

Repositorios de Información

NoSQL

Ingeniería Informática del Software

Escuela de Ingeniería Informática
Universidad de Oviedo

Darío Álvarez Gutiérrez

almacenamiento-en-columnas

Partición
transacciones
map-reduce
consistencia-relajada

HBase

maestro-esclavo

persistencia-políglota

BD-clave-valor

Teorema-CAP

MongoDB

BD-grafo

clusters

BASE

Big-Data

distribución

esquema-flexible

Cassandra

escalabilidad

entre-pares

Hypertable

ACID Disponibilidad

computación-en-la-nube

NoSQL JSON Neo4J

sharding

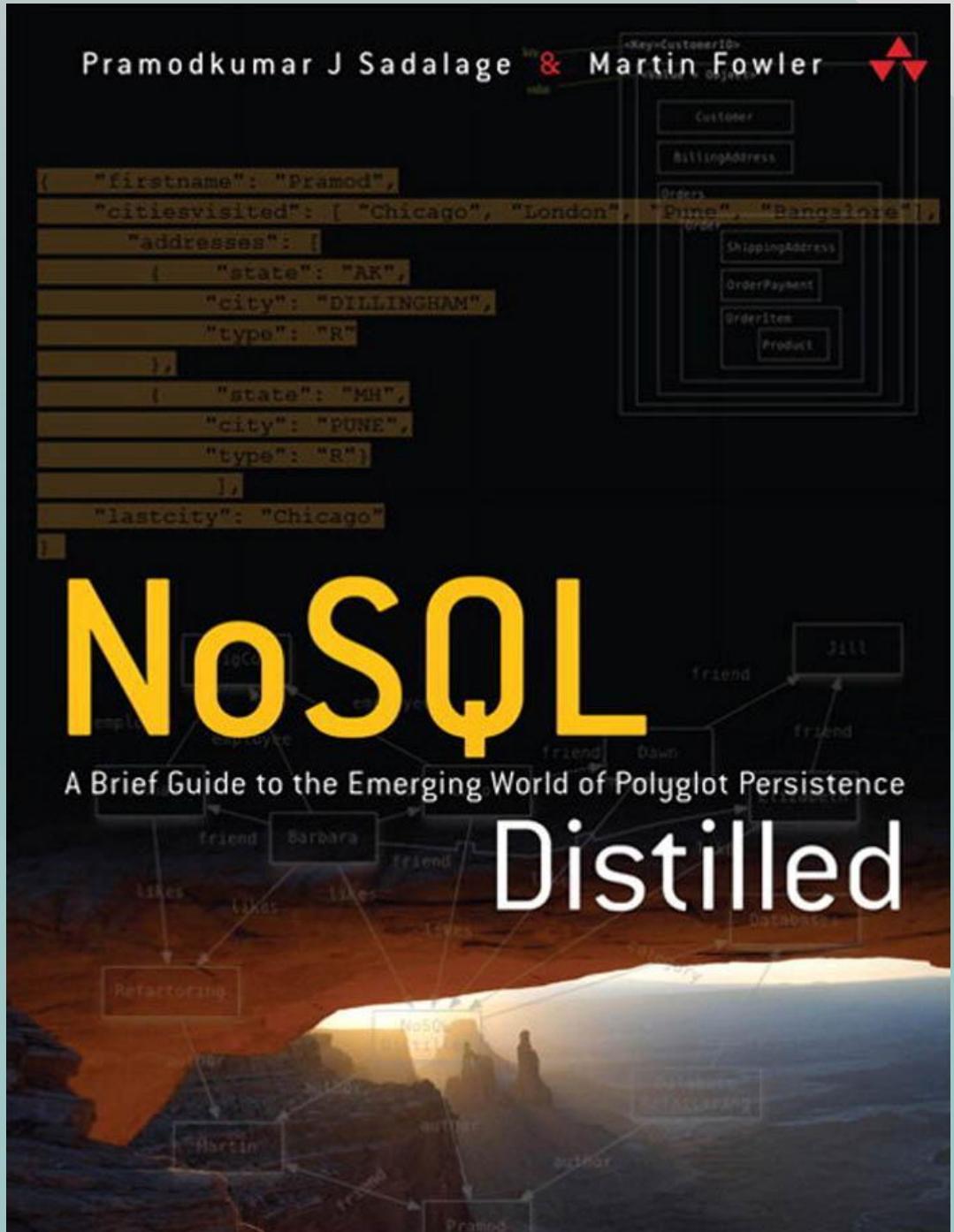
BD-documental

Riak CouchDB latencia

Consistencia

El (único) libro

- Agosto 2012



Introducción

Modelo y SGBD Relacional

- Exitoso y fiable
 - Persistencia
 - Control de concurrencia, Transacciones de sistema de alto nivel
 - Control de integridad y seguridad
 - Productividad (SQL consultas de alto nivel)
 - Integración de aplicaciones (BD común, SQL)
 - Agnosticismo en consultas
 - Conocido -> Estandarización -> Más profesionales
 - Optimizado
 - MADURO

Modelo y SGBD Relacional

- No siempre mejor solución
 - Adaptación a necesidades de aplicaciones, ej:
 - Esquema flexible
 - Muchos tipos de objeto, pocos elementos de cada tipo
 - Objetos complejos
 - Desadaptación de impedancias
- Evaluar alternativas
 - Sistemas con compromisos de diseño diferentes
- Persistencia Políglota
 - Usar el sistema de persistencia más adecuado para cada necesidad de persistencia del conjunto

Hacia otros modelos de datos

- Bases de datos de aplicación (no compartidas)
 - Integración mediante servicios
- *Big Data*
 - Procesamiento de cantidades ingentes de datos (sitios web masivos)
- Rendimiento para sitios web masivos
- Escalabilidad horizontal
 - *Clusters*. Adición de servidores
- Distribución

BB.DD. NoSQL

- Término “paraguas”
 - Sin definición exacta
 - *No SQL*
 - *Not only SQL*
 - En general, el “resto” de sistemas “no relacionales”
- Características generales asociadas al NoSQL
 - Modelo de datos no relacional, basado en agregados
 - Esquema flexible (sin esquema – *schemaless*)
 - Orientación a *clusters*
 - *Open Source*
 - Orientación a los problemas de los grandes servicios web

BB.DD. NoSQL

- No son la bala de plata
 - Menos maduras
 - Atomicidad elemental sin transacciones, o transacciones de bajo nivel
 - Funcionalidad típica de bases de datos limitada
 - No control seguridad
 - No control integridad
 - Consultas elementales
 - No lenguaje de consulta especializado (API básica)
 - Orientado a procesos -> no agnóstico en consultas
- Implementación del resto de funcionalidad típica en la aplicación

Modelos de datos NoSQL

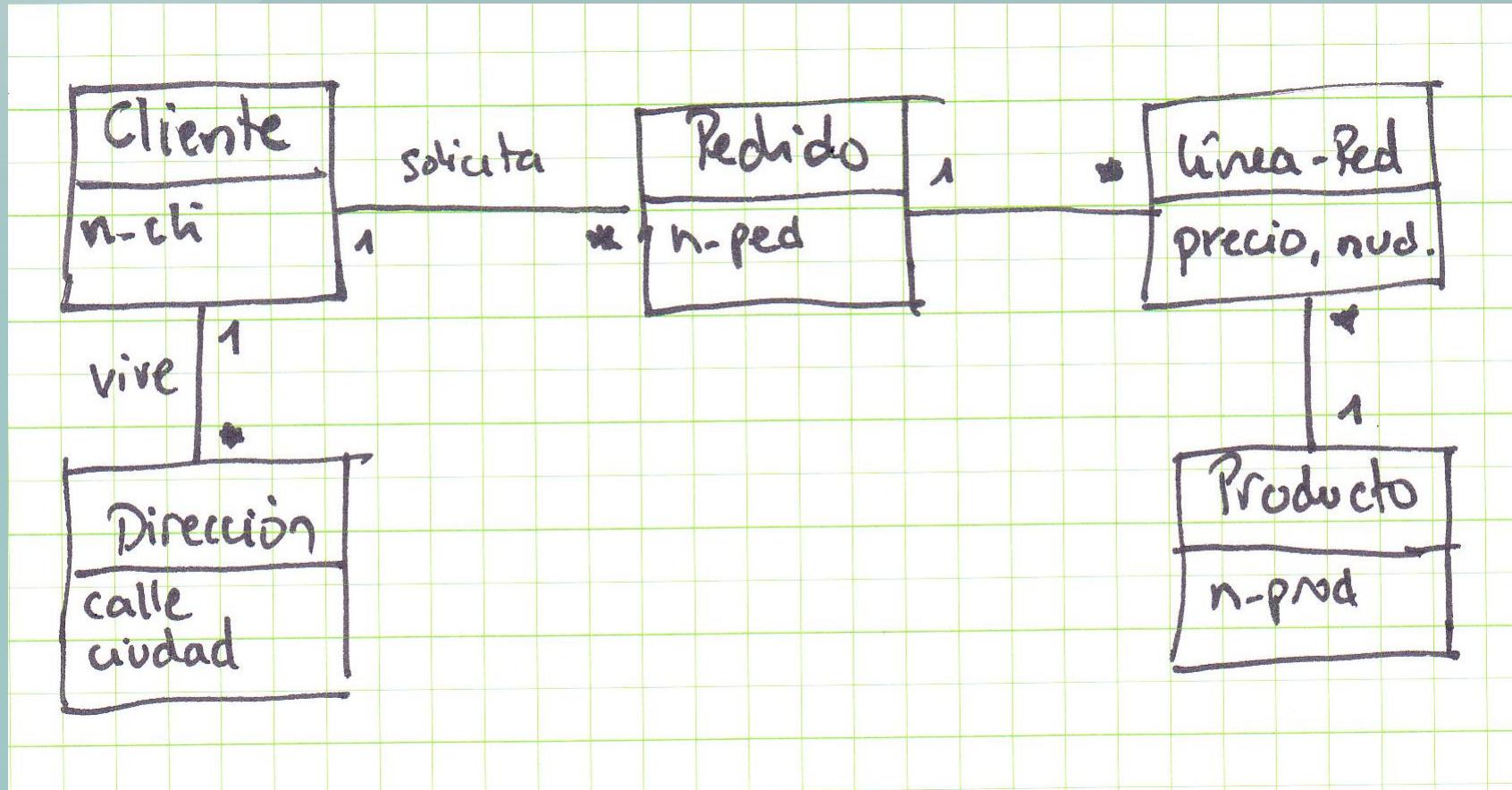
- Principales modelos de datos de sistemas NoSQL
 - Clave-Valor (*Key-Value Stores*)
 - Riak, BerkeleyDB, Redis...
 - Documental (*Document Databases*)
 - MongoDB, CouchDB, RavenDB...
 - Almacenamiento en columnas (*Column-Family Stores*)
 - Cassandra, HBase, Hypertable...
 - Grafo (*Graph Databases*)
 - Neo4J, OrientDB, Infinite Graph...

Modelos de datos orientados a agregados

Agregados

- Agregado
 - Conjunto de objetos relacionados tratado como unidad
 - Ej: un pedido junto con los productos incluidos
 - Unidad típica para
 - Persistencia (almacenamiento)
 - Organizado según orientación del proceso de la aplicación
 - » No agnóstico -> cambio en el esquema si evoluciona
 - Consistencia/atomicidad (transacciones)
 - Manipulación atómica sobre un agregado (Lectura/Escritura)
 - No transacciones ACID entre varios agregados
 - Distribución
 - Datos asociados (agregado) enteramente contenidos en el nodo
- Denormalización habitual

Modelo “relacional” sin agregar



Instancia relacional

<u>Cliente</u>	<u>id-cli</u>	<u>n-cli</u>
	1	Paco
	...	

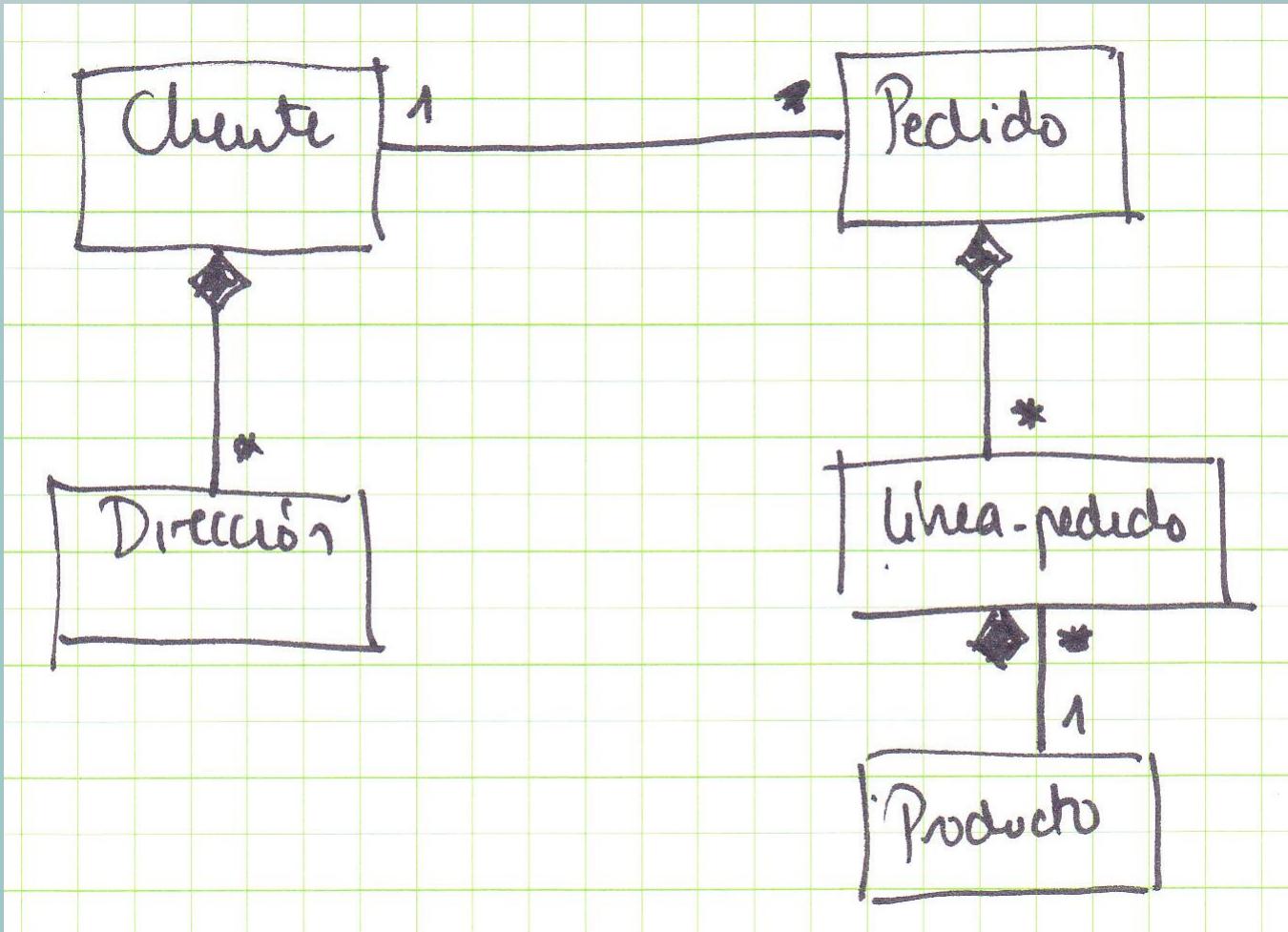
<u>Pedido</u>	<u>id-ped</u>	<u>n-ped</u>	<u>id-cli</u>
	66	Ped.66	1
	...		

<u>Producto</u>	<u>id-prod</u>	<u>nprod</u>
	77	Tuerca
	78	Dave
	...	

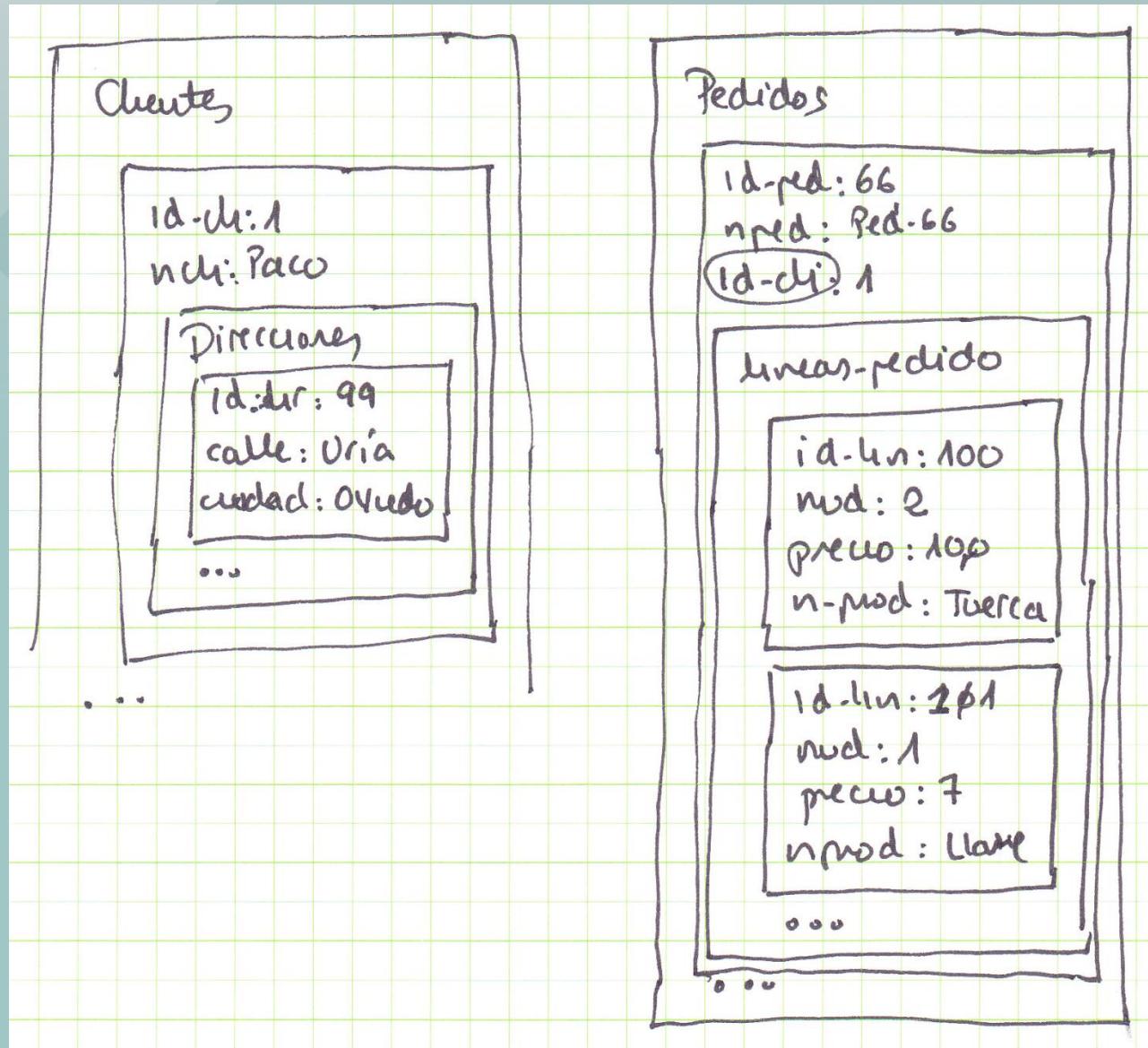
<u>Direccion</u>	<u>id-dir</u>	<u>calle</u>	<u>ciudad</u>	<u>id-cli</u>
	99	Uria	Oviedo	1

<u>linea-ped</u>	<u>id-lin</u>	<u>id-mod</u>	<u>n-ud</u>	<u>medio</u>	<u>idped</u>
	100	77	2	10	1
	101	78	1	7	1

Modelo “agregado”



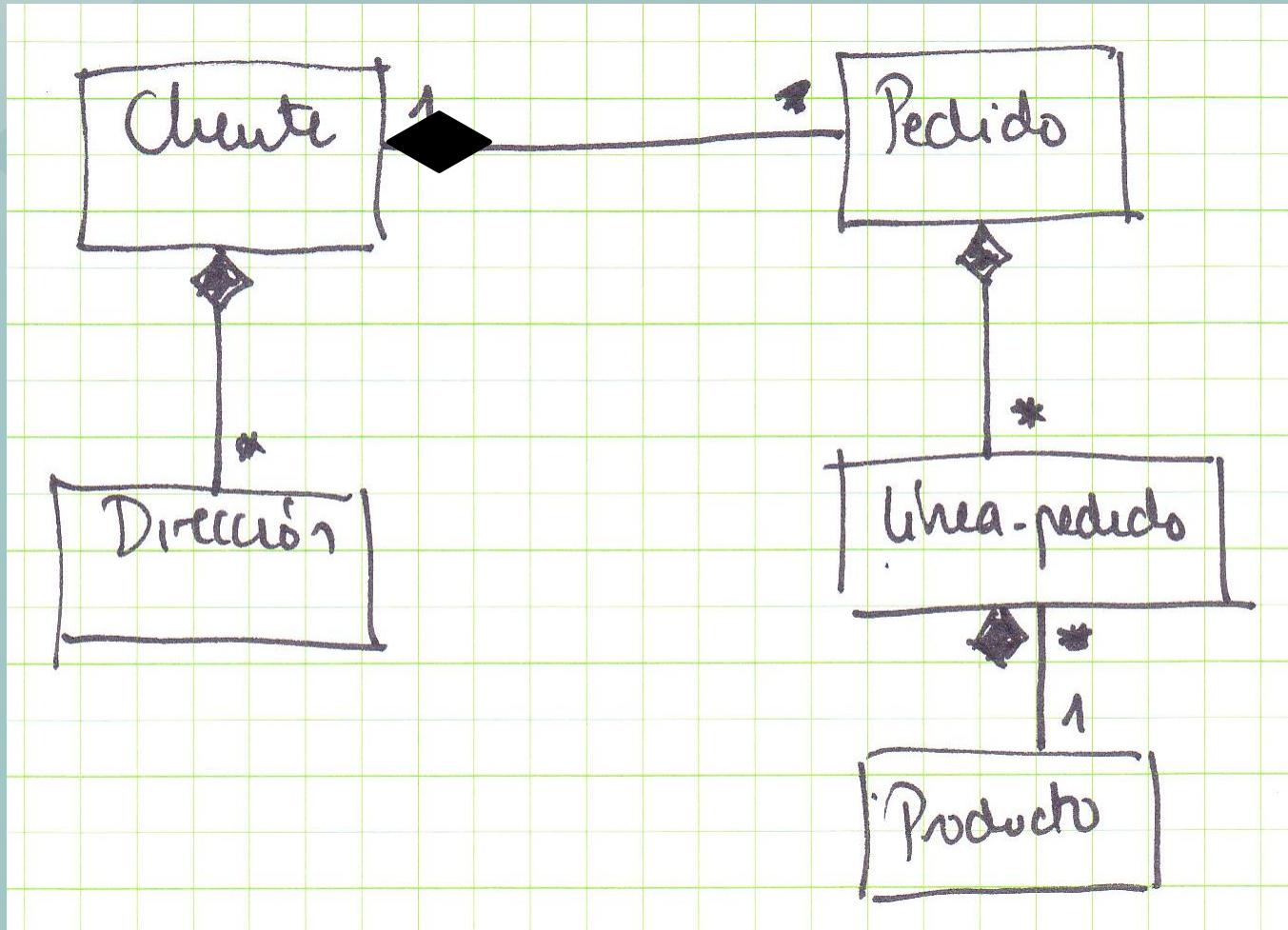
Instancia agregada



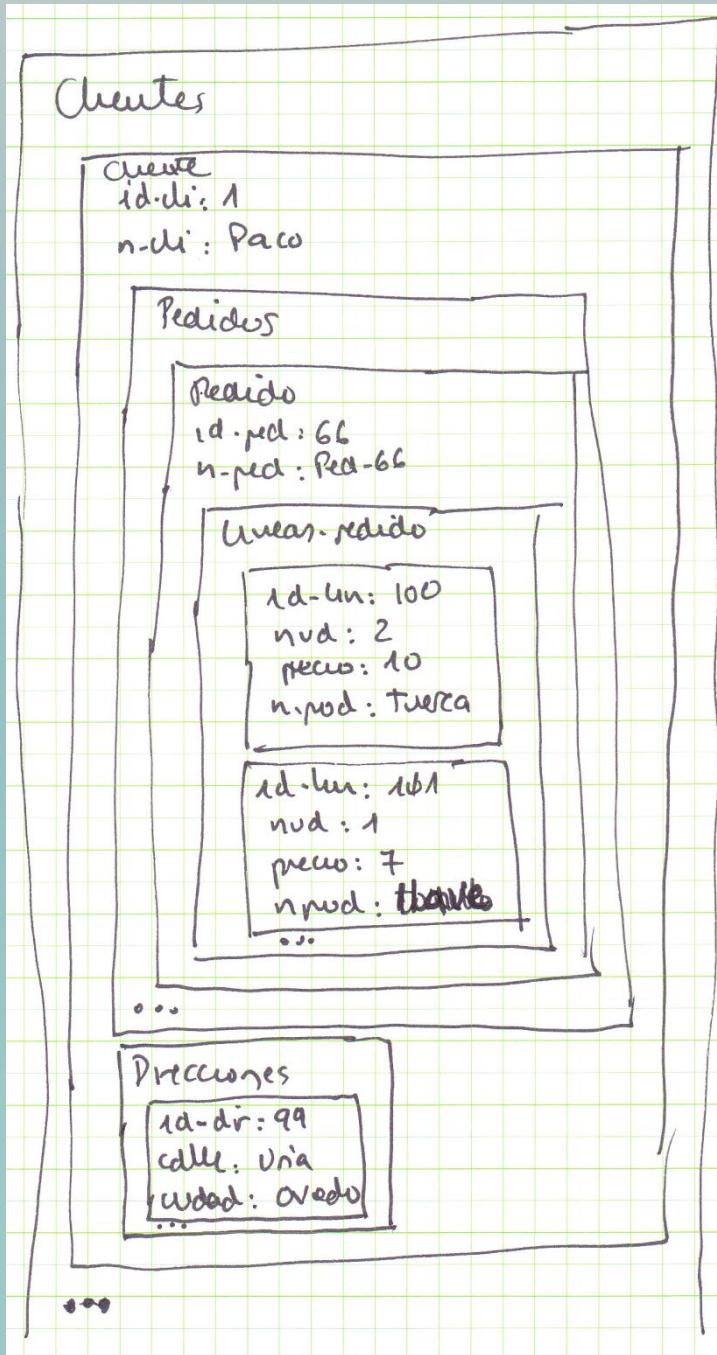
Instancia agregada JSON

```
// Clientes                                // Pedidos
{
  "id-cli":1,
  "ncli": "Paco",
  "Direcciones": [
    {
      "calle": "Uría",
      "ciudad": "Oviedo"
    }
  ],
}, ...
// Pedidos
{
  "id-ped": 66,
  "nped": "Ped-66",
  "id-cli":1,
  "línneas-Pedido": [
    {
      "id-lin": 100,
      "nud": 2,
      "precio": 10,
      "nprod": "Tuerca"
    },
    {
      "id-lin": 101,
      "nud": 1,
      "precio": 7,
      "nprod": "Llave"
    }
  ],
}, ...
...
```

Modelo agregado alternativo



Instancia agregada alternativa

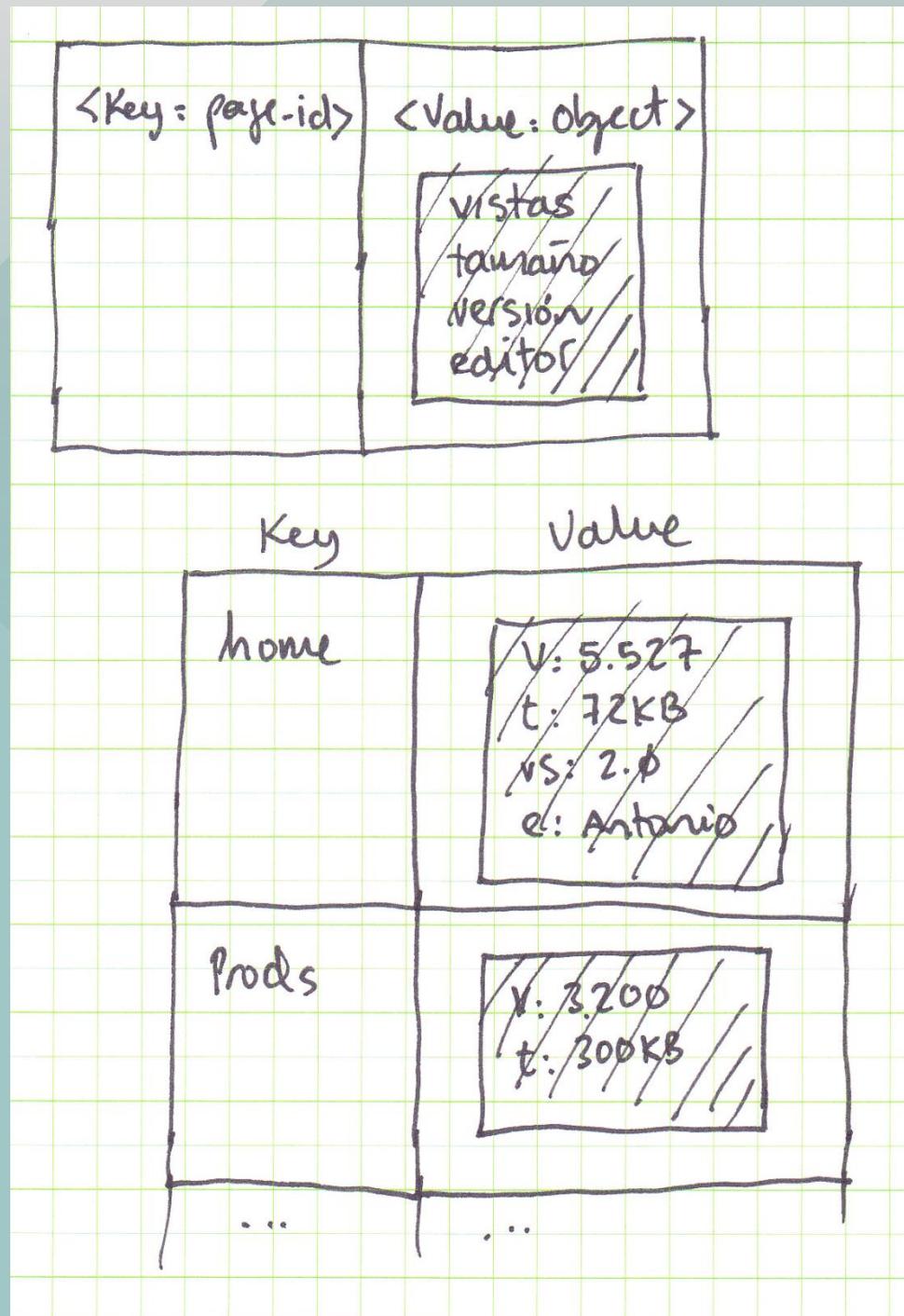


Instancia agregada 2 JSON

```
// Clientes
{
  "id-cli":1,
  "ncli": "Paco",
  "Direcciones": [ {"calle":"Uría", "ciudad", "Oviedo"} ]
  "Pedidos": [
    {
      "id-ped": 66,
      "nped": "Ped-66",
      "id-cli":1,
      "línneas-Pedido": [
        {
          "id-lin": 100, "nud": 2,
          "precio": 10, "nprod": "Tuerca"
        },
        {
          "id-lin": 101, "nud": 1,
          "precio": 7, "nprod": "Llave"
        }
      ]
    },
    ...
  ]
},
```

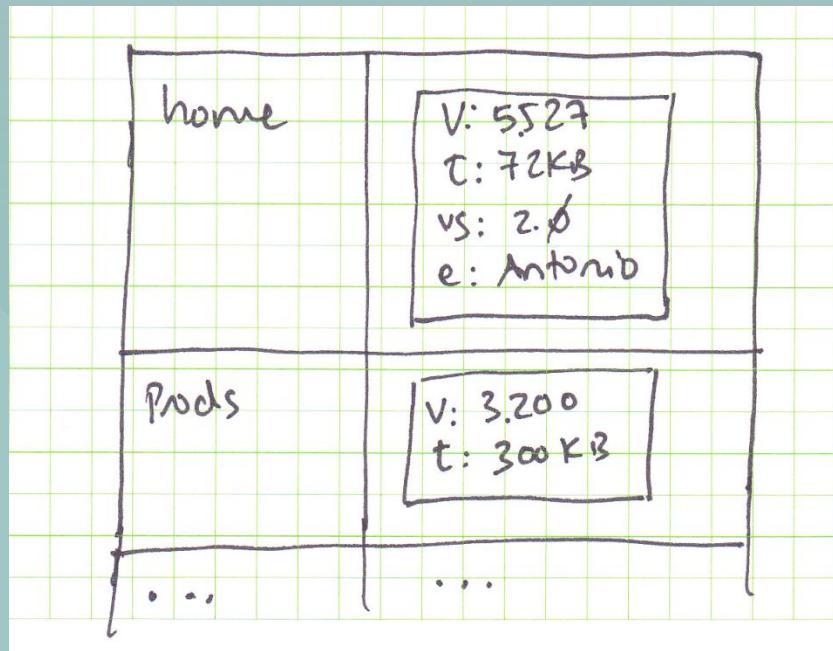
Almacenes Clave-Valor

- *Key-value stores*
- Asociación clave-valor
- Recuperación del valor buscando por clave
- Valor es un agregado opaco a la base de datos
 - Sin semántica, tipo void
 - La aplicación da semántica
- Unidad de consistencia/atomicidad



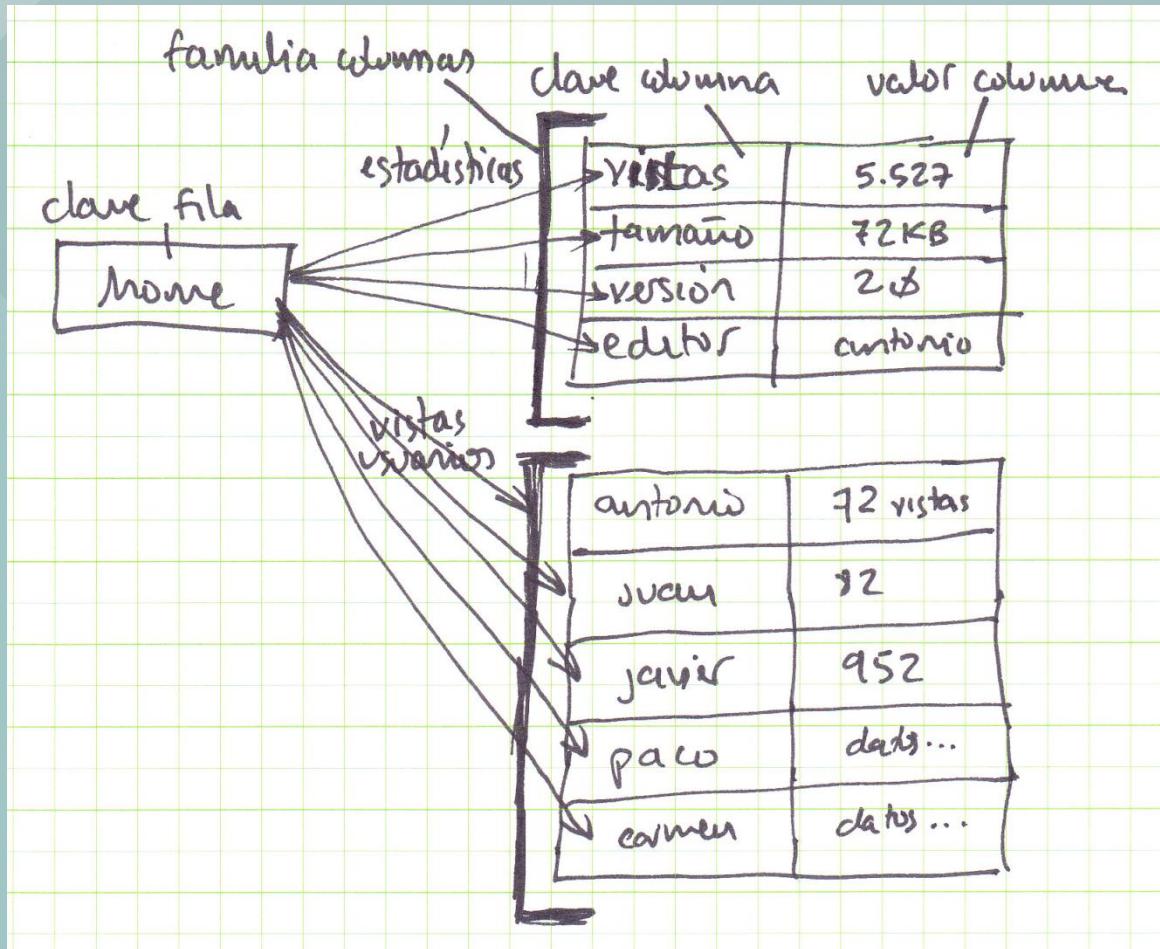
Bases de datos documentales

- *Document databases*
- Sistema clave-valor
- Valor no opaco, documento agregado con estructura
 - ej: valor JSON
- Consulta por clave
- *Consulta por contenido del documento*



Almacenes en columna

- *Column-family stores*
- Estructura de agregados clave-valor en dos niveles
 - Primer nivel
 - Clave de fila (*row identifier*)
 - Agregado asociado, mapa valores segundo nivel
 - Segundo nivel (columnas)
 - Clave de columna
 - Valor de la columna



Representación relaciones

- Intra-agregados, por estructura
 - Ej: Pedido con sus líneas de pedido en el mismo agregado
- Entre agregados, sin soporte directo
 - Ej: Acceder un producto y los pedidos en que sale que están en otro agregado
 - Introducción claves externas en los datos
 - Interpreta la aplicación
 - Soporte (limitado) dentro de la base de datos
 - Índices y consultas por contenido en documentales
 - Metadatos en los valores (metadato “id”) en clave-valor
- Transacciones inter-agregado sin soporte directo

Esquema flexible (sin esquema)

- *Schemaless databases*
- Admiten cambios en el esquema
 - Clave-valor
 - sin estructura en el valor
 - Admiten cualquier dato
 - Documentales
 - Estructura documento flexible
 - Objetos complejos sin restricción (estilo XML)
- Aplicaciones sin datos uniformes
- Esquema implícito en la aplicación
 - Código tiene que saber tratar lo almacenado

Vistas materializadas

- Vistas (consultas) precalculadas y almacenadas
 - Rendimiento
- Aplicación en NoSQL
 - Presentación datos que no encajen con la estructura agregada elegida
 - Manual (ejecutar consulta, guardar en BD) / sistema
- Actualización vistas materializadas
 - Ansiosa (*eager*)
 - Perezosa (*lazy*)
 - Por lotes (*batch*)

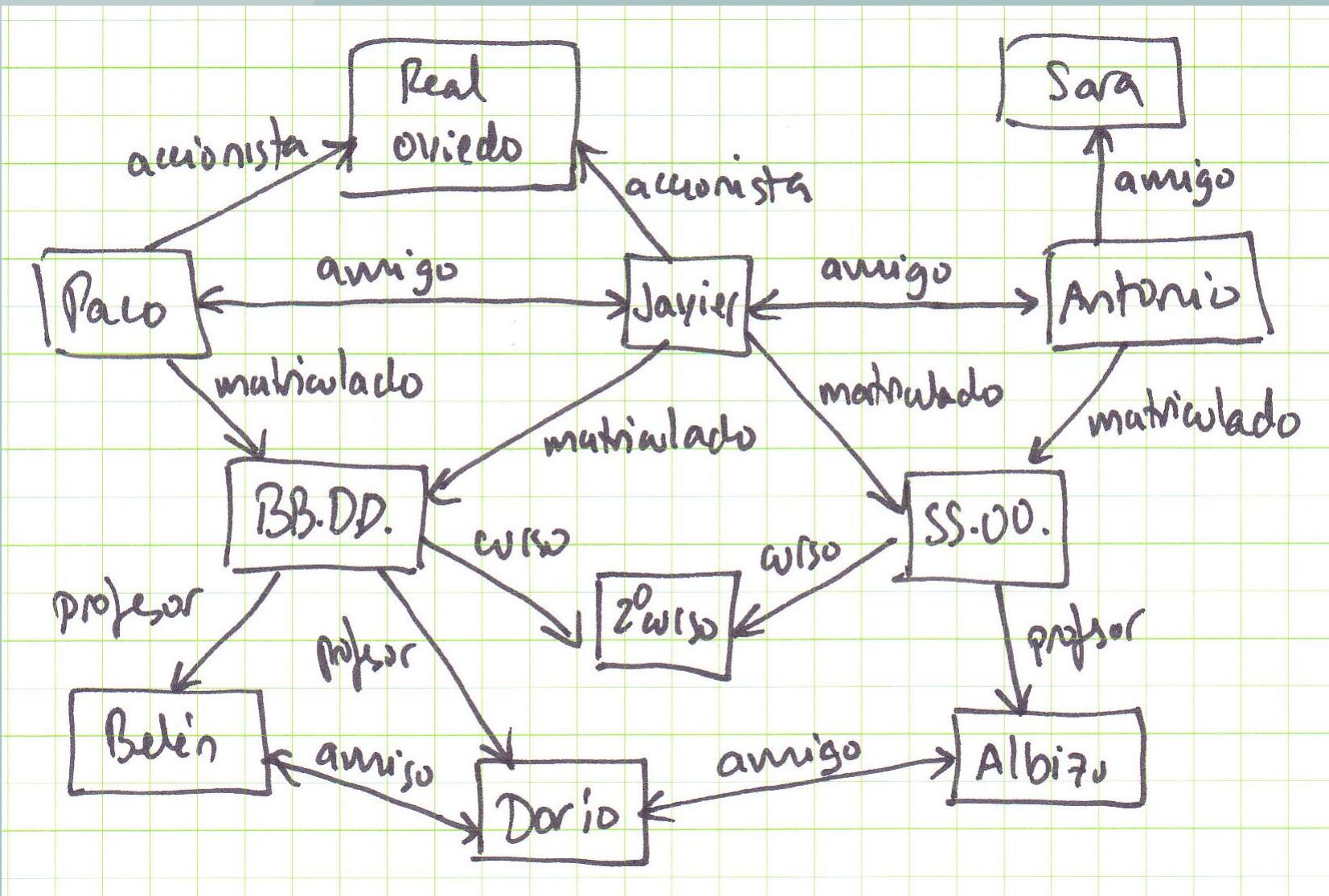
Orientación del modelado

- Modelado en agregados orientado al proceso
 - “direccionalidad” de los accesos principales
- Decidir agrupación en agregados
 - Ej: cliente junto con sus pedidos
 - Ej: cliente separado de pedidos (uso referencias)
 - Decisiones de modelado
- Denormalización
 - Análisis en tiempo real
 - Ej: añadir en el producto los pedidos que lo incluyen
 - Rendimiento en general
 - Ej: añadir en la línea de pedido datos del producto

Modelos de datos en grafo

Bases de datos de grafo

- Incluidas en el grupo NoSQL
 - Aunque distinta orientación
- Motivación: muchos registros pequeños con relaciones complejas entre ellos
- Estructura de datos de grafo
 - Nodos con propiedades
 - Relaciones entre nodos (arcos)
 - Unidireccionales / Bidireccionales
 - Propiedades en relaciones



Bases de datos de grafo

- Diferentes al resto de NoSQL
 - Énfasis en relaciones
 - Optimización recorrido grafo (vs. productos naturales)
 - Poco orientadas a distribución
 - Transacciones ACID
 - Involucrando varios nodos y relaciones
 - Consultas complejas
 - Ej: Quiénes son los amigos de los accionistas del Real Oviedo.
 - Normalmente se necesita un nodo de partida

Mecanismos de distribución

Distribución de los datos

- Escalabilidad de datos
 - Vertical -> aumentar capacidad servidor
 - Horizontal
 - añadir más servidores -> distribución
 - Bases de datos en clúster
- Distribución de datos
 - Repartirlos entre los distintos servidores
 - Mayor rendimiento, disponibilidad
 - Mayor complejidad
- Alternativas distribución
 - Servidor único, fragmentación, replicación

Servidor único

- No distribuir los datos
- Opción más sencilla
- Mejor opción, SI SE PUEDE
 - Según necesidades aplicación
 - También para NoSQL por modelo datos más adecuado para aplicación

Fragmentación (*sharding*)

- Fragmentar los datos y repartirlos entre servidores
- Cada servidor es responsable de un fragmento
 - Lectura / Escritura
 - Adecuado para agregados (datos necesarios en el mismo servidor)
- Equilibrio de carga
- Tradicionalmente realizado “a mano” en la aplicación
 - Ej: Tablas “clienteEspaña”, “clienteExtranjero”
- No mejora la robustez (*resilience*)

Replicación Maestro-Esclavo

- Replicación (replication)
 - Replicar los datos entre los nodos
- Maestro: fuente de referencia (*authoritative*)
 - Gestiona las actualizaciones
 - Retransmite las actualizaciones a los esclavos
 - Definidos manualmente o automáticamente
- Esclavos: repiten los datos
- Escalabilidad en lecturas
- Robustez en lecturas
- ¿Inconsistencias (actualizaciones pendientes)?

Replicación entre iguales

- Replicación entre iguales (*peer to peer*)
- Sin maestro. Todas las réplicas equivalentes
 - Todas pueden aceptar actualizaciones (escrituras)
 - Coordinación para sincronizar actualizaciones
- Robustez mayor
 - No hay un maestro único
- ¿Inconsistencias?
 - Escrituras conflictivas al mismo dato en distintos nodos (inconsistencia en escritura).
 - Esperar a sincronización para evitar conflicto
 - Aceptar inconsistencias de escritura -> aplicación

Fragmentación + replicación

- Combinación de ambas técnicas
- Factor de replicación
 - Número de nodos en los que está repetido cada fragmento
- Ej: Fragmentación con replicación maestro-esclavo
 - 7 fragmentos, cada uno con factor de replicación 3 -> 21 nodos servidores
 - Cada fragmento tiene un maestro -> 7 nodos maestros
- Almacenamiento en columnas
 - Típicamente fragmentación + replicación entre iguales

Mecanismos de gestión de consistencia

Consistencia

- Consistencia fuerte
 - Datos siempre correctos
 - Típico en sistemas relacionales
 - Más sencillo y seguro
 - Problemas en cluster (distribución) y rendimiento
- Consistencia eventual
 - Datos consistentes pasado un tiempo
 - Ej: después de actualizar un saldo, se acaba propagando a todas las réplicas
- Ventana de inconsistencia

Consistencia en actualizaciones

- Conflicto escritura-escritura (*write-write*)
 - Ej: 2 profesores actualizando la nota de un alumno con distintos valores
 - Sin control de concurrencia. Gana uno: actualización perdida
 - Escritura inconsistente
 - Resultado correcto: ejecución en serie
- Agravado con la replicación

Control de consistencia

- Técnicas pesimistas
 - Evitan conflictos (ej: bloqueos/*locks*)
- Técnicas Optimistas
 - Detección conflicto
 - Ej: actualización condicional
 - Ej: versionado
 - Recuperación del conflicto
 - Ej: versionado, reconciliar versiones
 - Puede ser manual y dependiente del dominio
- Compromiso seguridad (datos correctos) vs. viveza (tiempo respuesta)
 - *Safety vs. liveness*

Consistencia en lecturas

- Conflicto Lectura/Escritura (*Read-Write*)
 - Ej: Nota de un alumno calculada por un profesor en base a los trabajos mientras otro sigue añadiendo trabajos. El alumno lee su nota entre medias.
 - Lectura inconsistente
- Consistencia lógica
 - Transacciones SGBDR
- Ventana de inconsistencia
 - BB.DD. agregados, atomicidad dentro de un agregado
 - Posibilidad de lecturas inconsistentes en actualizaciones con varios agregados distintos
 - Puede ser de pequeña duración (o no: latencia red)

Consistencia eventual

- Consistencia de replicación
 - Problema consistencia agravado con réplicas
 - Debería leerse el mismo valor en todas las réplicas
 - Ej: reserva de una habitación de hotel con el dato replicado en dos servidores distintos A y B
 - En A se reserva la habitación, pero hay una ventana de inconsistencia: en B se ve la habitación como libre hasta que se propague la reserva
- Consistencia eventual
 - Se permite inconsistencia en la replicación, pero
 - Eventualmente todos los nodos estarán actualizados
- Dato desactualizado (*stale data*)

Gestión de la inconsistencia

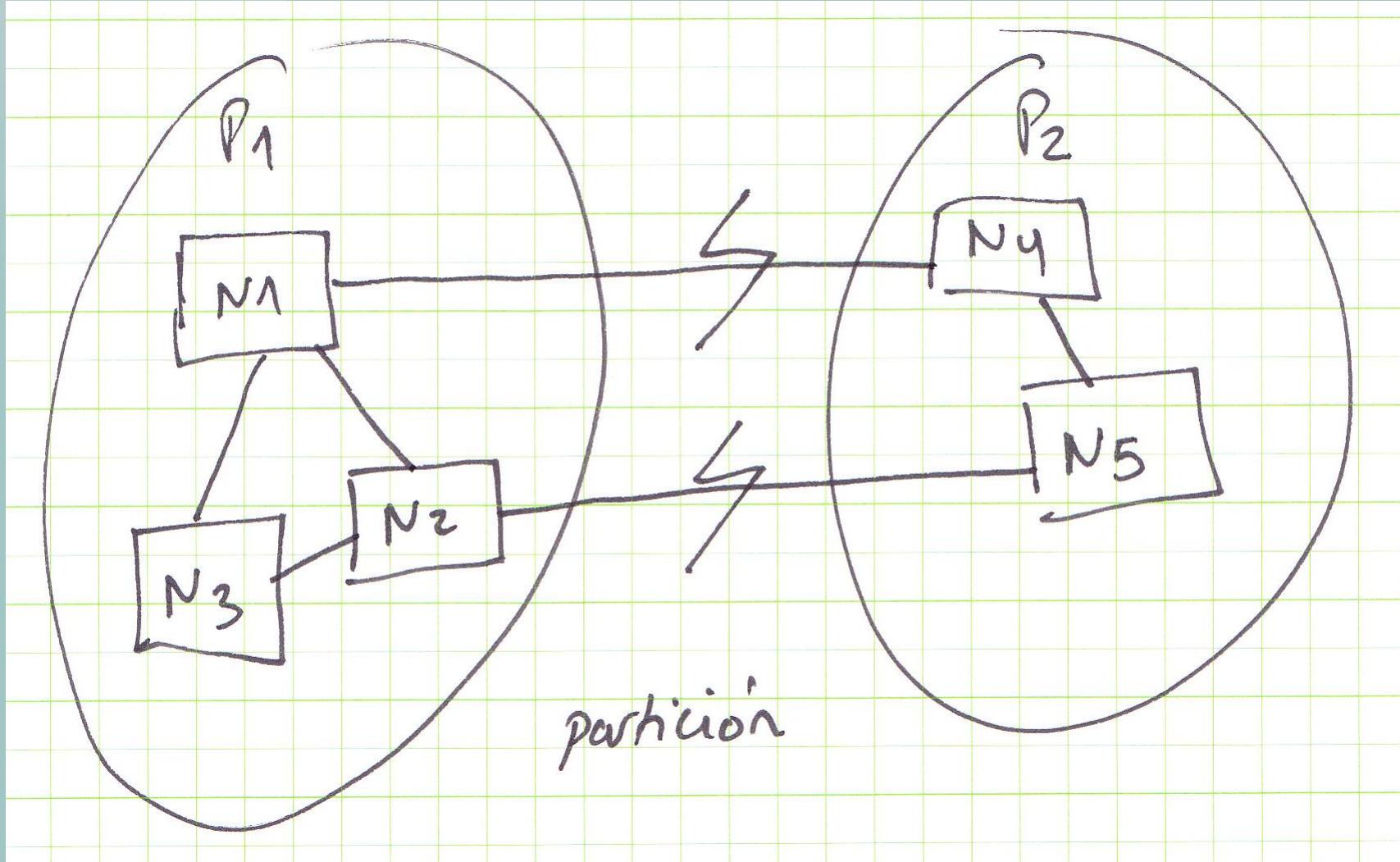
- Depende de la aplicación / dominio
 - Ej: Un tweet que no aparece en todos los nodos hasta unos minutos después de grabarse
- Leer tus escrituras (*read-your-writes*)
 - Garantiza que uno puede leer los datos que ha escrito.
 - Escritura fantasma (*phantom write*): lo contrario
- Consistencia de sesión
 - *Read-your-writes* garantizada en la sesión de usuario
 - Ej: asignar sesión a un único servidor (*sticky session*, sesión pegajosa). ¿y las escrituras al maestro?
- Marcas de versión (*version stamps*)

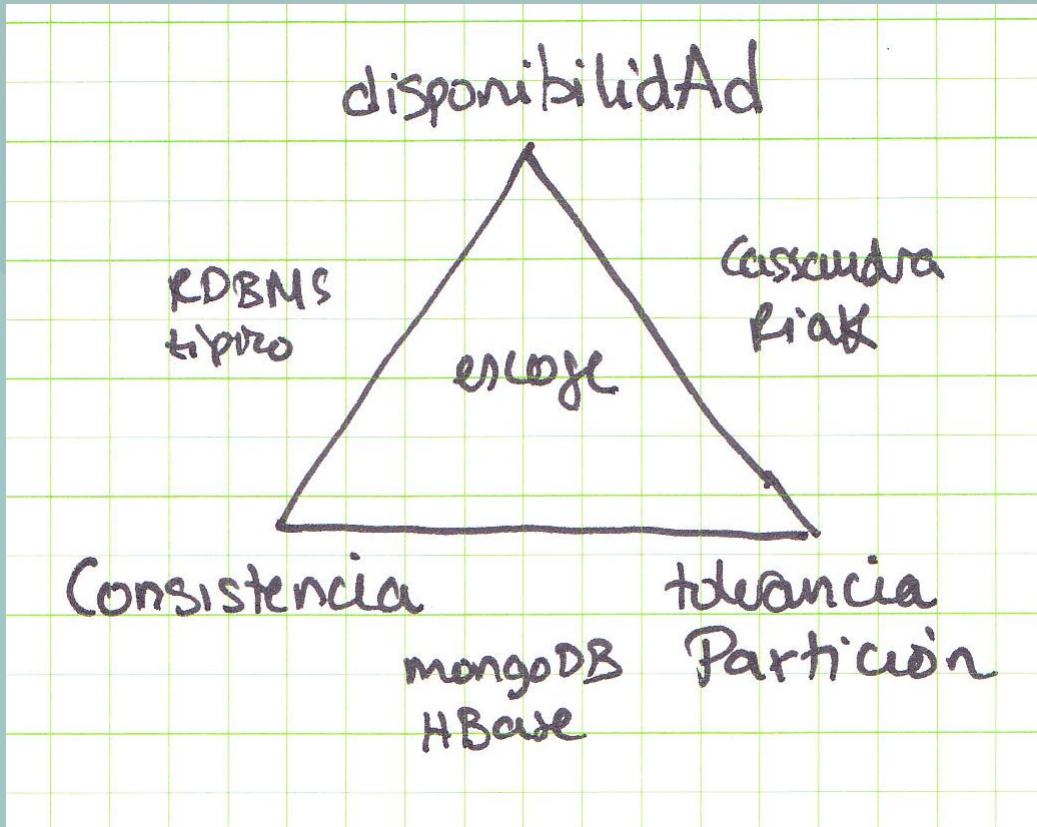
Relajación consistencia

- Impacto consistencia “fuerte”
- Intercambio consistencia por otros factores
 - Ej: rendimiento
 - Ej: relajación aislamiento transacciones relacionales (niveles de aislamiento)
 - Ej: transacciones “largas”, imposibilidad de incluir elementos remotos en la transacción

Teorema CAP

- Eric Brewer 2000
- Demostrado (ciertas condiciones)
- Consistencia
 - Respuestas “correctas” a los clientes
- disponibilidAd
 - Si hay acceso, el nodo contesta a lecturas/escrituras
- tolerancia Partición
 - Partición de cluster dividida por fallo de red, sigue respondiendo a los clientes
- Solo pueden tenerse dos a la vez





Alternativas arquitectónicas CAP

- En realidad, es un espacio de diseño
- Servidor único es CA (no puede particionarse)
- Compromisos CAP en cluster
 - En partición, escoger entre C y A
 - Diferentes niveles de intercambio C/A
- Escoger C en lugar de A
 - Ej. Sistema reservas con 2 nodos replicados
 - Consistencia completa: Reserva en el nodo N1 debe comunicarse al nodo N2 antes de confirmarla
 - (serialización de las escrituras)
 - Sin disponibilidad si se corta la red

Alternativas arquitectónicas CAP

- En realidad, es un espacio de diseño
- Servidor único es CA (no puede particionarse)
- Compromisos CAP en cluster
 - En partición, escoger entre C y A
 - Diferentes niveles de intercambio C/A
- Escoger C en lugar de A
 - Ej. Sistema reservas con 2 nodos replicados
 - Consistencia completa: Reserva en el nodo N1 debe comunicarse al nodo N2 antes de confirmarla
 - (serialización de las escrituras)
 - Sin disponibilidad si se corta la red

Alternativas arquitectónicas CAP

- Mejorar disponibilidad
 - Nodo maestro para un hotel determinado ej. N1
 - Escrituras al maestro
 - Permite hacer reservas a los clientes conectados a N1 aunque haya partición
 - Pero los clientes de N2 no pueden hacer reservas
 - Fallo disponibilidad (N2 conecta, no se puede reservar)
- Mejora adicional
 - Permitir hacer reservas en particiones
 - Gestión escrituras inconsistentes
 - Ej: reserva misma habitación en N1 y N2 (*overbooking*)
 - Ej: resolución “manual” *overbooking*

ACID vs. BASE

- ACID tradicional
 - Atomicidad
 - Consistencia
 - aislamiento
 - Durabilidad
- BASE NoSQL
 - Basically Available
 - Soft state
 - Eventual Consistency
- No disyuntiva, escoger en un espectro

de CAP a PACELC

- Disyuntiva Consistencia / Latencia
 - Más que partición y disponibilidad, latencia
 - Mejora consistencia comunicando nodos entre sí
 - Aumenta la latencia
 - disponibilidAd: latencia máxima que toleramos hasta declarar el dato no disponible
 - Daniel Abadi 2012
- PACELC: formulación mejor del CAP
 - Si hay una (P)artición, cuál es el compromiso del sistema entre disponibilid(A)d y (C)onsistencia.
 - (E)n otro caso, cuando el sistema funciona normalmente sin particiones, cuál es el compromiso del sistema entre (L)atencia y (Consistencia)

Relajación de la durabilidad

- Durabilidad: almacenamiento modificaciones
- Relajación: pérdida de modificaciones
 - En ocasiones puede intercambiarse la durabilidad por el rendimiento
 - Ej: bases de datos en memoria que se vuelcan a disco persistente periódicamente
 - Pérdida en caso de caída del servidor
 - Ej: almacenamiento de la sesión de usuario en memoria
 - Ej: Replicación. Escritura confirmada en maestro, que cae antes de propagarla
 - Mejora: esperar a que se propague a algunos esclavos antes de confirmar la escritura => mayor latencia

Quorums

- Replicación. Nivel de consistencia de una escritura aumenta con nodos implicados
- Quorum de escritura (*write quorum*) $W > N / 2$
 - Inconsistencia de escritura. Gana la escritura que alcanza la mayoría de nodos
 - W. Nodos que participan en la escritura
 - N. Factor de replicación (número de copias del dato)

Quorums

- Quorum de lectura (read quorum) R
 - Número de nodos a contactar para estar seguro de leer el dato actualizado. Depende del W.
 - Ej: $N = 3, W = 2 \Rightarrow R = 2$ asegura dato correcto
 - Ej: $N = 3, W = 1 \Rightarrow R = 3$ asegurar dato correcto
 - No hay quorum de escritura, así que hay que preguntar a todos porque puede detectarse conflicto de escritura.
- Lectura consistente fuerte $R + W > N$
 - Puede alcanzarse sin escritura consistente
- R, W, N configurables (replicación P2P)
 - Escoger en función de las necesidades
 - No todas las L/E necesitan el mismo tratamiento

Marcas de versión

- Técnica gestión concurrencia
- Añadir marca de versión a cada dato
 - Actualizada en cada actualización
- Ayuda detección conflictos concurrencia
 - Lectura dato
 - Actualización posterior dato
 - Comprobar marca versión no cambió => conflicto escritura
- Implementación
 - Contadores, GUIDs, hashes, marcas de tiempo, combinaciones

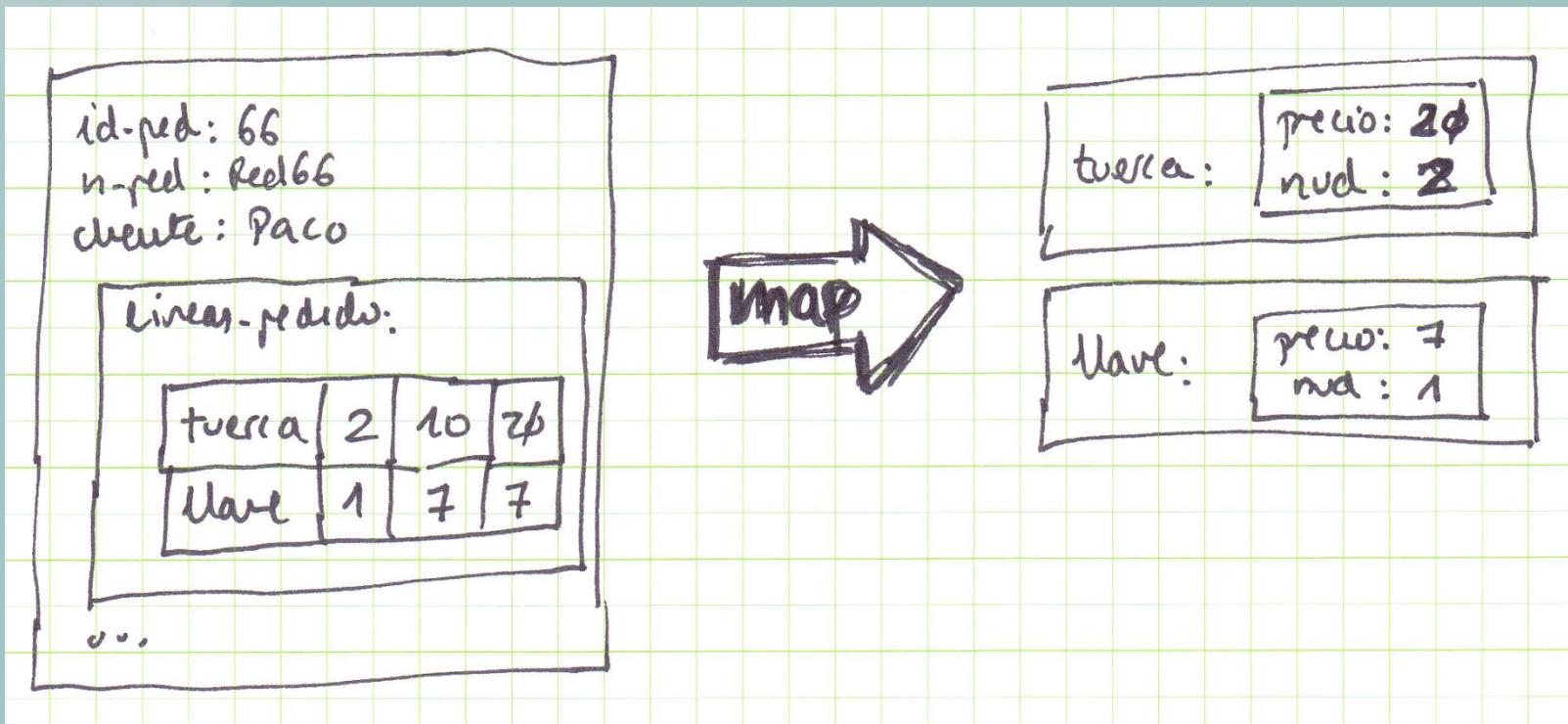
Map / Reduce

Map / Reduce

- Mapear / Reducir
- Patrón de parallelización de computación en cluster
- Tareas *Map*
 - Lee datos de agregado y genera listas clave-valor
 - Paralelizable. Ejecutado en el nodo del agregado
- Tareas *Reduce*
 - Toman valores pertenecientes a la misma clave y los reducen a un valor único.
 - Paralelizable por valor de clave
- Combinables en cadenas

Ejemplo Map

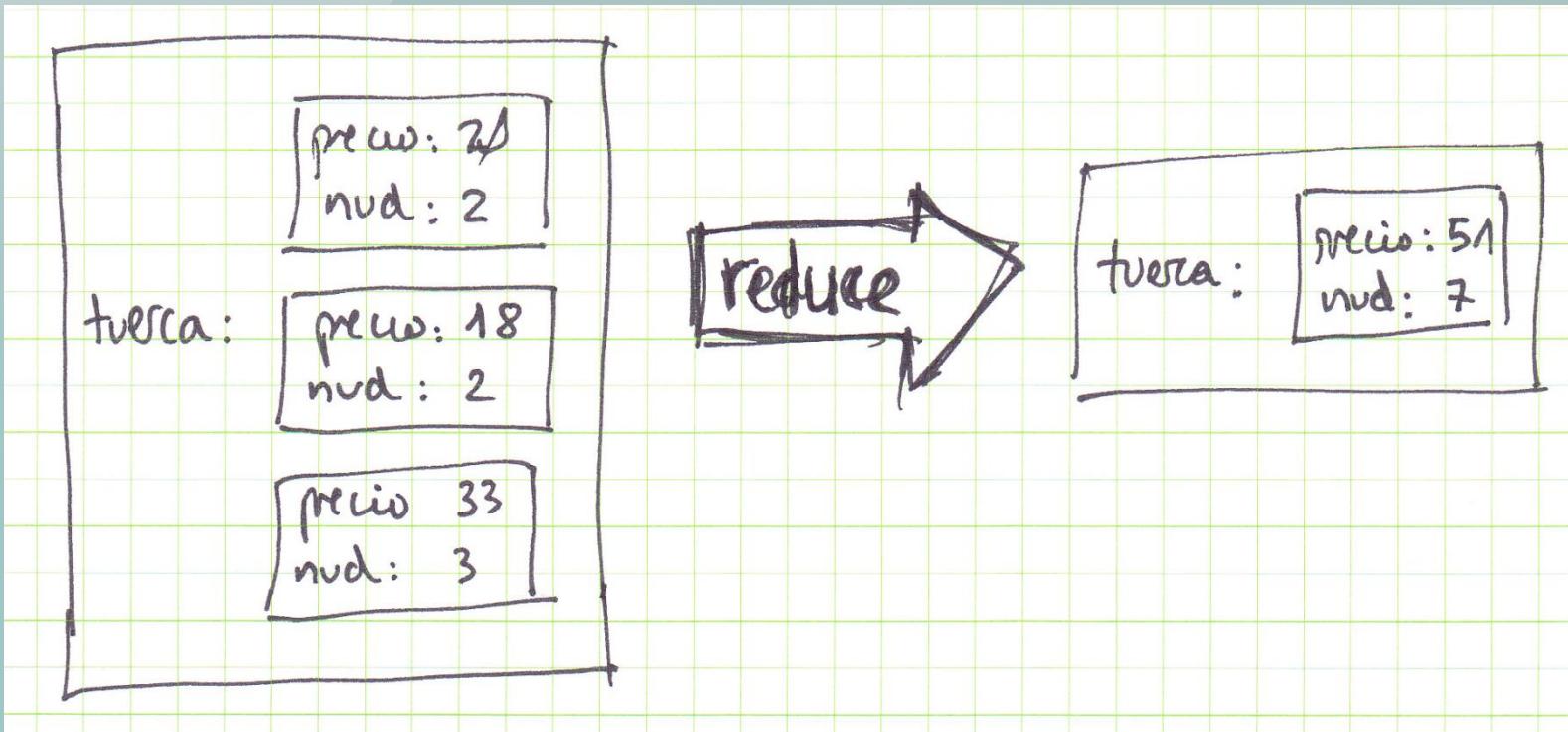
Objetivo: calcular el gasto y el número de unidades total acumulado para un producto en todos los pedidos



Tarea Map. Dado un pedido genera

Clave: producto / Valor: precio y unidades del pedido

Ejemplo Reduce



Tarea Reduce. Agrupados los valores de una clave los reduce

Clave: producto / Valor: precio y unidades acumuladas

Patrón

- Lectura datos (agregados)
- Map: extraer información de interés de los datos
 - Paralelizable
- Repartir y ordenar
 - Para paralelizar
- Reduce: generar los datos
 - Agregar, transformar, resumir, filtrar
 - Paralelizable
- Recolectar los resultados

Ejemplo clásico. WordCount

- Contar el número de palabras de un conjunto de textos
- Obtener los textos
- Map de cada texto
 - Generar lista <palabra>, 1 para cada palabra de un texto
- Agrupar por cada palabra
- Reduce de cada palabra
 - Acumular el número de elementos del grupo

Pseudocódigo WordCount

```
Map(in: texto) {  
    for each palabra in texto  
        emitResult(palabra, "1");  
}  
  
Reduce(in: palabraClave, Iterator contadorPalabra) {  
    int frecuencia := 0;  
    for each contador in contadorPalabra  
        frecuencia := frecuencia + contadorPalabra;  
    return frecuencia;  
}
```