



Microservicios con ASP .NET Core

Ing. José Díaz (@jamdiazdiaz) - 30 Mayo 2020

José Amadeo Díaz Díaz



Gerente de Arquitectura y
Productos Digitales



jadiaz@farmaciasperuanas.pe



[@jamdiazdiaz](https://twitter.com/jamdiazdiaz)



Fundador de JoeDayz.pe

Java Champion

Miembro de @PeruJUG



Agenda

- ¿Por qué micro servicios?
- Refactorizando el Monolito
- Implementando nuevas características como servicios
- Extrayendo módulos en servicios
- Tecnologías para micro servicios
- Patrones de Diseño

¿Por qué micro servicios?



+

El Mercado es volátil, incierto, complejo y ambiguo

+

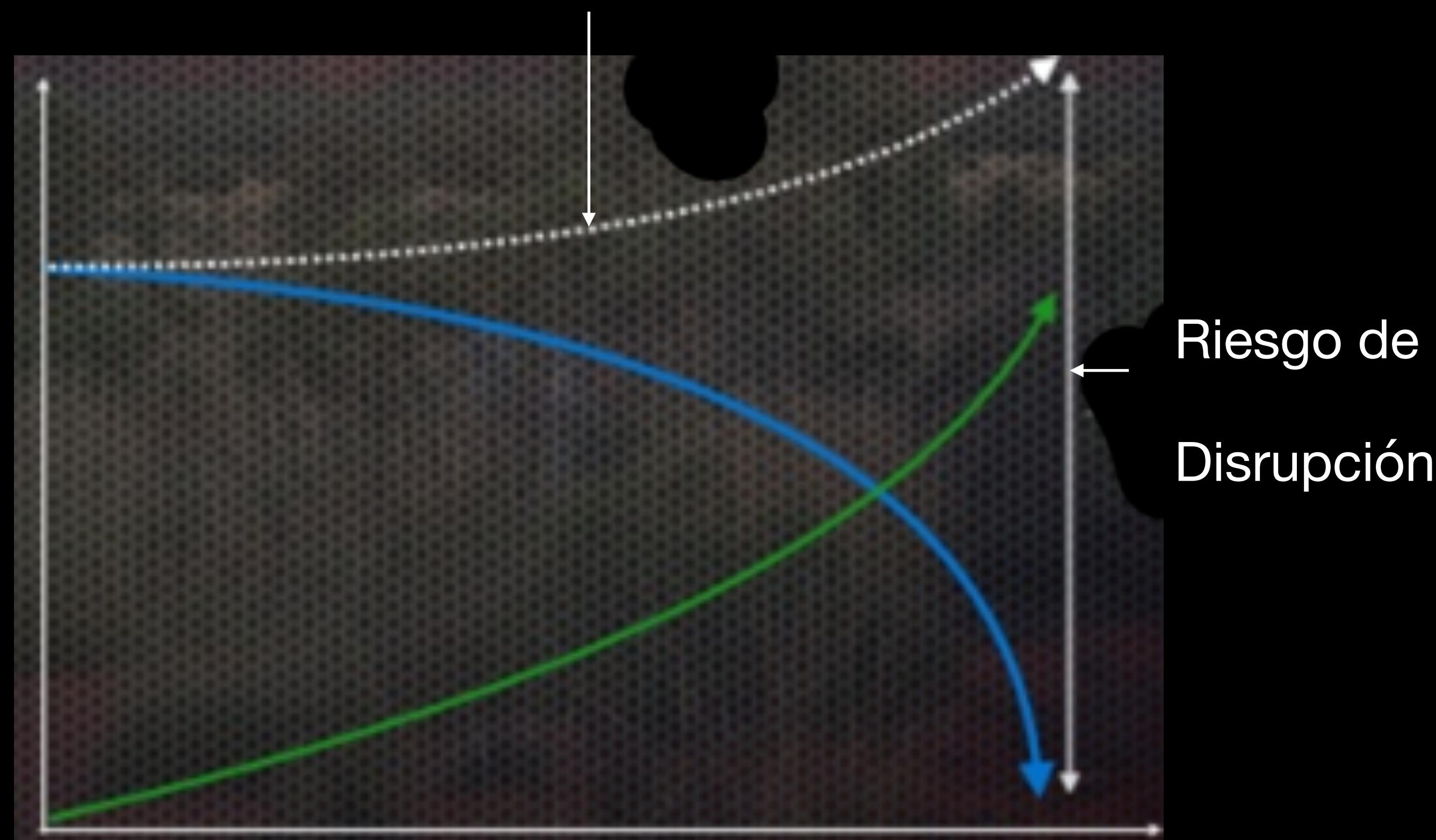
Los negocios deben innovar rápido

+

Entregar software rápido, frecuente y sostenible

Arquitectura Monolítica - Las ilusiones van disminuyendo con el tiempo

Tamaño/
Complejidad
Mantenimiento
Testable
Desplegable
Modular
Evolutivo



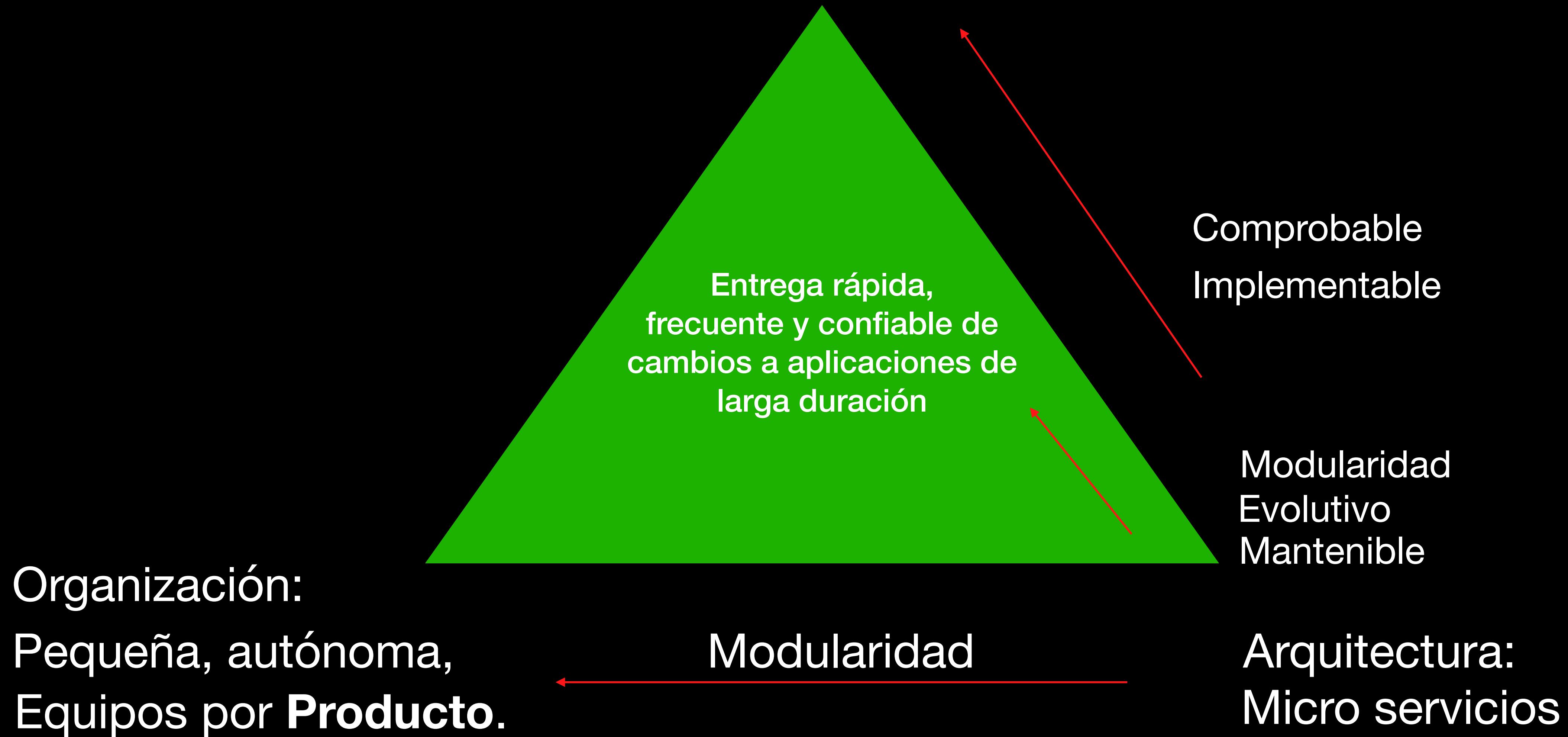
La arquitectura de micro servicios es
un estilo de arquitectura
que estructura una aplicación como un
conjunto de servicios*

Cada ~~micro~~servicio es:

- Altamente mantenible y testeable
- Pobremente acoplado
- Desplegable de forma independiente
- Organizado por capacidades de negocio
- Propiedad de un equipo pequeño

* Comienza con un servicio por equipo hasta que esto sea un problema.

Proceso: Lean + DevOps/Delivery Continuo & Despliegue



Agenda

- ¿Por qué micro servicios?
- Refactorizando el Monolito
- Implementando nuevas características como servicios
- Extrayendo módulos en servicios
- Tecnologías para micro servicios
- Patrones de Diseño

Refactorizando el Monolito

No, lo hagas!



Anti-patrón: Polvos mágicos

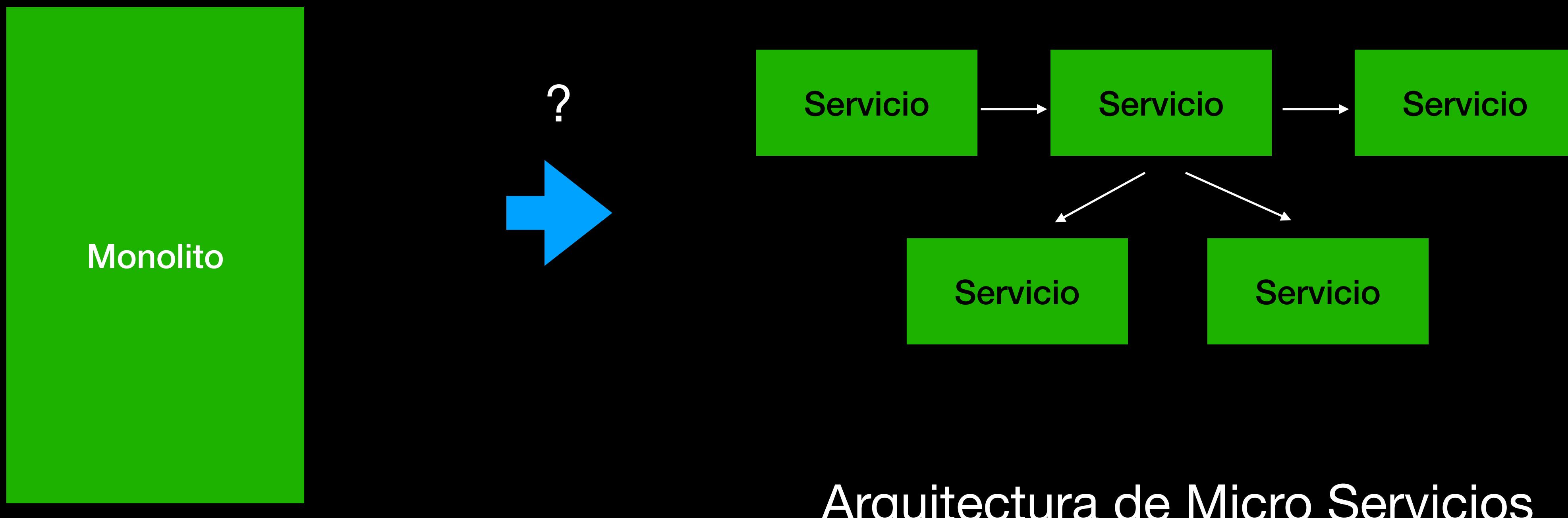
Aprovechemos la arquitectura monolítica

- La arquitectura monolítica no es un anti-patrón
- Si la entrega de software es lenta
 - Optimiza el proceso de desarrollo
 - Optimiza el pipeline de despliegue = mas automatización
 - Optimiza la autonomía del equipo
 - Modulariza el monolítico
 - Crea equipos multi-funcionales
- Si el stack de tecnología es obsoleto, moderniza a un nuevo monolito.
- ...

**Si y sólo si, esto no es suficiente*,
entonces considera
migrar a micro servicios**

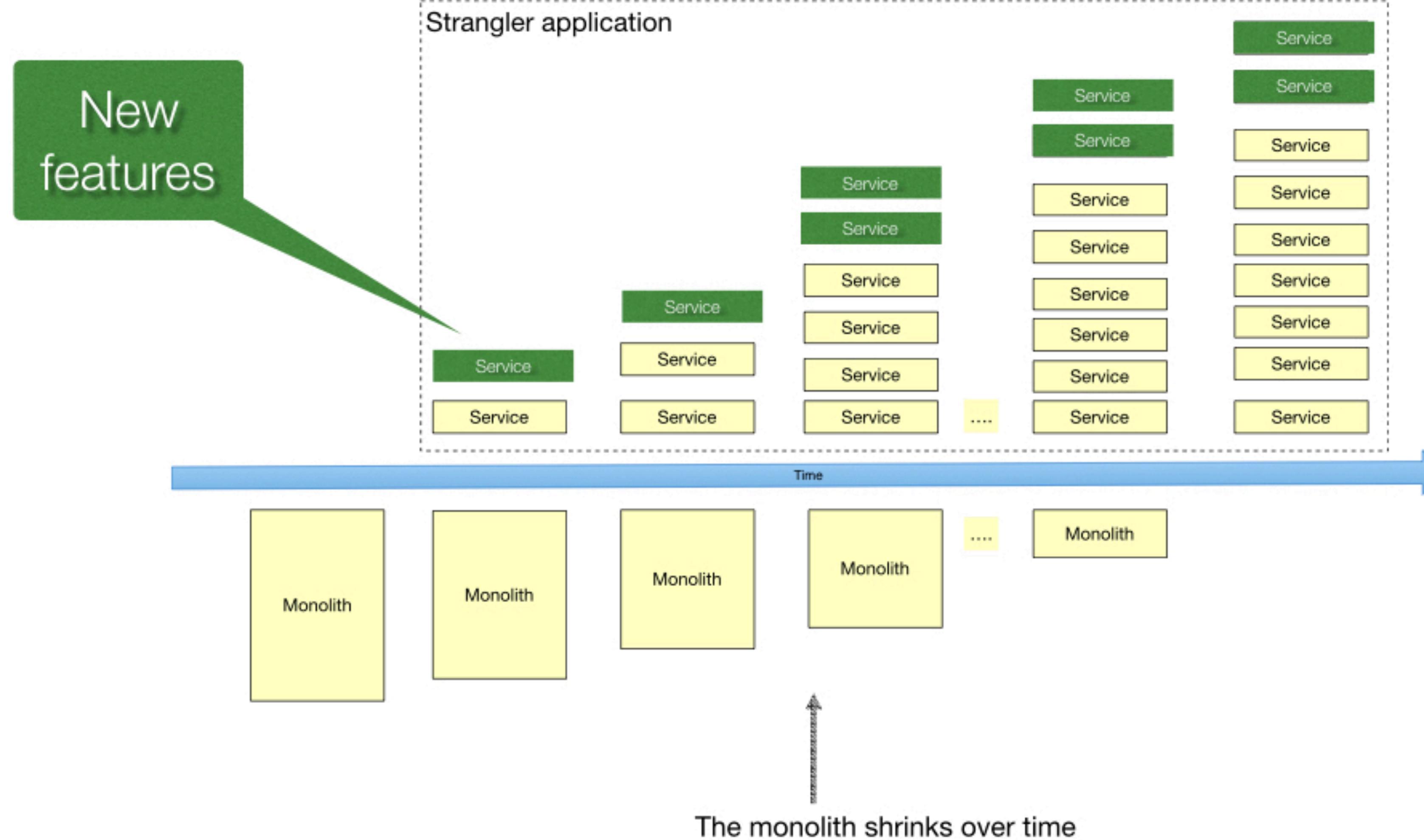
- * Grandes y complejas aplicaciones son desarrolladas (en general) por un gran equipo que necesita entregar rápido, de forma frecuente y sostenible.

¿Cómo descomponemos nuestra gran aplicación monolítica?

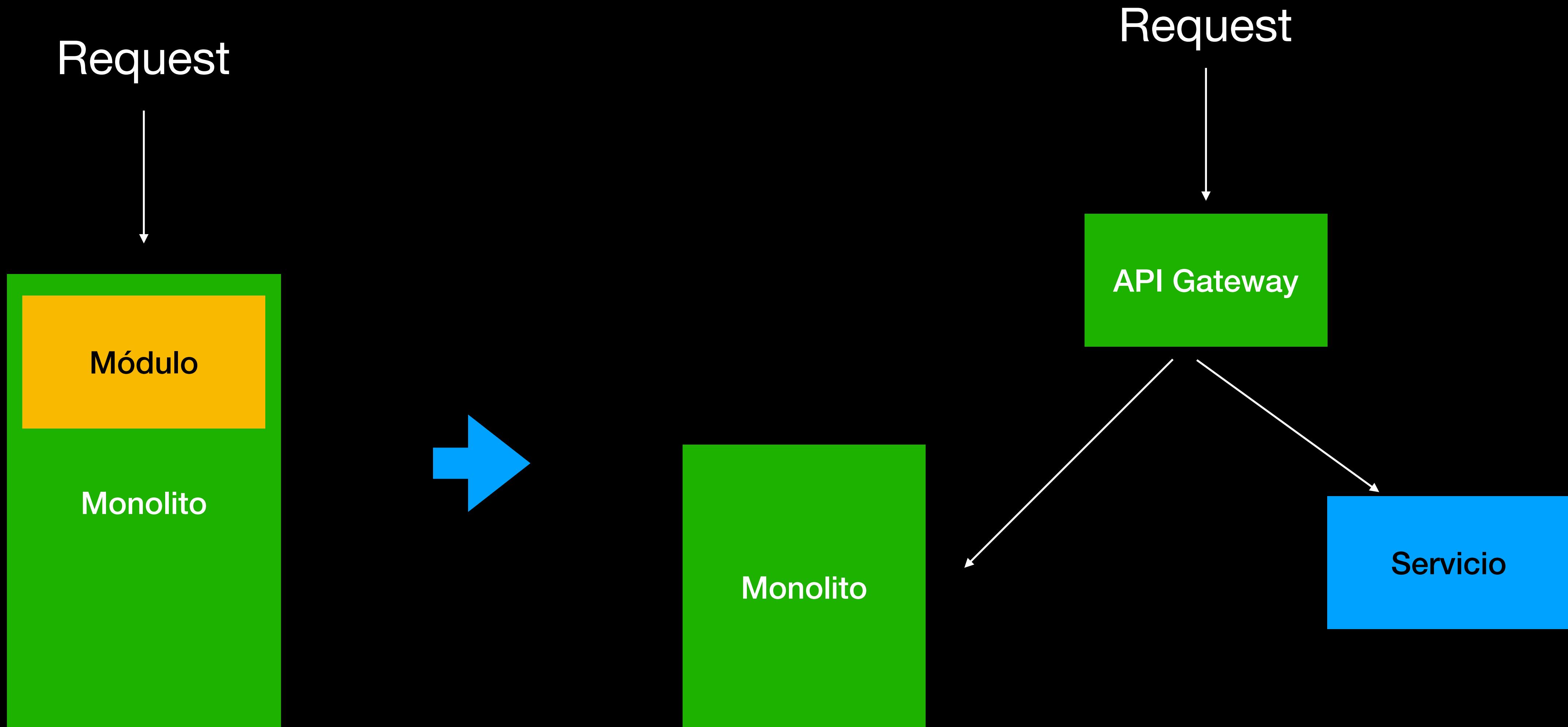


Strangling the monolith

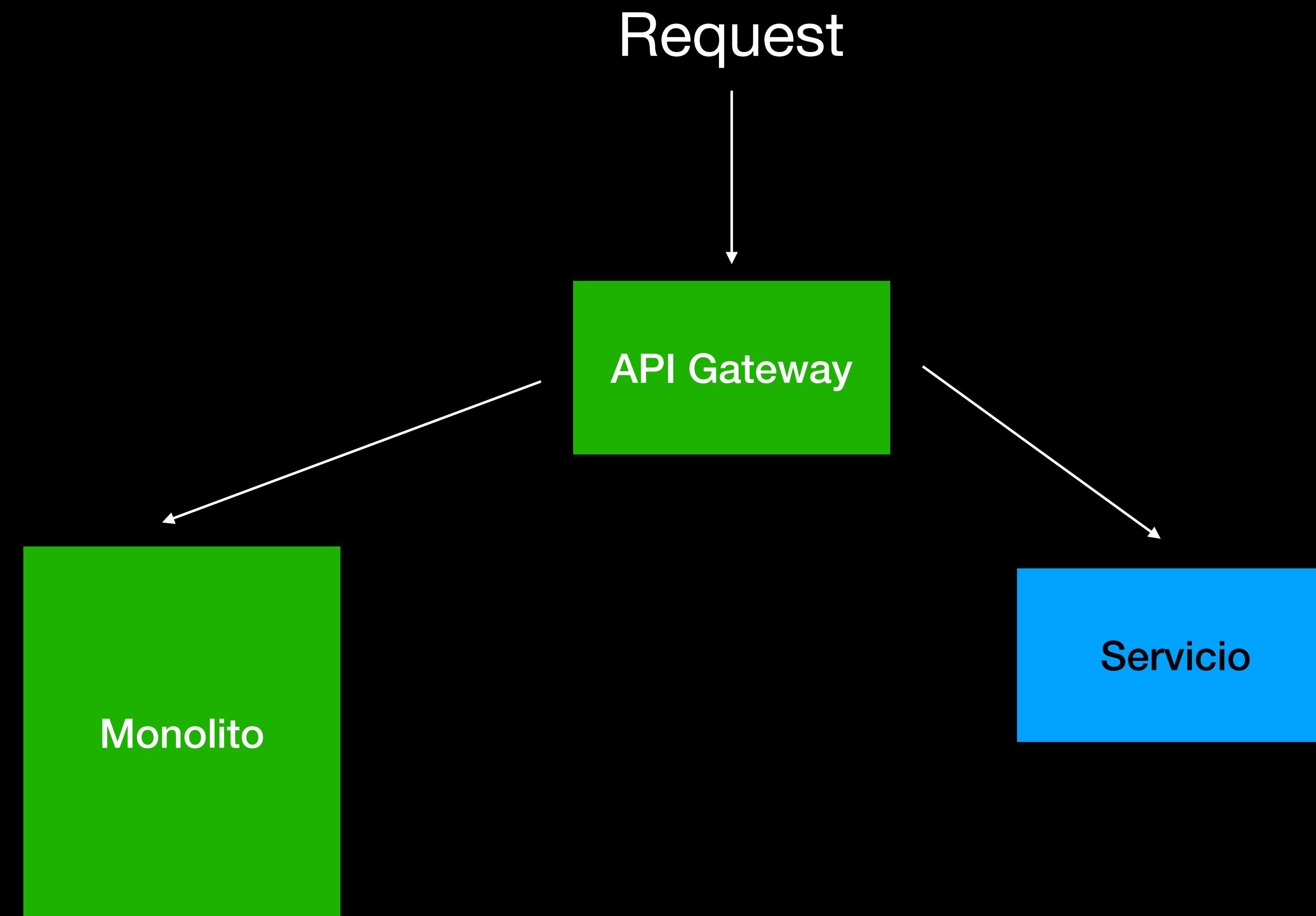
The strangler application grows larger over time



Iterativo: Módulo => Servicio



Iterativo: Nueva Característica => Servicio



Mide el éxito

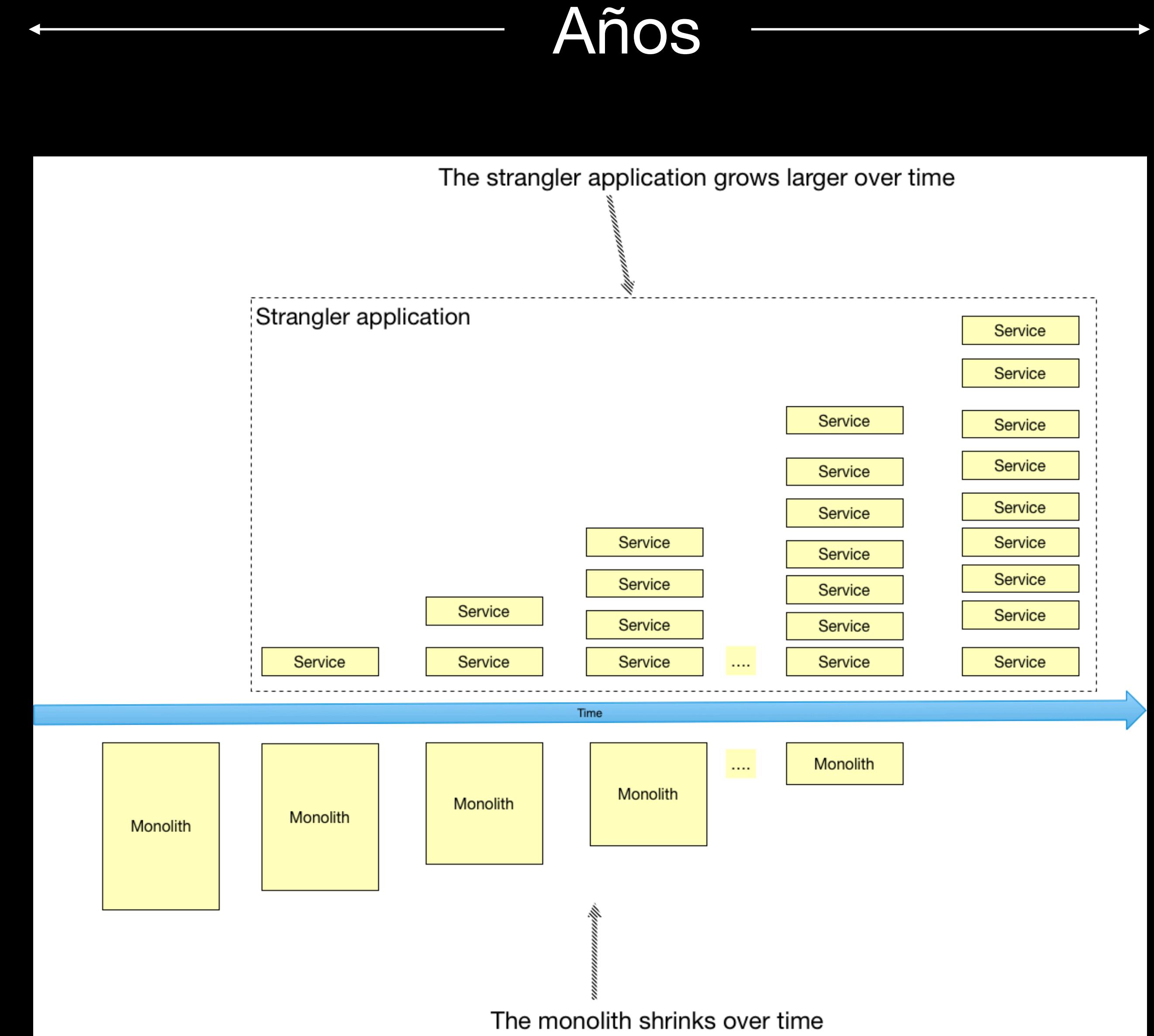
- Exito != número de micro servicios
- Mejora tus métricas:
 - Reduce el tiempo de espera
 - Incrementa la frecuencia de despliegue
 - Reduce el % de fallas por cambios
- Mejora en nuevos “features”
- ...

Anti-patrón:
Micro servicios es el objetivo!



Repetir extraer servicios hasta que:

- Eliminar el monolítico
- Problemas de entregas de software solucionados
- Trabajo con alta prioridad



¿Qué extraer?

- Te en cuenta la arquitectura de micro servicios ideal
 - El esfuerzo de definición de arquitectura es limitado
 - Siempre preparate para hacer una nueva revisión con el aprendizaje sobre la marcha
- Comienza con los módulos que te darán el mayor retorno de inversión (ROI)

Costo vs Beneficio de extracción

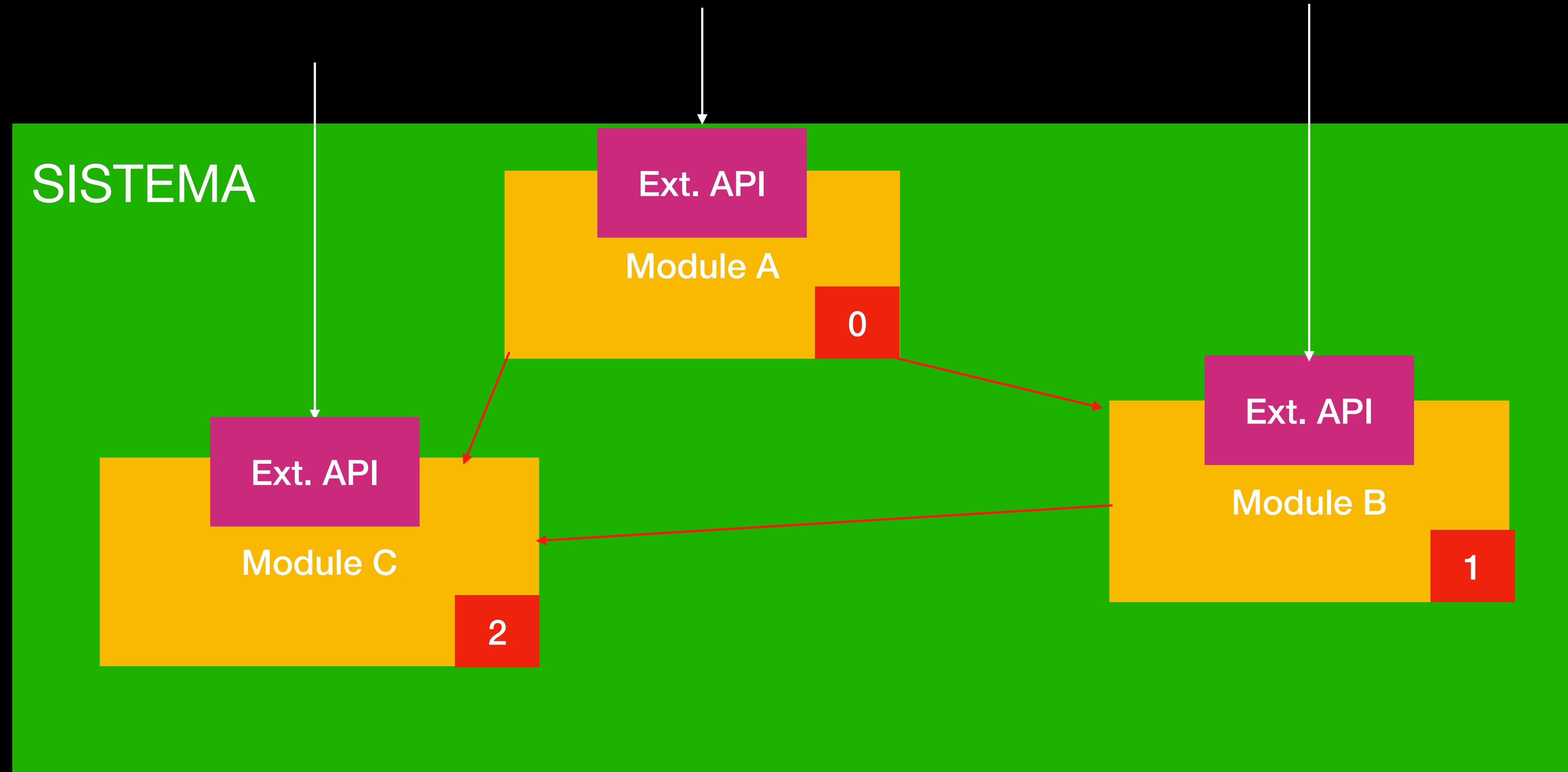
Beneficios

- Soluciona un problema significante
- Velocidad frecuente de actualización
- Escalabilidad, resuelve conflicto de requerimientos de recursos
- ...

Costo

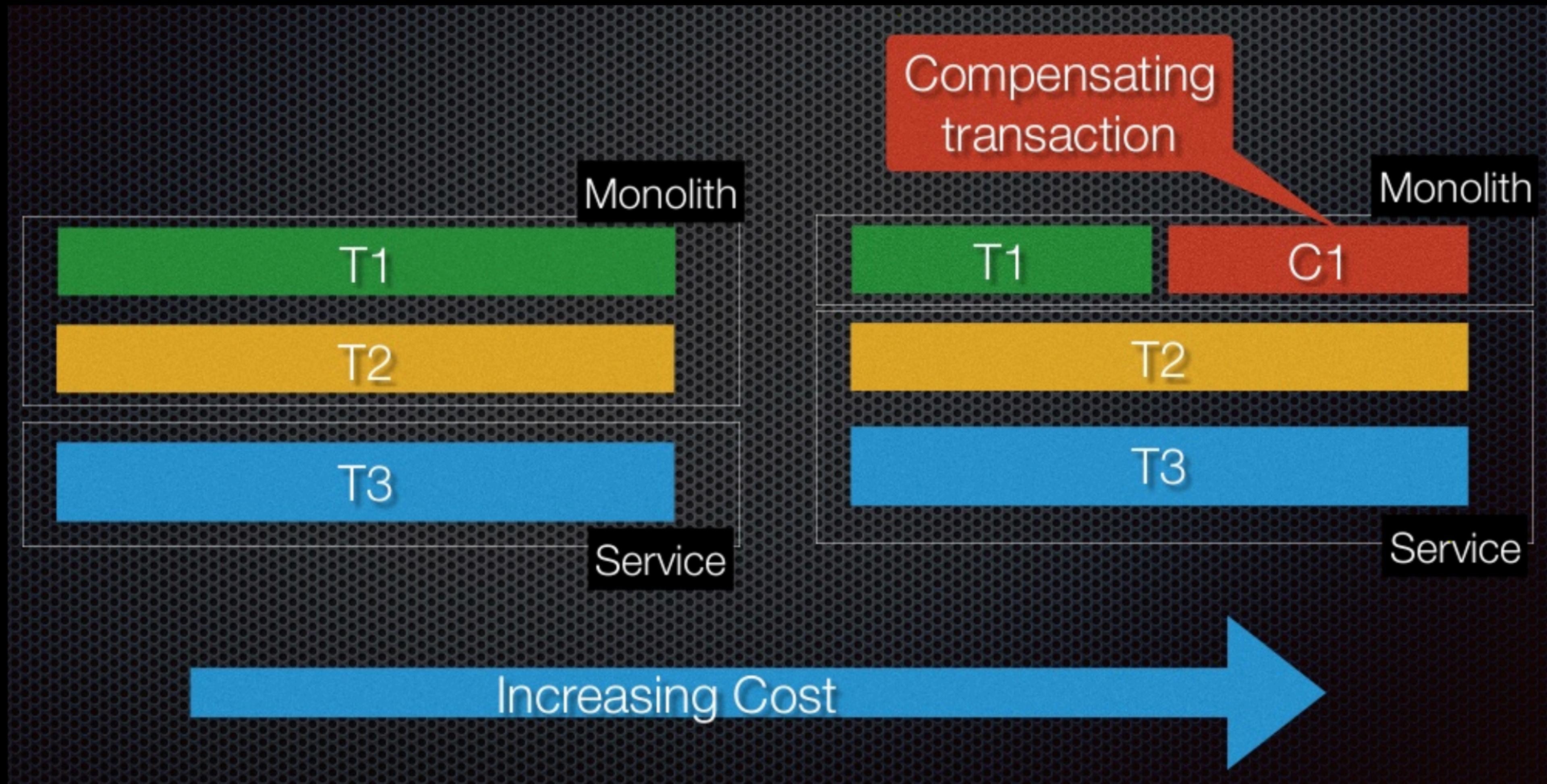
- Costo de cambiar el monolito y adaptación / reescribir el módulo
- Dificultad en desacoplar/ romper dependencias
- Necesidad de participar en SAGAS/transacciones de compensación

Costo de desacoplamiento: # de dependencias entrantes



Dependencias pueden determinar ordenamiento

Costo: administración de transacciones / Sagas



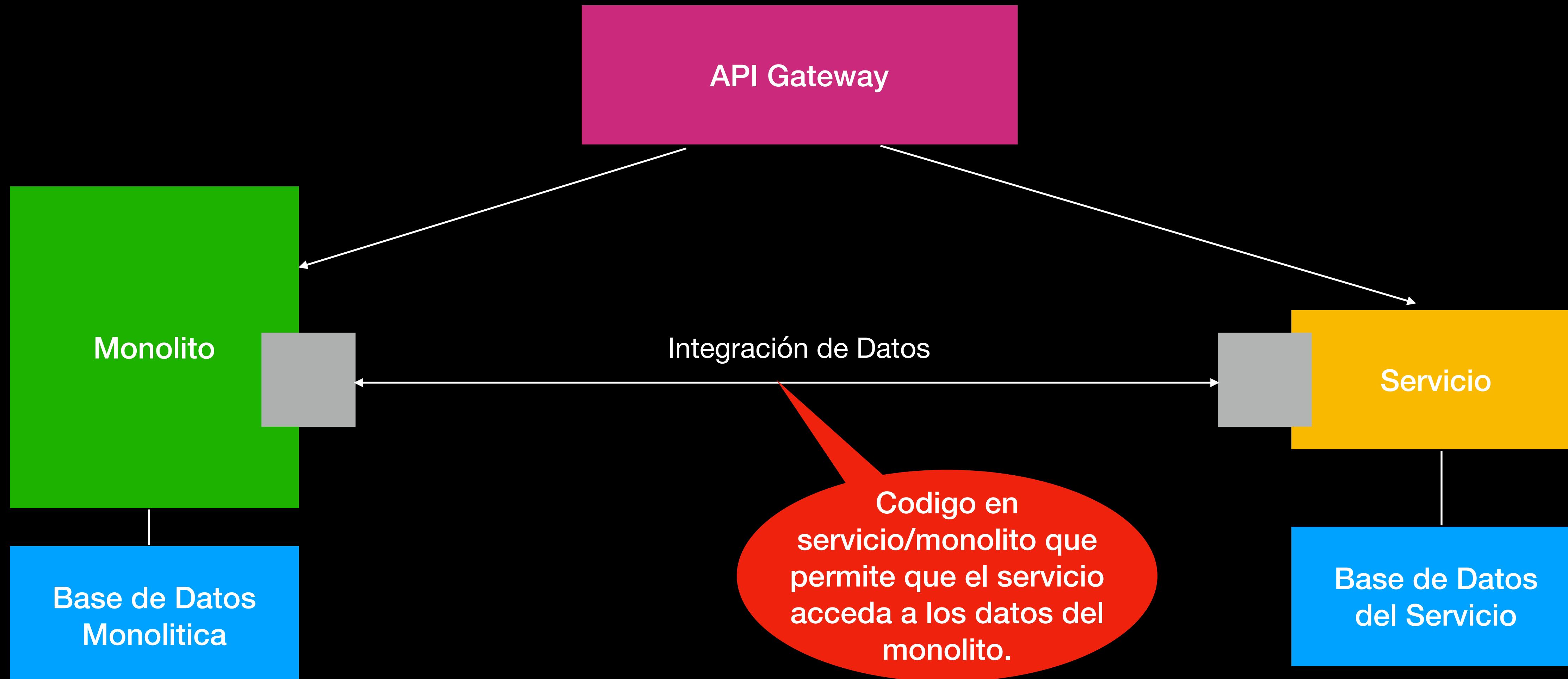
Agenda

- ¿Por qué micro servicios?
- Refactorizando el Monolito
- Implementando nuevas características como servicios
- Extrayendo módulos en servicios
- Tecnologías para micro servicios

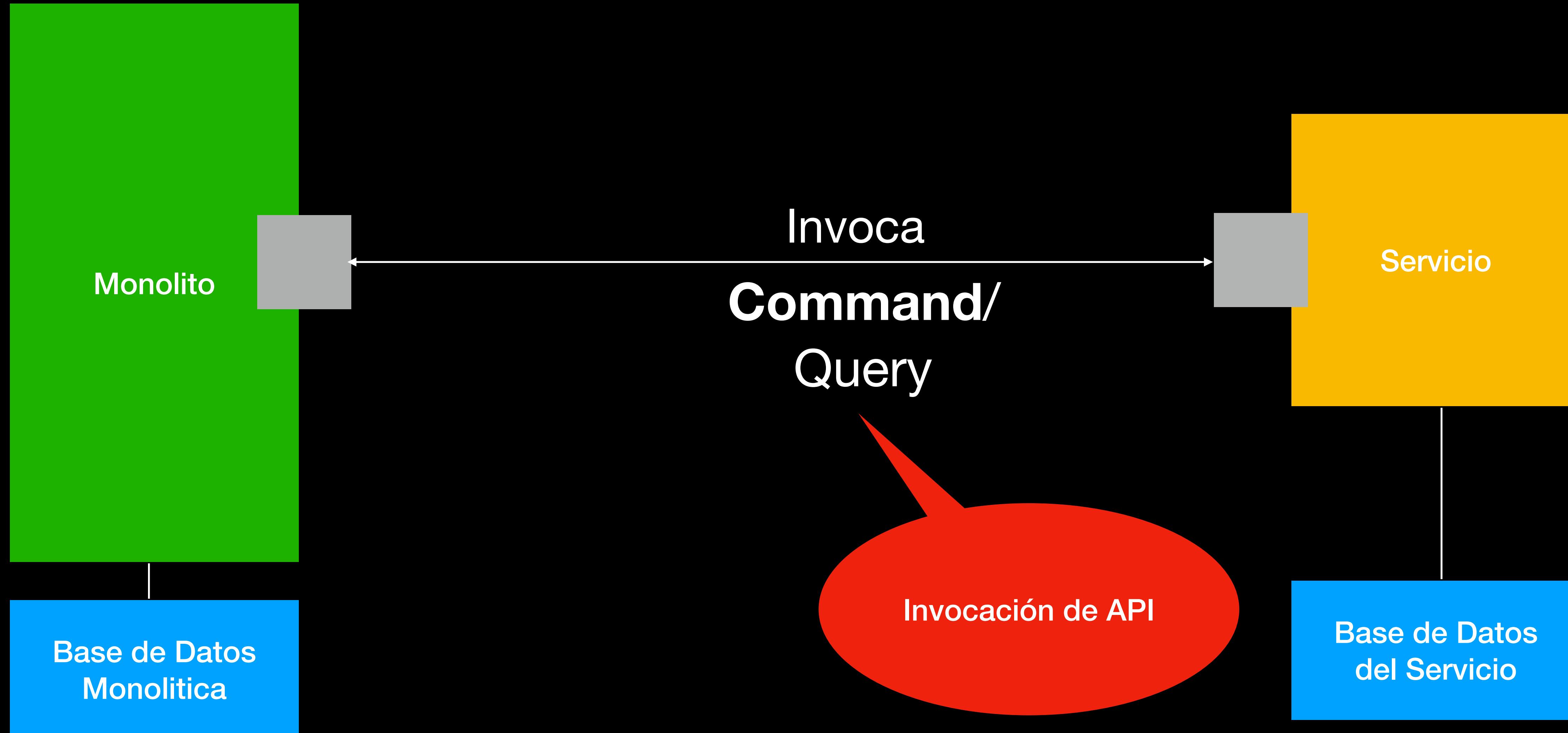
Implementando nuevas
características como servicios

Implementando nueva funcionalidad como servicio

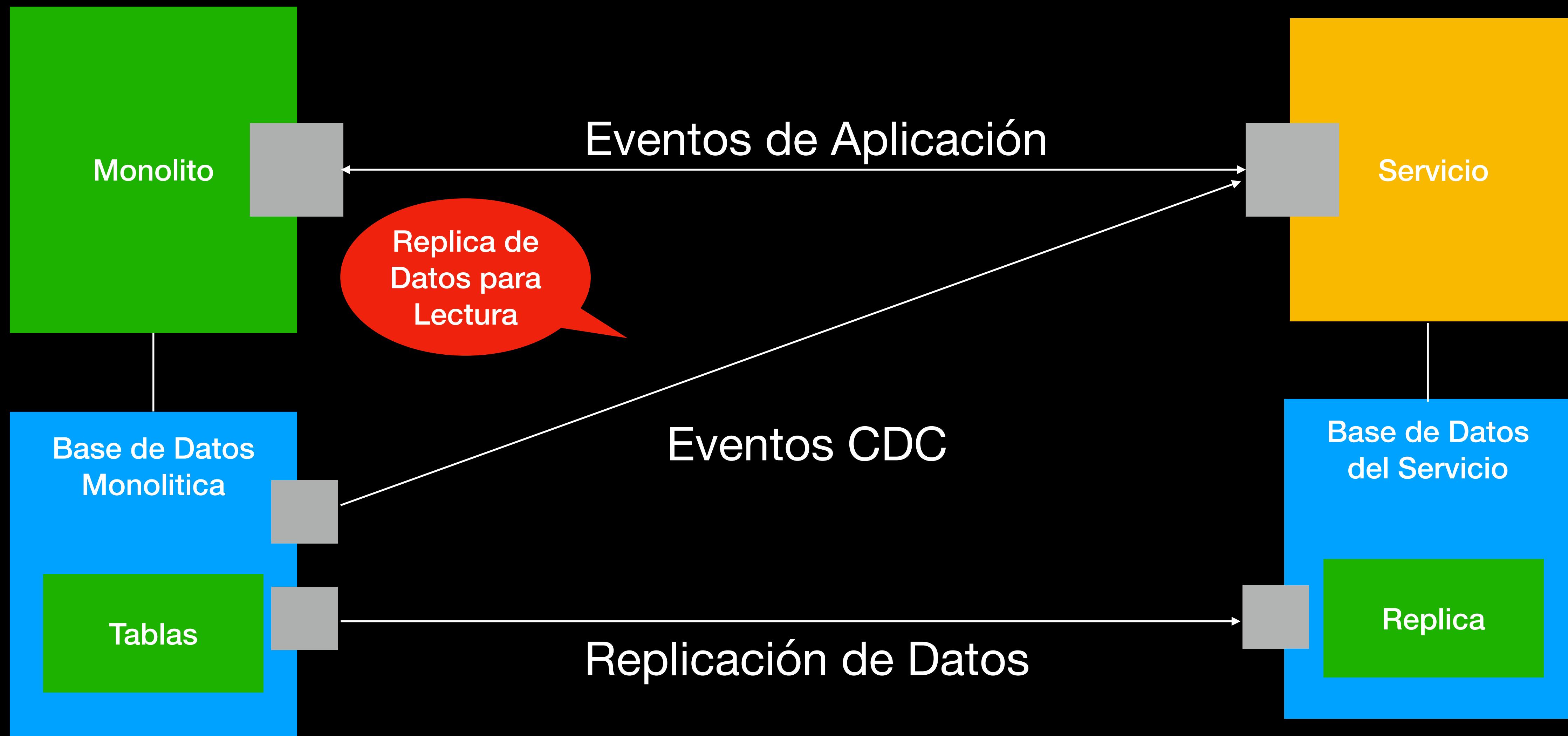
Request



Integración de Datos: Servicio - Monolito



Integración de Datos: Servicio - Monolito

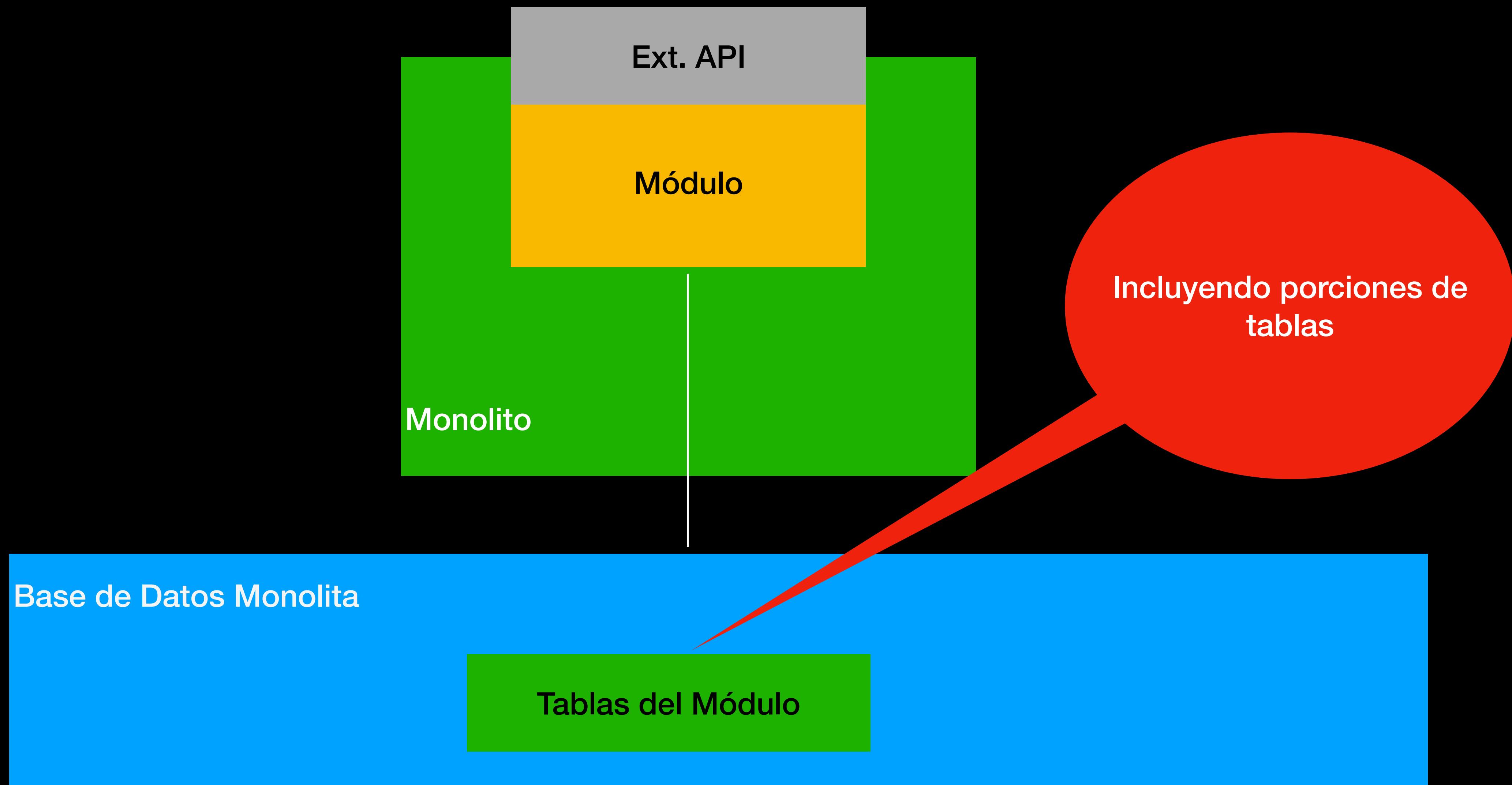


Agenda

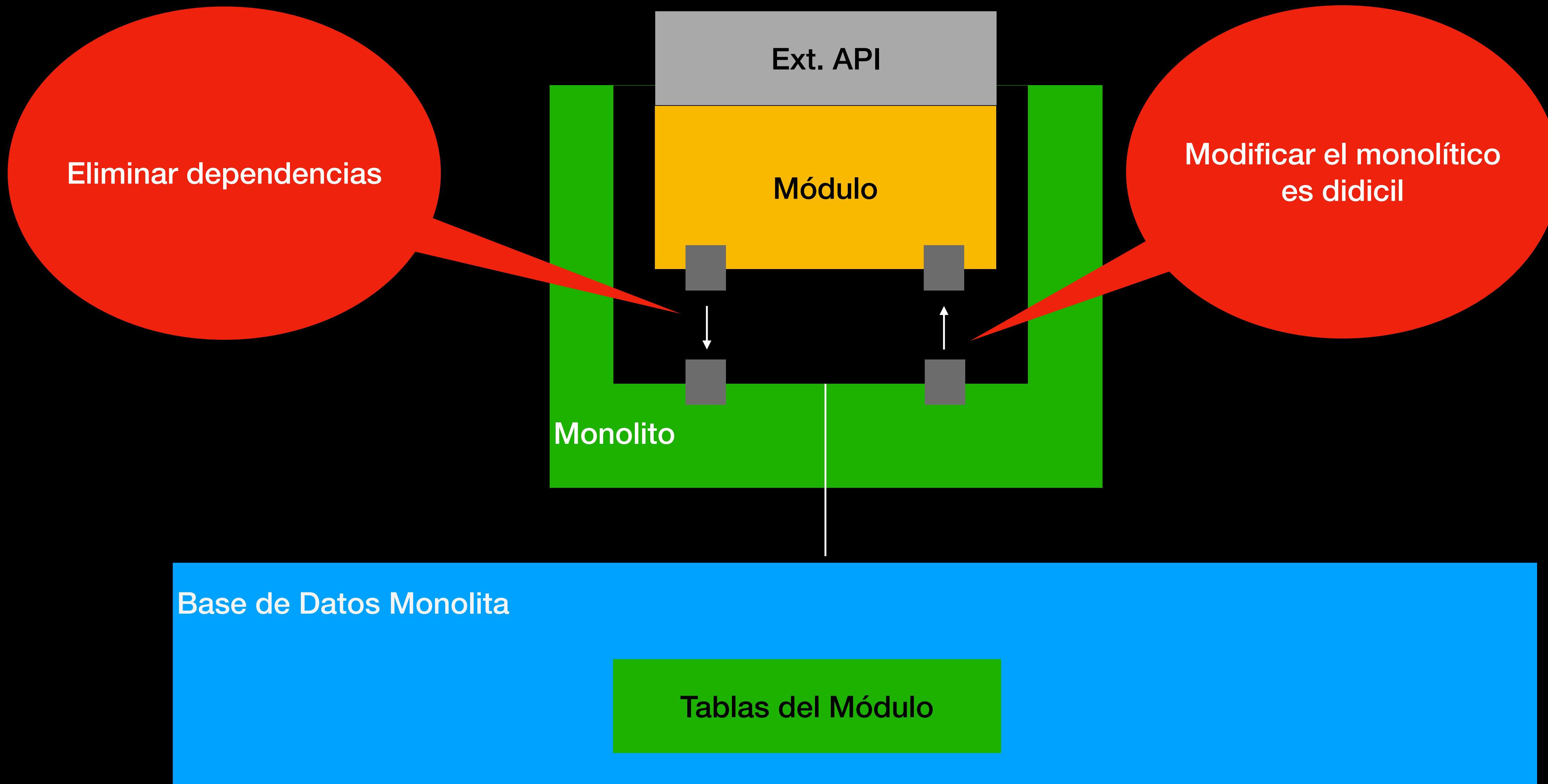
- ¿Por qué micro servicios?
- Refactorizando el Monolito
- Implementando nuevas características como servicios
- Extrayendo módulos en servicios
- Tecnologías para micro servicios
- Patrones de Diseño

Extrayendo módulos en servicios

Módulo => Servicio ...



Definiendo una interface “remotable” bidireccional

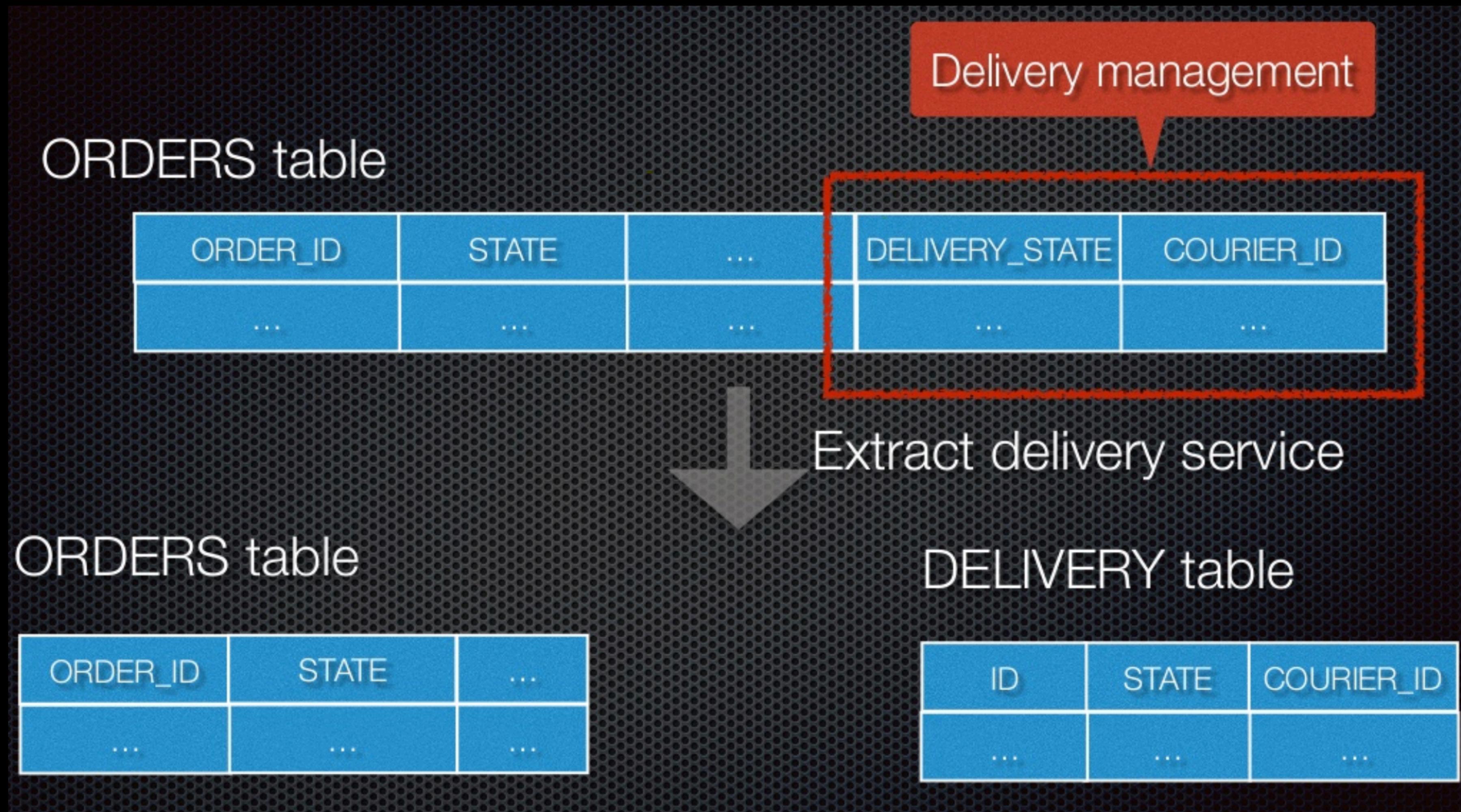


Incrementalmente modificar el monolito invocando al modulo vía nueva API*

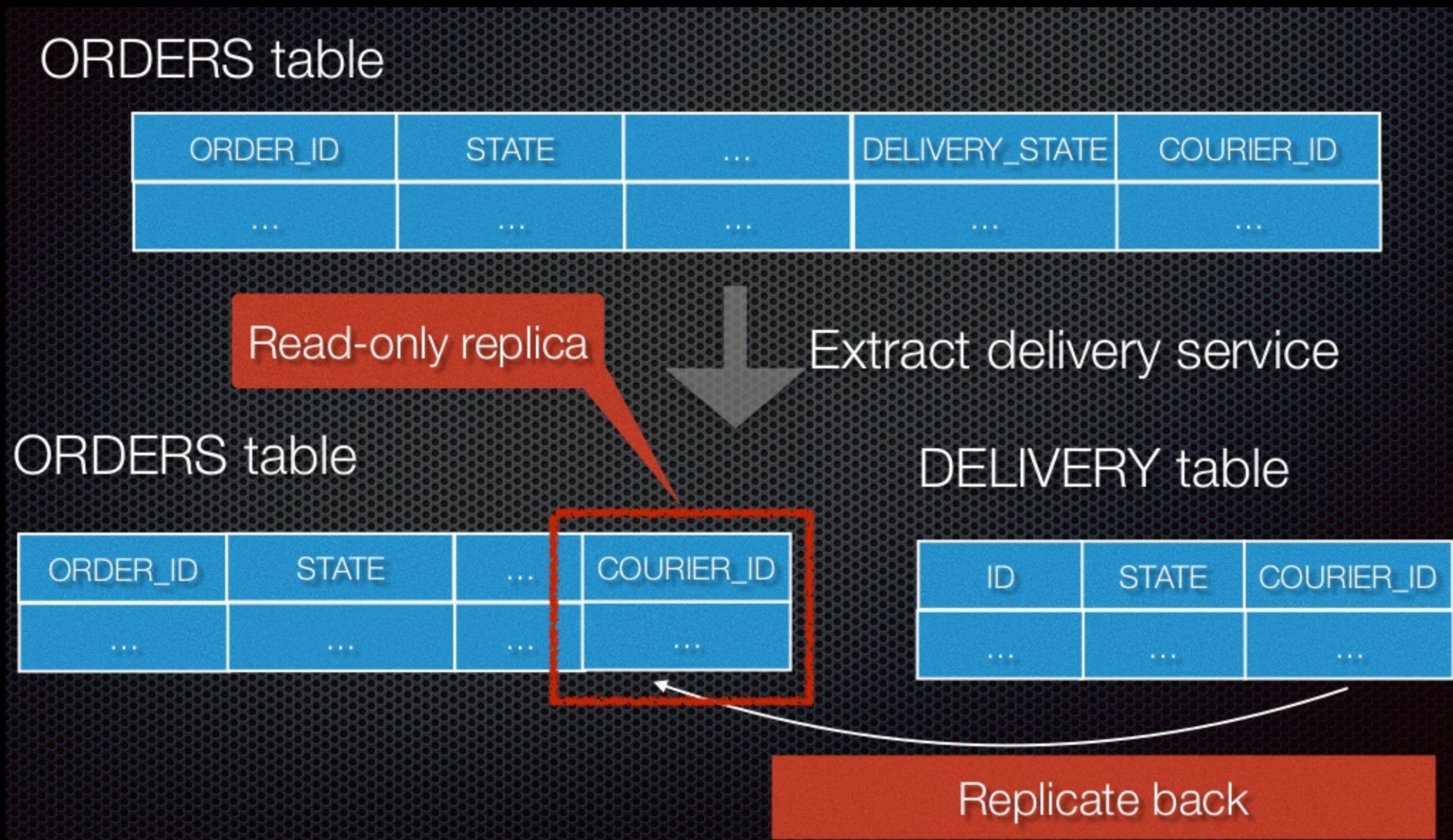


Patron: Branch by abstraction

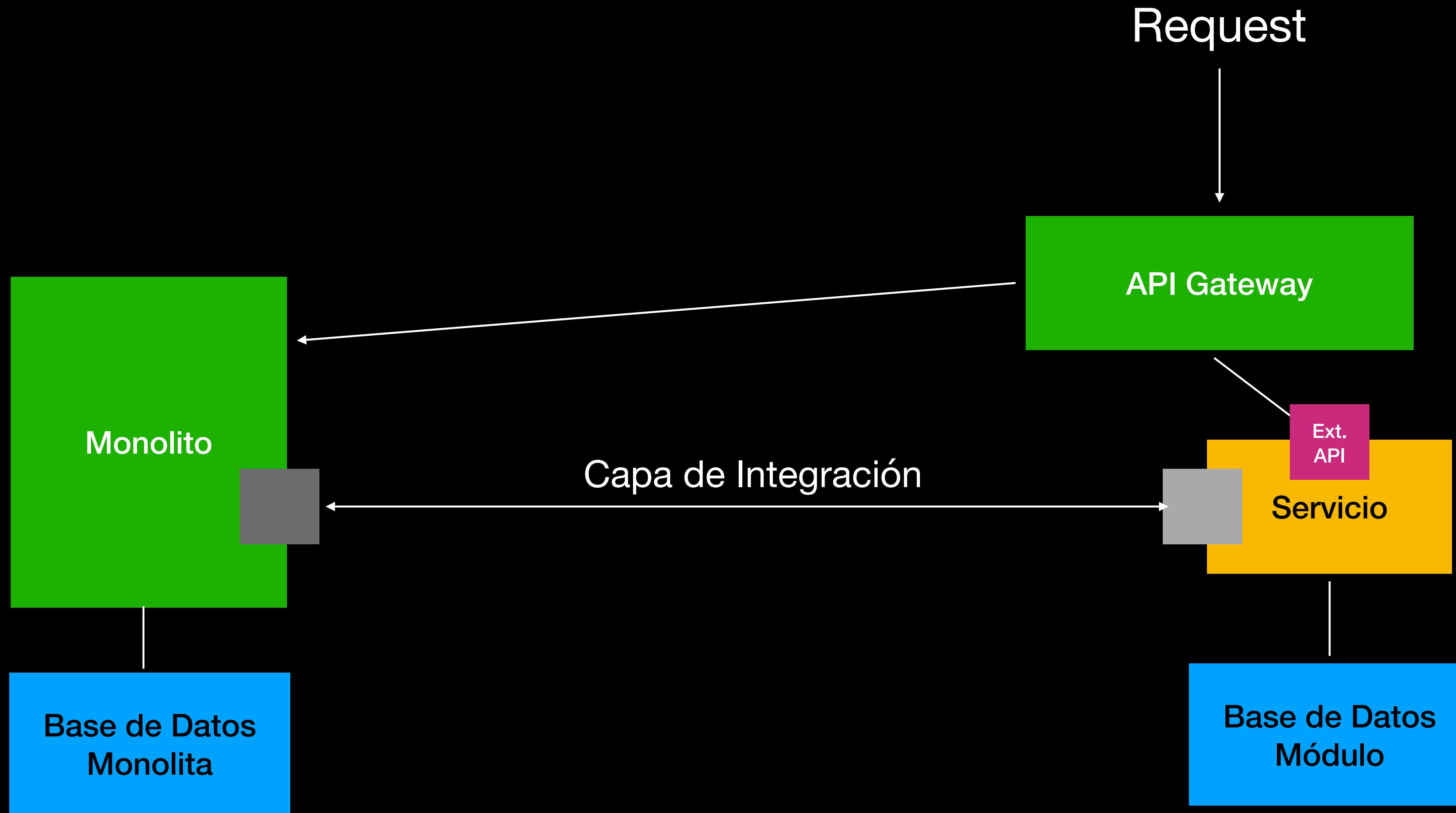
Transformar el esquema de base de datos



Replicar datos reduce el alcance del refactoring



Modulo => Servicio



Agenda

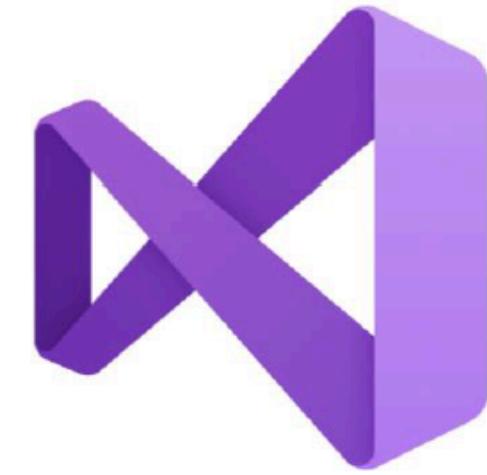
- ¿Por qué micro servicios?
- Refactorizando el Monolito
- Implementando nuevas características como servicios
- Extrayendo módulos en servicios
- **Tecnologías para micro servicios**
- Patrones de Diseño

Technologías para Micro servicios

Frameworks de Desarrollo

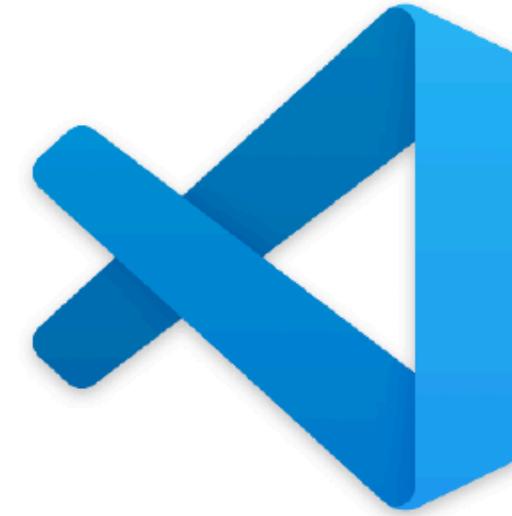


Herramientas de Desarrollo



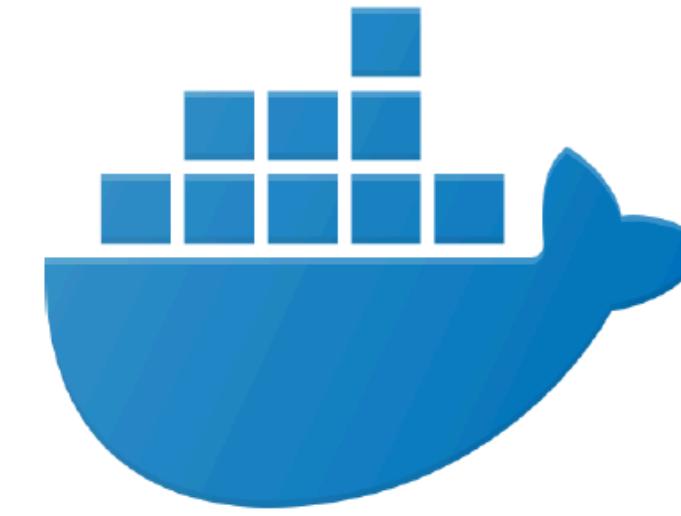
Visual Studio

www.visualstudio.microsoft.com/



VS Code

www.code.visualstudio.com/



Docker

www.code.docker.com/

API Management : API Gateways

- Ruteo
- Composición de API
- Caching
- Logging
- Seguridad
- Límite de velocidad

Micro servicios - API Gateways



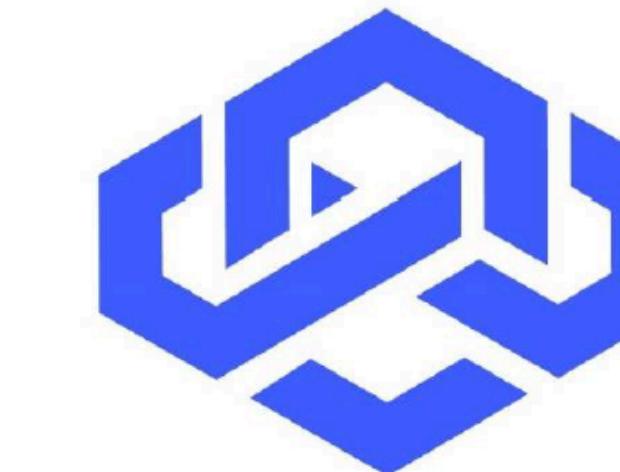
Azure API Management



Ocelot API Gateway



Express Gateway



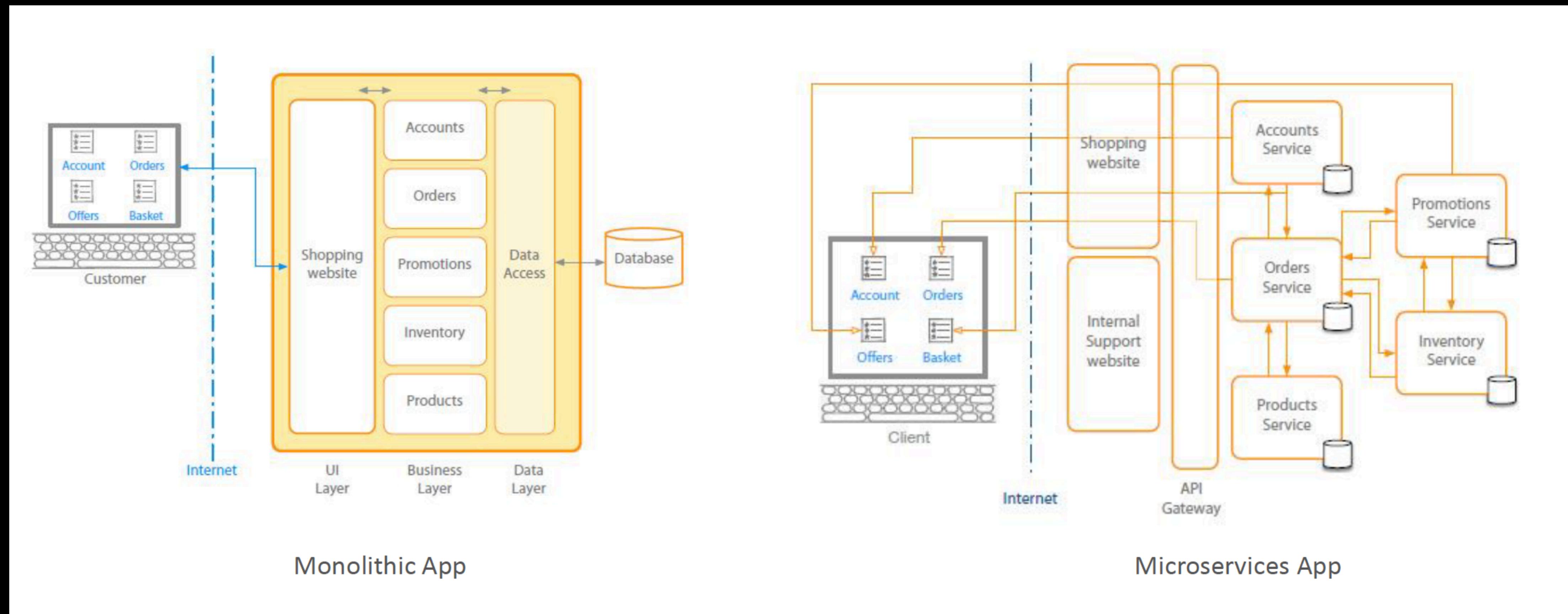
LoopBack API

Agenda

- ¿Por qué micro servicios?
- Refactorizando el Monolito
- Implementando nuevas características como servicios
- Extrayendo módulos en servicios
- Tecnologías para micro servicios
- Patrones de Diseño

Patrones de Diseño

Migrando de monolito a micro servicios



Patrones de Diseño para micro servicios

Decomposition Patterns

- Decompose by Business Capability
- Decompose by Subdomain
- Strangler Pattern

Integration Patterns

- API Gateway Pattern
- Aggregator Pattern
- Client-Side UI Composition Pattern

Database Patterns

- Database per Service
- Shared Database per Service
- CQRS Pattern
- Saga Pattern

Deployment Patterns

- Multiple service instances per host
- Service instance per host
- Service instance per VM
- Service instance per Container
- Serverless deployment

Observability Patterns

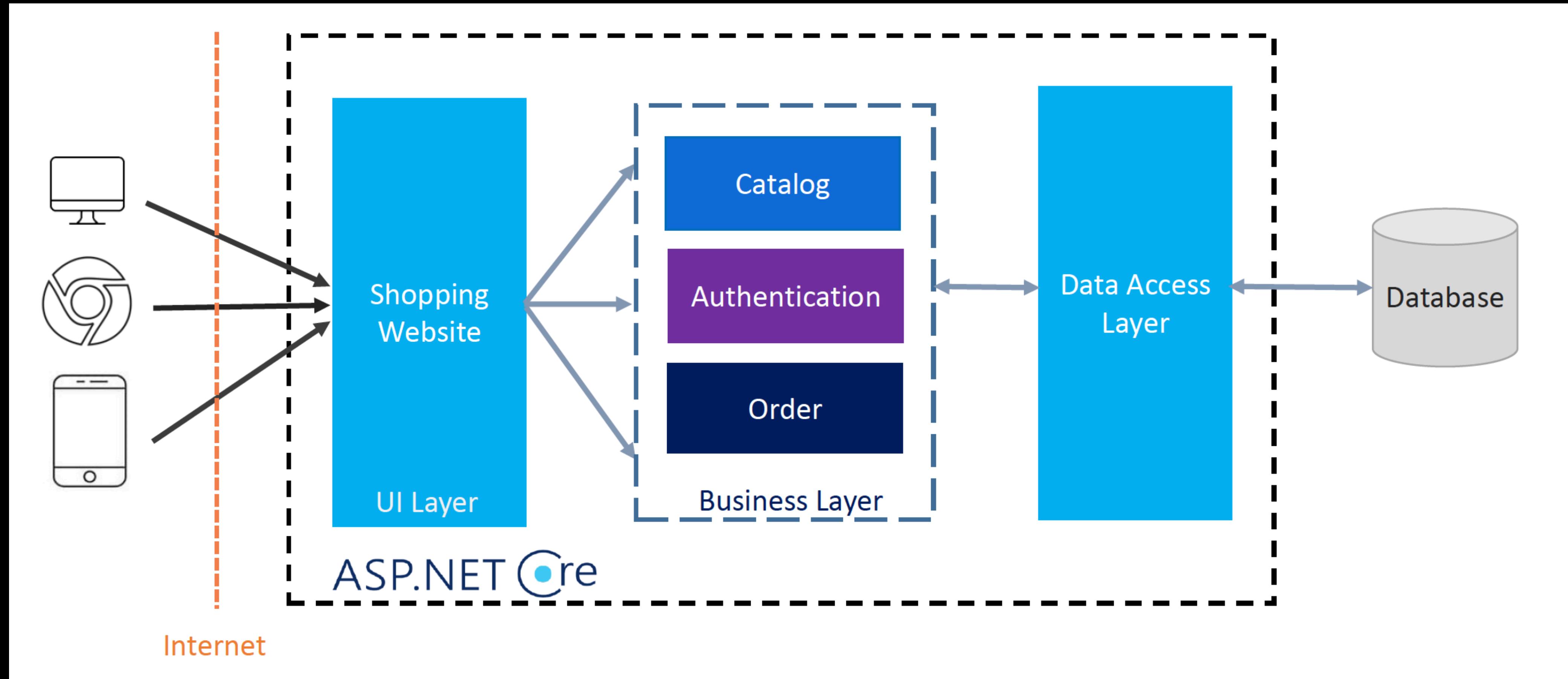
- Log Aggregation
- Performance Metrics
- Distributed Tracing
- Health Check

Cross-Cutting Concern Patterns

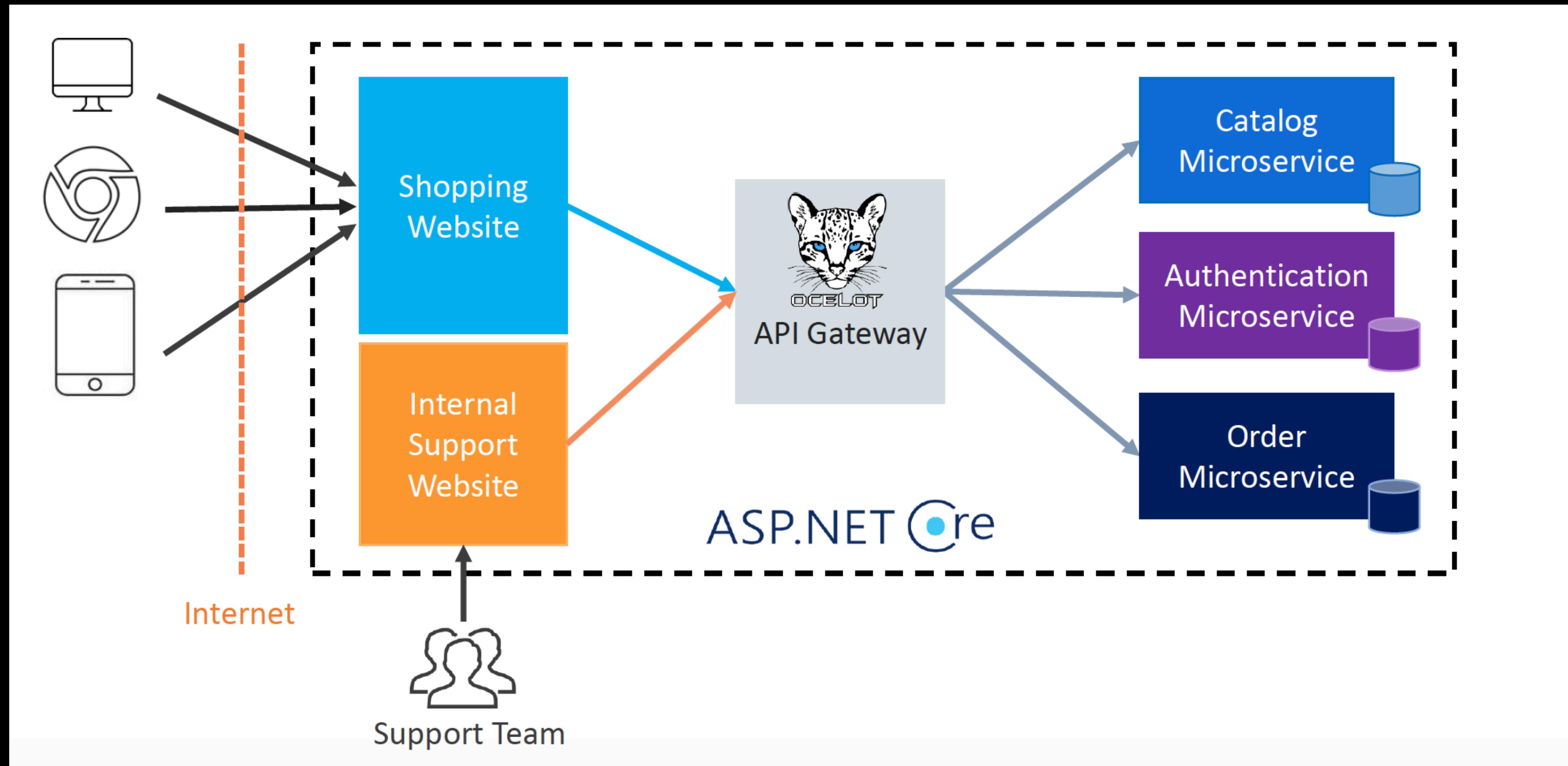
- Externalized Configuration
- Service Discovery Pattern
- Circuit Breaker Pattern

Construyendo micro servicios con ASP .NET Core

N-Layer APP



N-Layer App a Micro servicios



API Gateways de Micro servicios para .NET



Azure API Management



Ocelot API Gateway

Necesidad de un API Gateway

- Ruteo del tráfico
- Endpoint expuesto unificado
- Composición de API
- Caching
- Logging
- Authentication
- Authorization
- Balanceo de Carga
- Service Discovery

Demo de Microservicios



Ocelot API Gateway

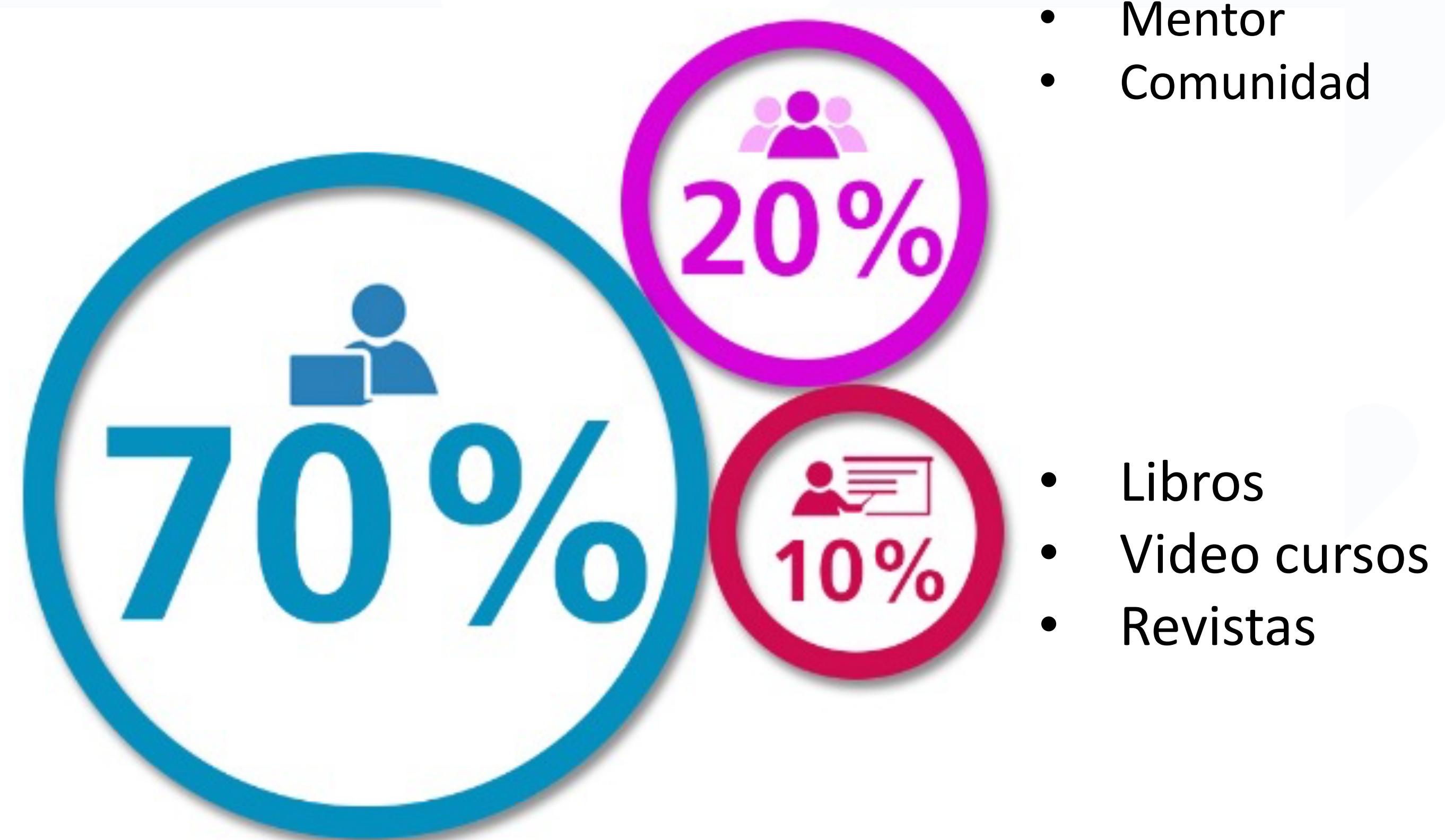
- API Gateway open source para la plataforma .NET
- Ligero y se ejecuta en plataformas que soportan ASP .NET Core
- Provee todas las características de un API Gateway
- Usado por el mismo Microsoft en sus proyectos

Demo con OCELOT



Método de Aprendizaje

- Hazlo tu mismo
- Úsalo en proyectos



Gracias

¿Alguna pregunta?

www.joedayz.pe

 @joedayz



<http://github.com/joedayz>