Message Routing

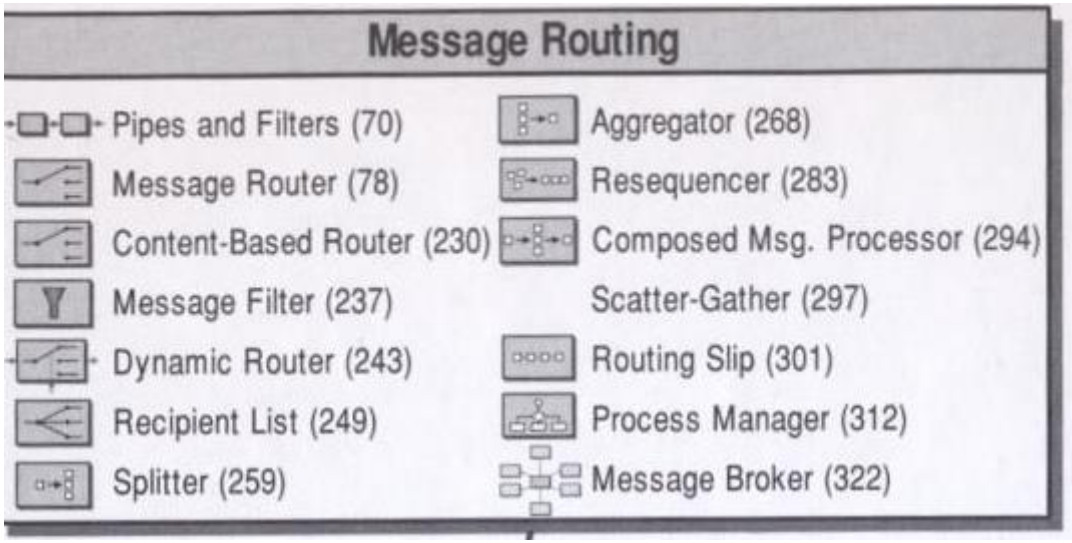
Systems Integration

PBA Softwareudvikling/BSc Software Development

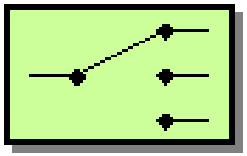
Tine Marbjerg

Fall 2018

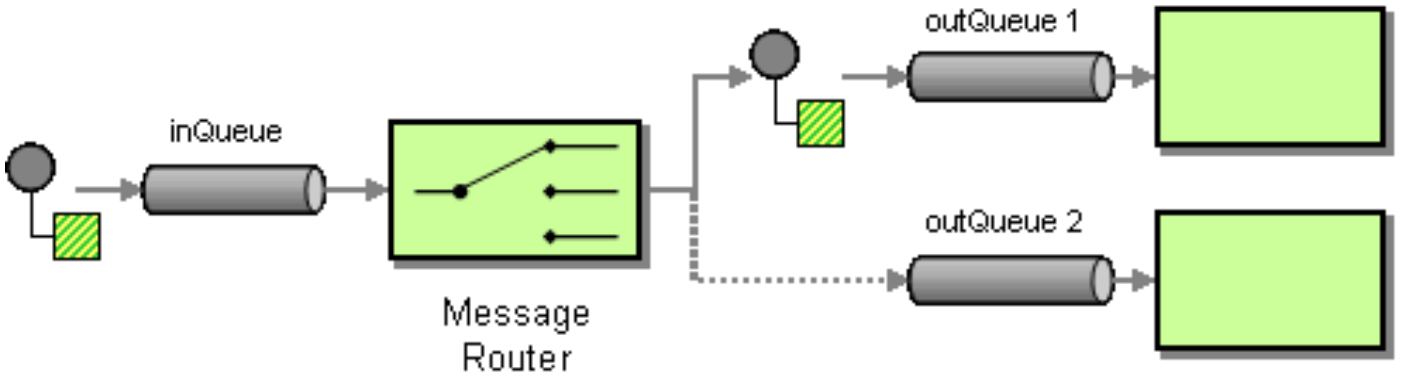
Overview of Message Routing in EIP chap 7



Message Router (78)



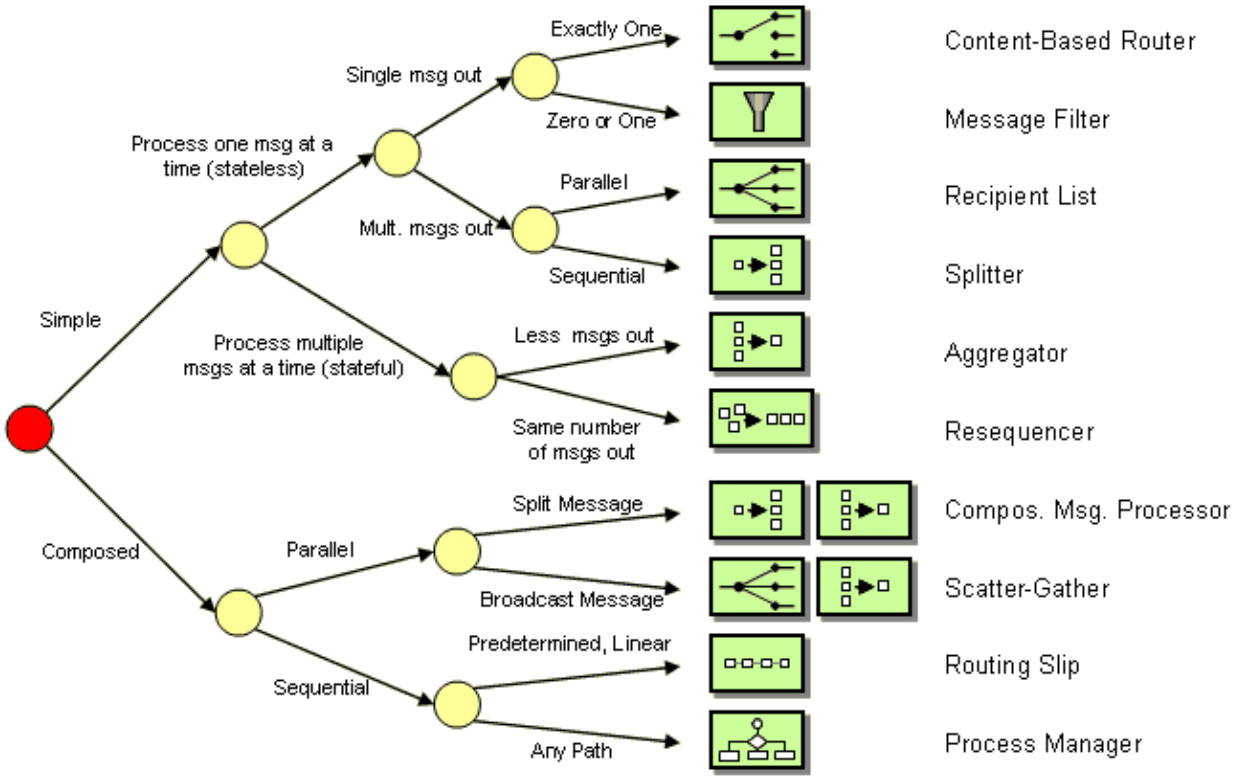
- Decouples a message source from the ultimate destination of the message
  - One input channel
  - One or more output channels
- Is the filter component in the Pipes and Filters architecture



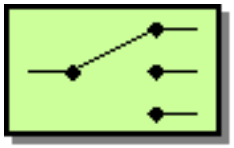
Right Router for the Right Purpose

How to read the diagram? Examples:

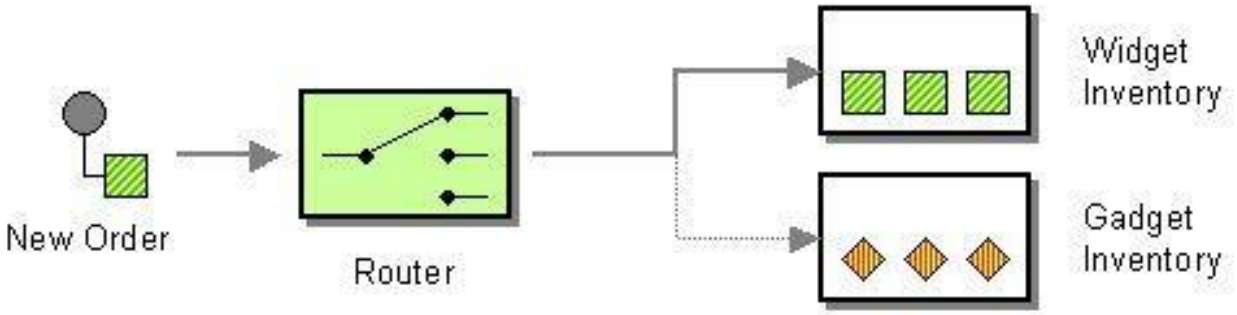
- If you are looking for a simple routing pattern that consumes one message at a time but publishes multiple messages in sequential order, you should use a Splitter.
- The diagram also illustrates how closely the individual patterns are related. For example, a *Routing Slip* and a *Process Manager* solve similar problems while a *Message Filter* does something rather different.



Content-Based Router (230)

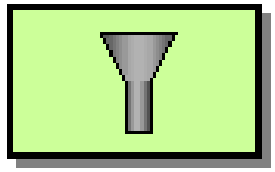


- Use a Content-Based Router to route each message to the correct destination based on the content of the message

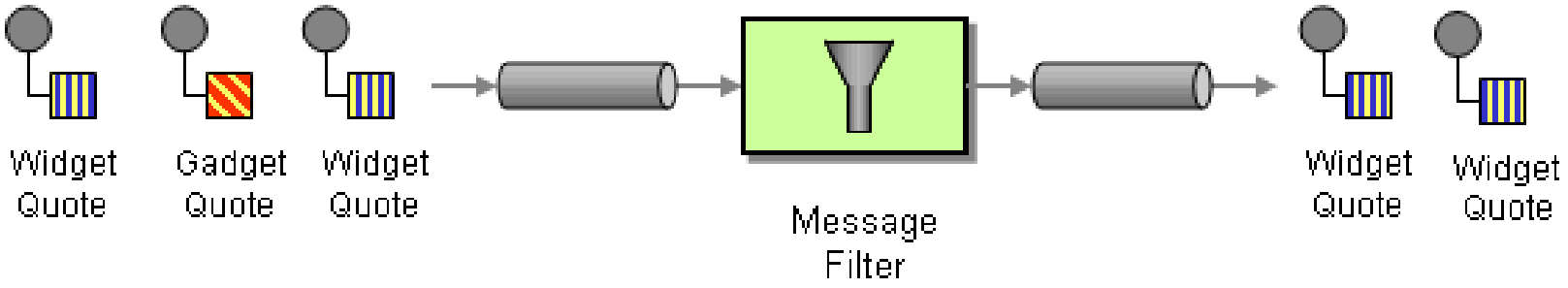


- Sometimes an implementation of a single logical function is spread across multiple physical systems, like in example above, but sending application doesn't need to know that.
- The *Content-Based Router* examines the message content and routes the message onto a different channel based on data contained in the message. The routing can be based on a number of criteria such as existence of fields, specific field values etc.

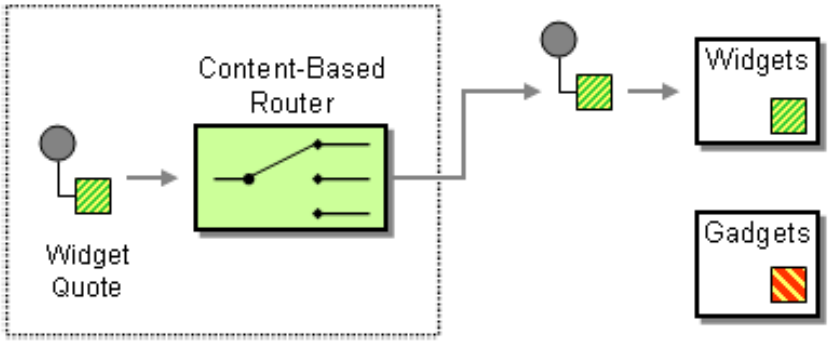
Message Filter (237)



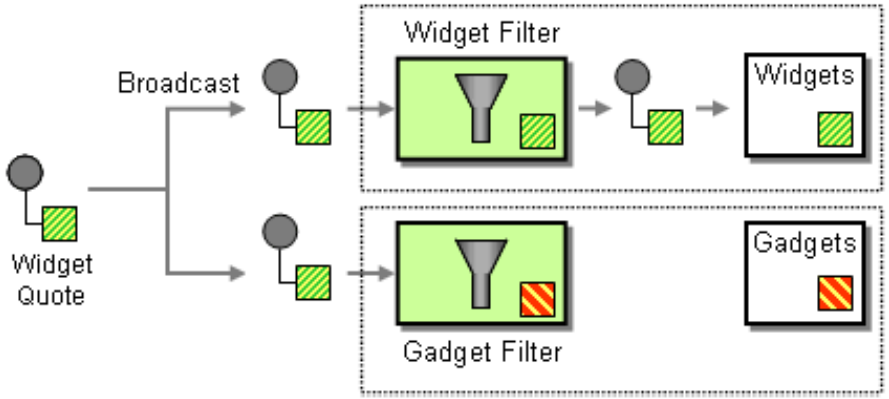
- How can a component avoid receiving uninteresting messages?
- Use a special Message Router called a Message Filter to remove undesired messages from a channel based on a set of criteria



Content-Based Router or Message Filter?



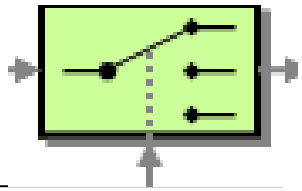
Option 1: Using a Content-Based Router



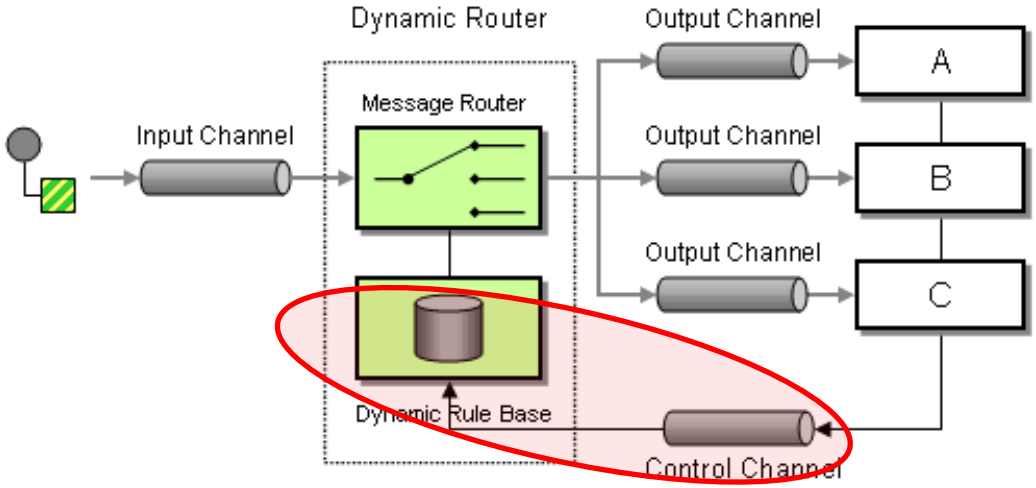
Option 2: Using a broadcast channel and a set of Message Filters

Content-Based Router	**Pub-Sub Channel with *Message Filter***
Exactly one consumer receives each message.	More than one consumer can consume a message.
Central control and maintenance -- predictive routing.	Distributed control and maintenance -- reactive filtering.
Router needs to know about participants. Router may need to be updated if participants are added or removed.	No knowledge of participants required. Adding or removing participants is easy.
Often used for business transactions, e.g. orders.	Often used for event notifications / informational messages.
Generally more efficient with queue-based channels.	Generally more efficient with publish-subscribe channels.

Dynamic Router (243)

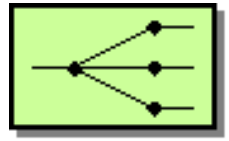


- How can you avoid the dependency of the router on all possible destinations while maintaining its efficiency?
- Use a Dynamic Router that can self-configure based on special configuration messages from participating destinations

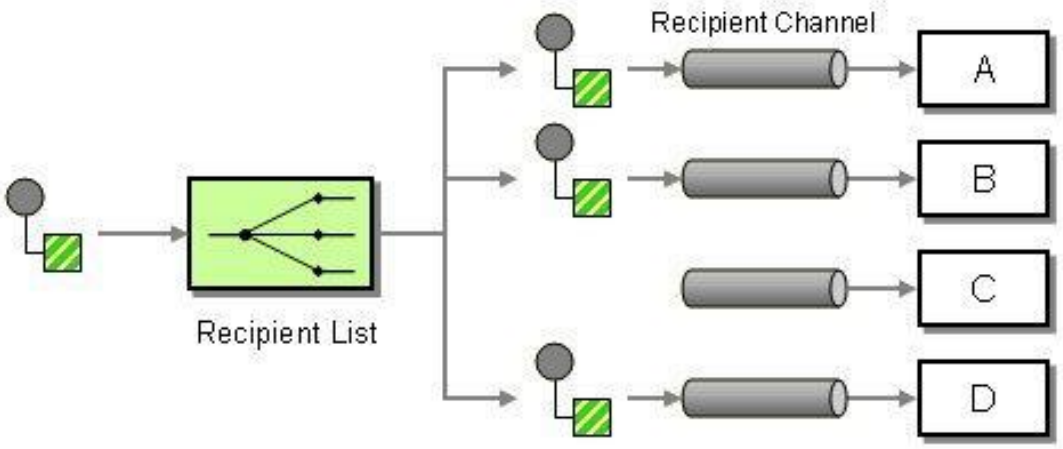


- In basic scenario, each participant announces it existence and routing preferences to the Dynamic Router at startup time
  - Participants must know control queue
  - Dynamic router must store rules in persistent way
  - Because recipients are independent from each other, the dynamic router has to deal with rule conflicts

Recipient List (249)

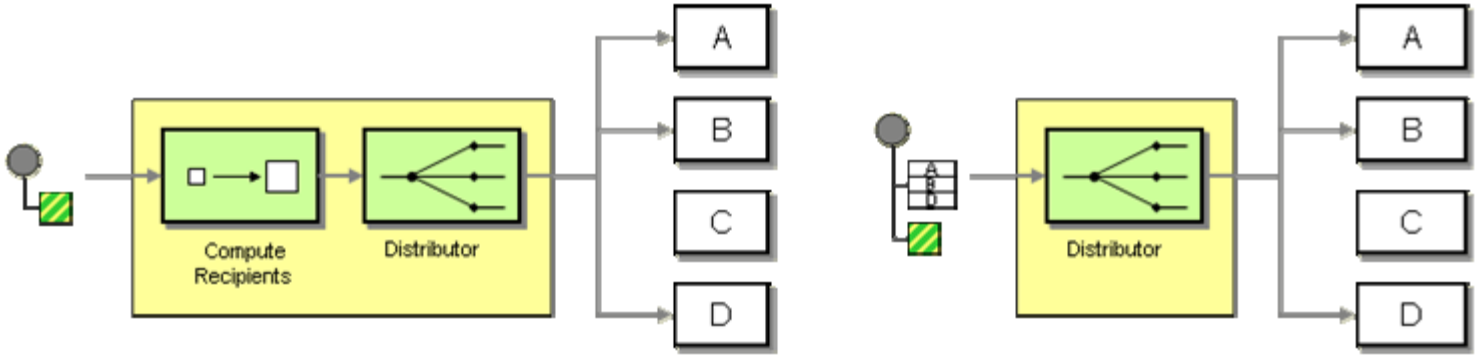


- How do we route a message to a dynamic list of recipients?
- Define a channel for each recipient. Then use a Recipient List to inspect an incoming message, determine the list of desired recipients, and forward the message to all channels associated with the recipients in the list.



Recipient List – How it works

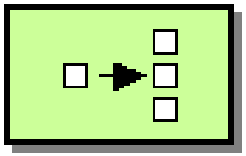
1. Compute a list of recipients
2. Traverse the list and send a copy of the received message to each recipient



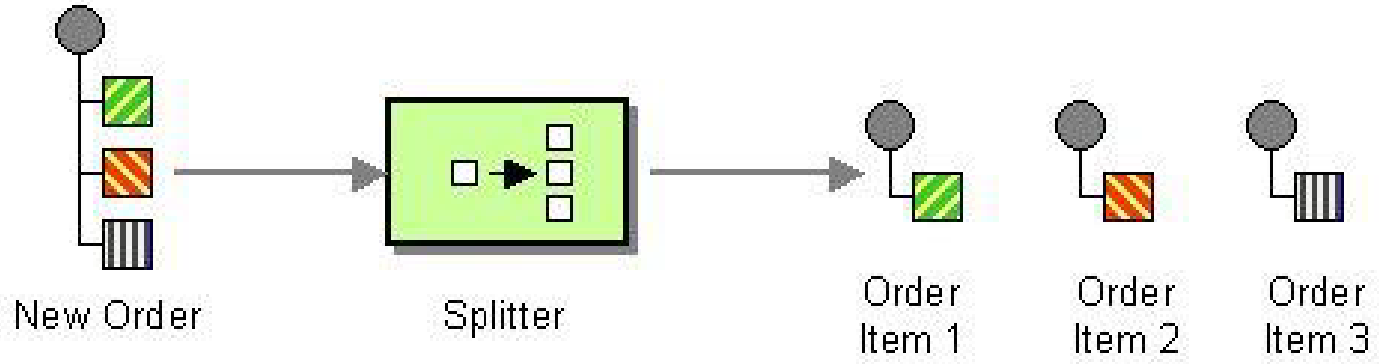
OBS! A Recipient list can compute the recipients (left) or have another (external) component provide a list (right)

A robust implementation must be able to process the incoming message, but only "consume" it after all outbound messages have been successfully sent.

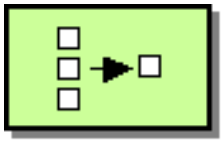
Splitter (259)



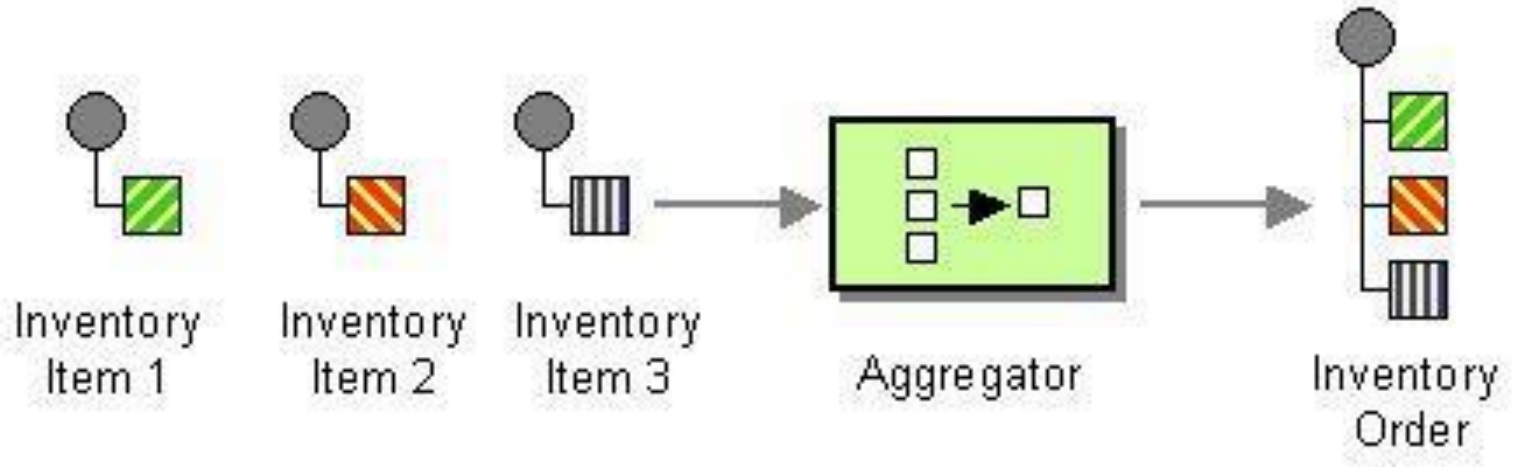
- How can we process a message if it contains multiple elements, each of which may have to be processed in a different way?
- Use a Splitter to break out the composite message into a series of individual messages, each containing data related to one item



Aggregator (268)

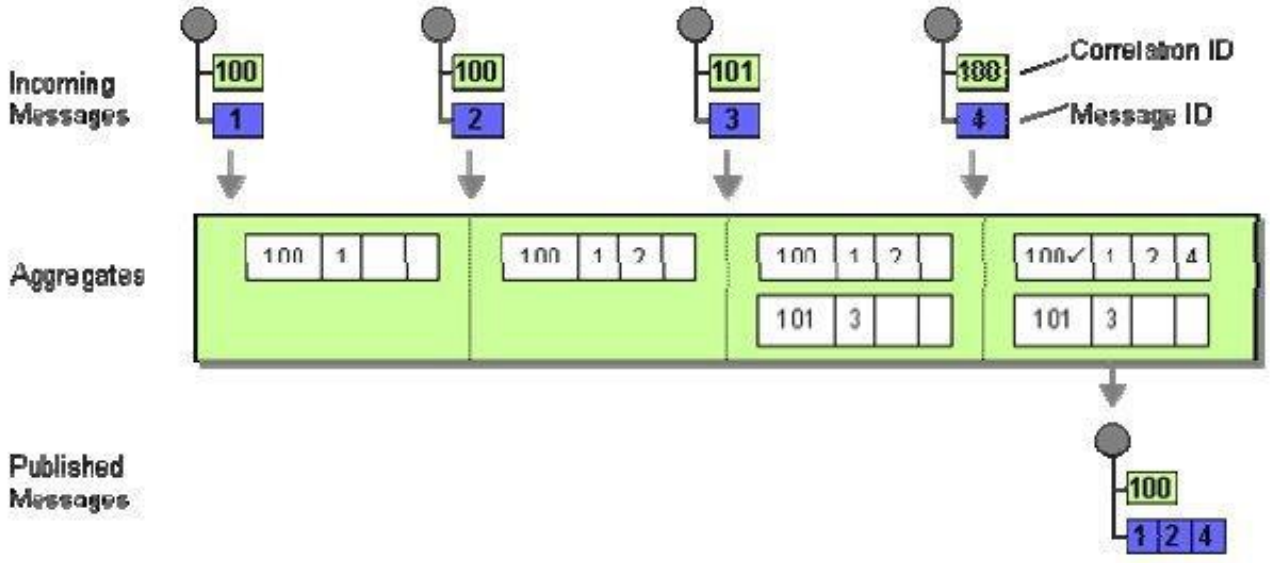


- How do we combine the results of individual, but related messages so that they can be processed as a whole?
- Use a stateful filter, an Aggregator, to collect and store individual messages until it receives a complete set of related messages. Then, the Aggregator publishes a single message distilled from the individual messages



Design of Aggregator

1. Correlation: Which messages belong together
2. Completeness Condition: When to publish the result
3. Aggregation Algorithm: How to combine the received messages into a single result message

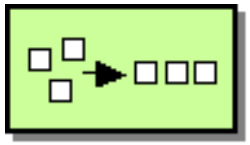


Aggregation Algorithm

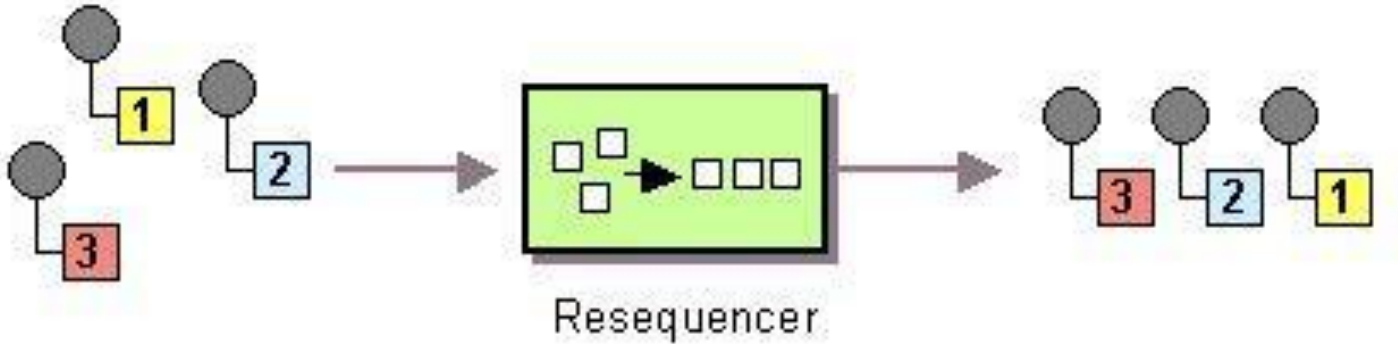
Possible completeness conditions:

1. Wait for all – all responses received
2. Timeout – wait only specified length of time
3. First Best – Wait until the first (fastest) response is received
4. Timeout with Override – wait specified time of length or until message with preset minimum score is received
5. External Event – typically external business event

Resequencer (127)

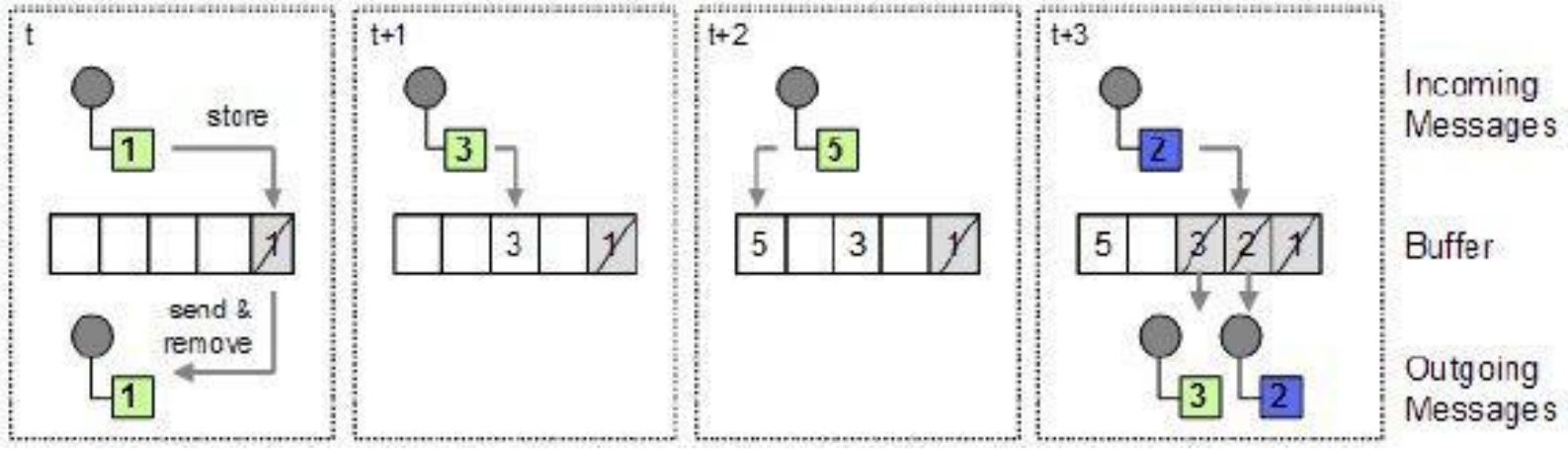


- How can we get a stream of related but out-of-sequence messages back into the correct order?
- Use a stateful filter, a Resequencer, to collect and re-order messages so that they can be published to the output channel in a specified order



Internal Operation of the Resequencer

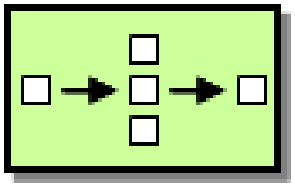
- Out-of-sequence messages stores (buffer) until all the "missing" messages are received
- When the buffer contains a consecutive sequence it sends this to the output channel
- Example: in-coming sequence: 1, 3, 5, 2



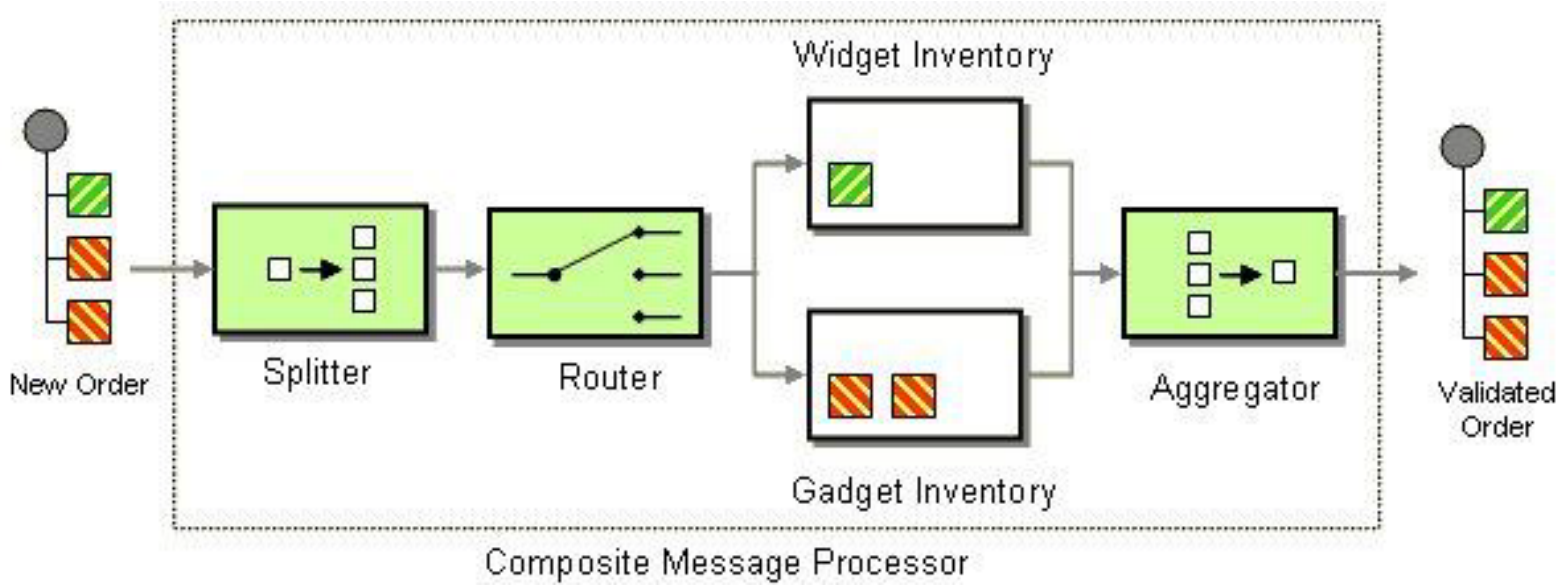
Message Router Variants

Pattern	# of Msgs Consumed	# of Messages Published	Stateful?	Comment
Content-Based Router	1	1	No (mostly)	
Message Filter	1	0 or 1	No (mostly)	
Recipient List	1	multiple (incl. 0)	No	
Splitter	1	multiple	No	
Aggregator	multiple	1	Yes	
Resequencer	multiple	multiple	Yes	Publishes same number it consumes

Composed Message Processor (294)

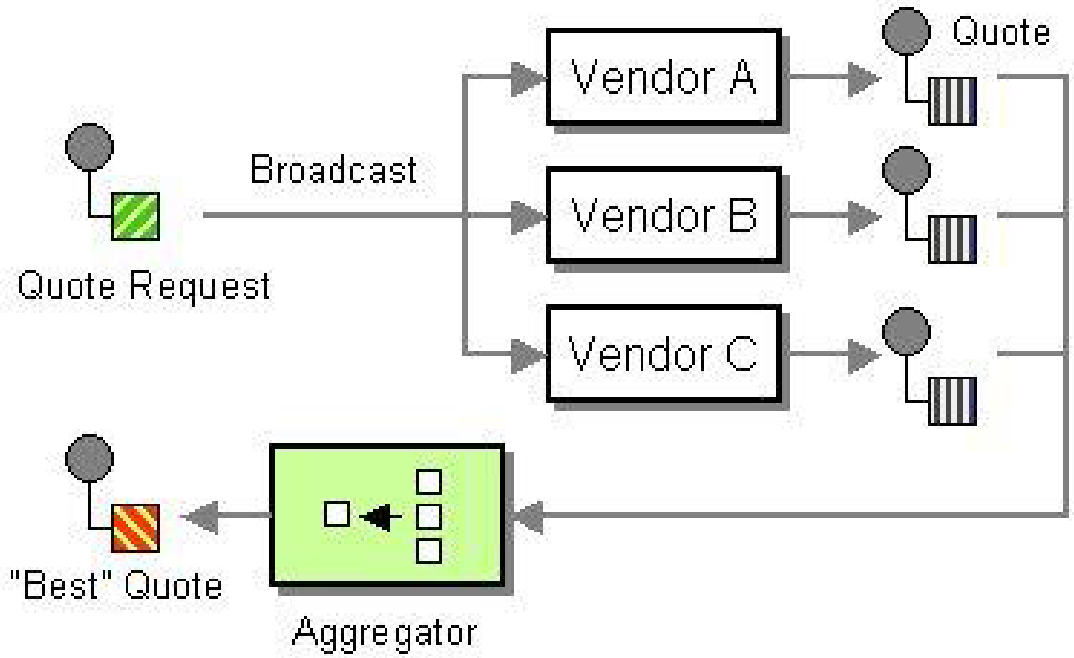


= Splitter + Router + Aggregator



Scatter-Gather (297)

= Broadcast+ Aggregator



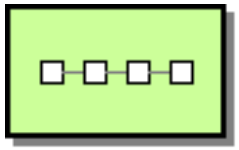
Scatter-Gather Variants

- Distribution via Recipient List
- Auction-style via Publish.subscribe Channel

Notice that solution is different from Composed-Message Processor:

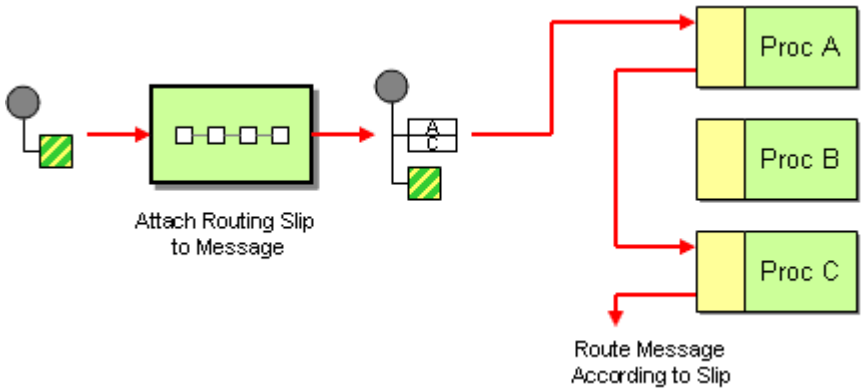
Instead of splitter, we broadcast the complete message to all involved parties.

Routing Slip (301)

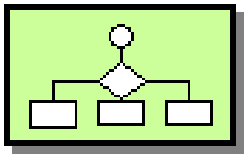


- How do we route a message consecutively through a series of processing steps when the **sequence of steps is not known at design time and may vary for each message**?

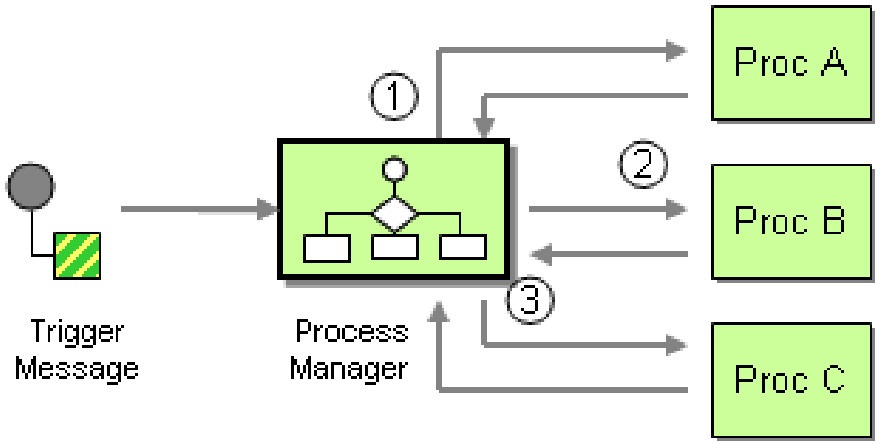
- Attach a *Routing Slip* to each message, specifying the sequence of processing steps. Wrap each component with a special message router that reads the *Routing Slip* and routes the message to the next component in the list.



Process Manager (312)



- How do we route a message through multiple processing steps when **the required steps may not be known at design time and may not be <u>sequential</u>**?
- Use a central processing unit, a Process Manager, to maintain the state of the sequence and determine the next processing step based on intermediate results



- Process modeling is design of workflow activity / business process management

Process Manager - alternatives

Multiple Content-Based Routers

Pro – maximum flexibility

Con – routing logic is spread across many routing components

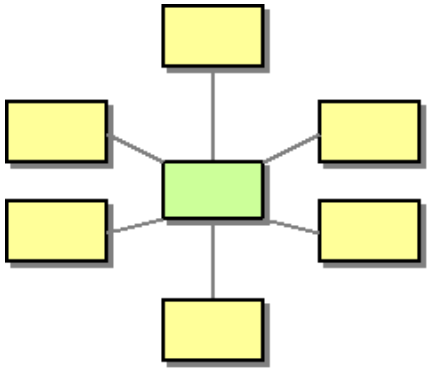
Routing Slip

Pro -Central point of control (computing message path up front)

Con – Cannot reroute based on intermediate results or execute multiple steps simultaneously

Message Broker (322)

- How can you decouple the destination of a message from the sender and maintain central control over the flow of messages?
  - Use a central Message Broker that can receive messages from multiple destinations, determine the correct destination, and route the message to the correct channel.



- It's a hub-and-spoke architectural style
  - *Message Broker* isn't monolithic component. Internally, it uses the design patterns presented in Routing chapter
  - RabbitMQ is message broker