

E2201



IBM Systems Group

BP07:Tuning Web-based Data Access for DB2 UDB for iSeries

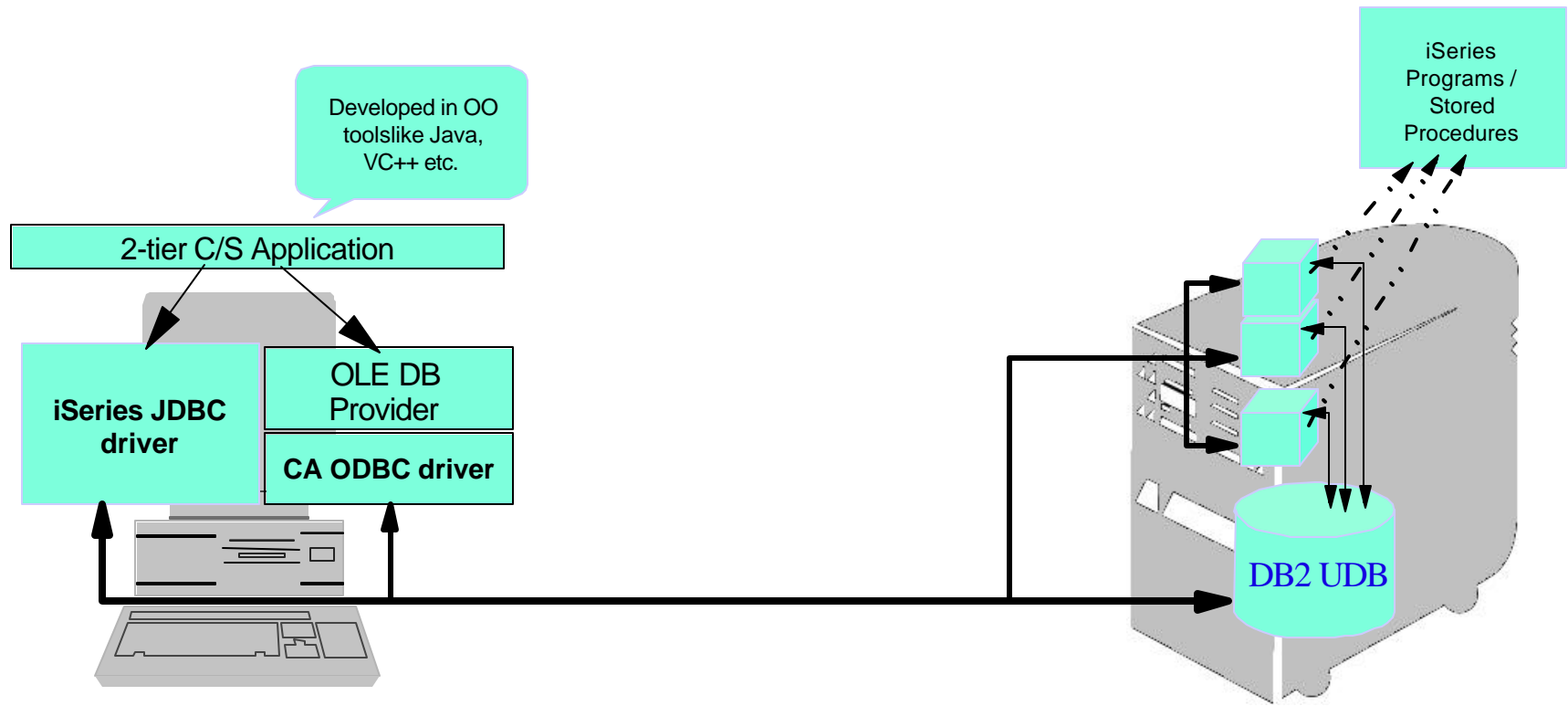
Jarek Miszczyk

PartnerWorld for Developers, eServer iSeries

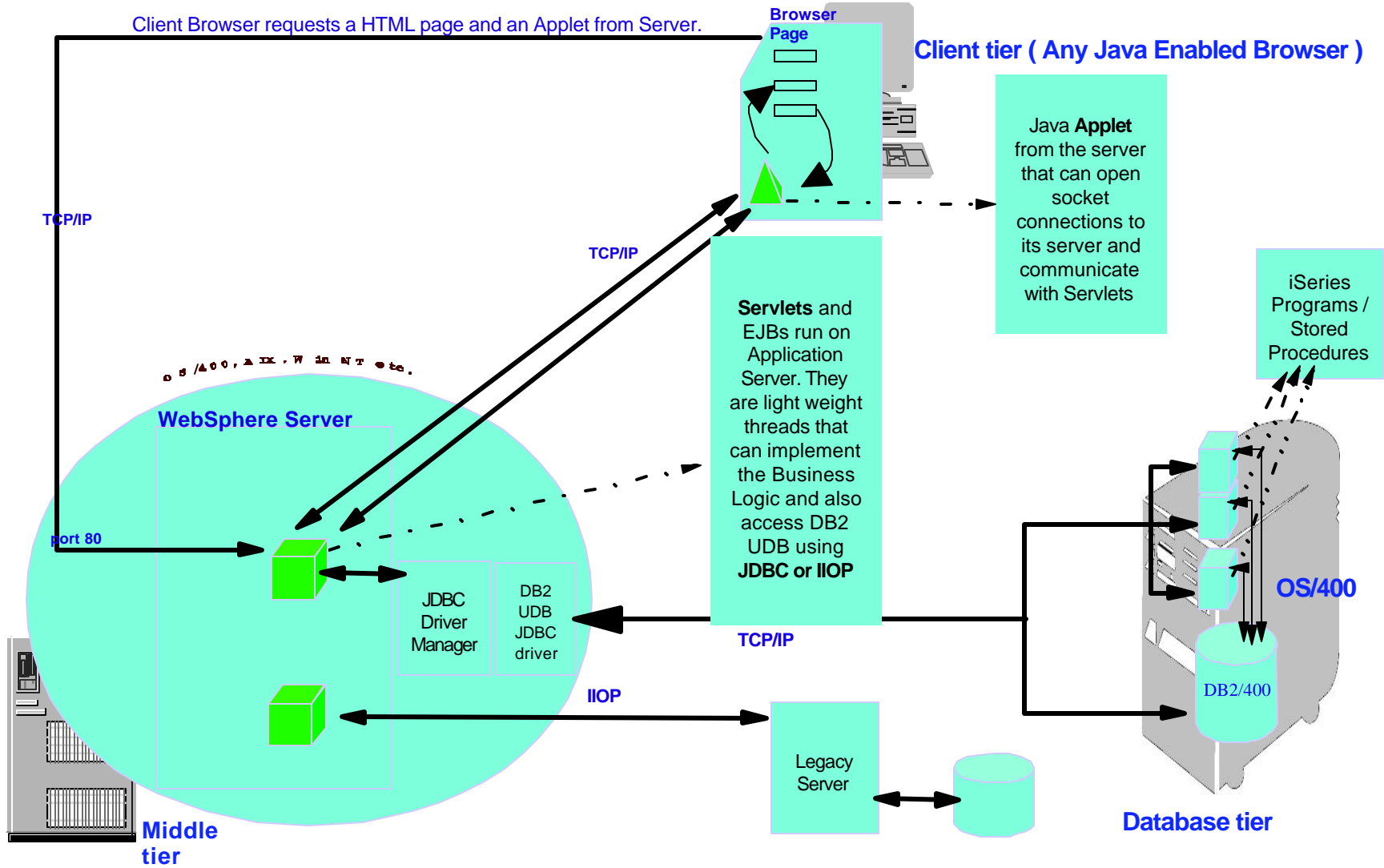
IBM eServer^J iSeries^J

ITSO iSeries Technical Forum

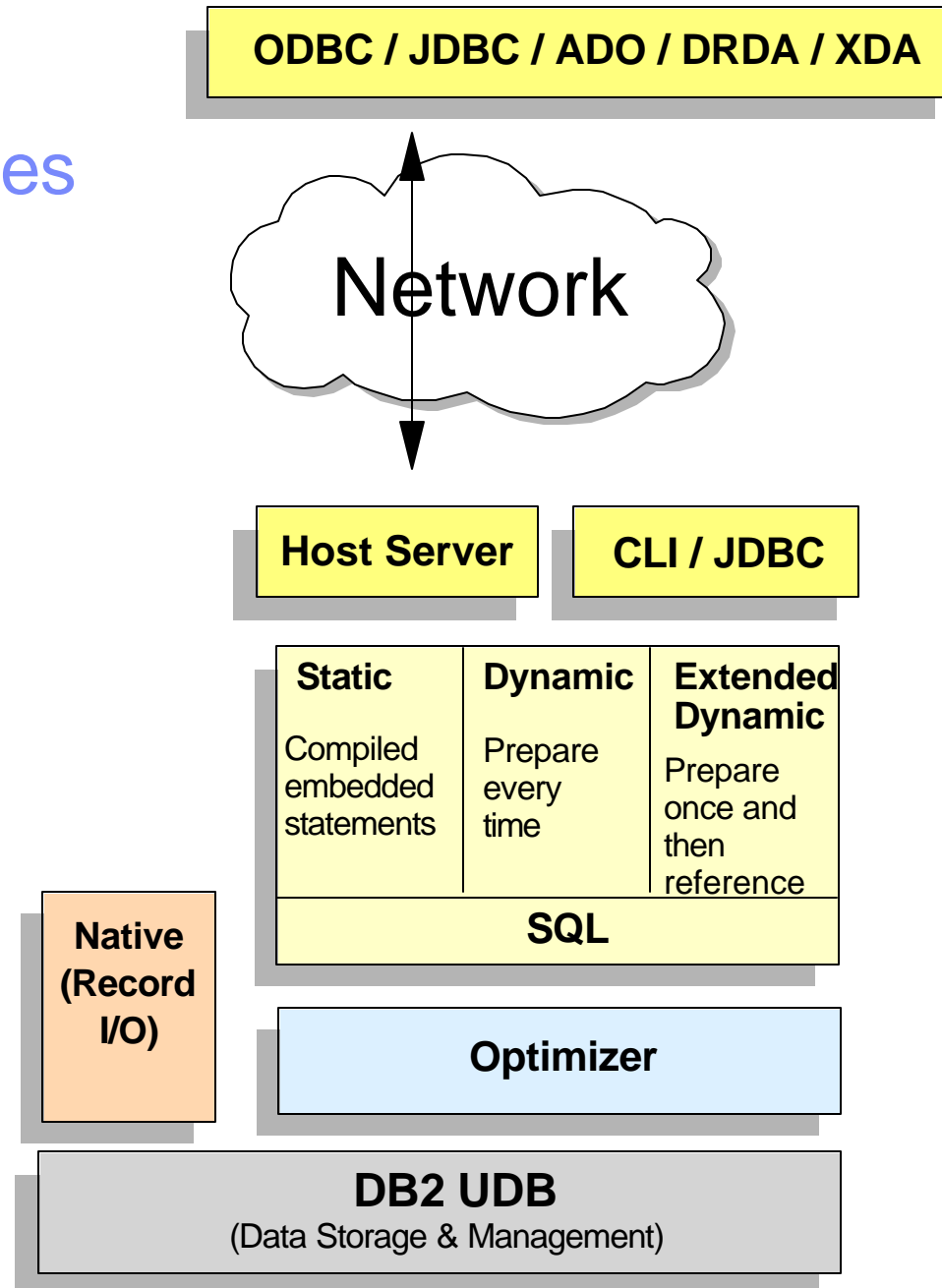
2-tier Application Architecture



3-tier application using Java



Series SQL Interfaces



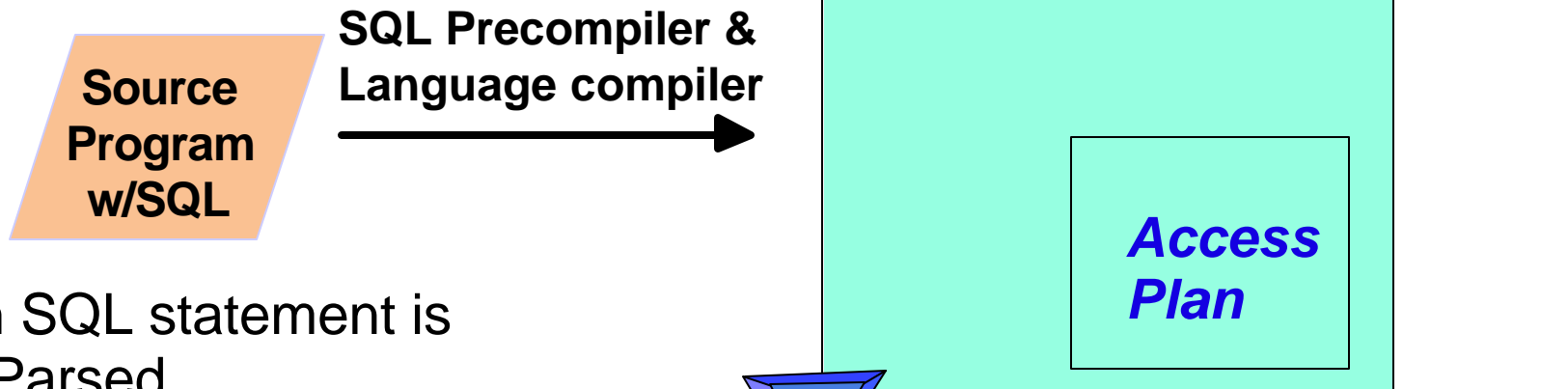
Static SQL

- Non-dynamic SQL statements that are 'hard-coded' in application programs
- Languages Supported:
 - ▶ SQL Stored Procedure Language for Procedures, Triggers & UDFs
 - ▶ RPG
 - ▶ COBOL
 - ▶ C, C++
 - ▶ PL/I
- Most efficient SQL interface on iSeries

Dynamic SQL

- SQL statements are dynamically created on the fly (Prepare/Execute or ExecuteImmediate) as part of the application logic
- Greater overhead since DB2 UDB does not know what SQL is being executed ahead of time
- DB2 UDB for iSeries Interfaces that utilize Dynamic SQL...
 - ▶ ODBC
 - ▶ CLI
 - ▶ JDBC/SQLJ
 - ▶ Net.Data
 - ▶ Interactive SQL (STRSQL)
 - ▶ RUNSQLSTM
 - ▶ Operations Navigator SQL requests
 - ▶ REXX
 - ▶ Query Manager

Access Plans - Static SQL View



Each SQL statement is

- Parsed
- Validated for syntax
- Optimized

as access plan created for the statement

Generic plan quickly generated first time
 Complete, optimized plan second time

Access Plans

Plan Contents

- A control structure that contains info on the actions necessary to satisfy each SQL request
- These contents include:
 - ▶ Access Method
 - Access path ITEM used for file 1.
 - Key row positioning used on file 1.
 - ▶ Info on associated tables and indexes
 - Used to determine if access plan needs to be rebuilt due to table changes or index changes
 - **EXAMPLE:** a column has been removed from a table since the last time the SQL request was executed
 - ▶ Any applicable program and/or environment info
 - **Examples:** Last time access plan rebuilt, DB2 SMP feature installed

Access Plans - Dynamic SQL View

**Dynamic
SQL
statement**

Each Dynamic SQL PREPARE is

- Parsed
- Validated for syntax
- Optimized

as access plan created
for the statement

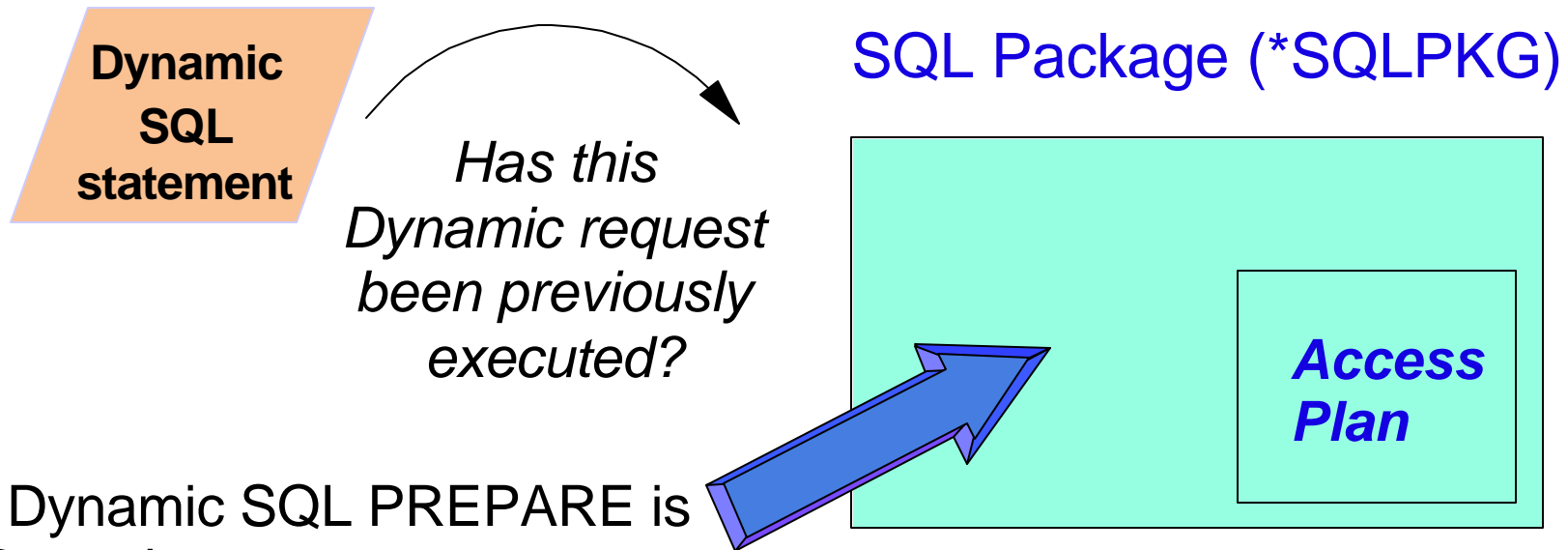
**Working Memory
for Job**

**Access
Plan**

Generic plan quickly generated on Prepare
Complete, optimized plan on Execute/Open

→ Less sharing & reuse of resources

Access Plans - Extended Dynamic View



Each Dynamic SQL PREPARE is

- Parsed
- Validated for syntax
- Optimized

as access plan created for the statement

Generic plan quickly generated on Prepare
 Complete, optimized plan on Execute/Open

OPENing the Access Plan

- Validate the Access Plan
- IF NOT Valid, THEN Reoptimize & update plan (late binding)
 - ▶ Some of the possible reasons:
 - Table size greatly increased
 - Index added/removed
 - Significant host variable value change
- Implement Access Plan: CREATE ODP (Open Data Path)

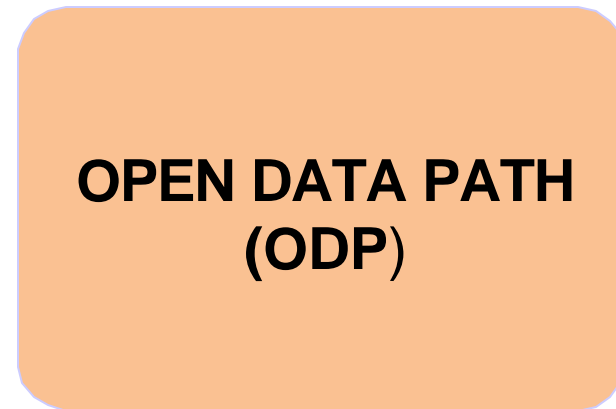
Access Plan to ODP

Internal Structures



CREATE

Executable code for all requested I/O operations



- Create process is **EXPENSIVE**
 - Longer execution time the first time an SQL statement is executed
- Emphasizes the need of **REUSABLE** ODPs
 - ▶ Reusable ODP 10 to 20 times less CPU resources

OPEN Optimization

- The request and environment determine if the OPEN requires an ODP Creation ("Full" Open)
 - ▶ DB2 UDB responsible for implementing Full vs Reusable Open
 - ▶ Reusable ODP usually happens after 2nd execution of statement within connection/job
- OPENs can occur on:
 - ▶ OPEN Statement
 - ▶ SELECT Into Statement
 - ▶ INSERT statement with a VALUES clause
 - ▶ INSERT statement with a SELECT (2 OPENs)
 - ▶ Searched UPDATE's
 - ▶ Searched DELETE's
 - ▶ Some SET statements in SQL procedure language
 - ▶ Certain subqueries may require one Open per subselect

Reusing the ODP steps

- IF First or Second Execution of Statement THEN...
- ELSE
 - IF Non-Reusable ODP THEN...
 - ELSE **Reusable ODP - Do Nothing**
- Run SQL request
- Delete ODP or Leave ODP open for Reuse?
 - ▶ ODP will not be deleted after second execution
- Loop back to Beginning

- Validate Access Plan
- IF NOT Valid, THEN Reoptimize & update plan (late binding)
- Create the ODP

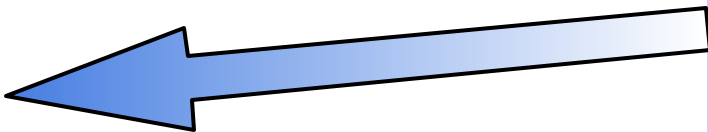
Reusing the ODP example

```
DECLARE c1  
  FOR SELECT empnumber, lastname  
  FROM employee  
  WHERE deptno = '503';
```

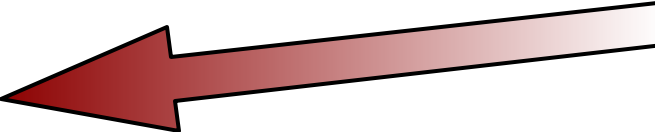
```
...  
OPEN c1;
```

```
WHILE more rows AND no error  
  FETCH c1 INTO :EmpNo, :EmpName  
END WHILE;
```

```
CLOSE c1;
```



ODP either created or reused depending on current mode



IF Reusable ODP,
THEN
 ODP is NOT deleted
 (pseudo-closed)
ELSE
 ODP is deleted

OPEN Optimization

Reusable ODP Example

```

INSERT INTO resultTable
  SELECT id, name
  FROM customers
  WHERE region = 'Central'
    
```

```

SQL7912 ODP created.
SQL7912 ODP created. ←
...
SQL7913 ODP deleted.
SQL7913 ODP deleted.
SQL7985 CALL statement complete
SQL7912 ODP created.
SQL7912 ODP created.
...
SQL7914 ODP not deleted.
SQL7914 ODP not deleted. ←
SQL7985 CALL statement complete
SQL7911 ODP reused.
SQL7911 ODP reused. ←
...
...
SQL7914 ODP not deleted.
SQL7914 ODP not deleted.
SQL7985 CALL statement complete
    
```


OPEN Optimization - Reuse Roadblocks

- Changed Table location - application using unqualified table names & the library list has changed since the ODP was opened (System naming mode - *SYS)
 - ▶ If table location is not changing, then default collection can be used to enable reuse
 - ▶ Default collection option/API exists to 'pin' table location
- Temporary index used by ODP
 - ▶ Temporary index does not always mean non-reusable ODP, optimizer does try to reuse temp index
 - If SQL run multiple times and index built each time, then creating a permanent index will probably make ODP reusable
 - If host variable value used to build selection into temporary index (ie, sparse), then ODP is not reusable because temporary index selection can be different on each execution of the query

OPEN Optimization - Reuse Considerations

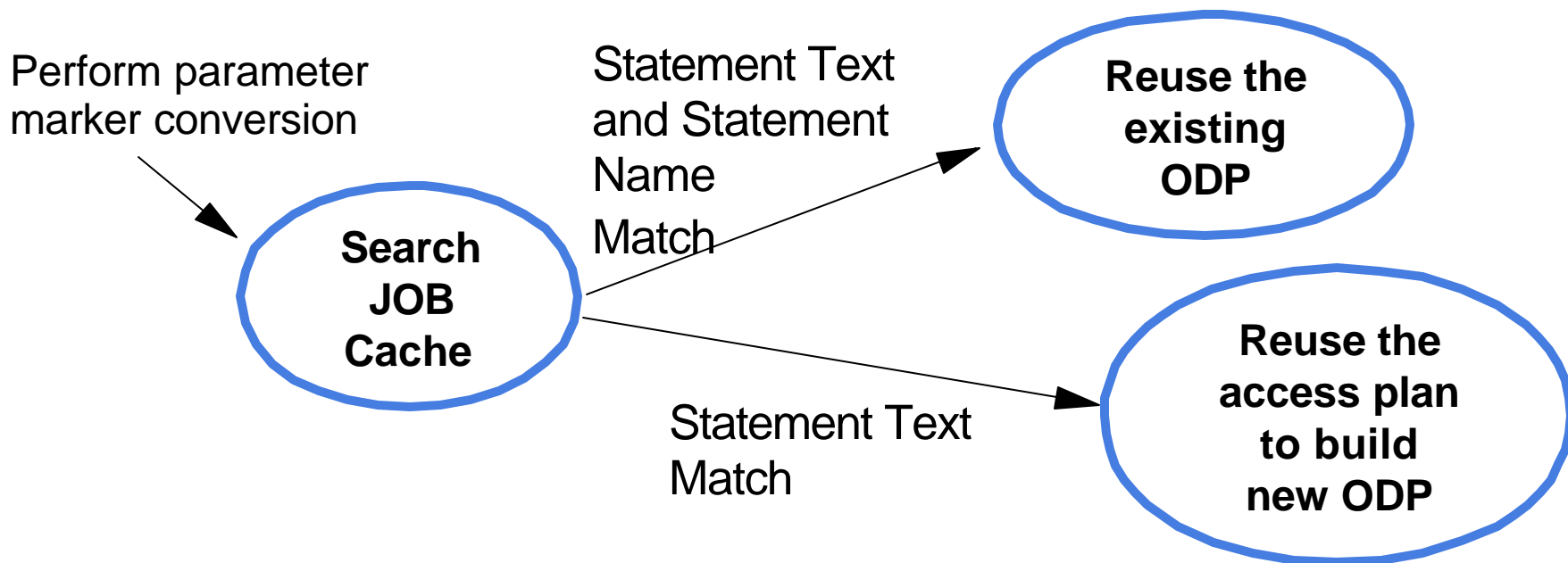
- Reusable ODP's do have one shortcoming... once reuse mode has started access plan is NOT rebuilt when the environment changes
 - ▶ What happens to performance if Reusable ODP is now run against a table that started out empty and has now grown 5X in size since the last execution?
 - ▶ What if selectivity of host variable or parameter marker greatly different on 5th execution of statement?
 - ▶ What if index added for tuning after 5th execution of statement in the job?
- Possible corrective actions:
 - ▶ DISCONNECT statement
 - ▶ CL command: *ALCOBJ <tablename> CONFLICT(*RQSRLS)*
 - ▶ Reclaim request on Application exit

Dynamic SQL Tuning

- With Dynamic interfaces, full opens are avoided by using a "PREPARE once, EXECUTE many" mentality when an SQL statement is going to be executed more than once
- A PREPARE does NOT automatically create a new statement and full open on each execution
 - ▶ DB2 UDB performs caching on Dynamic SQL PREPAREs within a job/connection....

Dynamic SQL Tuning - Job Cache

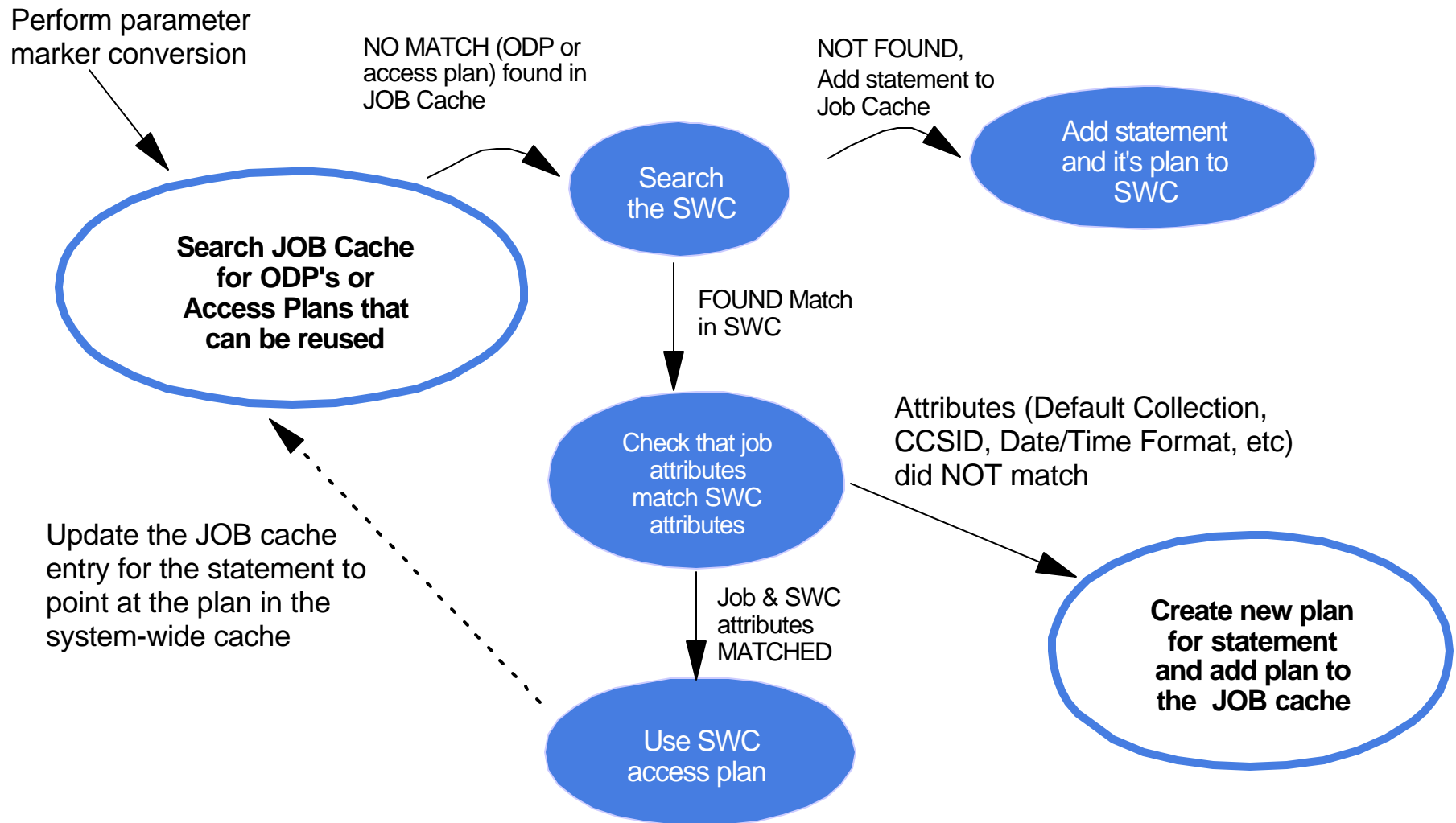
- Parse the statement and perform parameter marker conversion
- Compare with previously prepared statements in the job
 - ▶ If Statement Text AND Statement Name match, then ODP will be reused
 - ▶ If only Statement Text matches, then the access plan is reused from the previously prepared statement



Dynamic SQL Tuning - System Cache

- DB2 UDB for iSeries also caches access plans for Dynamic SQL requests in the SystemWide Statement Cache (SWC)
 - ▶ Only access plans are reused (No ODP reuse)
- SWC requires no administration
 - ▶ Cache storage allocation & management handled by DB2 UDB
 - ▶ Cache is created from scratch each IPL
 - ▶ Cache churn and contention avoided by allowing limited access plan updates
 - In some cases, optimizer will build a temporary access plan to use instead of the cached access plan
 - Might think about system IPL after your database is tuned
 - ▶ Cache contents cannot be viewed, max of 165,000+ statements
- SWC cache does interact with the job cache

Dynamic SQL Tuning - System Cache



Dynamic SQL Tuning - Parameter Markers

- Parameter Markers are one implementation method for "EXECUTE many"
 - ▶ Improves chance for reusable ODPs
 - ▶ Example: want to run the same SELECT statement several times using different values for customer state
 - 50 different statements/opens for each of the states OR...
 - *Single SQL statement that allows you to plug in the needed state value*
 - ▶ DB2 UDB does try to automate parameter marker usage...

Dynamic SQL Tuning - Parameter Markers

- DB2 UDB automatically tries to convert literals into parameter markers to make statement more reusable

**SELECT name, address FROM customers
WHERE orderamount > 1000.00 AND state = 'NY'**

CONVERTED
TO:

**SELECT name, address FROM customers
WHERE orderamount > ? AND state = ?**

**UPDATE customers SET status = 'A'
WHERE orderamount >= 10000**

CONVERTED
TO:

**UPDATE customers SET status = ?
WHERE orderamount >= ?**

- There are cases where the auto-conversion cannot be used:
 - ▶ Mix of parameter literals and parameter markers
 - ▶ Expressions used in SET or SELECT clause

Extended Dynamic SQL & Packages

- Extended Dynamic similar to Dynamic SQL, except that Package is searched to see if there is a statement with the same SQL and attributes
 - ▶ If a match is found, then a new statement entry name is allocated with a pointer to the existing access plan
- Advantages of using Extended Dynamic SQL Packages:
 - ▶ Shared resource available to all users
 - Access information is reused instead of every job and every user "re-learning" the SQL statement
 - ▶ Permanent object that saves information across job termination and system termination
 - Can even be saved & restored to other systems
 - ▶ Improved performance decisions since statistical information is accumulated for each SQL statement

Extended Dynamic & Packages

The Interfaces

- System API - QSQPRCED
 - ▶ API user responsible for creating package
 - ▶ API user responsible for preparing and describing statement into package
 - ▶ API user responsible for checking existence of statement and executing statements in the package
- XDA API set
 - ▶ Abstraction layer built on top of QSQPRCED for local and remote access
- Extended dynamic setting/configuration for IBM Client Access ODBC driver & iSeries Java Toolkit JDBC driver
 - ▶ Drivers handle package creation
 - ▶ Drivers automate the process of adding statements into the package
 - ▶ Drivers automate process of checking for existing statement and executing statements in the package

Extended Dynamic & Packages

Considerations

- Any SQL statement that can be prepared is eligible
 - ▶ ODBC & JDBC drivers have further restrictions

- Size limitations
 - ▶ Current size limit is 500 MB, about 16K statements
 - ▶ Package can grow without new statements being added. Access plan rebuilds require additional storage
 - ▶ Background package compression enabled with latest V4R5 Database Group PTF to increase life and usefulness of package objects

Design and Coding Tips

- Design and code to reduce network traffic
 - ▶ Use Blocked Inserts and Fetches
 - ▶ Utilize Stored Procedures
- Examine VARCHAR & LOB usage
- Leverage the interface (ODBC, JDBC, etc) based on DB2 UDB for iSeries SQL engine
- Tune your database by supplying the right indexes for the optimizer
 - ▶ Indexes used for more than just scans, iSeries optimizer also depends on them for their statistics

VARCHAR & LOB Considerations

- Variable length columns (VARCHAR/VARGRAPHIC)
 - ▶ If primary goal is space saving, include ALLOCATE(0) with VARCHAR definition
 - ▶ If primary goal is performance, ALLOCATE value should be wide enough to accommodate 90-95% of the values that will be assigned to the varying length column - this minimizes I/O
 - ▶ Due to stats limitations, not a good idea to use VARCHAR/VARGRAPHIC for primary or foreign key columns
- LOB & VARCHARs
 - ▶ Storage for LOB columns allocated in the same manner as VARCHAR columns
 - ▶ When a column stored in the overflow storage area is referenced, currently **ALL** of the columns in that area are paged into memory
 - A reference to a "smaller" VARCHAR column could potentially force extra paging of LOB columns
 - Example: VARCHAR(256) column retrieved by application has side-effect of paging in two 5 MB BLOB columns in the same row

Common iSeries SQL application interfaces

	CLIENT-based	SERVER-based
ODBC	Client Access ODBC Driver	DB2 UDB for iSeries Call Level Interface (CLI)
JDBC/SQLJ	iSeries Java Toolbox JDBC Driver	iSeries Native JDBC driver

ODBC Access Tips

- Use parameter markers & "Prepare Once - Execute Many"
 - ▶ SQLExecDirect not a good choice for a frequently executed statement
- Reduce the number of trips to the server with blocking, stored procedures, and result sets
- Code directly to the APIs (avoid MS Jet Engine)
- General ODBC tips & techniques
 - ▶ <http://www.iseries.ibm.com/developer/client/index.html>

ODBC Access Tips - Extended Dynamic

- Use the extended dynamic feature of Client Access ODBC driver (also available to iSeries Toolbox JDBC driver)
 - ▶ Feature externalized as data source option
 - ▶ Allows for private or shared packages
 - ▶ When used in conjunction with "Cache Package Locally" line trips can be reduced by up to 1/3
 - At connection time, portions (statement text, result & parameter marker descriptions) of the extended dynamic package are downloaded
 - Many of the ODBC "setup" APIs can just use the cached descriptive info on the client

ODBC Access Tips - Extended Dynamic

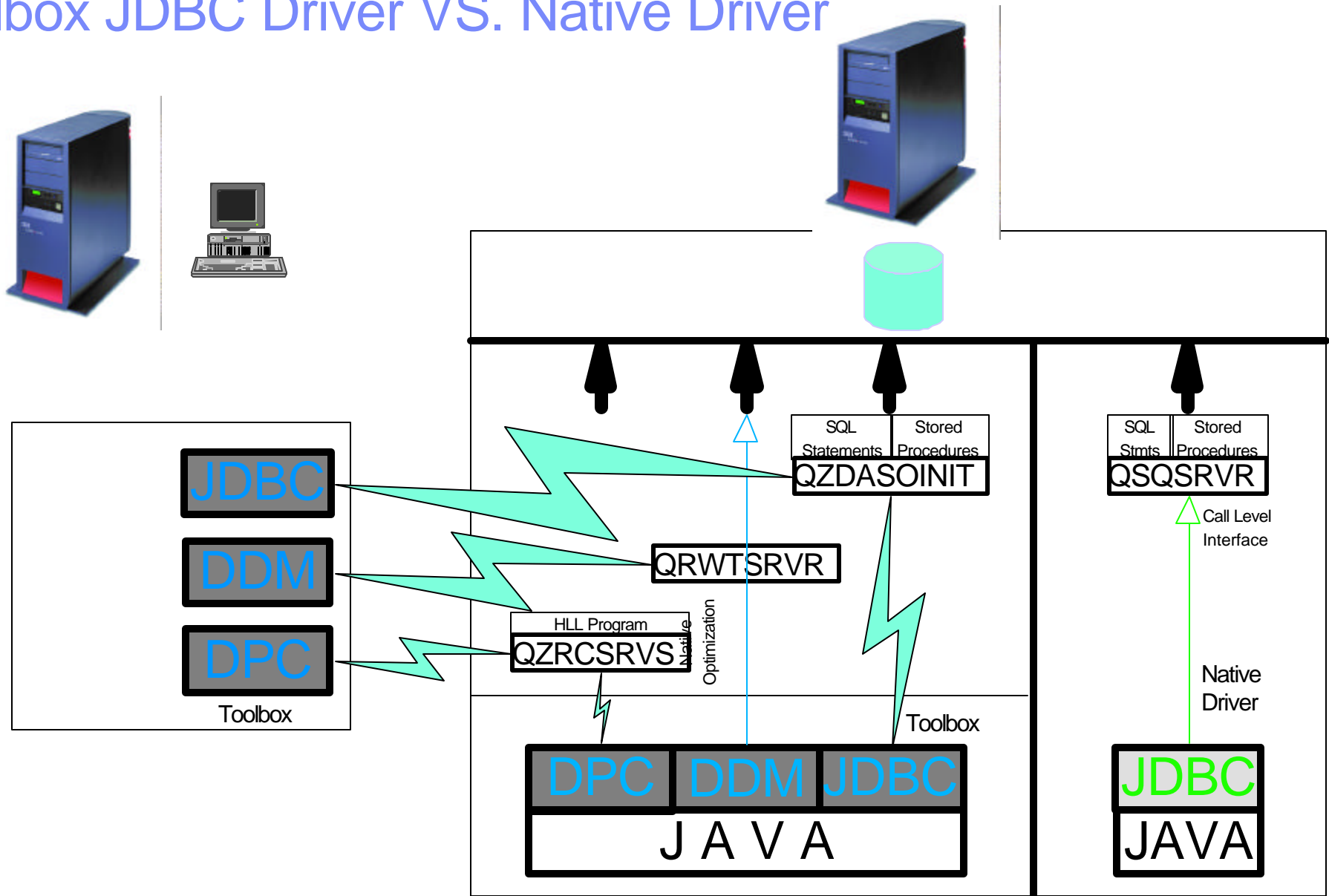
- Only the following SQL statements are eligible for extended dynamic packages
 - ▶ Statements that contain parameter markers
 - ▶ INSERT with subselect
 - ▶ Positioned UPDATE or DELETE
 - ▶ SELECT FOR UPDATE
 - ▶ Can force all SELECT statements into package by "hotwiring" data source config with Debug=5

- Packages cannot be used if application attributes do not match
 - ▶ Different CCSID, Date & time format attributes, decimal delimiter, default collection, etc
 - ▶ If package unusable, new requests are executed as "pure" Dynamic SQL

CLI Access Tips

- Allocate your CLI environment at the very beginning of processing and avoid freeing it unless the program/job ends
 - ▶ System resources will be unnecessarily wasted by setting up and freeing the environment over and over again
 - ▶ No drawbacks in just leaving CLI environment allocated
 - ▶ Deallocating environment will delete all ODP's
- Extended dynamic SQL NOT available for this interface
- PREPARE Once - Execute Many
- Avoid using unbound columns, use bound columns instead
 - ▶ Find any places in your code where SQLGetData or SQLGetCol is used after the SQLFetch, and replace this with SQLBindCol before the SQLFetch
 - ▶ Unbound columns cause data to be moved multiple times
- SQLFetch can only return 1 row at a time, use SQLExtendedFetch for blocked reads

Toolbox JDBC Driver VS. Native Driver



JDBC Access Tips

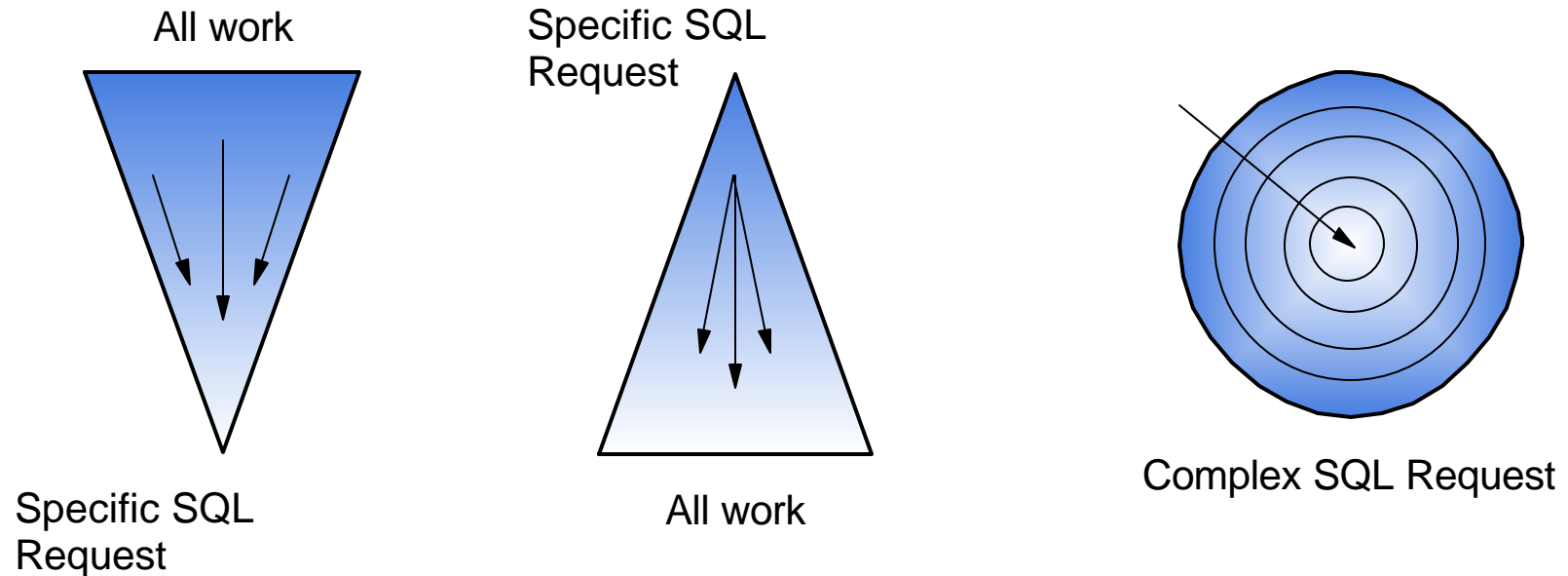
- Use preparedstatement class instead of statement class
- iSeries Java Toolkit JDBC Driver (distributed clients)
 - ▶ Do not connect/work/disconnect, maintain a connection.
 - ▶ Make sure the close() method is called before letting a statement handle go away for garbage collection
 - Create/Exec/Close instead of Create/Exec
 - ▶ get(col#) is currently about twice as efficient as get(ColumnName)
 - ▶ "extended dynamic" connection property
- Native JDBC driver
 - ▶ Most CLI considerations and restrictions apply
 - ▶ Performance can be improved by storing string data in Unicode format
 - ◆ ALTER TABLE mytable ALTER COLUMN mycol
SET DATA TYPE GRAPHIC (10) CCSID 13488
- SQLJ implementation provides performance similar to static and extended dynamic

Tools and Methodologies

Overview

- How do I know what's going on with my queries?
- How can I tell what the optimizer is doing?
- Answer: Tools and analysis
 - ▶ Query optimizer debug messages
 - ▶ Print SQL Information (PRTSQLINF)
 - ▶ Database Monitor Statistics
 - Detailed Monitor (STRDBMON)
 - Summary (Memory-based) Monitor (Operations Navigator)
 - ▶ Visual Explain
 - ▶ Change Query Attributes (CHGQRYA)
 - ▶ QAQQINI file attributes

Discovery Methodologies



- Proactive and reactive approaches
- All methodologies are iterative in nature

Debug Messages

- Informational messages written to the joblog about the implementation of a query
- Describes query implementation method
 - ▶ Indexes
 - ▶ Join order
 - ▶ Access plans
 - ▶ ODPs (Open Data Paths)
- Messages explain what happened during query optimization
 - ▶ Why index was or was not used
 - ▶ Why a temporary index result was required
 - ▶ Index advised by the optimizer
- **STRDBG UPDPROD(*YES) & STRSRVJOB** and **STRDBG** for batch jobs
- ODBC & JDBC Driver Exit program
- **MESSAGES_DEBUG = *YES** in **QAQQINI** file

Print SQL Information

PRTSQLINF OBJ(MY_PGM) OBJTYPE(*PGM)

PRTSQLINF OBJ(MY_PKG) OBJTYPE(*SQLPKG)

- OS/400 command that lists SQL information contained in a program, SQL package, or service program
- Creates a spooled file that contains:
 - ▶ SQL statements
 - ▶ Type of access plan used by each statement
 - ▶ Command (CRTSQLxxx) and parameters used to invoke the SQL precompiler
- iSeries version of SQL EXPLAIN utility
- Output similar to debug messages

Database Monitors

- Integrated tools used to gather database performance related statistics for SQL-based requests
- Monitor data dumped into table(s) where it can be queried to help identify and tune performance problem areas
 - ▶ Detailed monitor writes all of the information out to a single table as it's collected
 - Interface: STRDBMON & Operations Navigator
 - ▶ Summary monitor collects similar information at a summarized level in memory and then dumps the data into multiple tables
 - Interface: Operations Navigator & APIs

Database Monitors

- Provides all information from STRDBG or PRTSQLINF plus additional details:
 - ▶ SQL statement text
 - ▶ Start and end timestamp
 - ▶ Estimated processing time
 - ▶ Total rows in table queried
 - ▶ Number of rows selected
 - ▶ Estimated number of rows selected
 - ▶ Key fields for advised index...

- Once the data is collected, analysis is performed by running queries against the tables
 - ▶ EXAMPLES:
 - Which statements are the most time consuming?
 - Which statements are not having ODPs reused?
 - Which statements are causing temporary index builds?

- Operations Navigators provides some basic analysis reports

Predictive Query Governor

- Allows user to stop long-running queries before they even start. The query time limit is set on a per-job basis via CHGQRYA CL command
 - ▶ Can also be set via the system value QQRYSITMLMT or a QAQQINI option
- Query time limit is checked against estimated elapsed query time before initiating a query
 - ▶ Cost based optimization = costs and access plan are determined prior to execution
- An inquiry message is displayed to the end user showing the predicted runtime and asking if the query should be cancelled
- Debug messages will be written to the joblog if the query is canceled.
- Time limit of zero is used to optimize performance on queries without having to run through several iterations.

Query Performance Tuner - QAQQINI

- Provides central point of control for all attributes, options, and knobs that can impact query optimization

- Table design allows attributes to be set dynamically with just database updates or insert/delete

```
UPDATE mylib/QAQQINI SET QQVAL='600'
WHERE QQPARM='QUERY_TIME_LIMIT';
```

```
INSERT mylib/QAQQINI
VALUES('MESSAGES_DEBUG','*YES','Activated - 4pm');
```

- One row per attribute/parm and 3 character columns
 - QQPARM - the attribute/option name
 - QQVAL - value of the attribute/option
 - QQTEXT - optional description of the attribute or it values

QQPARM	QQVAL	QQTEXT
MESSAGES_DEBUG	*YES	Debug Set - 11pm
QUERY_TIME_LIMIT	600	New time limit - set 7/25
PARALLEL_DEGREE	*DEFAULT	
FORCE_JOIN_ORDER	*DEFAULT	
...	...	

Visual Explain

- Visualization of the query access plan
 - ▶ Details and attributes of the query plan, execution, and database objects involved
 - ▶ V5R1 includes auto-highlighting of icons
- Visual Explain can be used in one of two ways
 - ▶ Interactively with Ops Navigator SQL Script window
 - ▶ Reactively based on previously collected database monitor data (detailed monitor)
- Requires V4R5 or higher of OS/400 and IBM Client Access Operations Navigator

Tuning Tools Comparison

PRTSQLINF	STRDBG/CHGQRYA QAQQINI	STRDBMON	Memory -based Monitor
Available without running query (after access plan has been created)	Only available when the query is run	Only available when the query is run	Only available when the query is run
Displayed for all queries in SQL pgm or pkg, whether executed or not	Displayed only for those queries which are executed	Displayed only for those queries which are executed	Displayed only for those queries which are executed
Information on host variable implementation	Limited information on the implementation of host variables	All information on host variables, implementation, and values	All information on host variables, implementation and values
Available only to SQL users with pgms, packages, or service pgms	Available to all query users (OPNQRYF, SQL, QUERY/400)	Available to all query users (OPNQRYF, SQL, QUERY/400)	Available only to SQL interfaces
Messages printed to spool file	Messages displayed in job log	Performance records written to database file	Performance information collected in memory and then written to database file
Easier to tie messages to query with subqueries or unions	Difficult to tie messages to query with subqueries or unions	Uniquely identifies every query	Repeated query requests are summarized

Additional Resources

- DB2 UDB for iSeries home page: www.iseries.ibm.com/db2
- [iSeries SQL Performance Workshop \(Course #S6140\)](http://www-1.ibm.com/servers/eserver/series/service/igs/db2performance.html)
<http://www-1.ibm.com/servers/eserver/series/service/igs/db2performance.html>
- Online DB2 UDB publications www.iseries.ibm.com/db2/books.htm
 - ▶ Database Performance & Query Optimization
- SQL Interface FAQs:
 - ▶ CLI : www.iseries.ibm.com/db2/clifaq.htm
 - ▶ JDBC
 - Toolbox: www.iseries.ibm.com/toolbox/faqjdbc.htm
 - Native: www.iseries.ibm.com/developer/jdbc/index.html
- QAQQINI script builder:
www.iseries.ibm.com/developer/bi/tuner.html
- DB2 UDB for iSeries Online Education
www.iseries.ibm.com/developer/education/ibo/view.html?biz
- Third-party performance tools:
 - ▶ Centerfield Technology (www.centerfieldtechnology.com)