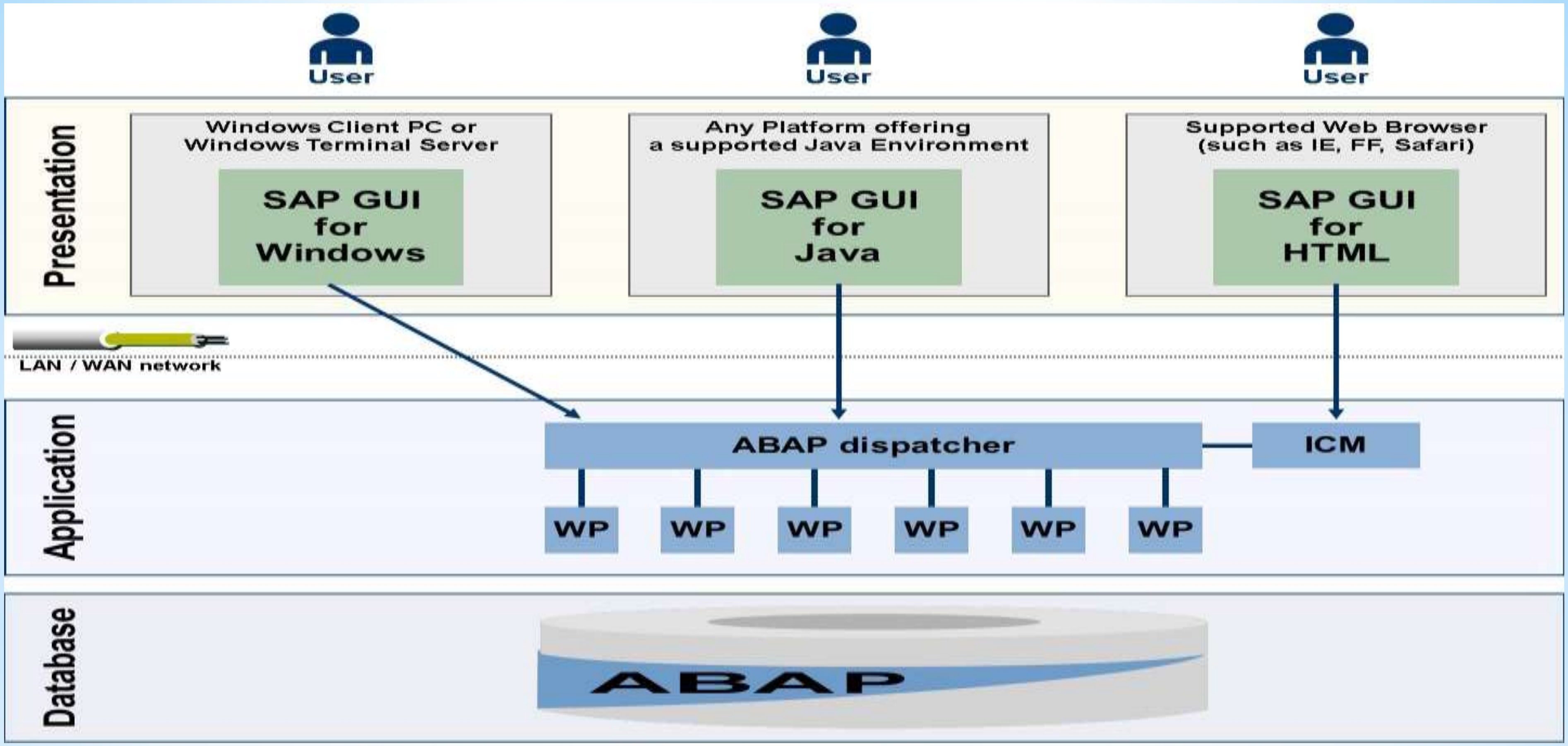
Training on SAP-ABAP Dispatcher and WP

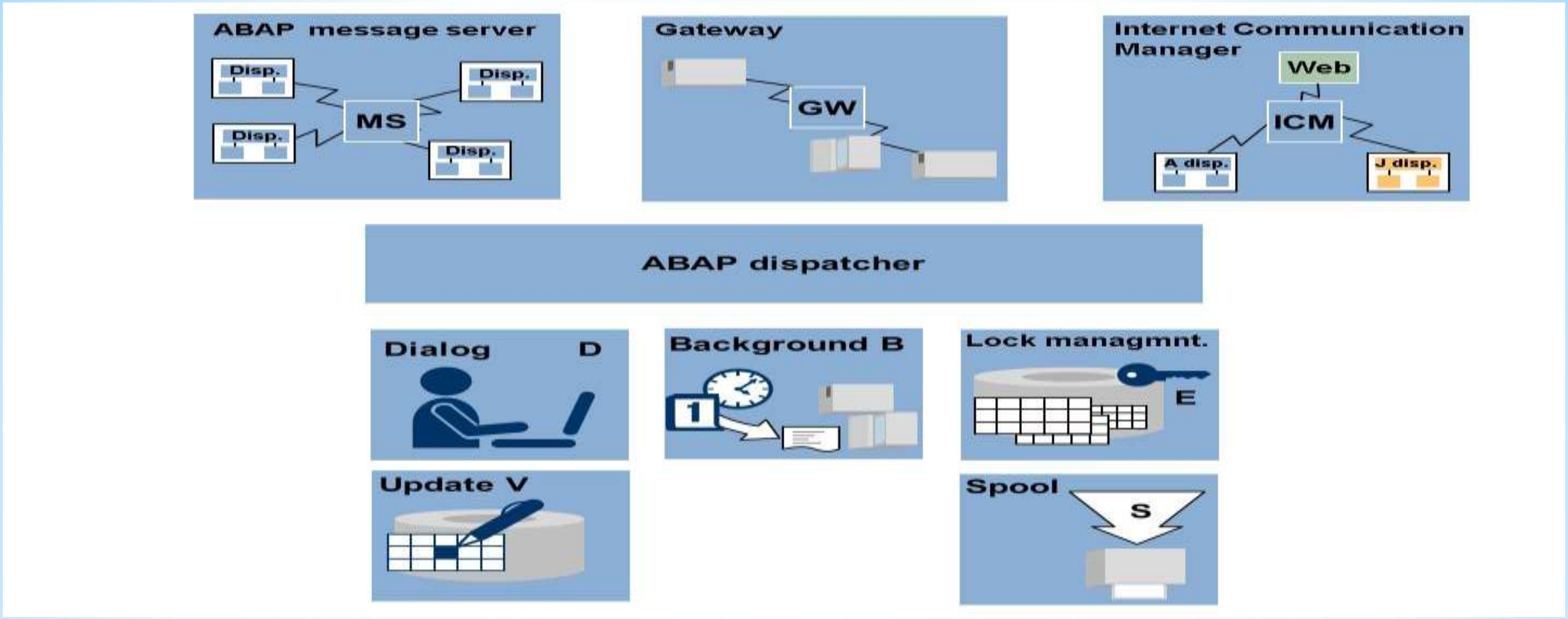
Jijnasee Dash
Faculty,Dept. of CSE
CET,Bhubaneswar

AGENDA

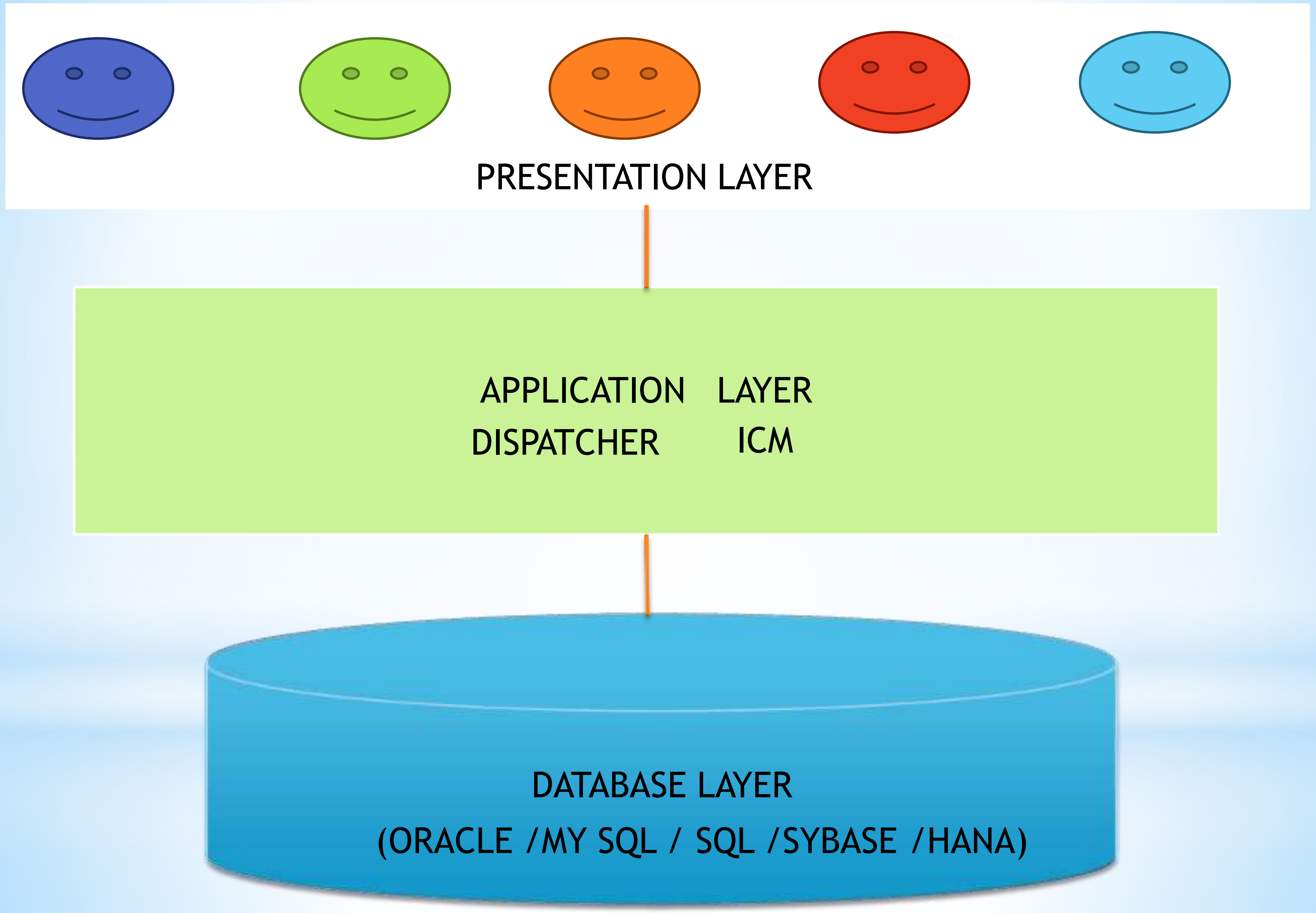
- Dispatcher and WP
- ABAP Workbench
- Standard data types
- Decisions
- Loops and loop control statements
- Q&A Session

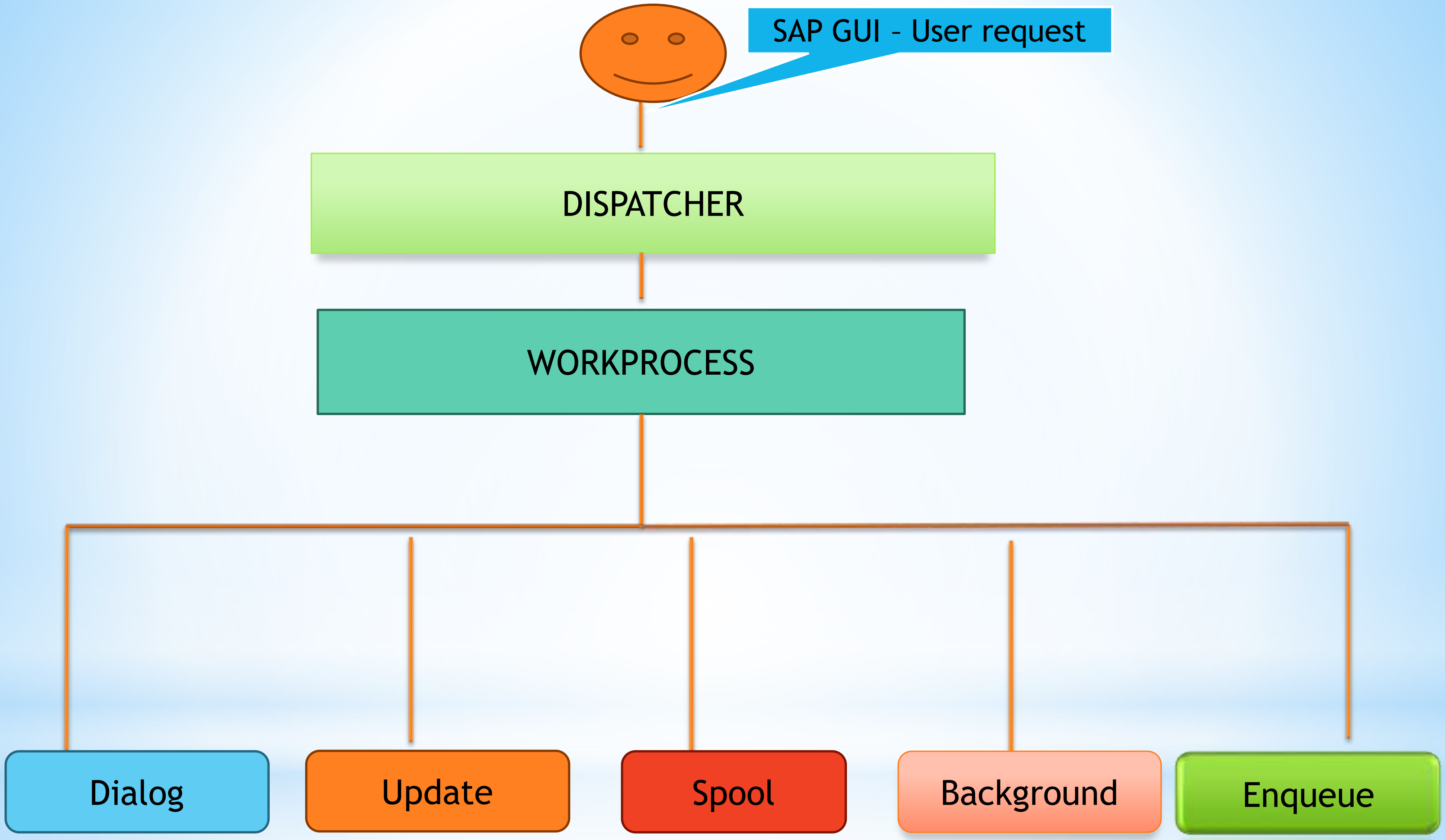


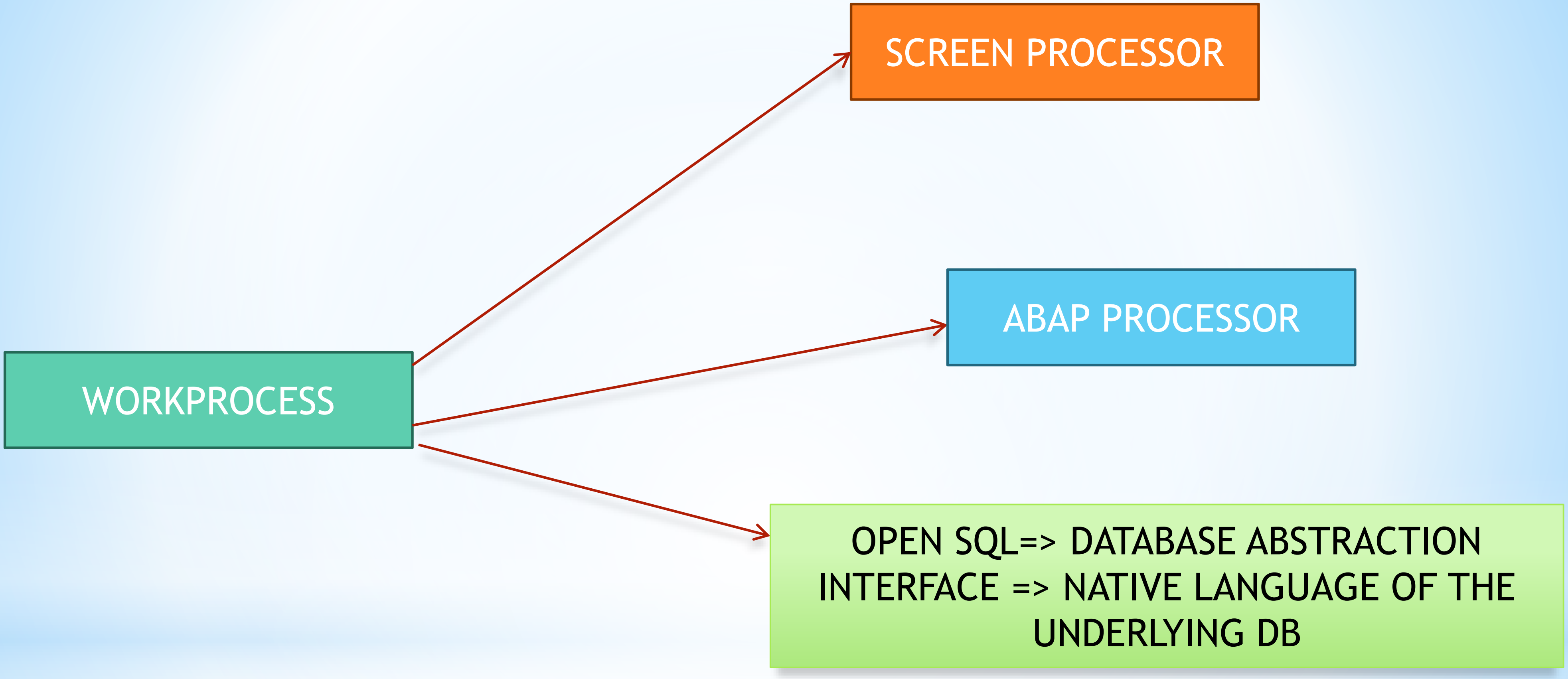
The figure, Variants of SAP GUI, shows the types of SAP GUI and their flow of communication with the ABAP dispatcher or the Internet Communication Manager (ICM)..



In AS ABAP, these ABAP processes include the dispatcher and a number of work processes on the application server, depending on the hardware resources


PRESENTATION LAYER
APPLICATION LAYER
DISPATCHER
ICM
DATABASE LAYER
(ORACLE /MY SQL / SQL /SYBASE /HANA)


SAP GUI – User request
DISPATCHER
WORKPROCESS
Dialog
Update
Spool
Background
Enqueue


SCREEN PROCESSOR
ABAP PROCESSOR
WORKPROCESS
OPEN SQL=> DATABASE ABSTRACTION INTERFACE => NATIVE LANGUAGE OF THE UNDERLYING DB

The dispatcher distributes the requests to the work processes.
The available work processes are as follows:

1. **Dialog work processes** fulfill all requests for the execution of the dialog steps triggered by an active user. Every dispatcher requires at least two dialog work processes.
2. **Spool work processes** pass a sequential flow of data to printers. At least one spool work process is required for each SAP system. It is possible to configure more than one spool work process for each dispatcher.
3. **Update work processes** execute update requests. Similar to spool work processes, you need at least one update work process per SAP system. You can configure more than one update work process per dispatcher.
4. **Background work processes** execute programs that run without interacting with a user. For the normal operation of an SAP system, a single background process would be sufficient. However, for an upgrade or the import of ABAP transport requests, you require a second background work processes. You can configure more than one background work process for each dispatcher.
5. **The enqueue work process** administers the lock table in the shared memory. This table contains the logical database locks of the ABAP runtime environment. Only one enqueue work process is needed for each SAP system.

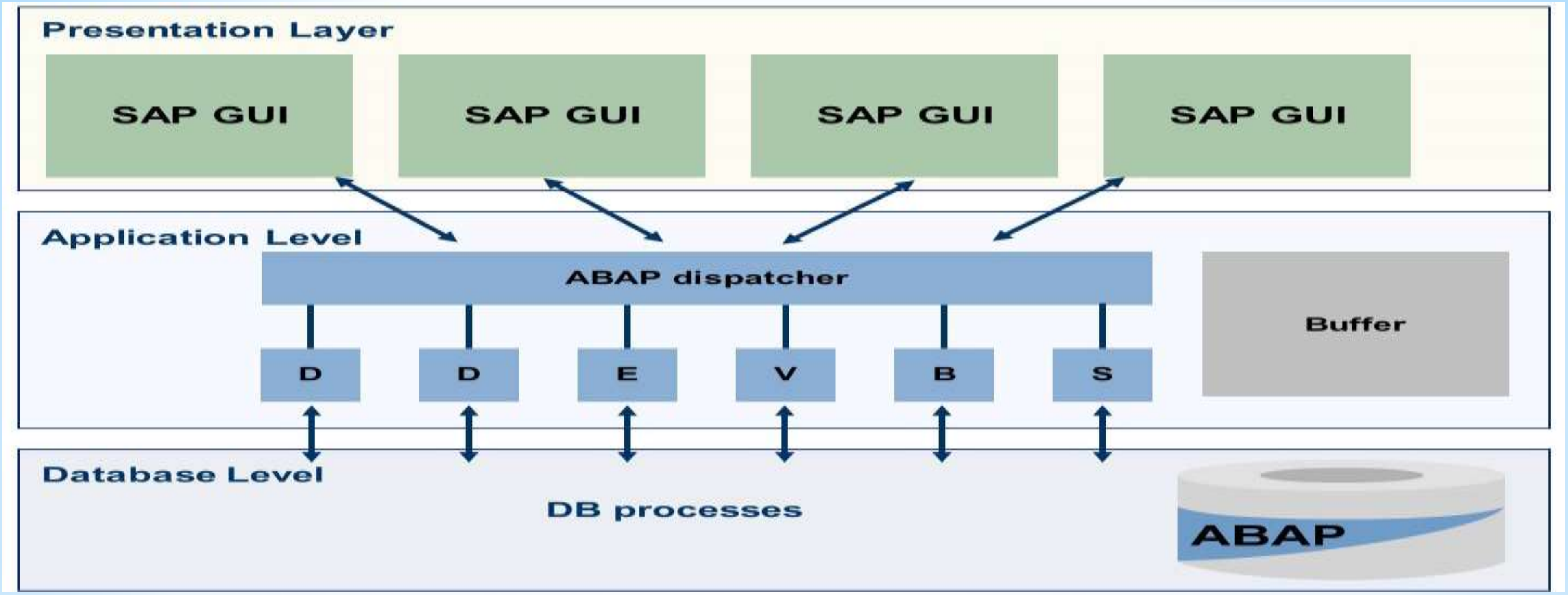
Note:

1. If a system is not used for printing, a spool work process is still required for storing the lists generated in the context of the background process in temporary sequential object (TemSe).

2. Configuration of additional enqueue work processes has to be done on the same instance as the first enqueue work process because these work processes have to access the same lock table.

3. To summarize, the dispatcher of an ABAP instance manages different types of work processes in its instance, such as the dialog, update, background, enqueue, and spool work processes. These work processes work on different tasks when executing the business procedures in an SAP system.

User Request Processing from SAP GUI



- When users log on by using the SAP GUI, the ABAP runtime environment processes the user requests.
- After the users log on via the message server or directly through the ABAP dispatcher, the user requests are executed by the work processes.
- The processing of a user request in AS ABAP, as outlined in the figure, involves different processes from all three layers (presentation, application, and database).
- The screen entries of a user are accepted by the SAP presentation program SAP GUI, converted to an internal format, and then forwarded to AS ABAP

.

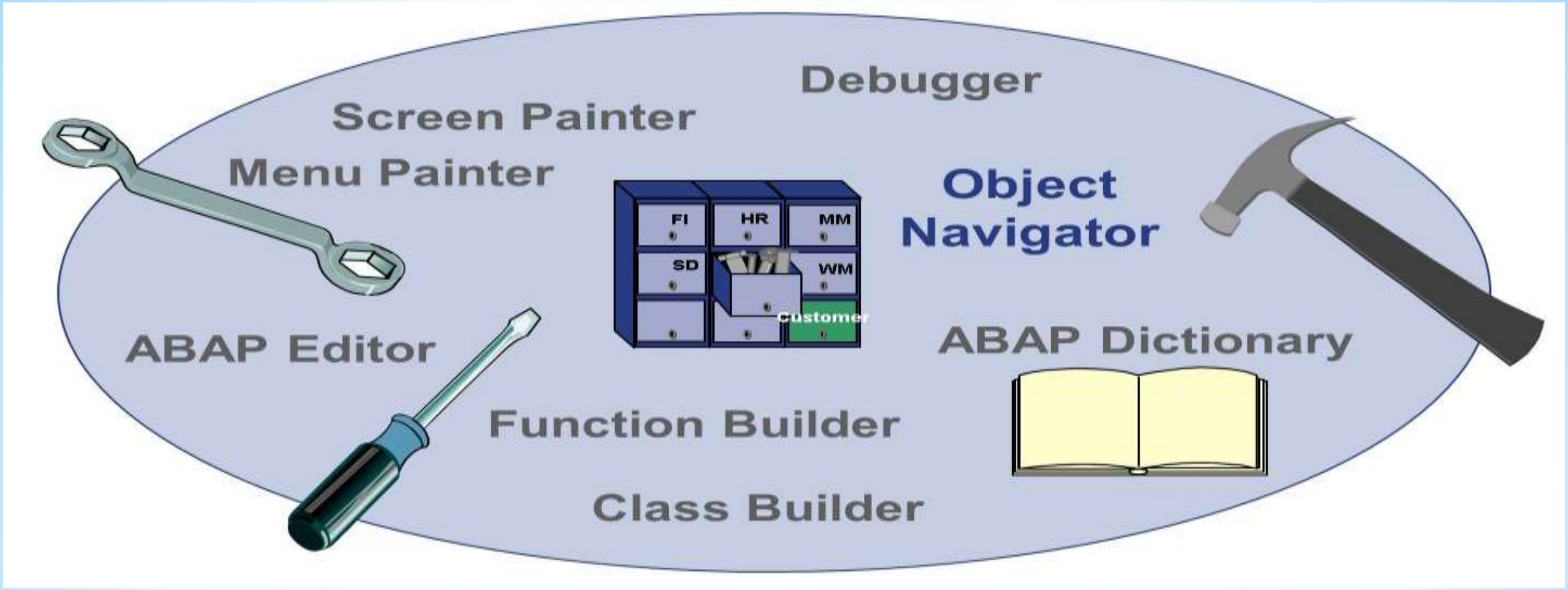
The ABAP dispatcher is the central process of AS ABAP, which performs the following functions:

- Manages the resources for the applications written in ABAP in coordination with the respective operating system
- Distributes the requests among the work processes
- Integrates the presentation layer
- Organizes the communication activities
- Saves the processing requests in request queues and then process them according to the first in, first out principle

The ABAP dispatcher distributes the requests one after the other to the available work processes.
Data is processed in a work process, and the user who creates a request using the SAP GUI is not always assigned the same work process. There is no fixed assignment of work processes to users. To process user requests, it is often necessary to read data from the ABAP schema of the database or to write to it. Every work process connects directly to the ABAP schema of the database to process user
requests.

- Once the work process is complete, its result is sent through the ABAP dispatcher back to the SAP GUI. The SAP GUI interprets the result and generates the output screen for the user.
- The buffers help to speed up the processing of user requests.

- Data that is often read but seldom changed (for example, programs or customizing data such as clients, currencies, or company codes) can be kept as a copy of the database content in the shared memory of the application server. This means that the data does not need to be read from the database every time it is needed and can be called quickly from the buffer. Each instance has its own buffers.



The ABAP Workbench includes all tools required for creating and editing Repository objects. These tools cover the entire software development
cycle.

ABAP Workbench Tools :

Some of the ABAP Workbench tools are as follows:

1. The ABAP Editor is used for editing the source code.

2. The ABAP Dictionary is used for editing database table definitions, data types, and other entities.

3. The Screen Painter is used for configuring screens together with functions for user dialogs.

4. The Menu Painter is used for designing user interface components: menu bar, standard toolbar, application toolbar, and function key settings.

5. The Function Builder is used for maintaining function modules.

6. The Class Builder is used for maintaining global classes and interfaces.

ABAP Editor: transaction SE38

ABAP Dictionary (display, change, create): transaction SE11

ABAP Dictionary (display only): transaction SE12

Screen Painter: transaction SE51

Menu Painter: transaction SE41

Function Builder: transaction SE37

Class Builder: transaction SE24

General Structure of an ABAP Statement

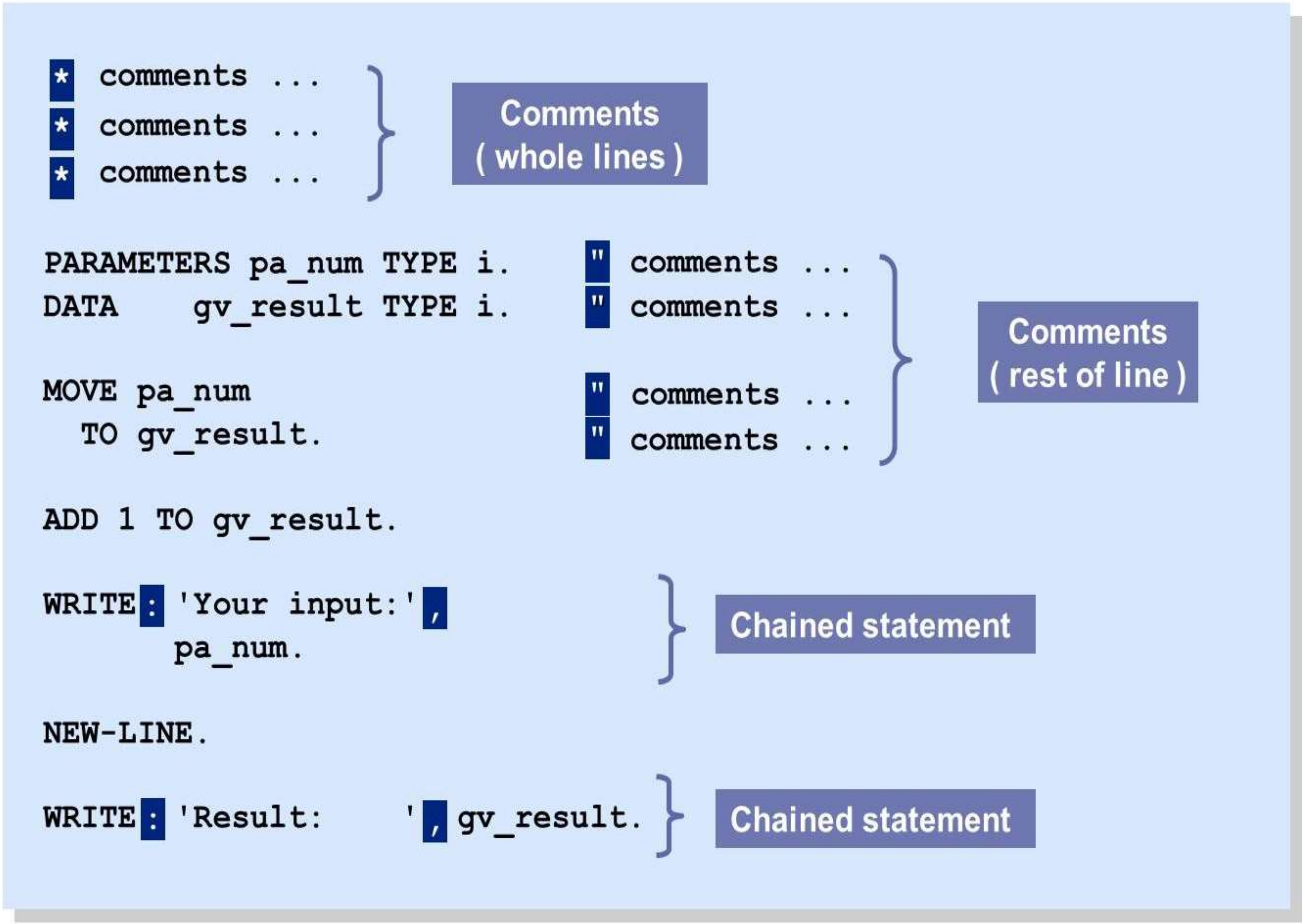
XXX	YYY	.
ABAP keyword	Additions and operands (keyword-specific)	Period to close the statement

Example program

```
PARAMETERS pa_num TYPE i.

DATA gv_result TYPE i.

MOVE pa_num TO gv_result.

ADD 1 TO gv_result.

WRITE 'Your input:'.
WRITE pa_num.

NEW-LINE.

WRITE 'Result:     '.
WRITE gv_result.
```

Characteristics of ABAP Syntax

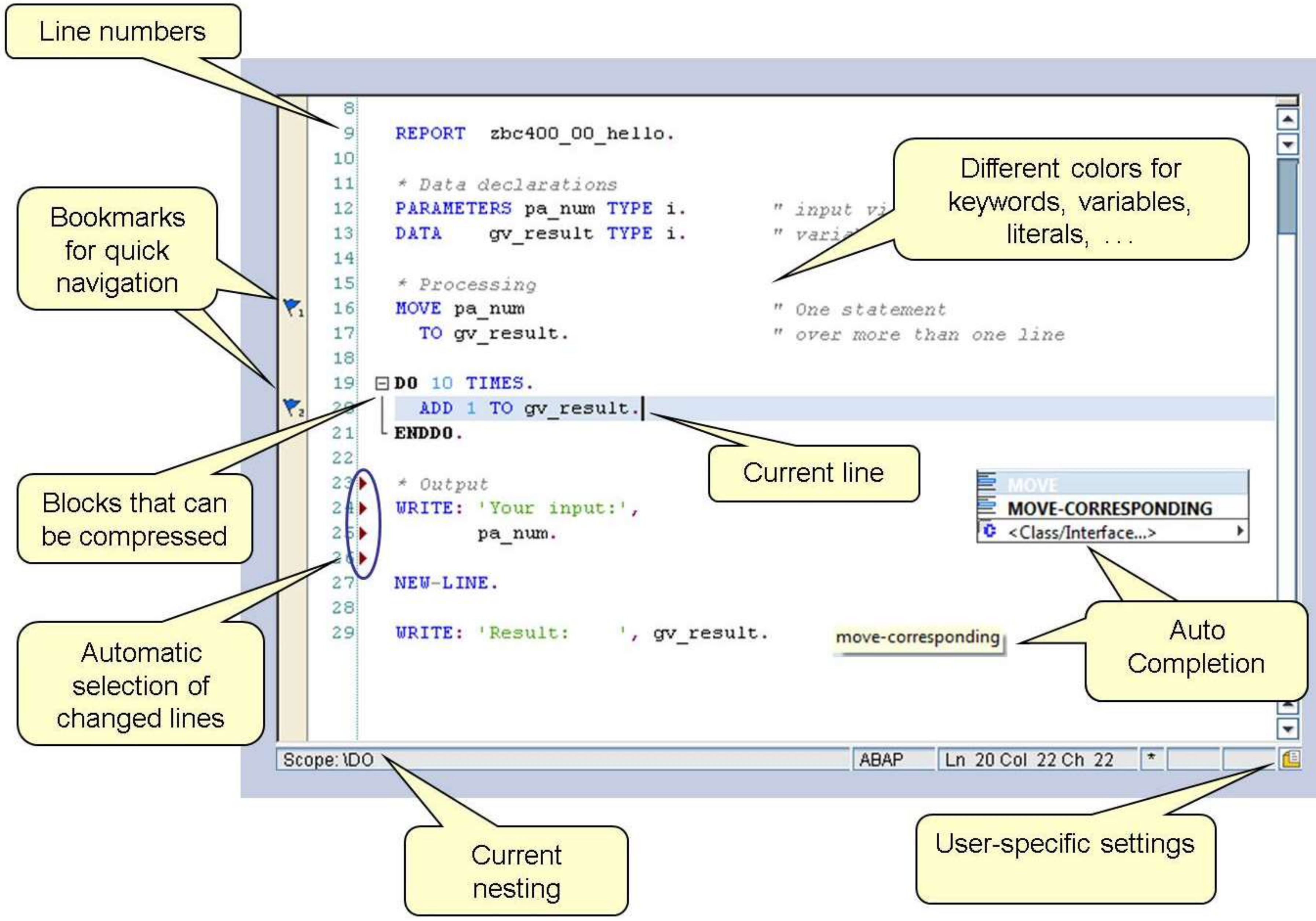
- ABAP programs consist of individual statements.
- The first word in a statement is called an ABAP keyword.
- Each statement ends with a period.
- A space must always separate two words.
- Statements can be indented.
- The ABAP runtime system does not differentiate between upper and lowercase in keywords, additions, and operands.



```
* comments ...
* comments ...
* comments ...

PARAMETERS pa_num TYPE i.      " comments ...
DATA    gv_result TYPE i.      " comments ...

MOVE pa_num                    " comments ...
  TO gv_result.                " comments ...

ADD 1 TO gv_result.

WRITE: 'Your input:',
       pa_num.

NEW-LINE.

WRITE: 'Result:     ', gv_result.
```

Note :

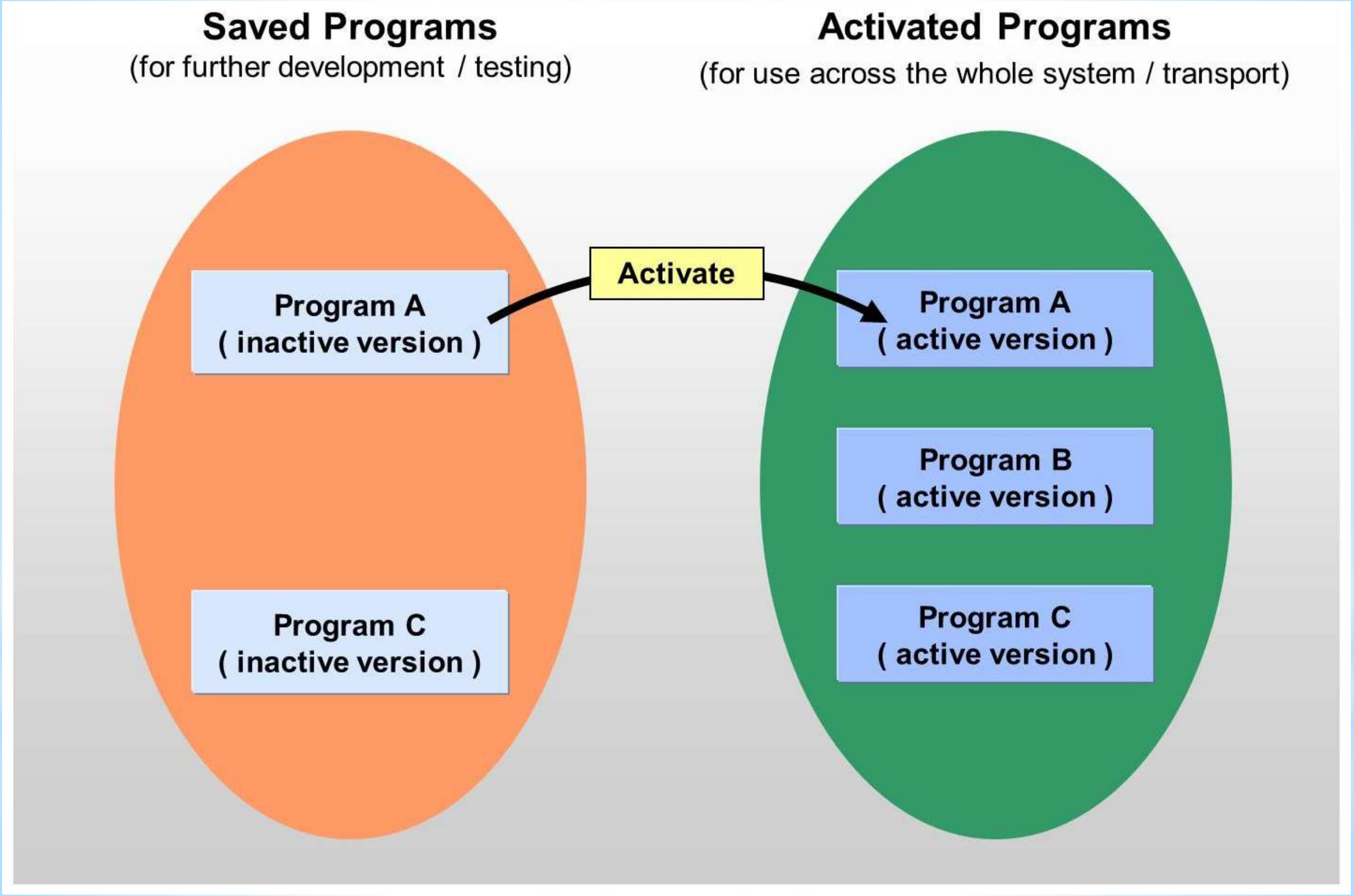
- Statements can extend beyond one line.
- Several statements can lie in a single line (although it is not recommended).
- Lines that begin with an asterisk (*) in the first column are recognized as comment lines by the ABAP runtime system and are ignored.
- A double quotation mark (") indicates that the rest of a line is a comment.
- Write the identical beginning part of the statements, followed by a colon.
- List the end parts of the statements, separated by commas.
- Use blank spaces and line breaks before and after separators, that is, colons, commas, and periods.



```
REPORT  zbc400_00_hello.

* Data declarations
PARAMETERS pa_num TYPE i.          " input vi
DATA     gv_result TYPE i.         " vari

* Processing
MOVE pa_num                        " One statement
  TO gv_result.                    " over more than one line

DO 10 TIMES.
  ADD 1 TO gv_result.
ENDDO.

* Output
WRITE: 'Your input:',
       pa_num.

NEW-LINE.

WRITE: 'Result:    ', gv_result.
```

Options Provided by the New ABAP Editor

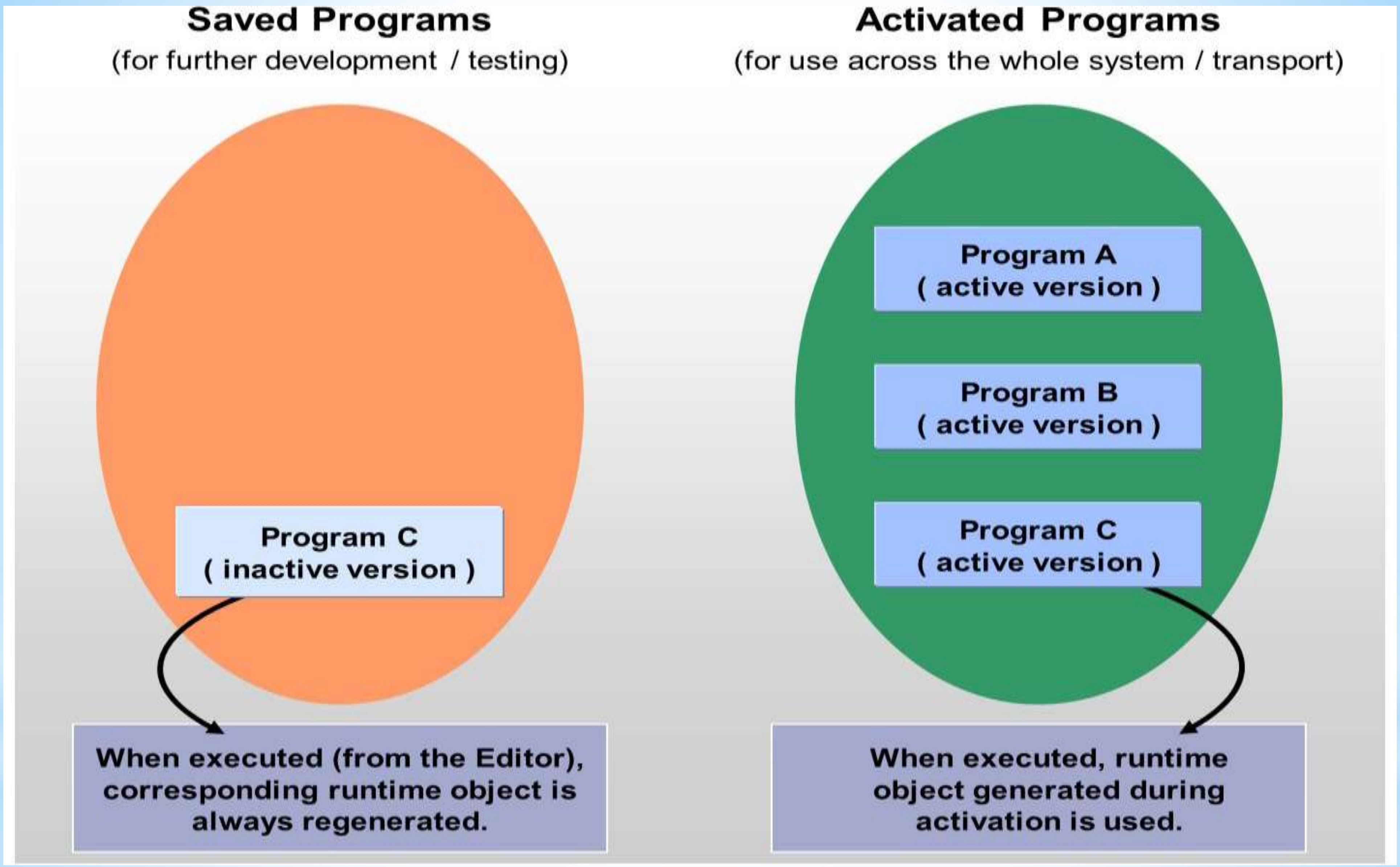
- Some of the important options provided by the new ABAP Editor are as follows:
- You can choose different display colors for different objects in the source code.
- You can set fonts and font sizes for each individual user.
- You can compress different blocks of source code, such as loops and conditional branches, to provide a better overview.
- You can use bookmarks to find relevant points in the source code faster.
- You can display line numbers and the current nesting to improve orientation.
- You can have complete words suggested by the ABAP Editor when you type the first few characters of ABAP keywords and data objects.
- You can have a small pop-up list generated by the new ABAP Editor with suitable suggestions for the current cursor position (since SAP NetWeaver 7.0 EhP2) when you press Ctrl + Spacebar.


Saved Programs
(for further development / testing)
Activated Programs
(for use across the whole system / transport)
Program A
(inactive version)
Program C
(inactive version)
Activate
Program A
(active version)
Program B
(active version)
Program C
(active version)

- Whenever you create or change a development object and then save it, the system stores only one inactive version in the Repository.
- You have an active version and an inactive version of the object. When you complete object development, you have to activate the inactive version (editing version) of the object. This version becomes the new active version of the object.
- The request release and the transport of the developed objects are only possible if all objects in the request have been activated.
- Whenever you activate a program, the system displays a list of all inactive objects that you have processed. This is your work list. Choose those objects that you wish to activate with your current activation transaction.

The activation of an object includes the following functions:

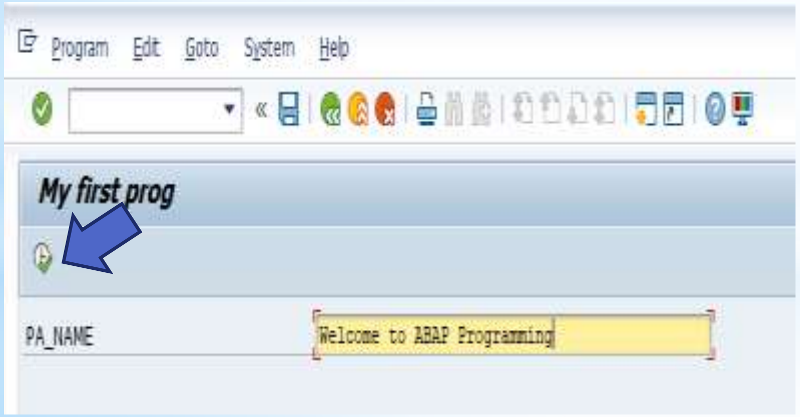
- Saving the object as an inactive version
- Checking the syntax or consistency of the inactive version
- Overwriting the previously active version with the inactive version (only after a successful check)
- Generating the relevant runtime object for later execution, if the object is a program


Saved Programs
(for further development / testing)
Activated Programs
(for use across the whole system / transport)
Program A
(active version)
Program B
(active version)
Program C
(inactive version)
Program C
(active version)
When executed (from the Editor), corresponding runtime object is always regenerated.
When executed, runtime object generated during activation is used.

Example -

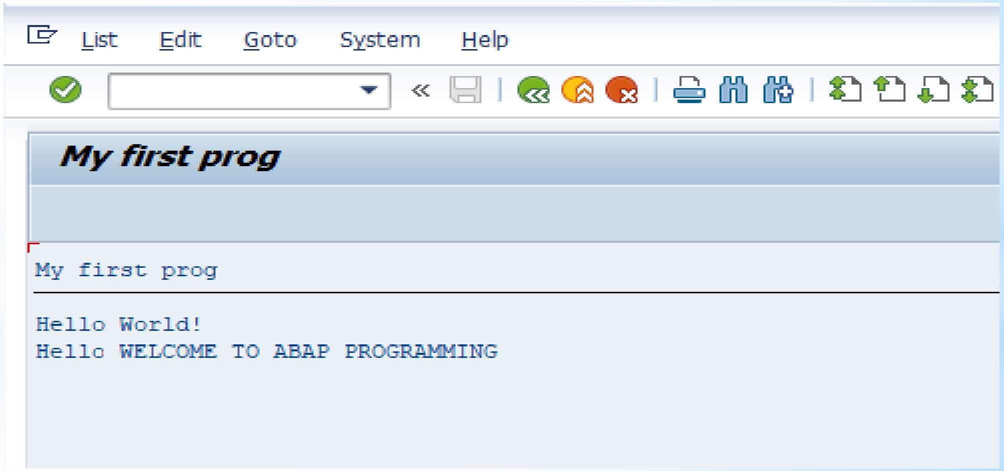
```
REPORT ZDEMO.

PARAMETERS: pa_name TYPE string.

WRITE 'Hello World!'.
NEW-LINE.
WRITE: 'Hello', pa_name.
```

Output :



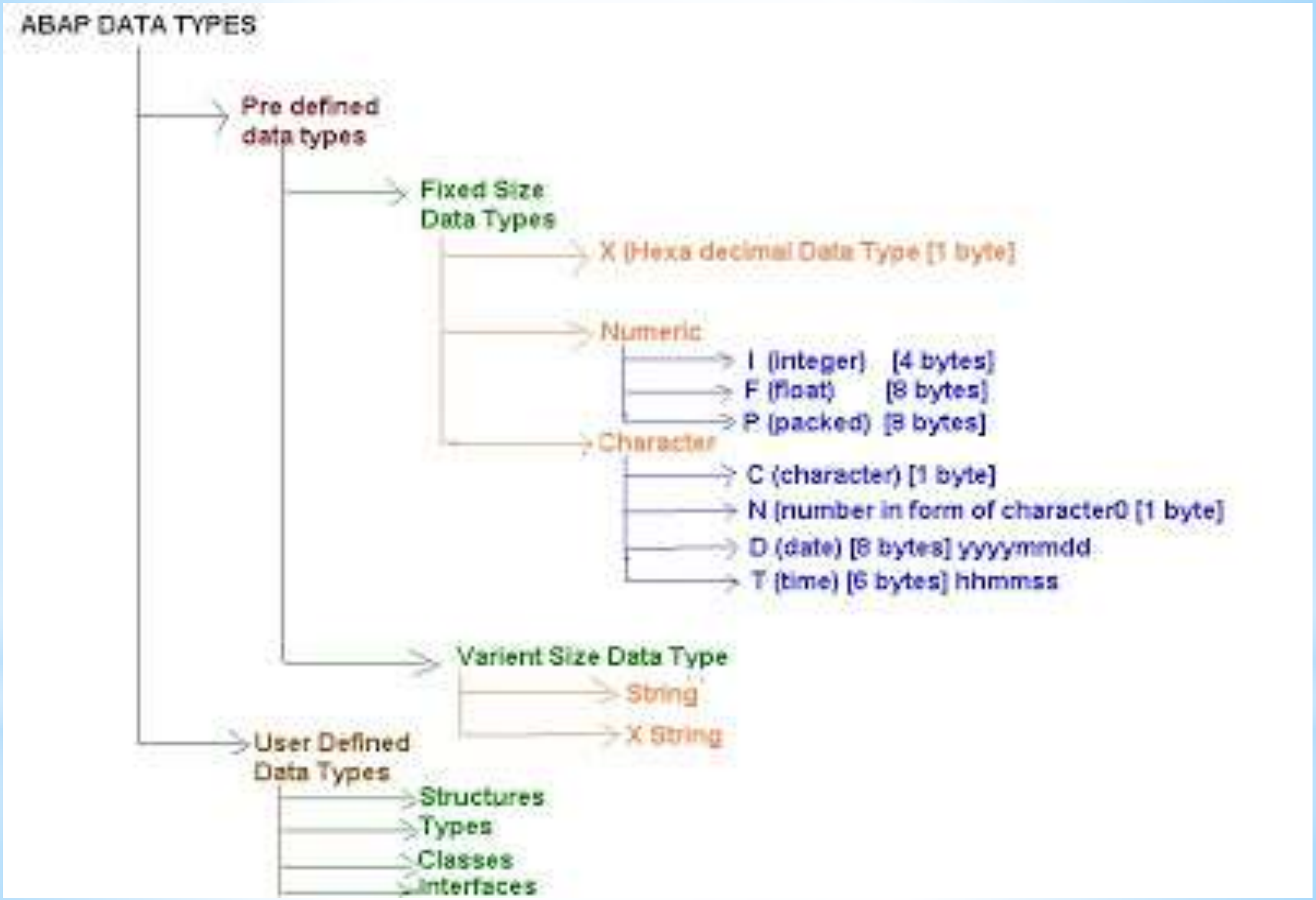
My first prog

PA_NAME Welcome to ABAP Programming



My first prog

```
My first prog

Hello World!
Hello WELCOME TO ABAP PROGRAMMING
```


ABAP DATA TYPES
Pre defined data types
Fixed Size Data Types
X (Hexa decimal Data Type [1 byte]
Numeric
I (integer) [4 bytes]
F (float) [8 bytes]
P (packed) [8 bytes]
Character
C (character) [1 byte]
N (number in form of character0 [1 byte]
D (date) [8 bytes] yyyymmdd
T (time) [6 bytes] hhmmss
Varient Size Data Type
String
X String
User Defined Data Types
Structures
Types
Classes
Interfaces

ABAP Standard Data Types

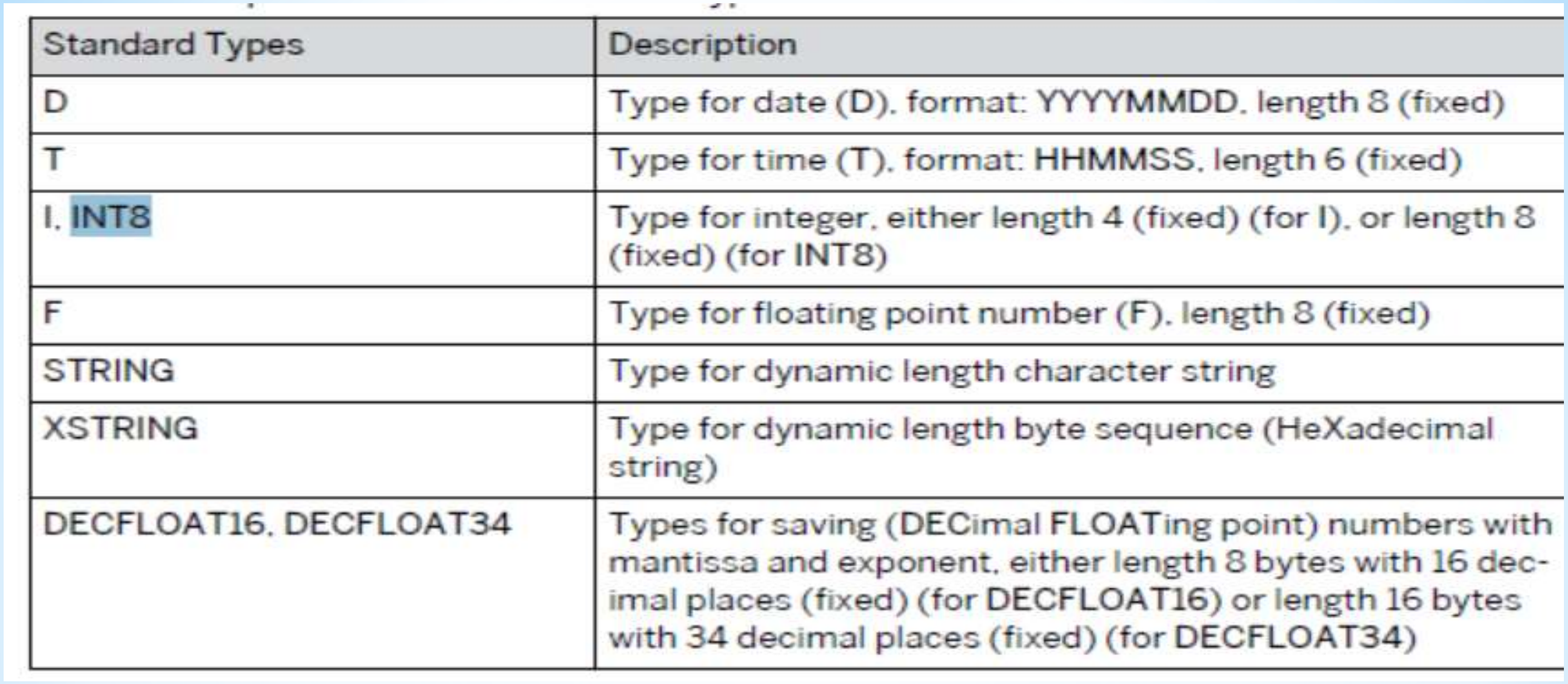
- ➢ ABAP standard data types (built-in data types) are divided into two groups:
- Complete
- Incomplete

Complete ABAP Standard Data Types

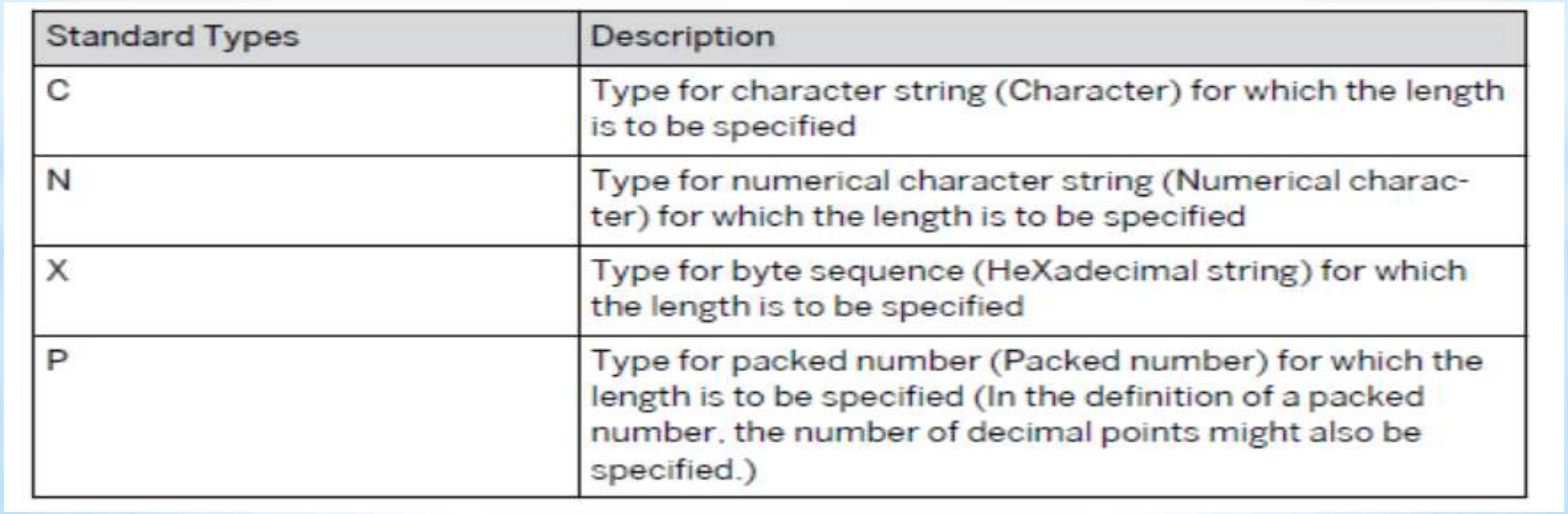
- The built-in ABAP standard data types that already contain a type-specific, fixed-length specification are considered complete data types.

Incomplete ABAP Standard Data Types

- The standard types that do not contain a fixed length are considered incomplete data types. When they are used to define data objects, you need to specify the length of the variable.

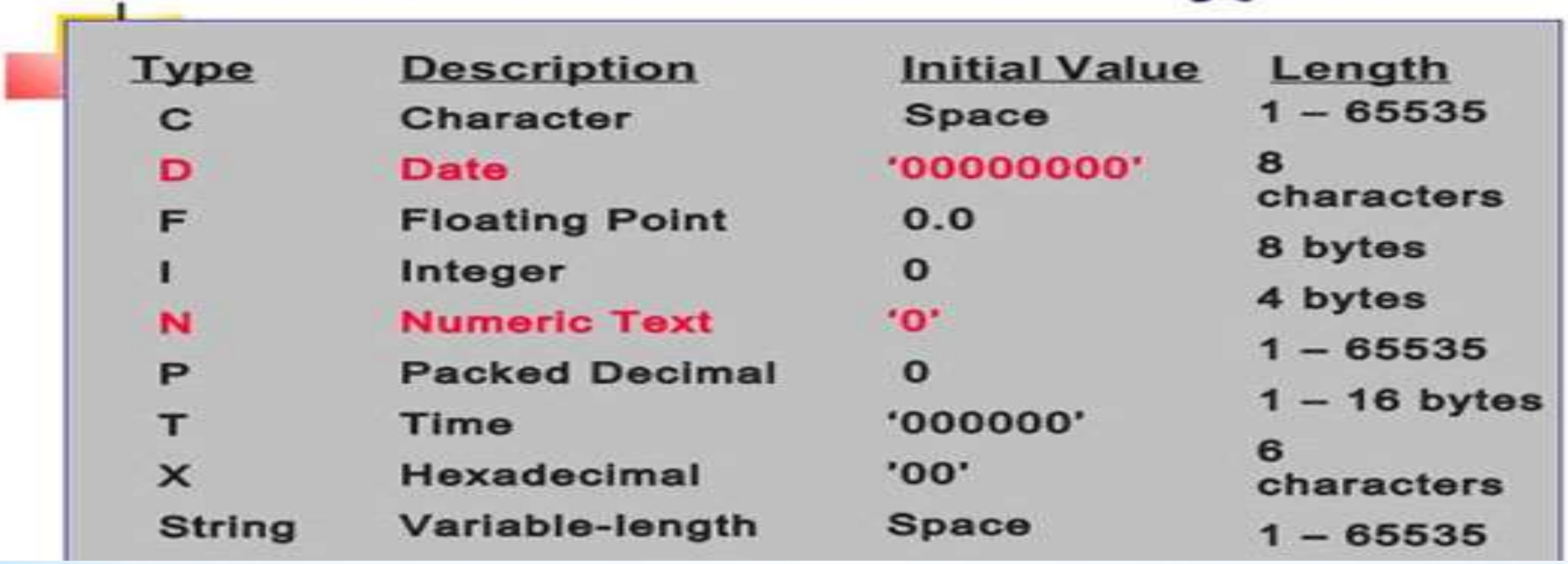


Standard Types	Description
D	Type for date (D), format: YYYYMMDD, length 8 (fixed)
T	Type for time (T), format: HHMMSS, length 6 (fixed)
I, INT8	Type for integer, either length 4 (fixed) (for I), or length 8 (fixed) (for INT8)
F	Type for floating point number (F), length 8 (fixed)
STRING	Type for dynamic length character string
XSTRING	Type for dynamic length byte sequence (HeXadecimal string)
DECFLOAT16, DECFLOAT34	Types for saving (DECimal FLOATing point) numbers with mantissa and exponent, either length 8 bytes with 16 decimal places (fixed) (for DECFLOAT16) or length 16 bytes with 34 decimal places (fixed) (for DECFLOAT34)

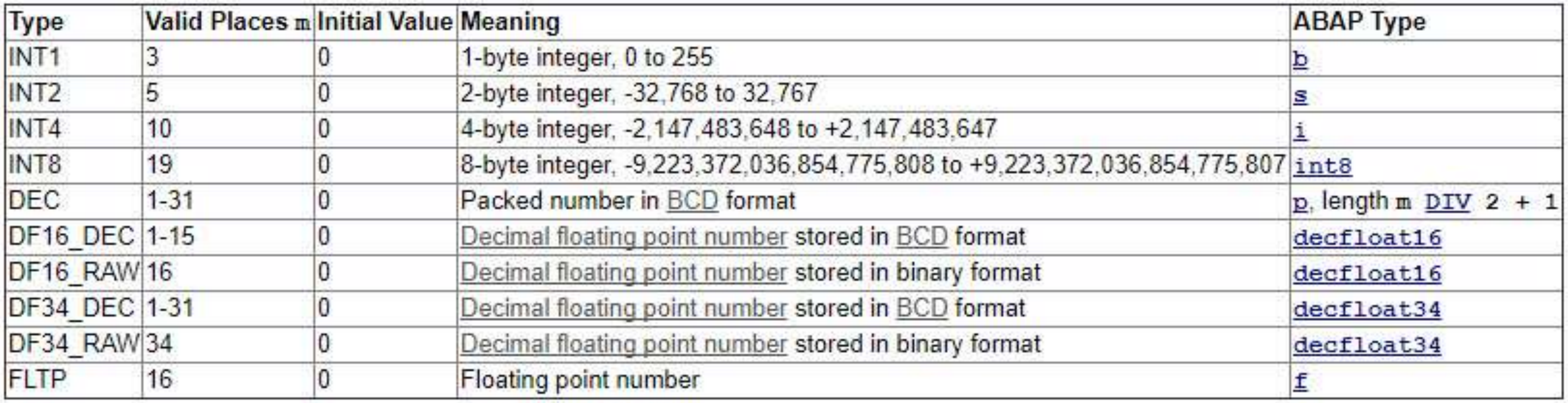


Standard Types	Description
C	Type for character string (Character) for which the length is to be specified
N	Type for numerical character string (Numerical character) for which the length is to be specified
X	Type for byte sequence (HeXadecimal string) for which the length is to be specified
P	Type for packed number (Packed number) for which the length is to be specified (In the definition of a packed number, the number of decimal points might also be specified.)

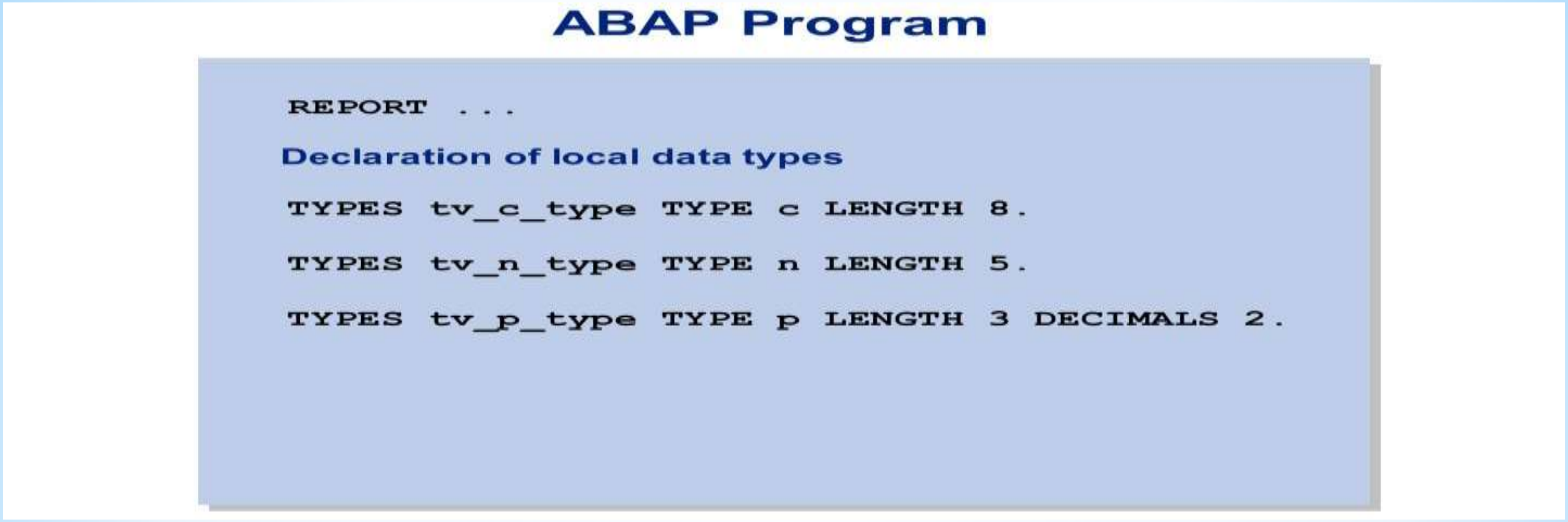
Predefined ABAP Data Types



Type	Description	Initial Value	Length
C	Character	Space	1 – 65535
D	Date	'00000000'	8 characters
F	Floating Point	0.0	8 bytes
I	Integer	0	4 bytes
N	Numeric Text	'0'	1 – 65535
P	Packed Decimal	0	1 – 16 bytes
T	Time	'000000'	6 characters
X	Hexadecimal	'00'	1 – 65535
String	Variable-length	Space	



Type	Valid Places m	Initial Value	Meaning	ABAP Type
INT1	3	0	1-byte integer, 0 to 255	b
INT2	5	0	2-byte integer, -32,768 to 32,767	s
INT4	10	0	4-byte integer, -2,147,483,648 to +2,147,483,647	i
INT8	19	0	8-byte integer, -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807	int8
DEC	1-31	0	Packed number in BCD format	p, length m DIV 2 + 1
DF16_DEC	1-15	0	Decimal floating point number stored in BCD format	decfloat16
DF16_RAW	16	0	Decimal floating point number stored in binary format	decfloat16
DF34_DEC	1-31	0	Decimal floating point number stored in BCD format	decfloat34
DF34_RAW	34	0	Decimal floating point number stored in binary format	decfloat34
FLTP	16	0	Floating point number	f



ABAP Program

```
REPORT ...
```

Declaration of local data types

```
TYPES tv_c_type TYPE c LENGTH 8.
TYPES tv_n_type TYPE n LENGTH 5.
TYPES tv_p_type TYPE p LENGTH 3 DECIMALS 2.
```

OR we can write :

```
TYPES tv_c_type(3) TYPE c.
TYPES tv_p_type(3) TYPE p DECIMALS 2.
```

- Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed, if the condition is determined to be true, and optionally, other statements to be executed, if the condition is determined to be false.

- ABAP programming language provides the following types of decision-making statements.

  1)IF Statement
  An IF statement consists of a logical expression followed by one or more statements.
  2)IF.. Else Statement
  An IF statement can be followed by an optional ELSE statement that executes when the expression is false.
  3)Nested IF Statement
  You may use one IF or ELSEIF statement inside another IF or ELSEIF statement.
  4) CASE Control Statement
  CASE statement is used when we need to compare two or more fields or variables.

- A **loop statement** allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages.

- **There are four kinds of loops in ABAP:**

- Unconditional **loops** using the DO statement.
- Conditional **loops** using the WHILE statement.
- **Loops** through internal tables and extract datasets using the **LOOP** statement.
- **Loops** through datasets from database tables using the SELECT statement.

Unconditional Loops

- To process a statement block several times unconditionally, use the following control structure:

```
DO [n TIMES] ...
  [statement_block]
ENDDO.
```

- Use the **TIMES** addition to restrict the number of loop passes to **n.**
- If you do not specify any additions, the statement block is repeated until it reaches a termination statement such as **EXIT** or **STOP** (see below). The system field **sy-index** contains the number of loop passes, including the current loop pass.

Simple example of a **DO** loop:

```
DO.
WRITE sy-index.
  IF sy-index = 3.
    EXIT.
  ENDIF.
ENDDO.
```

The output is:

```
    1       2       3
```

- The loop is processed three times. Here, the processing passes through the loop three times and then leaves it after the **EXIT** statement.

Example of two nested loops with the **TIMES** addition:

```
DO 2 TIMES.
  WRITE sy-index.
  SKIP.
  DO 3 TIMES.
    WRITE sy-index.
  ENDDO.
  SKIP.
ENDDO.
```

The output is:

```
    1
    1     2     3
    2
    1     2     3
```

- The outer loop is processed twice. Each time the outer loop is processed, the inner loop is processed three times. Note that the system field **sy-index** contains the number of loop passes for each loop individually.

Conditional Loops

- To repeat a statement block for as long as a certain condition is true, use the following control structure:

```
WHILE log_exp
  [statemaent_block]
ENDWHILE.
```

log_exp can be any logical expression.

- The statement block between **WHILE** and **ENDWHILE** is repeated as long as the condition is true or until a termination statement such as **EXIT** or **STOP** occurs.

- The system field **sy-index** contains the number of loop passes, including the current loop pass.

- You can nest **WHILE** loops to any depth, and combine them with other loop forms.

```
REPORT demo_flow_control_while.
DATA: length     TYPE i VALUE 0,
      strl       TYPE i VALUE 0,
      string(30) TYPE c VALUE 'Test String'.
strl = strlen( string ).
WHILE string NE space.
  WRITE string(1).
  length = sy-index.
  SHIFT string.
ENDWHILE.
WRITE: / 'STRLEN:          ', strl.
WRITE: / 'Length of string:', length.
```

The output appears as follows:

```
T e s t   S t r i n g
STRLEN:                  11
Length of String:        11
```

- Here, a **WHILE** loop is used to determine the length of a character string. This is done by shifting the string one position to the left each time the loop is processed until it contains only blanks. This example has been chosen to demonstrate the **WHILE** statement. Of course, you can determine the length of the string far more easily and efficiently using the **strlen** function.

Terminating Loops

- ABAP contains termination statements that allow you to terminate a loop prematurely. There are two categories of termination statement - those that only apply to the loop, and those that apply to the entire processing block in which the loop occurs.
- The termination statements that apply only to the loop in which they occur are **CONTINUE**, **CHECK** and **EXIT**.
- You can only use the **CONTINUE** statement in a loop. **CHECK** and **EXIT**, on the other hand, are context-sensitive.

- Within a loop, they only apply to the execution of the loop itself. Outside of a loop, they terminate the entire processing block in which they occur (subroutine, dialog module, event block, and so on).
- **CONTINUE**, **CHECK** and **EXIT** can be used in all four loop types in ABAP (**DO**, **WHILE**, **LOOP** and **SELECT**).

Terminating a Loop Pass Unconditionally

- To terminate a single loop pass immediately and unconditionally, use the **CONTINUE** statement in the statement block of the loop.

CONTINUE.

- After the statement, the system ignores any remaining statements in the current statement block, and starts the next loop pass.

Example -

```
DO 4 TIMES.
  IF sy-index = 2.
    CONTINUE.
  ENDIF.
  WRITE sy-index.
ENDDO.
```

The output is:

```
        1          3          4
```

- The second loop pass is terminated without the **WRITE** statement being processed.

Terminating a Loop Pass Conditionally

- To terminate a single loop pass conditionally, use the **CHECK condition** statement in the statement block of the loop.

CHECK condition.

- If the condition is not true, any remaining statements in the current statement block after the **CHECK** statement are ignored, and the next loop pass starts. **condition** can be any logical expression.

Example -

```
DO 4 TIMES.
  CHECK sy-index BETWEEN 2 and 3.
  WRITE sy-index.
ENDDO.
```

The output is:

```
        2          3
```

- The first and fourth loop passes are terminated without the **WRITE** statement being processed, because **sy-index** is not between 2 and 3.

Exiting a Loop

- To terminate an entire loop immediately and unconditionally, use the **EXIT** statement in the statement block of the loop.

EXIT.

- After this statement, the loop is terminated, and processing resumes after the closing statement of the loop structure (**ENDDO**, **ENDWHILE**, **ENDLOOP**, **ENDSELECT**).
- In nested loops, only the current loop is terminated.

EXAMPLE-

```
DO 4 TIMES.
  IF sy-index = 3.
    EXIT.
  ENDIF.
  WRITE sy-index.
ENDDO.
```

The output is:

```
         1          2
```

- In the third loop pass, the loop is terminated before the **WRITE** statement is processed.

Thank You