Bases de Datos

Clase 13: ORM – NoSQL y full-text search & retrieval

Programación y SQL

- Hasta ahora hemos visto a la Base de Datos como un componente aislado
- Pero una Base de Datos no tiene sentido si no podemos conectarla a una aplicación

Programación y SQL

Contamos con:

- Un DBMS
- Un entorno de programación (Java, Python, PHP…)

¿Podemos hacernos la vida más fácil? Si, con **frameworks**

Frameworks Web

Frameworks Web

Un framework es una librería que permite crear aplicaciones de manera más simple y rápida. Algunos ejemplos de frameworks web son:

- **Flask** (Python):

Framework minimalista. Permite una gran flexibilidad y control sobre el diseño y la funcionalidad de la aplicación. Ideal para proyectos pequeños y medianos.

- **Django** (Python):

Framework de alto nivel que fomenta el desarrollo rápido y el diseño limpio. Incluye muchas características integradas, como autenticación, administración, y **ORM**.

ORM

ORM

Un **ORM** (Object-Relational Mapping) es una técnica de programación que facilita la interacción entre modelos de datos en un lenguaje orientado a objetos y una base de datos relacional. Este traduce operaciones **CRUD** (Create, Read, Update, Delete) de objetos en el código a comandos SQL.

Tiene las siguientes características:

- **Mapeo automático:** Genera tablas y columnas desde clases y atributos.
- **Consultas expresivas:** Construye consultas complejas en el lenguaje de programación.
- **Validación y restricciones:** Asegura datos consistentes y válidos.
- **Gestión de relaciones:** Maneja relaciones entre entidades (uno a uno, uno a muchos, muchos a muchos).

ORM - Ejemplo

Un ORM permite linkear el código a una base de datos, veamos un ejemplo en SQLAlchemy y Python:

```python
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(DATABASE_URL)
Base = declarative_base()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    email = Column(String, unique=True, index=True)

Base.metadata.create_all(bind=engine)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
db = SessionLocal()
new_user = User(name="John Doe", email="johndoe@example.com")
db.add(new_user)
db.commit()
db.refresh(new_user)

users = db.query(User).all()
for user in users:
    print(f"ID: {user.id}, Name: {user.name}, Email: {user.email}")
```

ORM - Ejemplo

```python
from sqlalchemy import create_engine, Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker

DATABASE_URL = "sqlite:///./test.db"
engine = create_engine(DATABASE_URL)
Base = declarative_base()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True, index=True)
    name = Column(String, index=True)
    email = Column(String, unique=True, index=True)

Base.metadata.create_all(bind=engine)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
db = SessionLocal()
new_user = User(name="John Doe", email="johndoe@example.com")
db.add(new_user)
db.commit()
db.refresh(new_user)

users = db.query(User).all()
for user in users:
    print(f"ID: {user.id}, Name: {user.name}, Email: {user.email}")
```

Se conecta a la BDD

Se hace la tabla y se sube a la base de datos con SQLAlchemy

Se añade un User

Se hace una consulta

SQL en frameworks web

SQL en frameworks web

Los *frameworks* web tienen librerías para abstraerse de la base de datos

Un **ORM** (Object-Relational Mapping) es una técnica para tratar a los datos de un sistema como objetos de un lenguaje de programación. Veamos el uso de un ORM con **Django**

ORM

Ejemplo - Modelos

```
from django.db import models


class Musician(models.Model):
    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    instrument = models.CharField(max_length=100)

class Album(models.Model):
    artist = models.ForeignKey(Musician, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    release_date = models.DateField() n
    um_stars = models.IntegerField()
```

ORM

Ejemplo - Consultas

Obtener todos los músicos:

```
>>>
Musician.objects.all()
```

Obtener todos los músicos con nombre ‘James’:

```
>>>
Musician.objects.filter(first_name=‘James’)
```

Obtener todos los álbumes del artista con id 1:

```
>>>
Musician.objects.get(id=1).album_set.all()
```

ORM

Un ORM permite abstraerse de un sistema de bases de datos en particular

No es tan flexible como utilizar SQL, y no depende del desarrollador cómo se traducen las consultas

Nosotros instalamos la base de datos, pero el ORM se encarga de utilizarla

Los ORM nos puede ser de gran ayuda en las **migraciones**

Migraciones

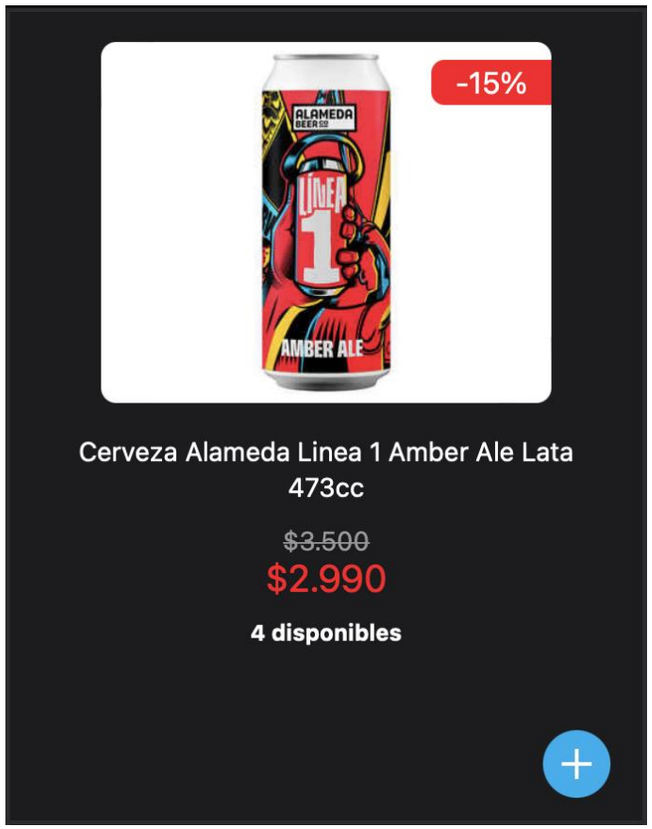
Migraciones - Ejemplo

Digamos que estamos haciendo un modelo de precios y stocks de cervezas en distintas tiendas. Tenemos el siguiente esquema que hasta ahora nos ha funcionado bien:

- Cervezas(id, nombre, ibu, abv, id_marca, precio)
- Marcas(id, nombre, país)
- Tiendas(id, nombre)
- Stocks(id_tienda, id_cerveza, cantidad)

Migraciones - Ejemplo

Tenemos un problema: ¡Nuestra jefa nos comenta que los precios de una misma cerveza no necesariamente son los mismos para tiendas distintas!



¿Cómo lo solucionamos?

Migraciones - Ejemplo

Una opción es:

```
CREATE TABLE Precios(
    id_cerveza INT NOT NULL,
    id_tienda INT NOT NULL,
    precio DECIMAL(10, 2),
    FOREIGN KEY id_cerveza REFERENCES Cervezas(id),
    FOREIGN KEY id_tienda REFERENCES Tiendas(id)
)
```

<copio los precios de la tabla Cervezas a la nueva tabla Precios para no perder información>

```
ALTER TABLE Cervezas
DROP COLUMN precio;
```

¿Existe una forma major de hacerlo? Si, con **migraciones**

Migraciones

El esquema de una base de datos define las tablas y sus relaciones.

En la vida de un proyecto los requerimientos van cambiando, y el esquema de la base de datos necesita cambiar.

Usamos migraciones para modificar el esquema de la base de datos en la medida que las necesidades van cambiando.

Ejemplo: Migración en Django

Primero definamos los modelos:

```python
from django.db import models

class Beer(models.Model):
    name = models.CharField(max_length=100)
    # Eliminaremos esta columna en la migración
    # price = models.DecimalField(max_digits=10, decimal_places=2)

class Price(models.Model):
    beer = models.OneToOneField(Beer, on_delete=models.CASCADE, primary_key=True)
    price = models.DecimalField(max_digits=10, decimal_places=2)
```

Ejemplo: Migración en Django

Ahora hagamos la migración:

```python
from django.db import migrations, models

class Migration(migrations.Migration):
    dependencies = [
        ('myapp', '0001_initial'),
    ]
    operations = [
        # Migracion para crear el modelo Price
        migrations.CreateModel(
            name='Price',
            fields=[
                ('beer', models.OneToOneField(on_delete=models.CASCADE,
                                              primary_key=True, serialize=False, to='myapp.Beer')),
                ('price', models.DecimalField(decimal_places=2, max_digits=10)),
            ],
        ),
        # Migracion para eliminar el campo price de Beer
        migrations.RemoveField(
            model_name='beer',
            name='price',
        ),
    ]
```

Migraciones

- Las creamos y probamos en nuestra copia local de la base de datos.
- Si no queda bien, deshacemos la migración (rollback), la arreglamos y la corremos de nuevo.
- Cuando está todo bien subimos nuestros cambios (a git por ejemplo).
- Al lanzar versiones nuevas se corren (automáticamente) las migraciones pendientes.
- Si el lanzamiento tiene problemas se puede hacer redeploy de la version anterior del servicio junto con un rollback de las migraciones relacionadas a ese lanzamiento.

Migraciones problemáticas

Las migraciones de la base de datos corren después del deploy, en la etapa del release de la aplicación (antes de que la versión nueva empiece a funcionar).

Los cambios que se hagan deben ser compatibles la versión anterior de la aplicación.

Para hacer cambios grandes a veces es necesario hacer varias migraciones en varios lanzamientos de la aplicación para evitar problemas.

Migraciones problemáticas: Renombrar una tabla o columna

Al renombrar una tabla o una columna la versión anterior de la aplicación no va a poder leer la tabla, para hacer el cambio es necesario hacerlo en varios pasos:

1. Crear una tabla o columna nueva con el nombre correcto.
2. Escribir los datos nuevos en ambas tablas/columnas.
3. Copiar los datos de la tabla/columna antigua a la nueva.
4. Cambiar las lecturas de la tabla/columna antigua a la nueva.
5. Cuando todo esté bien dejar de escribir a la tabla/columna nueva y borrarla.

Rediseñar una base de datos

Hacer todo de nuevo generalmente es la peor decisión.
Si es posible arreglar el modelo de datos sin partir de 0 suele ser preferible, pero a veces es necesario.

¿Cómo migramos una base de datos mal diseñada a otra nueva, pero bien hecha?

El proceso es similar a renombrar una tabla, pero es un proceso mucho más largo y engorroso.

Rediseñar una base de datos

1. Partimos de a poco modelando correctamente algunas tablas en la base de datos nueva.
2. Escribimos datos nuevos a ambas tablas. Hay que tener cuidado con referencias entre distintas bases de datos. PostgreSQL soporta llaves foráneas entre distintas bases de datos, pero las ORMs en general no).
3. Copiamos los datos de la base de datos vieja a la nueva.
4. Eventualmente estamos seguros de que la tabla en la base de datos nueva funciona correctamente y dejamos de usar la tabla de la base de datos vieja.

Modelación de datos

Modelación de datos

Más allá del uso que nuestra aplicación le dé a los datos, eventualmente los vamos a querer para hacer análisis, reportes, campañas, entre otros.

Si el esquema es engorroso y difícil de entender, va a ser difícil trabajar con los datos. Arreglar una base de datos mal modelada es generalmente una tarea poco abarcable.

Eventualmente queremos poder tomar decisiones a partir de los datos.

Modelación de procesos

A veces queremos modelar procesos que van cambiando de estados:

- Una compra (pagada, en preparación, despachada)
- Una revisión manual (en espera de revisión, distintos resultados de la revisión: aprobada, rechazada, u otros)

Modelación de procesos

Podríamos querer modelarlo como un campo de la tabla:

Order(id, state, comment, created_at, updated_at)
O también:

Order(id, comment, ordered_at, paid_at, prepared_at, shipped_at)

¿Estamos seguro que no perdemos información así? ¿Qué pasa si queremos guardar información adicional sobre las etapas del proceso? Para solucionar estos, creamos nuevas tablas.

Modelación de procesos

Order(id, comment, created_at)

PaidOrder(id, order_id, created_at, payment_method, payment_receipt)

PreparedOrder(id, order_id, created_at, comment)

ShippedOrder(id, order_id, created_at, tracking_number, shipping_company, expected_delivery_date)

Rendimiento y ORMs

Rendimiento y ORMs

Generalmente en una aplicación web lo más lento es la base de datos (cuando no hay mucha lógica implementada en el código).

Muchas veces la aplicación no funciona lento porque los datos estén mal modelados, o porque la consulta sea muy pesada.

¡Podríamos estar haciendo muchas consultas chicas!

También nos podría estar haciendo falta un índice.

Rendimiento y ORMs

```
def my_view(request, article_id):
    article = Article.objects.get(article_id)
    return render(
        request,
        "my_view.html",
        {"article": article, "author":
        article.author}
    )
```

Este pedazo de código hace (al menos) dos consultas a la base de datos:

- La primera para traer el artículo que venía como argumento de la vista.
- La segunda para buscar el autor del artículo cuando lo pedimos para el dict que va en lo que retorna la vista.

Rendimiento y ORMs

```
def my_view(request, article_id):
    article = Article.objects.get(article_id)
                    .select_related("author")

    return render(
        request,
        "my_view.html",
        {"article": article, "author": article.author}
    )
```

Podemos hacer las consultas juntas (con un join) usando select_related en Django.

Cuando queremos traer varias instancias de modelos relacionados podemos usar prefetch_related. Se usa para relaciones N-N.

Rendimiento y ORMs

No podemos optimizar nuestras aplicaciones si no sabemos dónde funcionan lento.

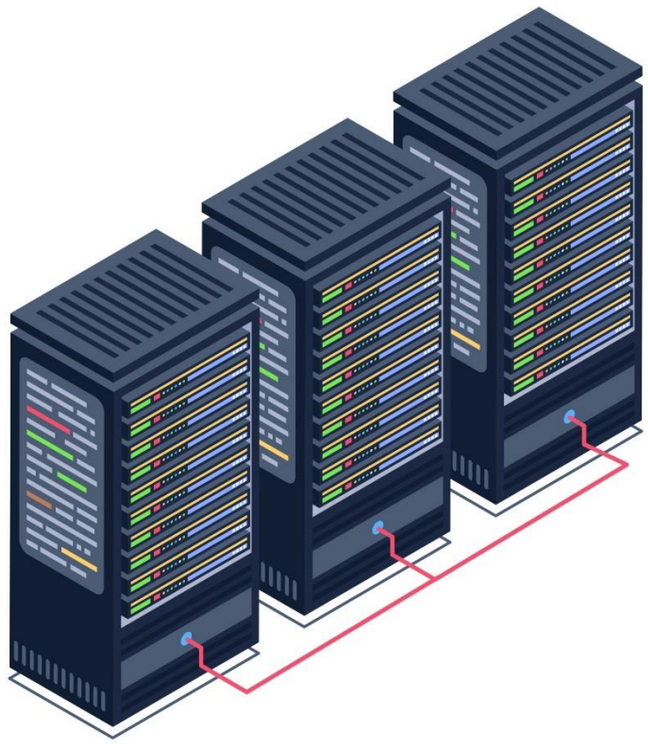
Por lo que necesitamos herramientas que nos muestren esa información.

En los entornos de desarrollo en Django se usa django-debug-toolbar, para Ruby on Rails existe Rails Mini Profiler.

En producción generalmente queremos más información por lo que integramos herramientas de monitoreo (APM) como Scout o NewRelic (las cuales verán en cursos posteriores).

Rendimiento y ORMs

Ya vimos que cuando queremos aproximar el tiempo que va a tomar una consulta revisamos cuantas páginas del disco se leen.



Para efectos de lecturas a una base de datos a través de una red, la cantidad de consultas que hacemos suele impactar más que la eficiencia de las consultas.

Recordatorio: ¡Eso no quita que tenemos que hacer las consultas bien!

NoSQL

Hasta ahora hemos visto…

- Bases de datos relacionales
- Algebra relacional
- SQL
- ORM
- Frameworks
- Índices / Complejidad
- Manejo de errores

La mayoría de estos temas asumen una **base de datos relacional**

Bases de datos relacionales

- Mucha estructura (un esquema fijo)
- Muchas garantías (ACID)
- Generalmente centralizadas (viven en un servidor)

Pero existen más tipos de bases de datos…

Las **NoSQL**

NoSQL

Término común para denominar bases de datos con:

- Menos restricciones que el modelo relacional
- Menos esquema
- Menos garantías de consistencia
- Más adecuadas para la distribución

NoSQL: ¿Por qué?

Sistemas de bases de datos relacionales **no** están pensadas para un entorno altamente **distribuido**, o sea, que no pueden manejar grandes volúmenes de datos y tráfico dividiendo la carga entre múltiples servidores, por ejemplo:

- WWW, google, twitter, instagram, etc.

Sistemas distribuidos

Un **sistema distribuido** es un conjunto de computadoras físicamente separadas pero conectadas a través de una red centralizada. Estas computadoras autónomas se comunican para compartir recursos, archivos y realizar tareas asignadas.

Los sistemas distribuidos buscan resolver dos problemas fundamentales:

1. Datos no caben en un computador

2. Servidores pueden fallar

Datos no caben en un computador

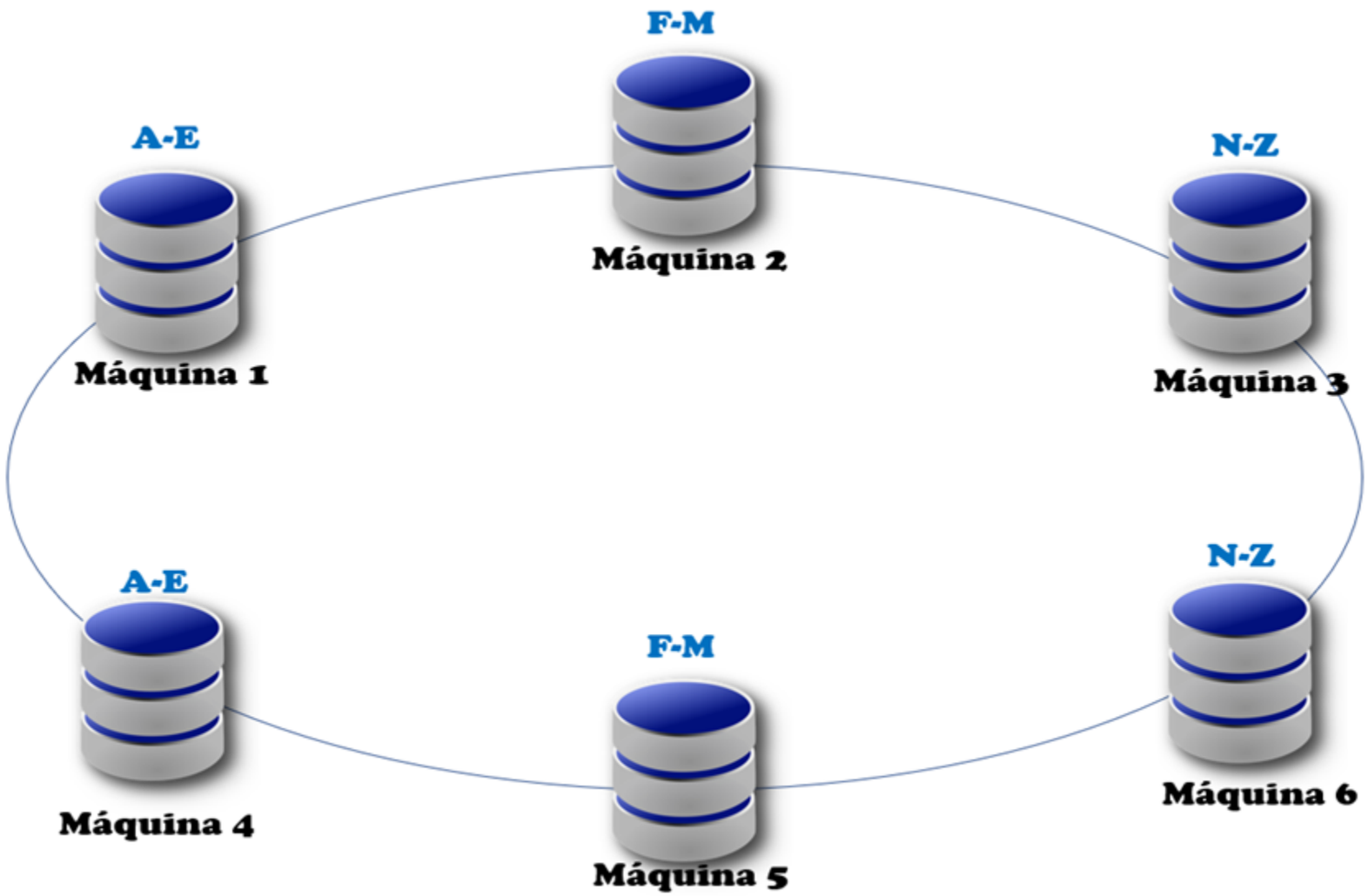
Fragmentación de los datos

Ej: **Usuarios** de twitter

Fragmentación de relación **Usuarios** en tres

Servidores fallan

Replicación de los datos



Replicación en un sistema distribuido

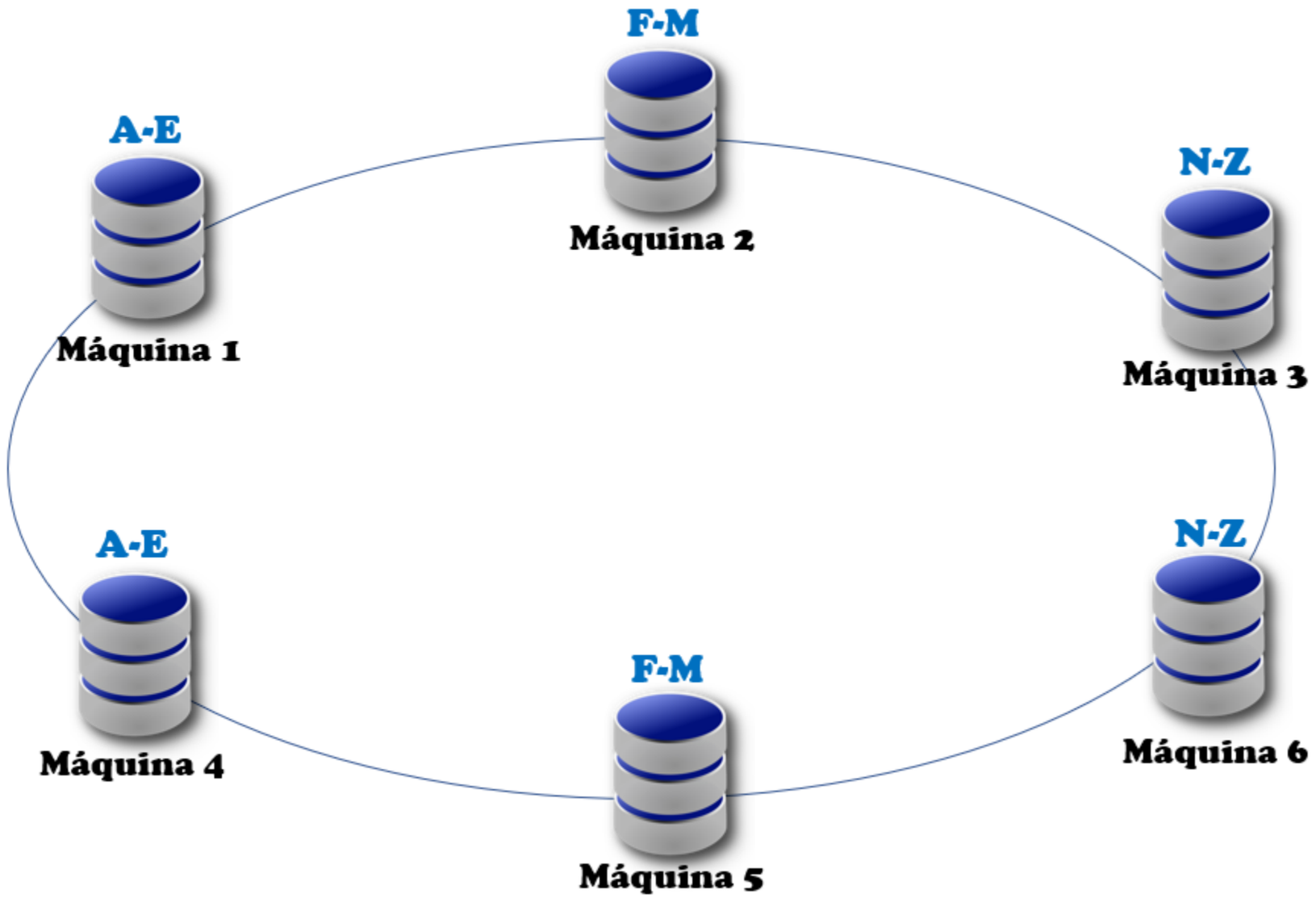
Garantías en un entorno distribuido

Los sistemas distribuidos tienen tres propiedades fundamentales:

- **Consistency** (todos los usuarios ven lo mismo)
- **Availability** (todas las consultas siempre reciben una respuesta, aunque sea errónea)
- **Partition tolerance** (el sistema funciona bien pese a estar físicamente dividido)

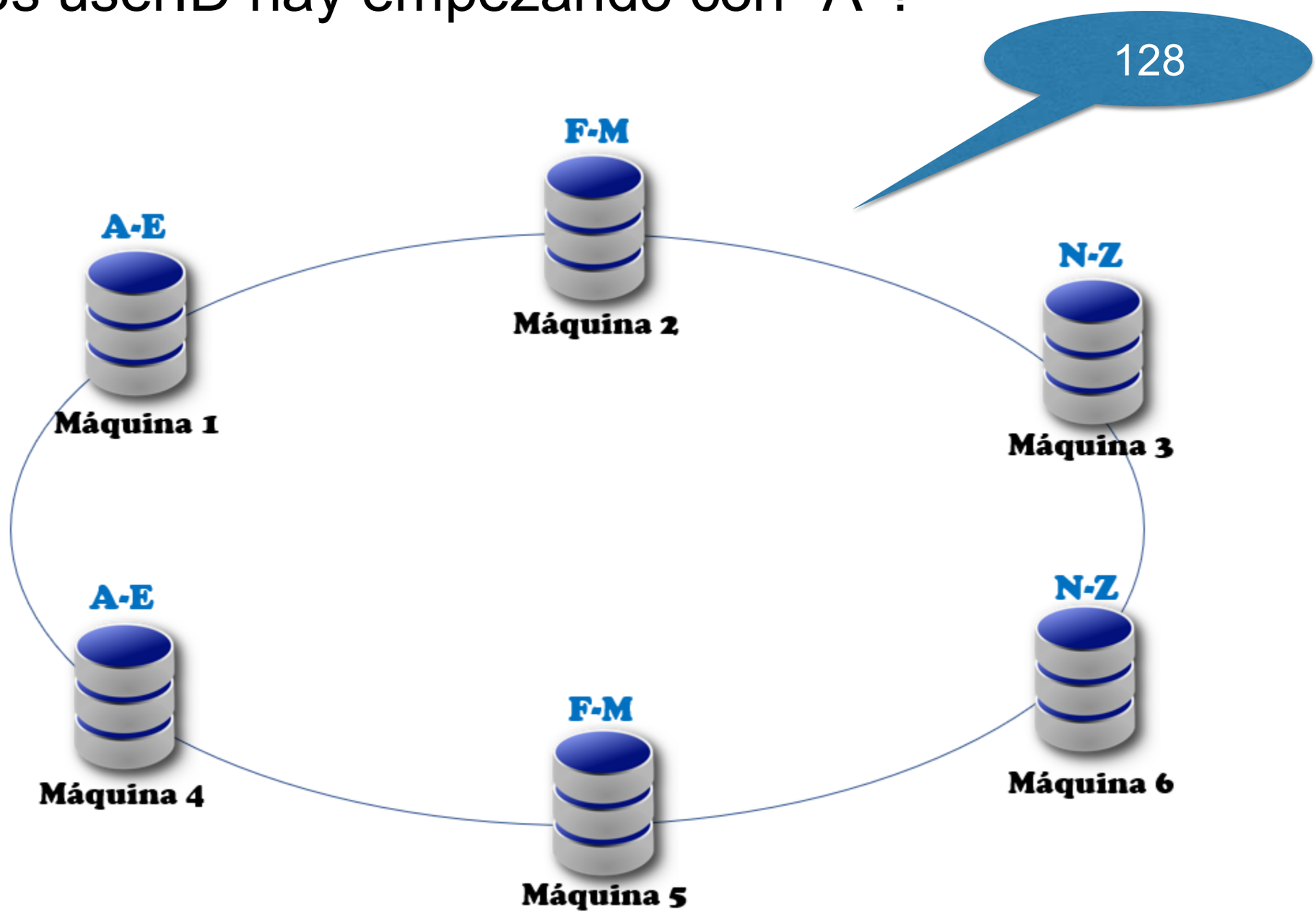
Consistencia

¿Cuántos userID hay empezando con "A"?



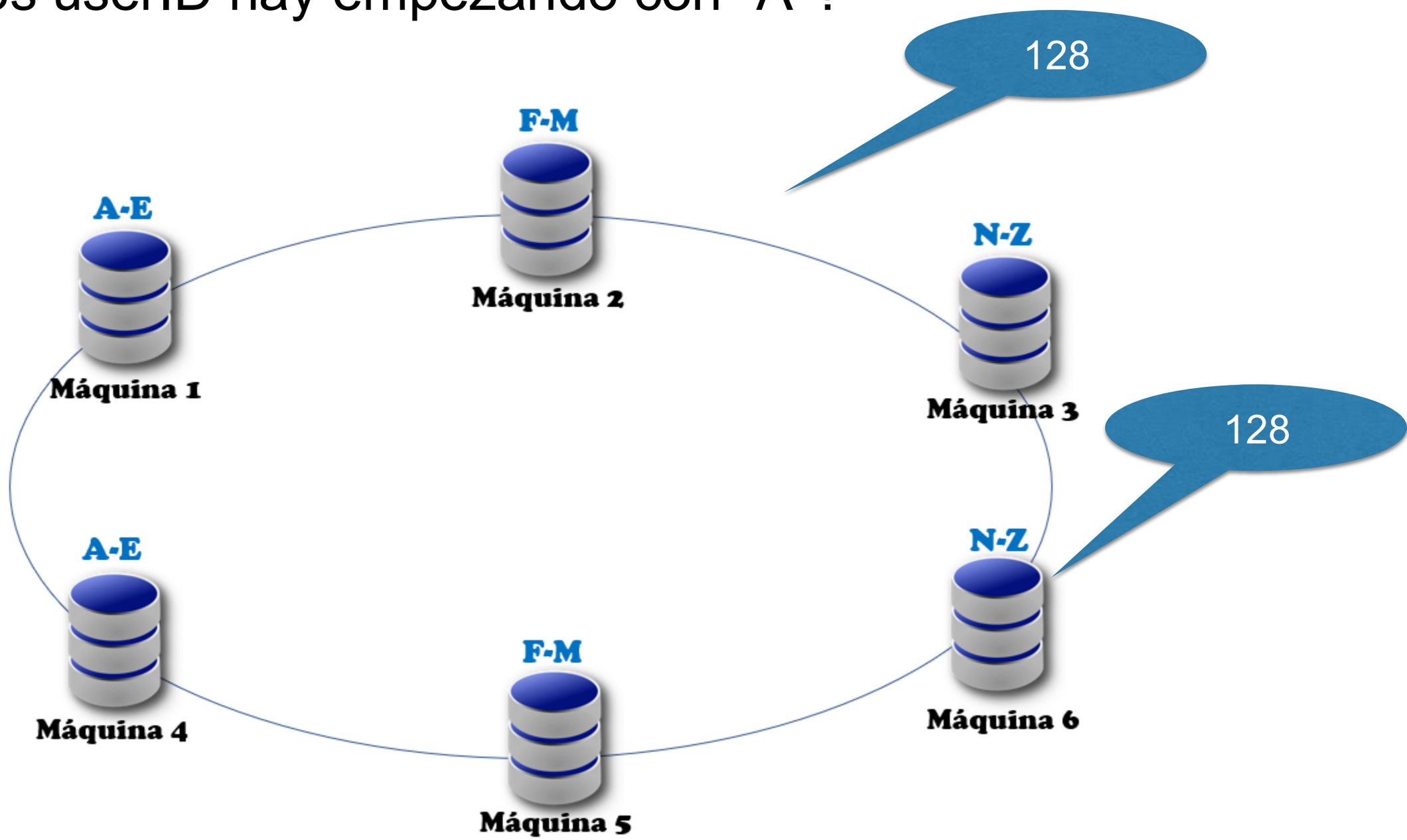
Consistencia

¿Cuántos userID hay empezando con "A"?



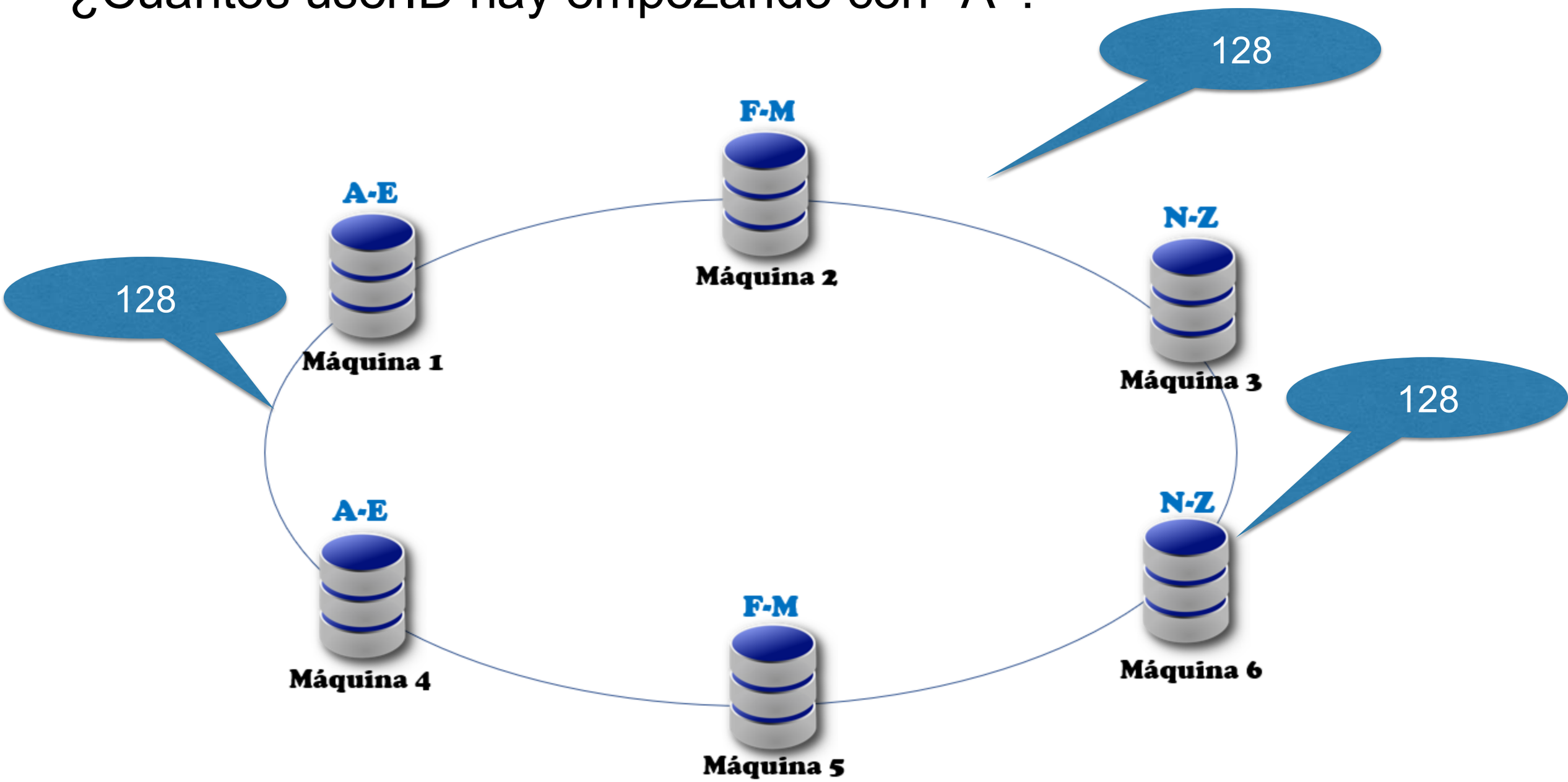
Consistencia

¿Cuántos userID hay empezando con "A"?



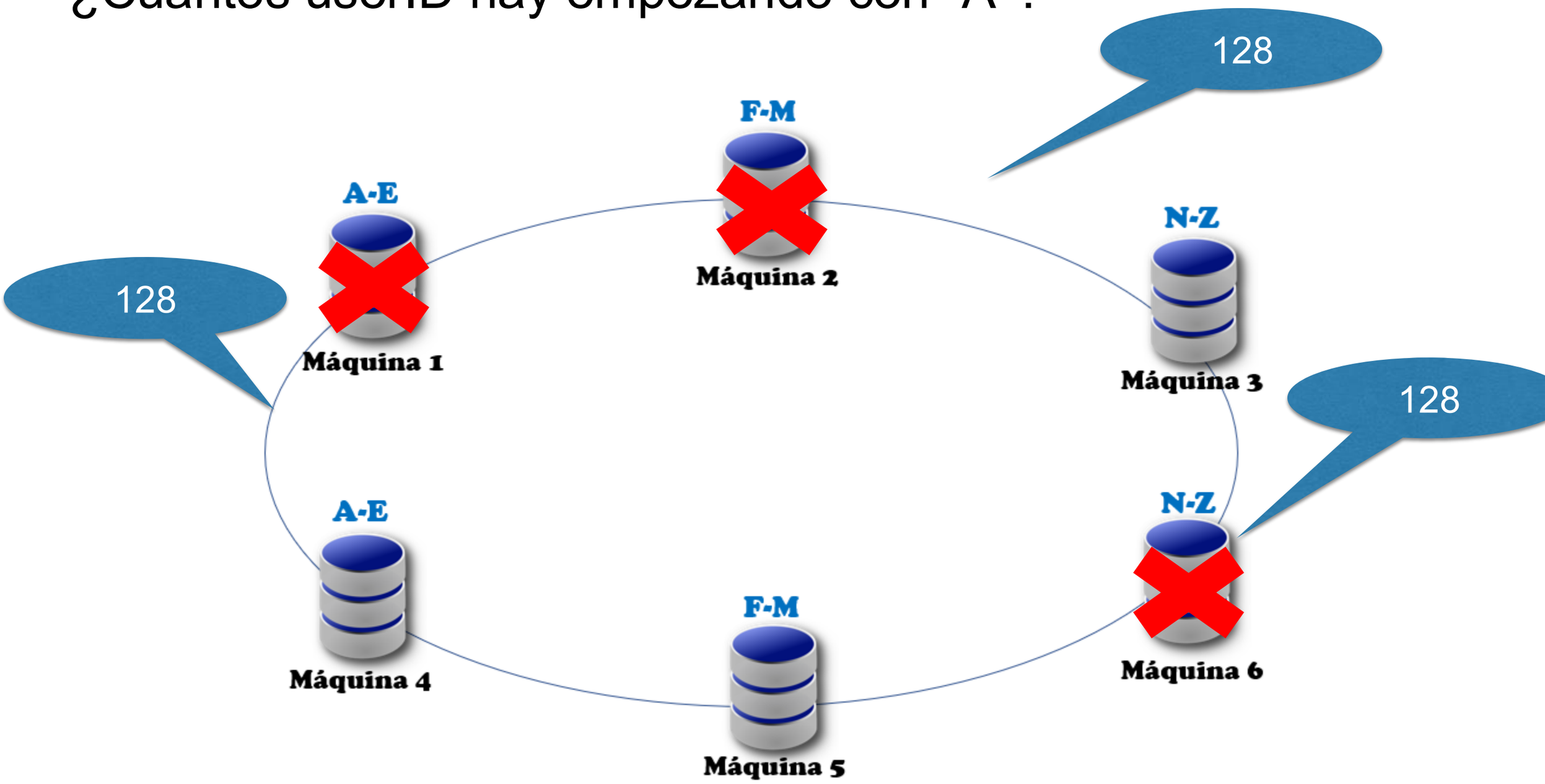
Consistencia

¿Cuántos userID hay empezando con "A"?



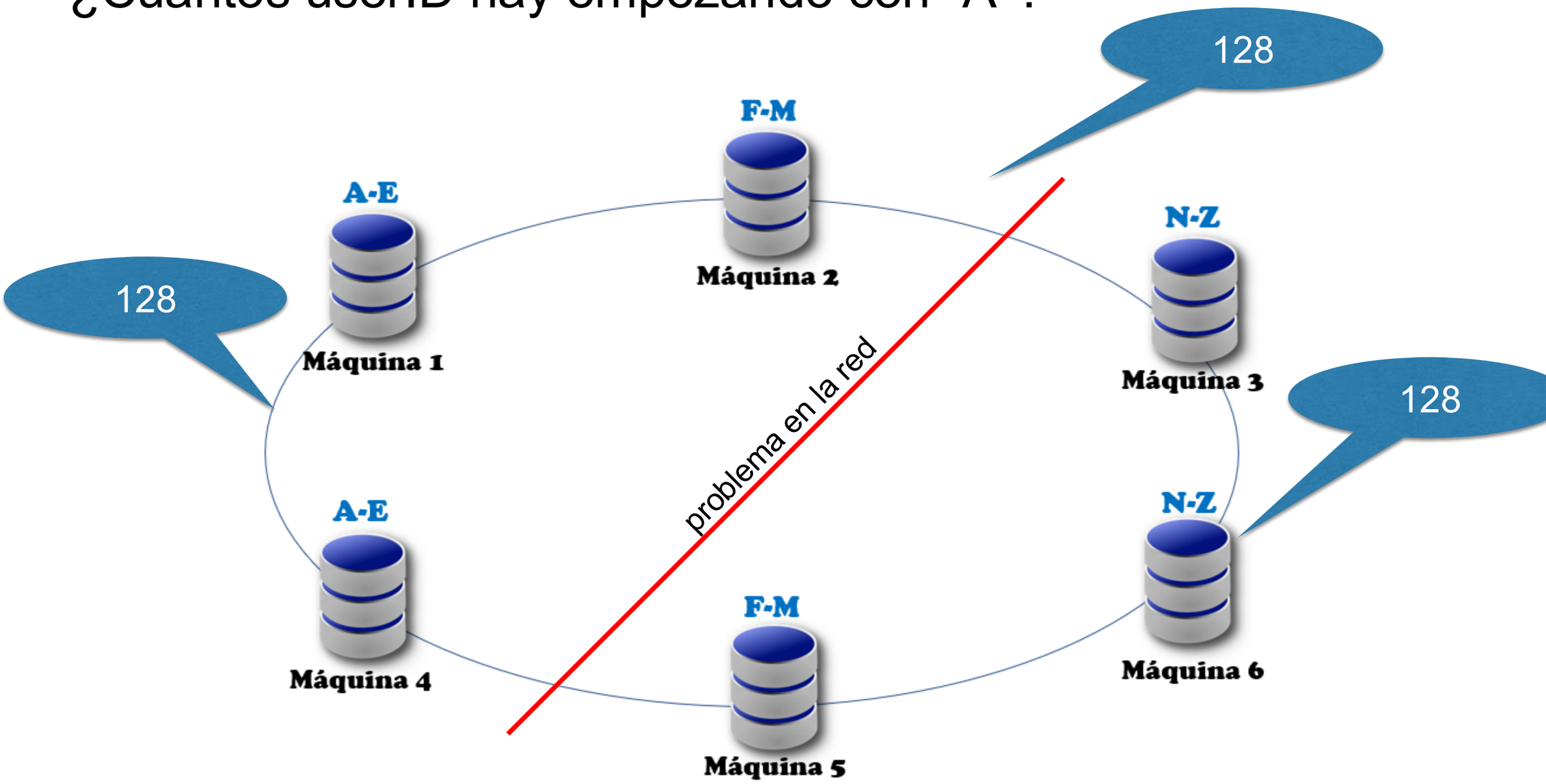
Consistencia

¿Cuántos userID hay empezando con "A"?



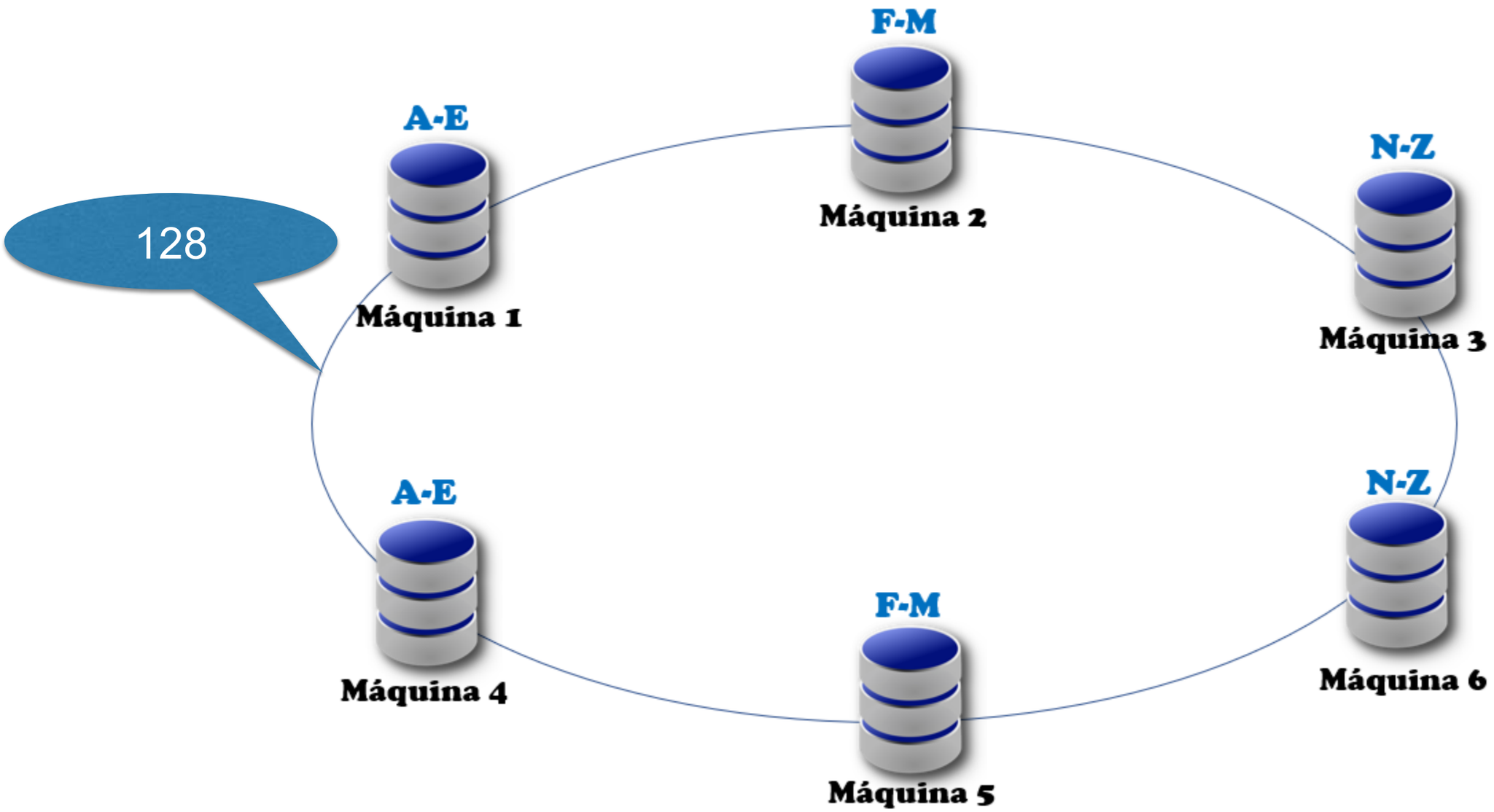
Consistencia

¿Cuántos userID hay empezando con "A"?



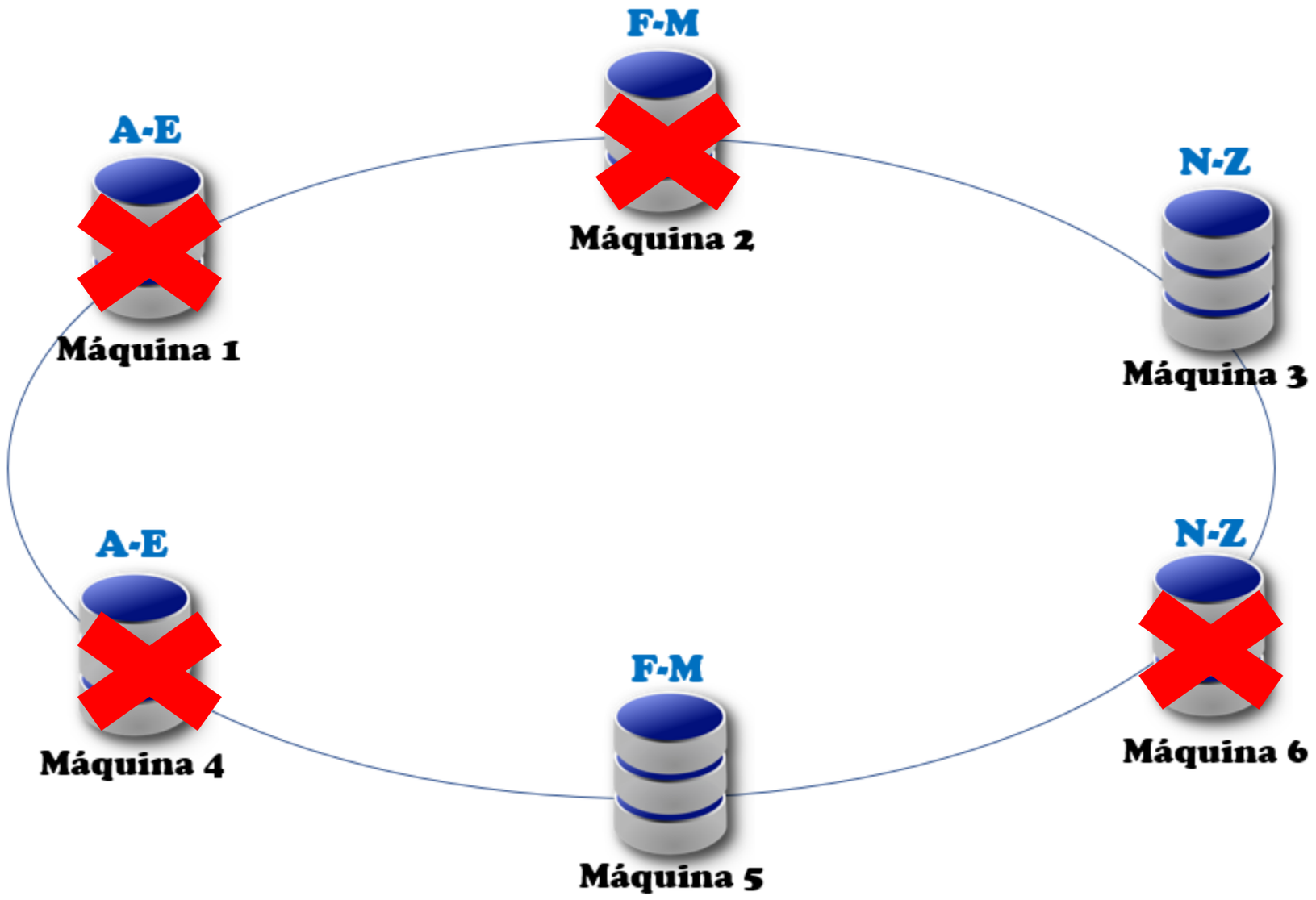
Availability

¿Cuántos userID hay empezando con "A"?



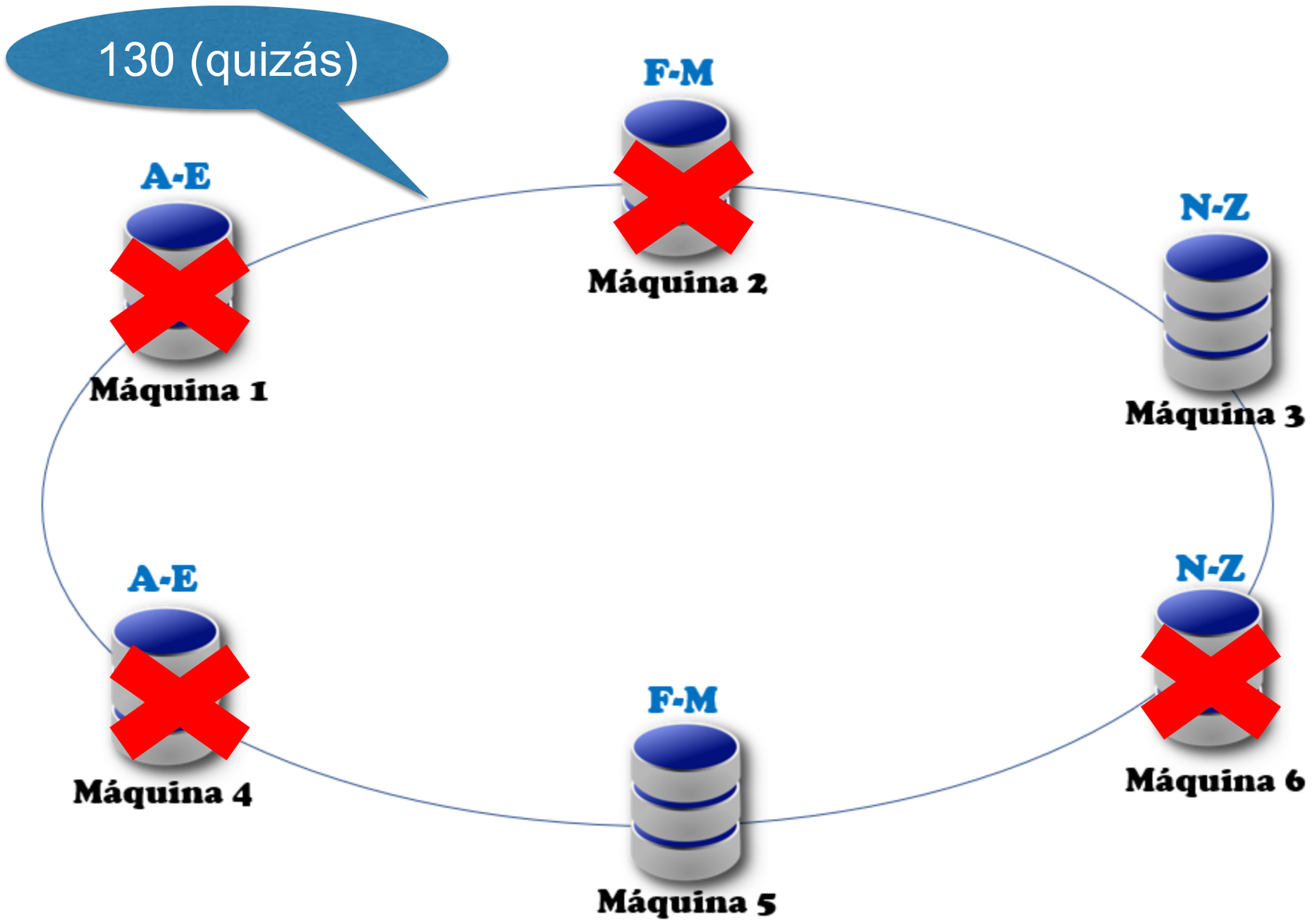
Availability

¿Cuántos userID hay empezando con "A"?



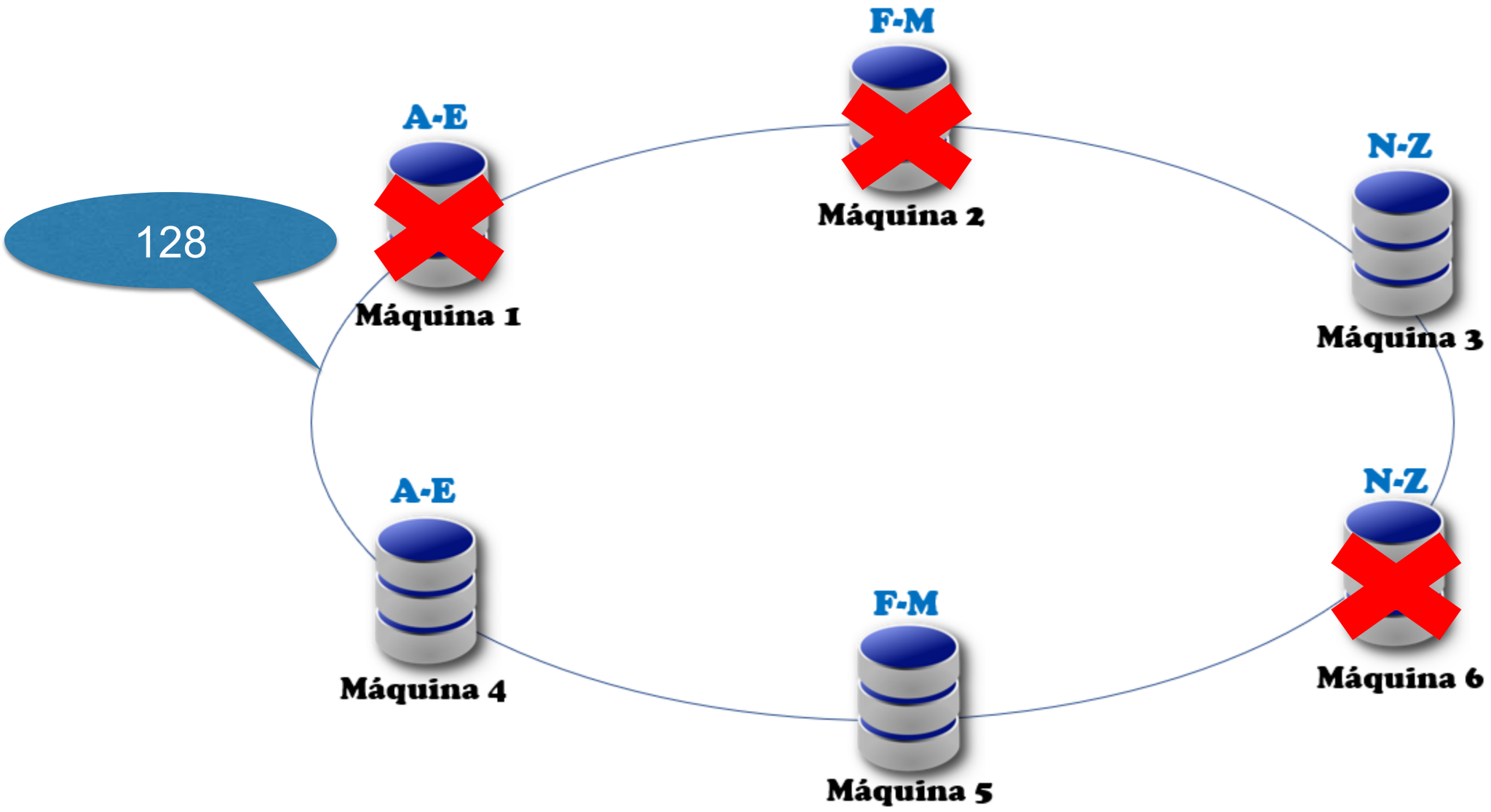
Availability

¿Cuántos userID hay empezando con "A"?



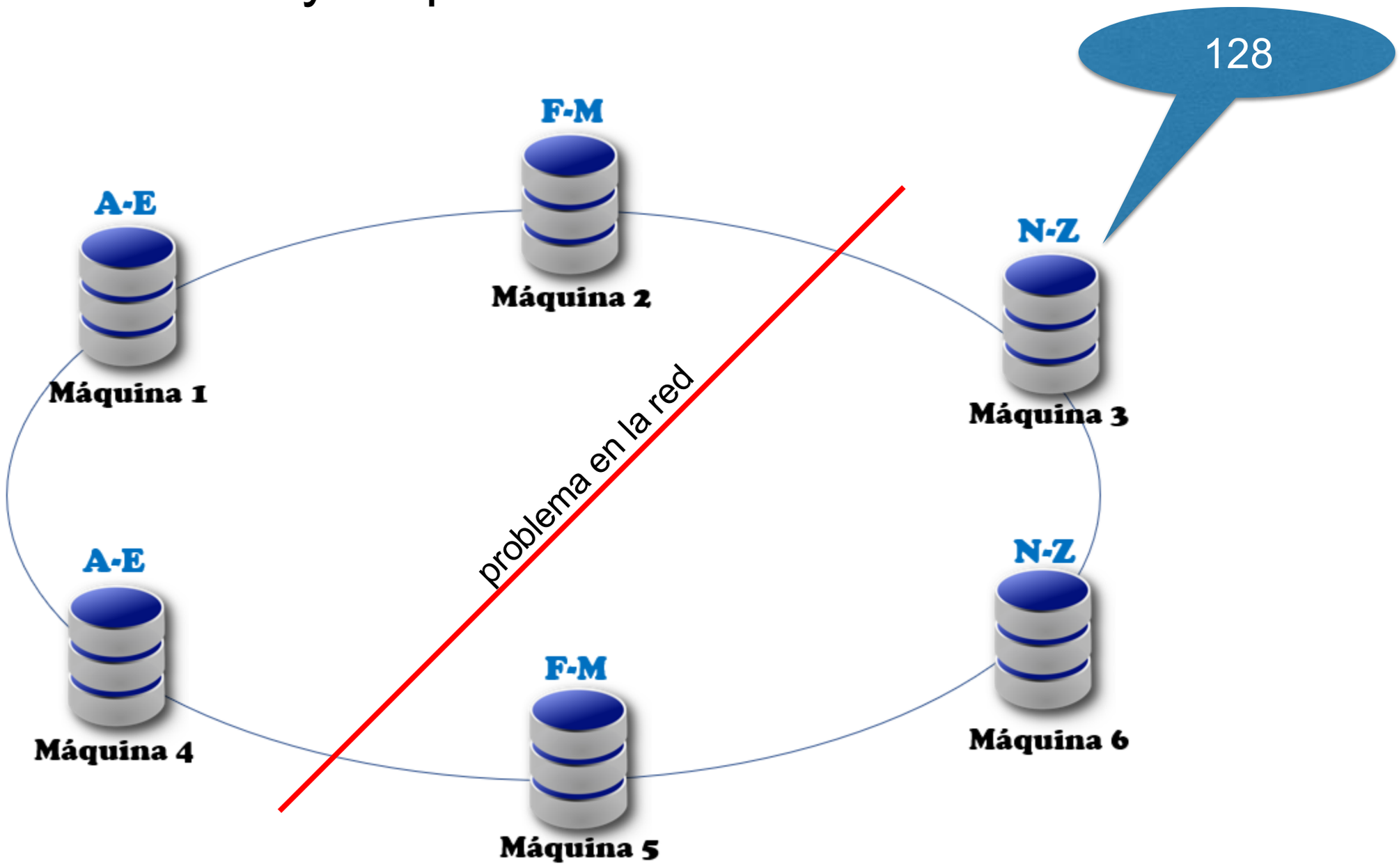
Partition tolerance

¿Cuántos userID hay empezando con "A"?



Partition tolerance

¿Cuántos userID hay empezando con "A"?



Teorema CAP

Plantea que para una base de datos distribuida es imposible mantener simultáneamente estas tres características:

- **Consistency**
- **Availablity**
- **Partition tolerance**

Teorema CAP

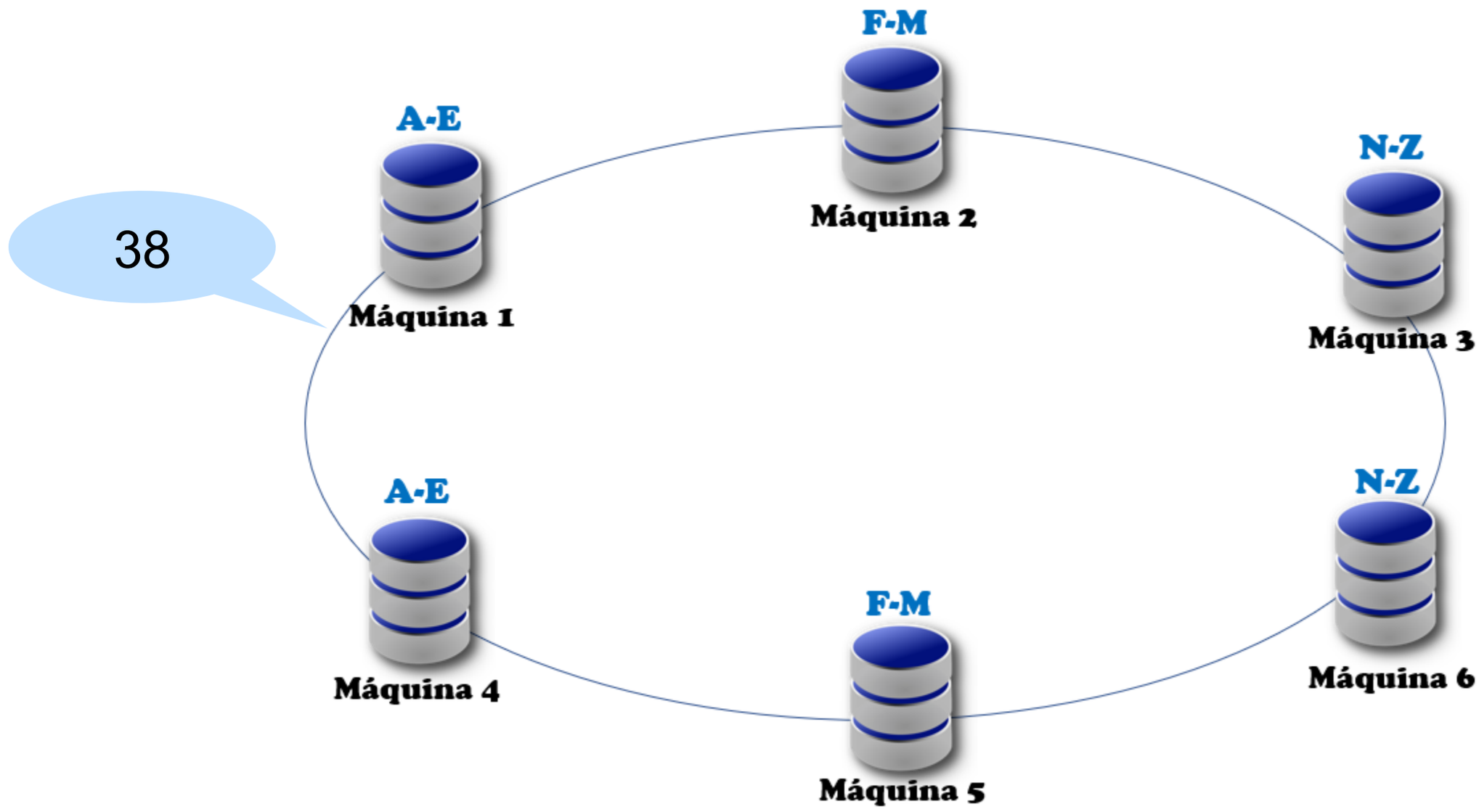
P es dado en cualquier sistema distribuido. Entonces, el Teorema CAP nos dice que hay que elegir entre:

- Consistency
- Availablity

Entonces tenemos sistemas CP y AP

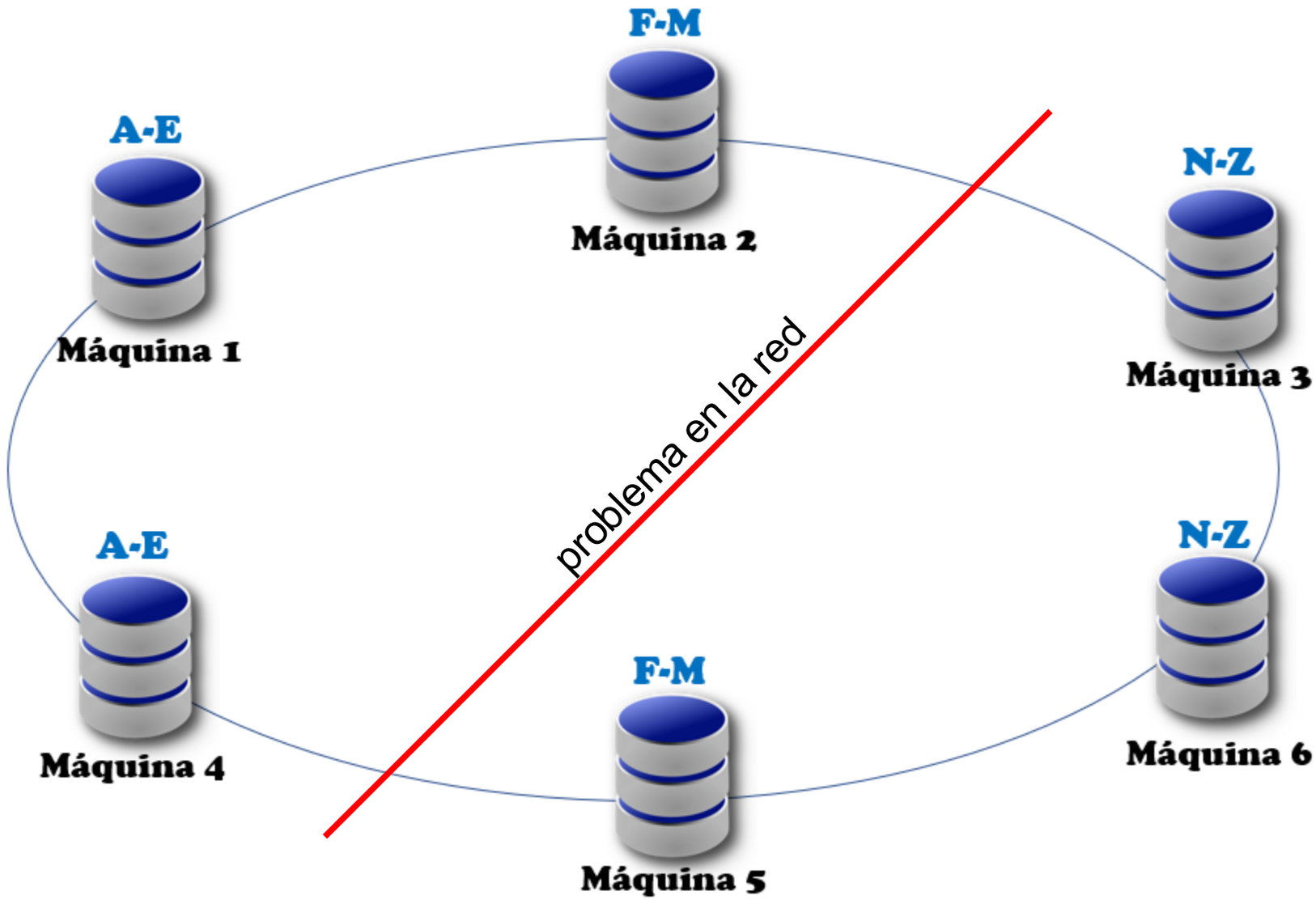
AP vs CP

¿Cuántos userID hay empezando con "Z"?



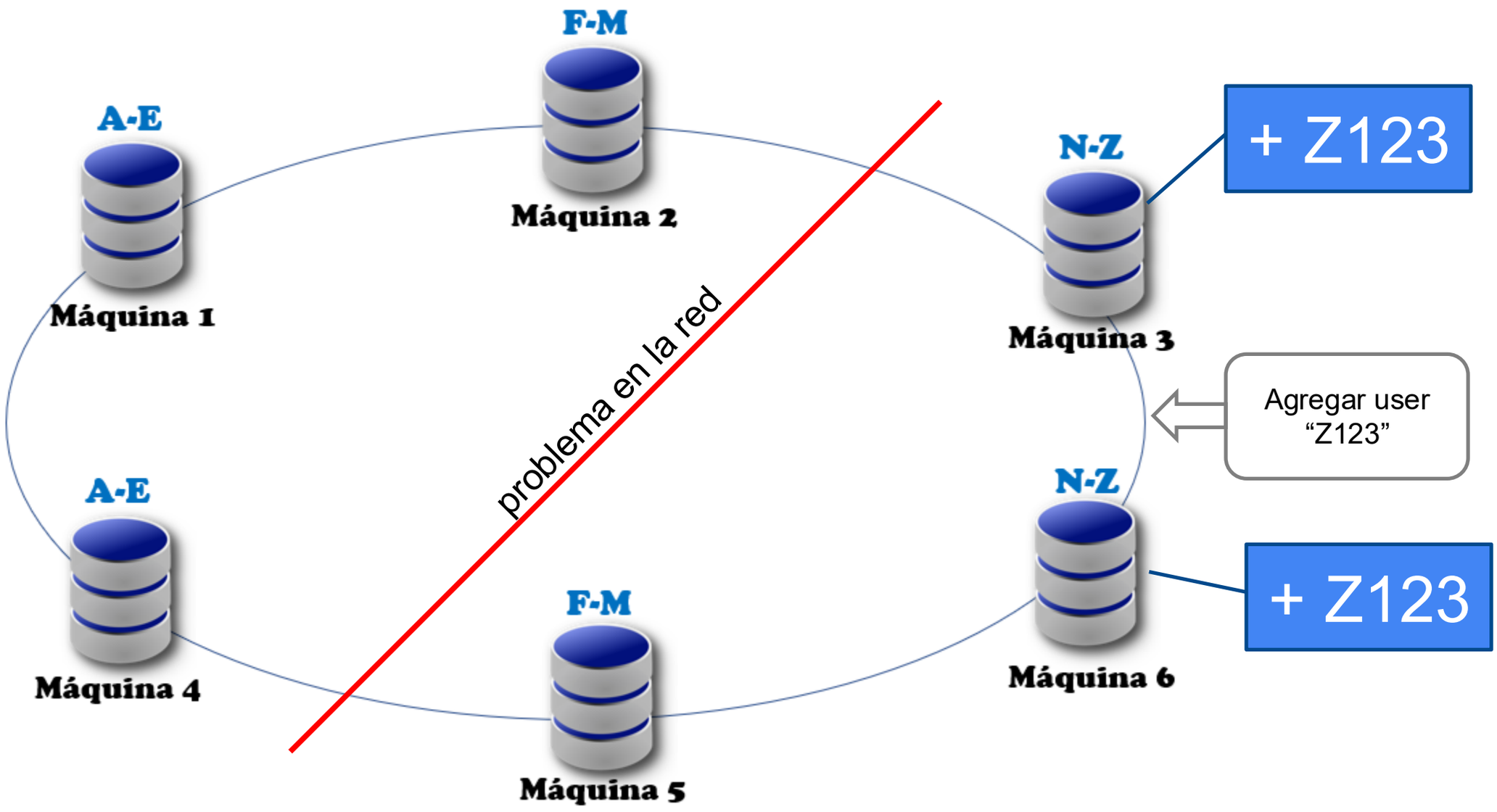
AP vs CP

¿Cuántos userID hay empezando con "Z"?



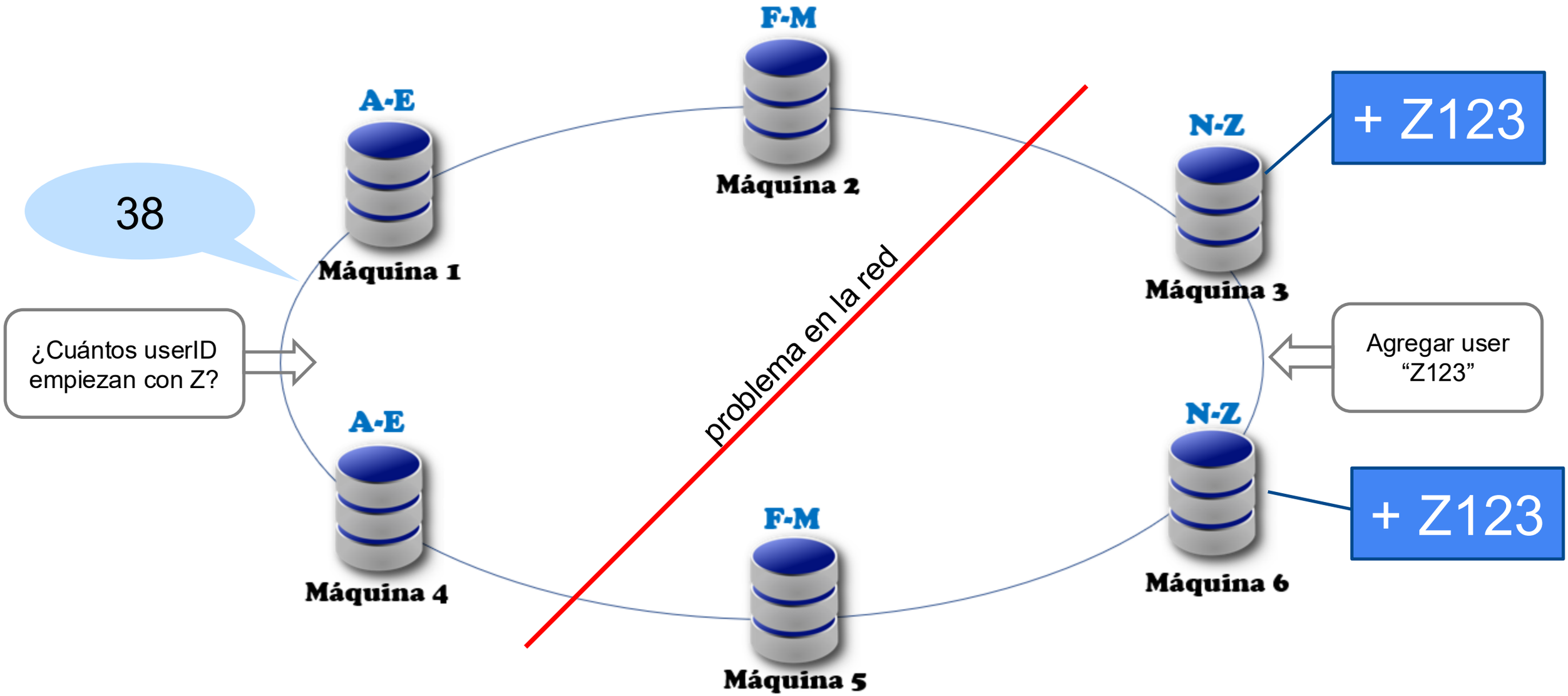
AP vs CP

¿Cuántos userID hay empezando con "Z"?



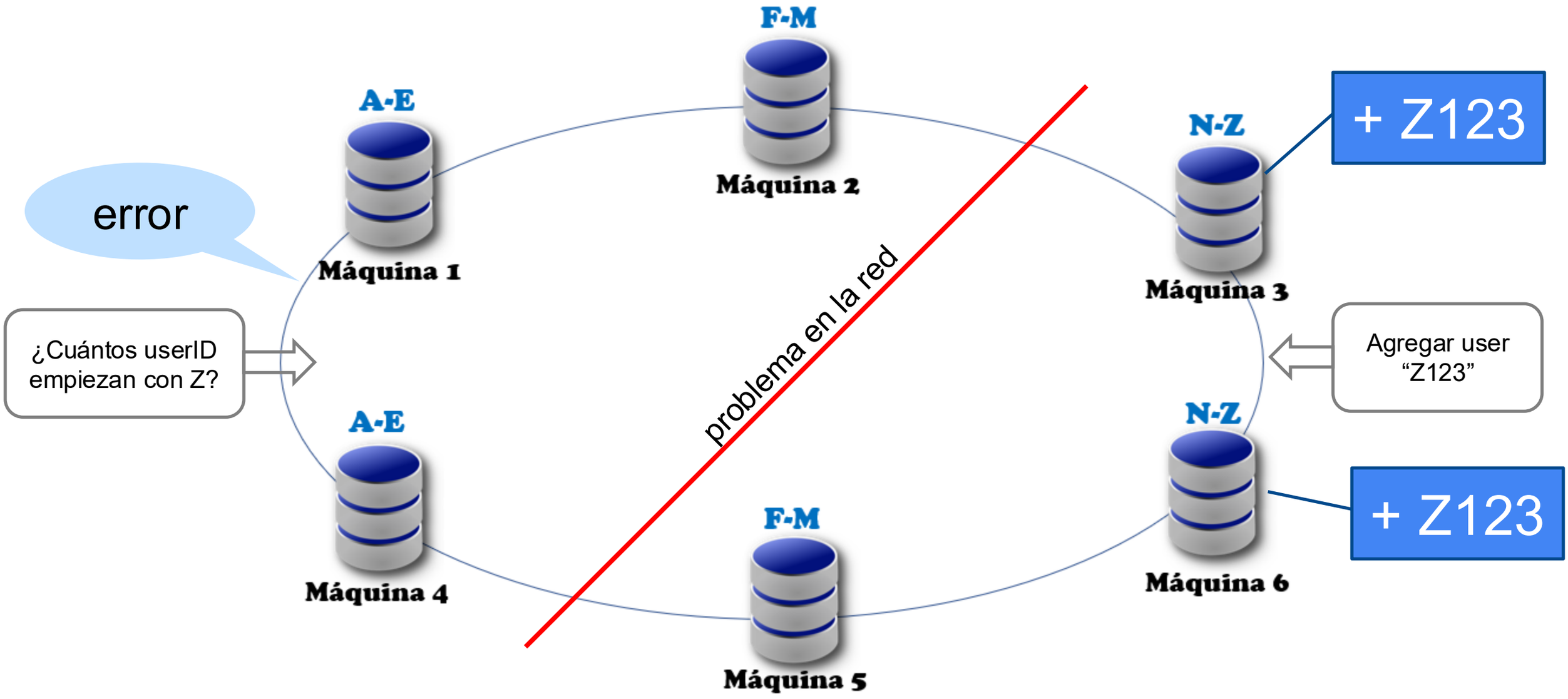
En un sistema AP

¿Cuántos userID hay empezando con "Z"?



En un sistema CP

¿Cuántos userID hay empezando con "Z"?



BASE

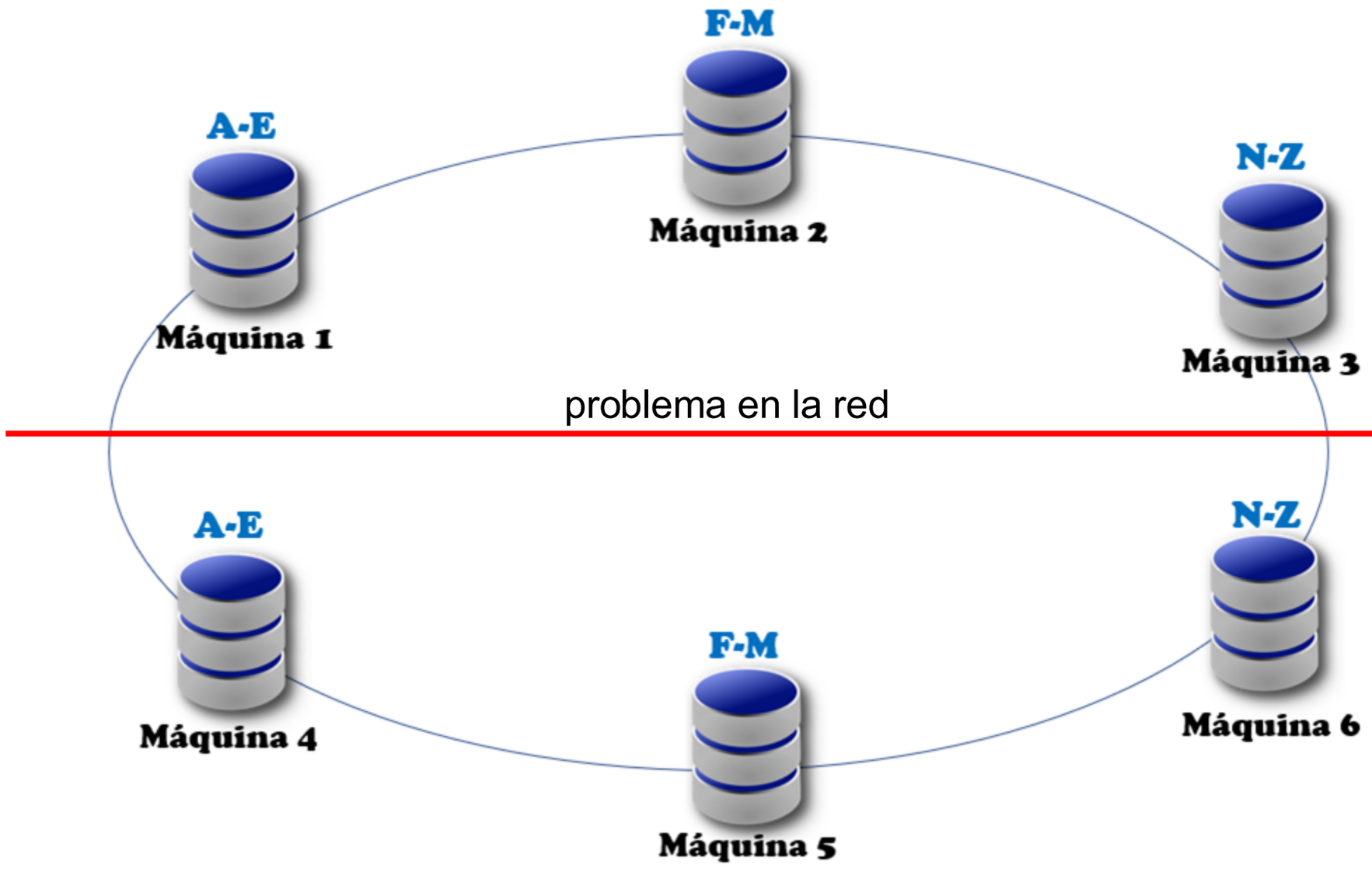
En la práctica, sistemas distribuidos fijan el P, y balancean entre C y A, sin elegir uno exclusivamente. Pero se tiene el paradigma BASE:

Basically Available: El sistema siempre está disponible para responder a las solicitudes, aunque algunas respuestas pueden no ser completas o recientes.

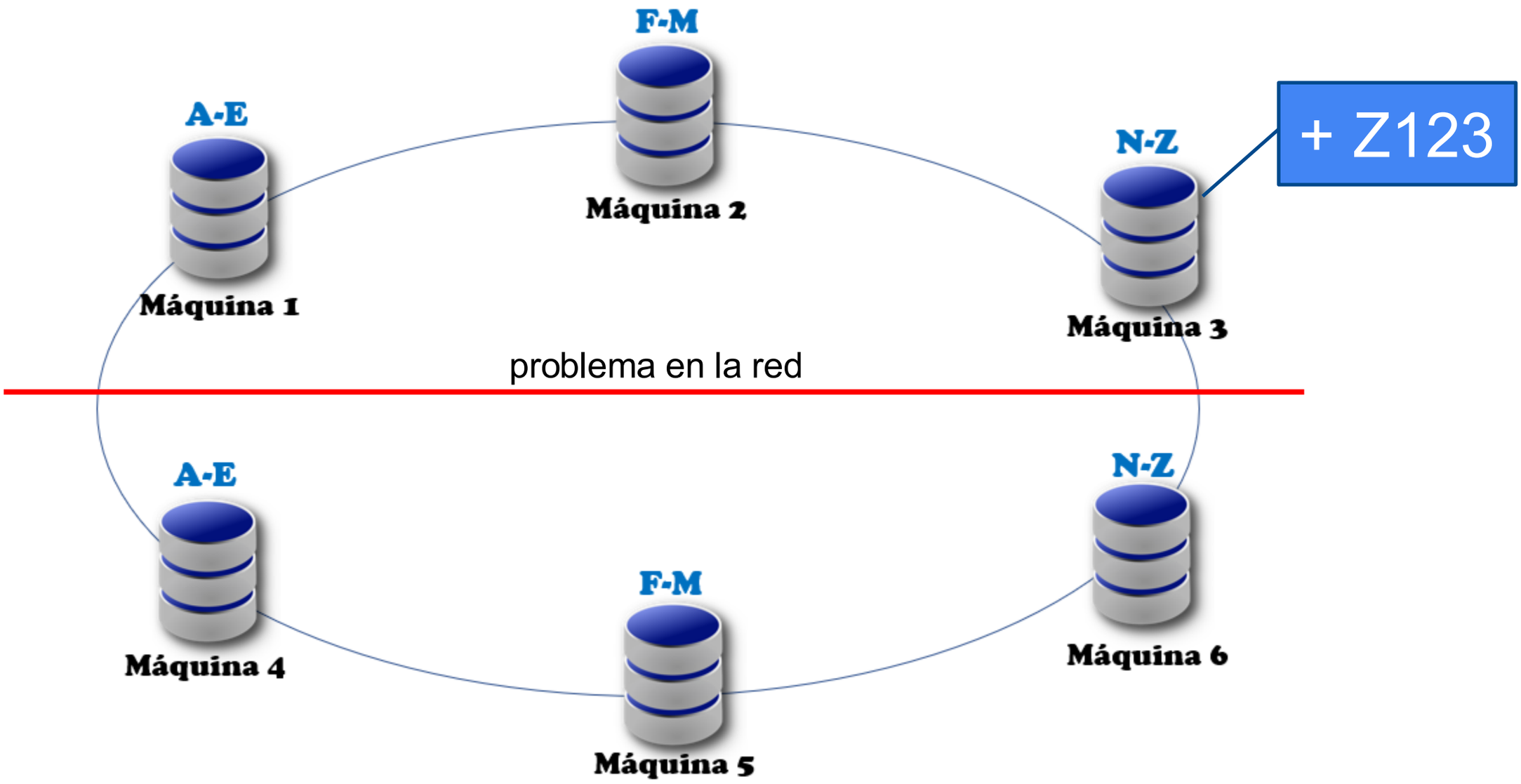
Soft State: El estado del sistema puede cambiar con el tiempo sin necesidad de recibir nuevas entradas, reflejando una actualización asincrónica y flexible.

Eventually Consistent: Con el tiempo, todos los nodos del sistema llegarán a un estado consistente, permitiendo alta disponibilidad y tolerancia a fallos, a costa de no tener consistencia inmediata.

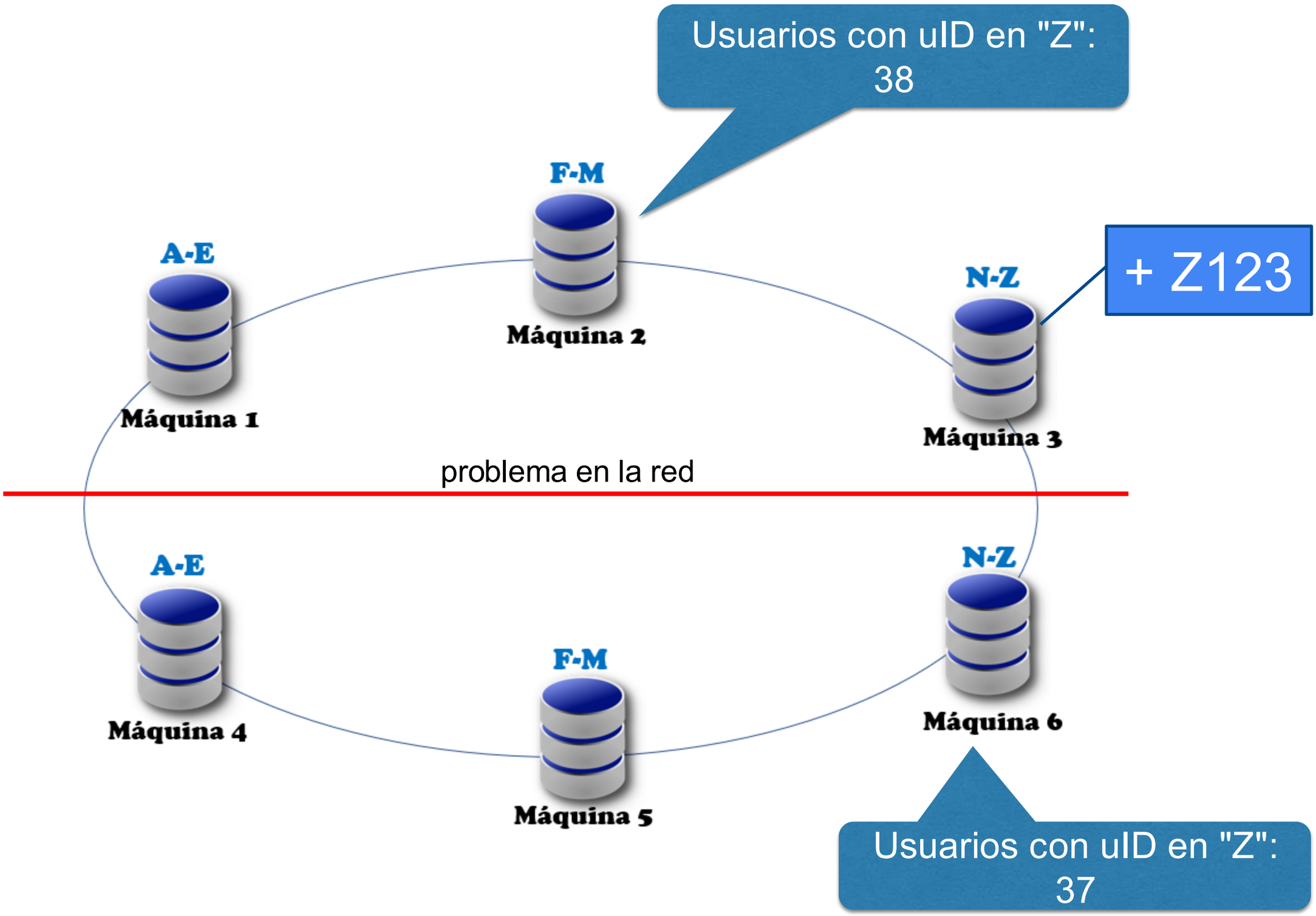
Consistencia eventual



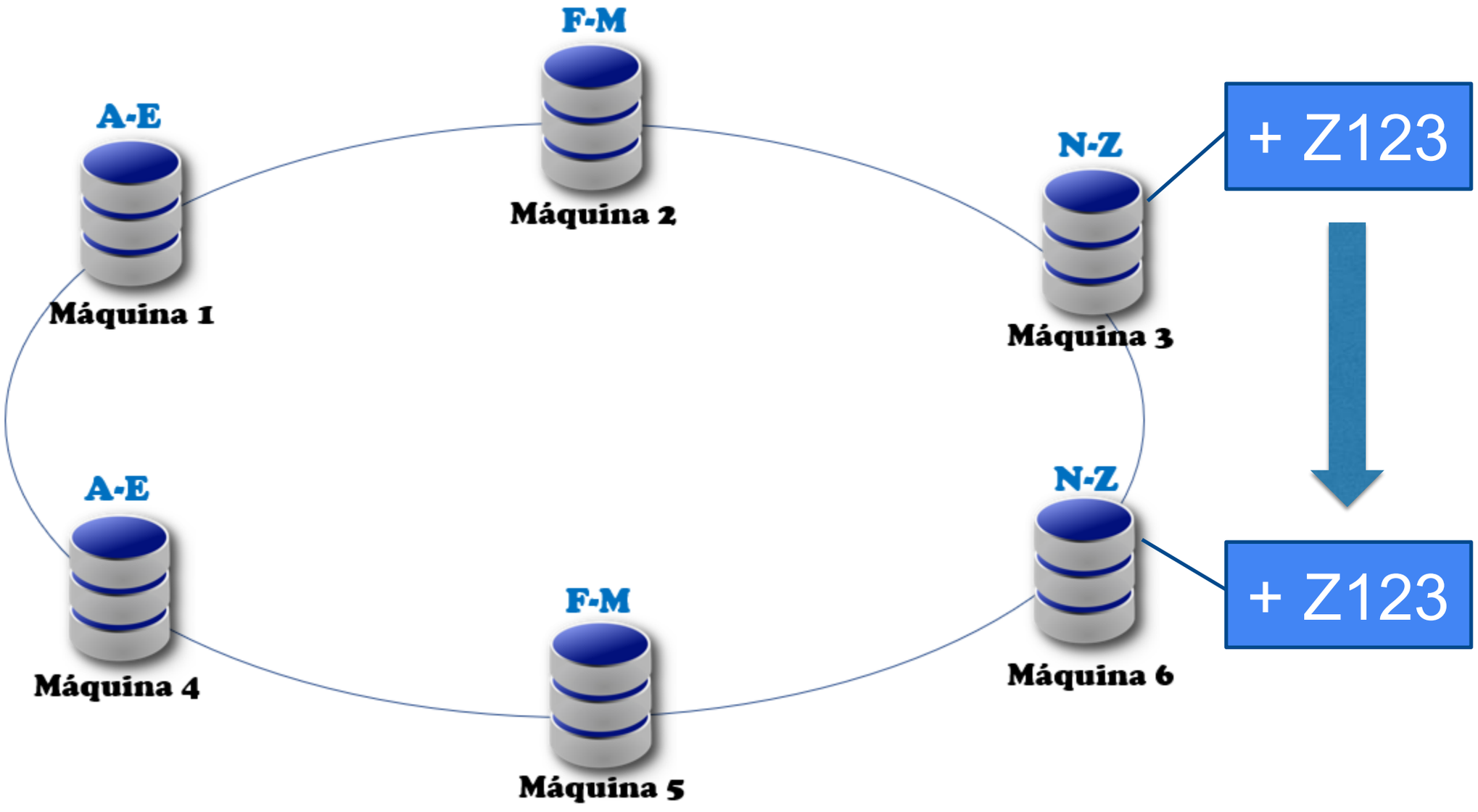
Consistencia eventual



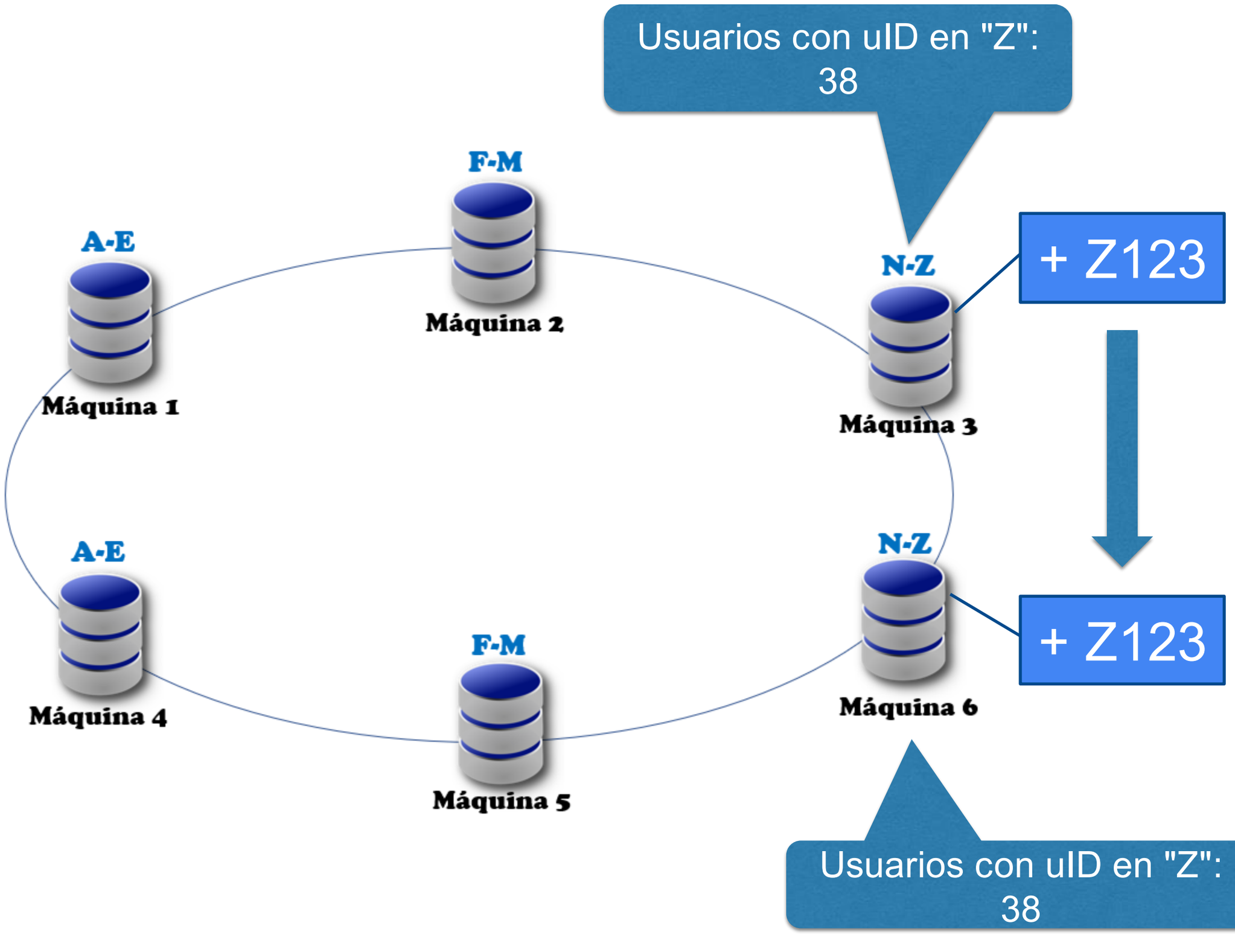
Consistencia eventual



Consistencia eventual



Consistencia eventual

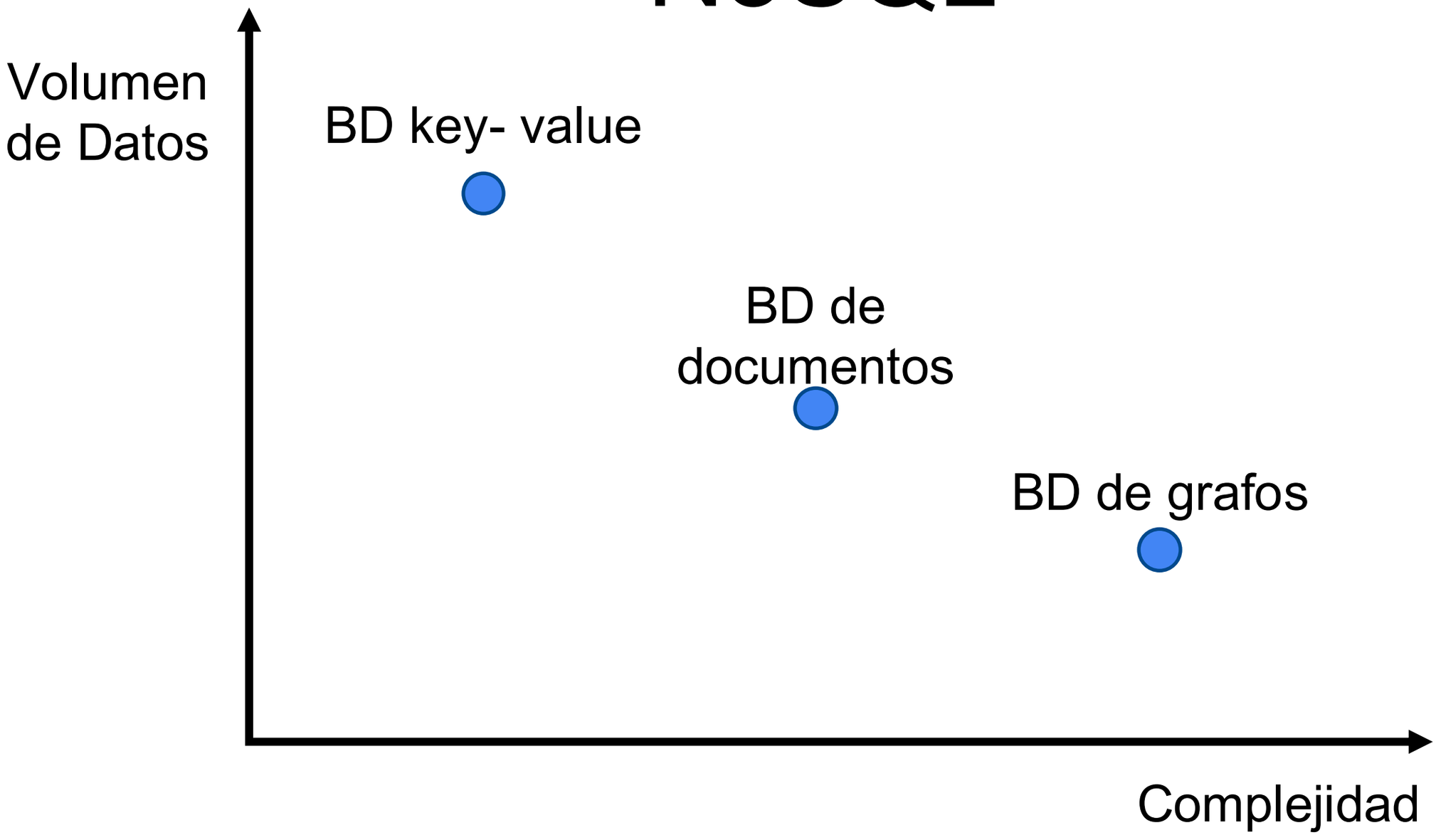


Sabores de NoSQL

Los más populares hoy en día:

- **BD key-value**
- **BD de grafos**
- **BD de documentos**

Sabores de NoSQL



BD Key - Value

Independientemente del esquema

- Arquitectura almacena información por medio de pares
- Cada par tiene una llave (identificador) y un valor

BD Key - Value

Operaciones cruciales:

- put(key,value)
- get(key)
- delete(key)

Key	Value
Chile	Santiago
Inglaterra	Londres
Escocia	Edinburgo
Francia	Paris
Alemania	Berlin
…	…

BD Key - Value

- Son grandes tablas de hash persistentes
- Esta categoría es difusa, pues muchas de las aplicaciones de otros tipos de BD usan key - value y hashing hasta cierto punto

Ejemplo más importante: Amazon Dynamo, otro es Redis

BD Key - Value

Puede representar cualquier valor

cartID	value
11789	usrID: "Juan", ítem: "Magic the Gathering Deck", value: …
12309	usrID: "Domagoj", item: "APEX XTX50 regulator set", value: …
…	…

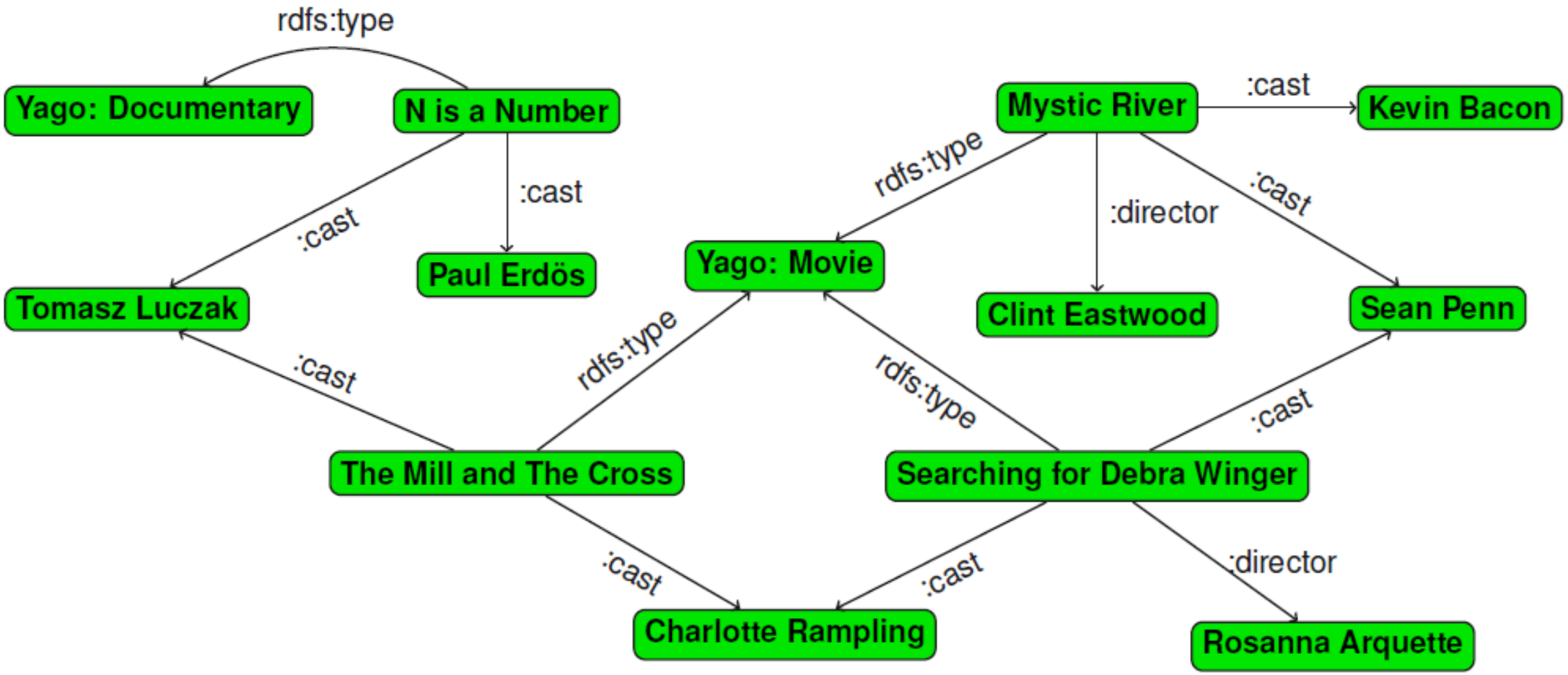
BD de Grafos y RDF

(Resource Description Framework)

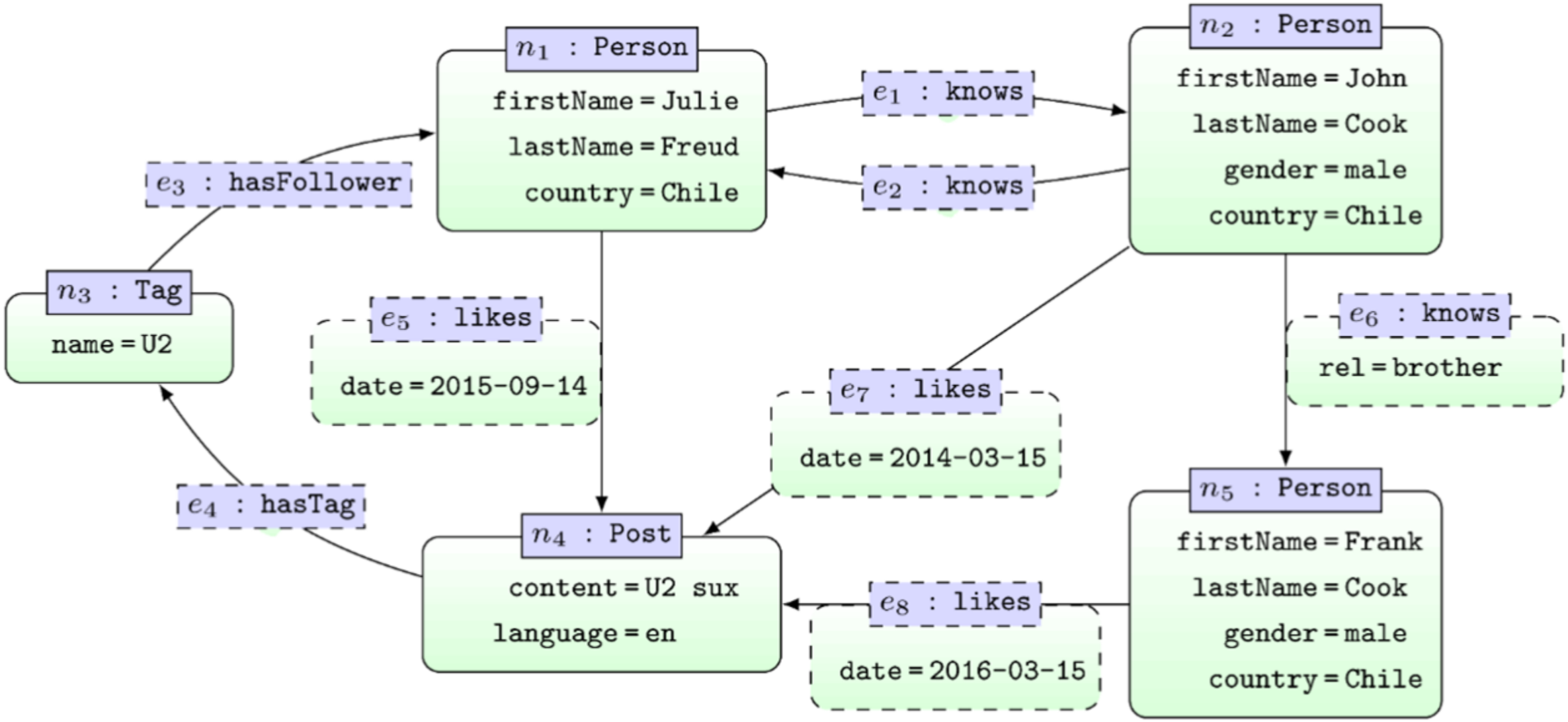
Especializadas para guardar relaciones

- En general, almacenan sus datos como property graphs
- Algunos ejemplos son Neo4J, Virtuoso, Jena, Blazegraph

BD de Grafos y RDF



BD de Grafos y RDF



BD Orientadas a Documentos

Especializadas en documentos

- CouchDB, **MongoDB** (estas y otras BD almacenan sus datos en documentos JSON)
- JSON no es el único estándar de documentos (por ejemplo, existe también XML)

Ahora nos centraremos en **BD Orientadas a documentos**, específicamente en **MongoDB**

BD Orientadas a Documentos

Parecidas a key-value stores:

- El valor es ahora un documento (JSON)
- Pueden agrupar documentos (colecciones)
- Lenguaje de consultas mucho más poderoso

BD Orientadas a Documentos

Usuarios	
Key	JSON
1	
2	…
…	…

JSON

Su nombre viene de JavaScript Object Notation

Estándar de intercambio de datos semiestructurados / datos en la Web

- JSON se acopla muy bien a los lenguajes de programación

JSON - Ejemplo


```
{
  "statuses": [
    {
      "id": 725459373623906304,
      "text": "@visitlondon: Have you been to any of these
               quirky London museums? https://t.co/tnrar8UttZ",
      "retweeted_status": {
        "metadata": {
          "result_type": "recent",
          "iso_language_code": "en"
        },
        "retweet_count": 239,
        "retweeted": false
      }
    }
  ]
}
```

JSON

La base son los pares key - value

```
{
  “nombre": “Matías”, “apellido": “Jünemann”
}
```

Valores pueden ser:

- Números
- Strings (entre comillas)
- Valores booleanos
- Arreglos (por definir)
- Objetos (por definir)
- null

JSON - Sintaxis

Los objetos se escriben entre {} y contienen una cantidad arbitraria de pares key - value

```
{
  “nombre": “Matías”, “apellido": “Jünemann”
}
```

JSON - Sintaxis

Los arreglos se escriben entre [] y contienen valores

```
{
  “profesores”: [
    {“nombre”: “Juan”, “apellido": “Reutter”},
    {“nombre”: “Cristian”, “apellido": “Riveros”},
    {“nombre”: “Marcelo”, “apellido": “Arenas”}
  ]
}
```

JSON vs SQL

SQL:

- Esquema de datos
- Lenguajes de consulta independientes del código

JSON:

- Más flexible, no hay que respetar necesariamente un esquema
- Más tipos de datos (como arreglos)
- Human - Readable

BD de documentos: ¿para qué?

Especializadas en documentos: almacenan muchos documentos JSON

- Si quiero libros: un documento JSON por libro
- Si quiero personas: un documento JSON por persona

Notar que esto es altamente jerárquico

BD de documentos

Qué hacen bien:

- Si quiero un libro o persona en particular
- Cruce de información **simple**

Muy útiles a la hora de desplegar información en la web

Además, verse como un **cache** de una pueden BD relacional

Caché de BD SQL

Students

StudentID	Nombre	Carrera
1	Alice Cooper	Computación
2	David Bowie	Todas
3	Charly García	Ingeniería Civil
…	…	…

Courses

courseID	name	year
IIC2413	Databases	2020
IMT3830	Game Theory	2020
…	…	…

Takes

courseID	StudentID
IIC2413	1
IIC2413	2
IMT3830	2
…	…

Caché de BD SQL

Lista de alumnos por curso:

- SQL tiene que hacer un join
- En BD documentos prepararemos esta información

Caché de BD SQL

Colección "**Courses**"

```
{
    "courseID": IIC2413,
    "name": "Databases",
    "year": 2020,
    "students":[
        {
            "studentID": 1,
            "name": "Alice Cooper"
        },
        {
            "studentID": 2,
            "name": "David Bowie"
        },
        ...
    ]
}
```

```
{
    "courseID": IMT3830,
    "name": "Game Theory",
    "year": 2020,
    "students":[
        {
            "studentID": 2,
            "name": "David Bowie"
        },
        {
            "studentID": 3,
            "name": "Charly García"
        },
        ...
    ]
}
```

BD de documentos

Qué hacen bien en este caso:

- Si quiero lista de alumnos de un curso
- Si quiero nombres de todos los cursos
- Si quiero todo los cursos tomados por “David”

BD de documentos

Qué hacen mal:

- Manejo de información que cambia mucho
- Cruce de información no trivial

BD de documentos

Colección "**Courses**“

- Todos los alumnos que toman IIC2413 y IMT3830

```
{
    "courseID": IIC2413,
    "name": "Databases",
    "year": 2020,
    "students":[
        {
            "studentID": 1,
            "name": "Alice Cooper"
        },
        {
            "studentID": 2,
            "name": "David Bowie"
        },
        ...
    ]
}
```

```
{
    "courseID": IMT3830,
    "name": "Game Theory",
    "year": 2020,
    "students":[
        {
            "studentID": 2,
            "name": "David Bowie"
        },
        {
            "studentID": 3,
            "name": "Charly García"
        },
        ...
    ]
}
```

BD de documentos

Colección "**Courses**“

- Todos los alumnos que toman IIC2413 y IMT3830

Efectivamente hay que hacer un nested loop join:

- Iterar por todos los alumnos de IMT3830
- Iterar por todos los alumnos de IIC2413
- Ver si hacen el join

MongoDB soporta JavaScript y Python

- Se puede hacer, pero no es elegante

En resumen

BD de documentos:

- Útiles para despliegue de información estática
- Búsquedas simples
- Cruces muy sencillos

BD SQL:

- Información cambia mucho
- Tengo que hacer cruces cada rato
- Necesito ACID

Consistencia Eventual

BD Documentos y BASE

- Distintas aplicaciones en una misma base de datos acceden a distintos documentos al mismo tiempo

- En general diseñadas para montar varias instancias que (en teoría) tienen la misma información

- Propagan updates en forma descoordinada

Proveen **Consistencia Eventual**: Con el tiempo, todos los nodos del sistema llegarán a un estado consistente, permitiendo alta disponibilidad y tolerancia a fallos, a costa de no tener consistencia inmediata.

Consistencia Eventual

Pero la consistencia eventual puede generar problemas:

Si dos aplicaciones intentan acceder al mismo documento en MongoDB, estas pueden ser versiones diferentes del documento

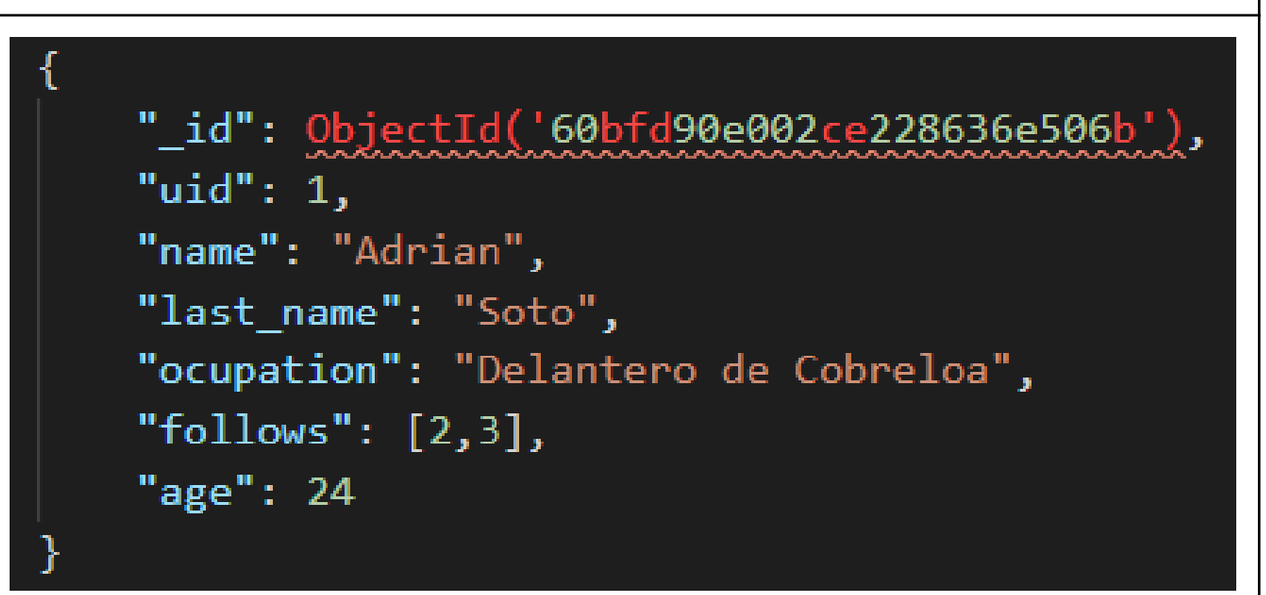
MongoDB

MongoDB

Es una base de datos NoSQL orientada a documentos que almacena datos en formato JSON (BSON). Estas son sus características:

- **Flexibilidad de Esquema**: Permite almacenar documentos con diferentes estructuras sin migraciones complejas.
- **Escalabilidad Horizontal**: Distribuye datos en múltiples servidores para mejorar rendimiento y capacidad.
- **Alto Rendimiento**: Optimizado para operaciones rápidas de lectura y escritura.
- **Replicación y Alta Disponibilidad**: Asegura disponibilidad y recuperación ante fallos mediante réplicas.

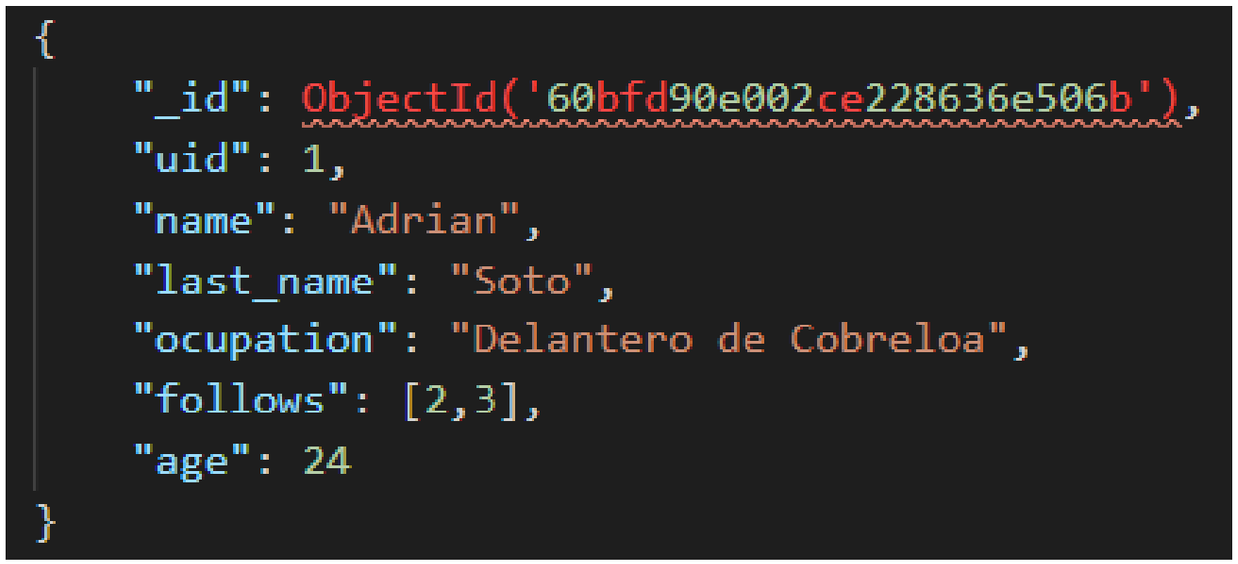
MongoDB

Usuarios	
Key	JSON
60bfd90e002ce228636e506b	
60bfd90e002ce228636e5215	...
...	...

MongoDB

Colección: una agrupación de documentos similares

Usuarios

Key	JSON
60bfd90e002ce228636e506b	
60bfd90e002ce228636e5215	…
…	…


```
{
    "_id": ObjectId('60bfd90e002ce228636e506b'),
    "uid": 1,
    "name": "Adrian",
    "last_name": "Soto",
    "ocupation": "Delantero de Cobreloa",
    "follows": [2,3],
    "age": 24
}
```

MongoDB

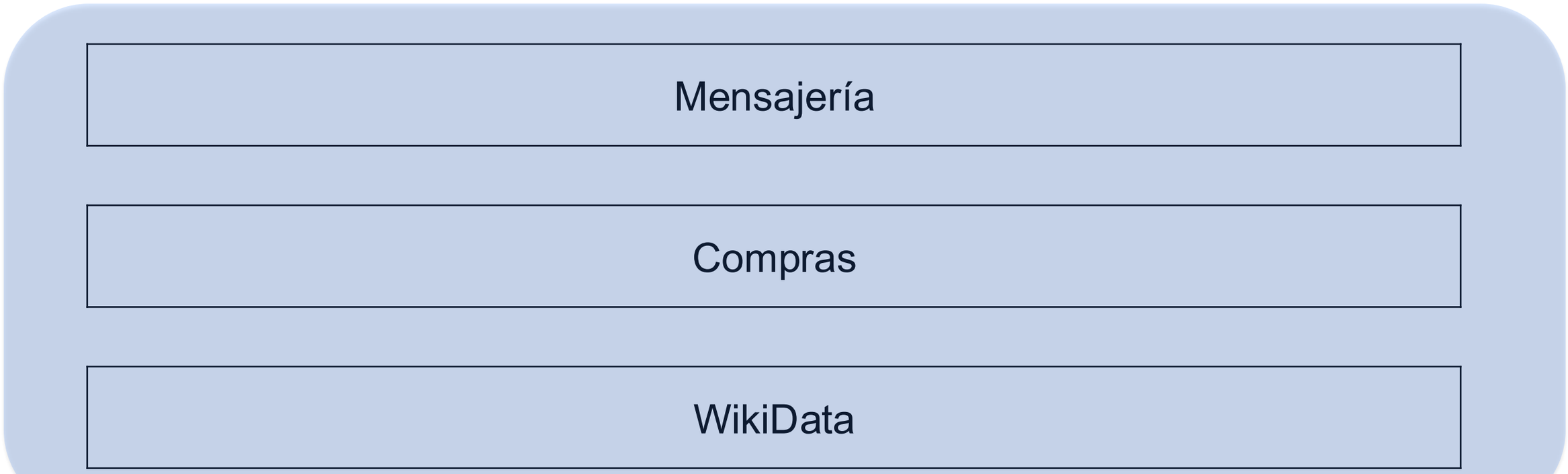
Base de Datos: contienen colecciones relacionadas

Usuarios	
Key	JSON
...	...

Mensajes	
Key	JSON
...	...

Likes	
Key	JSON
...	...

MongoDB



Un servidor contiene varias bases de datos

Consultando a MongoDB

show dbs … muestra bases de datos disponibles
use dbName … ahora usamos base de datos dbName
show collections … colecciones en nuestra base de datos

db.colName.find() … todos los documentos en la colección colName

db.colName.find().pretty() … muentra los datos de mejor forma

db.colName.find({"name": "Adrian"}) … selección

db.colName.find({"age": {$gte:23}}) … selección

db.colName.find({"age": {$gte:23}},{"name":1}) … proyección

Ver más en: http://aidanhogan.com/teaching/cc5212-1-2020/lectures/MDP2020-10.pdf

Full-text search & retrieval

¿Cómo buscar texto?

Cuando buscamos “Chilean Mammal”:

- El sistema encuentra todos los documentos que tienen Chilean y Mammal

¿Pero cómo lo hacemos para ordenar los resultados?
Puede ser problemático en grandes bases de datos

Índices Invertidos

Para hacer la búsqueda eficiente utilizamos **índices invertidos**

Para cada palabra del universo de documentos, guardamos punteros que nos indican dónde están los documentos

Índices Invertidos

	Documento 1	Documento 2	Documento 3	...
Chile	1	0	1	
of	1	1	1	
town	0	0	1	
commune	0	1	0	
...				

Text Search en MongoDB

Un índice especial … permite búsqueda rápida de texto

```
db.colName.createIndex({"attributeName":"text"})

db.users.find({$text: {$search: "Delantero de Cobreloa"}})
```

Ver más en: https://www.youtube.com/watch?v=vR97-4UG7x0

TF – IDF

(Frecuencia del Término - Frecuencia Inversa de los Documentos)

TF – IDF

Principio 1:

- El puntaje es proporcional a la cantidad de veces que aparece la palabra en el documento

Principio 2:

- El puntaje es inversamente proporcional a la cantidad de documentos en los que aparece la palabra

TF – IDF

Term Frequency:

- $F_D(t)$ = Número de veces que aparece t en D

Inverse Document Frequency:

- IDF(t) = log(número de documentos / númerode documentos en los que aparece t)

Entonces:

$$\textbf{TF-IDF} = F_D(t) \cdot \textbf{IDF}(t)$$

TF - IDF

Ejemplo

D1: Ojo por ojo, diente por diente

D2: Ojo por ojo, y el mundo acabará ciego

D3: Si luchas contra el mundo, ponte del lado del mundo

Entonces, calculamos el TF-IDF de “ojo” y “mundo” para cada documento…

TF – IDF: Ejemplo

Paso 1, Calcular TF: El TF se calcula como el número de veces que aparece un término en un documento dividido por el número total de términos en el documento.

Documento D1: "Ojo por ojo, diente por diente"

- Total de palabras: 6
- TF("ojo") = 2/6 = 0.333
- TF("mundo") = 0/6 = 0

Documento D2: "Ojo por ojo, y el mundo acabará ciego"

- Total de palabras: 8
- TF("ojo") = 2/8 = 0.25
- TF("mundo") = 1/8 = 0.125

Documento D3: "Si luchas contra el mundo, ponte del lado del mundo"

- Total de palabras: 10
- TF("ojo") = 0/10 = 0
- TF("mundo") = 2/10 = 0.2

TF – IDF: Ejemplo

Paso 2, calcular IDF: El IDF se calcula como el logaritmo del número total de documentos dividido por el número de documentos que contienen el término.

- Total de documentos: 3
- Documentos que contienen "ojo": 2 (D1 y D2)
- Documentos que contienen "mundo": 2 (D2 y D3)
- IDF("ojo") = log(3/2) = 0.176
- IDF("mundo") = log(3/2) = 0.176

TF – IDF: Ejemplo

Paso 3, Calcular TF-IDF: El TF-IDF se calcula multiplicando el TF por el IDF.

TF-IDF para “ojo”:

- Documento D1: 0.333 * 0.176 = 0.059
- Documento D2: 0.25 * 0.176 = 0.044
- Documento D3: 0 * 0.176 = 0

TF-IDF para “mundo”:

- Documento D1: 0 * 0.176 = 0
- Documento D2: 0.125 * 0.176 = 0.022
- Documento D3: 0.2 * 0.176 = 0.035

TF – IDF

Hay distintas funciones para TF e IDF

Generalmente se incorporan funciones para Stemming y Stop Words para procesamiento de lenguaje natural

Cada compañía tiene su receta, depende además del idioma

TF - IDF

Búsqueda de documentos

Se genera una matriz en donde las dimensiones son las palabras y los documentos

Cada “casillero” señala el TF-IDF de la palabra en cada documento

Cuando un usuario busca una frase, se genera un vector y se retornan los documentos con vectores más similares

Map Reduce

Map Reduce

- Algoritmo eficiente para computación paralela/distribuida
- Paradigma de programación - mezcla de datos con controlador
- Sacrificios para acelerar computación

Computación Distribuida

- Datos tan grandes que no caben en un computador
- Repartidos en muchos servidores
- Cada servidor no conoce lo que tiene el resto
- No podemos dar el lujo de comunicar todos los datos

Map Reduce

Ejemplo: ver cuantas veces ocurre cada palabra en un texto T

¿Cómo hacer esto en un archivo de 100 **petabytes***?

*100 petabytes = 100000000 gigabytes

Map Reduce

- **Map**: recibe datos y genera pares key – values
- **Reduce**: recibe pares con el mismo key y los agrega
- **Shuffle**: transfiere los datos desde mappers a reducers

Map Reduce - Arquitectura

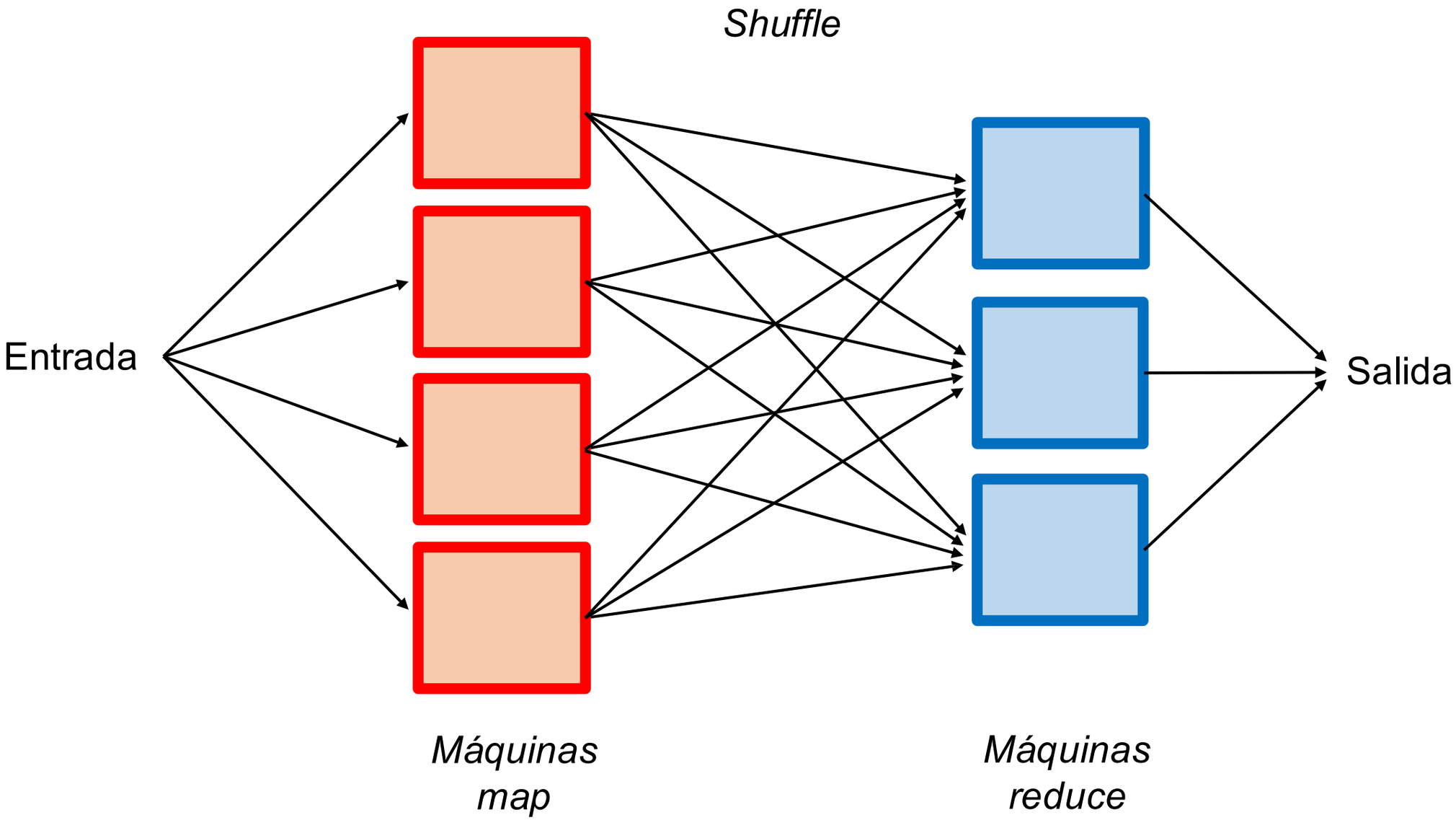
Mappers:

- Nodos encargados de hacer Map
- Reciben parte del documento y lo envían a los reducers (a través de **shuffle**)

Reducers:

- Nodos encargados de hacer Reduce
- Reciben los Map y los agregan
- El output es la unión de cada Reducer

Map Reduce



Map Reduce - Ejemplo

¿Cuántas veces cada palabra ocurre en un archivo de texto grande? Para saberlo, tenemos:

Map:

- Recibe un pedazo de texto
- Por cada palabra, emite el par (palabra, número de ocurrencias)

Reduce:

- Cada reduce recibe todos los pares asociados a la misma palabra
- Junta todos estos pares y suma las ocurrencias

INPUT

hola que
hola año
zzz hola
que zzz
que

Máquina map1

Máquina map2

Máquina map3

Máquina reduce1

Palabras A-M

Máquina reduce2

Palabras N-Z

SEPARAR INPUT

hola que hola
año zzz hola
que zzz que

Máquina map1

Máquina map2

Máquina map3

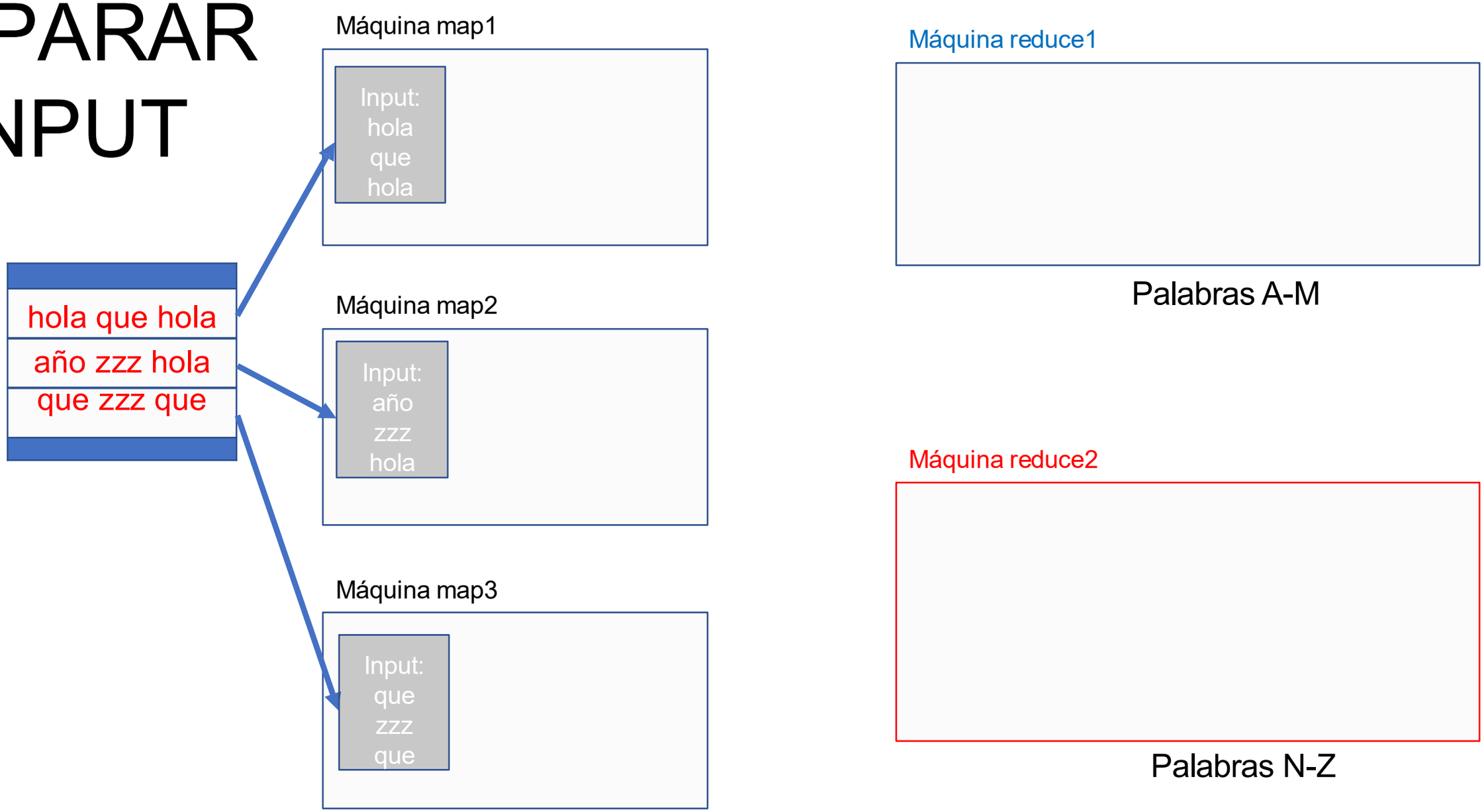
Máquina reduce1

Palabras A-M

Máquina reduce2

Palabras N-Z

SEPARAR INPUT



MAP

Input

hola que hola
año zzz hola
que zzz que

Máquina map1

Input: hola que hola

Cómputo

Output: (hola,2) (que,1)

Máquina map2

Input: año zzz hola

Cómputo

Output: (año,1) (zzz,1) (hola,1)

Máquina map3

Input: que zzz que

Cómputo

Output: (que,2) (zzz,1)

Máquina reduce1

Palabras A-M

Máquina reduce2

Palabras N-Z

SHUFFLE

Input

hola que hola

año zzz hola

que zzz que

Máquina map1

Input: hola que hola

Cómputo

Output: (hola,2) (que,1)

Máquina map2

Input: año zzz hola

Cómputo

Output: (año,1) (zzz,1) (hola,1)

Máquina map3

Input: que zzz que

Cómputo

Output: (que,2) (zzz,1)

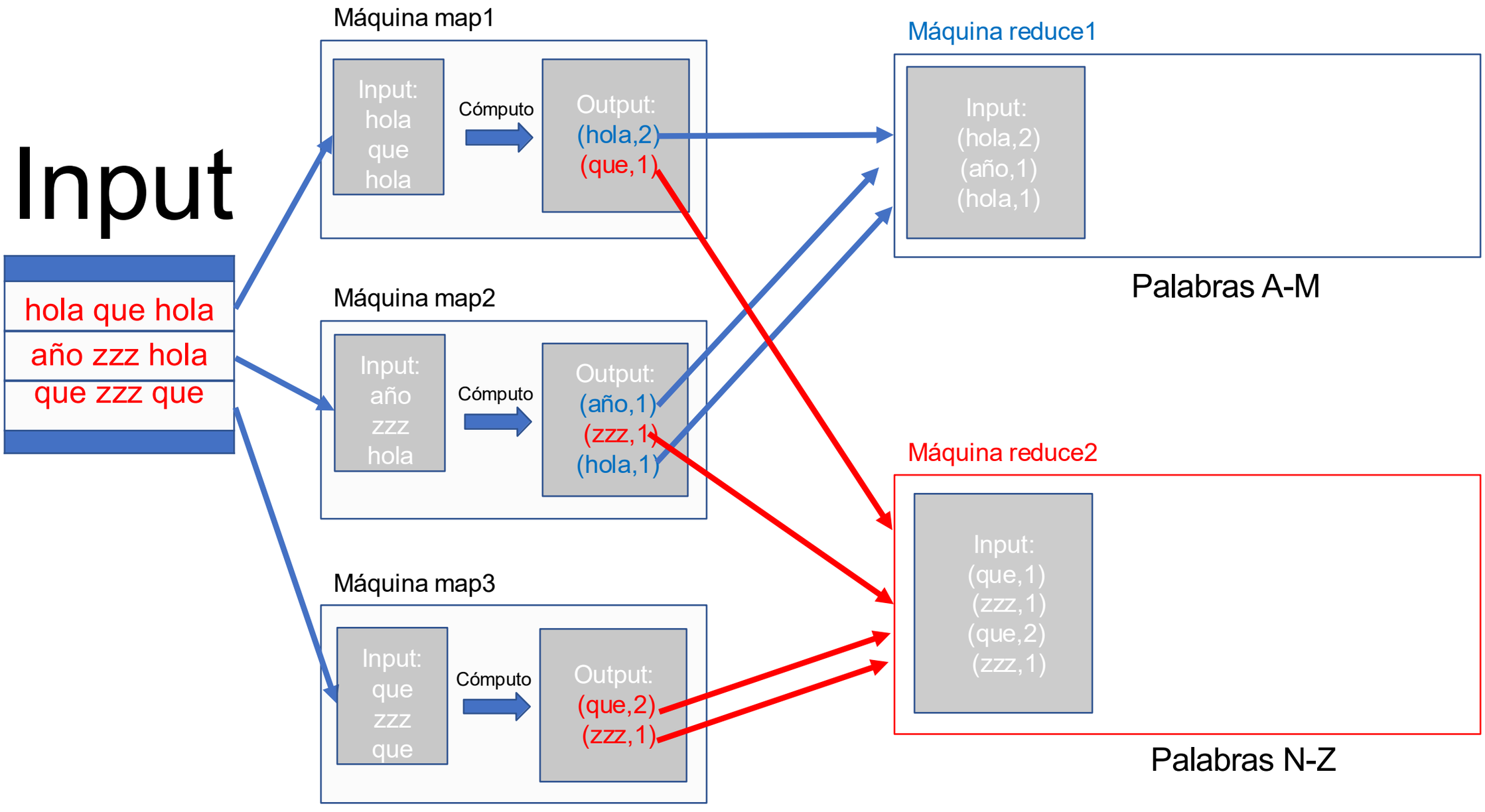
Máquina reduce1

Palabras A-M

Máquina reduce2

Palabras N-Z

SHUFFLE


Input
hola que hola
año zzz hola
que zzz que
Máquina map1
Input:
hola
que
hola
Cómputo
Output:
(hola,2)
(que,1)
Máquina map2
Input:
año
zzz
hola
Cómputo
Output:
(año,1)
(zzz,1)
(hola,1)
Máquina map3
Input:
que
zzz
que
Cómputo
Output:
(que,2)
(zzz,1)
Máquina reduce1
Input:
(hola,2)
(año,1)
(hola,1)
Palabras A-M
Máquina reduce2
Input:
(que,1)
(zzz,1)
(que,2)
(zzz,1)
Palabras N-Z

REDUCE

Input

hola que hola
año zzz hola
que zzz que

Máquina map1

Input: hola que hola

Cómputo

Output: (hola,2) (que,1)

Máquina map2

Input: año zzz hola

Cómputo

Output: (año,1) (zzz,1) (hola,1)

Máquina map3

Input: que zzz que

Cómputo

Output: (que,2) (zzz,1)

Máquina reduce1

Input: (hola,2) (año,1) (hola,1)

Sorted: (año,1) (hola,1) (hola,2)

Palabras A-M

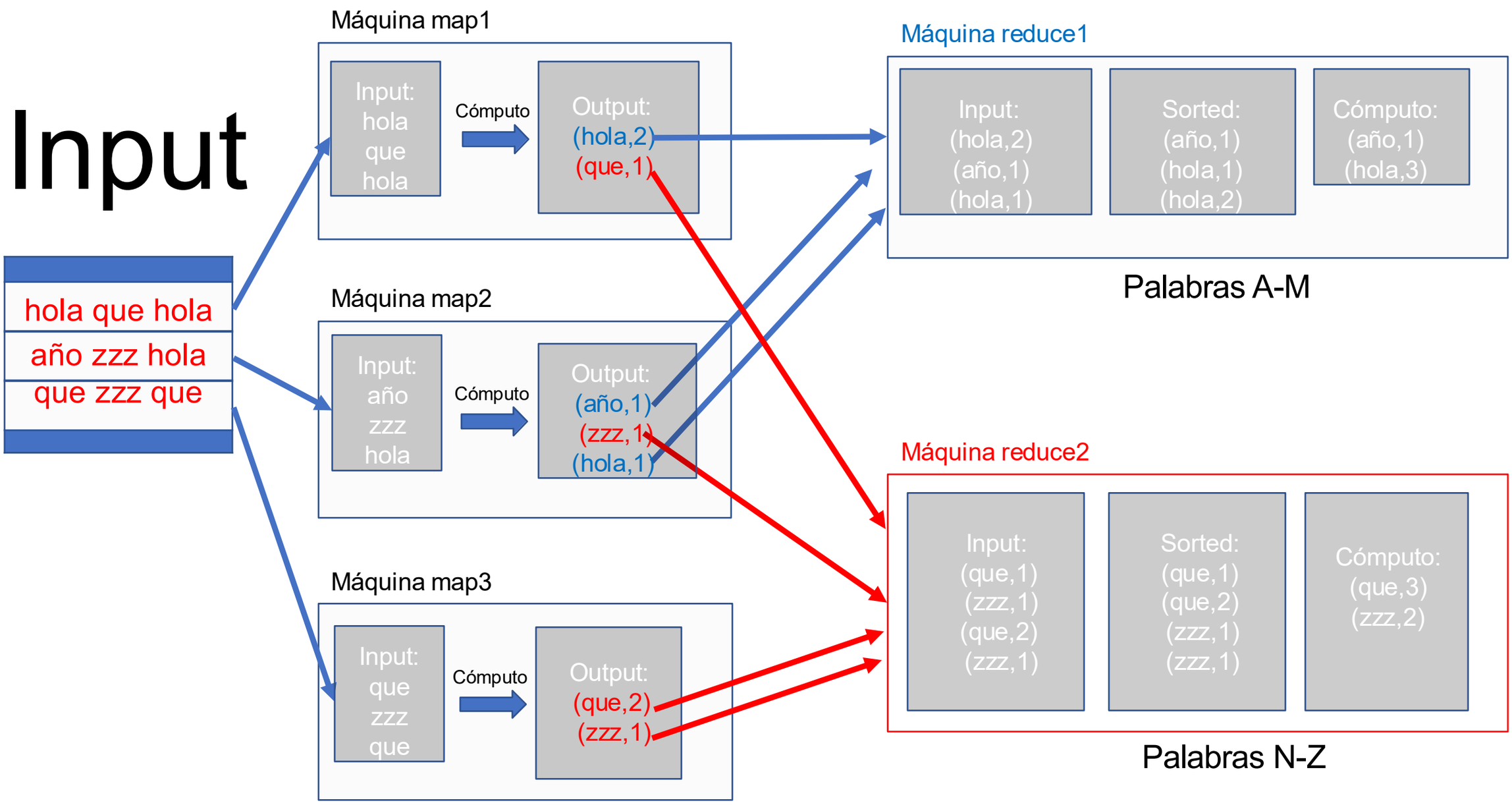
Máquina reduce2

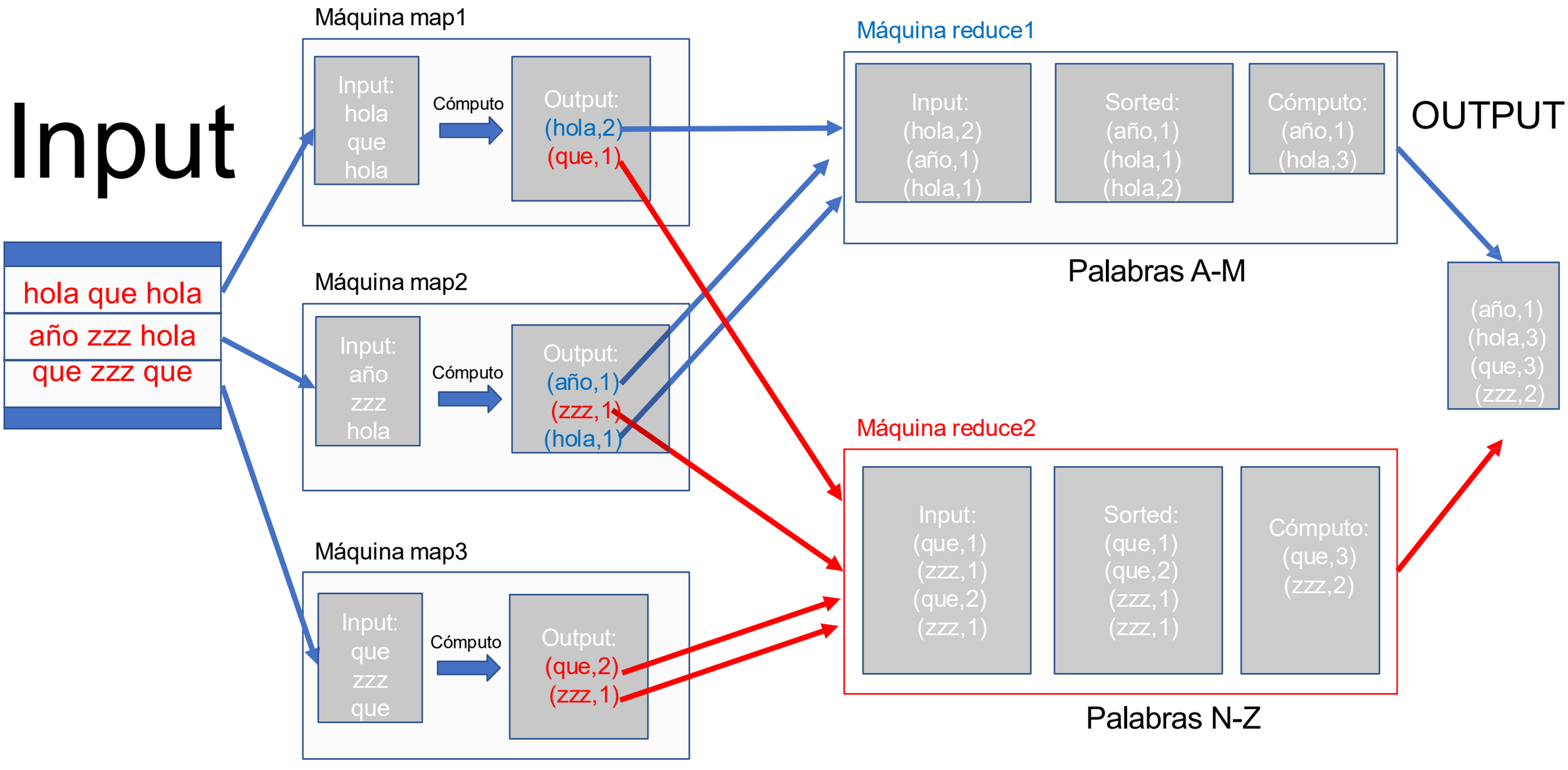
Input: (que,1) (zzz,1) (que,2) (zzz,1)

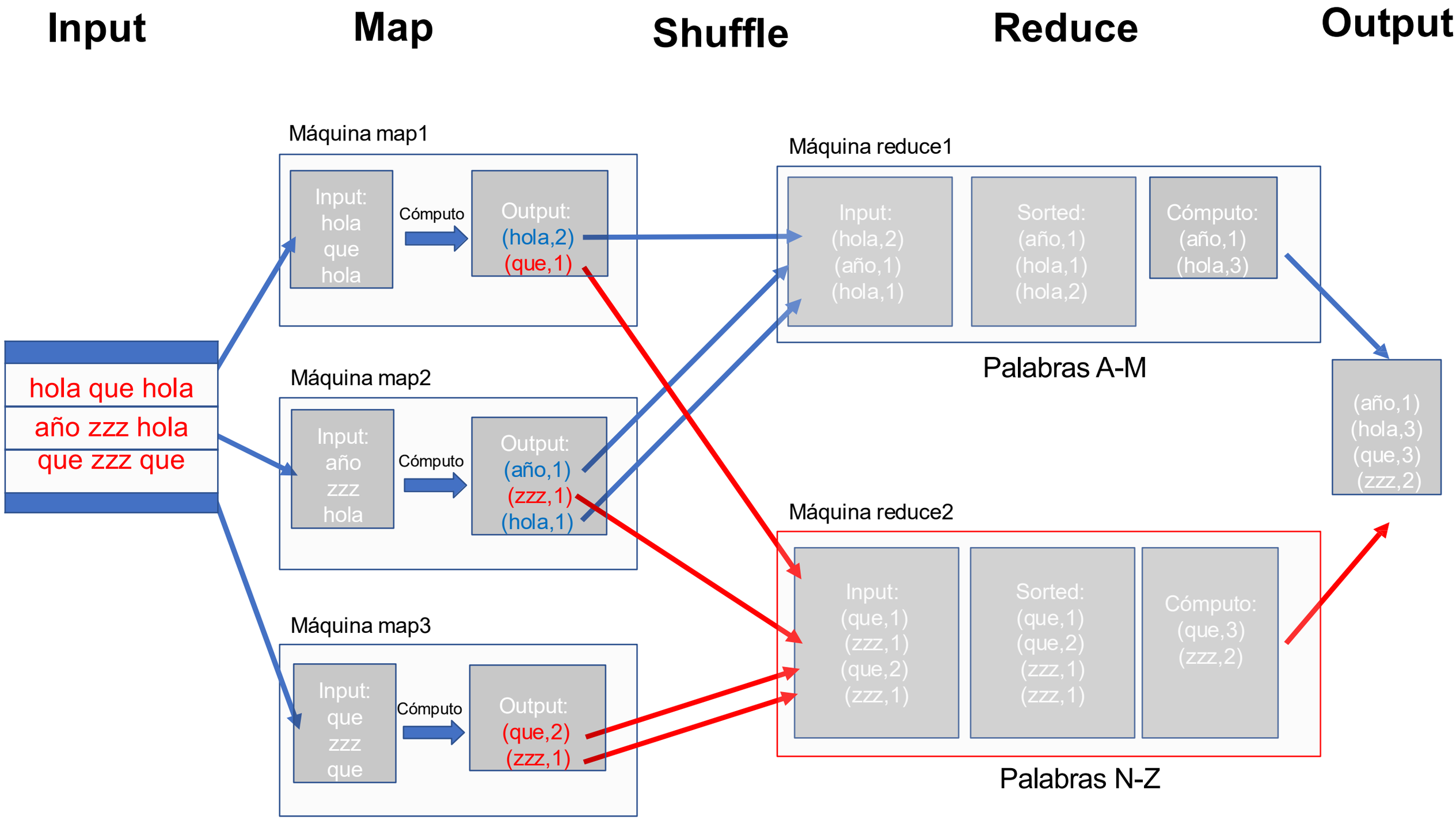
Sorted: (que,1) (que,2) (zzz,1) (zzz,1)

Palabras N-Z

REDUCE


Input
hola que hola
año zzz hola
que zzz que
Máquina map1
Input:
hola
que
hola
Cómputo
Output:
(hola,2)
(que,1)
Máquina map2
Input:
año
zzz
hola
Cómputo
Output:
(año,1)
(zzz,1)
(hola,1)
Máquina map3
Input:
que
zzz
que
Cómputo
Output:
(que,2)
(zzz,1)
Máquina reduce1
Input:
(hola,2)
(año,1)
(hola,1)
Sorted:
(año,1)
(hola,1)
(hola,2)
Cómputo:
(año,1)
(hola,3)
Palabras A-M
Máquina reduce2
Input:
(que,1)
(zzz,1)
(que,2)
(zzz,1)
Sorted:
(que,1)
(que,2)
(zzz,1)
(zzz,1)
Cómputo:
(que,3)
(zzz,2)
Palabras N-Z


Input
hola que hola
año zzz hola
que zzz que
Máquina map1
Input:
hola
que
hola
Cómputo
Output:
(hola,2)
(que,1)
Máquina map2
Input:
año
zzz
hola
Cómputo
Output:
(año,1)
(zzz,1)
(hola,1)
Máquina map3
Input:
que
zzz
que
Cómputo
Output:
(que,2)
(zzz,1)
Máquina reduce1
Input:
(hola,2)
(año,1)
(hola,1)
Sorted:
(año,1)
(hola,1)
(hola,2)
Cómputo:
(año,1)
(hola,3)
Palabras A-M
Máquina reduce2
Input:
(que,1)
(zzz,1)
(que,2)
(zzz,1)
Sorted:
(que,1)
(que,2)
(zzz,1)
(zzz,1)
Cómputo:
(que,3)
(zzz,2)
Palabras N-Z
OUTPUT
(año,1)
(hola,3)
(que,3)
(zzz,2)


Input
Map
Shuffle
Reduce
Output
hola que hola
año zzz hola
que zzz que
Máquina map1
Input:
hola
que
hola
Cómputo
Output:
(hola,2)
(que,1)
Máquina map2
Input:
año
zzz
hola
Cómputo
Output:
(año,1)
(zzz,1)
(hola,1)
Máquina map3
Input:
que
zzz
que
Cómputo
Output:
(que,2)
(zzz,1)
Máquina reduce1
Input:
(hola,2)
(año,1)
(hola,1)
Sorted:
(año,1)
(hola,1)
(hola,2)
Cómputo:
(año,1)
(hola,3)
Palabras A-M
Máquina reduce2
Input:
(que,1)
(zzz,1)
(que,2)
(zzz,1)
Sorted:
(que,1)
(que,2)
(zzz,1)
(zzz,1)
Cómputo:
(que,3)
(zzz,2)
Palabras N-Z
(año,1)
(hola,3)
(que,3)
(zzz,2)

Map Reduce - Ejemplo: Join

¿Cómo hago un join con Map Reduce?

- Modelo: un archivo con el nombre de la tabla y sus tuplas
- Map: Agrupo por el atributo que hace el join
- Reduce: Hago el producto cruz para las tablas distintas

INPUT

R

A	B
1	1
1	3
3	2
3	3

S

B	C
2	4
2	7
3	8
3	9

Máquina map1

Máquina map2

Máquina reduce llave 1

Máquina reduce llave 2

Máquina reduce llave 3

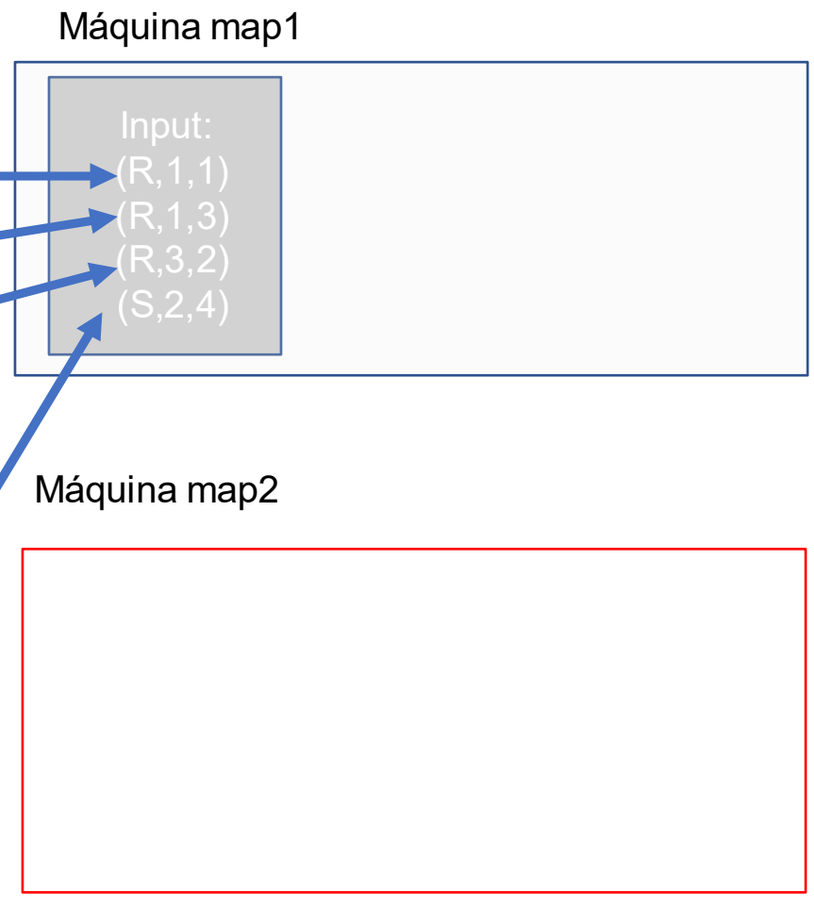
INPUT MAP

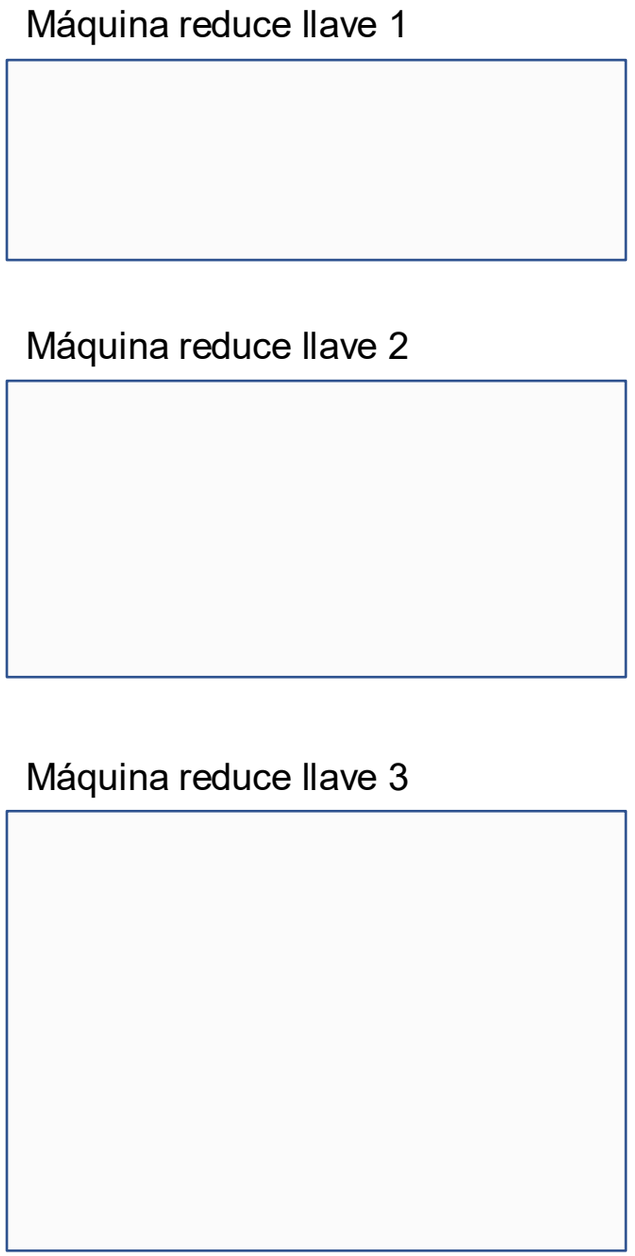
R

A	B
1	1
1	3
3	2
3	3

S

B	C
2	4
2	7
3	8
3	9





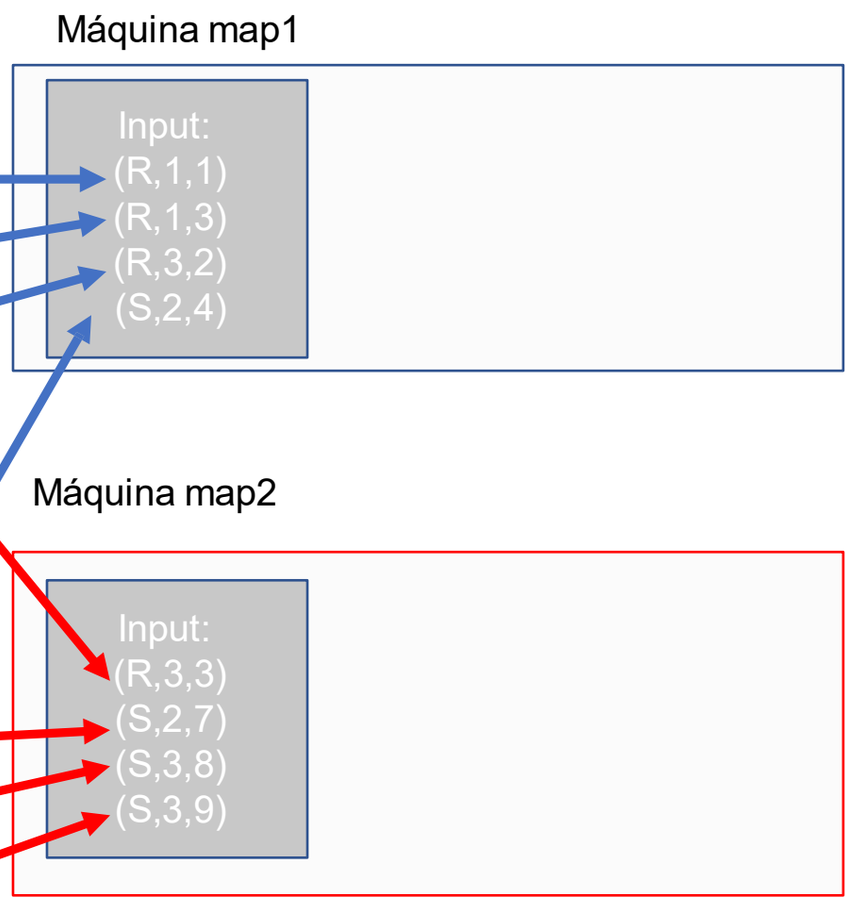
INPUT MAP

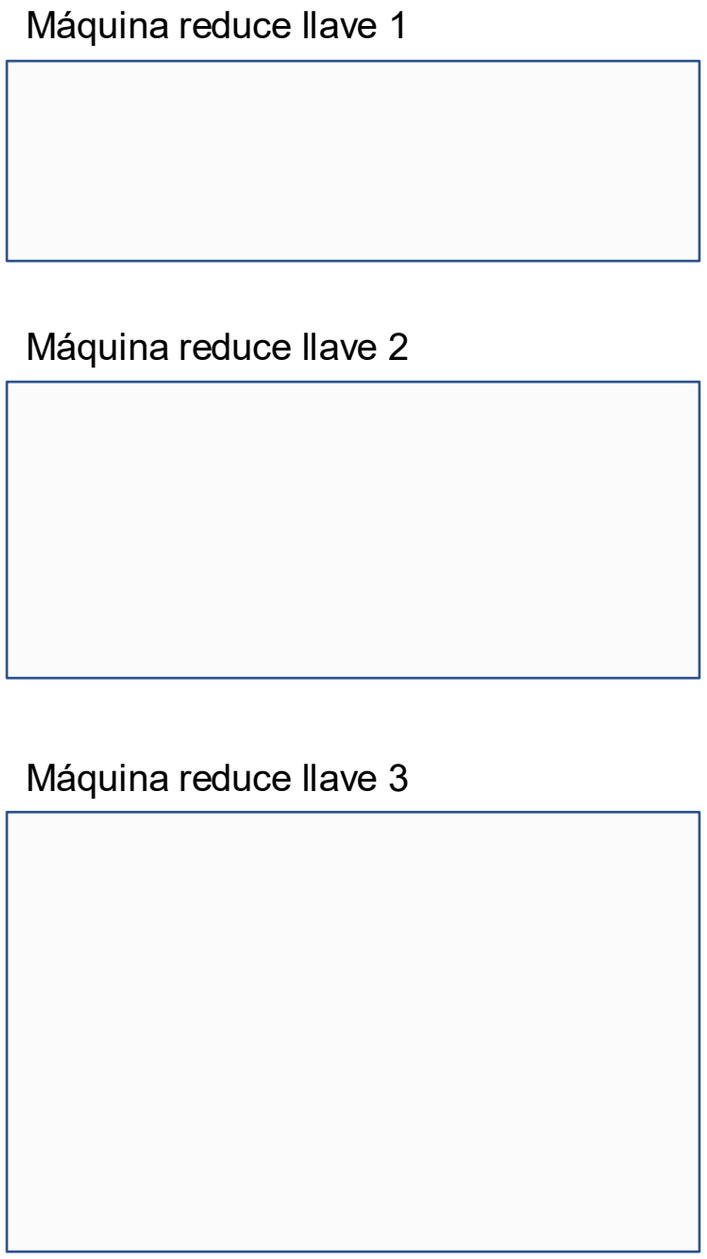
R

A	B
1	1
1	3
3	2
3	3

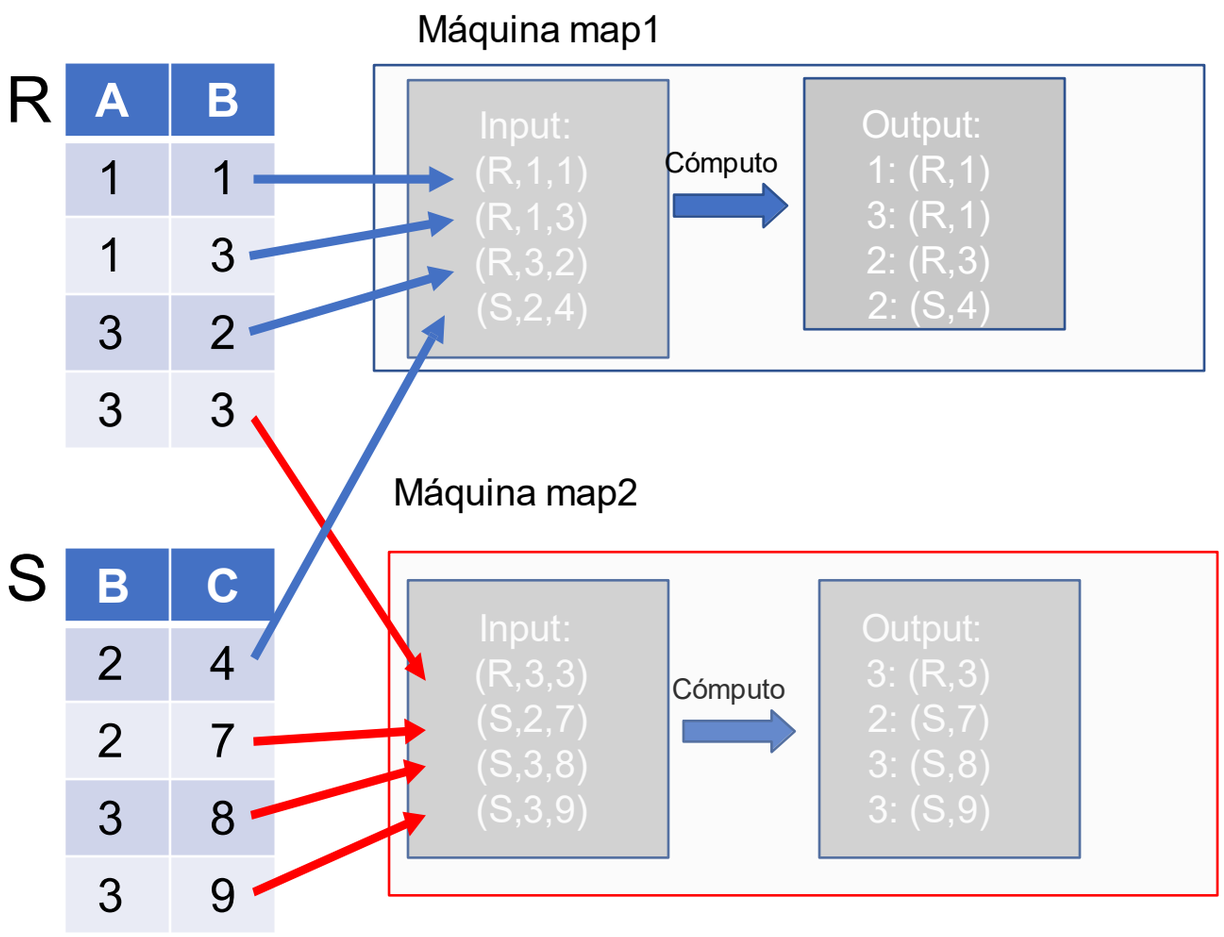
S

B	C
2	4
2	7
3	8
3	9





INPUT MAP

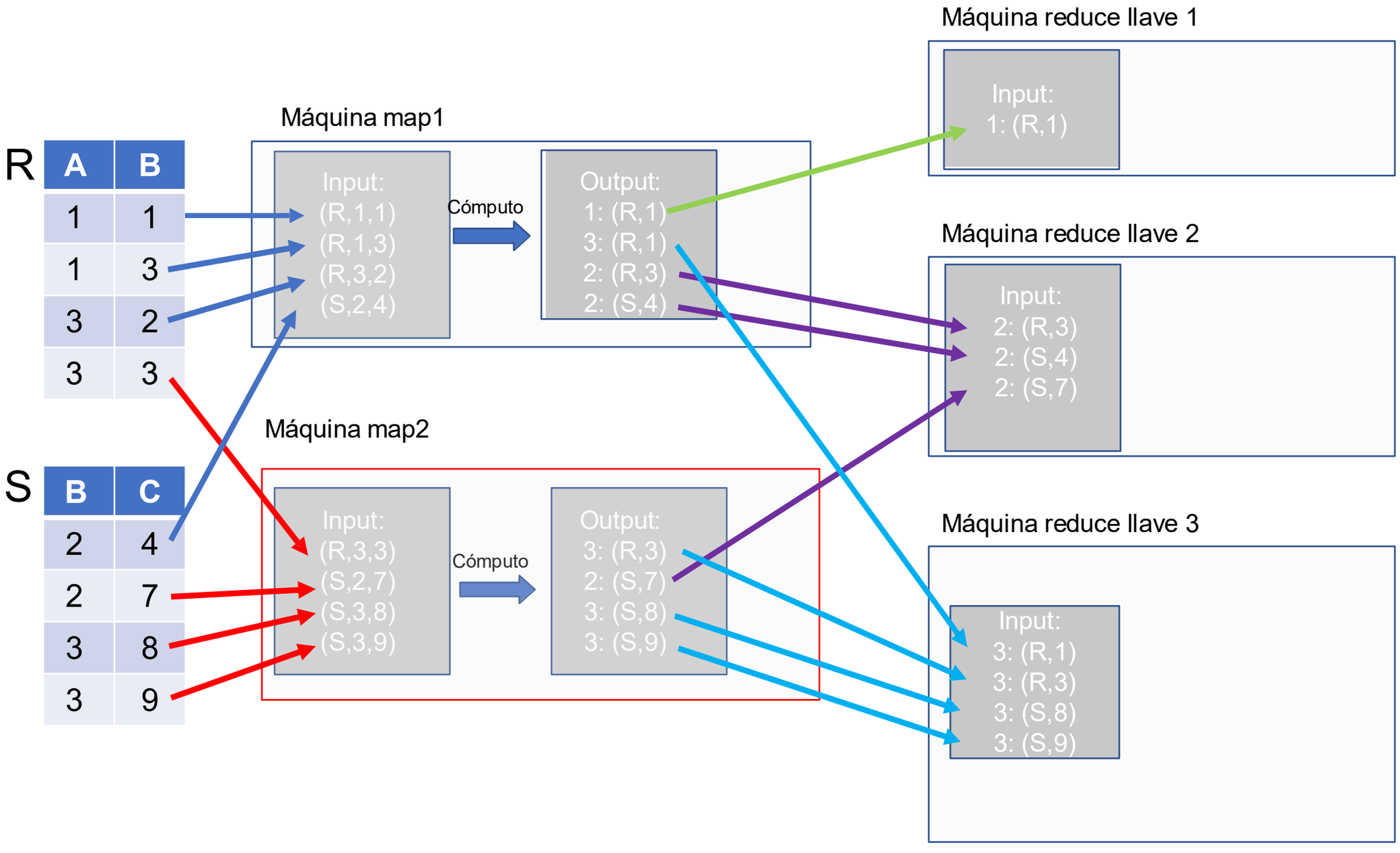


Máquina reduce llave 1

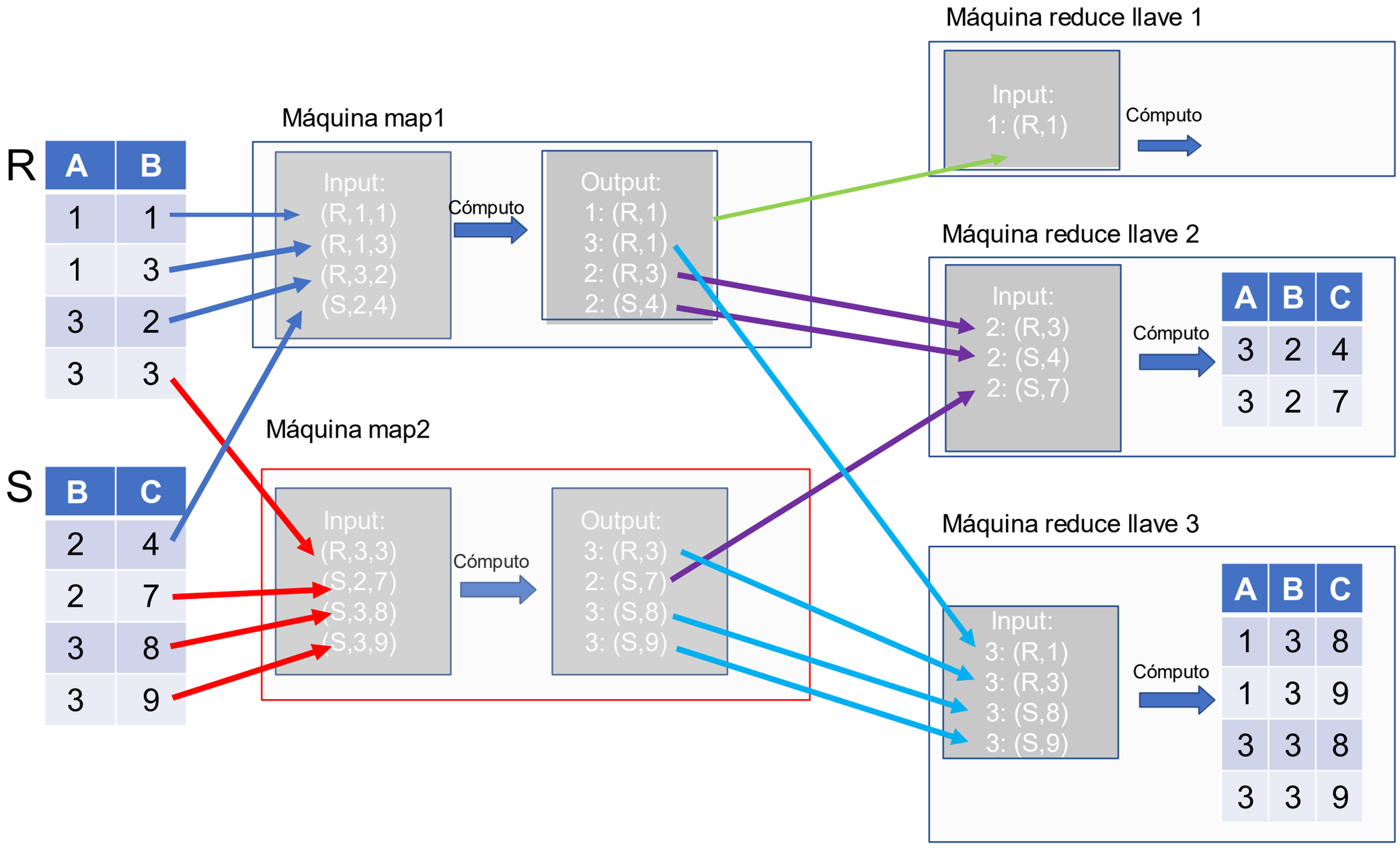
Máquina reduce llave 2

Máquina reduce llave 3

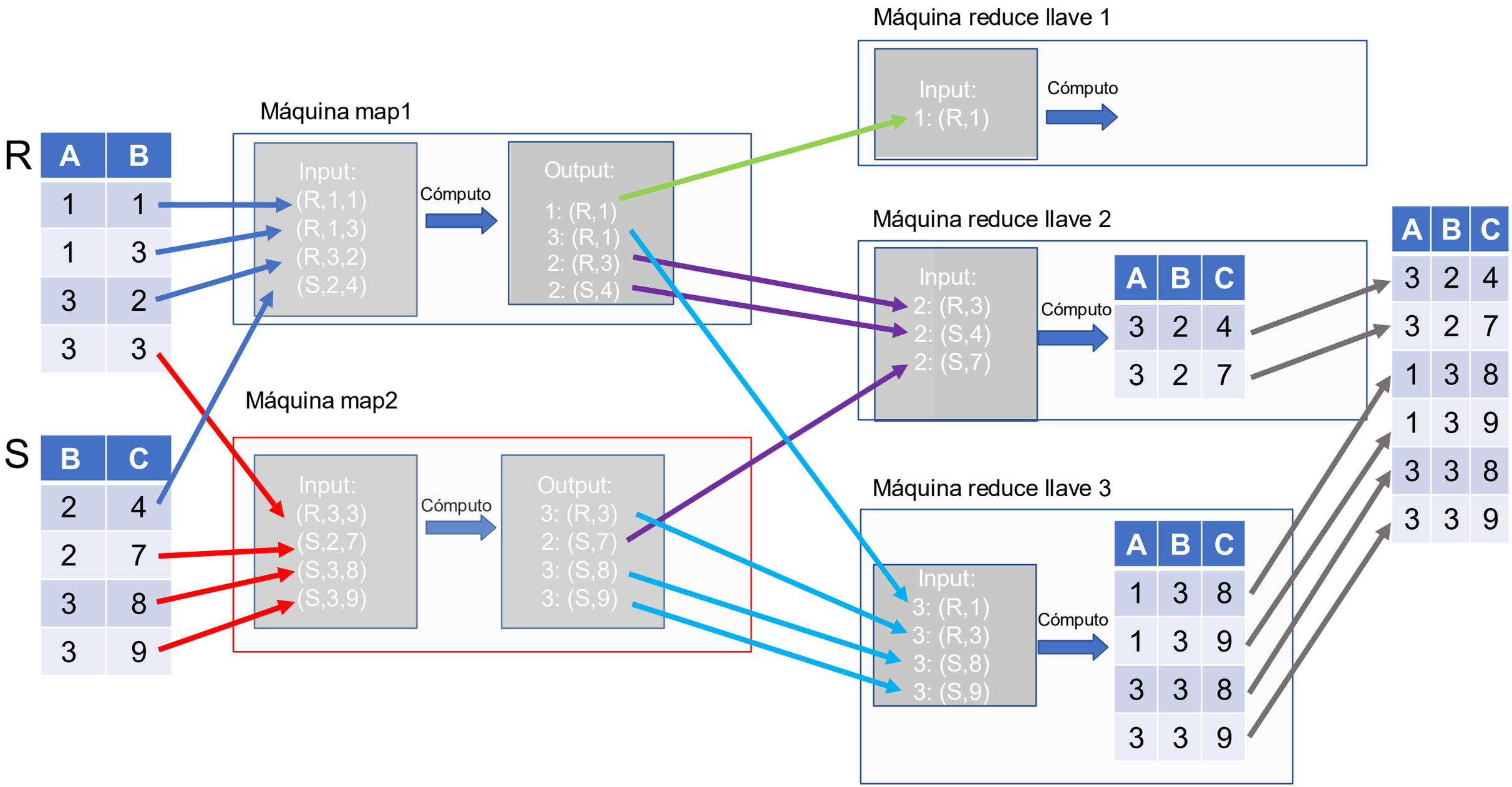
INPUT MAP SHUFFLE



INPUT MAP SHUFFLE REDUCE



INPUT MAP SHUFFLE REDUCE OUTPUT



Map Reduce y BDD

No es un descubrimiento nuevo, pero recientemente se ha visto calzar perfectamente con las necesidades de las grandes BD

Es la arquitectura más importante en sistemas que reciben grandes bases de datos

- Apache Hadoop: La implementación open source de Map -Reduce, presente en muchos sistemas con computación distribuida

Material adicional

https://www.youtube.com/watch?v=vR97-4UG7x0

https://bmdigitales-bibliotecas-uc-cl.pucdechile.idm.oclc.org/html5/DATABASE%20MANAGEMENT%20SYSTEMS/961/