



Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación

# Clase 18

# Arquitectura de Servicios

IIC2143 - Ingeniería de Software  
Sección 1

Rodrigo Saffie

[rasaffie@uc.cl](mailto:rasaffie@uc.cl)

23 de mayo de 2018

# ¿Qué es la arquitectura?

- “Técnica y estilo con los que se diseña, proyecta y construye un edificio o un monumento.”



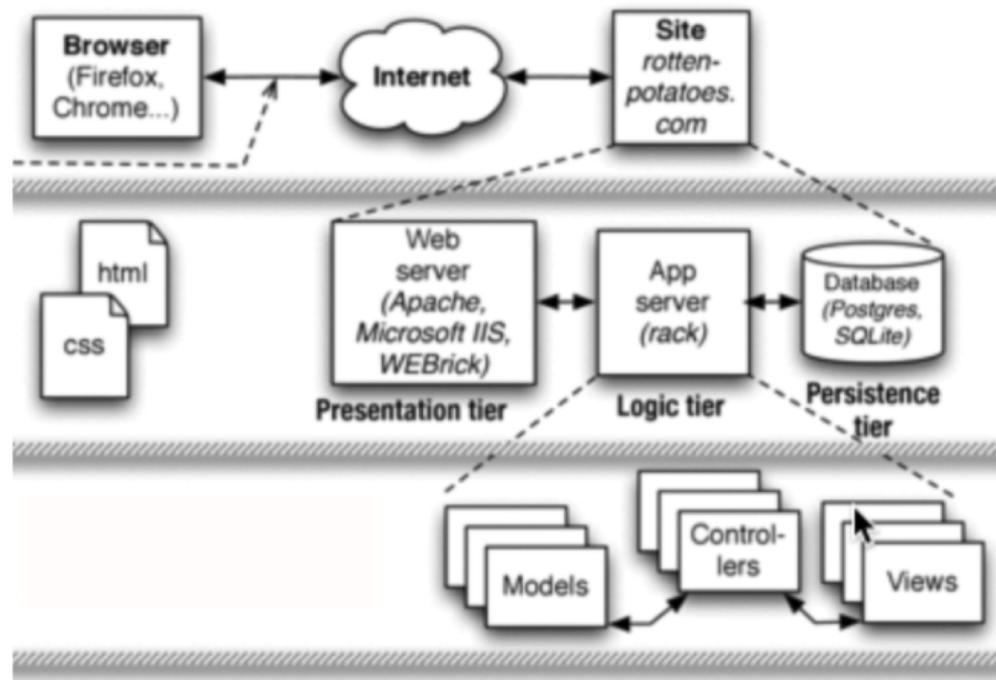
- La arquitectura está definida por los elementos principales de un edificio, su composición y organización.

# ¿Qué es la arquitectura?

- En el *software* es similar: “conjunto de las principales decisiones de diseño realizadas sobre un sistema”
- Es la estructura de los componentes y sus interacciones
- Sin embargo, tiene diferencias:
  - Las edificaciones son un concepto más intuitivo
  - Solamente con mirar un edificio se puede desprender mucha información de este
  - Un *software* con buena arquitectura se adapta a los cambios, los edificios son estáticos

# ¿Qué es la arquitectura?

- Sus decisiones son a nivel de sistemas, mientras que el diseño es a nivel de módulos
- Por este motivo es más costosa de modificar



# ¿Qué es un componente?

- Partes del contexto encapsuladas por los componentes son:
  - Interfaces expuestas para interactuar con otros componentes
  - Recursos (como archivos o directorios) de los cuales depende el componente
  - Dependencias de *software* para ejecutarse (como lenguajes, sistemas operativos, *drivers*, etc...)
  - Configuraciones de *hardware* para ejecutar el componente

# Factores que inciden en la arquitectura

- Requisitos no funcionales

- Seguridad
- Confidencialidad
- Desempeño
- Escalabilidad
- Disponibilidad
- Interoperabilidad
- Portabilidad
- Otros...

# Factores que inciden en la arquitectura

- Requisitos no funcionales
- Principios del sistema (consistencia y claridad del código)
- Restricciones del proyecto

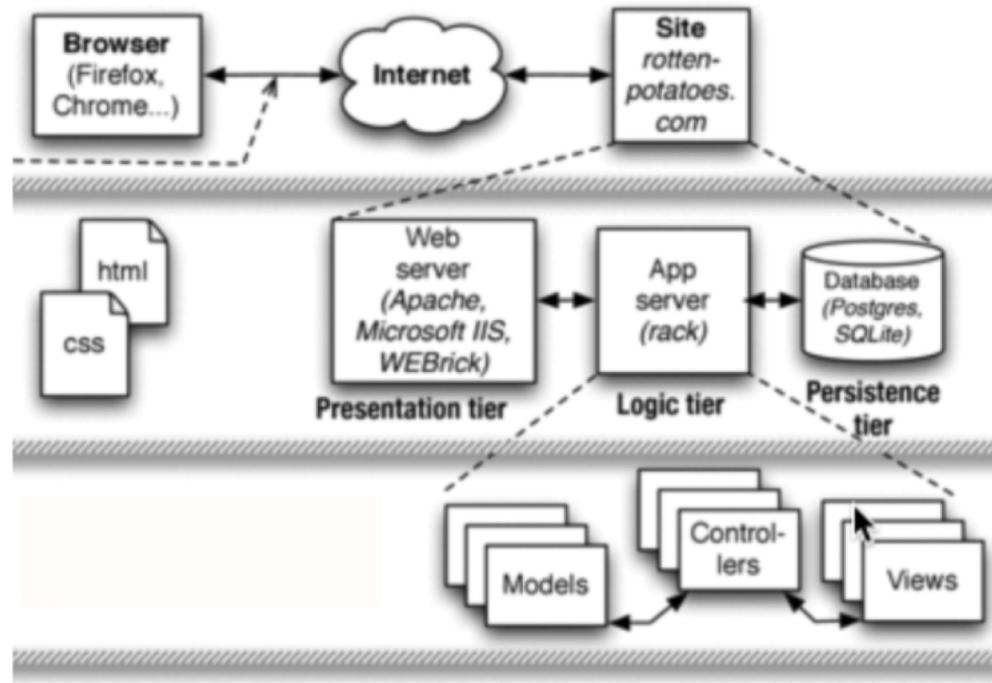
# Restricciones en la arquitectura

- Tiempo y presupuesto
- Tecnológicas
  - lista de tecnologías aprobadas
  - sistemas existentes e interoperabilidad
  - uso de sistemas *open source*
  - madurez de tecnologías
  - relación con los proveedores
- Relativas a personas
  - tamaño y experiencia del equipo
  - disponibilidad de especialistas
  - ¿quién dará mantenición al sistema?

# Patrones arquitectónicos

## *Client – Server*

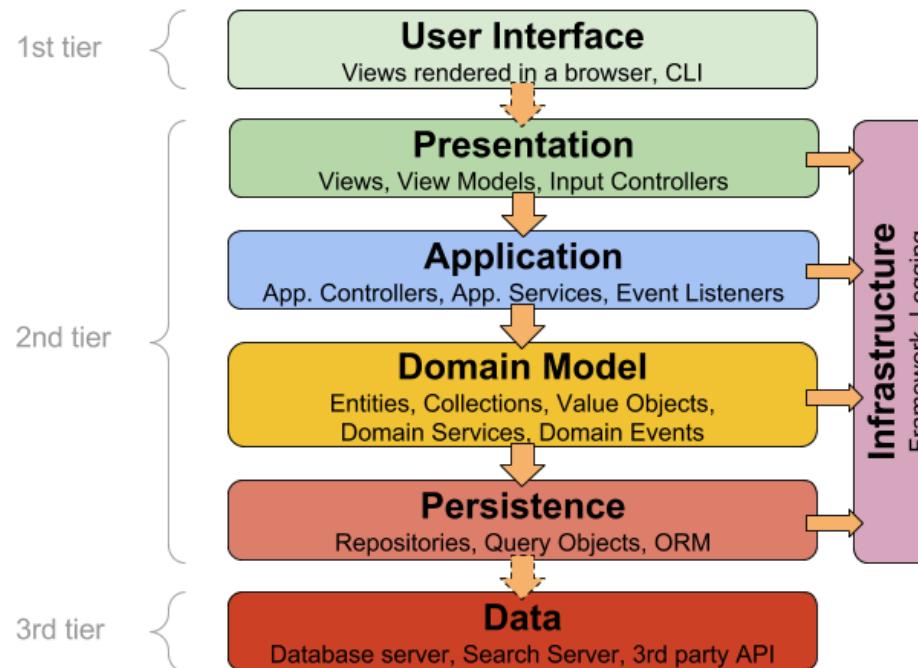
- Dos grandes componentes: cliente y servidor
- Servidor recibe y procesa solicitudes de clientes



# Patrones arquitectónicos

## *Layered (en capas)*

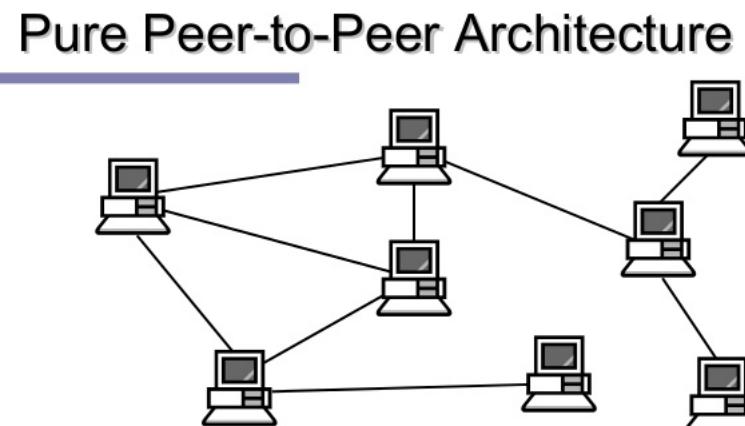
- Una capa es una colección coherente de componentes



# Patrones arquitectónicos

## *Peer-to-peer*

- Red de nodos conectados con las mismas capacidades y responsabilidades
- Permite el intercambio directo de información entre nodos



- No central server
- Clients connected to one or more peers
- Resilient
- Large #messages

# Componentes como Servicios

- Unidades de *software* que pueden ser reemplazadas o mejoradas en forma independiente
  - Ejemplos son módulos o librerías
- Un servicio es una componente que no corre en el mismo proceso o sistema, y se comunica a través de un protocolo de red (como *http* o *rpc*)

# Componentes como Servicios

- Ventajas:
  - *Deployable* de manera independiente
  - Interfaces explícitas
  - Heterogeneidad de tecnologías
  - Resiliencia
  - Escalamiento flexible

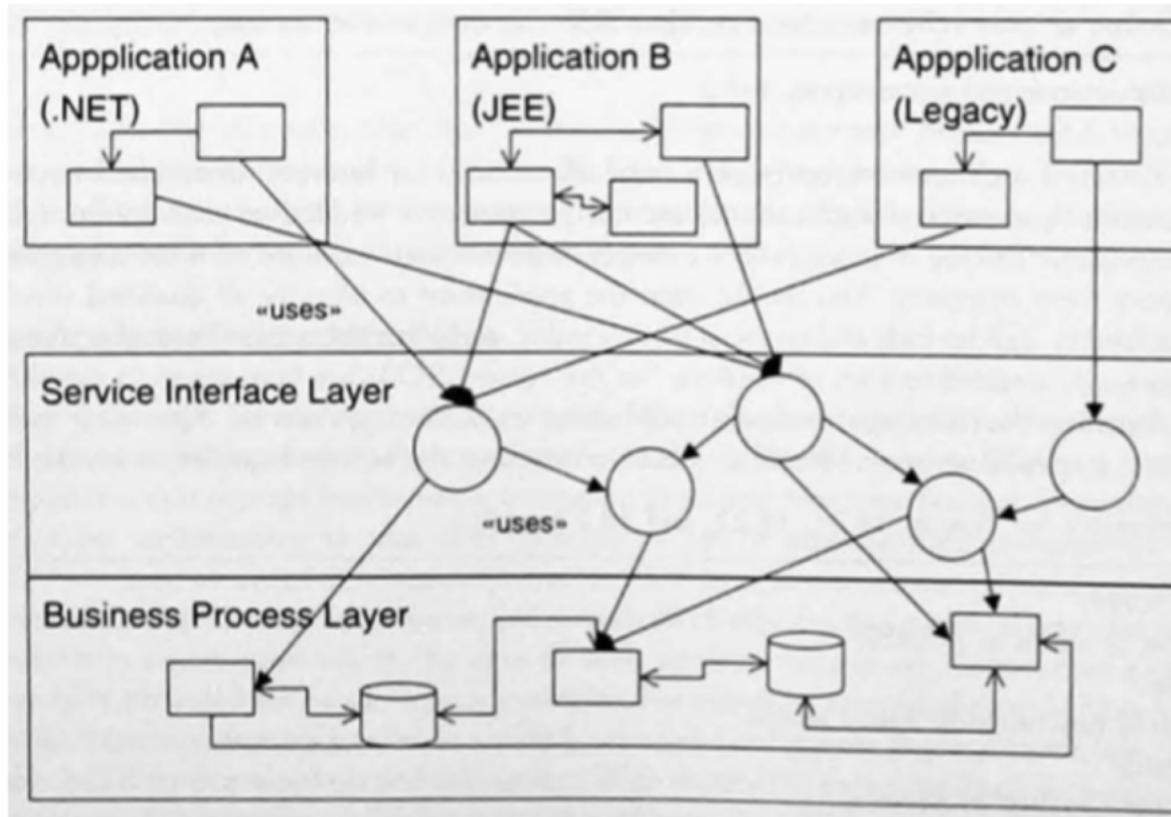
# Patrones arquitectónicos

## *Service Oriented Architecture (SOA)*

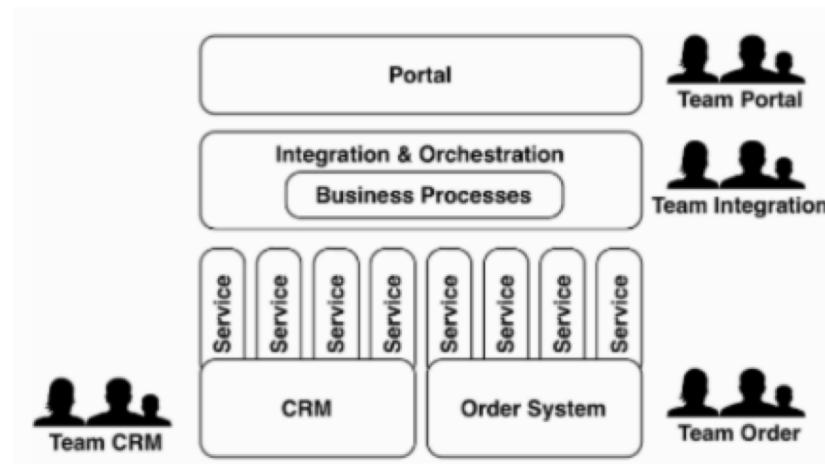
- Sistema se concibe como una combinación de servicios
- Servicios proveen funcionalidades a través de interfaces específicas
- Pueden ser combinados en forma dinámica
- Altamente relacionado con los servicios y APIs web

# Patrones arquitectónicos

*Service Oriented Architecture (SOA)*

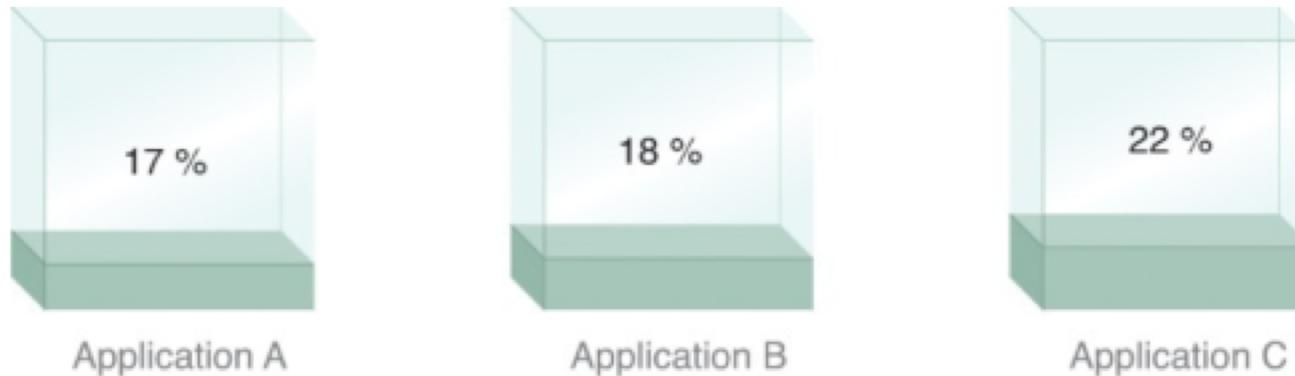


# Ejemplo SOA



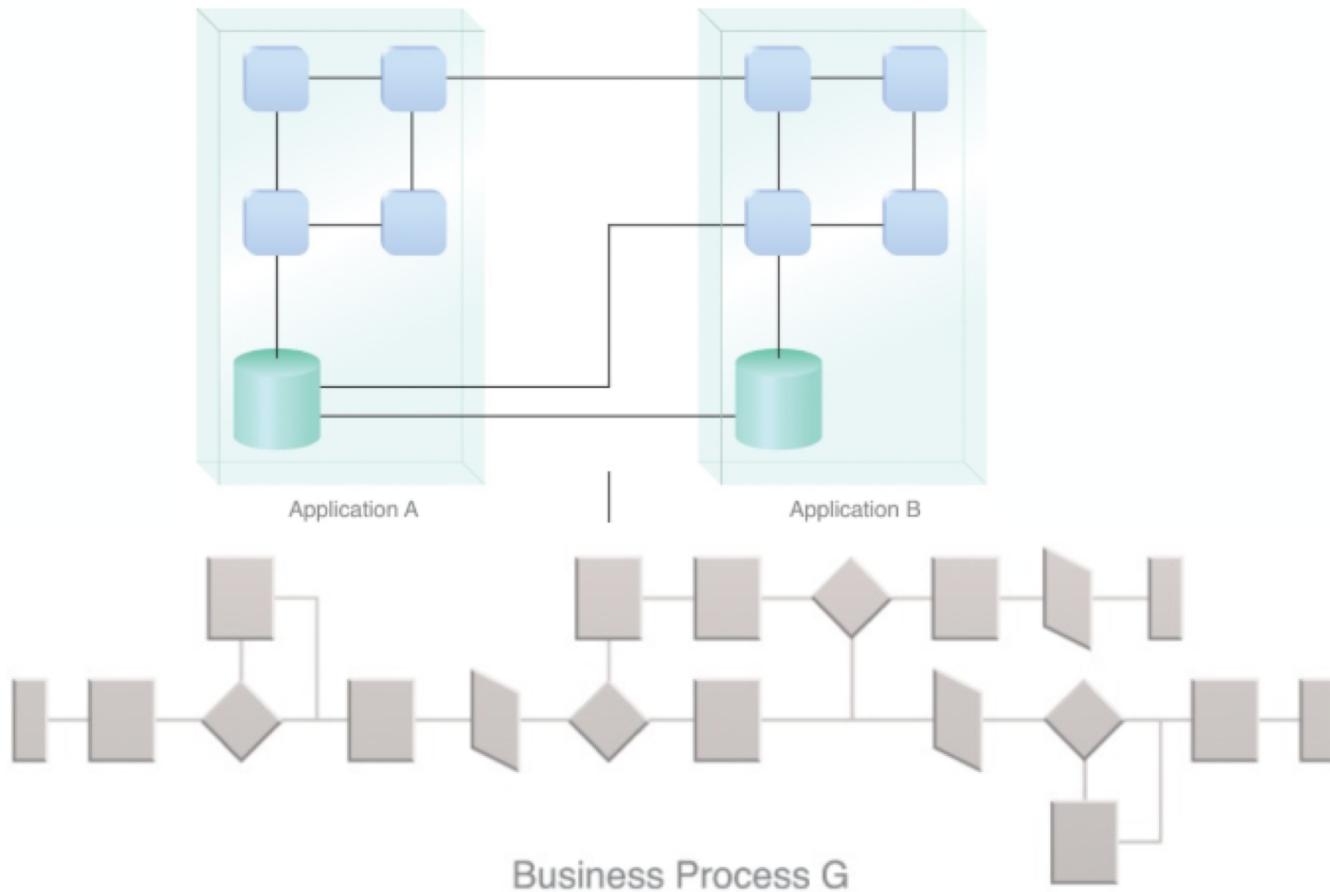
- *CRM* – aplicación para manejo de clientes
  - Servicios pueden ser creación de clientes nuevos o información de historia de un cliente
- *Order System* – aplicación que maneja órdenes
- *Integration Platform* – permite que los servicios interactúen
- *Portal* – interfaz para los usuarios

# SOA: Compartiendo código

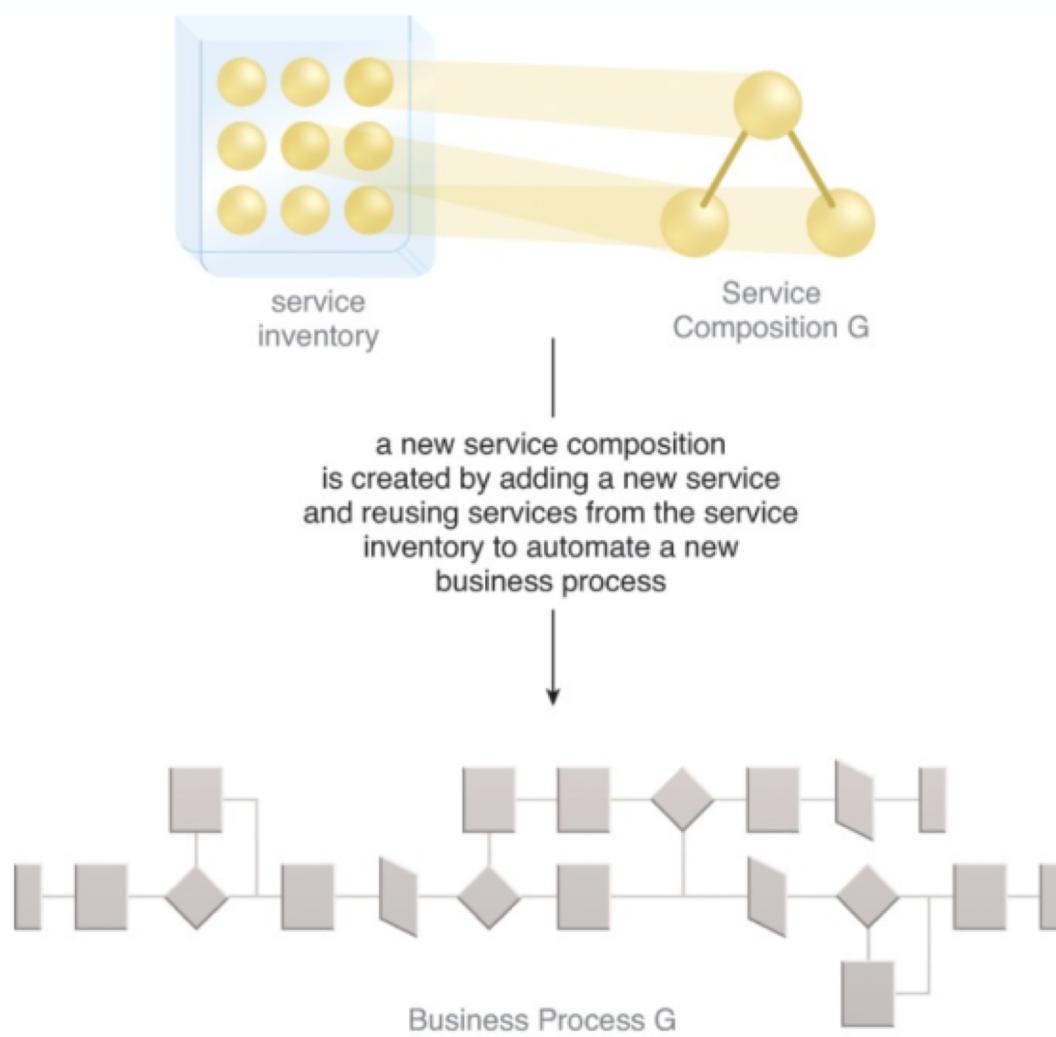


- En arquitectura clásica un proceso de negocios es implementado como una aplicación
- Necesariamente en cada aplicación hay código redundante (20%)
- Con SOA se puede compartir el código que es común

# A veces un proceso de negocio requiere integrar aplicaciones



# Con arquitectura de servicios



# Aparición de los microservicios

- Primera generación de SOA comenzó a hacerse demasiado compleja
  - SOAP -> middleware
  - *Monolithic REST API*
- Microservicios: representan una maduración de las ideas de *SOA* a la luz de la experiencia práctica
- No requieren capa de *middleware*
  - cada servicio expone una API (contrato)

# Características de arquitectura de microservicios

- Componentes son servicios
- Altamente cohesivos y desacoplados
- Usan principios y protocolos de la Web (HTTP, REST)
- Organización en base a lógica de negocio y no de especialización tecnológica (UI, *middleware*)
- 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 (mensajes simples sin inteligencia)
- UI es parte del microservicio
- Alcance no tiene que ser la organización completa

# Microservicios vs SOA

2000's

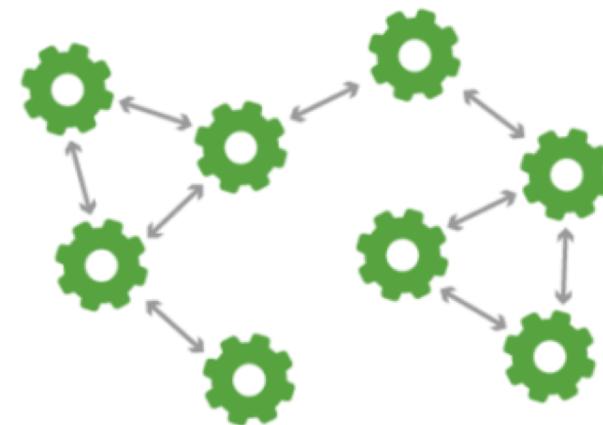
## SERVICE ORIENTED ARCHITECTURE



**SOA** based applications are comprised of more loosely coupled components that use an Enterprise Services Bus messaging protocol to communicate between themselves.

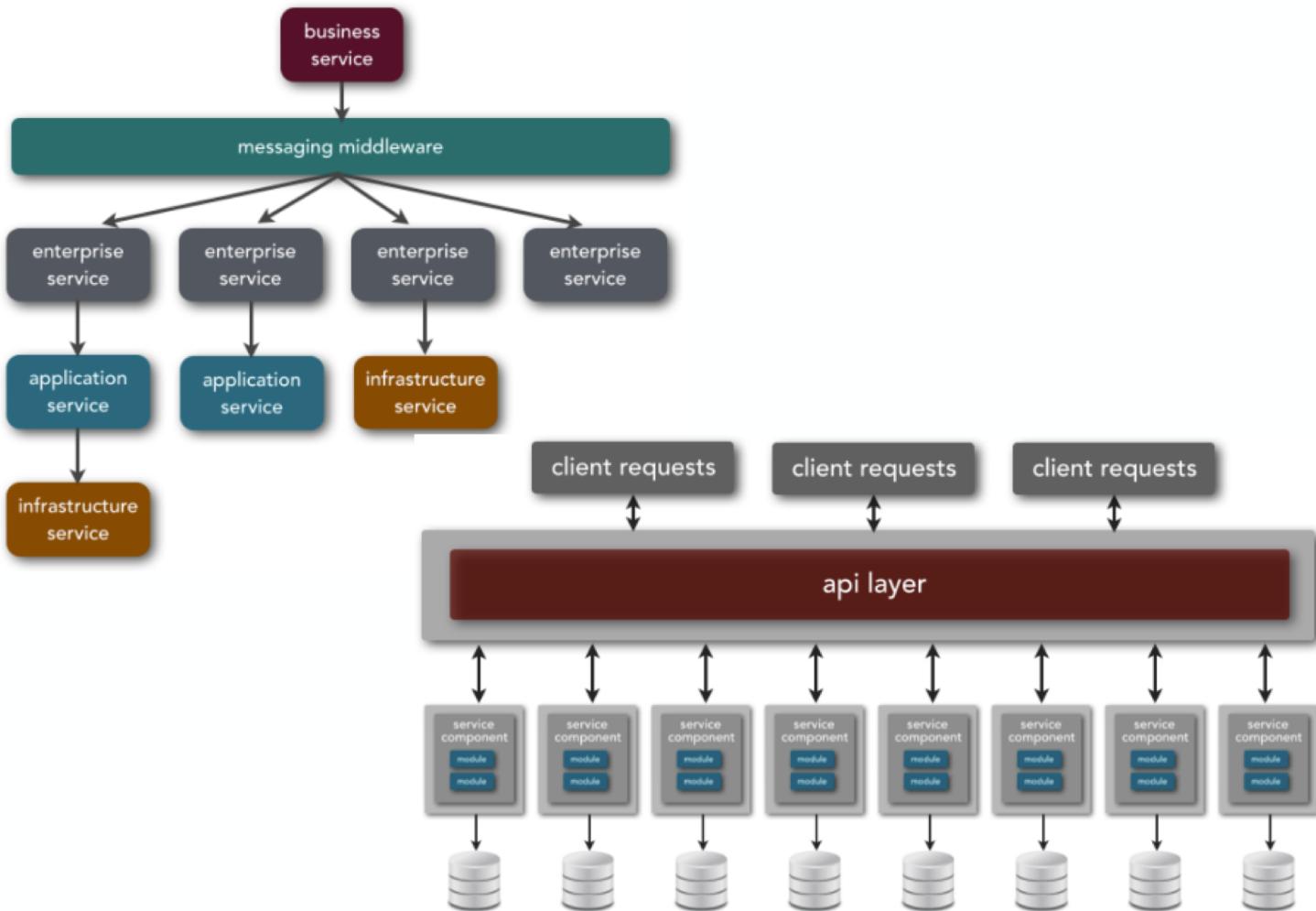
2010's

## MICROSERVICES ARCHITECTURE



**Microservices** are a number of independent application services delivering one single functionality in a loosely connected and self-contained fashion, communicating through light-weight messaging protocols such as HTTP, REST or Thrift API.

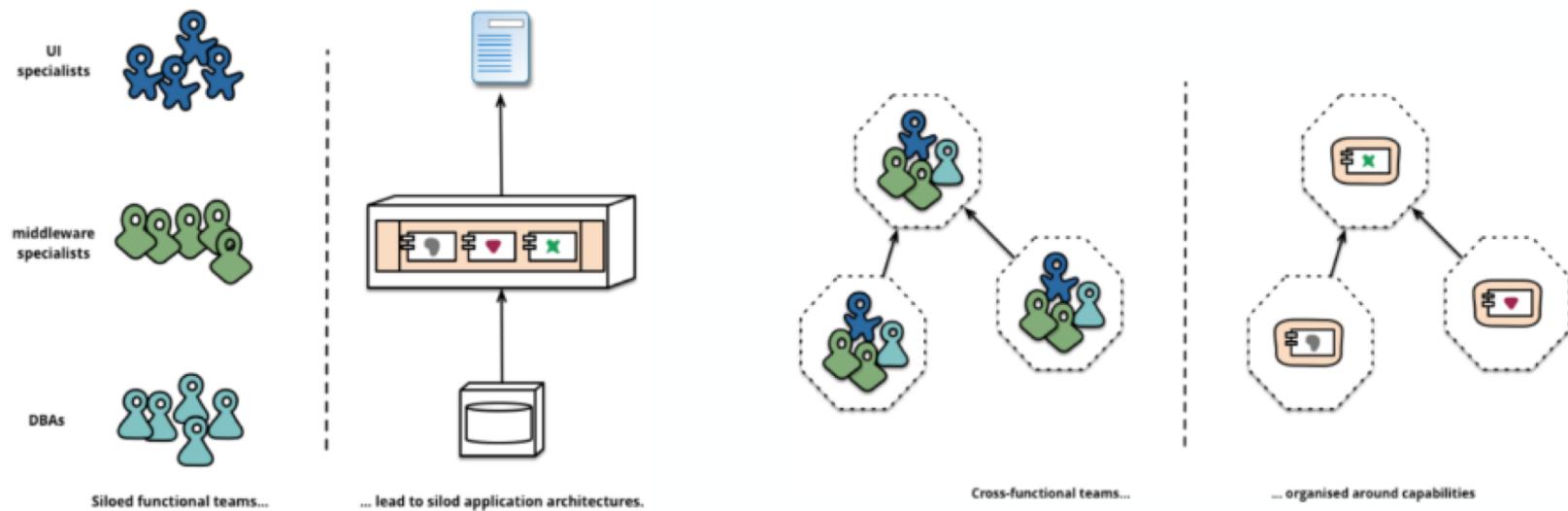
# Microservicios vs SOA



# Tipo de Microservicios

- **Servicios funcionales:** se exponen hacia fuera de la organización
- **Servicios de infraestructura:** Para uso interno de los otros servicios
  - Autenticación y autorización
  - Monitoreo
  - Logging y auditoría

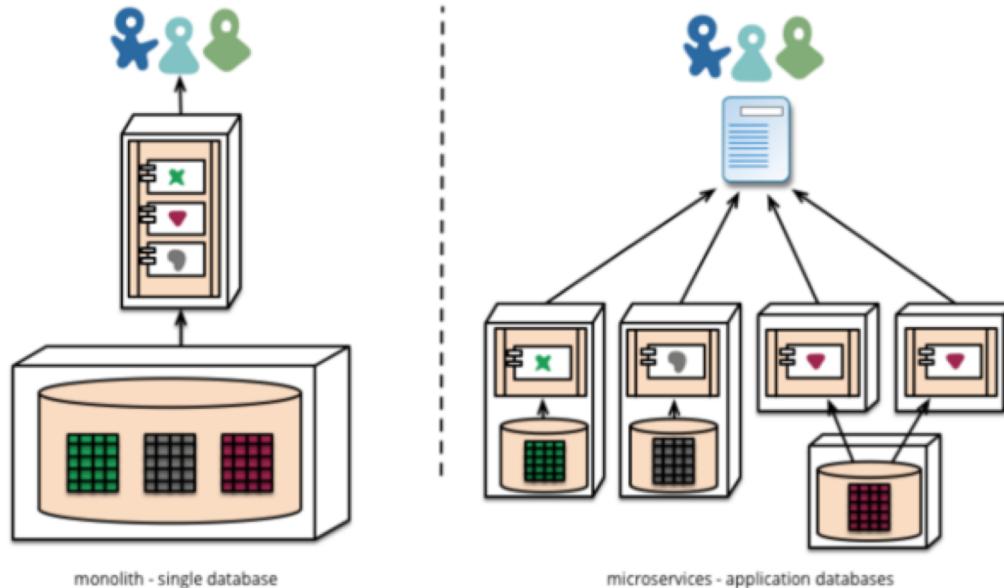
# Equipos en microservicios



**Clásico: front-end, back-end, DB**

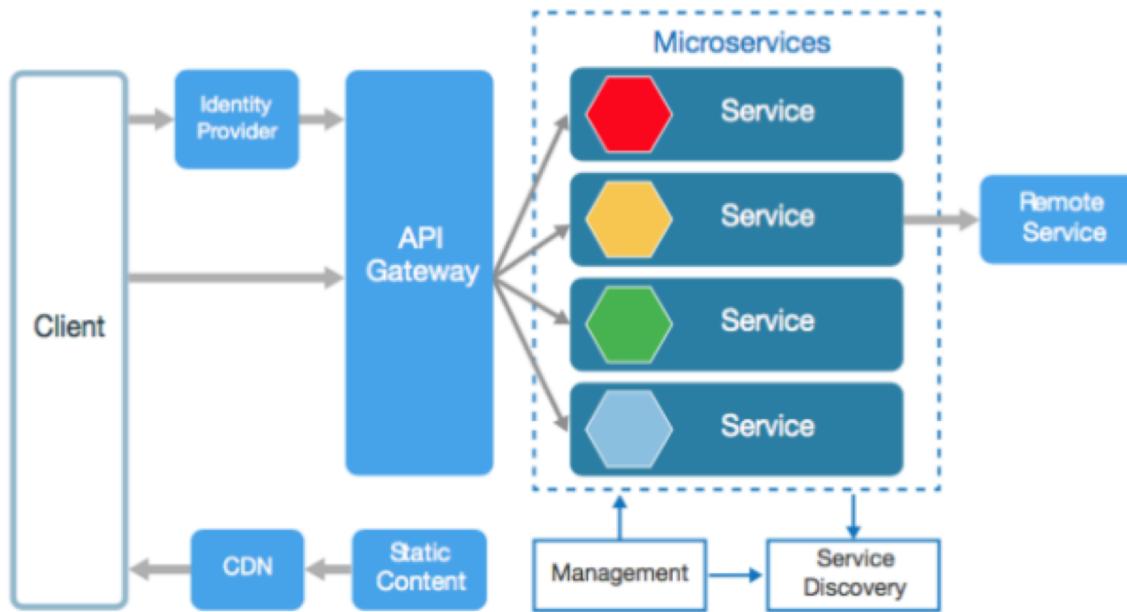
**Microservices: equipos multifuncionales**

# Manejo de datos descentralizados



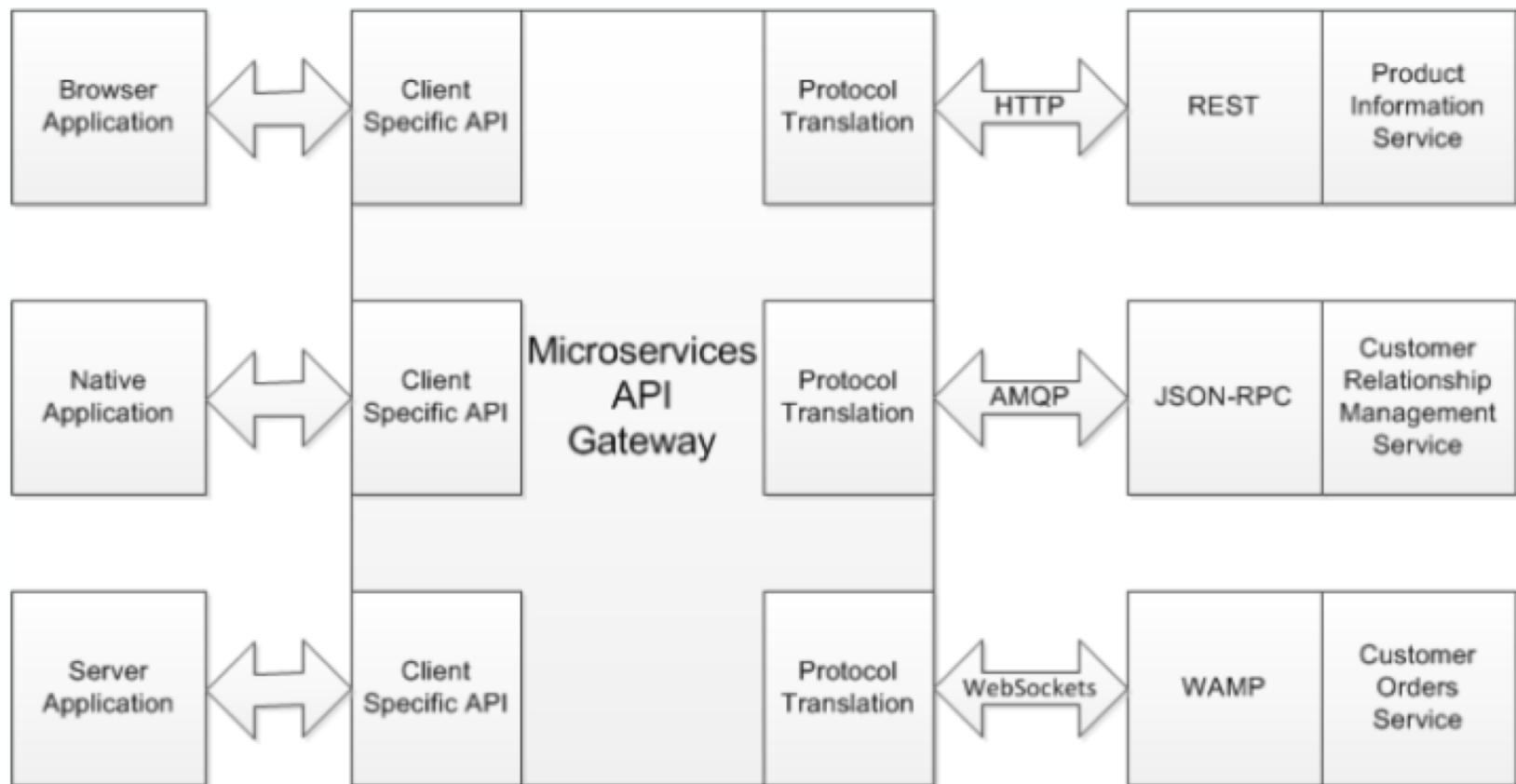
- Cada servicio maneja su propia DB
- No se usan transacciones para coordinar, consistencia eventual
- Respuesta rápida y escalabilidad se paga con posibles grados de inconsistencia temporal

# El API Layer



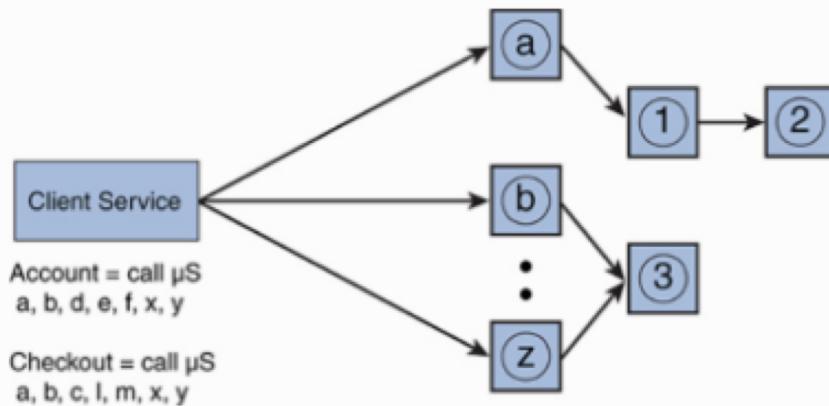
- Lo que se expone a clientes
- Puede agregar respuestas de varios servicios
- Permite cambiar interfaz de servicios
- Servicios pueden utilizar protocolos que no sean HTTP

# API Gateway

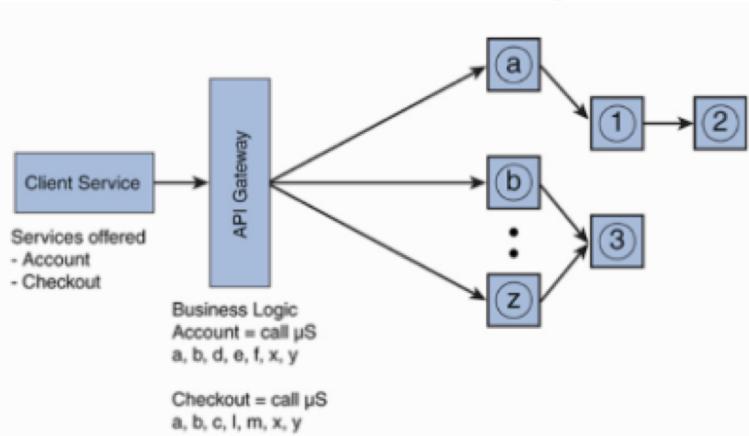


# Variaciones

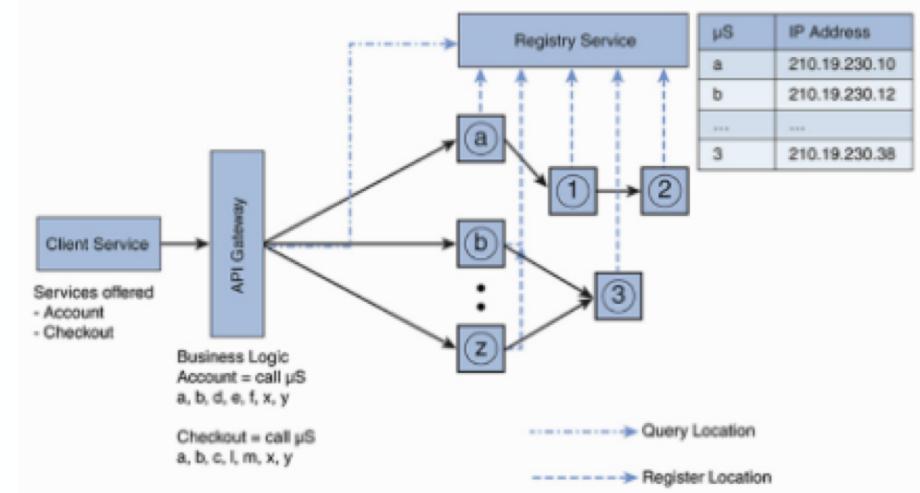
## Sin API Gateway



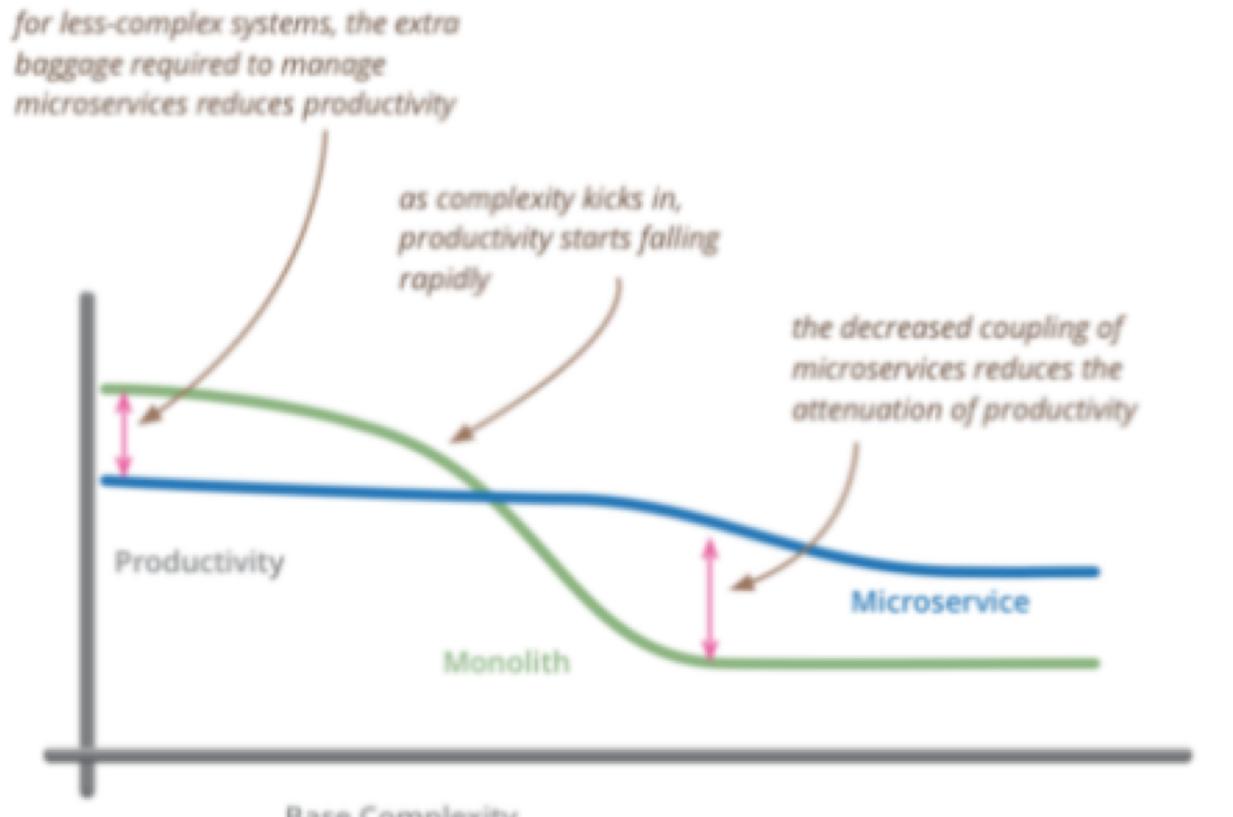
## Con API Gateway



## Con API Gateway y Registry



# No es la panacea



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



Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación

# Clase 18

# Arquitectura de Servicios

IIC2143 - Ingeniería de Software  
Sección 1

Rodrigo Saffie

[rasaffie@uc.cl](mailto:rasaffie@uc.cl)

23 de mayo de 2018