Comunicación en Micro-Servicios

Fernando Sonego
Solution Architect en Algeiba

Latino .Net Online
sábado, 22 de agosto de 2020

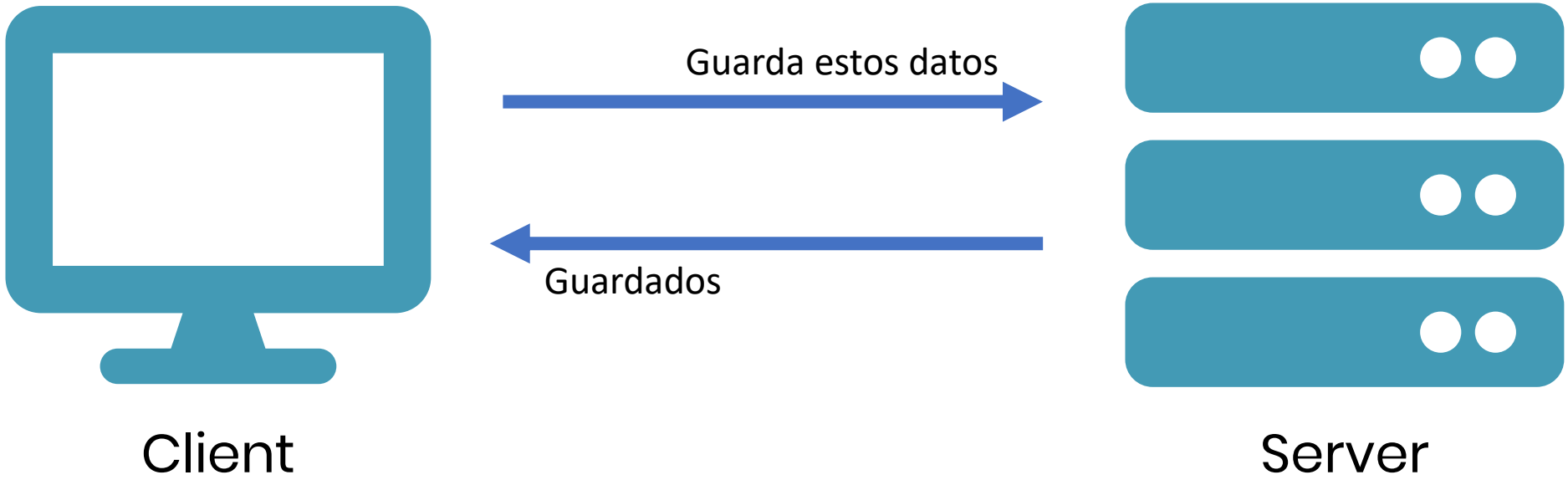
Agenda

- Comunicación Sincrónica
  - Introducción a RPC
  - Stateless y Stateful RPCs
  - Rendimiento de comunicaciones asincrónicas
- Comunicación Asincrónica
  - Introducción
  - Características de diseño
  - Flujo de la comunicación
  - Transacciones distribuidas
  - Patrones: Saga y Routing Slip

Sincrónica

Arquitectura de Comunicación

Introducción a RPC



RPCs

Ejemplos

- Trafico Http
  - “web server envíame un html”
- Consultas SQL
  - “Dame todos los clientes de la compañía”
- Servicios
  - Crear un nuevo cliente con el nombre Lautaro

Propiedades

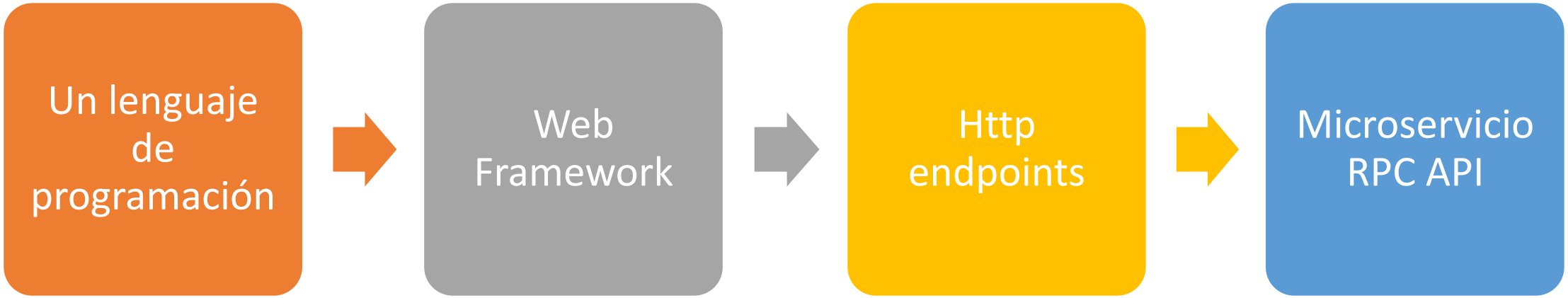
- Argumentos
- Devuelve un valor
- Se ejecuta inmediatamente
- El cliente controla la comunicación
- 2 Externos involucrados
- El cliente espera la respuesta del server.

RPCs son sincronicos

- Cliente siempre espera al servidor
- No usar async / await in c# or Javascript
- El cliente necesita resultados desde el servidor

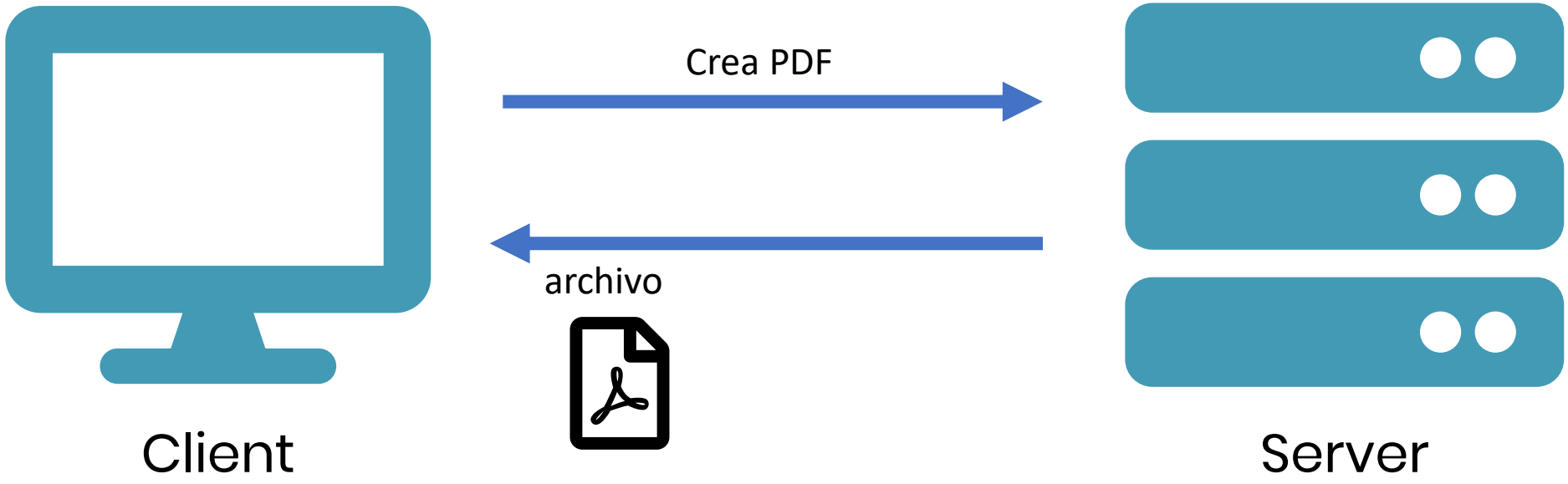
- Son poderosos en:
  - Simplicidad
  - Fáciles de pensar
  - Fácil de utilizar pruebas automáticas.

RPCs en microservicios

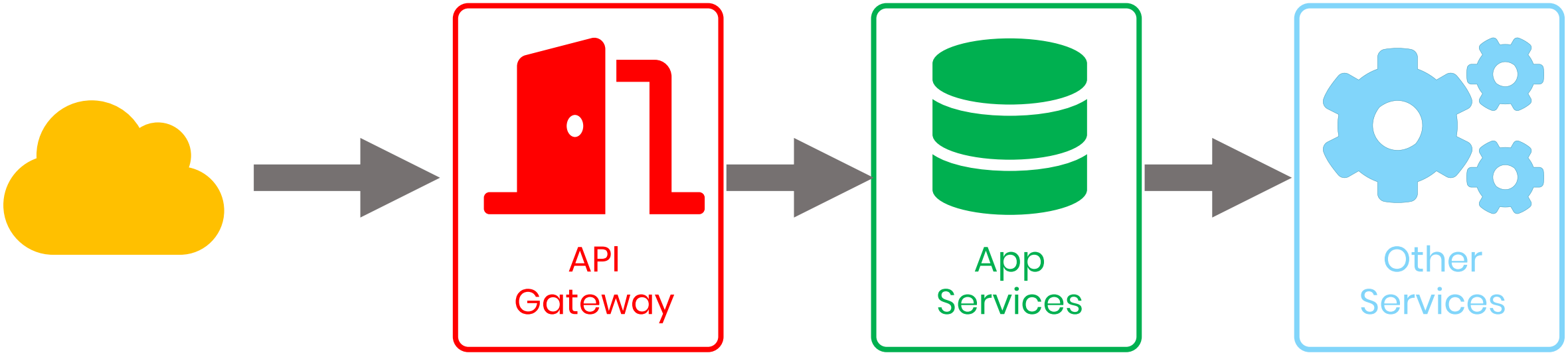


RPC: Stateless y Stateful

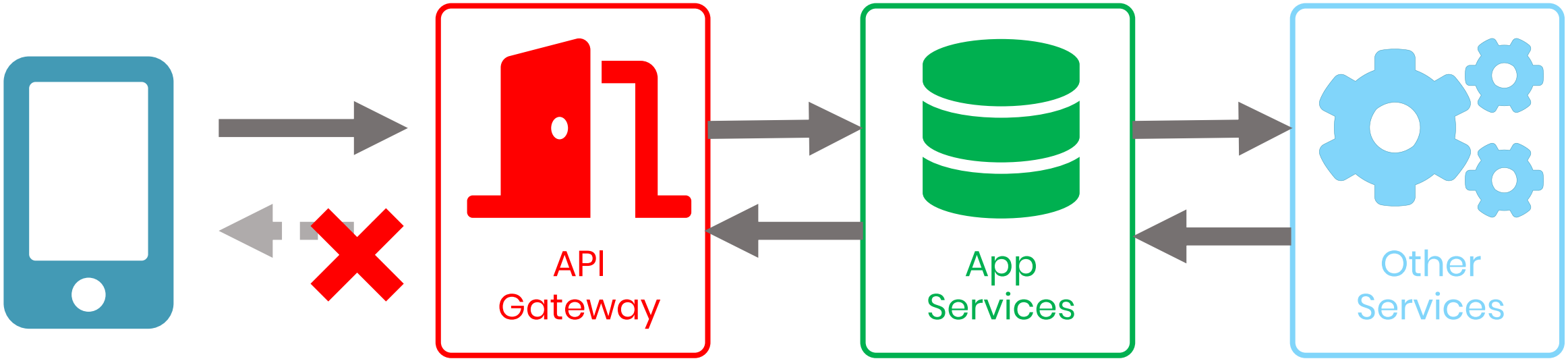
RPC: Stateless



API Gateway

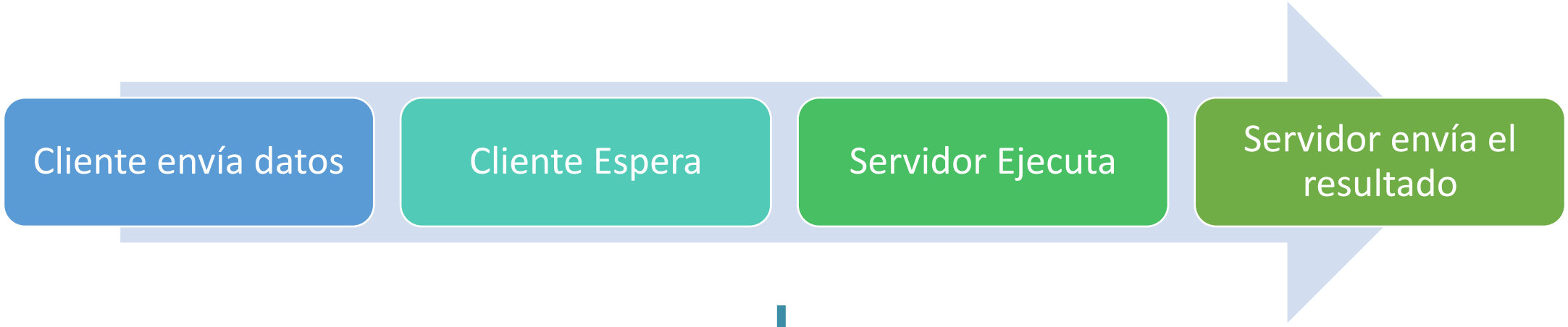


API Gateway



Coordinación de RPCs

Resumen: Remote Procedure Call



Stateless

Simple
Funcional
Portable

Stateful

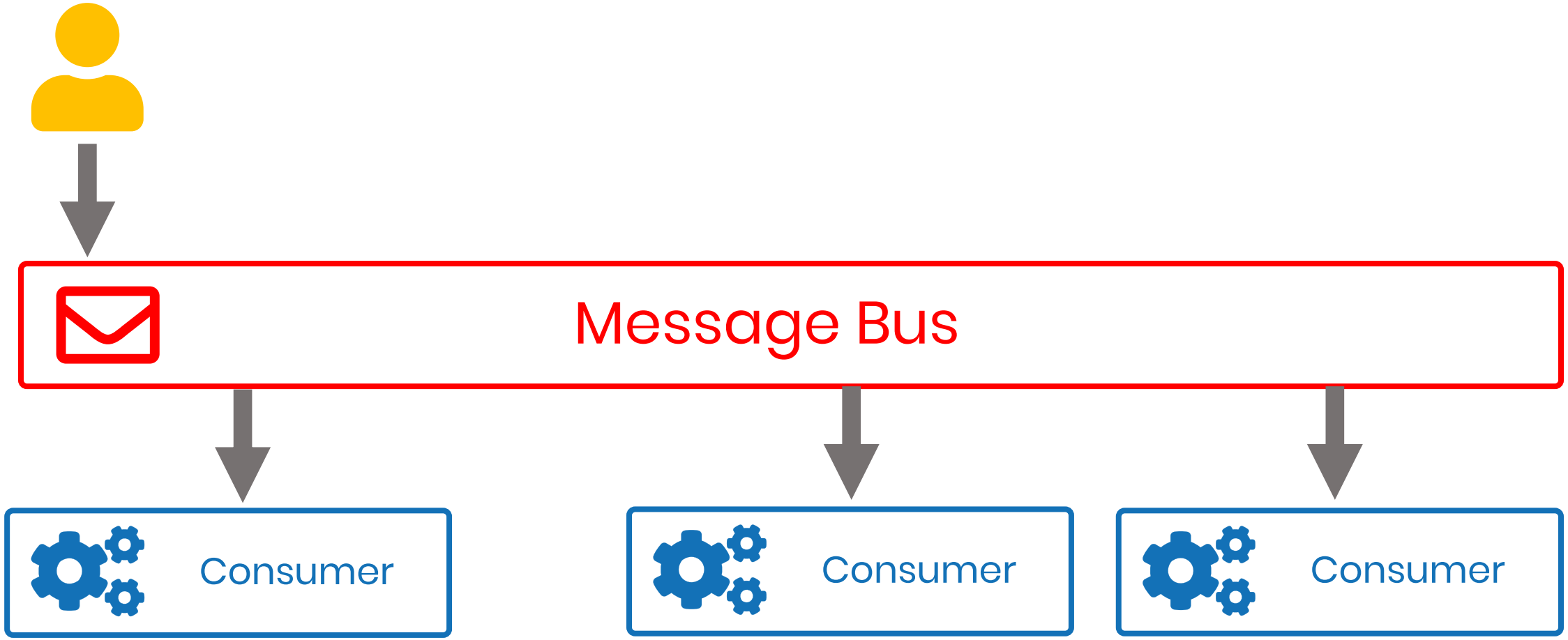
Idempotencia
Transacciones Distribuidas
Modos de falla

- Los servicios pueden tener manejo de fallas y baja latencia.
- Latencia y Disponibilidad difícil de coordinar, mejorar uno, dañar el otro

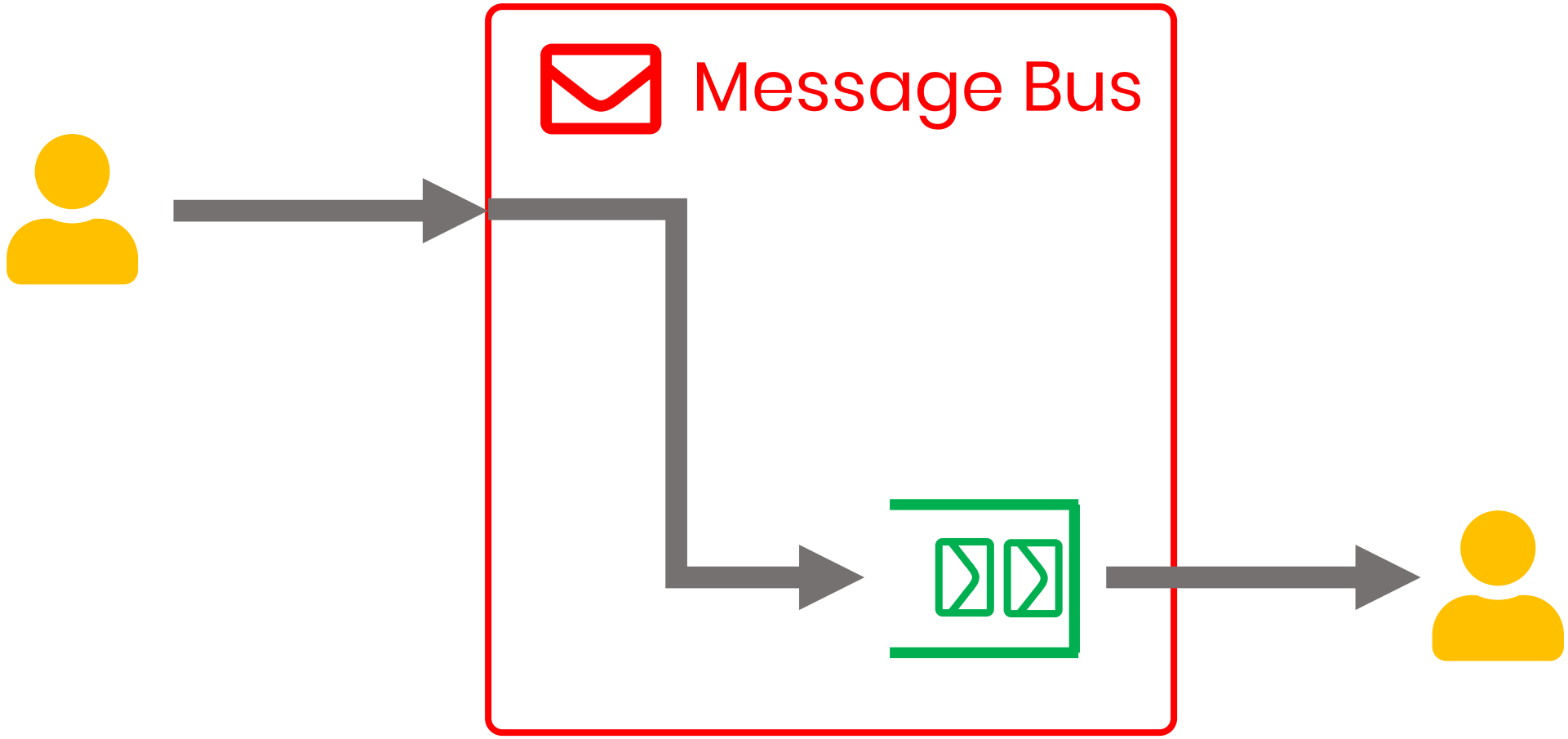
Asincrónico

Arquitectura de Comunicación

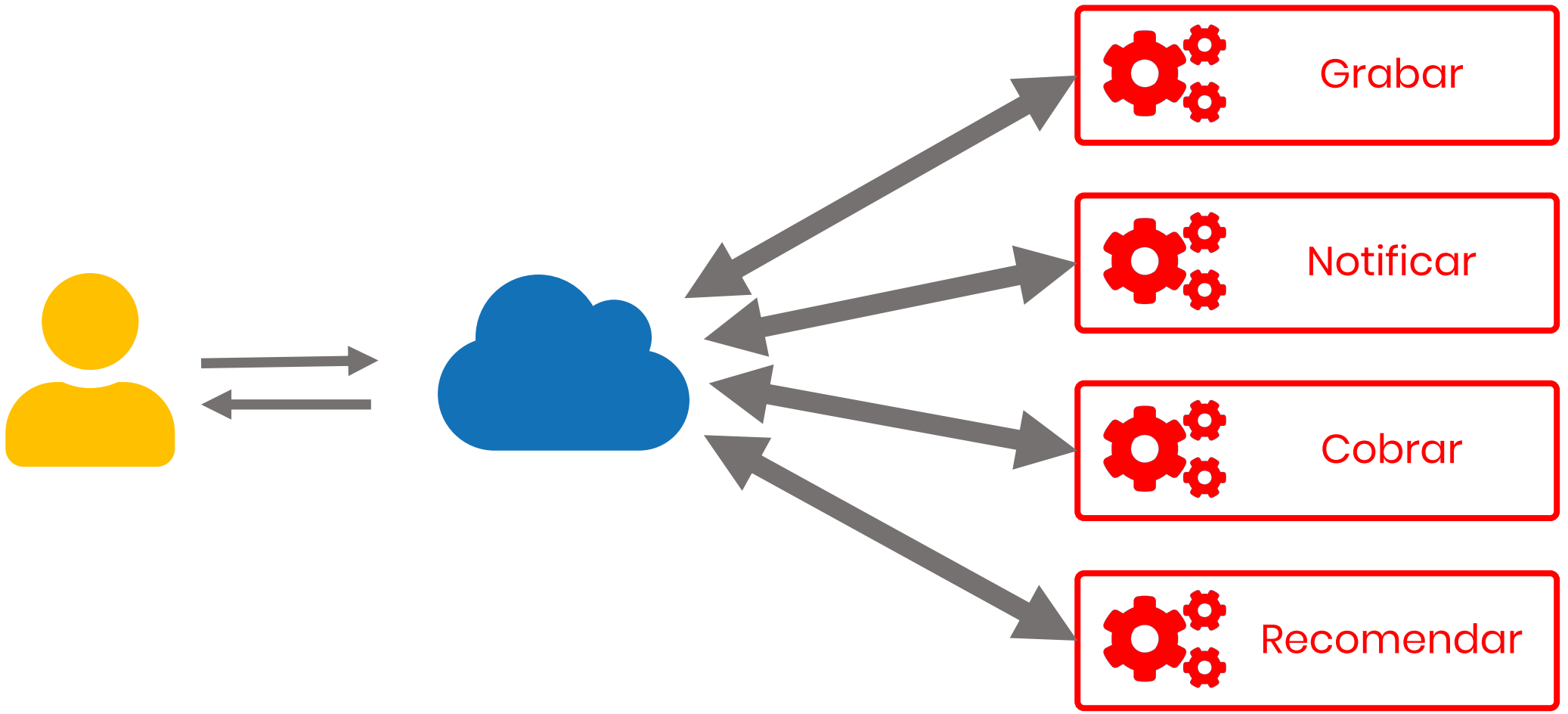
Múltiples consumidores



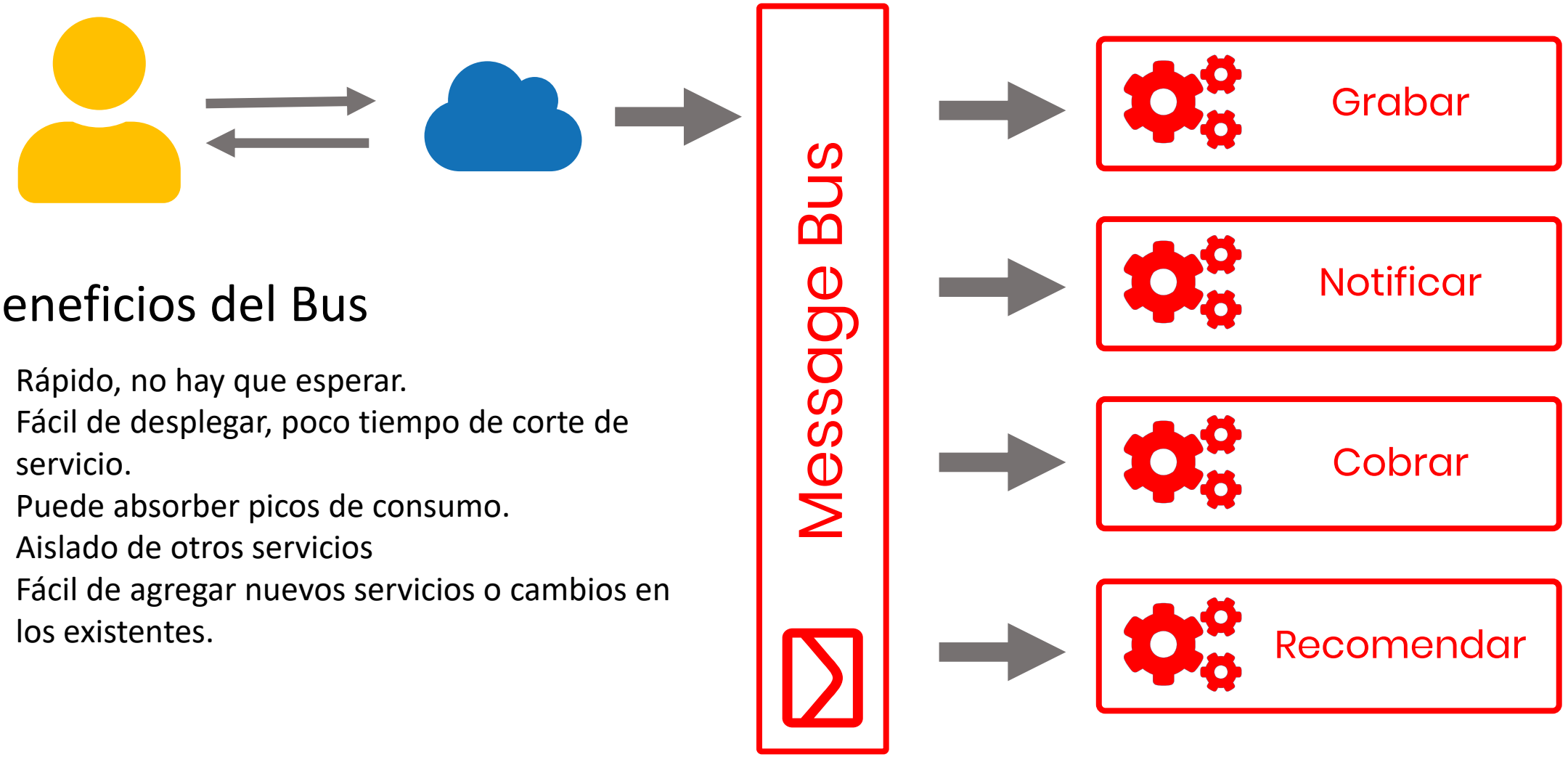
Cola de mensajería



Un post en RPC



Un post con un Bus



Beneficios del Bus

- Rápido, no hay que esperar.
- Fácil de desplegar, poco tiempo de corte de servicio.
- Puede absorber picos de consumo.
- Aislado de otros servicios
- Fácil de agregar nuevos servicios o cambios en los existentes.

Mensajes

Estructura

- Peña unidad de datos
- Solo información relevante del negocio
- Headers
  - Timestamp
  - Información del quien envía.
  - Id de Correlación
  - Información de errores
  - Header personalizados

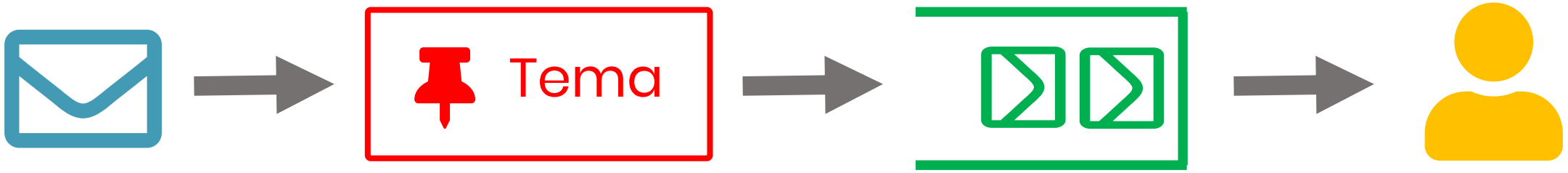
Tiene que ser pequeños

- Mensajes grandes afectan el rendimiento
- Grande generan una oportunidad de fracaso.
- Límites de tamaño.
- Descargas grandes en almacenamiento externo

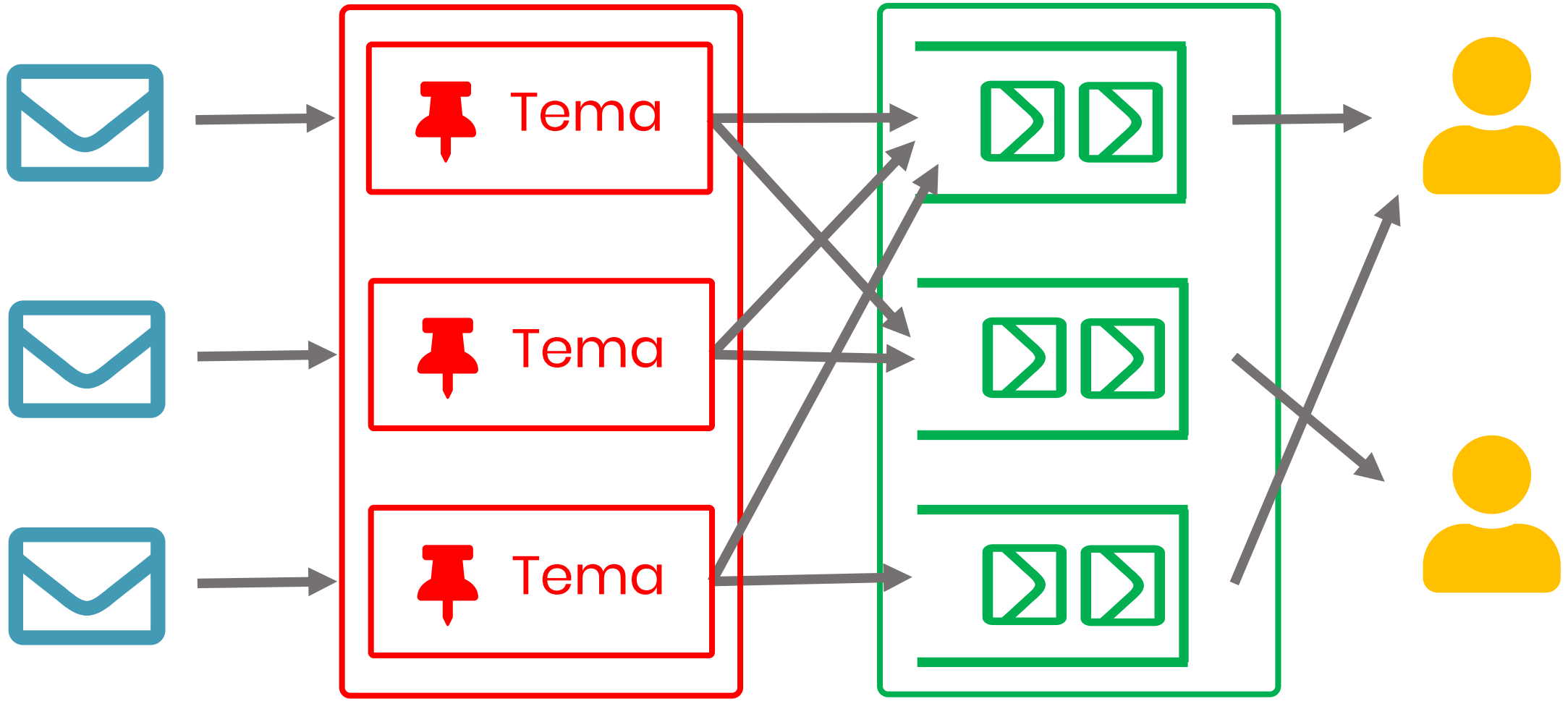
Definición Topic

- Post Liked, Post Created, Message Sent

Tema, ruteo y cola

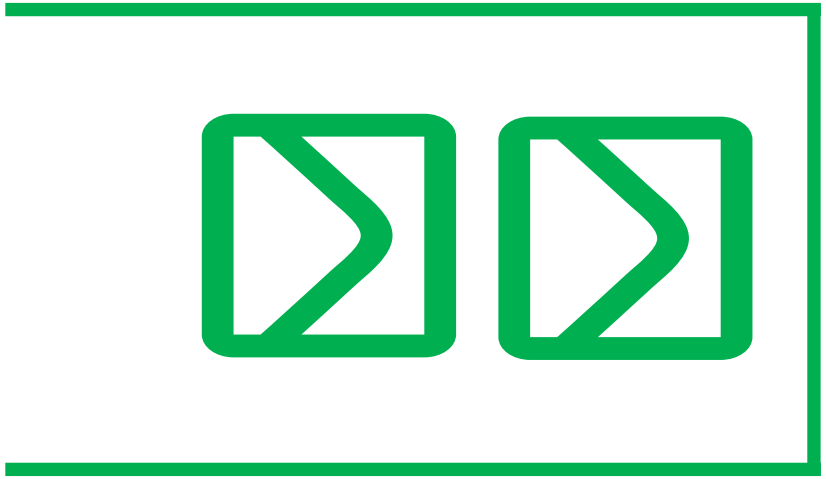


Tema, ruteo y cola

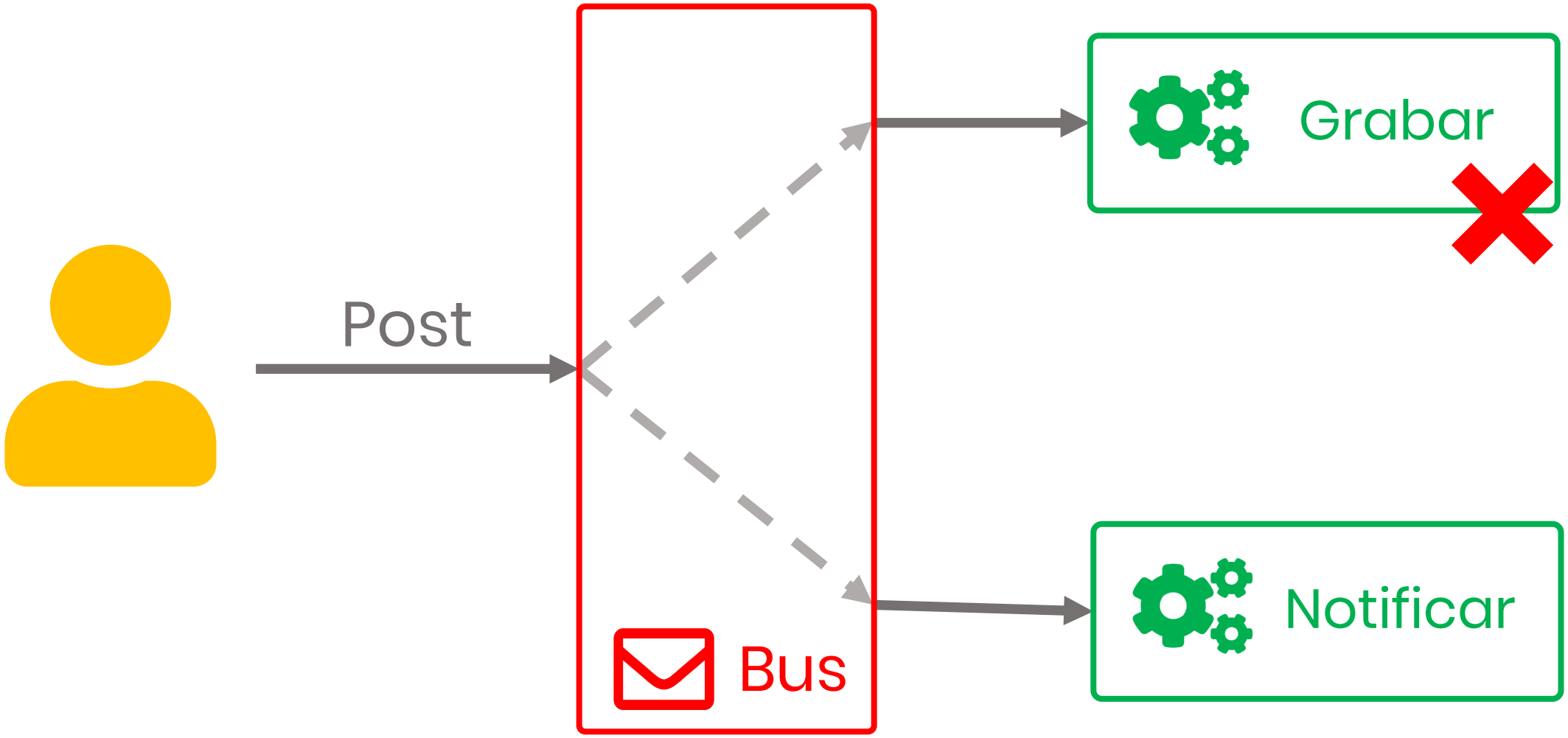


Cola de mensajes

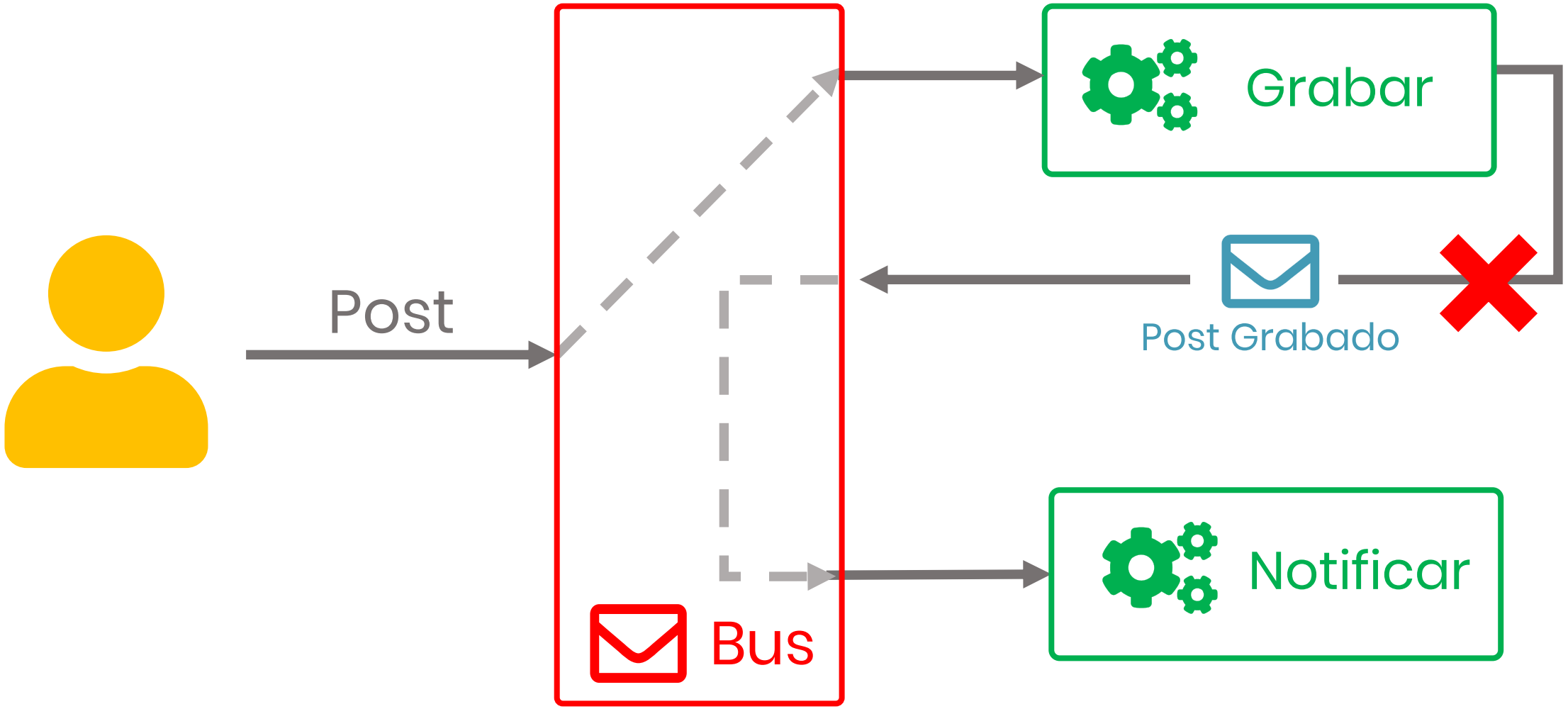
- Es un almacenamiento temporal
- Fácil recuperación ante un incidente
- Manejo de errores
  - Políticas de reintentos
  - Error de ruteo
    - Información de errores
    - Análisis y reprocesamiento
  - Letras muertas
    - Auto generadas
    - Fácil de reparar



Consistencia



Tema, ruteo y cola

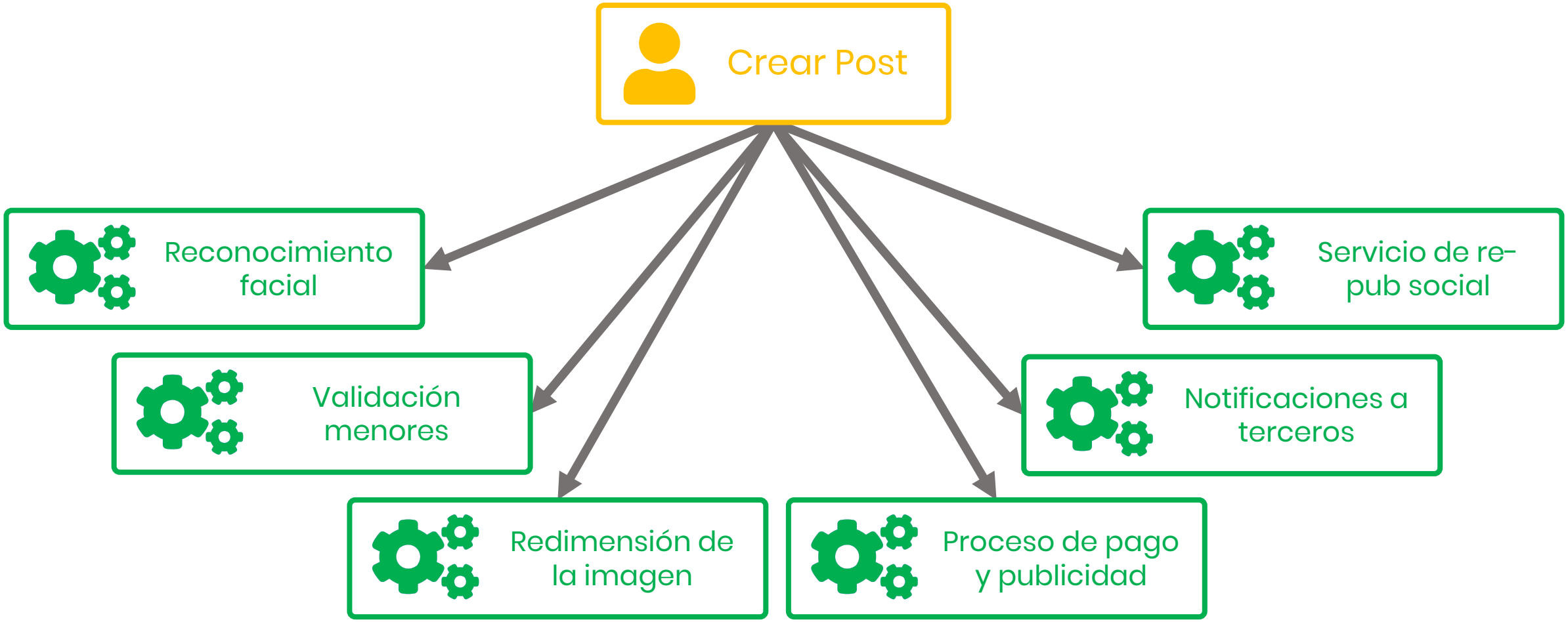


Arquitectura Event-driven

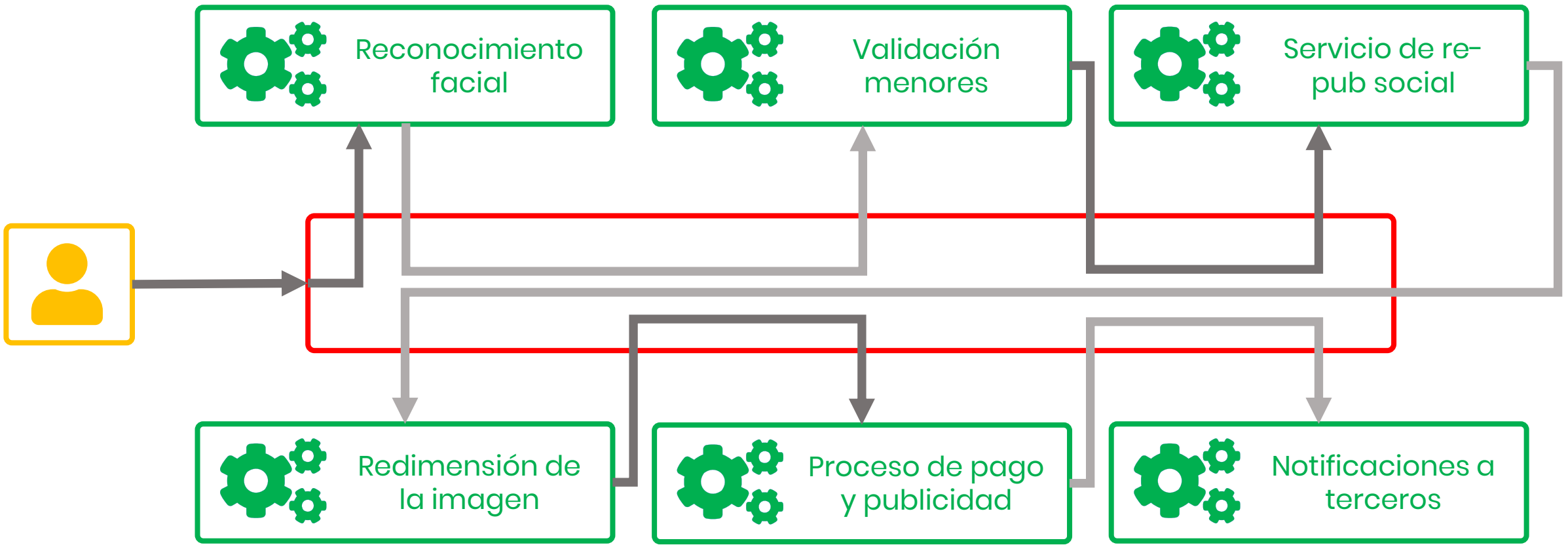
Impulsada por la generación, consumo, detección y reacción a los eventos

Transacción Distribuida

Transacción Distribuida



Event-Driven



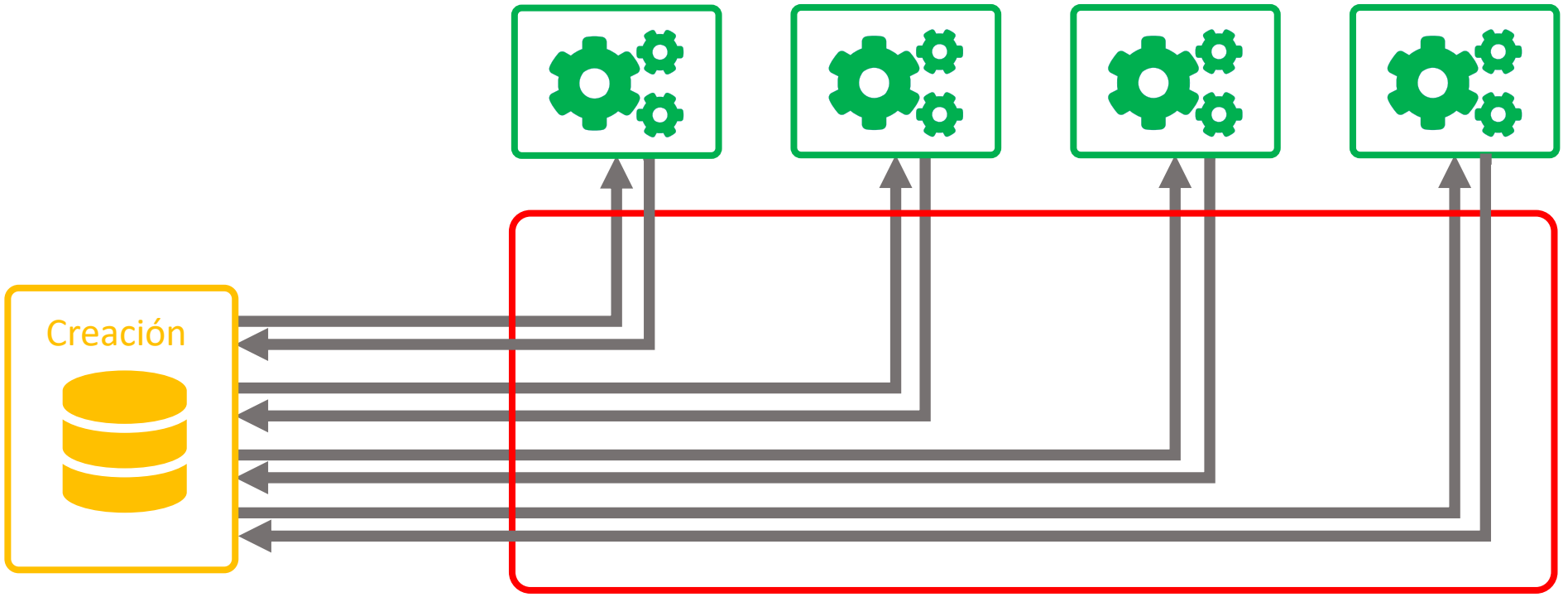
Saga y Routing Slip

Patrones de administration de mensajes

Patrón Saga

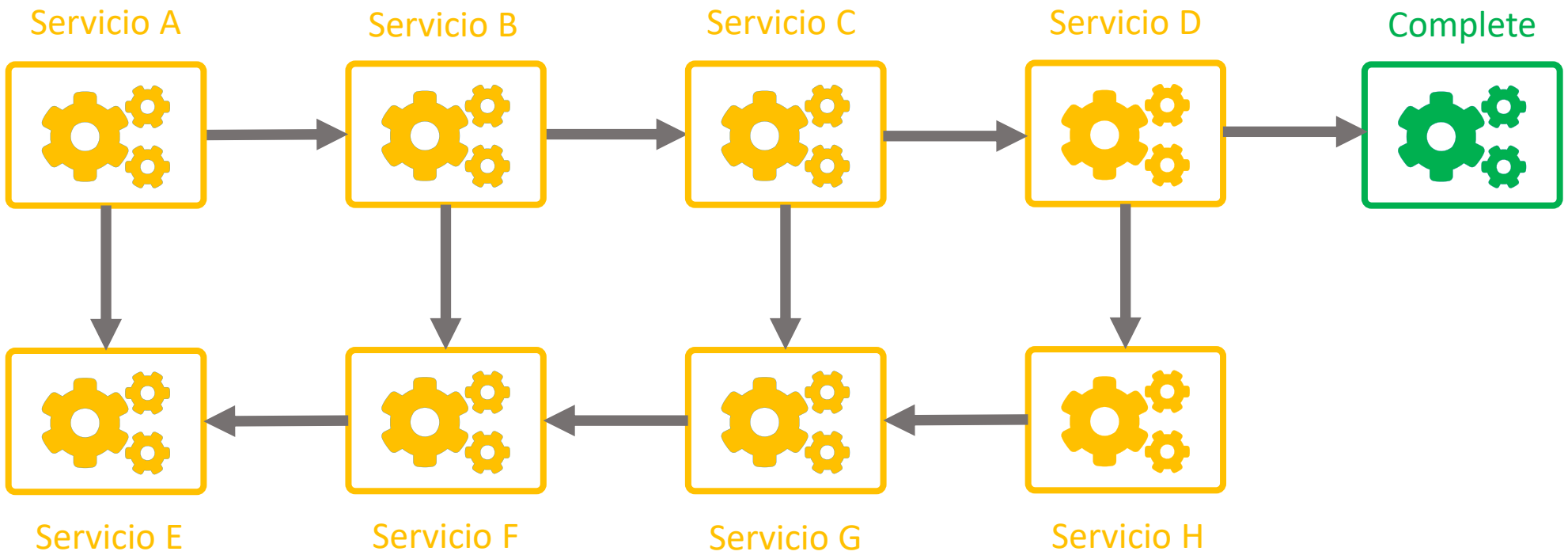
- Creado en la universidad de Princeton 1987
- Long-lived transactions (LLTs)
  - Que pueden durar horas o días
  - Una base de datos
  - Transacciones granulares
  - Compensa la carga de operaciones
- Para sistemas distribuidos extendidos
  - Se centraliza el estado de la transacción
  - Se maneja el ruteo de los mensajes
  - Fácil de tratar fallas

Patrón Saga



ID	State
Zxy856	Servicio 1
Dfe785	Servicio 2
Fgt874	Compledo

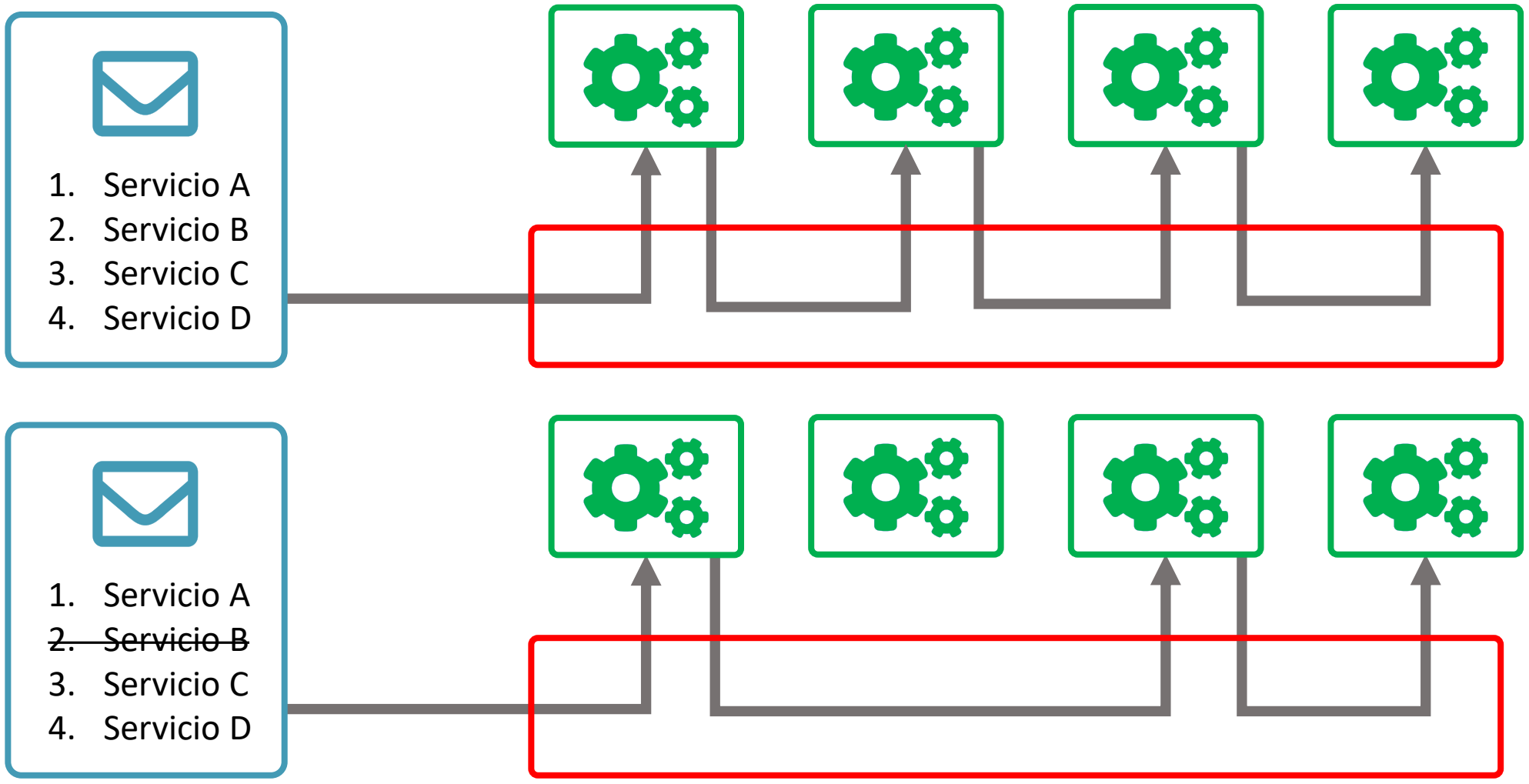
Compensación de fallas



Patrón Routing Slip

- Instrucciones adjuntas al elemento de trabajo
  - En que puntos
  - Tipo de trabajo por punto
  - Que instrucción
- En cada punto de parada
  - Hace el trabajo
  - Pasa al siguiente punto
- Determina los pasos del procesamiento
- Va adjunto al mensaje
- El mensaje tiene la secuencia de ruteo

Patrón Routing Slip



Saga Vs Routing Sleep

Cuando	Saga	Routing Slip
Ruteo del mensaje	Mejor cuando es simple	Flexible, especialmente cuando tenemos muchos servicios
Centralización del estado	Perfecto	No es posible
Compensación de fallas	Bueno cuando es simple, pero, fácilmente se vuelve complicado	Sencillo
Cancelación	Sencillo por la centralización del estado	Requiere una complejidad adición en cada servicio
Paralelizar carga	Se puede	La mayoría del tiempo debe ser lineal

Inconsistencia Eventual

- Algo que lo clientes deben empezar a conocer
- Los estados de la información estarán inconsistentes
- Servicios independientes
- Incomodos para los que les encanta seguir ACID

Resumen: Comunicación Asincrónica

Beneficios

- Invocadores y receptores están desacoplados
- Aislados para una mejor administración de carga
- Podemos aislar los servicios mas lentos.
  - Funcional
  - Portable

Consistencia

- Manejo granular
- Secuencia precisa
- Prevé estados inconsistentes

Comunicación
Asincrónica

La arquitectura es compleja

Conclusiones y Preguntas

Dale dinamismo a tu código C#

sábado, 27 de marzo de 2020

Fernando Sonego

Solution Architect

fernando.sonego@gmail.com

@FernandoSonego

www.withoutdebugger.com