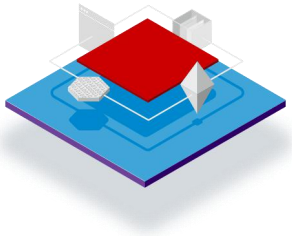


Red Hat Fuse – 7.x

A Distributed, Cloud-native Integration Platform

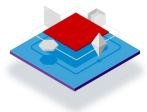
– Avadhut

Course Structure

- **Part – 2 : Integration in Action**
  - Integration options
  - History of Fuse
  - What is Enterprise Integration patterns?
  - **What is OSGi?**

Enterprise Integration Patterns

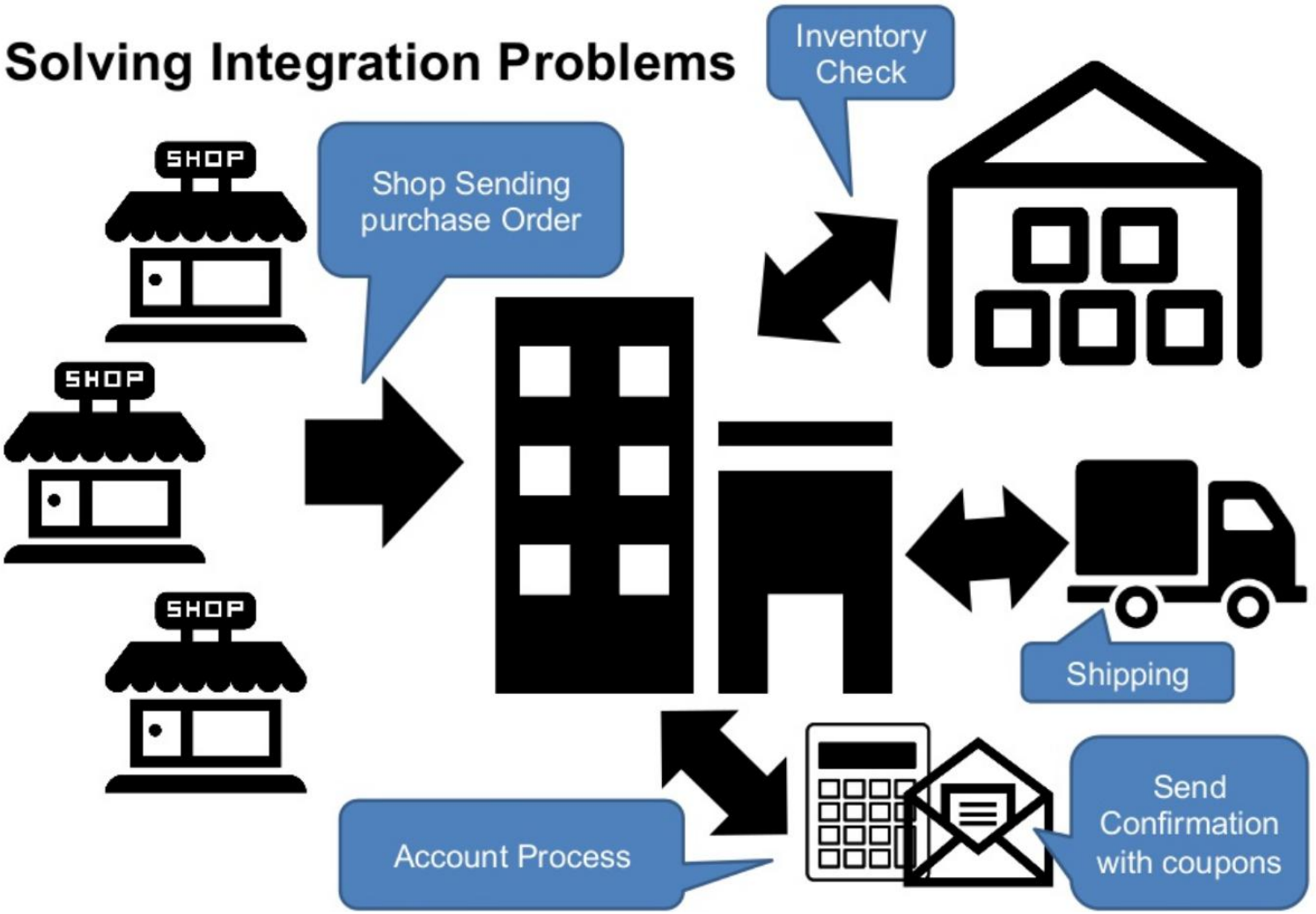
Examples and explanations



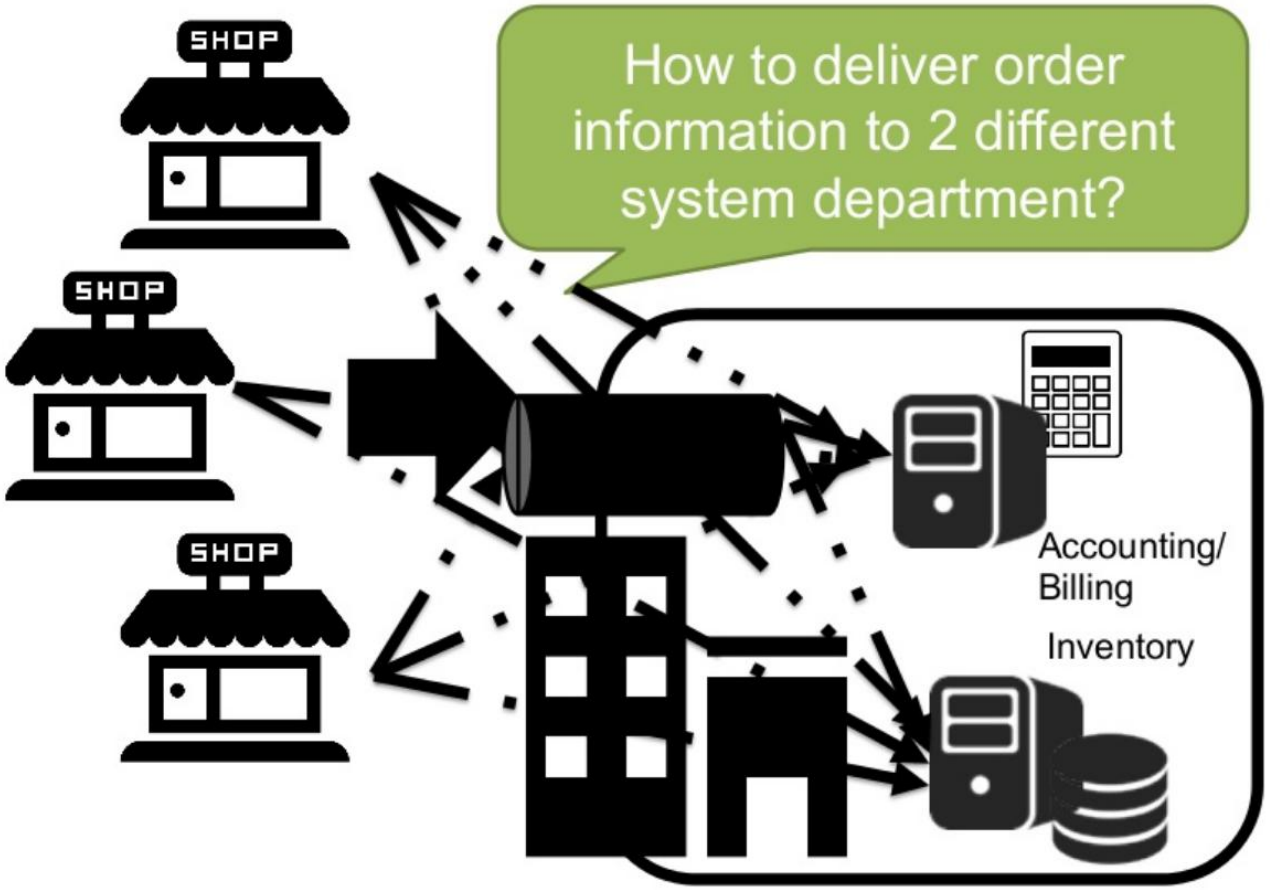
Middleware

- Messages are exchanged between middleware (message oriented middleware)
  - The middleware understands the message format
  - Does not always understand the contents of the message, in particular the body.
    - It does understand at least some of the metadata in the headers
- Middleware is the glue that combines the higher level applications.

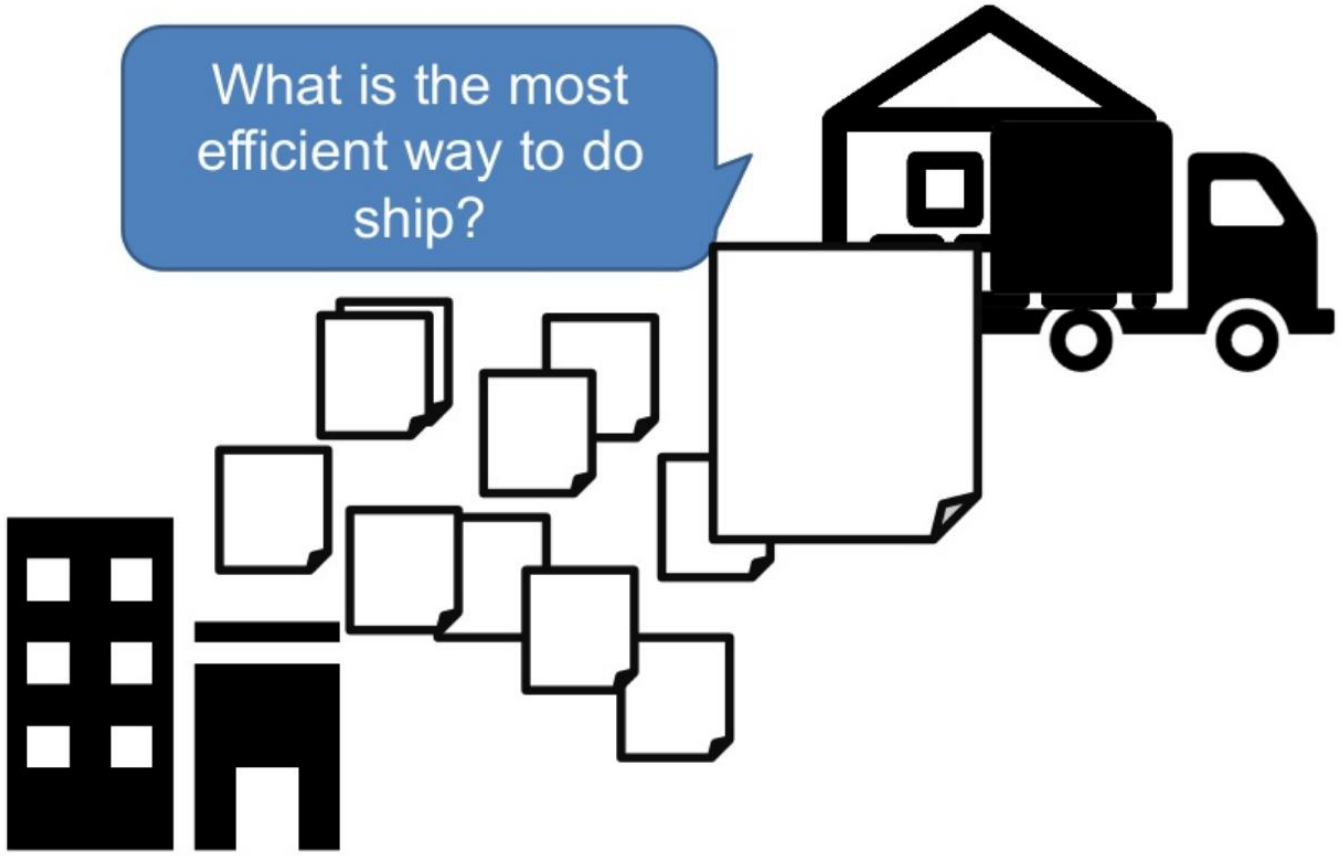
Enterprise Integration Patterns



Solving Integration Problems

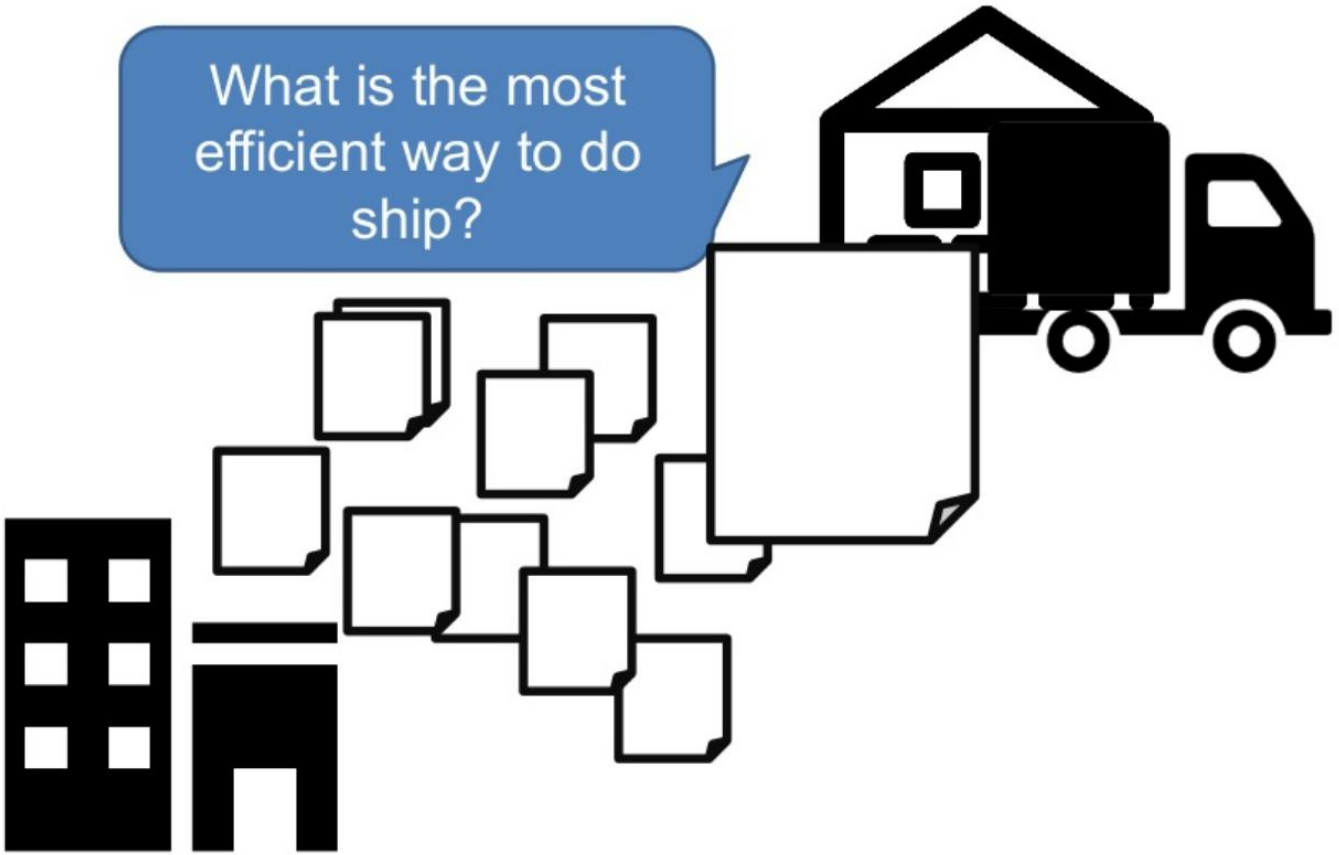


Enterprise Integration Patterns

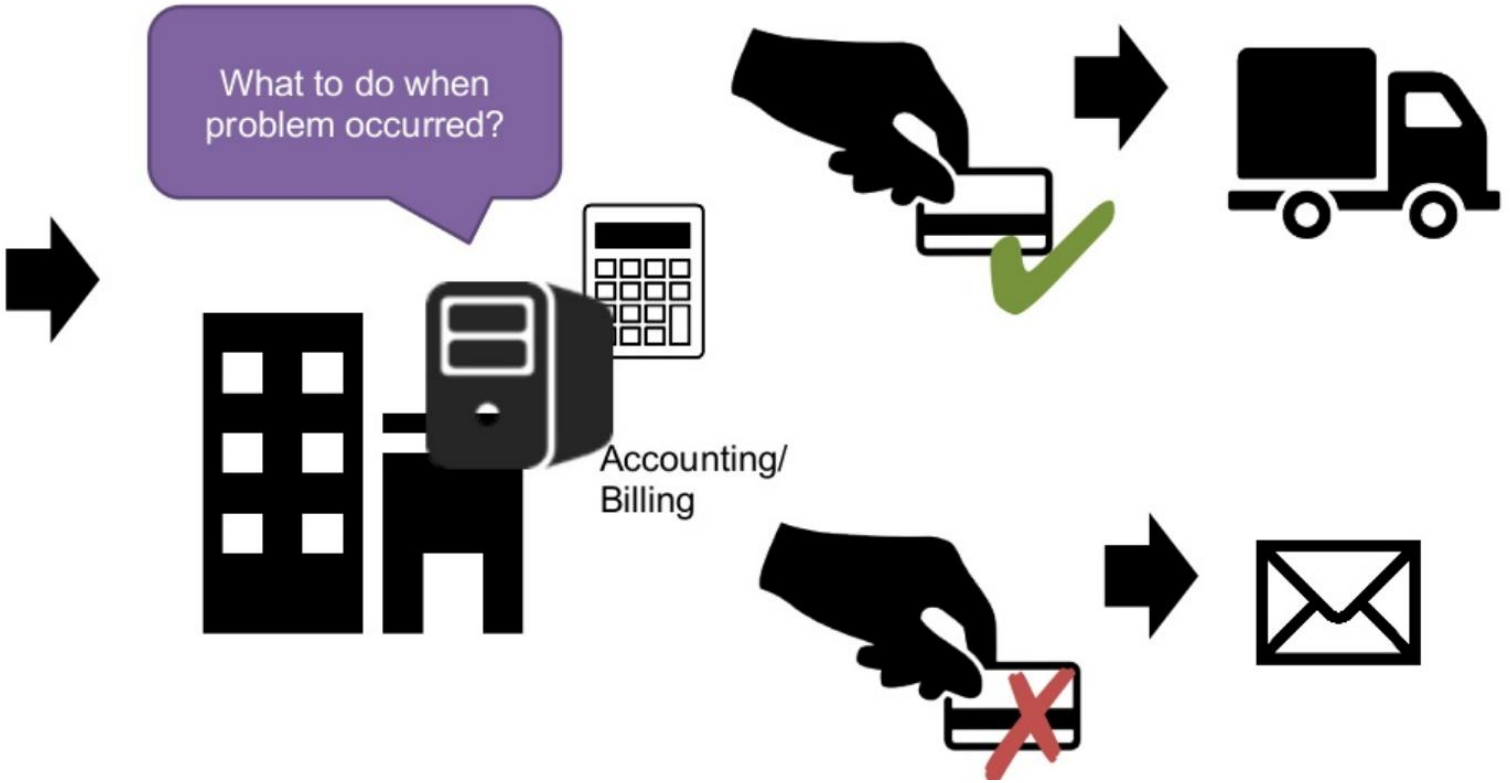


Enterprise Integration Patterns

Solving Integration Problems

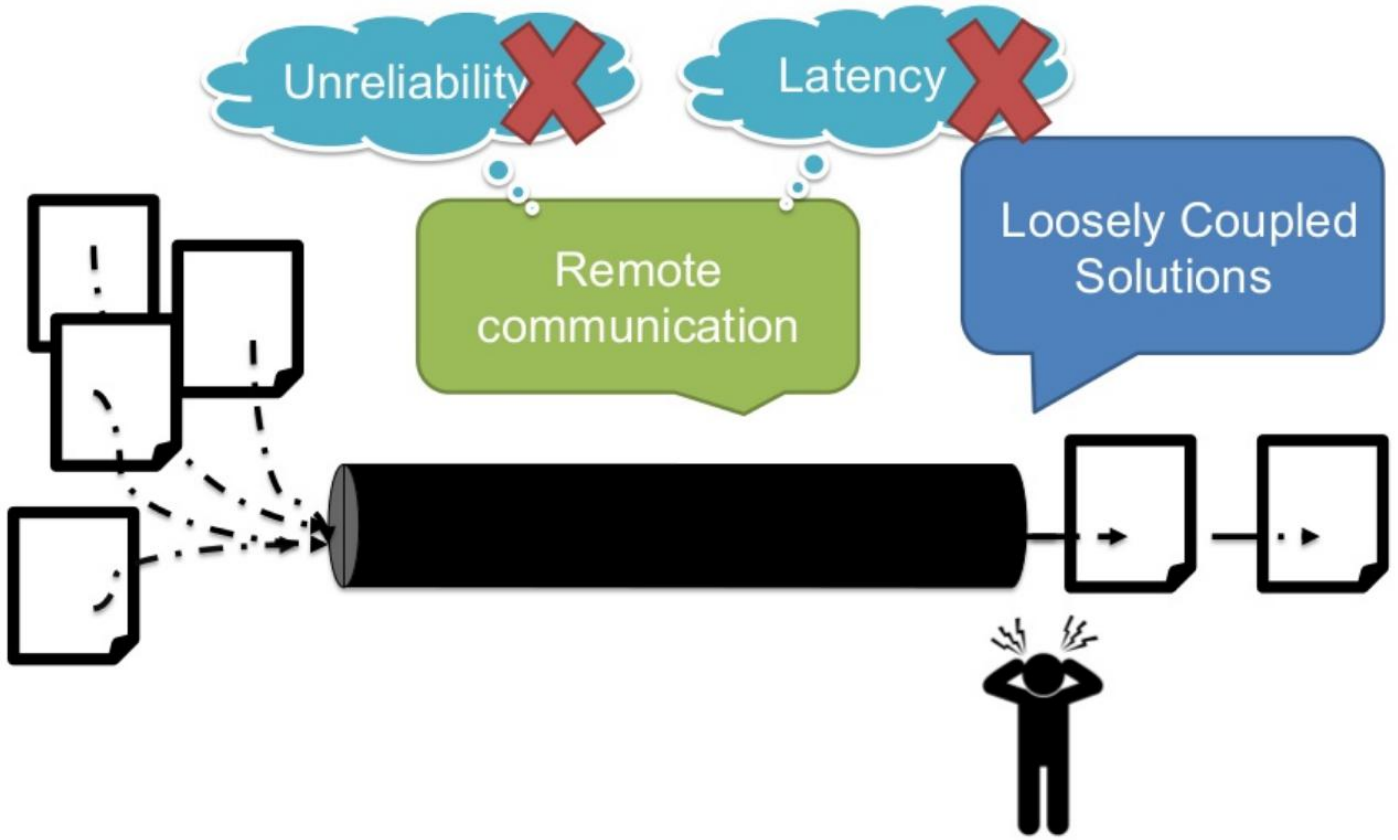


Enterprise Integration Patterns



Enterprise Integration Patterns

Asynchronous Messaging Architectures



What is a Pattern?

- An abstraction that defines high level
  - Relationships between components
  - Actors
  - Reusable
  - Independent of implementation
  - Encapsulates knowledge already learned
    - When you face a problem, consider existing patterns that are applicable – saves you from re-inventing the wheel and making the same mistakes as others
  - Patterns are the language by which system architecture and design are expressed and shared.

What is an Enterprise Integration Pattern?

- An enterprise is a ‘big’ business
- Often contain legacy apps, COTS apps, third party services and internal components
- What is integration?
- Making all those bits work together seamlessly to fulfill business workflow
- So an EIP is a reusable abstraction that is useful in solving the problems raised by enterprise Integration

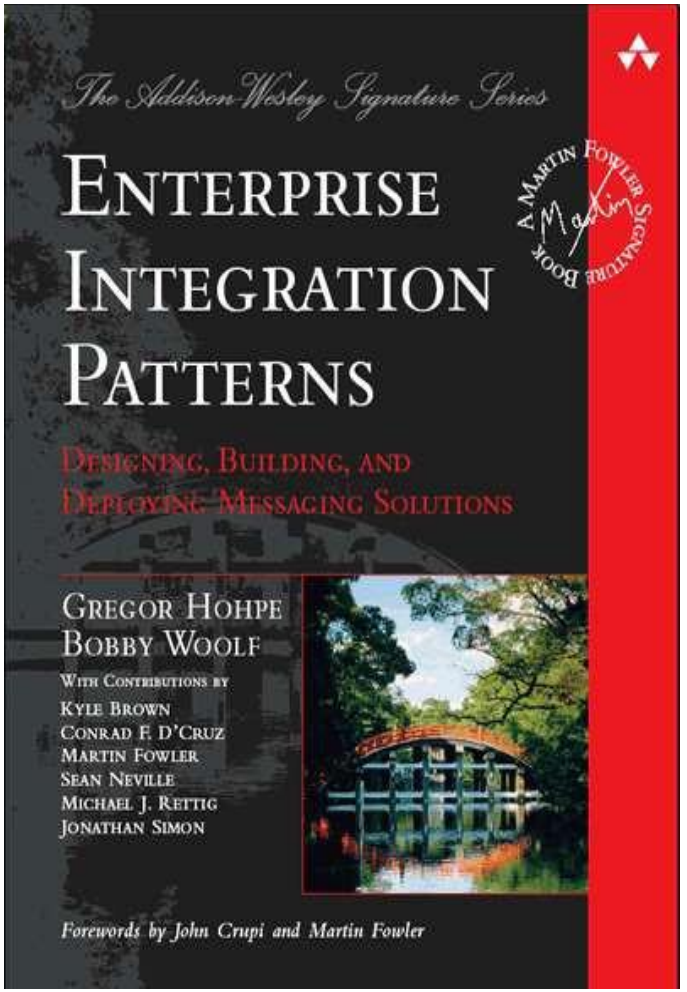
What is EIP?

- A design pattern is a general solution to a design problem that recurs repeatedly in many projects.
- A pattern describes the problem and its proposed solution and discuss any other important factors.
- EIP focuses on messaging patterns for enterprise application integration (EAI).
- Messaging makes it easier for programs to communicate across different programming environments (languages, compilers, and operating systems) because the only thing that each environment needs to understand is the common messaging format and protocol.
- Messaging patterns define the means by which different elements in a message-passing system connect and communicate to enable interaction among objects within programs and among various types of software -- which may be written in different languages and exist on different platforms in multiple locations

What is EIP?

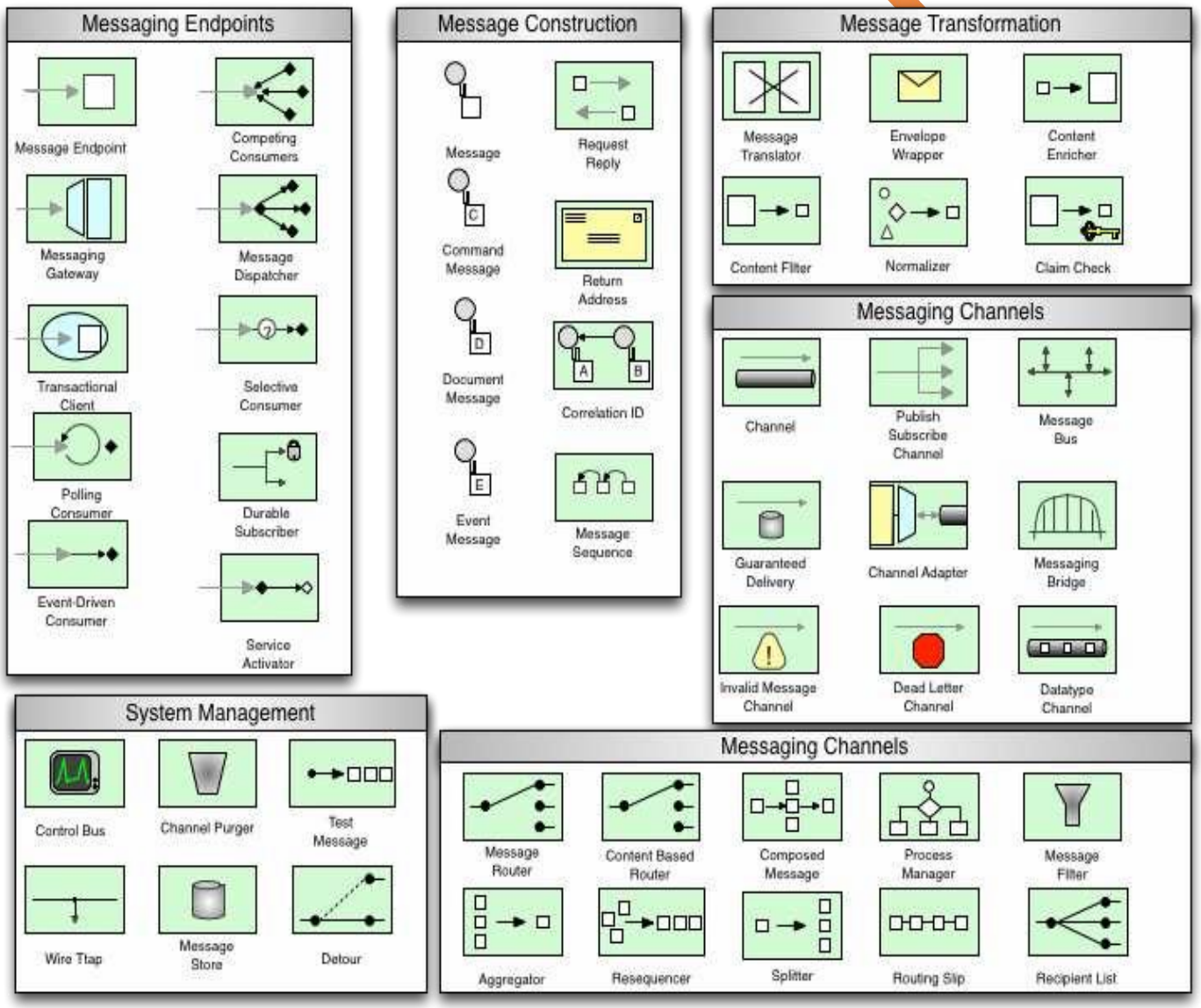
- A design pattern is a general solution to a design problem that recurs repeatedly in many projects.
- A pattern describes the problem and its proposed solution and discuss any other important factors.
- EIP focuses on messaging patterns for enterprise application integration (EAI).
- Messaging makes it easier for programs to communicate across different programming environments (languages, compilers, and operating systems) because the only thing that each environment needs to understand is the common messaging format and protocol.
- Messaging patterns define the means by which different elements in a message-passing system connect and communicate to enable interaction among objects within programs and among various types of software -- which may be written in different languages and exist on different platforms in multiple locations

The Bible of Enterprise Integration Patterns



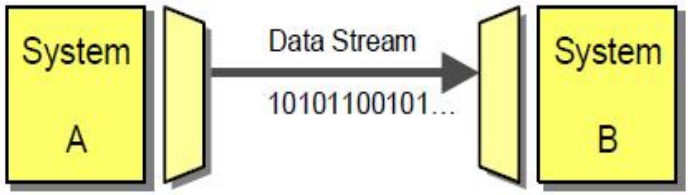
http://www.eaipatterns.com/toc.html

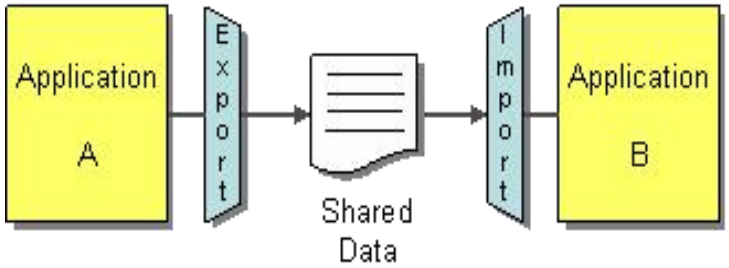
Visual Pattern Language

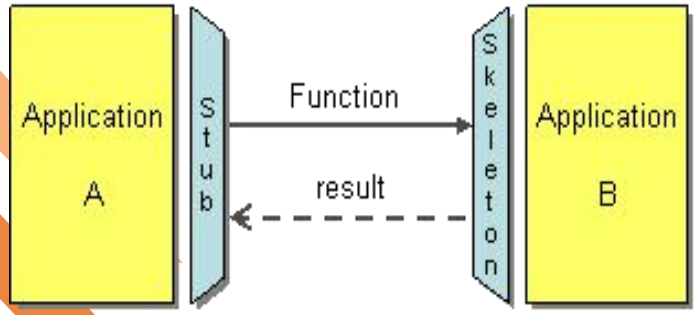


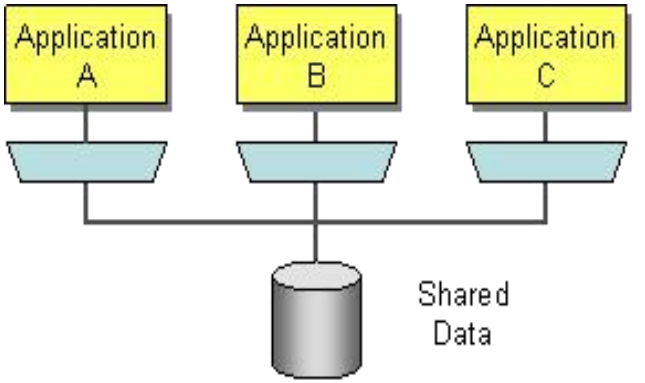
Basic Definitions

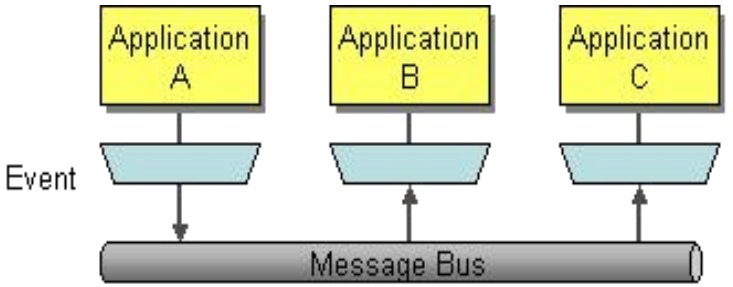
Integration styles



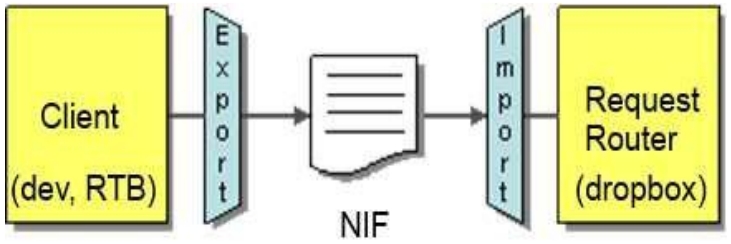


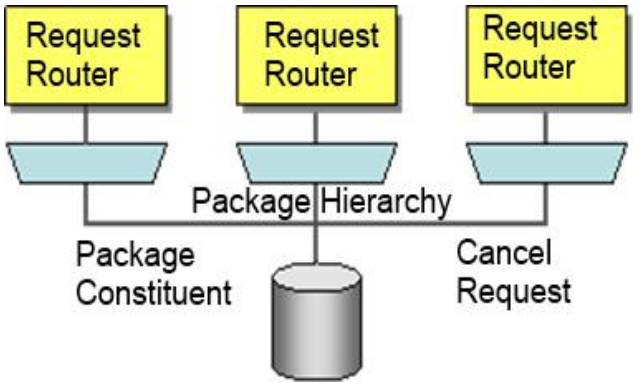


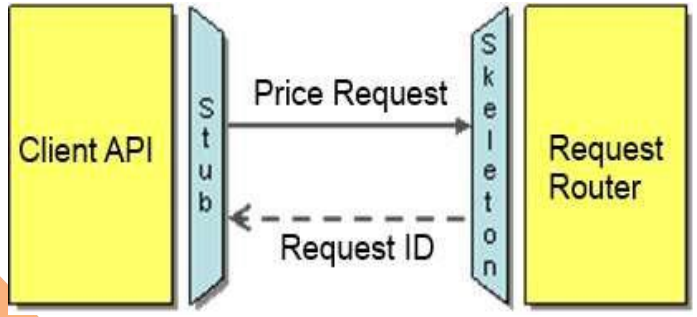


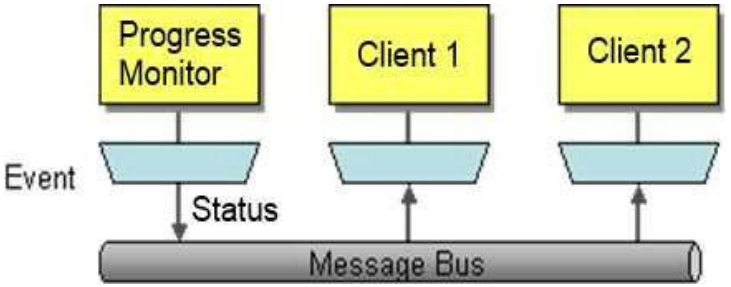


In Nova


Client
(dev, RTB)
Export
NIF
Import
Request
Router
(dropbox)

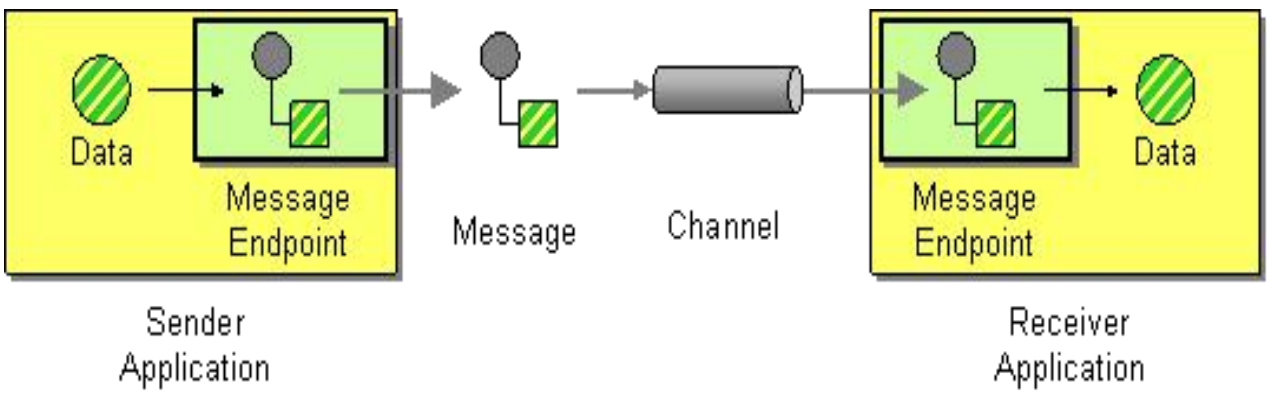

Request
Router
Request
Router
Request
Router
Package Hierarchy
Package
Constituent
Cancel
Request


Client API
Stub
Price Request
Request ID
Skeleton
Request
Router


Progress
Monitor
Client 1
Client 2
Event
Status
Message Bus

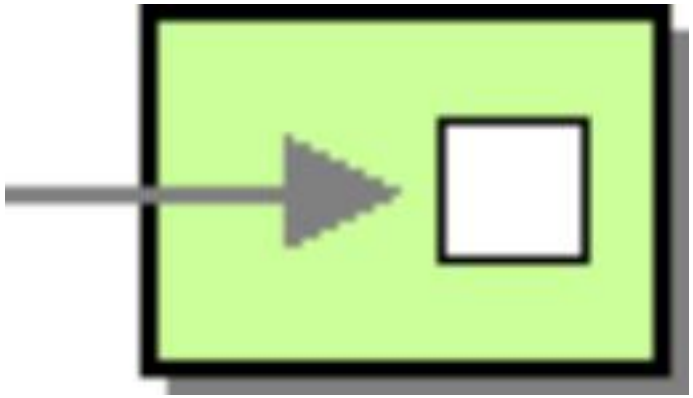
Main building blocks

- **Endpoint**
- **Channel**
- **Message**



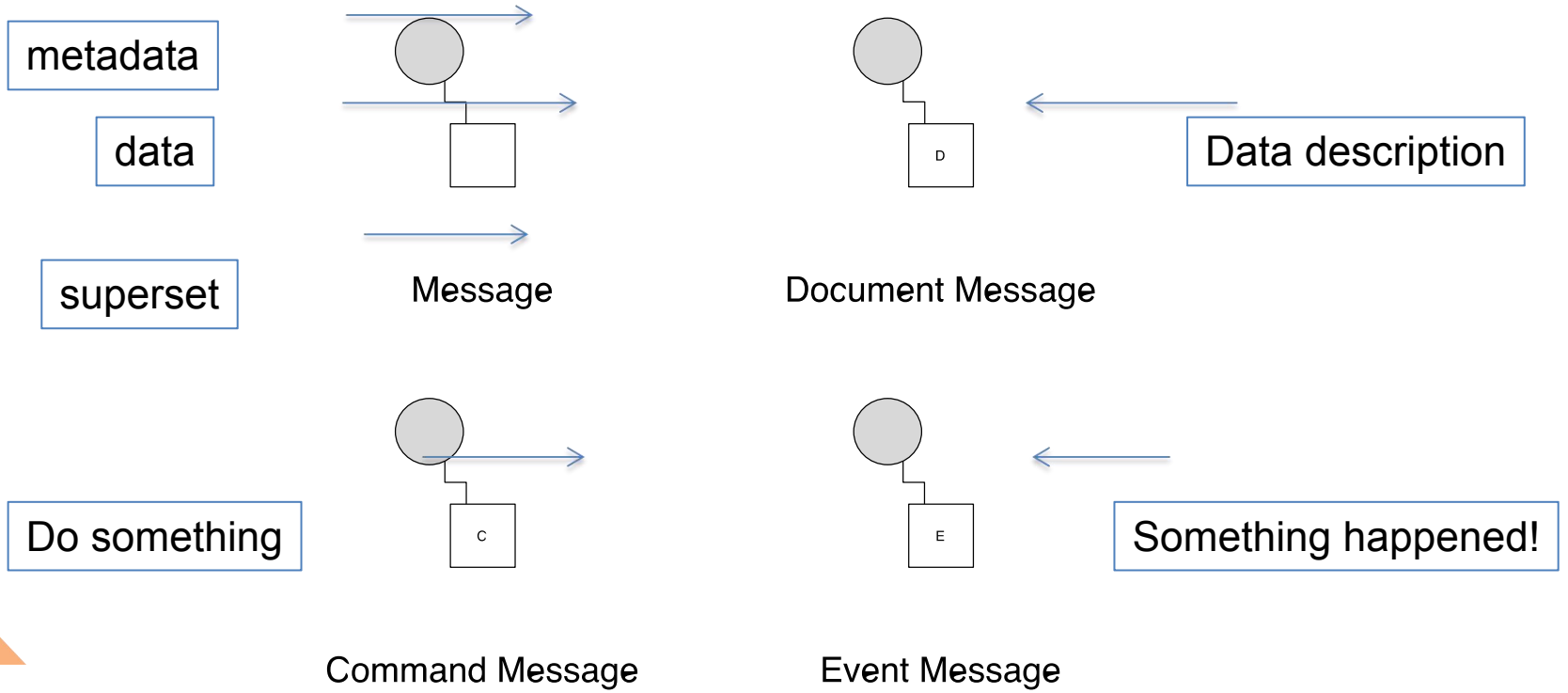
Messages

Message Endpoints

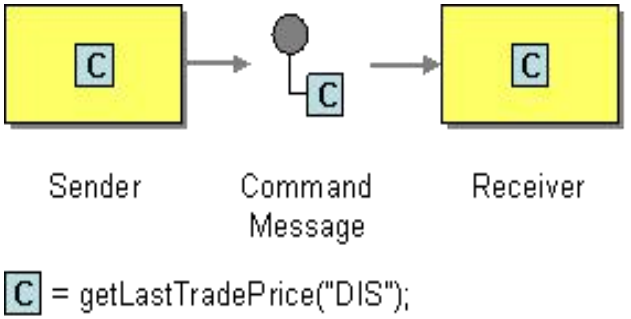


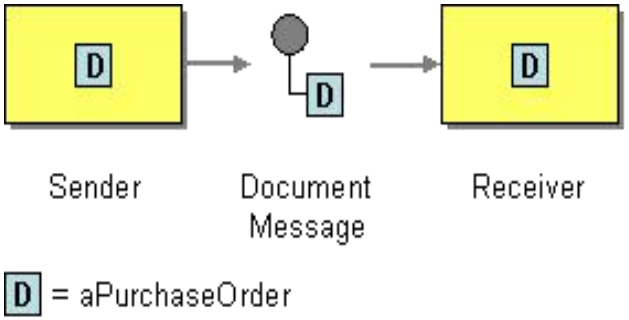
Messages

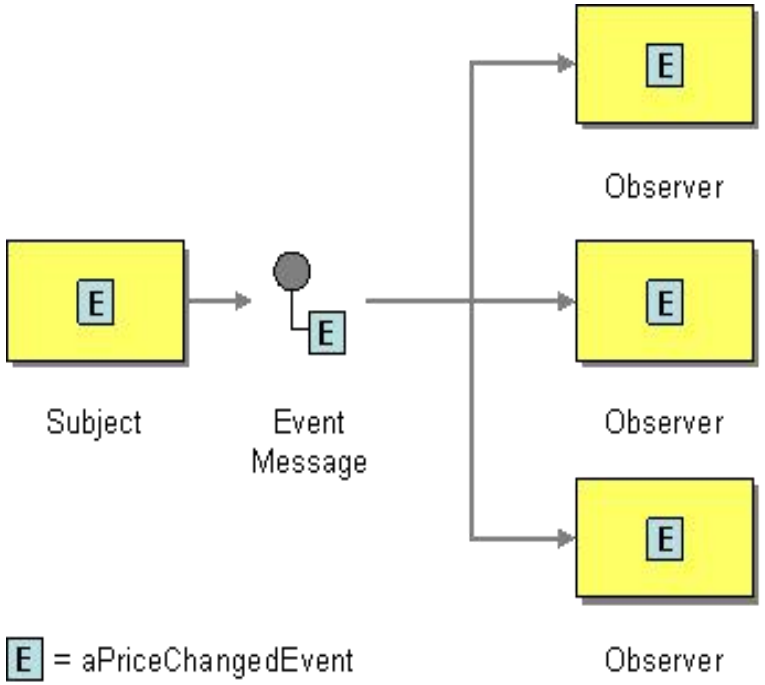
A message is a discrete piece of data sent from one application to another
Messages typically contain headers (metadata) and a body (application payload)



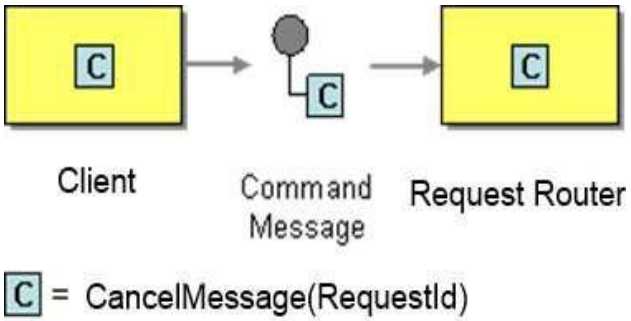
Message types

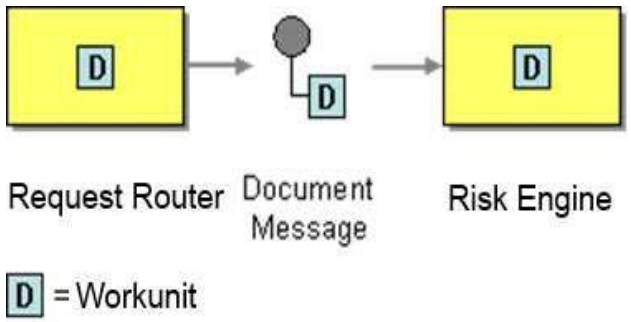


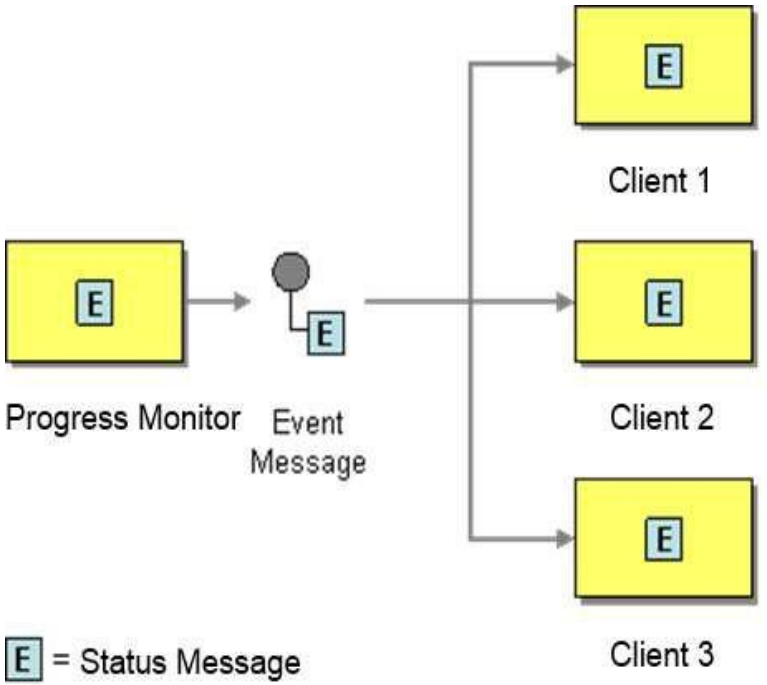




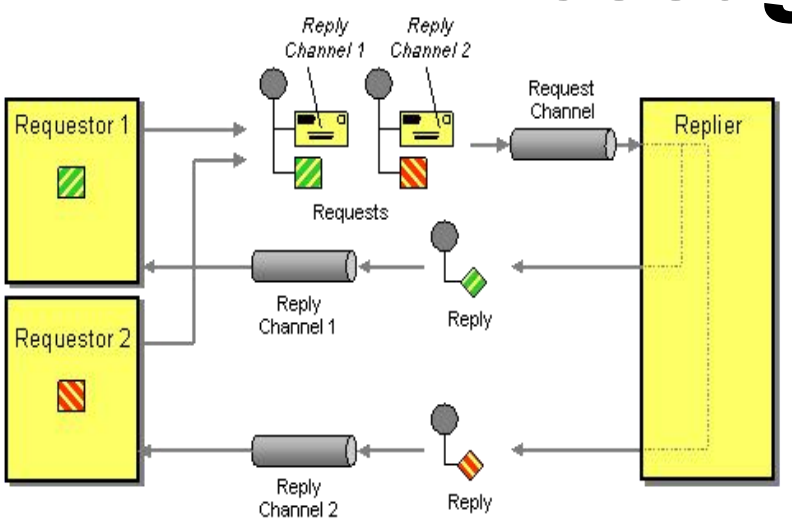
In Nova


C
C
C
Client
Command Message
Request Router
C = CancelMessage(RequestId)

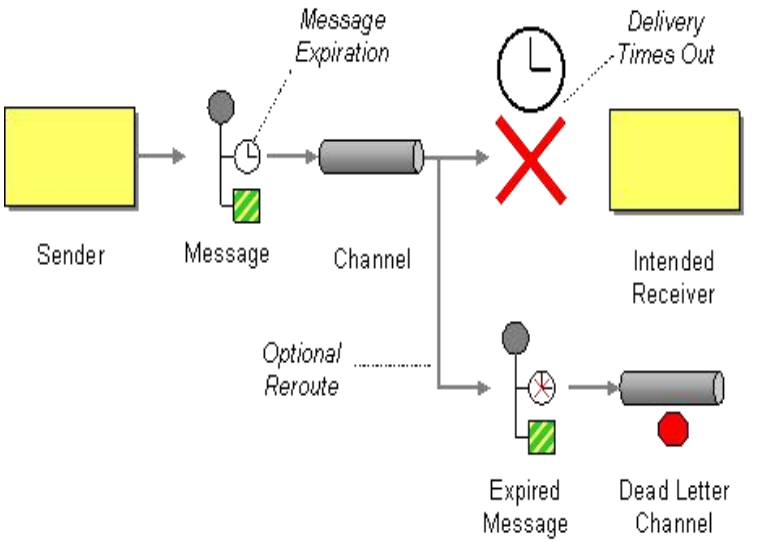

D
D
D
Request Router
Document Message
Risk Engine
D = Workunit


E
E
E
E
E
E
Progress Monitor
Event Message
Client 1
Client 2
Client 3
E = Status Message

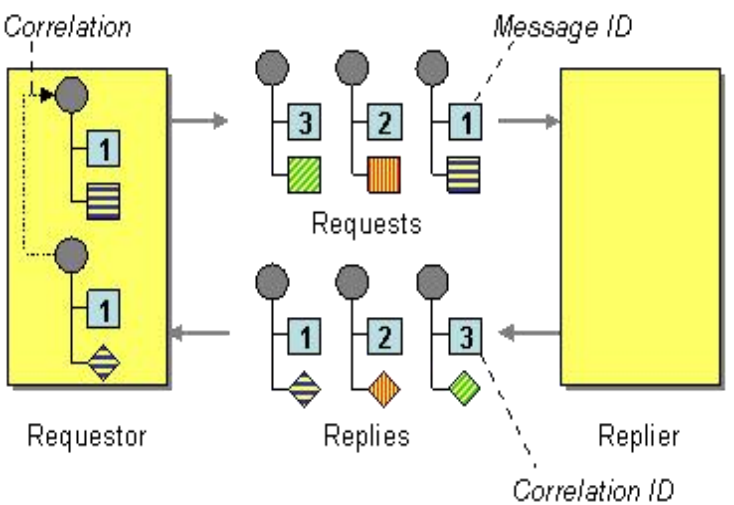
Message attributes



Return address

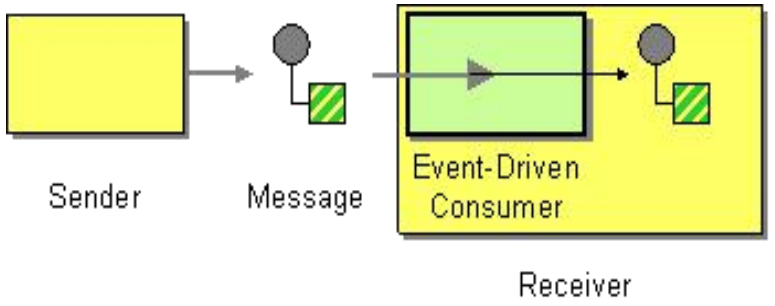


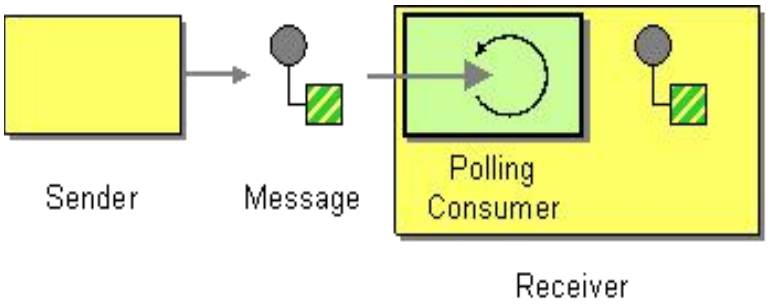
Expiration time



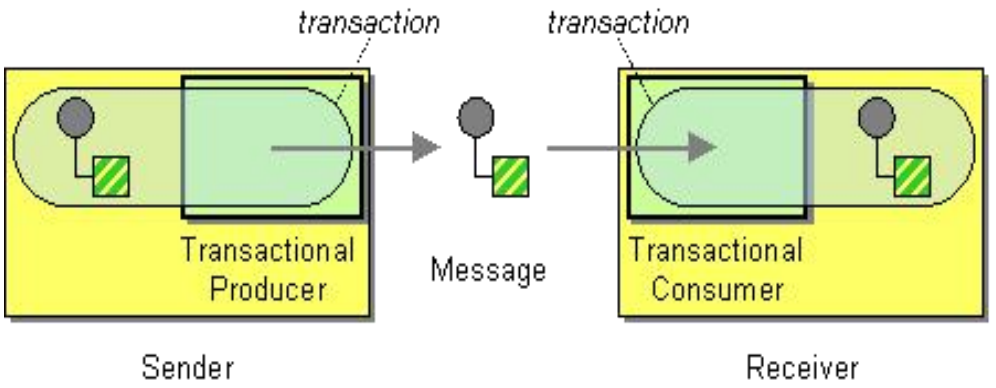
Correlation ID

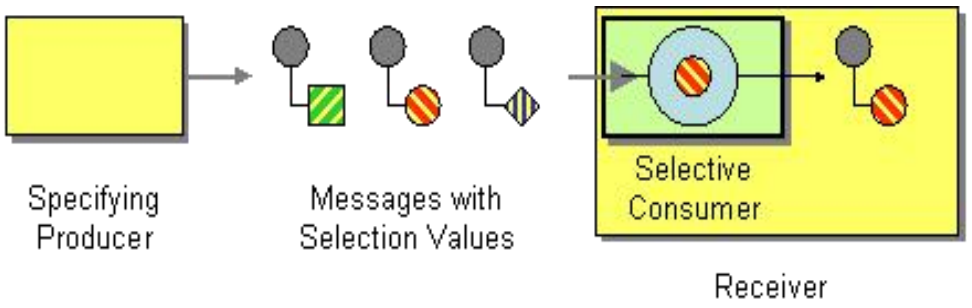
Messaging endpoints





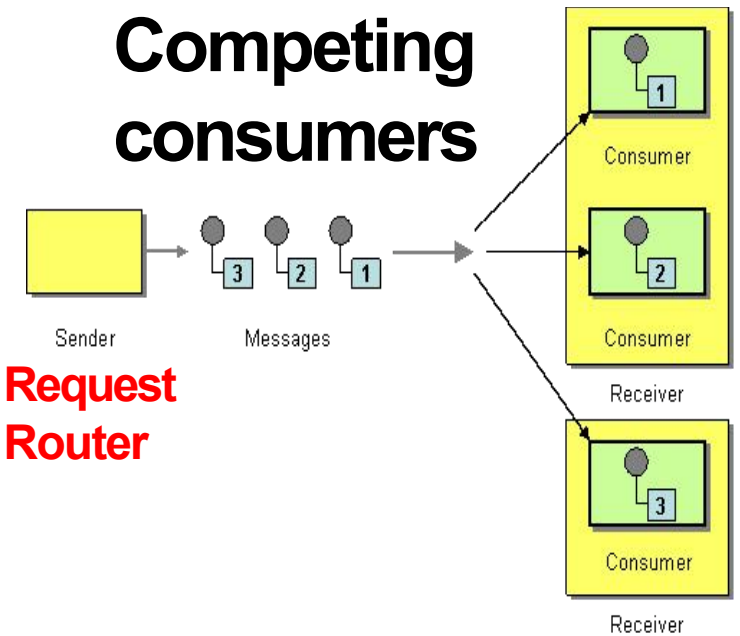
Messaging endpoints





Messaging endpoints

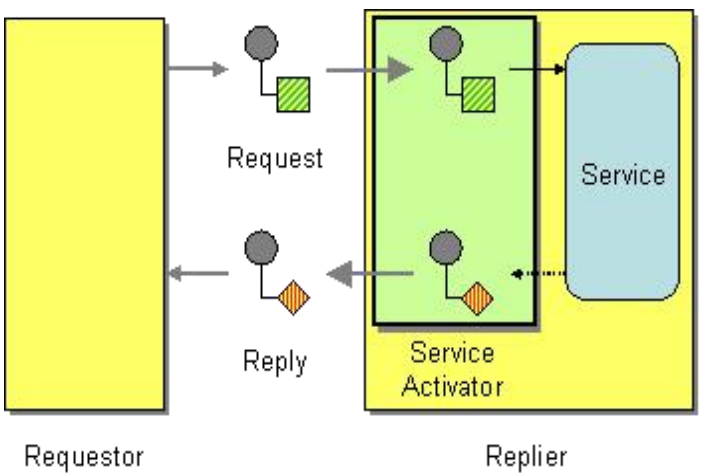
Competing consumers



Request Router

Risk Engine Farm

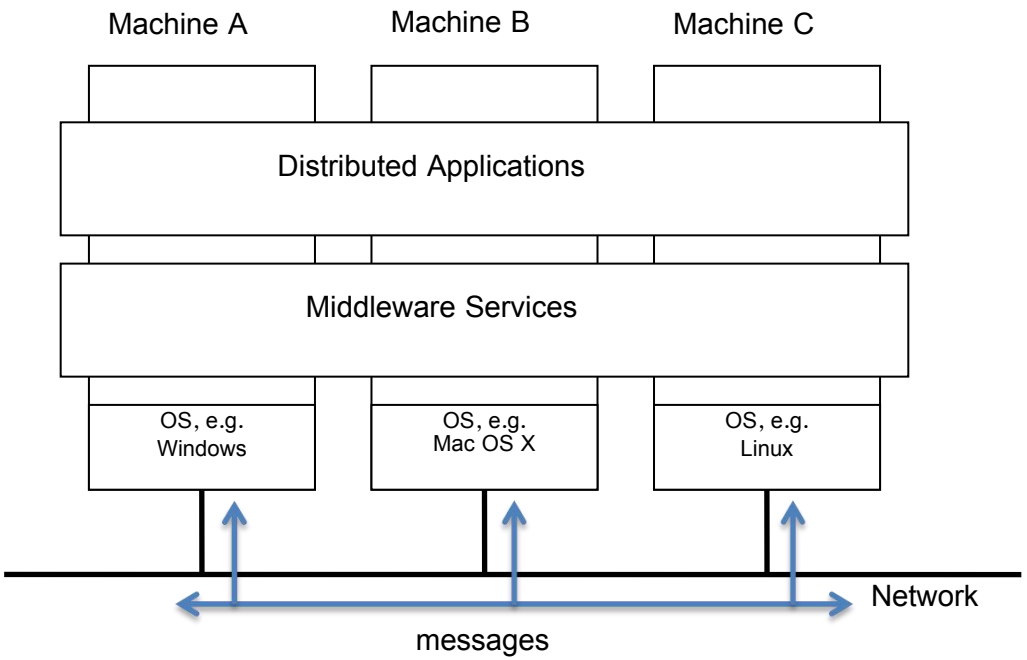
Service activator



In Nova - Batching

- ILS, Snap Service, Zenith
- Correlation ID = Request ID
- Completion = on time-out || on max count

Messages and Middleware

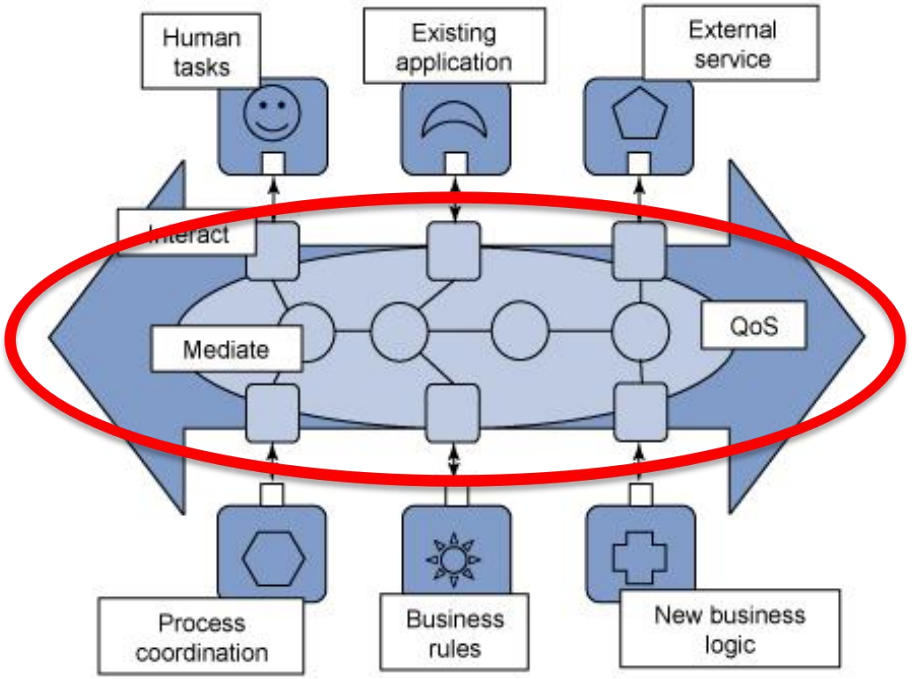


Loose Coupling

- Essential for
  - complexity management
  - Allowing change
  - Allowing growth
- Message oriented middleware helps to create loosely coupled systems
  - Based on the simplest exchange pattern (one-way)
  - Message format and metadata is independent of the components the middleware connects.

Middleware acting as an Enterprise Service Bus (ESB)

- The ESB is the glue between different components.
- It passes messages via the bus To different destinations.
- Means there is a many to one relationship between components.
- Components are glued to the bus, not each other.



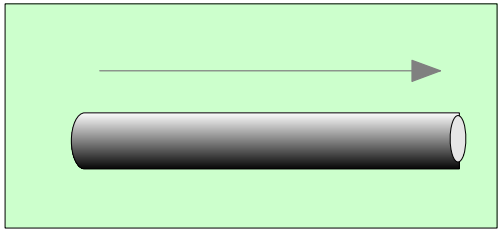
Messaging Channels

Sending a Message

- The second simple abstraction:
  - Channel

- A component that can send a message from a source to a destination
  - Could be implemented in many ways
    - ✓ Method invocation
    - ✓ TCP/IP message
    - ✓ Pigeon

Sending a Message



Channel

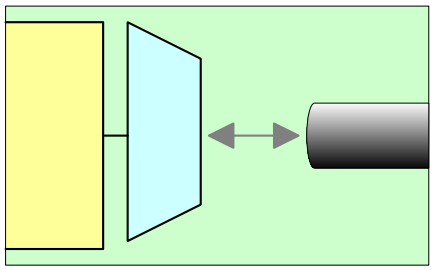
Channels are one-way

Which means communications are asynchronous

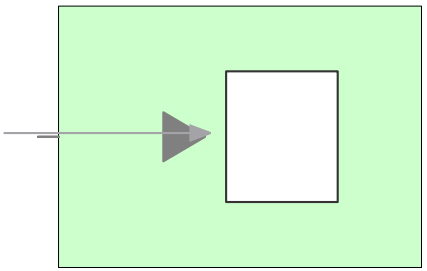
To model request/response use two channels

Binding Messages to Applications

- ✓ Applications are independent of the messaging system. So, you need adapters that are able to take application specific data and create a message and send it to a channel.
- ✓ At the other side you need Message endpoints that can receive a message from a channel and pass it to an application in an application specific way.
- ✓ These are the glue components to bind applications to the messaging system. If an application changes, then the adapter or endpoint needs changing isolates change across the system.



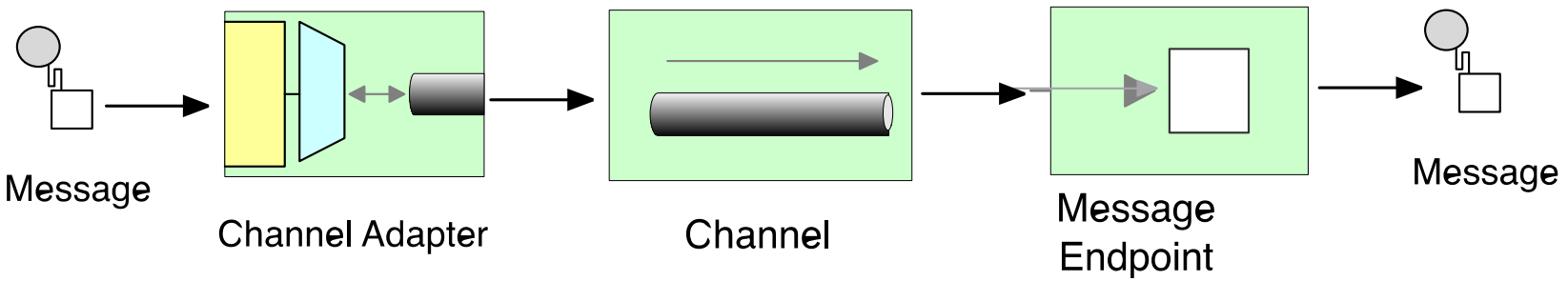
Channel Adapter



Message Endpoint

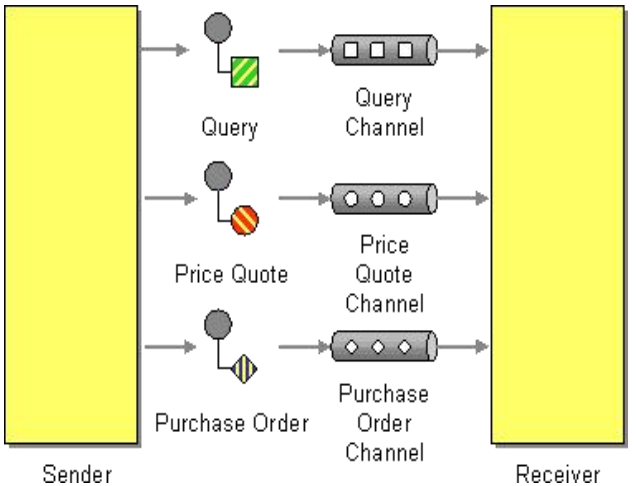
Putting it together

Message goes to adapter, then to channel, then to endpoint.

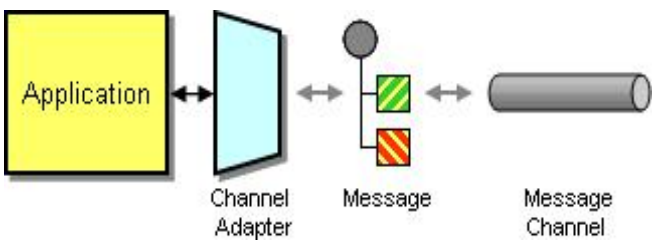


So far so simple. But we have really solved any integration problems yet

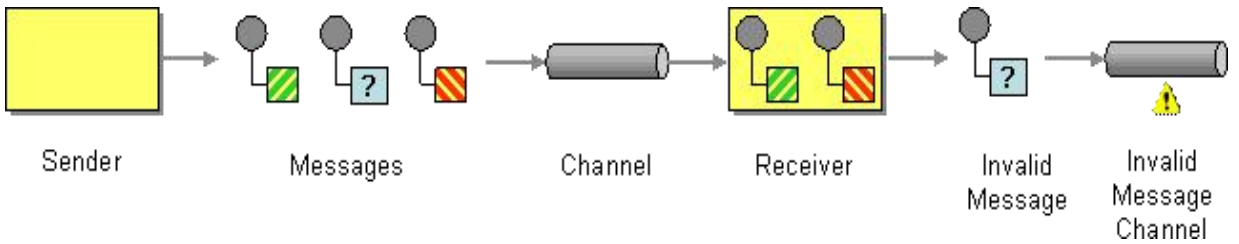
Datatype Channel



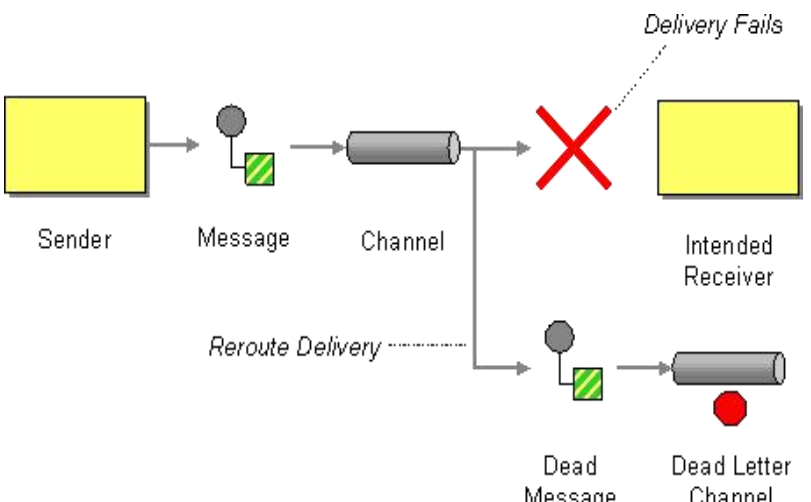
Channel Adapter



Invalid Message Channel



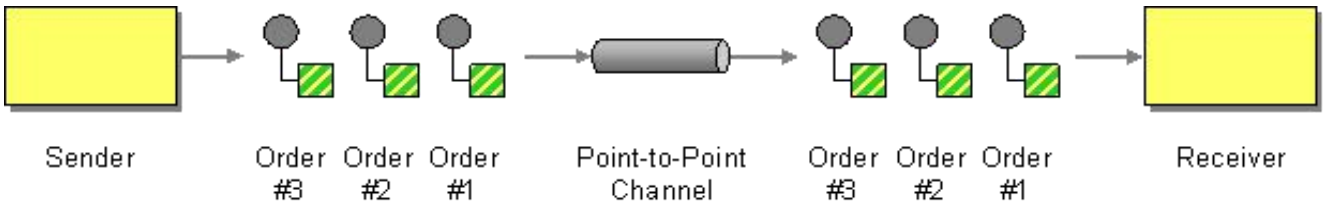
Dead Letter Channel

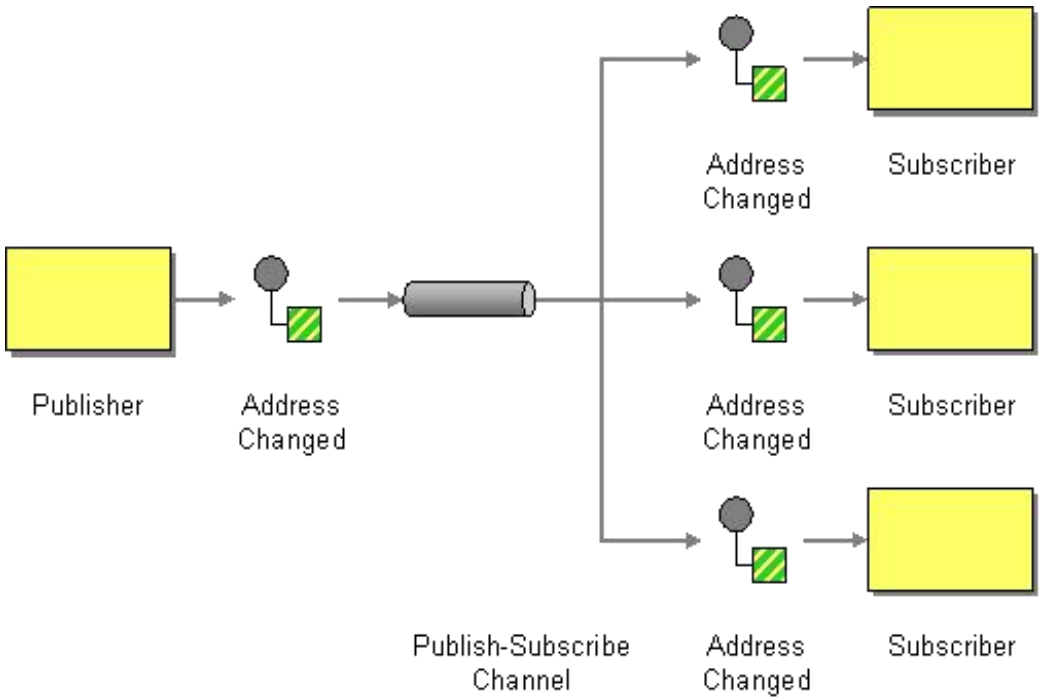


Message Exchange Patterns

- Channels are one way, so the basic building block of a messaging system is asynchronous communication.
- We have seen the pub/sub exchange pattern
- We have seen the scatter/gather exchange pattern
- The request/response is also very common
  - ✓ Web Services
  - ✓ HTTP

Message exchange styles

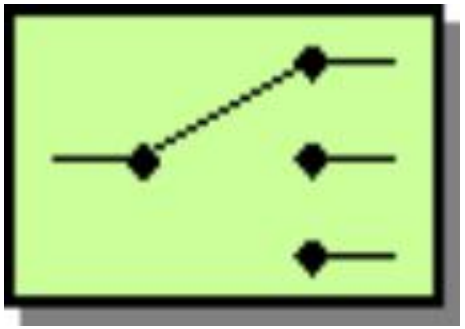




Pattern categories

- Message routing patterns
- Message transformation patterns
- Message management patterns

Message routing patterns

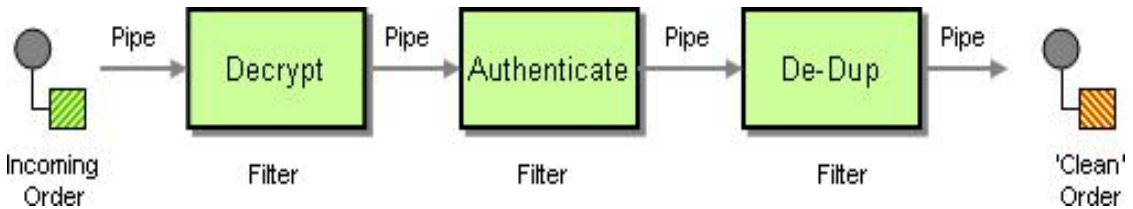


Pipes and Filters

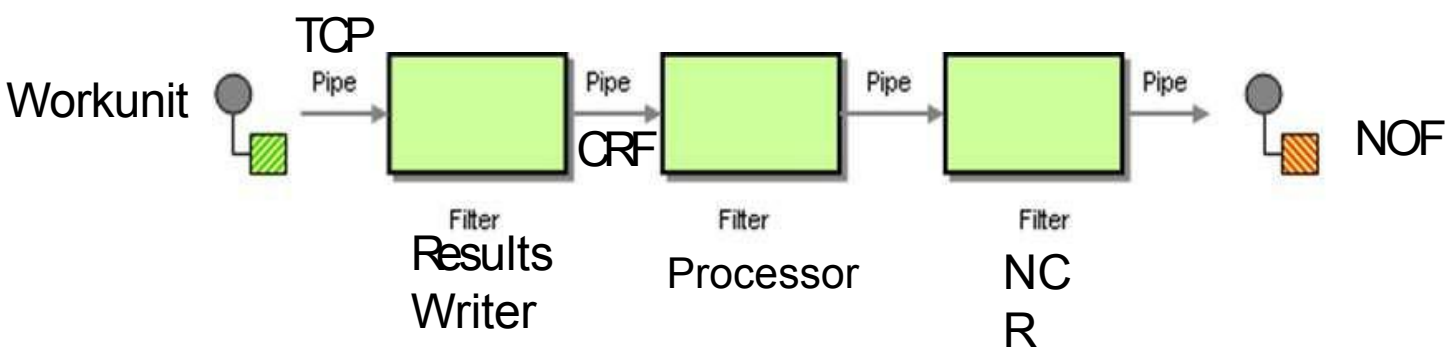
- The inclusion of the translator works for now
  - But what if there are other steps that need to take place, for example if the message has been signed and encrypted then we need to verify the signature and decrypt message.
  - Signature verification and decryption are common processes
    - We don't want to hard code these into endpoints or the translator. We want to reuse.
- A pipes and filters architecture allows us to do this.
  - The message passes through a number of components that process the message (filters)
  - They send the message down the pipe (channel) they are connected to.
  - They all deal with the same message/channel abstraction
  - They can be composed flexibly depending on the circumstance

Pipes and filters

Enterprise Integration Patterns

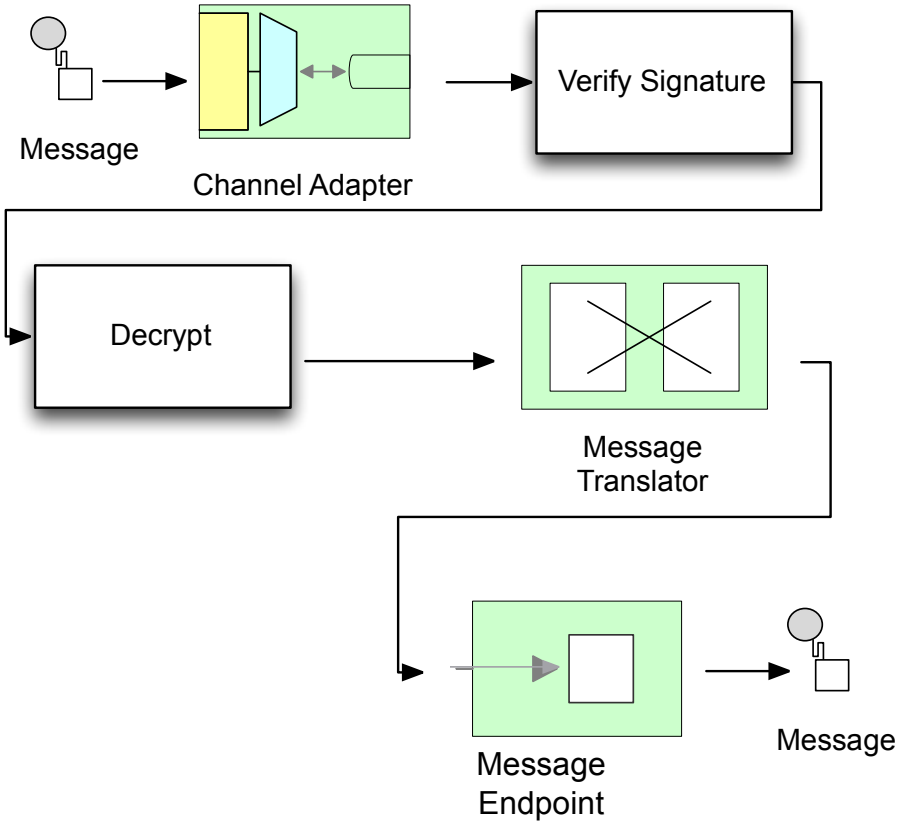


In Nova



NCRF - Non-Cancelled Request Filter

Processing Chain

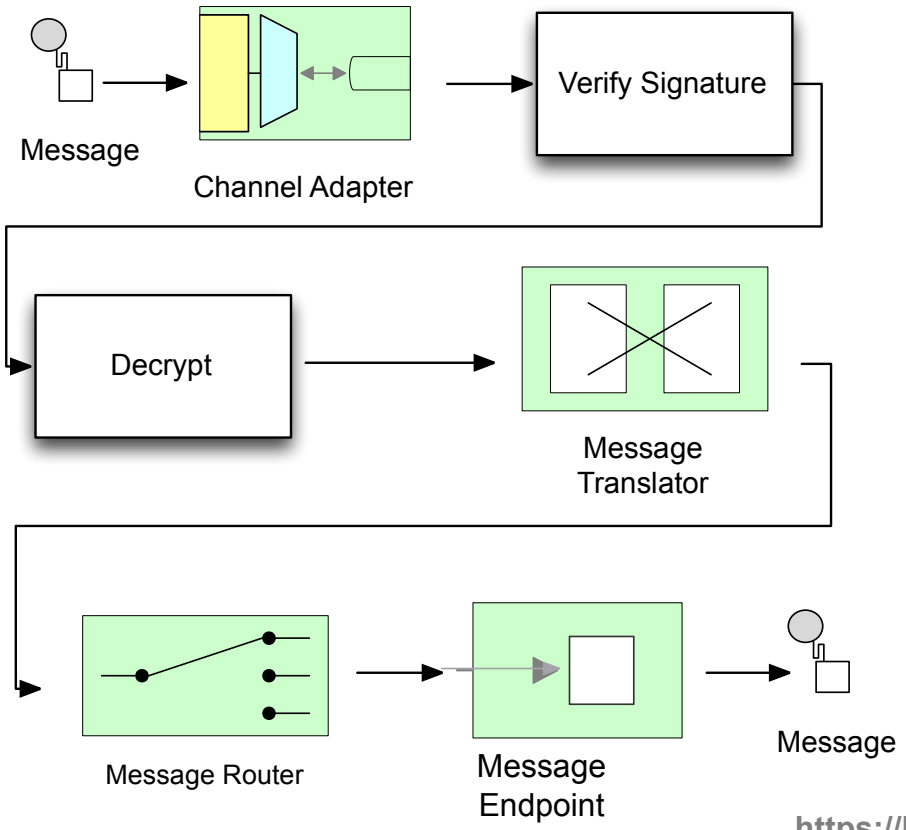


Routing

- We can now send a message down a channel and apply various processing/filtering steps.
  - ✓ Each filter is connected to an incoming and an outgoing channel
- But the endpoints are hard coded.
  - ✓ This is not flexible
  - ✓ What if an endpoint changes?
  - ✓ Introduce a Router
  - ✓ A router knows where to send a message
    - ➢ E.g. a Content-Based Router could use
      - ❖ Message type (in headers)
      - ❖ Message content (in body)
  - ✓ Or a Context Based Router
    - ➢ Receives context information from a central configuration location
    - ➢ For example a ‘testing’ context could route differently to a ‘production’ context.

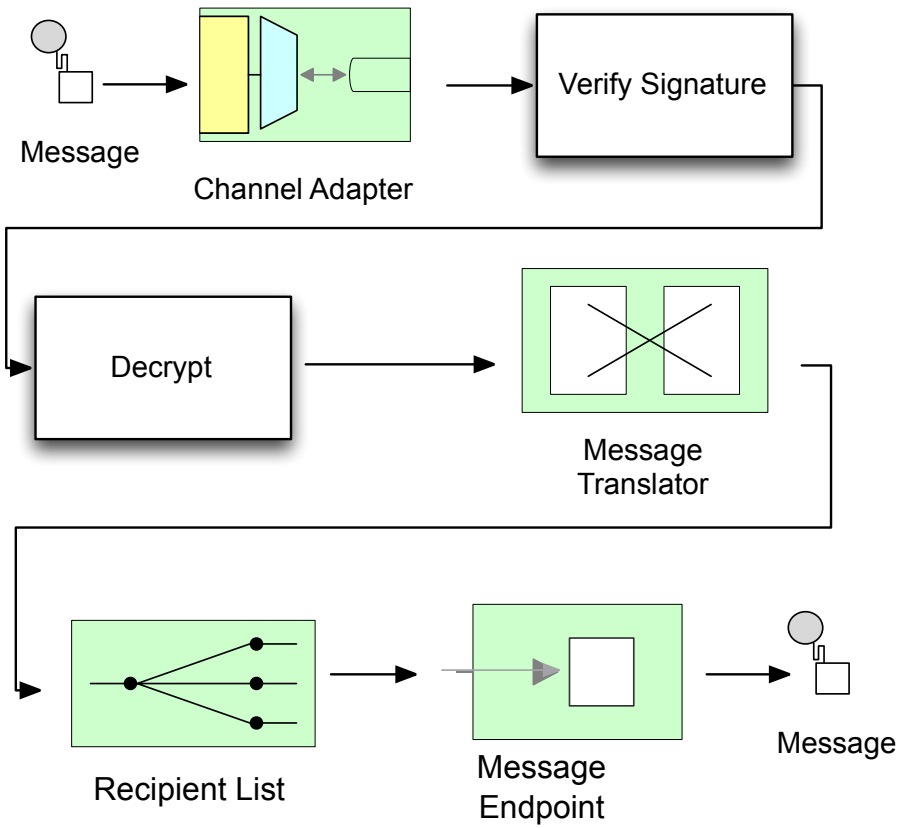
Routing

Router is connected to multiple channels & contains logic to decide which channel it should send to.



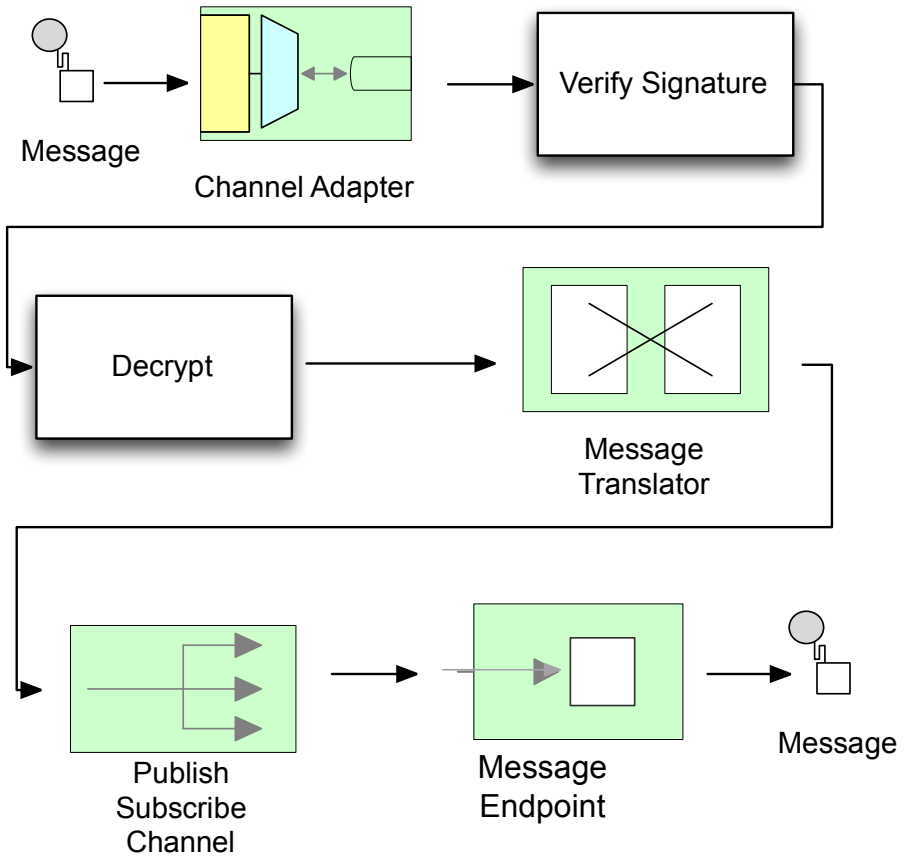
Multiple Recipients

Not all channels want to be point to point.
Here a recipient list router is used to send to a predetermined set of endpoints

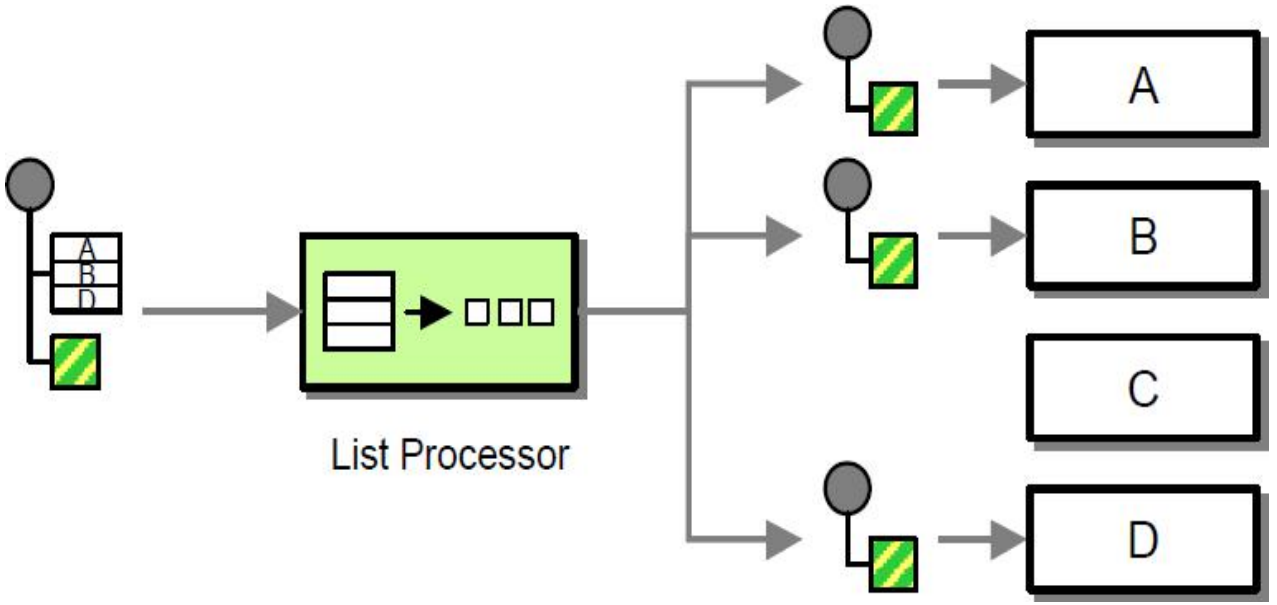


Multiple Recipients

- ✓ Here the publish/subscribe pattern is modeled.
- ✓ Endpoints subscribe to channels that are of interest to them.
- ✓ Subscribers receive messages.
- ✓ This moves the logic of knowing who should receive messages to the endpoints.
- ✓ The channel does not need to know who the subscribers are or why they have subscribed.



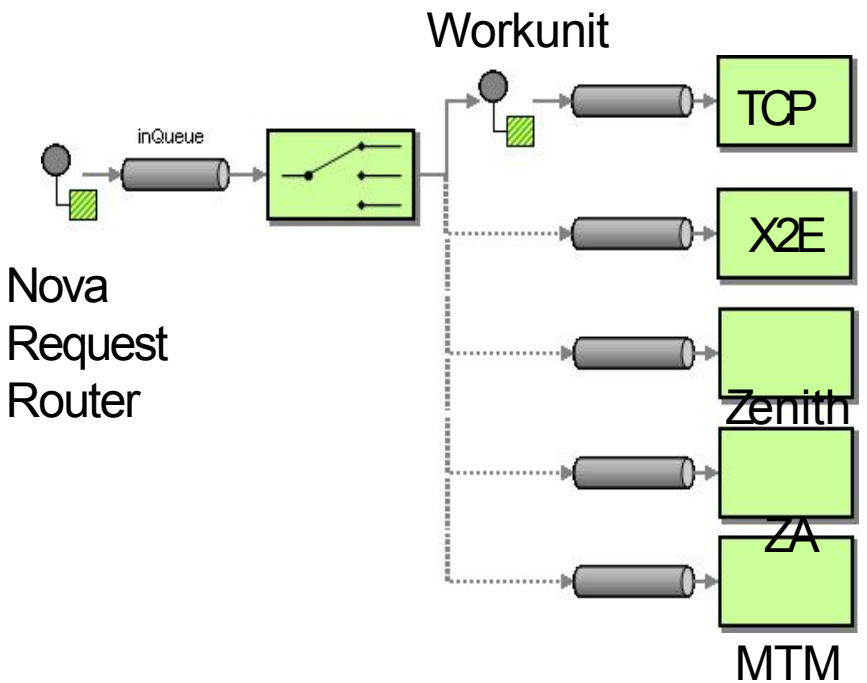
Recipient List



Routing

- A content based router and a recipient list are not that loosely coupled
  - The first needs to know how to map content to channels
    - ✓ What if either changes?
  - The latter has to know about all recipients
    - ✓ What if that needs to change?
- The pub/sub channel is more loosely coupled
  - It doesn't know or care about who gets the message
  - But there may be situations where you need to ensure that a message is only processed exactly once
    - ✓ E.g. credit card transaction

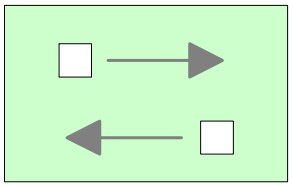
Content-Based Router



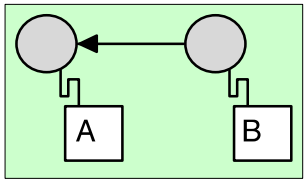
Instrument-Valuation Request

Request Response

- Channels are one-way which means communications are asynchronous
- To model request/response use two channels and a correlation ID in the messages. This potentially adds overhead. But remember that channels are an abstract pattern.
- You could implement a channel that returns data down an open TCP connection if request/response is required and an open connection is available.
- This is actually not uncommon. e.g. a channel that uses HTTP (in built request/response) will often use an HTTP Accept response header for asynchronous communications.
- For request/response messages it can send data down the open socket and pass the data to the message endpoint which experiences it as asynchronous.



Request
Reply

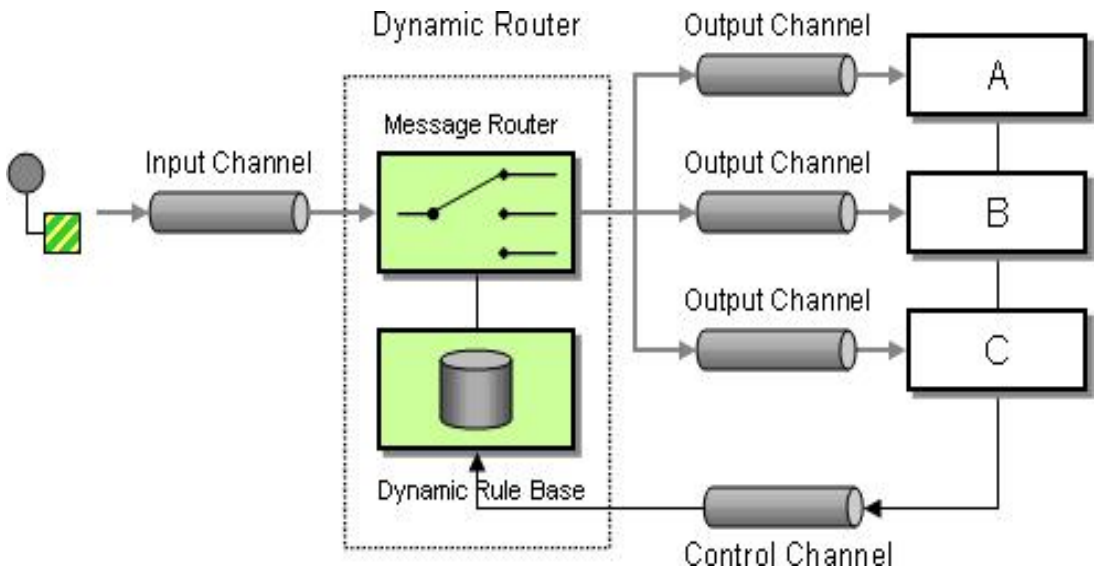


Correlation
ID

Dynamic Routing

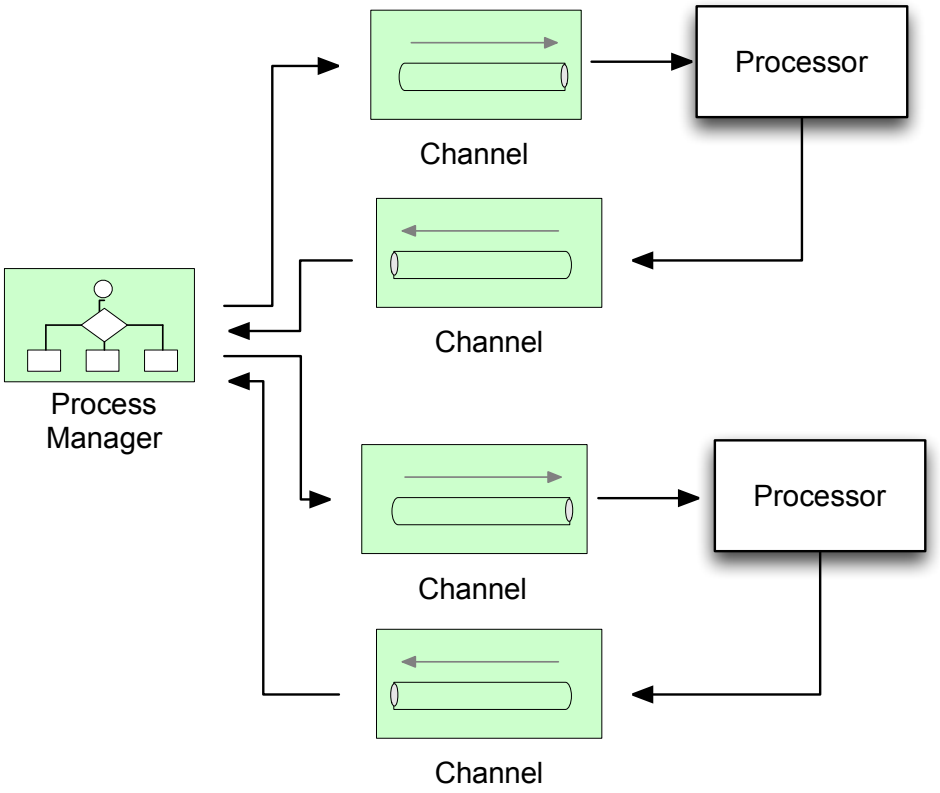
- In the content based router example, some steps are hard coded and routing decisions are made after some pre-processing.
  - ✓ To make it more flexible, we could move the router to the beginning of the route. But the route is still hard coded in the router.
- In more sophisticated routes, you may want to change the routing path as you move through the route.
  - ✓ The processing done by one filter may affect how later filter processing is done.
  - ✓ For example error processing
  - ✓ Routes that encapsulate one or more coarse grained business level functions that connect various underlying subsystems together.

Dynamic Router



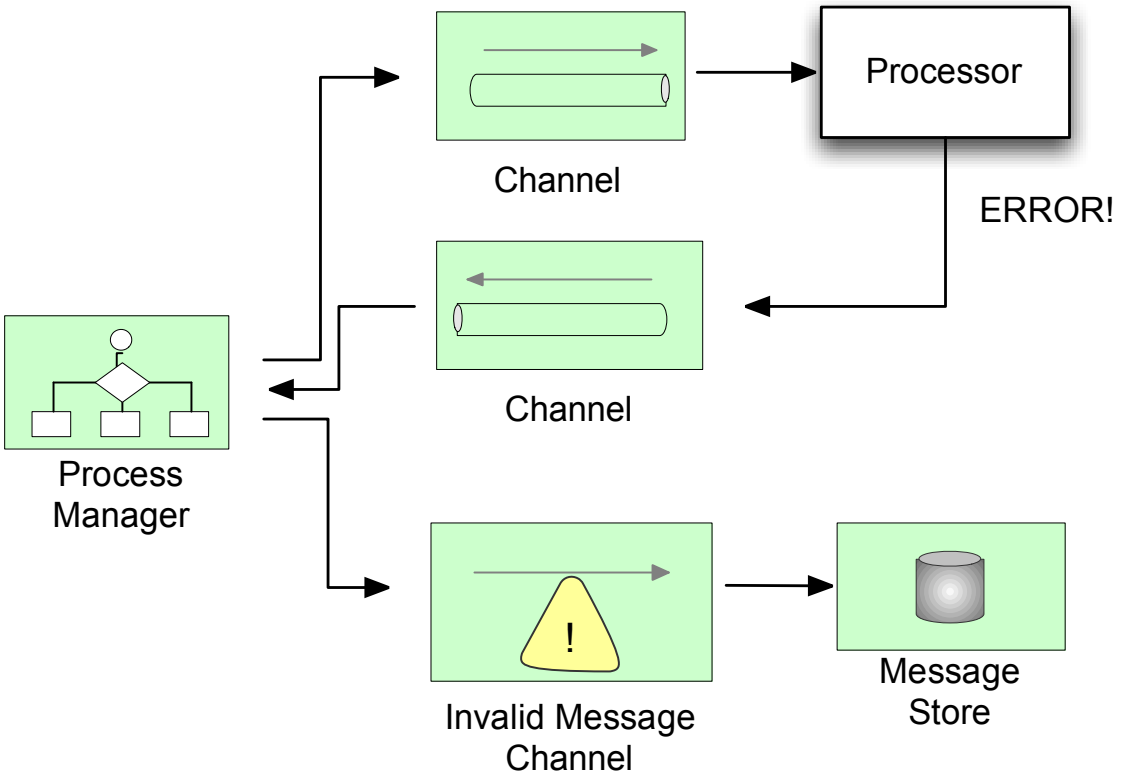
Process Manager

- Process Manager sends to channels and each component returns results to the Process manager.
- One central decision point that can evaluate at runtime.
- But messages always go back to manager which increases overhead.

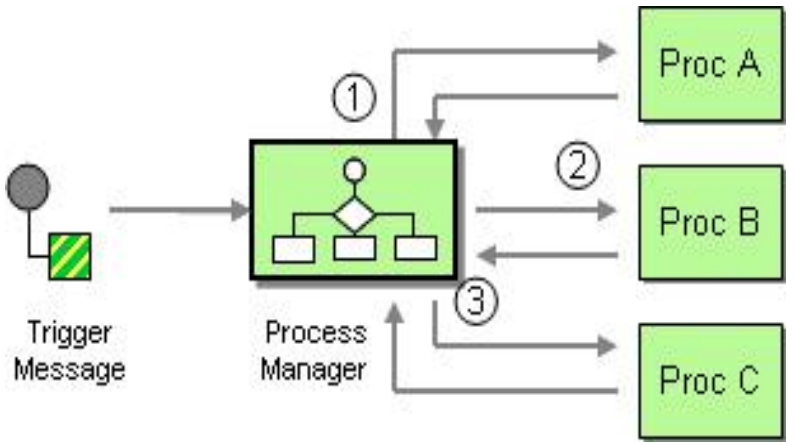


Process Manager

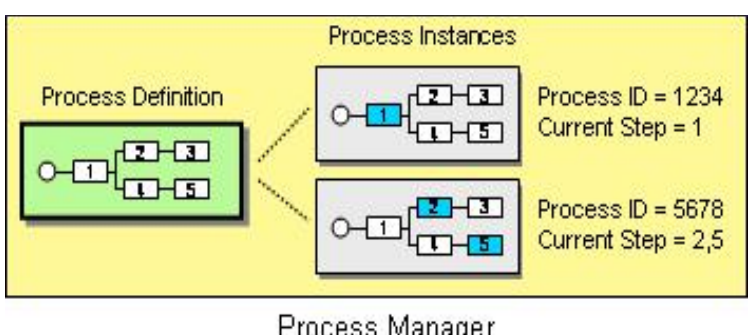
Exception Handling using Process Manager

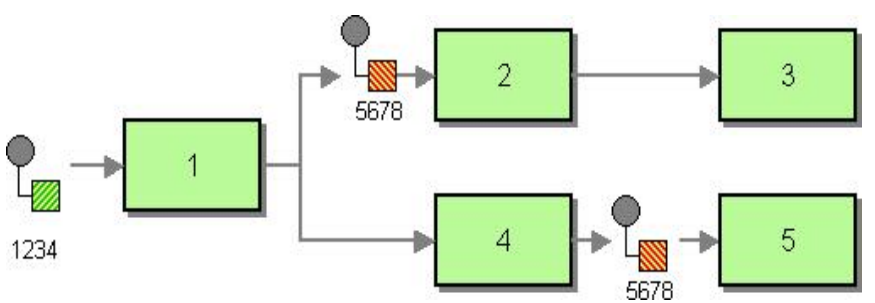


Process Manager



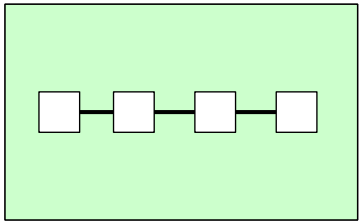
Complex message flow





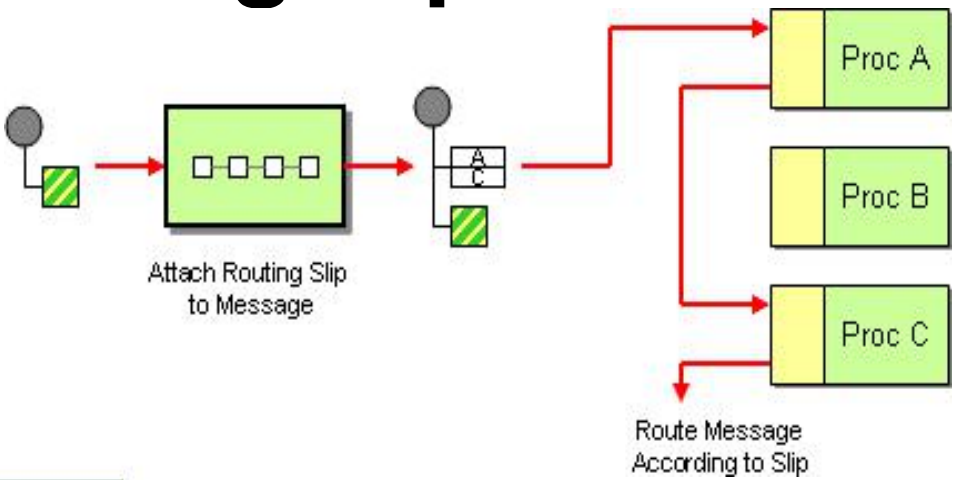
Routing Slip

- This allows each processor to pass the message to the next processor without going back to a central unit.
  - ✓ The processing chain is calculated up front
  - ✓ The ‘slip’ is the calculated route. It is put into the message metadata
  - ✓ Filters have small generic routers attached to them
  - ✓ These mini routers update the slip, then pass the message to the next channel in the list
- Can be used for service orchestration
  - ✓ Multiple underlying services performing a higher level business function
- Downside: route cannot be altered after it has been initialized

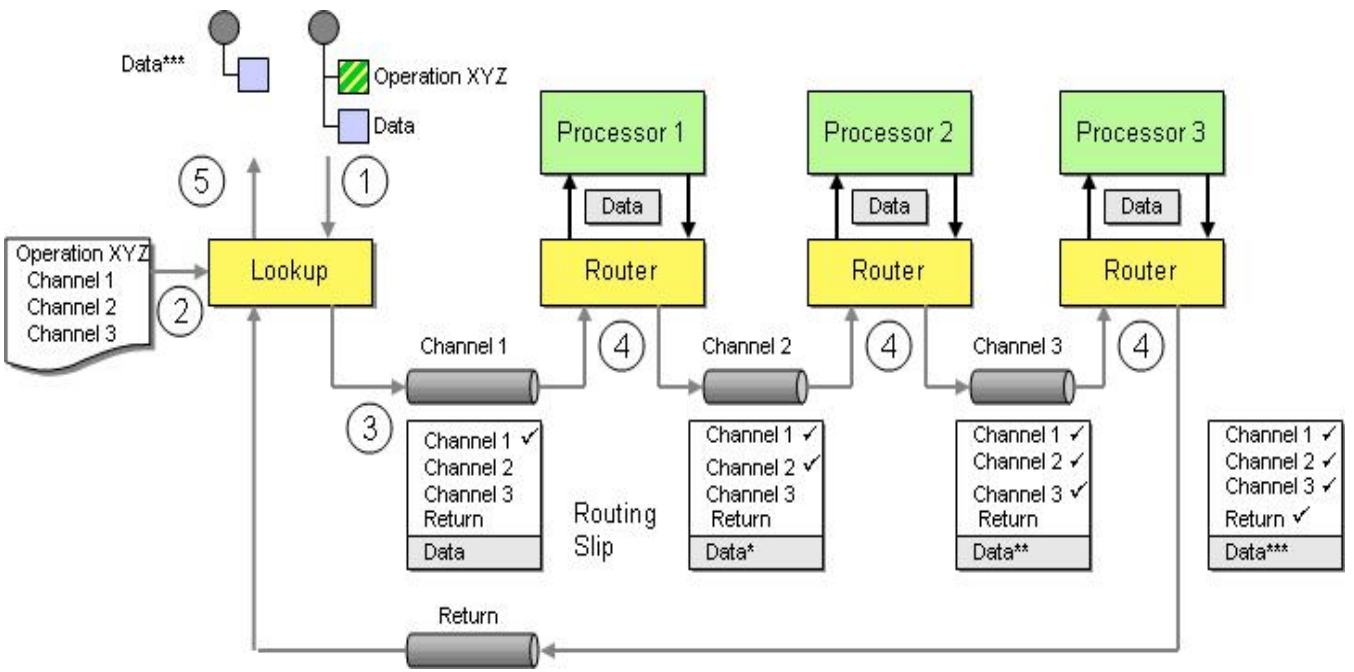


Routing Slip

Routing Slip



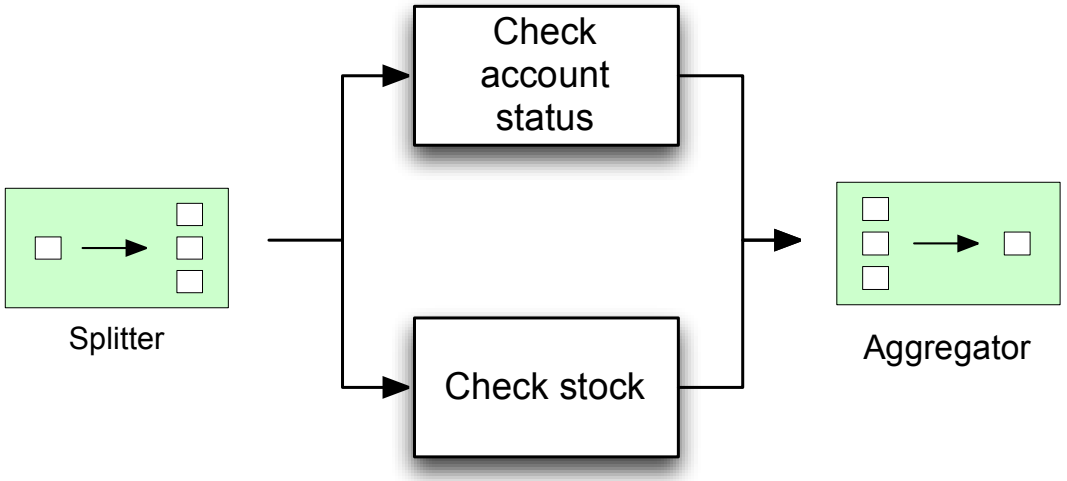
Linear flow



Scatter Gather

- Rather than defining long sequential processing chains, it may be possible to execute steps in parallel
  - ✓ e.g. you could check a user's account status and whether the item they have ordered is in stock at the same time.
  - ✓ These processes are not dependent on one another so they can be executed in parallel
  - ✓ The results of both processes can be aggregated and decisions made about how to proceed.

Splitter and Aggregator

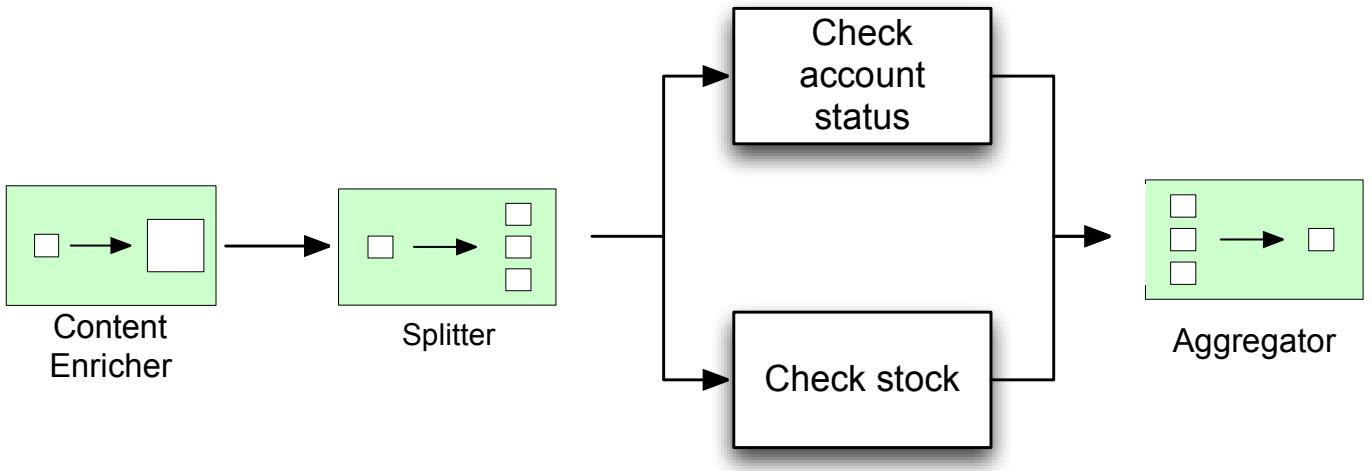


Scatter Gather

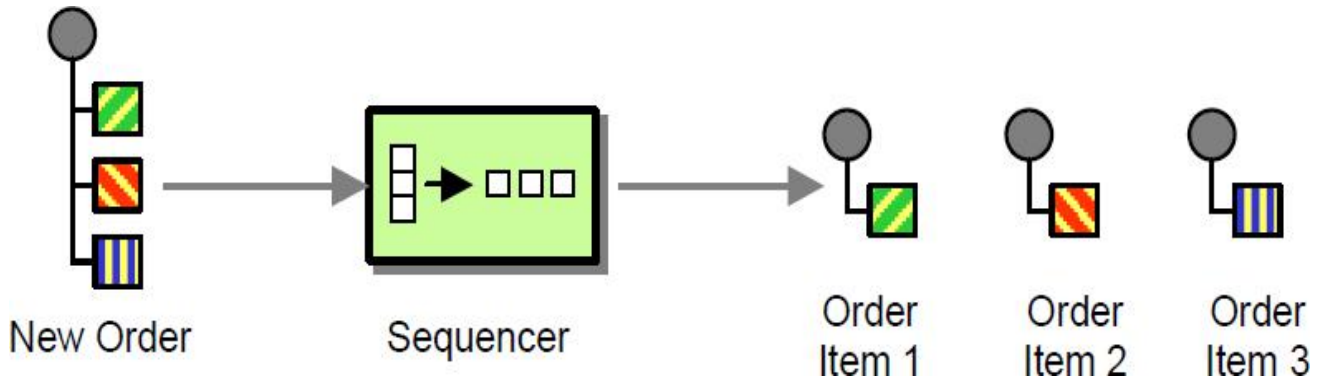
- To facilitate this, metadata could be added to the message header to define the processes that need to be completed and processing status values.

  - So the aggregator knows
    - ✓ When all processes have finished
    - ✓ What actions to take based on the return values of the processors

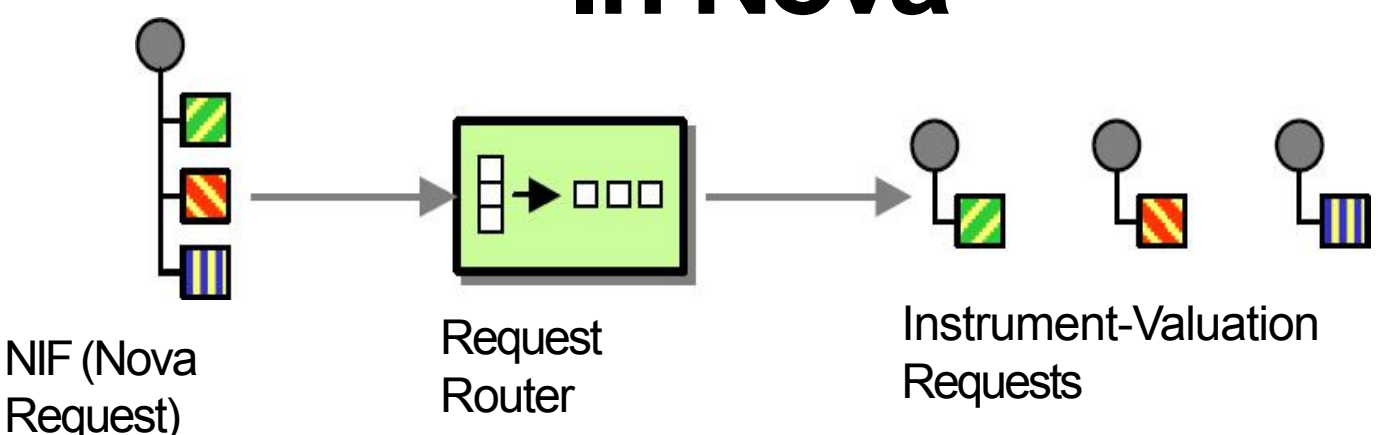
Splitter and Aggregator



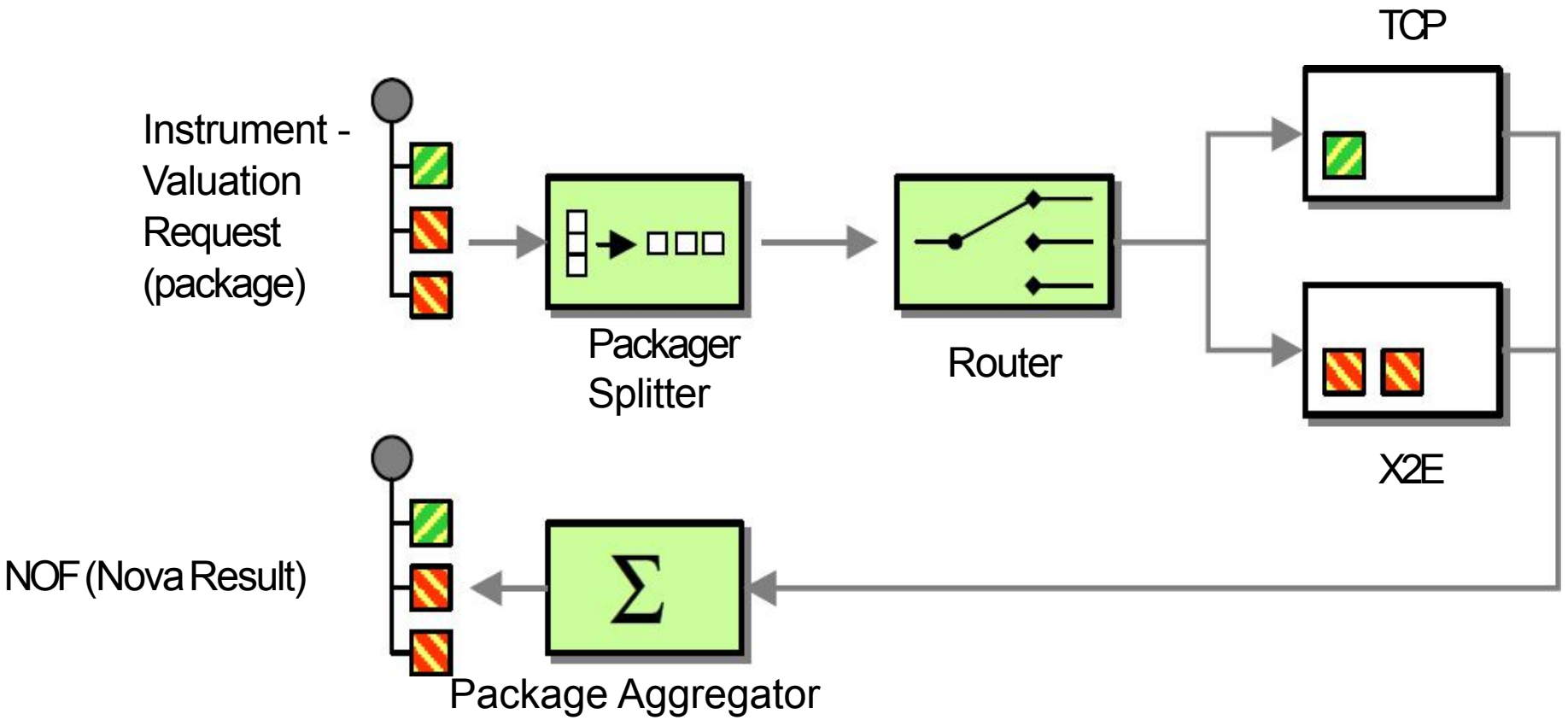
Sequencer (Splitter)



In Nova



Aggregator

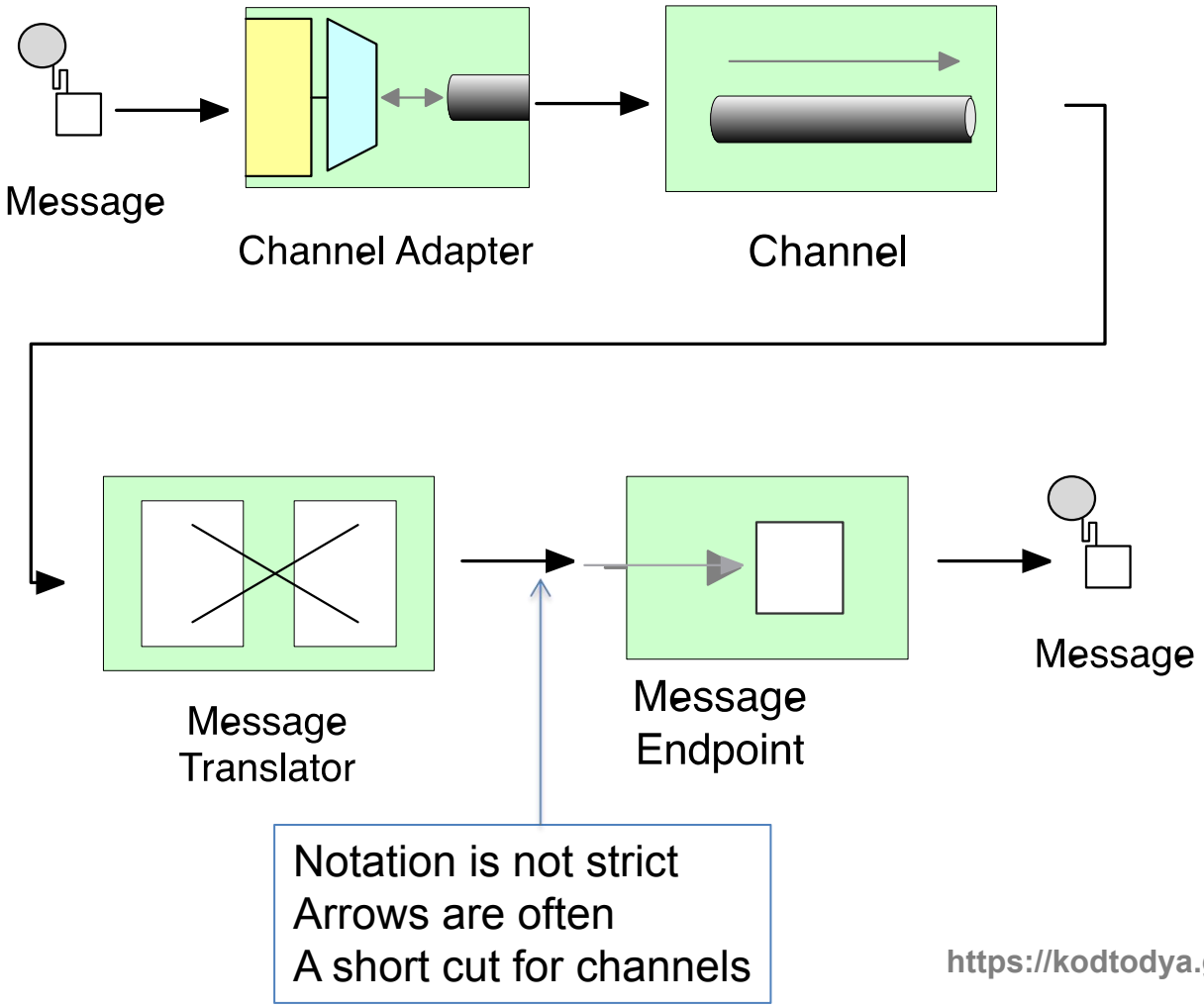

TCP
Instrument - Valuation Request (package)
Packager Splitter
Router
X2E
NOF (Nova Result)
Σ
Package Aggregator

Message Transformation Patterns

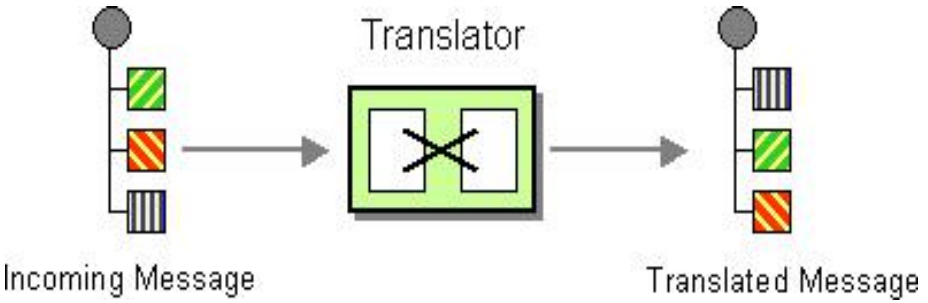
Transformation

- ✓ Sending and receiving applications may expect different data types or formats.
- ✓ So we may need to transform the message somehow to suit the receiving application.

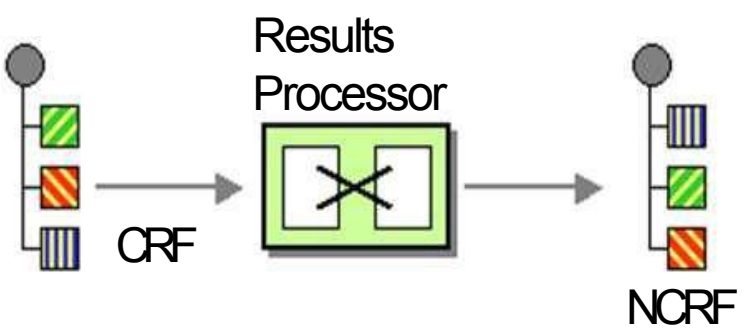
Transformation



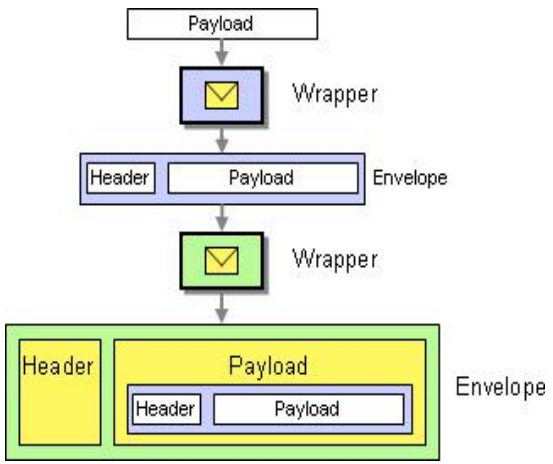
Message Translator

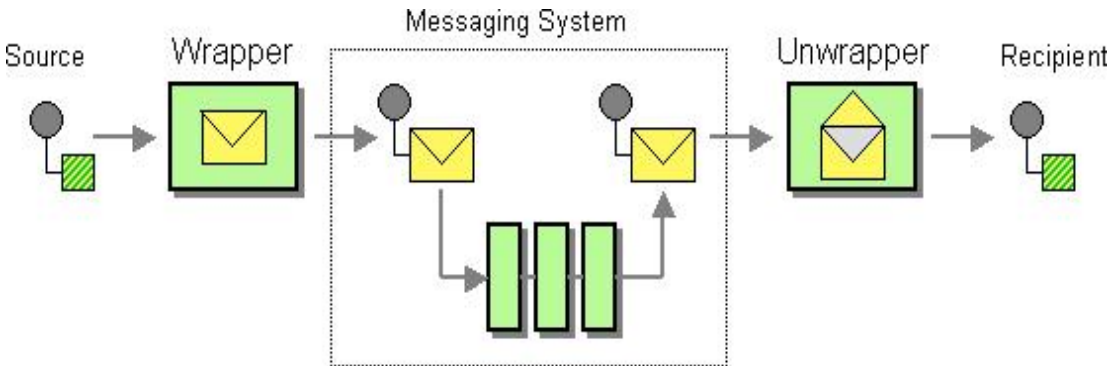


In Nova

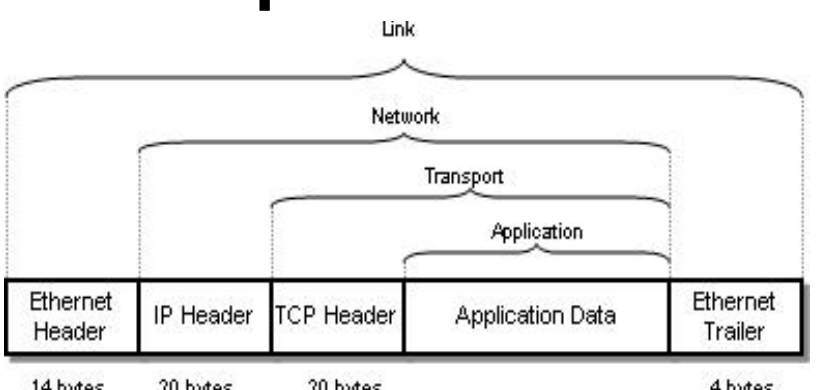


Envelope Wrapper / Un-wrapper

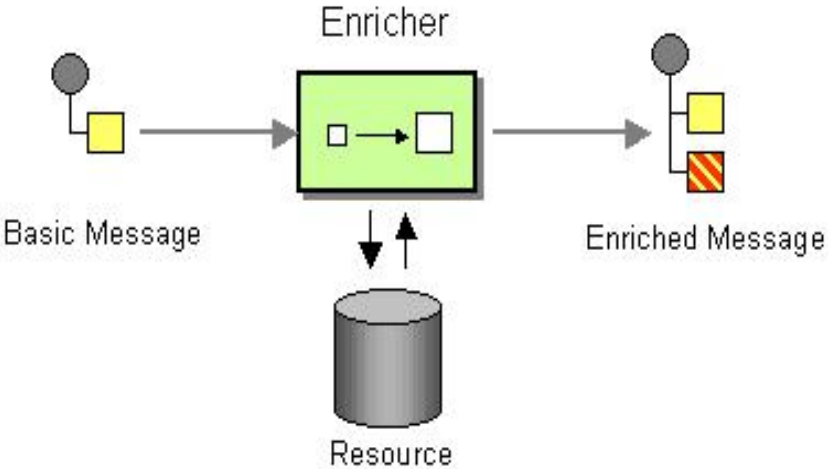




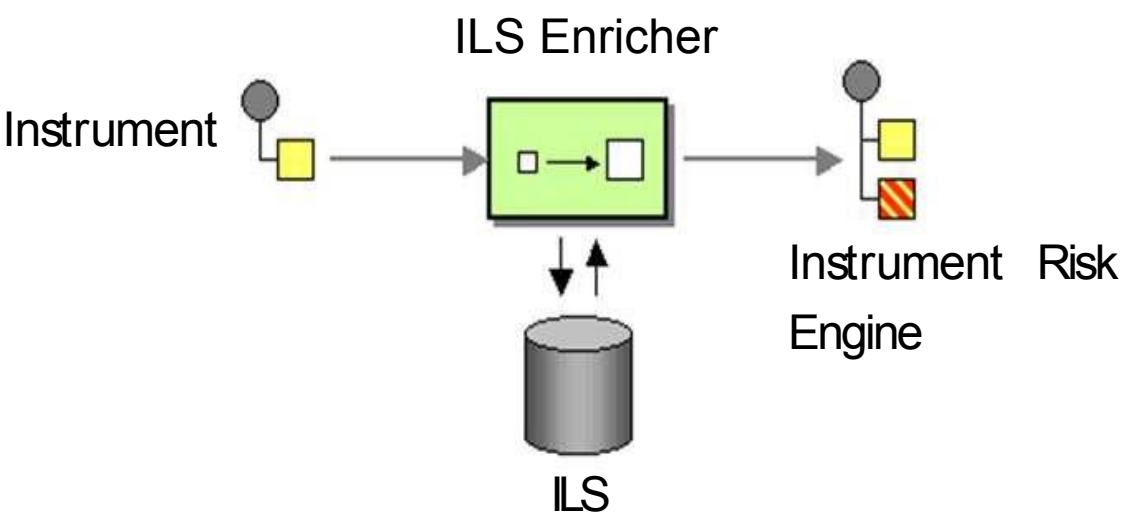
Example: TCP/IP



Content Enricher

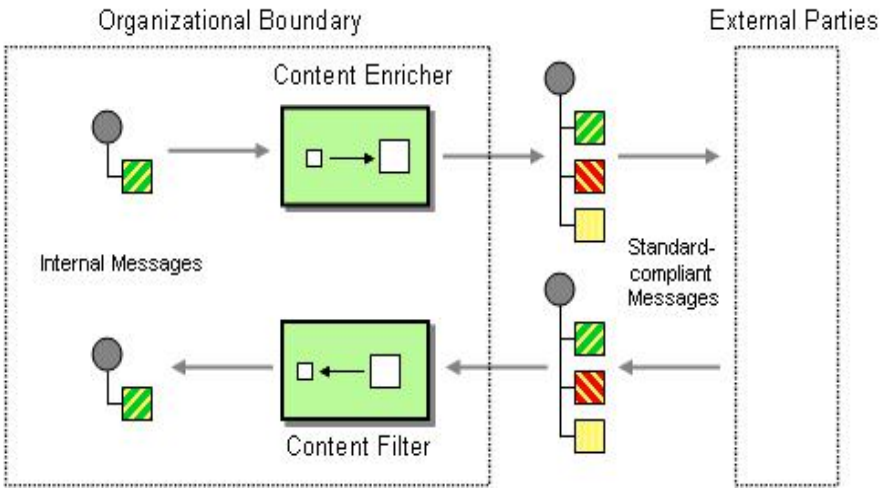


In Nova

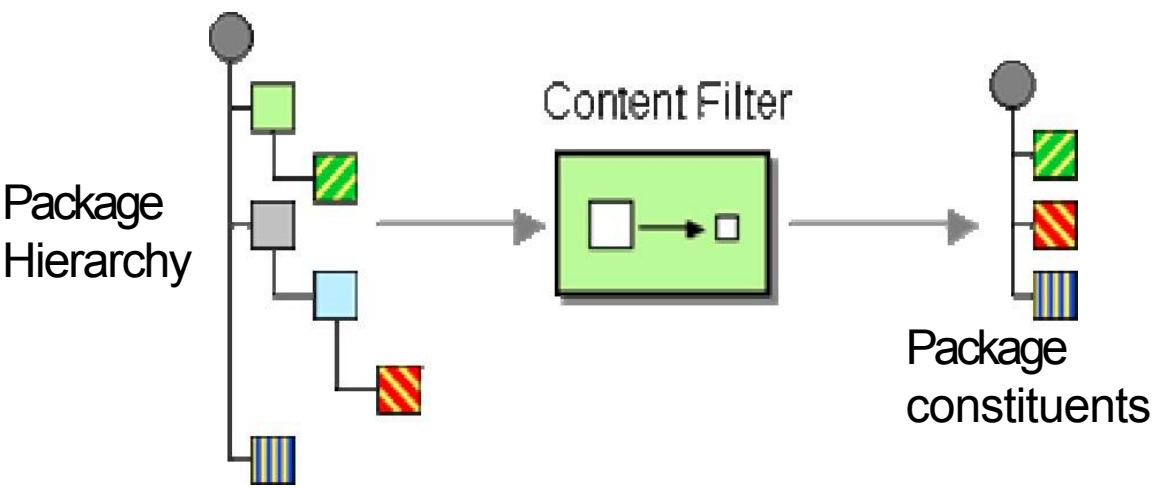


Content Filter

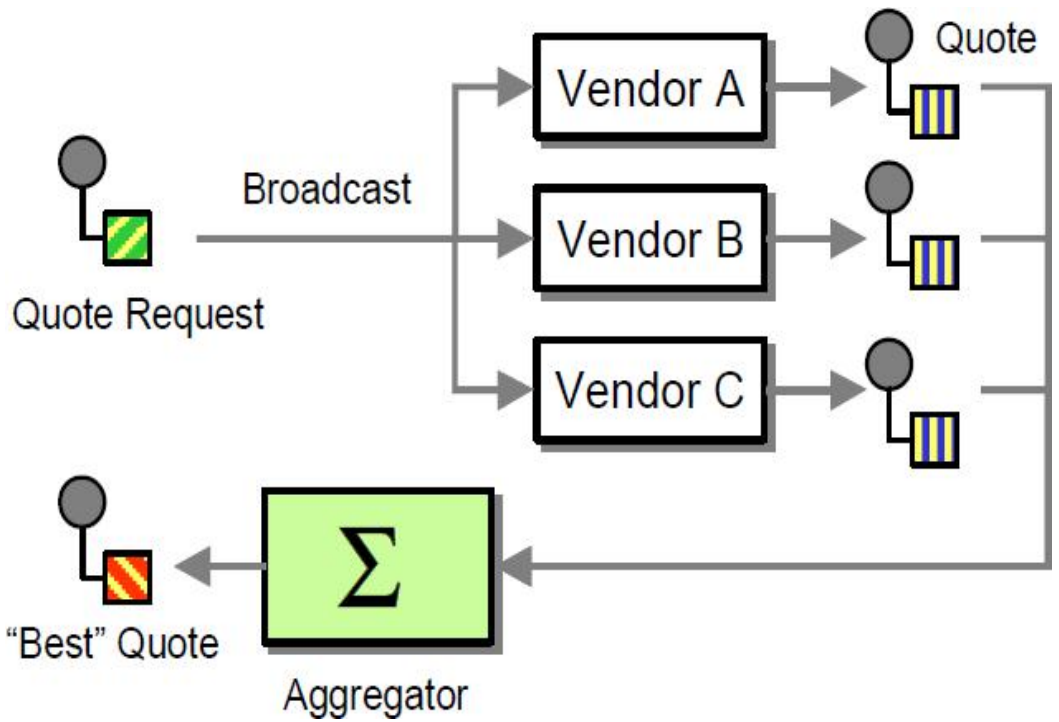
Enterprise Integration Patterns



In Nova - Message Flattening



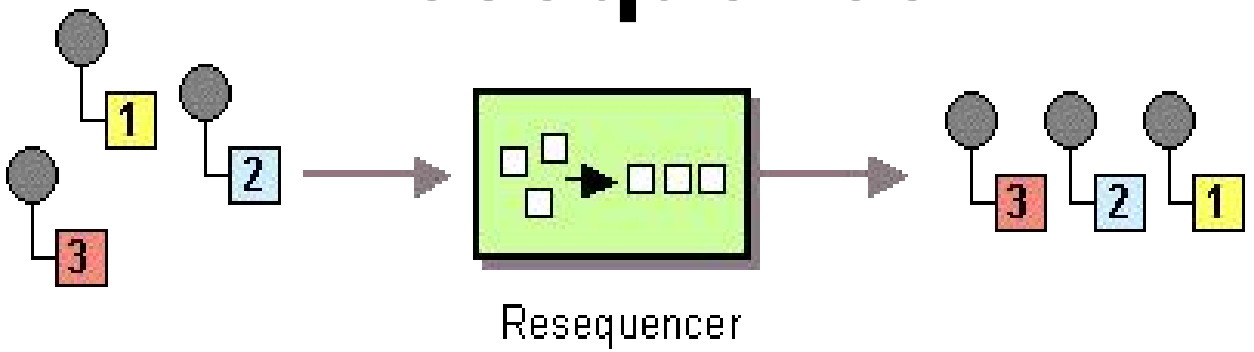
Broadcast with Aggregate Response

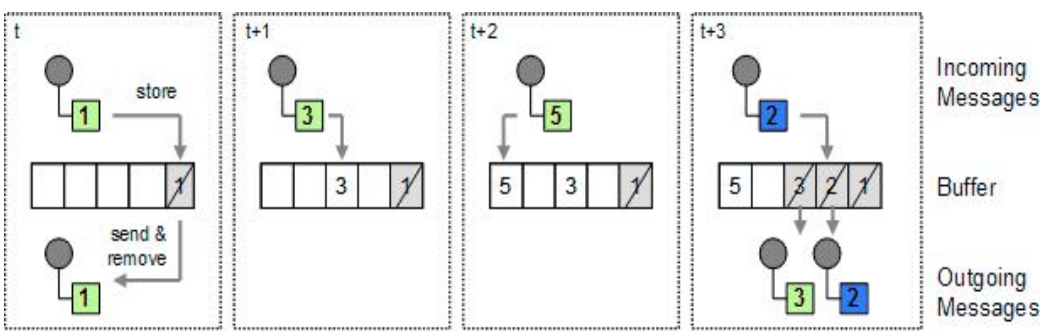


Completion criterion:

- Timeout
- Count
- External event

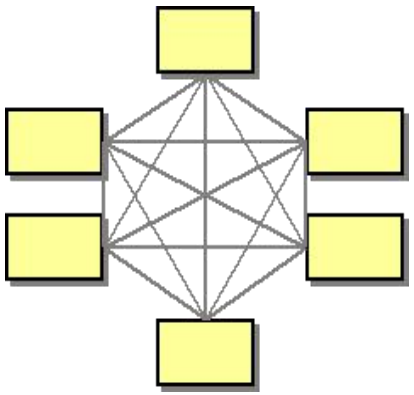
Resequencer



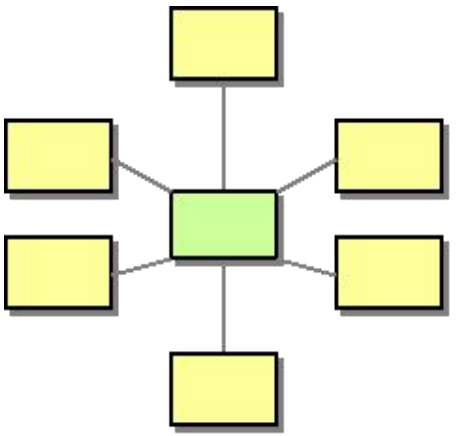


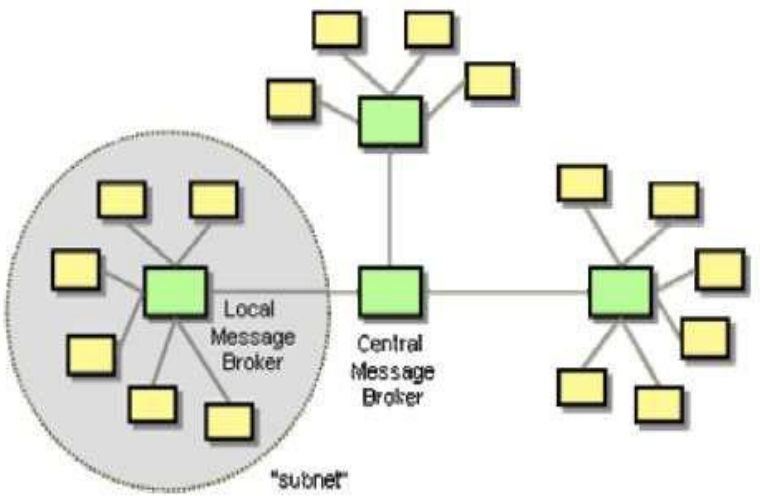
Example – TCP datagrams

Message broker



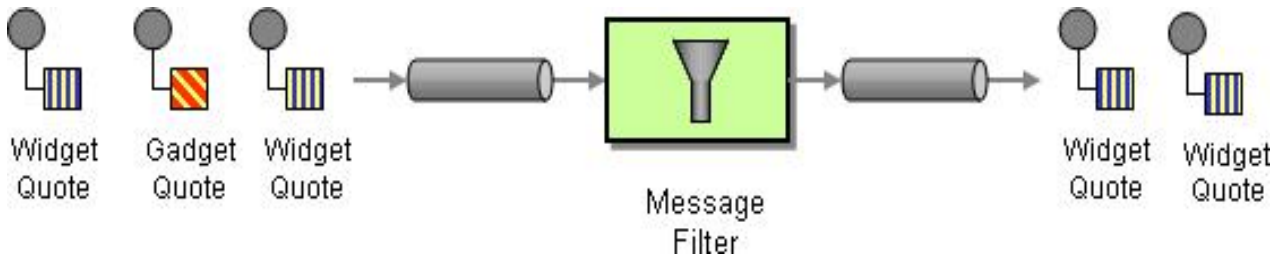
Integration Spaghetti as a Result of Point-to-Point Connections



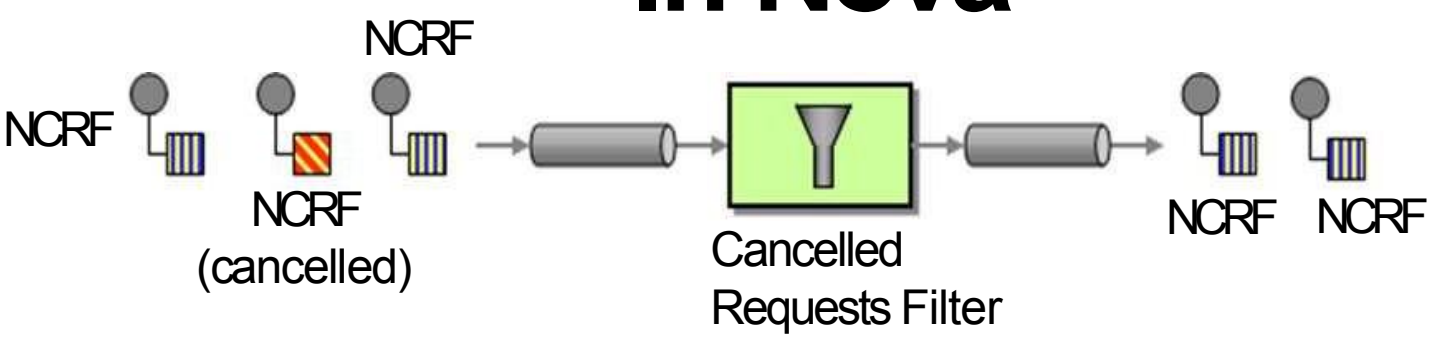


A Hierarchy of Message Brokers Provides Decoupling while Avoiding the "Über-Broker"

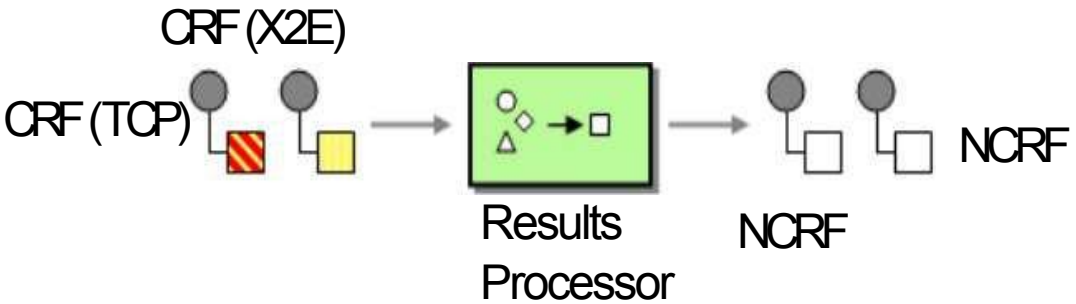
Message Filter

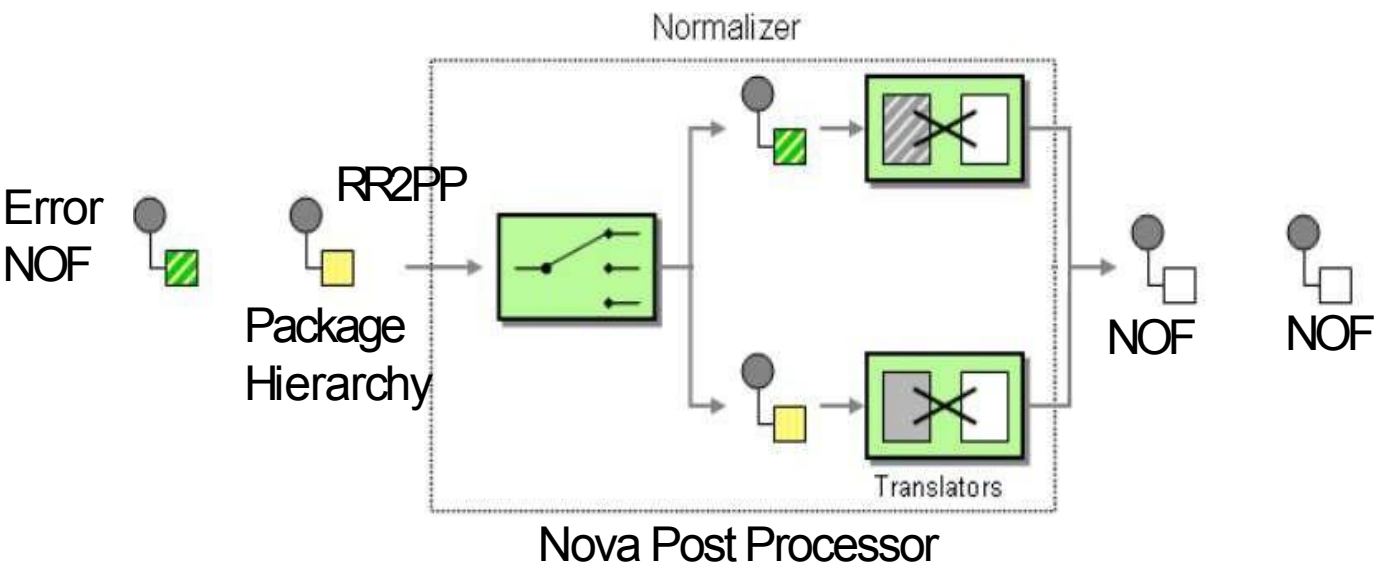


In Nova

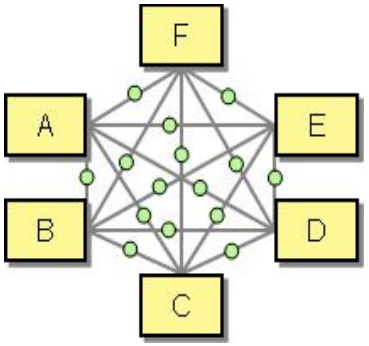


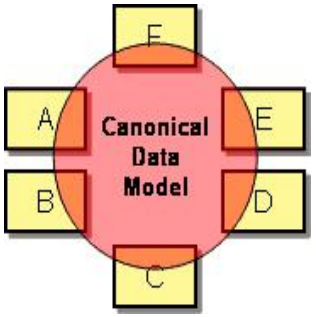
In Nova

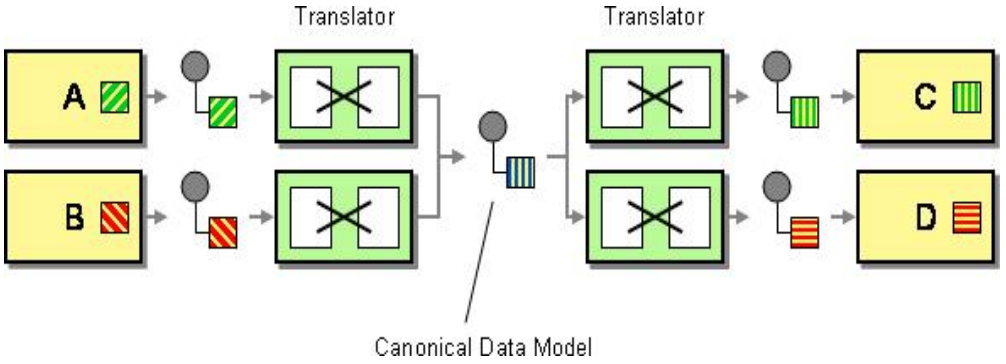

CRF (X2E)
CRF (TCP)
Results
Processor
NCRF
NCRF


Normalizer
Error
NOF
RR2PP
Package
Hierarchy
Translators
NOF
NOF
Nova Post Processor

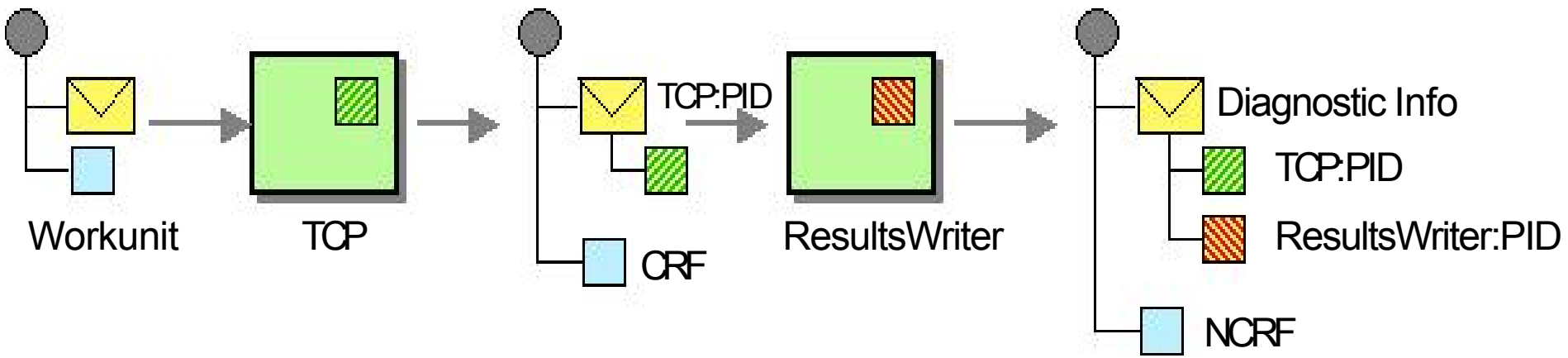
Canonical Data Model



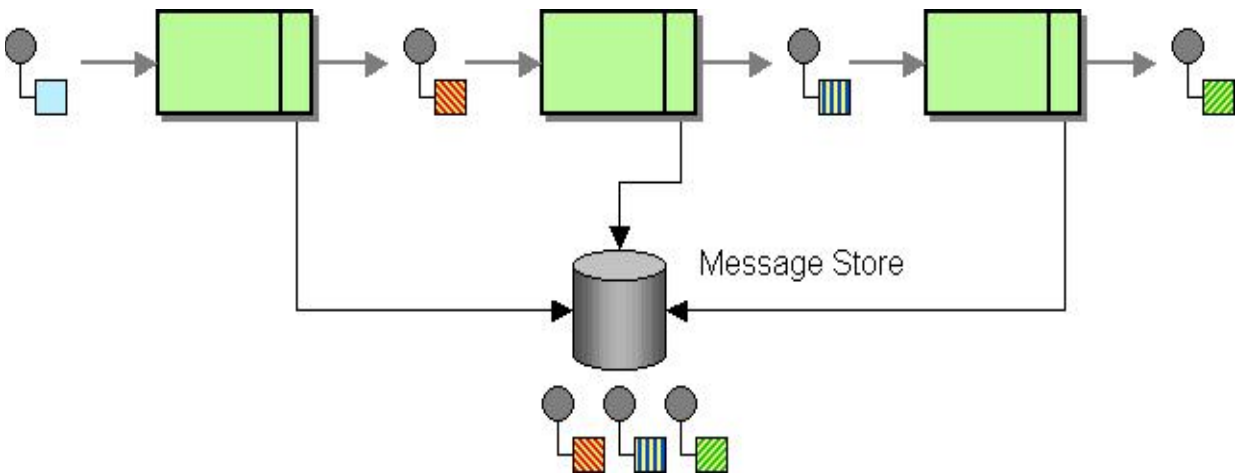




Message History

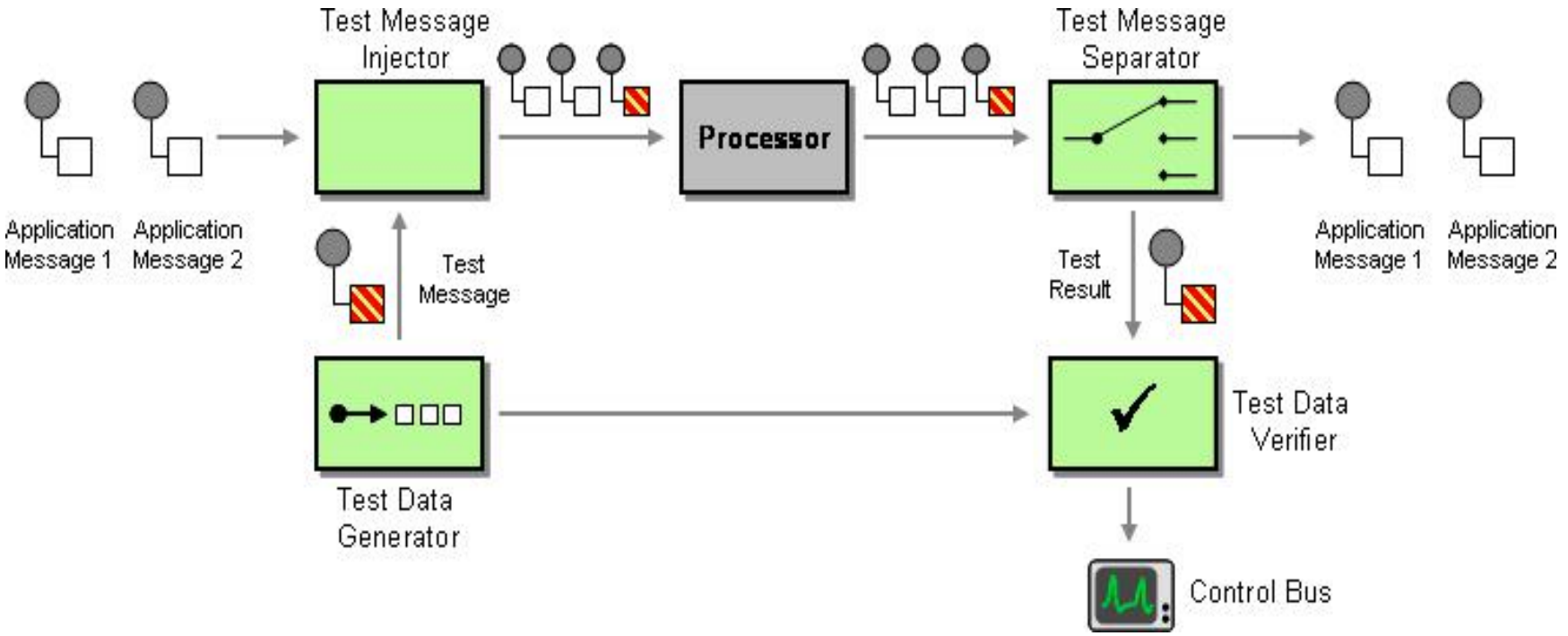


Message Store

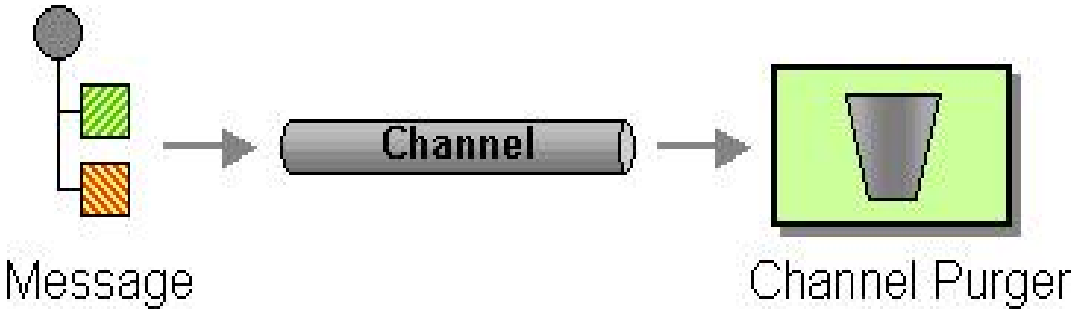


1) Xvis, Instrumentation DB
2) Message logging to disk in Nova

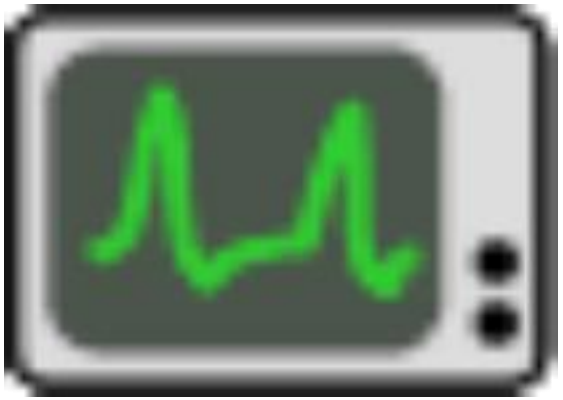
Test Message



Message Purger



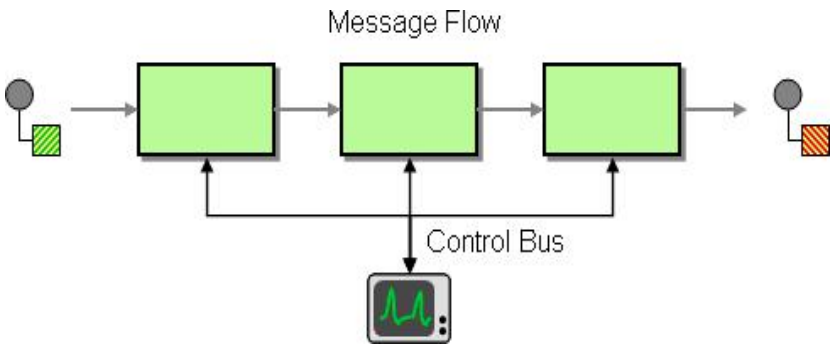
System Management Patterns

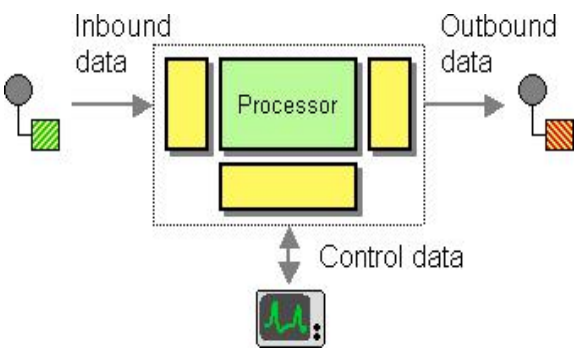


System Management

- Message oriented systems are loosely coupled
  - But that can make debugging and verifying message flow very difficult especially when distributed across many machines and service
  - There are various techniques for coping with this that use the messaging framework itself to control and monitor.
  - Two levels of monitoring
    - ✓ System level – message processing time, throughput
      - Look at most at some of the metadata in message
    - ✓ Business application monitoring
      - Look at the payload of the messages

Control Bus





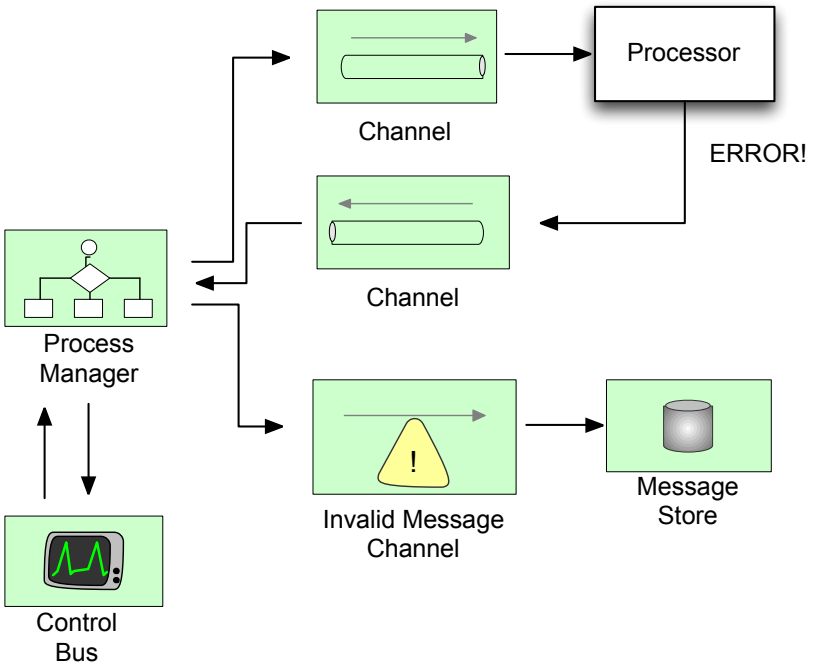
- Configuration
- Heartbeat
- Test Messages
- Exceptions
- Statistics
- Live Console

In Nova:

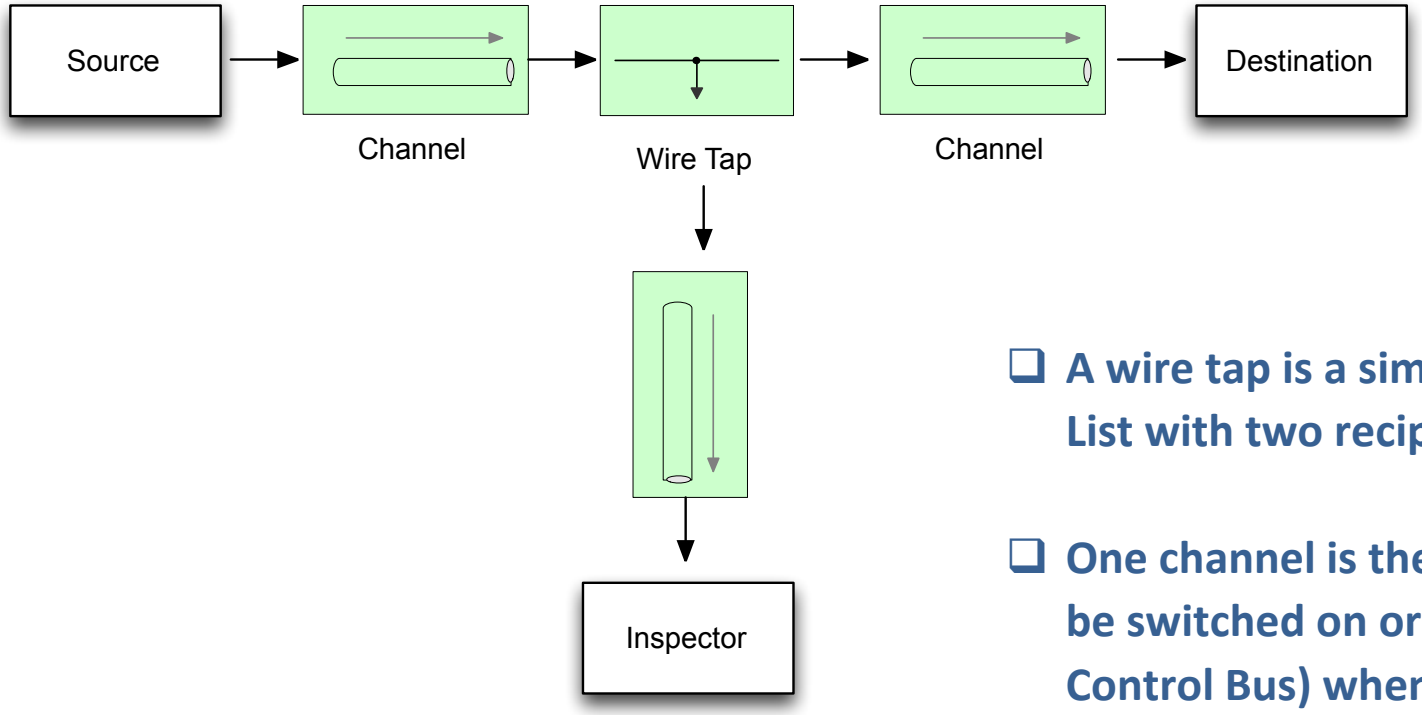
- Request Router - configuration
- Progress Monitor - Statuses

Control Bus

- Process Manager revisited, this time with a channel to a control bus.
- The control bus receives messages created by the Process Manager in response to the messages that it processes.
- Allows monitoring exceptions, throughput, processing time, keeping a message history.
- A separate channel from the control bus to the process manager allows runtime configuration of the manager.

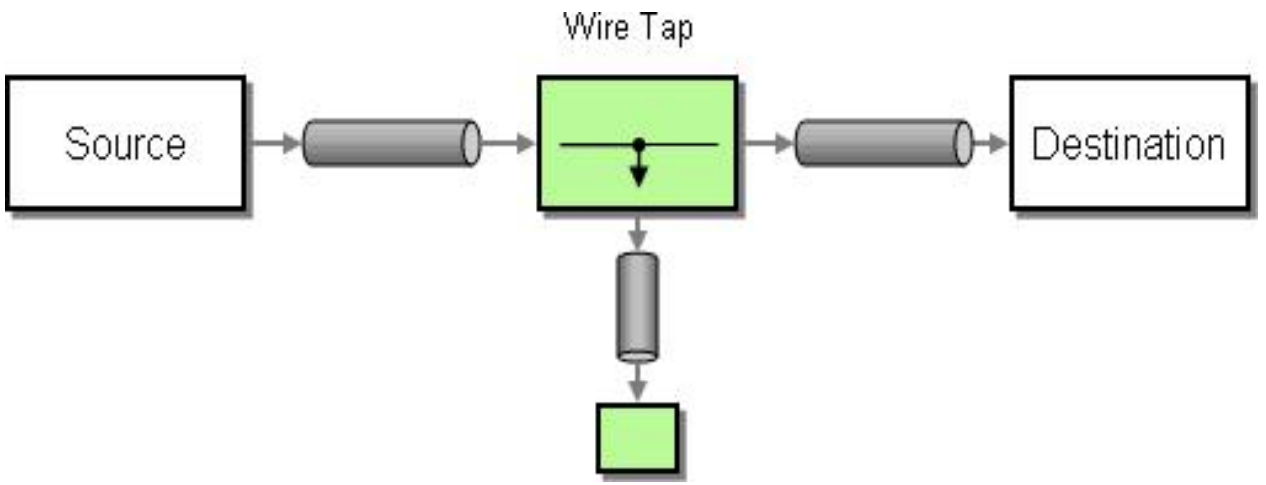


Wire Tap



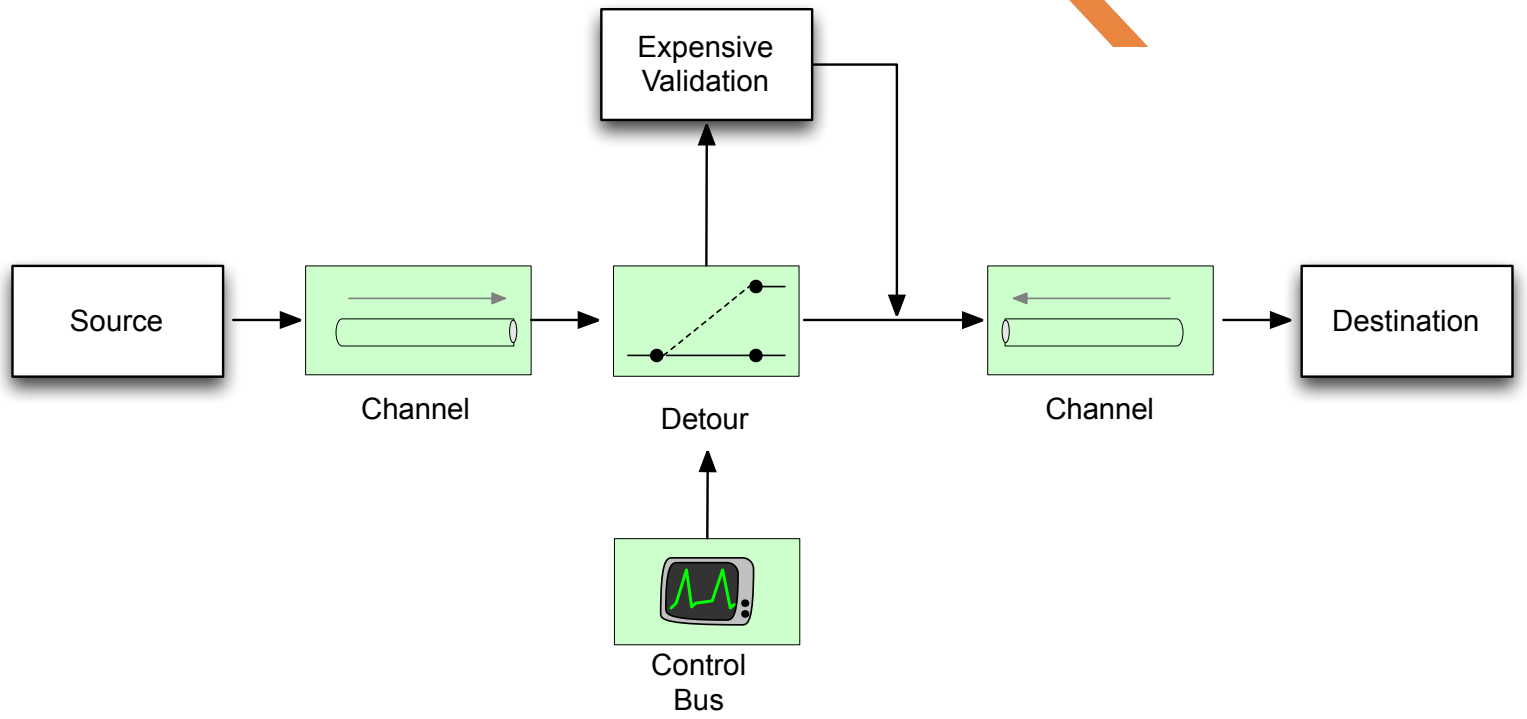
- A wire tap is a simple Recipient List with two recipients.
- One channel is the tap that can be switched on or off (via Control Bus) when messages need to be inspected in a point to point channel.

WireTap



In Nova – logging messages to disk

Detour



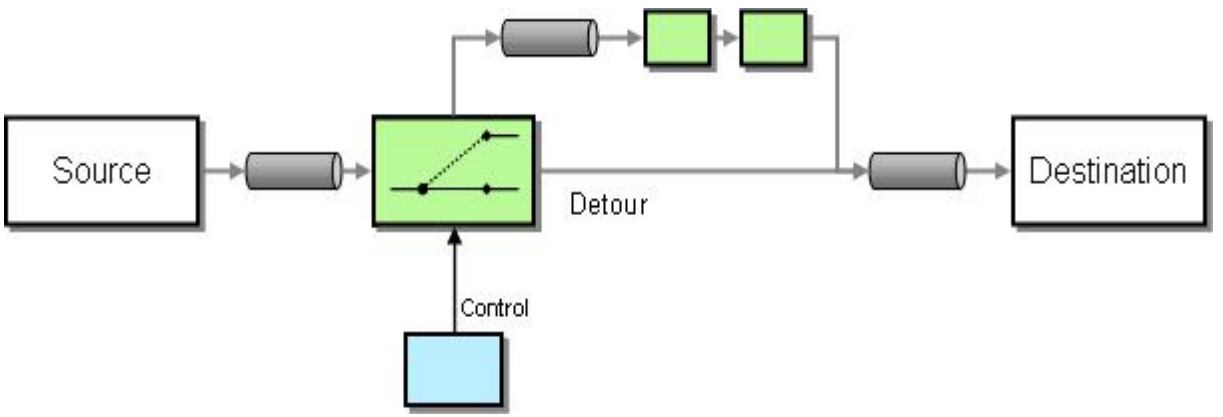
- A Detour routes a message via a different route for validation, testing and debugging purposes.
- A Context Based Router can be used to implement the pattern.

e.g. if the control bus signals to the router that the validation route should be taken.

Detour

Purposes:

- Debugging
- Validation
- Testing
- Inspection



Enterprise Message Brokers

IBM MQ

Tibco EMS

Frameworks that implement EIP

Questions?

Thank you..!!!

LinkedIn, GitHub, GitLab, Twitter: @kodtodya

https://kodtodya.github.io/talks/