ABAP RESTful Application Programming Model

Mentors Pool

Choose Your IT Expert

RAP ABAP

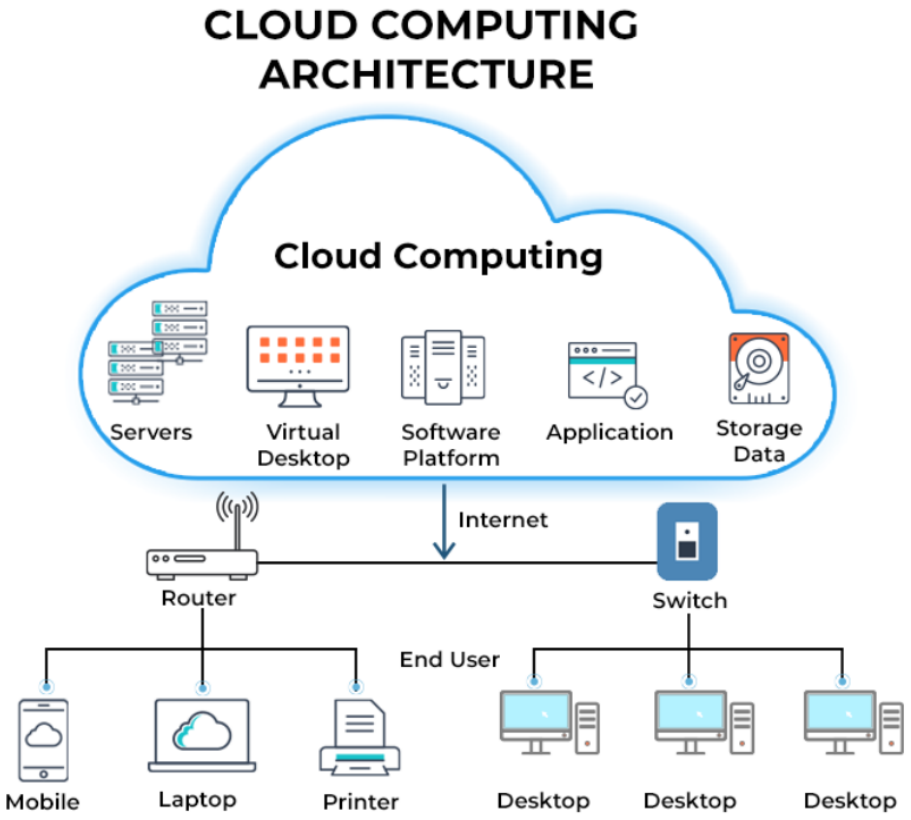
The Big Picture

1. Introduction to SAP BTP
2. [Beginner] Build SAP Fiori Apps with the ABAP RESTful Application Programming Model (RAP)
3. [Intermediate] Build SAP Fiori Apps with the ABAP RESTful Application Programming Model (RAP)
4. Use ABAP Cloud for SAP S/4HANA (Cloud) (new app on stack)
5. Side-by-side extensibility with SAP BTP ABAP Environment
6. Use ABAP Cloud for developer extensibility (extend an SAP app)

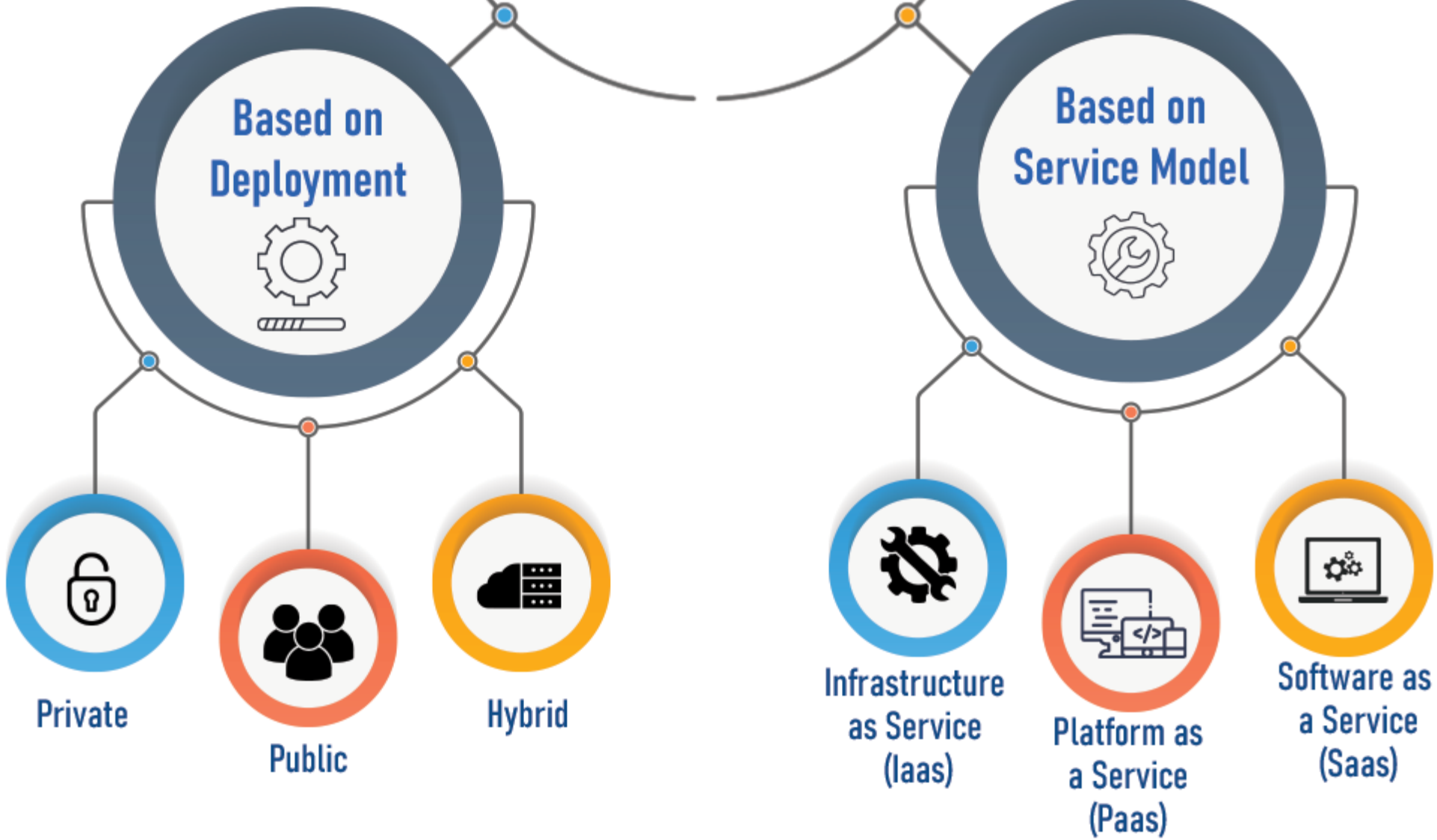
- Cloud Computing
- BTP Overview
- RAP Introduction
- Behaviour Definition Concept
- Behaviour Implementation
- Entity Manipulation Language
- Managed Processor App
- Actions Determination
- Introduce Draft Augment
- Approver Scenario
- Custom Entity
- ABAP Git
- Authorization & Authentication in RAP
- Live Build
- Dynamic Control
- Difference Between CAP & RAP
- Consuming External API's

What Is Cloud Computing?

Cloud computing refers to the use of hosted services, such as data storage, servers, databases, networking, and software over the internet. The data is stored on physical servers, which are maintained by a cloud service provider. Computer system resources, especially data storage and computing power, are available on-demand, without direct management by the user in cloud computing.



TYPES OF CLOUD COMPUTING



SAP BTP Cloud Services

Enterprise readiness with holistic lifecycle management

- Change and deployment management
- Technical ops automation
- Alerting
- Smooth integration option into existing ALM processes

Business Technology Platform

Discovery Center

Actionable & guided missions | Service catalog | Best practice enablement & learning content

Ecosystem and Marketplace

Integration packs | Pre-built industry & LoB content | Business workflows & bots | Business apps- & third-party data

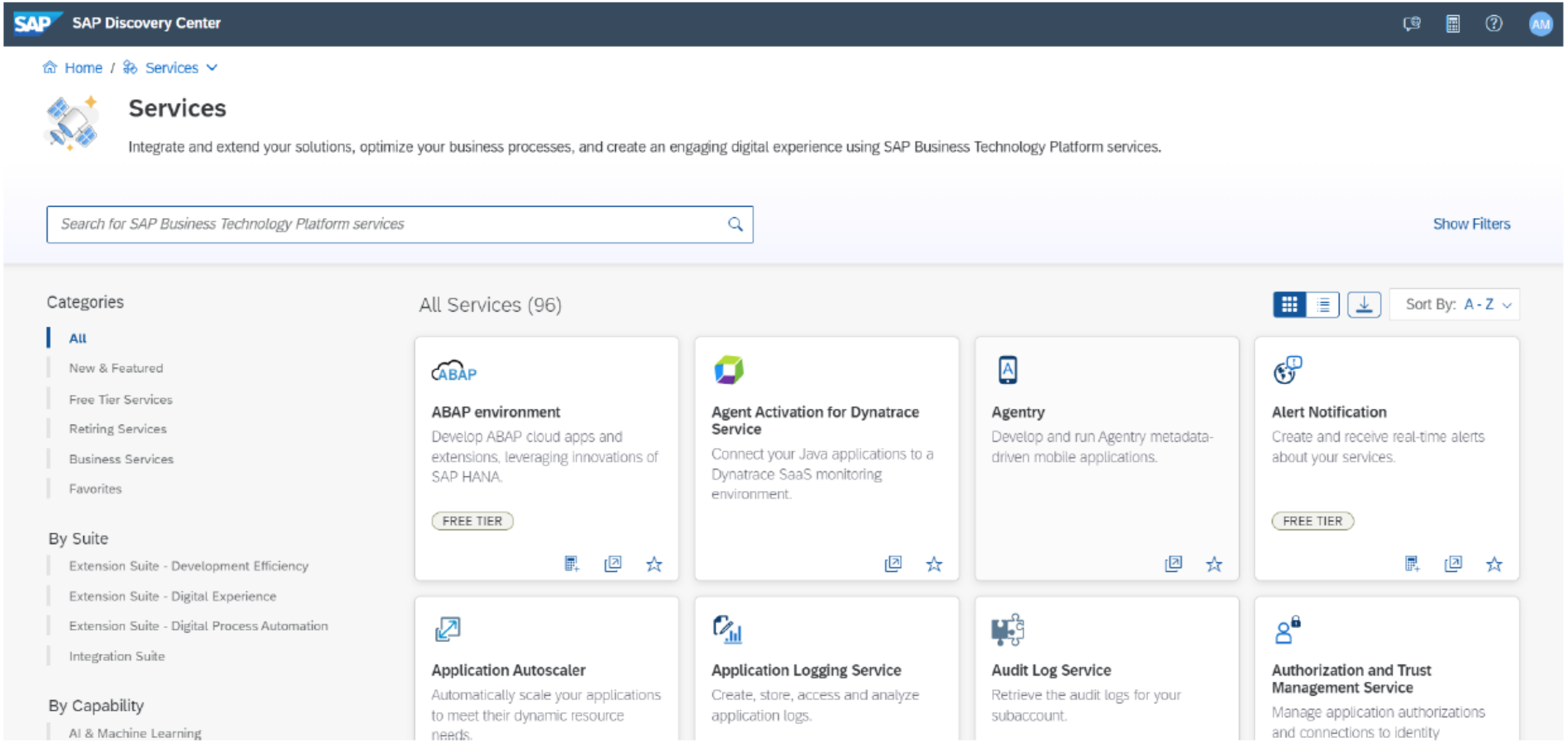
App Dev	Automation	Integration	Data and Analytics	AI
→ SAP Build Apps → SAP Business Application Studio • Mobile Services • Runtimes (Cloud Foundry, Kyma) • Document Management Services → SAP BTP ABAP Environment → SAP Build Work Zone	→ SAP Build Process Automation • Workflow Management • Decisions & Rules • RPA bots • Process Visibility → SAP Task Center	→ SAP API Business Hub → SAP Integration Suite • Cloud Integration • API Management • Open Connectors • Integration Adviser • Integration Assessment • Event Mesh • Trading Partner Management → SAP Master Data Integration → SAP Advanced Event Mesh	→ SAP Analytics Cloud • Business Intelligence • Augmented Analytics • Enterprise Planning → SAP Data Warehouse Cloud → SAP Data Intelligence Cloud → SAP Master Data Governance → SAP HANA Cloud	→ SAP AI Business Services • Business Entity Recognition • Data Attribute Recommendation • Document Classification • Document Information Extraction • Service Ticket Intelligence • Personalized Recommendation → SAP AI Core & SAP AI Launchpad → SAP Conversational AI

Enterprise security services and hybrid identity & access management

- Cloud identity services
- Risk & compliance
- Secure development

SAP Business Technology Platform Services

Visit SAP Discovery Center

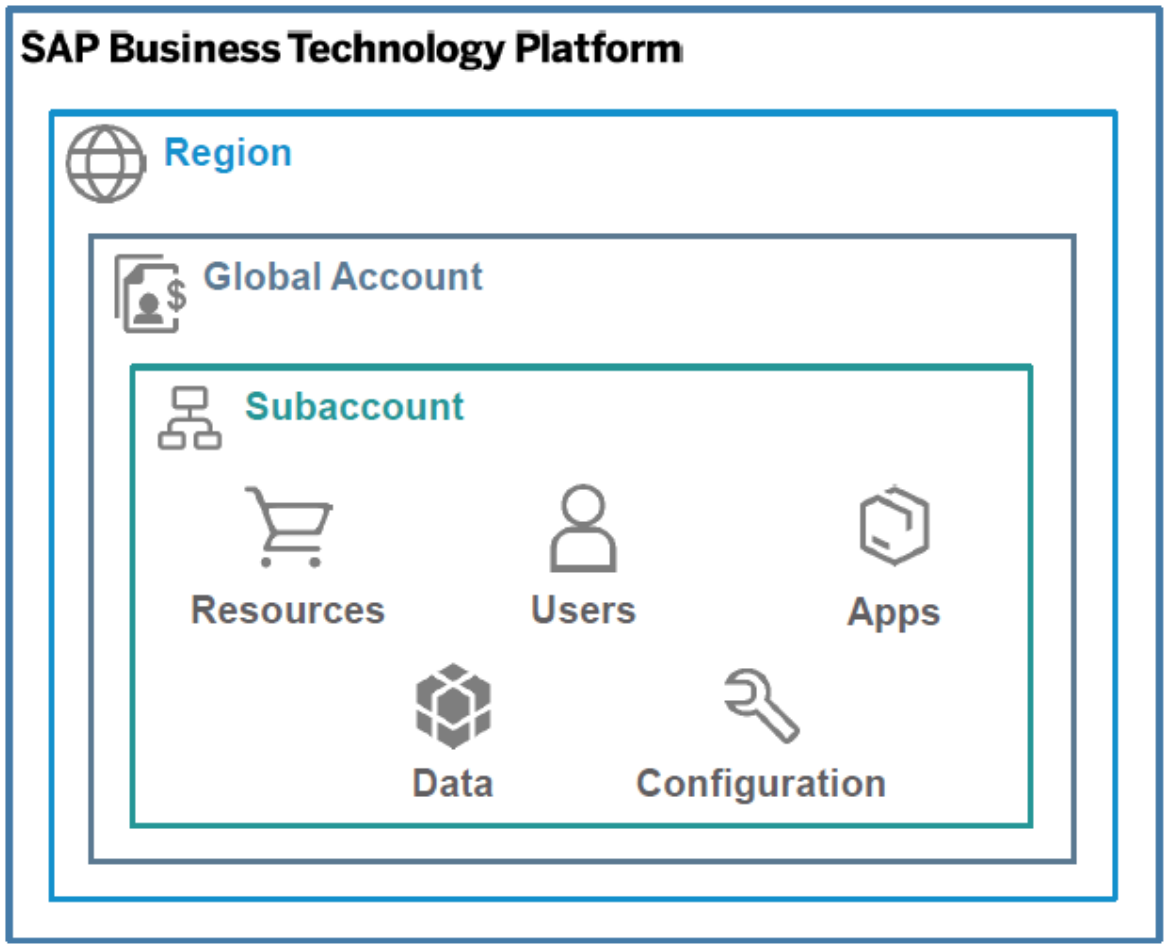


https://discovery-center.cloud.sap/viewServices?provider=all&regions=all

https://account.hana.ondemand.com/#/home/welcome

https://developers.sap.com/tutorials/abap-environment-trial-onboarding.html

Understanding SAP Business Technology Platform Subaccounts

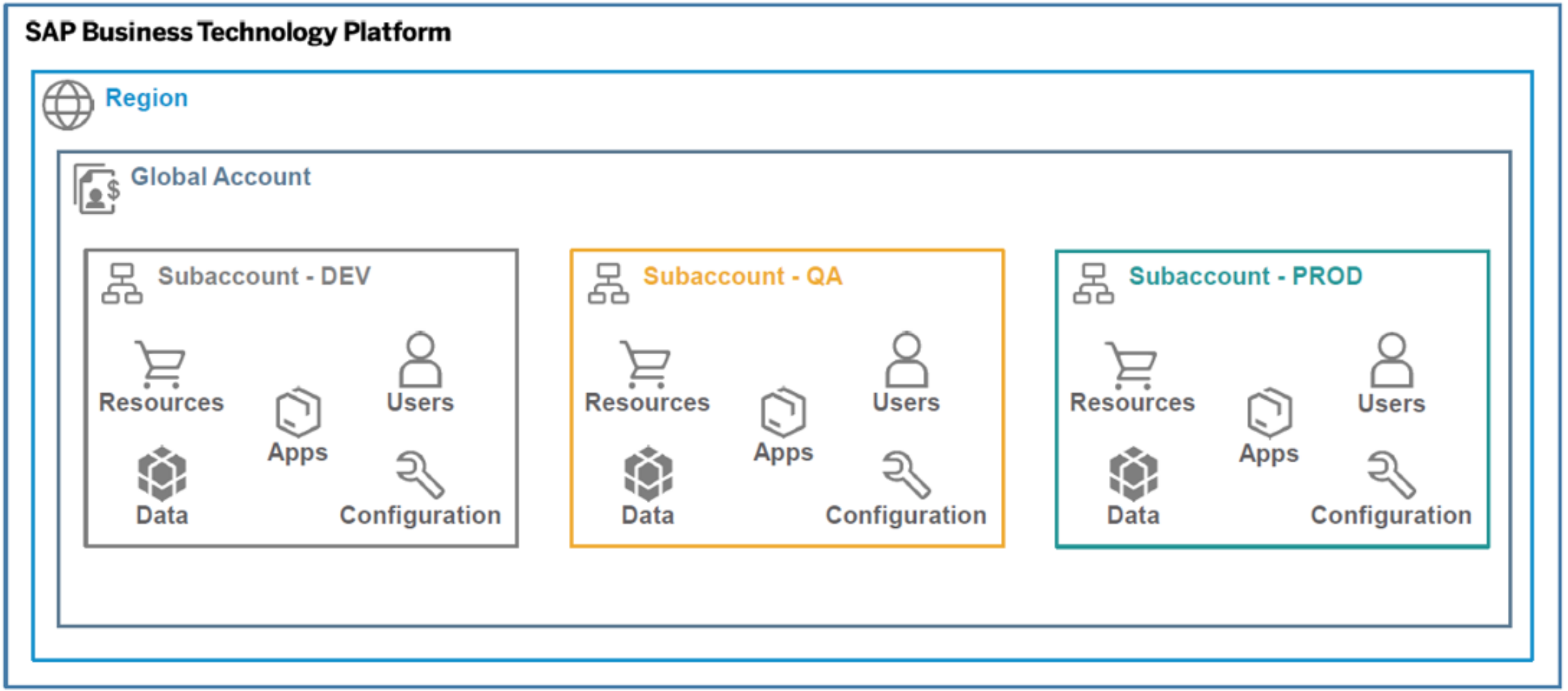


Each subaccount holds:

- Resources that can be consumed by apps
- Users allowed to work in the subaccount
- Apps deployed and running in the subaccount
- Data written by apps running in the subaccount
- Configuration for apps running in the subaccount

Each subaccount is assigned to a Global account and resides in a Region.

Landscape Design

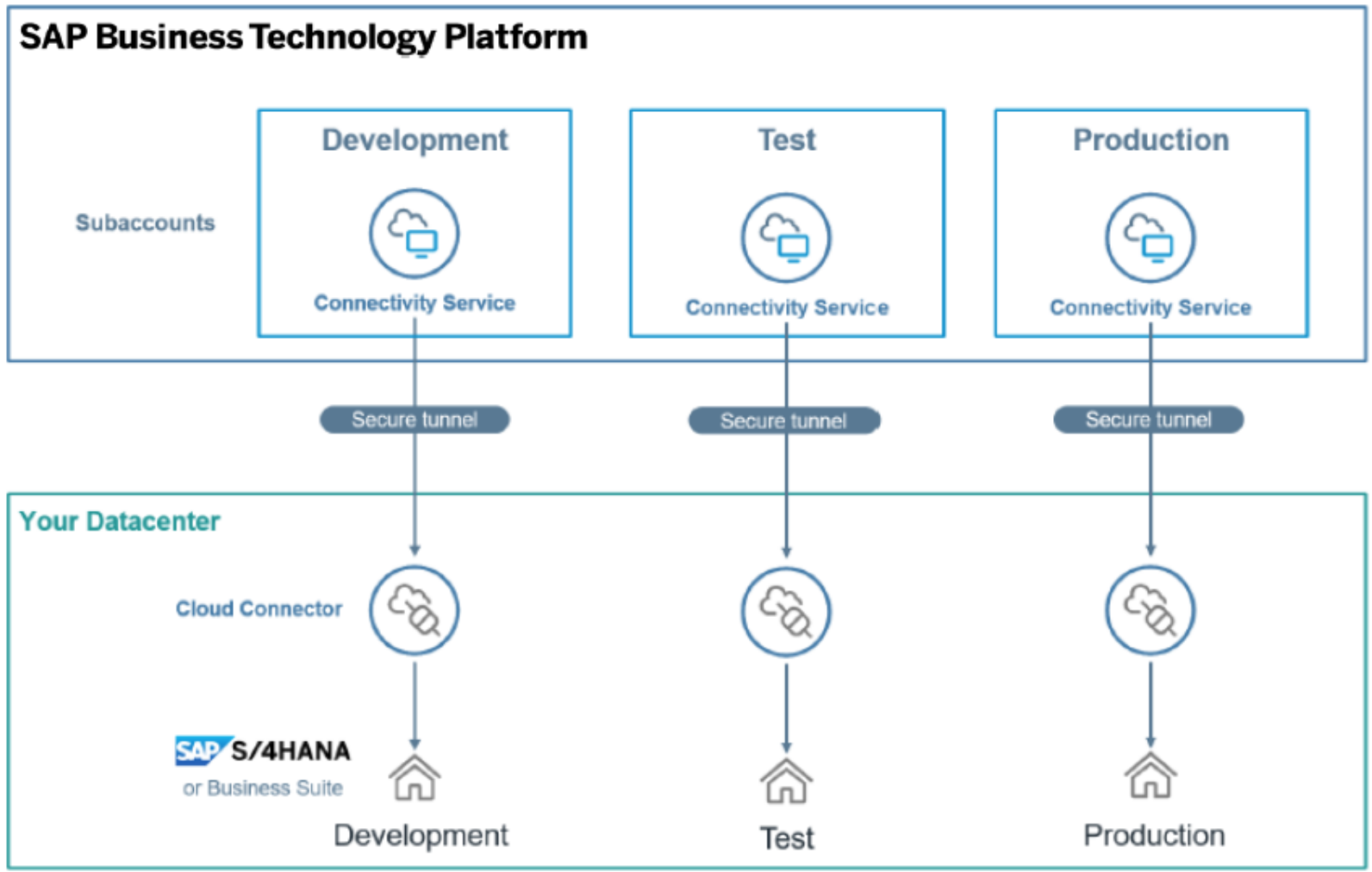


Stages are important in the cloud as well

Subaccounts or Spaces for staging

Different Cloud Connectors for each stage

For extension-scenarios: “Mimic“ on-prem landscape with cloud subaccounts



For more info refer to [Setting Up Your Account Model]

https://help.sap.com/docs/btp/best-practices/setting-up-your-account-model

SAP® Cloud Application Programming Model Versus ABAP® RESTful Application Programming Model

Common to Both Models

- RESTful OData services for communication between the front end (UI) and the application on SAP® Business Technology Platform (SAP BTP)
- Core data services (CDS) language to define the data model, business objects, and their attributes and behavior, and the generation of RESTful OData services from these definitions
- The universal nature of CDS views, which can be connected to any data source to fetch or change the data: SAP HANA® database, API of an SAP solution, or third-party REST service
- Built-in extensibility capabilities, enabling users to extend both SAP Cloud Application Programming Model and ABAP® RESTful application programming model in a similar way as with the key user (in-app) extensibility of SAP S/4HANA® (see above)

ABAP® RESTful Application Programming Model

- Supports the ABAP programming language
- Provides Git-enabled lifecycle management
- Offers a service consumption model for easy remote OData service calls
- Enables the possibility to reuse selected custom code in the cloud with SAP BTP, ABAP environment, while rebuilding the UI and back-end access
- Leverages modern ABAP development tools in Eclipse for back-end development on premise or in the cloud

SAP® Cloud Application Programming Model

- Supports Java or JavaScript (node.js)
- Enables applications originally written as single-tenant applications to be turned into multitenant ones through configuration
- Leverages event-based communication using the SAP Event Mesh capability
- Wraps the REST service calls to the underlying back-end system to Java or JavaScript functions using SAP Cloud SDK

An **ABAP development** model to build cloud-ready business apps, services, and extensions

Available on **SAP BTP, ABAP environment**, **SAP S/4HANA Cloud**, and **SAP S/4HANA**

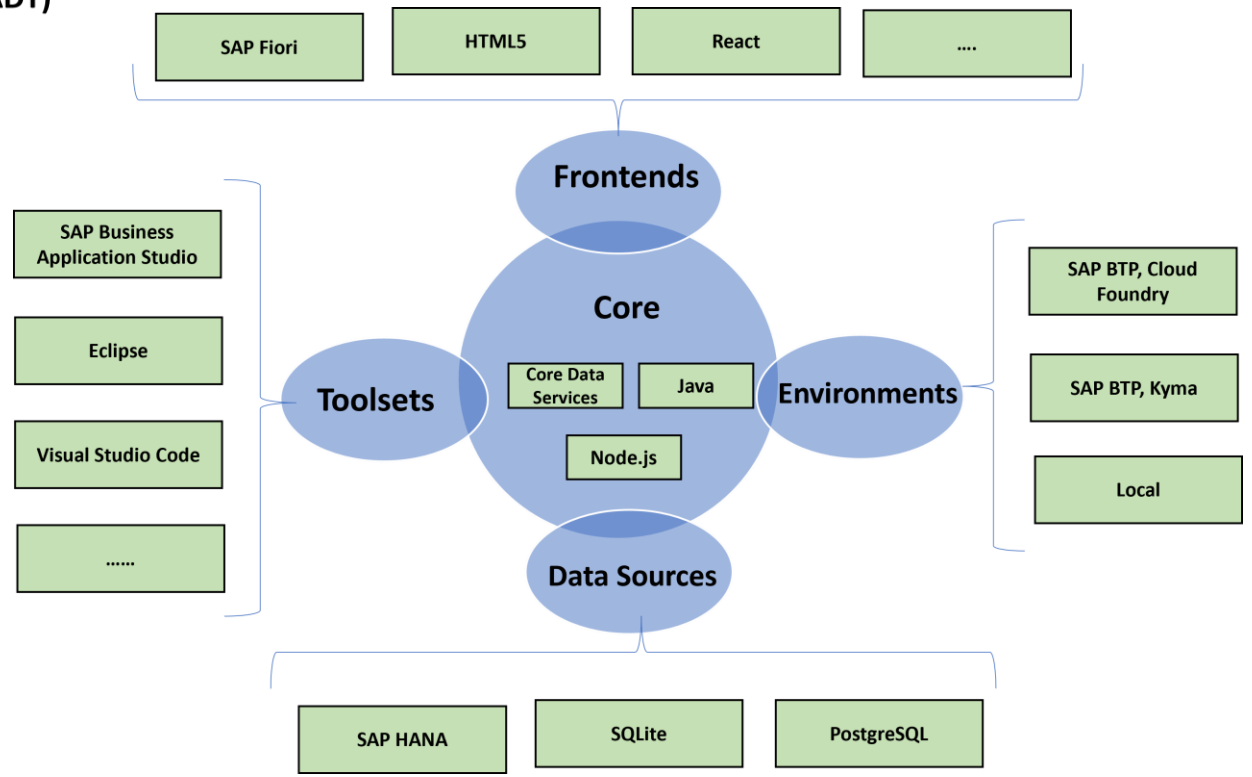
A set of **concepts, tools, languages**, and **powerful frameworks**

RAP

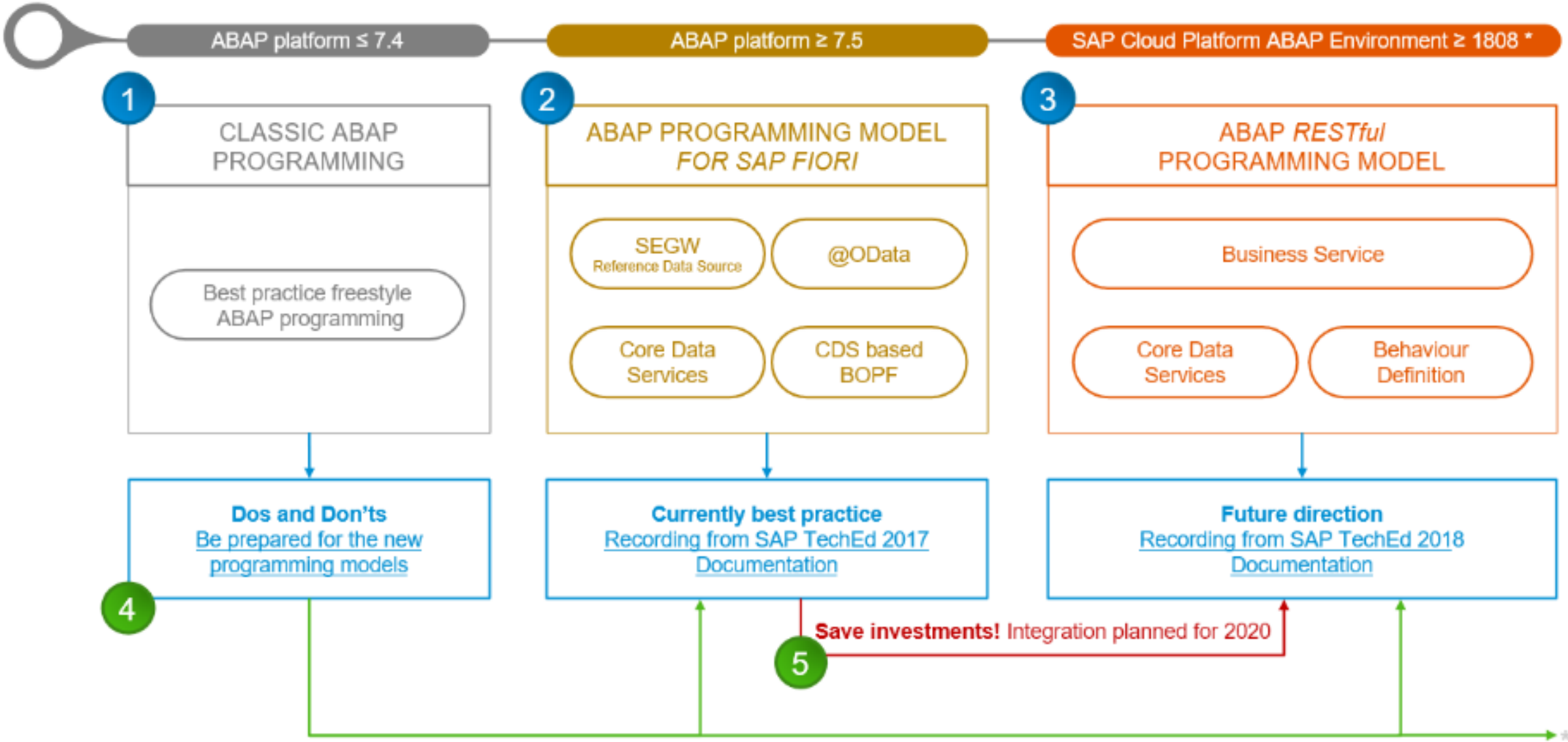
offers

- **Core Data Services (CDS)**
- **Modernized and extended ABAP language**
- Concept of **business object (BO), Business Service** etc.
- **Eclipse-based ABAP Development Tools (ADT)**

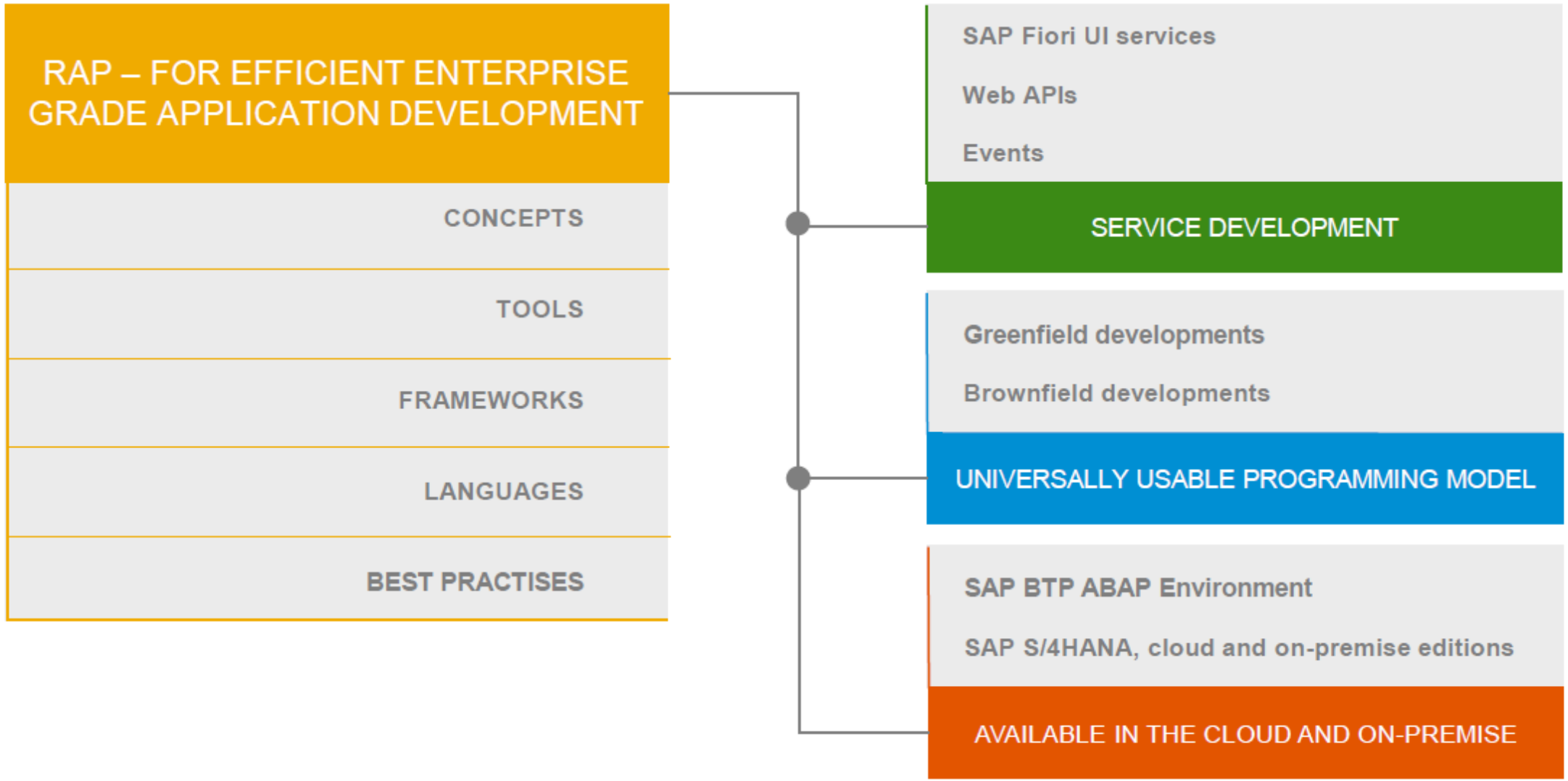
RAP V/S CAP



Evolution of the ABAP programming model



ABAP RESTful Application Programming Model (RAP) – At a glance



The key players

ABAP Development Tools in Eclipse for all development tasks

Easy developer onboarding
End-to-end development flow

Languages: ABAP and Core Data Services (CDS)

Standard implementation tasks via typed APIs supporting
static code checks, auto-completion, element info

Powerful frameworks

Take over technical implementation tasks
Business logic added in code exits on protocol-agnostic layers

ABAP RESTful Application Programming Model (RAP) – The big picture

CONSUMPTION

DATA INTEGRATION
Consume SQL based services

EVENTS
Consume business events

SAP FIORI UIs & WEB APIs
Consume OData based services

SAP ANALYTICS CLOUD
Consume InA based UI services for live data access

BUSINESS SERVICE PROVISIONING

SERVICE BINDING – Bind to protocol version and scenario

SERVICE DEFINITION – Define scope to be exposed

BUSINESS OBJECT PROJECTION AND INTERFACES
- CDS: BO projection views
- BDEF: Behavior projection
- ABAP: Behavior implementation[1]

ANALYTICAL QUERIES
- CDS: Analytical projection views

DATA MODELING & BEHAVIOR

CDS ENTITIES
- CDS: Data modeling

BUSINESS OBJECTS
- CDS: Data modeling
- BDEF: Behavior definition
- ABAP: Behavior implementation

ANALYTICAL MODEL
- CDS: Analytical cubes
- CDS: Analytical dimensions

EXTENSIBILITY

CDS ABAP/EML

[1] Not applicable for RAP BO interfaces

Core Concepts in RAP

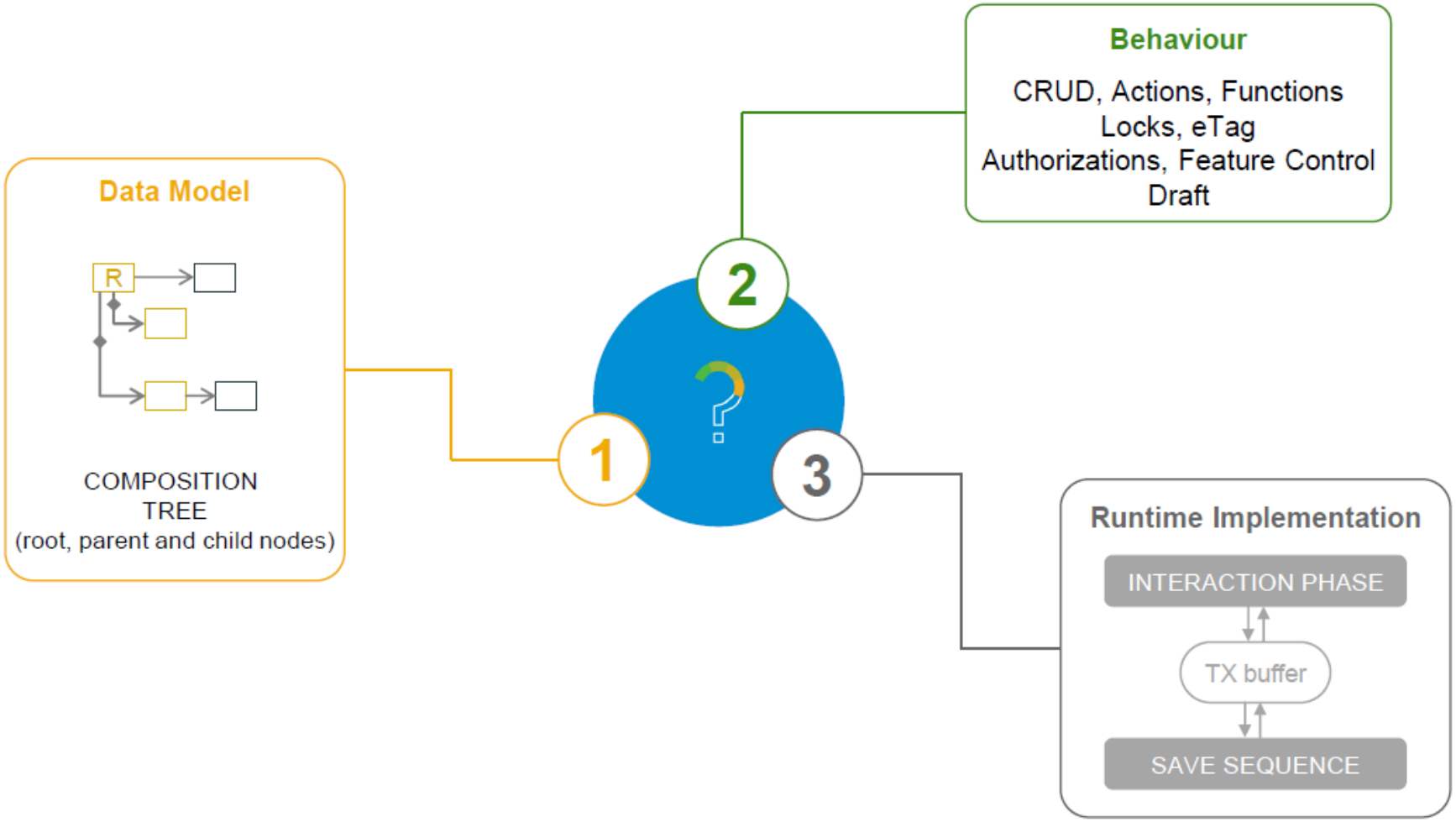
- Business Object
- Query
- Business Service

Implementation Languages

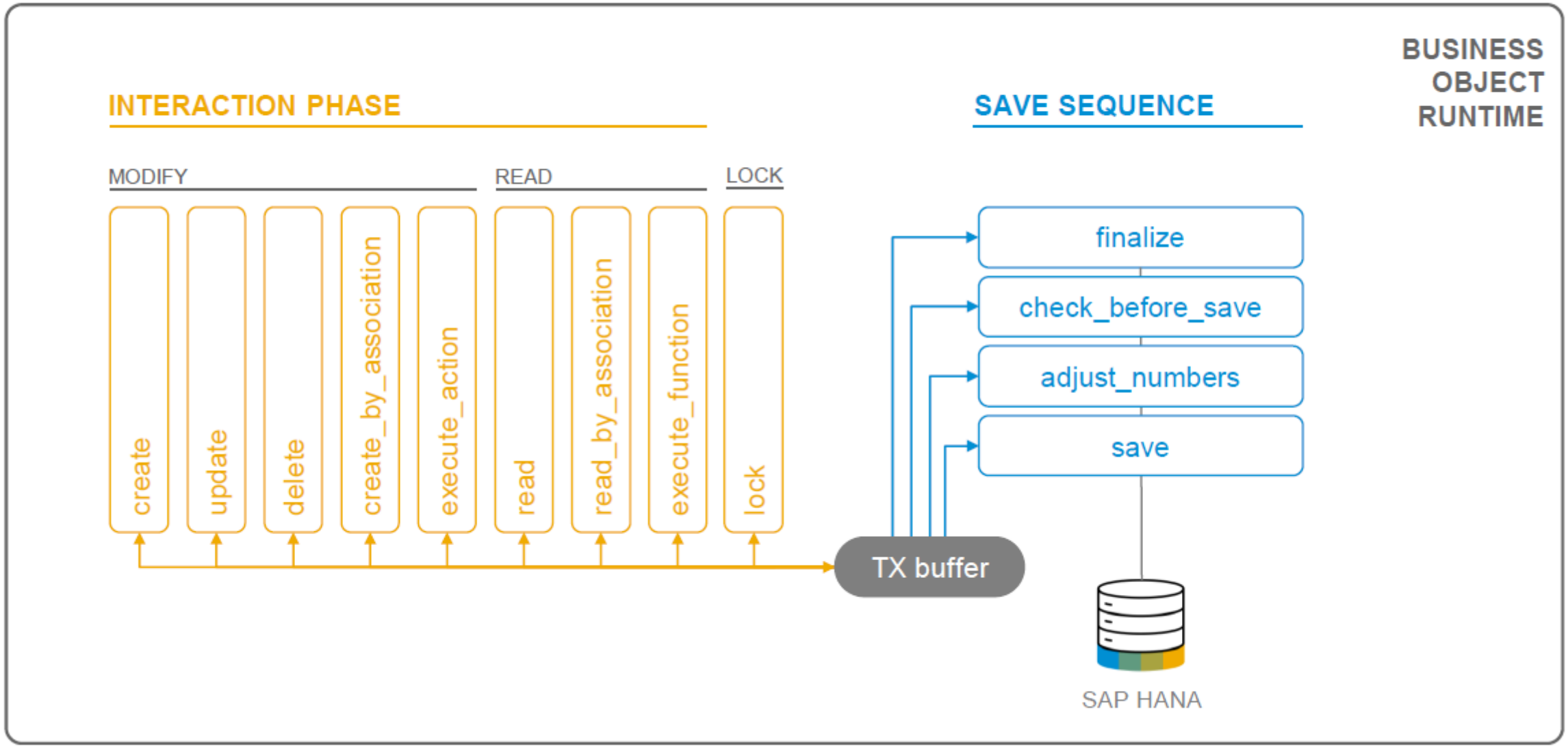
- ABAP Core Data Services (CDS)
- Modern ABAP Language
- Entity Manipulation Language (EML)

Key Concepts

Business object

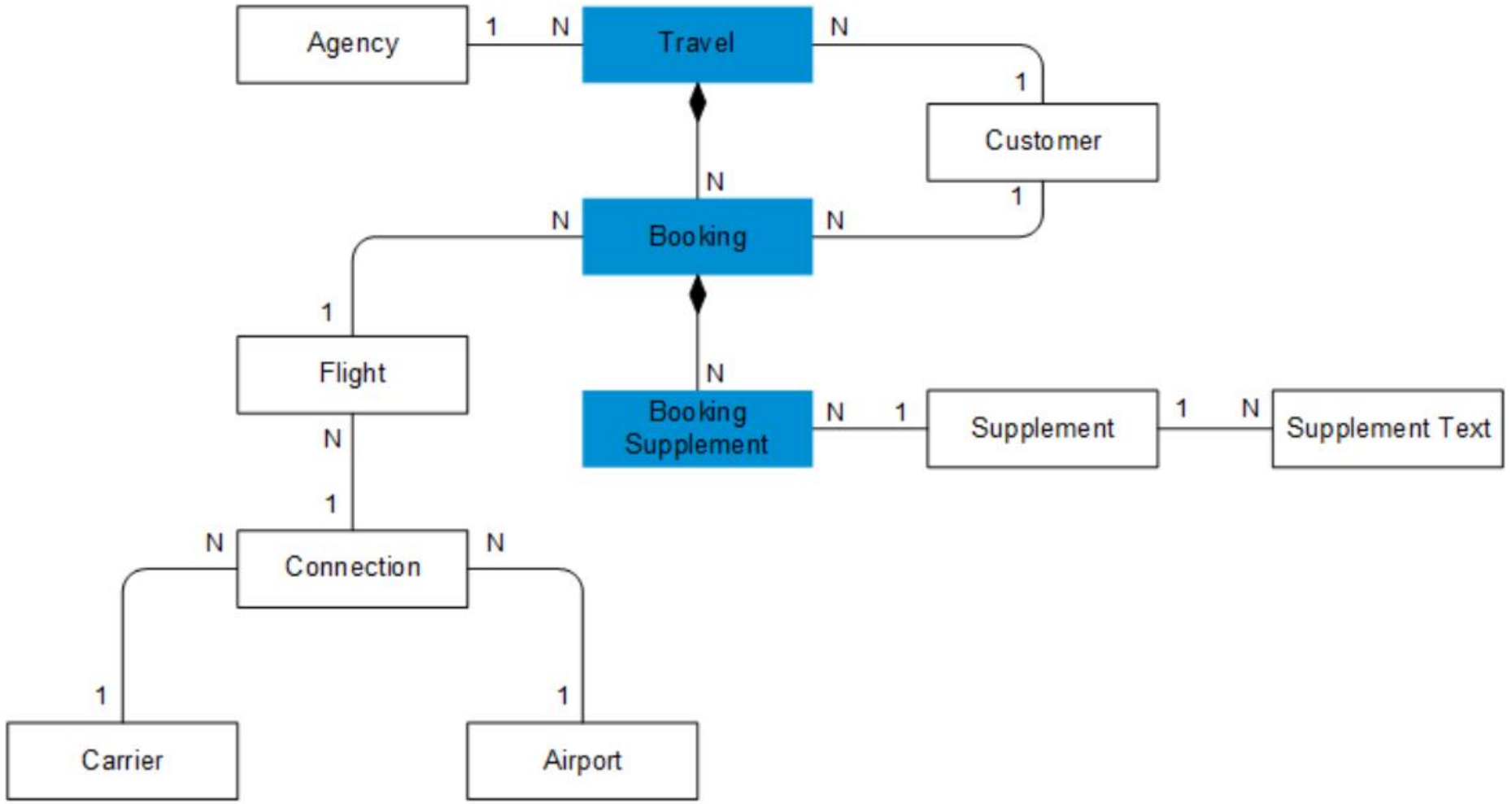


Business Object Runtime

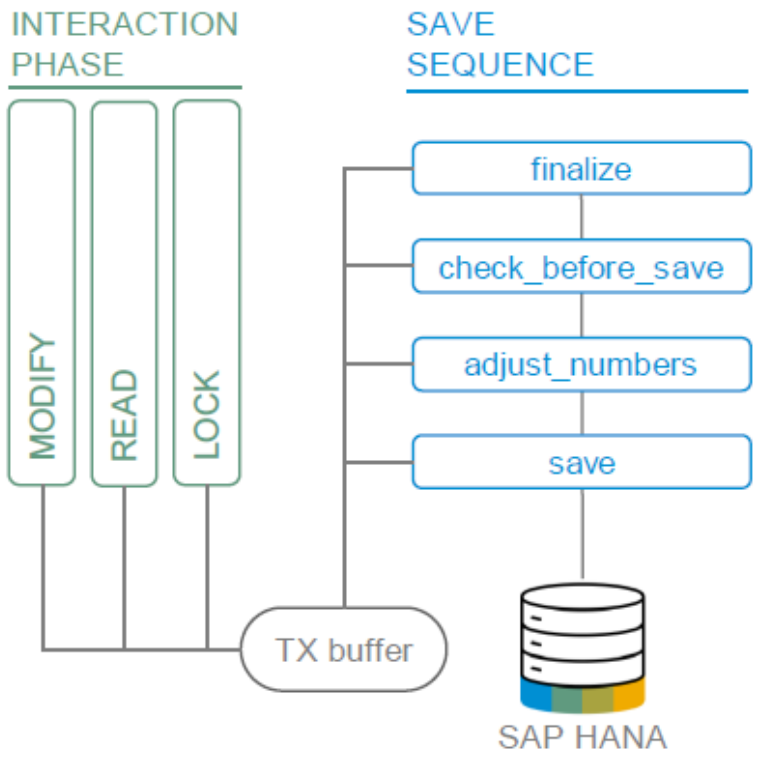


Key Concepts

Data Model



Business Objects – Status Overview/ Implementation types



1 UNMANAGED

Application coding available
(e.g. Purchase Order, Sales Order,…)

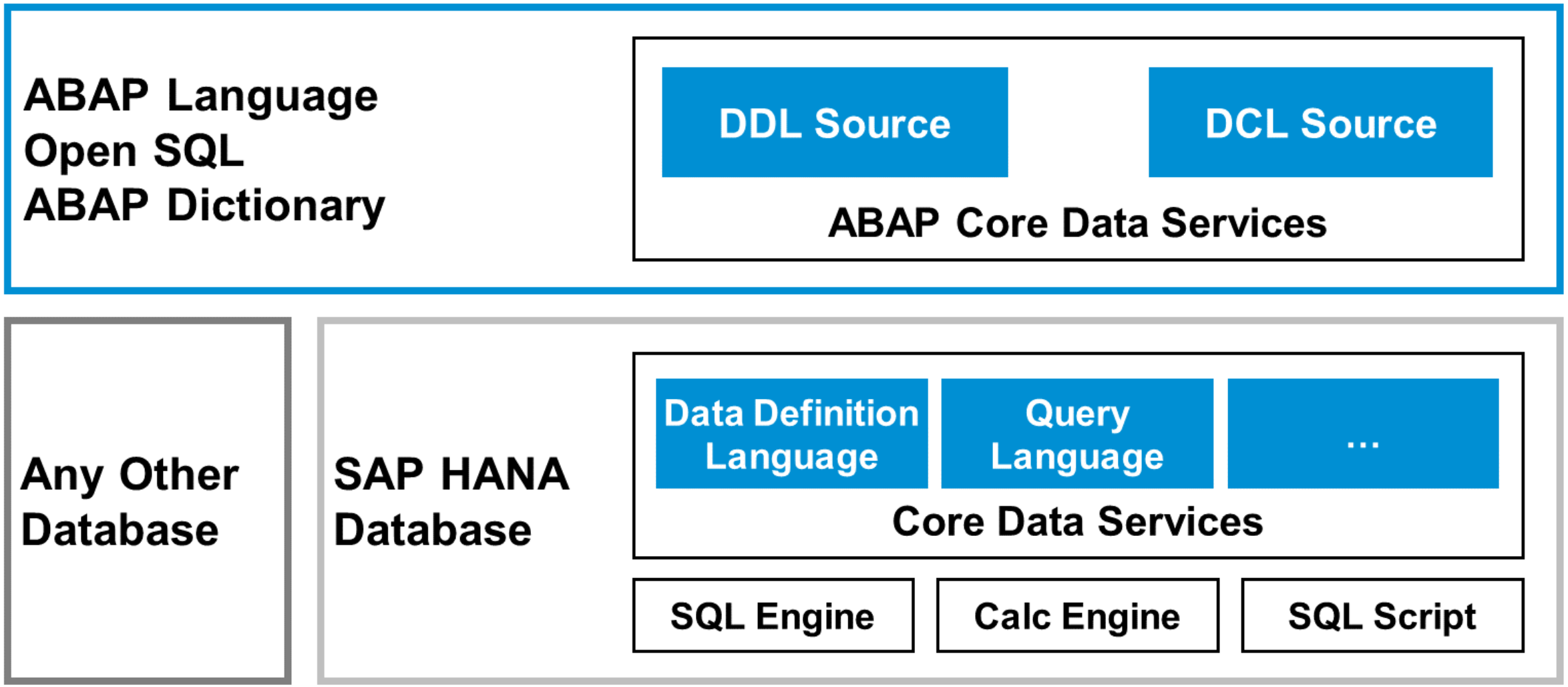
2 MANAGED

Green field development

3 (MANAGED WITH) SAVE SELF-IMPLEMENTED

Update task function module available
(e.g. Business Partner, Product,..)

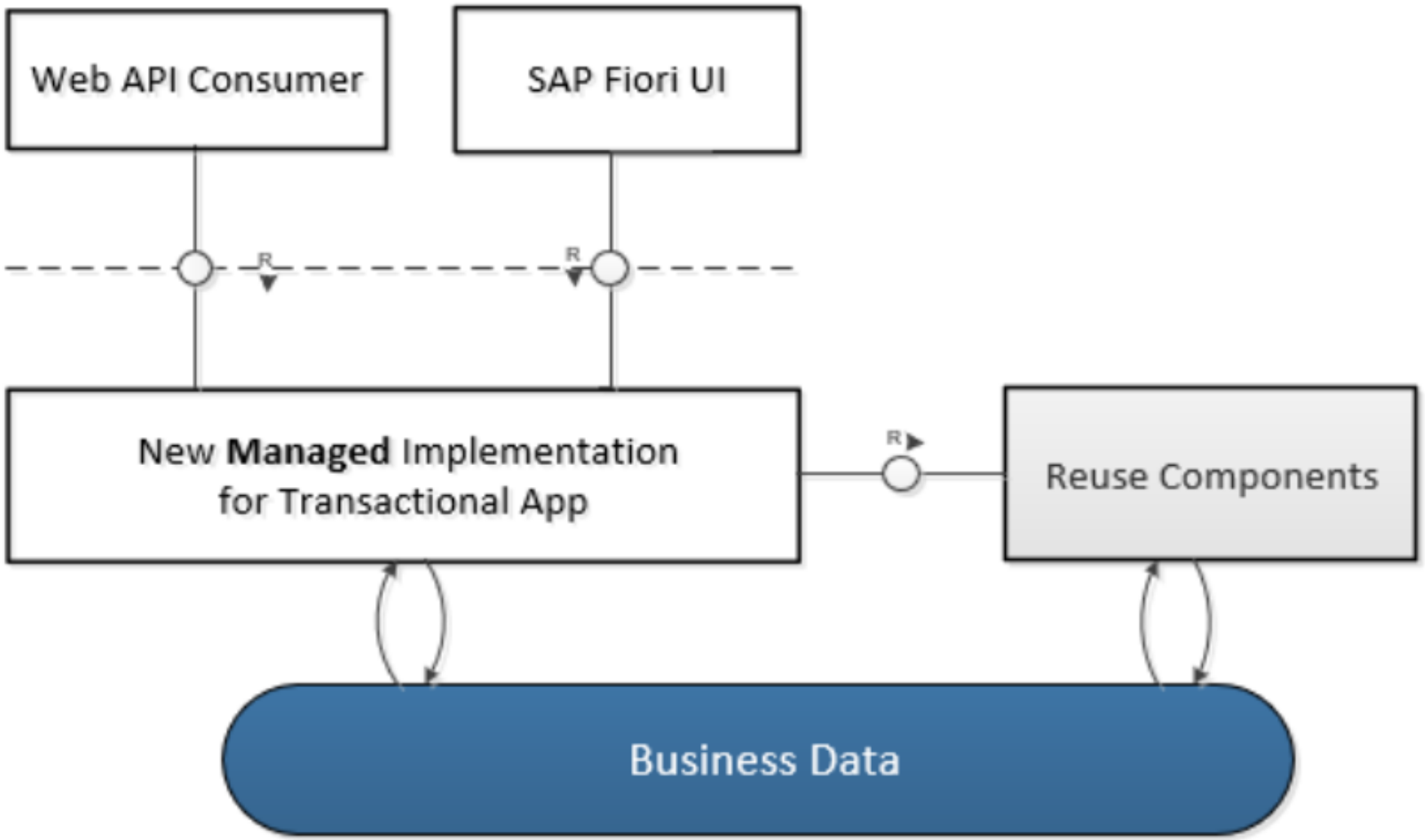
The Core Data Services (CDS) View



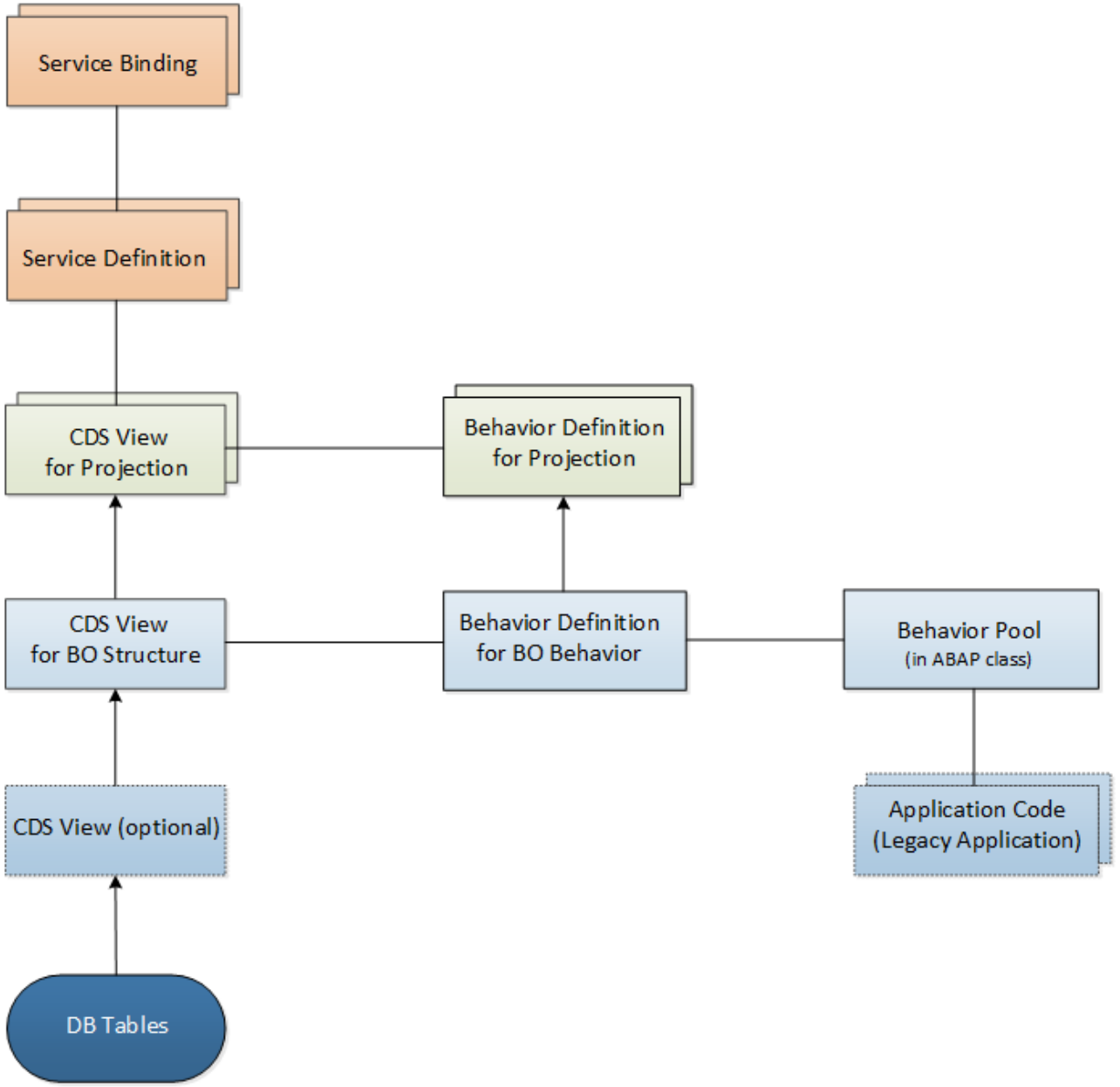
Managed Implementation

Architecture Overview -

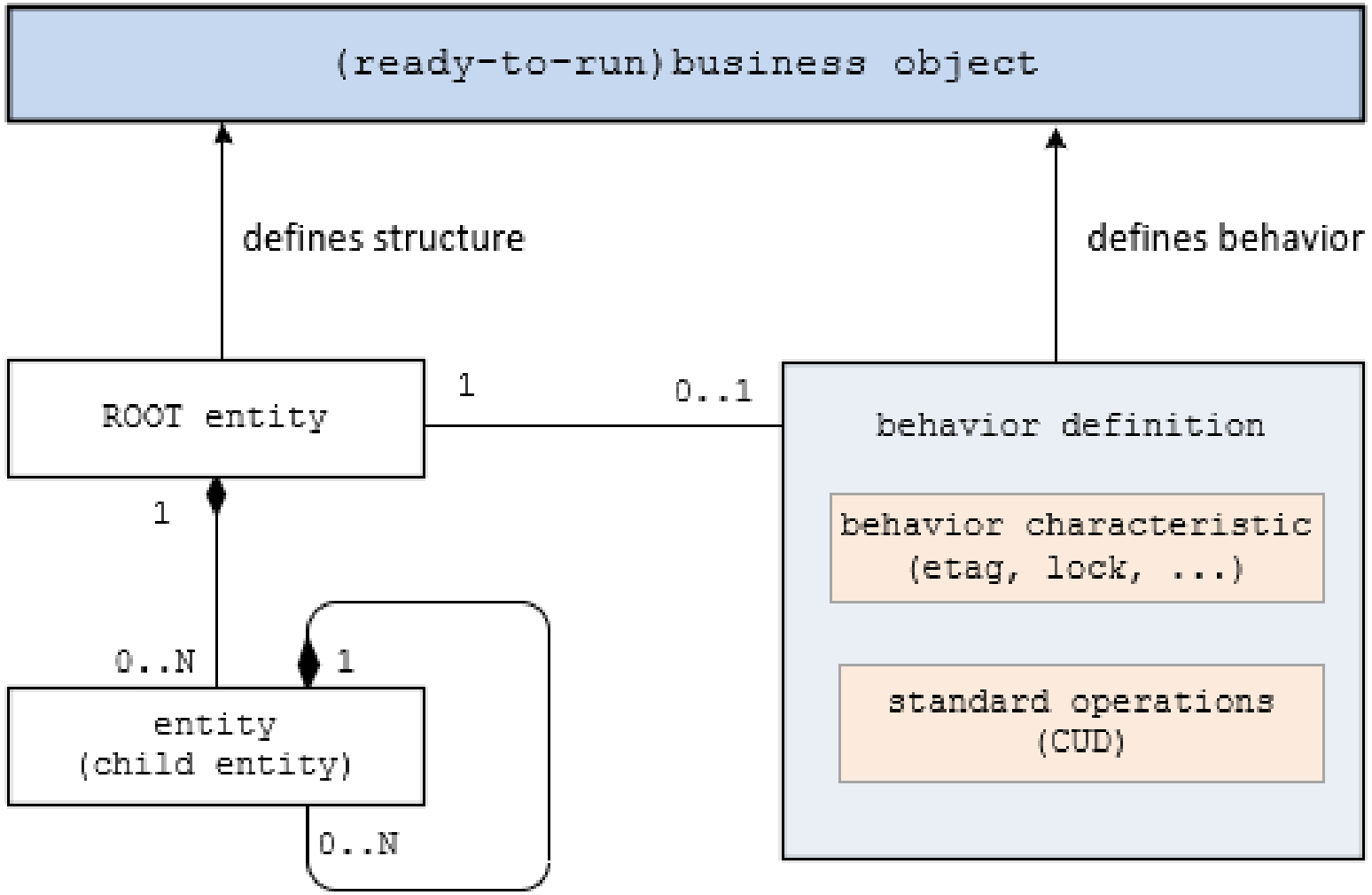
Managed implementations



Involved Development Objects



Data Model and Behavior a RAP Managed App



About the ABAP Flight Reference Scenario

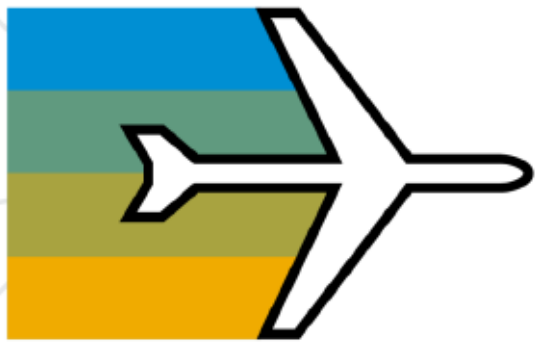
Simple to use and understand data model for demo purposes

Used to

Demonstrate how to use different RAP capabilities concretely

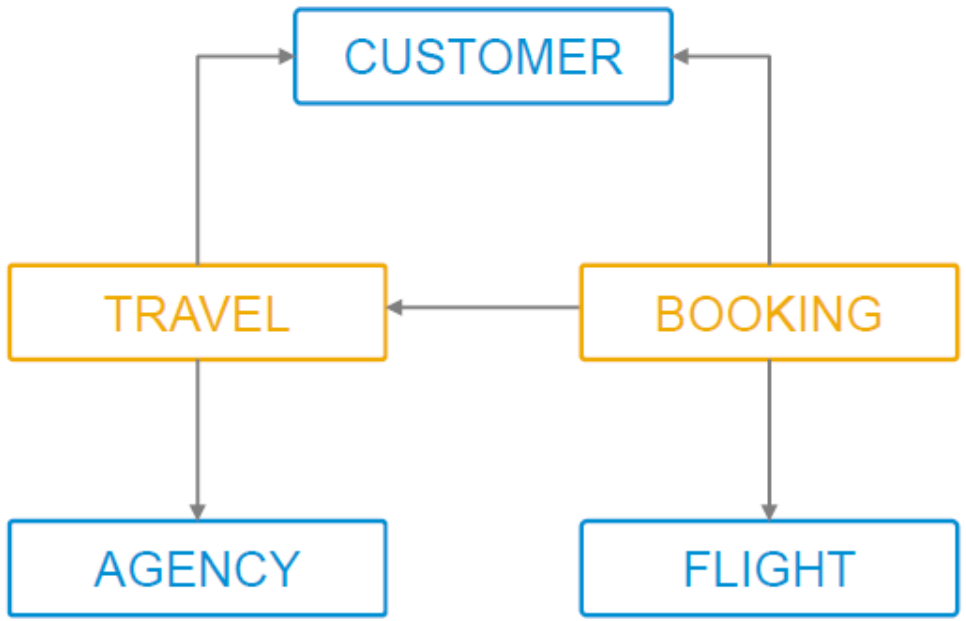
Simulate real-life standard scenarios and processes of a travel agency

Cloud and on-premise flavors available on GitHub for import into ABAP system



SFLIGHT RELOADED!

Simplified flight data model for this openSAP course

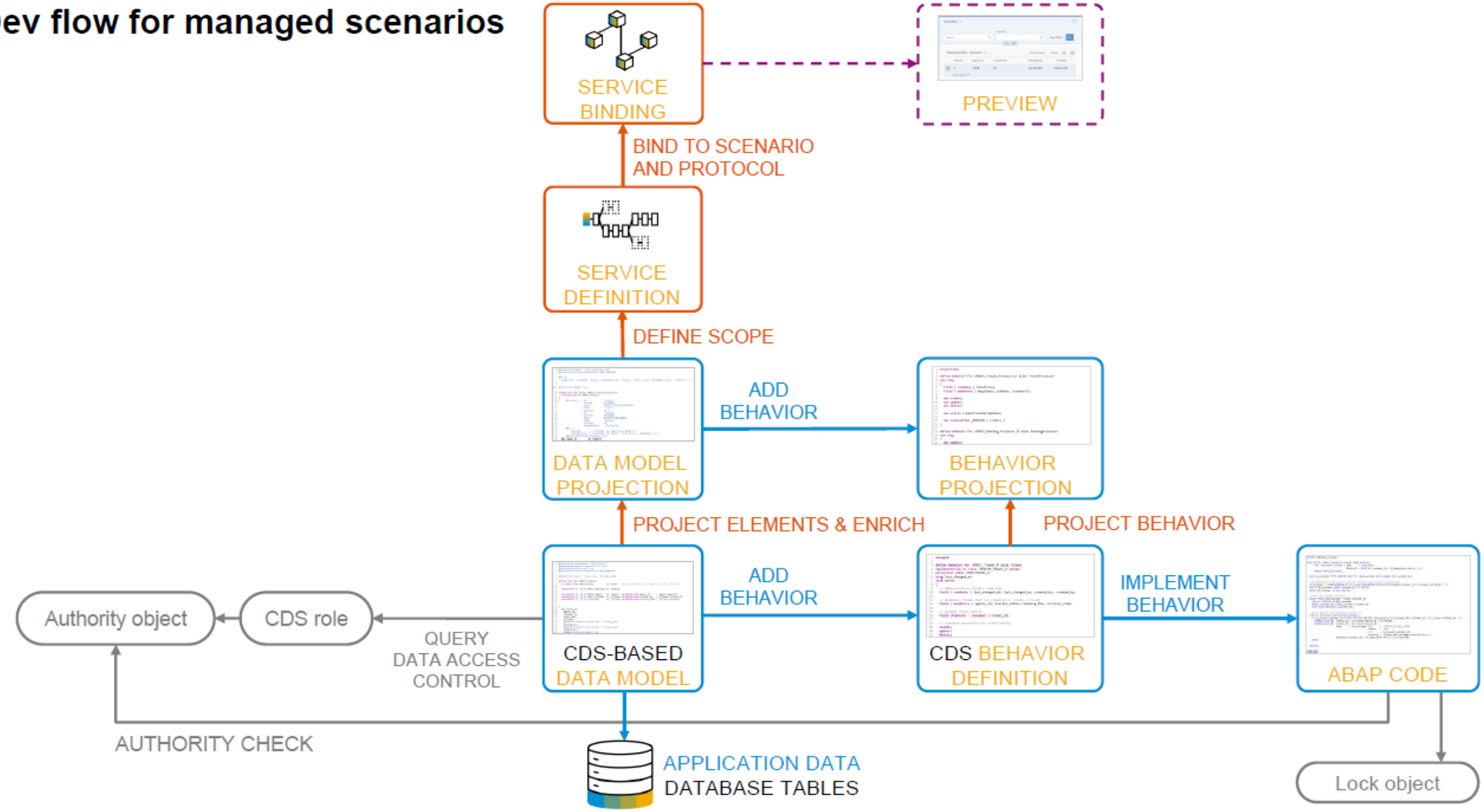


Main business entities in current scenario
Secondary business entities in current scenario

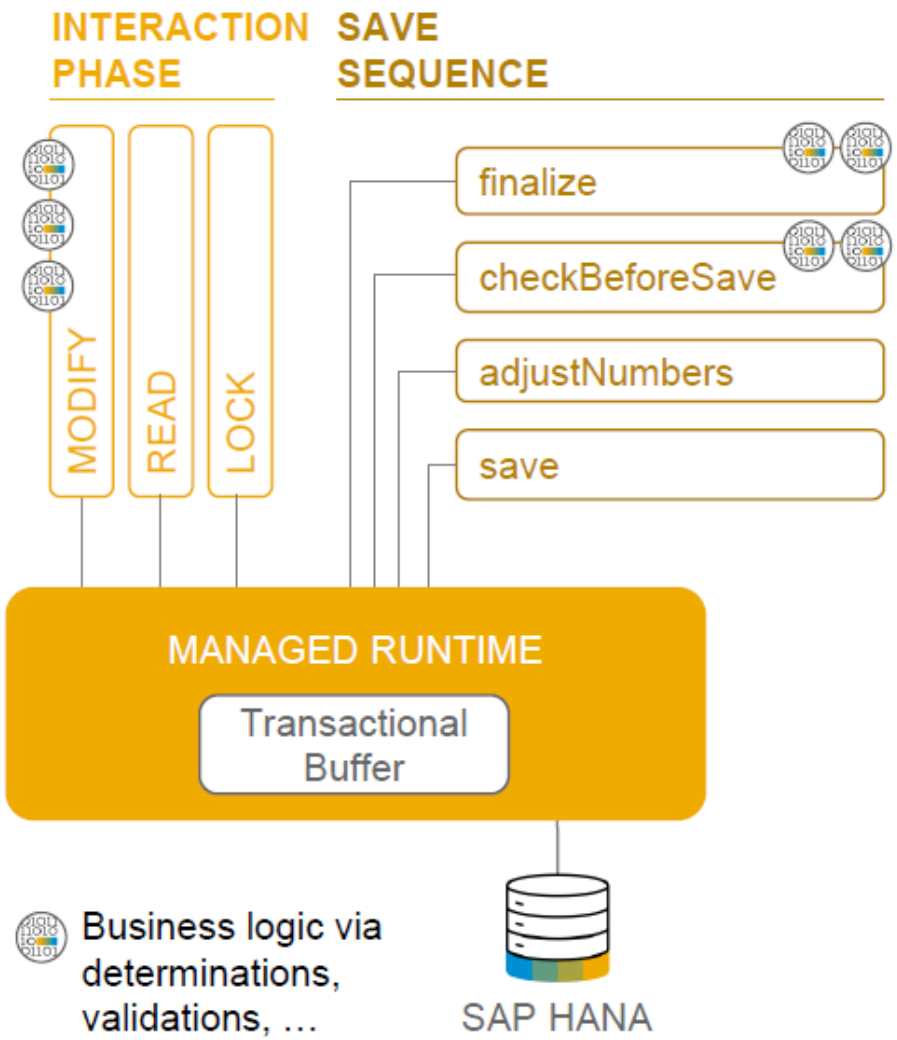
Travel	A Travel entity defines general travel data, such as the agency ID or customer ID, the status of the travel booking, and the price of travel. The travel data is stored in the database table `ZRAP_ATRAV_####`.
Booking	A Booking entity comprises general flight- and booking data as well as the related customer ID and travel ID. The booking data is stored in the database table `ZRAP_ABOOK_####`. The flight data model defines a 1:n cardinality between a Travel and the Booking entity.
Agency	An Agency entity defines travel agency data, such as the address and contact data. The corresponding data is stored in the database table `/DMO/AGENCY`. The flight data model defines a 1:n cardinality between Agency and Travel.
Flight	A Flight entity define general flight data for each connection. The flight data is stored in the database table `/DMO/FLIGHT`. The flight data model defines a 1:n cardinality between the Connection and the Flight entity.
Customer	A Customer entity provides a detailed description of a flight customer (passenger) such as the name, the address, and contact data. The corresponding data is stored in the database table `/DMO/CUSTOMER`. The flight data model defines a 1:n cardinality between Customer and Travel

Managed Scenario

Dev flow for managed scenarios



Managed business object runtime implementation



Application code

- Not yet available or fine granular reusable code available
- Technical implementation tasks taken over by BO infrastructure
- Standard CRUD operations available out-of-the-box
- Developer focuses on adding BO-specific business logic via dedicated code exits: determinations, validation and actions

Examples

- New applications on SAP Cloud Platform, ABAP environment and SAP S/4HANA

RAP For Fiori Elements

Providing Value Help

- Simple Value Help

  You want to provide a value help for an input or for a filter field on the UI.

  @Consumption.valueHelpDefinition: [{ entity: { name: 'entityRef' ,
  element: 'elementRef' } }]

```
    @Consumption.valueHelpDefinition: [{  entity: {   name: 'entityRef' ,
                                                      element: 'elementRef'  }     }]
```

```
define view /DMO/I_Booking as select from /dmo/booking
{…
       @Consumption.valueHelpDefinition: [{entity: { name: '/DMO/I_CUSTOMER_VH',
                                                     element: 'customer_id'  }}]
customer_id as CustomerID,
… }
```

RAP For Fiori Elements

Providing Value Help

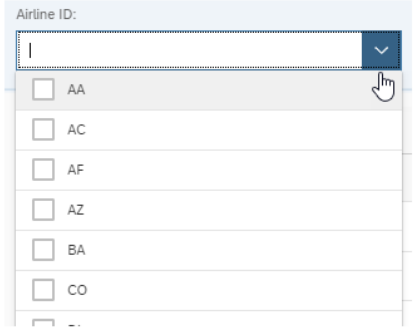
- Value Help as Dropdown List

  If you want to show the possible values for the input field as a dropdown list only instead of as a search mask with all other elements of the value help provider, annotate the value help provider with

  @ObjectModel : { resultSet.sizeCategory: #XS }

  on the view level.

```
<Property Name="CarrierID" sap:label="" sap:display-format="UpperCase" MaxLength="3" Type="Edm.String" sap:value-list="fixed-values"/>
```



RAP For Fiori Elements

Providing Value Help

- **Value Help Via Association**

If the value help provider is already associated with the source entity, you can use the association

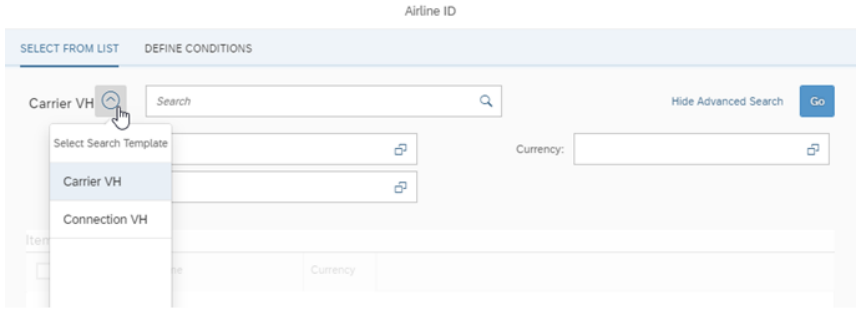
@Consumption.valueHelpDefinition.association: 'Assoc'

and reference the association instead of the entity. The binding condition is then already given in the on-condition of the association.

```
define view /DMO/I_Booking as select from /dmo/booking
association [0..1] to /DMO/I_Customer_VH as _Customer on $projection.CustomerID = _Customer.customer_id
{…
        @Consumption.valueHelpDefinition.association: '_Customer'
customer_id as CustomerID,
…
_Customer
}
```

RAP For Fiori Elements

Providing Value Help



- Multiple Value Helps on One Element

It is possible to provide more than one value help on one element. The end user selects which value help to use to find the correct value.

1. To implement two value helps on one element, proceed as described in Simple Value Help and add another entity as a value help provider

```
define view /DMO/I_Booking as select from /dmo/booking
{…
       @Consumption.valueHelpDefinition: [{ entity: {  name: '/DMO/I_Carrier_VH' ,
                                                       element: 'carrier_id' }}
                                           { entity: {  name: '/DMO/I_Connection_VH',
                                                       element: 'carrier_id'}} ]
carrier_id as CarrierID,
… }
```

2. Assign labels to the different value helps to differentiate them on the UI.

```
define view /DMO/I_Booking as select from /dmo/booking
{…
       @Consumption.valueHelpDefinition: [{ entity: {  name: '/DMO/I_Carrier_VH' ,
                                                       element: 'carrier_id' },
                                            label: 'Carrier VH' },
                                           { entity: {  name: '/DMO/I_Connection_VH',
                                                       element: 'carrier_id'},
                                            label: 'Connection VH'}]
carrier_id as CarrierID,
… }
```

3. Equip one value help with a qualifier.

```
define view /DMO/I_Booking as select from /dmo/booking
{…        @Consumption.valueHelpDefinition: [{ entity: {  name: '/DMO/I_Carrier_R' ,
                                                       element: 'carrier_id' },
                                            label: 'Carrier VH' },
                                           { entity: {  name: '/DMO/I_Connection_VH',
                                                       element: 'carrier_id'},
                                            label: 'Connection VH' ,
                                            qualifier: 'Secondary Value Help'}]
carrier_id as CarrierID,
… }
```

RAP For Fiori Elements

Providing Value Help

- Value Help with Additional Binding
  - You use an additional binding to define more than one binding condition between the source view and the value help provider. The value help is then automatically filtered by the additional binding. It proposes only entries that match the additional binding. This additional binding can either be another element or a parameter. These need to be present in the source view and in the value help provider view. When an entry is selected in the value help provider, the values of both binding elements are transferred to the input fields of the source CDS view.

1. Create a data definition for a CDS view that serves as a value help provider. It must contain a field with the available values for the input field in the source view. In addition, it must contain the field for which the additional binding is established.

```
define view /DMO/I_Connection_VH as select from /dmo/connection
{
key carrier_id,
key connection_id,
airport_from_id,
airport_to_id,
departure_time,
arrival_time,
distance,
distance_unit
}
```

1. In your CDS source view, add the following annotation to the element for which you want to provide the value help on the UI.

```
            @Consumption.valueHelpDefinition: [{  entity: {name: 'entityRef', element: 'elementRef},
                                  additionalBinding: [{  element: 'elementRef', localElement: 'elementRef' }]            }]
```

```
define view /DMO/I_Booking_VH as select from /dmo/booking
{…
    @Consumption.valueHelpDefinition: [{ entity:
                {name: '/DMO/I_Connection_VH' , element: 'connection_id' },
                additionalBinding: [{ localElement: 'CarrierID', element: 'carrier_id' }]
                }]
connection_id as ConnectionID,
… }
```

Understanding Entity Manipulation Language (EML) EML at a glance

ENTITY MANIPULATION LANGUAGE

- EXTENSION OF THE ABAP LANGUAGE
  with an SQL-like syntax
- CONTROL THE TRANSACTIONAL BO BEHAVIOR IN RAP CONTEXT
- DIRECT API-BASED ACCESS TO RAP BOs
- STANDARD EML API
  for type-safe read and modifying access to RAP BOs
- GENERIC EML API
  for generic integration of RAP BOs into other frameworks
- DATA CONSISTENCY ENSURED BY DATABASE LUW
  COMMIT operation required to persist changes

READ operation

READ ACCESS TO RAP BOs

EXAMPLE

```
READ ENTITIES OF ZI_RAP_Travel_1234
  ENTITY travel
    ALL FIELDS WITH VALUE #( ( TravelUUID = '<someUUID>' ) )

  RESULT DATA(travels)
  FAILED DATA(failed)
  REPORTED DATA(reported).
```

Failure handling (failed & reported)

MODIFY – CREATE operation

EXAMPLE

```
MODIFY ENTITIES OF ZI_RAP_Travel_1234
  ENTITY travel
    CREATE
      SET FIELDS WITH VALUE
        #( ( %cid        = 'MyContentID_1'
             AgencyID    = '70012'
             CustomerID  = '14'
             BeginDate   = cl_abap_context_info=>get_system_date( )
             EndDate     = cl_abap_context_info=>get_system_date( ) + 10
             Description = 'I like RAP@openSAP' ) )

 MAPPED DATA(mapped)
 FAILED DATA(failed)
 REPORTED DATA(reported).

COMMIT ENTITIES
  RESPONSE OF ZI_RAP_Travel_1234
  FAILED     DATA(failed_commit)
  REPORTED   DATA(reported_commit).
```

MODIFY ACCESS TO RAP BOs

Failure handling (failed & reported)

COMMIT statement for proper LUW handling

MODIFY – DELETE operation

DELETE ACCESS TO RAP BOs

EXAMPLE

```
MODIFY ENTITIES OF ZI_RAP_Travel_1234
  ENTITY travel
    DELETE FROM
      VALUE
        #( ( TravelUUID  = '<someUUID>' ) )

 FAILED DATA(failed)
 REPORTED DATA(reported).

COMMIT ENTITIES
  RESPONSE OF ZI_RAP_Travel_1234
  FAILED     DATA(failed_commit)
  REPORTED   DATA(reported_commit).
```

Failure handling (failed & reported)

COMMIT statement for proper LUW handling

MODIFY – EXECUTE ACTION operation

EXECUTION OF RAP BO-SPECIFIC ACTIONS

EXAMPLE

```
MODIFY ENTITIES OF ZI_RAP_Travel_1234
  ENTITY travel
    EXECUTE acceptTravel
      FROM VALUE
        #( ( TravelUUID  = '<someUUID>' ) )

 RESULT DATA(result)
 MAPPED DATA(mapped)
 FAILED DATA(failed)
 REPORTED DATA(reported).

COMMIT ENTITIES
  RESPONSE OF ZI_RAP_Travel_1234
  FAILED     DATA(failed_commit)
  REPORTED   DATA(reported_commit).
```

Failure handling (failed & reported)

COMMIT statement for proper LUW handling

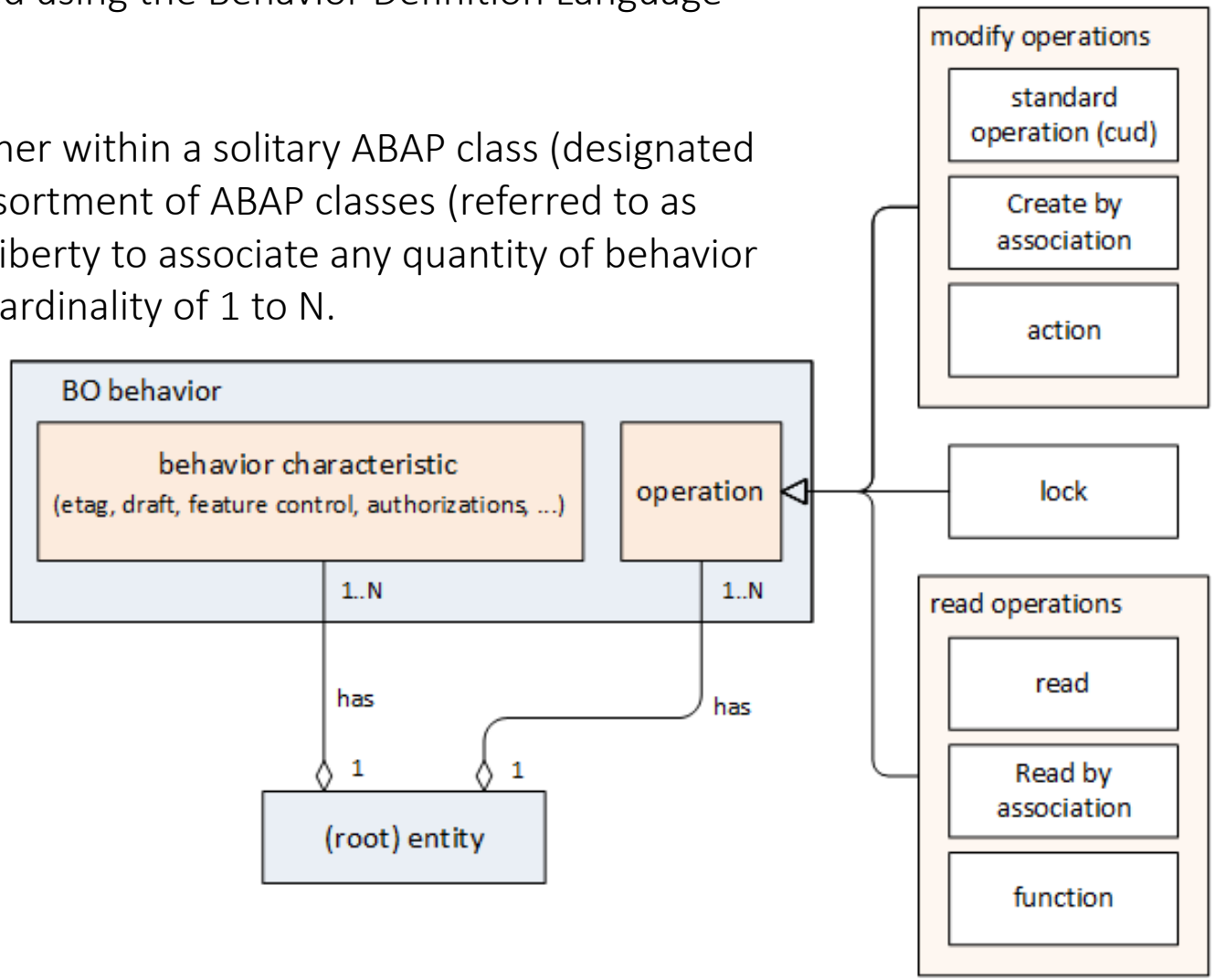
Behaviour Definition & Implementation- Managed

In the context of the ABAP RESTful application programming model, a behavior definition (abbreviated as behavior definition) stands as an ABAP Repository entity that elucidates the conduct of a business object. This description is crafted using the Behavior Definition Language (BDL).

The realization of a behavior definition is achieved either within a solitary ABAP class (designated as a behavior pool) or can be partitioned across an assortment of ABAP classes (referred to as behavior pools). The application developer holds the liberty to associate any quantity of behavior pools with a singular behavior definition, affording a cardinality of 1 to N.

Within this framework, a behavior defines the actions and attributes pertinent to an individual business object within the ABAP RESTful programming model.

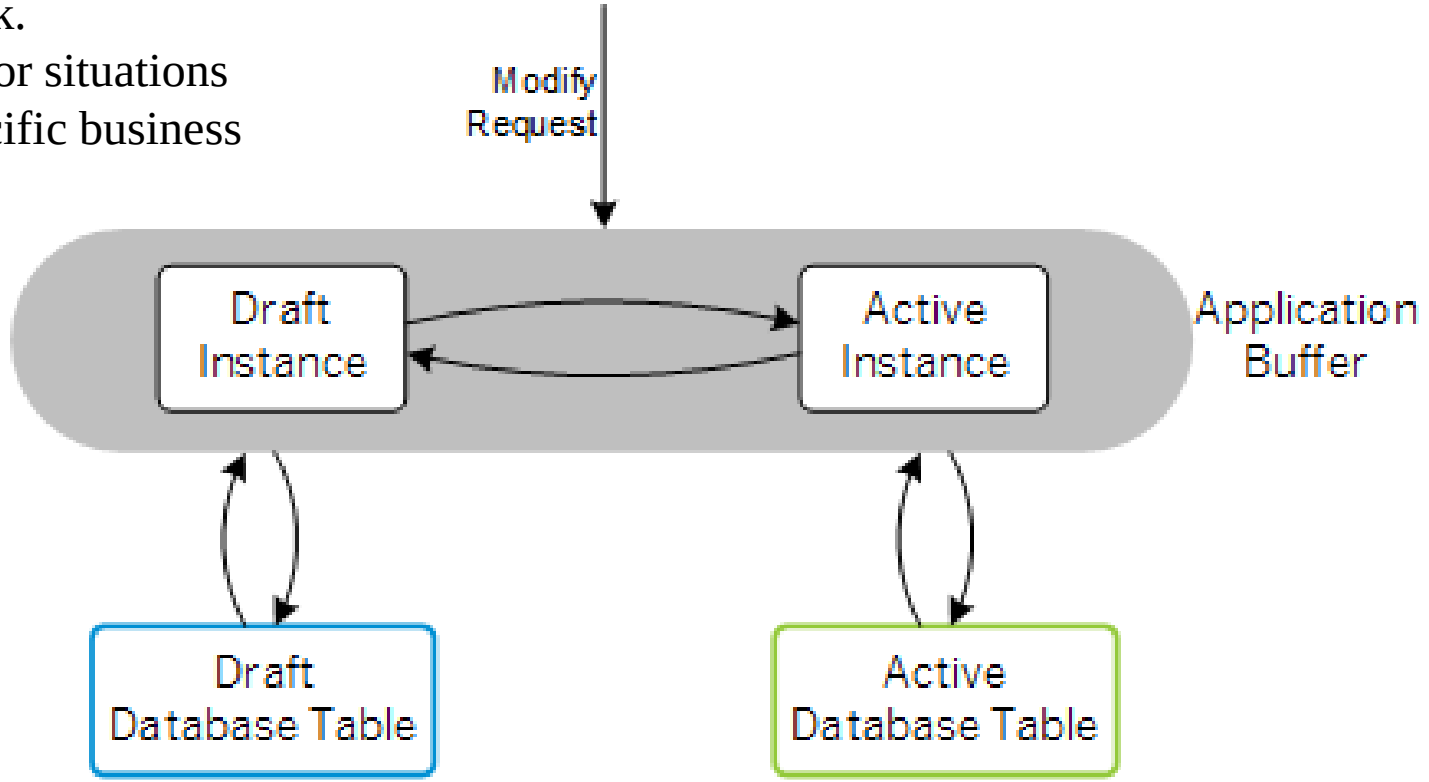
It encompasses both a behavioral characteristic and a series of operations for every entity constituting the business object's composition tree.



Draft handling

Draft-enabled applications enable users to save modified data in the backend and resume their work at a later time or on a different device, even if the application unexpectedly terminates.

Draft is an option that you can use for application development with both, the managed and unmanaged implementation type and with mixed scenarios, for example, managed with unmanaged save. In all scenarios, the draft is managed. That means, it is handled by the RAP runtime framework. Furthermore, RAP provides implementation exits for situations where you require draft capabilities tailored to specific business services that influence how drafts are managed.



Numbering (Early)
Numbering is about setting values for primary key fields of entity instances during runtime. There are two types of numbering, Early and Late numbering. Managed scenarios support only Early Numbering. The term 'early numbering' in numbering types signifies the assignment of a value to the primary key field at an early stage. This key value can originate from either the consumer (externally) or the framework itself (internally).

External Numbering
We refer to external numbering if the consumer hands over the primary key values for the CREATE operation, just like any other values for non-key fields. The runtime framework, whether it's managed or unmanaged, assumes control of the value and manages it until it is ultimately written to the database.

Internal Numbering Managed:
When employing managed numbering, the RAP managed runtime automatically generates a UUID during the CREATE request.

Unmanaged:
The concept of unmanaged early numbering is established at the entity level of the business object with early numbering. The implementation is applied to all keys of an entity, so that all key fields must be mapped in the implementation. The signature of this handler method is defined by the keyword FOR NUMBERING, followed by the input parameters entities, the implicit changing parameters reported, failed, and mapped.

Authorization Definition

Authorization Master

An entity is defined as authorization master (authorization master ()) if the operations of this entity have their own authorization implementation.

Authorization Dependent

An entity is considered authorization-dependent (designated as 'authorization dependent by _Assoc') when the authorization controls established for the authorization master entity are intended to be enforced for the operations involving this entity as well.

Global Authorization

By defining global authorization (authorization master (global)), you implement authority control for the following operations of the entity: Create, Create-by-association, Update, Delete, Static Actions and Instance Actions.

Instance Authorization

By defining instance authorization (authorization instance ()), the following operations of the entity can be checked against unauthorized access: Create-by-association, Update, Delete and Instance Actions.

Implementing Instance Authorization

Definition

```
managed;
with draft;

define behavior for /DMO/R_Travel_D alias Travel
implementation in class /DMO/BP_TRAVEL_D unique
persistent table /DMO/A_TRAVEL_D
...
authorization master ( instance )
{...
```

Implementation

```
CLASS lhc_travel DEFINITION INHERITING FROM cl_abap_behavior_handler.

  PRIVATE SECTION.

...
    METHODS get_instance_authorizations FOR INSTANCE AUTHORIZATION
      IMPORTING keys REQUEST requested_authorizations FOR travel RESULT result.

ENDCLASS.
```

```
READ ENTITIES OF /DMO/R_Travel_D IN LOCAL MODE
  ENTITY Travel
    FIELDS ( AgencyID )
    WITH CORRESPONDING #( keys )
RESULT DATA(travels)
```

```
update_requested = COND #( WHEN requested_authorizations-%update                = if_abap_behv=>mk-on OR
                                requested_authorizations-%action-Edit            = if_abap_behv=>mk-on
                           THEN abap_true ELSE abap_false ).
```

```
  IF update_requested = abap_true.
    update_granted = is_update_granted( <travel_agency_country_code>-country_code  ).
    IF update_granted = abap_false.
      APPEND VALUE #( %tky = travel-%tky
                      %msg = NEW /DMO/CM_FLIGHT_MESSAGES(
                                             textid    = /DMO/CM_FLIGHT_MESSAGES=>not_authorized_for_agencyid
                                             agency_id = travel-AgencyID
                                             severity  = if_abap_behv_message=>severity-error )
                      %element-AgencyID = if_abap_behv=>mk-on
                    ) TO reported-travel.
    ENDIF.
```

Implementing Instance Authorization

Definition

```
managed;
with draft;

define behavior for /DMO/R_Travel_D alias Travel
implementation in class /DMO/BP_TRAVEL_D unique
persistent table /DMO/A_TRAVEL_D
...
authorization master ( instance )
{...
```

Implementation

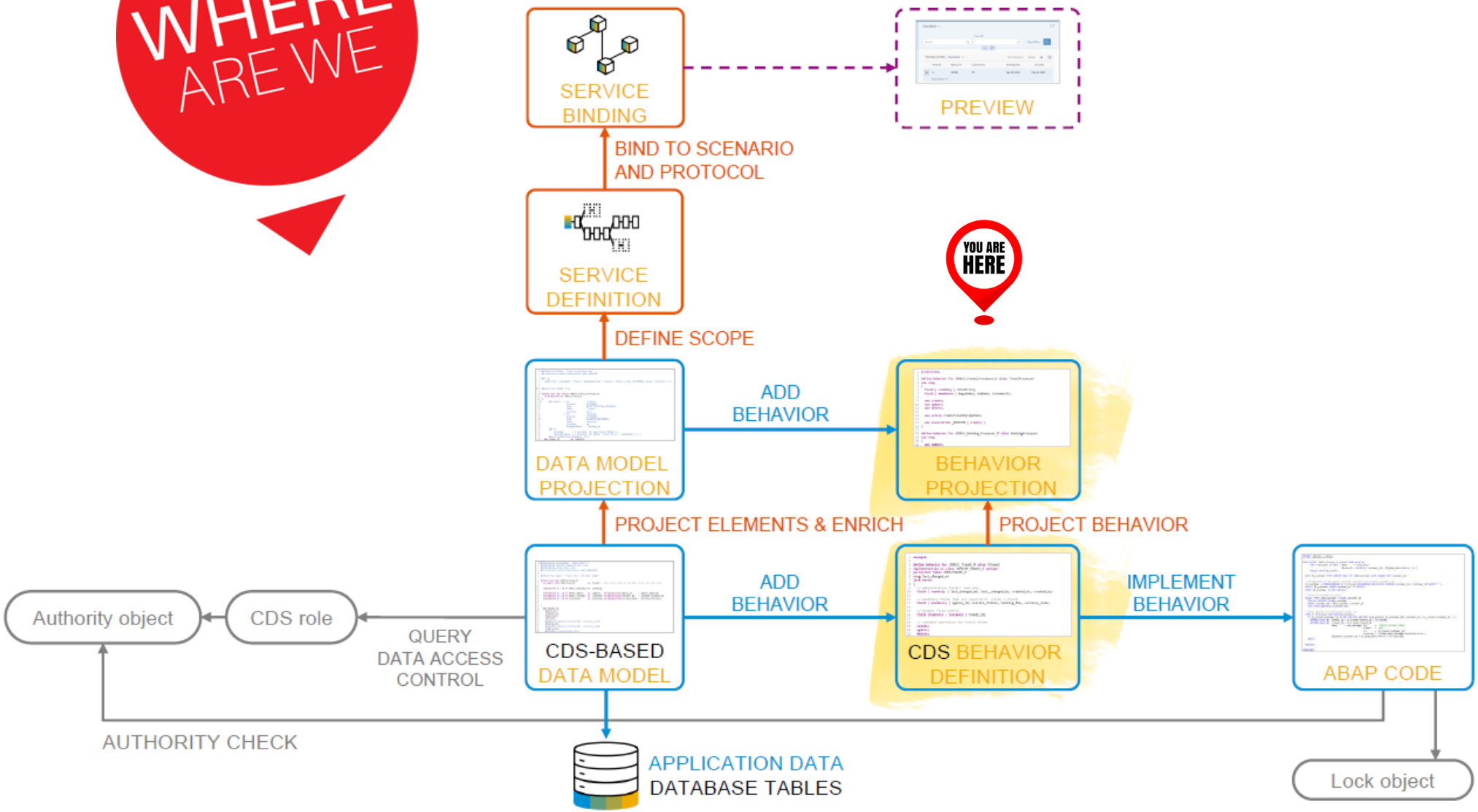
```
CLASS lhc_travel DEFINITION INHERITING FROM cl_abap_behavior_handler.

  PRIVATE SECTION.

...
    METHODS get_instance_authorizations FOR INSTANCE AUTHORIZATION
      IMPORTING keys REQUEST requested_authorizations FOR travel RESULT result.

ENDCLASS.
```

```
READ ENTITIES OF /DMO/R_Travel_D IN LOCAL MODE
  ENTITY Travel
    FIELDS ( AgencyID )
    WITH CORRESPONDING #( keys )
RESULT DATA(travels)
```

```
update_requested = COND #( WHEN requested_authorizations-%update                = if_abap_behv=>mk-on OR
                                requested_authorizations-%action-Edit            = if_abap_behv=>mk-on
                           THEN abap_true ELSE abap_false ).
```

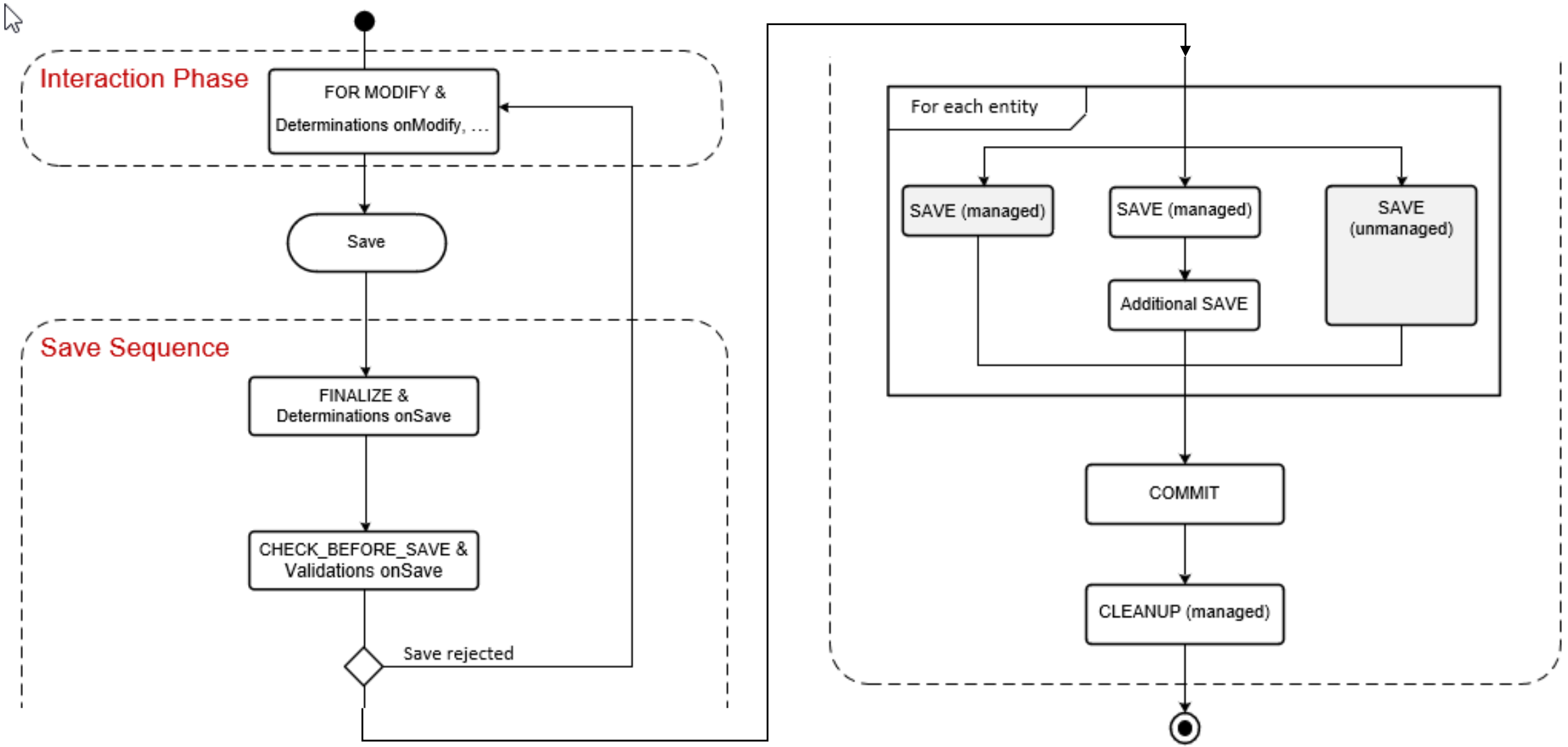
```
  IF update_requested = abap_true.
    update_granted = is_update_granted( <travel_agency_country_code>-country_code  ).
    IF update_granted = abap_false.
      APPEND VALUE #( %tky = travel-%tky
                      %msg = NEW /DMO/CM_FLIGHT_MESSAGES(
                                             textid    = /DMO/CM_FLIGHT_MESSAGES=>not_authorized_for_agencyid
                                             agency_id = travel-AgencyID
                                             severity  = if_abap_behv_message=>severity-error )
                      %element-AgencyID = if_abap_behv=>mk-on
                    ) TO reported-travel.
    ENDIF.
```

Development Flow :



Save Sequence (Managed)

The save sequence is an integral component of the business object runtime and is invoked after at least one successful modification has been made during the interaction phase.



Service Binding

The business service binding, often referred to as service binding, is an ABAP Repository entity employed to associate a service definition with a client-server communication protocol, such as OData.
As illustrated in the diagram below, a service binding directly depends on a service definition that originates from the underlying CDS-based data model. Multiple service bindings can be generated from an individual service definition. This division between the service definition and the service binding enables a service to effortlessly integrate various service protocols without requiring any re-implementation. Services created in this fashion maintain a distinct separation between the service protocol and the underlying business logic.

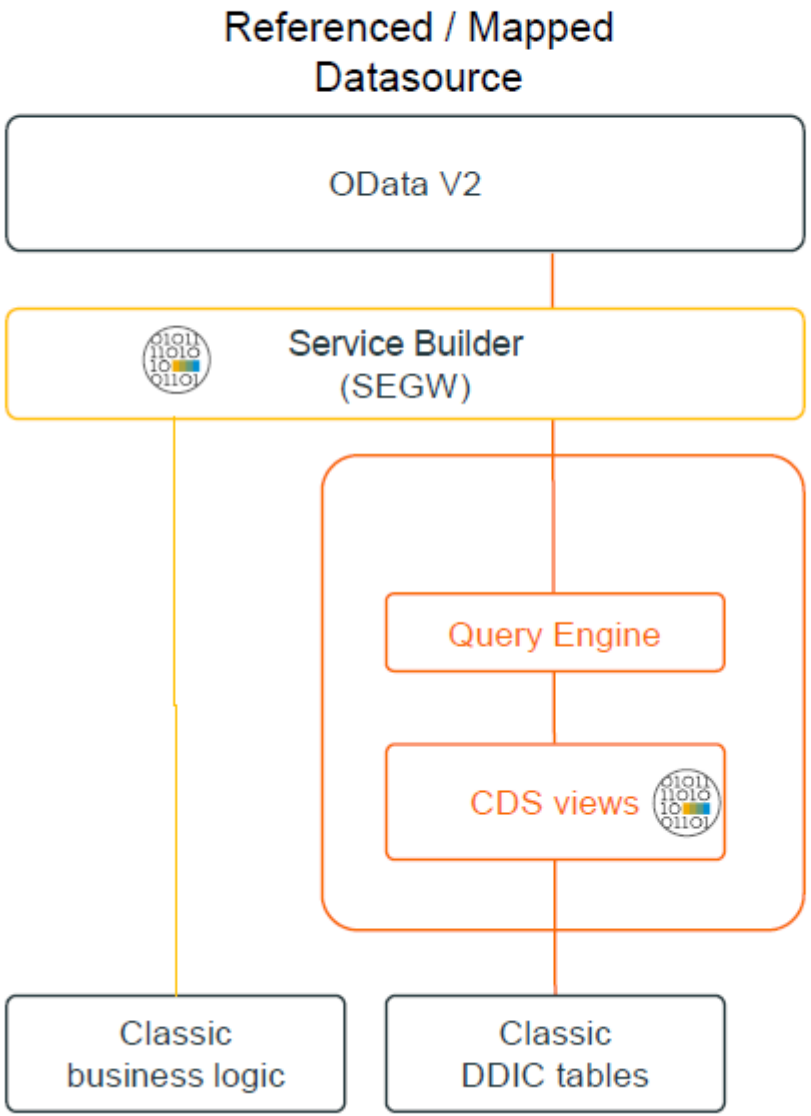
Binding Type

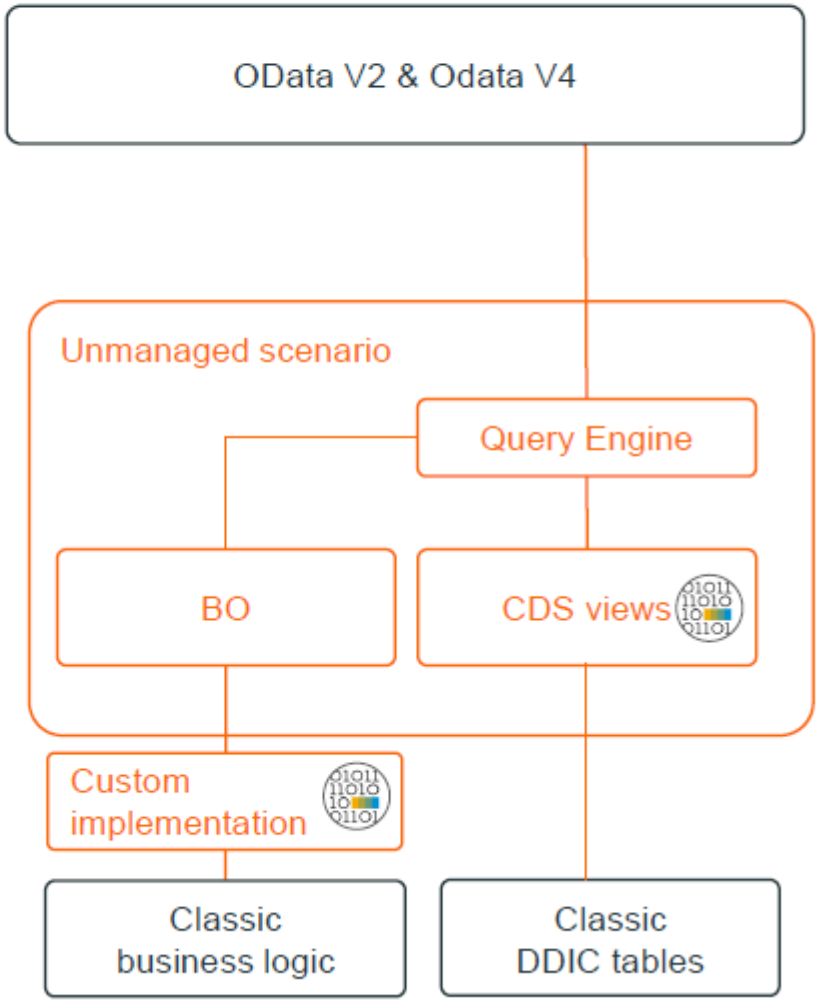
The binding type defines the service category and the particular protocol that the service binding implements.
OData in versions 2 and 4

OData V4 services have a wider range than OData V2 services. Use OData V4 wherever possible for transactional services.
OData for UIs or Web APIs

OData for UIs is designed for access to business services using UI technologies like SAPUI5. The data provided by a business service contains control elements for user interfaces.
OData for Web APIs is limited to the data-only content of the business services and does not contain any control elements for user interfaces.

Referenced / Mapped Datasource vs. RAP

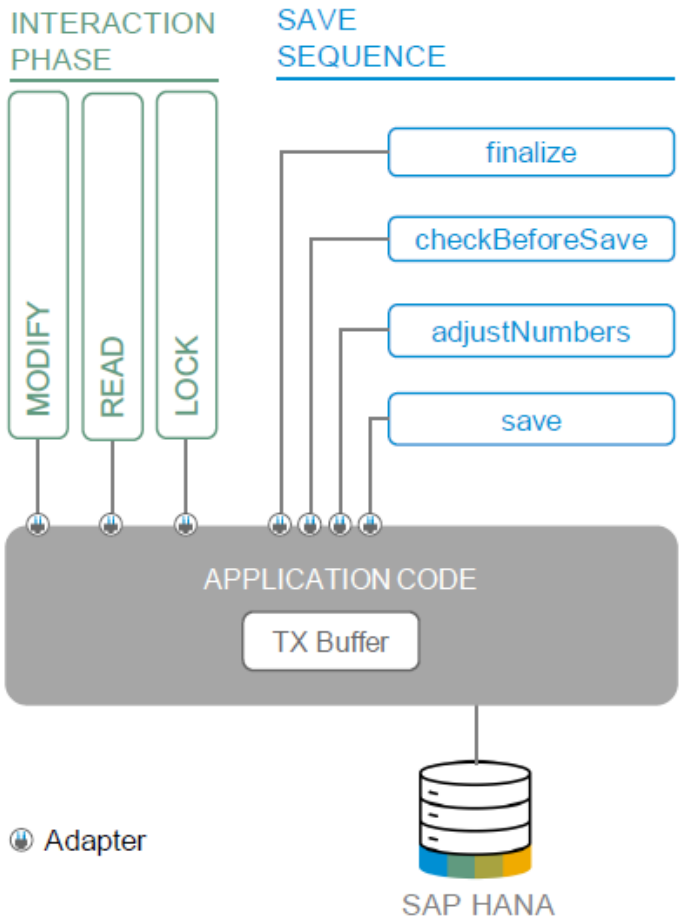




Un-Managed

Business Objects - Unmanaged

Unmanaged implementations



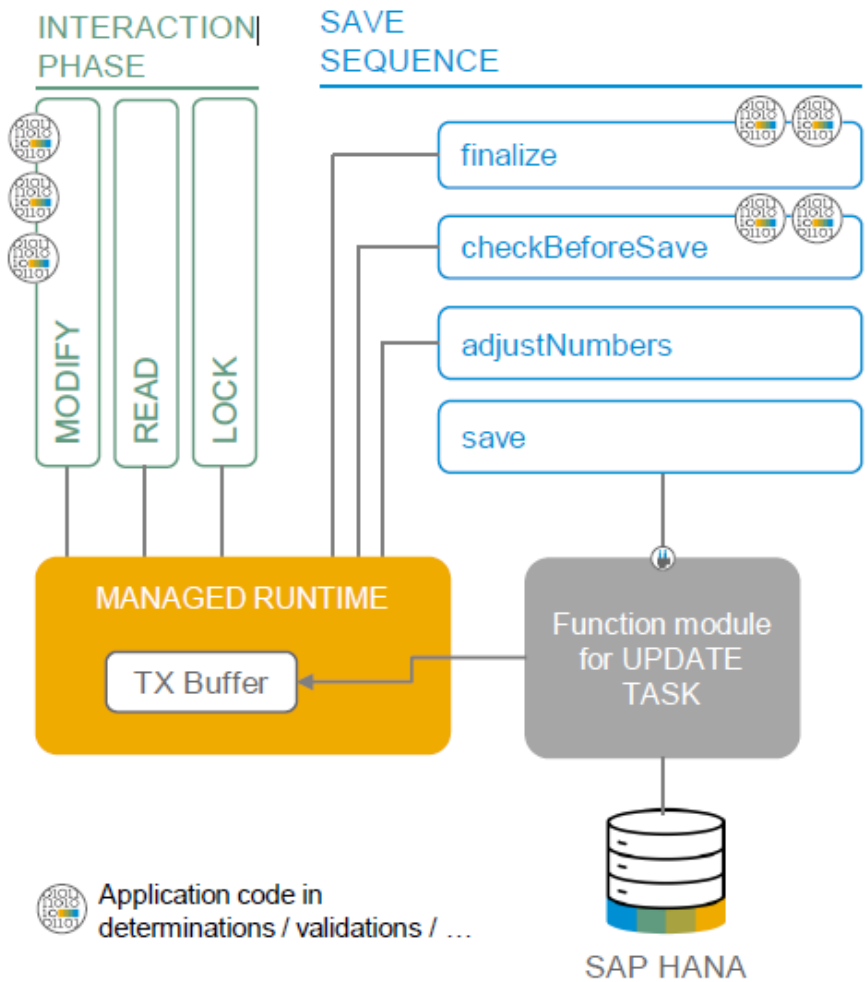
Application coding

- already available
- for interaction phase, transactional buffer and save sequence
- decoupled from UI technology

Examples

- Sales Order, Purchase Order

Business Objects – Save Unmanaged



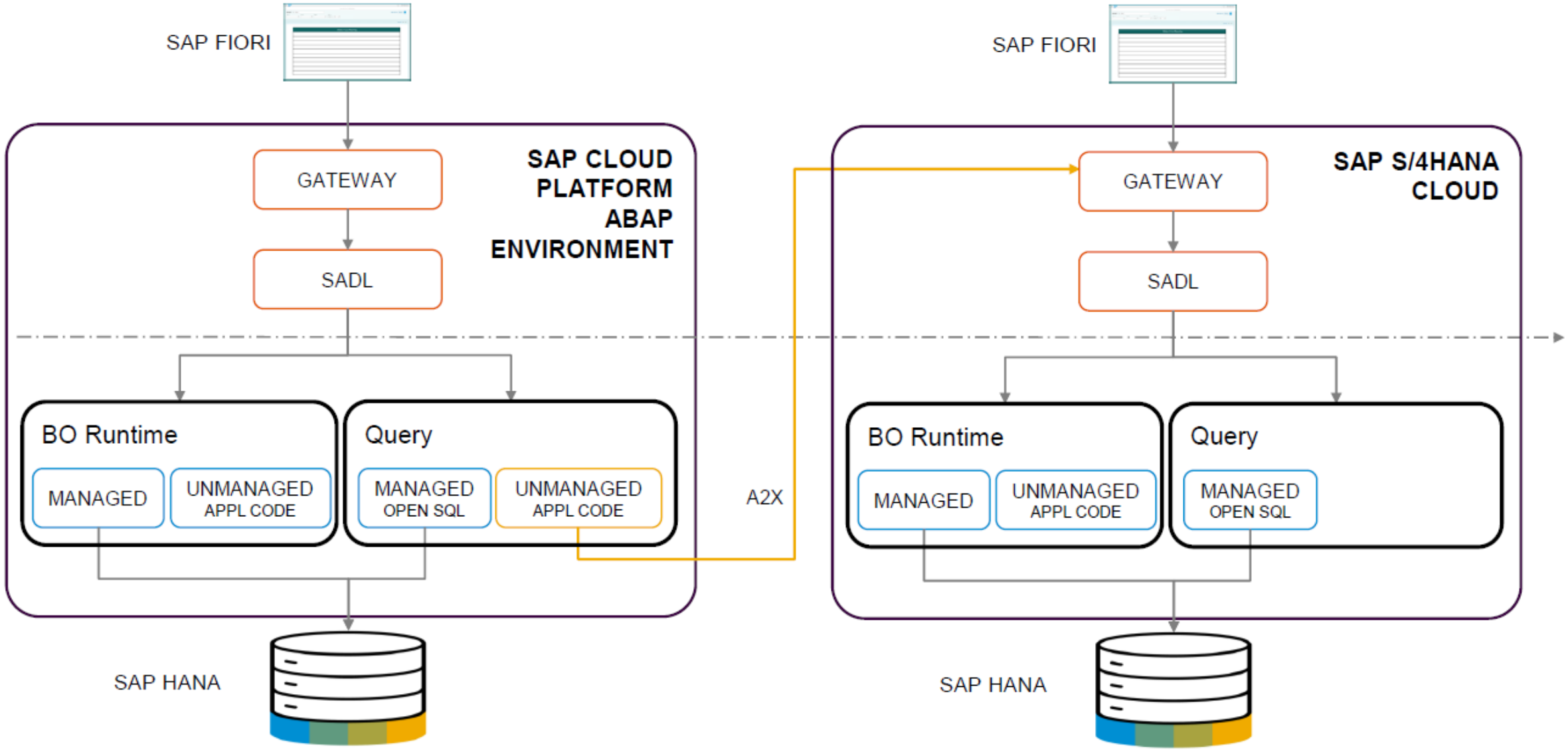
Application coding

- "update-task function module" available
- coding for interaction phase not available
  (e.g. highly coupled in older UI technology: DYNP - PBO / PAI)
- technical implementation aspects to be taken over by BO infrastructure

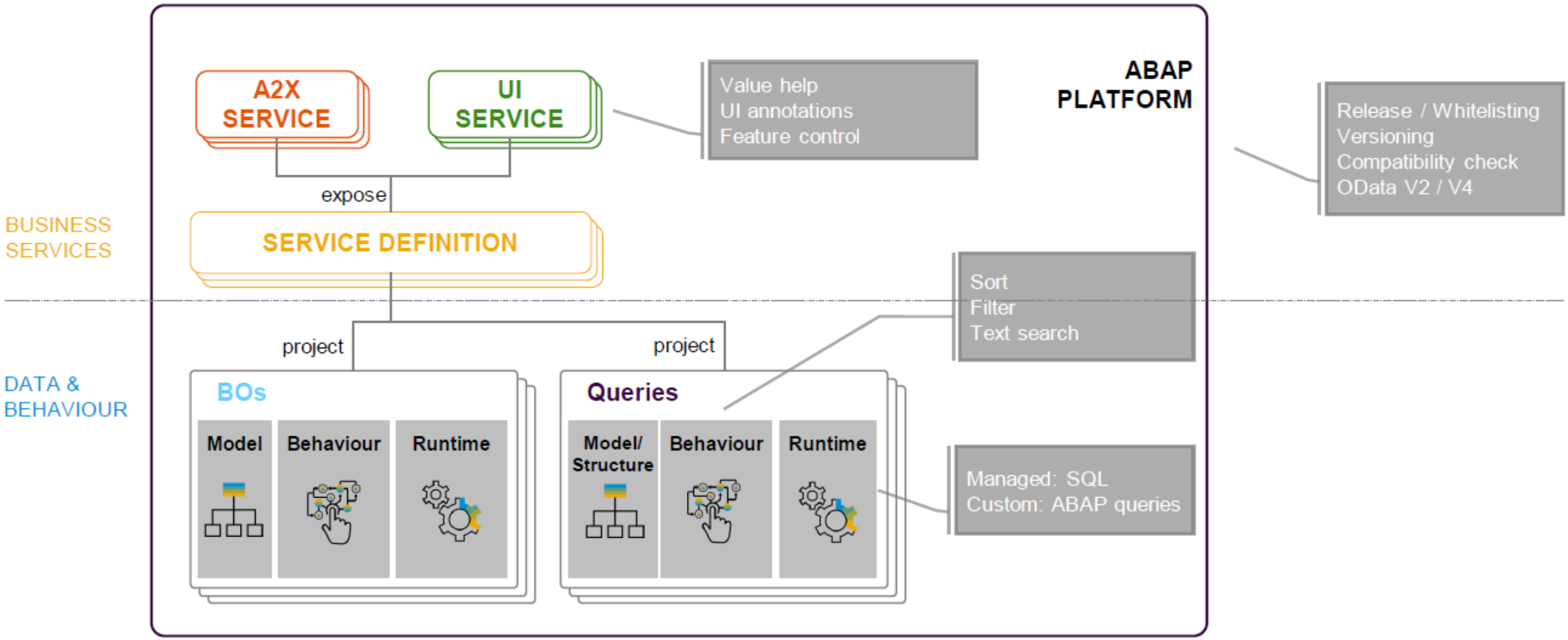
Examples

- Business Partner, Product

Business Services Big Picture– Runtime stack



Business Services Big Picture– Design Time stack



Runtime Frameworks

The runtime frameworks **SAP Gateway** and the **RAP Runtime Engine** are the frameworks that manage the generic runtime for OData services built with the ABAP RESTful Programming Model. As a developer you do not have to know the concrete inner functioning of these frameworks, as many development tasks are automatically given. However, the following sections provide a high-level overview.

SAP Gateway

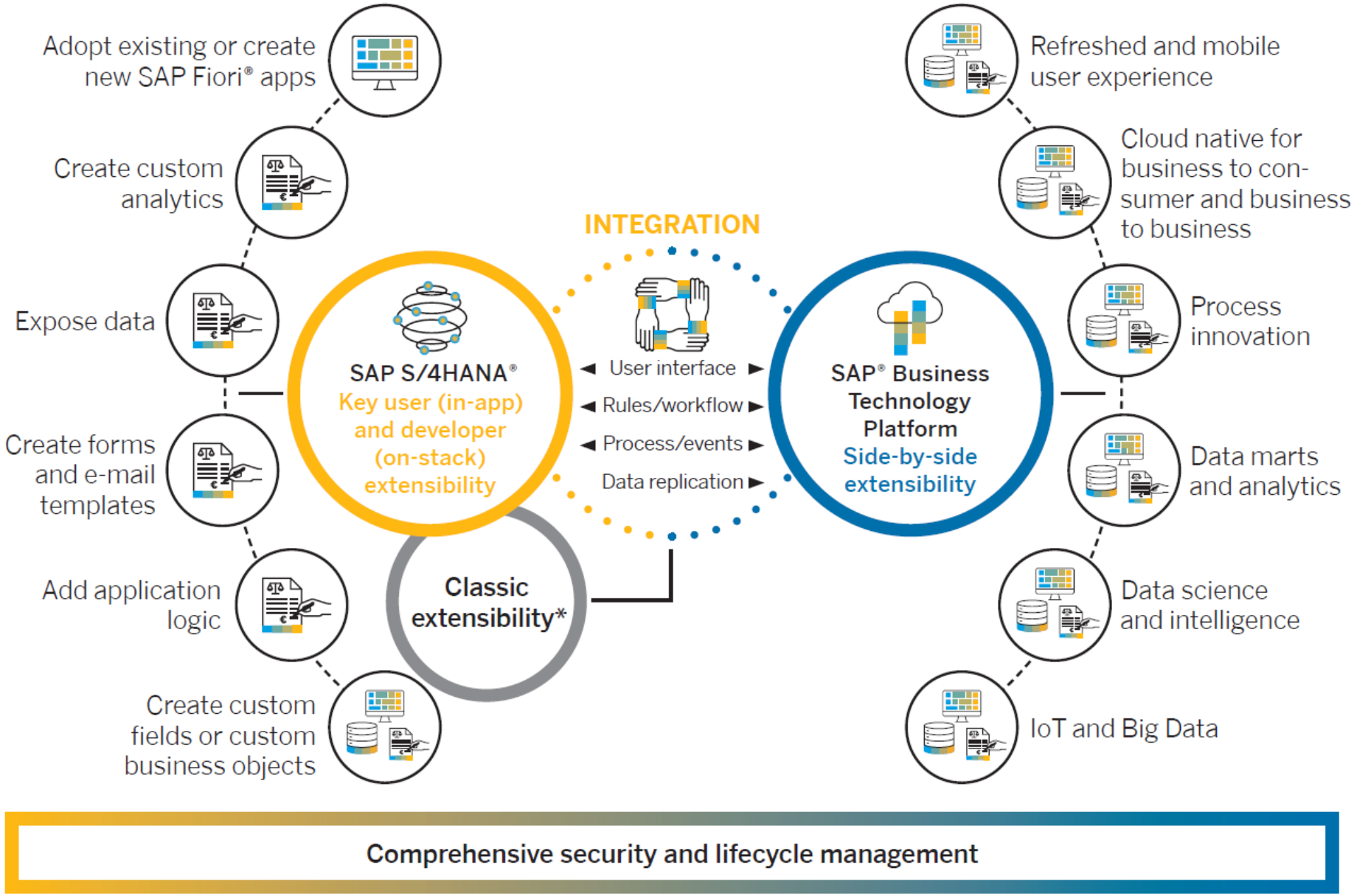
- SAP Gateway provides an open, REST-based interface that offers simple access to SAP systems via the Open Data Protocol (OData).
- As the name suggests, the gateway layer is the main entry point to the ABAP world. All services that are created with the ABAP RESTful Programming Model provide an OData interface to access the service.
- However, the underlying data models and frameworks are based on ABAP code. SAP Gateway converts these OData requests into ABAP objects to be consumed by the ABAP runtime.

RAP Runtime Engine

- The RAP runtime engine dispatches the requests for the business object (BO) or the query. It receives the ABAP consumable OData requests from the Gateway layer, forwards it to the relevant part of the business logic and interprets the matching ABAP calls for it.
- For transactional requests, the RAP runtime engine delegates the requests to the BO and calls the respective method of the BO implementation.
- For query requests, the framework executes the query. Depending on the implementation type, the BO or the query runtime is implemented by a framework or by the application developer.
- If locks are implemented, the RAP runtime engine executes first instance-independent checks and sets locks. For the eTag handling, the framework calls the necessary methods before the actual request is executed.

Note
The RAP runtime engine is also known under the name SADL (Service Adaptation Description Language). Apart from the runtime orchestration, the SADL framework is also responsible for essential parts in the query and BO runtime.

Extensibility Framework of SAP S/4HANA



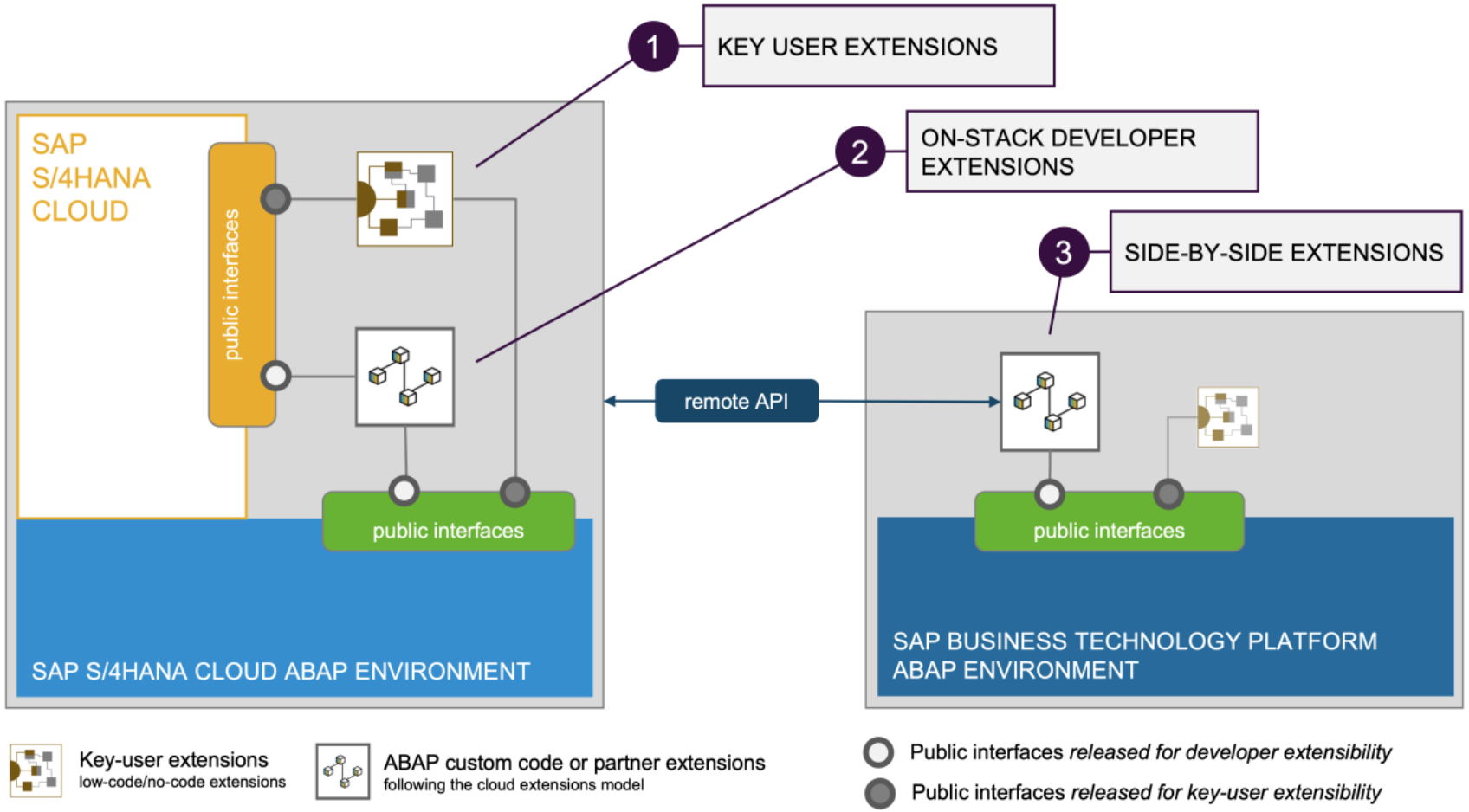
SAP S/4HANA Cloud covers these dimensions with the following extensibility options:

SAP S/4HANA CLOUD EXTENSIBILITY OPTIONS		
PERSONAS	**Tightly coupled extension on SAP S/4HANA (on-stack extensions)**	**Loosely-coupled extension on the side-by-side Platform SAP BTP (side-by-side extensions)**
BUSINESS EXPERT, KEY USER, CITIZEN DEVELOPER	SAP S/4HANA Cloud key user extensibility	Low-code/no-code solutions (SAP Build apps, SAP Business application studio, SAP Build Process automation and others)
DEVELOPER	Developer extensibility using the SAP S/4HANA Cloud ABAP Environment (a.k.a. Embedded Steampunk[1])	Java, Node.js SAP BTP environments SAP BTP ABAP Environment (a.k.a. Steampunk[1])

	WHY CLOUD EXTENSIBILITY?	WHAT WILL BE EXPLAINED
SAP S/4HANA Cloud public edition	**Mandatory** Classical ABAP extensibility is not supported	• The SAP S/4HANA cloud extensibility model • When to use which cloud extensibility option • The ABAP Cloud development model
SAP S/4HANA Cloud private edition and on-premise **Greenfield installation**	**Start with the superior cloud extensibility model** Smoother SAP software updates Future-safe extensions for the next cloud transformation steps	• Technical setup to use the new cloud extensibility model • When to use the cloud extensibility model and when classic extensibility still has to be used • Guidance on how to handle classic extensibility in parallel with the new cloud extensions
SAP S/4HANA Cloud private edition and on-premise **Installation with converted classic extensions**	**Get the benefits described above for new and existing extensions** Developers learn the new extensibility model and can transform classic extensions step-by-step to cloud-ready extensions	• Best practices on how to manage, eliminate or transform existing classic extensions

Overview of extensibility options

The three extensibility options are not isolated from each other. In many scenarios they are combined, for example developing a side-by-side application in conjunction with a thin on-stack extensibility layer that offers more suitable remote APIs to access the SAP S/4HANA Cloud functionality.



Key user extensibility

SCENARIO	Low-code/no-code adaptations and extensions of SAP S/4HANA applications
USE CASES	Adapting UIs, adding custom fields, adding custom business objects and more.
PERSONA	Business expert, implementation consultant, citizen developer, key user.

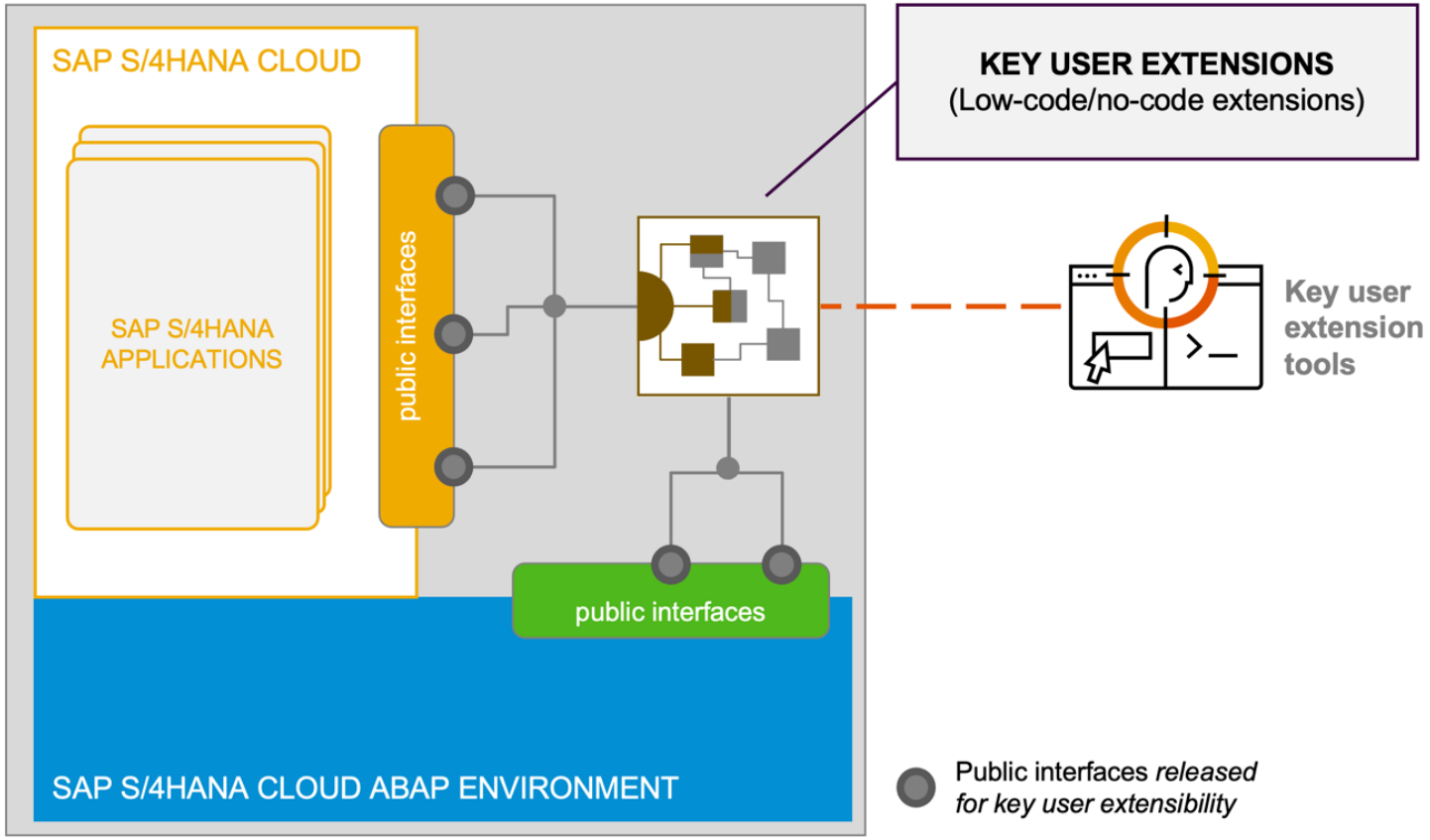
Key user extensibility (formerly also known as in-app extensibility) empowers business experts to add extensions to SAP S/4HANA Cloud solutions without the need to dive deeply into the implementation details of the underlying SAP applications. Key users typically have a deep knowledge of the SAP S/4HANA processes and business configuration. The focus is on so-called *last mile extensions* extending SAP S/4HANA user interfaces, processes, or forms using low-code/no-code tools.

Key users typically have no or only limited coding skills and therefore do not need a fully integrated development environment, with capabilities such as versioning, dependency handling, refactoring, or debugging.

The main argument for using key user extensibility is that simple extensions can be realized quicker than with developer extensibility, because the communication overhead between the business expert (responsible for specification of the extension, and later for testing and approval) and the developer (responsible for development and developer test) is avoided.

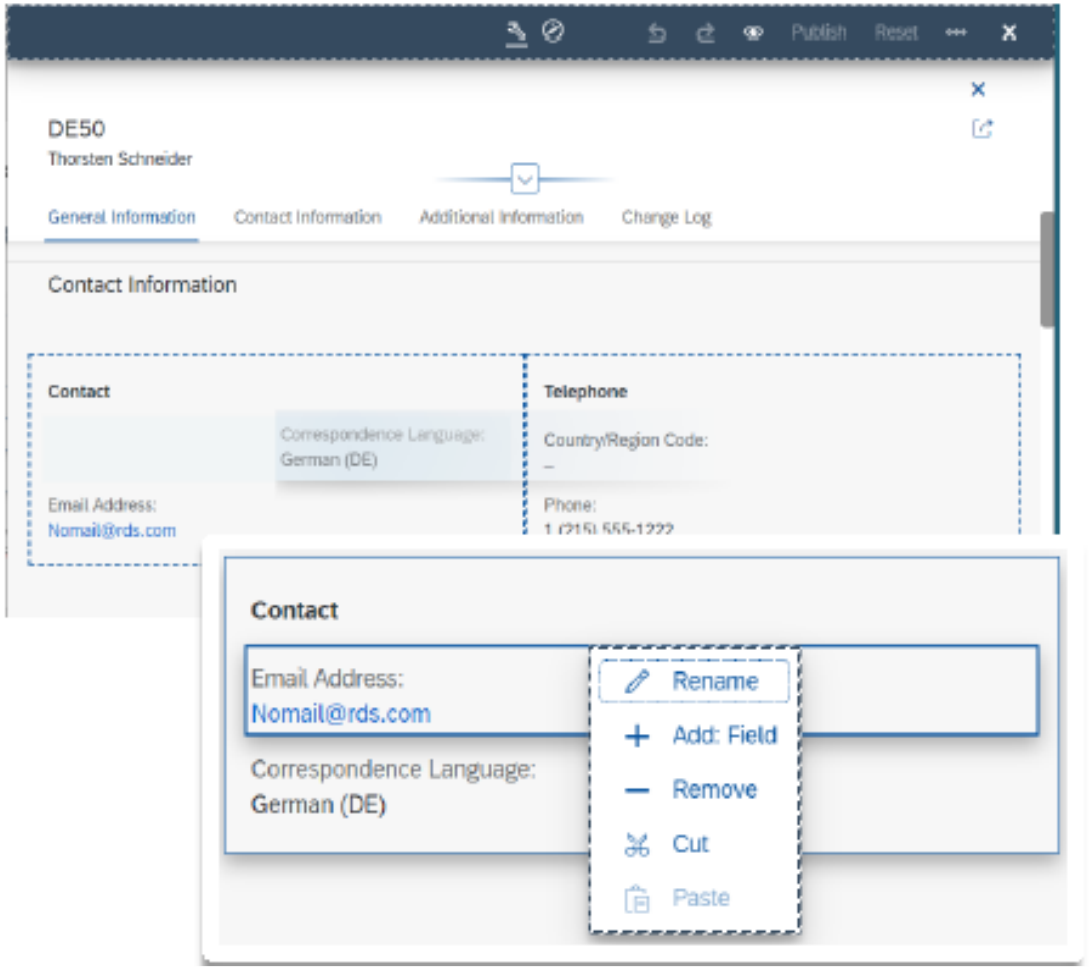
With the provided key user tools the key user can achieve the following:

- Adapt the screen layout such as moving fields and field groups, hiding fields, changing labels etc.
- Add custom fields to business objects. The custom field is then available in the entire application stack (from the UI to the database tables or for developer extensibility).
- Add custom business objects to handle custom data with very little coding efforts.
- Add custom logic to validate the custom fields using cloud BAdIs.
- Add custom fields to a process group (e.g., from sales quotation and sales order to delivery and invoice) to provide a consistent end-to-end extensibility.
- Add custom Core Data Service (CDS) views and create new analytical applications (reports, KPIs, ...).
- Copy and adapt print and email form templates.

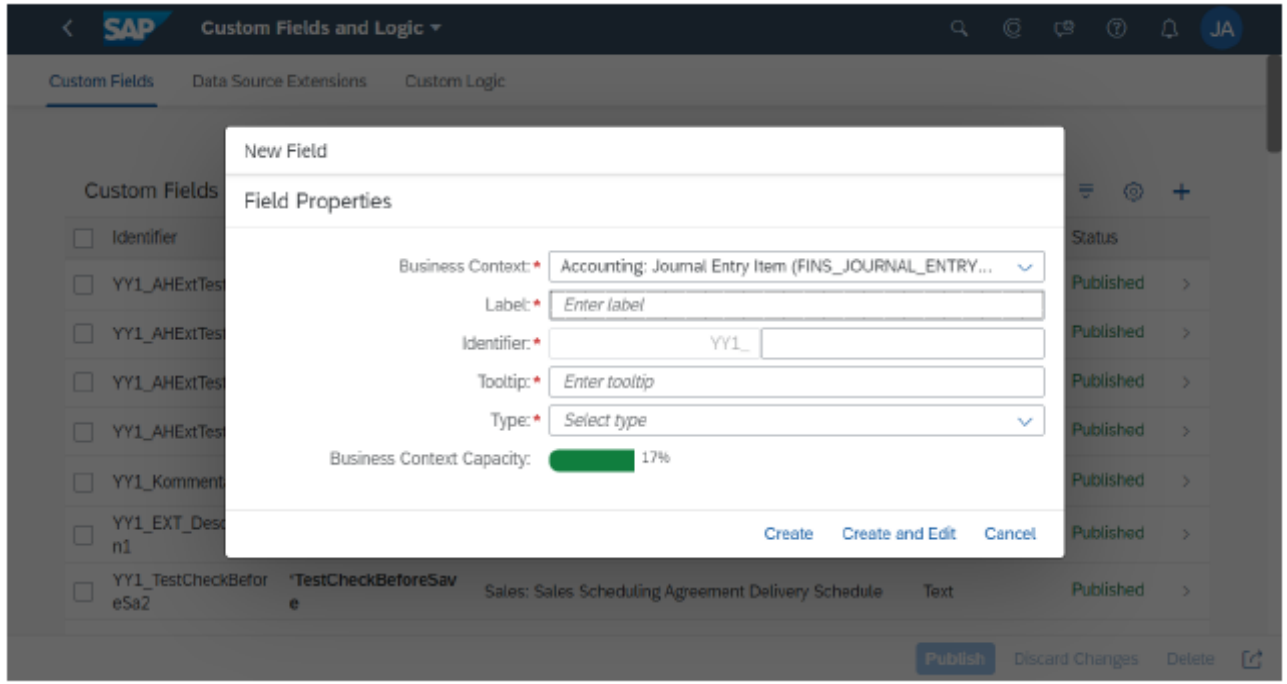


The adaptations made by a key user are registered in transport requests which can be propagated from a development environment to quality assurance and production.

Key user adaptation



Adding a custom field with key user extensibility



On-stack developer extensibility with SAP S/4HANA Cloud ABAP Environment

SCENARIO	Custom ABAP development projects that need proximity or coupling to SAP S/4HANA data, transactions, or apps
USE CASES	Custom applications with frequent or complex SQL access to SAP S/4HANA data. Custom extensions running in the same logical unit of work[2] (LUW) as SAP applications. Tailored custom remote APIs or services which serve side-by-side SAP BTP apps.
PERSONA	ABAP developer.

On-stack developer extensibility[3] is intended for development projects requiring proximity to or coupling with SAP S/4HANA data, transactions, or apps. The requirements of the extension project go beyond the scope of key user extensions and therefore require full access to development capabilities like debugging, refactoring support, version control etc.

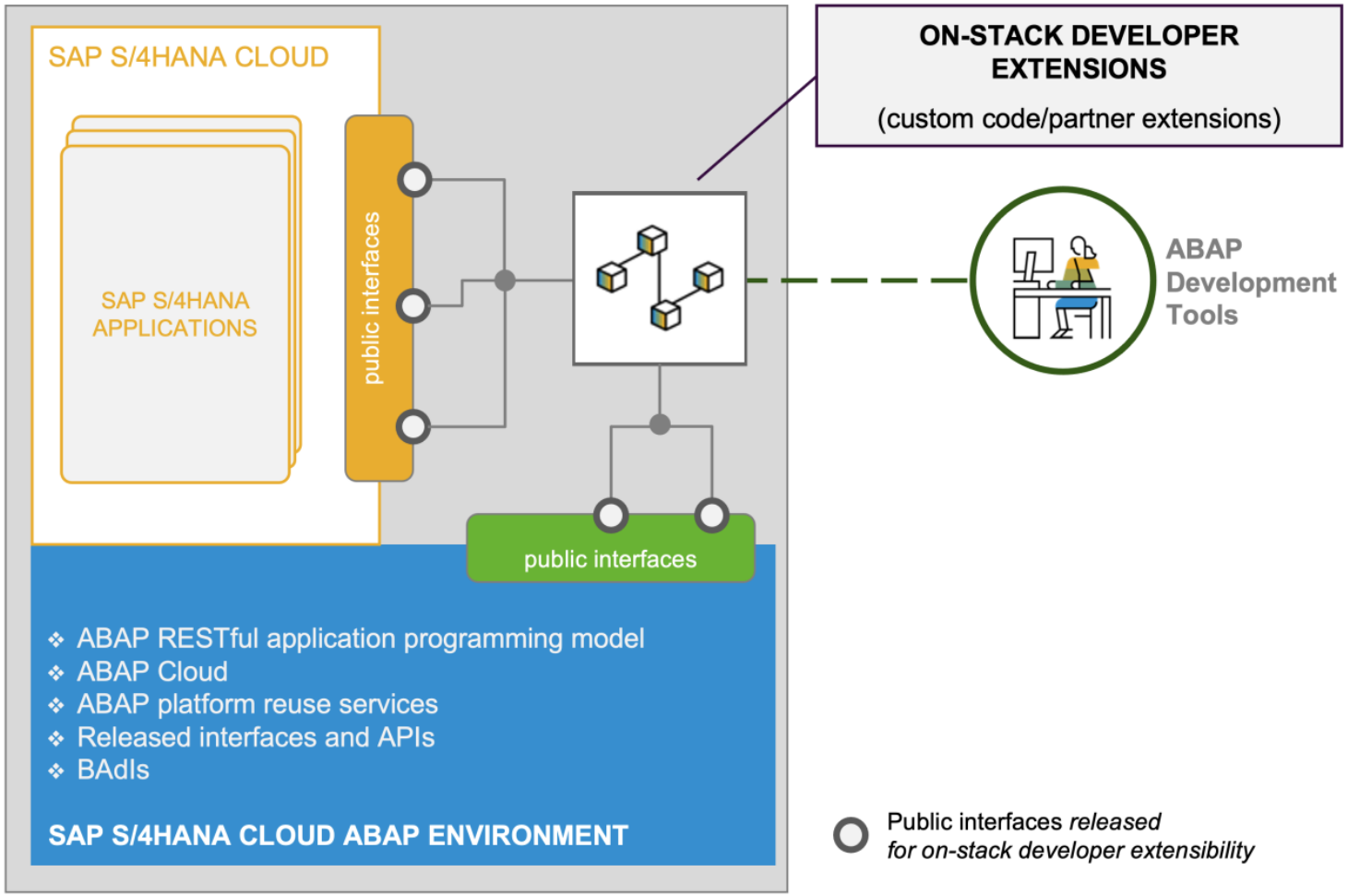
In contrast to side-by-side extensions, on-stack extensions are developed and run on the same software stack as the underlying SAP S/4HANA Cloud system. This allows extensions to access SAP S/4HANA logic and data via SAP extension points, local SAP APIs or via SQL queries.

Typical on-stack scenarios are extensions with frequent or complex SQL access to SAP data (e.g., SQL queries that join customer and SAP data), which cannot be realized easily by remote data access or data replication.

Another typical on-stack pattern are extensions that store custom data in the same logical unit of work[3] as the extended SAP S/4HANA app.

Depending on the use case, one of these options can be used to expose the functionality built using on-stack developer extensibility on an SAP Fiori UI:

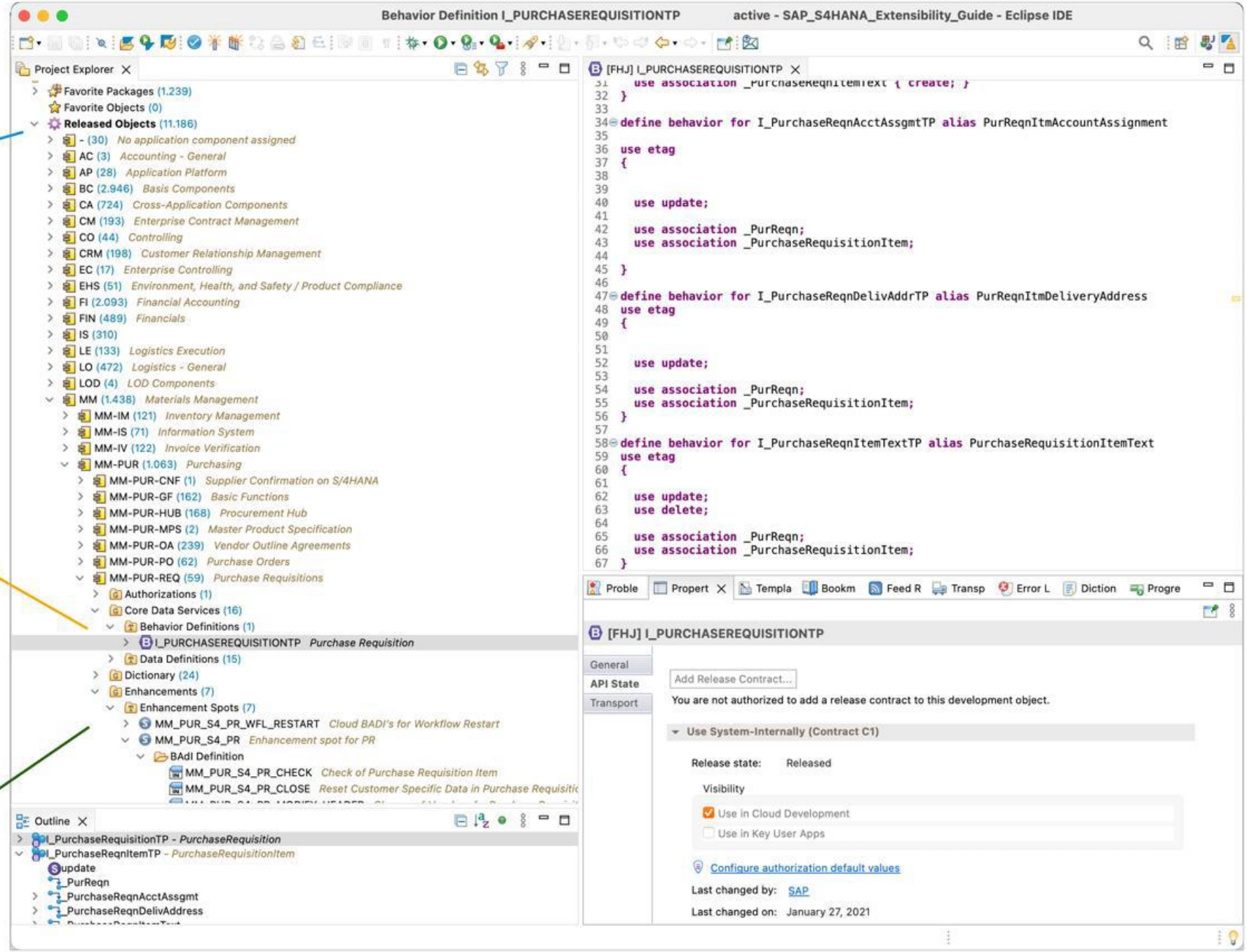
- Create a custom SAP Fiori elements or freestyle SAPUI5 app using SAP Fiori tools.
- Extend an SAP-delivered app, e.g., with additional fields using Developer Adaptation[5].
- Extend an SAP-delivered SAP Fiori elements app using ABAP CDS annotations or XML annotations



RELEASED OBJECTS TREE

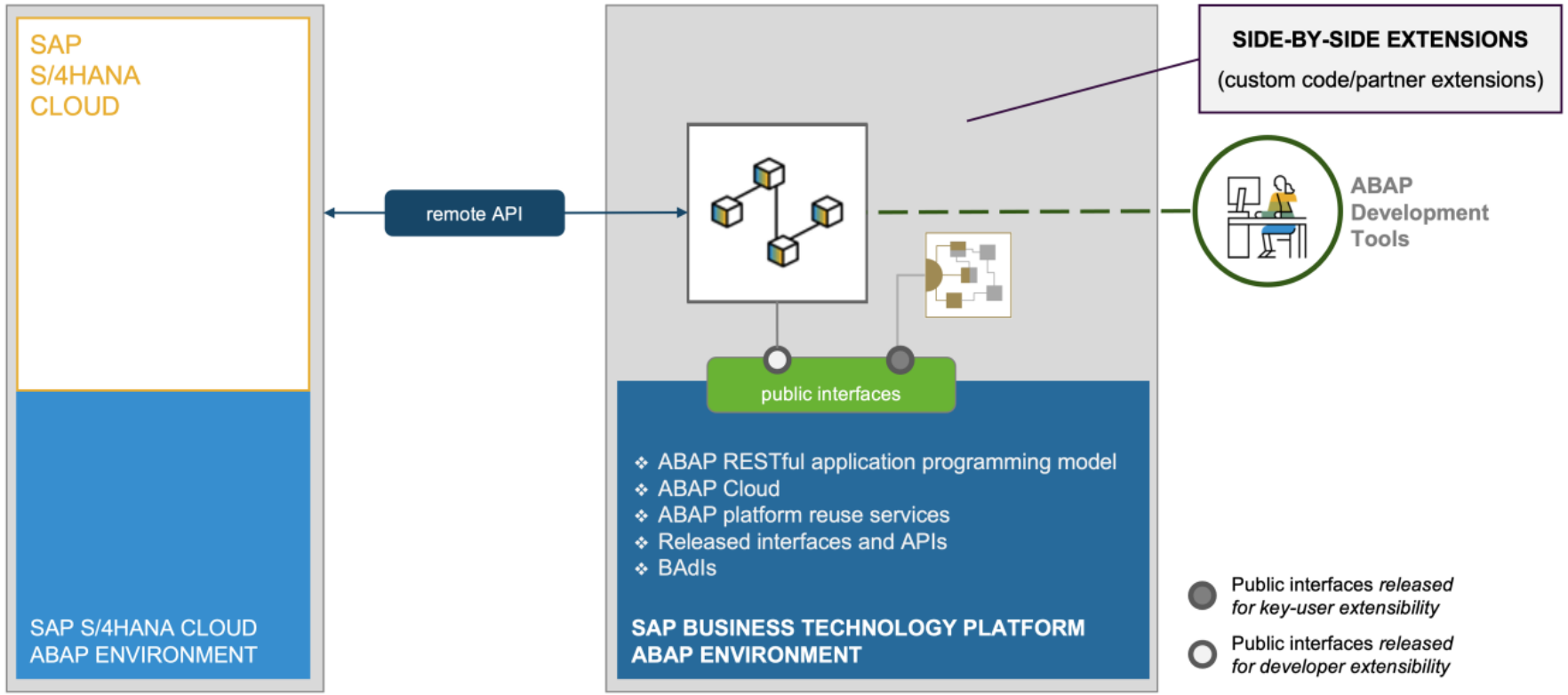
LOCAL PUBLIC APIs (RAP INTERFACES)

LOCAL EXTENSION POINTS (e.g., BAdIs)

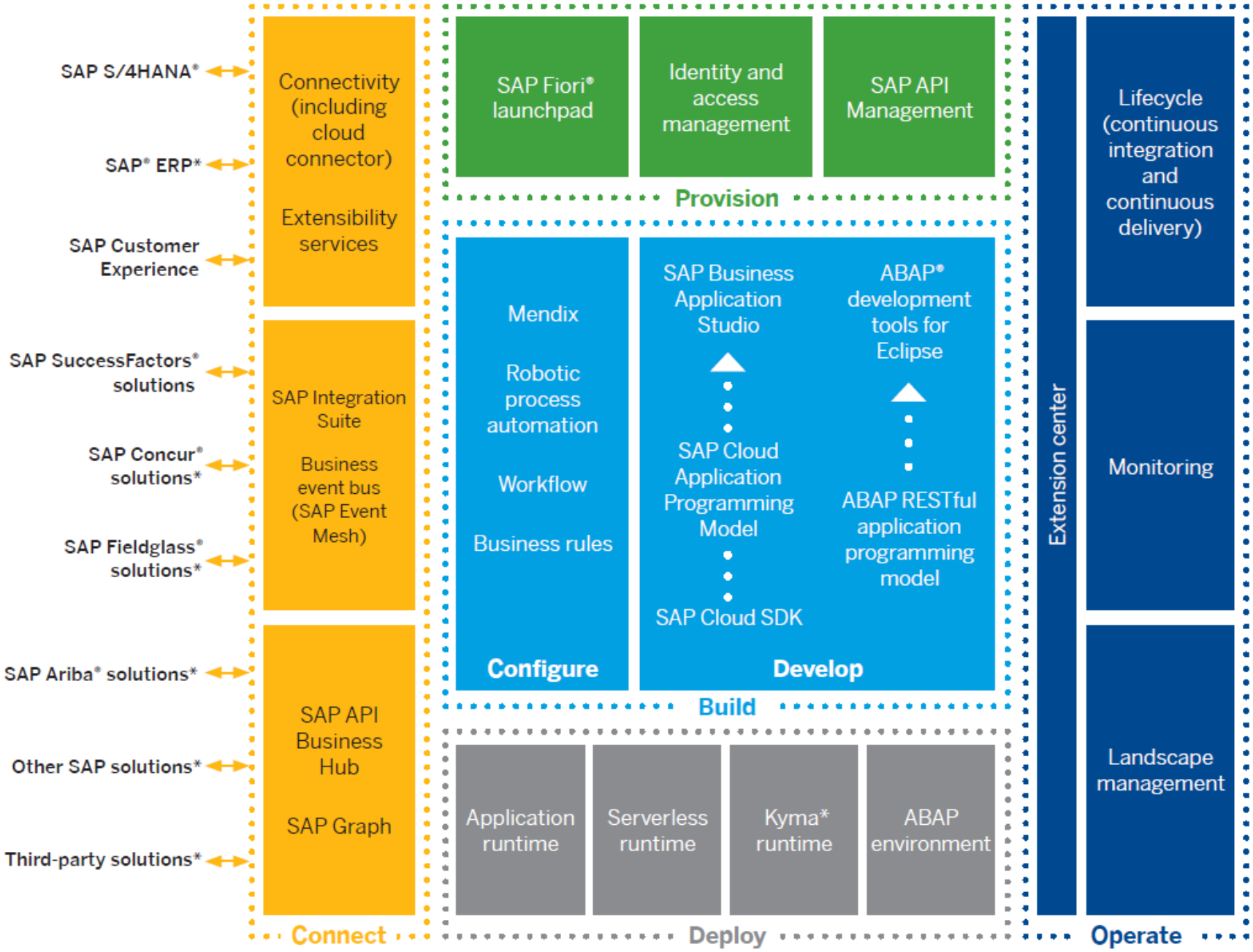


Side-by-side extensibility using the SAP BTP ABAP Environment

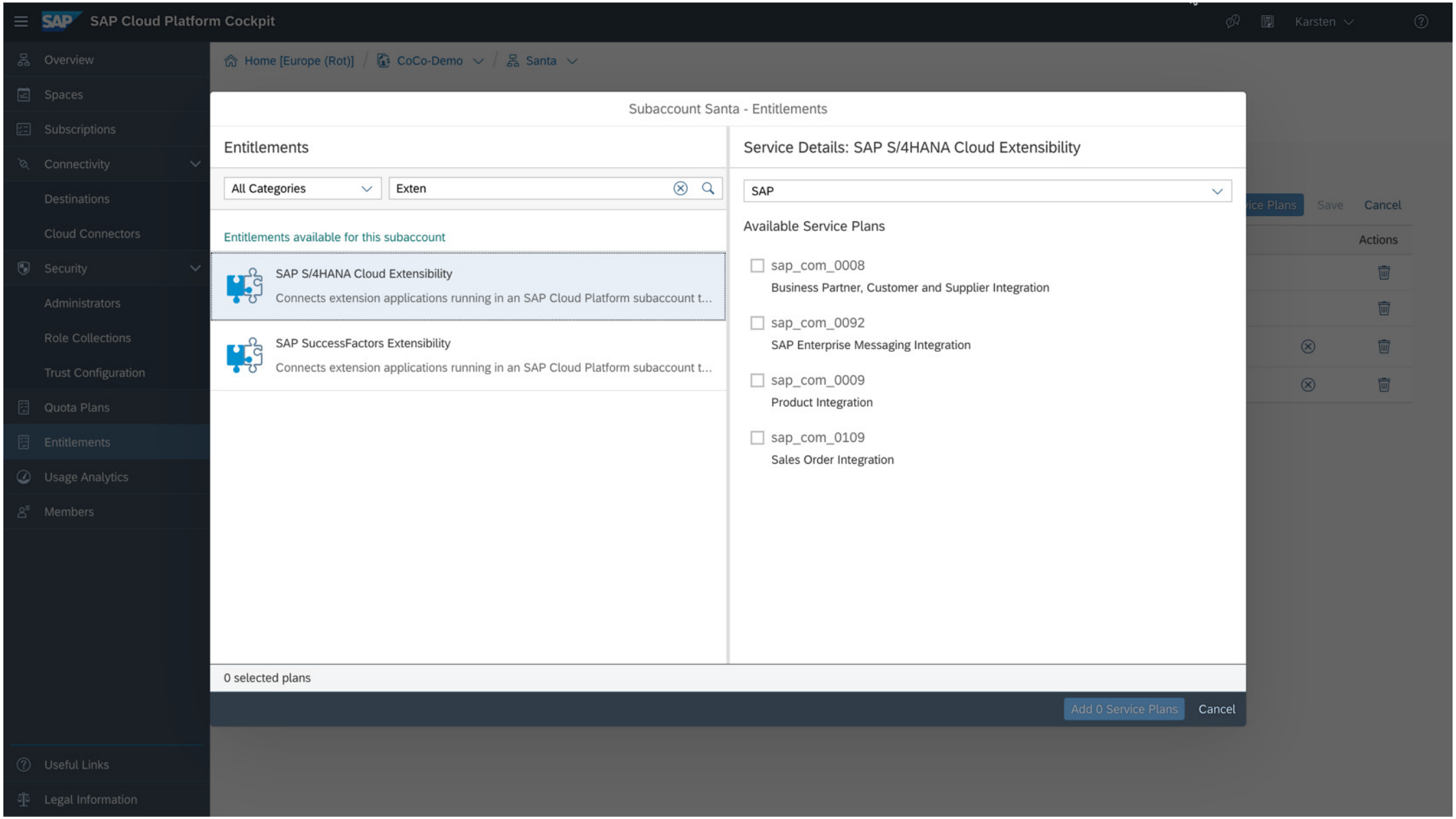
SCENARIO	Loosely-coupled applications and partner SaaS solutions
USE CASES	Custom applications for a separate target group (no ERP users). Custom application workload that shall run separated from ERP. Custom applications needing proximity to intelligent SAP BTP services like machine learning, AI etc. Solutions integrating with several ERP systems and cloud services. SaaS applications provided by partners.
PERSONA	ABAP developer.



Side-by-Side Extensibility with SAP® BTP



Available APIs Exposed Using Extensibility Services to Connect to SAP® Business Technology Platform



RAP Developer extensibility

Extensibility is available for the for the following products:

SAP BTP, ABAP environment as of 2208
SAP S/4HANA Cloud as of 2208
AP S/4HANA and SAP S/4HANA Cloud, private edition as of 2022

Side-by-Side Extensibility using SAP BTP ABAP Environment

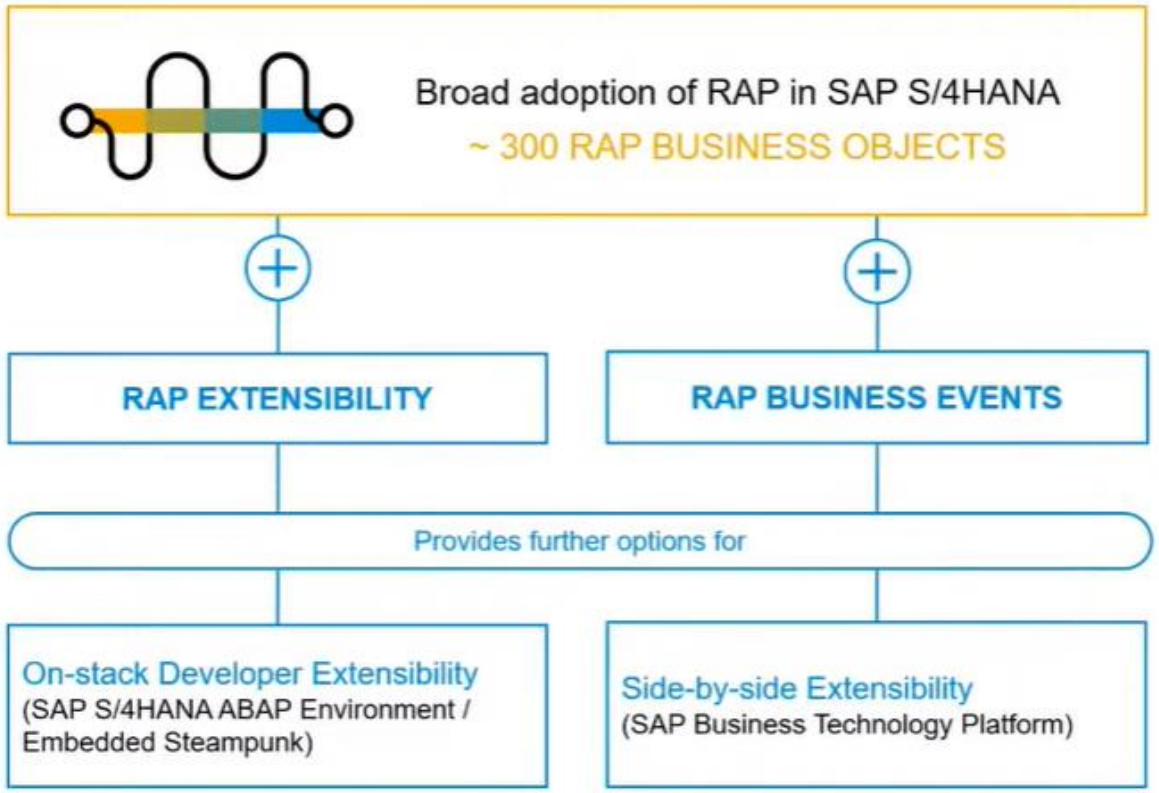
The side-by-side extensibility leverages the SAP BTP ABAP Environment to develop and run ABAP applications in the SAP Business Technology Platform as a side-by-side extension to SAP software.

Developer Extensibility

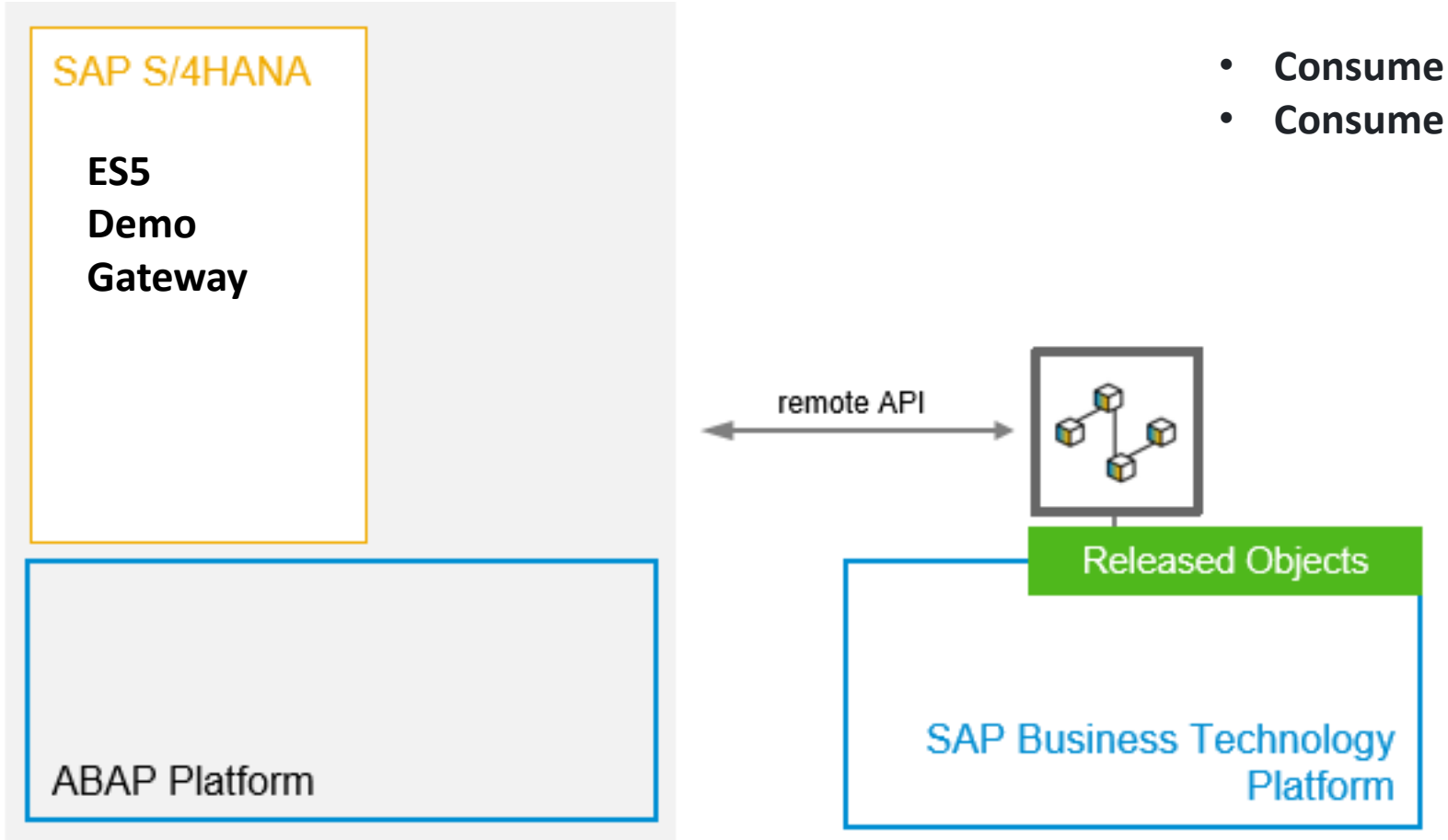
The developer extensibility offers the opportunity to build tightly coupled, cloud-ready, and upgrade-stable extensions directly on the SAP S/4HANA stack. The technical enablement of ABAP cloud-ready development requires restrictions such as the use of the cloud-optimized ABAP language and released local and remote APIs and objects. The use of the developer extensibility makes it easier to reach a clean core when building extensions which must run on the SAP S/4HANA stack.

RAP Extensions

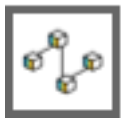
RAP extensions enable you to extend existing RAP BOs from other software components or SAP delivered RAP BOs to optimize and adapt their functionality to your specific business requirements. The RAP extension approach focuses on separation-of concerns between original BO as basis for your extensions and the actual extension artifacts that contain your additions to the data model and behavior of the original BO. Each layer in the stack of a RAP business object can be extended and adapted depending on your business use case. You can add additional fields to the data model, add new validations or determinations, actions and functions. You can also extend the BO with an additional BO entity and extend the service definition to include the new BO node. The available extensibility options depend on the extensibility-enablement of the original BO.



Side-by-side extensibility with SAP BTP ABAP Environment

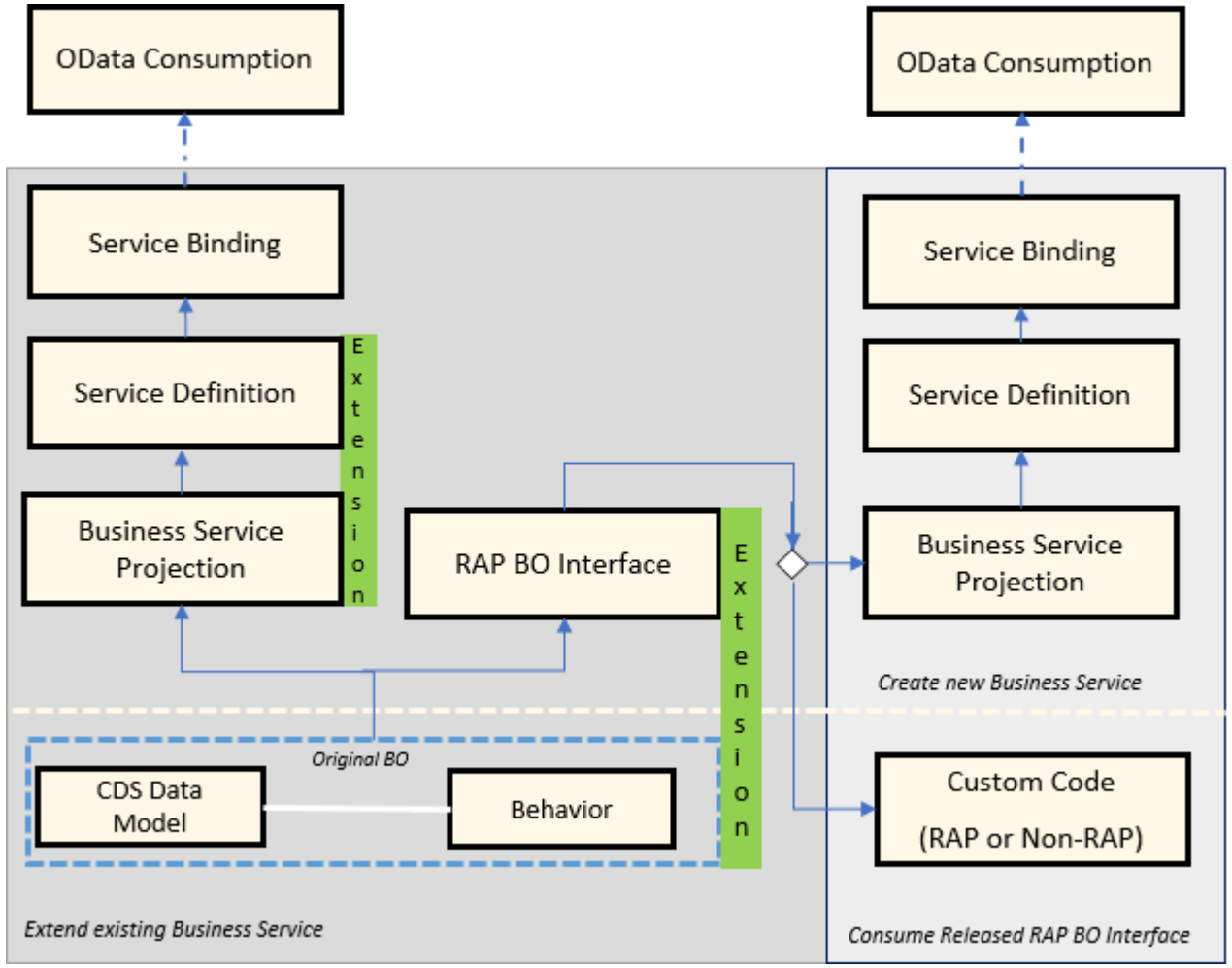


- Consume a remote API based on an OData service
- Consume a SOAP Web Service



ABAP custom code or partner add-ons based on released objects

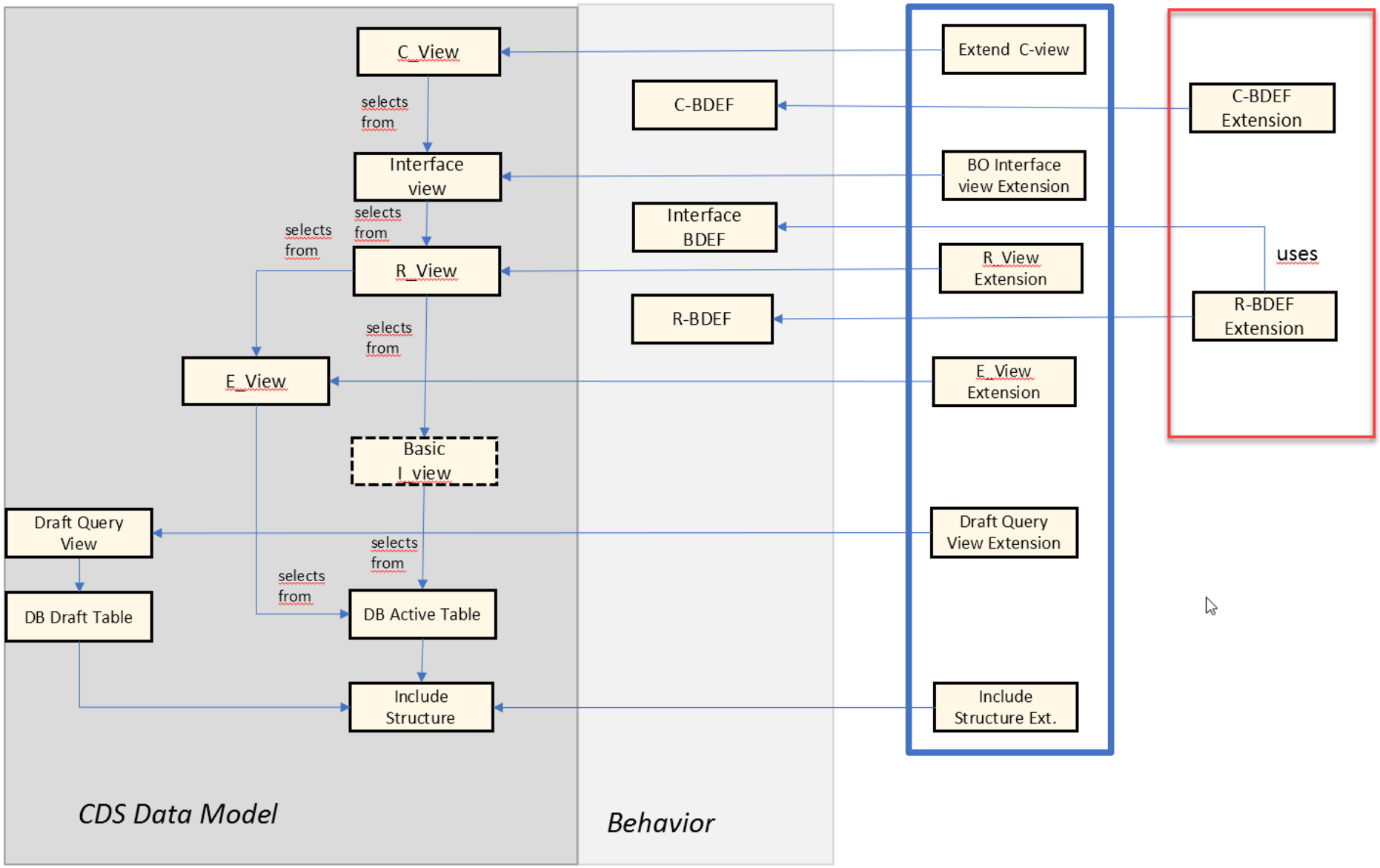
Use ABAP Cloud for developer extensibility



Extensibility Use Case

- Data Model Extension
- Behavior and Field-Related Behavior Extensions
- Node Extensibility

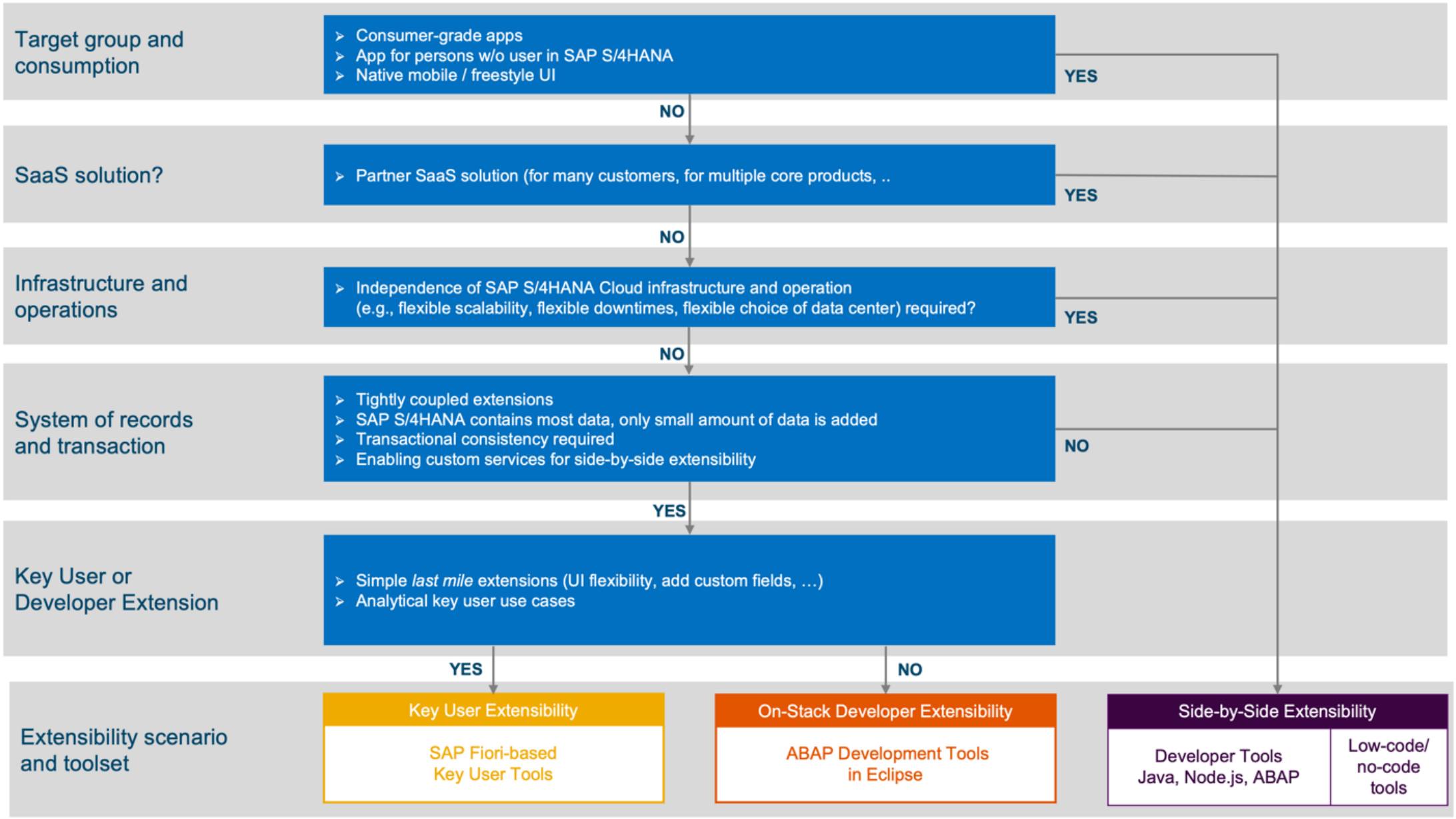
Developer extensibility



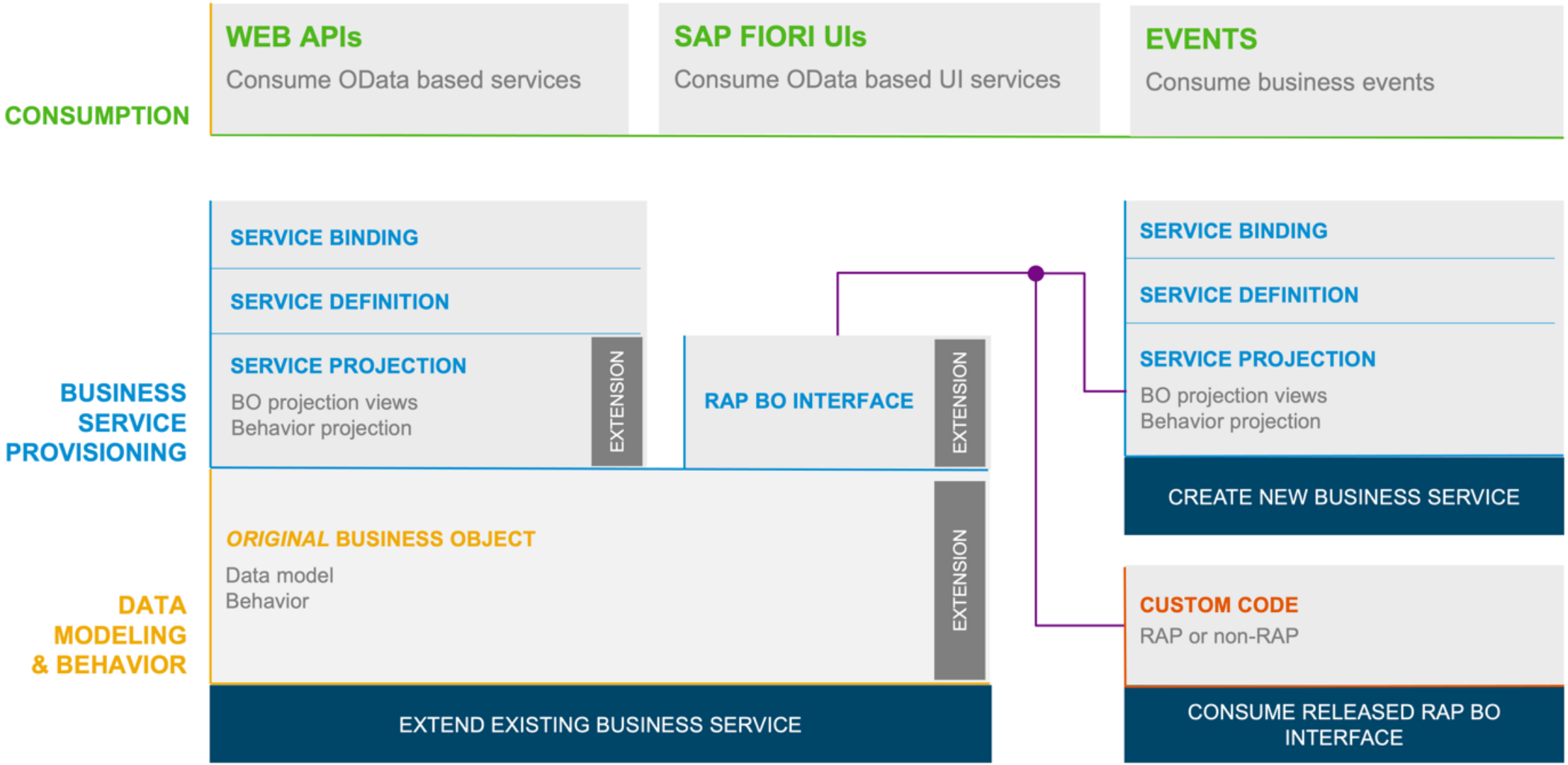
Summary of the extensibility options for SAP S/4HANA Cloud

	KEY USER EXTENSIBILITY Business expert, implementation consultant, citizen developer, key user	ON-STACK DEVELOPER EXTENSIBILITY ABAP developer	SIDE-BY-SIDE EXTENSIBILITY ABAP developer
SCENARIO	Low-code/no-code adaptations and extensions of SAP S/4HANA applications	Custom ABAP development projects that need proximity or coupling to SAP S/4HANA data, transactions, or apps	Loosely-coupled applications and partner SaaS solutions
USE CASES	Adapting UIs, adding custom fields, adding custom business objects etc.	Custom applications with frequent or complex SQL access to SAP S/4HANA data Custom extensions running in the same logical unit of work (LUW) as SAP applications Tailored custom remote APIs or services which serve side-by-side SAP BTP apps	Custom applications for a separate target group (no ERP users) Custom application workload that shall run separated from ERP Custom applications needing proximity to intelligent SAP BTP services like machine learning, AI etc. Solutions integrating with several ERP systems and cloud services SaaS applications provided by partners
BENEFIT	Fully managed and integrated in SAP S/4HANA Cloud No or only very basic development skills required	Development of extensions inside the SAP S/4HANA Cloud system No remote access or data replication Use and extend released SAP S/4HANA Cloud objects	Decoupled extensions independent of SAP S/4HANA Cloud operation and lifecycle management
	On-stack extension domain		Side-by-side extension domain

WHEN TO USE WHICH CLOUD EXTENSIBILITY OPTION



RAP Extensibility options overview



The extensibility options for RAP BOs.

EXTENSIBILITY OPTION	EXTENSIBILITY TASK
DATA MODEL EXTENSION Build full-stack data model extensions by adding new fields and associations including corresponding behavior characteristics and authorization control.	**Extensibility enablement:** Add annotations and extension include structures to the original RAP BO to enable data model extensions. For more information about enabling full-stack data extensibility, see Extensibility-Enablement for CDS Data Model Extensions. For an implementation example, see Enabling Field Extensions.
	Extension development: Extends the original RAP BO with new fields or associations including field characteristics depending on the options defined by the extensibility-enabler. For more information about how to develop data model extensions, see CDS Data Model Extensions. For an implementation example, see Develop Field Extensions.

Cont..

EXTENSIBILITY OPTION	EXTENSIBILITY TASK
BEHAVIOR EXTENSION Build additional behavior like new validations, determinations or actions including dynamic feature control and other field-related behavior.	**Extensibility enablement:** Enables data model extensibility and behavior extensibility on the original RAP BO. For more information about how to enable your BO for behavior extensions, see Extensibility Enablement for Behavior Extensions. For an implementation example, see Enabling Non-Standard Behavior and Field-Related Behavior and Enabling Standard Behavior Extensions. **Extension development:** Extends the original RAP BO with new validations, determinations or actions depending on the options defined by the extensibility-enabler. For more information about how to develop different behavior extensions, see Behavior Extensions. For an implementation example, see Develop Behavior Extensions.

Cont..

NODE EXTENSION Build additional BO nodes with own behavior and data model with node extensibility.	**Extensibility development**: Enables node extensibility on the original RAP BO. For more information about node extensibility enabling, see Extensibility-Enablement for the Node Extensibility. For an implementation example, see Enabling Node Extensions.
	Extension provisioning: Extend the original BO with new nodes that have their own data model and behavior. For more information about how to develop node extension, see Node Extensions.

Cloud-optimized ABAP language

ABAP is a programming language optimized for business applications, both at the small and at the large scale. It is designed to minimize the total costs of development for business applications. As such, the ABAP language has evolved over a long time from procedural and dynamic programming to ABAP objects and is the foundation of the ABAP RESTful application programming model. Thereby, both its capabilities and the number of ABAP keywords increased steadily, allowing for enormous flexibility but leading to complexity at the same time.

Now, for the ABAP Cloud development model, the scope of ABAP language commands has been refined to simplify and standardize ABAP development and enable cloud-ready programming. In the new language version named **ABAP for Cloud development**, only modern ABAP statements and concepts, with a focus on cloud-enabled, object-oriented design and modern programming models are supported.

Using non-supported statements results in syntax errors. To ensure the integrity of the cloud extensibility model, direct access to the file system, profile parameters, or ABAP server operations is not permitted.
ABAP Cloud is mandatory in SAP S/4HANA Cloud public edition and in SAP BTP ABAP environment and **can** be enabled in SAP S/4HANA Cloud private edition and SAP S/4HANA on-premise when desired.

NOT TO BE USED IN CLOUD DEVELOPMENT
Direct table access
BAPIs
BOPF, BOR
Classic programming models/UI technologies like Dynpro, Web Dynpro, BSP
System access via SAP GUI
Customer functions/enhancement frameworks

TO BE USED IN CLOUD DEVELOPMENT
Core Data Services (CDS)
RAP facades
RAP business objects
SAP Fiori UIs
System access via ABAP Development Tools
Released extension points for cloud development

EXTENDING A NEW SAP S/4HANA CLOUD PRIVATE EDITION OR SAP S/4HANA ON-PREMISE SYSTEM

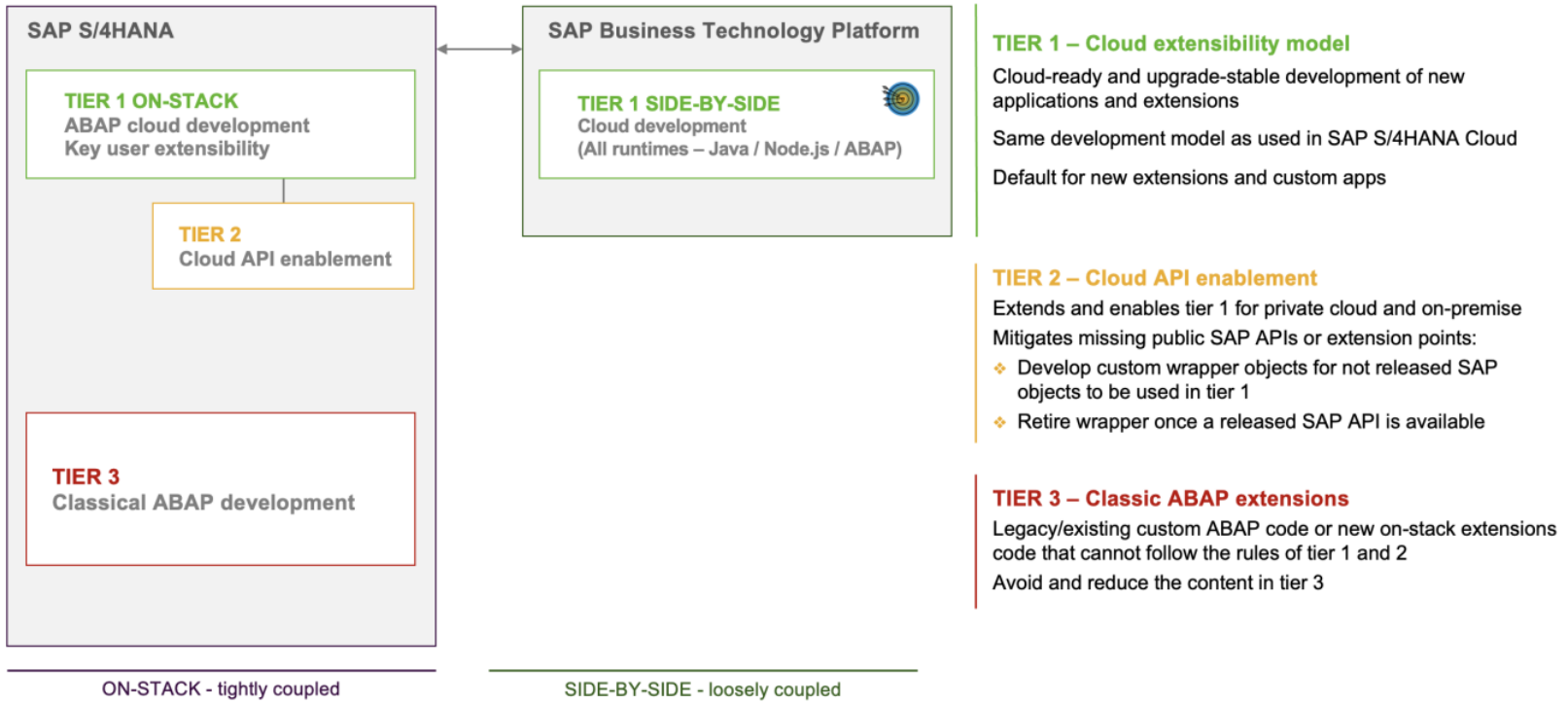
Today, in SAP S/4HANA Cloud private edition and SAP S/4HANA on-premise deployments, classic ABAP custom code is generally used for extensions. This endangers smooth upgrades and makes further cloud transformation steps more difficult. Hence, as explained in chapter 1 and chapter 2, reusing the SAP S/4HANA Cloud extensibility model also in private cloud or on-premise deployments is beneficial and recommended.
In this chapter we provide guidance on how to achieve this goal. For simplicity, we will only refer to the SAP S/4HANA private cloud, keeping in mind that the statements also hold for on-premise.
In the private cloud we need compromises with respect to the pure cloud extensibility model since the broader functional scope of SAP S/4HANA Cloud private edition is not covered by public SAP APIs and typically existing classic ABAP custom code must be handled.
Therefore, the private cloud must allow extension use cases that are not supported in the public cloud (e.g., using non-released APIs, extending Dynpro-based transactions).

To manage the coexistence of these different extensibility models in private cloud deployments we propose to work with three extensibility tiers:

• **Tier 1 – Cloud development:** default choice for all new extensions following the SAP S/4HANA Cloud extensibility model.

• **Tier 2 – Cloud API enablement:** mitigation of missing local public APIs or extension points. Here, custom wrappers for non-released SAP objects are built and released for cloud development so that they can be used in tier 1. Once SAP provides a public local API, the custom wrapper can be removed. In this sense, tier 2 serves as an extension to tier 1 which will follow the same ABAP Cloud development model besides the usage of the wrapped non-released SAP object.

• **Tier 3 – Legacy Development:** classic extensibility based on classic ABAP custom code that is not supported in the ABAP Cloud development model. The goal is to avoid developments in this tier and follow the ABAP Cloud development model as much as possible (governed via ATC) to minimize the risk of upgrade issues. Guidance on how to achieve this for transformed legacy code is given in the next chapter.

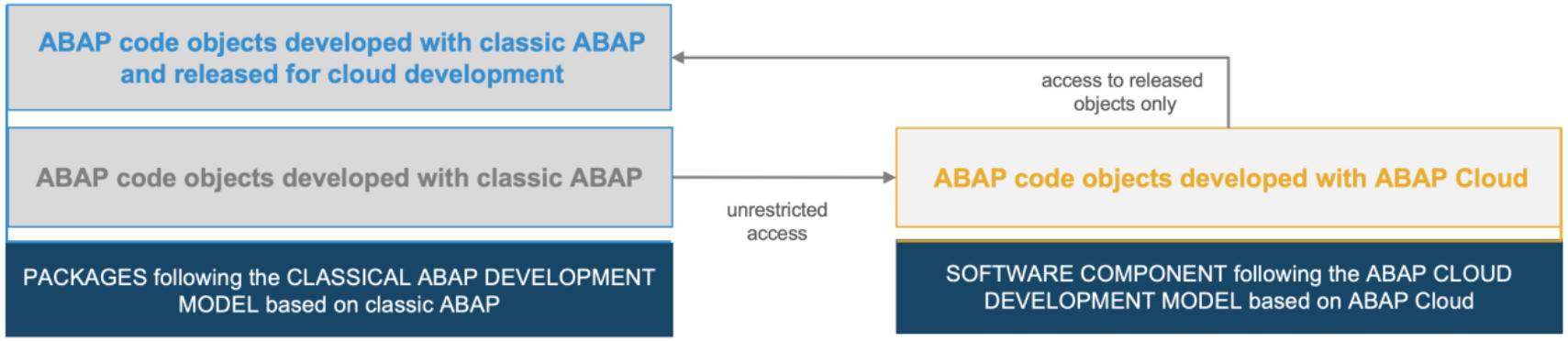


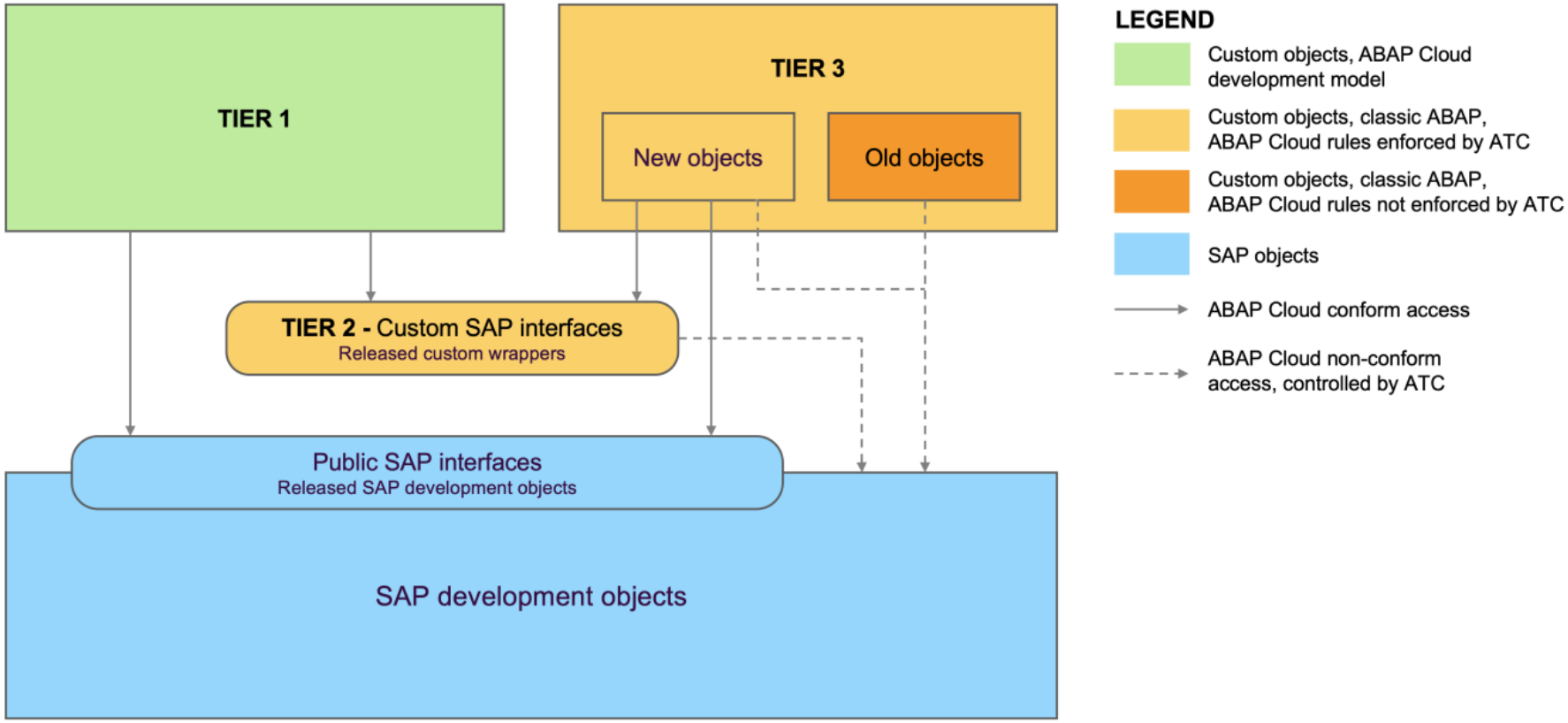
ABAP development objects in software components following the ABAP Cloud development model can only use:

- Development objects inside the own software component.
- Released SAP development objects.
- Custom development objects released for cloud development of a software component following the classic ABAP development model.

Development objects in software components following the classic ABAP development model have unrestricted access to other objects:

- Development objects inside their own software component.
- Released and non-released SAP development objects.
- Custom development objects in software components based on classic ABAP and ABAP Cloud.





GLOSSARY

Subject	Topics
Cloud Computing	What is Cloud Computing?
	Benefits of Cloud
BTP Overview	What is BTP?
	BTP Account Model (Architecture)
	Environments - CF, Kyma, ABAP on Cloud
	Global Accounts and Sub-Accounts
	Understanding ABAP Envirment
	Setup ABAP Environment
RAP Introduction	History of ABAP on Cloud
	Motivation for ABAP on Cloud – Scenario
	Common Challenges in ABAP Development
	Evolution of ABAP Programming Model
	What is RAP (Restful Application Programming)
	Flow of Restful Application Programming
	Types of Scenarios - Managed & Un-managed
	Explanation of Flight Data Model
	Defining Unmanaged Scenario
	Flow of Development (Unmanaged Scenario)
	Defining CDS Views
Behavior Definition Concept	**Introduction to Classes**
	Local Types/Classes in ABAP with Global Classes
	Implementing Class Pool Concept
	Behavior Definition Introduction
	Create Behavior Definition
Behavior Implementation	**Create Behavior implementation for Unmanaged Scenario**
	Implementing Create, Update and Delete
	Make field read only
	Making field Mandatory
	Add validation to the code
	Implement Actions - set_status_booked

Subject	Topics
Entity Manipulation Language	**Introduction to EML**
	Understanding Why we need EML
	Uses of EML
	Syntax for EML
	Implementing EML in Class
Managed Processor App	**Defining Managed Scenario**
	Use Case for Managed Scenario
	Describing Functionality of Scenario
	Creating Data Model
	Building Behavior Definition
	Implementing Processor CDS
	Building Service definition
	Creating Service Binding
Actions Determination	**Implementing the Action createTravelByTemplate**
	Implementing the Action approveTravel
	Implementing the Action rejectTravel
	Modeling Static and Dynamic Feature Control
	Implementing Validations
	Implementing the Determination
Introduce Draft Augment	**Defining Draft**
	Enable Draft Handling
	Expose Draft handling to projection Layer
	Enable Early Numbering for Travel and Booking
	Understanding Augmentation
	Integrating augment in Managed Object
Approver Scenario	Understand the persona for Approver
	Implementing Approver projection
	Annotations for Approver Scenario
	Behavior Definition for Approver
	Service Binding for Approver Scenario
	Fiori App for Approver
	Deploy Fiori App to SAP CF

Subject	Topics
Custom Entity	**Fiori App with Launchpad in BTP**
	Service Consumption Model
	Describing the Concept of Custom Entity
	Why we use Custom Entity
	Implement Custom Entity
	Create Class for Custom Entity
	Test Custom Entity
ABAP Git	**ABAP Git**
	Using Open Source Packages
	Custom Code Transformation Using abapGit
Authorization & Authentication in RAP	Global & Instence based authorization
	Authorization & Access Control
Live Build	Building a managed and an unmanaged and a hybrid RAP based application (Live)
Dynamic Control	Dynamic feature Control
Difference Between CAP & RAP	SAP Cloud Application Programming Model Vs the ABAP RESTful Application
Consuming Extenrnal API's	**I. Connecting to On-Premise SAP Systems**
	a. Configuring the HTTP and RFC Connection to an On-Premise System
	b. Consuming the RFC Function Module
	c. Consuming the OData Service from the Backend
	d. Consuming the SOAP Web Service
	e. Inbound RFC Concepts
	II. Connecting to Cloud Systems
	a. SAP API Business Hub
	b. Consuming an SAP S/4HANA Cloud System API
	III. SAP Business Technology Platform Services
	a. APIs from the SAP API Business Hub
	b. Implementation Examples
	IV. Services on Non-SAP Platforms
	a. APIs from Other Providers
	v. Integration with Adobe (services & form development)
Monitoring	Debugging ABAP Applications
	Technical Monitoring Cockpit
	SQL Trace
	Authorization Model, Business Roles, and Authorization Trace
Security	Maintaining the Certificate Trust List
	Maintaining Clickjacking Protection - In Genaral
	Managing Content Security - In Genaral
	Manage Software Components App - Object Management
	gCTS
	XCO Library