



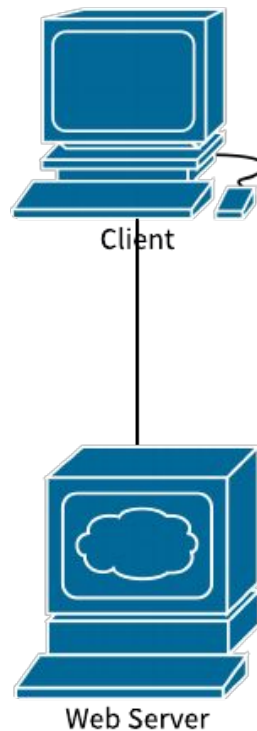
# Sistemas Distribuidos

# Cliente/Servidor

El sistema que vimos hasta ahora donde tenemos un cliente y un servidor, no es la norma.

Es extremadamente raro que estos sean los únicos dos componentes

Con el esquema tradicional cliente-servidor, nos quedamos cortos porque de base necesitamos una base de datos.



# Cliente/Servidor

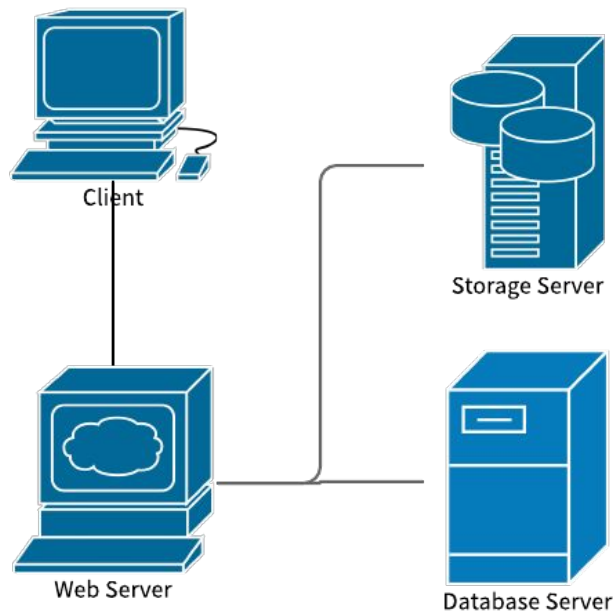
## RDBMS / Logs

En general se tiene separada la persistencia.

Y para la persistencia se tiene dos elementos mínimos

- » Una base de datos
- » Los archivos de logs

Las imagenes no se suelen guardar en bases de datos. Generalmente, se guardan en una CDN. Entonces, lo unico que hace falta guardar es el link de CDN para llegar a dicha imagen. No tiene sentido gastar recursos de mi webserver y DB para devolver una imagen, cuando esta puede estar en otro webserver preparada para devolver contenido estatico.

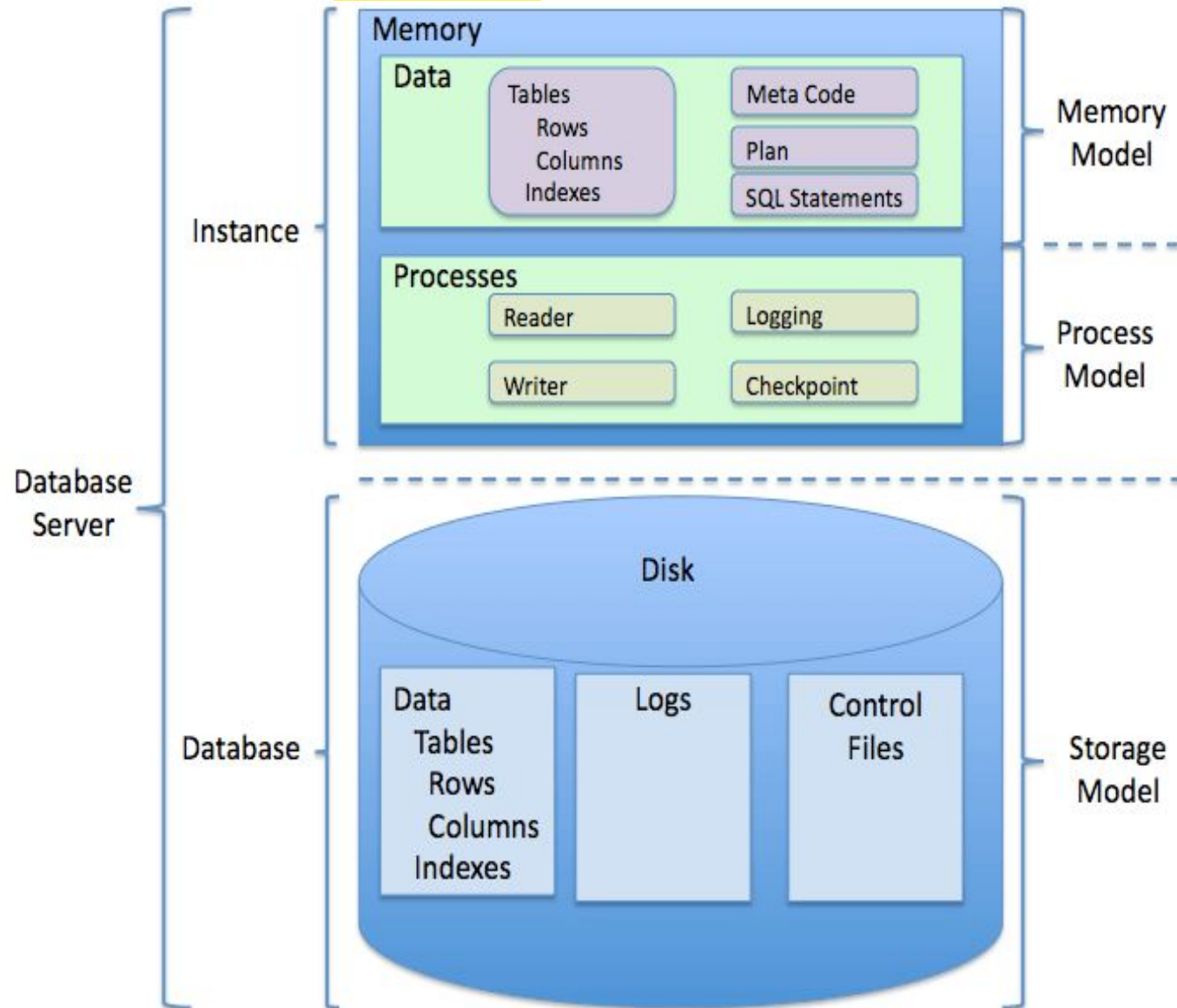




# RDBMS

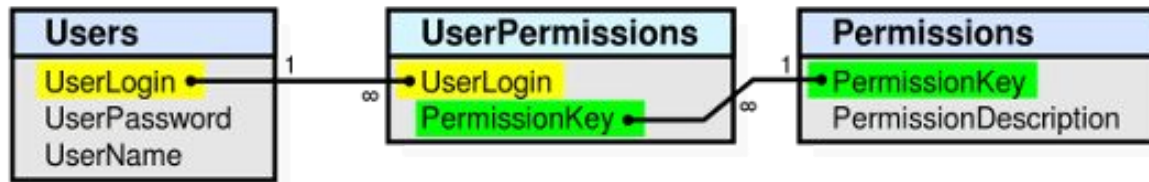
# ¿Qué es una RDBMS?

Un RDBMS es un programa/servicio que permite crear, modificar y administrar bases de datos relacionales y la información contenida en las mismas.



# Beneficios de los RDBMS

## Modelo Relacional:

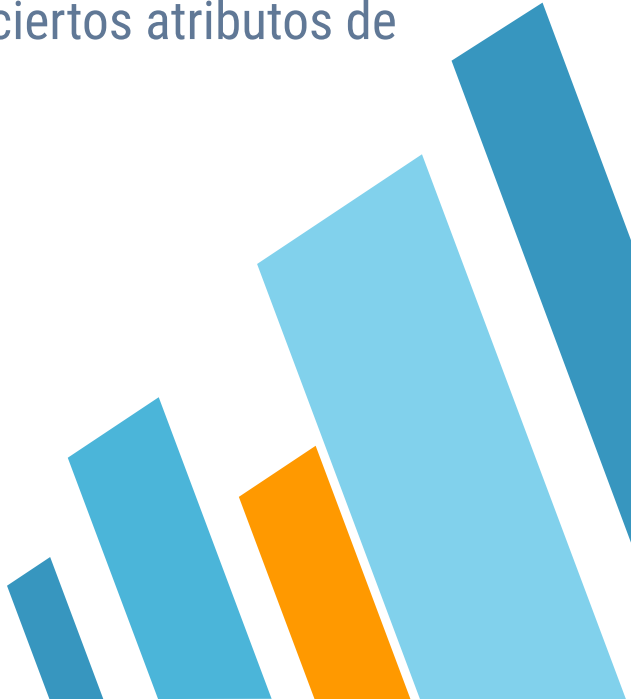


- » Guarda la información como un **"conjunto de relaciones"** representadas por tablas
- » **Una tabla contiene filas y columnas.** Cada fila representa a un registro de la tabla. Cada columna un atributo de dicho registro.
- » Cada fila puede tener una **clave que lo identifica** llamado **primary key** (que puede ser compuesta por varios atributos).
- » Las relaciones entre tablas se dan por atributos comunes (**foreign key**).
- » Se pueden validar **relaciones y unicidad** mediante las **constraints** sobre las keys.



# Beneficios de los RDBMS


Un **esquema** es la **implementación física del modelo de datos** y da la posibilidad de definir:

- **Tipos de datos** para cada atributo de una tabla.
    - **Constraints**: para restringir los datos ingresados al sistema mediante validaciones de unicidad y consistencia.
    - Permite **optimizaciones de guardado**.
  - **Índices**: para optimizar la búsqueda y filtrado por ciertos atributos de las tablas
- 



# Beneficios de los RDBMS

Transacciones **ACID**: Permite que varias operaciones como una operación lógica que cumple:


- » **Atomicidad**: indica que cada transacción es todo o nada. No se persisten cambios por la mitad.
  - » **Consistencia**: Toda información escrita es válida de acuerdo a todas las restricciones definidas.
  - » **Aislamiento**: Ninguna transacción será afectada por transacciones que se realizan de manera concurrente.
  - » **Durabilidad**: indica que una vez que se escribió la información, la misma es permanente y lecturas posteriores retornarán el nuevo dato escrito.
- 






# SQL

SQL (Structured Query Language) es un lenguaje de dominio específico utilizado para manejar y consultar información disponible en bases de datos relacionales (RDBMS), o para procesar en sistema de procesamiento de streams relacionales (RDBMS).





# SQL

- Originalmente basado en el **álgebra relacional**. SQL consiste de 3 sub lenguajes **DDL** (data definition language), **DML** (data manipulation language) y **DCL** (data control language), además de un **lenguaje de consulta** y la posibilidad de ejecutar bloques procedurales.
  - Fue uno de los primeros lenguajes comerciales para el modelo relacional de Edgar F. Codd. Se convirtió en el estándar de la **ANSI** y la **ISO** para dicho modelo.
  - Eventualmente se volvió sinónimo de **RDBMS**.
- 

# SQL Statements

## DDL

```
CREATE TABLE Book(  
  isbn INTEGER,  
  title VARCHAR(50),  
  publication DATE NOT NULL,  
  PRIMARY KEY (isbn)  
);  
  
ALTER TABLE Book ADD type VARCHAR(3);  
  
DROP TABLE Book;
```

## Query

```
SELECT Book.isbn AS Title,  
       count(*) AS Authors  
  
FROM   Book  
JOIN   Book_author  
       ON Book.isbn =  
Book_author.isbn  
GROUP BY Book.title;
```

## DML

```
INSERT INTO Book (isbn, title) VALUES (978-1501142970, 'IT');  
  
UPDATE Book SET isbn = 'IT: a Novel' WHERE isbn = ' 978-1501142970';  
  
DELETE FROM Book WHERE isbn = 978-1501142970;
```



## Beneficios

Durante unas décadas el sistema cliente/servidor con una RDBMs se lo consideraba el **sistema “norma”**.

Mientras los requerimientos del sistema se mantengan dentro del esquema “book keeping” esta solución es “mágica” y “one tool fits all”.

Lo cual lo hace simple de aprender, usar y mantener.



# Diversos Problemas

El gran problema es que **aparecieron nuevas necesidades**, y se intentó (forzó) que este mismo modelo las respondiera.

Lo que trajo diversos problemas debido a que no estaba utilizando la herramienta correcta.

Lo cual eventualmente dió origen a otras herramientas que resolvieran mejor esos problemas.





# **Problema 1**


## **Crecimiento Orgánico del Sistema**



# Problemas de escala con una RDBMS

## Lanzamiento inicial:

Se pasa de un “entorno de desarrollo” en una máquina a un servidor con una base de datos externa con un **esquema** “bien definido”, **normalizado** .



# Problemas de escala con una RDBMS

El servicio se vuelve popular:  
*Tenemos **muchas Lecturas!!!***

Agregamos un memcache para las  
consultas más comunes.

Por lo que las lecturas dejan de cumplir  
ACID. El cache debe ser expirado.

Alternativa es agregar réplicas, que requiere  
mantenimiento y mayor infraestructura.






# Problemas de escala con una RDBMS

El servicio se vuelve aún popular:  
*Tenemos **muchas Escrituras!!!***

**Escalar** la base de datos **verticalmente**,

Debemos **gastar plata en mejores componentes.**






# Problemas de escala con una RDBMS

Nuevas features generan **queries más complejas**:

***Demasiados Joins!!!! lecturas lentas!!!***

Hay que **desnormalizar** la base de datos..

Hay que **mantener la redundancia y gastar en más capacidad.**  
**Actualizaciones más complejas**

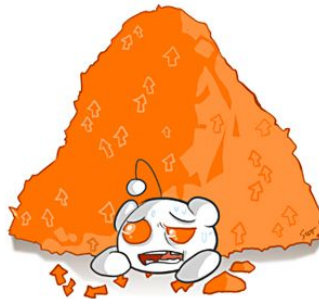


# Problemas de escala con una RDBMS

***Mucho volumen***

***las escrituras son cada vez más lentas***


**Eliminar triggers e índices secundarios.**



reddit is under heavy load right now



# ¿Cuál es el problema de los RDBMS?

- » Son **demasiados rígidos** para los nuevos escenarios
  - » La idea que **es una herramienta para todo** no aplica más.
  - » No es ideal para guardar **data no estructurada**.
  - » **Difícil de escalar** y mantener millones de registros.
  - » **Las estructuras de datos** utilizadas por estos sistemas están optimizadas para lecturas utilizando poca cantidad de memoria.
- 



# **Problema 2**

## **BI - Queries Análíticas**

# BI - Nuevas Preguntas

A partir de la información que se encuentra en la base de datos relacional, y algunas otras fuentes, las empresas empezaron a notar que **se puede generar “nuevo conocimiento”** y a partir del mismo **“nuevas decisiones y acciones”**.

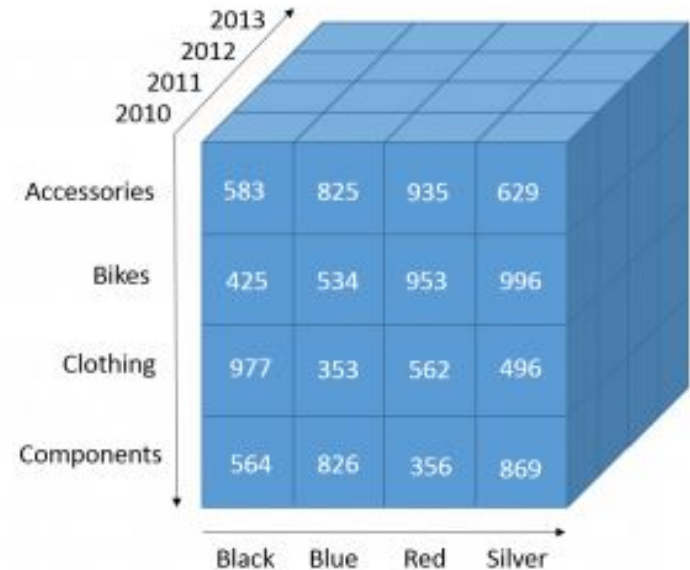
Pero eso hizo que **surgieran nuevas preguntas**



# BI - Nuevas Preguntas

Muchas de estas preguntas generalmente comienzan con

- » ¿Cuántos (elementos) ... ?
- » ¿Cuánto (cuesta) ...?
- » ¿Cuál es el promedio (o media u otra medida estadística) ...?



Y esos valores se calculan a partir de la combinación de un **conjunto de dimensiones**


Estas preguntas suelen ser agregaciones. Se puede hacer en una base de datos relacional, pero hay una cantidad muy grande de joins, y esto gasta muchos recursos. El hecho de tirar queries analíticas sobre mi base de datos productivas no es la mejor, ya que le estoy sacando poder de procesamiento. Lo ideal es hacer estas queries en otra base de datos.



# BI - Nuevos Problemas

El tipo de preguntas **se responden a partir de queries de agregación.**

Aunque las RDBMS tienen agregaciones la performance de las mismas sufre porque:

- » El **modelo por fila no es el más óptimo para agregar.**
  - » Al **agregar por dimensiones no indizadas se requiere full scans.**
  - » Las **operaciones requieren más tiempo para procesar.**
  - » Si tiene locks (ya sea por tabla o fila) los tiene que mantener por más tiempo.
  - » Todo esto **afecta no solo a la query lanzada, sino a las que corran concurrentemente en la base de datos.**
- 



# BI - Online Analytical Processing

Los OLAPs son softwares cuyo modelo permiten mantener la información por las dimensiones y agregar de manera fácil

Available Dimensions

ETHNICITY\_SD

AGE\_RANGE\_SD

CALENDAR\_MO

Columns

Calendar Year

Gender

ETHNICITY\_SD

AGE\_RANGE\_SD

Calendar Year (2)			1993					1994	
Gender	ETHNICITY_SD (2)	ANNUAL_SA...	HOURLY_FTE	GENDER_SD	ETHNICITY_SD	AGE_RANGE	ANNUAL_SA...	HOURLY_FTE	
		Value	Value	Value	Value	Value	Value	Value	
Female		5218709.00	110.00	206.00	206.00	206.00	13600473.00	250.00	
	African-Ame...	1012032.00	36.00	52.00	52.00	52.00	2478184.00	66.00	
	Asian Pacific Islander	309120.00	18.00	20.00	20.00	20.00	1802640.00	42.00	
	Black Caribbean	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	Caucasian	3452757.00	44.00	114.00	114.00	114.00	8051809.00	106.00	
	Hispanic-Ot	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	Indian	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	Mexican	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	Other	416000.00	0.00	8.00	8.00	8.00	1181440.00	0.00	
	Pakistani	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	Puerto Rican	0.00	0.00	0.00	0.00	0.00	0.00	0.00	
	Sioux	28800.00	12.00	12.00	12.00	12.00	86400.00	36.00	
Male		7292082.00	52.00	172.00	172.00	172.00	17922115.00	124.00	
Total by COLUMNS		12510791.00	162.00	378.00	378.00	378.00	31522588.00	374.00	

# BI - OLAP

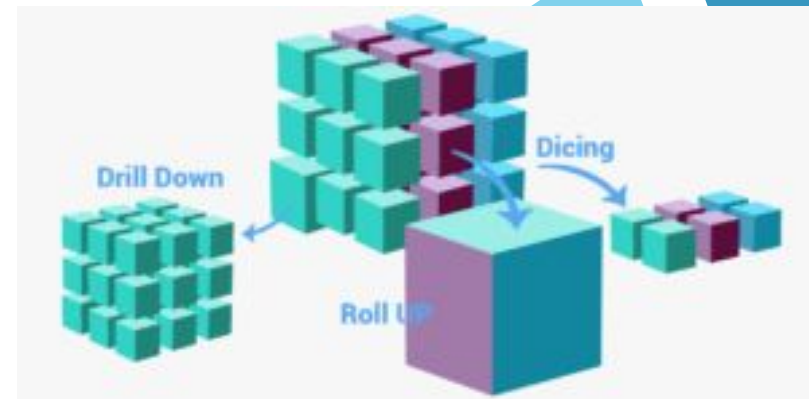
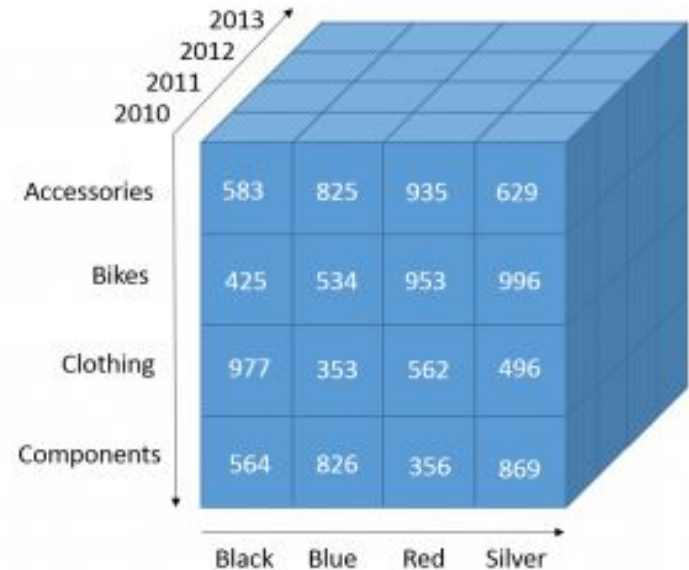
Conceptualmente se genera un “cubo” con la información dividida a partir de la granularidad de las dimensiones que se quieran utilizar.

Y a partir de acciones:

- » **Drill down:** agregar
- » **Roll up:** desagregar
- » **Slice:** Fijar una dimensión.
- » **Dice:** Fijar más de una dimensión
- » **Pivot:** Rotar la selección de dimensiones

Se obtienen las respuestas esperadas.

Esto surgió en un momento en el que las computadoras no eran tan potentes. En la base de datos productiva, la data histórica se va borrando. En la base de datos OLAP del data warehouse, se va guardando la data desnormalizada de las distintas bases OLTP de distintos servicios de mi empresa (mediante un ETL). Las queries de BI se tendrían que hacer sobre la base OLAP. Estos procesos solían tardar de días a semanas.





## BI - Data Warehouses

Las organizaciones requieren llevar la información desde los RDBMs a los modelos OLAPs.

Para lo cual se requiere un pipeline de operaciones y transformaciones.

Para las mismas las organizaciones cuentan con un Data Warehouse que es un sistema que permite realizar las tareas básicas:

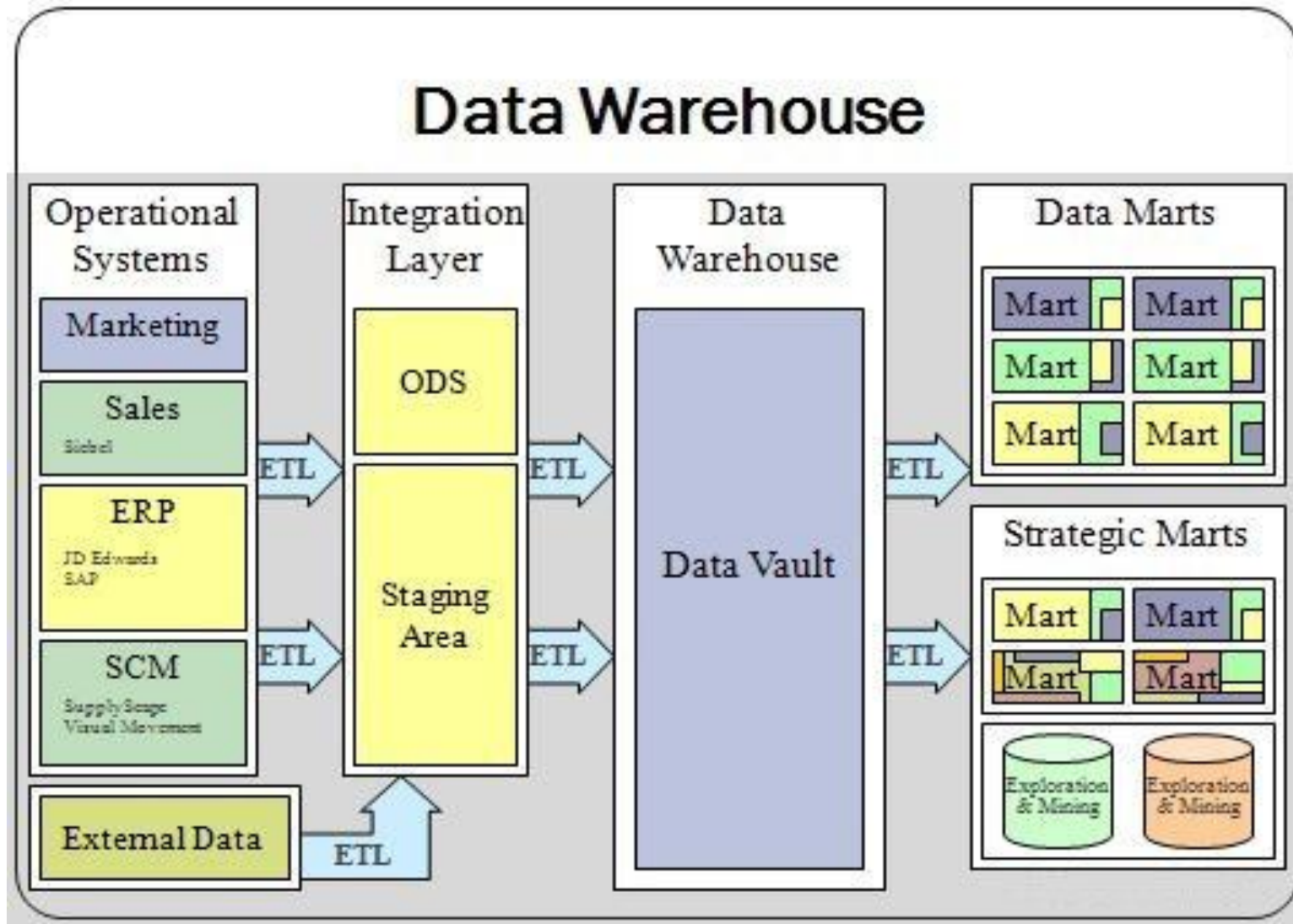
- » Obtención/Extracción (E)
- » Transformación (T)
- » Carga (L)

Esto puede pasar en varios momentos y con stores intermedios para llegar a los modelos finales.



# BI - Data Warehouses


Esto muestra la complejidad de los sistemas que hacen ETL para pasar de las OLTP a las OLAP.  
Antiguamente, la data podia tardar semanas hasta llegar a la OLAP (en empresas muy grandes).  
Este es el costo que pagamos por poder hacer queries mas rapido en una OLAP.





# BI - DW / OLAP

Este tipo de modelo tiene ciertas características:

- » Los procesos de ETL se realizan de manera Batch Periódicamente.
  - » Generalmente estos procesos llevan mucho tiempo en terminar su tarea.
  - » Por una cuestión de espacio no se guarda la información histórica granular sino que se agrega al mínimo necesario para los modelos por los OLAPs.
  - » Por esta razón los modelos y preguntas a realizar son predefinidas y tienen una cierta rigidez.
  - » Si quiero hacer nuevas preguntas sobre la data del año pasado no puedo
- 



# **Problema 3**

## **Big Data**





# ¿Cómo se define Big Data?

Basicamente big data es medio un chamuyo, hoy en día se le dice big data a cualquier cosa

**Jon Bruner**, Editor-at-Large, O'Reilly Media

“Big Data is the result of collecting information at its most granular level — it’s what you get when you instrument a system and keep all of the data that your instrumentation is able to gather.”

**Ryan Swanstrom**, Data Science Blogger, Data Science 101

“Big data used to mean data that a single machine was unable to handle. Now big data has become a buzzword to mean anything related to data analytics or visualization.”

<http://datascience.berkeley.edu/what-is-big-data/>





# ¿Cómo se define Big Data?

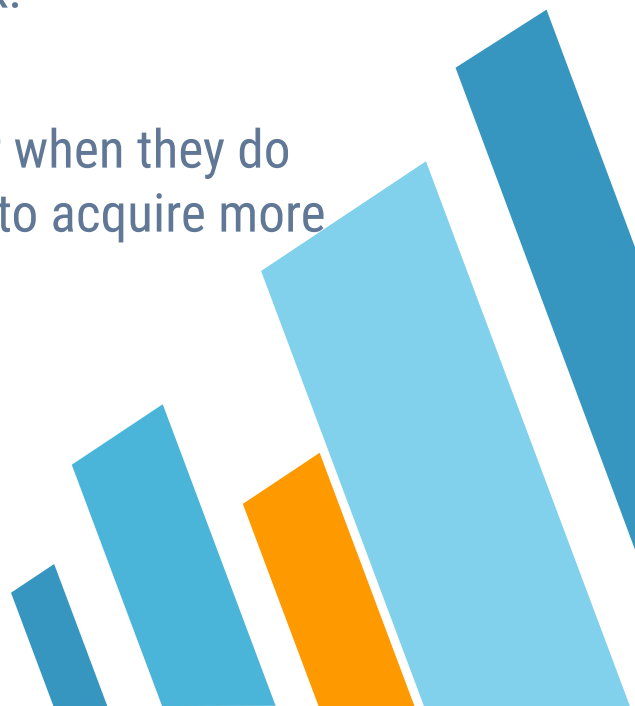
En 1997 “del problema de **Big Data**” en un paper de la NASA escrito por **Michael Cox and David Ellsworth**:

El origen de big data es que tenían muchas fotos del espacio y no podían procesarlas todas en la misma PC

“Visualization provides an interesting challenge for computer systems: **data sets are generally quite large, taxing the capacities of main memory**, local disk, and even remote disk.

We call this the problem of big data.

When data sets do not fit in main memory (in core), or when they do not fit even on local disk, the most common solution is to acquire more resources.”







# ¿Cómo se define Big Data?

En 2001 Doug Laney (de Gartner) “creó” la definición de las **3 Vs**.

## Volumen

La cantidad de datos **crece de manera geométrica**.


## Variedad

Aparecen **nuevos tipos de datos**, que deben ser analizados que no cumplen con una estructura “fija”.

Son **semi estructurados o no estructurados**

## Velocidad

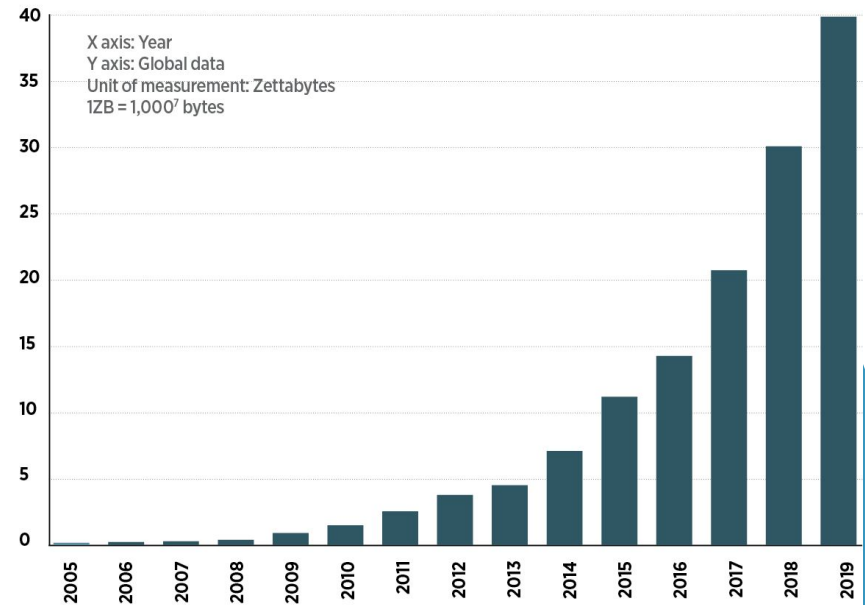
La **velocidad con que los datos se generan crece**



# Volumen

- » **2003:** Se generaron 5 exabyte.
- » **2004:** Se consolidan las primeras redes sociales.
- » **2006:** Se generan 161 exabytes. Se predice que la cantidad se duplicará cada 18 meses. (Ley de Moore)
- » **2007:** Primer iPhone. Infraestructuras en la Nube Pública se vuelve de uso común.
- » **2013-2014:** En este período se crean más datos que en toda la historia de la humanidad
- » **2015:** En Agosto Mil millones de personas usan Facebook en un día en promedio

DATA GROWTH



Note: Post-2013 figures are predicted. Source: UNECE



# Velocidad

Durante 2015 se midieron estas estadísticas:

- » Los usuarios de Facebook enviaban 31 millones de mensajes y ven 2.7 millones de videos por minuto.
  - » Cada minuto se subian hasta 300 horas
  - » Se sacaron 1 trillón de fotos de las cuales mil millones se compartieron de manera online
  - » En promedio se enviaban 6000 tweets por segundo
- 

# Variedad

Los datos pueden venir en varios formatos:

## Estructurados

La información tiene un esquema fijo que puede replicarse fácilmente en una RDBMs



## Semi estructurados

Generalmente Text-based, tienen un “esquema” que se puede cumplir en su totalidad o no. Ejemplos JSON y XML.

```
<?xml version="1.0" encoding="UTF-8" ?>
<listado>
  <persona>
    <nombre>Juan</nombre>
    <apellidos>Palomo</apellidos>
    <fecha>10/10/1980</fecha>
  </persona>
  <persona>
    <nombre>Juan</nombre>
    <apellidos>Palomo</apellidos>
    <fecha>10/10/1980</fecha>
  </persona>
</listado>
```

```
var myJSONObject = { "listado": [
  {
    "nombre": "Juan",
    "apellidos": "Palomo",
    "fecha": "10/10/1980"
  },
  {
    "nombre": "Juan",
    "apellidos": "Palomo",
    "fecha": "10/10/1980"
  }
]
}
```

## Desestructurados

La información no tiene un esquema aparente y puede no ser textual.

Ejemplos:

Mails.

Archivos de procesadores de texto o pdfs.

Imágenes.



# 3Vs - Nuevas fuentes

Las organizaciones comenzaron a tomar datos generados por fuera de sus sistemas. Por lo cual “máquinas” y “personas” son nuevas fuentes de información:

## Datos

### Organizacionales

- » Stocks.
- » Compras y ventas
- » Transacciones
- » Datos de usuarios

## Máquinas

- » Sensores
- » Satélites
- » Logs
- » IOT
  - bandas deportivas
  - Casas inteligentes.

## Personas

- » Redes sociales
- » Fotos
- » Videos.
- » Comentarios




FitBit can produce several gigabytes a day



# Big Data - Problemas

Por “definición” trabajar con Big Data excede las capacidades de “una máquina”, se requieren implementar nuevas estrategias para poder resolver los problemas que surgen a cada paso de la cadena de procesamiento:

- » **Adquisición**: tenemos que soportar tanto el volumen, como la velocidad dando soporte a la variedad de fuentes. Y probablemente *buffereando* para evitar problemas de procesamiento
  - » **Procesamiento**: Realizar el procesamiento dentro de un tiempo que sea útil para la organización. Tal vez particionando la operación de acuerdo a prioridad y complejidad.
  - » **Guardado**: se deben utilizar estrategias de guardado acordes a los datos y necesidades de respuesta.
  - » **Vista**: tener formas de mostrar la información que sea útil para los usuarios del sistema.
- 

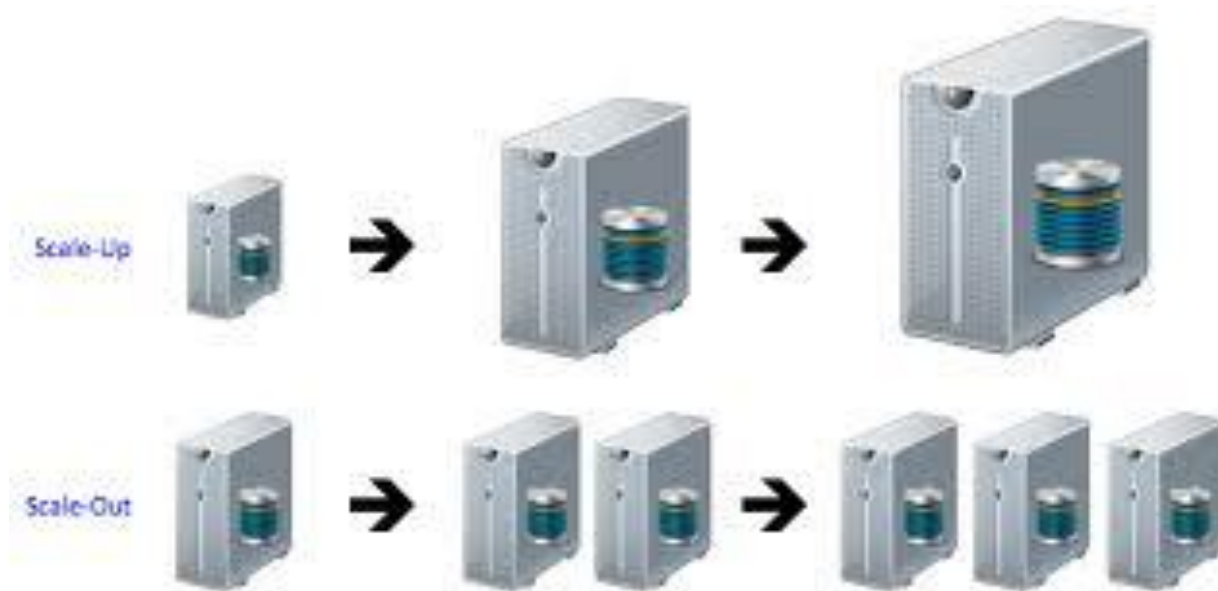


# Sistemas Distribuidos

# Sistemas Distribuidos

Todos estos problemas que se fueron mencionando llevan a un sinceramiento de que el sistema en Base y Server y el escalamiento “poniendo fierros más grandes” (**escalamiento vertical**) tiene un límite.

Entonces ahora es más común o simple buscar sistemas que **escalen de manera horizontal**.

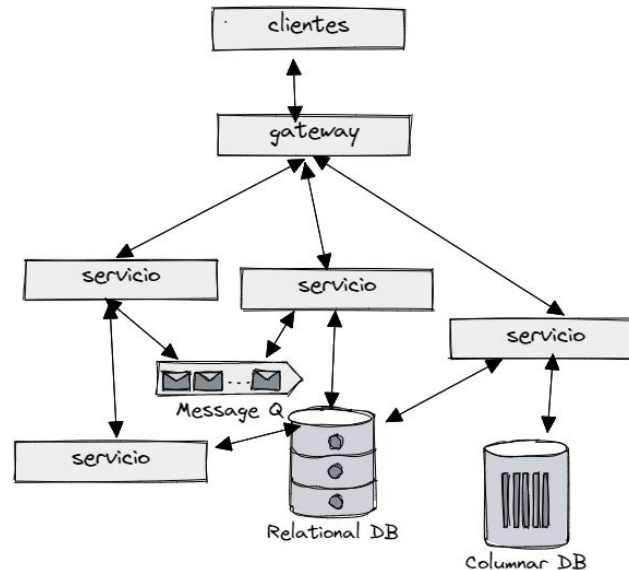




# Sistema Distribuido

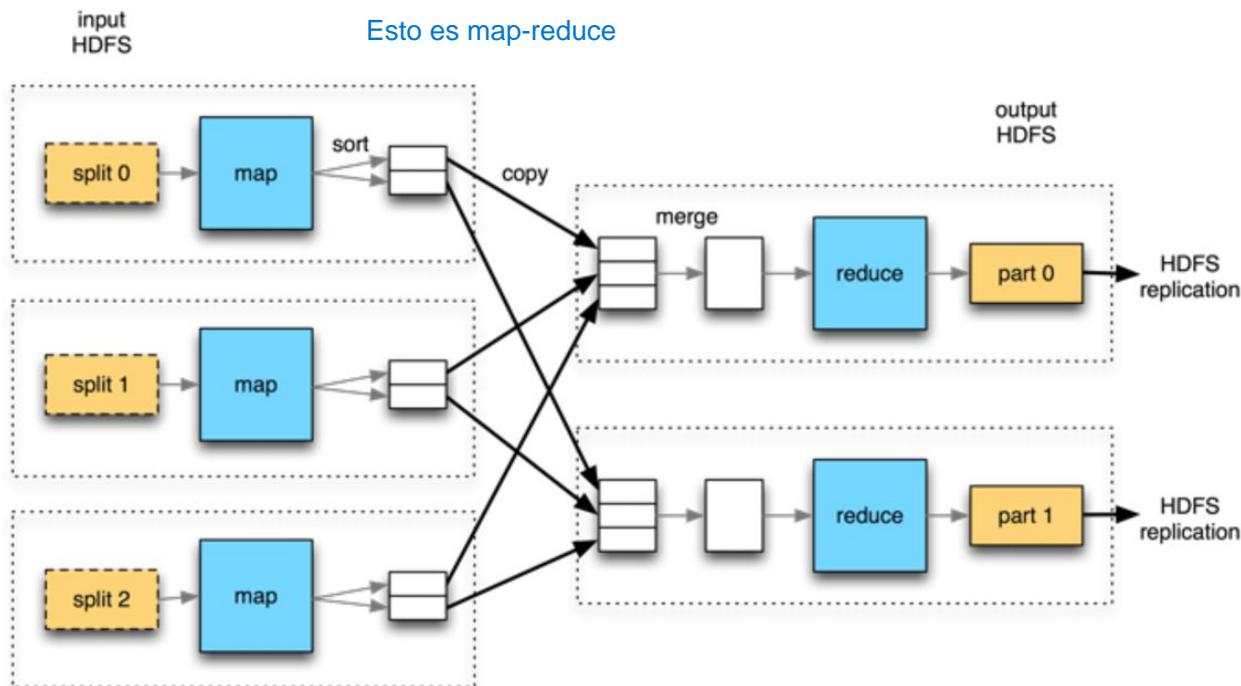
Es un sistema computacional cuyos componentes están distribuidos en diferentes computadoras (o elementos computacionales), que se conectan y comunican por medio de mensajes de una red.

Este sistema se presenta ante sus usuarios como un único sistema coherente.



# Sistema Distribuido - Objetivo

Las componentes del sistema **dividen una tarea** coordinando sus recursos **para completarla de manera más eficiente** que si se ejecutara en una sola computadora.





# Sistema Distribuido - Tipos de tareas

Las tareas a ejecutar serán particionadas y repartidas entre las diferentes partes del sistema.

Las tareas se pueden “clasificar” en 3 tipos:

## Almacenamiento

Para guardar elementos o cantidades de elementos que no entrarían en un nodo


## Procesamiento

Para subdividir una tarea y obtener resultados parciales de cada subset

## Responsabilidades


Dividir una tarea en diferentes partes “lógicas”

probablemente con sentido semántico independiente





# Sistema Distribuido - Características

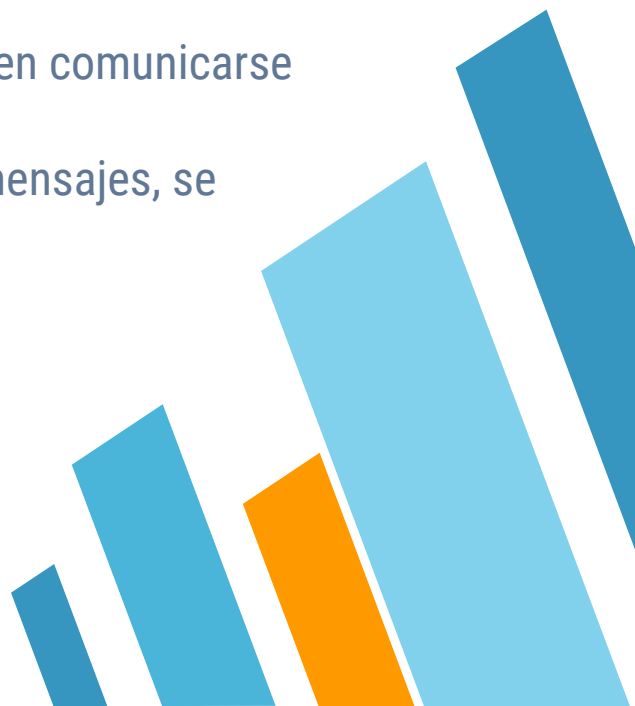
- Hay **más de un elemento computacional** llamado nodo que pueden ser computadoras, máquinas virtuales, dispositivos, etc.
  - Cada uno de los elementos es **autónomo y débilmente acoplado** o sea **tiene TODO** para operar por sí solo.
  - Cada proceso ejecutado en un nodo tiene su **propio espacio de direccionamiento**.
  - Las **computadoras se comunican entre sí por una red**.
  - Cada **nodo ofrece** acceso a sus recursos a través de **"SERVICIOS"**, ya sea a otros nodos o a clientes finales.
- 



# Sistema Distribuido - Problemas

Los sistemas distribuidos traen beneficios pero al mismo tiempo traen problemas inherentes a la manera en la cual se distribuye.

Algunos de estos problemas **puede ser fallas en:**

- Alguno de los componentes lógicos del sistema: **bugs, falta de memoria,** etc.
  - Los nodos del sistema: ya sea en **hardware** o recursos.
  - La **comunicación por red**: los nodos andan pero no pueden comunicarse entre ellos.
  - La **coordinación entre las tareas y/o nodos**: se pierden mensajes, se distribuyen mal las tareas, etc.
- 



# **Modelo de referencia**




# Modelo de referencia

La ISO definió un modelo de referencia para procesamiento abierto distribuido **(RM-ODP)**.

El mismo **recopila una serie de conceptos, condiciones y buenas prácticas**, en su mayoría pre existentes, que intentan estandarizar el desarrollo de aplicaciones distribuidas.


La intención es simplificar la definición de sistemas que suelen ser inherentemente complejos.





# Modelo de referencia - **Partes**

El modelo tiene 4 elementos fundamentales para especificar el sistema:

- Se especifica a partir de un **modelado de objetos**.
  - Se especifica a partir de **"viewpoints" separados pero interrelacionados**.
  - Tiene que proveer una serie de **transparencias**.
  - Tiene que tener un framework para **medir la conformidad del sistema mediante la realización de pruebas** (manuales o automáticas) y/o **revisión de cumplimiento de requisitos**.
- 






# Modelo de referencia - Modelado de Objetos

El modelo utiliza el modelado de los sistemas mediante el análisis y diseño orientado a objetos.

Utiliza este paradigma de programación porque entiende que es el más natural al describir una serie de elementos independientes que interactúan entre sí para realizar las tareas y/o operaciones del sistema.





# Modelo de referencia - Viewpoints

Los viewpoints son basicamente documentacion del sistema.

Los sistemas tienden a ser tan complejos y hay tantos involucrados que una descripción única sería larga y poco útil.

Entonces el modelo indica que al sistema se lo puede describir mediante “diferentes viewpoints”. Cada uno de los viewpoints describe desde cierto objetivo y con una audiencia específica en mente.

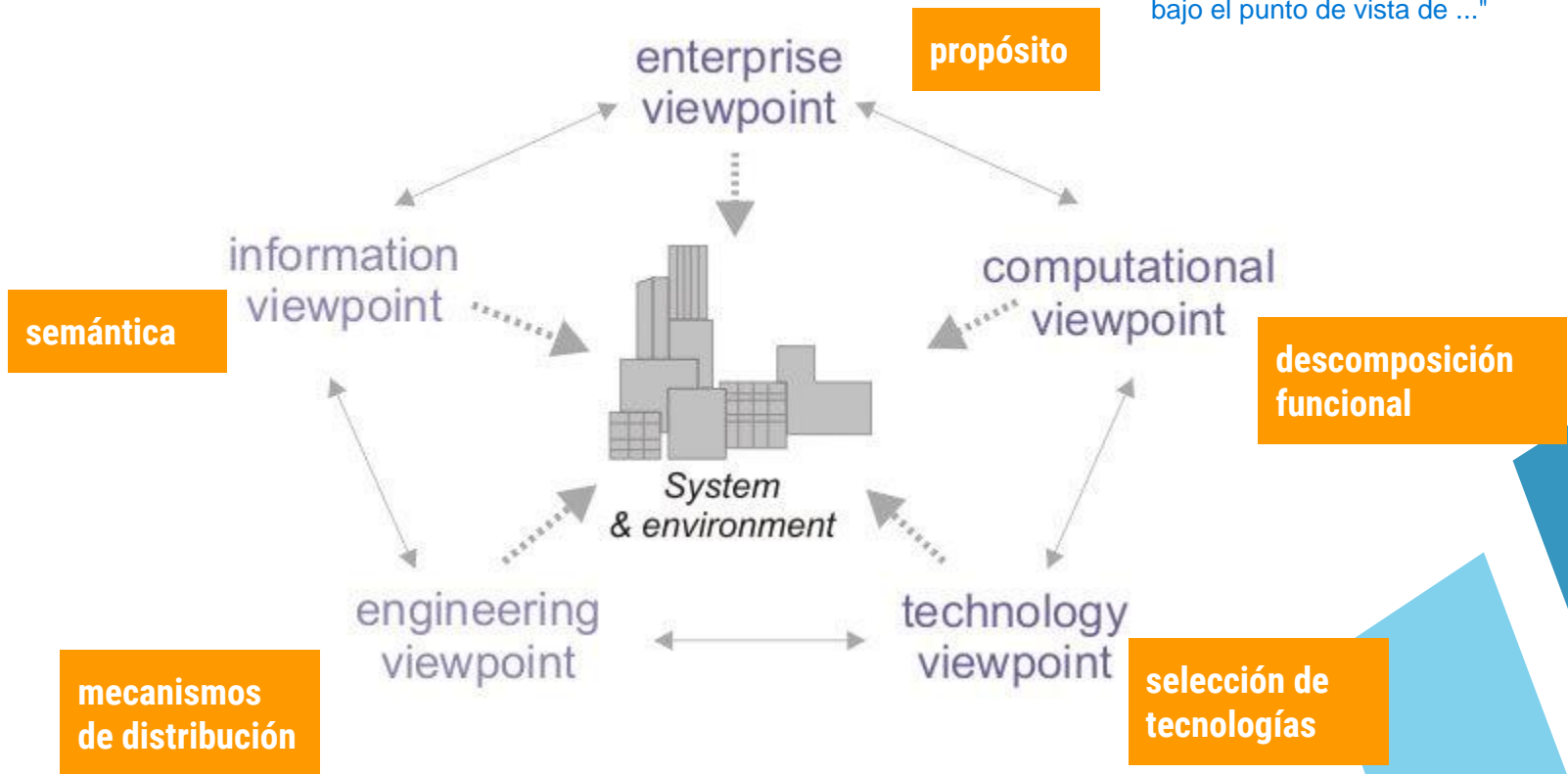
La forma de implementar cada viewpoint no está determinada, se puede hacer documentos de texto, gráficos, diagramas, etc.

Muchos tienen cierta relación con los diagramas UML, así que son naturalmente utilizados para esto.



# Modelo de referencia - Viewpoints

Cada cuadrado naranja es un Viewpoint distinto. Significa "describo mi sistema bajo el punto de vista de ..."





# Modelo de referencia - **Transparencias**

Se trata de hacer transparente para el usuario que esta usando un sistema distribuido.

**RM-ODP** define una serie de transparencias cuyo objetivo es reducir y/o simplificar los potenciales problemas inherentes a la comunicación de sistemas distribuidos.

Se dice que una transparencia se “cumple” si la misma queda oculta (o invisibilizada) para el cliente del sistema.

Si todas las transparencias se cumplen (lo cual es un ideal difícil), quedaría oculto el hecho que estamos realizando una comunicación entre sistemas distribuidos.





# Modelo de referencia - **Transparencias**

## **ACCESO**


Los usuarios del sistema deben acceder a los componentes del mismo de una manera uniforme.

Esconde la **complejidad de invocar** servicios remotos y la **heterogeneidad de representación**.

## **FALLAS**

Los errores, reintentos y recoveries deben pasar desapercibidos por el usuario.

Acceso: tengo que acceder al sistema distribuido como si fuera un solo sistema  
Fallas: si hay fallas en la red, el usuario ni se entera



# Modelo de referencia - Transparencias

## UBICACIÓN

El usuario no debe saber la ubicación física de los componentes del sistema.

Provee un **acceso lógico independiente de la ubicación** física del servicio.

## MIGRACIÓN

Si un objeto cambia de ubicación entre usos el usuario no debe notarlo.

## REUBICACIÓN

Si un objeto cambia de ubicación mientras se está usando el usuario no debe notarlo.




# Modelo de referencia - Transparencias

## REPLICACIÓN

Si un componente está replicado en múltiples nodos, para el usuario deben aparentar ser el mismo componente.

## TRANSACCIÓN

Oculta la **interacción de varios objetos** para proveer una operación transaccional.

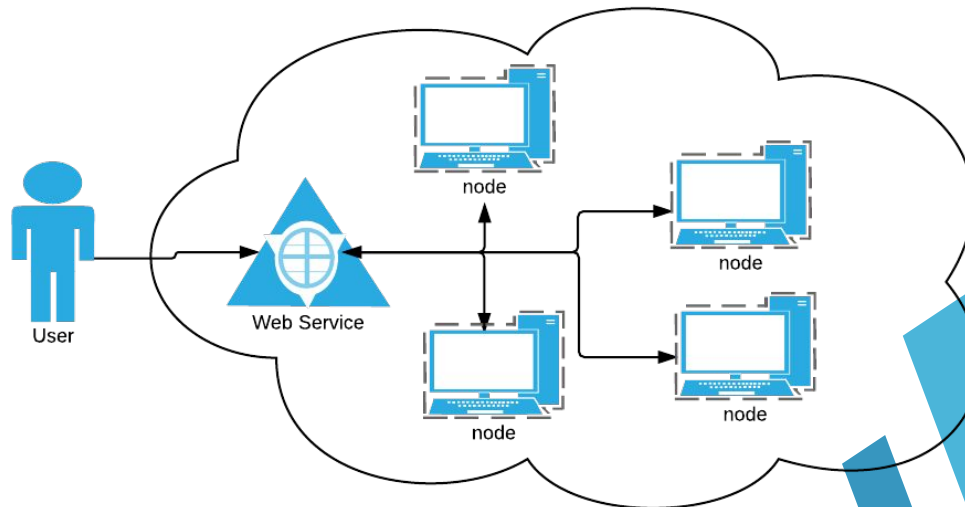


# Sistemas Distribuidos

Para el trabajo distribuido escalado horizontalmente, todavía **queremos mantener las transparencias**, especialmente la de acceso. Esto significa **trabajar con un servicio sin saber** (o sabiendo lo menos posible) de **que el mismo es distribuido**.

Y hablamos de “sistemas”, porque no es solo una aplicación funcionando distribuida (por ejemplo una base NoSql), sino un conjunto de sistemas que interactúan para proveer un servicio para el cliente, sin que él mismo sepa de sus partes.

Generalmente, los mismos frameworks y herramientas se encargan de sumar transparencias.





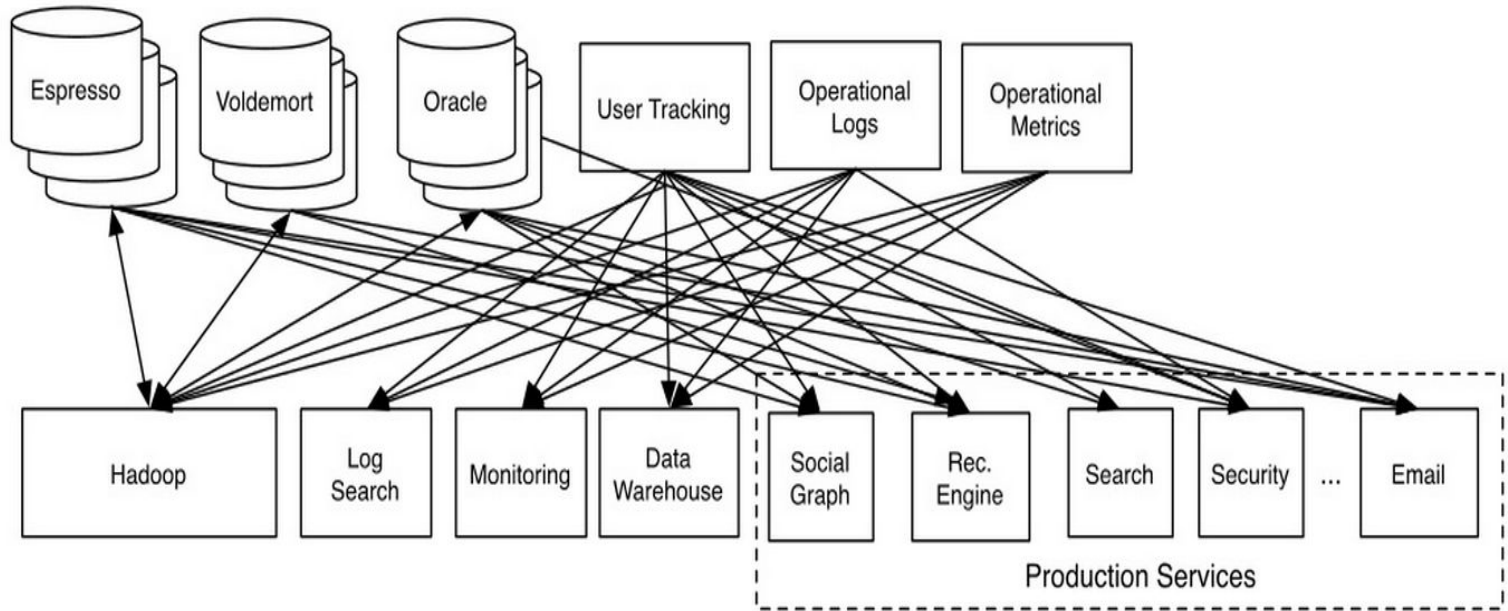


# Ejemplos



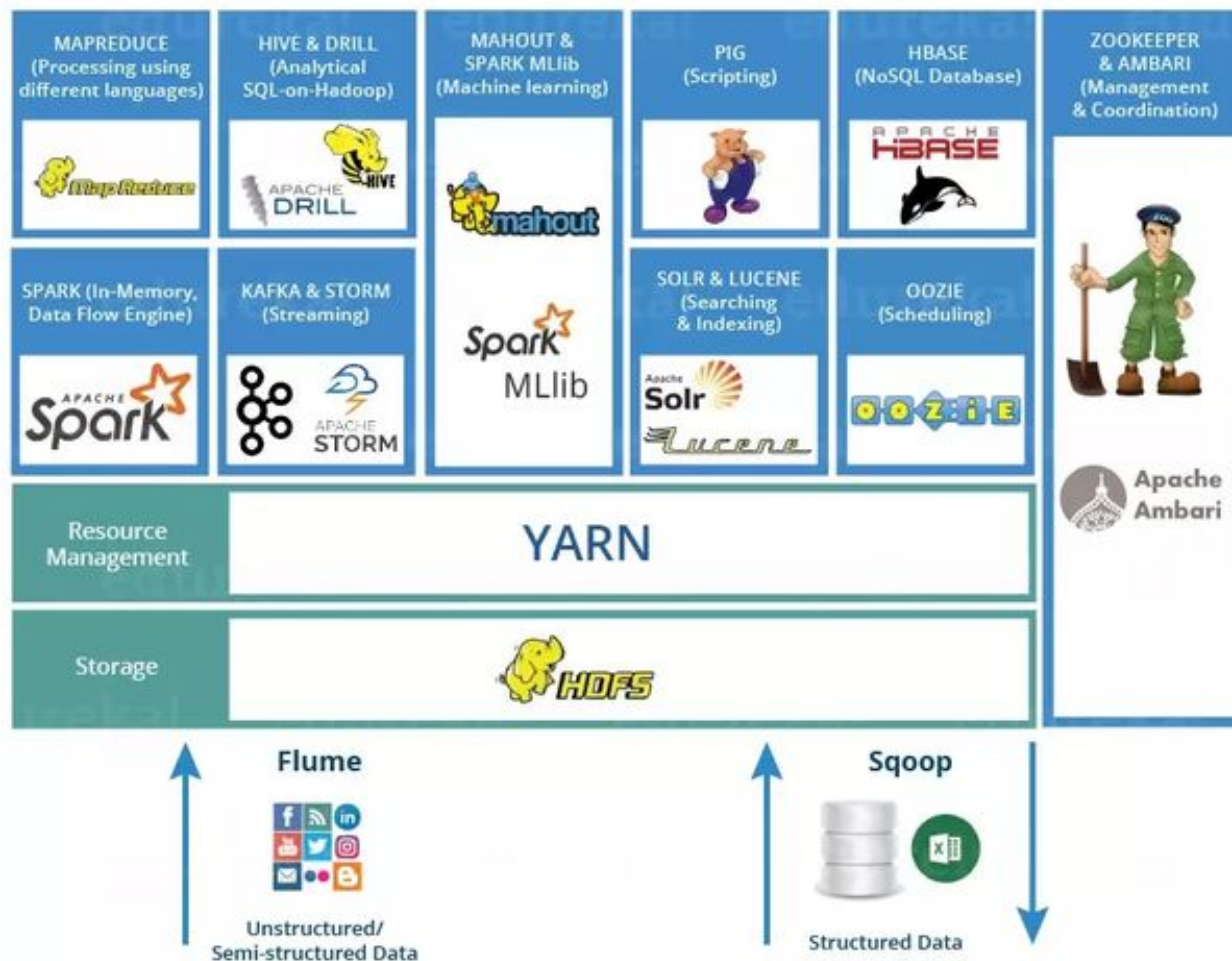
# Sistemas Distribuidos

Arquitectura de microservicios de LinkedIn pre kafka



# Sistemas Distribuidos

## Stack de Hadoop

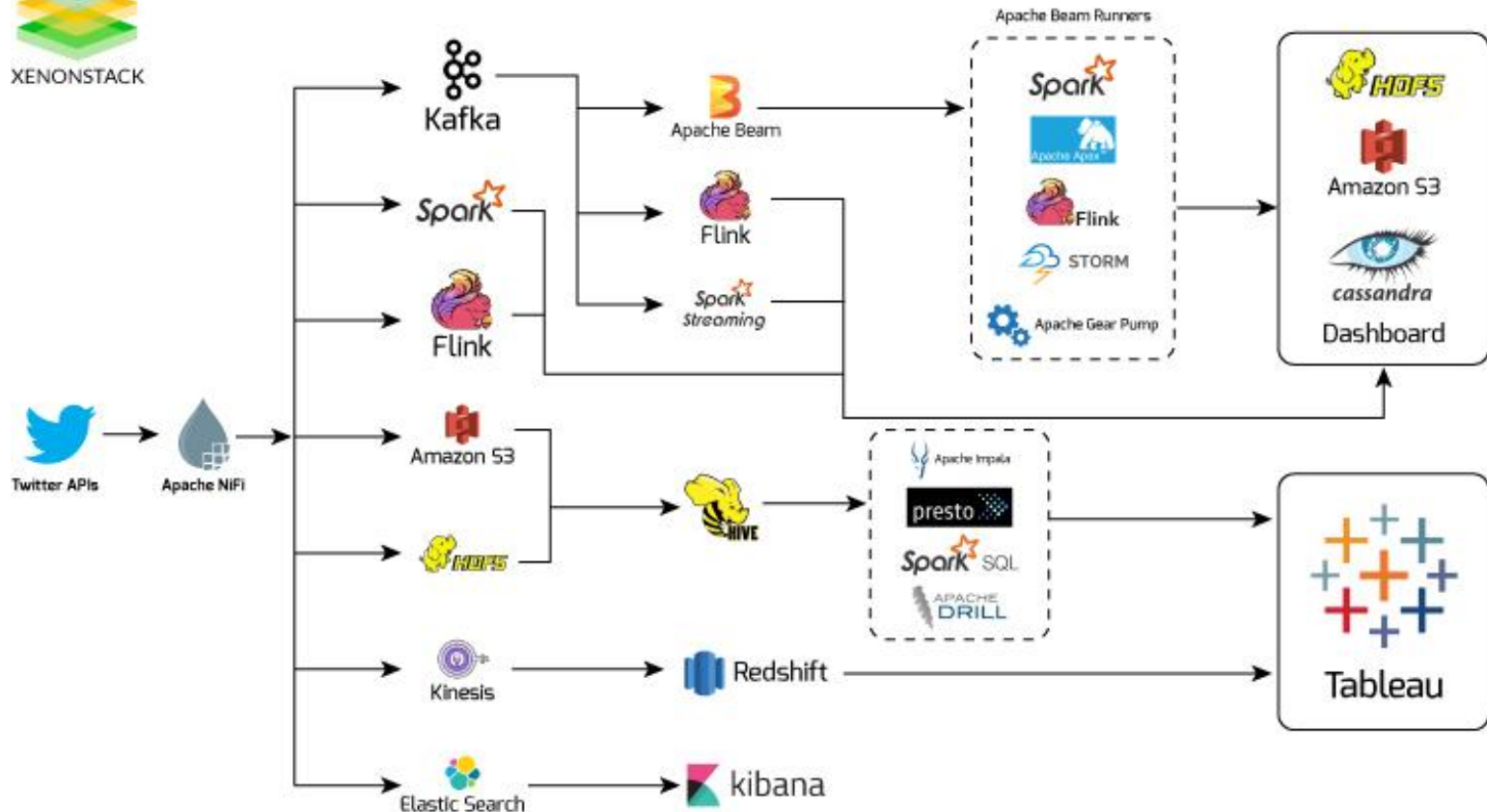


# Sistemas Distribuidos

Sistema de captura y procesamiento analítico utilizando Twitter como fuente

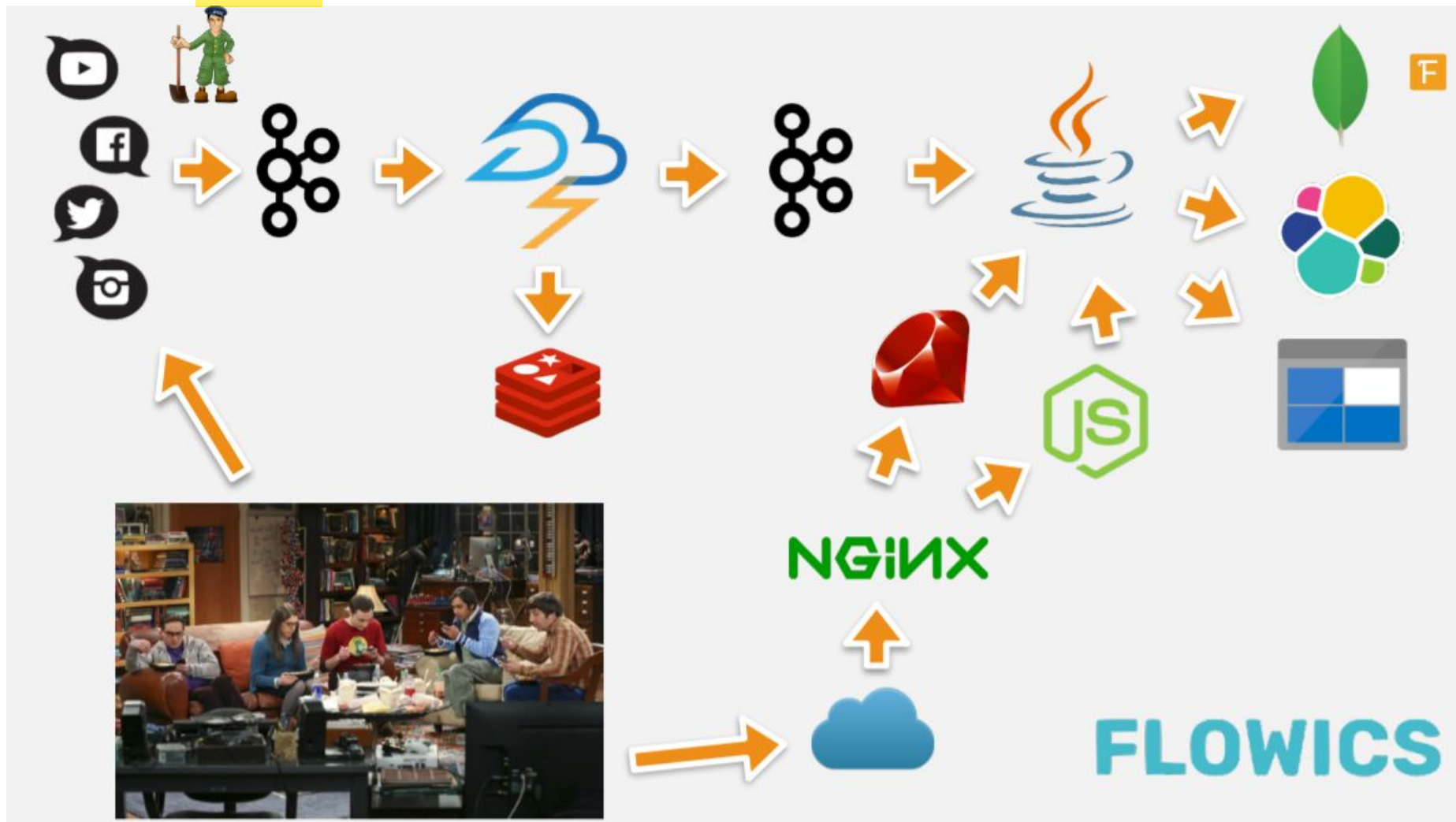


## Building Data Lake with Apache NiFi



# Sistemas Distribuidos

Stack de Flowics





**Store distribuido**

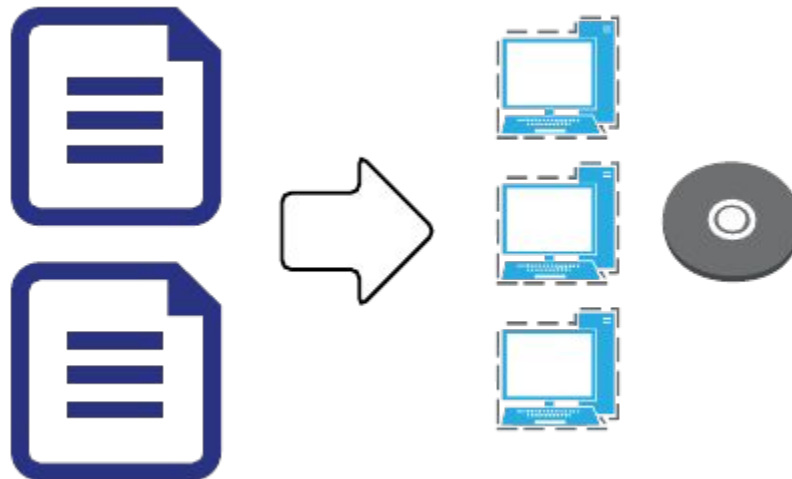
# Almacenamiento Distribuido

Lo que se quiere resolver al utilizar almacenamiento distribuido es poder almacenar información que por su volumen no entra en un nodo servidor.

Este volumen puede estar dado porque:

- » Un elemento por sí solo no entra en un nodo.
- » El volumen de elementos es tal, que la suma de sus tamaños supera los nodos.

El particionado suele ser mas facil en el segundo de los casos






# Almacenamiento Distribuido

Se hace mención de "elementos" porque existen herramientas y frameworks que permiten distribuir diferentes tipos de valores, que dan la oportunidad de utilizar abstracciones mayores a la hora de guardar.

Por lo cual se encuentran *stores* distribuidos de diferentes tipos:

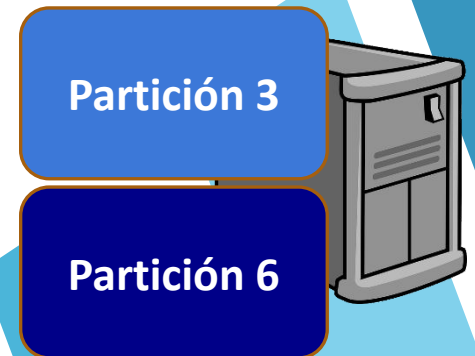
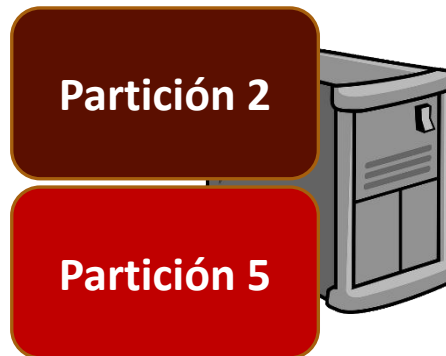
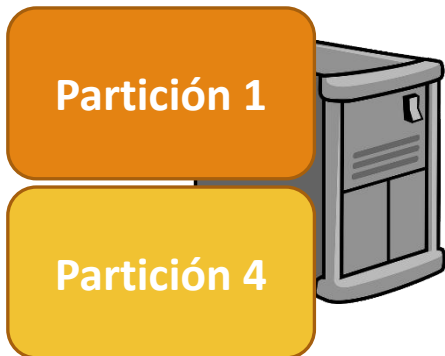
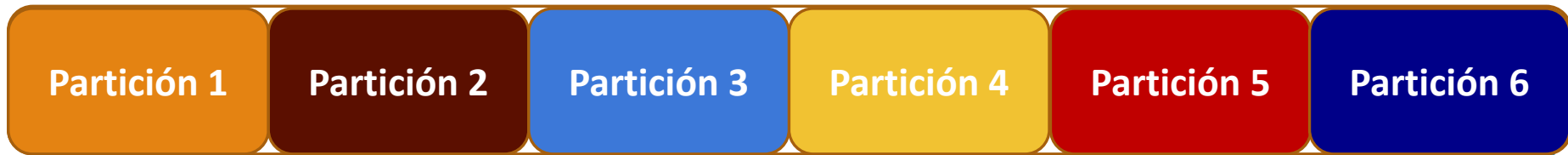
- » File Systems
  - » Bases de Datos
  - » Colecciones de Objetos.
- 



# Particionado

El primer inconveniente a resolver es **cómo distribuir la información dentro del cluster.**

Teniendo varios nodos dentro del cluster lo más lógico es "dividir" la tarea (almacenar) entre los nodos. O sea particionar la información.






# Particionado - Distribución

Una buena partición de los datos nos permite

- » Realizar una escritura más rápida y eficiente.
- » Que el lookup del dato sea más rápido.

Tener en cuenta tiene sus contras:

- » Hay que coordinar el cluster para saber dónde va cada partición y a cuál partición va cada valor.
  - » De elegir mal la estrategia de partición, los datos pueden quedar sesgados en uno o más servidores, haciendo que la carga de los mismos lo haga ineficiente.
- 

# Replicación

Tener una copia de cada partición en el sistema es peligroso, porque **si se cae un nodo, la información queda corrupta** (esa partición dejó de existir)

Para resolver este potencial problema **se tienen varias copias** de cada partición dentro del cluster o sea se **replican las particiones**.

Partición 1

Partición 2

Partición 3

Partición 4

Partición 5

Partición 6

Obviamente no poner la particion y la replica en el mismo nodo

Partición 1

Réplica  
Partición 6

Partición 2

Réplica  
Partición 4

Partición 3

Réplica  
Partición 1

Partición 4

Réplica  
Partición 2

Partición 5

Réplica  
Partición 3

Partición 6

Réplica  
Partición 5

De acuerdo a las necesidades del sistema pueden ser más de una réplica

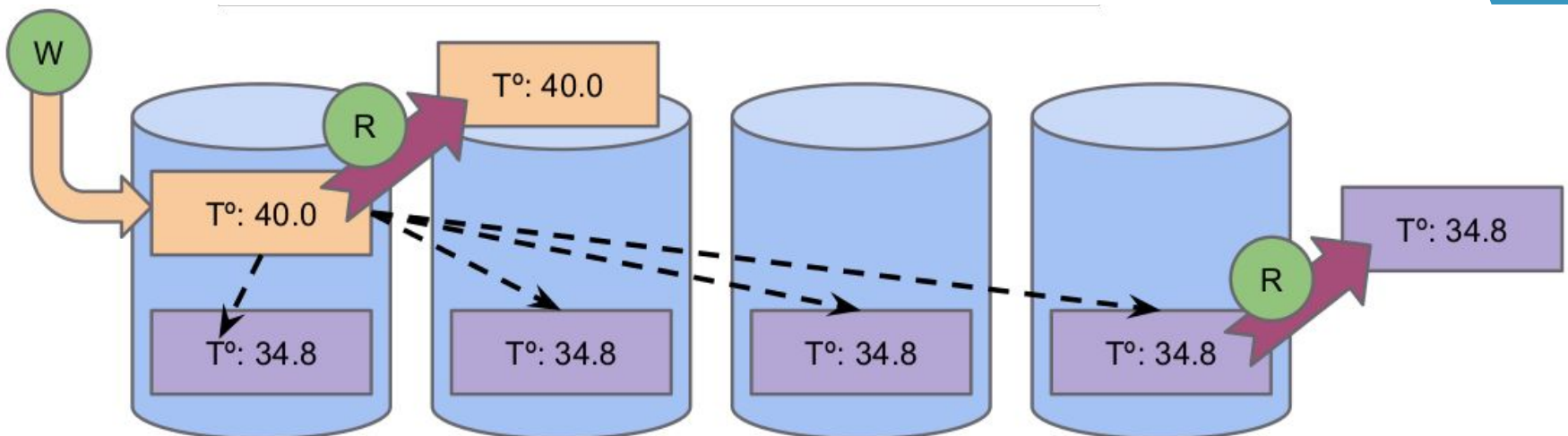
# Consistencia

Al tener **particionado y replicación**, una decisión que hay que tomar es **cuando considero al dato escrito (y cuándo se lo puede leer)**.

Algunas opciones pueden ser:

1. Al momento que se escribió en la partición principal inicial.
2. Cuando hay al menos una réplica escrita + la partición.
3. Cuando estén todas las réplicas escritas.


E inclusive una decisión más a realizar ¿se puede leer o escribir en las réplicas?





# Consistencia


Cualquiera de las soluciones es válida, de acuerdo a las necesidades del sistema a implementar.

- » En el caso de necesitar mucha capacidad de lectura y data consistente (similar a una base de datos ACID), la opción 3 será la elegida.
  - » En cualquier otro caso, 1 y 2 son opciones válidas.
- 



# Consistencia


De la misma manera que se puede configurar la cantidad de réplicas y particiones existen dos factores que indican la preferencia en la consistencia:

- » **Consistencia en escritura:** Indica cuántas copias tienen que reportar que la escritura fue hecha antes de que se le indique al cliente que la escritura está terminada.
  - » **Consistencia en lectura:** Indica cuántas particiones tienen que reportar el mismo valor antes de responder ese valor como cierto en una lectura.
- 



# Modelos de Consistencia

**Los modelos de consistencia se utilizan en sistemas de store distribuidos y de memoria distribuida compartida (DSM).  
Cada modelo indica una serie de reglas para la lectura/escritura de un valor, que asegura que, si el programador sigue las reglas, sus lecturas serán consistentes con las escrituras hechas.**






# Modelos de Consistencia

## Modelo de **consistencia estricto**

Es el modelo ideal, por el cual **toda escritura estará inmediatamente disponible para lectura para todos**. Además se respetará el orden de cada operación de manera global.

## Modelo de **consistencia eventual**

Indica que **luego de realizar una escritura, y de no mediar nuevas escrituras, pasado cierto tiempo todas las lecturas reflejarán el valor escrito**.








# Modelos de Consistencia

Los dos modelos mencionados son los más comunes en store distribuido.

Los modelos de consistencia son muchos y tienen diversos campos de aplicación. Una lista no completa de los mismos se puede hallar en la [wiki del tema](#).

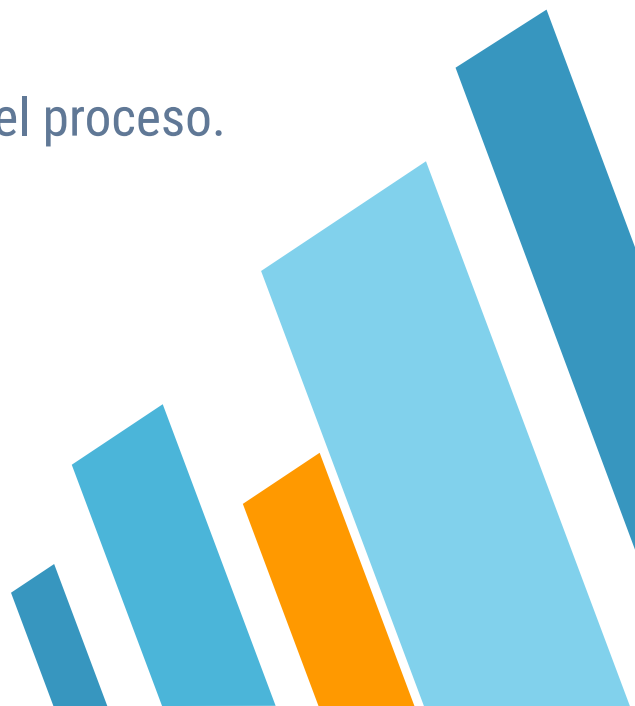




# Elasticidad

La elasticidad es la **capacidad de un cluster de modificar la cantidad de nodos** que lo componen y que éstos puedan participar de la operación del mismo **sin que el cluster quede offline** completamente.

En un cluster de storage se requería que:

- » El cluster detecte al nuevo nodo.
  - » Le asigne particiones primarias y de backup.
  - » Coordine la transferencia de esas particiones.
  - » Lo marque disponible para continuar como parte del proceso.
- 



# Tolerancia a Fallos

Parte de las tareas de controlar es que, **si algún nodo se cae, las particiones** del mismo (primarias y de backup) **sean reasignadas.**

Todo esto **requiere coordinación** y hay varias estrategias para que esto sea posible, en general hay **un nodo master o coordinador** y alguna redundancia (o no).

Lo que varía es si el master es un nodo especial o cualquier nodo puede asumir el rol de coordinador





# CREDITS

Content of the slides:

- » POT - ITBA

Images:

- » POD - ITBA
- » Or obtained from: [commons.wikimedia.org](https://commons.wikimedia.org)

Slides theme credit:

Special thanks to all the people who made and released these awesome resources for free:

- » Presentation template by [SlidesCarnival](https://slidescarnival.com)
  - » Photographs by [Unsplash](https://unsplash.com)
- 