Arquitectura de Sistemas de Software

Departamento de Ciencia de la Computación
Escuela de Ingeniería – PUC
Hans Findel {hjfindel@uc.cl}

Elementos básicos
Patrones y Estilos

Elementos fundamentales

- “Basics”
  - Componentes
  - Conectores
  - Datos
  - Configuraciones

Estilos arquitectónicos

- Patrones arquitectónicos

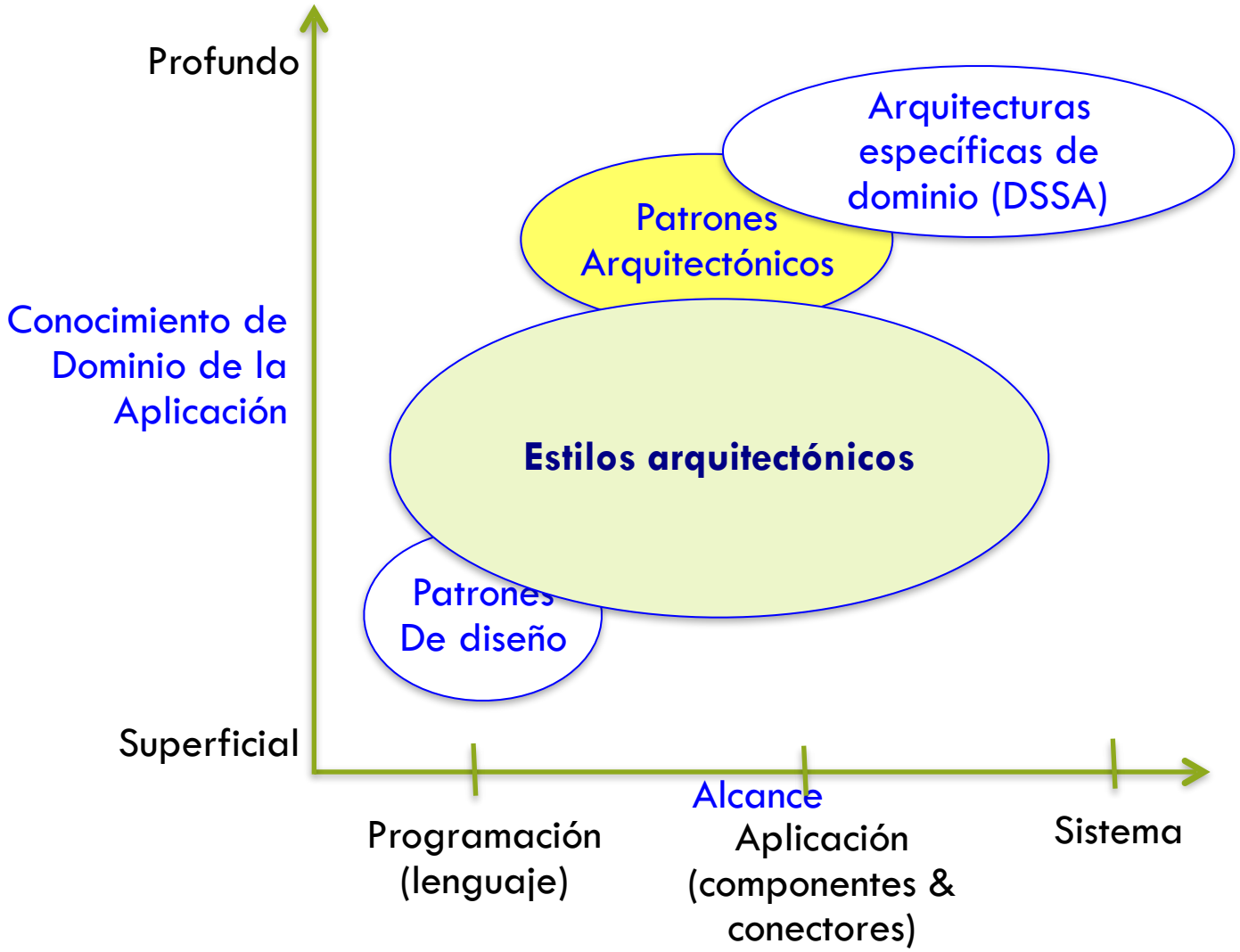
Componente

- Entidad arquitectónica que encapsula un subconjunto de la funcionalidad y/o estado (data) del sistema
- Restringe el acceso a ese subconjunto a través de una interfaz definida explícitamente
- Tiene dependencias explícitamente definidas sobre su contexto de ejecución
  - Interfaz con otros componentes, recursos, software de sistema (ambientes de ejecución, middleware, dispositivos, sistema operativo, etc), o hardware

Conector y configuración

- Conector
  - "Elemento" arquitectónico encargado de efectuar y regular las interacciones entre componentes
    - Llamadas a procedimientos
    - Acceso a memoria compartida
    - Paso de mensajes
    - Streaming
    - Acceso a componentes distribuidos
    - Wrapper/Adaptador
- Configuración Arquitectónica o Topología
  - Conjunto de asociaciones específicas entre componentes y conectores de una arquitectura

Patrones, Estilos y DSSAs



Estilo arquitectónico

- Colección nombrada (con nombre) de decisiones de diseño arquitectónico que:
  - Se aplican a un contexto de desarrollo dado
  - Restringen las decisiones de diseño que son específicas a un sistema particular dentro de ese contexto
  - Obtienen las ventajas de cada sistema resultante

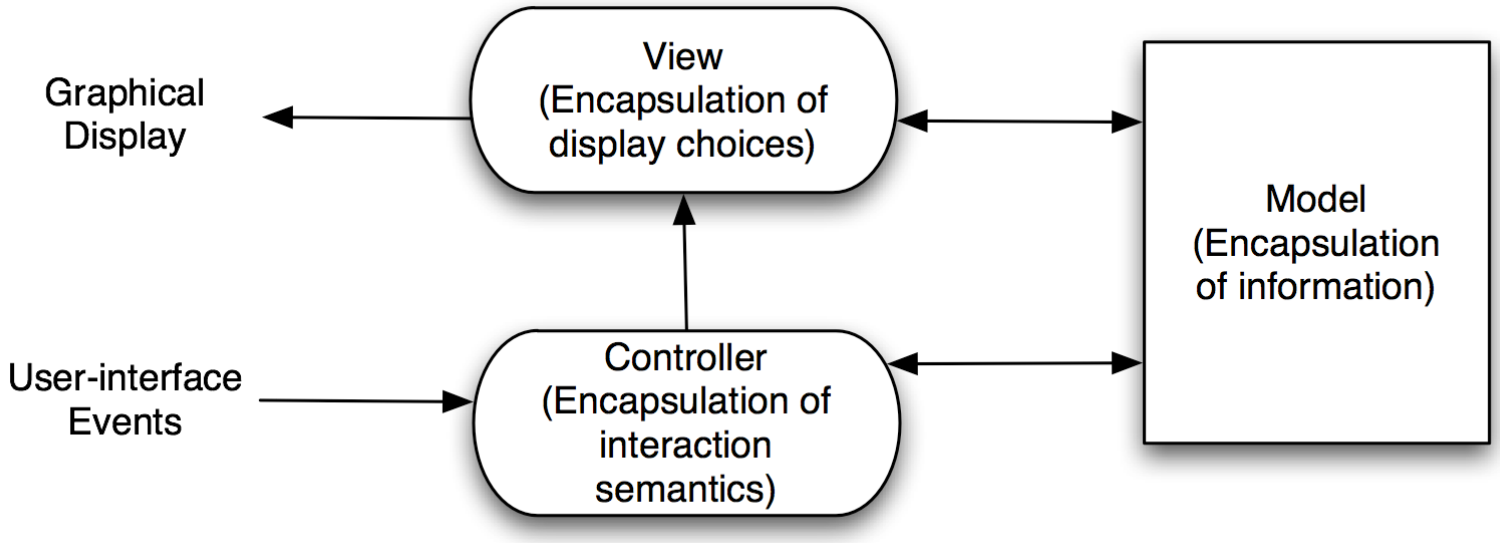
Patrón arquitectónico

- Colección nombrada de decisiones de diseño arquitectónico aplicadas a un problema de diseño recurrente que se parametriza según los contextos de desarrollo en que aparece el problema
- Diferencias con Estilo arquitectónico:
  - **Alcance**: contexto de desarrollo (estilo-estratégico), problema de diseño (patrón-táctico)
  - **Abstracción**: requiere interpretación humana (estilo), fragmentos arquitectónicos parametrizados (patrón)
  - **Relación**: Un patrón puede aplicarse a sistemas que siguen varios estilos, un sistema que sigue un estilo puede usar varios patrones

Ejemplos de Patrones

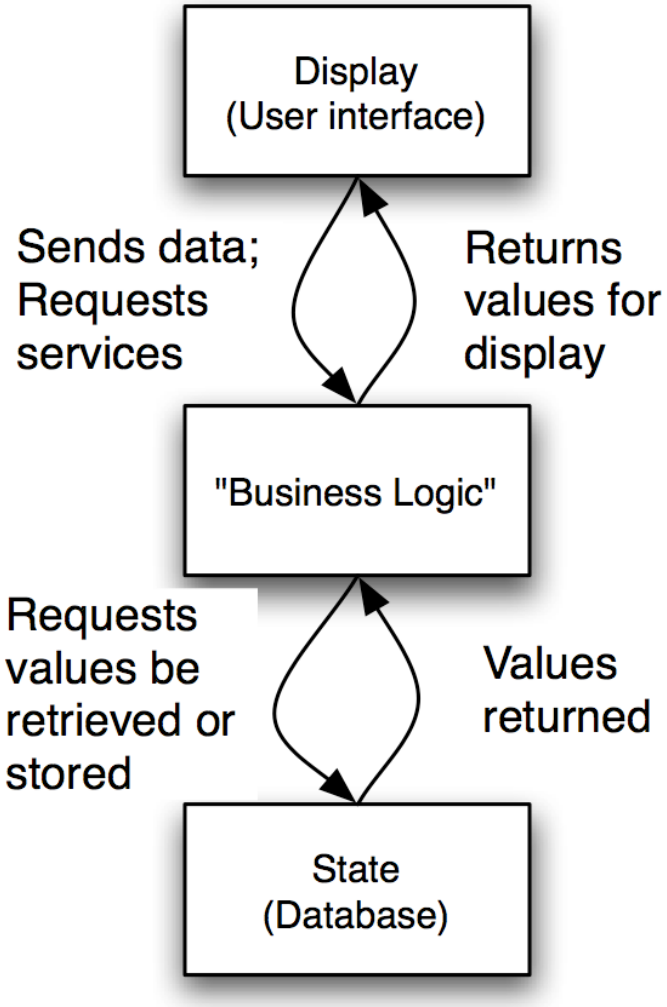
Patrón
Modelo – Vista – Controlador (MVC)

- **Objetivo:** Separar la información, presentación e interacción del usuario.
  - Cambia el modelo? -> enviar notificación a la vista y al controlador
  - Las acciones del usuario (interfaz) generan un evento que se envía al controlador que a su vez actualizará el modelo



Patrón
Estado-Lógica-Presentación (3capas)

- **Objetivo:** Separación entre estado, presentación y funcionalidad.
  - Estado: Datos
  - Aplicaciones de negocios
  - Juegos multi-usuario
  - Aplicaciones Web

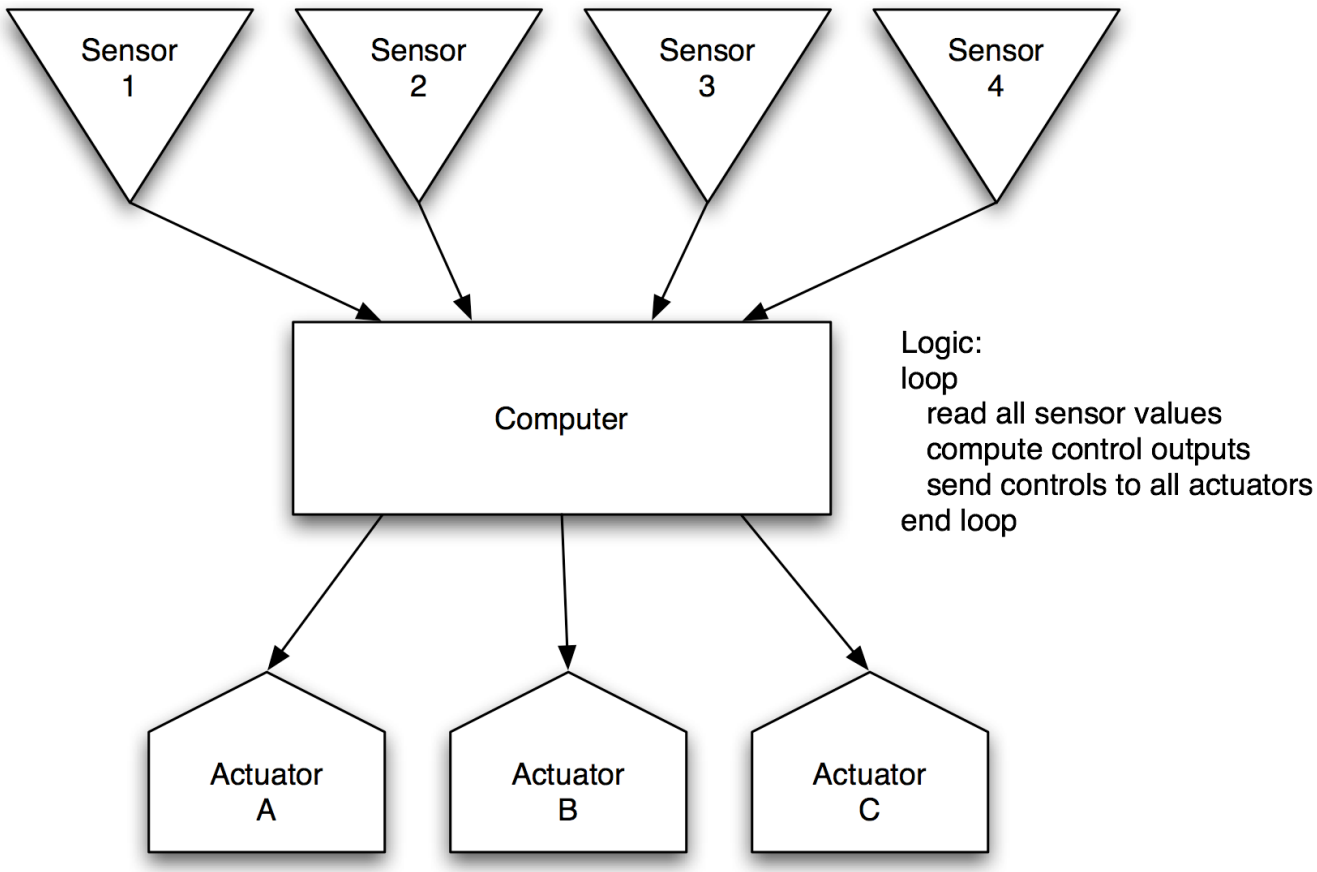


Patrón
Sensor – Controlador – Actuador

Objetivo: Separar la captura de información (leer), del procesamiento y de la manipulación de objetos (actuar).

Juego: Lunar Lander

Joysticks, robots, domótica, …



Estilos Arquitectónicos

Estilo arquitectónico

- Colección nombrada (con nombre) de decisiones de diseño arquitectónico que:
  - Se aplican a un contexto de desarrollo dado
  - Restringen las decisiones de diseño que son específicas a un sistema particular dentro de ese contexto
  - Obtienen las ventajas de cada sistema resultante

Propiedades básicas del Estilo Arq.

- Vocabulario de elementos de diseño
  - Tipos de conectores, componentes y elementos de datos
    - Ej. Servidores, objetos, mensajes, filtros, etc.
- Conjunto de reglas de configuración
  - Restricciones topológicas que determinan cómo pueden componerse los elementos
    - Ej. Cardinalidad: un elemento puede conectarse a lo más con 2 otros componentes
- Interpretación semántica
  - Las composiciones tienen un significado bien definido
  - Es posible analizar los sistemas creados

Beneficios de los estilos

- Reuso de diseño
  - Soluciones bien entendidas aplicadas a nuevos problemas
- Reuso de código
  - Implementación compartida de los aspectos invariantes de un estilo
- Facilita la comprensión de la organización del sistema
  - Ej: “Cliente-Servidor” ya indica cómo es el sistema
- Interoperabilidad
  - Debido a la estandarización del estilo y su mayor comprensión
- Análisis específico al estilo
  - Permite restringir el espacio de diseño
- Visualizaciones
  - Figuras específicas al estilo calzan con el modelo mental “ingenieril"

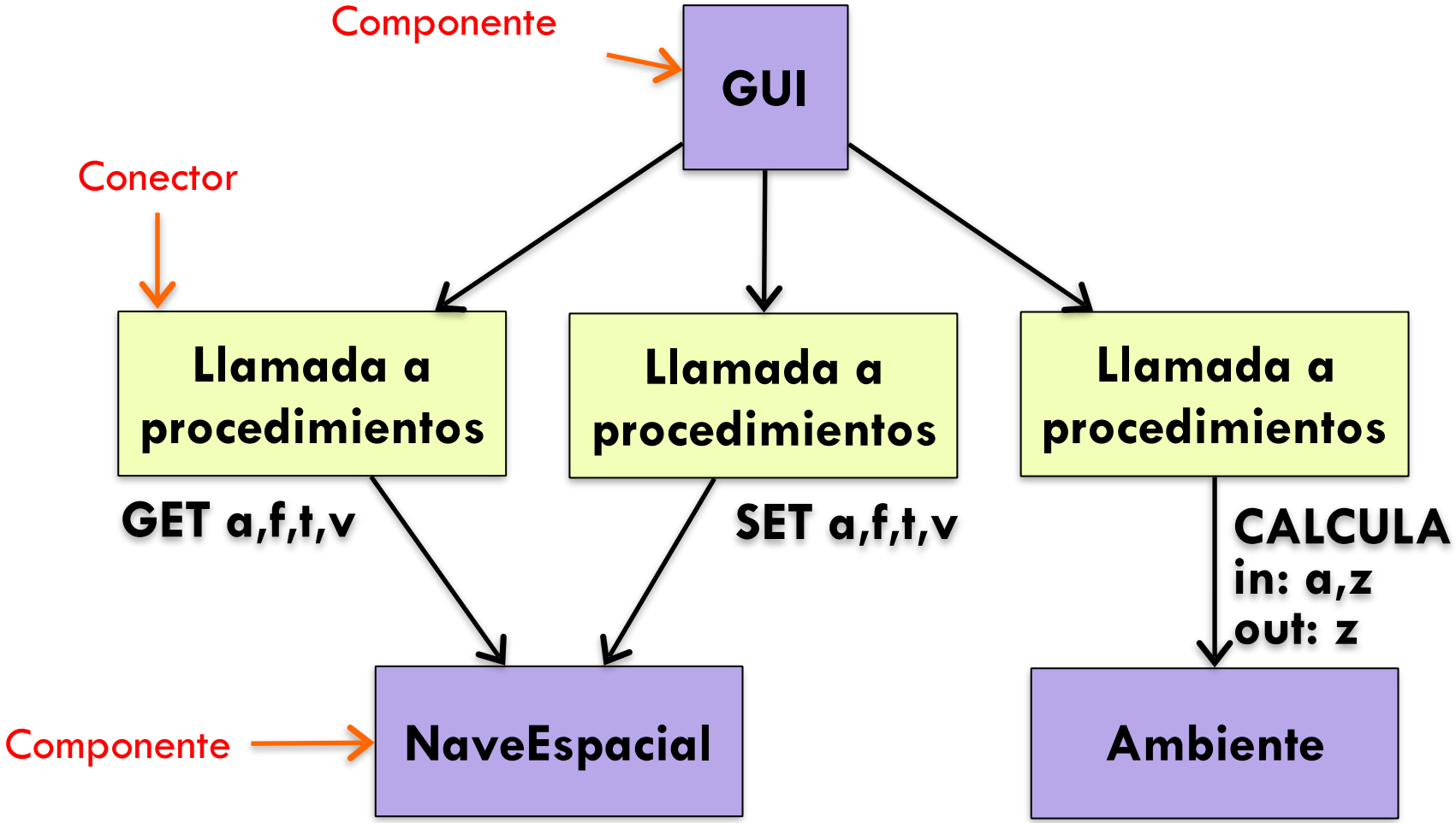
Estilos comunes

- 1.Influenciados por el lenguaje
  - 1.1 Main y sub-rutinas
  - 1.2 Orientado al objeto
- 2. En capas
  - 2.1 Máquinas virtuales
  - 2.2 Cliente-servidor
- 3. Flujo de datos
  - 3.1 Por Lotes, secuencial
  - 3.2 Tubería y Filtro (pipe & filter)
- 4. Memoria compartida
  - 4.1 Pizarra
  - 4.2 Basado en reglas
- Intérprete
  - Intérprete
  - Código móvil
- Invocación implícita
  - Basado en eventos
  - Publisher-Subscribe
- Peer-To-Peer
- Derivados
  - C2, Corba

Estilo: Orientado al Objeto

- Vocabulario:
  - Componentes: Objetos (data + operaciones)
  - Conectores: Mensajes e invocación de métodos
- Invariantes del estilo:
  - Los objetos son responsables de la integridad de su representación interna
  - La representación interna se oculta a otros objetos
- Ventajas:
  - Maleabilidad infinita de la implementación interna del objeto
  - Descomposición del sistema en conjuntos de agentes que interactúan
- Desventajas:
  - Objetos deben conocer las identidades de los servidores
  - Hay efectos secundarios en las invocaciones de los métodos

Estilo: Orientado al Objeto



Estilo: En capas

- Componentes:
  - Capas.
    - Cada capa expone una interfaz (API)
  - Modelo cerrado
    - Capas, *rol servidor* (proveedor para capas superiores), *rol cliente* (consumidor de capas inferiores)
  - Modelo abierto
    - Algunas capas pueden ser ignoradas
- Conectores:
  - Protocolos de interacción entre capas

Estilo: En capas

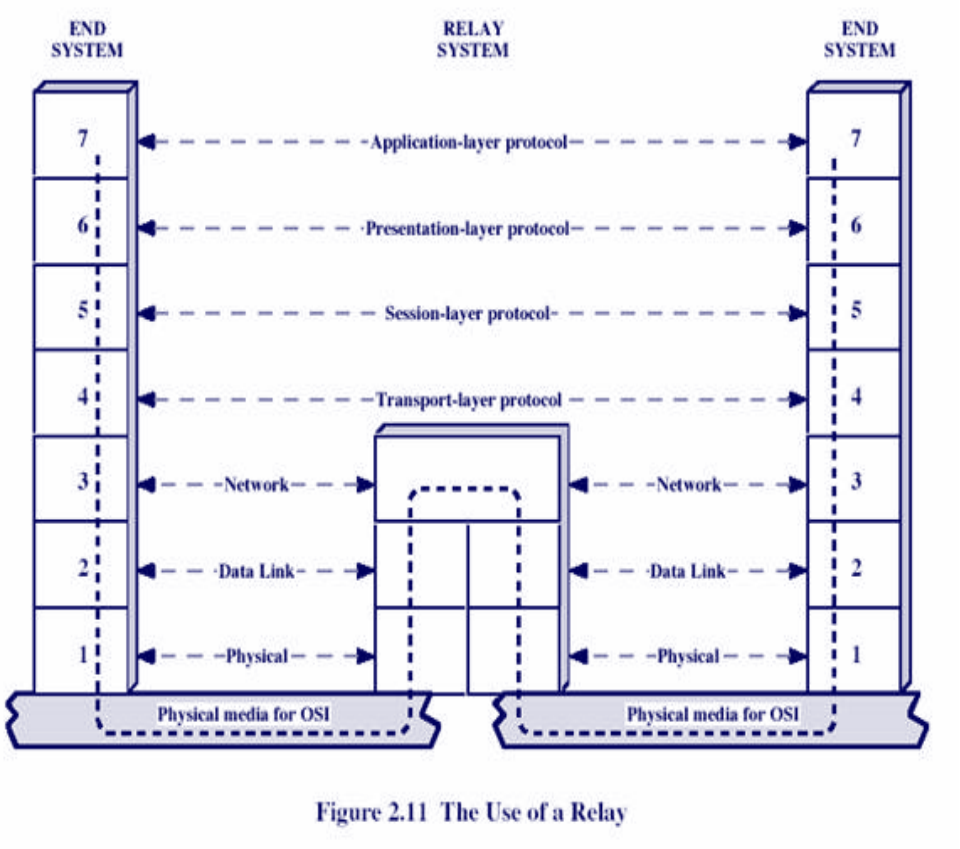
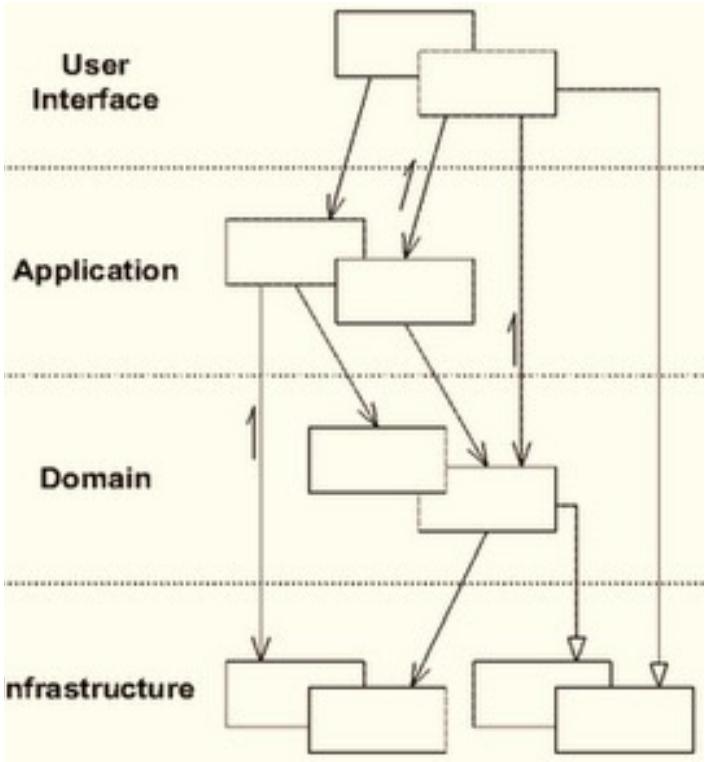


Figure 2.11 The Use of a Relay

Modelo cerrado OSI



Modelo abierto

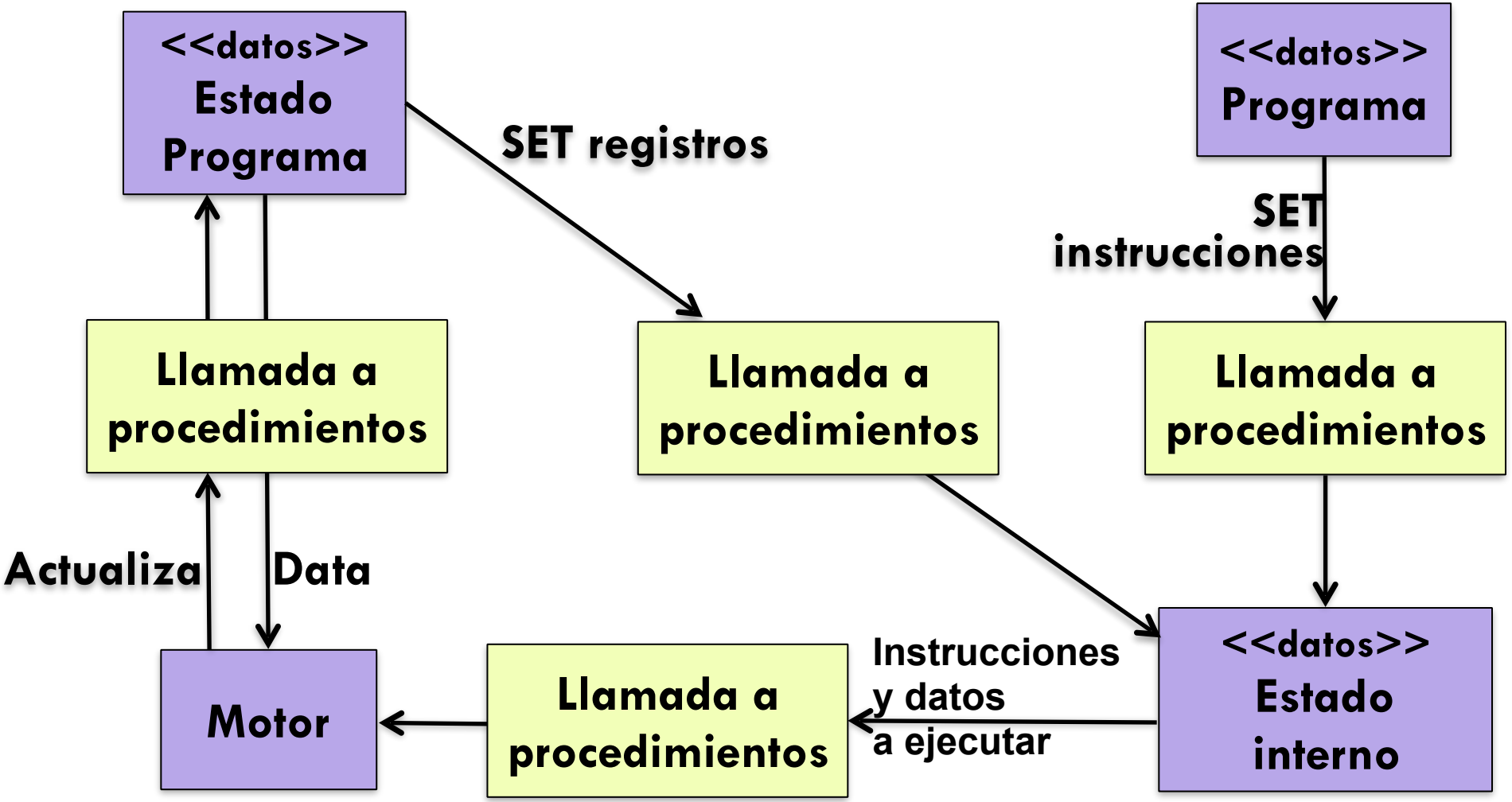
Estilo: En capas

- Ventajas:
  - Aumenta los niveles de abstracción
  - Evolución y reuso
    - Cambios en una capa afecta solo a las capas adyacentes
  - Se pueden tener diferentes implementaciones de una capa manteniendo la interfaz
  - Pueden definirse interfaces estandarizadas para librerías y frameworks
- Desventajas:
  - No es aplicable universalmente
  - Desempeño
    - Saltarse capas, nivel de abstracción correcto

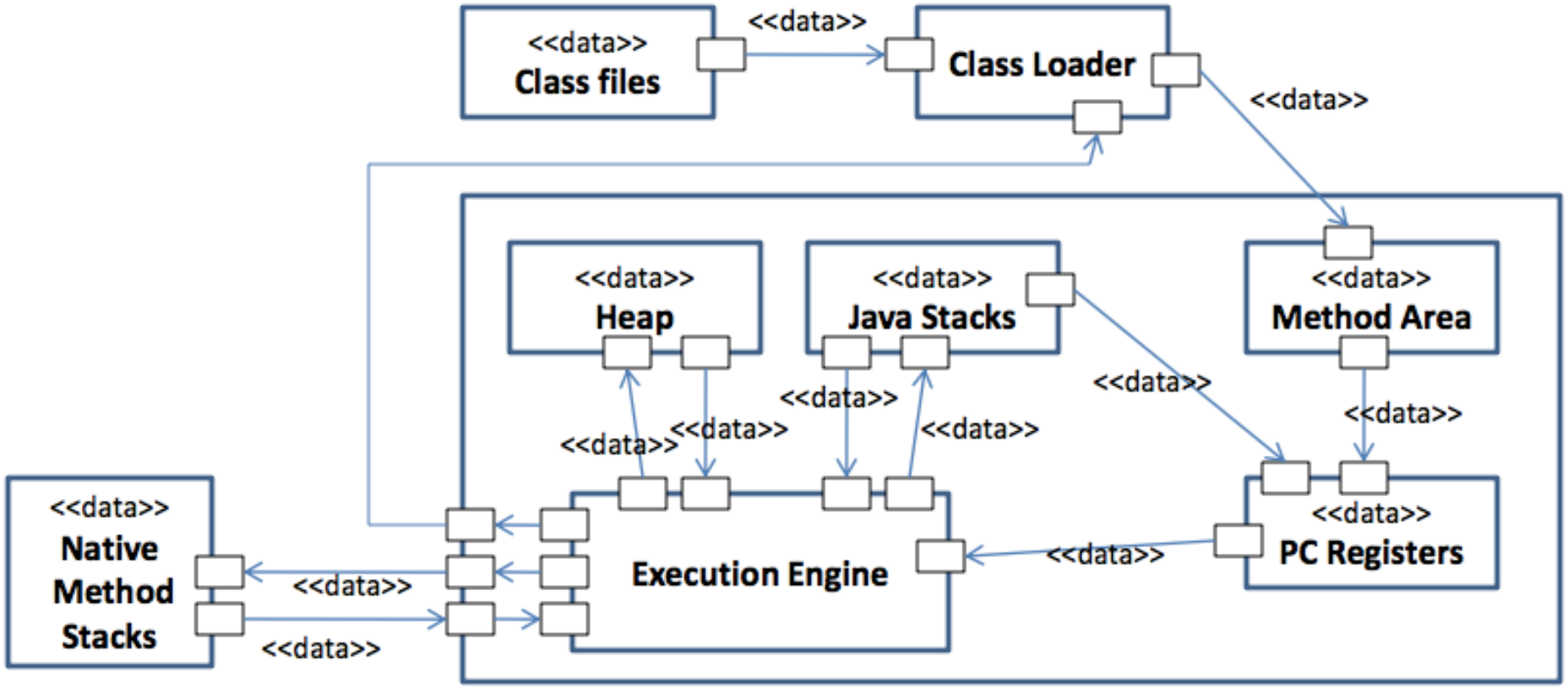
Estilo: En capas

- Máquinas virtuales:
  - Componentes: Programas (subcomponentes) a ser ejecutados, motor de ejecución, estado (datos) de los programas, estado (datos) internos del motor
  - Conectores: llamada a procedimientos
  - Topología: Capas cerradas o Grafo Acíclico Dirigido (DAG)
  - Ventajas: Portabilidad y flexibilidad
  - Desventaja: Difícil de implementar

Estilo: Máquina virtual



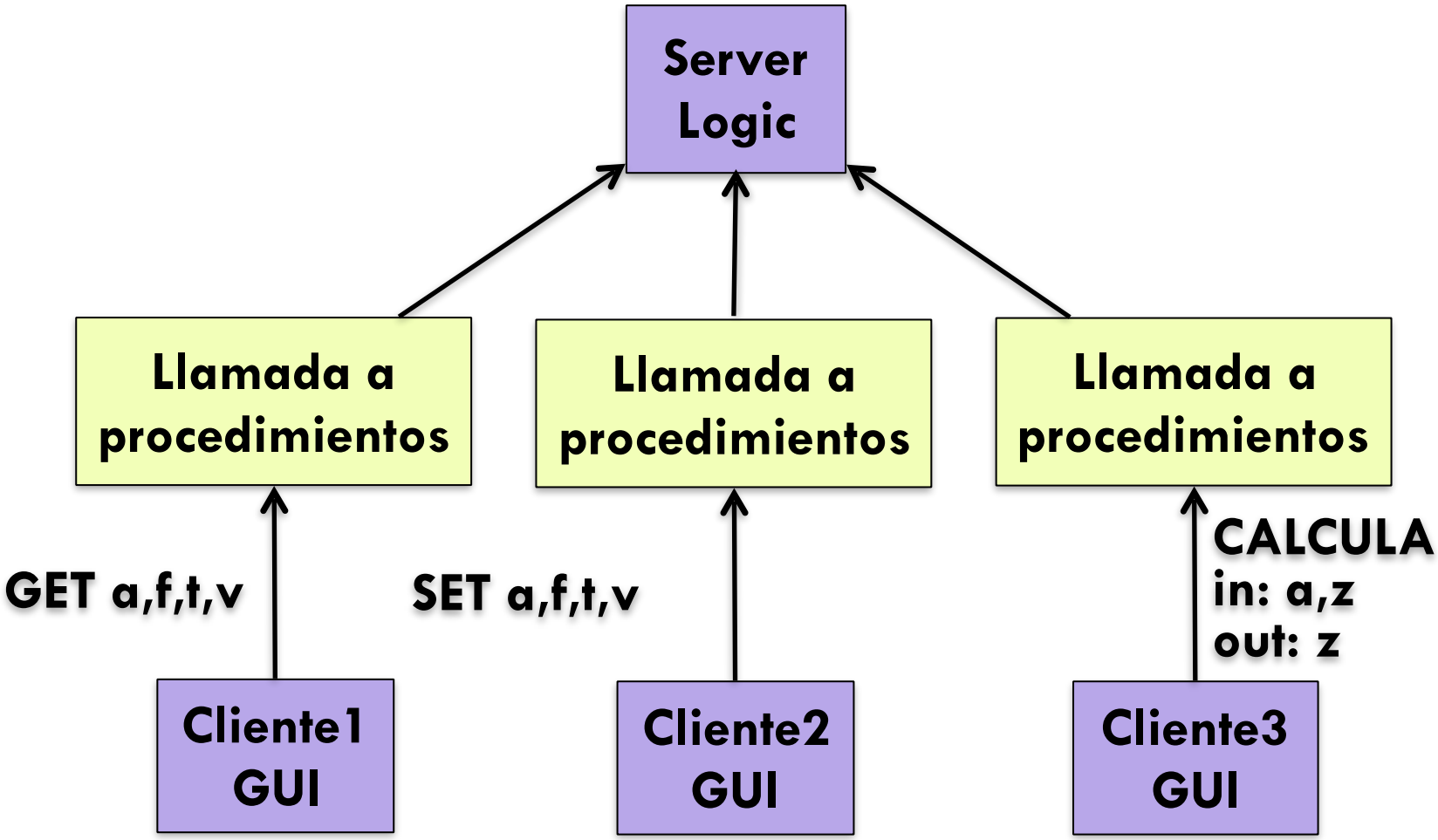
Ejemplo: Máquina virtual de Java



Estilo: En capas

- Cliente/Servidor:
  - Componentes: clientes y servidores
  - Servidores desconocen a los clientes y su número
  - Clientes conocen la identidad del servidor
  - Los clientes no pueden hablar con otros clientes
  - Conectores: protocolo basado en RPC

Estilo: En capas



Estilo: Cliente/Servidor

- RPC: Remote Procedure Call (RPI: - Invocation)
  - Cliente invoca a un método o procedimiento que se ejecuta remotamente y se queda ***esperando*** la respuesta
  - ***Cliente debe saber*** cómo hacer esa llamada
    - Conoce la ***interfaz*** del procedimiento (parámetros y tipos) y empaqueta (marshalling) y desempaqueta (unmarshalling) los mensajes
    - Conoce la dirección del servidor

Estilo: Flujo de datos

- Por lotes, secuencial
  - Componentes:
    - Programas separados se ejecutan en orden
  - Conectores:
    - Personas (sneaker-net), sistemas de mensajería, colas, etc.
  - Elementos de datos pasados explícitamente entre programas
    - Ej.: Procesamiento de transacciones bancarias
  - El “abuelo” de los estilos arquitectónicos

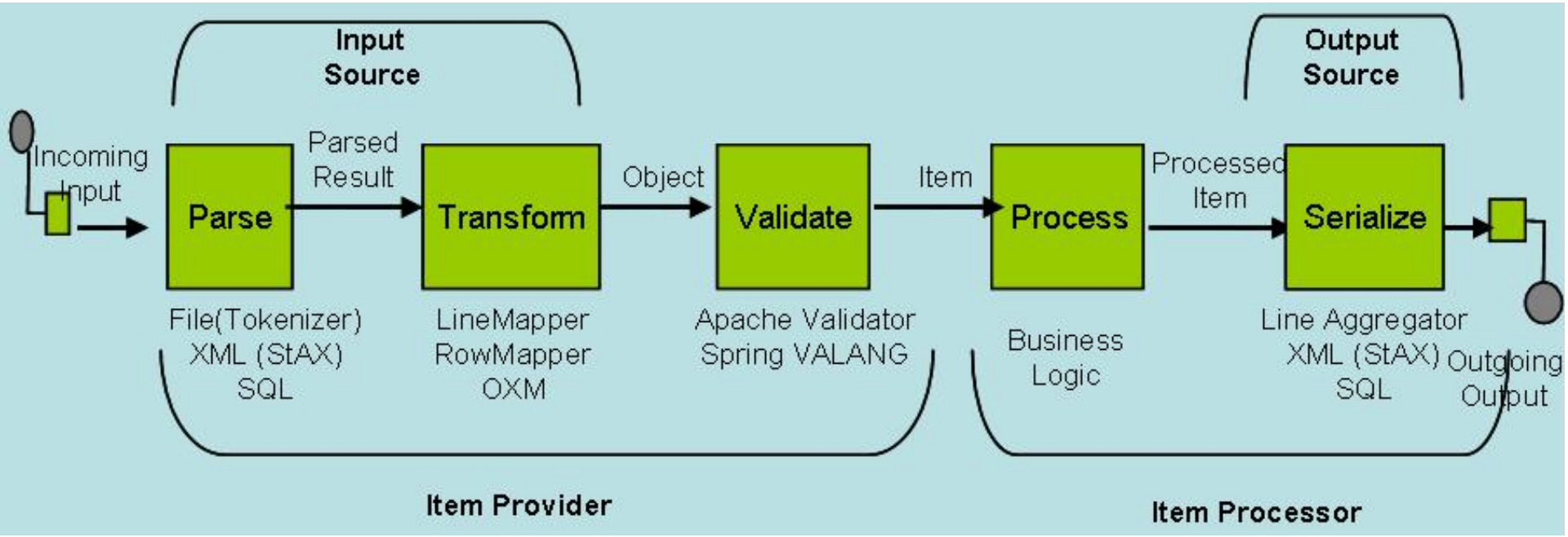
Estilo: Flujo de datos

- Pipe & Filter
  - Componentes:
    - Filtros: Transforman los flujos de datos de entrada en flujos de salida
    - Posiblemente producen salida incrementalmente
  - Conectores: Pipes (conducen los datos)
  - Invariantes:
    - Filtros son independientes (ni hay estado compartido)
    - Los filtros no conocen la direccionalidad de las tuberías (pipes)

Estilo: Flujo de datos

- Pipe & Filter.
  - Ej. UNIX shell

```
ls orden | grep -e Agosto | sort
```



Estilo: Flujo de datos

- Pipe & Filter
  - Variaciones:
    - Pipelines: Secuencias lineales de filtros
    - Pipelines acotados: Se limita la cantidad de datos del pipe
    - Pipes tipificados: Data fuertemente tipificada
  - Ventajas:
    - Añadir, reemplazar y reusar filtros
      - Es posible enlazar (hook) 2 filtros
    - Análisis
      - Rendimiento (throughtput), latencia, deadlock
    - Ejecución concurrente

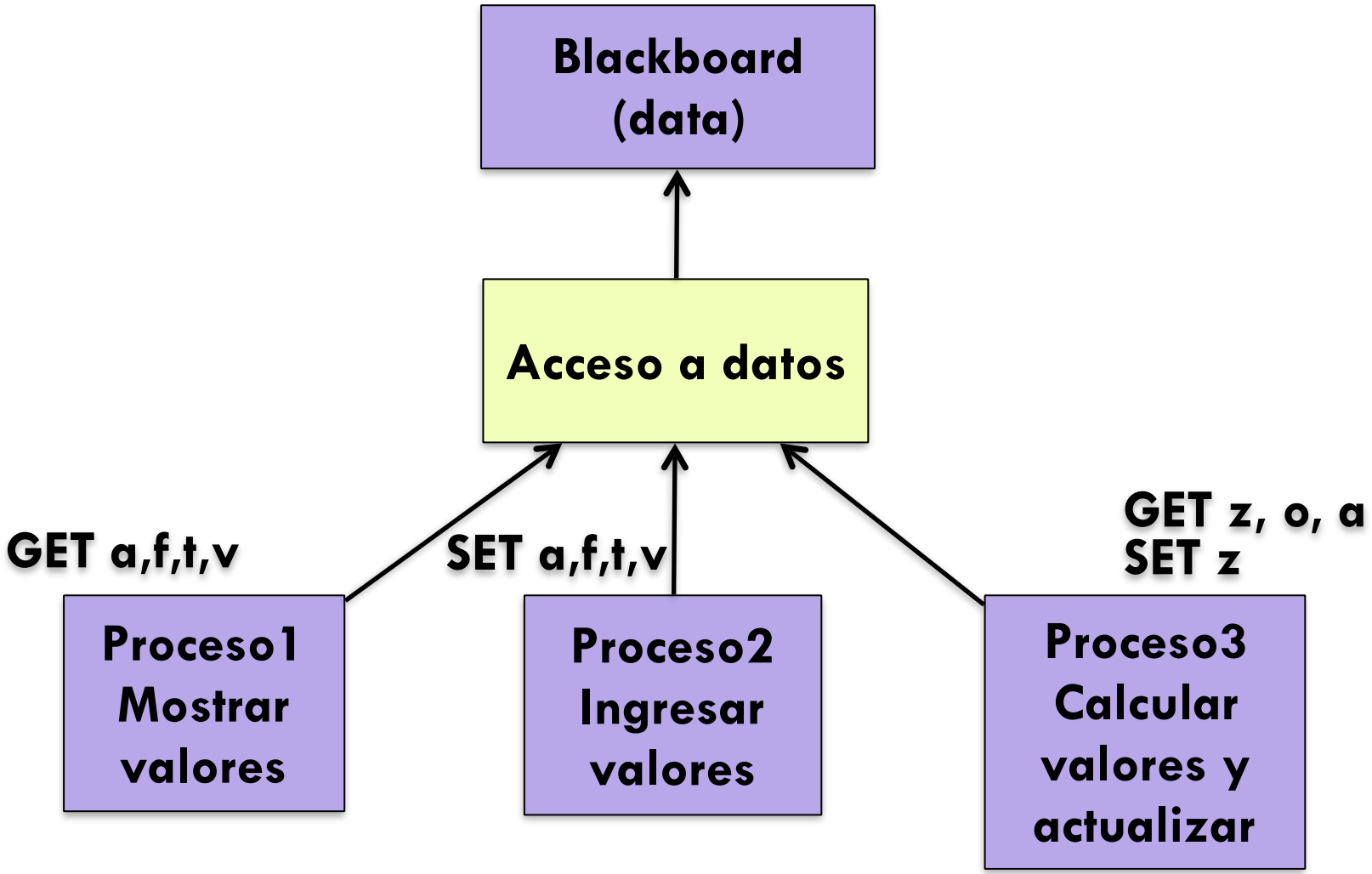
Estilo: Flujo de datos

- Pipe & Filter
  - Desventajas:
    - Requiere organización del procesamiento
    - No soporta aplicaciones interactivas
    - Mínimo común denominador en la transmisión de datos

Estilo: Memoria Compartida

- Pizarra (Blackboard)
  - Componentes:
    - Estructura de datos compartida (pizarra)
    - Componentes que operan sobre la pizarra
  - El estado de la pizarra controla el flujo de ejecución
    - AI (robótica)
    - Circuitos integrados
    - Compiladores

Estilo: Memoria Compartida



Estilo: Memoria Compartida

- Basado en reglas
  - Un motor de inferencia parsea las entradas de los usuarios y determina si es un “hecho”, una “regla” (y los añade a la base de conocimiento) o una “consulta” (y la resuelve)
  - Los hechos y reglas se añaden a la base de conocimiento
  - Las consultas se ejecutan sobre la base de conocimiento para evaluar las reglas que se puedan aplicar y resolver la consulta

Estilo: Memoria Compartida

- Basado en reglas
  - Componentes:
    - Interfaz de usuario, Motor de inferencia, Base de conocimiento
  - Conectores:
    - Componentes fuertemente interconectados vía llamadas a procedimientos y /o llamadas a memoria compartida
  - Elementos de datos:
    - Hechos y consultas

Estilo: Memoria Compartida

- Basado en reglas.

  Jess (Java Expert System Shell).

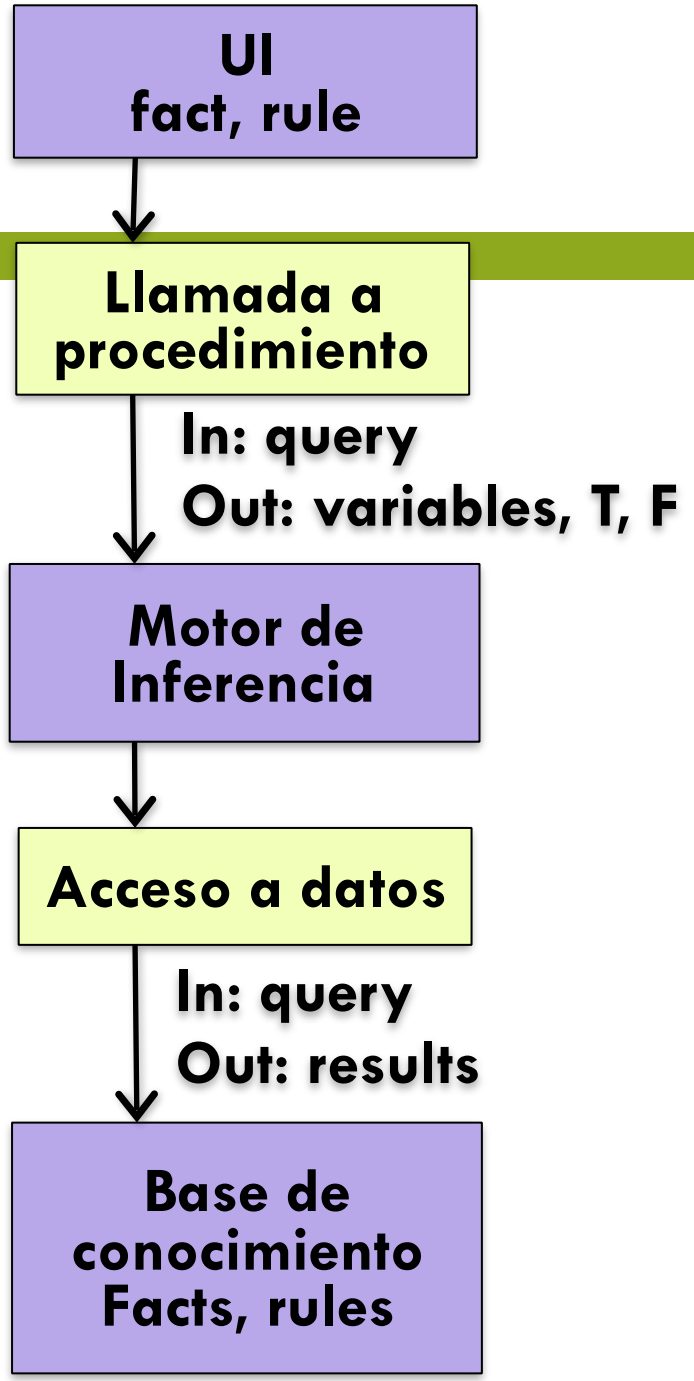
```
(defrule welcome-toddlers  "Saludo a niños!"
(person {age < 3})
 =>
((System.out) println "Hola, pequeño!"))
```

```
(person (age ?a) (firstName ?f) (lastName ?l))
```

```
(assert (person(age 12)(firstName maria)(lastName lopez)))
(assert (person(age 2)(firstName lucas)(lastName martin)))
(assert (person(age 5)(firstName pedro)(lastName perez)))
```

Estilo:
Memoria compartida

- Basado en reglas
- La “conducta” de la aplicación se extiende mediante reglas
- Riesgo: Al aumentar el número de reglas es ***muy*** dificil entender las interaccciones entre las reglas



Estilo: Intérprete

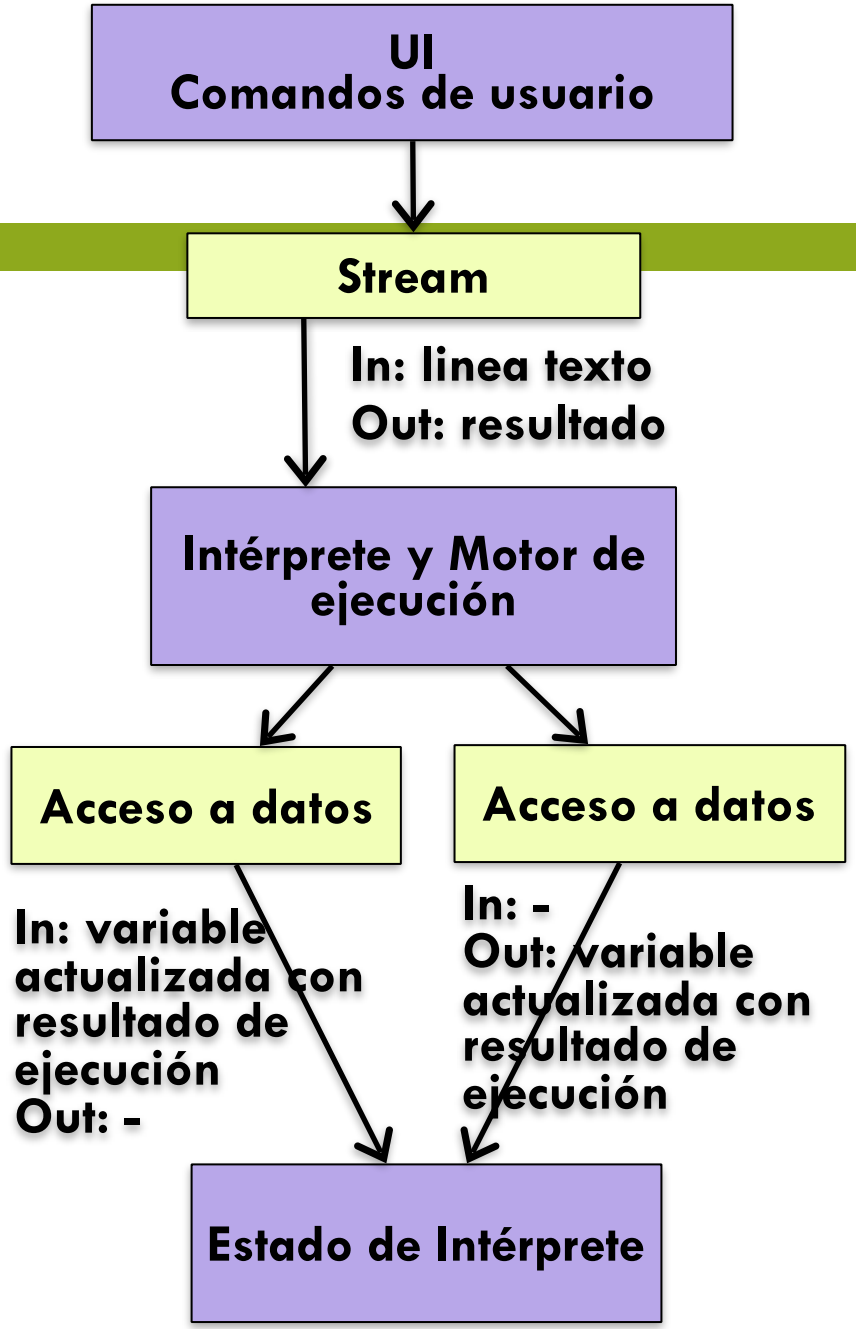
- El intérprete parsea y ejecuta comandos de entrada, actualiza el estado mantenido por el intérprete

- Componentes:
  - Intérprete de comandos, programa de interpretación del estado, interfaz de usuario

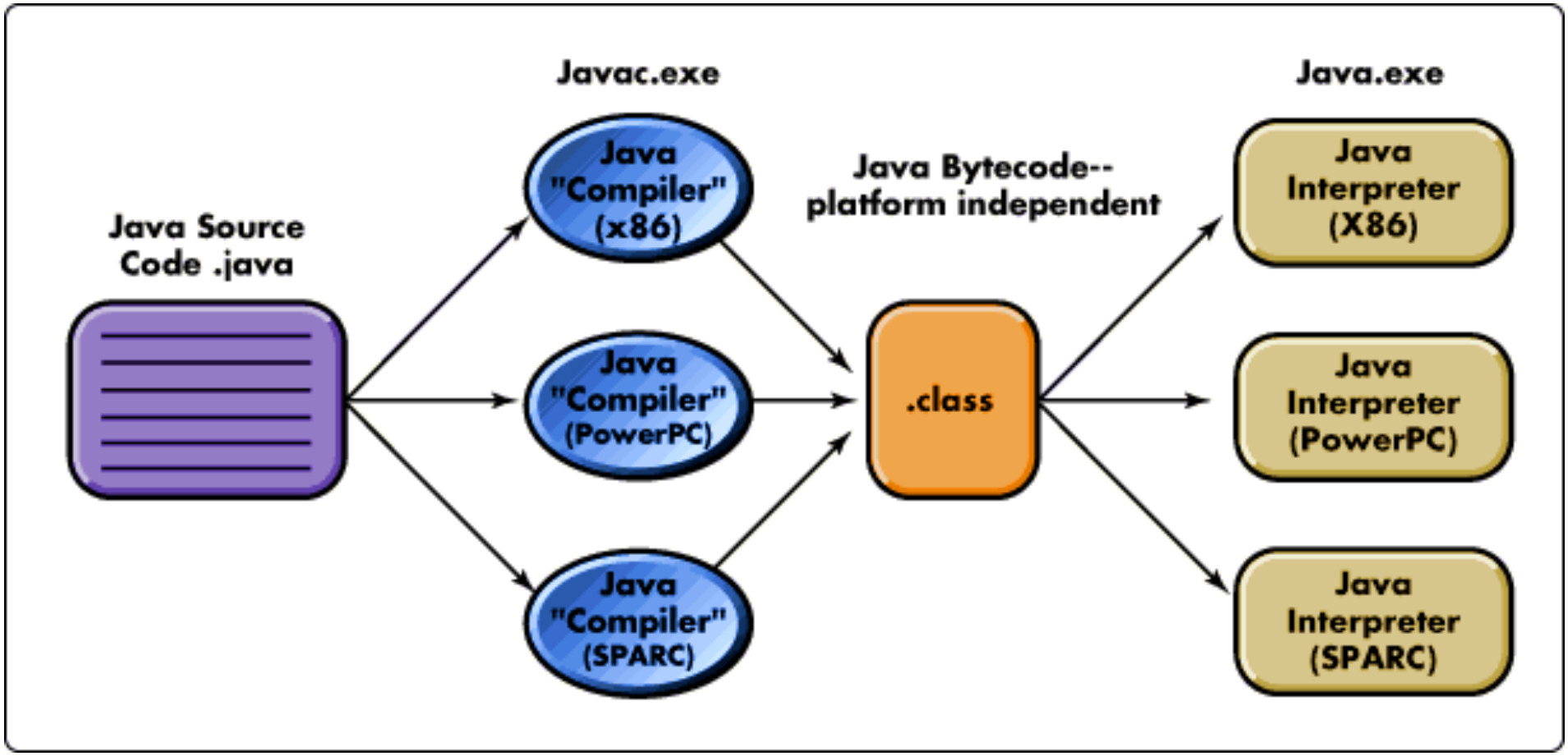
- Conectores:
  - Llamadas a procedimientos y estado compartido

Estilo: Intérprete

- El conjunto de comandos se modifica en tiempo de ejecución
- La arquitectura permanece constante
- Programación de interfaces de usuario
- Lisp, Scheme



Ejemplo Intérprete: Java

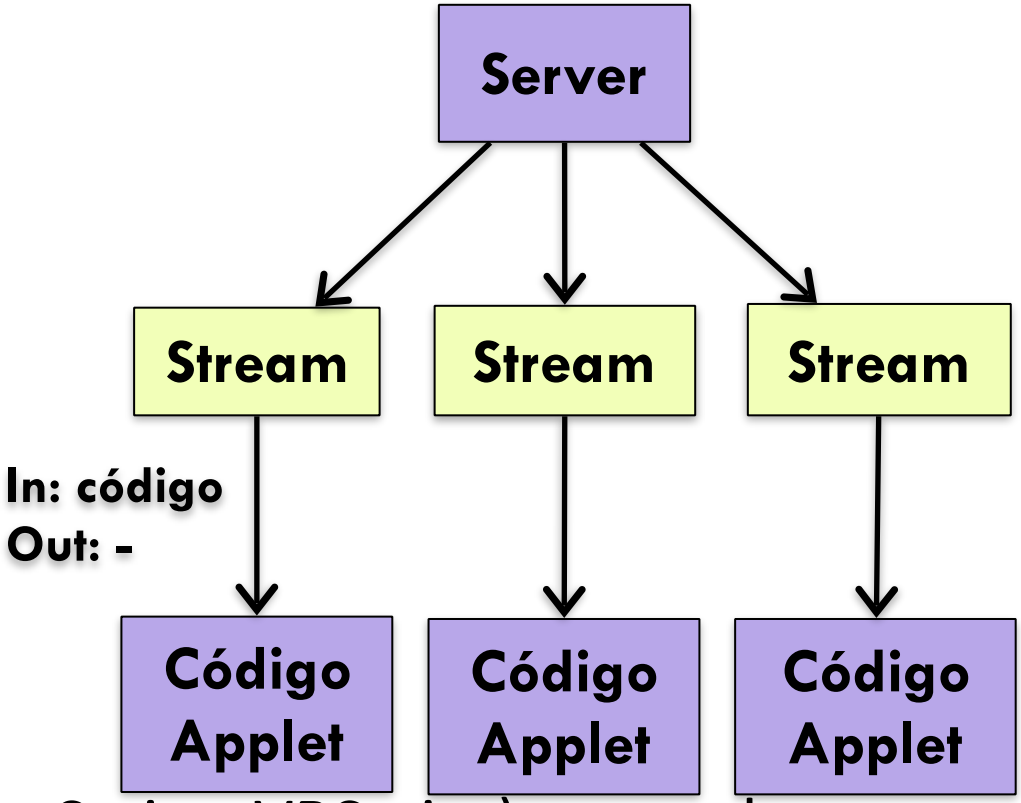


Estilo: Código móvil

- Un elemento de datos se transforma dinámicamente en un componente de procesamiento de datos
- Componente:
  - Puerto de ejecución, maneja la recepción del código y del estado; compilador de código / intérprete
- Conectores:
  - Protocolos de red y elementos para empaquetar código y datos para transmisión
- Elementos de datos:
  - Representaciones de código como datos; estado del programa y datos

Estilo: Código móvil

- Variantes:
  - Código bajo demanda
  - Evaluación remota
  - Agentes móviles



- Lenguajes de scripting (JavaScript, VBScript), controles ActiveX, macros embebidas Word/Excel, etc

Estilo: Invocación implícita

- Anuncio de eventos en lugar de invocación de métodos
- Componentes:
  - “Listeners” registran el interés en algún componente y asocian métodos a eventos
  - Los métodos se registran de manera implícita
  - Interfaces de componentes son métodos y eventos
- Conectores:
  - Invocación explícita e implícita en respuesta a eventos

Estilo: Invocación implícita

- Invariantes del estilo:
  - “Publicadores” no conocen los efectos de los eventos
  - No se asume cómo se procesara la respuesta a eventos
- Ventajas:
  - Reuso de componentes, evolución del sistema (desarrollo y ejecución)
- Desventajas:
  - Estructura de sistema no es intuitiva
  - Componentes no permiten control del sistema
  - No hay conocimiento de la respuesta de los componentes al evento, ni del orden de las respuestas

Estilo: Invocación implícita

- Publisher - Subscriber:
  - Subscriptores se registran / de-registran para recibir mensajes específicos o contenido específico
  - Publicadores difunden mensajes a subscriptores de manera síncrona o asíncrona
- Topología:
  - Subscriptores conectan a publicadores directamente o reciben notificaciones vía protocolo de red de intermediarios

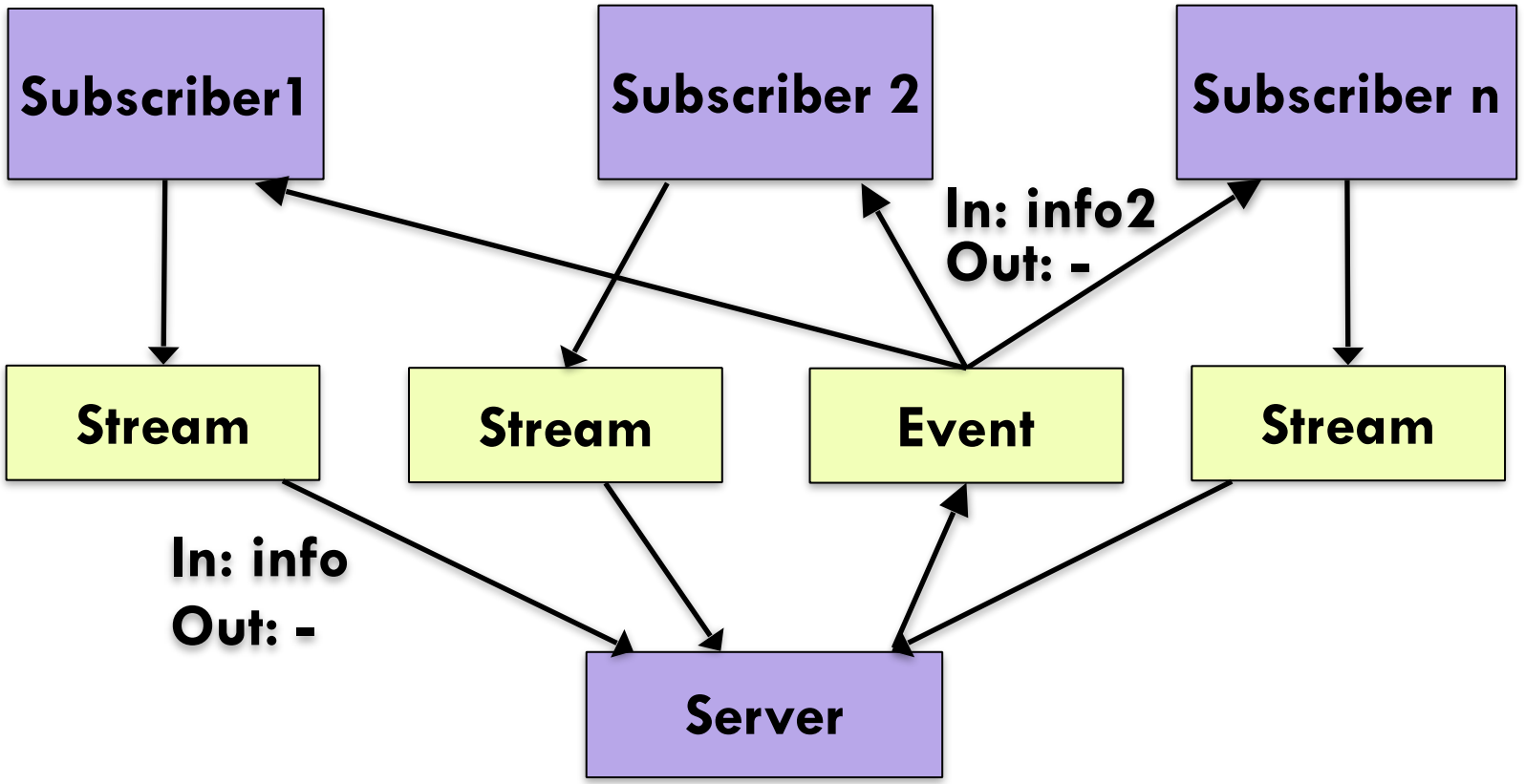
Se publica información de manera muy eficiente con un acoplamiento muy bajo

Estilo: Invocación implícita

- Publisher - Subscriber:
  - Componentes:
    - Publishers, subsribers, proxies (distribución)
  - Conectores:
    - Protocolo de red
    - Conectores especializados para subscripción basada en contenido
  - Elementos de datos:
    - Subscripciones, notificaciones, información publicada
  - Altamente eficiente para diseminación de contenido en una dirección con componentes de muy bajo acoplamiento

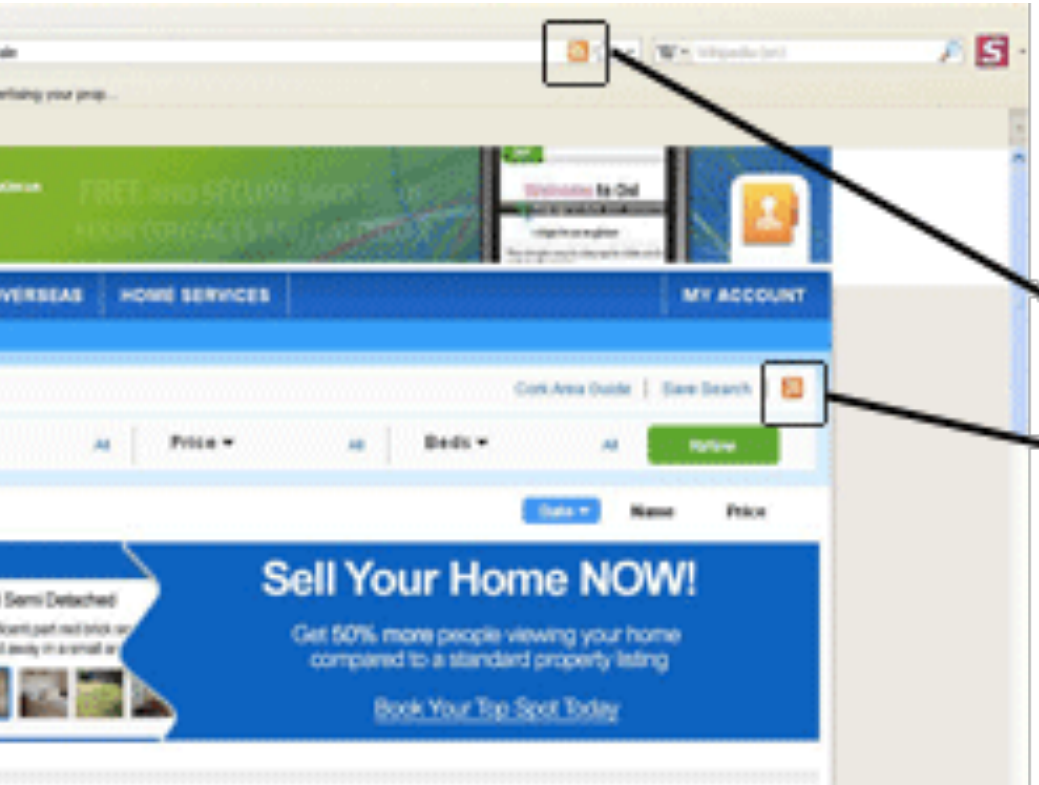
Estilo: Invocación implícita

Publisher – Subscriber:



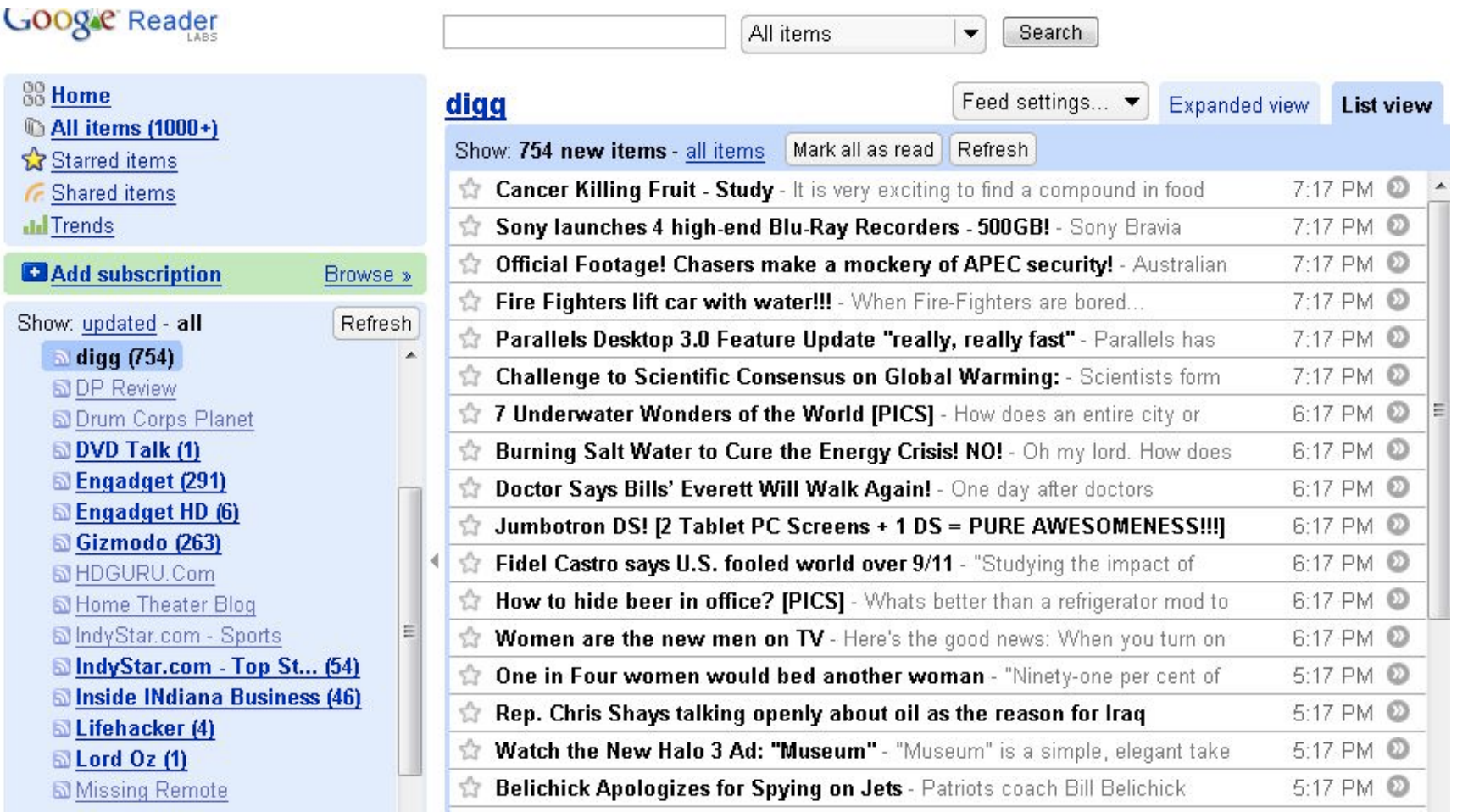
Ej. Sindicación de contenido

- RSS: Rich Site Summary.
- Atom Syndication.



Ej. Feedly/Feedspot/(Google Reader)

- Un “subscriber” que recibe “feeds” desde un “publisher”



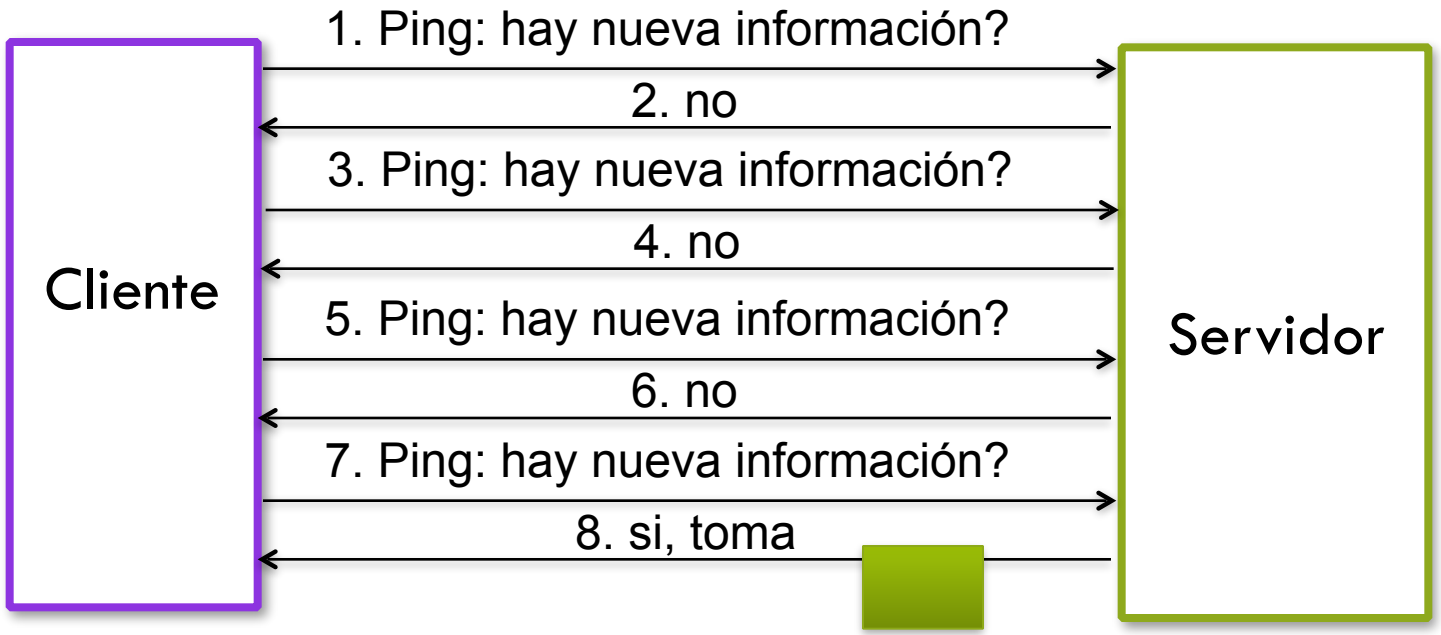
Cómo se ve un feed? … en XML

http://www.flickr.com/services/feeds/photos_public.gne?format=atom

```
<rss xmlns:content="http://purl.org/rss/1.0/modules/content/" version="2.0”>
   <channel>
      <title>Feedeater sample</title>
      <atom:link href="http://darwin.ing.puc.cl/wordpress/?feed=rss2"
        rel="self" type="application/rss+xml"/>
      <link>http://darwin.ing.puc.cl/wordpress</link>
      <description>Otro sitio realizado con WordPress</description>
      <lastBuildDate>Tue, 07 Jun 2011 04:38:04 +0000</lastBuildDate>
      <language>en</language>
      <atom:link rel="hub" href="http://pubsubhubbub.appspot.com"/>
      <item>
        <title>hcard</title>
        <link>http://darwin.ing.puc.cl/wordpress/?p=173</link>
        <dc:creator>   ….
```

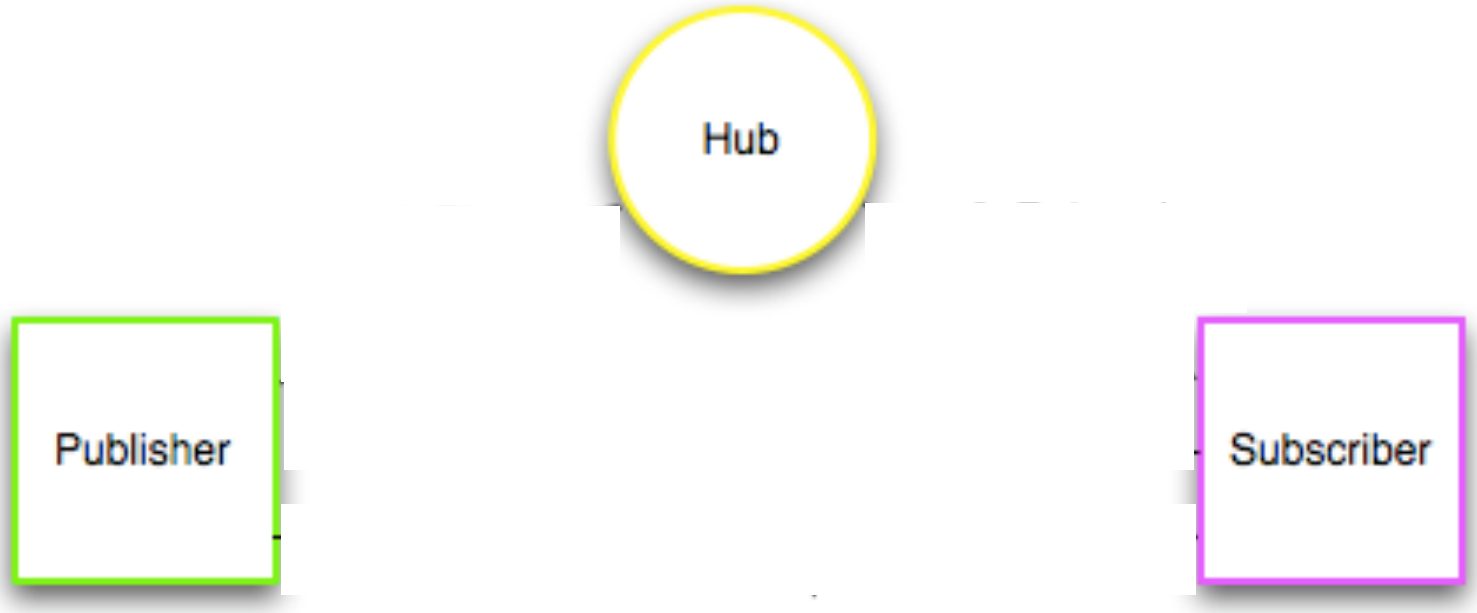
Pero cómo se entrega la información?

- En la versión Cliente – Servidor: Polling

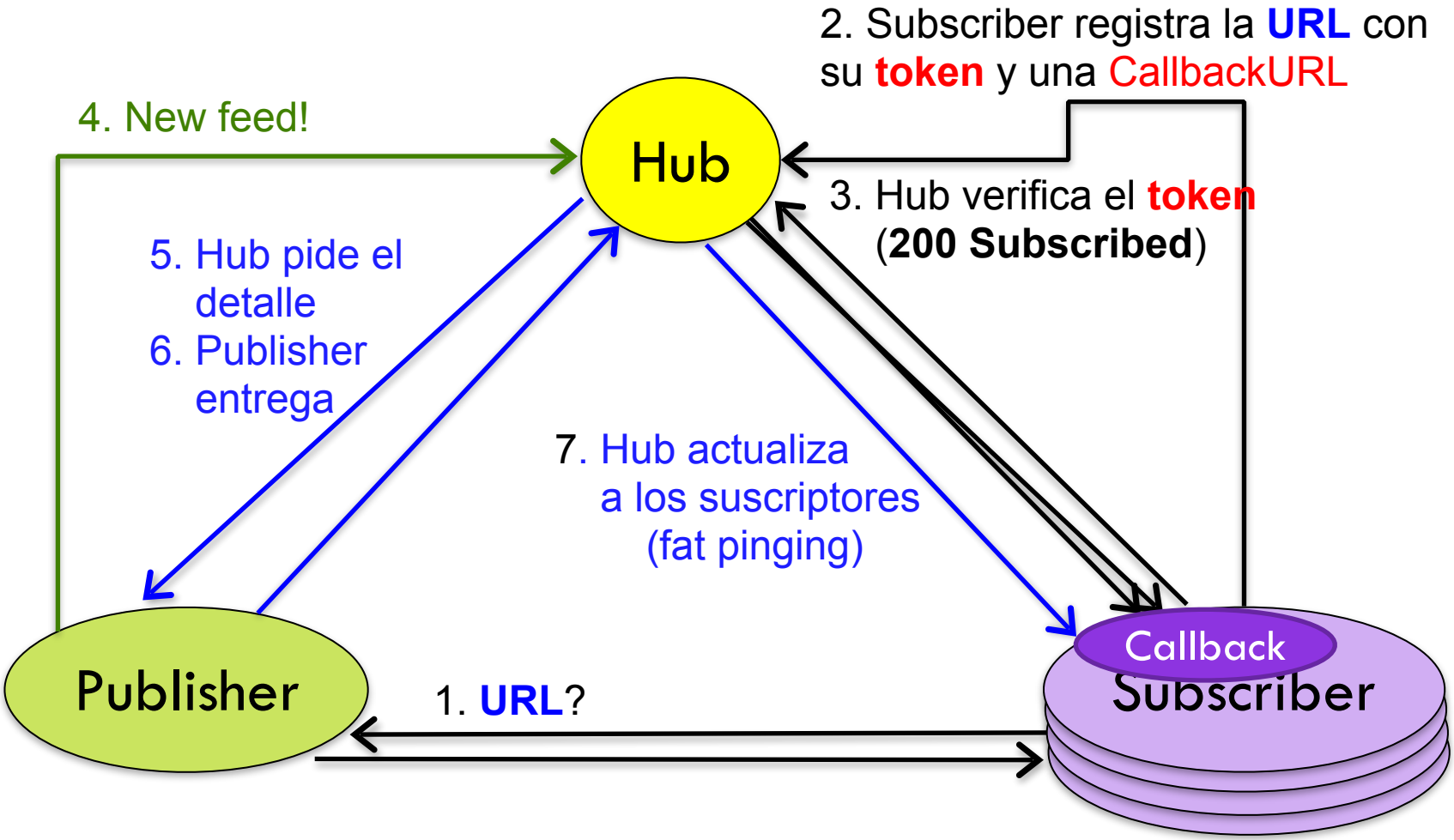


Pero cómo se entrega la información?

- Para operaciones a escala masiva se usa un intermediario: el Hub.



PubSubHubBub (PuSH) … 2009



Pub/Sub sobre la Web

- Light pings:
  - Se envía al subscriber la URL del Feed que ha sido actualizado
  - El subscriber usa la URL para obtener el contenido
- Fat pings:
  - Se envía, al subscriber, el contenido actualizado del feed
- Polling:
  - Se pregunta continuamente al hub para descubrir nuevo contenido

Light pingining Vs Fat pininging

Criterio	XML-RPC ping	changes.xml	SUP	SLAP	XMPP pubsub	rssCloud	PubSubHubbub
Transporte	HTTP	HTTP	HTTP/ HTTPS	UDP	TCP/XMPP	HTTP	HTTP/ HTTPS
Estilo distribución	Ping/Poll	Polling	Polling	Ping/Poll	Push	Ping/Poll	Push
Latencia	Bajo	Alto	Alto	Bajo	Minimo	Bajo	Minimo
Thundering herd	Si	Si	Si	Si	No	Si	No
Spamable	Si	Si	No	No	No	No	No
DoS Publishers	Prevenible	No	No	Prevenible	Prevenible	Prevenible	Prevenible
DoS Relay	Si	No	No	No	No	Si	No
Hosting barato	No	No	No	No	No	Maybe	Yes
Formato msjes	XML schema	XML schema	JSON	Binary packet	XMPP	XML schema	RSS o Atom
Notificaciones seguras	No	No	Algo	No	Si	No	Si
Complejidad publisher	XML-RPC client	XML-RPC client	SUP IDs	UDP send	XMPP send	XML-RPC/ REST ping	REST ping
Complejidad subscriber	Crawl pipeline	Crawl pipeline	Crawl pipeline	Crawl pipeline	XMPP client	Crawl pipeline	WebApp

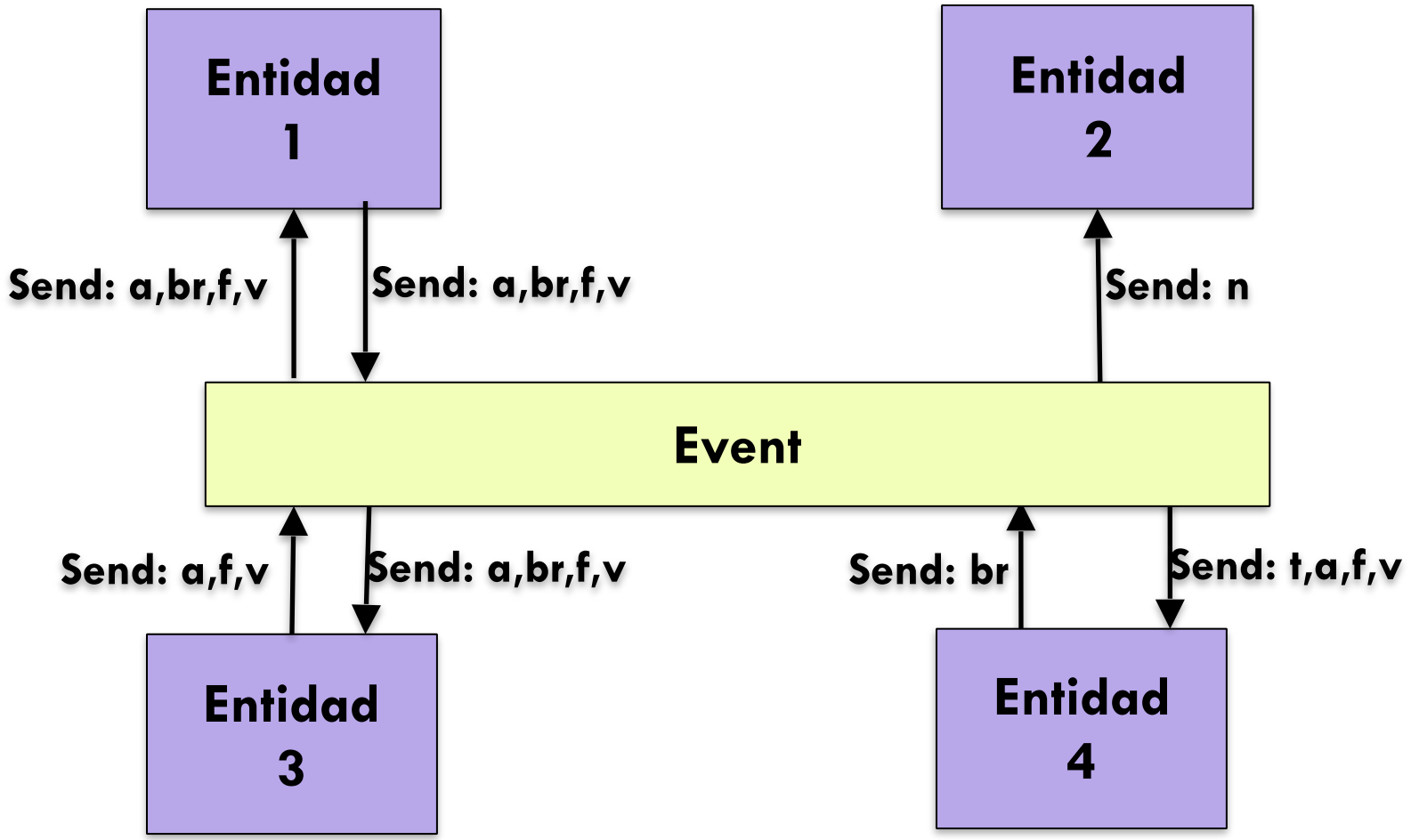
Estilo: Invocación implícita

- Basada en eventos
  - Componentes independientes emiten y reciben eventos asíncronamente sobre *buses de eventos*
  - Evento: Cambio significativo de estado
- Componentes:
  - Generadores y/o consumidores de eventos independientes y concurrentes
- Conectores:
  - Buses de eventos
- Elementos de datos:
  - Datos (entidades de primera clase) enviados sobre el bus

Estilo: Invocación implícita

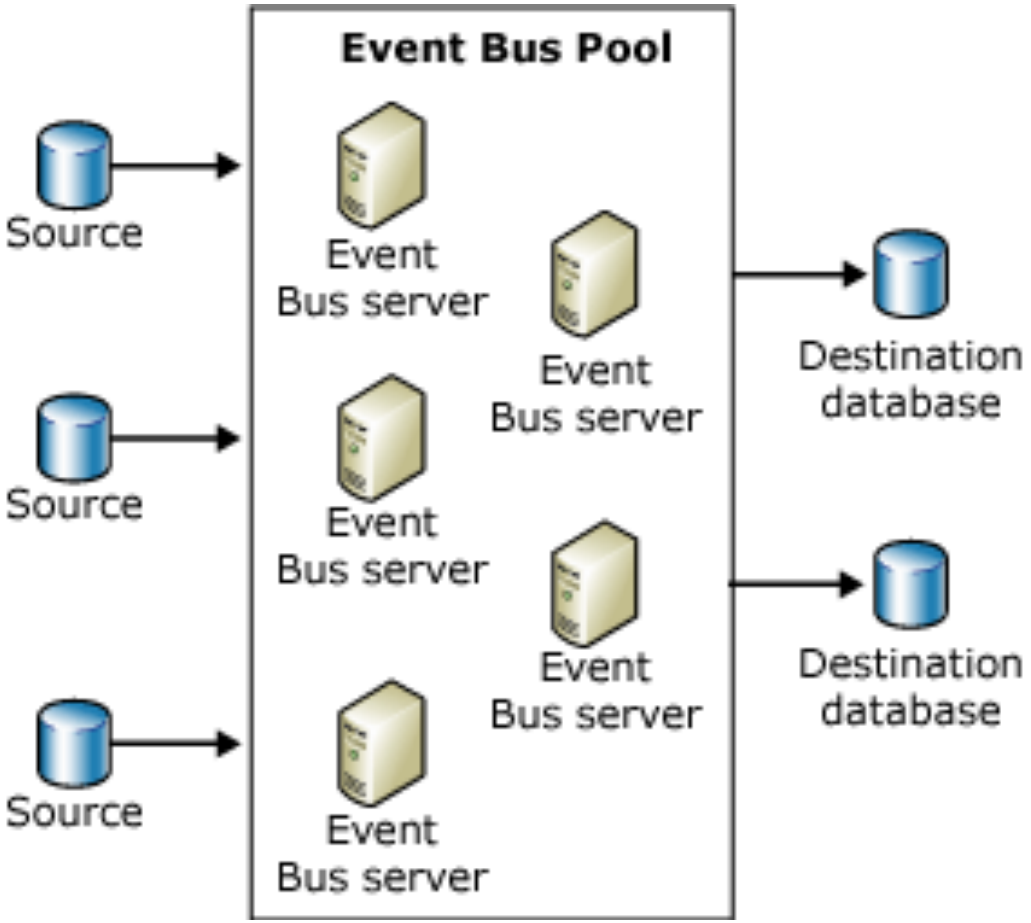
- Topología:
  - Los componentes se comunican mediante los buses de eventos, no directamente unos con otros
- Variaciones:
  - Push: Los componentes reciben los datos de los buses (son empujados hacia ellos)
  - Pull: Los componentes sacan los datos de los buses (tiran de ellos)
- Altamente escalable, fácil de evolucionar, efectivo por aplicaciones altamente distribuidas

Estilo: Invocación implícita



Estilo: Invocación implícita

Microsoft
Biztalk



Business Activity Monitor (BAM) **Event Bus Service**

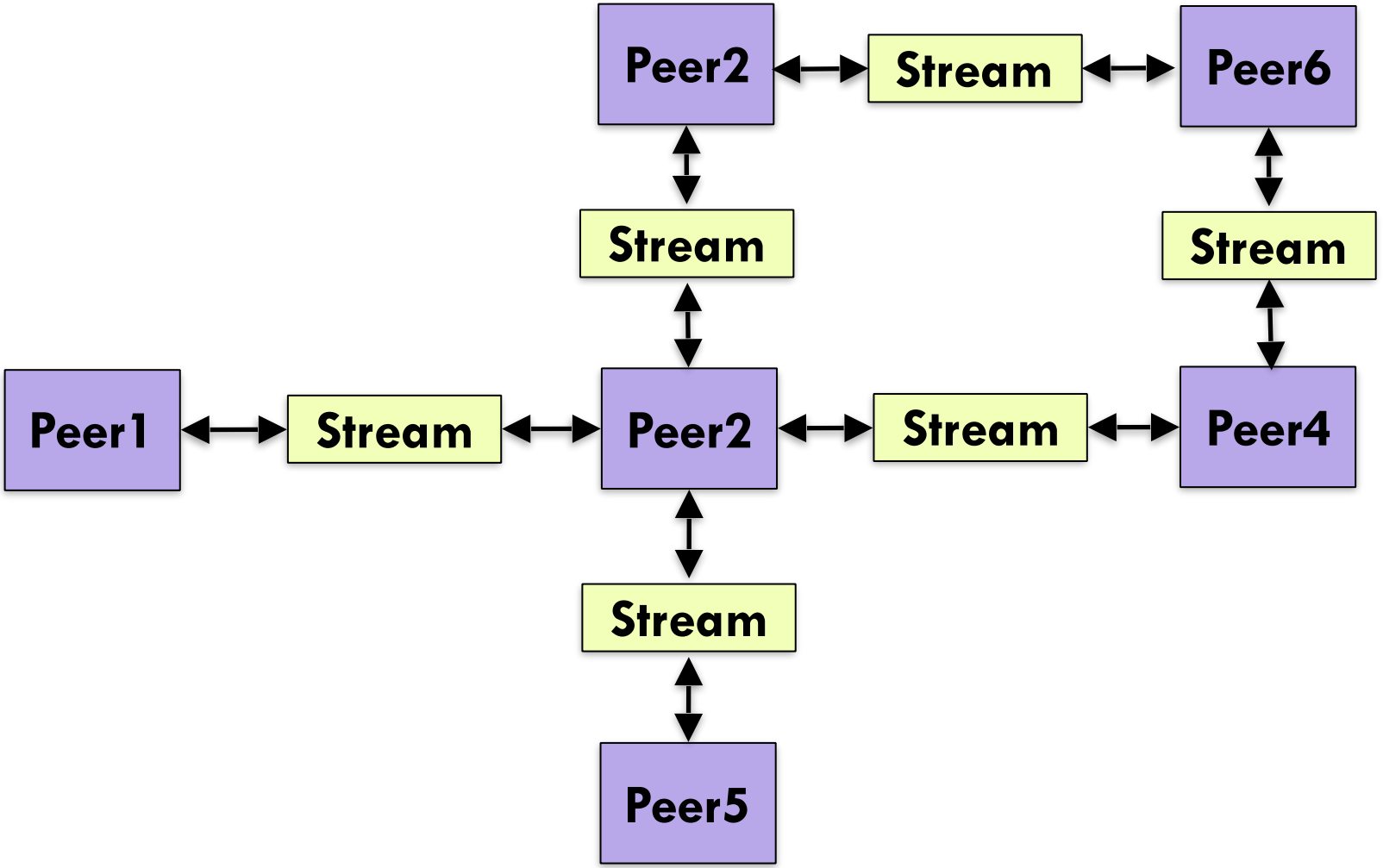
Estilo: Peer-to-Peer

- Estado y conducta se distribuyen entre peers (pares) que actúan como clientes o como servidores
- Componentes:
  - Peers: Componentes independientes tienen un estado propio y controlan su hilo de ejecución
- Conectores:
  - Protocolos de red (a veces ad-hoc)
- Elementos de datos:
  - Mensajes de red

Estilo: Peer-to-Peer

- Topología:
  - La red puede tener conexiones redundantes entre los peers
  - Puede varias arbitrariamente en tiempo de ejecución
- Computación descentralizada:
  - Flujo de control y recursos están distribuidos entre pares
  - Altamente robusto ante fallas de algún nodo
  - Escalable en cuanto a acceso a recursos y procesamientos
  - Protocolo complejo

Estilo: Peer-to-Peer



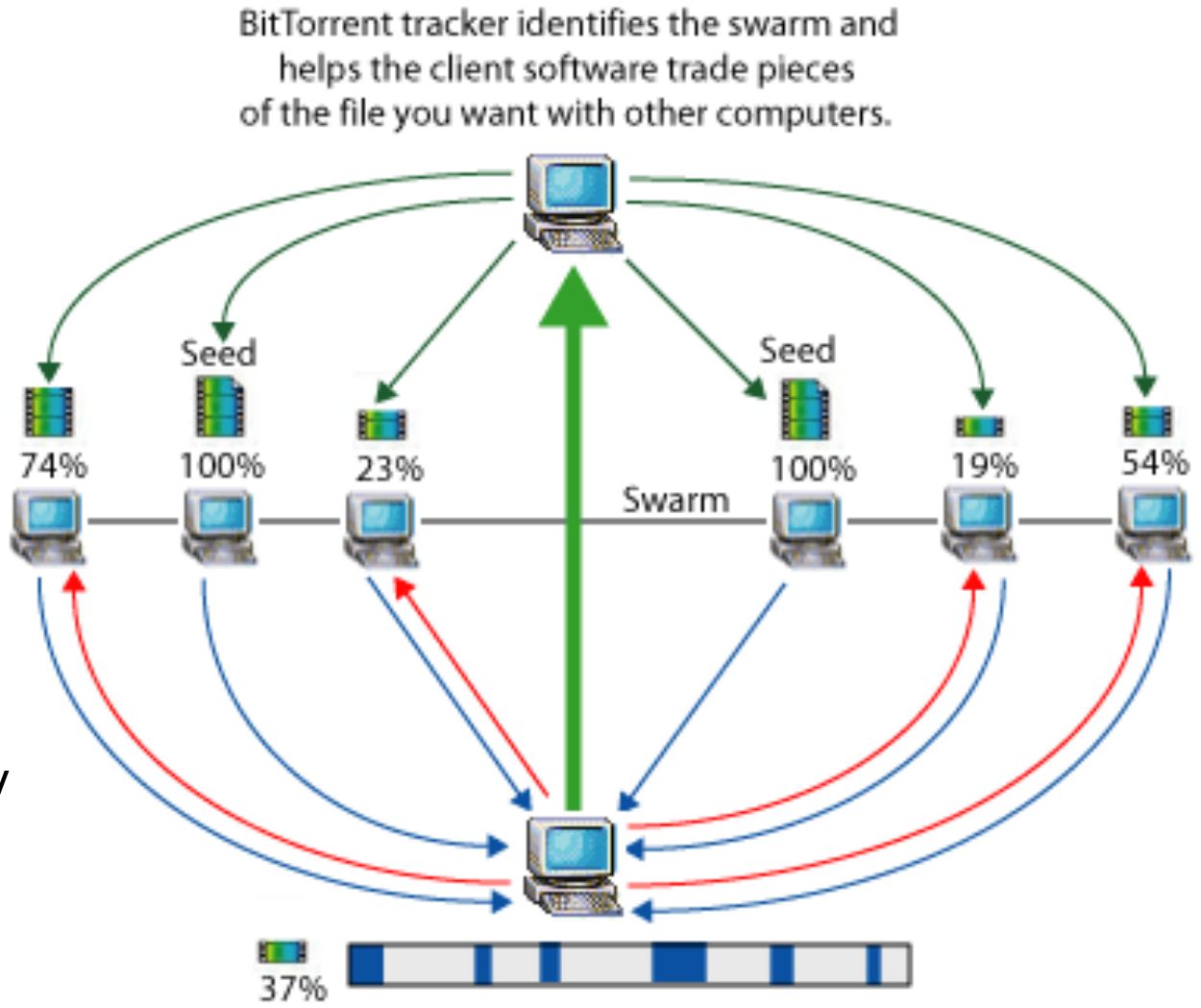
Ejemplo: BitTorrent

Cliente: Peer.
Elemento de datos:
- Tabla de hash distribuida.
- Archivos.

Seed:
- Peer contiene archivo completo.

Swarm:
- Grupo de peers que comparten un archivo.

Tracker:
- Servidor registra seeds y swarms.



Cuándo usar qué?

Estilo	Procesamiento	Cuándo si?	Cuándo no?	Impacto en Calidad
Influencia-do x lenguaje	*- El orden del procesamiento es conocido* *- Los componentes esperan la respuesta de otros componentes para continuar.*			
Main y subrutinas	*- El problema puede des-componerse en problemas pequeños*	*Aplicación es pequeña y simple*	*Se requieren estructuras de datos complejas.* *Se modificará*	*Performance (+)* *Reusabilidad (+)*
Orientado a objetos	*- El problema puede mode-larse via entidades que interactúan.*	*Hay un mapeo claro entre entidades del pro-blema y objetos. Estructuras de datos complejas e interrelacionadas*	*Aplicaciones distribui-das en redes hetero-géneas. Componentes fuertemente indepen-dientes. Se requiere alto performance.*	*Flexibilidad (+)* *Integridad (+)*

Cuándo usar qué?

Estilo	Procesamiento	Cuándo si?	Cuándo no?	Impacto en Calidad
Capas	*- Las tareas se pueden dividir en capas de abstracción.* *- El orden del procesamiento es conocido*			
Máquinas virtuales	*- La data tiene 2 partes: Información y control que indica cómo procesar la información.*	*Una única capa sirve de servicio para otras aplicaciones.* *La interfaz de la capa base es estable.*	*Se requieren varias capas.* *Los datos se acceden por varias capas.*	*Portabilidad (+)*
Cliente Servidor	*- El servidor centraliza y ejecuta un proceso para varios clientes.* *- El cliente hace pedidos al servidor y espera la respuesta para seguir.*	*Puede centralizarse algunas tareas.* *El procesamiento del cliente es limitado.*	*La centralización es punto único de falla. Ancho de banda limitado.* *La capacidad del cliente compite o excede a la del servidor.*	*Escalabilidad (+)* *Flexibilidad (+)*

Cuándo usar qué?

Estilo	Procesamiento	Cuándo si?	Cuándo no?	Impacto en Calidad
Flujo de datos	*- Datos de entrada/salida bien definidos* *- Salida producida al procesar en forma secuencial el input, independiente del tiempo (no hay espera, send-and-forget)*			
En lotes, secuencial	*- La lectura y procesamiento de un conjunto de datos (lote) genera una sola salida.* *- La entrada se procesa por una serie de transformaciones conectadas secuencialmente.*	*Problema dividible en una secuencia de pasos*	*Se requiere: interactividad, concurrencia o acceso a datos random*	*Reusabilidad (+)* *Modificabilidad (+)* *Performance no importa (.)*
Pipe and filter	*- Transformaciones sobre un stream continuo de datos* *- Transformaciones incrementales* *- Una transformación sólo puede empezar cuando la anterior terminó* *- Las transformaciones pueden ser concurrentes.*	*Idem.* *Filtros reusables* *Estructuras de datos serializa-bles*	*Se requiere interacción entre componentes*	*Escalabilidad (+)* *Performance (+)*

Cuándo usar qué?

Estilo	Procesamiento	Cuándo si?	Cuándo no?	Impacto en Calidad
Memoria compar-tida	*- El elemento central es el almacenamiento, representación, gestión y recuperación de grandes cantidades de datos interrelacionados que deben graduarse por largo tiempo.*			
Pizarra	*- Programas inde-pendientes se comunican única-mente a través de un repositorio global.*	*Los cálculos usan/ cambian datos compartidos. El orden de procesa-miento se define on runtime, según datos.*	*Programa accede a partes indepen-dientes de datos. La interfaz de datos puede cambiar. Interacción entre*	*Escalabilidad(+) Modificabili-dad (+)*
Basado en reglas	*- Usa hechos/reglas de una KB para responder una consulta.*	*Los datos y pregun-tas del problema se pueden modelar como reglas de inferencia simples.*	*El número de reglas es grande. Hay interacción entre reglas. Requiere gran performance.*	*Escalabilidad (+) Modificabili-dad(+) Performance (-)*

Cuándo usar qué?

Estilo	Procesamiento	Cuándo si?	Cuándo no?	Impacto en Calidad
Intérprete	*- Se enfoca en la interpretación dinámica (en tiempo de ejecución) de comandos explícitos*			
Intérprete	*- El intérprete parsea y ejecuta un stream de entrada y actualiza su estado.*	*Se requiere conducta dinámica y mucha personalización de usuario.*	*Se requiere alto performance*	*Portabilidad (+) Modificabilidad (+) Performance (-)*
Código móvil	*- Se mueve el código a un host que lo ejecuta.*	*Es más eficiente mover el "proceso" cerca de los datos que al revés. Se customiza código local con código externo, dinámicamente.*	*No se puede garantizar la seguridad del código móvil, o tiene acceso restringido. Se requiere fuerte control de versiones.*	*Modificabilidad(+) Extensibilidad (+) Performance (+)*

Cuándo usar qué?

Estilo	Procesamiento	Cuándo si?	Cuándo no?	Impacto en Calidad
Invocación implícita	*- Se caracteriza por llamadas indirectas e implícitas, tales como, respuestas a notificaciones o eventos.*			
Publisher-Subscribe	*Un publicador difunde mensajes a subscriptores.*	*Componentes débilmente acoplados. Los datos de subscripción son pequeños y fáciles de transportar.*	*No hay middleware para soportar muchos datos.*	*Escalabilidad (+) Flexibilidad (+) Performance (+)*
Basada en eventos	*Componentes independientes emiten y reciben eventos asíncronamente.*	*Componentes concurrentes e independientes. Componentes heterogéneos, distribuidos y en red.*	*Se requieren garantías de procesamiento de eventos en tiempo real.*	*Escalabilidad (+) Flexibilidad (+) Performance (+)*

Cuándo usar qué?

Estilo	Procesamiento	Cuándo si?	Cuándo no?	Impacto en Calidad
Peer-To-Peer	*Los peers guardan estado (datos) y actúan como clientes y como servidores.*	*Los peers se distribuyen en redes que pueden ser heterogéneas e independientes. Se requiere robustez ante fallas independientes. Se requiere alta escalabilidad.*	*No se puede garantizar la confiabilidad de peers independientes. Hay nodos designados para descubirir recursos que pueden no estar disponibles*	*Escalabilidad (+) Flexibilidad (+) Performance (+)*

Preguntas

Proceso de Diseño de Arquitectura

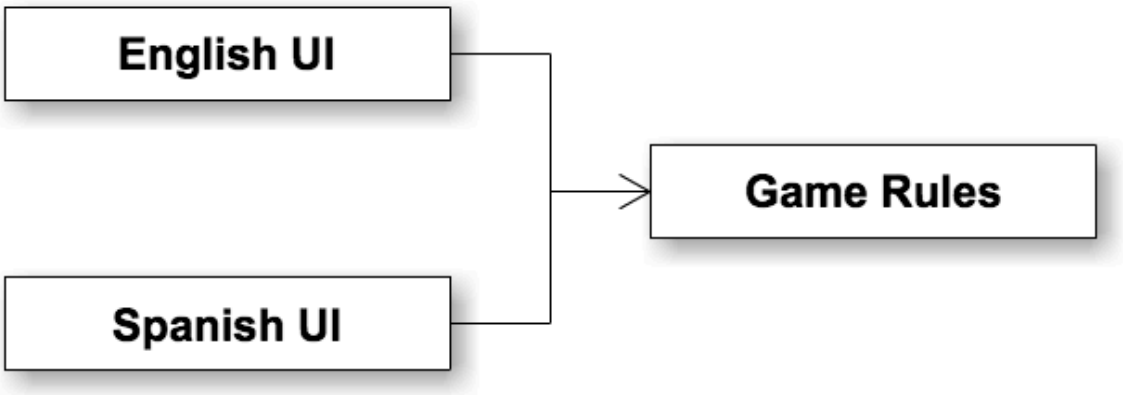


Diagrama simple

Diagrama con datos

Proceso de Diseño de Arquitectura

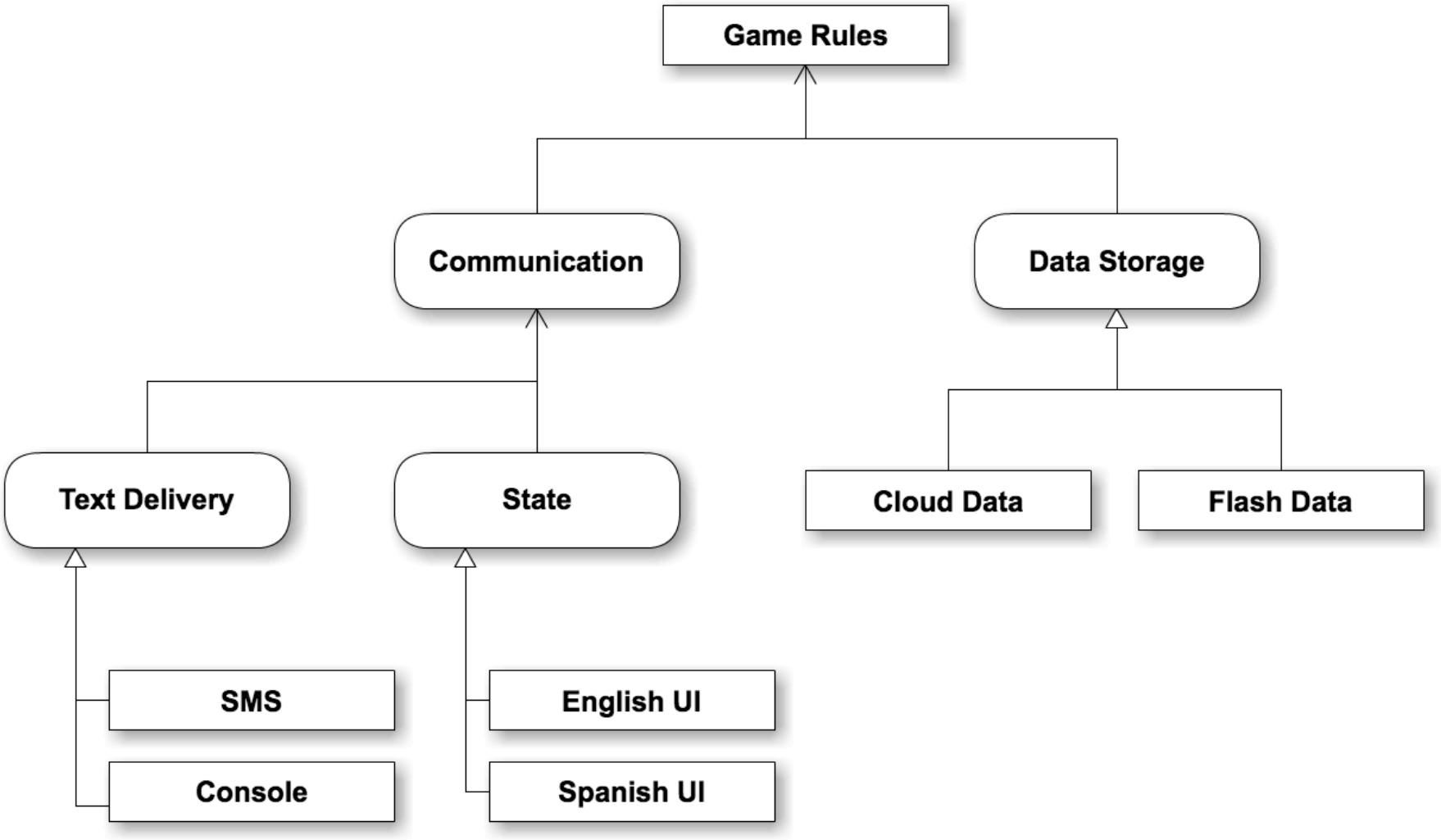


Diagrama Ordenado con Comunicación y estado

Proceso de Diseño de Arquitectura

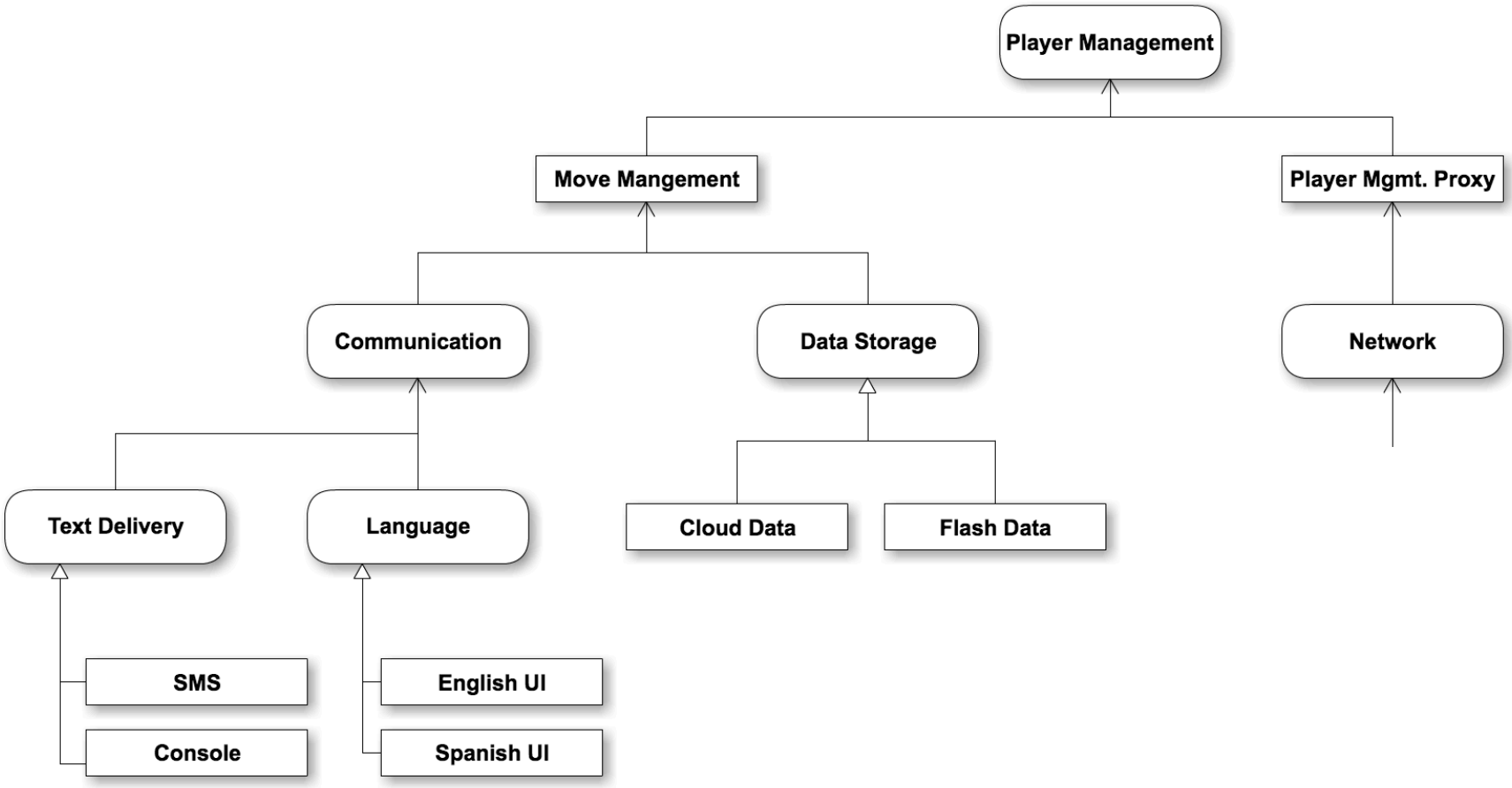


Diagrama enfocado en servicios y extensibilidad

Principio de diseño

- Single responsibility Principle
- Open/Closed Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

URL:https://sub.watchmecode.net/episode/solid-javascript-presentation/