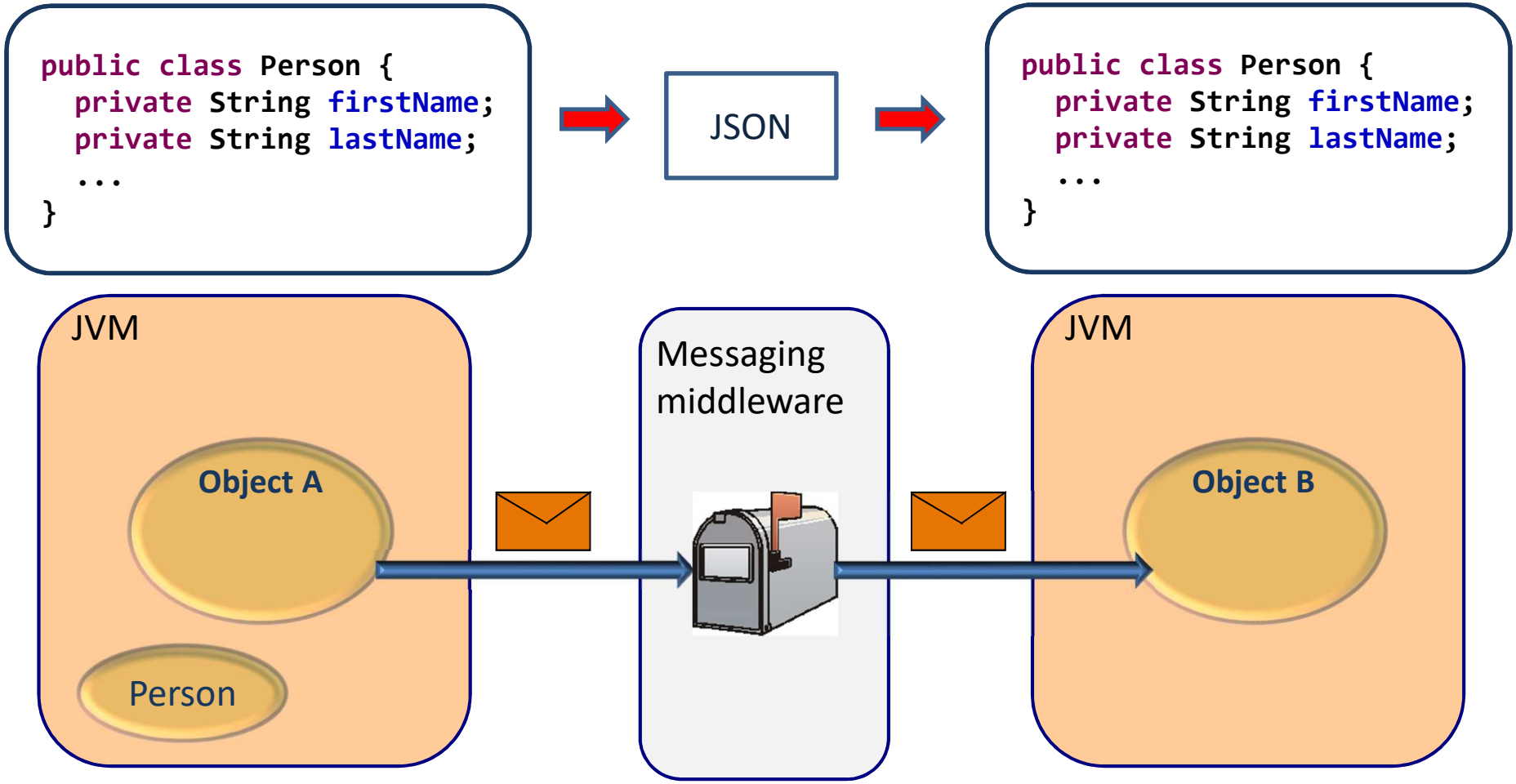
JMS

Java Message Service

Sending an object



in the message we can have text or serialize object

Spring ActiveMQ libraries

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-activemq</artifactId>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
</dependency>
```

Spring JMS sender

```
@Component
public class JmsSender {
  @Autowired
  JmsTemplate jmsTemplate;

  public void sendJMSMessage(Person person) {
    System.out.println("Sending a JMS message.");
    jmsTemplate.convertAndSend("testQueue",person);
  }
}
```

Name of the queue

application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin
```

Spring JMS sender

```
@SpringBootApplication
@EnableJms
public class SpringJmsSenderApplication implements CommandLineRunner{
  @Autowired
  JmsSender jmsSender;

  @Bean
  public JmsTemplate jmsTemplate(final ConnectionFactory connectionFactory) {
    final JmsTemplate jmsTemplate = new JmsTemplate(connectionFactory);
    jmsTemplate.setMessageConverter(jacksonJmsMessageConverter());
    return jmsTemplate;
  }

  @Bean // Serialize message content to json
  public MessageConverter jacksonJmsMessageConverter() {
    MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();
    converter.setTargetType(MessageType.TEXT);
    converter.setTypeIdPropertyName("_type");
    return converter;
  }
```

Spring JMS sender

```
  public static void main(String[] args) {
    SpringApplication.run(SpringJmsSenderApplication.class, args);
  }

  @Override
  public void run(String... args) throws Exception {
    jmsSender.sendJMSMessage(new Person("Frank","Brown"));
    jmsSender.sendJMSMessage(new Person("Mary","Smith"));
  }
}
```

Spring JMS receiver

```
@Component
public class PersonMessageListener {

    @JmsListener(destination = "testQueue")
    public void receiveMessage(final Person person) {
      System.out.println("JMS receiver received message:" + person.getFirstName()+"
       "+person.getLastName());
    }
}
```

Name of the queue

sender and receiver must have the same attributes

Spring JMS receiver

```
@SpringBootApplication
@EnableJms
public class SpringJmsReceiverApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringJmsReceiverApplication.class, args);
  }

  @Bean
  public MessageConverter jacksonJmsMessageConverter() {
    MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();
    converter.setTargetType(MessageType.TEXT);
    converter.setTypeIdPropertyName("_type");
    return converter;
  }

}
```

rest is async
Jms is Async => you have queue

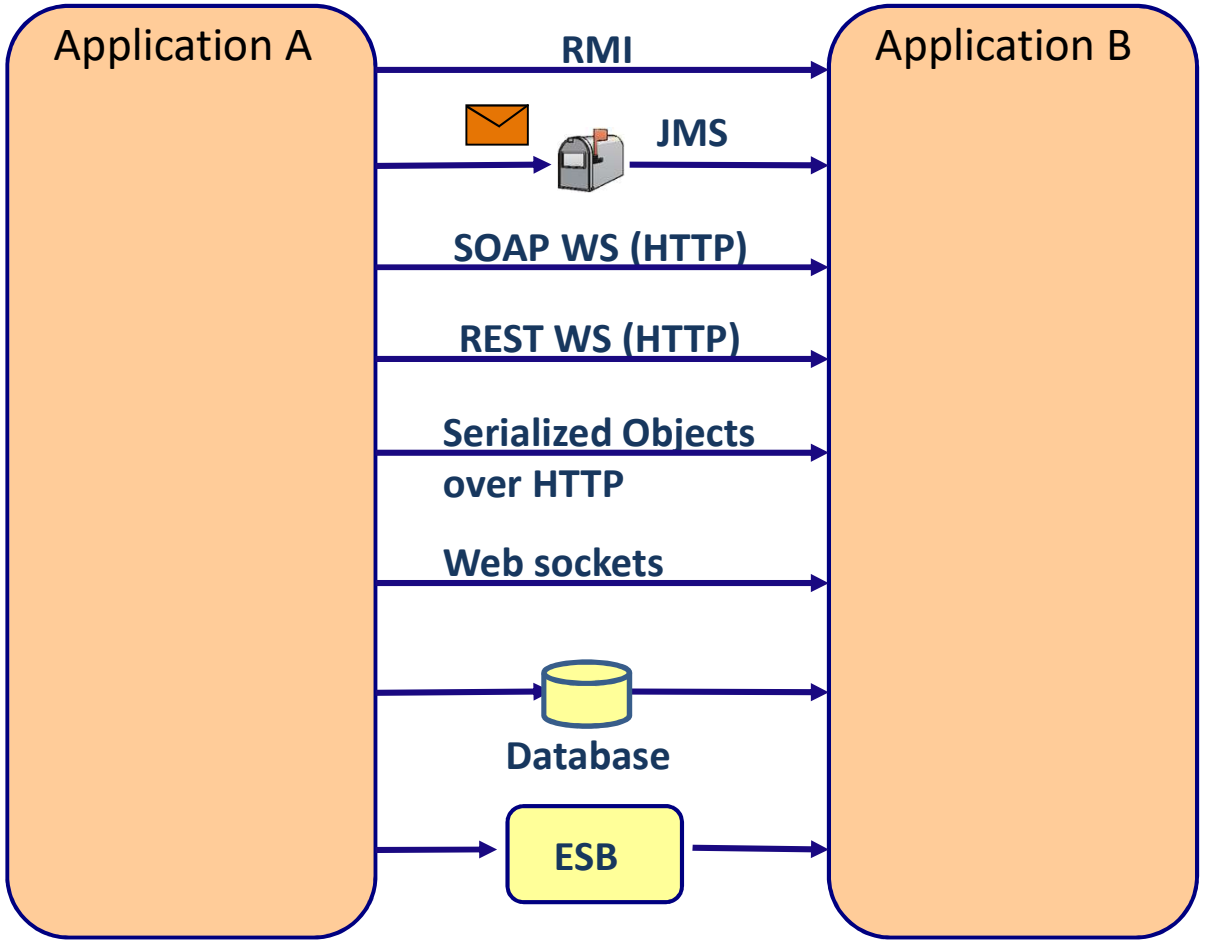
application.properties

```
spring.activemq.broker-url=tcp://localhost:61616
spring.activemq.user=admin
spring.activemq.password=admin
```

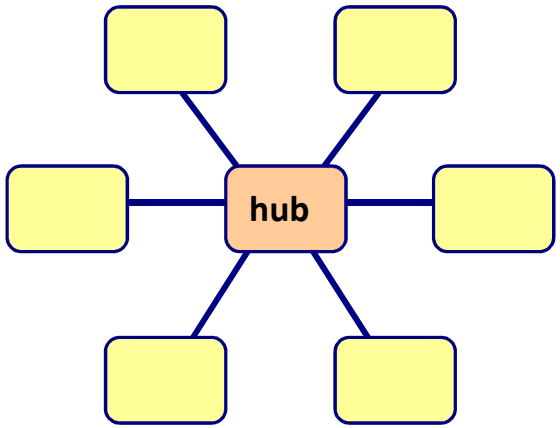
INTEGRATION PATTERNS

Integration possibilities

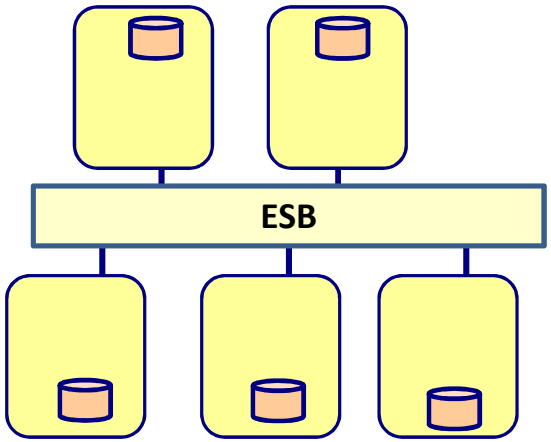
all of these are point to point connections



Architecture styles



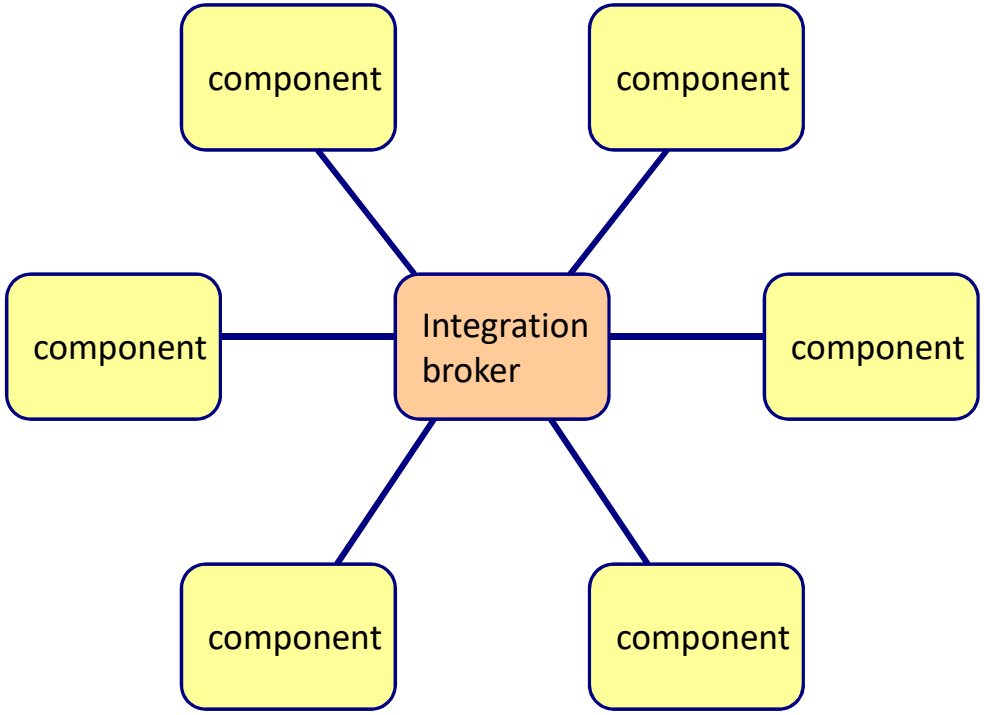
Hub and Spoke



Service oriented

Hub and Spoke

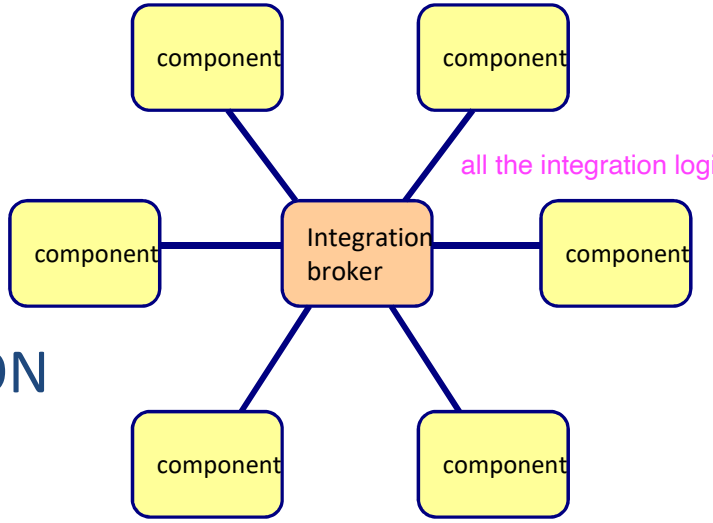
- Integration broker



Hub and Spoke

connect all together

- Functionality:
  - Transport
  - Transformation
    - For example from XML to JSON
  - Routing
    - Send the message to a component based on certain criteria (content based routing, load balancing, etc.)
  - Orchestration
    - The business process runs within the integration broker



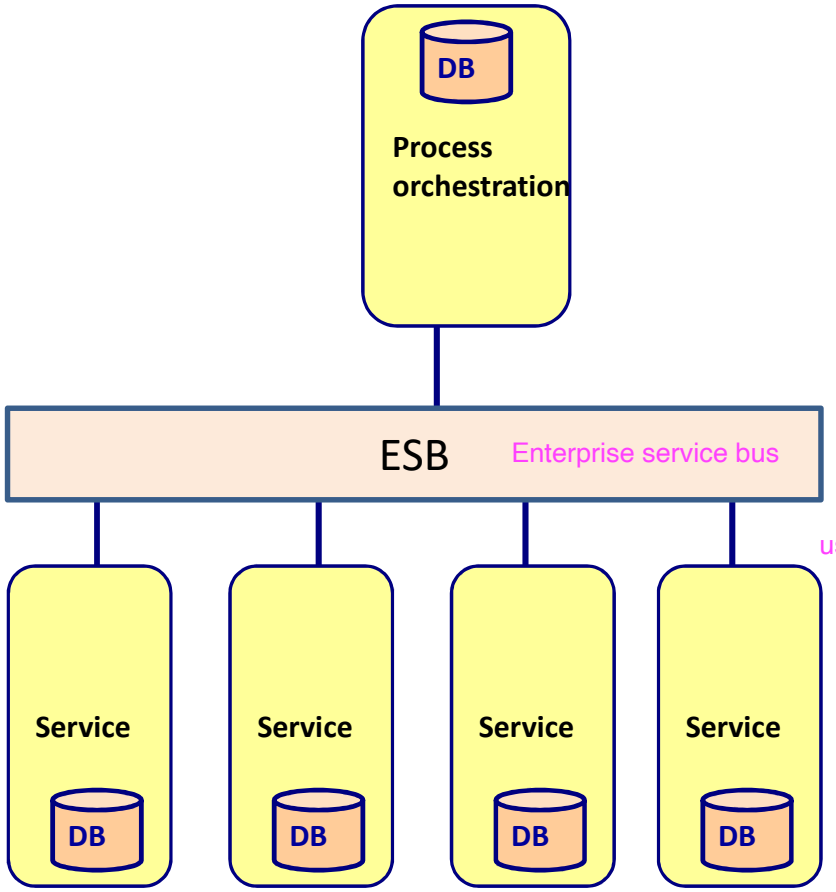
all the integration logic is in the broker

Hub and spoke

- Benefits
  - Separation of integration logic and application logic
  - Easy to add new components
  - Use adapters to plugin the integration broker
- Drawbacks
  - Single point of failure
  - Integration brokers are complex products
  - Integration broker becomes legacy itself leghacy means use, not old

Service Oriented Architecture

accidental architecture has 3 problems
1. stove pipes.-
2.- redundant data and the logic
3.- business processes



use BPEL (BUSINESS PROCESS EXECUTION LANGUAGE)

3 different aspects of a SOA

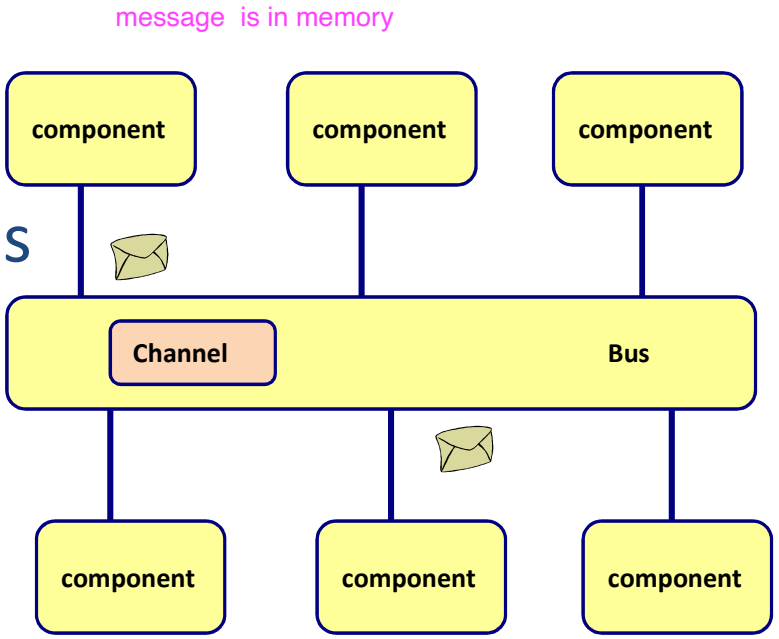
1. Communication through ESB
   - Standard protocols
2. Decompose the business domain in services
   - Often logical services
3. Make the business processes a 1st class citizen
   - Separate the business process from the application logic

Benefits of SOA

- Independent services
- Separation of business processes and service logic
- Architecture is optimized for the business
- Reuse of services
- Architecture flexibility

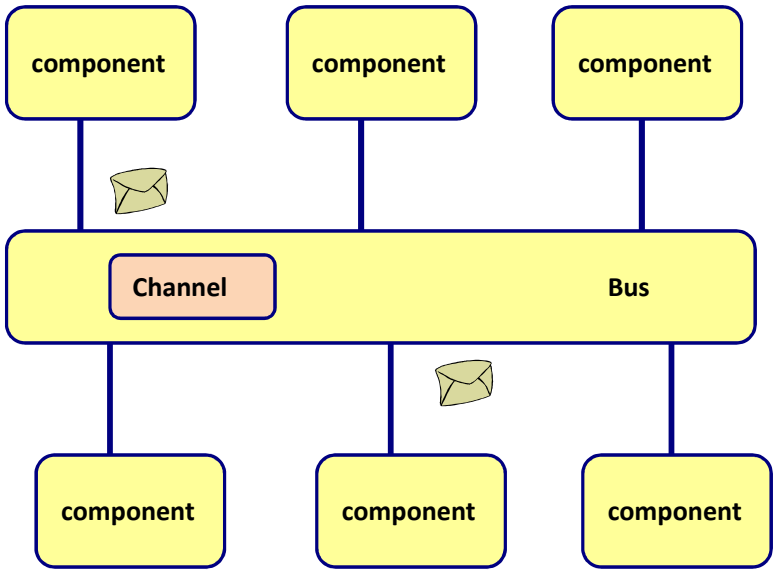
Event bus

- Publish-subscribe
  - Subscribers subscribe to particular channels
  - Publishers publish messages to particular channel
  - Subscribers receive messages from channels



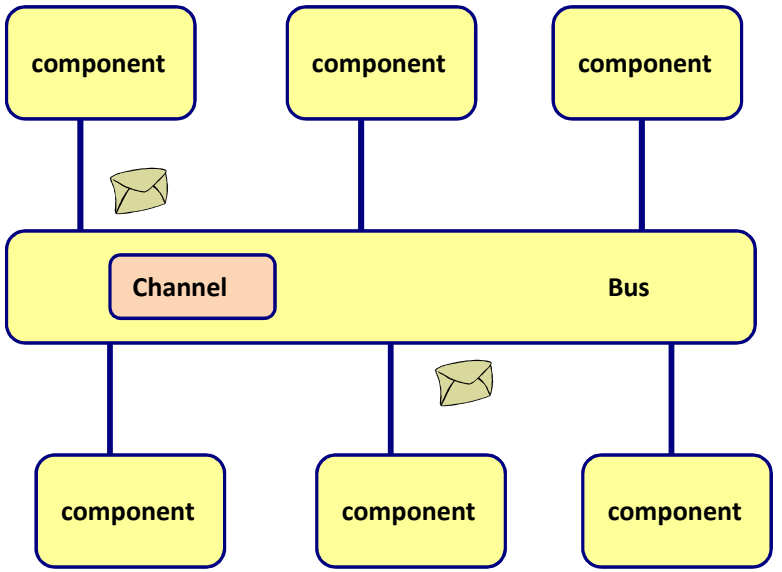
Event bus characteristics

- Asynchronous
- Loose coupling
- Easy to add new components
- Separation of integration logic and application logic



Responsibility of the bus

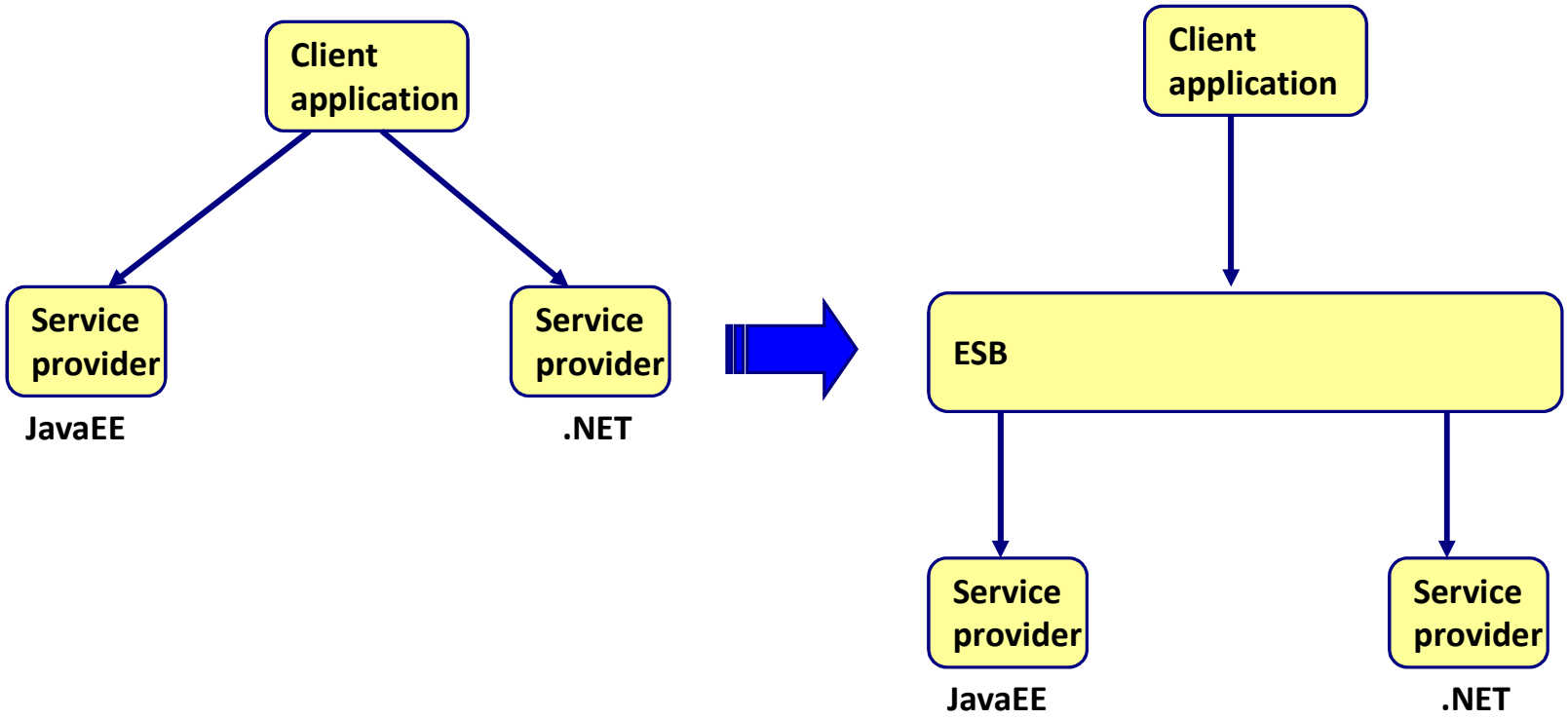
- Routing
  - Static
  - Content based
  - Rule based
  - Policy based
- Message transformation
- Message enhancing/filtering
- Protocol transformation
  - Input transformation
  - Output transformation
- Service mapping
  - Service name, protocol, binding variables, timeout, etc.
- Message processing
  - Guaranteed delivery
- Process choreography
  - Business process
  - BPEL
- Transaction management
- Security



ESB

- Benefits
  - Separation of integration logic and application logic
  - Easy to add new components
- Drawbacks
  - ESB's are complex products
  - Performance can be an issue

ESB architecture pattern



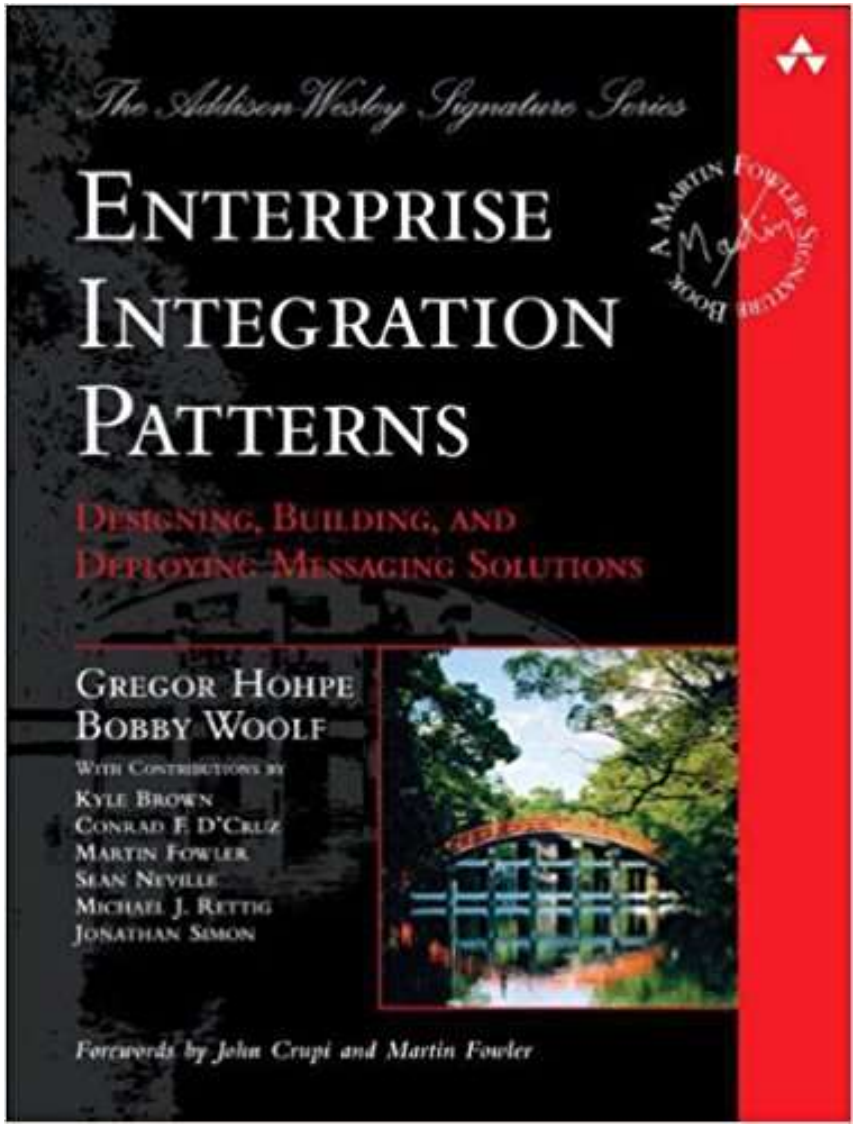
Main point

- The ESB contains all logic to connect all services together.
- The Unified Field, the home of all the laws of nature is the source of everything in creation.

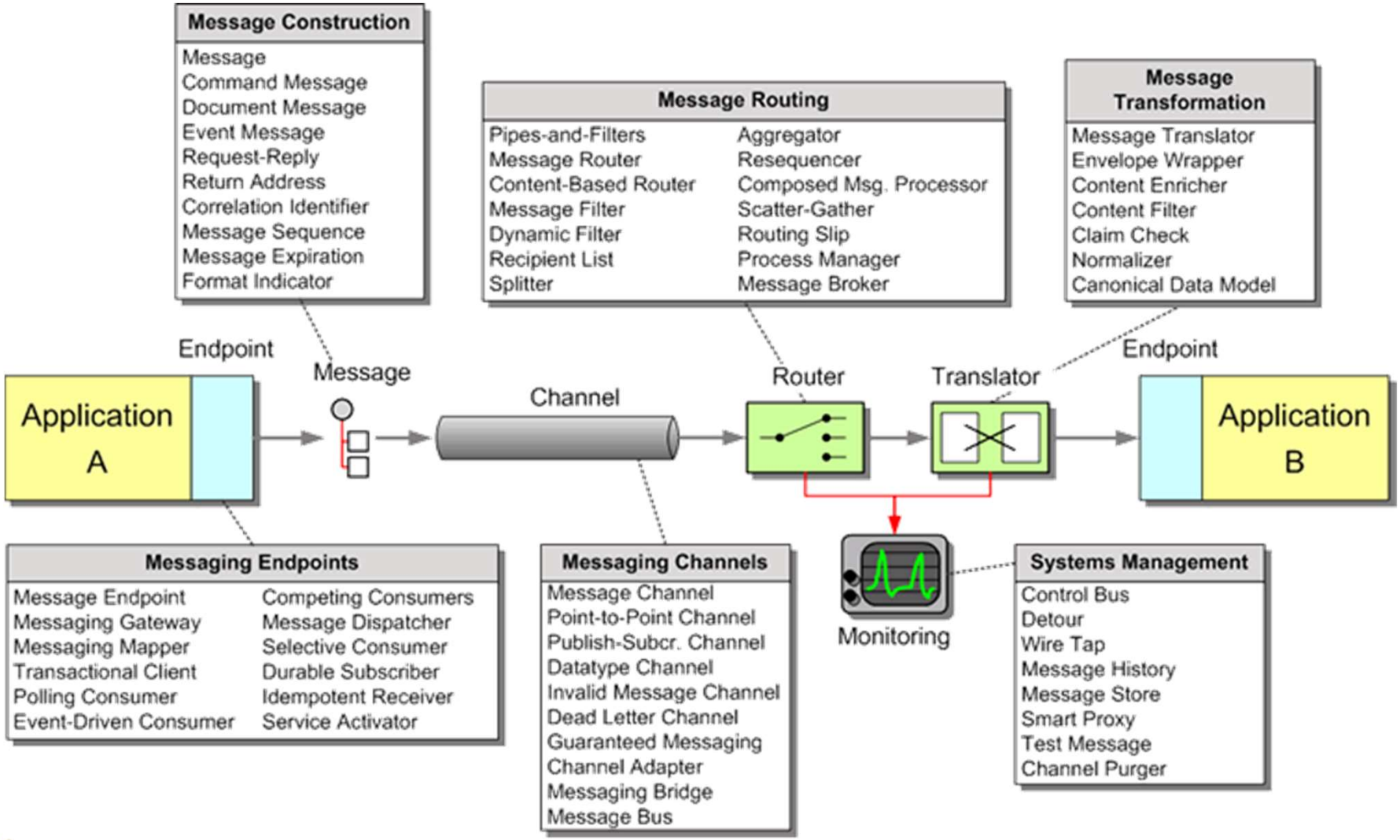
Integration

ENTERPRISE INTEGRATION PATTERNS

Enterprise Integration Patterns

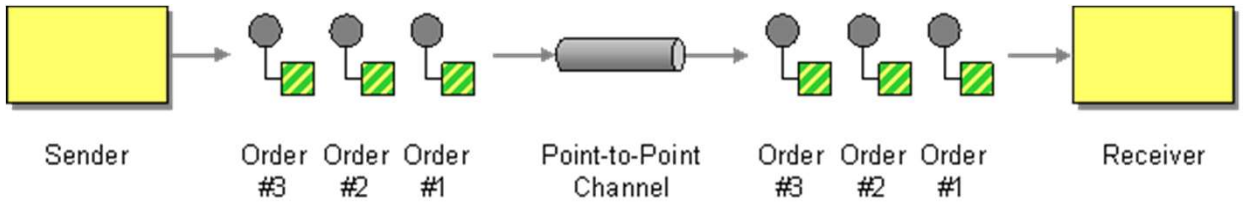


Enterprise Integration Patterns



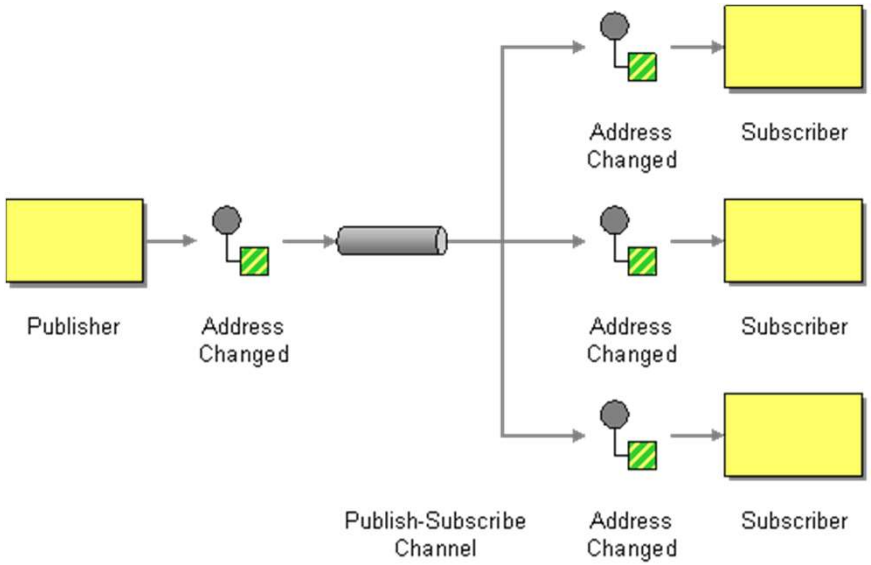
Messaging channel patterns

chanels are message bus



only one receive the message

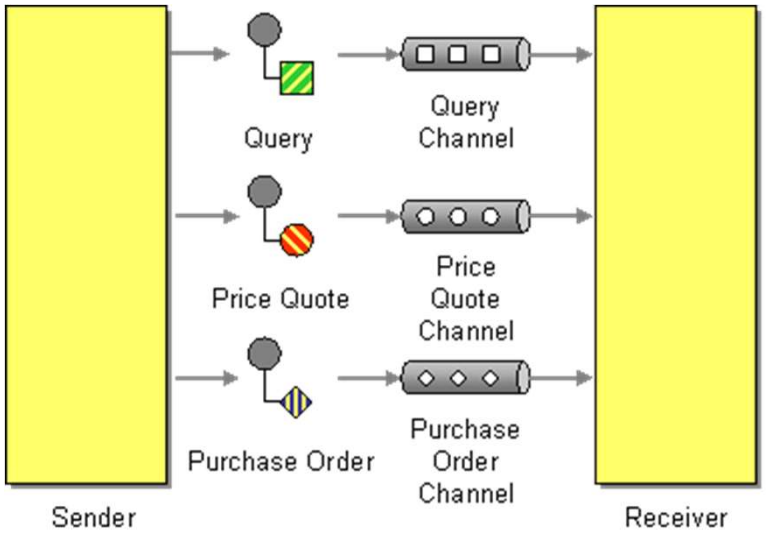
Point-to-point: only one receiver will receive the message



Publish-Subscribe : every subscriber will receive the message

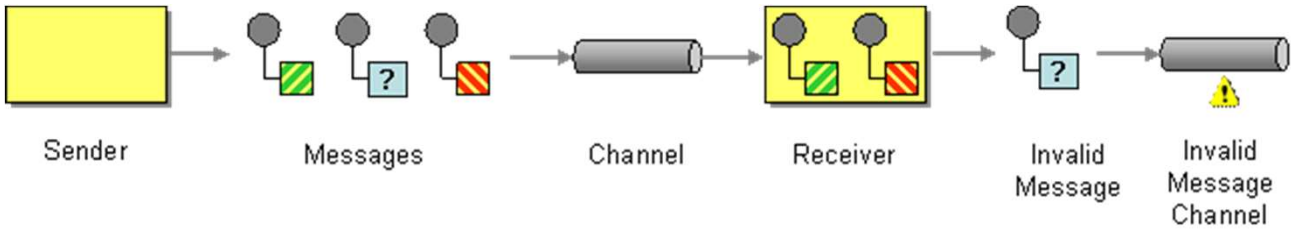
all the suscribers receive the message

Messaging channel patterns

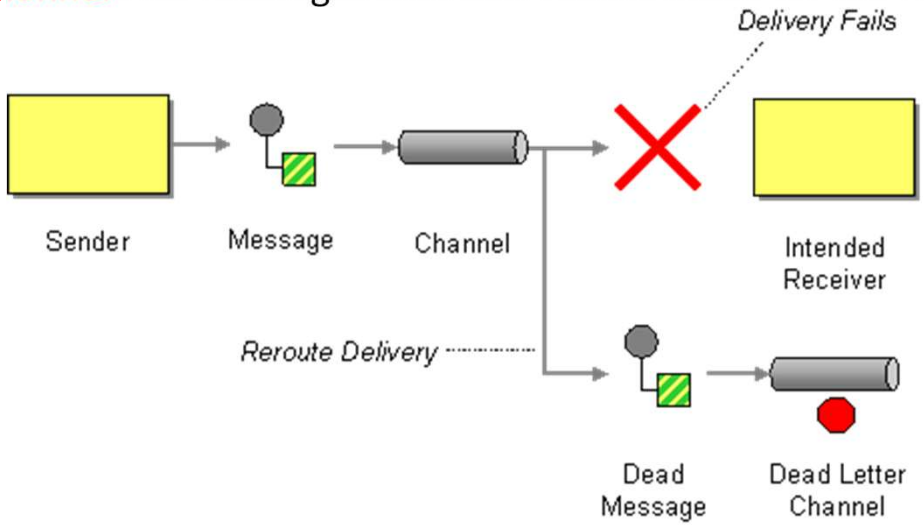


Datatype Channel : use a channel for each data type, so that the receiver know how to process it

Messaging channel patterns

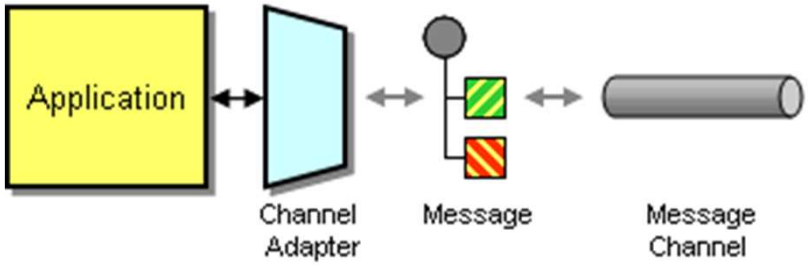


Invalid Message Channel : for messages that don't make sense for the receiver

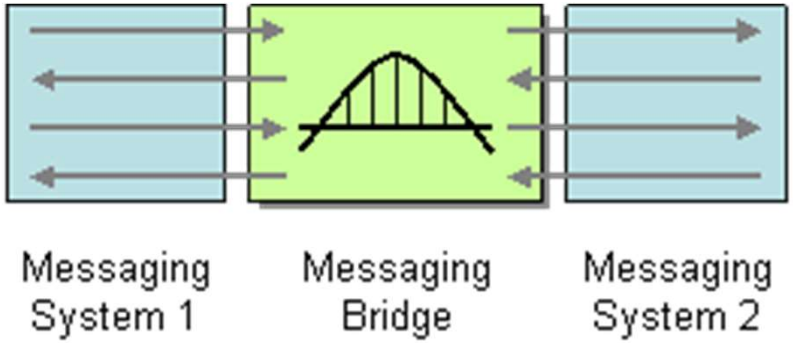


Dead Letter Channel : for messages that can't be delivered

Messaging channel patterns



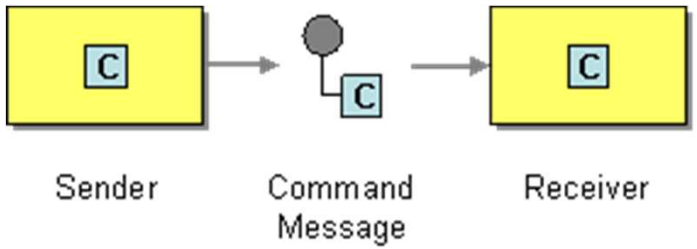
Channel adapter : connect the application to the messaging system



Message bridge : connect 2 messaging systems

Message construction patterns

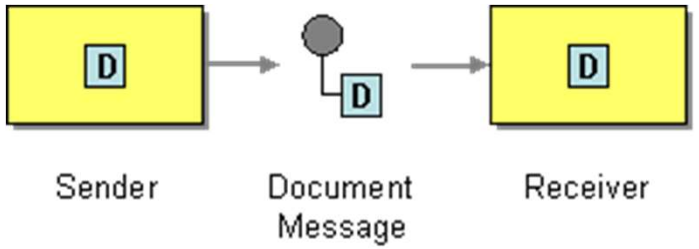
Send command as message succh as last trade price



C = getLastTradePrice("DIS");

Command message

Doc with data as a message such as Order

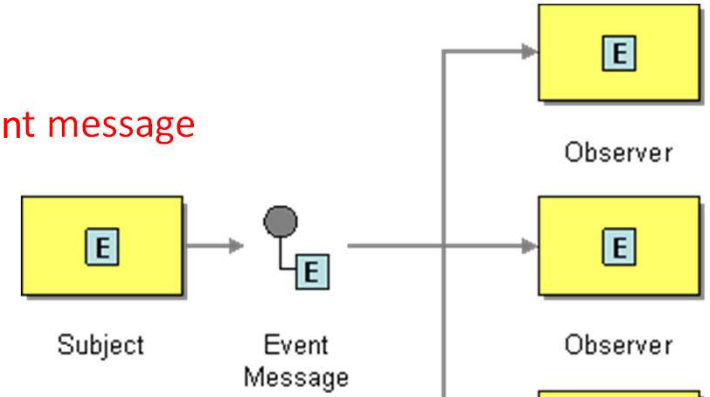


D = aPurchaseOrder

Document message

Event message

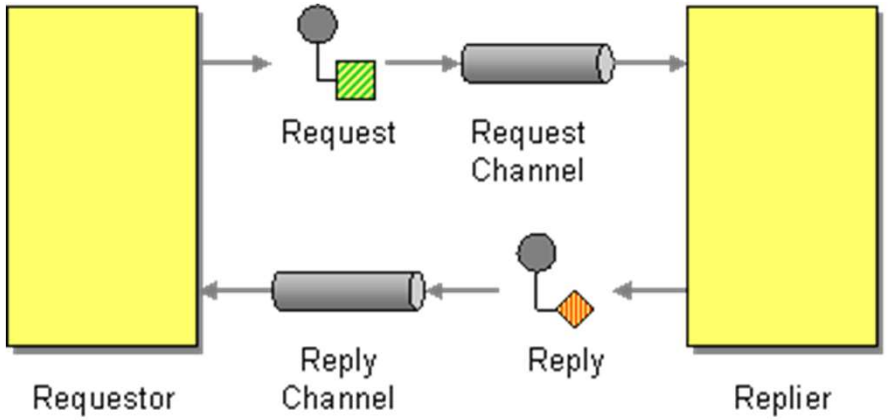
Event =something that happened in the past
Cmmd => something need to happen



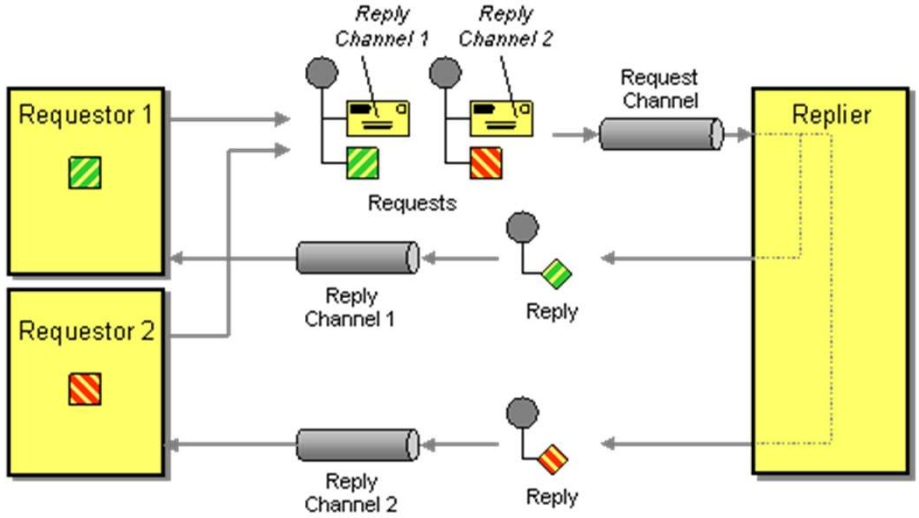
Event as message such as Price change event

E = aPriceChangedEvent

Message construction patterns

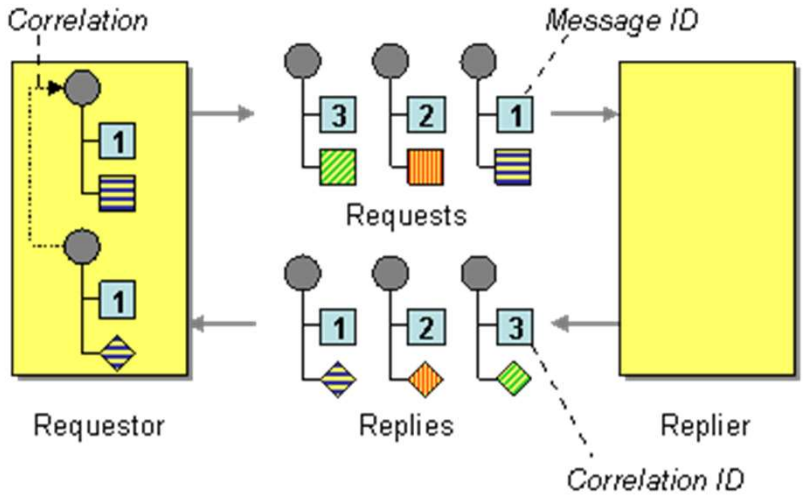


Request-Reply

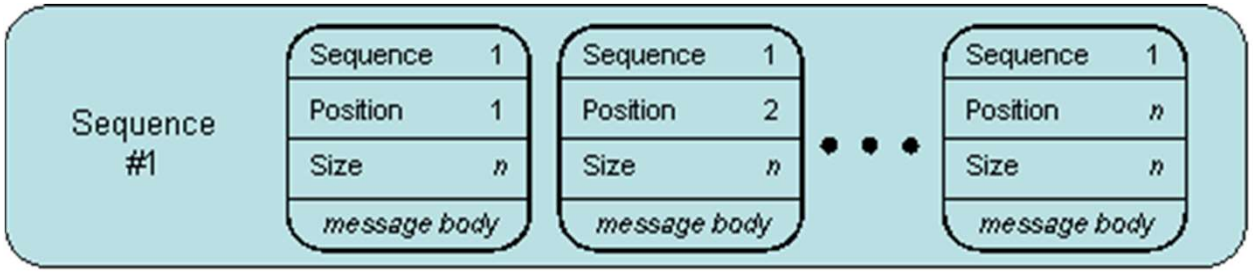


Return address

Message construction patterns

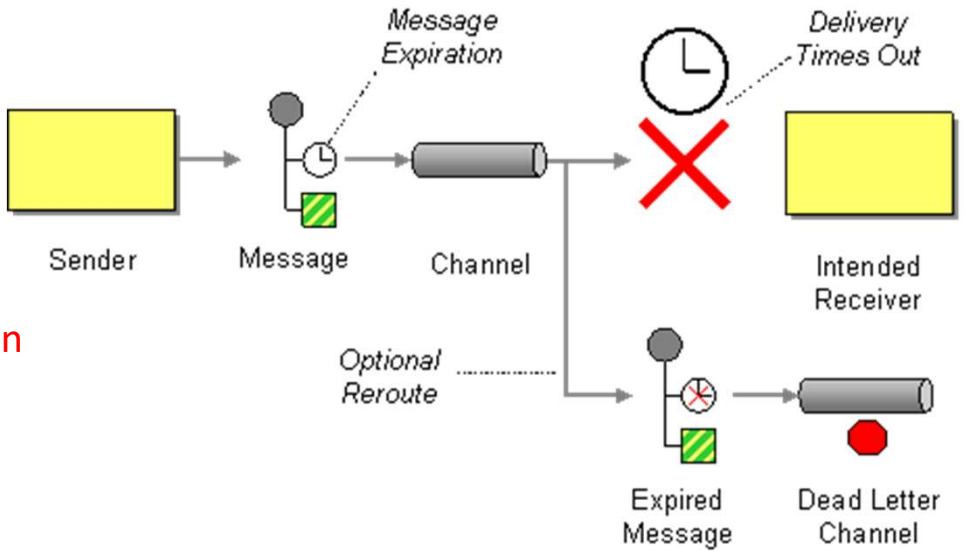


Each reply message should contain a Correlation Identifier, a unique identifier that indicates which request message this reply is for



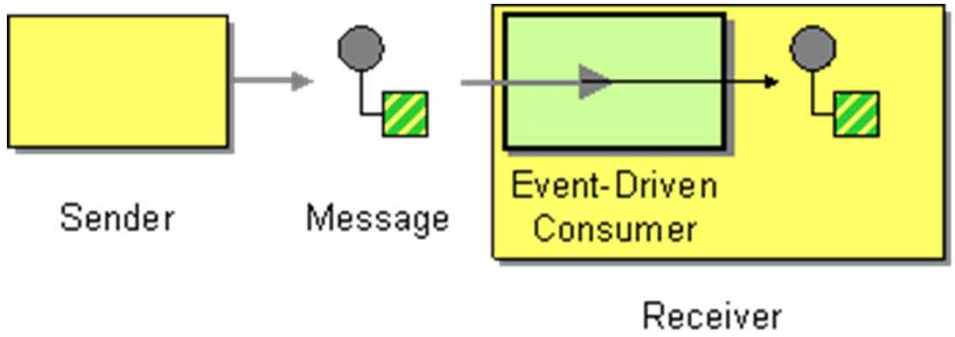
Whenever a large set of data may need to be broken into message-size chunks, send the data as a Message Sequence and mark each message with sequence identification fields.

Message construction patterns

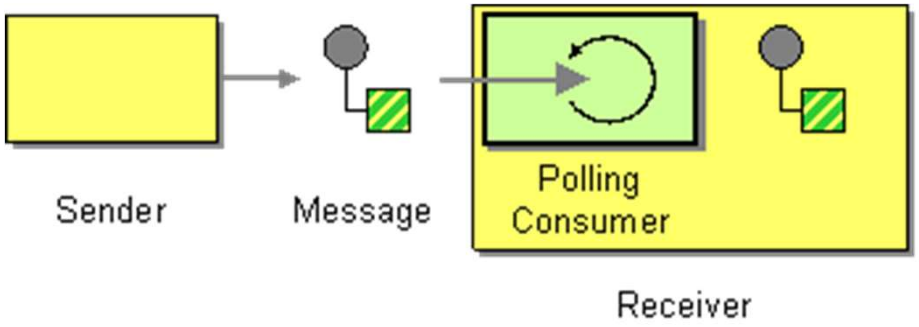


Message expiration

Message Endpoint

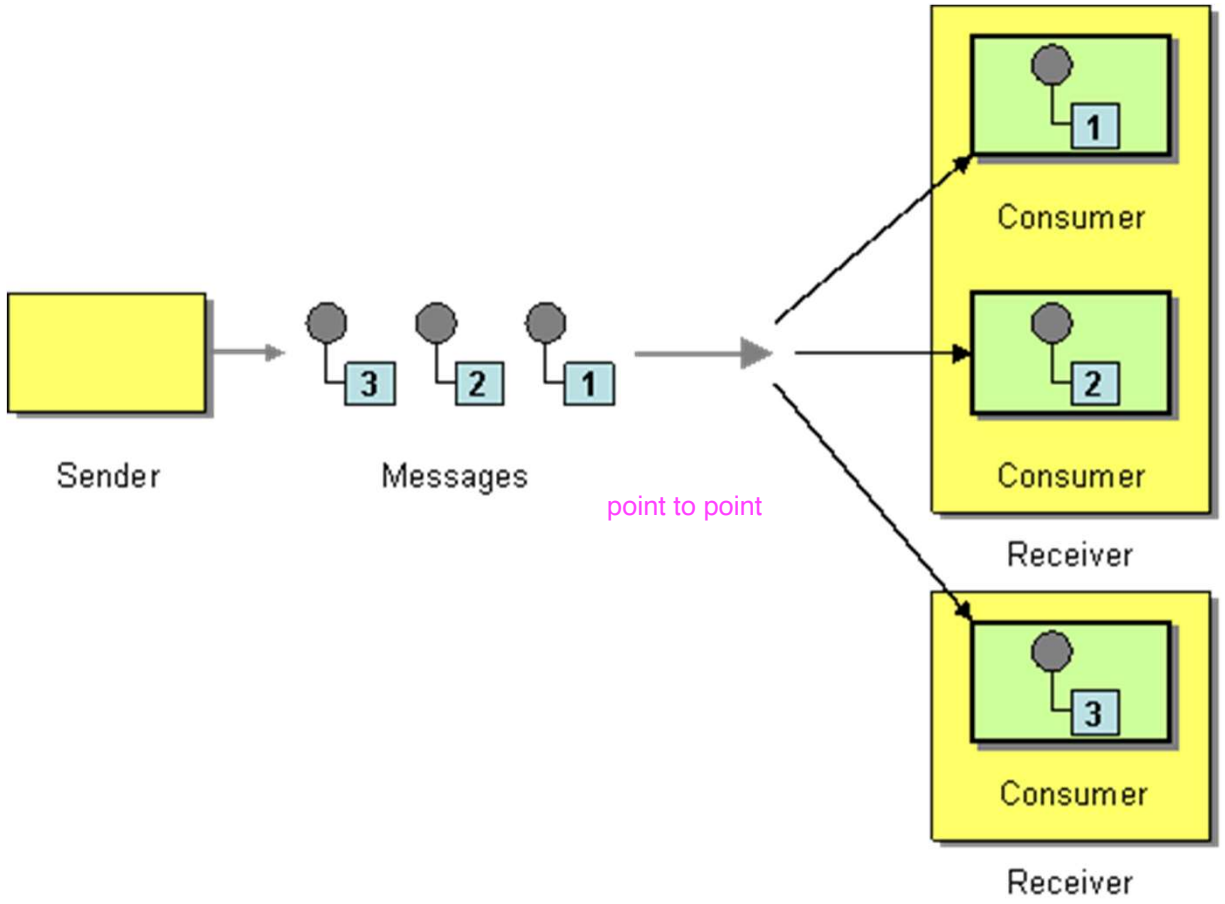


Event driven consumer



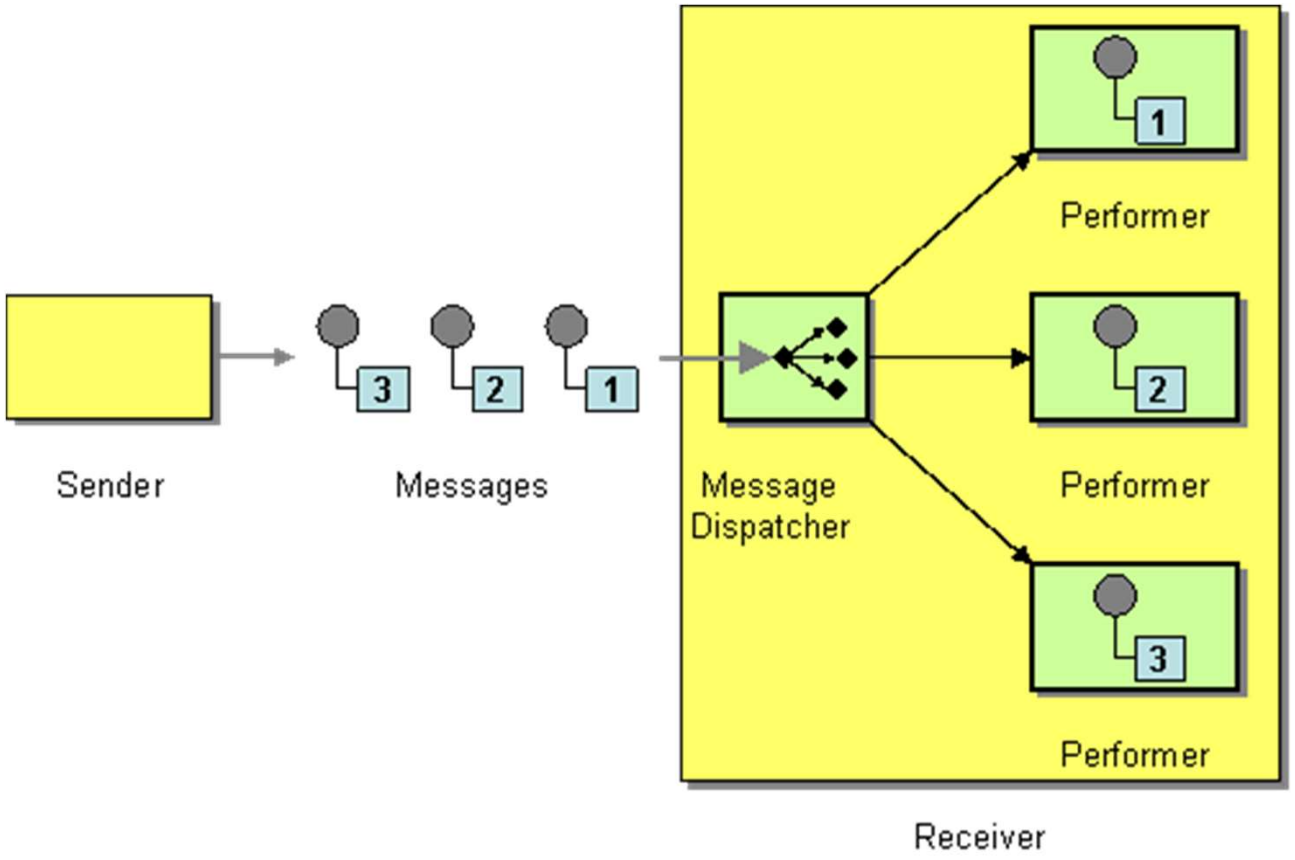
Polling consumer

Message Endpoint



Competing consumers

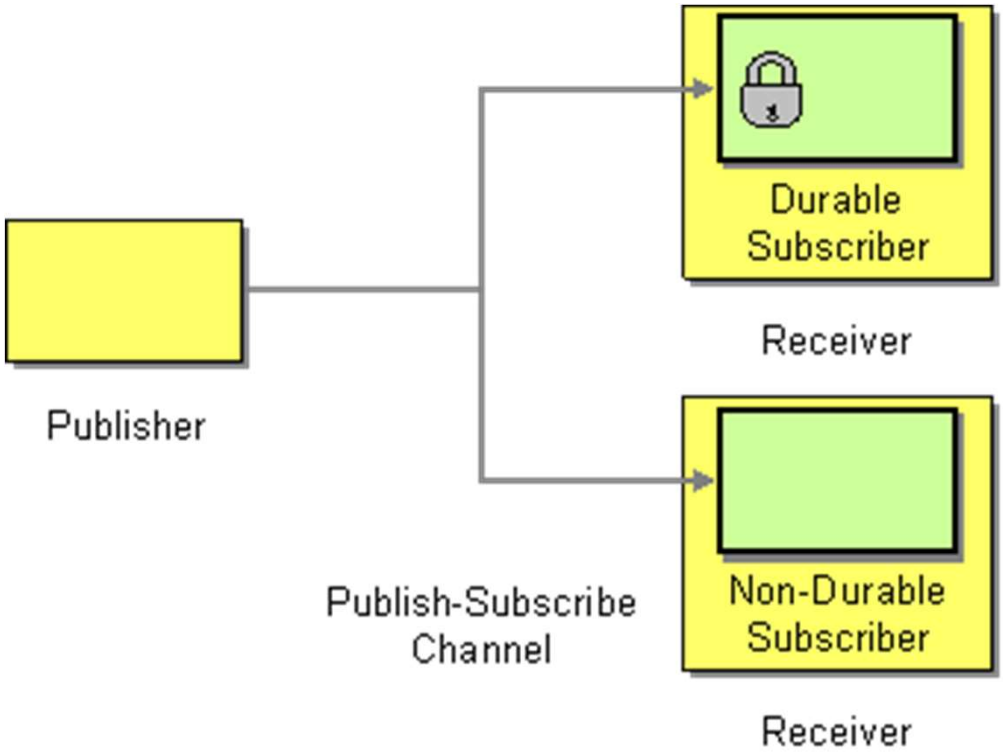
Message Endpoint



Message dispatcher

Message Endpoint

When is offline the channnel keeps the msg , when become online, will send to it

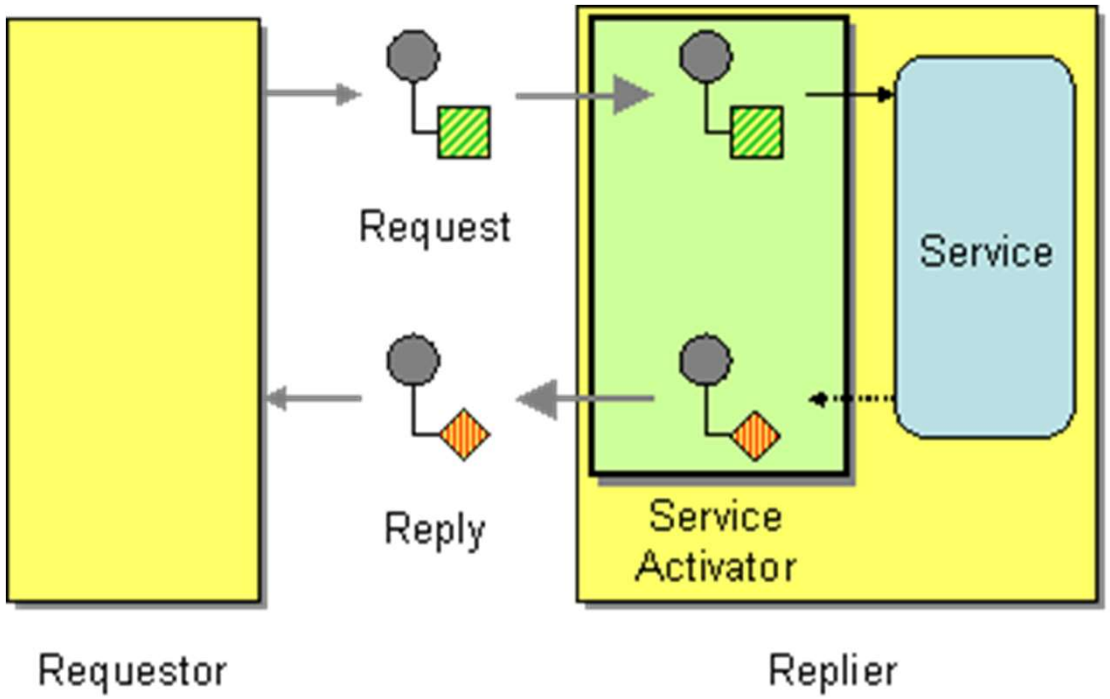


only gets the msg when it is online

Durable and Non-Durable subscribers

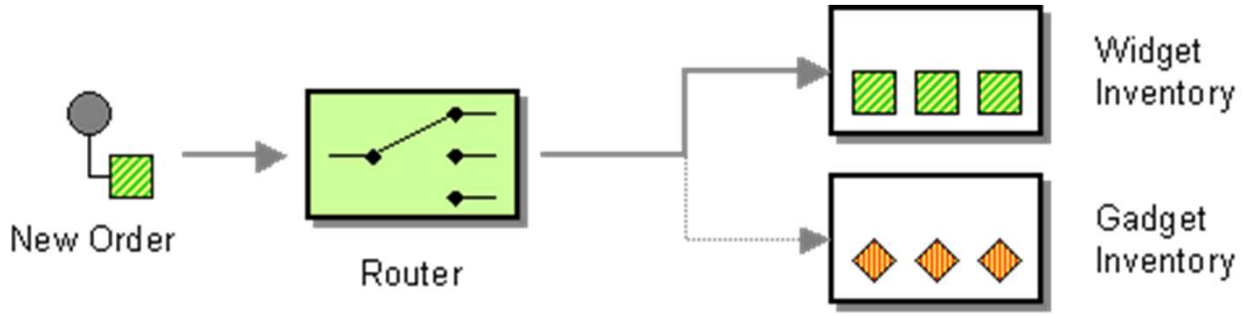
Message Endpoint

if a msg comes to this channel=> call a service
,service activator calls a method on an object



Service activator

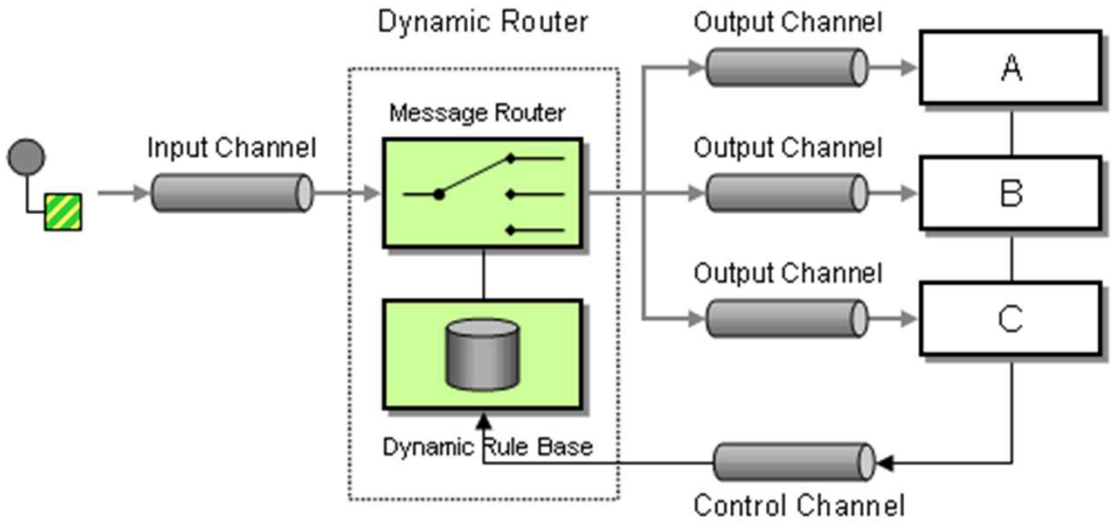
Message Routing



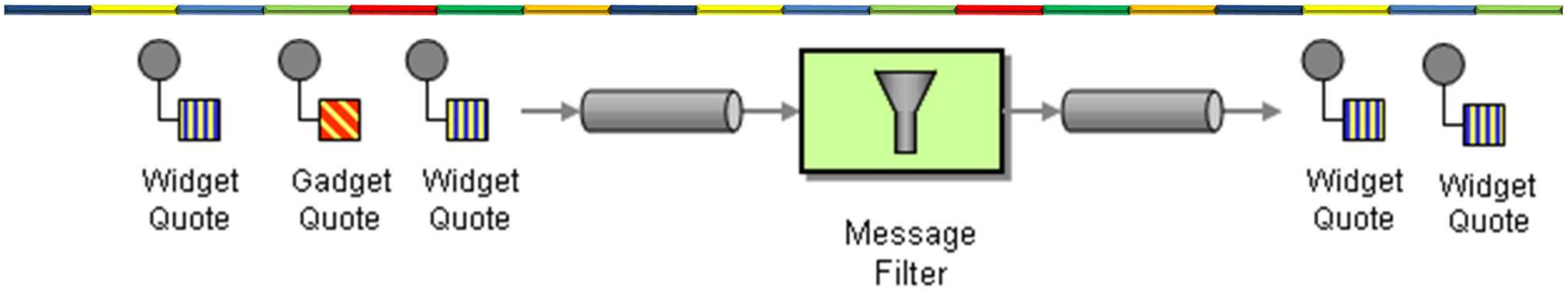
Content based router

Dynamic router has some rules, knows when gets some thing , where should send to

Dynamic router



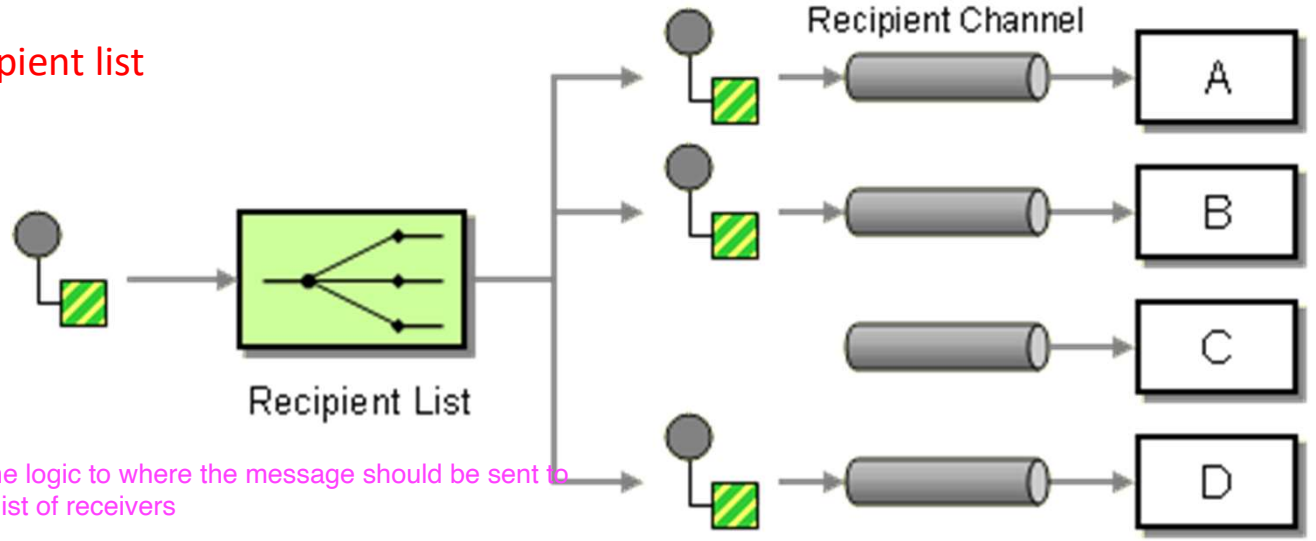
Message Routing



It filter gadgets => just widget can continue

Message filter

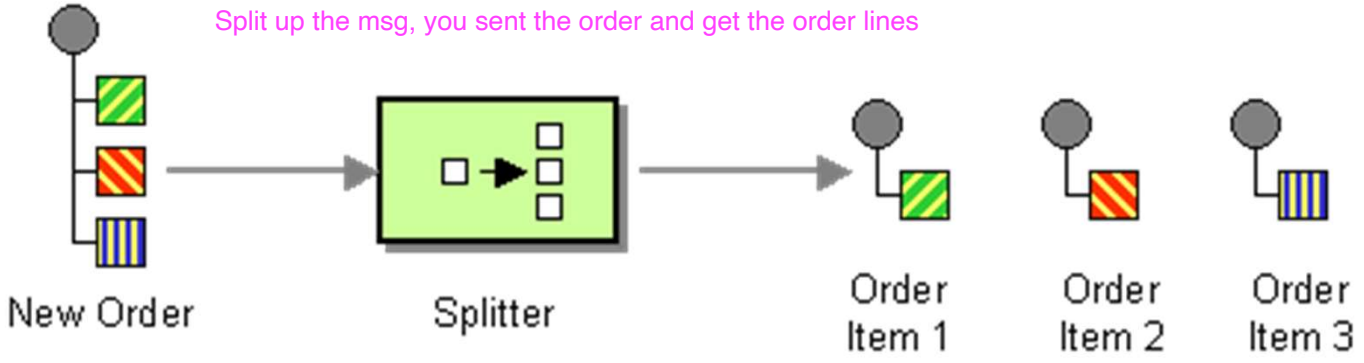
Recipient list



contains the logic to where the message should be sent to the whole list of receivers

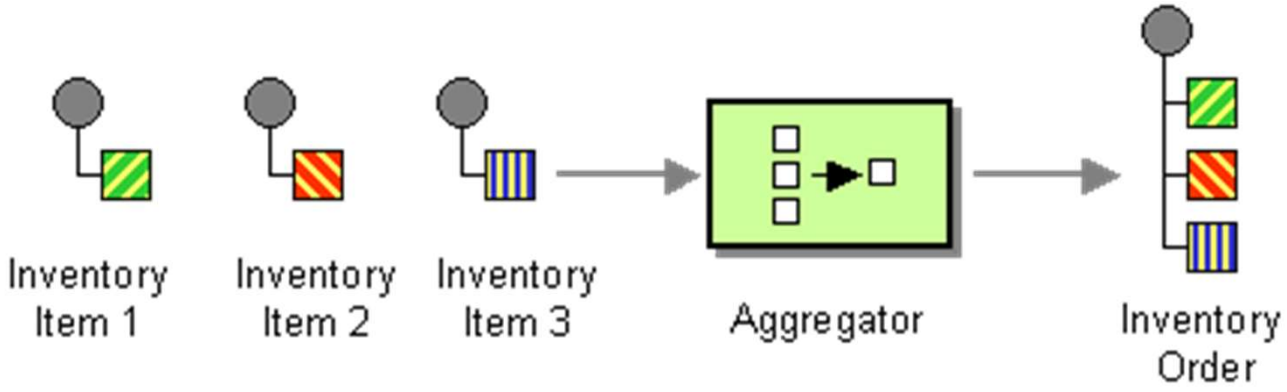
Message Routing

Split up the msg, you sent the order and get the order lines



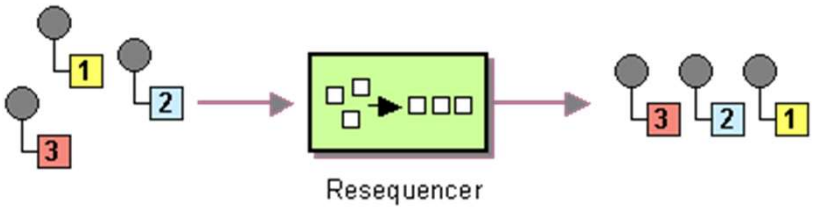
most of the time you need to write logic

Splitter

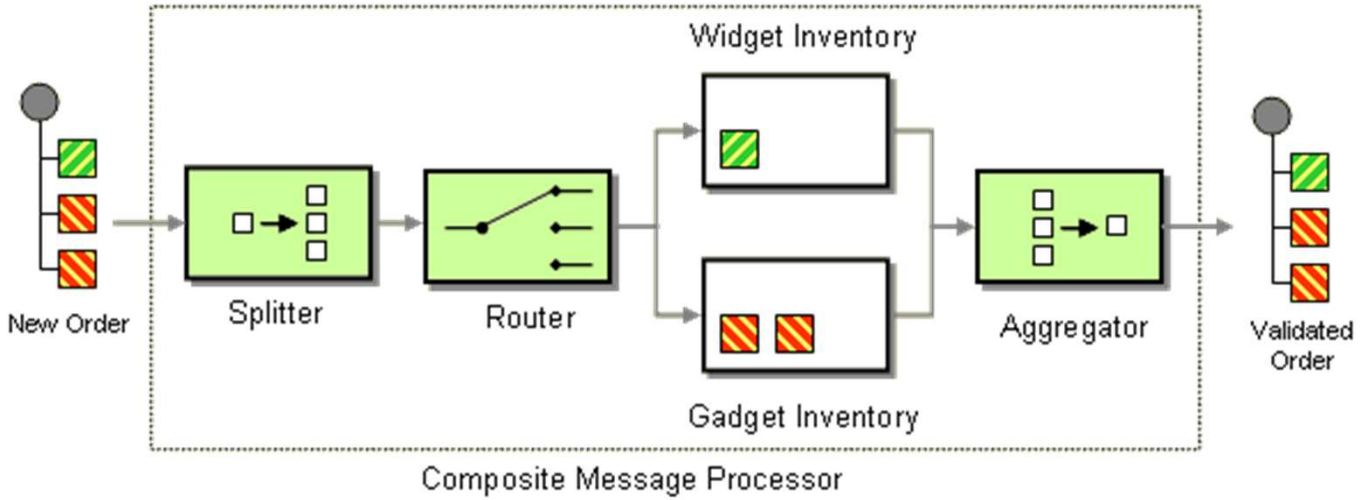


Aggregator

Message Routing

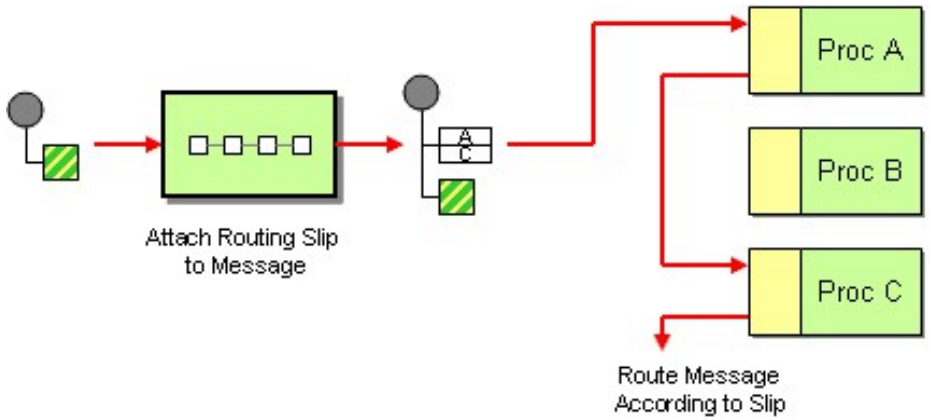


Sequencer



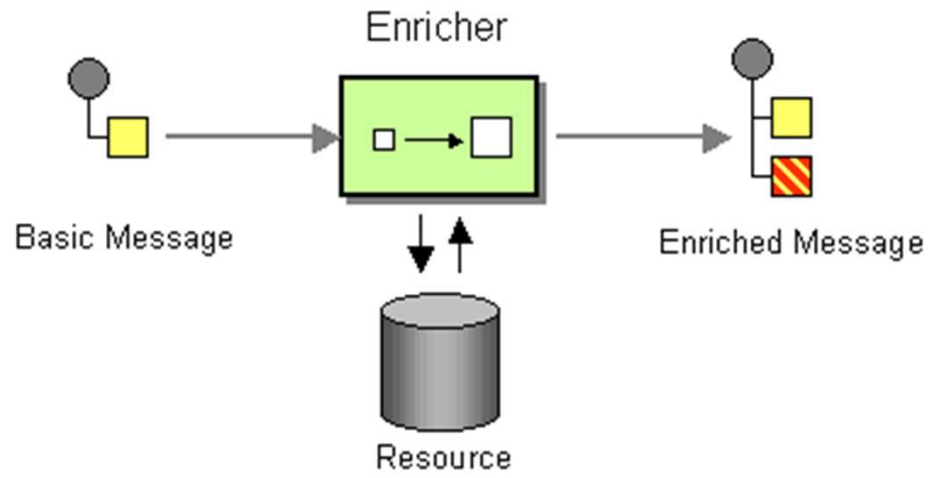
Composite Message Processor

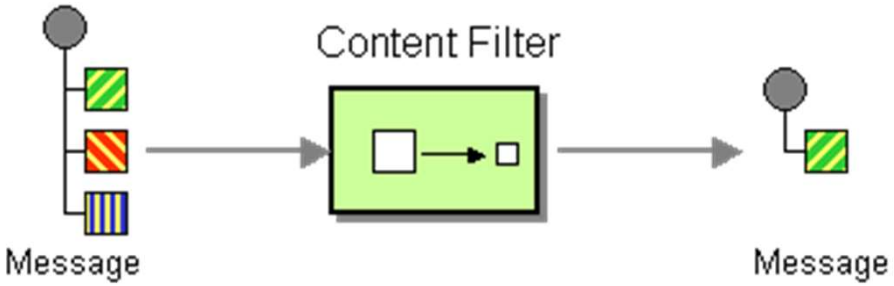
Message Routing



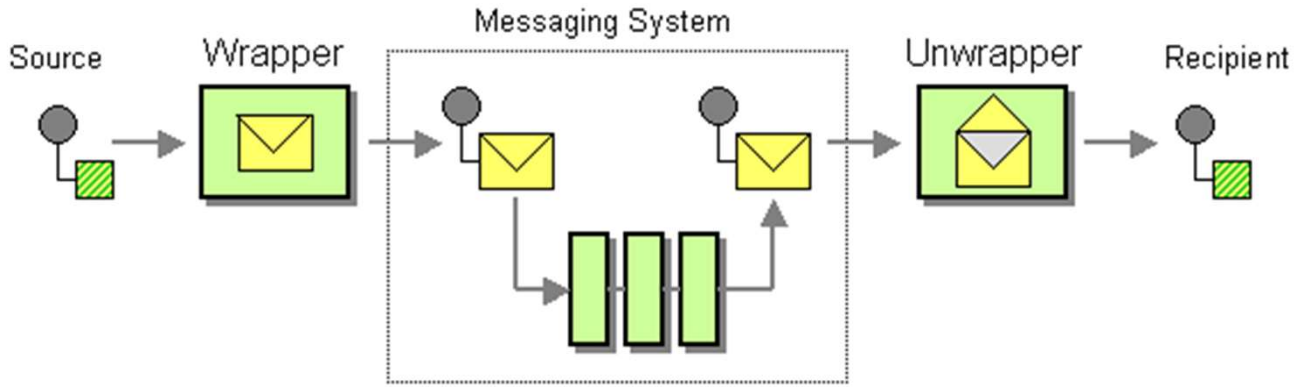
Attach a Routing Slip to each message, specifying the sequence of processing steps. Wrap each component with a special message router that reads the *Routing Slip* and routes the message to the next component in the list

Message Transformation

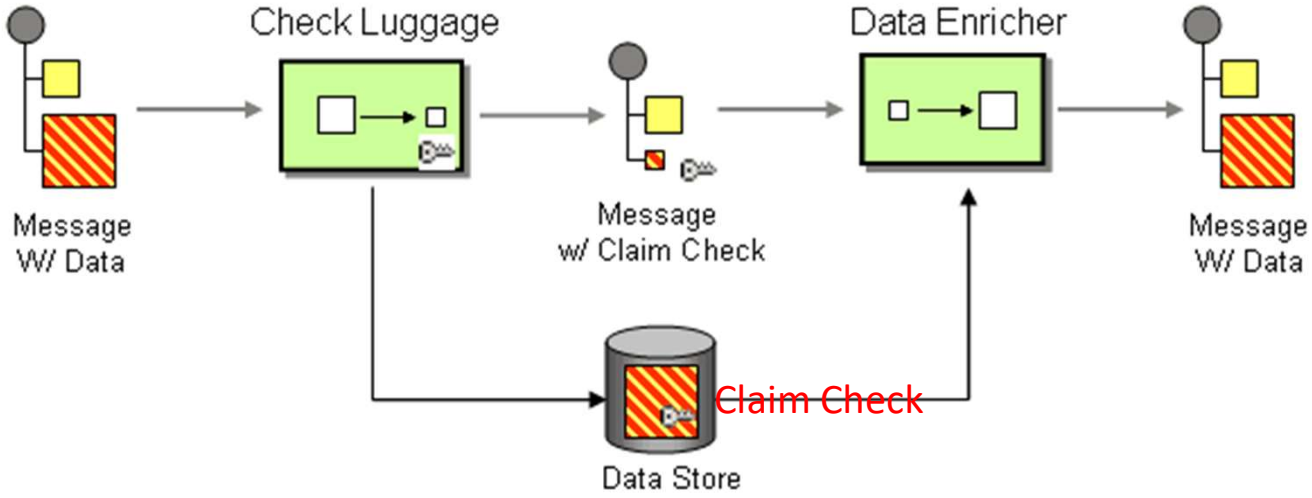




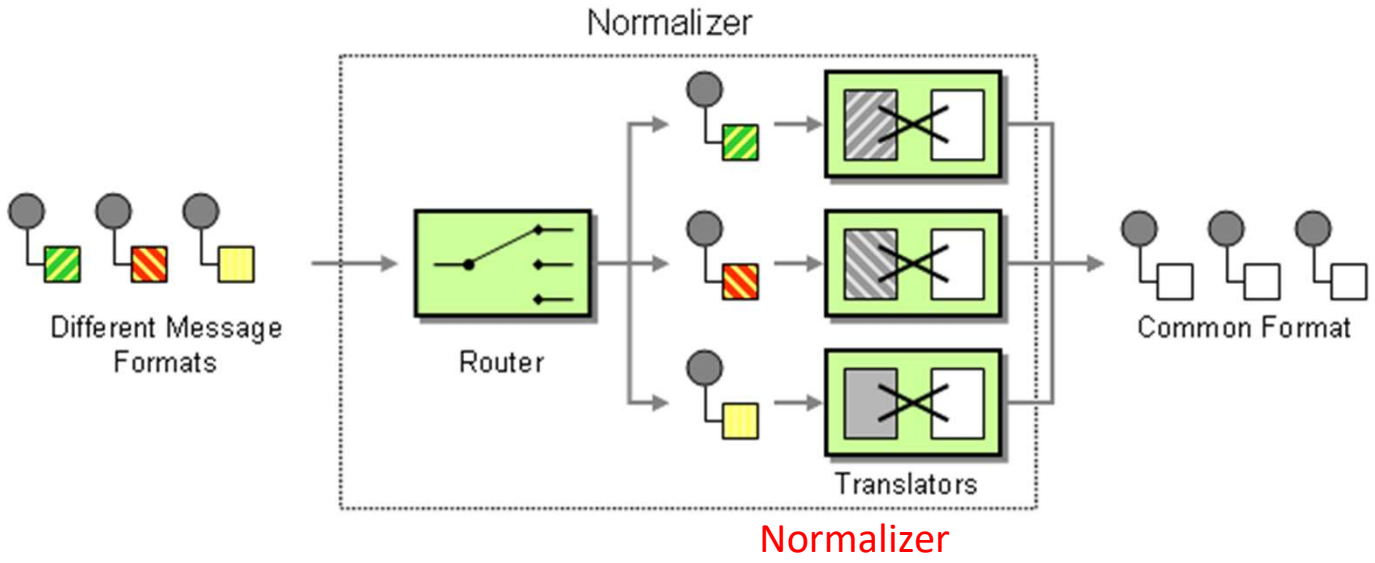
Message Transformation



Envelope Wrapper

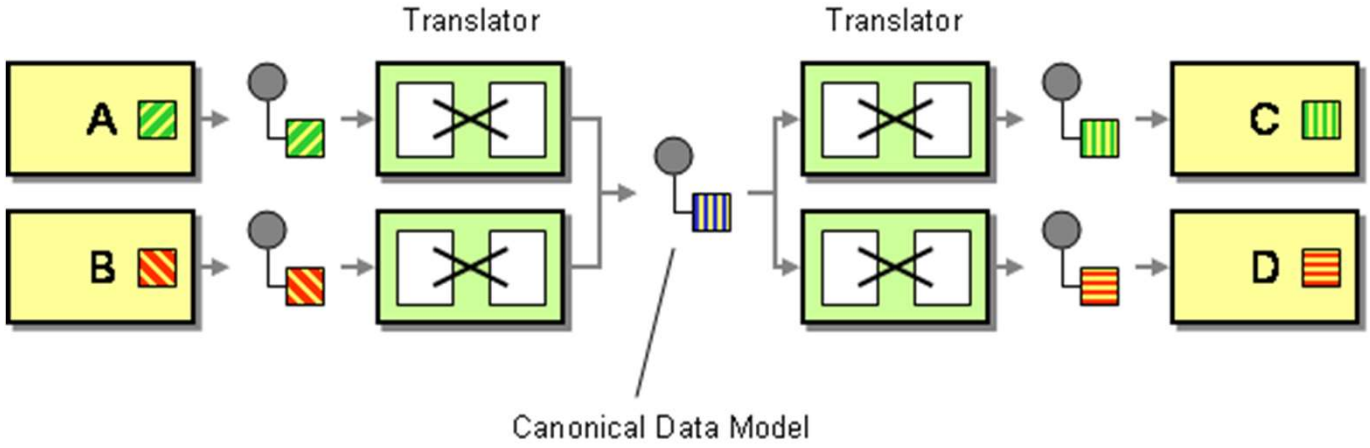


Message Transformation

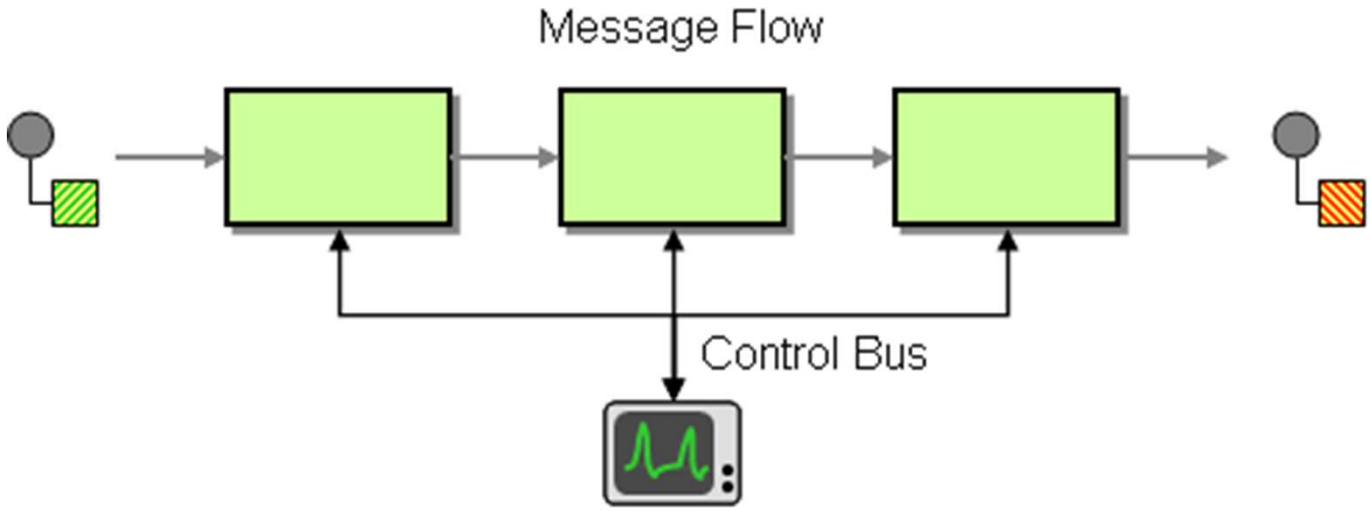


Normalizer

Canonical Data Model



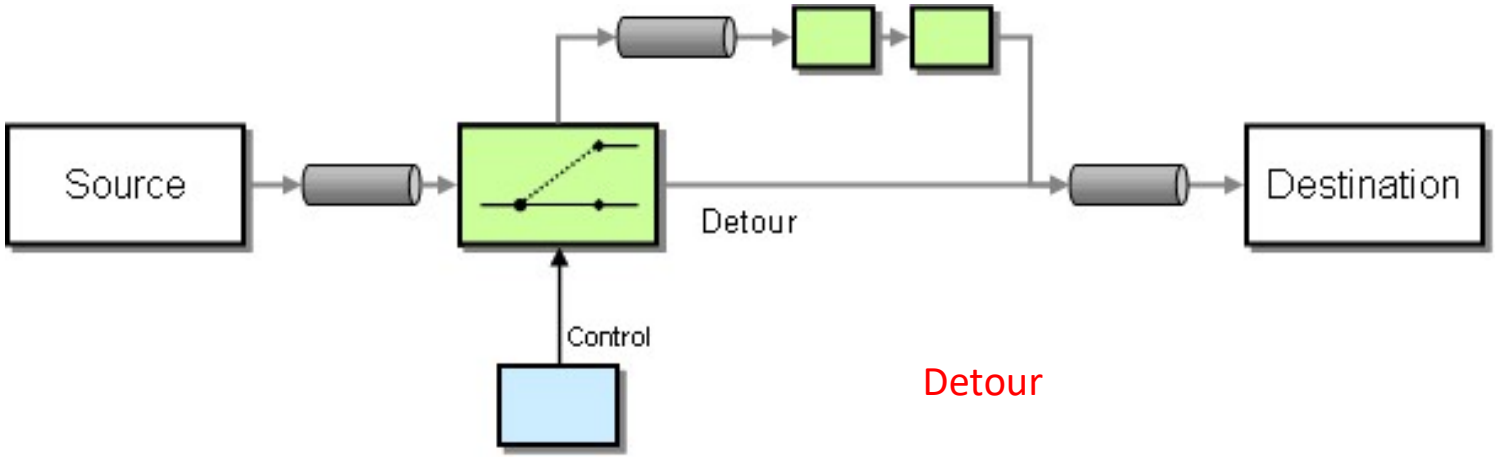
Management



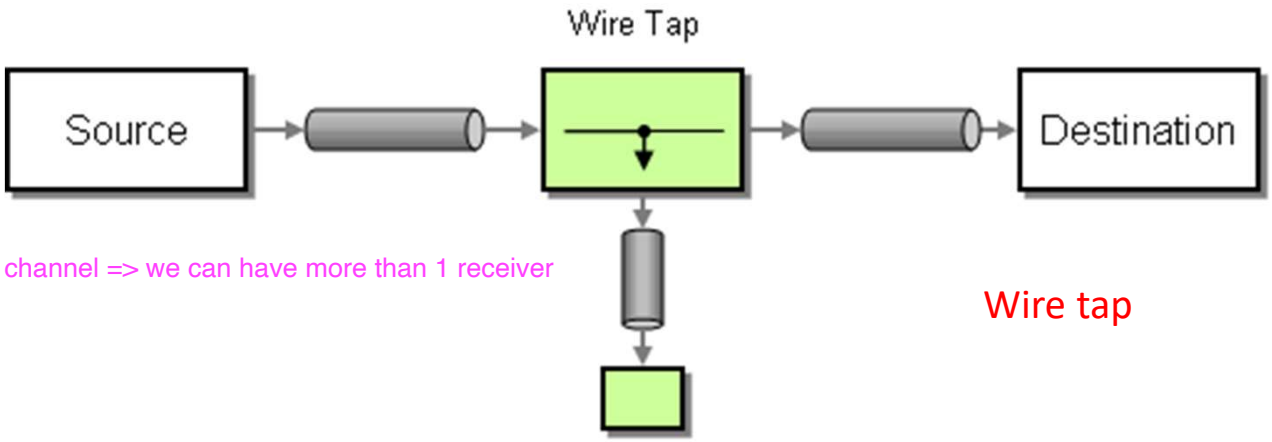
Use a *Control Bus* to manage an enterprise integration system.

- same messaging mechanism
- separate channels to transmit management relevant data

Management



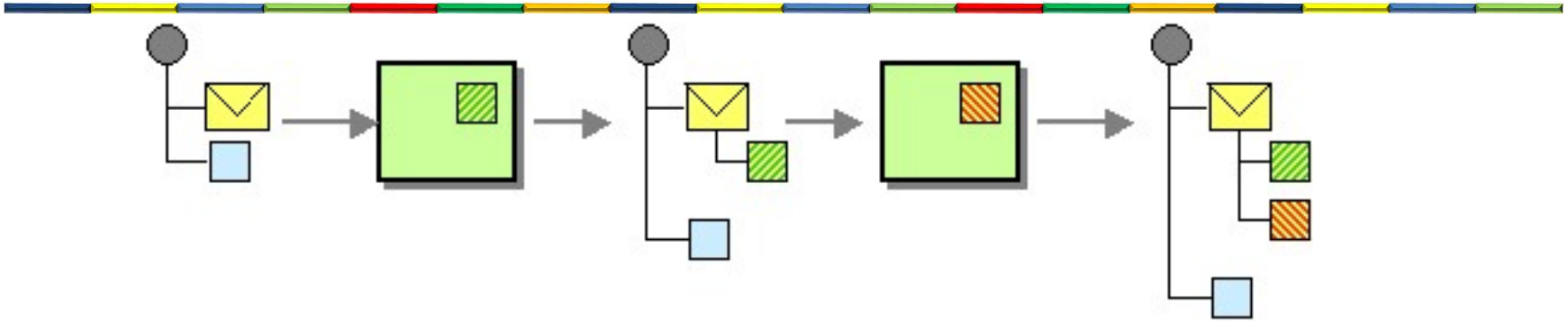
Detour



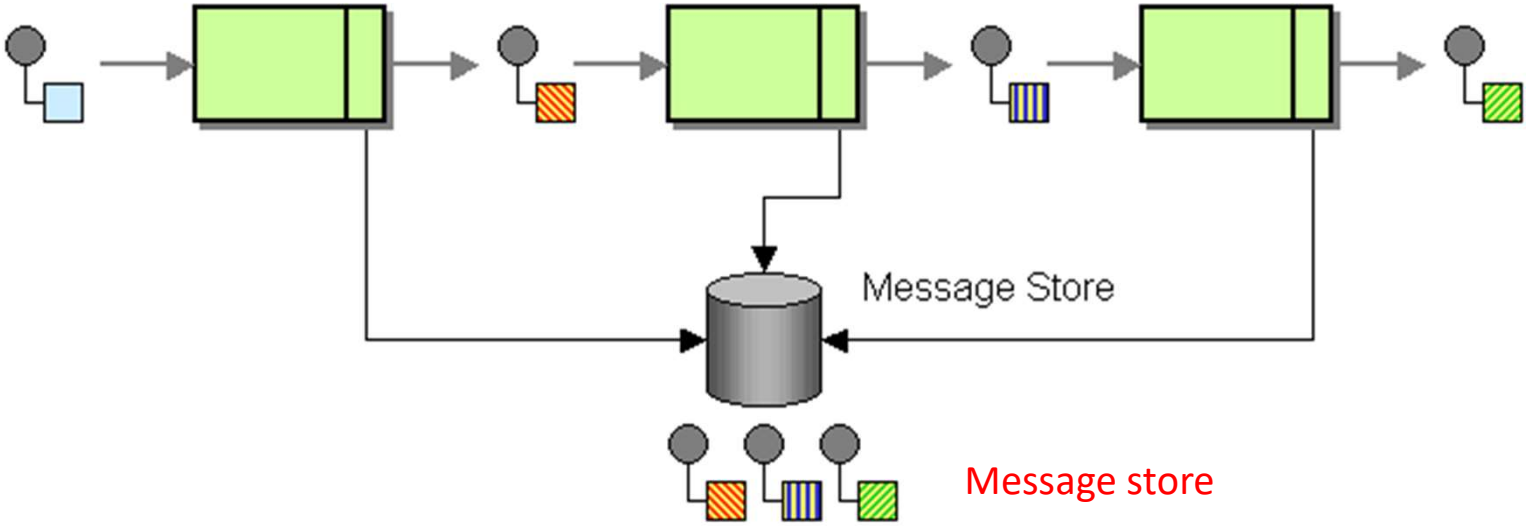
for point to point channel => we can have more than 1 receiver

Wire tap

Management

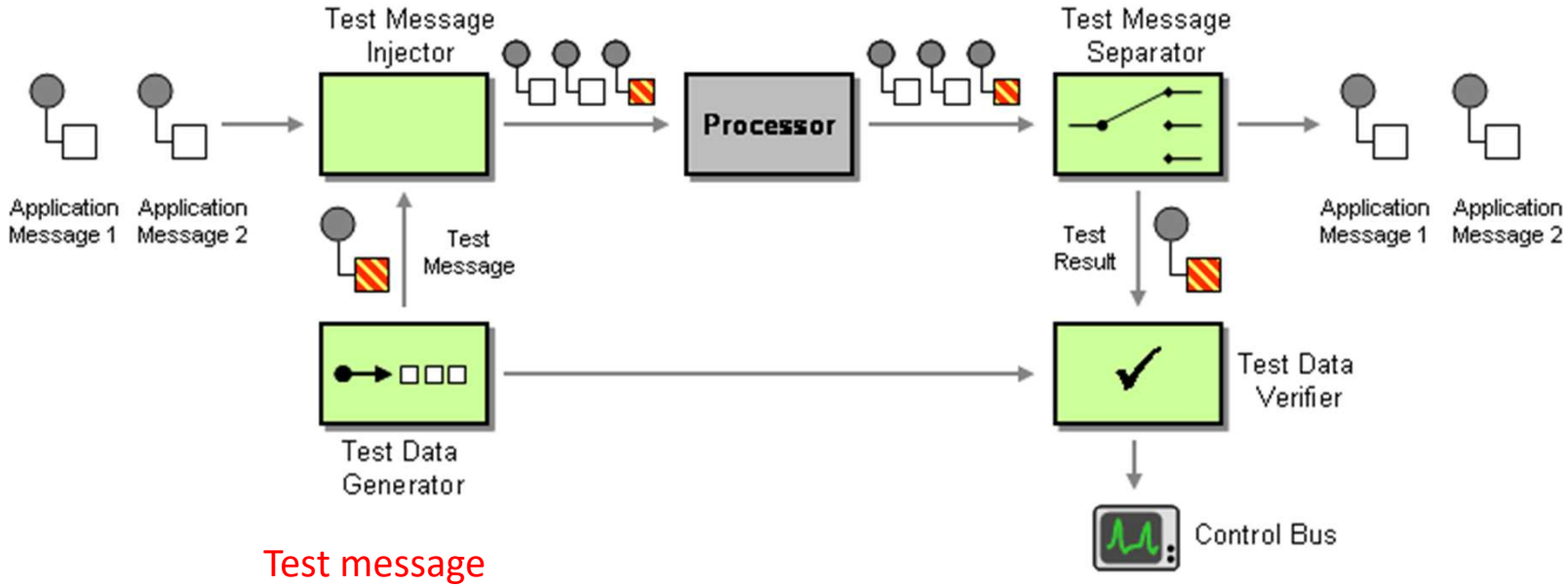


Message history



Message store

Management

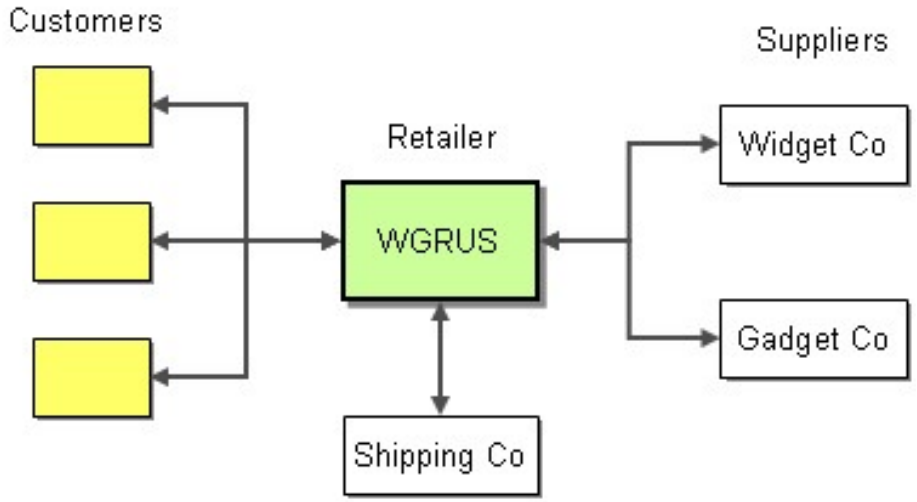


Test message

Main point

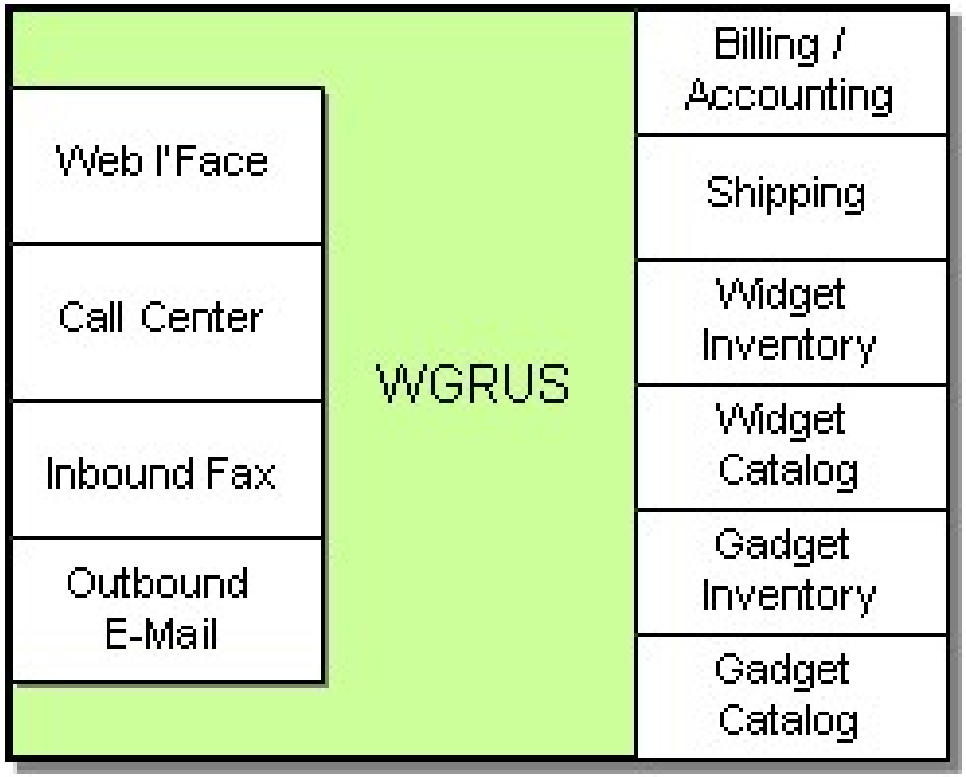
- There are many integration patterns that one can use to integrate different systems together.
- Support of Nature increases when one's thoughts and actions become more in tune with the laws of Nature.

Example: Widgets&Gatchets ‘R Us (WGRUS)

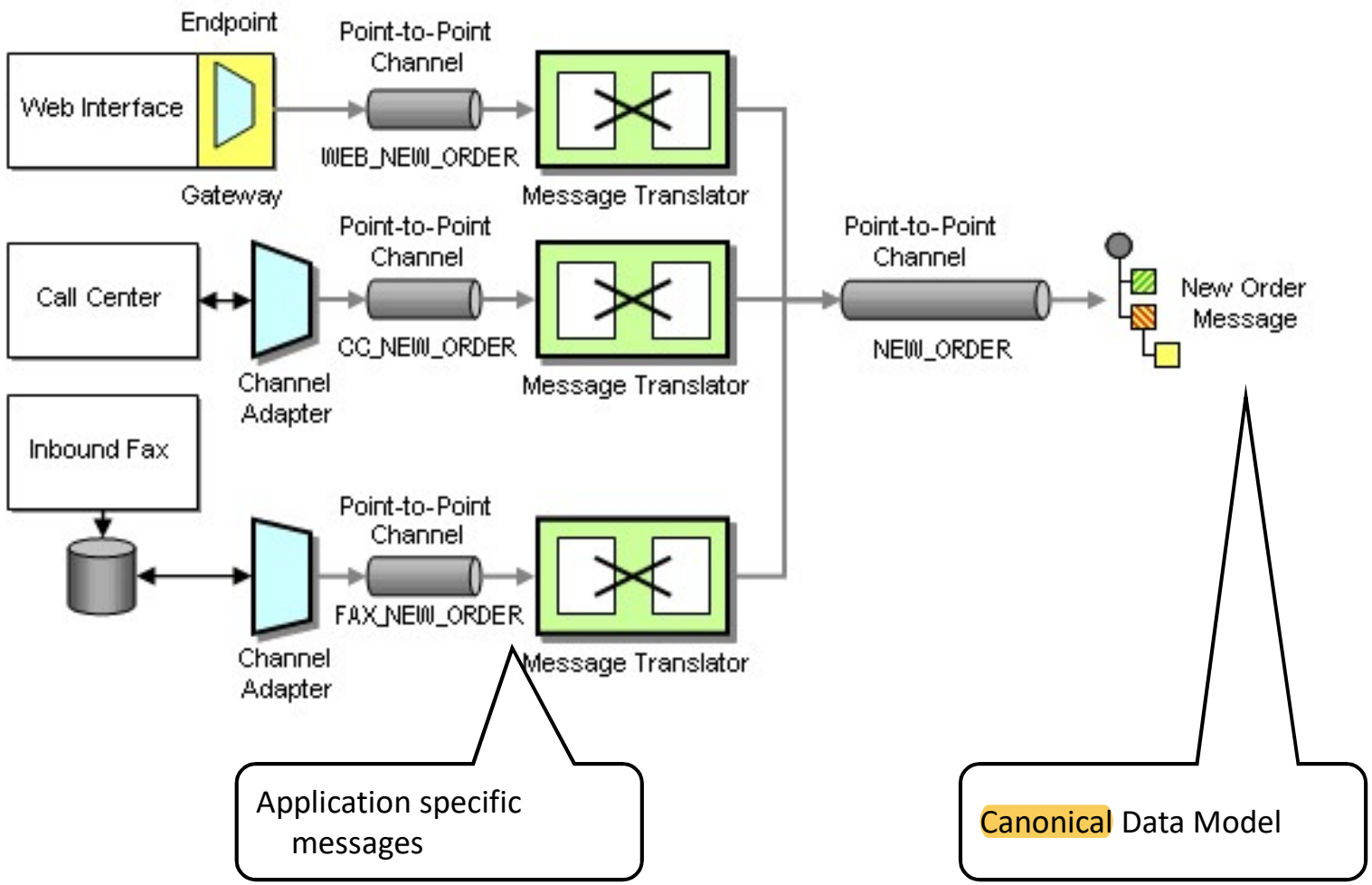


- **Take Orders**: Customers can place orders via Web, phone or fax
- **Process Orders**: Processing an order involves multiple steps, including verifying inventory, shipping the goods and invoicing the customer
- **Check Status**: Customers can check the order status
- **Change Address**: Customers can use a Web front-end to change their billing and shipping address
- **New Catalog**: The suppliers update their catalog periodically. WGRUS needs to update its pricing and availability based in the new catalogs.
- **Announcements**: Customers can subscribe to selective announcements from WGRUS.
- **Testing and Monitoring**: The operations staff needs to be able to monitor all individual components and the message flow between them.

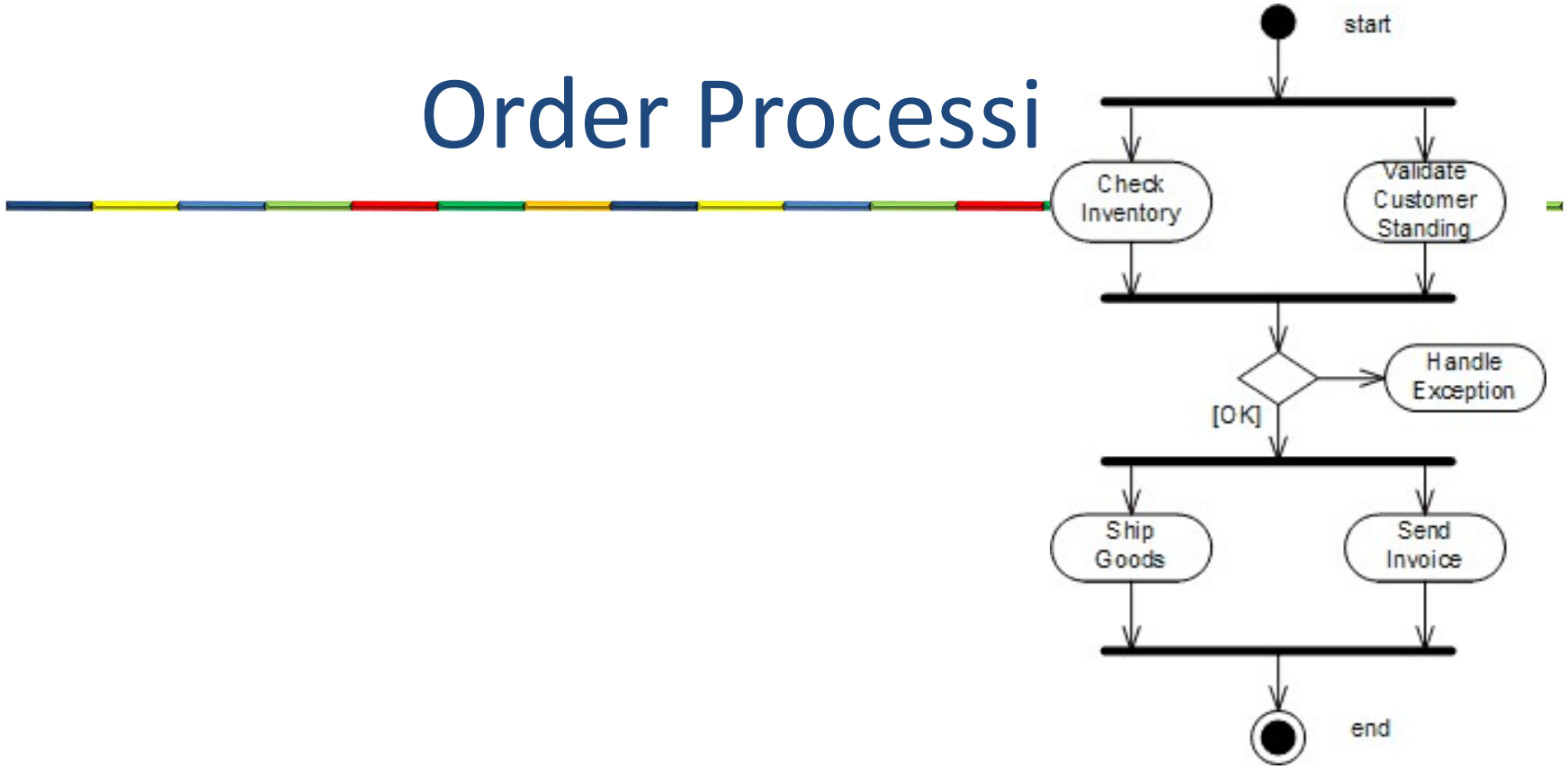
WGRUS internal IT infrastructure

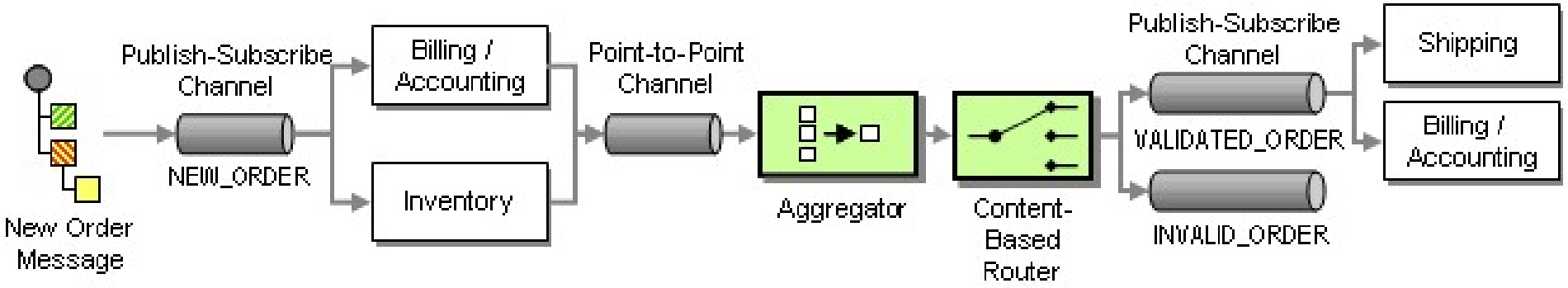


Taking orders from 3 different channels

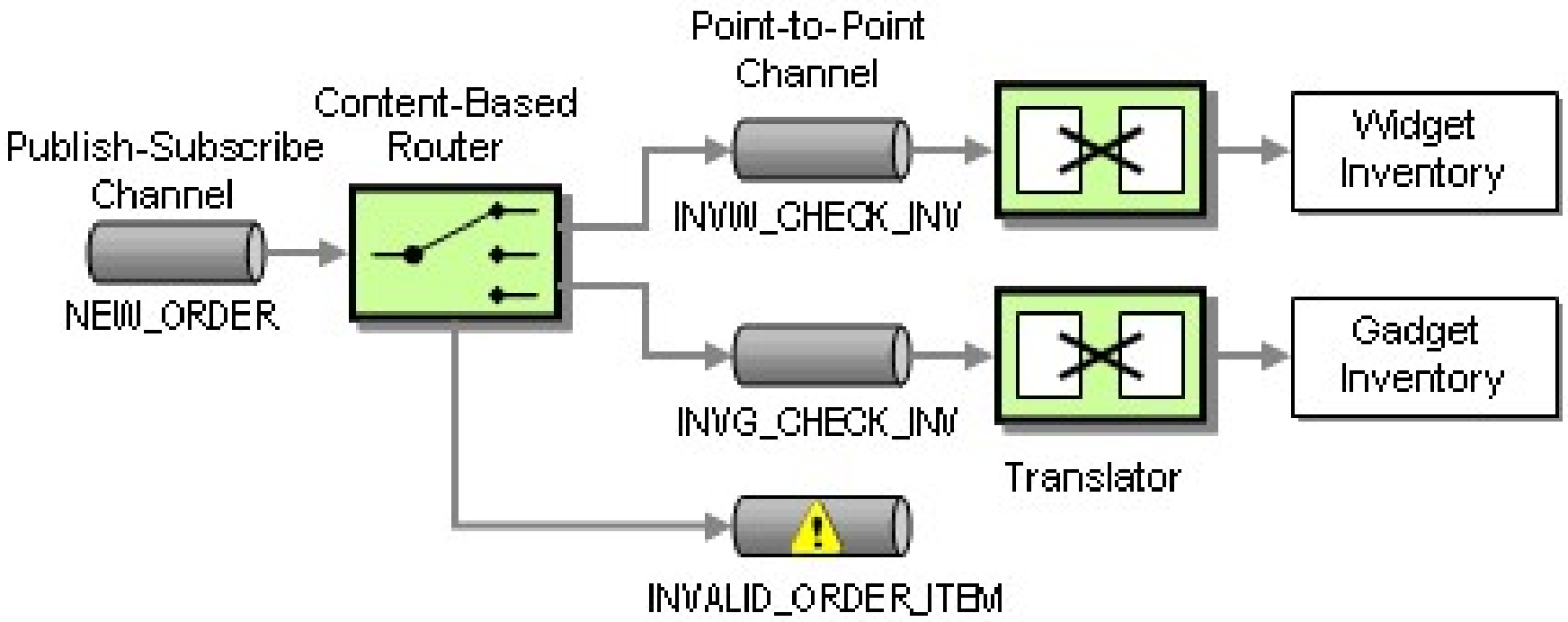


Order Processi



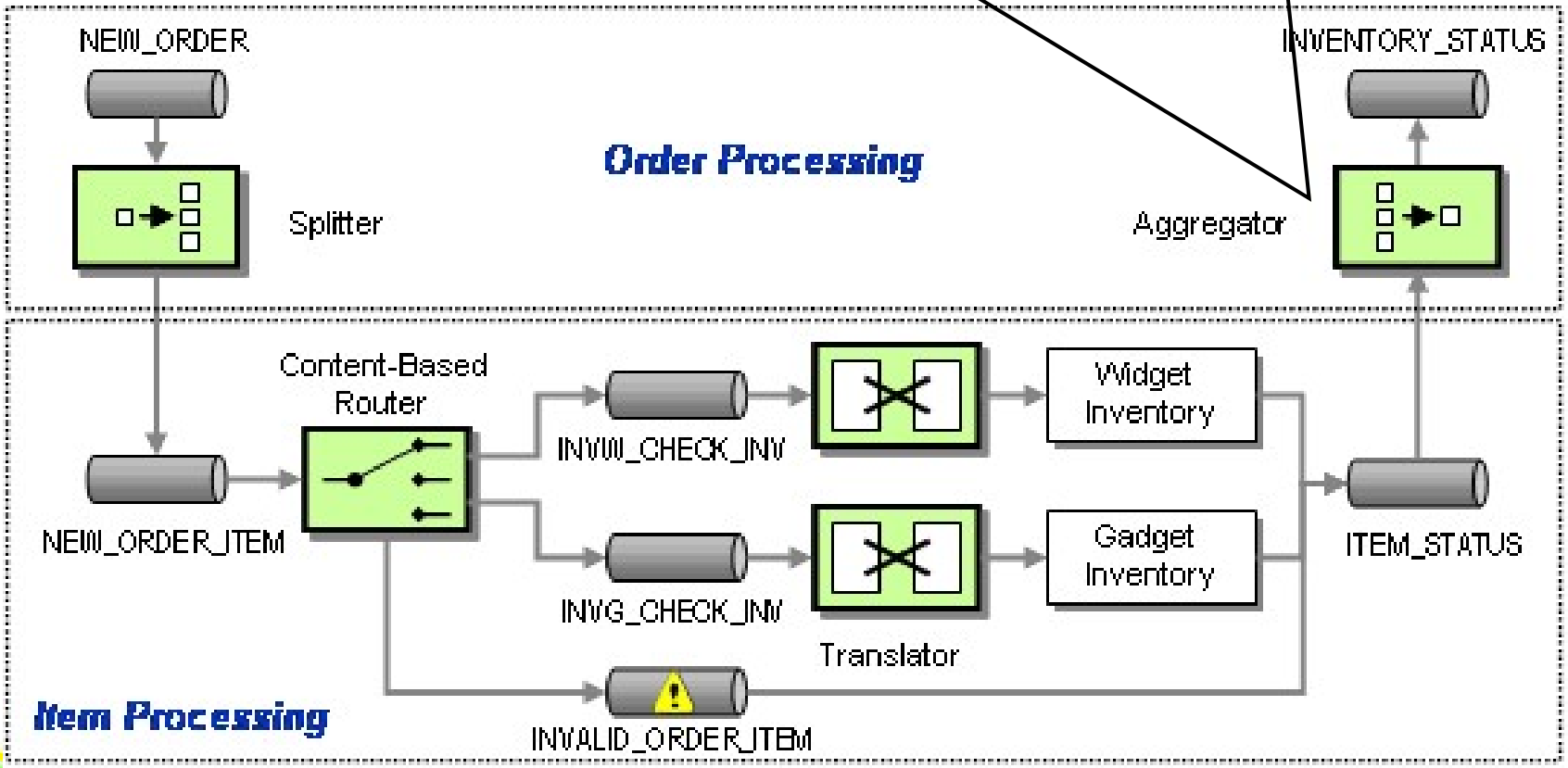


Routing the inventory request

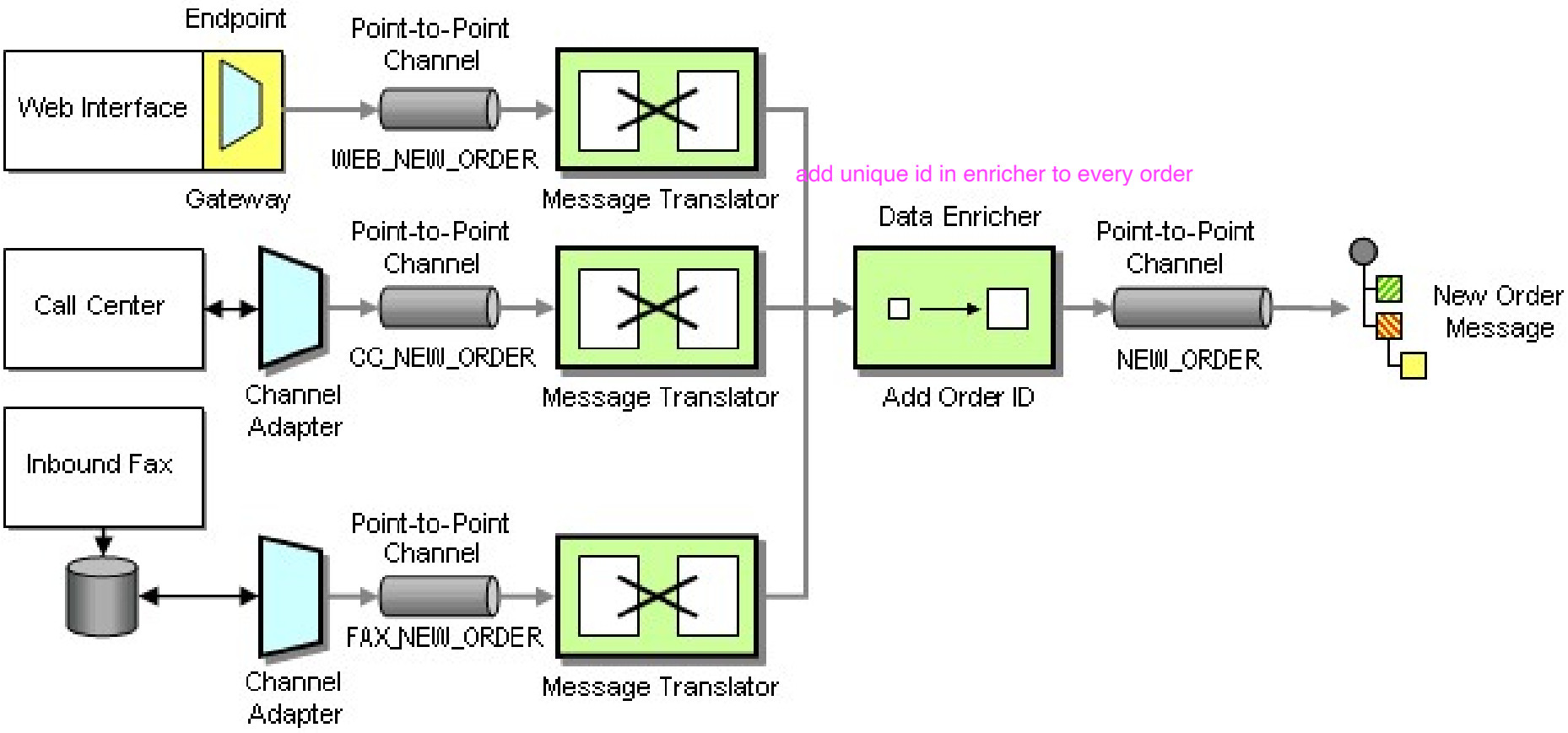


Orders can contain multiple items

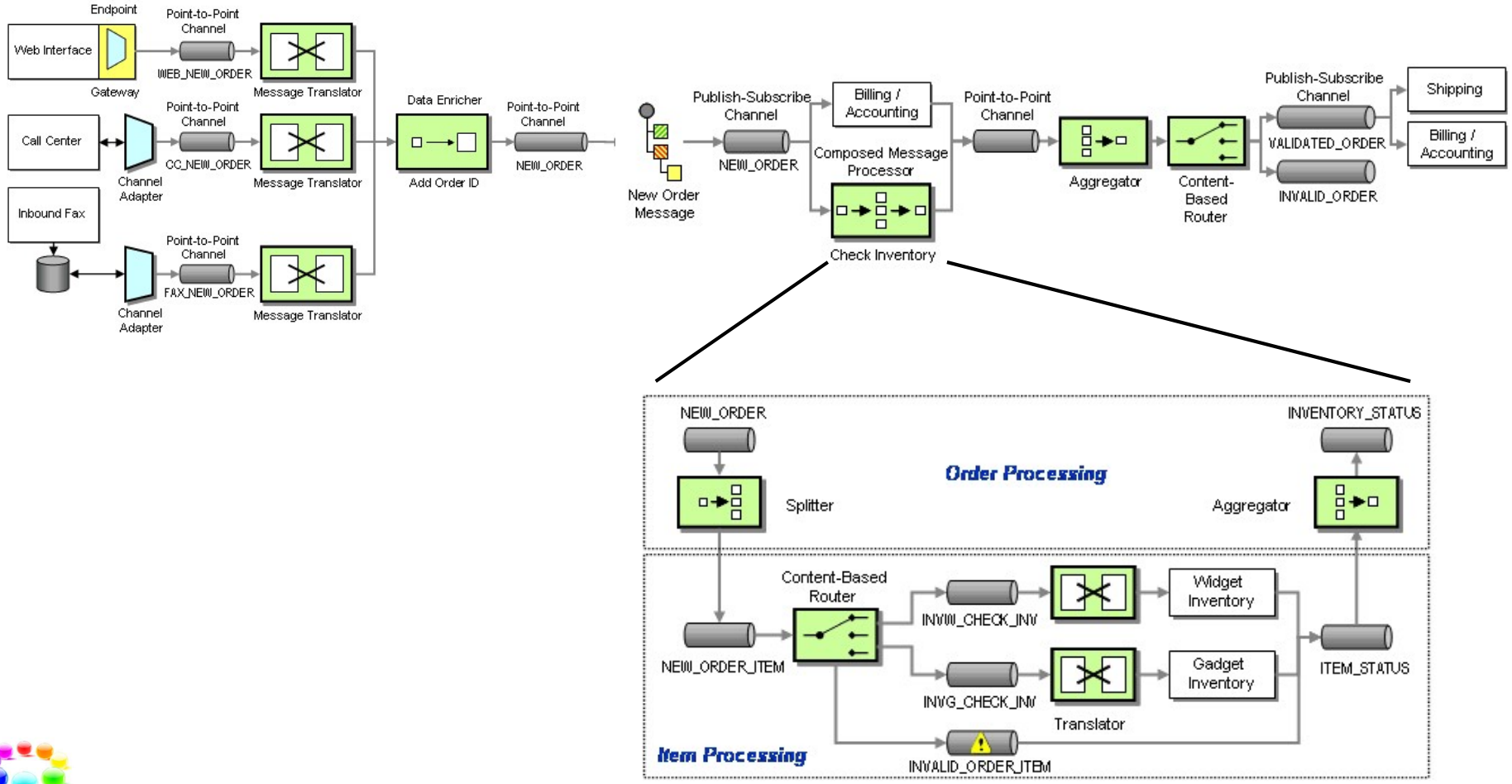
1. Correlation: which messages belong together? We need an unique order ID
2. Completeness: how do we know that all messages are received? Count
3. Aggregation algorithm: how do we combine the individual messages into one result message? Append based on order ID



Add an unique order ID



Result so far



Connecting the parts of knowledge with the wholeness of knowledge

1. By externalizing integration logic from the application into an ESB, the applications become more loosely coupled.
2. Integration logic can be designed with a basic set of integration patterns.

3. **Transcendental consciousness** is the field that connects everything together.
4. **Wholeness moving within itself:** In Unity Consciousness, one realizes that everything else in creation is just an expression of ones own Self.