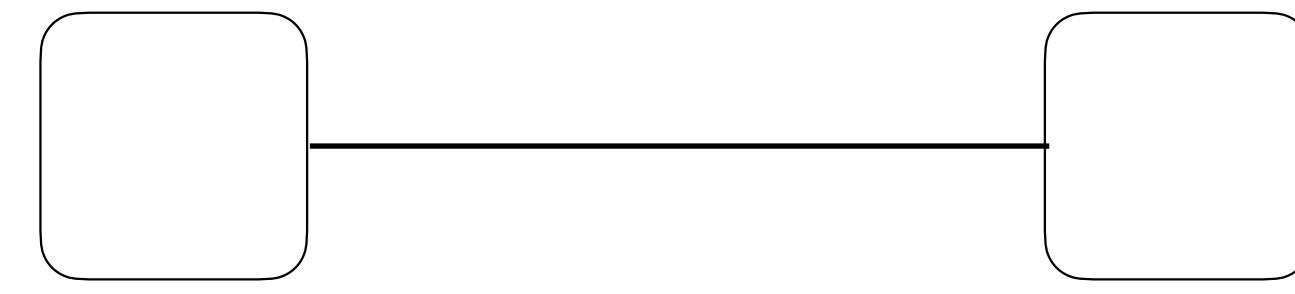


Patrones Arquitecturales

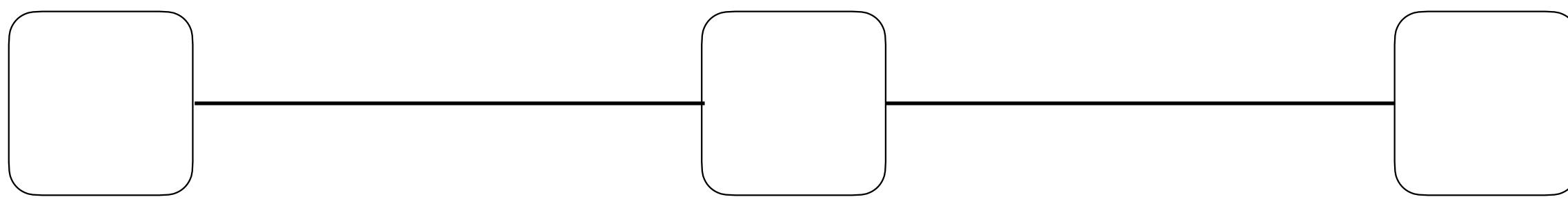
Luis Daniel Benavides Navarro, Ph.D.

18-9-2019

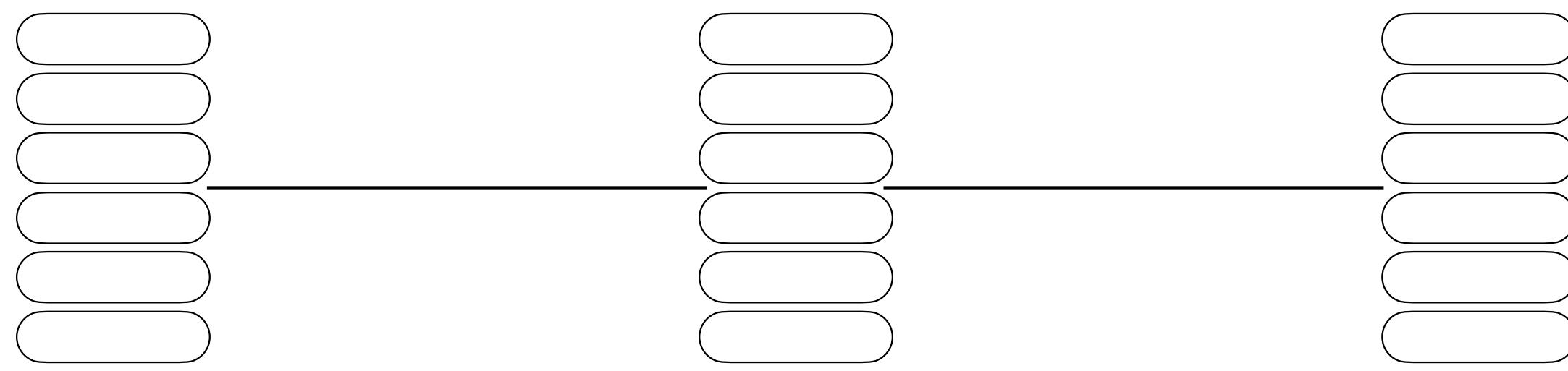
Complejidad



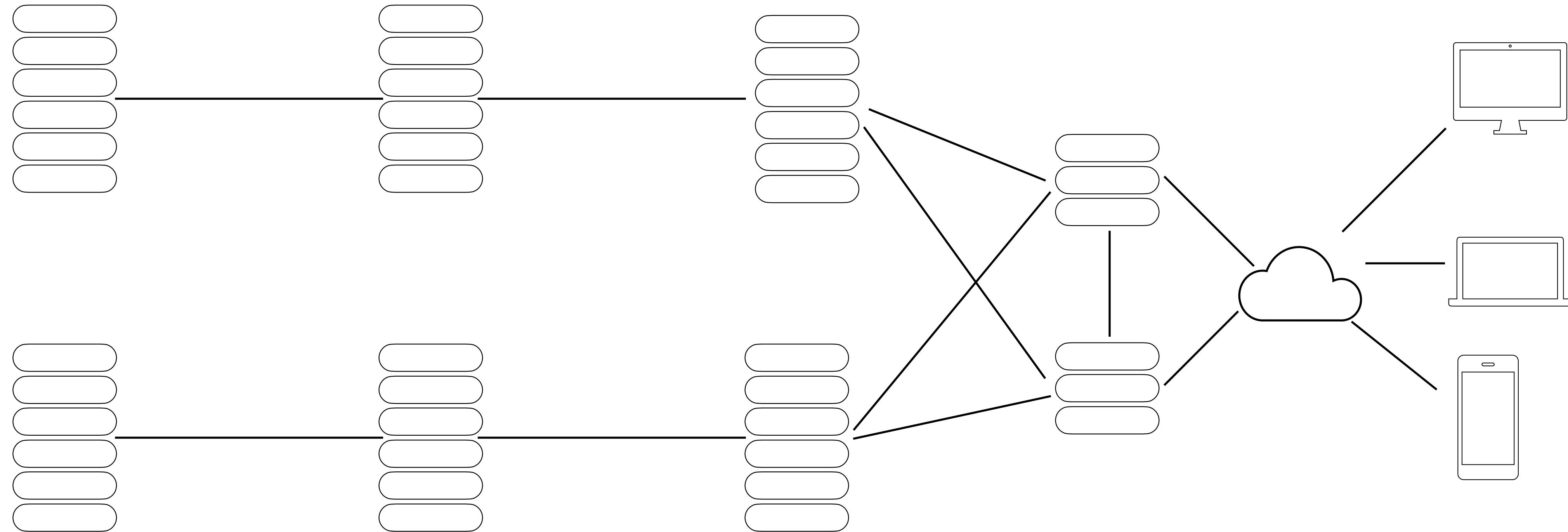
Client - Server



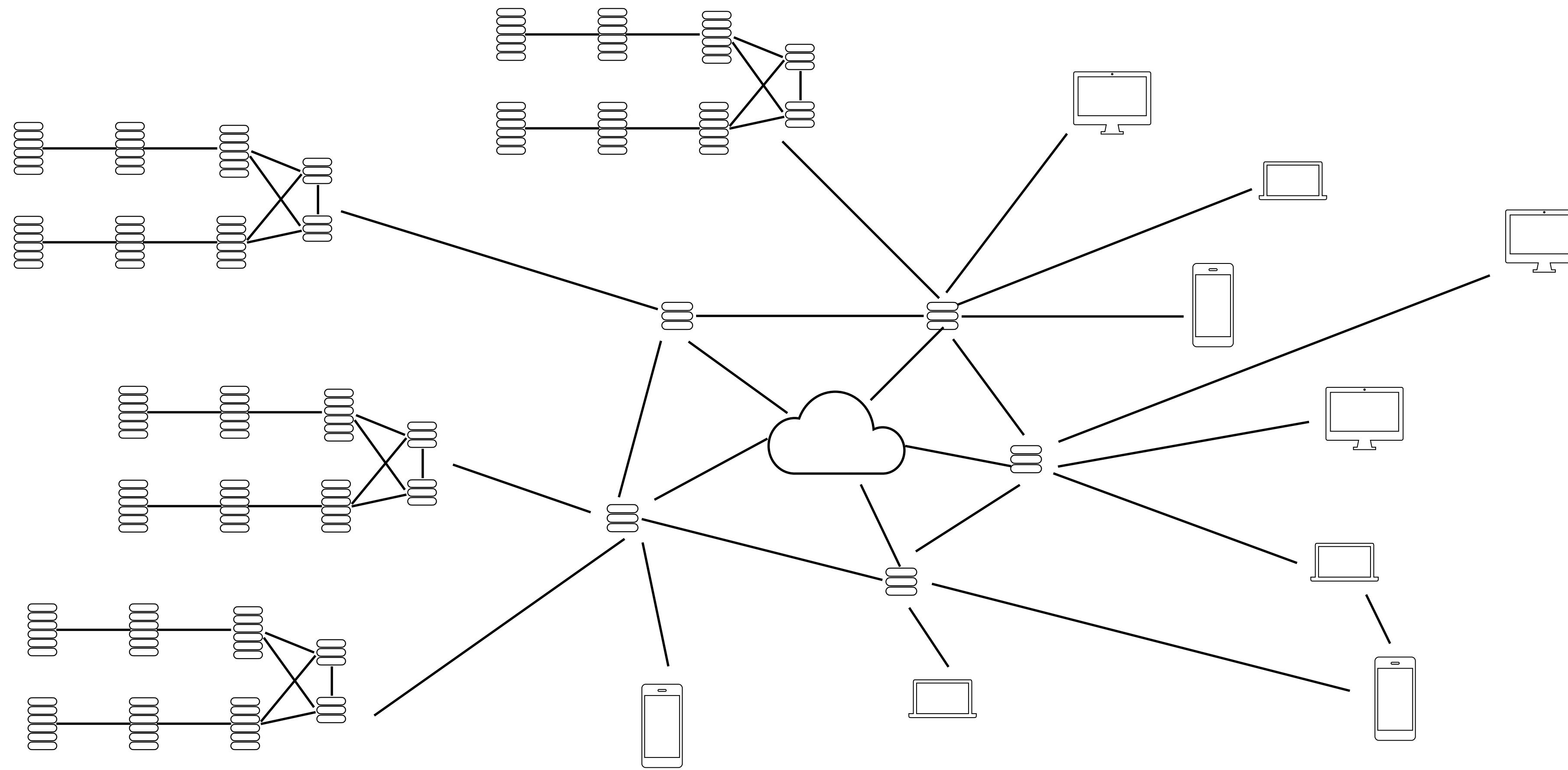
Multi Tier



Elastic/Clustered multi tier



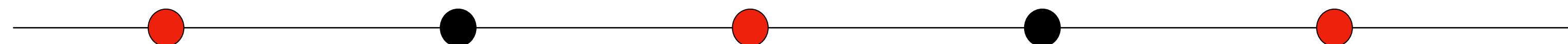
Redundant, Multitenant, load balanced, and elastic multi tier deployed on the cloud



Global applications with complex cloud topologies

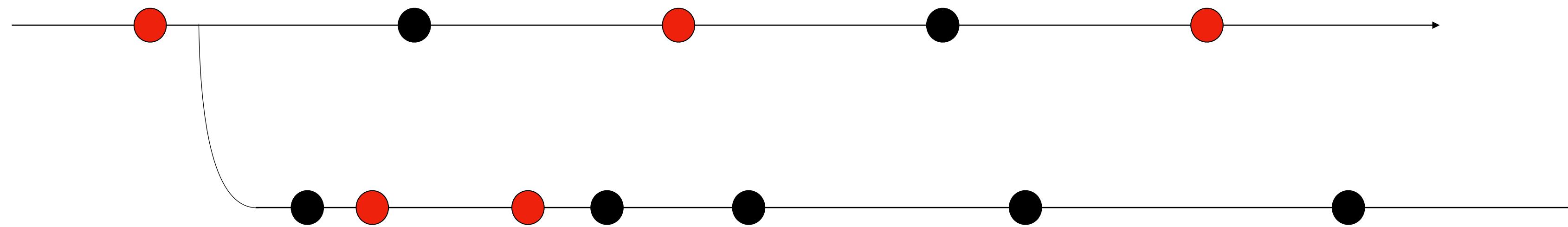
**Design, Implementation, maintenance
and evolution of distributed
applications is a complex task**

time →



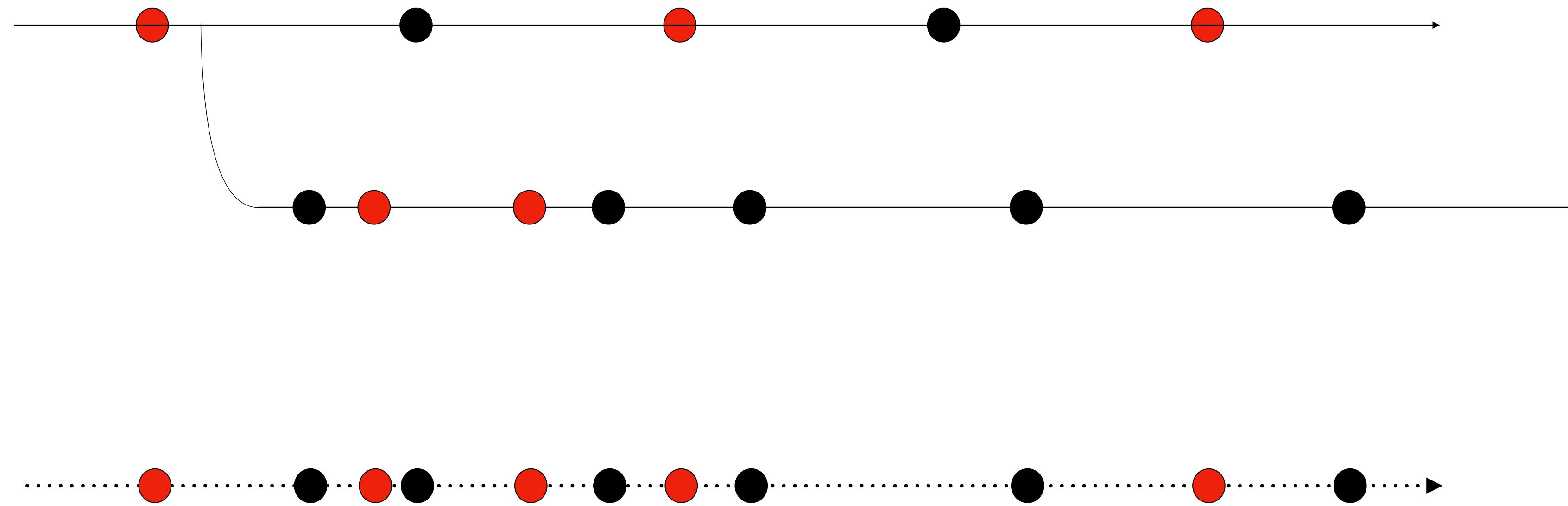
Sequential application in one computer

time →

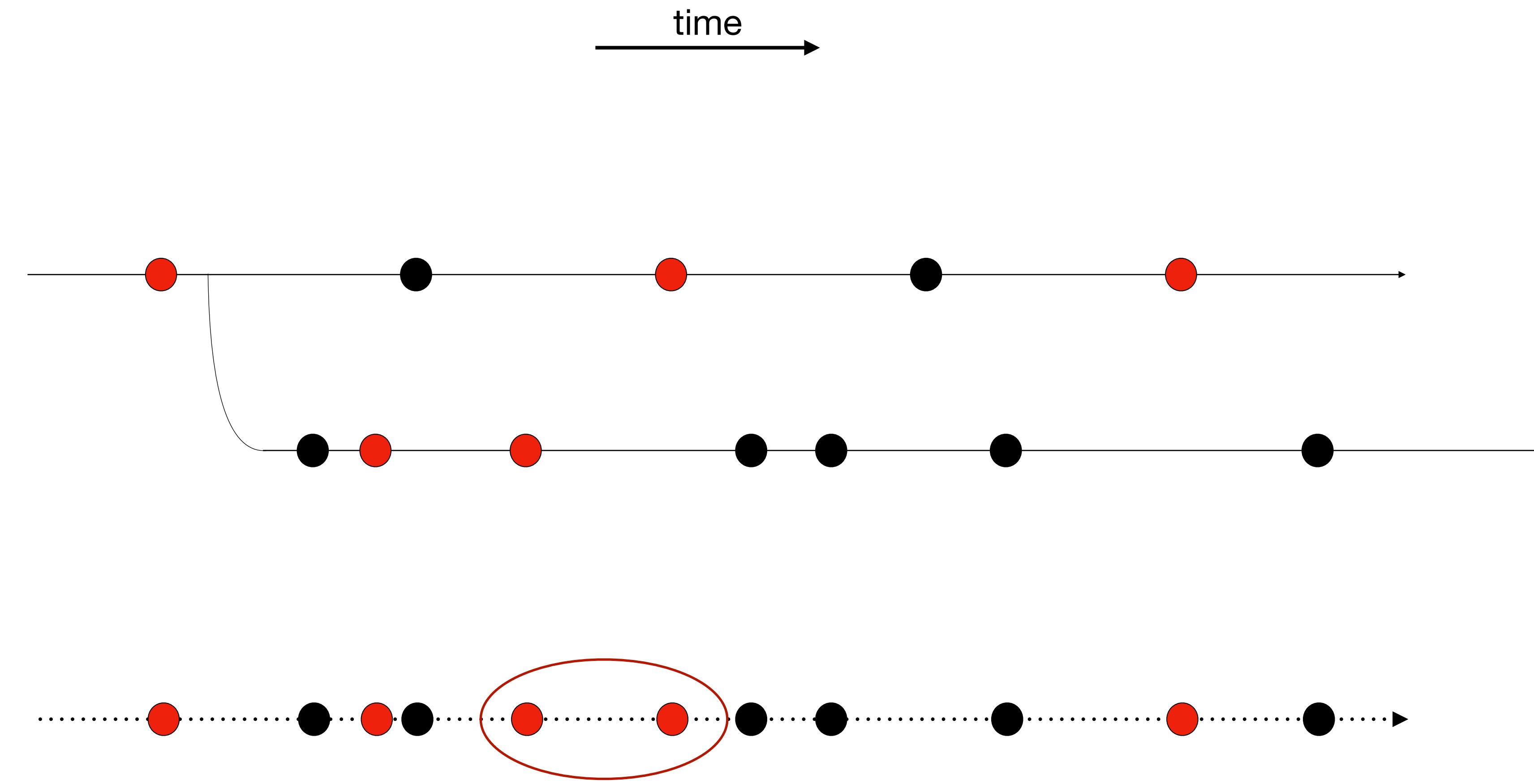


Multi threaded application in one computer

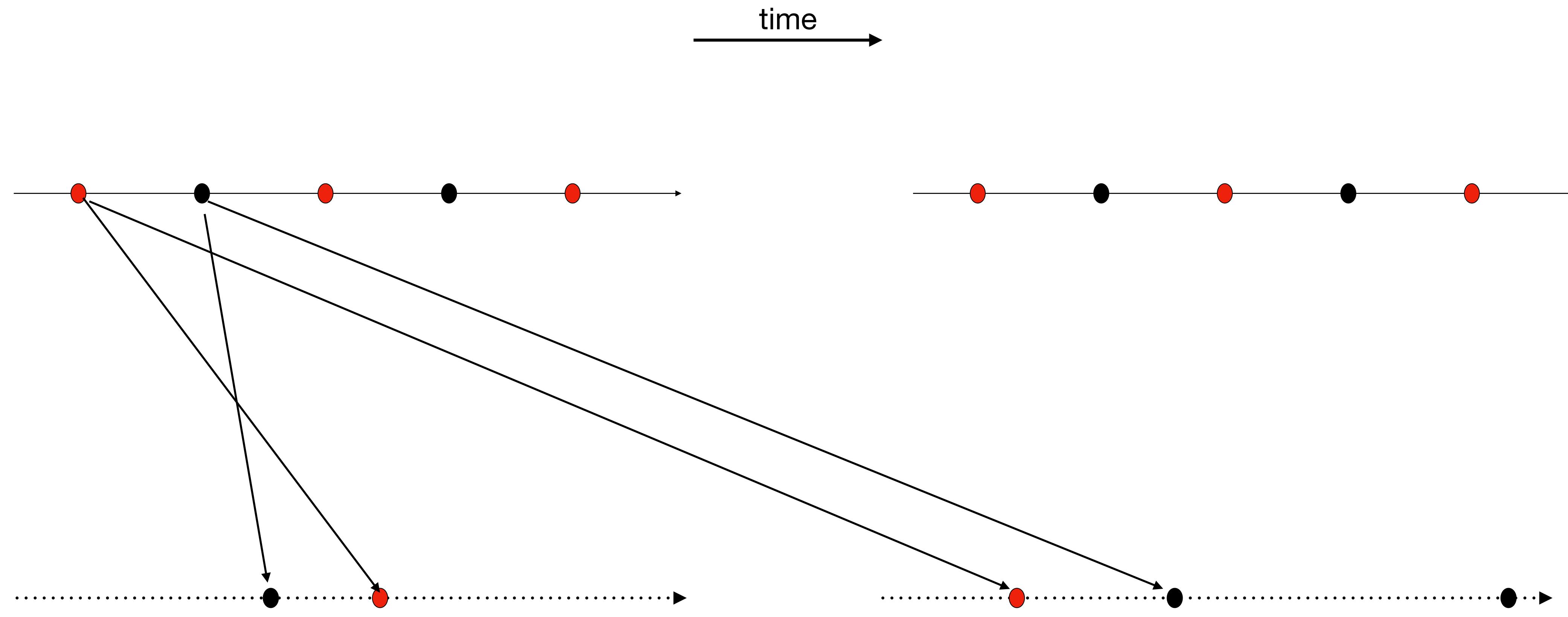
time →



Correct interleaving of events



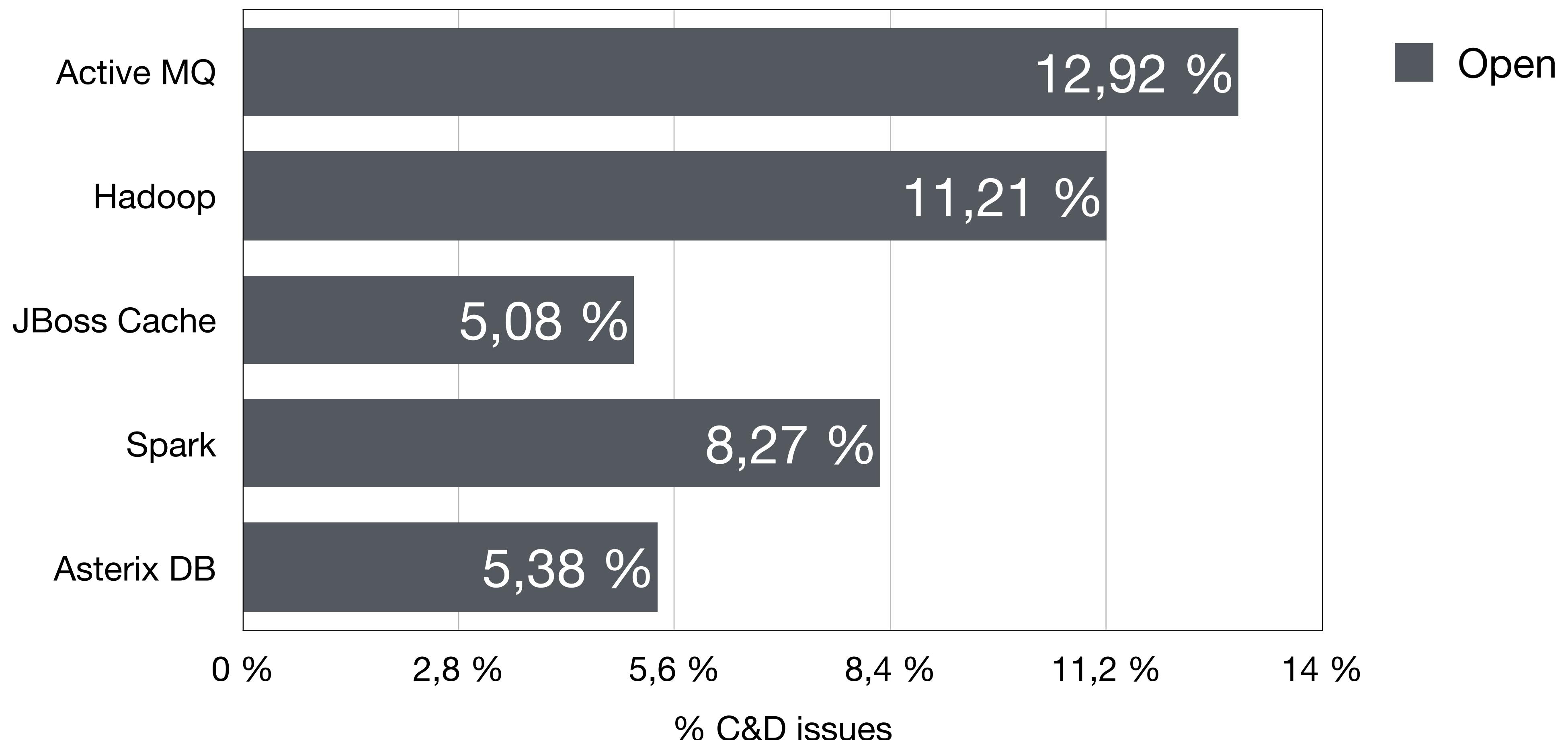
Unexpected and erroneous interleaving of events



Unexpected and erroneous interleaving of events

Correct interleaving of events

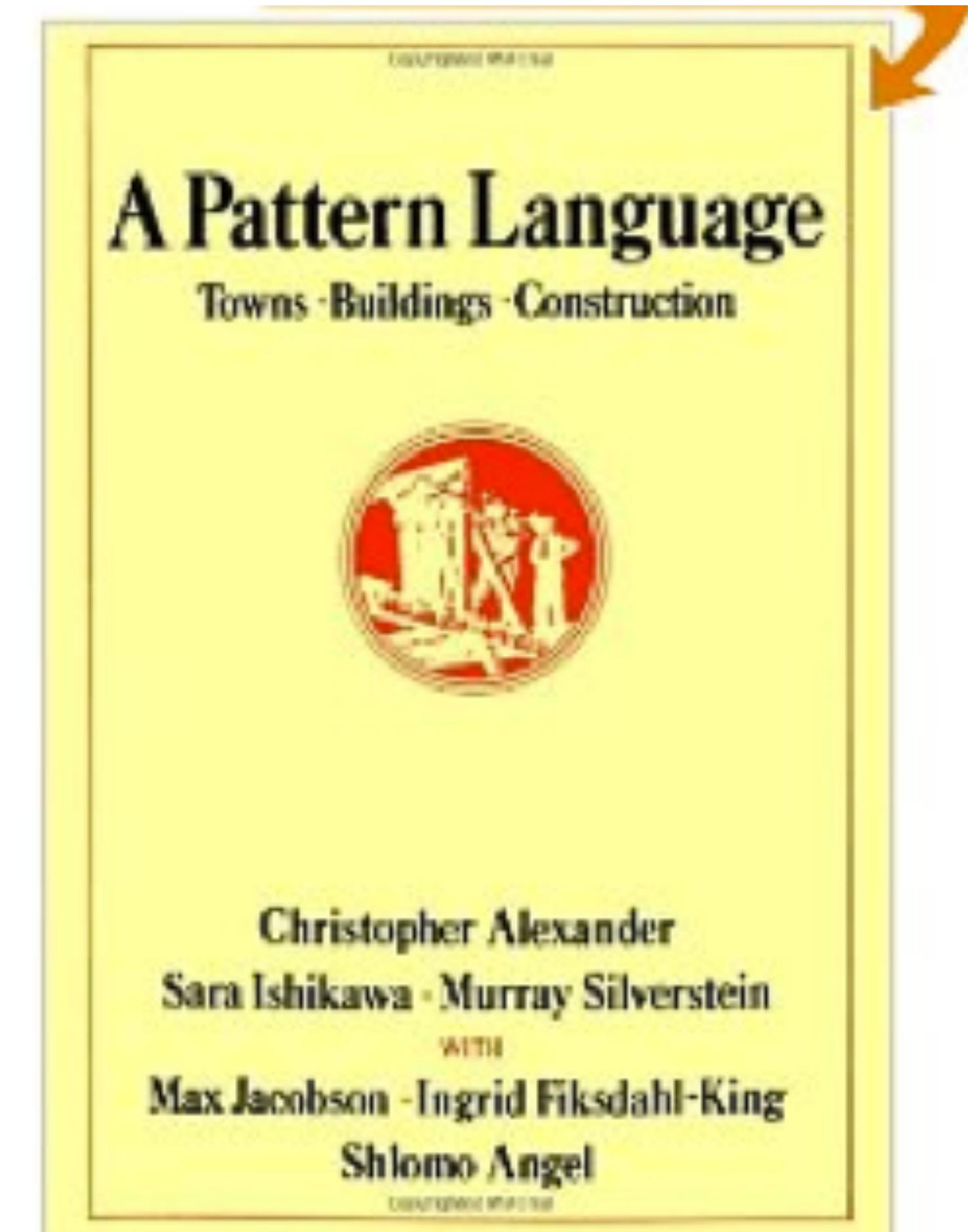
Percentage of concurrent and distributed issues still open in selected distributed application



Lenguaje de Patrones

A pattern Language

- A Pattern Language by Christopher Alexander (1977)
 - Lenguaje de patrones arquitecturales para construir casas, barrios, pueblos y ciudades.

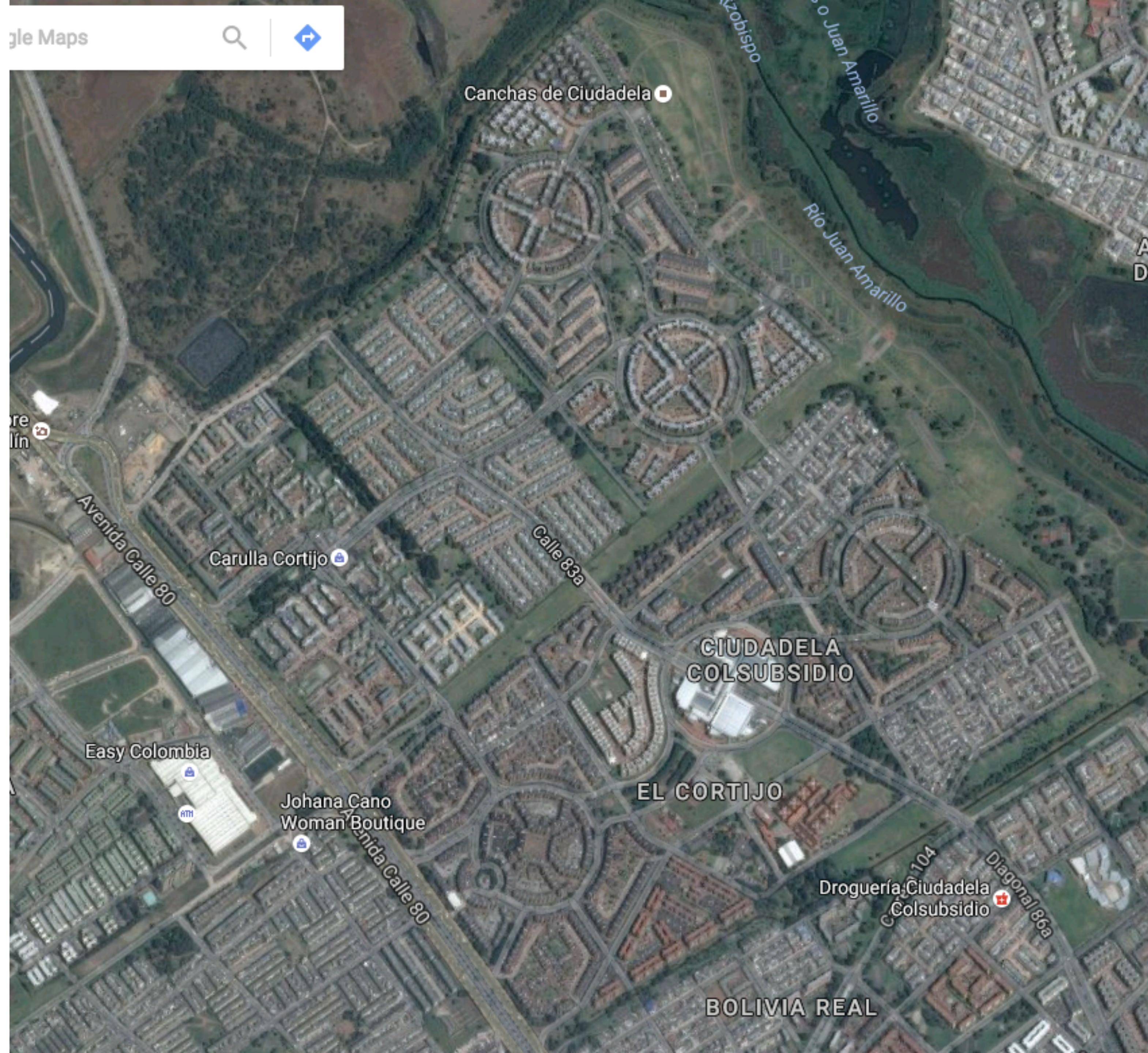


Patrones

- Un patrón describe un problema que ocurre una y otra vez en nuestro entorno, y luego describe la esencia de la solución al problema, de una manera que la pueda utilizar miles de veces sin nunca repetir la misma solución.
- Tienen un estructura genérica
 - Desc. Problema
 - Rel. con otros patrones
 - Descripción de la solución
 - Ejemplos de aplicación

Ejemplo del lenguaje de patrones

- Regiones independientes
 - La distribución de los pueblos
 - Dedos de ciudad y campo
 - Valles de agricultura
 - Lazos de calles veredales





Patrones arquitecturales

Creadores y servicios (Distribución)

- Sistemas complejos son en general distribuidos
- Distribución sirve para manejar complejidad
- Técnicas de comunicación
 - Base de datos
 - RPC
 - Archivos
 - Mensajería



The Earth Simulator in Yokohama was the world's fastest supercomputer in 2004, but 7 years later the K computer in Kobe became over 60 times faster. [wikipedia.org](https://en.wikipedia.org)

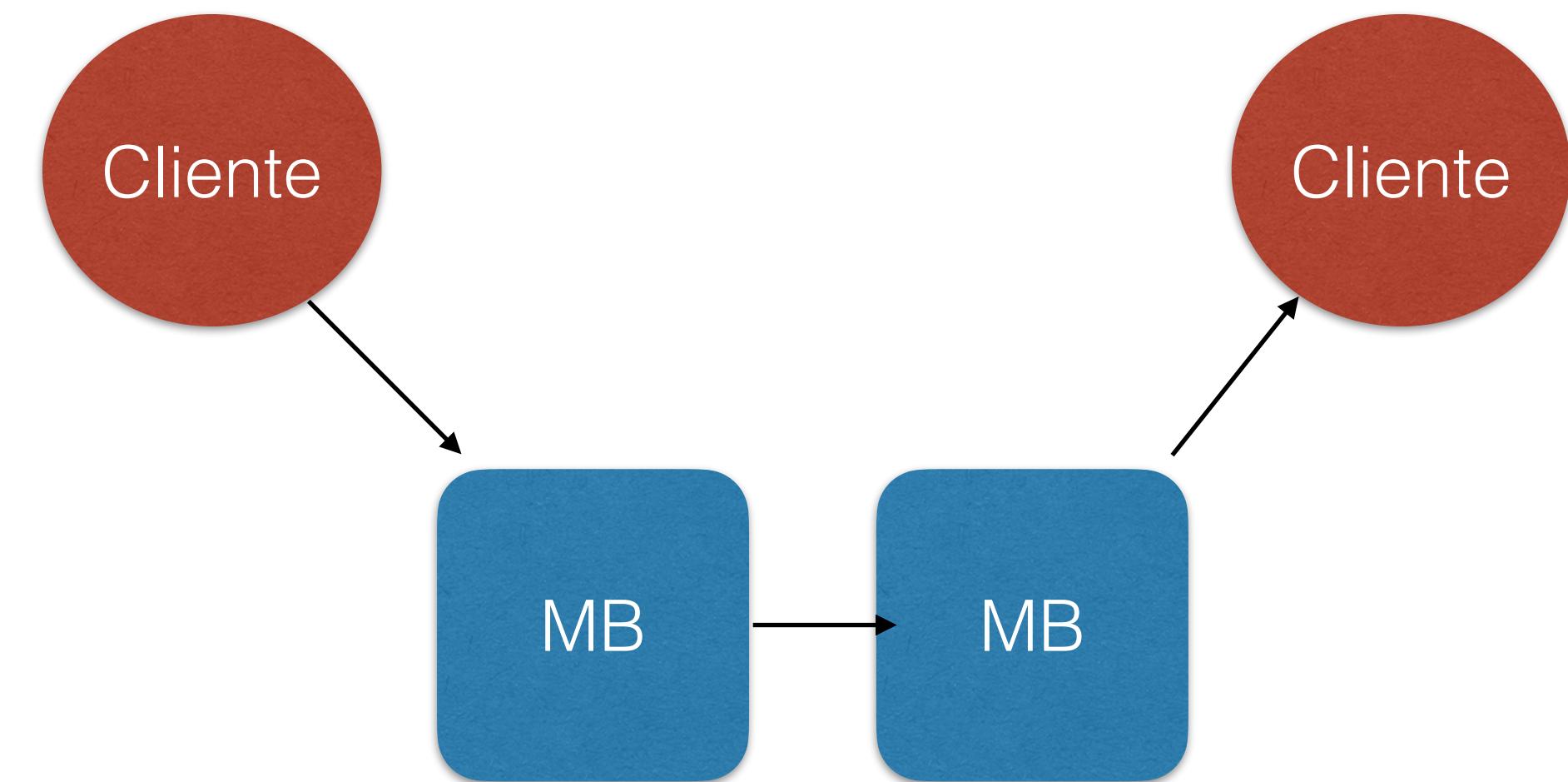
Cliente Servidor

- Clientes se conectan a un servidor que maneja la aplicación.
 - Cliente
 - Red
 - Server
- Comunicación de diferentes maneras



Mensajería

- Intermediario confiable y eficiente: Message Broker
- Conceptos fundamentales
 - Send and Forget
 - Store and forward



Multi capas

- Estructura de sistemas jerárquica
- Con capas adyacentes que solo se comunican entre ellas
- Los framework empresariales generalmente las promueven, e.g., JEE

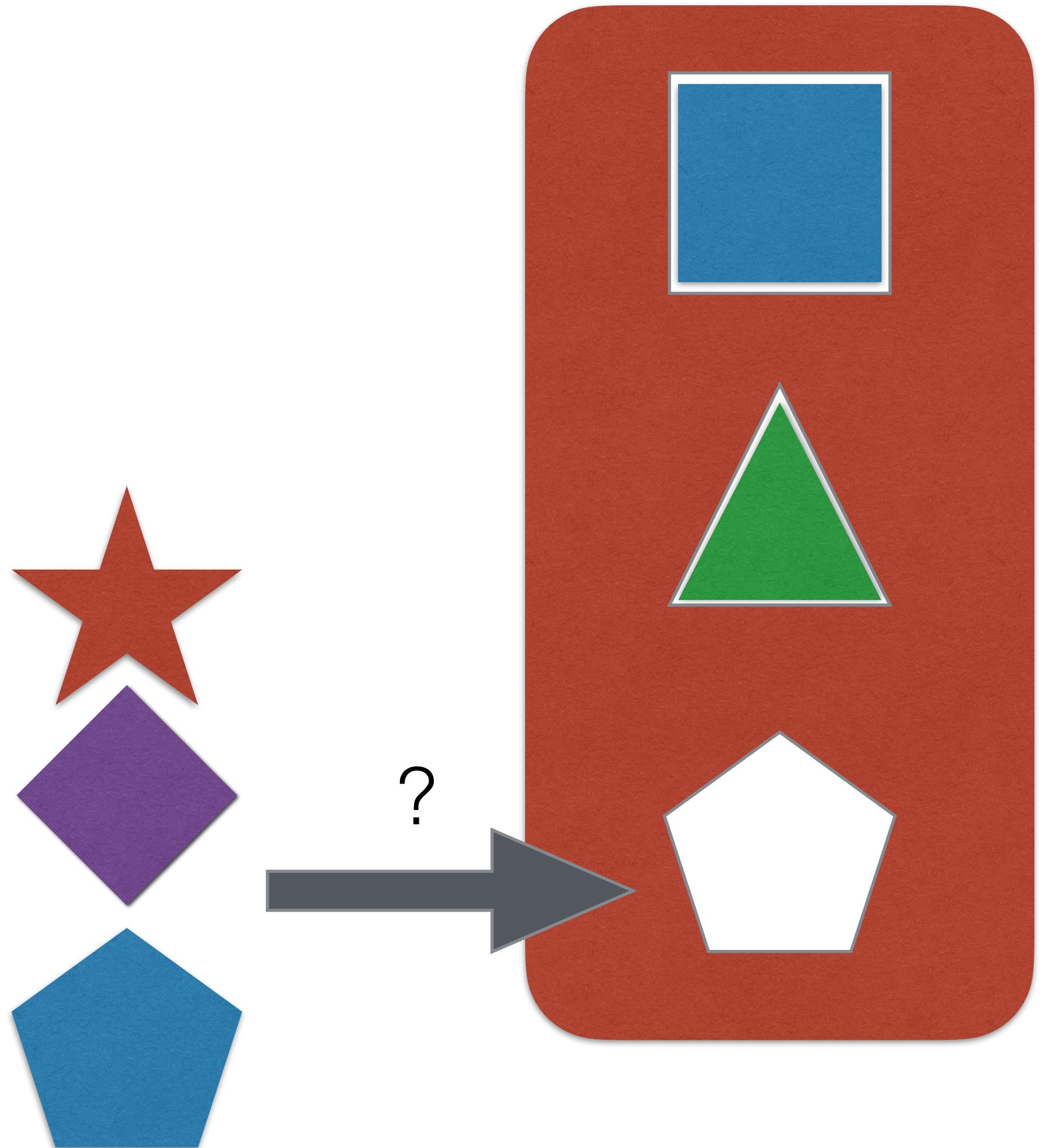
Presentación

Lógica de negocio

Datos

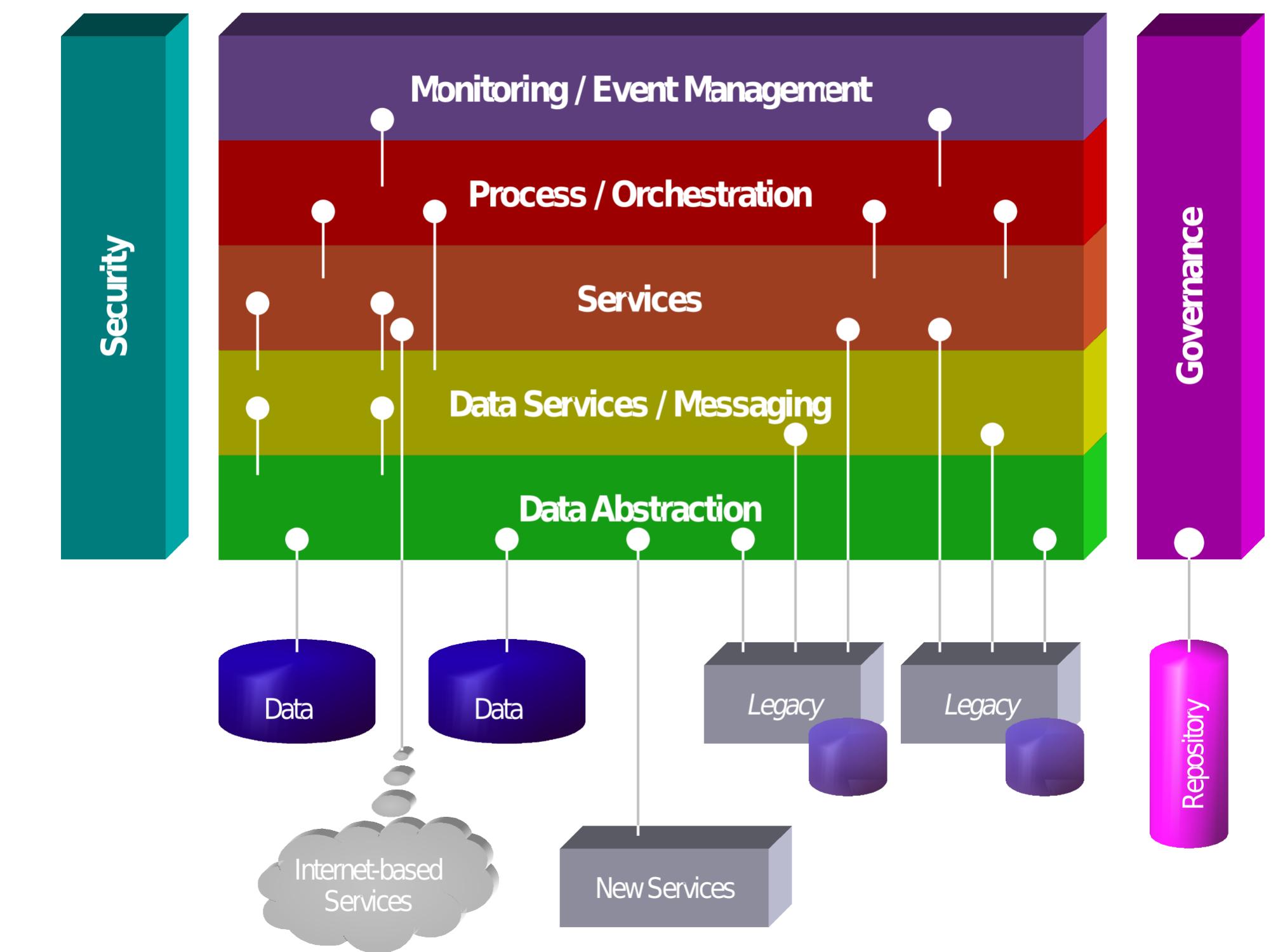
Componentes

- Encapsular funcionalidades en componentes autónomos e independientes.
- Construir aplicaciones al estilo lego
- Se comunican por diferentes medios
- Se refuerzan los contratos de la interfaz
- No son remotos necesariamente



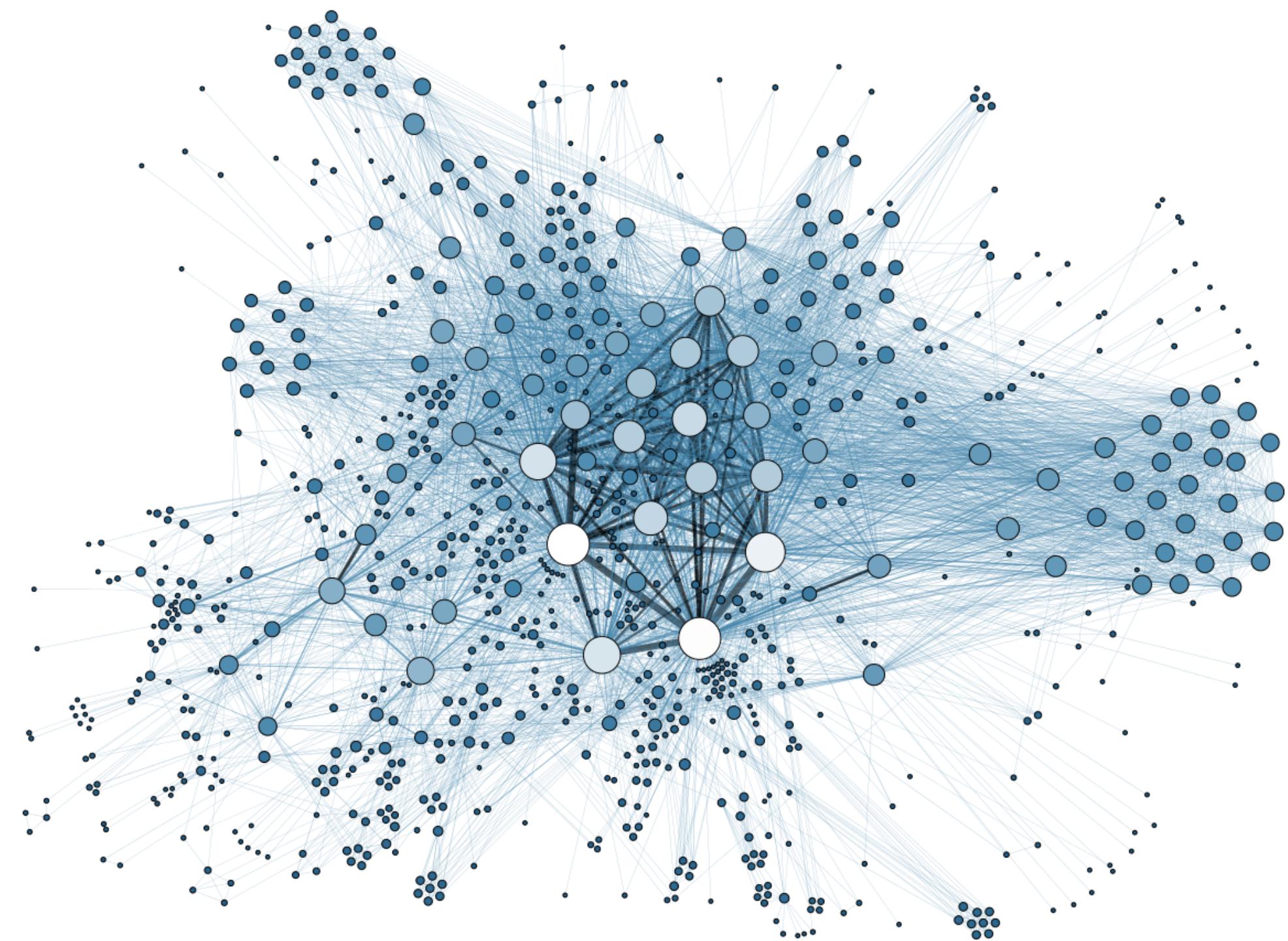
SOA

- Elemento esencial y atómico: servicio
- Estructurada con servicios atómicos distribuidos
- Bus de comunicación asume otras responsabilidades
- Facilita desarrollo, y creación de sistemas complejos
- Video: https://www.youtube.com/watch?v=jp41M_V1VnE



Microservicios

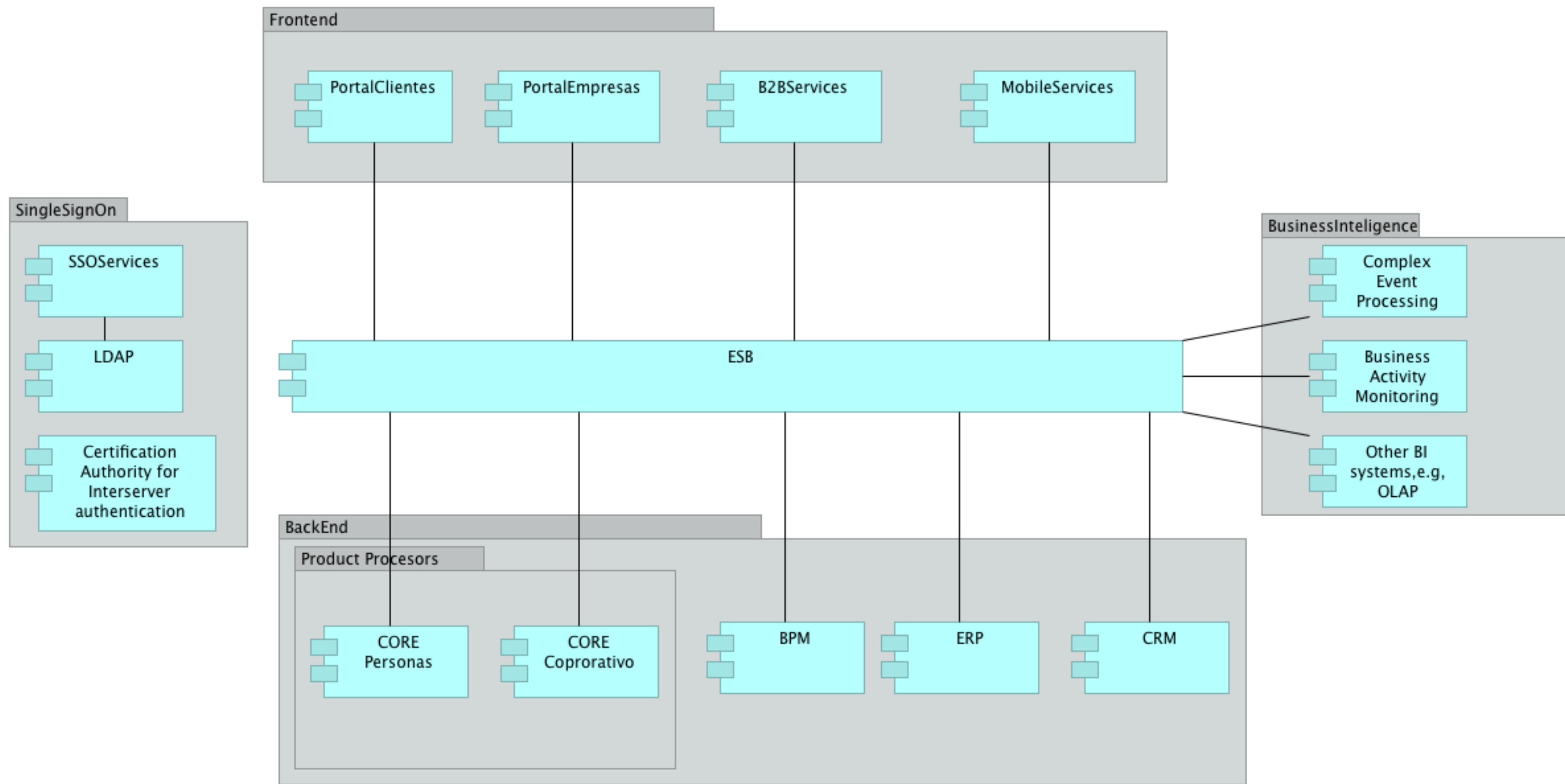
- **Microservicios:** atómicos, independientes
- **Servicios se comunican** via protocolos síncronos como HTTP/REST o protocolos asíncronos como AMQP.
- Servicios **desarrollados y desplegados** de manera independiente, desde la presentación hasta la data
- Cada servicios tiene su **propia persistencia**
- **Consistencia** se maneja con eventos o replicaron de base de datos
- **Escalan** de manera independiente y según lo necesitan
- **Diseñados para comunicarse** con clientes web, clientes móviles, otras aplicaciones, otros servicios
- **Desarrollo promociona** ideas de continuous integration, continuous delivery, devops, etc.



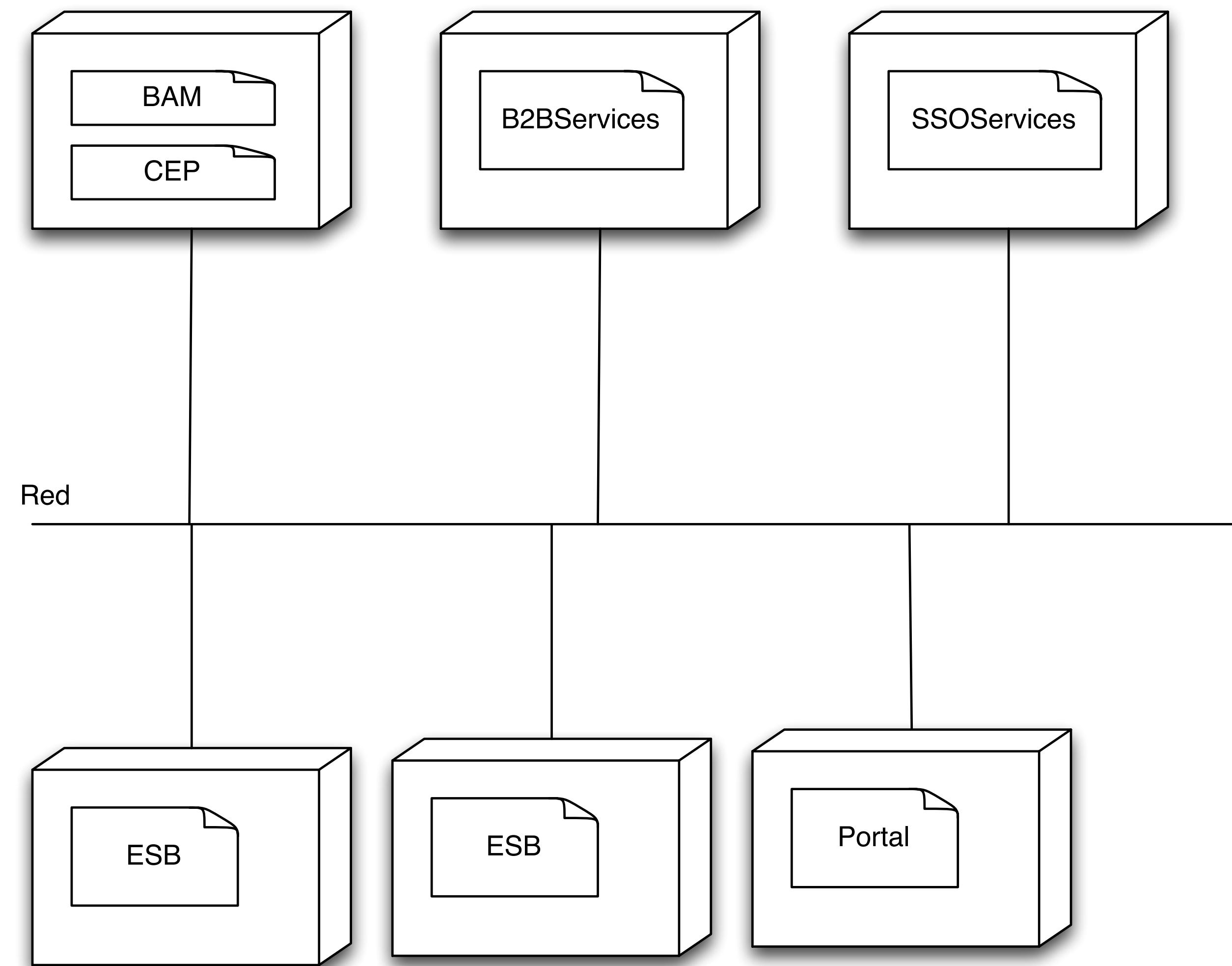
Pregunta:
¿Cuál es la diferencia entre
Monolito y Microservicios?

Ejemplo: Una Arquitectura SOA Empresarial

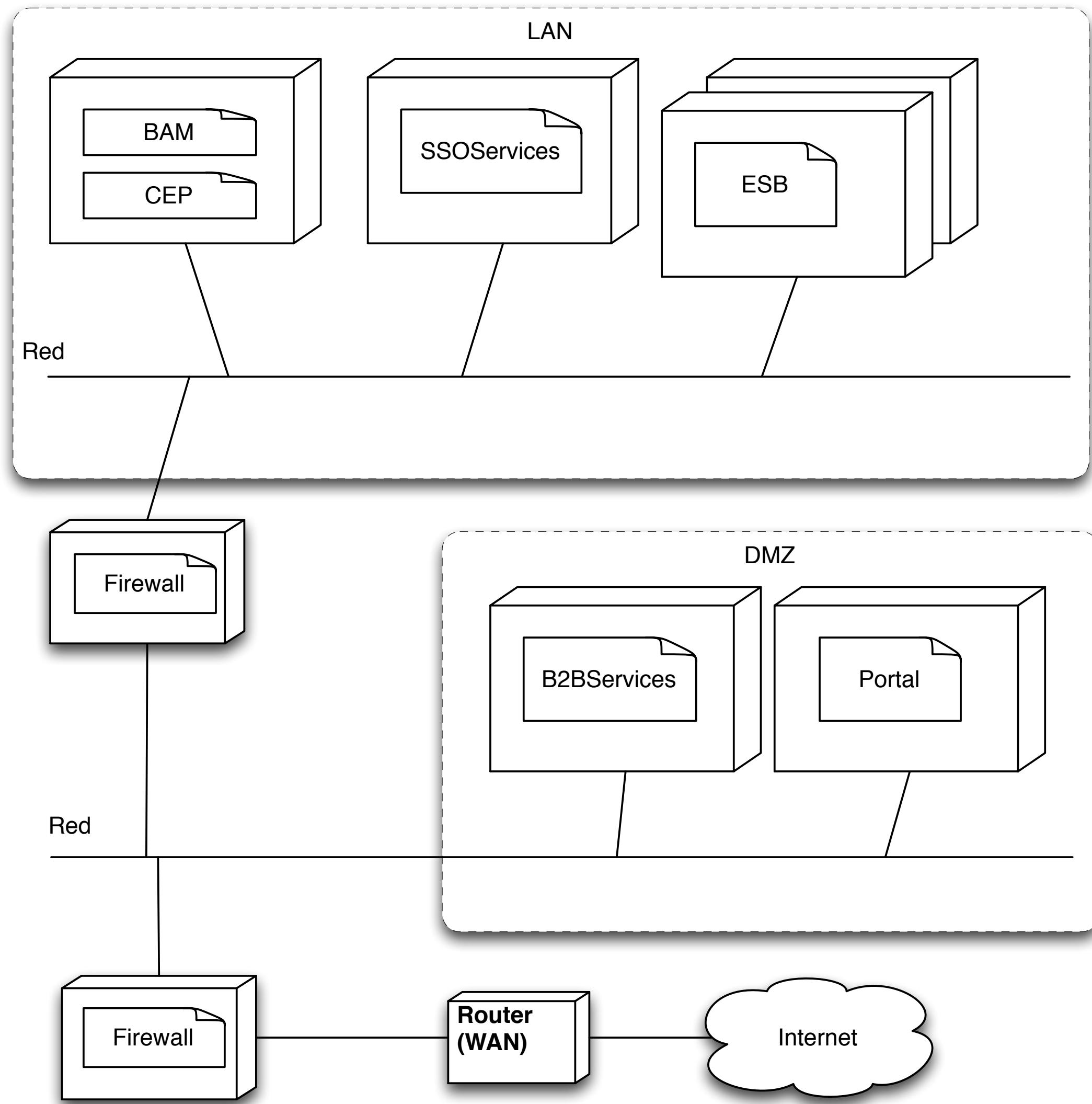
Arquitectura SOA



¿Cómo se ve SOA físicamente?

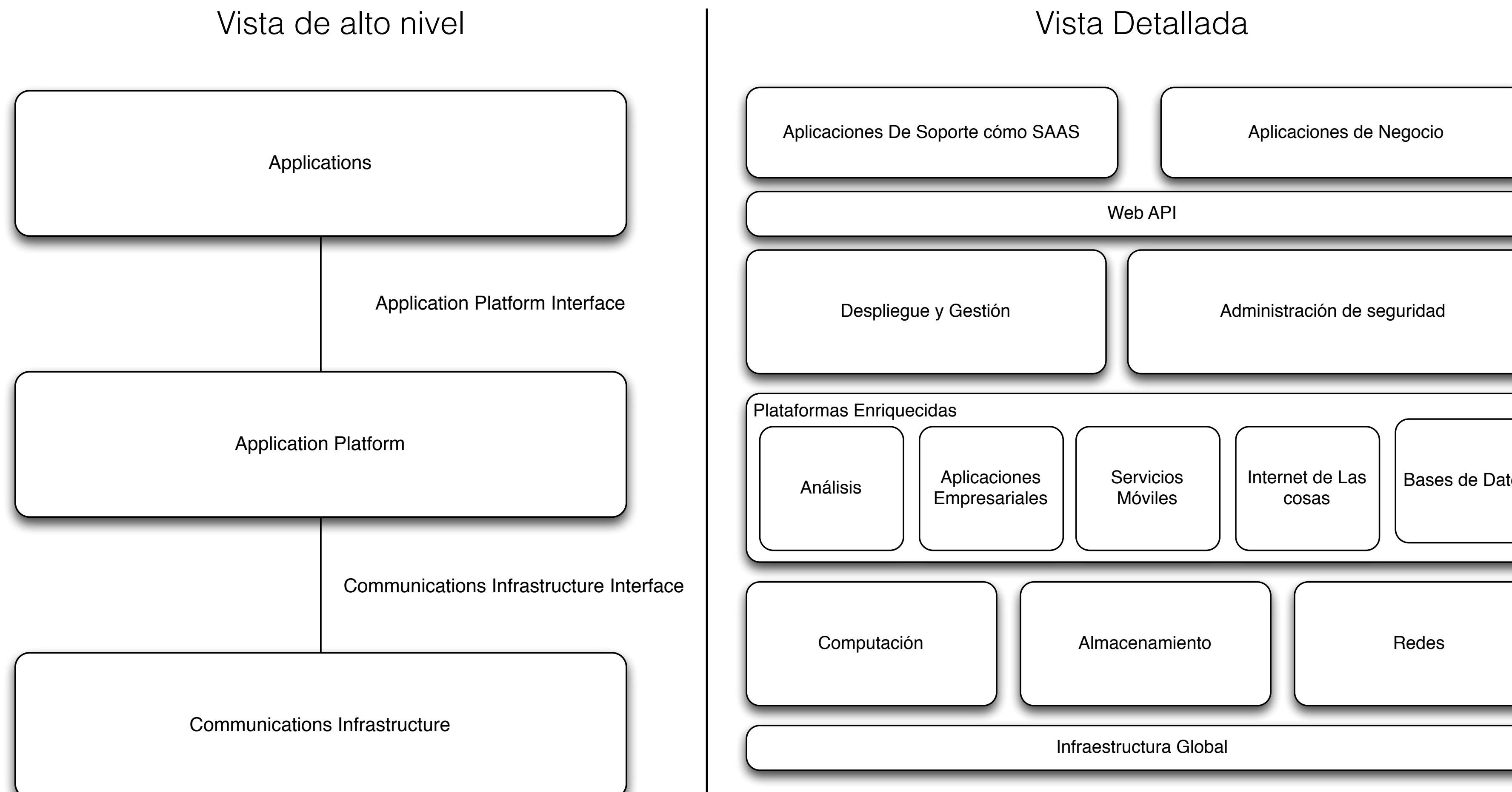


Otra alternativa



La Arquitectura de la nube

Arquitectura de referencia cloud



Diseñando sistemas modernos

Clients pesados: xxx-JS



```
1. <!doctype html>
2. <html ng-app>
3.   <head>
4.     <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.5/
5.       angular.min.js"></script>
6.   </head>
7.   <body>
8.     <div>
9.       <label>Name:</label>
10.      <input type="text" ng-model="yourName" placeholder="Enter a name her
11.        e">
12.      <hr>
13.      <h1>Hello {{yourName}}!</h1>
14.    </div>
15.  </body>
16. </html>
```

Name:

Hello !



Backend alternatives

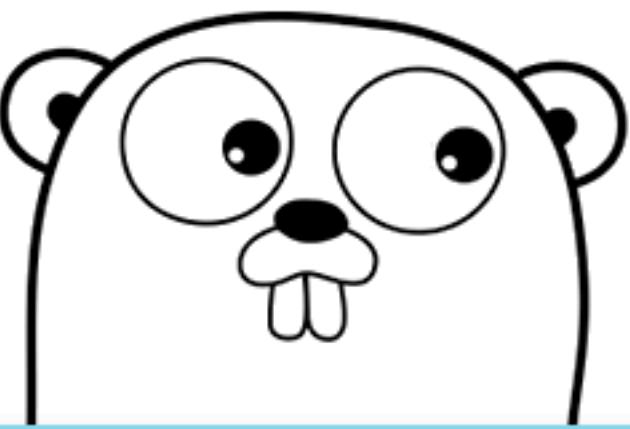


Documents

Packages

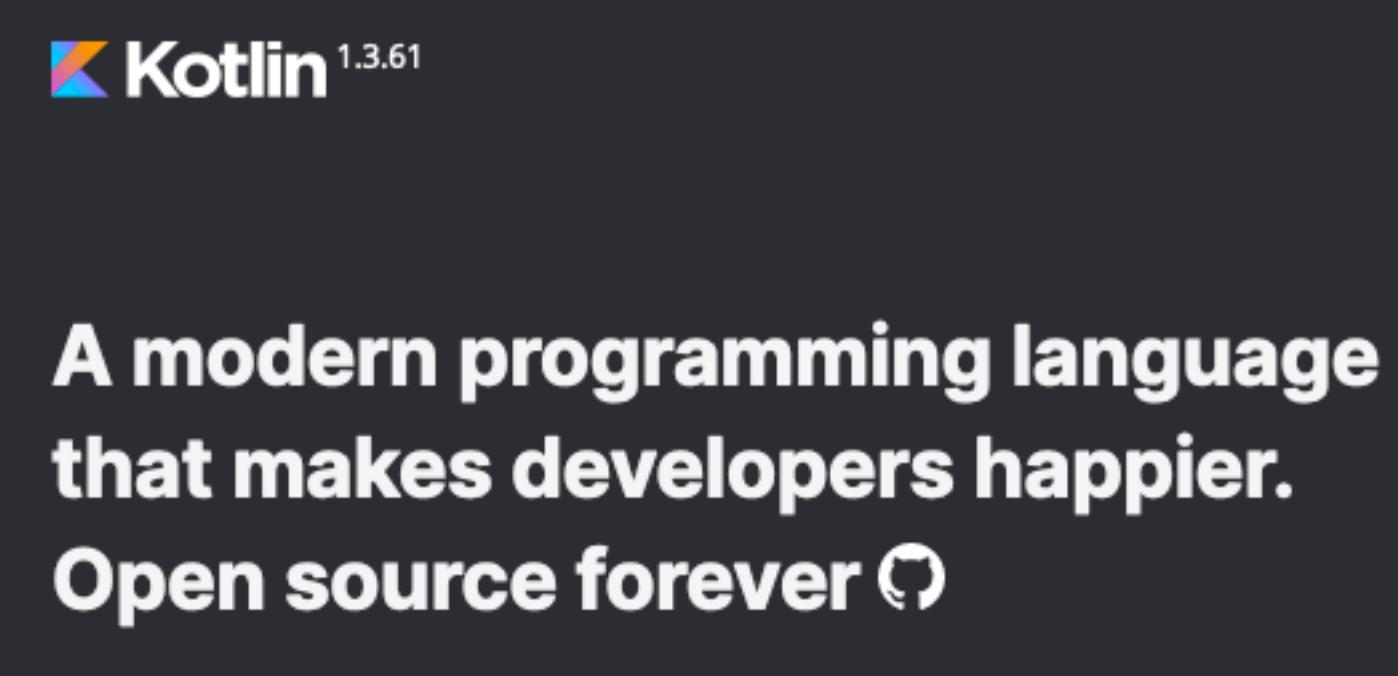
T

Go is an open source programming language that makes it easy to build simple, reliable, and efficient software.



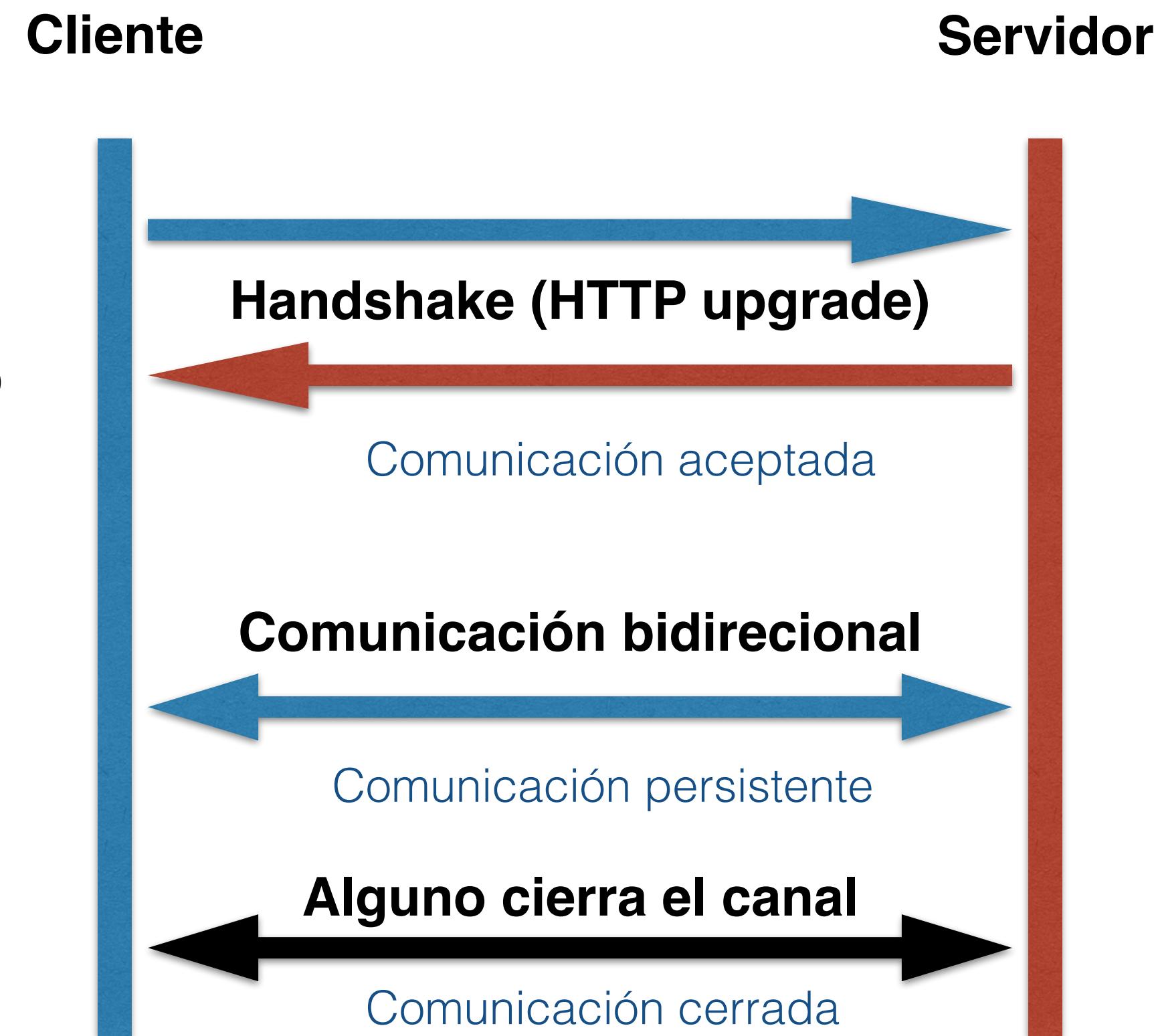
Download Go

Binary distributions available for Linux, macOS, Windows, and more.



WebSockets

- Comunicación **full-duplex** entre clientes y servidores
 - Actualización desde el servidor en tiempo real
 - Juegos en línea sin plugins
 - Comunicación persistente



JSON

- **según [w3schools.com](#):**
- JSON stands for JavaScript Object Notation
- JSON is a lightweight data-interchange format
- JSON is language independent *
- JSON is "self-describing" and easy to understand

```
{  
  "638307884": {  
    "prediction": false,  
    "probability": {  
      "false": 0.9919783122093391,  
      "true": 0.008021687790661005  
    }  
  }  
}
```

“638307884”	“prediction”	false
“probability”	“false”	0.9919783122093391
	“true”	0.008021687790661005

Microservicios: Heroku-Docker

- Múltiples Clientes
 - Web pesados
 - Móviles
 - API para otras aplicaciones B2B
 - Otros servicios
- **Servidores livianos eficientes** y escalares
 - Docker
 - Heroku
 - Spark
- Comunicación via **RESTful** Web services



Amazon lambda

Bases de Datos NoSQL

- Diseñadas para:
 - Escalar fácilmente
 - Desarrollo rápido
 - Big data
 - simple API y simple data
 - OO maping más facil
- Tres tipos
 - Document
 - Graph
 - Key value stores
 - Wide column stores



Amazon
Dynamo

Infraestructura de aplicaciones

- Concentrarse en lógica de negocio
- Dejar atributos de calidad y funciones genéricas a la infraestructura
 - Seguridad
 - Escalabilidad
 - Otros atributos
- Objetos simples
- API simple

Desarrollo
simple

Infraestructura
maneja atributos de
calidad

Agile, CI/CD, DevOps y otros

- Los procesos son **Ágiles (Agile)**
- **CI/CD:** El **ciclo de vida de soluciones soportadas en software** están automatizadas. Foco en herramientas: maven, oradle, Jenkins, circleCI, git, github, etc.
- **DevOps** se enfoca en la cultura organizacional
- **Lean**, se enfoca en proceso, en disminuir el desperdicio a 0 y mejorar la calidad. Solo son principios (Tarea averiguar)

GOF: Patrones de Arquitectura de Software

- **Creacionales**
 - Abstract Factory
 - Prototype
 - Builder
 - Factory Method
 - Singleton
- **Estructurales**
 - Adapter
 - Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy
- **Comportamiento**
 - Chain of responsibility
 - Command
 - Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template method
- Visitor

Nuestro catálogo de patrones hasta ahora

- **Arquitectura (Empresarial)**
 - Heurísticas
 - MDA, Modelos y Metamodelos
 - Vistas, Puntos de Vista
 - Sistema
 - Arq. = Planear, Diseñar, Construir, Mantener, Evolucionar Sistema
- **Capas de representación de AE**
 - Procesos (BPMN)
 - Datos (Clases para representar conceptos)
 - Aplicaciones (Componentes)
 - Tecnología (Despliegue)
- **Estrategias de Diseño de sistemas**
 - Modularizar
 - Abstracción
 - Capas
 - Jerarquías
 - Nombres
 - Iteración
 - KISS
- **Patrones de Arquitectura**
 - Clientes y servicios
 - BD, Archivos, RPC, Mensajería
 - Virtualización
- **Patrones de Arq. Soft**
 - Pool de hilos
 - IoC
 - MVC

Fin