NoSQL

Sesión 1 - **No Sólo SQL**

© 2015-2016 Depto. Ciencia de la Computación e IA

Índice

- No Sólo SQL
  - Características
  - NoSQL vs SQL
  - Implantando NoSQL
  - Modelos NoSQL
  - Consistencia
  - Teorema CAP
- *MongoDB*
  - Elementos
  - Hola *MongoDB*

© 2015-2016 Depto. Ciencia de la Computación e IA

No Sólo SQL

- Grupo de tecnologías que permiten el procesamiento rápido y eficiente de conjuntos de datos dando la mayor importancia al **rendimiento**, la **fiabilidad** y la **agilidad**.
- NoSQL
  - **No** SQL
  - ***Not Only*** *SQL*

- Los sistemas NoSQL se centran en sistemas complementarios a los SGBD relaciones, que fijan sus prioridades en la **escalabilidad** y la **disponibilidad** en contra de la atomicidad y consistencia de los datos.

Tipos de sistemas NoSQL

- Clave-Valor
  - Cada elemento se almacena con un nombre de atributo (o clave) junto a su valor
- Grafos
  - Almacenan información sobre redes, como pueden ser conexiones sociales
- Columnas
  - Los datos se almacenan como columnas, no como filas
  - Cada fila puede contener un número diferente de columnas
- Documental
  - Cada clave almacena un documento con una estructura similar a JSON

key-value

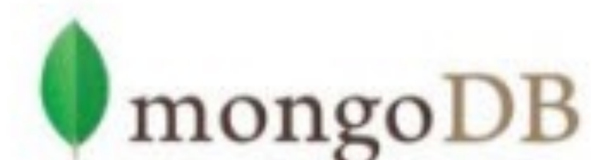
Amazon DynamoDB (Beta)

ORACLE BERKELEY DB 11g

graph

column

document



© 2015-2016 Depto. Ciencia de la Computación e IA

1.1 Características de NoSQL - Beneficios

- Soporte para **grandes volúmenes de datos** estructurados, semi-estructurados y sin estructurar.
  - Permiten comenzar con un modelo sencillo y con el paso del tiempo, añadir nuevos campos, ya sean sencillos o anidados a datos ya existentes.
- **Sprints ágiles**, iteraciones rápidas y frecuentes *commits/pushes* de código.
  - El modelo crece de la mano del código de aplicación
- **Arquitectura eficiente y escalable** diseñada para trabajar con *clusters* de ordenadores
  - más económica
  - transparente para el desarrollador.

© 2015-2016 Depto. Ciencia de la Computación e IA

1.1.1 Esquemas Dinámicos

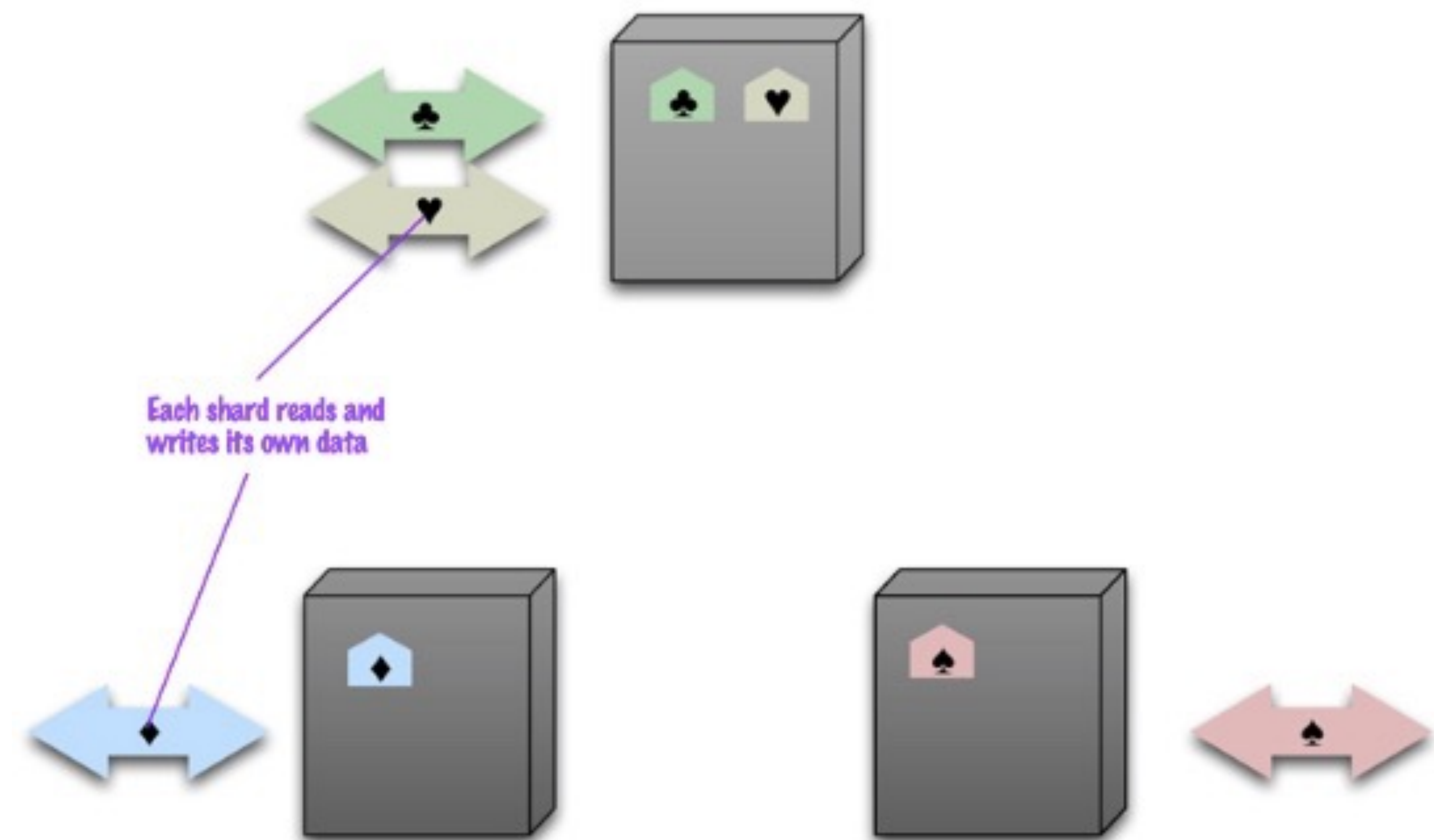
- En BBDD relacionales se definen los esquemas antes de añadir los datos.
- En el desarrollo ágil, al añadir funcionalidad, el esquema se modifica

- En NoSQL, se permite la inserción de datos sin un esquema predefinido.
  - Facilita la modificación de la aplicación en tiempo real
  - No hay interrupciones de servicio
  - Desarrollo más rápido
  - Integración de código más robusto
  - Menos tiempo empleado en la administración de la base de datos.

© 2015-2016 Depto. Ciencia de la Computación e IA

1.1.2 Particionado

- Las bases de datos SQL escalan verticalmente
  - un único servidor que almacena toda la base de datos.
- Los costes se incrementan rápidamente, con un límites definidos por el propio hardware, y con puntos críticos de fallo dentro de la infraestructura de datos.
- Solución: escalar horizontalmente, añadiendo nuevos servidores en vez de concentrarse en incrementar la capacidad de un único servidor.
- *Sharding* o **Particionado**.



Particionando Sistemas SQL

- Conlleva el uso de SANs y código de aplicación para hacer que el hardware actúe como un único servidor.
- Los equipos de desarrollo deben:
  - Almacenar los datos de cada instancia de base de datos de manera autónoma
  - Desarrollar el código de aplicación para distribuir los datos y las consultas y agregar los resultados de los datos a través de todas las instancias de bases de datos
  - Desarrollar código adicional para gestionar los fallos sobre los recursos, para realizar *joins* entre diferentes bases de datos, balancear los datos y/o replicarlos, etc…
- La integridad transaccional se ven comprometidos o incluso eliminados

© 2015-2016 Depto. Ciencia de la Computación e IA

Auto-Sharding

- Dividen los datos entre un número arbitrario de servidores, sin que la aplicación sea consciente de la composición del *pool* de servidores.
- Los datos y las consultas se balancean entre los servidores.

- Métodos de particionado:
  - Por **rangos** de su id: por ejemplo "los usuarios del 1 al millón están en la partición 1" o "los usuarios cuyo nombre va de la A a la E" en una partición, en otra de la M a la Q, y de la R a la Z en la tercera.
  - Por **listas**: dividiendo los datos por la categoría del dato

    libros: novelas en una partición, recetas de cocina en otra, etc..
  - **Función *hash***, la cual devuelve un valor para un elemento que determine a que partición pertenece.

© 2015-2016 Depto. Ciencia de la Computación e IA

Cuando Particionar

- Motivos:
  - **Limitaciones de almacenamiento**: los datos no caben en un único servidor, tanto a nivel de disco como de memoria RAM.
  - **Rendimiento**: al balancear la carga, las escrituras serán más rápidas.
  - **Disponibilidad**: si un servidor esta ocupado, otro servidor puede devolver los datos

    La carga de los servidores se reduce.

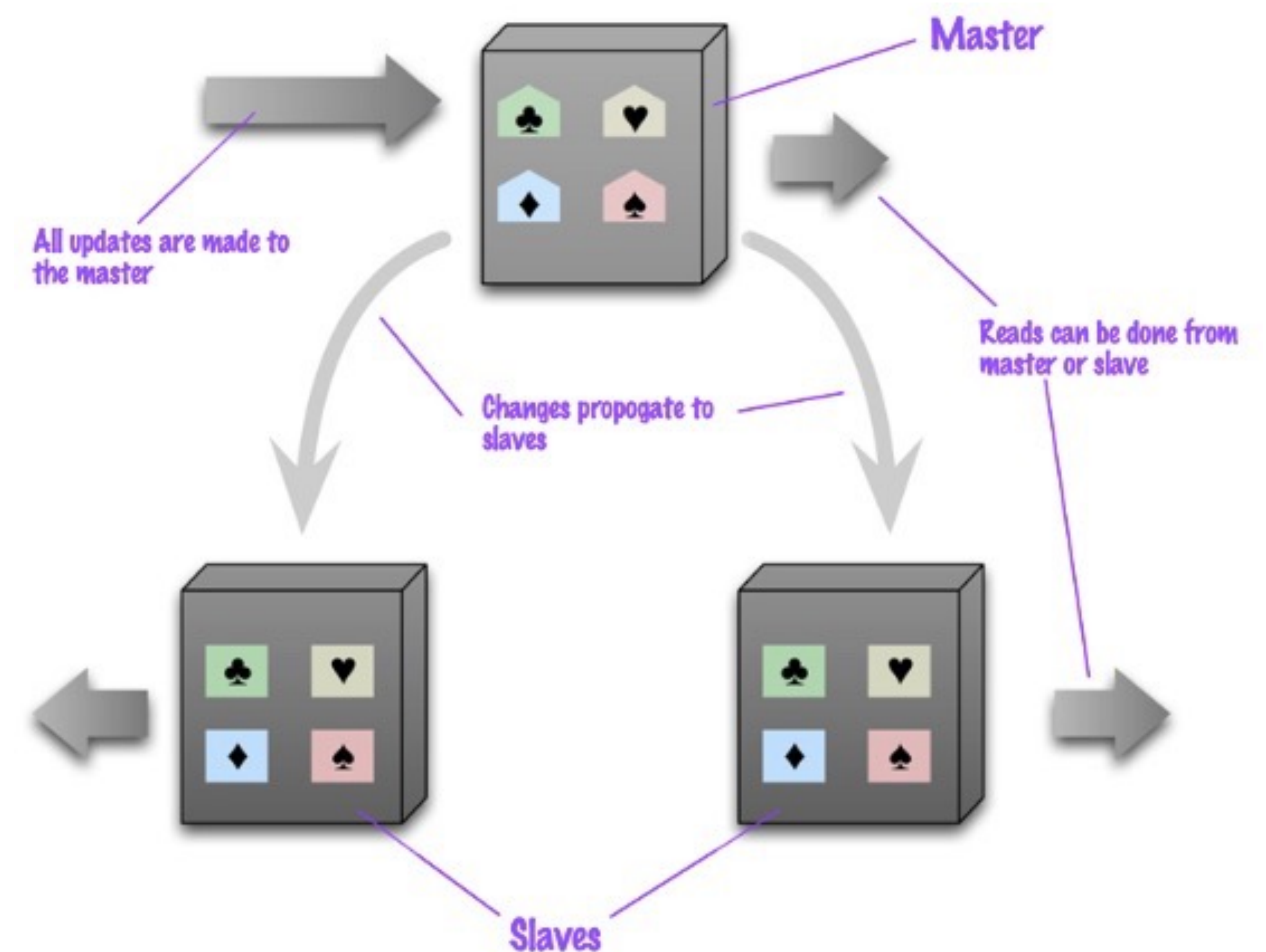
- No particionaremos cuando la cantidad de datos sea pequeña → distribuir los datos conlleva unos costes que pueden no compensar con un volumen de datos insuficiente.
- No esperar a particionar cuando tengamos muchísimos datos → el proceso de particionado puede provocar sobrecarga del sistema.

- Las herramientas *Cloud* facilitan este escalado → capacidad ilimitada bajo demanda

1.1.3 Replicación

- Mantiene copias idénticas de los datos en múltiples servidores
- Facilita aplicaciones robustas, incluso si alguno de los servidores sufre algún problema.

- La mayoría de sistemas NoSQL soportan la replicación automática
- Desde el punto de vista del desarrollador, el entorno de almacenamiento es virtual y ajeno al código de aplicación.



Beneficios de la Replicación

- **Escalabilidad y rendimiento** → distribuye las consultas en diferentes nodos
- **Disponibilidad**, ofreciendo tolerancia a fallos de hardware o corrupción de la base de datos.
  - Al replicar los datos vamos a poder tener una copia de la base de datos, dar soporte a un servidor de datos agregados/informes, o tener nodos a modo de copias de seguridad que pueden tomar el control en caso de caída del nodo principal.
- **Aislamiento** (la i en ACID - *isolation*), entendido como la propiedad que define cuándo y cómo al realizar cambios en un nodo se propagan al resto de nodos.
  - Copias sincronizadas para separar procesos de la base de datos de producción
  - Informes o copias de seguridad en nodos secundarios → elimina el impacto negativo en el nodo principal
  - Sistema sencillo para separar el entorno de producción del de preproducción.

© 2015-2016 Depto. Ciencia de la Computación e IA

Caché integrada

- Los datos se mantiene en memoria y se persisten de manera periódica.
- La **consistencia** de los datos puede no ser completata → consistencia eventual.

- Existen productos que ofrecen caché para los sistemas SQL.
  - Incrementan el rendimiento de las lecturas de manera sustancial, pero no mejoran el de las escrituras
  - Añaden un capa de complejidad al despliegue del sistema.
  - Por ej: si en una aplicación predominan las lecturas, una cache distribuida como *MemCached* puede ser una buena solución.

© 2015-2016 Depto. Ciencia de la Computación e IA

1.2 NoSQL vs SQL I

	Bases de Datos SQL	Bases de Datos NoSQL
Tipos / Modelos	Modelo relacional, con algunas variantes por producto	Muchos tipos: almacenes clave-valor, base de datos documentales, basado en columnas o en grafos
Historia	Desarrollado en los 70 para tratar la primera hornada de aplicaciones que almacenaban datos	Desarrollado en el 2000 para resolver las limitaciones de las bases de datos SQL, particularmente la escalabilidad, replicación y almacenamiento de datos desestructurados
Ejemplos	MySQL, PostgreSQL, Oracle	MongoDB, Cassandra, HBase, Neo4j
Modelo de almacenamiento	Los registros individuales se almacenan como filas en tablas, donde cada columna almacena un atributo específico.	Varía dependiendo del tipo de base de datos, empleando estructuras similares a mapas o documentos JSON.
Esquemas	La estructura y los tipos de datos se conocen de antemano. Para almacenar un nuevo atributo hay que modificar la base de datos, tiempo durante el cual el sistema queda inactivo.	Normalmente dinámicos. Los registros pueden añadir información en caliente y almacenar información con diferente estructura en un mismo campo.

© 2015-2016 Depto. Ciencia de la Computación e IA

NoSQL vs SQL II

	Bases de Datos SQL	Bases de Datos NoSQL
Escalado	Vertical: un único servidor que debe incrementar su potencia para tratar la demanda creciente. Se pueden repartir bases de datos SQL entre muchos servidores, pero requiere una configuración especial y desarrollo teniendo en cuenta esta configuración	Horizontal: para añadir capacidad, se añaden más servidores virtuales o instancias en la nube, de manera transparente a la aplicación. Los datos se propagan de manera automática entre los diferentes servidores añadidos.
Modelo de Desarrollo	Repartido entre *open source* (p.ej. *Postgres*, *MySQL*) y software propietario (p.ej., *Oracle*, *DB2*)	*Open source*
Soporte de transacciones	Si, las modificaciones se pueden configurar para que se realizan todas o ninguna	En algunas circunstancias y ciertos niveles (a nivel de documento *vs* a nivel a de base de datos)
Manipulación de Datos	SQL	Mediante APIs orientadas a objetos
Consistencia	Configurable para consistencia alta/estricta	Depende del producto. Algunos ofrecen gran consistencia (p.ej. *MongoDB*) mientras otros ofrecen consistencia eventual (p.ej. *Cassandra*)

© 2015-2016 Depto. Ciencia de la Computación e IA

1.3 Implantando NoSQL

- Comenzar con una prueba de baja escalabilidad (un único nodo) NoSQL
  - Familiarizarse con la tecnología asumiendo muy poco riesgo.
- La mayoría son *open-source* → *sin inversión económica inicial*
- Al tener unos ciclos de desarrollo más rápidos, facilitan innovar con mayor velocidad y mejorar la experiencia de sus cliente a un menor coste.

- Factores de decisión:
  - Escalabilidad y/o rendimiento superior a las capacidades del sistema existente
  - Identificar alternativas viables respecto al software propietario
  - Incrementar la velocidad y agilidad del proceso de desarrollo

© 2015-2016 Depto. Ciencia de la Computación e IA

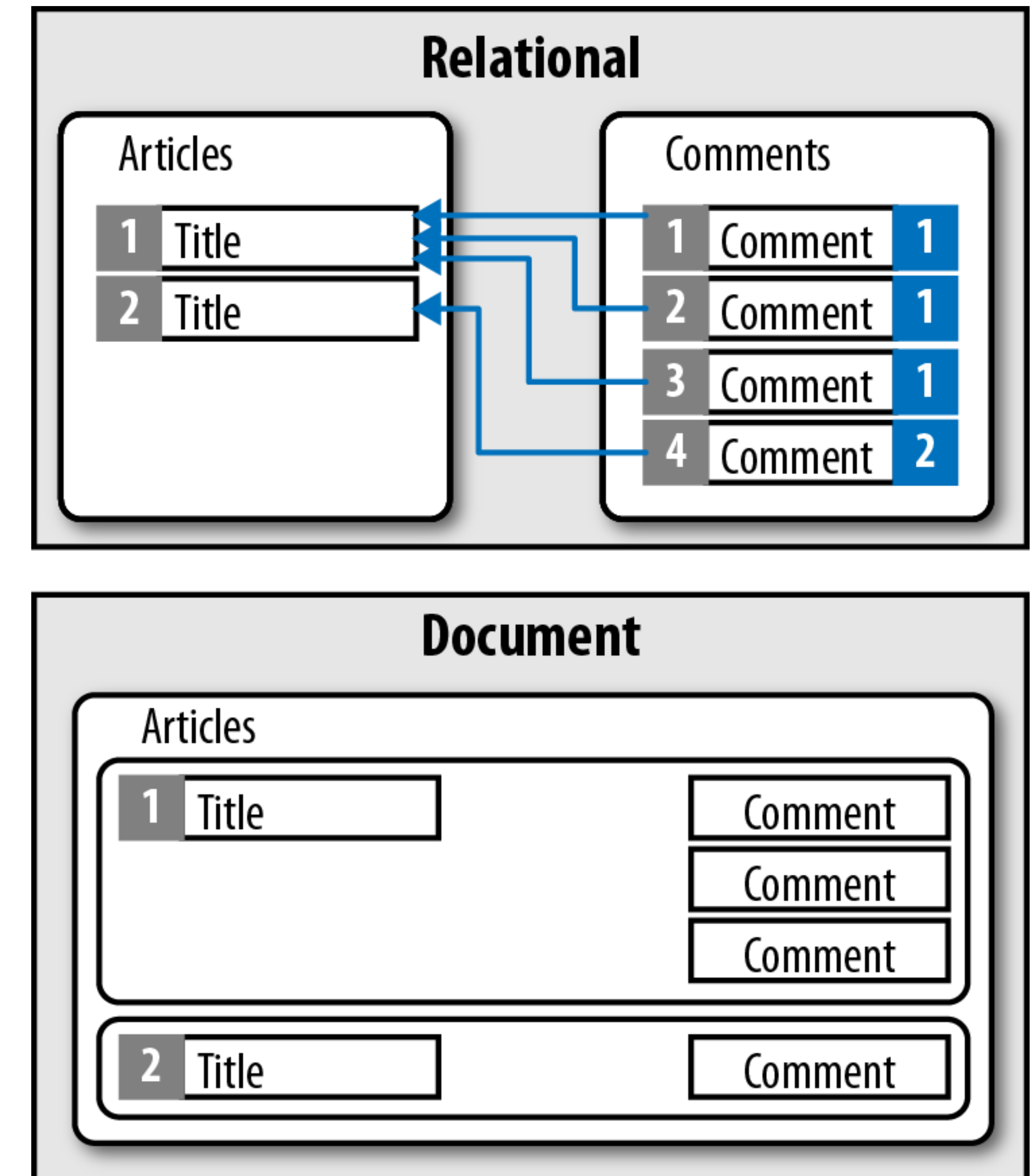
Elección de Solución NoSQL

- **Modelo de Datos**: A elegir entre un modelo documental, basado en columnas, de grafos o mediante clave-valor.
- **Modelo de Consultas**: Por clave primaria, por un determinado atributo/campo.
  - Consultas sencillas o complejas.
  - Uso de índices
- **Modelo de Consistencia**: La replicación conlleva que los datos puedan estar obsoletos. Los sistemas NoSQL tienden a ser consistentes o eventualmente consistentes.
- **APIs**: No existe un estándar para interactuar con los sistemas NoSQL. Cada sistema presenta diferentes diseños y capacidades para los equipos de desarrollo.
  - La madurez de un API puede suponer una inversión en tiempo y dinero a la hora de desarrollar y mantener el sistema NoSQL.
- **Soporte** Comercial y de la Comunidad.
  - Resolución de problemas, Perfiles de desarrollares, Retención del talento

© 2015-2016 Depto. Ciencia de la Computación e IA

1.4 Modelos de Datos. Bases de Datos Documentales

- En vez de filas y columnas, la información se almacena en documentos.
- Estructura similar a JSON
- Se agrupan en colecciones o bases de datos
- Contienen uno o más campos → cada campo contiene un valor con un tipo (cadena, fecha, binario o array).
- En vez de repartir los datos entre múltiples columnas y tablas, cada registro y sus datos asociados se almacenan de manera unida en un único documento.
  - Simplifica el acceso a los datos y reduce (y en ocasiones elimina) la necesidad de *joins* y transacciones complejas.



Bases de Datos Documentales: Características

- El **esquema es dinámico**: cada documento puede contener diferentes campos.
  - Permite modelar datos desestructurados y polimórficos
- Cada documento contiene un elemento clave, sobre el cual se puede obtener un documento de manera unívoca.
- Mecanismo completo de consultas → cualquier campo del documento.
- Algunos productos ofrecen opciones de indexado
  - Índices compuestos, dispersos, con tiempo de vida (TTL), únicos, de texto o geoespaciales.
- Ofrecen productos que permiten analizar los datos, mediante funciones de agregación o implementación de *MapReduce*.
- Los documentos (y sus datos anidados) se pueden modificar mediante una única sentencia.

```
{
  "EmpleadoID": "BW1",
  "Nombre"    : "Bruce",
  "Apellido"  : "Wayne",
  "Edad"      : 35,
  "Salario"   : 10000000
}

{
  "EmpleadoID": "JK1",
  "Nombre"    : "Joker",
  "Edad"      : 34,
  "Salary"    : 5000000,
  "Direccion" : {
    "Lugar"   : "Asilo Arkham",
    "Ciudad"  : "Gotham"
  },
  "Proyectos" : [
    "desintoxicacion-virus",
    "top-secret-007"
  ]
}
```

 © 2015-2016 Depto. Ciencia de la Computación e IA

Bases de Datos Documentales: Casos de Uso

- Propósito general:
  - Amplio abanico de aplicaciones → flexibilidad del modelo de datos
  - Consultas sobre cualquier campo
  - Modelar de manera natural, similar a la Programación Orientada a Objetos.

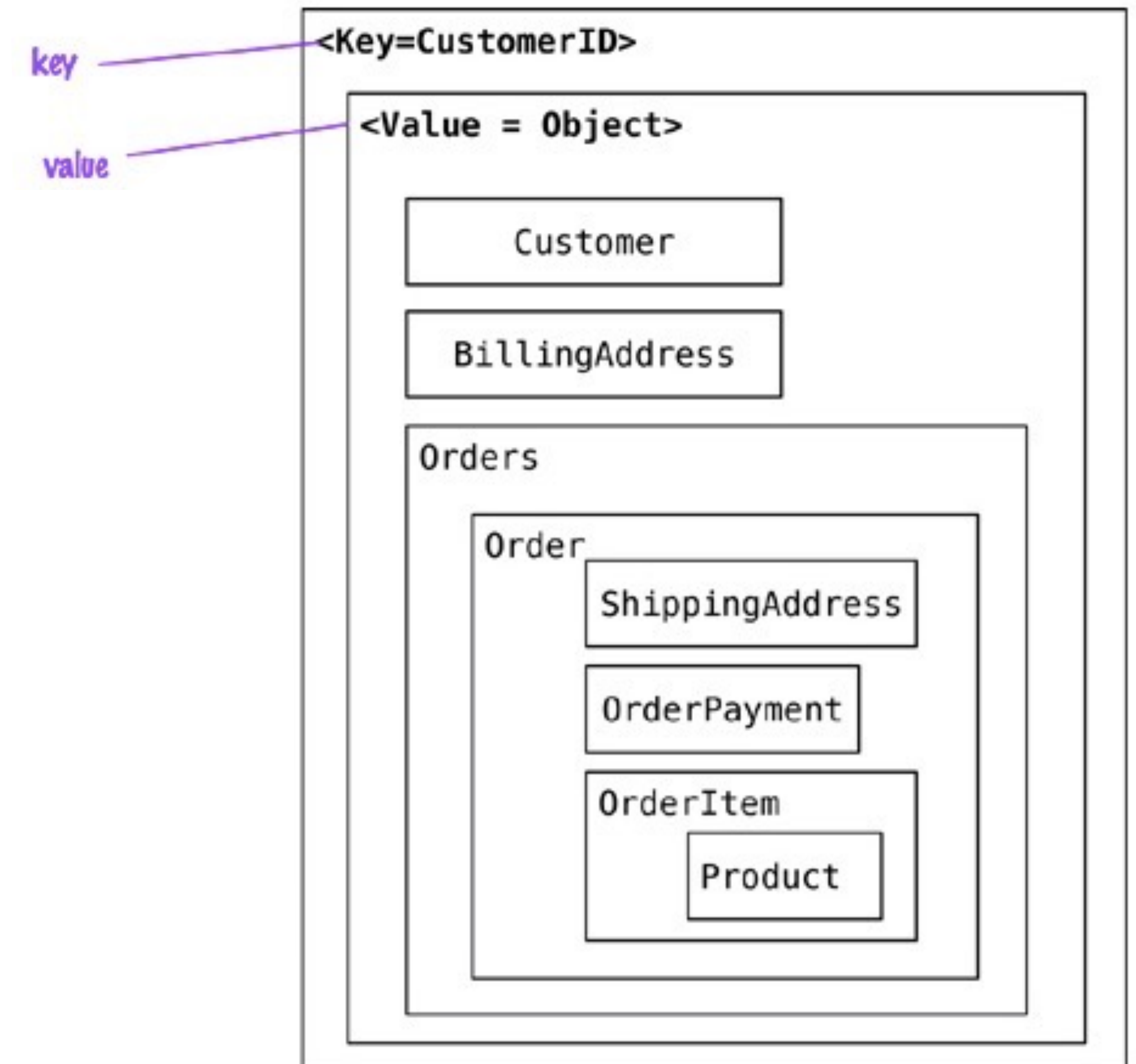
- Casos de Uso:
  - Sistemas de flujo de eventos
  - Gestores de Contenido, plataformas de Blogging
  - Analíticas Web, datos en Tiempo Real
  - Aplicaciones *eCommerce*

- **MongoDB**: http://www.mongodb.com
- CouchDB: http://couchdb.apache.org

 © 2015-2016 Depto. Ciencia de la Computación e IA

1.4.2 Clave-Valor

- Mapa *hash* donde todos los accesos a la base de datos se realizan a través de la clave primaria.
- Sistema NoSQL más básico.
- Similar a tener una tabla relacional con dos columnas `[id,valor]`
  - el valor puede ser de un dato simple o un objeto.
- Operaciones:
  - obtener el valor por la clave
  - asignar un valor a una clave
  - eliminar una clave del almacén.
- El valor es opaco al sistema → los datos sólo se pueden consultar por la clave
  - La aplicación es responsable de saber qué hay almacenado en cada valor.



© 2015-2016 Depto. Ciencia de la Computación e IA

Clave-Valor: Operaciones

Riak →

```
Bucket bucket = getBucket(bucketName);
IRiakObject riakObject = bucket.store(key, value).execute();

IRiakObject riakObject = bucket.fetch(key).execute();
byte[] bytes = riakObject.getValue();
String value = new String(bytes);
```

```
curl -v -X PUT http://localhost:8091/riak/heroes/ace -H "Content-Type:
application/json" -d {"nombre" : "Batman", "color" : "Negro"}
```

- Algunos almacenes clave-valor, p.ej *Redis*, permiten:
  - almacenar datos mediante diferentes estructura → listas, conjuntos, hashes
  - realizar operaciones como intersección, unión, diferencia y rango.

Redis →

```
SET nombre "Bruce Wayne"      // String
HSET heroe nombre "Batman"    // Hash – set
HSET heroe color "Negro"
SADD "heroe:amigos" "Robin" "Alfred"   // Set – create/update
```

- Pueden permitir un segundo nivel de consulta o definir más de una clave

© 2015-2016 Depto. Ciencia de la Computación e IA

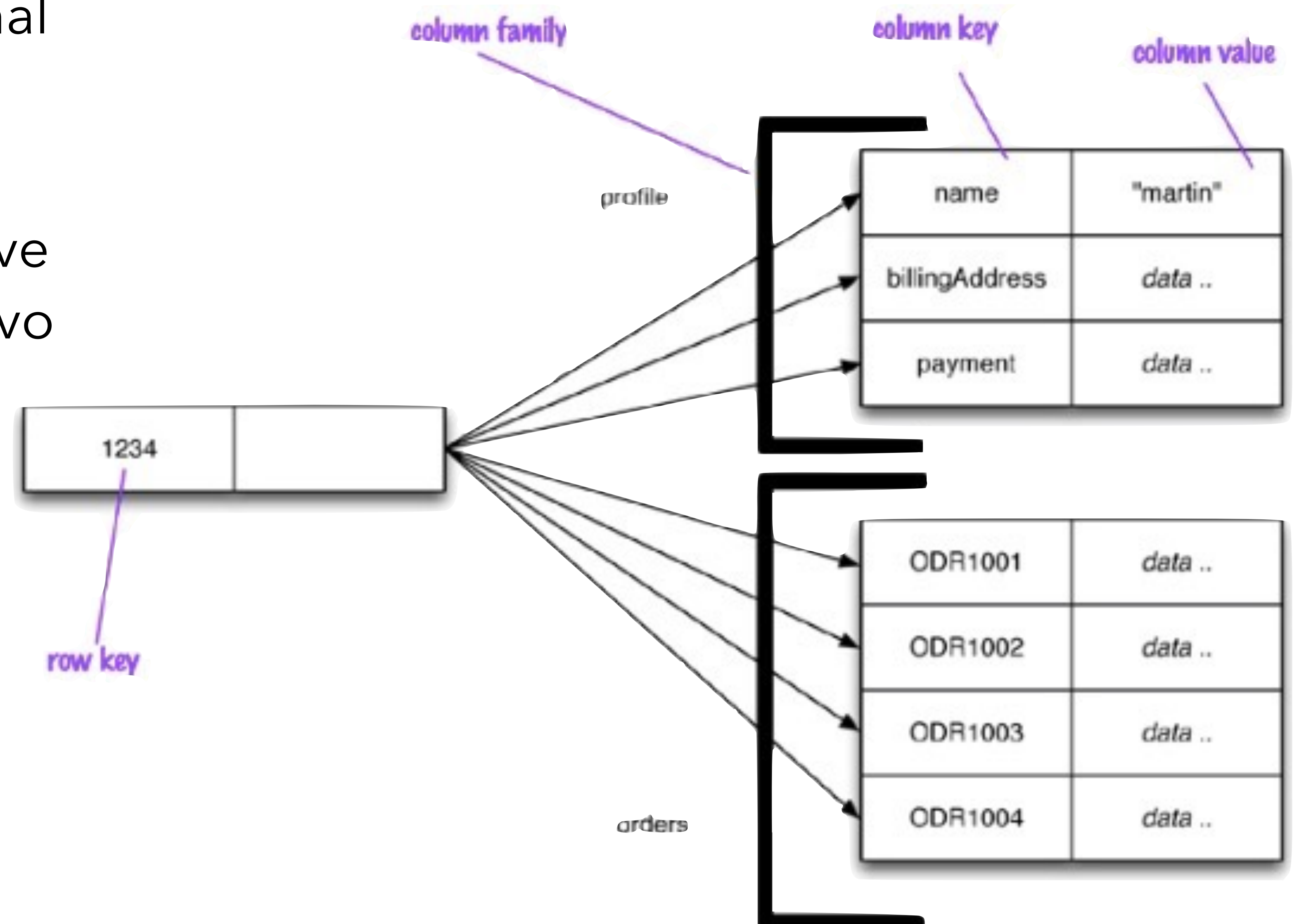
Clave-Valor: Prestaciones y Casos de Uso

- Al usar sólo accesos por clave primaria → gran rendimiento y fácilmente escalables.
  - Rendimiento máximo → toda la información en memoria y se serializa de manera periódica, a costa de una consistencia eventual de los datos.
- Casos de uso para almacenar:
  - Información sobre la sesión de navegación (`sessionid`)
  - Perfiles de usuario, preferencias
  - Datos del carrito de la compra
  - Caché de datos
- No utilizar cuando queremos realizar:
  - Relaciones entre datos
  - Transacciones entre varias operaciones
  - Consultas por los datos del valor
  - Operaciones con conjuntos de claves
- Almacenes más destacados:
  - ***Riak***: http://basho.com/riak/
  - ***Redis***: http://redis.io
  - *Voldemort*: https://github.com/voldemort/voldemort implementación open-source de *Amazon DynamoDB*

© 2015-2016 Depto. Ciencia de la Computación e IA

1.4.3 Basados en Columnas

- Sistemas *Big Data* o tabulares → *BigTable* de *Google*
- Utilizan un mapa ordenado multi-dimensional y distribuido para almacenar los datos mediante 2 niveles:
  - Nivel 1: almacén clave-valor, siendo la clave el identificador de la fila, y el valor un nuevo mapa con los datos agregados de la fila (familias de columnas).
  - Nivel 2: Los valores son las columnas.
- Pensados para que cada fila tenga una gran número de columnas (± 1.000.000), almacenando las diferentes versiones que tenga una fila (±1.000.000.000).



© 2015-2016 Depto. Ciencia de la Computación e IA

Basados en Columnas: Familia de Columnas

- Columna → pareja `name-value`, más atributo `timestamp` para expirar datos y resolver conflictos de escritura.
- Fila → colección de columnas agrupadas a una clave.
- Familia de columnas → Agrupación de filas relacionadas a las que se accede de manera conjunta
  - Cada registro puede almacenar diferente número de columnas.
- Super-columna → columnas anidadas dentro de otras, donde el valor es un nuevo mapa de columnas.
  - Al utilizar super columnas para crear familias de columnas → familia de super columnas.

- En resumen, los datos se almacenan en familias de columnas como filas, las cuales tienen muchas columnas asociadas al identificador de una fila.

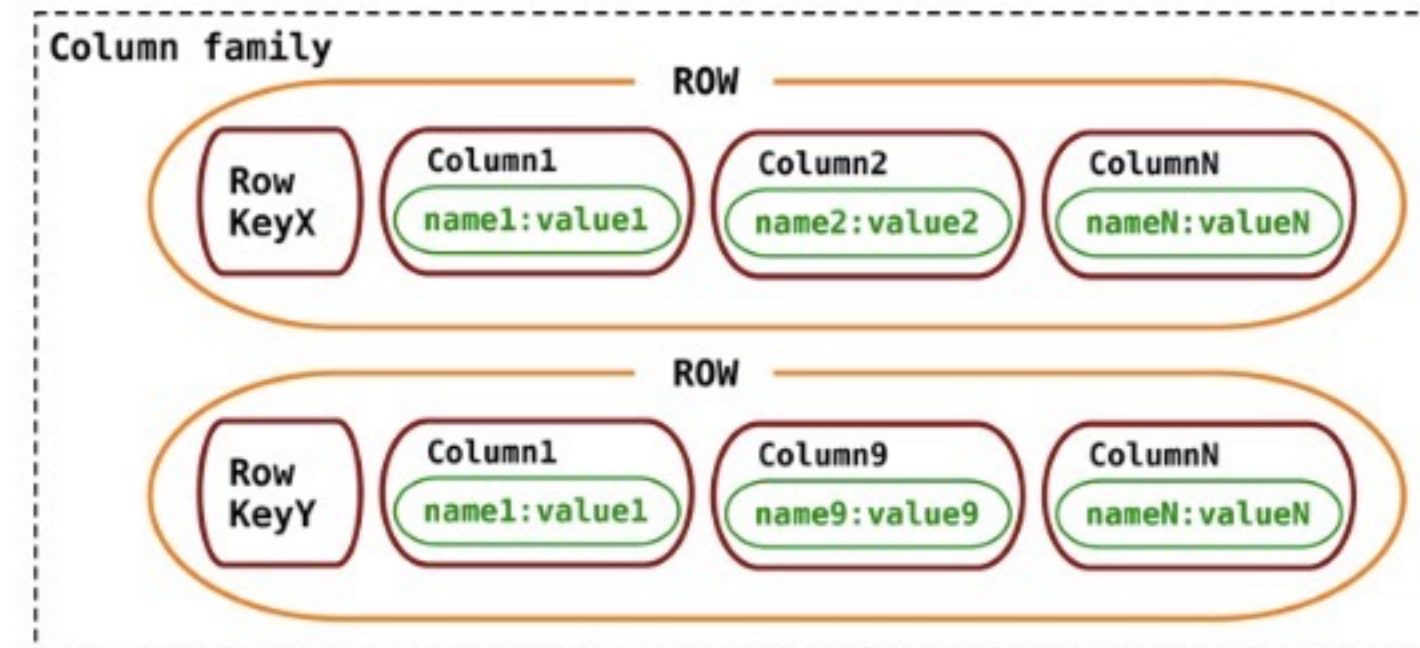
```
// columna
{
  name: "nombre",
  value: "Bruce",
  timestamp: 12345667890
}
```

```
// familia de columnas
{
  // fila
  "tim-gordon" : {
    nombre: "Tim",
    apellido: "Gordon",
    ultimaVisita: "2015/12/12"
  }
  // fila
  "bruce-wayne" : {
    nombre: "Bruce",
    apellido: "Wayne",
    lugar: "Gotham"
  }
}
```

```
{
  name: "libro:978-84-16152-08-7",
  value: {
   // super-columna
    autor: "Grant Morrison",
    titulo: "Batman - Asilo Arkham",
    isbn: "978-84-16152-08-7"
  }
}
```


© 2015-2016 Depto. Ciencia de la Computación e IA

Basados en Columnas: Operaciones



- Consultas por la clave primaria de la familia.
  - Podemos obtener toda una familia, o la columna de una familia:

```
// Mediante Cassandra
GET Clientes['bruce-wayne'];  // familia
GET Clientes['bruce-wayne']['lugar']; // columna
```

- Algunos productos ofrecen soporte limitado para índices secundarios, pero con restricciones.
  - *Cassandra* ofrece el lenguaje CQL similar a SQL pero sin *joins*, ni subconsultas donde las restricciones de `where` son sencillas:

```
SELECT * FROM Clientes
SELECT nombre,email FROM Clientes
SELECT nombre,email FROM Clientes WHERE lugar='Gotham'
```

- Actualizaciones en dos pasos: primero encontrar el registro y segundo modificarlo.
- Una modificación puede suponer una re-escritura completa del registro independientemente que hayan cambiado unos pocos bytes del mismo.

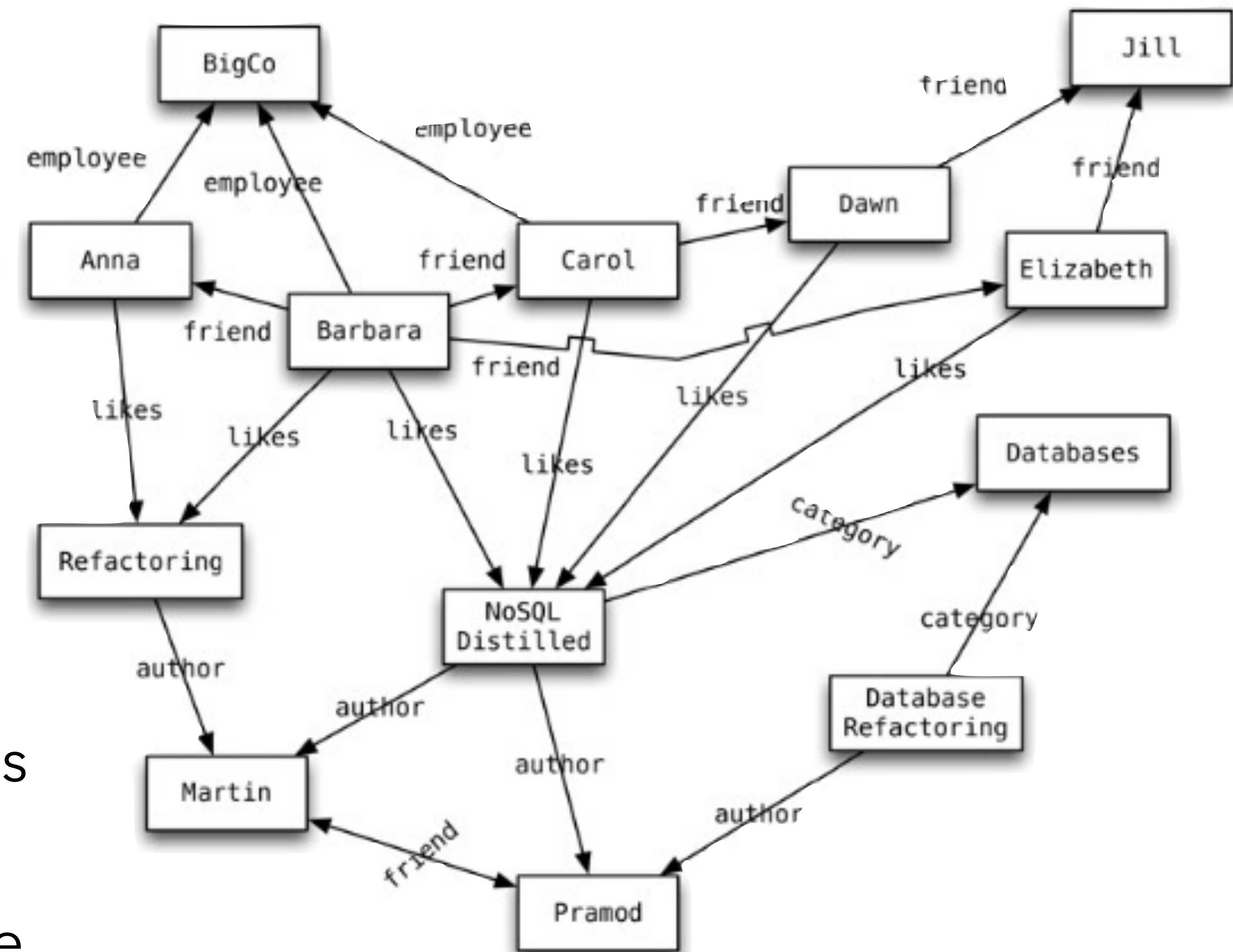
Basados en Columnas: Casos de Uso

- Aplicaciones que necesitan consultar los datos por un único valor
  - Grandes volúmenes de datos, alto rendimiento y escalabilidad
- Casos de uso:
  - Sistemas de flujo de eventos: estados de las aplicaciones, errores.
  - Gestores de Contenido, plataformas de *Blogging* → mediante familias de columnas
  - Contadores: para poder almacenar las visitas de cada visitante a cada apartado de un *site*
- No utilizar en:
  - Sistemas operacionales con transacciones complejas o consultas agregadas
  - Prototipado inicial o sistemas donde el esquema no esté fijado de antemano
- Productos más destacados:
  - *HBase* : http://hbase.apache.org, el cual se basa en ***Hadoop*** - http://hadoop.apache.org
  - ***Cassandra*** : http://cassandra.apache.org
  - *Amazon SimpleDB*: http://aws.amazon.com/simpledb

© 2015-2016 Depto. Ciencia de la Computación e IA

1.4.4 Basados en Grafos

- Almacenan entidades y las relaciones entre ellas.
- Las entidades (*aka* nodos) tienen propiedades.
  - Similar a una instancia de un objeto.
- Las relaciones (*aka* vértices) a su vez tienen propiedades, y su sentido es importante.
  - Para incluir bidireccionalidad tenemos que añadir dos relaciones en sentidos opuestos
- Se organizan mediante relaciones que facilitan encontrar patrones de información existente entre los nodos.
  - Los datos se almacenan una vez y se interpretan de diferentes maneras dependiendo de sus relaciones.



© 2015-2016 Depto. Ciencia de la Computación e IA

Basados en Grafos: *Traversing*

- *Traversing* → Recorrer el grafo para realizar una consulta
- Podemos cambiar los requisitos de *traversing* sin tener que cambiar los nodos o sus relaciones.
- Recorrer las relaciones es muy rápido → no se calculan en tiempo de consulta, sino que se persisten
- Modelo rico de consultas
  - Permite investigar las relaciones simples y complejas entre los nodos para obtener información directa e indirecta.

```
Node martin = graphDb.createNode();
martin.setProperty("name", "Martin");
Node pramod = graphDb.createNode();
pramod.setProperty("name", "Pramod");

martin.createRelationshipTo(pramod, FRIEND);
pramod.createRelationshipTo(martin, FRIEND);
```

```
Node martin = nodeIndex.get("name", "Martin").getSingle();
allRelationships = martin.getRelationships(Direction.INCOMING);
```


© 2015-2016 Depto. Ciencia de la Computación e IA

Basados en Grafos: Casos de Uso

- Modelo no muy intuitivo y con una importante curva de aprendizaje
- Casos de Uso:
  - Datos conectados: redes sociales con diferentes tipos de conexiones entre los usuarios.
  - Enrutamiento, entrega o servicios basados en la posición: si las relaciones almacenan la distancia entre los nodos → consultas sobre lugares cercanos, trayecto más corto, etc…
  - Motores de recomendaciones: de compras, de lugares visitados, etc…
- No usar cuando necesitemos modificar todos o un subconjunto de entidades
  - Modificar una propiedad en todos los nodos es una operación compleja.
- Productos más destacados:
  - *Neo4j*: http://neo4j.com
  - *FlockDB*: https://github.com/twitter/flockdb
  - *HyperGraphDB*: http://www.hypergraphdb.org/index

 © 2015-2016 Depto. Ciencia de la Computación e IA

1.5 Consistencia

- En un sistema **consistente**, las escrituras son visibles en posteriores consultas.
- Con una **consistencia eventual**, las escrituras no son visibles inmediatamente, pero lo serán en un breve periodo de tiempo (ventana de inconsistencia)
  - Ejemplo: En un sistema de control de stock, si el sistema ofrece:

    Consistencia → cada consulta obtendrá el estado real del inventario,

    Consistencia eventual → puede no ser el estado real en un momento concreto pero terminará siéndolo.

- Cada aplicación tiene diferentes requisitos para la consistencia de los datos → posibilidad de asumir consistencia eventual

- Las bases de datos documentales y basadas en grafos son consistentes, pero se pueden configurar para ser eventualmente consistentes.

© 2015-2016 Depto. Ciencia de la Computación e IA

Consistencia Eventual

- Hay un período de tiempo en el que todas las copias de los datos no están sincronizados.
- Aceptable para
  - aplicaciones de sólo-lectura
  - almacenes de datos que no cambian frecuentemente, como los archivos históricos.
  - aplicaciones con alta tasa de escritura donde las lecturas sean poco frecuentes, como un archivo de *log*.
- Los almacenes de clave-valor y los basados en columnas son sistemas eventualmente consistentes → soportan conflictos en las actualizaciones de registros individuales.
  - Como las escrituras se pueden aplicar a cualquier copia de los datos, puede ocurrir, y no sería muy extraño, que hubiese un conflicto de escritura.

 © 2015-2016 Depto. Ciencia de la Computación e IA

Consistencia Eventual - Conflictos

- Enfoques respecto a conflictos en las escrituras:
  - *Riak* utiliza vectores de reloj para determinar el orden de los eventos → la operación más reciente gana en caso de un conflicto.
  - *CouchDB* retiene todos los valores conflictivos y permiten al usuario resolver el conflicto.
  - *Cassandra* sencillamente asume que el valor más grande es el correcto.

- Las escrituras se comportan bien en sistemas eventualmente consistentes
- Las actualizaciones conllevan sacrificios que complican la aplicación.

© 2015-2016 Depto. Ciencia de la Computación e IA

1.6 Teorema de CAP (*Eric Brewer* - 2009)

- Se puede crear una base de datos distribuida con **dos de** estas **tres** características:
  - **Consistencia**: las escrituras son atómicas y todas las peticiones consecuentes obtienen el nuevo valor actualizado, independientemente del origen de la petición.
  - **Disponibilidad** (Available): la base de datos devolverá siempre un valor.
  - **Tolerancia a Particiones**: el sistema funcionará incluso si la comunicación con un servidor se interrumpe de manera temporal.

- Posibilidades:
  - Consistente y tolerante a Particiones (CP).
  - Disponible y tolerante a Particiones (AP).
  - Consistente y Disponible (CA).

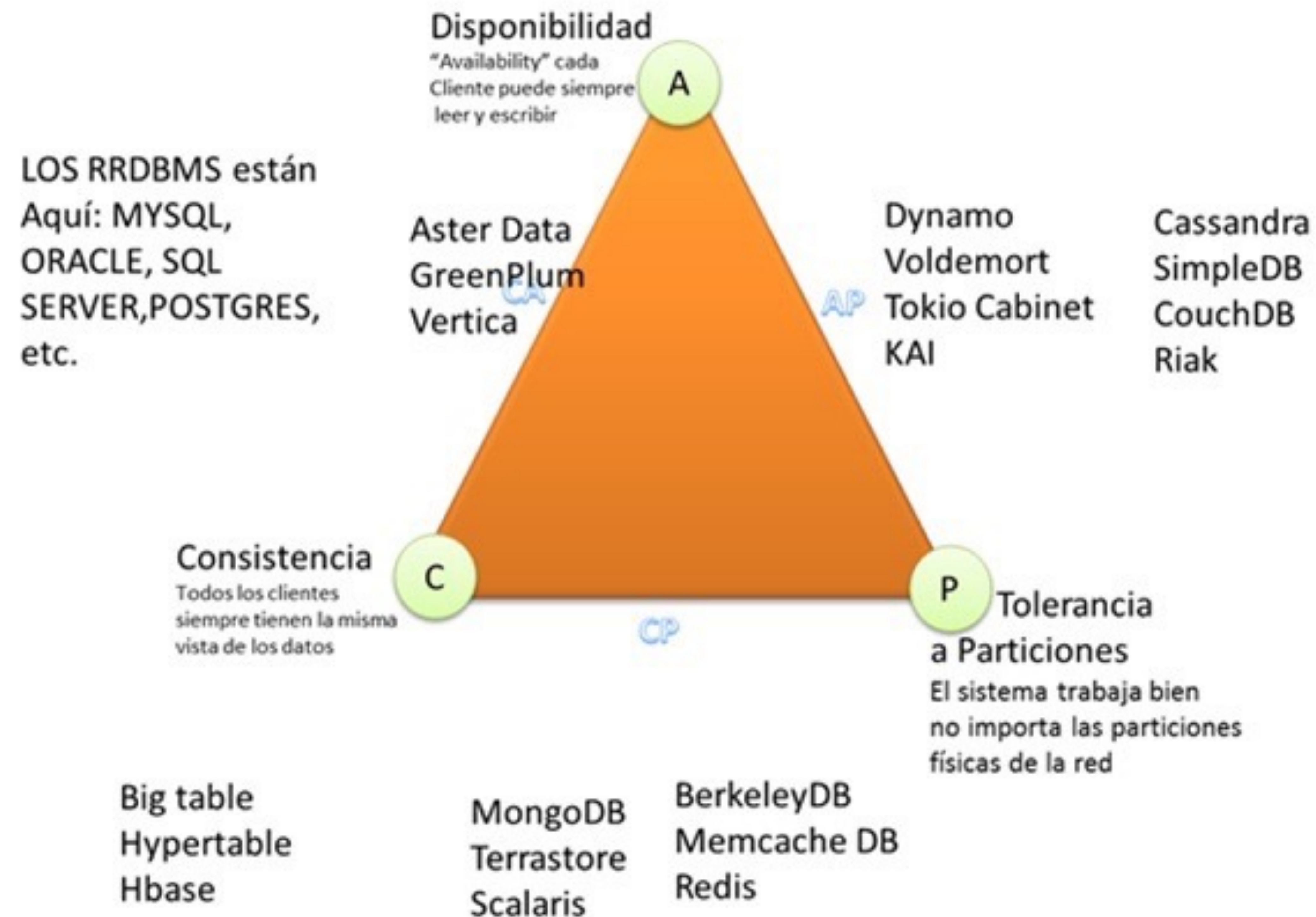
© 2015-2016 Depto. Ciencia de la Computación e IA

Uso del Teorema CAP

- Útil cuando consideramos el sistema de base de datos que necesitamos.
- Permite decidir cual de las tres características vamos a descartar.
- La elección realmente se centra entre la disponibilidad y la consistencia, ya que la tolerancia a particiones es una decisión de arquitectura (sea o no distribuida).

- Aunque el teorema dicte que si en un sistema distribuido elegimos disponibilidad no podamos tener consistencia, todavía podemos obtener consistencia eventual.
- Algunas bases de datos tolerantes a particiones (como *Riak* o *MongoDB*) se pueden ajustar para ser más o menos consistentes o disponibles a nivel de petición

Clasificación de Sistemas según CAP



Remember…

Sandra and Woo by Oliver Knörzer (writer) and Powree (artist) – www.sandraandwoo.com

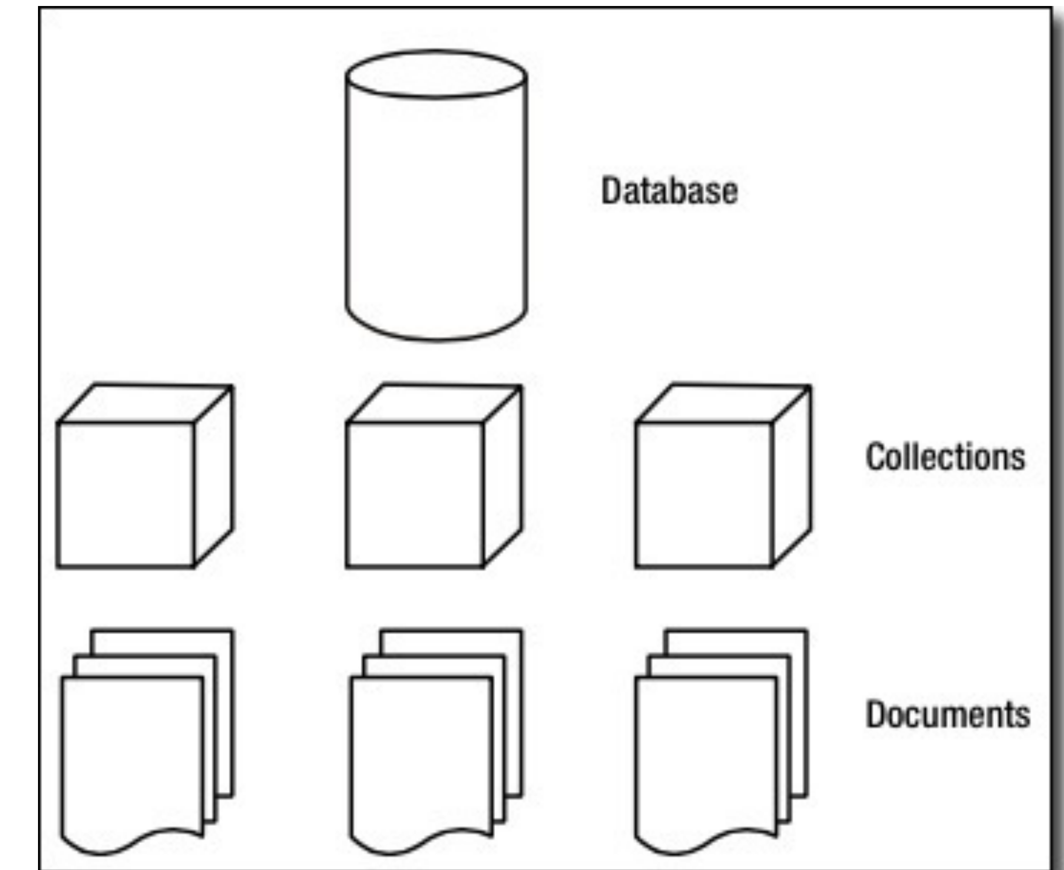
1.7 *MongoDB*

- http://www.mongodb.org
- Posiblemente, la base de datos NoSQL más conocida.
- Modelo documental → los documentos se basan en JSON.
- Marzo 2015 → versión 3.0
- Actualidad → versión 3.2

- Destaca porque:
  - Soporta esquemas dinámicos: diferentes documentos de una misma colección pueden tener atributos diferentes.
  - No soporta *joins*, ya que no escalan bien.
  - No soporta transacciones. Lo que en un SGDB puede suponer múltiples operaciones, con *MongoDB* se puede hacer en una sola operación al insertar/actualizar todo un documento de manera atómica.

© 2015-2016 Depto. Ciencia de la Computación e IA

Elementos de *MongoDB*



- Dentro de una instancia podemos tener 0 o más **bases de dato**s
  - Contenedor de alto nivel
- Cada base de datos tendrá 0 o más **colecciones**.
  - Similar a una tabla.
  - Tipos: normal, limitadas
- Las colecciones contiene 0 o más **documento**s
- Cada documento contiene 0 o más atributos, compuestos de **parejas clave/valor**.
  - No siguen ninguna esquema
  - Dos documentos de una misma colección pueden contener todos los atributos diferentes entre sí.
- Soporta índices para acelerar la búsqueda de datos.
- Al realizar cualquier consulta, se devuelve un cursor
  - Permite contar, ordenar, limitar o saltar documentos.

© 2015-2016 Depto. Ciencia de la Computación e IA

Instalación

- http://www.mongodb.org/downloads
- 64 bits
  - funciona en 32 bits (no recomendable) → restringe el tamaño de los ficheros a 2GB
- Necesita una carpeta para guardar los datos → `/data/db`

```
mkdir -p /data/db
chown `id -u` /data/db
```

- Si no esta instalado como servicio → `mongod`
- Cloud: https://www.mongodb.com/partners/cloud
  - *Amazon Web Services*
  - *MongoLab / Mlab*: https://mongolab.com

```
aitormedrano — mongod — 80×24
MacBook-Air-de-Aitor:~ aitormedrano$ mongod
mongod --help for help and startup options
2015-02-27T10:01:28.446+0100 [initandlisten] MongoDB starting : pid=1351 port=27
017 dbpath=/data/db 64-bit host=MacBook-Air-de-Aitor.local
2015-02-27T10:01:28.447+0100 [initandlisten] db version v2.6.7
2015-02-27T10:01:28.447+0100 [initandlisten] git version: nogitversion
2015-02-27T10:01:28.447+0100 [initandlisten] build info: Darwin miniyosemite.loc
al 14.1.0 Darwin Kernel Version 14.1.0: Fri Dec  5 06:49:27 PST 2014; root:xnu-2
782.10.67~9/RELEASE_X86_64 x86_64 BOOST_LIB_VERSION=1_49
2015-02-27T10:01:28.447+0100 [initandlisten] allocator: tcmalloc
2015-02-27T10:01:28.447+0100 [initandlisten] options: {}
2015-02-27T10:01:28.546+0100 [initandlisten] journal dir=/data/db/journal
2015-02-27T10:01:28.546+0100 [initandlisten] recover : no journal files present,
 no recovery needed
2015-02-27T10:01:29.212+0100 [initandlisten] preallocateIsFaster=true 13.12
2015-02-27T10:01:29.247+0100 [initandlisten] waiting for connections on port 270
17
```


© 2015-2016 Depto. Ciencia de la Computación e IA

Herramientas

- Demonio → `mongod`
- Shell → `mongo`
- Importar / Exportar → `mongoimport` / `mongoexport`

```
mongoimport -d nombreBaseDatos -c coleccion —-file nombreFichero.json
mongoexport -d nombreBaseDatos -c coleccion nombreFichero.json
```

- Backup → `mongodump` / `mongorestore`

```
mongodump -d nombreBaseDatos nombreFichero.bson
mongorestore -d nombreBaseDatos nombreFichero.bson
```

- BSON a JSON → `bsondump`

```
bsondump file.bson > file.json
```

- Rendimiento → `mongostat`
- *Drivers* → http://docs.mongodb.org/ecosystem/drivers/
- GUI → *RoboMongo*: http://robomongo.org/

```
<dependency>
  <groupId>org.mongodb</groupId>
  <artifactId>mongo-java-driver</artifactId>
  <version>2.14.1</version>
</dependency>
```

 © 2015-2016 Depto. Ciencia de la Computación e IA

1.8 Hola MongoDB

```
Last login: Mon Mar 17 18:14:29 on ttys002
MacBook-Air-de-Aitor:db aitormedrano$ mongo
MongoDB shell version: 2.4.9
connecting to: test
Server has startup warnings:
Mon Mar 17 18:20:46.633 [initandlisten]
Mon Mar 17 18:20:46.633 [initandlisten] ** WARNING: soft rlimits too low. Number
 of files is 256, should be at least 1000
> show dbs
jtech   0.203125GB
local   0.078125GB
> use jtech
switched to db jtech
> show collections
grades
students
system.indexes
> db
jtech
>
```

- Lanzar demonio `mongod`
- Lanzar *shell* `mongo`
  - *JavaScript*
- Utilizar base de datos `expertojava`

```
use expertojava
```

- Insertar una persona

```
db.people.insert({ nombre: "Aitor", edad: 38, profesion: "Profesor" })
```

- Recuperar una persona

```
db.people.findOne()
```

```
{
  "_id" : ObjectId("53274f9883a7adeb6a573e64"),
  "nombre" : "Aitor",
  "edad" : 38,
  "profesion" : "Profesor"
}
```


© 2015-2016 Depto. Ciencia de la Computación e IA

¿Preguntas?