

Practical Messaging

A 101 guide to messaging

Ian Cooper

Twitter: ICooper

Who are you?

- Software Developer for more than 25 years
 - Stuff I care about: Messaging, EDA, Microservices, TDD, XP, OO, RDD & DDD, Code that Fits in My Head, C#
 - Places I have worked: DTI, Reuters, Sungard, Beazley, Huddle, Just Eat Takeaway
- No smart folks
 - Just the folks in this room



Welcome to Brighter

This project is a Command Processor & Dispatcher implementation with support for task queues that can be used as a lightweight library.

It can be used for implementing [Ports and Adapters](#) and [CQRS \(PDF\)](#) architectural styles in .NET.

It can also be used in microservices architectures for decoupled communication between the services

[GET STARTED](#)

Day One Agenda

- Distribution
- Integration Styles
- Request Driven Architectures
- Event Driven Architectures
- Messaging Patterns

When do you get to write code?

When we get to messaging patterns, hopefully by the afternoon of Day One and then into the morning of Day Two

Prerequisites

We will use Rabbit MQ for examples. Either you need to have RMQ installed on your machine, or you should have Docker installed on your machine, as exercises provide a Docker Compose file to spin up RMQ.

You will need to be able to author C#, Python, or JavaScript with an editor/IDE of your choice.

You will need Python 3, .NET Core, or ES6

Exercise Code

<https://github.com/iancooper/Practical-Messaging-Sharp>

<https://github.com/iancooper/Practical-Messaging-Python>

<https://github.com/iancooper/Practical-Messaging-JavaScript>

Day One

What is driving messaging

1.DISTRIBUTED SYSTEMS

Why Distribute?

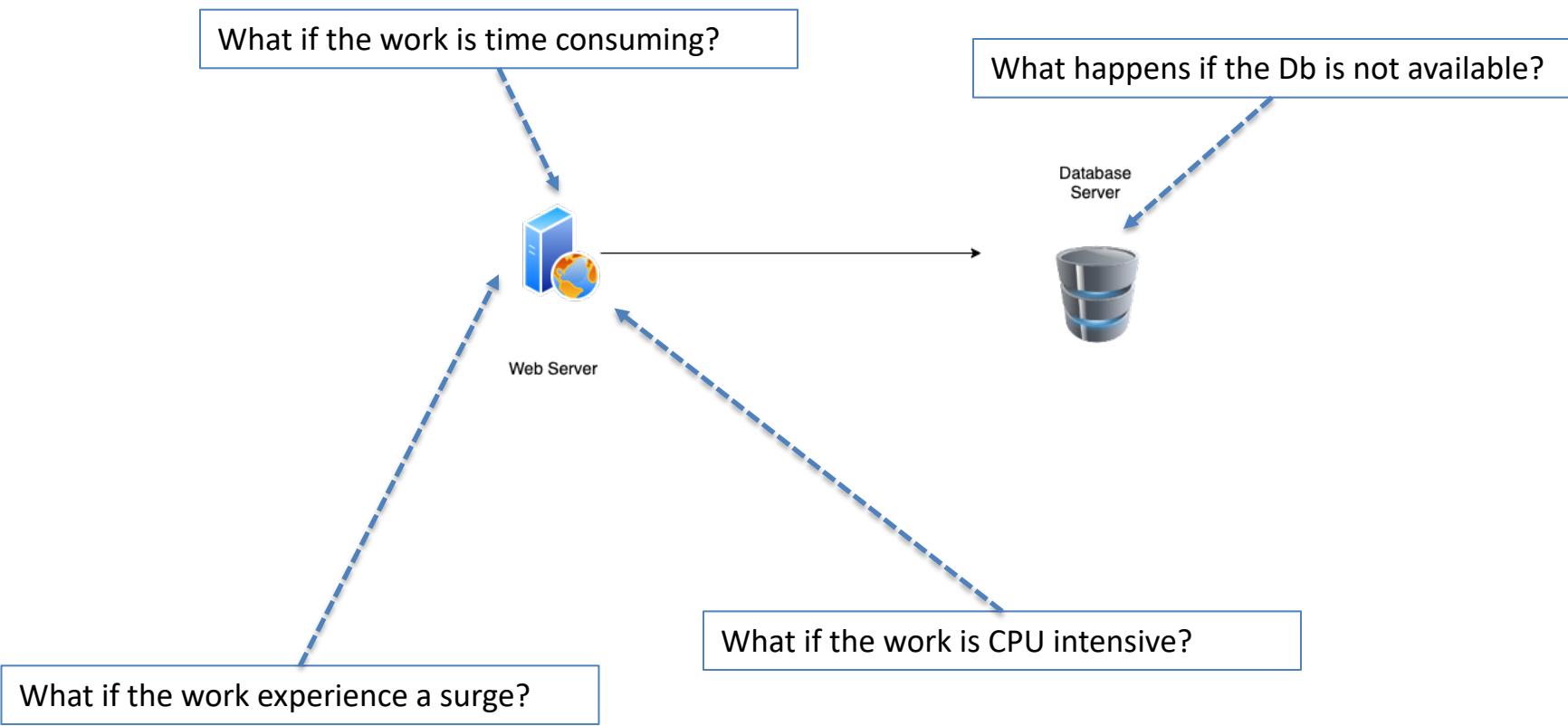
Performance and Scalability

Availability

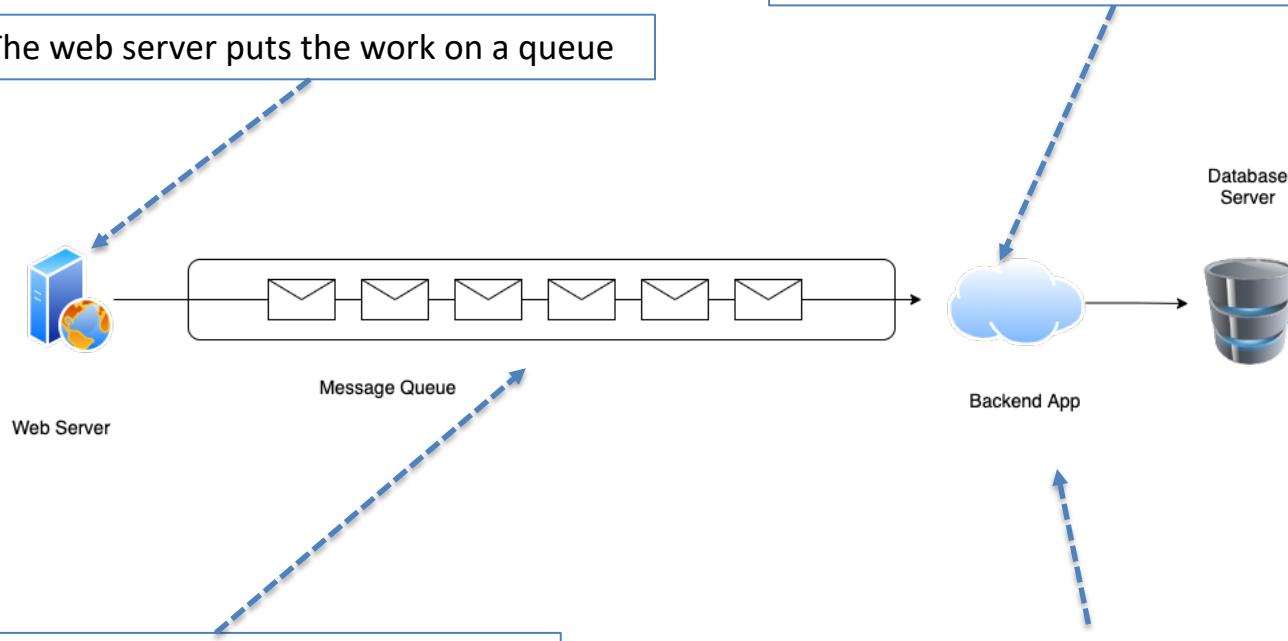
Maintainability

Inherent Distribution

Example: Task Queues



The web server puts the work on a queue



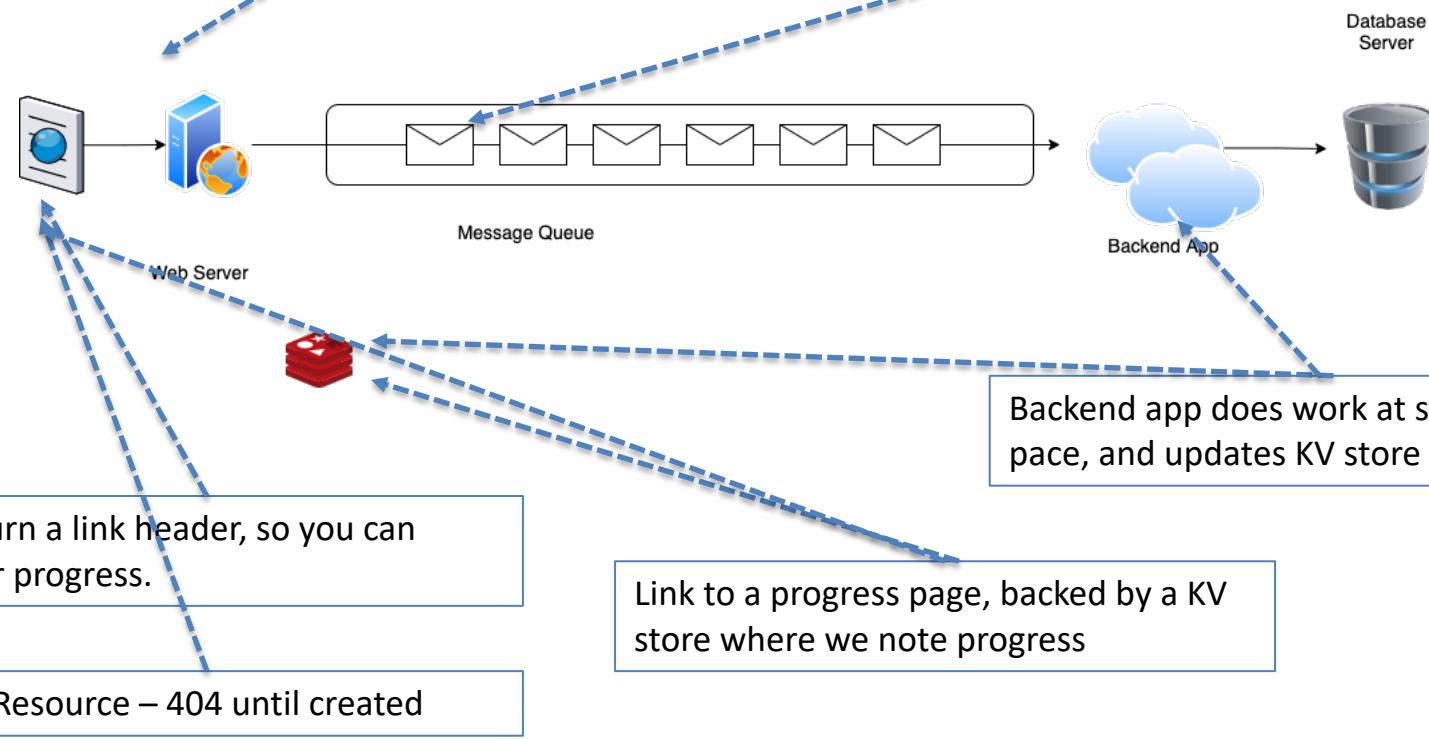
The queue stores work until we are ready to consume it. We can *throttle* to prevent surges.

A backend application can perform long-running or CPU intensive work, allowing the web server to service new requests.

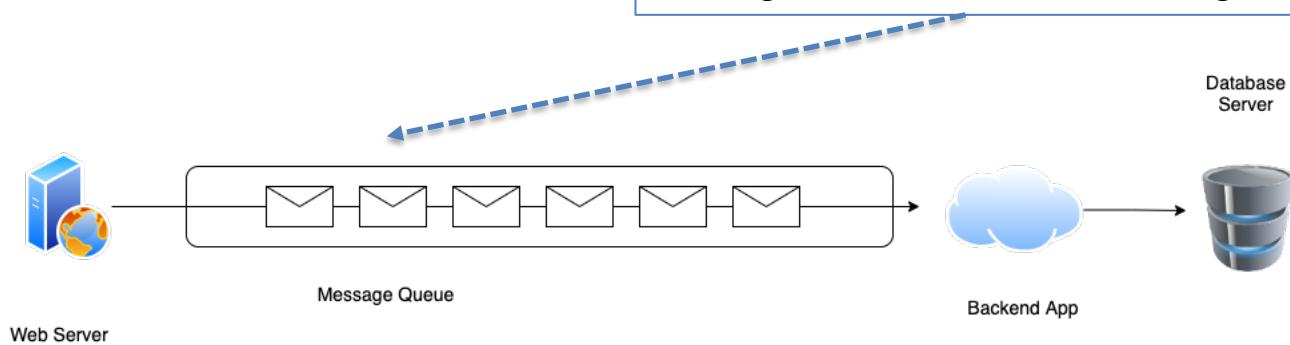
We can scale out the backend services using a competing consumers approach, to ensure the queue does not backup.

We return 202 Accepted – we have your work request, and won't lose it.

We enqueue a work item for the request.



This general technique is known as Decoupled Invocation – we separate building the command from executing it.



Decoupled Invocation Pattern

Use Decoupled Invocation. A producer puts a message onto a queue at the service endpoint. A consumer reads messages from the queue.

The queue stores messages for eventual processing. If the queue is durable we gain guaranteed delivery, and at-least once guarantees.

If the rate of arrival at the endpoint is unpredictable, the queue acts as a buffer that makes it possible to predict the rate of consumption.

This makes it simpler to do capacity planning because peaks of requests are smoothed out by the queue.

The consumer must be able to control the rate of processing, otherwise a spike is simply passed down the wire.

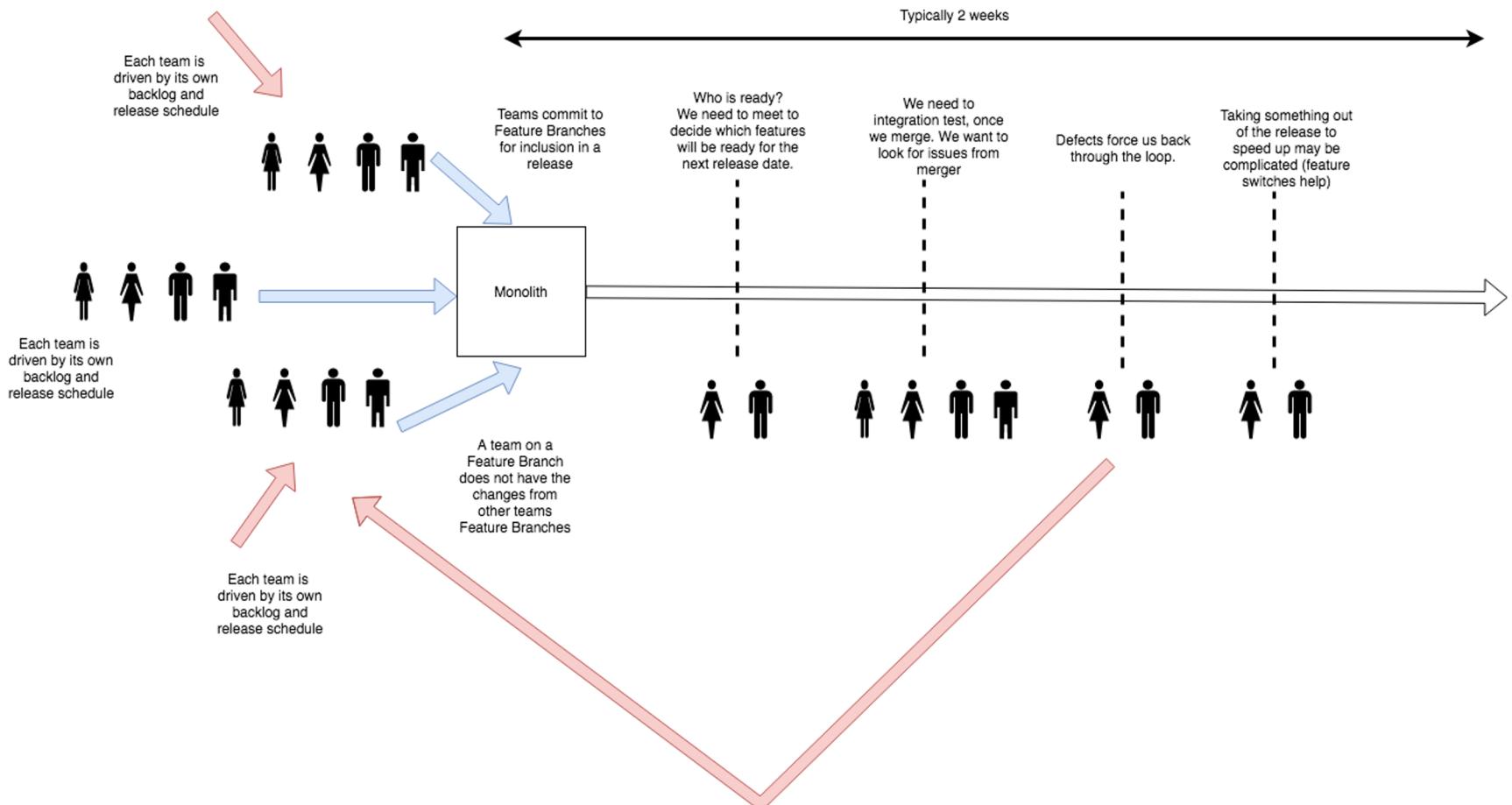
Example: Microservices

It's all about velocity!!!

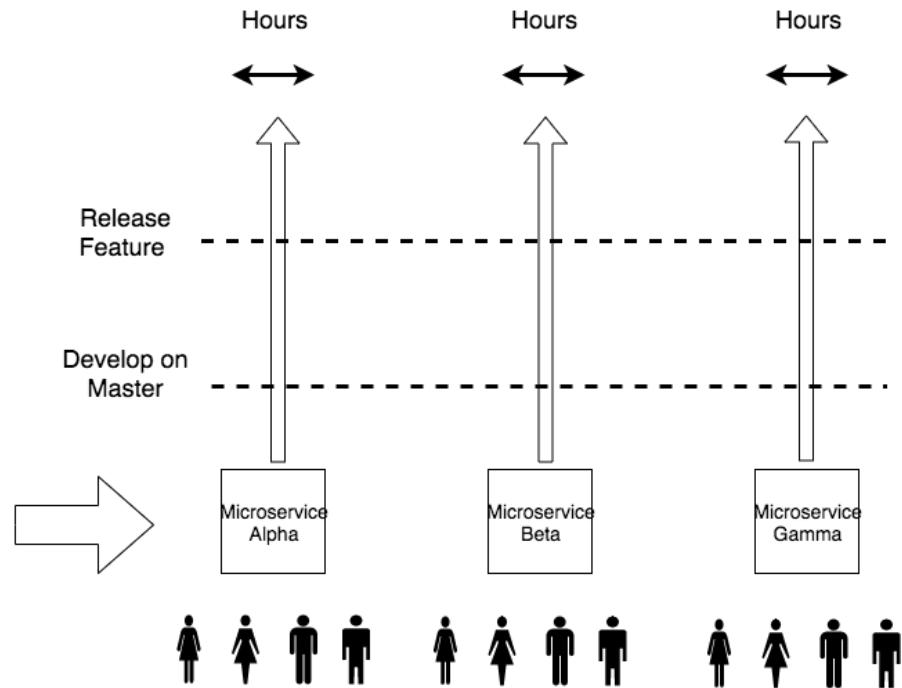
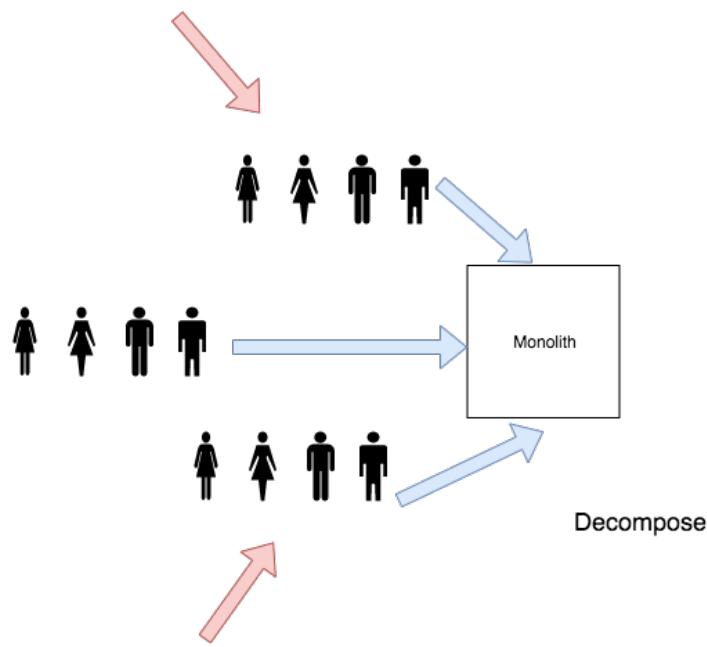
“Speed wins in the marketplace”

Adrian Cockcroft, former lead architect at Netflix

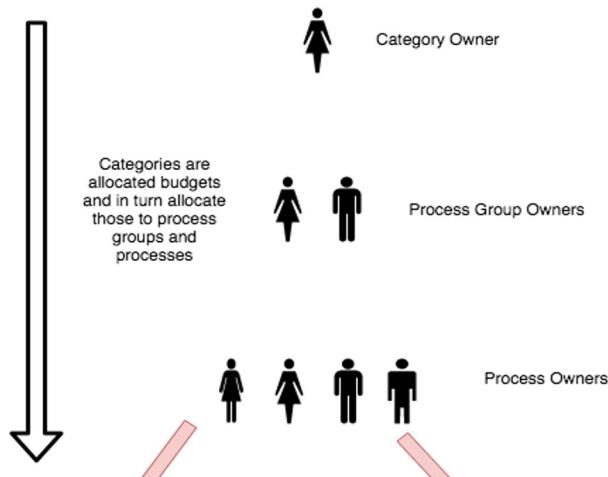
Monoliths Do Not Scale To Many Teams!



Microservices let us scale an organisation

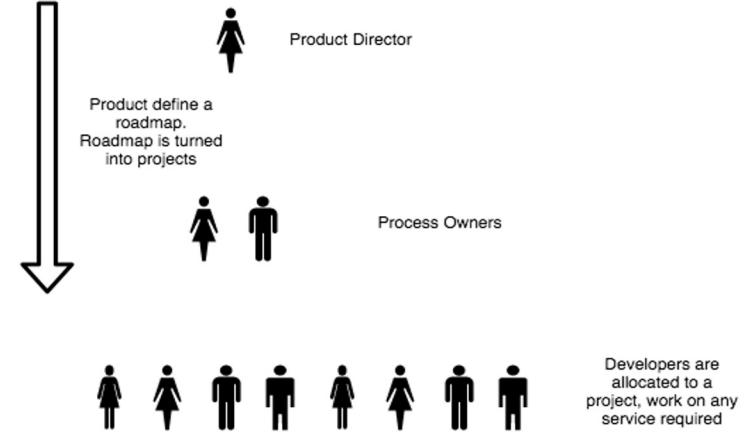


Product Mode



Teams work on one microservice.
They work iteratively, on the next most important thing.
Product defines the next most important thing from metrics on the product

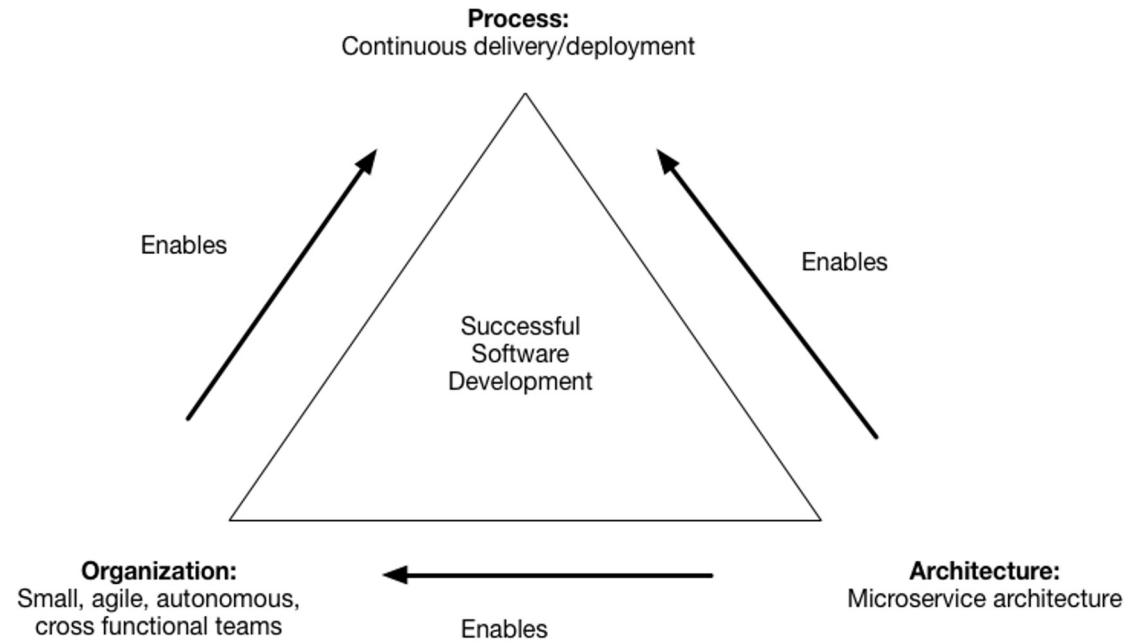
Product Mode



Developers are allocated to a project, work on any service required

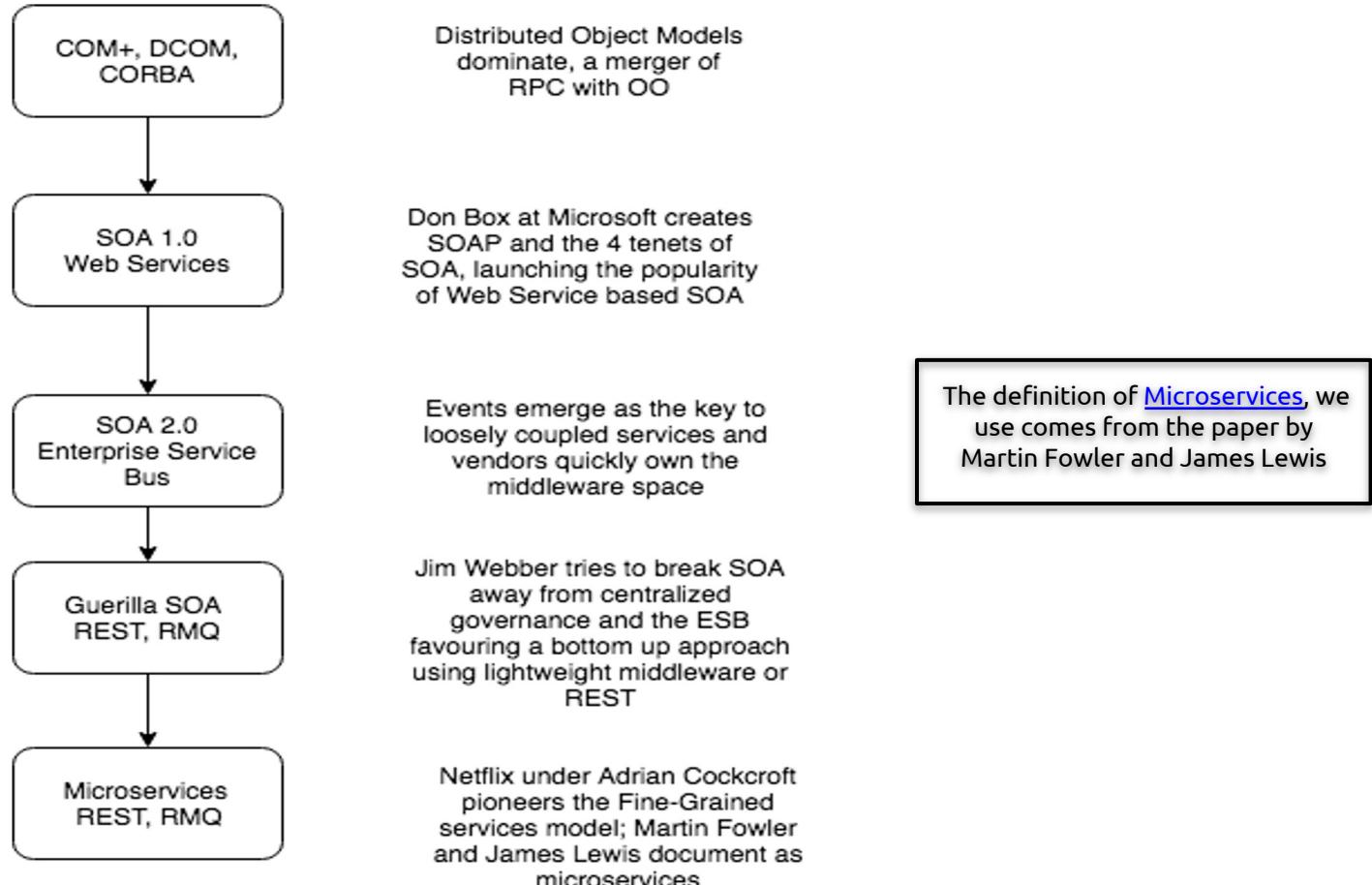
Projects

Microservices enable agility

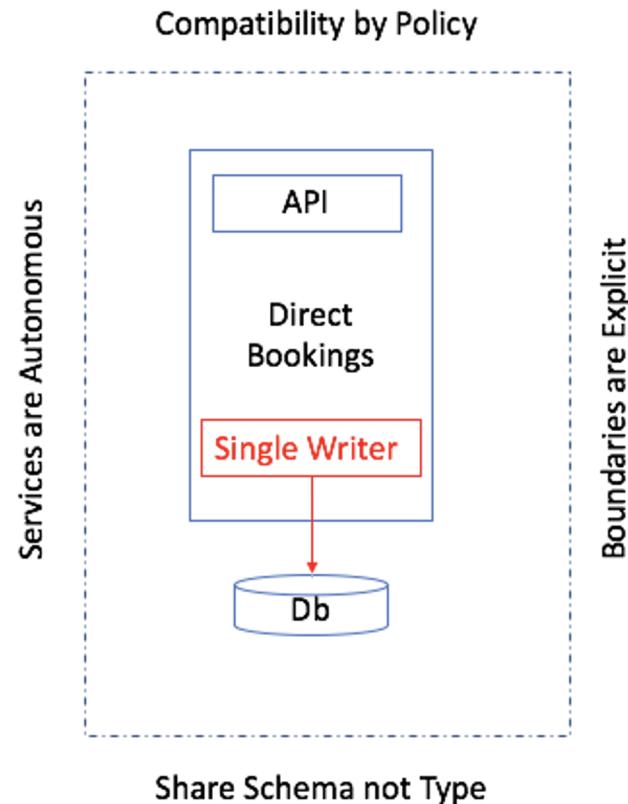


<https://microservices.io/patterns/decomposition/decompose-by-business-capability.html>

A Brief History of Microservices



Microservices are partitions of software



The Price of Distribution

Fallacies of Distributed Computing

The network is reliable.

Latency is zero.

Bandwidth is infinite.

The network is secure.

Topology doesn't change.

There is one administrator.

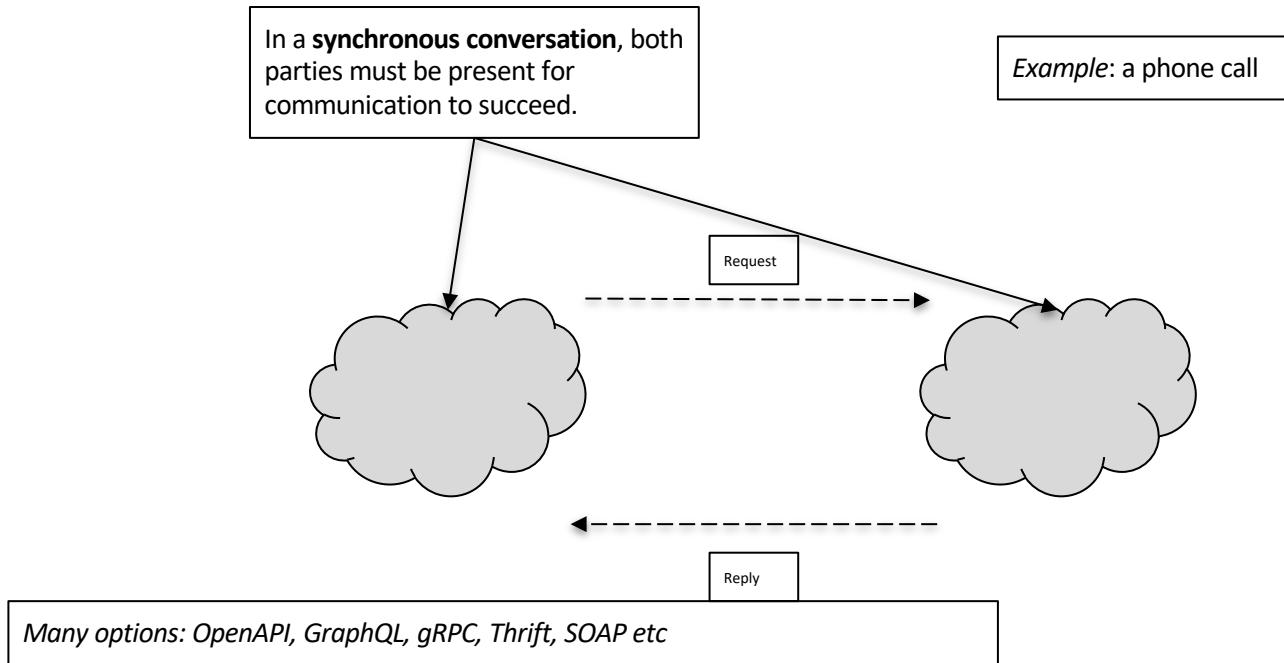
Transport cost is zero.

The network is homogeneous.

How do we communicate between microservices?

2. INTEGRATION STYLES

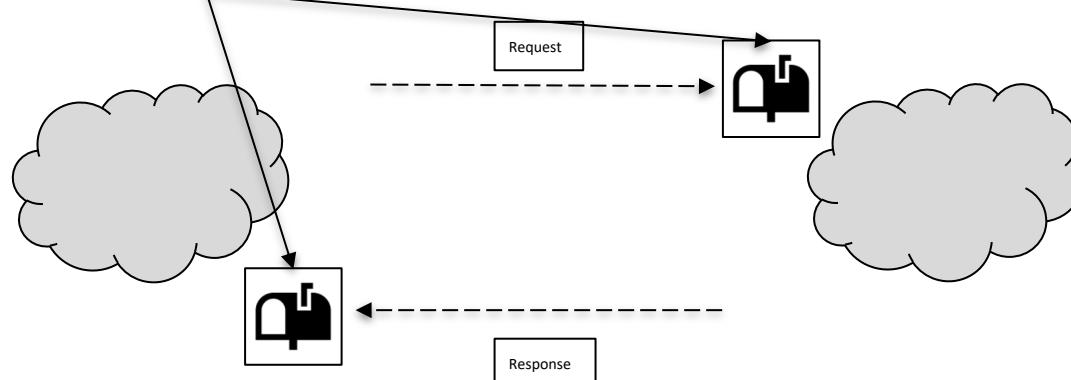
Synchronous Conversation



Asynchronous Conversation

In an **asynchronous conversation**, the *receiver* does not need to be present at the time the *sender* communicates with them, using **store and forward** to pick up the message later.

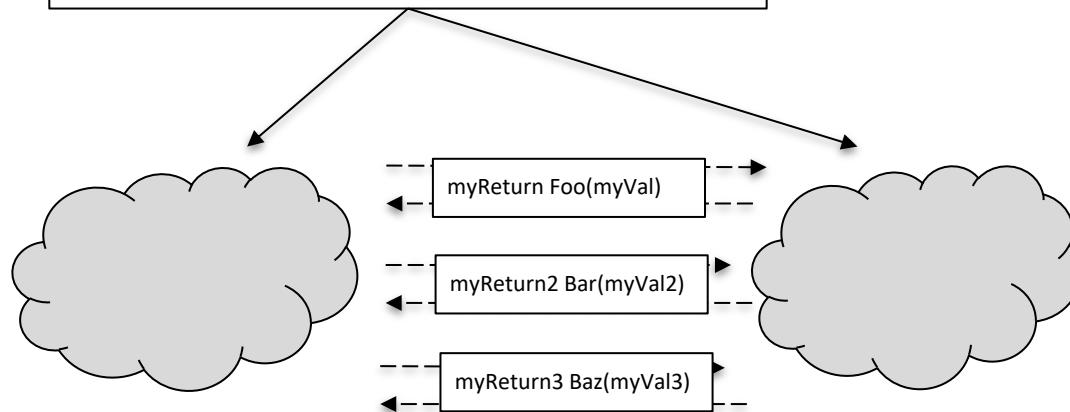
Example: snail mail



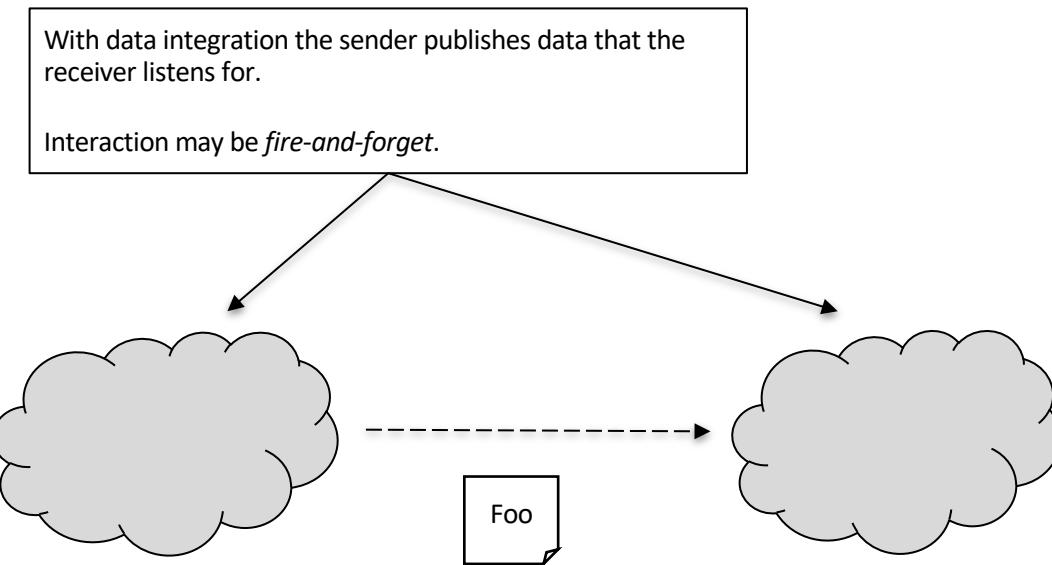
Many options: SQS, Kafka, AMQP 0-9-1 (RMQ), AMQP 1-0, MQTT, S3

Behavioral Integration

With behavioral integration the receiver exposes an API of parameterized method calls to the sender.
Interaction is usually a request and a response pair

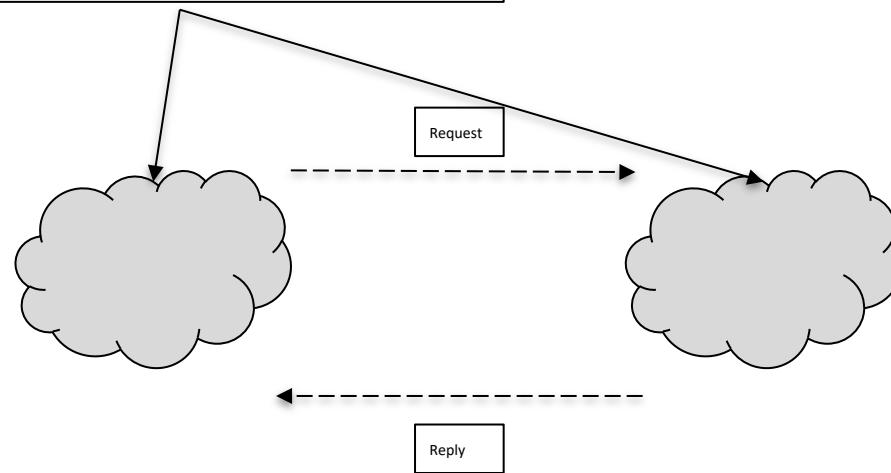


Data Integration



Temporal Coupling

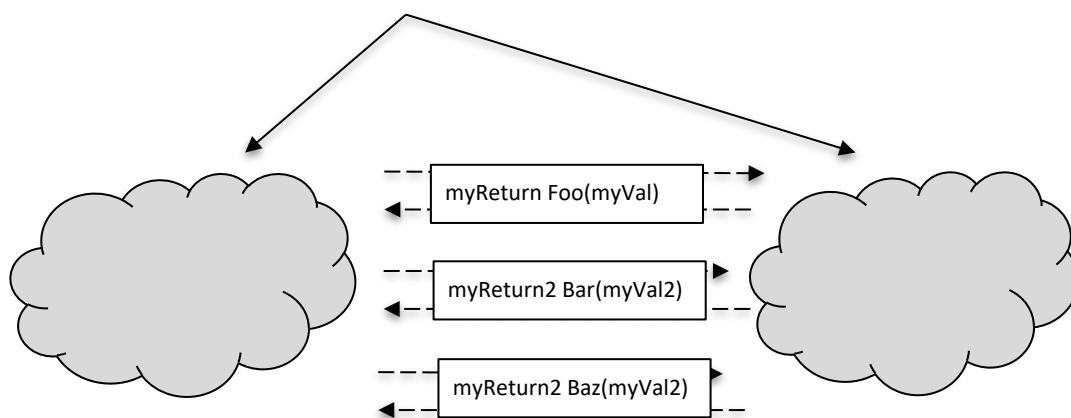
If both parties must be present to succeed we say they are *temporally coupled*. The availability of one has an impact on the availability of another.



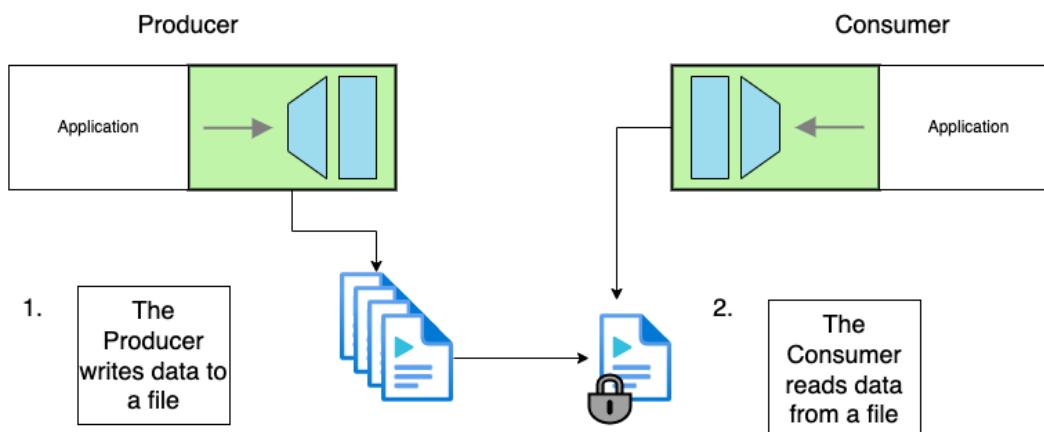
Behavioral Coupling

Where two systems use behavioral integration, they are coupled by the set of calls in the interaction. We find it hard to change who we talk to; we find it hard to change how we talk about something.

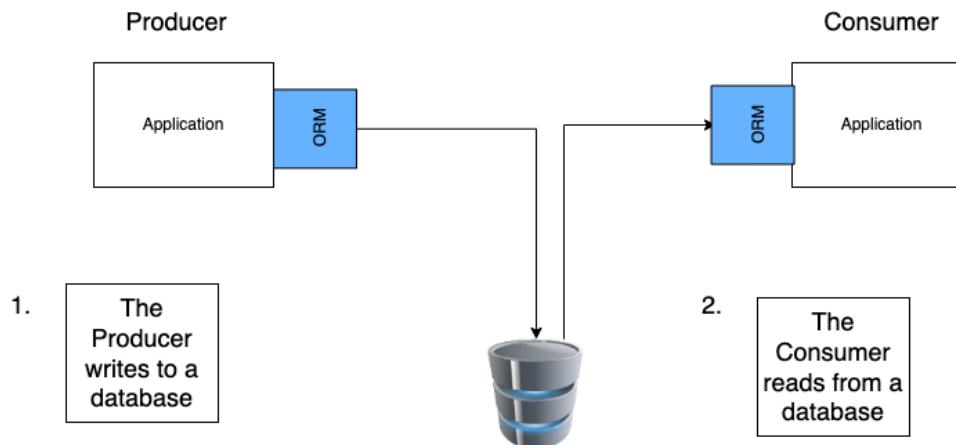
We say such systems are **behaviorally coupled**



File Transfer



Shared Database



Remote Procedure Call



1. The Client calls a remote procedure on the Server

2. The Server listens for calls, actions them, and returns results

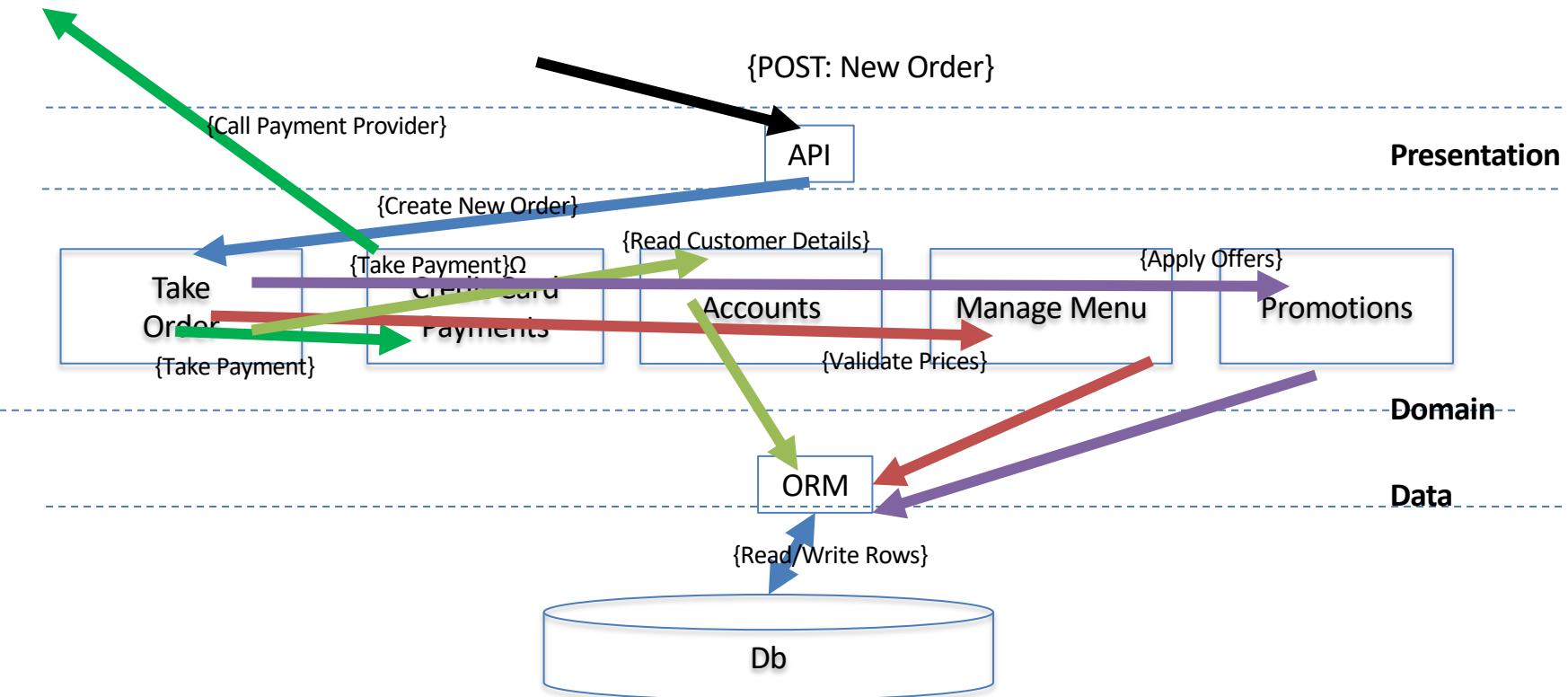
Messaging



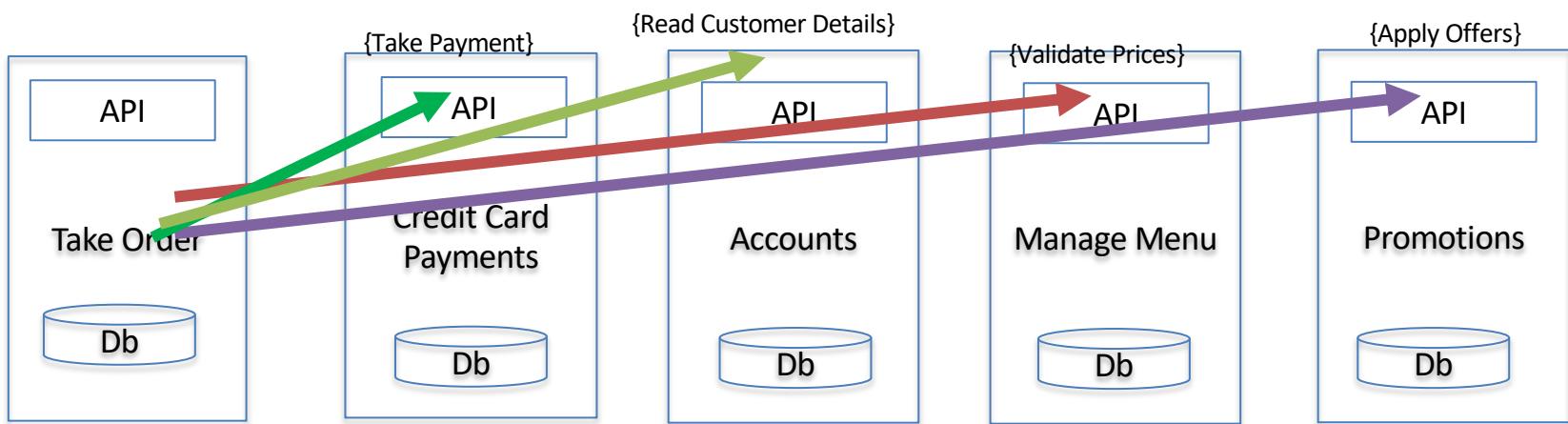
Integrating using RPC or REST

3.REQUEST DRIVEN ARCHITECTURES

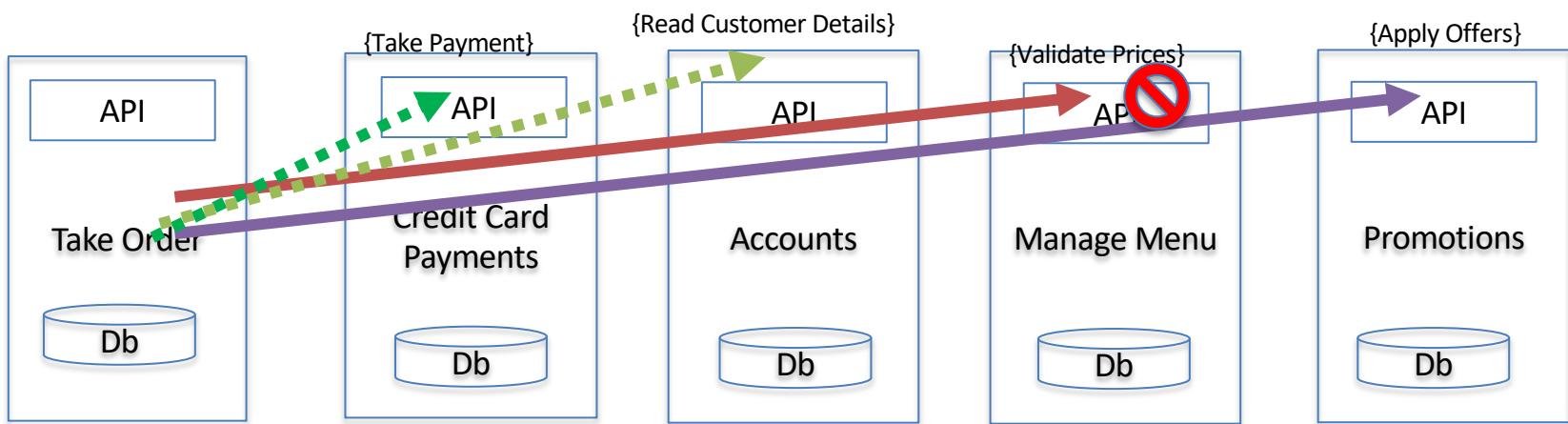
Getting work done in a Monolith



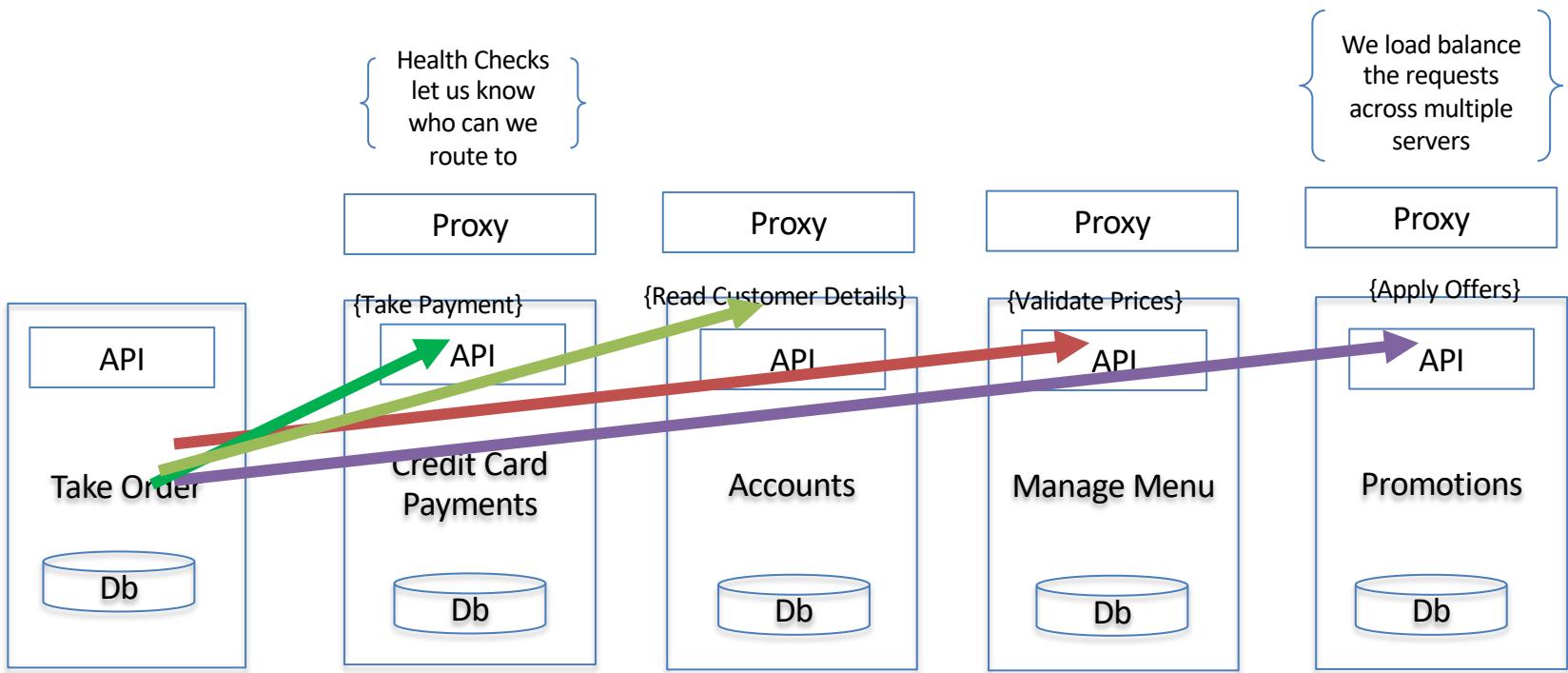
Microservices



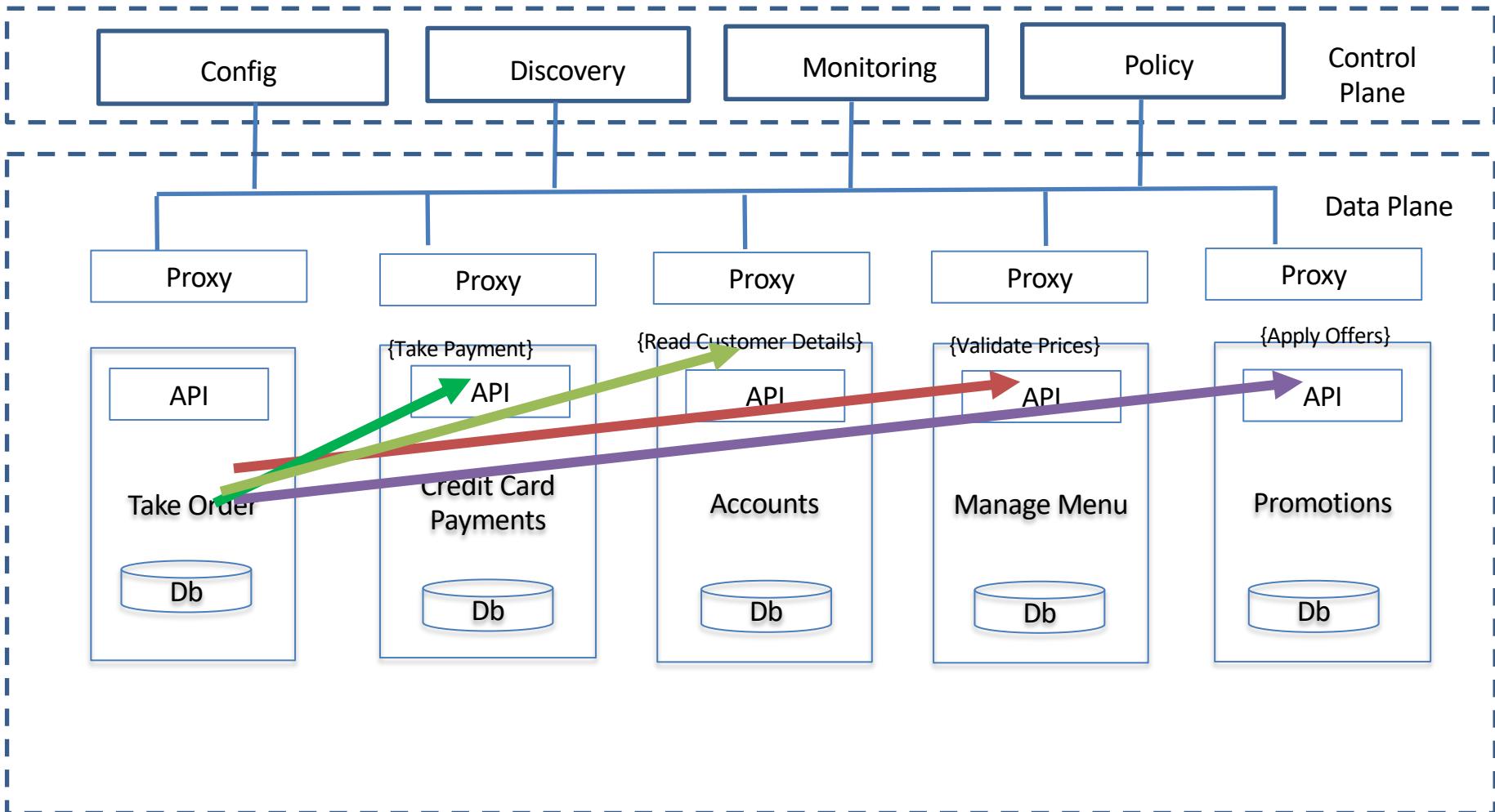
Temporal Coupling



Using A Proxy for Availability

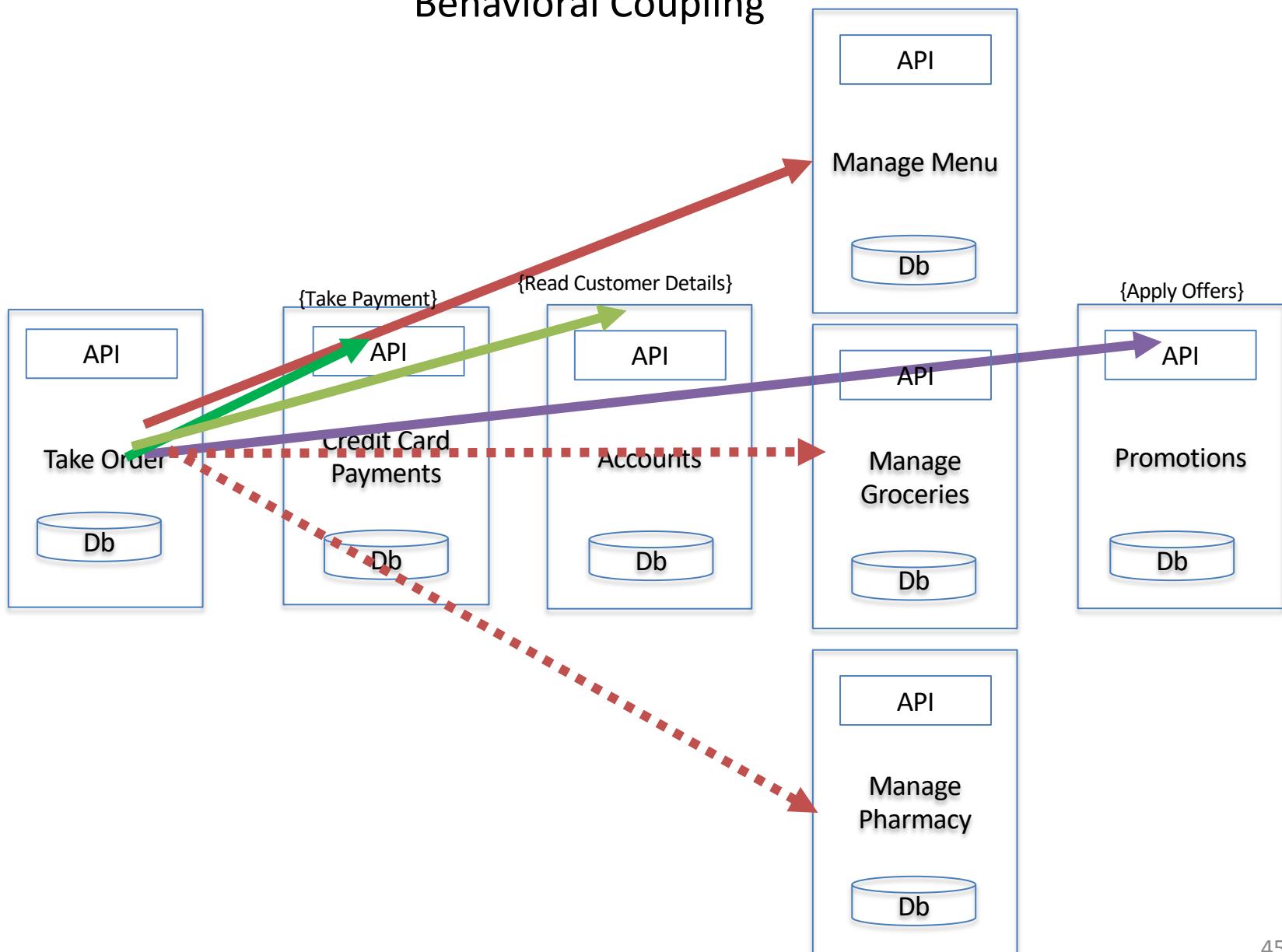


Service Mesh

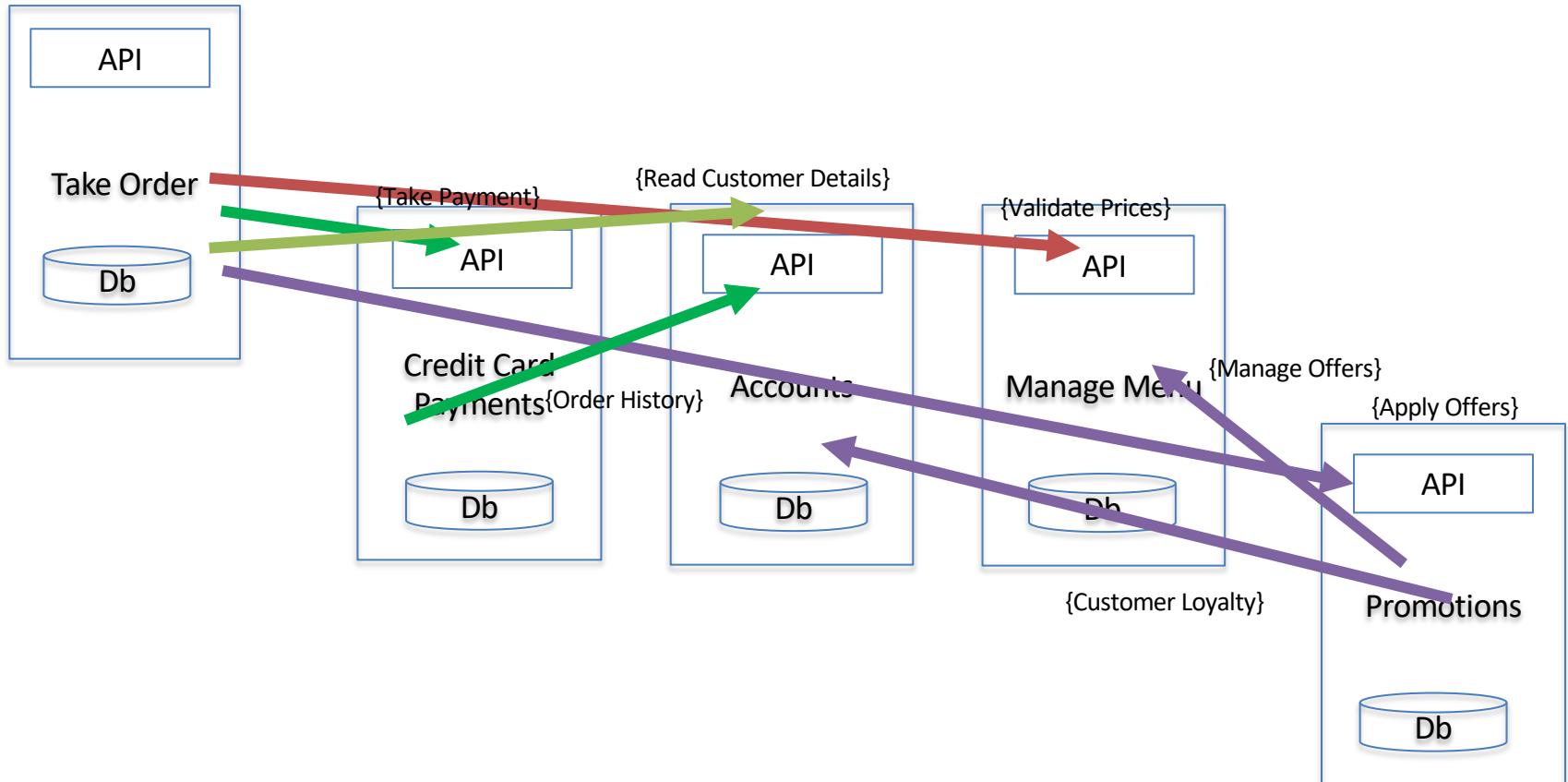


{Validate Prices}

Behavioral Coupling



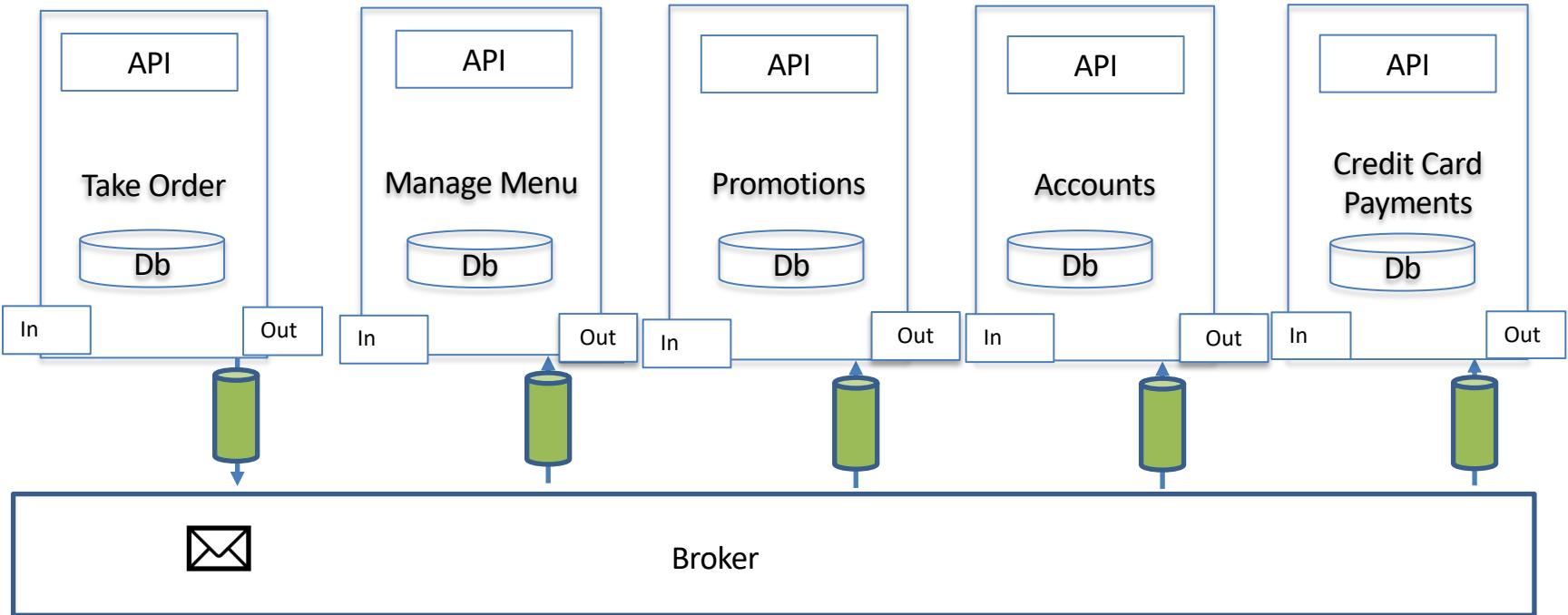
$N(N-1)/2$ connections



Integrating using events

4.EVENT DRIVEN ARCHITECTURES

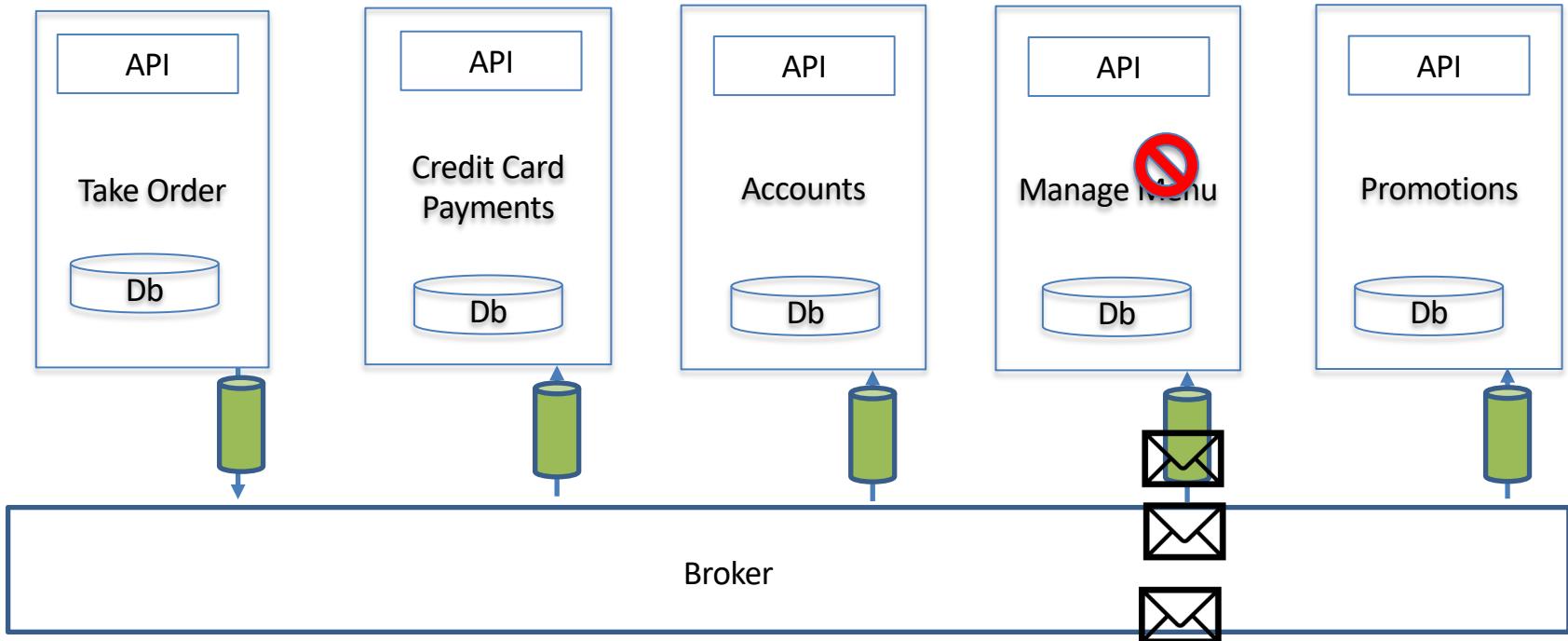
Event Driven Architecture



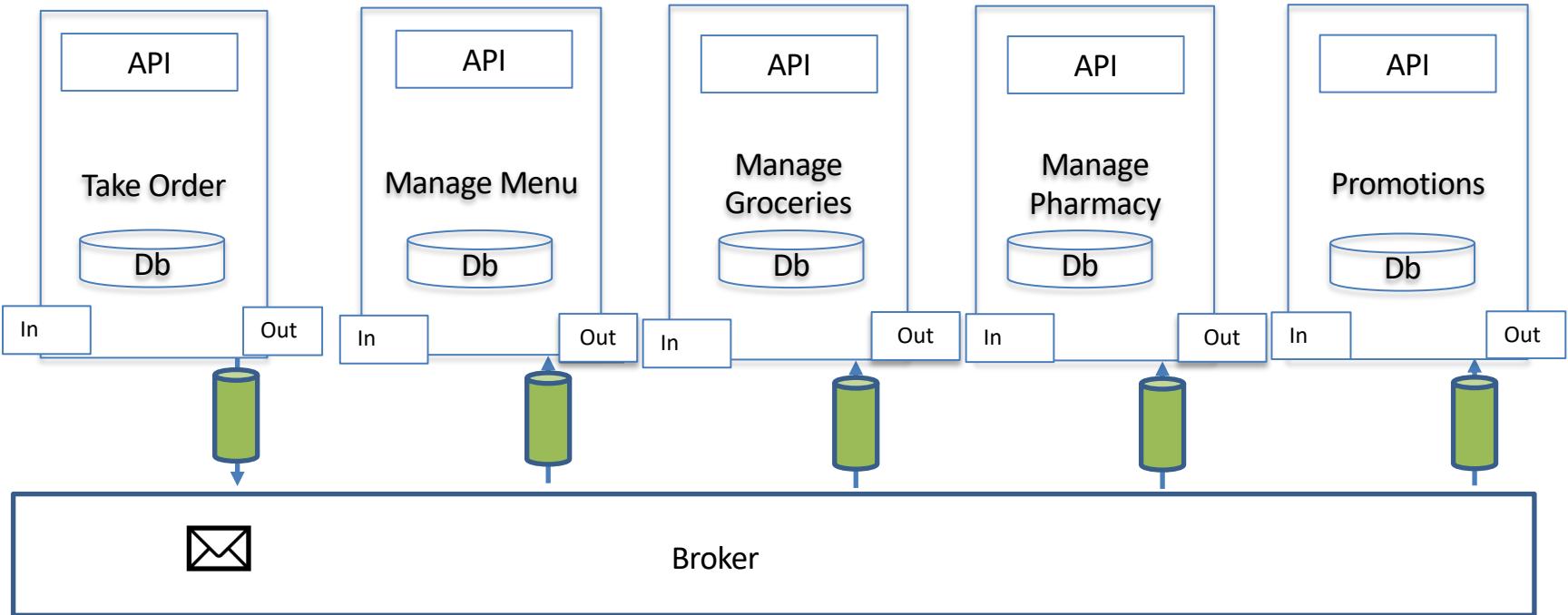
Consumer has no notion of producer, just a topic on the broker

Channels subscribes to a 'topic' on the broker

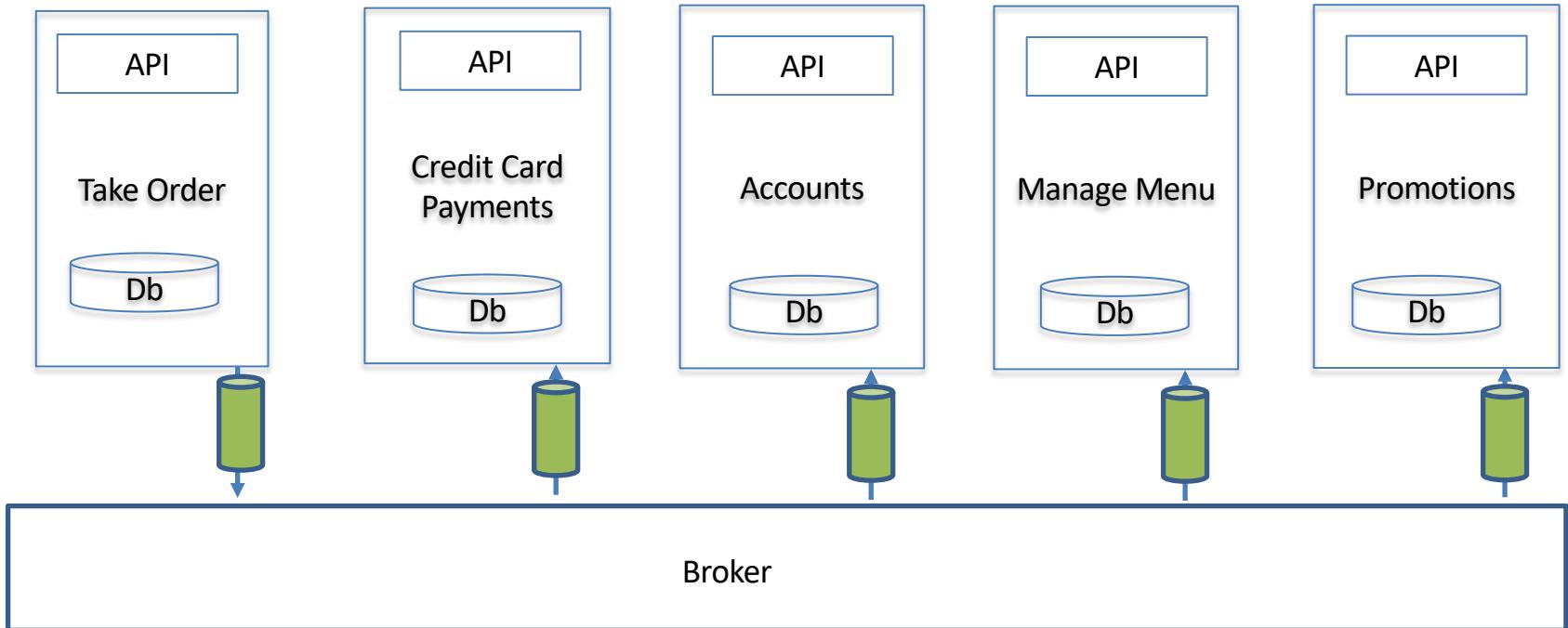
Guaranteed, At Least Once



Publish Subscribe

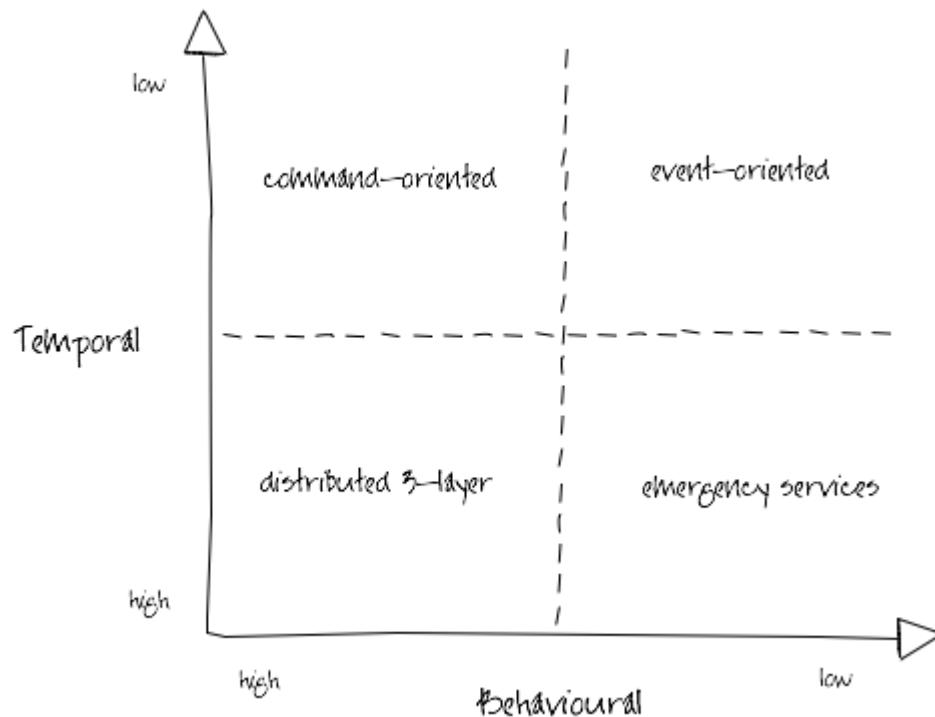


Hub and Spoke



Integrating using events

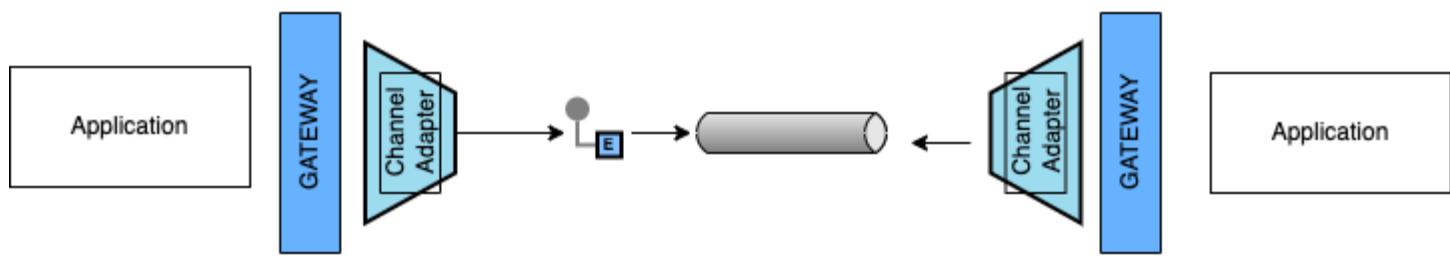
CONTRASTING REQUESTS AND EVENTS



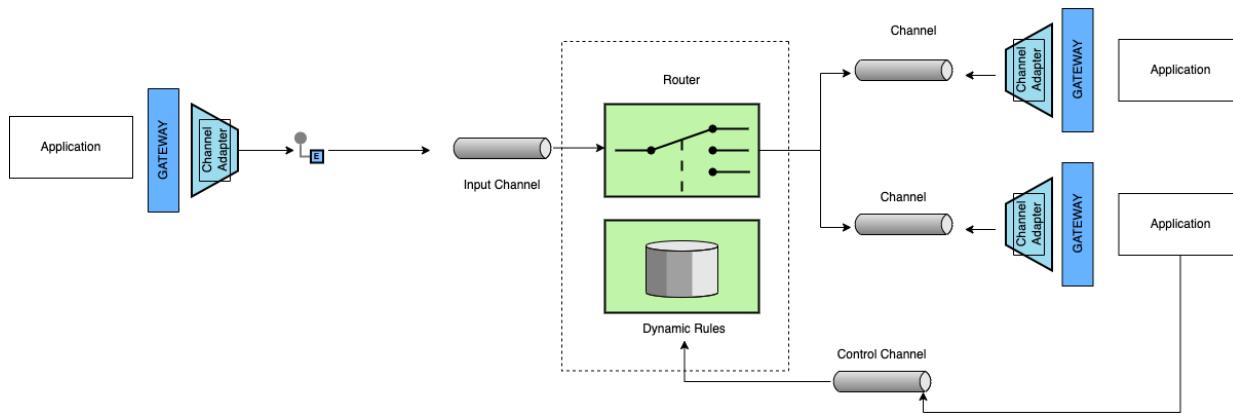
Ian Robinson: <http://iansrobinson.com/2009/04/27/temporal-and-behavioural-coupling/>

Integrating using events

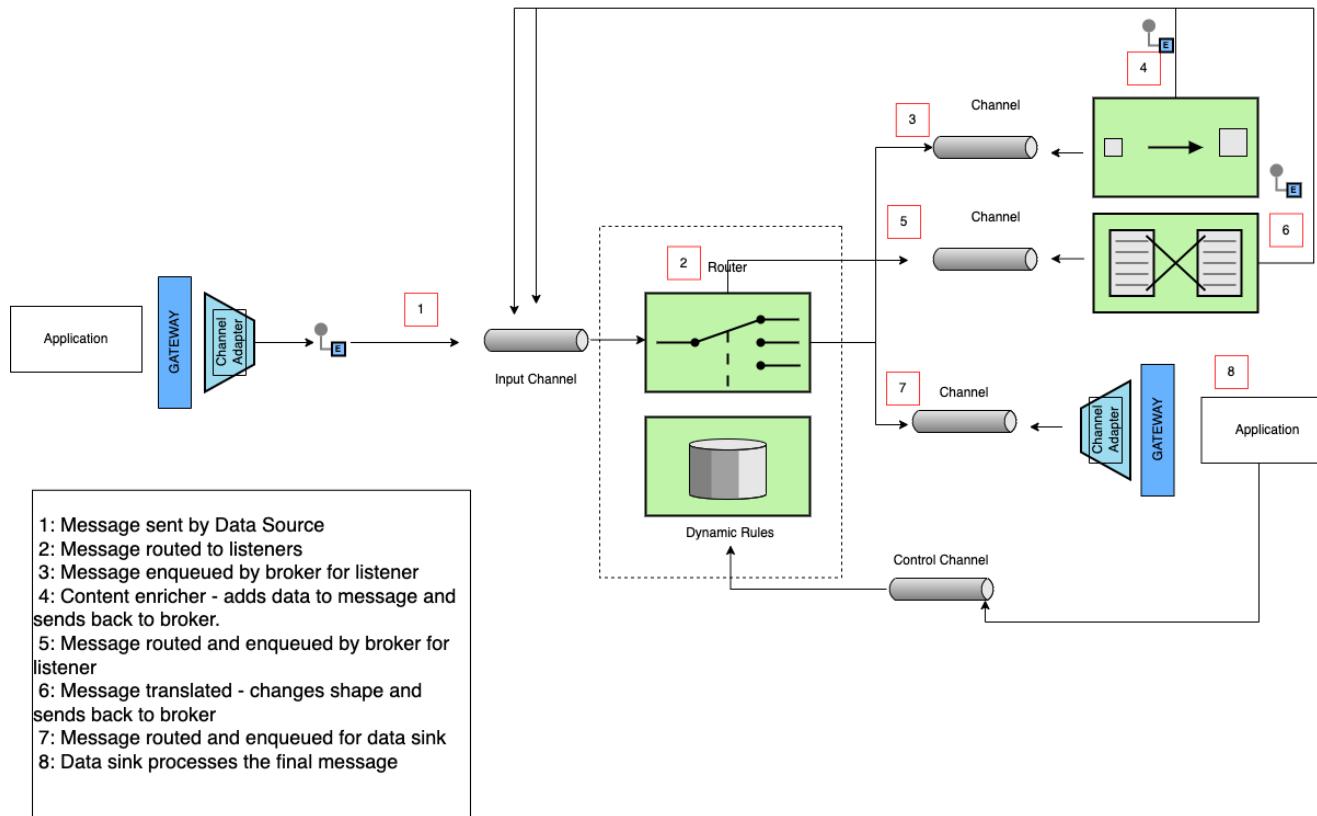
5.MESSAGING PATTERNS



Messaging with A Broker



Pipes and Filters



5.1 TYPES OF MESSAGES

Command Message

Use a Command Message to reliably invoke a procedure in another application

Uses the well-established pattern for encapsulating a request as an object. The Command pattern [GoF] turns a request into an object that can be stored and passed around.

Document Message

Use a Document Message to reliably transfer a data structure between applications.

The receiver decides what, if anything, to do with the data

Event Message

Use an Event Message for reliable, asynchronous event notification between applications.

The difference between an Event Message and a Document Message is a matter of timing and content. An event's contents are typically less important.

Request-Reply

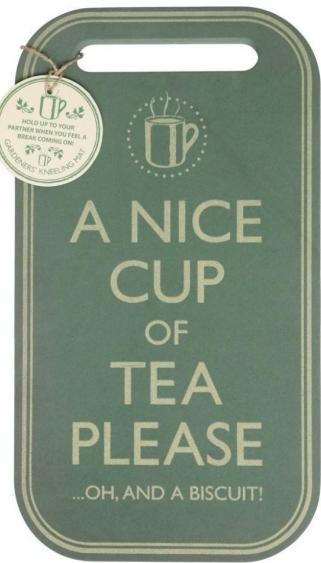
When an application sends a message, how can it get a response from the receiver?

Send a pair of Request-Reply messages, each on its own channel.

The request message should contain a Return Address that indicates where to send the reply message

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

Command or Event?



A Command



An Event

[View original](#)[Flag media](#)**Adam Dymitruk** @adymitruk

I'm lying in the middle of the exhibition hall. Caught in a catch-22. Too tired/sleepy to get a strong tea. Somebody help? #buildstuffua

38m





Adam Dymitruk

@adymitruk

Oh. I drink me tea black, no
sugar, no milk. Assam would be
nice ;) #buildstuffua

12:26pm · 21 Nov 2016 · Twitter for Android



...

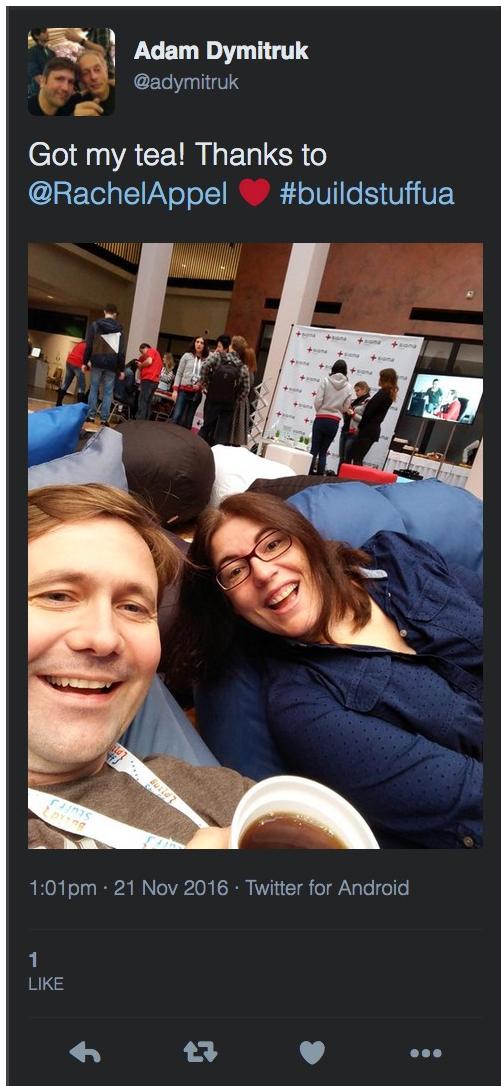


Dylan Beattie ● @dylanbeattie 2h
Somebody @Buildstuffua please
bring @adymitruk a black tea...
he's in the middle of the
exhibition hall and he won't stop
complaining... :)

Adam Dymitruk @adymitruk

Oh. I drink me tea black, no
sugar, no milk. Assam would be
nice ;) #buildstuffua





What is a message?

5.2 A MESSAGE

Message Construction

A message has a header and body

The body contains data for the consumer

The header contains metadata for any *filter* in the pipeline.

The header should indicate the format of the body

Break a large message into pieces as a
Message Sequence

5.3 CHANNELS

Channels

A virtual pipe that connects producer and consumer

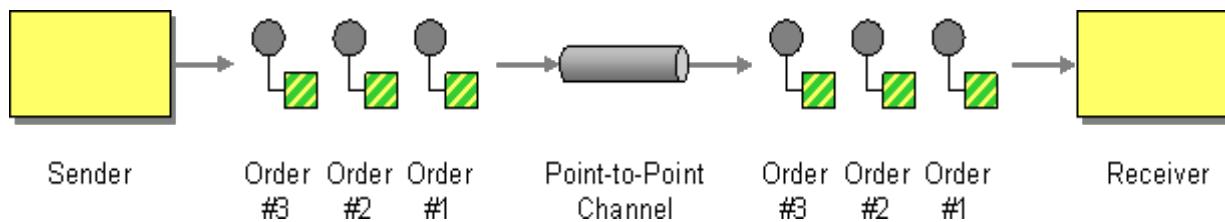
Logical Address (Topic or Routing Key)

Unidirectional

One-to-One or One-to-Many

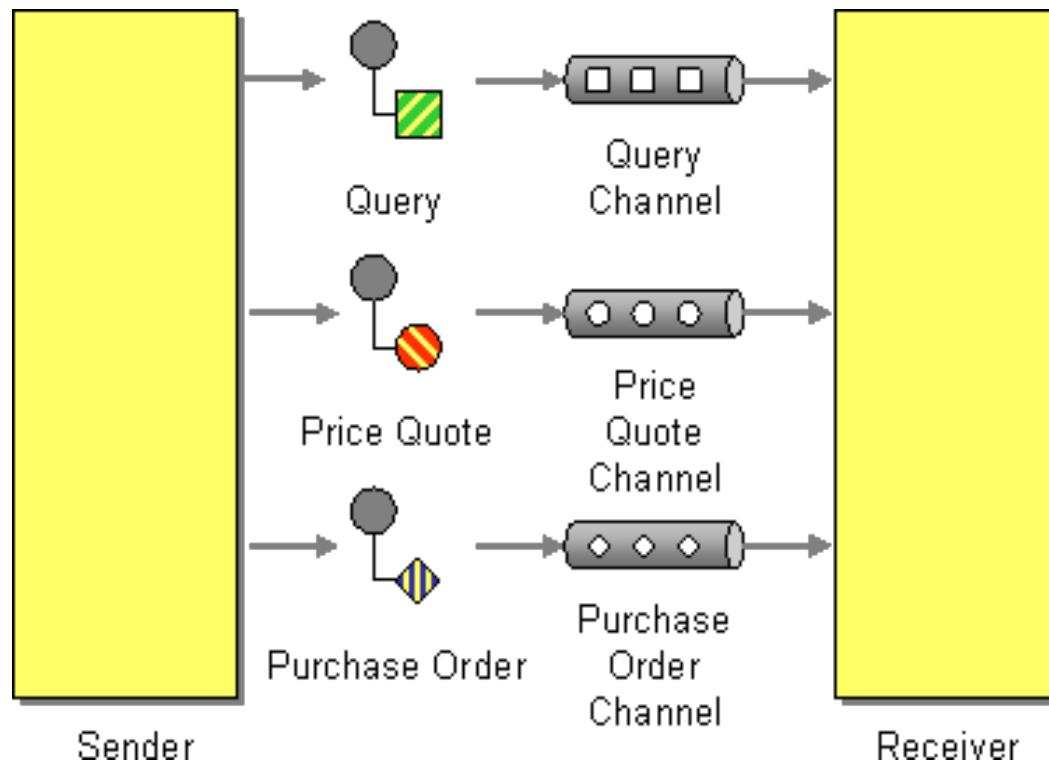
Messaging is a ‘pipe’ not a ‘bucket’.

Point-to-Point Channel



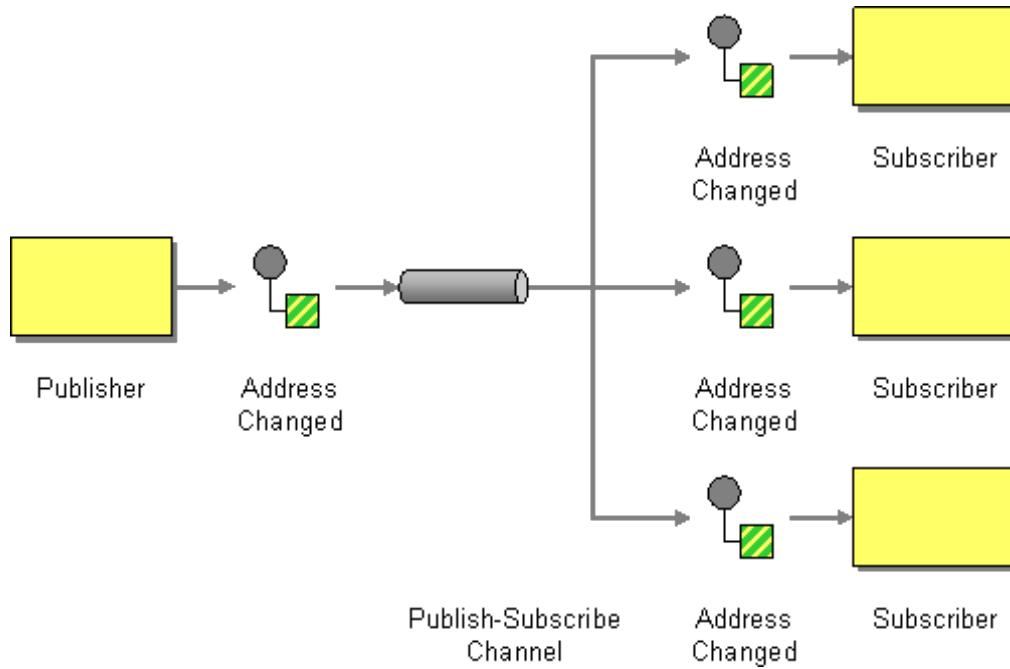
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/PointToPointChannel.html>

Datatype Channel



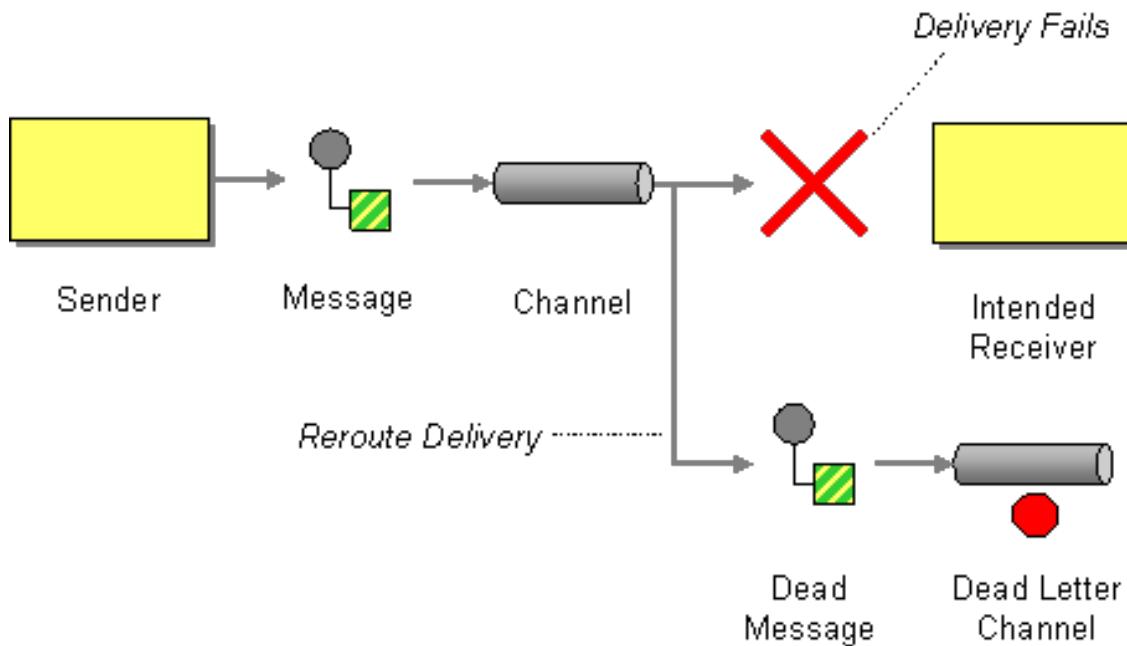
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/DatatypeChannel.html>

Publish-Subscribe Channel



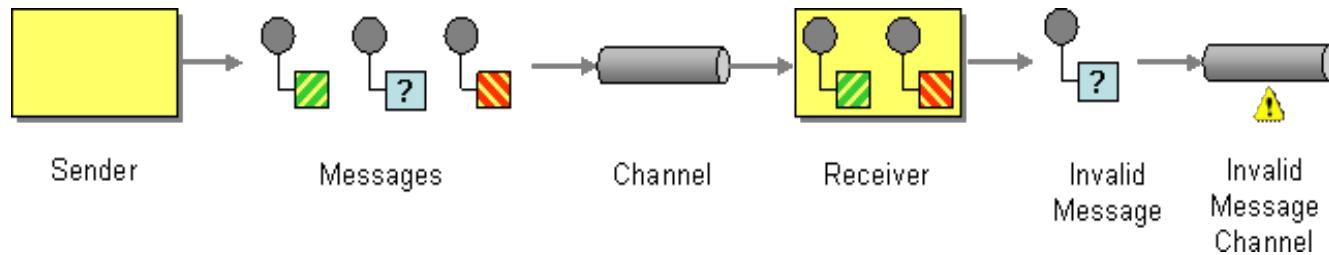
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/PublishSubscribeChannel.html>

Dead Letter Channel



<http://www.enterpriseintegrationpatterns.com/patterns/messaging/DeadLetterChannel.html>

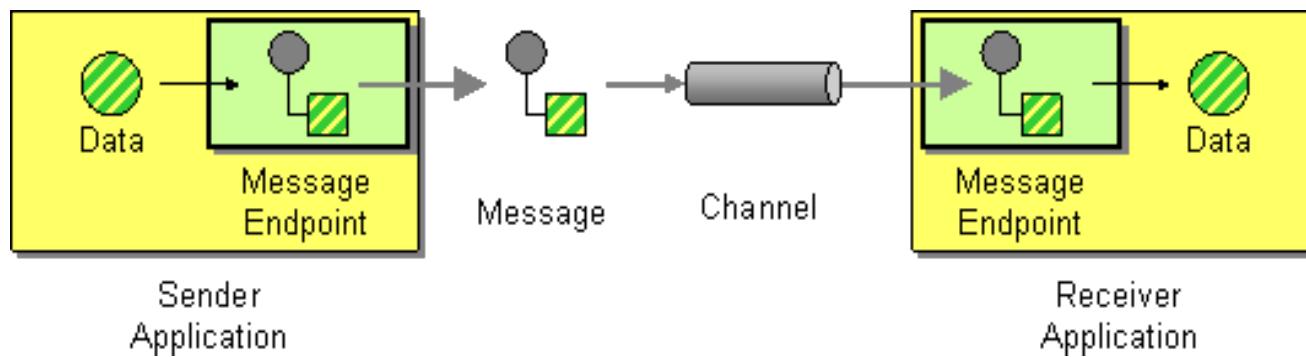
Invalid Message Channel



<http://www.enterpriseintegrationpatterns.com/patterns/messaging/InvalidMessageChannel.html>

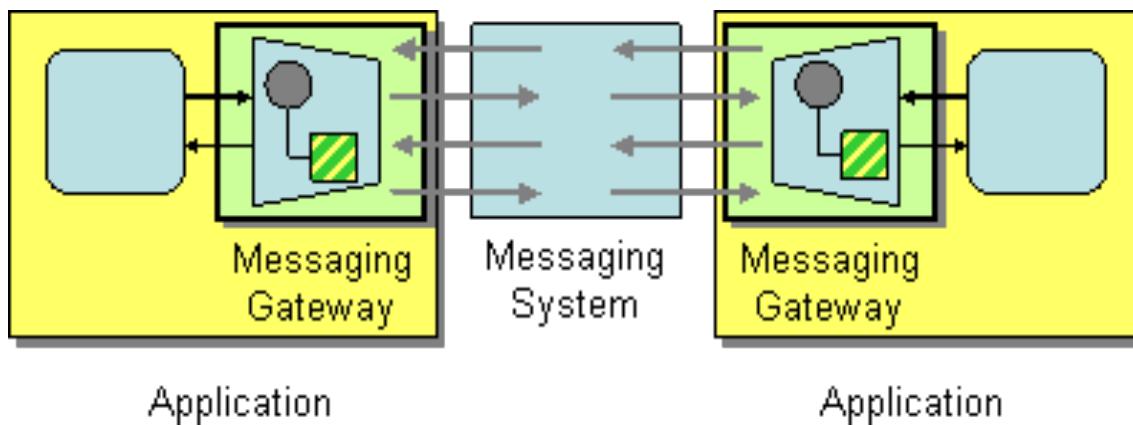
5.4 ENDPOINTS

Message Endpoint



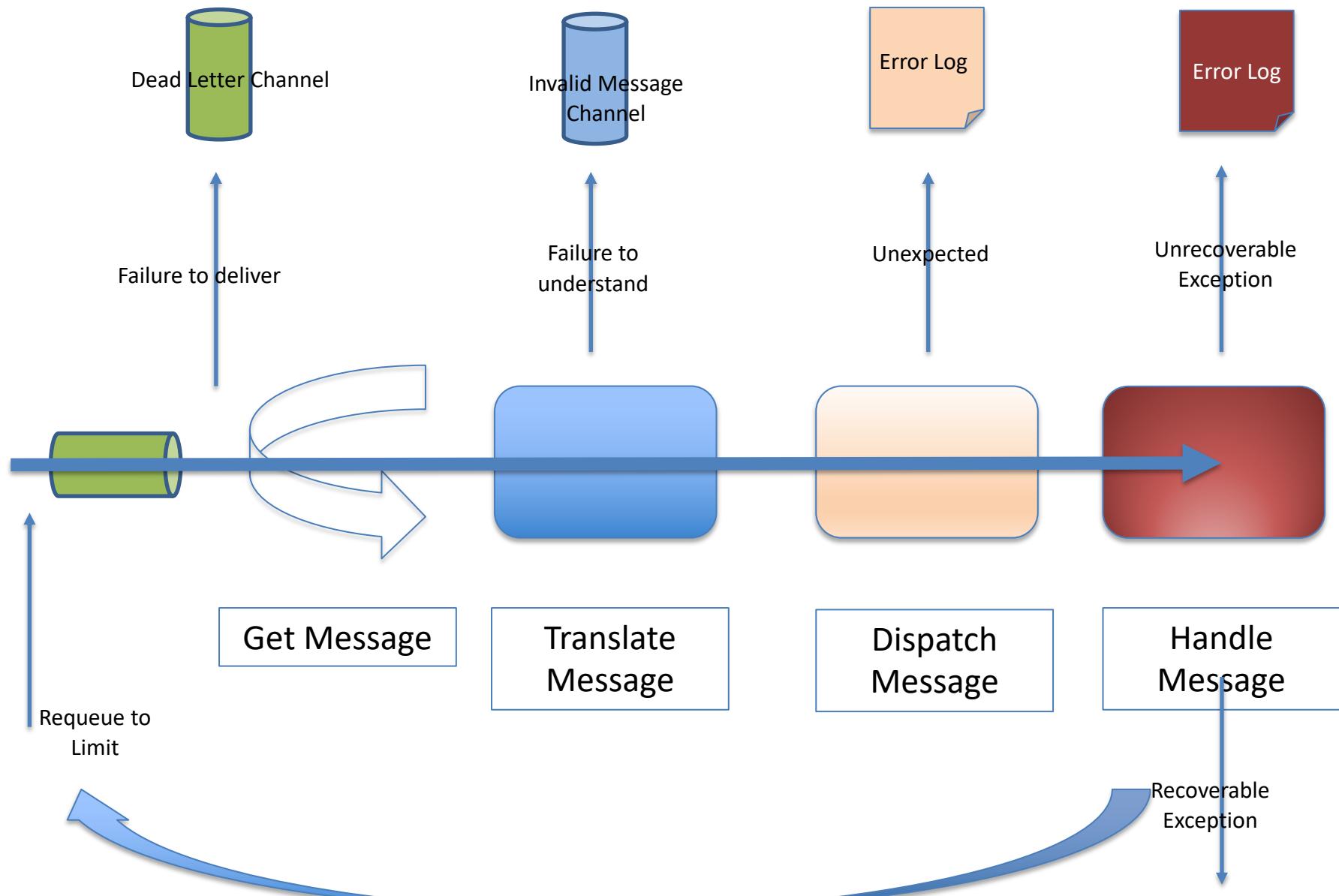
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageEndpoint.html>

Messaging Gateway

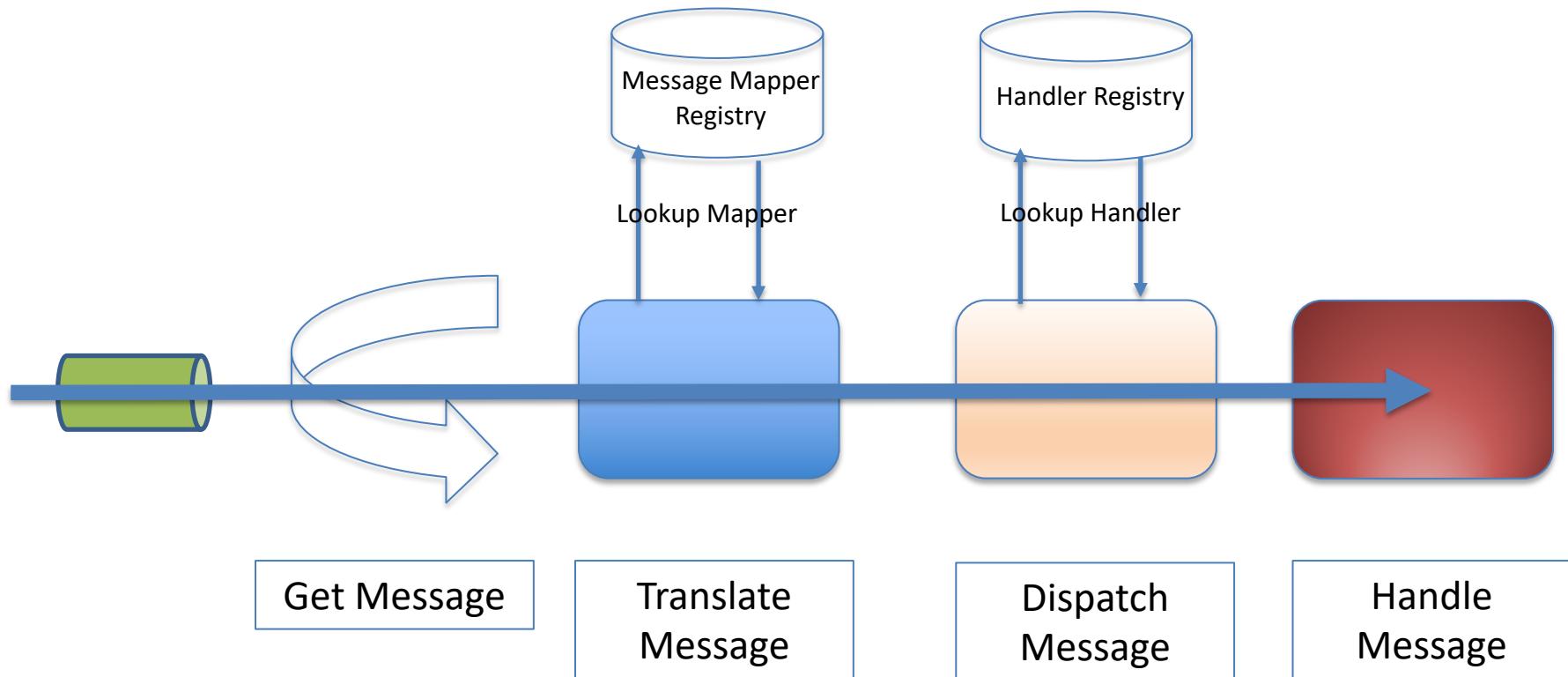


<http://www.enterpriseintegrationpatterns.com/patterns/messaging/MessagingGateway.html>

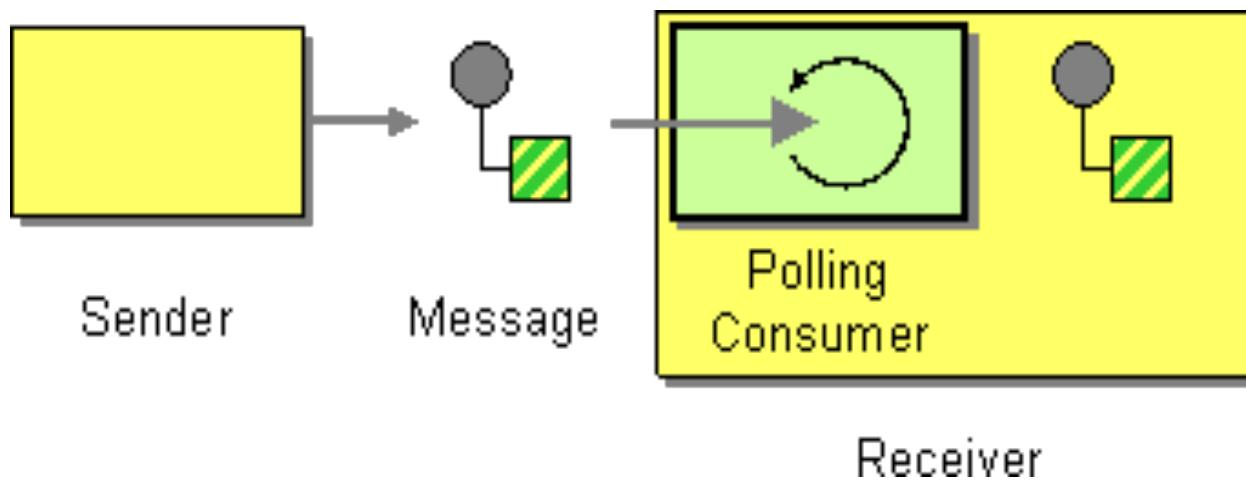
5.5 THE MESSAGE PUMP



Translate and Dispatch

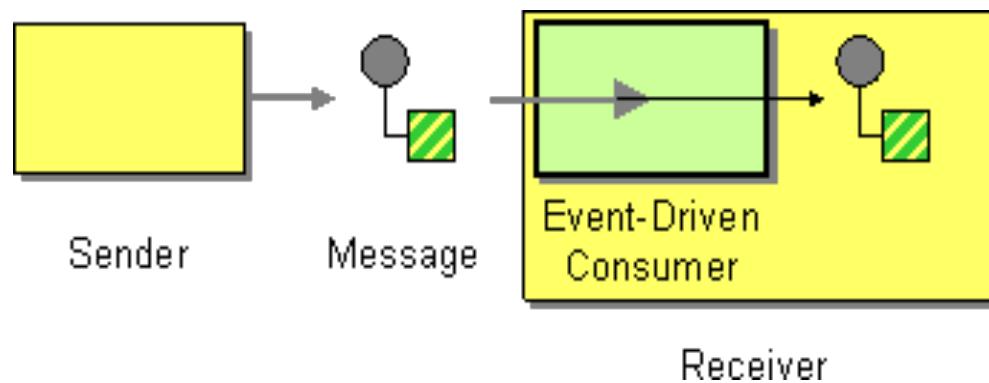


Polling Consumer



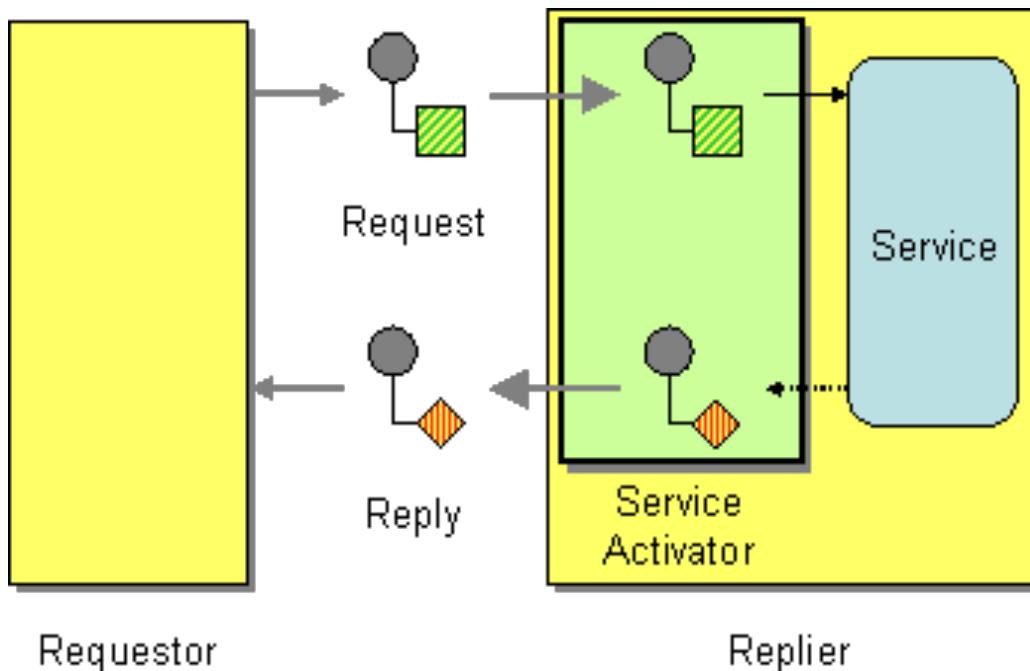
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/PollingConsumer.html>

Event Driven Consumer



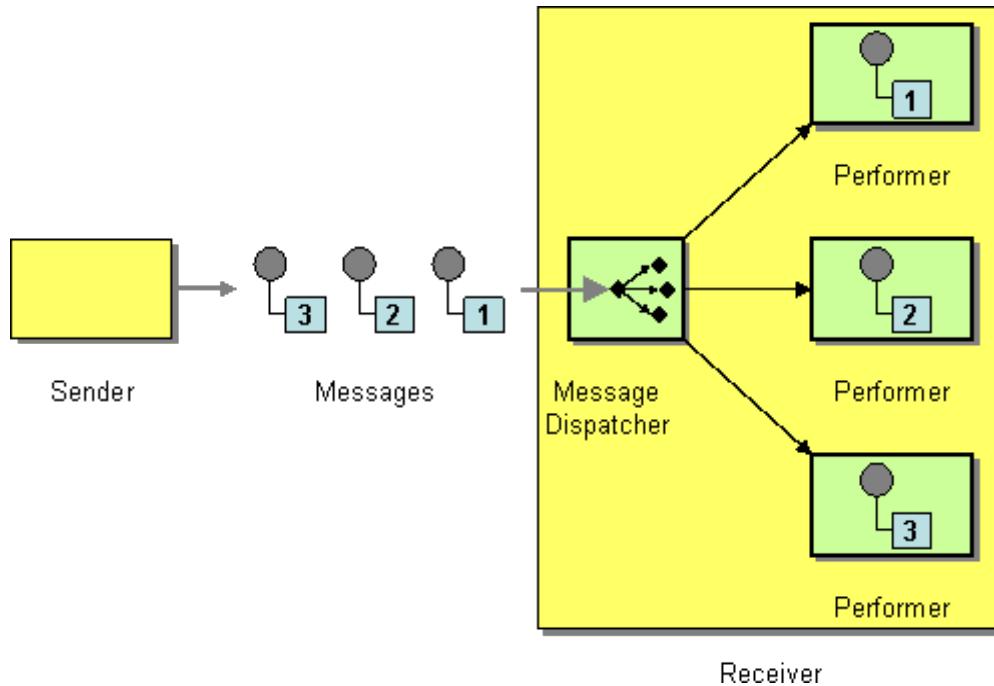
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/EventDrivenConsumer.html>

Service Activator



<http://www.enterpriseintegrationpatterns.com/patterns/messaging/MessagingAdapter.html>

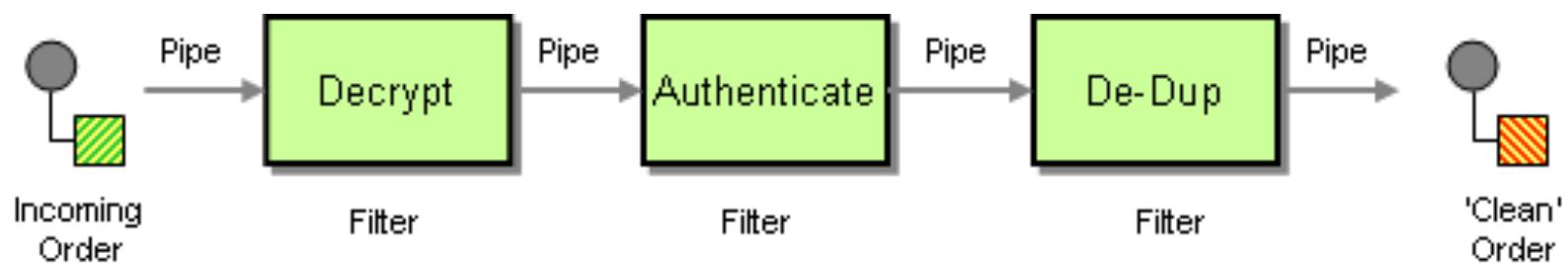
Competing Consumers



<http://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageDispatcher.html>

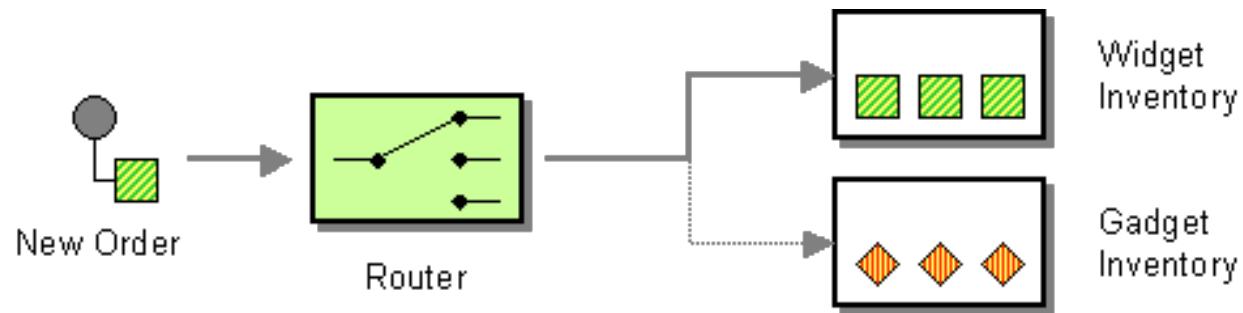
5.6 PIPELINES

Pipes and Filters



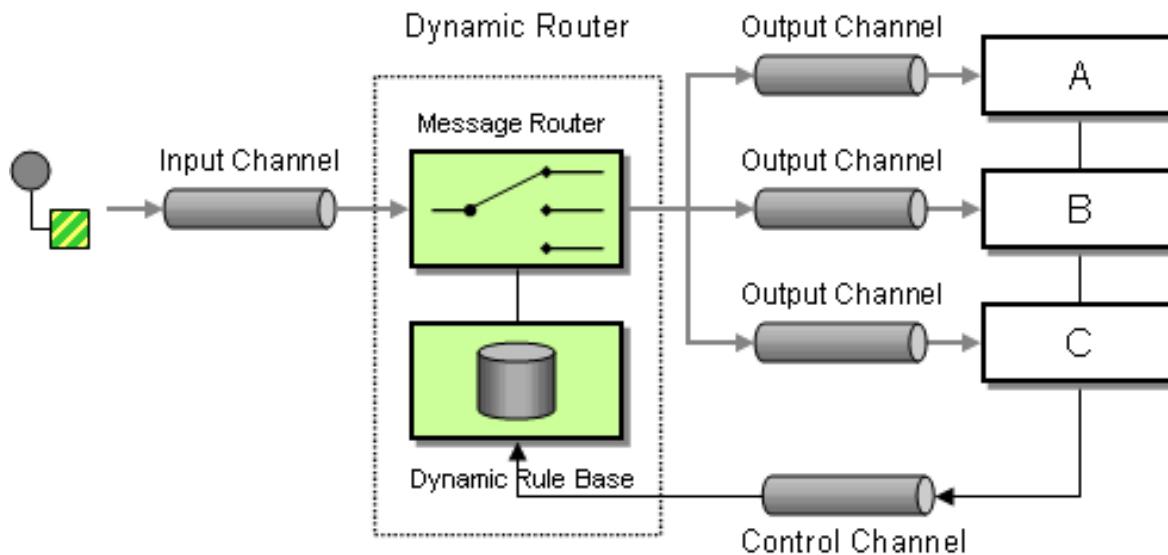
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/PipesAndFilters.html>

Content Based Router



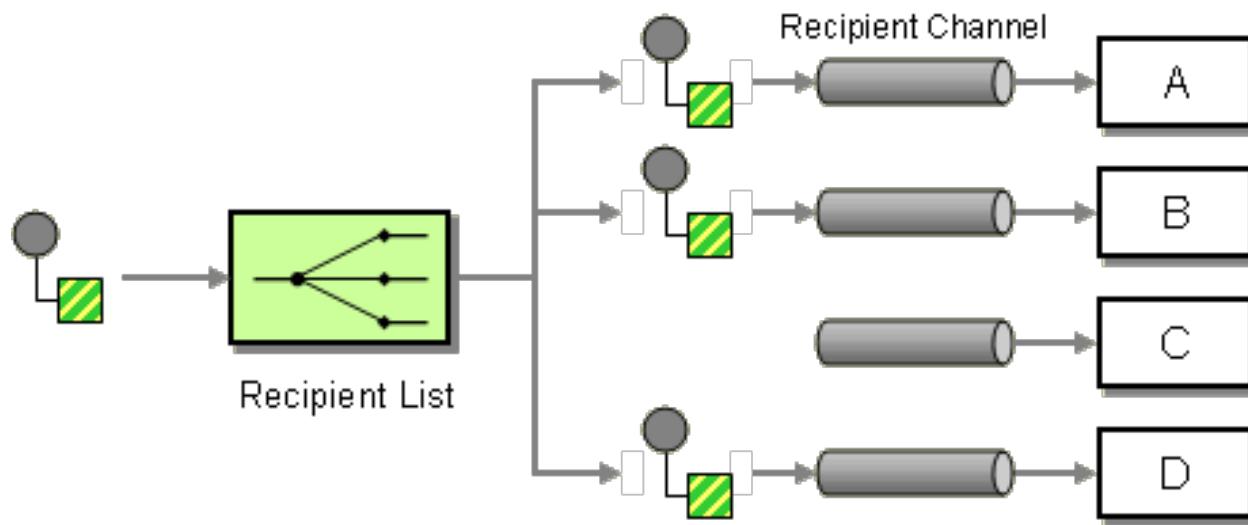
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/ContentBasedRouter.html>

Dynamic Router



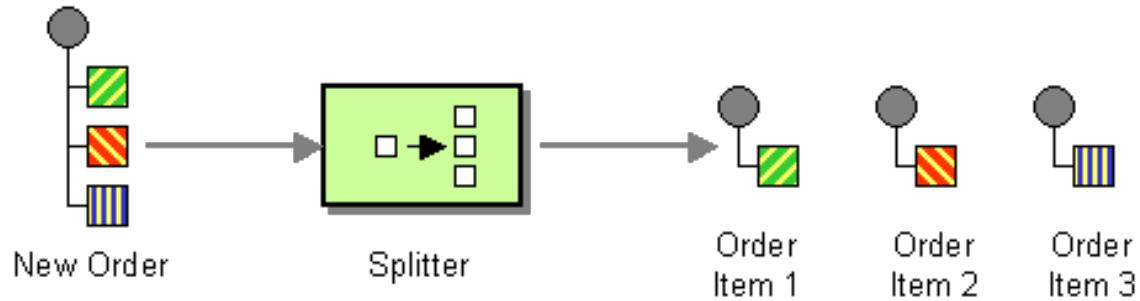
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/DynamicRouter.html>

Recipient List



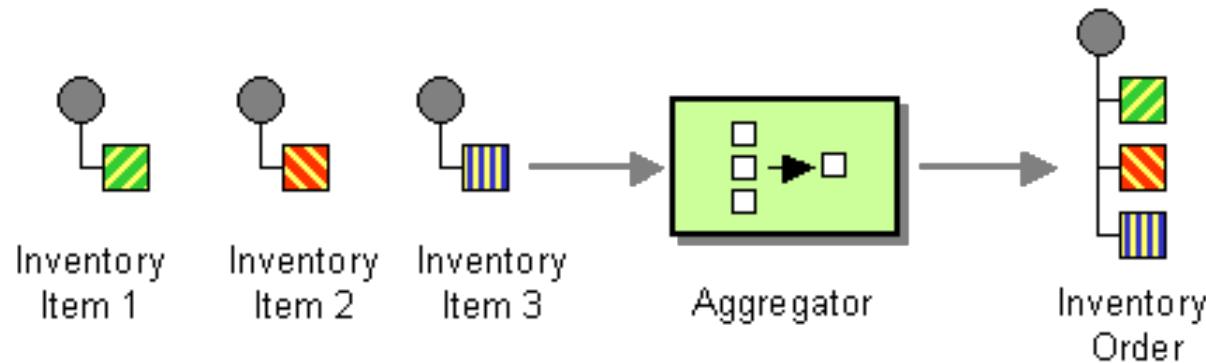
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/RecipientList.html>

Splitter



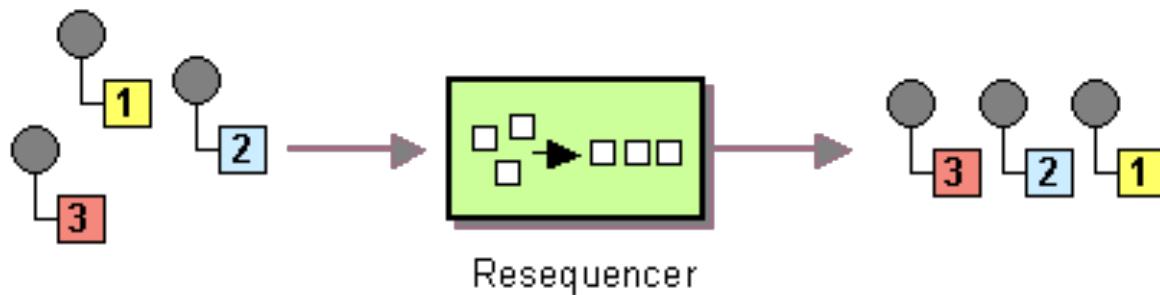
<http://www.enterpriseintegrationpatterns.com/patterns/messaging/Sequencer.html>

Aggregator



<http://www.enterpriseintegrationpatterns.com/patterns/messaging/Aggregator.html>

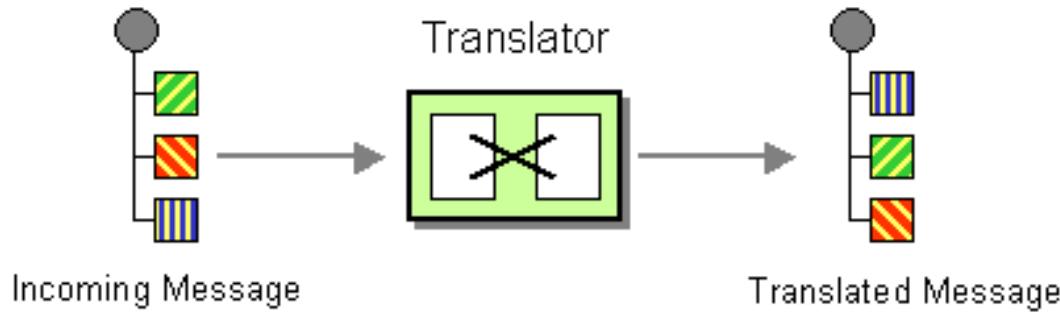
Resequencer



<http://www.enterpriseintegrationpatterns.com/patterns/messaging/Resequencer.html>

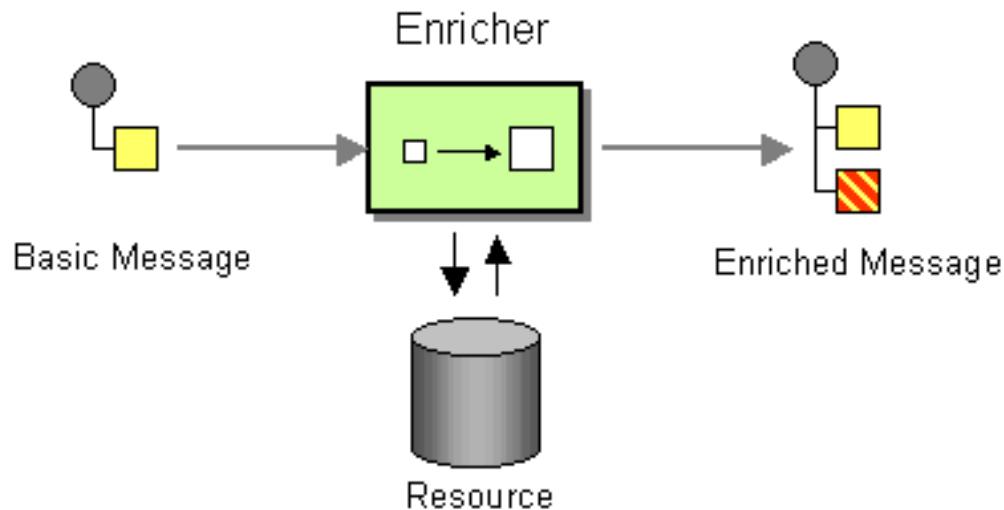
5.7 TRANSFORMATION

Message Translator



<http://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageTranslator.html>

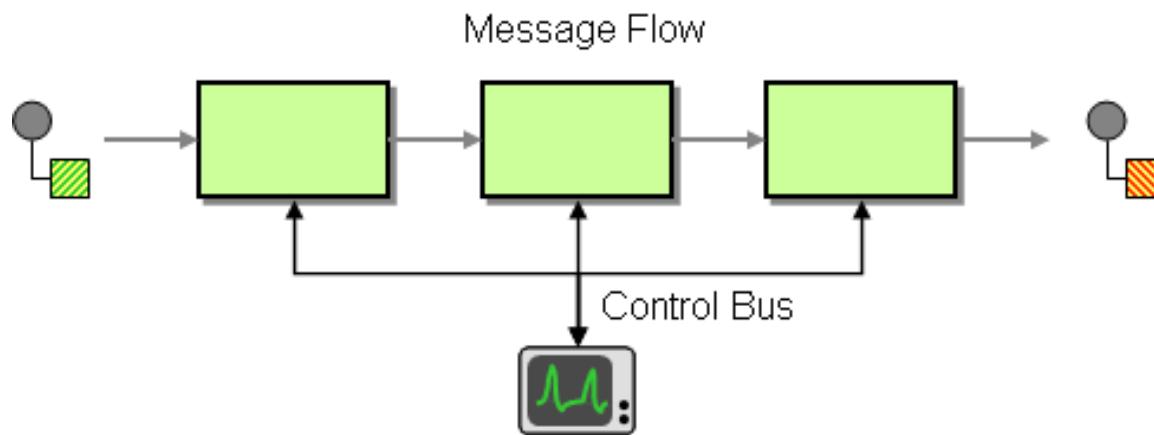
Content Enricher



<http://www.enterpriseintegrationpatterns.com/patterns/messaging/DataEnricher.html>

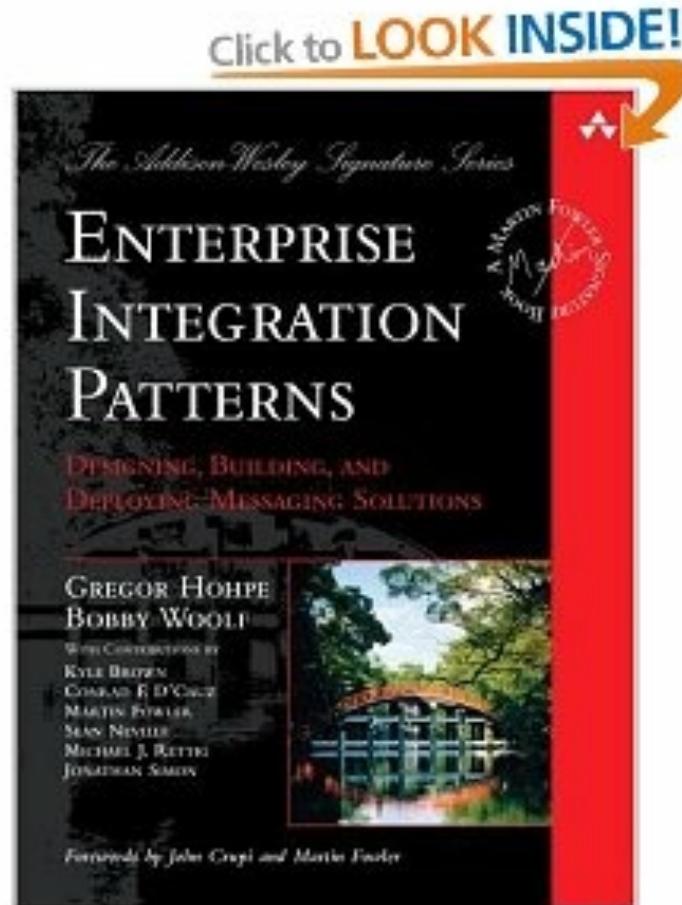
5.8 MANAGEMENT

Control Bus



<http://www.enterpriseintegrationpatterns.com/patterns/messaging/ControlBus.html>

Further Reading



Q&A