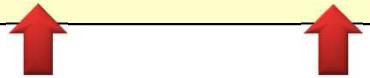


Not selective enough
Would return ~10% of the rows on its own

Improve the Selectivity

- **Add additional columns to the index**

```
CREATE INDEX ix_applicants_state
    ON applications
    (state, last_name, first_name);
```



Additional columns

- **Use a combination of columns in your SQL statement**

```
SELECT
    application_id, first_name, last_name, …
FROM applications
WHERE state = 'CA'
    AND last_name = '<<last name>>'
```

Leading Wildcards in a LIKE Clause

```
SELECT
    application_id, first_name, last_name, …
FROM Applications
WHERE last_name LIKE ‘%Smith%’
```

Will not use index due
to leading wildcard

Searching Textual Data

- **Eliminate the leading wildcard**

```
SELECT
    application_id, first_name, last_name, …
FROM Applications
WHERE last_name LIKE ‘Smith%’
```

- **Full text indexing and search solutions**
  - Oracle Text - http://bit.ly/RoGeht
  - Multiple third party products

LIKE and Index Selectivity

```
SELECT
    application_id, first_name, last_name, …
FROM applications
WHERE last_name LIKE ‘S%’
    AND first_name LIKE ‘R%’
```

Will not use index due
to lack of selectivity

Require Enough Characters To Be Selective

- Require enough characters to be selective

```
SELECT
    application_id, first_name, last_name, …
FROM applications
WHERE last_name LIKE ‘Sch%’
    AND first_name LIKE ‘Ro%’
```

- Include full data for other index columns

```
SELECT
    application_id, first_name, last_name, …
FROM applications
WHERE last_name LIKE ‘B%’
    AND first_name LIKE ‘David%’
```

Are You Using a Function in Your Where Clause?

```
SELECT
    application_id, first_name, last_name, …
FROM Applications
WHERE UPPER(last_name) LIKE ‘NELSON%’
```

Function applied to column in WHERE clause
Oracle will not use standard indexes

Solutions for Functions in the WHERE Clause

Use a function based index

- Case insensitive searches
- Phonetic searches

Evaluate data modeling choices

- Trimming of string data
- Converting strings to dates
- Embedded strings within other fields

Implicit Data Type Cast

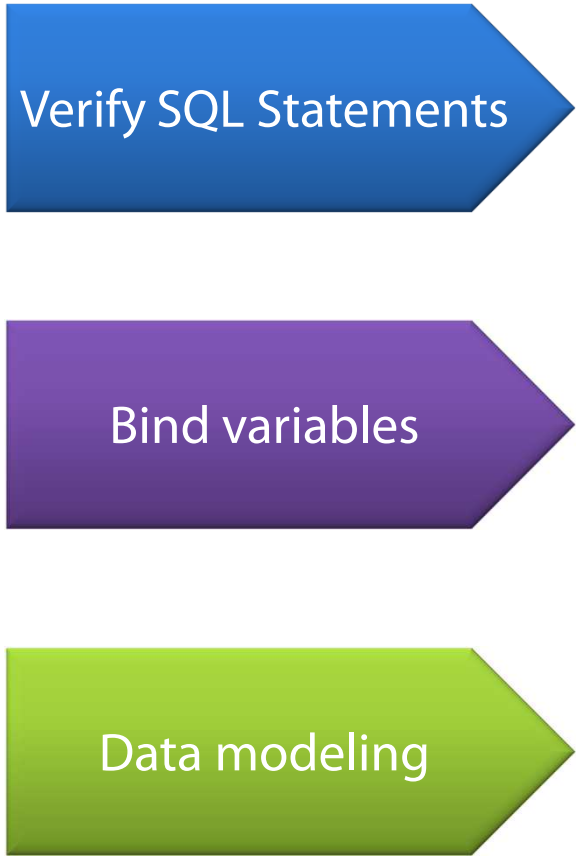
- Table definition

```
desc courses;
Name                 Null      Type
-------------------- --------  -------------
DEPARTMENT_CODE      NOT NULL  VARCHAR2(2)
COURSE_NUMBER        NOT NULL  VARCHAR2(3)
COURSE_TITLE         NOT NULL  VARCHAR2(64)
COURSE_DESCRIPTION NOT NULL  VARCHAR2(512)
CREDITS              NOT NULL  NUMBER(3,1)
```

- Query with mismatched data type

```
SELECT *
    FROM courses
    WHERE department_code = 'PH'
    AND course_number = 101
```

Data Types in SQL Statements

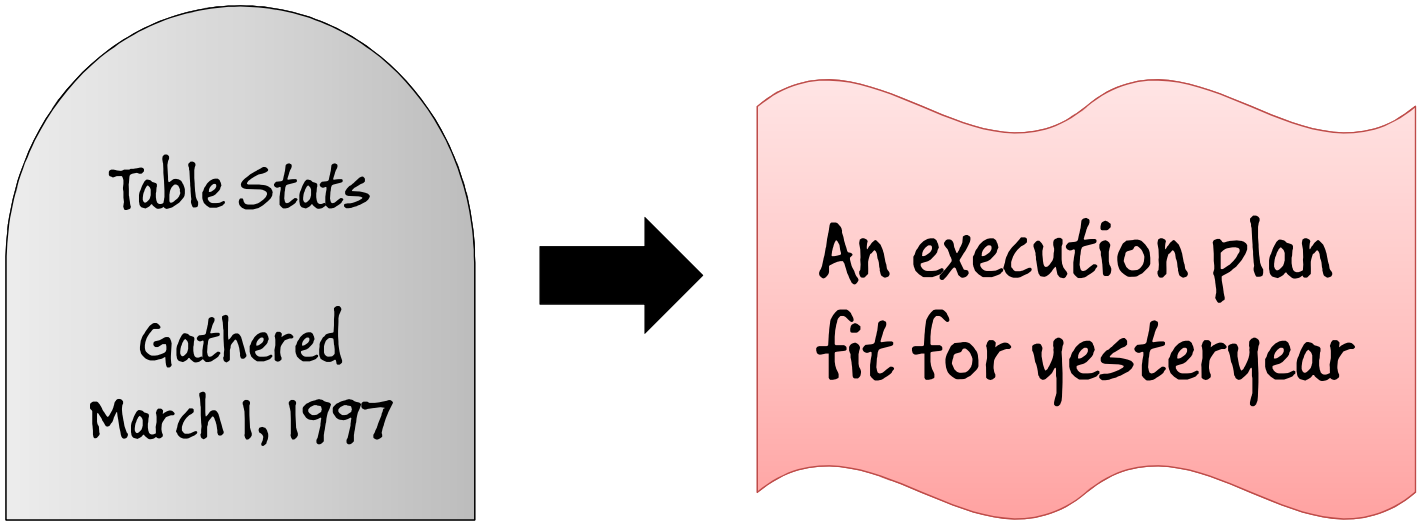


Make sure no implicit type conversions are occurring

Explicitly specify data types with bind variable definition

Analyze what data type a column should really use

Database Statistics Up to Date



When Are Stats Updated

Production Environments	Stats are usually automated to update nightly
Test Environments	Stats collection may not be automated
Times to Update Statistics	After a bulk load/deletion of data After creating a new index

Syntax to Update Database Statistics

```
-- Update for one table and all indexes
EXEC DBMS_STATS.GATHER_TABLE_STATS(
    ownname => '<<Table Owner Name>>',
    tabname => '<<Table Name>>',
    estimate_percent => <<percent>>);


-- Update for all objects in a schema
EXEC DBMS_STATS.GATHER_TABLE_STATS(
    ownname => '<<Table Owner Name>>');
```

Module Summary

What to Index

- Column with unique values
- Foreign key columns
- Columns used in WHERE clauses

Index Costs

- Indexes impact DML operations
- Modify or drop unused and sparingly used indexes

Why Isn't Oracle Using My Index

- Missing leading edge
- Lack of selectivity
- Outdated statistics

Monitoring Application Performance

David Berry
http://buildingbettersoftware.blogspot.com/

Need For Monitoring Applications

- Application performing as expected
- Identify additional tuning opportunities

- How does the app perform
- Correlate database data with app code

- Identify what SQL is currently running
- Find resource intensive statements

- Same queries can be used during testing
- Understand relative performance of SQL

Real Time Performance Data

Dynamic Performance Views

- **Also known as V$ views**
- **Always on, always being updated**
- **Reflect data currently in memory**

Data Dictionary Performance Views

- **Information about database objects (tables, indexes, etc.)**
- **Three sets**
  - dba_* - All database objects
  - all_* - All objects a user has permissions too
  - user_* - All objects a user owns

AWR Views

Licensing

- Use of Oracle AWR requires a Diagnostics Pack license

AWR Views

- All views starting with DBA_HIST
- V$ACTIVE_SESSION_HISTORY

Usage Monitoring

- Use of these views is monitored inside of Oracle
- You are liable for any unlicensed use of these views

Required Permissions

Security Concerns

- Ability to see sessions, SQL run by all users
- Access to wide range of database metadata
- Some organizations consider this a security risk

Knowledge Concerns

- Some DBA's concerned developers do not understand data
- Some DBA's see this data as DBA data, not developer data

Using a Built In Role For Permissions

Grant user access to the SELECT_CATALOG_ROLE role

Easiest and fastest route to gain access

Gives access to a wide range of database objects

Creating a Performance Viewer Role

```
CREATE VIEW perf_viewer;

GRANT SELECT ON V_$SESSION TO perf_viewer;
GRANT SELECT ON V_$SQL TO perf_viewer;
GRANT SELECT ON V_$SQLSTATS TO perf_viewer;
…

GRANT perf_viewer TO <<username>>;
```

Full script at http://bit.ly/1hUCpHz

Strategies For Obtaining Permissions

Explain what you are trying to accomplish

Demonstrate you are knowledgeable

Focus on mutual benefits

Introduction to Queries

- Keep in a text document for quick access
- Experiment to find what works best for your situation

Who is Logged in

```
SELECT
        username, osuser, program, module,
        machine, terminal, process,
        to_char(logon_time, 'YYYY-MM-DD HH24:MI:SS')
            AS logon_time,
        status,
        CASE status
            WHEN 'ACTIVE' THEN NULL
            ELSE last_call_et
        END as idle_time
    FROM v$session
    WHERE type = ‘USER’;
```

Where Are My Logins Coming From

```
SELECT
        username, osuser, program, module,
        machine, process,
        count(1) as login_count
    FROM v$session
    WHERE type = 'USER'
    GROUP BY
        username, osuser, program, module,
        machine, process;
```

What SQL is Executing Right Now

```
SELECT
        s.sid, s.username, s.osuser,
        s.machine, s.process, s.program, s.module,
        q.sql_text, q.optimizer_cost,
        s.blocking_session, bs.username as blocking_user,
        bs.machine as blocking_machine, bs.module as blocking_module,
        bq.sql_text AS blocking_sql, s.event AS wait_event,
        q.sql_fulltext
    FROM v$session s
    INNER JOIN v$sql q
        ON s.sql_id = q.sql_id
    LEFT OUTER JOIN v$session bs  -- blocking sessions
        ON s.blocking_session = bs.sid
    LEFT OUTER JOIN v$sql bq  -- blocking queries
        ON bs.sql_id = bq.sql_id
    WHERE s.type = 'USER‘;
```

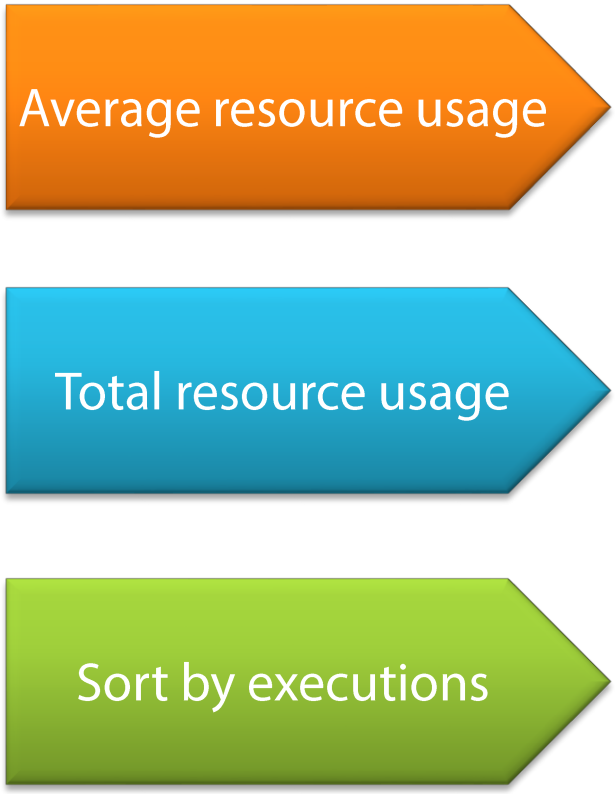
What Sessions Are Consuming Resources

```
SELECT
        s.sid, s.username, s.osuser,
        to_char(sm.begin_time, 'HH24:MI:ss') AS interval_start,
        to_char(sm.end_time, 'HH24:MI:ss') AS interval_end,
        s.machine, s.process, s.program, s.module,
        sm.cpu, sm.pga_memory, sm.logical_reads, sm.physical_reads,
        sm.hard_parses, sm.soft_parses,
        s.logon_time
    FROM v$session s
    INNER JOIN v$sessmetric sm
        ON sm.session_id = s.sid
    WHERE s.type = 'USER'
    ORDER BY sm.cpu DESC;
```

What Statements Consume the Most Resources

```
SELECT * FROM
(
    SELECT sql_id, sql_text, executions,
        elapsed_time, cpu_time, buffer_gets, disk_reads,
        elapsed_time / executions AS avg_elapsed_time,
        cpu_time / executions AS avg_cpu_time,
        buffer_gets / executions as avg_buffer_gets,
        disk_reads / executions as avg_disk_reads
    FROM v$sqlstats
    WHERE executions > 0
    ORDER BY elapsed_time / executions DESC
)
WHERE rownum <= 25;
```

Different Sort Criteria, Different View



- Most intensive statements per run
- Should be evaluated for performance

- Highest overall resource consumption
- Expensive statement executed many times?

- Frequently executing statements
- Could data be cached

SQL Statements And the Shared SQL Area

Shared SQL Area

- Source of data for V$Sqlstats, V$Sql views

Time in Cache

- Varies according to amount of memory, database workload
- Could range from several minutes to hours

Execute Multiple Times

- Different times of day will give different results
- Combine all executions for holistic picture

What Statements Are Performing Full Scans

```
SELECT
        pl.object_owner, pl.object_name,
        pl.sql_id, q.sql_text, q.module,
        pl.operation, pl.options, pl.cost, pl.cpu_cost, pl.io_cost,
        q.executions
    FROM v$sql_plan pl
    INNER JOIN v$sql q
        ON pl.sql_id = q.sql_id
    WHERE
        (pl.operation = 'TABLE ACCESS' AND pl.options = 'FULL')
        OR (pl.operation = 'INDEX' AND pl.options = 'FAST FULL SCAN')
        OR (pl.operation = 'INDEX' AND pl.options = 'FULL SCAN
   (MIN/MAX)')
        OR (pl.operation = 'INDEX' AND pl.options = 'FULL SCAN')
    ORDER BY pl.object_owner, pl.object_name;
```

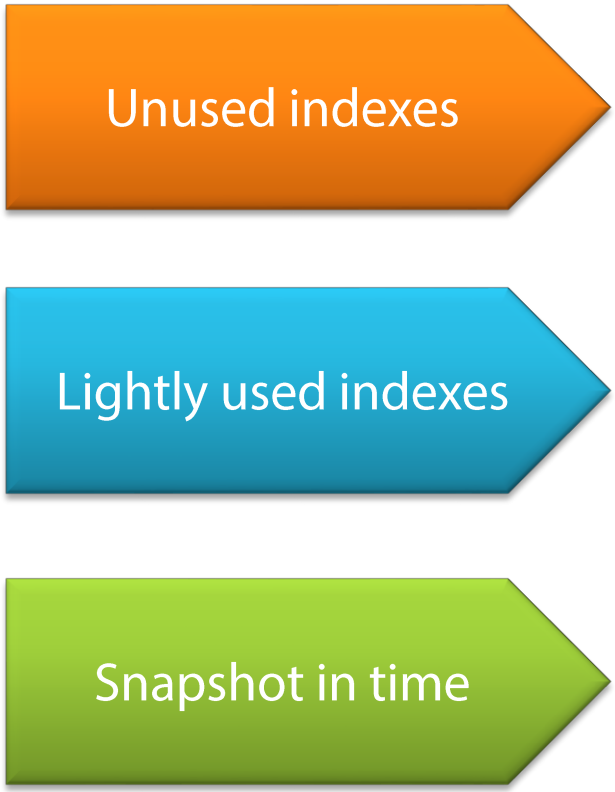
Displaying the Execution Plan of Cached SQL

```
SELECT plan_table_output
    FROM
    table(dbms_xplan.display_cursor('<<sql id>>',
        null,'typical'));
```

What Indexes Are Being Used

```
WITH index_usage AS
(
    SELECT pl.sql_id, pl.object_owner, pl.object_name, pl.operation,
            pl.options, count(1) as use_count
        FROM v$sql_plan pl
        WHERE pl.operation = 'INDEX'
        GROUP BY pl.sql_id, pl.object_owner, pl.object_name, pl.operation,
            pl.options
)
SELECT
        ix.table_owner, ix.table_name, ix.index_name, iu.operation,
        iu.options, ss.sql_text, ss.executions
    FROM all_indexes ix
    LEFT OUTER JOIN index_usage iu
             ON ix.owner = iu.object_owner
             AND ix.index_name = iu.object_name
    LEFT OUTER JOIN v$sqlstats ss
        ON iu.sql_id = ss.sql_id
    WHERE ix.owner = '<<owner name>>'
    ORDER BY ix.table_name, ix.index_name;
```

Index Usage Questions



- Why isn't this index being used
- Can this index be dropped

- Why is the index seldom used
- Can statements use another index

- Data drawn from shared SQL area
- Only reflects statements that have run recently

How Much Hard Parsing is Oracle Performing

```
SELECT sn.statistic#, sn.name, s.value
    FROM v$sysstat s
    INNER JOIN v$statname sn
        ON s.statistic# = sn.statistic#
    WHERE sn.name in (
        'parse time cpu',
        'parse count (hard)')
```

For all available statistics, query v$statname

Statements Not Using Bind Variables

```
WITH statements AS
(
    SELECT force_matching_signature,
        count(1) OVER (PARTITION BY force_matching_signature) AS statement_count,
        row_number() OVER (PARTITION BY force_matching_signature
            ORDER BY last_load_time DESC) AS row_index,
        parsing_schema_name,
        sql_text
    FROM v$sql
    WHERE force_matching_signature > 0
        AND force_matching_signature <> exact_matching_signature
)
SELECT
        force_matching_signature, parsing_schema_name,
        sql_text, statement_count
    FROM statements
    WHERE row_index = 1
        AND statement_count >= 5;
```

Table Information

```
SELECT
        t.owner, t.table_name, t.num_rows, t.avg_row_len,
        t.blocks AS blocks_below_hwm, t.empty_blocks,
        s.blocks AS segment_blocks,
        s.bytes / 1048576 AS size_in_mb,
        to_char(t.last_analyzed, 'YYYY-MM-DD HH24:MI')
            AS last_analyzed
    FROM all_tables t
    INNER JOIN dba_segments s
        ON t.owner = s.OWNER AND t.table_name = s.segment_name
    WHERE t.owner = '<<schema owner>>';
```

Column Information

```
SELECT column_name, avg_col_len,
        num_distinct, num_nulls,
    FROM all_tab_columns
    WHERE table_name = '<<table name>>'
    ORDER BY column_id DESC
```

Index Information

```
SELECT ix.owner, ix.index_name, ix.table_name,
        ix.distinct_keys, ix.leaf_blocks,
        s.blocks AS segment_blocks,
        s.bytes / 1048576 AS size_in_mb,
        to_char(ix.last_analyzed, 'YYYY-MM-DD HH24:MI')
            AS last_analyzed
    FROM all_indexes ix
    INNER JOIN dba_segments s
        ON ix.owner = s.OWNER
            AND ix.index_name = s.segment_name
    ORDER BY ix.table_name, ix.index_name
```

Summary

Monitoring Oracle Applications

- Queries useful for monitoring your application

Dynamic Performance Views

- Over 700 in Oracle 12c
- Full list at http://bit.ly/1mbIzYM

Try Out Your Own Variations

- Experiment with provided queries
- Web searches will locate many more variations

Practices and Pitfalls

David Berry
http://buildingbettersoftware.blogspot.com/

Module Outline

- Separate transactional and reporting functions
- Load only data that is needed
- Use transactions and commit correctly
- Avoiding issues with ORMs

One Database – Multiple Uses

Business Applications

Transactional Database

Reporting Applications

Different Use Cases

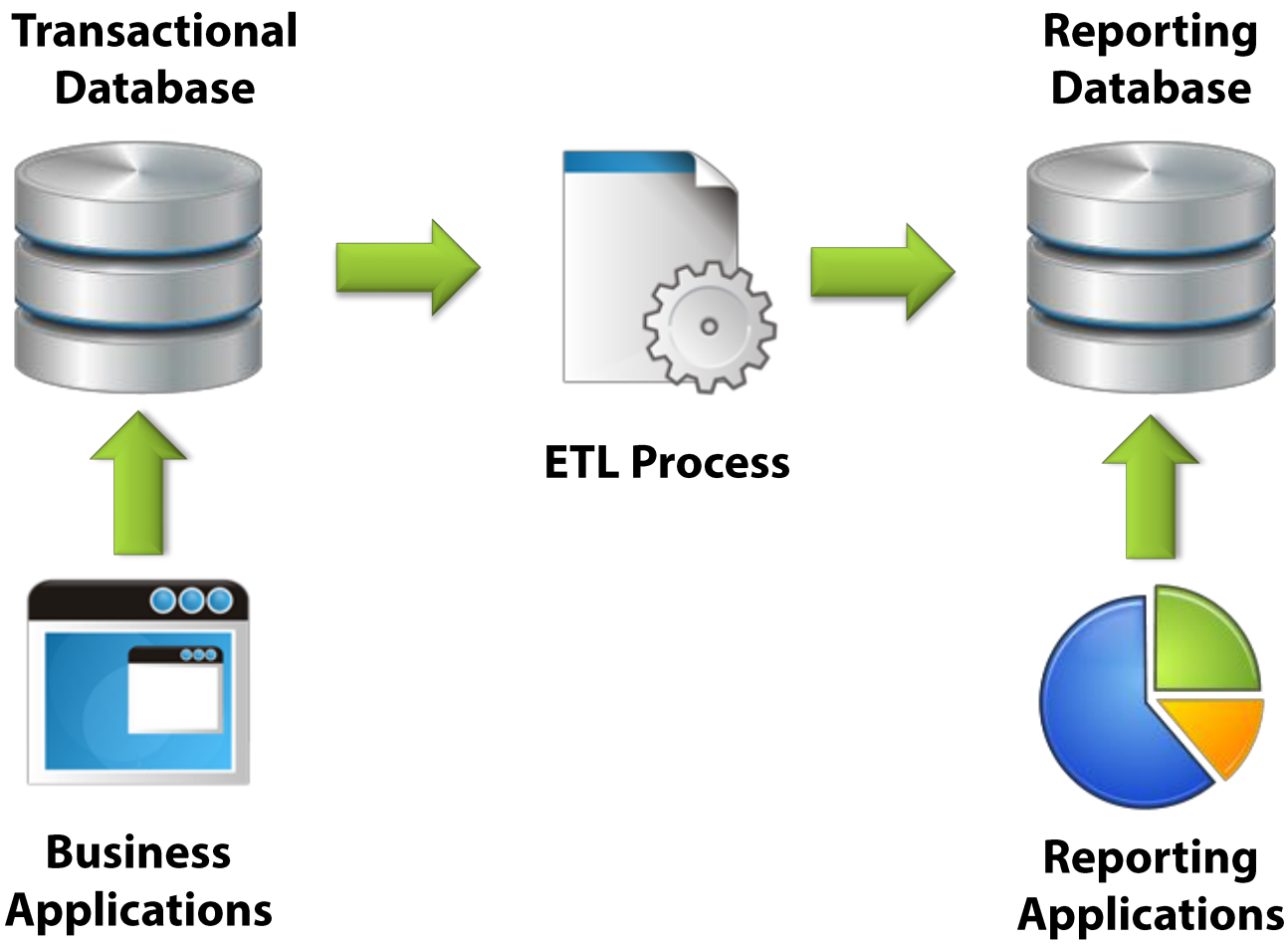
Business Applications

- High user count
- Quick, focused statements
- Reading and writing data
- Prefer normalized data

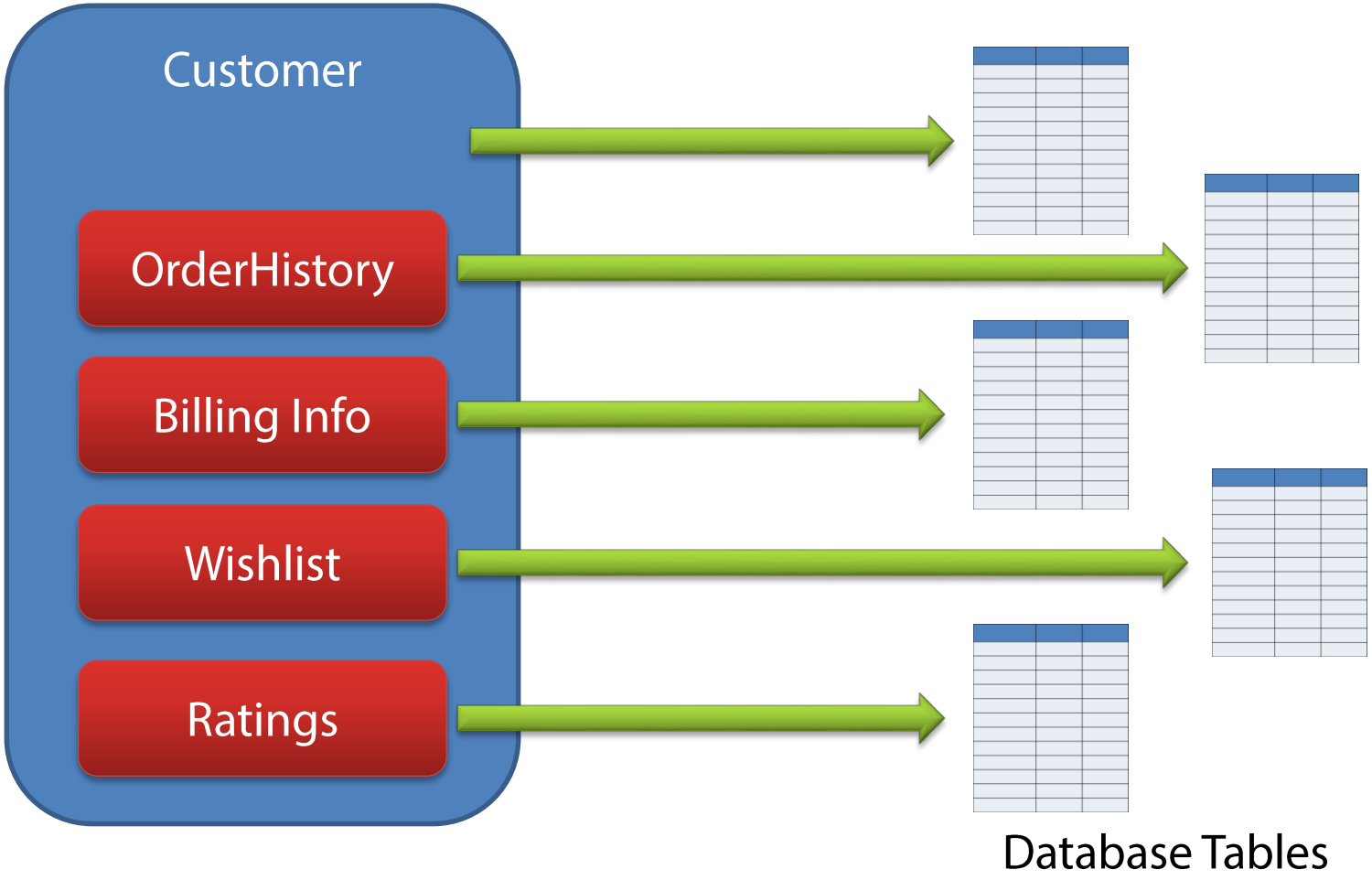
Reporting Applications

- Longer running queries
- Resource intensive queries
- Read only data
- Prefer denormalized data

Solution: Separate Databases



Load Only the Data You Need



What Data Does the User Need

Load my entire order history every time I log in?

Only load this data when I go to the order history page?

What Data Should You Load

- Focus on data needed for task
- Consider loading commonly used data

- Data is loaded only when needed
- Can appear seamless to the application

Overcommitting

```
INSERT INTO course_enrollments …;
commit;

INSERT INTO course_enrollments …;
commit;

INSERT INTO course_enrollments …;
commit;

INSERT INTO course_enrollments …;
commit;
```

Autocommit in ODP.NET, JDBC

- **Autocommit is on by default**
- **SQL Plus and SQL Developer**
  - Must commit transactions explicitly
- **Autocommit implications**
  - May be incorrect functional behavior
  - May impact performance of DML operations

Overcommitting

```
INSERT INTO course_enrollments …;
commit;

INSERT INTO course_enrollments …;
commit;

INSERT INTO course_enrollments …;
commit;
```

Overcommitting Summary

- **How often should I commit?**
  - Commit when your business transaction dictates
- **Proper transaction management can improve performance**
  - Fewer writes to redo logs
  - Write when the block completes, not each statement

Object Relational Mappers

Work at higher abstraction levels

Reduce boilerplate code

Increase productivity

Less visibility to database operations

Less control over data access paths

Using ORMs

Remember ORMs generate SQL

Rules about indexing still apply

ORMs and the n+1 Issue

- **Recognize lazy loading inside of loops**
  - Is this what you want?
- **Eager load data when appropriate**
  - One query can be more efficient than many small queries

Summary

- Divide transactional and reporting functions
- Understand transaction scope
- Understand how your ORM works

- Break larger processes into smaller ones
- Understand what each step is doing

Keep Learning

Oracle Documentation

- ODP.NET Developers Guide - http://bit.ly/1hp6fEa
- JDBC Developers Guide - http://bit.ly/1k7VJkl
- Performance Tuning Guide - http://bit.ly/1jTfAcU
- SQL Tuning Guide - http://bit.ly/1jTfAcU

Oracle Magazine

- Online at http://bit.ly/1pq6SlK