Big Data

Diplomado de Datos 2021

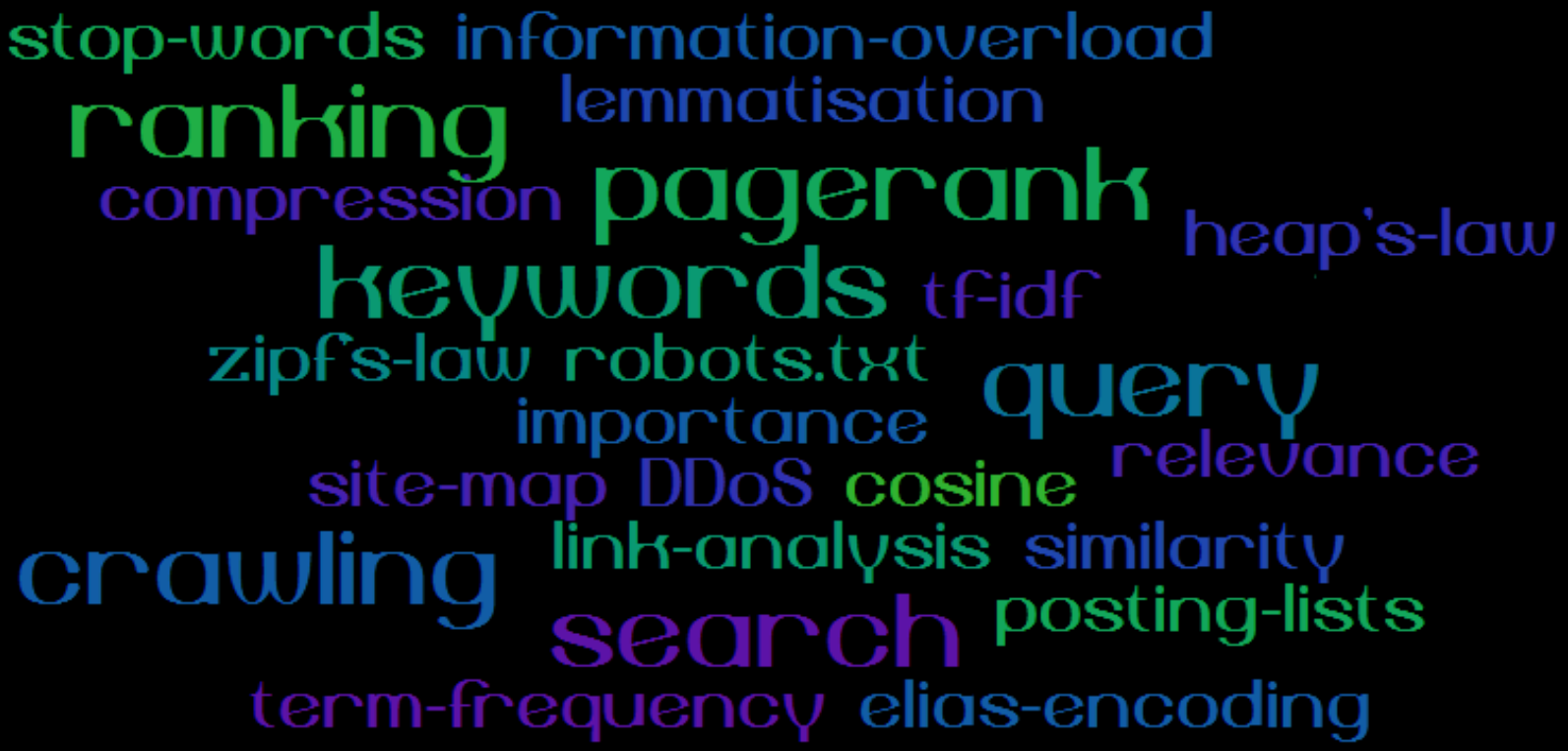
Clase 7: NoSQL (I)

Aidan Hogan
aidhog@gmail.com

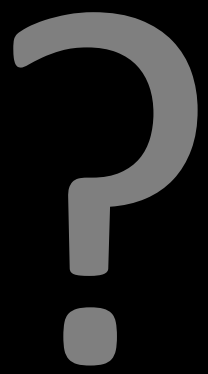
Hadoop/MapReduce/Pig/Spark:
Procesando datos masivos

Recuperación de información:

Almacenando datos masivos no estructurados



¿Almacenando datos masivos estructurados?



Datos masivos:
Almacenar datos estructurados

Bases de datos relacionales

Bases de datos relacionales: ¿Talla única?

"One Size Fits All": An Idea Whose Time Has Come and Gone

Michael Stonebraker
Computer Science and Artificial Intelligence Laboratory, M.I.T., and StreamBase Systems, Inc.
stonebraker@csail.mit.edu

Uğur Çetintemel
Department of Computer Science Brown University, and StreamBase Systems, Inc.
ugur@cs.brown.edu

Abstract

The last 25 years of commercial DBMS development can be summed up in a single phrase: "One size fits all". This phrase refers to the fact that the traditional DBMS architecture (originally designed and optimized for business data processing) has been used to support many data-centric applications with widely varying characteristics and requirements.

In this paper, we argue that this concept is no longer applicable to the database market, and that the commercial world will fracture into a collection of independent database engines, some of which may be unified by a common front-end parser. We use examples from the stream-processing market and the data-warehouse market to bolster our claims. We also briefly discuss other markets for which the traditional architecture is a poor fit and argue for a critical rethinking of the current factoring of systems services into products.

of multiple code lines causes various practical problems, including:

- *a cost problem*, because maintenance costs increase at least linearly with the number of code lines;
- *a compatibility problem*, because all applications have to run against every code line;
- *a sales problem*, because salespeople get confused about which product to try to sell to a customer; and
- *a marketing problem*, because multiple code lines need to be positioned correctly in the marketplace.

To avoid these problems, all the major DBMS vendors have followed the adage "put all wood behind one arrowhead". In this paper we argue that this strategy has failed already, and will fail more dramatically off into the future.

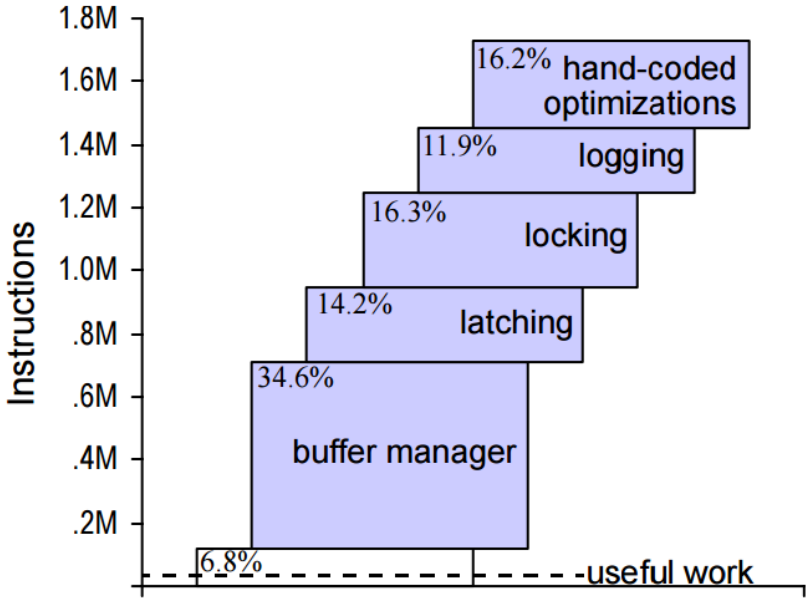
The rest of the paper is structured as follows. In Section 2, we briefly indicate why the single code-line strategy has failed already by citing some of the key characteristics of the data warehouse market. In Section


Spongecell
Hula
kiko
Trumba
eskobo
mayomi
Pageflakes
vimeo
skobee
shadows
grovee
YouTube
Zimbra
LookSmart FURL | Your Personal Web File
smugmug
newsgator
Blogniscient
TiNFiNGER
shutterfly
PodDater
Feedster
favoor
Planzo.com
ZAZZLE
Tailrank
TagWorld
nuvvo
dogear
yakalike
grouper
ODDPOST
QOOP
iNods
Lulu
BR
blish
flagr
FireAnt
simplyhired
Veoh
theadcloud
rbloc.com
pixagogo
gather
Agatra
browsr
oyogi
cafepress
Renkoo
standpoint
meebo
EXTRA TASTY!
last·fm
Jotspot
Frappr!
jeteye
dabble
YEDDA
Writeboard
SHOUTWIRE
iKarma
KaNOODLE
AirSet
tech memeorandum
CalendarHub
Suprglu
PIECING YOUR WEB TOGETHER
pando
zigtag
Findory
backfence
clipmarks
wayfaring
gOFFICE
AllPeers
orb
Rallypoint
Zoozio
blogbeat
Ziggs
zoto
vSocial
Boltfolio
wink
riya
Wordcast
Opinity
reddit
measuremap
gumshoo
bluepulse
imvu BETA
STREAMLOAD
Ta-da Lists
FeedSky
jellyBam.INC
nativetext
CONGOO
PODZINGER
RSS MAD
FeedTier
phanfare
Fruitcast
PubSub
dPolls
flickr
Ning
Ookles
Strongspace
zoominfo
CASTPOST
WIKIPEDIA
yubnub
AC
BLOOP
ProjectSpaces
FeedBurner
Bloglines
purevolume.com
FOTOLOG
ourmedia
Yub.com
Spot Runner
myspace
NewsAlloy
Allmydata.com
gabbacom
Gcast
blinkx
openomy
riffs
ajchat
chatsum
PANDORA
looklater
jambo
ROLLYO
ClipShack
squishr
PROTOPAGE
WebJay
PLAZES
Noodly
30 BETA
woofdir
diigo
box
Jots
Xdrive
vizu
digg
del.icio.us
omni drive
AlmondRocks
Tagyu
writely
Simpy
Gtalkr
TRUVEO
egoSurf
mozy
Quimble
pegasus
remember the milk
SQUIDOO
picturecloud
newsvine
Clipfire
Basecamp
Netvibes
goowy
My Videokaraoke
Lexxe alpha
powered by advanced natural language technology
LOOKSTER
PXN8
measuremap
facebook
43 Things
Technorati
Platial
Backpack
Gmail

Bases de datos relacionales: Aspectos costosos

- "Structured Query Language" (SQL):
  - Lenguaje declarativo
  - Muchas buenas características
  - **Difícil de optimizar**
- "Atomicity, Consistency, Isolation, Durability" (ACID):
  - Asegura correctitud de la base de datos
    - ¡Incluso si hay mucho tráfico!
  - **¡Las transacciones incurren en altos gastos!**
    - bloqueos multi-fase, multi-versionamiento, logging
- La distribución no es directa

Gastos en transacciones: el costo de ACID



- 640 transacciones por segundo en un sistema que garantiza ACID
- 12.700 transacciones por segundo en un sistema sin logging, transacciones o bloqueos

OLTP Through the Looking Glass, and What We Found There

Stavros Harizopoulos
HP Labs
Palo Alto, CA
stavros@hp.com

Daniel J. Abadi
Yale University
New Haven, CT
dna@cs.yale.edu

Samuel Madden Michael Stonebraker
Massachusetts Institute of Technology
Cambridge, MA
{madden, stonebraker}@csail.mit.edu

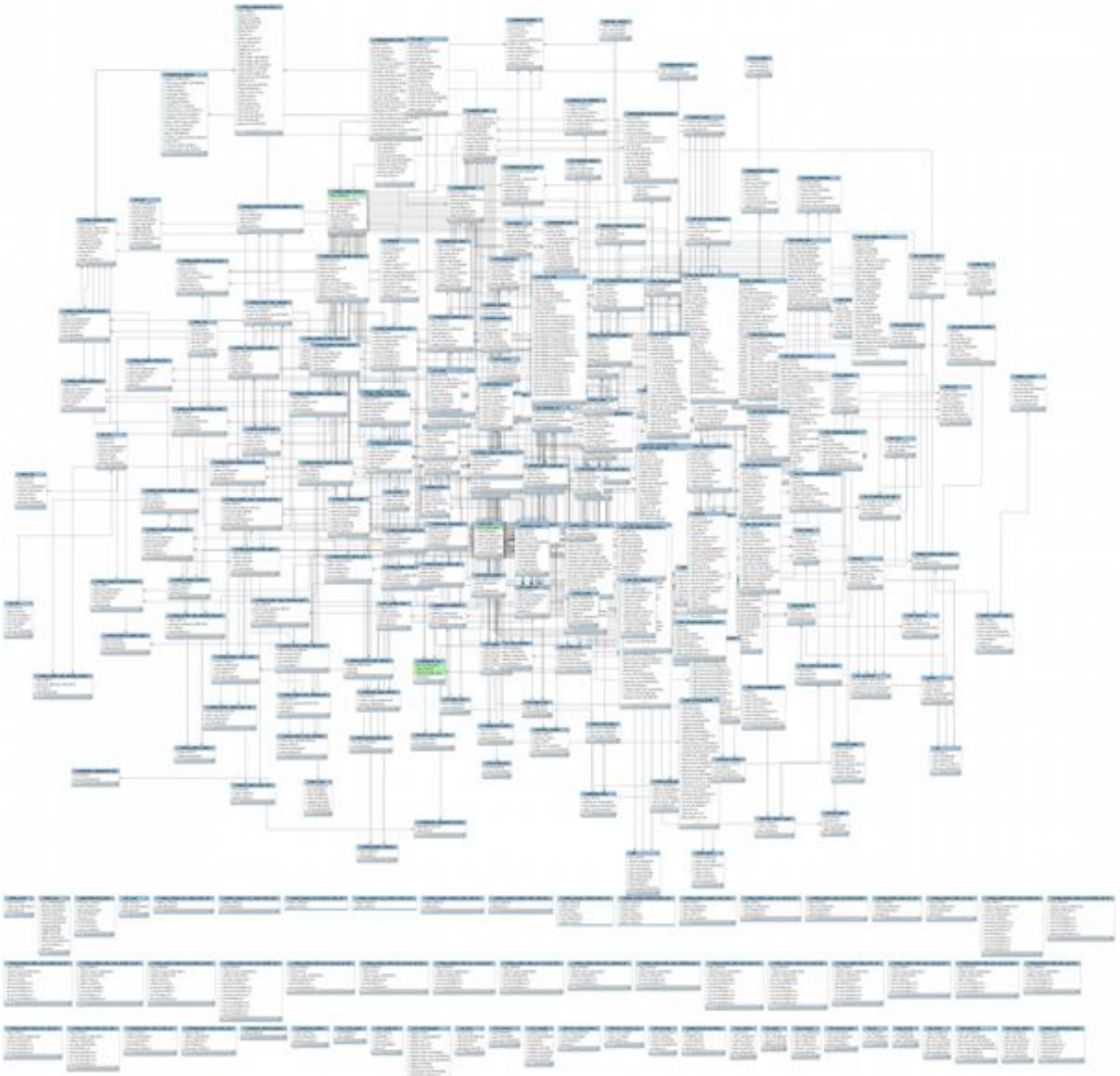
ABSTRACT

Online Transaction Processing (OLTP) databases include a suite of features — disk-resident B-trees and heap files, locking-based concurrency control, support for multi-threading — that were optimized for computer technology of the late 1970's. Advances in modern processors, memories, and networks mean that today's computers are vastly different from those of 30 years ago, such that many OLTP databases will now fit in main memory, and most OLTP transactions can be processed in milliseconds or less. Yet database architecture has changed little

1. INTRODUCTION

Modern general purpose online transaction processing (OLTP) database systems include a standard suite of features: a collection of on-disk data structures for table storage, including heap files and B-trees, support for multiple concurrent queries via locking-based concurrency control, log-based recovery, and an efficient buffer manager. These features were developed to support transaction processing in the 1970's and 1980's, when an OLTP database was many times larger than the main memory, and when the computers that ran these databases cost hundreds of thousands to

Bases de datos relaciones: Complejidad



¿Alternativas a bases de datos relacionales para gestionar datos masivos?

NoSQL

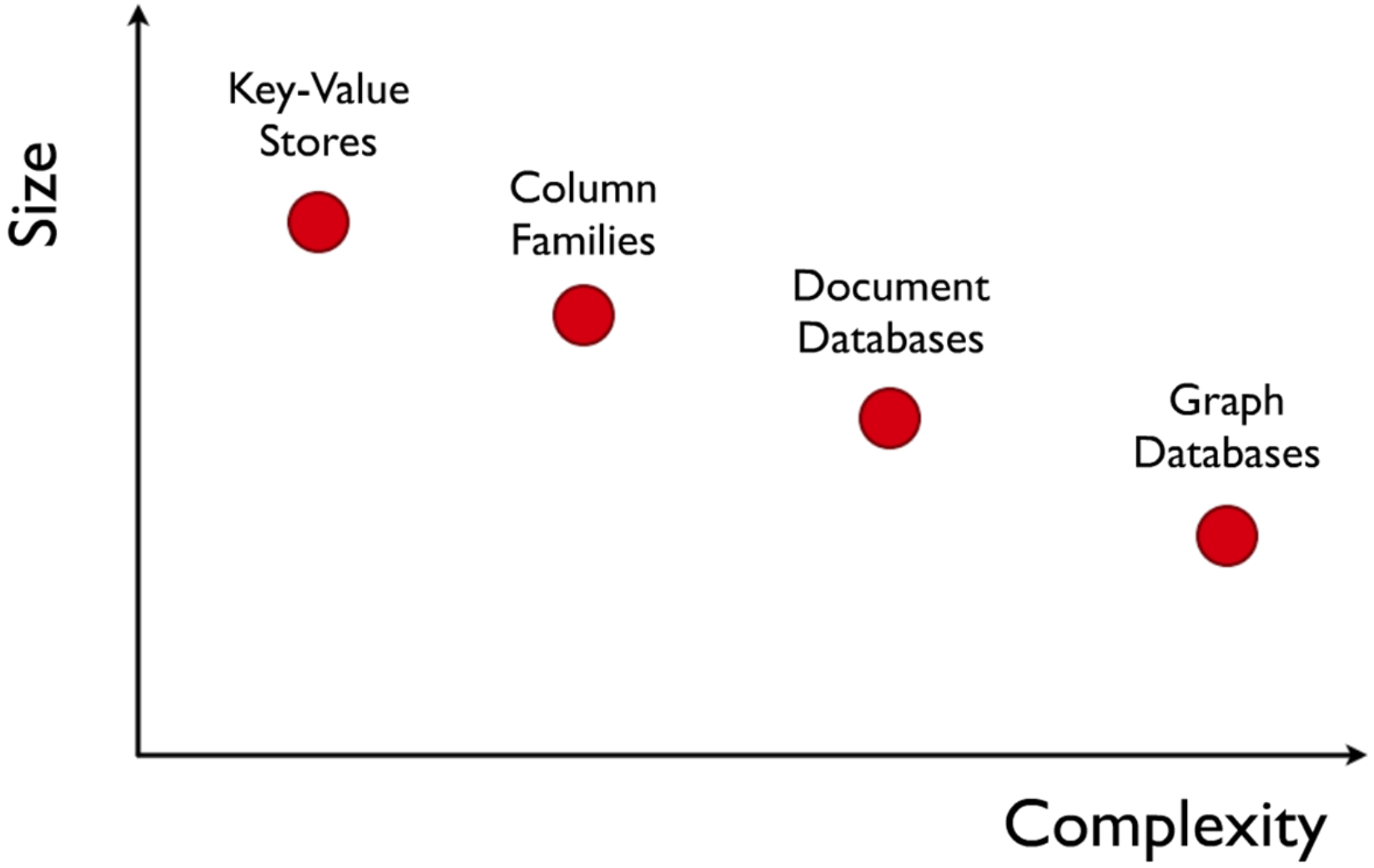
¿Qué saben ustedes de NoSQL?

373 systems in ranking, August 2021

Rank			DBMS	Database Model	Score		
Aug 2021	Jul 2021	Aug 2020			Aug 2021	Jul 2021	Aug 2020
1.	1.	1.	Oracle	Relational, Multi-model	1269.26	+6.59	-85.90
2.	2.	2.	MySQL	Relational, Multi-model	1238.22	+9.84	-23.36
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	973.35	-8.61	-102.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	577.05	-0.10	+40.28
5.	5.	5.	MongoDB	Document, Multi-model	496.54	+0.38	+52.98
6.	6.	↑ 7.	Redis	Key-value, Multi-model	169.88	+1.58	+17.01
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	165.46	+0.31	+3.01
8.	8.	8.	Elasticsearch	Search engine, Multi-model	157.08	+1.32	+4.76
9.	9.	9.	SQLite	Relational	129.81	-0.39	+3.00
10.	↑ 11.	10.	Microsoft Access	Relational	114.84	+1.39	-5.02
11.	↓ 10.	11.	Cassandra	Wide column	113.66	-0.35	-6.18
12.	12.	12.	MariaDB	Relational, Multi-model	98.98	+0.99	+8.06
13.	13.	13.	Splunk	Search engine	90.60	+0.55	+0.69
14.	14.	↑ 15.	Hive	Relational	83.93	+1.26	+8.64
15.	15.	↑ 17.	Microsoft Azure SQL Database	Relational, Multi-model	75.15	-0.06	+18.31
16.	16.	16.	Amazon DynamoDB	Multi-model	74.90	-0.30	+10.15
17.	17.	↓ 14.	Teradata	Relational, Multi-model	68.82	-0.13	-7.96
18.	18.	↑ 21.	Neo4j	Graph	56.95	-0.21	+6.77
19.	19.	19.	SAP HANA	Relational, Multi-model	55.57	+1.76	+2.46
20.	20.	20.	Solr	Search engine, Multi-model	51.06	-0.73	-0.63

http://db-engines.com/en/ranking

NoSQL


Size
Key-Value Stores
Column Families
Document Databases
Graph Databases
Complexity

NoSQL: No solo SQL ("Not only SQL")

- **Distribuido**
  - "Sharding": partición "horizontal" de datos
  - Replicación
  - Garantías diferentes a ACID

- A menudo **lenguajes más simples** que SQL
  - APIs no estandarizadas
  - Más trabajo para la aplicación

- **Sabores diferentes** (para escenarios diferentes)
  - Garantías diferentes
  - Perfiles diferentes de escalabilidad
  - Funcionalidades de consulta diferentes
  - Modelos de datos diferentes

Limitaciones de almacenamiento distribuido: Teorema CAP

Pero primero … ACID

Para bases de datos tradicionales sin distribución …

1. **A**tomicity:
   - Atomicidad: Las transacciones son todo o nada
2. **C**onsistency:
   - Coherencia: No rompe las restricciones / reglas
3. **I**solation:
   - Aislamiento: Transacciones paralelas actúan como si fueran secuenciales
4. **D**urability
   - Durabilidad: El sistema recuerda los cambios

¿Qué es CAP?

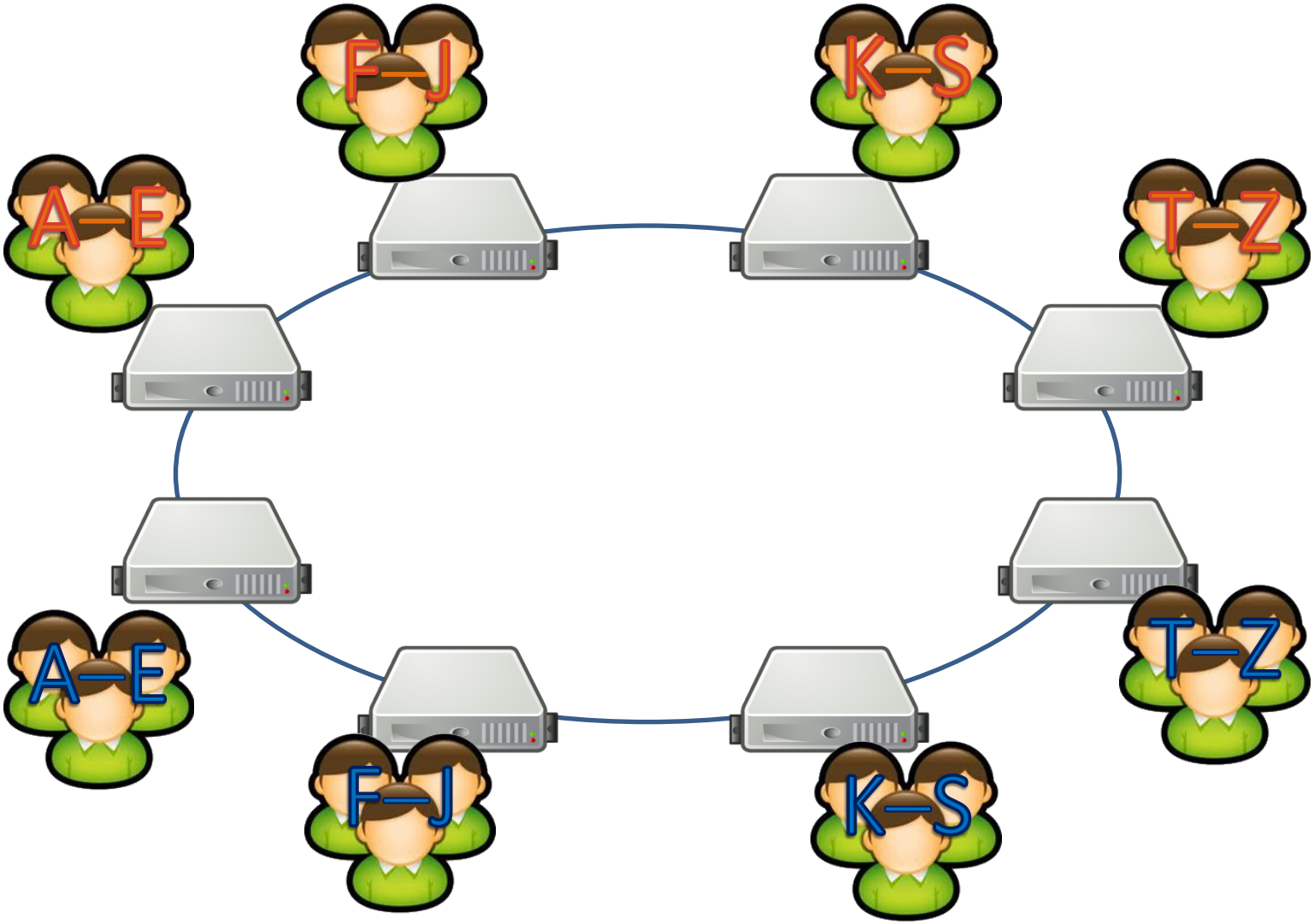
Tres garantías que un sistema distribuido podría ofrecer

1. **C**onsistency:
   - Coherencia: Todos los nodos tienen una visión coherente del sistema
2. **A**vailability:
   - Disponibilidad: Toda petición de escritura/lectura es atendida
3. **P**artition-tolerance:
   - Tolerancia a particiones: El sistema funciona incluso si hay pérdida de mensajes

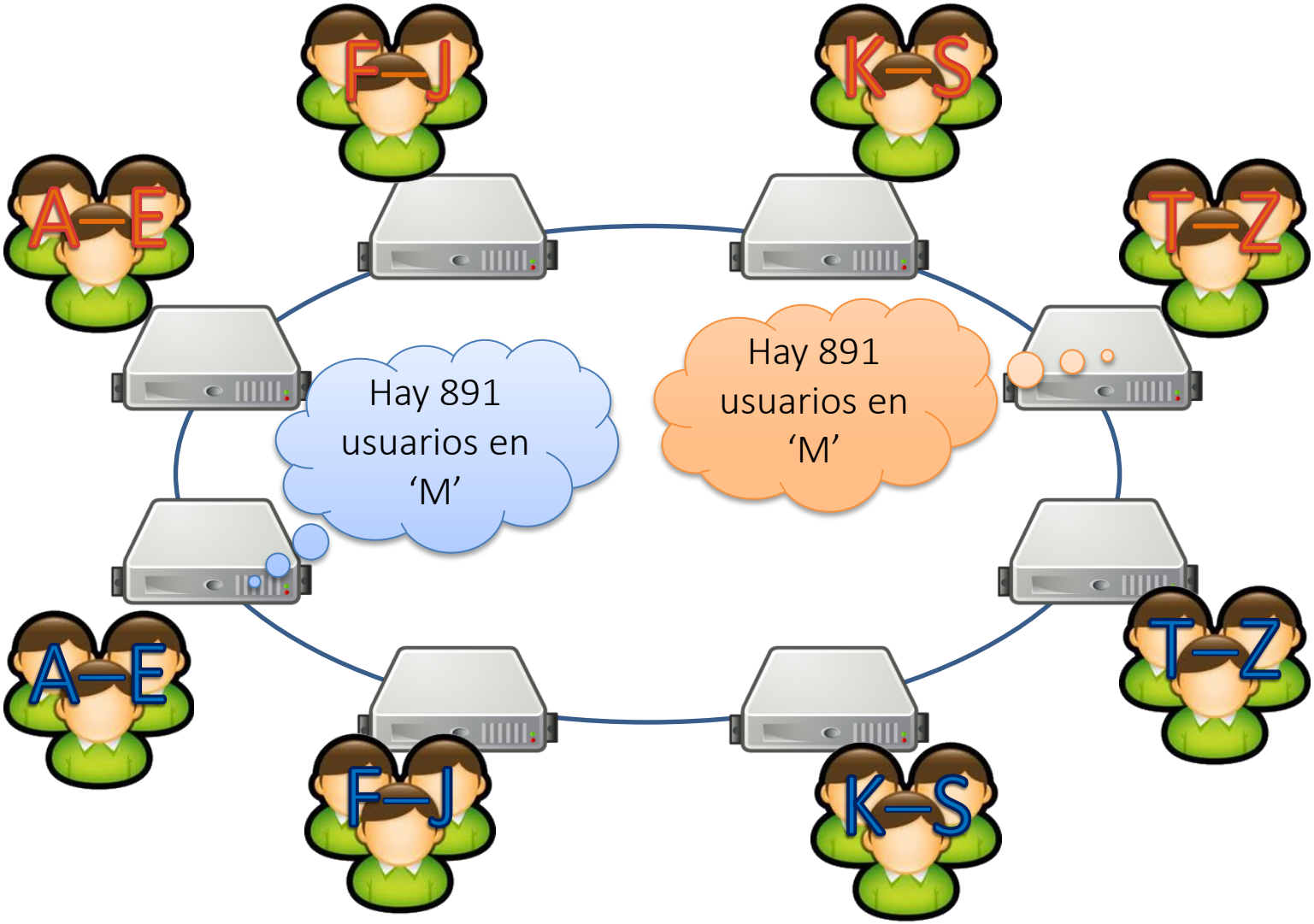
¡El CA en CAP no es el mismo CA que en ACID! 

¡Partición de la red, no de los datos! 

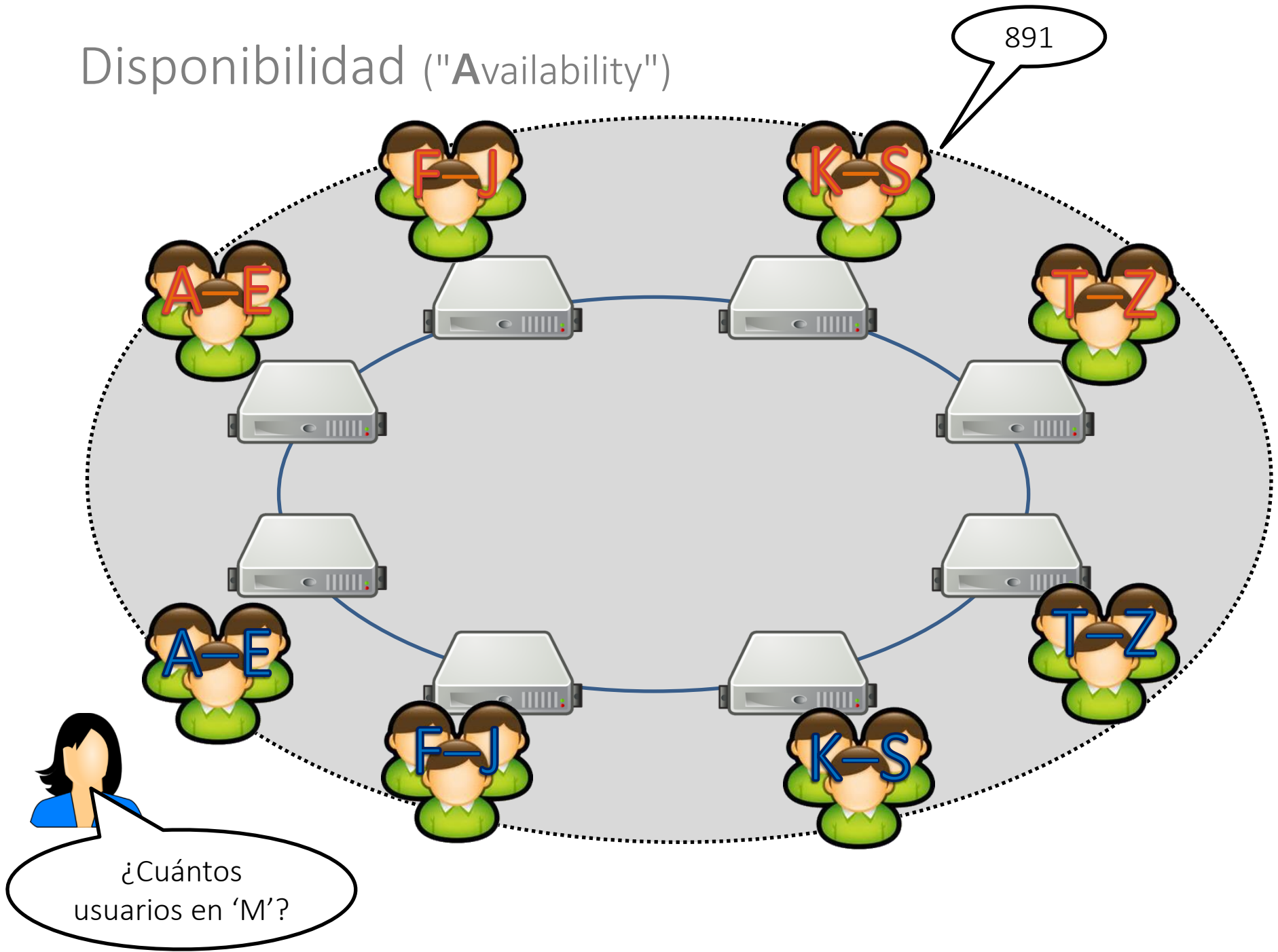
Un sistema distribuido (con Replicación)



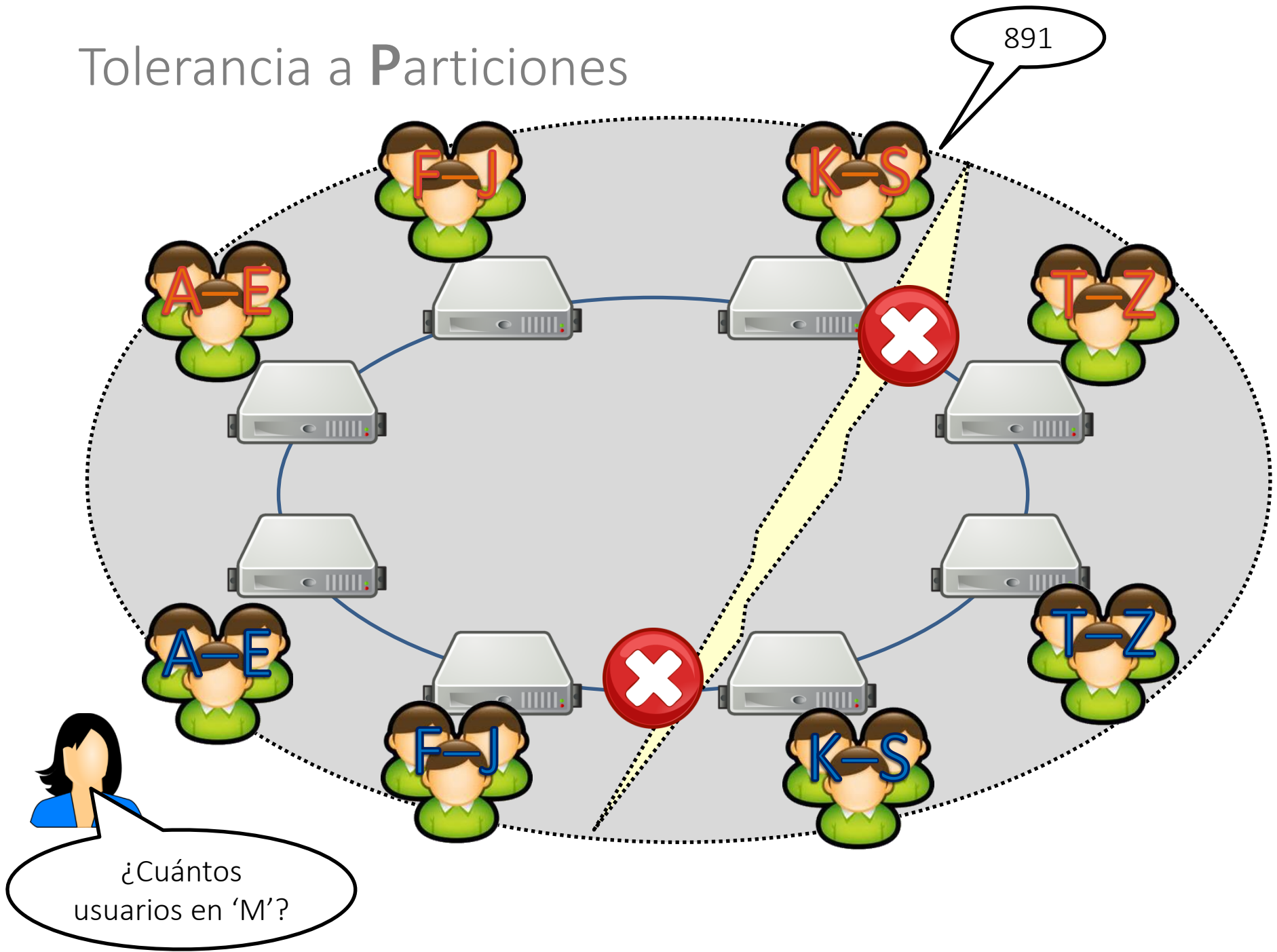
Coherencia


F—J
K—S
A—E
T—Z
Hay 891 usuarios en 'M'
Hay 891 usuarios en 'M'
A—E
T—Z
F—J
K—S

Disponibilidad ("**A**vailability")



Tolerancia a **P**articiones



La pregunta **CAP**

¿Puede un sistema distribuido garantizar:

Coherencia (todos los nodos tienen una visión actualizada),

disponibilidad/**A**vailability (toda escritura/lectura es atendida) y

tolerancia a **P**articiones (el sistema funciona incluso si hay mensajes perdidos)

al mismo tiempo?

¿Qué opinas? 

La respuesta CAP

El teorema **CAP**

¡Un sistema distribuido **no puede** garantizar:

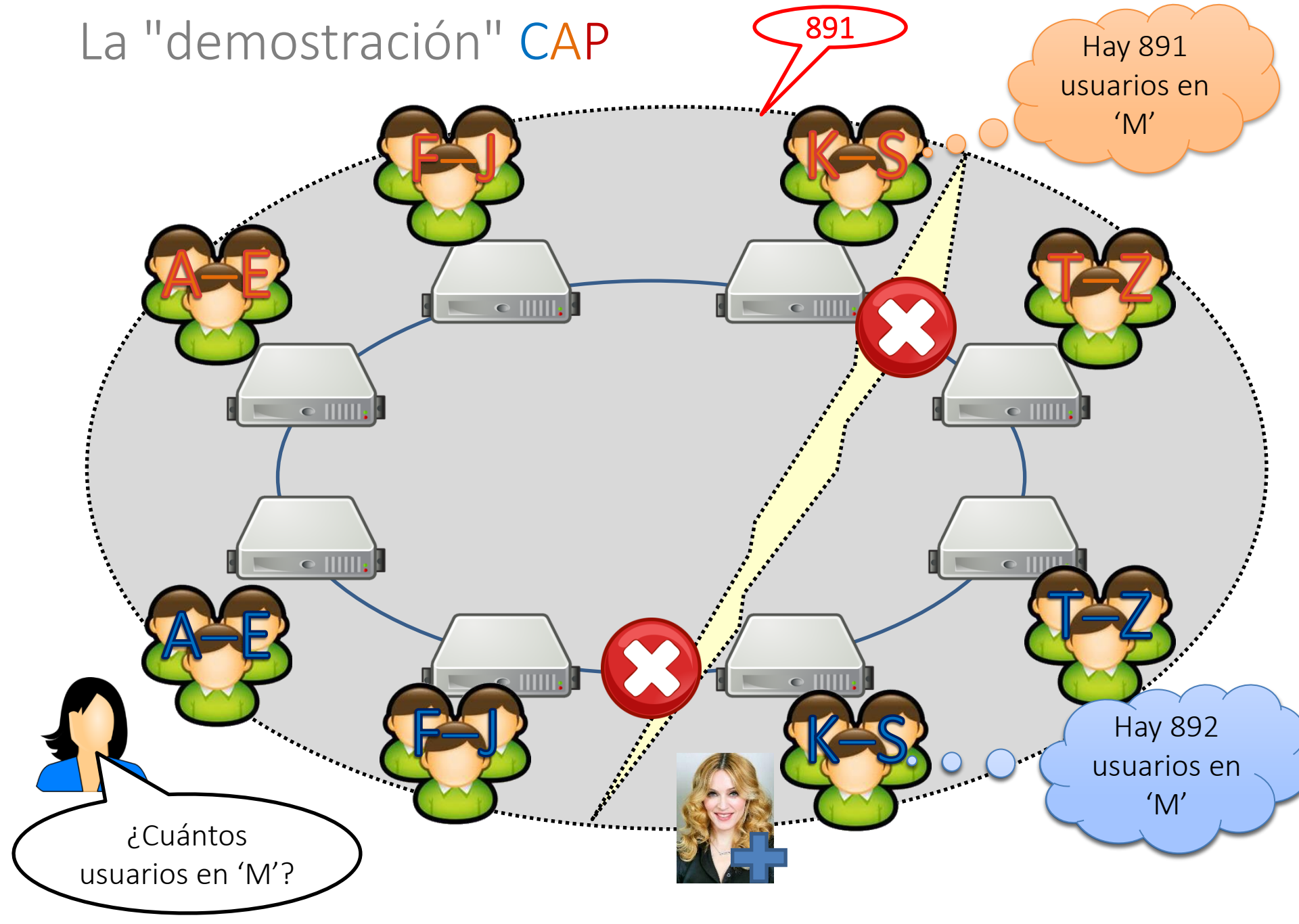
Coherencia (todos los nodos tienen una visión actualizada),

disponibilidad/**A**vailability (toda escritura/lectura es atendida) y

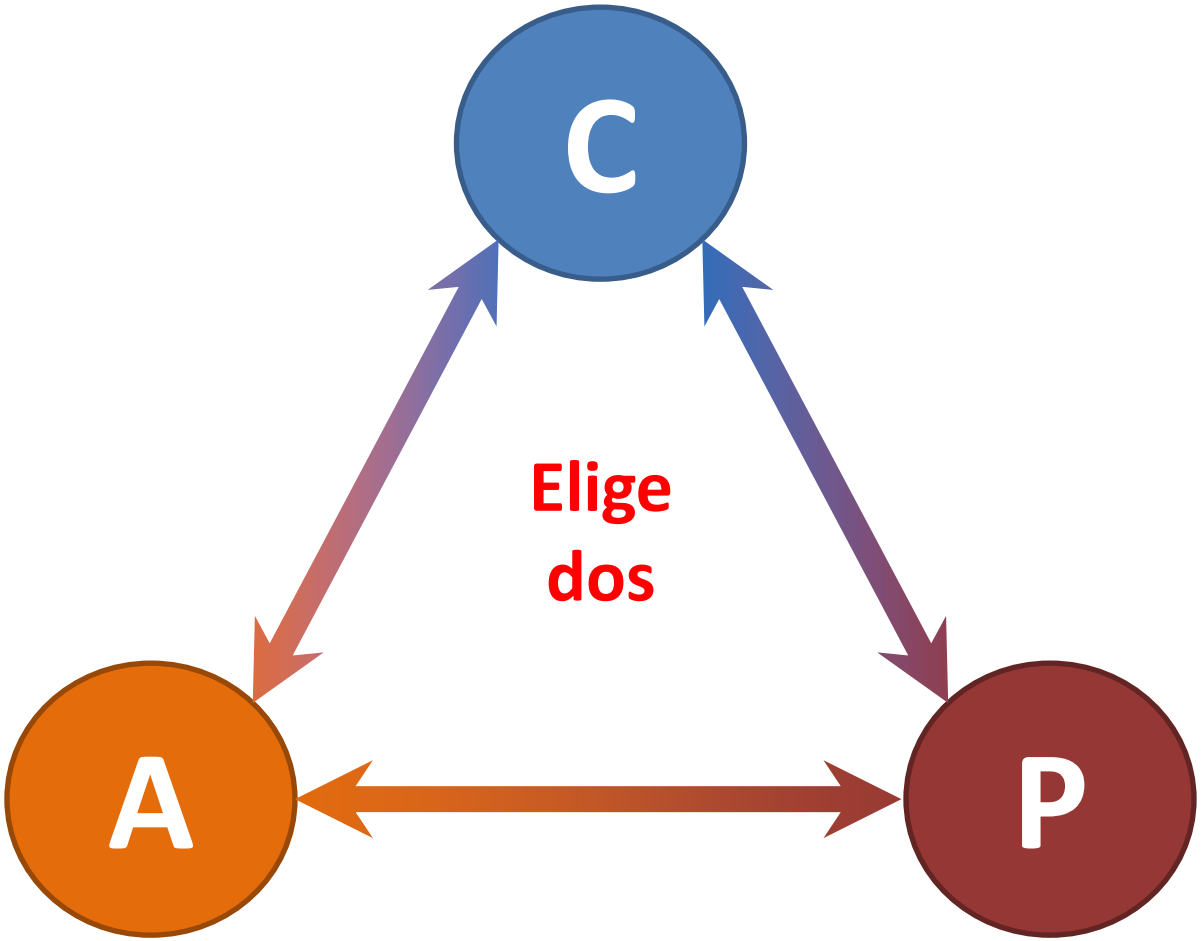
tolerancia a Particiones (el sistema funciona incluso si hay mensajes perdidos)

al mismo tiempo!

La "demostración" CAP



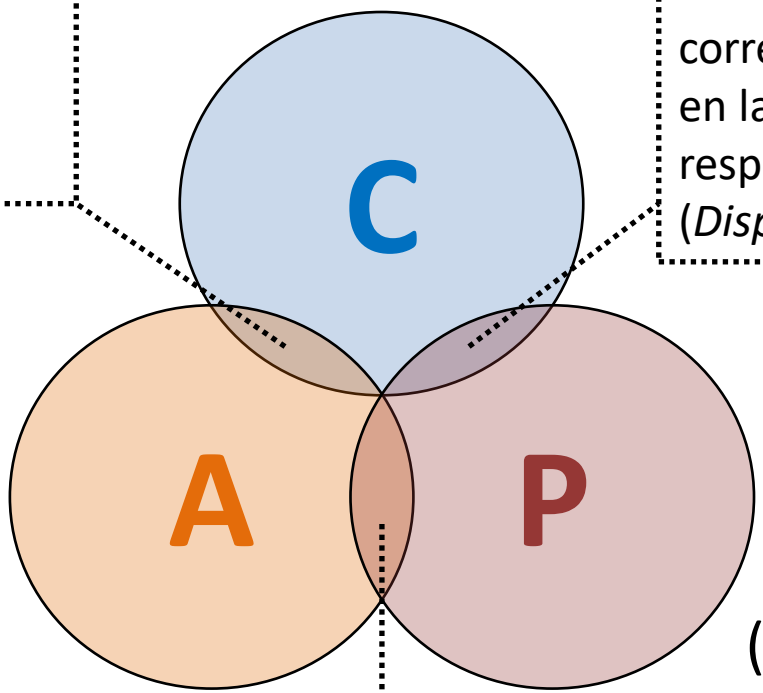
El triángulo CAP



Sistemas CAP

CA: Garantiza respuestas correctas sólo mientras la conexión funcione bien (*Centralizado / Tradicional*)

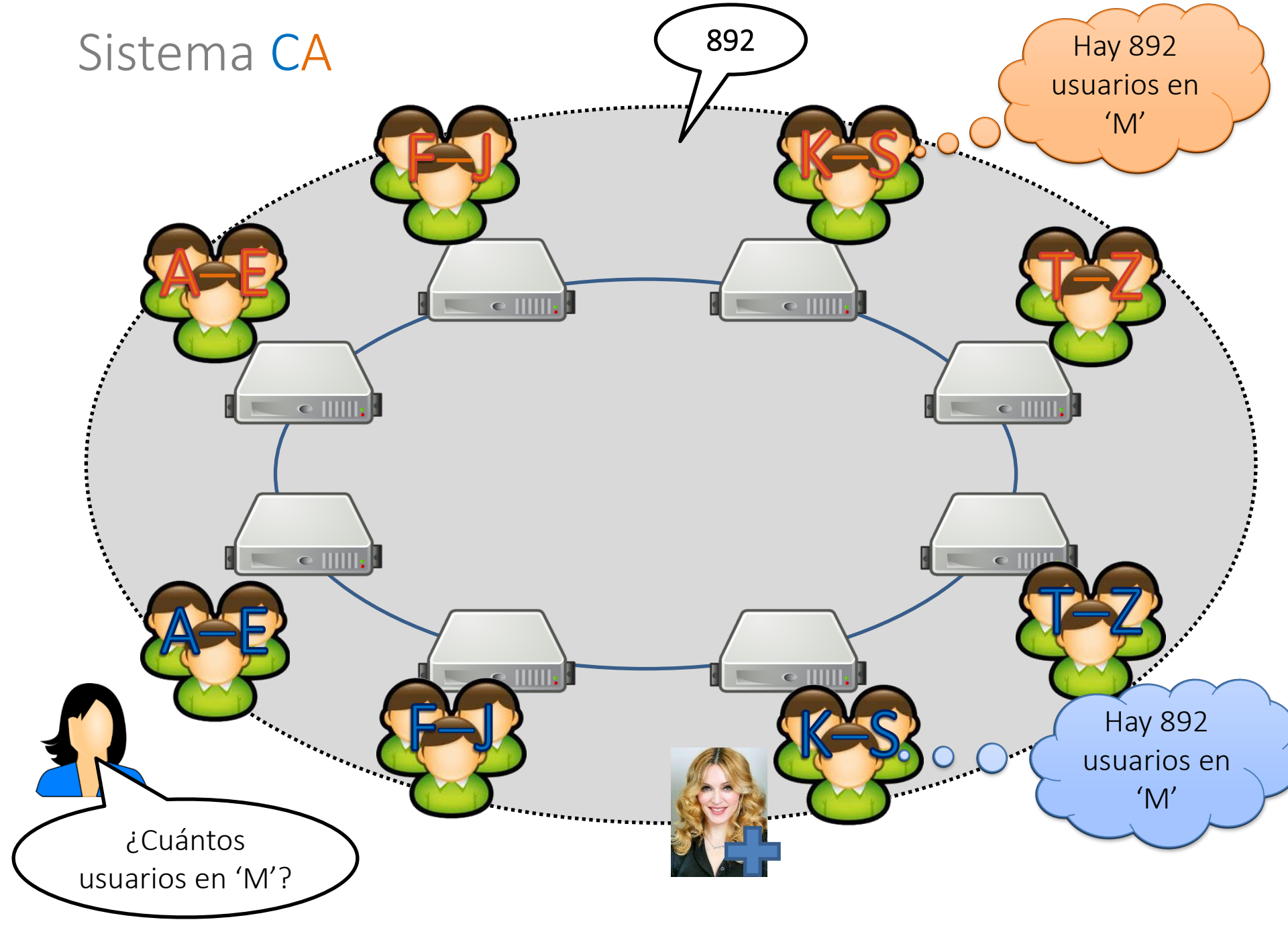
CP: Garantiza respuestas correctas incluso si hay fallas en la conexión, pero la respuesta puede fracasar (*Disponibilidad debíl*)



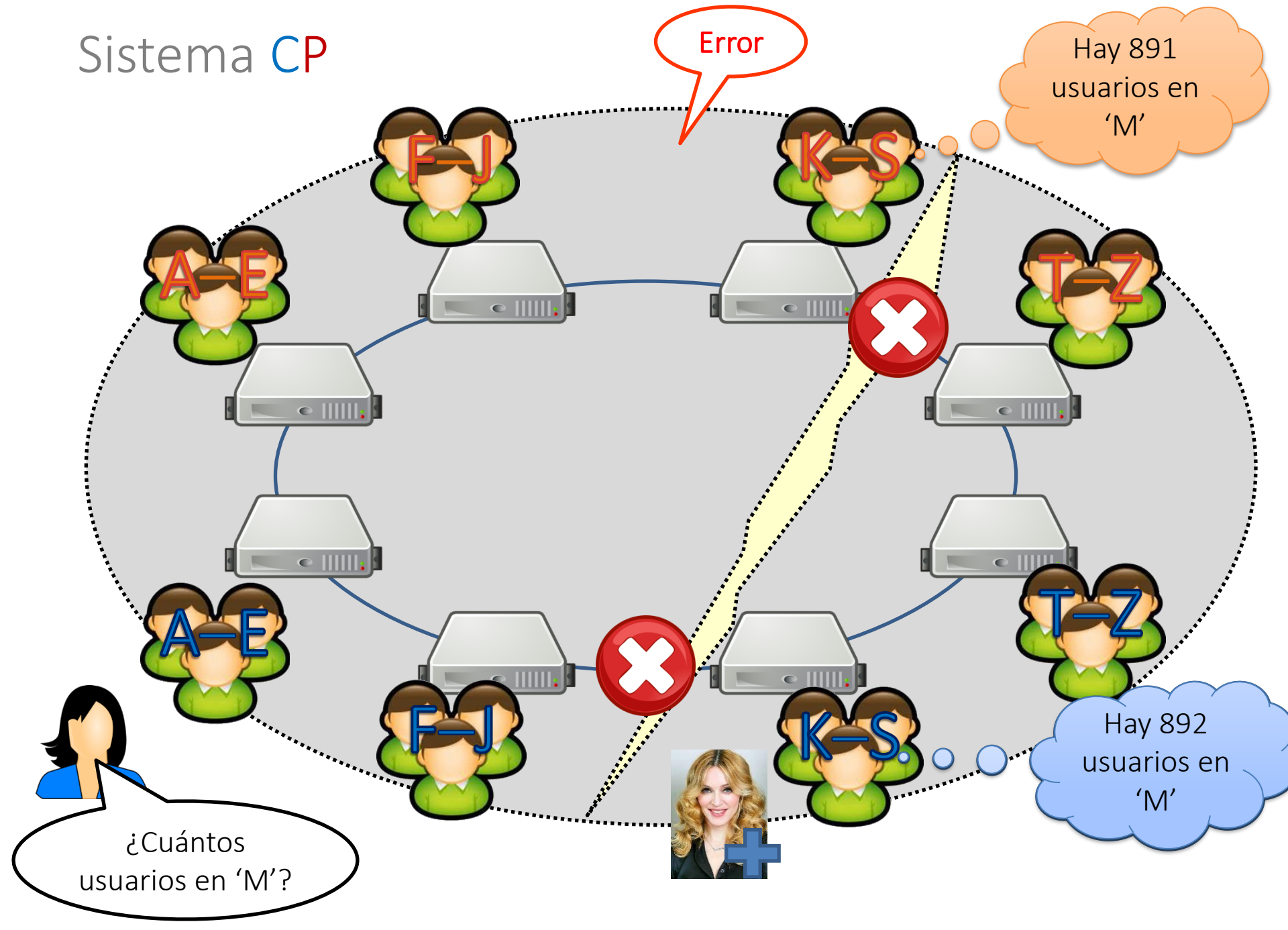
(No hay intersección)

AP: Siempre provee "la mejor" respuesta que puede incluso en presencia de fallas en la conexión (*Coherencia eventual*)

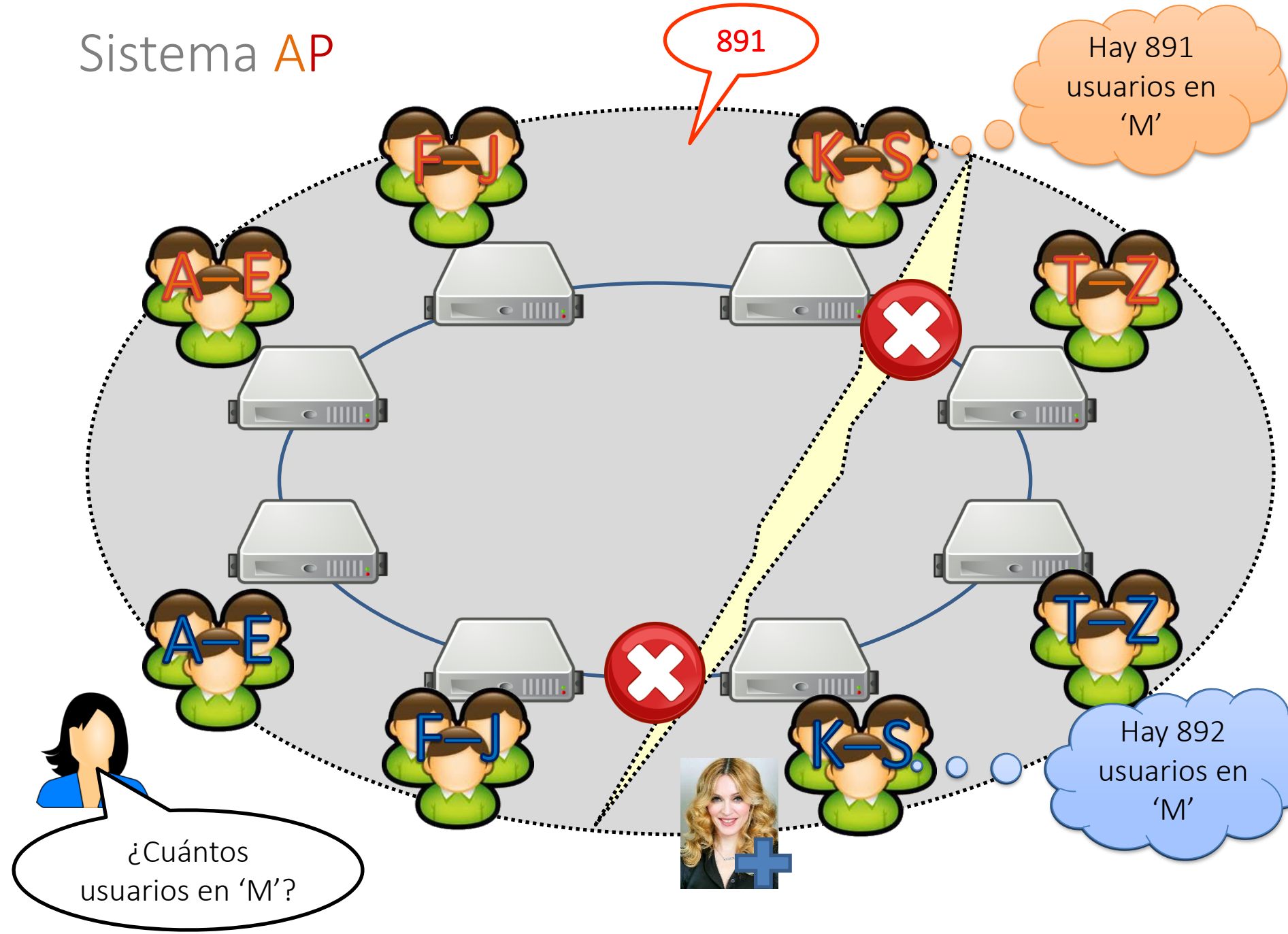
Sistema CA


892
Hay 892 usuarios en 'M'
F–J
K–S
A–E
T–Z
A–E
T–Z
F–J
K–S
Hay 892 usuarios en 'M'
¿Cuántos usuarios en 'M'?

Sistema CP


Error
Hay 891 usuarios en 'M'
F–J
K–S
A–E
T–Z
A–E
T–Z
F–J
K–S
Hay 892 usuarios en 'M'
¿Cuántos usuarios en 'M'?

Sistema AP


891
Hay 891 usuarios en 'M'
F–J
K–S
A–E
T–Z
A–E
T–Z
F–J
K–S
Hay 892 usuarios en 'M'
¿Cuántos usuarios en 'M'?

BASE (AP)

- **B**asically **A**vailable (básicamente disponible)
  - Casi siempre funciona (está operando)
- **S**oft State (estado virtualizado)
  - Información replicada / en caché
- **E**ventual Consistency (coherencia eventual)
  - Se toleran, por un tiempo, datos obsoletos

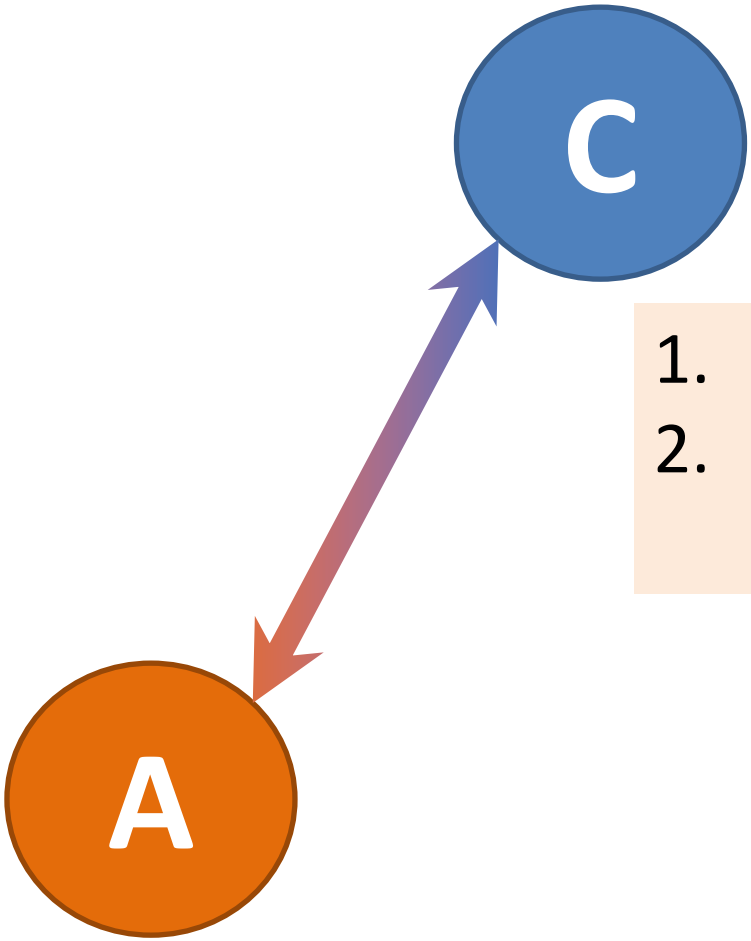
¿De qué manera Twitter opera como un sistema BASE (AP)?

Lady Gaga crea una “partición” de la red

Los usuarios pueden ver los retweets de los tweets de las celebridades antes del tweet original.

Más tarde cuando llega el tweet original, el timeline se ordenará de manera coherente.

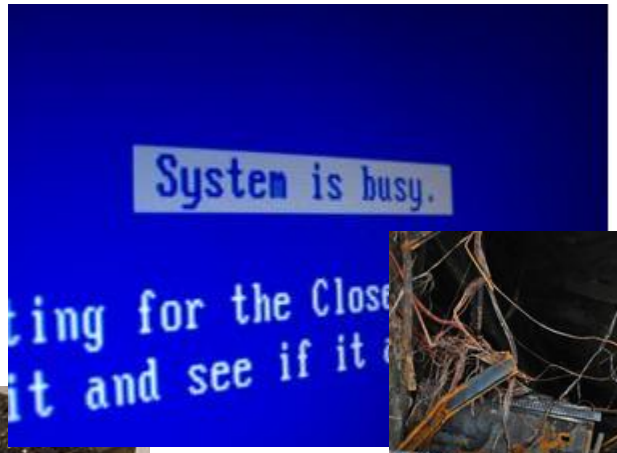
CAP en la práctica



1. Fijar P
2. Elegir un punto intermedio entre C y A

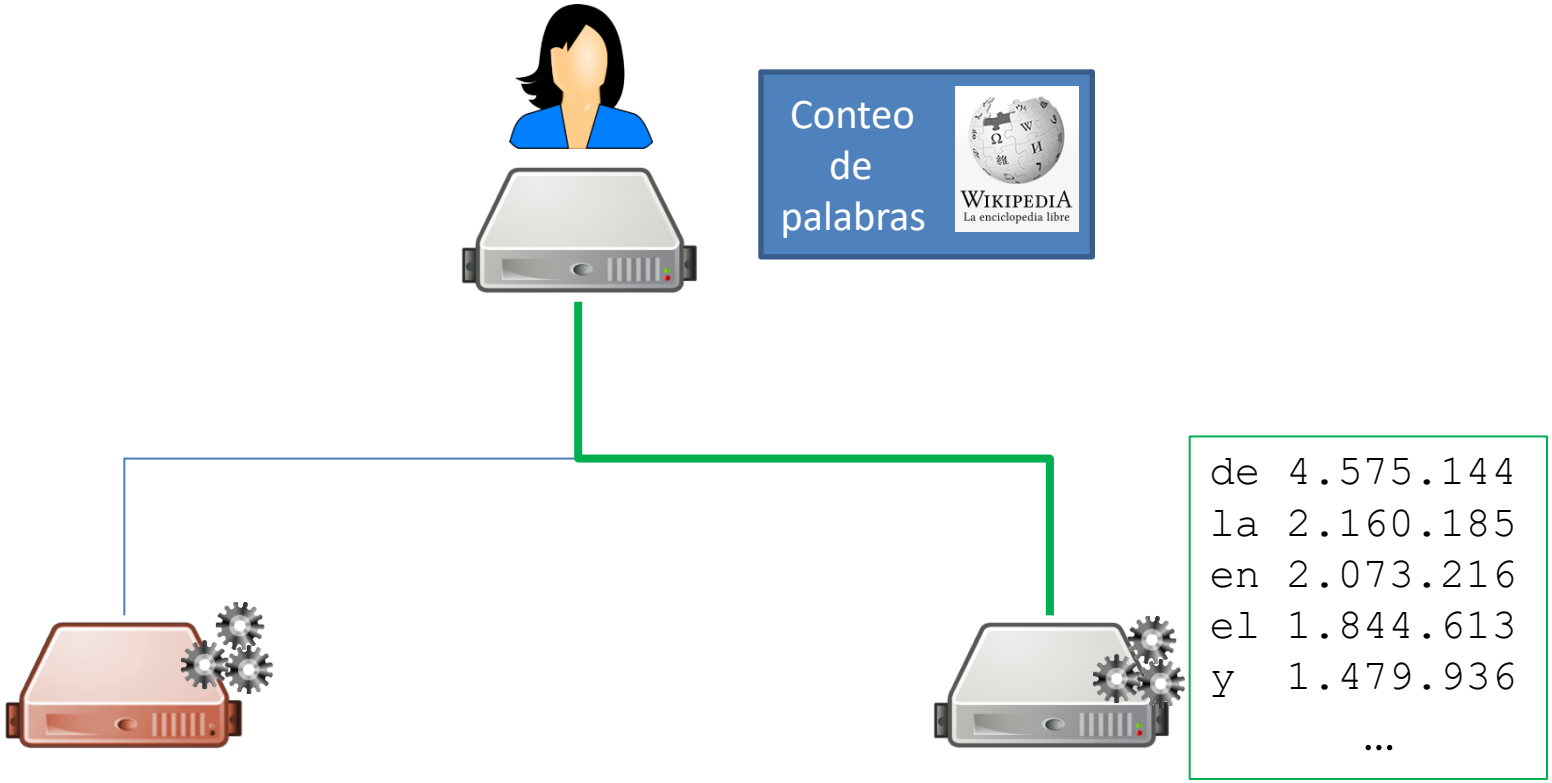
Tolerancia a particiones

Fallas



Falla "Fail–Stop"

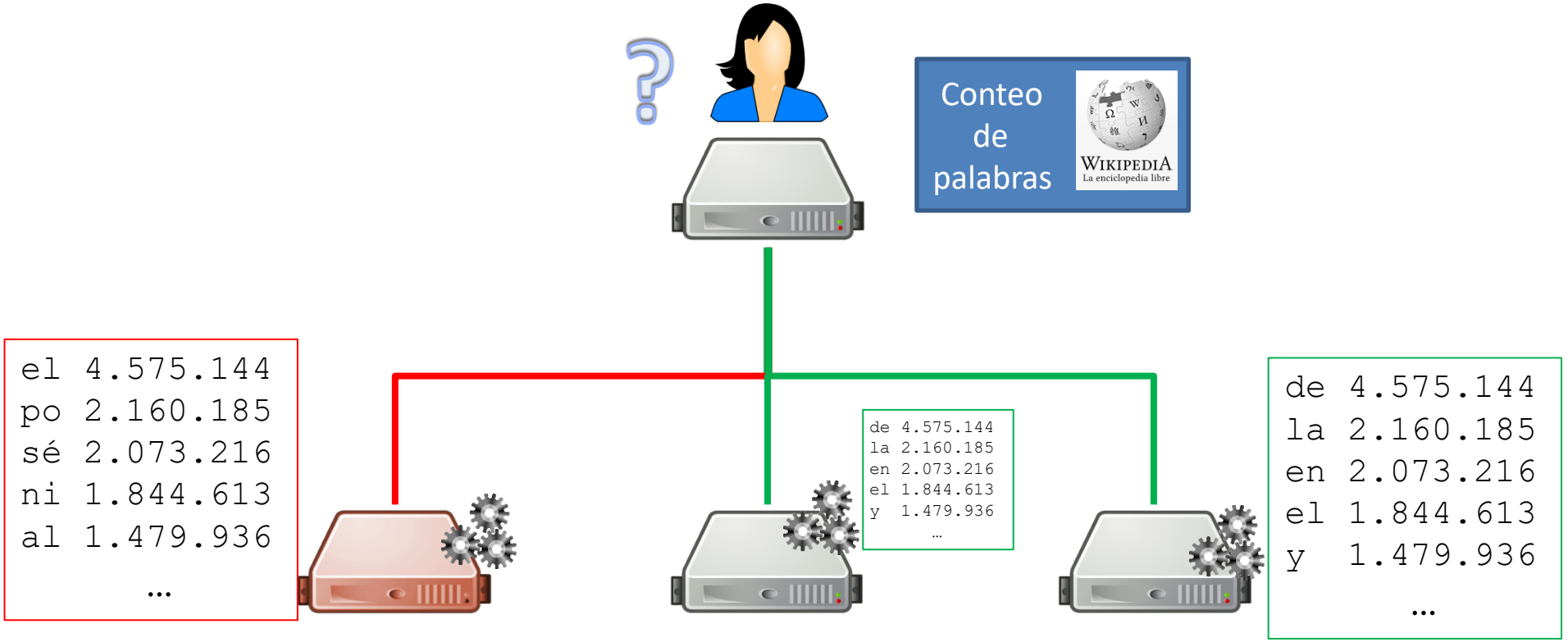
- Una máquina no responde (a tiempo)
  - a menudo, por falla del hardware o por sobrecarga
  - se necesita al menos $f + 1$ máquinas replicadas
    - f = número de fallas "fail–stop"



Falla Bizantina

- Una máquina responde de forma incorrecta

¿Cuántas máquinas operativas necesitamos para corregir una falla bizantina?

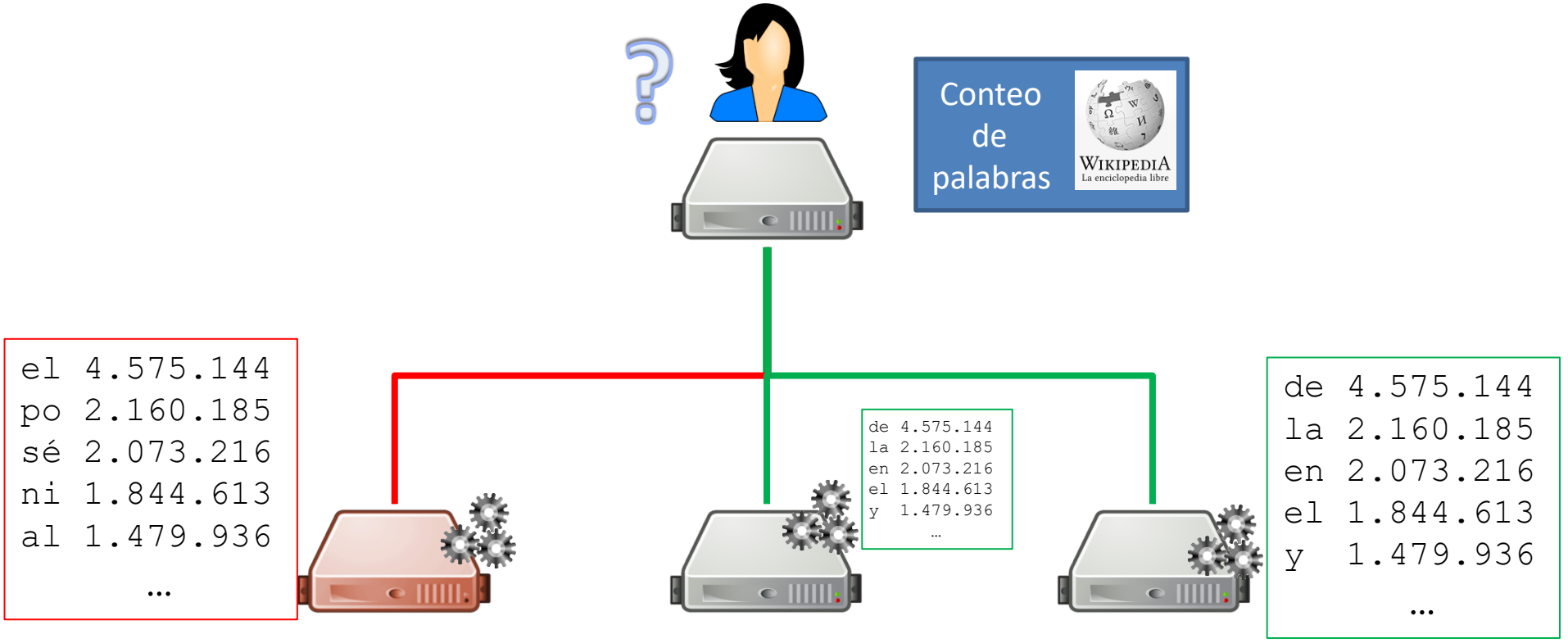


Falla Bizantina

- Una máquina responde de forma incorrecta

¿Cuántas máquinas operativas necesitamos para corregir una falla bizantina?

Se necesitan **2*f*+1** máquinas replicadas
- *f* = número de fallas (posiblemente) bizantinas

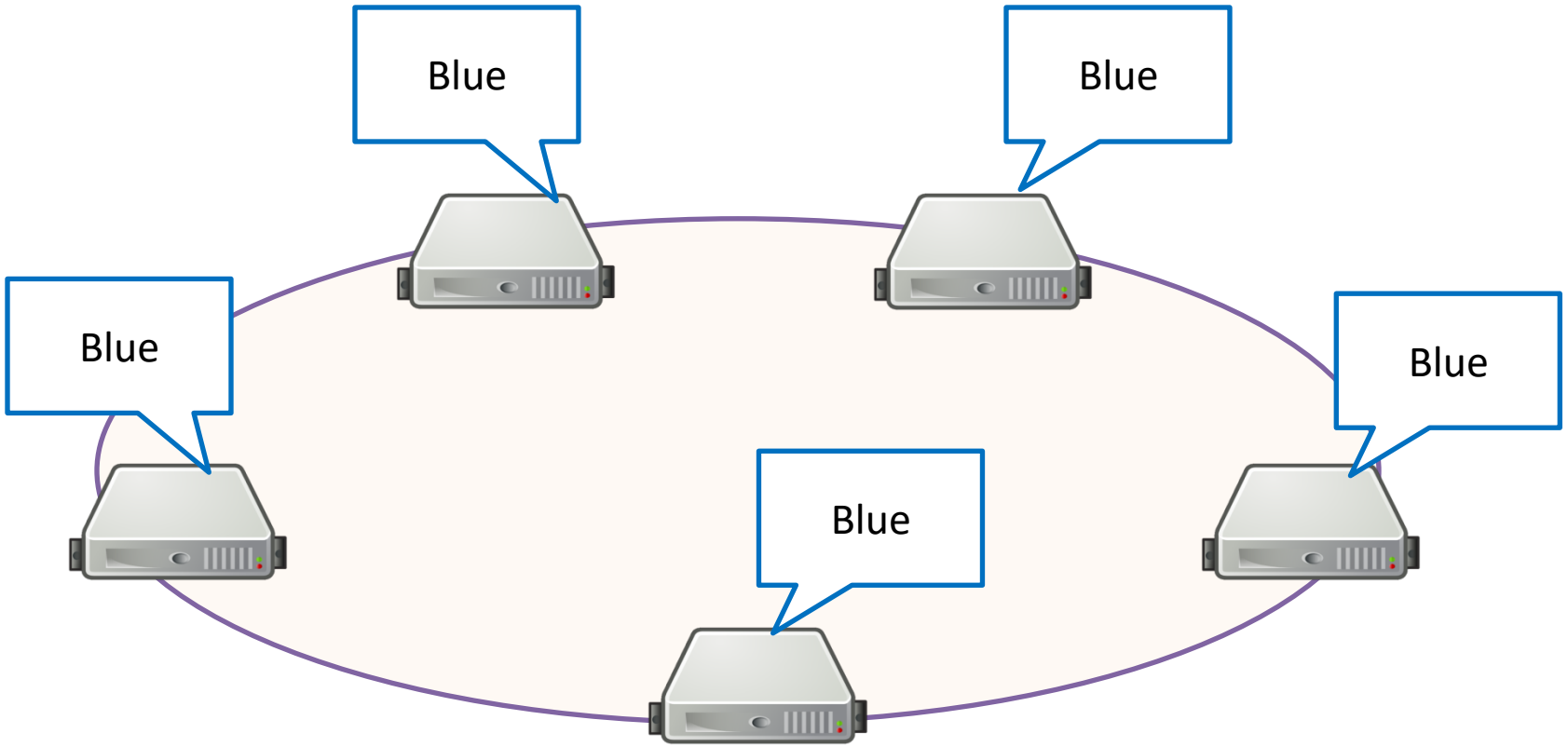


¿Consenso distribuido?

¿Color del vestido?

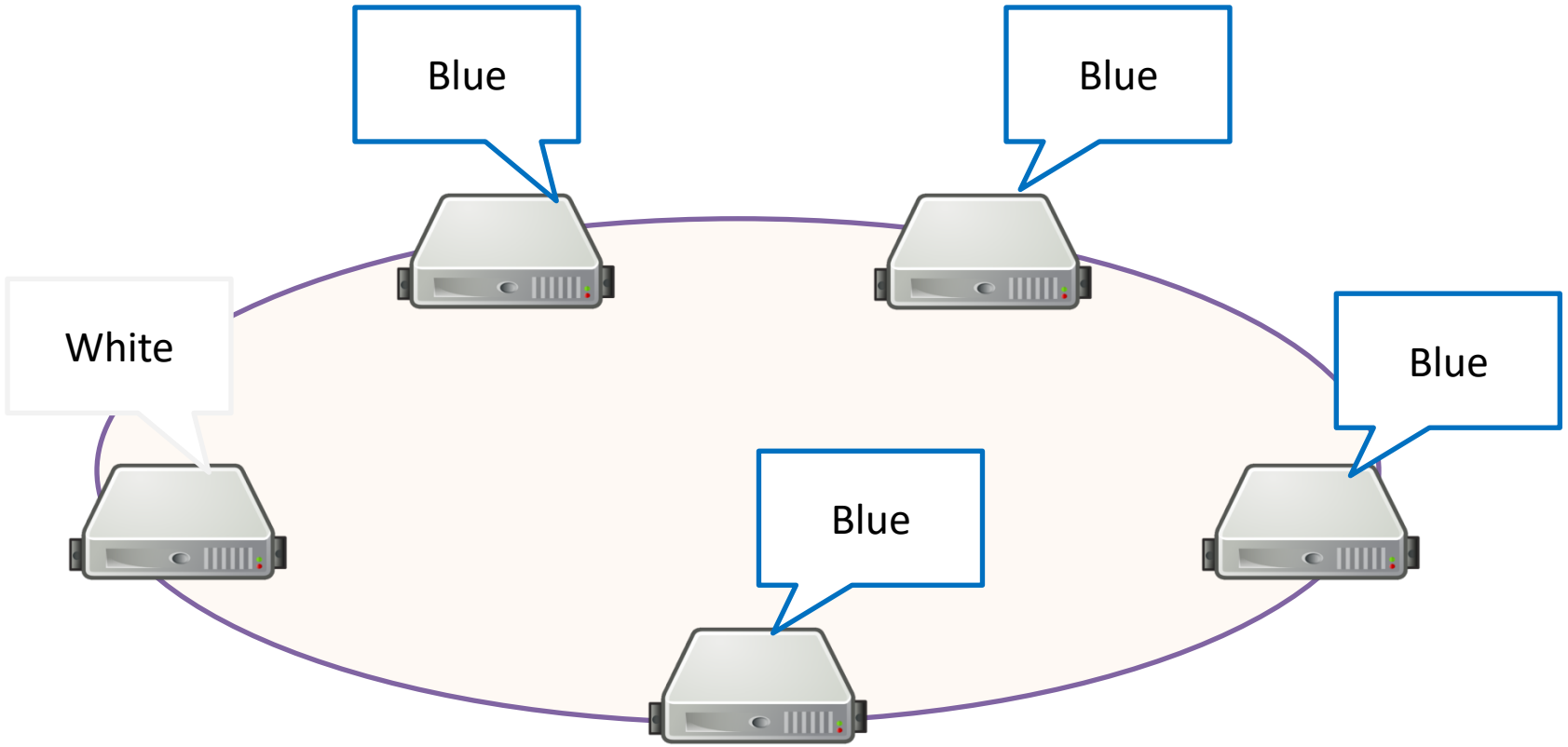
Consenso distribuido

Consenso fuerte: Todos los nodos deben concordar



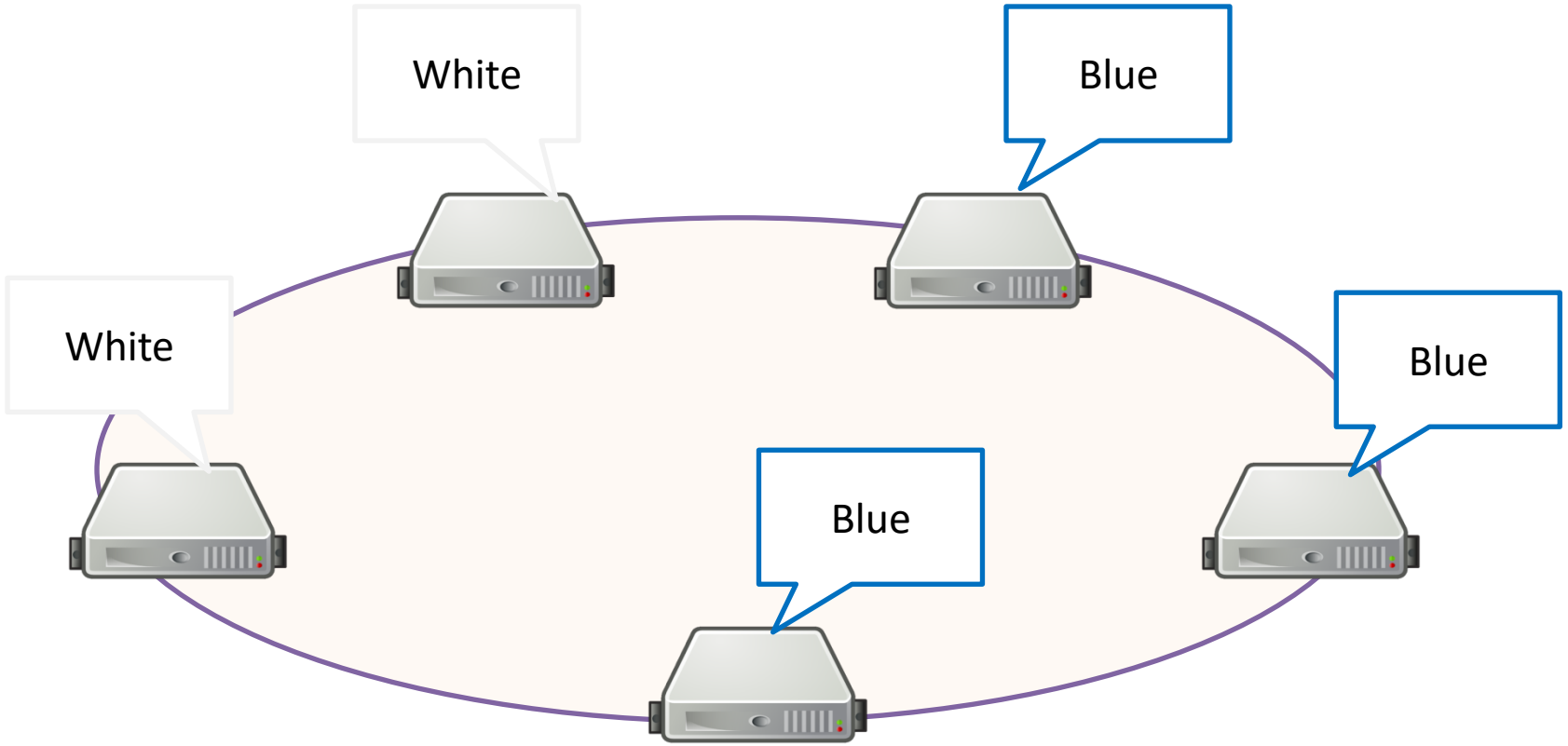
Consenso distribuido

Consenso fuerte: Todos los nodos deben concordar



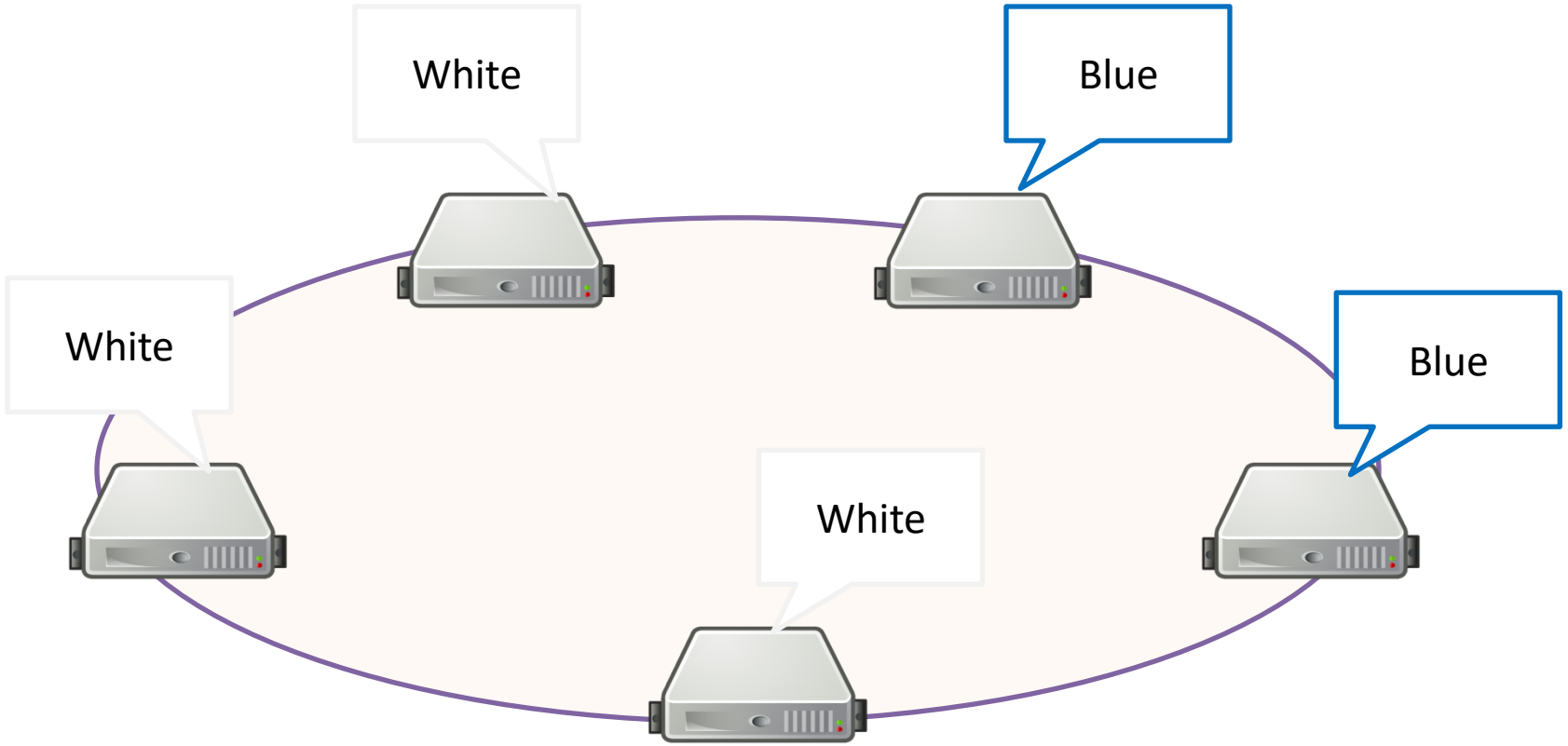
Consenso distribuido

Consenso mayoritario: Debe concordar una mayoría



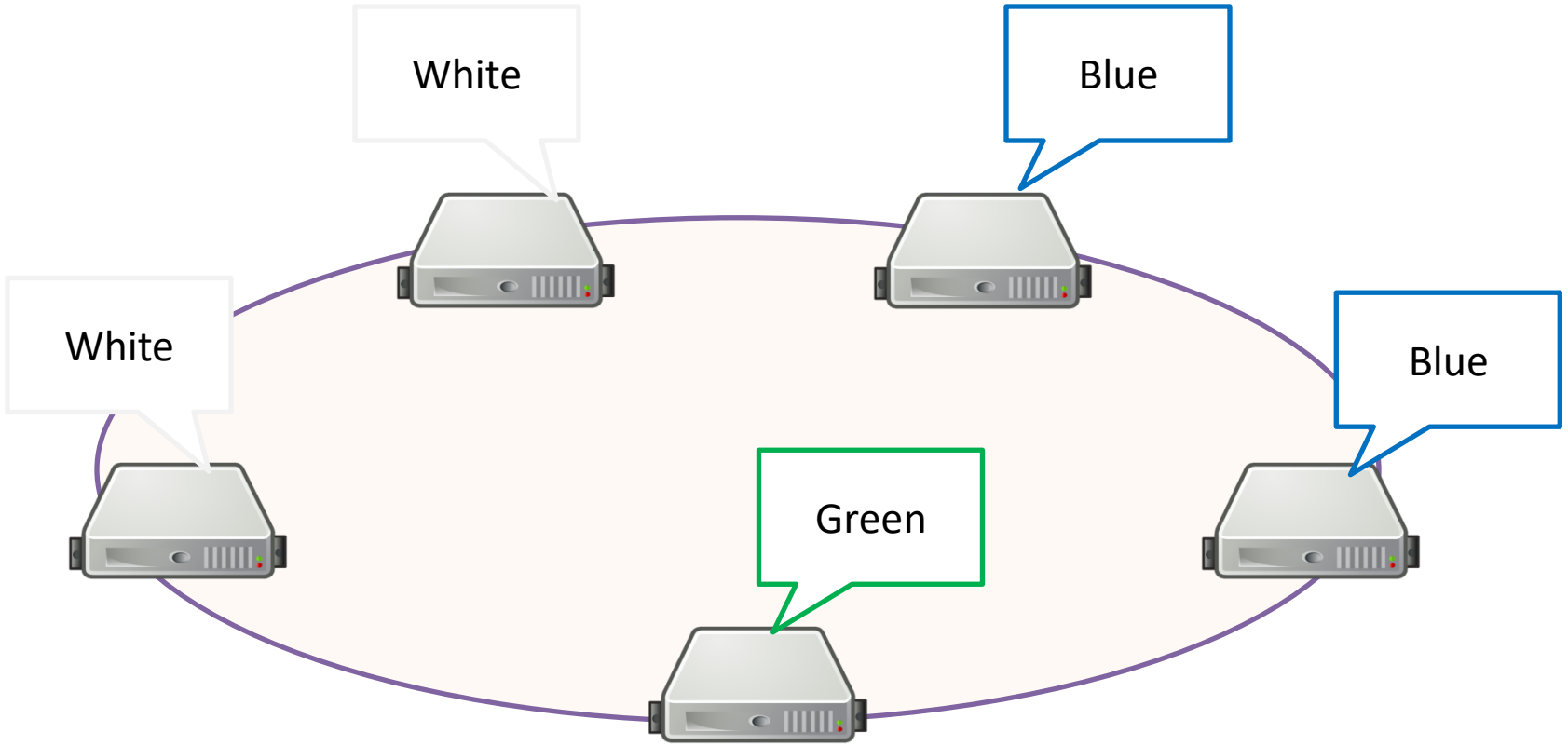
Consenso distribuido

Consenso mayoritario: Debe concordar una mayoría



Consenso distribuido

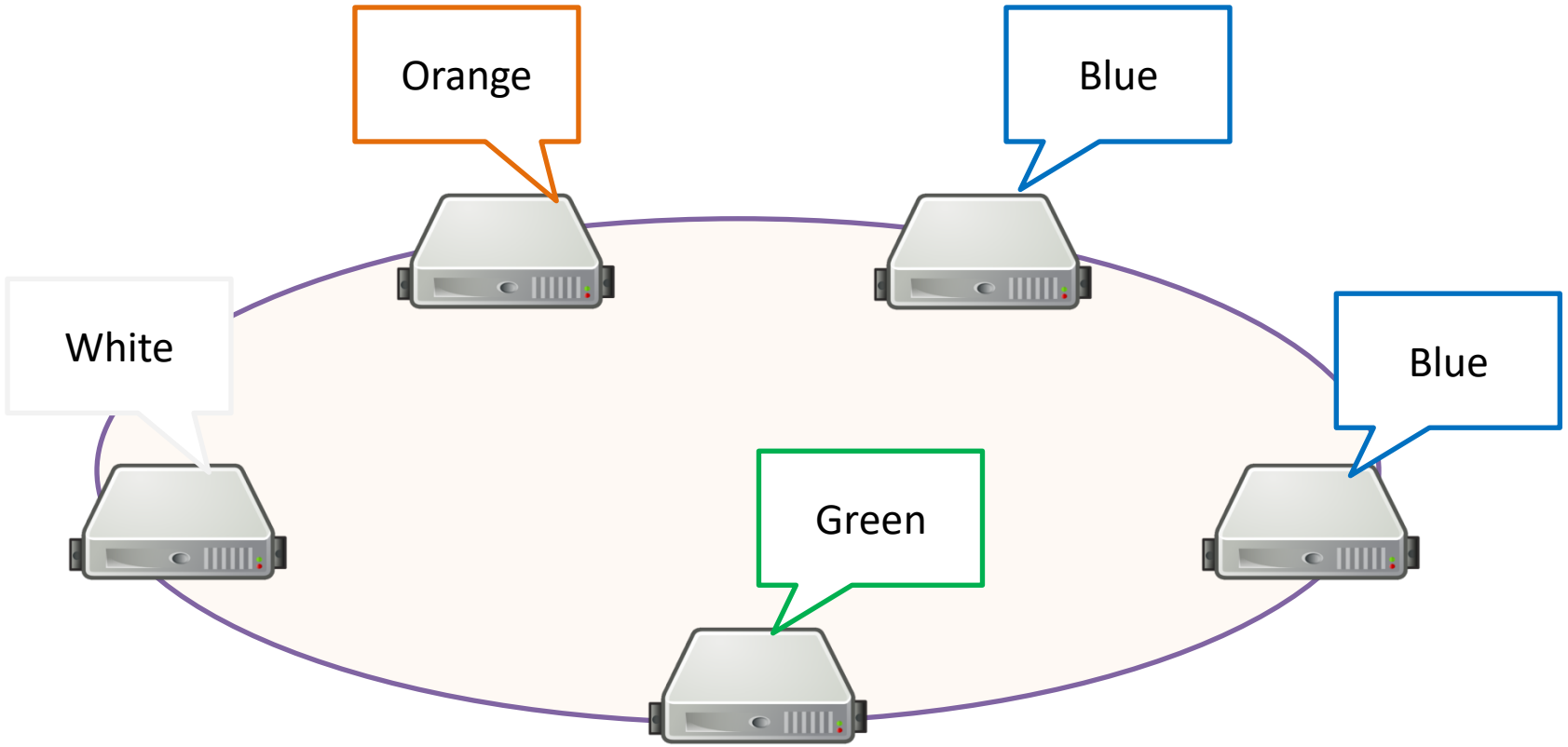
Consenso mayoritario: Debe concordar una mayoría



Sin consenso.

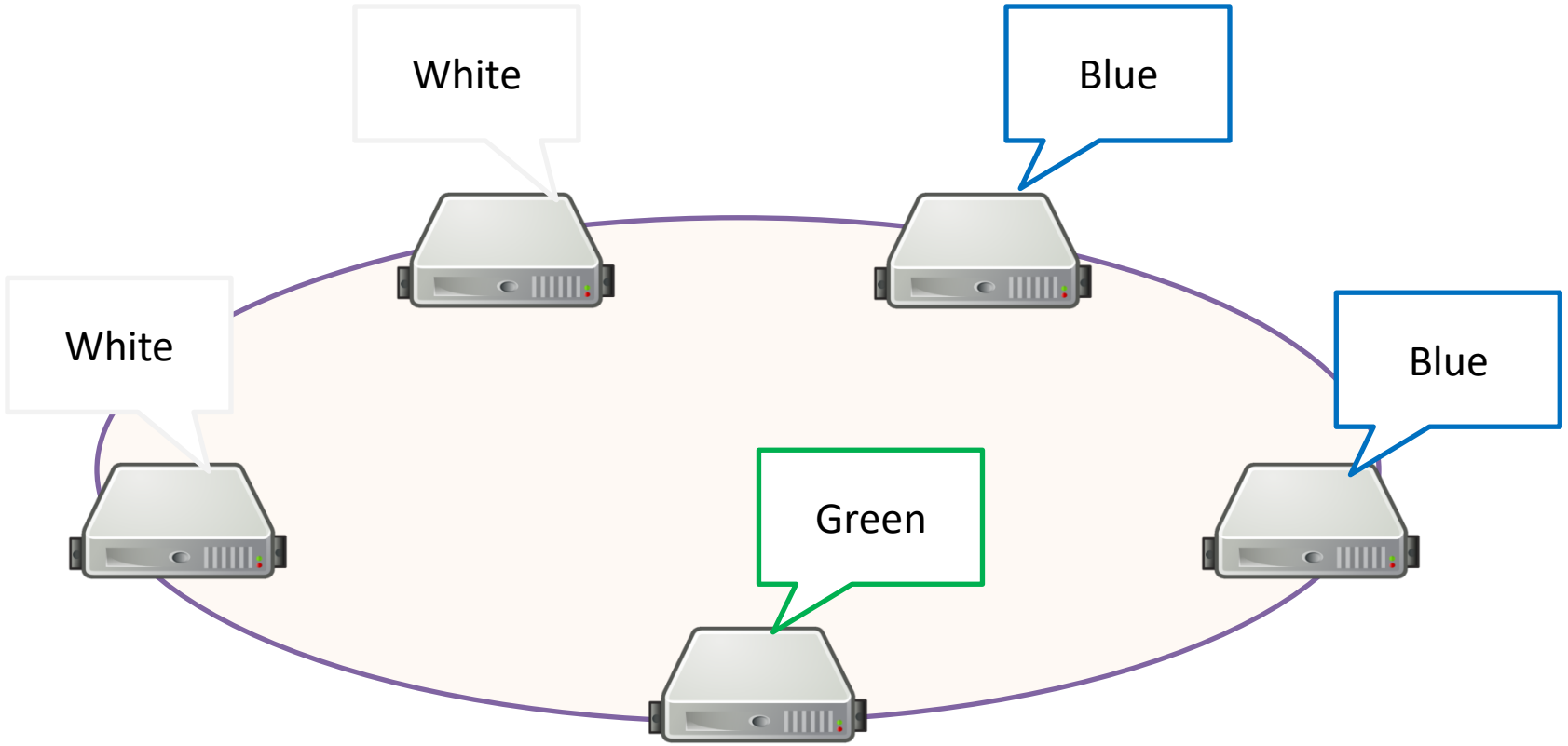
Consenso distribuido

Consenso plural: Debe concordar una pluralidad



Consenso distribuido

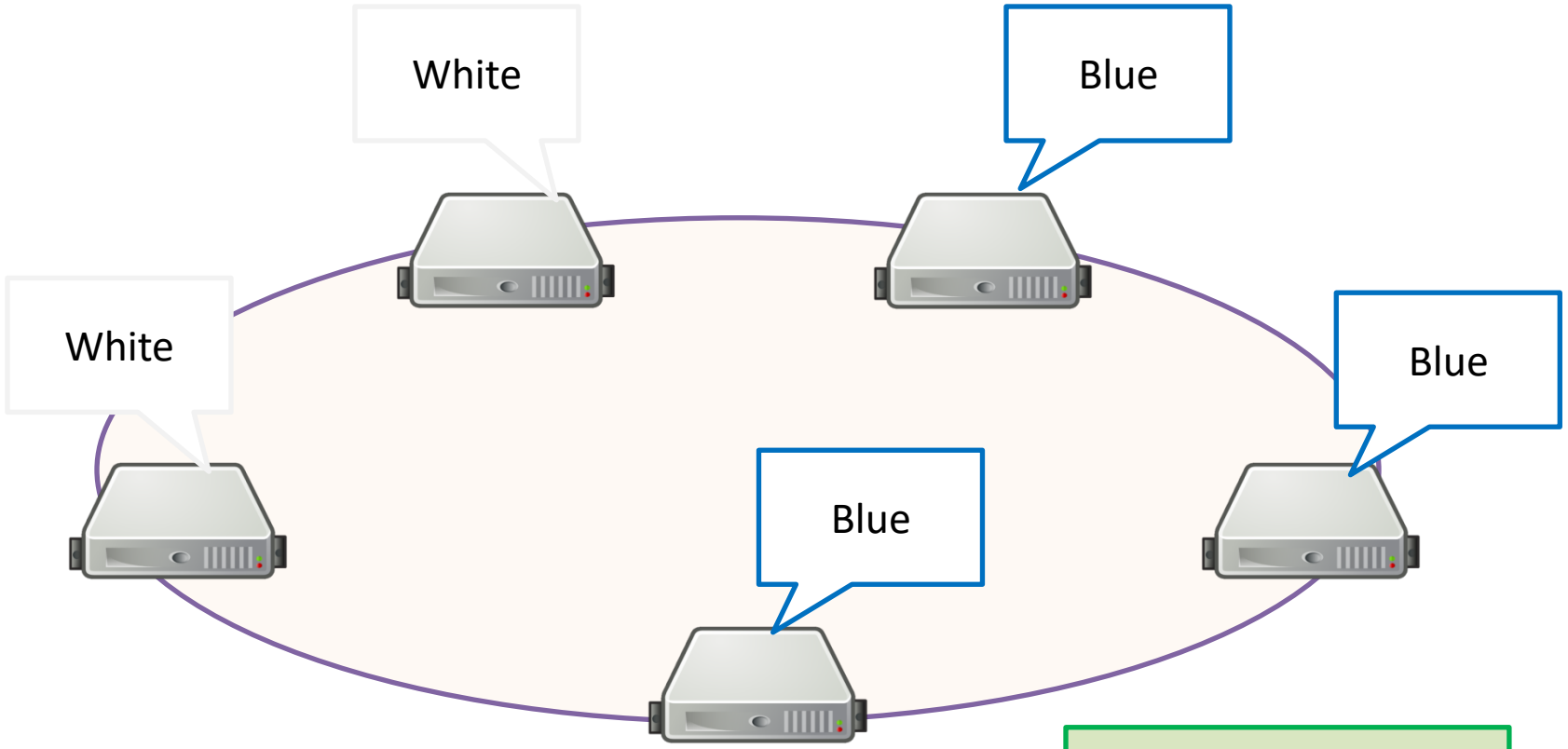
Consenso plural: Debe concordar una pluralidad



Sin consenso.

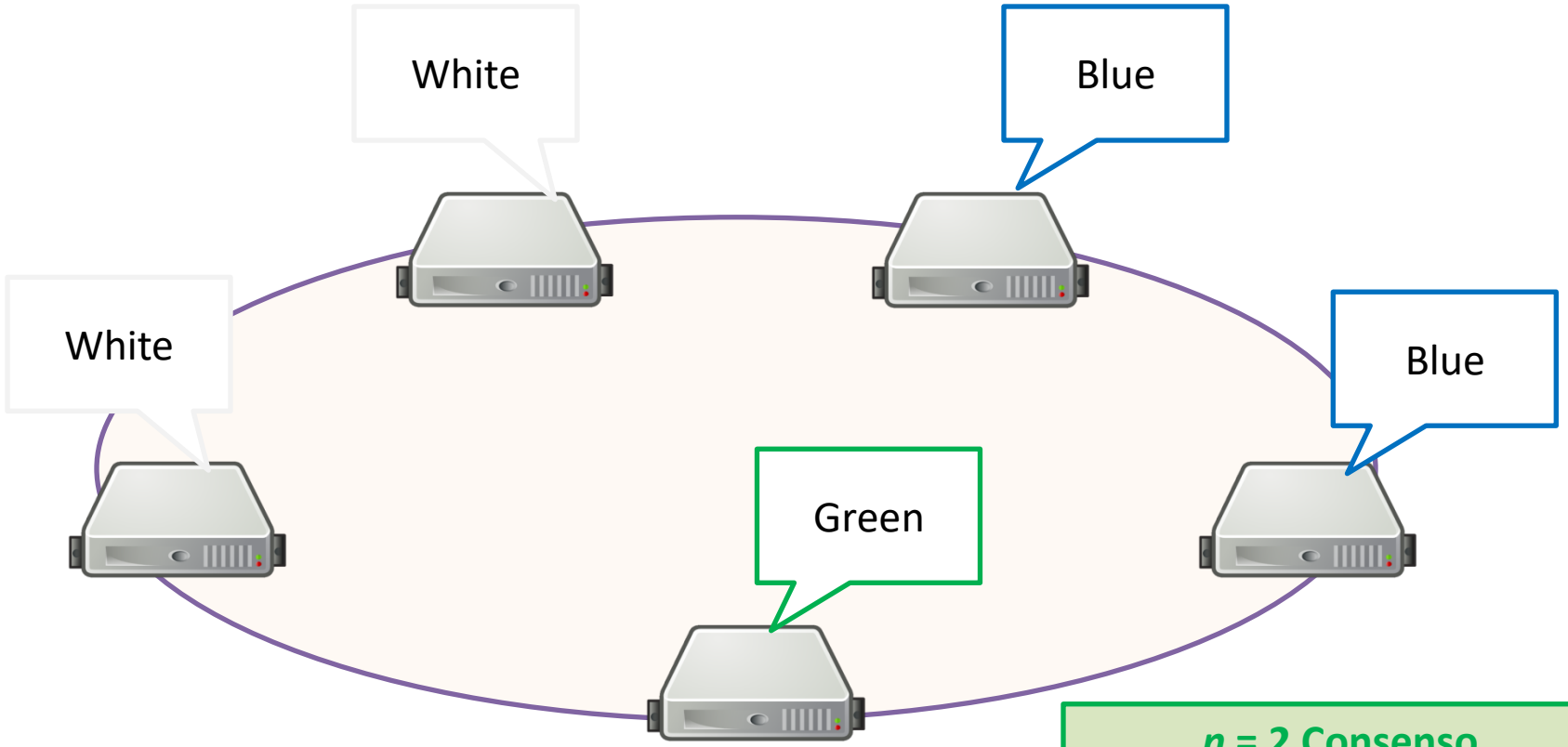
Consenso distribuido

Consenso de quorum: n nodos deben concordar



Consenso distribuido

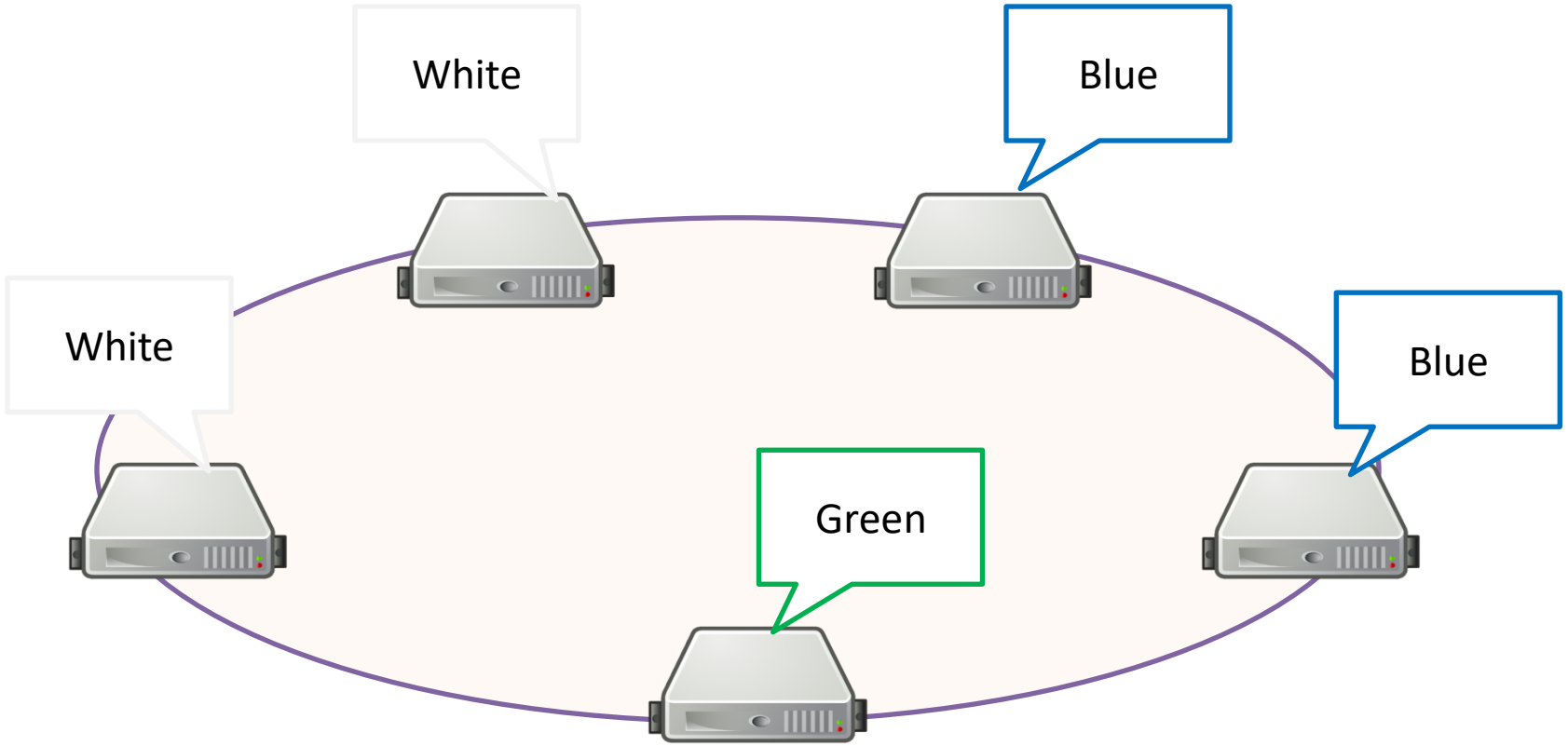
Consenso de quorum: n nodos deben concordar



***n* = 2 Consenso.**
(Primeras 2 máquinas consultas pero no es único!)

Consenso distribuido

Consenso de quorum: n nodos deben concordar

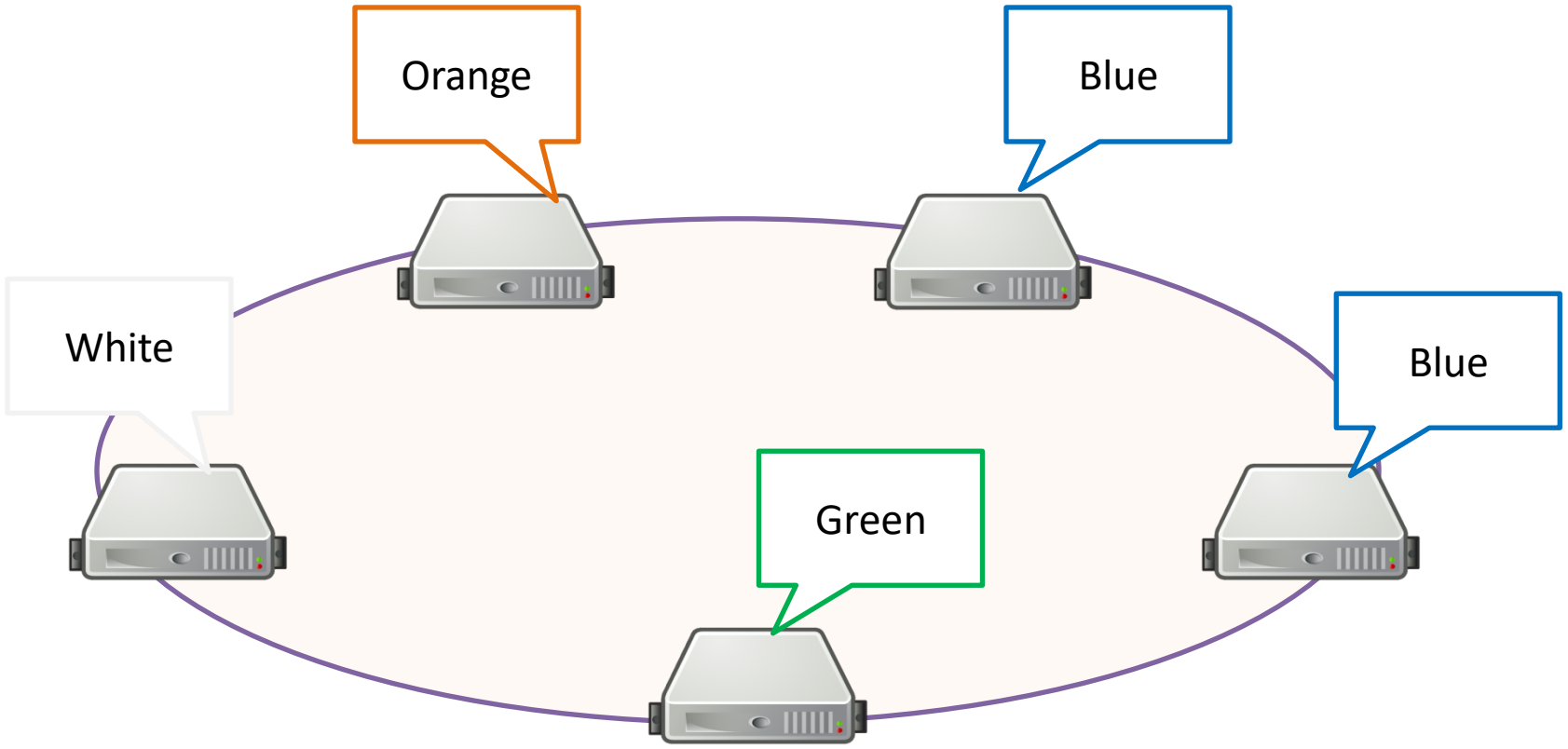


¿Valor de n necesario para un consenso único con N nodos?

$n > N/2$

Consenso distribuido

Consenso deshabilitado: Toma el primer resultado



Consenso distribuido

¿CP vs. AP? 

Consenso fuerte: Todos los nodos deben concordar **CP**

Consenso mayoritario: Deben concordar una mayoría

Consenso plural: Deben concordar una pluralidad

Consenso de quorum: Deben concordar ***n*** nodos

Consenso deshabilitado: Toma el primer resultado

AP

Consenso distribuido

¿La escalabilidad? 

Consenso fuerte: Todos los nodos deben concordar

Más mensajes

Consenso mayoritario: Deben concordar una mayoría

Consenso plural: Deben concordar una pluralidad

Consenso de quorum: Deben concordar ***n*** nodos

Consenso deshabilitado: Toma el primer resultado

Menos mensajes

Consenso distribuido

Consenso fuerte: Todos los nodos deben concordar

Consenso mayoritario: Deben concordar una mayoría

Consenso plural: Deben concordar una pluralidad

Consenso de quorum: Deben concordar *n* nodos

Consenso deshabilitado: Toma el primer resultado

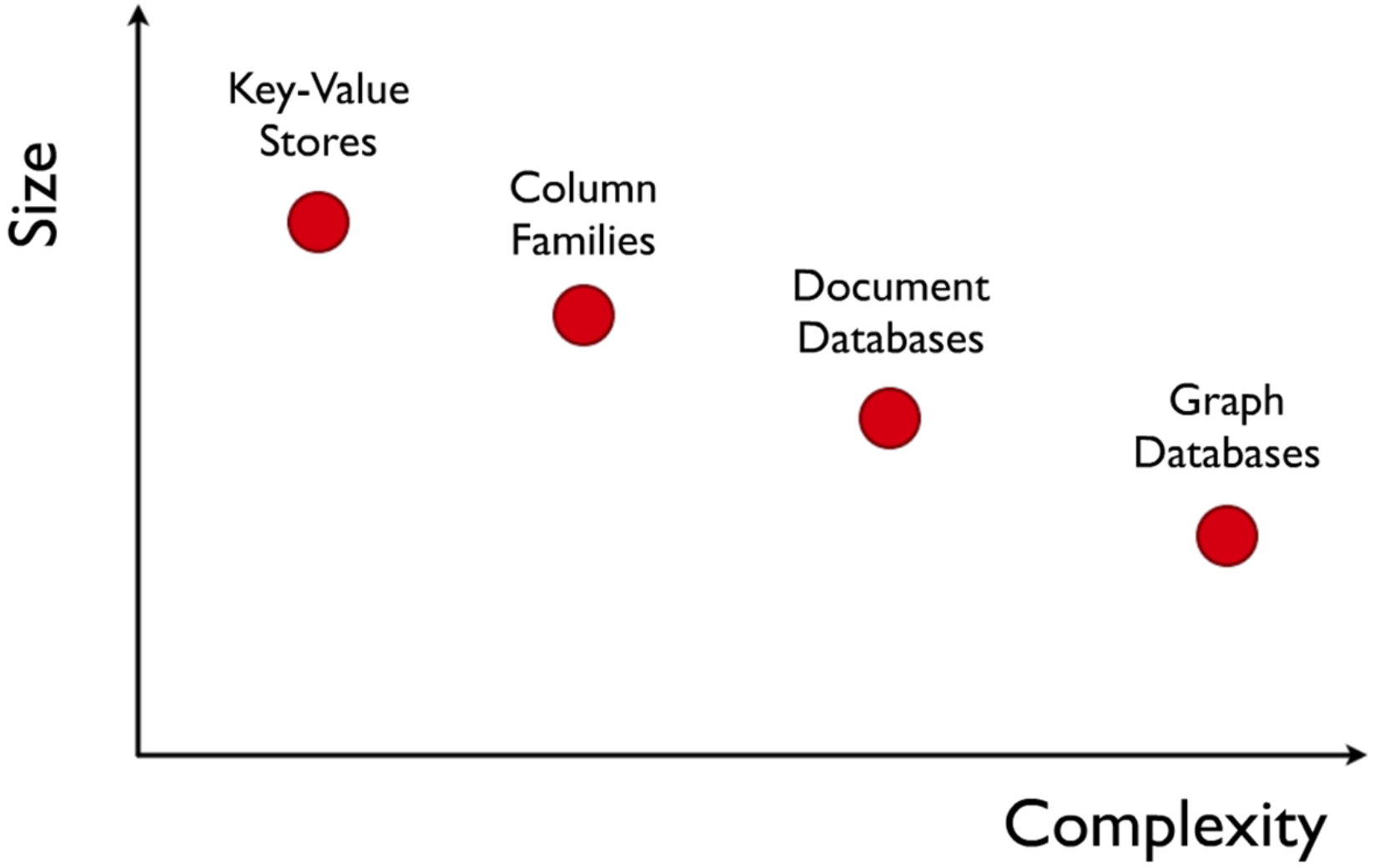
La mejor elección depende de la aplicación:

Muchos sistemas NoSQL permiten elegir
el nivel de consenso/replicación

NoSQL:

Sistemas de Llave–Valor ("Key–Value")

NoSQL


Size
Key-Value Stores
Column Families
Document Databases
Graph Databases
Complexity

Modelo de Llave–Valor

Es sólo un mapa ☺

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

Key	Value
Afghanistan	Kabul
Albania	Tirana
Algeria	Algiers
Andorra la Vella	Andorra la Vella
Angola	Luanda
Antigua and Barbuda	St. John’s
…	…

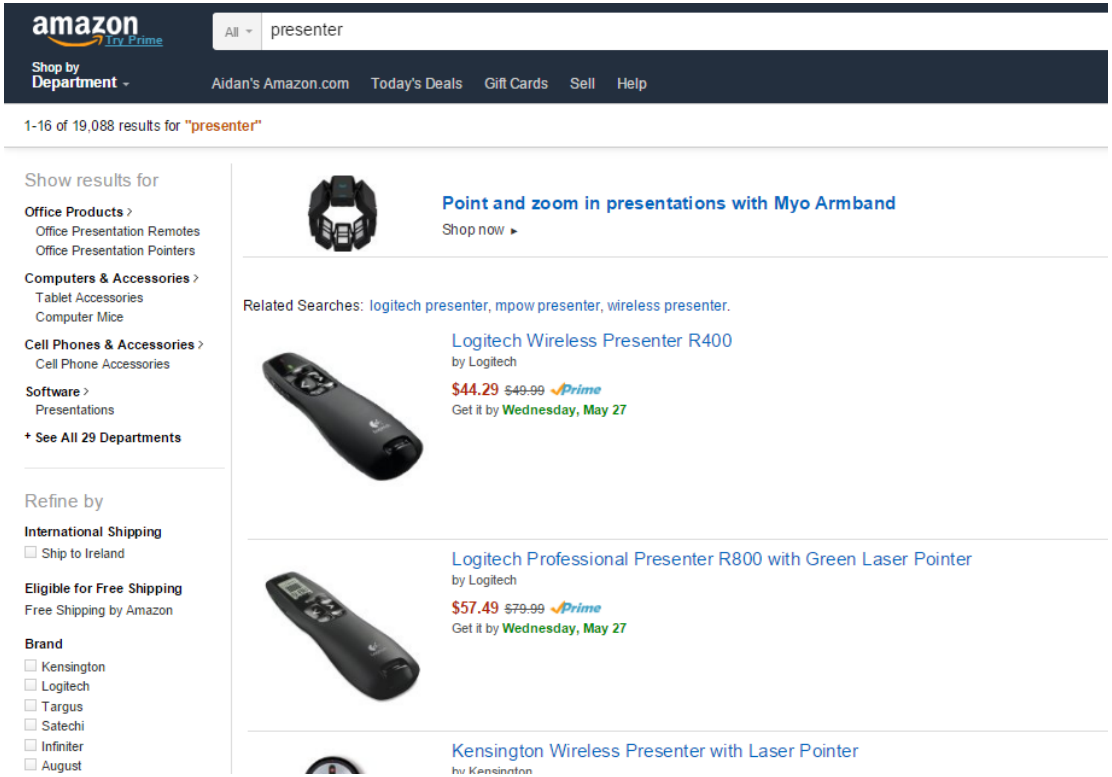
Pero se puede hacer mucho con un mapa

Key	Value
country:Afghanistan	capital@city:Kabul,continent:Asia,pop:31108077#2011
country:Albania	capital@city:Tirana,continent:Europe,pop:3011405#2013
…	…
city:Kabul	country:Afghanistan,pop:3476000#2013
city:Tirana	country:Albania,pop:3011405#2013
…	…
user:10239	basedIn@city:Tirana,post:{103,10430,201}
…	…

El caso de Amazon

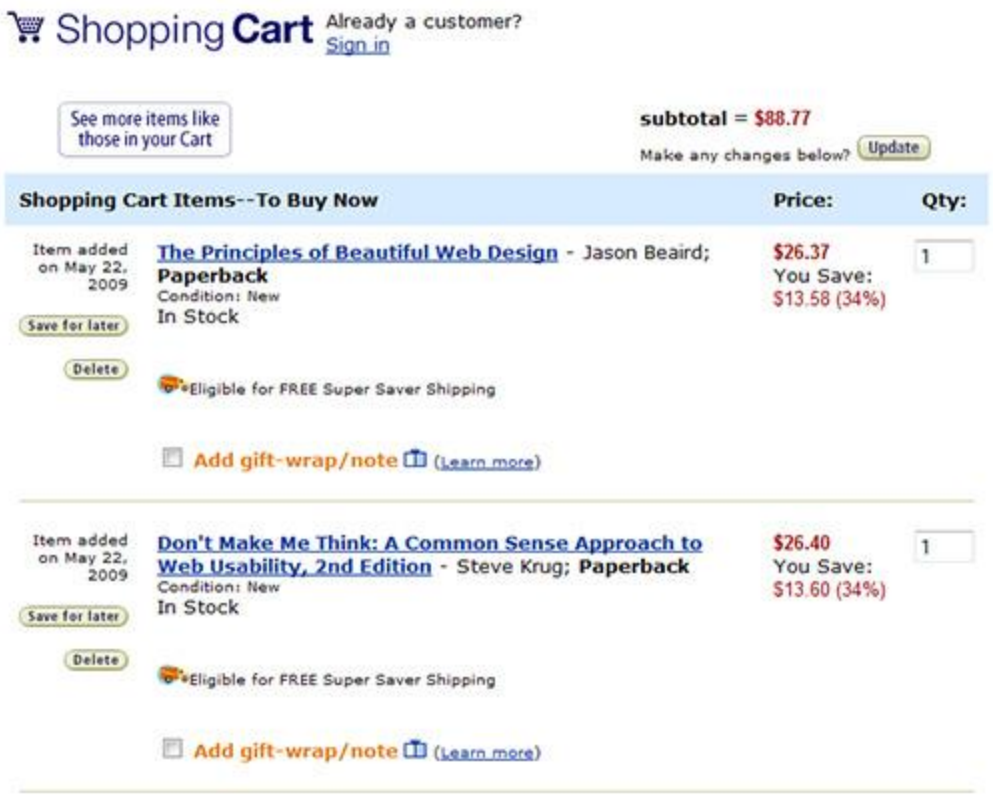
El escenario en Amazon

Listado de productos: precios, detalle, stock, etc.



El escenario en Amazon

Información de clientes: carro de compras, cuenta, etc.

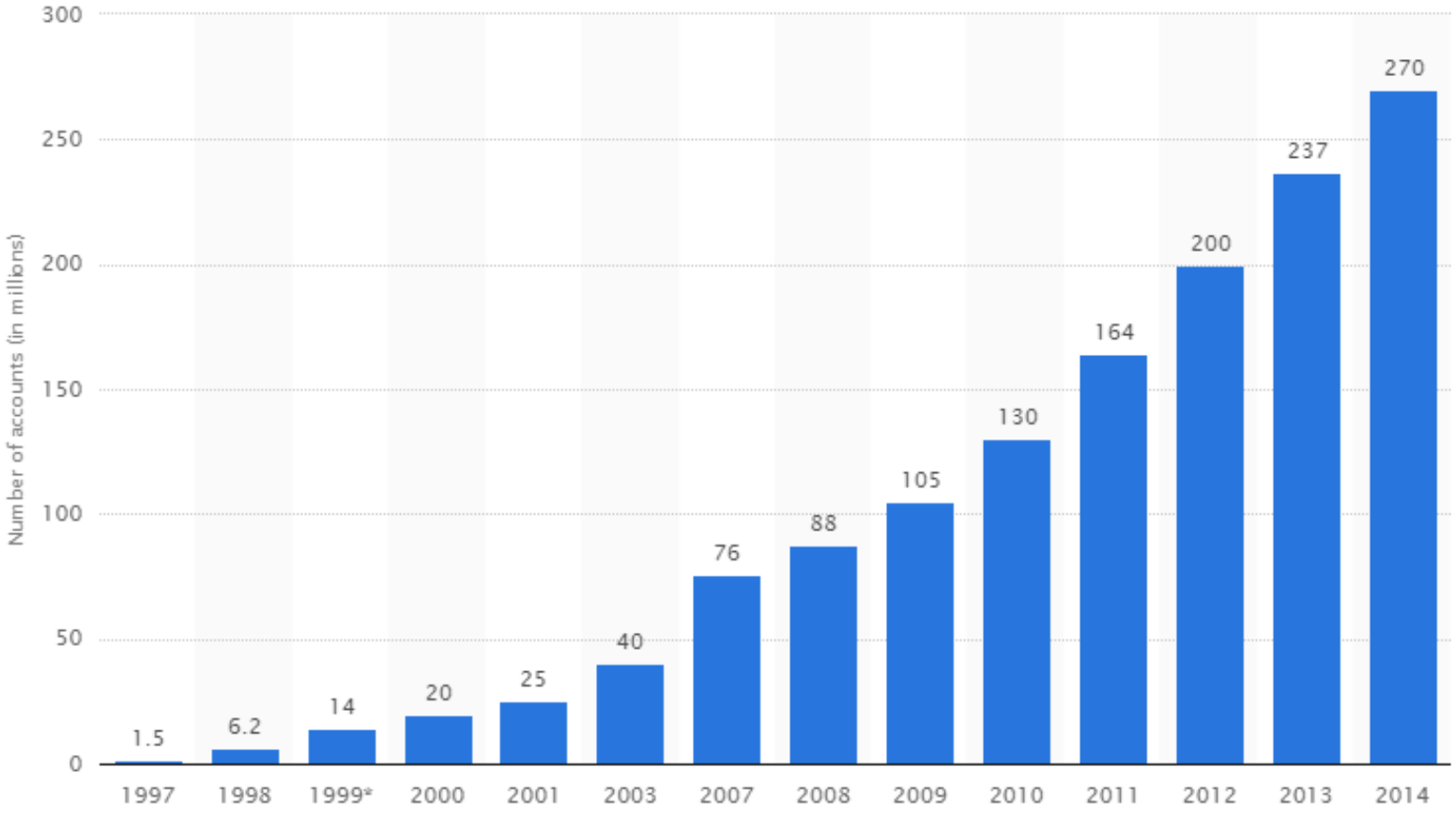


El escenario en Amazon

Recomendaciones, etc.:

El escenario en Amazon

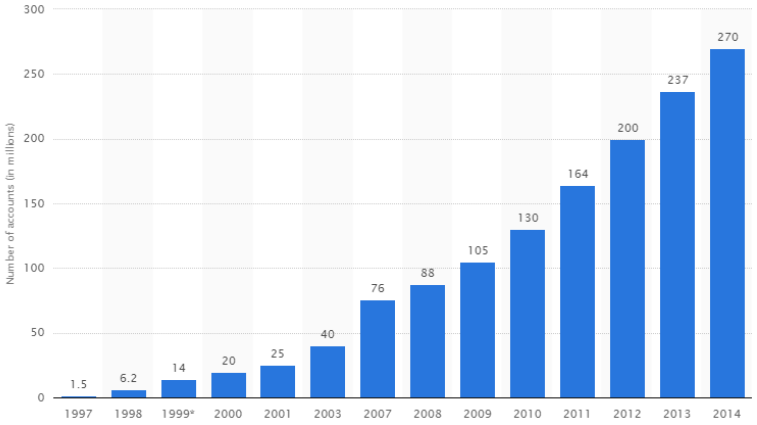
- Clientes de Amazon:



El escenario en Amazon

El escenario en Amazon

Bases de datos sobreexigidas …



Pero muchos servicios de Amazon no necesitan:

- SQL (a menudo un mapa simple basta)

o incluso:

- transacciones, coherencia fuerte, etc.

Guardado Llave-Valor: Amazon Dynamo(DB)

Dynamo: Amazon's Highly Available Key-value Store

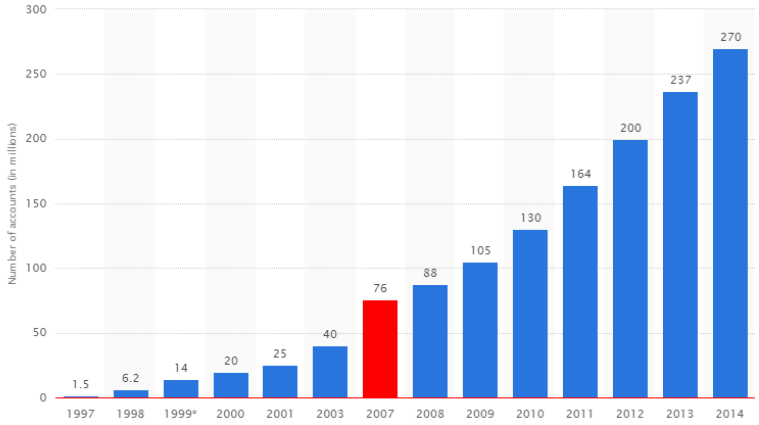
Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are



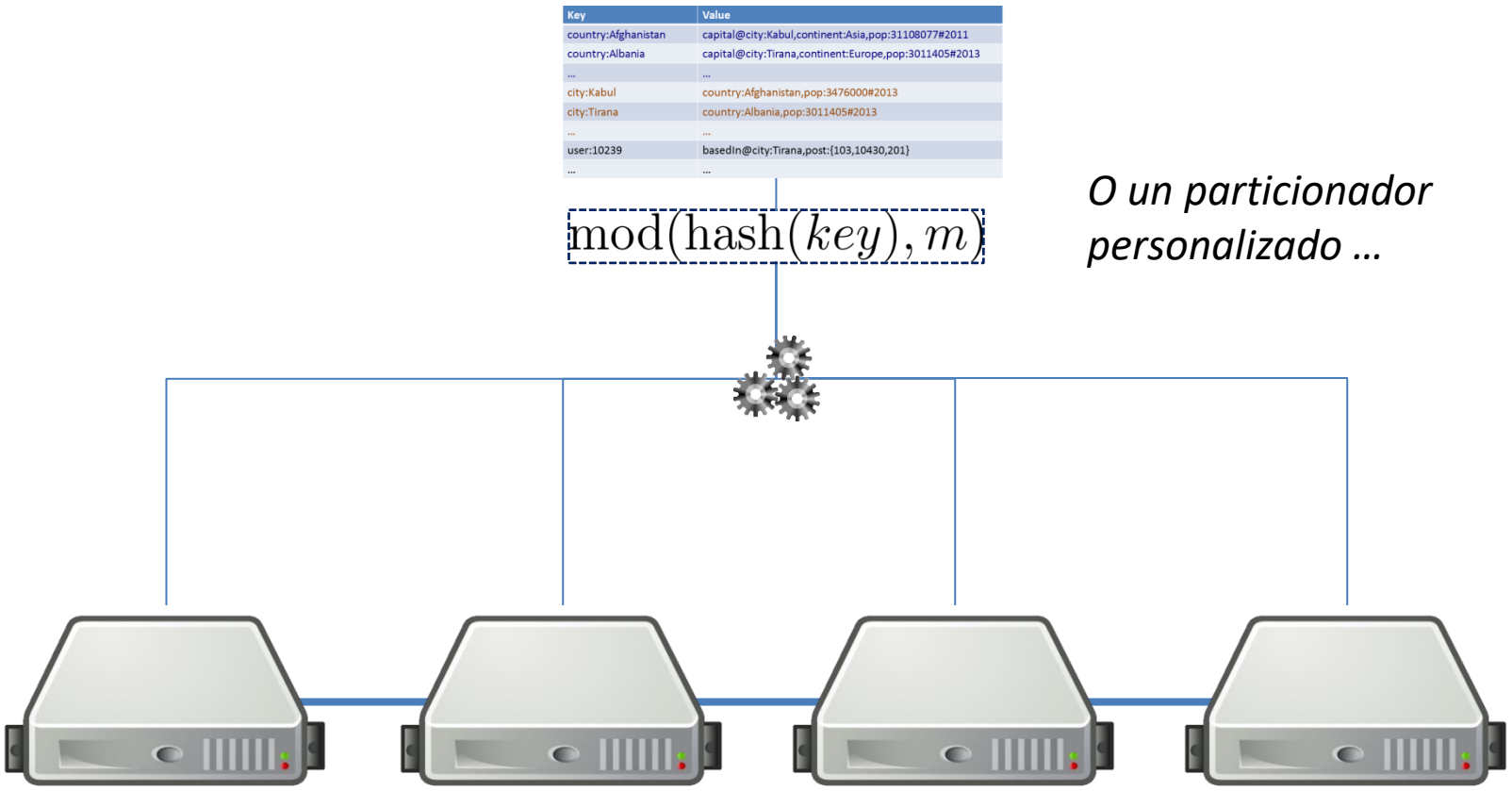
Objetivos:

- Escalabilidad
- Alta disponibilidad
- Rendimiento

No se necesita SQL ni ACID completos

Guardado Llave-Valor: Distribución

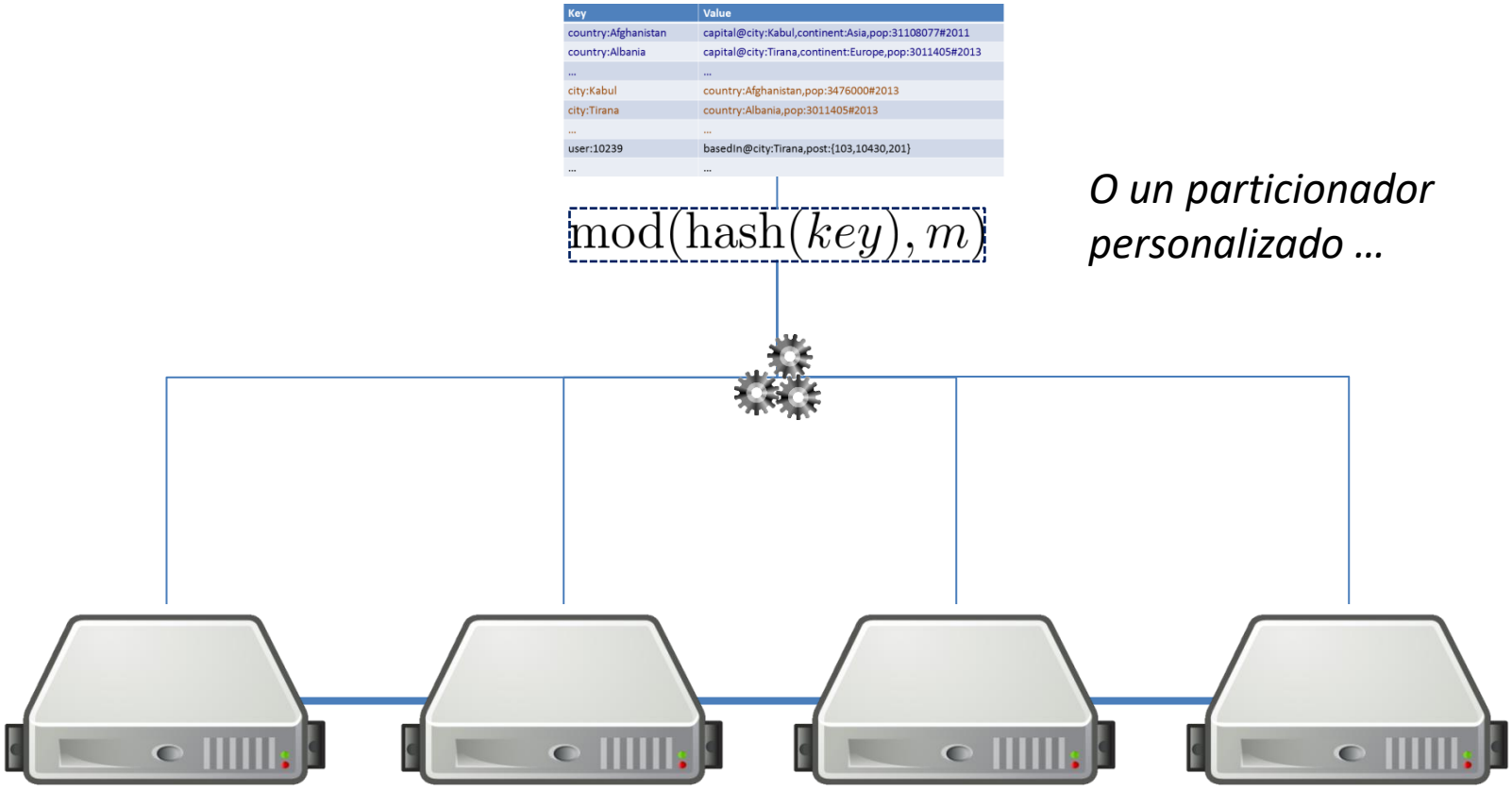
¿Cómo se podría distribuir el guardado llave-valor?



Guardado Llave-Valor: Distribución

¿Pero qué pasa si una máquina llega o se va de repente?

¿Cómo podemos evitar volver a hashear todo de nuevo?

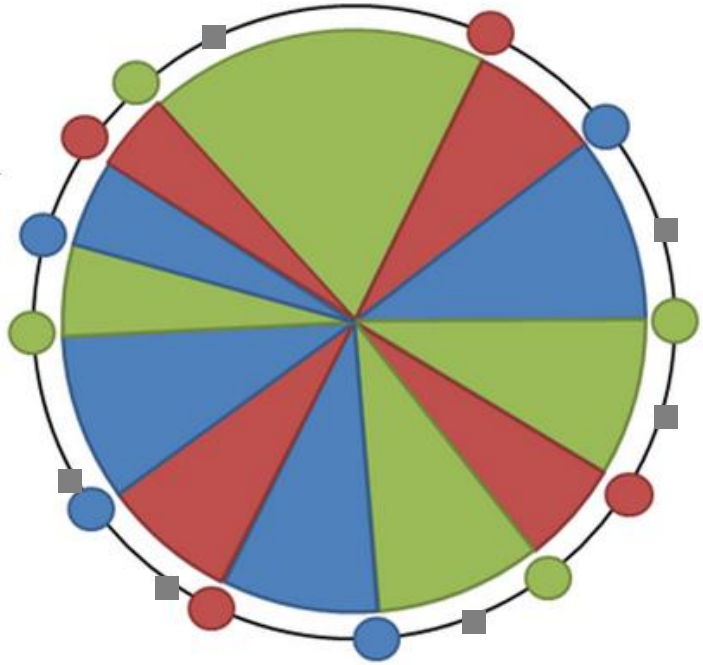


Hashing coherente ("Consistent Hashing")

Evitar volver a hashear todo de nuevo

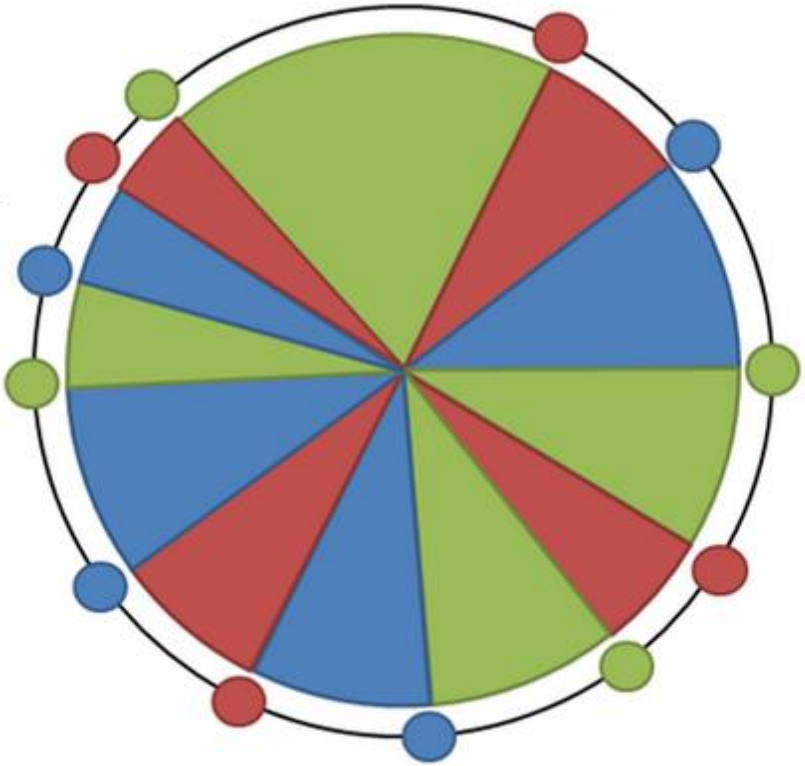
- Hashear usando un anillo
- Cada máquina escoge n puntos pseudoaleatorios en el anillo
- La máquina es responsable del arco después de su punto
- Si una máquina se va, su rango se mueve a la máquina anterior
- Si una máquina se une, elige nuevos puntos
- Objetos mapeados al anillo

¿Cuántas llaves en promedio se necesitan mover si una máquina se une o se va?



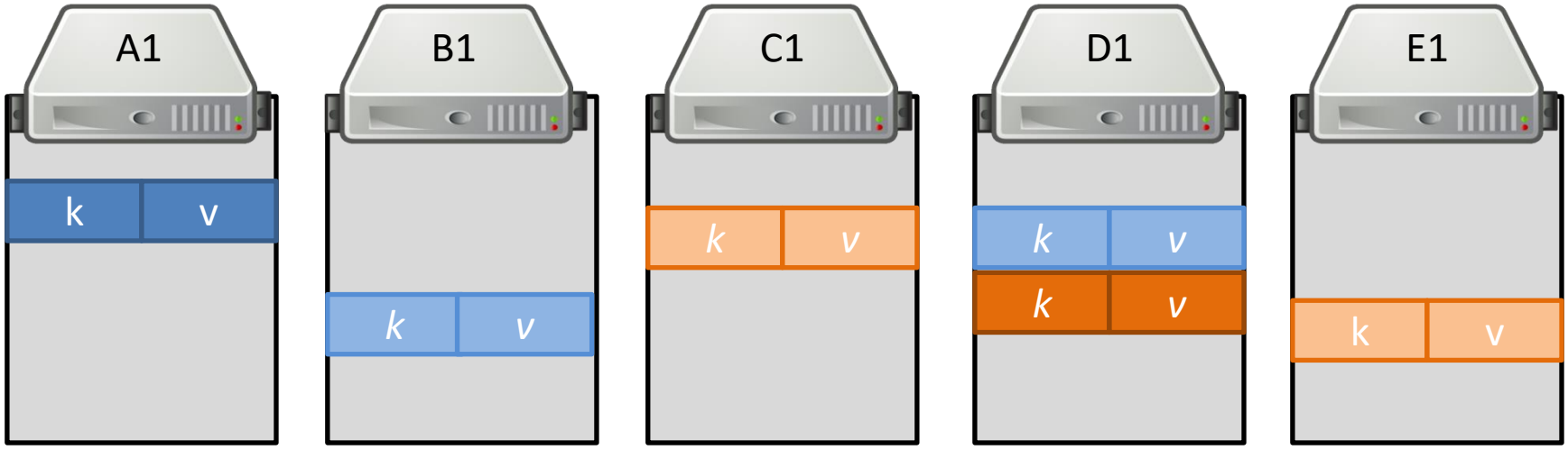
Amazon Dynamo: Hashing

- Hashing coherente (128-bit MD5)



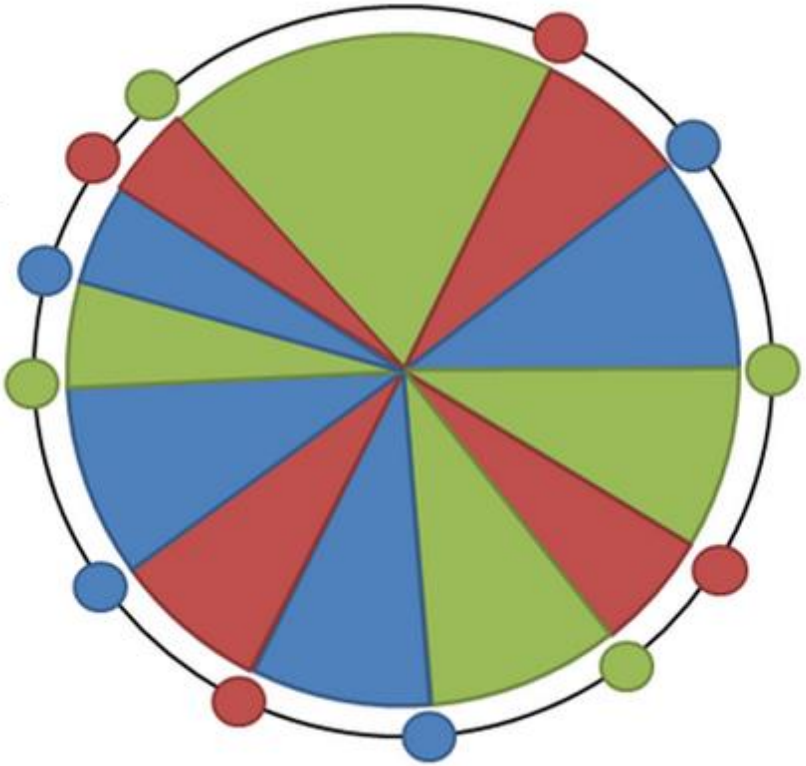
Guardado llave-valor: Replicación

- Un factor de replicación fijo (aquí 3)
- Comúnmente hay réplicas primarias / secundarias
  - Una nueva réplica primaria es elegida desde las réplicas secundarias en caso de que falle la primaria



Amazon Dynamo: Replicación

- ¿Factor de replicación de n?
  - ¡Fácil! Elegir n arcos siguientes (¡con máquinas diferentes!)



Amazon Dynamo: Modelo

- Tabla nombrada con llave primaria y valor
- La llave primaria es hasheada / sin orden

Countries	
Primary Key	**Value**
Afghanistan	capital:Kabul,continent:Asia,pop:31108077#2011
Albania	capital:Tirana,continent:Europe,pop:3011405#2013
…	…

Cities	
Primary Key	**Value**
Kabul	country:Afghanistan,pop:3476000#2013
Tirana	country:Albania,pop:3011405#2013
…	…

Amazon Dynamo: CAP

Dos opciones para cada tabla:

- AP: Coherencia eventual, Alta disponibilidad
- CP: Coherencia fuerte, Menor disponibilidad

Para los interesados …

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati,
Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall
and Werner Vogels

Amazon.com

ABSTRACT

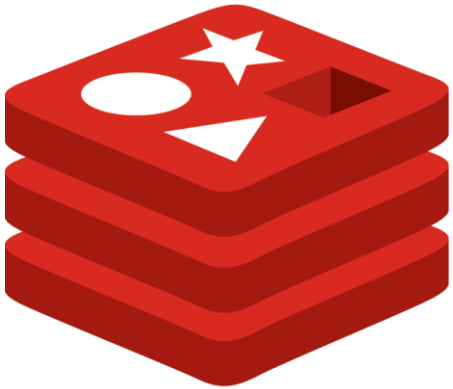
Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are

Otros guardados llave-valor

Otros sistemas de guardado llave-valor

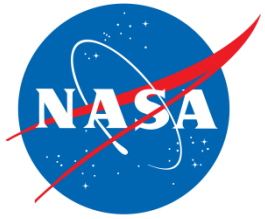
Otros sistemas de guardado llave-valor



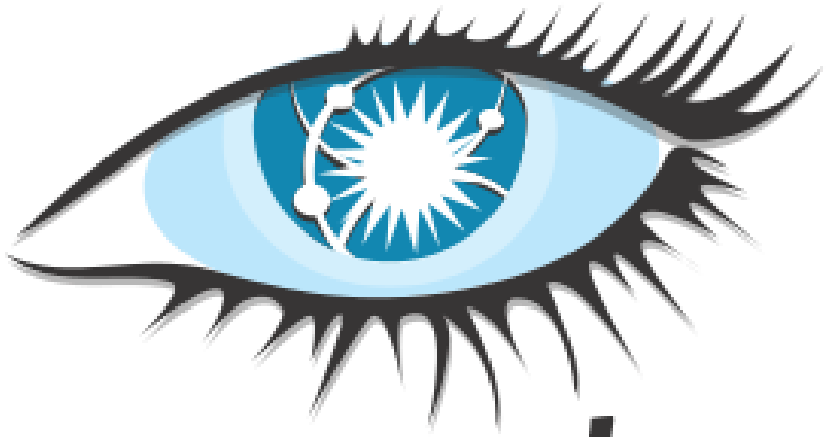
redis

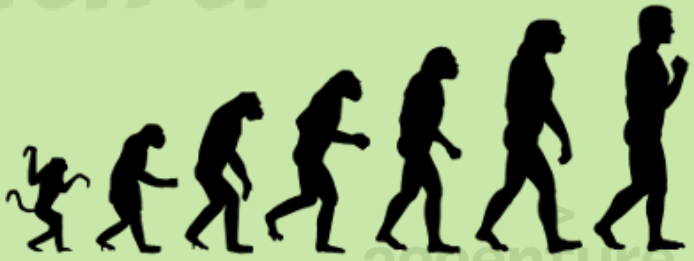
StackExchange

Otros sistemas de guardado llave-valor



Otros sistemas de guardado llave-valor





Ahora uso un modelo tabular

373 systems in ranking, August 2021

Rank			DBMS	Database Model	Score		
Aug 2021	Jul 2021	Aug 2020			Aug 2021	Jul 2021	Aug 2020
1.	1.	1.	Oracle	Relational, Multi-model	1269.26	+6.59	-85.90
2.	2.	2.	MySQL	Relational, Multi-model	1238.22	+9.84	-23.36
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	973.35	-8.61	-102.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	577.05	-0.10	+40.28
5.	5.	5.	MongoDB	Document, Multi-model	496.54	+0.38	+52.98
6.	6.	↑ 7.	Redis	Key-value, Multi-model	169.88	+1.58	+17.01
7.	7.	↓ 6.	IBM Db2	Relational, Multi-model	165.46	+0.31	+3.01
8.	8.	8.	Elasticsearch	Search engine, Multi-model	157.08	+1.32	+4.76
9.	9.	9.	SQLite	Relational	129.81	-0.39	+3.00
10.	↑ 11.	10.	Microsoft Access	Relational	114.84	+1.39	-5.02
11.	↓ 10.	11.	Cassandra	Wide column	113.66	-0.35	-6.18
12.	12.	12.	MariaDB	Relational, Multi-model	98.98	+0.99	+8.06
13.	13.	13.	Splunk	Search engine	90.60	+0.55	+0.69
14.	14.	↑ 15.	Hive	Relational	83.93	+1.26	+8.64
15.	15.	↑ 17.	Microsoft Azure SQL Database	Relational, Multi-model	75.15	-0.06	+18.31
16.	16.	16.	Amazon DynamoDB	Multi-model	74.90	-0.30	+10.15
17.	17.	↓ 14.	Teradata	Relational, Multi-model	68.82	-0.13	-7.96
18.	18.	↑ 21.	Neo4j	Graph	56.95	-0.21	+6.77
19.	19.	19.	SAP HANA	Relational, Multi-model	55.57	+1.76	+2.46
20.	20.	20.	Solr	Search engine, Multi-model	51.06	-0.73	-0.63

http://db-engines.com/en/ranking

Modelo Tabular / Familia de Columnas

Llave–Valor = un mapa distribuido

Countries	
Primary Key	**Value**
Afghanistan	capital:Kabul,continent:Asia,pop:31108077#2011
Albania	capital:Tirana,continent:Europe,pop:3011405#2013
…	…

Tabular = un mapa multi-dimensional

Countries				
Primary Key	**capital**	**continent**	**pop-value**	**pop-year**
Afghanistan	Kabul	Asia	31108077	2011
Albania	Tirana	Europe	3011405	2013
…	…	…	…	…

Bigtable: El "Whitepaper" Original

Autores de MapReduce

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

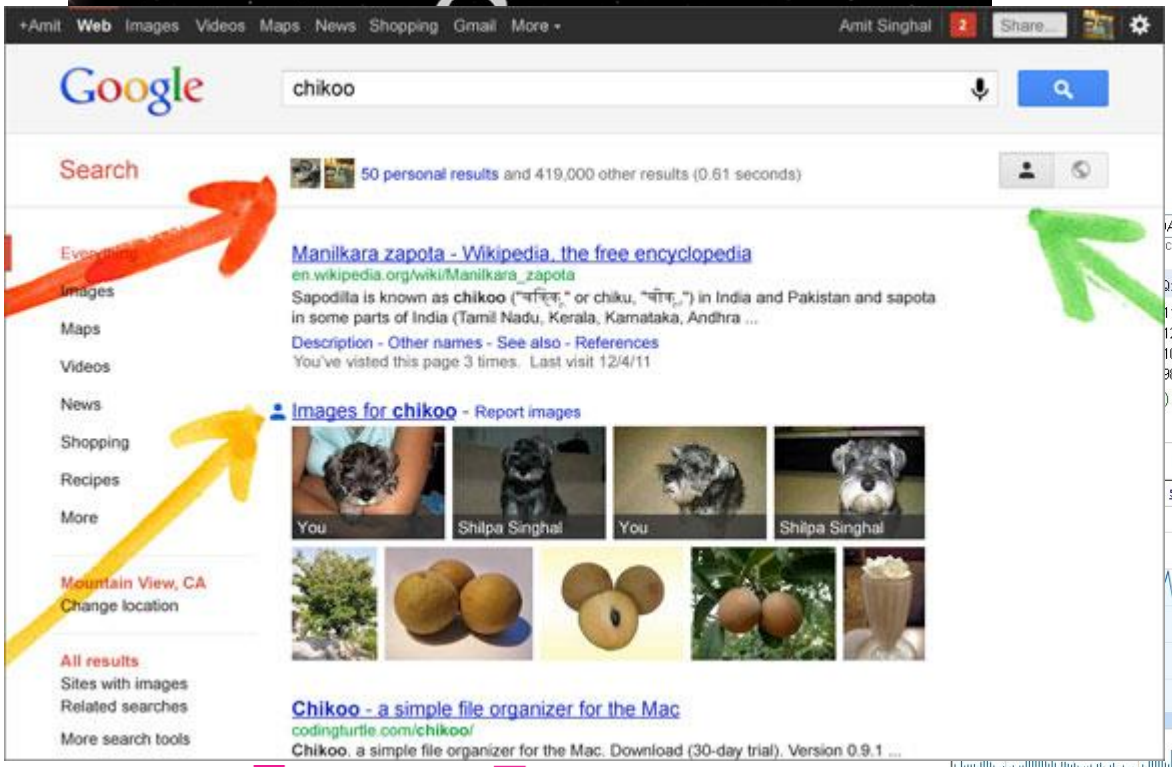
Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Sec-

Bigtable es/era usado para …



orkut by Google™

. . .

Bigtable: Modelo de Datos

"Un mapa disperso ("sparse"), distribuido, persistente, multi-dimensional y ordenado"

- Disperso: No todos los valores forman un cuadrado denso
- Distribuido: Muchas máquinas
- Persistente: Almacenamiento en disco (GFS)
- Multi-dimensional: Valores con columnas
- Ordenado: Lexicográficamente por llave de la fila
- Mapa: Buscas una llave, obtienes un valor

Bigtable: en resumen

(fila, columna, tiempo) → valor

- fila: el string identificador de una fila
  - ej., "`Afganistan`"
- columna: el nombre de una columna
  - ej., "`pop-value`"
- tiempo: un time-stamp (long / 64-bit)
  - ej., `18545664`
- valor: el elemento de una celda
  - ej., "`31120978`"

Bigtable: en resumen

(fila, columna, tiempo) → valor

(**Afganistan**,**pop-value**,**t_4**) → **31108077**

Primary Key	capital		continent		pop-value		pop-year	
Afghanistan	t_1	Kabul	t_1	Asia	t_1	31143292	t_1	2009
					t_2	31120978		
					t_4	31108077	t_4	2011
Albania	t_1	Tirana	t_1	Europe	t_1	2912380	t_1	2010
					t_3	3011405	t_3	2013
…	…		…			…		…

Bigtable: llaves ordenadas

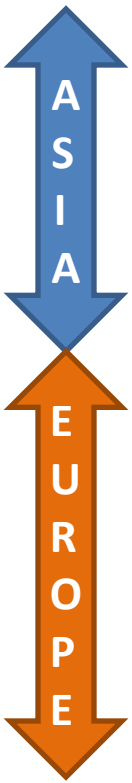
SORTED

Primary Key	capital		pop-value		pop-year	
Asia:Afghanistan	t_1	Kabul	t_1	31143292	t_1	2009
			t_2	31120978		
			t_4	31108077	t_4	2011
Asia:Azerbaijan	…	…	…	…	…	…
…	…	…	…	…	…	…
Europe:Albania	t_1	Tirana	t_1	2912380	t_1	2010
			t_3	3011405	t_3	2013
Europe:Andorra	…	…	…	…	…	…
…	…	…	…	…	…	…

¿Las ventajas de llaves ordenadas?

Consultas por rango y …

Bigtable: Tablets



Primary Key	capital		pop-value		pop-year	
Asia:Afghanistan	t_1	Kabul	t_1	31143292	t_1	2009
			t_2	31120978		
			t_4	31108077	t_4	2011
Asia:Azerbaijan	…	…	…	…	…	…
…	…	…	…	…	…	…
Europe:Albania	t_1	Tirana	t_1	2912380	t_1	2010
			t_3	3011405	t_3	2013
Europe:Andorra	…	…	…	…	…	…
…	…	…	…	…	…	…

¿Las ventajas de llaves ordenadas?

Consultas por rango y …

… localidad del procesamiento

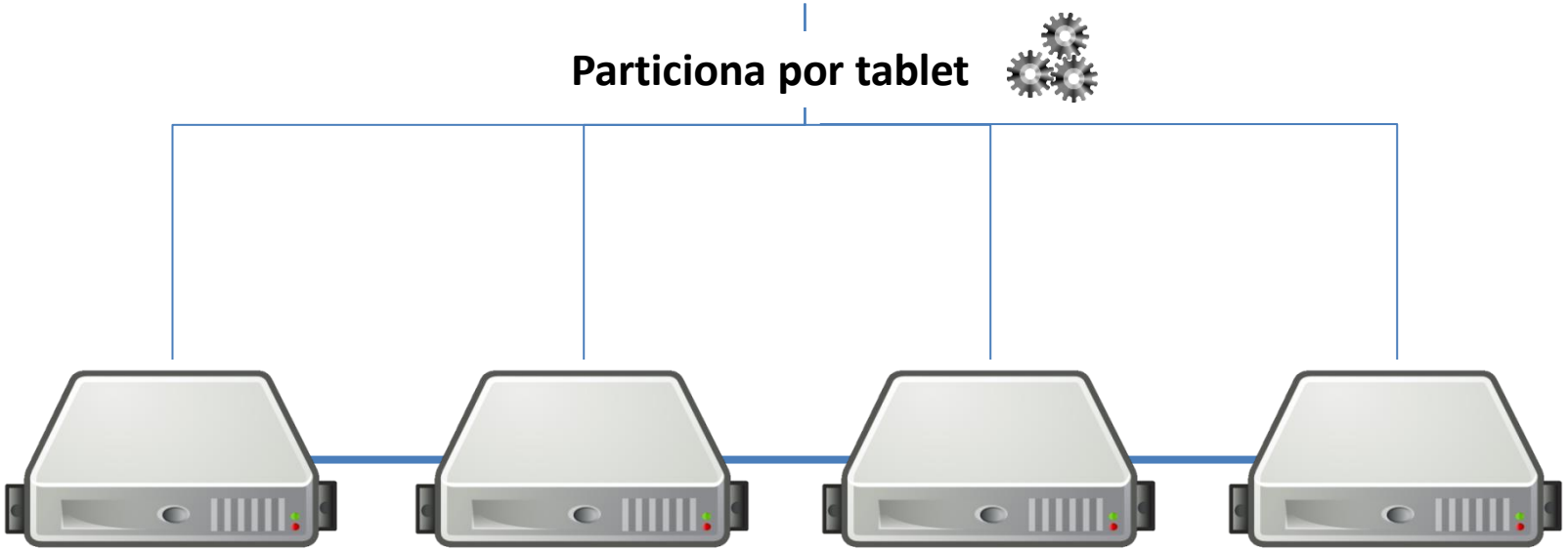
Un ejemplo real de localidad

Primary Key	language		title		links	
com.imdb	t_1	en	t_1	IMDb Home	t_1	…
			t_2	IMDB - Movies		
			t_4	IMDb	t_4	…
com.imdb/title/tt2724064/	t_1	en	t_2	Sharknado	t_2	…
com.imdb/title/tt3062074/	t_1	en	t_2	Sharknado II	t_2	
…	…	…	…	…	…	…
org.wikipedia	t_1	multi	t_1	Wikipedia	t_1	…
			t_3	Wikipedia Home	t_3	…
org.wikipedia.ace	t_1	ace	t_1	Wikipèdia bahsa Acèh	…	…
…	…	…	…	…	…	…

Bigtable: Distribución

Asia:Afghanistan	t_1	Kabul	t_1	31143292	t_1	2009
			t_2	31120978		
			t_4	31108077	t_4	2011
Asia:Azerbaijan	...	...	...	...	...	...
...	...	...	...	...	...	...

Europe:Albania	t_1	Tirana	t_1	2912380	t_1	2010
			t_3	3011405	t_3	2013
Europe:Andorra	...	...	...	...	...	...
...	...	...	...	...	...	...



Partición horizontal por rango

Bigtable: Familias de Columnas

Primary Key	pol:capital		demo:pop-value		demo:pop-year	
Asia:Afghanistan	t_1	Kabul	t_1	31143292	t_1	2009
			t_2	31120978		
			t_4	31108077	t_4	2011
Asia:Azerbaijan	…	…	…	…	…	…
…	…	…	…	…	…	…
Europe:Albania	t_1	Tirana	t_1	2912380	t_1	2010
			t_3	3011405	t_3	2013
Europe:Andorra	…	…	…	…	…	…
…	…	…	…	…	…	…

- Agrupar columnas relacionadas
  - Se acceden juntas de manera eficiente
  - Control de acceso a nivel de familia de columnas

Para los interesados …

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and

Tabular: Apache HBase

Tabular: Cassandra

CASSANDRA

Almacena Tabular

Ventas

producto	tienda	cliente	fecha	valor
1L_Leche	Santiago	412	2020-03-31T08:47:57Z	900
La_Tercera	Providencia	413	2020-03-31T08:47:59Z	2000
Nescafe	Providencia	413	2020-03-31T08:47:59Z	3500
Nescafe	Providencia	413	2020-03-31T08:48:00Z	3500
Comfort	Valparaíso	414	2020-03-31T08:48:04Z	2300
Nescafe	Providencia	415	2020-03-31T08:48:04Z	3500
Comfort	Valparaíso	416	2020-03-31T08:48:07Z	2300
1L_Leche	Valparaíso	416	2020-03-31T08:48:07Z	800
Nescafe	Santiago	412	2020-03-31T08:48:08Z	3700
La_Tercera	Santiago	412	2020-03-31T08:48:08Z	2000
Comfort	Santiago	412	2020-03-31T08:48:08Z	2500
…	…	…	…	…

Cassandra Query Language (CQL)

```
SELECT [ JSON | DISTINCT ] ( select_clause | '*' )
 FROM table_name
 [ WHERE where_clause ]
 [ GROUP BY group_by_clause ]
 [ ORDER BY ordering_clause ]
 [ PER PARTITION LIMIT (integer | bind_marker) ]
 [ LIMIT (integer | bind_marker) ]
 [ ALLOW FILTERING ]
```

- `JSON`: Devolver datos en el formato de JSON
- `PER PARTITION LIMIT`: Limitar el número de filas por partición
- `ALLOW FILTERING`: Permitir consultas que puedan filtrar filas leídas

https://cassandra.apache.org/doc/latest/cql/index.html

Cassandra Query Language (CQL)

Ventas

producto	tienda	cliente	fecha	valor
1L_Leche	Santiago	412	2020-03-31T08:47:57Z	900
La_Tercera	Providencia	413	2020-03-31T08:47:59Z	2000
Nescafe	Providencia	413	2020-03-31T08:47:59Z	3500
Nescafe	Providencia	413	2020-03-31T08:48:00Z	3500
Comfort	Valparaíso	414	2020-03-31T08:48:04Z	2300
Nescafe	Providencia	415	2020-03-31T08:48:04Z	3500
Comfort	Valparaíso	416	2020-03-31T08:48:07Z	2300
1L_Leche	Valparaíso	416	2020-03-31T08:48:07Z	800
Nescafe	Santiago	412	2020-03-31T08:48:08Z	3700
La_Tercera	Santiago	412	2020-03-31T08:48:08Z	2000
Comfort	Santiago	412	2020-03-31T08:48:08Z	2500
…	…	…	…	…

```
SELECT * FROM Ventas WHERE producto = ‘Comfort’ AND valor < 2500;
```

Llave primaria

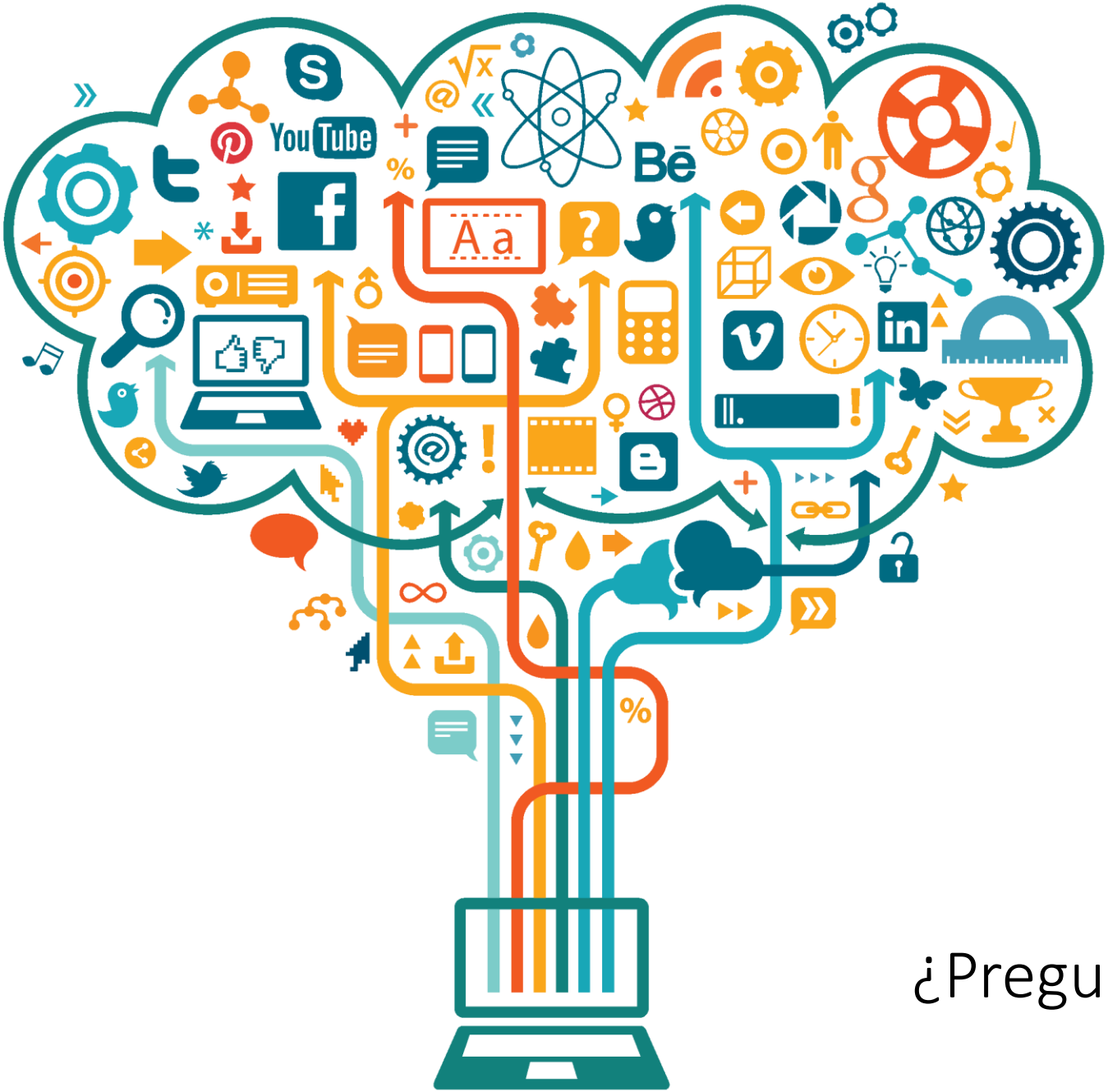
Ventas(**producto**,**tienda**,fecha,cliente,valor)

producto	tienda	fecha	cliente	valor
1L_Leche	Santiago	2020-03-31T08:47:57Z	412	900
Comfort	Valparaíso	2020-03-31T08:48:04Z	414	2300
Comfort	Valparaíso	2020-03-31T08:48:07Z	416	2300

producto	tienda	fecha	cliente	valor
La_Tercera	Providencia	2020-03-31T08:47:59Z	413	2000
Comfort	Santiago	2020-03-31T08:48:08Z	412	2500

producto	tienda	fecha	cliente	valor
Nescafe	Providencia	2020-03-31T08:47:59Z	413	3500
Nescafe	Providencia	2020-03-31T08:48:00Z	413	3500
Nescafe	Providencia	2020-03-31T08:48:04Z	415	3500

...



¿Preguntas?