F16 Migrating Stored Procedures to DB2 Universal Database from Sybase, Oracle, and Informix

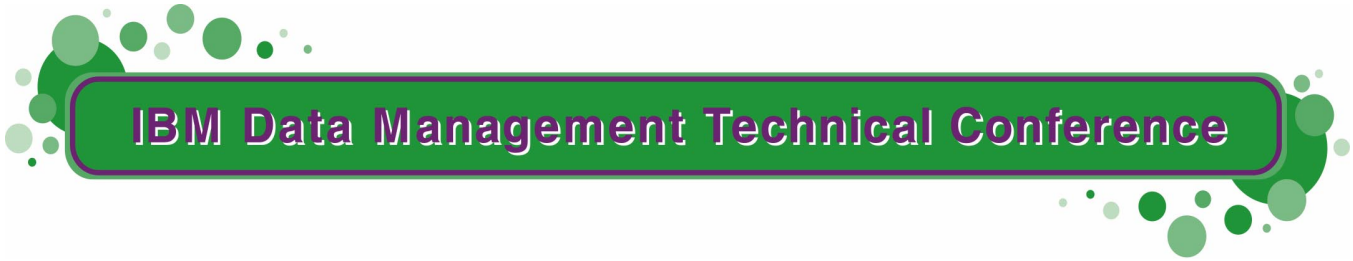
William O'Connell, Senior Technical Staff member, IBM Toronto Lab

DB2 family supports standardized SQL Control Statements with a subset of the Persistent Stored Modules specification of SQL 99 ANSI/ISO standard. Extending across the whole DB2 family, it is functionally comparable to Sybase, Microsoft's TSQL and Oracle's PL/SQL. Migrating stored procedures from these platforms is now easier than ever before. This session provides an overview of the SQL language extensions and demonstrates features that significantly facilitate migration from Sybase, Microsoft SQL Server, Informix and Oracle; for example, temporary tables, application savepoints, identities, sequences, results sets and nested procedure calls. Finally, the latest enhancements to DB2 SQL Procedures will be discussed. The session will focus on SQL, and not the tools that exist to automate the SQL conversion.

Session F16

Migrating Stored Procedures to DB2 Universal Database from Sybase, Oracle, and Informix

William O'Connell, IBM Toronto Lab, boconnel@ca.ibm.com



Anaheim, CA **Sept 9 - 13, 2002**

RETURN TO INDEX

Agenda

- SQL PL Overview
- Different ways to create and run SQL PL stored procedures
  - Stored Procedure Builder (SPB)
  - Command Line Processor (CLP)
- Related features
  - Temporary tables
  - Application savepoints
  - Identity Columns / Sequences
  - Nested procedure calls (A calls B calls C ...)
- Enhancements on V8 - Both performance and functionality
- Architecture / System Requirements /
- Pointer to migration tool website
- Summary

SQL Procedures

- What are they ?
  - For Those Coming from DB2: SP in which procedural logic is contained in CREATE PROCEDURE statement
  - For Those Adopting DB2: Similar to TSQL, PL/SQL, SPL
- Simple, high-level language (SQL PL = SQL + Control)
- Powerful condition-handling model
- Can contain Static and Dynamic SQL
- Parameter passing: IN, OUT, INOUT
- Subset of SQL/PSM (SQL99 ANSI/ISO standard)
- Extensions to standard: GOTO, RETURN, Result Sets, /*...*/
- Text and binary stored in the catalogs (backup, restore)
- Supported across DB2 UDB family (UNIX, NT, AS400, OS390)

Language Features

- Translated to C, compiled and installed transparently
- Text and binary stored in the catalogs (backup, rollback)
- Nested Stored Procedures
- Temporary Tables
- Identities (also Sequences)
- Application Savepoints
- Result Sets
- Static and dynamic SQL
- Powerful condition handling model
- Extensions to facilitate Sybase migration (RETURN, GOTO, C-style comments)

- Migration from ***rowid,*** use one or more of generated columns, identities, sequences, generate_unique, or rownumber

SQL PL: Procedure Structure

```
CREATE PROCEDURE <name> (<parameters>)
LANGUAGE SQL
<statement>
```

Any statement

- Example: multiply by 2

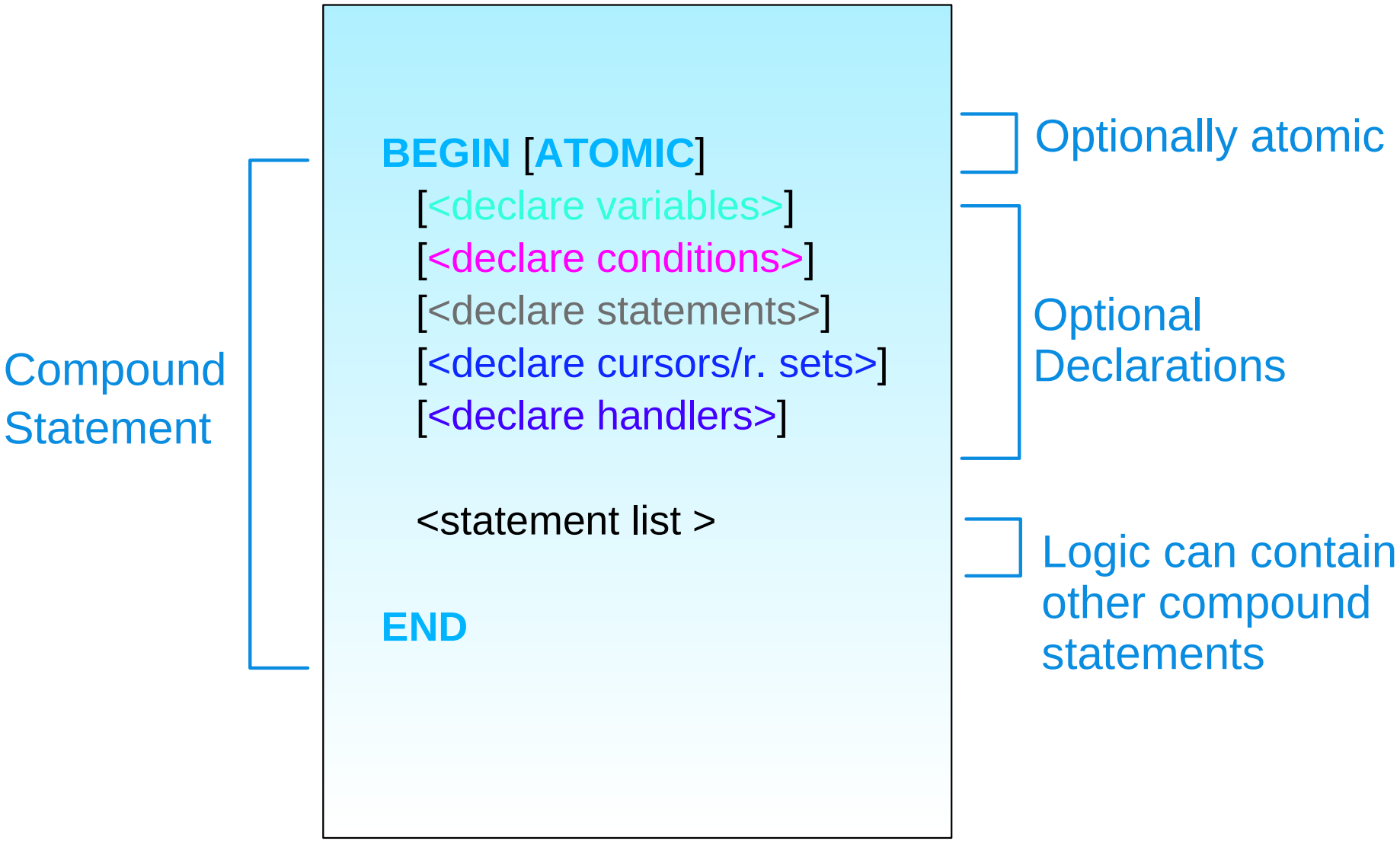
```
CREATE PROCEDURE times2(IN i INT, OUT o INT)
LANGUAGE SQL      /* multiply by 2 */
SET o = i * 2
```

SQL PL: Statements

- Assigment (SET a = b)
- Compound statement (BEGIN ... END)
- Control flow (IF, WHILE, CASE, FOR, GOTO, RETURN, etc.)
- Condition Handling (CONTINUE, EXIT, UNDO, SIGNAL)
- SQL DML (SELECT, DELETE, INSERT, UPDATE)
- SQL DDL ([CREATE | DROP] [TABLE | INDEX | VIEW])
- Dynamic SQL (PREPARE, EXECUTE)
- Cursor Manipulation (OPEN, FETCH, CLOSE, R. SETS)
- Transactions (COMMIT, ROLLBACK, SAVEPOINT)

NOTE: see SQL Ref for complete list

SQL PL: Compound Statement



SQL PL: Scoping

```
CREATE PROCEDURE P1 () LANGUAGE SQL
lab1: BEGIN
        DECLARE a INT DEFAULT 100;
      BEGIN
        DECLARE a INT DEFAULT NULL;

        SET a = a + lab1.a;
        UPDATE T1 SET b = b * 1.1
                  WHERE b = a; -- WARNING (who is 'a' ?)
      END;
  END lab1;
```

- NOTE: scoping also applies to cursors, conditions, dyn. stmts.

SQL PL: First Example (from CLP)

- Increase the salary of all managers by a given percentage.
- Return the salary of the highest paid employee.
- Return a result set with all the employees that make at least 50K
- Create and call the procedure from the SP Builder or CLP

SQL PL: Example 1

```
CREATE PROCEDURE P1 (IN raise INT, OUT top DECIMAL(9,2))
LANGUAGE SQL
BEGIN
    DECLARE cur1 CURSOR WITH RETURN TO CLIENT FOR
                    SELECT * FROM employee
                    WHERE salary >= 50000;

    UPDATE employee SET salary = salary * (1+ raise/100.0)
                        WHERE job = 'MANAGER';

    SET top = (SELECT max(salary) FROM employee);
    OPEN cur1;
END
```

SQL PL: Conditions

- Conditions Express the Degree of Success in the Execution of SQL Statements
- Possible Conditions:
  - Success: SQLSTATE '00000'
  - Not Found: SQLSTATE '02XXX'
  - Warning: SQLSTATE '01XXX'
  - Exception: Any other SQLSTATE value
- Handlers Called Implicitly When Condition in not Success
  - Not Found and Warning are Tolerated
    - If not handled, execution continues to next statement
  - But Exceptions Must be Handled !!!
    - If not, Control Returns to the Caller

SQL PL: Condition Handling Example

- 3 Parts: Resumption point, Handled Condition(s), Action(s)

```
BEGIN
   DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      INSERT INTO LogTable VALUES ('Exception', SQLSTATE);

   DECLARE EXIT HANDLER FOR SQLSTATE '17424'  BEGIN
      DELETE FROM MyTable WHERE col1 = ID;
      SET Out1 = 'failed';
   END;

   <Handlers will be implicitly called if exception occurs here>
END;
```

SQL PL: Conditions (Cont.)

- Handlers Separate Error Handling from Business Logic
- Conditions can be raised explicitly: SIGNAL SQLSTATE '13344'
- Can Simulate "Traditional" Error Handling:

```
BEGIN
   DECLARE CONTINUE HANDLER for SQLEXCEPTION
      SET Saved_SQLSTATE = SQLSTATE;
   . . .
   IF(Saved_SQLSTATE <> '0000')THEN
       <code to handle the condition>
   END IF;
END;
```

Temporary Tables

- "Lightweight" tables. Fast due to:
  - ✓ No Catalog Overhead
  - ✓ No Locks
  - ✓ No logging (though logging support is available on V8)
- Life span: since creation to connection close
- Can Inherit Column Definitions from table or queries
- N connections => N instances (like progr. variables)
- Schema is always SESSION (e.g.: SESSION.tab1)
- Creator is granted all privileges
- Unlike perm. tables, can reference them in same session
- Similar to Sybase's # temp tables

Temporary Tables (cont.)

- Example:

```
DECLARE GLOBAL TEMPORARY TABLE T1(c1 INT)
    -- LIKE REAL_T1
    ON COMMIT PRESERVE ROWS -- Default is to delete them
    NOT LOGGED
    IN USR_TEMP_TS;

INSERT INTO SESSION.T1
        (SELECT * FROM REAL_T1 WHERE C1 = 100);
```

SQL PL: Second Example

- Call P1 (first example) from another procedure
- Create a temporary table as a "log file"
- Process the result set returned by P1
- Return another result set consisting of the contents of the temp table we used as "log file"

```
DECLARE GLOBAL TEMPORARY
    TABLE tt (msg CHAR(30)) NOT LOGGED; -- default delete rows
```

SQL PL: Example 2 (a)

```
CREATE PROCEDURE P2 (IN raise INT, OUT top DECIMAL(9,2), OUT total DECIMAL(9,2))
LANGUAGE SQL
BEGIN
    DECLARE SQLCODE int;
    DECLARE empno CHAR(5);
    DECLARE salary, total DECIMAL(9, 2) DEFAULT 0;
    DECLARE cur1 CURSOR WITH RETURN TO CALLER FOR SELECT * FROM
SESSION.tt;

    INSERT INTO SESSION.tt VALUES('Before Call');
    CALL P1(raise, top);
    ASSOCIATE RESULT SET loc WITH PROCEDURE P1;
    ALLOCATE cur2 CURSOR FOR RESULT SET loc;
    WHILE (SQLCODE = 0) DO
        SET total = total + salary;
        INSERT INTO SESSION.tt VALUES('Before Fetch');
        FETCH cur2 INTO empno, salary;
    END WHILE;
    INSERT INTO SESSION.tt VALUES('Before Open');
    SET P2.total = total;
    OPEN cur1;
END
```

SQL PL: Example 2 (b)

```
CREATE PROCEDURE P1 (IN raise INT, OUT top DECIMAL(9,2))
LANGUAGE SQL
BEGIN
    DECLARE cur1 CURSOR WITH RETURN TO CALLER FOR
                    SELECT empno, salary FROM employee
                    WHERE salary > 50000;
    INSERT INTO SESSION.tt VALUES('Before Update');
    UPDATE employee SET salary = salary * (1+ 1/raise)
                        WHERE job = 'MANAGER';
    INSERT INTO SESSION.tt VALUES('Before Set');
    SET top = (SELECT max(salary) FROM employee);
    INSERT INTO SESSION.tt VALUES('Before Open');
    OPEN cur1;

END
```

Identity Columns

- Fast, automatic generation of primary key
  - ✓ Avoids contention due to application-generated identity
  - ✓ Uncommited transaction that updated counter does not block other transactions (=> "gaps" can appear among values)
- User can specify type, increment, initial value
- Function identity_val_local() allows access to last identity value generated

- Identities are used for Sybase/SQL Server migration;
- Sequences are used for Oracle migration

Identity Columns: Example

```
DECLARE id INT;
. . .
CREATE TABLE MyTable (
   COL1 INT NOT NULL GENERATED ALWAYS AS IDENTITY
                (START WITH 100, INCREMENT BY 5),
   COL2 DOUBLE,
   COL3 INT
);

INSERT INTO MyTable (COL2, COL3) VALUES (5.6, 3)

SET id = identity_val_local();
```

Application Savepoints

- Control atomicity of a sequence of db operations
- If error, can rollback to point where savepoint was taken
- Savepoint-handling Statements:

```
SAVEPOINT sp
    [UNIQUE]
    ON ROLLBACK RETAIN CURSORS
    [ON ROLLBACK RETAIN LOCKS]

RELEASE SAVEPOINT sp

ROLLBACK TO SAVEPOINT sp
```

Application Savepoints: Example

```
SAVEPOINT S1 ON ROLLBACK RETAIN CURSORS;
INSERT INTO MYTAB (C1, C2) VALUES (2,3);
INSERT INTO MYTAB (C1, C2) VALUES (2,1);
INSERT INTO MYTAB (C1, C2) VALUES (245,5);
ROLLBACK TO SAVEPOINT S1;

SAVEPOINT S1 ON ROLLBACK RETAIN CURSORS;
INSERT INTO MYTAB (C1, C2) VALUES (2,300);
INSERT INTO MYTAB (C1, C2) VALUES (200,3);
COMMIT;
```

Contents of MYTAB:

C1	C2
2	300
200	3

Sybase Migration (Result Sets)

- **Sybase: a SELECT statement without INTO clause is a result set**
- **DB2 UDB: a CURSOR WITH RETURN is a result set**

```
CREATE PROCEDURE PROC1
AS
BEGIN
  SELECT empno, empname
  FROM emp
  WHERE empno BETWEEN 100 and 200

  SELECT name, address
  FROM staff
END
```

```
CREATE PROCEDURE PROC1()
LANGUAGE SQL
BEGIN
  DECLARE c1 CURSOR WITH RETURN FOR
     SELECT empno, empname
     FROM emp
     WHERE empno BETWEEN 100 and 200;
  DECLARE c2 CURSOR WITH RETURN FOR
    SELECT name, address
    FROM staff;

  OPEN c1;
  OPEN c2;
END
```

Sybase Migration (Temp. Tables)

- **Sybase**
  - **local temp table name has # sign as first char**
  - **global temp table name has ## as first char**
  - **TSQL programmers generally use local temps**

  Example:

```
SELECT * INTO #temptab FROM tab1 WHERE ...
```

- **DB2 UDB**
  - **The semantics of our temp tables is very close to #temptable**

  Example:

```
DECLARE GLOBAL TEMPORARY TABLE temptab ...
INSERT INTO SESSION.temptab
        SELECT * FROM tab1 WHERE ...
```

Sybase Migration (FROM in UPDATE)

- **Sybase**
  - **Can Join tables in UPDATE statement**

Example:

```
UPDATE Big
SET Big.C1 = Small.C1
FROM Big, Small
WHERE Big.Key = Small.Key
```

- **DB2 UDB**
  - **Must use correlated subqueries**

Example:

```
UPDATE Big
 SET Big.C1 = (SELECT Small.C1
                FROM Small
                WHERE Big.Key = Small.Key)
 WHERE EXISTS (SELECT 1
                 FROM Small
                 WHERE Big.Key = Small.Key)
```

Sybase Migration (Error Handling)

- Sybase: execution continues after error
- DB2 UDB: handler is implicitly executed (No handler => return)

```
CREATE PROCEDURE PROC1
AS
BEGIN
 DELETE  FROM emp
 WHERE empno
       BETWEEN 100 and 200
 IF (@@error = 45)
   INSERT ...

 UPDATE staff
       SET salary = salary * 1.25
 IF (@@error <> 0)
   RETURN -1
END
```

```
CREATE PROCEDURE PROC1()
LANGUAGE SQL
BEGIN
 DECLARE SQLCODE, MyCode INT;
 DECLARE CONTINUE HANDLER FOR
                  SQLEXCEPTION,
                  SET MyCode = SQLCODE;

 DELETE  FROM emp
 WHERE empno BETWEEN 100 and 200;
 IF (MyCode = -147 ) THEN
   INSERT . . .

 UPDATE staff SET salary = salary * 1.25;
 IF (MyCode = -147 ) THEN RETURN -1; END IF;
END
```

Oracle Migration (Defaults for parms)

```
CREATE PROCEDURE PROC1
(acc_in IN NUMBER DEFAULT 0,
total OUT NUMBER DEFAULT -1)
AS
BEGIN
  . . .
END
```

► To call:

```
PROC1(total => 100);
```

```
CREATE PROCEDURE PROC1
(IN acc_in DECIMAL(19, 2),
OUT total DECIMAL(19, 2))
LANGUAGE SQL
BEGIN
  IF (acc_in IS NULL) THEN
    SET acc_in = 0;
  END IF;
 . . .

  IF (total IS NULL) THEN
    SET total = -1;
  END IF;
END
```

► To call:

```
CALL PROC1(NULL, 100);
```

Other useful functionality

- Native Debugger
  - Breakpoints, Variable Watch, Call Stack
  - Step-in, Step-out, run to line
- Distribution of compiled SQL Procs
  - Compile on one machine, deploy to many
  - No C compiler needed in target machines
  - Must be same platform

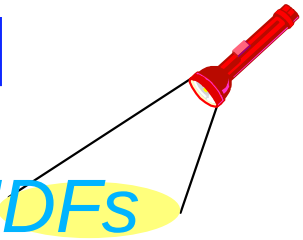
Performance Enhancements on V8

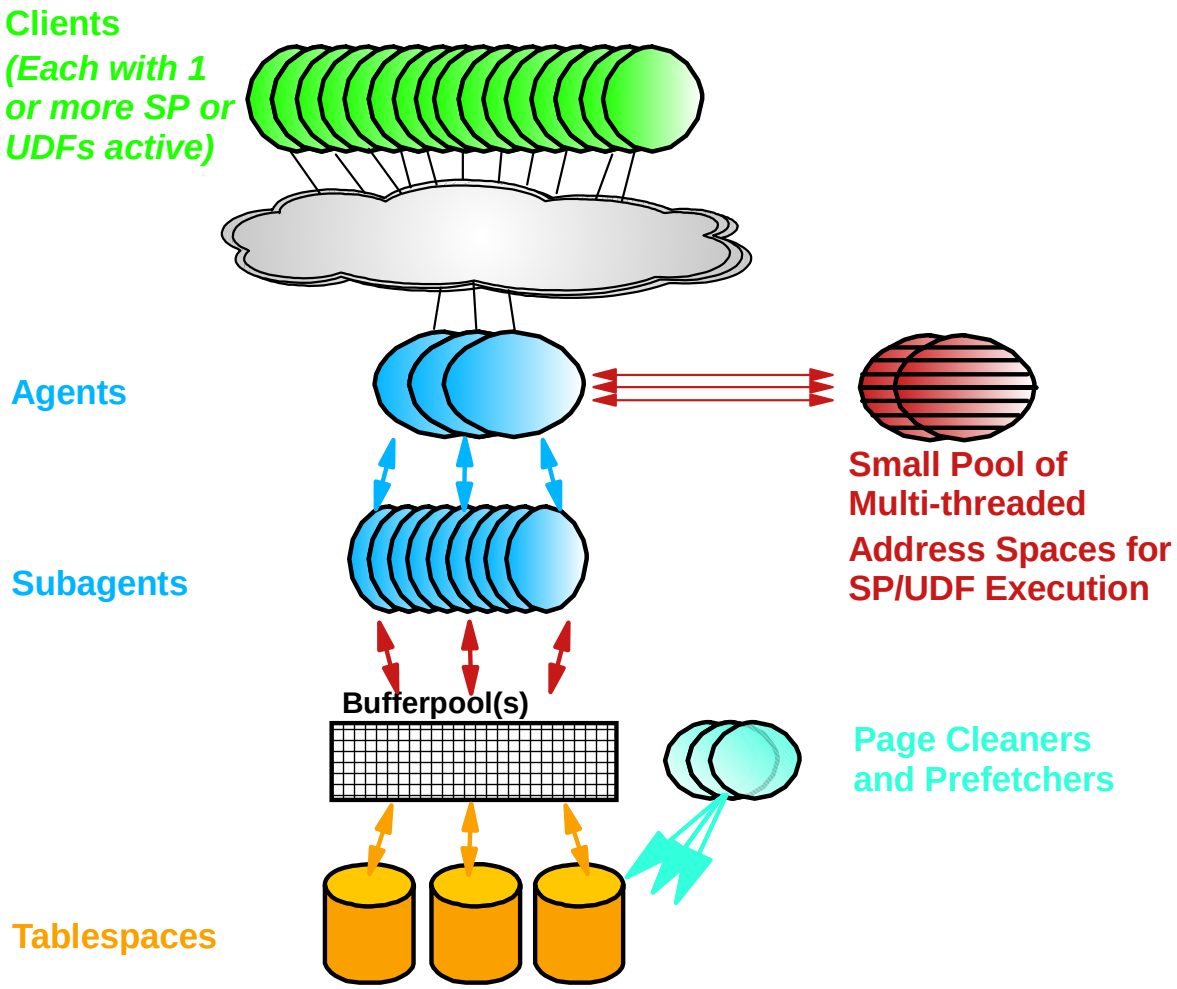
- General performance infrastructure
  - CALL statement is a compilable statement
    - Dynamically preparable
    - Section reuse
  - More efficient catalog lookups
  - Reduced catalog locking
- Process model change
  - Reduced memory footprint
  - More efficient invocation and library caching
  - SQL PL stored procedures run in trusted mode
- Trusted Result sets and recursion

Internal results show V8 approx 25% faster than V7

Stored Procedure Process Model

Threaded Architecture for Stored Procedures & UDFs





✓ ***Reduces Server Memory Consumption***

- *Especially significant for Java routines (limits number of JVMs)*

✓ ***Routines must be THREADSAFE to exploit***

- *Existing non-Java routines migrated as non-threadsafe*
- *Existing Java routines migrated as threadsafe*

```
CREATE PROCEDURE foo ...
[NOT] THREADSAFE ....

CREATE FUNCTION foo ...
[NOT] THREADSAFE ....
```

✓ ***SQL PL Stored Procs now run in Agent Address Space***

- *Inherent memory protection*
- *Major performance boost!*

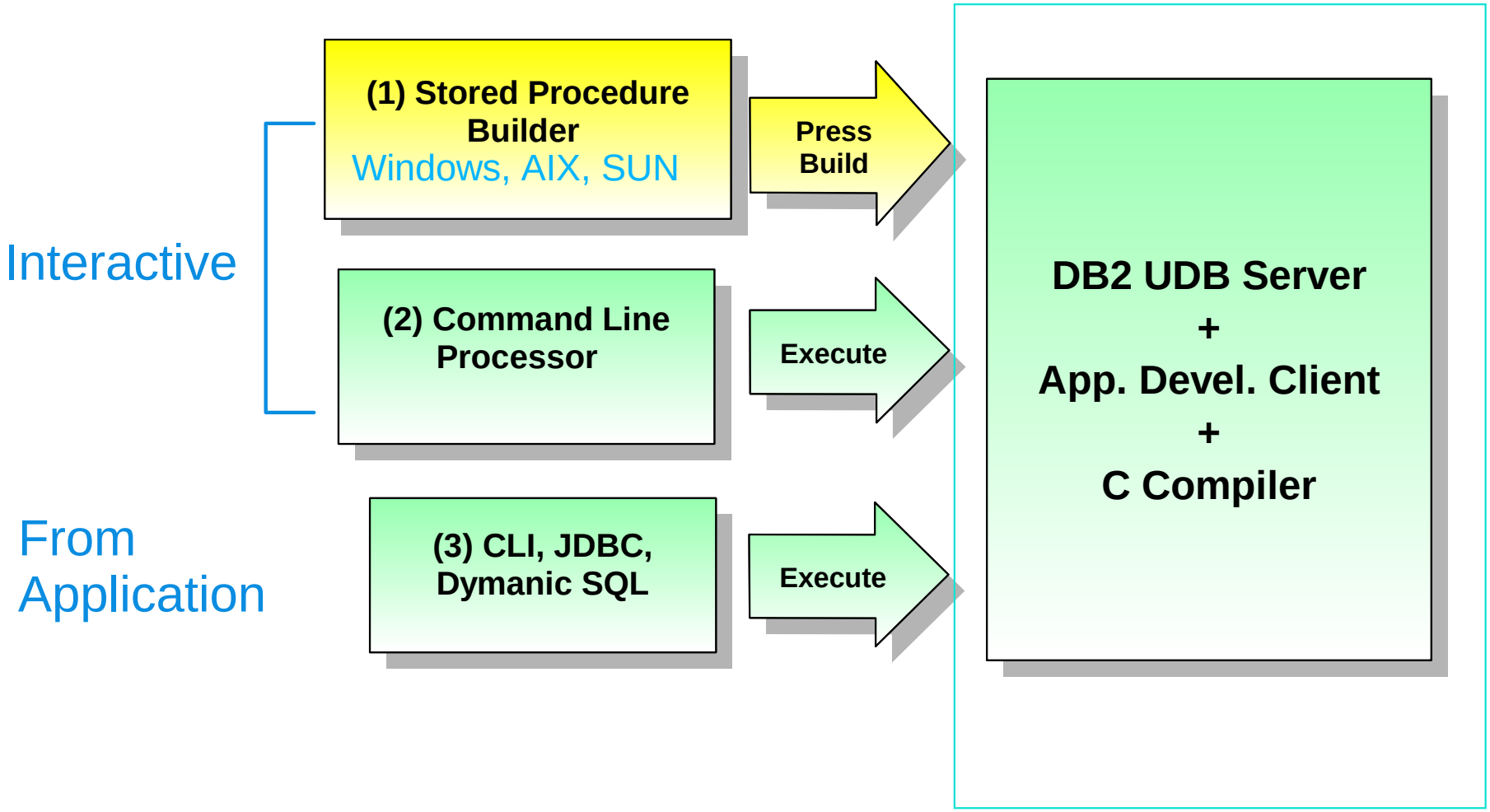
Declared Global Temporary Table Enhancements on V8

- Minimal undo logging
  - NOT LOGGED clause manditory in V7, and now optional in V8
  - Supports the rollback of data changes to DGTT
- Index support
  - Any standard index can be created on a temporary table
- RUNSTATS supported against the table

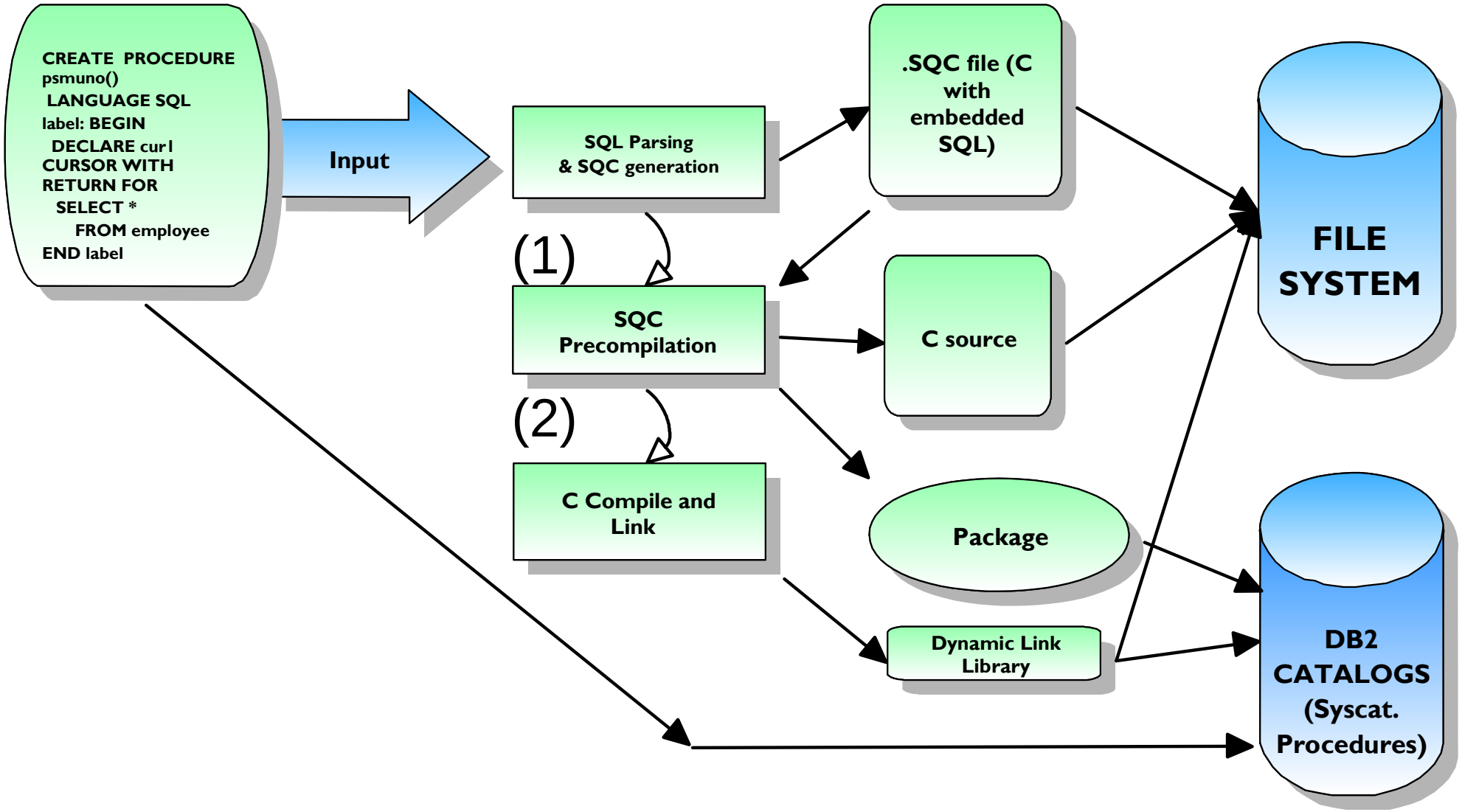
Other Enhancements on V8

- Call statement can take complex expressions as parameters
- Parameter limit increased to 32K; V7 has 90 parameter limit
- Parameters and SQL PL variables can be based on DISTINCT types
- No restriction on nested stored procedure calls
  - For example, Java can call SQL PL, and vice versa
- CALL statement is dynamically prepareable
- Full ISO compliance on signal handlers
  - Nested signal handler support
- Subset of SQL PL can be defined in UDFs and Triggers

Procedure Creation



Architecture (DB2 for UWO)



System Requirements

- C Compiler and Application Development Client installed on the SERVER
- All platforms supported, except Win95/98
- Same compilers as for the SDK (aka ADC)
- One compiler set by default for each platform
- E.g.: MSVC is the default for NT
- Other compilers (e.g. gcc) might work too. (But risky if not supported)
- See documentation for setup details by platform
  - UNIX usually works "out of the box"
  - NT requires C compiler configuration

Migration Tools

Websites:

http://www-4.ibm.com/software/data/db2/migration/

http://www.ibm.com/solutions/softwaremigration

More Information

- DB2 SQL Reference, App. Development Guide and App. Building Guide
- Jim Melton's "Understanding SQL/PSM", Morgan Kaufmann, 1998
- DB2 Migration Home Page: http://www-4.ibm.com/software/data/db2/migration/
  - ✓ White Papers for migration from Oracle, Sybase, Informix, SQL Server
  - ✓ Red Books (SG24-5485-00)
  - ✓ Etc...

Summary

- Overviewed SQL PL Stored Procedure Language
- Two ways to create and run
  - ✓ Stored Procedure Builder (SPB)
  - ✓ Command Line Processor (CLP)
- Discussed related features
  - ✓ Temporary tables
  - ✓ Application savepoints
  - ✓ Identity Columns
  - ✓ Nested procedure calls (A calls B calls C ...)
  - ✓ Native Debugger
  - ✓ Distribution of compiled SQL Procs
- Overviewed V8 enhancements
- Architecture / System Requirements
- Migration issues