

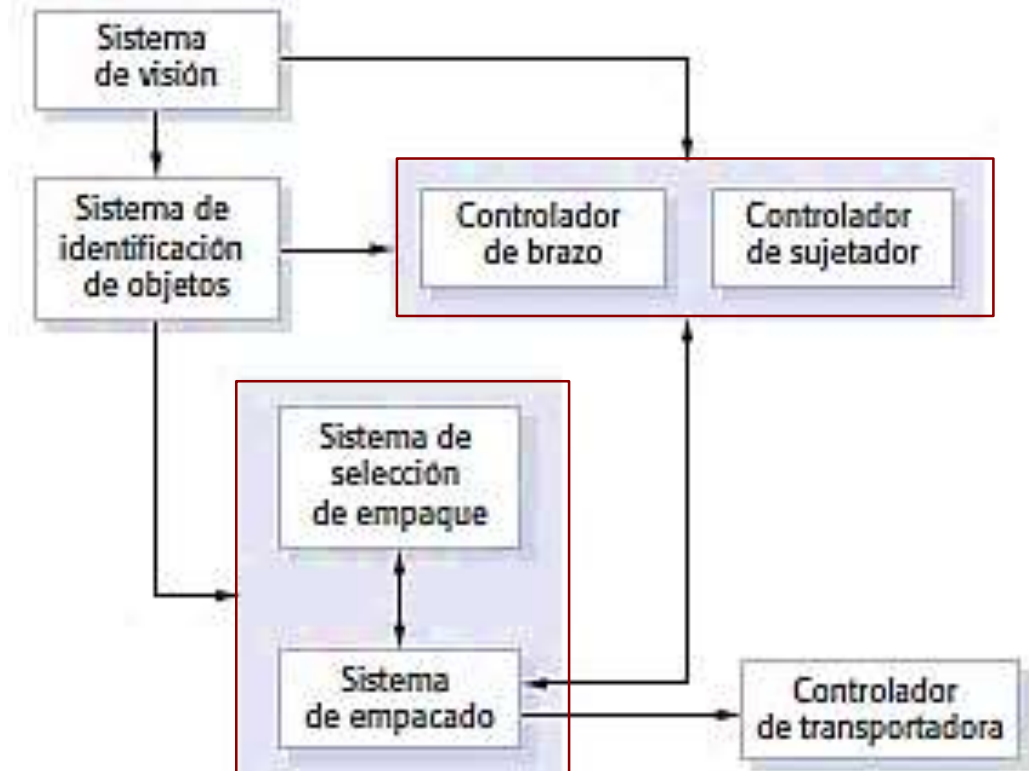


# Ingeniería de software II

Diseño de Software – Diseño arquitectónico

# Diseño Arquitectónico

- ❖ Define la relación entre los elementos estructurales, para lograr los requisitos del sistema.
- ❖ Es el proceso de identificar los subsistemas dentro del sistema y establecer el marco de control y comunicación entre ellos.
- ❖ Los grandes sistemas se dividen en subsistemas que proporcionan algún conjunto de servicios relacionados

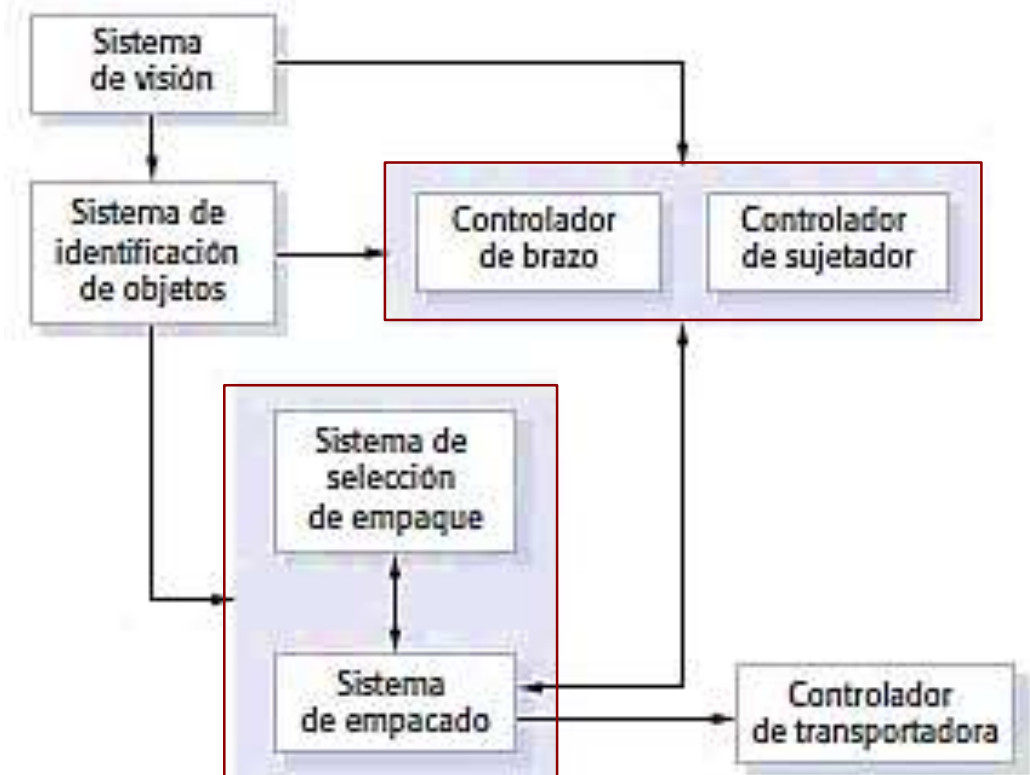


2

# Diseño Arquitectónico

En la figura se presenta un modelo abstracto de la arquitectura para un sistema de robot de empaquetado, que indica los componentes que tienen que desarrollarse.

El modelo arquitectónico presenta dichos componentes y los vínculos entre ellos.



3

# Diseño Arquitectónico

---

» La arquitectura afecta directamente a los requerimientos no funcionales más CRÍTICOS:

- ❖ *Rendimiento*
- ❖ *Protección*
- ❖ *Seguridad*
- ❖ *Disponibilidad*
- ❖ *Mantenibilidad*

## Requerimientos No Funcionales que impactan en la Arquitectura de Software

# Diseño Arquitectónico



5

# Diseño Arquitectónico

---

1. Organización del sistema
2. Descomposición modular
3. Modelos de control
4. Arquitectura de los Sistemas Distribuidos

6



# Organización del Sistema

» La organización del sistema representa la estrategia básica usada para estructurar el sistema

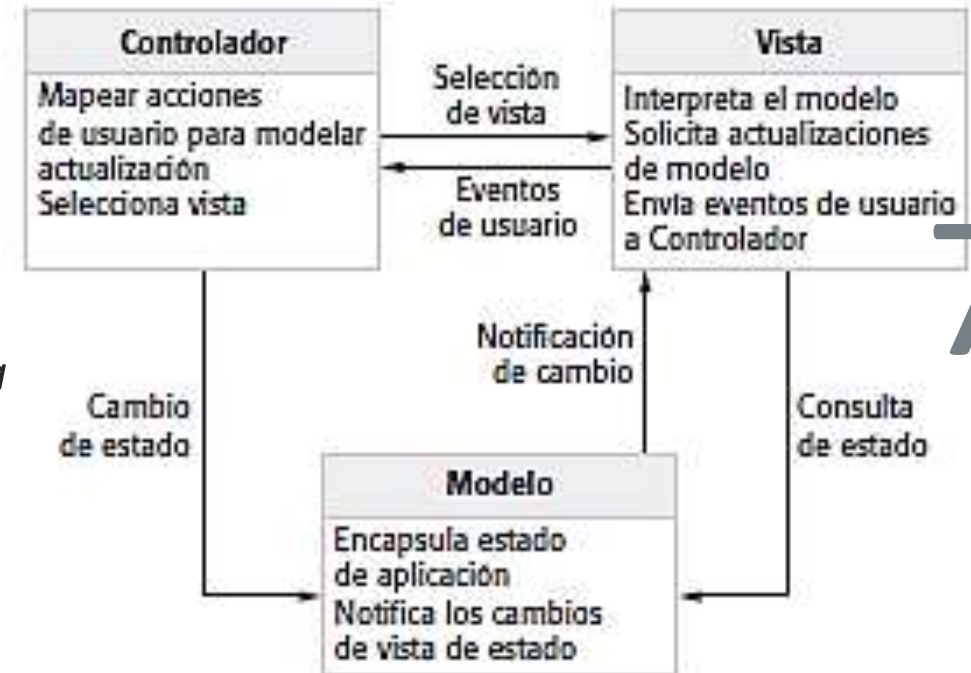
Los subsistemas de un sistema deben intercambiar información de forma efectiva

*Todos los datos compartidos, se almacenan en una base de datos central*

*Cada subsistema mantiene su información y los intercambia entre los subsistemas*

Estilos organizacionales (**Patrones** arquitectónicos)

*Repositorio, cliente-servidor, capas o combinaciones entre ellos, entre otros*



# Organización del Sistema

---

## ***Patrón*** de repositorio

*La mayoría de los sistemas que usan grandes cantidades de datos se organizan alrededor de una base de datos compartida (repositorio)*

*Los datos son generados por un subsistema y utilizados por otros subsistemas*

8

## ***Ejemplo***

Sistemas de gestión, Sistemas CAD, Herramientas Case, etc.



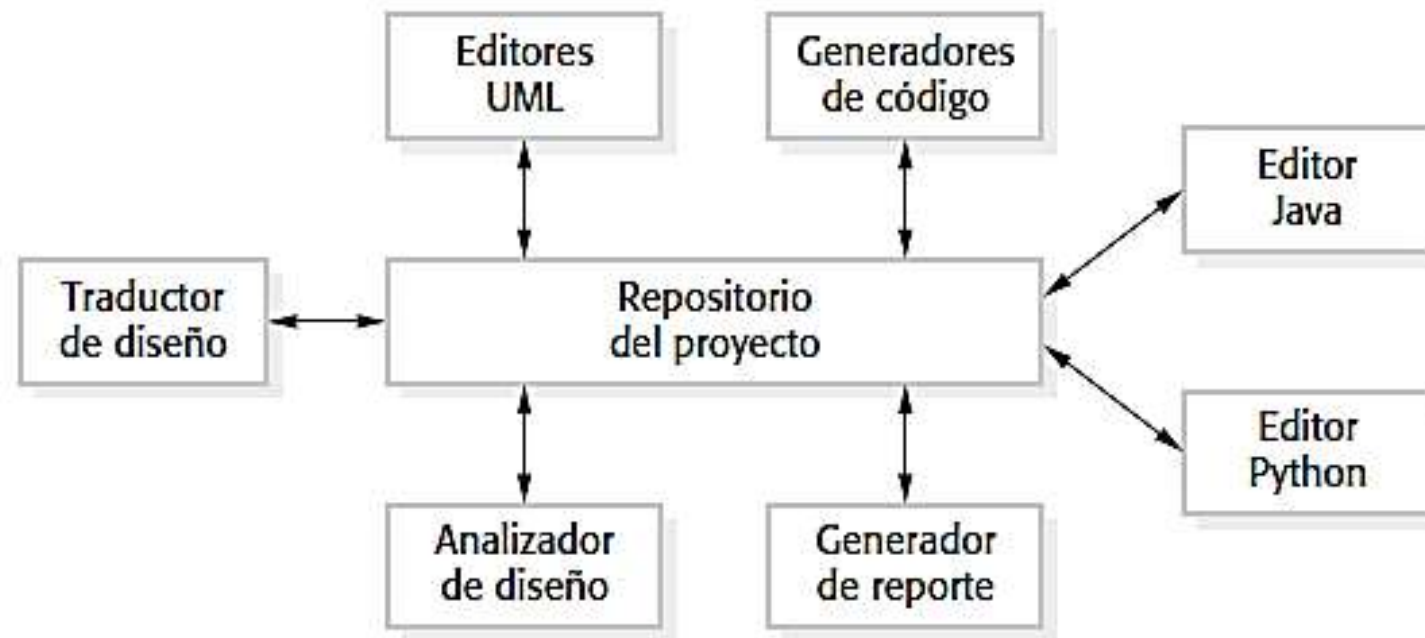
# Ejemplos de patrones de repositorios

## Comparación de Sistemas de Gestión, CAD y CASE

Característica	Sistema de Gestión	Sistema CAD	Herramienta CASE
 <b>Repositorio Central</b>	Base de datos con información crítica	Archivo de diseño con entidades gráficas	Repositorio con información del proyecto de software
 <b>Componentes Independientes</b>	Módulos que acceden y manipulan datos centrales	Herramientas que operan en el archivo de diseño central	Herramientas que interactúan con el repositorio central

# Organización del Sistema

## *Patrón* de repositorio



10

# Organización del Sistema

## ***Patrón*** de repositorio

### *Ventajas*

- ❖ Forma eficiente de compartir grandes cantidades de datos, no hay necesidad de transmitir datos de un subsistema a otro
- ❖ Los subsistemas que producen datos no deben saber como se utilizan
- ❖ Las actividades de backup, protección, control de acceso están centralizadas.
- ❖ El modelo compartido es visible a través del esquema del repositorio. Las nuevas herramientas se integran de forma directa, ya que son compatibles con el modelo de datos

11

# Organización del Sistema

## ***Patrón*** de repositorio

### *Desventajas*

- ❖ Los subsistemas deben estar acordes a los modelos de datos del repositorio. Esto en algunos casos puede afectar el rendimiento.
- ❖ La evolución puede ser difícil a medida que se genera un gran volumen de información de acuerdo con el modelo de datos establecido. La migración de estos modelos puede ser muy difícil, en algunos casos imposible.
- ❖ Diferentes subsistemas pueden tener distintos requerimientos de protección o políticas de seguridad y el modelo de repositorio impone las mismas para todos.
- ❖ Es difícil distribuir el repositorio en varias máquinas, existen repositorios centralizados lógicamente pero pueden ocasionar problemas de redundancia e inconsistencias.

12

# Organización del Sistema

## **Patrón** cliente-servidor

*Es un modelo donde el sistema se organiza como un conjunto de servicios y servidores asociados, más unos clientes que utilizan los servicios*

### *Componentes*

Un conjunto de servidores que ofrecen servicios

Un conjunto de clientes que llaman a los servicios

Una red que permite a los clientes acceder a los servicios

Caso particular cuando los servicios y el cliente corren en la misma máquina

*Los clientes conocen el nombre del servidor y el servicio que brinda, pero el servidor no necesita conocer al cliente*

13

# Ejemplos de Arquitecturas de Software en Capas

## Aplicaciones web con bases de datos

Aplicaciones web con bases de datos usan múltiples capas.



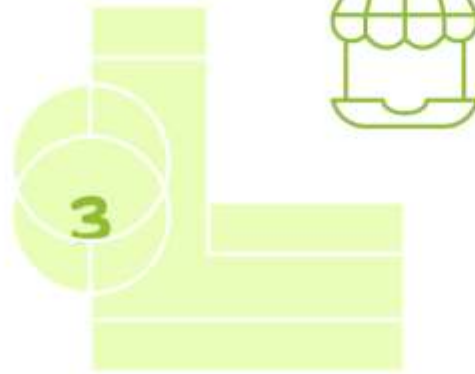
## Aplicaciones de gestión de inventario

Aplicaciones de gestión de inventario separan interfaz y lógica de datos.



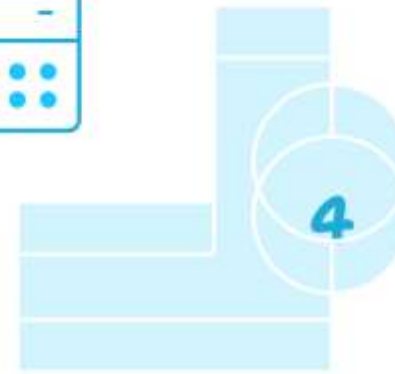
## Plataformas de comercio electrónico

Plataformas de comercio electrónico organizan capas para pedidos.



## Aplicaciones de contabilidad de escritorio

Aplicaciones de contabilidad de escritorio separan interfaz y datos.

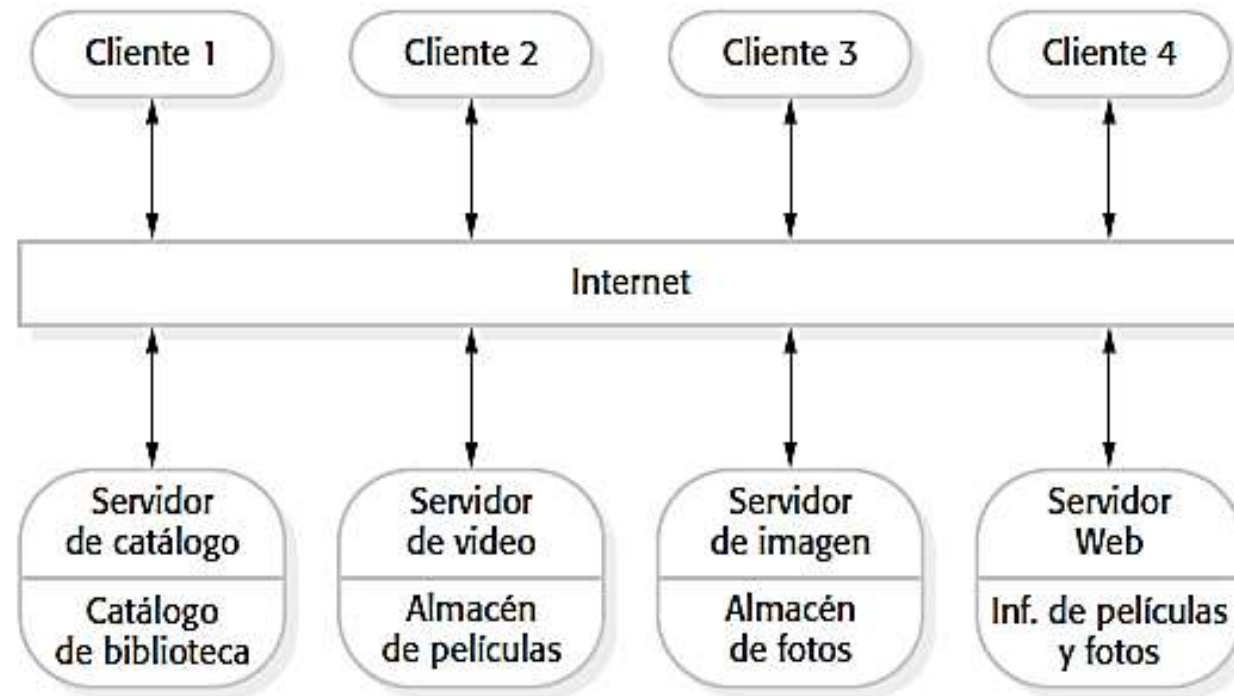


Made with Napkin



# Organización del Sistema

## *Patrón* cliente-servidor



15

# Organización del Sistema

## » **Patrón** de arquitectura en capas

El sistema se organiza en capas, donde cada una de ellas presenta un conjunto de servicios a sus capas adyacentes

### Ventajas

- ❖ *Soporta el desarrollo incremental*
- ❖ *Es portable y resistente a cambios*
- ❖ *Una capa puede ser reemplazada siempre que se mantenga la interfaz, y si varía la interfaz se genera una capa para adaptarlas*
- ❖ *Permite generar sistemas multiplataforma, ya que solamente las capas más internas son dependientes de la plataforma (se genera una capa interna para cada plataforma)*

16

# Organización del Sistema

---

## » **Patrón** de arquitectura en capas

### Desventajas

- ❖ *Difícil de estructurar*
- ❖ *Las capas internas proporcionan servicios que son requeridos por todos los niveles*
- ❖ *Los servicios requeridos por el usuario pueden estar brindados por las capas internas teniendo que atravesar varias capas adyacentes*
- ❖ *Si hay muchas capas, un servicio solicitado de la capa superior puede tener que ser interpretado varias veces en diferentes capas*

17

## Arquitectura de Aplicaciones en Capas

### Capa de Presentación

Interactúa con el usuario y muestra información



### Capa de Lógica de Negocio

Procesa datos y aplica reglas de negocio



### Capa de Acceso a Datos

Gestiona la interacción con el origen de datos



### Capa de Base de Datos

Almacena y recupera datos de manera segura

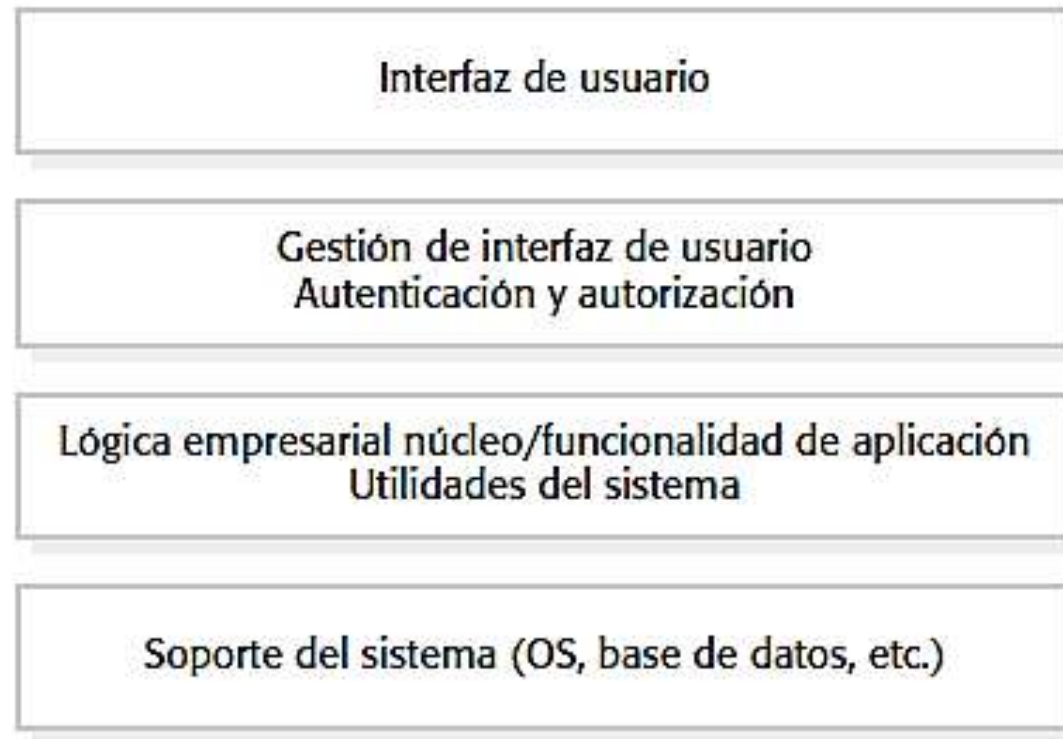


Capas típicas del software

18

# Organización del Sistema

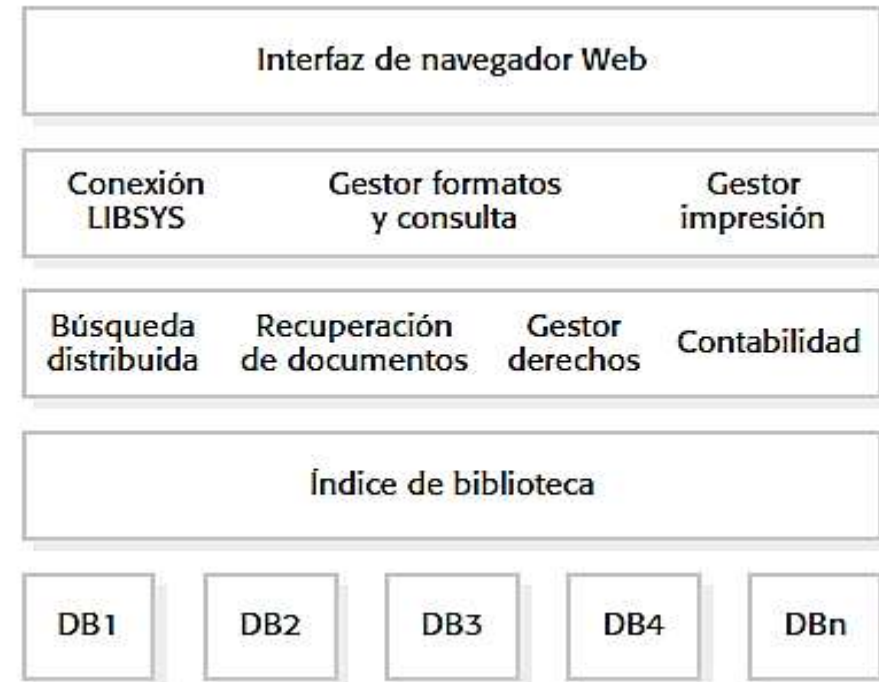
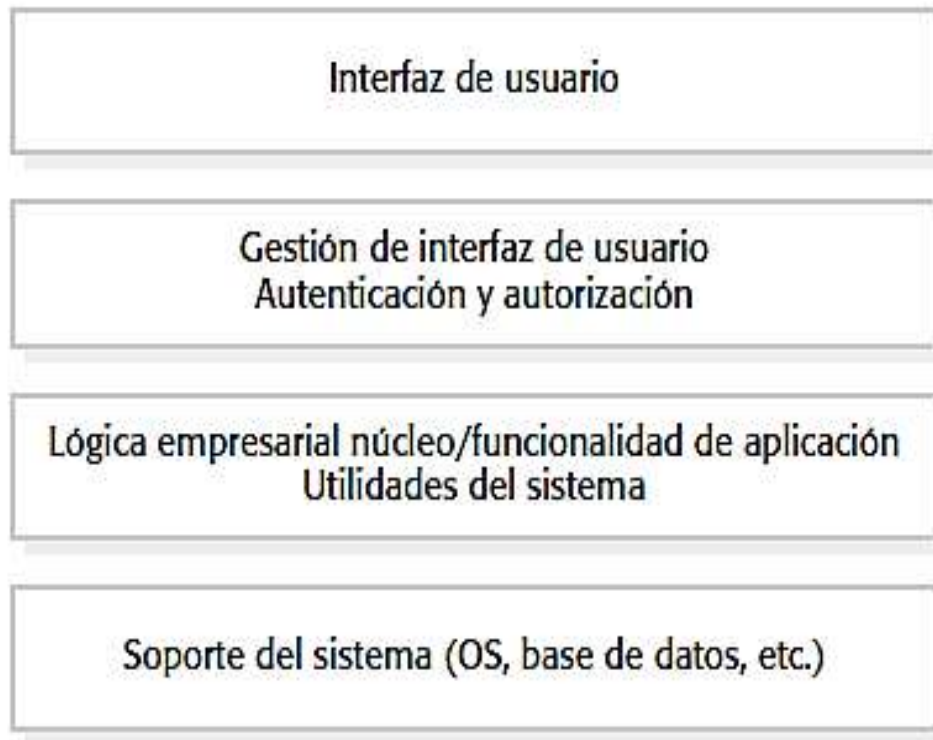
## » *Patrón* de arquitectura en capas



19

# Organización del Sistema

»Ejemplo de **Patrón** de arquitectura en capas:



20



1. Organización del sistema
2. Descomposición modular
3. Modelos de control
4. Arquitectura de los Sistemas Distribuidos

# Descomposición Modular

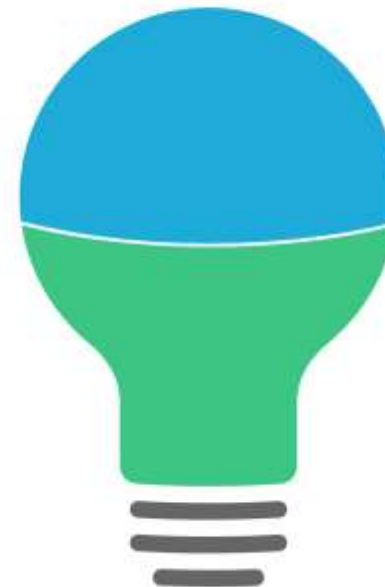
» Una vez organizado el sistema, a los subsistemas los podemos dividir en módulos, se puede aplicar los mismos criterios que vimos en la organización, pero la descomposición modular es más pequeña y permite utilizar otros estilos alternativos.

## Estrategias de Descomposición Modular

### Descomposición Orientada a Flujo de Funciones



Enfoque en la transformación de datos a través de módulos funcionales



### Descomposición Orientada a Objetos

Enfoque en la comunicación entre objetos

22

# Descomposición Modular

---

## » Definiciones

### Subsistema

*Es un sistema en sí mismo cuyo funcionamiento no depende de los servicios proporcionados por otros. Los subsistemas se componen de módulos con interfaces definidas que se utilizan para comunicarse con otro subsistemas.*

### Módulo

*Es un componente de un subsistema que proporciona uno o más servicios a otros módulos. A su vez utiliza servicios proporcionados por otros módulos. Por lo general no se los considera un sistema independiente.*

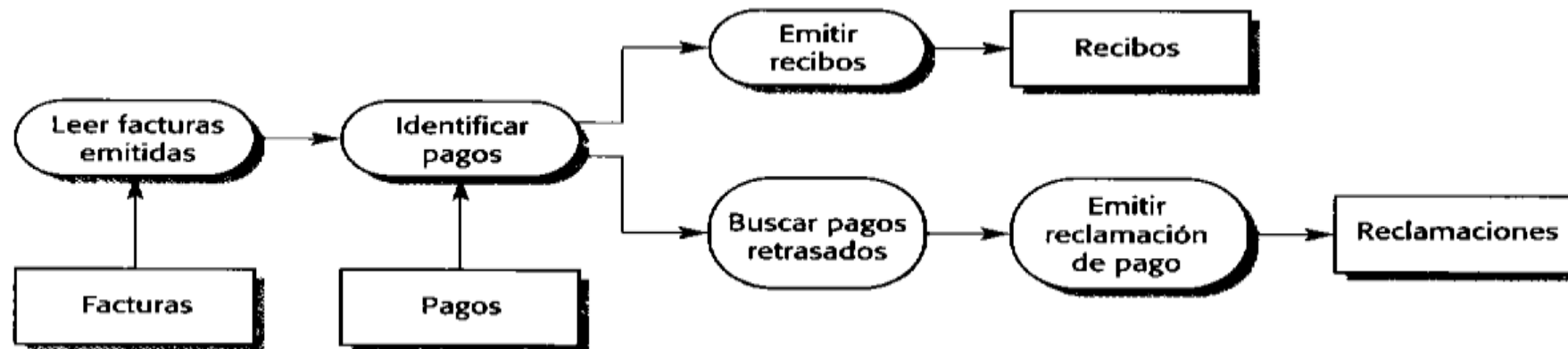
23

# Descomposición Modular

## » Descomposición orientada a flujo de funciones

*En un Modelo orientado a flujo de funciones , los datos fluyen de una función a otra y se transforman a medida que pasan por una secuencia de funciones hasta llegar a los datos de salida. Las transformaciones se pueden ejecutar en secuencial o en paralelo.*

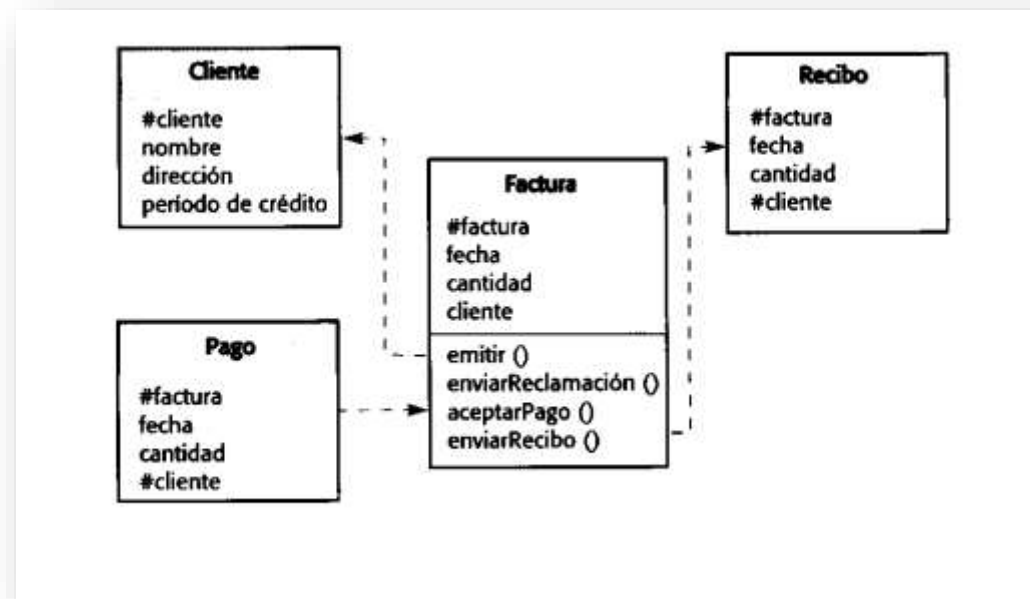
24



# Descomposición Modular

## » Descomposición orientada a objetos

Un modelo arquitectónico orientado a objetos estructura al sistema en un conjunto de objetos débilmente acoplados y con interfaces bien definidas.



25

1. Organización del sistema
2. Descomposición modular
3. Modelos de control
4. Arquitectura de los Sistemas Distribuidos



# Modelos de Control

» En un sistema, los subsistemas están controlados para que sus servicios se entreguen en el lugar correcto en el momento preciso.



27

# Modelos de Control

## » **Control** Centralizado

Un subsistema se diseña como controlador y tiene la responsabilidad de gestionar la ejecución de otros subsistemas, la ejecución puede ser secuencial o en paralelo

28

### *Modelo de llamada y retorno*

Modelo de subrutinas descendentes

Aplicable a modelos secuenciales

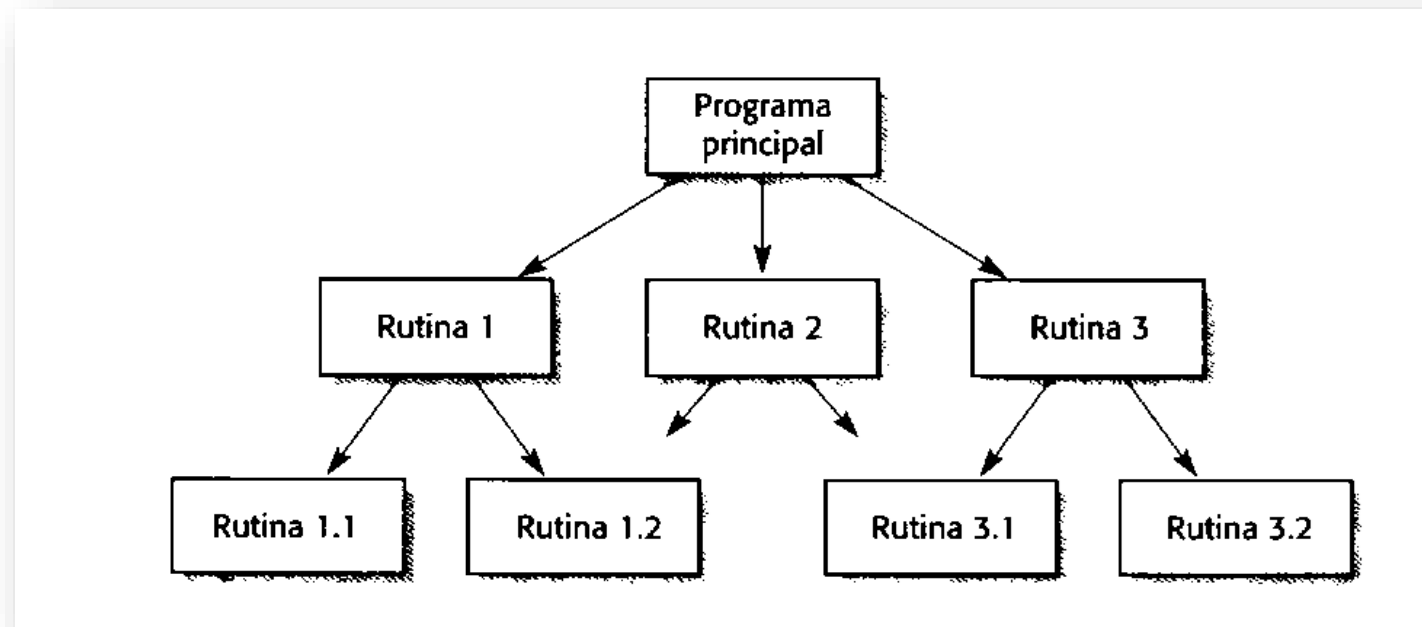
### *Modelo de gestor*

Un gestor controla el inicio y parada coordinado con el resto de los procesos

Aplicable a modelos concurrentes

# Modelos de Control

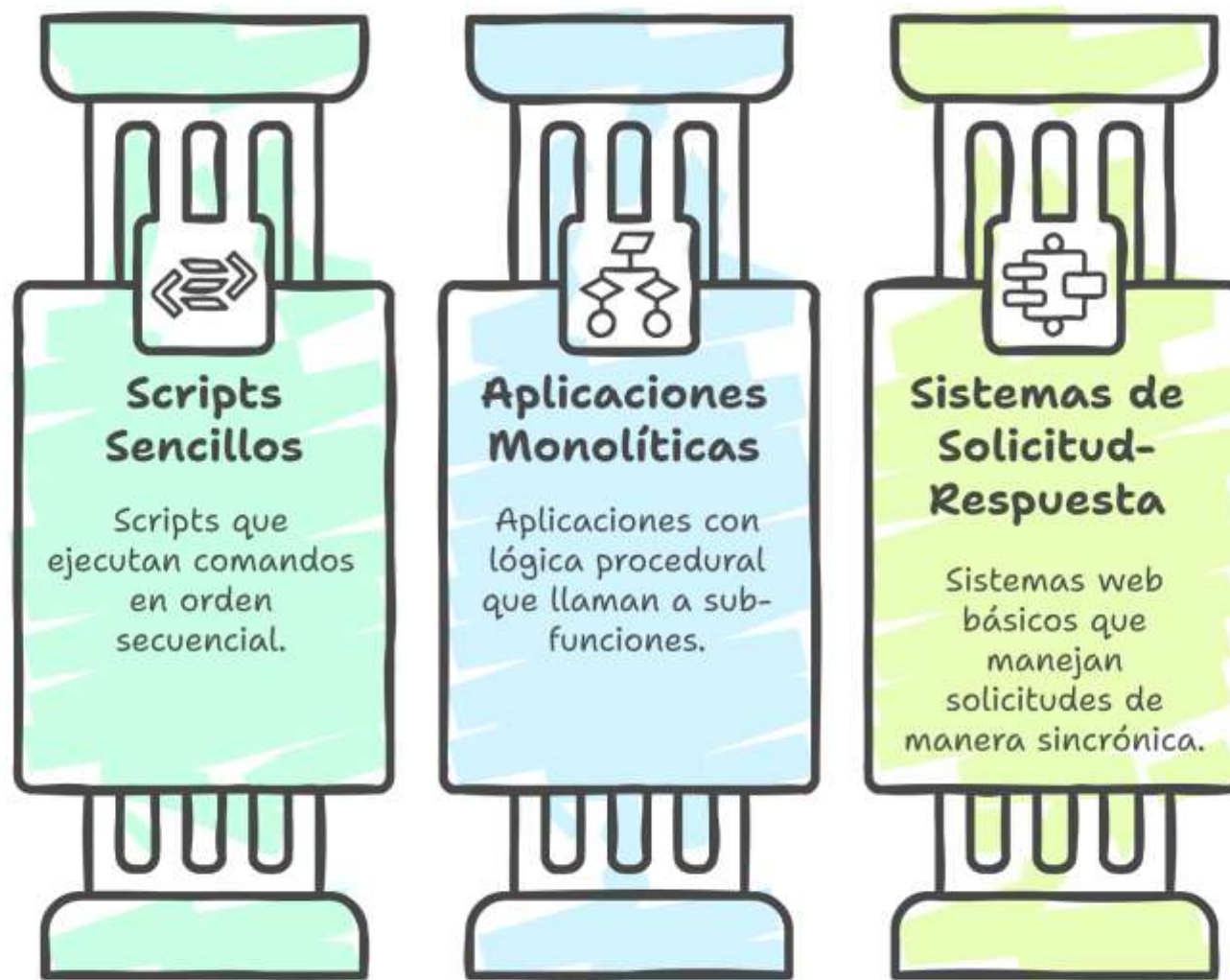
## » **Control** Centralizado Modelo de llamada y retorno



29

## Estructura de Llamada y Retorno

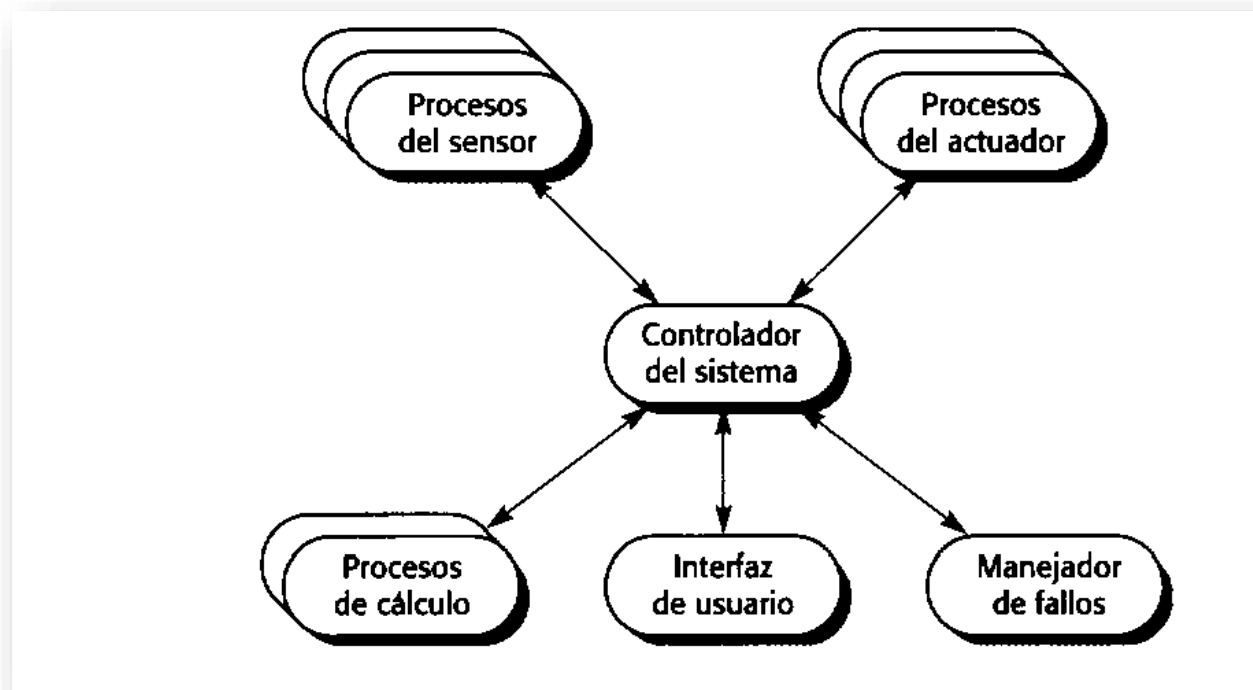
### Ejemplos de control centralizado- Llamada retorno



30

# Modelos de Control

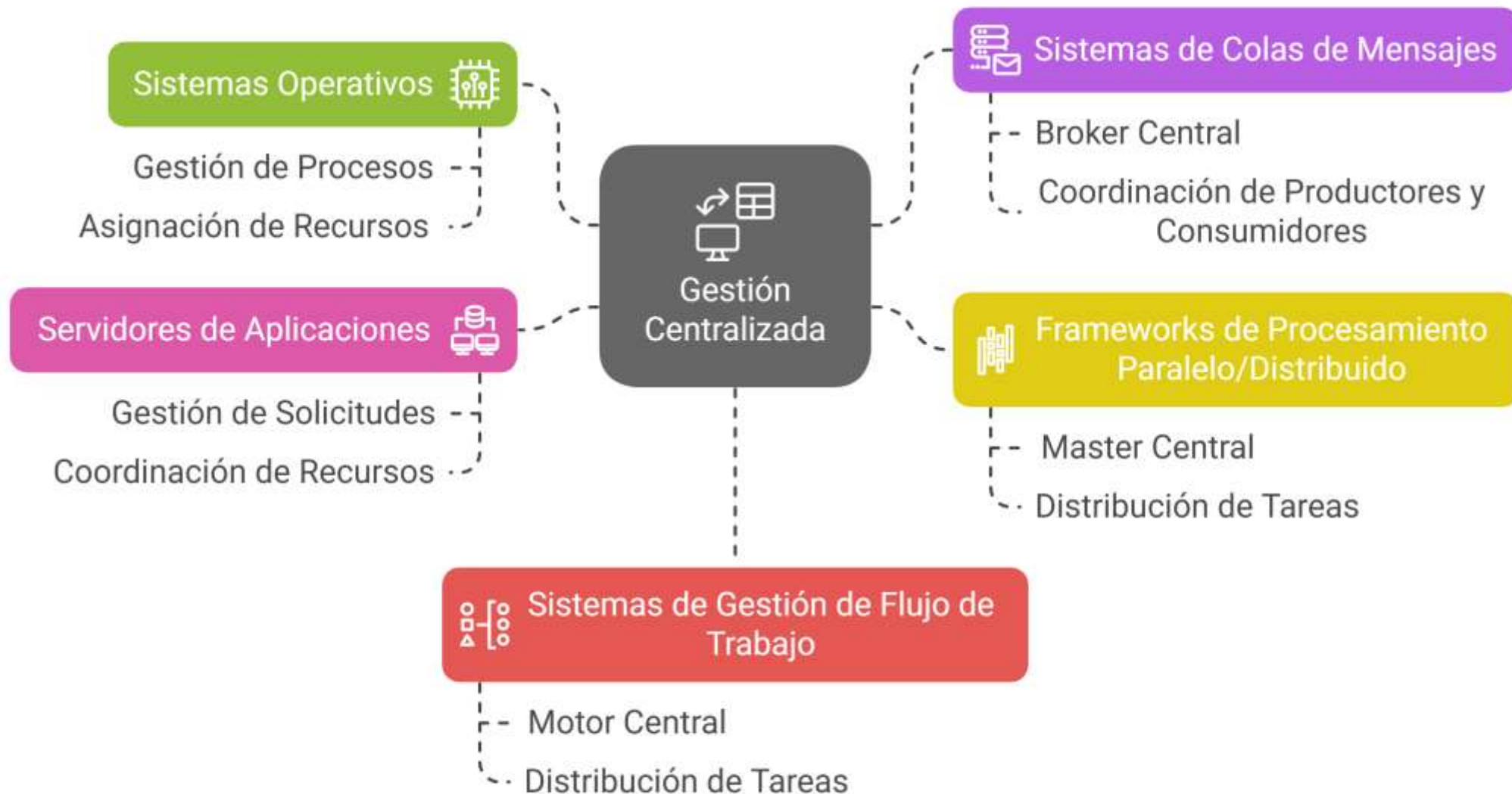
## » **Control** Centralizado Modelo de gestor



31

## Gestión Centralizada en Sistemas de Software

### Ejemplos de control centralizado – Modelo de gestor





# Modelos de Control

---

## » Sistemas Dirigidos Por Eventos

- Se rigen por eventos generados externamente al proceso

- Eventos

*Señal binaria*

*Un valor dentro de un rango*

*Una entrada de un comando*

*Una selección del menú*

- Modelos de sistemas dirigidos por eventos

*Modelos de transmisión (Broadcast)*

*Modelo dirigido por interrupciones*

33

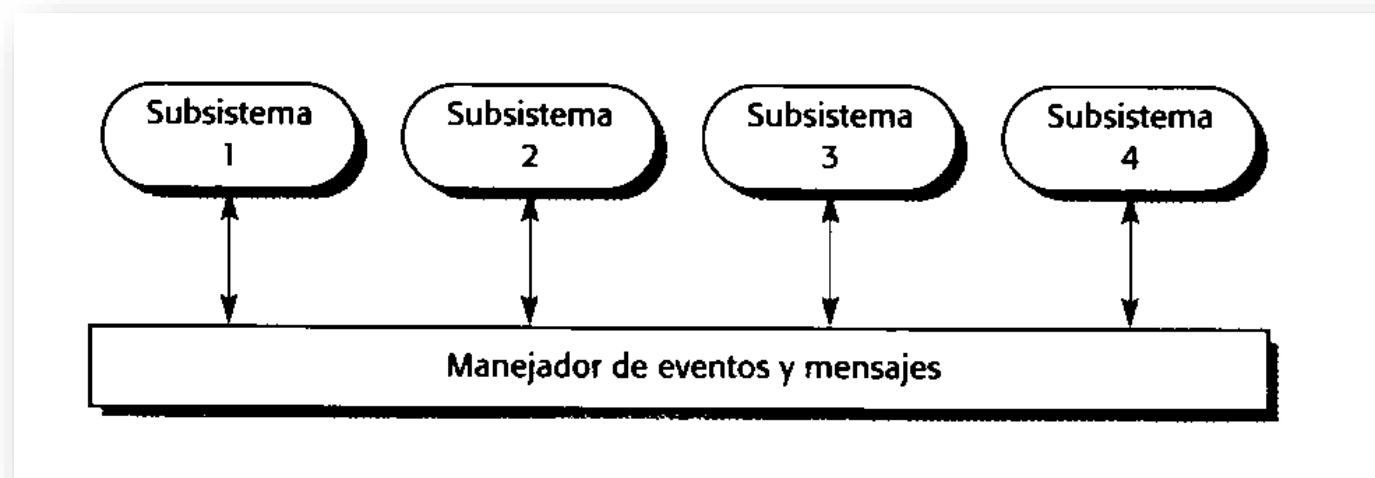
# Modelos de Control

## » Sistema **Dirigido** Por Eventos

### Modelos de transmisión (Broadcast)

*Un evento se trasmite a todos los subsistemas, cualquier subsistema programado para manejar ese evento lo atenderá*

34



## Ejemplos de Arquitectura Basada en Eventos por broadcast

### Ejemplo de modelo de control por eventos basado en broadcast

#### Interfaces de Usuario Gráficas



Interacciones del usuario con elementos de la interfaz de usuario

#### Sistemas de Monitoreo y Alerta



Detección y respuesta a eventos del sistema



#### Sistemas de Notificación y Mensajería

Envío de mensajes a múltiples suscriptores



#### Arquitecturas de Microservicios Desacopladas

Comunicación entre microservicios sin conocimiento directo

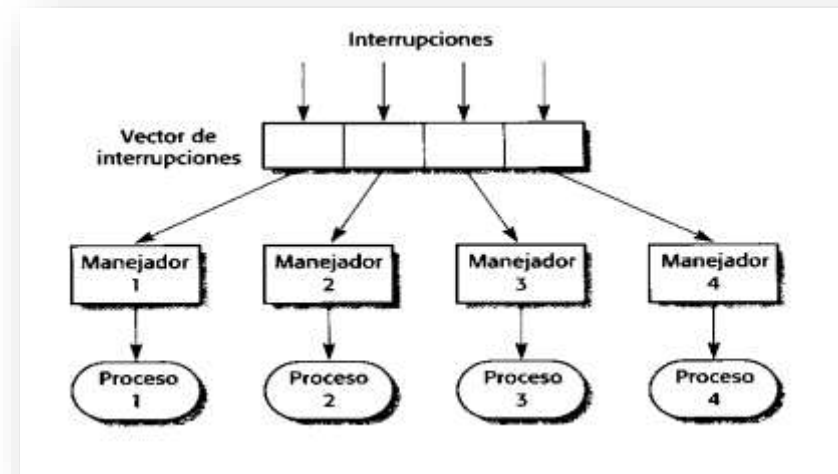
# Modelos de Control

## » Sistema **Dirigido** Por Eventos

### Modelo **Dirigido** por interrupciones

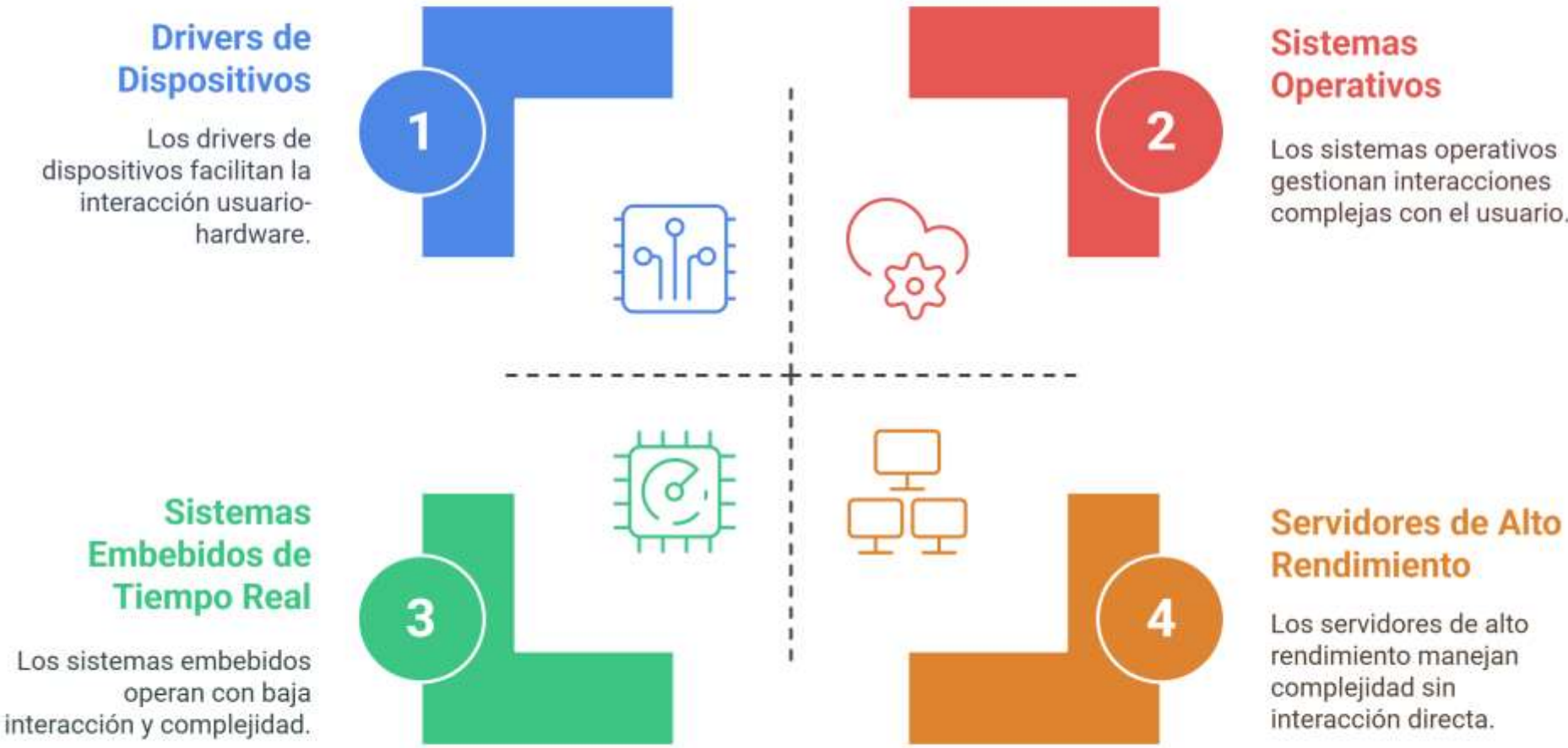
*Se utilizan en sistemas de tiempo real donde las interrupciones externas son detectadas por un manejador de interrupciones y se envía a algún componente para su procesamiento*

36



# Ejemplos de modelos de control dirigido por interrupciones

## Aplicaciones de Control Dirigido por Interrupciones



1. Organización del sistema
2. Descomposición modular
3. Modelos de control
4. Arquitectura de los Sistemas Distribuidos

# Arquitectura de los Sistemas Distribuidos

---

» Un sistema distribuido es un sistema en el que el procesamiento de información se distribuye sobre varias computadoras.

» Tipos genéricos de sistemas distribuidos

    Cliente-Servidor

    Componentes distribuidos

39



# Arquitectura de los Sistemas Distribuidos

## »Características de los sistemas distribuidos

### **Compartir recursos**

*Un sistema distribuido permite compartir recursos*

### **Apertura**

*Son sistemas abiertos y se diseñan con protocolos estándar para simplificar la combinación de los recursos*

### **Concurrencia**

*Varios procesos pueden operar al mismo tiempo sobre diferentes computadoras*

### **Escalabilidad**

*La capacidad puede incrementarse añadiendo nuevos recursos para cubrir nuevas demandas*

### **Tolerancia a fallos**

*La disponibilidad de varias computadoras y el potencial para reproducir información hace que los sistemas distribuidos sean más tolerantes a fallos de funcionamiento de hardware y software*

40

# Arquitectura de los Sistemas Distribuidos

## » Desventajas de los sistemas distribuidos

### **Complejidad**

*Son más complejos que los centralizados, además del procesamiento hay que tener en cuenta los problemas de la comunicación y sincronización entre los equipos*

### **Seguridad**

*Se accede al sistema desde varias computadoras generando tráfico en la red que puede ser intervenido*

### **Manejabilidad**

*Las computadoras del sistema pueden ser de diferentes tipos y diferentes S.O. lo que genera más dificultades para gestionar y mantener el sistema*

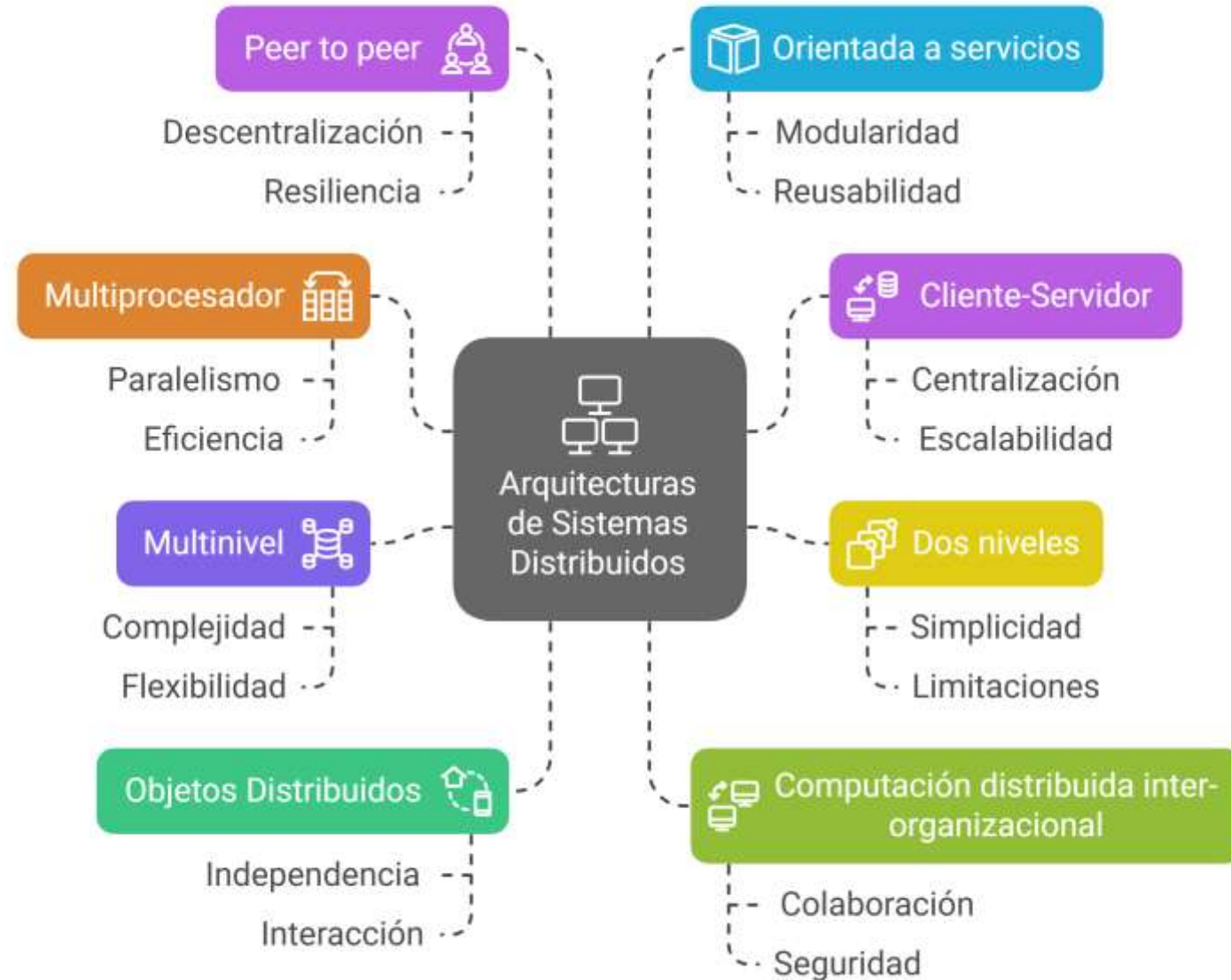
### **Impredecibilidad**

*La respuesta depende de la carga del sistema y del estado de la red, lo que hace que el tiempo de respuesta varíe entre una petición y otra*

41

## Arquitecturas de Sistemas Distribuidos

### Tipos de Arquitectura de los Sistemas Distribuidos (SD)



42

# Arquitectura de los Sistemas Distribuidos

---

## » Arquitectura Multiprocesador

El sistema de software está formado por varios procesos que pueden o no ejecutarse en procesadores diferentes

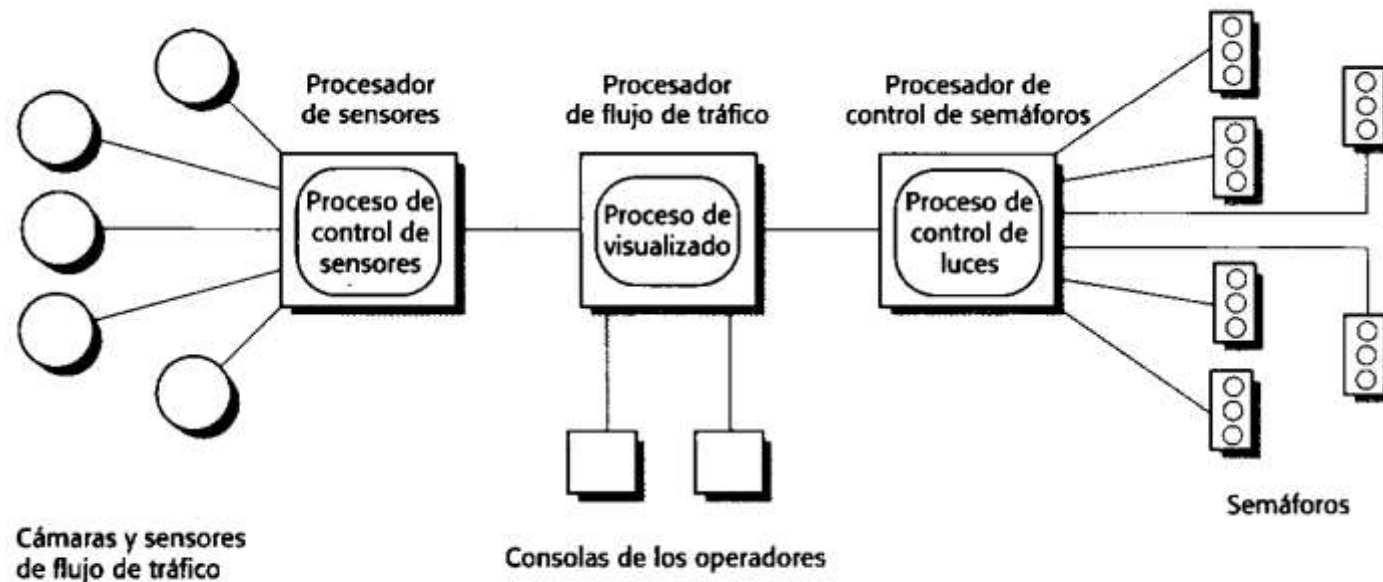
La asignación de los procesos a los procesadores puede ser predeterminada o mediante un dispatcher

Es común en sistemas grandes de tiempo real que recolectan información, toman decisiones y envían señales para modificar el entorno

43

# Arquitectura de los Sistemas Distribuidos

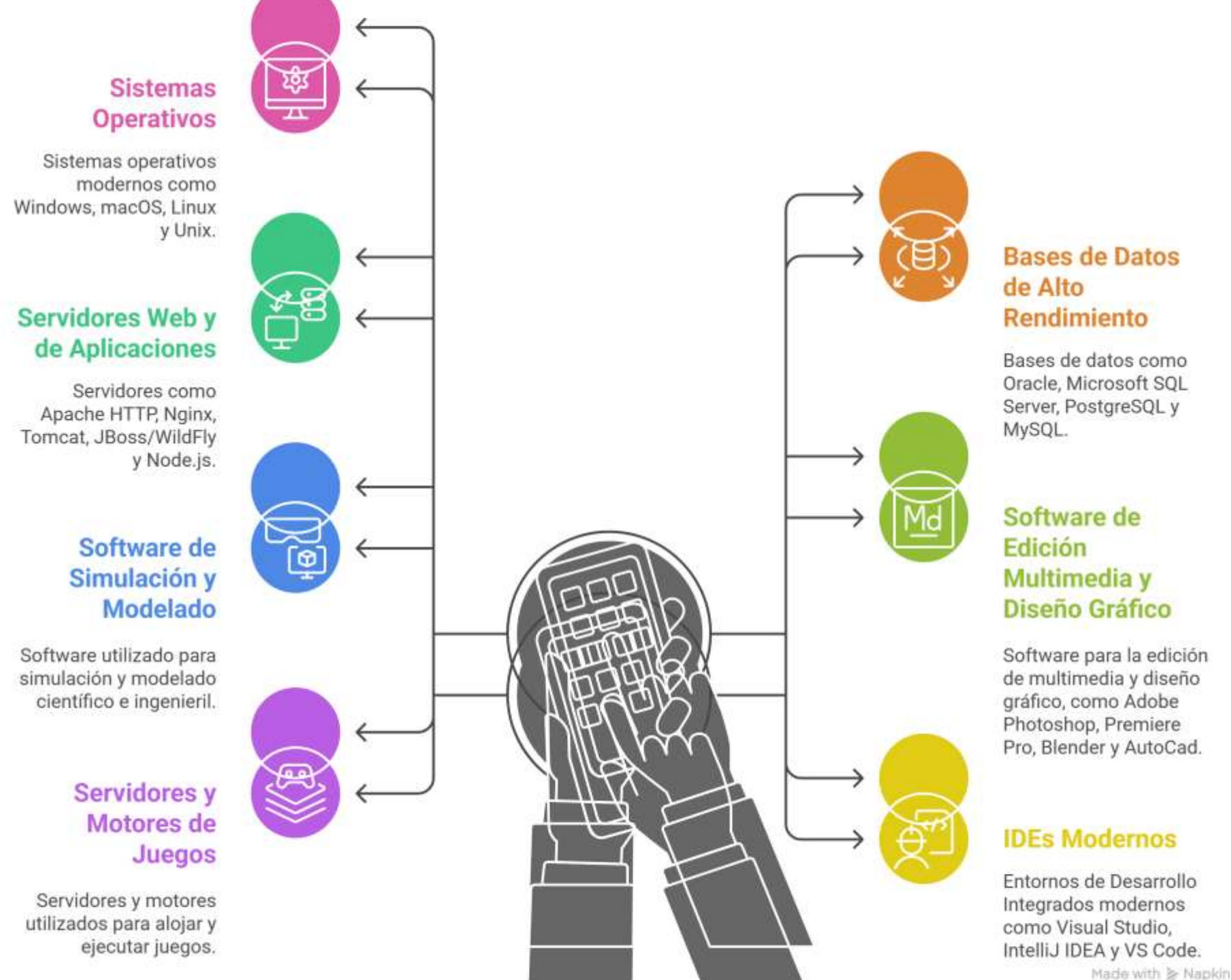
## » Arquitectura Multiprocesador



**Figura 12.1** Un sistema multiprocesador de control de tráfico.

44

# Ejemplos arquitecturas multiprocesador





# Arquitectura de los Sistemas Distribuidos

---

## » Arquitectura Cliente-Servidor (C-S)

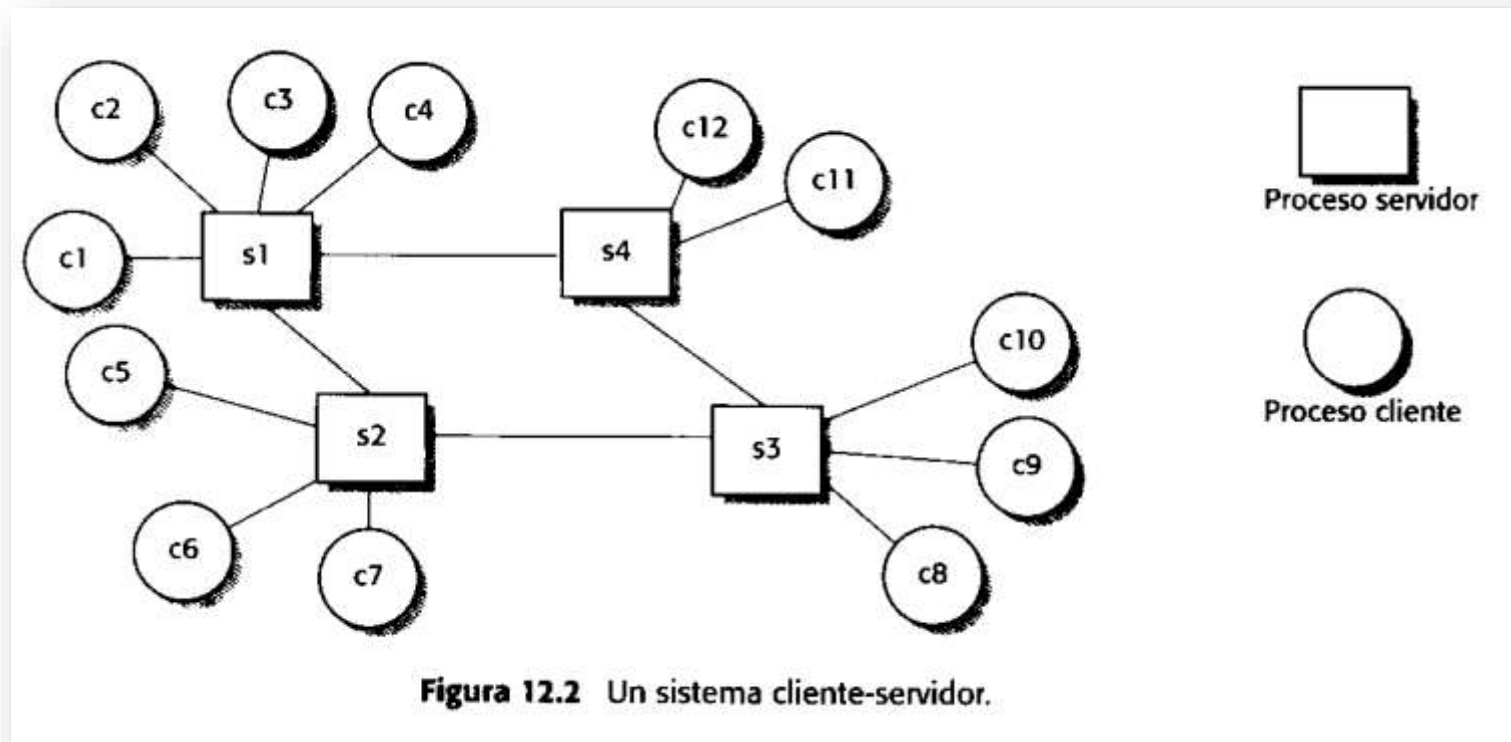
- ❖ Una aplicación se modela como un conjunto de servicios proporcionado por los servidores y un conjunto de clientes que usan estos servicios
- ❖ Los clientes y servidores son procesos diferentes
- ❖ Los servidores pueden atender varios clientes
- ❖ Un servidor puede brindar varios servicios
- ❖ Los clientes no se conocen entre sí

46



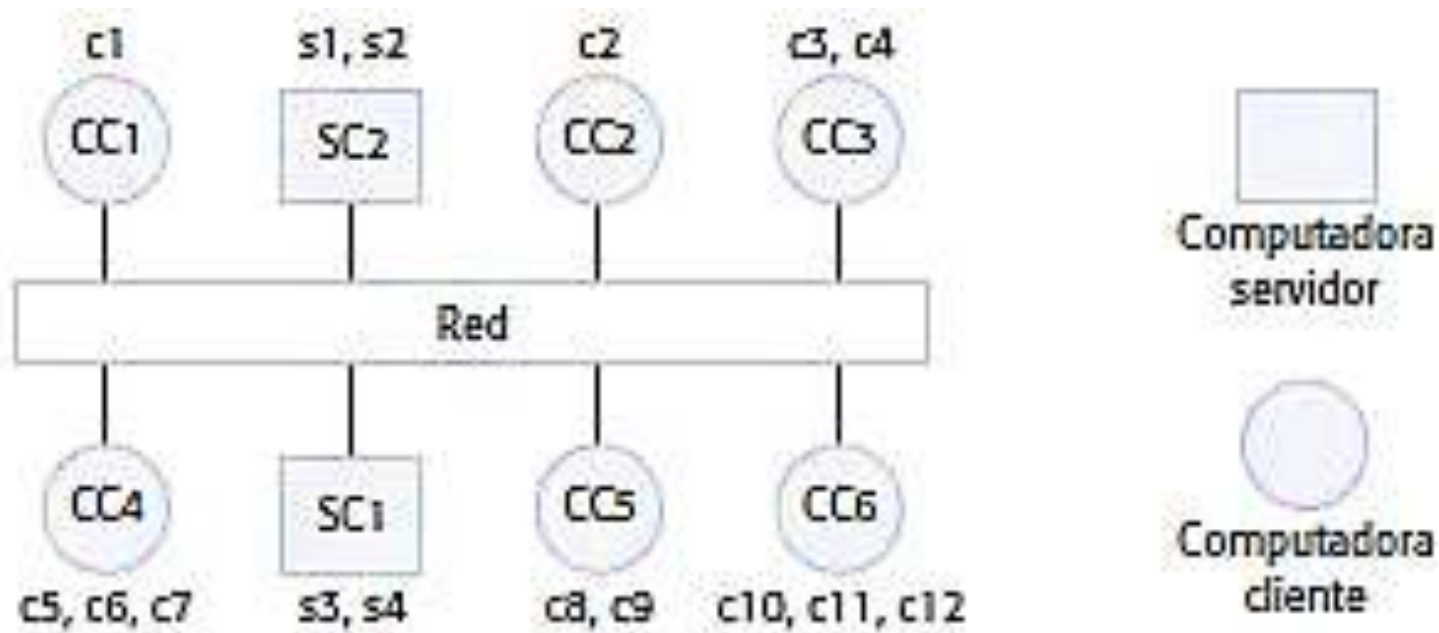
# Arquitectura de los Sistemas Distribuidos

## » Arquitectura Cliente-Servidor



# Arquitectura de los Sistemas Distribuidos

## » Arquitectura C-S



48

# Arquitectura de los Sistemas Distribuidos

---

## » Clasificación de Arquitectura C-S

### *Dos Niveles*

Cliente ligero

El procesamiento y gestión de datos se lleva a cabo en el servidor

Cliente pesado

El cliente implementa la lógica de la aplicación y el servidor solo gestiona los datos

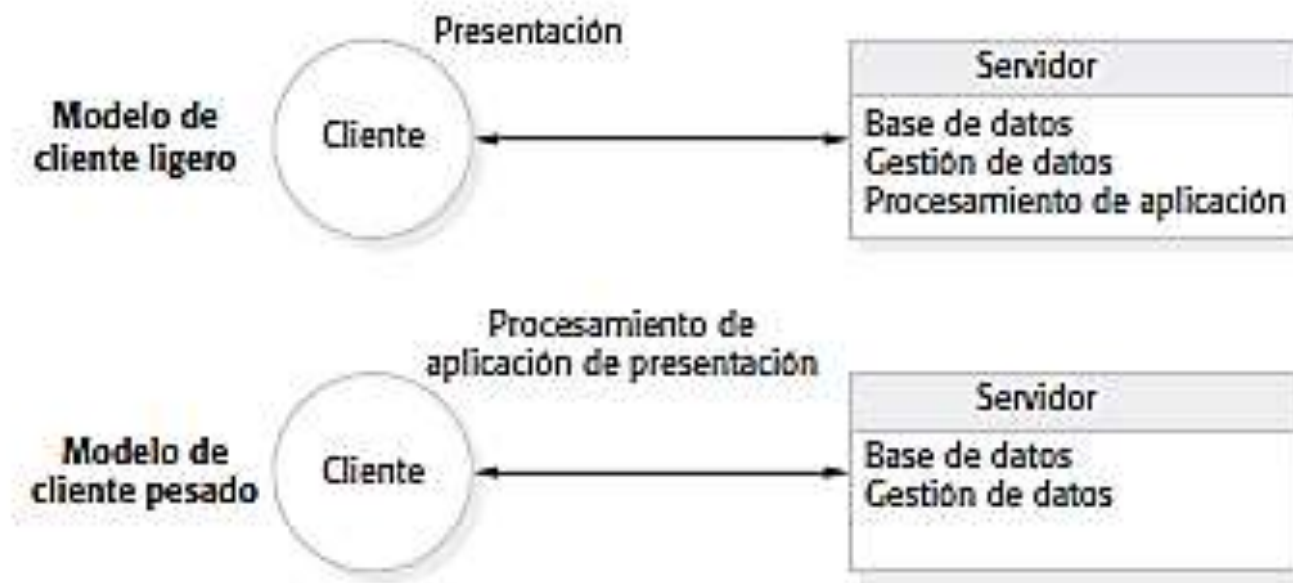
### *Multinivel*

La presentación, el procesamiento y la gestión de los datos son procesos lógicamente separados y se pueden ejecutar en procesadores diferentes

49

# Arquitectura de los Sistemas Distribuidos

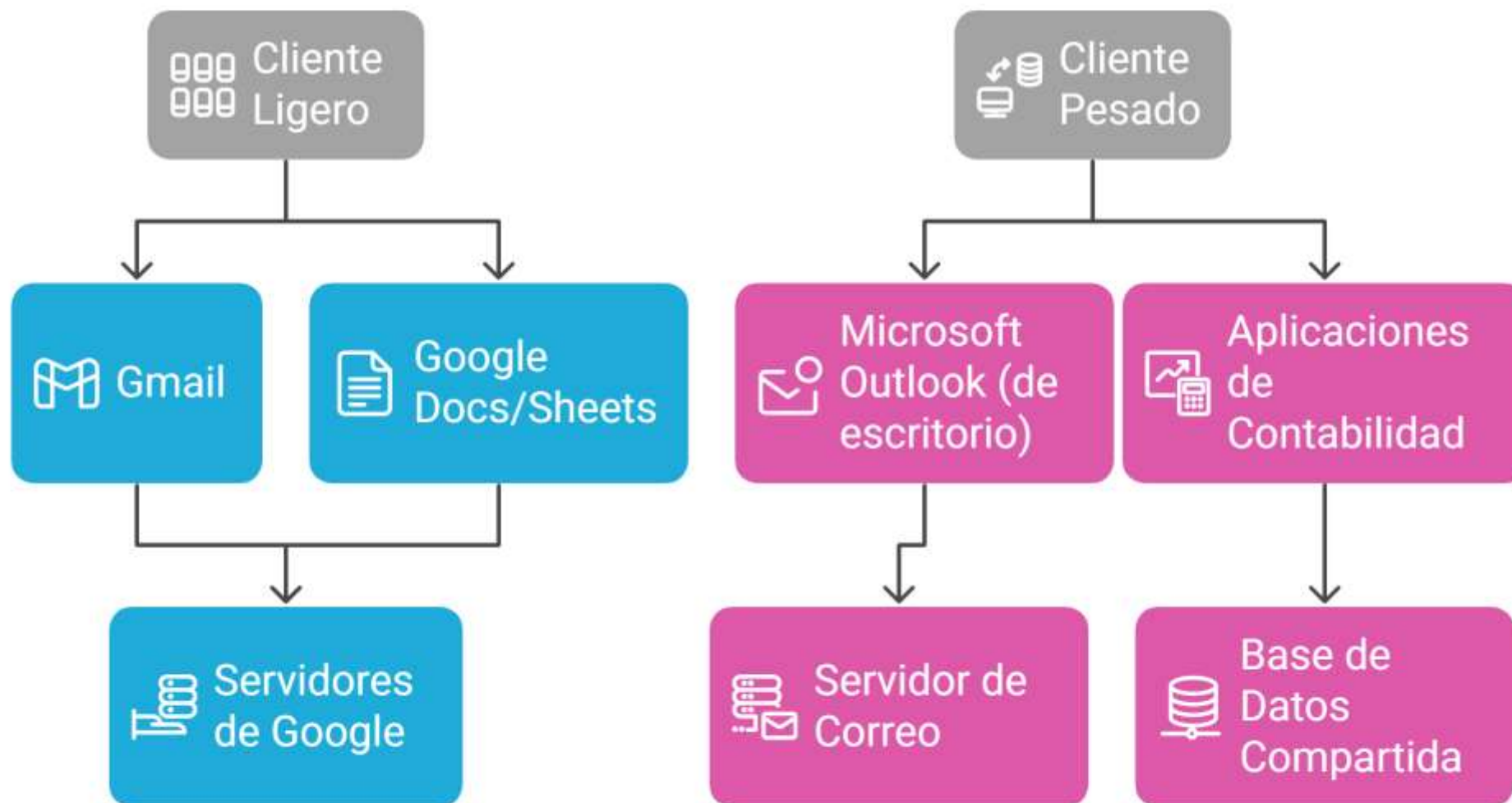
## » Arquitectura C-S *Dos Niveles*



50

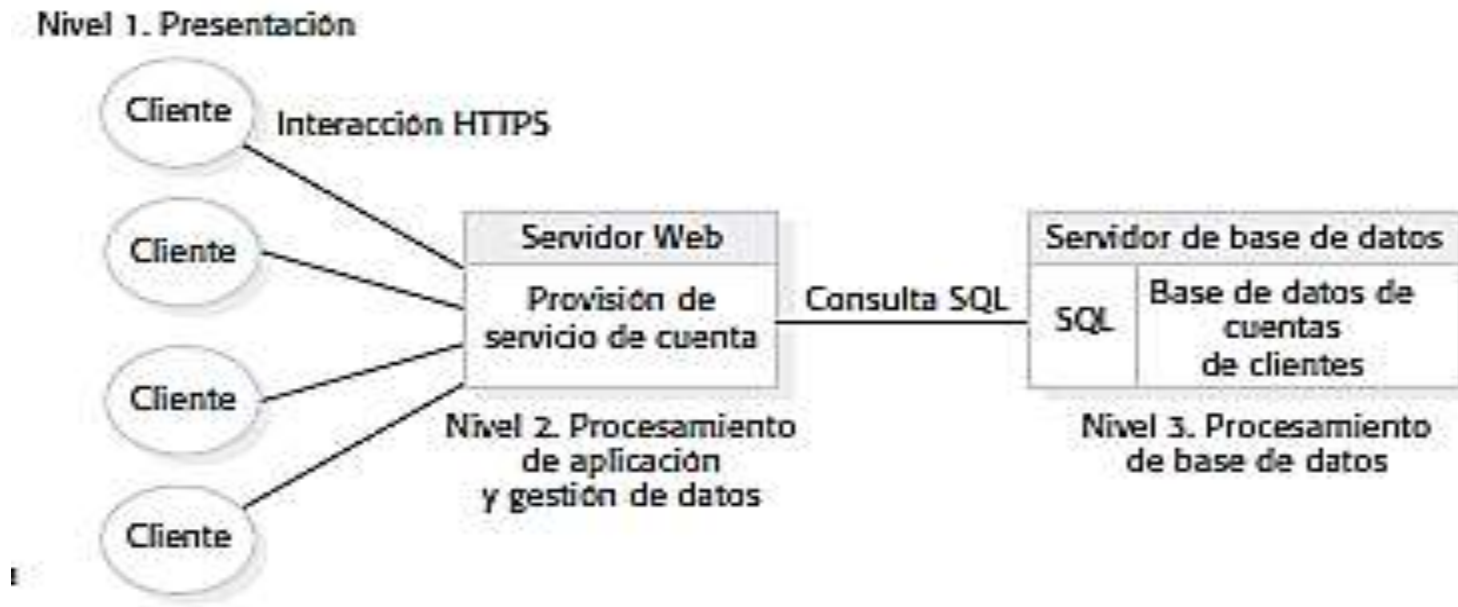
## Arquitectura Cliente-Servidor de Dos Niveles

### Ejemplos



# Arquitectura de los Sistemas Distribuidos

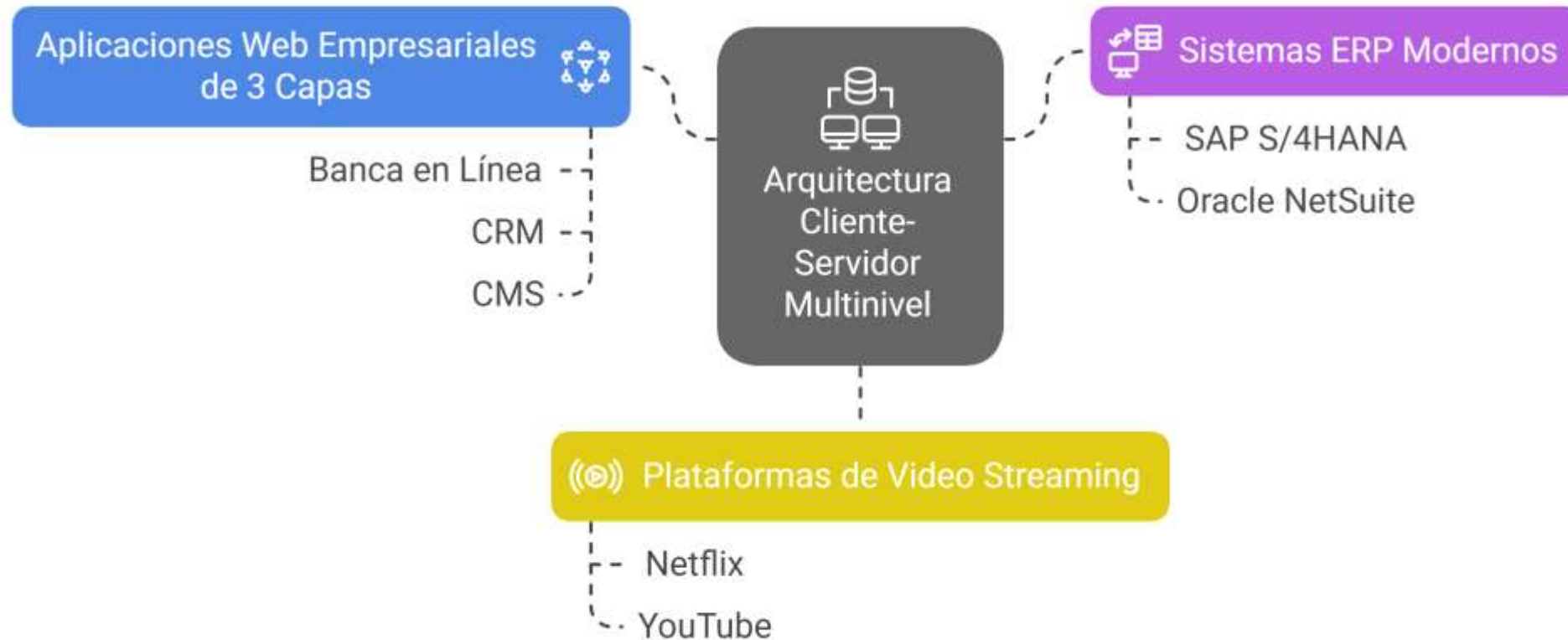
## » Arquitectura C-S *Multinivel*



52

# Ejemplos Arquitectura Cliente – servidor multinivel

## Arquitecturas Cliente-Servidor Multinivel



53



# Arquitectura de los Sistemas Distribuidos

---

## » Arquitectura de Componentes Distribuidos

Diseña al sistema como un conjunto de componentes u objetos que brindan una interfaz de un conjunto de servicios que ellos suministran. Otros componentes u objetos solicitan estos servicios. No hay distinción tajante entre clientes y servidores.

Los componentes pueden distribuirse en varias máquinas a través de la red utilizando un middleware como intermediario de peticiones

54

# Arquitectura de los Sistemas Distribuidos

## »Arquitectura de Objetos Distribuidos



El **middleware** es un software que actúa como una **capa intermedia** entre las aplicaciones y los sistemas operativos o las redes subyacentes. Su función principal es **facilitar la comunicación y la gestión** de los componentes distribuidos, permitiendo que diferentes partes de un sistema interactúen sin tener que conocer los detalles complejos de la infraestructura de red o del sistema operativo de cada una.

55

# Ejemplos de arquitectura de objetos distribuidos

## CORBA son las siglas de **Common Object Request Broker Architecture**

(Arquitectura de Agente de Petición de Objetos Común). Es un estándar definido por el Object Management Group (OMG) diseñado para permitir que componentes de software escritos en diferentes lenguajes de programación y ejecutándose en diferentes máquinas (y sistemas operativos) puedan **interoperar** y trabajar juntos como si fueran objetos locales.



## RMI son las siglas de **Remote Method Invocation** (Invocación de Método Remoto).

Es una API (Interfaz de Programación de Aplicaciones) nativa del lenguaje de programación **Java** que permite que un objeto que se ejecuta en una Máquina Virtual Java (JVM) pueda **invocar métodos** de un objeto que se está ejecutando en **otra JVM diferente**. Estas JVMs pueden estar en la misma máquina física o en máquinas separadas conectadas por una red.

## DCOM son las siglas de **Distributed Component Object Model** (Modelo de Objetos Componentes Distribuidos).

Es una tecnología desarrollada por **Microsoft** que permite que componentes de software (objetos COM) se comuniquen directamente a través de una red. Es, esencialmente, una **extensión del Component Object Model (COM)** de Microsoft para soportar la comunicación distribuida.

# Arquitectura de los Sistemas Distribuidos

---

## » Computación Distribuida inter-organizacional

Una organización tiene varios servidores y reparte su carga computacional entre ellos.

Extender este concepto a varias organizaciones.

57

Pueden ser arquitecturas del tipo:

- ☐ *Peer-to-Peer*
- ☐ *Orientados a servicios*

# Arquitectura de los Sistemas Distribuidos

## » Computación Distribuida inter-organizacional

Arquitecturas Peer-to-Peer (P2P)

Sistemas descentralizados en los que el cálculo puede llevarse a cabo en cualquier nodo de la red

Se diseñan para aprovechar la ventaja de la potencia computacional y el almacenamiento a través de una red

Pueden utilizar una arquitectura

*Descentralizada*

donde cada nodo rutea los paquetes a sus vecinos hasta encontrar el destino

*Semi-centralizada*

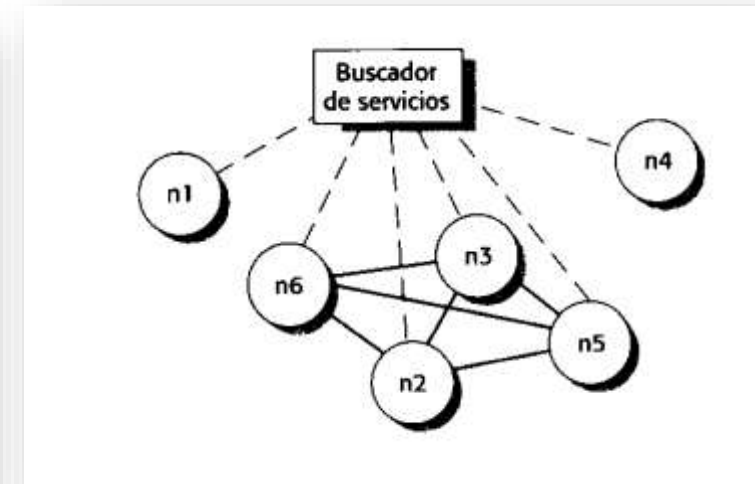
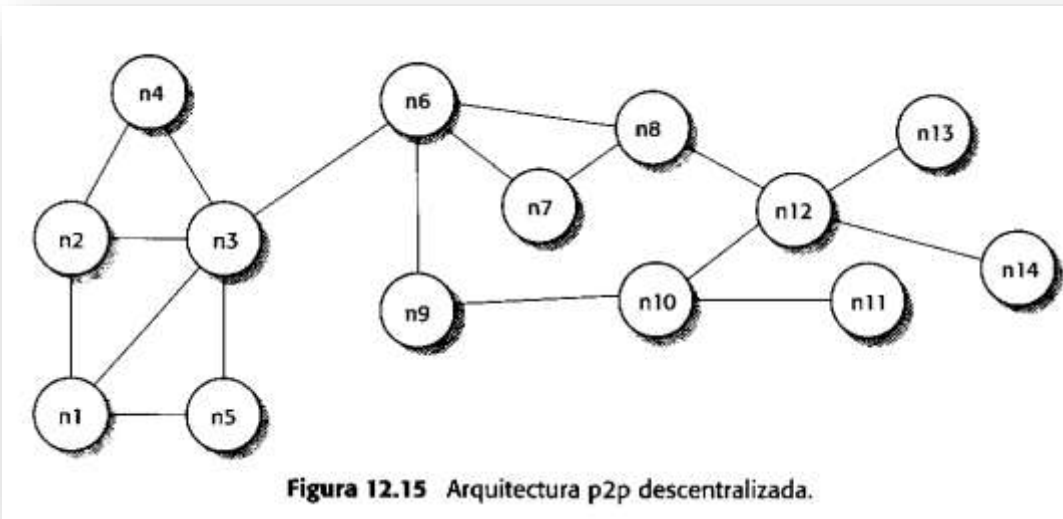
donde un servidor ayuda a conectarse a los nodos o coordinar resultados

58

# Arquitectura de los Sistemas Distribuidos

## » Computación Distribuida inter-organizacional Arquitecturas Peer-to-Peer (P2P)

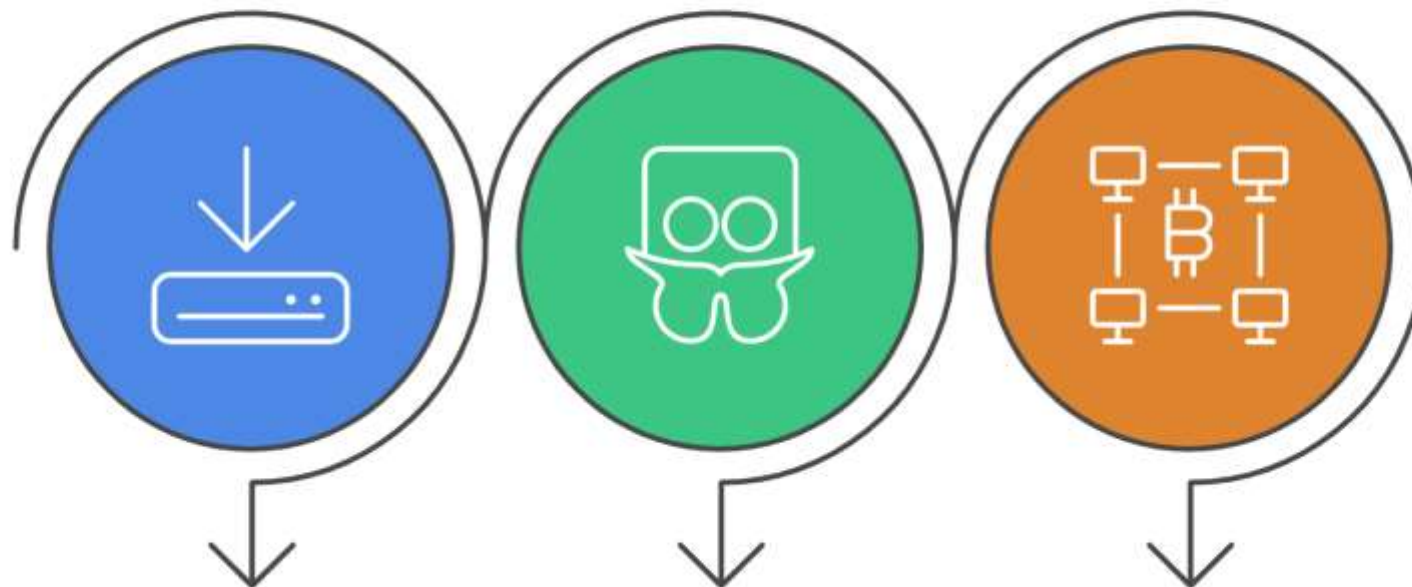
59





## Ejemplos de Software P2P

### Ejemplos de P2P



#### Cientes de BitTorrent

Software que implementa el protocolo BitTorrent

#### Skype

Aplicación que utiliza P2P para llamadas directas

#### Sistemas de Criptomonedas

Redes que operan con software P2P

60



# Arquitectura de los Sistemas Distribuidos

## » Computación Distribuida inter-organizacional Arquitectura de sistemas orientadas a servicios

### *Servicio*

- ❖ Representación de un recurso computacional o de información que puede ser utilizado por otros programas.
- ❖ Un servicio es independiente de la aplicación que lo utiliza
- ❖ Un servicio puede ser utilizado por varias organizaciones
- ❖ Una aplicación puede construirse enlazando servicios
- ❖ Las arquitecturas de las aplicaciones de servicios web son arquitecturas débilmente acopladas

61

# Arquitectura de los Sistemas Distribuidos

## » Computación Distribuida inter-organizacional

### Arquitectura de sistemas orientadas a servicios

#### *Funcionamiento*

- ❖ Un proveedor de servicios oferta servicios definiendo su interfaz y su funcionalidad
- ❖ Para que el servicio sea externo, el proveedor publica el servicio en un “registro de servicio” con información del mismo
- ❖ Un solicitante enlaza este servicio a su aplicación, es decir
  - ❖ que el solicitante incluye el
  - ❖ código para invocarlo y procesa el
  - ❖ resultado del mismo



62

## Ejemplos de Computación Distribuida inter-organizacional

### Arquitectura de sistemas orientadas a servicios

Tipo de Integración	Ejemplos
 <b>Integración de Aplicaciones Empresariales (EAI) y Buses de Servicios Empresariales (ESB)</b>	IBM Integration Bus, Oracle Service Bus
 <b>Aplicaciones Empresariales Modulares y de Gran Escala</b>	SAP S/4HANA, Oracle Fusion Applications
 <b>Servicios Web Públicos y APIs de Terceros</b>	PayPal, Google Maps Platform
 <b>Plataformas de Software como Servicio (SaaS) que exponen APIs</b>	Dropbox, Google Drive
 <b>Sistemas de Gobierno Electrónico y Servicios Públicos Digitales</b>	Plataformas para la interacción ciudadano-gobierno



# Ingeniería de software II

Codificación

# Codificación

- » Una vez establecido el diseño, se deben escribir los programas que implementen dicho diseño.
- » Esto puede resultar una tarea compleja por distintos motivos:
  - Los diseñadores pueden no haber tenido en cuenta las particularidades de la plataforma y el ambiente de programación.
  - Las estructuras e interrelaciones que son fáciles de describir mediante diagramas, no siempre resultan sencillas de escribir en código.
  - Es indispensable escribir el código de forma que resulte comprensible para otras personas.
  - Se deben sacar beneficios de las características de diseño creando código que sea reutilizable.

65



# Codificación: Pautas Generales

» Resultan útiles para conservar la calidad del diseño en la codificación:

**Localización de entrada y salida:** es deseable localizarlas en componentes separados del resto del código ya que generalmente son más difíciles de probar.

**Inclusión de pseudocódigo:** Es útil avanzar el diseño, realizando un pseudocódigo para adaptar el diseño al lenguaje elegido.

**Revisión y reescritura,** no a los remiendos: Es recomendable realizar un borrador, revisarlo y reescribirlo tantas veces como sea necesario.

**Reutilización:** Hay dos clases de reutilización:

*Productiva: se crean componentes destinados a ser reutilizados por otra aplicación*

*Consumidora: Se usan componentes originalmente desarrollados para otros proyectos.*

66

# Codificación: Documentación

---

» Se considera como Documentación del programa al conjunto de descripciones escritas que explican al lector qué hace el programa y cómo lo hace.

» Se divide en:

**Documentación interna:** Es concisa, escrita en un nivel apropiado para un programador. Contiene información dirigida a quienes leerán el código fuente. Incluye información de algoritmos, estructuras de control, flujos de control.

**Documentación externa:** Se prepara para ser leída por quienes, tal vez, nunca verán el código real. Por ejemplo, los diseñadores, cuando evalúan modificaciones o mejoras.

67