1132203/1142202 - PASTI

# Introduction to Messaging System

Week 13, Session 1
Tahun Ajaran 2023/2024

Lecturer : RCH, OPH
Teaching Assistant: MAL

INSTITUT TEKNOLOGI DEL
Jl. Sisingamangaraja, Sitoluama Laguboti
Kab. Tobasa SUMUT, Indonesia. 22381
https://www.del.ac.id/

# Outlines

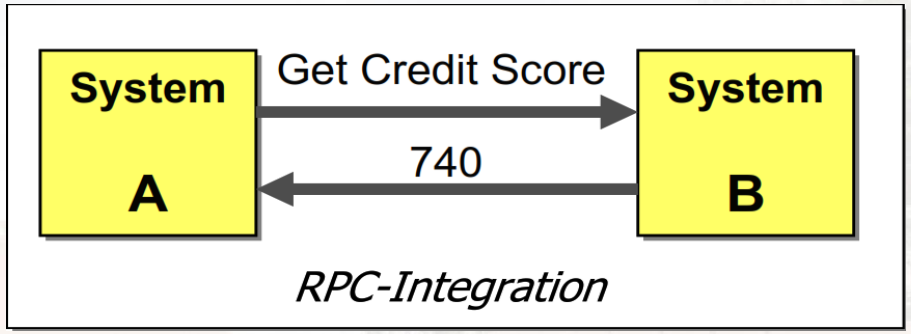
- Integration Challenges
- Thinking Asynchronously
- Asynchronous Messaging Architectures
- Message-Oriented Integration
- Integration Pattern

# Integration Challenges

- Networks are slow
- Networks are unreliable
- No two applications are alike
- Change is Inevitable
- Inherently large-scale and complex
- Limited control over entities / applications
- Far-reaching implications, business critical
- Intertwined with corporate politics
- Few standards exist, still evolving

# Loosely Coupling

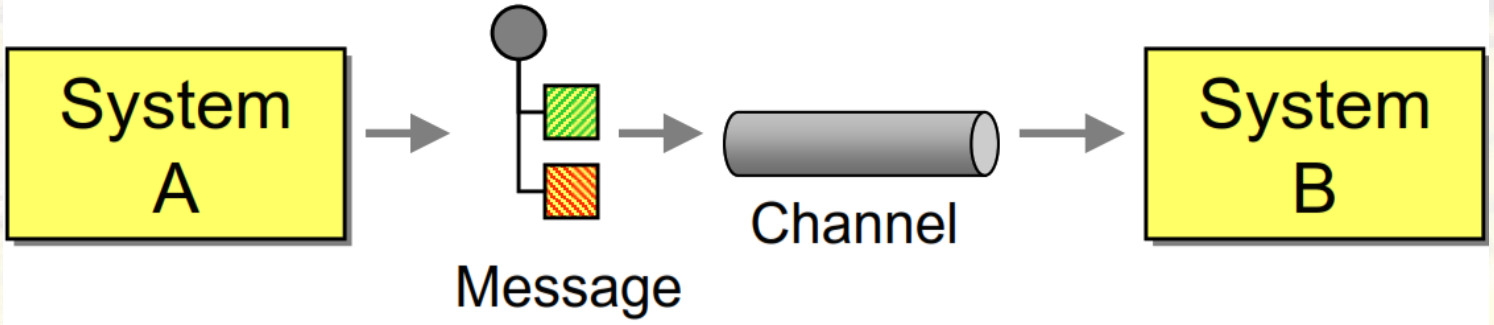
- Coupling = Measure of dependencies between applications:
  - Technology Dependency
  - Location Dependency
  - Temporal Dependency
  - Data Format Dependency



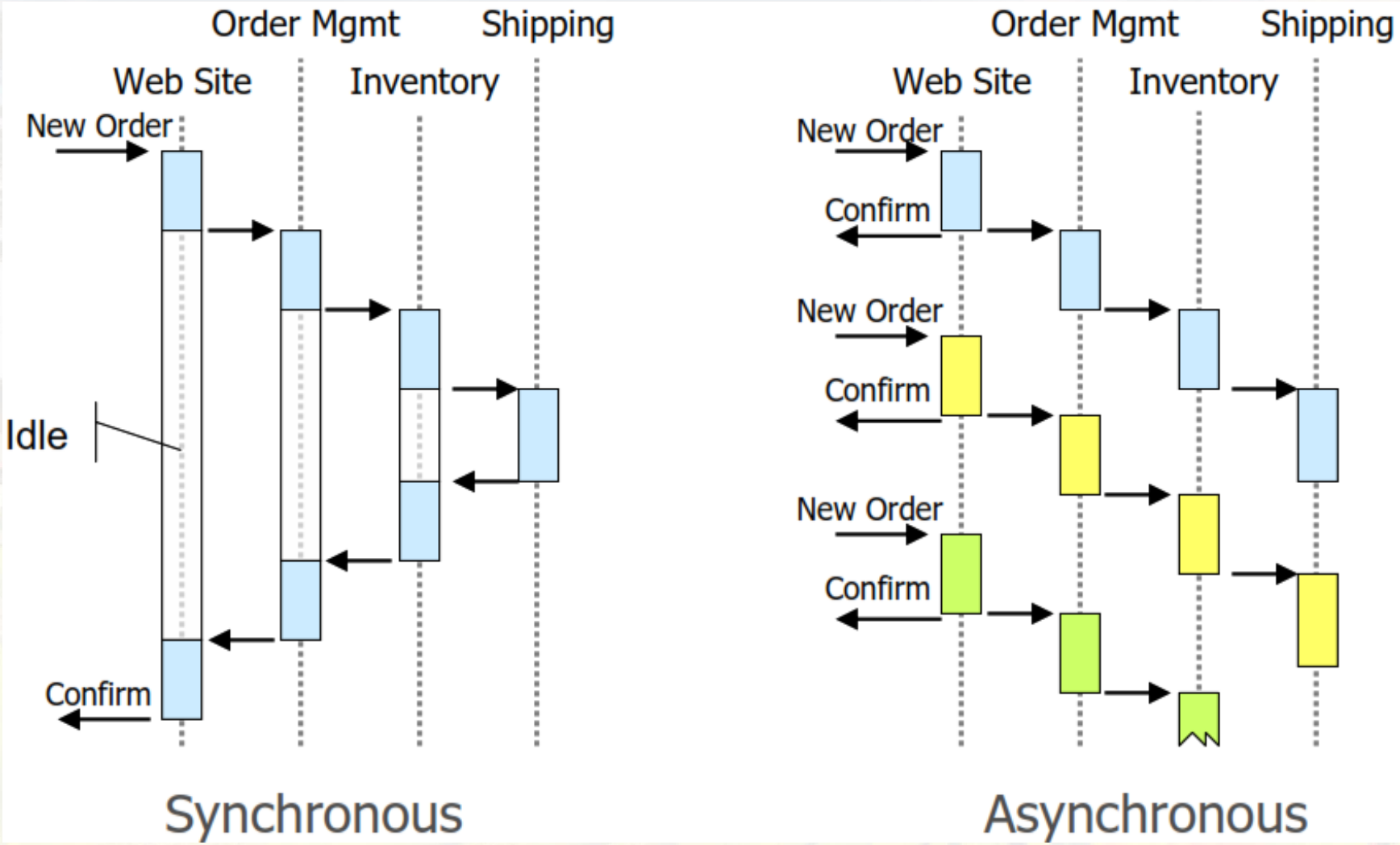

- Waldo et al, 1994:
  – " Objects that interact in a distributed system need to be dealt with in ways that are intrinsically different from objects that interact in a single address space

# Message-Oriented Middleware

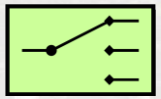
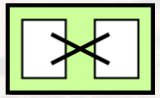
- Channels are separate from applications
- Remove location dependencies
- Channels are asynchronous & reliable
- Remove temporal dependencies
- Data is exchanged in self-contained messages
- Remove data format dependencies



# Thinking Asynchronously



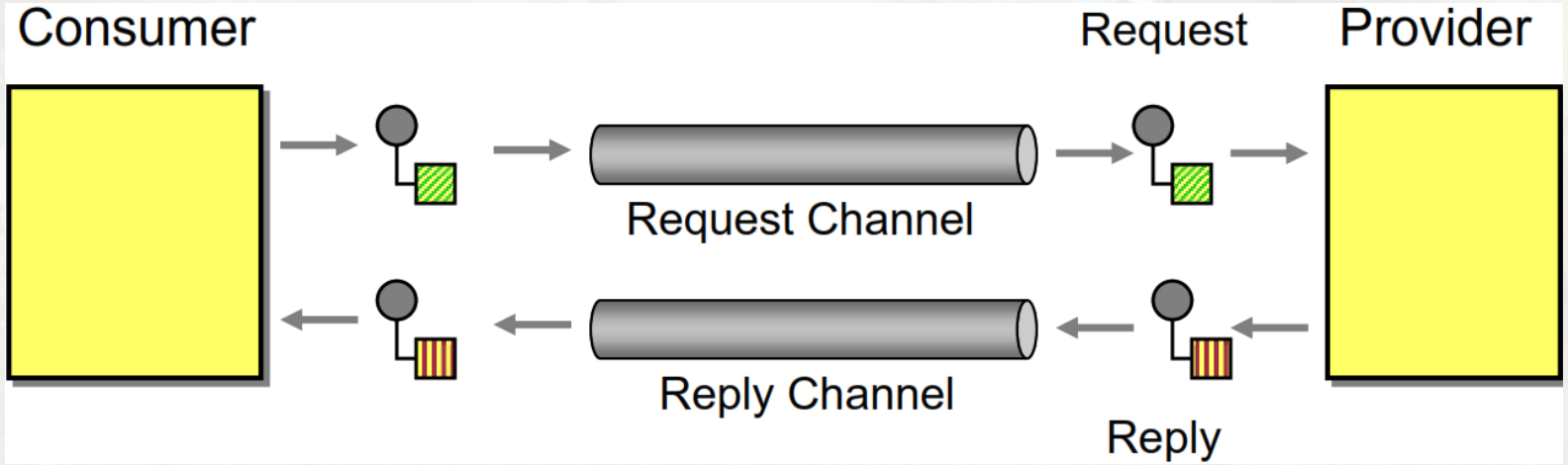
# Integration Patterns

1. Transport messages ⇨ Channel Patterns 
2. Design messages ⇨ Message Patterns 
3. Route the message to the proper destination ⇨ Routing Patterns 
4. Transform the message to the required format ⇨ Transformation Patterns 
5. Produce and consume messages ⇨ Endpoint Patterns 

6. Manage and Test the System ⇨ Management Patterns 

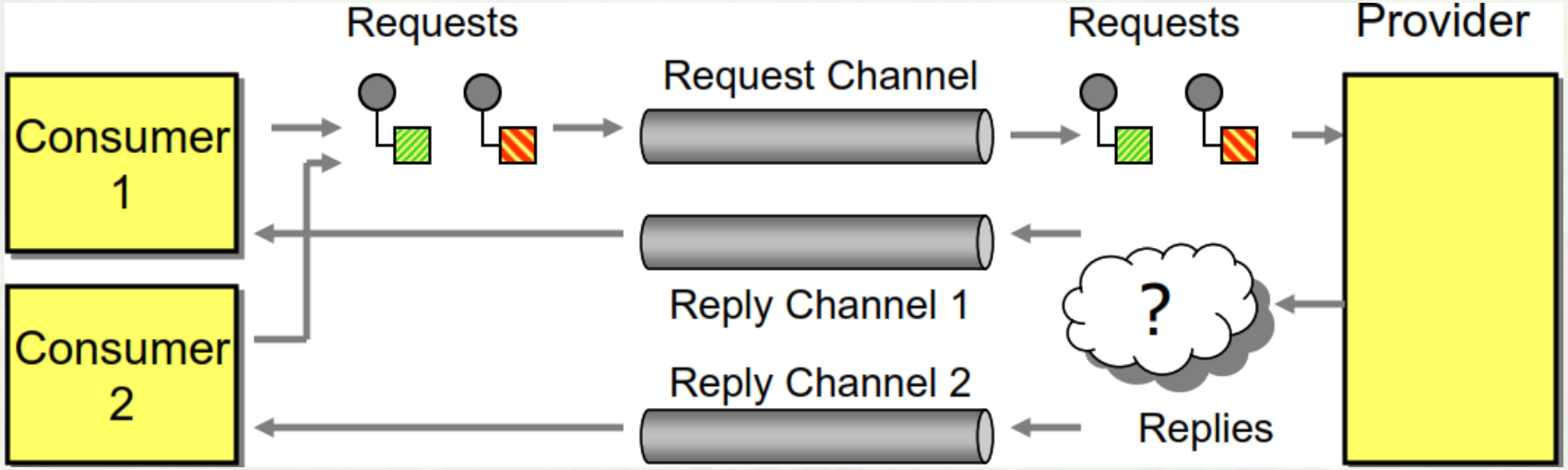
# Asynchronous

- Service Provider and Consumer
- Request-Reply(similar to RPC)
- Two asynchronous Point-To-Point Channels
- Channels are unidirectional
- Separate request and response messages



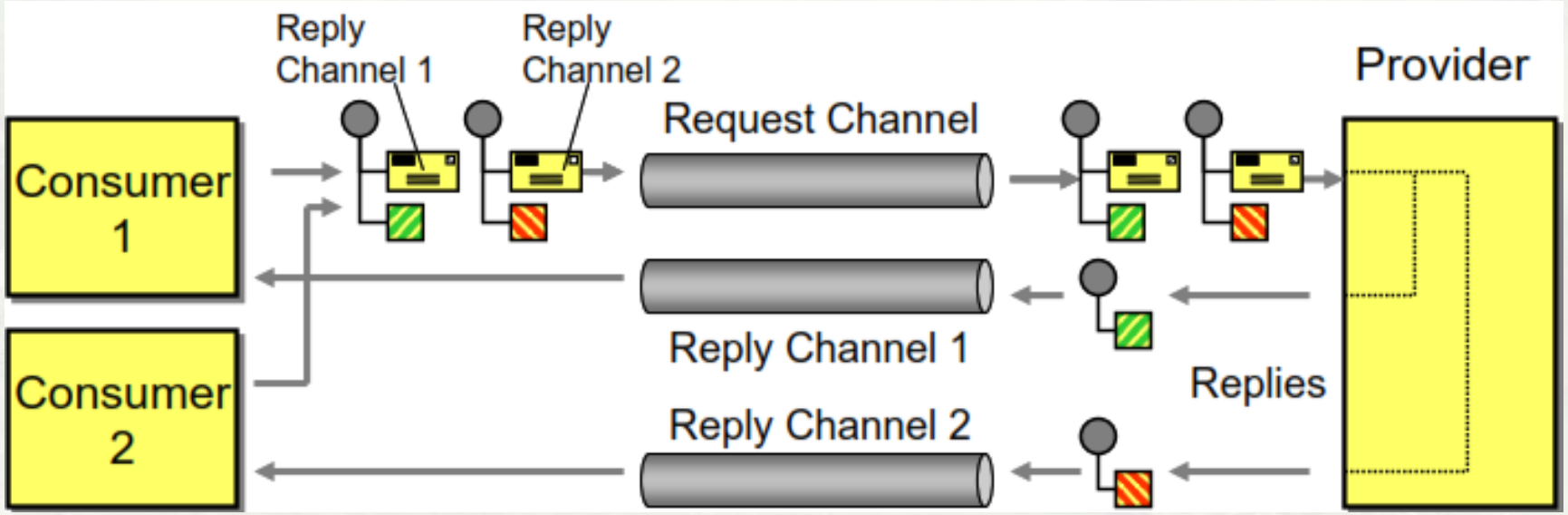
# Multiple Consumers

- Each consumer has its own reply queue
- How does the provider know where to send the reply?
  - Could send to all consumers -> very inefficient
  - Hard code -> violates principle of service



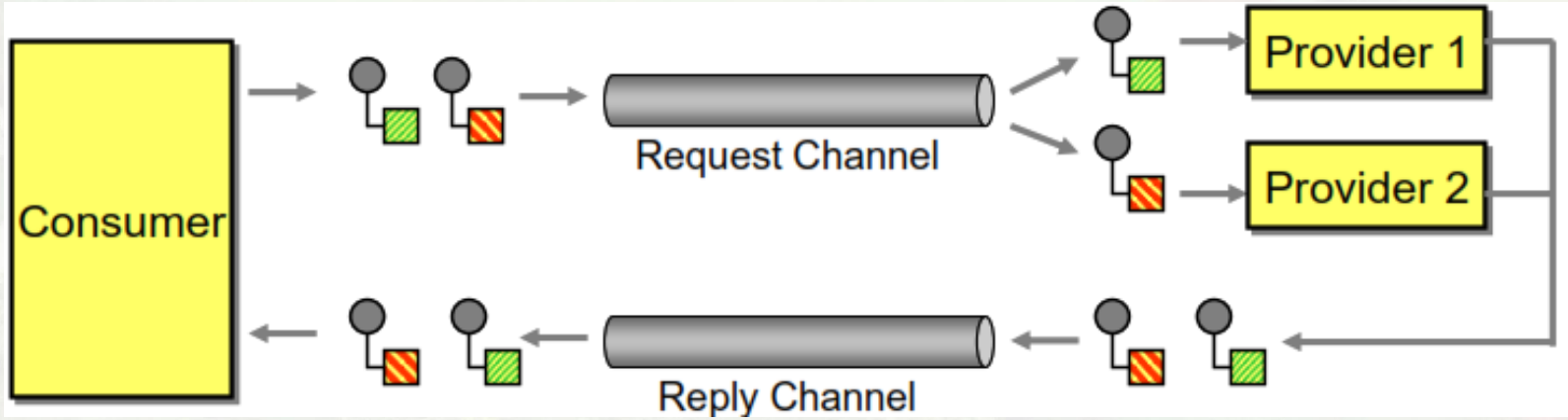
# Pattern: Return Address

- Consumer specifies Return Address
- Service provider sends reply message to specified channel
- Return Address can point to a component different from the consumer -> chaining



# Multiple Service Providers

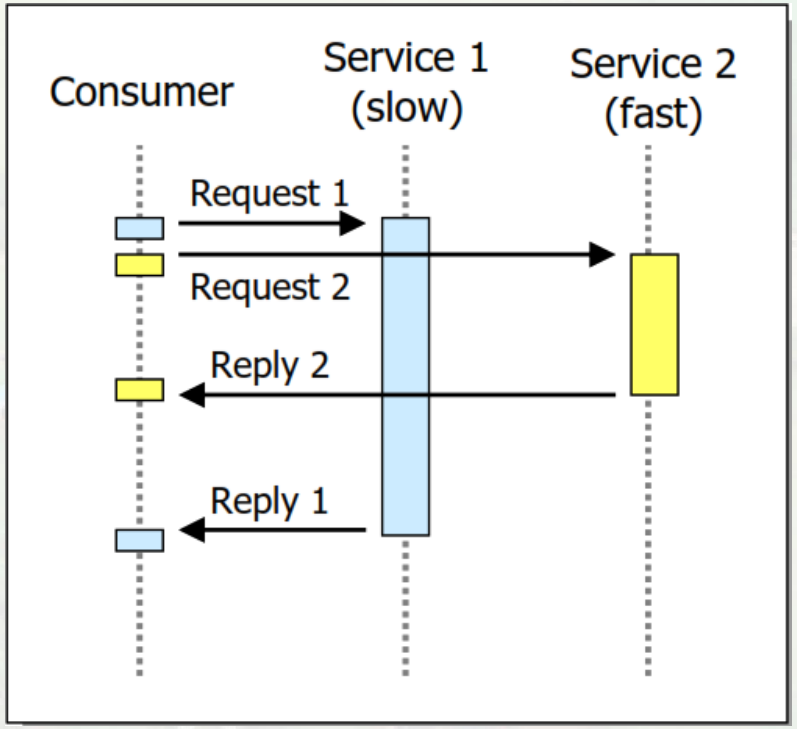
- Request message can be consumed by more than one service provider
- Point-to-Point Channel supports Competing Consumers, only one service receives each request message
- Channel queues up pending requests



# Multiple Service Providers

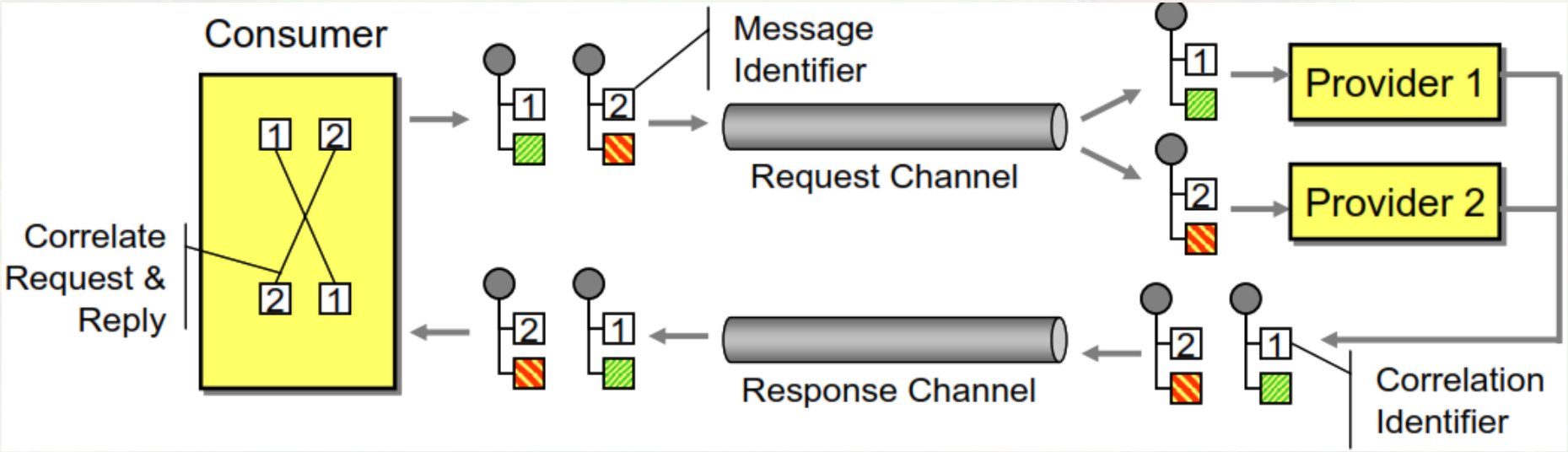
- Messages can be processed by different consumers
  - Competing Consumers (load balancing)
  - Content-Based Router
- This causes messages to get out of sequence

- How to match request and reply messages?
  - Only send one request at a time very inefficient
  - Rely on natural order bad assumption



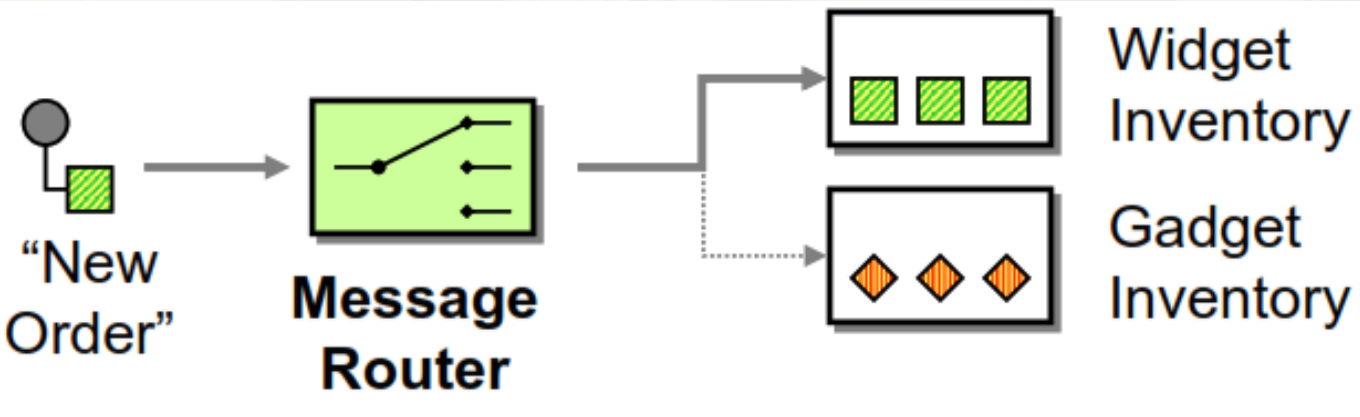
# Pattern: Correlation

- Equip each message with a unique identifier
  - Message ID (simple, but has limitations)
  - GUID (Globally Unique ID)
  - Business key (e.g. Order ID)
- Provider copies the ID to the reply message
- Sender can match request and response



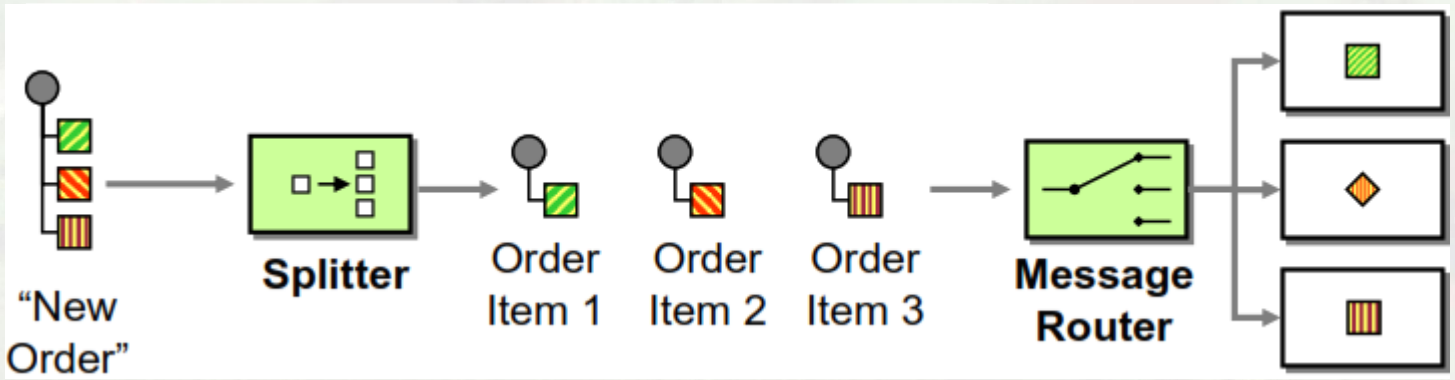
# Routing Pattern: Message Router

- How can we decouple individual processing steps so that messages can be passed to different components depending on some conditions?
  - Different channels depending on message content, run-time environment (e.g. test vs. production), …
  - Do not want to burden sender with decision (decoupling)
- Use a special component, a Message Router, to route messages from one channel to a different channel.



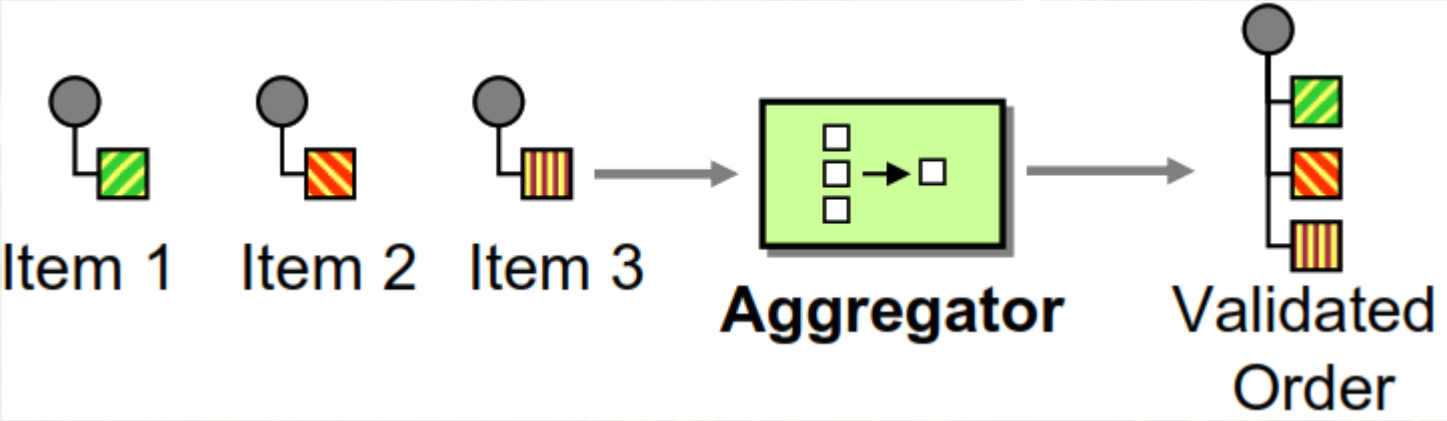
# Routing Pattern: Splitter

- How can we process a message if it contains multiple elements, each of which may have to be processed in a different way?
  - Treat each element independently
  - Need to avoid missing or duplicate elements
  - Make efficient use of network resources
- Use a Splitter to break out the composite message into a series of individual messages, each containing data related to one item



# Routing Pattern: Aggregator

- How do we combine the results of individual, but related messages back into a single message?
  - Responses may be out of sequence
  - Responses may be delayed
- An Aggregator manages the reconciliation of multiple, related messages into a single message
  - Stateful component



# Routing Pattern: Aggregator

- Correlation
  - Which incoming messages belong together?
- Completeness Condition
  - When are we ready to publish the result message?
  - Wait for all
  - Time out (absolute, incremental)
  - First best
  - Time box with override
  - External event
- Aggregation Algorithm
  - How do we combine the received messages into a single result message?
    - Concatenate data for later analysis
    - Single best answer
    - Condense data (e.g., average)

# Source

**Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions**
By Gregor Hohpe, Bobby Woolf :ISBN-10: 0-321-20068-3

**https://www.enterpriseintegrationpatterns.com/MessagingComponentsIntro.html**