December 7, 2011

Baltimore/Washington
DB2 Users Group

DB2 for z/OS Stored Procedures – Trends and Technology

Robert Catterall
IBM
rfcatter@us.ibm.com

Agenda

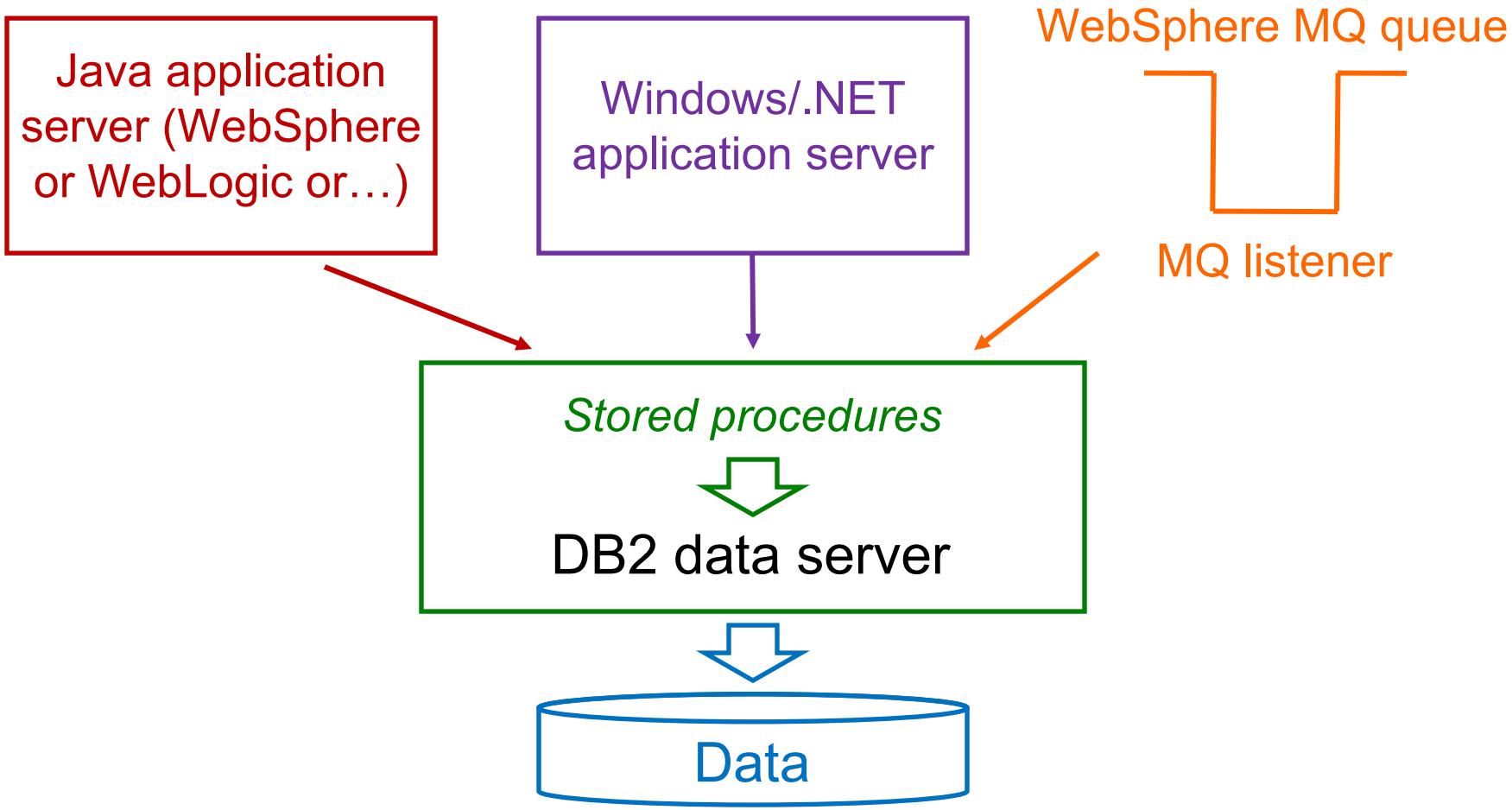
- A vision of a modern DB2 for z/OS data-serving system
- A brief review of advances in DB2 for z/OS stored procedure functionality since DB2 V4
- Native SQL procedures
- Some go-forward recommendations
- Hints, tips, etc.

Information Management

IBM

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

The big picture



Points about the “vision” diagram

- The DB2 server platform is not specifically identified – could be z/OS, or Linux, or UNIX, or Windows
- A pet peeve of mine
  - Quit using “DB2” as shorthand for DB2 for z/OS and “UDB” as shorthand for DB2 for Linux/UNIX/Windows
  - Perpetuates the unhelpful notion DB2 for z/OS and DB2 for LUW incompatibility, whereas from a client/server application development and architecture perspective, the two DB2 platforms are virtually identical
  - Just say “DB2”
    - If you want to refer more specifically to a DB2 server platform, say “DB2 for z/OS” or “DB2 for LUW”

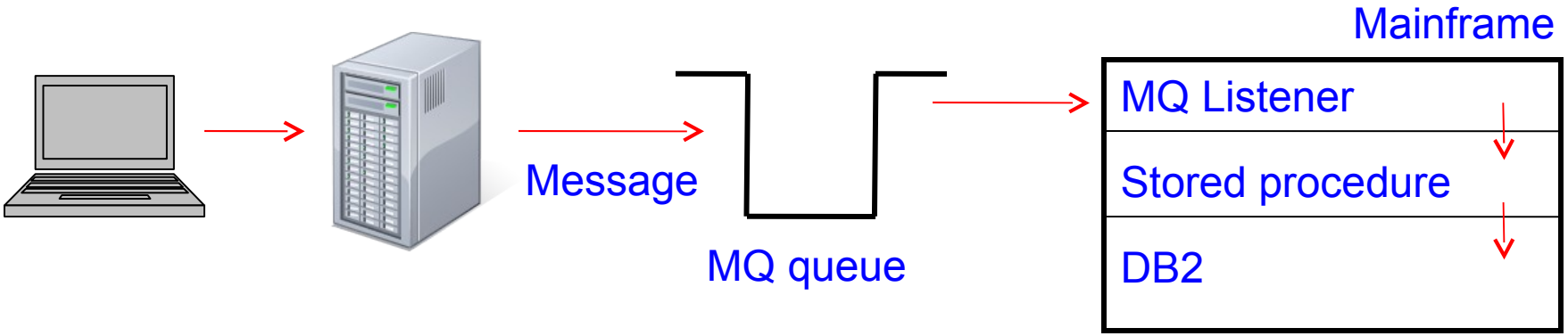
More on the vision diagram

- The DB2 server is a pure database server – there is no transaction management subsystem on the server
- This has been the standard set-up for some time in the distributed systems world
  - Mainframes with DB2 often have CICS or IMS, as well – usually because the organization ran a DB2-accessing transactional workload before DB2 stored procedure functionality was available
    - Static, server-side SQL (important for scalability) can be packaged in CICS or IMS transaction programs – or in DB2 stored procedures
    - Many mainframers believe that you have to have CICS or IMS/TM to support a high-volume DB2 for z/OS-based transactional workload

Not so

And a little more…

- MQ is a very important part of the picture
- How MQ and DB2 stored procedures can work together:
  - Client program puts a message (some information) on an MQ queue
  - A process called the MQ Listener can perform an action in response to a message arriving on a queue
    - Example: call a DB2 stored procedure, with message providing input to the stored procedure (MQ can also invoke a CICS transaction)

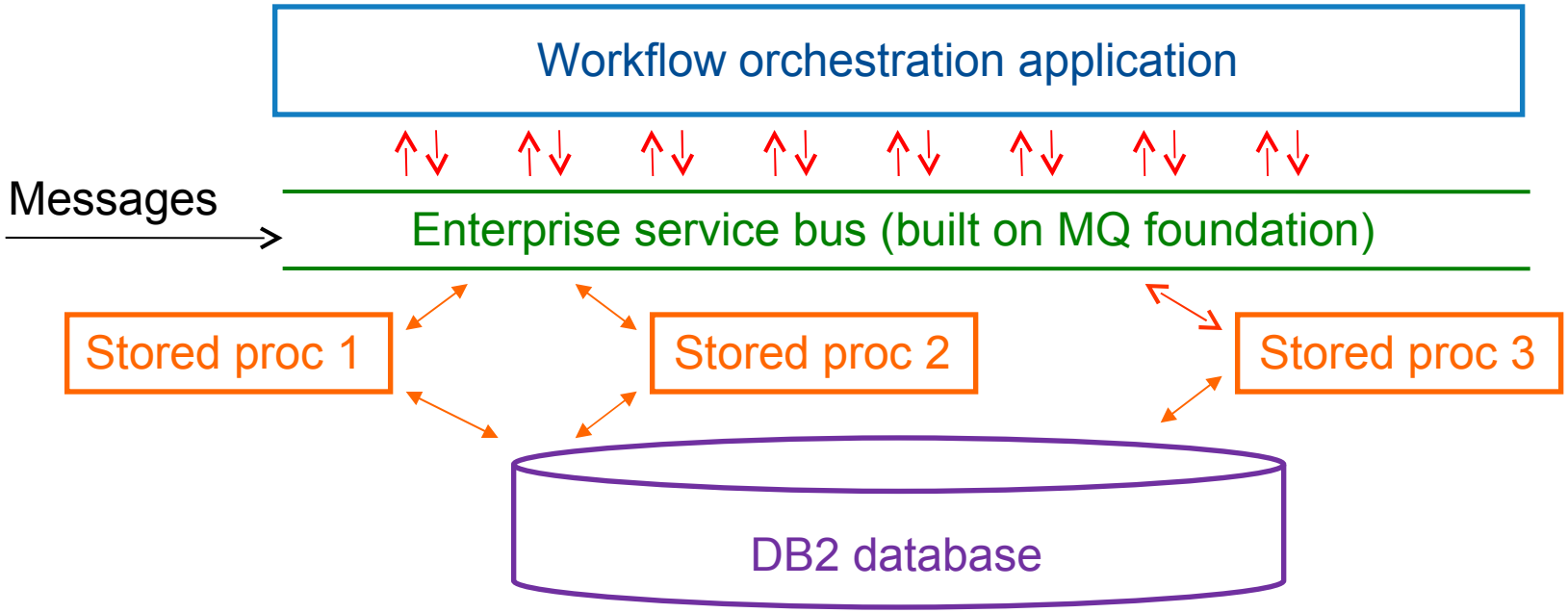


Great use of MQ and DB2 SPs (1)

- 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 immediately 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 (very 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

Great use of MQ and DB2 SPs (2)

- For automatic orchestration of complex business workflows involving multiple related transactions (e.g., order processing)
  - Workflow orchestration application ensures that required transactions are completed in the necessary order



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

V4: stored procedures introduced

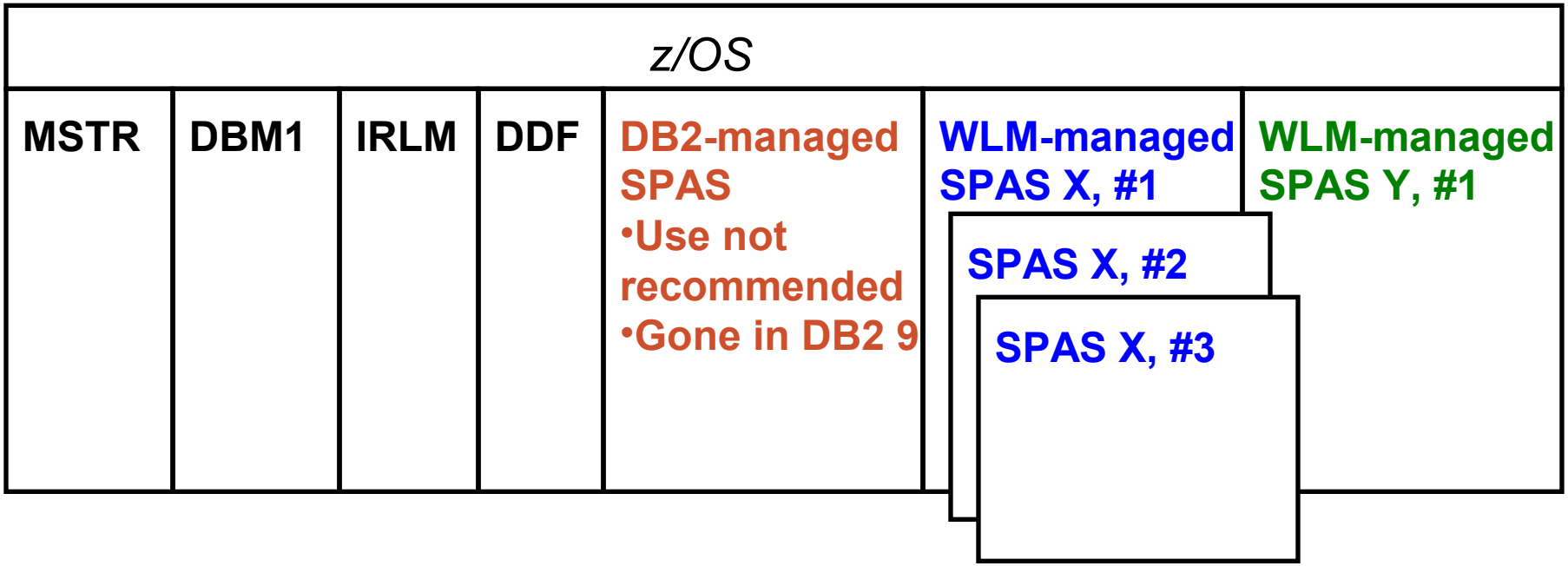
- New address space: DB2-managed stored procedure address space (aka SPAS)

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



More on WLM-managed SPASs

- You can have several of them
  - Useful for stored procedure management and monitoring
  - Maybe have a SPAS for newer stored procedure programs
  - Maybe group stored procedures by application
- WLM can automatically fire up additional instances of a given SPAS in response to workload demands
- Stored procedure two-phase commit capability via RRSAF
  - Recoverable resource services attach facility – required for stored procedure programs executing in a WLM-managed SPAS
  - Coordinated commit/rollback for update of DB2 & VSAM or DB2 & MQ
- Provide support for stored procedures written in Java
- Enable stored procedures to access LOBs (large objects)

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)
  - By the way: SYSPROCEDURES 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 DML – executes as an 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 (after being fixed, restarted via -START PROCEDURE command)
  - Higher abend limit might be appropriate for newer stored procedures
- Better synergy with z/OS Workload Manager
  - DB2, z/OS work together to optimize number of tasks in a SPAS (a stored procedure TCB is conceptually like a CICS-DB2 subtask TCB)
  - “Just right” number of tasks in a SPAS helps to improve 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

(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)
- A native SQL procedure (available beginning with DB2 9 in new function mode) is just a package – a “runtime structure” based on the SQL statements to be executed
  - A native SQL procedure runs 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

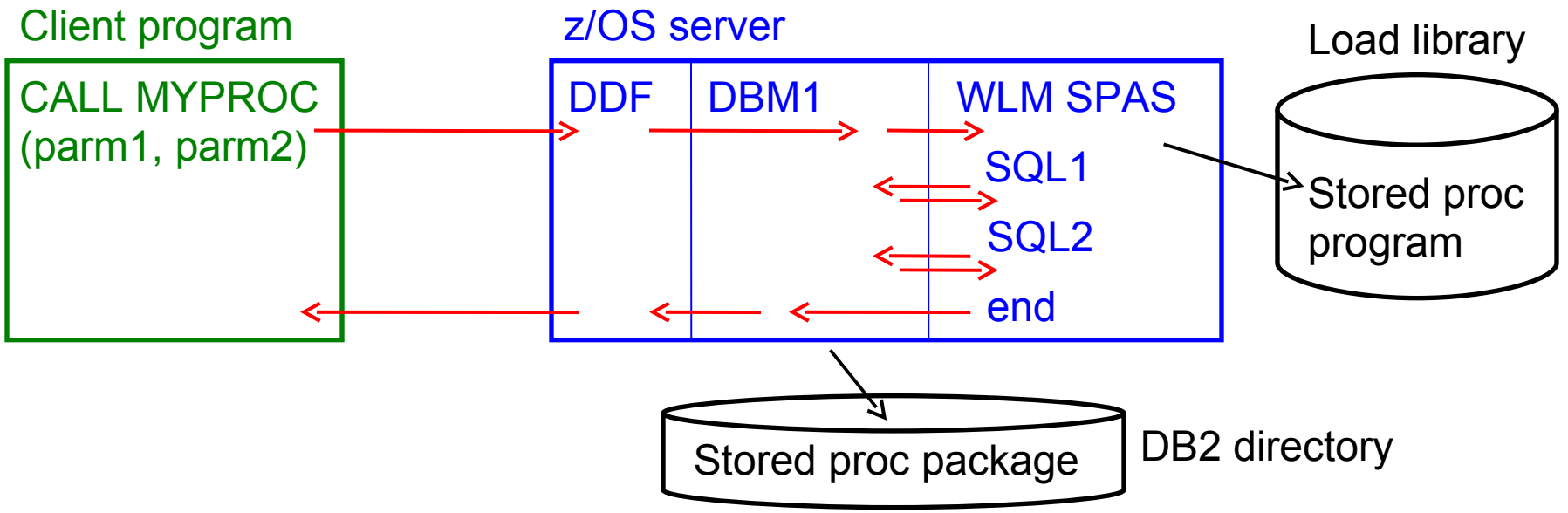
- A native SQL procedure runs under the calling program's task
  - No queuing, no delays related to thread-switching

Native SQL procedure efficiency (2)

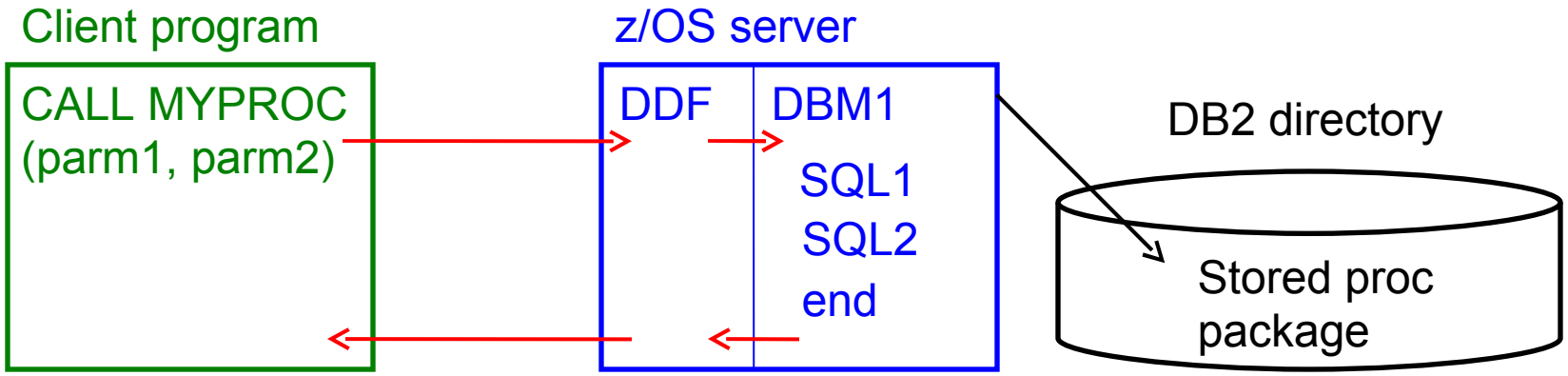
- For every SQL statement in an external stored procedure (or any other external-to-DB2, SQL-issuing program), an "addressability round trip" is required
  - Program's task switches addressability from "home" address space (for external SQL procedure, that's a WLM-managed stored procedure address space) 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 the stored procedure completes

External vs. native procedure

External



Native



The zIIP factor

- zIIP: specialty mainframe engine that costs less than a general-purpose processor and does not factor into 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 around 55% for native SQL procedures called through DDF

Some CPU figures

- From a presentation delivered by IBM's John Campbell at a recent IDUG DB2 Tech 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 couple of years ago, using DB2 9 – native SQL procedure picture probably looks even better now
  - **DB2 10 delivered improved SQL procedure language performance**

Enhanced functionality, too

- 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 compatibility across DB2 Family (DB2 for Linux, UNIX, and Windows already provided support for nested compound statements in SQL procedures) – important for cross-platform development

Native SQL procedure lifecycle

- Simpler creation, management, maintenance versus external stored procedures
  - No external-to-DB2 resources involved (e.g., no source / object / 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)
- Among other good things: no worries about program/package coordination
  - Native SQL procedures have a consistency token (used to match external-to-DB2 programs with corresponding package), but it's just a "synonym" for the procedure's version ID

Some go-forward recommendations

Getting from here to there…

- "There" being a situation in which you're getting maximum benefit from the use of DB2 stored procedures
- First: use stored procedures (if not already doing so)
- If you are in a DB2 V8 or DB2 9 CM environment, code and deploy some SQL stored procedures to gain familiarity with their development
  - 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
    - **DB2 10 allows user-defined functions (UDFs) to be written in SQLPL, too**

If you use CICS…

- Try making functionality of one or more CICS-DB2 transactions available to DRDA requesters via stored procedure calls
  - A more “open” way to expose the transaction functionality
- Could be done by converting COBOL CICS program to COBOL stored procedure (often involves little change), or replicating transaction functionality in SQL procedure
- 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, or
    - EXEC CICS (easier to code)

    (analogous to DB2 call attach versus TSO attach)

Running DB2 9 (NFM) or DB2 10?

- 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
- If you don't already have zIIP engines, consider getting some (or maybe adding to what you have)
  - Native SQL procedures could be a good driver of zIIP utilization on your system

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)
- As appropriate, choose native SQL procedures for new DB2 application development

Hints, tips, etc.

A common 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

Another common question

- "Multiple stored procedure address spaces boost availability (e.g., you can isolate new stored procedures in their own 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

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
    - If an external procedure: DB2 will check first to see if a 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
    - If native: DB2 will first check to see if the 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 one of the procedure's parameters
  - Solution: either use qualified names, or use a naming convention that clearly identifies parameters and variables (e.g. use p_ or v_ prefixes)

External-to-native conversion (3)

- 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
  - 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 SQL procedure, referencing the 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

Fetching result sets (1)

- An area in which there is a good bit of confusion
- YES, the caller of a stored procedure can fetch result set rows from a cursor declared and opened in the stored procedure
- Details can be found in the DB2 Application Programming and SQL Guide, but here are the basics:
  - The stored procedure is created with DYNAMIC RESULT SETS n, with “n” being the number of result sets that can be returned by the stored procedure
    - OK if “n” > actual number of cursors to be declared and opened by the stored procedure

Fetching result sets (2)

- The stored procedure declares one or more cursors WITH RETURN
  - If stored procedure created with COMMIT ON RETURN YES, cursor declaration must also specify WITH HOLD (this to prevent the cursor from being closed when stored procedure completes)
- The stored procedure opens the cursor, but does NOT fetch from the cursor
  - Any rows fetched by the stored procedure opening the cursor will not be available to the caller of the stored procedure

Fetching result sets (3)

- The calling program declares result set locator variables for stored procedure-opened cursors that will return rows to the caller
- The calling program calls the stored procedure (or procedures) that returns a result set (or sets)
- The calling program issues an ASSOCIATE LOCATORS statement for each stored procedure called that will return one or more result sets
  - If called stored procedure XYZ will open several cursors, the ASSOCIATE LOCATORS statement for XYZ in the calling program will include a result set locator variable for each cursor
    - Example: ASSOCIATE LOCATORS (:LOC1, :LOC2) WITH XYZ

Fetching result sets (4)

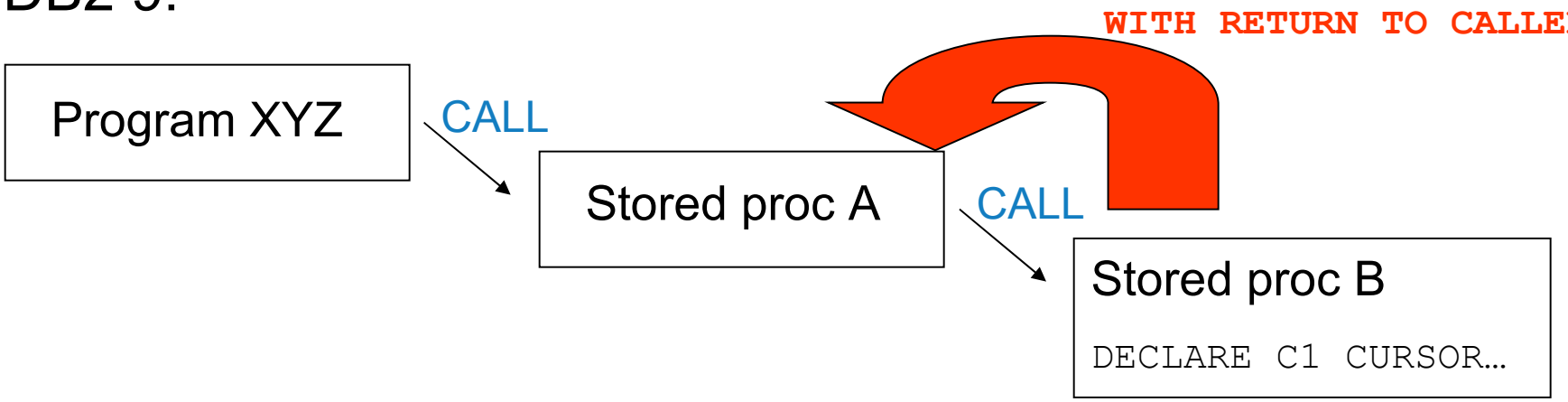
- The calling program issues an ALLOCATE CURSOR statement for each result set locator variable named in an ASSOCIATE LOCATORS statement
  - No need for the cursor name in an ALLOCATE CURSOR statement to match the name of the cursor declared in the called stored procedure
- The calling program fetches rows from the result sets, using the cursors named in the ALLOCATE CURSOR statements

DB2 10: RETURN TO CLIENT cursors

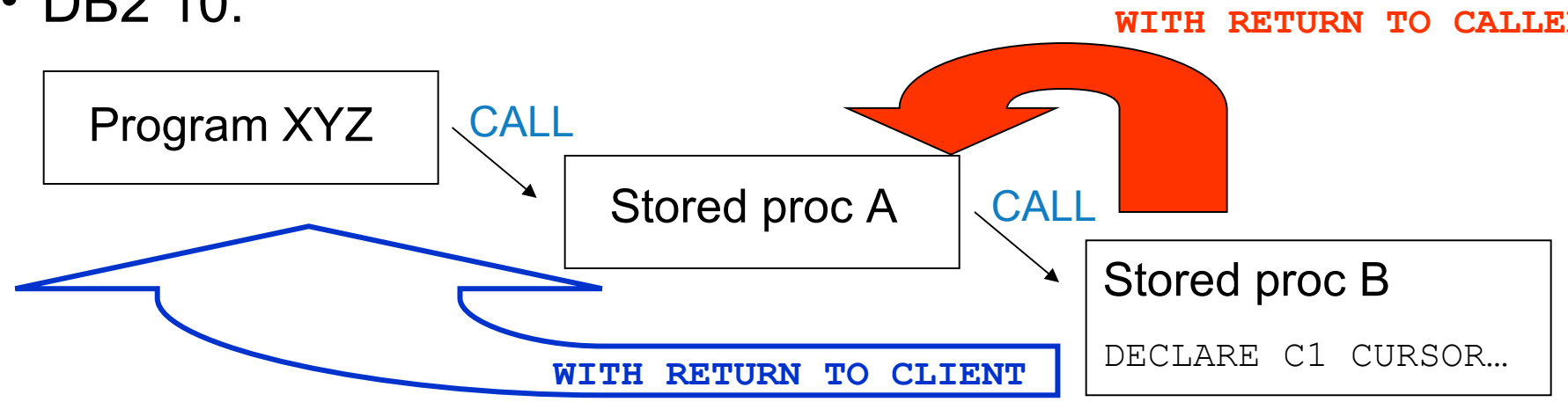
- DB2 9: stored procedure can return result set one level up in chain of nested procedures (WITH RETURN TO CALLER)
  - For example, if program PROG_A calls stored procedure PROC_B, and PROC_B calls PROC_C, PROC_B can fetch from a cursor declared and opened in PROC_C, but PROG_A cannot
    - If PROG_A 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 (one that calls first stored procedure, which then calls another stored procedure) can fetch rows, but the 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