Ingeniería de Software III

Diseño de Sistemas Distribuidos

Parte 2

Objetivo

- Diseño de Sistemas Distribuidos
- Patrones de Software

Diseño de Sistemas

- De alto nivel -> arquitectura
- De bajo nivel -> diseño

Patrones de Software

- Definición
  - "Soluciones conocidas y reusables a problemas recurrentes en el software"

Patrones de Diseño de Software

- Clases
- Resuelven un problema punctual

- Ejemplos:
  - AbstractFactory
  - State
  - Strategy
  - Proxy
  - Monads
  - Etc.

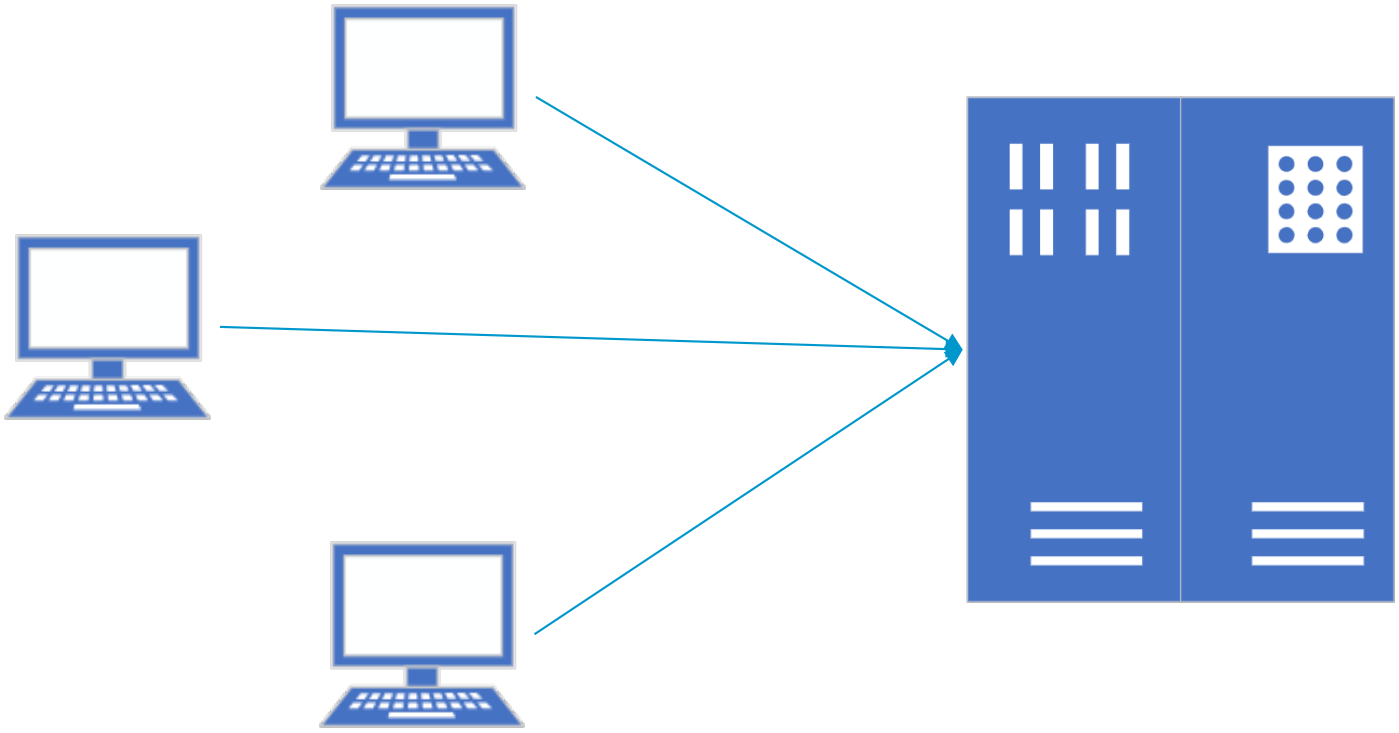
Patrones de Arquitectura de Software

- Componentes
- Entender como encajan las principales partes
- Como fluye la información
- Otros aspectos estructurales

- Ejemplos:
  - En capas
  - Service Oriented
  - Microservices
  - Microkernel

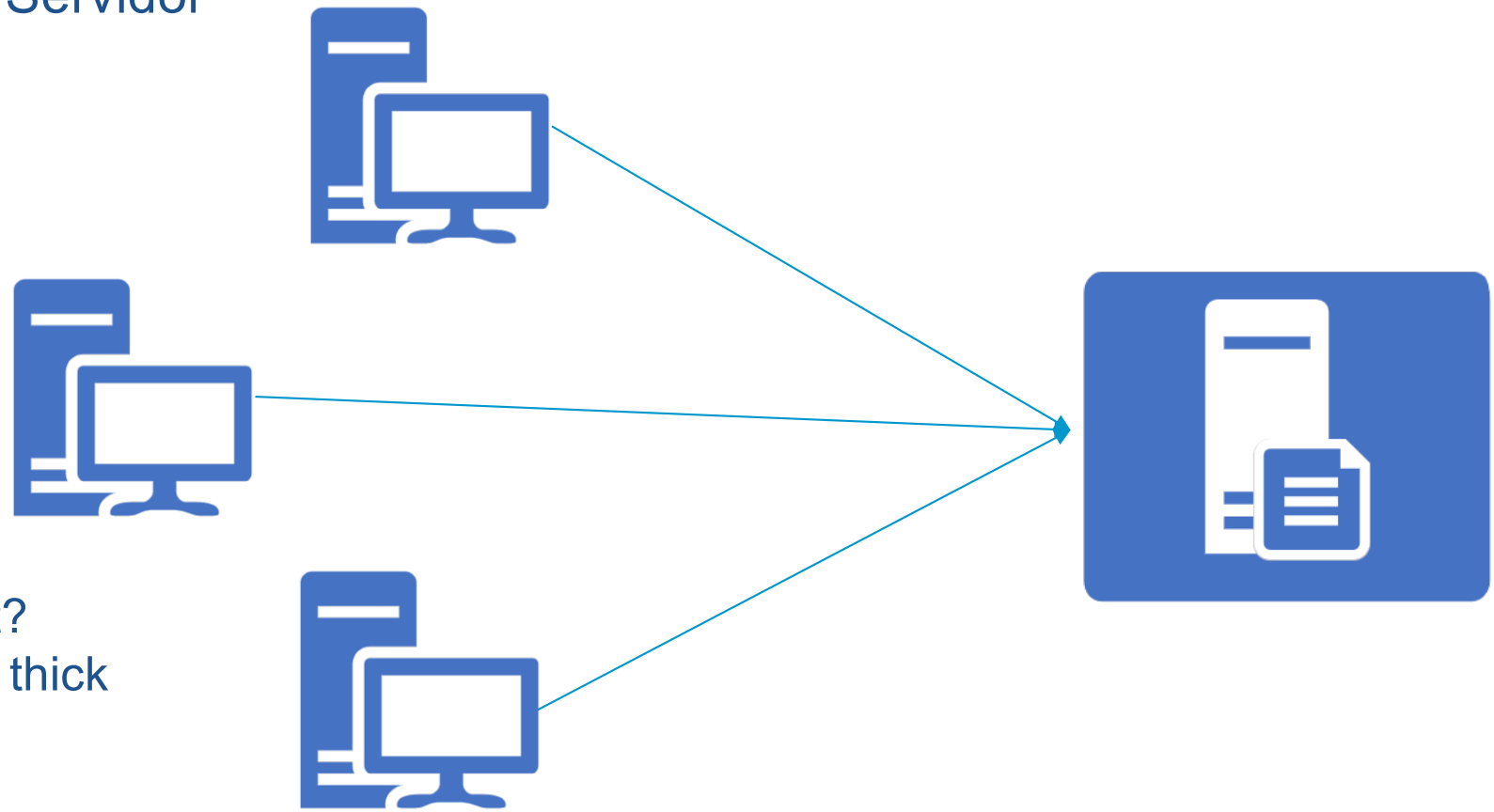
Evolución de la Arquitectura de los Sistemas

- En el comienzo…
  - Mainframe
  - Terminales bobas



Evolución de la Arquitectura de los Sistemas

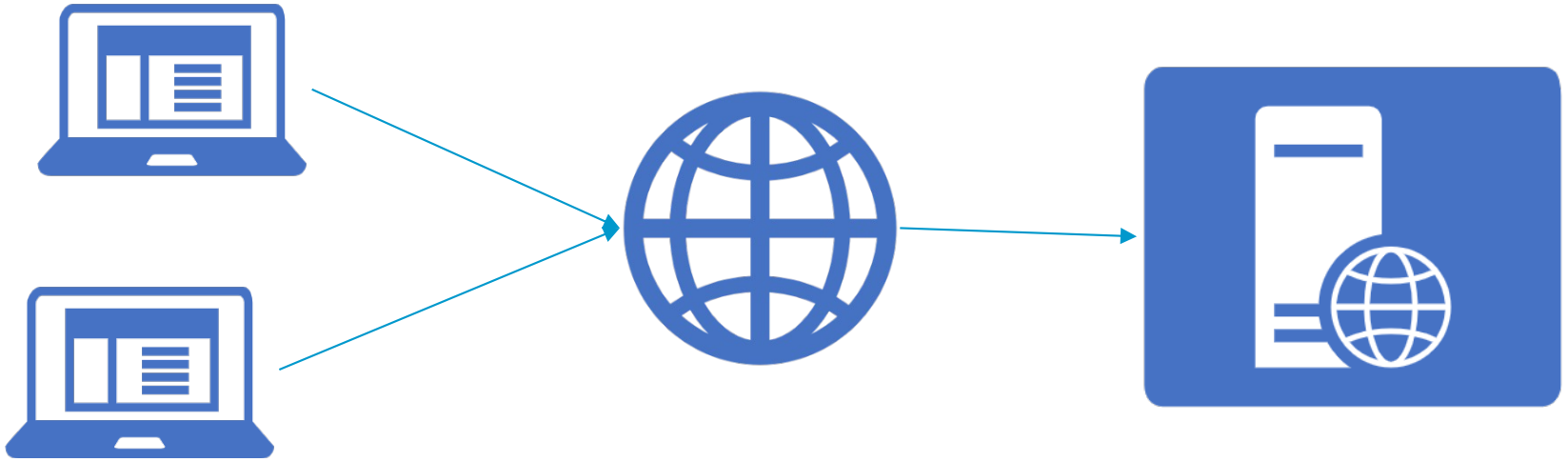
- Aparece la computadora personal
  - Cliente
  - Servidor



Client?
thin o thick

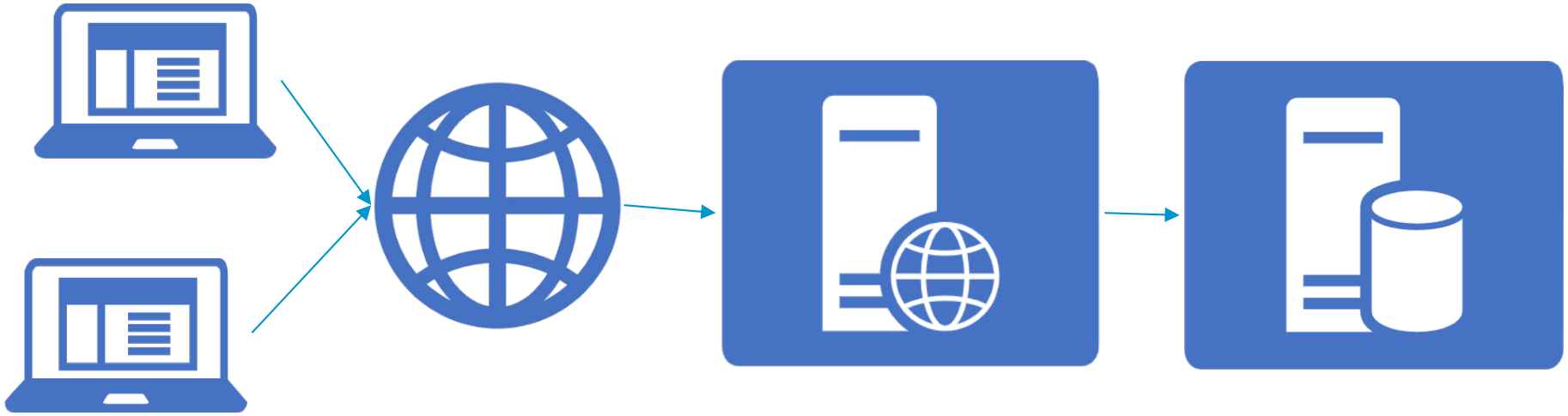
Evolución de la Arquitectura de los Sistemas

- Aparece Internet
  - Cliente
  - Servidor



Evolución de la Arquitectura de los Sistemas

- Dividir responsabilidades y distribuirlas en diferentes nodos para escalar

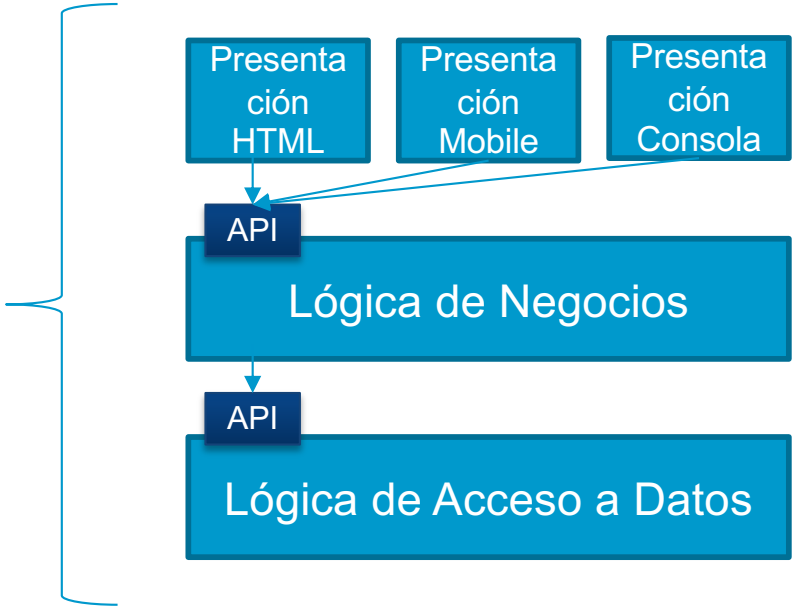


Evolución de la Arquitectura de los Sistemas

- Arquitectura en capas (layers)
  - Las capas de abajo proveen servicios a las capas de arriba
  - Los pedidos fluyen de arriba hacia abajo
  - Cada capa puede ser reemplazada o modificada sin afectar toda la arquitecrtura

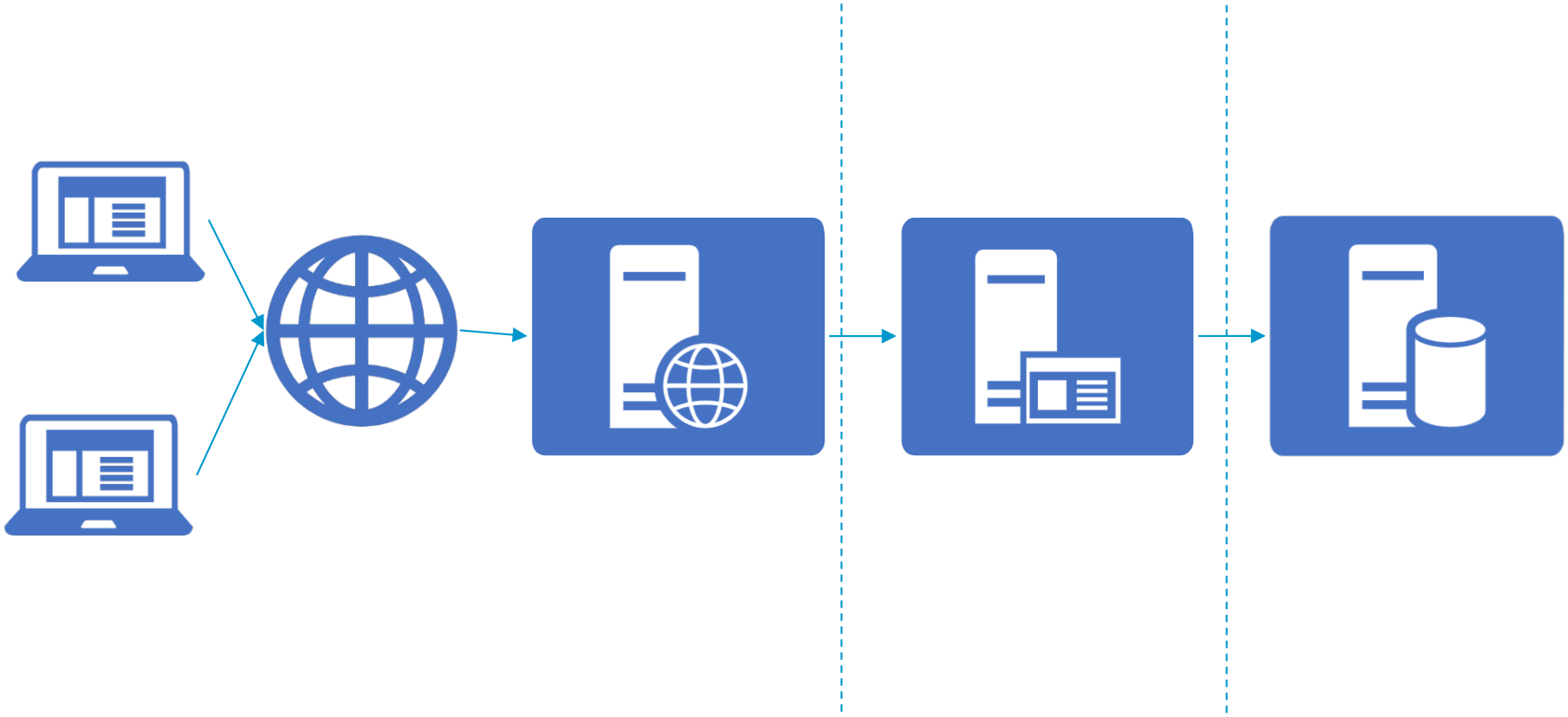
Evolución de la Arquitectura de los Sistemas

- Arquitectura en capas (layers)
  - Las capas de abajo proveen servicios a las capas de arriba
  - Los pedidos fluyen de arriba hacia abajo
  - Cada capa puede ser reemplazada o modificada sin afectar toda la arquitecrtura



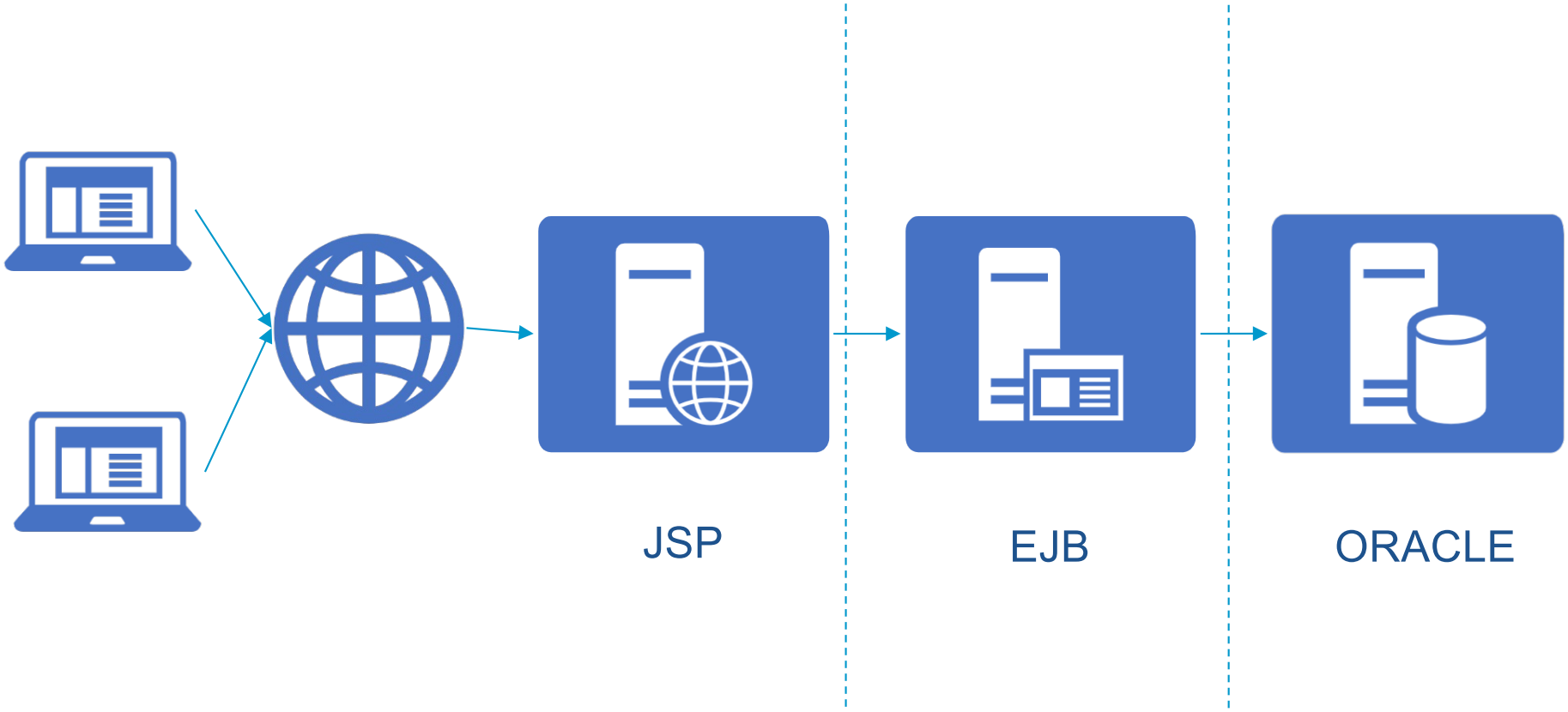
Evolución de la Arquitectura de los Sistemas

- Arquitectura N-Capas (N-Tier)



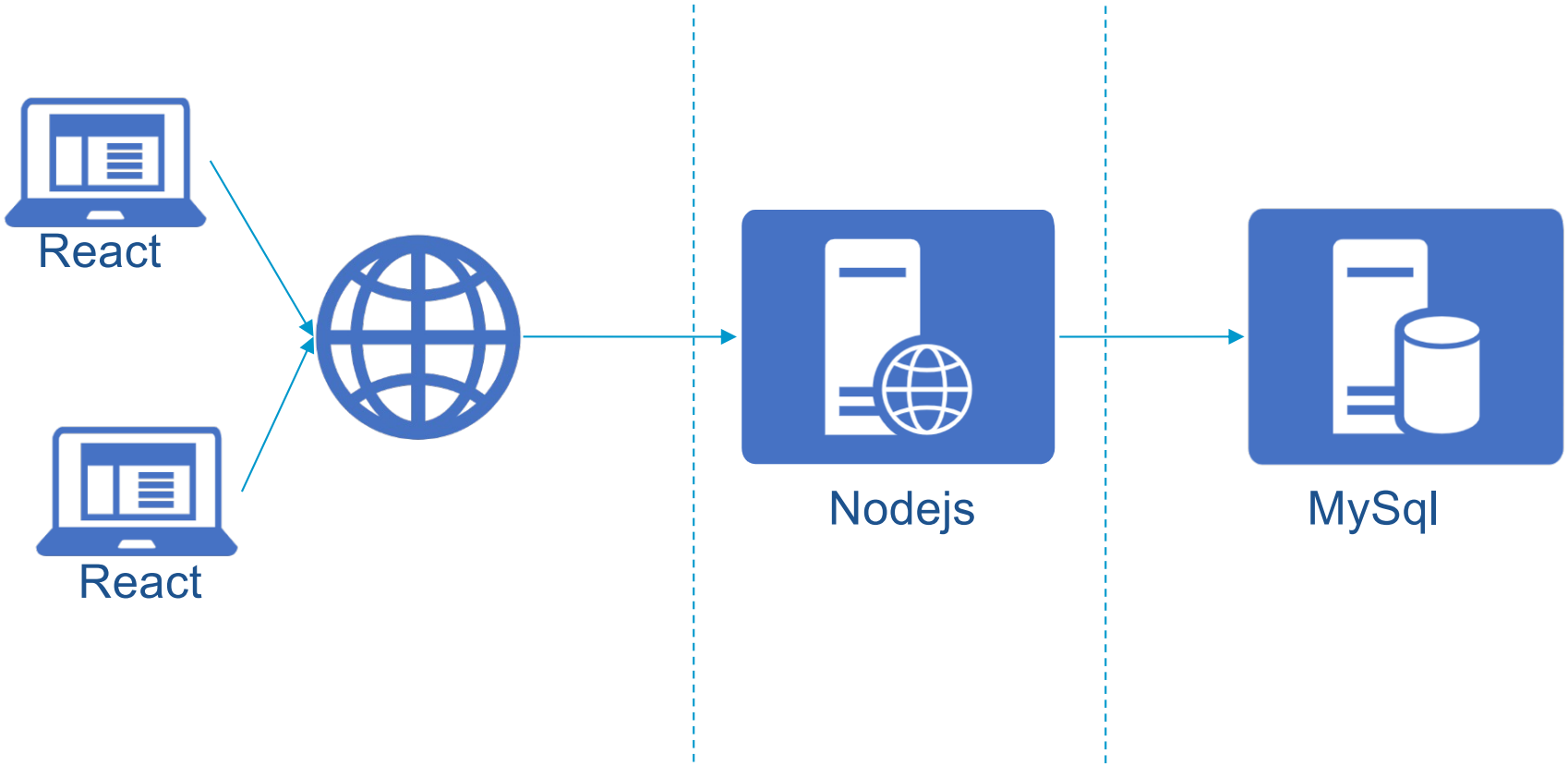
Evolución de la Arquitectura de los Sistemas

- Arquitectura N-Capas (N-Tier)



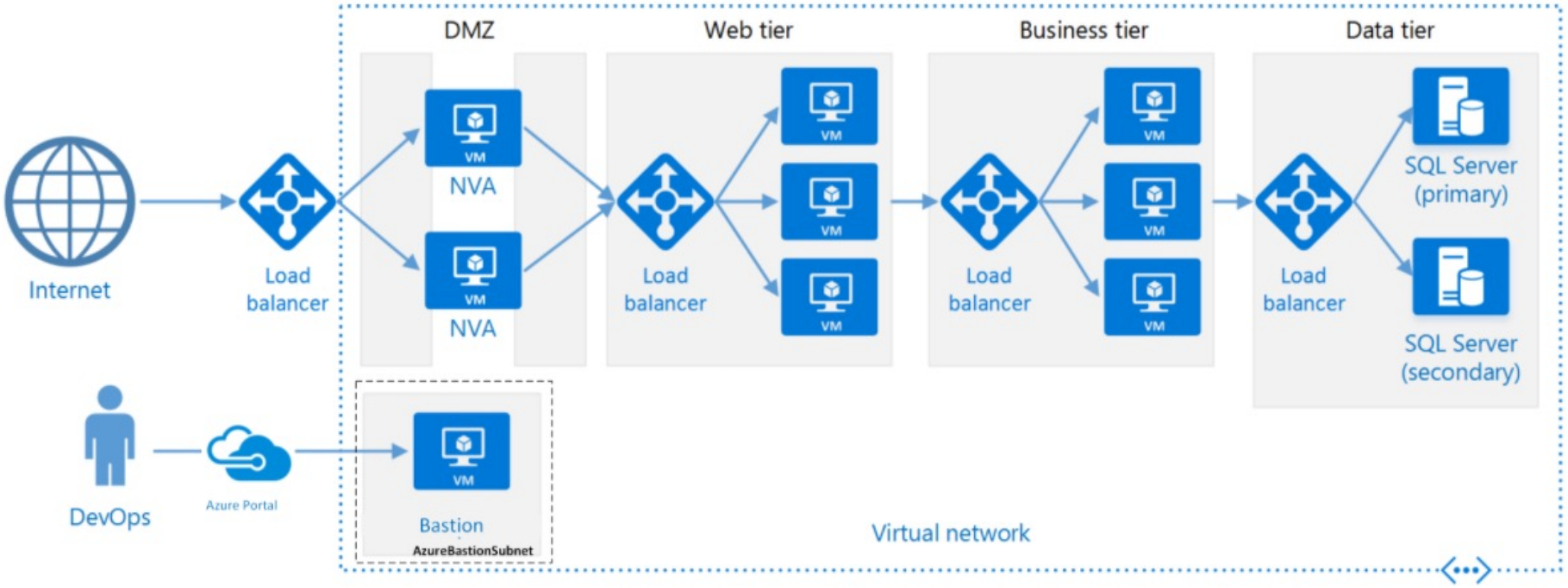
Evolución de la Arquitectura de los Sistemas

- Arquitectura N-Capas (N-Tier) moderna



Evolución de la Arquitectura de los Sistemas

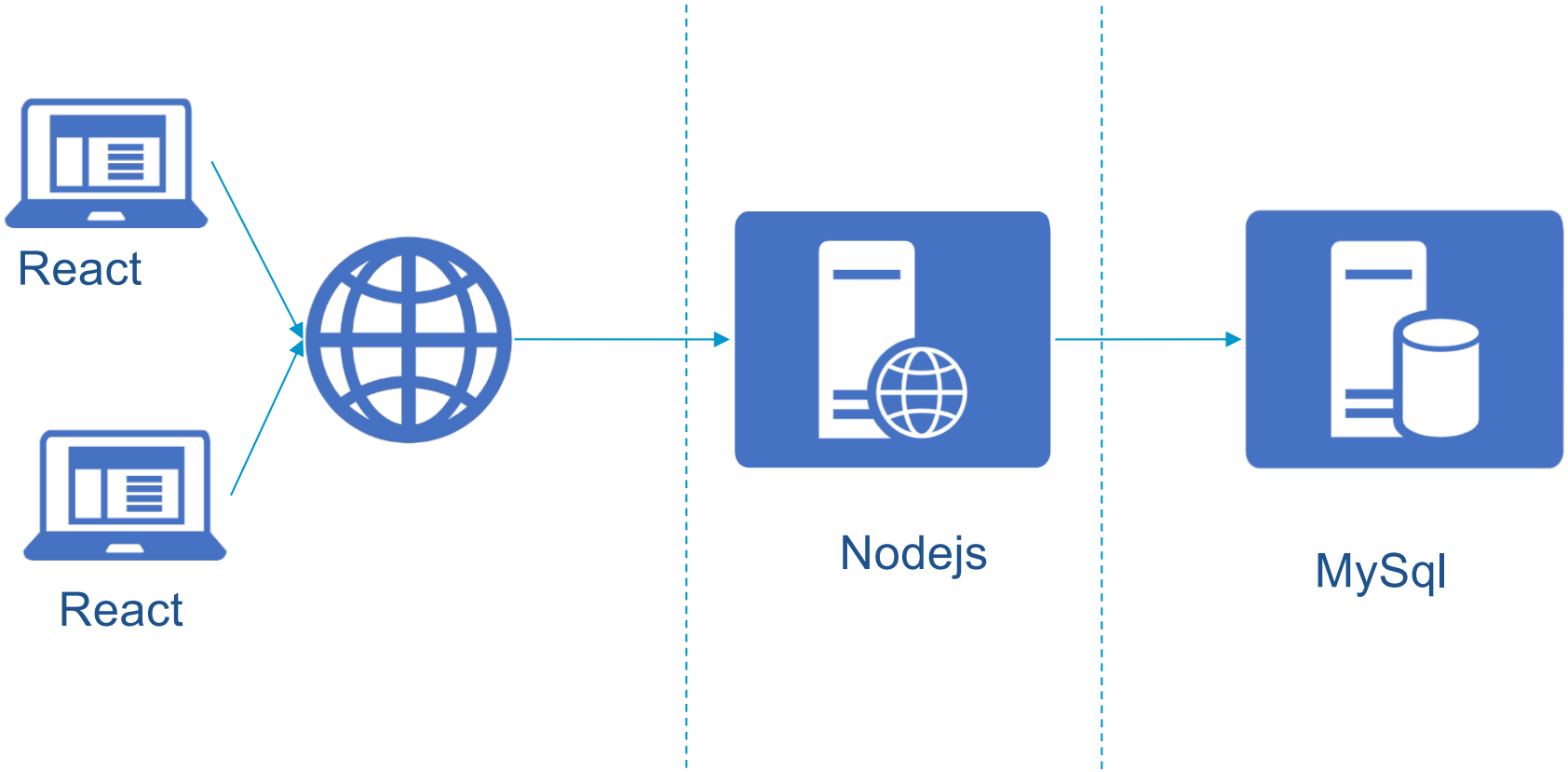
- Arquitectura N-Capas (N-Tier) moderna



https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/n-tier

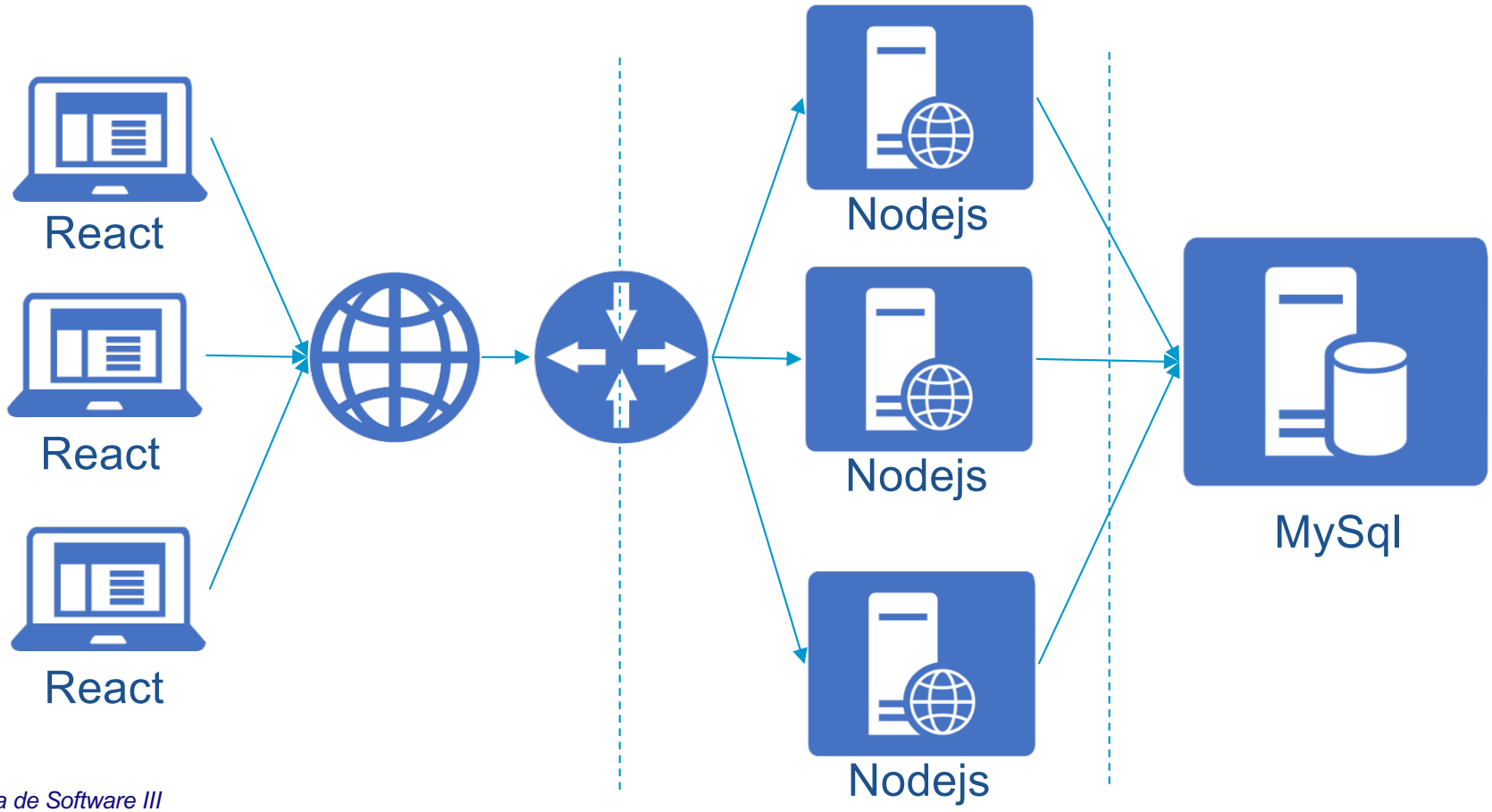
Evolución de la Arquitectura de los Sistemas

- Arquitectura N-Capas (N-Tier) moderna
  - Desventaja -> Monolith



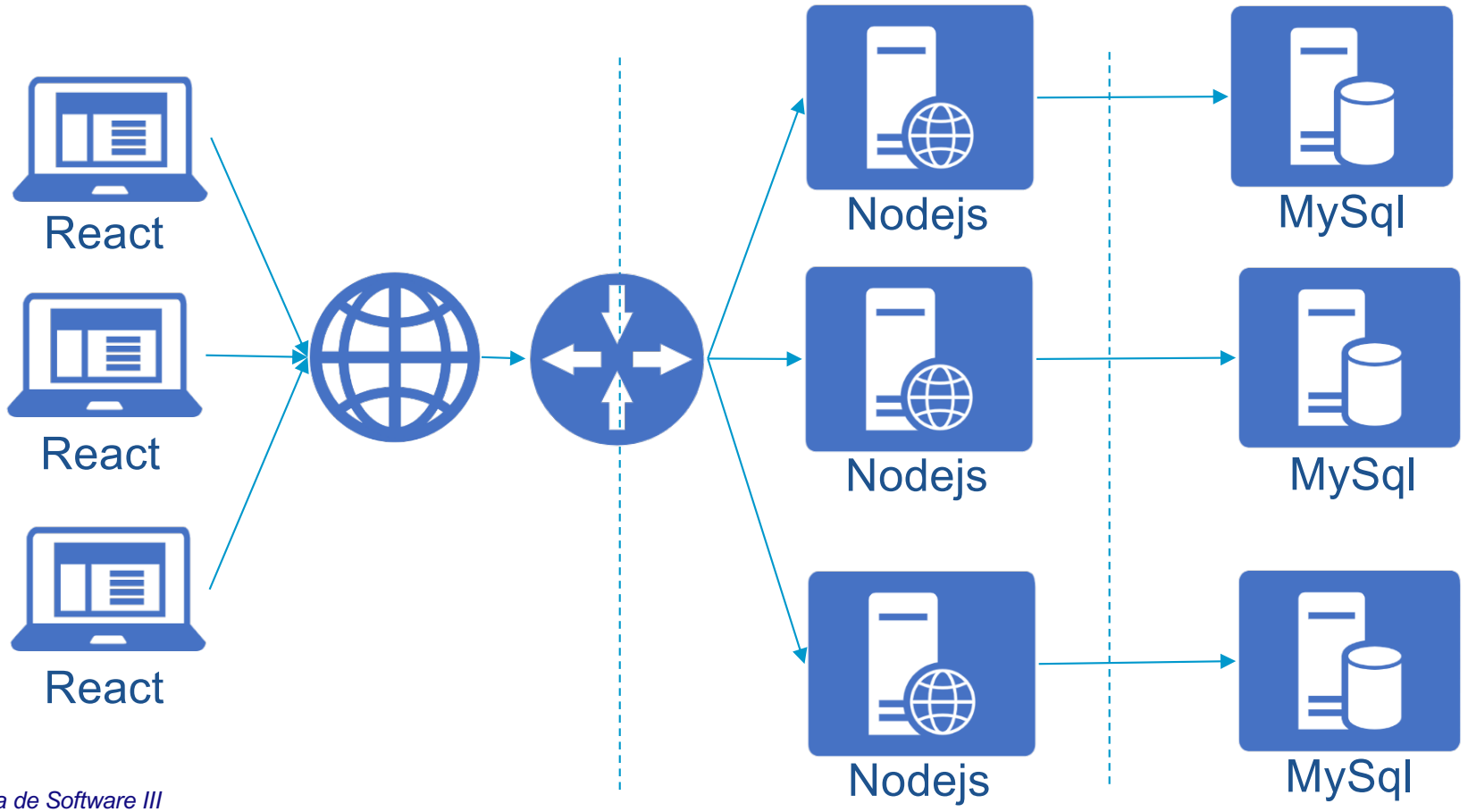
Evolución de la Arquitectura de los Sistemas

- Arquitectura N-Capas (N-Tier) moderna
  - Escalar clonando la capa de negocios (en el eje de las X)



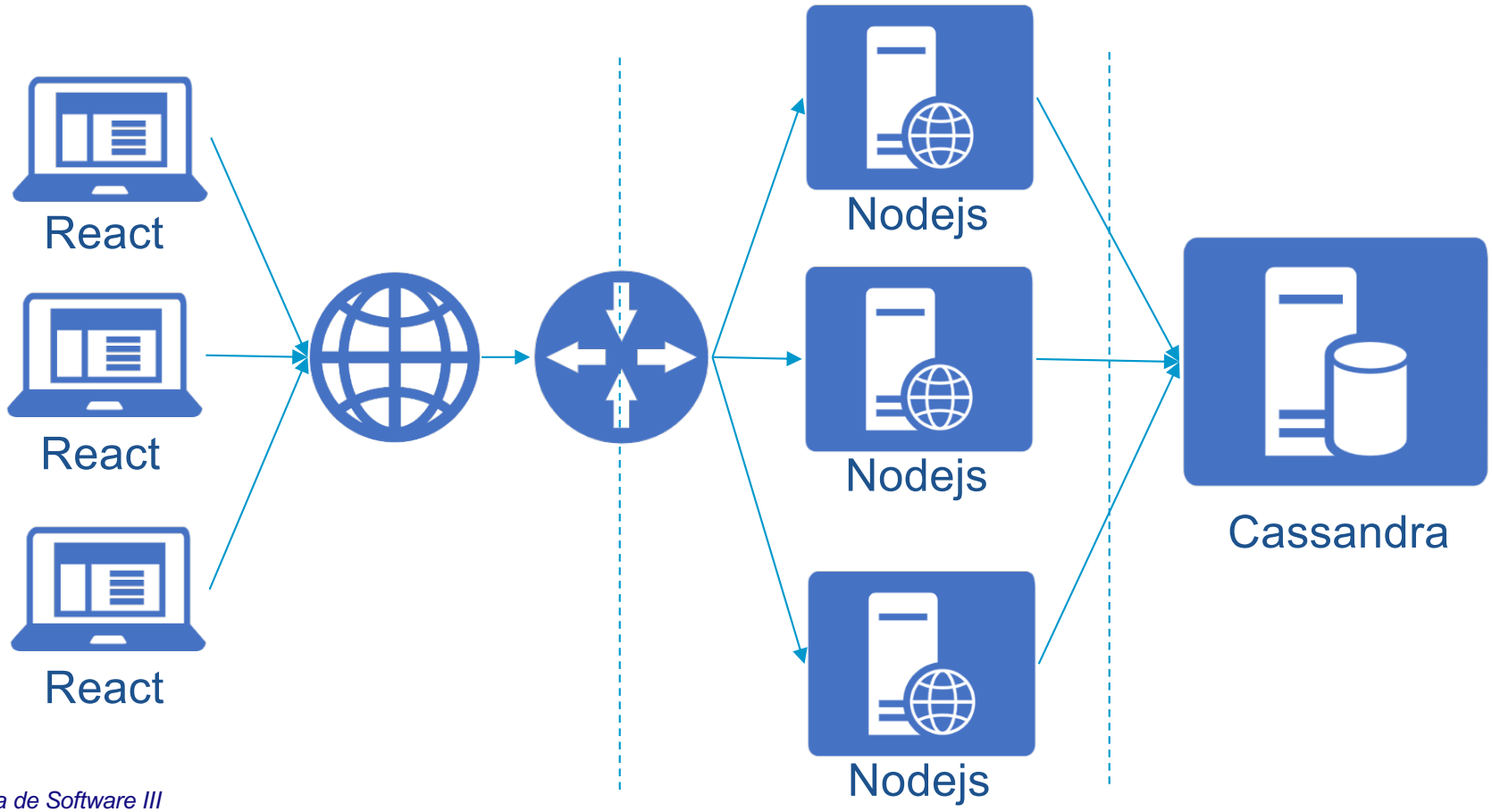
Evolución de la Arquitectura de los Sistemas

- Arquitectura N-Capas (N-Tier) moderna
  - Escalar la capa de datos haciendo sharding (en el eje de las z)



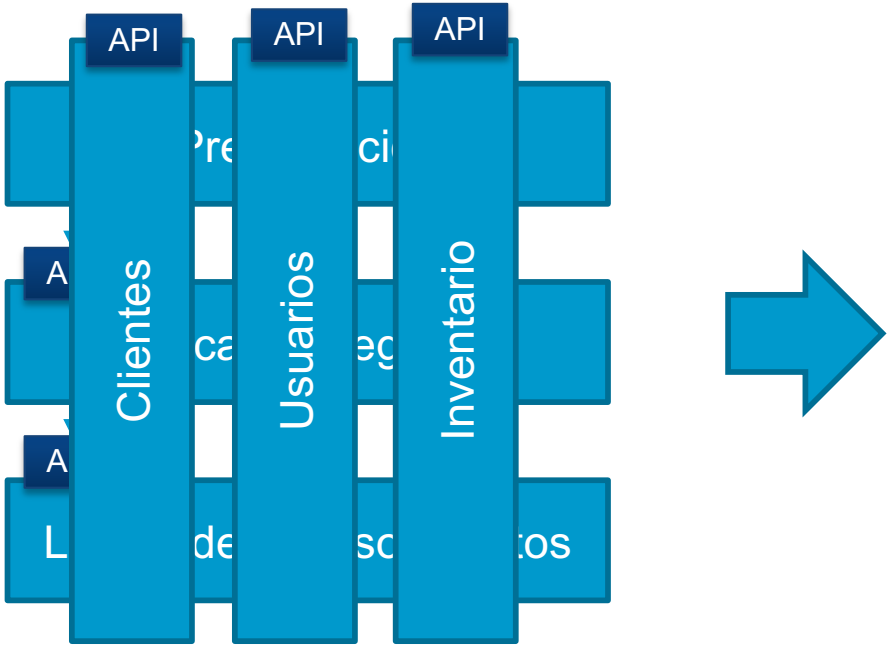
Evolución de la Arquitectura de los Sistemas

- Arquitectura N-Capas (N-Tier) moderna
  - Escalar la capa de datos con una base de datos distribuida



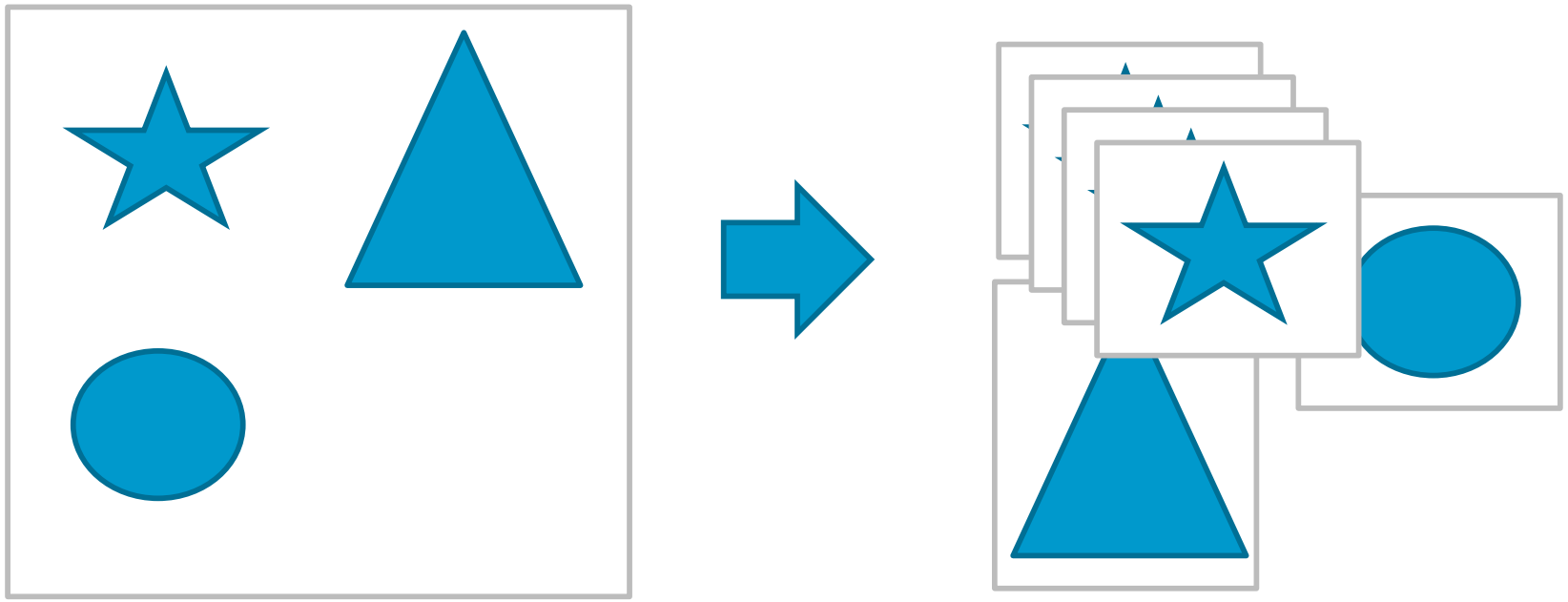
Evolución de la Arquitectura de los Sistemas

- Rompiendo el monolito
  - ¿Por qué?



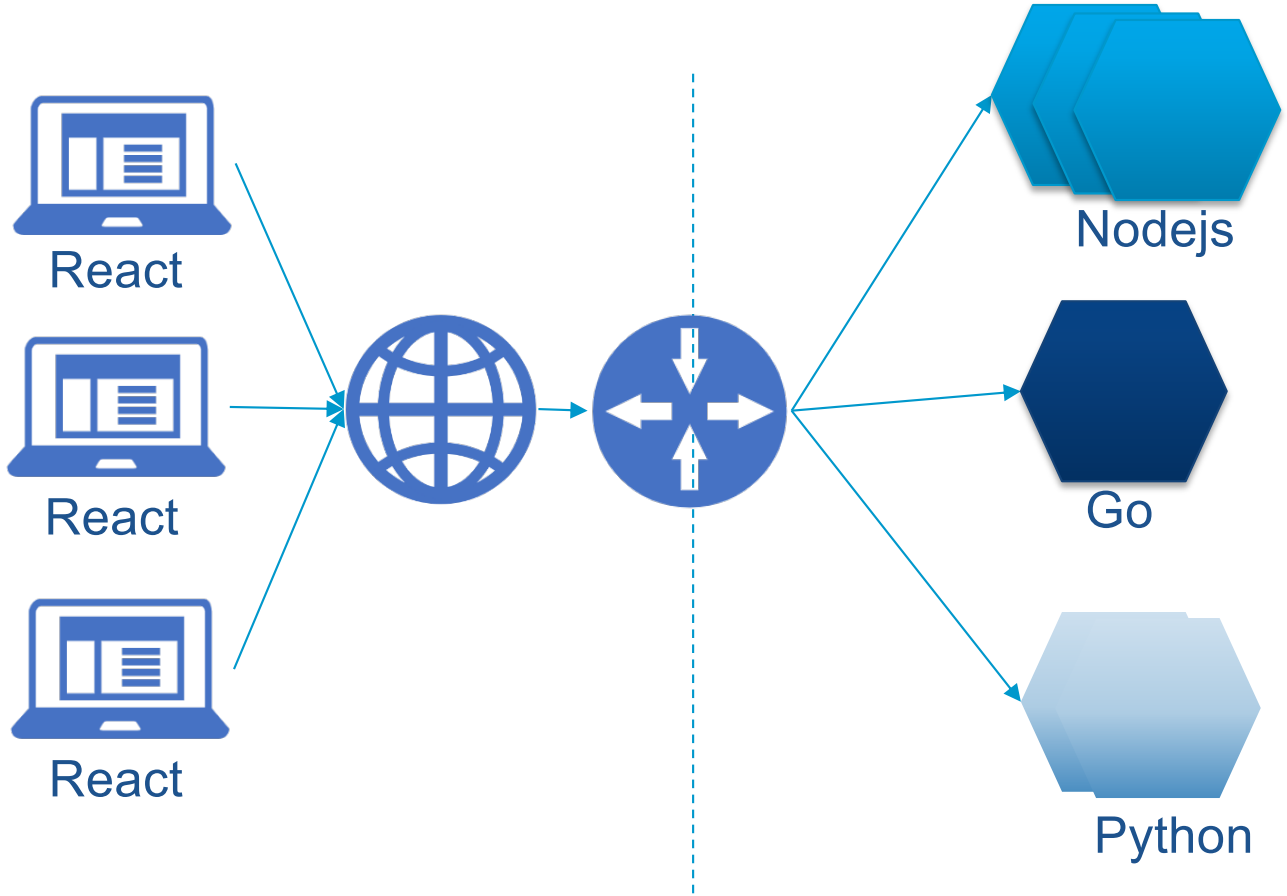
Evolución de la Arquitectura de los Sistemas

- Rompiendo el monolito
  - ¿Por qué?



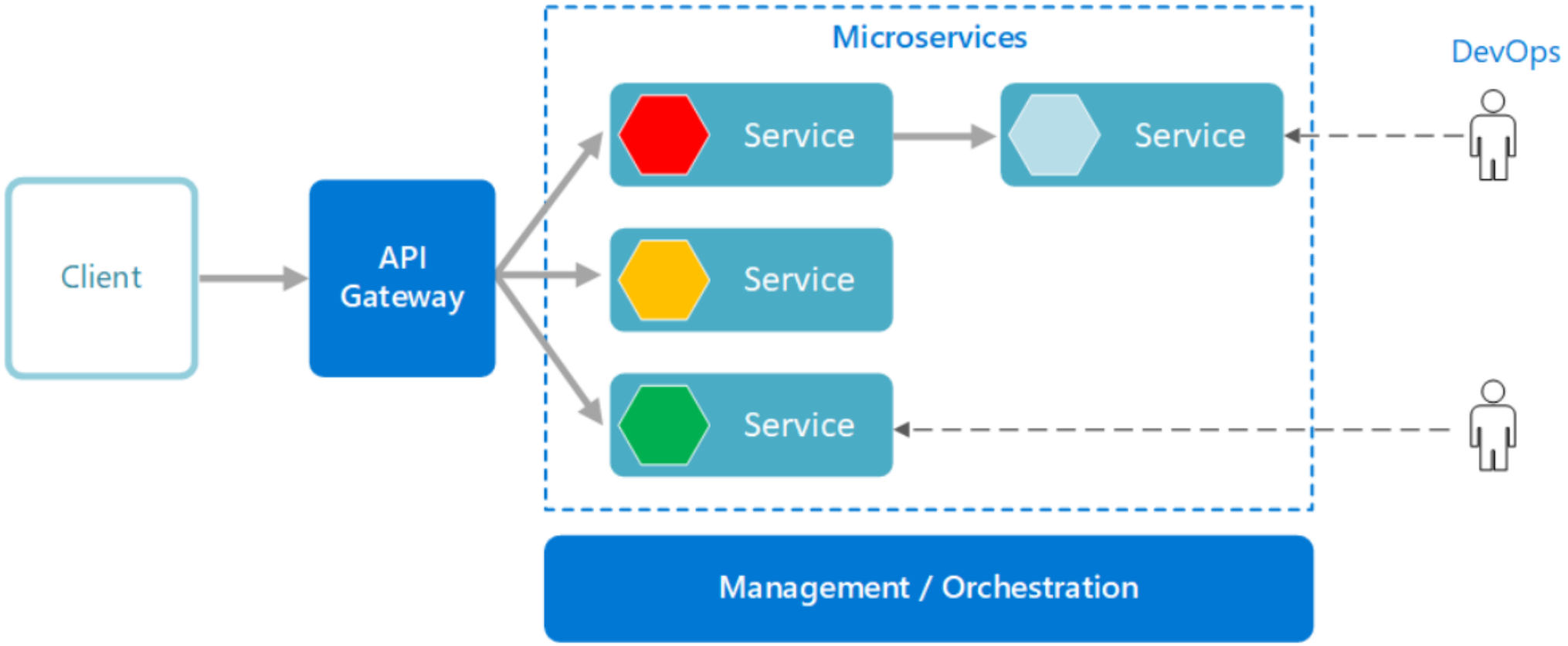
Evolución de la Arquitectura de los Sistemas

- Arquitectura Microservicios
  - Escalar por funcionalidad (en el eje de las Y)



Evolución de la Arquitectura de los Sistemas

- Arquitectura Microservicios

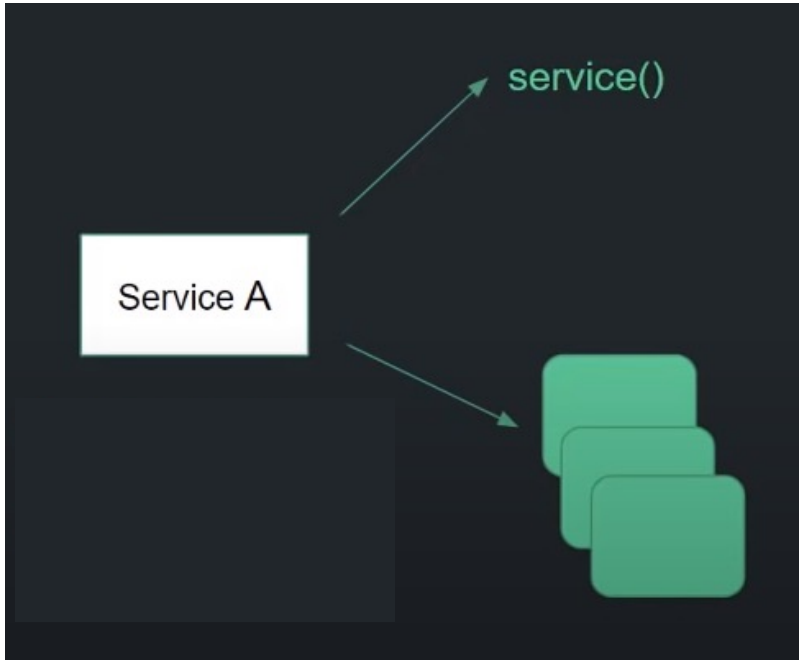


https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

Microservicios

- ¿Qué es un microservicio?

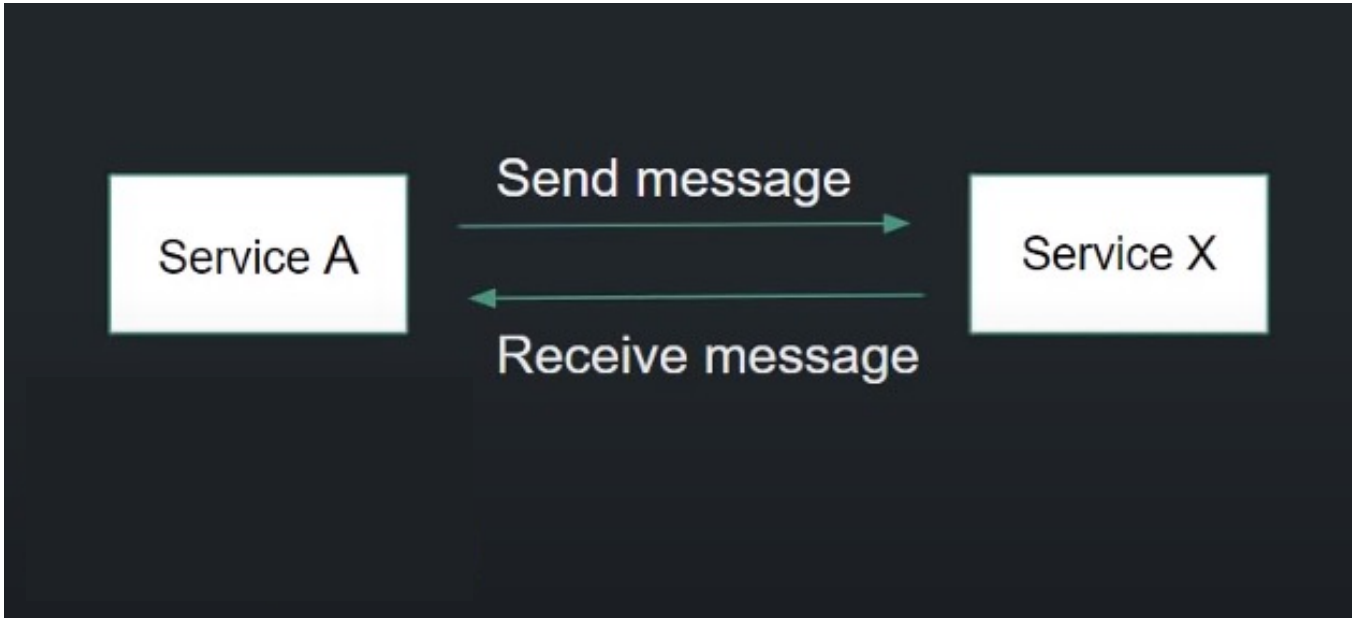
Microservices are small in size, messaging-enabled, bounded by contexts, autonomously developed, independently deployable, decentralized and built and released with automated processes.



Microservicios

- ¿Qué es un microservicio?

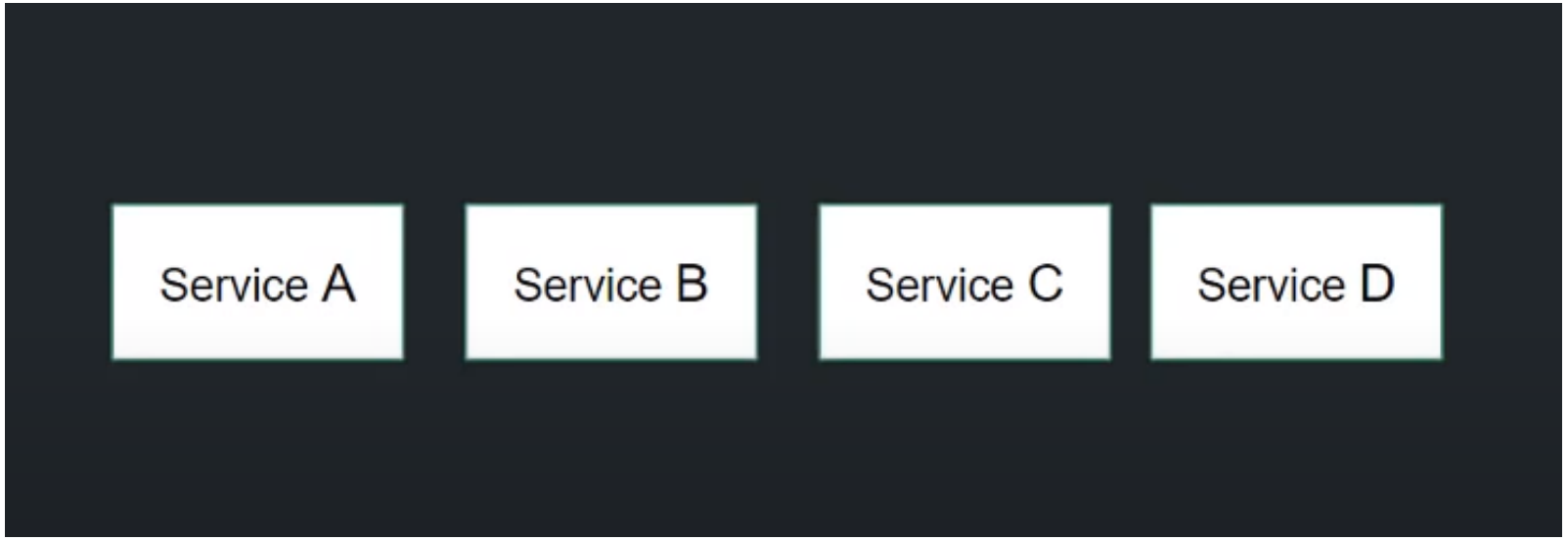
Microservices are small in size, messaging-enabled, bounded by contexts, autonomously developed, independently deployable, decentralized and built and released with automated processes.



Microservicios

- ¿Qué es un microservicio?

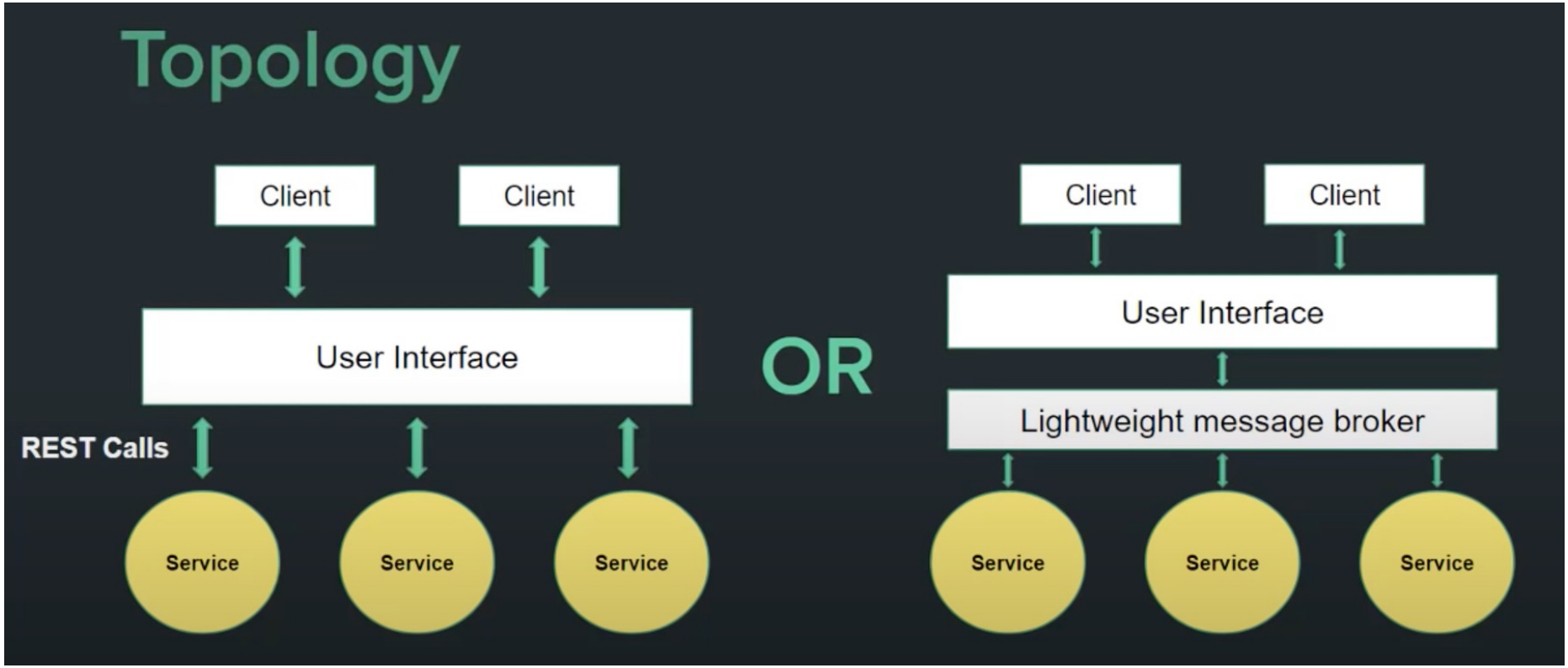
Microservices are small in size, messaging-enabled, bounded by contexts, autonomously developed, independently deployable, decentralized and built and released with automated processes.



Definiciones al momento de diseño

- Estructura
  - Usando DDD
- Comunicación
  - Request/Response, Messaged based
- Coordinación
  - Orchestration vs Choreography

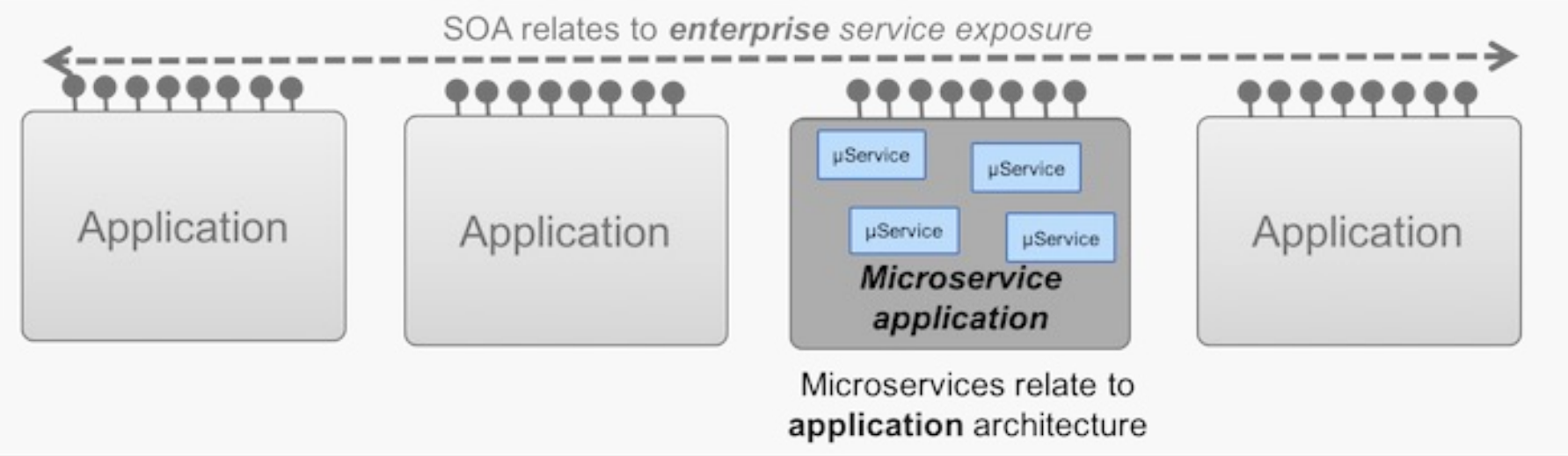
Topología


Topology
Client
Client
User Interface
REST Calls
Service
Service
Service
OR
Client
Client
User Interface
Lightweight message broker
Service
Service
Service

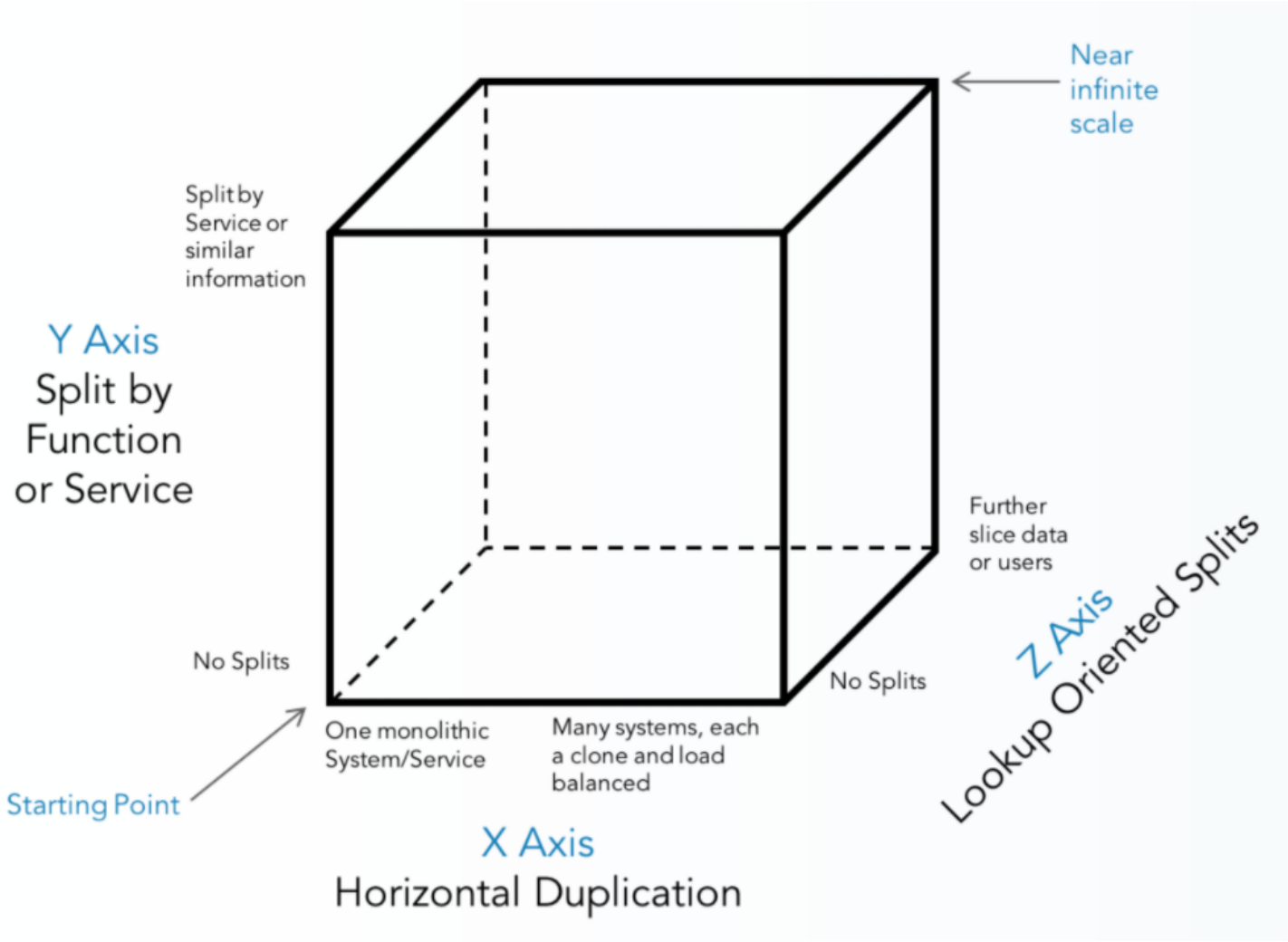
Microservicios

- Trade-offs
  - Los servicios son más simples
  - La arquitectura se vuelve más compleja
- Complejidad manejada con
  - Herramientas
  - Automatización
- Monitoreo

SOA vs Microservices



Cubo de escalabilidad



Sistemas Distribuidos: Patrones de Diseño

- Retry
- Circuit Breaker
- Bulkhead
- Throttling
- Gateway Aggregation
- Saga

Sistemas Distribuidos: Patrones de Diseño

UNIVERSIDAD CATÓLICA DE CÓRDOBA
Universidad Jesuita

Retry:

- Intención/Propósito:
  - Permite que una aplicación maneje fallas transitorias de conexión a procesos o recursos remotos de manera transparente
  - Mejora la estabilidad de la aplicación

Sistemas Distribuidos: Patrones de Diseño

Retry:

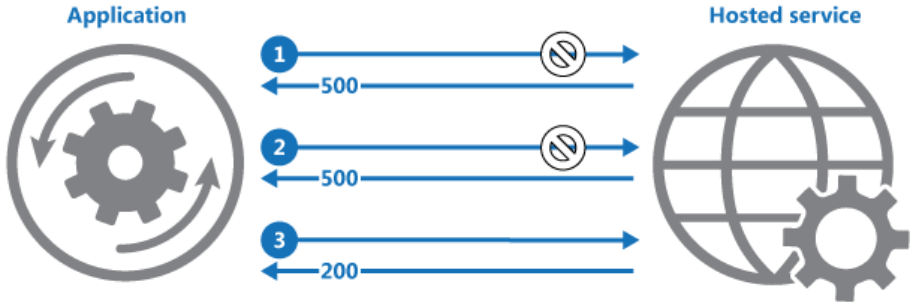
- Problema:
  - Las aplicaciones distribuidas tienen que ser tolerantes a fallos transitorios esperables en el entorno
  - Perdidas temporales de la comunicación
  - Falta de respuesta de un recurso temporalmente ocupado

- Estos fallos temporales tipicamente se corrijen solos después de un tiempo corto.

Sistemas Distribuidos: Patrones de Diseño

Retry:

- Solución:
  - Manejar las fallas de usando las siguientes estrategias:
    - Cancelar:
      - Si el fallo no es transitorio. Por ejemplo: fallo de autenticación
    - Reintentar:
      - fallo inusual o raro, tal vez causado por un mensaje corrupto o perdido. En este caso se puede reintentar inmediatamente
    - Reintentar después de un retardo:
      - Fallo causado por un lugar donde se espera problemas de conectividad o problemas de recursos ocupados temporalmente. En este casi se puede demorar un tiempo y luego reintentar para darle tiempo al recurso a recuperarse.



1: Application invokes operation on hosted service. The request fails, and the service host responds with HTTP response code 500 (internal server error).
2: Application waits for a short interval and tries again. The request still fails with HTTP response code 500.
3: Application waits for a longer interval and tries again. The request succeeds with HTTP response code 200 (OK).

Sistemas Distribuidos: Patrones de Diseño

Circuit Breaker:

- Intención/Propósito:
  - Manejar fallas de conección a un proceso o recurso remoto que puede requerir una cantidad variable de tiempo para recuperarse
  - Mejorar la estabilidad de la aplicación

Sistemas Distribuidos: Patrones de Diseño

Circuit Breaker

- Problema:
  - Además de las fallas transitorias, pueden haber fallos debido a eventos no anticipados que requieran más tiempo para resolverse.
  - No tiene sentido seguir gastanto recursos en seguir intentando accedes a un servicio o recurso defectuoso o ocupado.
  - Evitar fallos en cascadas, por ejemplo por timeouts.

Sistemas Distribuidos: Patrones de Diseño

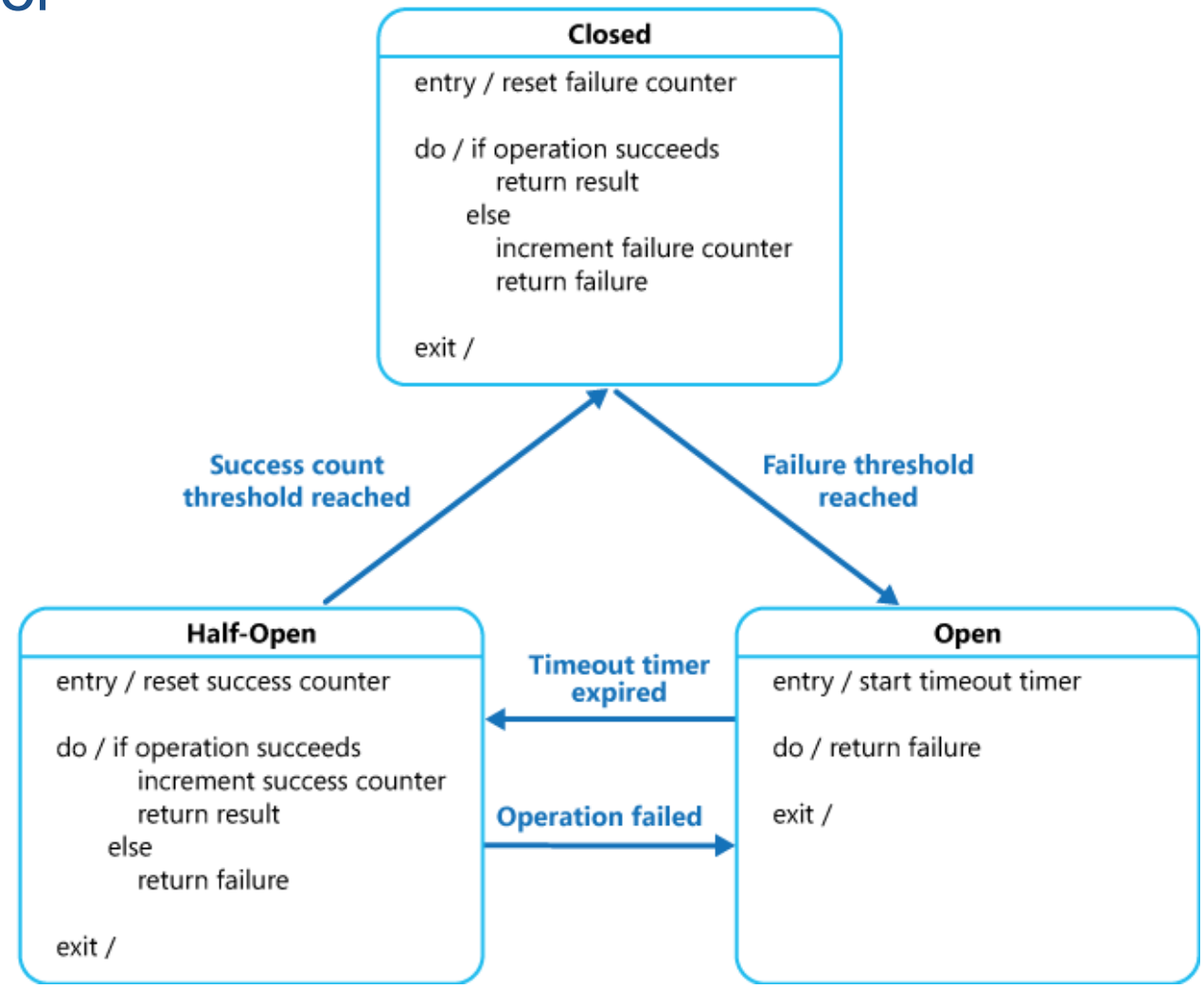
Circuit Breaker

- Solución:
  - Usar un proxy para las operaciones remotas que pueden fallar, implementado como una máquina de estado.
    - Cerrado:
      - Permite el paso de requests
      - Cuenta el nro de fallos
      - Si hay más fallos que un threshold predeterminado en un tiempo dado se abre el circuito
      - Se comienza un timer y cuando el timer expira se pasa a semi abierto
    - Abierto
      - En este estado todos los requests fallan de manera inmediata
      - Dando tiempo a que el recurso o proceso remoto se recupere
    - Medio abierto
      - Permite pasar un numero limitado de requests para testear si el problema está resuelto.
      - Si los requests son exitosos vuelve a Cerrado, de lo contratio vuelve a Abierto y reincia el timer

Sistemas Distribuidos: Patrones de Diseño

Circuit Breaker

- Solución:



Sistemas Distribuidos: Patrones de Diseño

Bulkhead:

- Intención/Propósito:
  - Aislar recursos usados para consumir diferentes servicios
  - Aislar clientes críticos de clientes comunes
  - Proteger la aplicación de fallos en cascada

Sistemas Distribuidos: Patrones de Diseño

UNIVERSIDAD CATÓLICA DE CÓRDOBA
Universidad Jesuita

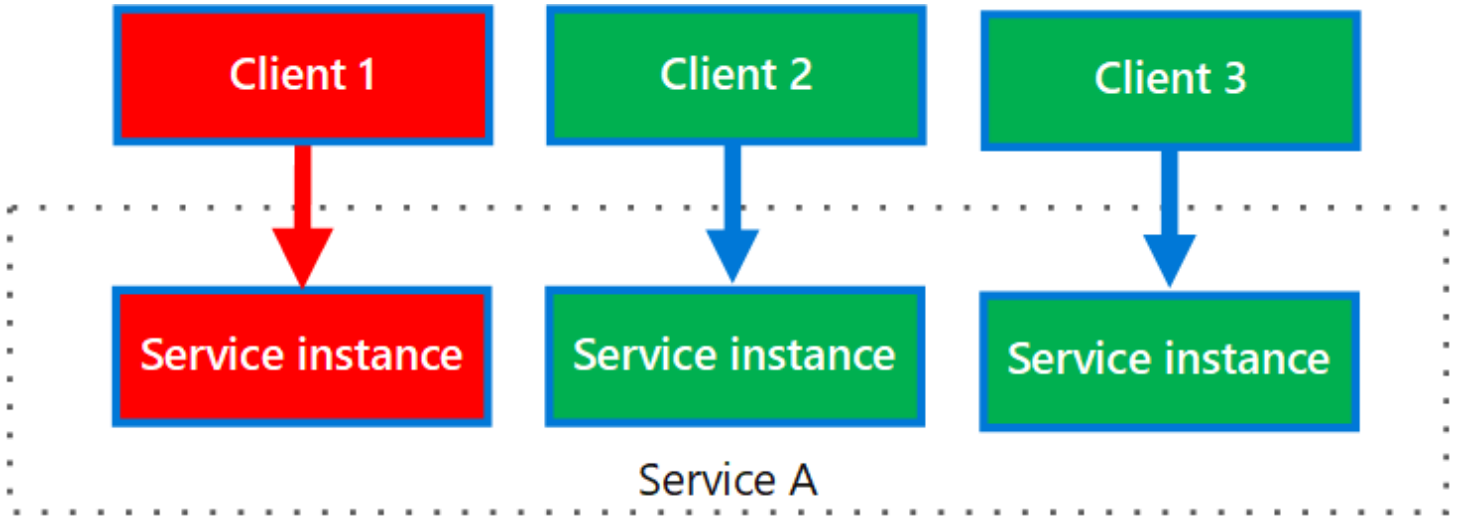
Bulkhead:

- Problema:
  - Tengo multiples servicios con multiples clientes.
  - Fallos o carga excesiva en un servicio afecta a todos los clientes.
  - Fallos en un servicio pueden consumir los recursos de un cliente haciendo fallar la comunicación con otros servicios (i.e. connection pools)

Sistemas Distribuidos: Patrones de Diseño

Bulkhead

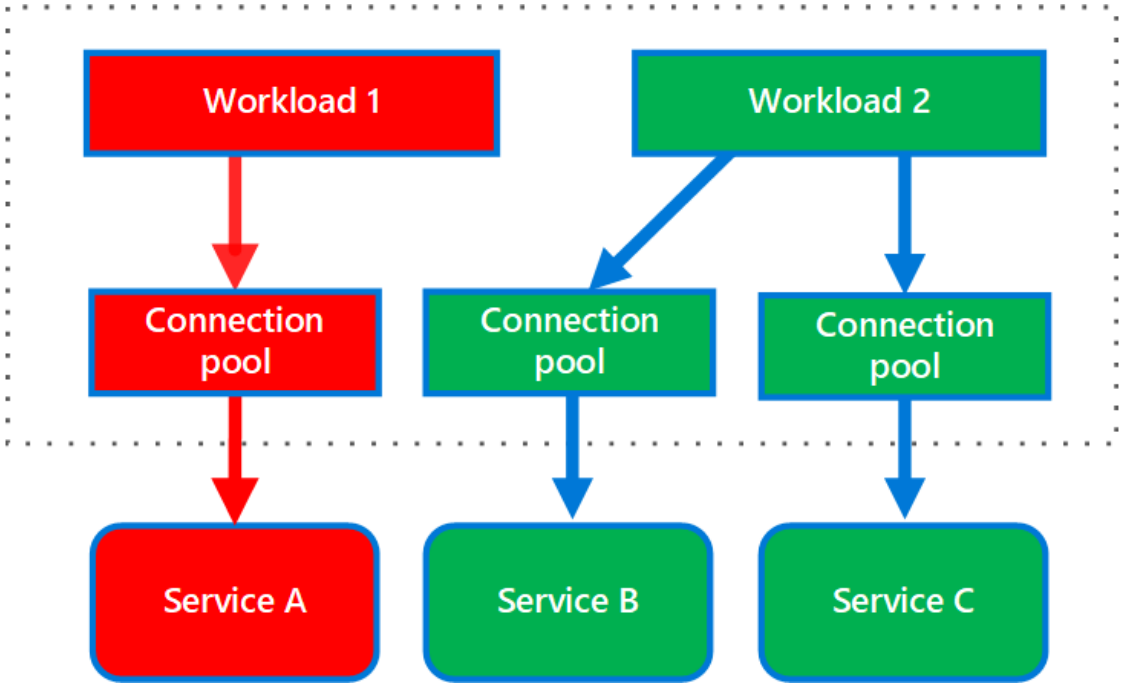
- Solución:
  - Aislar Servicios, particionarlos en diferentes grupos
  - Aislar clientes críticos de clientes comunes



Sistemas Distribuidos: Patrones de Diseño

Bulkhead

- Solución:
  - El cliente puede particionar sus recursos
  - Especialmente cuando puede seguir funcionando incluso si algunos sericios no responden



Sistemas Distribuidos: Patrones de Diseño

Throttling

- Intención/Propósito:
  - Controlar el consumo de los recursos de un cliente, servicio, etc
  - Permite seguir cumpliendo los SLAs incluso con altas demandas de algunos recursos
  - Manejar ráfagas de actividad

Sistemas Distribuidos: Patrones de Diseño

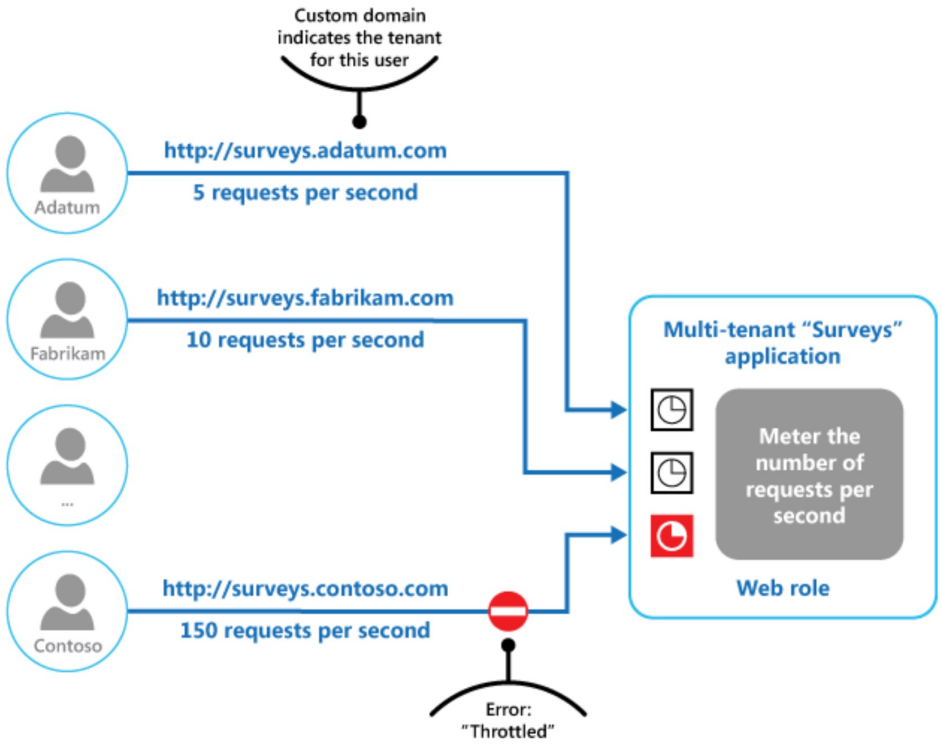
Throttling

- Problema:
  - La carga de una aplicación varia
    - con el tiempo
    - el numero de usuarios activos
    - El tipo de actividad
    - Ráfagas de actividad/busy hours

Sistemas Distribuidos: Patrones de Diseño

Throttling

- Solución:
  - Alternativa al auto scaling
  - Limitar los recursos
  - Throttle cuando se llega a un límite
    - Rechazar operaciones cuando se supera el límite de op/s permitidas
    - Desabilitar temporalmente o degrader funcionalidad de servicios no escenciales. Por ejemplo calidad del video
    - Suspender o rechazar indicando que se debe reintentar más tarde



FIN