#SHAREorg

DB2 for z/OS Stored Procedures – Trends and Technology

Robert Catterall
IBM

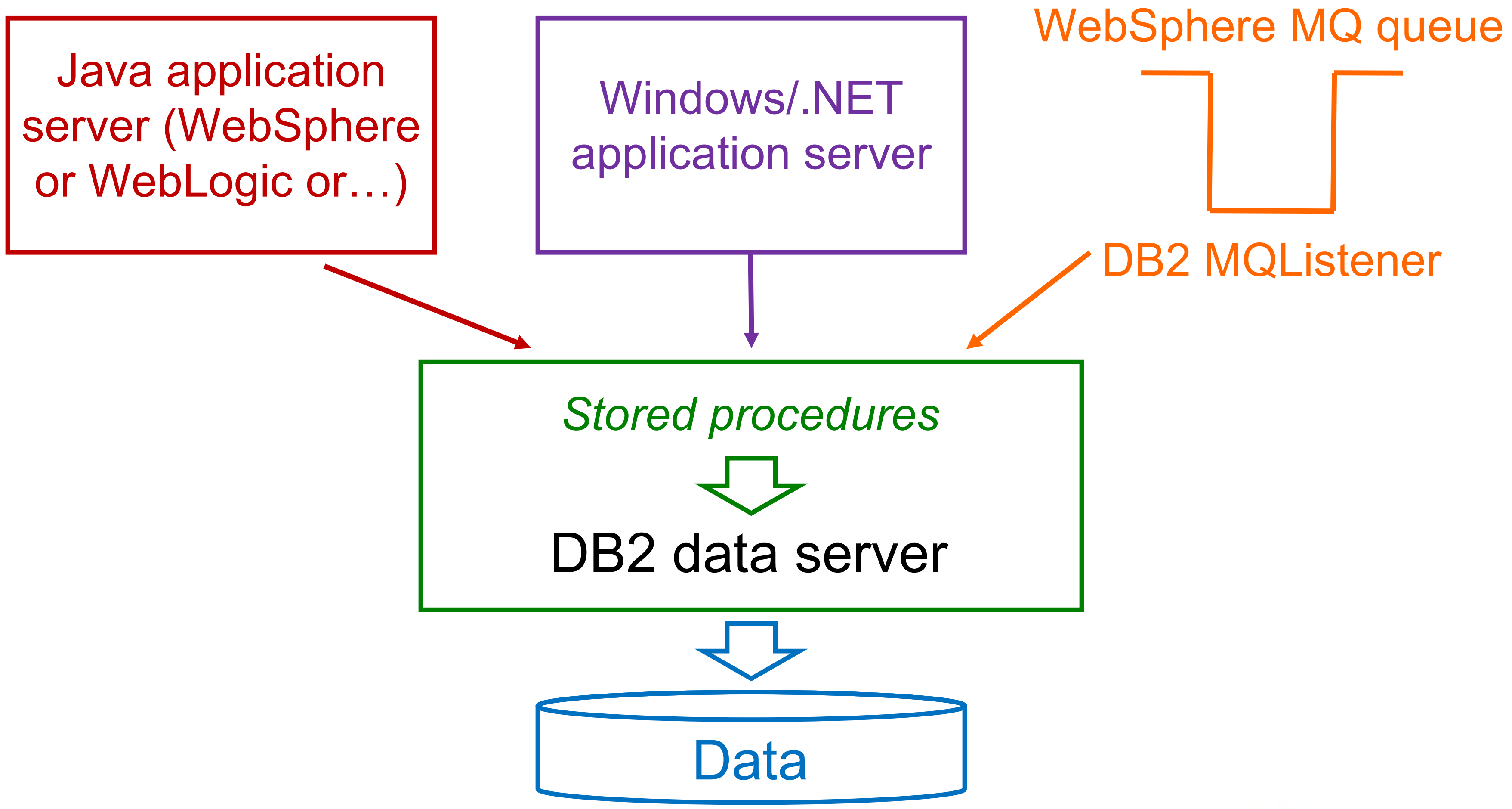
March 12, 2012
Session 10223

Agenda

- A vision of a modern DB2 for z/OS data-serving system
- A brief review of advances in DB2 for z/OS stored procedure functionality since DB2 V4
- Native SQL procedures
- Some go-forward recommendations
- Hints, tips, etc.

A vision of a modern DB2 for z/OS data-serving system

The big picture

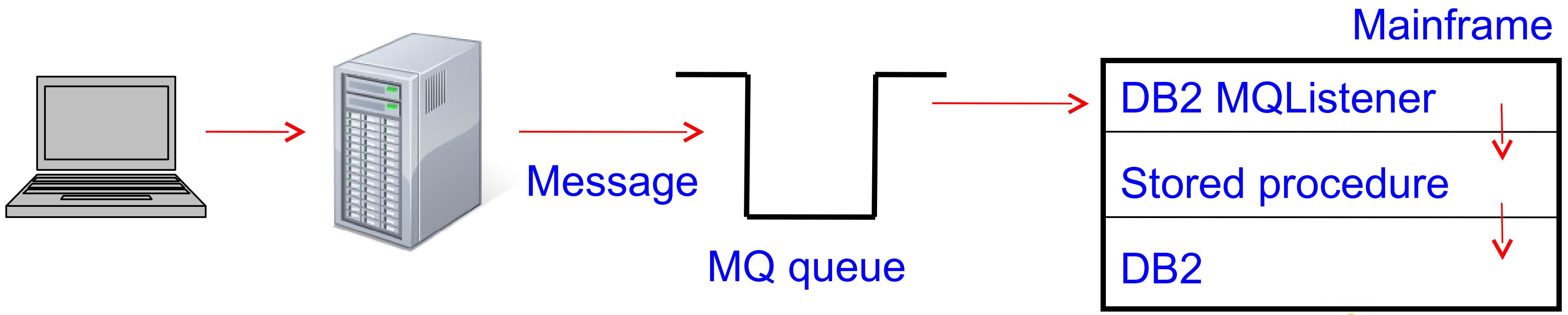

Java application server (WebSphere or WebLogic or…)
Windows/.NET application server
WebSphere MQ queue
DB2 MQListener
Stored procedures
DB2 data server
Data

Points about the “vision” diagram

- The DB2 server platform is not specifically identified – could be z/OS, or Linux, or UNIX, or Windows
- The DB2 server is a pure database server – there is no transaction management subsystem on the server
    - Standard set-up for some time in distributed systems world
- Mainframes with DB2 often have CICS or IMS, too – usually because the organization ran a DB2-accessing transactional workload before stored procedure functionality was available
    - Static, server-side SQL (good for scalability) can be packaged in CICS or IMS transaction programs – or in stored procedures
    - You <u>can</u> have a high-volume transactional workload with the mainframe functioning as a data server (i.e., all SQL requests come through the DB2 Distributed Data Facility)

More on the vision diagram

- MQ and DB2 stored procedures can be a good combination
  - Client program puts a message on an MQ queue
  - A process called the DB2 MQListener (provided with DB2) can call a stored procedure in response to a message arriving on a queue – the message is the input to the stored procedure
    - Stored procedure takes input, might drive a DB2 table insert
  - Another option: MQ can invoke a CICS transaction to process a message



A great use of MQ and DB2 stored procedures

- For database updates that need to occur in near-real-time – but not synchronously – relative to end user input
  - Possible examples: customer changes personal information (e.g., address), or makes an online payment
    - User clicks on “Submit,” input information captured in MQ message
    - Application can respond to end user with (for example) “Your update has been received and will be applied to your profile momentarily”
    - Back-end DB2 database updates likely to occur within seconds
  - Advantages of asynchronous approach:
    - Potentially better end-user response time (fast reply after “Submit”)
    - Improved system availability (from user’s perspective): if back-end database server is unavailable, messages simply accumulate on queue and are processed when database server is back online

A brief review of advances in DB2 for z/OS stored procedure functionality since DB2 V4

V4: stored procedures introduced

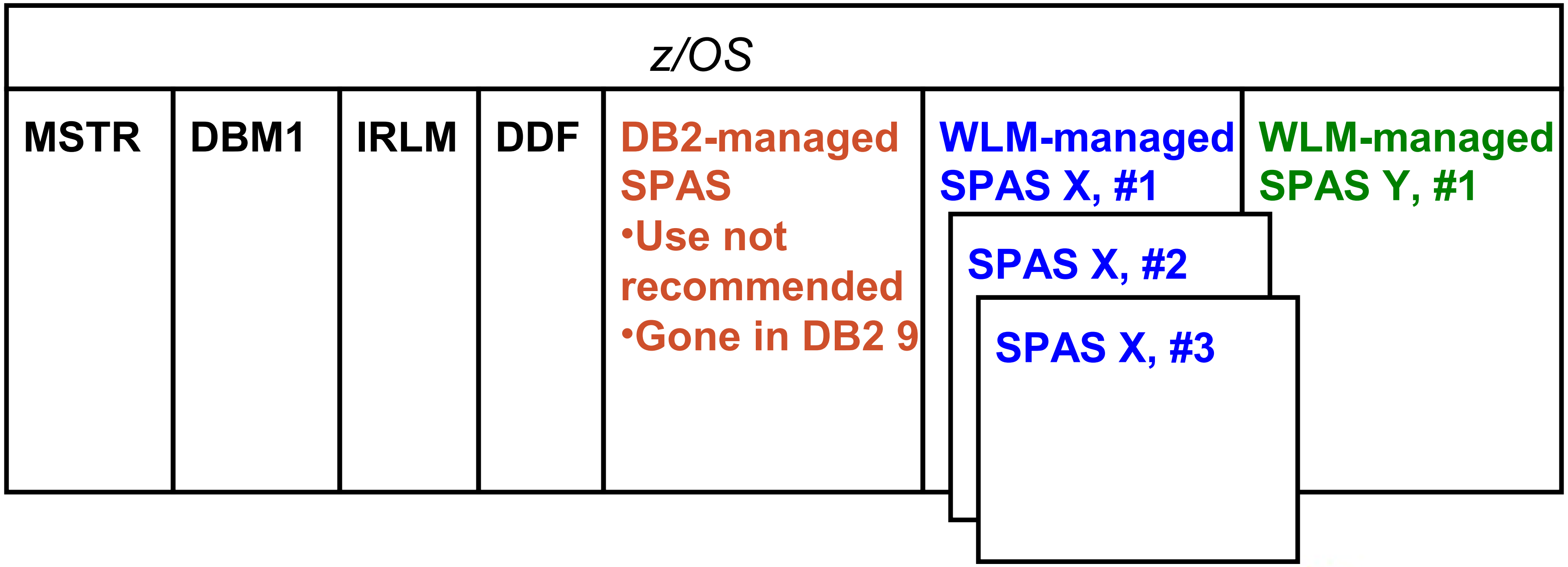
- New address space: DB2-managed stored procedure address space (aka SPAS)

z/OS				
MSTR	**DBM1**	**IRLM**	**DDF**	**SPAS** **•Stored procedure programs run here, using DB2 call attach facility interface**

- A shortcoming: caller could not fetch results of cursor declared and opened in a stored procedure
  - Had to use output parameters (not ideal for result sets with indeterminate number of rows, not feasible for large result sets)

V5: two significant enhancements

- Caller of a stored procedure could fetch rows from a cursor declared and opened in the stored procedure
- WLM-managed stored procedure address spaces introduced



V6 and V7: DDL, SQLPL, COMMIT

- DB2 V6: CREATE/ALTER/DROP PROCEDURE statements added to DDL
  - Before that, DBA had to insert/update/delete rows in SYSPROCEDURES catalog table (tedious, error-prone)
  - Catalog: SYSPROCEDURES table out, SYSROUTINES in
- DB2 V7: SQL Procedure Language introduced (SQLPL)
  - Stored procedures could be written entirely in SQL
    - SQL was extended to include logic flow-control statements such as GOTO, IF, ITERATE, LEAVE, LOOP, REPEAT, and WHILE
    - SQL procedure converted "under the covers" to C program with embedded SQL – executes as external stored procedure program
- DB2 V7 also allowed for COMMIT and ROLLBACK to be issued from stored procedure

V8: flexible abend limit, WLM synergy

- Stored procedure abend limit can be set at individual stored procedure level, versus a DB2 subsystem-wide setting
  - If a stored procedure abends n times, placed in stopped status (once fixed, restarted via -START PROCEDURE command)
  - Might want a higher abend limit for newer stored procedures
- Better synergy with z/OS Workload Manager
  - DB2, z/OS work together to optimize number of tasks in a SPAS (stored procedure TCB conceptually like CICS-DB2 subtask TCB)
  - "Just right" number of tasks in a SPAS is good for CPU efficiency
  - Number of tasks in a SPAS may be varied up or down, but NUMTCB (parameter specified when defining WLM execution environment) remains upper bound for a SPAS

V9: “native” SQL procedures

- As far as I’m concerned, the most important advance in DB2 for z/OS stored procedure technology since stored procedures were introduced with DB2 V4

Gets its own section in this presentation…

(DB2 10 stored procedure enhancements will be covered later in this session)

Native SQL procedures

Native SQL procedures: big change

- Before: SQL procedure turned into a C language program under the covers
  - Runs as an external stored procedure in a WLM-managed SPAS
  - Not-in-DB2 part of a C program generally consumes more CPU than does equivalent COBOL code (though less than Java)
- For a native SQL procedure (available beginning with DB2 9 in new function mode), the one and only executable is the procedure's package
  - So, a native SQL procedure runs entirely in the DB2 database services address space (DBM1)

Native SQL procedure efficiency (1)

- An external stored procedure runs under its own TCB
  - Caller's task (TCB or SRB) is suspended, and stored proc task uses caller's thread for communication with DB2
    - In some cases, there can be processing delays and a build-up of DBM1 virtual storage consumption associated with the switching of threads from calling-program tasks to stored procedure tasks

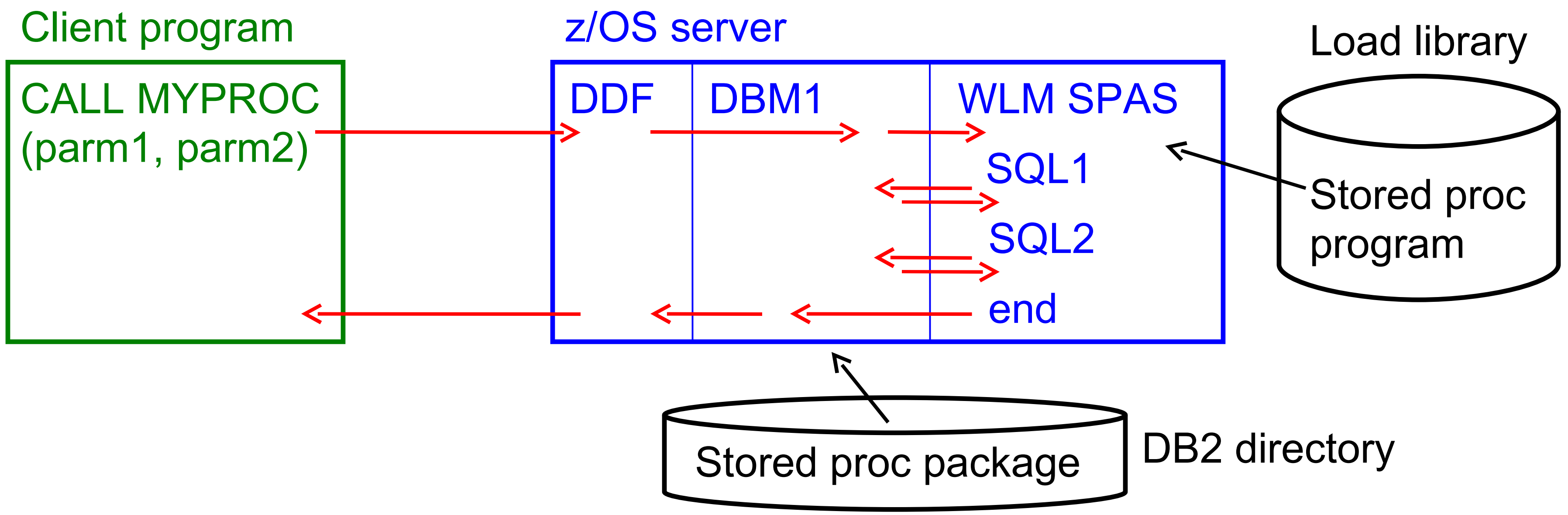
- Native SQL procedure runs under the calling program's task
  - No queuing, no delays related to thread-switching

Native SQL procedure efficiency (2)

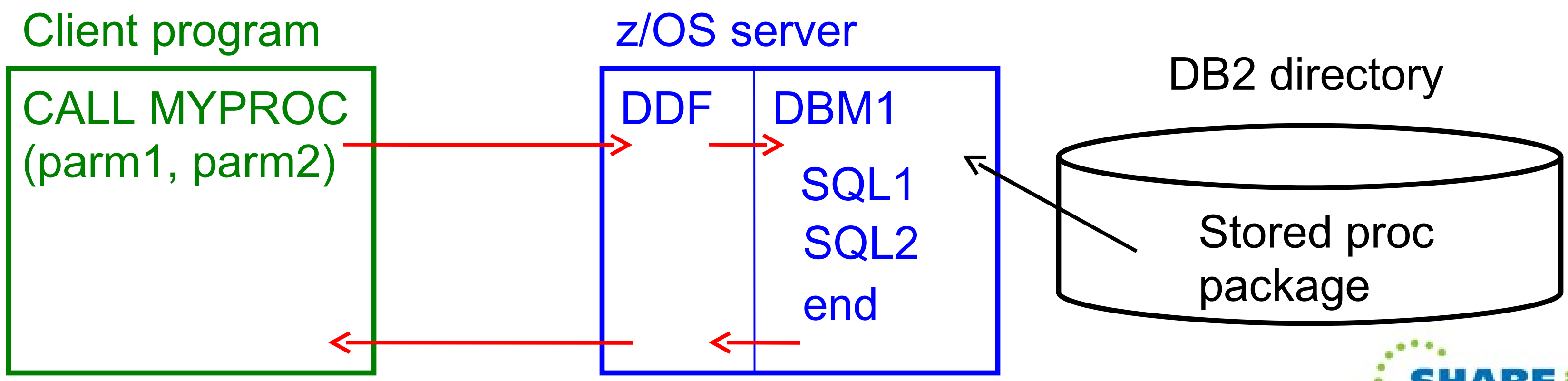
- For every SQL statement in an external stored procedure (or any other external-to-DB2, SQL-issuing program), an “addressability round trip” is required
  - Program’s task switches addressability from “home” address space (could be a WLM-managed stored procedure address space) to DB2 DBM1 for SQL execution, then switches back
  - Each round trip probably consumes a few thousand instructions, and that’s just the back-and-forth – not SQL execution in DBM1
- Native SQL procedures eliminate this extra path length
  - With the CALL to the native SQL procedure, you’re already in DBM1, and you stay there until stored procedure completes

External vs. native procedure

External



Native



The zIIP factor

- zIIP: specialty engine that costs less than a general-purpose processor and does not factor into mainframe software pricing
- A native SQL procedure is zIIP-eligible if it is invoked via a remote call through the DB2 Distributed Data Facility (DDF)
  - Why restricted to remote vs. local CALLs (“local CALLs” being those that are issued by programs running on the same server as DB2)?
    - Technically, because DDF requests run under enclave SRBs
    - My opinion: IBM is encouraging organizations to use DB2 for z/OS as a data server in multi-tier client-server application environments
- Amount of CPU processing directed to a zIIP engine tends to be 55-60% for native SQL procedures called through DDF

Some CPU figures

- From a presentation delivered by IBM's John Campbell at a conference
  - Results obtained using IRWW OLTP workload (a standard workload used by IBM for benchmarking purposes)
  - Stored procedures invoked via DRDA and DDF

Type of procedure	Base cost (CPU/tran)	Cost after zIIP redirect
COBOL	1X (base)	.88X
External SQL	1.62X	1.49X
Native SQL	1.14X	.65X

- Note: test was run a while ago using DB2 9 – native SQL procedure performance picture is even better now
  - **DB2 10 delivered improved SQL procedure language performance**

Native SQL procedure functionality benefit

- A nested compound statement (a compound SQL statement within another compound SQL statement) is allowed in a native SQL procedure, not in an external SQL procedure
  - Compound statement: a group of SQL statements, bounded by BEGIN and END
    - Within a compound statement, variables, cursors, and condition handlers can be declared
    - A SQL procedure will very often contain a compound SQL statement
  - With nested compound statements, condition handlers can have their own compound statements (enables more sophisticated error handling)
  - Also: better DB2 Family compatibility (DB2 for LUW already supported nested compound statements in SQL procedures)
    - Important for cross-platform development

Native SQL procedure lifecycle benefit

- Simpler creation, management, maintenance versus external stored procedures
  - No external-to-DB2 resources involved (e.g., no source, object, or load libraries)
    - The native SQL procedure package is the executable, and it is stored in the DB2 directory
  - No external-to-DB2 processes involved (e.g., no need for compile and link processes)
- No worries about mismatch between program and package when you execute a native SQL procedure, because the program <u>is</u> the package
  - Native SQL procedure has a consistency token but it's just a "synonym" for the procedure's version ID

"Simple is good"

Some go-forward recommendations

Getting from here to there…

- “There” being a situation in which you’re leveraging DB2 stored procedure functionality
- First: use stored procedures (if not already doing so)
- If not yet in DB2 9 NFM (or DB2 10 CM9 or NFM), consider developing some external SQL stored procedures…
  - …even if COBOL has been your preferred stored procedure programming language
  - I believe that SQL procedures are the way of the future, and it would be a good idea to get ready for that future
  - Can later convert external SQL procedures to native

Note: DB2 10 allows user-defined functions (UDFs) to be written in SQLPL, too

If you use CICS…

- Consider making functionality of some CICS-DB2 transactions available to DRDA requesters via stored procedure calls
- Could be done by converting COBOL CICS program to COBOL stored procedure (often involves little change)
- Could also invoke CICS transaction via stored procedure
  - One option: DSNACICS stored procedure that comes with DB2
  - Alternative: code to CICS EXCI interface yourself, using either EXCI CALL (more control), or EXEC CICS (easier to code)
- Note that combination of stored procedure and CICS transaction can be very good for VSAM data access
  - Once opened, VSAM file stays allocated to CICS (avoids open/close overhead that can limit throughput if file directly accessed from stored procedure)

Running DB2 9 NFM or DB2 10 CM9 or NFM?

- Use native SQL stored procedures, but be deliberate about this if you're already using external SQL procedures
  - The simpler lifecycle processes of native SQL procedures are less compelling if your external SQL procedure infrastructure is well established and mature
  - Bottom line: advantages of native SQL procedures versus external SQL procedures make conversion worthwhile, but you'll want to do that in a non-disruptive fashion
- Remember that native SQL procedures can be a good driver of zIIP utilization

Using external SQL procedures?

- Get familiar and comfortable with the different lifecycle processes of native SQL procedures
  - New DEPLOY option of BIND PACKAGE
  - New ACTIVATE VERSION option of ALTER PROCEDURE
- Maybe convert external SQL procedures to native SQL procedures when upgrading existing application
  - Sometimes, as simple as dropping and recreating the procedure without the FENCED and EXTERNAL options, and without a WLM ENVIRONMENT specification
    - May need WLM ENVIRONMENT FOR DEBUG MODE
  - Sometimes, not so simple (more on this to come)
- Consider using native SQL procedures for new DB2 application development

Hints, tips, etc.

Native SQL procedures and scalability

- Question: “I thought that having multiple WLM-managed stored procedure address spaces was good for scalability. Native SQL procedures run in one address space – won’t that have a constraining effect on throughput?”
- Answer: NO, it will not
  - Think about it: a native SQL procedure’s executable is a package, and packages always run in DBM1
    - If you’re running 1000 CICS-DB2 transactions per second from multiple CICS regions, each one has a package that runs in DBM1
  - Don’t worry about DBM1 “running out of tasks” – a native SQL procedure runs under the caller’s task, which is external to DBM1

Native SQL procedures and system stability

- Question: “Multiple stored procedure address spaces boost availability (e.g., you can isolate new stored procedures in an address space). With native SQL procedures all running in DBM1, won’t that negatively impact application stability?
- Answer: NO, it will not
  - Think about it: everything that executes in DBM1 is DB2-generated, DB2-managed code
    - Multiple address spaces for external stored procedures help to protect the system from an error that might exist in user-written code – that’s not a problem with native SQL procedures

Native SQL procedure source code management (SCM)

- Existing vendor-supplied SCM tools applicable when there is an external-to-DB2 executable and associated source code
  - What to do when you're working with a native SQL procedure, for which the language is SQLPL and the "source" is the CREATE PROCEDURE statement?
- In response to user requests for help in this area, APAR PM29226 (DB2 9 and 10 PTFs available in September, 2011) modified sample job DSNTEJ67
  - The job facilitates conversion of an external SQL procedure to a native SQL procedure, but don't get hung up on that if you don't have any external SQL procedures!
    - The new functionality is intended to assist with native SQL procedure source code management

More on APAR PM29226

- The external-to-native SQL procedure conversion accomplished via modified DSNTEJ67 illustrates use of new services (implemented via REXX routines)
  - One service extracts SQL procedure source from catalog, places it in a file (or, if you wish, into a string)
  - Another service invokes the SQLPL precompiler, and produces a listing of a SQL procedure
  - Another service enables one to change various elements of the SQLPL source for a procedure: schema, version ID, all the options
    - Can specify new values, or have values removed
  - Also a service to deploy SQLPL source

External-to-native conversion (1)

- Some SQL source code changes may be required
- One reason for that: some error-handling logic that worked for an external SQL procedure won't produce the desired behavior in a native SQL procedure
  - As previously noted, native SQL procedures allow nested compound statements, providing a means of coding multi-statement error handlers
    - Lacking that option, people coding external SQL procedures would sometimes use an IF block to implement the multi-statement handler
    - Problem: an "always true" condition used to enter an IF-based handler (IF 1=1 THEN…) will – in a native SQL procedure – clear the diagnostics area (oops)
    - In "going native", change these condition handlers to compound SQL statements set off by BEGIN and END

External-to-native conversion (2)

- Another potential reason for SQL source code changes when converting from external to native: differences in unqualified column/variable/parameter name resolution
    - You could have in your SQL procedure an unqualified variable or parameter name that's the same as the name of a column in a table accessed by the procedure
        - External procedure: DB2 will check first to see if variable of that name has been declared, then if it's the name of one of the procedure's parameters – if neither is true, assumption is that it's a column name
        - Native: DB2 will check first to see if name is that of a column of a table referenced by the procedure, then if a variable of that name has been declared, then if it's the name of a parameter
    - Solution: use qualified names, or a naming convention that identifies parameters and variables (e.g. use p_ or v_ prefixes)

External-to-native conversion (3a)

- What about the external procedure’s collection?
  - The package of an external SQL procedure can be bound into any collection, and that collection name can be specified via the COLLID option of CREATE PROCEDURE
    - By default, calling program will search in COLLID collection for external procedure’s package
  - When a native SQL procedure is created, the collection name for the package will be the same as the procedure’s schema name

External-to-native conversion (3b)

- If the package of a to-be-converted external SQL procedure was bound into a collection with a name other than the procedure's schema name:
  - Ensure that the collection with the same name as the procedure's schema will be searched when the native SQL procedure is called,

    -or-
  - Put a SET CURRENT PACKAGESET in the body of the native SQL procedure, referencing external procedure's collection name, and bind a copy of the native SQL procedure's "root" package into that collection

External-to-native conversion (4)

- For more conversion information, check out the brief (just a few pages) but highly informative IBM “Technote” at this URL:

http://www-01.ibm.com/support/docview.wss?uid=swg21297948

PROGRAM TYPE SUB vs. MAIN

- An option on the CREATE (or ALTER) PROCEDURE statement for an external stored procedure
- TYPE SUB has been observed to reduce CPU consumption associated with a stored procedure by 10% in some cases
  - HOWEVER, TYPE SUB means that the program is responsible for initialization of work areas
  - Some users tried TYPE SUB, then went back to TYPE MAIN because former led to “unpredictable results,” due to stored procedure programs not effectively initializing work areas
- TYPE SUB is good for performance, but ensure that your stored procedure programs are well suited to run as subroutines

STAY RESIDENT YES or NO

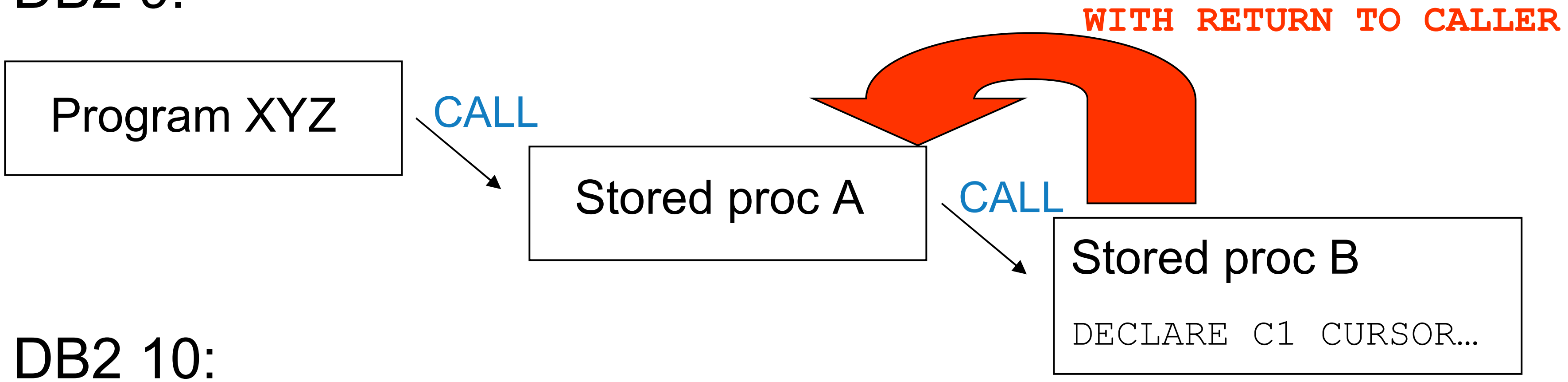
- Another option of CREATE (or ALTER) PROCEDURE for an external stored procedure
- YES can improve stored procedure CPU efficiency, but it should NOT be used for stored procedure programs compiled and linked as non-reentrant and non-reusable
  - Go with STAY RESIDENT NO for stored procedure programs that are non-reentrant and non-reusable
  - If STAY RESIDENT NO is specified for a frequently-executed stored procedure, module load time can be reduced by loading from the z/OS Virtual Lookaside Facility (VLF)

DB2 10: RETURN TO CLIENT cursors

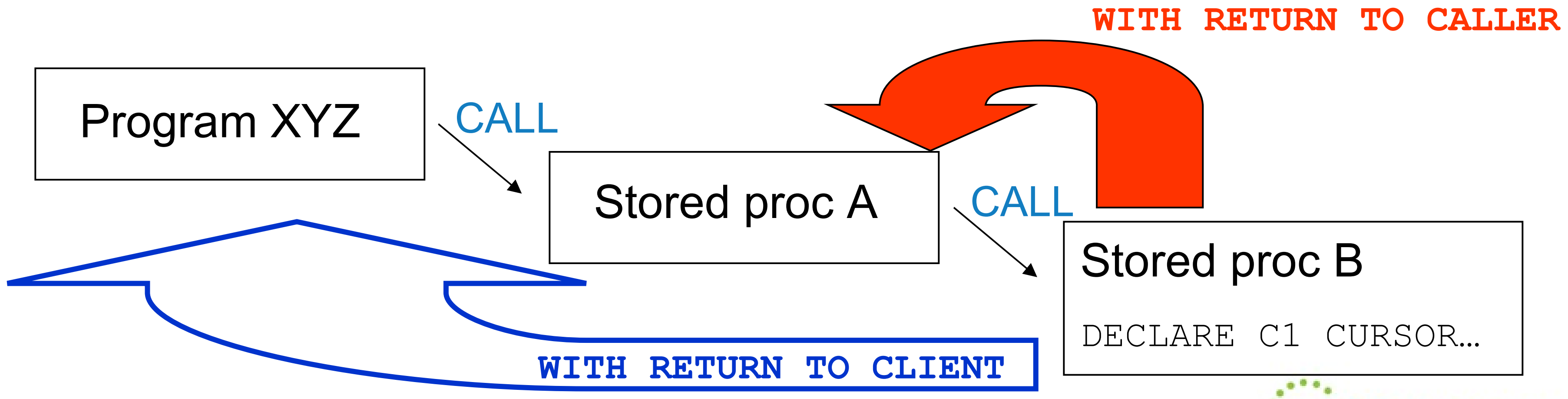
- DB2 9: stored procedure can return result set one level up in chain of nested procedures (WITH RETURN TO CALLER)
  - For example, if program PROG_A calls stored procedure PROC_B, and PROC_B calls PROC_C, PROC_B can fetch from a cursor declared and opened in PROC_C, but PROG_A cannot
    - If PROG_A needs that result set, PROC_C can put it in a temporary table, and PROG_A can get the rows from that temp table, OR
    - PROC_B can declare and open a cursor referencing the temp table, and PROG_A can fetch the result set rows through that cursor)
- A DB2 10 stored procedure can declare a cursor WITH RETURN TO CLIENT (just like DB2 for LUW)
  - “Top-level” program (caller of first stored procedure, which calls another stored procedure) can fetch rows, but cursor’s result set is invisible to stored procedures between it and top-level program

Previous slide’s point, in a picture…

- DB2 9:



- DB2 10:



Thanks for your time!

Robert Catterall

rfcatter@us.ibm.com