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

BRIGHTER

HOME INTRO TUTORIALS CONCEPTS DOCS

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

The Price of Distribution

Fallacies of Distributed Computing

The network is reliable.

Latency is zero.

Bandwidth is infinite.

The network is secure.

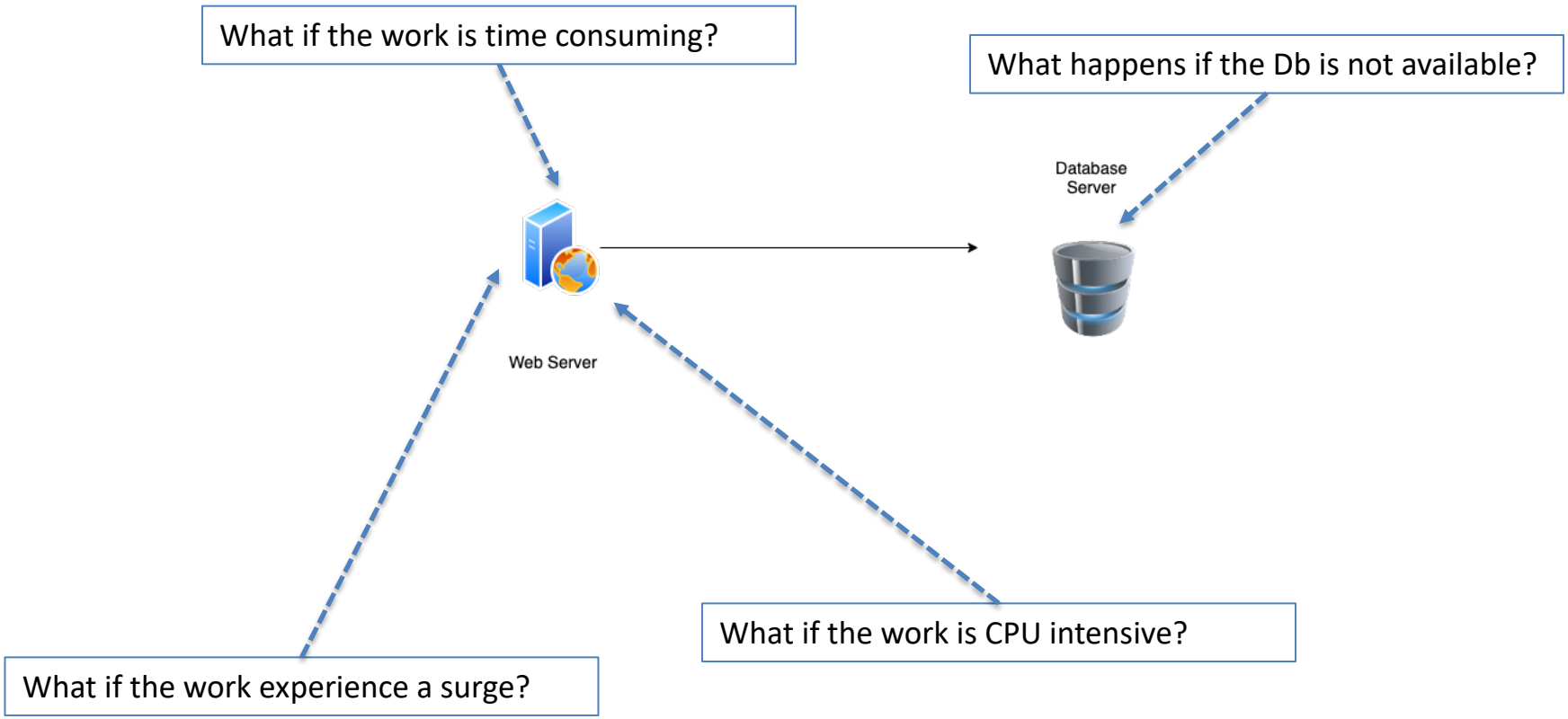
Topology doesn't change.

There is one administrator.

Transport cost is zero.

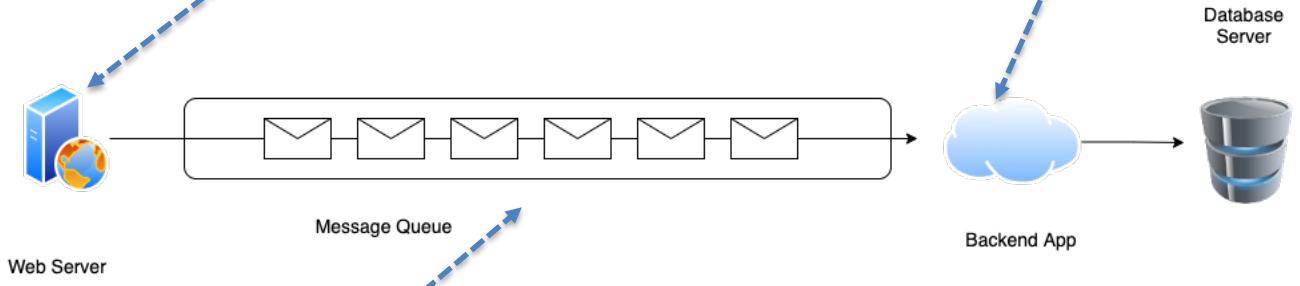
The network is homogeneous.

Example: Task Queues


What if the work is time consuming?
What happens if the Db is not available?
Database Server
Web Server
What if the work is CPU intensive?
What if the work experience a surge?

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

The web server puts the work on a queue

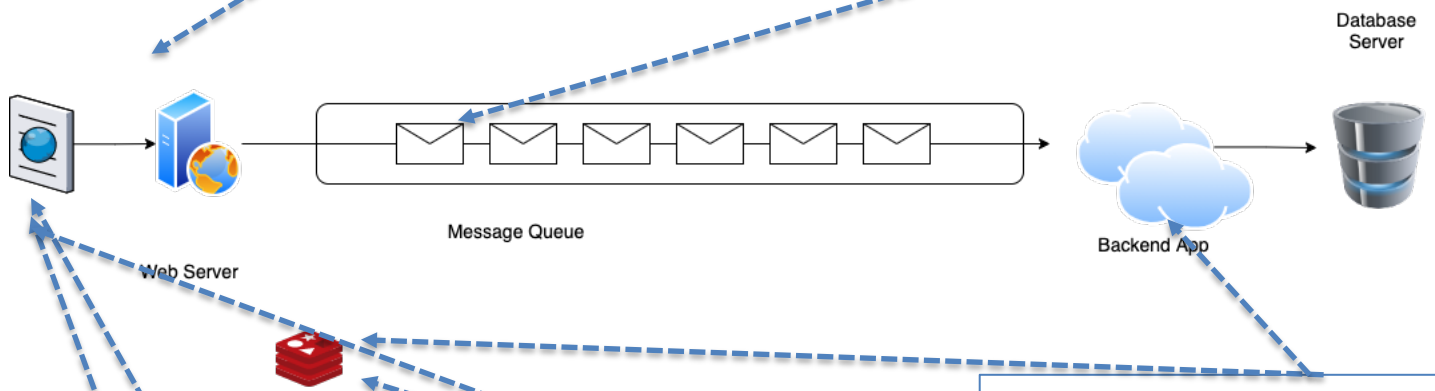


The queue stores work until we are ready to consume it. We can *throttle* to prevent surges.

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 lost it.

We enqueue a work item for the request.



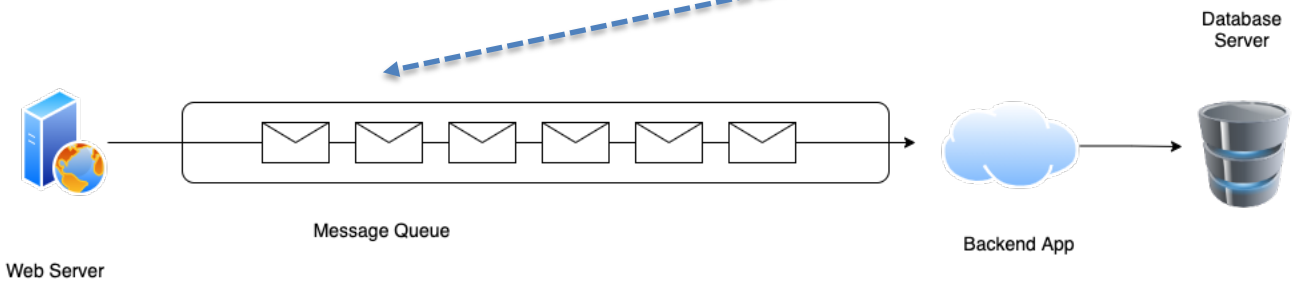
Backend app does work at sustainable pace, and updates KV store if required.

We return a link header, so you can monitor progress.

Link to a progress page, backed by a KV store where we note progress

Link to Resource – 404 until created

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.

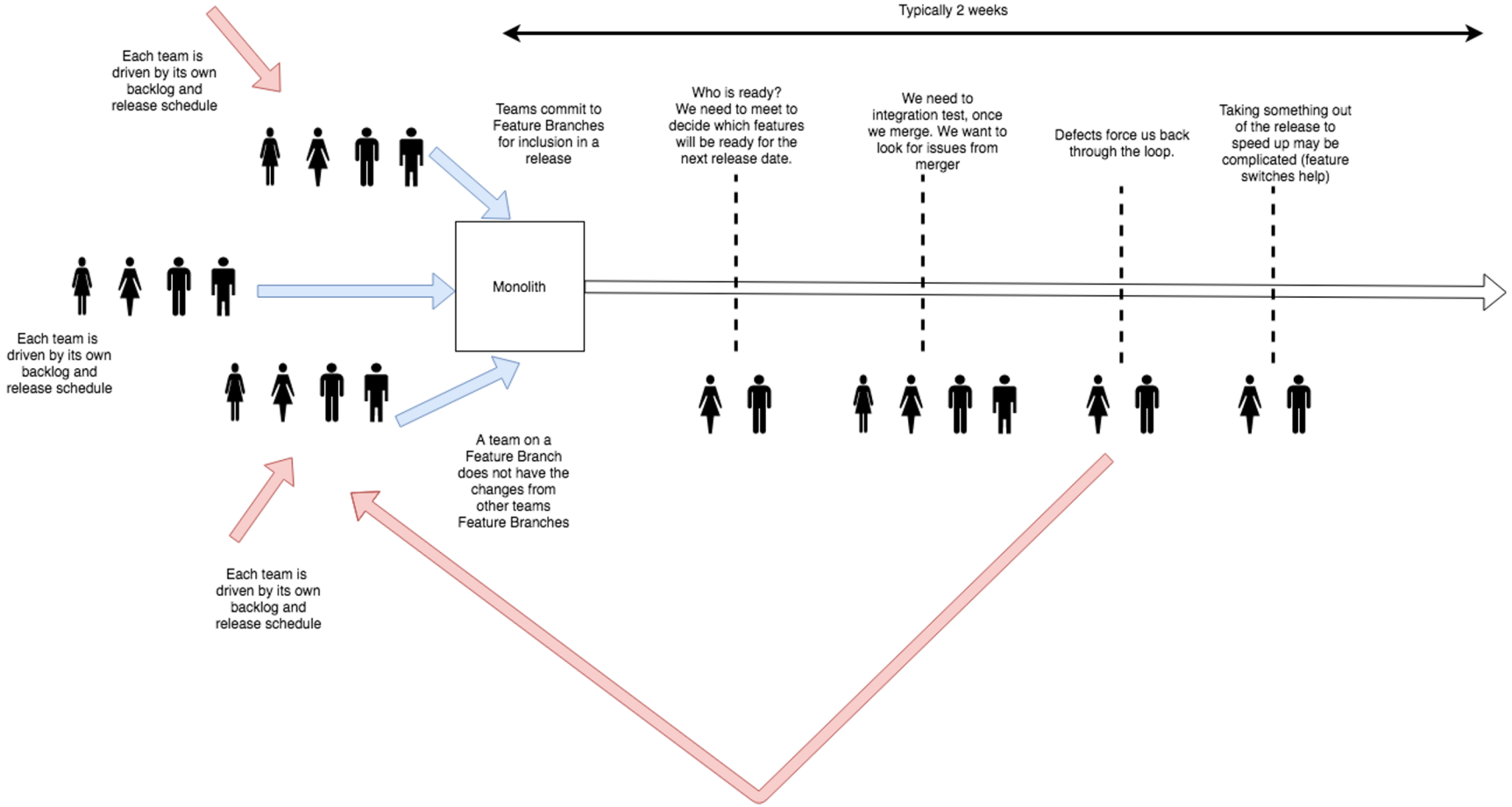
Example: Microservices

It’s all about velocity!!!

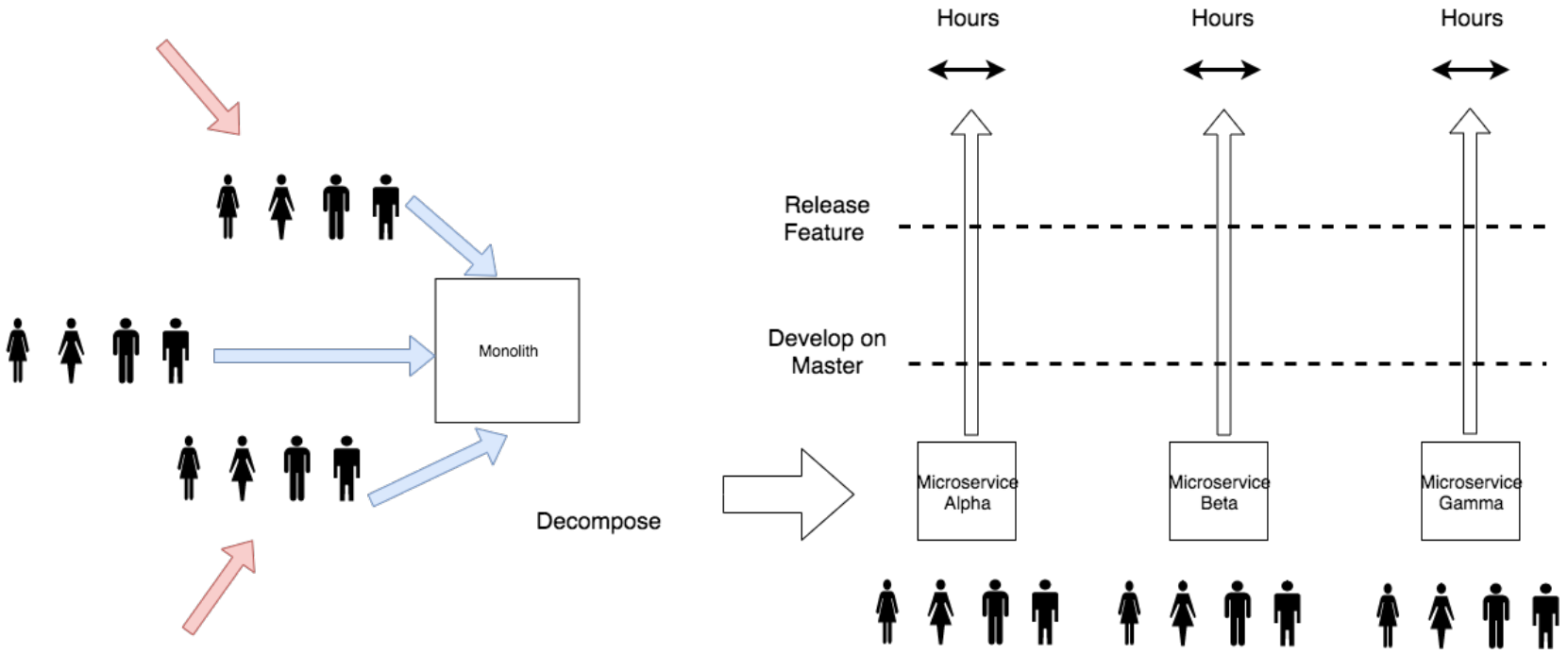
“Speed wins in the marketplace”

Adrian Cockcroft, former lead architect at Netflix

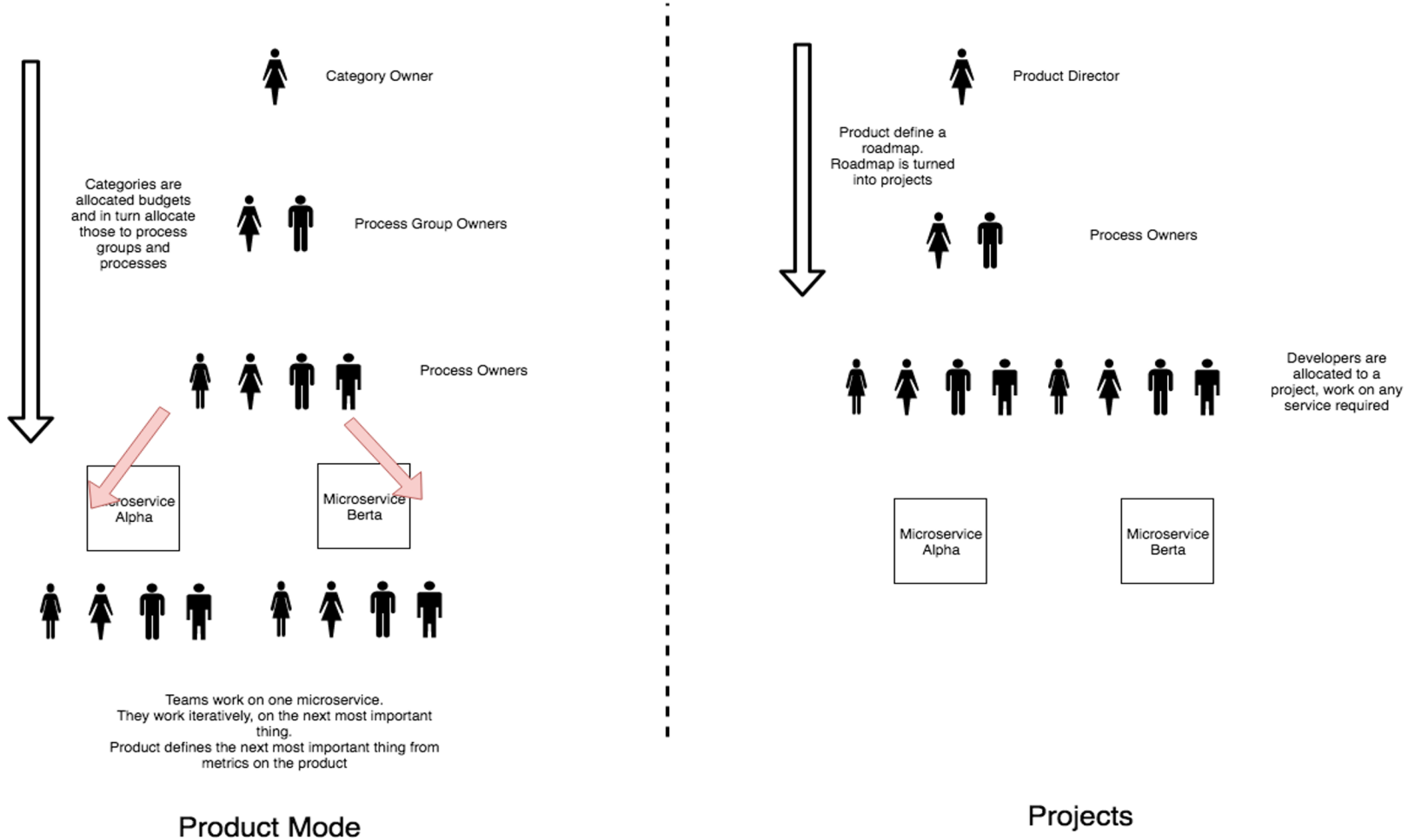
Monoliths Do Not Scale To Many Teams!



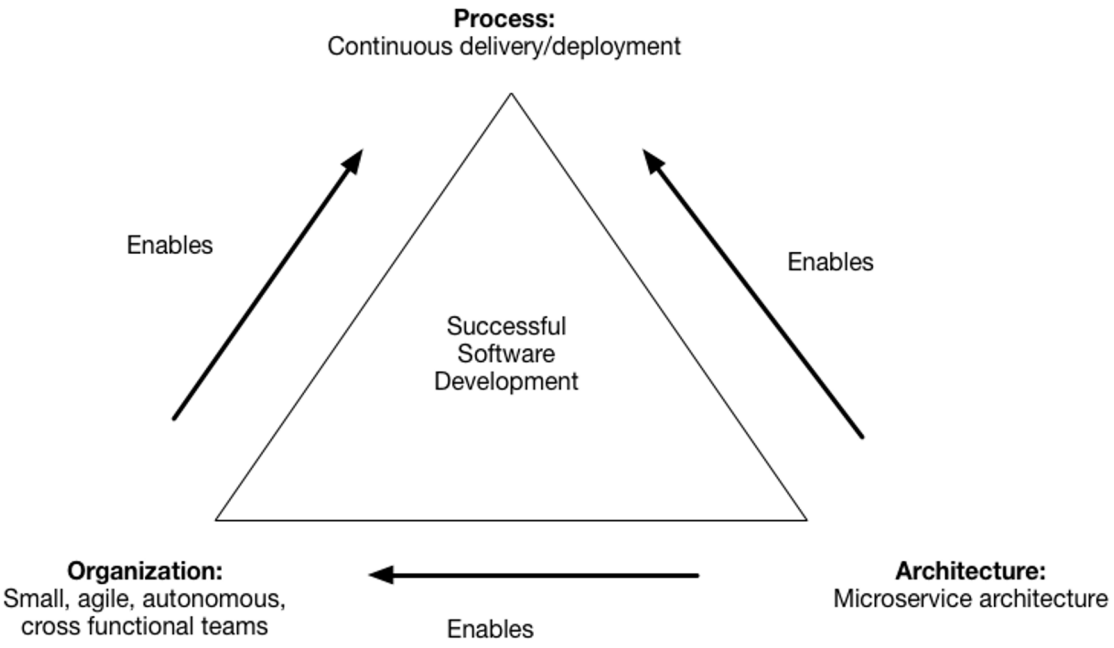
Microservices let us scale an organisation



Product Mode

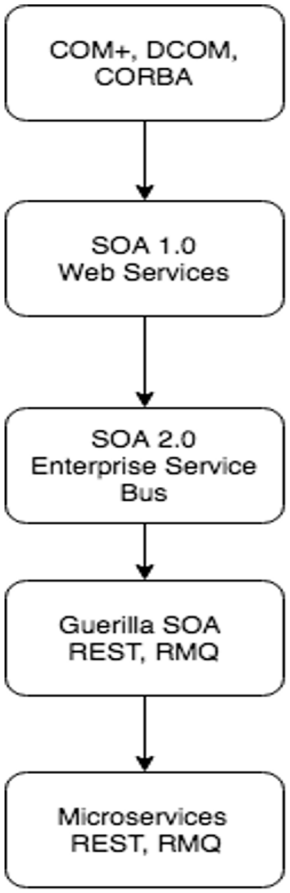


Microservices enable agility



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

A Brief History of Microservices



Distributed Object Models dominate, a merger of RPC with OO

Don Box at Microsoft creates SOAP and the 4 tenets of SOA, launching the popularity of Web Service based SOA

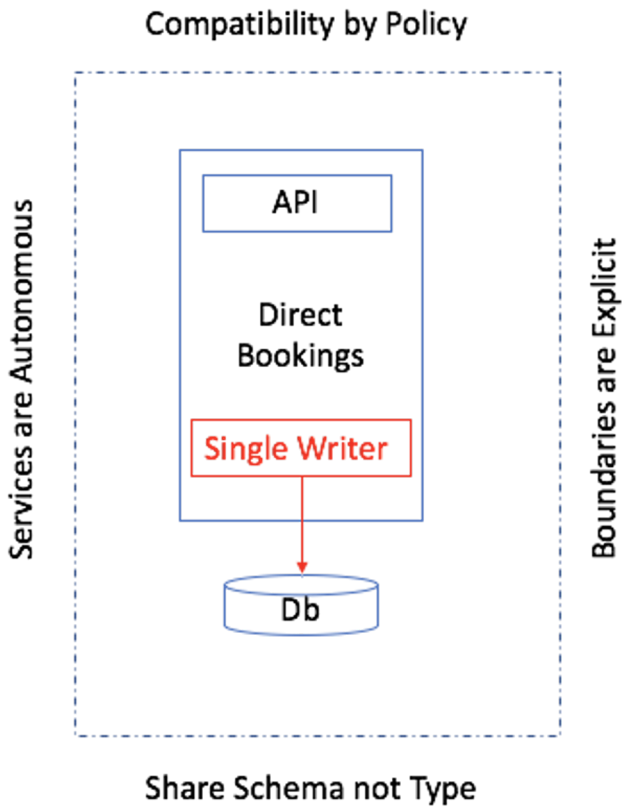
Events emerge as the key to loosely coupled services and vendors quickly own the middleware space

Jim Webber tries to break SOA away from centralized governance and the ESB favouring a bottom up approach using lightweight middleware or REST

Netflix under Adrian Cockcroft pioneers the Fine-Grained services model; Martin Fowler and James Lewis document as microservices

The definition of Microservices, we use comes from the paper by Martin Fowler and James Lewis

Microservices are partitions of software



How do we communicate between microservices?

2.INTEGRATION STYLES

File Transfer

Two processes communicate via the Producer writing to a file, and the Consumer reading from it.

A common data transfer mechanism that can be used by a variety of languages and platforms and feels neutral towards each

Requires agreement: file names, locations, who manages files

Will create eventual consistency between systems due to periodic nature of publication

Shared Database

Two processes communicate by the Producer writing to a database and the consumer reading from it.

Breaks encapsulation and causes change to ripple across all applications

Creating a unified schema that can meet the needs of all applications is a challenge

Often the Db supporting many enterprise-wide applications becomes the bottleneck

Remote Procedure Call

Two processes communicate by the client causing a procedure to execute in another address space belonging to the server, coded as if it were a local call.

Integrates functionality not data

Behavioral coupling can tie the systems together in a knot, particularly due to sequencing

Messaging

Two processes communicate by the Producer sending a packet of data to a channel and the Sender reading that packet of data from the channel.

Asynchronous communication does not require both systems to be up and ready at the same time.

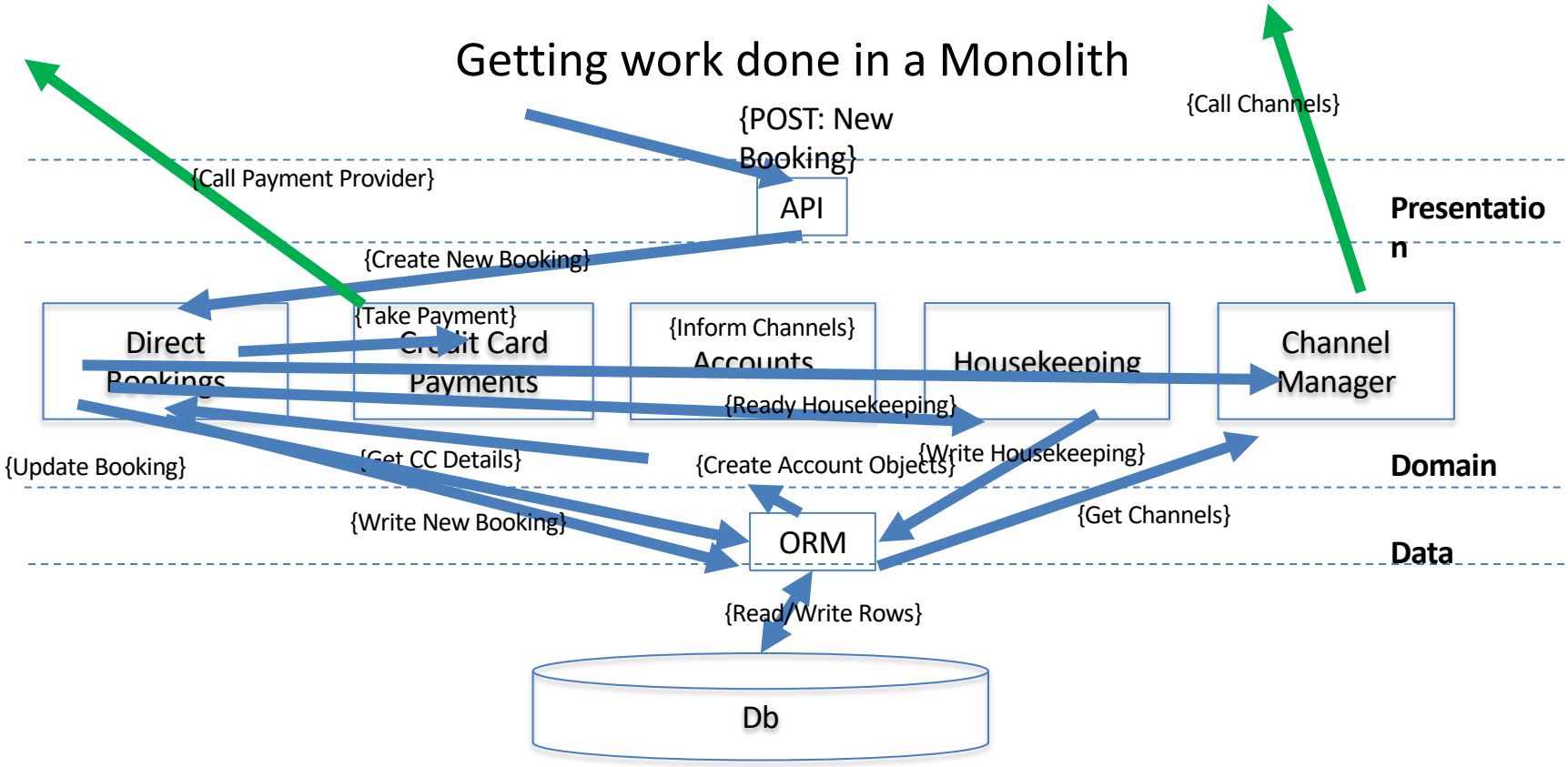
Messages can be transformed in transit without sender or receiver knowing

Small messages frequently allows behavioral as well as data collaboration

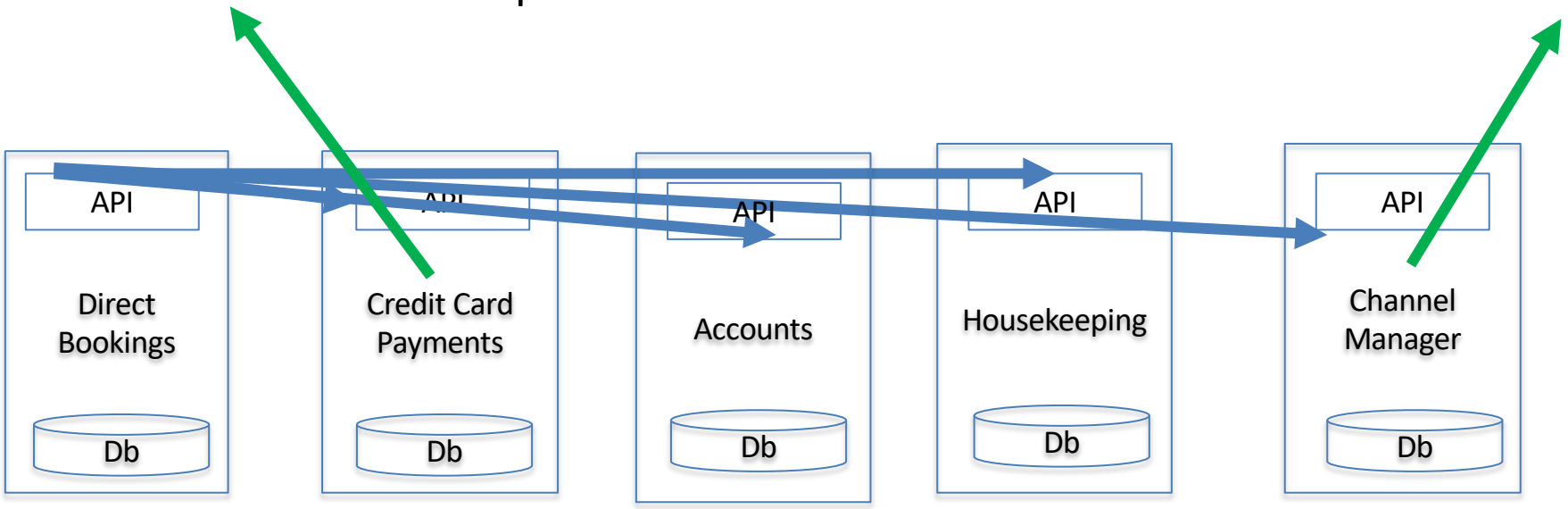
Integrating using RPC or REST

3.REQUEST DRIVEN ARCHITECTURES

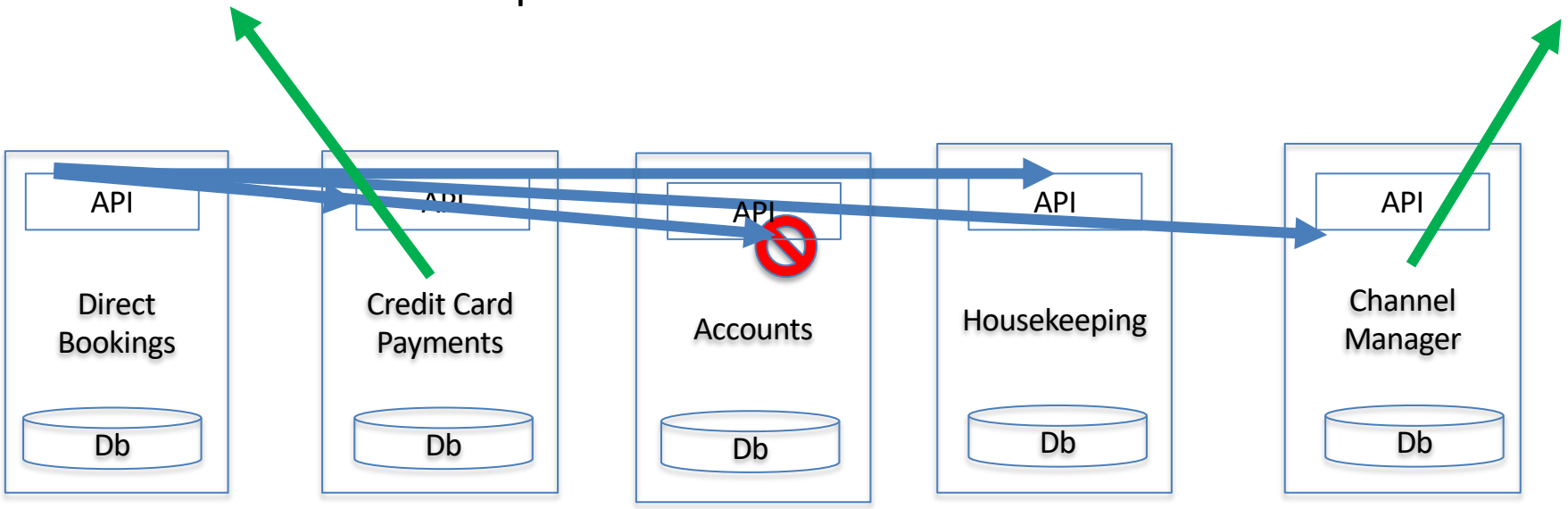
Getting work done in a Monolith

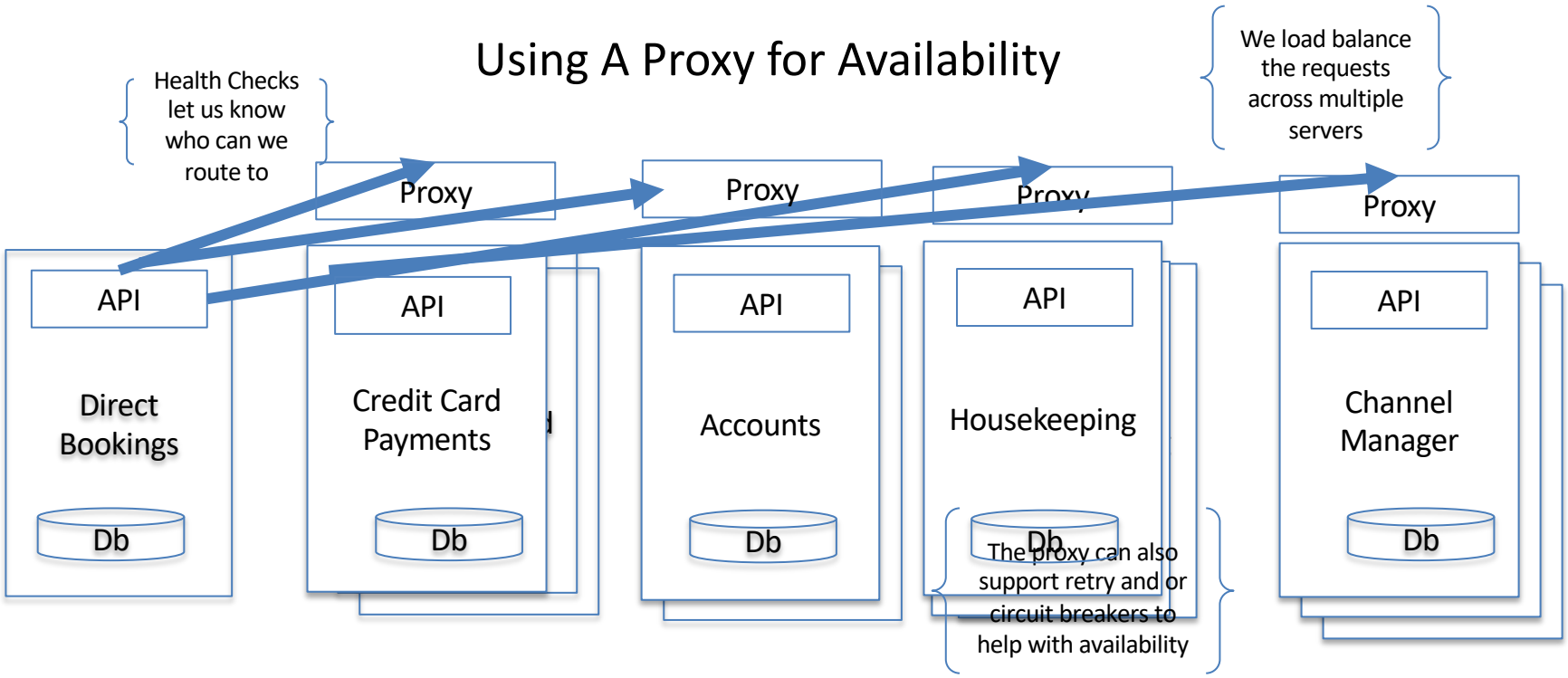


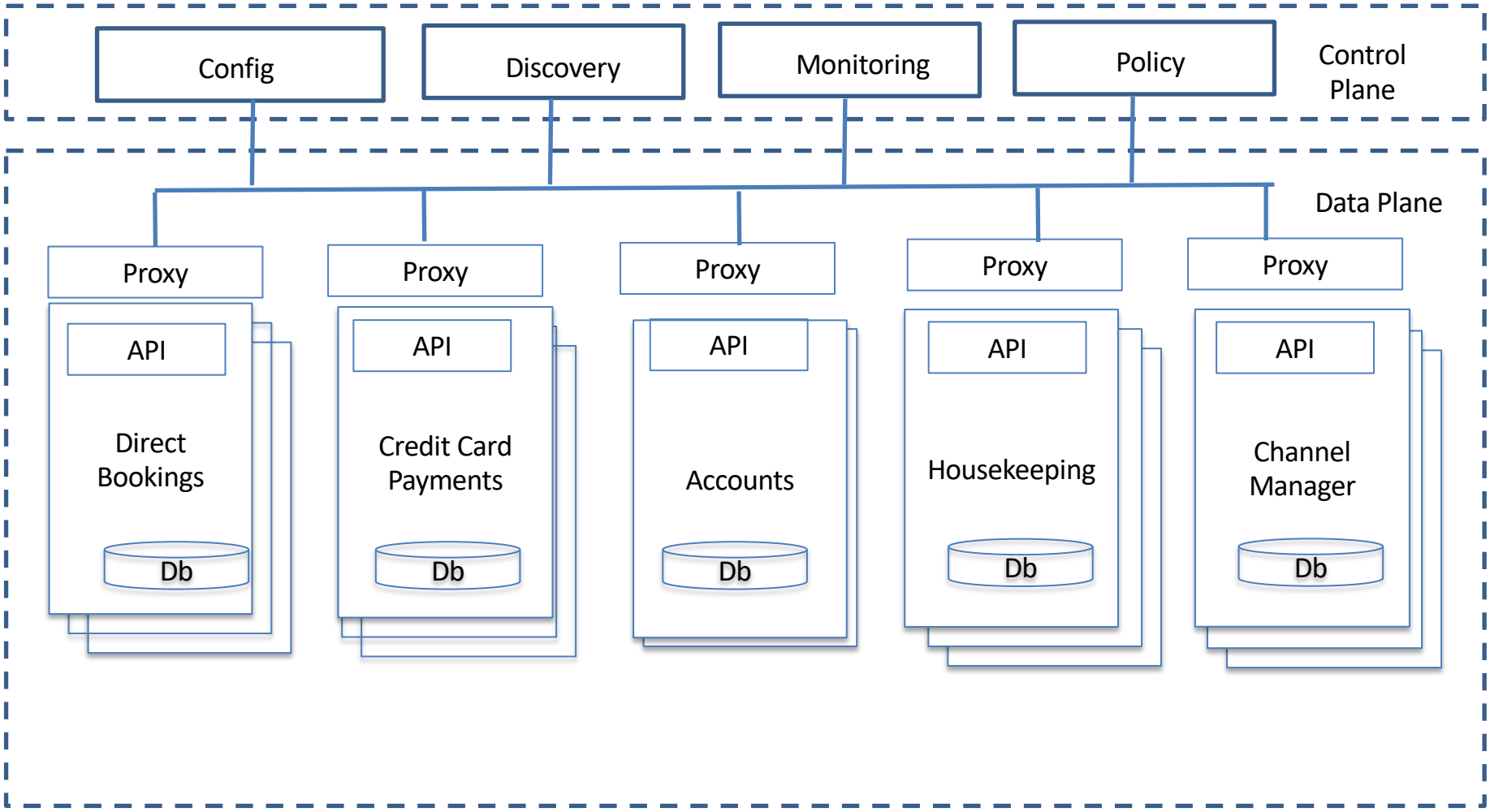
Request Driven Architecture



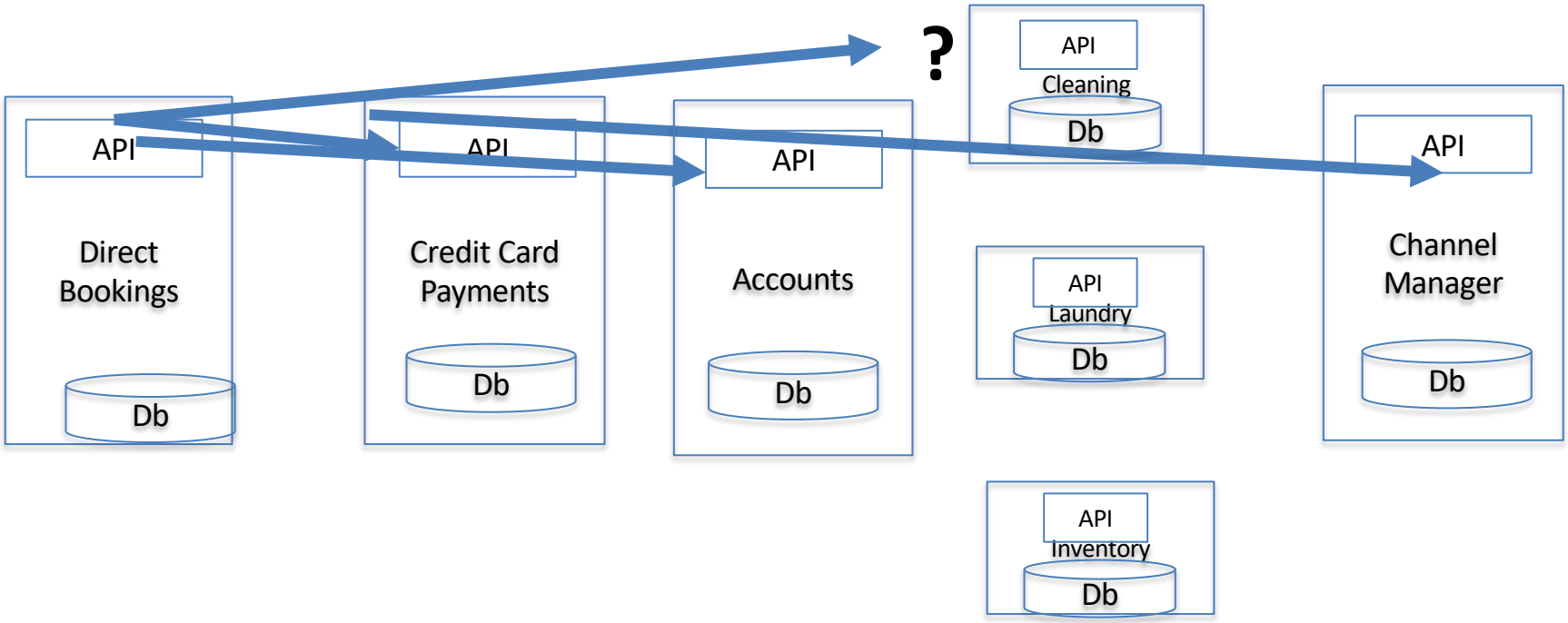
Request Driven Architecture




Using A Proxy for Availability
Health Checks let us know who can we route to
We load balance the requests across multiple servers
Proxy
Proxy
Proxy
Proxy
API
API
API
API
API
Direct Bookings
Credit Card Payments
Accounts
Housekeeping
Channel Manager
Db
Db
Db
Db
Db
The proxy can also support retry and or circuit breakers to help with availability

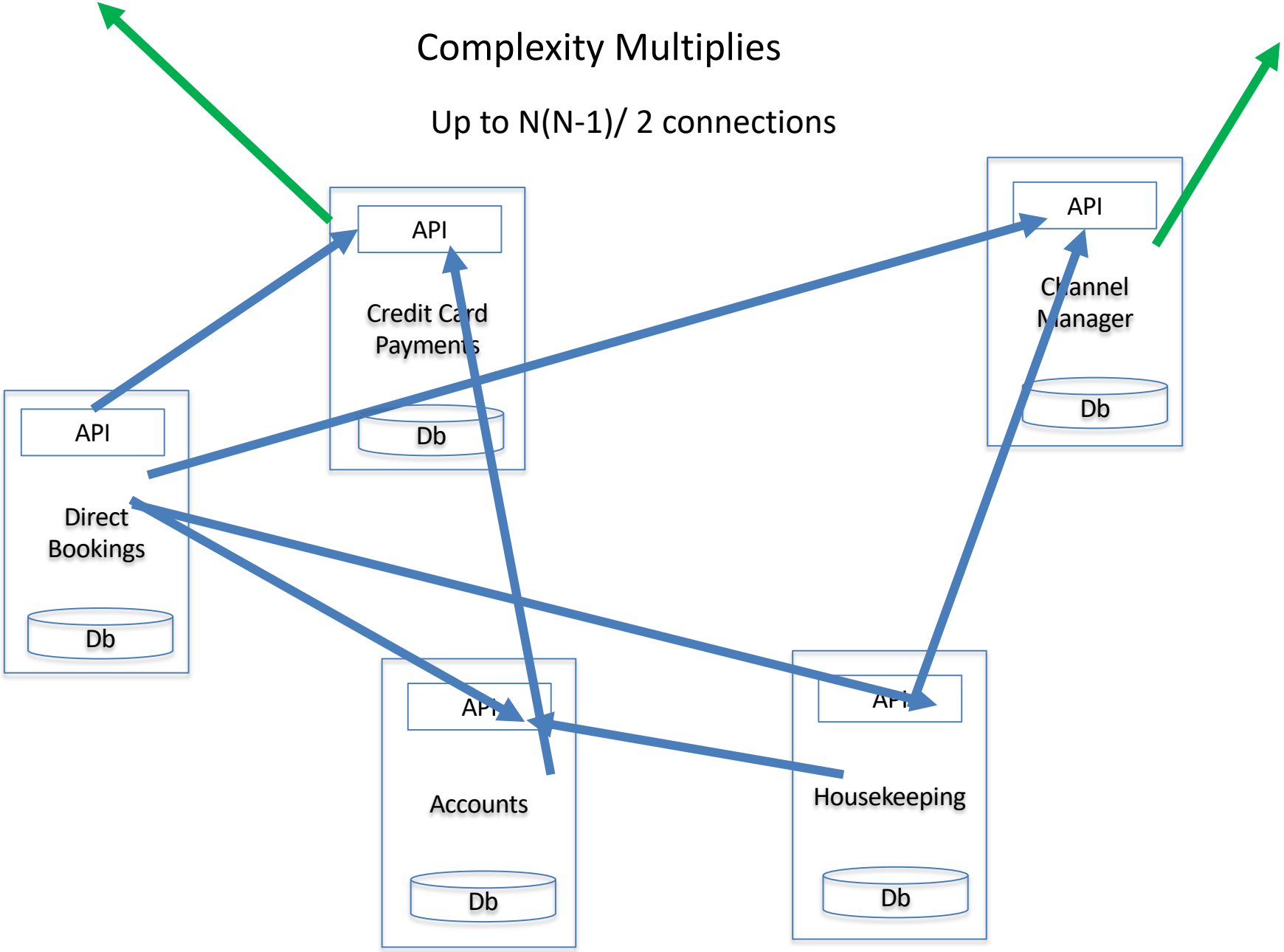

Config
Discovery
Monitoring
Policy
Control Plane
Data Plane
Proxy
Proxy
Proxy
Proxy
Proxy
API
API
API
API
API
Direct Bookings
Credit Card Payments
Accounts
Housekeeping
Channel Manager
Db
Db
Db
Db
Db

Change Happens



Complexity Multiplies

Up to N(N-1)/ 2 connections



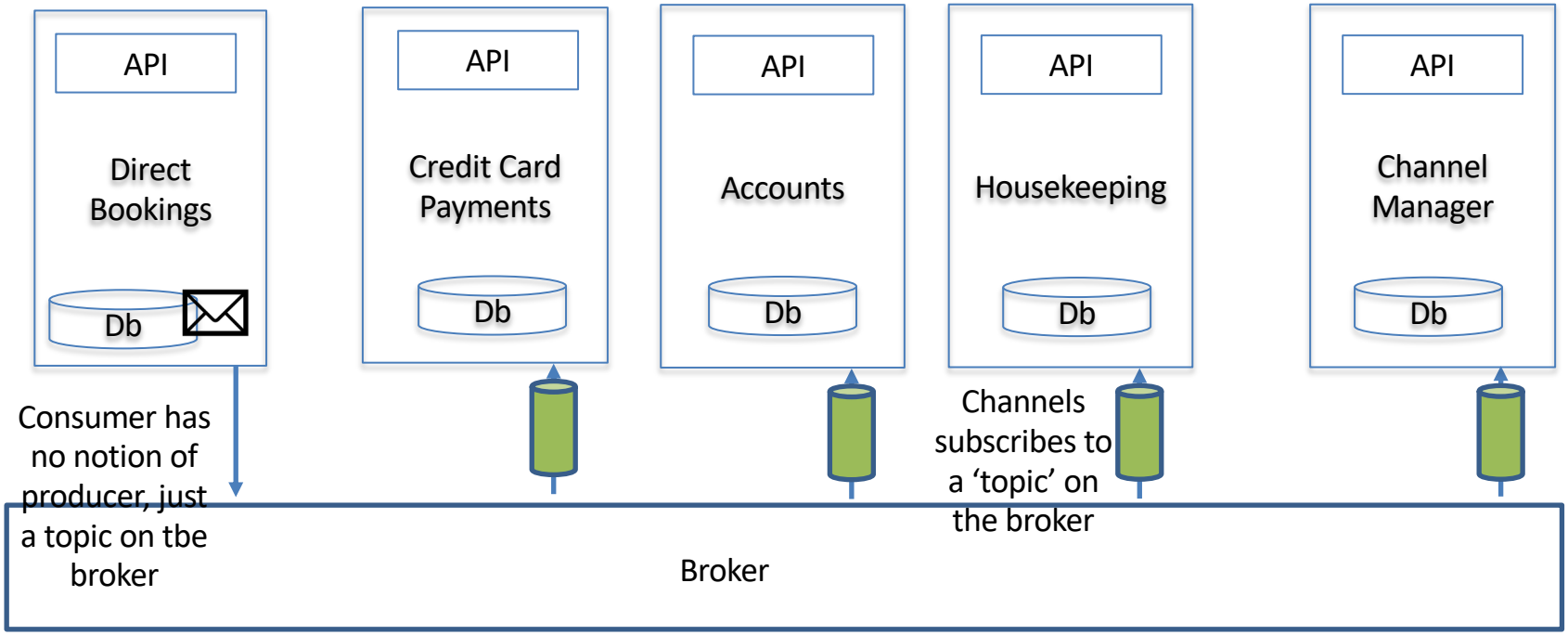
Integrating using events

4.EVENT DRIVEN ARCHITECTURES

Event Driven Architecture

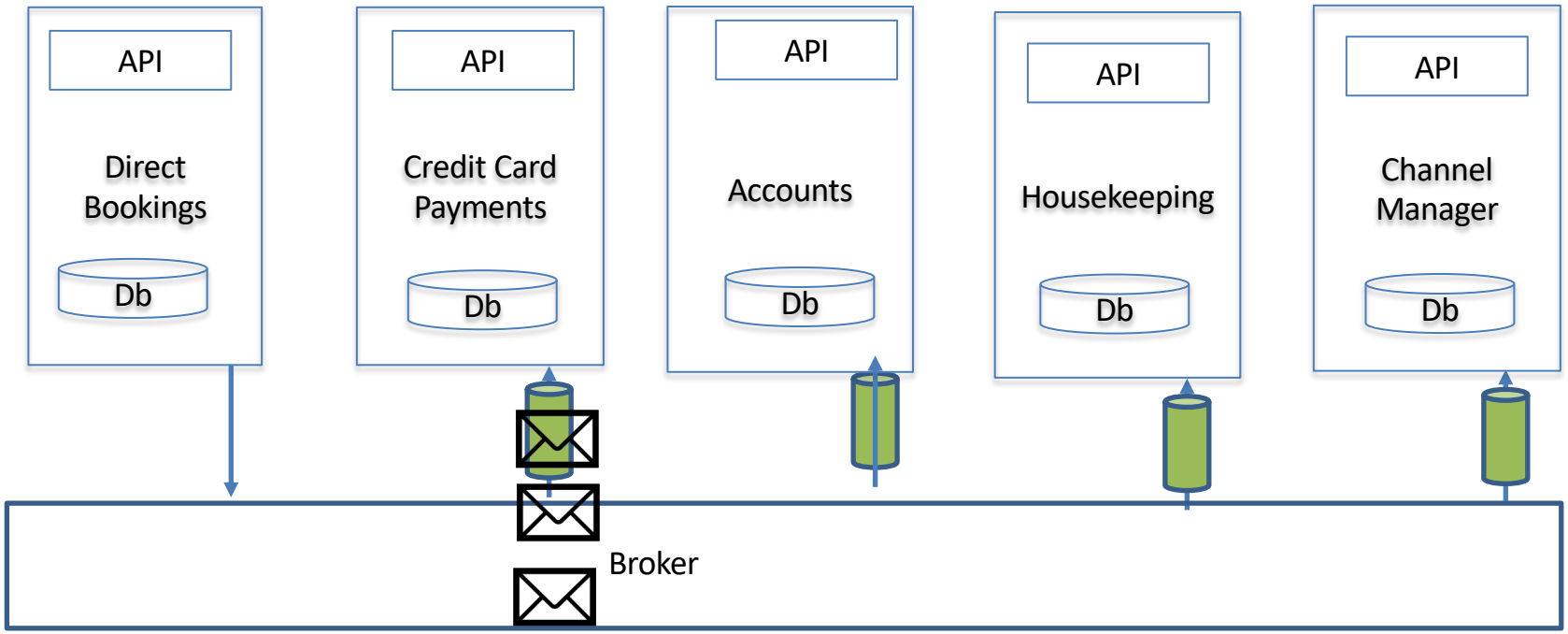
"Messaging over a lightweight message bus such as RabbitMQ"

Fowler and Lewis

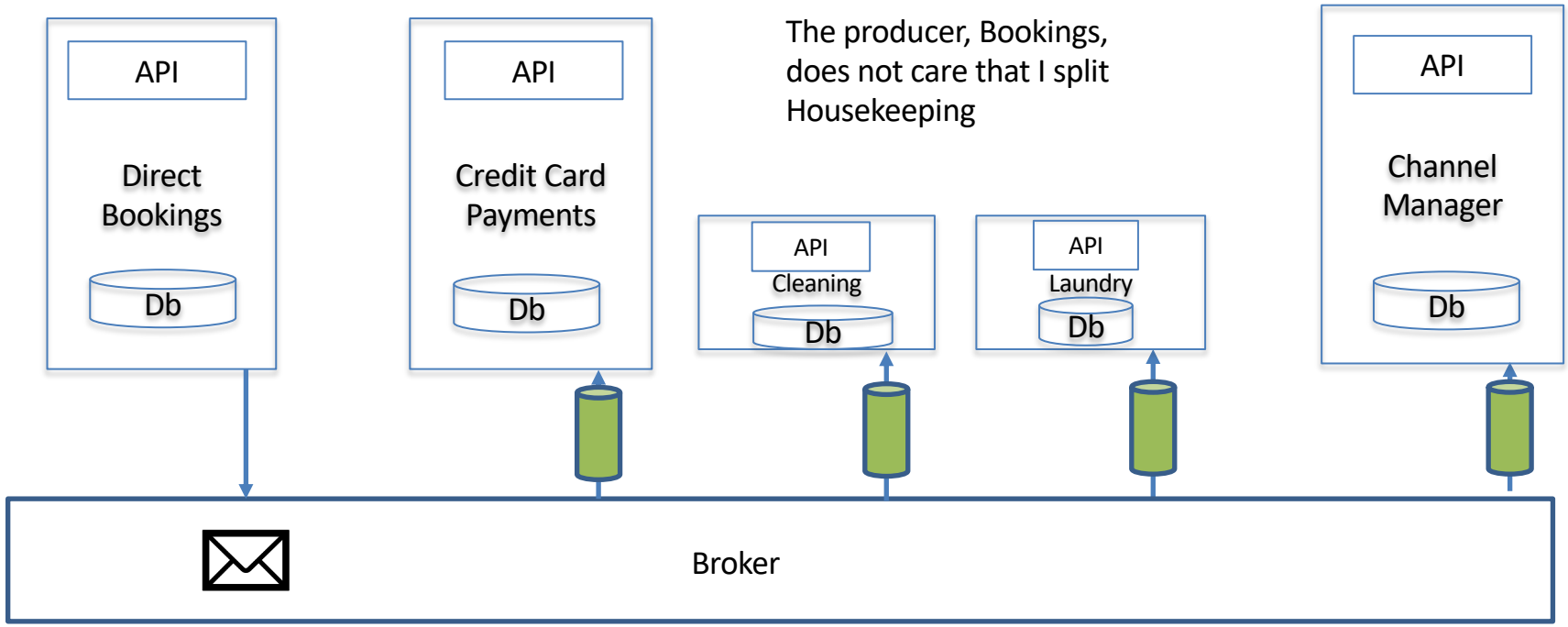


Event Driven Architecture

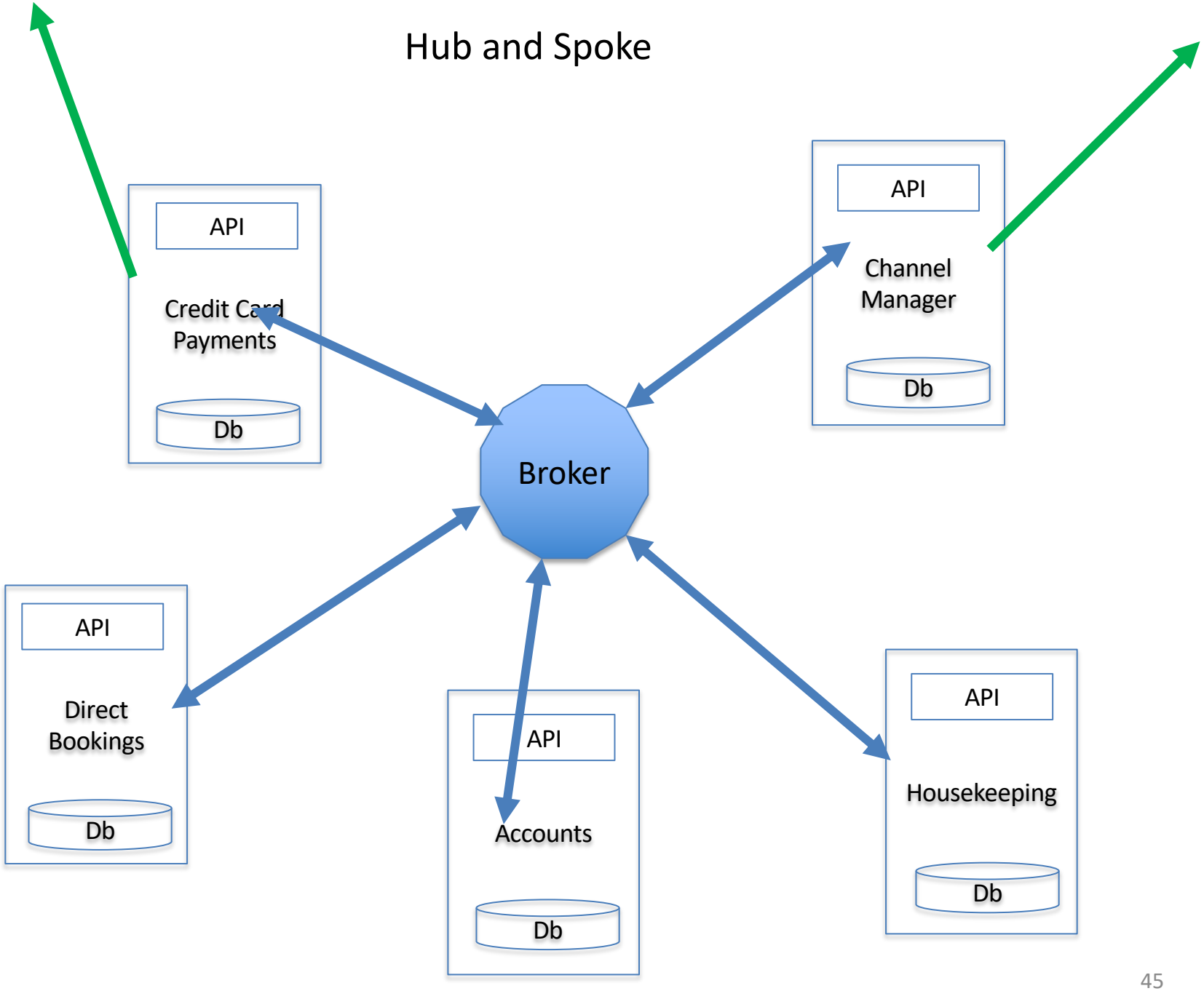
Channel provides guaranteed at least once delivery



Event Driven Architecture

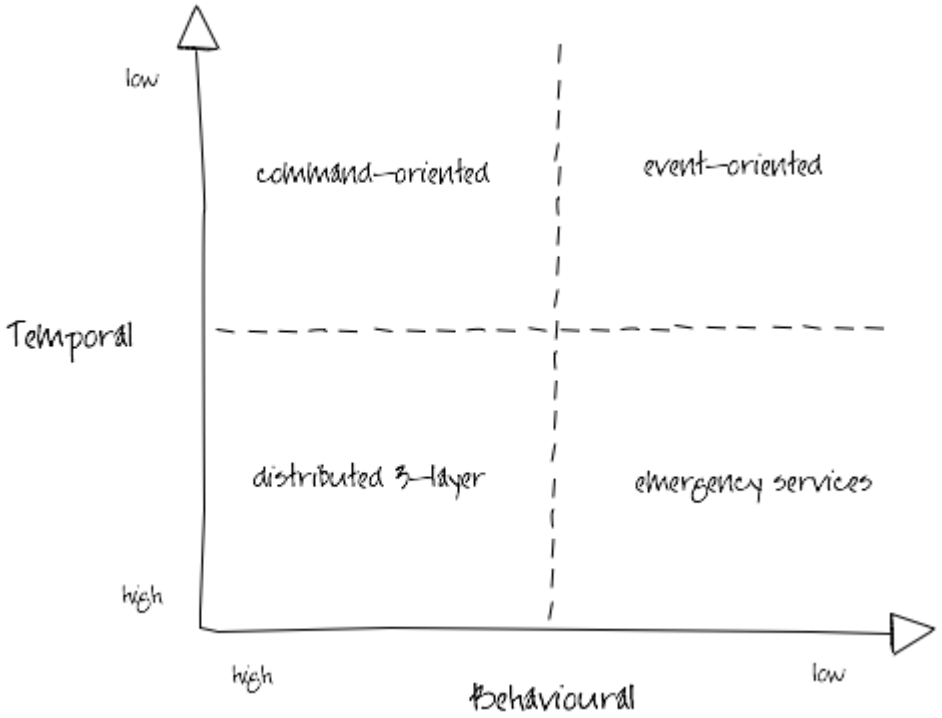


Hub and Spoke



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

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

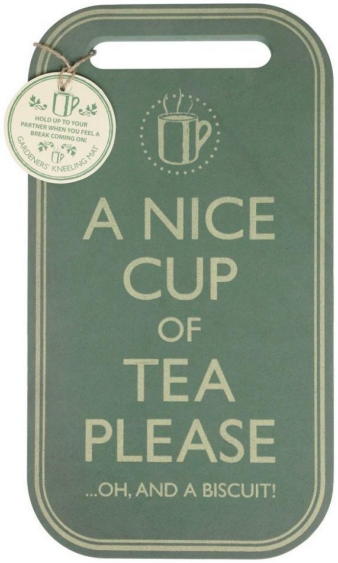
When an application sends a message, how can it get a response from the receiver?

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.

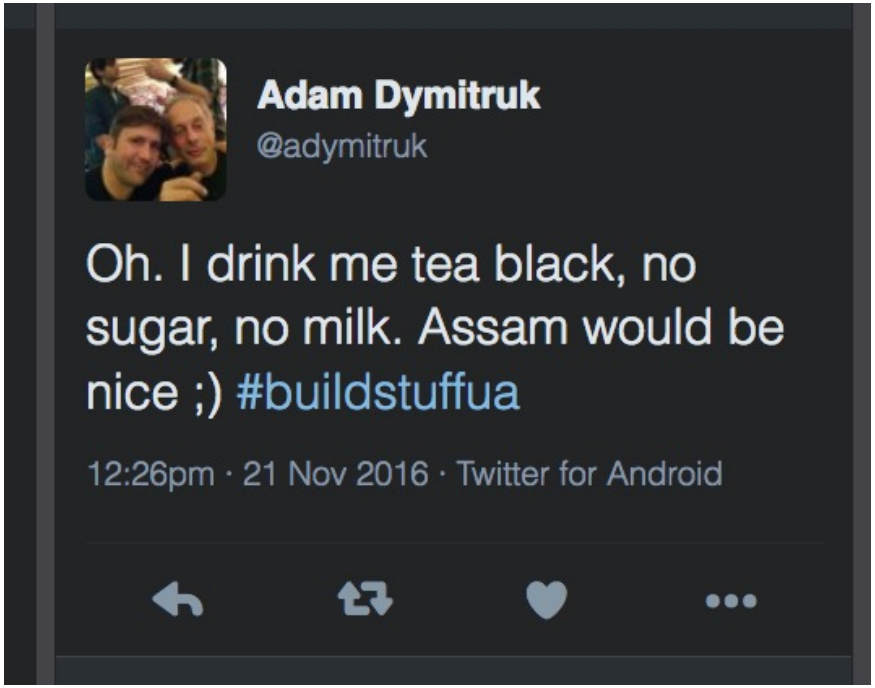
Command or Event?



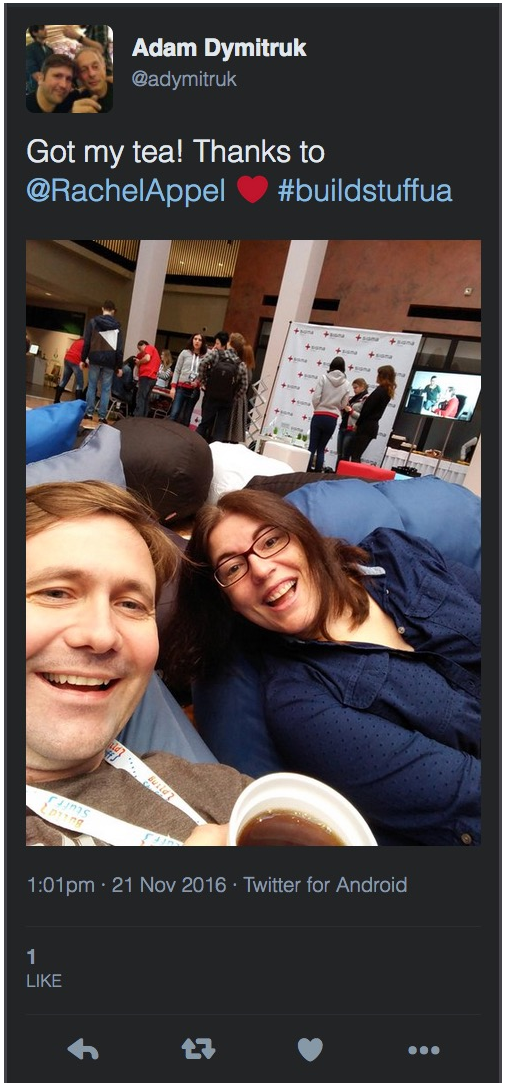
A Command

An Event


View original
Flag media
Adam Dymitruk @adymitruk
38m
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


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


Adam Dymitruk
@adymitruk
Got my tea! Thanks to
@RachelAppel ❤ #buildstuffua
1:01pm · 21 Nov 2016 · Twitter for Android
1
LIKE

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

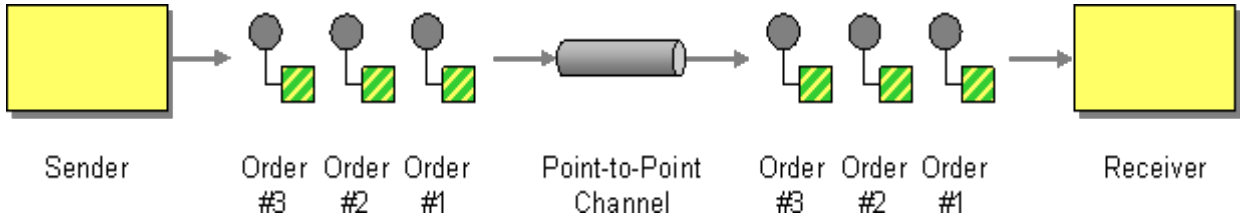
Logical Address (Topic or Routing Key)

Unidirectional

One-to-One or One-to-Many

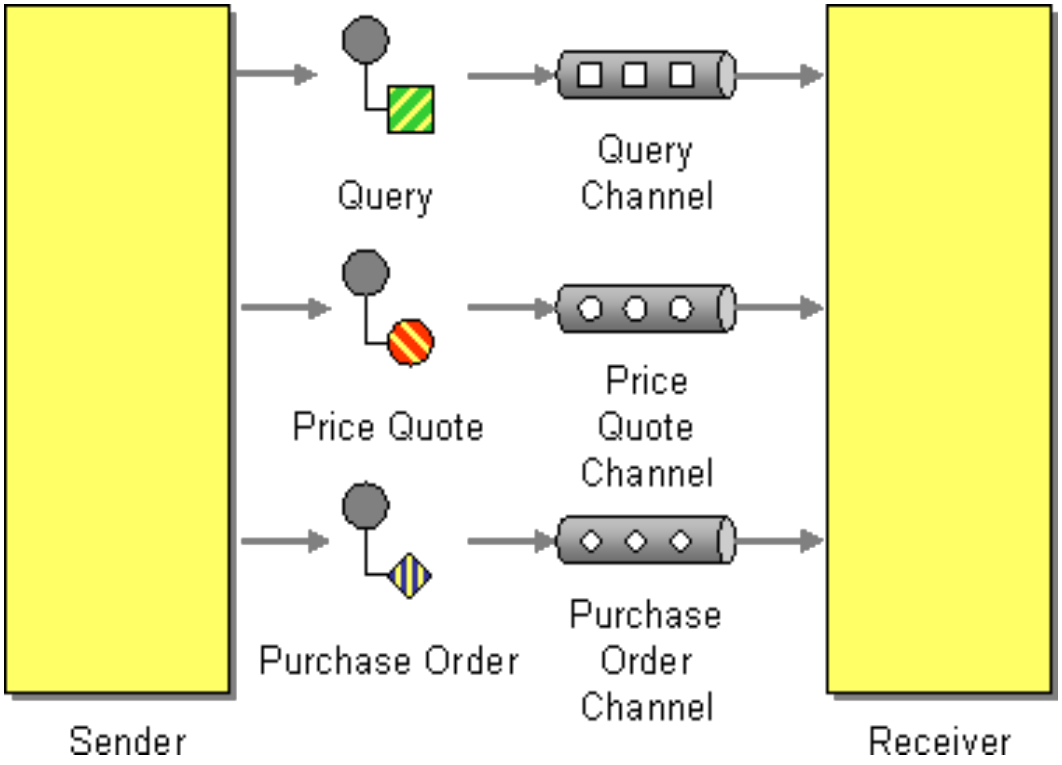
Messaging is a ‘pipe’ not a ‘bucket’.

Point-to-Point Channel



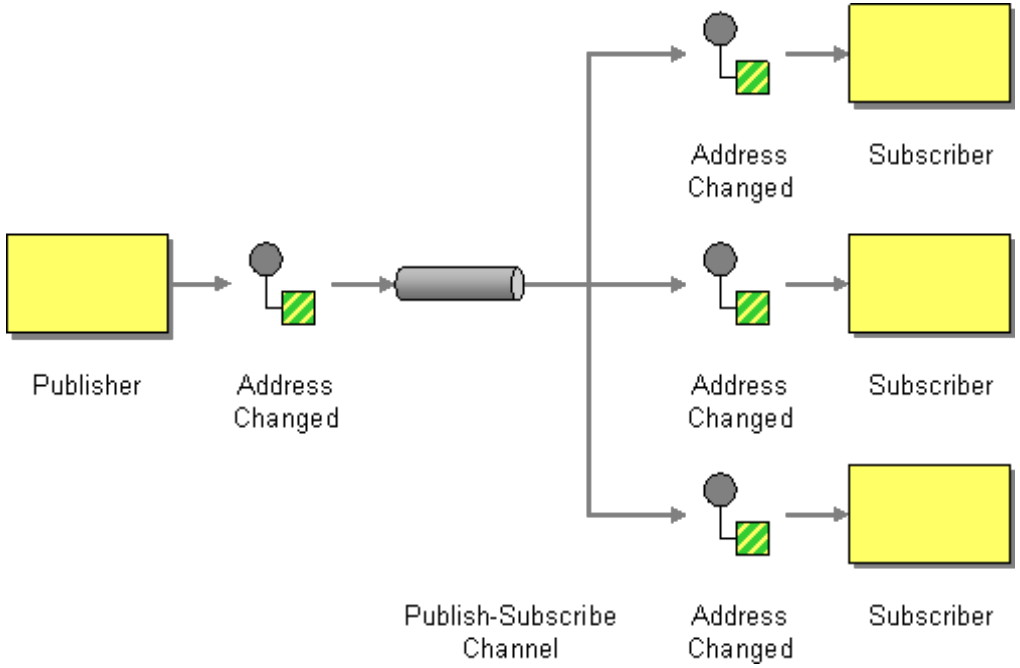
http://www.enterpriseintegrationpatterns.com/patterns/messaging/PointToPointChannel.html

Datatype Channel



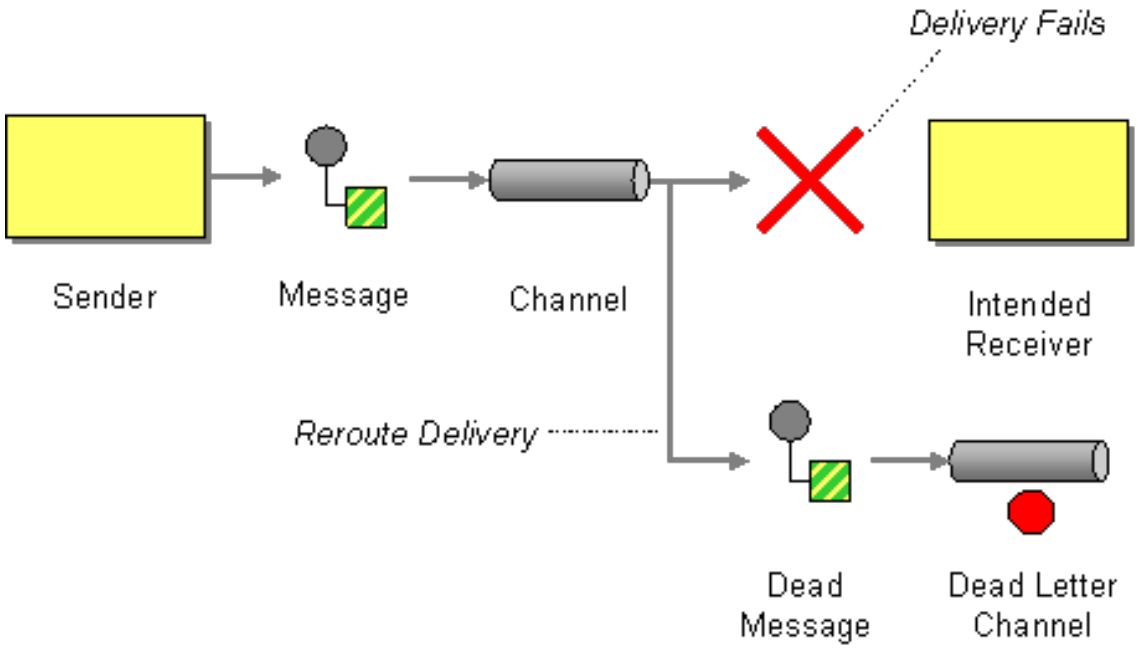
http://www.enterpriseintegrationpatterns.com/patterns/messaging/DatatypeChannel.html

Publish-Subscribe Channel



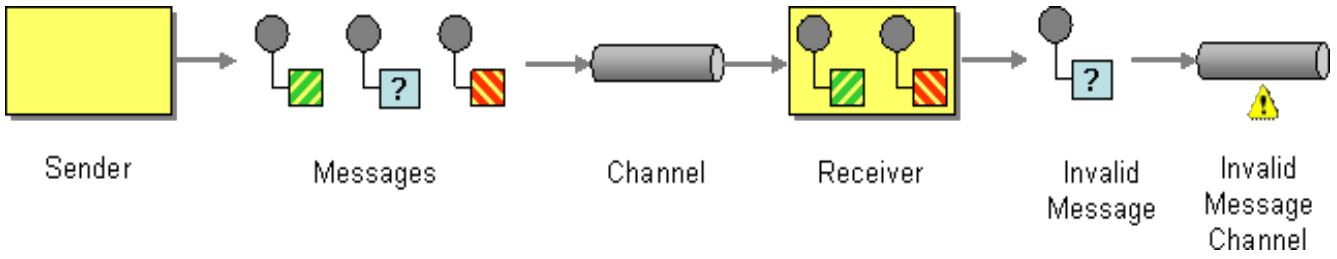
http://www.enterpriseintegrationpatterns.com/patterns/messaging/PublishSubscribeChannel.html

Dead Letter Channel



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

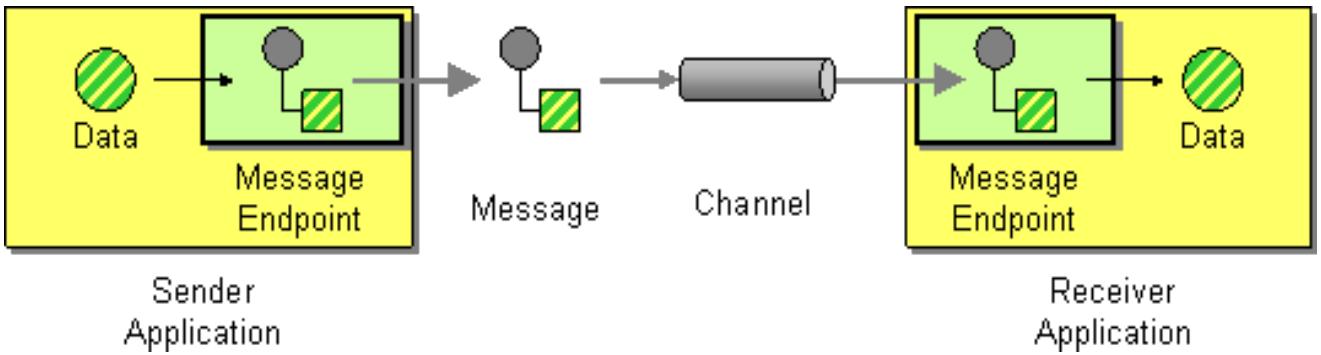
Invalid Message Channel



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

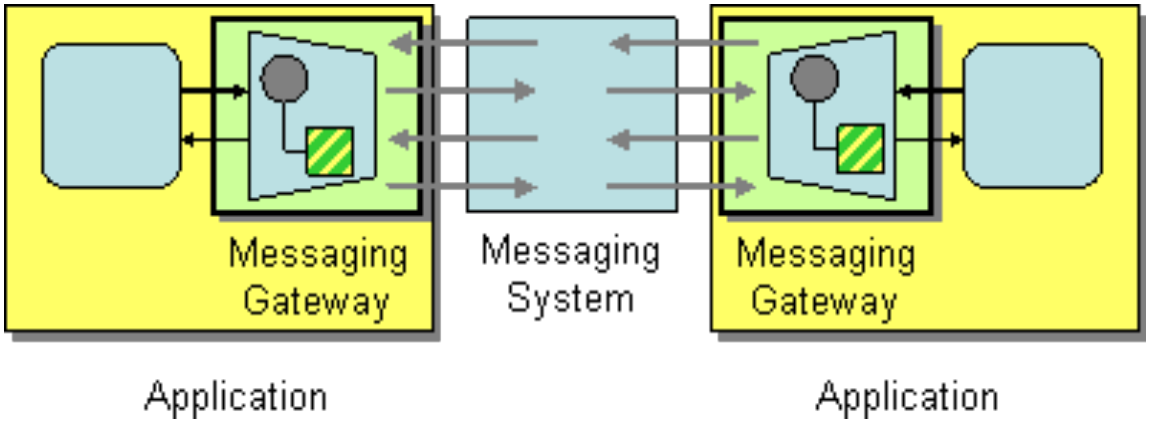
5.4 ENDPOINTS

Message Endpoint



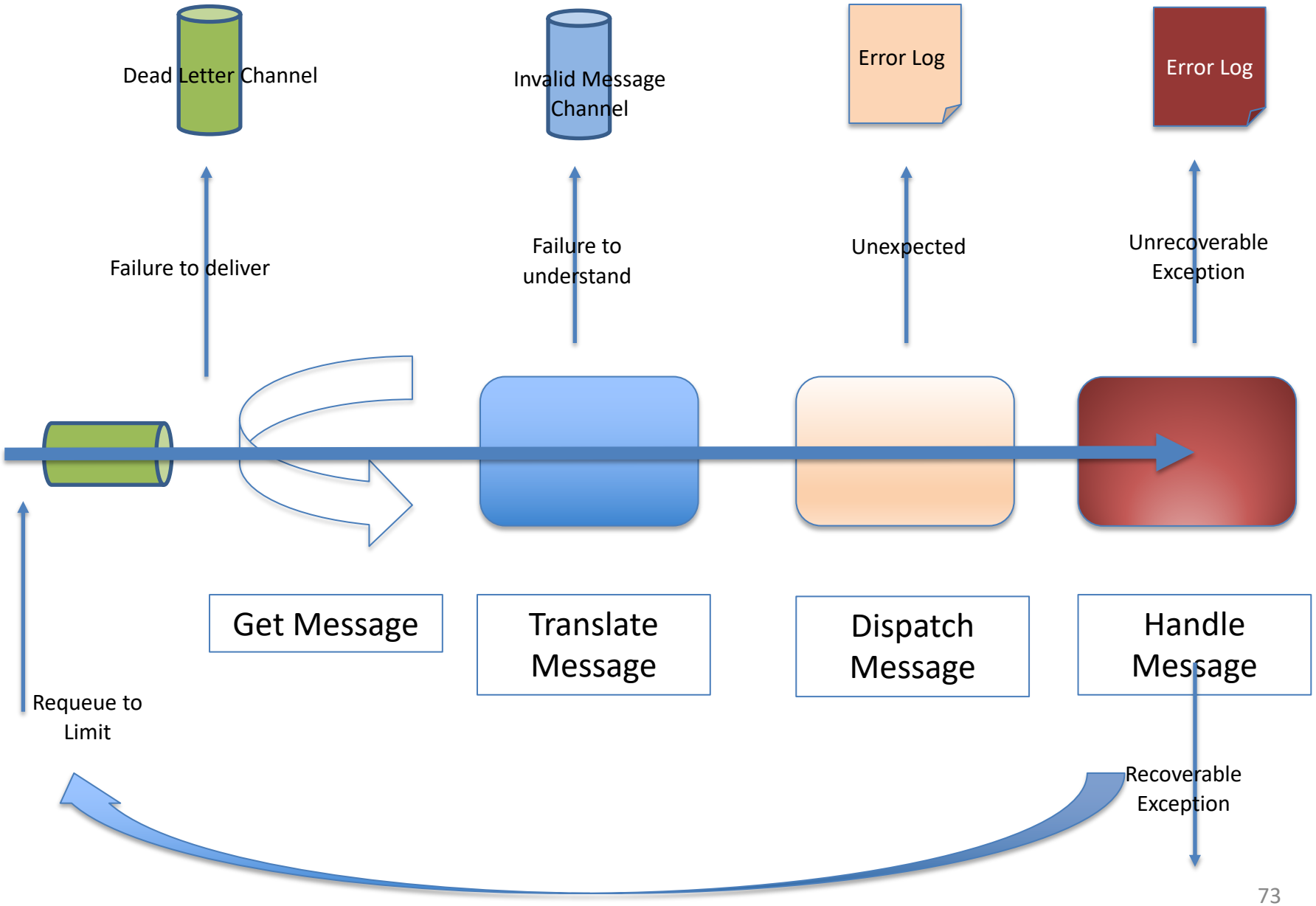
http://www.enterpriseintegrationpatterns.com/patterns/messaging/MessageEndpoint.html

Messaging Gateway

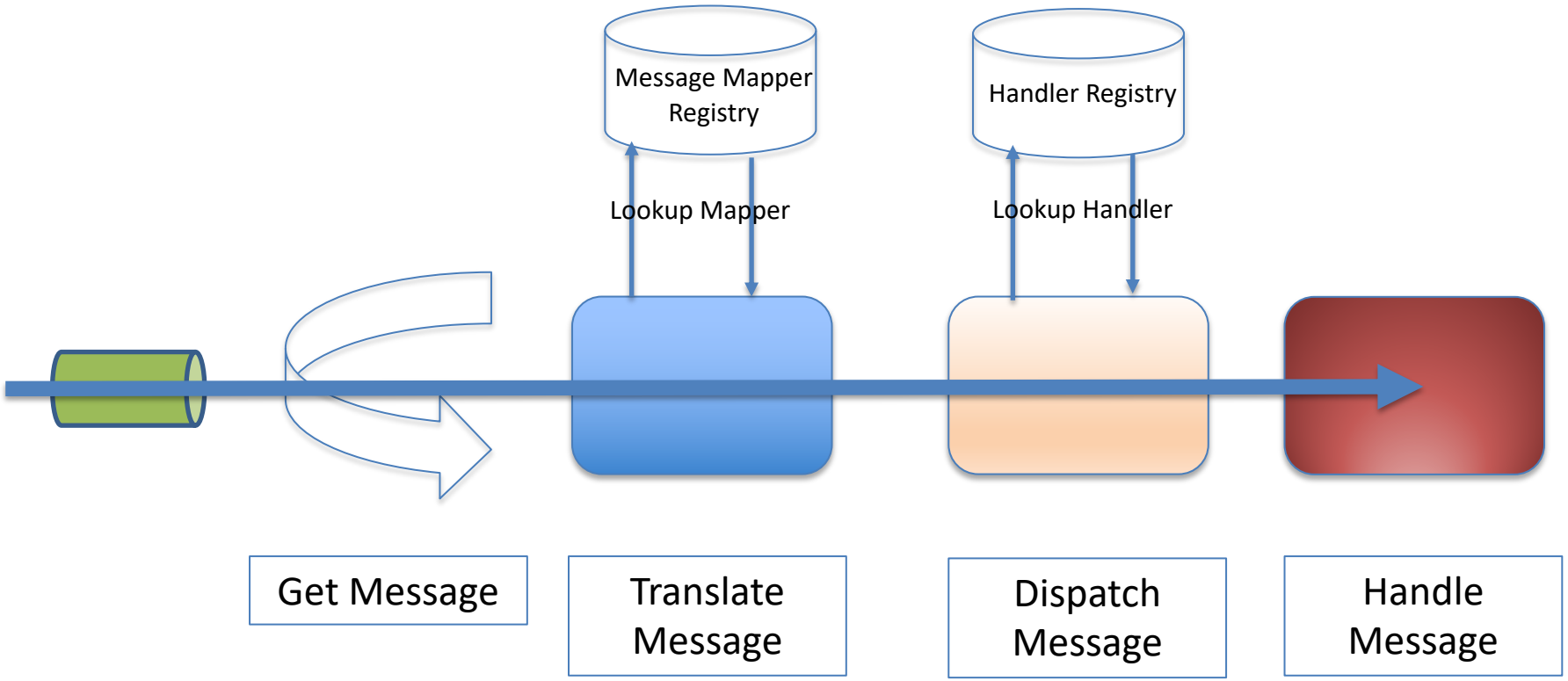


http://www.enterpriseintegrationpatterns.com/patterns/messaging/MessagingGateway.html

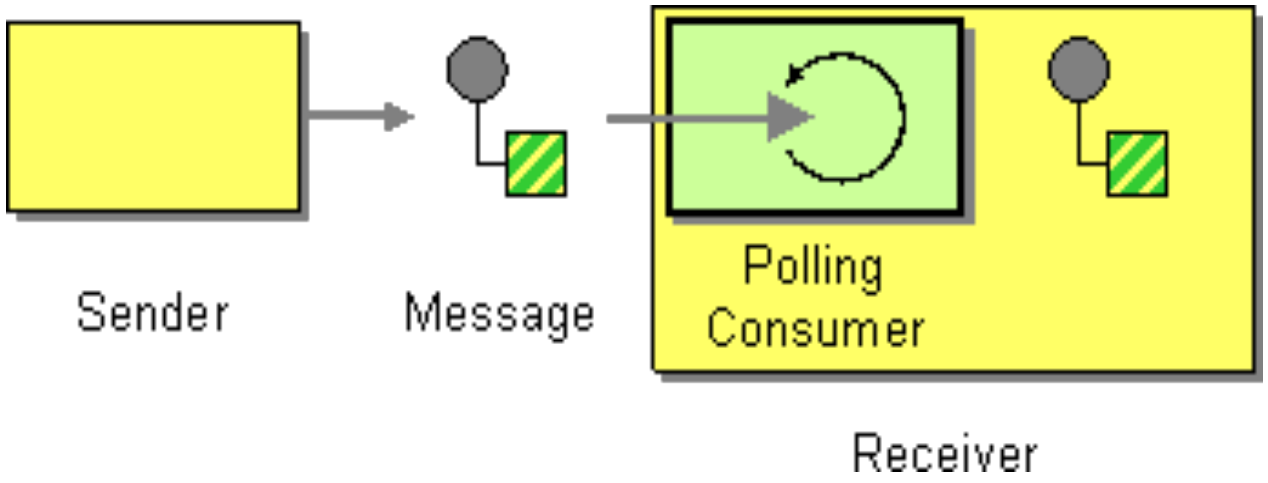
5.5 THE MESSAGE PUMP


Dead Letter Channel
Invalid Message Channel
Error Log
Error Log
Failure to deliver
Failure to understand
Unexpected
Unrecoverable Exception
Get Message
Translate Message
Dispatch Message
Handle Message
Requeue to Limit
Recoverable Exception

Translate and Dispatch

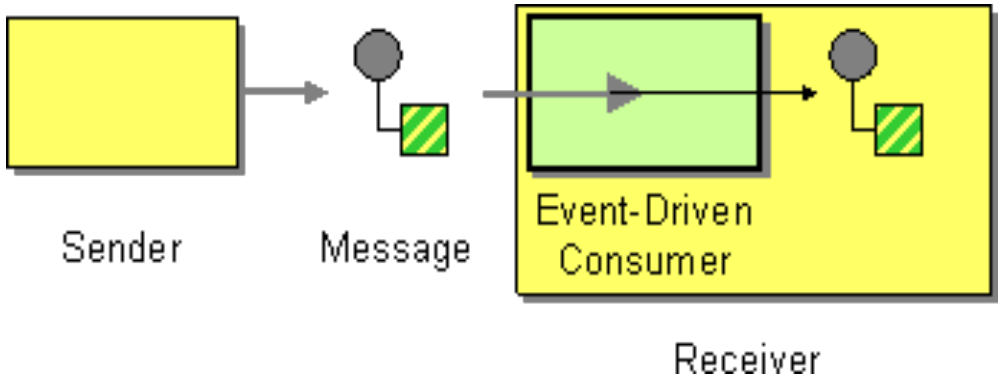


Polling Consumer



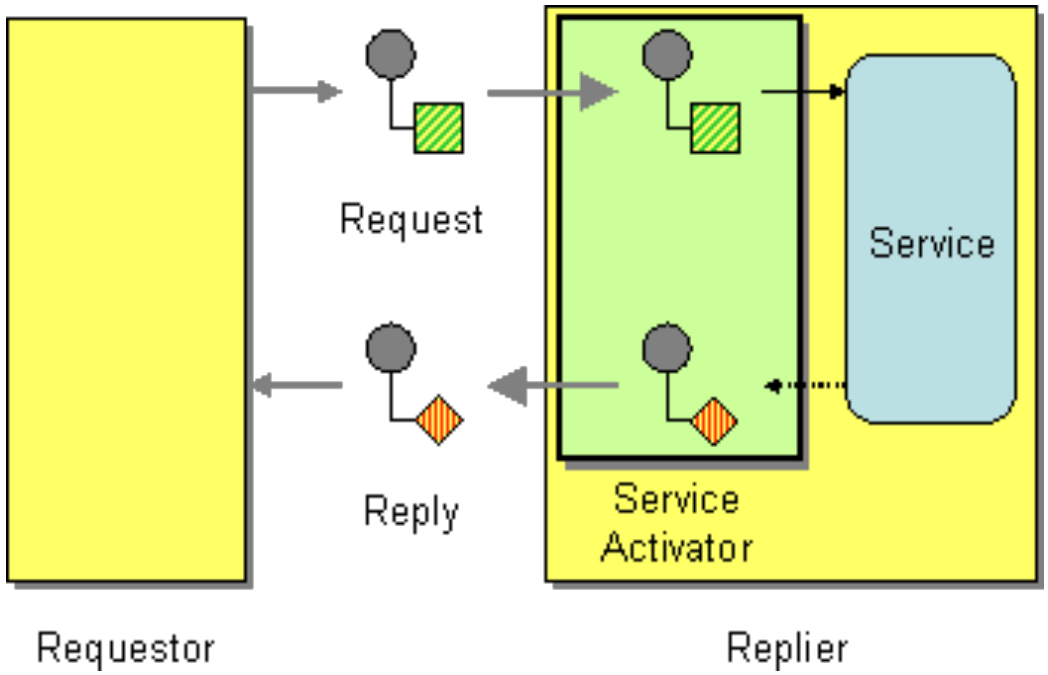
http://www.enterpriseintegrationpatterns.com/patterns/messaging/PollingConsumer.html

Event Driven Consumer



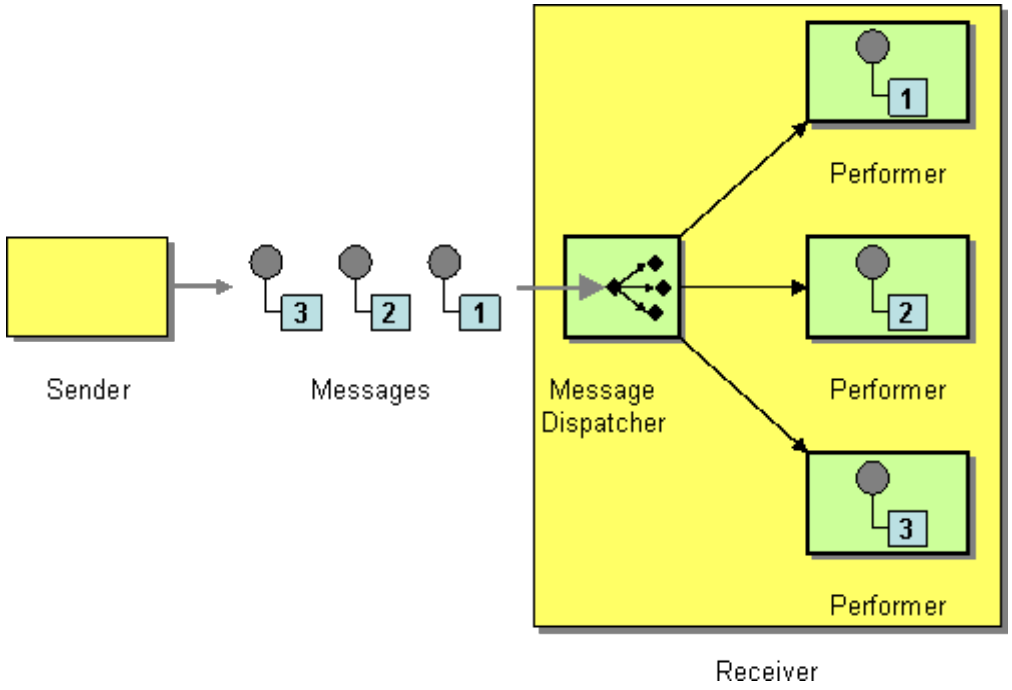
http://www.enterpriseintegrationpatterns.com/patterns/messaging/EventDrivenConsumer.html

Service Activator



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

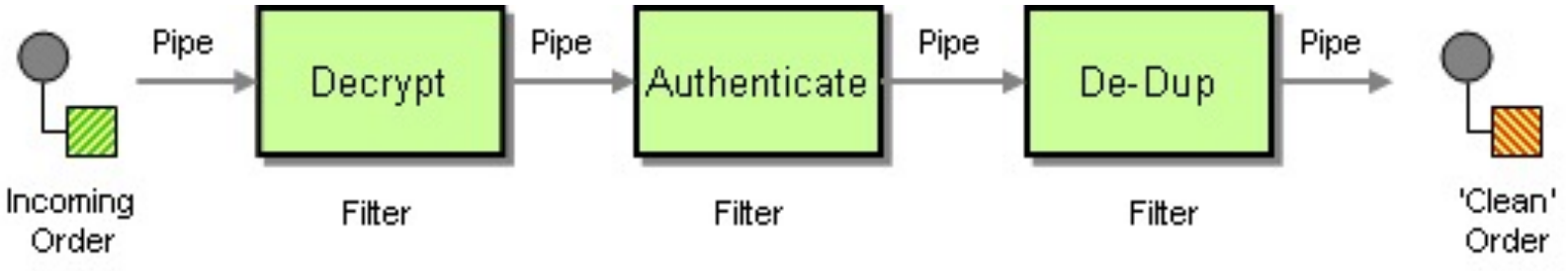
Competing Consumers



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

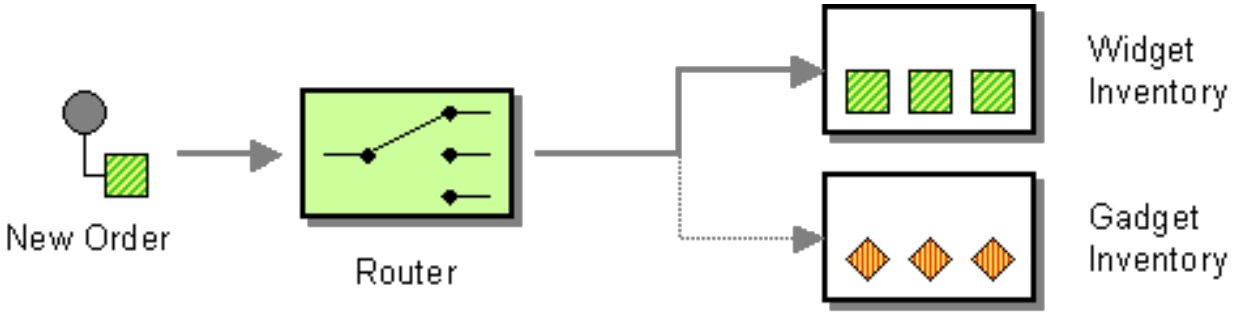
5.6 PIPELINES

Pipes and Filters



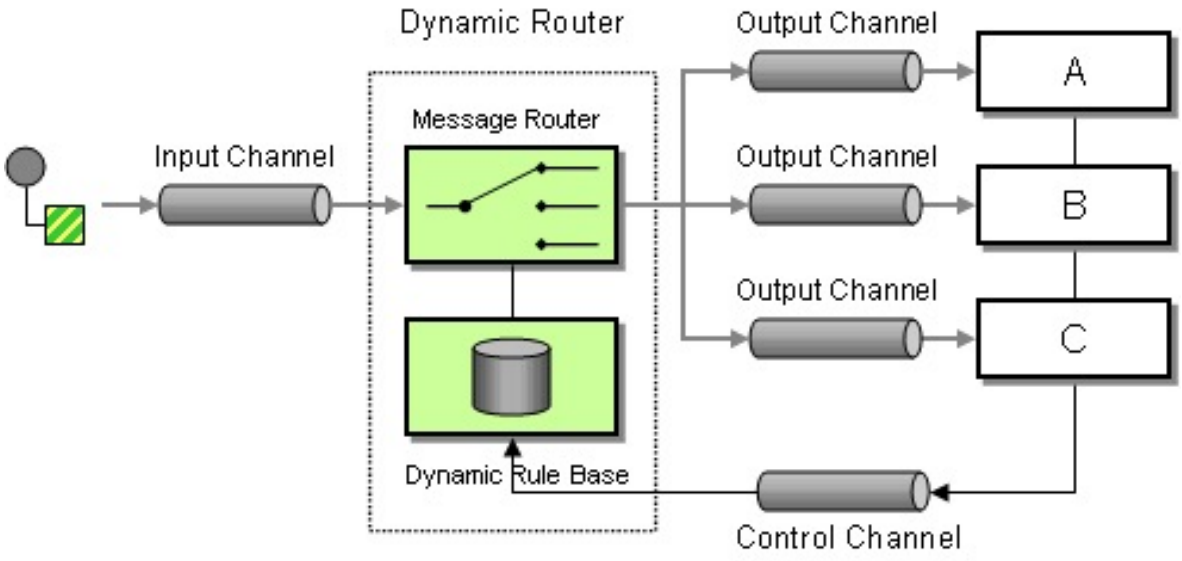
http://www.enterpriseintegrationpatterns.com/patterns/messaging/PipesAndFilters.html

Content Based Router



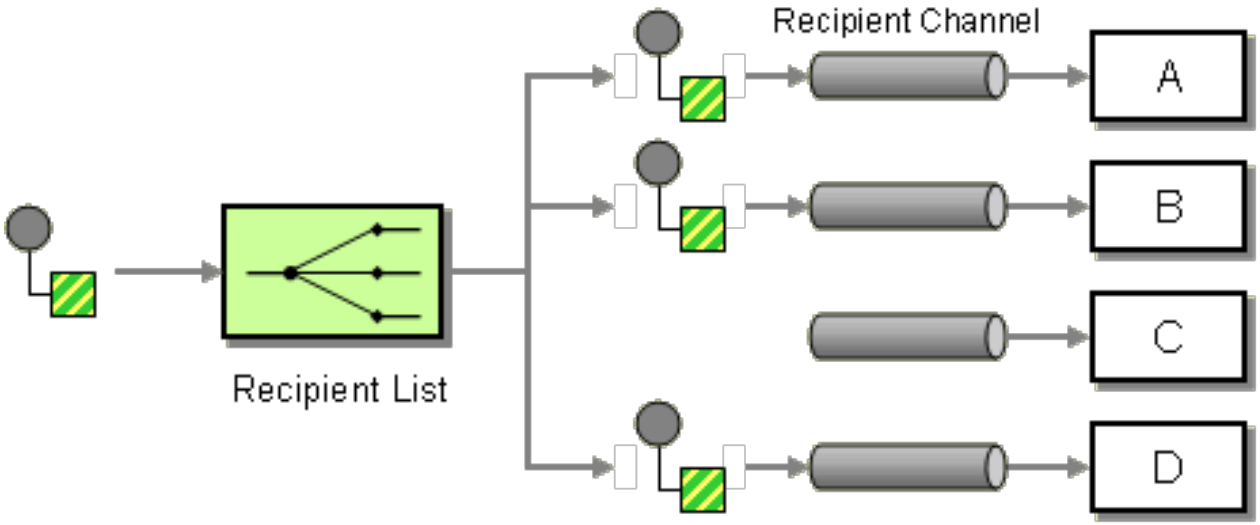
http://www.enterpriseintegrationpatterns.com/patterns/messaging/ContentBasedRouter.html

Dynamic Router



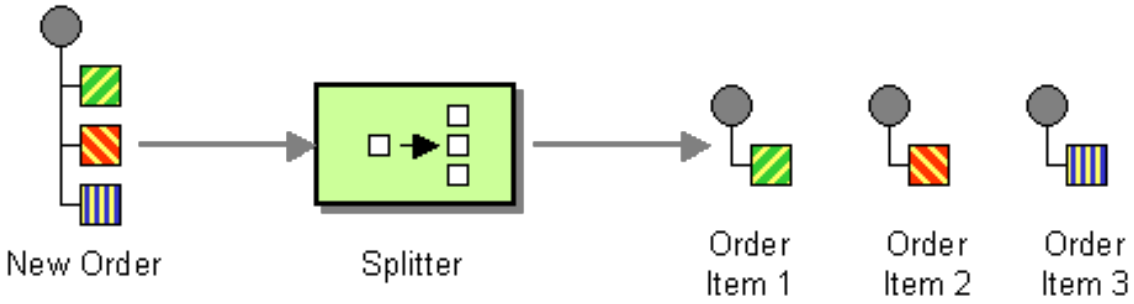
http://www.enterpriseintegrationpatterns.com/patterns/messaging/DynamicRouter.html

Recipient List



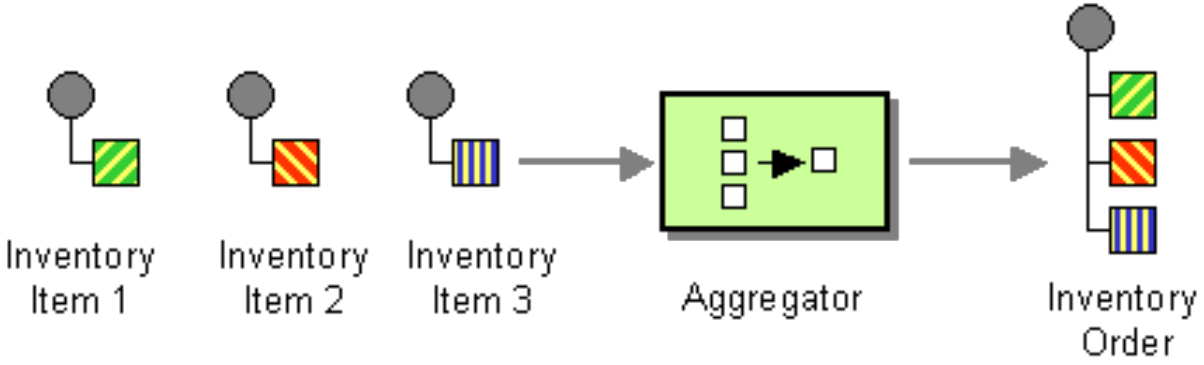
http://www.enterpriseintegrationpatterns.com/patterns/messaging/RecipientList.html

Splitter



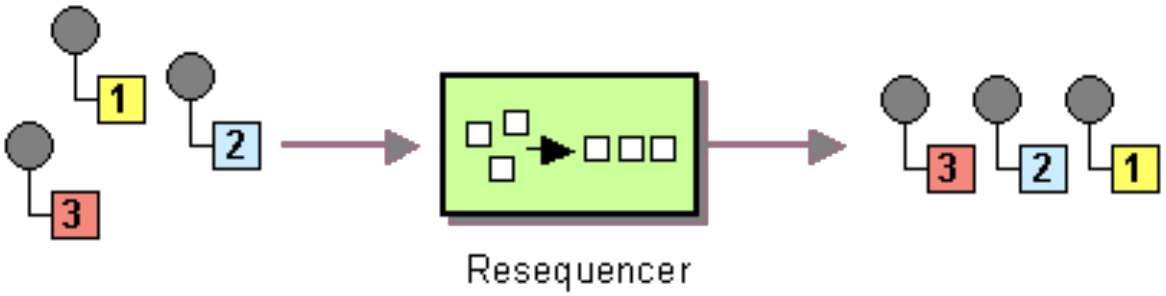
http://www.enterpriseintegrationpatterns.com/patterns/messaging/Sequencer.html

Aggregator



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

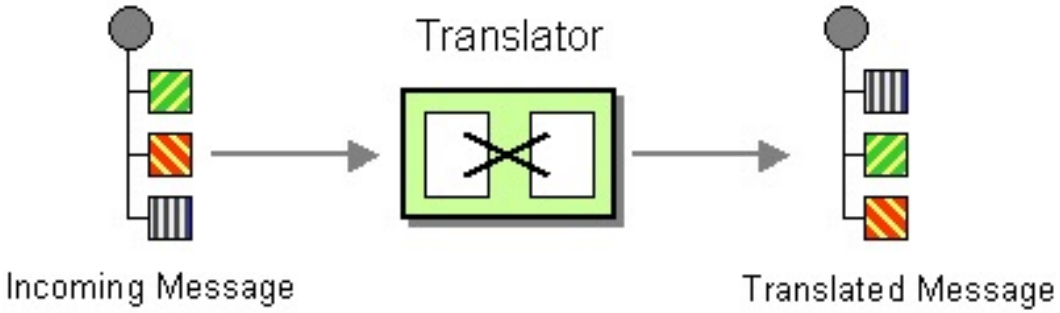
Resequencer



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

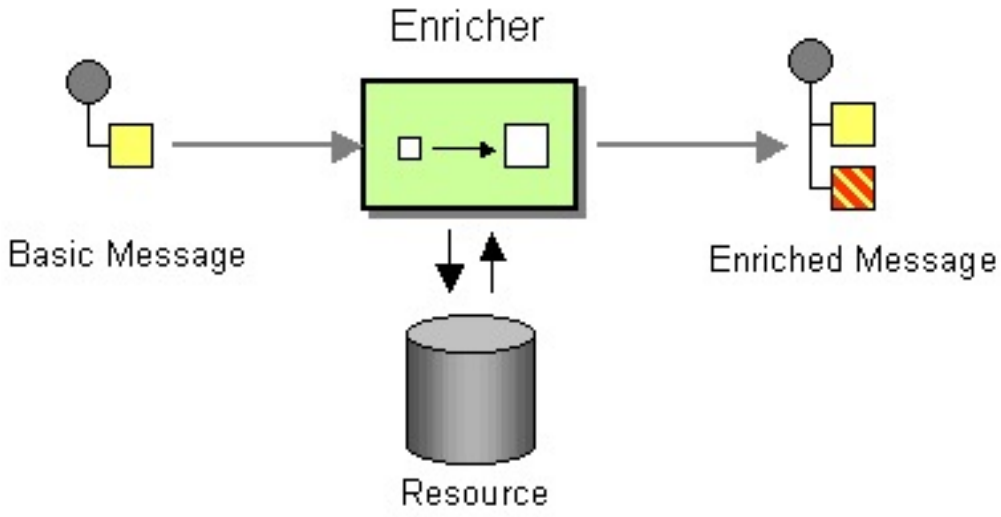
5.7 TRANSFORMATION

Message Translator



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

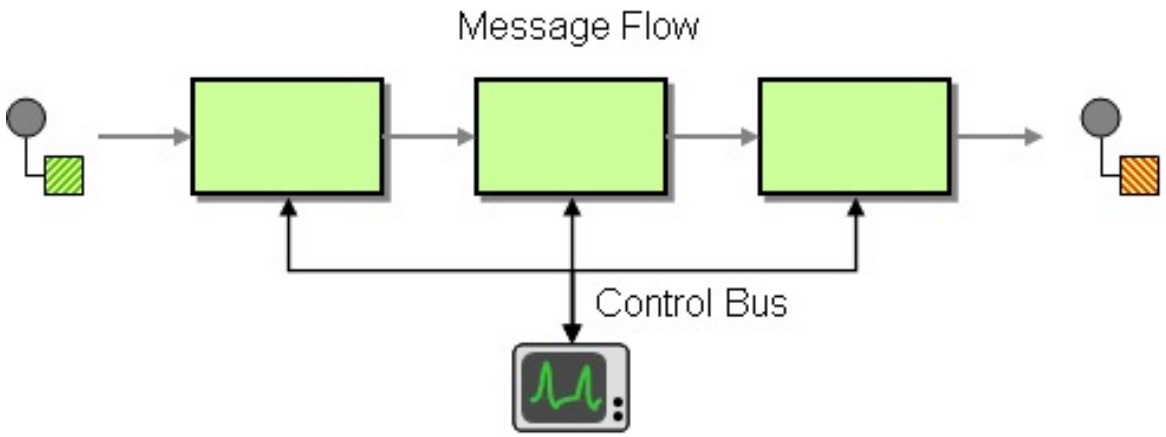
Content Enricher



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

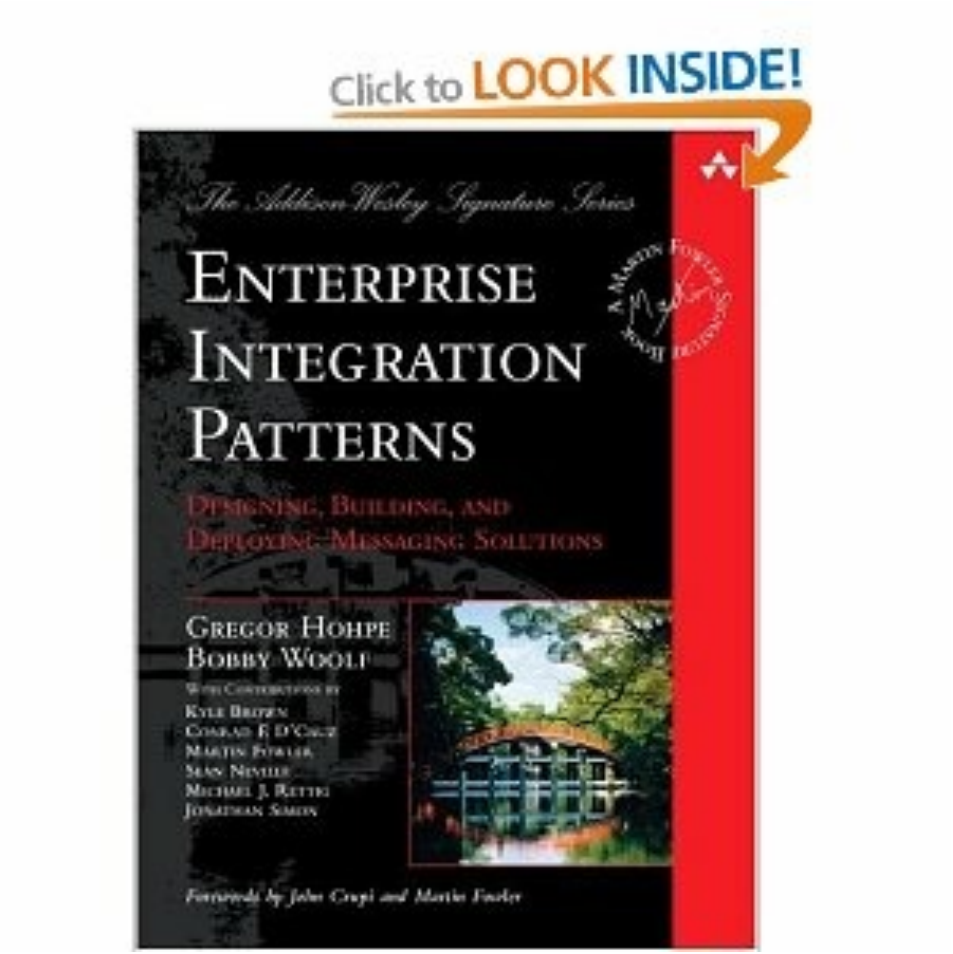
5.8 MANAGEMENT

Control Bus



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

Further Reading



Q&A