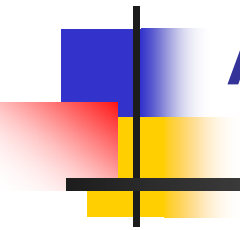
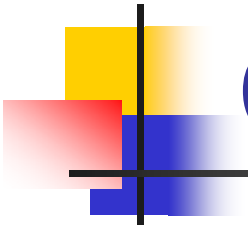


Arquitecturas de Software

Cesar Julio Bustacara Medina

Pontificia Universidad Javeriana

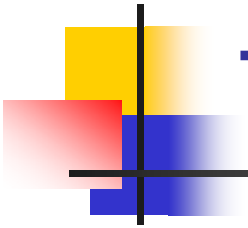
2008



Concepto de patrón

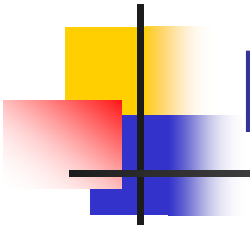
Una solución probada que se puede aplicar con éxito a un determinado tipo de problemas que aparecen repetidamente en el desarrollo de software.

- Es una **solución** a un **problema** en un **contexto**.
- Permiten reutilizar soluciones a problemas comunes.
- Son un esqueleto básico que cada diseñador adapta a las particularidades de su aplicación.



Tipos de patrones

- **Patrones de arquitectura**
  - Son esquemas de organización general de un sistema.
  - Especifican una serie de subsistemas y sus responsabilidades.
  - Incluyen reglas para organizar las relaciones entre ellos.
- **Patrones de diseño**
  - Tienen un nivel menor, están más próximos a la implementación.
  - Su uso no se refleja en la estructura global del sistema.



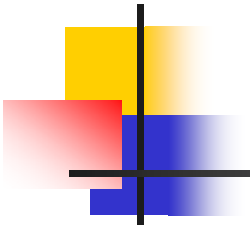
Patrones de Diseño

- Describen una estructura de diseño recurrente.
  - Están avalados por la experiencia.
  - Son soluciones a problemas concretos.
  - Son flexibles para adaptarse a necesidades específicas.
  - Su elevado número dificulta la catalogación.
- Hace abstracción de los sistemas concretos.
  - Identifica objetos y sus colaboraciones.
  - Proporciona una implementación.
  - No es posible reutilizar el código.

Catálogo (Gamma, et al, 1995)

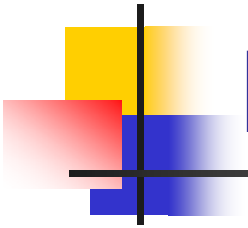
Creación (Creación de objetos)	**Estructural** (Composición de objetos)	**Comportamiento** (Interacción de objetos)
Abstract Factory	Adapter	Command
Factory Method	Bridge	Mediator
Singleton	Composite	Observer
Builder	Decorator	State
Prototype	Facade	Strategy
	Flyweight	Chain of Responsibility
	Proxy	Interpreter
		Template Method
		Memento
		Visitor

Estilos / Patrones de arquitectura



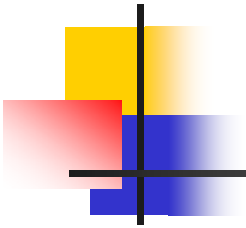
Introducción

- Un estilo arquitectónico es un conjunto de patrones para crear una o más arquitecturas en una forma consistente.
- Un estilo es una caracterización parcial de un sistema. No es la representación completa de una arquitectura, dado que es una plantilla para especificar la arquitectura de un sistema específico.
- Los estilos son usados como arquitecturas de referencia, frameworks, o idioms, y existen muchas formas para capturar y comunicar un estilo.



Niveles de representación

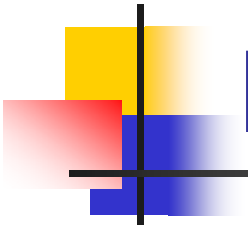
- **Estilos arquitectónicos**
  - familias de sistemas que siguen el mismo patrón estructural
- **Modelos y arquitecturas de referencia**
  - particularización de un estilo
- **Marcos de trabajo**
  - arquitectura especializada para un dominio de aplicación
- **Familias y líneas de productos**
  - arquitectura de una aplicación con diferentes configuraciones
- **Instancias**
  - arquitectura de una aplicación concreta



Estilos arquitectónicos

Un **estilo arquitectónico esta definido por**:

- Un conjunto de reglas y restricciones que definen:
  - Cuáles tipos de componentes, interfaces & conectores pueden ser usados en un sistema (Vocabulario/Metáforas). Posible inclusión de tipos de dominio-especifico
  - Cómo los componentes y conectores pueden ser combinados (estructura)
  - Cómo se comporta el sistema



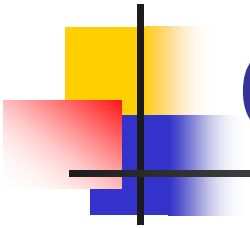
Estilos arquitectónicos

- Un conjunto de guías que soportan la aplicación del estilo (Cómo lograr ciertas propiedades del sistema)
- Un estilo arquitectónico define una familia de sistemas en términos de un patrón de organización estructural.

Estilos arquitectónicos

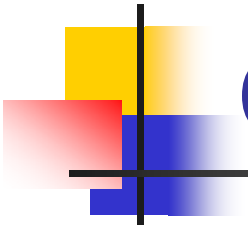
Un **estilo arquitectónico define**:

- Un vocabulario de tipos de componentes y conectores
- Un conjunto de restricciones sobre cómo pueden ellos ser combinados
- Uno o más modelos semánticos que especifican como todas las propiedades de un sistema pueden ser determinadas a partir de las propiedades de sus partes.



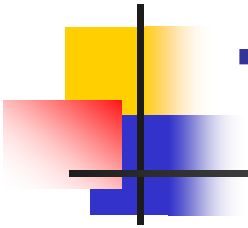
Conclusión

- Los estilos están abiertos a nuevos estilos que emerjan a medida que madura la tecnología.
- Una arquitectura puede usar varios estilos arquitectónicos.
- Los estilos arquitectónicos no son disjuntos.



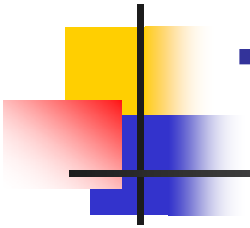
Componentes y Conectores

- Colección de módulos de software (Componentes) interactuando a través de un paradigma de comunicación bien definido (conectores)
- Los componentes son los bloques de construcción para describir una arquitectura.
- No existe aún una notación estándar



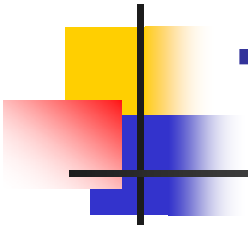
Tipos de Componentes

- **Computacional**: realiza el procesamiento en algún orden. E.g. función matemática, filtros.
- **Memoria**: mantiene una colección de datos persistentes. E.g. bases de datos, sistemas de archivos, tablas de símbolos.
- **Manejador**: contiene estado + operaciones asociadas. El estado es mantenido entre invocaciones de operaciones. E.g. Tipos de Datos Abstractos, Servidores.
- **Controlador**: gobierna la secuencia de tiempo de otros eventos. E.g. módulo de control de alto nivel, scheduler.



Tipos de Conectores

- **Procedure call**: simple thread de control entre el invocado (called) y el invocador (callee). E.g. tradicional y RPC.
- **Data flow**: Interacción de procesos a traves de flujos de datos. E.g. pipes.
- **Implicit invocation**: el proceso se inicia hasta que un evento ocurra. E.g. listas de correo.

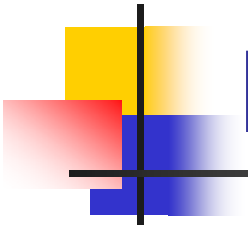


Tipos de Conectores

- **Message passing**: la interacción se realiza a traves de transferencia explicita o de datos discretos. E.g. TCP/IP.
- **Shared data**: el acceso a datos es concurrente, con algún esquema de bloqueo para prevenir los conflictos. E.g. Pizarra, bases de datos compartidas.
- **Instantiation**: espacio de localización para un estado requerido por otro componente. E.g. Tipos Abstractos de Datos.

Clasificación de los estilos

- **Clasificación** de los sistemas de software en **grandes familias** cuyos integrantes comparten un **patrón estructural** común.
  - Ejemplos: Tubos y Filtros, Organizados en Capas, Cliente/Servidor, etc.



Elementos para clasificar

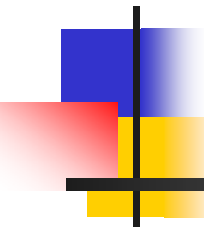
- Componentes
  - unidades computacionales y de datos
- Conectores
  - mecanismos de interacción entre componentes
- Patrones y restricciones de interconexión
  - invariantes del estilo
- Mecanismos de control
  - coordinación entre componentes
- Propiedades
  - ventajas e inconvenientes

Estilos arquitectónicos

Sistemas de flujo de datos	*Tubos y Filtros*
	Procesamiento por lotes
Sistemas basados en llamada y retorno	*Programa principal y subrutinas*
	Orientados a objetos
	Organizados en capas
Sistemas de componentes independientes	*Comunicación entre procesos*
	Cliente/servidor
	Basados en eventos

Estilos arquitectónicos

Sistemas centrados en los datos	*Repositorios* *Pizarras*
Máquinas virtuales	*Intérpretes* *Basados en reglas*
Sistemas heterogéneos	*Localmente heterogéneos* *Jerárquicamente heterogéneos* *Simultáneamente heterogéneos*



Sistema de flujo de Datos

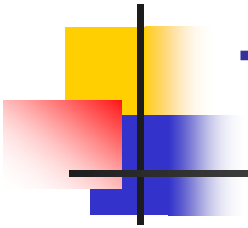
- Tubos y Filtros
- Procesamiento por Lotes

Sistema de flujo de Datos

Tiene como objetivo el aseguramiento de cualidades de reutilización y modificabilidad

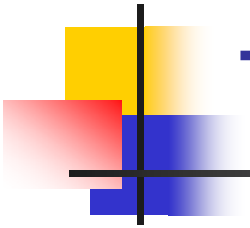
Este estilo es caracterizado por ver el sistema como una serie de transformaciones sobre elementos sucesivos de datos de entrada

Los datos entran al sistema y luego navegan a través de componentes al mismo tiempo, antes de ser asignados a su destino final.



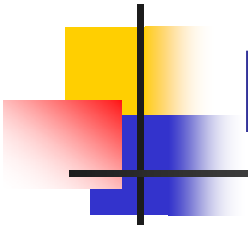
Tubos y Filtros

- Cada **componente** tiene un conjunto de entradas y salidas
- Un componente lee flujos de datos (streams) desde sus entradas y genera flujos de datos.
- Usan pequeña información contextual y no retienen información de su estado entre invocaciones
- Los pipes son stateless
- Las restricciones indican la forma en la cual los pipes y los filtros pueden ser encadenados



Tubos y Filtros

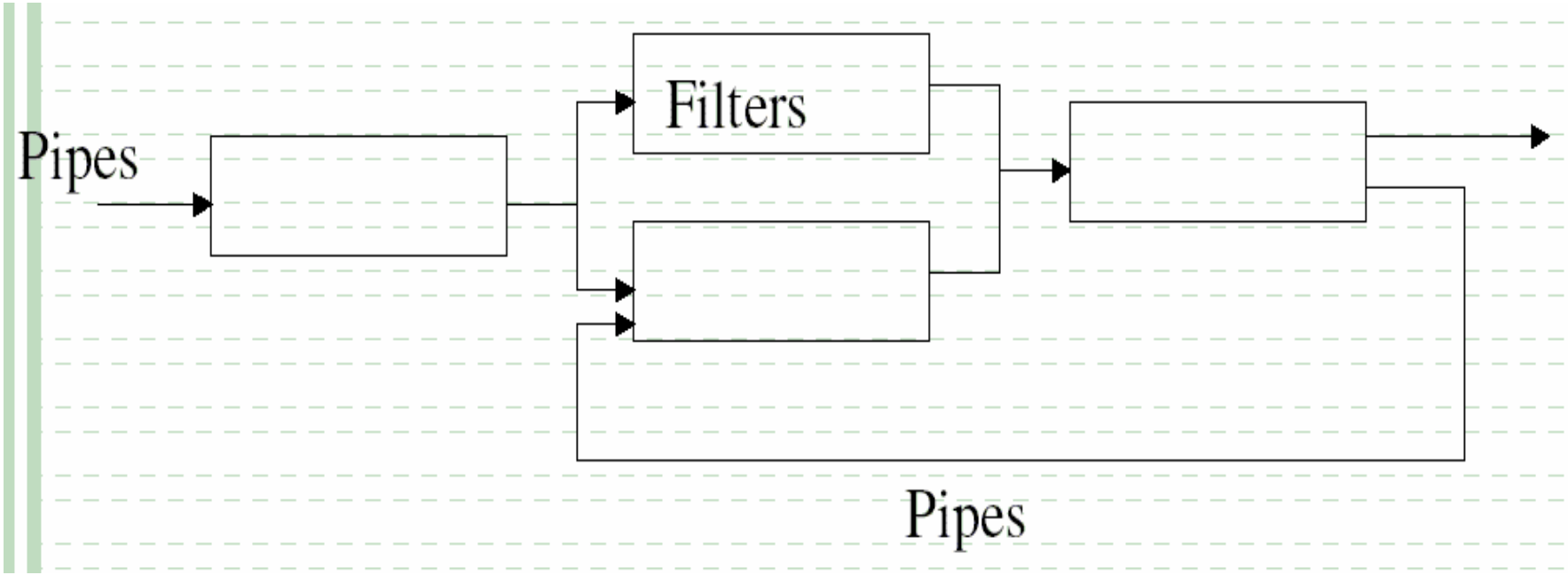
- Especializaciones de este estilo
  - **Pipelines**, restringen las tipologías a secuencias lineales de filtros
  - **Bounded pipes**, restringen el tamaño de datos que puede residir en un pipe.
  - **Typed pipes**, requiere que los datos pasados entre dos filtros tengan un tipo bien definido
- Ej: Compiladores (analizador léxico, parser, analizador semántico)

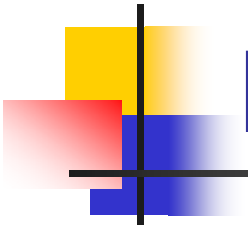


Elementos Tubo-Filtro

- Filtro:
  - Deben ser entidades independientes
  - Puede NO compartir estado con otros filtros
  - Los filtros no conocen la identidad de sus vecinos
  - No preserva su estado entre invocaciones
- Pipe:
  - Mueve datos de un filtro de salida a un filtro de entrada
  - La fuente de un pipe solo puede ser conectada a la salida de un filtro o a su entrada (restricción)

Ej: Tubos y Filtros

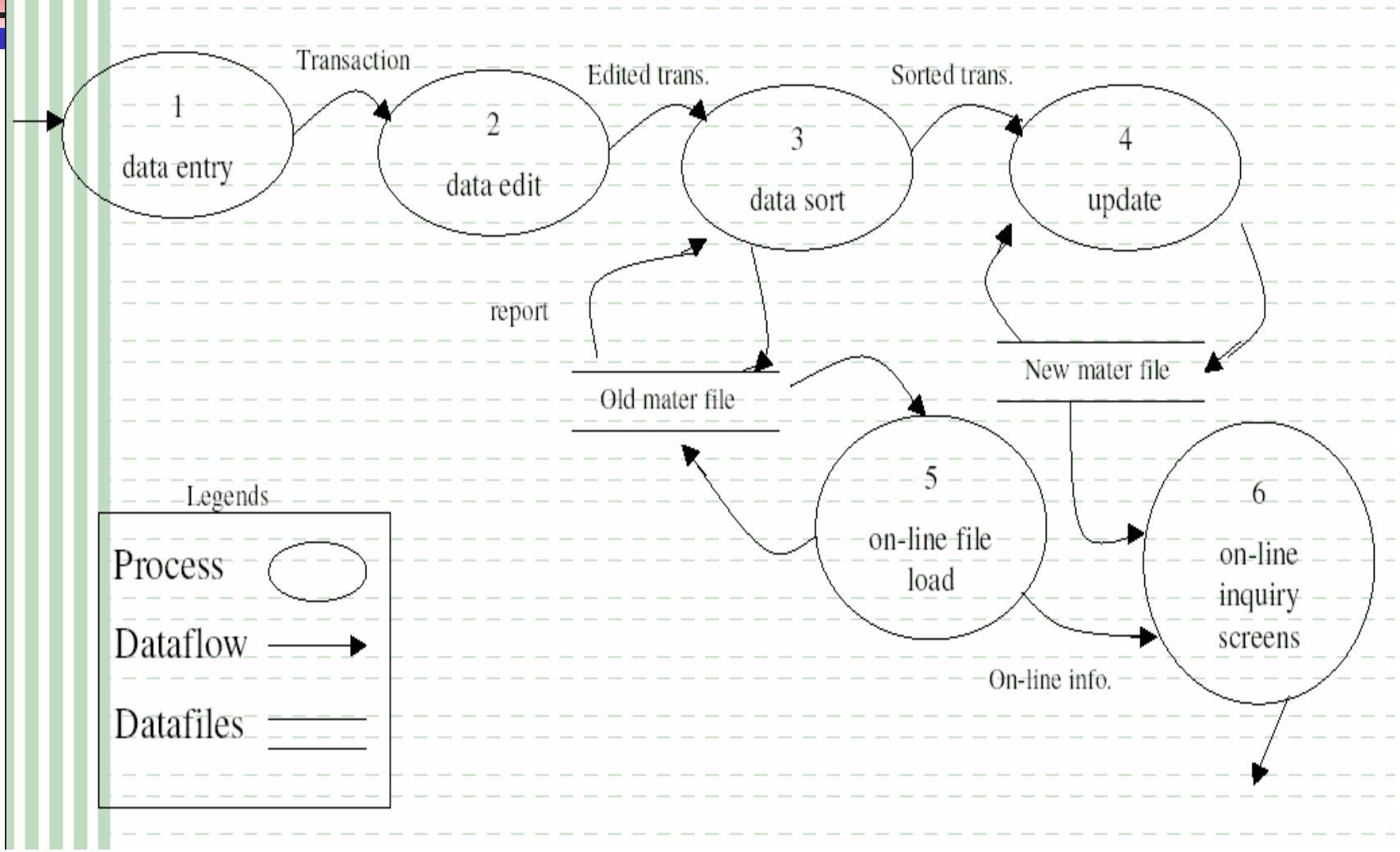




Procesamiento por lotes

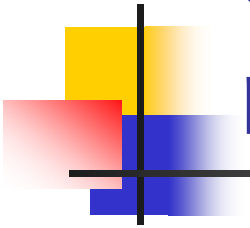
- Los pasos de procesamiento o componentes son programas independientes y se supone que cada paso se ejecuta completamente antes de seguir al siguiente.
- Los pasos son programas independientes, y corren en una secuencia predefinida.
- Aplicaciones típicas:
  - Procesamiento de datos clásica
  - Desarrollo de programas

Ej: Procesamiento por lotes

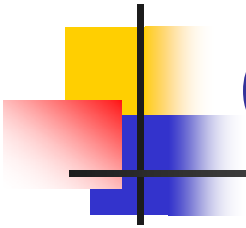


Sistemas basados en llamada y retorno

Sistemas basados en llamada y retorno

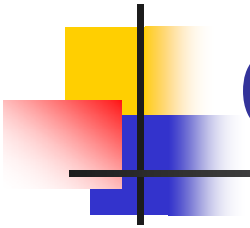


- Persigue obtener cualidades de
  - Escalabilidad y modificabilidad
- Este estilo a dominado en grandes sistemas de software
- Tiene tres variaciones:
  - Capas
  - O.O
  - Programa principal y subrutinas



Orientado a Objetos

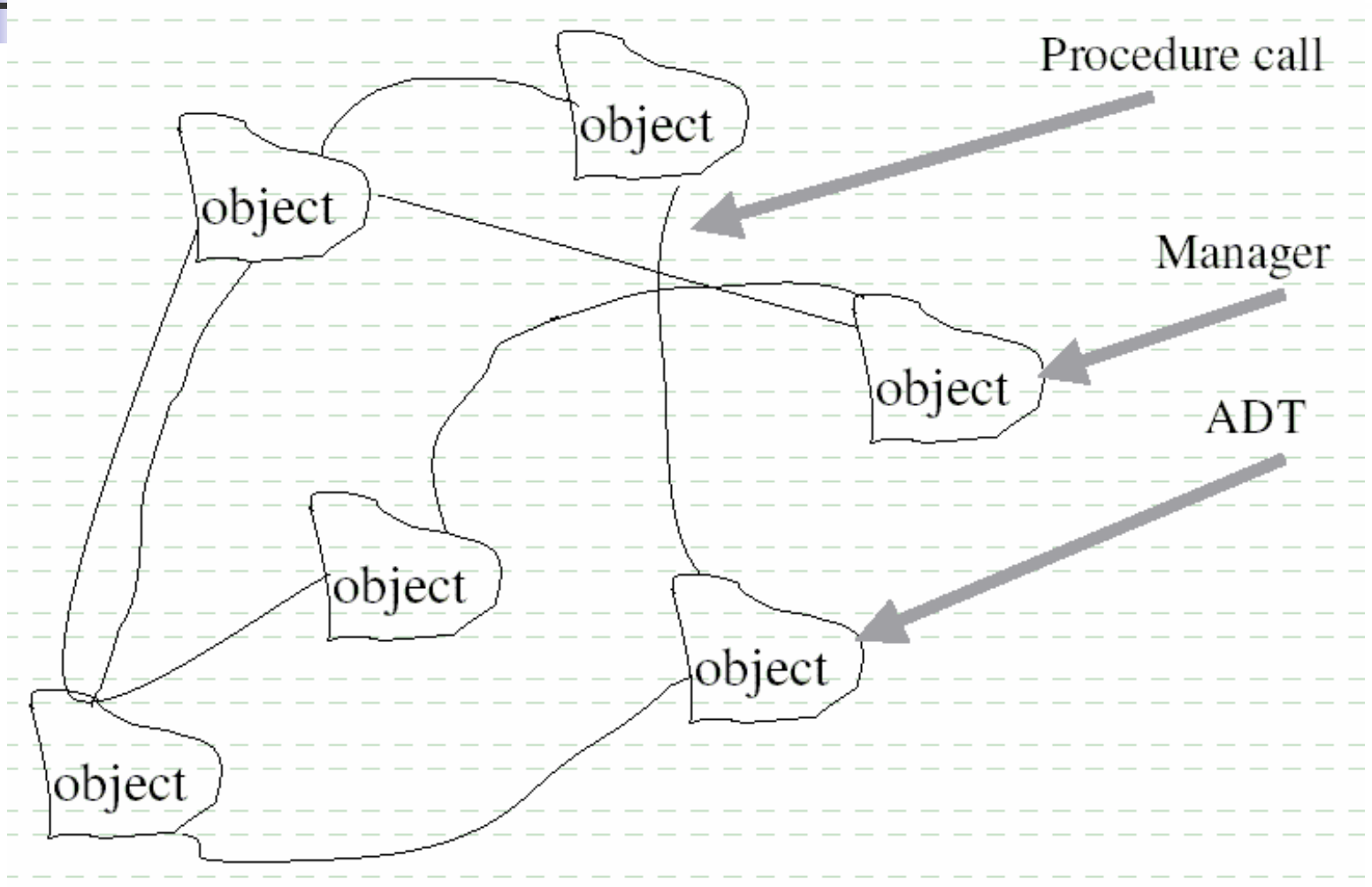
- Basado en abstracción de datos y organización O.O
- Los componentes son Objetos o TADs
- Los objetos interactuan a través de invocación de funciones y procedimientos
- Algunos sistemas permiten ejecución concurrente de tareas; otras permiten objetos con múltiples interfaces



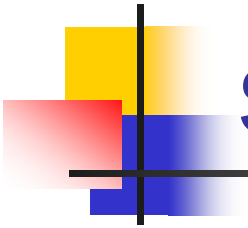
Orientado a Objetos

- Es posible cambiar la implementación de objetos sin afectar a los clientes
- Los diseñadores pueden descomponer el problema en colecciones de agentes interactuando
- Persigue obtener cualidades de
  - Modificabilidad

Orientado a Objetos

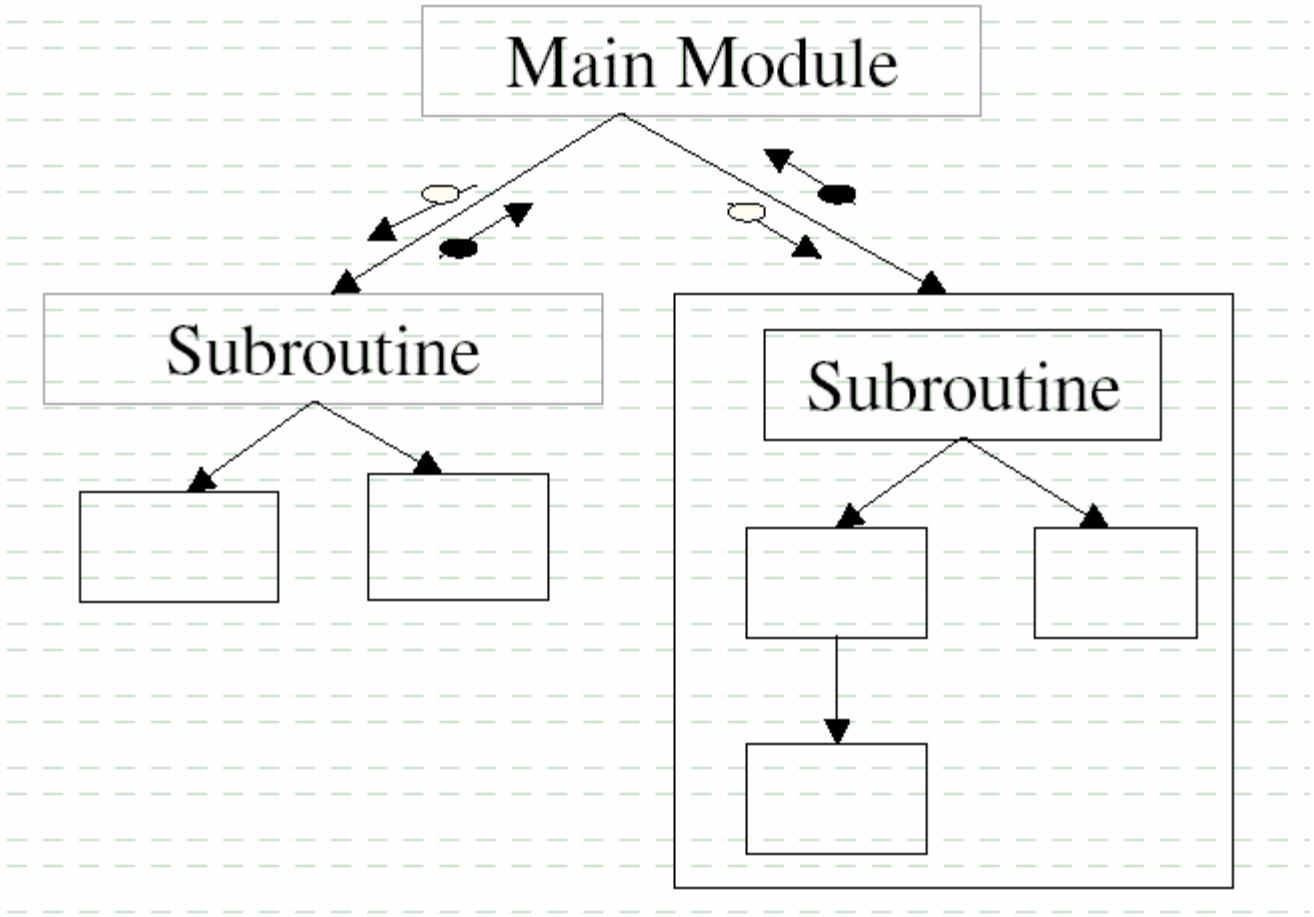


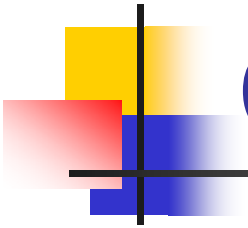
Programas principales y subrutinas



- Descomposición jerárquica
  - Basado en la definición y uso de relaciones
- Simple hilo de control
  - Soportado por los lenguajes de programación
- Implícitamente usa una estructura de subsistemas
- Razonamiento jerárquico
  - Modificación en cascada (cambios en una subrutina implica cambios en las subrutinas invocadas)
- Meta: Incrementar el desempeño distribuyendo el trabajo en múltiples procesadores

Programas principales y subrutinas





Capas

- Organización Jerárquica
- Cada capa provee servicios a sus capas vecinas
- Los conectores son definidos por los protocolos que determinan como interactúan las capas
- Restricciones topológicas incluyen limitación de interacciones a capas adyacentes
- Cada capa sucesiva es construida basada en su antecesor
- Ej: Modelo OSI, Sistema X-Windows , etc.

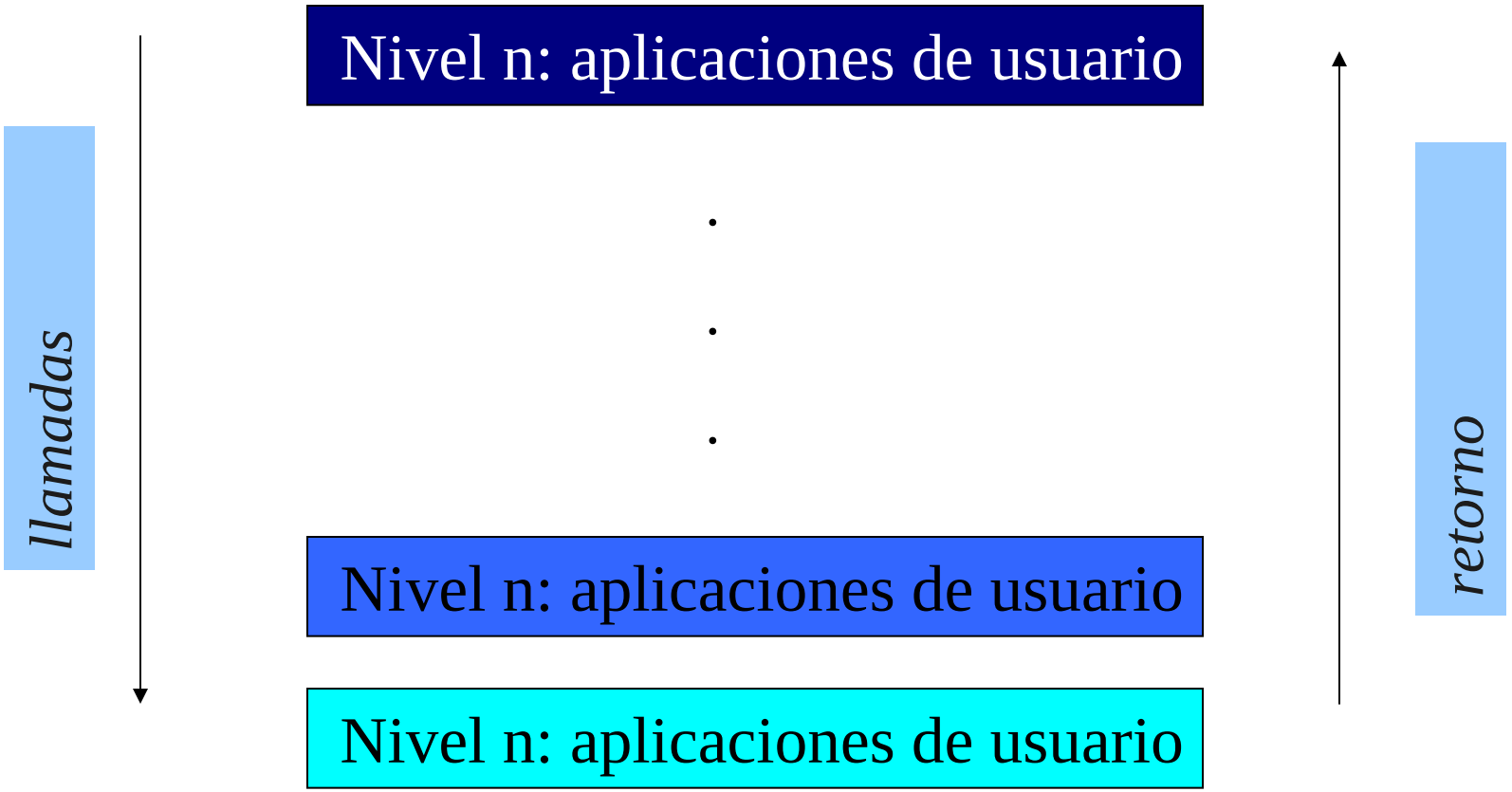
Organización en capas

- Aplicabilidad:
  - Sistemas grandes que están caracterizados por una mezcla de elementos de alto y bajo nivel, donde los elementos de alto nivel dependen de los de bajo nivel.
- Componentes:
  - son las capas o niveles que pueden estar implementadas internamente por objetos o procedimientos.
  - Cada nivel tiene asociada una funcionalidad:
    - Niveles bajos: Funciones simples, ligadas al hardware o al entorno.
    - Niveles altos: Funciones más abstractas.

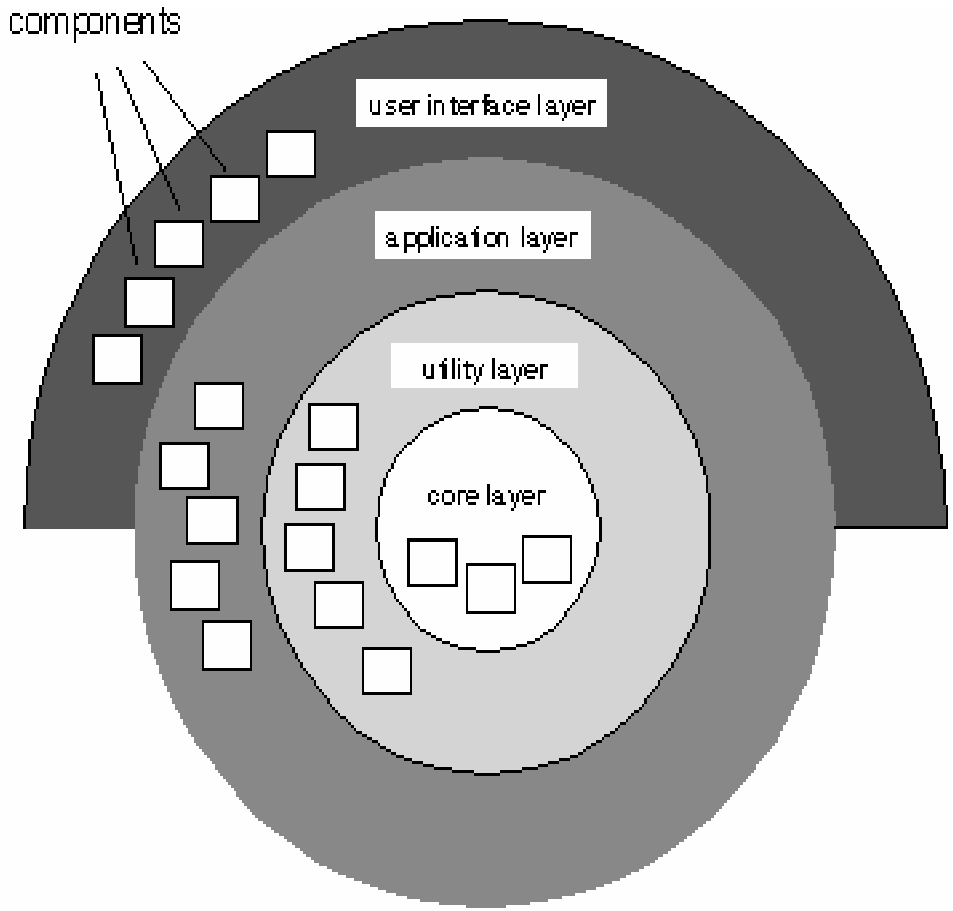
Organización en capas

- Mecanismos de interacción entre componentes:
  - Llamadas a procedimientos.
  - Llamadas a métodos.
- Invariantes/Restricciones:
  - Sólo llamadas de niveles superiores a inferiores.
  - (Variante) Sólo llamadas entre niveles adyacentes.
- Aplicación:
  - Torres de protocolos de comunicación,
  - Sistemas operativos,
  - Compiladores.

Organización en capas

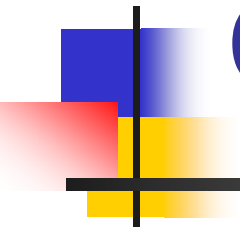


Organización en capas

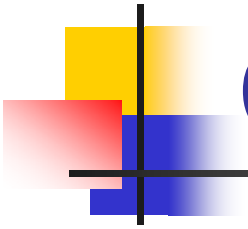


Organización en capas

- Propiedades:
  - Facilita la migración. El acoplamiento con el entorno está localizado en las capas inferiores. Estas son las únicas a re-implementar en caso de transporte a un entorno diferente.
  - Cada nivel implementa unas interfaces claras y lógicas, lo que facilita la sustitución de una implementación por otra.
  - Permite trabajar en varios niveles de abstracción. Para implementar los niveles superiores no se necesita conocer el entorno subyacente, solo las interfaces que proporcionan los niveles inferiores.



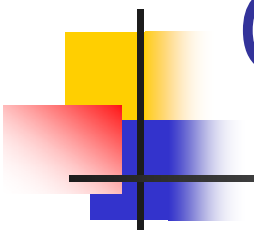
Componentes Independientes



Componentes Independientes

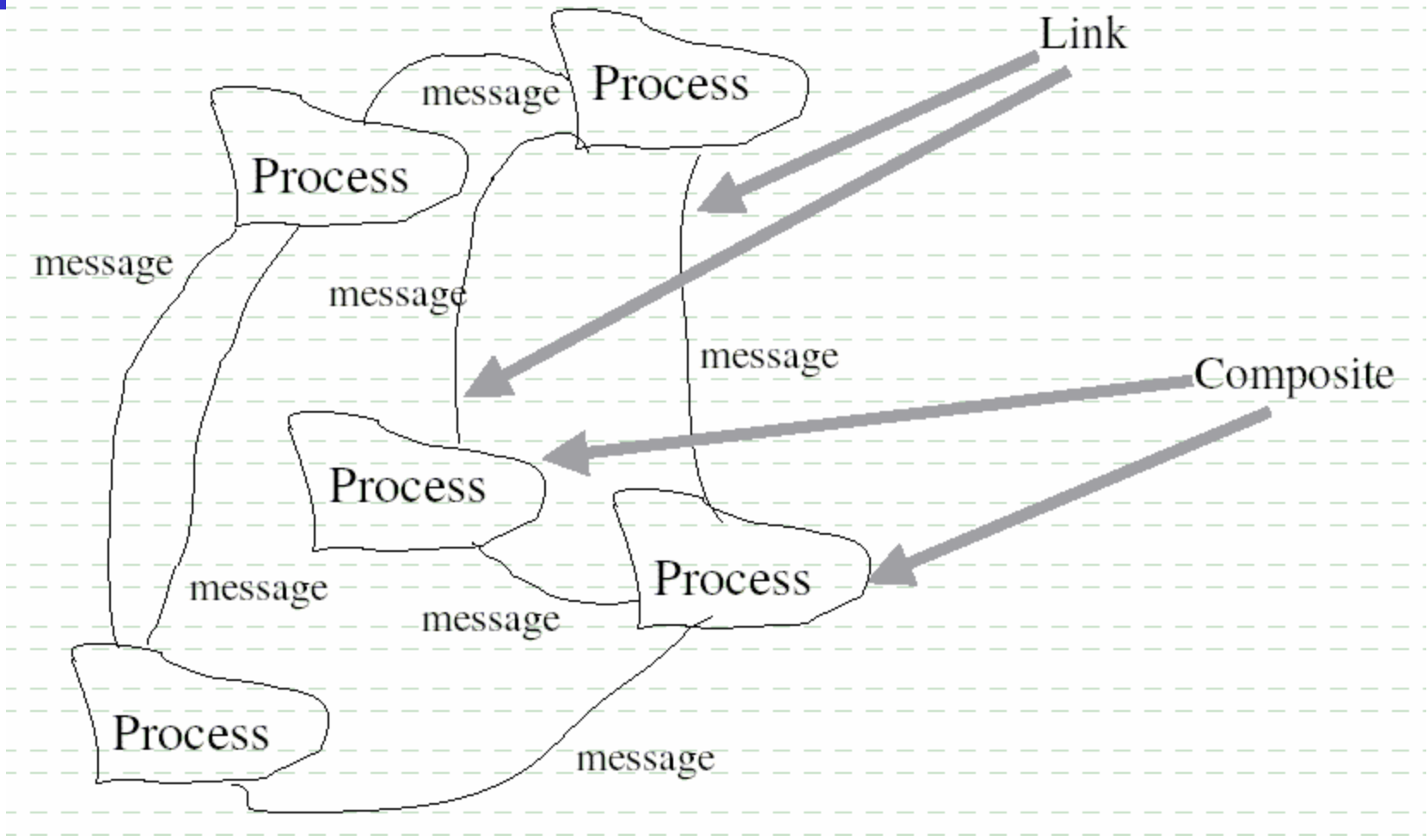
- Consiste de un número de **objetos** o **procesos** independientes que se **comunican** a través de **mensajes.**
- La modificabilidad se logra por el **desacoplamiento** en varias porciones de procesamiento.
- Solo se envían mensajes entre los objetos, sin tener control directamente.

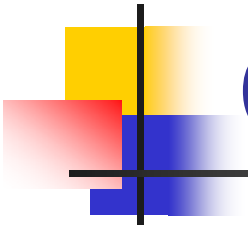
Comunicación entre procesos



- Corresponde a los sistemas de multiprocesamiento clásicos.
- Su objetivo es alcanzar la **escalabilidad.**
- Cliente-Servidor es un subtipo bien conocido.
- Los componentes:
  - Procesos independientes
    - Implementados como tareas separadas
- Conectores:
  - Paso de mensajes
    - Punto a punto
    - Asincrónicos y sincrónicos
    - RPC y otros protocolos

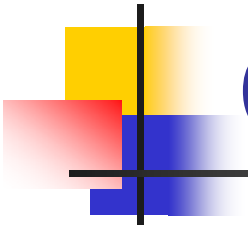
Comunicación entre procesos





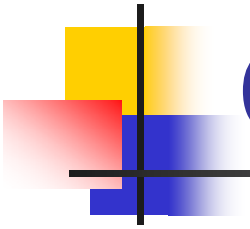
Cliente/Servidor

- Modelo de Sistemas Distribuido, el cual muestra como los datos y procesamiento están distribuidos entre un rango de componentes.
- Conjunto de servidores “stand-alone”, los cuales proporcionan servicios específicos como impresión, manejo de datos, etc.
- Conjunto de clientes que llaman a estos servicios.
- Redes que permiten que los clientes acceden a los servidores



Cliente/Servidor

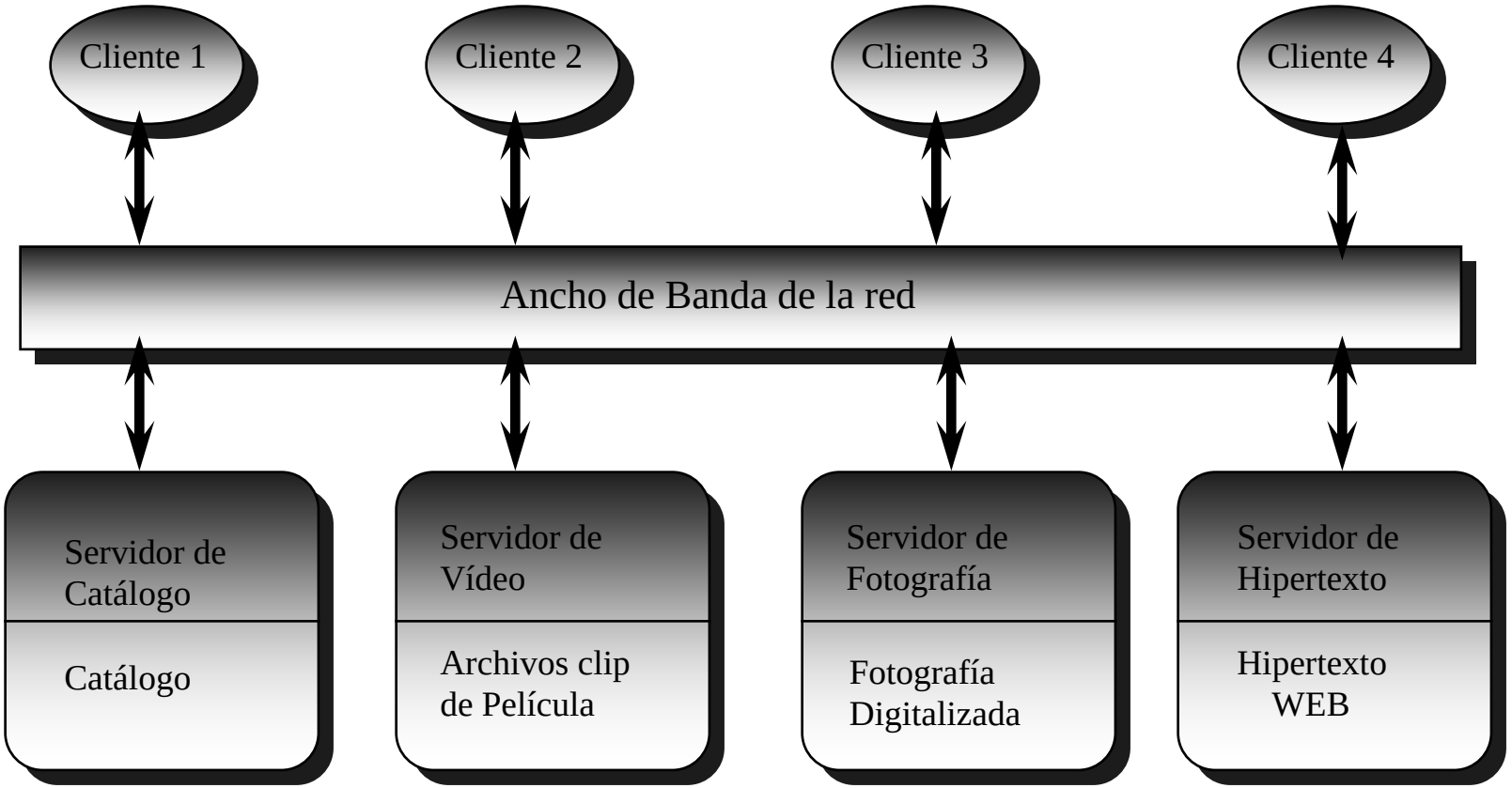
- Ventajas
  - La Distribución de datos es directa.
  - Permite el uso efectivo de sistemas de red.
  - Puede requerir hardware barato.
  - Es fácil añadir nuevos servidores o actualizar los existentes.

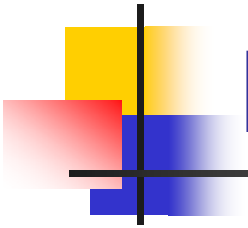


Cliente/Servidor

- Desventajas
  - El modelo no comparte datos con los diferentes subsistemas empleados en la organización.
  - El intercambio de datos puede ser ineficiente.
  - Administración redundante en cada servidor.
  - No existen registros centrales de nombres y servicios - esto hace difícil encontrar los servidores y servicios disponibles.

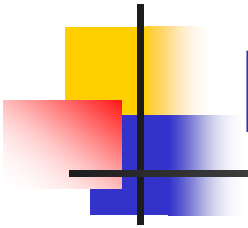
Ejemplo


Cliente 1
Cliente 2
Cliente 3
Cliente 4
Ancho de Banda de la red
Servidor de Catálogo
Catálogo
Servidor de Vídeo
Archivos clip de Película
Servidor de Fotografía
Fotografía Digitalizada
Servidor de Hipertexto
Hipertexto WEB



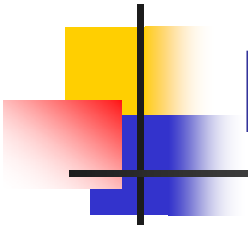
Basados en Eventos

- Los componentes interactúan por medio de invocaciones explícitas de procedimientos o funciones
- Los componentes exponen datos que son compartidos con su entorno
- Los componentes pueden registrarse a una clase de datos de interés
- Existe un manejador de mensajes que coordina la comunicación entre componentes, invocando al componente cuando un mensaje que llega es para ese determinado componente.



Basados en Eventos

- Existen algunas técnicas de integración alternativas, tales como:
  - Invocación implícita
  - Integración reactiva
  - Integración selectiva
  - Cuando un evento llega, el sistema invoca todos los componentes que han sido registrados para ese evento
  - Los componentes en un invocación implícita pueden ser módulos cuyas interfaces proveen tanto una colección de procedimientos como un conjunto de eventos



Basados en Eventos

- **Componentes**:
  - Objetos y procesos
    - Las Interfaces definen un conjunto de llamadas entrantes de procedimientos.
    - Las interfaces definen un conjunto de eventos salientes
- **Conexiones**:
  - Encadenamiento de eventos-procedimientos
    - Los procedimientos son registrados con eventos
    - Los componentes se comunican por eventos definidos apropiadamente
    - Cuando un evento es recibido, el procedimiento asociado es invocado
    - El orden de invocación es no-deterministico
    - En algunos casos los conectores son evento-evento

Basados en Eventos – propiedades arquitectónicas

- Ventajas
  - Simplicidad
  - Evolución: se pueden reemplazar componentes suscriptores
  - Modularidad: una sola modalidad para eventos diversos
  - Puede mejorar eficiencia, eliminando la necesidad de polling por ocurrencia de eventos

Basados en Eventos – propiedades arquitectónicas

- Ventajas:
  - Provee gran soporte para la reutilización
  - Cualquier componente puede ser introducido en el sistema, basta registrarlo a los eventos del sistema
  - Invocaciones implícitas facilita la evolución del sistema, esto es, reemplazar componentes sin afectar las interfaces de otros componentes

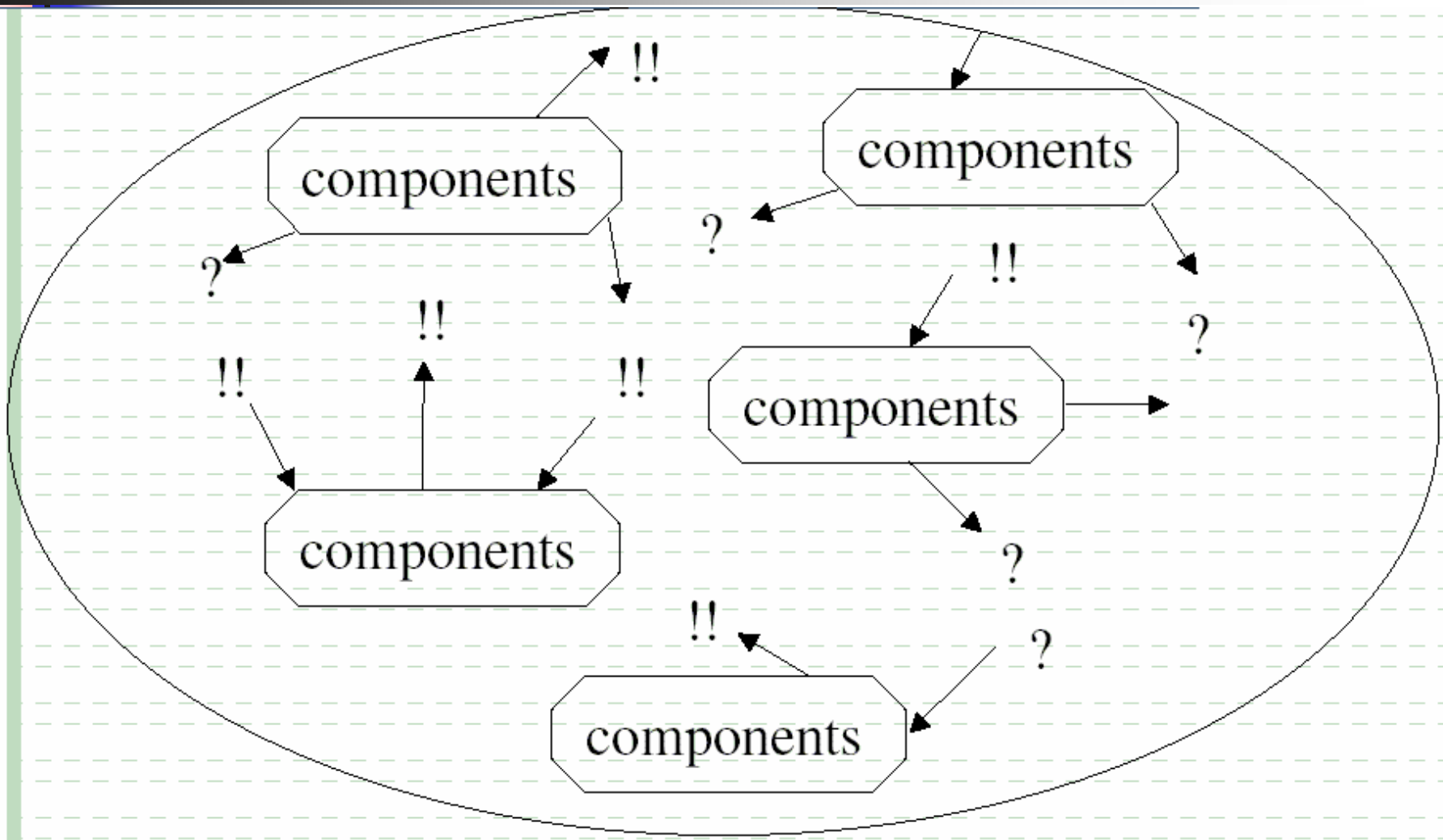
Basados en Eventos – propiedades arquitectónicas

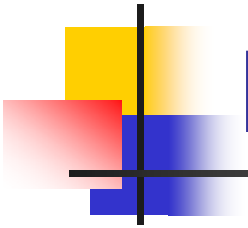
- Desventajas
  - Posibilidad de desborde
  - Potencial imprevisión de escalabilidad
  - Pobre comprensibilidad: Puede ser difícil prever qué pasará en respuesta a una acción
  - No hay garantía del lado del publisher que el suscriptor responderá al evento

Basados en Eventos – propiedades arquitectónicas

- Desventajas
  - No hay mucho soporte de recuperación en caso de falla parcial
  - La mayor desventaja es el efecto de los componentes sobre el desempeño del sistema
  - Cuando un componente genera un evento, no hay garantía de que será atendido por otro componente

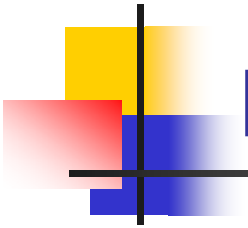
Basados en Eventos





Ejemplos

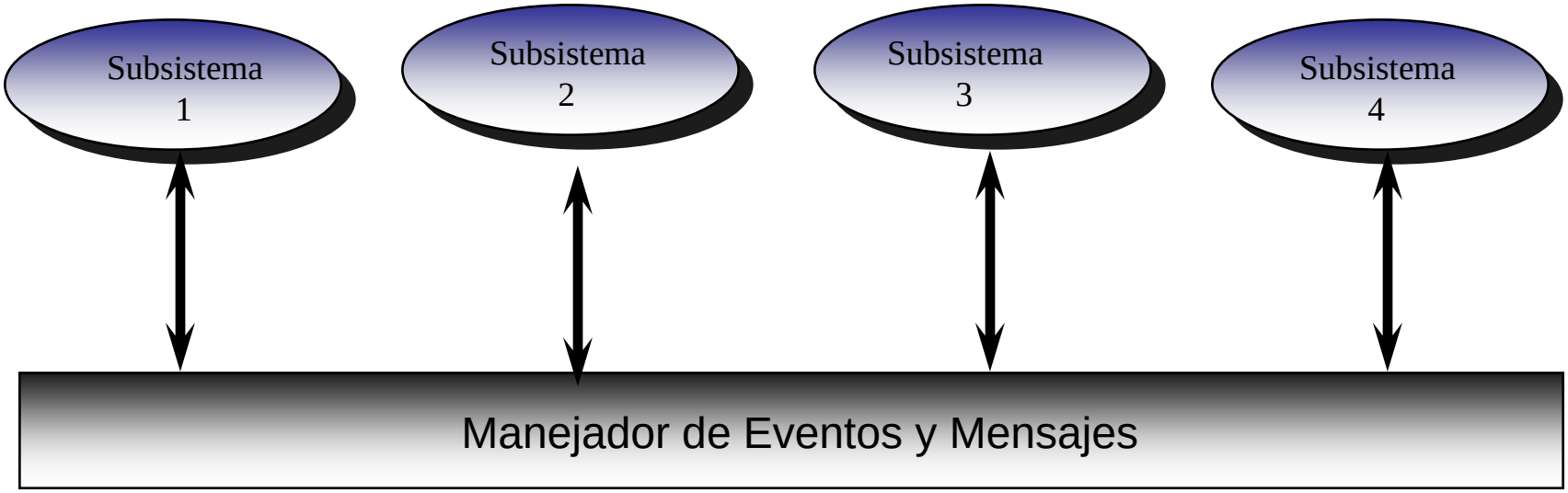
- Dos de los principales modelos manejadores de eventos
  - **Modelo de Transmisión** (Broadcast). Un evento es transmitido a todos los subsistemas. Cualquier subsistema puede manejar el evento
  - **Modelos manejadores de interrupciones**. Utilizados en sistemas en tiempo real donde una interrupción es detectada por un manejador de interrupciones y es pasada a otros componentes para ser procesada



Modelo de Transmisión (Broadcast)

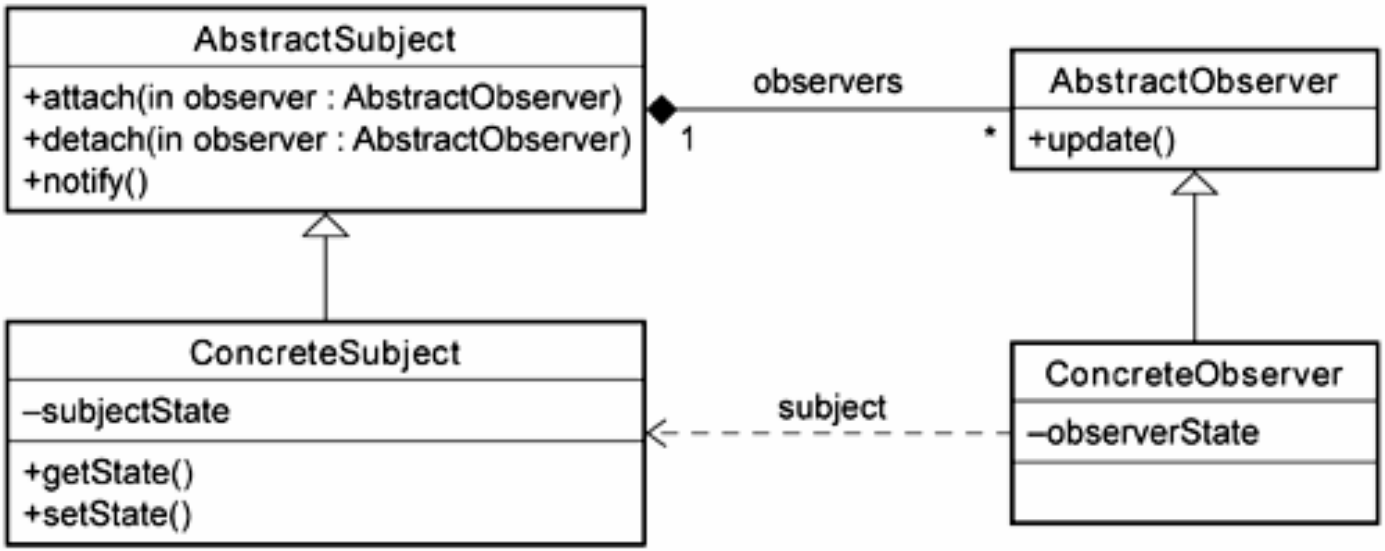
- Es efectivo en la integración de subsistemas en diversos computadores en una red
- Los subsistemas registran la petición de eventos específicos. Cuando esto ocurre, el control es transferido a los subsistemas que pueden manejar el evento
- Las políticas de control no están contenido dentro del evento o del manejador de eventos. Los subsistemas deciden cuales eventos son de su interés
- No obstante, los subsistemas no saben cuando un evento será manejado

Transmisión Selectiva

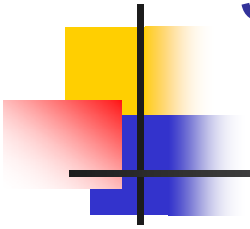


Ejemplo
Arquitectura basada en eventos

- Modelo de *push* a veces se vincula con patrón Observador (Observer pattern)

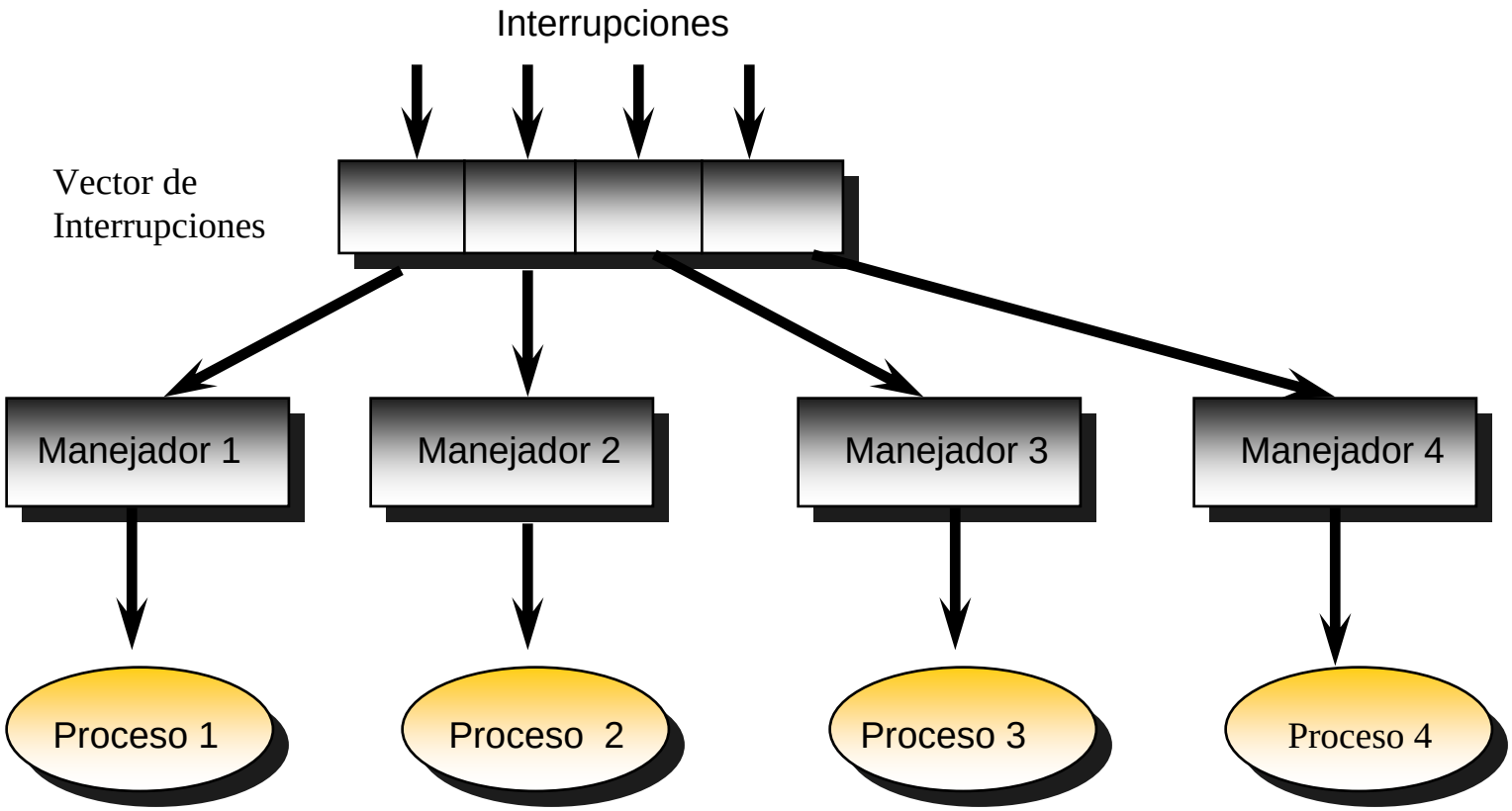


Sistemas Manejados por Interrupciones

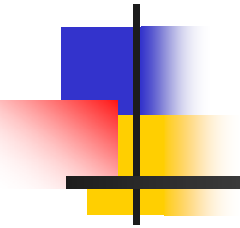


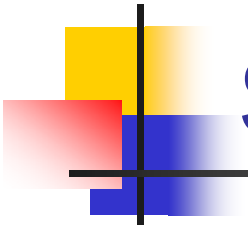
- Utilizado en Sistemas de tiempo real donde una respuesta rápida es esencial
- Hay tipos de interrupciones con un manejador definido para cada tipo
- Cada tipo está asociado con una localidad de memoria y un switch de hardware ocasiona transferencias al manejador
- Una respuesta rápida pero compleja de programar y difícil de validar

Control de Manejo de Interrupciones



Arquitecturas centradas en los datos

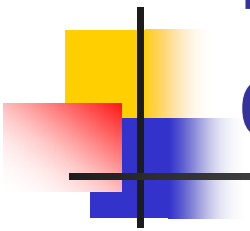




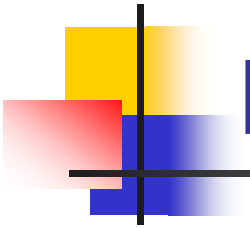
Sistemas centrados en los datos

- Enfatiza en la integración de datos
- Es apropiada para sistemas que se fundamentan en el acceso y actualización de datos en estructuras de almacenamiento.
  - Sub-estilos
    - Repositorios
    - Bases de datos
    - Hipertextos
    - pizarras

Repositorios (Sistemas centrados en los datos)

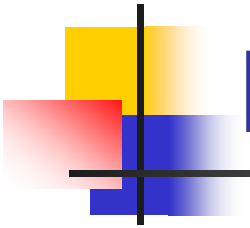


- En un estilo repositorio existen dos clases de componentes:
  - Una estructura de datos central que representa el estado actual
  - Una colección de componentes independientes que operan sobre los datos centrales
  - Las interacciones entre el repositorio y los componentes externos puede variar significativamente entre sistemas



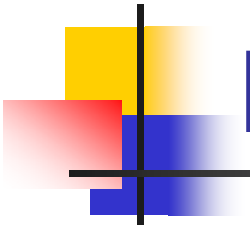
Repositorios

- Bases de datos clásicas
  - Repositorio central tiene esquemas diseñados específicamente para la aplicación
  - Operadores independientes
    - Operaciones sobre la BD



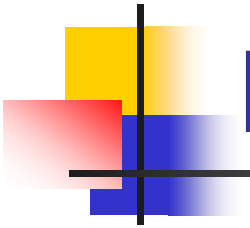
Repositorio

- Ventajas
  - Es una forma eficiente de compartir grandes cantidades de datos.
  - Los Subsistemas no necesitan proporcionar un manejo centralizado de como los datos son producidos. Por ejemplo: respaldo, seguridad, etc.



Repositorio

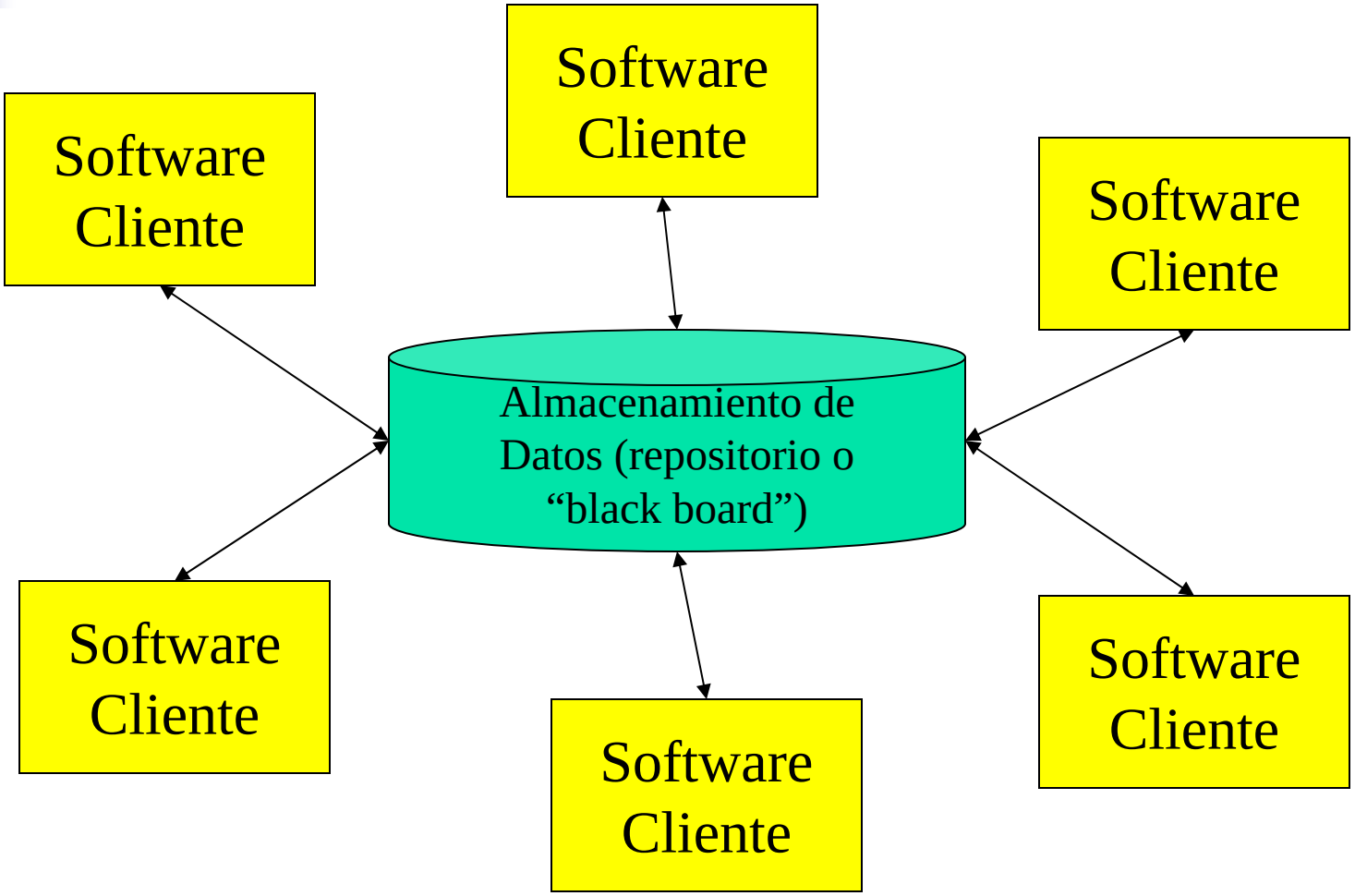
- Desventajas
  - Los sub-sistemas deben coincidir en modelo de datos del repositorio, lo cual es inevitablemente un compromiso
  - La evolución de los datos es difícil y costosa.
  - No existen políticas para un manejo específico.
  - Se dificulta una distribución eficiente.



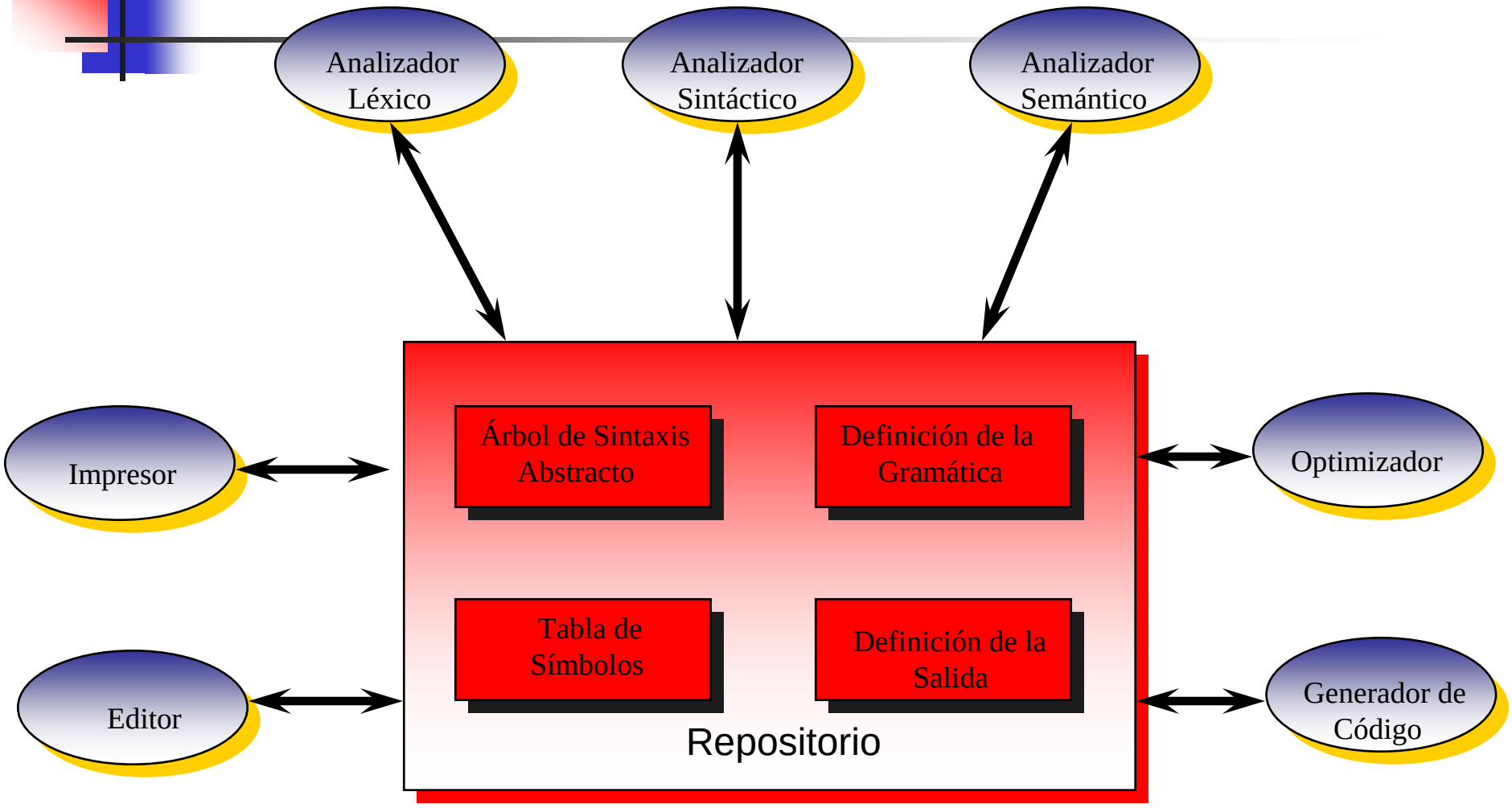
Pizarras (Blackboard)

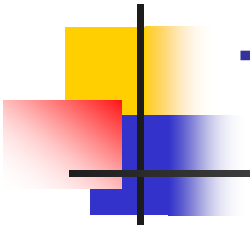
- La pizarra envía notificación a los suscriptores cuando los datos cambian
- En ocasiones esto se refiere a repositorios activos
- Los datos almacenados son independientes de los clientes, este estilo es escalable; nuevos clientes pueden ser adicionados fácilmente

Repositorios/blackboard



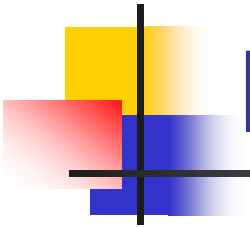
Sistema de Procesamiento de un Lenguaje





Taller

- Revisar los ejemplos de la página:
  - http://www.cs.cmu.edu/People/ModProb/index.html



Problema No. 1

- Se esta construyendo un sistema de reconocimiento de voz; asuma que el sistema tiene que ejecutar operaciones de segmentación a fonemas, creación de silabas, creación de palabras y posee una tabla de vocabulario; asuma que estas tareas cooperan sobre el problema de reconocimiento y no existe un algoritmo simple y ordenado para ejecutar la tarea; también, el sistema debe ser fácil de extender con nuevos algoritmos.
  - ¿Cuál es la AS más apropiada para este problema?

Problema No. 2

- ***Se desea construir un controlador de televisión, el cual responde a señales enviadas desde una unidad de control remoto.***
  - ¿Cuál es la AS más apropiada para este problema?