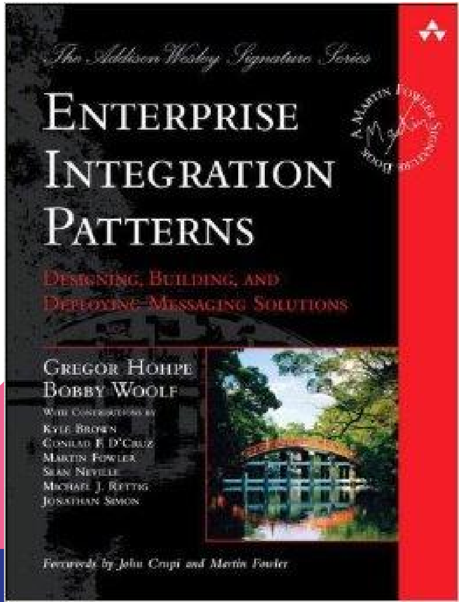
Enterprise Integration Patterns
With RabbitMQ & MicroServices

Franco Poveda

What is Messaging?

- Messaging enables high-speed, asynchronous, program-to-program communication.
- Programs communicate by sending messages to each other
- A producer is a program that sends a message by writing the message to a channel.
- A consumer is a program that receives a message by reading it from a channel.


The Addison Wesley Signature Series
Enterprise Integration Patterns
Designing, Building, and Deploying Messaging Solutions
A Martin Fowler Signature Book
Gregor Hohpe
Bobby Woolf
With Contributions by
Kyle Brown
Conrad F. D'Cruz
Martin Fowler
Sean Neville
Michael J. Rettig
Jonathan Simon
Forewords by John Crupi and Martin Fowler

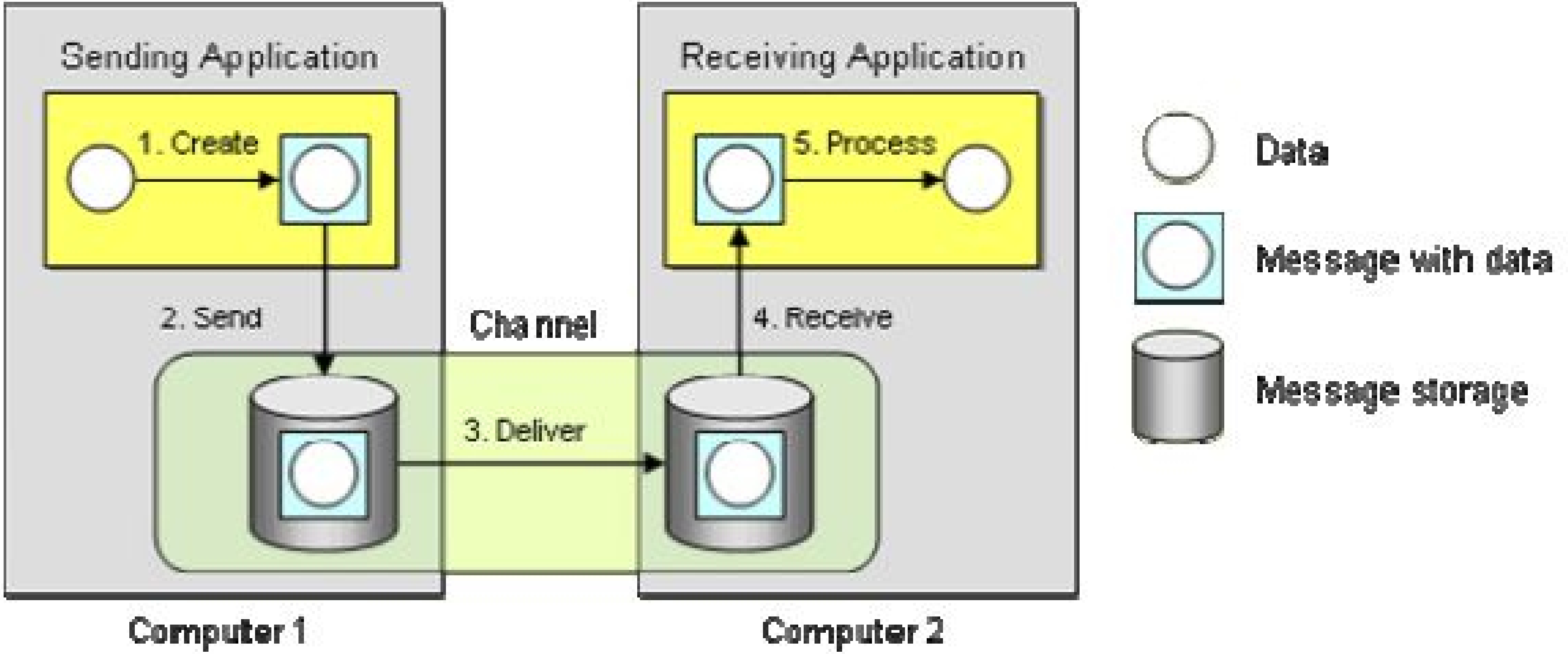
What is a Messaging System?

- Messaging capabilities are typically provided by a separate software (messaging system).

- The main task of a messaging system is to move messages from the sender’s computer to the receiver’s computer in a reliable fashion.

- The reason a messaging system is needed to move messages from one computer to another is that computers and the networks that connect them are inherently unreliable

Message Transmission



Demo

1. **RabbitMQ Simulator**: http://tryrabbitmq.com/
2. NodeJS example

Why Messaging Systems?

- Remote Communication
- Platform/Language Integration
- Asynchronous Communication. Messaging enables a send and forget approach.
- Throttling. Too many calls on a single receiver at the same time can overload the receiver.
- Reliable Communication.

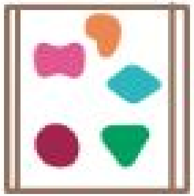
Microservices Architecture Patterns

- Particular way of designing software applications as independently deployable services.
- Each service running in its own process and communicating with lightweight mechanisms.
- Becoming the default style for building enterprise applications.
- Decentralized control of languages and data.

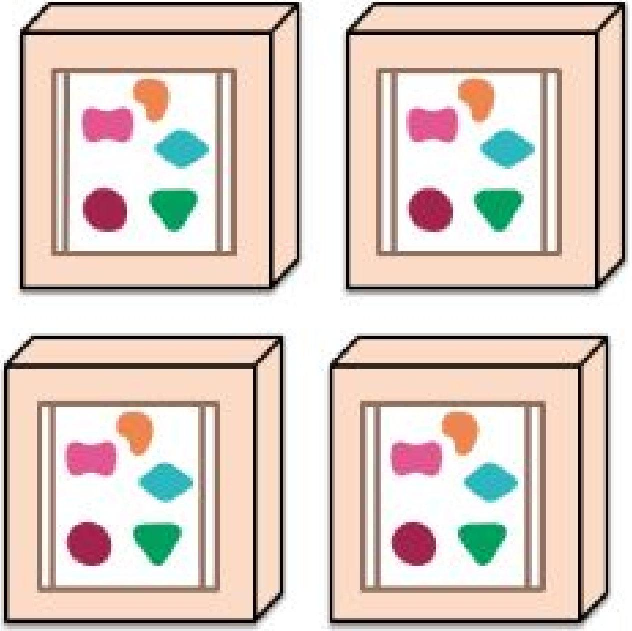
Monolithic VS Microservices

- A monolithic application is built as a single unit
- Any changes to the system involve building and deploying entire app.
- Hard to keep a good modular structure.
- Scaling requires scaling of the entire application rather than parts of it that require greater resource.

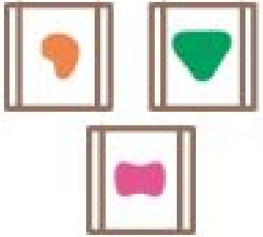
A monolithic application puts all its functionality into a single process...



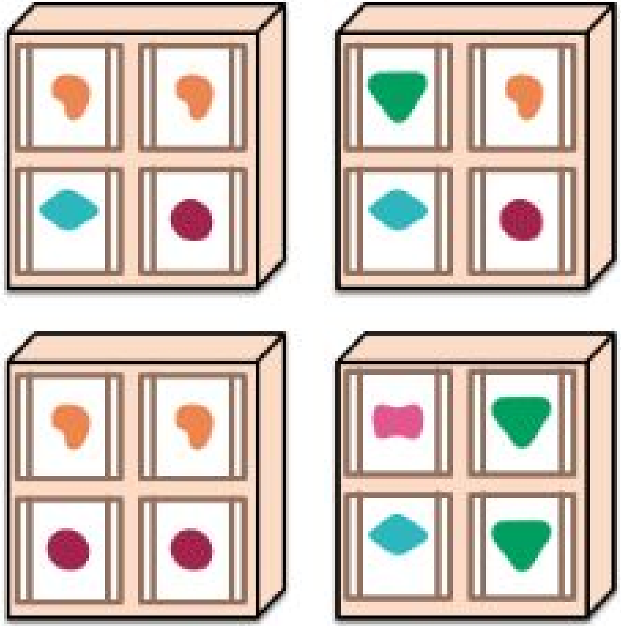
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



Pros

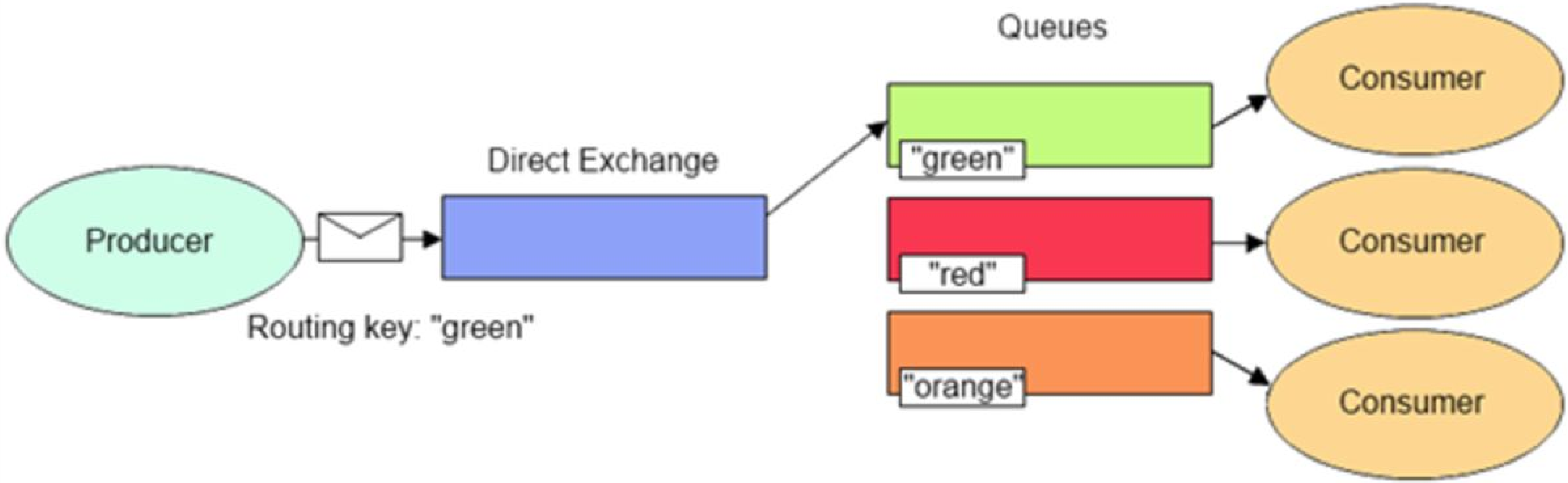
- **Deployability**: more agility to roll out new versions of a service due to shorter build+test+deploy cycles.
- **Reliability**: a microservice fault affects that microservice alone and its consumers.
- **Availability**: rolling out a new version of a microservice requires little downtime.
- **Scalability**: each microservice can be scaled independently using pools, clusters, grids.
- **Modifiability**: more flexibility to use new frameworks, libraries, datasources.
- **Management**: the application *development* effort is divided across teams.
- **Design autonomy**: the team has freedom to employ different technologies.

Cons

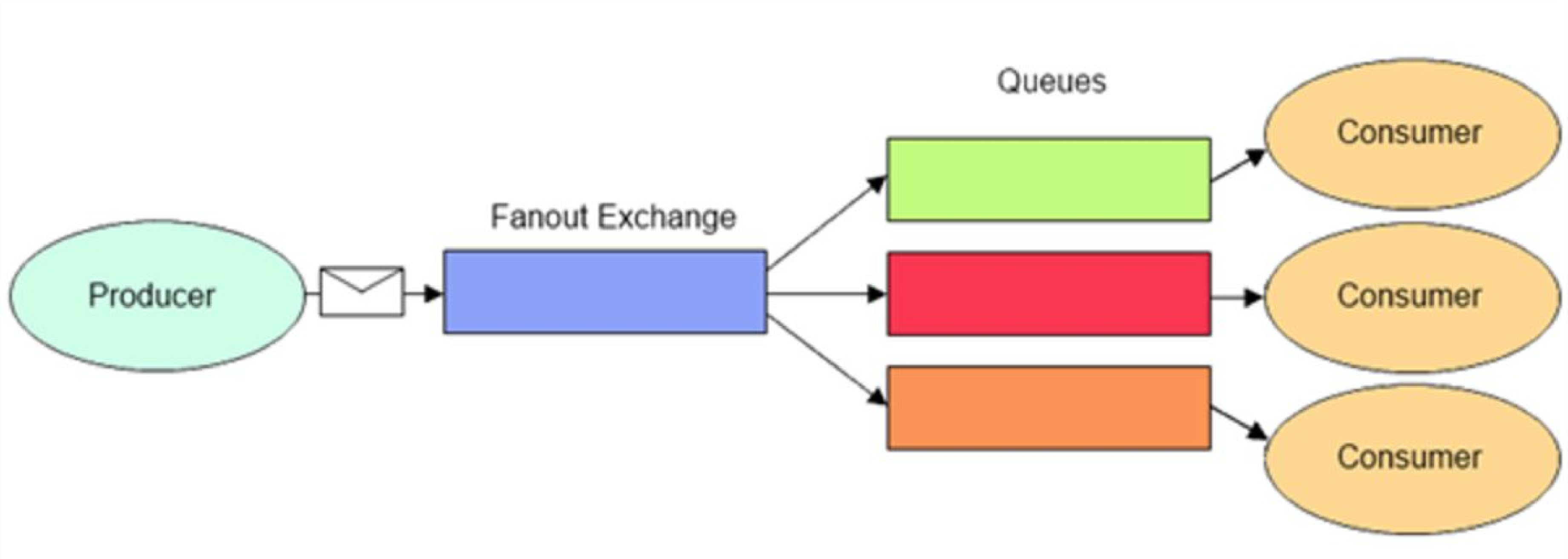
- **Deployability**: deployment may become more complex with many jobs, scripts and config files.
- **Performance**: services more likely need to communicate over the network, whereas services within the monolith may benefit from local calls. Performance overhead.
- **Modifiability**: Also, mechanisms to improve autonomy, such as eventual consistency and asynchronous calls, add complexity to microservices.
- **Testability**: automated tests are harder to setup and run because they may span different microservices on different runtime environments.
- **Management**: the application operation effort increases because there are more runtime components, log files, and point-to-point interactions to oversee.

Example: image upload and processing

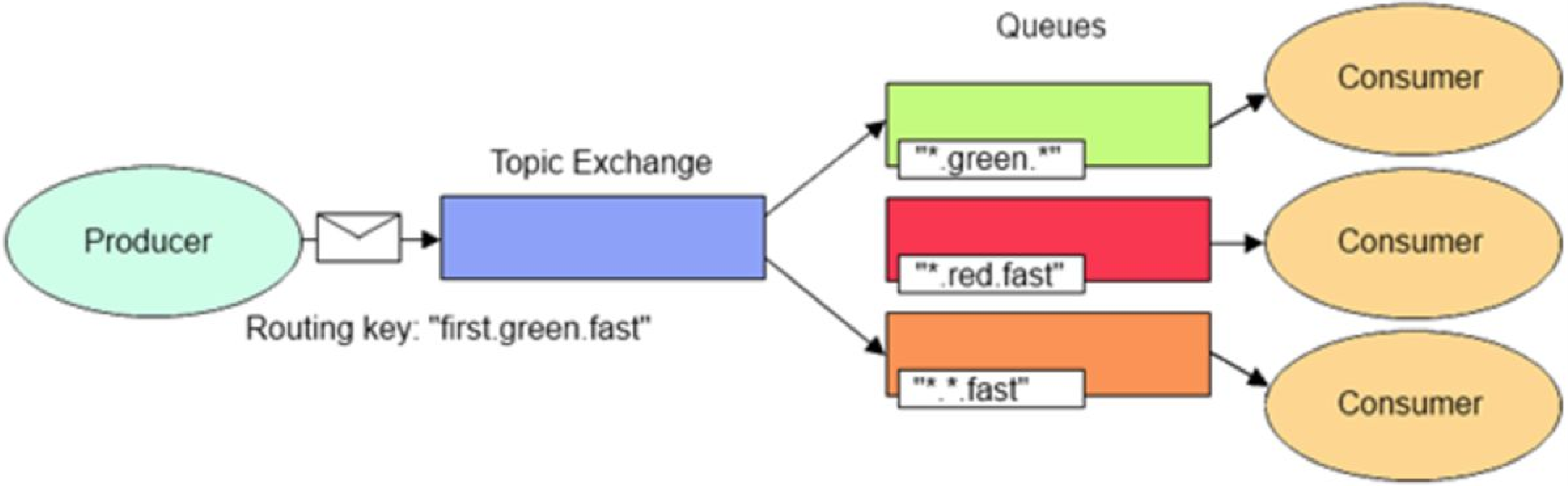
Exchange Types: Direct



Exchange Types: Fanout



Exchange Types: Topic



New request: video upload & processing

- The microservices approach facilitates the introduction of new features and reduces the chances of compromising the rest of the system.

- Complexity is also reduced by using different types of **exchanges** to handle routing.

- Applications should be designed with a async webhook/callback in mind.