

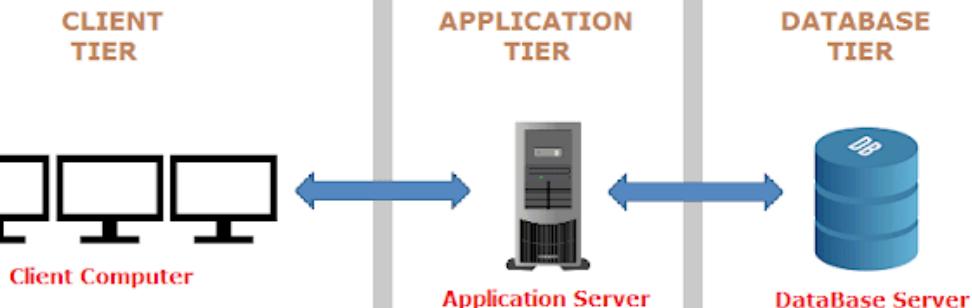
# Patrones Arquitectónicos

- ▶ Similar a patrones de diseño
- ▶ Soluciones que se encuentran a menudo en el mundo real
- ▶ Permiten no tener que partir de cero

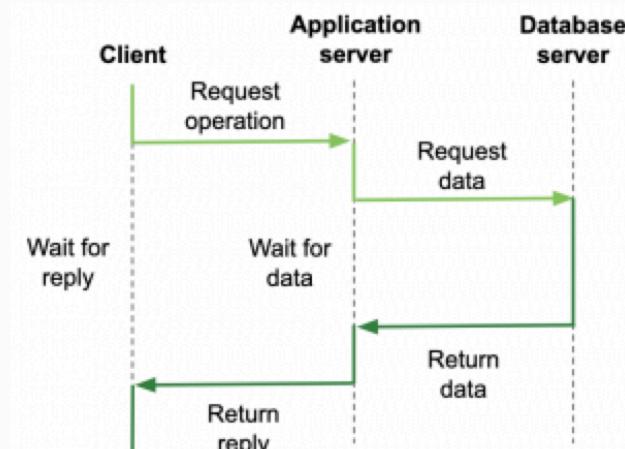
# Client-Server y Tiered

- ▶ Dos grandes roles: cliente y servidor (2 Tier)
- ▶ Servidor recibe y procesa solicitudes de cliente
- ▶ Tremendamente popular en los 80s y 90s dio origen a una variación conocida como “3 capas” o “thin client” (3 Tier)
- ▶ Thin client
  - ▶ tier 1: server
  - ▶ tier 2: application server
  - ▶ tier 3: client

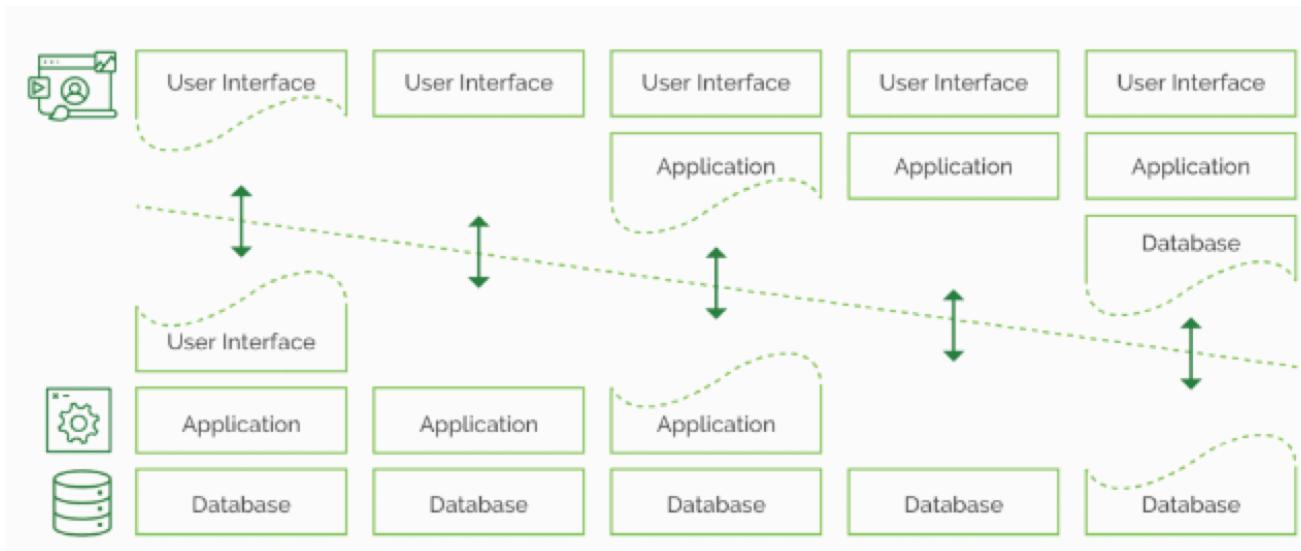
## THREE-TIER ARCHITECTURE



© www.SoftwareTestingMaterial.com

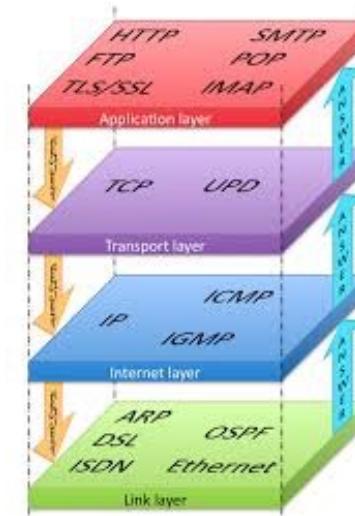
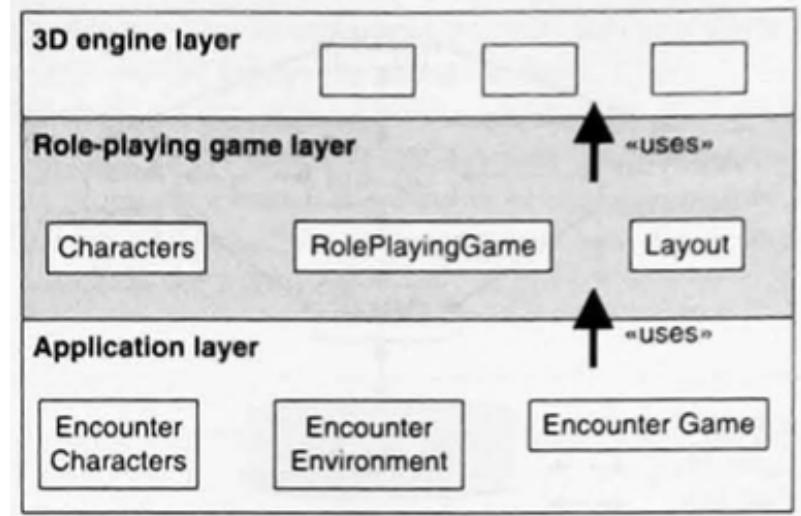


# Variaciones en Cliente Servidor



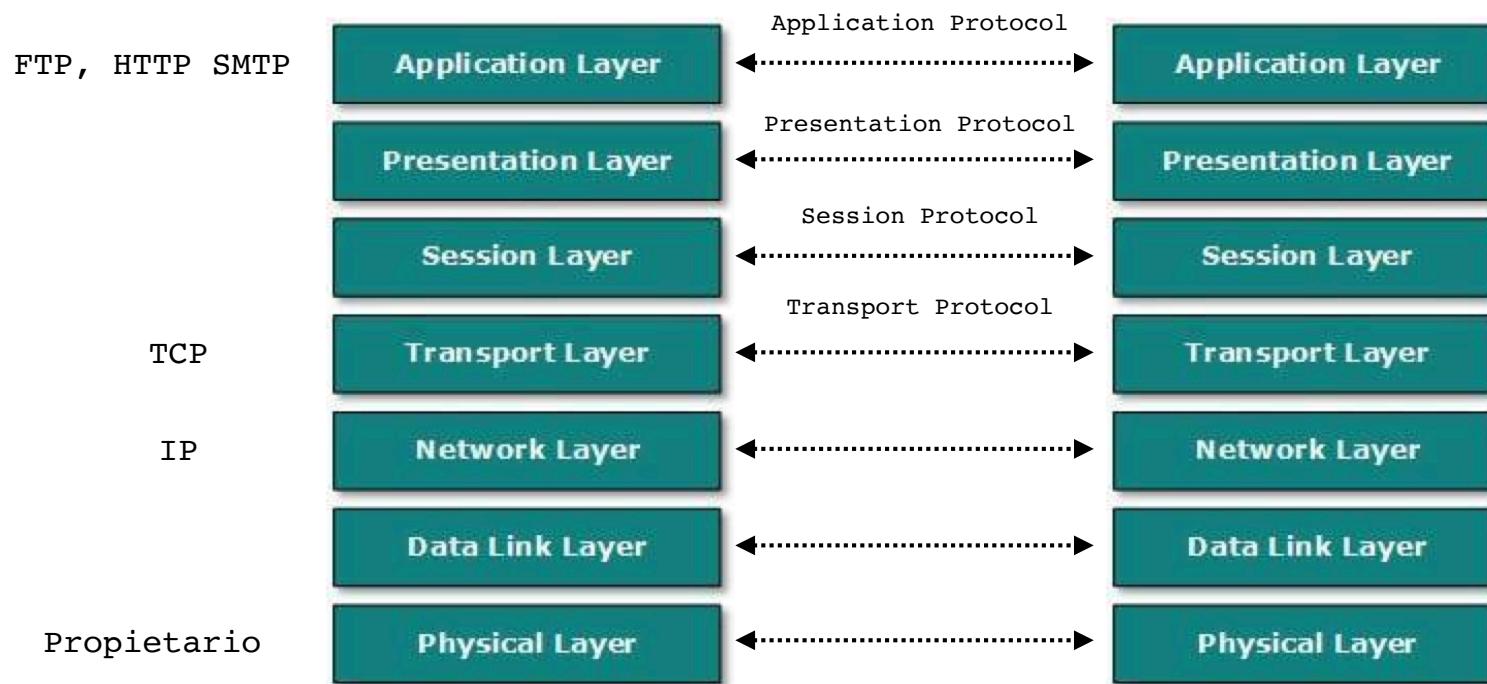
# Arquitectura de Capas

- ▶ una capa es una colección (lógica) coherente de componentes
- ▶ usa y es usada por a lo más una capa



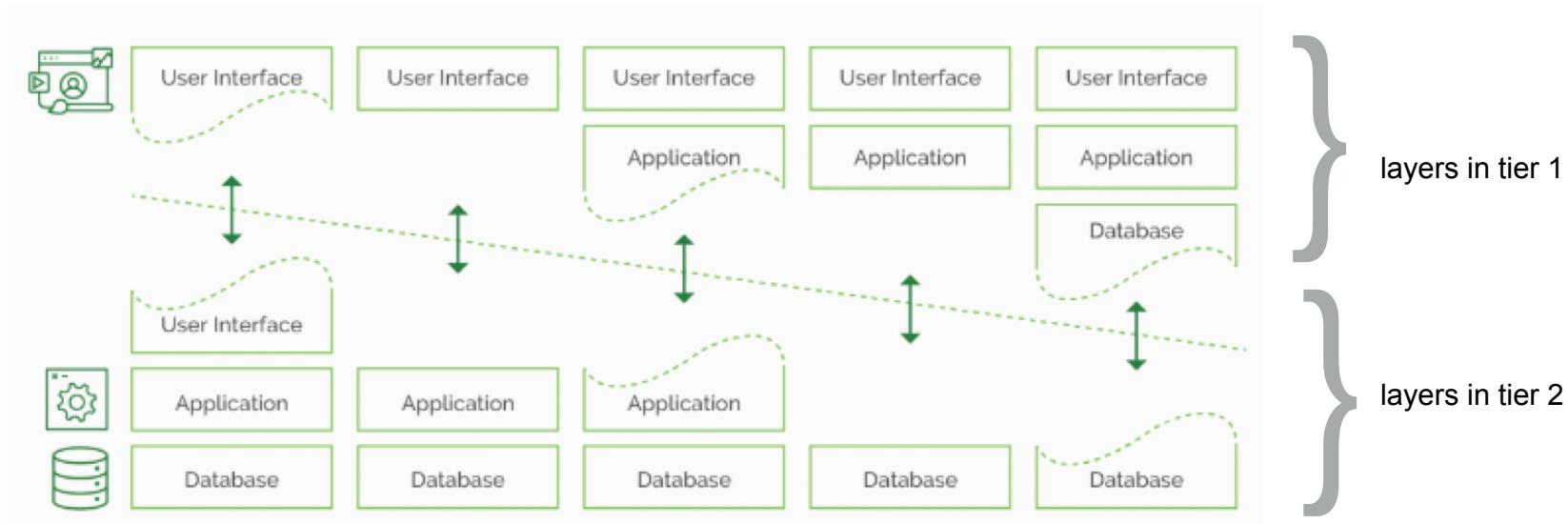
# Ejemplo de Arquitectura de Capas

- El modelo OSI para comunicación entre computadores



# Pueden aparecer combinados

- ▶ 2 Tier + layered

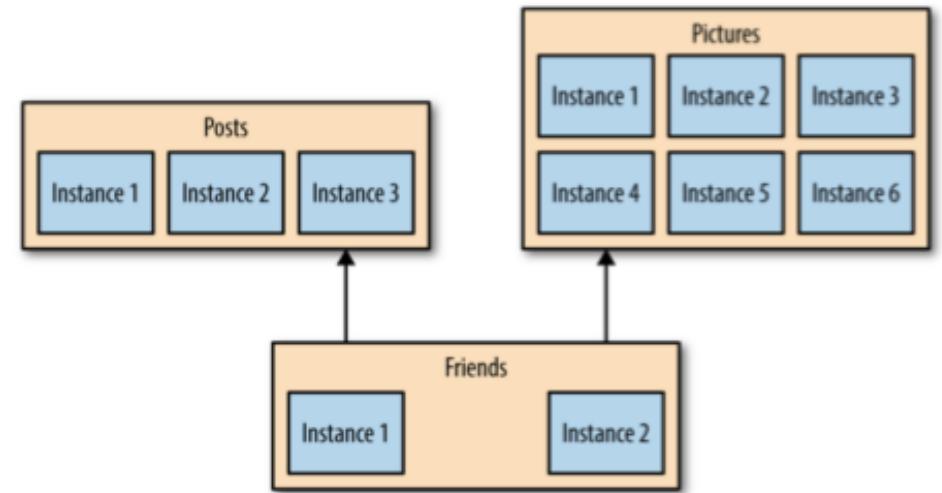
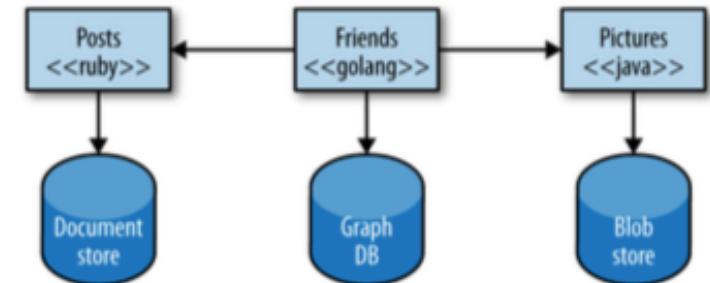


# Arquitectura de Servicios: componentes débilmente acopladas

- ▶ una componente es una unidad de software que puede ser reemplazada o mejorada (upgrade) en forma independiente
- ▶ tradicionalmente las componentes se presentan en forma de módulos o librerías compartidas
- ▶ un servicio es una componente que no corre en el mismo proceso y se comunica con el resto mediante un protocolo como http

# Ventajas

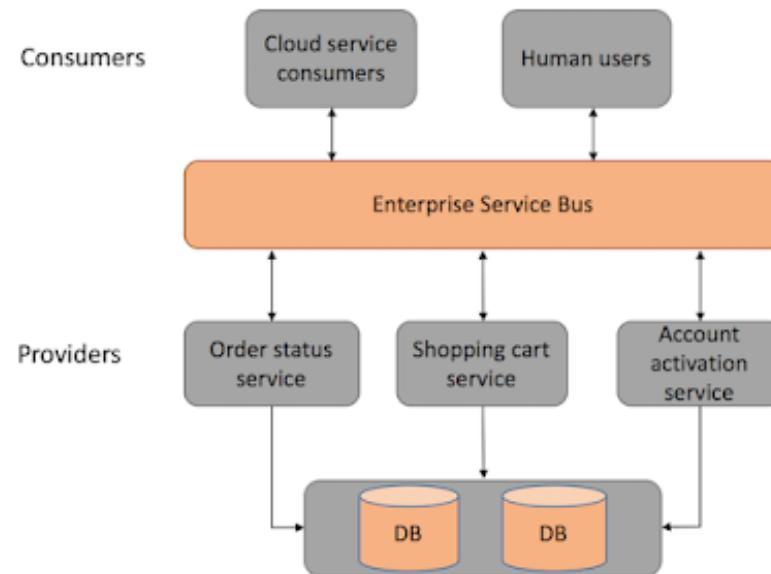
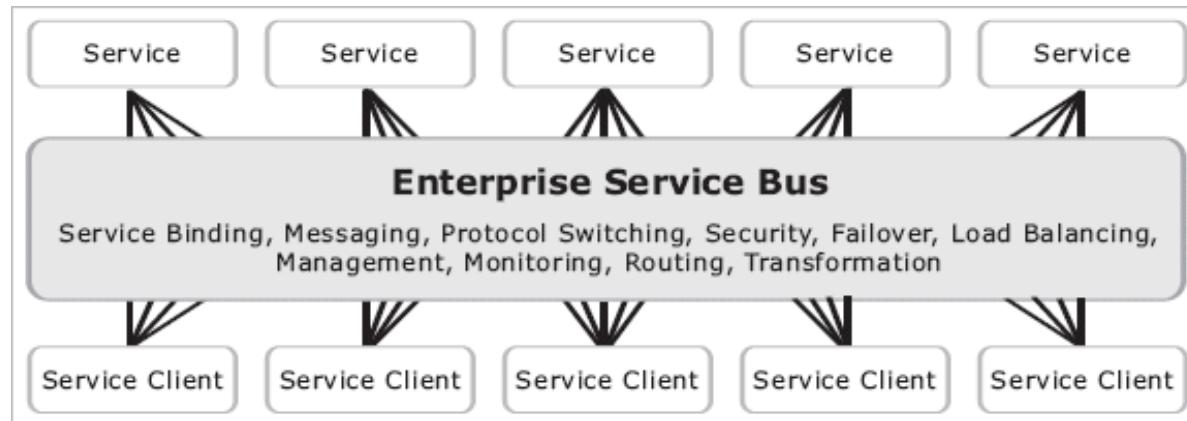
- ▶ deployable en forma independiente
  - ▶ si se hace un cambio de una componente no es necesario redeployar la aplicación
- ▶ interfaz explícita
  - ▶ hace más difícil violar acuerdos
- ▶ heterogeneidad tecnológica
- ▶ resiliencia
- ▶ escalamiento flexible



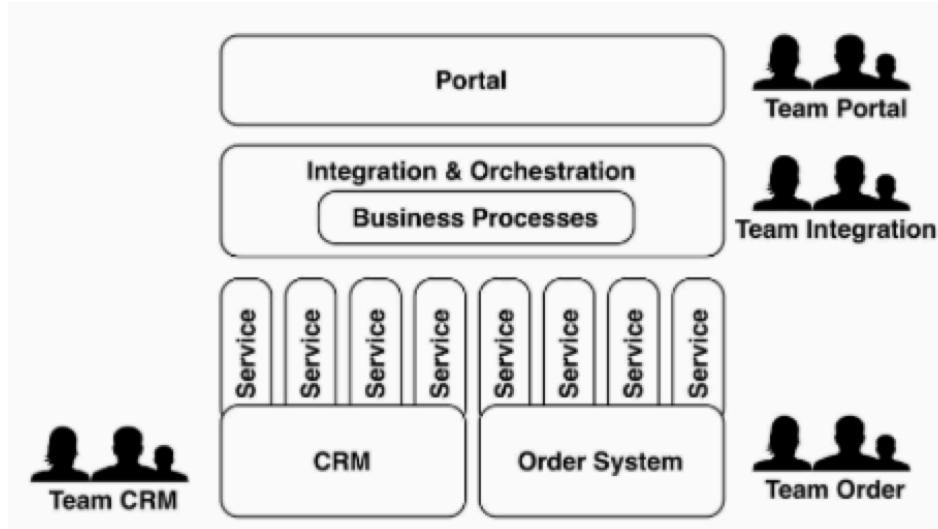
# Service Oriented Architecture (SOA)

- ▶ mediados de los 2000, asociado a la aparición de los Web Services
- ▶ Sistema se concibe como una combinación de servicios
- ▶ Servicios proveen funcionalidades de acuerdo a una especificación de interfaz
- ▶ Pueden ser combinados en forma dinámica
- ▶ Intimamente ligado a la popularidad de los Web Services y a la formalización de los procesos de negocio

# SOA

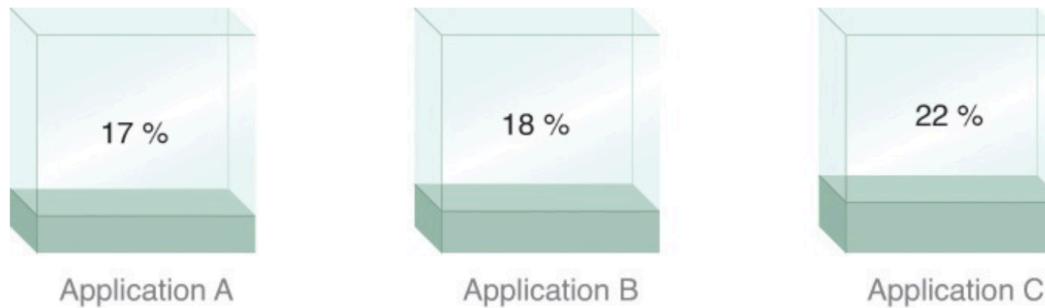


# Ejemplo



- ▶ CRM - aplicación para manejo de clientes
  - ▶ servicios pueden ser creación de clientes nuevos o información de historia de un cliente
- ▶ Order System - aplicación que maneja órdenes
- ▶ Integration Platform - permite que los servicios interactúen
- ▶ Portal - interfaz para los usuarios

# Compartiendo Código



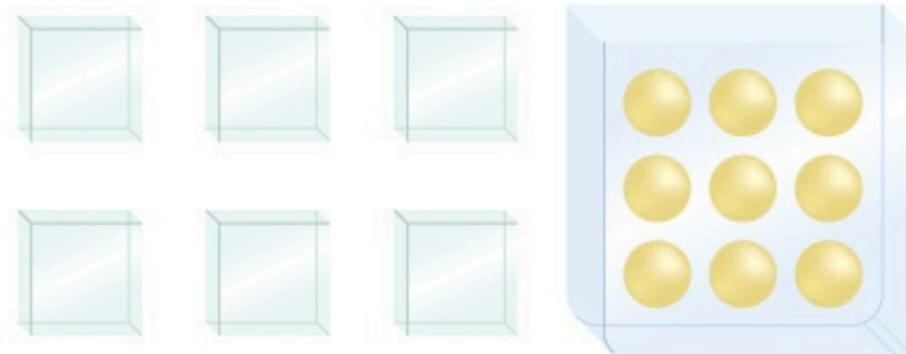
- ▶ En arquitectura clásica un proceso de negocios es implementado como una aplicación
- ▶ Necesariamente en cada aplicación hay código redundante (20%)
- ▶ Con SOA se puede compartir el código que es común



quantity of overall  
automation logic =  $x$



enterprise with an inventory of standalone applications



quantity of overall  
automation logic = 85% of  $x$

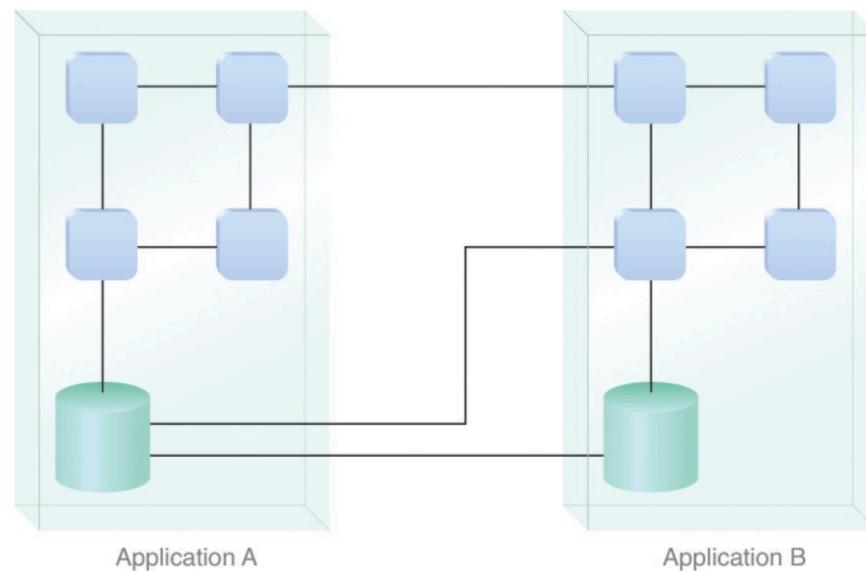
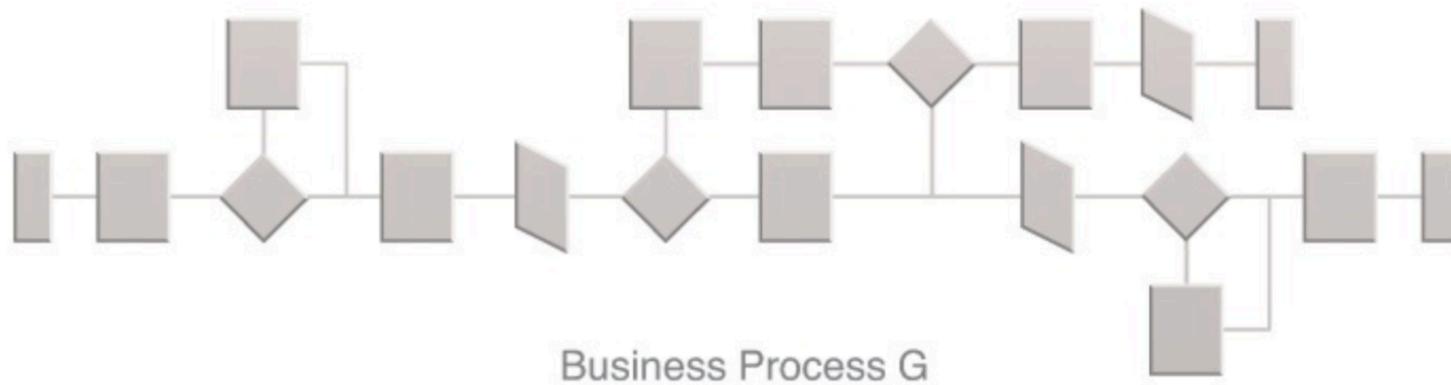
enterprise with a mixed inventory of standalone  
applications and services



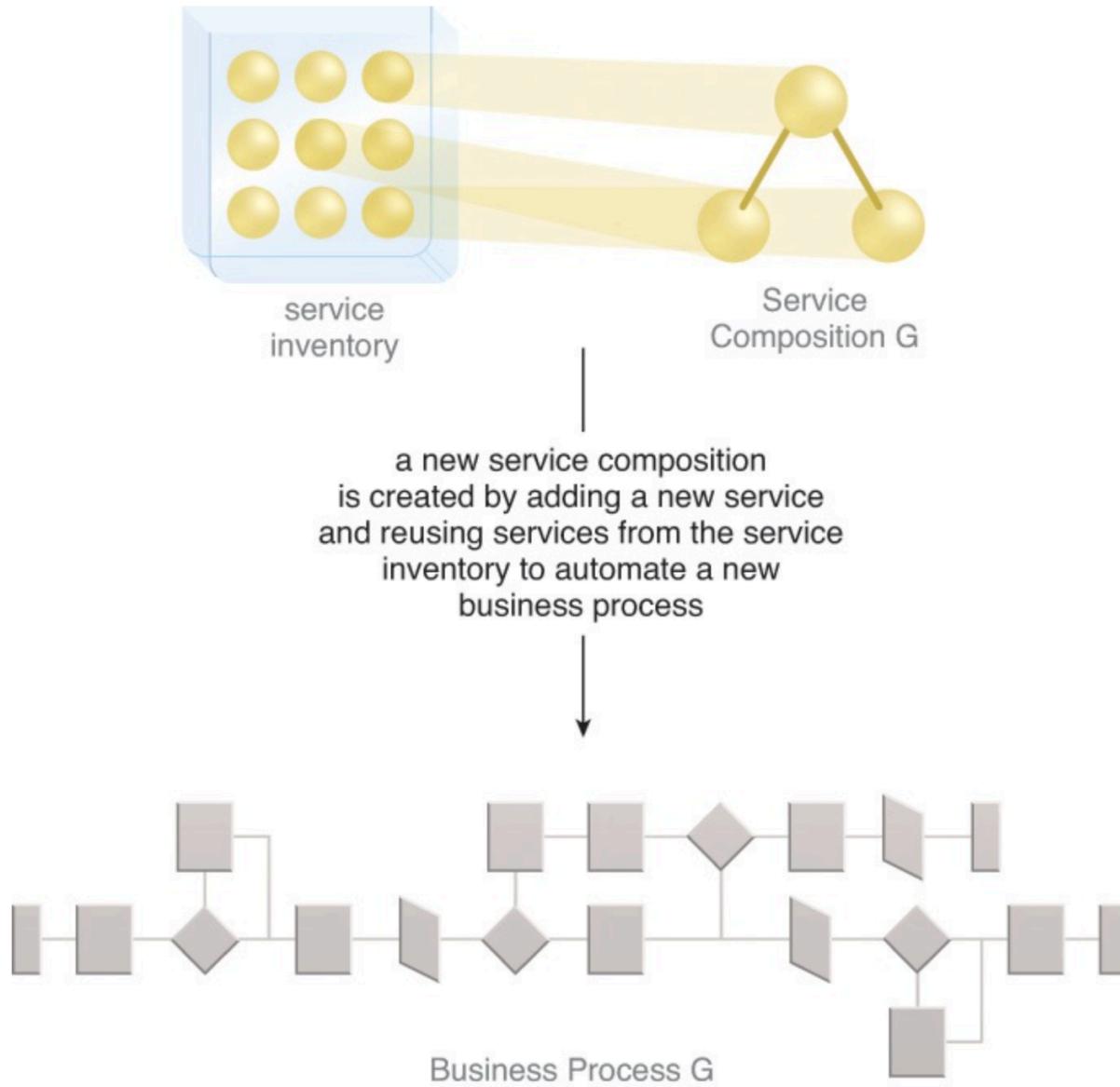
quantity of overall  
automation logic = 65% of  $x$

enterprise with an inventory of services

# A veces un proceso de negocio requiere integrar aplicaciones

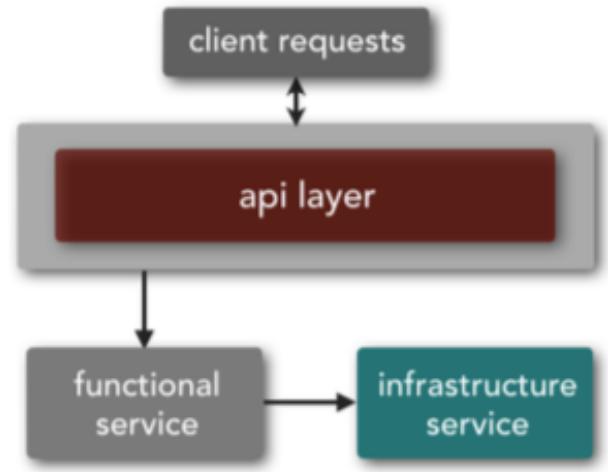


# Con Arquitectura de Servicios



# La llegada de los microservicios

- ▶ 2011 - 2012
- ▶ algunos se refieren a "SOA hecho bien"
- ▶ primera generación de SOA comenzó a hacerse demasiado compleja
  - ▶ SOAP -> middleware
- ▶ representa una maduración de las ideas de SOA a la luz de la experiencia práctica
- ▶ no requiere capa de middleware
  - ▶ cada servicio expone una API (contrato)
- ▶ la mayor parte son servicios funcionales con algunos servicios de infraestructura (no se exponen al mundo externo)



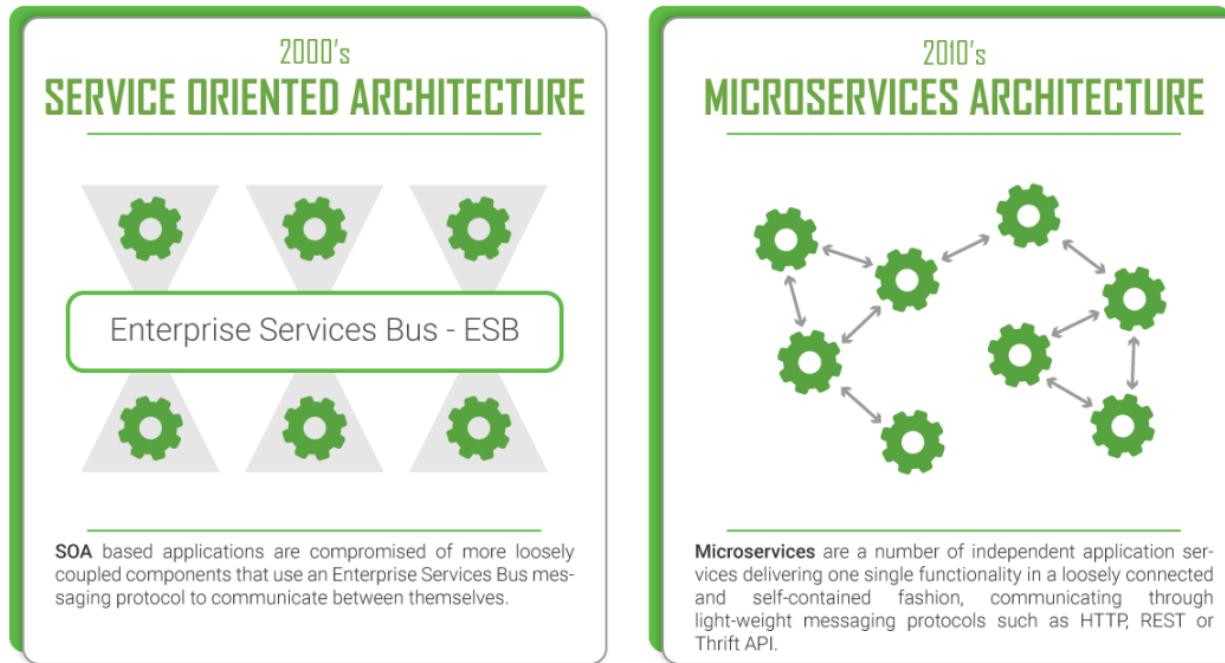
# Microservicios

- ▶ Una aplicación se construye en base a un conjunto de pequeños servicios (HTTP, Rest APIs)
- ▶ Servicios pueden escribirse en lenguajes distintos y usar distintos almacenamientos
- ▶ Rompe en pequeños servicios la componente que usualmente corresponde al server
- ▶ En lugar de tener que replicar la aplicación para que escale se replican los servicios que lo requieran

# Características de Arquitectura de Microservicios

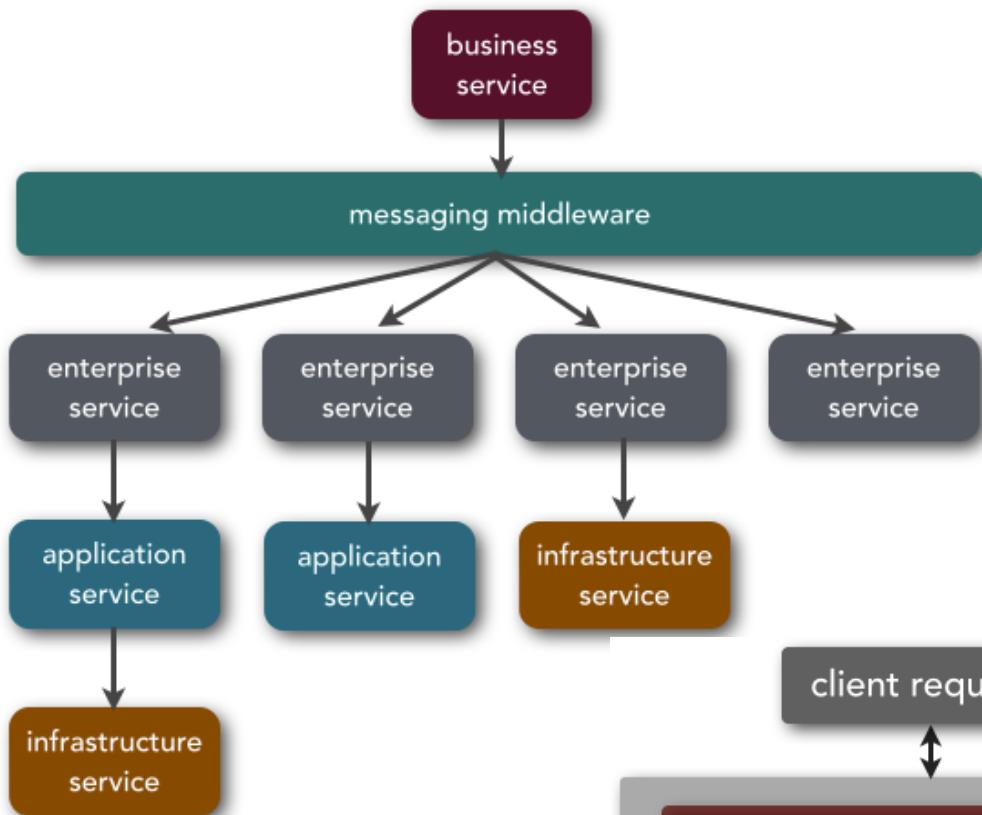
- ▶ Componentes son Servicios
- ▶ Altamente cohesivas y desacopladas (smart endpoints, dumb pipes)
- ▶ Usan principios y protocolos de la Web (Http, Rest)
- ▶ Organización en base a capacidades de negocio y no de especializaciones tecnológicas (UI, DBA, middleware)
- ▶ Desarrollo y deployment descentralizado
- ▶ Gobierno descentralizado (un servicio es totalmente independiente)
- ▶ Manejo de datos descentralizado

# Microservicios vs SOA

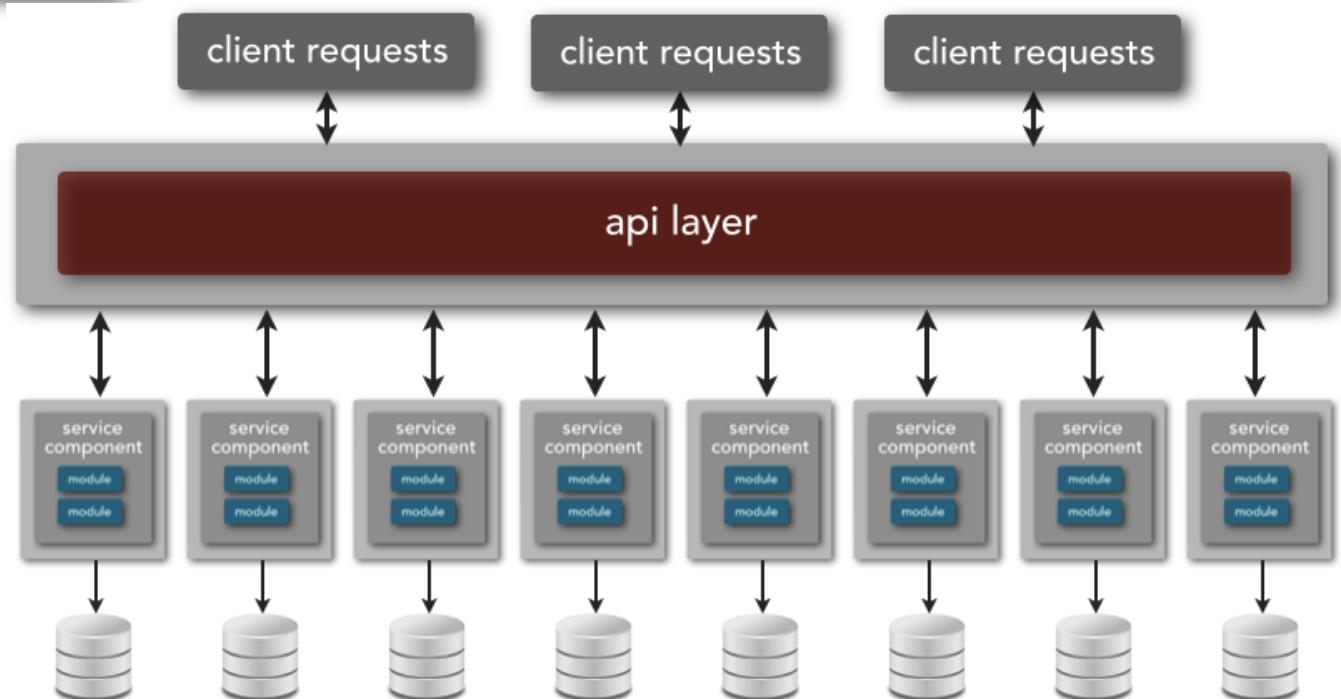


- granularidad mucho menor
- No hay necesidad de capa de integración/orquestación
- Cada microservicio es responsable de comunicarse con quien lo requiera (mensaje simples sin inteligencia)
- Alcance no tiene que ser la organización completa

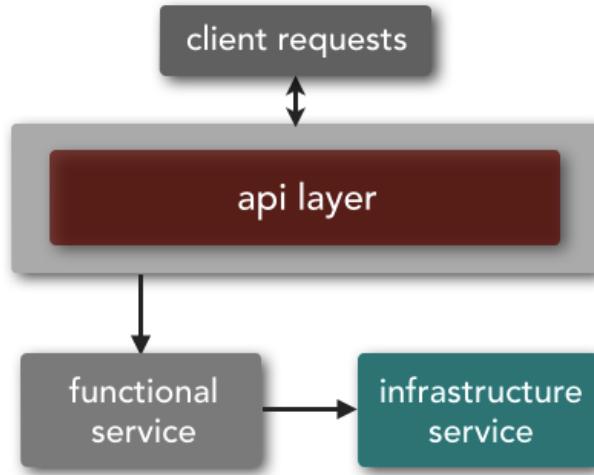
## SOA



## MicroServicios



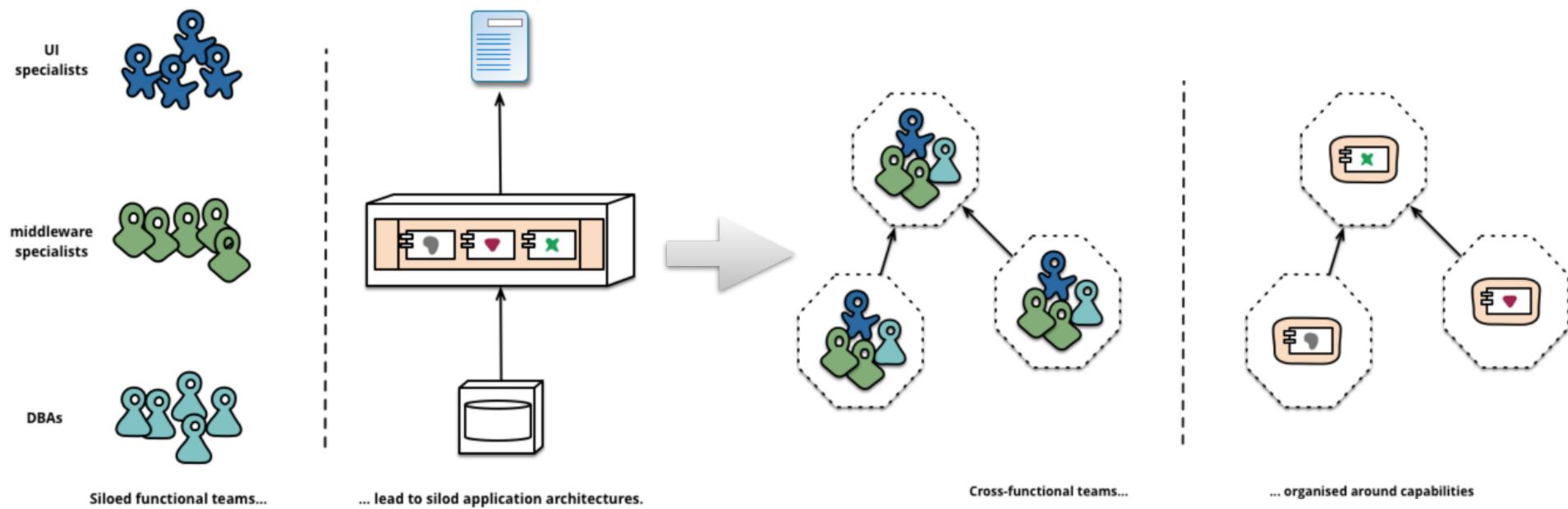
# Microservicios Internos



- ▶ Servicios funcionales - Se exponen hacia afuera
- ▶ Servicios de infraestructura - Para uso interno (servicios internos)
  - ▶ autenticación y autorización
  - ▶ monitoreo
  - ▶ logging y auditoría

# Equipos multidisciplinarios independientes

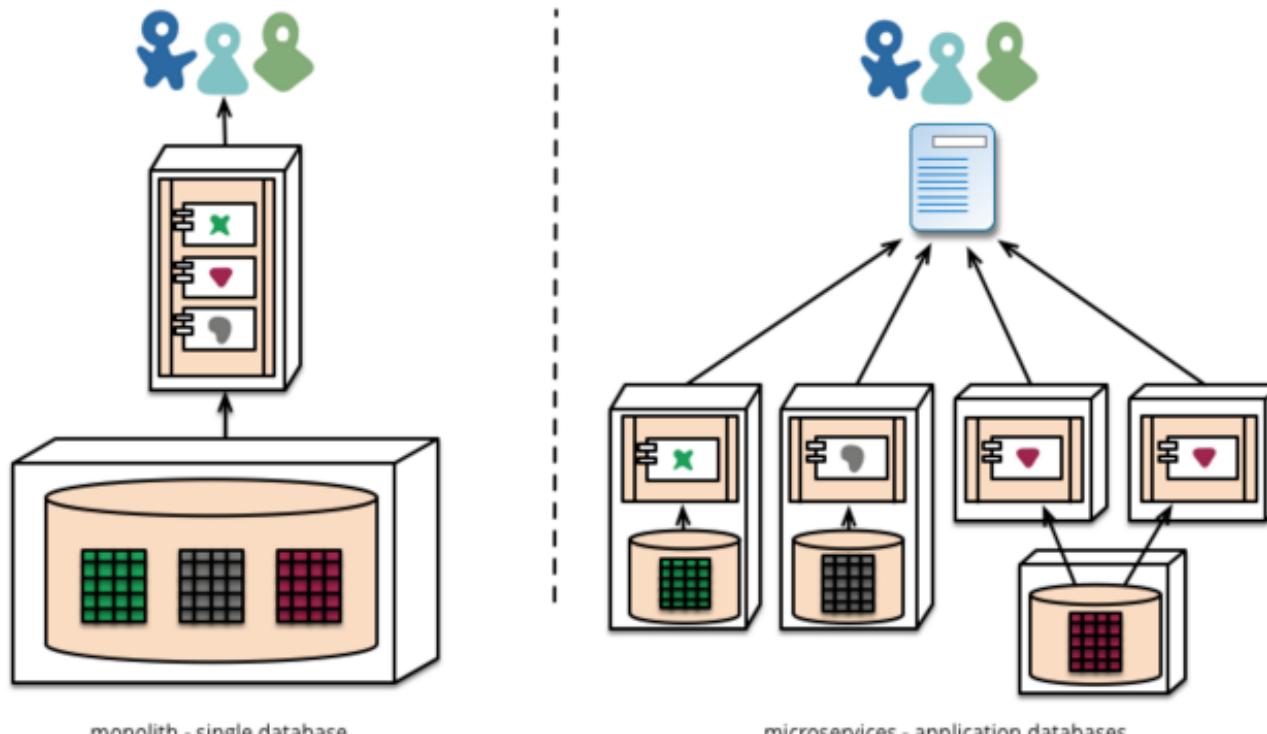
- enfoque clásico es generar equipos de front-end, back-end, bases de datos
- cambios requieren que equipos conversen
- cliente necesita conversar con varios equipos



**Clásico: front-end, back-end, DB**

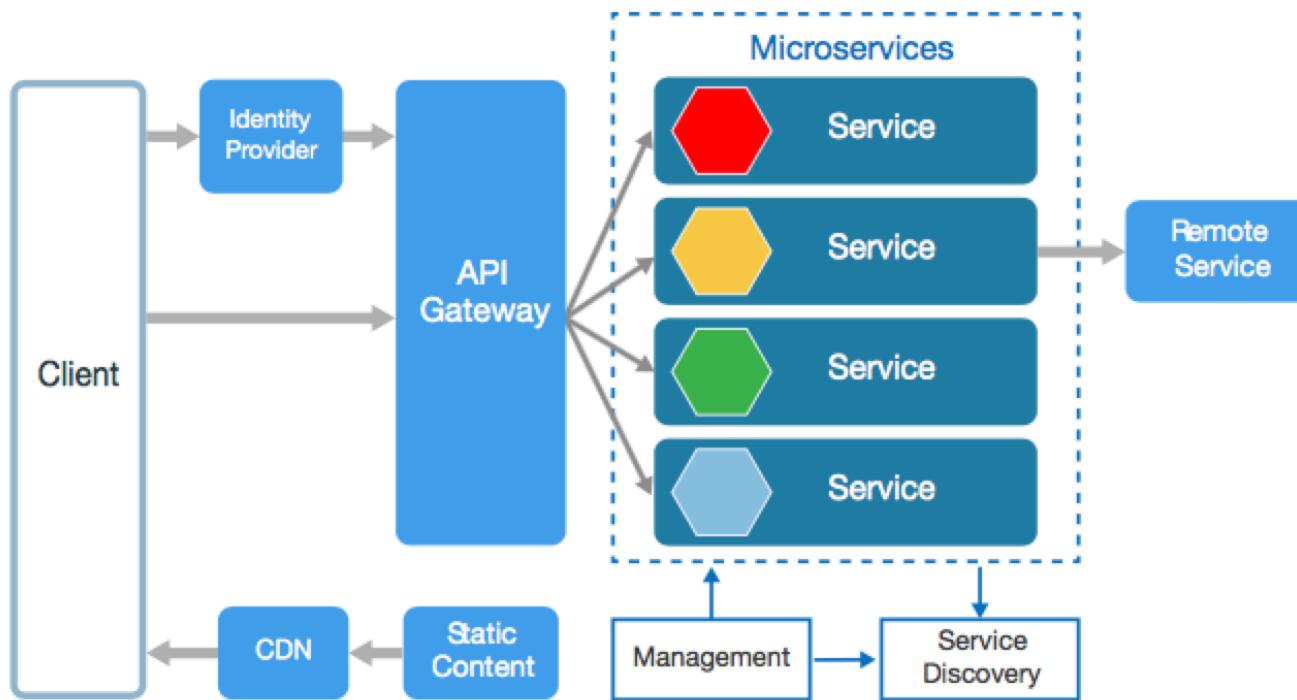
**Microservices: equipos multifuncionales**

# Manejo de Datos Descentralizado



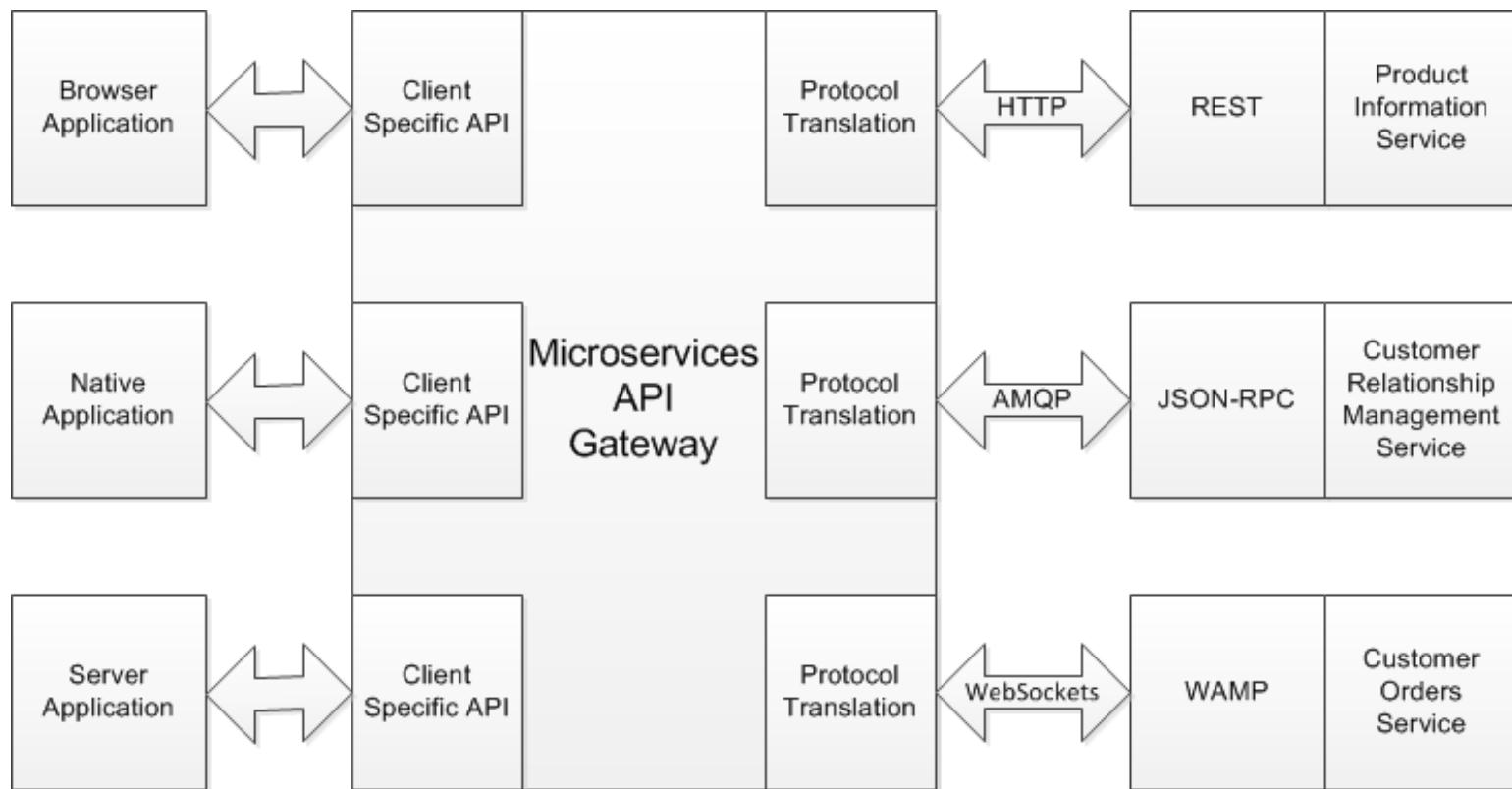
- cada servicio maneja su propia BD
- pueden ser instancias de una BD o tecnologías completamente diferentes
- no se usan transacciones para coordinar, consistencia eventual
- respuesta rápida y escalabilidad se paga con posibles grados de inconsistencia temporal

# El API Layer



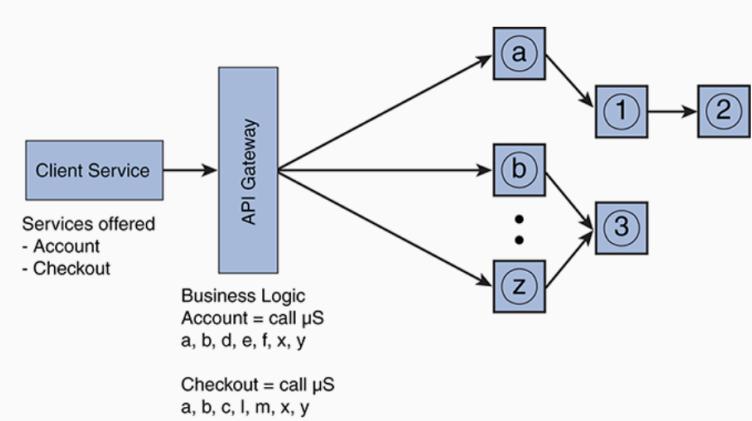
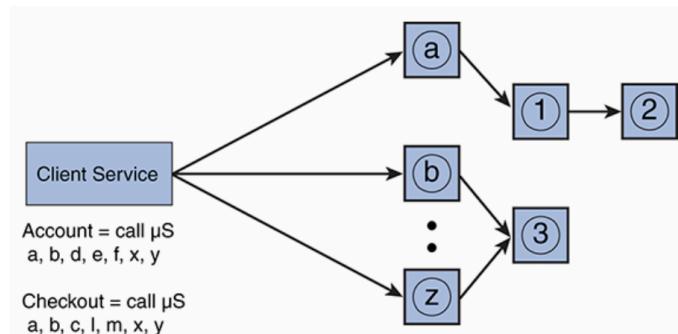
- ▶ es una fachada
- ▶ lo que se expone a clientes
- ▶ puede agregar respuestas de varios servicios (simplifica el uso desde el cliente)
- ▶ permite cambiar interfaz de servicios
- ▶ servicios pueden usar protocolos que no son http
- ▶ puede proporcionar servicios generales (logging, load balancing, etc)

# API Gateway

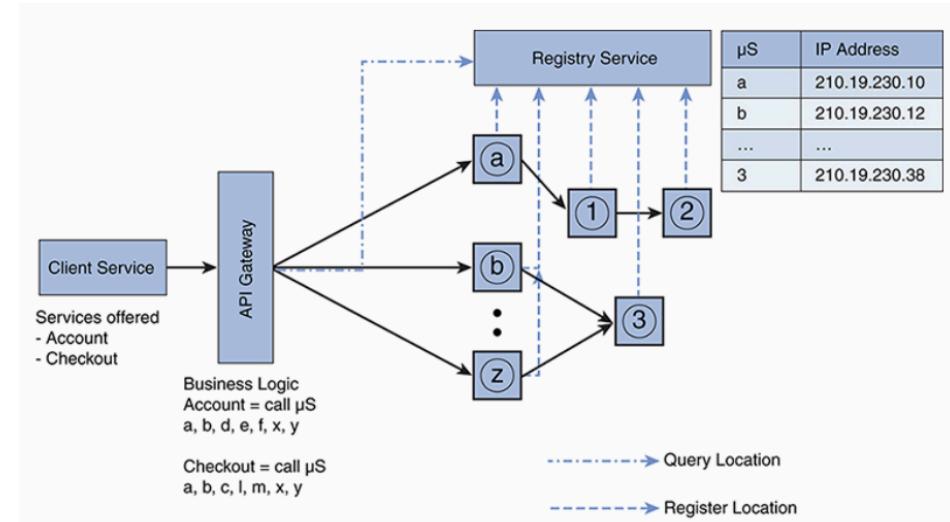


# Variaciones

## Sin API Gateway

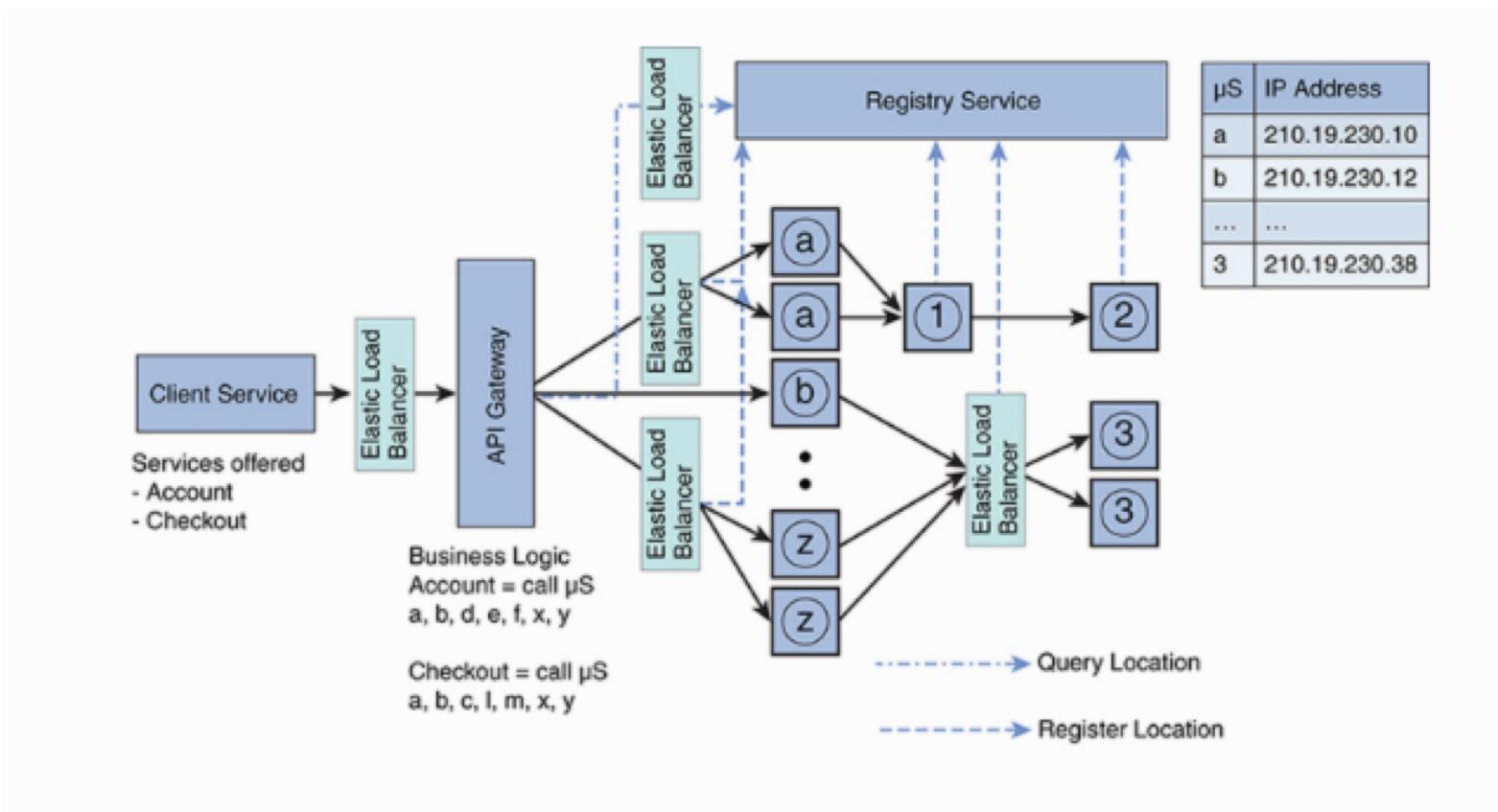


Con API Gateway



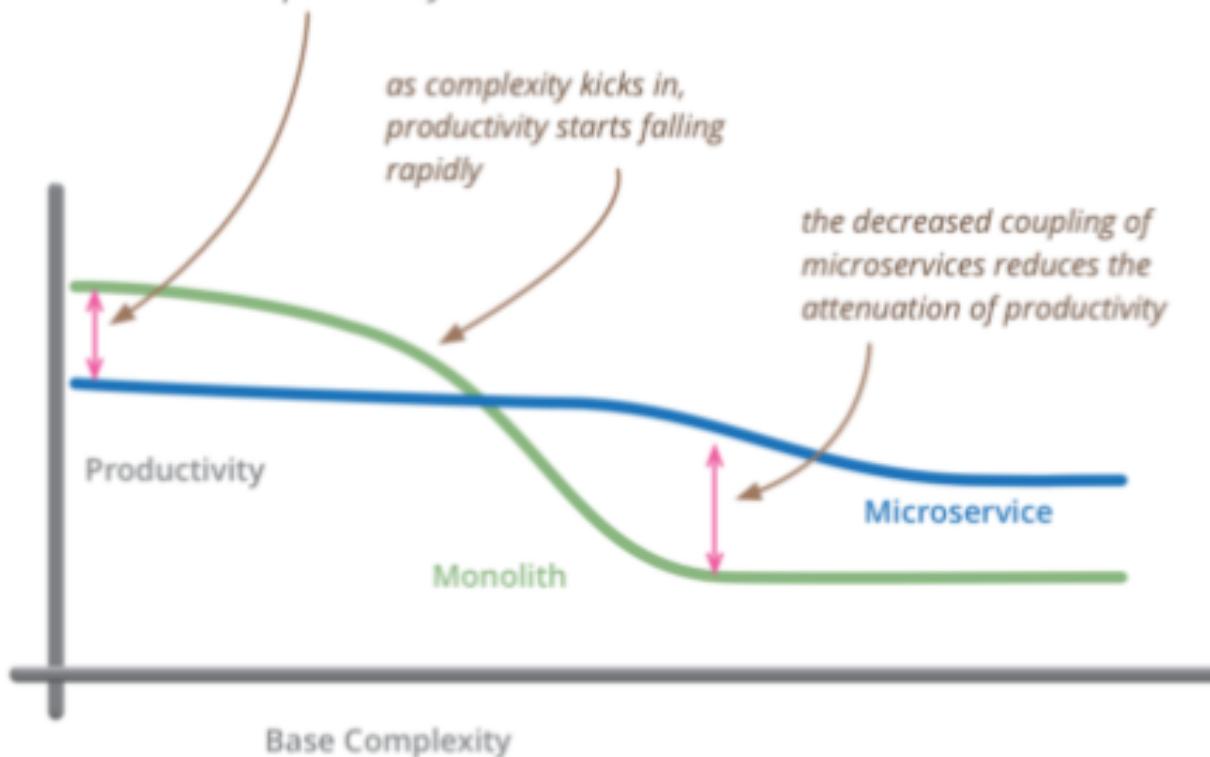
Con API Gateway y Registry

# Con balanceadores de carga



# No es la panacea

*for less-complex systems, the extra baggage required to manage microservices reduces productivity*



*but remember the skill of the team will outweigh any monolith/microservice choice*

# Ventajas y Desventajas

- ▶ simplicidad y escalabilidad
- ▶ facilita la entrega continua
- ▶ segregación y descentralización de los datos
- ▶ flexibilidad de elegir lenguajes, bases de datos, plataforma, etc
- ▶ desempeño (mayor tiempo de latencia)
- ▶ versionamiento complejo
- ▶ complejidad del todo
- ▶ puede requerir mayor esfuerzo de desarrollo total
- ▶ transacciones, consistencia

# Desafíos para la Organización

- ▶ necesidad de mantener y monitorear cientos de microservicios
- ▶ problema de descubrimiento, se debe manejar un inventario y proveer formas de consultar
- ▶ como asegurar SLAs cuando hay servicios que dependen de otros
- ▶ no es suficiente asegurar correctitud de los servicios
- ▶ muchas opciones de escalamiento (flexible pero mas complejo)
- ▶ decisiones de upgrading mas complejas
- ▶ asegurar seguridad es mas complejo

# Cambiar o no a Microservicios

- ▶ Supongamos que una organización decide cambiar su tecnología monolítica después de 5 años de uso
  - ▶ necesidad de acomodar nuevas demandas de la competencia y de los usuarios
  - ▶ no es capaz de escalar
  - ▶ tecnologías obsoletas
  - ▶ lento
- ▶ Análisis debe considerar: costos de construcción (ctb), mantenición (ctm), escalamiento (cts), y time to marjet (ttm)

# Comparación de seguir con monolítica (M) o ir a Microservicios (S)

- ▶ En general los costos totales resultan menores al ir a microservicios
  - ▶ Costo de Construir -  $M_{CTB} < S_{CTB}$
  - ▶ Costo de Mantener -  $S_{CTM} < M_{CTM}$
  - ▶ Costo de Escalar -  $S_{CTS} < M_{CTS}$
  - ▶ Time to Market -  $S_{TTM} < M_{TTM}$

# Migración gradual

