Introduction to Enterprise Integration

PV207 – Business Process Management

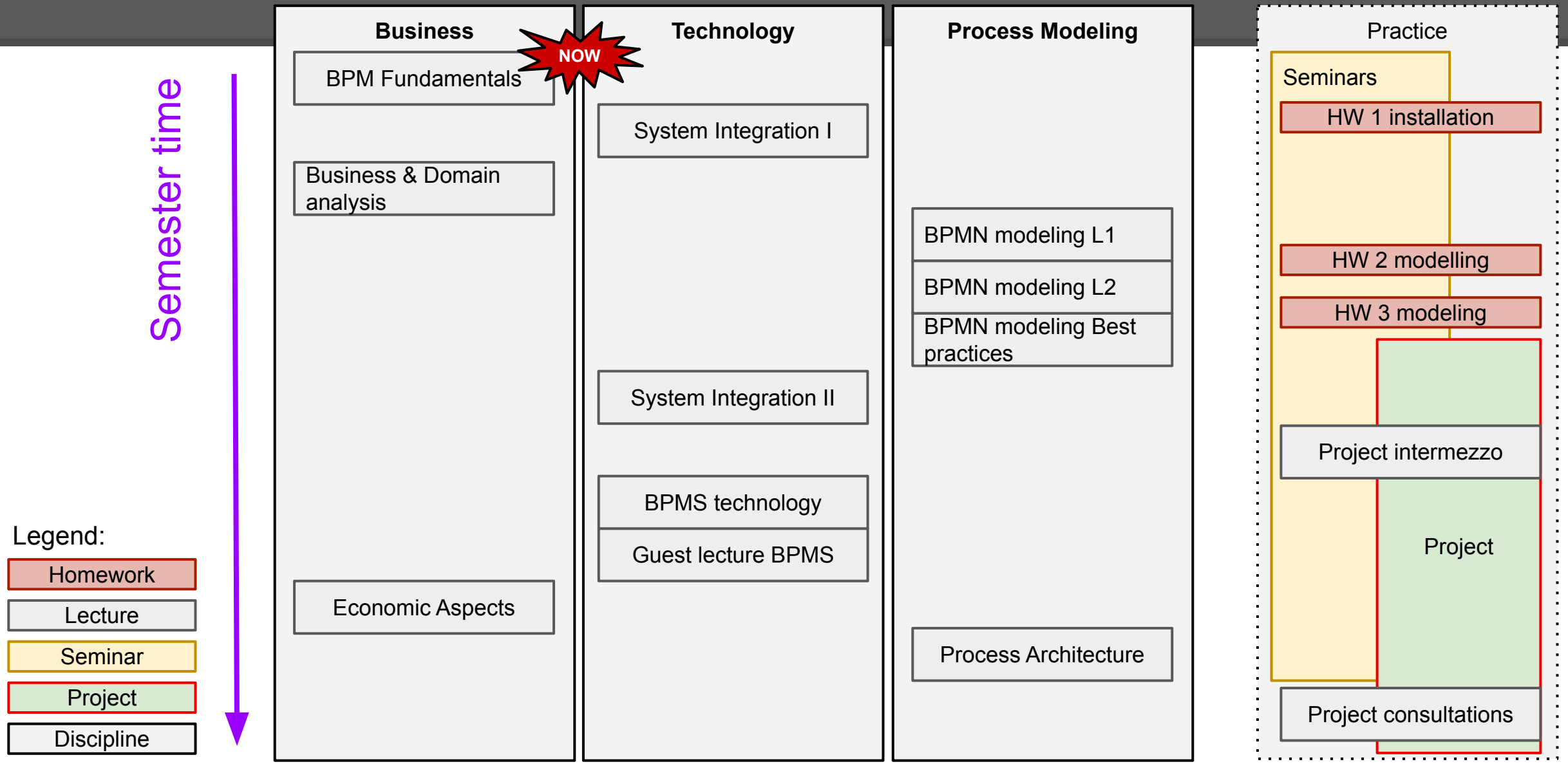
Mgr. Ivo Bek
Senior Product Manager

Red Hat

Already covered

- Business process
- Business process management
- Business process management systems

PV207 helicopter view



What we'll discuss today

- Implementing business processes connecting to enterprise systems
- Core principles of interoperability in software systems
- Handling the differences between Requester and Provider
- Message processing
- Integration patterns

Process-driven Applications

Insurance

Claims
Processing
Underwriting
Quoting
Rating
Commissioning

Banking

KYC
Loan Origination
Credit
Decisioning
Sales Advisory
Payments

Retail

Recommendation
Campaign Mgmt
Order Mgmt
Pricing
Self-service

Transportation

Workforce Mgmt
Loyalty Programs
Customer Service
Billing

Telco

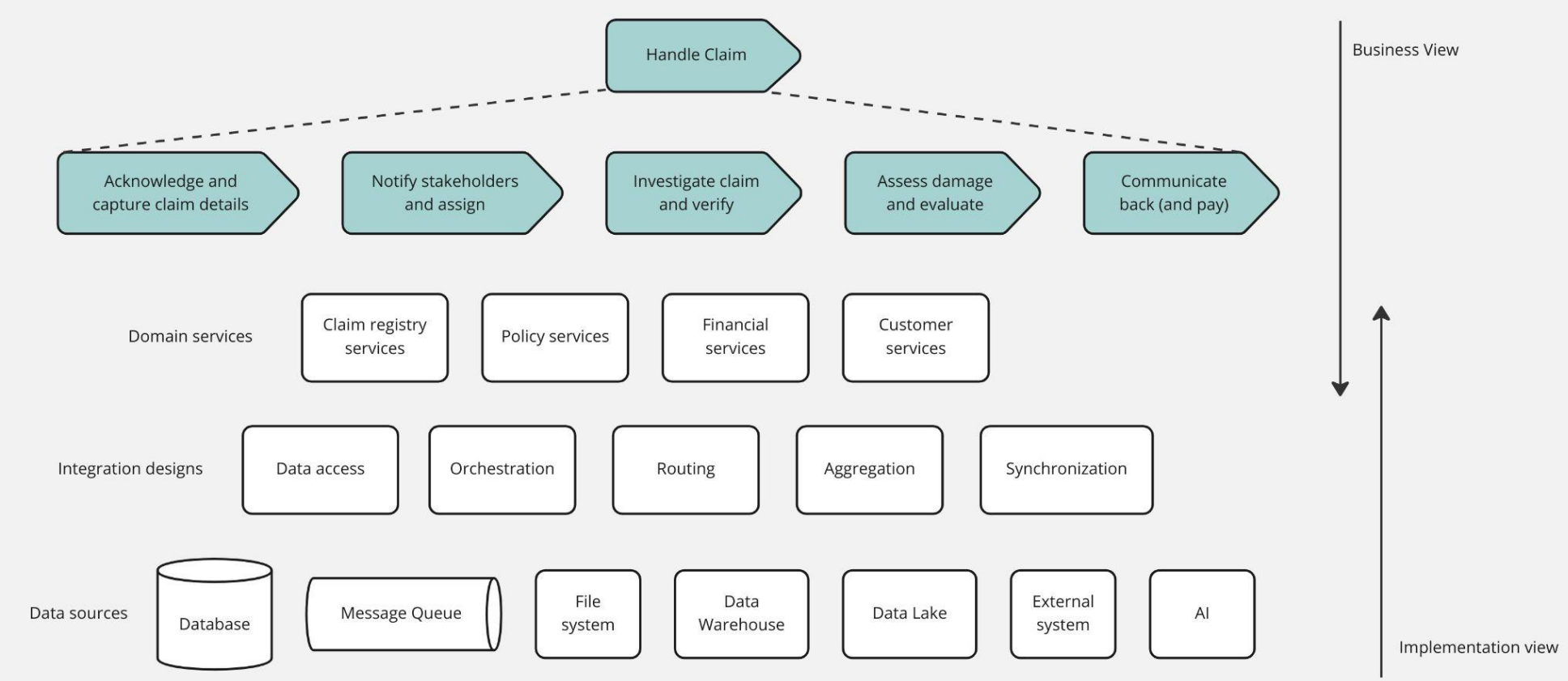
Offer
Configuration
Order Mgmt
Fraud Detection
Loyalty Programs
Network
Monitoring

Manufacturing

Order Mgmt
Billing
Contract Mgmt

Implementing business processes connecting to enterprise systems

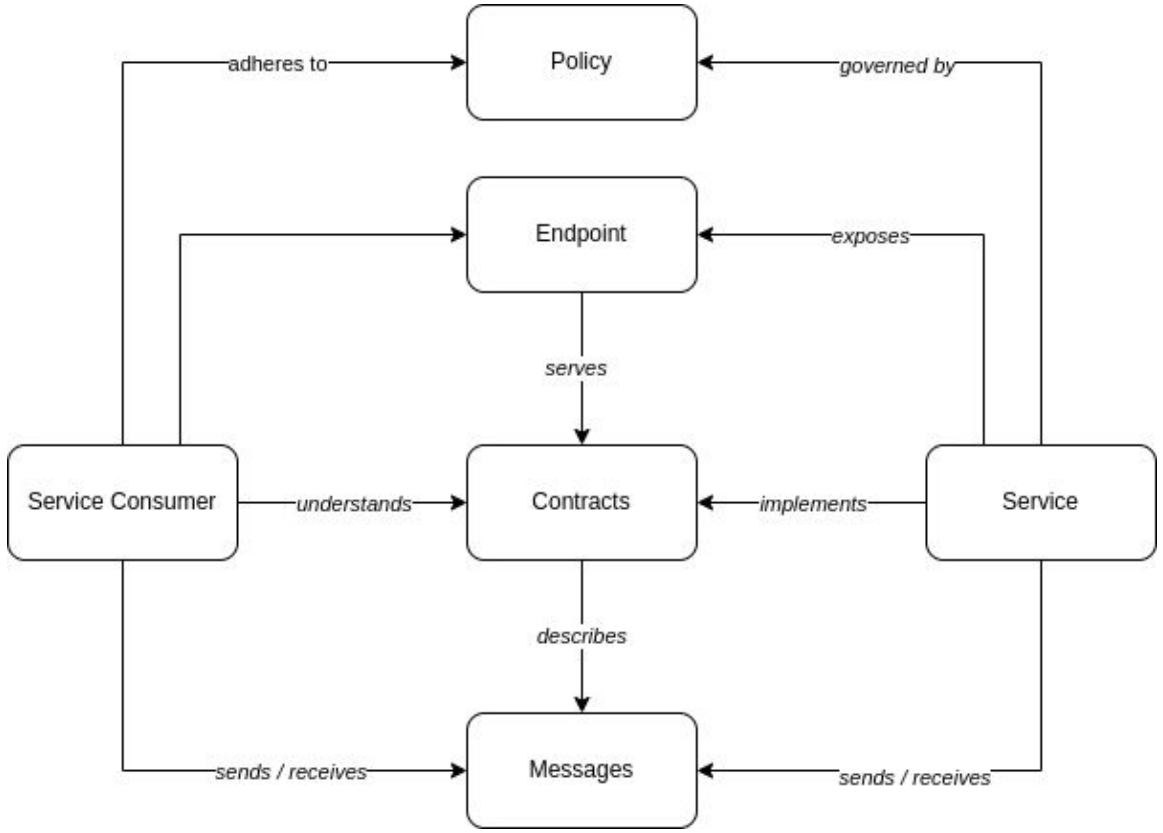
Example: Insurance claim processing



Red Hat

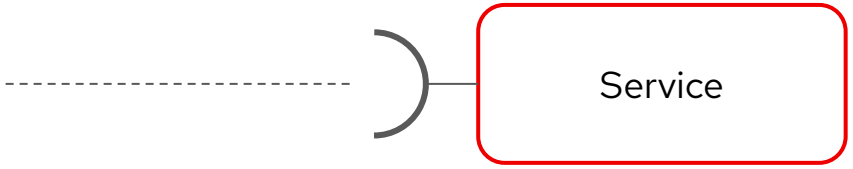
Key enterprise integration elements

to enable communication and functionality across enterprise applications and systems



Red Hat

Interface Characteristics



- Data
- Technical interface - transport, protocol, data format
- Interaction type - request/response, fire/forget, sync/async, individual/batch, publish/subscribe
- Integrity - validation, transactional, stateful, idempotence
- Security - identity, privacy
- Reliability - availability, delivery assurance
- Error handling
- Performance - response times, throughput, concurrency, message size, volumes over time

- *Implements contracts*
- *Exposes endpoint*
- *Sends / receives messages*
- *Governed by policy*

Red Hat

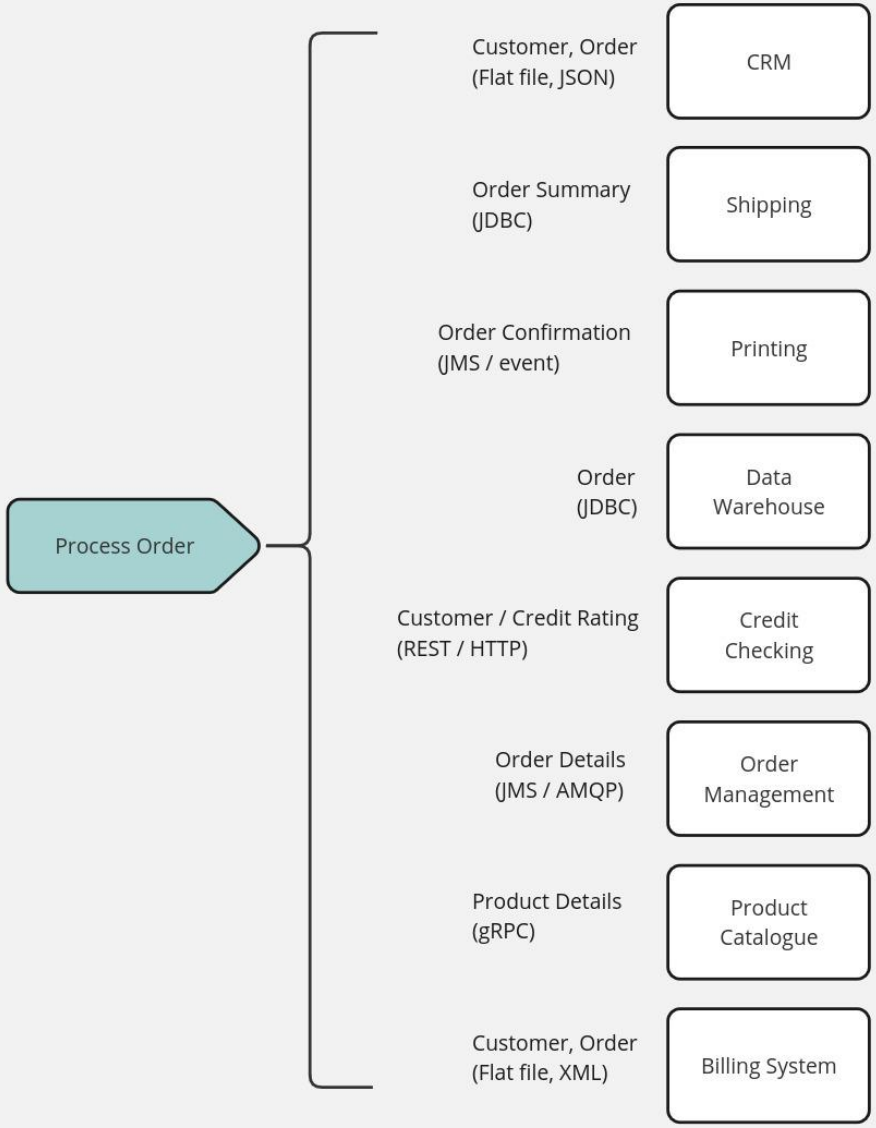
Communication and API architectural styles

Defining how services communicate over a network

- **REST and OpenAPI** – in web and mobile applications
- **SOAP Web Services** – in enterprise-level web services, financial services, payment gateways, and telco services which require comprehensive security and transactional reliability
- **gRPC** – in high-performance APIs and microservices which require low latency and real-time communication, can be used for request-response or streaming
- **GraphQL** – in data-driven applications which require complex, nested data queries
- **Message-driven** (JMS / AMQP / MQTT) – in distributed systems which require asynchronous communication
- **Streaming / Event-driven** (Kafka, Serverless) – in real-time systems which require streaming data processing and immediate reactions
- **Websockets** – in web applications requiring continuous data exchange in real-time
- **Webhooks** – in web applications which require to receive / send external events (notifications) in realtime, asynchronously

Interface Catalog

System Name	Transmission Protocol	Data Format	Data Object(s)	Interaction	...
CRM	HTTP / REST	JSON	Customer, Order	Request-response, sync	
Shipping	JDBC	SQL Text	Order Summary	Request-response, sync	
Printing	JMS	String	Order Confirmation	Fire-forget, message, async	
Product Catalogue	gRPC	Binary protobuf	Product Details	Request-response, sync	
Billing System	HTTP / SOAP	XML	Customer, Order	Request-response, sync	



Red Hat

Implementing business processes connecting to enterprise systems

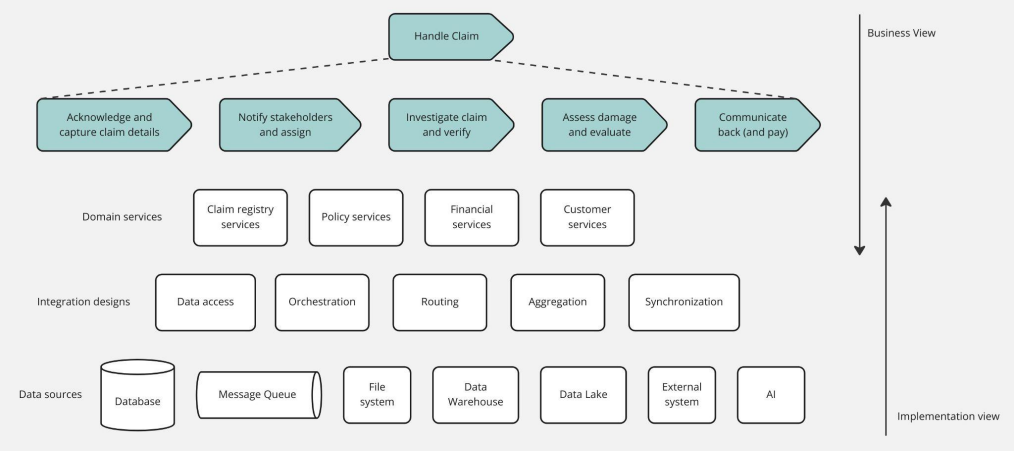
Investigation and Assessment

Business analyst

- 1. Create high level process model
- 2. Establish data objects used by process
- 5. Establish functional usage (operations) of data

Integration Specialist

- 3. Establish systems containing the data objects
- 4. Establish technical interfaces exposing the data



Team Project

Integration Task #1

As an integration specialist **build an interface catalog** listing at least 5 different system connections and their interfaces, including their transmission protocol, data format, data objects and interaction types / behavior. The system connections need to be used in business processes but they don't need to be implemented. The chosen service interfaces should be realistic to what technology you would select today (e.g. a preference for REST API + JSON) or how they were implemented historically.

Deliverables

- Interface catalog in project documentation

Red Hat

Core principles of interoperability in software systems

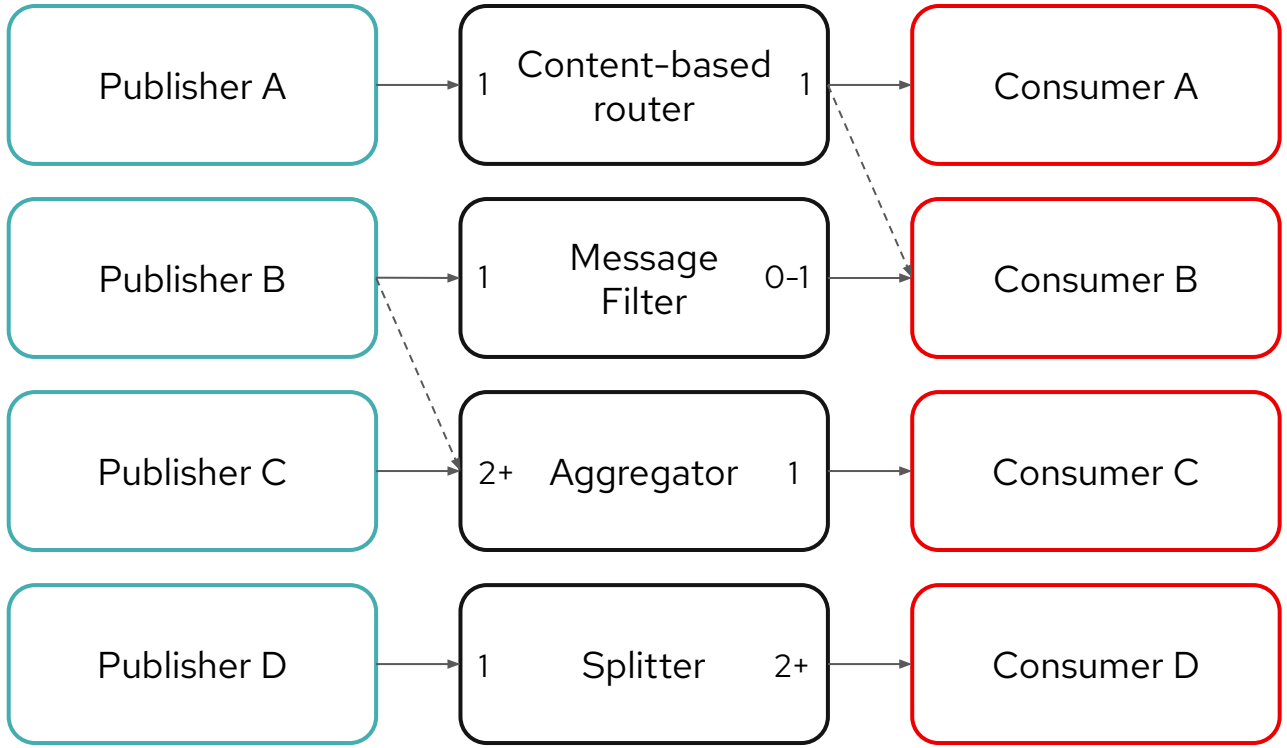
Publish / subscribe messaging	Message routing	Event streaming
Data integration	API management	Workflow orchestration

Red Hat

Direct messages between services based on predefined rules

Message routing

- Dynamic path selection - decouples producers and consumers
- **Content-based routing** – receive a message and examine the content of the message to determine its destination
  - **Message filter** - passes the message to another channel if the message content matches certain criteria, otherwise discards the message
- **Aggregator** – receive a stream of messages, identify related messages and combine them into a single message
- **Splitter** – receive a single message, break it down into multiple messages, each containing a subset of the data from the original message
  - **Recipient list** - determine a list of recipients based on criteria within the message, and send copies of the message to each determined recipient



Facilitate flow of data between systems

Data integration

Data Migration

Integrate disparate systems, transition to cloud, consolidate data centers or upgrade systems.

Data Processing

Improve data quality, remove unnecessary data, combine data from multiple sources, convert data from one form or structure to another, and enrich data.

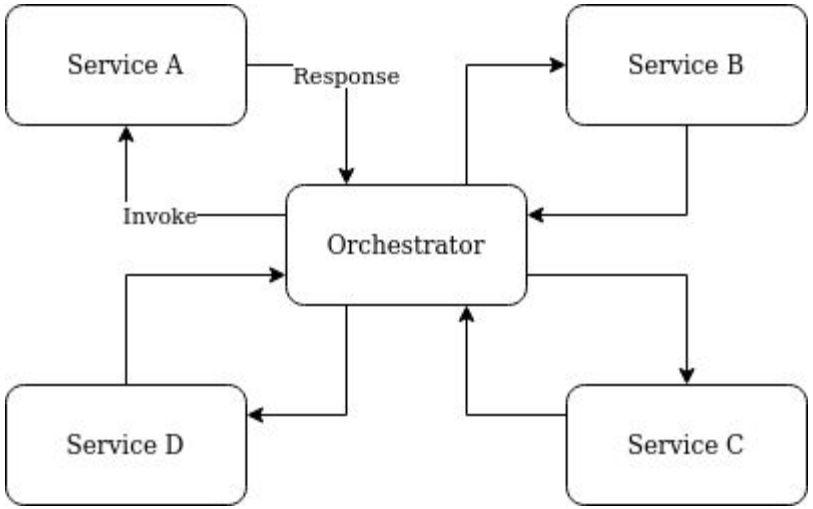
Data Synchronization

Facilitate seamless operations across disparate systems, maintain data integrity, replicate database and apply change data capture.

Coordinate and automate flow of tasks

Workflow orchestration

- **Coordinate** interactions between **microservices**
- Orchestrate **data pipelines** using workflows
- Features
  - Parallel execution
  - Branching
  - Timeouts
  - Callbacks
  - Compensation
  - Error handling
- **Operational control** over workflows to troubleshoot with the ability to pause, resume, restart, retry, debug, and terminate
- **Monitoring and tracking** progress of workflows, identifying bottlenecks and failures for continuous improvement



Red Hat

Questions?

Break 10mins

Handling the differences between Requester and Provider

Integration

- What is the primary objective of the integration?
- What do we need to do with the message?
- When do events occur?
- Are there specific performance targets or SLAs?
- Does the requester need to ensure (eventual) consistency?
- Will the requester produce duplicate requests?
- Should the requester get access to all data?

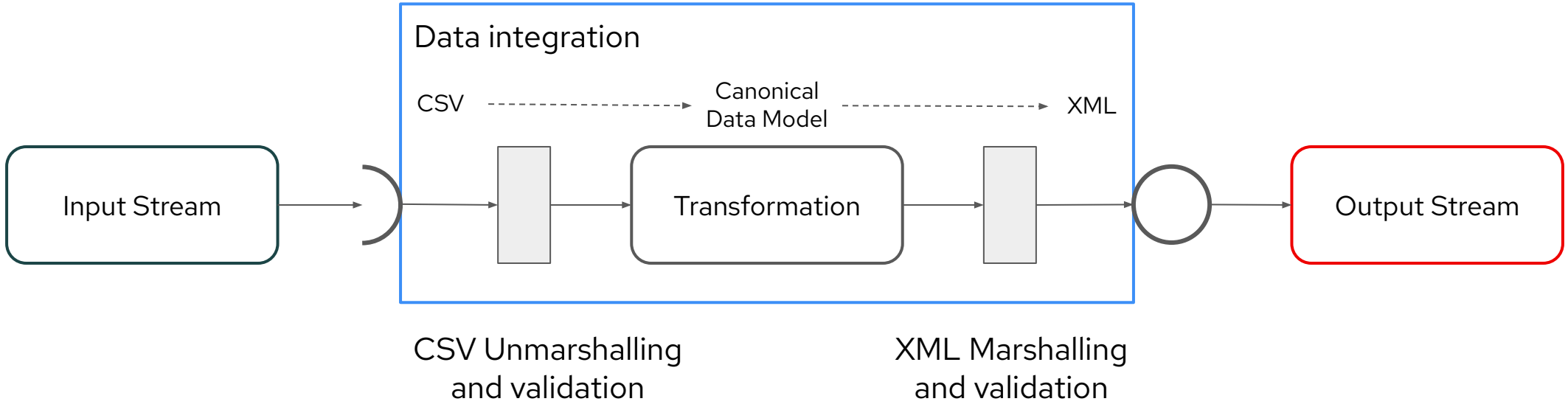
- What are the interface specifications?
- How are authentication and authorization handled in both systems?
- What are the error handling and retry mechanisms in case of failed requests or responses?
- Are there any rate limiting or quota restrictions imposed by the service provider?

Architecture and Design Patterns
Enterprise Integration Patterns

Red Hat

Data integration from CSV to XML

Format transformation



Unmarshalling transforms data in some binary or textual format (such as received over the network) into a Java object; or some other representation according to the data format being used.

Marshalling transforms the message body (such as Java object) into a binary or textual format, ready to be wired over the network

Exchange and translate data

Message processing

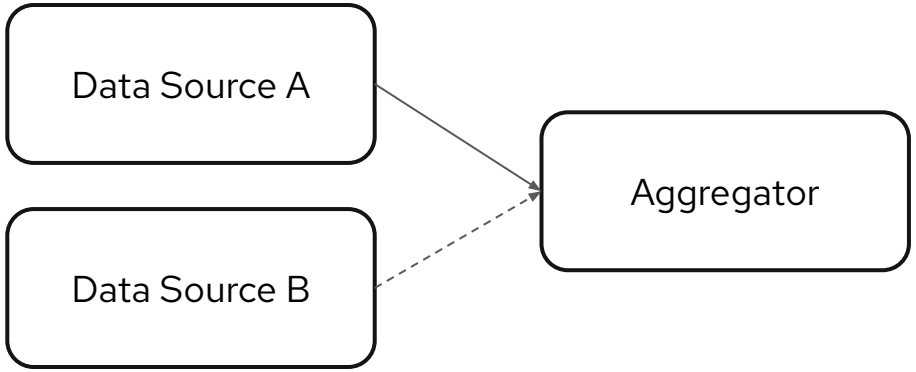
- Data Formats and Structures - XML, JSON, Zipfile, Avro, Protobuf, SOAP
- Data Transformation - XSLT, JSLT, XJ
- Expression Languages - JQ, JSONPath, XPath, XQuery
- Data Mapping - Bindy, Mapstruct, JAXB, Gson, Xstream
- Healthcare Data - HL7, FHIR
- Data Encryption - PGP, JCE, SSL/TLS

Data Validation	Data Model Translation	Data Integrity
Data Enrichment	Data Filtering	Data Privacy

Combine data from multiple sources into a unified format

Data aggregation

- **Aggregator** is a stateful filter that collects and stores source messages until it can fully combine all source data for the aggregated message
- **Message correlation** determines which messages belong together
- Aggregator **strategies** – wait for all, time out, delta processing, number of messages, completion value, ..
- Examples – insert data to DB in batches



Bridge different API architectural styles

REST-SOAP mediator

OpenAPI contract

WSDL contract

```
openapi: 3.0.0
info:
  title: WeatherService
  description: Provides weather information
  version: "1.0.0"
servers:
  - url: 'http://example.com/weatherservice'
paths:
  /weather:
    get:
      summary: Get weather information
      description: Returns weather information for a
given city
      parameters:
        - in: query
          name: city
          required: true
          schema:
            type: string
          description: The name of the city
      responses:
        '200':
          description: Successful response
          content:
            application/json:
              schema:
                type: object
                properties:
                  temperature:
                    type: number
                    description: The current temperature
                  description:
                    type: string
                    description: A brief description of
the weather conditions
```

JSON

Data format transformation

XML-SOAP

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
                  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  targetNamespace="http://example.com/weatherservice"
                  xmlns:tns="http://example.com/weatherservice">

  <wsdl:types>
    <xsd:schema targetNamespace="http://example.com/weatherservice">
      <xsd:element name="GetWeatherRequest">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="City" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="GetWeatherResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="Temperature" type="xsd:float"/>
            <xsd:element name="Description" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>

  <wsdl:message name="GetWeatherInput">
    <wsdl:part name="parameters" element="tns:GetWeatherRequest"/>
  </wsdl:message>

  <wsdl:message name="GetWeatherOutput">
    <wsdl:part name="parameters" element="tns:GetWeatherResponse"/>
  </wsdl:message>

  <wsdl:portType name="WeatherServicePortType">
    <wsdl:operation name="GetWeather">
      <wsdl:input message="tns:GetWeatherInput"/>
      <wsdl:output message="tns:GetWeatherOutput"/>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="WeatherServiceBinding" type="tns:WeatherServicePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="GetWeather">
      <soap:operation soapAction="http://example.com/GetWeather"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="WeatherService">
    <wsdl:port name="WeatherServicePort" binding="tns:WeatherServiceBinding">
      <soap:address location="http://example.com/weatherservice"/>
    </wsdl:port>
  </wsdl:service>

</wsdl:definitions>
```


Red Hat

Typical structure of a data processing flow

VETRO pattern

Check incoming data for correctness and completeness to prevent errors and inconsistencies downstream.

- Data format
- Data type
- Value ranges

Incorporate related data from other sources to make incoming data more complete and useful

- Enricher
- Aggregation

Convert data from one format or structure to another to align data with the target system interface

- Format transformation - JSON <=> XML
- Data transformation - 1:1, N:1, merging, complex manipulations

Direct processed data to the appropriate destination(s) based on predefined criteria

- Dynamic path selection
- Content based routing

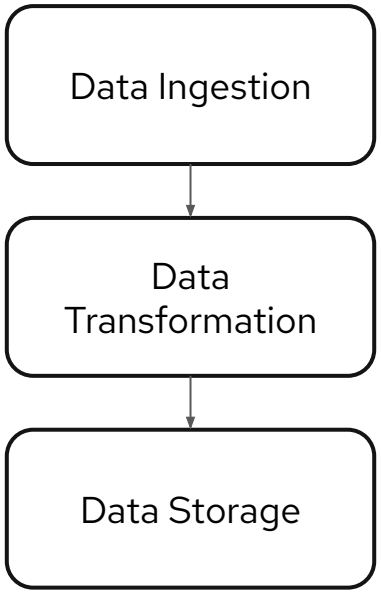
Execute tangible actions supporting business operations based on the processed data

- Trigger workflow
- Update database
- Generate alert

Move data - collect, transform, and store

Data pipeline

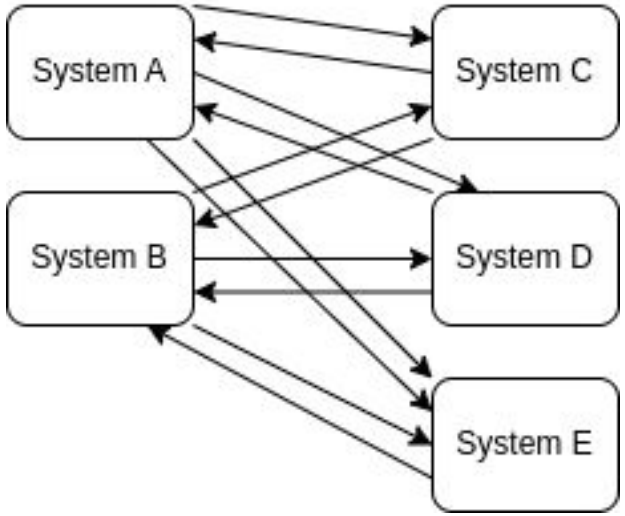
- Data pipeline is a workflow consisting of one or more tasks that **ingest, move, and transform** raw data from one or more data sources to a **data storage**
- Common in **data science / machine-learning** projects and **business intelligence dashboards** (e.g. monthly accounting)
- **Change data capture** pipeline - Database changes (create, update, delete) externalized as events
- **Extract-transform-load (ETL)** pipeline extracts data from the source system, transform it into the desired format, and load it into the target system.
  - Extract-load-transform (ELT) pipelines are popular in the cloud-native solutions - transformations can scale horizontally, handling varying volumes of data more efficiently than traditional ETL pipelines



Red Hat

Facilitate flow of data between systems

Point-to-point integration

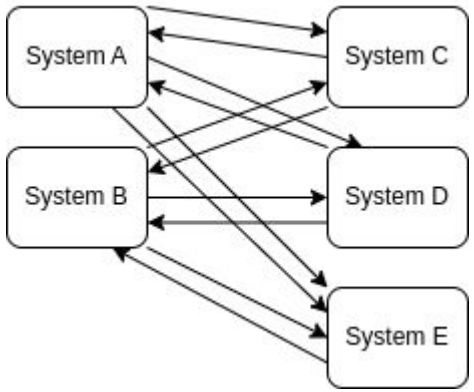


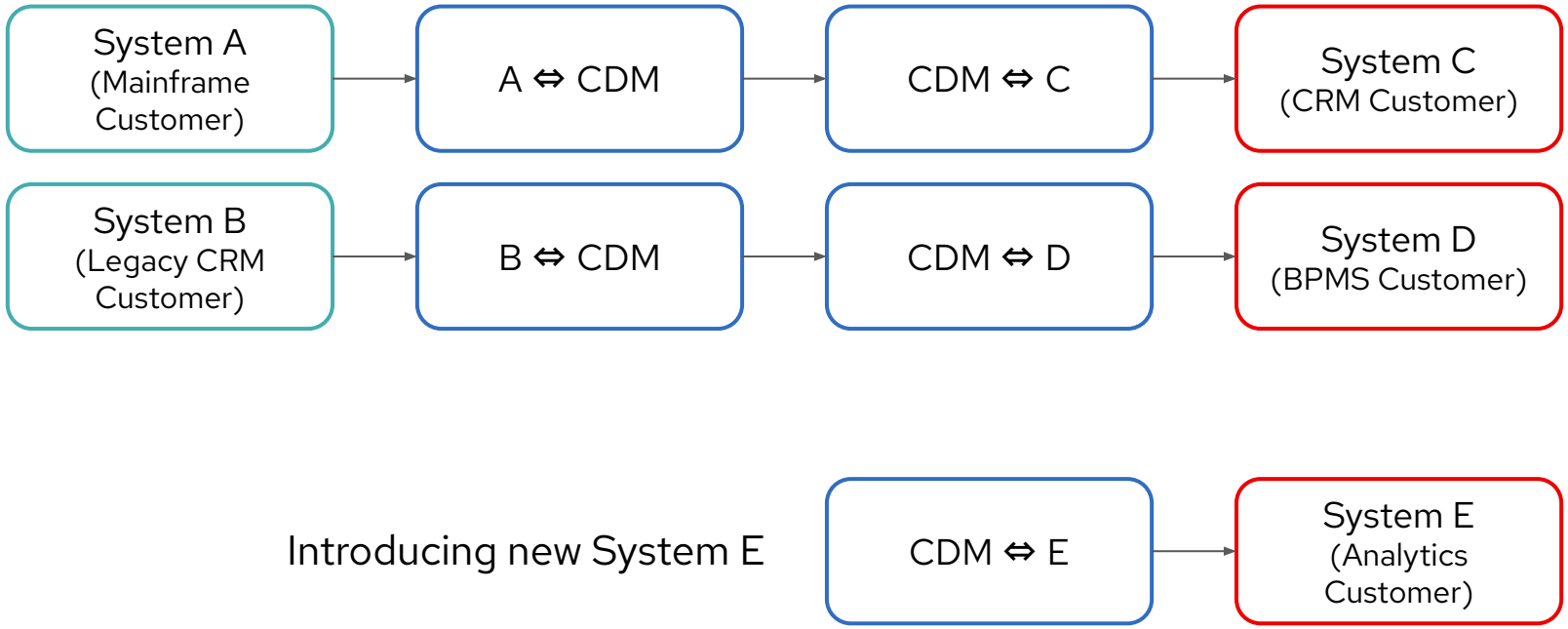
Red Hat

Add more endpoints without direct data format dependency

Canonical Data Model pattern

Without CDM





Red Hat

Efficiently connect disparate systems

using integration patterns

Message Bridge
Routing
Publish-subscribe
Message transformation
Exactly once delivery
Event sequencing
Batch processing
Dead letter queue
Correlation
Idempotence resolution
Event sourcing

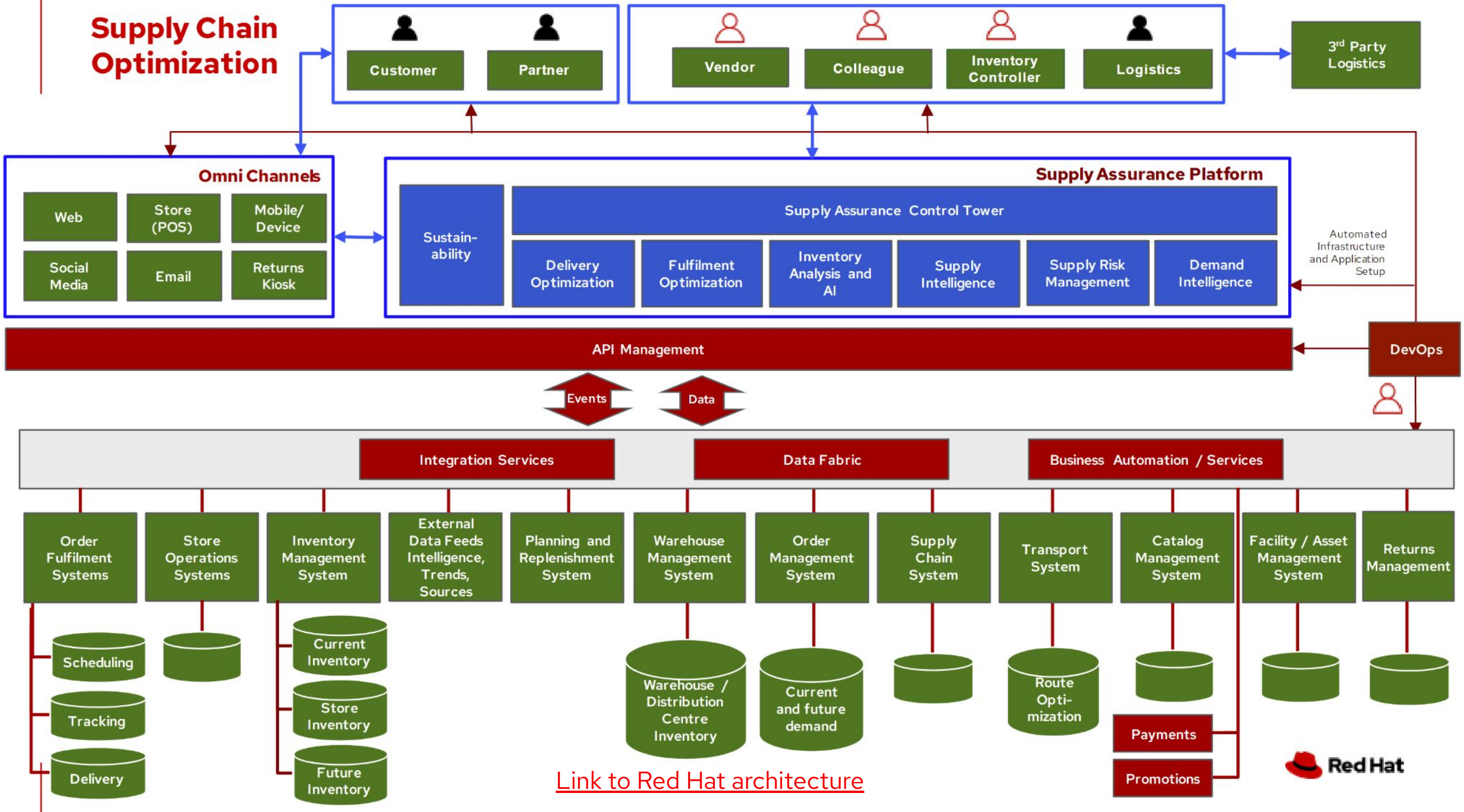
Data Pipeline
Data migration
Data enrichment
Data synchronization / Strangler
Aggregation
Claim check
Correlation
Retry
Normalizer

Service Mediator
Routing
Composition
Orchestration
Parallel pipeline
Error handling
Correlation
Retry
Throttling
Interface translation
Health check
Load leveling
Saga

Enterprise Integration Patterns → Solution implementation

Red Hat

Supply Chain Optimization



Link to Red Hat architecture

Labs today

- BPMS Hands-on implementing the first business process

Red Hat

Next lesson

- Business and domain analysis

Red Hat

Resources

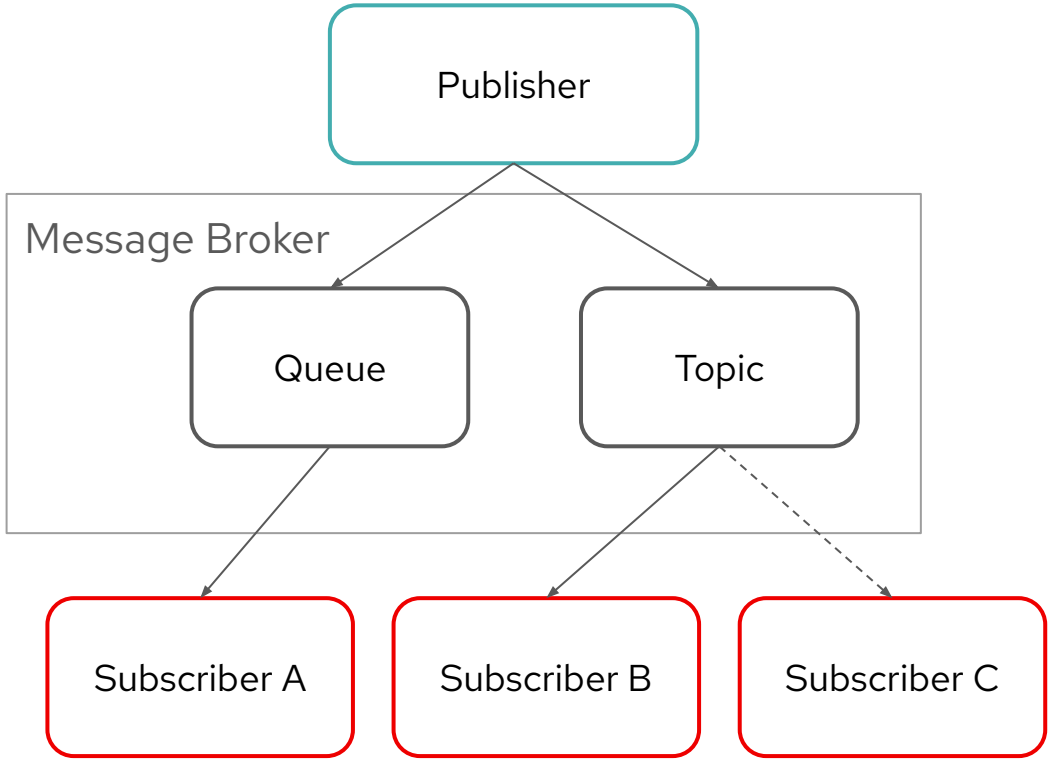
- Interface characteristics by Kim Clark and Brian Petrini
- Enteprise integration patterns by Gregor Hohpe and Bobby Woolf
- Red Hat Developer

Extra

Enable decoupled communication in distributed systems

Publish / subscribe messaging

- Allow publishers to send messages without knowing the subscribers, who receive messages based on their subscribed interests
- Message types
  - Volatile - not stored, sent only to online consumers, best performance with lowest possible latency
  - Durable - stored until read by all registered consumers
  - Replayable - stored and published for a specific time or until storage capacity is reached
- Messaging systems
  - Multi-protocol message brokers for transactional messaging (volatile and durable) - guaranteed message delivery, message ordering, atomic operations, message redelivery, push-pull
  - Event streaming platforms with focus on storing and processing streams of records efficiently

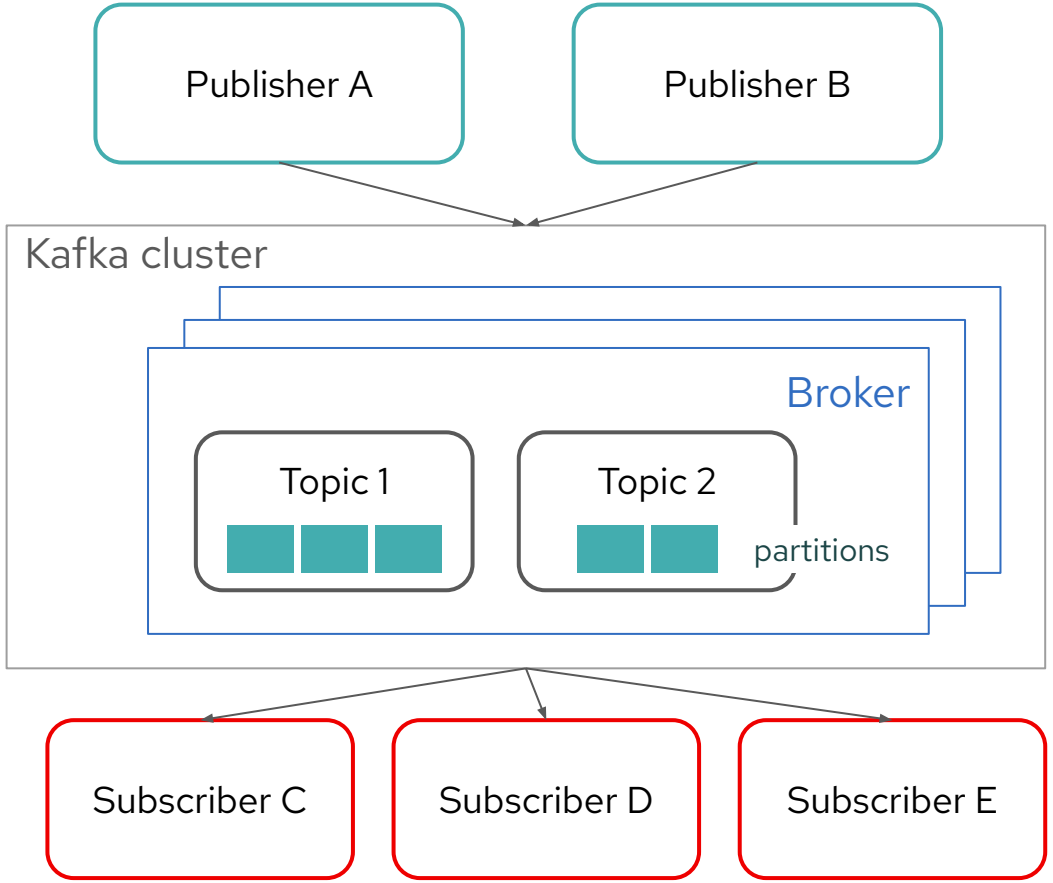


Red Hat

Transfer and process real-time data streams

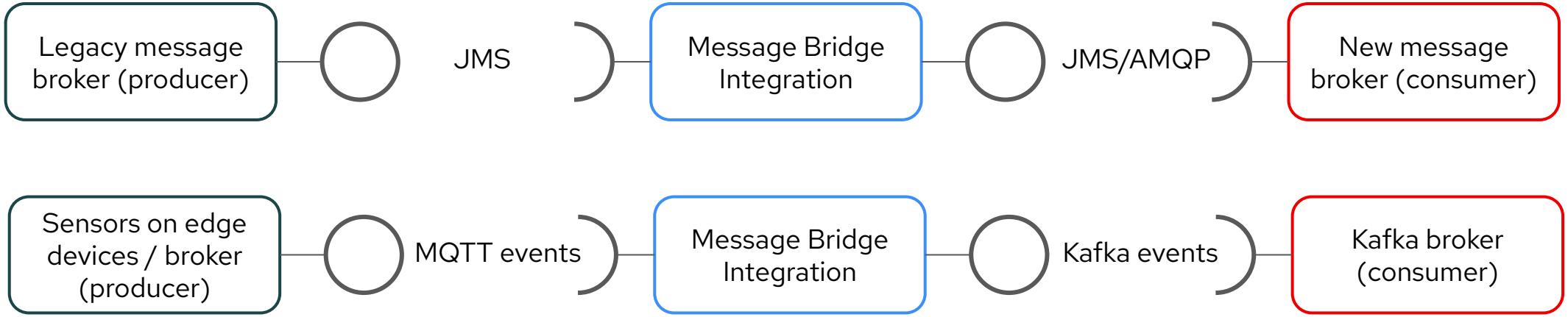
Event streaming

- Ideal for scenarios requiring high throughput and efficient handling of large volumes of data for real-time analytics, log aggregation, stream processing, and building data pipelines
- Replayable events
- Performance is less affected by the size of the data it stores, thanks to its design
- At-least-once delivery by default - need for idempotent consumer



Connect messaging systems

Message bridge

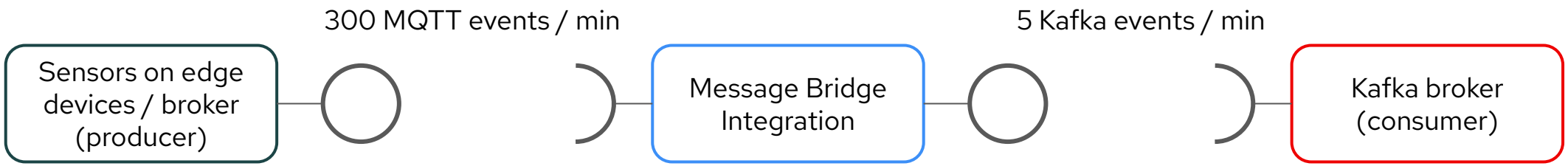


- Protocol translation
- Message transformation
- Routing
- Enrichment and filtering
- Key considerations
  - Performance
  - Compliance

Red Hat

Process large data sets periodically

Batch processing



- 300 sensors at a smart farm monitoring various conditions
- 1 sensor sends 1 MQTT message / min
- 4 types of readings
  - Temperature
  - Humidity
  - Soil moisture
  - Nutrient levels

- Batch processing interval every 5 minutes
- Aggregate data
- Average readings
- Identify outliers / thresholds

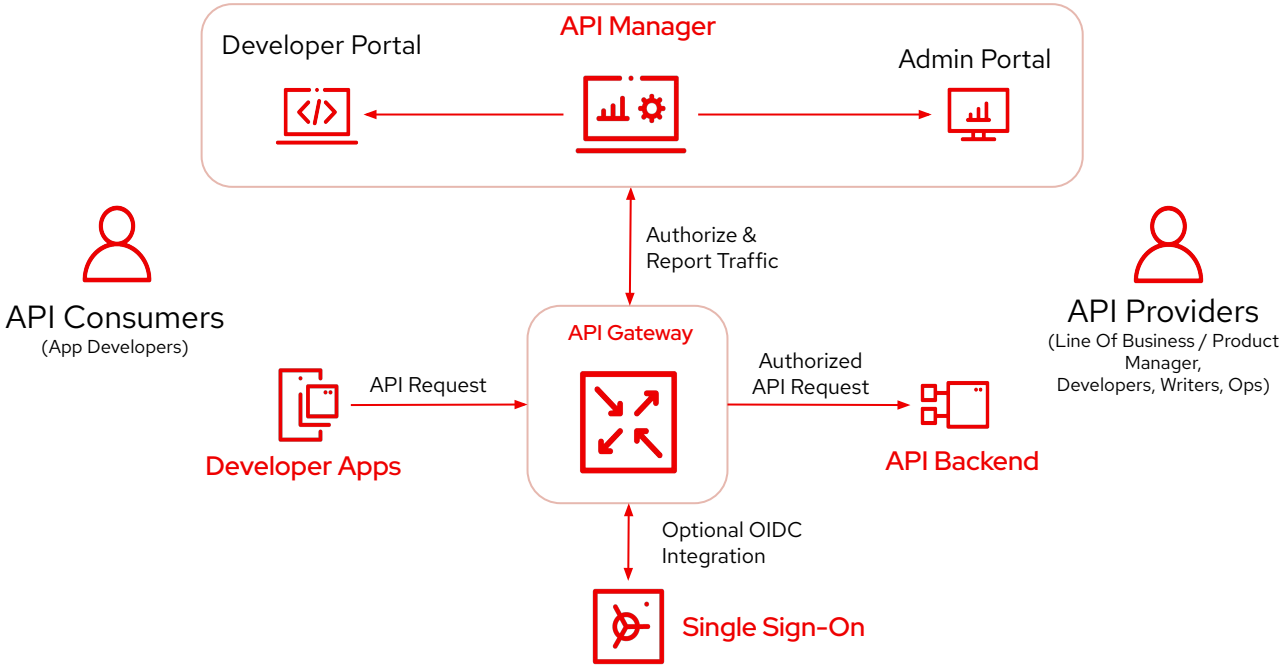
- 4 events for aggregated readings (one per type)
- 1 alert event

Red Hat

Control API interactions to maintain system integrity

API Management

- **API Gateway** serves as a central point of control and entry for API calls, routing requests to the appropriate services
- API **traffic control**
  - authentication
  - policy enforcement
- Application & user **access control**
  - access tiers
  - throttling and rate limits
- **API contracts**
  - package APIs
  - store and validate **data model schemas and API contracts in schema registry**
- Measure the success of APIs using analytics
- Apply pricing rules and automatic invoicing

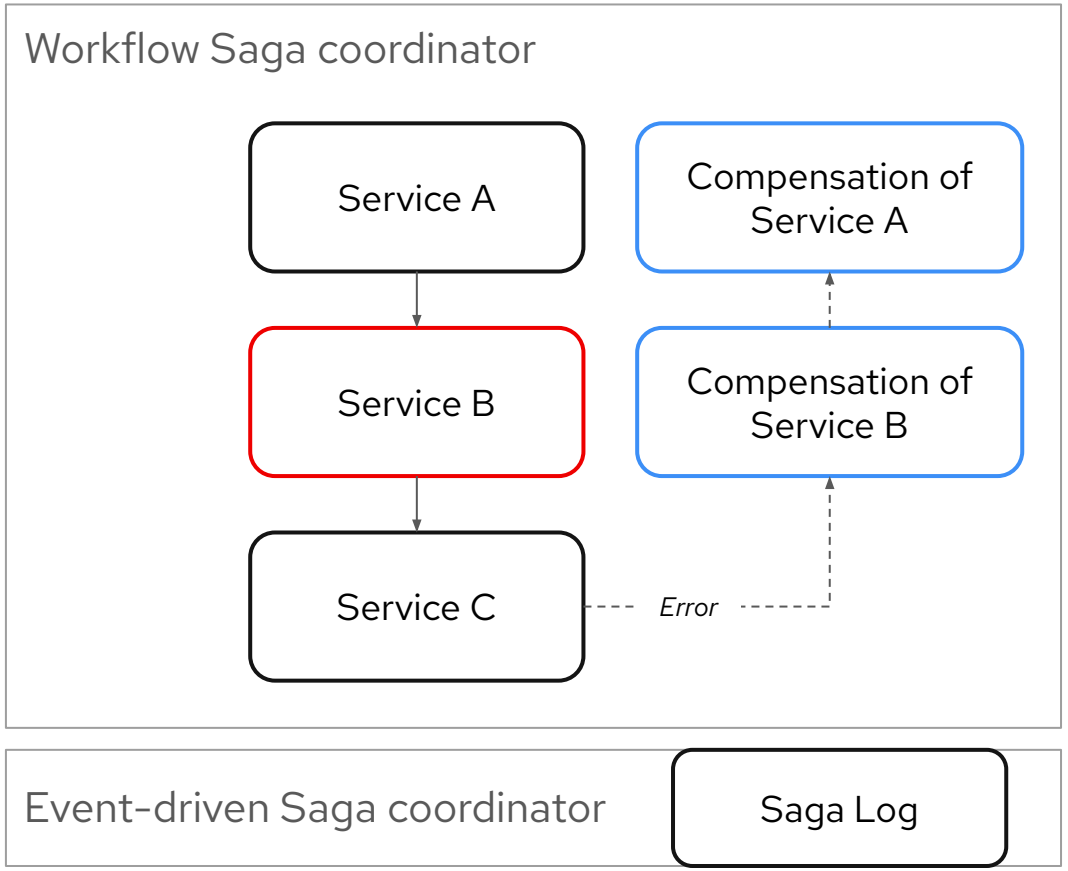


Red Hat

Maintain data consistency and integrity when a service fails

Compensation

- Ensure a consistent outcome across multiple, independent providers
- Compensation is a mechanism used to **revert an operation**
- **Saga** - coordinates multiple compensations performed in reverse order when a multi-step transaction fails at any point during execution
  - As part of workflow orchestration
  - In event-driven Saga coordinator



Red Hat

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

facebook.com/redhatinc

youtube.com/OpenShift

twitter.com/OpenShift

Red Hat