State-of-the-Art
ABAP

Horst Keller, SAP
05, 2021

PUBLIC

Run Simple

Agenda

Role of ABAP – From Yesterday to Today

Extent of ABAP

Expression Enabled ABAP

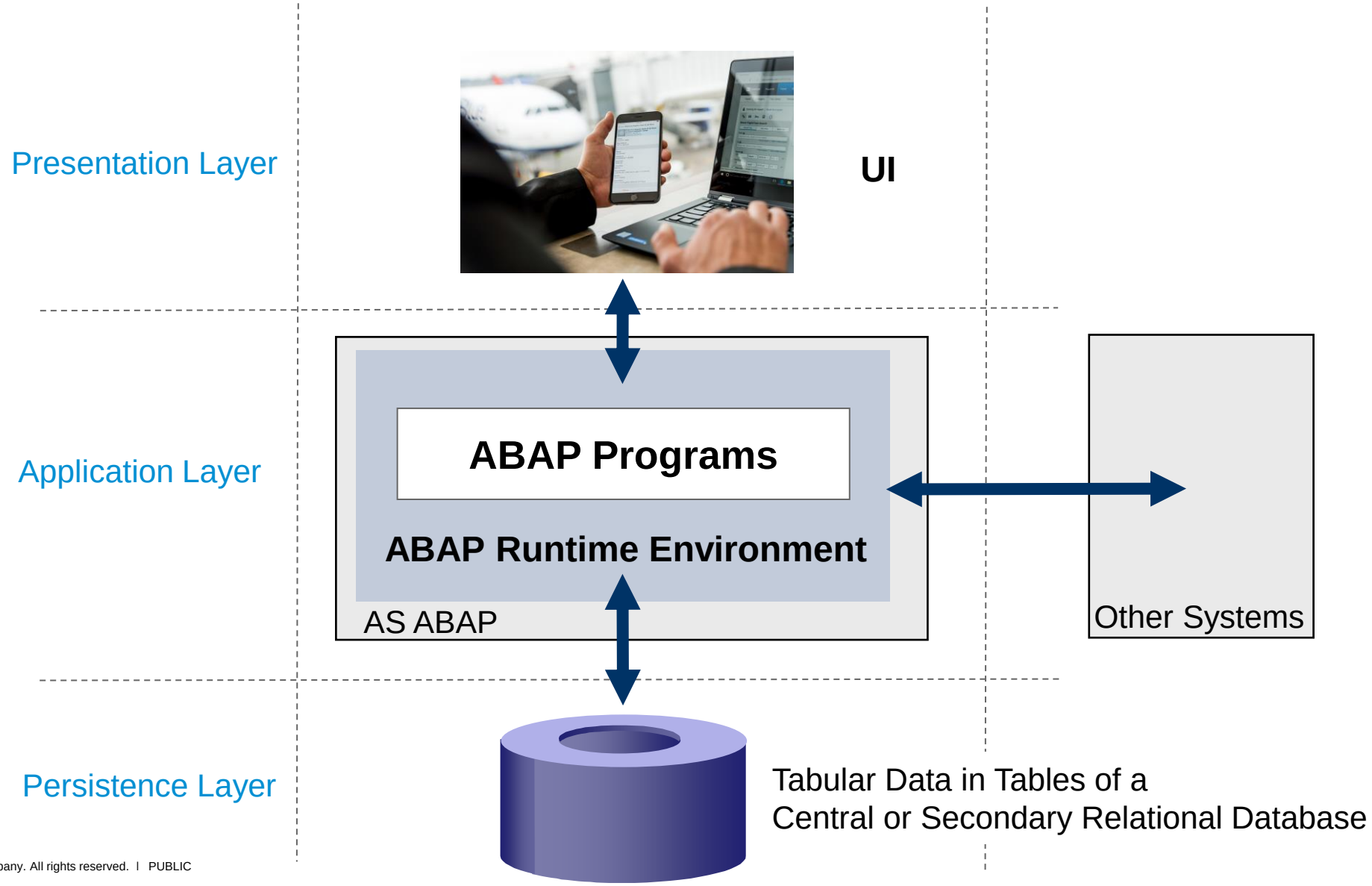
- Expression Enabled Positions
- Expression Enabled String Processing
- Inline Declarations
- Constructor Expressions
- Table Expressions

Further Improvements

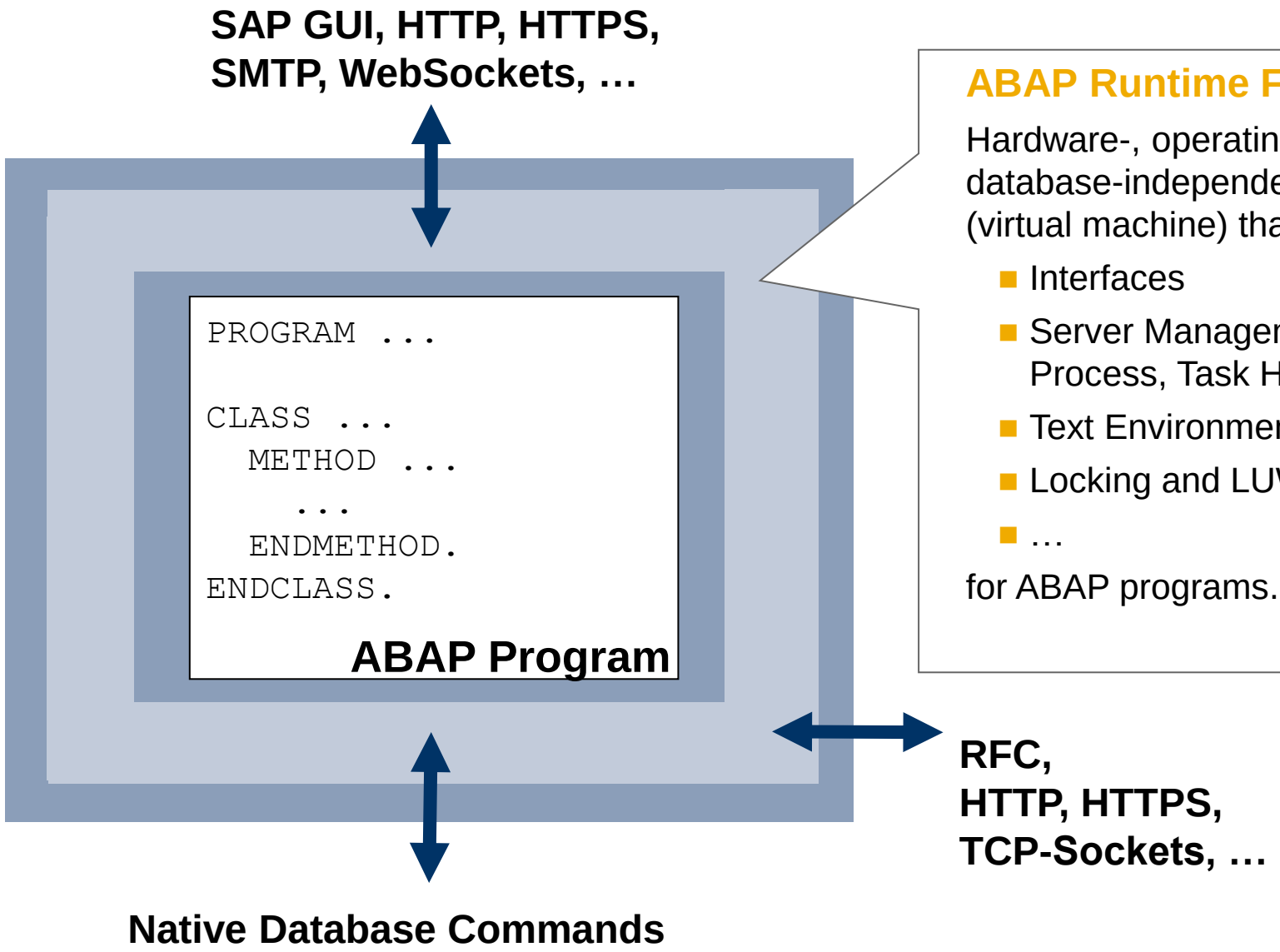
Advanced ABAP SQL

Role of ABAP – From Yesterday to Today

ABAP - Language for Business Application Programming



ABAP Runtime Framework



ABAP Runtime Framework

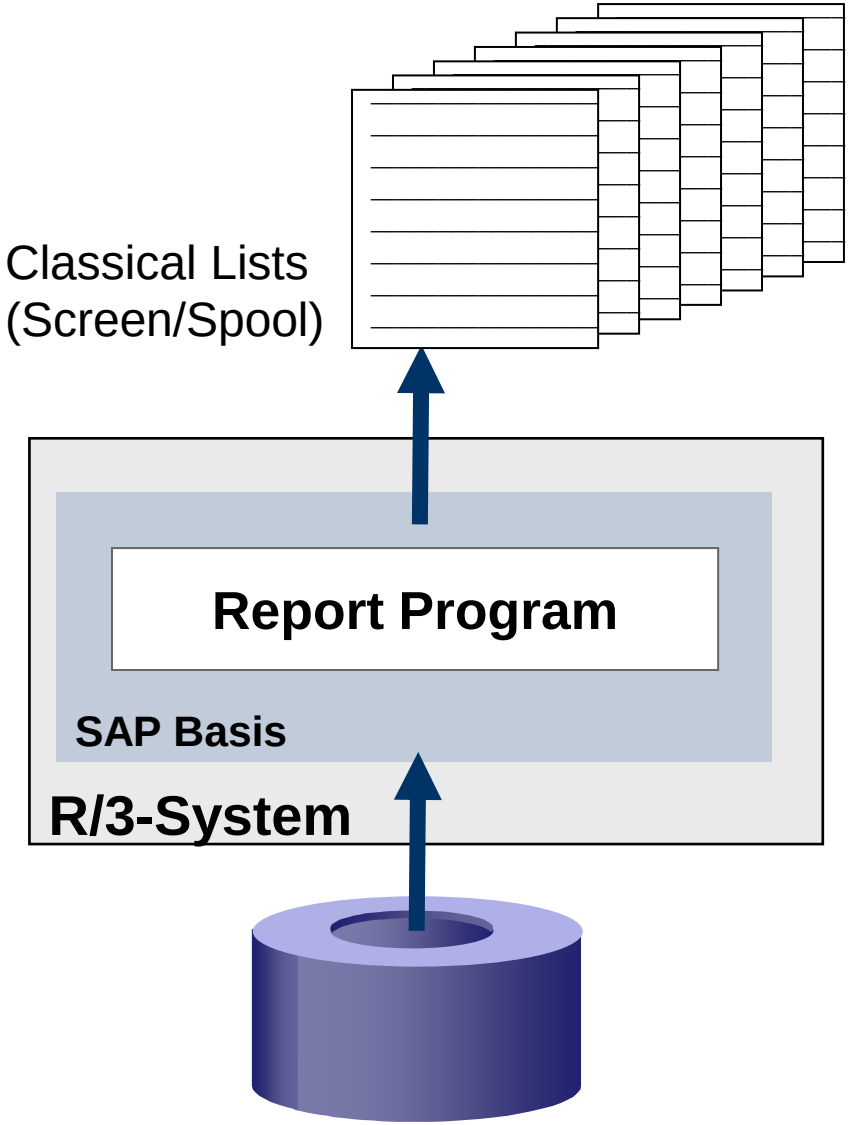
Hardware-, operating-system-, and database-independent platform (virtual machine) that provides

- Interfaces
- Server Management (Memory, Process, Task Handling)
- Text Environment
- Locking and LUW Services
- …

for ABAP programs.

Classical ABAP - Reporting

Demo




```
REPORT demo_report.

NODES: spfli, ...

INITIALIZATION.
  ...

START-OF-SELECTION.
  WRITE ...

GET spfli ...
  WRITE: spfli-carrid, spfli-connid,
         spfli-cityfrom, spfli-cityto.

...

END-OF-SELECTION.
  ULINE.
```

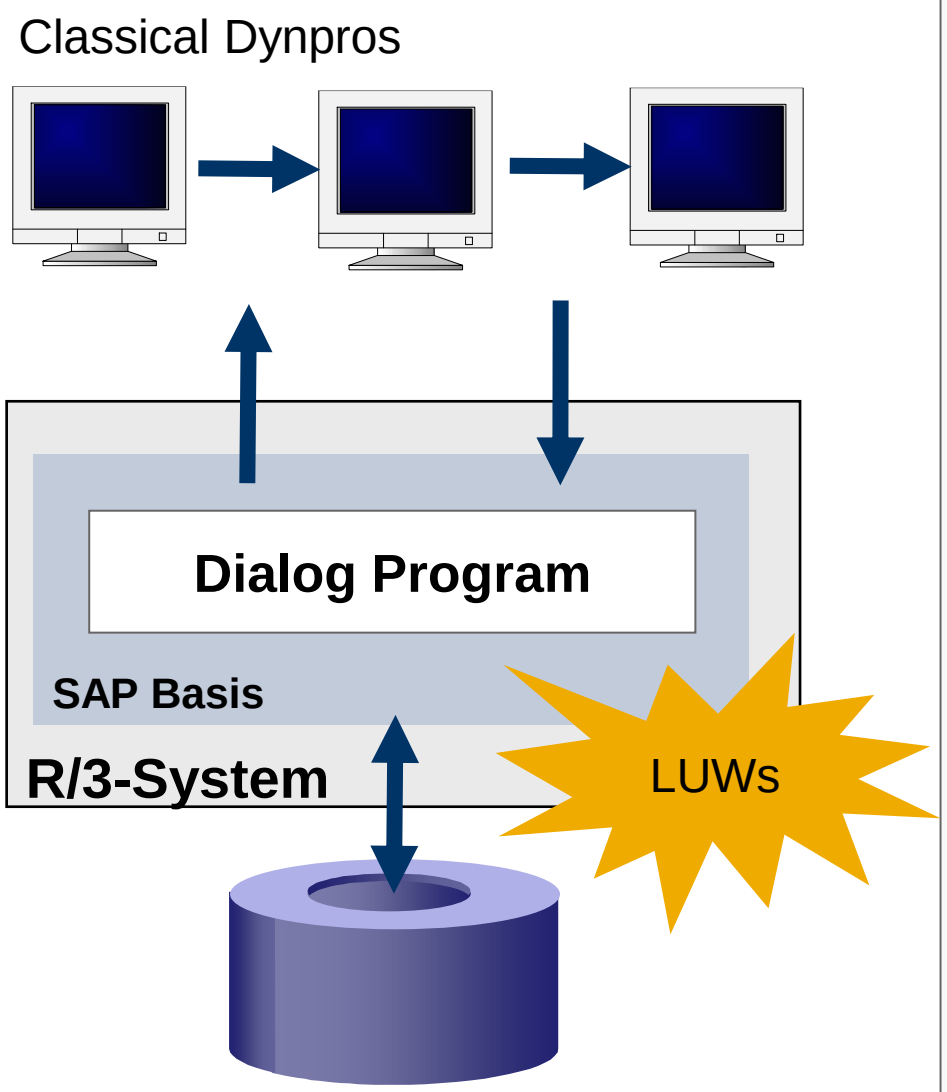
Event Handler Blocks

That is, what WRITE was made for …

ABAP –
Allgemeiner **B**erichts **A**ufbereitungs **P**rozessor

Classical ABAP - Transactions

Demo



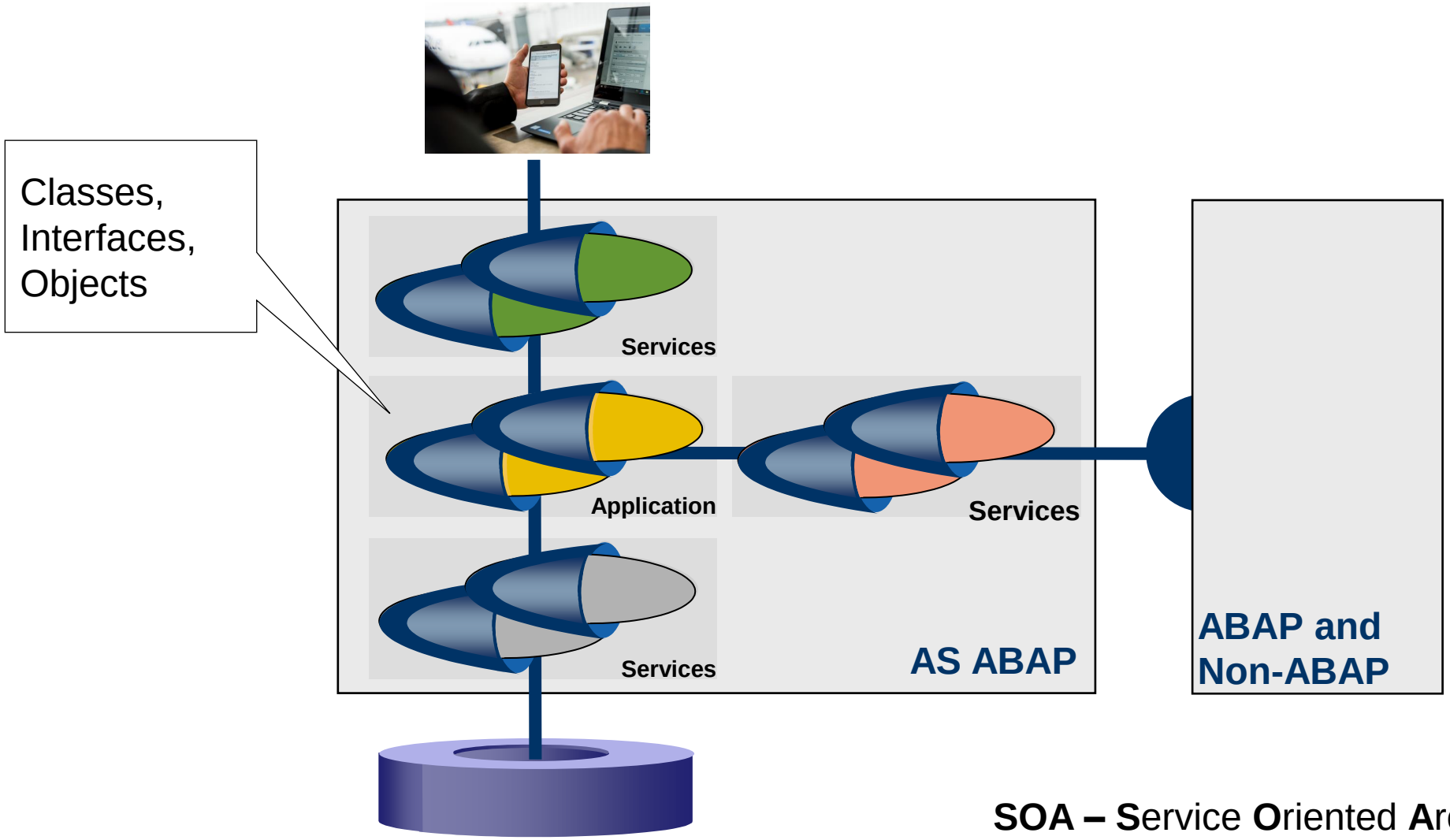

```
PROGRAM sapmdemo_transaction.

MODULE status_0100 OUTPUT.
  SET PF-STATUS 'TD0100'.
  SET TITLEBAR '100'.
ENDMODULE.

MODULE user_command_0100 INPUT.
  CASE ok_code.
    WHEN 'SHOW'.
      CLEAR ok_code.
      SELECT SINGLE *
             FROM spfli
             WHERE carrid = @spfli-carrid
               AND connid = @spfli-connid
             INTO @spfli.
      spfli_wa = spfli.
    WHEN space.
    WHEN OTHERS.
      CLEAR ok_code.
      SET SCREEN 0. LEAVE SCREEN.
  ENDCASE.
ENDMODULE.
```

Dialog Modules

ABAP –
Advanced
Business
Application
Programming

Modern ABAP – ABAP Objects

Demo



SOA – Service **O**riented **A**rchitecture
SOC – **S**eparation **O**f **C**oncerns

Conclusion

Demo

ABAP programming takes place in methods of classes, providing frameworks or being called by frameworks.

How to code there?

State-of-the-Art!

RAP

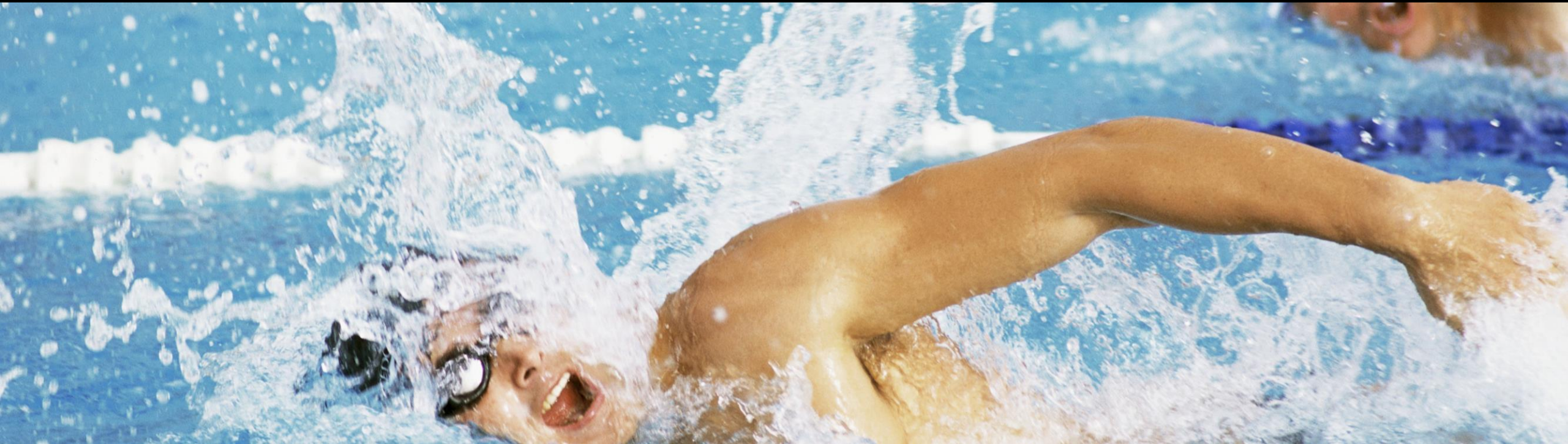
```
CLASS ...

  METHOD ...
    ...
  ENDMETHOD.

ENDCLASS.
```

ABAP Program

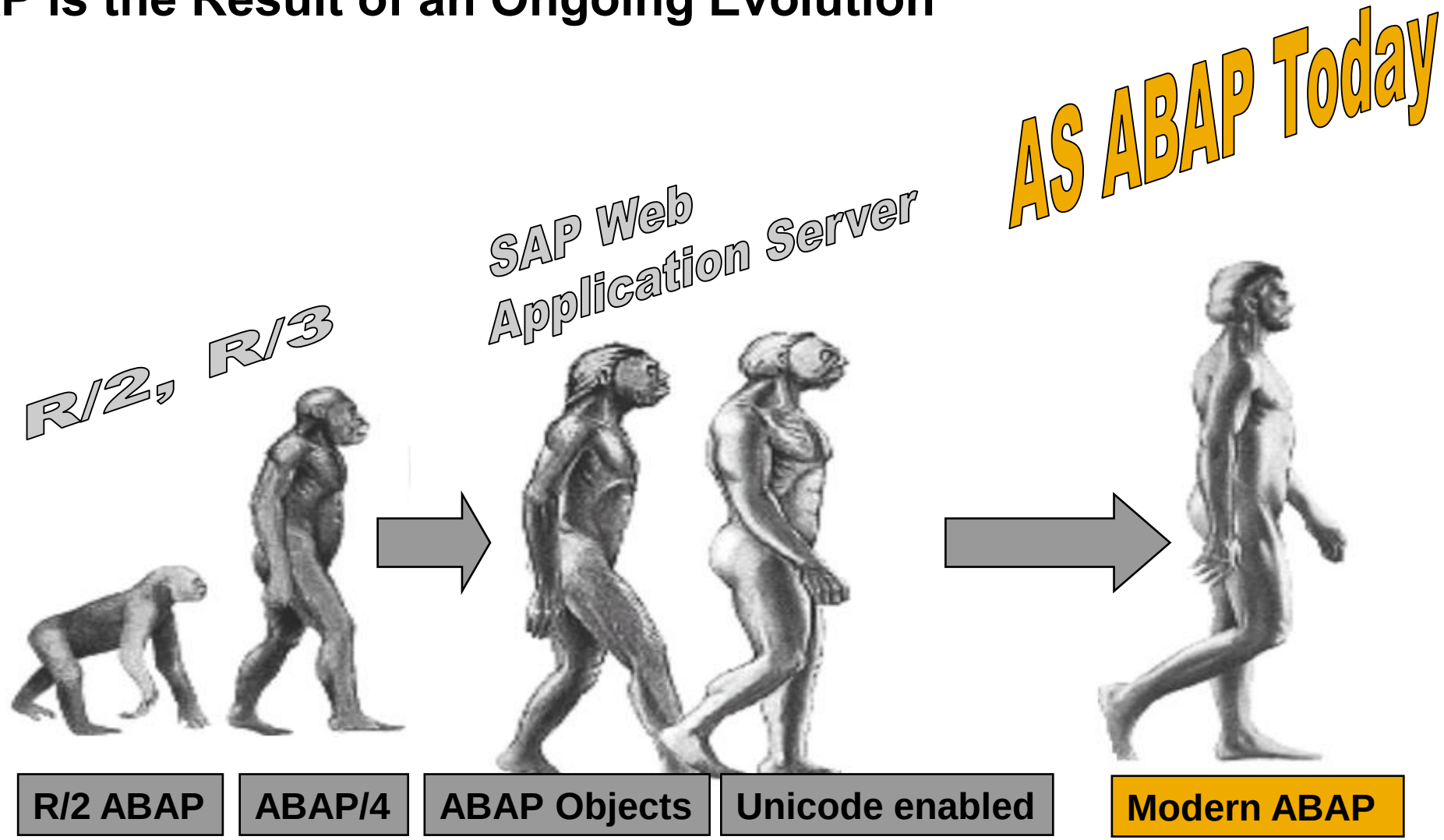
Other Systems

LUWs

Extent of ABAP



Today's ABAP is the Result of an Ongoing Evolution



BC400 BC401 BC???

Unfortunately, There was Never a Cleanup …*

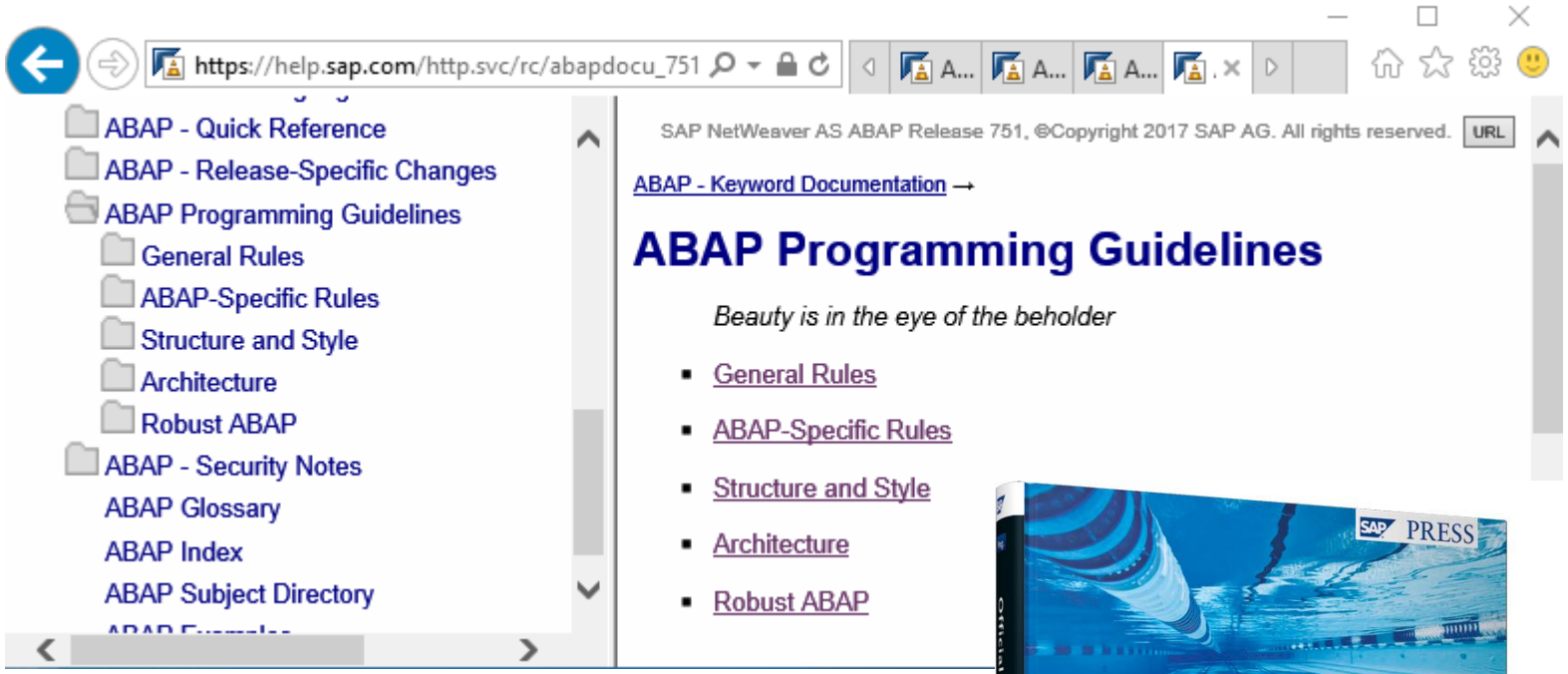
ABAP Words

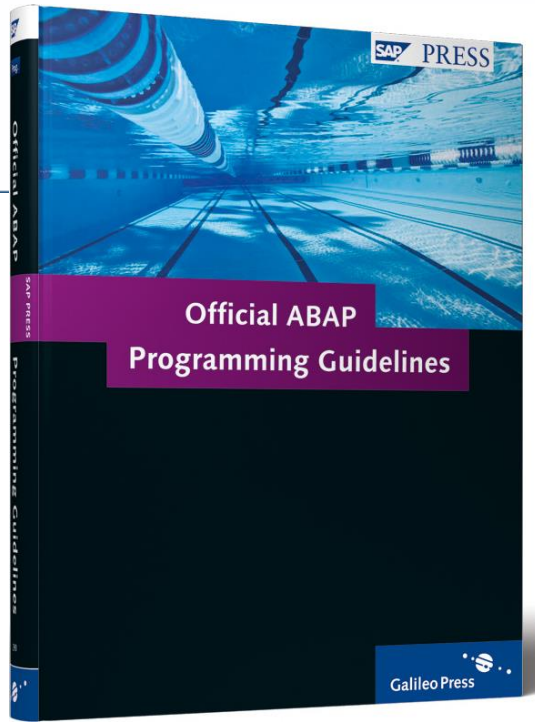
ABAP_SYSTEM_TIMEZONE	ABAP_USER_TIMEZONE	ABAP-SOURCE	ABBREVIATED	ABS
ABSTRACT	ACCEPT	ACCEPTING	ACCORDING	ACTIVATION
ACTUAL	ADABAS	ADD	ADD-CORRESPONDING	ADJACENT
AFTER	ALIAS	ALIASES	ALIGN	ALL
ALLOCATE	ALPHA	ANALYSIS	ANALYZER	AND
ANNOTATE	ANY	APPEND	APPENDAGE	APPENDING
APPLICATION	ARCHIVE	AREA	ARITHMETIC	AS
AS400	ASCENDING	ASPECT	ASSERT	ASSIGN
ASSIGNED	ASSIGNING	ASSOCIATION	ASYNCHRONOUS	AT
ATTRIBUTES	AUTHORITY	AUTHORITY-CHECK	AVG	AVG,
BACK	BACKGROUND	BACKUP	BACKWARD	BADI
BASE	BEFORE	BEGIN	BETWEEN	BIG
BINARY	BINTOHEX	BIT	BIT-AND	BIT-NOT
BIT-OR	BIT-XOR	BLACK	BLANK	BLANKS
BLOB	BLOCK	BLOCKS	BLUE	BOUND
BOUNDARIES	BOUNDS	BOXED	BREAK-POINT	BT
BUFFER	BY	BYPASSING	BYTE	BYTE-CA
BYTE-CN	BYTE-CO	BYTE-CS	BYTE-NA	BYTE-NS
BYTE-ORDER	CA	CALL	CALLING	CASE
CAST	CASTING	CATCH	CEIL	CENTER
CENTERED	CHAIN	CHAIN-INPUT	CHAIN-REQUEST	CHANGE
CHANGING	CHANNELS	CHAR	CHAR-TO-HEX	CHARACTER
CHECK	CHECKBOX	CI_	CIRCULAR	CLASS
CLASS-CODING	CLASS-DATA	CLASS-EVENTS	CLASS-METHODS	CLASS-POOL
CLEANUP	CLEAR	CLIENT	CLNT	CLOB
CLOCK	CLOSE	CN	CO	COALESCE
CODE	CODING	COL_BACKGROUND	COL_GROUP	COL_HEADING
COL_KEY	COL_NEGATIVE	COL_NORMAL	COL_POSITIVE	COL_TOTAL

What to choose?

*but:
restricted
language scope in
strict ABAP

Get Some Guidance …





“Readable Programs”

“ABAP Specifics”

“Programming Model”

“Correct and Robust Programs”

Use the Latest and Greatest

ABAP - Keyword Documentation →

ABAP - Release-Specific Changes

Changes in Releases 7.5x

Changes in Release 7.40 and its SPs

Changes in Release 7.0 and its EhPs

SAP

Home / Community / Blogs

ABAP Language News for Release 7.40

July 22, 2013 | 44,764 Views | Edit

ABAP Language News for Release 7.50

November 27, 2015 | 14,268 Views | Edit

ABAP News for Release 7.51

November 4, 2016 | 7,009 Views | Edit

Demo

```
DATA itab TYPE TABLE OF scarr.
SELECT *
       FROM scarr
       INTO TABLE itab.

DATA wa LIKE LINE OF itab.
READ TABLE itab WITH KEY carrid = 'LH' INTO wa.

DATA output TYPE string.
CONCATENATE 'Carrier:'
            wa-carrname INTO output
                        SEPARATED BY space.

cl_demo_output=>display( output ).
```

```
SELECT *
       FROM scarr
       INTO TABLE @DATA(itab).

cl_demo_output=>display(
  |Carrier: {
     itab[ carrid = 'LH' ]-carrname }| ).
```

Never Use Obsolete Elements!

ABAP - Keyword Documentation → ABAP - Reference →

Obsolete Language Elements

The language elements described in this subnode are obsolete and are still available only for reasons of compatibility with older releases. These statements may still be encountered in older programs but should not be used in new programs.

Most of the obsolete language elements listed here are forbidden in the syntax of classes. This means they can now only be used outside of classes. There are replacement constructions for all obsolete language elements, which improve the efficiency and readability of programs.

- Obsolete Program Attributes
- Obsolete Syntax
- Obsolete Predefined Data Objects
- Obsolete Modularization
- Obsolete Declarations
- Obsolete Object Creation
- Obsolete Calls
- Obsolete Exit
- Obsolete Program Flow
- Obsolete Processing of Internal Data
- Obsolete Processing of External Data
- Obsolete User Dialogs
- Obsolete Text Environment
- Obsolete Program Editing
- Obsolete Data and Communication Interfaces

Obsolete Syntax

```
FORM subr [TABLES table parameters]
          [USING parameters]
          [CHANGING parameters]
          [RAISING exc1|RESUMABLE(exc1) exc2|RESUMABLE(exc2) ...].
  ...
ENDFORM.
```

Obsolete Syntax

```
MOVE {[EXACT] source  TO destination}
   | {        source ?TO destination}.
```

```
2
3 MOVE `Hello ` && `World!` TO DATA(text).
```

Templates | Bookmarks | Task Reposit... | Task List

1 error, 0 warnings, 0 others

Description

Errors (1 item)

"&& `World!`" is not valid.

New syntax!

```
3 DATA(text) = `Hello ` && `World!`.
```

Templates | Problems | Bookmarks | Task Reposit...

0 items

Description

Expression Enabled ABAP

Demo

From Quaint ABAP to a Bit Less Quaint ABAP

```
DATA out TYPE REF TO if_demo_output.
CALL METHOD cl_demo_output=>new
  EXPORTING
    mode   = cl_demo_output=>text_mode
  RECEIVING
    output = out.

DATA text TYPE c LENGTH 100.
SET COUNTRY 'US'.
WRITE sy-datlo TO text.
CONCATENATE 'Today is the' text INTO text SEPARATED BY space.
CALL METHOD out->display( text ).
```

```
cl_demo_output=>new( cl_demo_output=>text_mode
  )->display( |Today is the { sy-datlo COUNTRY = 'US ' }| ).
```

Write Less ABAP

```
COMPUTE lhs = rhs.
```

```
lhs = rhs.
```

KISS

```
DATA result TYPE ...
CALL METHOD meth
  EXPORTING ... = ...
  RECEIVING ... = result.
```

```
DATA result TYPE ...
result = meth( ... ).
```

```
DATA(result) = meth( ... ).
```

```
CONCATENATE text1 text2
            INTO result
            SEPARATED BY space.
```

```
result = text1 && ` ` && text2.
```

```
result = |{ text1 } { text2 }|.
```

Expression Enabled Positions

ABAP 7.02 and Up

```
v1 = a + b.
v2 = c - d.
v3 = meth( v2 ).
IF v1 > v3.
  ...
ENDIF.
```

→

```
IF a + b > meth( c - d ).
  ...
ENDIF.
```

```
len = strlen( txt ) - 1.
DO len TIMES.
  ...
```

→

```
DO strlen( txt ) - 1 TIMES.
  ...
```

```
idx = lines( itab ).
READ TABLE itab INDEX idx ...
```

→

```
READ TABLE itab
     INDEX lines( itab ) ...
```

```
regex = oref->get_regex( ... ).
FIND REGEX regex IN txt.
```

→

```
FIND REGEX oref->get_regex( ... )
     IN txt.
```

```
CONCATENATE txt1 txt2 INTO txt.
CONDENSE txt.
```

→

```
txt = condense( txt1 && txt2 ).
```

```
DATA oref TYPE REF TO c1.
oref = c2=>m2( ).
v = a + b.
oref->m1( v ).
```

→

```
c2=>m2( )->m1( a + b ).
```

Expression Enabled String Processing

Demo

ABAP 7.02 and Up

Concatenation Operator

```
DATA text TYPE string VALUE `Hello`.

CONCATENATE text ` world!`
            INTO text.
```

```
DATA text TYPE string VALUE `Hello`.

text  = text && ` world!`.
```

String Templates

Literal text, embedded expressions, and control characters:

```
|Hello { sy-uname }!\nToday is { sy-datlo DATE = ISO }.\nThe hour is { sy-uzeit DIV 3600 }.|
```

```
|Hello {
 sy-uname }!\nToday is {
 sy-datlo DATE = ISO }.\nThe hour is {
 sy-uzeit DIV 3600 }.|
```

```
|Hello { sy-uname }!\n| &&
|Today is { sy-datlo DATE = ISO }.\n| &&
|The hour is { sy-uzeit DIV 3600 }.|
```

Demo

ABAP 7.02 and Up

String Functions

```
condense, concat_lines_of, contains, count, distance, escape, find, find_end,
find_any_of, find_any_not_of, insert, match, matches, repeat, replace, reverse,
segment, shift_left, shift_right, substring, substring_after substring_from,
substring_before, substring_to, to_upper, to_lower, to_mixed, from_mixed, translate,
...
```

```
result = count( val = `xxx123yyy` regex = `\d+` ).
```

Description functions

```
html = replace( val   = `<title>This is the <i>Title</i></title>`
                regex = `i` && `(?![^<>]*>)`
                with  = `<b>$0</b>`
                occ   =   0 ).
```

Processing functions

7.55,
POSIX→PCRE

```
IF matches( val   = email
            regex = `\w+(\.\w+)*@(\w+\.)+(\w{2,4})` ).
```

Predicate functions

Inline Declarations

7.40 Expression Explosion

Inline Declarations

```
... DATA(var) ...
... FIELD-SYMBOL(<fs>) ...
```

- Declaration operators DATA and FIELD-SYMBOL allow inline declarations of variables and field symbols with declaration expressions in declaration positions.

- Declaration positions are write positions where the operand type can be determined from the context statically.

Demo

Inline Declarations – DATA( )

```
LOOP AT itab INTO DATA(wa).
  ...
ENDLOOP.

READ TABLE itab INTO DATA(wa) ...
```

```
oref->meth( IMPORTING p1 = DATA(a1)
            IMPORTING p2 = DATA(a2)
            ... ).
```

```
FIND ... IN ... MATCH COUNT DATA(cnt).
```

```
DATA(ref) = class=>factory( ... ).
```

```
SELECT FROM spfli
         INNER JOIN scarr
           ON spfli~carrid = scarr~carrid
       FIELDS scarr~carrname AS name,
              spfli~connid   AS connection
       INTO TABLE @DATA(result).
```

```
CALL TRANSFORMATION id SOURCE ...
     RESULT XML DATA(xml).
```

… and much more!

Demo

Inline Declarations – FIELD-SYMBOL( )

```
ASSIGN field TO FIELD-SYMBOL(<fs>).
```

```
LOOP AT itab ASSIGNING FIELD-SYMBOL(<line>).
  ...
ENDLOOP.
```

```
READ TABLE itab ASSIGNING FIELD-SYMBOL(<line>) ...
```

… and that‘s that

Constructor Expressions

7.40 Expression Explosion

Constructor expressions

```
... VALUE
  | NEW
  | CONV
  | EXACT
  | CORRESPONDING
  | REF
  | CAST
  | REDUCE
  | FILTER
  | COND
  | SWITCH dtype|#( ... ) ...
```

dtype or **#**

Construct results of a specific type in general expression positions.

The data type is defined explicitly with **dtype** or inferred from operand position with **#**.

Demo

Constructor Expressions – VALUE( )

Available since 7.40,SP02

```
struct = VALUE #( LET x = struct IN
                  col1 = x-col2
                  col2 = x-col1
                  col4 = 555 ).
```

```
DATA itab TYPE TABLE OF i WITH EMPTY KEY.
itab = VALUE #( FOR i = 1 UNTIL i > 10
                ( ipow( base = i exp = 2 ) ) ).
```

```
itab = VALUE #( ( col1 = 1 col2 = 2 )
                ( col1 = 3 col2 = 4 ) ).
```

```
INSERT scarr FROM TABLE @( VALUE #(
      ( carrid   = 'FF'
        carrname = 'Funny Flyers'
        currcode = 'EUR'
        url      = 'http://www.funnyfly.com' )
      ( carrid   = 'XXL'
        carrname = 'Extra Large Line'
        currcode = 'USD'
        url      = 'http://www.xxlline.com' )
       ) ).
```

```
itab = VALUE #( (    )
                ( 1 )
                ( 2 )
                ( LINES OF jtab ) ).
```

Host expression in ABAP SQL!

For a complete overview and more examples see the documentation.

Demo

Constructor Expressions – NEW( )

```
DATA dref TYPE REF TO data.
dref = NEW i( 555 ).
```

```
DATA dref TYPE REF TO i.
dref = NEW #( 555 ).
```

```
DATA oref TYPE REF TO intf.
oref = NEW class( p1 = ... ).
```

```
DATA oref TYPE REF TO class.
oref = NEW #( p1 = ... ).
```

```
...
METHODS meth
  IMPORTING struct TYPE REF TO scarr.
...
meth( struct = NEW #( carrid   = '...'
                      carrname = '...' ) ).
```

```
...
METHODS meth
  IMPORTING table TYPE REF TO array.
...
meth( table = NEW #( ( … ) ( … ) ( … ) ) ).
```

Replaces CREATE DATA and CREATE OBJECT, same **value construction** capabilities as VALUE( ).

Demo

Constructor Expressions – CONV( )

```
DATA text TYPE c length 255.

DATA(xstr) = cl_abap_codepage=>convert_to(
               source = CONV #( text ) ).
```

```
IF 1 / 3 > 0.
 ...
ENDIF.

IF CONV decfloat34( 1 / 3 ) > 0.
 ...
ENDIF.
```

```
DATA(txt) = VALUE abap_bool(  ).
DATA(str) = ` `.

IF txt = str.
  ...
ENDIF.

IF txt = CONV abap_bool( str ).
  ...
ENDIF.
```

Enforces conversions, for a complete overview and more examples see the documentation.

Demo

Constructor Expressions – EXACT( )

```
TYPES numtext TYPE n LENGTH 10.

cl_demo_output=>display( CONV numtext( '4 Apples + 2 Oranges' ) ).
```

v.s.

```
TYPES numtext TYPE n LENGTH 10.

cl_demo_output=>display( EXACT numtext( '4 Apples + 2 Oranges' ) ).
```

```
TRY.
    DATA(exact_result) = EXACT #( 3 * ( 1 / 3 ) ).
  CATCH cx_sy_conversion_rounding INTO DATA(exc).
    DATA(rounded_result) = exc->value.
ENDTRY.
```

Enforces lossless assignments and calculations,
for a complete overview and more examples see the documentation.

Demo

Constructor Expressions – CORRESPONDING( )

```
target = CORRESPONDING #( source
                          MAPPING foo = bar
                          EXCEPT  exc ).
```

```
TYPES:
  BEGIN OF flight,
    carrid   TYPE spfli-carrid,
    connid   TYPE spfli-connid,
    cityfrom TYPE spfli-cityfrom,
    cityto   TYPE spfli-cityto,
  END OF flight,
  flights TYPE SORTED TABLE OF flight WITH UNIQUE KEY carrid connid.

SELECT *
       FROM spfli
       INTO TABLE @DATA(spfli_tab).

cl_demo_output=>display( CORRESPONDING flights( spfli_tab ) ).
```

Replaces MOVE-CORRESPONDING and adds **mapping**, for many more examples see the documentation.

Demo

Constructor Expressions – REF( )

```
TYPES: cref  TYPE REF TO char1,
       crefs TYPE TABLE OF cref WITH EMPTY KEY.
DATA text TYPE c LENGTH 10 VALUE '0123456789'.

DATA(drefs) = VALUE crefs(
  FOR j = 0 UNTIL j > 9 ( REF #( text+j(1) ) ) ).
```

```
DATA(ptab) = VALUE abap_parmbind_tab( (
                     name = 'name'
                     kind = cl_abap_objectdescr=>exporting
                     value = REF #( para ) ) ).

CALL METHOD ('class')=>('meth')
  PARAMETER-TABLE ptab.
```

```
DATA result TYPE db_table.
DATA(query) = NEW cl_sql_statement( )->execute_query( `...` ).
query->set_param_struct( struct_ref = REF #( result ) ).
WHILE query->next( ) > 0.
  ... result ... ).
ENDWHILE.
```

Replaces GET REFERENCE, for more examples see the documentation.

Demo

Constructor Expressions – CAST( )

```
DATA(components) =
  CAST cl_abap_structdescr(
    cl_abap_typedescr=>describe_by_name( 'T100' ) )->components.
```

Replaces **?=**

```
DATA structdescr TYPE REF TO cl_abap_structdescr.
structdescr ?= cl_abap_typedescr=>describe_by_name( 'T100' ).
DATA(components) = structdescr->components.
```

Also available:

```
IF cl_abap_typedescr=>describe_by_name( 'T100' ) IS INSTANCE OF cl_abap_structdescr.
  ...
ENDIF.
```

```
CASE TYPE OF cl_abap_typedescr=>describe_by_name( 'T100' ).
  WHEN TYPE cl_abap_structdescr.
    ...
ENDCASE.
```

For more examples see the documentation.

Demo

Constructor Expressions – FOR iterations

```
DATA(itab2) = VALUE t_itab2(
                FOR wa IN itab1 WHERE ( col1 < 30 )
                ( col1 = wa-col2 col2 = wa-col3 ) ).
```

```
DATA(itab) = VALUE t_itab(
     FOR j = 11 THEN j + 10 UNTIL j > 40
     ( col1 = j col2 = j + 1 col3 = j + 2  ) ).
```

```
... REDUCE type( INIT ...
                 FOR  ...
                 NEXT ... ) ...
```

For more examples see the documentation.

Demo

Constructor Expressions – REDUCE( )

```
DATA(sum) =
  REDUCE i( INIT s TYPE i
            FOR  i = 1 UNTIL i > 10
            NEXT s = s + i ).
```

```
DATA(result) =
  REDUCE string( INIT text = `Count up:`
                 FOR n = 1 UNTIL n > 10
                 NEXT text = text && | { n }| ).
```

```
DATA itab TYPE STANDARD TABLE OF i WITH EMPTY KEY.
itab = VALUE #( FOR j = 1 WHILE j <= 10 ( j ) ).

DATA(sum) = REDUCE i( INIT x = 0
                      FOR wa IN itab
                      NEXT x = x + wa ).
```

For more examples see the documentation.

Demo

Constructor Expressions – FILTER( )

```
DATA messages TYPE SORTED TABLE OF t100 WITH NON-UNIQUE KEY sprsl.
...
DATA(messages_d) = FILTER #( messages WHERE sprsl = 'D' ).
DATA(messages_e) = FILTER #( messages WHERE sprsl = 'E' ).
```

```
SELECT *
       FROM scarr
       INTO TABLE @DATA(carriers).

DATA filter TYPE SORTED TABLE OF scarr-carrid
                WITH UNIQUE KEY table_line.
filter = VALUE #( ( 'AA ' ) ( 'LH ' ) ( 'UA ' ) ).

DATA(extract) = FILTER #(
  carriers IN filter WHERE carrid = table_line ).
```

For more examples see the documentation.

Demo

Constructor Expressions – COND( ) and SWITCH( )

```
DATA(time) =
  COND #( LET t = '120000' IN
          WHEN sy-timlo < t THEN
            |{ sy-timlo TIME = ISO } AM|
          WHEN sy-timlo > t AND sy-timlo < '240000' THEN
            |{ CONV t( sy-timlo - 12 * 3600 ) TIME = ISO } PM|
          WHEN sy-timlo = t THEN
            |High Noon|
          ELSE
            THROW cx_cant_be( ) ).
```

```
DATA(number) =
    SWITCH string( sy-index
                   WHEN 1 THEN 'one'
                   WHEN 2 THEN 'two'
                   WHEN 3 THEN 'three'
                   ELSE THROW cx_overflow( ) ).
```

For more examples see the documentation.

Constructor Expressions – LET expressions and BASE addition

Demo

```
TYPES text TYPE STANDARD TABLE OF string WITH EMPTY KEY.

cl_demo_output=>new( )->write(
  VALUE text( LET it = `be` IN
                ( |To { it } is to do|          )
                ( |To { it }, or not to { it }| )
                ( |To do is to { it }|          )
                ( |Do { it } do { it } do|      ) ) )->display( ).
```

Demo

```
target = CORRESPONDING #( BASE ( target ) source ).
```

Possible in many constructor expressions.

Table Expressions

7.40 Expression Explosion

Table Expressions

```
... itab[ ] ...
```

Table expressions itab[…] enable read and write access to internal tables at operand positions.

Demo

Table Expressions - Line

```
DATA itab TYPE SORTED TABLE OF spfli
          WITH UNIQUE KEY primary_key COMPONENTS carrid connid
          WITH NON-UNIQUE SORTED KEY mykey COMPONENTS cityfrom cityto.

...

wa = itab[ idx ].

wa = itab[ KEY mykey INDEX idx ].

wa = itab[ KEY primary_key carrid   = '...' connid = '...' ].

wa = itab[ KEY mykey       cityfrom = '...' cityto = '...' ].

wa = itab[ airpfrom = '...' airpto = '...' ].
```

Table expressions replace READ TABLE, see also the documentation.

Demo

Table Expressions - Result

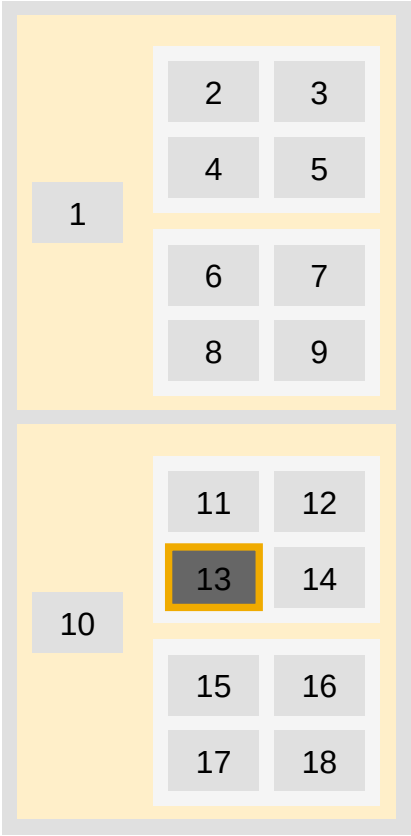
```
... itab[ ... ] ...                                "READ TABLE ASSIGNING


... VALUE #( itab[ ... ] ) ...                     "READ TABLE INTO

... VALUE #( itab[ ... ] OPTIONAL    ) ...         "Exception handling
... VALUE #( itab[ ... ] DEFAULT ... ) ...


... REF #( itab[ ... ] ) ...                       "READ TABLE REFERENCE INTO
```

Table Expressions - lhs

```
itab[ ... ] = ...

itab[ ... ]-comp = ...
```

Demo

Table Expressions - Chaining




```
READ TABLE itab      INTO DATA(wa1) INDEX 2.
READ TABLE wa1-col2 INTO DATA(wa2) INDEX 1.
READ TABLE wa2       INTO DATA(wa3) INDEX 2.
DATA(num) = wa3-col1.
```

```
DATA(num) = itab[ 2 ]-col2[ 1 ][ 2 ]-col1.
```

See also the documentation.

Demo

Internal Tables – Built-in Functions

```
DATA(idx) = line_index(  itab[ col1 = '...'  col2 = '...' ] ).
```

```
IF line_exists(  itab[ col1 = '...'  col2 = '...' ] ).
  ...
ENDIF.
```

```
... COND string( WHEN line_exists( itab[col1 = '...'  col2 = '...' ] ]  )
                 THEN `...` ) ...
```

Grouping of Internal Tables

ABAP 7.53

Demo

Internal Tables – GROUP BY

```
DATA flights TYPE TABLE OF spfli WITH EMPTY KEY.
DATA members LIKE flights.

LOOP AT flights INTO DATA(flight)
     GROUP BY ( carrier = flight-carrid cityfr = flight-cityfrom
                size = GROUP SIZE index = GROUP INDEX )
               ASCENDING REFERENCE INTO DATA(group_ref).
  CLEAR members.
  LOOP AT GROUP group_ref ASSIGNING FIELD-SYMBOL(<flight>).
    members = VALUE #( BASE members ( <flight> ) ).
  ENDLOOP.
  ...
ENDLOOP.
```

```
... FOR GROUPS group OF flight IN flights
        GROUP BY ( carrier = flight-carrid cityfr = flight-cityfrom
                   size = GROUP SIZE index = GROUP INDEX ) ASCENDING
        LET members = VALUE spfli_tab( FOR <flight> IN GROUP group ( <flight> ) )IN ...
```

GROUP BY replaces group level processing with AT, see also the documentation.

Calculation Assignments

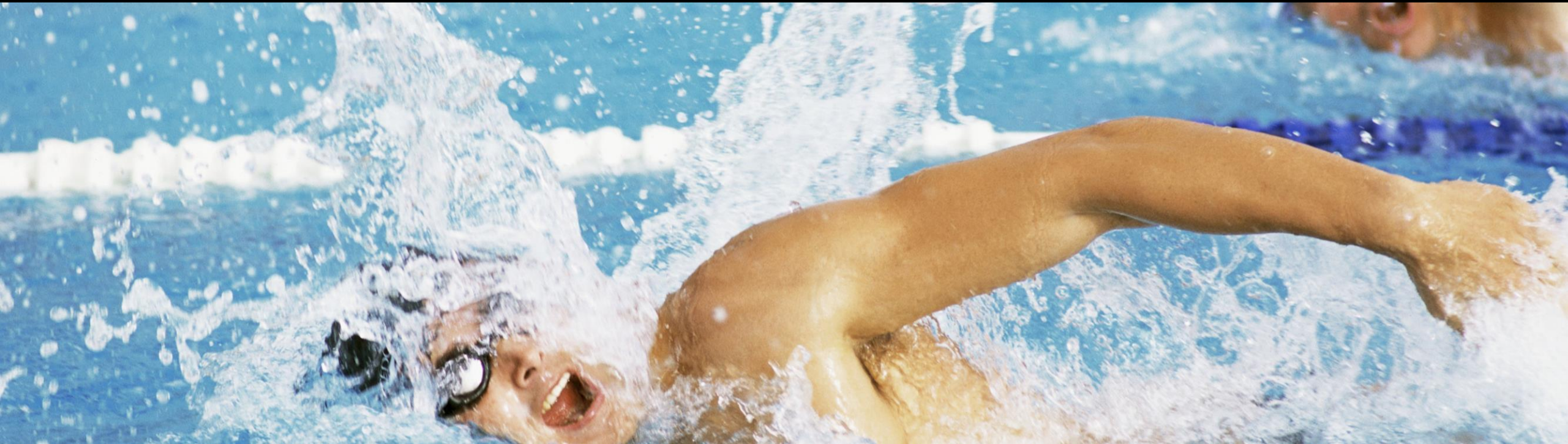
Demo

+=, -=, *=, /=, &&= - Calculation Assignments

```
DO 10 TIMES.
  sum_old  = sum_old  + 1.
  text_old = text_old && sy-abcde+sy-index(1).
ENDDO.
```

```
DO 10 TIMES.
  sum_new  += 1.
  text_new &&= sy-abcde+sy-index(1).
ENDDO.
```

Time Stamp Fields



ABAP 7.54

Demo

Time Stamp Fields with real Time Stamp Type

ABAP:

```
... TYPE utlcong ...
```

DDIC:

```
3  @AbapCatalog.tableCategory : #TRANSPARENT
4  @AbapCatalog.deliveryClass : #A
5  @AbapCatalog.dataMaintenance : #ALLOWED
6  define table demo_ddic_types {
7    key mandt   : abap.clnt not null;
8    key id      : abap.char(1) not null;
42   utcl        : abap.utclong;
```

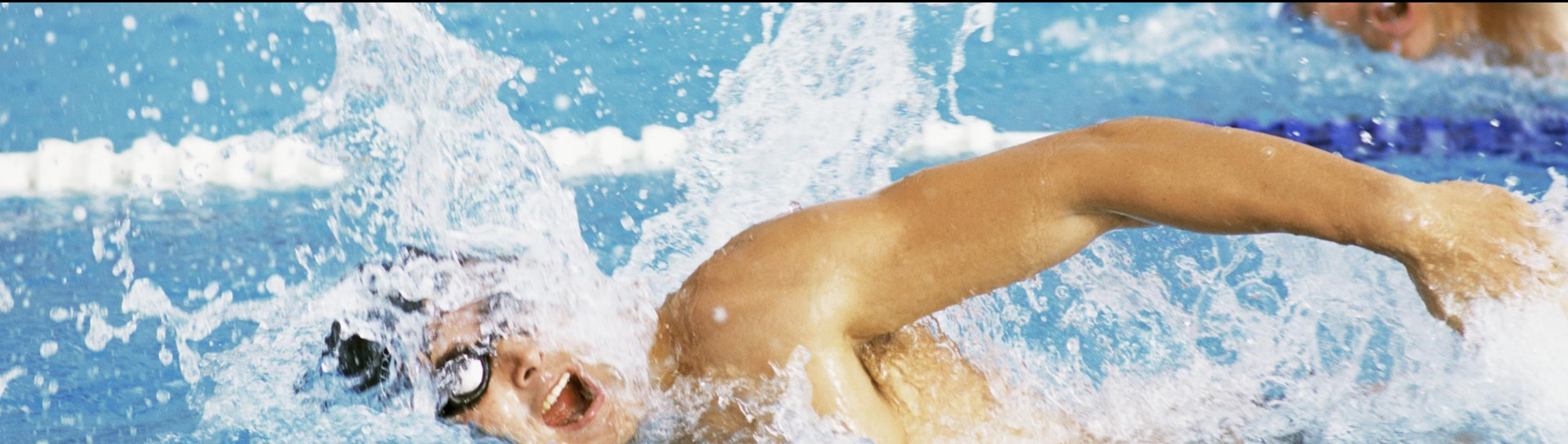
```
DATA(utcl) = utclong_current( ).

utcl = utclong_add( val     = utcl
                    days    = 10
                    hours   = 10
                    minutes = 10
                    seconds = 10 ).

CONVERT UTCLONG utcl
        INTO DATE dat
             TIME tim
             FRACTIONAL SECONDS DATA(fs)
             DAYLIGHT SAVING TIME dst
             TIME ZONE 'CET'.
```

Supported by:

- **Functions**
- **Conversions**
- **System classes**
- **ABAP SQL**
- **ABAP CDS**

Type safe!

Enumerations



ABAP 7.54

Demo

Enumerations

```
CLASS shirt DEFINITION.
  PUBLIC SECTION.
    TYPES:
      BEGIN OF ENUM tsize,
        size_s,
        size_m,
        size_l,
        size_xl,
      END OF ENUM tsize.
    METHODS
      constructor IMPORTING size TYPE tsize.
    ...
  PRIVATE SECTION.
    DATA
      size TYPE tsize.
ENDCLASS.

CLASS shirt IMPLEMENTATION.
  METHOD constructor.
    me->size = size.
  ENDMETHOD.
ENDCLASS.
```

```
DATA(shirt) = NEW shirt( shirt=>size_xl ).
```

```
22  DATA(shirt) = NEW shirt( 333 ).
```

```
TYPES:
  BEGIN OF ENUM tsize STRUCTURE size,
    s, m, l, xl,
  END OF ENUM tsize STRUCTURE size.

DATA dobj TYPE tsize.
...
CASE dobj.
  WHEN size-s.
    ...
  WHEN size-m.
    ...
  WHEN size-l.
    ...
  WHEN size-xl.
    ...
ENDCASE.
```

Perl Compatible Regular Expressions

ABAP 7.55

Demo

PCRE Syntax in ABAP

Non greedy!

```
FIND PCRE `<i>(.*?)</i>` in `<i>abc</i>def<i>ghi</i>`.
```

```
DATA(str) = `4 Apples + 2 Oranges`.
REPLACE ALL OCCURRENCES OF PCRE `\D` IN str WITH ``.
```

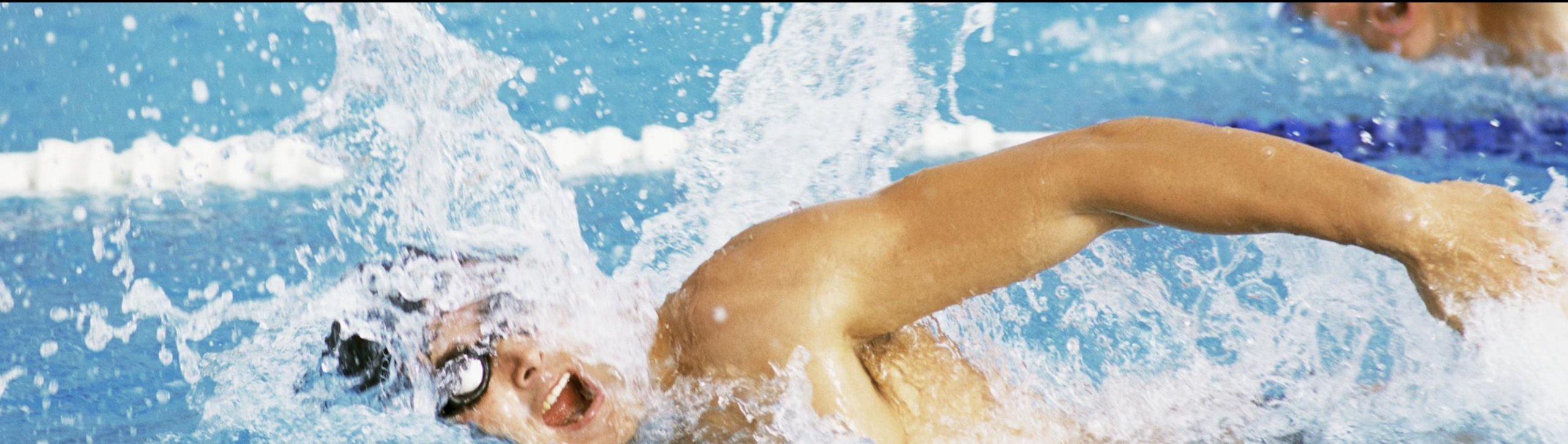
```
ASSERT matches( val = 'abcde' pcre = '[[:alpha:]]*' ).
```

```
DATA(regobj) = cl_abap_regex=>create_pcre( `\b.at\b` ).

FIND REGEX regobj IN text.

DATA(matcher) = regobj->create_matcher( text = `\b.at\b` ).

IF matcher->match( ).
  ... matcher->get_submatch( ... ) ...
ENDIF.
```

PCRE syntax is more powerful and performs better than the now obsolete POSIX syntax.

From 756 on, also XPATH and XSD regular expressions are supported.

Dereferencing Data References



ABAP 7.56

Dereferencing Data References with Generic Type in all Operand Positions

```
DATA dref TYPE REF TO data.

DATA(number) = 5.

dref = REF #( number ).
ASSIGN dref->* TO FIELD-SYMBOL(<fs>).

DATA(result) = 10 + <fs>.
```

Ouch!

```
DATA dref TYPE REF TO data.

DATA(number) = 5.

dref = REF #( number ).

DATA(result) = 10 + dref->*.
```

Advanced ABAP SQL

From 7.40 to Today

```
SELECT FROM sflight
       FIELDS carrid,
              connid,
              fldate
       WHERE carrid = @carrier AND
             connid = @connection
       ORDER BY carrid,
                connid
       INTO TABLE @DATA(sflight_tab).
```

- Comma separated lists
- Optional FIELDS addition
- Denote host variables with @
- INTO clause as last clause
- Inline declarations

Basic syntax rules enforce strict syntax checks, enable new functions …

Support of HANA database mainly or only: Open SQL became ABAP SQL

-> More support for HANA functions

Demo

SQL Expressions

```
SELECT id, num1, num2,
       @flag as flag,
       CAST( num1 AS FLTP ) / CAST( num2 AS FLTP ) AS ratio,
       div( num1, num2 ) AS div,
       mod( num1, num2 ) AS mod,
       @offset + abs( num1 - num2 ) AS sum,
       CASE WHEN num1 > num2 THEN 'X' END AS bigger
       FROM demo_expressions
       WHERE concat( char1,char2 ) = 'duh'
       ORDER BY sum DESCENDING
       INTO TABLE @DATA(results).
```

- Elementary expressions
- Built-in functions
- Arithmetic expressions
- String expressions
- CAST
- CASE

About the same set as for ABAP CDS views and more …

Demo

SQL Functions

There are so many of them …

- Numeric functions (abs, ceil, division, floor, mod, round, …)
- String functions (concat, concat_with, like_regexpr, upper, …)
- Coalesce, UUID, …
- Conversion functions (type conversions, unit and currency conversions, …)
- Date functions and time functions (date function, time functions, time stamp functions)

```
SELECT arbgb, msgnr, text
       FROM t100
       WHERE sprsl = 'E' AND
             like_regexpr( pcre = @regex,
                           value = text,
                           case_sensitive = ' ' ) = 1
       ORDER BY arbgb, msgnr, text
       INTO TABLE @DATA(regex_new)
       UP TO 100 ROWS.
```

Search case insensitively with a PCRE directly on the DB

Demo

Host Expressions @(…)

```
SELECT FROM scarr
       FIELDS carrname
       WHERE  url = @( NEW cls( )->get_url( ) )
       INTO TABLE @DATA(itab).
```

```
INSERT scarr FROM TABLE @( VALUE #(
      ( carrid   = 'FF'
        carrname = 'Funny Flyers'
        currcode = 'EUR'
        url      = 'http://www.funnyfly.com' )
      ( carrid   = 'XXL'
        carrname = 'Extra Large Line'
        currcode = 'USD'
        url      = 'http://www.xxlline.com' ) ) ).
```

```
DATA(rnd) = cl_abap_random_int=>create(
                seed = CONV i( sy-uzeit )
                min = 1
                max = 100 ).

INSERT demo_expressions FROM TABLE @(
   VALUE #(
    FOR i = 0 UNTIL i > 9
      ( id = i
        num1 = rnd->get_next( )
        num2 = rnd->get_next( ) ) ) ).
```

Especially also as arguments of SQL expressions …

Demo

Typed Literals

```
... dtype`...`
```

built-in DDIC types

```
SELECT *
       FROM demo_ddic_types
       WHERE int8 = int8`32984723948723`
       INTO TABLE @DATA(result).
```

```
SELECT SINGLE
       FROM demo_expressions
       FIELDS
         utcl_add_seconds( utclong`2020-04-01T12:01:01,2`,50 ) AS utcl,
         datn_add_months( datn`17890101`,15 ) AS add_months,
         tims_is_valid( tims`121300` ) AS tims
       INTO @DATA(result).
```

Demo

Path Expressions \ …\...

```
SELECT scarr~carrname,
       \_spfli-connid AS connid,
       \_spfli\_sflight-fldate AS fldate,
       \_spfli\_sairport-name AS name
       FROM demo_cds_assoc_scarr AS scarr
       WHERE scarr~carrid = @carrid
       ORDER BY carrname, connid, fldate
       INTO TABLE @DATA(result).
```

```
@AbapCatalog.sqlViewName: 'DEMO_CDS_ASC_CAR'
@AccessControl.authorizationCheck: #NOT_ALLOWED
define view demo_cds_assoc_scarr
  as select from scarr
  association to demo_cds_assoc_spfli as _spfli on
    scarr.carrid = _spfli.carrid
  {
    _spfli,
    carrid,
    carrname
  }
```

CDS view

Demo

Window Expressions

```
SELECT char1 && '_' && char2 AS group,
          num1,
          COUNT(*)      OVER( PARTITION BY char1, char2 ) AS cnt,
          ROW_NUMBER( ) OVER( PARTITION BY char1, char2 ) AS rnum,
          MIN( num1 )   OVER( PARTITION BY char1, char2 ) AS min,
          MAX( num1 )   OVER( PARTITION BY char1, char2 ) AS max,
          SUM( num1 )   OVER( PARTITION BY char1, char2 ) AS sum,
          division( 100 * num1,
                    SUM( num1 ) OVER( PARTITION BY char1, char2 ),
                    2 ) AS perc
          FROM demo_expressions
          ORDER BY group
          INTO TABLE @DATA(windowed).
```

Window functions combined with **partitions** behind OVER.

Demo

Unions

```
SELECT FROM scarr
       FIELDS carrname,
              CAST( '-' AS CHAR( 4 ) ) AS connid,
              '-' AS cityfrom,
              '-' AS cityto
       WHERE carrid = 'LH'
  UNION
    SELECT FROM spfli
           FIELDS '-' AS carrname,
                  CAST( connid AS CHAR( 4 ) ) AS connid,
                  cityfrom,
                  cityto
           WHERE carrid = 'LH'
   ORDER BY carrname DESCENDING, connid, cityfrom, cityto
   INTO TABLE @DATA(result).
```

7.56: INTERSECT, EXCEPT

Demo

Common Table Expressions (CTE)

```
WITH
  +connections AS (
    SELECT spfli~carrid, carrname, connid, cityfrom, cityto
           FROM spfli
           INNER JOIN scarr
             ON scarr~carrid = spfli~carrid
           WHERE spfli~carrid BETWEEN @from_id AND @to_id ),
  +sum_seats AS (
    SELECT carrid, connid, SUM( seatsocc ) AS sum_seats
           FROM sflight
           WHERE carrid BETWEEN @from_id AND @to_id
           GROUP BY carrid, connid ),
  +result( name, connection, departure, arrival, occupied ) AS (
    SELECT carrname, c~connid, cityfrom, cityto, sum_seats
           FROM +connections AS c
             INNER JOIN +sum_seats AS s
               ON c~carrid = s~carrid AND
                  c~connid = s~connid )
  SELECT *
         FROM +result
         ORDER BY name, connection
         INTO TABLE @DATA(result).
```

Demo

SELECT FROM @itab - Internal Tables as Data Sources

```
DATA itab TYPE HASHED TABLE OF scarr
          WITH UNIQUE KEY mandt carrid.

itab =  VALUE #( ( carrid = 'LH' carrname = 'L.H.' )
                 ( carrid = 'UA' carrname = 'U.A.' ) ).

SELECT scarr~carrid, scarr~carrname, spfli~connid
       FROM @itab AS scarr
         INNER JOIN spfli ON scarr~carrid = spfli~carrid
       INTO TABLE @DATA(result)
       ##db_feature_mode[itabs_in_from_clause] ##itab_db_select.
```

```
DATA itab TYPE SORTED TABLE OF i WITH UNIQUE KEY table_line.
itab =  VALUE #( ( 1 ) ( 2 ) ( 3 ) ).

DATA(dyn_clause) =  'table_line = 2'.

SELECT SINGLE table_line AS number
       FROM @itab AS numbers
       WHERE (dyn_clause)
       INTO @DATA(result).
```

Bye, bye READ TABLE?

Demo

SELECT FROM hierarchy – Hierarchies as Data Sources

Hierarchies are

- **CDS hierarchies**
- **ABAP SQL hierarchy generators**
- **CTE hierarchies**

based on hierarchy sources

- **CDS views exposing a hierarchy association**
- **CTEs exposing a hierarchy association**
- **ABAP SQL hierarchies**

```
SELECT FROM HIERARCHY( SOURCE abap_docu_tree_source
                       CHILD TO PARENT ASSOCIATION _tree
                       START WHERE node_key = 'ABENNEWS-75'
                       SIBLINGS ORDER BY tab_index
                       MULTIPLE PARENTS NOT ALLOWED )
       FIELDS node_key, relatkey, hierarchy_level
       INTO TABLE @DATA(result).
```

```
define view entity ABAP_DOCU_TREE_SOURCE
  as select from
    abapdocu_nodes
    association [0..*] to ABAP_DOCU_TREE_SOURCE as _tree on
      $projection.relatkey = _tree.node_key
    { _tree,
  key tab_index,
      node_key,
      relatkey }
```

Hierarchy navigator

```
SELECT FROM HIERARCHY_DESCENDANTS(
         SOURCE HIERARCHY(
                  SOURCE abap_docu_tree_source
                  CHILD TO PARENT ASSOCIATION _tree
                  START WHERE node_key = 'ABENABAP'  )
         START WHERE node_key = 'ABENNEWS-75' )
       FIELDS node_key AS object
       APPENDING TABLE @DATA(descendants).
```

Demo

GTTs, INSERT FROM subquery

Global Temporary Table (GTT)

```
INSERT demo_sumdist_agg FROM
  ( SELECT
      FROM scarr AS s
        INNER JOIN spfli AS p ON s~carrid = p~carrid
      FIELDS s~carrname,
             p~distid,
             SUM( p~distance ) AS sum_distance
      GROUP BY s~mandt, s~carrname, p~distid ).
```

And what about

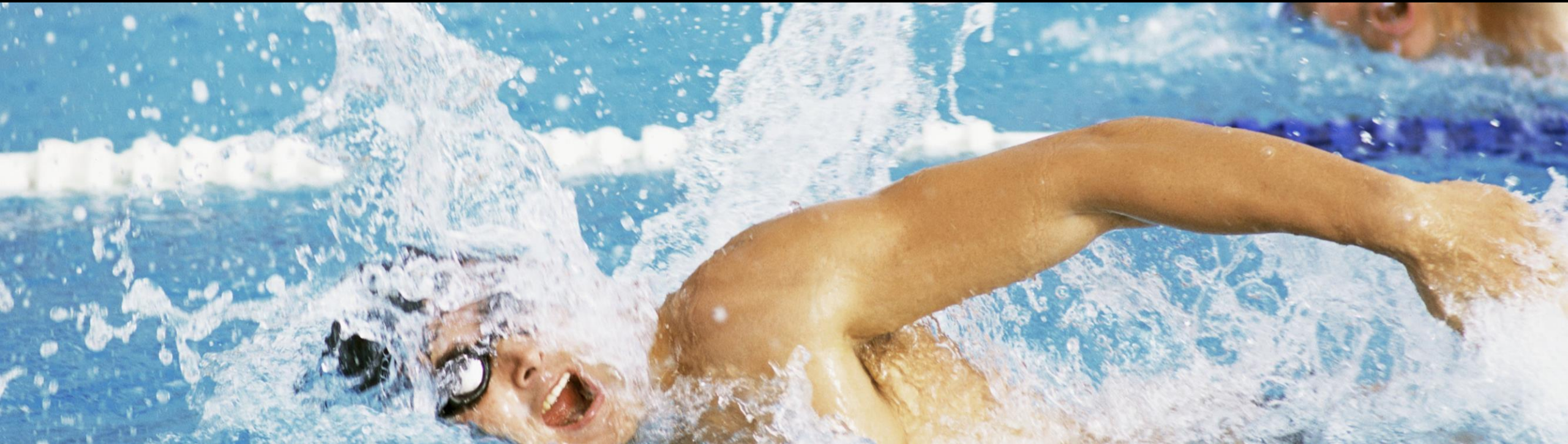
- **AMDP**
- **Native SQL**
  - **ADBC**
  - **EXEC SQL** ???

ABAP programs should use ABAP SQL as the primary method for accessing ABAP-managed database objects *

*Unfortunately, rumors persist, some of them started by SAP itself, that AMDP is always better than ABAP SQL for accessing the SAP HANA database.

Demo

Warning



Pitfalls of Expression Enabling

Obfuscation

```
lcl=>cm_ii(
  lcl=>factory2(
    f_in1 = lcl=>factory0( )->im_ii( i )
    f_in2 = lcl=>cm_ii( 2 ** i ) - 3 +
            itab[ 2 ]-s-xtab[ x = 'abc' ]
  )->m_ii(
      lcl=>factory1( f_in = itab[ a = 2
                b = 4711 / 2
                       ]-x
      )->m_lif_i( lcl=>cm_ii( 5 ) )->im_ii( 6 )
        )
        ).
```

Performance

```
itab1[ 1 ]-a = itab2[ 1 ]-x.
itab1[ 1 ]-b = itab2[ 1 ]-y.
itab1[ 1 ]-c = itab2[ 1 ]-z.
```

Intermediate variables can be helpful and necessary!

Exercises

Exercises

\\abap\Documents\Langu\ABAP_Education\State_of_the_Art_ABAP_2021

Thank you.

Contact information:

Horst Keller
Docu Writer
WDF03, OS4T

Run Simple