Unit 1 – Introducing the SAP Integration Suite

Developing with SAP Integration Suite

C_CPI_2404

Agenda

- Distributed architecture
- API first approach
- Exercise scenario
- Operating modes of API architecture
- REST, OData
- SAP Gateway Demo System
- SAP Graph
- Key summary points

Distributed Architecture

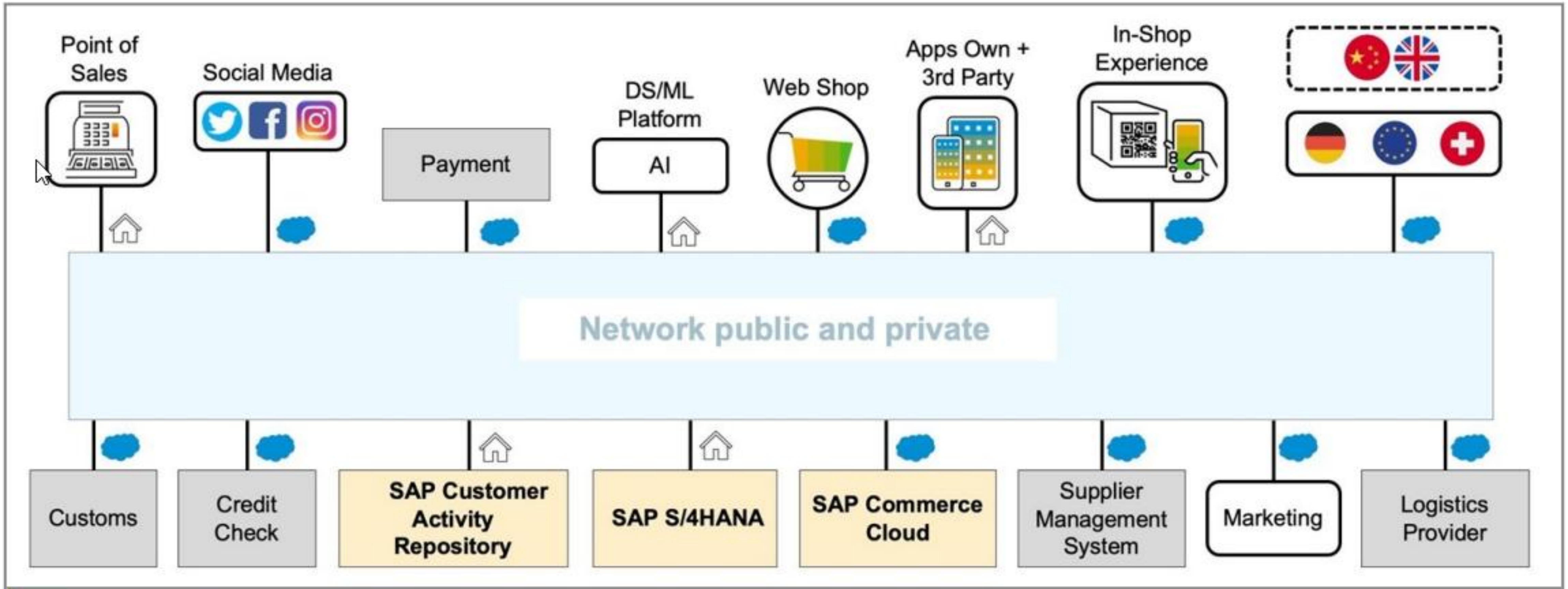


Figure 1: Example of a Dstributed Architecture

Challenges of Distributed Architecture

- Different transport and message protocols
- Release management
- Monitoring, Observability
- Security
- Latency, Quality of service
- Documentation

Heterogenous systems: Expensive, complex solutions are needed

API first approach

- Focus on the API to create applications that can be easily connected to each other
- API Provider – Provides the interface
- API Consumer – Consumes the interface

Types of APIs

- Databased APIs
- Object Oriented APIs
- Remote APIs
- Messaging APIs

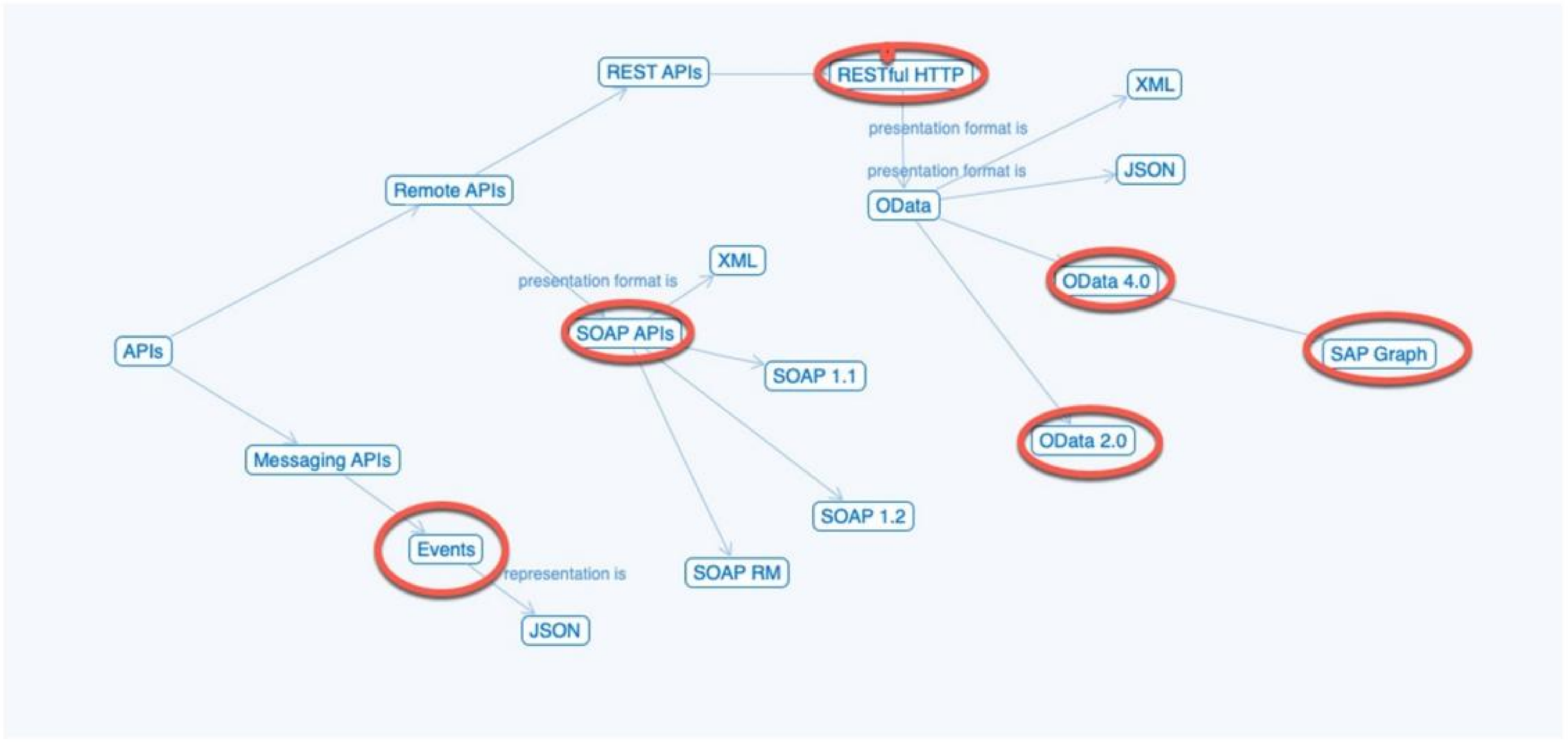


Figure 5: Relationship between APIs

SOAP, REST

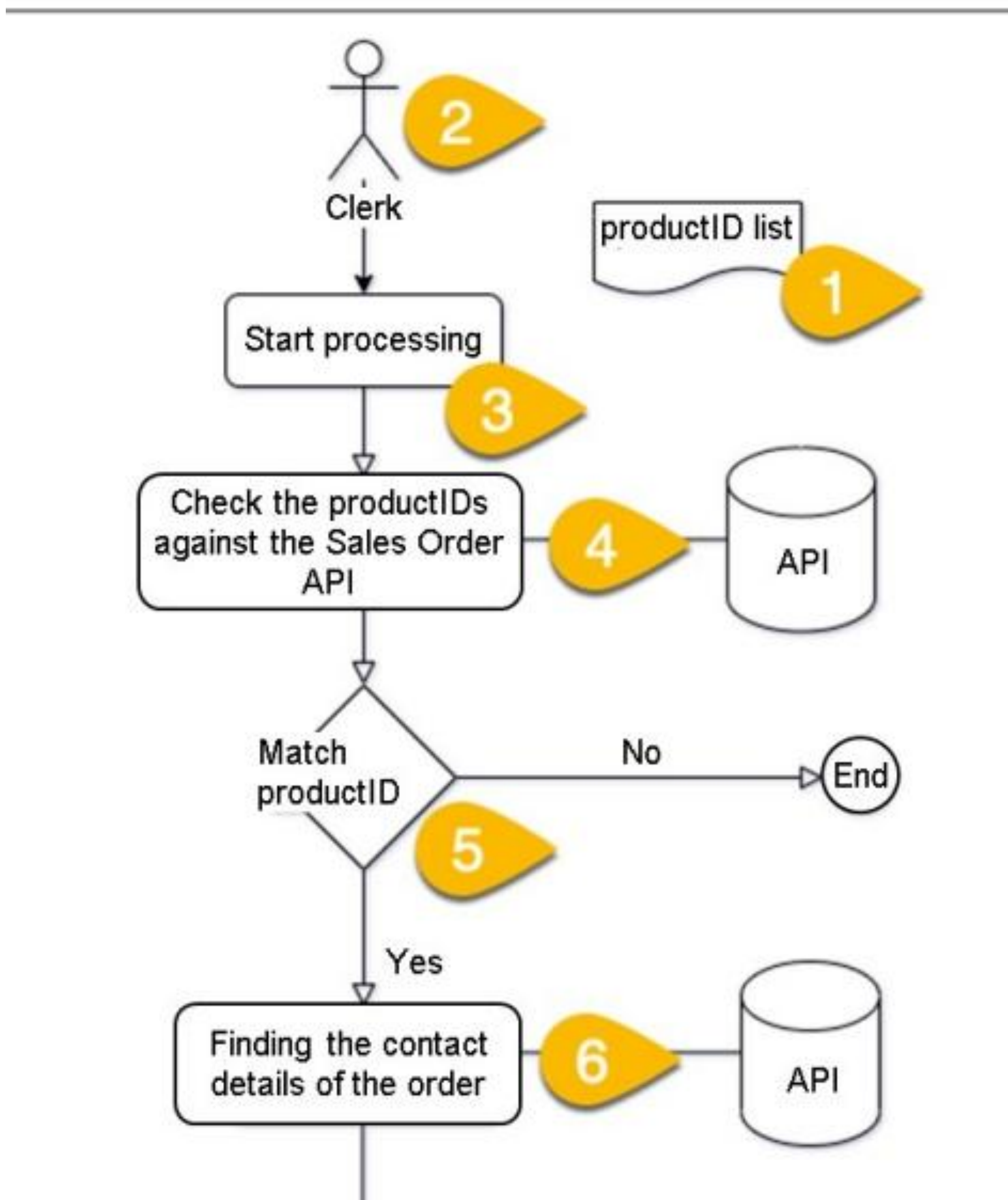
Type of API	Description language
SOAP	Web Services Description Language (WSDL)
REST	Open API • Used in API management • Interface definition language for describing, producing, consuming and visualizing RESTful web services RAML

Contract between API Provider and API consumer

- Implementation first approach
  - Implementation created first by API Provider
  - Contract generated automatically which is used by API Consumer
- Contract first approach
  - Contract created first
  - Both API Provider and API Consumer can simultaneously start working against the contract

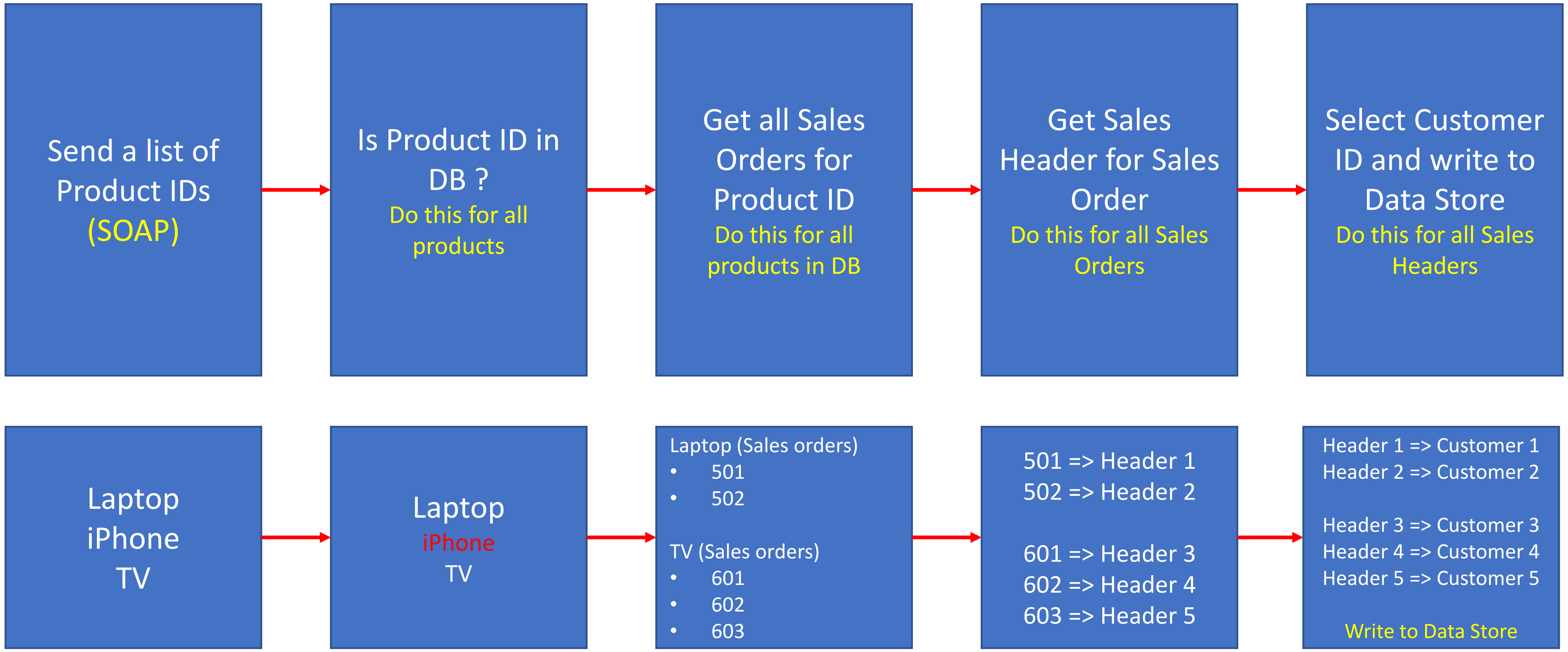
Business Scenario

- Company A sells goods to customers
- Some products cannot be delivered on time
- Inform customers who ordered these products about delay



2
Clerk
productID list
1
Start processing
3
Check the productIDs against the Sales Order API
4
API
Match productID
No
End
5
Yes
Finding the contact details of the order
6
API

Task flow


Send a list of Product IDs (SOAP)
Is Product ID in DB ? Do this for all products
Get all Sales Orders for Product ID Do this for all products in DB
Get Sales Header for Sales Order Do this for all Sales Orders
Select Customer ID and write to Data Store Do this for all Sales Headers
Laptop iPhone TV
Laptop iPhone TV
Laptop (Sales orders) • 501 • 502 TV (Sales orders) • 601 • 602 • 603
501 => Header 1 502 => Header 2 601 => Header 3 602 => Header 4 603 => Header 5
Header 1 => Customer 1 Header 2 => Customer 2 Header 3 => Customer 3 Header 4 => Customer 4 Header 5 => Customer 5
Write to Data Store

Demo of the exercise

- SOAP Adapter $top=2
- SOAP Adapter everything

Parts of the Exercise

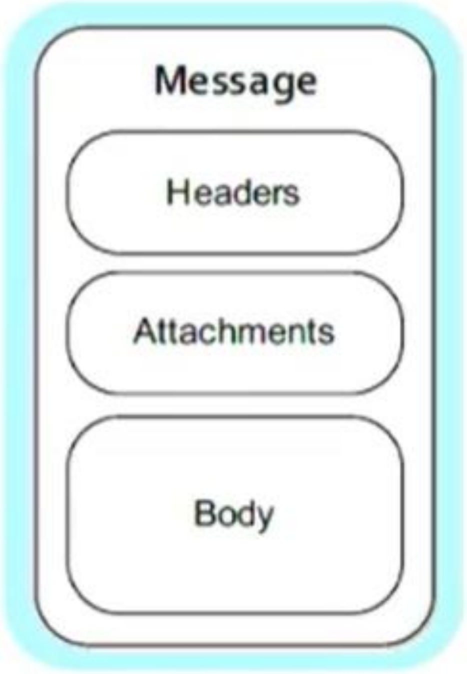
- Part 1
  - Create APIs on API Business Hub Enterprise Portal (API Management)
- Part 2
  - Consume the APIs and process data (Cloud Integration)
- Part 3 (independent)
  - Investigate sample integration flows (Cloud Integration)

Basic concepts of Cloud Integration flow…

Message

Fundamental entity **containing the data** being carried and routed in Camel

- Messages have a body (a payload), headers, and optional attachments
- Messages are uniquely identified with an identifier of type `java.lang.String`
- *Headers*
  - Headers are values associated with the message
    - ⇨ Sender identifier, hints about content encoding, authentication information,…
  - Headers are name-value-pairs
    - ⇨ Name is a unique, case-insensitive string
    - ⇨ Value is of type `java.lang.Object`
- *Attachments*
  - Optional – typically used for Web service and e-mail components
- *Body*
  - Type: `java.lang.Object` → any kind of content is allowed

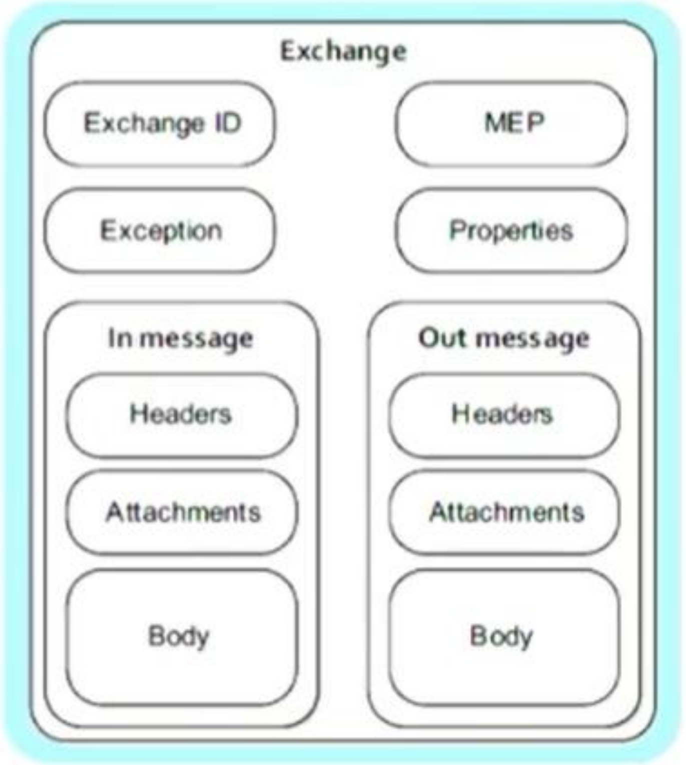


Basic concepts of Cloud Integration flow…

Exchange

The **message's container** during routing

- Provides support for various interaction types between systems, known as Message Exchange Patterns (MEP)
  - `InOnly`: a one-way message (e.g. JMS messaging)
  - `InOut`: a request-response message (e.g. HTTP-based transports)
- *Exchange ID*: a unique ID that identifies the exchange
- *MEP*
  - `InOnly`: exchange contains an "in message" **only**
  - `InOut`: exchange contains an "in message" **and** an "out message" containing the reply message for the caller
- *Exception*: If an error occurs during runtime, the Exception field will be filled
- *Properties*: Similar to message headers, but they last for the duration of the entire exchange; they contain global-level information; you can store and retrieve properties at any point during the lifetime of an exchange



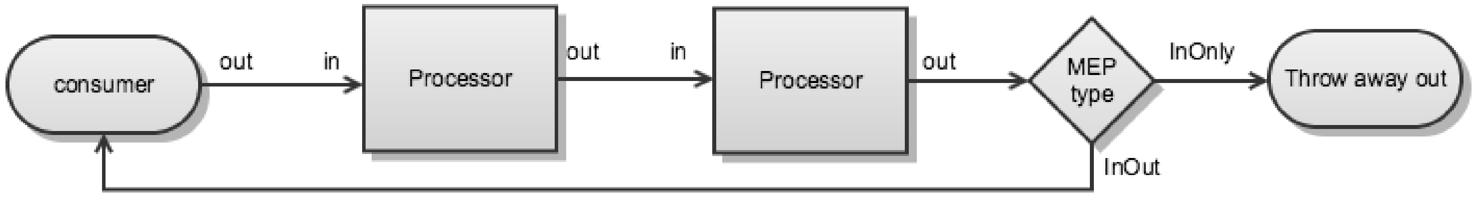
MESSAGE EXCHANGE PATTERNS AND THE EXCHANGE OBJECT

The Camel API is influenced by APIs such as JBI specification, CXF which defines a concept called Message Exchange Patterns (MEP for short).

The MEP defines the messaging style used such as one-way (InOnly) or request-reply (InOut), which means you have IN and optionally OUT messages. This closely maps to other APIs such as WS, WSDL, REST, JBI and the likes.

The Exchange API provides two methods to get a message, either `getIn` or `getOut`. Obviously the `getIn` gets the IN message, and the `getOut` gets the OUT message.

FLOW OF AN EXCHANGE THROUGH A ROUTE



- The out message from each step is used as the in message for the next step
- if there is no out message then the in message is used instead
- For the InOut MEP the out from the last step in the route is returned to the producer. In case of InOnly the last out is thrown away

TIP

Beware of `getOut` to check if there is an out message

`exchange.getOut` creates an out message if there is none. So if you want to check if there is an out message then you should use `exchange.hasOut` instead.

Operating Modes of API Architectures

- Request driven architecture
- Event driven architecture
- Combination

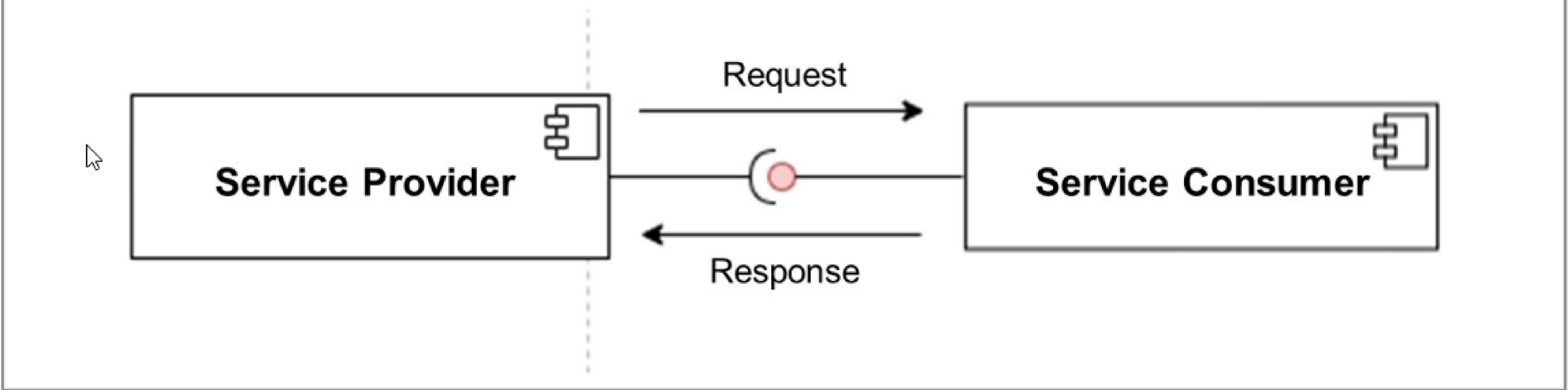


Figure 6: Request Driven Architecture

Event driven architecture

Pull Variant

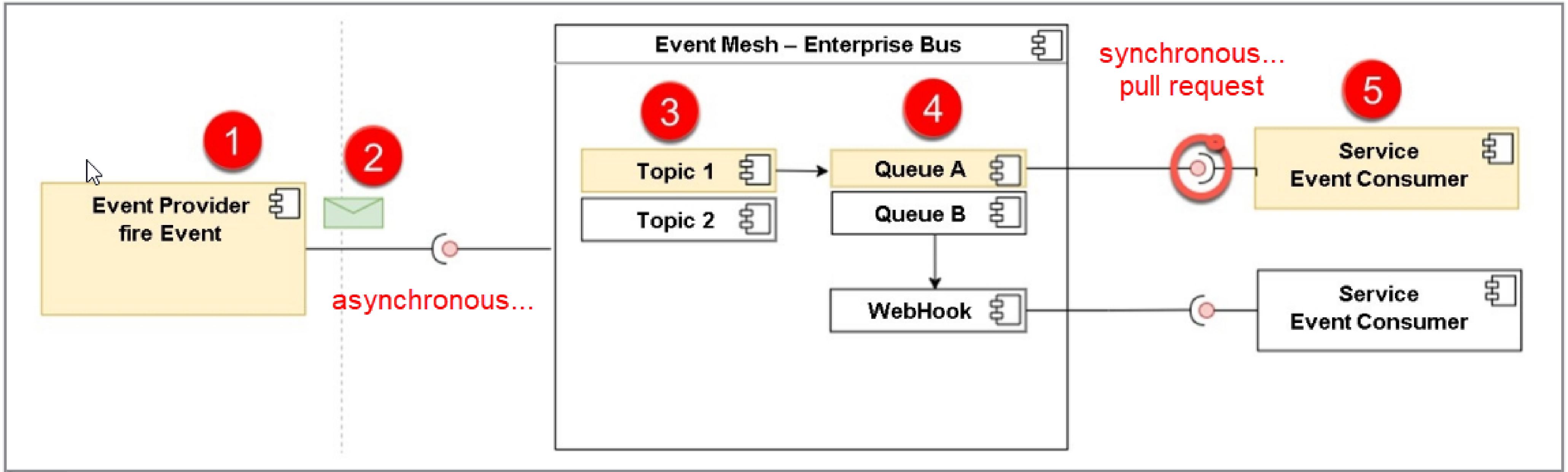


Figure 7: Pull Variant

Event driven architecture

Push Variant

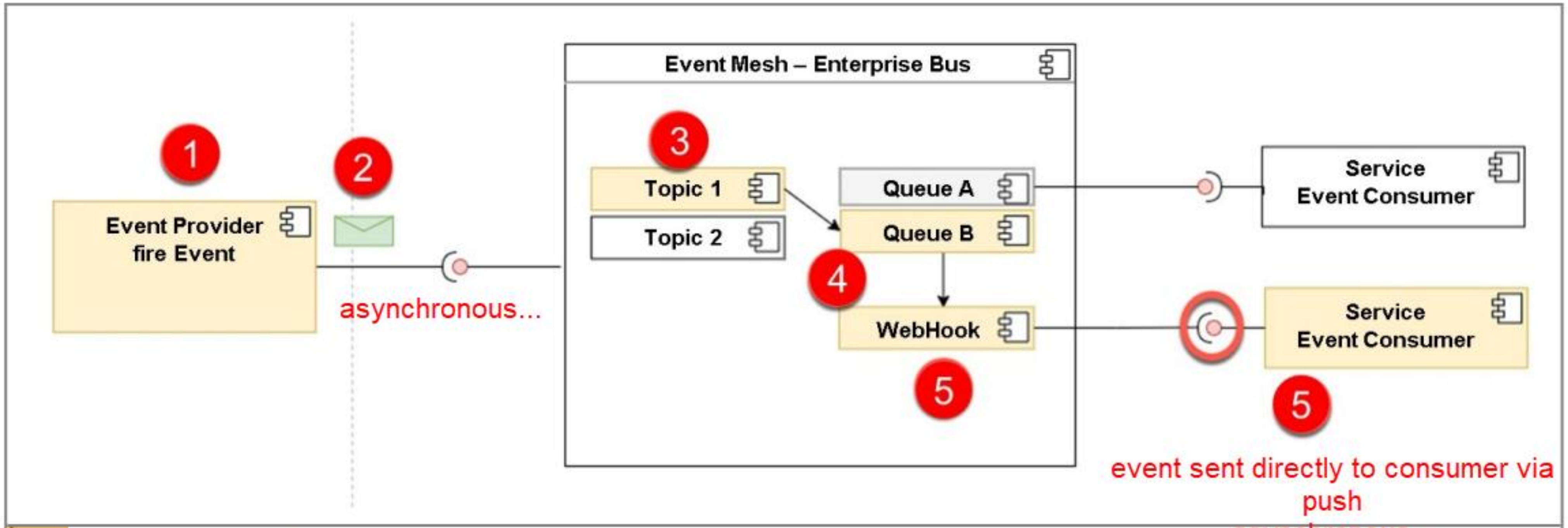


Figure 8: Push Variant

Combination: Request-Driven & Event-Based

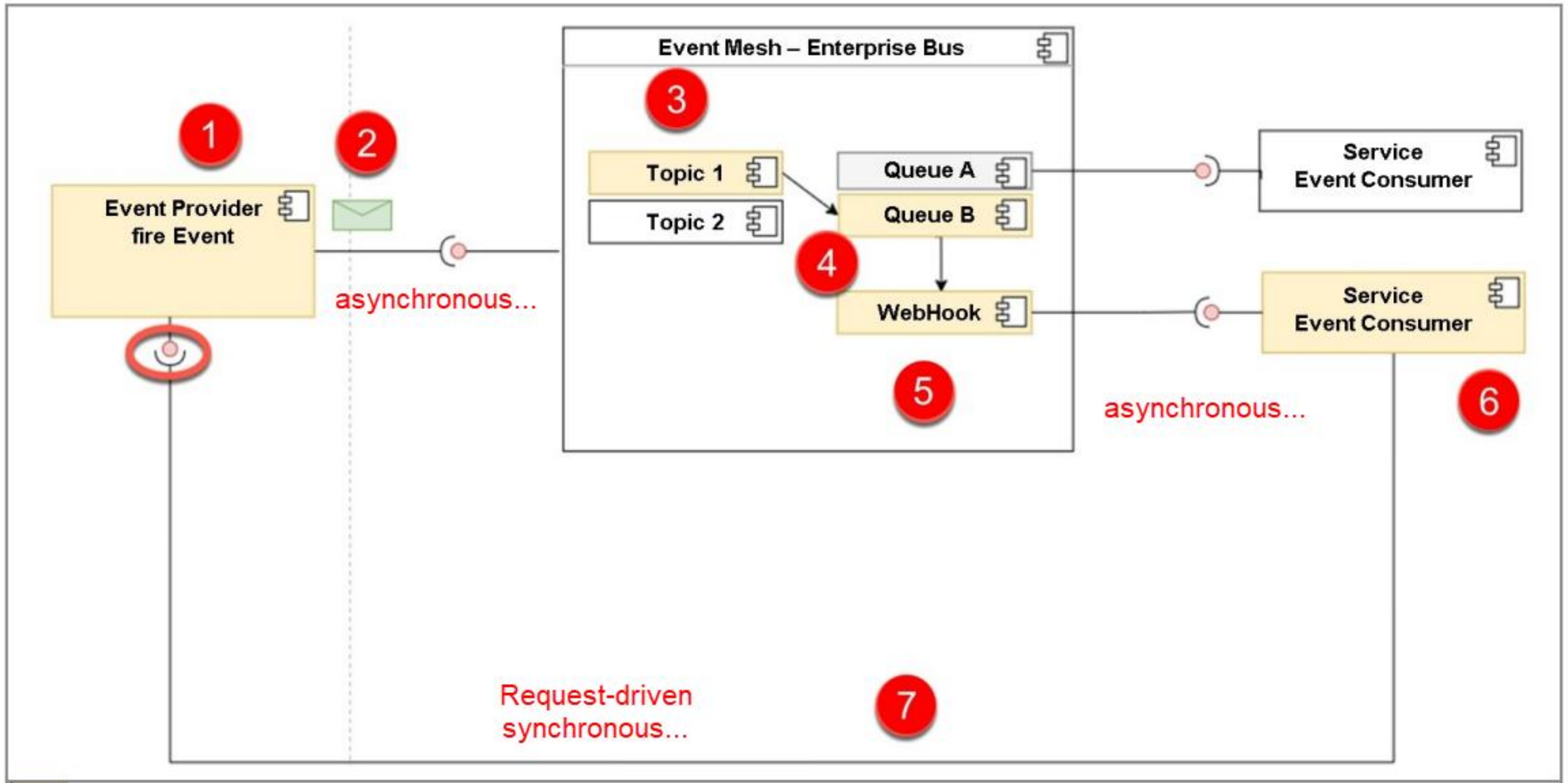


Figure 9: Combination of Request-Driven and Event-Based Architecture

REST

- Architectural properties
  - Simplicity of uniform interface
  - Scalability, modifiability, reliability etc.
- Architectural constraints (6)
  - Client server architecture
  - Stateless
  - Cache ability
  - Layered system
  - Code on demand (optional)
  - Uniform interface
- Uses standard HTTP methods and supports many media types

OData

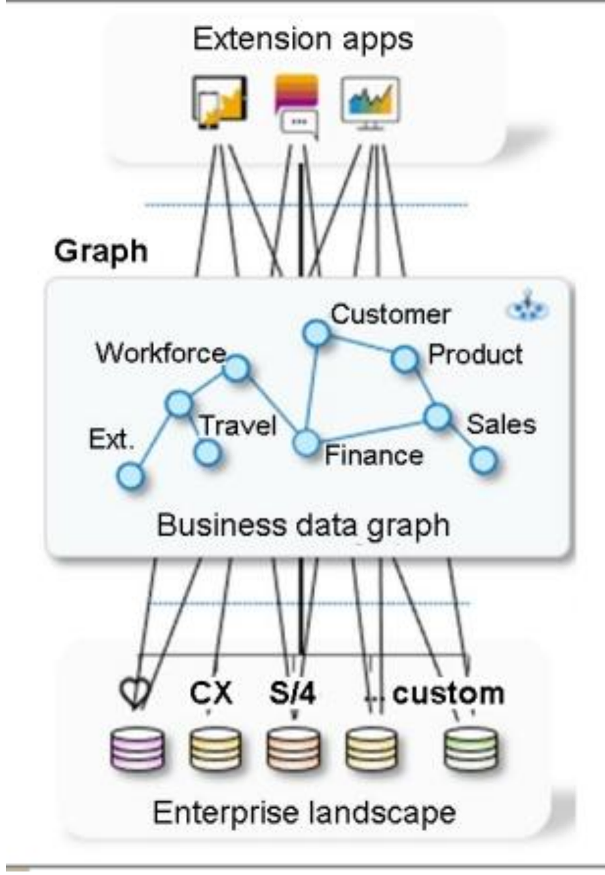
- Architectural constraints
  - Resource identification
  - Fixed documents
    - Service document
    - Metadata document
  - Dynamic resources
  - Resource operation
  - Querying
  - Resource representation

SAP Gateway Demo System

Publicly accessible environment provided by SAP to allow developers and integrators to explore and test OData Services – typically through SAP Gateway

- Check if product is available: HT-1000
- Get all sales order for product: HT-1000
- Get customer details for each sales order for product: HT-1000

SAP Graph



- API based on OData v 4.0
- Connects entities from different sources in one API
  - For example, SAP S/4HANA Cloud, Sales Cloud and others
- Existing SAP Graph APIs for various entities
- Can be programmed with Node.js, SAP API Management (low code)

Key Summary Points – Unit 1

- Request driven architecture (Synchronous)
- Event driven architecture (Synchronous + Asynchronous)
  - Pull variant, Push variant
- Combination

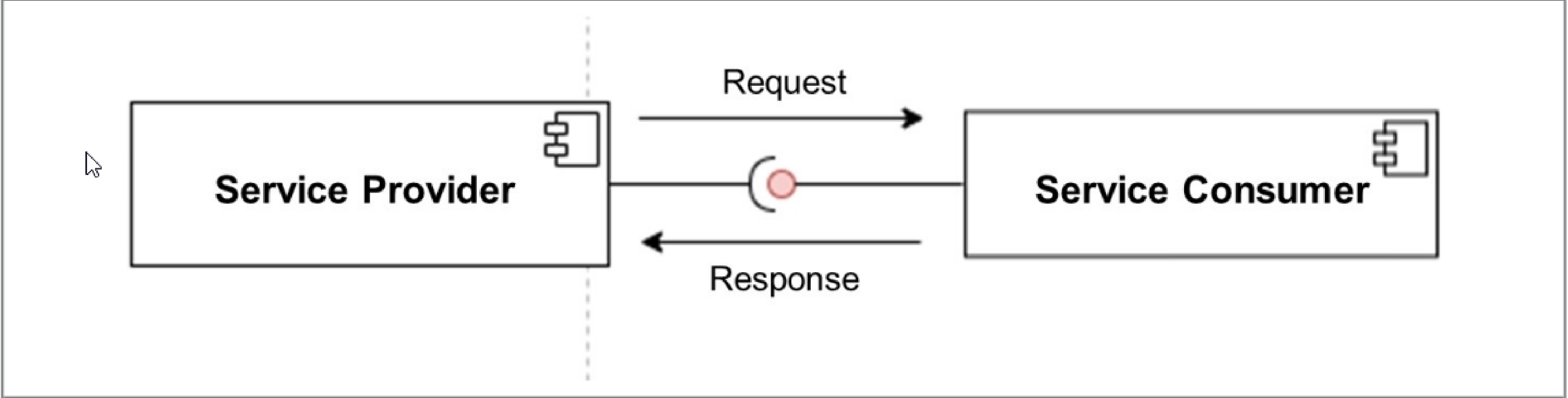


Figure 6: Request Driven Architecture

Key Summary Points – Unit 1

Pull Variant

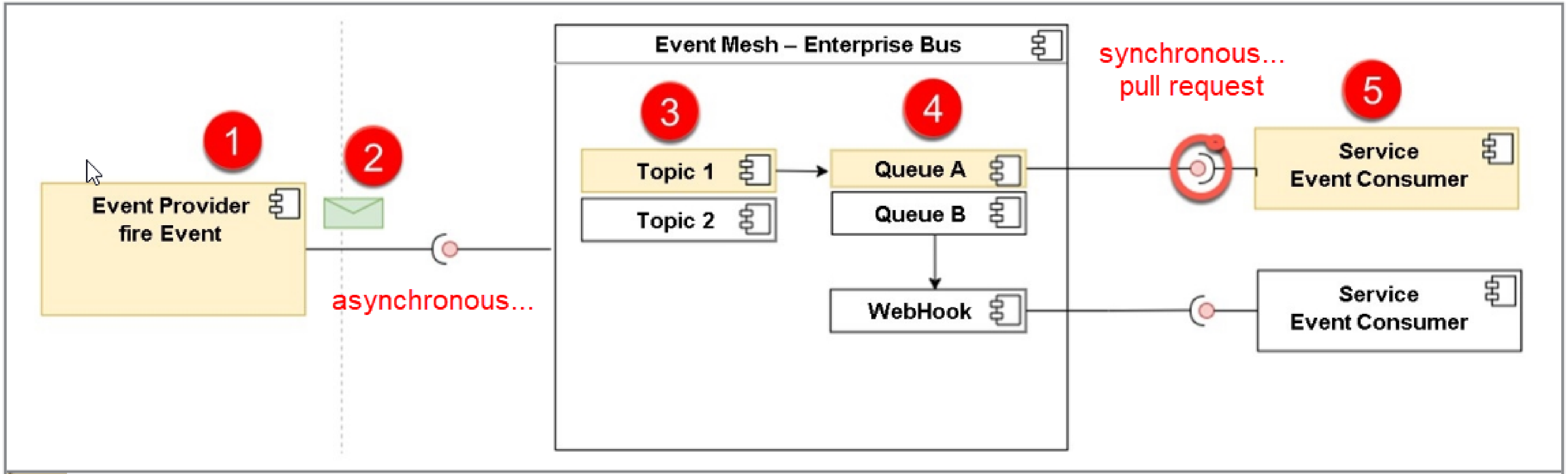
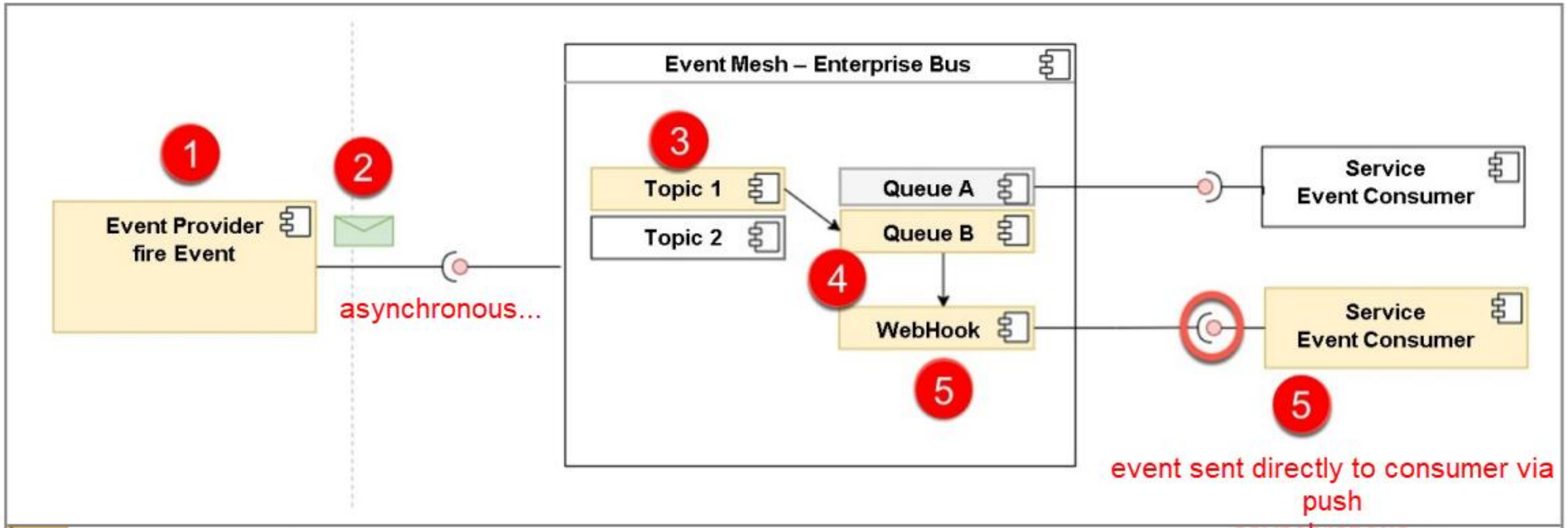


Figure 7: Pull Variant

Key Summary Points – Unit 1

Push Variant



Event Mesh – Enterprise Bus
1
2
3
Event Provider fire Event
asynchronous...
Topic 1
Topic 2
Queue A
Queue B
4
WebHook
5
Service Event Consumer
Service Event Consumer
5
event sent directly to consumer via push asynchronous...

Figure 8: Push Variant

Key Summary Points – Unit 1

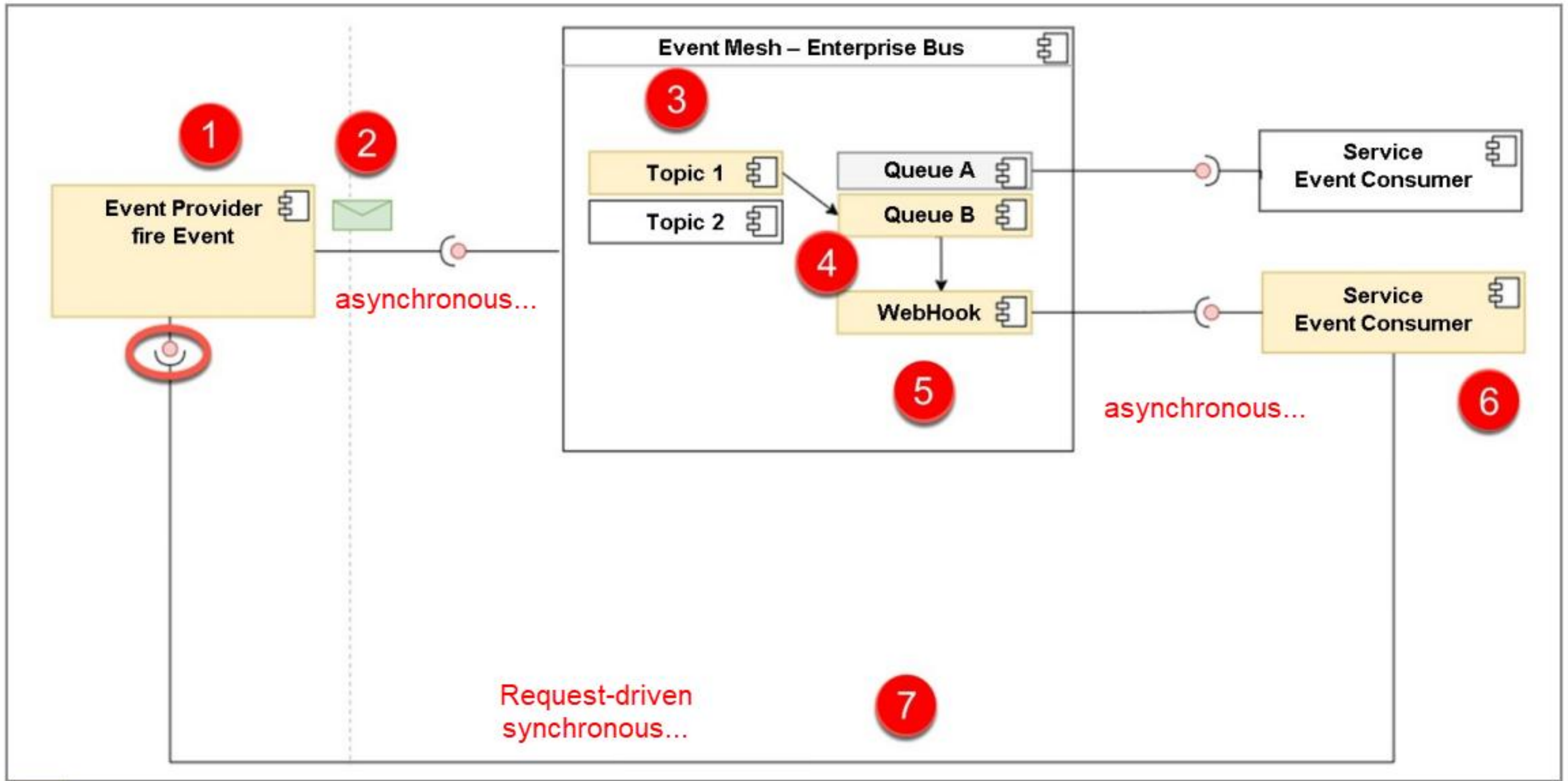
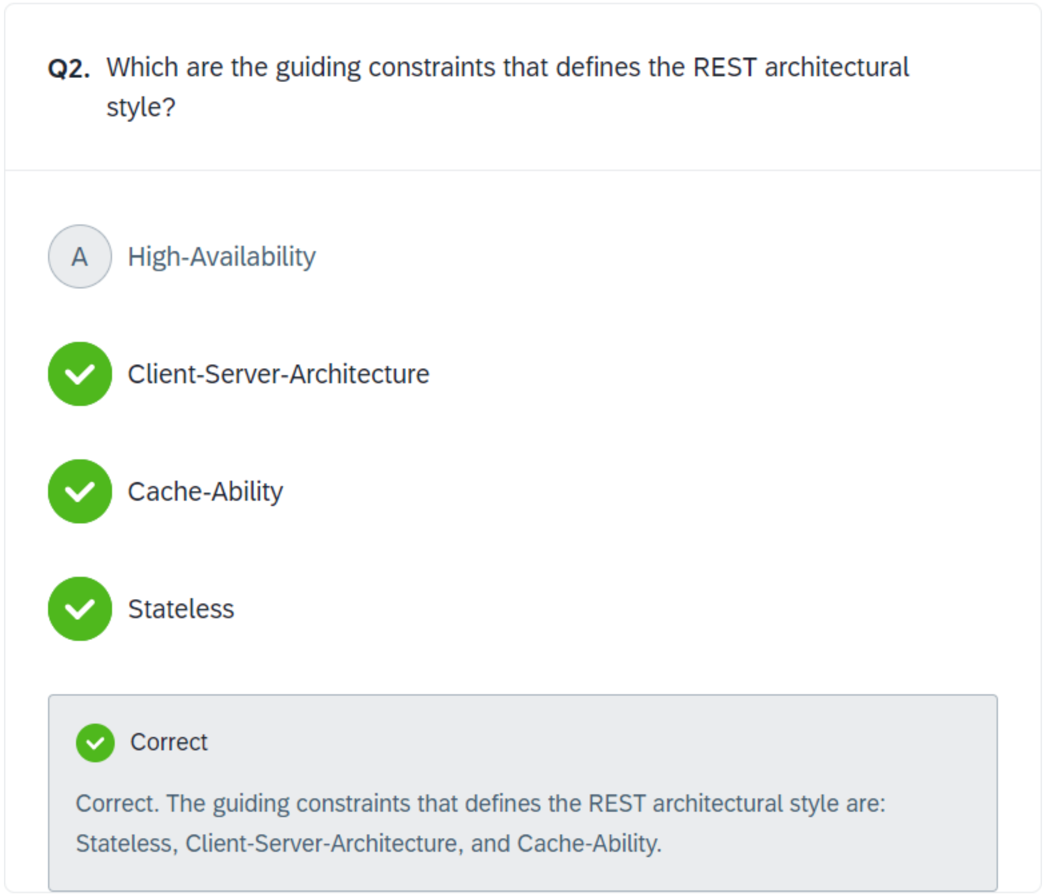


Figure 9: Combination of Request-Driven and Event-Based Architecture

Key Summary Points – Unit 1



Q2. Which are the guiding constraints that defines the REST architectural style?

A High-Availability

✓ Client-Server-Architecture

✓ Cache-Ability

✓ Stateless

✓ Correct

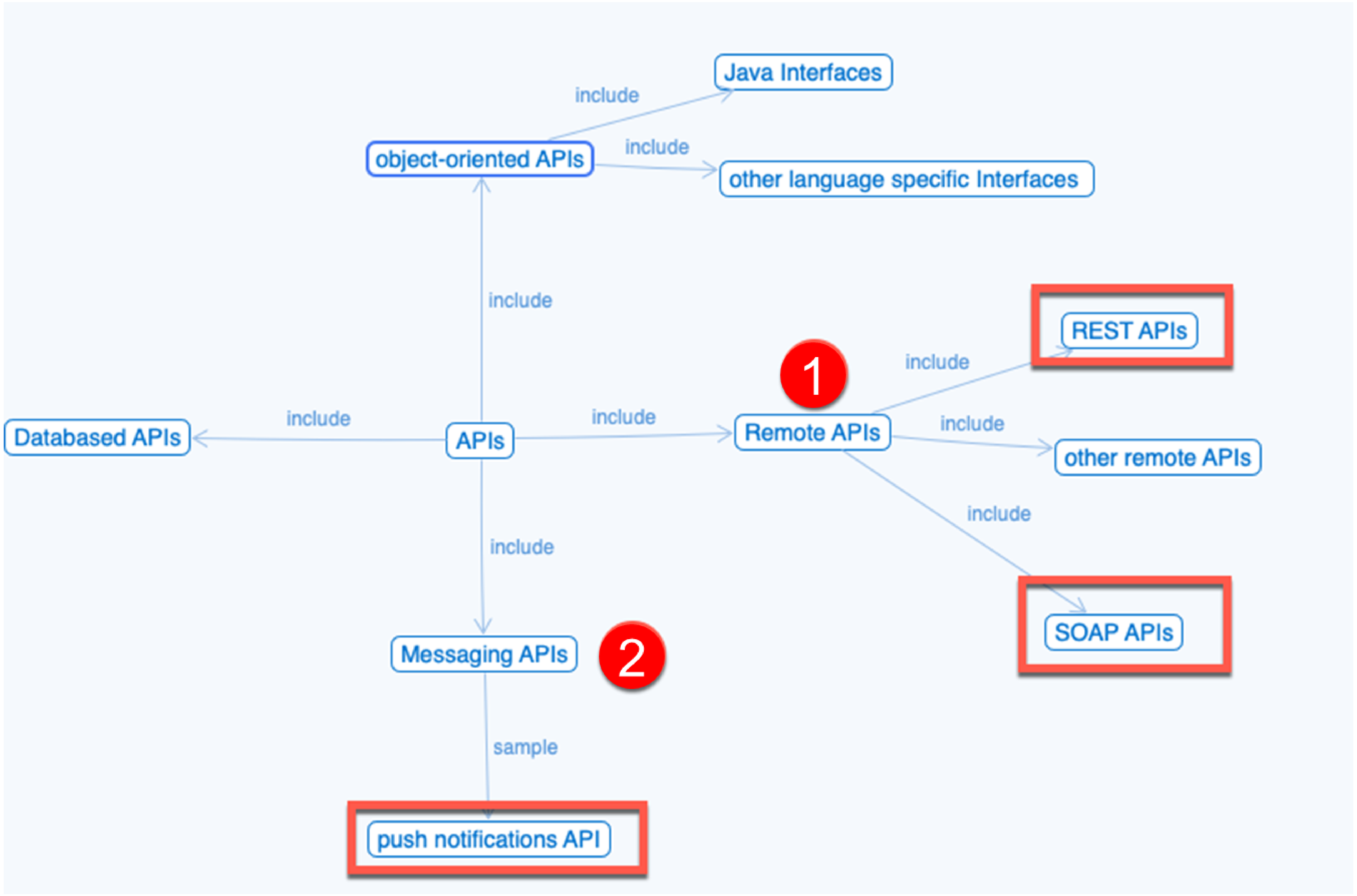
Correct. The guiding constraints that defines the REST architectural style are: Stateless, Client-Server-Architecture, and Cache-Ability.

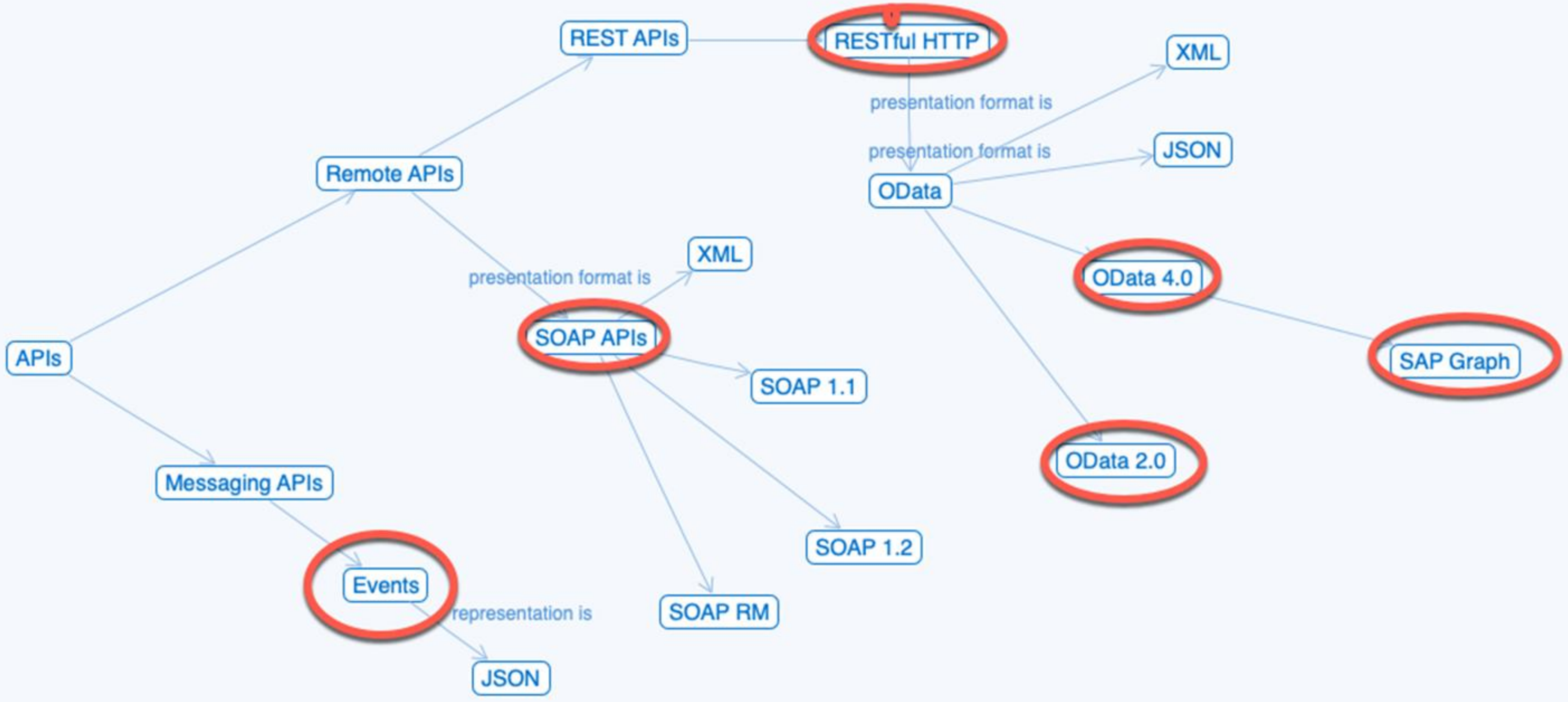
REST

- Architectural properties
  - Simplicity of uniform interface
  - Scalability, modifiability, reliability etc.
- Architectural constraints (6)
  - Client server architecture
  - Stateless
  - Cache ability
  - Layered system
  - Code on demand (optional)
  - Uniform interface
- Uses standard HTTP methods and supports many media types

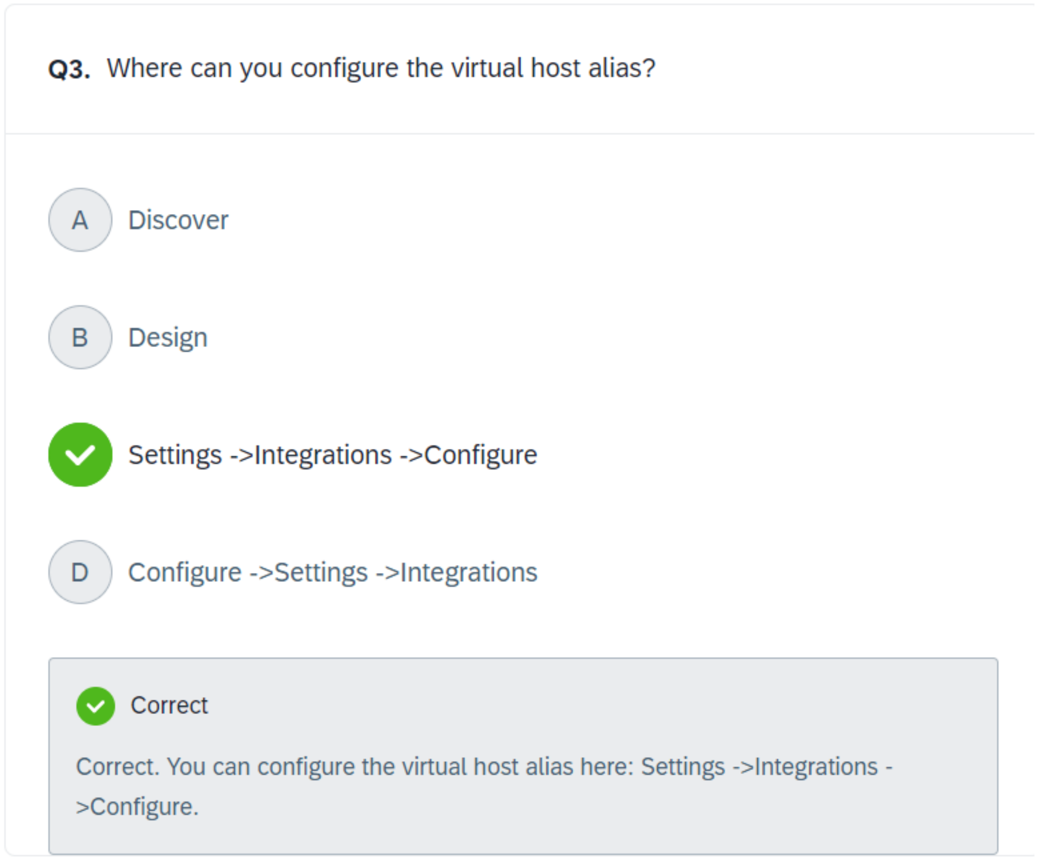
OData

- Architectural constraints
    - Resource identification
    - Fixed documents
        - Service document
            - Lists entity sets, functions, singletons
        - Metadata document
            - Describes types, sets, functions, actions
    - Dynamic resources
    - Resource operation
    - Querying
    - Resource representation


Java Interfaces
include
object-oriented APIs
include
other language specific Interfaces
include
REST APIs
1
include
Databased APIs
include
APIs
include
Remote APIs
include
other remote APIs
include
include
SOAP APIs
Messaging APIs
2
sample
push notifications API


REST APIs
RESTful HTTP
XML
presentation format is
presentation format is
JSON
Remote APIs
OData
XML
presentation format is
OData 4.0
SOAP APIs
APIs
SAP Graph
SOAP 1.1
OData 2.0
Messaging APIs
SOAP 1.2
Events
representation is
SOAP RM
JSON

Key Summary Points – Unit 1



Q3. Where can you configure the virtual host alias?

A Discover

B Design

[✓ Settings ->Integrations ->Configure]

D Configure ->Settings ->Integrations

✓ Correct

Correct. You can configure the virtual host alias here: Settings ->Integrations ->Configure.

Key Summary Points – Unit 1

- API Provider
  - Provides an interface
- API Consumer
  - Consumes the interface

SOAP, REST

Type of API	Description language
SOAP	Web Services Description Language (WSDL)
REST	Open API • Used in API management • Interface definition language for describing, producing, consuming and visualizing RESTful web services RAML

Contract between API Provider and API consumer

- Implementation first approach
  - Implementation created first by API Provider
  - Contract generated automatically which is used by API Consumer
- Contract first approach
  - Contract created first
  - Both API Provider and API Consumer can simultaneously start working against the contract