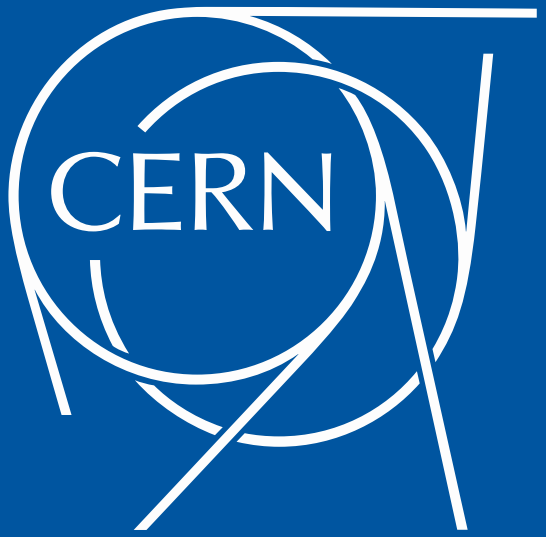

CERN

Stephan Petit – GS/ASE-EDS

Oracle Tutorials
PL/SQL Best Practices

PL/SQL Best Practices - Agenda

- Why best practices ?
- The Black Box Paradigm
- Coding conventions
  - Why coding conventions ?
  - One set of coding conventions that works
- Error handling
  - Trapping
  - Reporting
  - Recovering
  - Summary
- References

Why Best Practices ?

Why Best Practices ?

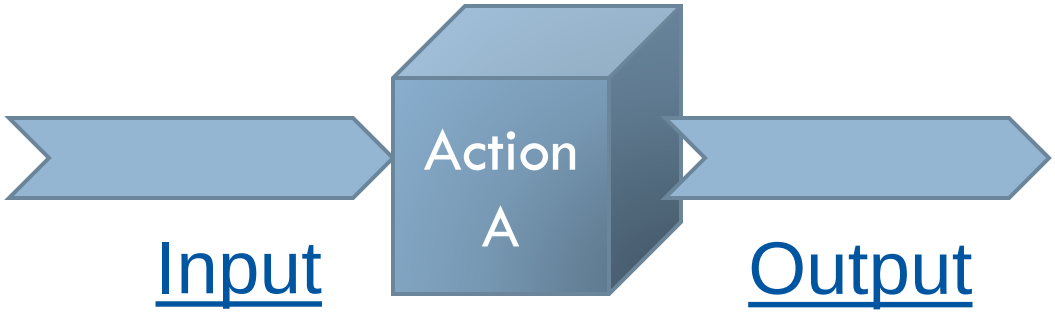
- Proven efficiency
- Tuned over the years
- Shared by many
- Bring more efficiency in coding/maintenance
- Avoid common mistakes

The Black Box Paradigm

The Black Box Paradigm

- Not dedicated to PL/SQL
- It works very well with PL/SQL !
- A Black Box:
  - Performs a well identified action
    - Autonomously : all steps from a to z
    - Uniquely : two different boxes cannot do the same action
  - Has a clear list of input necessary for its action
  - Always checks the input
  - Checks the action is allowed
  - Always returns an output (status/data)

The Black Box Paradigm



Input

- mandatory parameters
- optional parameters
- user id (action allowed ?)

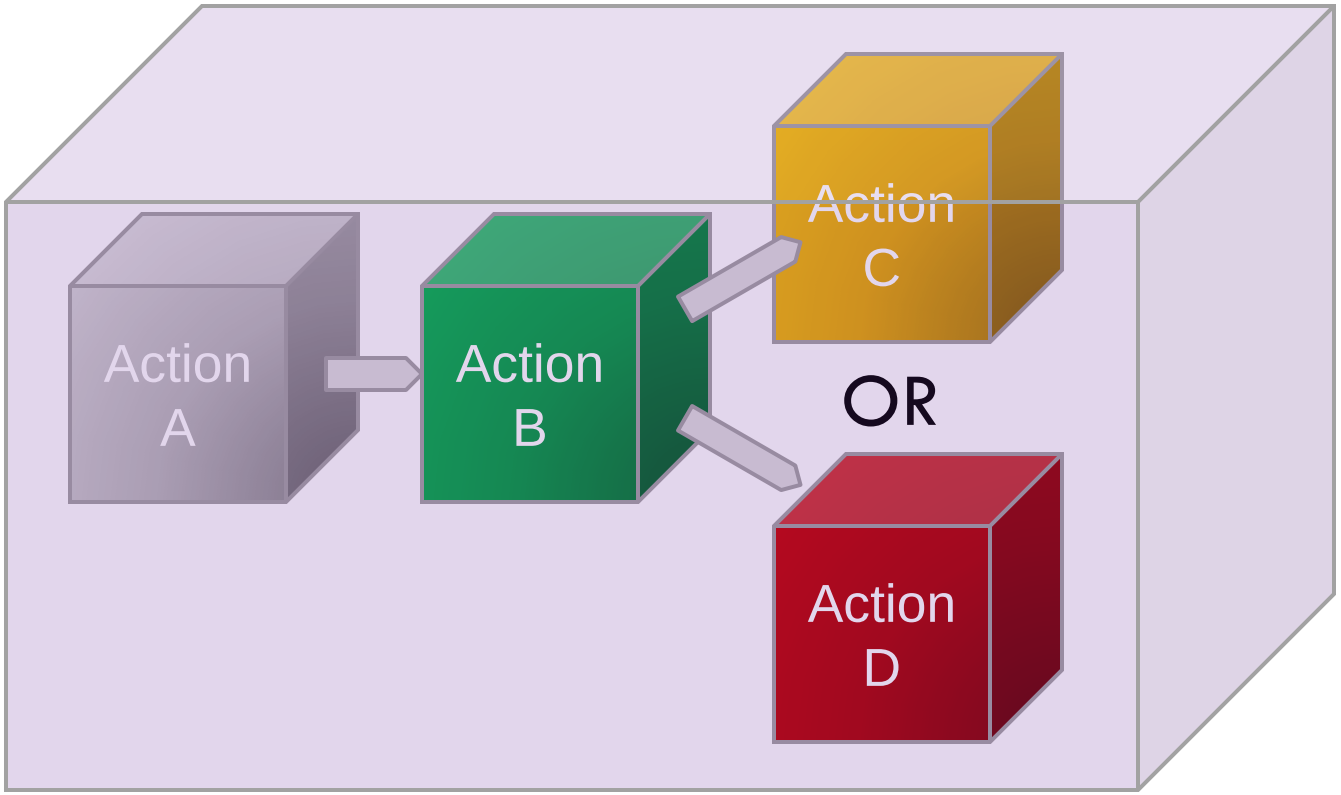
Output

- status (done/warning/error)
- data (fetched/computed)
- user interface

The Black Box Paradigm

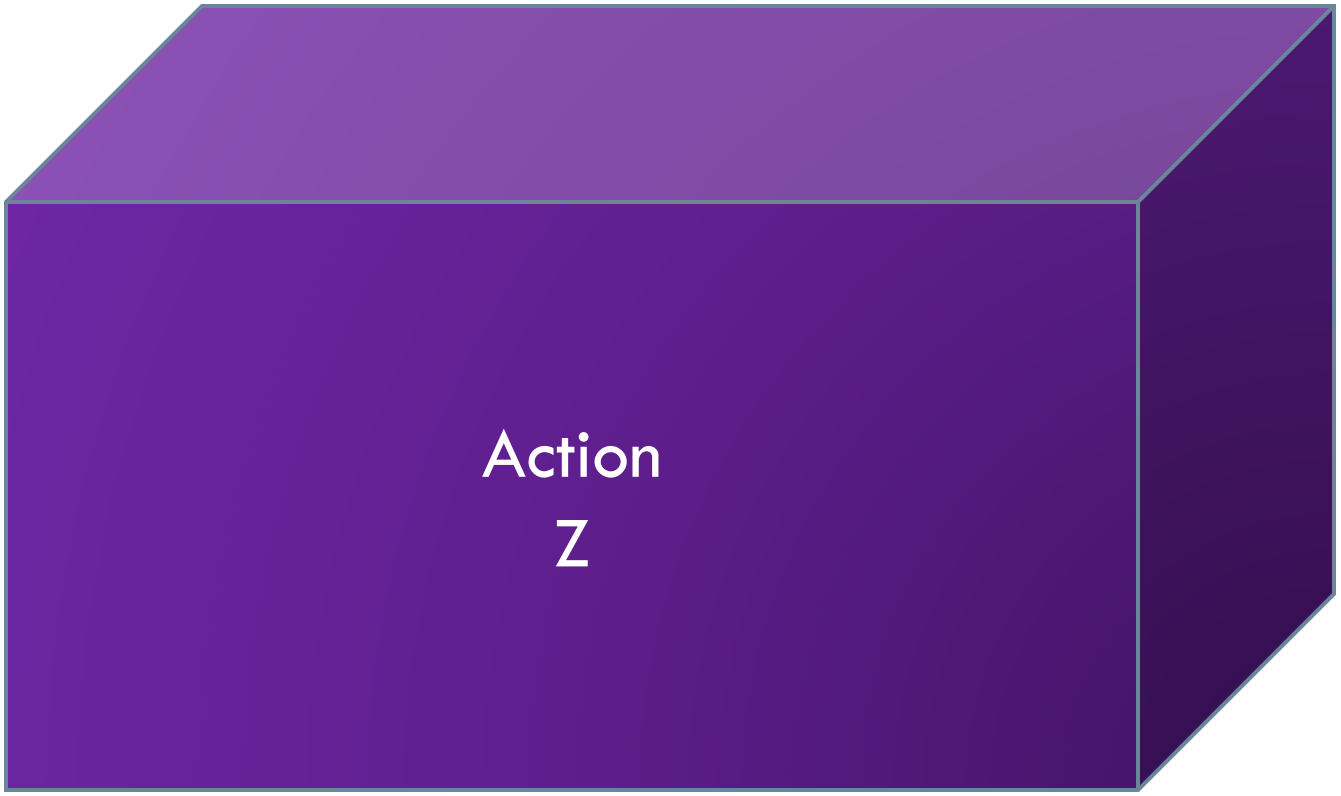
How it works inside is not your business !
The input and output are for you,
the rest is the job of the box

The Black Box Paradigm



Boxes can be combined to implement more complex actions

The Black Box Paradigm



Leading to new black boxes ! Etc…

PL/SQL & Black Boxes

- Being involved in programming, you will:
  - Use existing black boxes
  - Write new black boxes
- In both cases:
  - You want to be able to trust them
  - You want to be able to understand then, maintain or debug them, even years after they were created
- Consider every single PL/SQL procedure or function as a black box

PL/SQL Modules Classification

- It is advisable to classify PL/SQL modules according to the type of actions they perform
  - Better modularity
  - Better reusability
  - Prevents from code duplication

- In the end: powerful library of modules, like with OO approach

PL/SQL Modules Classification

- One efficient PL/SQL modules classification:
  - Data action (insert, update, delete)
  - Data fecthing / computation — *Kernel*
  - Data checking / authorization
  - User interface display — *Interface*

PL/SQL Module Grouping

- It is advisable to group PL/SQL modules by themes
  - Easier to find the module you need
  - Use **packages**
    - Example: one package for data checkers, one for authorization checkers etc…

Coding Conventions

Why Coding Conventions ?

- Not slow understanding
  - The code becomes more easy to read
  - It is faster to see and understand with your eyes what the structure of a piece of code is, hence quickly understanding how it works
  - Better for using the same code among several people in the same team of even coming from various different teams
  - Higher rate of 1/amount of misunderstanding

Why Coding Conventions ?

- **Quick understanding**
  - Code is easy to read
  - Code structure visible in a glance
  - Easier code sharing
  - Less misunderstanding

Why Coding Conventions ?

- Quick understanding
- **Reliability**
  - Easier **code review**
    - Code is understandable
    - Obvious bugs are smashed
    - Easier to find a reviewer !

Why Coding Conventions ?

- Quick understanding
- Reliability
- **Maintainability**
  - Easier debugging
  - Easier modifications
  - Crucial for long lifetime systems and quick turnover in teams

Why Coding Conventions ?

- Quick understanding
- Reliability
- Maintainability
- **Security**
  - Systematic use of proven code patterns
    - Example: use of bind variables against SQL injection

Why Coding Conventions ?

- Quick understanding
- Reliability
- Maintainability
- Security
- **Trainability**
  - Crucial when quick turnover in teams
  - Any team member can train any new comer the same way

Why Coding Conventions ?

- Quick understanding
- Reliability
- Maintainability
- Security
- Trainability
- **Speed in coding**
  - Prevents from reinventing the wheel
  - Less thinking about « style »
  - Easier reuse of existing pieces of code

Why Coding Conventions ?

- Quick understanding
- Reliability
- Maintainability
- Security
- Trainability
- Speed in coding
- **Error handling**
  - Errors are handled in a systematic and standard way

Why Coding Conventions ?

- Quick understanding
- Reliability
- Maintainability
- Security
- Trainability
- Speed in coding
- Error handling

Coding Conventions bring great things
but they require some efforts

Coding Conventions

One example

One Set of Coding Conventions

- The following coding conventions are being in use in GS/ASE for more than 15 years
- They have proven their efficiency
- They are given as an example, other conventions may also be good

- The most important:

  Have and follow coding conventions !

Coding Conventions: Case

- Use UPPER CASE for:
  - SQL and PL/SQL keywords
  - Module names
  - Exceptions
  - Constants and types
- Use lower case for
  - Variables
  - Comments
  - Tables, views etc… names
  - Column names

Coding Conventions: Comments

- Use `--` instead of `/*…*/`
  - Easier to comment a whole block of code when debugging

```
BEGIN
  IF p_book_id < 0 THEN
    -- This case should not happen unless the book was lost
    -- Carry on by checking the list of lost books
    …
  END IF;
END;
```

Coding Conventions: Naming

- Parameter naming
  - **p_*name***
- Local variable naming
  - **l_*name***
- Constant naming
  - **C_*NAME***
- Type naming
  - **T_*NAME***
- etc…

Coding Conventions: Indenting

- General indentation:
  - Two blanks indicate a new logical block
  - Example

```
BEGIN
  l_author := ‘Pierre Boulle’;
  IF p_book_id = 12345 THEN
    FOR l_counter IN 1..100 LOOP
      …
    END LOOP;
  END IF;
END;
```

Coding Conventions: Indenting

- SELECT statement:

```
SELECT editor
      ,publication_date
      ,title
  FROM books
 WHERE book_id = 12345
    OR (     title = 'Planet of the Apes'
         AND author = 'Pierre Boulle'
        )
 ORDER BY title;
```

Coding Conventions: Indenting

- INSERT statement:
  - It is much safer to specify the column names

```
INSERT INTO books (
            book_id
           ,title
           ,author
           )
   VALUES (
            12345
           ,'Planet of the Apes'
           ,'Pierre Boulle'
           );
```

Coding Conventions: Indenting

- IF statement:
  - Important: make sure there is always an ELSE statement

```
IF l_var IS NULL THEN
  …
ELSE
  IF l_var > 0 AND l_var < 100 THEN
    …
  ELSE
    …
  END IF;
END IF;
```

Coding Conventions: Indenting

- Concatenation:

```
l_text := ‘Today we are’
          ||TO_CHAR(SYSDATE, ‘DD-MM-YYYY’)
          ||’ and the time is ‘
          ||TO_CHAR(SYSDATE, ‘HH24:MI’);
```

Coding Conventions: Indenting

- Commas:
  - Better at the beginning of each line, rather than at the end (lines are easier to add or remove)

```
SELECT col_1
      ,col_2
      ,col_3
  FROM table
 WHERE col_1 > 0
   AND col_2 IS NOT NULL
   AND col_3 LIKE ‘Hello%’;
```

Coding Conventions: Parameters

- To declare a procedure

```
PROCEDURE GET$BOOK_AUTHOR(
         p_book_id        IN     NUMBER   := NULL
        ,p_title          IN     VARCHAR2 := NULL
        ,p_author            OUT VARCHAR2
        ) IS
BEGIN
  …
END;
```

- Advice: if a parameter is optional, use NULL as default value for easier debugging

Coding Conventions: Parameters

- To call a procedure, use the syntax **=>**
  - No ambiguity regarding which parameter gets which value

```
l_author          books.author%TYPE;

KNL_LIBRARY.GET$BOOK_AUTHOR(
         p_book_id => 12345
        ,p_author  => l_author
        );
```

Coding Conventions: Constants

- Constants (declared in package headers) are a must when strings or numbers have to be compared

```
C_ANSWER_1    CONSTANT VARCHAR2(50) := ‘Blue’;

IF p_answer = ‘blue’ THEN
  …
END IF;


IF p_answer = C_ANSWER_1 THEN
  …
END IF;
```

Coding Conventions: Dynamic Code

- Use bind variables
  - Very good protection against code injection

```
l_statement := 'INSERT INTO log_table (
                            log_date
                           ,log_text
                           )
                    VALUES (
                            :l_date
                           ,:l_text
                           )';
EXECUTE IMMEDIATE l_statement
  USING IN SYSDATE
       ,IN 'Hello World !';
```

Coding Conventions: Metadata

- Very useful: a block of comments before all modules

```
/*-------------------------------------------------------------------------*/
/*                                                                         */
/* Module   : EXE$PROCEDURE_NAME                                           */
/* Goal     : Short description of the module/procedure.                   */
/* Keywords : Few keywords describing what the module does.                */
/* Type     : CHECK INTERFACE DATA_ACTION DATA_RETRIEVER                   */
/*                                                                         */
/*-------------------------------------------------------------------------*/
/* Description:                                                            */
/*                                                                         */
/* Long description of the procedure: its goal.                            */
/* Explanation about parameters (Input and Output).                        */
/* How the procedure works, the "tricks", etc.                             */
/*                                                                         */
/*-------------------------------------------------------------------------*/
/* History:                                                                */
/*                                                                         */
/* YYYY-MM-DD : First name and Name - Creation.                            */
/*                                                                         */
/* YYYY-MM-DD : First name and Name - Review                               */
/*                                                                         */
/* YYYY-MM-DD : First name and Name                                        */
/*              Description of the modification.                           */
/*                                                                         */
/*-------------------------------------------------------------------------*/
PROCEDURE EXE$PROCEDURE_NAME(
            p_param1              IN      VARCHAR2
            …
```

Error Handling

Error Handling

- Errors can produce
  - A crash of the system
  - A result that is not correct (without crashing) or not understandable
- Lots of time may be spent on support / debugging
  - Hence the importance of instrumenting the code

- Three types of error handling
  - Trapping
  - Reporting
  - Recovering

Error Handling: Trapping

- Use custom exceptions
- Advice: always have a ‘when others’ exception
  - Possibility to add useful info in case of crash

```
BEGIN
  …
EXCEPTION
  WHEN L_MY_EXCEPTION THEN
    -- Specific treatment for this error
    …
  WHEN OTHERS THEN
    -- Generic handling (output of parameters for ex.)
    …
END;
```

Error Handling: Reporting

- Once caught, errors have to be reported
  - To the system manager
    - System values, parameters, failing module name etc…
  - To the user
    - Friendly and clear texts
  - From a module to its caller
    - Stuff that can be used by a piece of code to react the best possible way
- Error messages
  - For humans: text
  - For machines: codes

Error Handling: Reporting

- Basic skeleton of a kernel stored procedure (1/3)

```
PROCEDURE GET$BOOK_AUTHOR(
         p_book_id       IN     NUMBER   := NULL
        ,p_title         IN     VARCHAR2 := NULL
        ,p_author           OUT VARCHAR2
        ,p_exitcode         OUT NUMBER
        ,p_exittext         OUT VARCHAR2
        ) IS
  L_PB_FATAL                    EXCEPTION;
BEGIN
  p_exitcode := 0;
  p_exittext := NULL;
  …
EXCEPTION
  …
END; -- GET$BOOK_AUTHOR
```

Systematically in all kernel procedures

Error Handling: Reporting

- Basic skeleton of a kernel stored procedure (2/3)

```
BEGIN
  …
  IF p_book_id IS NULL AND p_title IS NULL THEN
    -- We have no input to compute the author of the book !
    p_exitcode := 20150; -- Invalid input
    p_exittext := 'At least an id or a title has to be provided';
    RAISE L_PB_FATAL;
  END IF;
  …
EXCEPTION
  WHEN L_PB_FATAL THEN
    IF p_exitcode = 0 THEN
      p_exitcode := 20000; -- Error not documented
    END IF;
END;
```

Error Handling: Reporting

- Basic skeleton of a kernel stored procedure (3/3)

```
BEGIN
  …
EXCEPTION
  WHEN L_PB_FATAL THEN
    IF p_exitcode = 0 THEN
      p_exitcode := 20000; -- Error not documented
    END IF;
  WHEN OTHERS THEN
    p_exitcode := SQLCODE;
    p_exittext := SUBSTR('Unexpected error: '
                         ||SQLERRM
                         ||' in GET$BOOK_AUTHOR with parameters '
                         ||NVL(p_book_id, 'NULL')
                         ||'. Please contact sys.support@cern.ch';
END;
```

The original error is forwarded, with more intersting info

Error Handling: Reporting

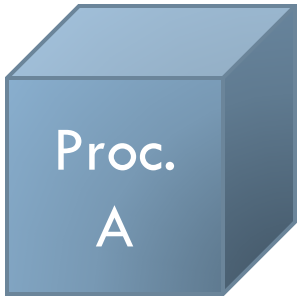
- Standard call to a kernel module:

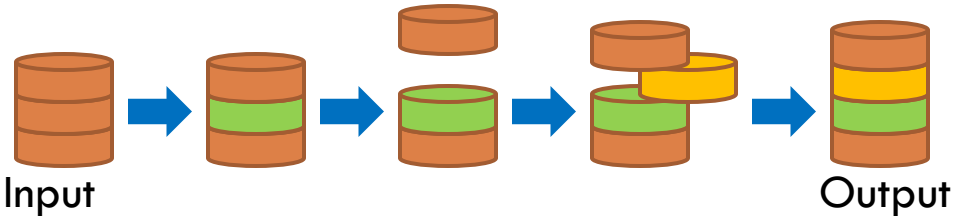
```
PROCEDURE GET$BOOK_DATA(
        p_book_id       IN     NUMBER
       ,p_author           OUT VARCHAR2
       ,p_editor           OUT VARCHAR2
       ,p_exitcode         OUT NUMBER
       ,p_exittext         OUT VARCHAR2
       ) IS
  L_PB_FATAL                  EXCEPTION;
```

```
BEGIN
  KNL_LIBRARY.GET$BOOK_AUTHOR(
        p_book_id  => p_book_id
       ,p_author   => p_author
       ,p_exitcode => p_exitcode
       ,p_exittext => p_exittext
       );
  IF p_exitcode <> 0 THEN
    RAISE L_PB_FATAL;
  END IF;
  …
EXCEPTION
  WHEN L_PB_FATAL THEN
    …
END;
```

Error Handling: Recovering

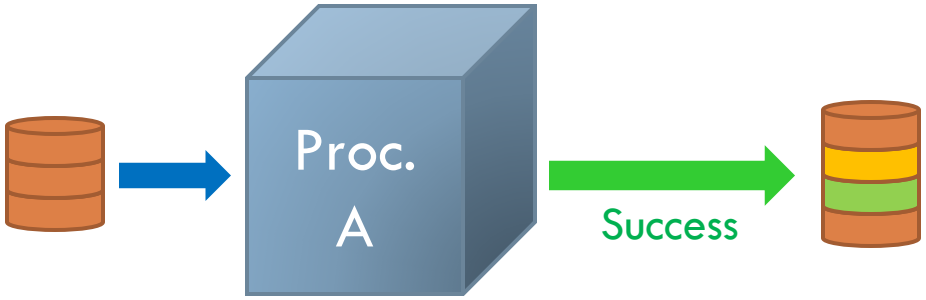
- What if a data action procedure fails ?

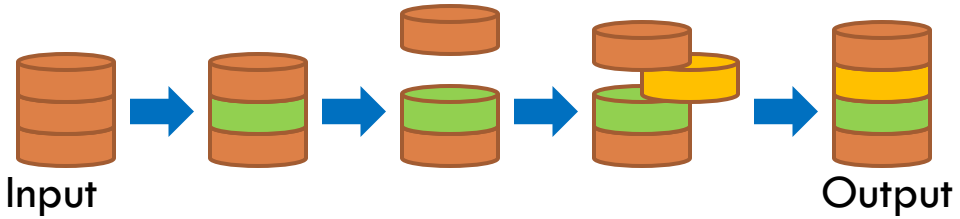




Error Handling: Recovering

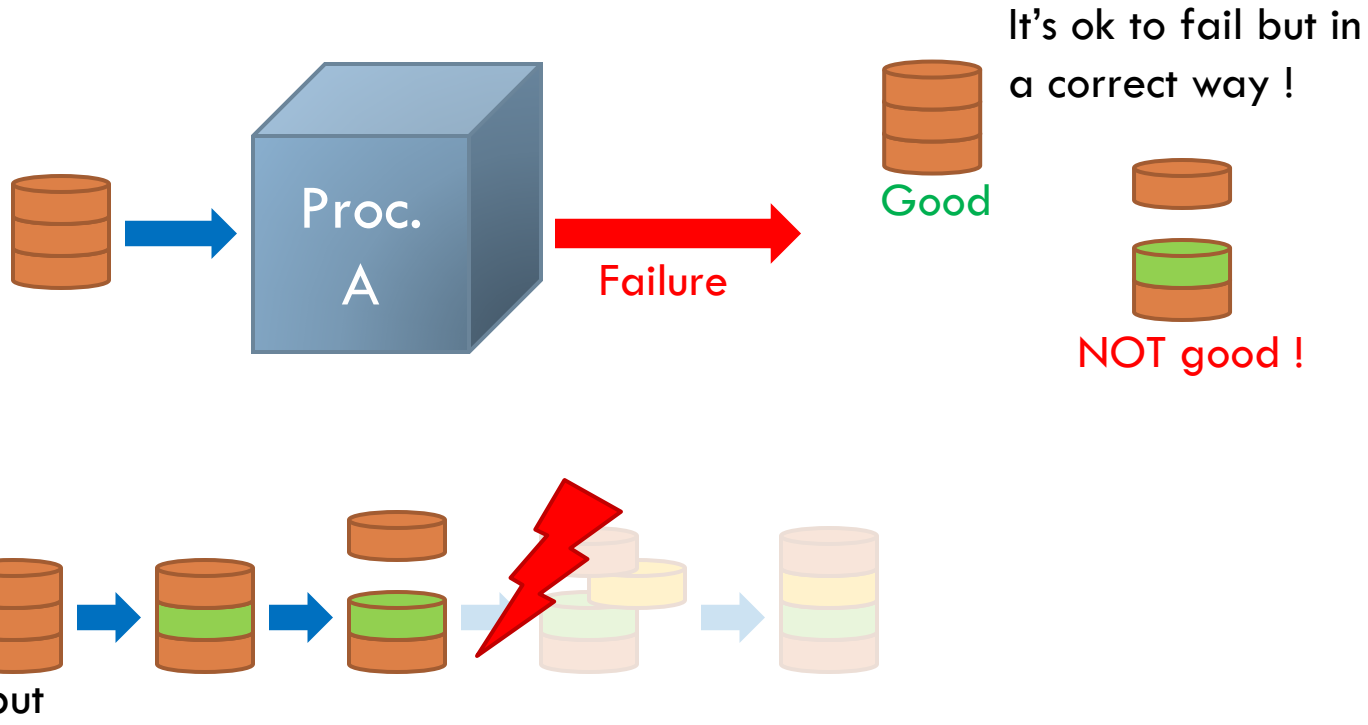
- What if a data action procedure fails ?





Error Handling: Recovering

- What if a data action procedure fails ?



Error Handling: Recovering

- Use savepoints in all data action modules

```
PROCEDURE REGISTER$BOOK(
          …
        ,p_exitcode          OUT NUMBER
        ,p_exittext          OUT VARCHAR2
        ) IS
  L_PB_FATAL                     EXCEPTION;
BEGIN
  SAVEPOINT BEFORE_REGISTERING_BOOK;
  …
EXCEPTION
  WHEN L_PB_FATAL THEN
    ROLLBACK TO BEFORE_REGISTERING_BOOK;
 WHEN OTHERS THEN
    ROLLBACK TO BEFORE_REGISTERING_BOOK;
END; -- GET$BOOK_AUTHOR
```

Error Handling: Procs vs. Funcs

- What about functions ?
  - Functions should return one single value and have no OUT parameters (although its is possible)
    - Therefore, difficult to have a precise error reporting
  - Functions must return something
    - What does a NULL return mean ? Error or not ?
- Advice: use functions only for very simple computations, that never crash (!)

Error Handling: Display Modules

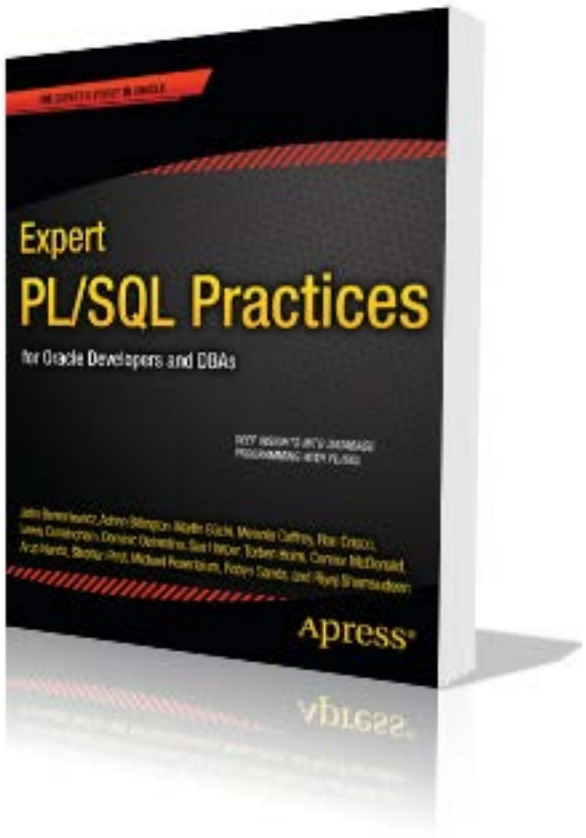
- Test first. Display second.
  - First check all parameters (using kernel modules)
  - Then compute everything that can be computed (idem)
  - If no error was found, display the interface, otherwise gracefully show a nice error message

Summary

Summary

- Keep the **black box** mechanism in mind
  - Build that great library you'd love to use !
  - Asemble components like Lego® elements
- Use **coding conventions**
  - It's a treat to yourself in the future
  - It's a sign of respect to your colleagues today
- **Instrument** your code as much as possibe
  - The worse will always happen at the worst moment !

References



« Expert PL/SQL Practices »
Apress edition

ISBN13: 978-1-4302-3485-2
August 2011

Thank you for your attention !

Stephan.Petit@cern.ch

www.cern.ch