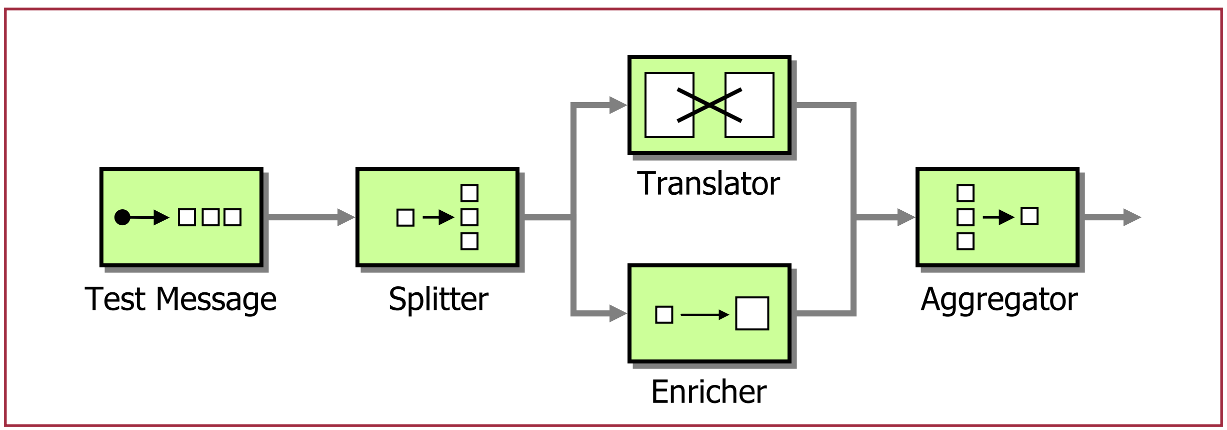
Enterprise Integration Patterns

Asynchronous Messaging Architectures in Practice



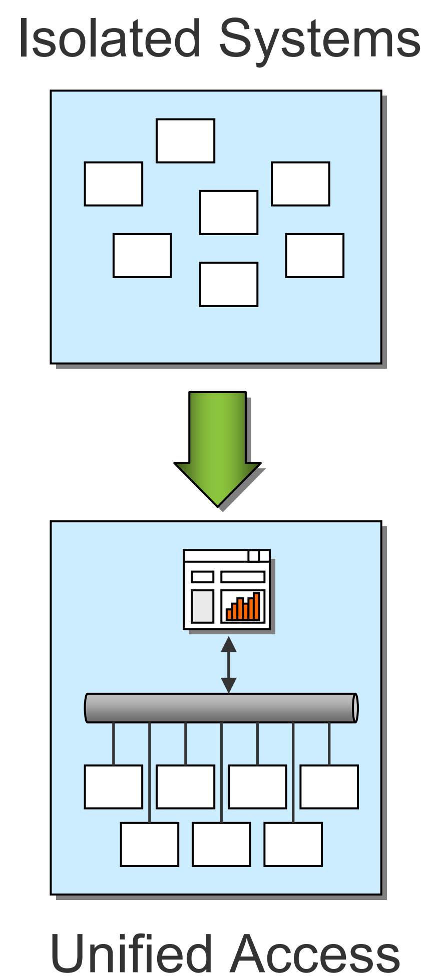
Gregor Hohpe

ThoughtWorks
The art of heavy lifting.

JAOO conference 2003

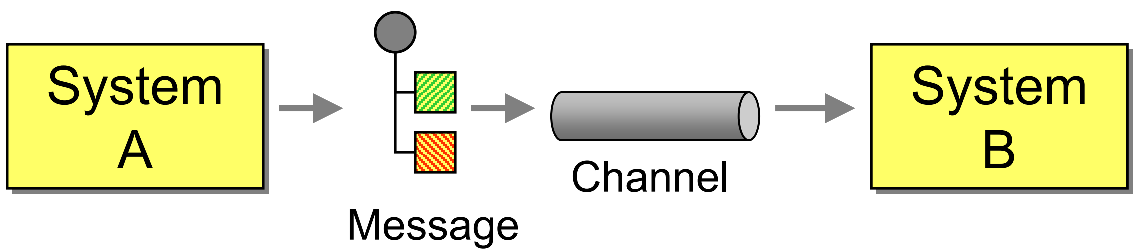
Integration Challenges

- Users want to execute business functions that span multiple applications
- Requires disparate applications to be connected to a common integration solution
- However:
  - Networks are slow
  - Networks are unreliable
  - No two applications are alike
  - Change is Inevitable



Message-Oriented Middleware

Message-oriented architectures provide *loose coupling* and reliability



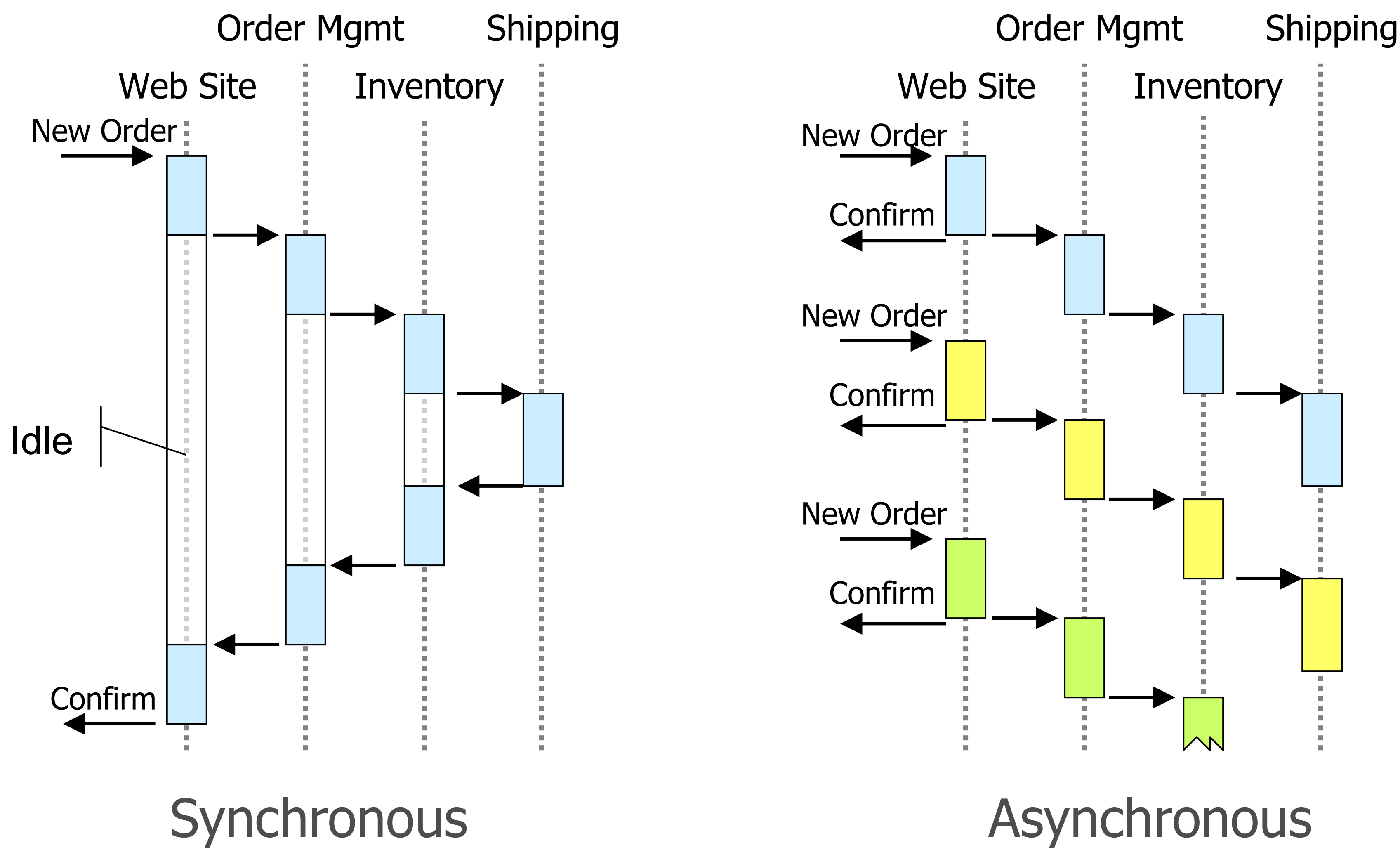
- Channels are separate from applications
- Channels are asynchronous & reliable
- Data is exchanged in self-contained messages

➢ Remove location dependencies

➢ Remove temporal dependencies

➢ Remove data format dependencies

Thinking Asynchronously



Many Products & Implementations

- Message-oriented middleware (MOM)
  - IBM WebSphere MQ
  - Microsoft MSMQ
  - Java Message Service (JMS) Implementations
- EAI Suites
  - TIBCO, WebMethods, SeeBeyond, Vitria, ...
- Asynchronous Web services
  - WS-ReliableMessaging, ebMS
  - Sun's Java API for XML Messaging (JAXM)
  - Microsoft's Web Services Extensions (WSE)

HOT

➡ The Underlying Design Principles Are the Same!

Catalog of 65 Patterns

1. Request-Reply Example

2. Order Management Example

Message Construction

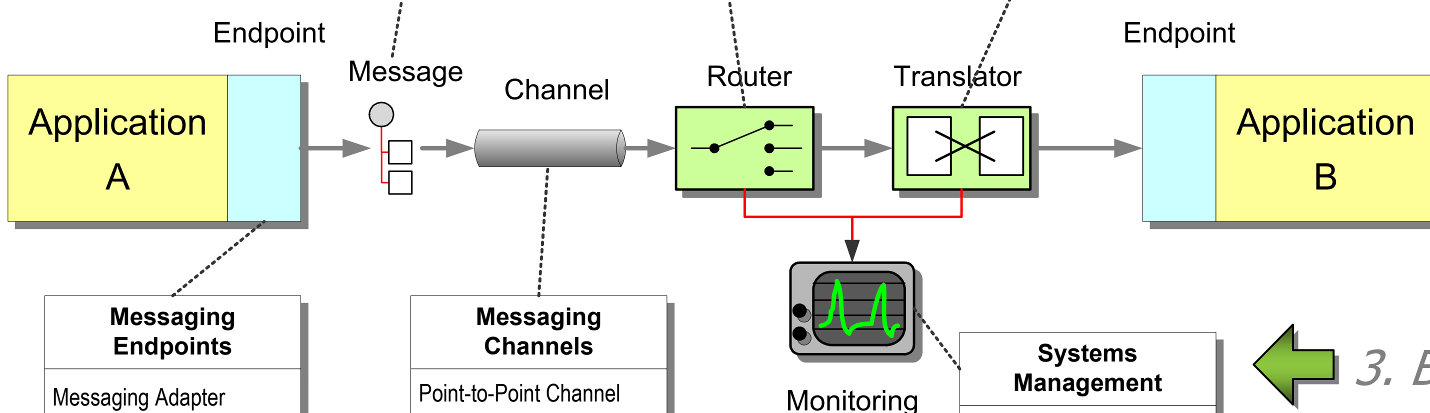
- Command Message
- RPC Message
- Query Message
- Document Message
- Event Message
- Reply Message
- Return Address
- Correlation Identifier
- Message Sequence
- Message Expiration
- Canonical Data Model
- Format Indicator

Message Routing

- Content-Based Router
- Message Filter
- Recipient List
- Splitter
- Aggregator
- Resequencer
- Distribution w. Aggr. Resp.
- Auction
- Routing Table
- ProcessManager

Message Transformation

- Data Enricher
- Content Filter
- Check Luggage



Messaging Endpoints

- Messaging Adapter
- Polling Consumer
- Event-Driven Consumer
- Transactional Client
- Competing Consumers
- Message Dispatcher
- Message Selector
- Idempotent Receiver
- Messaging Mapper

Messaging Channels

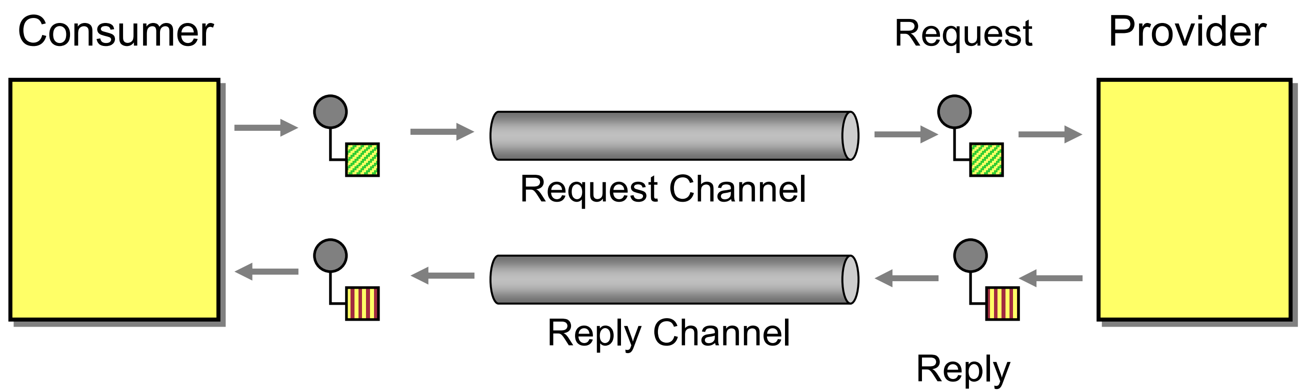
- Point-to-Point Channel
- Publish-Subcr. Channel
- Durable Subscriber
- Datatype Channel
- Invalid Message Channel
- Dead Letter Channel
- Guaranteed Messaging
- Channel Adapter

Systems Management

- Control Bus
- Message Header
- Envelope Wrapper
- Message History
- Message Store
- Channel Purger
- Test Message

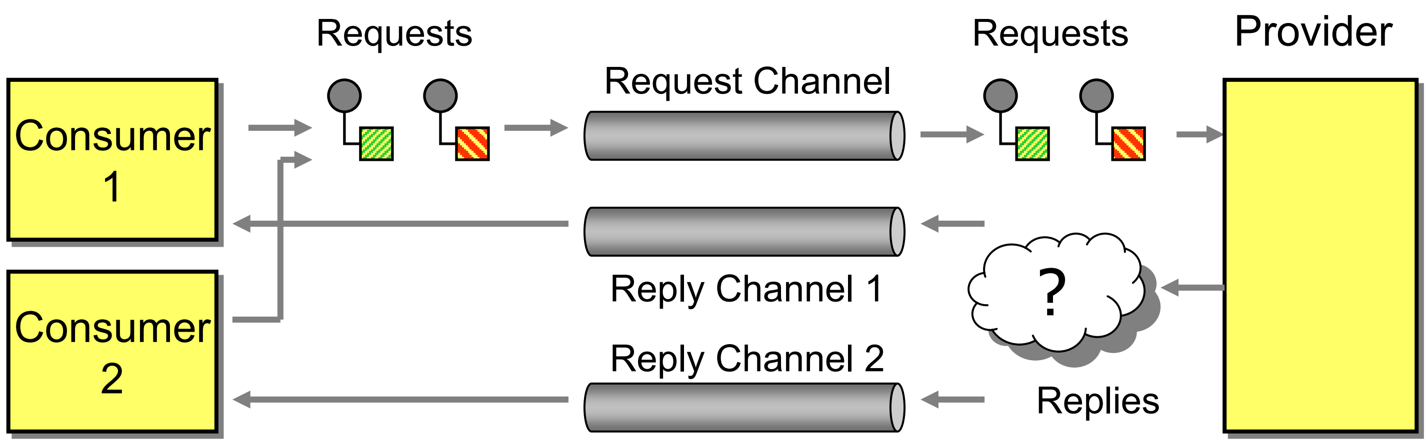
3. Bonus

Pattern: *Request-Reply*



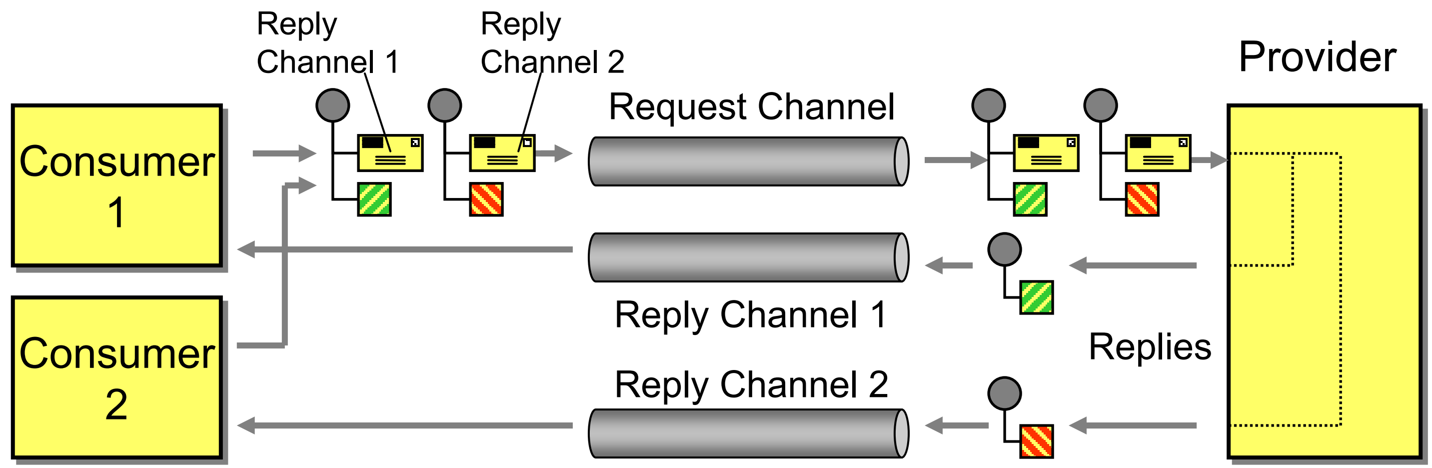
- Service Provider and Consumer (similar to RPC)
- Channels are unidirectional
- Two asynchronous *Point-To-Point Channels*
- Separate request and reply messages

Multiple Consumers



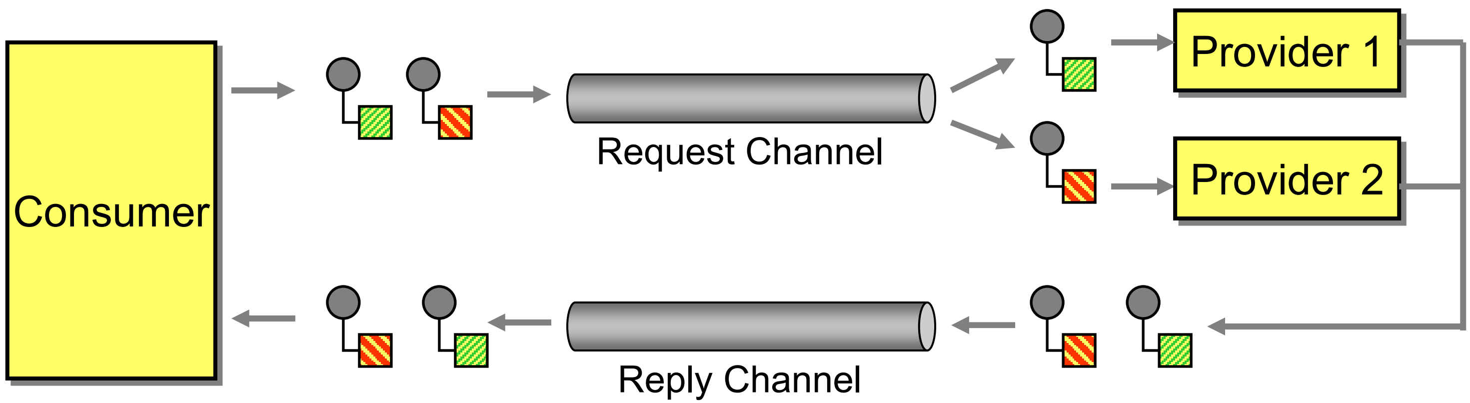
- Each consumer has its own reply queue
- How does the provider know where to send the reply?
  - Could send to all consumers → very inefficient
  - Hard code → violates principle of context-free service

Pattern: *Return Address*



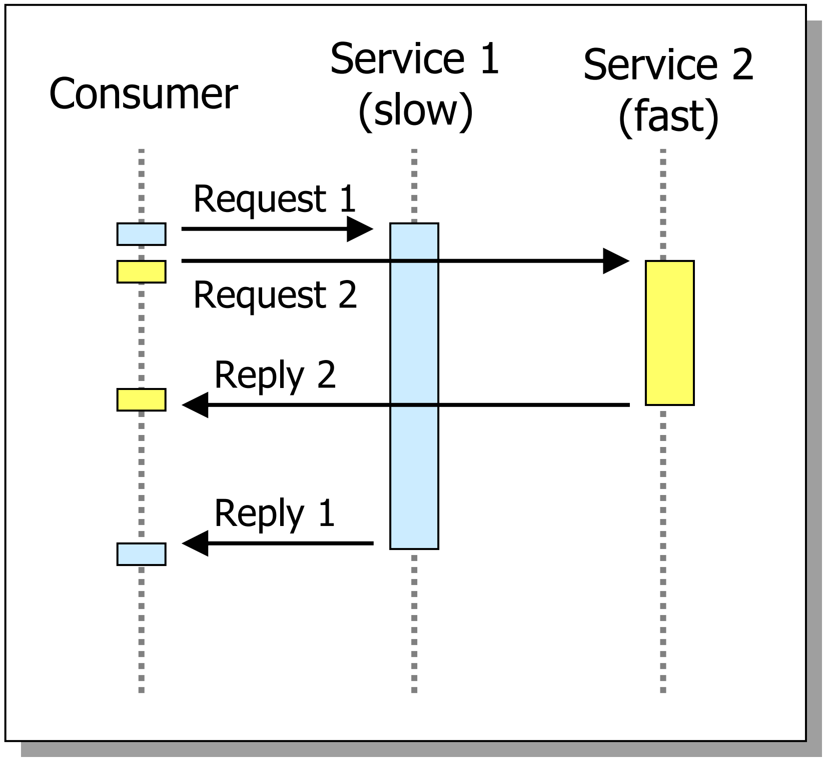
- Consumer specifies *Return Address* (reply channel) in the request message
- Service provider sends reply message to specified channel

Multiple Service Providers



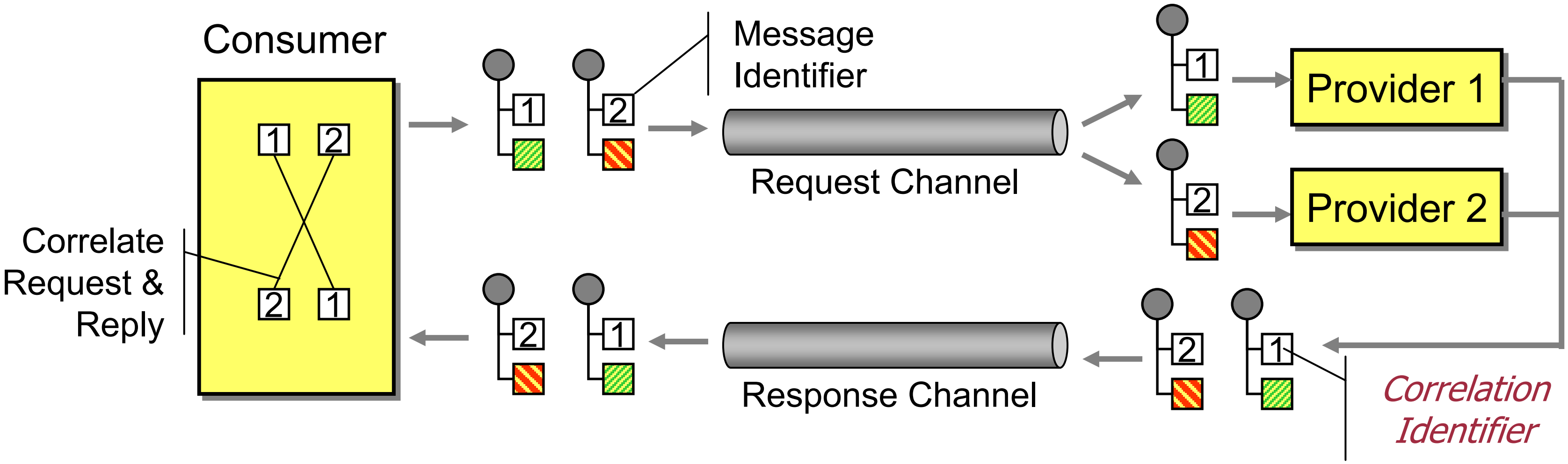
- Request message can be consumed by more than one service provider
- *Point-to-Point Channel* supports *Competing Consumers,* only one service receives each request message
- Channel queues up pending requests

Multiple Service Providers



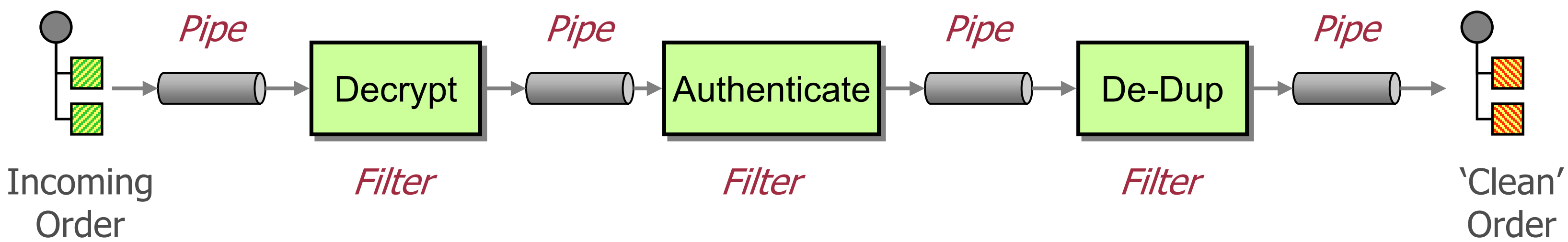
- Reply messages get out of sequence
- How to match request and reply messages?
  - Only send one request at a time
    → very inefficient
  - Rely on natural order
    → bad assumption

Pattern: *Correlation Identifier*



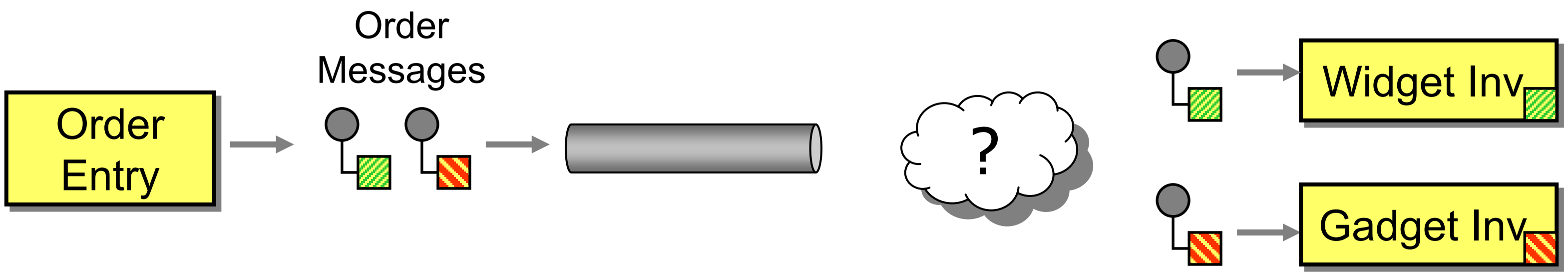
- Equip each message with a unique identifier
  - Message ID (simple, but has limitations)
  - GUID (Globally Unique ID)
  - Business key (e.g. Order ID)
- Provider copies the ID to the reply message
- Consumer can match request and response

Pattern: *Pipes-And-Filters*



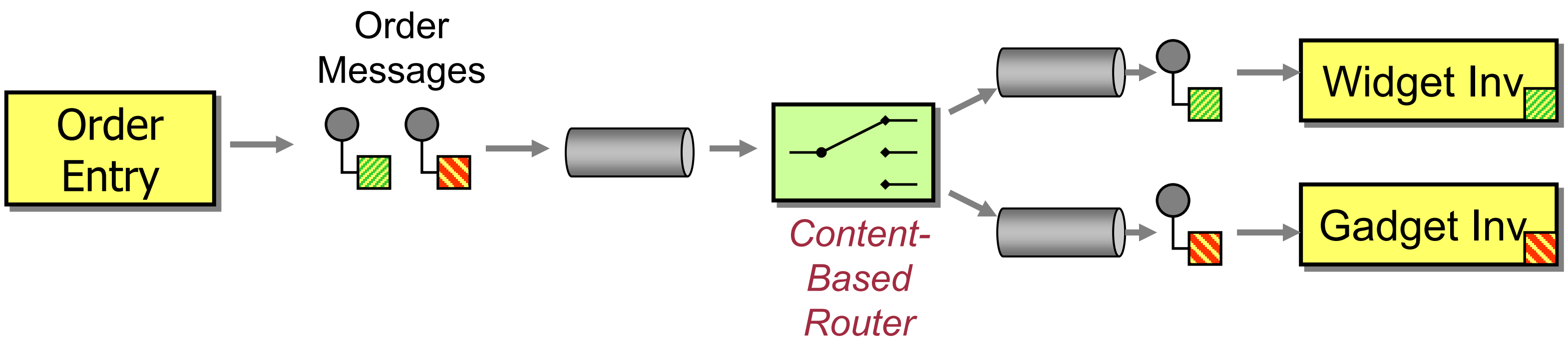
- Connect individual processing steps (filters) with message channels (pipes)
  - Pipes decouple sender and receiver
  - Participants are unaware of intermediaries
  - Compose patterns into larger solutions

Multiple Specialized Providers



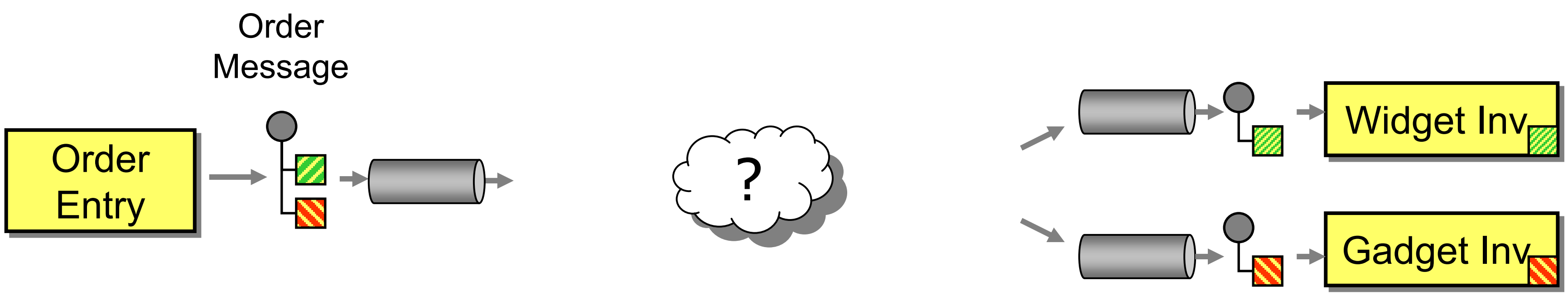
- Each provider can only handle specific type of message
- Route request to the “appropriate” provider based on the content of the request message
  - Do not want to burden sender with decision (decoupling)
  - Letting each consumer “pick out” desired messages requires distributed coordination

Pattern: *Content-Based Router*



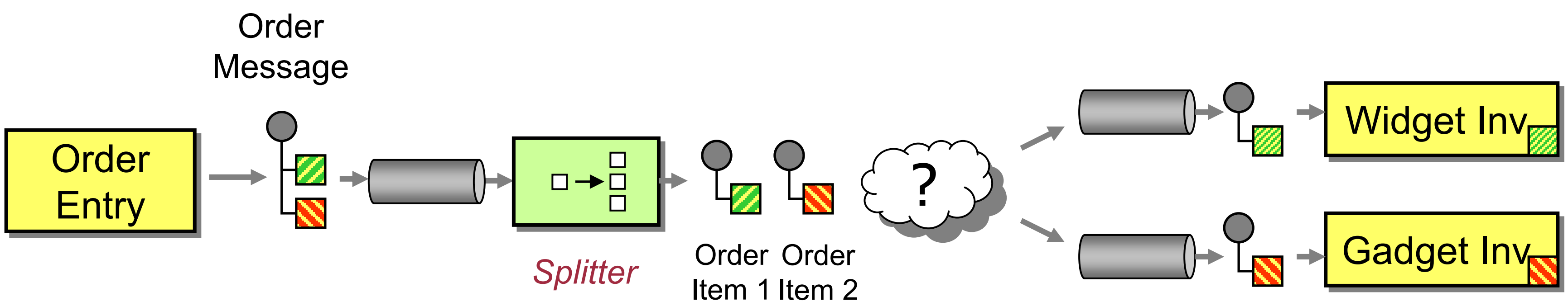
- Insert a Content-Based Router
- Message routers forward incoming messages to different output channels
- Message content not changed
- Mostly stateless, but can be stateful (e.g. de-duper)

Composite Message



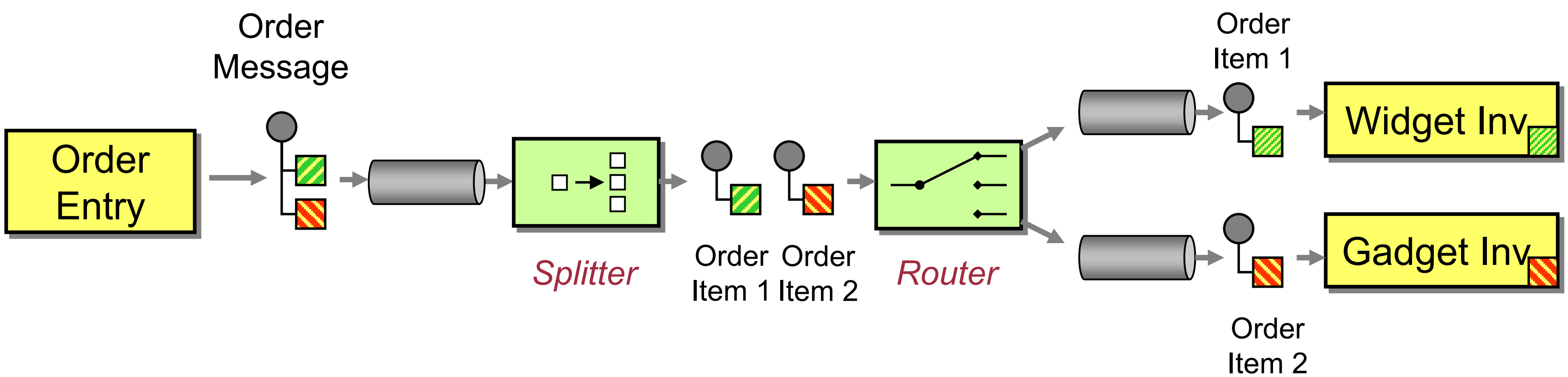
- How can we process a message if it contains multiple elements, each of which may have to be processed in a different way?
  - Treat each element independently
  - Need to avoid missing or duplicate elements
  - Make efficient use of network resources

Pattern: *Splitter*



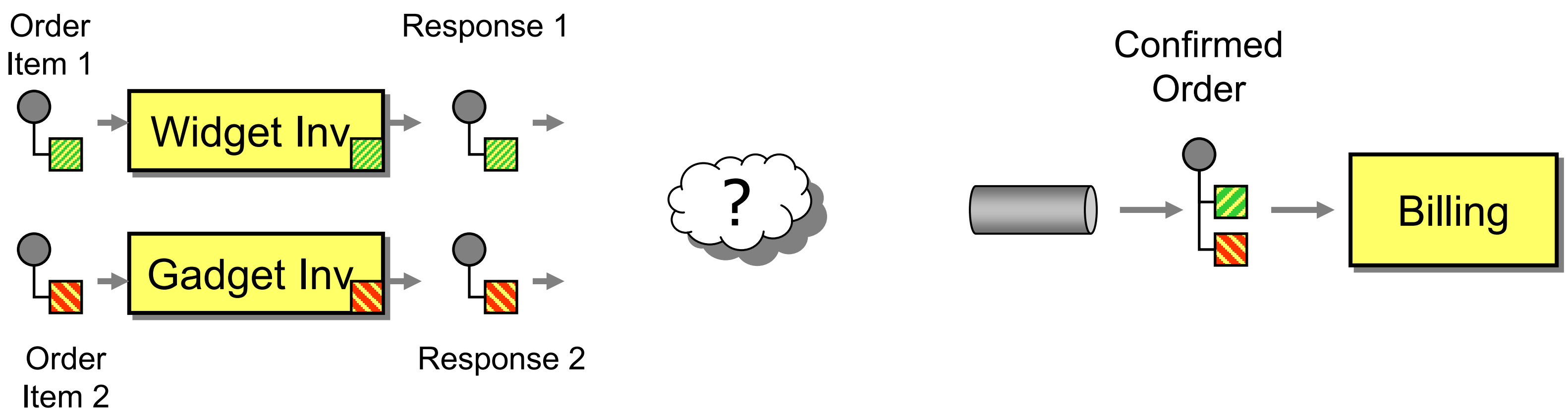
- Use a Splitter to break out the composite message into a series of individual messages, each containing data related to one item.

Composite: *Splitter & Router*



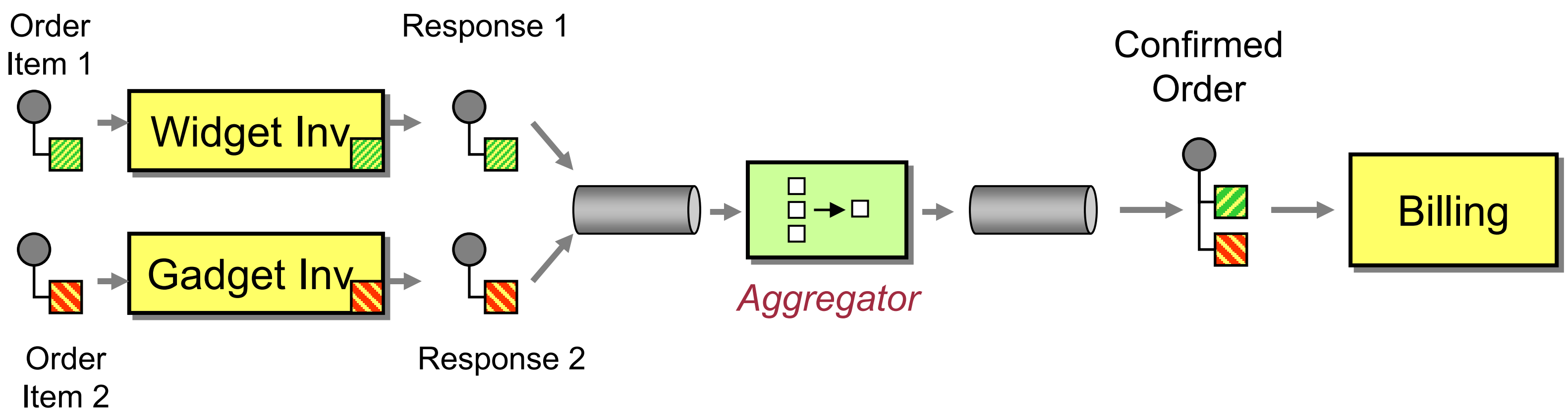
- Use a Splitter to break out the composite message into a series of individual messages, each containing data related to one item.
- Then use a Content-Based Router to route the individual messages to the proper destination

Producing a Single Response



- How to combine the results of individual, but related messages so that they can be processed as a whole?
  - Messages out of order
  - Message delayed
  - Which messages are related?
  - Avoid separate channel for each system

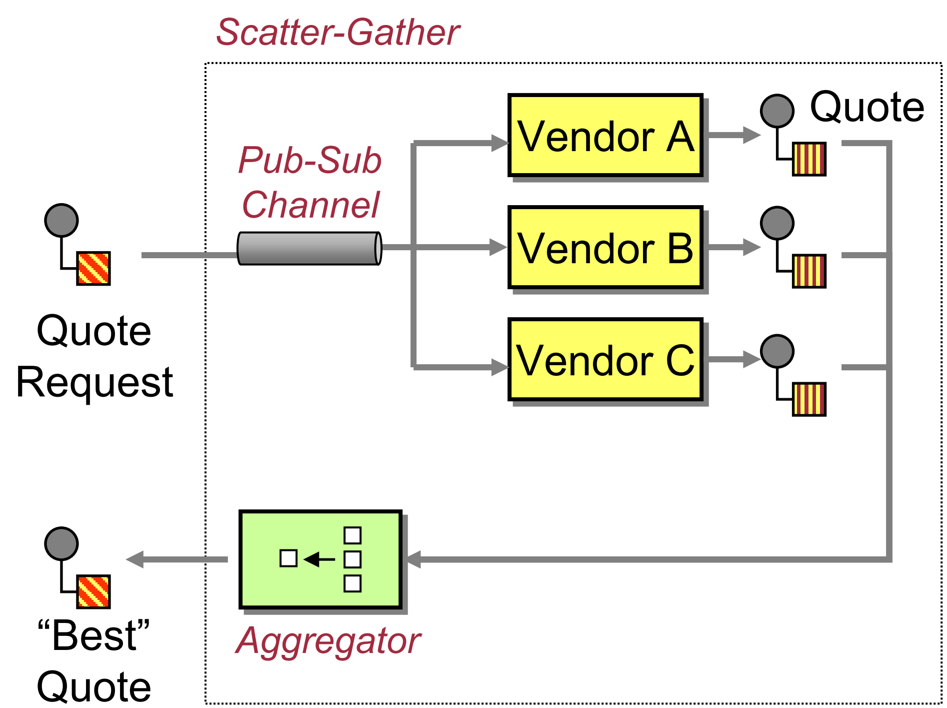
Pattern: *Aggregator*



- Use a stateful filter, an Aggregator, to collect and store individual messages until a complete set of related messages has been received.
  - Aggregator publishes a single message distilled from the individual messages.
  - Correlation
  - Completeness Condition
  - Aggregation Algorithm

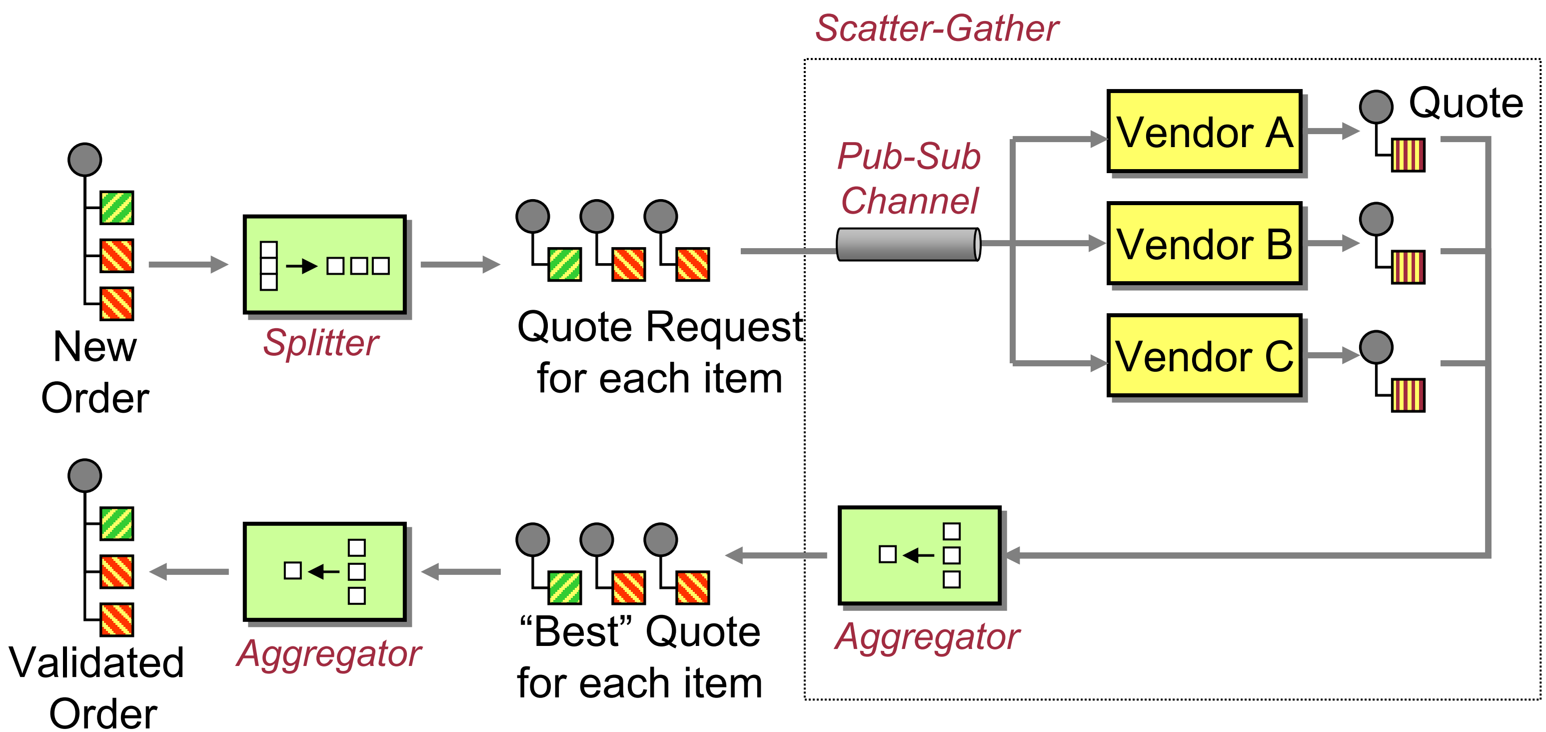
Pattern: *Scatter-Gather*

- Send a message to a dynamic set of recipients, and return a single message that incorporates the responses.



Composing Patterns

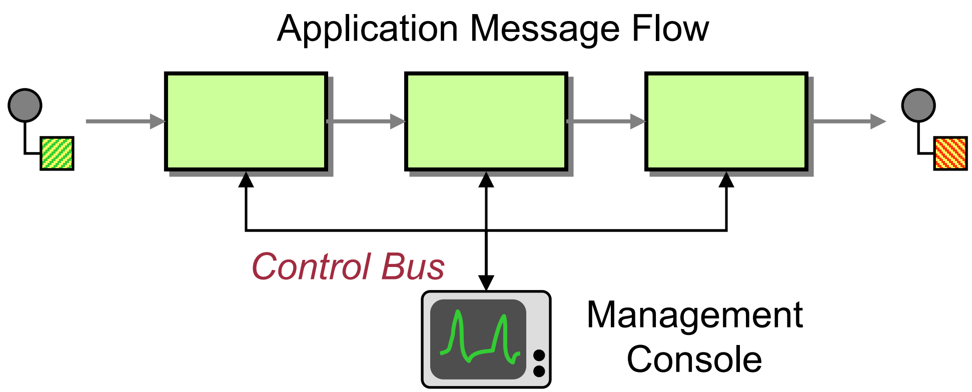
- Receive an order, get best offer for each item from vendors, combine into validated order.



System Management

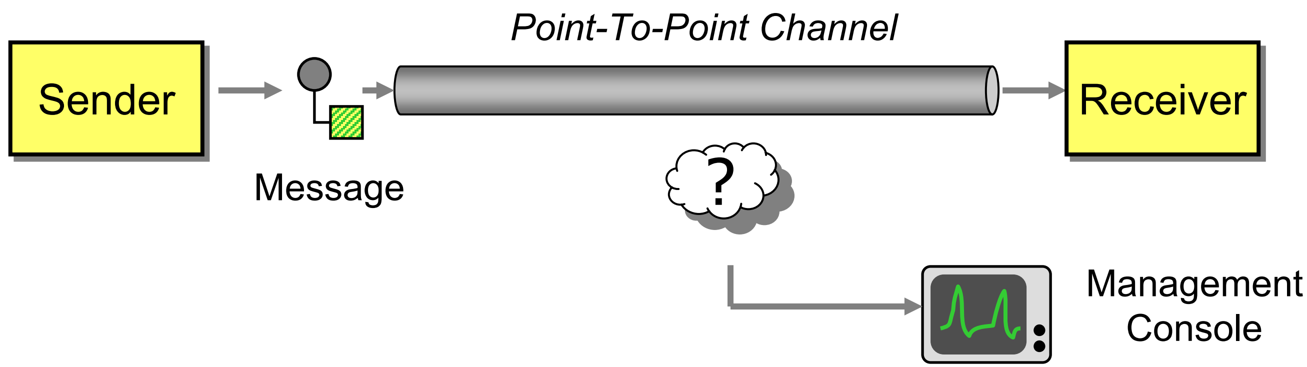
- Messaging systems are asynchronous and distributed
  – Multiple platforms
  – Difficult to detect errors
  – Difficult to configure (property file hell)
- How can we effectively administer a messaging system that is distributed across multiple platforms and a wide geographic area?

Pattern: *Control Bus*



- Configuration
- Heartbeat
- Test messages
- Exceptions / logging
- Statistics / Quality-of-Service (QoS)
- Live console

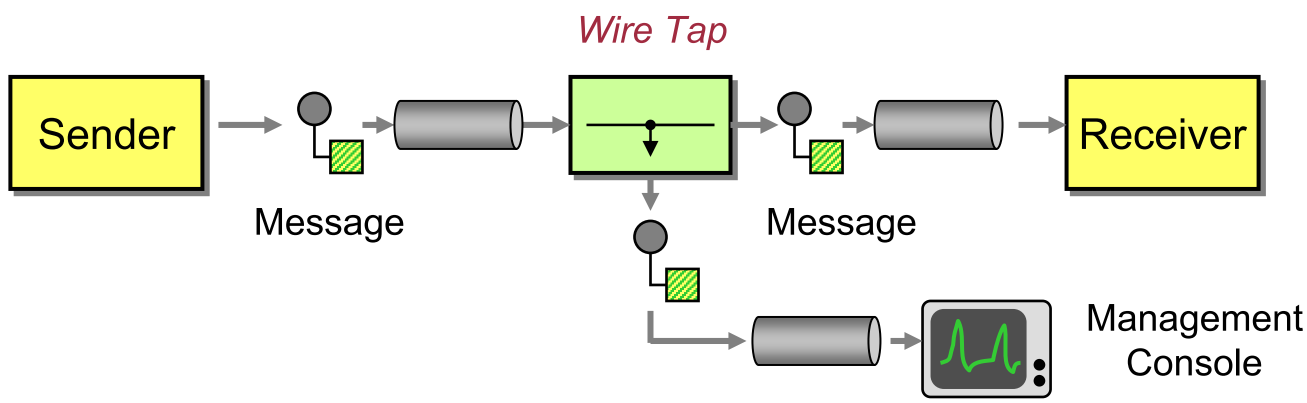
How To Inspect Messages?



- Cannot add another receiver because it would consume the message
- Cannot switch to Publish-Subscribe-Channel because may already have *Competing Consumers*

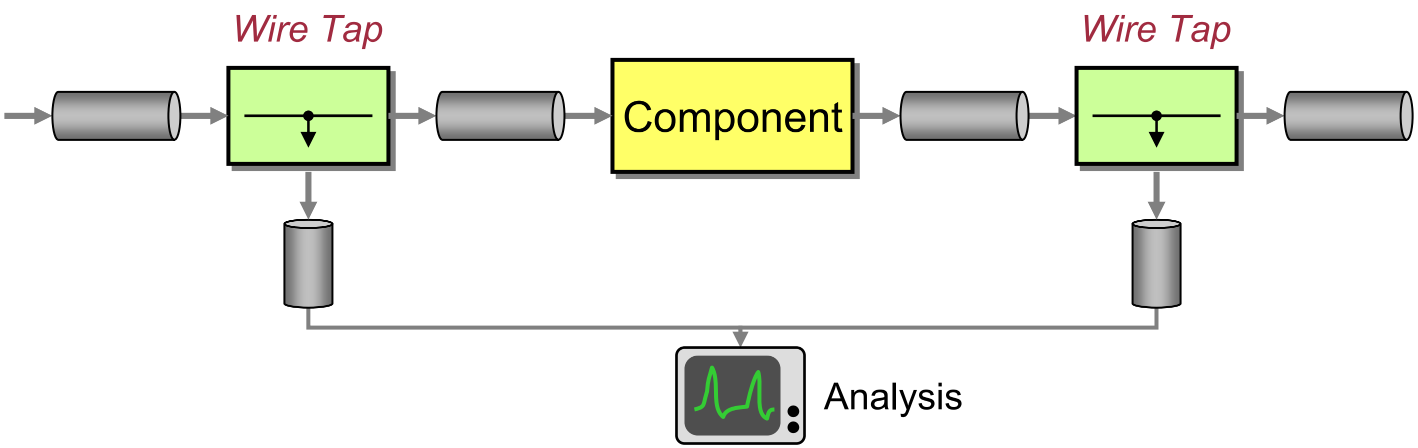
Point-To-Point Channel

Pattern: *Wire Tap*



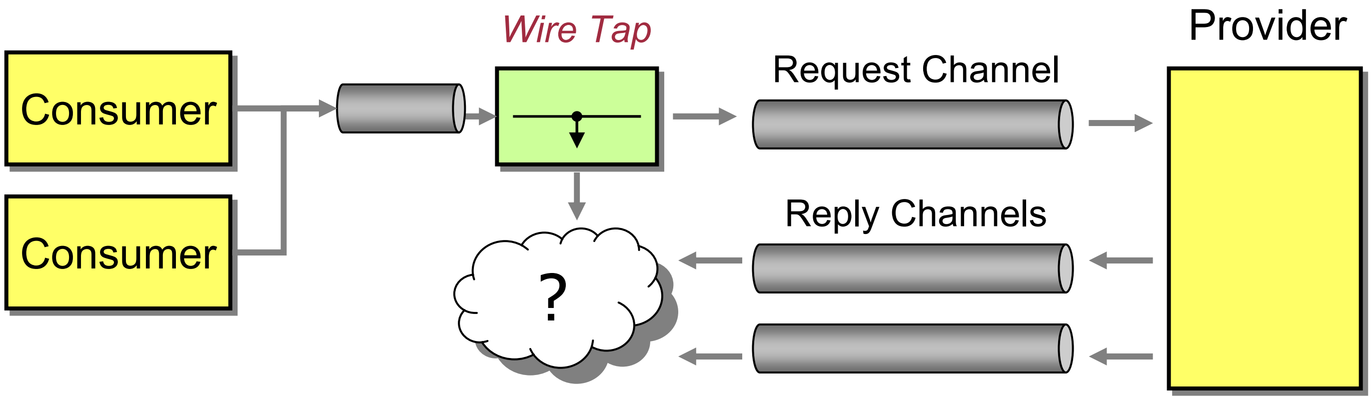
- Simple *Router* that duplicates message to two output channels
- Also known as *Tee*
- Some side effects: Message ID changes, latency

Track Messages



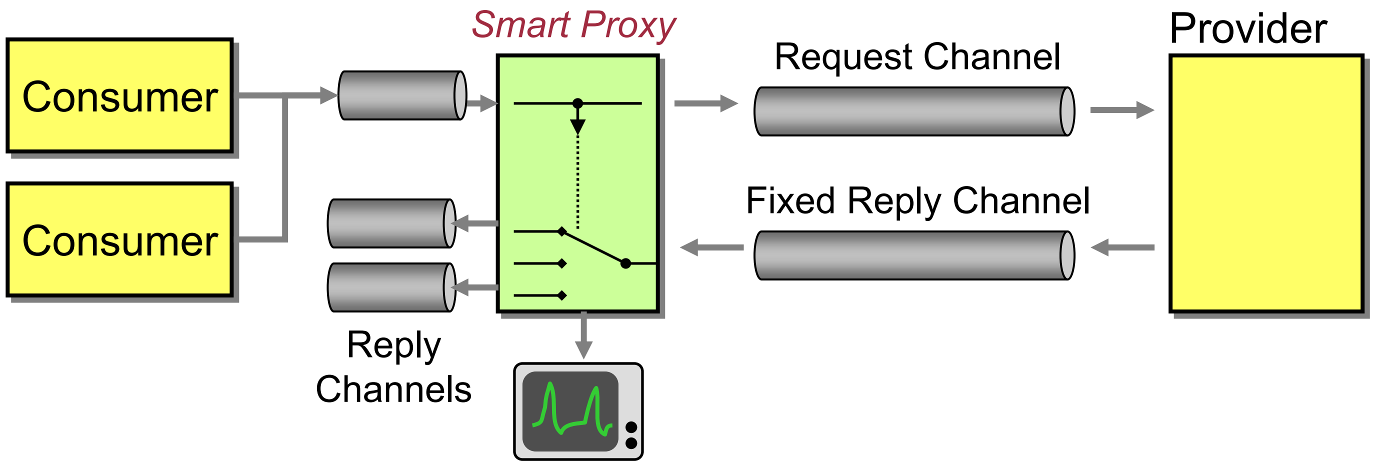
- E.g., message run time, message volume
- Missed messages if channels or component unreliable

What if *Return Address* is Used?



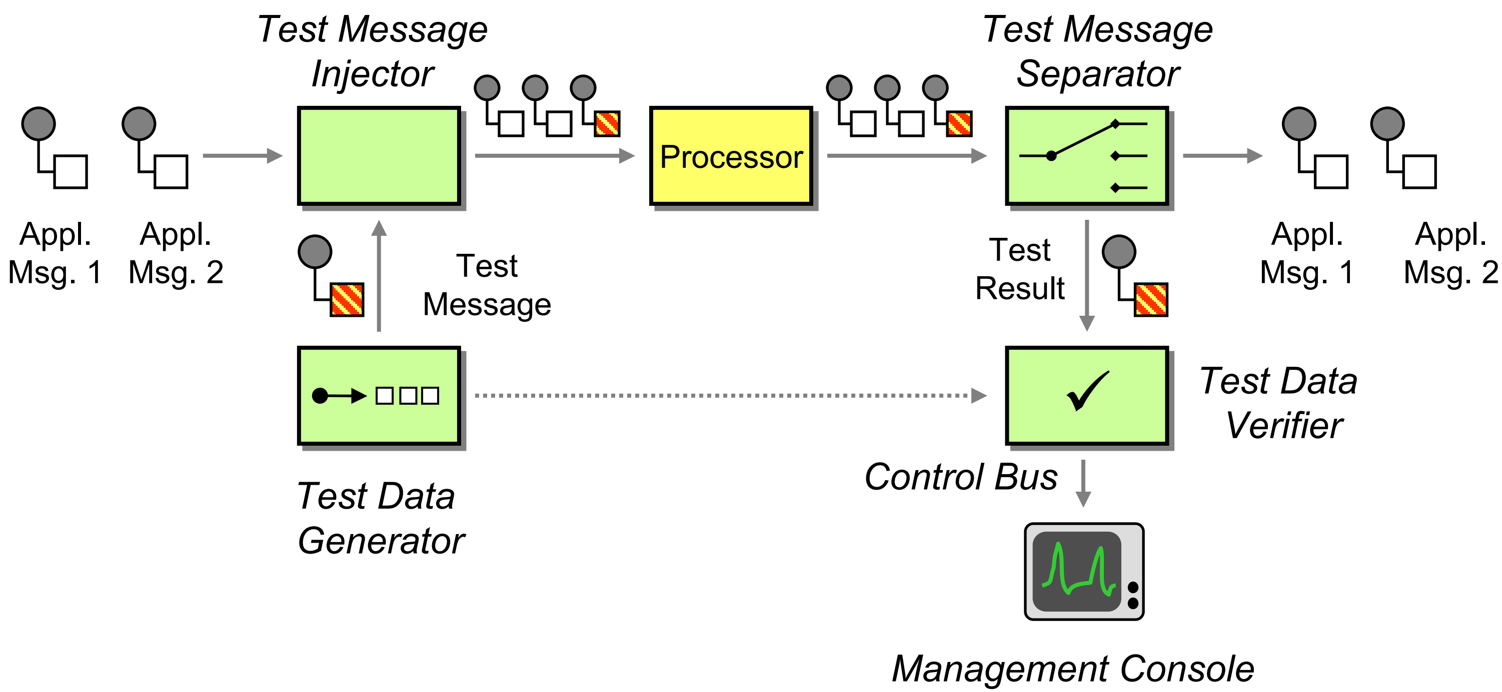
- Provider routes reply message to dynamic channel
- Cannot dynamically inject *Wire Tap*

Pattern: *Smart Proxy*



- A *Smart Proxy* stores original *Return Address* and replaces it with a fixed channel address
- Intercepts reply messages and forwards them to correct channel
- Allows analysis of request and reply messages

Pattern: *Test Message*



- Inject application specific test messages
- Extract result from regular message flow
- Compare result against predefined (or computed) result

In Summary…

- Visual and verbal language to describe integration solutions
- Combine patterns to describe larger solutions
- No fancy tools – whiteboard or PowerPoint
- No vendor jargon
- Not a precise specification language
  - (e.g., see OMG UML Profile for EAI)
- Not a new "methodology"
- Each pattern describes trade-offs and considerations not included in this overview

Resources

- Book (late October):
  - Enterprise Integration Patterns
  - Addison-Wesley, 0-321-20068-3
- Contact
  - Gregor Hohpe
  - ghohpe@thoughtworks.com
- Web Site
  - http://www.eaipatterns.com
  - Pattern catalog
  - Bibliography, related papers
  - info@eaipatterns.com
- www.thoughtworks.com