Tecnología de Gestión de Datos

Tema 3: Tecnología NoSQL

MU Ingeniería y Tecnología de Sistemas Software
Profesor: Juan Carlos Casamayor

Objetivos

- Conocer las razones del auge de la tecnología NoSQL
- Conocer los modelos de datos en los que se basa esta tecnología
- Conocer los modelos de distribución de datos
- Conocer la gestión de la consistencia y del marcado de versiones
- Conocer algunos sistemas NoSQL

Índice

1. Introducción a la tecnología NoSQL
2. Modelos de datos
   2.1. Modelos de datos agregados
   2.2. Otros modelos y aspectos de modelado
3. Modelos de distribución de datos
4. Consistencia
5. Marcados de versión

1. Introducción a la tecnología NoSQL

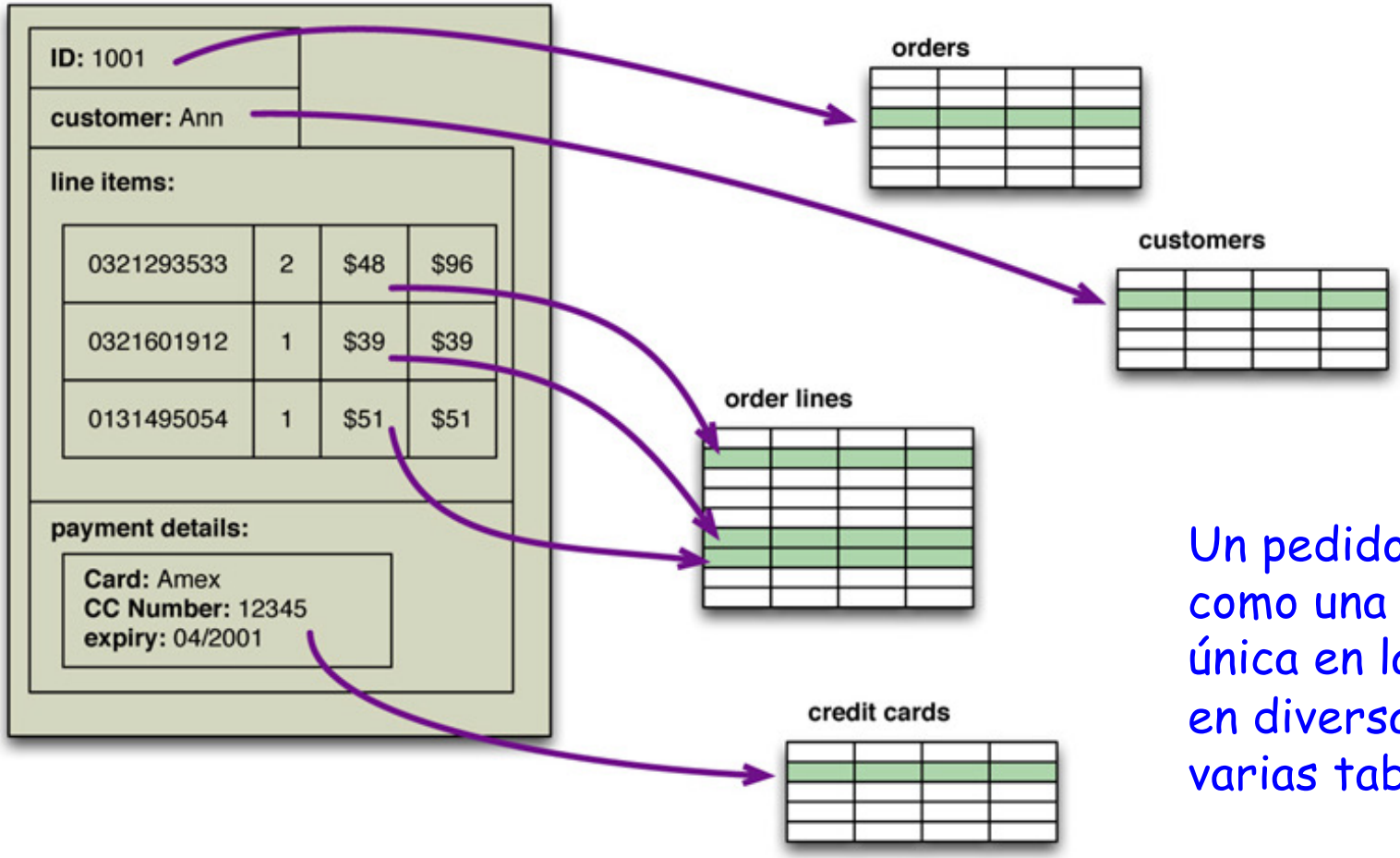
- Bases de datos relacionales:
  - Tecnología predominante desde 1980 – principios de 2000
  - Razones:
    - Tecnología adecuada para dar **persistencia** a los datos: flexible, segura, buen rendimiento
    - Gestión de la **concurrencia**: asegura la corrección de la actividad concurrente sobre pequeñas porciones de datos. Uso del concepto de transacción
    - **Integración**: Organizaciones usan distintos tipos de aplicaciones. Los datos se integran en una base de datos y se gestionan los accesos concurrentes de las distintas aplicaciones.
    - **Modelo estándar**: las bases de datos relaciones son el núcleo común y conocido, permitiendo la comunicación entre equipos y la compartición de conocimiento.

1. Introducción a la tecnología NoSQL

- Discordancia de la impedancia:
  - Diferencia entre la representaciones de los datos en el modelo relacional y en la estructuras de memoria de los programas de aplicación.
  - **Modelo relacional**:
    - Relaciones y tuplas
    - Todas los operadores del SQL actúan sobre relaciones y devuelven relaciones
    - Elegante y simple, pero introduce limitaciones
  - **Modelos de los datos en memoria en los programas de aplicación**:
    - Más ricos
    - Permiten estructura complejas con datos agregados.

1. Introducción a la tecnología NoSQL

- Discordancia de la impedancia:



Un pedido, que se ve como una estructura única en la IU, se divide en diversas filas de varias tablas en la BDR

1. Introducción a la tecnología NoSQL

- Discordancia de la impedancia:
  - En los 90 se pensó que este problema provocaría que las BDR fueran sustituidas por otras donde se replicará en disco las estructuras de datos en memoria → **Bases de Datos Orientadas a Objetos**
  - BDR se apoyaron en su papel de integración y en su capacidad de estandarización.
  - **Frameworks de mapeo objeto-relacional (Hibernate):** resolvieron en parte el problema

1. Introducción a la tecnología NoSQL

- Bases de datos de integración y de aplicación:
  - **Bases de datos de integración:**
    - BDR son el principal mecanismo de integración entre aplicaciones.
    - Aumenta la complejidad de la estructura almacenada.
    - Aumenta las disparidades de criterios de manejo de los datos por parte de las diversas aplicaciones y de necesidades de rendimientos
  - **Base de datos de aplicación:**
    - Almacena datos de una única aplicación
    - Sólo un equipo de desarrollo necesita conocer la estructura de la BD
    - Sólo un equipo controla la BD y el código de la aplicación → integridad controlada en BD o aplicación

1. Introducción a la tecnología NoSQL

- Bases de datos de integración y de aplicación:
  - **Web y Bases de datos:**
    - Entrada de los servicios web: comunicación HTTP.
    - Integración vía servicios web: mayor flexibilidad en los datos intercambiados ←→ uso SQL implica estructura relacional.
    - Servicios web capaces de usar estructuras más ricas: listas y registros anidados (uso de XML o JSON)
    - Reducción en el número de accesos en las interacciones
    - Servicios web (comunicación con el exterior) y BD (almacenamiento de datos):

      libertad de elección de la BD → Opciones no relacionales

1. Introducción a la tecnología NoSQL

- Necesidad de la clusterización:
  - **Sitios Web:**
    - Seguimiento de la actividad y la estructura con detalle.
    - Enlaces, redes sociales, diarios de actividad, datos de ubicuidad…
    - Crecimiento de los usuarios… necesidades de dar servicio a un número inmenso de usuarios
    - Aumento enorme de necesidades de computación… dos soluciones: a lo alto, o a lo ancho
    - A lo alto: aumentar la capacidad de los servidores → Muy caro
    - A lo ancho: aumentar el número de servidores, muchos y pequeños → aparición de los cluster

1. Introducción a la tecnología NoSQL

- Necesidad de la clusterización:
  - **Cluster:**
    - Multitud de pequeños ordenadores funcionando conjuntamente
    - Usan hardware común y corriente → instalaciones más baratas
    - Resistentes: aunque el ordenador individual puede fallar, el impacto en el cluster es asumido, lo que aumenta su fiabilidad
    - Grandes compañías se orientaron a esta solución

1. Introducción a la tecnología NoSQL

- Necesidad de la clusterización:
  - **Bases de datos relacionales:**
    - No están diseñadas con esta orientación
    - Concepto de cluster existe pero en dos formas:
      - Cluster con un único **subsistema de discos** de alta disponibilidad
      - Servidores distintos con división de los datos (***sharding***): las aplicaciones han de ser sensibles, se pierde representación de restricciones
    - Costes de licencias muy altos
    - Consideración de otras alternativas:
      - En la década de 2000
      - Google: BigTable
      - Amazon: Dynamo

1. Introducción a la tecnología NoSQL

- Emergencia de NoSQL:
  - **Término NoSQL:**
    - En 2009, reunión de bases de datos "no-relacionales, distribuidas, y open-source": Voldemort, Cassandra, Dynomite, HBase, Hypertable, CouchDB, and MongoDB
    - El término no se ha confinado a esta reunión original
    - No hay una definición aceptada, ni una autoridad que lo proporcione

1. Introducción a la tecnología NoSQL

- Emergencia de NoSQL:
  - **Carácterísticas NoSQL:**
    - Proyectos open-source, generalmente
    - Orientados a funcionar sobre clusters (no todas): efectos
      - Modelo de datos: no usan el modelo relacional, modelos mucho más ricos
      - Gestión de la consistencia: alteran las propiedades ACID
    - Nacen en el siglo XXI: no se consideran modelos antiguos (pre Codd)
    - Operan sin esquema de base de datos: mucha mayor flexibilidad para hacerse cargo de las modificaciones de diseño

1. Introducción a la tecnología NoSQL

- Clasificación (por modelo de datos):

Clase	Base de datos
Familia Columnas	Accumulo, Cassandra, Druid, Hbase, BigTable, Hypertable, …
Documento	Clusterpoint, CouchDB, Couchbase, MarkLogic, MongoDB, …
Clave-Valor	Dynamo, FoundationDB, MemcacheDB, Redis, Riak, Voldemort, …
Grafo	Allegro, Neo4J, InfiniteGraph, OrientDB, Virtuoso, Stardog, …

Índice

1. Introducción a la tecnología NoSQL
2. Modelos de datos
   2.1. Modelos de datos agregados
   2.2. Otros modelos y aspectos de modelado
3. Modelos de distribución de datos
4. Consistencia
5. Marcados de versión

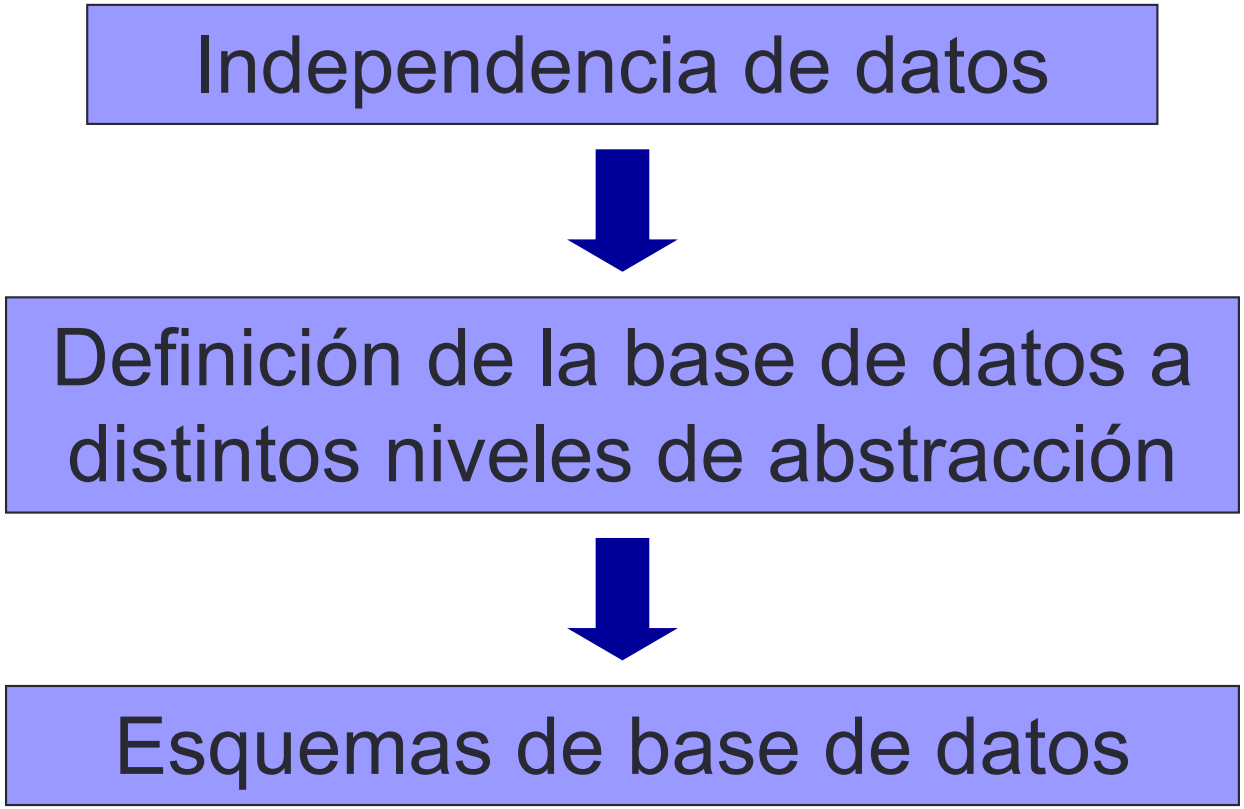
2. Modelos de datos

- Modelo de datos:
  - Herramienta intelectual que permite representa las propiedades estáticas y dinámicas de la parcela del mundo real bajo estudio
  - Están compuestos por:
    - Estructuras de datos
    - Operadores asociados (lenguajes de definición y manipulación)

- **Modelo de datos ←→ Esquema de datos**

2. Modelos de datos

- Independencia de los datos:



2. Modelos de datos

- Arquitectura de niveles del SGBD:
  - **Esquema lógico**: definición de las estructuras de datos de la base de datos (modelos de datos lógico)
  - **Esquema físico**: implementación de las estructuras de datos del esquema lógico (modelo de almacenamiento o físico)
  - **Esquemas externos**: subconjuntos del esquema lógico (vistas parciales)

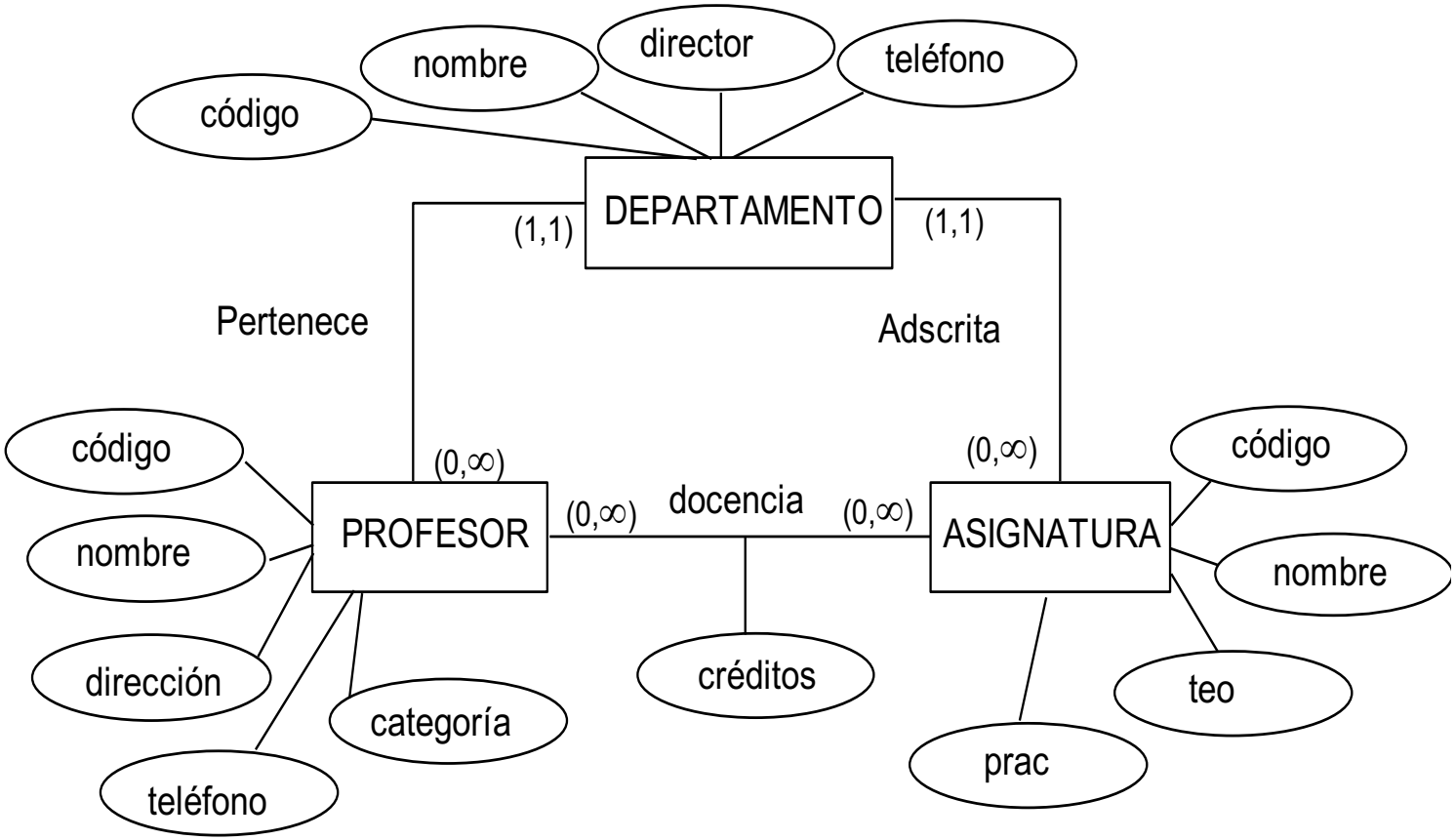
2. Modelos de datos

Familias de SGBD:

Década	SGBD	Modelo	Estructuras
50	Sistemas de ficheros	Secuencia registros	Registro, secuencia (fichero)
60	Jerárquico	Jerárquico	Registro (segmento), árbol
60-70	En red	Red	Registro, lista (set)
80-00	Relacionales	Relacional	Registro (tupla), relación
90-07	Objeto-Relacionales	Relacional+ OO	Registro (tupla), relación, constructores de tipos
90-07	OO	OO	Constructores de tipos
08-act.	NoSQL	Diversos	Múltiples (agregación, complejidad)

2. Modelos de datos

Ejemplo: Esquema BD en Entidad-Relación



2. Modelos de datos

Hay dos departamentos con la siguiente información:

- código = DSIC, nombre = Sistemas Informáticos y Computación, director = Juan García, teléfono = 3570
- código = DFA, nombre = Física, director = José Ruíz, teléfono = 3540.

Hay tres profesores:

- código = JCP, nombre = Juan Cerdá Pérez, dirección = Olta 23, teléfono = 3222, categoría = TEU
- código = LBP, nombre = Luis Bos Pérez, dirección = Jesús 91, teléfono = 3545, categoría = TEU
- código = PMG, nombre = Pedro Martí García, dirección = Cuenca 12, teléfono = 3412, categoría = TEU

Hay tres asignaturas:

- código = AD1, nombre = Algoritmos y estructuras de datos 1, teo = 3, prac = 3
- código = IP, nombre = Introducción a la programación, teo = 1.5, prac = 1.5
- código = AD2, nombre = Algoritmos y estructuras de datos 2, teo = 3, prac = 3

Respecto a las relaciones, la información es la siguiente:

- los tres profesores pertenecen al DSIC.
- las tres asignaturas están adscritas al DSIC.
- Juan Cerdá Pérez imparte 9 créditos de AD1 y 9 de IP, Luis Bos Pérez imparte 9 créditos de IP y Pedro Martí García imparte 9 créditos de AD1.

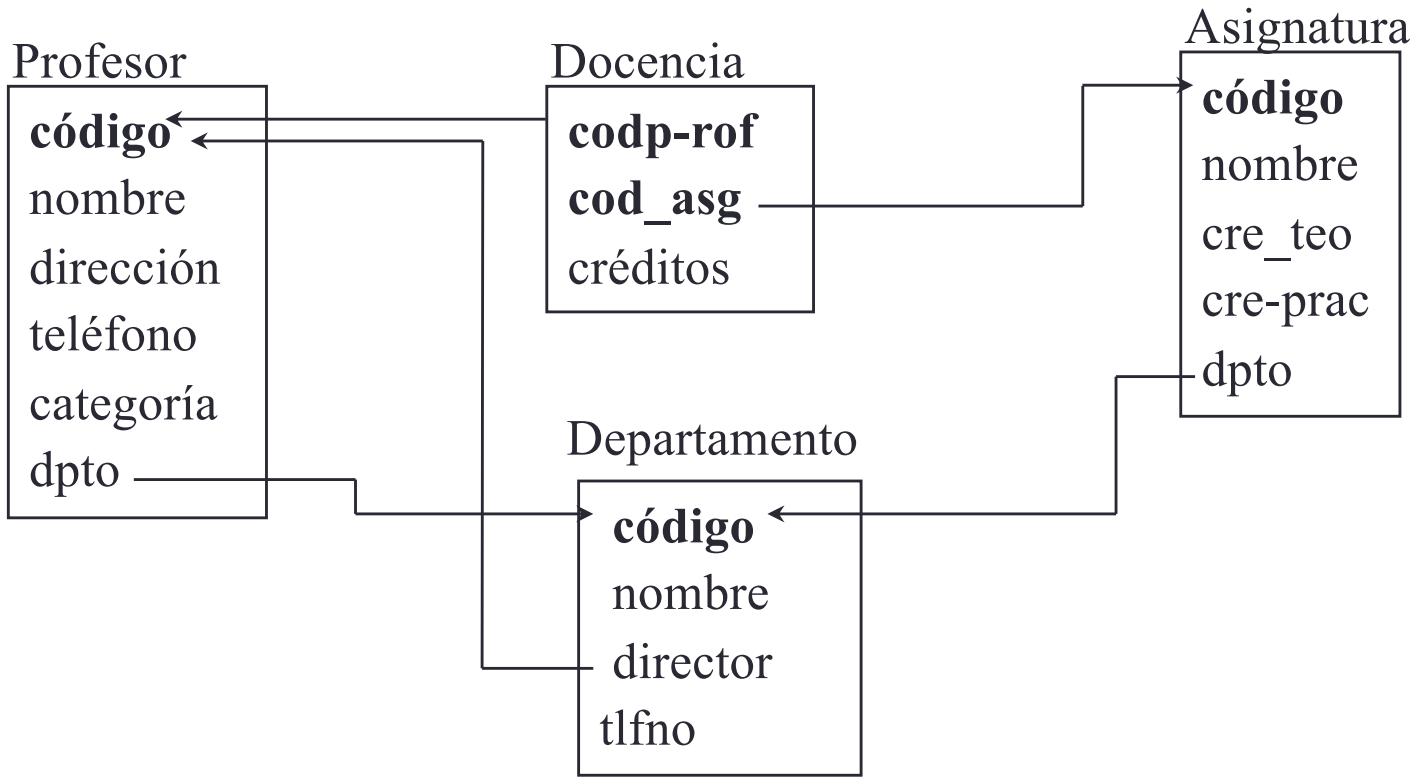
2. Modelos de datos

Ejemplo: Esquema BD en Modelo Relacional

Departamento(código:dom_codd, nombre:dom_nomd,
director:dom_dir, teléfono:dom_tel)
Clave Primaria: {código}
Profesor (código:dom_codp, nombre:dom_nomp, dirección: dom_dir,
teléfono: dom_tel, categoría: dom_cat, dpto:dom_codd)
Clave Primaria: {código}
Clave Ajena: {dpto} referencia a Departamento
Valor No Nulo: {dpto}
Asignatura(código:dom_coda, nombre:dom_noma, cre_teo:dom_cre,
cre_prac:dom_cre, dpto:dom_codd)
Clave Primaria: {código}
Clave Ajena: {dpto} referencia a Departamento
Valor No Nulo: {dpto}
Docencia (cod_prof:dom_codp, cod_asg:dom_coda,
créditos:dom_cre)
Clave Primaria: {cod_prof,cod_asg}
Clave Ajena: {cod_prof} referencia a Profesor
Clave Ajena: {cod_asg} referencia a Asignatura

2. Modelos de datos

Ejemplo: Esquema BD en Modelo Relacional



2. Modelos de datos

Ejemplo: Base de Datos estructurada según Modelo Relacional

Relación Departamento

Código	Nombre	Director	Teléfono
DSIC	Sistemas Informáticos y Computación	Juan García	3570
DFA	Física	José Ruíz	3540

Relación Profesor

Código	Nombre	Dirección	Categoría	Teléfono	Dpto
JCP	Juan Cerdá Pérez	Olta 23	TEU	3222	DSIC
LBP	Luis Bos Pérez	Jesús 91	TEU	3545	DSIC
PMG	Pedro Martí García	Cuenca 12	TEU	3412	DSIC

Relación Asignatura

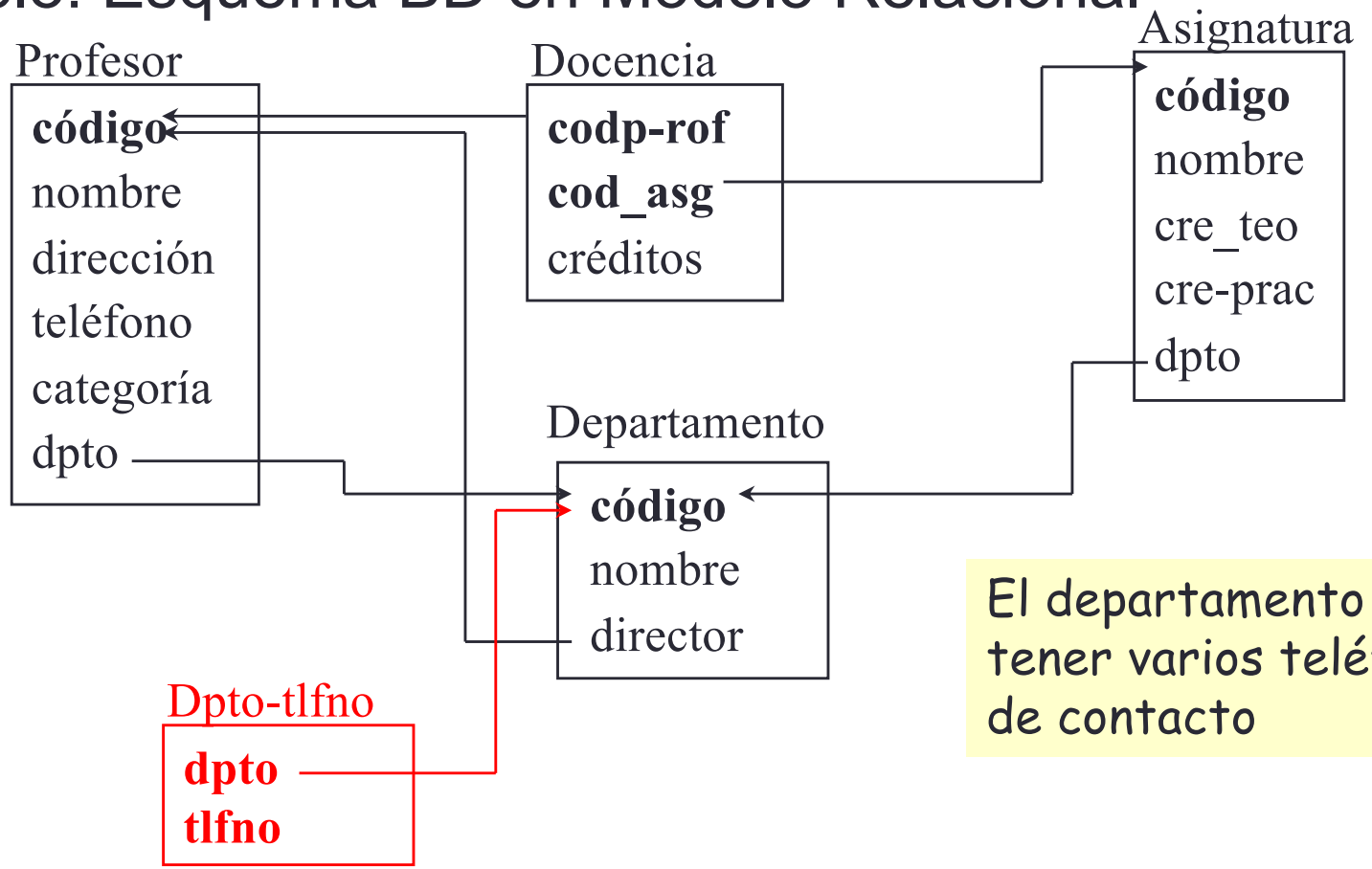
Código	Nombre	Teo	Prac	Dpto
AD1	Algoritmos y estructuras de datos I	3	3	DSIC
IP	Introducción a la programación	1.5	1.5	DSIC
AD2	Algoritmos y estructuras de datos II	3	3	DSIC

Relación Docencia

cod_prof	cod_asg	créditos
JCP	AD1	9
JCP	IP	9
LBP	IP	9
PMG	AD1	9

2. Modelos de datos

Ejemplo: Esquema BD en Modelo Relacional



El departamento puede tener varios teléfonos de contacto

Limitaciones del modelo relacional (SQL) en la definición de información compleja

- dominios escalares
- ausencia de primitivas para definir objetos complejos

2. Modelos de datos

Ejemplo: Esquema BD en Modelo Relacional

" Obtener las asignaturas impartidas por el profesor de código 'LBP' ":

```
SELECT  Asignatura.nombre
FROM  Asignatura, Docencia
WHERE  Asignatura.código  = Docencia.cod_asg
                                    and
                          Docencia.cod_prof  = 'LBP'
```

Consulta de la base de datos

- lenguajes declarativos (SQL)
- manipulación explícita de las relaciones (JOIN) para reconstruir la información de un objeto.

2.1. Modelos de datos agregados

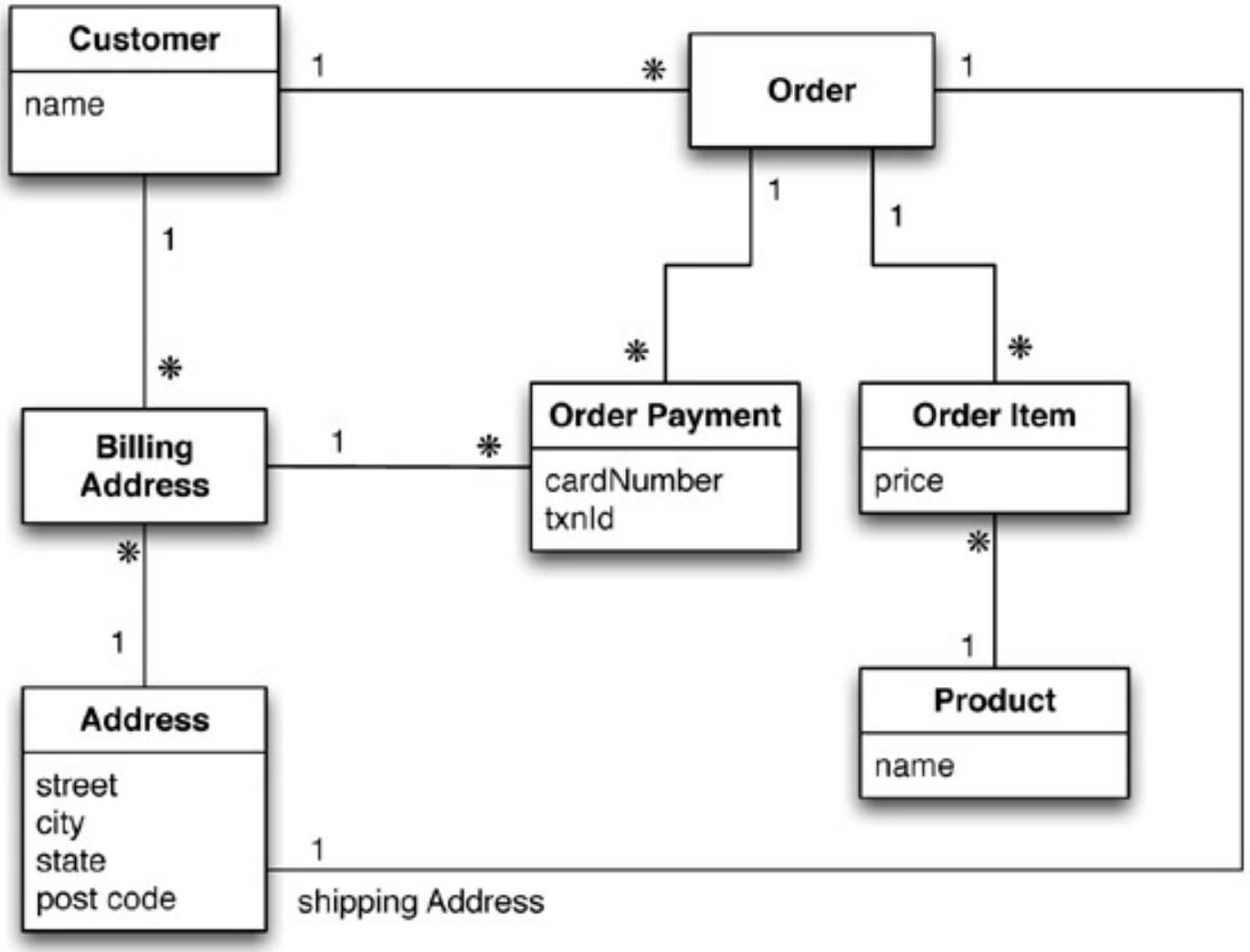
- Agregados en MR:
  - Modelos Relacional exige simplicidad y mínima complejidad en los datos representados: Tupla (registro, fila), Relación (conjunto, tabla)
  - En el Modelo Relacional sólo se admite la agregación que permite la estructura tupla
  - Lenguajes relacionales de manipulación se apoyan en esta simplicidad, son declarativos y comprensibles.

2.1. Modelos de datos agregados

- Agregados:
  - Sistemas de bases de datos basados en clave-calor, familias de columnas y documentos usan el concepto de registro complejo (registro con campos que pueden ser estructuras complejas, listas, registros, etc.)
  - Definición:
    - **Colección de objetos relacionados que se desea tratar como una unidad**
  - Agregado es una unidad para la manipulación de datos y para la gestión de la consistencia.
  - La gestión de datos agregados es mucho más sencilla para bases de datos en cluster: unidad natural para la replicación y la división.
  - Programación es también más natural.

2.1. Modelos de datos agregados

- Ejemplo en UML orientado a BDR:



2.1. Modelos de datos agregados

- Ejemplo: ocurrencia de BDR

Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

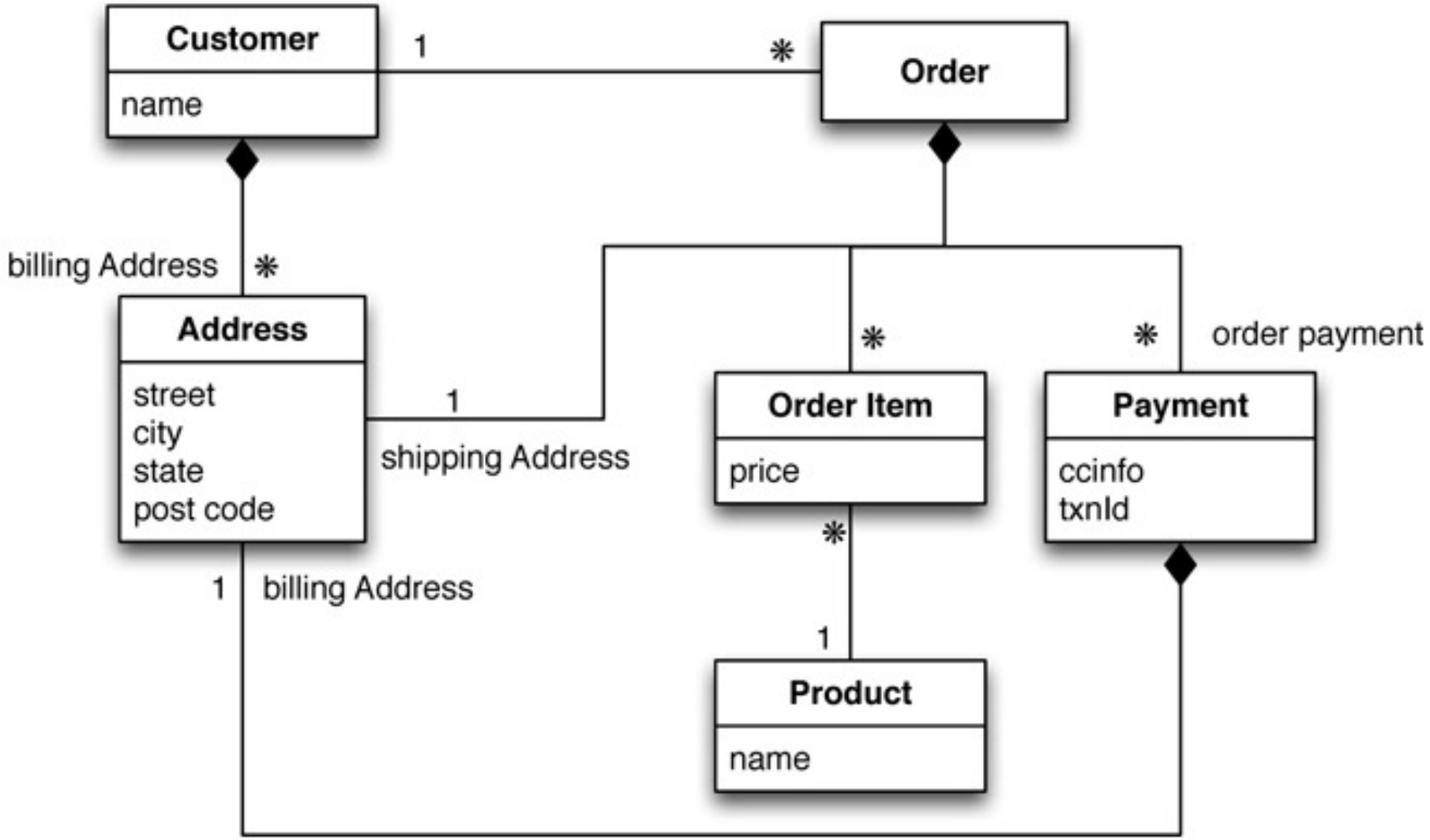
Id	City
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

2.1. Modelos de datos agregados

- Ejemplo en modelo de datos agregados: **Customer Order**



2.1. Modelos de datos agregados

- Ejemplo: ocurrencia en JSON (**Customer Order**)

```
// in customers
{  "id":1,
   "name":"Martin",
   "billingAddress": [ {"city":"Chicago"}] }

// in orders
{  "id":99,
   "customerId":1,
   "orderItems":
         [ {   "productId":27,
               "price": 32.45,
               "productName": "NoSQL Distilled" }  ],
   "shippingAddress": [ { "city":"Chicago" } ],
   "orderPayment":
         [ {   "ccinfo":"1000-1000-1000-1000",
               "txnId":"abelif879rft",
               "billingAddress": {"city": "Chicago"}  }  ]
}
```

2.1. Modelos de datos agregados

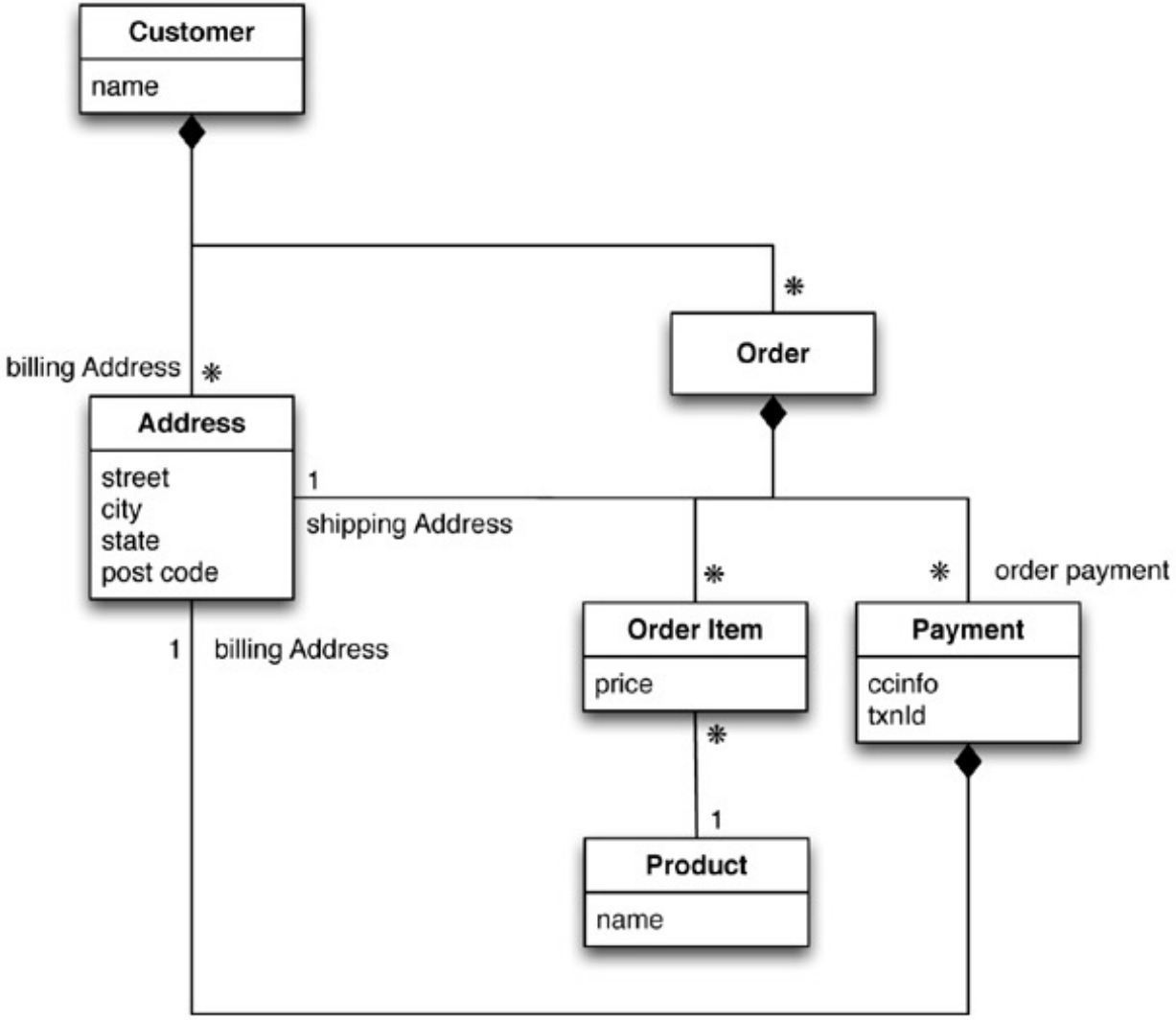
- Ejemplo en modelo de datos agregados: **Customer Order**
  - Dos agregados principales: Customer y Order
  - Composición UML para formar los agregados (◆)
  - En la base de datos el mismo registro de *billing address* aparece repetido 3 veces
    - *Billing address* y *Shipping address* no pueden cambiar
    - BDR se complica está representación (necesidad de restricciones dinámicas)
    - Con la agregación se copia el valor agregado
  - Asociación (relación) entre *Customer* y *Order*
  - Asociación entre *Order Item* y Product:
    - Desnormalización: **nombre** del producto dentro de *Order Item*
    - Muy habitual en modelo de datos agregados para reducir el número de datos agregados a los que se accede en la interacción con los datos.

2.1. Modelos de datos agregados

- Es necesario pensar en el acceso a los datos para obtener el esquema de la base de datos
- Ejemplo en modelo de datos agregados: **Customer**
  - Realizamos otra forma distinta de agregación
  - Sólo consideramos un objeto agregado

2.1. Modelos de datos agregados

- Ejemplo en modelo de datos agregados: **Customer**



2.1. Modelos de datos agregados

- Ejemplo: ocurrencia en JSON (**Customer**)

```
// in customers
{  "customer": {
      "id":1,
      "name":"Martin",
      "billingAddress": [ {"city":"Chicago"}],
      "orders":  [
         {  "id":99,
            "customerId":1,
            "orderItems":  [
                { "productId":27,
                  "price": 32.45,
                  "productName": "NoSQL Distilled" }  ],
      "shippingAddress": [ { "city":"Chicago" } ],
      "orderPayment": [
          { "ccinfo":"1000-1000-1000-1000",
            "txnId":"abelif879rft",
            "billingAddress": {"city": "Chicago"}  } ]
         }]
   } }
```

2.1. Modelos de datos agregados

- No hay una respuesta universal sobre la forma de realizar la agregación de objetos
- Depende de la estructura y de la **forma de manipulación de los datos**

2.1. Modelos de datos agregados

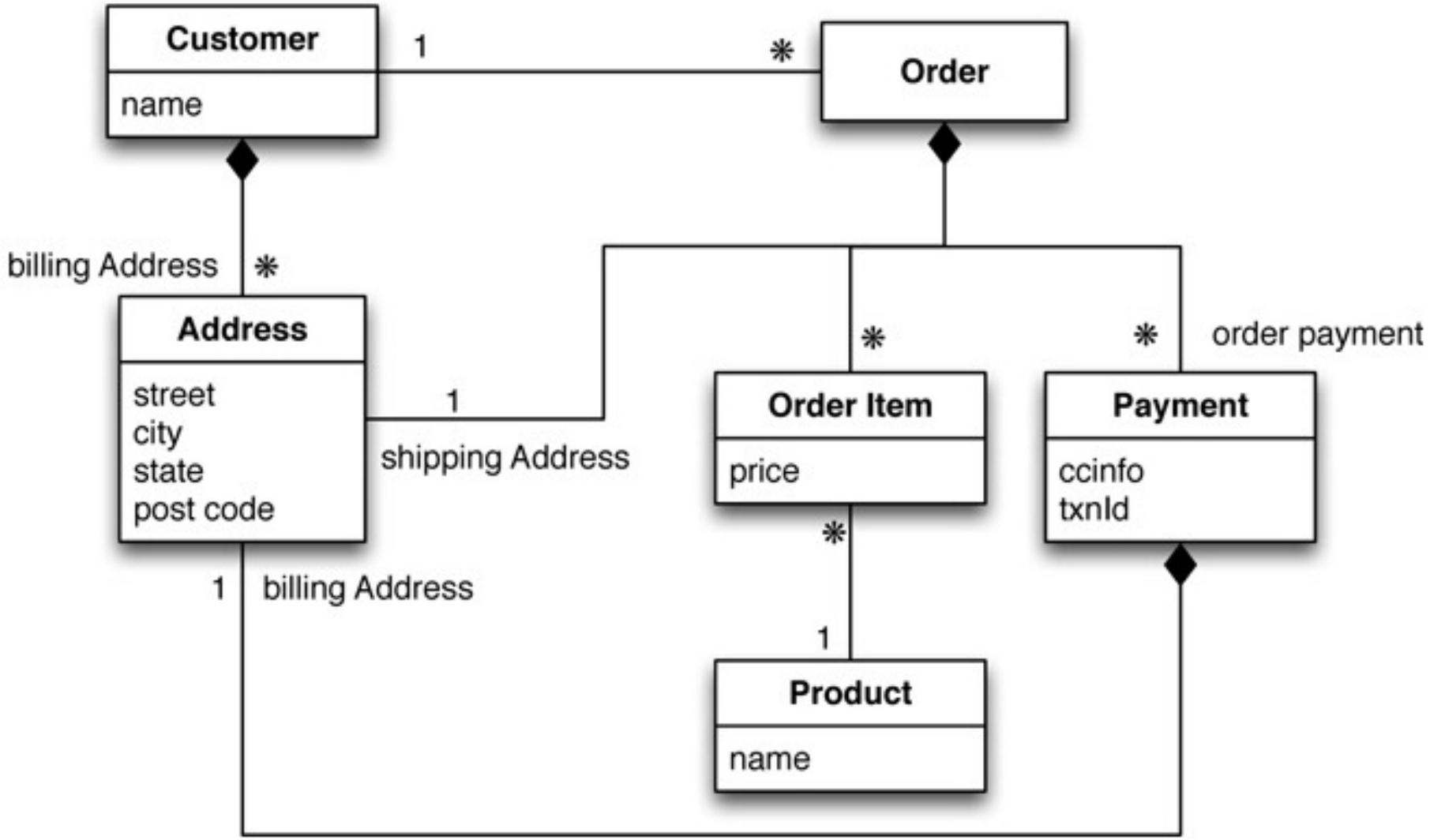
- Consecuencias de la orientación a los agregados:
  - Modelo relacional
    - Puede capturar adecuadamente varios elementos y las relaciones existentes entre ellos
    - No tiene conocimiento de la noción de agregación
    - EJEMPLO: un pedido consiste en artículos solicitados, una dirección de envió y un pago
    - Las claves ajenas sirven para establecer y conocer las relaciones entre objetos (**asociaciones y agregaciones**)
    - La base de datos no puede usar el conocimiento de la estructura de las agregaciones para almacenar y distribuir los datos
    - Las agregaciones tiene que ver con cómo se usan los datos por parte de las aplicaciones, no con la estructura de los datos

2.1. Modelos de datos agregados

- Consecuencias de la orientación a los agregados:
  - Modelo relacional
    - Ignorante a la agregación
    - Bases de datos NoSQL basadas en grafos también
    - Ser conocedor y realizar una buena representación es complicado
    - Muchas veces hay usos diversos de los datos y las agregaciones no están claras
    - EJEMPLO:
      - Aplicaciones que gestionan la información por pedidos
      - Aplicaciones que gestionan las ventas de cada artículos
    - Los modelos ignorantes de la agregación permiten una forma flexible de manipular los datos

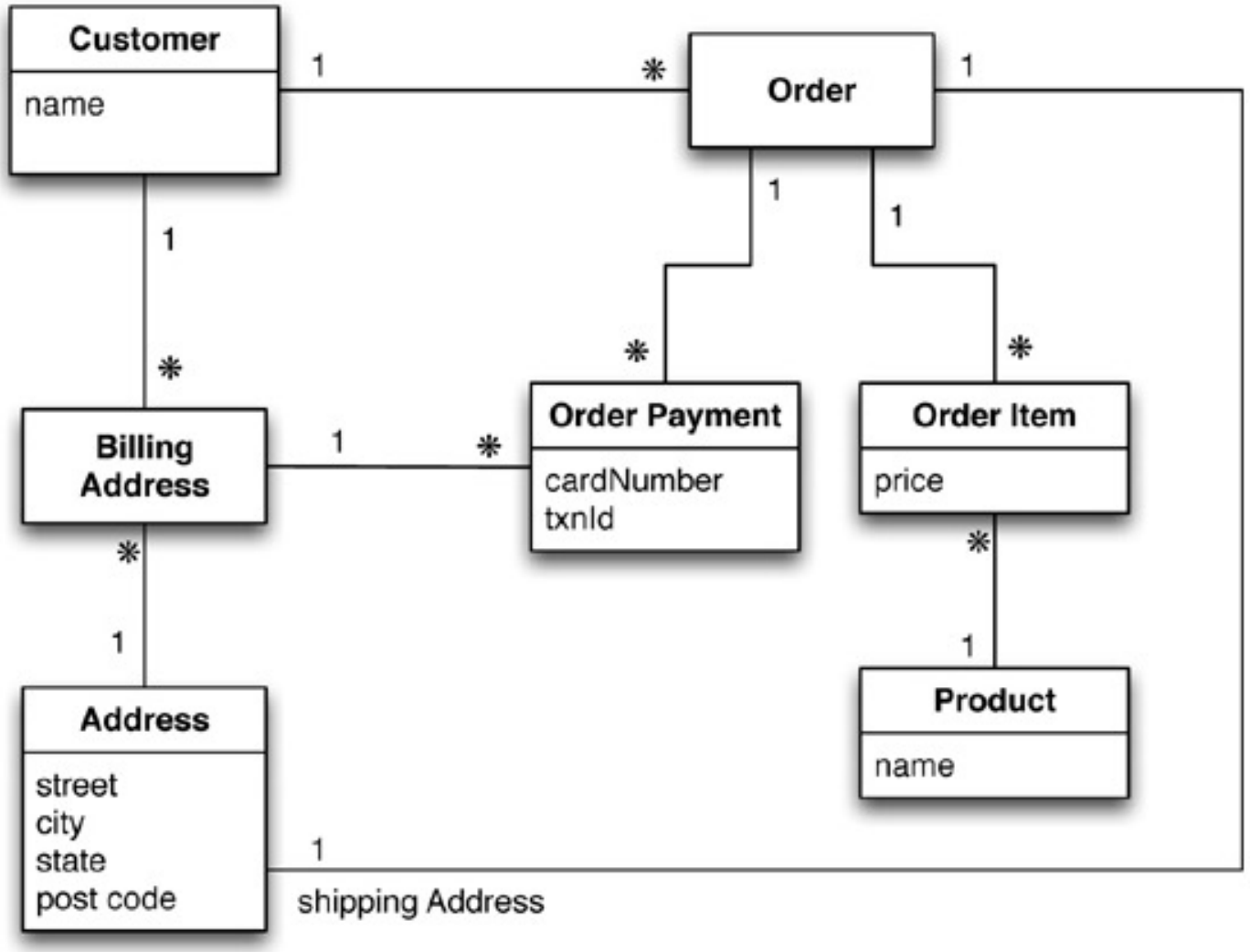
2.1. Modelos de datos agregados

- Ejemplo en modelo de datos agregados: **Customer Order**



2.1. Modelos de datos agregados

- Ejemplo en UML orientado a BDR:



2.1. Modelos de datos agregados

- Consecuencias de la orientación a los agregados:
  - Transacciones (ACID en BDR)
    - BDR manejan un conjunto de múltiples tuplas de varias relaciones en una transacción
    - **Atomicidad** asegura que no hay problemas con ejecuciones parciales
    - **Aislamiento** asegura que no va a haber efectos anómalos por la concurrencia

2.1. Modelos de datos agregados

- Consecuencias de la orientación a los agregados:
  - Transacciones (NoSQL)
    - Sistemas NoSQL no tienen propiedades ACID que aseguran una correcta manipulación para operaciones que abarcan varios objetos agregados
    - Si se desea una manipulación atómica se ha de programar en el código de la aplicación
    - Habitualmente un buen diseño de los objetos agregados permite que la manipulación se circunscriba exclusivamente a uno de ellos, con la que la atomicidad está asegurada
    - Bases de datos basadas en grafos y otras, que son ignorantes de la agregación, soportan ACID

2.1. Modelos de datos agregados

- Modelos de datos de clave-valor y de documentos:
  - Orientados a agregación
    - Principal forma de estructurar los datos
    - Base datos almacena un conjunto de objetos agregados, cada uno de ellos con una clave o ID que lo identifica y sirve para acceder a él.
  - Clave-valor:
    - La agregación es opaca: el objeto es un blob sin estructura interna
    - Se puede almacenar cualquier cosa dentro de un objeto
    - Puede haber algún limite de tamaño
    - Acceso a los objetos en los lenguajes de manipulación es siempre haciendo uso de la clave

2.1. Modelos de datos agregados

- Modelos de datos de clave-valor y de documentos:
  - Orientados a agregación
    - Principal forma de estructurar los datos
    - Base datos almacena un conjunto de objetos agregados, cada uno de ellos con una clave o ID que lo identifica y sirve para acceder a él.
  - Documentos:
    - Estructura interna conocida
    - Limita lo que se puede almacenar dentro del objeto agregado
    - Mayor flexibilidad al acceder al conocerse la estructura
    - Acceso a los objetos refiriéndose a los campos de la estructura interna de los objetos agregados (físicamente se pueden crear índices sobre estos campos)

2.1. Modelos de datos agregados

- Modelos de datos de clave-valor y de documentos:
  - La diferencia entre ellos es borrosa
    - BD de documentos se incluyen campos ID en los documentos
    - BD clave-valor admiten estructura para los datos agregados, eliminando la opacidad
  - Aunque la idea general y el uso de los datos es el comentado

2.1. Modelos de datos agregados

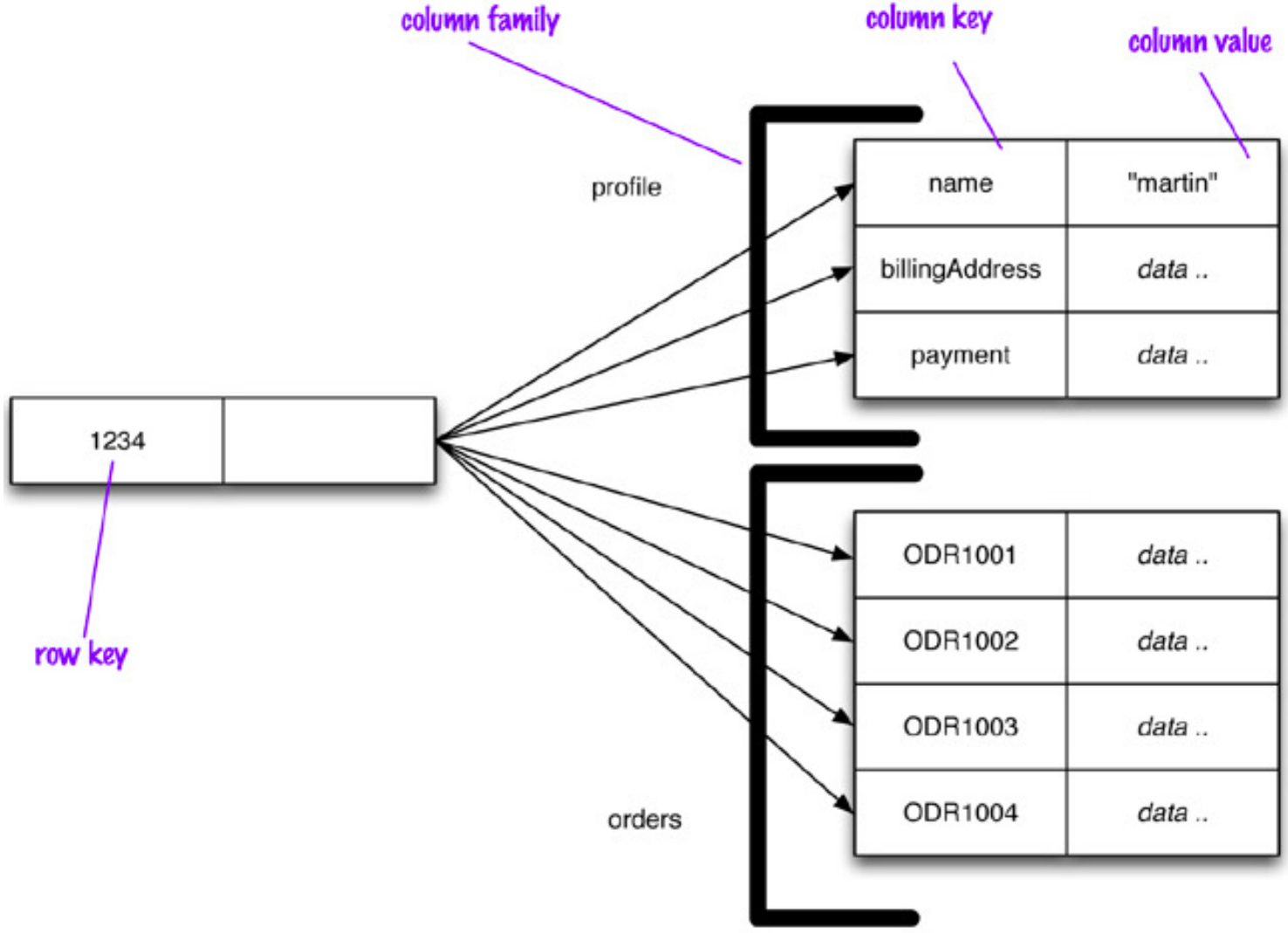
- Modelos de datos de familia de columnas:
  - BigTable de Google
  - Estructura tabular con columnas dispersas y sin esquema
  - Se asemeja a un mapeo de dos niveles
  - Influencia sobre Hbase y Cassandra
  - También se conocen como almacenes de columnas
    - BDR la unidad de almacenamiento fila (registro): ayuda para tener un buen rendimiento en la escritura
    - Almacenes de columnas: las escrituras son raras y las lecturas son de algunas columnas de muchas filas → la unidad de almacenamiento son grupos de columnas de todas las filas

2.1. Modelos de datos agregados

- Modelos de datos de familia de columnas:
  - Estructura agregada de dos niveles
  - 1er nivel: objetos (*filas*) se agregan y se accede por una clave (similar al modelo clave-valor)
  - 2º nivel: el *valor* está estructurado en familias de columnas
  - Operaciones de acceso pueden recuperar todo un objeto (fila) o una columna en particular

2.1. Modelos de datos agregados

- Modelos de datos de familia de columnas:



2.1. Modelos de datos agregados

- Modelos de datos de familia de columnas:
  - Agrupación de columnas en familias es por razones de forma de acceso: se accede conjuntamente a todas las columnas de una familia
  - Modelo:
    - Orientado a fila: cada fila es un objeto agregado con familias de columnas que representan trozos de datos del objeto
    - Orientado a columna: cada familia de columnas define un tipo de registro, donde las filas serían los registros
  - Las columnas de las filas no son homogéneas, se puede añadir columnas en cualquier momento.

2.1. Modelos de datos agregados

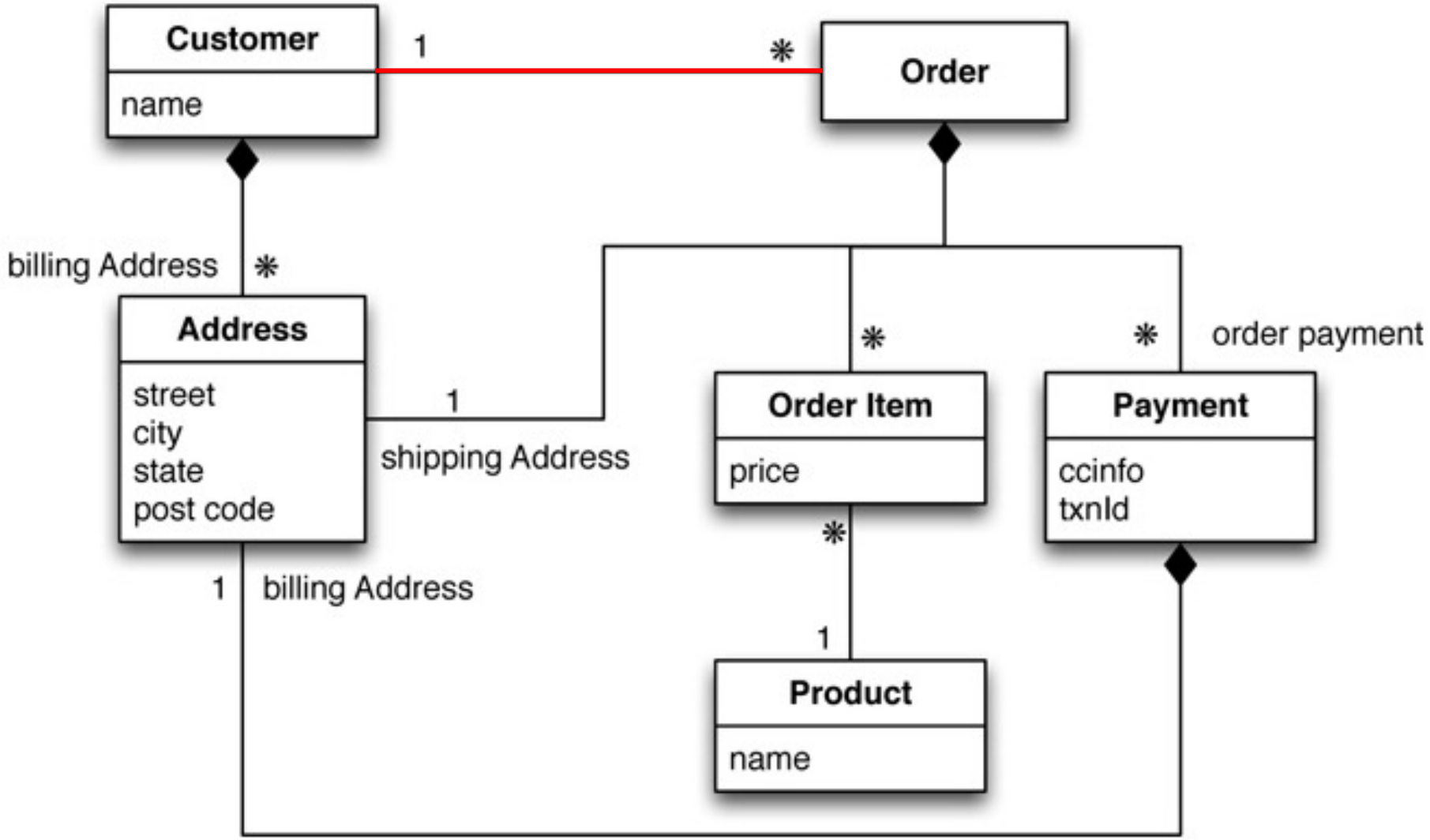
- Modelos de datos de familia de columnas:
  - Familia de columnas *Profile*:
    - Similar a una tabla, fuerte homogeneidad en el número de columnas entre filas
  - Familia de columnas *Orders*:
    - Los nombre de columnas pueden ser valores de otras columnas.
    - Listas de ítems se modelan como columnas
    - Muy poca homogeneidad en el número de columnas entre filas
  - Ordenación de las columnas:
    - Las columnas en una familia de columnas están ordenadas.
    - Favorece los accesos en algún caso (fecha, tiempo, etc.)

2.2. Otros modelos y aspectos de modelado

- Relaciones:
  - Agregación es útil para reunir los datos a los que se accede a la vez.
  - Pero hay otros accesos donde se usan datos relacionados
  - Ejemplo:
    - Relación entre *Customer* y *Order*
    - Algunas aplicaciones quieren acceder al cliente y todos sus pedidos → Agregar *Order* en *Custormer*
    - Otras aplicaciones quieren acceder a un pedido de forma individualizada → *Order* y *Customer* dos objetos agregados distintos

2.2. Otros modelos y aspectos de modelado

- Relaciones: Ejemplo **Customer Order**



2.2. Otros modelos y aspectos de modelado

- Ejemplo: ocurrencia en JSON (**Customer Order**)

```
// in customers
{  "id":1,
   "name":"Martin",
   "billingAddress": [ {"city":"Chicago"}] }

// in orders
{  "id":99,
   "customerId":1,
   "orderItems":
         [ {   "productId":27,
               "price": 32.45,
               "productName": "NoSQL Distilled" }  ],
   "shippingAddress": [ { "city":"Chicago" } ],
   "orderPayment":
         [ {   "ccinfo":"1000-1000-1000-1000",
               "txnId":"abelif879rft",
               "billingAddress": {"city": "Chicago"}  }  ]
}
```

2.2. Otros modelos y aspectos de modelado

- Relaciones:
  - Si la aplicación que accede a *Order* necesita conocer el cliente, realiza otro acceso a *Customer* con el *customerId*
  - A veces es conveniente que la base de datos sea conocedora de esta relación entre los dos objetos
  - Distintos sistemas (orientados a clave-valor o a documentos) permiten que esta relaciones sean visibles de diversas formas (indexación, navegación de enlaces entre objetos, etc.)
  - Actualización:
    - La unidad atómica de actualización es el objeto agregado.
    - La actualización de objetos relacionados ha de ser tratada en la aplicación
    - Los sistemas relacionales ayudan forzando la gestión transaccional ACID

2.2. Otros modelos y aspectos de modelado

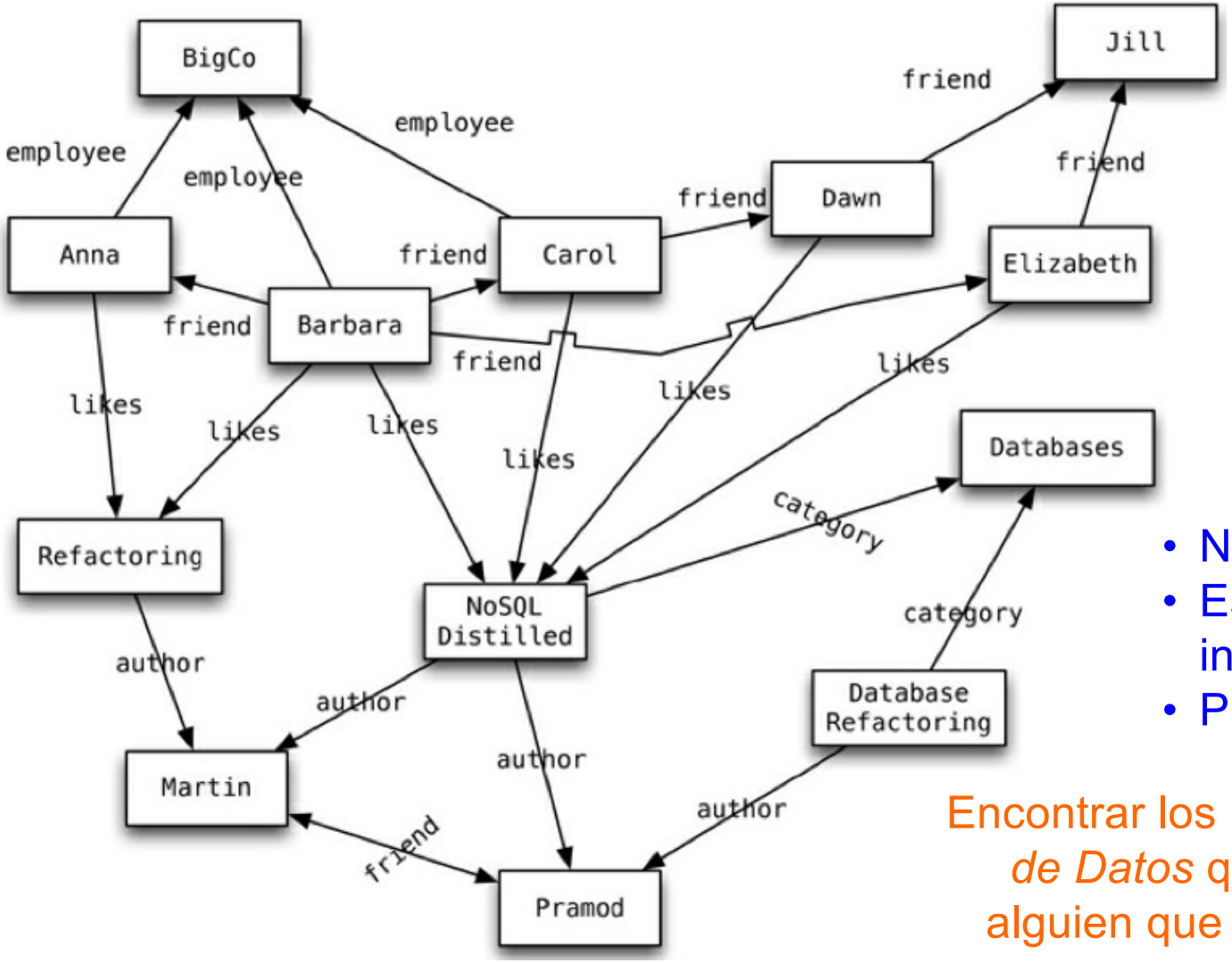
- Relaciones:
  - Si hay muchas operaciones con objetos agregados relacionados hay que plantearse si las bases de datos NoSQL orientadas a datos agregados son las más adecuadas
  - Solución puede ser BDR, aunque si hay mucha complejidad relacional tampoco son la solución:
    - Consultas con muchos JOINs
    - Complejidad en su definición
    - Rendimiento en su ejecución

2.2. Otros modelos y aspectos de modelado

- Bases de datos orientadas a grafos:
  - Se las incluye en NoSQL, aunque no comparten algunas de sus características
  - NoSQL:
    - Ejecución en cluster
    - Modelo de datos agregados
  - Bases de datos orientadas a grafos: registros pequeños con interrelaciones complejas

2.2. Otros modelos y aspectos de modelado

- Bases de datos orientadas a grafos:



- Nodos con pocos datos
- Estructura de interconexiones muy rica
- Preguntas complejas:

Encontrar los libros de la categoría *Bases de Datos* que han sido escritos por alguien que le agrada a un amigo mío

2.2. Otros modelos y aspectos de modelado

- Bases de datos orientadas a grafos:
  - Especializadas en manejar este tipo de datos
  - Modelo de datos: en esencia
    - Nodos
    - Arcos
  - Variaciones en los datos que se pueden representar en los nodos y arcos
  - Lenguajes de interrogación orientados a grafos
    - BDR es necesario realizar JOINs para interrogar sobre datos muy interconectados → complejo y mal rendimiento
    - BD orientadas a grafos:
      - Las consultas están preparadas para este tipo de datos
      - Convierten el coste de las consultas sobre grafos a coste de actualización

2.2. Otros modelos y aspectos de modelado

- Bases de datos orientadas a grafos:
  - Modelo de datos:
    - Especializadas en manejar datos en forma de grafos
    - Muy diferentes a las BD orientadas a datos agregados
  - Implicaciones:
    - Cluster: están orientadas a funcionar en un único servidor
    - Gestión transaccional: ACID, necesidad de actualizar múltiples nodos y arcos de forma consistente
  - ¿Por qué en la familia NoSQL?:
    - Rechazo al Modelo Relacional
    - Nacimiento común

2.2. Otros modelos y aspectos de modelado

- Bases de datos sin esquema:
  - BDR es necesario lo primero definir un esquema, después ya se puede empezar a almacenar datos
  - NoSQL el almacenamiento de datos es más relajado:
    - Clave-valor: se puede almacenar lo que sea asociado con la clave
    - Documento: más o menos lo mismo, con algo más de estructura
    - Familia de columnas: en una fila se puede almacenar cualquier valor asociado a una columna
    - Grafos: se pueden añadir cuando se desee más tipos de arcos y propiedades a los nodos y arcos

2.2. Otros modelos y aspectos de modelado

- Bases de datos sin esquema:
  - Ausencia de esquema: libertad y flexibilidad
    - A veces no se tiene el conocimiento preciso de los datos
    - Según avanza el conocimiento se incluyen más datos
    - Si algún dato se queda obsoleto se deja de almacenar
  - Mejor manejo de datos heterogéneos:
    - Cada objeto de un tipo presenta información distinta
    - Presencia de valores nulos en BDR → desaprovechamiento
  - La ausencia de esquema provoca problemas:
    - No existe metalenguaje sobre tu universo de discurso → dificultades para las aplicaciones
    - No existe tipado de los datos → dificulta la manipulación de los datos

2.2. Otros modelos y aspectos de modelado

- Bases de datos sin esquema:
  - Siempre existe esquema aunque no sea explícito → esquema implícito
  - Esquema implícito dentro del código de las aplicaciones que usan los datos
    - Conocimiento del esquema a través del código:
      - Dificulta el entendimiento
      - ¿Múltiples aplicaciones?
      - Encapsular el acceso en un única aplicación que dé servicio al resto
      - División de la base de datos por áreas de trabajo (aplicaciones distintas)
    - Base de datos ignora el esquema → imposibilita una buena gestión

2.2. Otros modelos y aspectos de modelado

- Bases de datos sin esquema:
  - BDR y las anteriores tienen un esquema fijo porque es un **valor**, es útil.
  - BDR no tienen un esquema fijo: se puede cambiar con algo de coste
  - BDR no son adecuadas para tratar datos heterogéneos
  - NoSQL las facilidades de variación de esquema sólo se dan dentro de los datos agregados, si los cambios van más allá, el impacto es similar al de las BDR

2.2. Otros modelos y aspectos de modelado

- Vistas materializadas:
  - Modelo de datos agregados: agregación de los datos a los que se accede → beneficio en el acceso

    Pedido es un objeto agregado
  - Desventaja: consultas de muchos objetos agregados

    Cuántas unidades se han vendido de un artículo
    - Es necesario revisar todos los objetos Pedido buscando ese artículo (se puede indexar, pero aún así es costoso)
  - BDR no tienen este problema ya que no realizan agregaciones
  - BDR la forma de interrogar es flexible

2.2. Otros modelos y aspectos de modelado

- Vistas materializadas:
  - Vista es una consulta almacenada: cuando el usuario la usa, la consulta asociada es ejecutada y se devuelve el resultado
  - Vista materializada es una vista cuyo resultado se almacena explícitamente en disco → Necesidad de mantener la redundancia
  - Mantenimiento de la vista materializada:
    - En la actualización de los datos base de los que depende → sobrecarga de la actualización
    - Regularmente: se establece un periodo máximo de obsolescencia
  - La vista materializada se suele definir y almacenar dentro del objeto agregado (en el modelo orientado a familia de columnas, una vista materializada es una familia)

Índice

1. Introducción a la tecnología NoSQL
2. Modelos de datos
   2.1. Modelos de datos agregados
   2.2. Otros modelos y aspectos de modelado
3. Modelos de distribución de datos
4. Consistencia
5. Marcados de versión

3. Modelos de distribución de datos

- NoSQL:
  - Necesidad de ejecutarse en grandes cluster
  - Incremento del volumen de los datos → Incremento del coste de almacenarlo en un único servidor
  - Escalar la instalación → un gran cluster de servidores
  - La orientación a datos agregados se adecúa a esta arquitectura → el dato agregado es la unidad de distribución

3. Modelos de distribución de datos

- NoSQL: Distribución
  - Necesaria para cuando se quieren gestionar grandes volúmenes de datos
  - Procesando un tráfico alto de lecturas y escrituras
  - Hacer frente a problemas en la red (cortes o lentitud)
  - Incrementa la complejidad:
    - La arquitectura es más compleja
    - El equipo humano ha de tener el conocimiento y la experiencia

3. Modelos de distribución de datos

- NoSQL: Formas de distribución
  - Replicación:
    - Copiar el mismo dato en varios nodos
  - Fragmentación (*Sharding*):
    - Colocar datos distintos en distintos nodos
  - Técnicas ortogonales: se puede aplicar una de ellas o las dos
  - Arquitecturas de replicación:
    - Servidor único
    - Maestro-Esclavo
    - *Peer-to-peer* ("Entre pares")

3. Modelos de distribución de datos

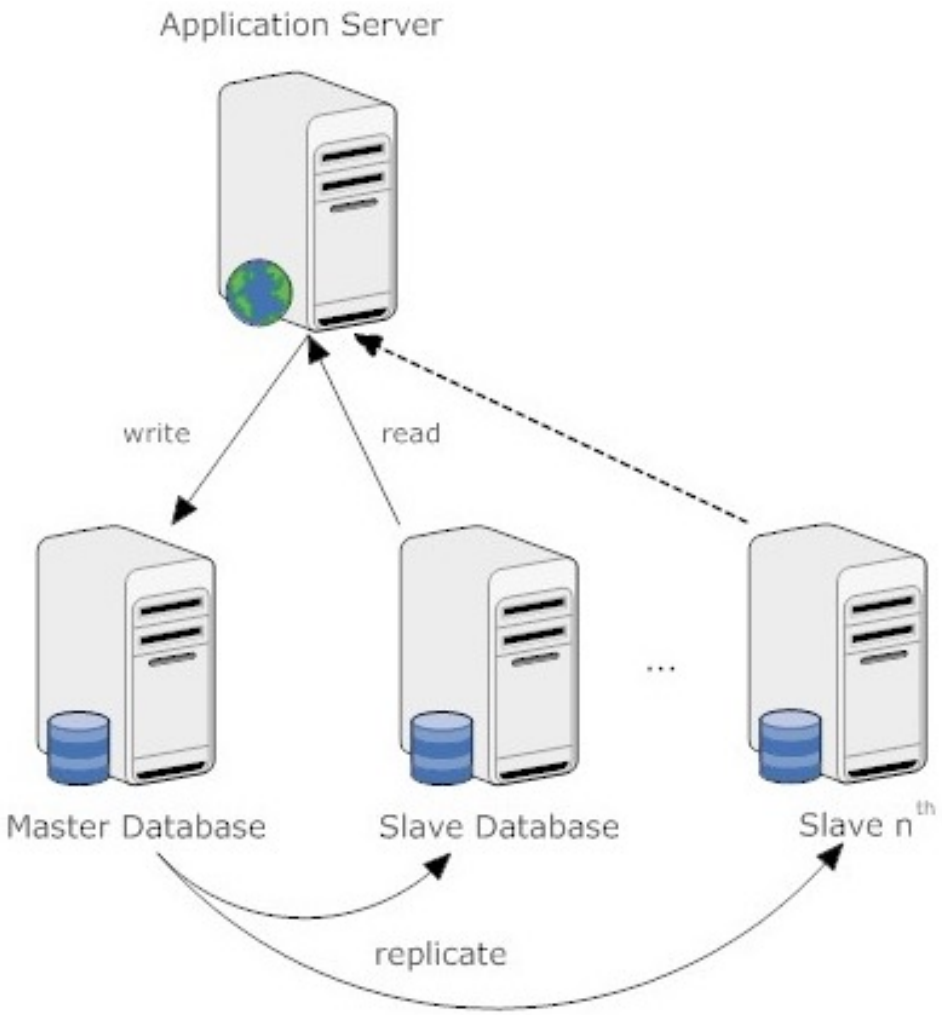
- Formas de fragmentación/replicación: Servidor único
  - No hay distribución
  - Evita complejidades:
    - Gestión transaccional centralizada
    - Integración de los datos
  - NoSQL y servidor único:
    - Puede tener sentido si el modelo de los datos es cercano a NoSQL
      - Bases de datos orientadas a grafos
      - Bases de datos orientada a agregados

3. Modelos de distribución de datos

- Formas de fragmentación/replicación: Maestro/Esclavo
  - Se replican los datos completos en múltiples nodos
  - Un nodo se designa como Maestro:
    - Fuente primaria y acreditada de los datos
    - Responsable de procesar las actualizaciones
    - Responsable de replicar los datos
  - Nodos esclavos:
    - Sólo atienden las operaciones de lectura

3. Modelos de distribución de datos

- Formas de fragmentación/replicación: Maestro/Esclavo



3. Modelos de distribución de datos

- Formas de fragmentación/replicación: Maestro/Esclavo
  - Las lecturas las puede atender cualquier nodo
  - Las escrituras las atiende el maestro:
    - Recibe una petición de escritura
    - Replica la escritura sobre todos los nodos de forma síncrona
      - Retraso de algún esclavo: la escritura no se pierde, pero
      - **Las lecturas del mismo dato no se pueden llevar a cabo si se requiere consistencia estricta**
    - Retraso en el procesamiento de escritura

3. Modelos de distribución de datos

- Formas de fragmentación/replicación: Maestro/Esclavo
  - Escalado:
    - Lecturas:
      - Muy adaptable
      - Si sube tu carga de lecturas aumentas el número de nodos esclavos
      - Distribuyes las lecturas entre todos los nodos esclavos (viejos y nuevos)
    - Escrituras:
      - Problemático
      - Maestro/esclavo es adecuado si el ratio lectura/escritura es alto

3. Modelos de distribución de datos

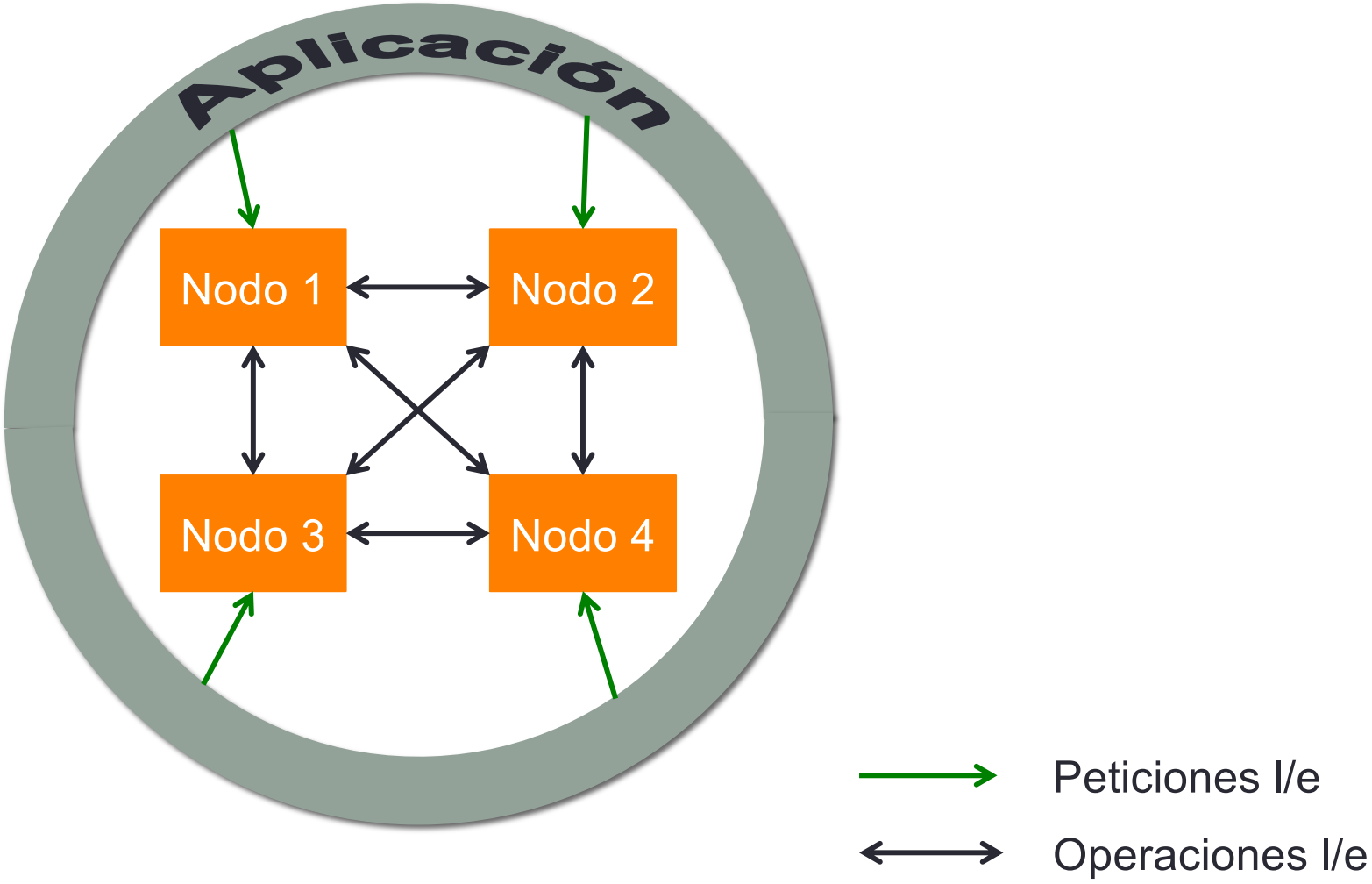
- Formas de fragmentación/replicación: Maestro/Esclavo
  - Resistencia/Disponibilidad:
    - Lecturas:
      - Si el maestro falla, los esclavos se pueden hacer cargo de las lecturas
      - Si un esclavo falla no ocurre nada, sólo hay que eliminarlo de la lista de servidores esclavos para que no se le lancen lecturas
    - Escrituras:
      - Si el maestro falla, las escrituras no se pueden llevar a cabo
      - Un esclavo es elegido como nuevo maestro y el sistema puede seguir atendiendo las escrituras
  - Maestro/esclavos adecuado para sistema que no requieren escalado:
    - Un maestro/un esclavo: backup en caliente y soporte inmediato ante fallos del maestro

3. Modelos de distribución de datos

- Formas de fragmentación/replicación: Entre pares
  - Soluciona el problema de la falta de escalabilidad y resistencia para la escritura del Maestro/esclavo
  - La replicación entre pares resuelve este problema eliminando al maestro
  - Todos los nodos tienen la misma función y soportan las mismas responsabilidades
  - Todos los nodos aceptan lecturas y escrituras
  - La perdida de cualquier nodo no impide el acceso a los datos

3. Modelos de distribución de datos

- Formas de fragmentación/replicación: Entre pares



3. Modelos de distribución de datos

- Formas de fragmentación/replicación: Entre pares
  - El fallo de un nodo no impide el acceso a los datos
  - Se puede mejorar el rendimiento añadiendo nuevos nodos al cluster
  - **Consistencia**: principal problema
    - Escrituras inconsistentes: se puede querer escribir a la vez un mismo dato en varios nodos
    - Lecturas inconsistentes: se puede querer leer un dato en un nodo, dato que está siendo escrito en otro nodo

3. Modelos de distribución de datos

- Formas de fragmentación/replicación: Entre pares
  - **Consistencia**: dos soluciones en los extremos del espectro
    - **Coordinación de replicas**: cuando se escribe sobre un dato en un nodo, el resto de replicas se coordinan para asegurar la consistencia: consistencia similar a la de maestro/esclavo, pero con un fuerte incremento del tráfico de red
      - No se necesario que se coordinen todas, sólo una mayoría
    - **Permitir la escritura inconsistente**: se pueden escribir sobre el mismo dato en dos nodos a la vez
      - En algunos contexto esto no es problemático
      - Total rendimiento, ya que es posible escribir cuando se desee sobre cualquier replica
  - **Consistencia vs Disponibilidad**

3. Modelos de distribución de datos

- Combinación de la fragmentación y la replicación
  - Maestro/esclavo:
    - Se define la estructura maestro/esclavo para cada objeto que se quiere almacenar
    - Existen maestros
    - Un nodo puede hacer a la vez de maestro para un dato y esclavo para otro dato

3. Modelos de distribución de datos

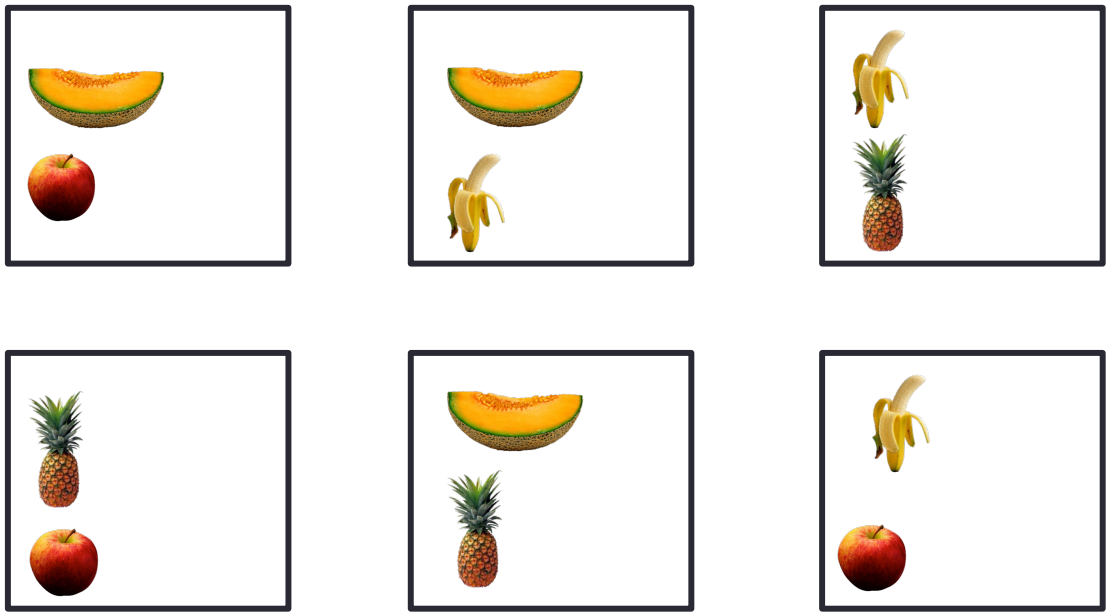
- Combinación de la fragmentación y la replicación
  - Maestro/esclavo:

3. Modelos de distribución de datos

- Combinación de la fragmentación y la replicación
  - Entre pares:
    - Para sistemas orientados a familias de columnas
    - Se pueden tener decenas o centenares de nodos en un cluster con fragmentación de datos
    - Un factor de replica común suele ser 3
    - Si un nodo falla, las particiones de ese nodo son generadas en otros nodos

3. Modelos de distribución de datos

- Combinación de la fragmentación y la replicación
  - Entre pares:



Índice

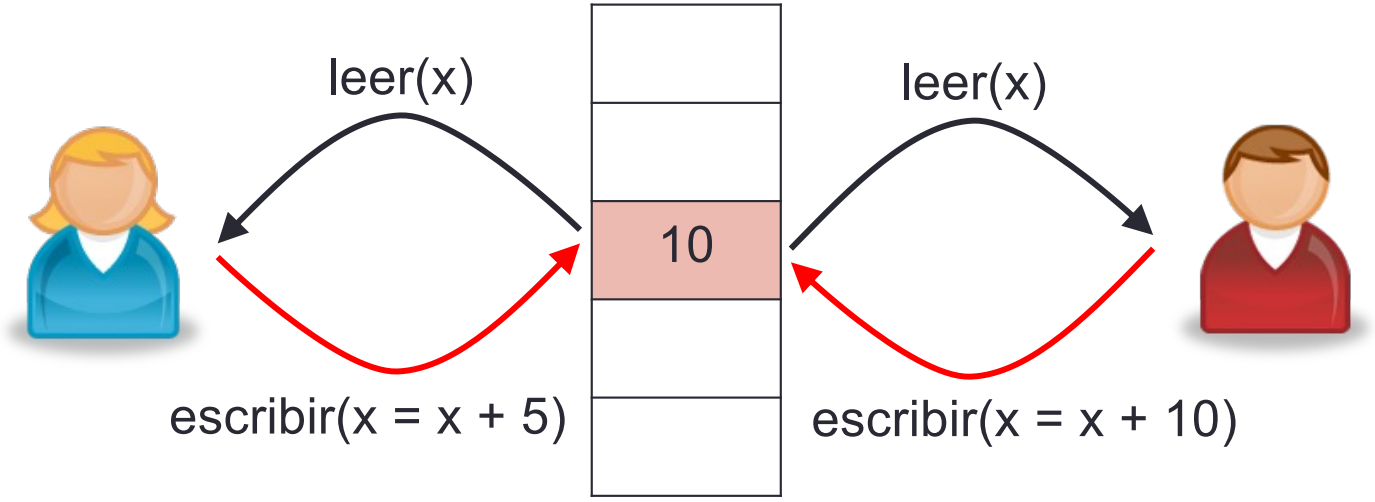
1. Introducción a la tecnología NoSQL
2. Modelos de datos
   2.1. Modelos de datos agregados
   2.2. Otros modelos y aspectos de modelado
3. Modelos de distribución de datos
4. Consistencia
5. Marcados de versión

4. Consistencia

- Consistencia en BDR
  - **Consistencia fuerte**: gestión transaccional ACID
  - Definición de consistencia
    - Consistencia semántica
    - Consistencia / Concurrencia
      - Consistencia de actualización
      - Consistencia de lectura
    - Consistencia / Persistencia
- Sistemas no SQL
  - Relajar la consistencia para conseguir otros objetivos: disponibilidad, rendimiento, …

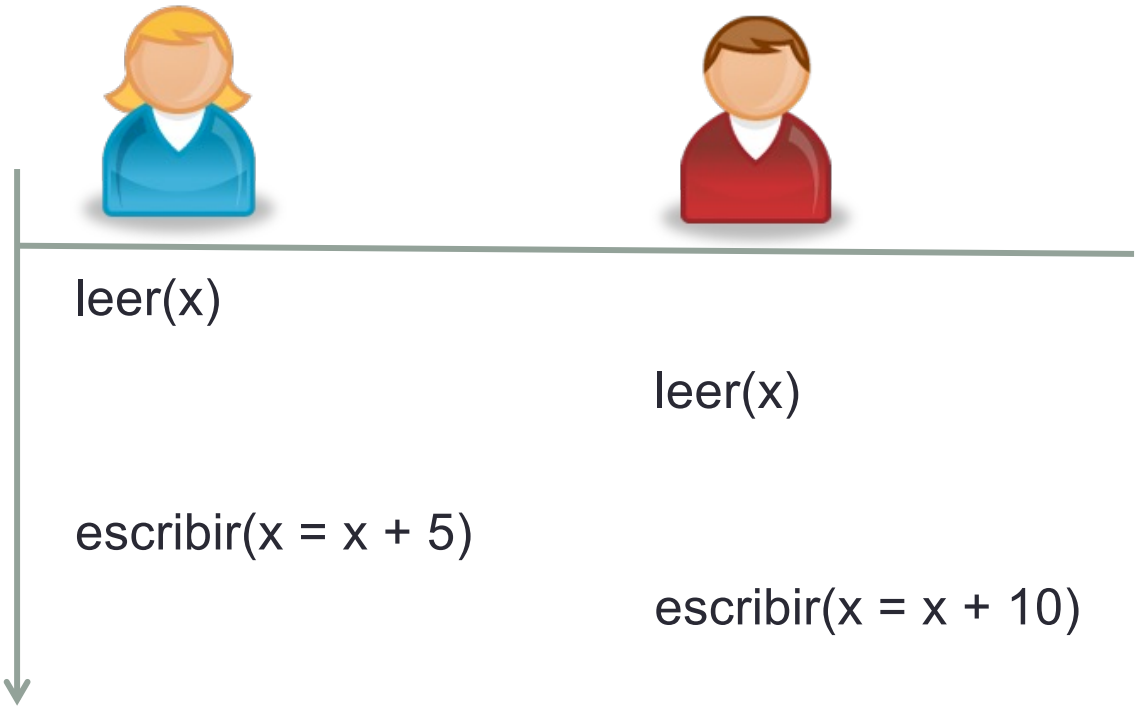
4. Consistencia

- Consistencia de actualización
  - Concurrencia: problema de pérdida de actualización (conflicto escritura-escritura)
    - Dos usuarios actualizan el mismo data a la vez



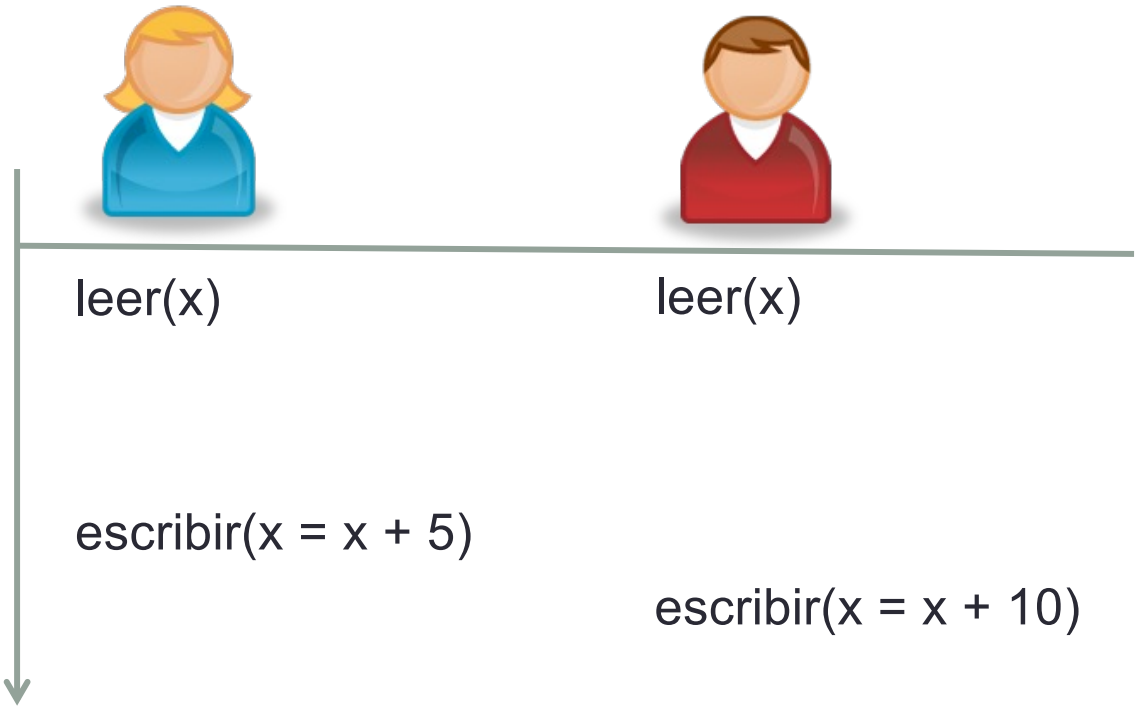
4. Consistencia

- Consistencia de actualización
  - Concurrencia: problema de pérdida de actualización (conflicto escritura-escritura)
    - Dos usuarios actualizan el mismo dato a la vez
  - Servidor: serializa las operaciones (escrituras)



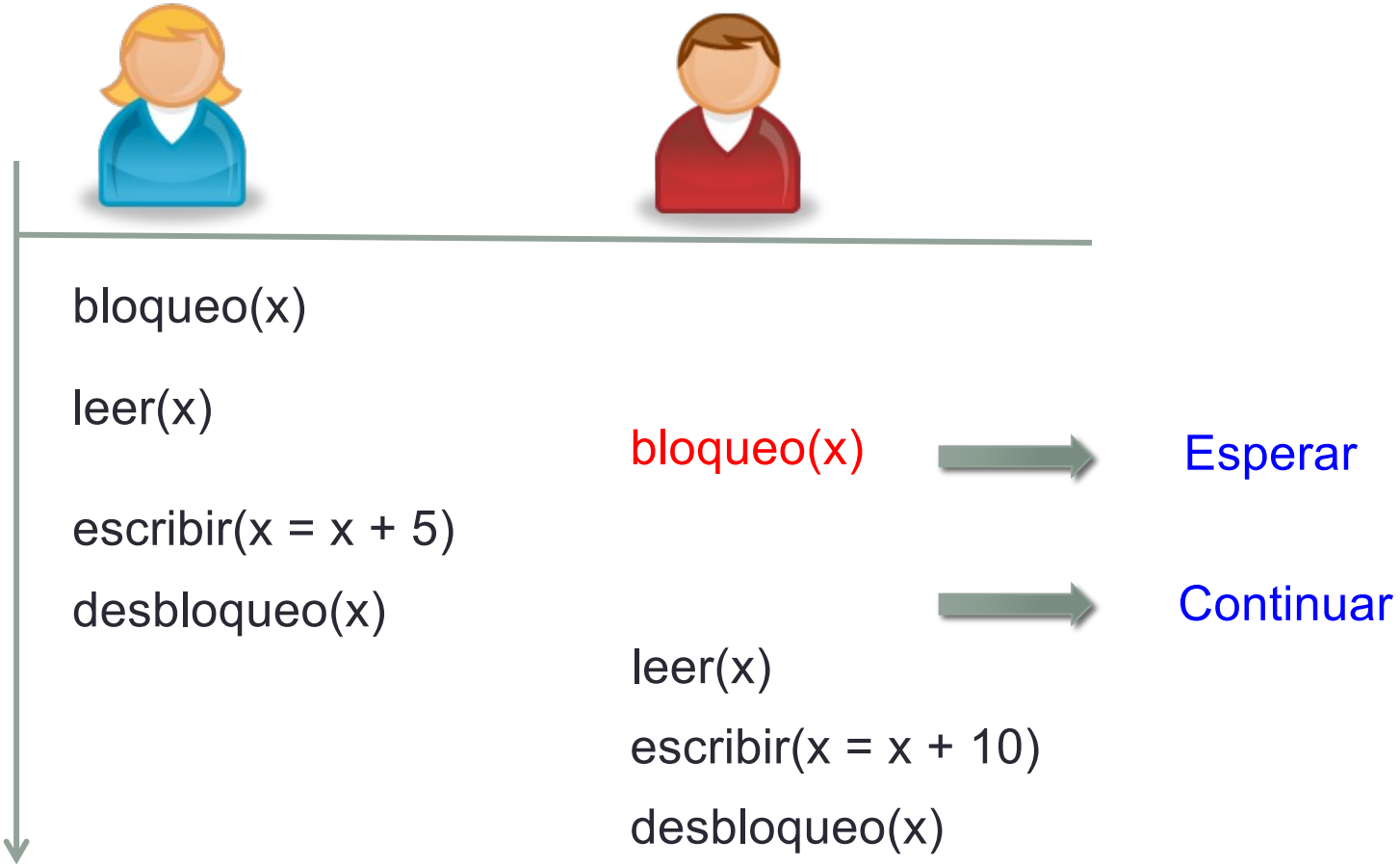
4. Consistencia

- Consistencia de actualización
  - Concurrencia: problema de pérdida de actualización (conflicto escritura-escritura)
    - Dos usuarios actualizan el mismo dato a la vez
  - Servidor: serializa las operaciones (escrituras)



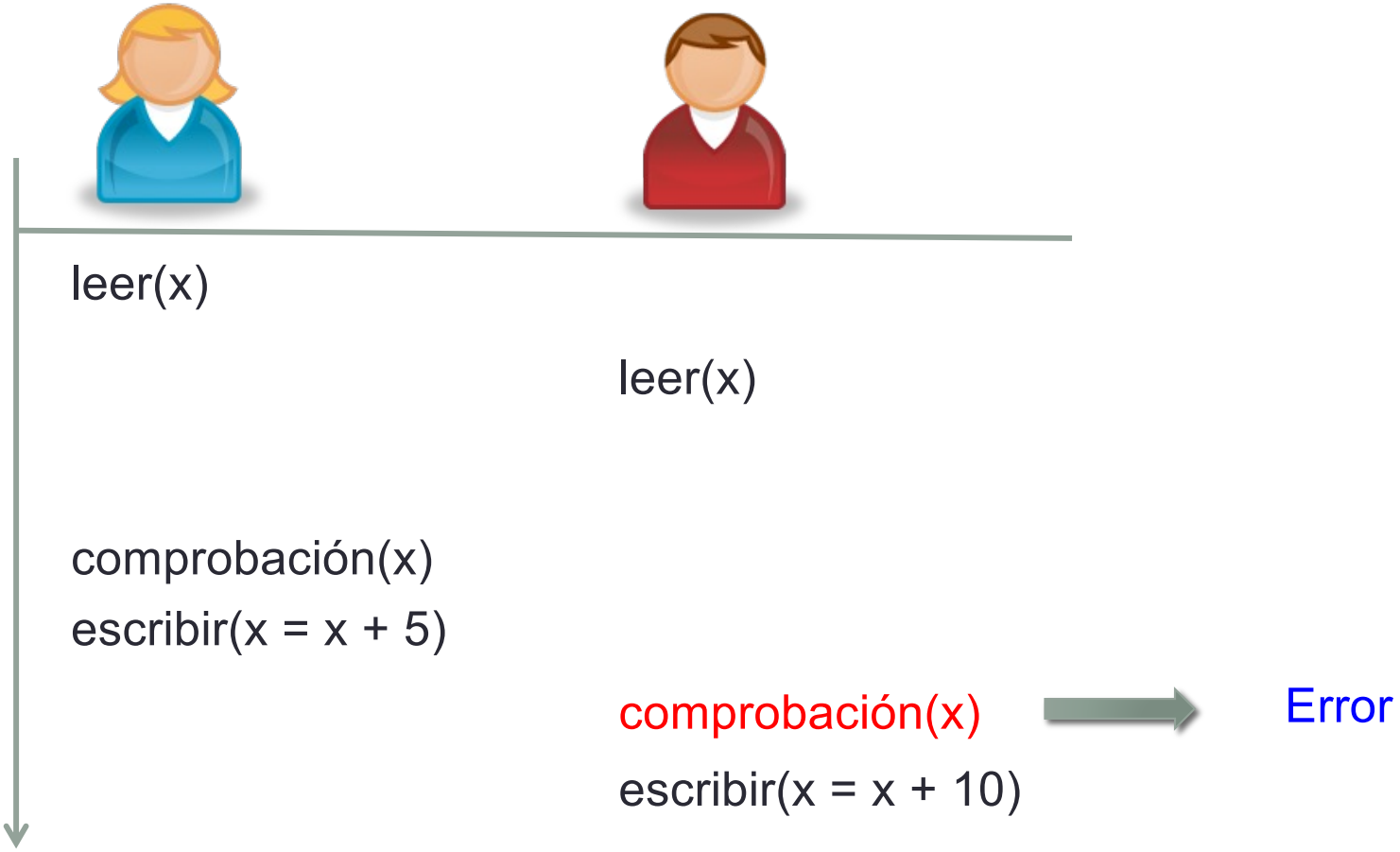
4. Consistencia

- Consistencia de actualización
  - Solución pesimista: se evita el problema
    - Uso de bloqueos



4. Consistencia

- Consistencia de actualización
  - Solución optimista: cuando se da el conflicto, se resuelve
    - Actualización condicional



4. Consistencia

- Consistencia de actualización
  - Concurrencia: problema de pérdida de actualización (conflicto escritura-escritura)
    - Solución pesimista y optimista se basan en la serialización de las operaciones
    - En un servidor es sencillo
    - Con múltiples nodos la serialización es más compleja: consistencia secuencial (asegurarse de que todos los nodos aplican las operaciones en el mismo orden)

4. Consistencia

- Consistencia de actualización
  - Solución optimista sin serialización:
    - Se dejan realizar las escrituras y se registra que están en conflicto
    - Sistemas de control de versiones distribuidas
    - Es necesario fundir las dos actualizaciones: solución dependiente del dominio
      - Preguntar al usuario
      - Deducirlo de alguna manera

4. Consistencia

- Consistencia de actualización
  - Pesimistas: evitan errores, pero degradan la capacidad de respuesta del sistema (interbloqueos, etc.)
  - Optimistas: admiten la posibilidad de errores y los tratan, el sistema siempre responde
    - Balance entre ambas soluciones
  - Replicación entre múltiples nodos acentúa la posibilidad de los conflictos escritura-escritura
    - Distintos nodos tienen copias del mismo dato que pueden ser actualizadas independientemente
    - Nodo único hace mucho más sencillo mantener la consistencia

4. Consistencia

- Consistencia de lectura
  - Las lecturas de datos consistentes pueden obtener información inconsistente: lectura inconsistente o conflicto lectura-escritura

4. Consistencia

- Consistencia de lectura
  - Lectura inconsistente o conflicto lectura-escritura

Línea

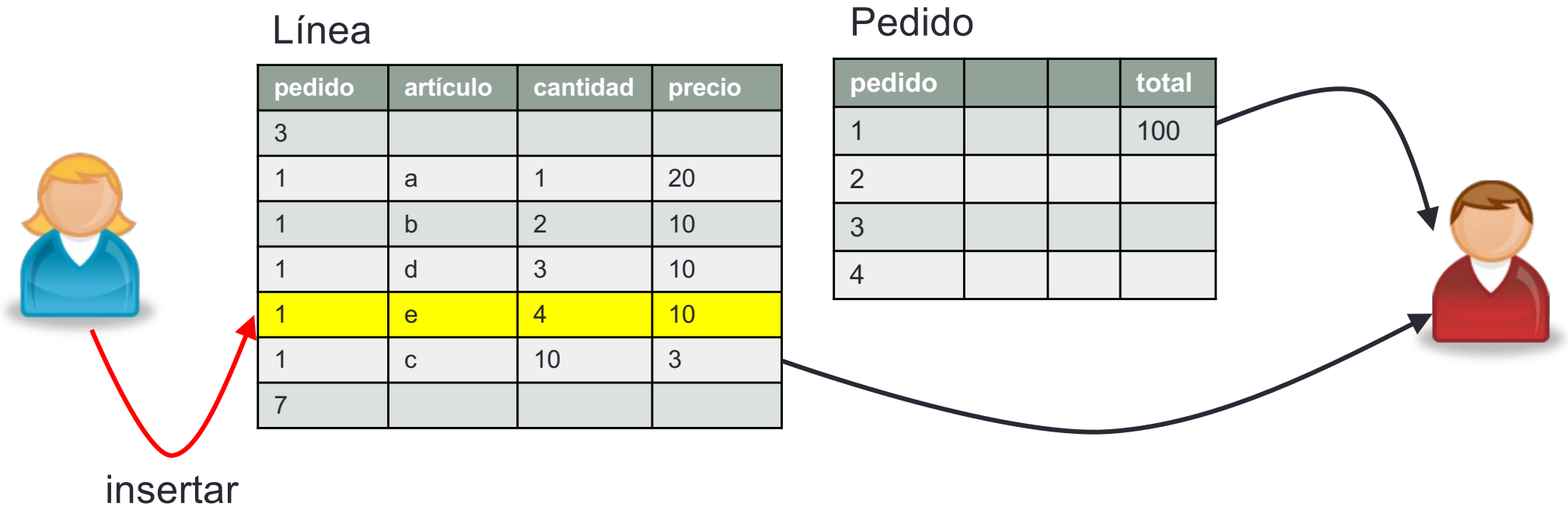
pedido	artículo	cantidad	precio
3			
1	a	1	20
1	b	2	10
1	d	3	10
1	c	10	3
7			

Pedido

pedido			total
1			100
2			
3			
4			

4. Consistencia

- Consistencia de lectura
  - Lectura inconsistente o conflicto lectura-escritura



4. Consistencia

- Consistencia de lectura
  - Lectura inconsistente o conflicto lectura-escritura

insertar

Línea

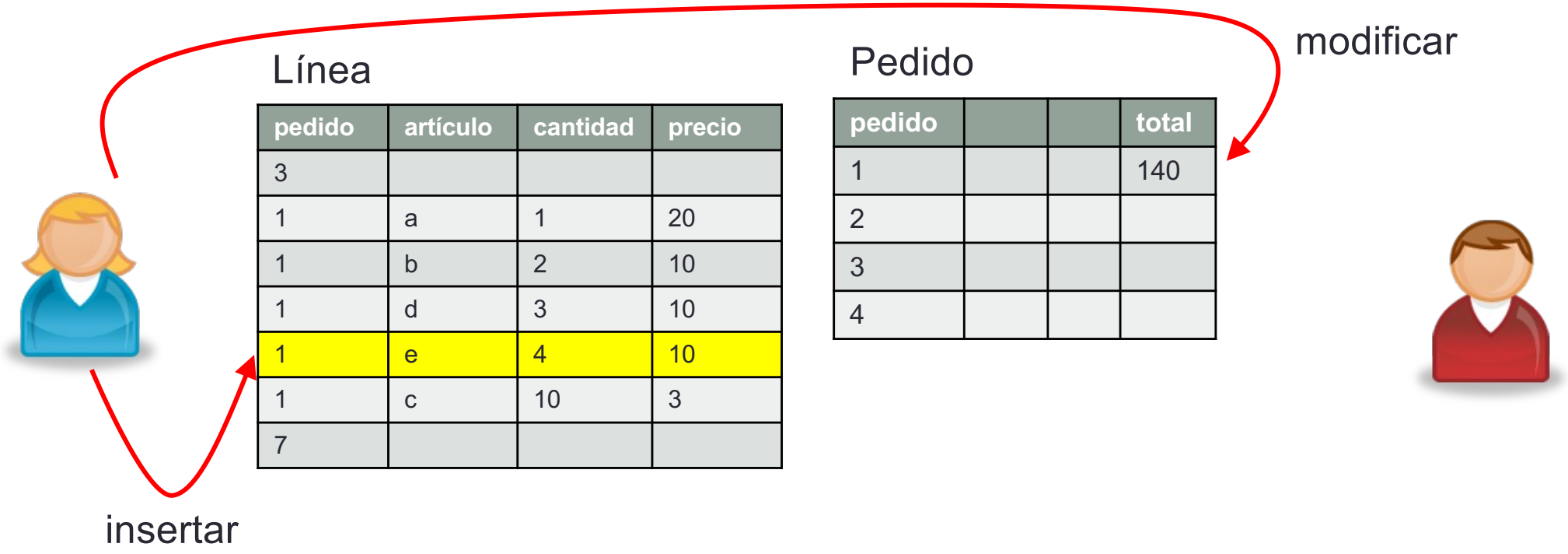
pedido	artículo	cantidad	precio
3			
1	a	1	20
1	b	2	10
1	d	3	10
1	e	4	10
1	c	10	3
7			

Pedido

pedido			total
1			100
2			
3			
4			

4. Consistencia

- Consistencia de lectura
  - Lectura inconsistente o conflicto lectura-escritura

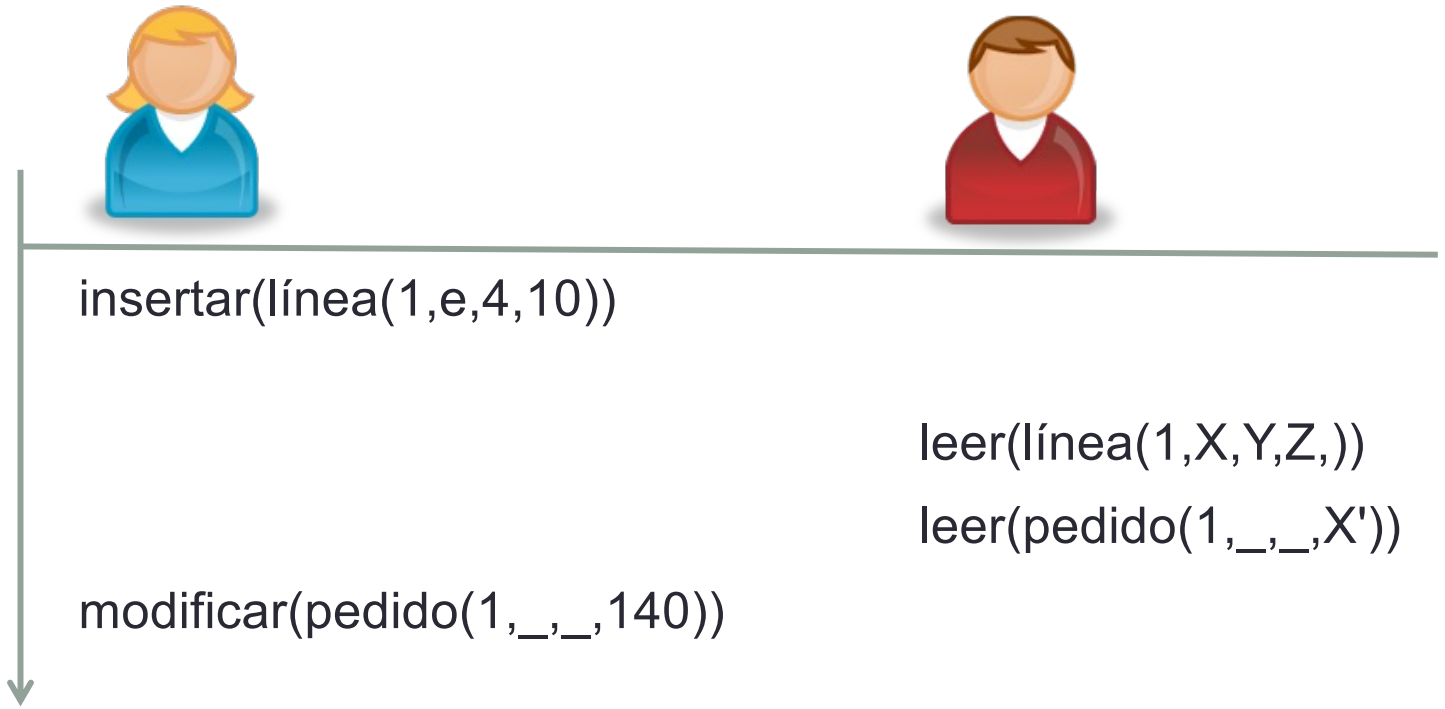


pedido	artículo	cantidad	precio
3			
1	a	1	20
1	b	2	10
1	d	3	10
1	e	4	10
1	c	10	3
7			

pedido			total
1			140
2			
3			
4			

4. Consistencia

- Consistencia de lectura
  - Lectura inconsistente o conflicto lectura-escritura



4. Consistencia

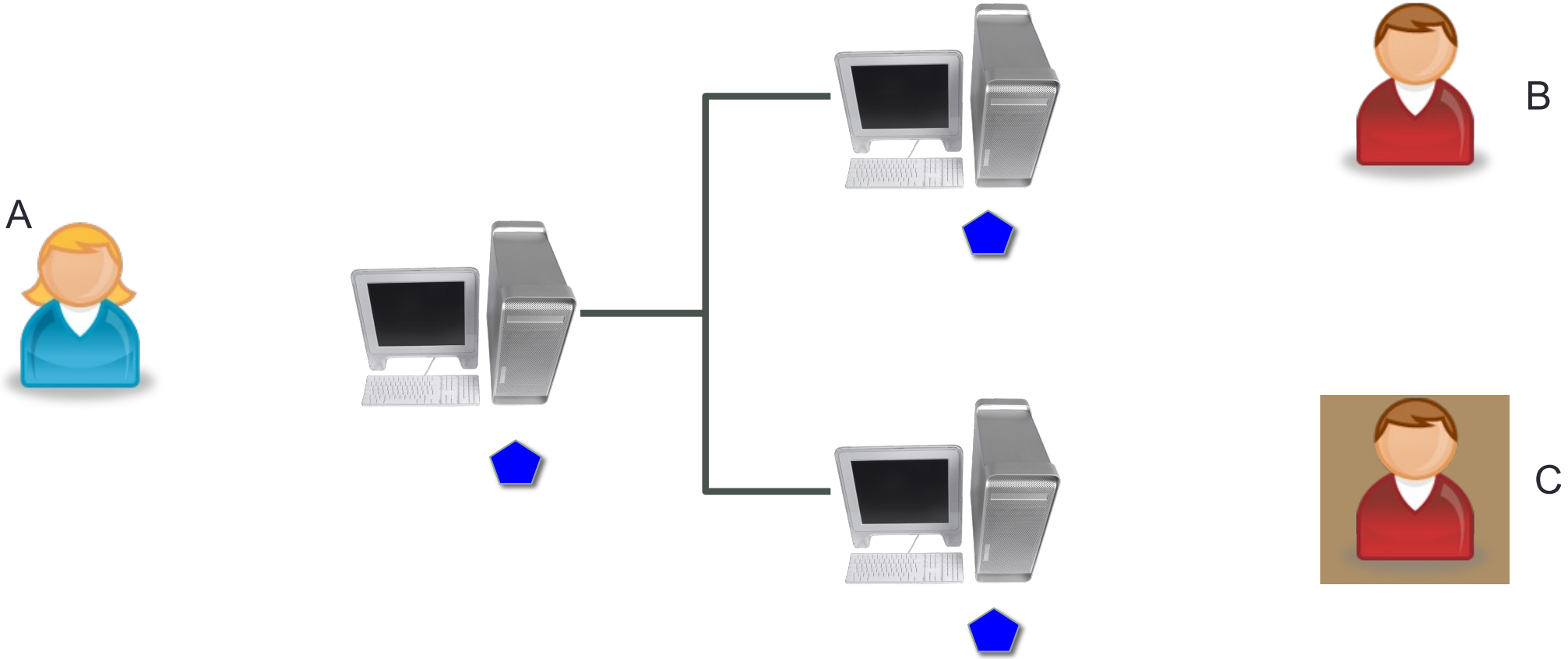
- Consistencia de lectura
  - **Consistencia lógica**
  - BDR evitan este tipo de conflictos imponiendo la noción de transacción para el acceso a la base de datos
    - El usuario rojo leería los datos antes o después de que el azul los actualice (pero nunca entre medias)

4. Consistencia

- Consistencia de lectura
  - NoSQL: **¿no soportan transacciones?**
    - Orientadas a grafos soportan ACID
    - Orientadas a datos agregados: no soportan transacciones
      - Actualizaciones atómicas sobre un único agregado: si el pedido agrega las líneas no hay problema
      - Actualizaciones sobre múltiples agregados:
        - Deja un cierto tiempo en el que se pueden producir lecturas inconsistentes: **ventana de inconsistencia**
        - Algunos sistemas conviven con ello (la ventana de inconsistencia de SimpleDB de Amazon es < 1 seg)

4. Consistencia

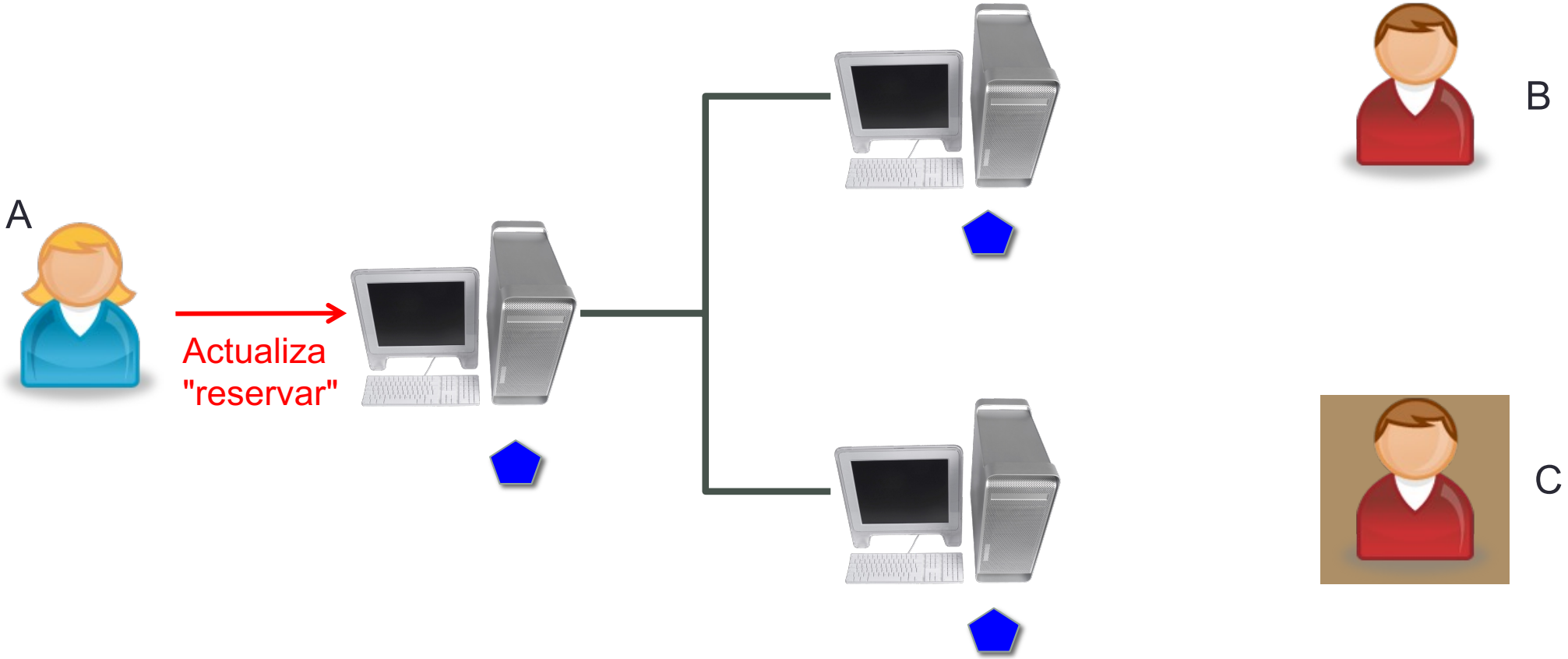
- Consistencia de lectura
  - Replicación: incrementa el problema



Última habitación disponible

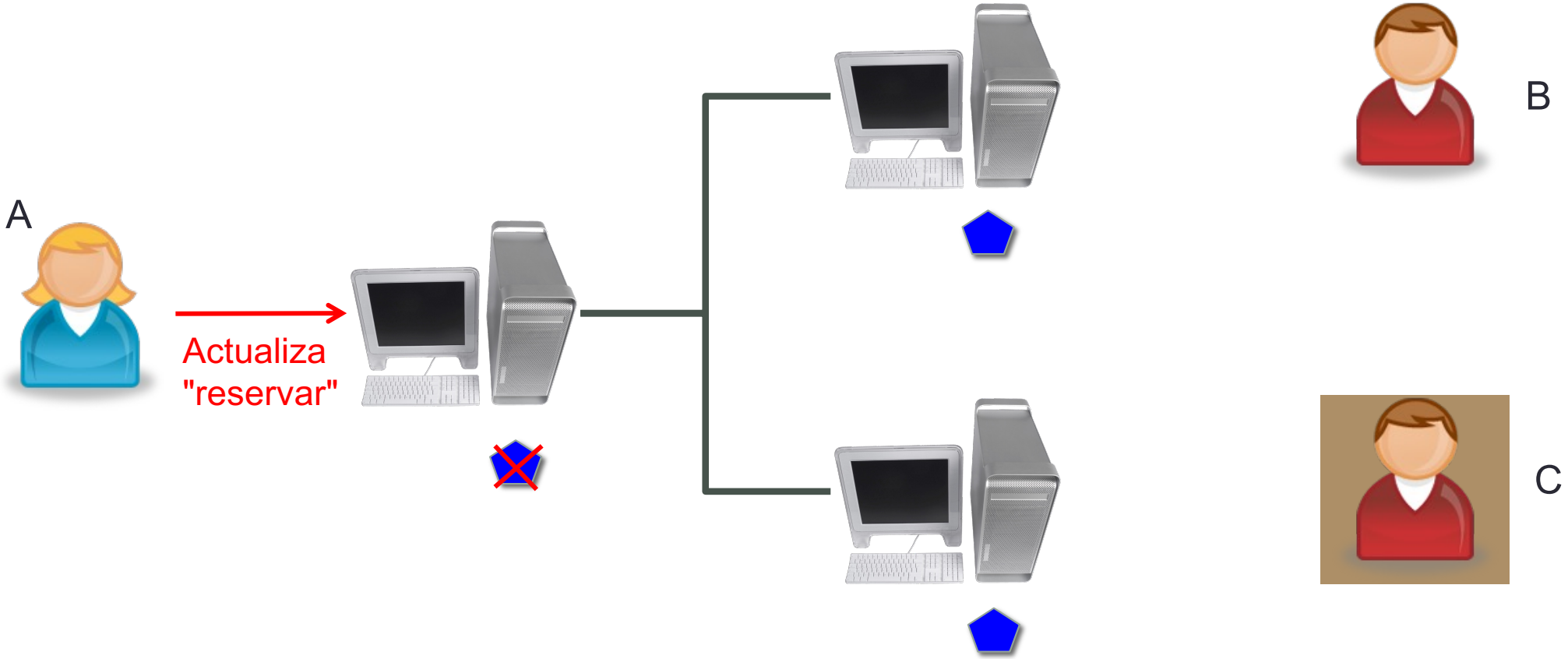
4. Consistencia

- Consistencia de lectura
  - Replicación: incrementa el problema



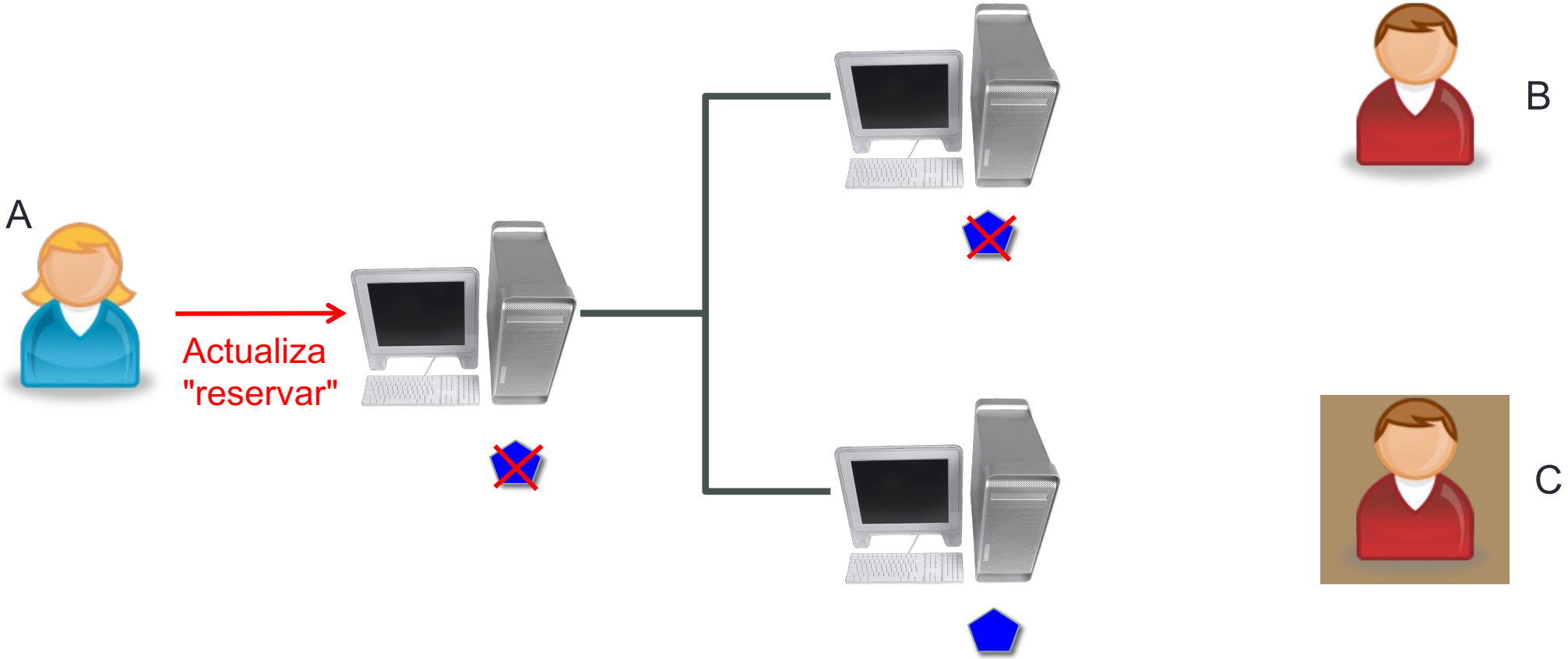
4. Consistencia

- Consistencia de lectura
  - Replicación: incrementa el problema



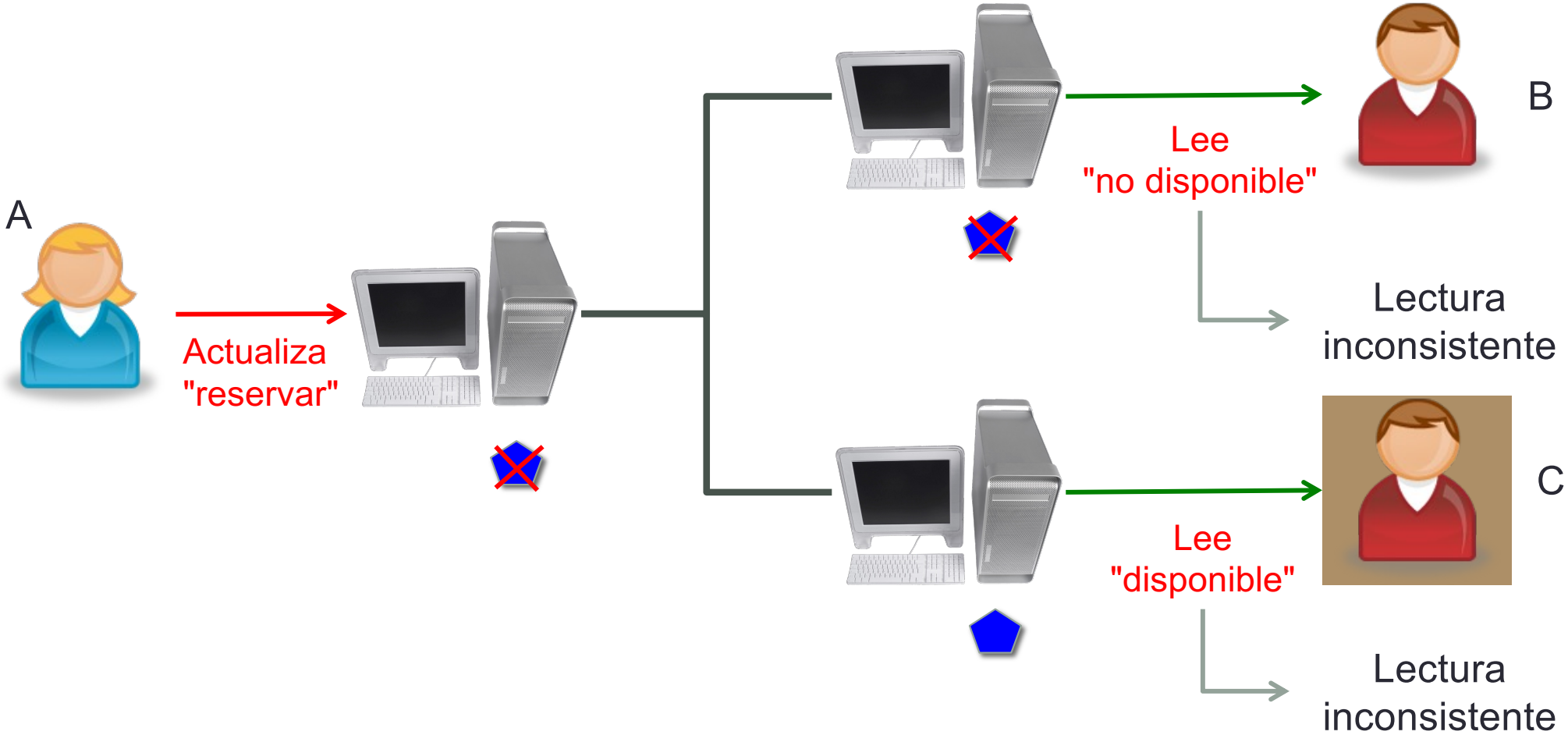
4. Consistencia

- Consistencia de lectura
  - Replicación: incrementa el problema



4. Consistencia

- Consistencia de lectura
  - Replicación: incrementa el problema



4. Consistencia

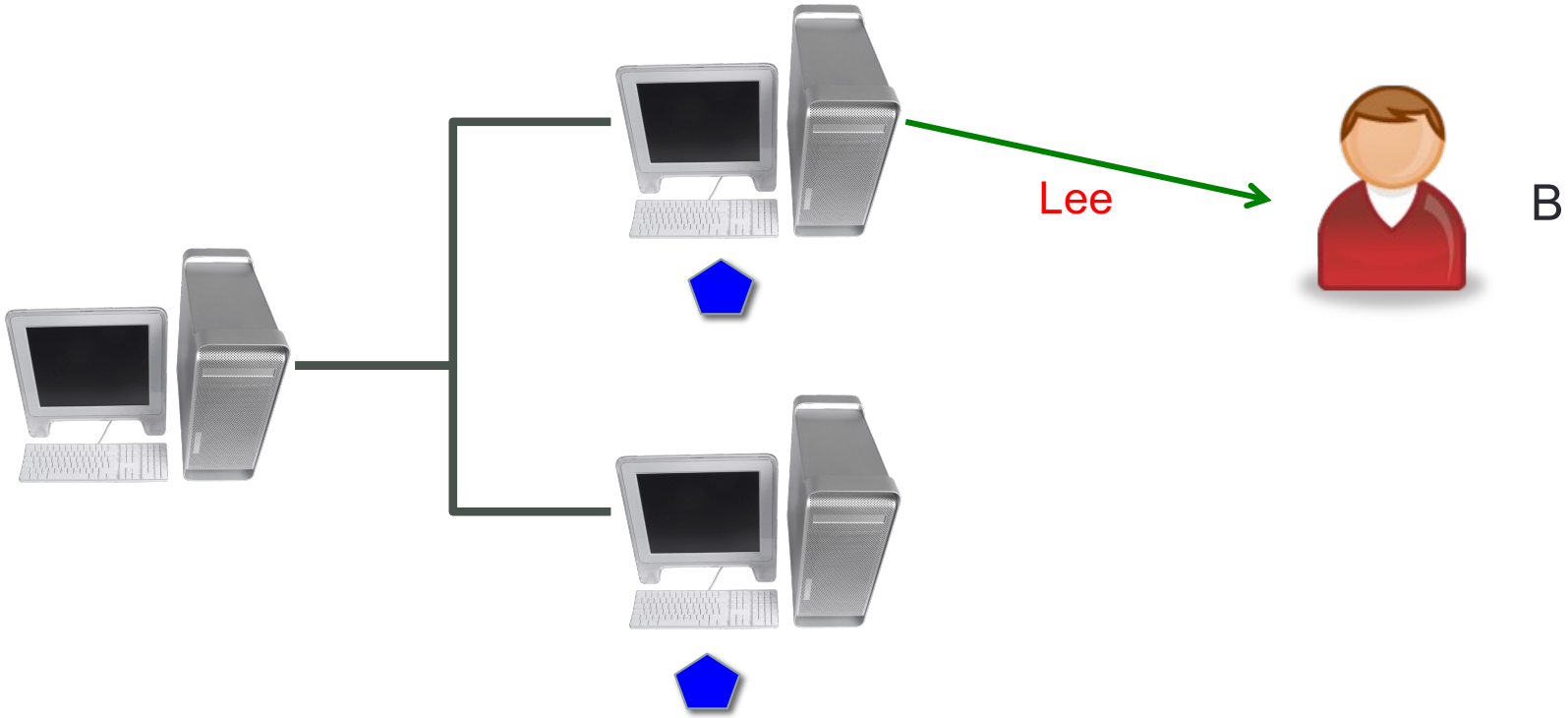
- Consistencia de lectura
  - Replicación: **consistencia de replicación**
    - Cada dato tiene el mismo valor en todas sus réplicas
  - **Eventualmente consistente**:
    - En un cierto instante hay inconsistencia de replicación
    - Si no hay mas actualizaciones, eventualmente todos los nodos actualizarán el dato con el mismo valor
    - Los datos que no están al día se denominan **obsoletos**

4. Consistencia

- Consistencia de lectura
  - Replicación: **consistencia de replicación**
    - Independiente de la consistencia lógica
    - Incrementa la ventana de inconsistencia
      - Inconsistencia lógica: lapso entre dos actualizaciones en un servidor
      - Inconsistencia de replicación: actualización de todas las réplicas de un dato en todos los nodos (Red → mucho mayor)

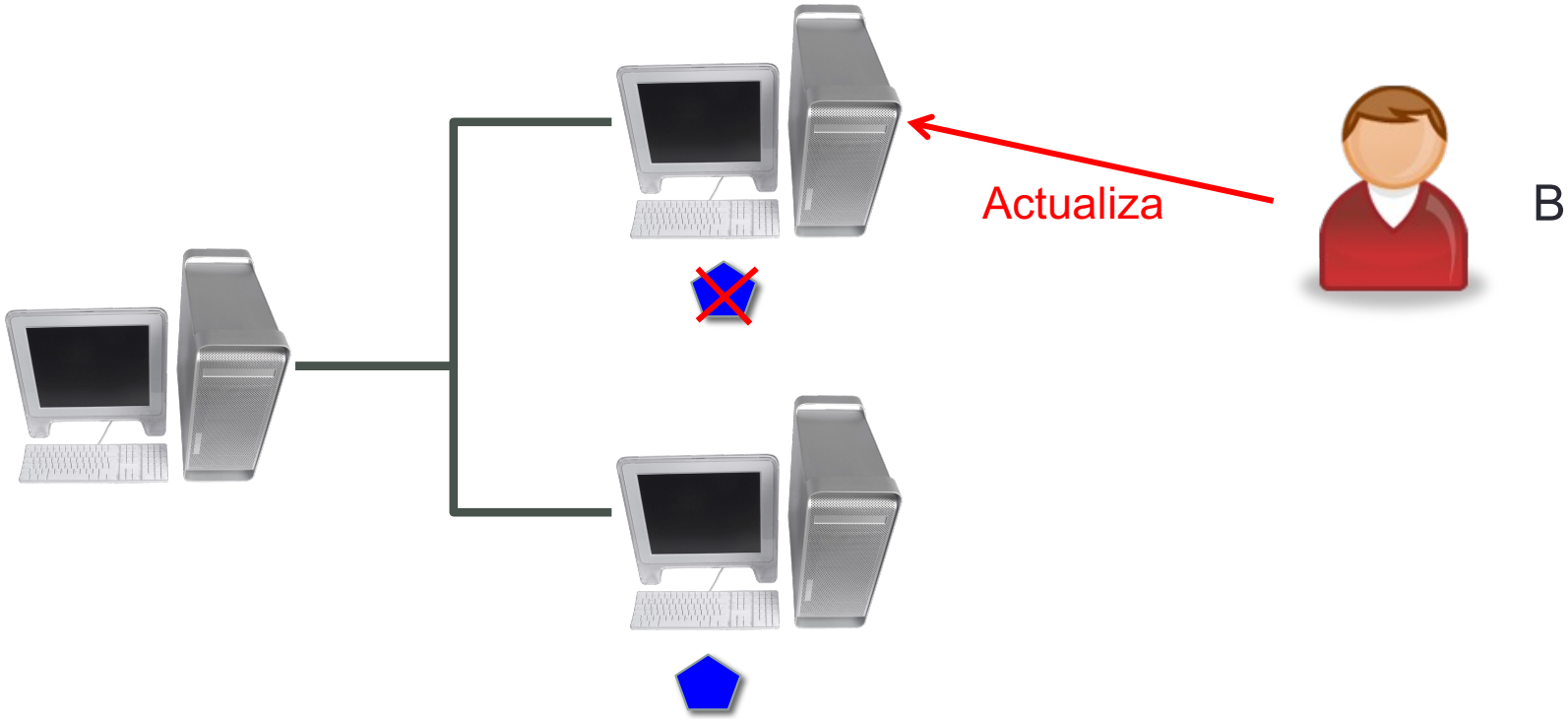
4. Consistencia

- Consistencia de lectura
  - Replicación: **consistencia de replicación**
    - Las garantías de la consistencia no suelen ser globales para una aplicación, depende de cada petición individual
    - **Consistencia de leer-tus-escrituras: consistencia de sesión**



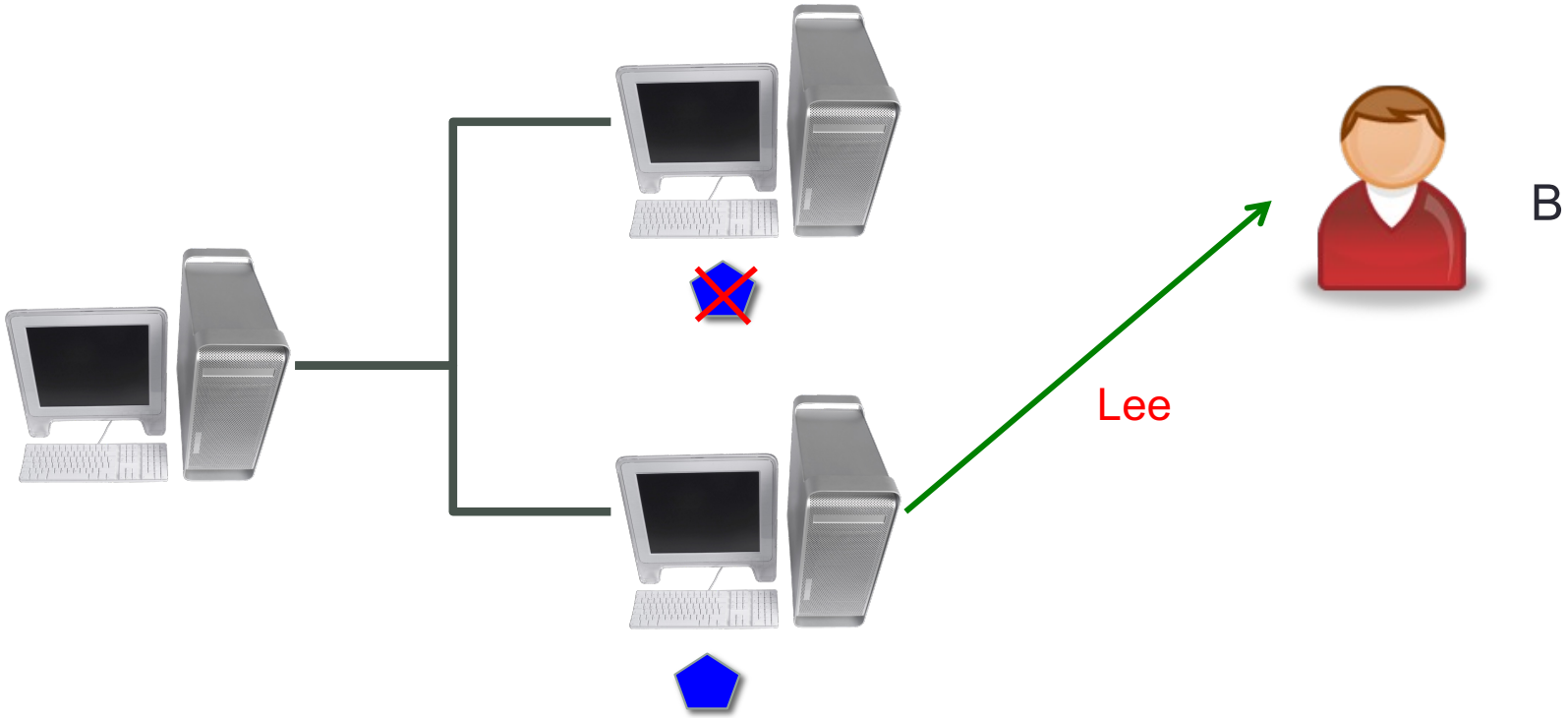
4. Consistencia

- Consistencia de lectura
  - Replicación: **consistencia de replicación**
    - Las garantías de la consistencia no suelen ser globales para una aplicación, depende de cada petición individual
    - **Consistencia de leer-tus-escrituras: consistencia de sesión**



4. Consistencia

- Consistencia de lectura
  - Replicación: **consistencia de replicación**
    - Las garantías de las consistencia no suelen ser globales para una aplicación, depende de cada petición individual
    - **Consistencia de leer-tus-escrituras: consistencia de sesión**



4. Consistencia

- Consistencia de lectura
  - Consistencia de sesión
    - Sesión adherida:
      - Una sesión se ejecuta en un nodo (**afinidad de sesión**): todas las instrucciones de acceso se ejecutan en un solo nodo
      - Más inflexible para el balanceo de carga
    - Marcado de versiones:
      - Cada interacción con el almacén de datos, se incluye la última marca de versión
      - El nodo servidor debe asegurar que tiene las actualizaciones que incluyen esa marca de versión antes de responder al requerimiento

4. Consistencia

- Consistencia de lectura
  - Consistencia de sesión: replicación maestro/esclavo
    - No hay consistencia de sesión:
      - Lecturas en el esclavo para incrementar el rendimiento
      - Escrituras en el maestro
    - Soluciones:
      - Esclavo responsable de enviar las escrituras al maestro y mantener la consistencia de sesión
      - Trasladar la sesión al maestro y que se haga cargo temporalmente de las lecturas, hasta que se replique el valor

4. Consistencia

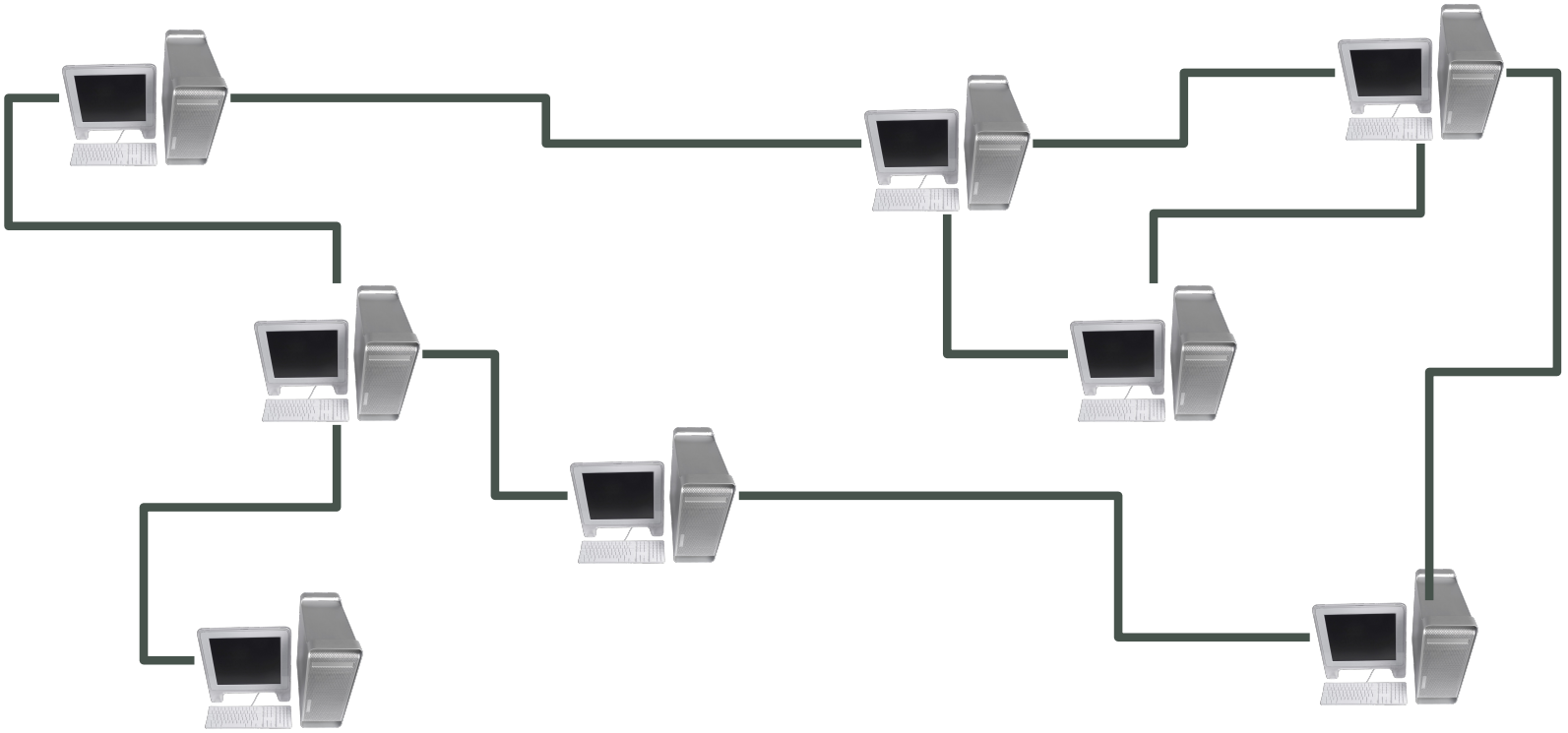
- Relajando la consistencia
  - Rebajar las exigencias en la consistencia total
  - Consigue cubrir otros objetivos: rendimiento, disponibilidad, etc.
  - Distintos dominios tienen diferentes niveles de tolerancia para la inconsistencia
  - BDR:
    - Distintos niveles de aislamiento: LNC, **LC**, LR y SER
    - Cada aplicación puede tolerar un nivel más o menos grave de inconsistencia
    - MySQL: funcionó durante mucho tiempo con éxito sin proporcionar gestión transaccional
    - Mejoras en el rendimiento

4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Eric Brewer en 2000
  - "Un sistema computacional distribuido no puede ofrecer simultáneamente consistencia, disponibilidad y tolerancia a la partición"
  - Como máximo se pueden asegurar dos
  - Definiciones:
    - Consistencia: consistencia de actualización, consistencia de lectura, …
    - Disponibilidad (*Availability*): si se tiene acceso a un nodo en el cluster, éste debe poder responder a peticiones de lectura y escritura
    - Tolerancia a la partición (*Partition tolerance*): el cluster debe poder sobrevivir a divisiones de su red de comunicación que lo fragmenten en partes incomunicadas entre sí

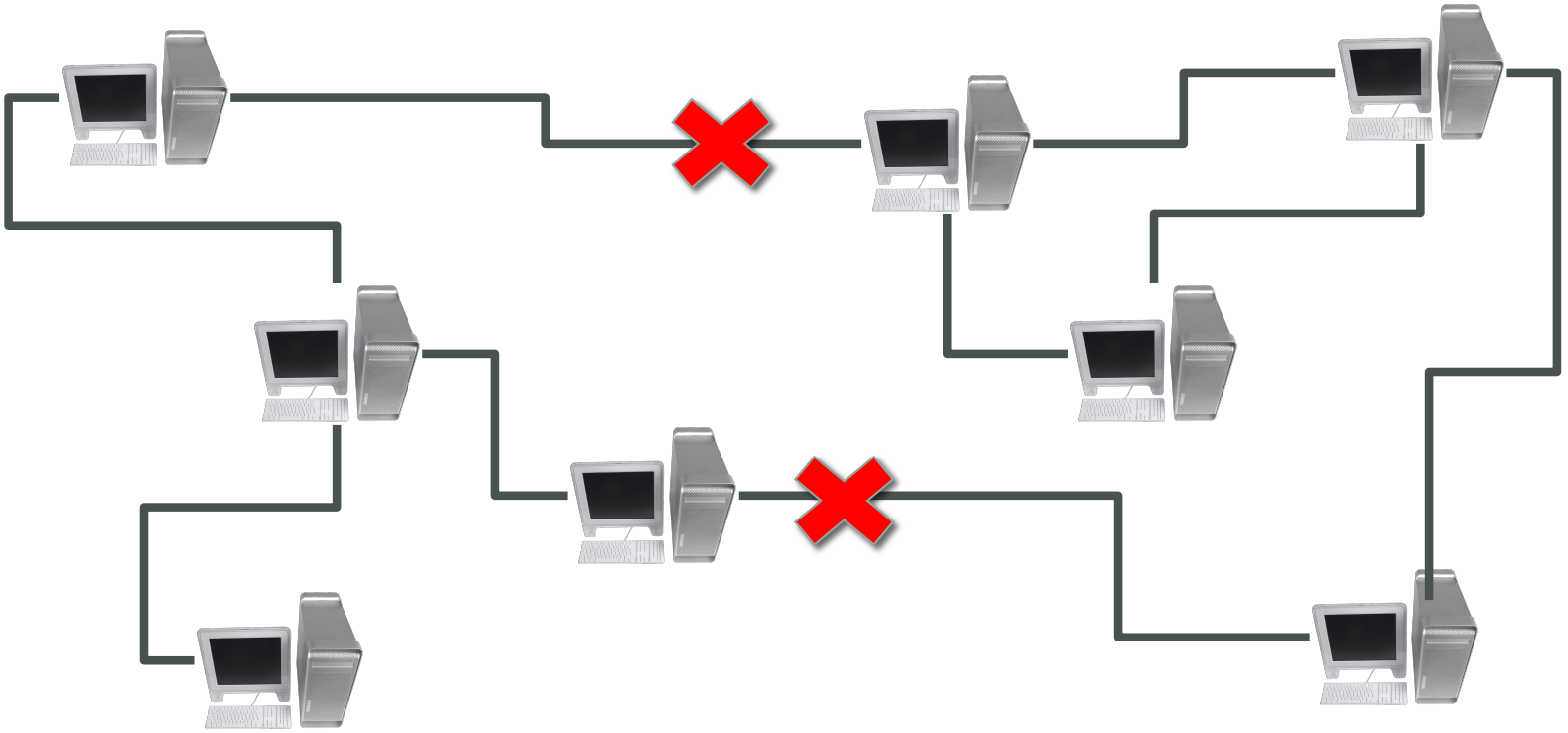
4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Partición de un cluster



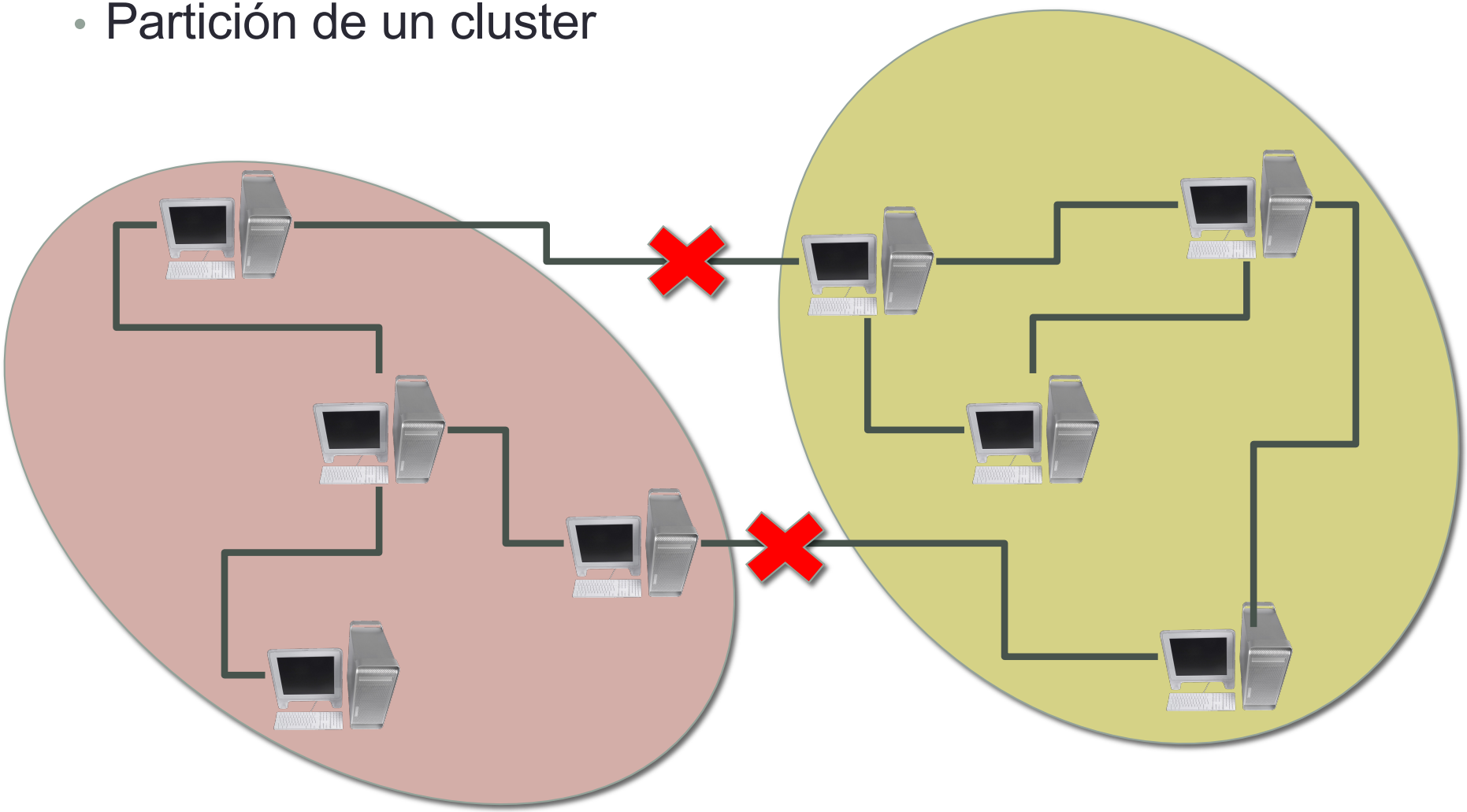
4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Partición de un cluster



4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Partición de un cluster



4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Servidor único
    - Sistema CA: proporciona consistencia y disponibilidad
    - Sistemas relacionales

4. Consistencia

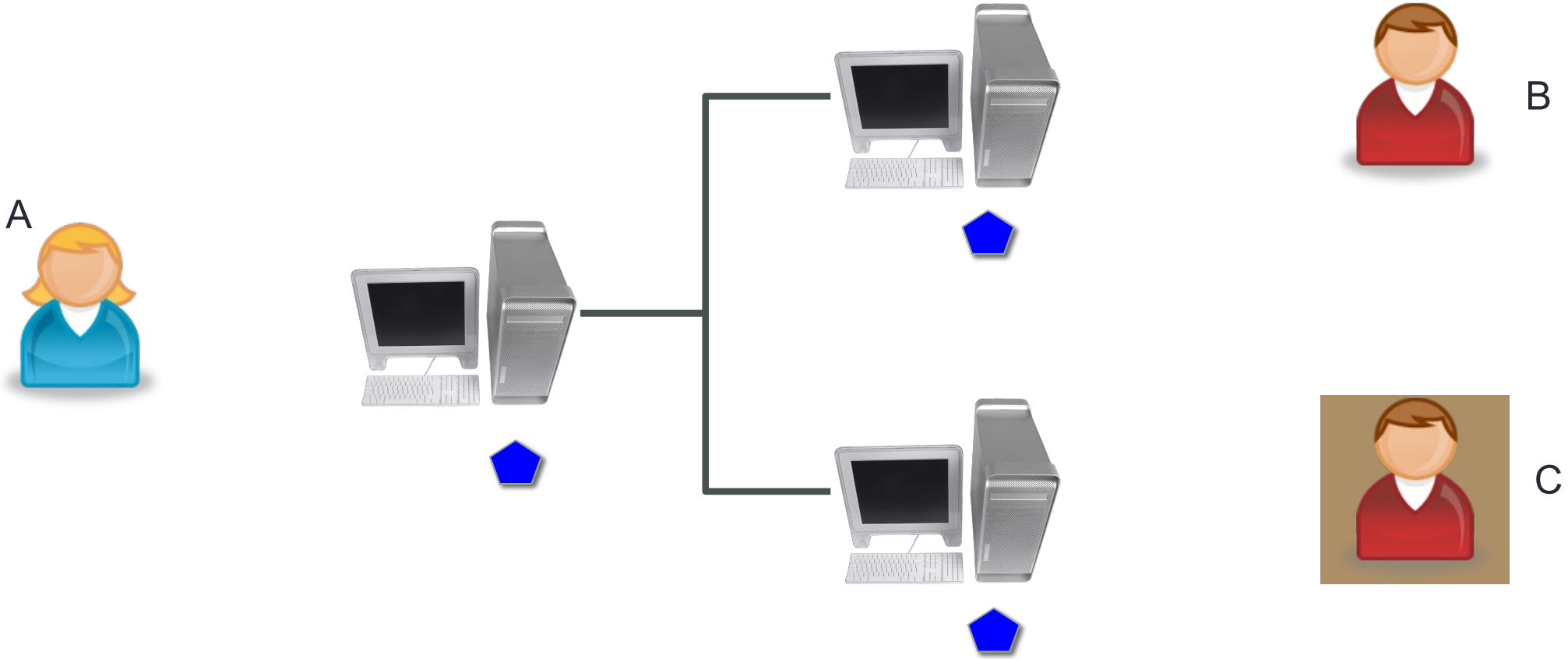
- Relajando la consistencia: Teorema CAP
  - Cluster: ¿sistema CA?
    - Teóricamente sí.
    - Se produce una partición:
      - Todos los nodos tienen que pasar a **no disponibles**
      - Ningún cliente puede acceder a ningún nodo
    - Disponibilidad: cualquier petición recibida por un nodo (que está activo) debe ser respondida.
    - Detectar que se produce una partición y hacer caer todos los nodos inmediatamente (complejo)

4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster:
    - **Sistema tolerante a particiones de la red** → compromiso entre consistencia y disponibilidad
    - No es una elección binaria
      - Un poco menos de consistencia a cambio de algo de disponibilidad
      - Poca consistencia y bastante disponibilidad
    - Todo o nada no es una buena elección
    - Elección de compromiso adecuada a las necesidades de la organización

4. Consistencia

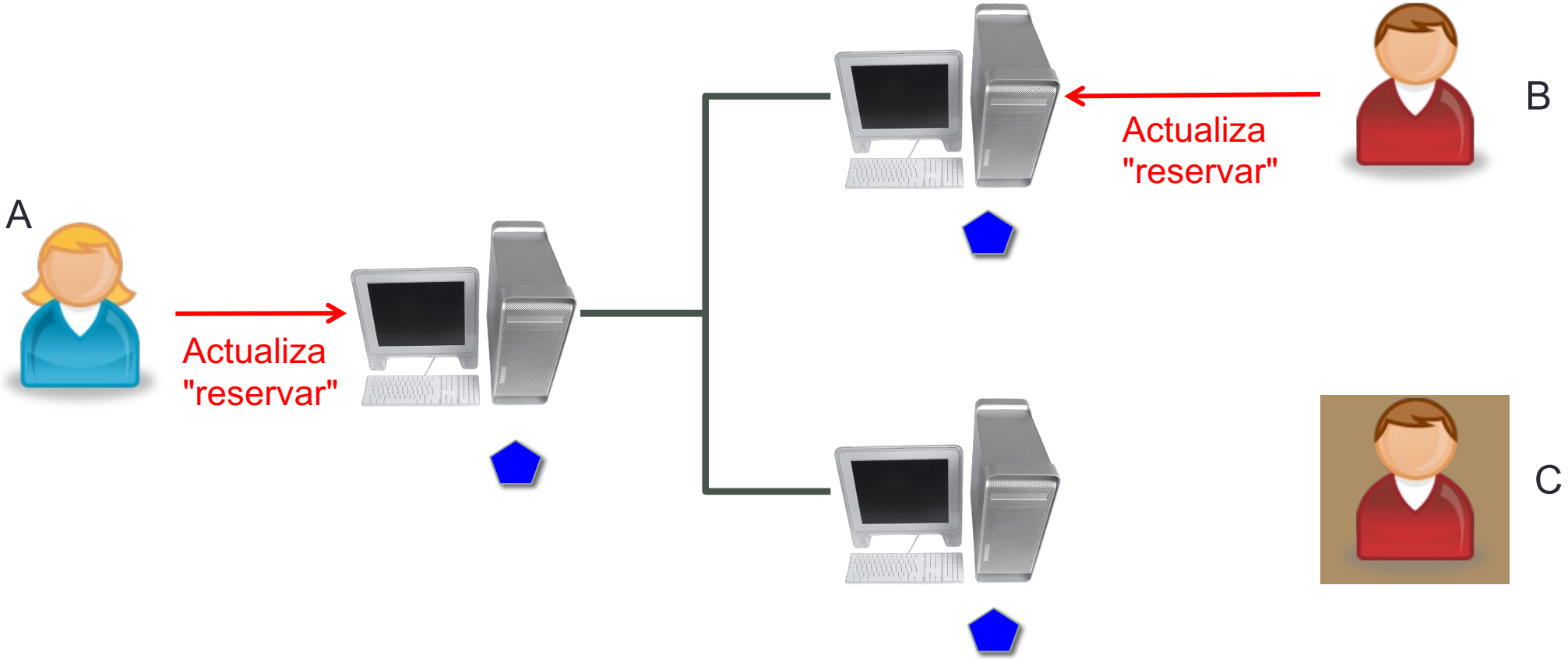
- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación entre pares



Última habitación disponible

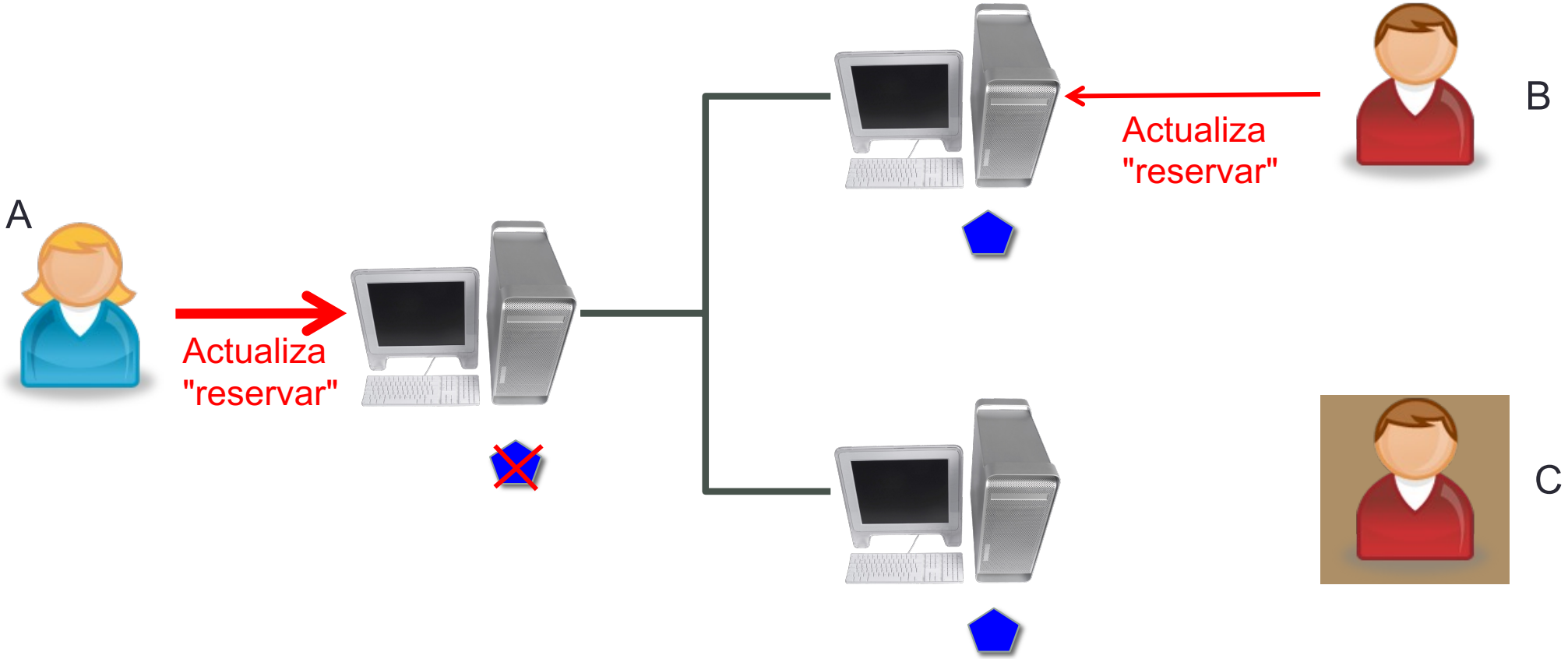
4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación entre pares



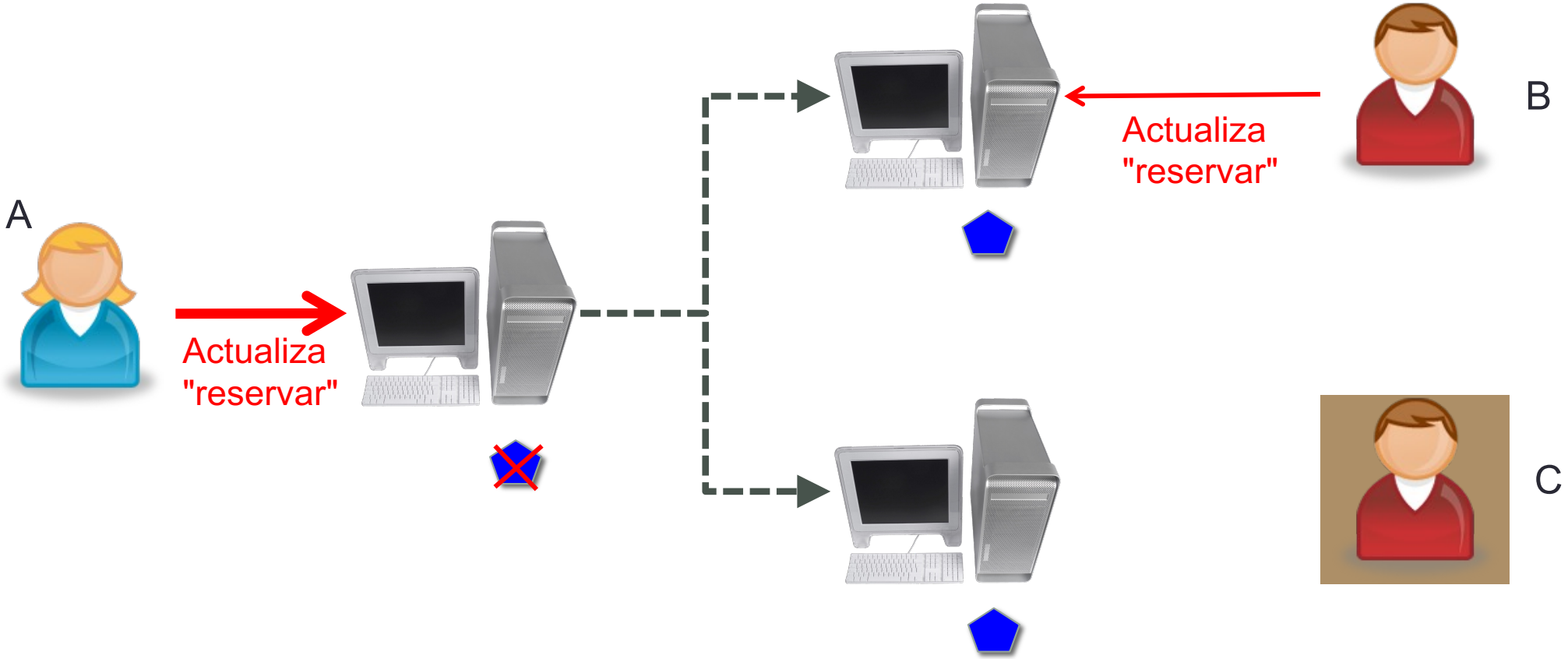
4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación entre pares



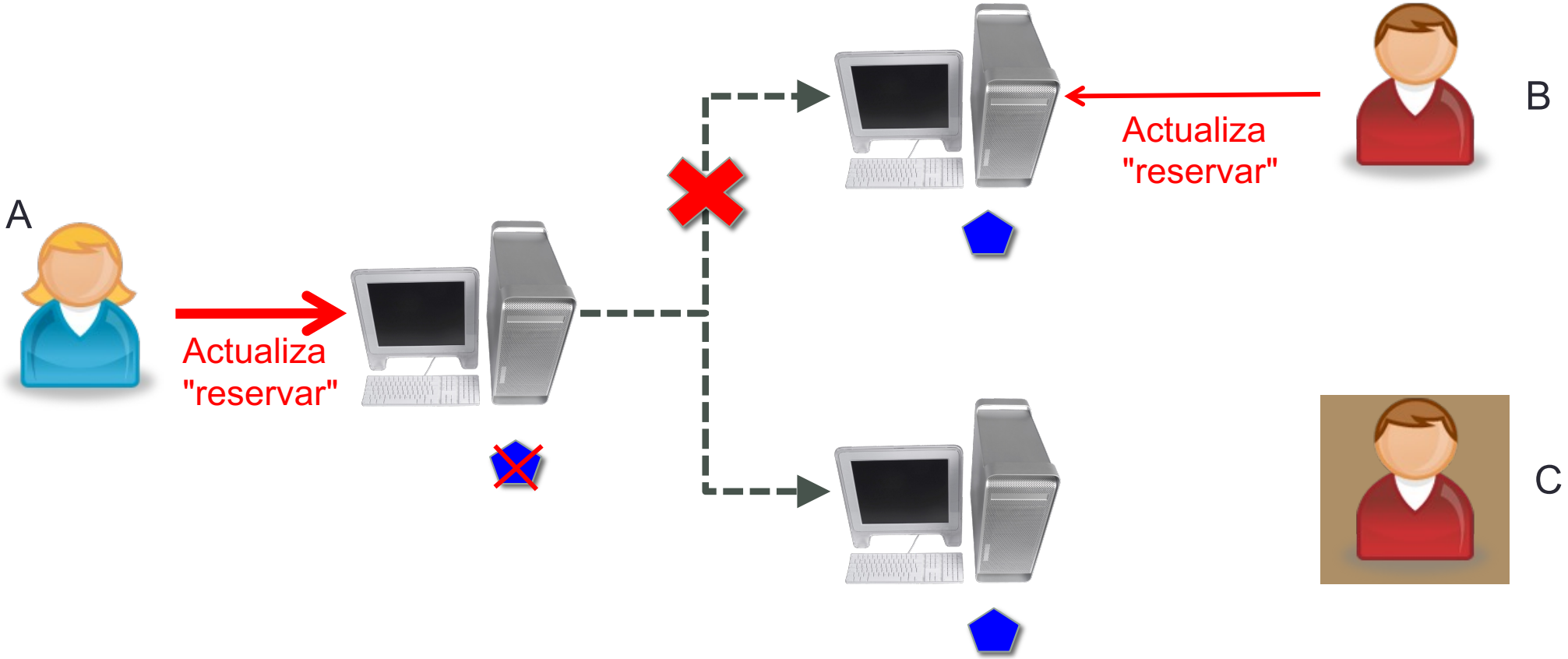
4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación entre pares



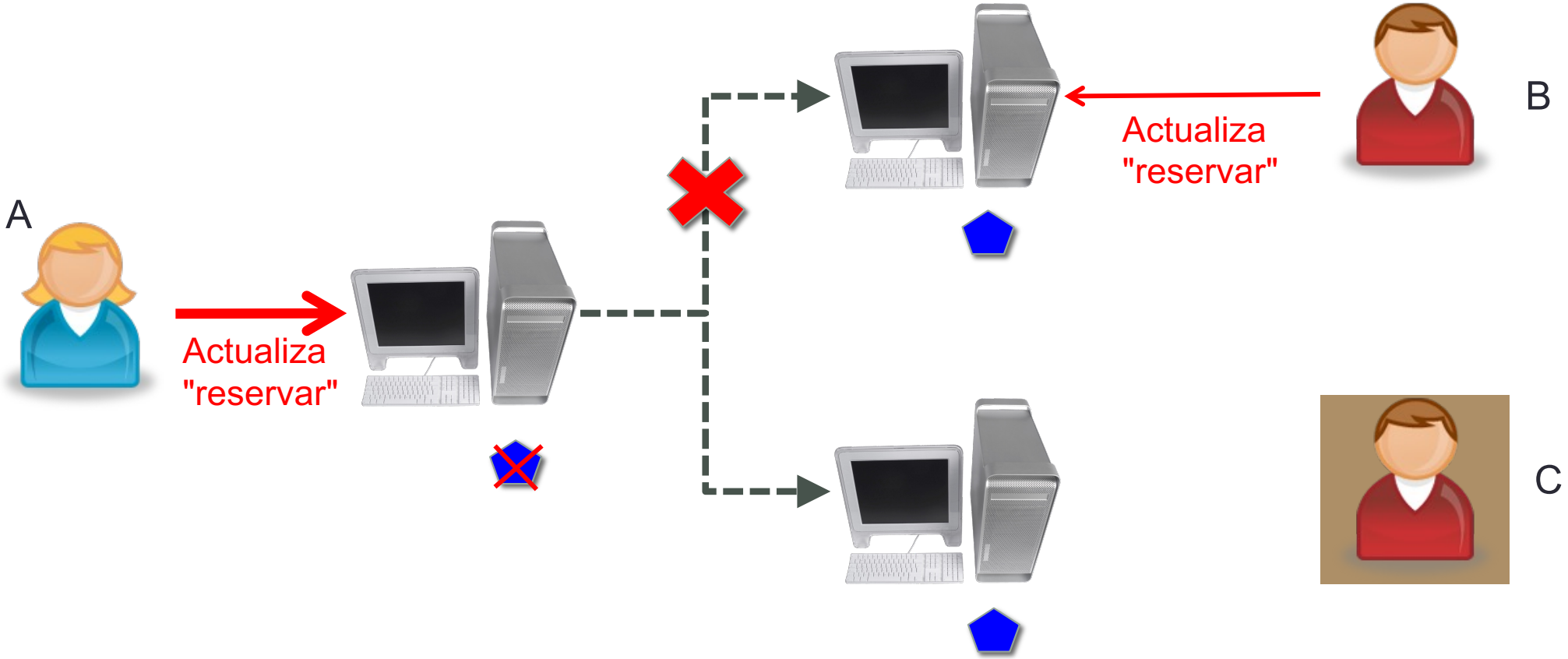
4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación entre pares



4. Consistencia

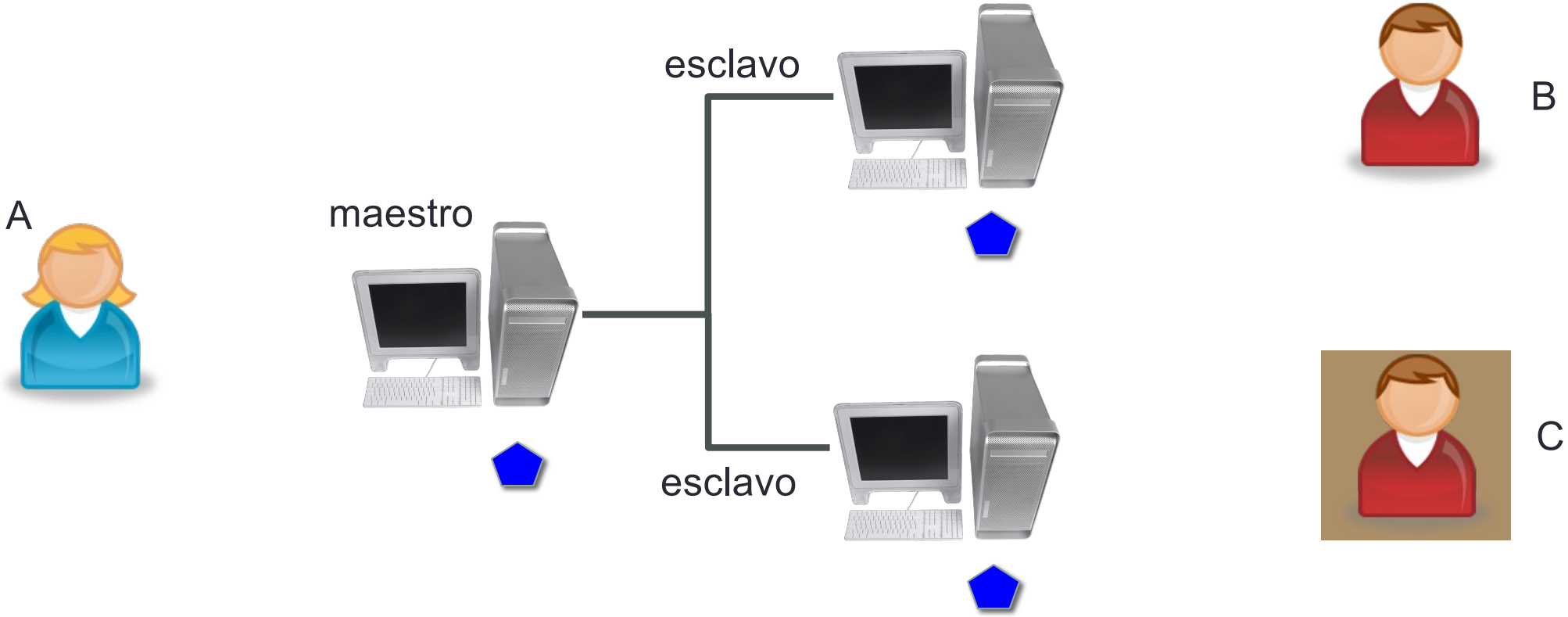
- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación entre pares



Sistema no puede reservar la habitación, disponibilidad sacrificada

4. Consistencia

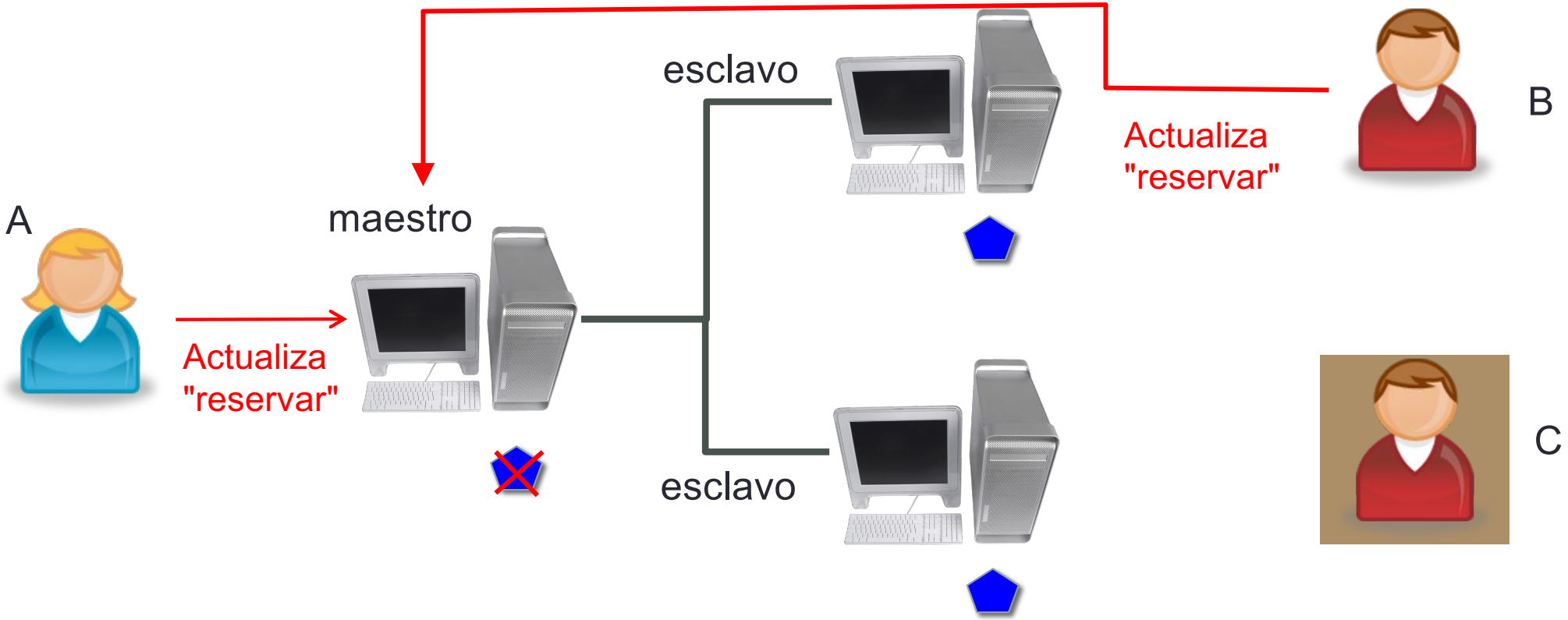
- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación maestro/esclavo



Última habitación disponible

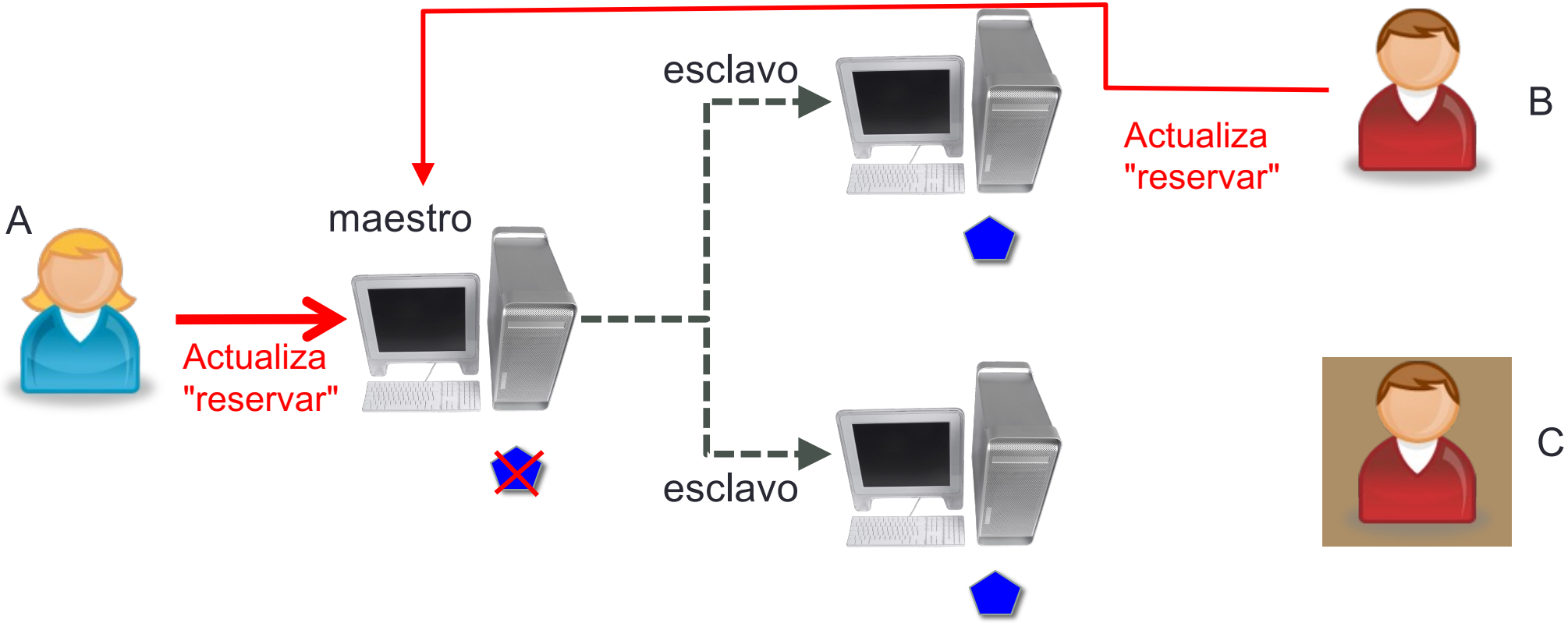
4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación maestro/esclavo



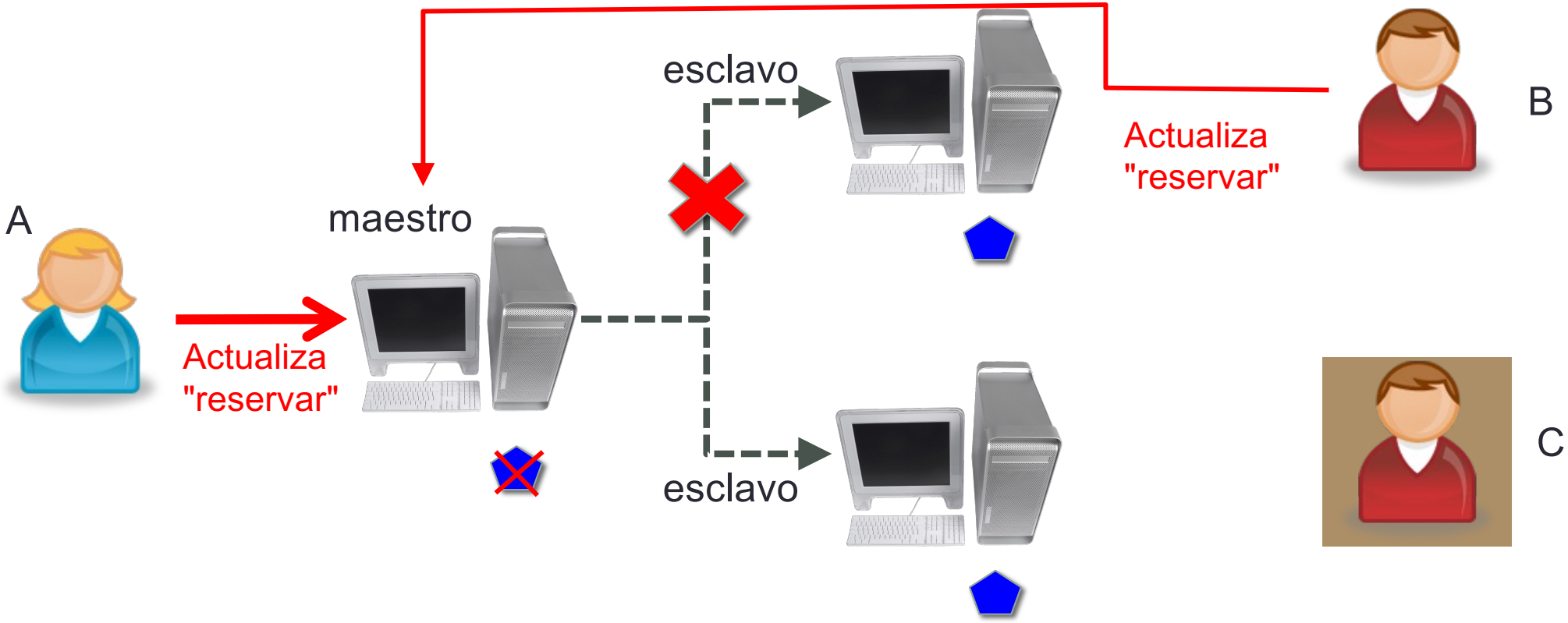
4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación maestro/esclavo



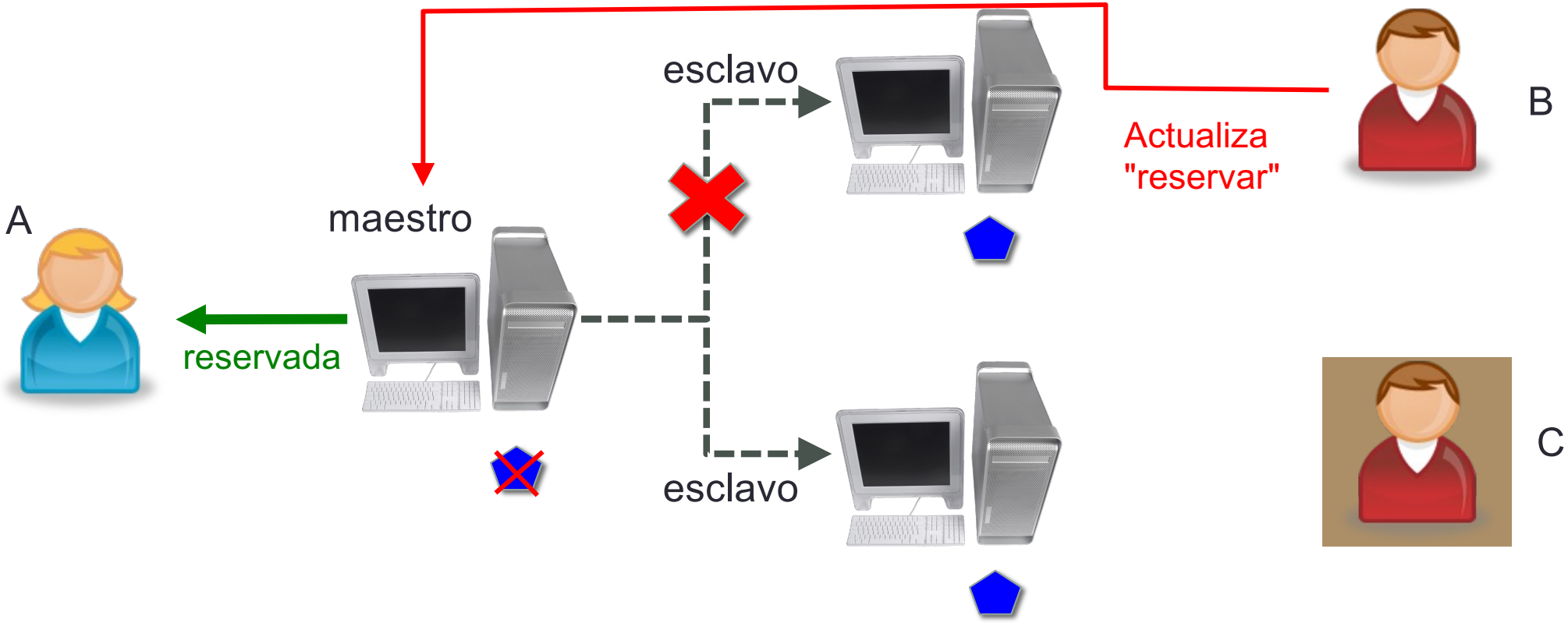
4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación maestro/esclavo



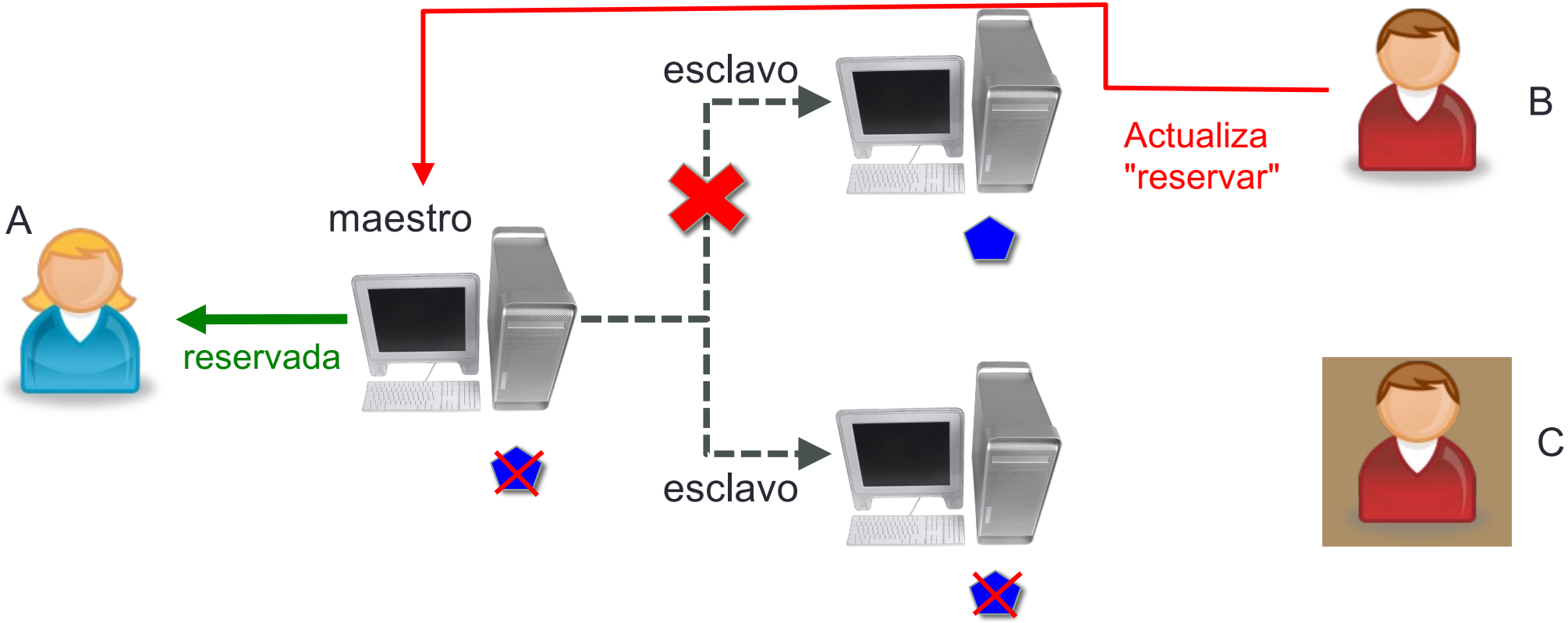
4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación maestro/esclavo



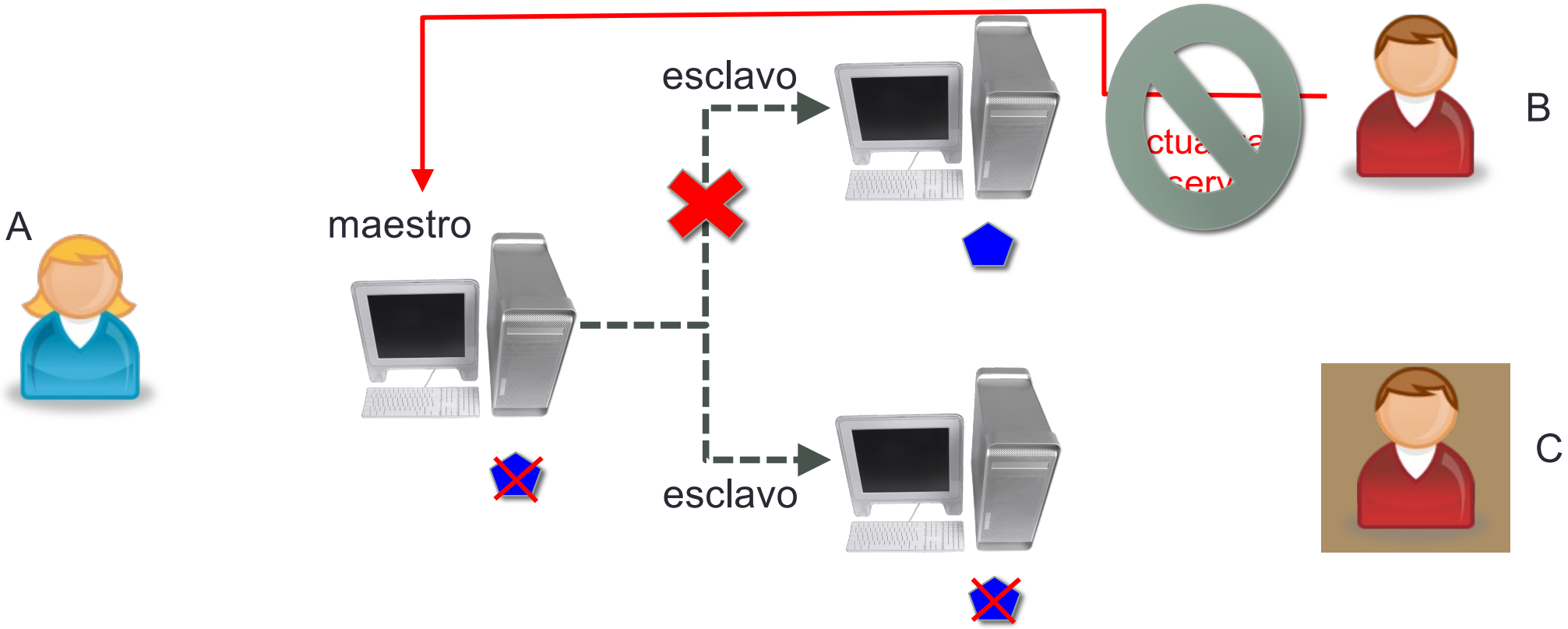
4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación maestro/esclavo



4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación maestro/esclavo



Inconsistencia de lectura, pero no inconsistencia de actualización
Disponibilidad limitada, el usuario B no puede reservar la habitación
Correcto, si es lo esperado por los usuarios

4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Cluster con fragmentación/replicación entre pares:
    - Incrementar la disponibilidad permitiendo que el sistema acepte reservas incluso con la red rota
    - Peligro**: Se reserve dos veces la misma habitación**
      - Aceptable si la organización admite overbooking
      - Aceptable si la organización se reserva algunas habitaciones
      - Aceptable si la organización admite la cancelación de reservas firmes
    - Amazon (Dynamo) admite escribir en tu carrito de la compra incluso con fallos en la red. Se pueden generar más de un carro. Se funden y se permite al usuario la modificación final

4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Tratamiento de la consistencia orientado al dominio
    - Consistencia de actualización
    - Consistencia de lectura
  - Tolerancia:
    - A las lecturas obsoletas: por tipo de datos
    - Duración de la ventana de inconsistencia: media, en el peor de los casos, distribución
  - NoSQL
    - No ACID
    - BASE (Basically Available, Soft state, Eventually consistency)

4. Consistencia

- Relajando la consistencia: Teorema CAP
  - Ante la tolerancia a la partición
    - Compromiso entre Consistencia vs Disponibilidad
    - **Compromiso entre Consistencia vs Latencia**
      - Disponibilidad: limite de latencia que se está preparado a tolerar

4. Consistencia

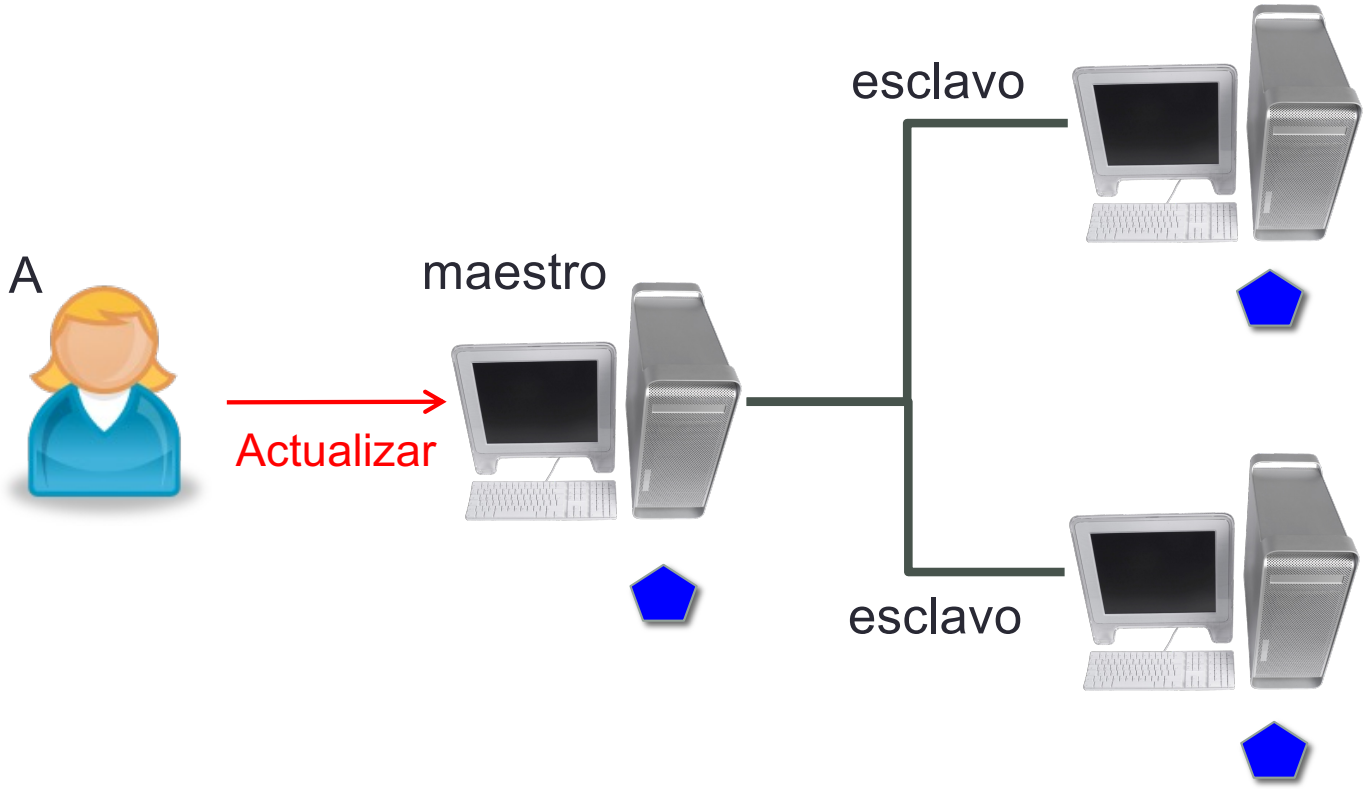
- Relajando la persistencia:
  - ACID se asocian con Consistencia
  - Persistencia: ¿sensato el relajarla? ¿útil?
    - ¿sensato el relajarla?: Almacén de datos que pierde datos
    - ¿útil?: Persistencia vs Rendimiento

4. Consistencia

- Relajando la persistencia:
  - Bases de datos sólo en memoria principal:
    - Cambios se aplican a las estructuras de memoria
    - Periódicamente se graba en disco
    - Riesgo de pérdida de información ante fallos o accidentes
  - Ejemplos:
    - Website que almacena el estado de la sesión de usuario
      - Mucha actividad, muchos cambios
      - La pérdida de la información de sesión no importante
    - Captura de datos telemétricos de un dispositivo físico
      - Mejora del ratio a cambio de perder algunos datos si el servidor falla

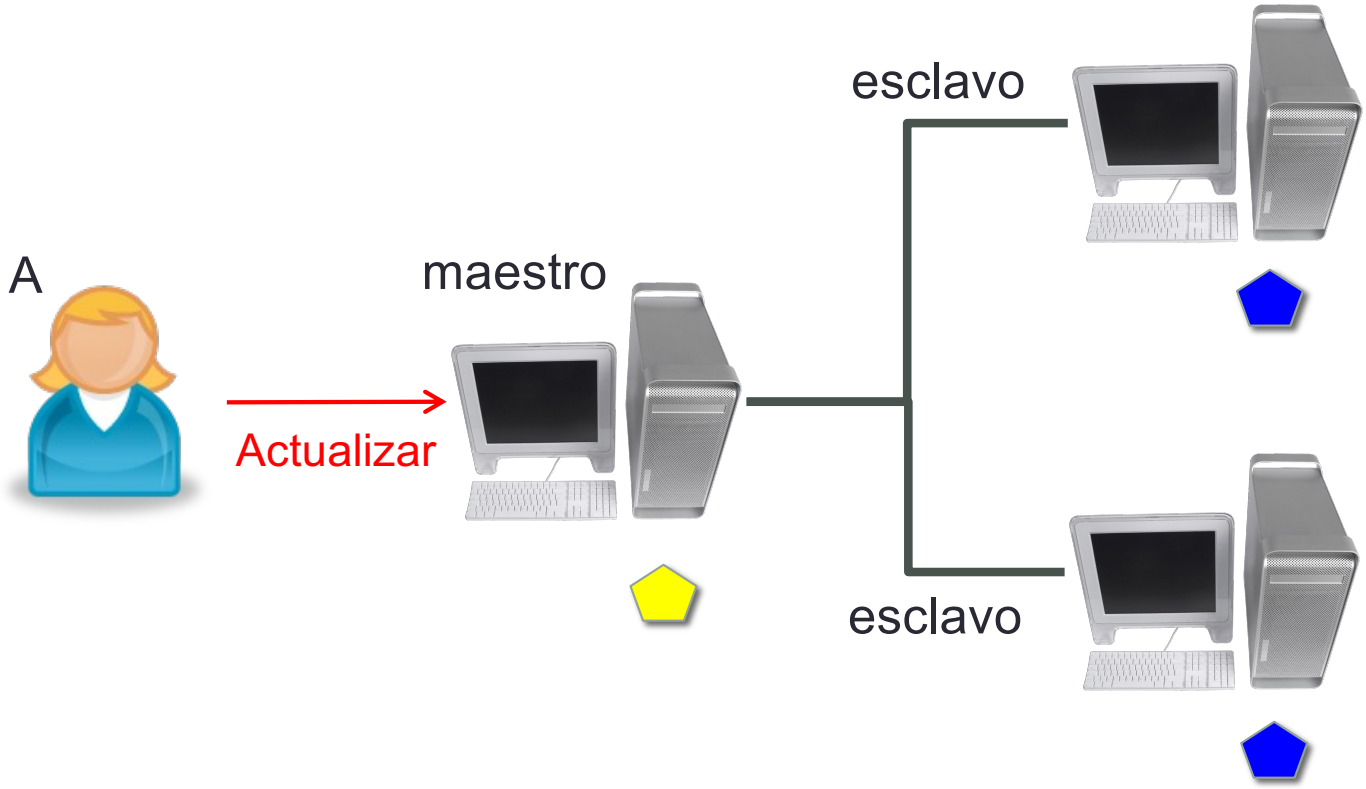
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Un nodo procesa una actualización pero falla antes de que ésta se haya replicado en los otros nodos



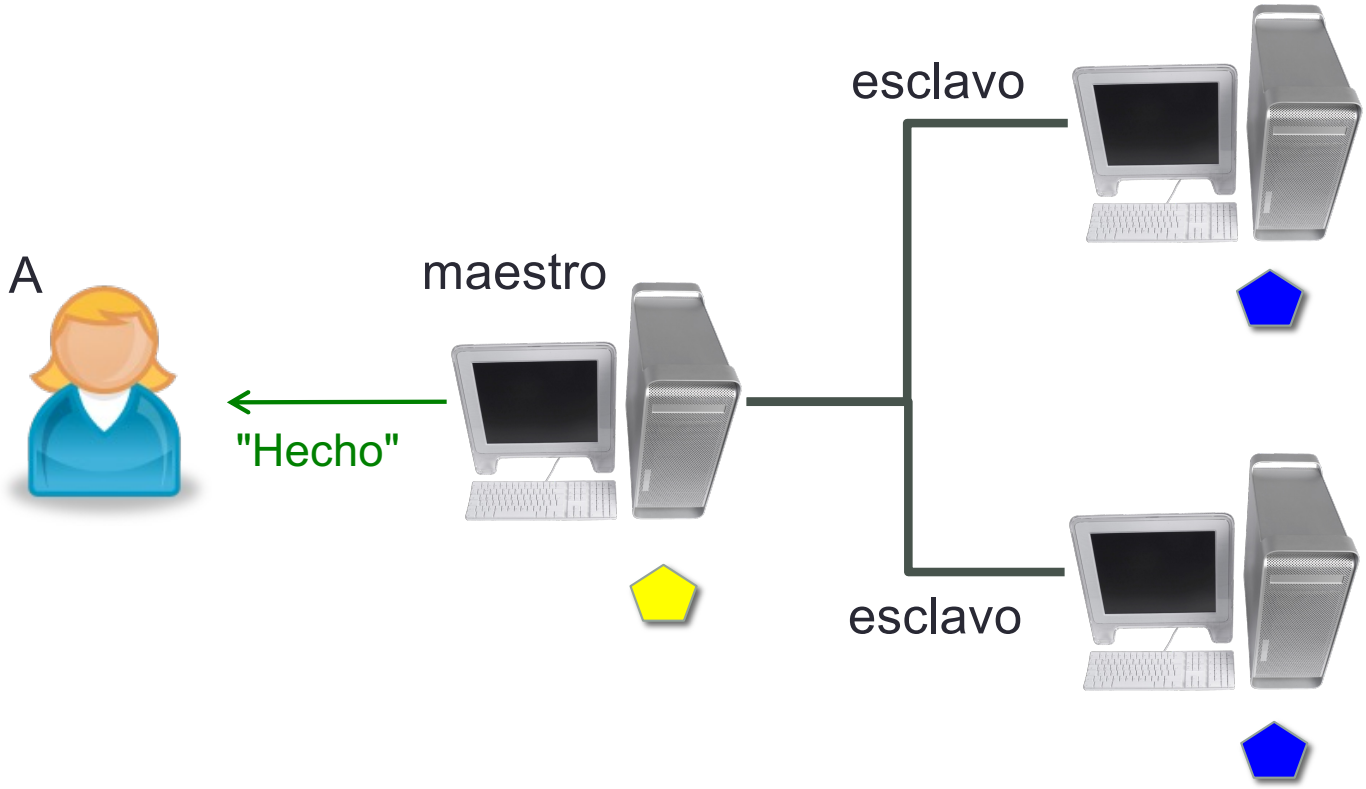
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Un nodo procesa una actualización pero falla antes de que ésta se haya replicado en los otros nodos



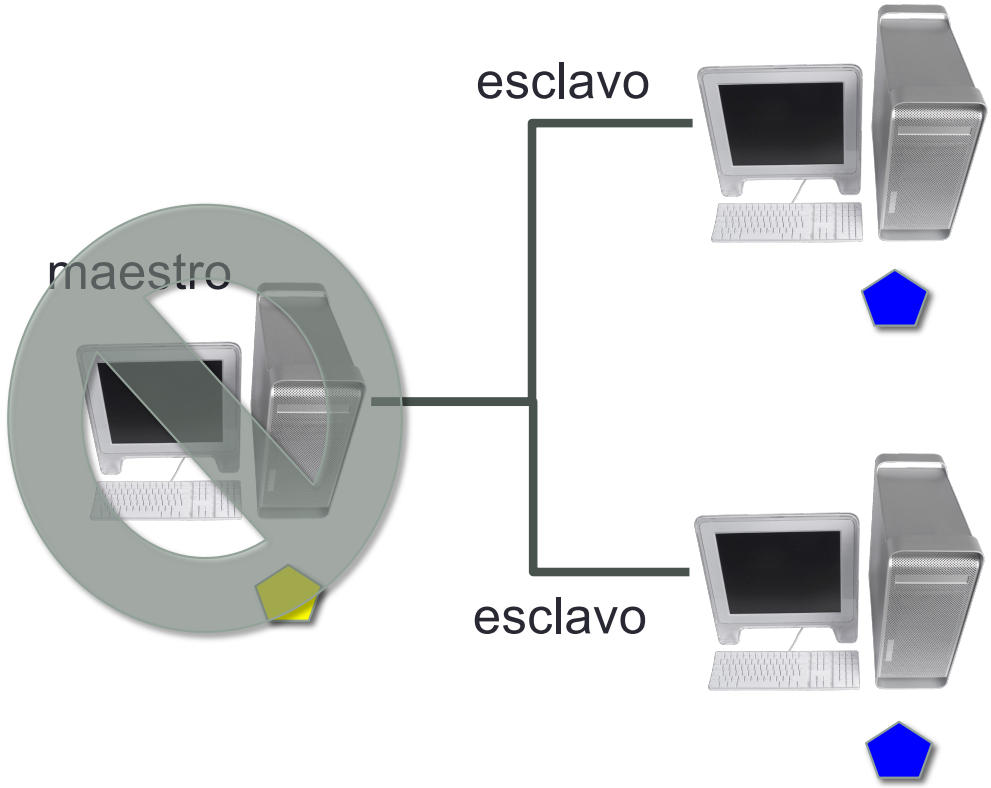
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Un nodo procesa una actualización pero falla antes de que ésta se haya replicado en los otros nodos



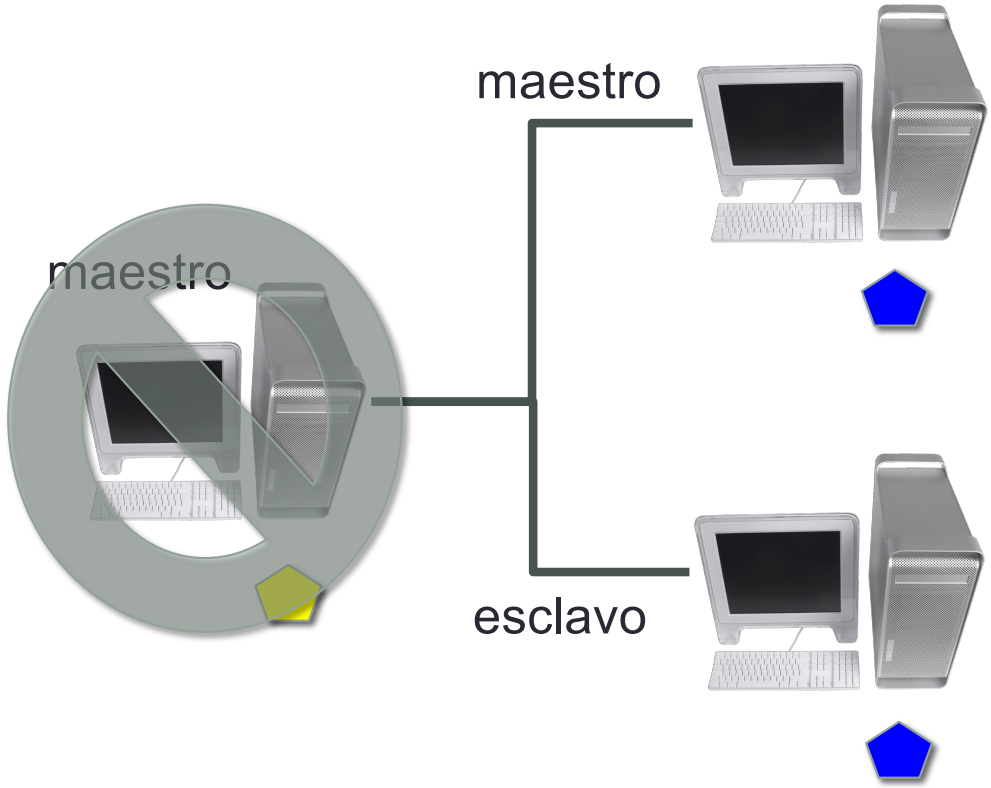
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Un nodo procesa una actualización pero falla antes de que ésta se haya replicado en los otros nodos



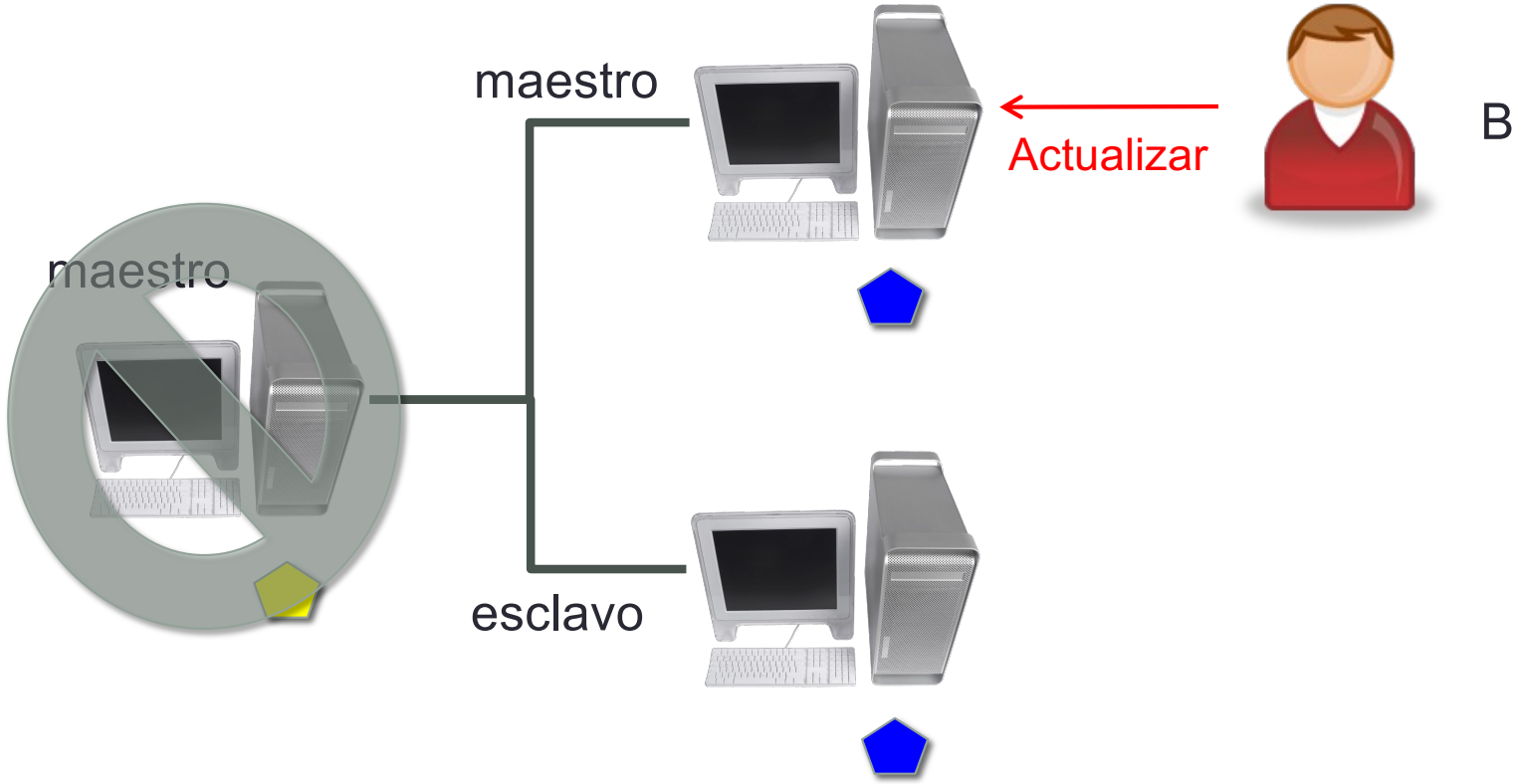
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Un nodo procesa una actualización pero falla antes de que ésta se haya replicado en los otros nodos



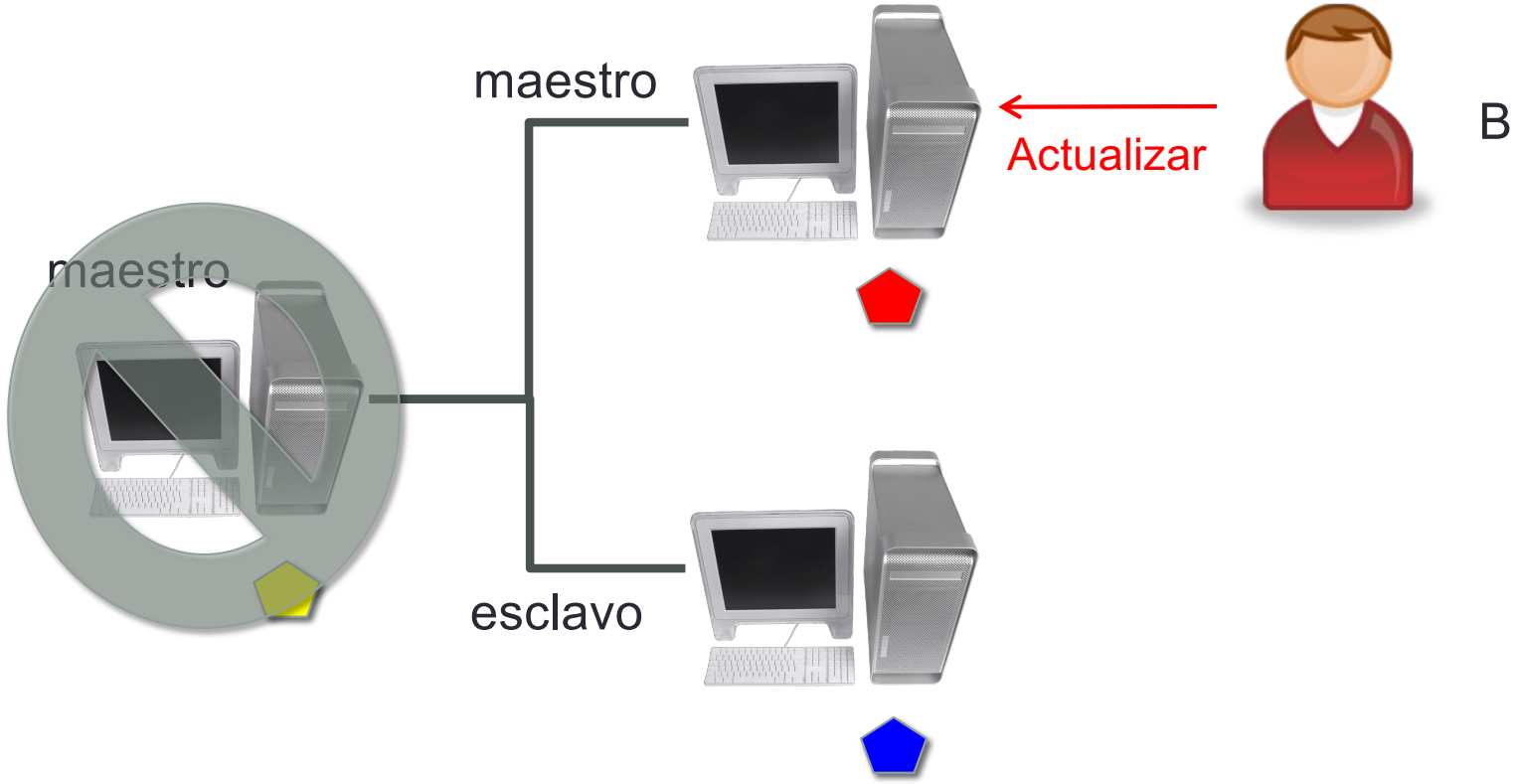
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Un nodo procesa una actualización pero falla antes de que ésta se haya replicado en los otros nodos



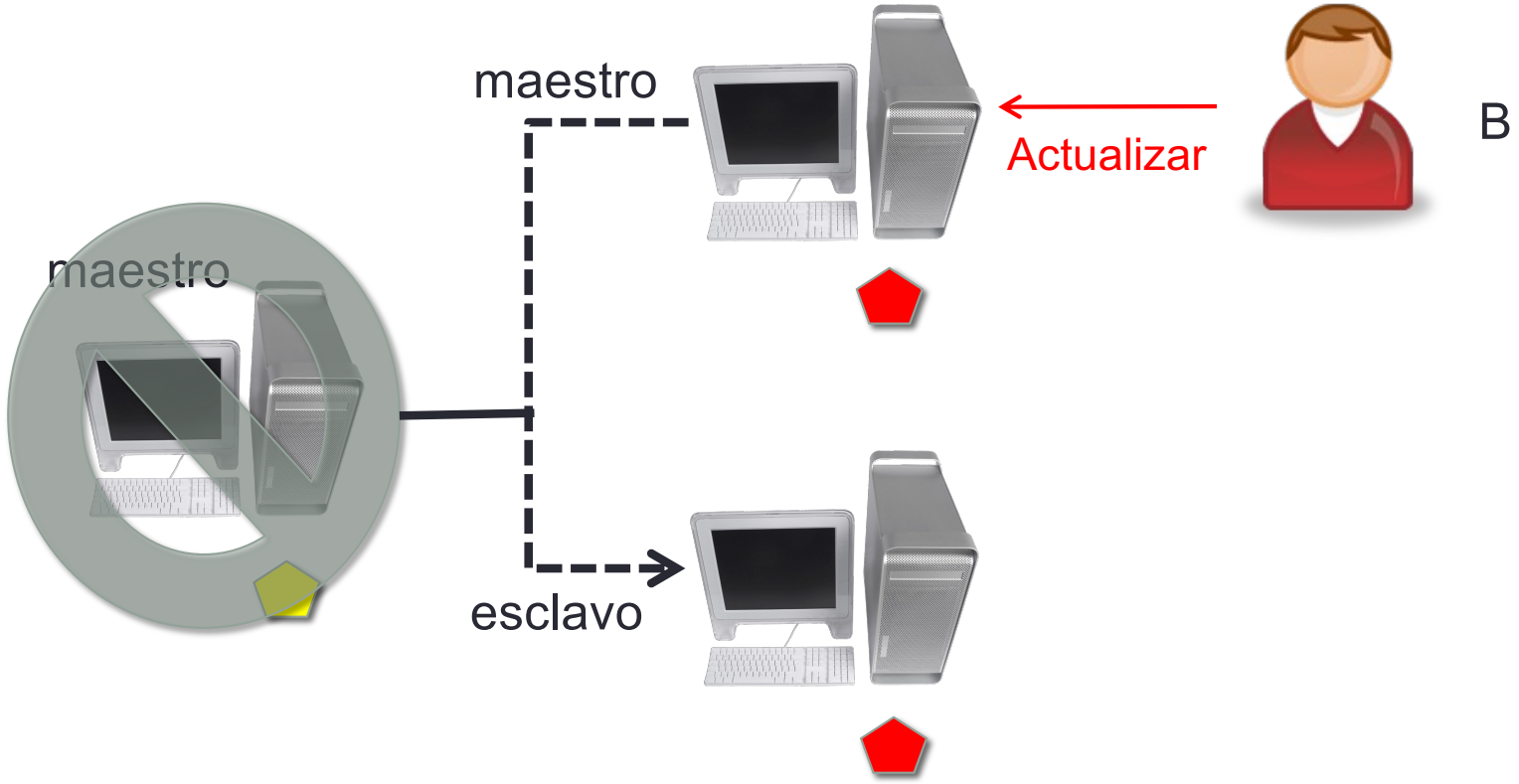
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Un nodo procesa una actualización pero falla antes de que ésta se haya replicado en los otros nodos



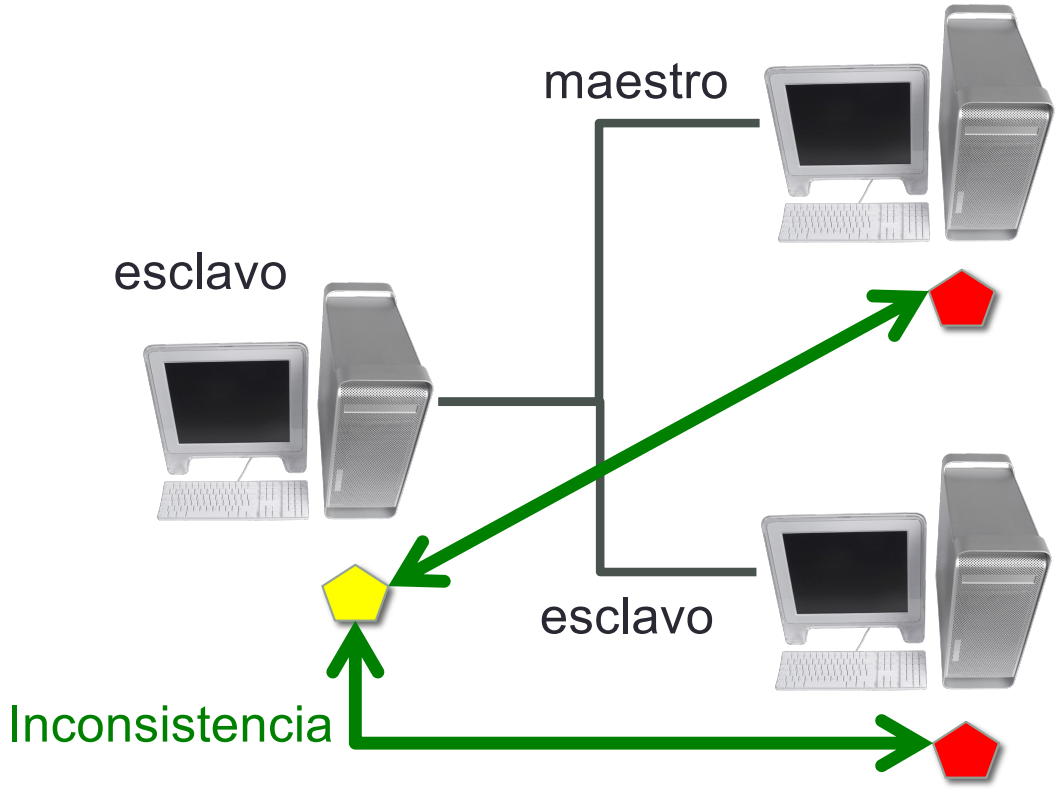
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Un nodo procesa una actualización pero falla antes de que ésta se haya replicado en los otros nodos



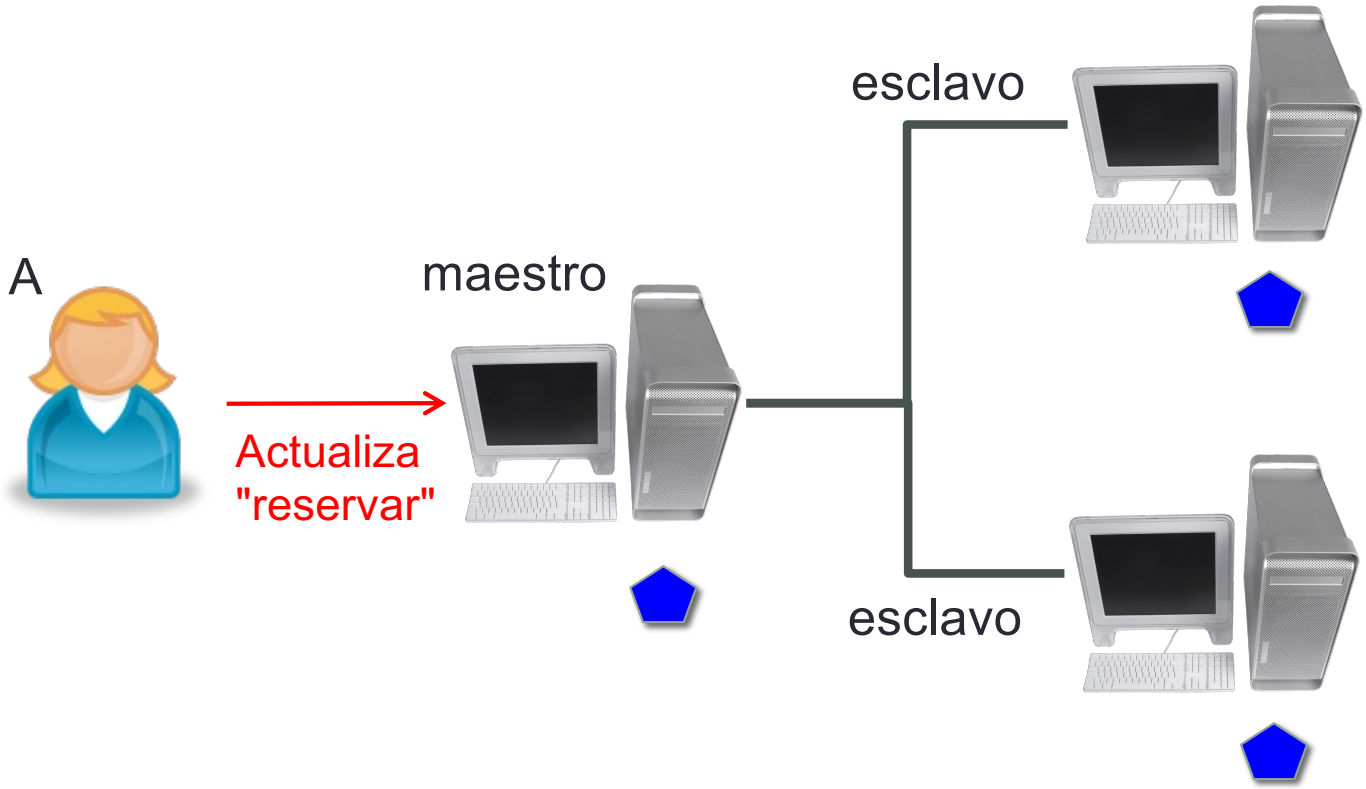
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Un nodo procesa una actualización pero falla antes de que ésta se haya replicado en los otros nodos



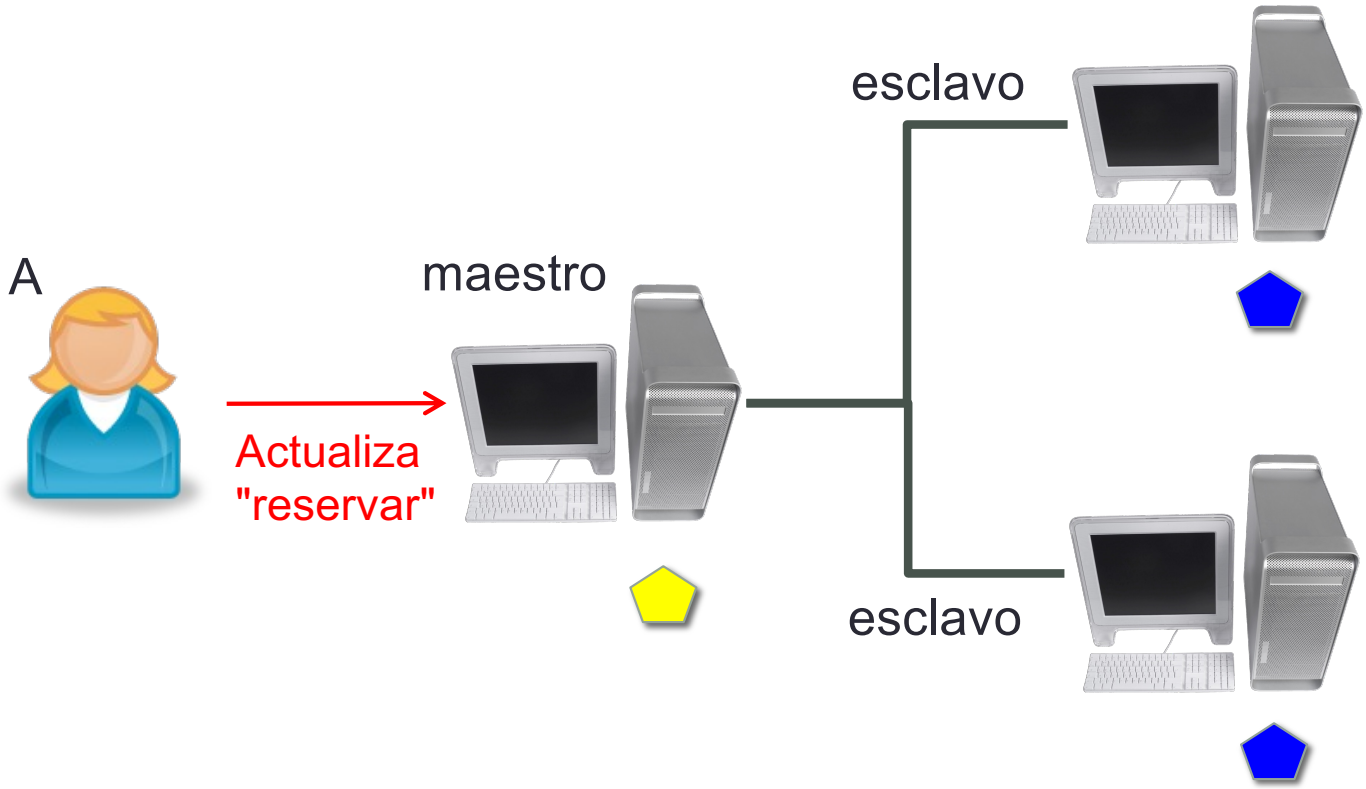
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Solución que el maestro espere la respuesta de la replicación en los esclavos



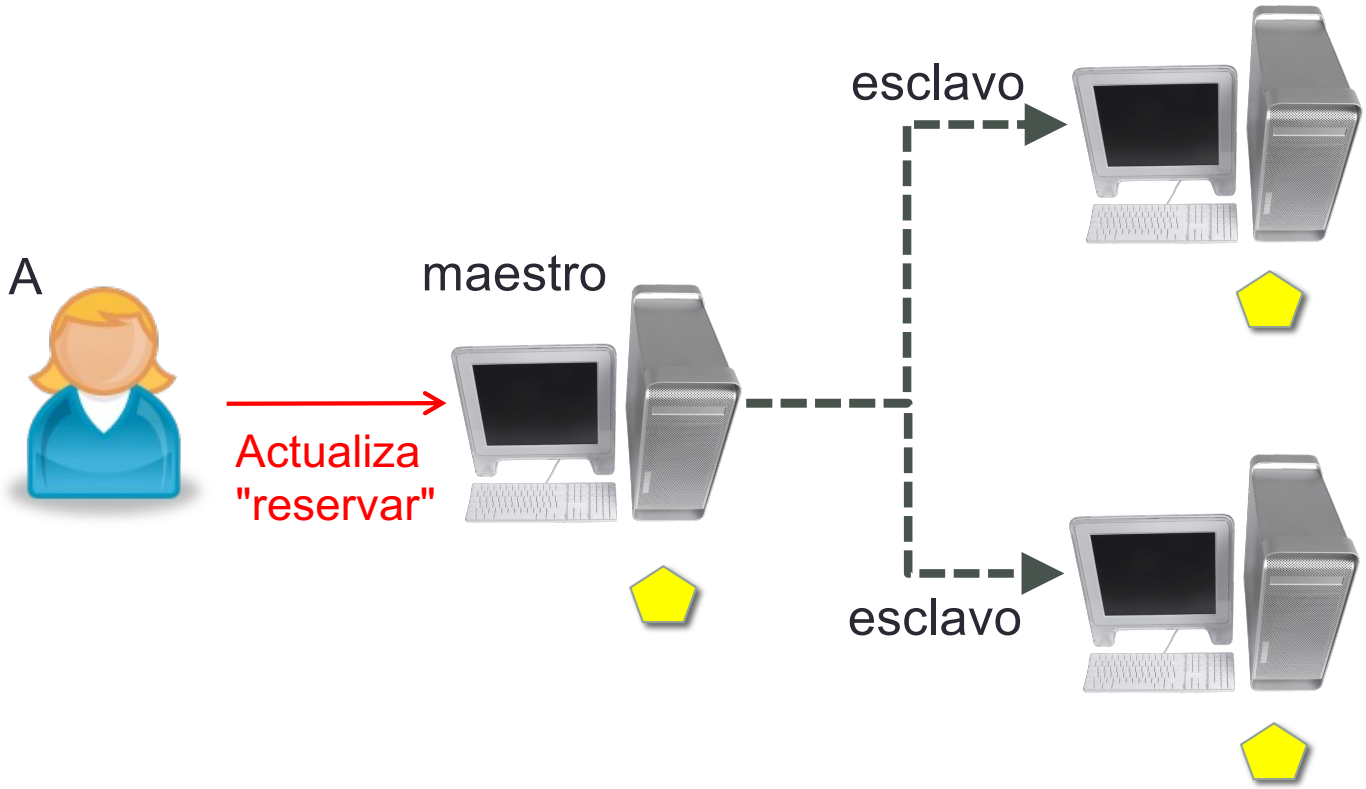
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Solución que el maestro espere la respuesta de la replicación en los esclavos



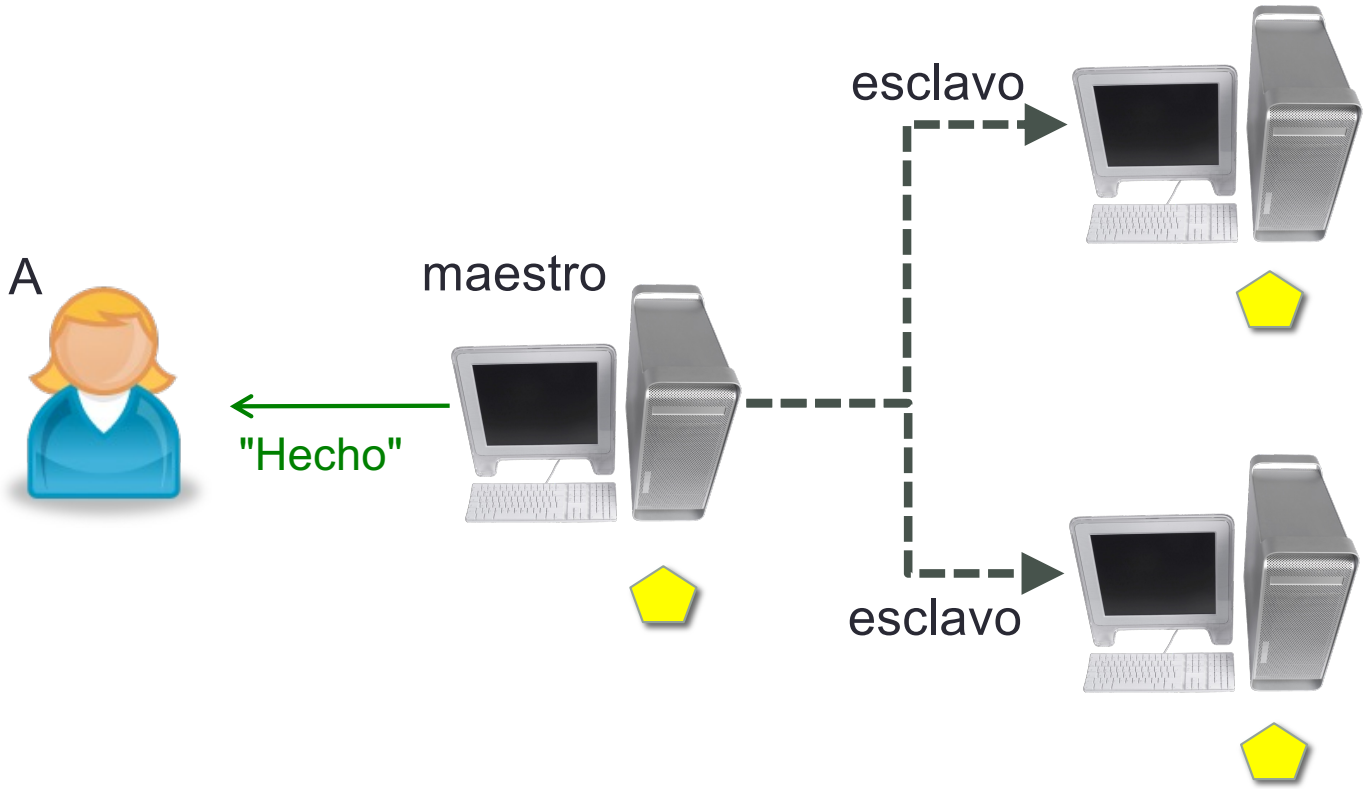
4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Solución que el maestro espere la respuesta de la replicación en los esclavos



4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Solución que el maestro espere la respuesta de la replicación en los esclavos



4. Consistencia

- Relajando la persistencia:
  - Persistencia de replicación:
    - Solución que el maestro espere la respuesta de la replicación en los esclavos
    - Actualizaciones más lentas
    - Disponibilidad amenazada si un esclavo falla

4. Consistencia

- Quórums:
  - Soluciones intermedias con Consistencia o Persistencia: no es un blanco/negro
  - Cuantos más nodos hay en el cluster más sencillo es evitar la inconsistencia
  - ¿Cuántos hacen falta que estén implicados para asegurar consistencia fuerte?

4. Consistencia

- Quórums:
  - Cluster de M nodos, replicación entre pares
  - Factor de replicación: N
    - Número de réplicas que se almacenan de los datos (número de nodos implicados en la replicación)
  - Quórum de escritura: W
    - Número de nodos que participan en una escritura para que se dé por válida
    - W > N/2
  - Quórum de lectura: R
    - Número de nodos a los que se ha de acceder para asegurar que se lee el valor más actualizado de una dato
  - Consistencia de lectura fuerte: R + W > N

4. Consistencia

- Quórums:
  - Ejemplo 1:
    - Cluster de 100 nodos, replicación entre pares
    - Factor de replicación: N=3
    - Quórum de escritura: W=2
    - Quórum de lectura: R=2
    - Consistencia de lectura fuerte: R + W > N
  - Equilibrio entre lecturas y escrituras
  - Permite el fallo de un nodo con réplicas y que el sistema siga disponible frente a lecturas y escrituras

4. Consistencia

- Quórums:
  - Ejemplo 2
    - Cluster de 100 nodos, replicación entre pares
    - Factor de replicación: N=3
    - Quórum de escritura: W=3
    - Quórum de lectura: R=1
    - Consistencia de lectura fuerte: R + W > N
  - Favorece las lecturas frente a las escrituras
  - Escrituras más lentas
  - Si un nodo falla, el sistema no estaría disponible para algunas escrituras

4. Consistencia

- Quórums:
  - La especificación del factor de réplica, el quórum de escritura y lectura se puede realizar por tipo de dato
  - Ajusta el equilibrio entre consistencia y disponibilidad de manera más precisa
  - Da más flexibilidad pero es más compleja

Índice

1. Introducción a la tecnología NoSQL
2. Modelos de datos
   2.1. Modelos de datos agregados
   2.2. Otros modelos y aspectos de modelado
3. Modelos de distribución de datos
4. Consistencia
5. Marcados de versión

5. Marcados de versión

- NoSQL:
  - Críticas por no soportar concepto **transacción**
  - Si se usa adecuadamente la agregación no son tan necesarias
- Transacción:
  - Tiene sus limitaciones:
    - Actualizaciones con intervenciones humanas → muy largas
- Marcado de versión:
  - Solución para estos problemas
  - Buen funcionamiento cuando nos movemos lejos del modelo de servidor único

5. Marcados de versión

- Transacciones de negocio y de sistema:
  - Sistemas necesitan soportar la consistencia de actualización sin transacciones
  - Se pueden construir sobre un sistema relacional transaccional
- Transacción de negocio:
  - Son las que tienen en mente los usuarios
  - Tienen que ver con la manera con la que el usuario quiere procesar sus datos
  - *"Busca en el catálogo de artículos, elige el artículo A con precio menor que B, rellena los datos de la tarjeta de crédito y confirma el pedido"*

5. Marcados de versión

- Transacción de sistema:
  - Son mucho más cortas y fragmentan una transacción de negocio en varias
  - Problema de la concurrencia y los bloqueos de datos
  - Aplicaciones:
    - Realizan una transacción de sistema al final de la interacción con el usuario
    - Cálculos y decisiones se pueden haber realizado sobre datos obsoletos (han sido cambiados por otro usuario)

5. Marcados de versión

- Bloqueo Optimista Offline:
  - Actualización condicional: antes de actualizar se comprueba si el dato ha cambiado
  - Se aplica en NoSQL
  - **Marcado de versión**: un campo que cambia cada vez que un dato en el registro (objeto, fila, etc.) cambia
  - Aplicaciones:
    - Cuando se lee el dato se toma nota de la marca de versión
    - Cuando se va actualizar se comprueba si la marca ha cambiado
  - Técnica típica cuando se modifican fuentes de datos vía HTTP: **etags**

5. Marcados de versión

- Bloqueo Optimista Offline:
  - Bases de datos proporcionan un mecanismo de actualización condicional
  - Evitan que la aplicación tenga que hacerlo inadecuadamente
  - Operación CAS (*Compare-And-Set*)
    - Antes de realizar la operación de actualización el sistema compara las marcas del cliente que solicita la operación y del dato que quiere actualizar

5. Marcados de versión

- Formas de marcado:
  - Contador: se incrementa en cada actualización
    - Sencillo
    - Permite conocer la "edad" (comparativas temporales)
    - Necesario una generación única
    - Adecuado en servidor único o maestro/esclavo;
  - GUID (*global unique identifier*):
    - Combinación de tiempo, hardware, etc.
    - Fácil garantizar la unicidad
    - No permite la comparativa temporal

5. Marcados de versión

- Formas de marcado:
  - Hash de contenido de la fuente:
    - Tamaño de clave de hash lo suficientemente grande para que se garantice la unicidad
    - Puede ser generado por cualquiera
    - Determinista: cualquier nodo puede generar el mismo contenido hash para el mismo dato de la fuente
    - No permite la comparativa temporal
    - Pueden ser largos

5. Marcados de versión

- Formas de marcado:
  - Marcas de tiempo:
    - Parecido a los contadores
    - Tamaño pequeño
    - No es necesario un maestro, pueden generarlos múltiples nodos → Los relojes se deben sincronizar
      - Un nodo con su reloj desincronizado puede causar inconsistencia
    - Necesario que el gránulo sea fino para asegurar unicidad: depende de la carga de actualización

5. Marcados de versión

- Formas de marcado:
  - Se pueden usar varias: CouchDB
    - Contador y hash de contenido
    - Permite la mayoría de las veces comparativas temporales
    - Si dos nodos actualizan a la vez:
      - Combinación del mismo contador y diferentes claves de contenido hace sencillo identificar el conflicto
- También son útiles para proporcionar consistencia de sesión

5. Marcados de versión

- Marcado de versión básico:
  - Funciona correctamente cuando se tiene una fuente de datos única acreditadora
    - Servidor único
    - Replicación maestro/esclavo: el esclavo es dirigido en su marcado por el maestro
  - No funciona en un cluster con replicación entre pares: no hay un nodo único responsable del marcado

5. Marcados de versión

- Marcado de versión con múltiples nodos:
  - Problema: Lectura de un dato
    - Devuelven dos valores dos nodos distintos
    - La reacción puede variar dependiendo de la causa que provoca esa diferencia

      A) Un actualización que ha llegado a un nodo pero no al otro: se acepta la última

      B) Actualización inconsistente:
      - Hay que decidir que hacer
      - Un GUID o etag no es suficiente porque no establece relaciones temporales entre los dos valores

5. Marcados de versión

- Marcado de versión con múltiples nodos:
  - Replicación maestro/esclavo: contador
    - Un maestro y dos esclavos (**A** y **B**)
    - Si se pide acceder a un dato
    - **A** devuelve un valor con marca 6 y **B** un valor con marca 4
    - Se sabe que el valor de **A** es más reciente

5. Marcados de versión

- Marcado de versión con múltiples nodos:
  - Replicación entre pares: contador
    - No sirve, ya que debería haber un nodo que generase las marcas

5. Marcados de versión

- Marcado de versión con múltiples nodos:
  - Replicación entre pares: marca temporal
    - Difícil
    - Todos los nodos han de tener una noción de tiempo consistente
    - Se incrementa la dificultad si las actualizaciones se suceden con rapidez
    - Si un nodo se desincroniza puede generar inconsistencias
    - Se pueden dar conflictos de escritura-escritura

5. Marcados de versión

- Marcado de versión con múltiples nodos:
  - Replicación entre pares: vector de marcas
    - Vector de contadores, uno por nodo
    - Nodos: A, B y C en el cluster
    - Vector de marcas en el nodo A → [A: 43, B: 54, C:12]
    - Cada vez que un nodo realiza una actualización, incrementa su contador y actualiza el vector de marcas
    - Cada vez que dos nodos entran en comunicación sincronizan sus vectores de marcas
    - Hay diversas formas de sincronización:
      - Vector de relojes: establecer un orden parcial entre eventos de un sistema distribuido
      - Vector de versión: establecer un preorden entre las replicas en un sistema distribuido

5. Marcados de versión

- Marcado de versión con múltiples nodos:
  - Replicación entre pares: vector de marcas – orden
    - [A: 43, **B: 54**, C:12] es más reciente que [A: 43, **B: 52**, C:12]
    - Entre [**A: 43**, **B: 54**, C:12] y [**A: 44**, **B: 52**, C:12] no hay orden, ya que cada uno de ellos tiene un contador mayor que el otro → conflicto escritura-escritura
    - Si faltan contadores en un vector, se asimilan a contador 0

→

permite introducir nuevos nodos en el cluster sin invalidar los vectores de marcas existentes

5. Marcados de versión

- Marcado de versión con múltiples nodos:
  - Replicación entre pares: vector de marcas
    - Útil para detectar inconsistencias
    - No las resuelve
    - La resolución de los conflictos se realiza en la aplicación y dependerá del dominio en el que ésta se desarrolle
    - Parte del compromiso Consistencia/Latencia (disponibilidad)