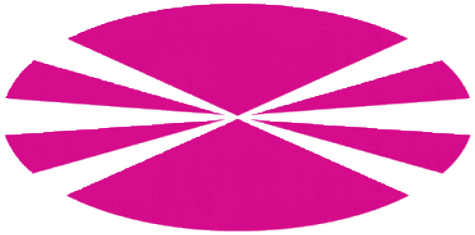


UNIVERSIDADE DA CORUÑA

Arquitectura Software

Laura M. Castro
lcastro@udc.es D4.15
www.madsgroup.org/staff/laura

(2) Modelos y arquitecturas de referencia

(2) Modelos y arquitecturas de referencia

1) Tipos de arquitecturas

   a) Arquitecturas no distribuidas

   b) Arquitecturas distribuidas

   c) Otras arquitecturas

Tipos de arquitecturas

- El diseño de la **arquitectura** trata de entender cómo **organizar globalmente** un sistema
- Es el **primer paso en el diseño** de software
- Produce un **modelo de arquitectura** que describe cómo se organiza el sistema (conjunto de componentes que se comunican)
- Es el **enlace crítico** entre la ingeniería de **requisitos** y el diseño software, ya que **identifica** los principales **componentes** en los que se estructura un sistema y las **relaciones** entre ellos

Tipos de arquitecturas

- Incluso en *desarrollo ágil*, se reconoce que una primera etapa debe ocuparse de establecer una arquitectura general para el sistema
- El *desarrollo incremental* de una arquitectura no suele tener éxito
  - La *refactorización de componentes* es habitual y relativamente sencilla
  - La *refactorización de la arquitectura* de un sistema es normalmente muy complejo y costoso

Tipos de arquitecturas

- Idealmente, la **especificación** de un sistema no debe incluir **ninguna información de diseño**
  - Esto *no es realista*, salvo para sistemas muy pequeños
- Se necesita diseñar la arquitectura para **estructurar y organizar la especificación**

Tipos de arquitecturas

- Podemos hablar de arquitectura software a **dos niveles de abstracción**:
  - *Arquitectura interna*, que se ocupa de la estructura y organización de programas individuales
  - *Arquitectura externa*, que estructura y organiza varios sistemas, programas o componentes

Tipos de arquitecturas

- Podemos hablar de arquitectura software a **dos niveles de abstracción**:
  - *Arquitectura interna*, que se ocupa de la estructura y organización de programas individuales
  - *Arquitectura externa*, q estructura y organiza varios sistemas, progra o componentes

los modelos y patrones que veremos aplican muchas veces a ambos niveles

Tipos de arquitecturas

- Como ya sabemos, los **componentes** de un sistema **implementan los requisitos funcionales**
- Los **requisitos no funcionales** (*disponibilidad, flexibilidad al cambio, rendimiento, seguridad, facilidad de prueba, usabilidad...*) **dependen de la arquitectura** del sistema, de la manera en que los componentes se organizan y comunican

Tipos de arquitecturas

- La arquitectura puede:
  - Favorecer la **disponibilidad**
  - Favorecer el **rendimiento**
  - Favorecer la **seguridad**
  - Favorecer los **cambios**

Tipos de arquitecturas

- La arquitectura puede:
  - Para favorecer la **disponibilidad**, deben incluirse componentes redundantes de manera que sea posible reemplazarlos y actualizarlos sin detener el sistema
  - Favorecer el rendimiento
  - Favorecer la seguridad
  - Favorecer los cambios

Tipos de arquitecturas

- La arquitectura puede:
  - Favorecer la disponibilidad
  - Para favorecer el **rendimiento**, las operaciones críticas deben concentrarse en un número limitado de componentes, que se ubiquen preferiblemente en la misma localización
  - Favorecer la seguridad
  - Favorecer los cambios

Tipos de arquitecturas

- La arquitectura puede:
  - Favorecer la disponibilidad
  - Favorecer el rendimiento
  - Para favorecer la **seguridad**, los elementos a proteger (datos, procesos) deberían estar detrás de una serie de barreras según su criticidad
  - Favorecer los cambios

Tipos de arquitecturas

- La arquitectura puede:
  - Favorecer la disponibilidad
  - Favorecer el rendimiento
  - Favorecer la seguridad
  - Para favorecer los **cambios**, los componentes deben ser auto-contenidos, los productores de datos deben separarse de los consumidores, debe evitarse la compartición de estructuras

Tipos de arquitecturas

- El diseño de la arquitectura de un sistema es un **proceso creativo**
  - Las actividades a realizar dependen de la *experiencia*, *entorno tecnológico* y de *negocio*
- Es más adecuado pensar en el diseño de la arquitectura como una serie de **decisiones en lugar de** una secuencia de **actividades**
- Las decisiones de arquitectura son decisiones estructurales que afectan profundamente el sistema que se va a desarrollar

Tipos de arquitecturas

- El arquitecto/a (según sus influencias sociales, técnicas y de negocio) debe responder:
  - ¿Qué **organización es más adecuada** para garantizar los **requisitos no funcionales**?
  - ¿Hay una **arquitectura genérica/patrones** que puedan servir de “plantilla”?
  - ¿Cómo **evaluaremos** la arquitectura?
  - ¿Cómo **documentaremos** la arquitectura?

Tipos de arquitecturas

- El arquitecto/a (según sus influencias sociales, técnicas y de negocio) debe responder:
  - ¿Qué **organización es más adecuada** para garantizar los **requisitos no funcionales**?
    - ¿Cómo se van a **descomponer los componentes** en sub-componentes?
    - ¿Cómo se van a **distribuir físicamente** los componentes del sistema?
    - ¿Qué estrategia vamos a usar para **controlar/comunicar** los componentes?
  - ¿Hay una **arquitectura genérica/patrones** que puedan servir de “plantilla”?
  - ¿Cómo **documentaremos** la arquitectura?

Arquitectura en capas

Arquitectura en capas

Descripción	Organiza el sistema en *capas* Cada capa *agrupa funcionalidades* Cada capa *inferior* proporciona servicios a la capa *superior*
Aplicabilidad	Construcción de sistemas a partir de otros ya existentes División de responsabilidades por capas Se requieren varios niveles de seguridad Se quiere dar una visión centrada en la organización conceptual del sistema

Arquitectura en capas

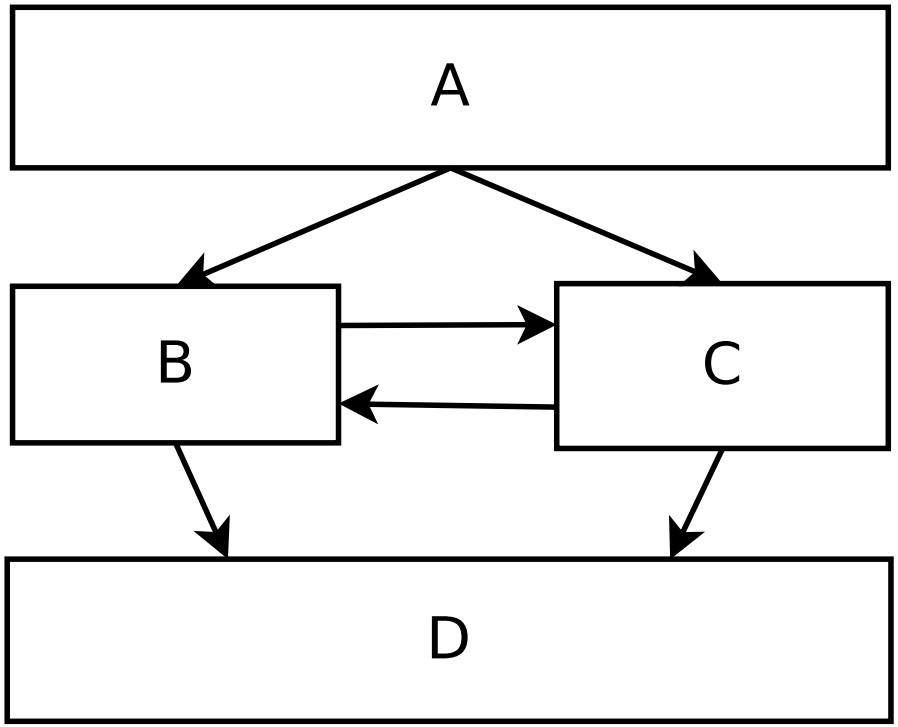
Descripción	Organiza el sistema en *capas* Cada capa *agrupa funcionalidades* Cada capa *inferior* proporciona servicios a la capa *superior*
Ventajas	Permite que aislar los cambios a nivel de capa, o a capas adyacentes Permite sustituir capas completas siempre que se mantenga su interfaz Permite introducir redundancia por niveles para incrementar la fiabilidad Permite fácilmente la portabilidad y el soporte multi-plataforma/protocolo

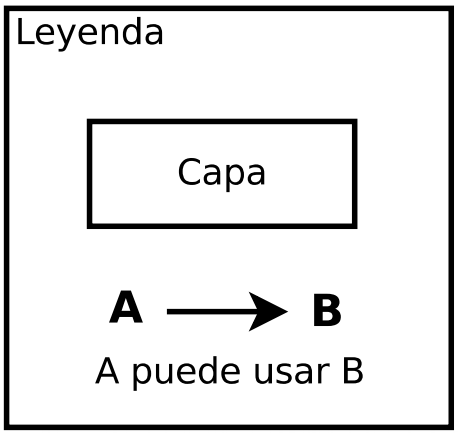
Arquitectura en capas

Descripción	Organiza el sistema en *capas* Cada capa *agrupa funcionalidades* Cada capa *inferior* proporciona servicios a la capa *superior*
Desventajas	Puede ser difícil realizar una división “limpia” en capas Puede ser necesaria comunicación entre capas no adyacentes Puede haber problemas de rendimiento debido a los múltiples niveles que debe atravesar una petición

Arquitectura en capas

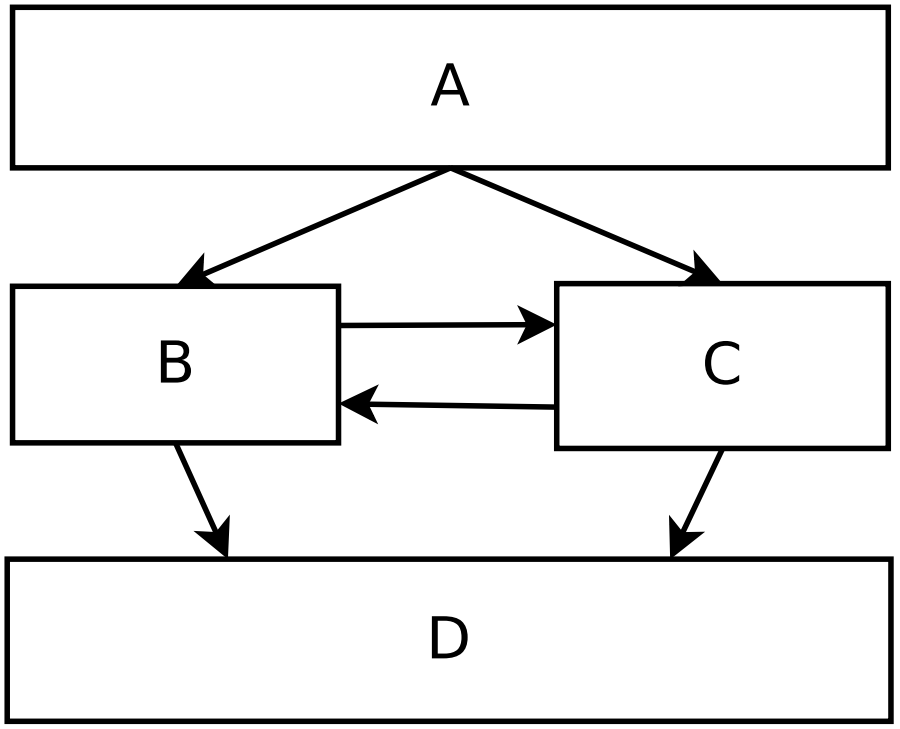
Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

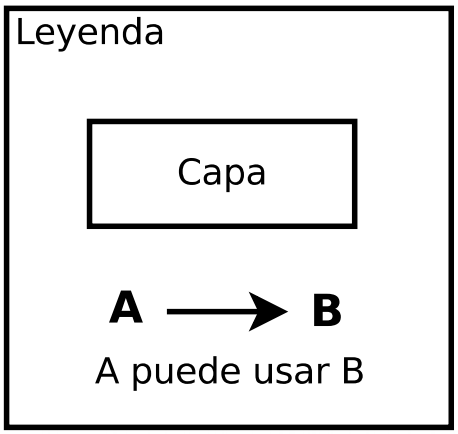




Arquitectura en capas

Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

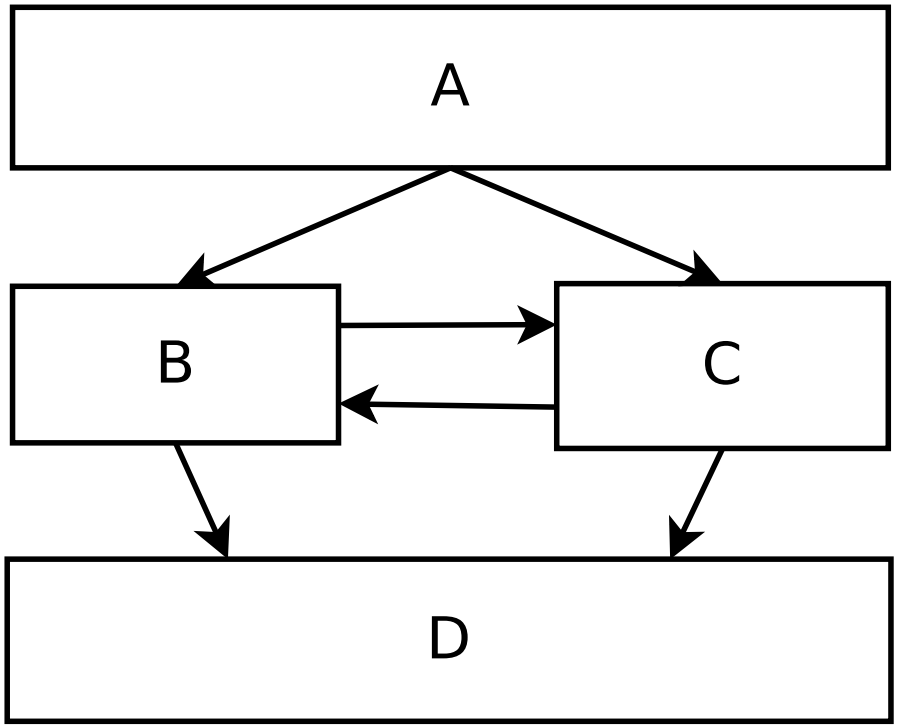


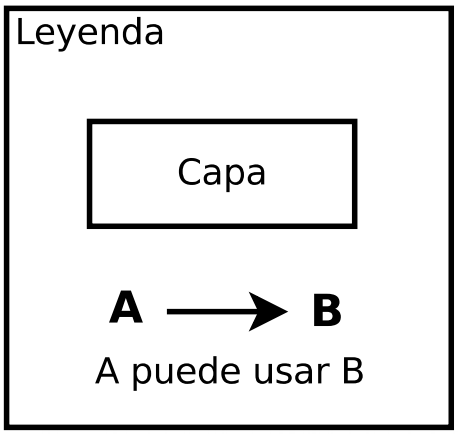


¿naturaleza de los elementos?

Arquitectura en capas

Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

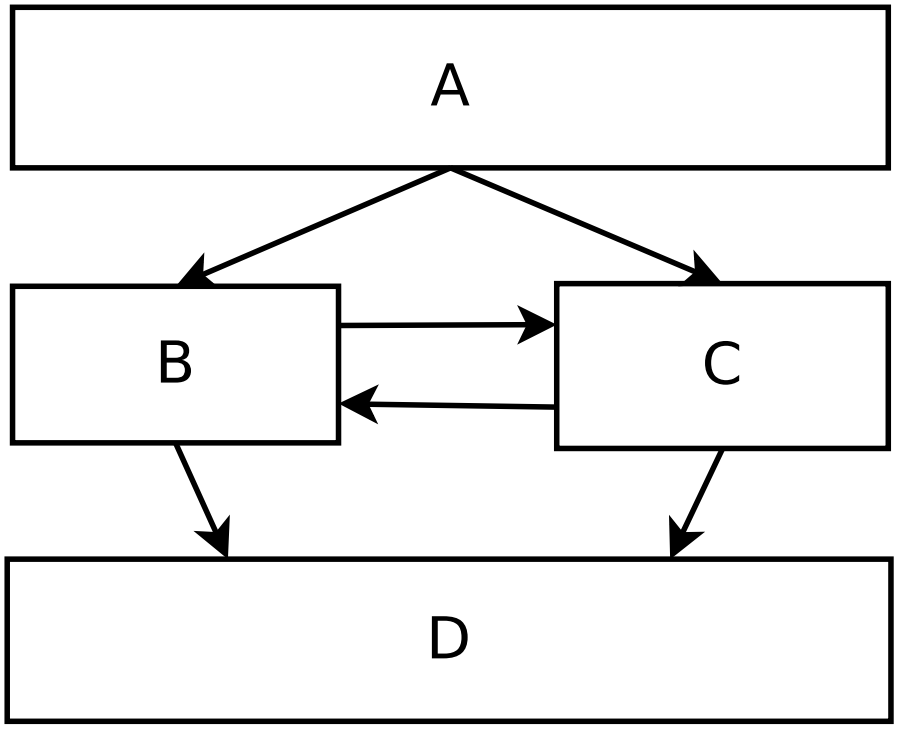


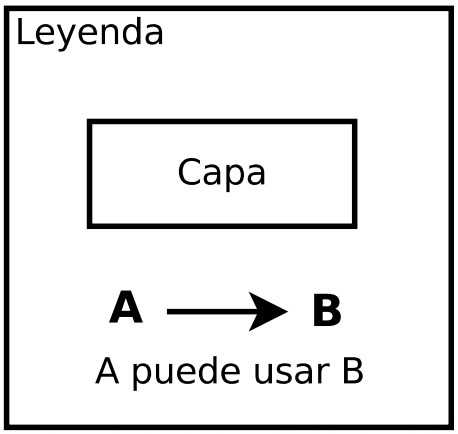


¿responsabilidades de los elementos?

Arquitectura en capas

Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

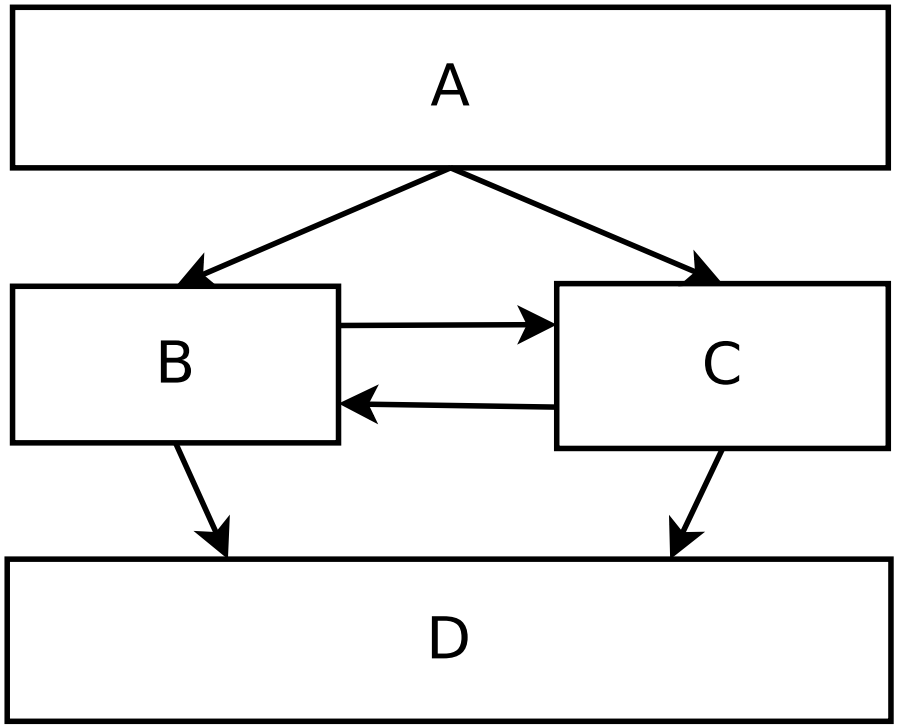


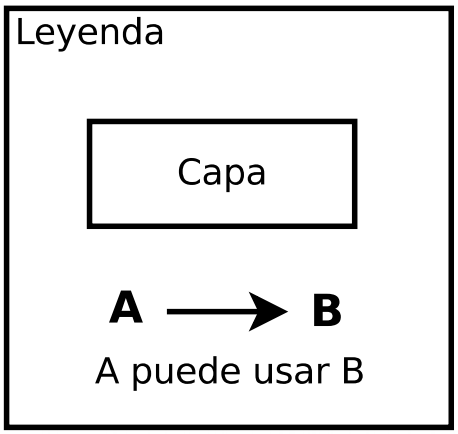


¿significado de las conexiones?

Arquitectura en capas

Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

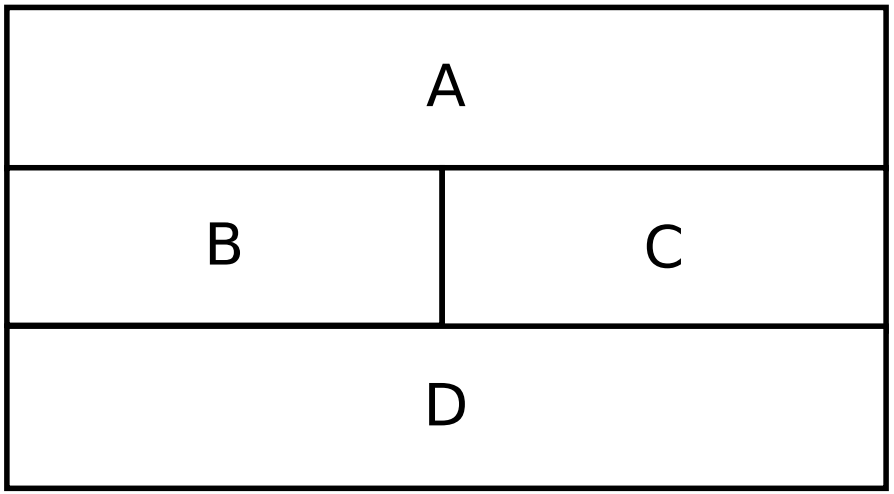


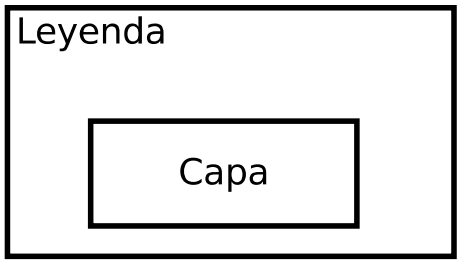


¿colocación de los elementos?

Arquitectura en capas

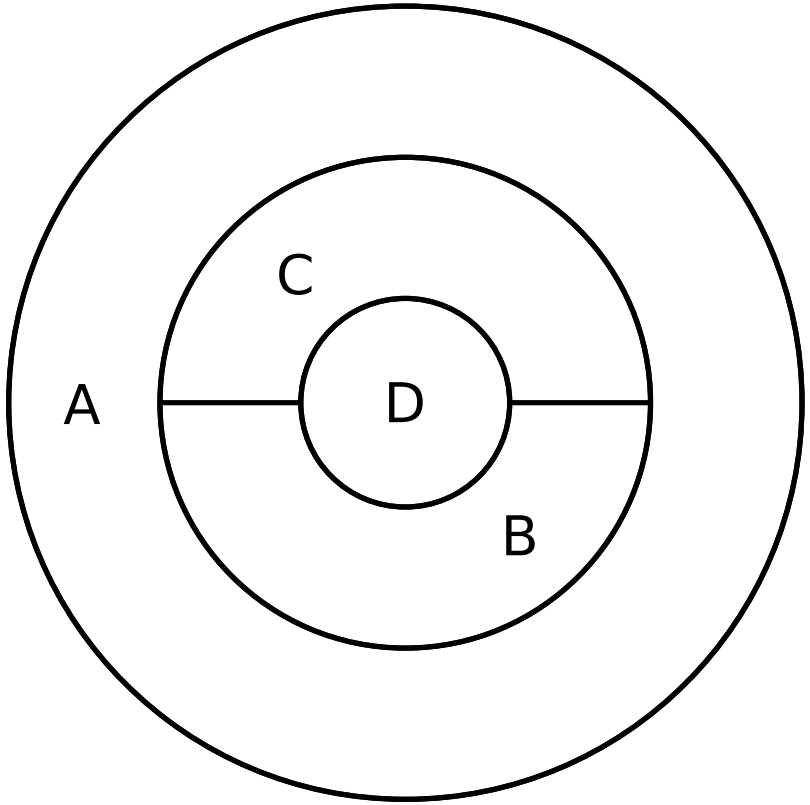
Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

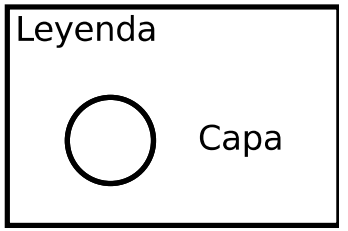




Arquitectura en capas

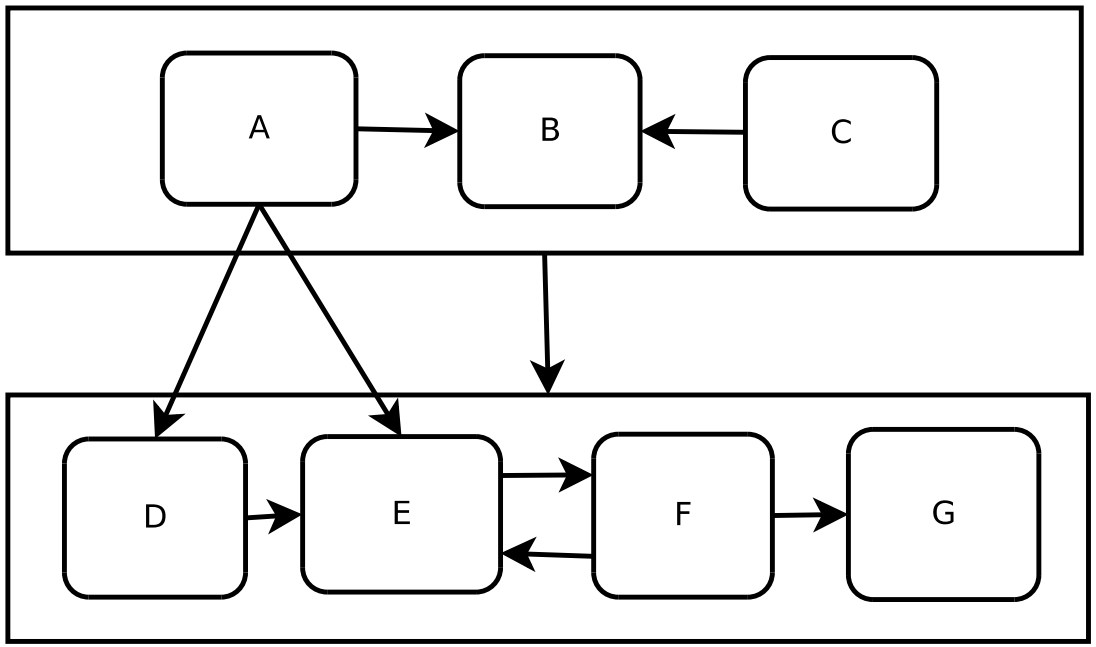
Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

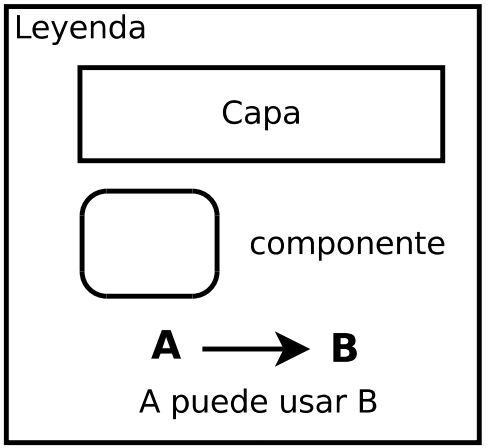




Arquitectura en capas

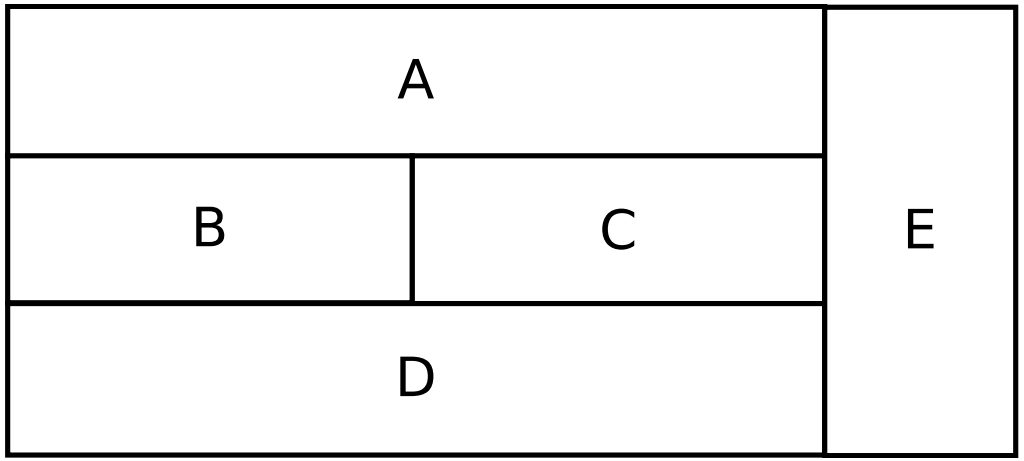
Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

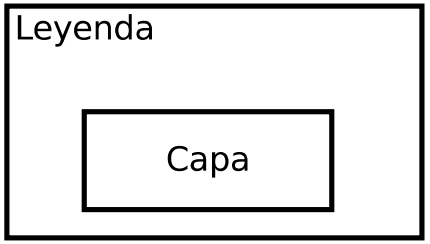




Arquitectura en capas

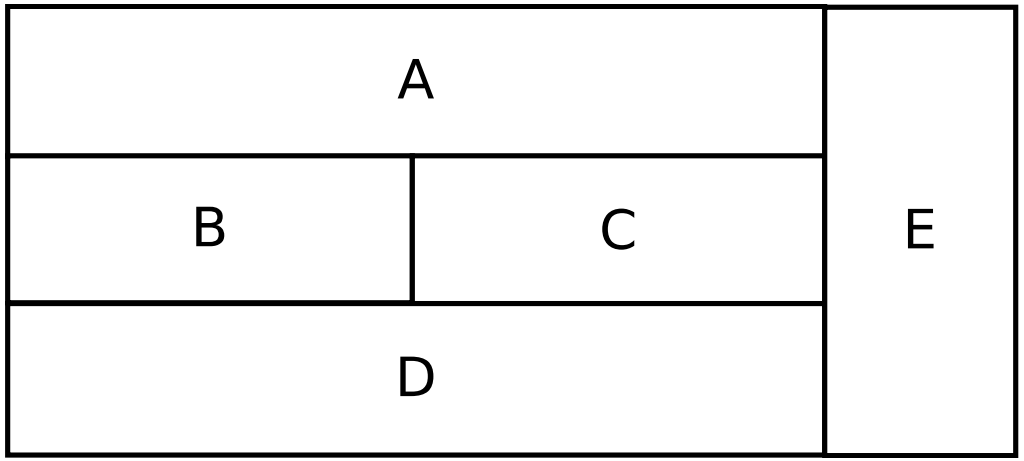
Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

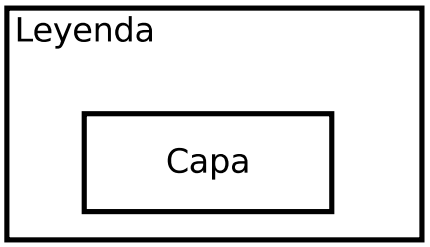




Arquitectura en capas

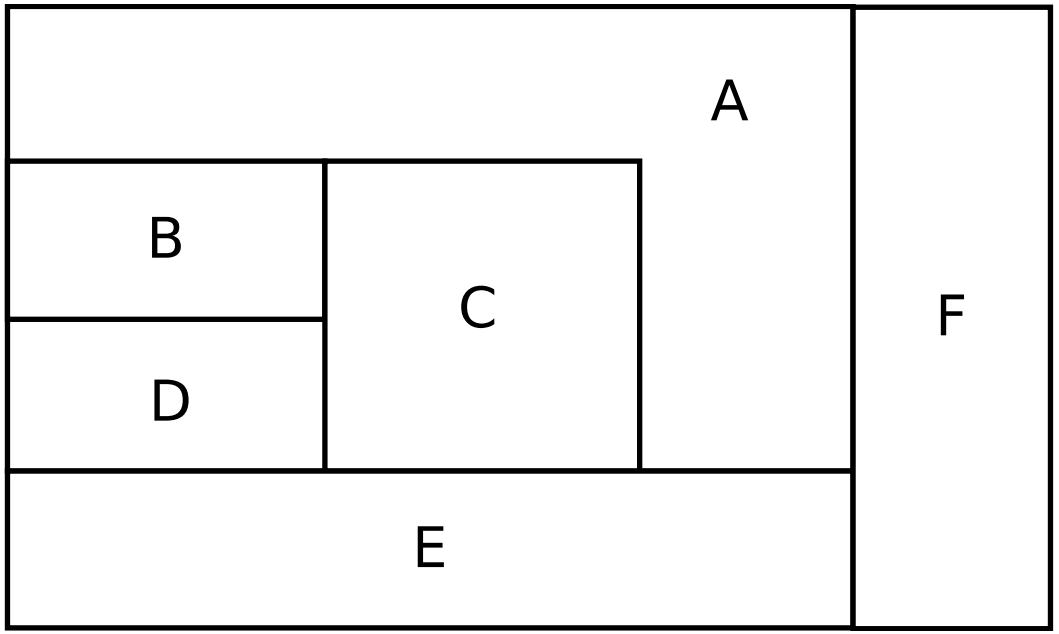
Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

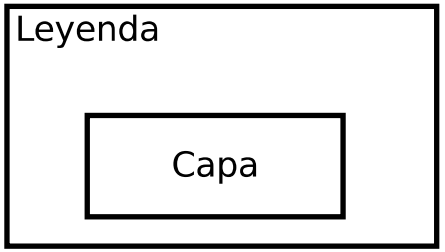




Arquitectura en capas

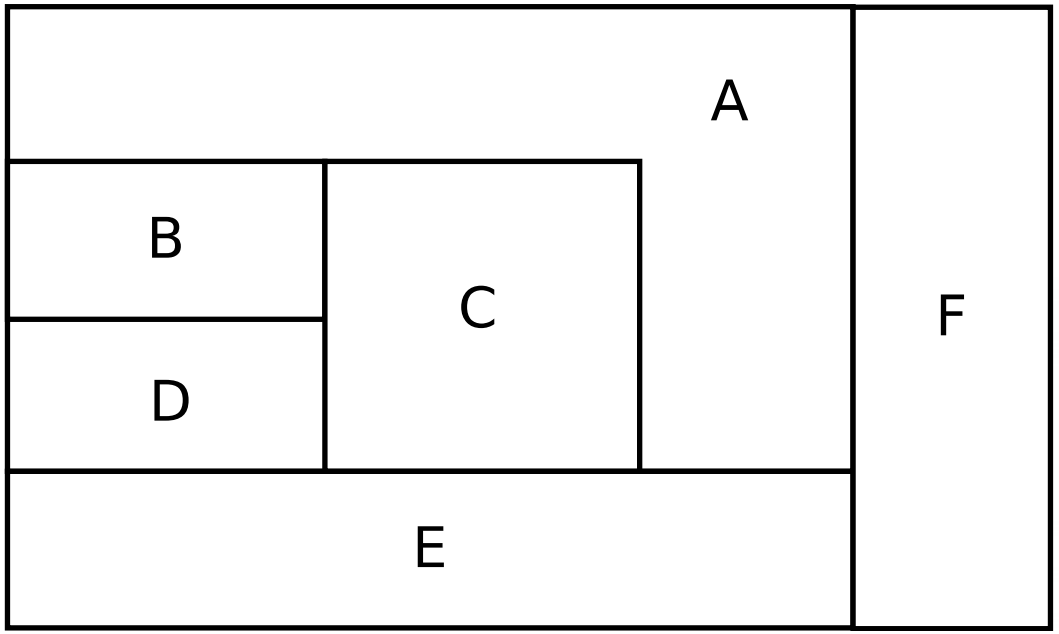
Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

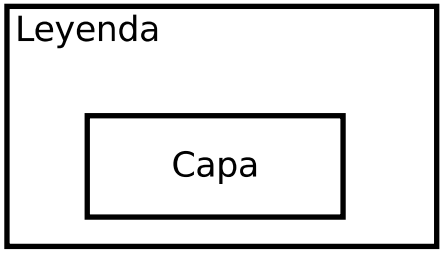


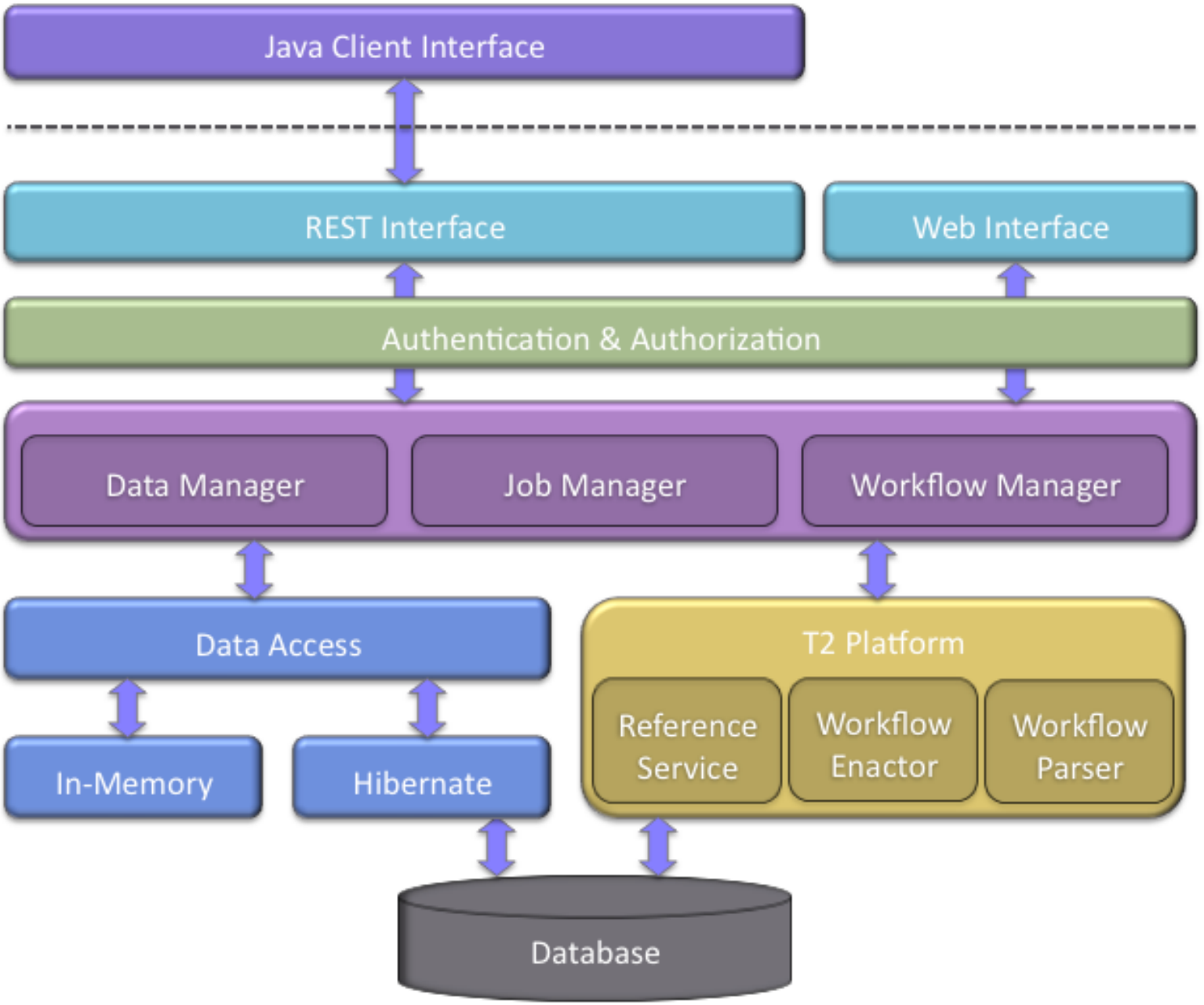


Arquitectura en capas

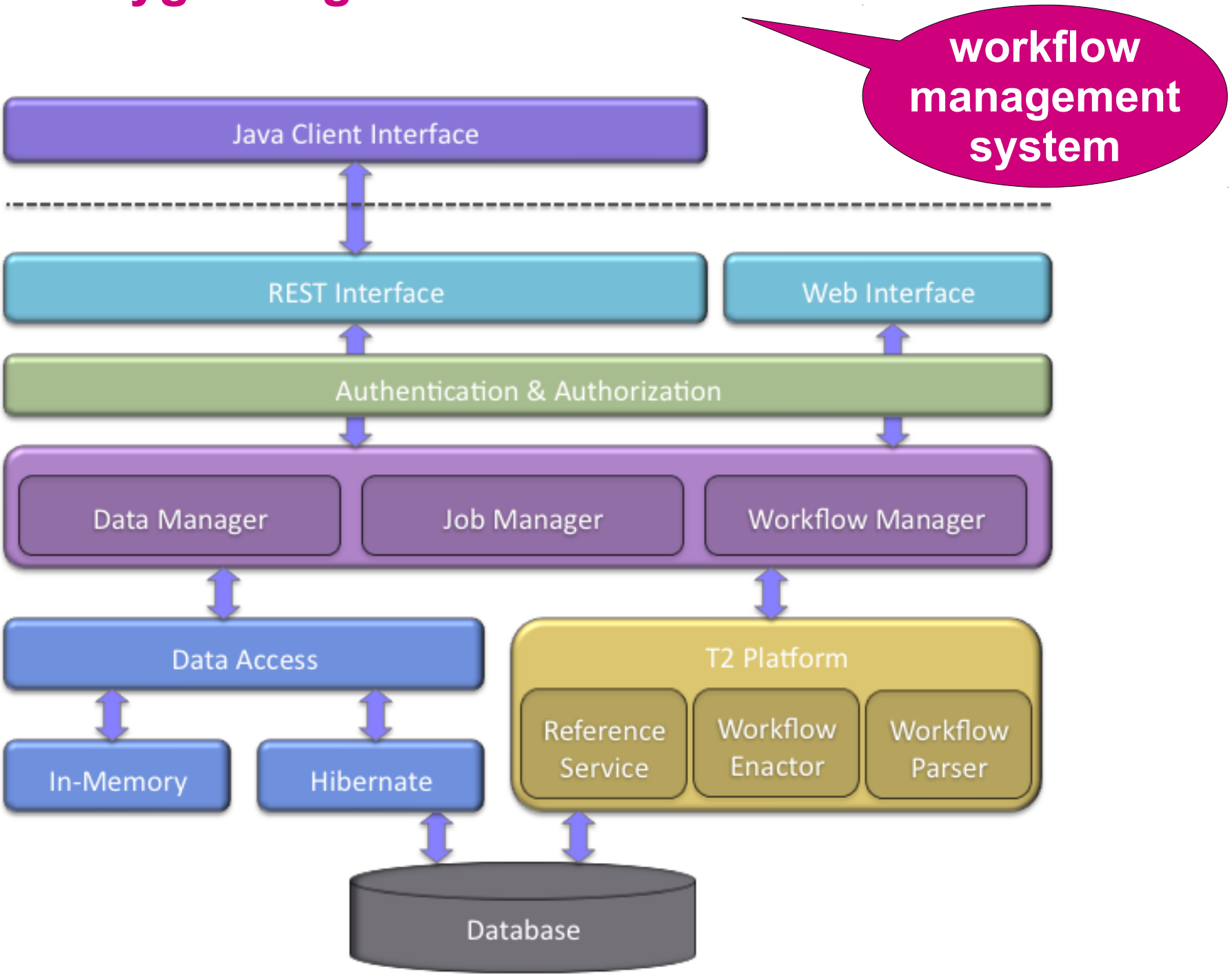
Descripción Organiza el sistema en *capas*
Cada capa *agrupa funcionalidades*
Cada capa *inferior* proporciona servicios a la capa *superior*

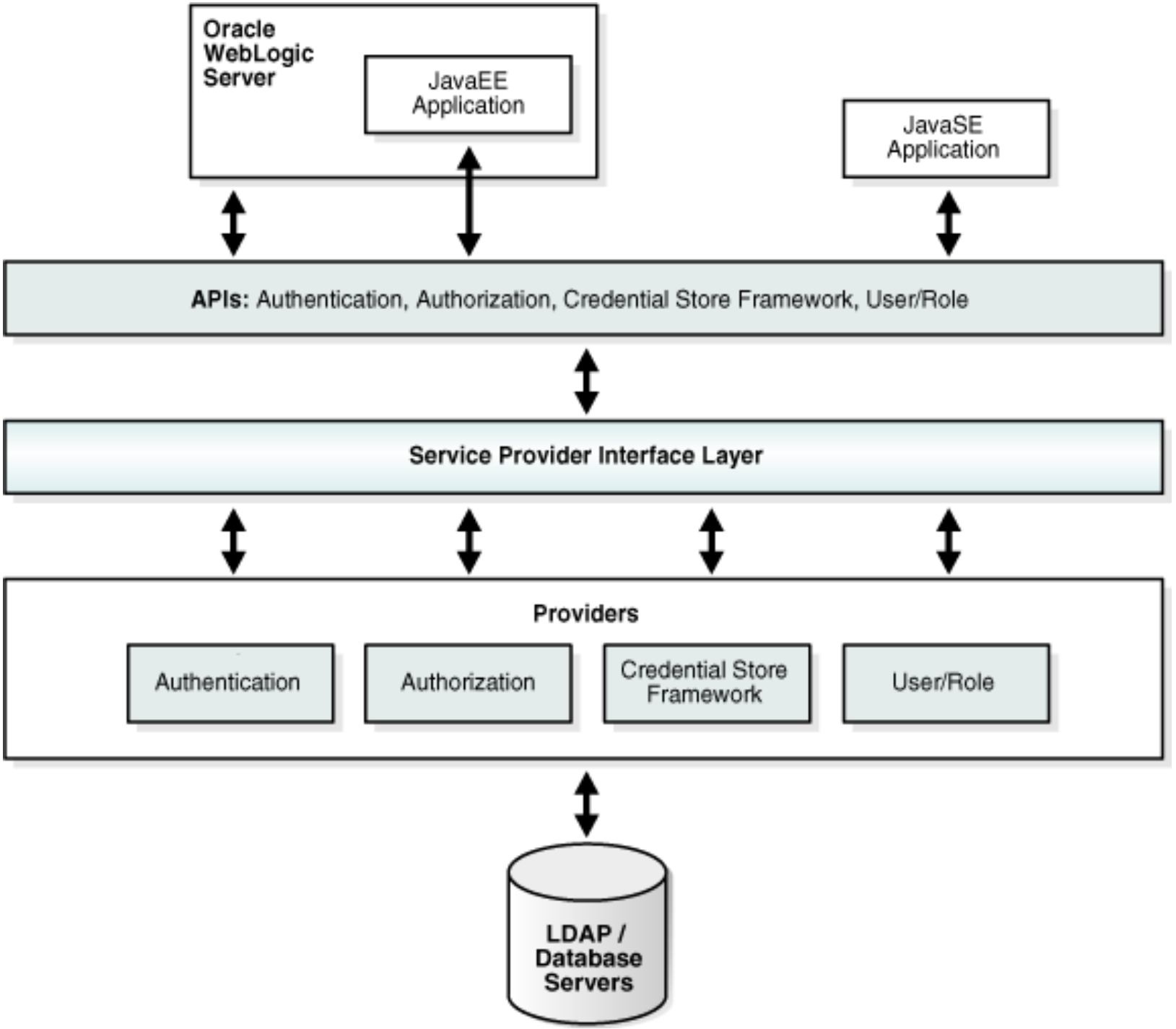




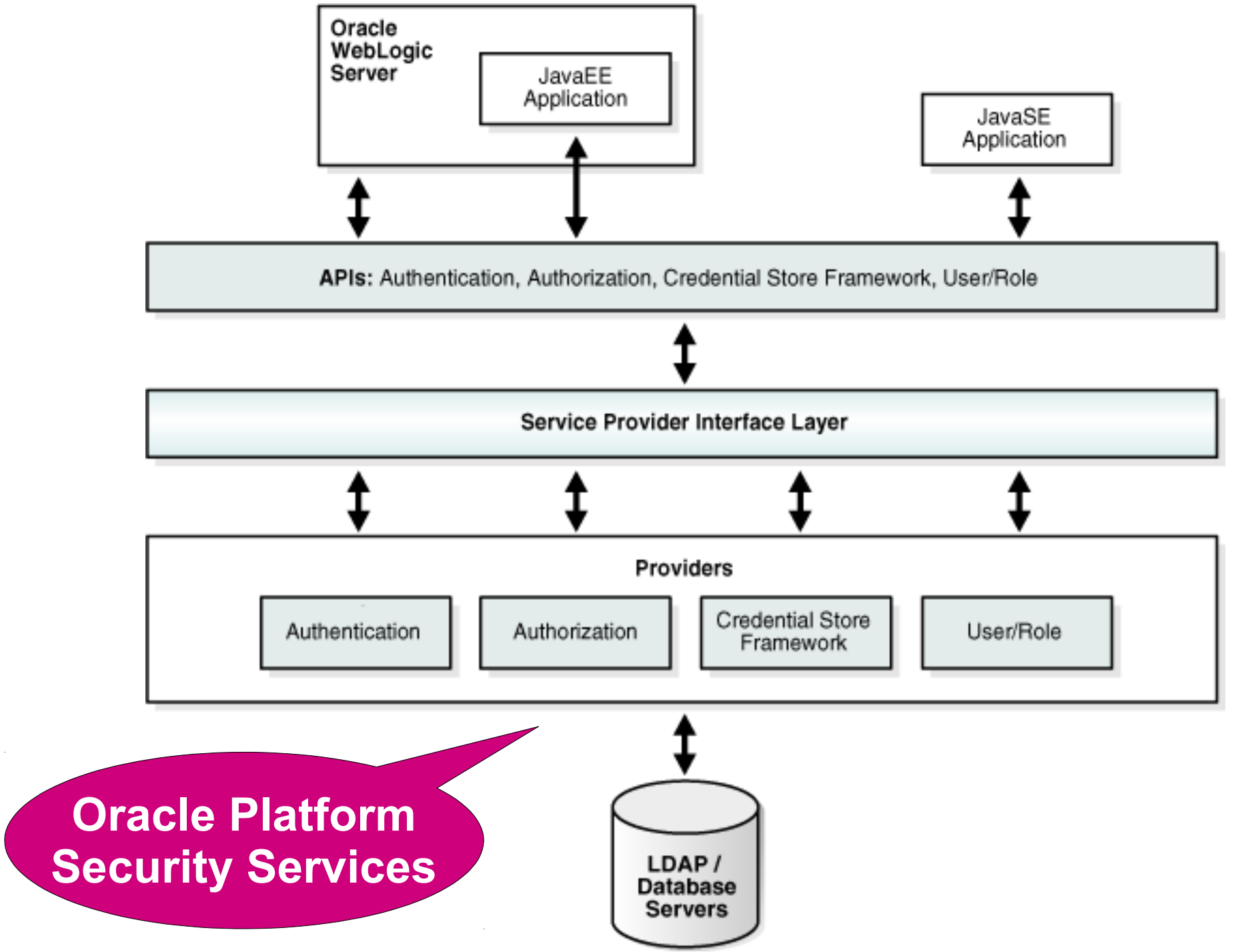

Java Client Interface
REST Interface
Web Interface
Authentication & Authorization
Data Manager
Job Manager
Workflow Manager
Data Access
T2 Platform
Reference Service
Workflow Enactor
Workflow Parser
In-Memory
Hibernate
Database

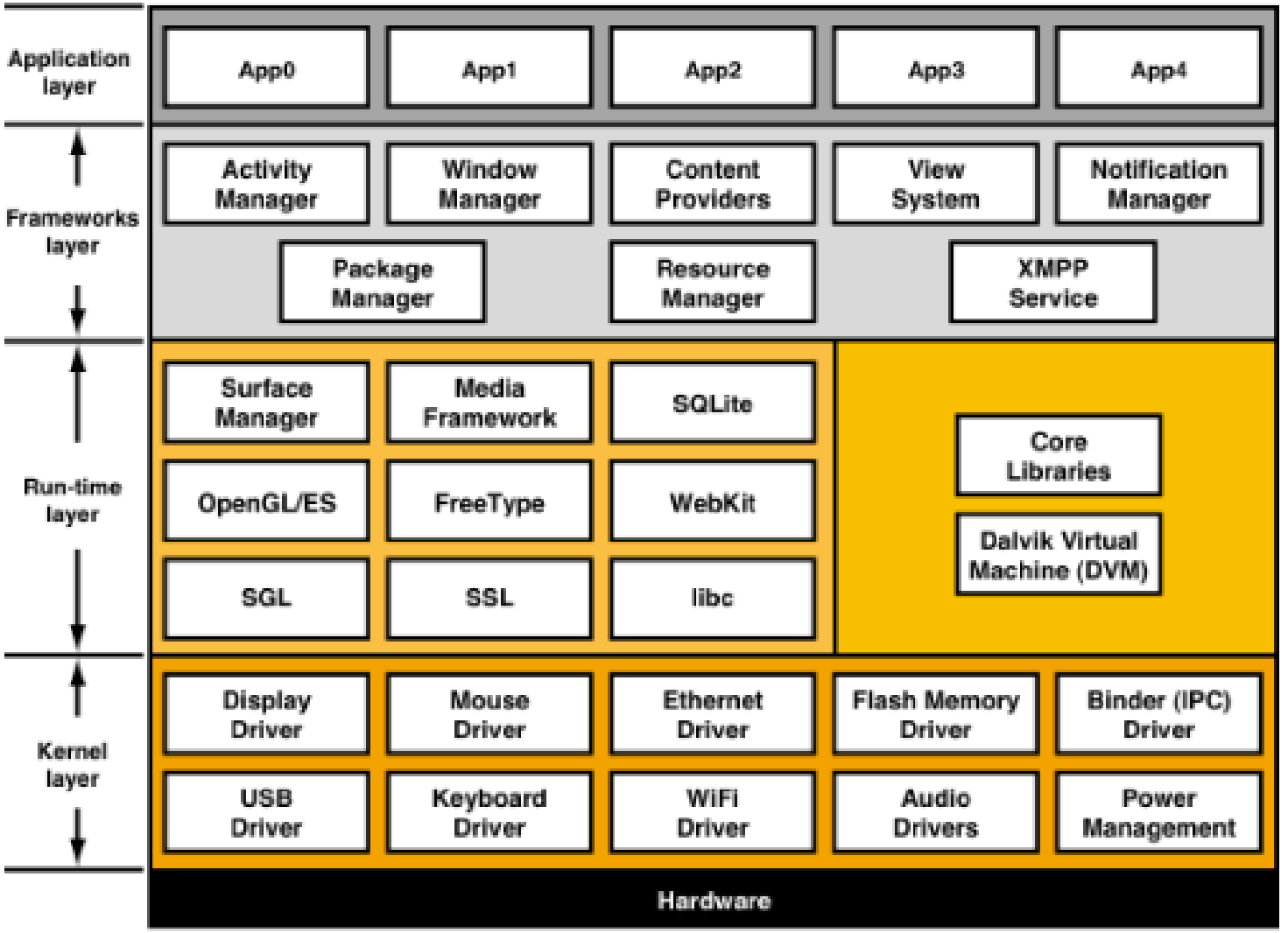
http://dev.mygrid.org.uk/taverna-server/architecture.html

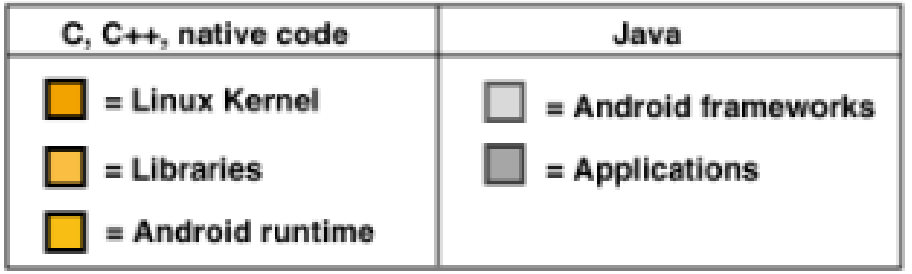



Oracle
WebLogic
Server
JavaEE
Application
JavaSE
Application
APIs: Authentication, Authorization, Credential Store Framework, User/Role
Service Provider Interface Layer
Providers
Authentication
Authorization
Credential Store
Framework
User/Role
LDAP /
Database
Servers

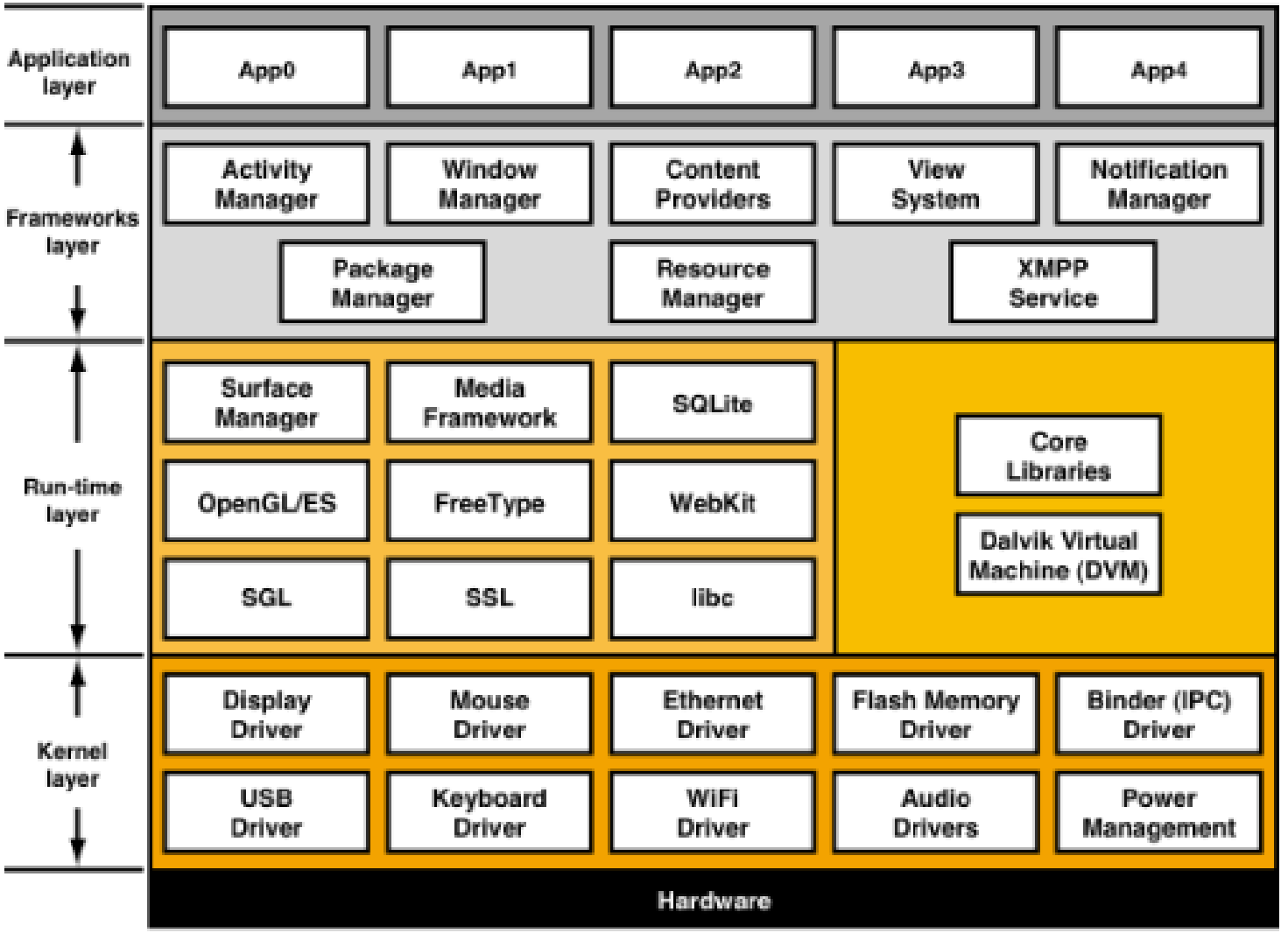
http://docs.oracle.com/cd/E21764_01/core.1111/e10043/underjps.htm



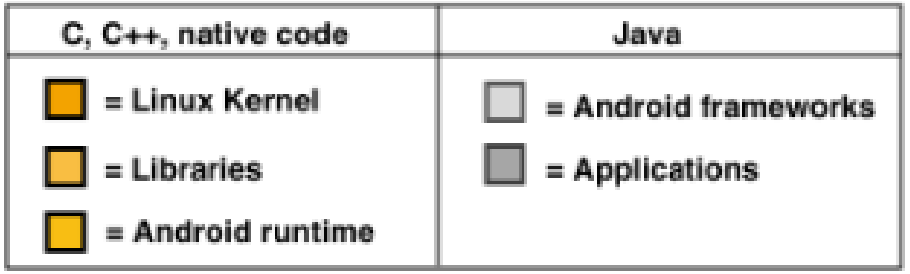

Application layer
App0
App1
App2
App3
App4
Frameworks layer
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Resource Manager
XMPP Service
Run-time layer
Surface Manager
Media Framework
SQLite
OpenGL/ES
FreeType
WebKit
SGL
SSL
libc
Core Libraries
Dalvik Virtual Machine (DVM)
Kernel layer
Display Driver
Mouse Driver
Ethernet Driver
Flash Memory Driver
Binder (IPC) Driver
USB Driver
Keyboard Driver
WiFi Driver
Audio Drivers
Power Management
Hardware


C, C++, native code
Java
= Linux Kernel
= Libraries
= Android runtime
= Android frameworks
= Applications

https://community.freescale.com/community/the-embedded-beat/blog/2010/05/24/android-makes-the-move-to-power-architecture-technology



Android



Arquitectura de repositorio

Arquitectura de repositorio

Descripción	Organiza el sistema en torno a un *almacenamiento de datos centralizado* El almacenamiento de datos es accesible a todos los componentes Los *componentes no interactúan* entre sí directamente, sino que lo hacen a través del repositorio de datos
Aplicabilidad	Construcción de sistemas que gestionan gran cantidad de datos persistentes Construcción de sistemas en los que la inserción/modificación/eliminación de información desencadena acciones Se quiere dar una visión centrada en la gestión de datos

Arquitectura de repositorio

Descripción	Organiza el sistema en torno a un *almacenamiento de datos centralizado* El almacenamiento de datos es accesible a todos los componentes Los *componentes no interactúan* entre sí directamente, sino que lo hacen a través del repositorio de datos
Aplicabilidad	Construcción de sistemas que gestionan gran cantidad de datos persistentes Construcción de sistemas en los que la inserción/modificación/eliminación de información desencadena acciones Se quiere dar una visión centrada en la gestión de datos

productores/ consumidores

Arquitectura de repositorio

Descripción	Organiza el sistema en torno a un *almacenamiento de datos centralizado* El almacenamiento de datos es accesible a todos los componentes Los *componentes no interactúan* entre sí directamente, sino que lo hacen a través del repositorio de datos
Ventajas	Los componentes pueden ser completamente independientes Los cambios en los datos son inmediatamente visibles a los demás Toda la información se puede gestionar uniforme y consistentemente (por ejemplo, copias de seguridad)

Arquitectura de repositorio

Descripción

Organiza el sistema en torno a un *almacenamiento de datos centralizado*

El almacenamiento de datos es accesible a todos los componentes

Los *componentes no interactúan* entre sí directamente, sino que lo hacen a través del repositorio de datos

Ventajas

Los componentes pueden ser completamente independientes

Los cambios en los datos son inmediatamente visibles a los demás

Toda la información se puede gestionar uniforme y consistentemente (por ejemplo, copias de seguridad)

perfecto para división del trabajo

Arquitectura de repositorio

Descripción	Organiza el sistema en torno a un *almacenamiento de datos centralizado* El almacenamiento de datos es accesible a todos los componentes Los *componentes no interactúan* entre sí directamente, sino que lo hacen a través del repositorio de datos
Ventajas	Los componentes pueden ser completamente independientes Los cambios en los datos son inmediatamente visibles a los demás Toda la información se puede gestionar uniforme y consistentemente (por ejemplo, copias de seguridad)

no hay necesidad de transmitir datos

Arquitectura de repositorio

Descripción	Organiza el sistema en torno a un *almacenamiento de datos centralizado* El almacenamiento de datos es accesible a todos los componentes Los *componentes no interactúan* entre sí directamente, sino que lo hacen a través del repositorio de datos
Desventajas	El repositorio es un punto único de fallo, un problema en el repositorio afecta a todos los componentes Modelo de datos común Problemática distribución del repositorio Eficiencia crítica de acceso al repositorio

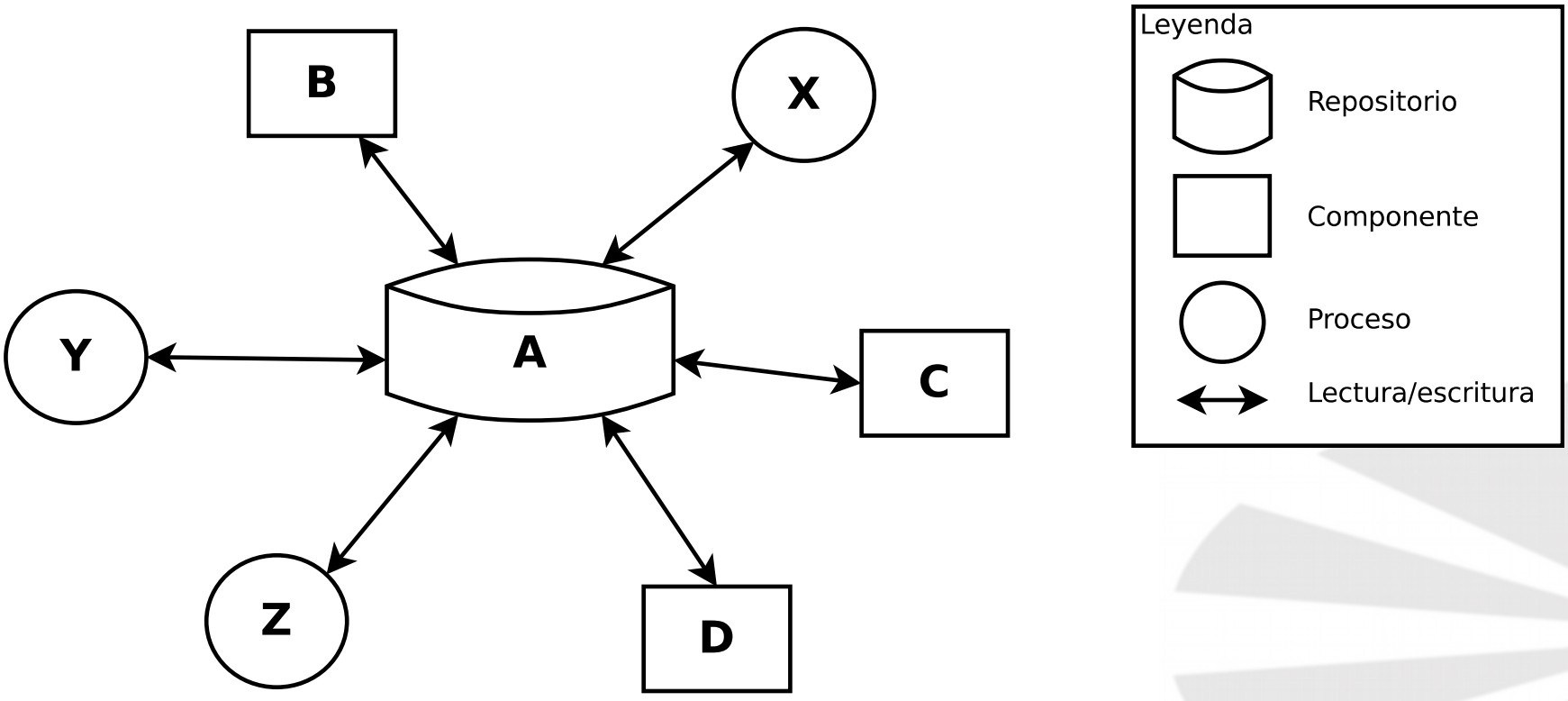
Arquitectura de repositorio

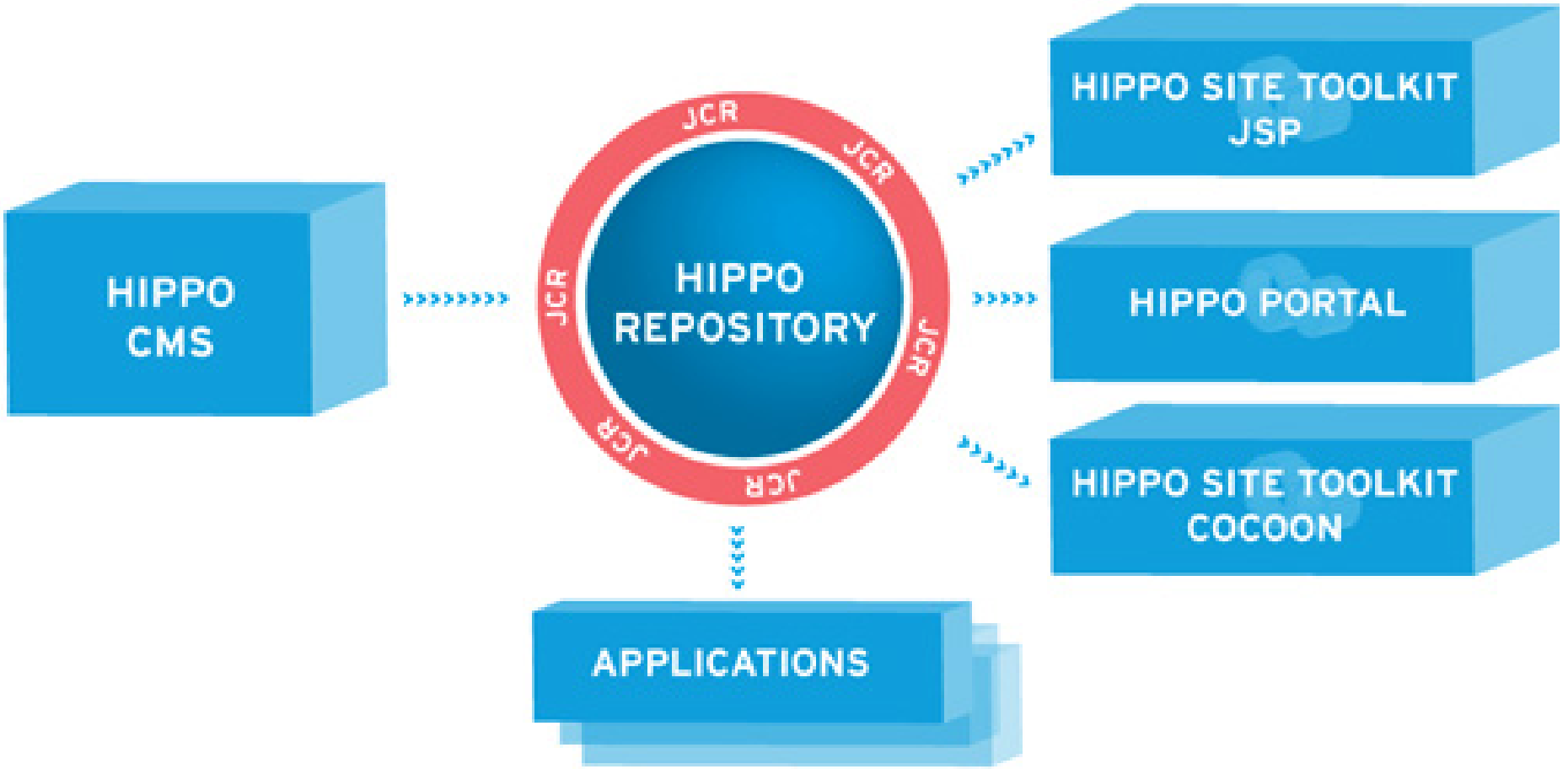
Descripción

Organiza el sistema en torno a un *almacenamiento de datos centralizado*

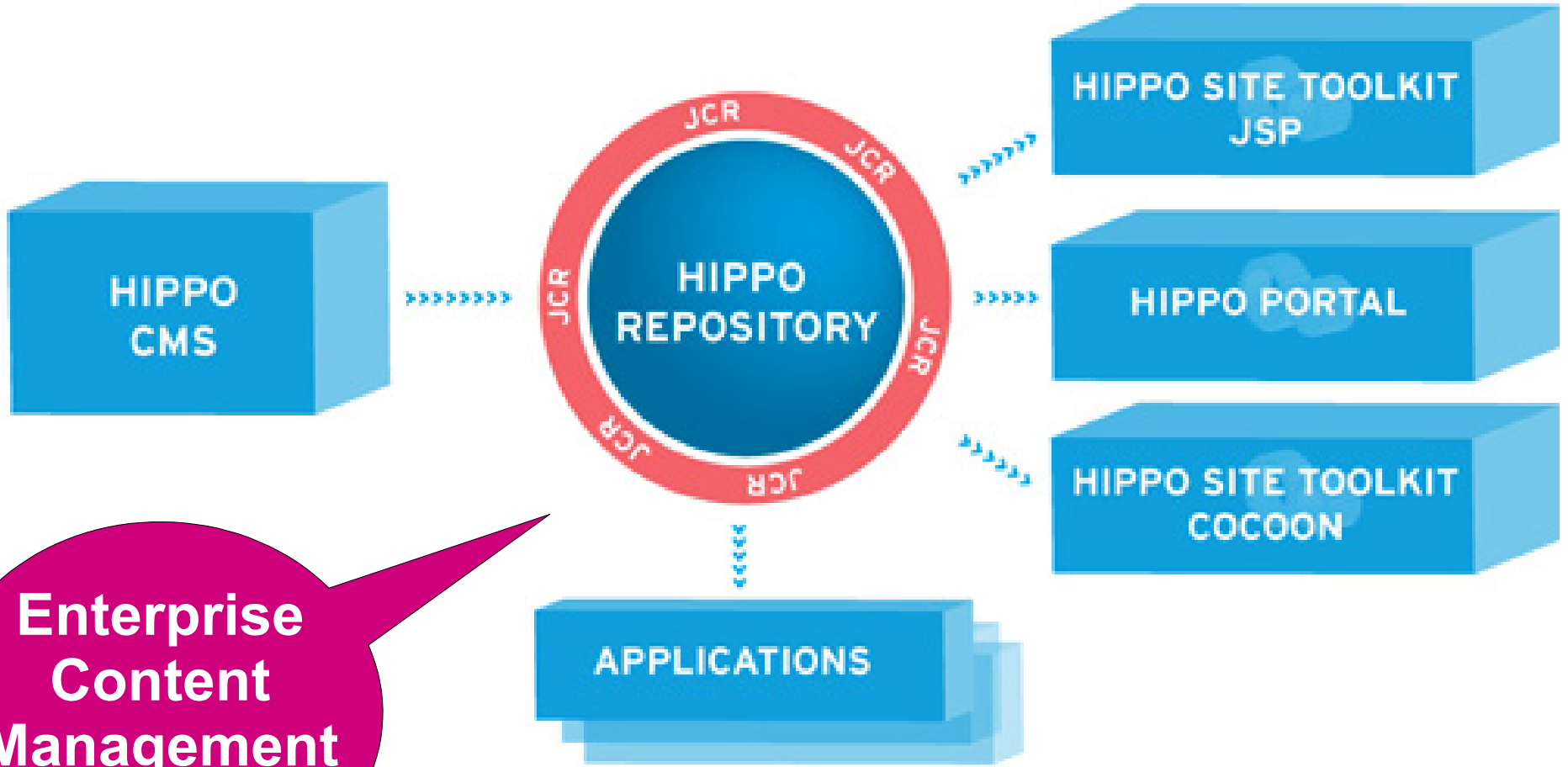
El almacenamiento de datos es accesible a todos los componentes

Los *componentes no interactúan* entre sí, sino que lo hacen a través del repositorio de datos

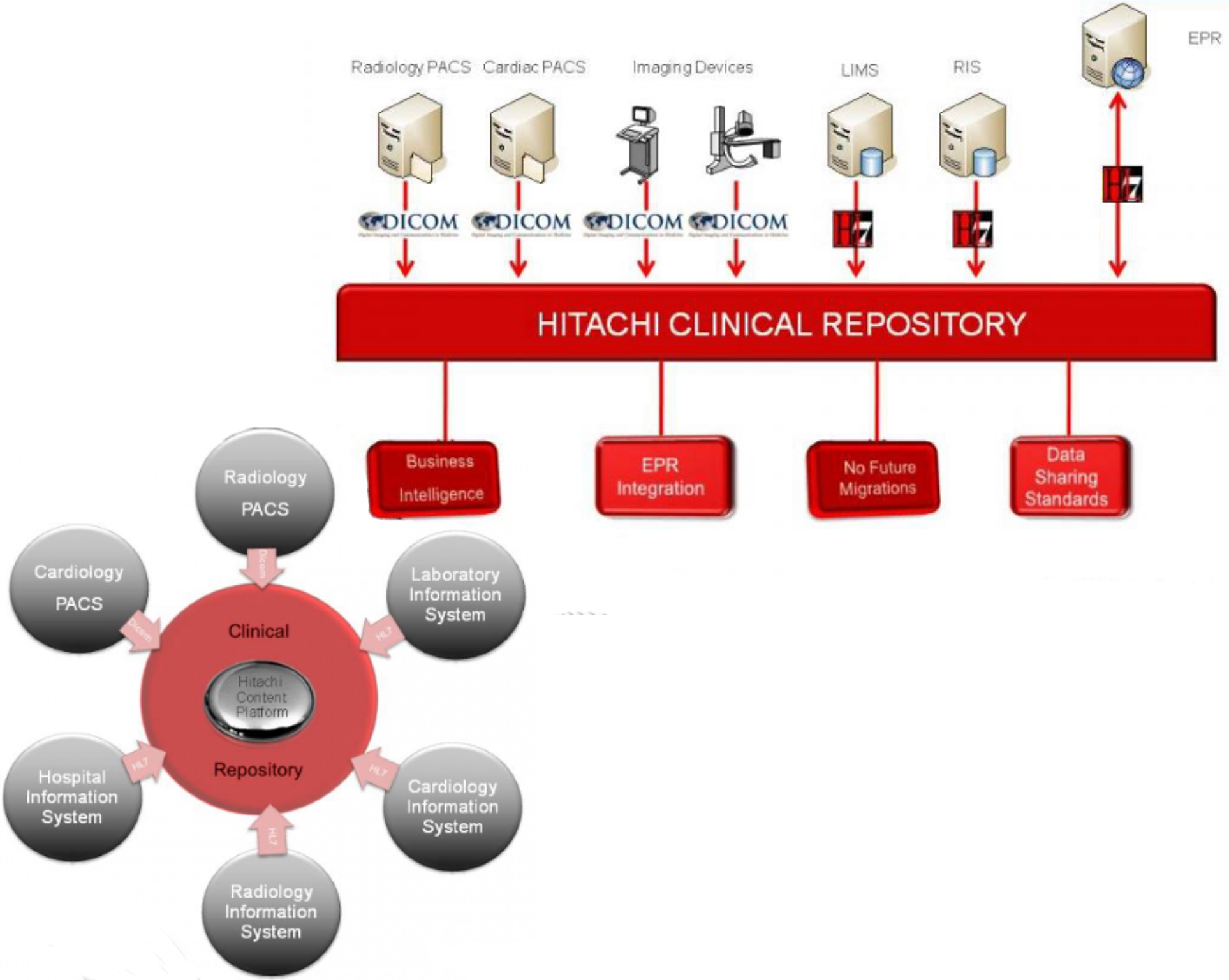



HIPPO CMS
JCR
JCR
JCR
JCR
JCR
HIPPO REPOSITORY
HIPPO SITE TOOLKIT JSP
HIPPO PORTAL
HIPPO SITE TOOLKIT COCOON
APPLICATIONS

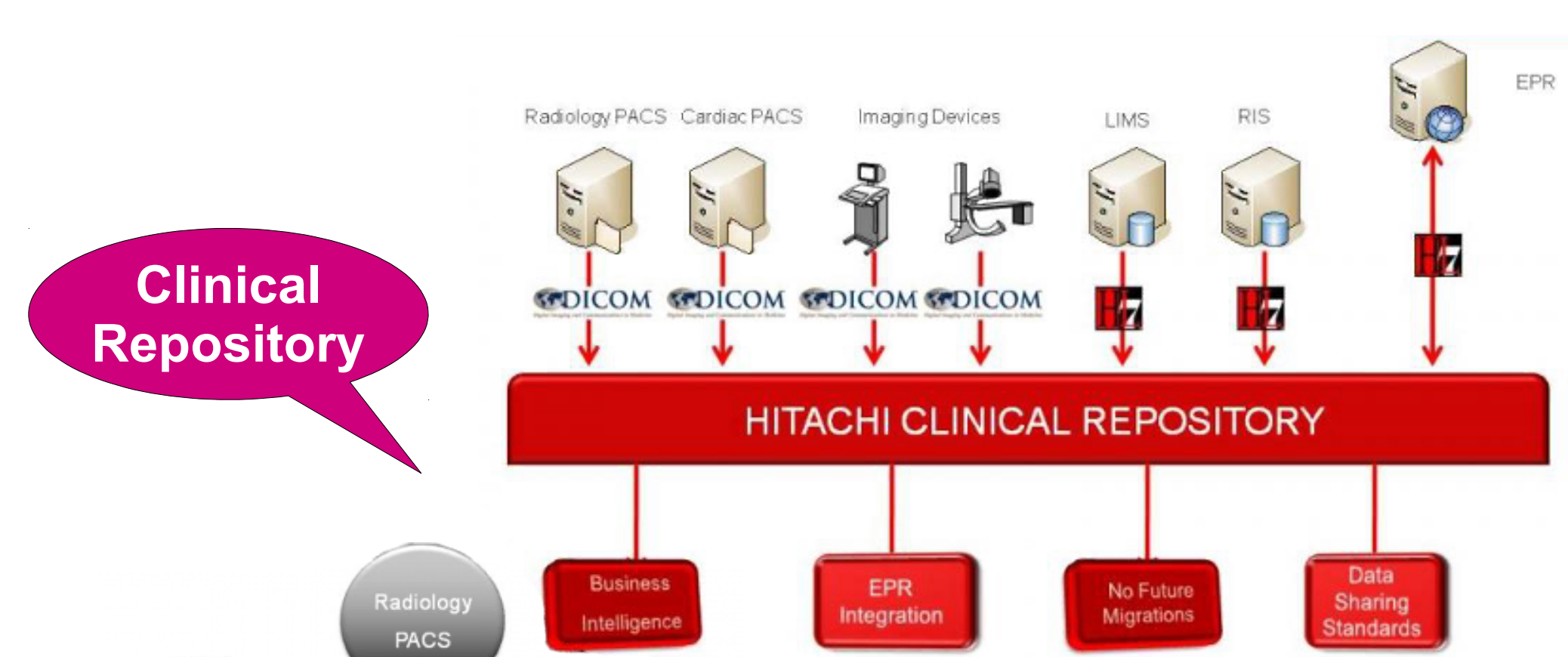
http://www.cmswire.com/cms/enterprise-cms/open-source-hippo-cms-70-gets-revamped-core-003822.php

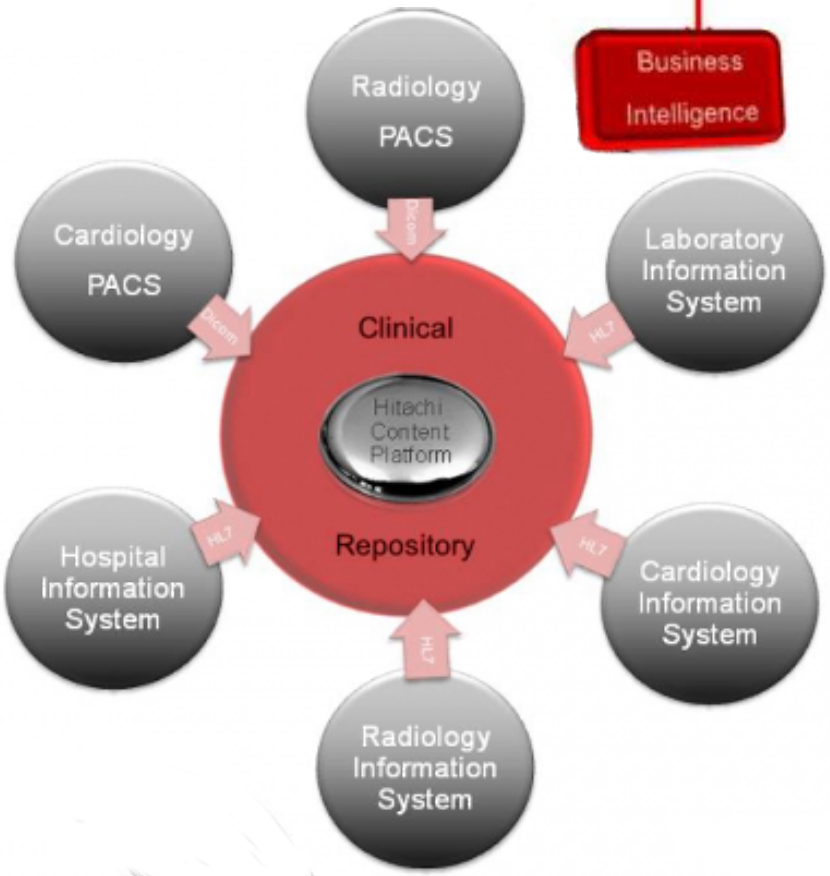


Enterprise Content Management System

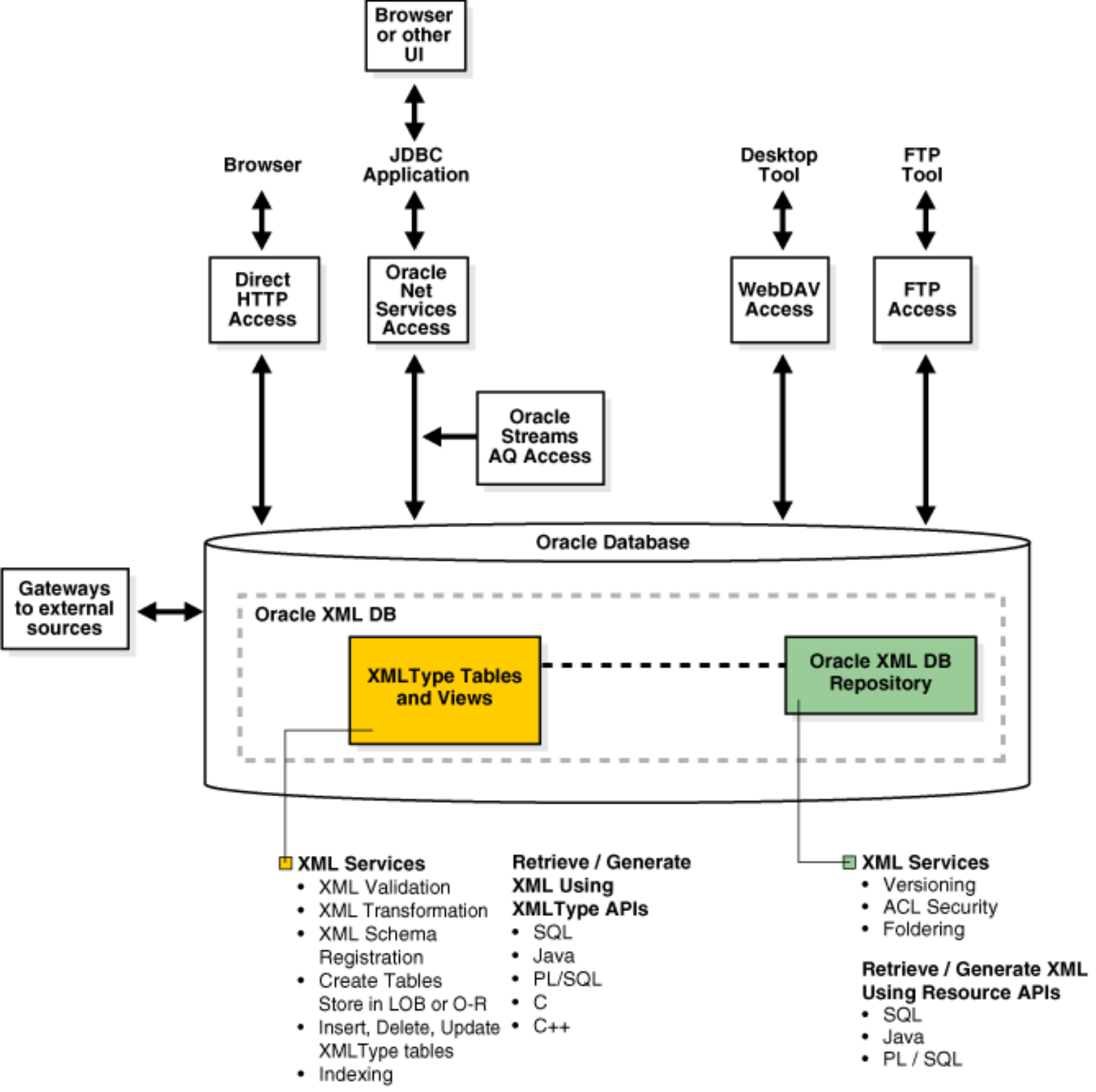

Radiology PACS
Cardiac PACS
Imaging Devices
LIMS
RIS
EPR
DICOM
DICOM
DICOM
DICOM
HL7
HL7
HL7
HITACHI CLINICAL REPOSITORY
Business Intelligence
EPR Integration
No Future Migrations
Data Sharing Standards
Radiology PACS
Cardiology PACS
Laboratory Information System
Clinical
Hitachi Content Platform
Repository
Hospital Information System
Cardiology Information System
Radiology Information System
Dicom
Dicom
HL7
HL7
HL7

Clinical Repository

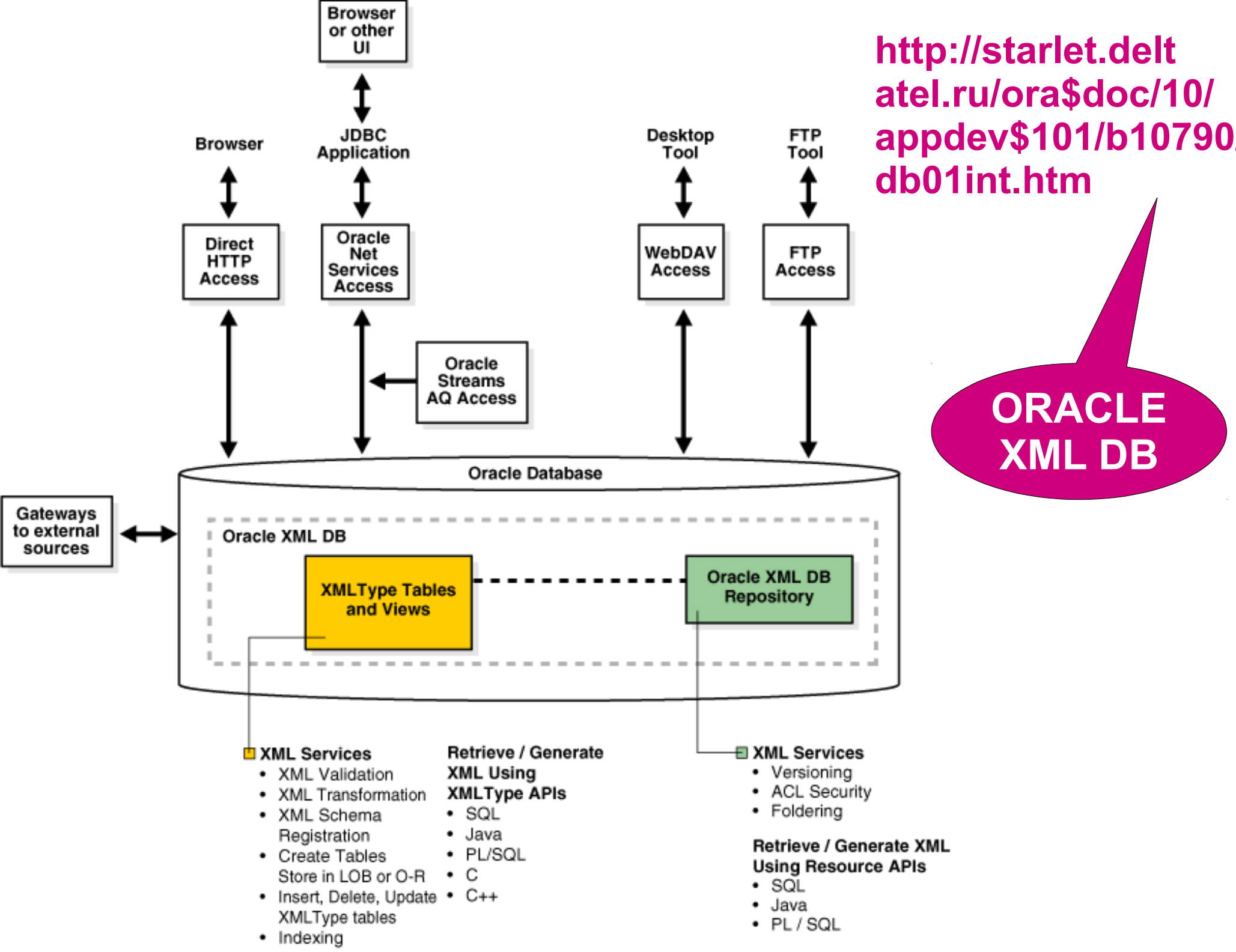




http://www.whospital.com/Store/SuiteStorepage.aspx?SuiteGuid=097ef4a4-3205-4f5d-95c3-021c60145aef


Browser or other UI
Browser
JDBC Application
Desktop Tool
FTP Tool
Direct HTTP Access
Oracle Net Services Access
WebDAV Access
FTP Access
Oracle Streams AQ Access
Oracle Database
Gateways to external sources
Oracle XML DB
XMLType Tables and Views
Oracle XML DB Repository
XML Services
• XML Validation
• XML Transformation
• XML Schema Registration
• Create Tables Store in LOB or O-R
• Insert, Delete, Update XMLType tables
• Indexing
Retrieve / Generate XML Using XMLType APIs
• SQL
• Java
• PL/SQL
• C
• C++
XML Services
• Versioning
• ACL Security
• Foldering
Retrieve / Generate XML Using Resource APIs
• SQL
• Java
• PL / SQL

http://starlet.deltatel.ru/ora$doc/10/appdev$101/b10790/xdb01int.htm



ORACLE XML DB

Arquitectura *pipe & filter*

Arquitectura *pipe & filter*

Descripción	Organiza el procesado de los datos en un sistema en *pasos discretos* Cada paso es realizado por un *componente diferente* (*filter*) Los *datos fluyen* de un componente a otro (*pipe*)
Aplicabilidad	Construcción de sistemas dedicados al procesado de información, donde la entrada se procesa por etapas para producir la salida deseada
Nomenclatura	Arquitectura basada en componentes Arquitectura *pipeline*

Arquitectura *pipe & filter*

Descripción	Organiza el procesado de los datos en un sistema en *pasos discretos* Cada paso es realizado por un *componente diferente* (*filter*) Los *datos fluyen* de un componente a otro (*pipe*)
Ventajas	Facilidad de comprensión del procesado Reutilización de componentes Correspondencia con flujos de negocio Facilidad de evolución (incorporación de etapas, sustitución de etapas) Los distintos pasos pueden ejecutarse secuencial o concurrentemente

Arquitectura *pipe & filter*

Descripción	Organiza el procesado de los datos en un sistema en *pasos discretos* Cada paso es realizado por un *componente diferente* (*filter*) Los *datos fluyen* de un componente a otro (*pipe*)
Ventajas	Facilidad de comprensión del procesado Reutilización de componentes Correspondencia con flujos de negocio Facilidad de evolución (incorporación de etapas, sustitución de etapas) Los distintos pasos pueden ejecutarse secuencial o concurrentemente

hoy, muy frecuente en embebidos

Arquitectura *pipe & filter*

Descripción	Organiza el procesado de los datos en un sistema en *pasos discretos* Cada paso es realizado por un *componente diferente* (*filter*) Los *datos fluyen* de un componente a otro (*pipe*)
Desventajas	El formato y modo de transferencia de los datos debe acordarse entre cada par de componentes Puede ser difícil reusar componentes si el formato de datos no es compatible

Arquitectura *pipe & filter*

Descripción	Organiza el procesado de los datos en un sistema en *pasos discretos* Cada paso es realizado por un *componente diferente* (*filter*) Los *datos fluyen* de un componente a otro (*pipe*)
Desventajas	El formato y modo de transferencia de los datos debe acordarse entre cada par de componentes Puede ser difícil reusar componentes si el formato de datos no es compatible

cada componente realiza un procesado a la entrada y otro a la salida

Arquitectura *pipe & filter*

Descripción	Organiza el procesado de los datos en un sistema en *pasos discretos* Cada paso es realizado por un *componente diferente* (*filter*) Los *datos fluyen* de un componente a otro (*pipe*)
Desventajas	El formato y modo de transferencia de los datos debe acordarse entre cada par de componentes Puede ser difícil reusar componentes si el formato de datos no es compatible

puede suponer una barrera a la eficiencia

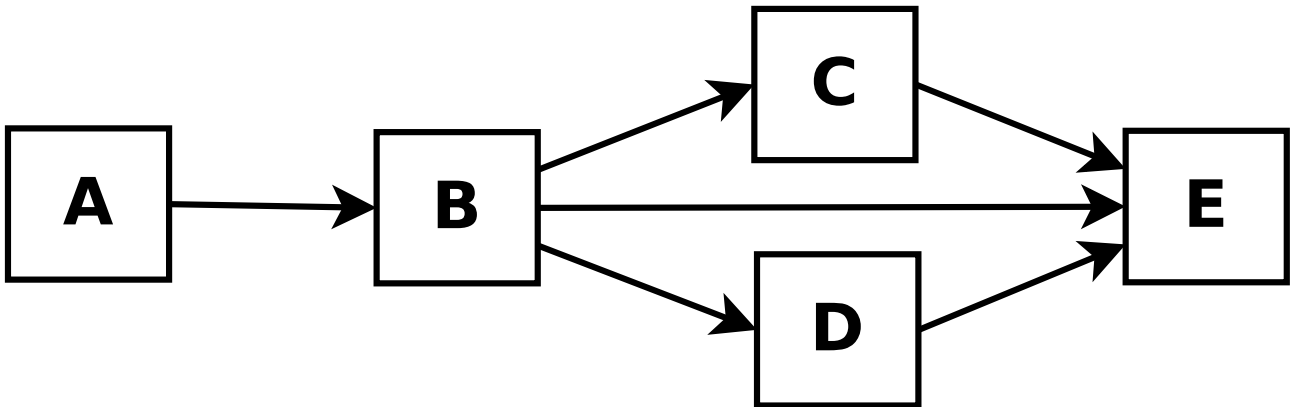
Arquitectura *pipe & filter*

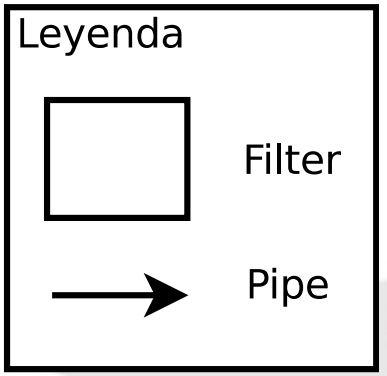
Descripción	Organiza el procesado de los datos en un sistema en *pasos discretos* Cada paso es realizado por un *componente diferente* (*filter*) Los *datos fluyen* de un componente a otro (*pipe*)
Desventajas	El formato y modo de transferencia de los datos debe acordarse entre cada par de componentes Puede ser difícil reusar componentes si el formato de datos no es compatible

no es adecuado para sistemas interactivos

Arquitectura *pipe & filter*

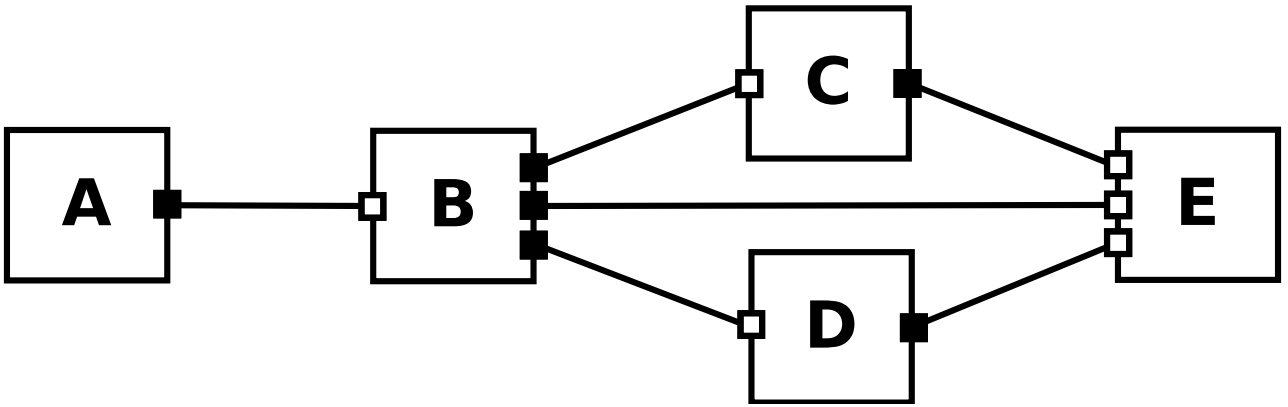
Descripción Organiza el procesado de los datos en un sistema en *pasos discretos*
Cada paso es realizado por un *componente diferente* (*filter*)
Los *datos fluyen* de un componente a otro (*pipe*)

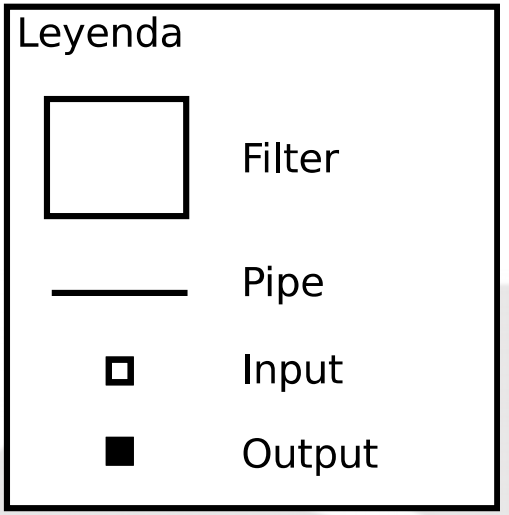


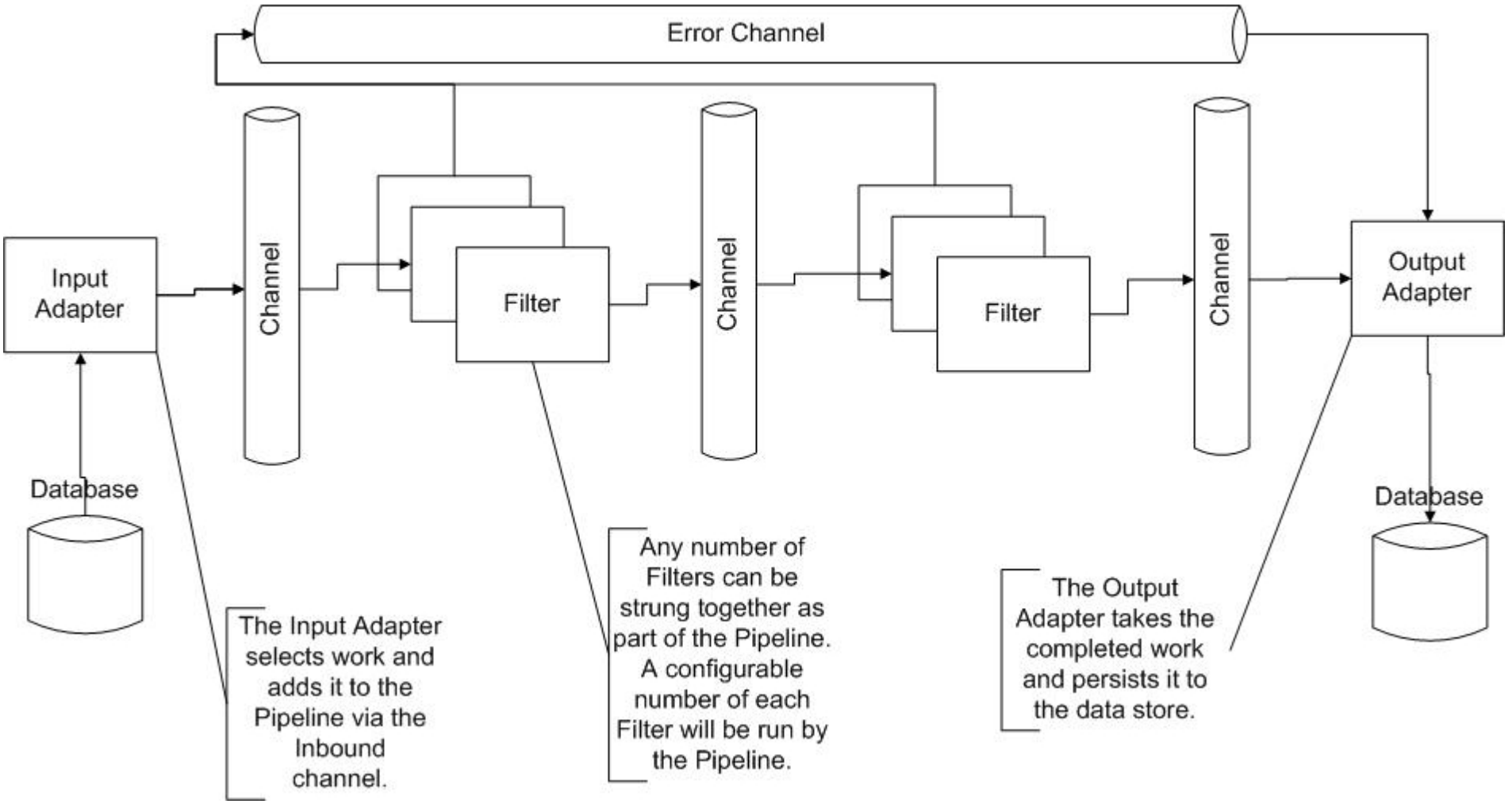


Arquitectura *pipe & filter*

Descripción Organiza el procesado de los datos en un sistema en *pasos discretos*
Cada paso es realizado por un *componente diferente* (*filter*)
Los *datos fluyen* de un componente a otro (*pipe*)

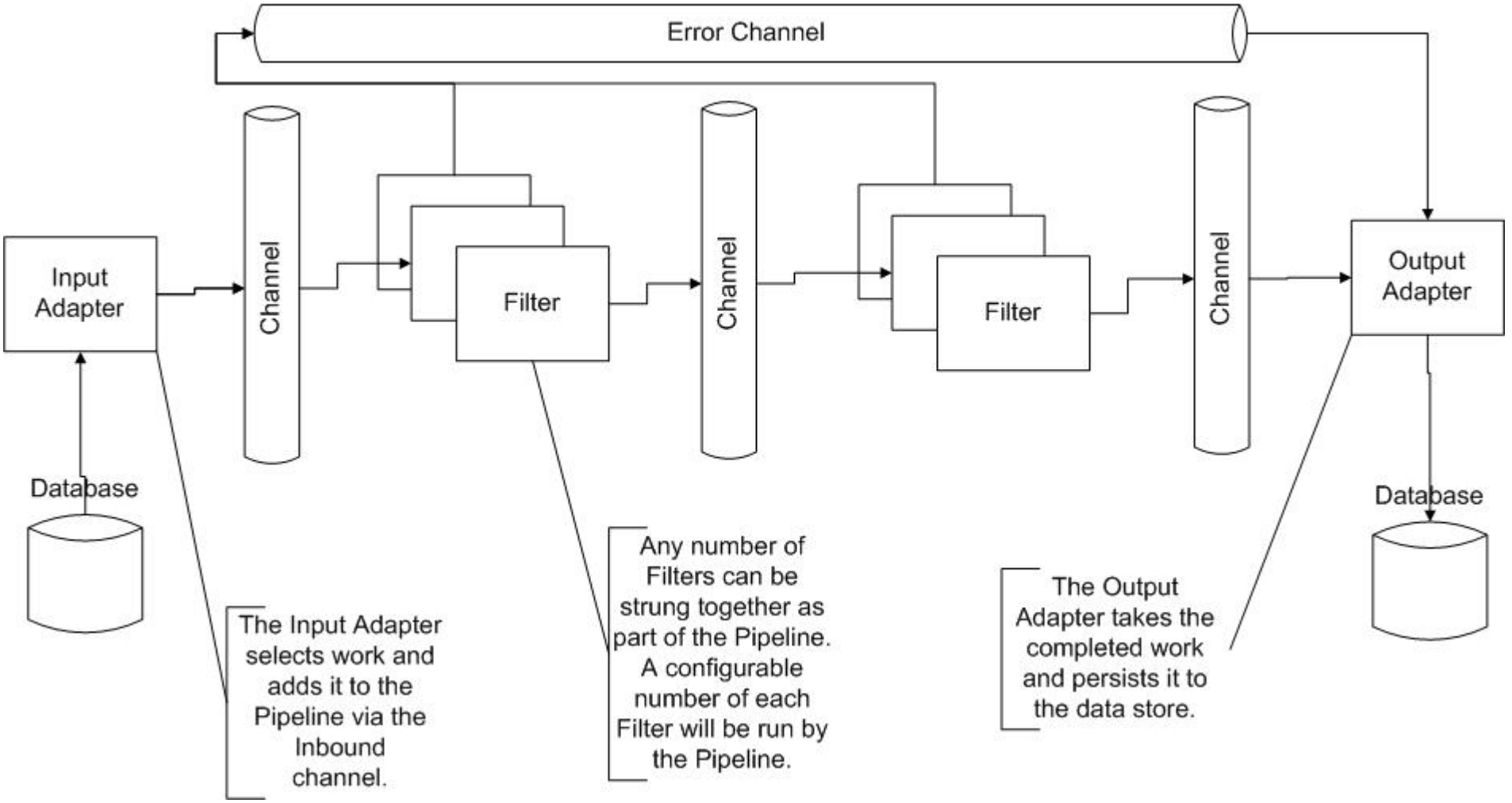


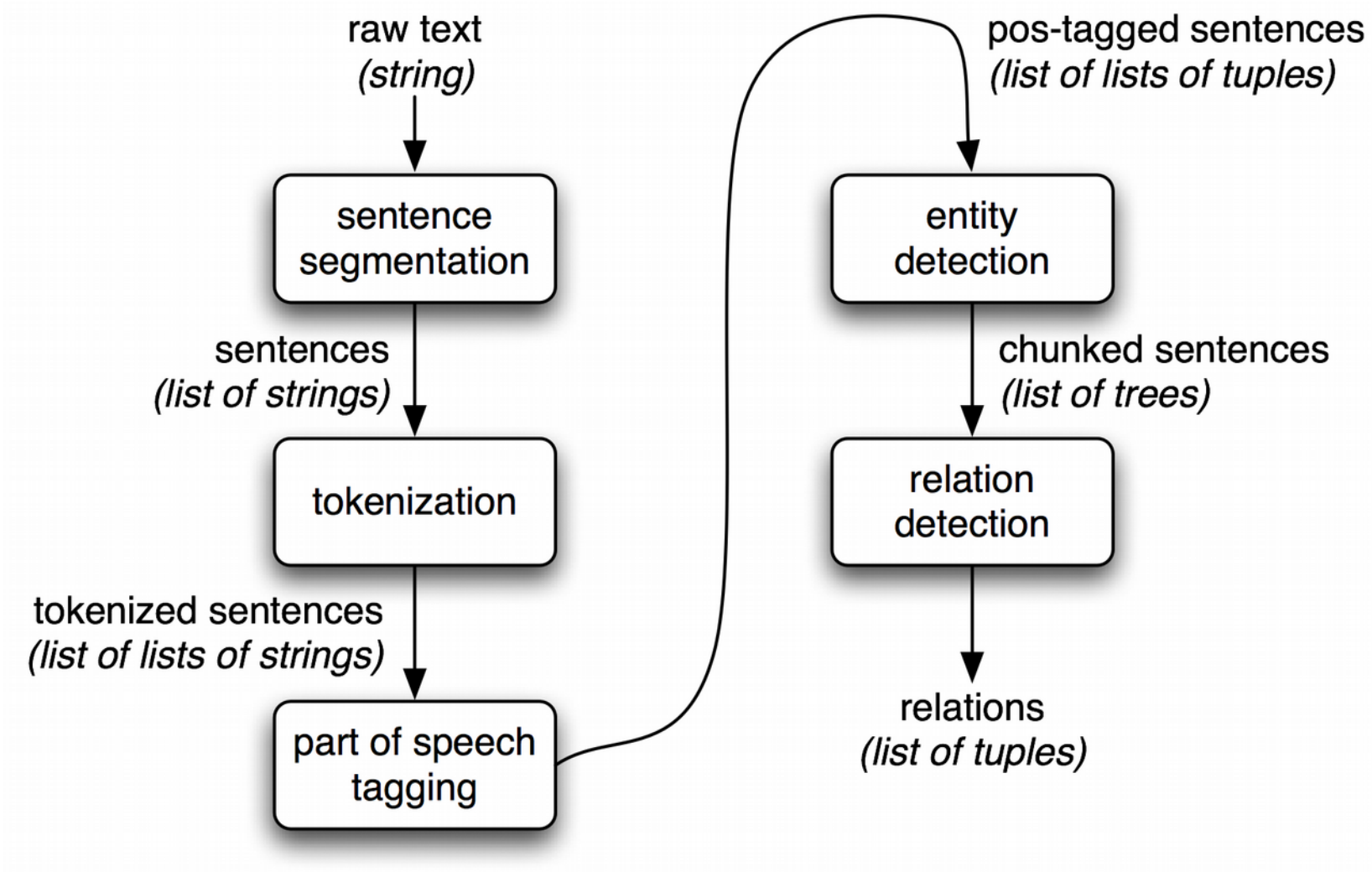



Error Channel
Input Adapter
Channel
Filter
Channel
Filter
Channel
Output Adapter
Database
Database
The Input Adapter selects work and adds it to the Pipeline via the Inbound channel.
Any number of Filters can be strung together as part of the Pipeline. A configurable number of each Filter will be run by the Pipeline.
The Output Adapter takes the completed work and persists it to the data store.

http://parc.sourceforge.net/intro.html

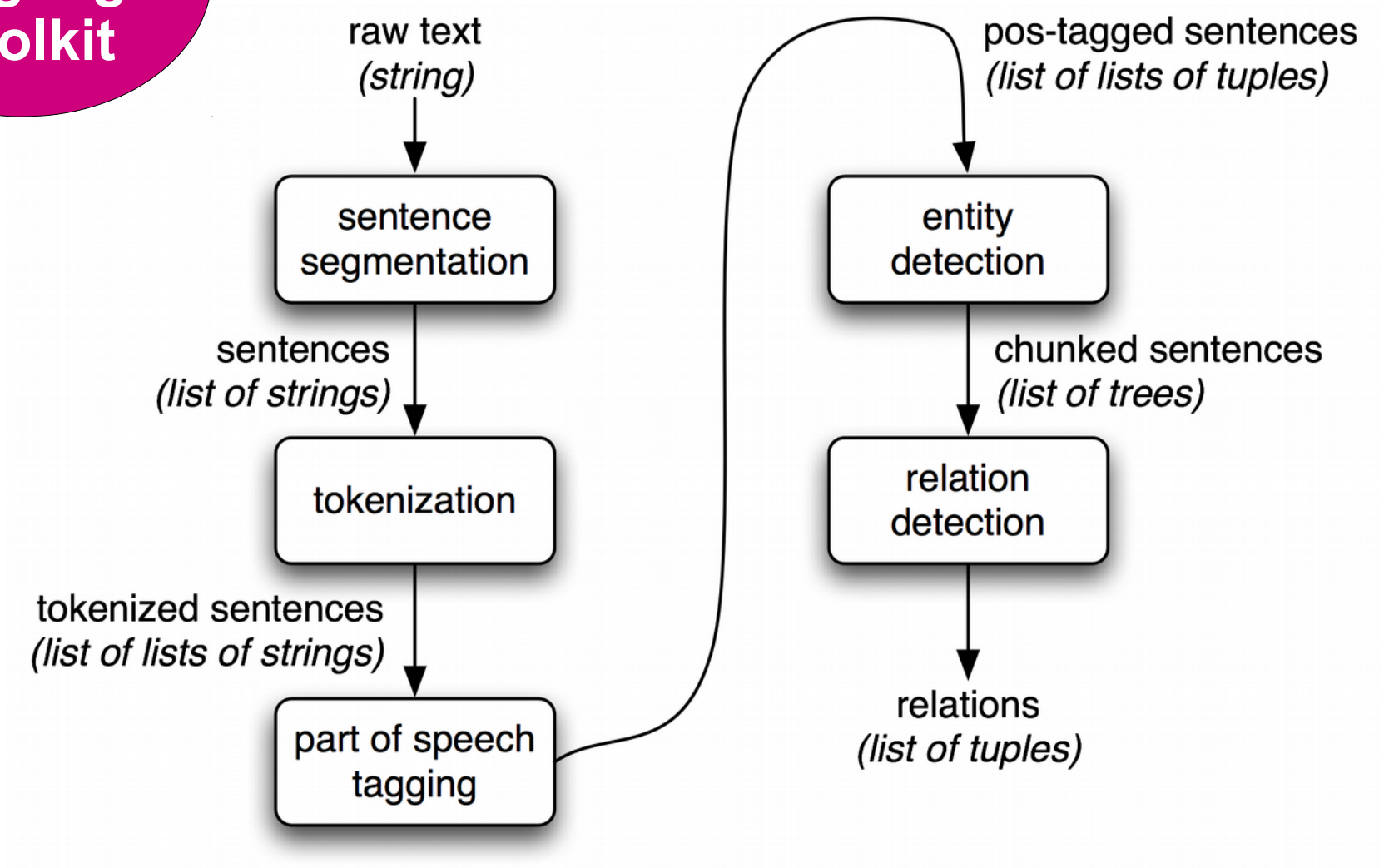
pipeline framework

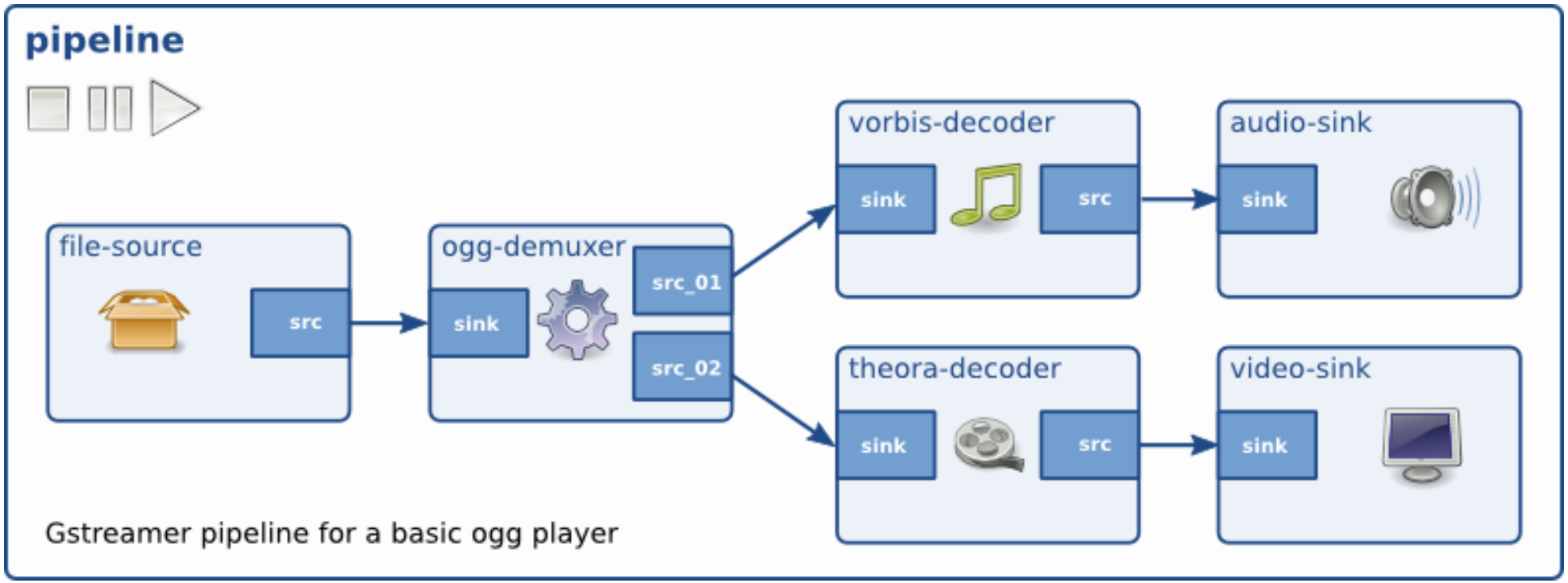



raw text
(string)
sentence segmentation
sentences
(list of strings)
tokenization
tokenized sentences
(list of lists of strings)
part of speech tagging
pos-tagged sentences
(list of lists of tuples)
entity detection
chunked sentences
(list of trees)
relation detection
relations
(list of tuples)

http://nltk.googlecode.com/svn/trunk/doc/book/ch07.html#fig-ie-architecture

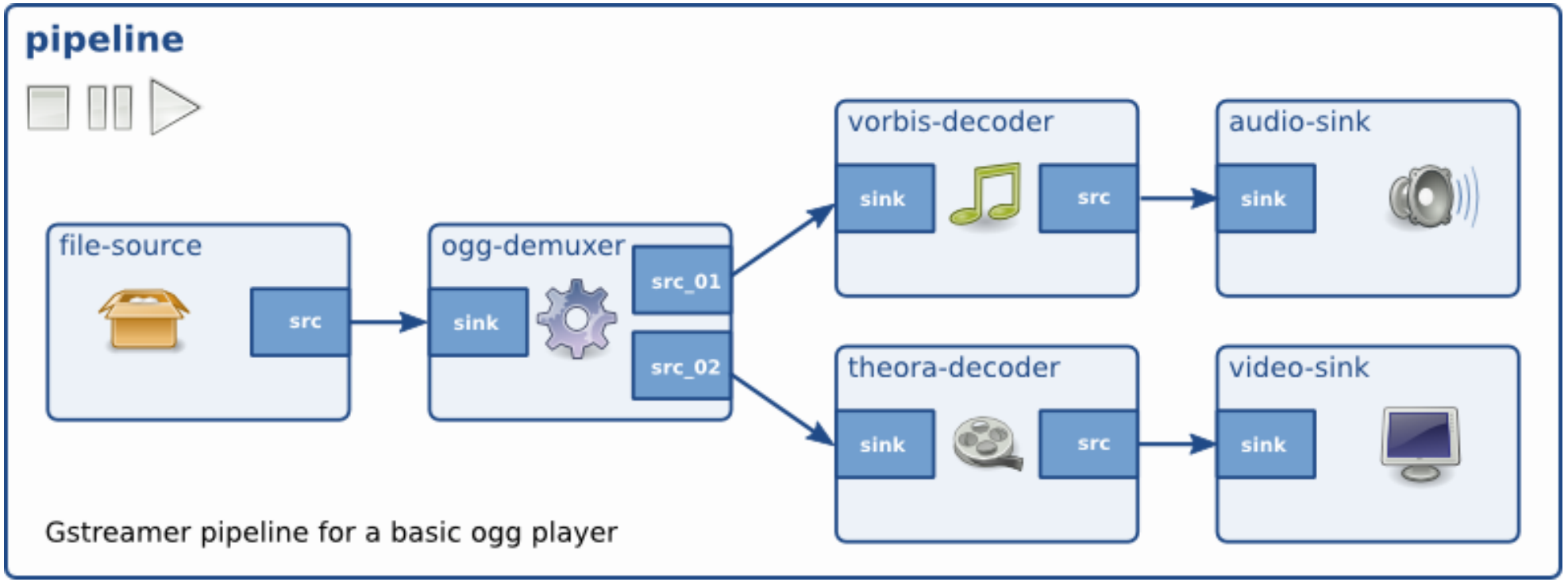
Natural Language Toolkit




pipeline
file-source
src
ogg-demuxer
sink
src_01
src_02
vorbis-decoder
sink
src
audio-sink
sink
theora-decoder
sink
src
video-sink
sink
Gstreamer pipeline for a basic ogg player

http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/html/section-intro-basics-bins.html

GStreamer



Arquitectura cliente/servidor

Arquitectura cliente/servidor

Descripción	Organiza la funcionalidad del sistema en *servicios*, cada uno proporcionado por un *servidor diferente* Los clientes son usuarios de los servicios y sus servidores correspondientes
Aplicabilidad	Construcción de sistemas en los que se necesita acceder a la información desde diferentes perspectivas Construcción de sistemas con carga variable
Nomenclatura	Arquitectura orientada a servicios

Arquitectura cliente/servidor

Descripción	Organiza la funcionalidad del sistema en *servicios*, cada uno proporcionado por un *servidor diferente* Los clientes son usuarios de los servicios y sus servidores correspondientes
Ventajas	Los servicios pueden ponerse a disposición de todos/algunos clientes Los servidores son independientes Los servidores pueden distribuirse Los servidores pueden replicarse Se potencia la interoperabilidad con el uso de protocolos estándar (RPC, HTTP)

Arquitectura cliente/servidor

Descripción	Organiza la funcionalidad del sistema en *servicios*, cada uno proporcionado por un *servidor diferente* Los clientes son usuarios de los servicios y sus servidores correspondientes
Desventajas	Cada servidor es un punto único de fallo El rendimiento es difícil de predecir porque también depende de la red Puede presentar problemas de gestión de los diferentes servidores Los clientes necesitan una “guía” o “directorio”

Arquitectura cliente/servidor

Descripción	Organiza la funcionalidad del sistema en *servicios*, cada uno proporcionado por un *servidor diferente* Los clientes son usuarios de los servicios y sus servidores correspondientes
Desventajas	Cada servidor es un punto único de fallo El rendimiento es difícil de predecir porque también depende de la red Puede presentar problemas de gestión de los diferentes servidores Los clientes necesitan una "guía" o "directorio"

puede desplegarse en una única máquina

Arquitectura cliente/servidor

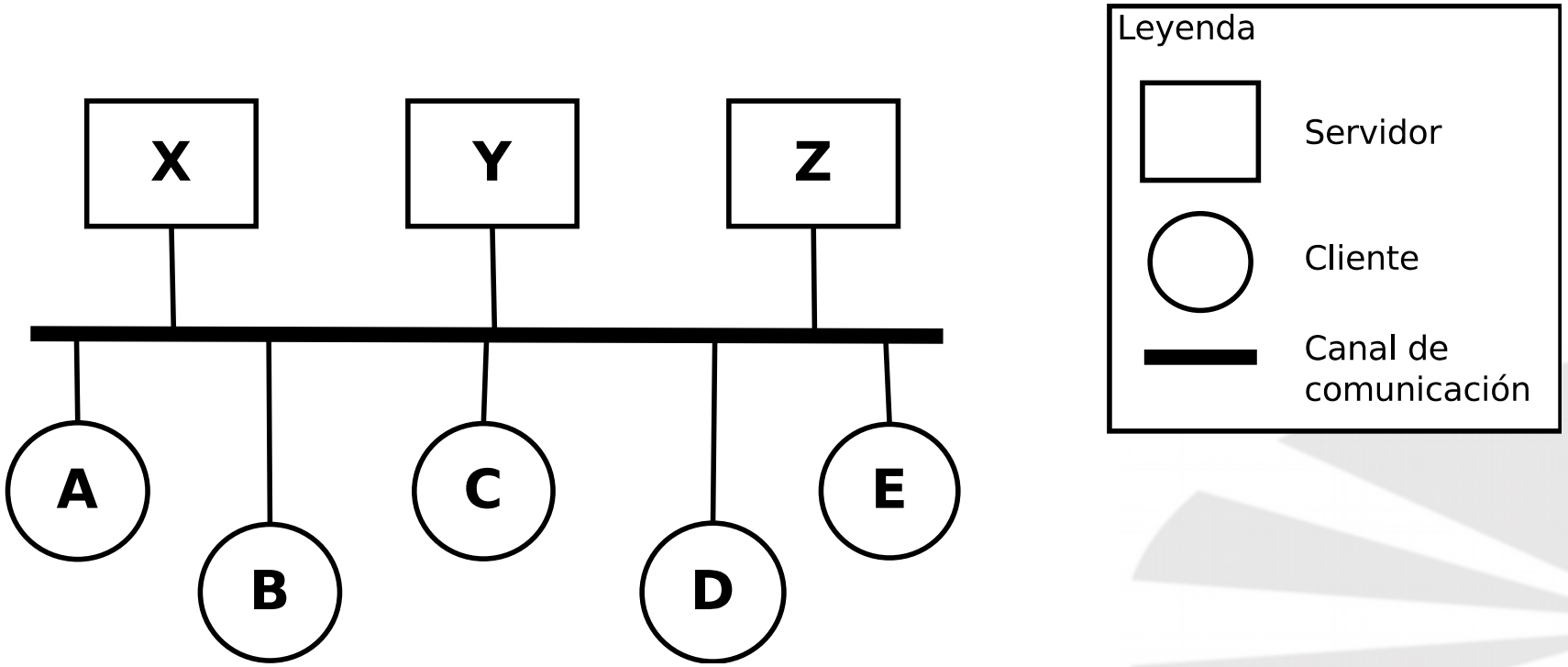
Descripción	Organiza la funcionalidad del sistema en *servicios*, cada uno proporcionado por un *servidor diferente* Los clientes son usuarios de los servicios y sus servidores correspondientes
Desventajas	Cada servidor es un punto único de fallo El rendimiento es difícil de predecir porque también depende de la red Puede presentar problemas de gestión de los diferentes servidores Los clientes necesitan una "guía" o "directorio"

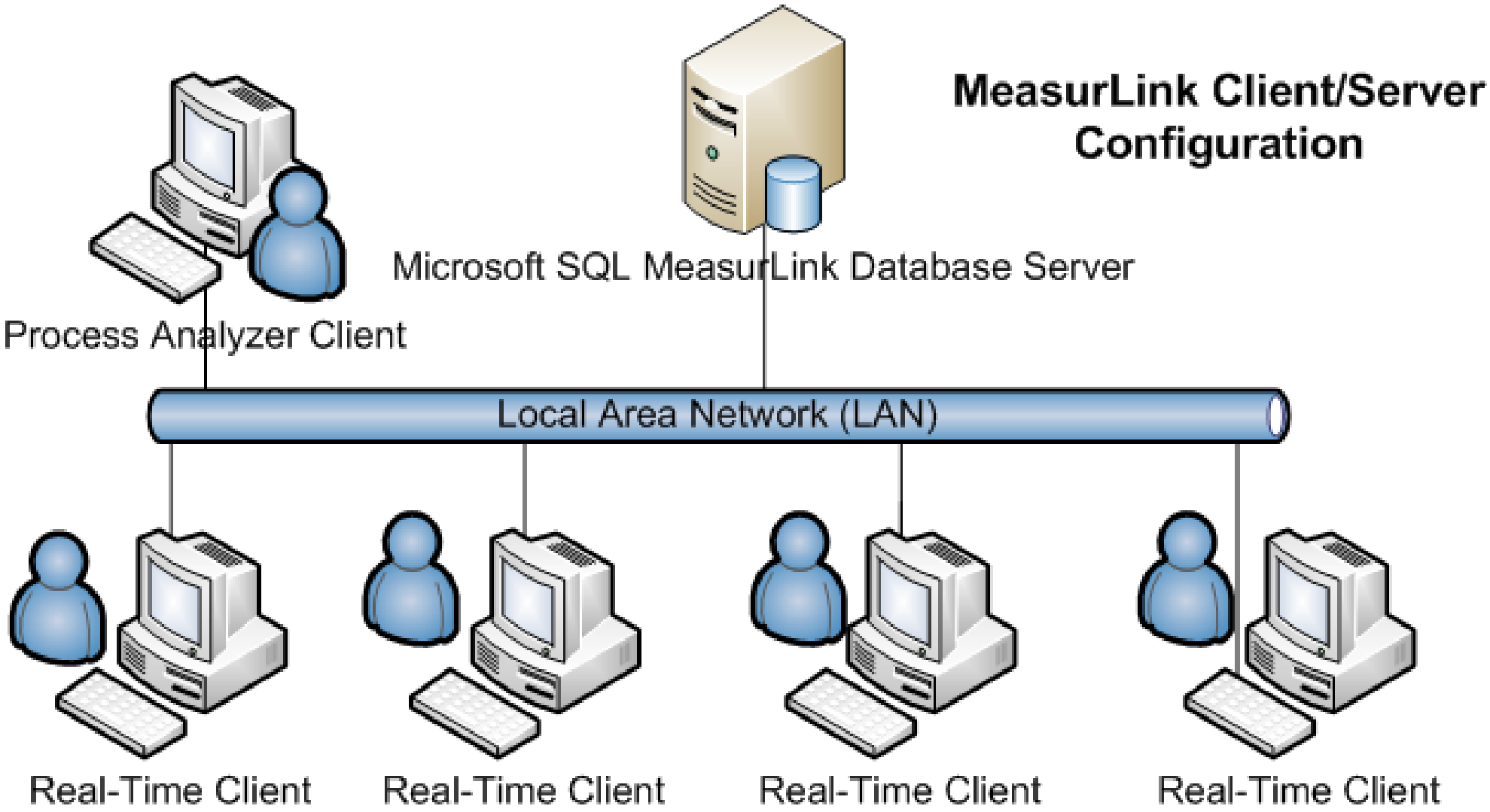
puede desplegarse en una única máquina

Arquitectura cliente/servidor

Descripción Organiza la funcionalidad del sistema en *servicios*, cada uno proporcionado por un *servidor diferente*

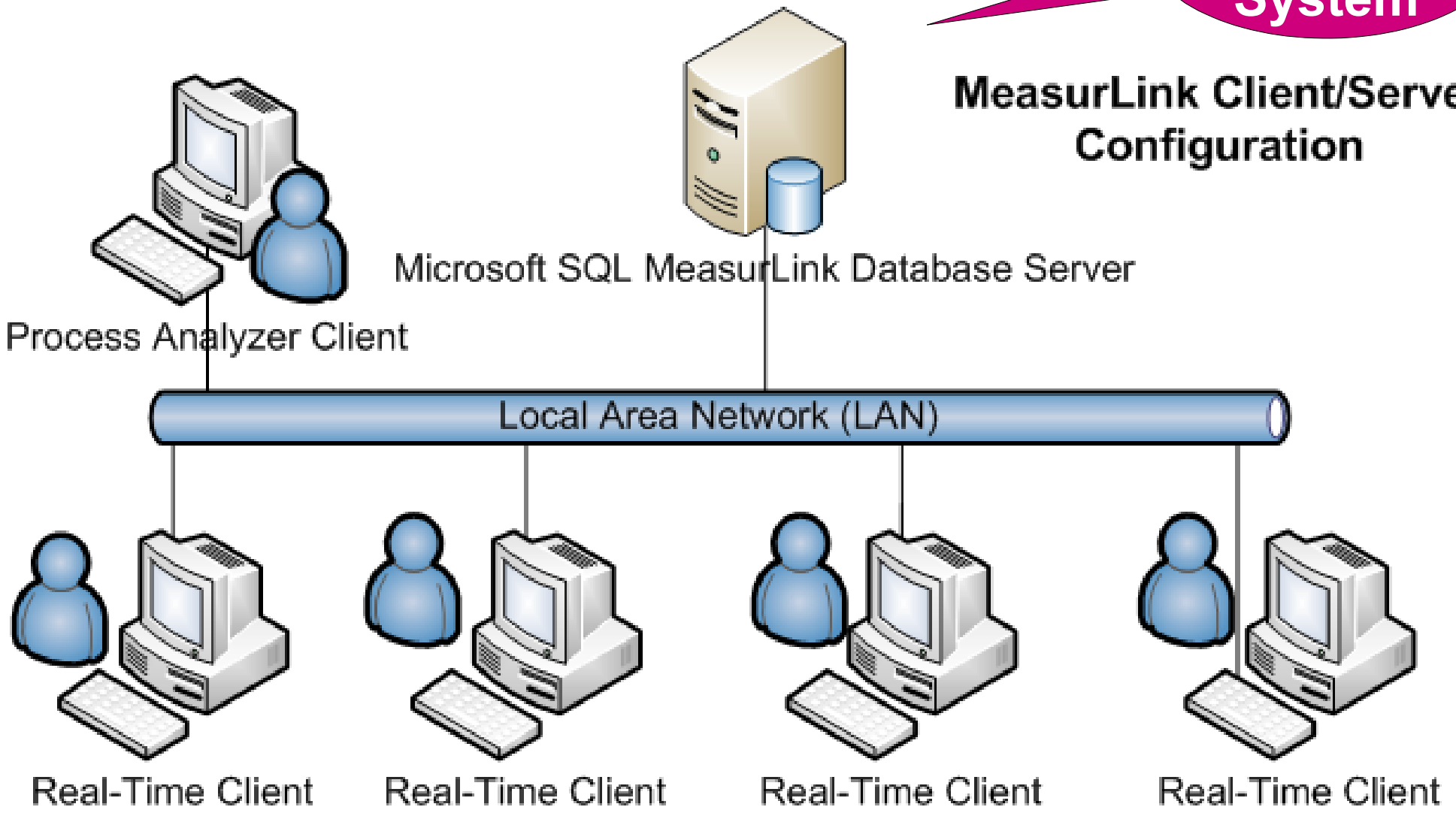
Los clientes son usuarios de los servicios y sus servidores correspondientes

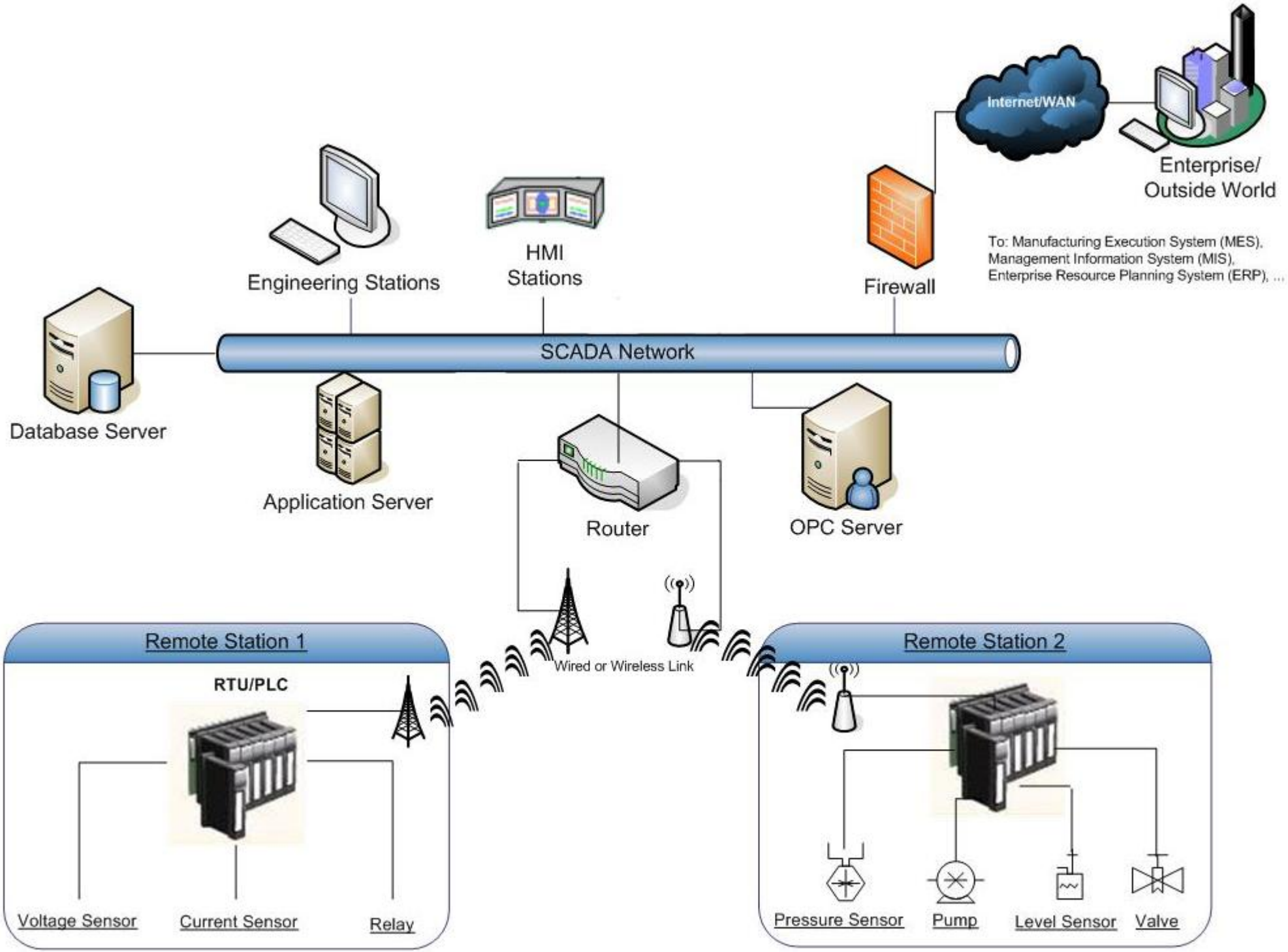



MeasurLink Client/Server Configuration
Microsoft SQL MeasurLink Database Server
Process Analyzer Client
Local Area Network (LAN)
Real-Time Client
Real-Time Client
Real-Time Client
Real-Time Client

http://www.measurlink.com/clientserver.html

Quality Management System

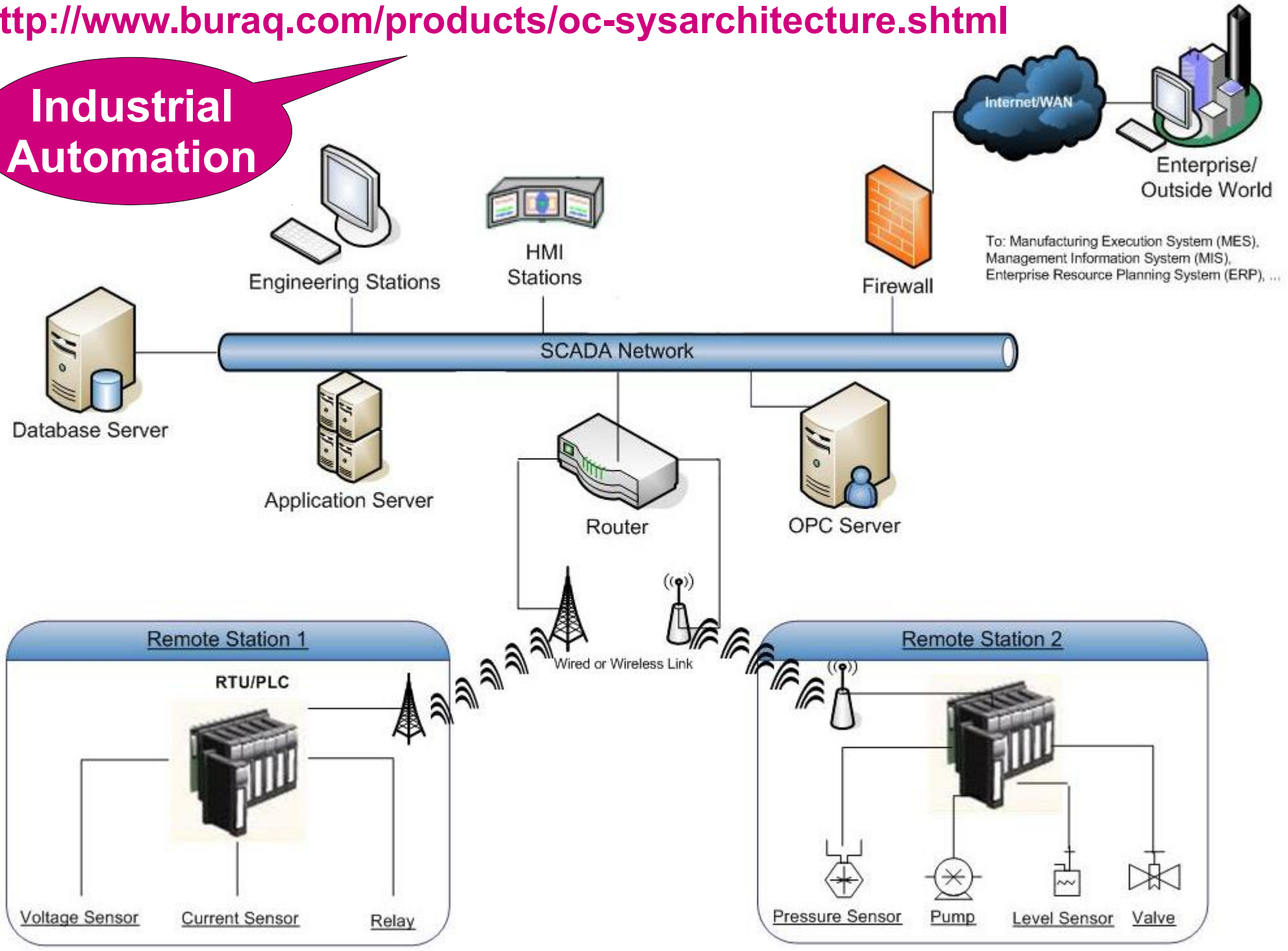




OpenControl SCADA Network Architecture

http://www.buraq.com/products/oc-sysarchitecture.shtml

Industrial Automation



OpenControl SCADA Network Architecture