Conceptos Básicos de Diseño de Sistemas Distribuidos

Luis Daniel Benavides 22-08-2024

Conceptos Básicos de Diseño de Sistemas

Importante Leer

- Capítulo 2 de “Principles of Computer system Design" by Saltzer and Kaashoek

Estrategias básicas de diseño de sistemas

Atacando complejidad

- Modularización: divida y conquiste
- Abstracción: separación de interfaces e implementación
- División por capas
- Jerarquía: Interconexión en forma de árbol
- Uniendo todo: nombres para hacer conexiones
- * Iteración
- * Manténgalo simple

Las tres abstracciones fundamentales

Las tres abstracciones fundamentales

- Memoria
- Interpretes
- Enlaces de comunicación

Memoria

- Algunas veces llamada almacenamiento
  - *write(name, value)*
  - *value <——— read (name)*
- Ejemplos
  - Dispositivos de memoria de hardware
    - RAM chip, Flash memory,Disco magnético, CD
  - Sistemas de memoria de alto nivel
    - RAID, Sistema de archivos, Sistema de gestión de bases de datos

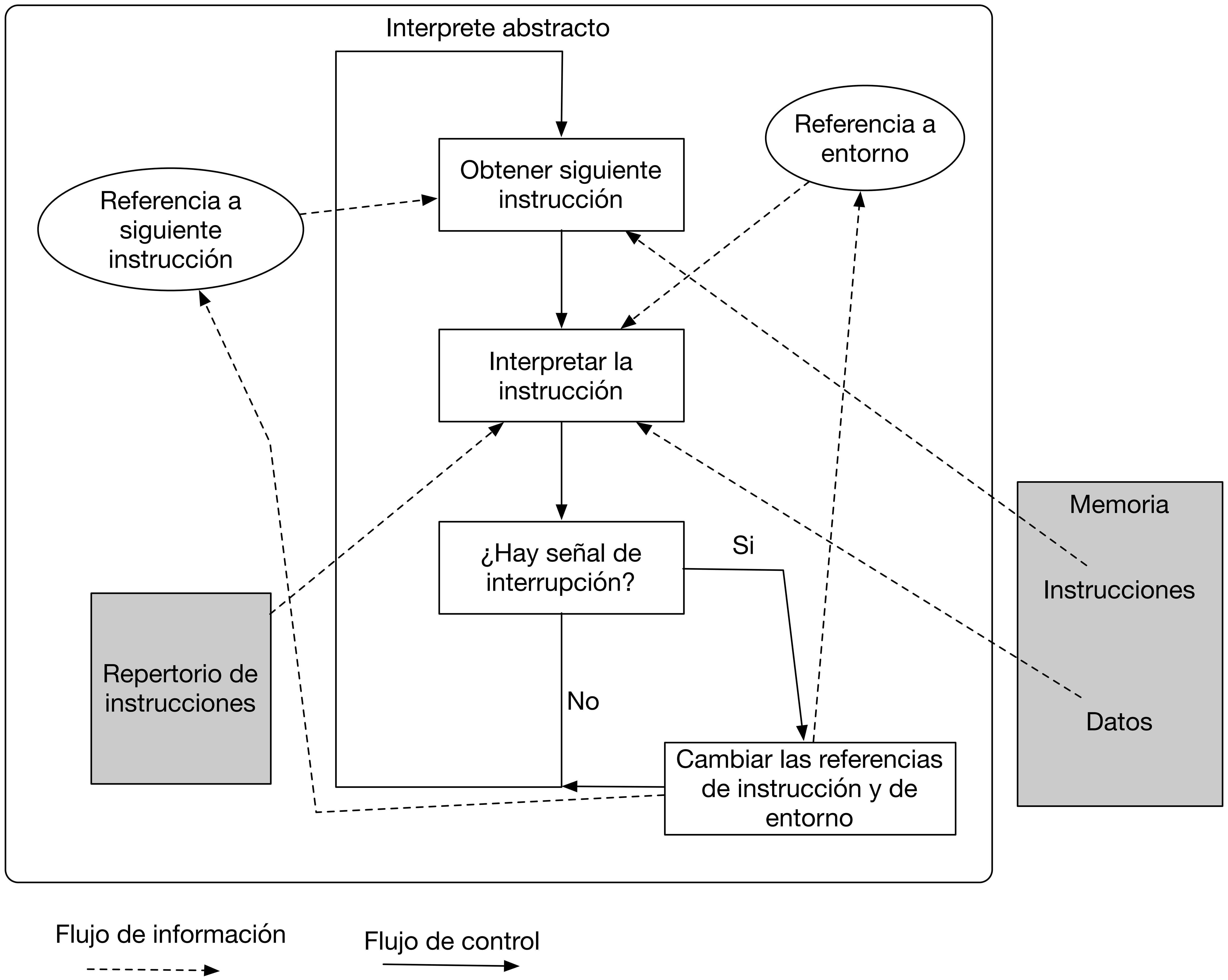
Coherencia y atomicidad del Read/Write

- Coherencia: READ es simple igual al mas reciente WRITE
- Atomicidad Antes-o-Después: Todo READ o WRITE ocurre completamente antes o después de cualquier otro READ or WRITE
- Amenazas a la coherencia y atomicidad:
  - Concurrencia (Palabra synchronized en JAVA)
  - Almacenamiento remoto (demoras pueden alterar el orden de los WRITES o READS)
  - Mejoramiento de Desempeño (Memorias Caché, optimización de los compiladores)
  - Tamaño de celdas no acomodada tamaño del valor
  - Almacenamiento replicado

Interpretes

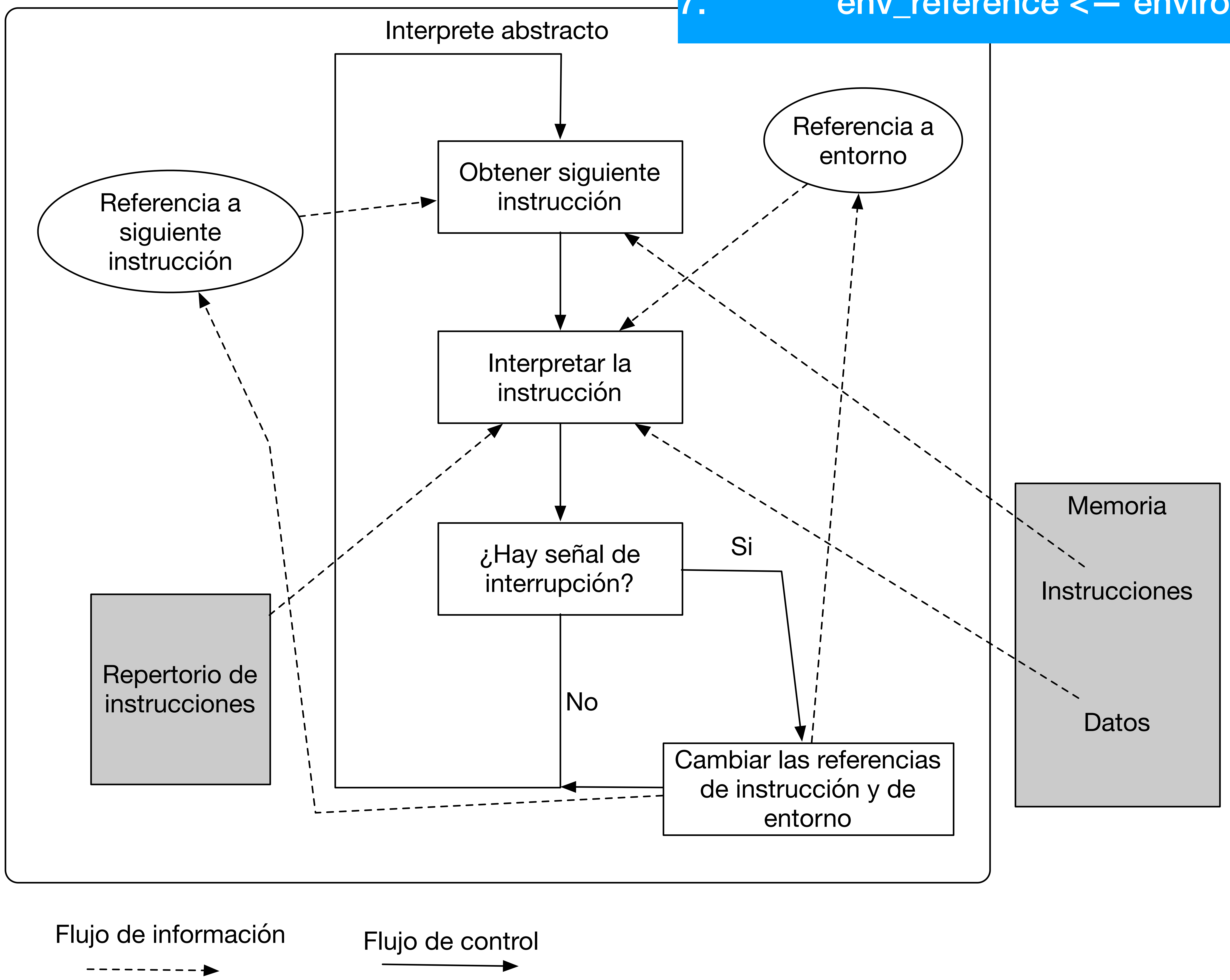
- El elemento activo de los sistemas de información.
- Se pueden modelar con
  - Una referencia a la siguiente instrucción
  - Un repertorio de acciones
  - Una referencia a un entorno
- Ejemplos
  - Hardware: Pentium 4, PowerPC 970, controlador de disco, controlador de pantalla
  - Software: Java, LISP, Pascal, C++, Smalltalk, Excel, Internet Explorer, Firefox

Interprete abstracto



Interprete a

```
procedure INTERPRET()
do forever
Instruction <— READ (instruction_reference)
Perform instruction in the context of env_reference
if interrupt_signal = TRUE then
instruction_reference <— entry point of int_handler
env_reference <— environment reference of int_handler
```



Capas de Interpretes

- Generalmente los interpretes se organizan en capas
- La capa inferior es el hardware
- Cada capa le entrega un repertorio de instrucciones a la capa superior (API)
- **Ejercicio:** Puede representar en un dibujo el concepto de capas de interpretes tomando como ejemplo un programa calendario escrito en java?

Enlaces de comunicación

- Proveen un mecanismos para mover información entre componentes físicamente separados
- Un modelo simple
    - *send (nombre_Del_Enlace, espacio_memoria_salida)*
    - *receive (nombre_Del_Enlace, espacio_memoria_entrada)*
- Ejemplos
    - Hardware: par trenzado, cable coaxial, fibra óptica
    - Alto nivel: ethernet, Universal Serial Bus, la internet, el sistema telefónico, el *pipe* de Unix.

Nombres en sistemas de computo

Nombres

- Los sistemas manipulan objetos
- Los nombres referencian objetos
- Ejemplos
    - R5 (registro de procesador)
    - 174FFF (Dirección de memoria)
    - escuelaing.edu.co (nombre de punto de conexión de red)
    - 18.72.0.151 (dirección de punto de conexión de red)
    - alice (nombre de usuario)
    - /proyecto/planeación/plan.doc
    - http://www.escuelaing.edu.co/sistemas/respuestasaProblemas.txt

Nombres desde la perspectiva de objetos

- Computador manipula objetos
- Objeto puede ser estructurado, es decir está conformado por otros objetos
- Un objeto puede usar otros objetos por copia o por referencia
- ¿Recuerda cuál es la diferencia?
- Nombres me sirven …
  - Como herramienta de comunicación y organización
  - Para desacoplar un objeto de los otros. El enlace se puede hacer más tarde en el tiempo
- Escriba algunos ejemplos del suyo de nombres para comunicarse y para desacoplar.

Esquema de nombres

- El diseñador crea un esquema de nombres
  - Espacio de nombres, alfabeto y sintaxis de los nombres
  - Algoritmo de mapéo de nombres, asocia nombres con valores
  - Universo de valores, un valor puede ser un objeto, o otro nombre del mismo o otro espacio de nombres
- La resolución de nombres depende de un contexto
  - Ej.: Búsqueda de nombre en directorio de una ciudad (contexto=ciudad)
  - Ej.: Búsqueda relativa de un archivo en un subdirectorio

API simple para el manejo de nombres

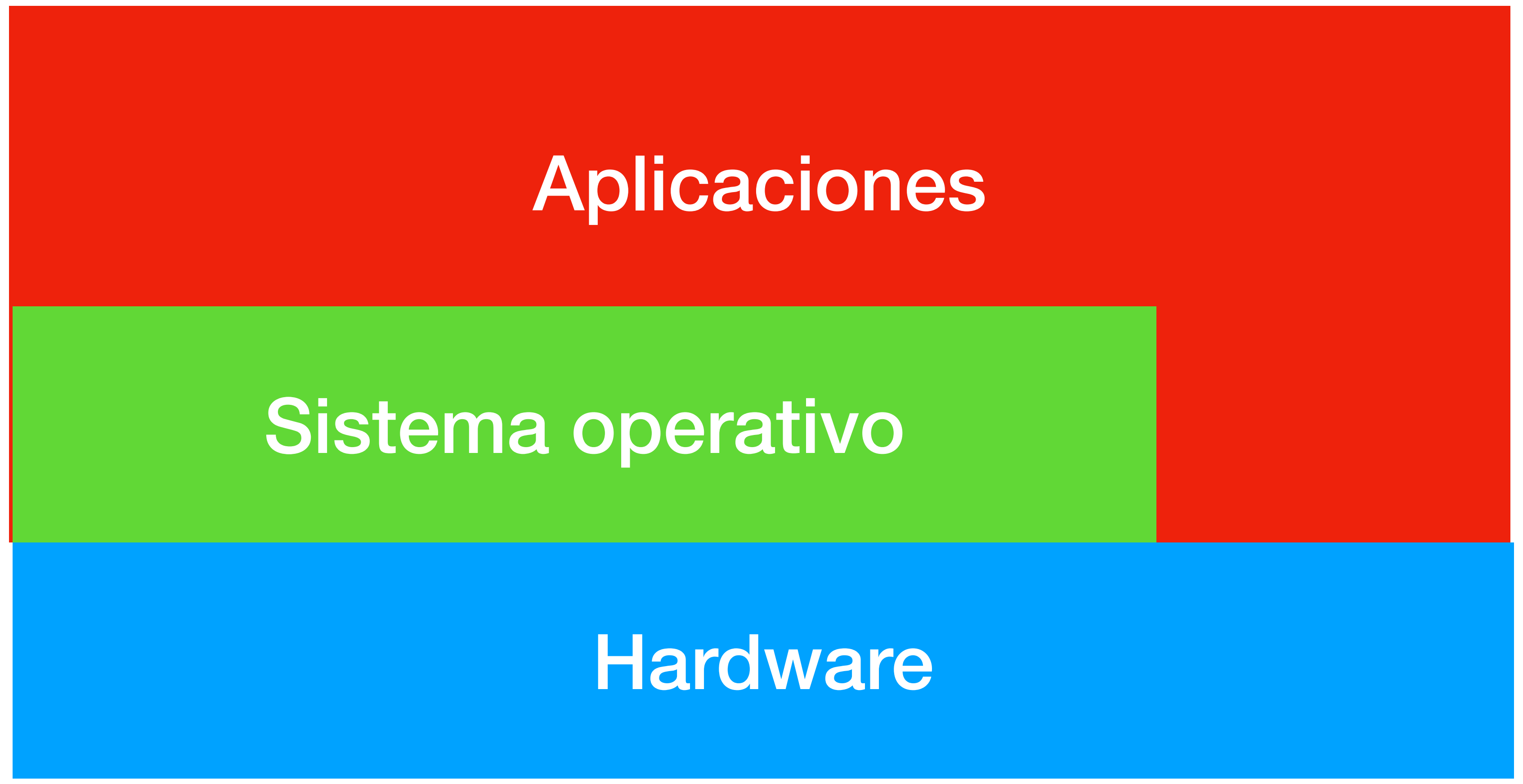
- Value <— RESOLVE(name, context)
- status <— BIND (name, value, context)
- status <— UNBIND (name, context)
- list <— ENUMERATE (context)
- result <— compare(name1, name2)

¿Cómo resolver nombres?

- Nombre bien conocido, ej., google.com
- Broadcast: publicar el nombre en un espacio de alta difusión
- Búsqueda, por ejemplo en un buscador
- Broadcast búsqueda, ej., Gritar “Alguien conoce un nombre para...(algo)”
- Resolver el nombre de un espacio de nombres hacia otro espacio de nombres, ej., DNS (www.gg.com -> 200.12.12.1)
- Presentación, alguien me presenta
- Encuentro físico, ej., una cita

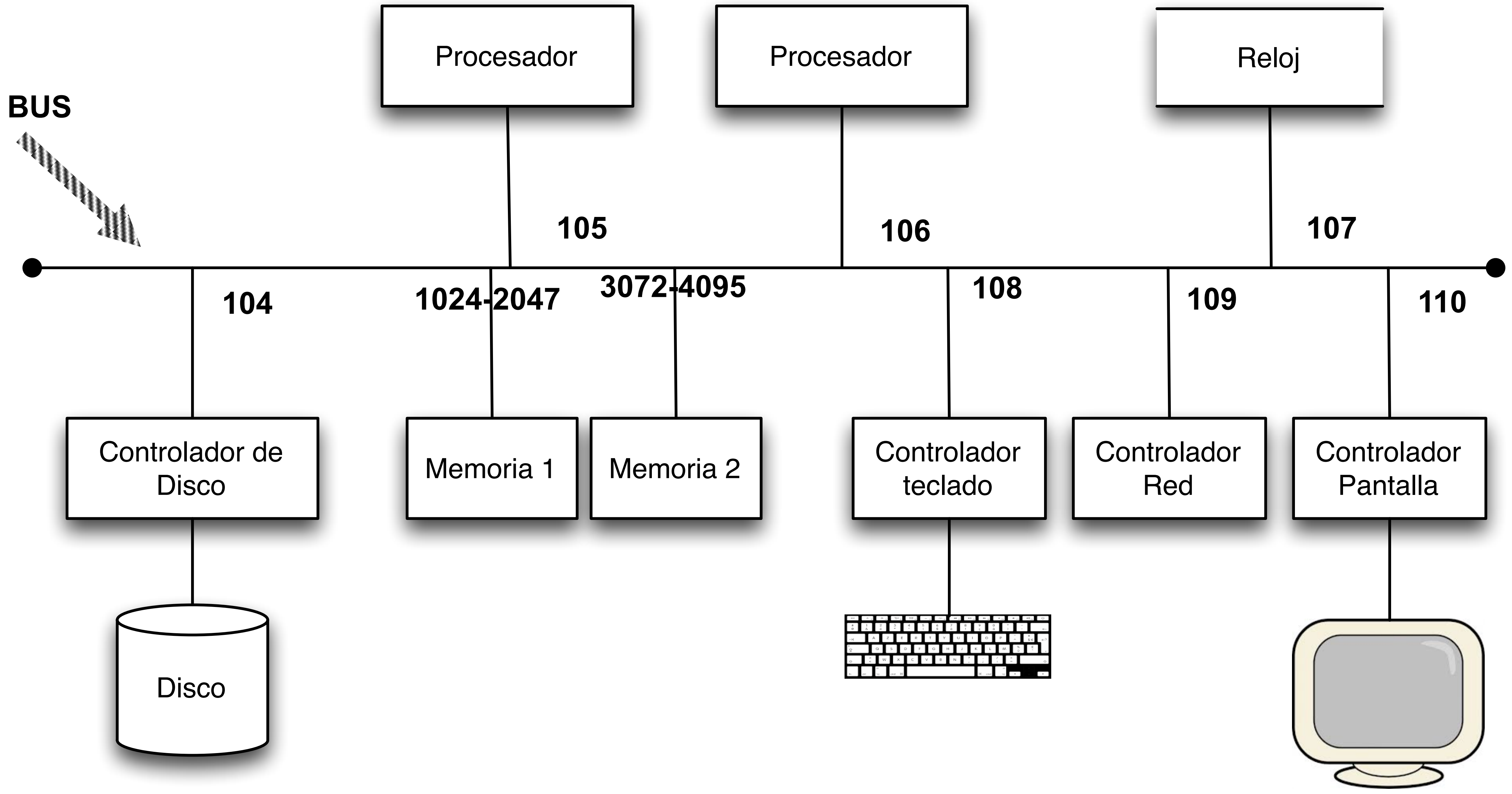
Organizando sistemas con nombres y capas

La organización típica de un computador



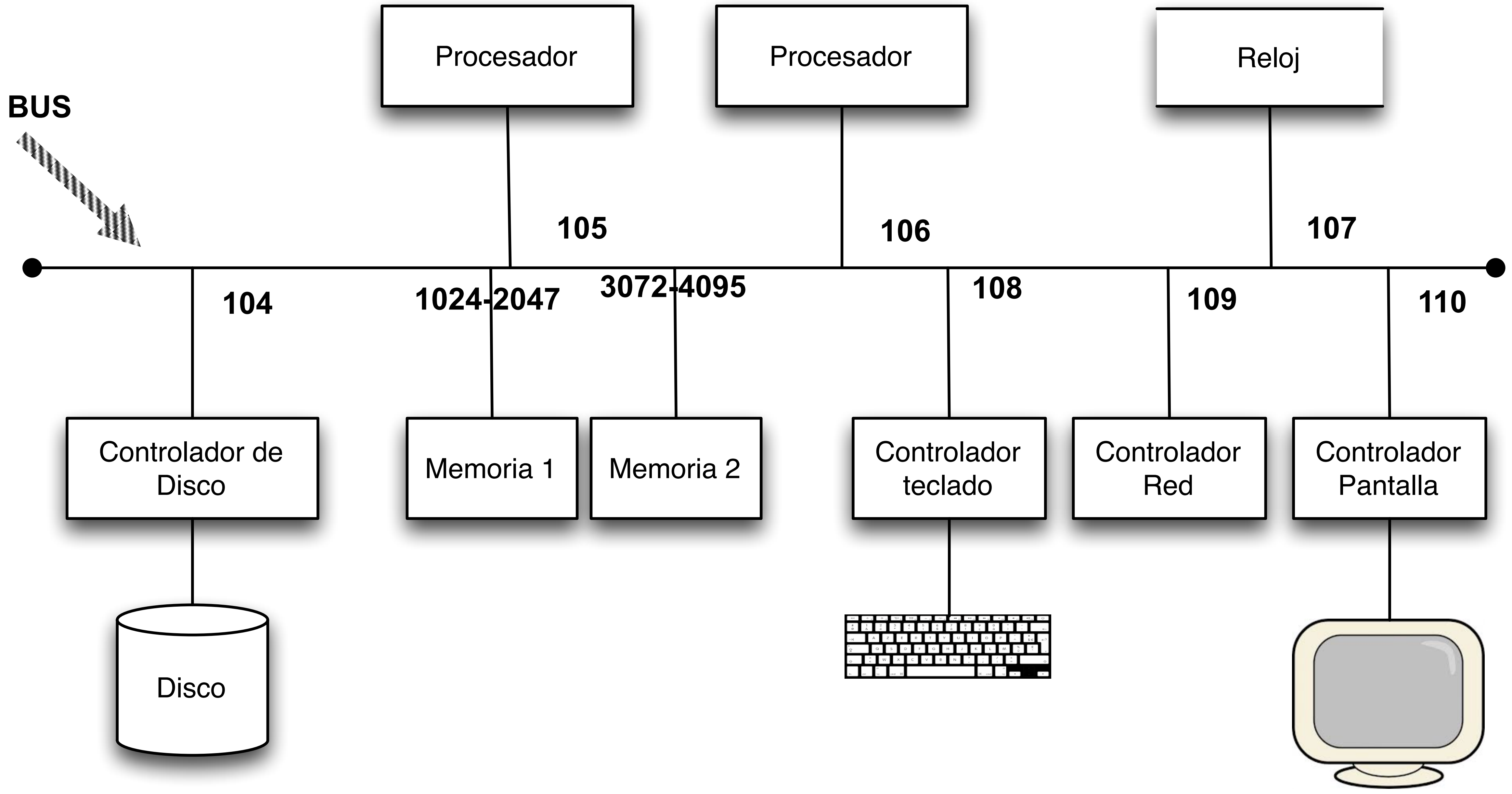
Una capa de hardware: el Bus

- La capa de hardware de un computador típico está construida por módulos que implementan las tres abstracciones



Una capa de hardware: el Bus

- La capa de hardware de un computador típico está construida por módulos que implementan las tres abstracciones

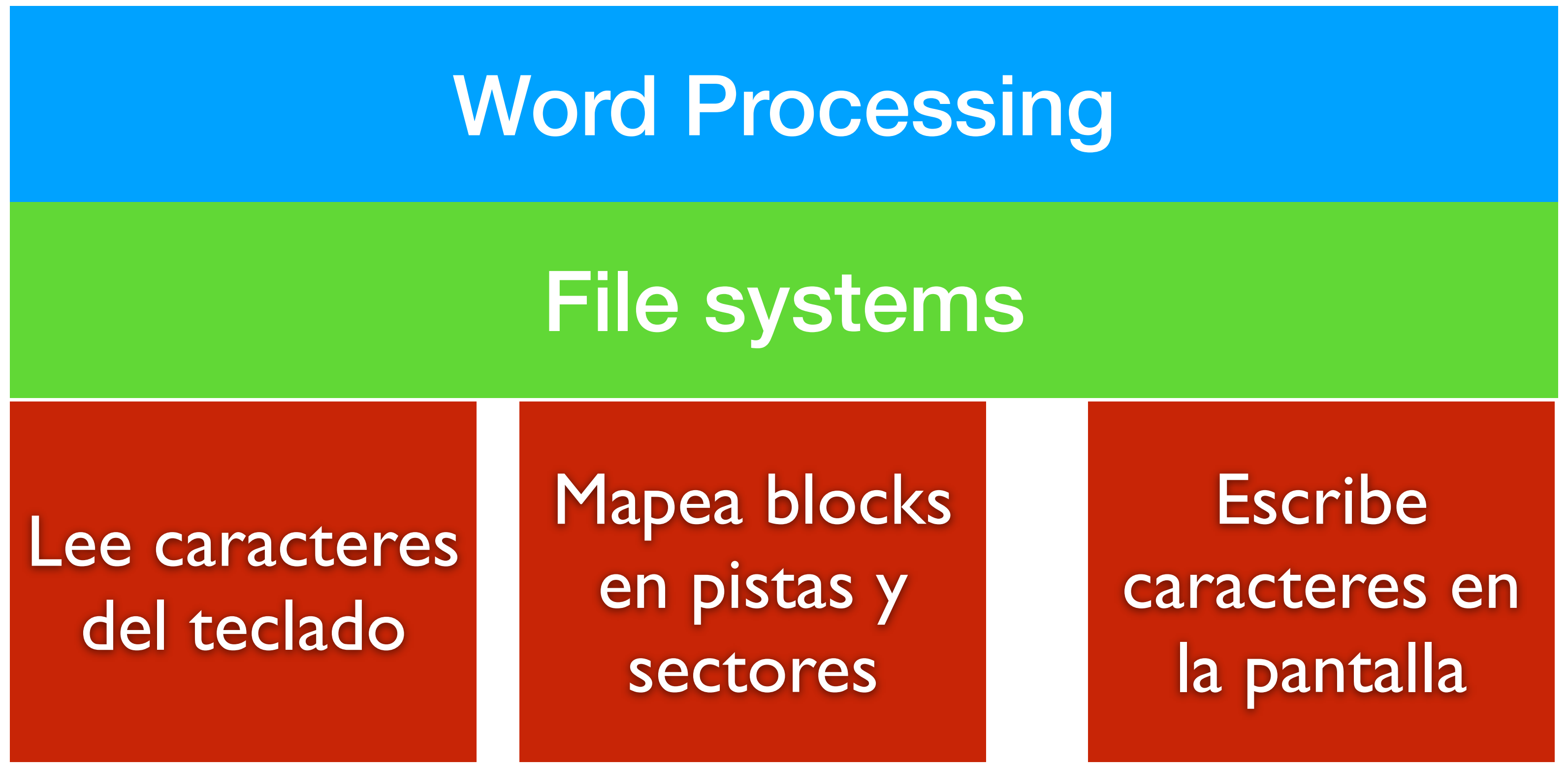


Ejemplo de comunicación en el bus
El procesador 2 quiere leer el contenido del espacio de memoria 1742 en el registro 1:

Mensaje 1: **{1742, READ, 106}**
En la memoria: **value <— READ(1742)**
Respuesta 1: **{106, value}**

Una capa de software: El sistema de archivos

- En Unix todo es un archivo, inclusive dispositivos externos proveen una interfaz de archivo
- El sistema se organiza por capas
- Se utilizan nombres de directorios y de archivos para referenciar archivos y dispositivos



Ejemplo de uso del sistema de archivos en Unix

- Programa simple para leer del teclado, almacenar en un archivo e imprimir en la pantalla.

```
character buf

file <— OPEN(“/Users/Alumno/Documento de ARQUITECTURA.doc”,READWRITE)

input <— OPEN (“keyboard”, READONLY)

display <— OPEN (“display”, WRITEONLY)

While not End_Of_File(input) do

  READ (input, buf, 1)

  WRITE (file, buf, 1)

  WRITE (display, buf,1)

CLOSE (file)

CLOSE (input)

CLOSE (display)
```

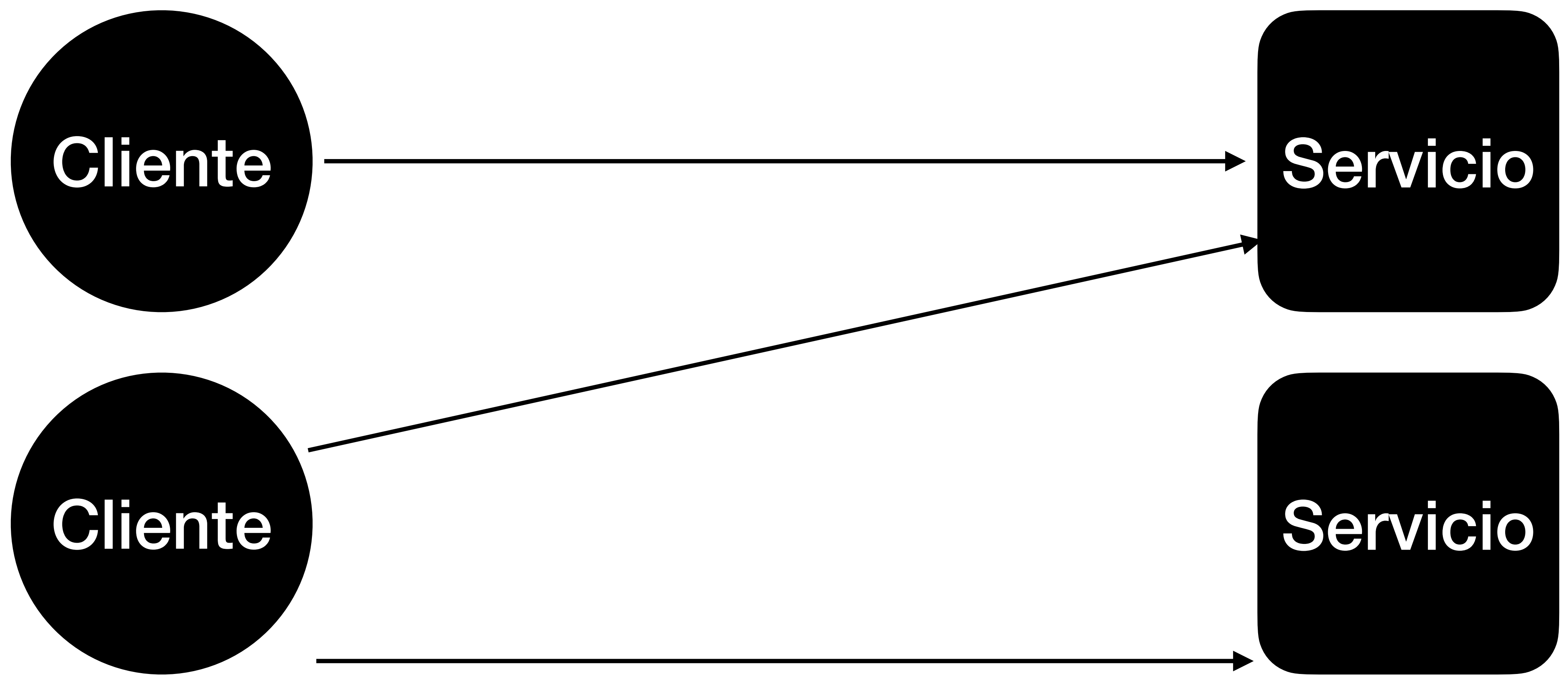
Sistemas distribuidos e integración

Modularidad por medio de clientes y servicios

Problema

- Aprendimos que dividir un sistema en módulos es una buena estrategia de diseño
- Si los programadores/Diseñadores no cometieran errores esto es todo lo que necesitaríamos.
- La modularización no impide que se propaguen errores en el sistema. Hay interacciones implícitas inesperadas.
- Ante el error necesitamos formas más fuertes de modularización que protejan el sistema de errores.
- Organizar el sistema con clientes y servicios protege el sistema de interacciones inesperadas.

Clientes y Servicios



Al menos tres beneficios:

- Los mensajes son la única forma de interacción (limita interacción y protege contra violación de modularidad)
- Los mensajes son la única forma en que los errores se propagan
- Los mensajes son la única forma en que un atacante penetra un servicio

Esta simple estratégia permite crear sistemas modulares, tolerantes a fallas y seguros.

Es LA estrategia primordial para diseñar sistemas complejos.

Algunas consideraciones de diseño

- La división en funciones es una forma débil de modularidad. Ya que las funciones comparten memoria y pueden crear interacciones inesperadas y propagar errores.

- Por ahora consideremos que cada cliente y cada servicio corre en su propio computador y se comunican por un cable.

- Ejemplo: Aplicación Web con al menos dos servicios. Al menos dos opciones: Un servidor con dos métodos, o dos servicios independientes.

Mecanismos de integración

Puntos de decisión para la integración de aplicaciones

- Acoplamiento de la aplicación
- Intrusividad: minimizar cambios en aplicación y en mecanismo de integración
- Selección de tecnología
- Formato de los datos

Puntos de decisión para la integración de aplicaciones (cont.)

- Tiempo de vida de los datos
- Datos o funcionalidad
- Comunicación remota: Sincronía, parámetros, fallos etc.
- Confiabilidad

Estrategias de integración

- Transferencia de Archivos
- Base de datos compartida
- Invocación remota de métodos
- Mensajería

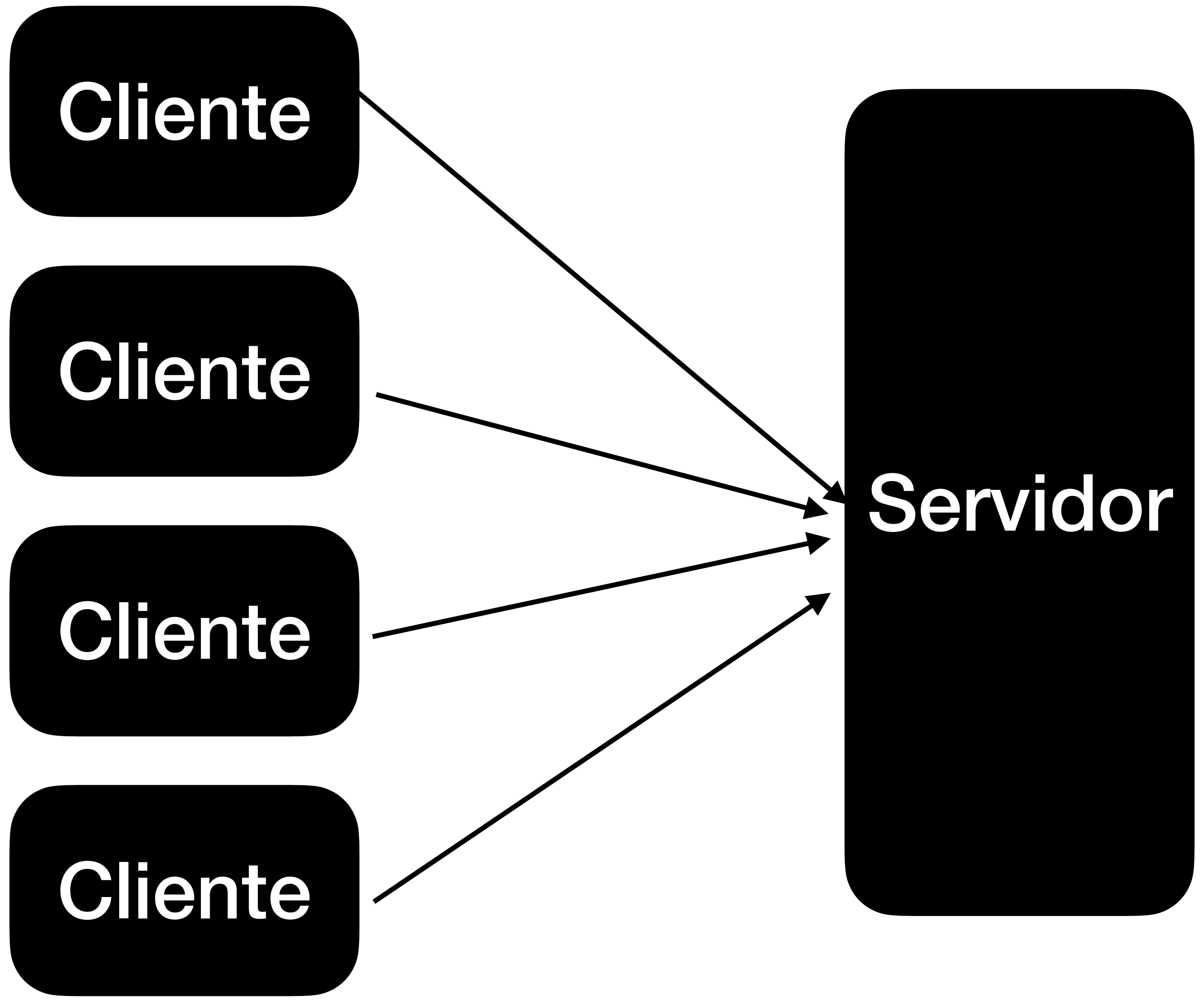
Patrones de integración

Patrón 1: Remote procedure call (RPC)

- Solicito un servicio y espero por la respuesta
- Simula un llamado local de procedimiento
- Pero no lo es
  - La semántica de invocación es diferente (debe prepararse para los errores y considerar el marshaling)
  - Se demora más tiempo la respuesta
- Ejemplos
  - Bases de datos

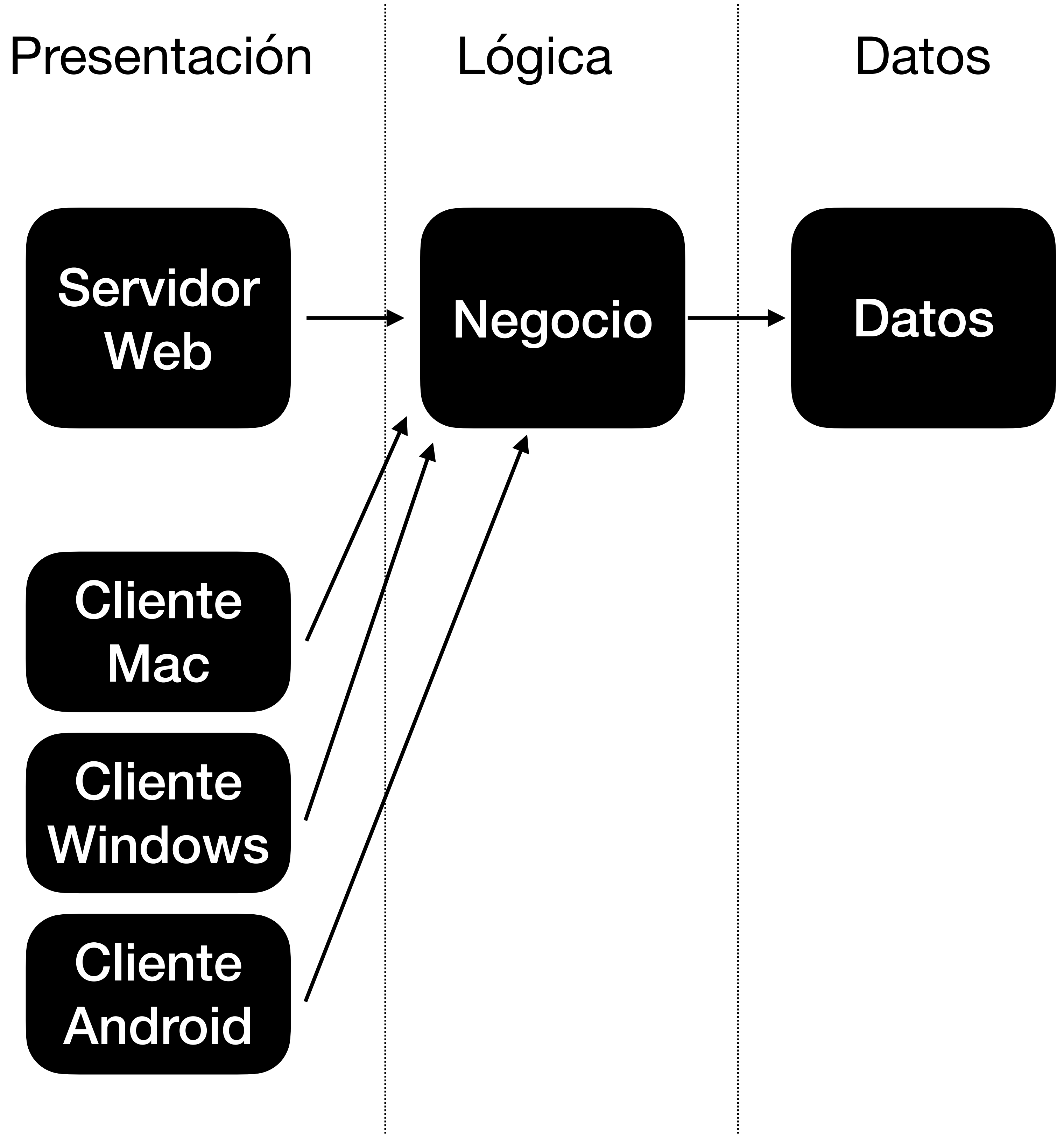
Patrón 2: Cliente-Servidor

- Múltiples clientes generalmente iguales, un servidor
- Arquitectura simple
- Hay que actualizar todos los clientes cuando hay un cambio
- Servidor único punto de falla
- Ejemplo: App conectada a base de datos



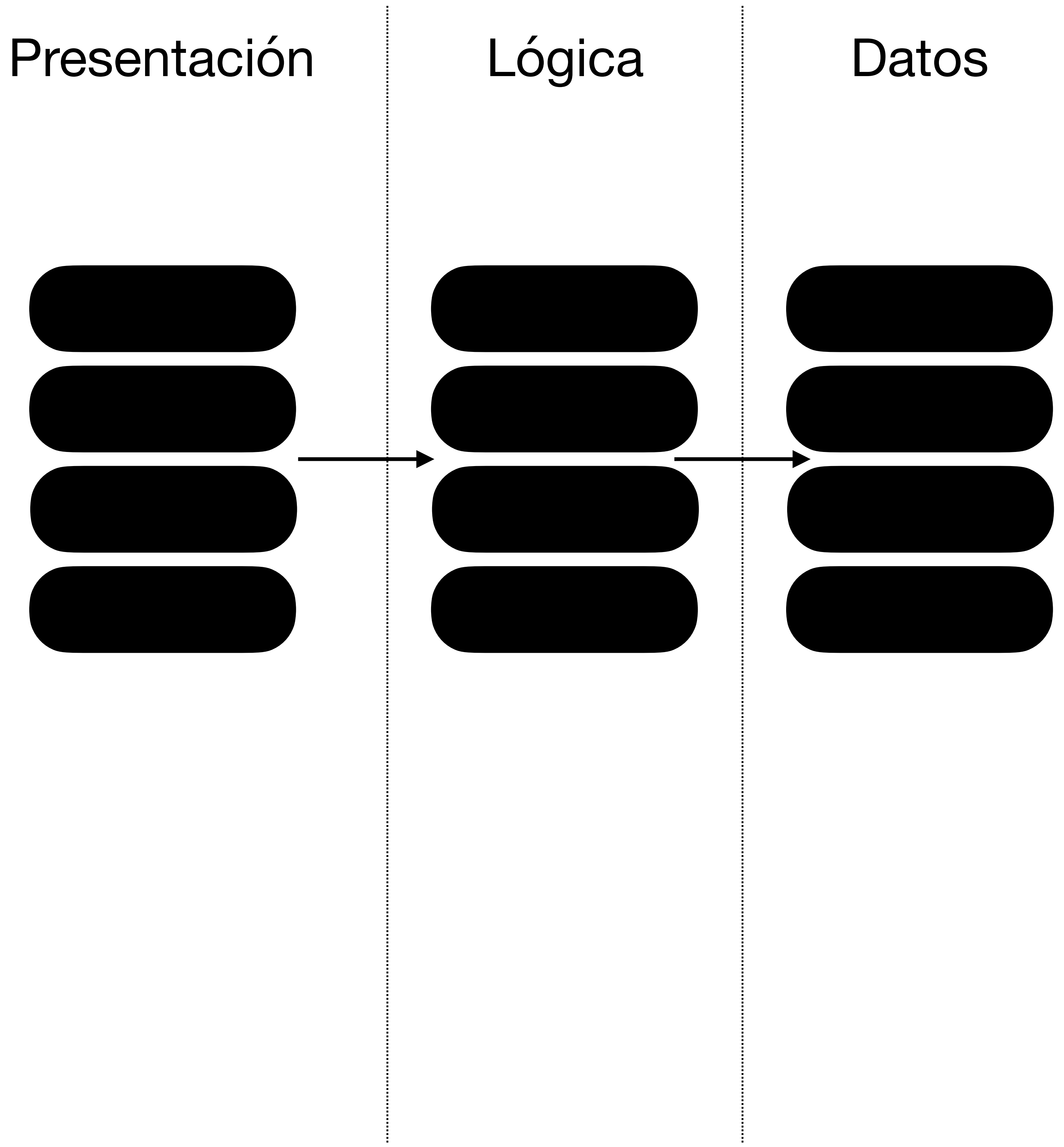
Patrón 3: 3 Capas

- Se separa en tres capas típicas: Presentación, lógica, persistencia
- Arquitectura simple
- Facilidad para soportar múltiples clientes
- Cuándo se actualiza la lógica no es necesario actualizar clientes.



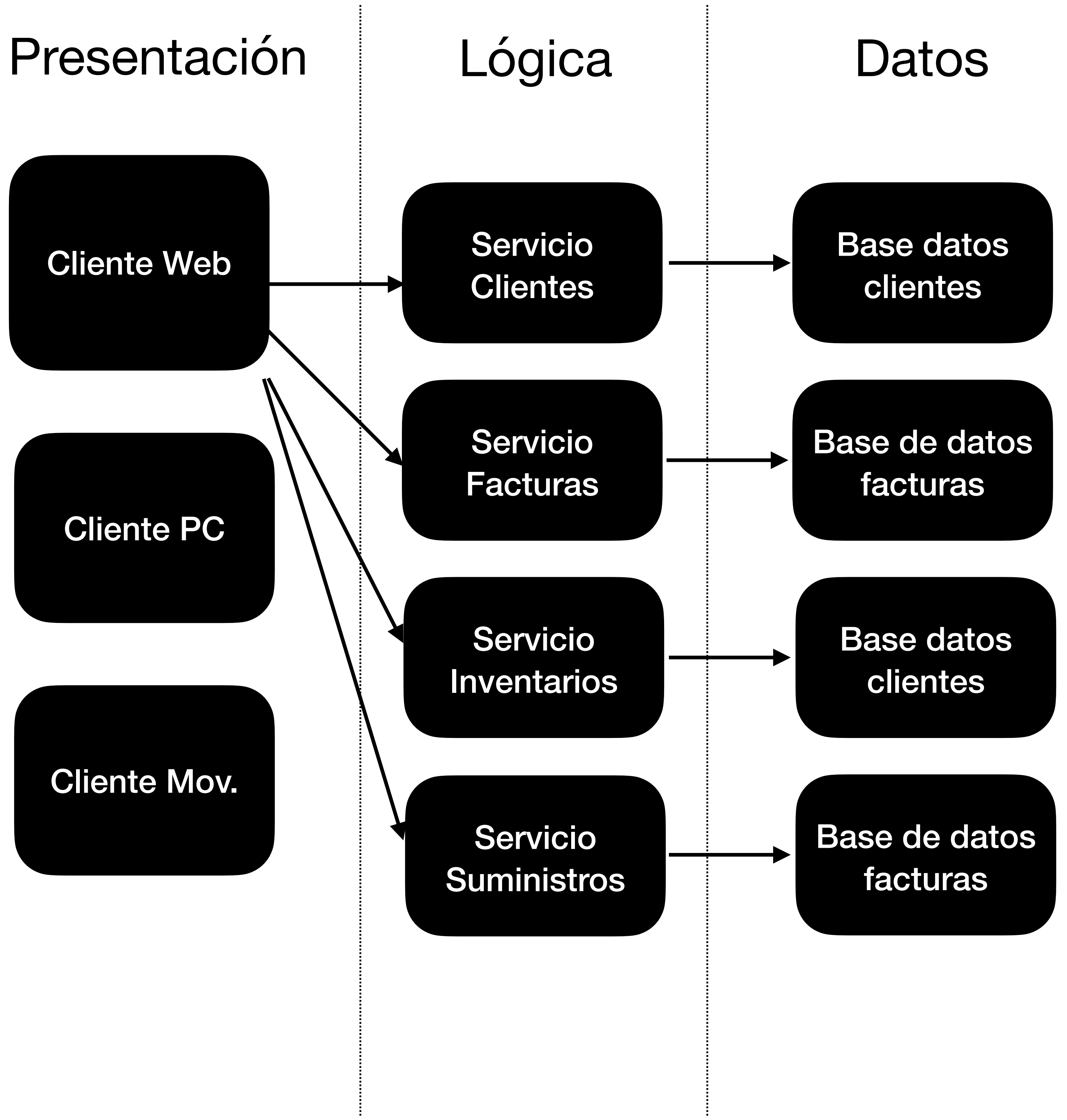
Patrón 4: Clustering

- Separación en capas
- Alta disponibilidad
- Mejor desempeño
- Permite estrategias elásticas

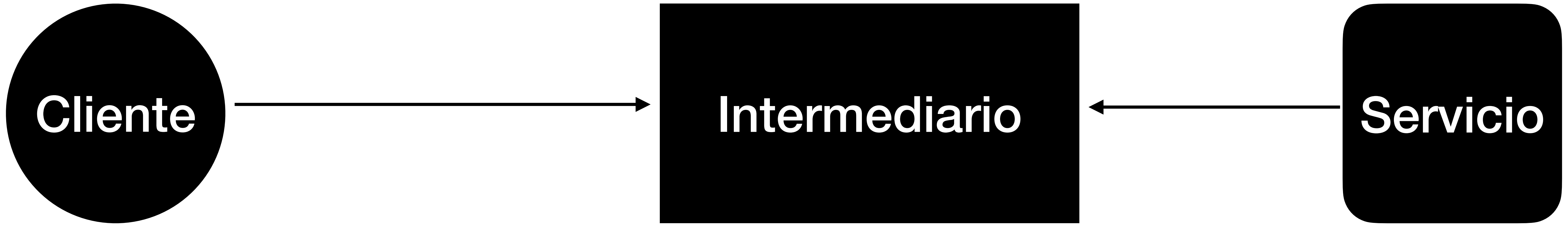


Patrón 5: Microservicios

- Cada servicios es autónomo
- No es monolítico
- Facilidad de desarrollo y despliegue
- Alta complejidad
- Coordinación compleja
- Permite ultra-escalabilidad
- OJO con la granularidad del servicio!!
- Ejemplo: Netflix, Amazon, Facebook



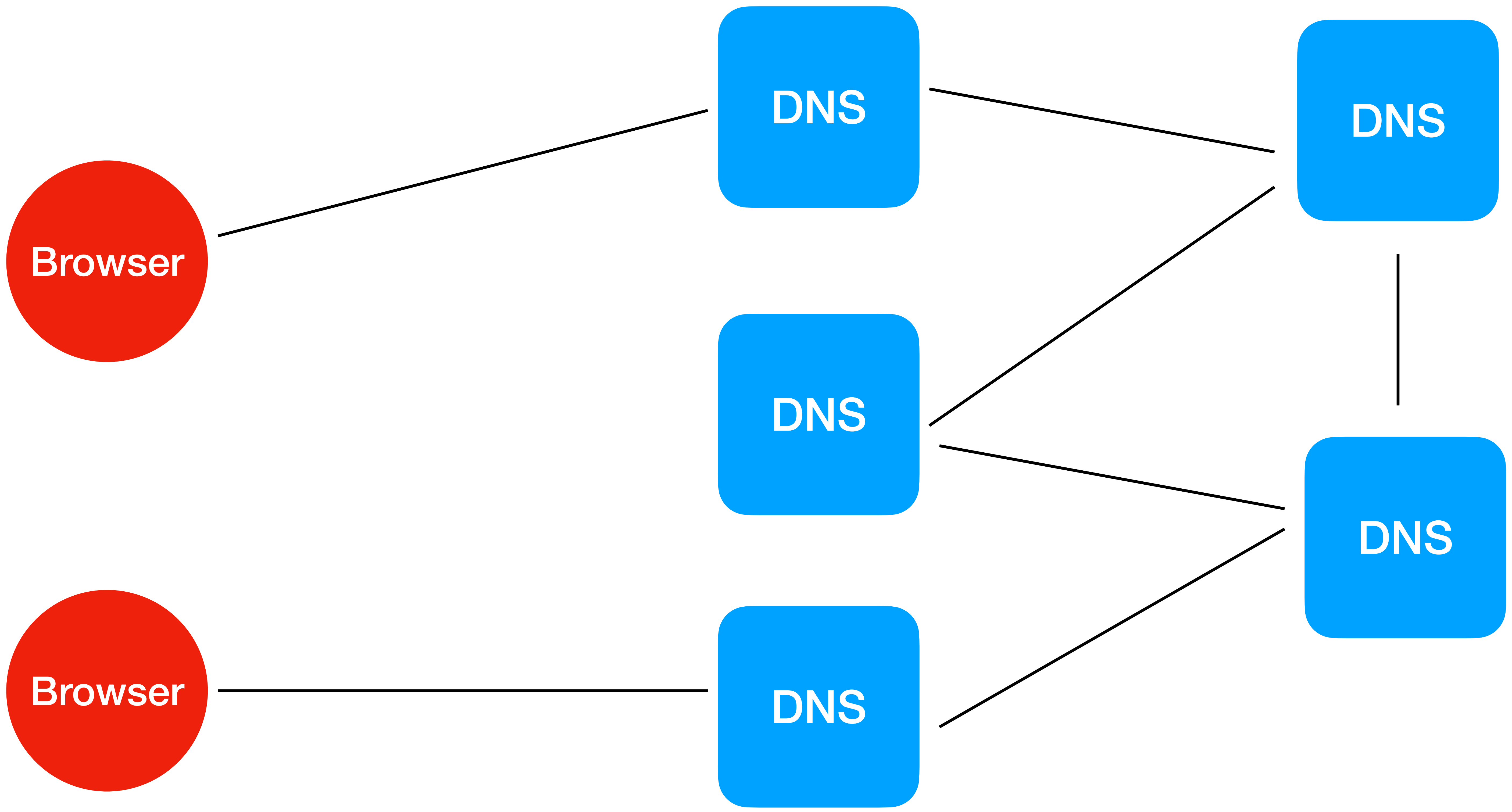
Patrón 6: Intermediario



- Comunicación por medio de un intermediario
- Permite patrones de comunicación asíncronos
    - Push vs. Pull
    - Desacoplar con indirección (Por ejemplo, enviar a decano@escuelaing.edu.co)
    - La indirección permite decidir cuando duplicar mensajes (Ej, enviar a lista de correos)
    - **Publish subscribe**
- Ejemplos:
    - Email, SOA, MENSAJERIA

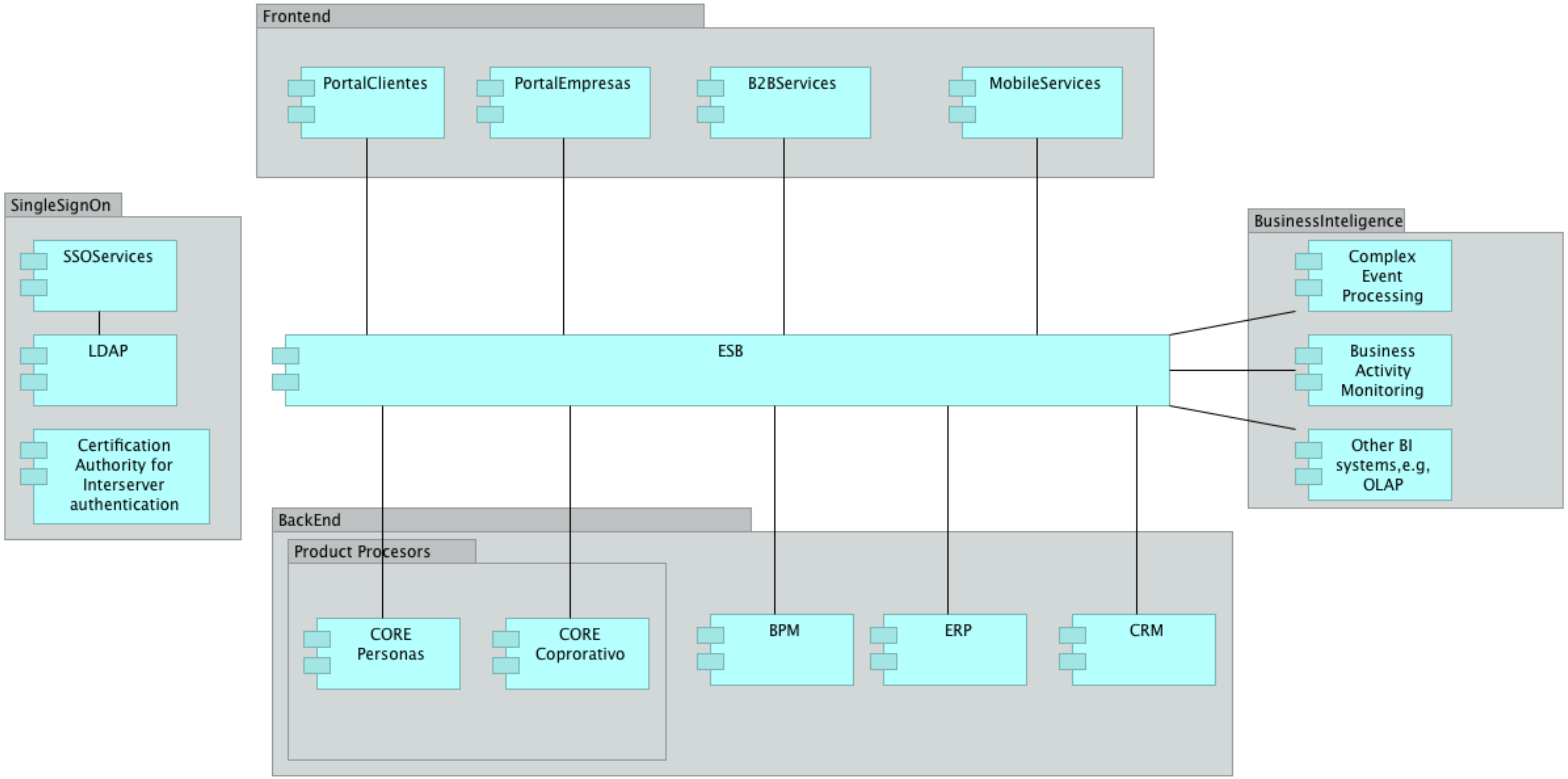
Ejemplos

Ejemplo: The internet Domain Name System



SOA

Patrón que extiende el patrón intermediario



Bibliografía

- Martin, Robert C. . Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin Series) (English Edition).Addison-Wesley. New York..2018

- Saltzer, Kashoek. Principles of Computer System Design. Morgan Kaufmann. 2009.

- Hohpe et al. Enterprise Integration Patterns. Adison Wesley. 2004.

- O'Brien,Bass,Merson,. Quality Attributes and service Oriented Architectures. Tecnical Report, Carneige & Mellon SEI. 2005.

Fin