Platform: z/OS and OS/390

z/OS and OS/390 Application Development

DB2, Java and Design for High Performance

John J. Campbell
Senior Consultant IT Specialist
DB2 for z/OS and OS/390 Development
IBM Silicon Valley Lab

Session: D13

Thursday 16th May 2002 at 12:30am

Agenda

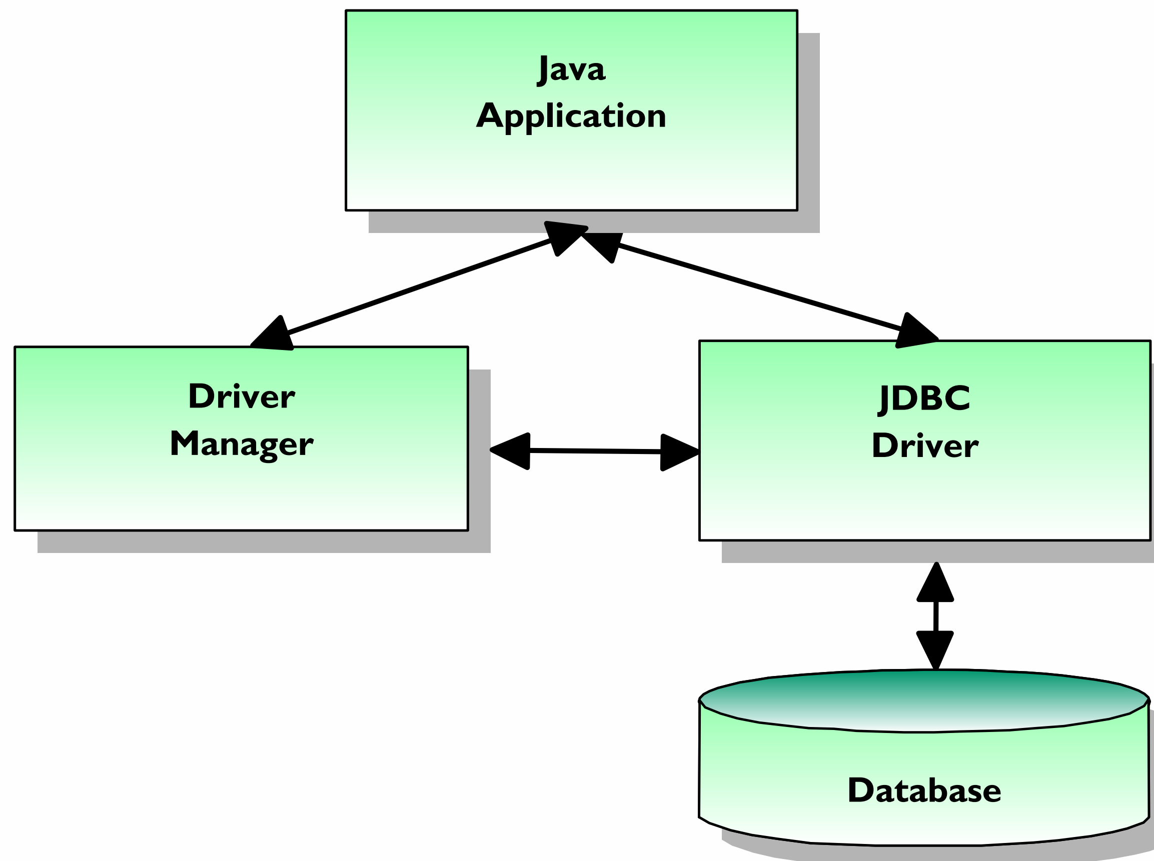
- **Overview about JDBC**
- **SQLJ versus JDBC**
- **Design Guidelines for Application**
- **Environment Tuning Hints**
- **Performance Measurements**

Disclaimer

- The information contained in this document has not been submitted to any formal IBM review and is distributed on an "As Is" basis without any warranty either express or implied. The use of this information is a customer responsibility.
- The measurement results presented here were run in a controlled laboratory environment using specific workloads. While the information here has been reviewed by IBM personnel for accuracy, there is no guarantee that the same or similar results will be obtained elsewhere. Performance results depend upon the workload and the environment. Customers attempting to adapt this data to their own environments do so at their own risk.
- In addition, the materials in this document may be subject to enhancements or Programming Temporary Fixes (PTFs) subsequent to the level used in this study.

Overview about JDBC

- A standard Java API for executing SQL statements
- JDBC API offers portability across platforms and database systems

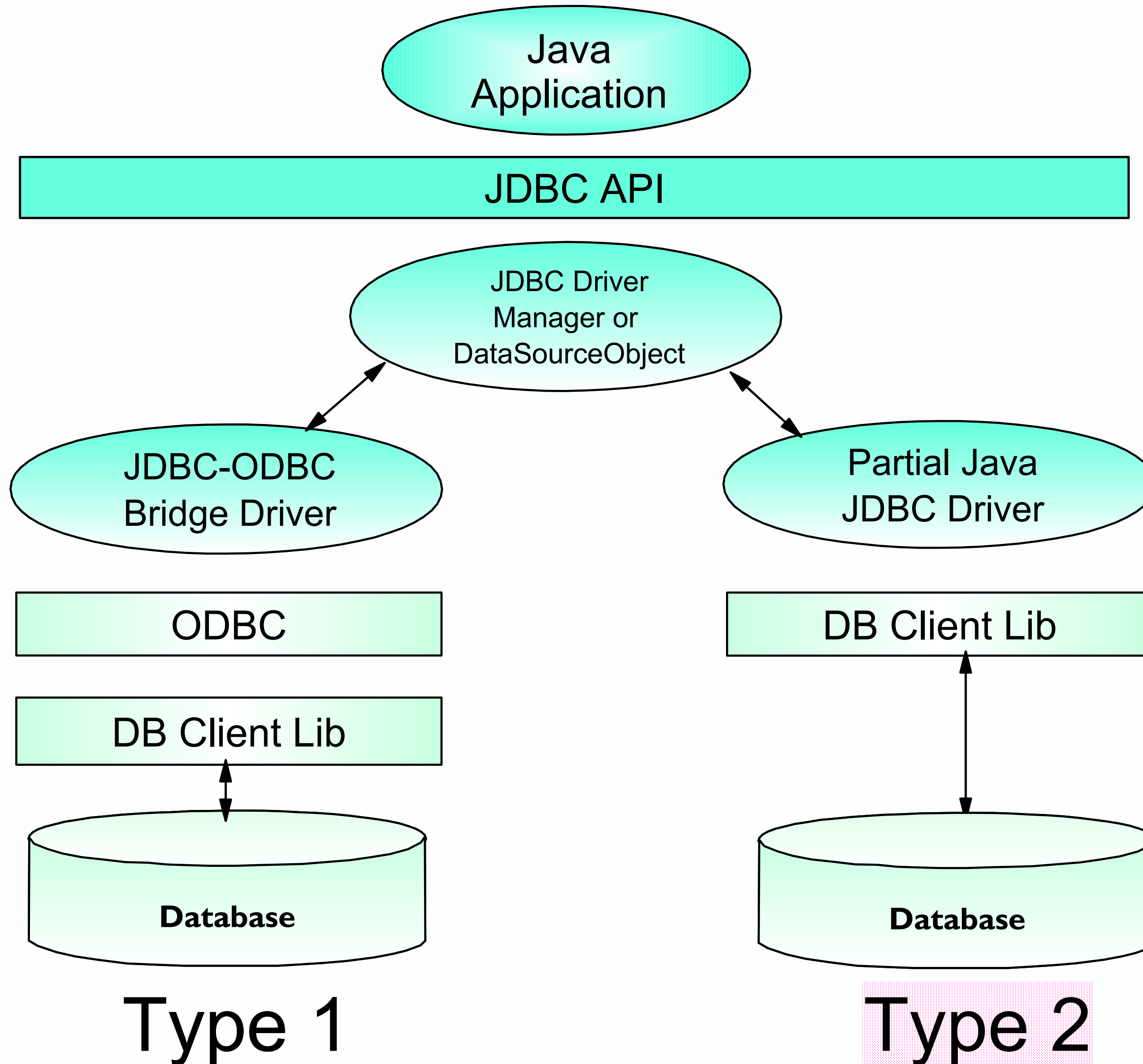


Basic Tasks of a JDBC Application

- Establish a connection with a database
- Execute SQL statements
- Process the results

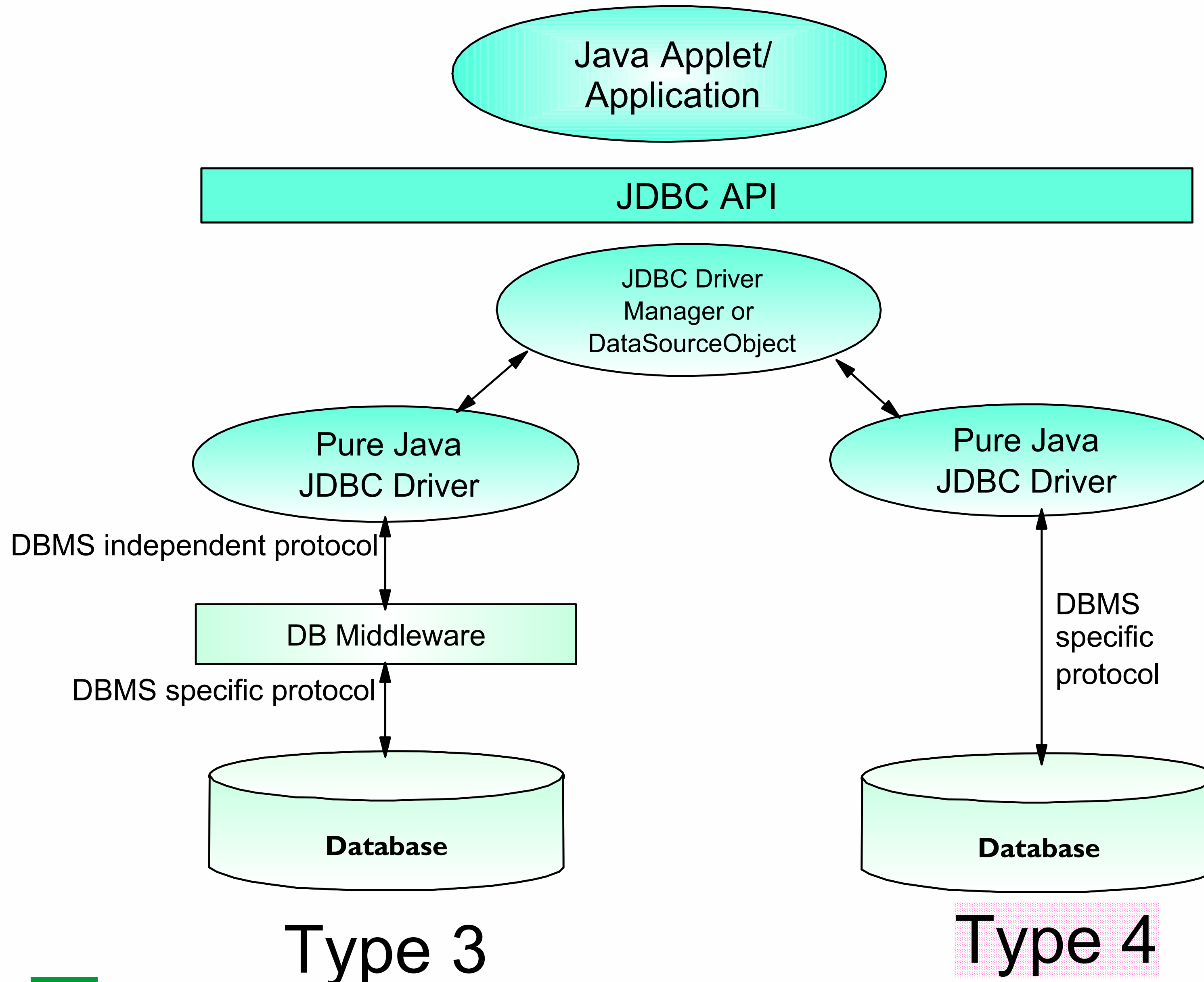
```
Connection con = DriverManager.getConnection (
      "jdbc:db2:sample", "login", "password");
PreparedStatement stmt = con.prepareStatement(
      "SELECT a, b, c FROM Table1");
ResultSet rs = stmt.executeQuery();
while (rs.next()) {
      int x = getInt("a");
      String s = getString("b");
      float f = getDate("c");
}
```

JDBC Driver Types



Type 1

JDBC Driver Types



SQLJ Overview

- **Static SQL syntax in Java**
- **Potential for wide DBMS vendor acceptance**
  - **IBM, Oracle, Sybase, Informix, Tandem...**
  - **SQLJ has been accepted by ANSI and is included in SQL99 standard**

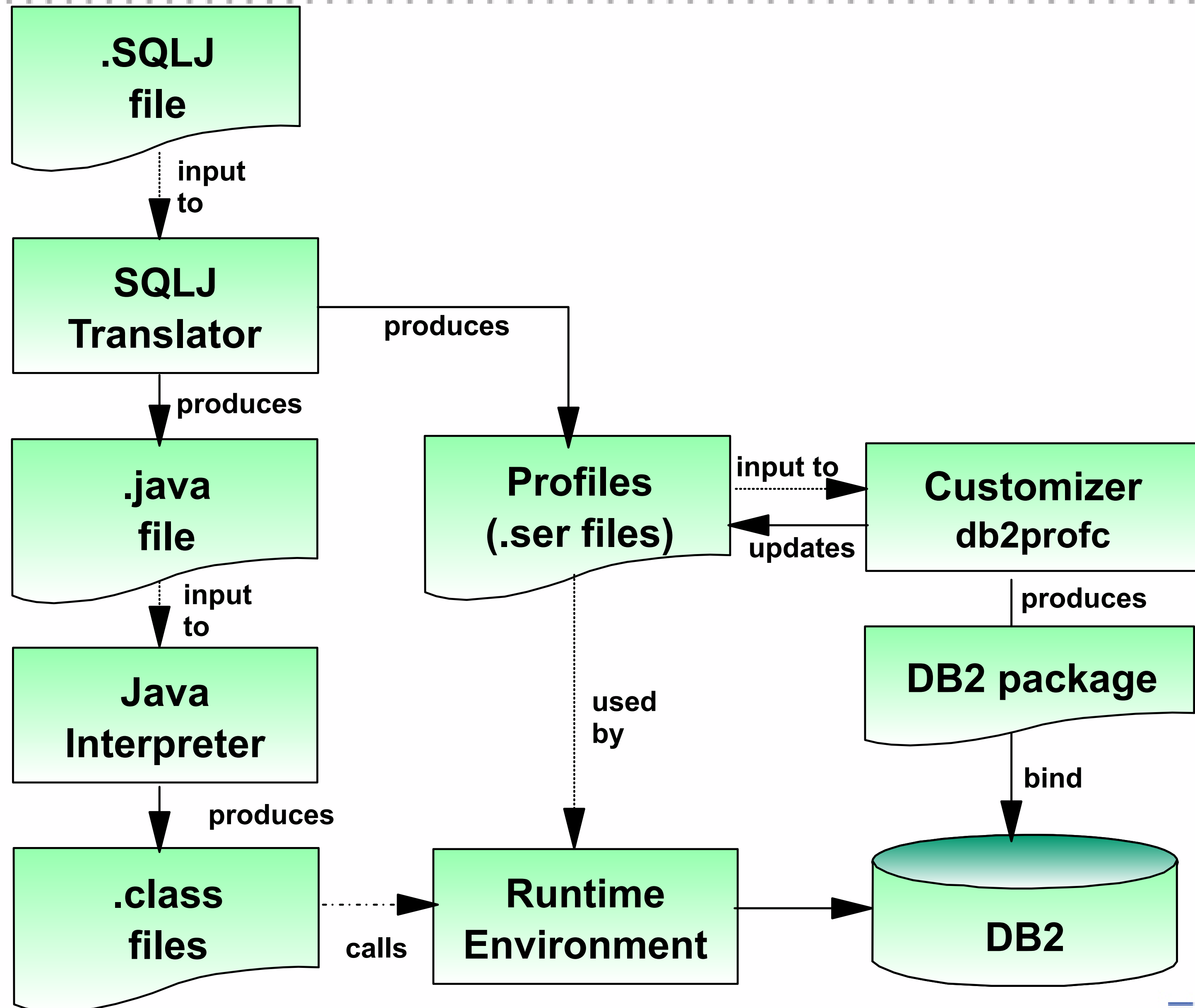
JDBC

```
java.sql.PreparedStatement ps =
    con.prepareStatement("SELECT ADDRESS FROM EMP  WHERE NAME=?");
ps.setString(1, name);
java.sql.ResultSet rs = ps.executeQuery();
rs.next();
addr = rs.getString(1);
rs.close();
```

SQLJ

```
#sql [con] { SELECT ADDRESS INTO :addr FROM EMP
                              WHERE NAME=:name };
```

SQLJ - Embedded SQL in Java



SQLJ versus JDBC...

Reasons to use SQLJ:

- Less complex & more concise than JDBC
- For DB2 (with optional customization step)
  - Better performance (not prepared at run-time)
  - Users can be authorized to programs, not tables
- Optional SQL checking at design-time
  - Syntax
  - Type mappings

Reasons you might not use SQLJ:

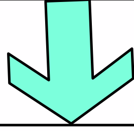
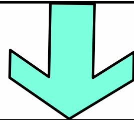
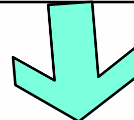
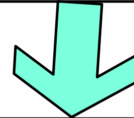
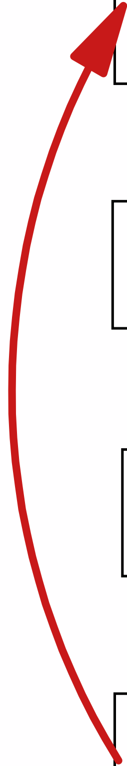
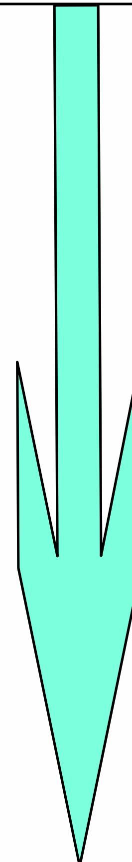
- More steps in build process
- Less flexible at run-time

SQLJ and JDBC inter operability

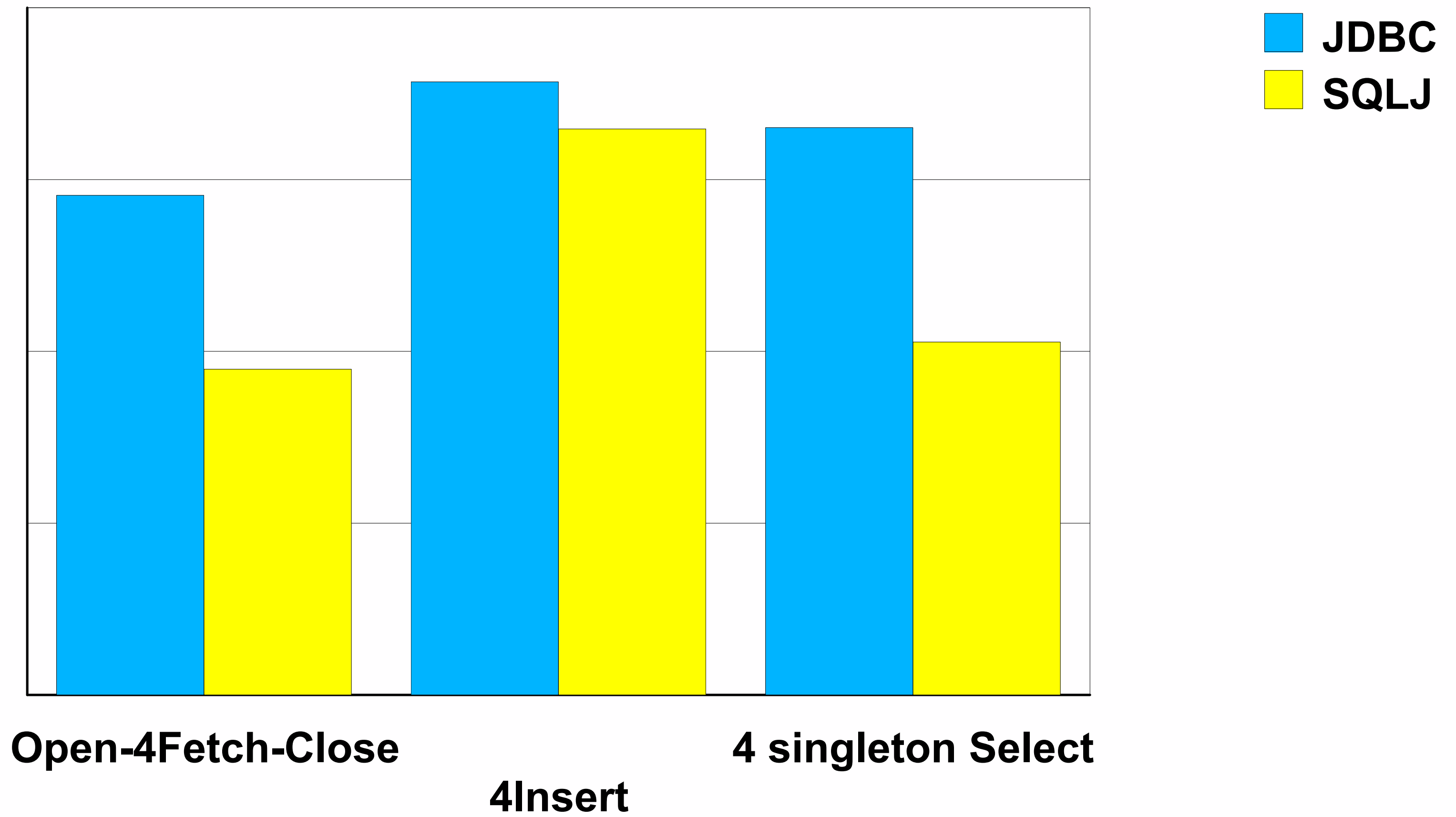
- You can mix SQLJ and JDBC in the same application
- SQLJ and JDBC can share the same connections
- JDBC result sets can be turned into SQLJ iterators, and vice versa

Static SQL is FASTER!

Dynamic SQL	Static SQL
Check auth for plan/pkg	Check auth for plan/pkg
Parse SQL statement	
Check table/view auth	
Calculate access path	
Execute statement	Execute statement

Static SQL is FASTER!



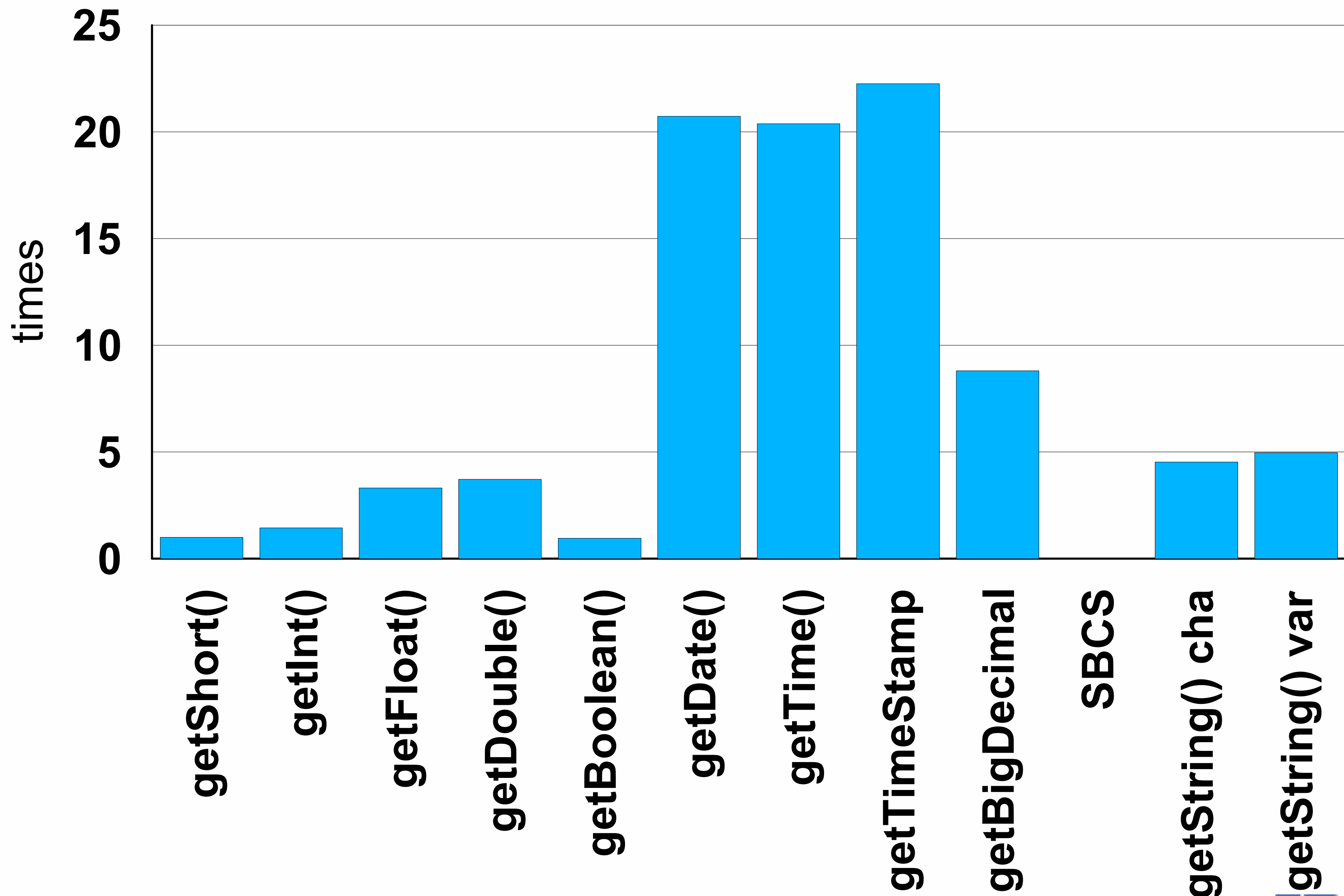
Design Guidelines for Application

- **Make sure that Java data types match DB2 data types**

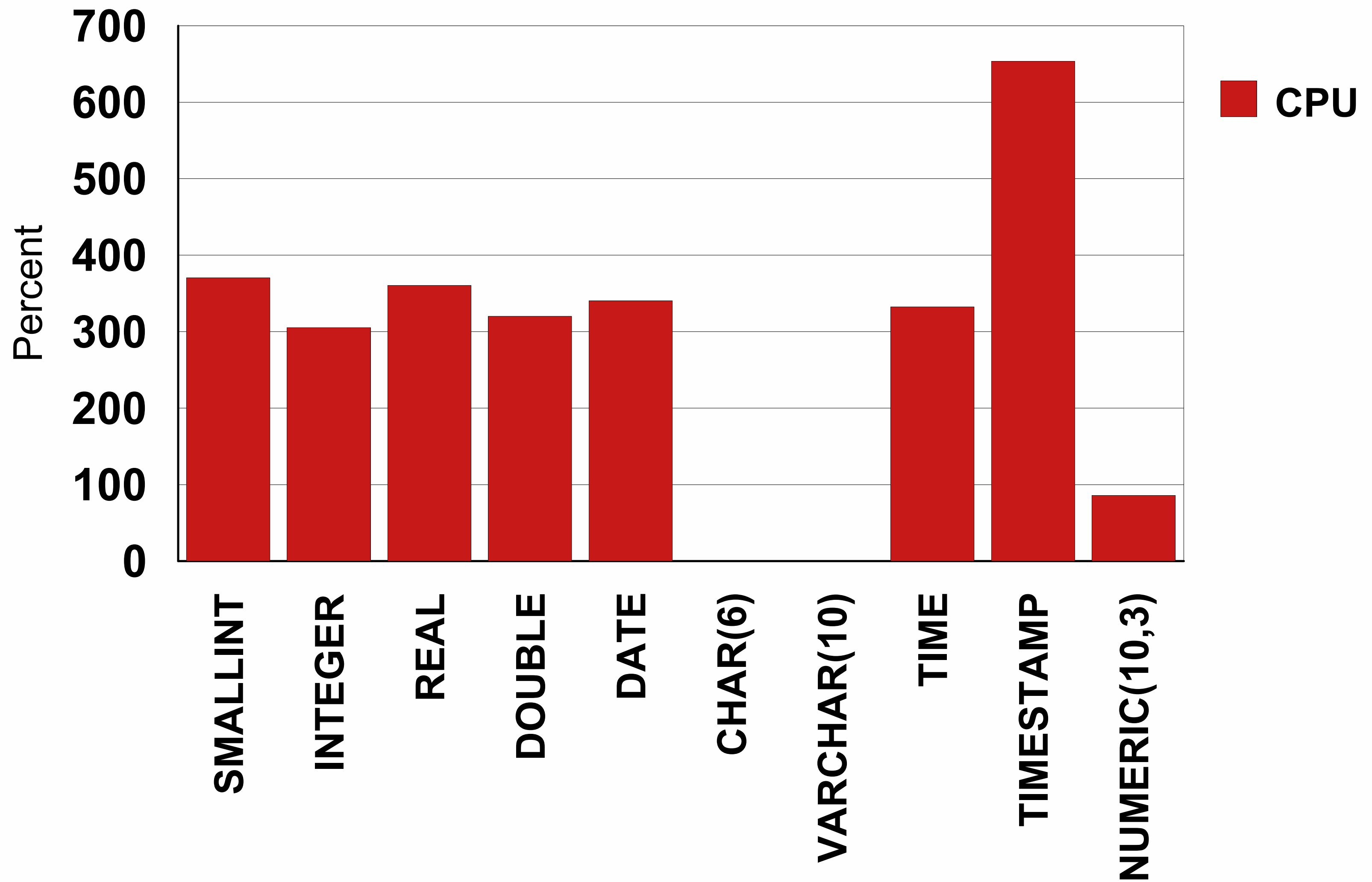
DB2 Data Type	Java Data Type	Comment
SMALLINT	short,boolean	no direct mapping for bit in DB2
INTEGER	int	
REAL	float	single precision
FLOAT, DOUBLE	double	double precision
DECIMAL(p,s), NUMERIC(p,s)	java.math.bigDecimal	with p=precision,s=scale keeps scale and precision in Java
CHAR, VARCHAR, GRAPHIC,VARGRAPHIC	String	
CHAR, VARCHAR FOR BIT DATA	byte[]	
TIME	java.sql.Time	
DATE	java.sql.Date	
TIMESTAMP	java.sql.Timestamp	

Design Guidelines for Application...

Relative Cost of getxxx() Processing



Design Guidelines for Application...



Design Guidelines for Application...

- **Only select and update columns as necessary**
  - **A Java object is created for every retrieved column.**
- **Store numbers as numbers**
  - **Conversion from EBCDIC/ASCII to Unicode required for Character data**
  - **Numbers are not dependent on encoding schema**
- **Turn autocommit off**
  - **Example: conn.setAutoCommit(false);**
  - **The default is on,**
forces a commit after every single SQL statement

Design Guidelines for Application...

- **Use JDBC DataSource connection pooling**
  - **"signon" support to reuse DB2 connection threads**
  - **Keeps JDBC connection objects**

Example of DataSource Definition

```
//executed only once by DBA
 ds = new com.ibm.db2.jcc.DB2DataSource();
 ds.setDatabaseName("TESTDB");
```

Example of Connection Pooling

```
//get connection from pool
Connection Conn1 = ds.getConnection("user","password");

// Turn off auto commit default
 Conn1.setAutoCommit(false);

....

 Conn1.close();
```

Design Guidelines for Application...

- **DB2 datatype CHAR vs. VARCHAR in database design**

- **Usage of CHAR**
  - **requires use of Java trim() function to eliminate the trailing blanks**
  - **higher CPU cost for Java applications**
  - **non-Java applications are not affected**

- **Usage of VARCHAR**
  - **easier for Java application**
  - **somewhat higher CPU cost "in DB2"**
  - **all applications are effected**

Design Guidelines - JDBC

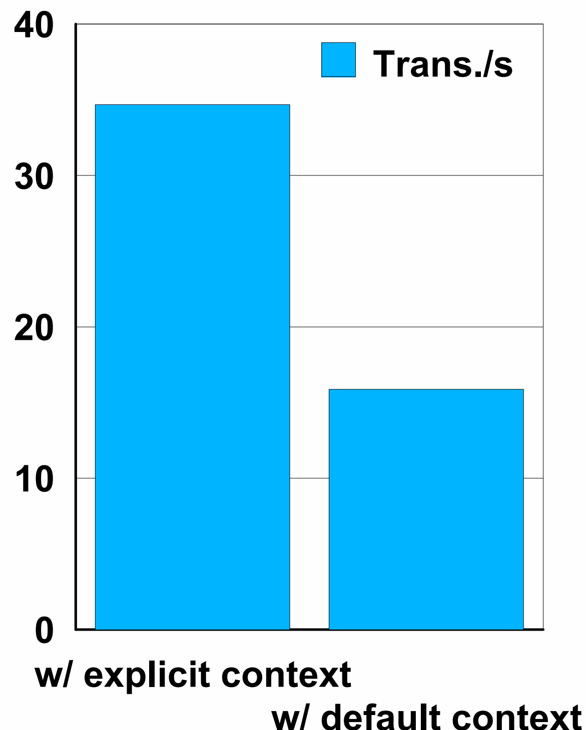
- **Release Resources**
  - **Close ResultSets**
    otherwise running out of available cursor
  - **Close PreparedStatements**
    otherwise running out of available cursor
    closing ResultSets is not sufficient
  - **Close CallableStatements**
    otherwise running out of available call sections
- **Use db2genJDBC**
  to adjust required JDBC resources

Design Guidelines - SQLJ

- **Customize SQLJ serialized profile with online checking**
  - **called by: db2profc ... -online=<db2_location_name>**
  - **without customization -
    SQLJ application is executed dynamically**
  - **online checking**
    - **access DB2 catalog to check
      SQLJ-supported compatibility/convertability**
    - **determines the length of string columns**

Design Guidelines - SQLJ...

- **Use explicit connection context objects**
  - **If connection context object is omitted a default connection context object is used**
  - **default connection context is not thread-safe**
  - **can create throughput bottle-neck**



Example of explicit connection context

```
// Connection context declaration
#sql context ctx;
...
//get context
myconn=new ctx(Conn1);
...
//use context in SQL
#sql [myconn] {set transaction isolation level read committed};
...
#sql [myconn] cursor001 = {SELECT FKEY,FSMALLINT,FINT
                           FROM WRKTB01 WHERE FKEY >= :wfkey};
...
//close context but keep database connection
myconn.close(ConnectionContext.KEEP_CONNECTION);
```

Design Guidelines - SQLJ...

- **Use positioned iterators**
  - **Named iterator uses positioned iterator under the cover plus name hashing**

Example of Named Iterator

```
// Named Iterator
#sql iterator TestCase001A (short Fkeycr,
           Time Ftime,  BigDecimal Fnum);
....
short wfkeycr;
Time wftime;
BigDecimal wfnum;
...
#sql [myconn] cursor002 = {SELECT FKEY, FTIME, FNUM
                    FROM WRKTB01};

while (cursor002.next()) {
      wfkeycr = cursor002.Fkeycr();
      wftime = cursor002.Ftime();
      wfnum = cursor002.Fnum();
}
```

Example of Positioned Iterator

```
// Positioned Iterator
#sql iterator TestCase001(short, Time, BigDecimal);
....
short wfkeycr;
Time wftime;
BigDecimal wfnum;
...
 #sql  [myconn] cursor001 = {SELECT FKEY, FTIME, FNUM
                    FROM WRKTB01};

#sql {FETCH :cursor001 INTO :wfkeycr,  :wftime, :wfnum};
```

What do the Acronyms?

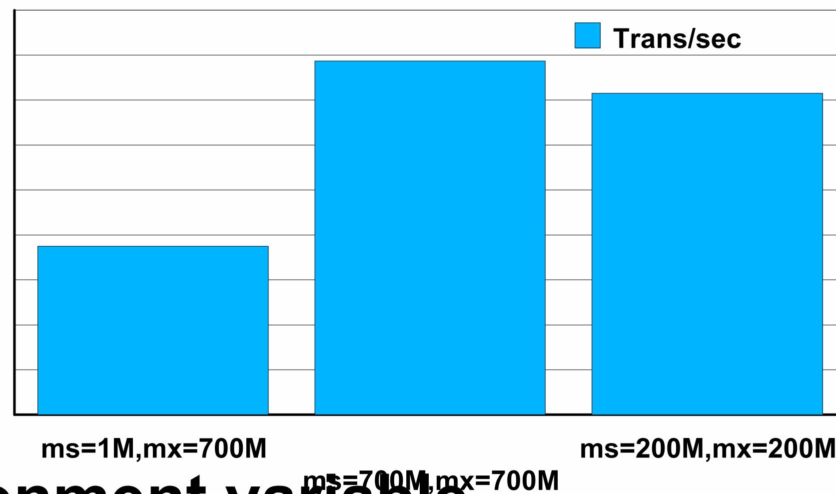
- JSP
- JNDI
- EJB
- RMI
- XML
- LDAP
- HTTP

Use Current System Levels

- **Hardware support for IEEE floating point**
  - **in G5 and higher S/390 processors**
  - **Use OS/390 V2R6 and above to exploit**
- **Keep current with JDK releases**
  - **still major performance improvements in each release or PTF**
- **Keep current with JDBC driver**
  - **major performance enhancements based on DB2 V7, JDBC 2.0 driver**

Environment Tuning Hints

- **Tune your JVM heap**
  - **Default heap size is**
    ms = 1M, mx = 8M
  - **Values between 300M and 400M are common in a production environment**
  - **Set ms and mx to an equal value**



- **Set environment variable _cee_runopts = "heappools(on)"**

Environment Tuning Hints

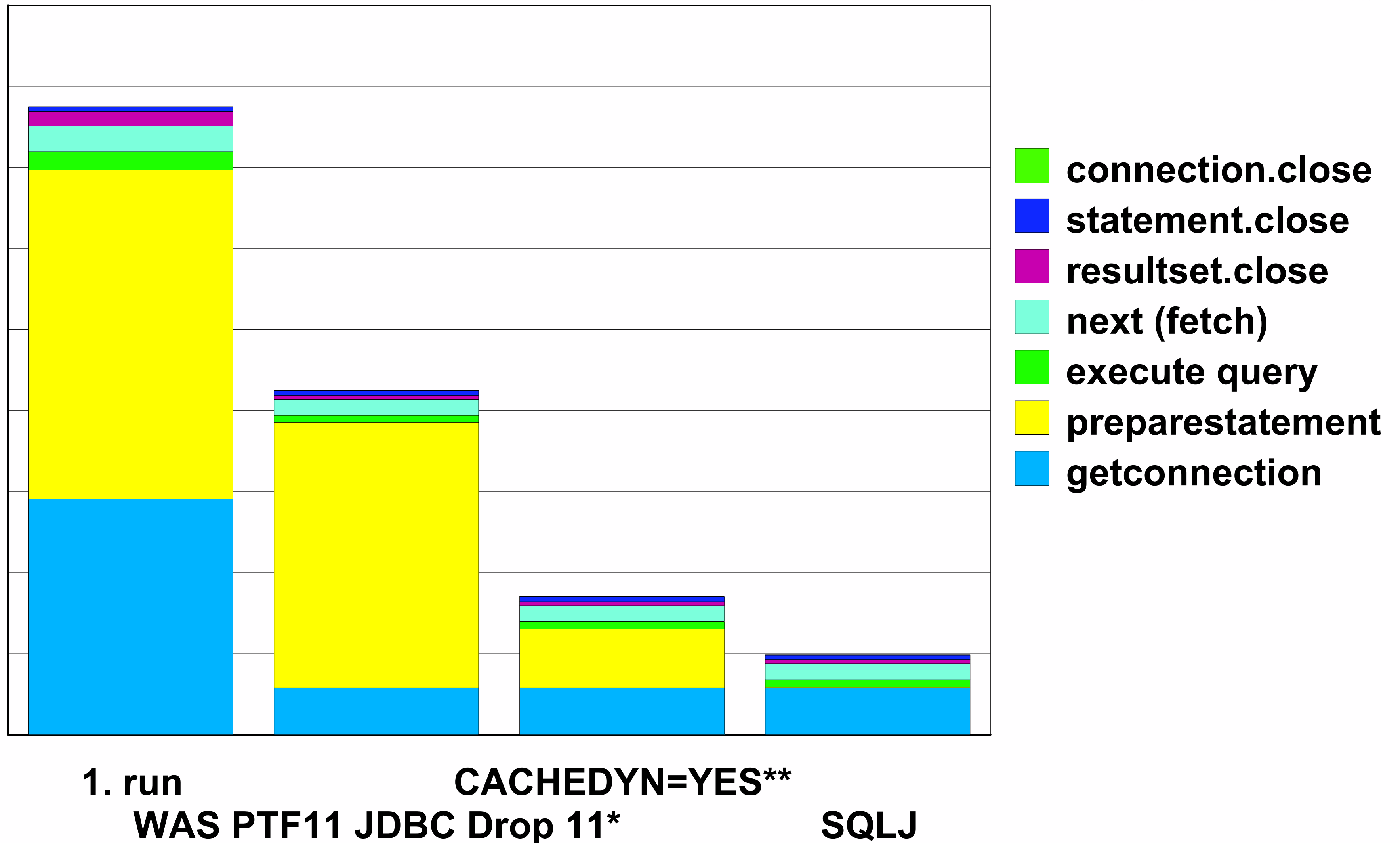
- **Recommended DB2 BIND options**
  - **DYNAMICRULES(BIND)**
    table access privileges of the binder used during program execution
  - **QUALIFIER**
    creator (schema name) for unqualified tables and views

- **Use dynamic SQL statement caching**
  - **Avoids full cost of preparing SQL**
  - **Processing cost close to static SQL**
  - **Recommended for JDBC/SQLJ**
    Cursor controlled update/delete executed dynamically in SQLJ

SQLJ/JDBC Driver Improvements

- **Context switching performance improvement**
- **Lower cost for column processing**
- **fewer JNI crossings (allows JIT to be more effective)**
- **Presume abort logging in RRS Attach**
- **JVM improvements:**
  - **"intrinsics" for inline JDBC column movement**
  - **"intrinsics" for code page conversion**
  - **JDK 1.3 JNI callback performance improvements**
- **SQLJ performance improvements for UPDATE or DELETE WHERE CURRENT OF**

SQLJ/JDBC Performance Study



* JDBC 2.0 connection pooling

** Hit in dynamic statement cache

DB2 V7 Java Column Processing Cost

Get function	V6 Delta
getString (char)**	-91%
getString (varchar)**	-91%
getBigDecimal	-66%
getInt	-8%
getShort	-27%
getDate	-50%
getTime	-51%
getTimestamp	-64%
getDouble	-6%
getFloat	-6%
getBoolean	-31%

Shipped via PQ48383

** Measured code pages other than 500 and 37

DB2 V7 Java Column Processing Cost...

Set function	V6 Delta
setString (char)**	-92%
setString (varchar)**	-92%
setBigDecimal	-3%
setInt	-1%
setShort	-23%
setDate	-4%
setTime	-4%
setTimestamp	-2%
setDouble	-37%
setFloat	-28%
setBoolean	-15%

Shipped via PQ48383

** Measured code pages other than 500 and 37

Summary

- **Use SQLJ for performance critical applications**
- **Stay current on maintenance**
- **Implement given guidelines**

Session Title: DB2, Java and Design for High Performance
Session #: D13

John J. Campbell
DB2 for z/OS and OS/390 Development
campbelj@uk.ibm.com