

# CC5212-1

PROCESAMIENTO MASIVO DE DATOS

OTOÑO 2019

## Lecture 9

### NoSQL: Overview

Aidan Hogan

[aidhog@gmail.com](mailto:aidhog@gmail.com)

# Hadoop/MapReduce/Pig/Spark: Processing Un/Structured Information



Information Retrieval:

Storing Unstructured Information

A word cloud of information retrieval terms. The words are arranged in a roughly circular pattern, with some larger than others. The colors of the words range from light blue to dark purple. The terms include: stop-words, information-overload, ranking, lemmatisation, compression, pagerank, heap's-law, keywords, tf-idf, zipf's-law, robots.txt, query, importance, relevance, site-map, DDoS, cosine, crawling, link-analysis, similarity, search, posting-lists, term-frequency, and elias-encoding.

stop-words information-overload  
ranking lemmatisation  
compression pagerank heap's-law  
keywords tf-idf  
zipf's-law robots.txt query  
importance relevance  
site-map DDoS cosine  
crawling link-analysis similarity  
search posting-lists  
term-frequency elias-encoding

Storing Structured Information??



BIG DATA:

STORING STRUCTURED INFORMATION

# Relational Databases





# Relational Databases: One Size Fits All?

## “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*In this paper, we argue that this concept is no longer applicable to the database market, and that the commercial world will fracture into a collection of independent database engines, some of which may be unified by a common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.

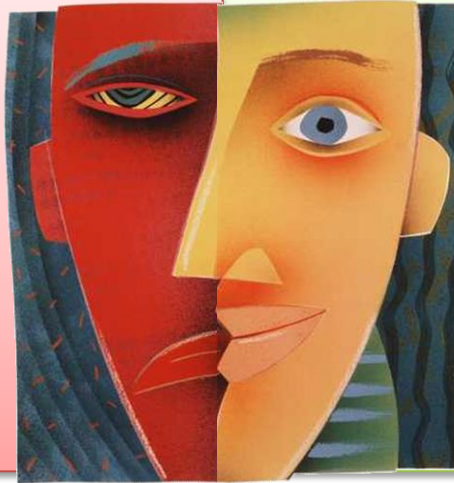
The rest of the paper is structured as follows. In Section 2, we briefly indicate why the single code-line strategy has failed already by citing some of the key characteristics of the data warehouse market. In Section



# SQL

Difficult to optimise

Difficult to distribute



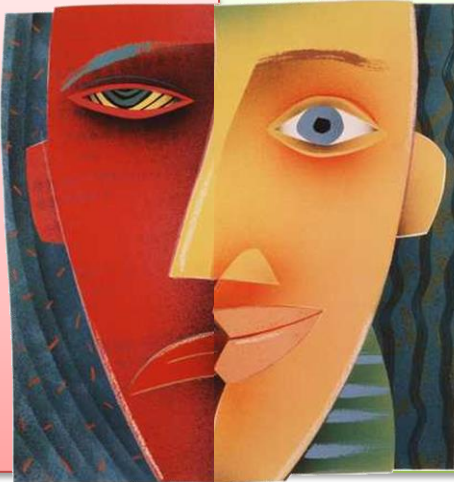
Declarative language

Expressive

# ACID

Costly to implement

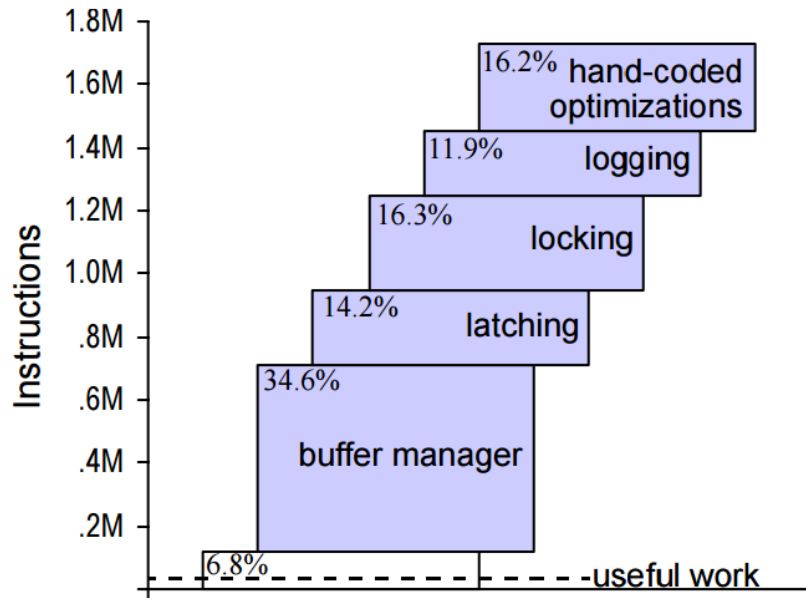
Difficult to distribute



Guarantees correct behaviour

Support transactions

# Transactional overhead: the cost of ACID



- 640 transactions per second for system with full transactional support (ACID)
- 12,700 transactions per second for system without logs, transactions or lock scheduling

## 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

# RDBMS: Complexity



# ALTERNATIVES TO RELATIONAL DATABASES FOR BIG DATA?

# NoSQL

Anybody know anything about NoSQL?



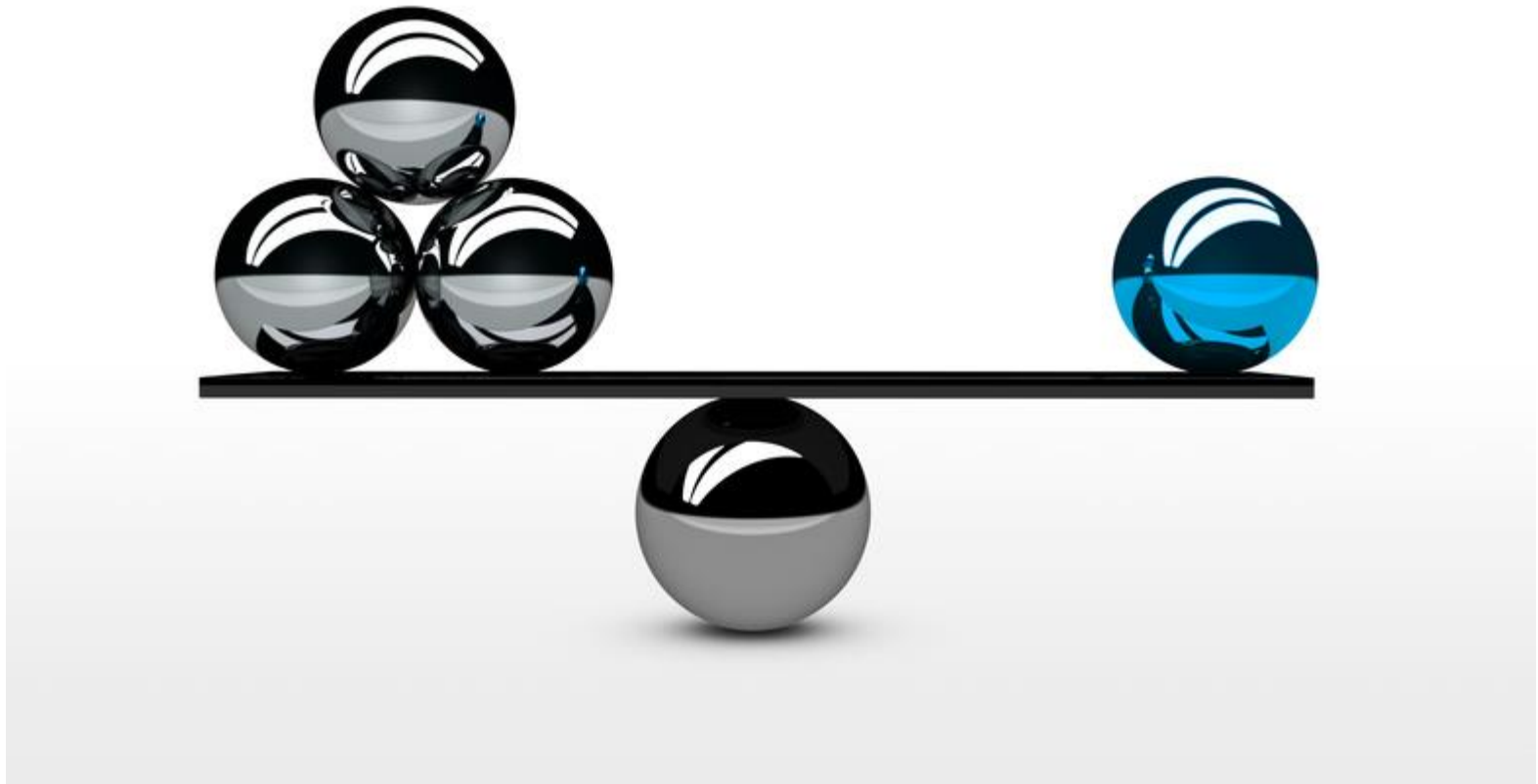
**N**ot  
**O**nly **SQL**



May 2019	Rank		DBMS	Database Model	Score		
	Apr 2019	May 2018			May 2019	Apr 2019	May 2018
1.	1.	1.	Oracle	Relational, Multi-model	1285.55	+5.61	-4.87
2.	2.	2.	MySQL	Relational, Multi-model	1218.96	+3.82	-4.38
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	1072.19	+12.23	-13.66
4.	4.	4.	PostgreSQL	Relational, Multi-model	478.89	+0.17	+77.99
5.	5.	5.	MongoDB	Document	408.07	+6.10	+65.96
6.	6.	6.	IBM Db2	Relational, Multi-model	174.44	-1.61	-11.17
7.	8.	9.	Elasticsearch	Search engine, Multi-model	148.62	+2.62	+18.18
8.	7.	7.	Redis	Key-value, Multi-model	148.40	+2.03	+13.06
9.	9.	8.	Microsoft Access	Relational	143.78	-0.87	+10.67
10.	11.	10.	Cassandra	Wide column	125.72	+2.11	+7.89
11.	10.	11.	SQLite	Relational	122.90	-1.32	+7.44
12.	12.	14.	MariaDB	Relational, Multi-model	86.52	+1.29	+21.53
13.	13.	13.	Splunk	Search engine	85.24	+2.15	+20.15
14.	15.	18.	Hive	Relational	77.90	+3.19	+20.93
15.	14.	12.	Teradata	Relational	76.04	+0.69	+1.63
16.	16.	15.	Solr	Search engine	60.80	+0.57	-0.72
17.	17.	17.	HBase	Wide column	59.77	+1.11	-0.18
18.	18.	19.	FileMaker	Relational	58.51	+0.09	+3.84
19.	19.	21.	Amazon DynamoDB	Multi-model	55.93	-0.08	+11.74
20.	21.	20.	SAP HANA	Relational, Multi-model	55.74	+0.39	+7.37
21.	20.	16.	SAP Adaptive Server	Relational	55.44	-0.36	-6.07
22.	22.	22.	Neo4j	Graph	51.03	+1.54	+10.45
23.	23.	24.	Couchbase	Document	34.67	-1.61	+2.26
24.	25.	23.	Memcached	Key-value	28.90	+0.17	-4.66
25.	24.	26.	Microsoft Azure SQL Database	Relational, Multi-model	28.77	-0.02	+3.56

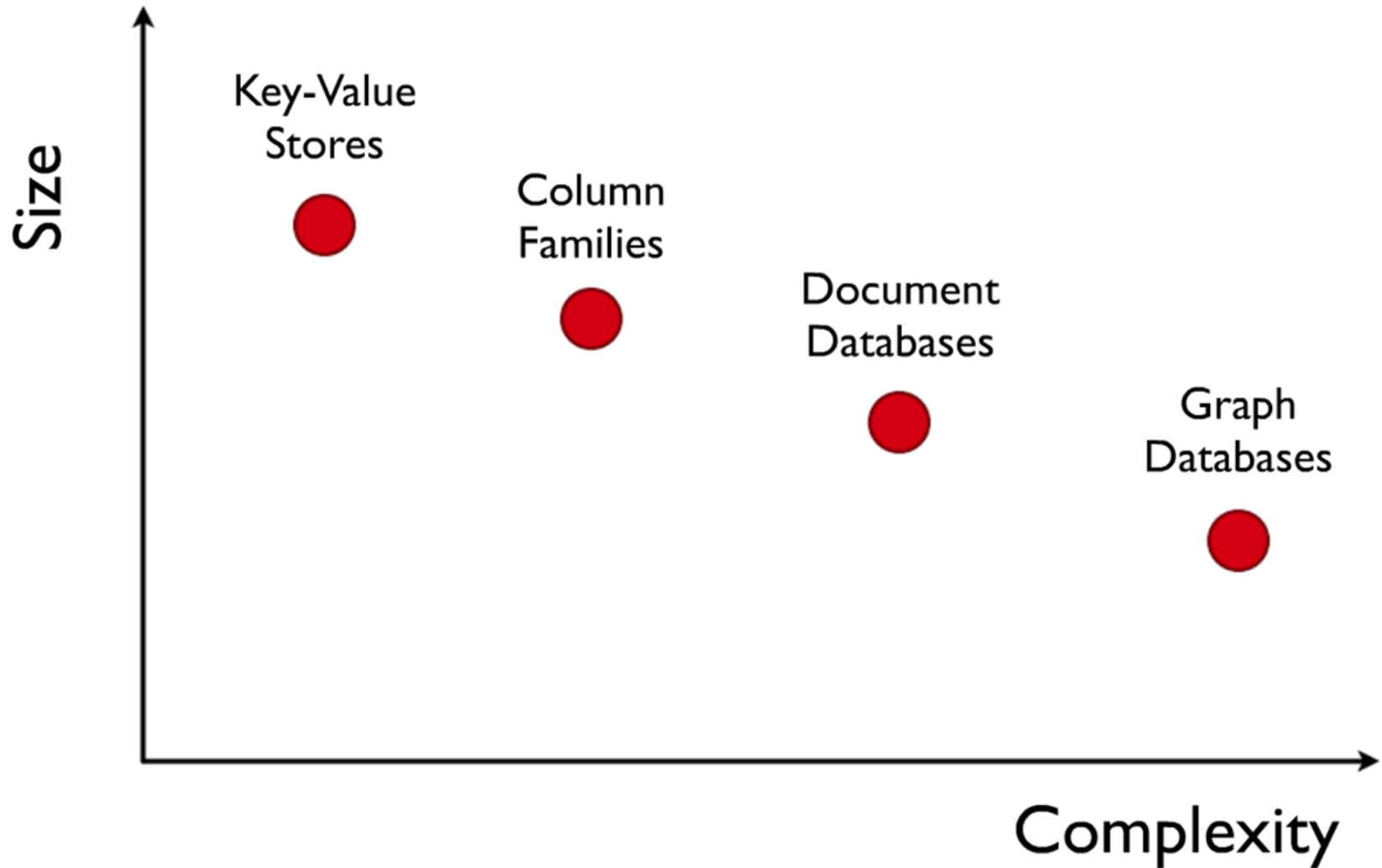


# NoSQL: features vs. scale/performance





# NoSQL: four main types



# NoSQL: common characteristics

- Often **distributed**
  - Sharding: splitting data over servers “horizontally”
  - Replication
  - Different guarantees: typically not ACID
- Often **simpler** languages than SQL
  - Simpler ad hoc APIs
  - More work for the application
- **Different flavours** (for different scenarios)
  - Different availability/consistency emphasis
  - Different scalability profiles
  - Different query functionality
  - Different data models

# LIMITATIONS OF DISTRIBUTED COMPUTING: CAP THEOREM

# But first ... ACID

For traditional (non-distributed) databases ...

## 1. Atomicity:

- Transactions all or nothing: fail cleanly

## 2. Consistency:

- Doesn't break constraints/rules

## 3. Isolation:

- Parallel transactions act as if sequential

## 4. Durability

- System remembers changes

# What is CAP?

Three *guarantees* a distributed sys. could make

## 1. Consistency:

- All nodes have a consistent view of the system

## 2. Availability:

- Every read/write is acted upon

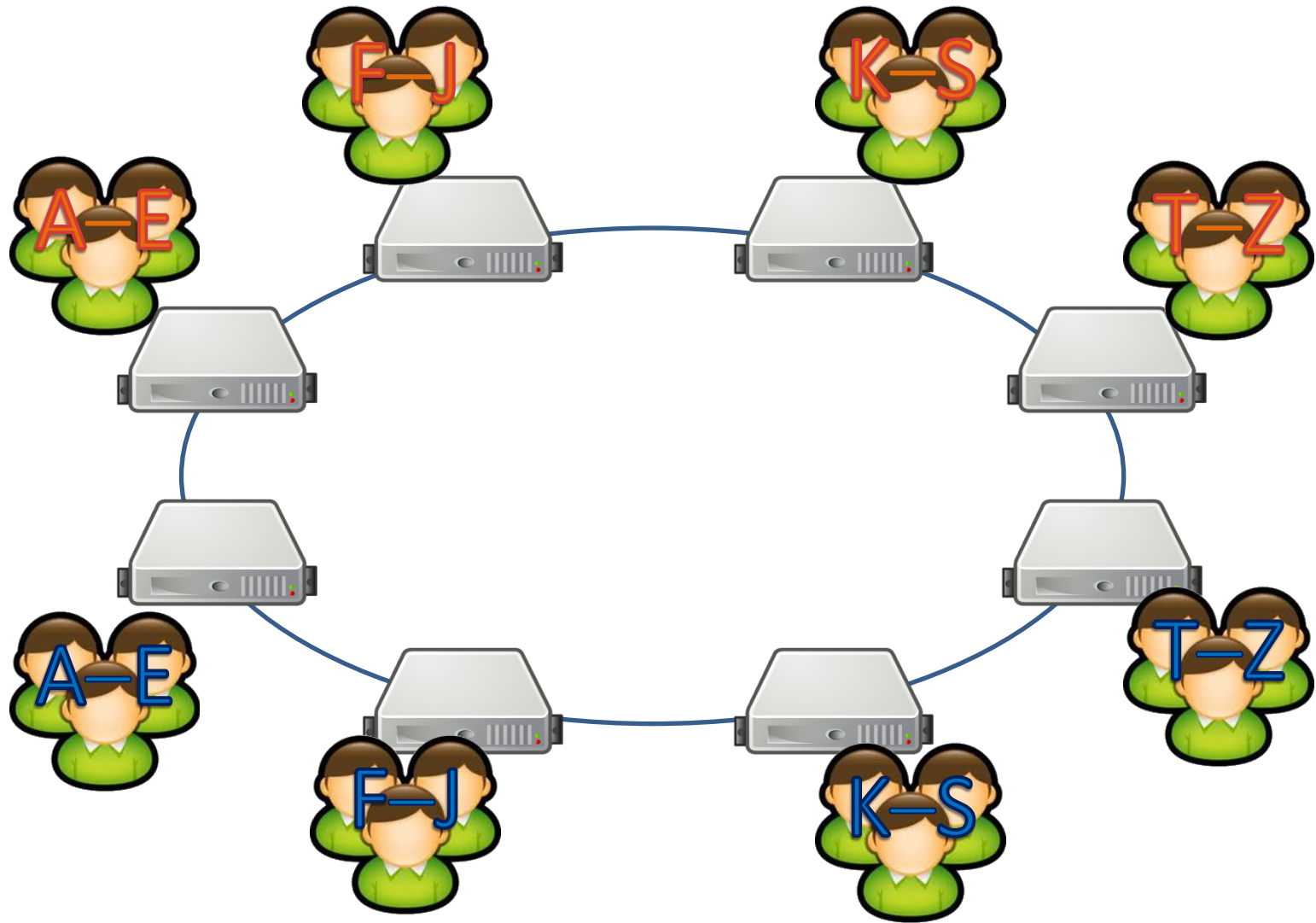
## 3. Partition-tolerance:

- The system works even if messages are lost

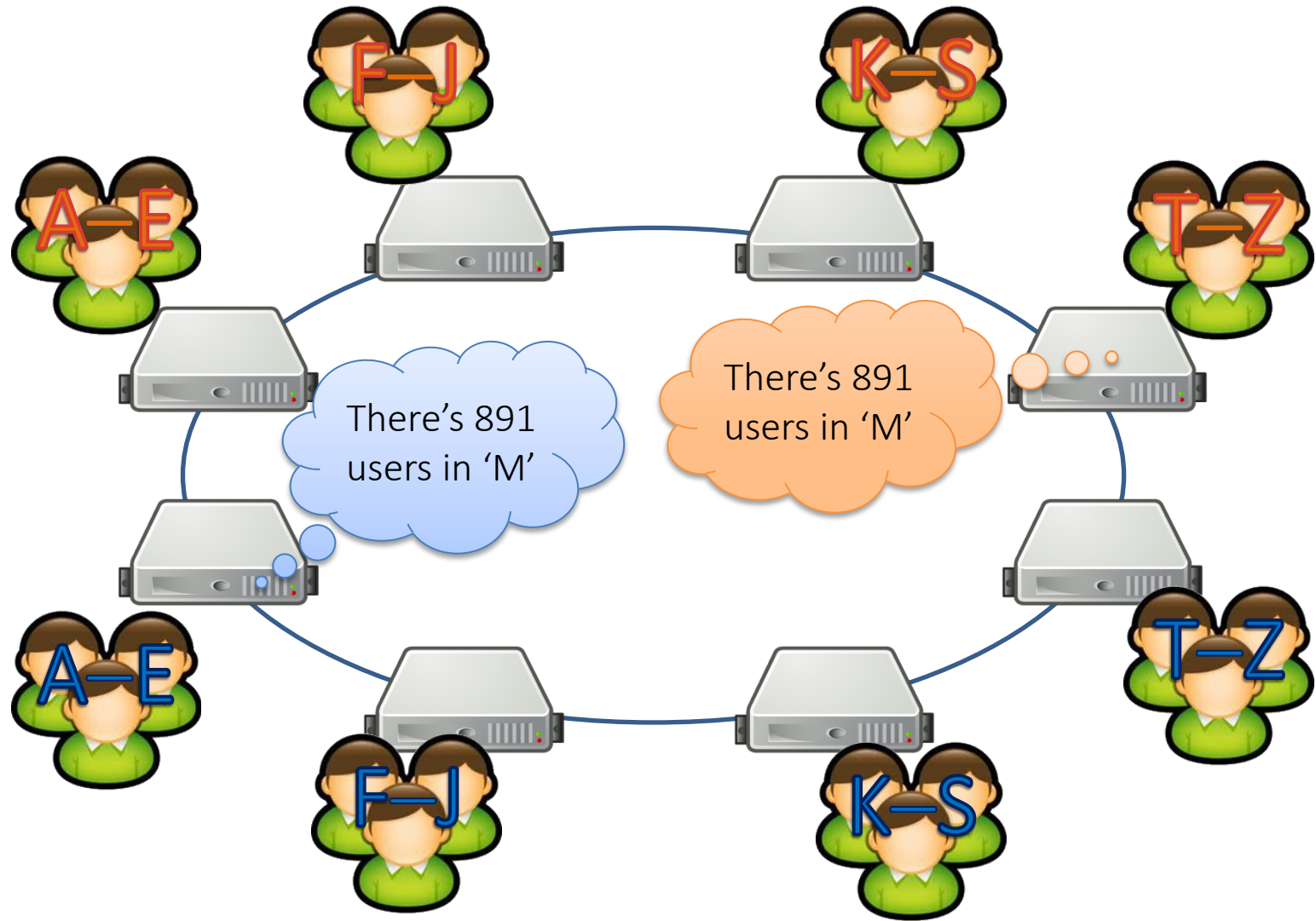
CA in CAP not the same as CA in ACID!!



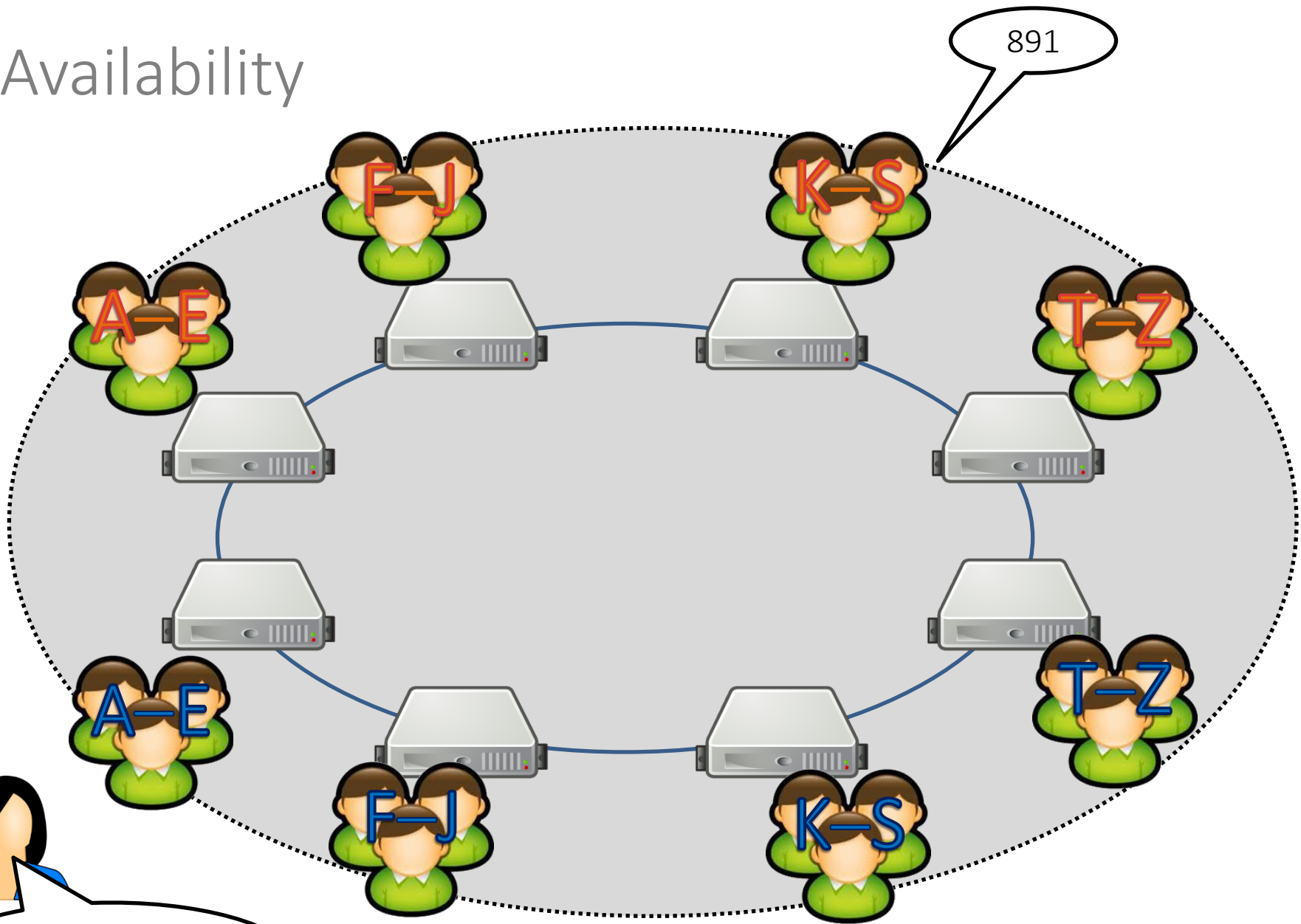
# A Distributed System (with Replication)



# Consistency



# Availability

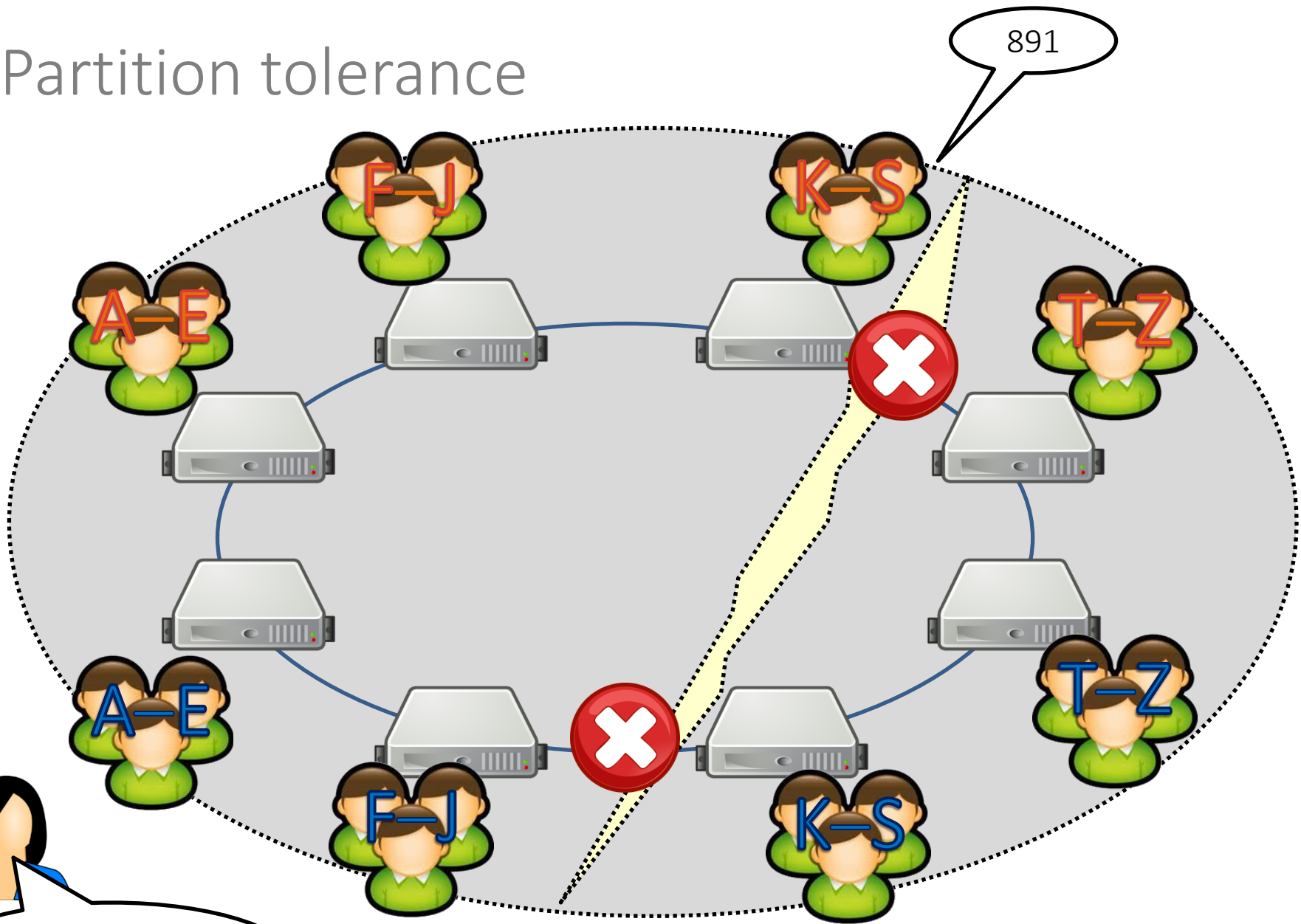


891

How many users start with 'M'



# Partition tolerance



891

How many users start with 'M'

# The CAP Question

Can a distributed system guarantee

**consistency** (all nodes have the same up-to-date view),

**availability** (every read/write is acted upon) **and**

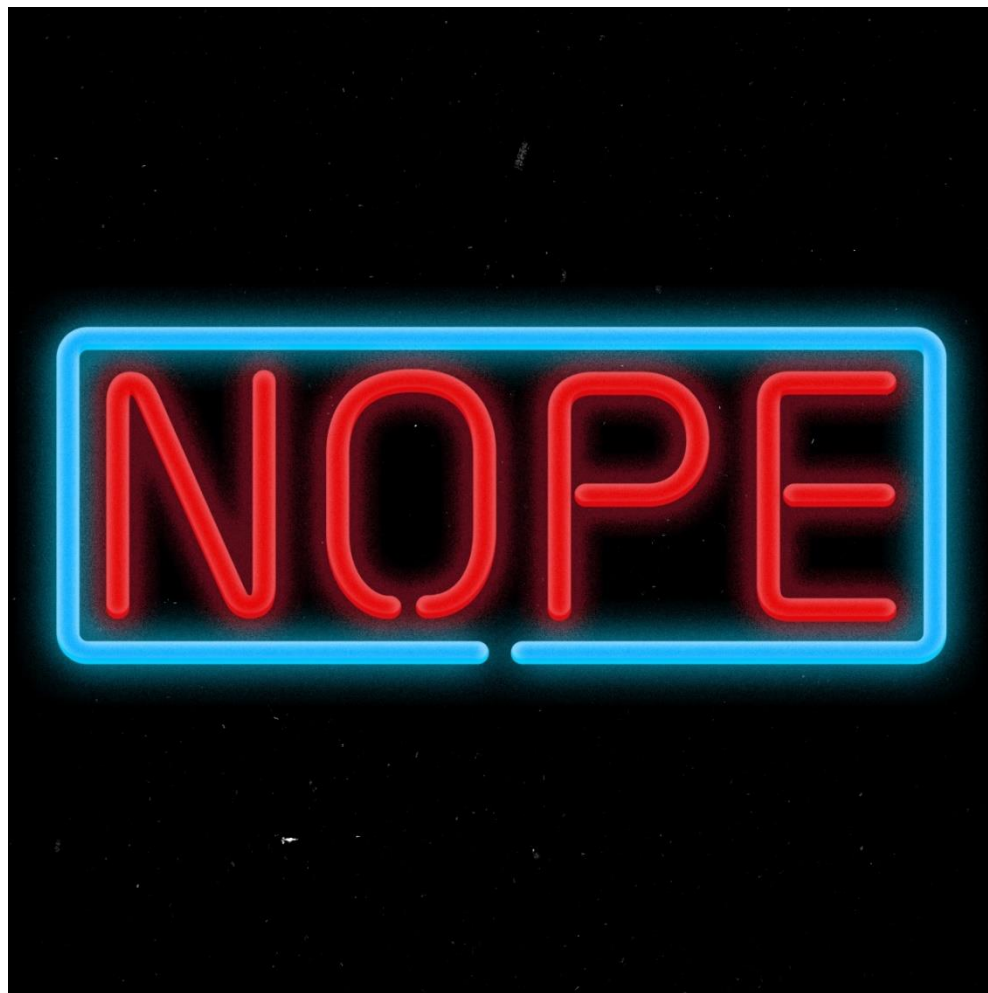
**partition-tolerance** (the system works if messages are lost)

at the same time?

What do you think?



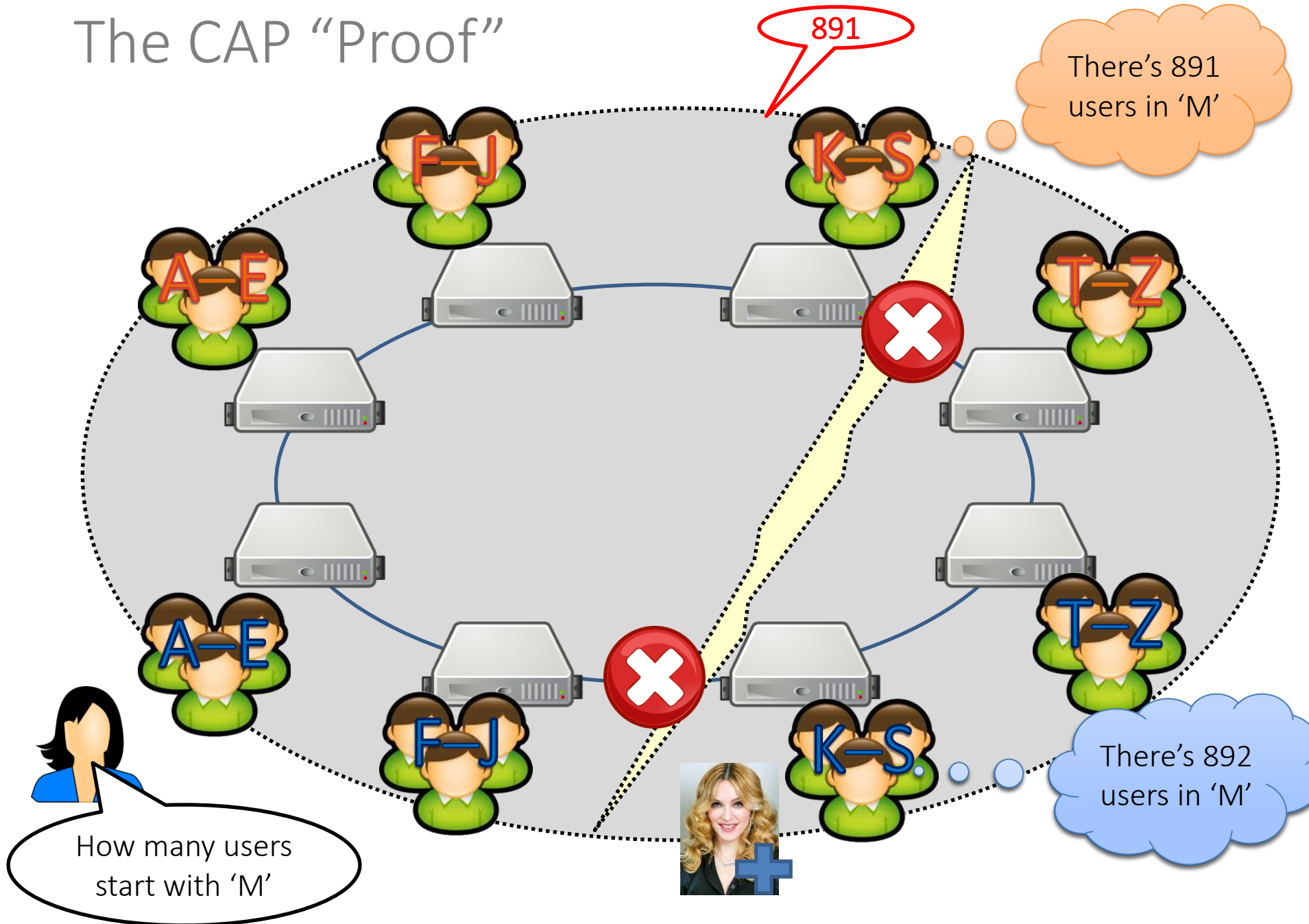
# The CAP Answer



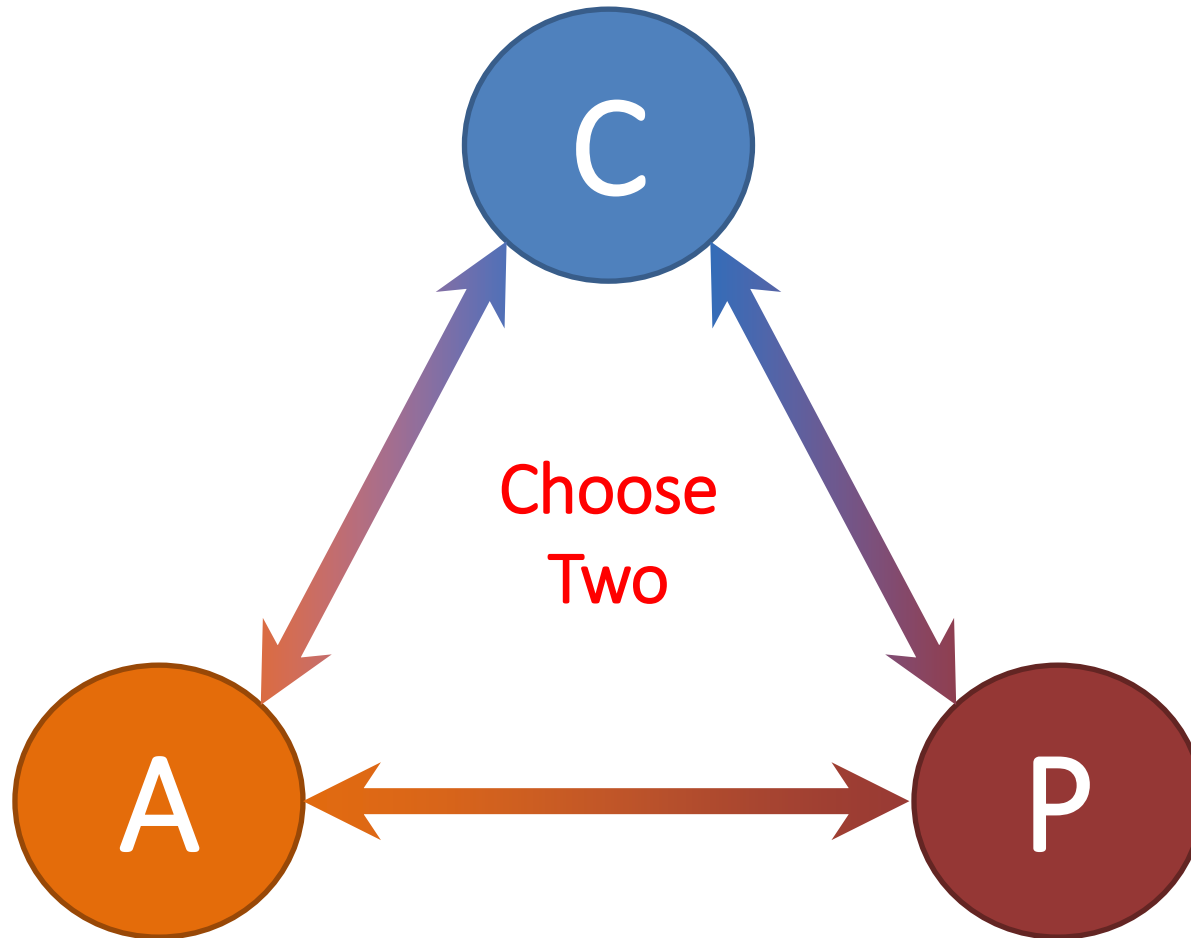
# The CAP Theorem

A distributed system cannot guarantee  
**consistency** (all nodes have the same up-to-date view),  
**availability** (every read/write is acted upon) and  
**partition-tolerance** (the system works if messages are lost)  
at the same time!

# The CAP “Proof”



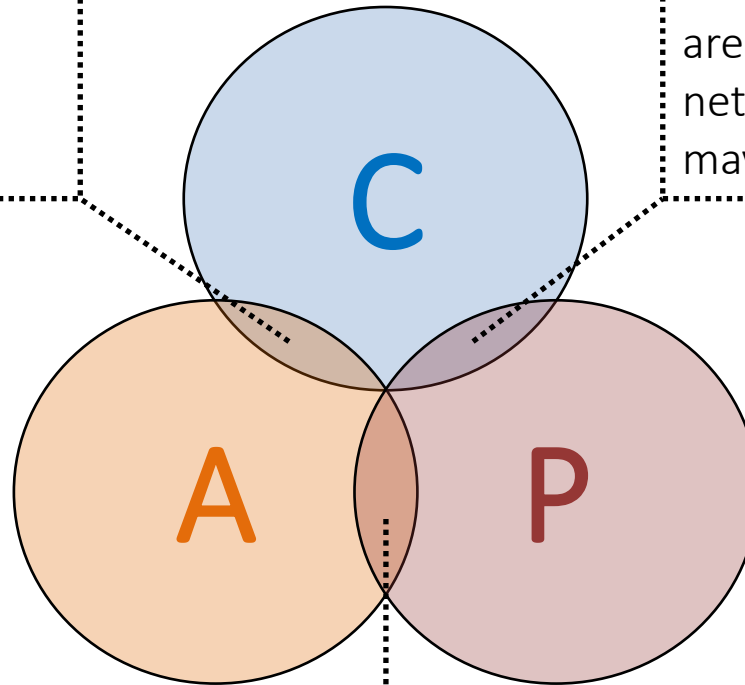
# The CAP Triangle



# CAP Systems

**CA**: Guarantees to give a correct response but only while network works fine (*Centralised / Traditional*)

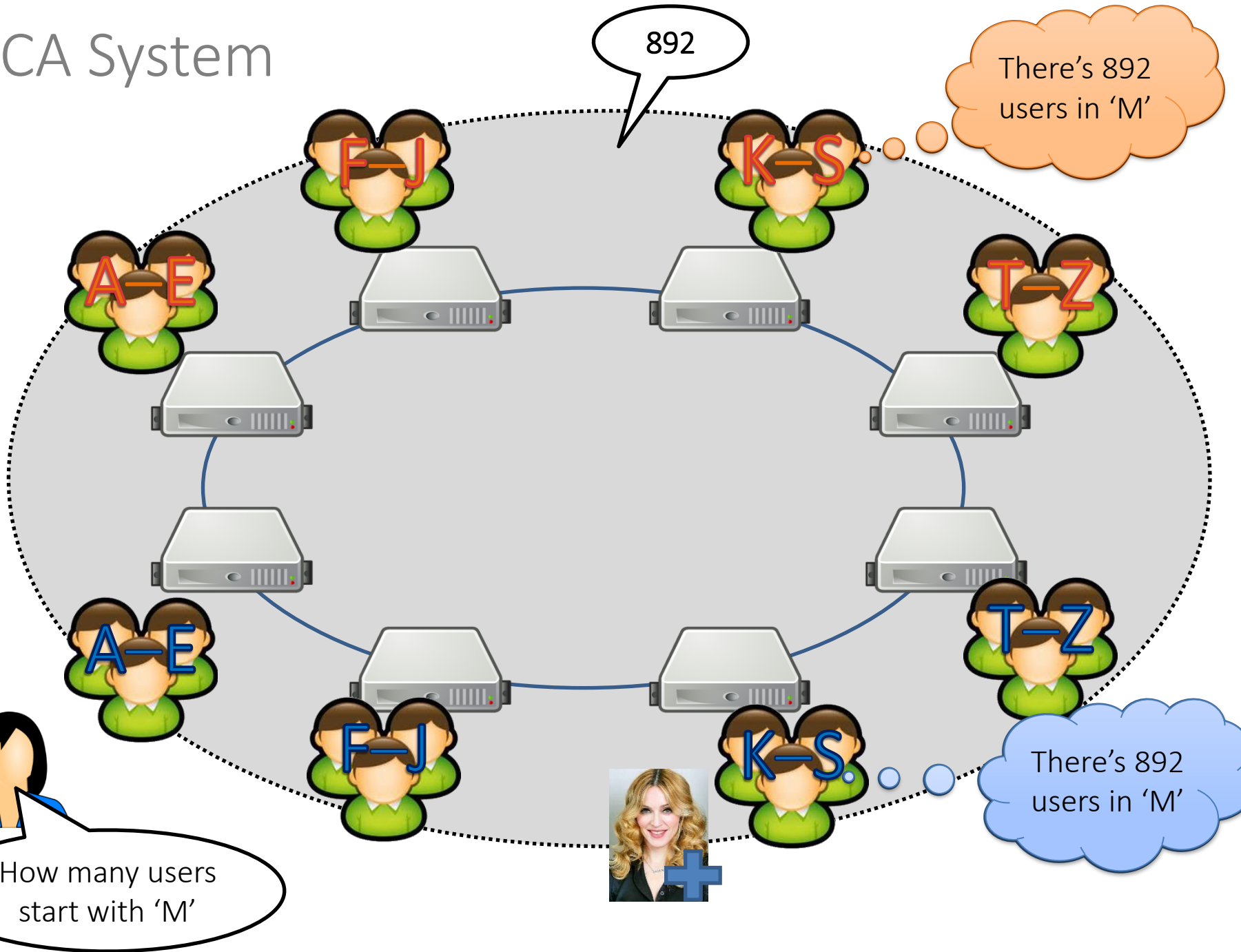
**CP**: Guarantees responses are correct even if there are network failures, but response may fail (*Weak availability*)



(No intersection)

**AP**: Always provides a “best-effort” response even in presence of network failures (*Eventual consistency*)

# CA System

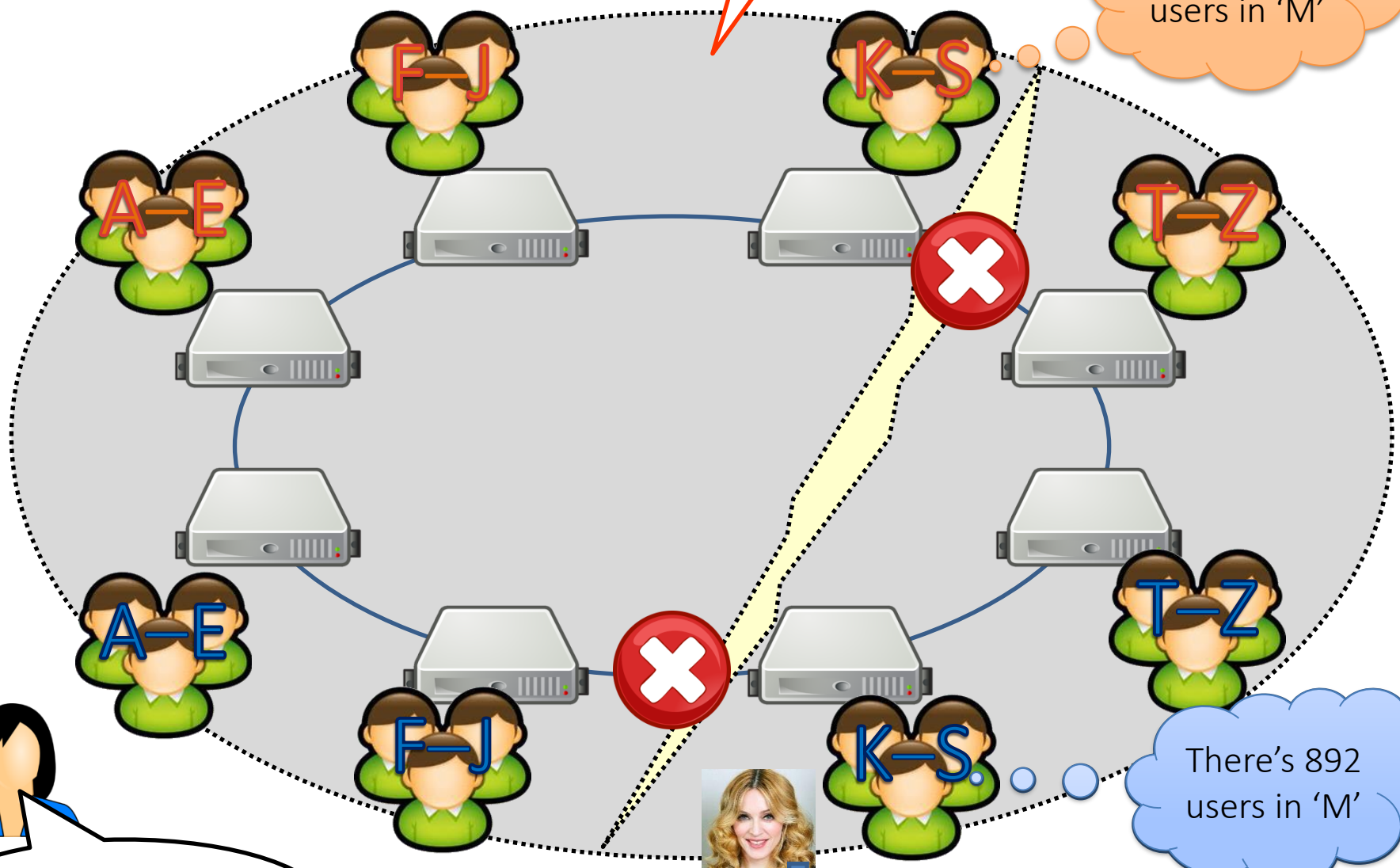




# CP System

Error

There's 891 users in 'M'



How many users start with 'M'

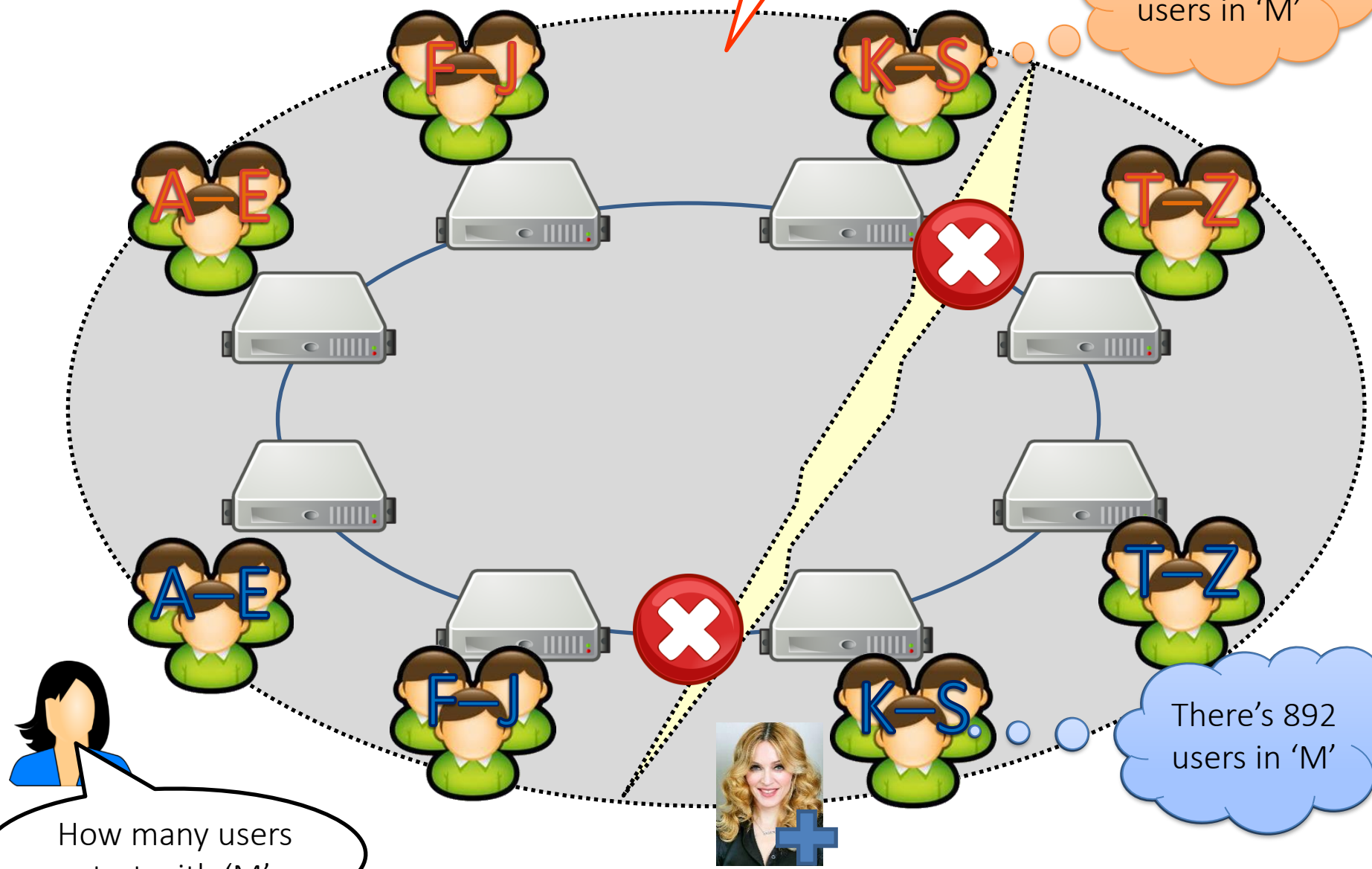
There's 892 users in 'M'



# AP System

891

There's 891 users in 'M'



How many users start with 'M'

There's 892 users in 'M'

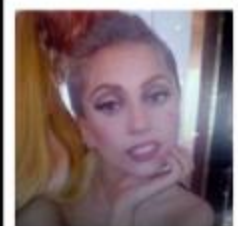
# BASE (AP)

- **B**asically **A**vailable
  - Almost always “up”
- **S**oft State
  - Replicated, cached data
- **E**ventual Consistency
  - Stale data tolerated, for a while

In what way does Twitter act as a BASE (AP) system?



# High-fanout creates a “partition”



**@ladygaga** ✓  
31 million followers

Users may see retweets of celebrity tweets before the original tweet.

Later when the original tweet arrives the timeline will be reordered and made consistent.

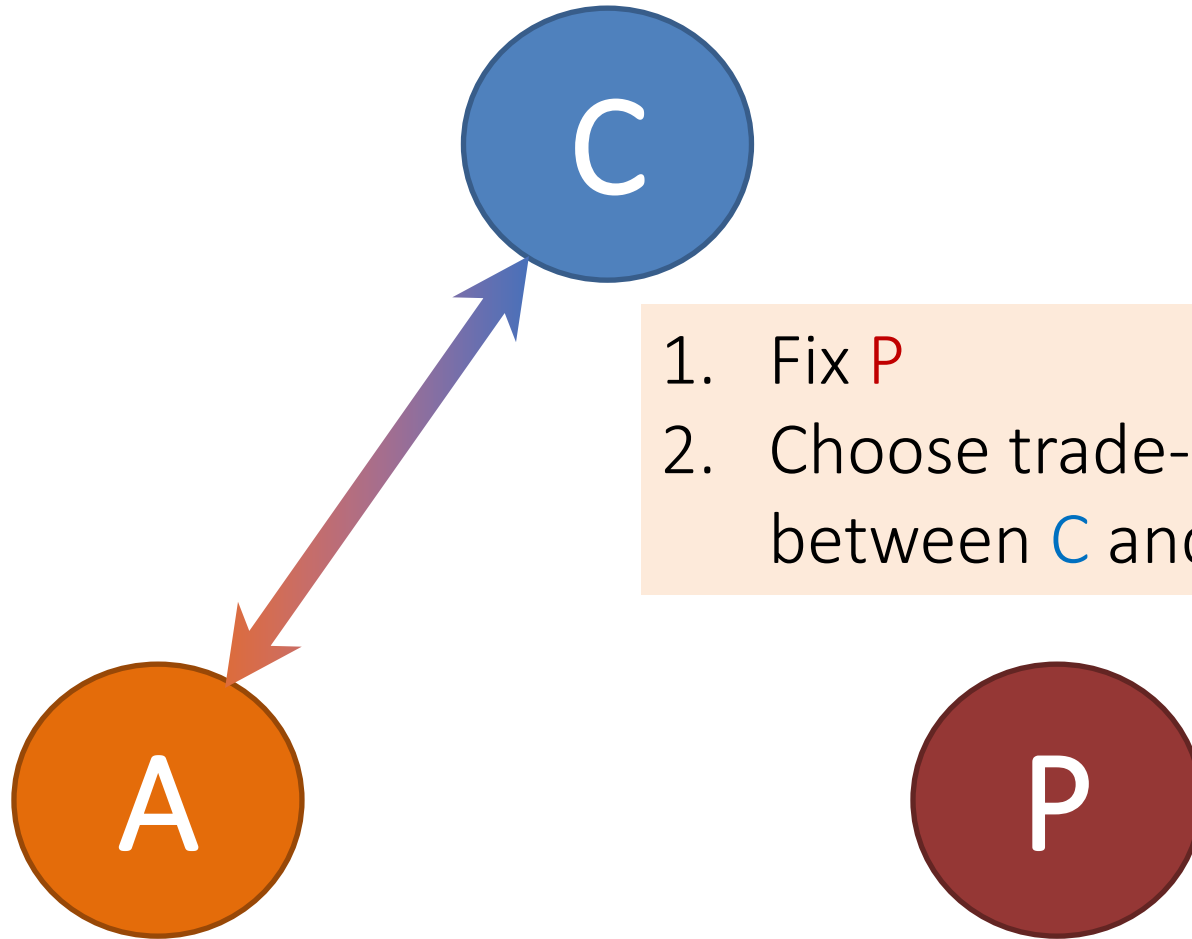


28 million followers



**@barackobama** ✓  
23 million followers

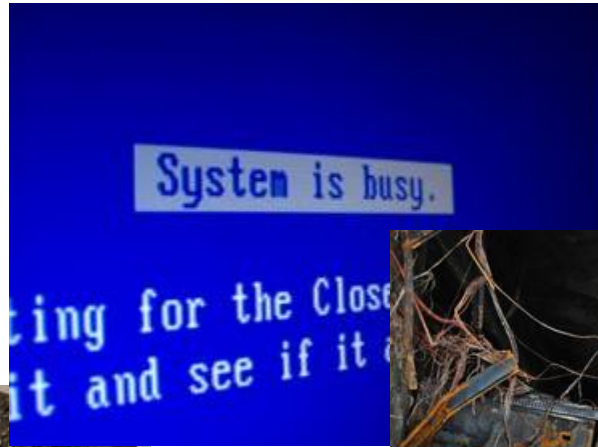
# CAP in practical distributed systems



1. Fix **P**
2. Choose trade-off point between **C** and **A**

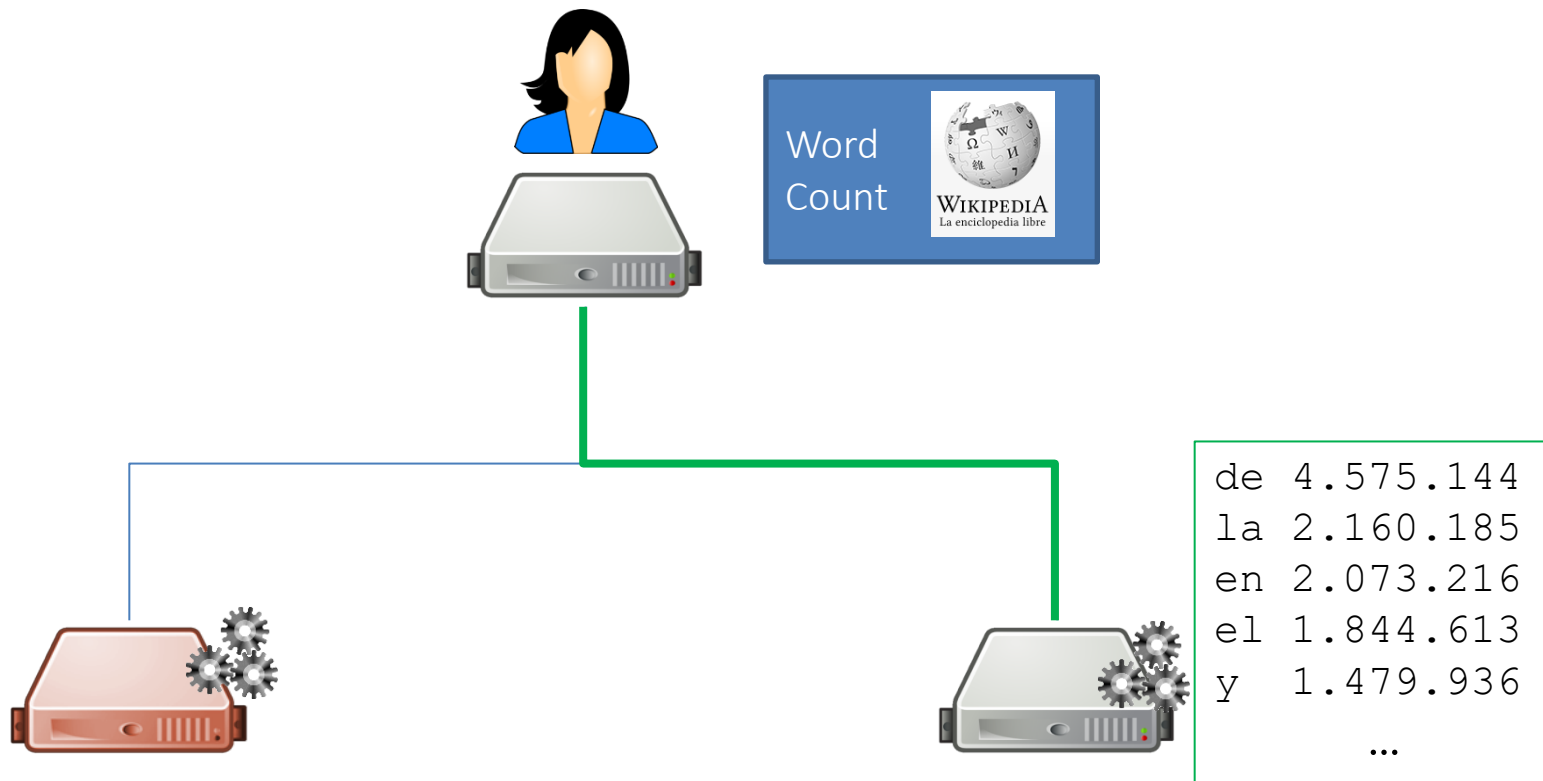
# PARTITION TOLERANCE

# Faults



# Fail-Stop Fault

- A machine fails to respond or times-out
  - often hardware or load
  - need at least  $f + 1$  replicated machines
    - $f$  = number of fail-stop failures

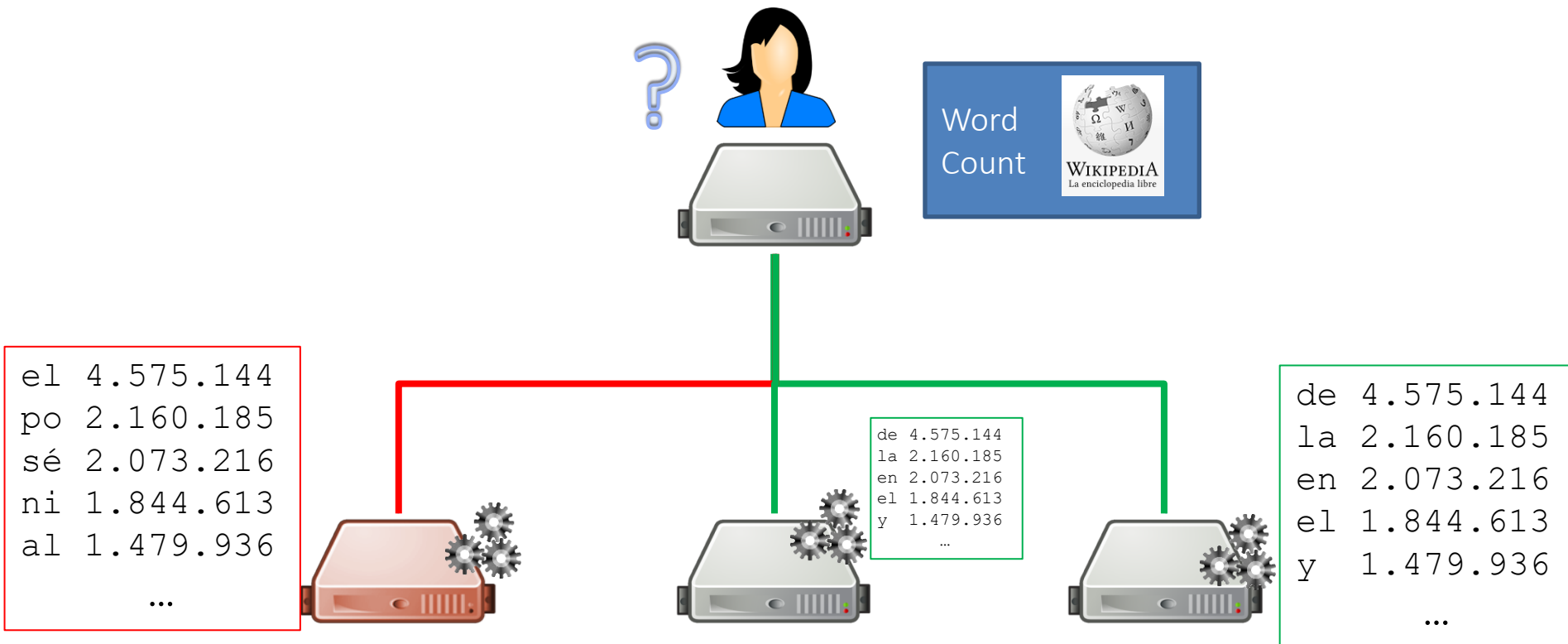




# Byzantine Fault

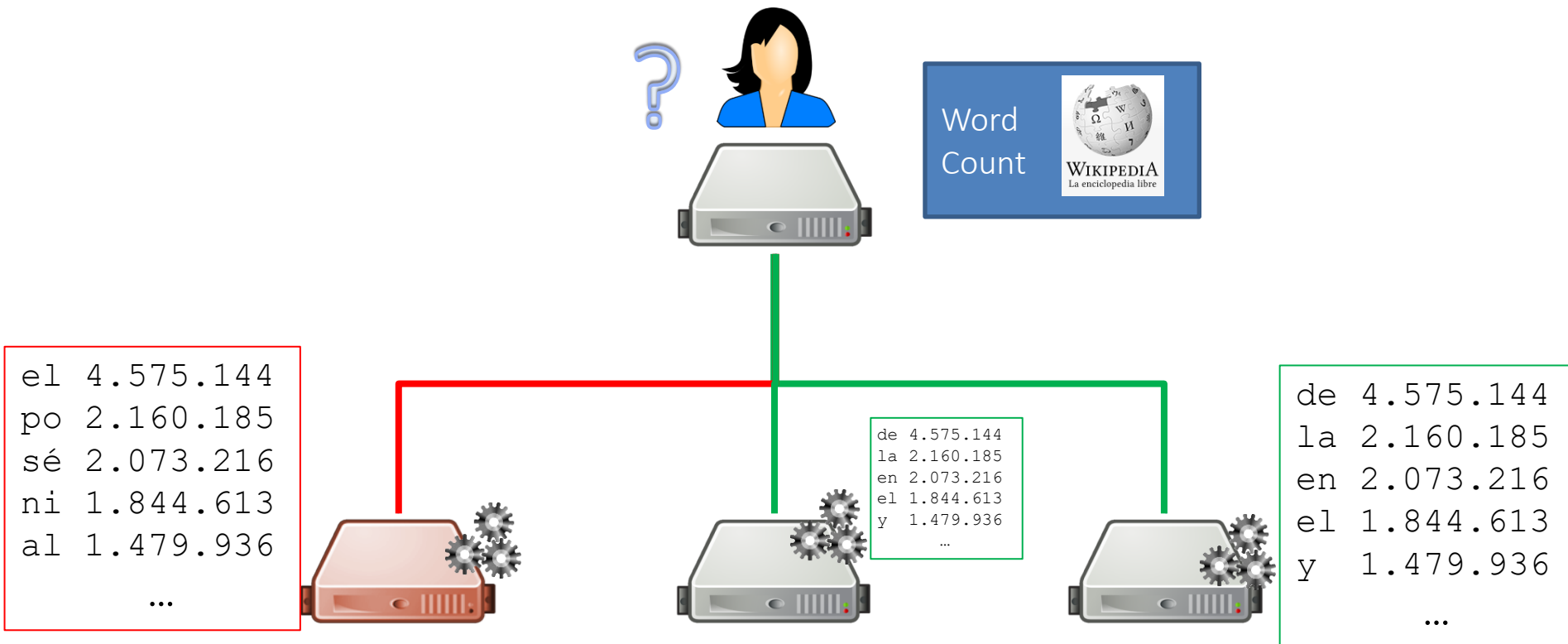
- A machine responds incorrectly/maliciously

How many working machines do we need in the general case to be robust against Byzantine faults?



# Byzantine Fault

- A machine responds incorrectly/maliciously
  - Need *at least*  $2f+1$  replicated machines
    - $f$  = number of (possibly Byzantine) failures



# DISTRIBUTED CONSENSUS

# Distributed Consensus

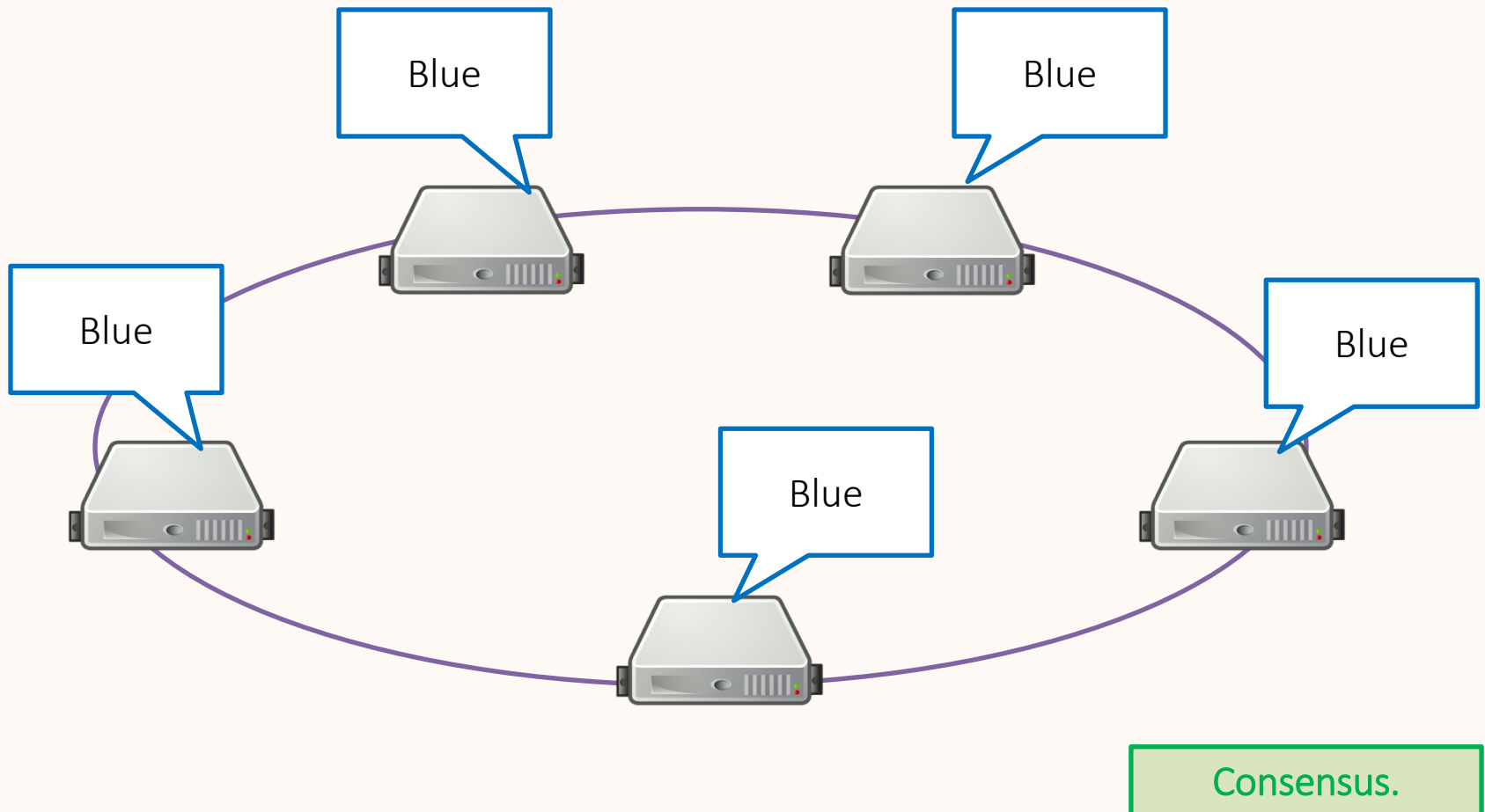


Colour of the dress?



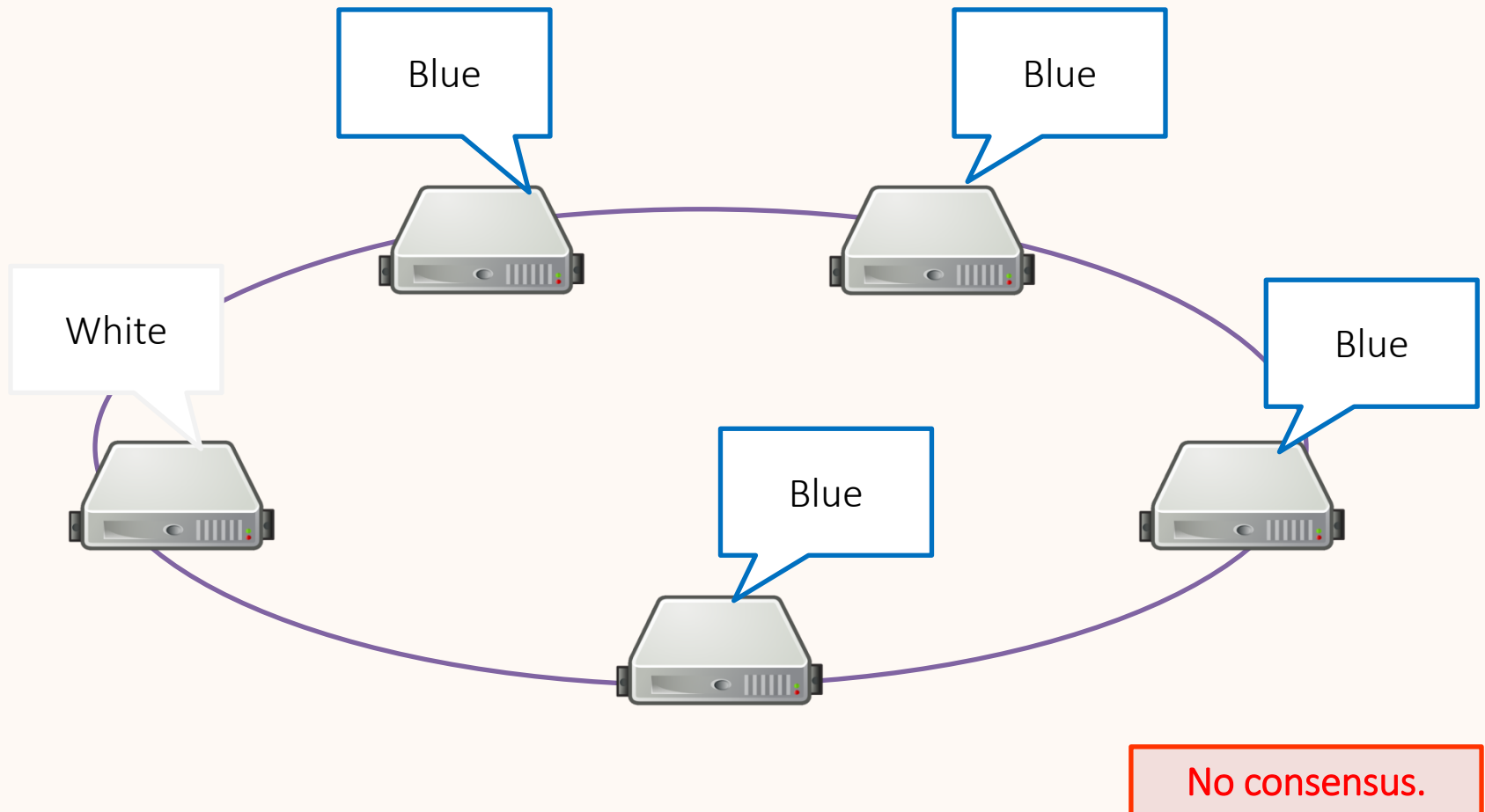
# Distributed Consensus

Strong consensus: All nodes need to agree



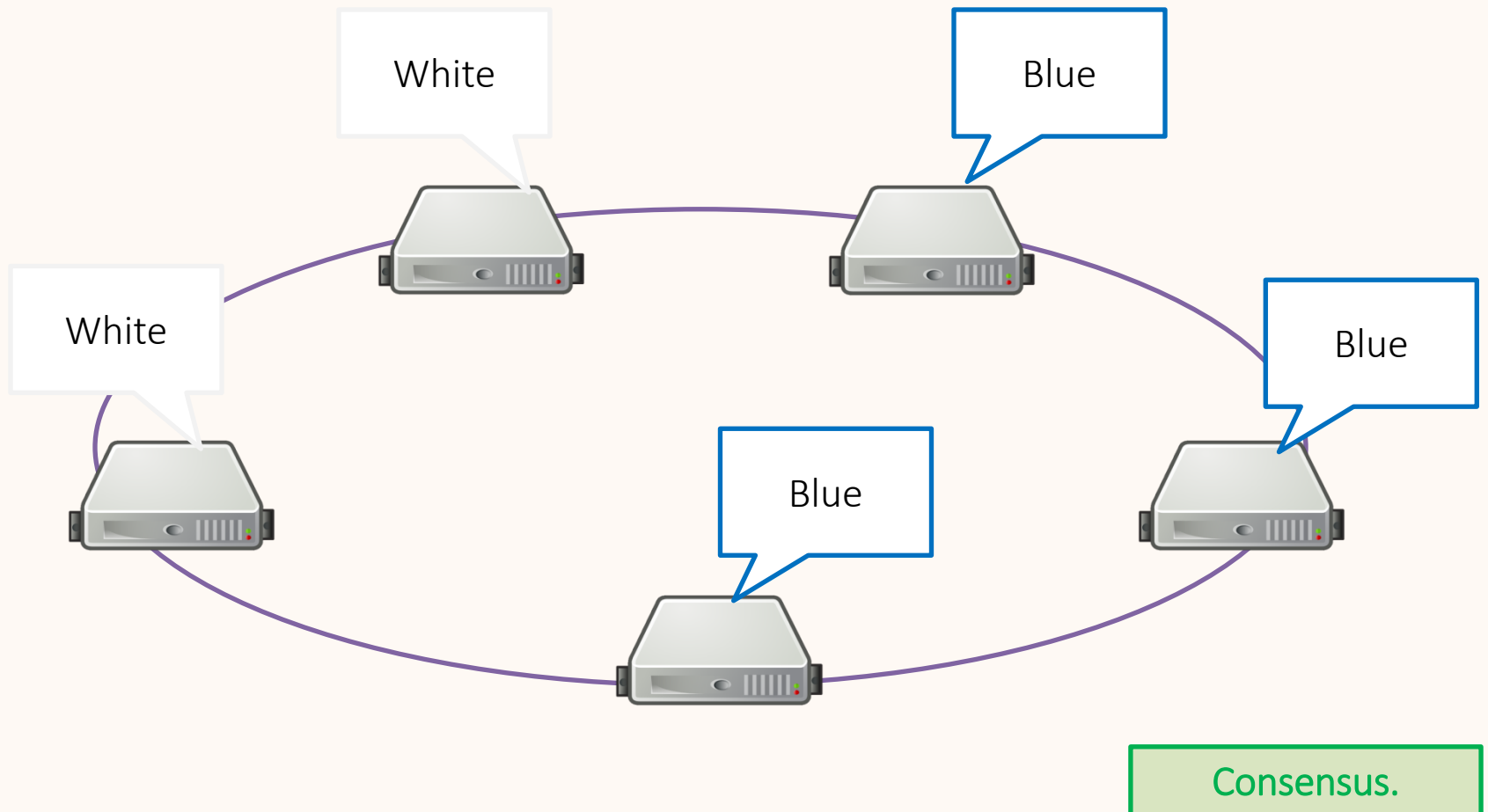
# Distributed Consensus

Strong consensus: All nodes need to agree



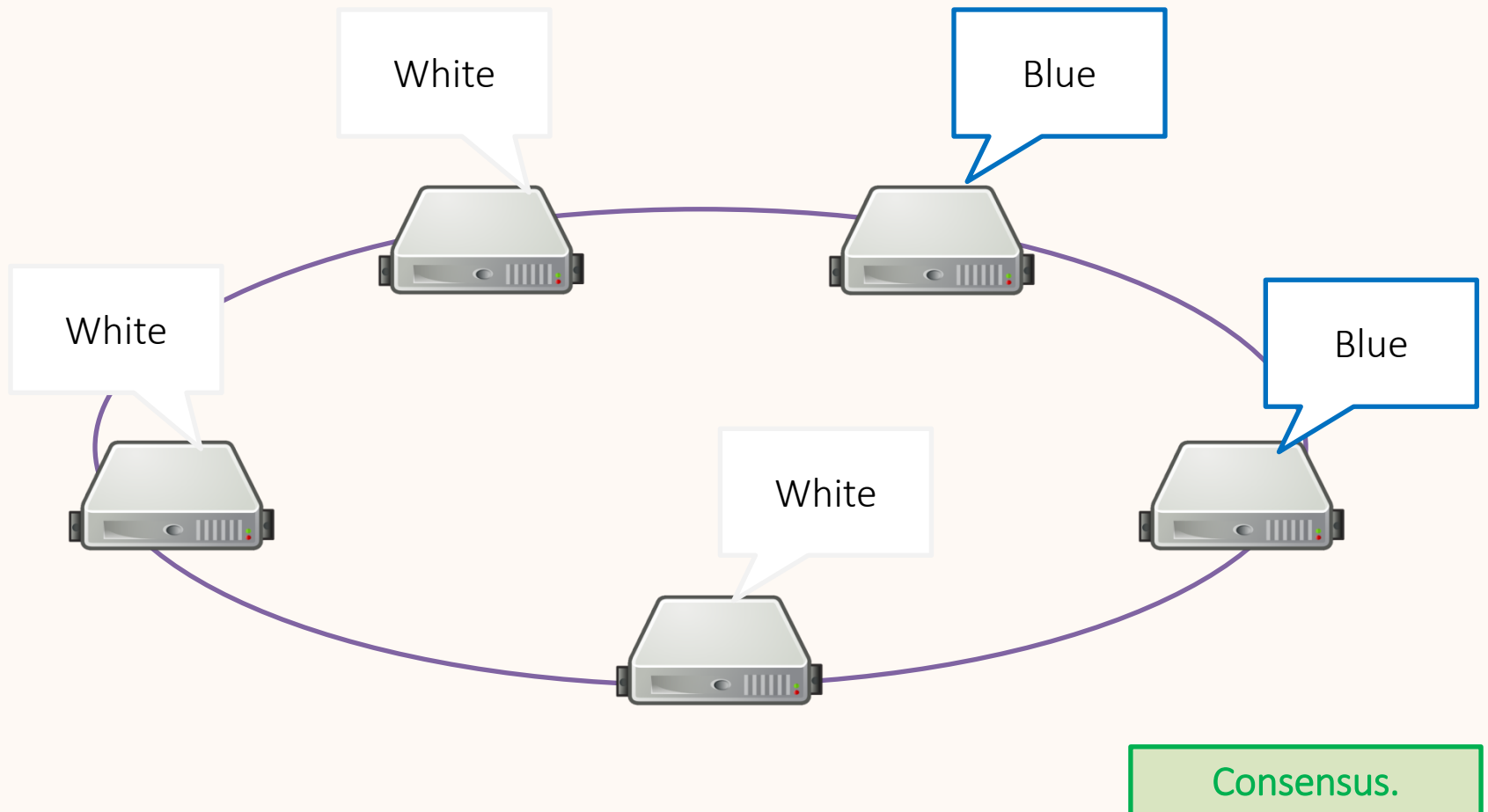
# Distributed Consensus

Majority consensus: A majority of nodes need to agree



# Distributed Consensus

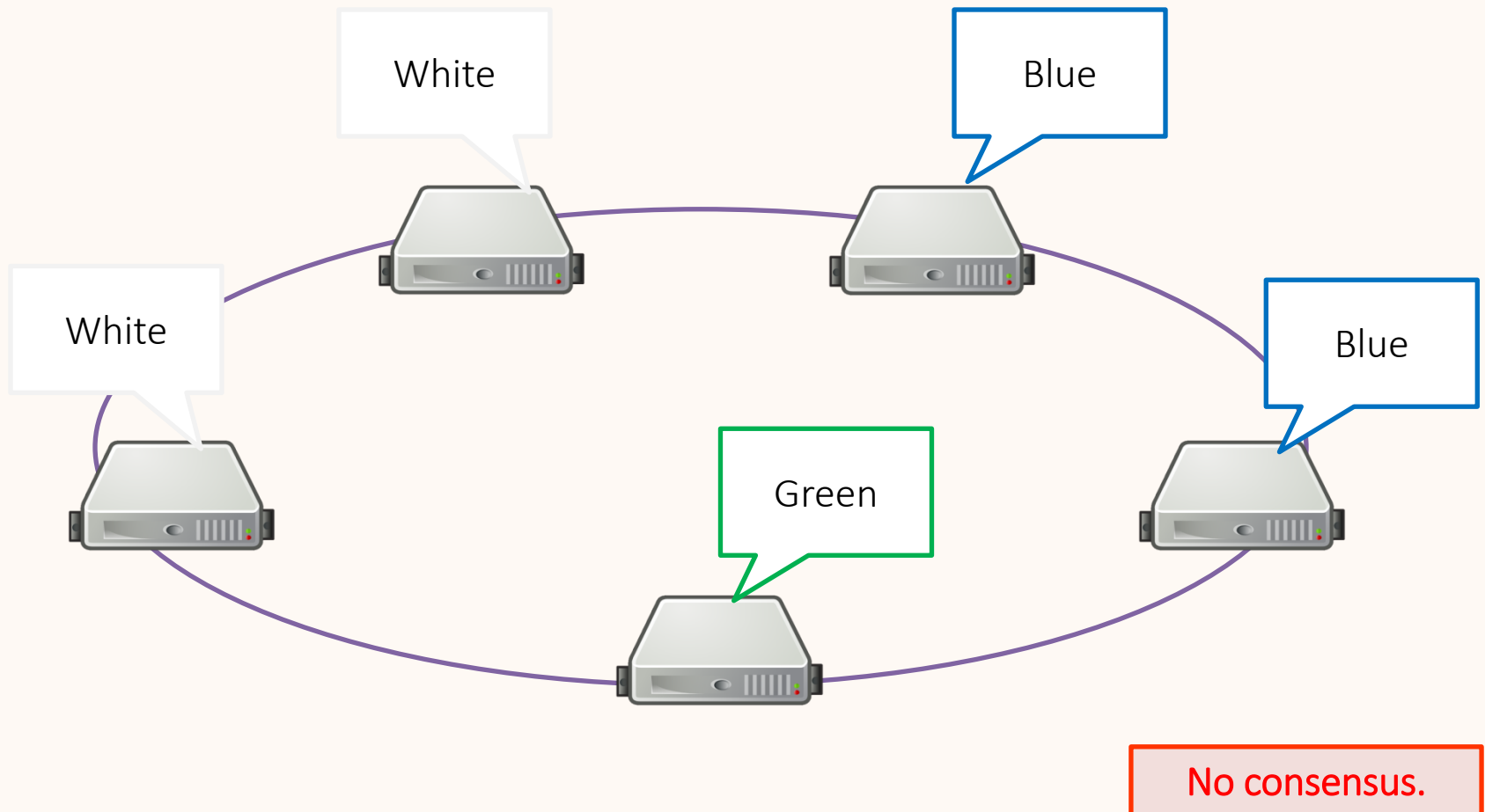
Majority consensus: A majority of nodes need to agree





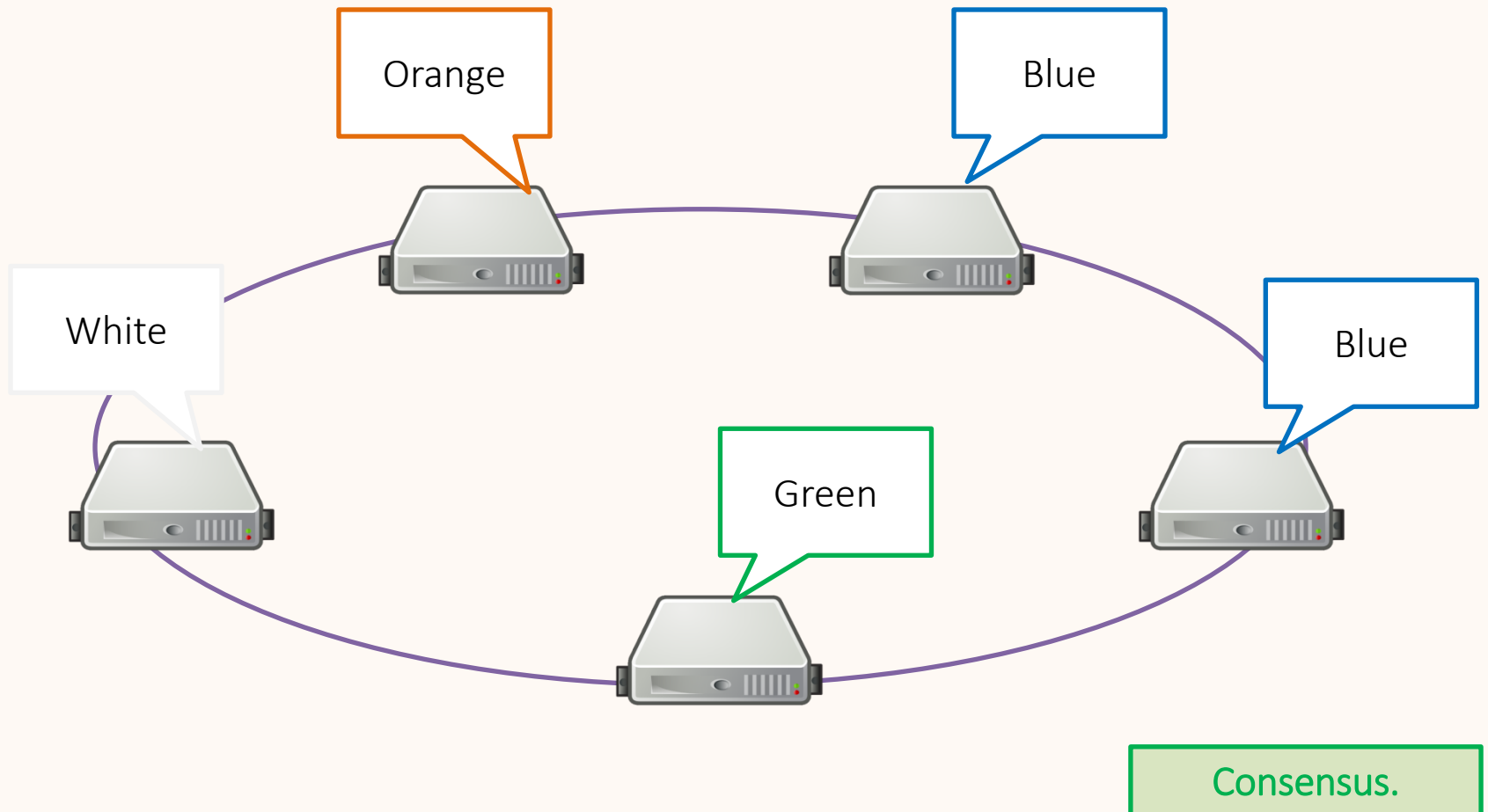
# Distributed Consensus

Majority consensus: A majority of nodes need to agree



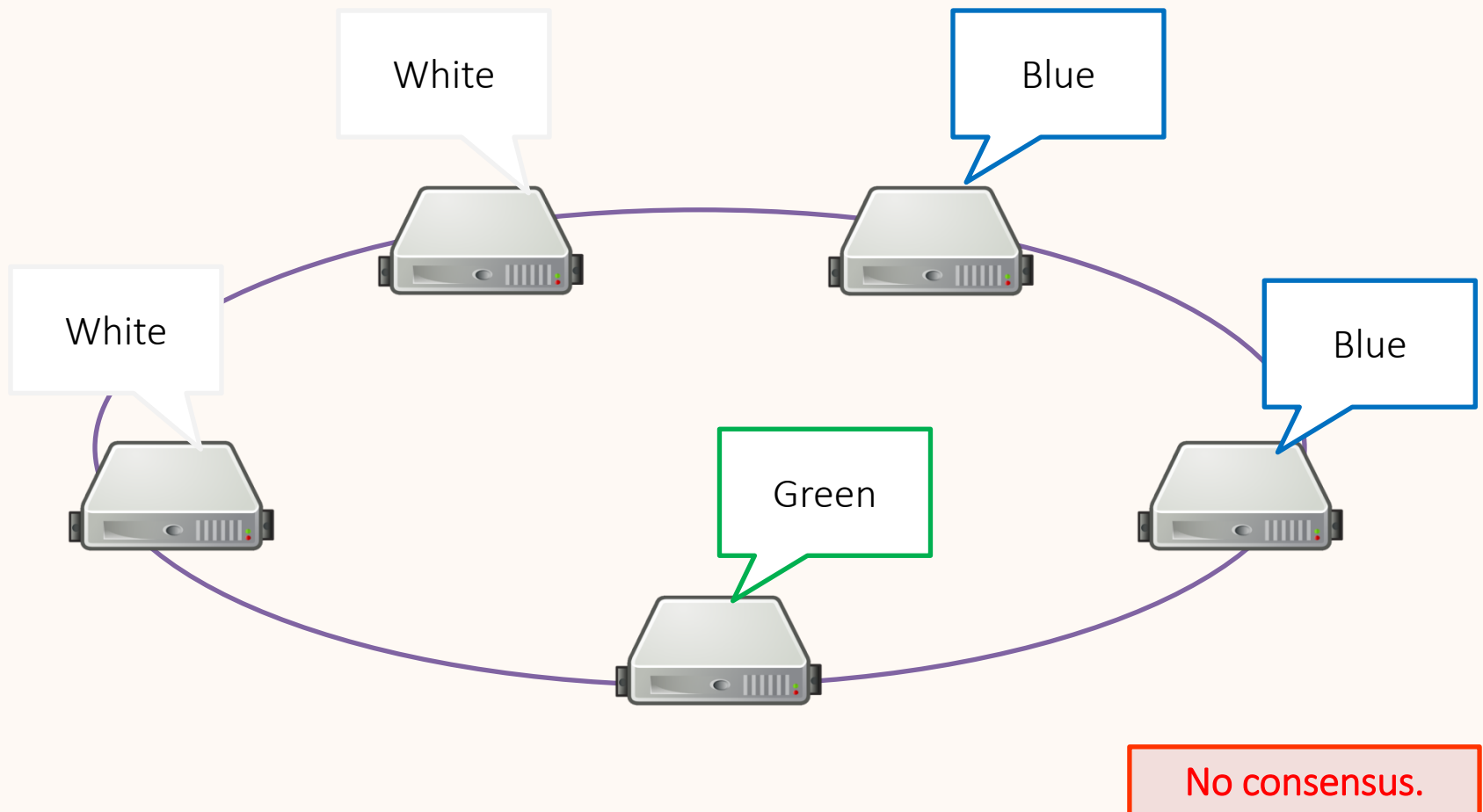
# Distributed Consensus

Plurality consensus: A plurality of nodes need to agree



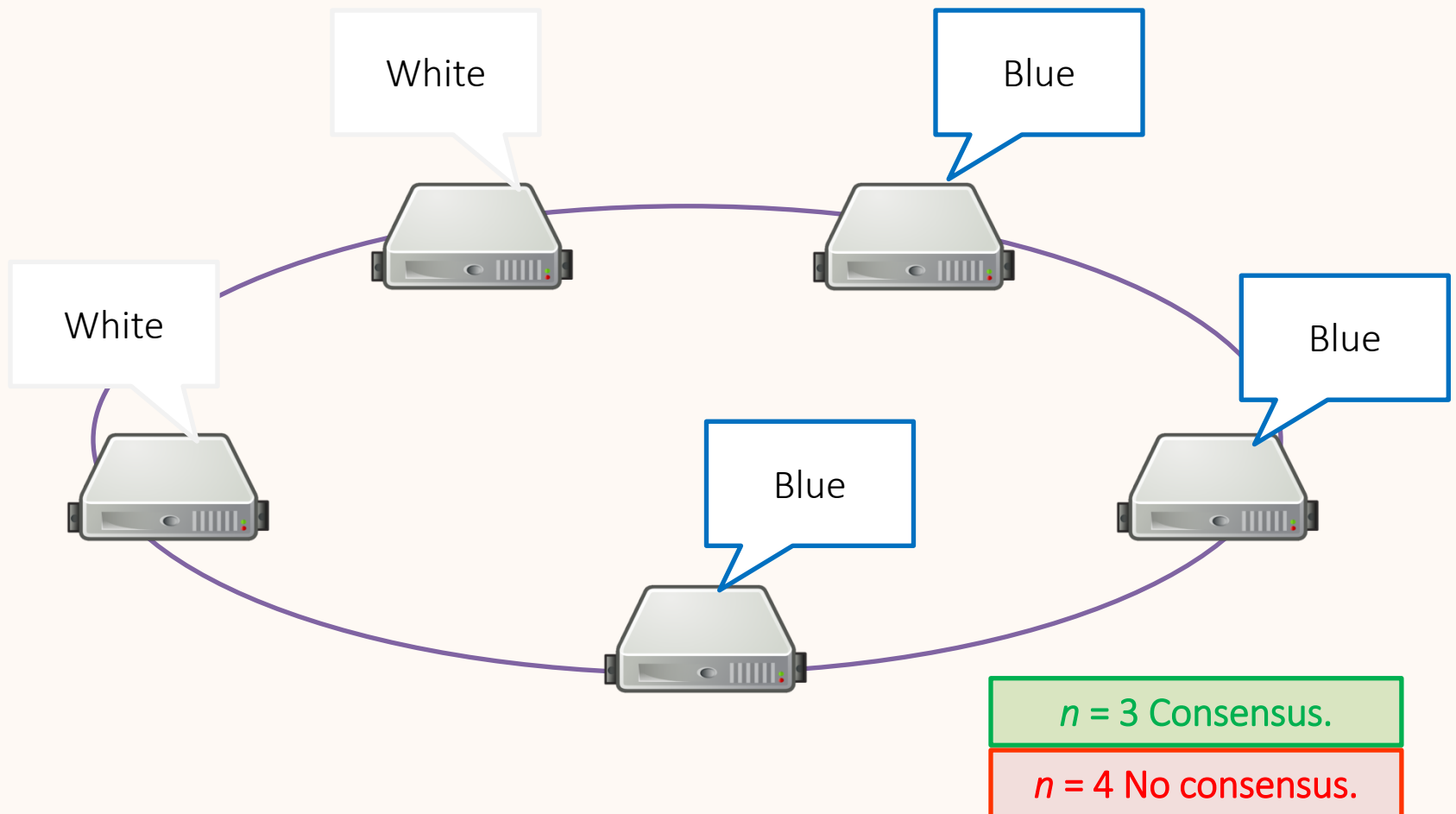
# Distributed Consensus

Plurality consensus: A plurality of nodes need to agree



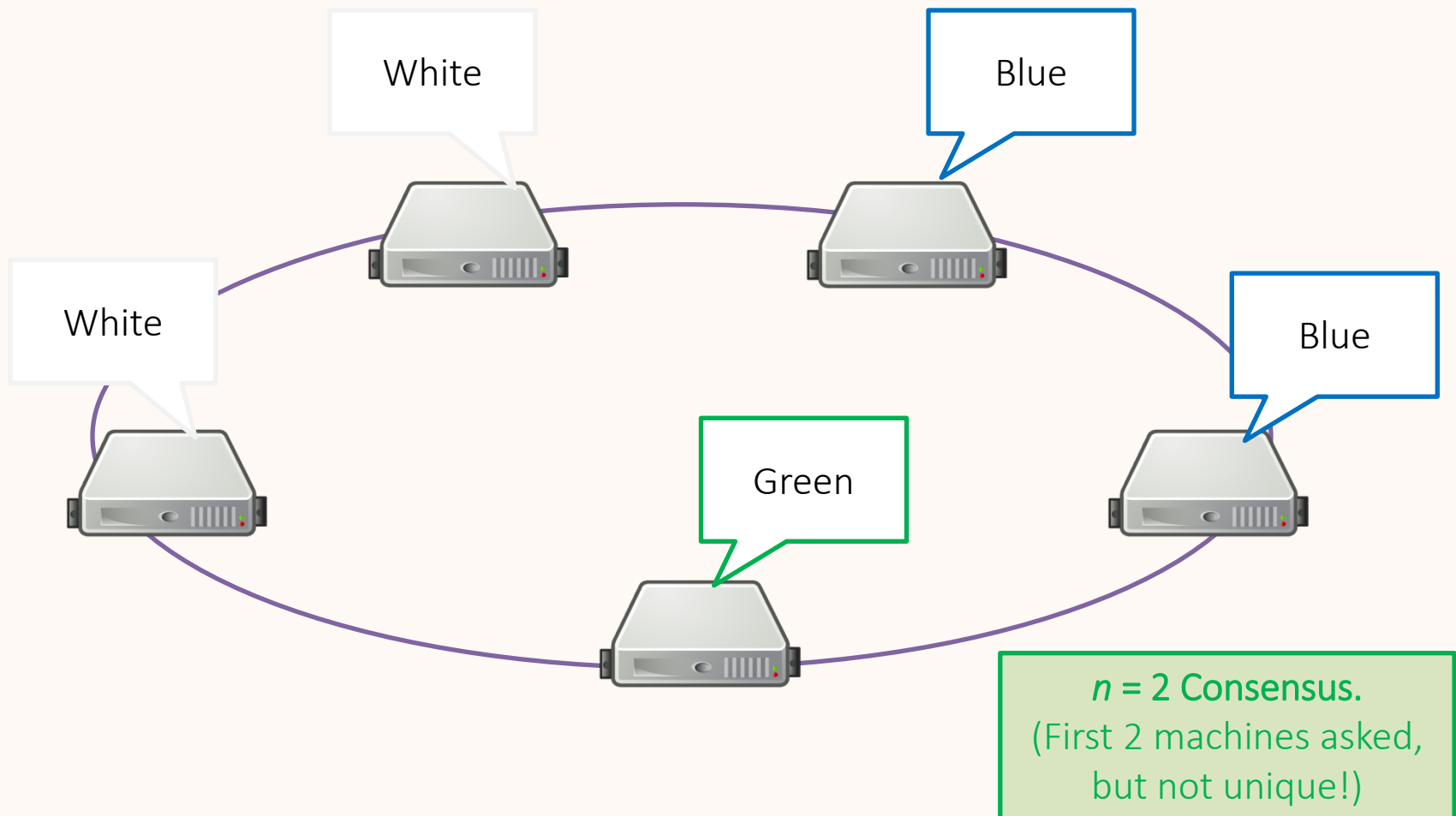
# Distributed Consensus

Quorum consensus:  $n$  nodes need to agree



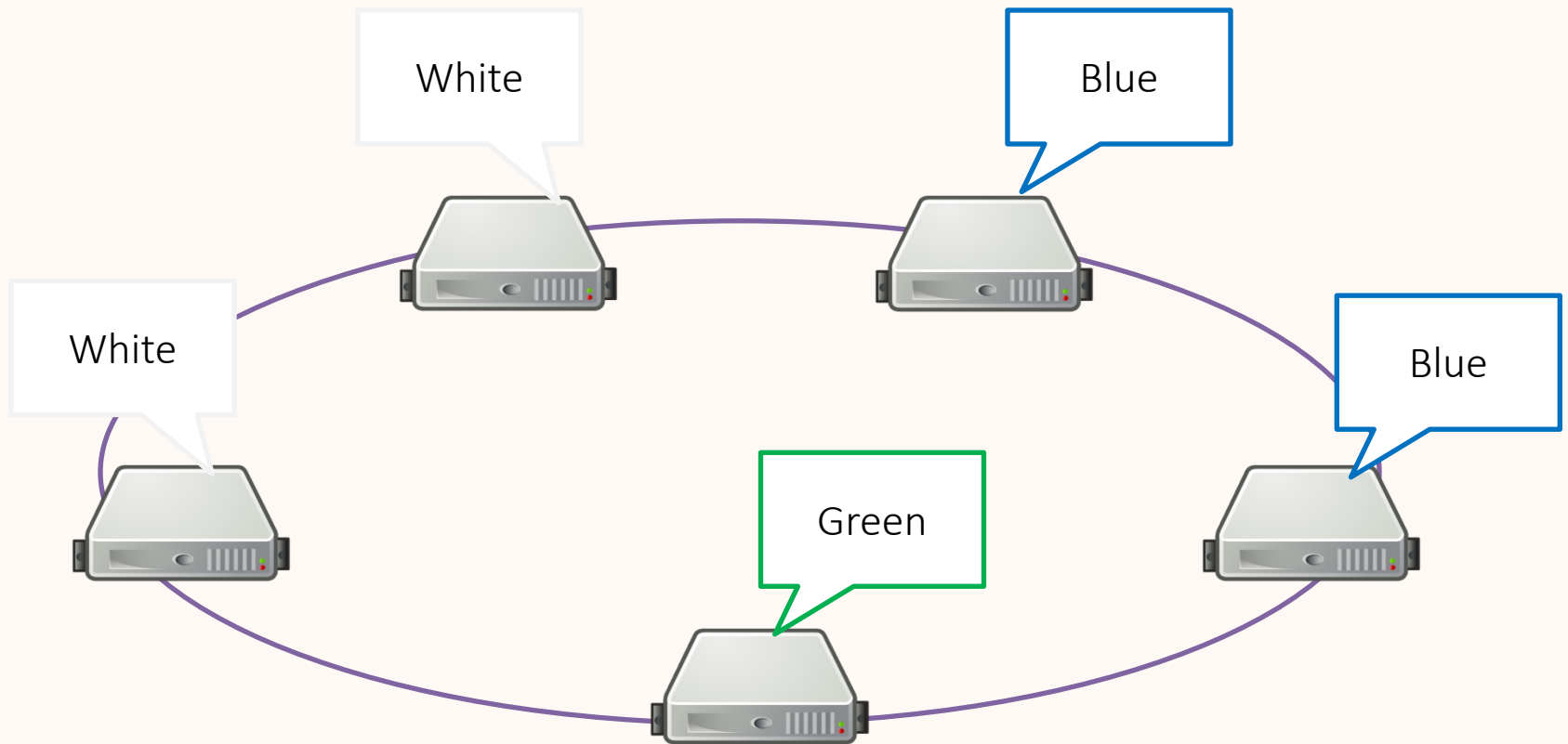
# Distributed Consensus

Quorum consensus:  $n$  nodes need to agree



# Distributed Consensus

Quorum consensus:  $n$  nodes need to agree



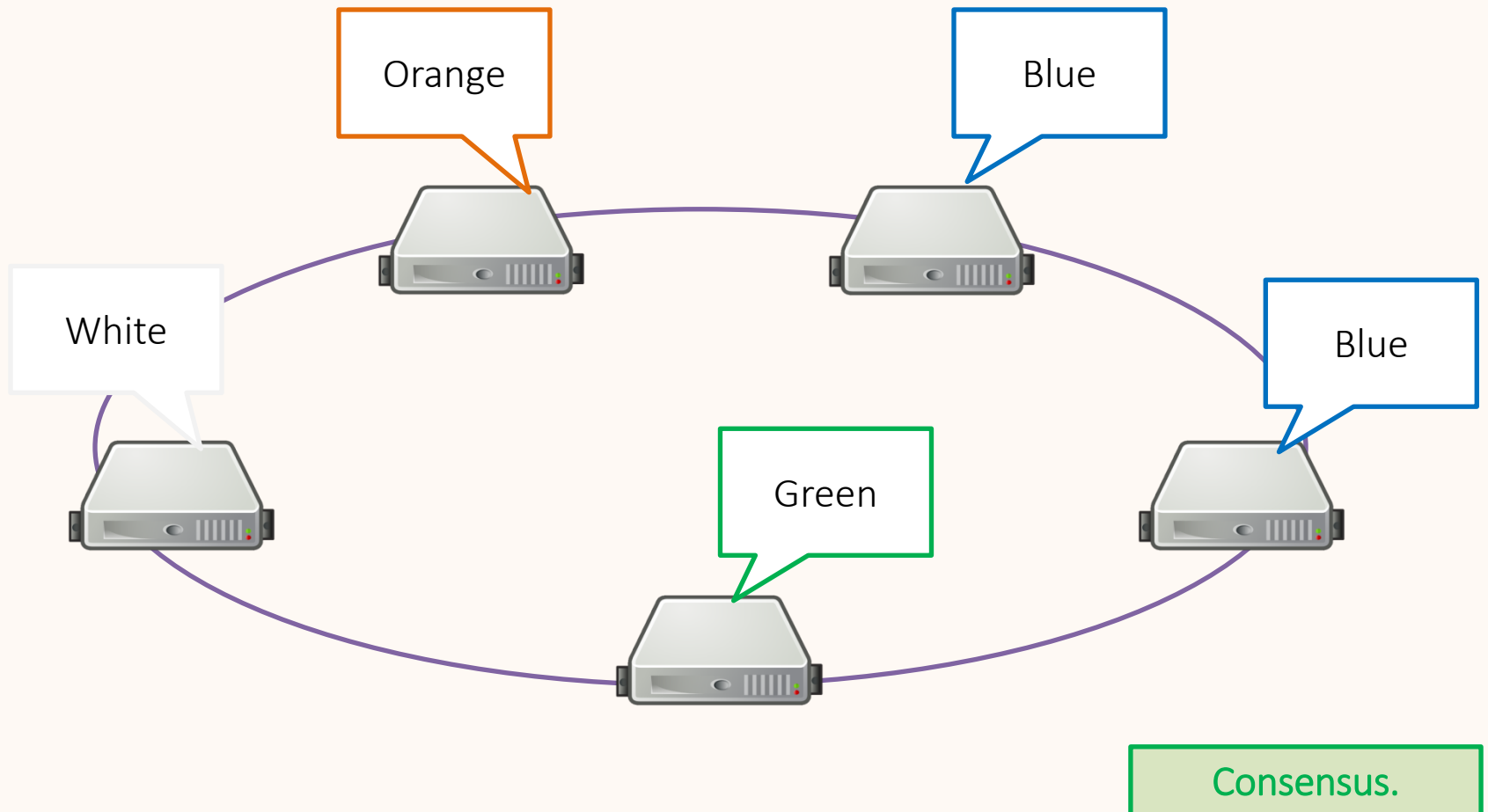
Value of  $n$  needed for unique consensus with  $N$  nodes?



$n > N/2$

# Distributed Consensus

Consensus off: Take first answer



# Distributed Consensus

CP vs. AP?



Strong consensus: All nodes need to agree

CP

Majority consensus: A majority of nodes need to agree

Plurality consensus: A plurality of nodes need to agree

Quorum consensus: “Fixed”  $n$  nodes need to agree

Consensus off: Take first answer

AP



# Distributed Consensus

Scale?



Strong consensus: All nodes need to agree

More replication

Majority consensus: A majority of nodes need to agree

Plurality consensus: A plurality of nodes need to agree

Quorum consensus: “Fixed”  $n$  nodes need to agree

Consensus off: Take first answer

Less replication

# Distributed Consensus

Strong consensus: All nodes need to agree

Majority consensus: A majority of nodes need to agree

**Choice is application dependent:**

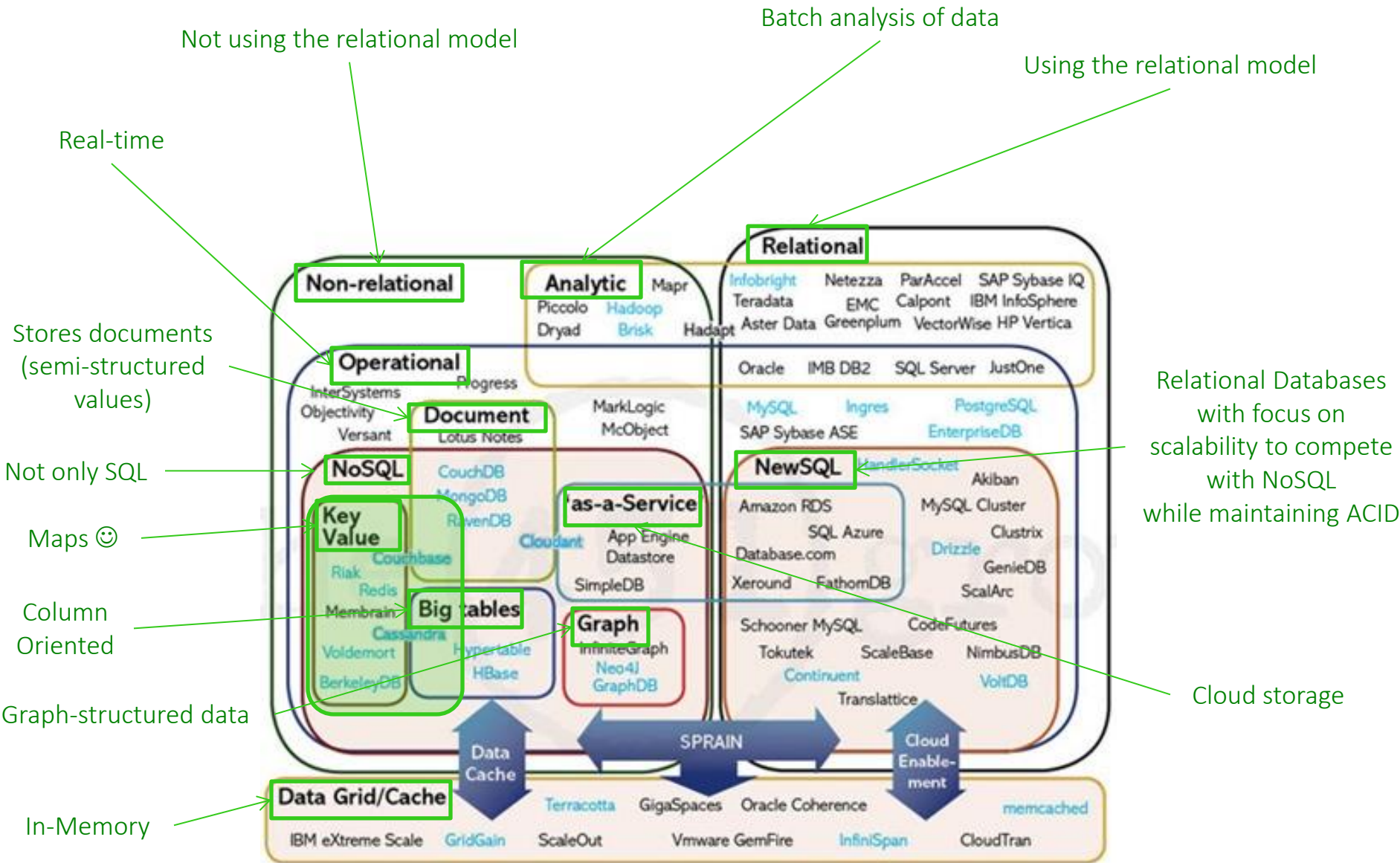
Plurality Many NoSQL stores allow you to choose  
level of consensus/replication

Quorum consensus: “Fixed”  $n$  nodes need to agree

Consensus off: Take first answer

# NoSQL: KEY-VALUE STORES

# The Database Landscape



# Key–Value Store Model

It's just a Map / Associate Array / Dictionary 😊

- `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
...	...

# But You Can Do a Lot With a Map

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}
...	...

*... actually you can model any data in a map (but possibly with a lot of redundancy and inefficient lookups if unsorted).*

# THE CASE OF AMAZON

# The Amazon Scenario

Products Listings: prices, details, stock

The screenshot shows an Amazon search results page for the query "presenter". The top navigation bar includes the Amazon logo, a search bar with "presenter" entered, and a "Shop by Department" dropdown menu. Below the navigation bar, the search results are displayed in a grid. On the left side, there is a sidebar with navigation links for "Office Products", "Computers & Accessories", "Cell Phones & Accessories", and "Software". Below these links, there are filters for "Refine by", "International Shipping", "Eligible for Free Shipping", and "Brand". The main content area shows three product listings. The first listing is for a "Point and zoom in presentations with Myo Armband" with a "Shop now" button. The second listing is for a "Logitech Wireless Presenter R400" with a price of \$44.29 (discounted from \$49.99) and a Prime delivery date of Wednesday, May 27. The third listing is for a "Logitech Professional Presenter R800 with Green Laser Pointer" with a price of \$57.49 (discounted from \$79.99) and a Prime delivery date of Wednesday, May 27. The fourth listing is for a "Kensington Wireless Presenter with Laser Pointer" by Kensington.

amazon  
Try Prime

All presenter

Shop by Department  
Aidan's Amazon.com Today's Deals Gift Cards Sell Help

1-16 of 19,088 results for "presenter"

Show results for

Office Products >  
Office Presentation Remotes  
Office Presentation Pointers

Computers & Accessories >  
Tablet Accessories  
Computer Mice

Cell Phones & Accessories >  
Cell Phone Accessories

Software >  
Presentations


• See All 29 Departments

Refine by



International Shipping  
 Ship to Ireland



Eligible for Free Shipping  
Free Shipping by Amazon


Brand  
 Kensington  
 Logitech  
 Targus  
 Satechi  
 Infiniter  
 August

 **Point and zoom in presentations with Myo Armband**  
Shop now ▶

Related Searches: [logitech presenter](#), [mpow presenter](#), [wireless presenter](#).

 **Logitech Wireless Presenter R400**  
by Logitech  
**\$44.29** ~~\$49.99~~   
Get it by **Wednesday, May 27**


 **Logitech Professional Presenter R800 with Green Laser Pointer**  
by Logitech  
**\$57.49** ~~\$79.99~~   
Get it by **Wednesday, May 27**

 **Kensington Wireless Presenter with Laser Pointer**  
by Kensington







# The Amazon Scenario

Customer info: shopping cart, account, etc.

 **Shopping Cart** Already a customer?  
[Sign in](#)

[See more items like those in your Cart](#)

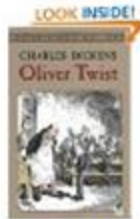
**subtotal = \$88.77**  
Make any changes below? [Update](#)

Shopping Cart Items--To Buy Now		Price:	Qty:
<small>Item added on May 22, 2009</small>	<a href="#">The Principles of Beautiful Web Design</a> - Jason Beard; <b>Paperback</b> Condition: New In Stock	<b>\$26.37</b> You Save: \$13.58 (34%)	<input type="text" value="1"/>
<a href="#">Save for later</a>	<a href="#">Delete</a>	 Eligible for FREE Super Saver Shipping	
	<a href="#">Add gift-wrap/note</a>  <a href="#">(Learn more)</a>		
<small>Item added on May 22, 2009</small>	<a href="#">Don't Make Me Think: A Common Sense Approach to Web Usability, 2nd Edition</a> - Steve Krug; <b>Paperback</b> Condition: New In Stock	<b>\$26.40</b> You Save: \$13.60 (34%)	<input type="text" value="1"/>
<a href="#">Save for later</a>	<a href="#">Delete</a>	 Eligible for FREE Super Saver Shipping	
	<a href="#">Add gift-wrap/note</a>  <a href="#">(Learn more)</a>		

# The Amazon Scenario

Recommendations, etc.:

## Customers Who Bought This Item Also Bought



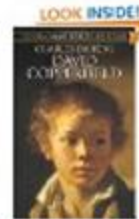
Oliver Twist (Dover Thrift Editions)

> Charles Dickens

★★★★☆ (213)

Paperback

\$3.50



David Copperfield (Dover Thrift Editions)

> Charles Dickens

★★★★☆ (196)

Paperback

\$5.00



JANE EYRE

> Charlotte Bronte

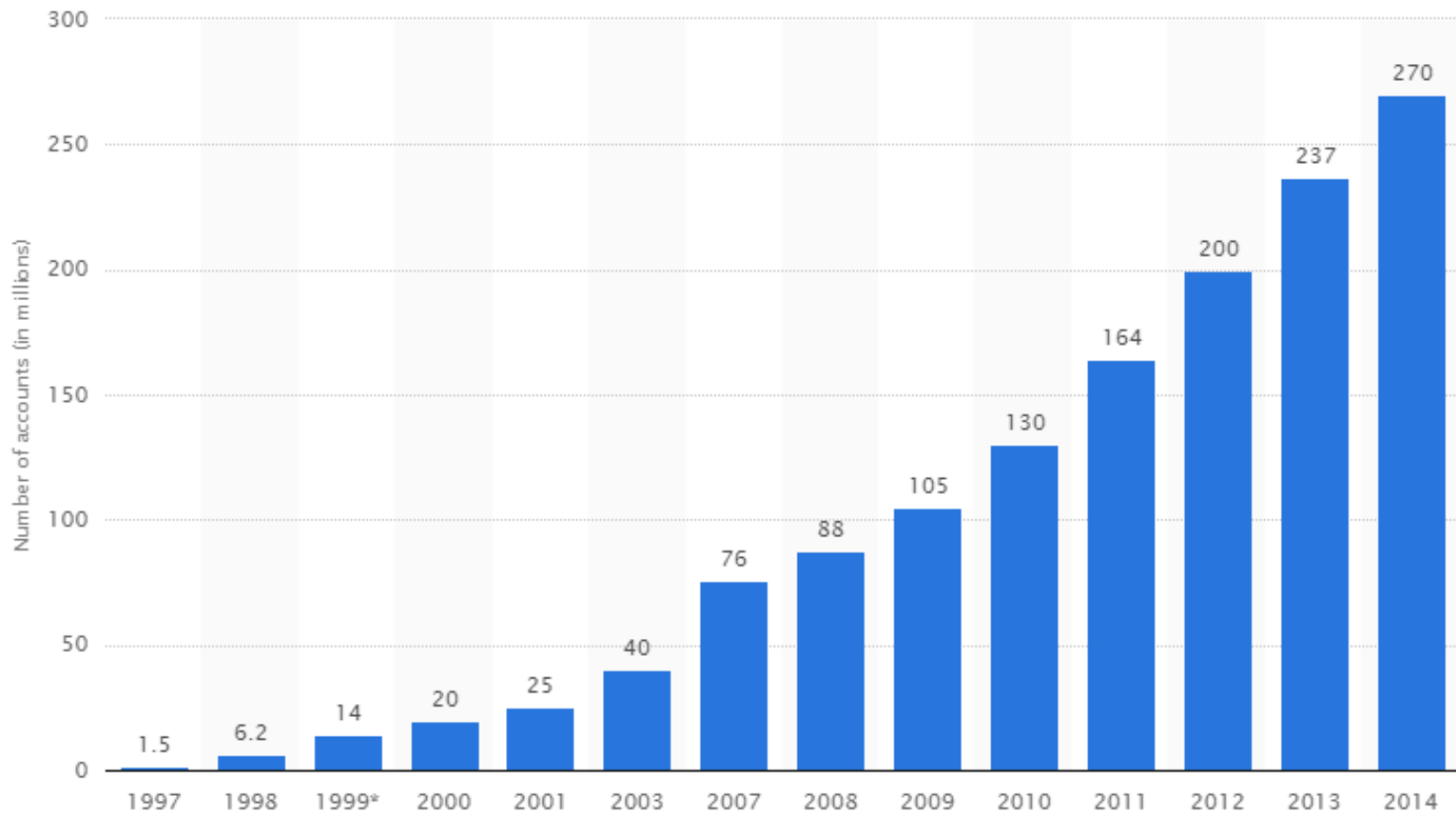
★★★★☆ (1,045)

Paperback

\$2.99

# The Amazon Scenario

- Amazon customers:

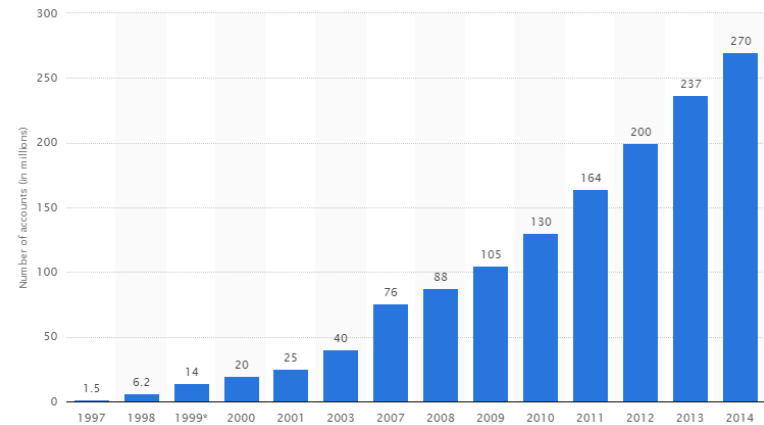


# The Amazon Scenario



# The Amazon Scenario

Databases struggling ...



But many Amazon services don't need:

- **SQL** (a simple map often enough)

or even:

- **transactions, strong consistency, etc.**

# Key–Value Store: Amazon Dynamo(DB)

## Dynamo: Amazon's Highly Available Key-value Store

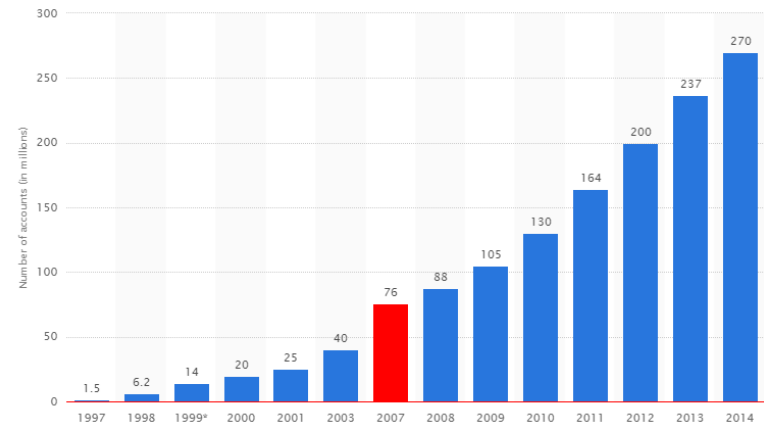
Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

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 failing, network routes are changing, or data centers are being



## Goals:

- Scalability (able to grow)
- High availability (reliable)
- Performance (fast)

Don't need full SQL, don't need full ACID

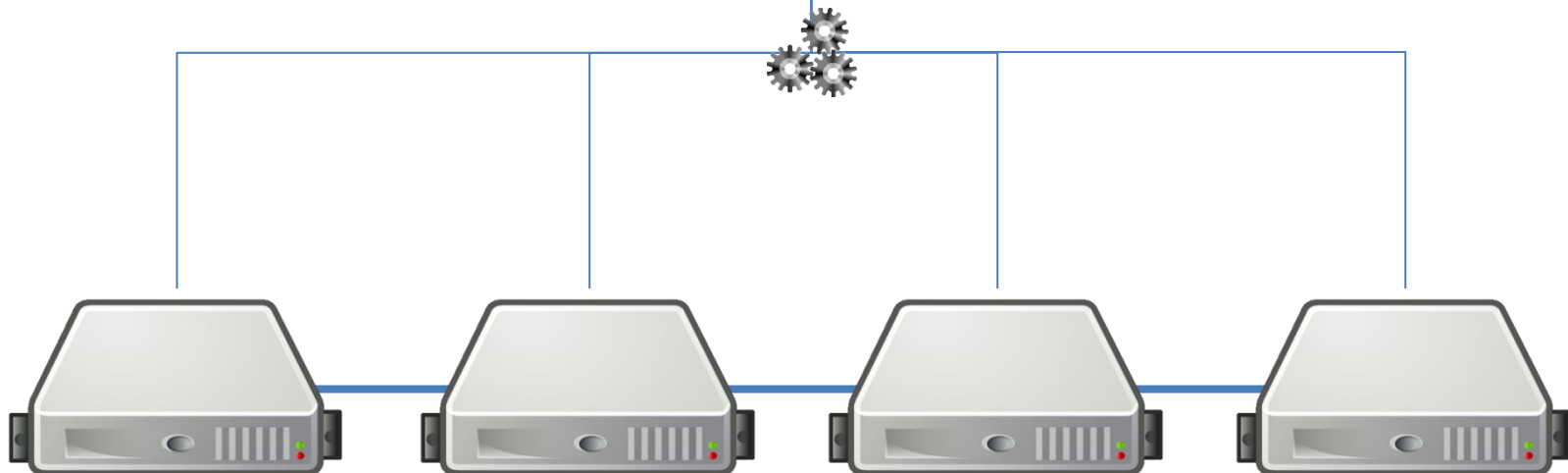
# Key–Value Store: Distribution

How might we distribute a key–value store over multiple machines?



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}
...	...

$$\text{mod}(\text{hash}(\textit{key}), m)$$



# Key–Value Store: Distribution

What happens if a machine leaves or joins afterwards?

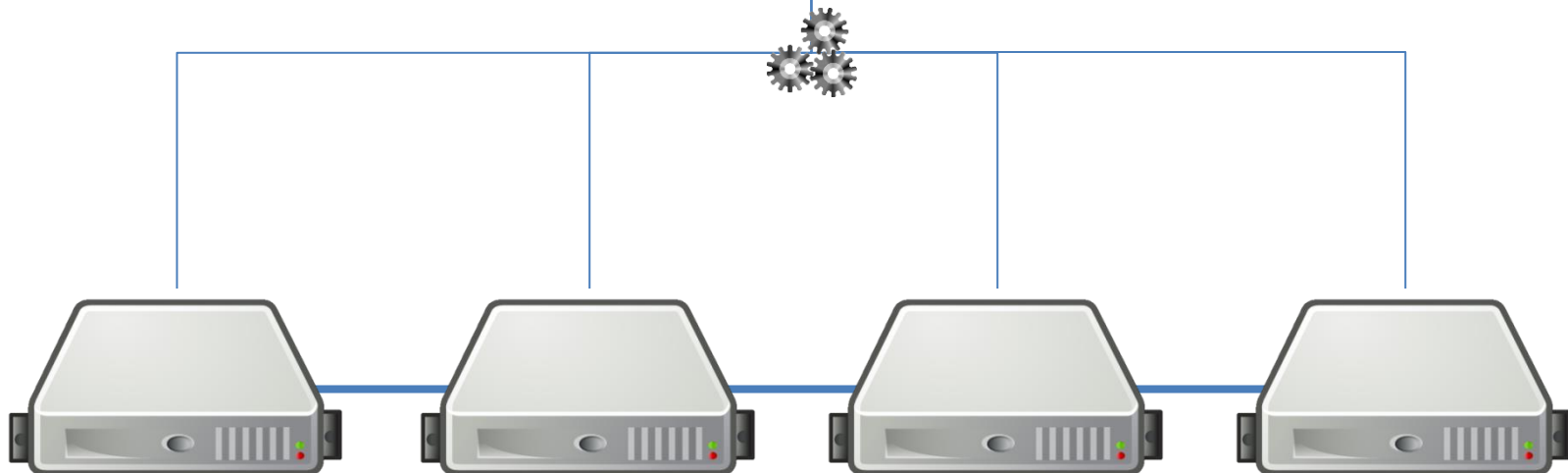


How can we avoid rehashing everything?



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}
...	...

$$\text{mod}(\text{hash}(\textit{key}), m)$$



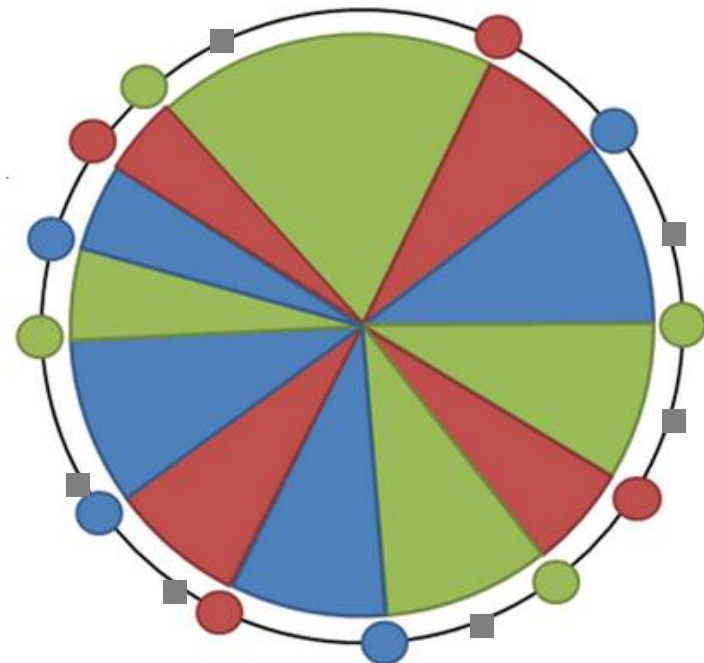


# Consistent Hashing

## Avoid re-hashing everything

- Hash using a ring
- Each machine picks  $n$  pseudo-random points on the ring
- Machine responsible for arc after its point
- If a machine leaves, its range moves to previous machine
- If machine joins, it picks new points
- Objects mapped to ring 😊

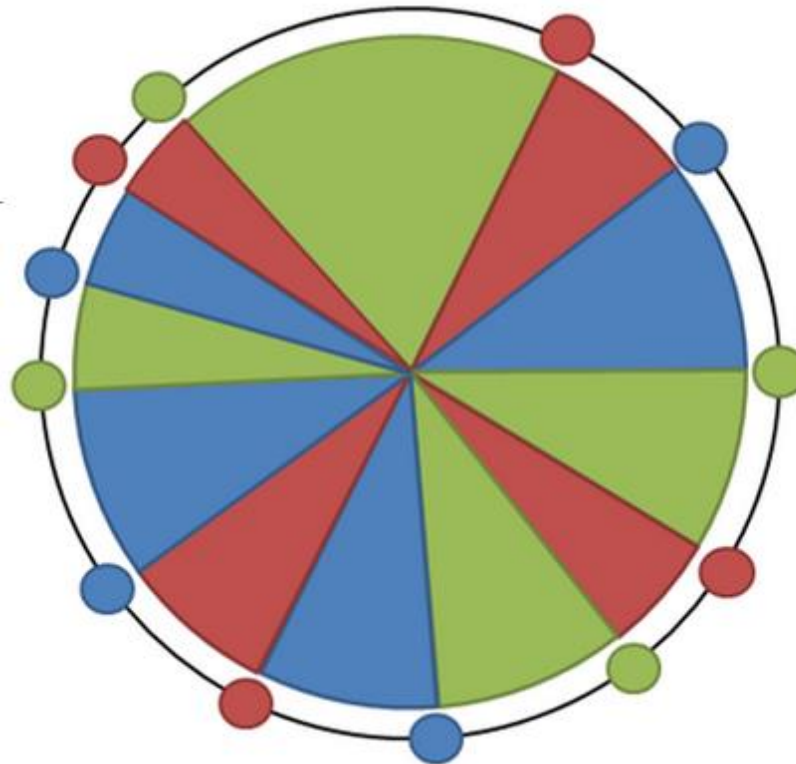
How many keys (on average) would need to be moved if a machine joins or leaves?



# Amazon Dynamo: Hashing

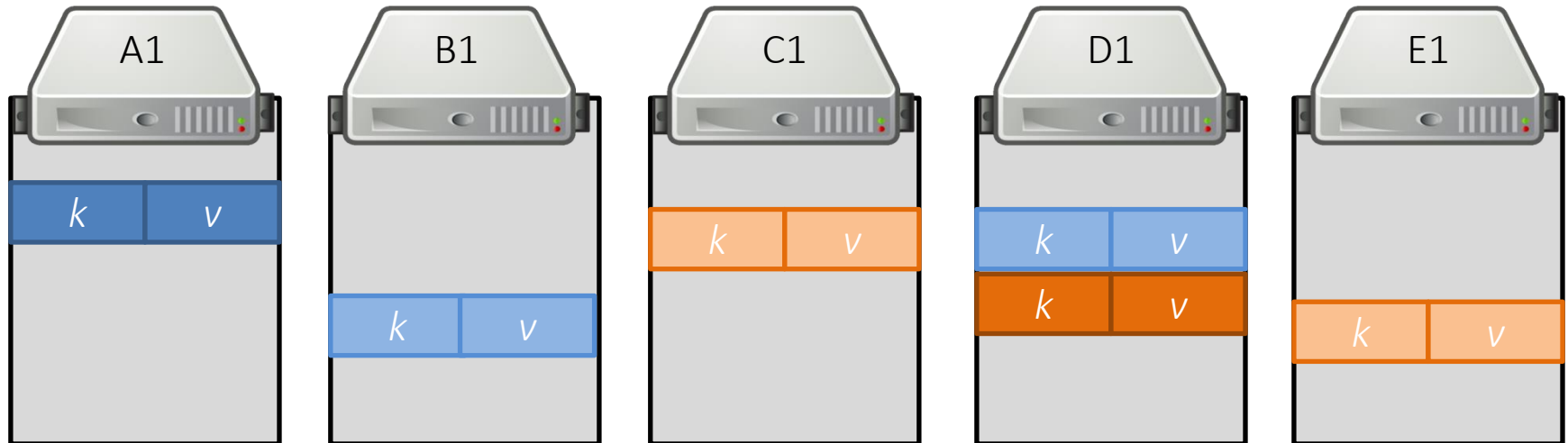


- Consistent Hashing (128-bit MD5)



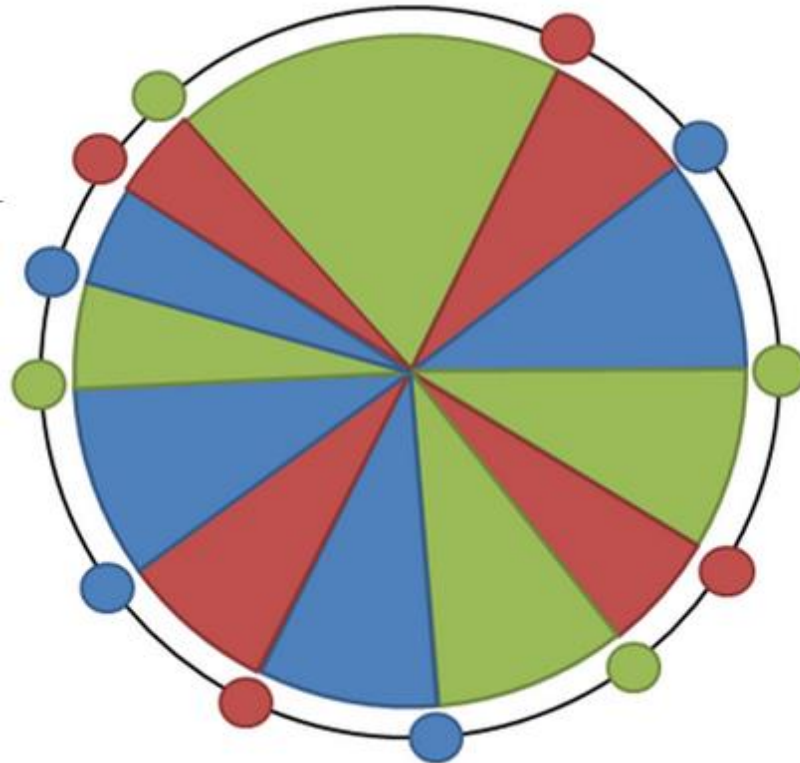
# Amazon Dynamo: Replication

- A set replication factor (e.g., 3)
- Commonly primary / secondary replicas
  - Primary replica elected from secondary replicas in the case of failure of primary



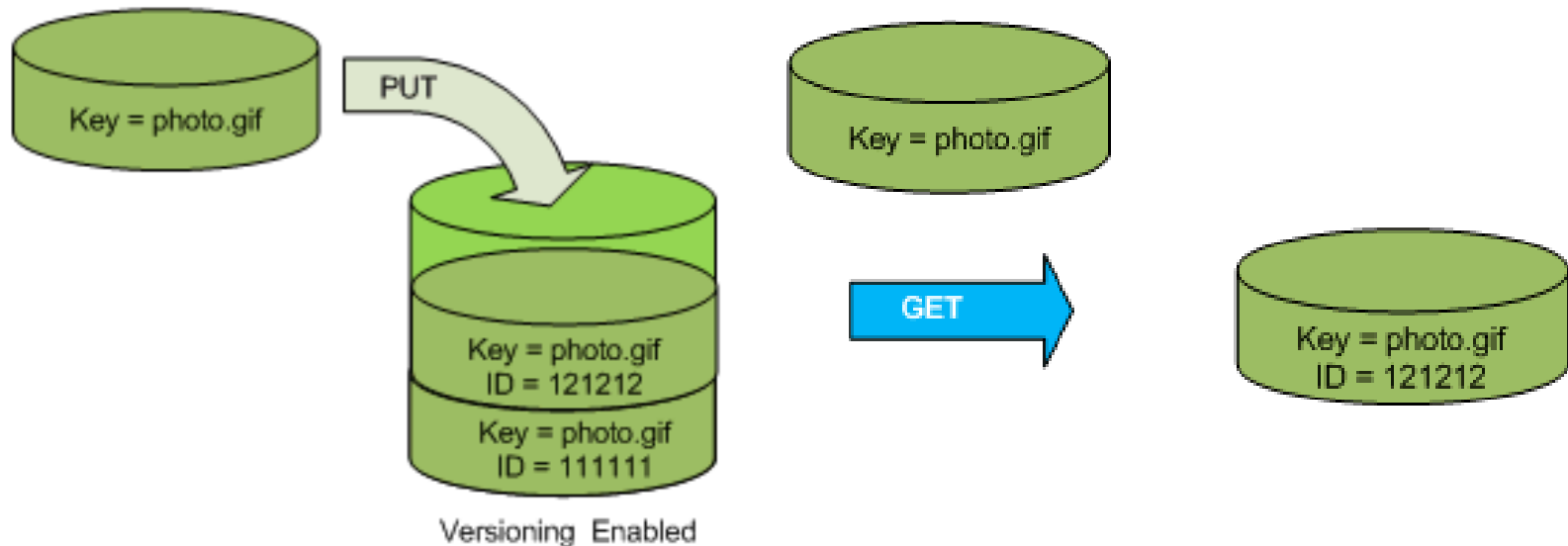
# Amazon Dynamo: Replication

- Replication factor of  $n$ ?
  - Easy: pick  $n$  next buckets (different machines!)



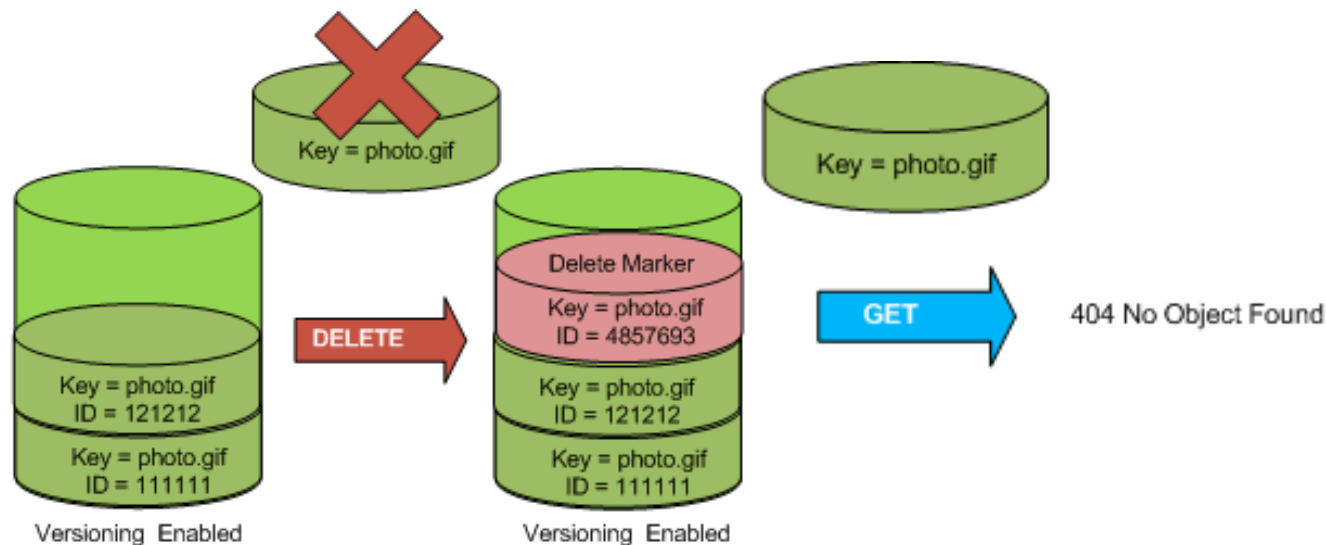
# Amazon Dynamo: Object Versioning

- Object Versioning (per bucket)
  - PUT doesn't overwrite: pushes version
  - **GET** returns most recent version



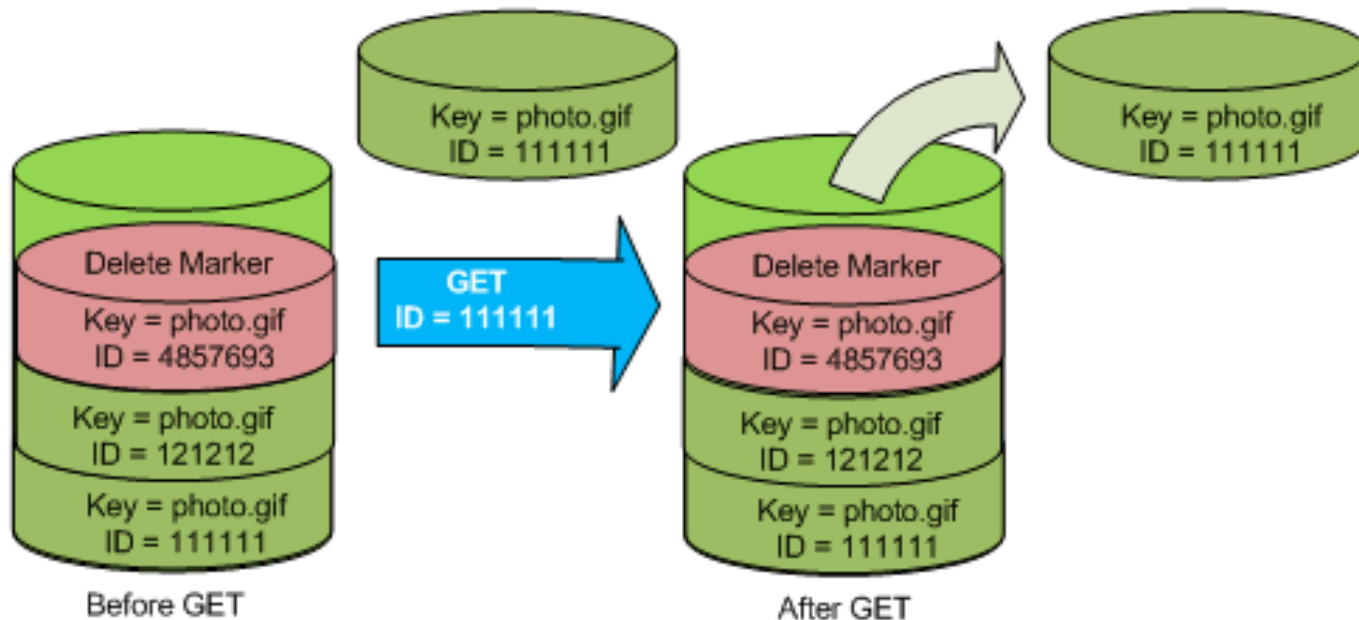
# Amazon Dynamo: Object Versioning

- Object Versioning (per bucket)
  - **DELETE** doesn't wipe
  - **GET** will return not found



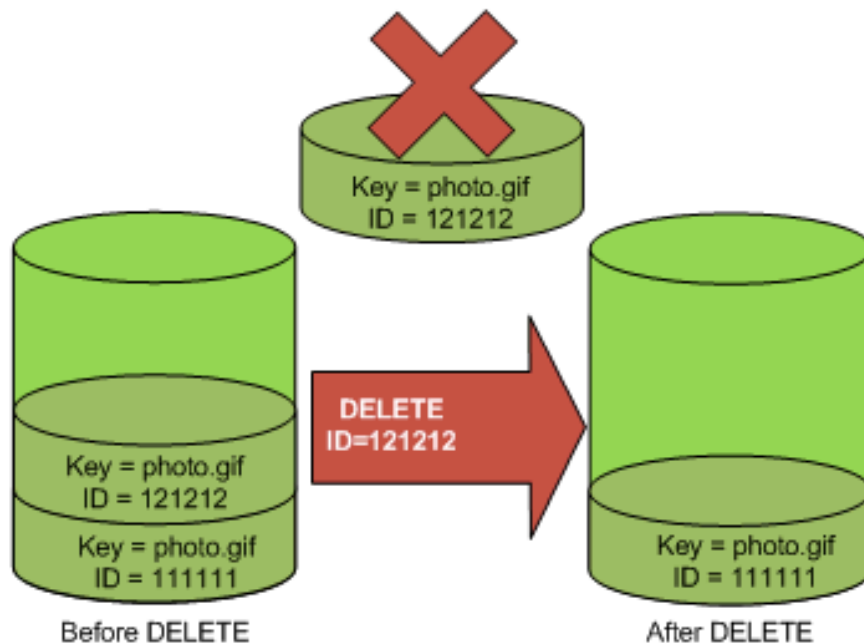
# Amazon Dynamo: Object Versioning

- Object Versioning (per bucket)
  - **GET** by version



# Amazon Dynamo: Object Versioning

- Object Versioning (per bucket)
  - **PERMANENT DELETE** by version ... wiped





# Amazon Dynamo: Model

- Named table with primary key and a value
- Primary key is hashed / unordered

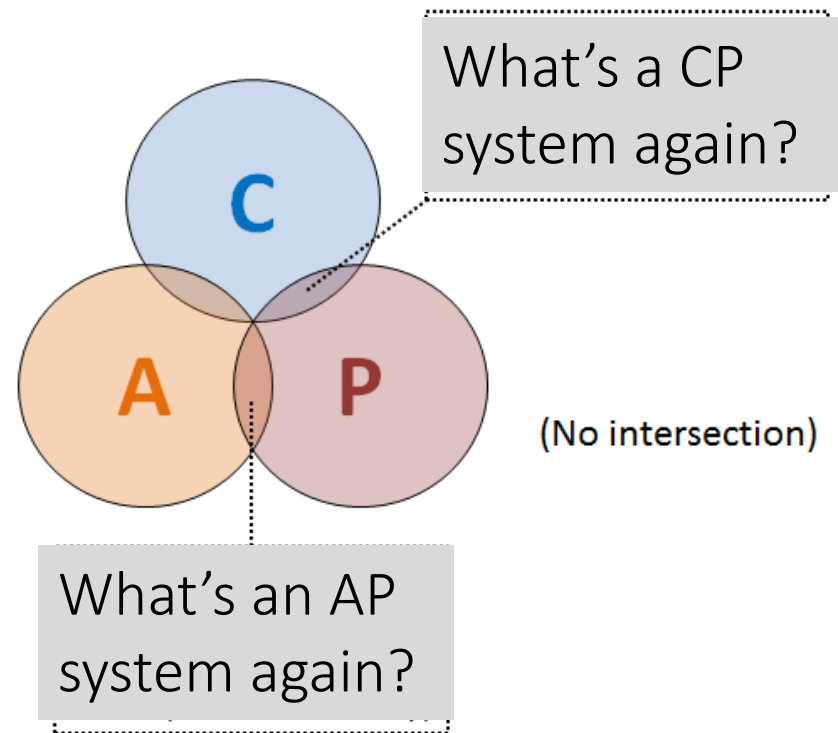
Countries	
Primary Key	Value
Afghanistan	capital:Kabul,continent:Asia,pop:31108077#2011
Albania	capital:Tirana,continent:Europe,pop:3011405#2013
...	...

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

# Amazon Dynamo: CAP

Two options for each table:

- **AP**: Eventual consistency,  
High availability
- **CP**: Strong consistency,  
Lower availability



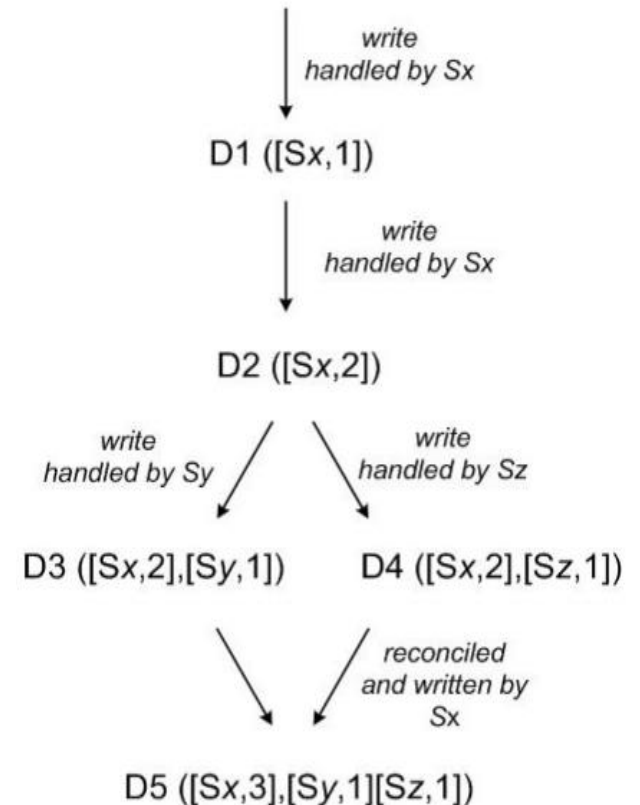
# Amazon Dynamo: Consistency

- **Gossiping**
  - Keep-alive messages sent between nodes with state
  - Dynamo largely decentralised (no master node)
- **Quorums:**
  - Multiple nodes responsible for a read ( $R$ ) or write ( $W$ )
  - At least  $R$  or  $W$  nodes acknowledge for success
  - Higher  $R$  or  $W$  = Higher consistency, lower availability
- **Hinted Handoff**
  - For transient failures
  - A node “covers” for another node while it is down

# Amazon Dynamo: Consistency

- **Vector Clock:**


- A list of pairs indicating a node and operation count
- Used to track branches of revisions




# Amazon Dynamo: Consistency

- Two versions of one shopping cart:

## Shopping Cart

	Price	Quantity
 <b>WD My Passport Ultra 2TB Portable External USB 3.0 Hard Drive with Auto Backup - Red</b> by Western Digital In Stock Eligible for FREE Shipping <input type="checkbox"/> This is a gift <a href="#">Learn more</a> <a href="#">Delete</a> <a href="#">Save for later</a>	<b>\$90.99</b> You save: \$49.00 (35%)	1
 <b>Logitech Wireless Presenter R400</b> by Logitech In Stock Eligible for FREE Shipping <input type="checkbox"/> This is a gift <a href="#">Learn more</a> <a href="#">Delete</a> <a href="#">Save for later</a>	<b>\$44.29</b> You save: \$5.70 (11%)	1
<b>Subtotal (2 items): \$135.28</b>		
Total savings: <b>\$54.70</b>		

## Shopping Cart

	Price	Quantity
 <b>AKG Perception P120 Professional Studio Microphone, Silver</b> by AKG Pro Audio Only 2 left in stock. Shipped from: Sam Ash Gift options not available. <a href="#">Learn more</a> <a href="#">Delete</a> <a href="#">Save for later</a>	<b>\$99.00</b> You save: \$30.00 (23%)	1
 <b>Logitech Wireless Presenter R400</b> by Logitech In Stock Eligible for FREE Shipping <input type="checkbox"/> This is a gift <a href="#">Learn more</a> <a href="#">Delete</a> <a href="#">Save for later</a>	<b>\$44.29</b> You save: \$5.70 (11%)	1
<b>Subtotal (2 items): \$143.29</b>		
Total savings: <b>\$35.70</b>		


How best to merge multiple conflicting versions of a value  
(known as reconciliation)?




Application knows best

(... and must support multiple versions being returned)

# Amazon Dynamo: Consistency



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:{103,10430,201,408}
...	...



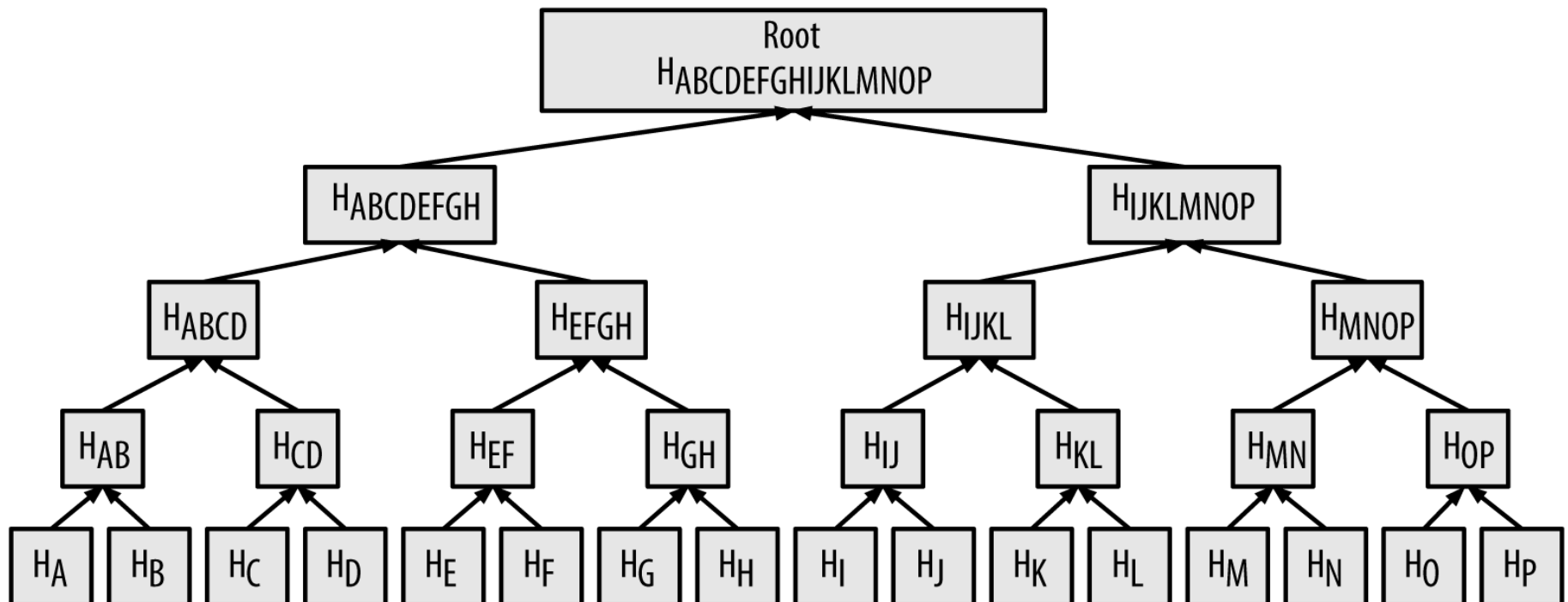
Key	Value
country:Afghanistan	capital@city:Kabul,continent:Asia,pop:31108077#2011
country:Albania	capital@city:Tirana,continent:Europe,pop:3011405#2013
...	...
city:Kabul	country:Afghanistan,pop:3476000#2013
city:Tirana	country:Albania,pop:3011405#2013
...	...
user:10239	basedIn@city:Tirana,post:{103,10430,201}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

How can we efficiently verify that two copies of a block of data are the same (and find where the differences are)?

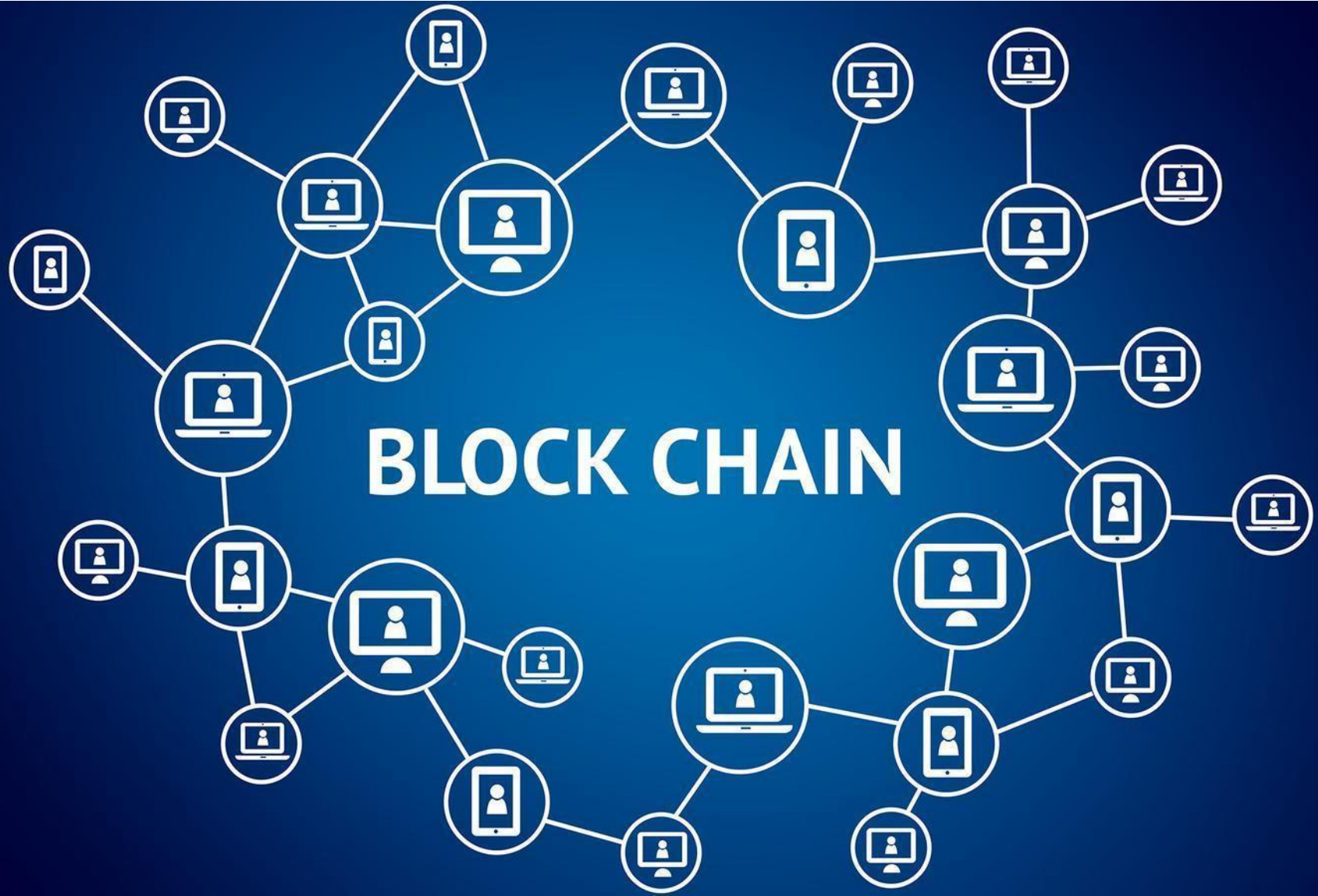


# Amazon Dynamo: Merkle Trees

- **Merkle tree:** A hash tree
  - Leaf node compute hashes from data
  - Non-leaf nodes have hashes of their children
  - Find differences between two trees level-by-level



Merkle Trees also used in ...





Read More ...



## **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

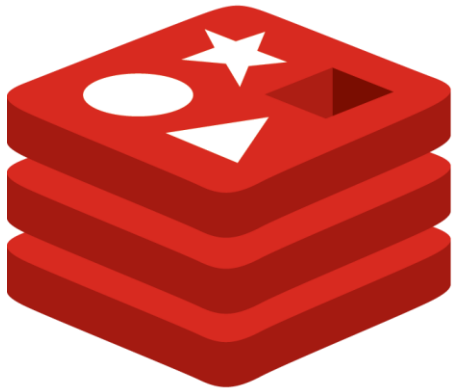
One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. 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 failing, network routes are flapping, or data centers are being

# OTHER KEY-VALUE STORES

# Other Key–Value Stores



# Other Key–Value Stores



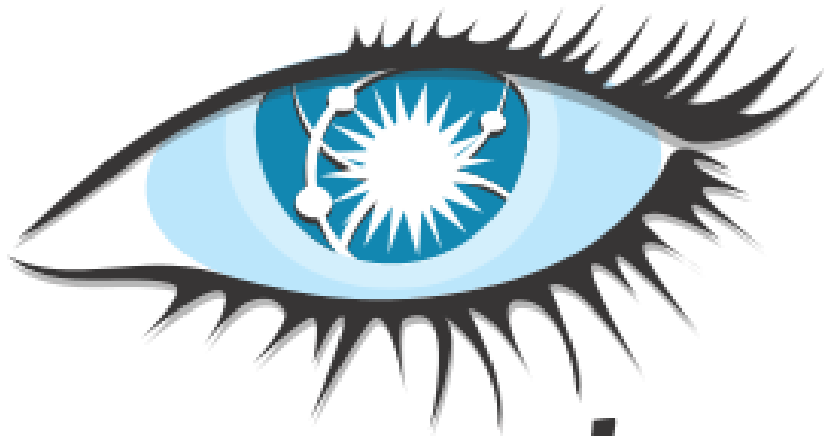
redis



StackExchange 



# Other Key-Value Stores



***cassandra***



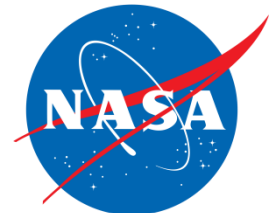
**Instagram**  
Fast beautiful photo sharing

**accenture**  
High performance. Delivered.

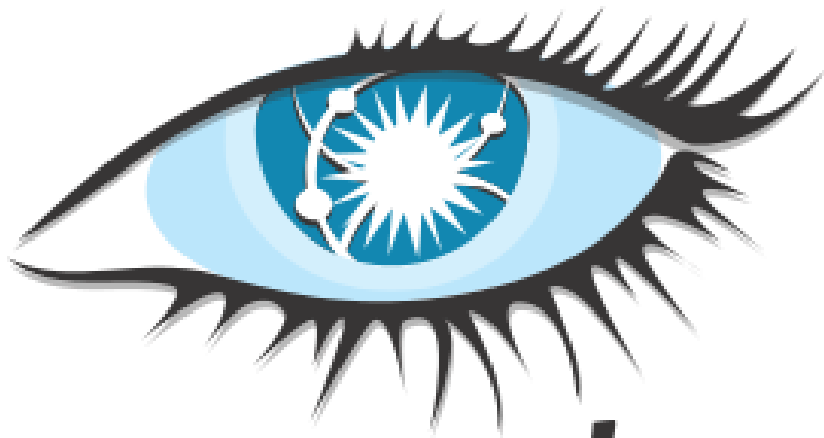
**Answers.com**



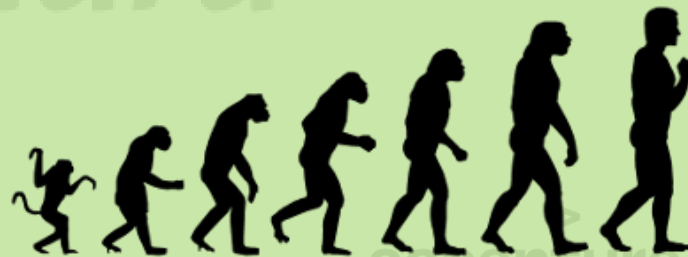
**Disney**



# Other Key-Value Stores



cassandra



Instagram  
Fast beautiful

accenture  
High performance. Delivered.

Answers.com

Evolved into a  
tabular store ...



DISNEY



TABULAR / COLUMN FAMILY

# Key–Value = a Distributed Map

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

# Tabular = Multi-dimensional Maps

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



# Bigtable: The Original Whitepaper

MapReduce  
authors

## 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 service). 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 serialise 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 used for ...



Google Analytics



+Amit Web Images Videos Maps News Shopping Gmail More

Amit Singhal 2 Share

Google chikoo

Search 50 personal results and 419,000 other results (0.61 seconds)

Manilkara zapota - Wikipedia, the free encyclopedia  
en.wikipedia.org/wiki/Manilkara\_zapota  
Sapodilla is known as **chikoo** ("चिकू," or chiku, "चोफ़,") in India and Pakistan and sapota in some parts of India (Tamil Nadu, Kerala, Karnataka, Andhra ...  
Description - Other names - See also - References  
You've visited this page 3 times. Last visit 12/4/11

Images for chikoo - Report images

You	Shilpa Singhal	You	Shilpa Singhal

Chikoo - a simple file organizer for the Mac  
codingturtle.com/chikoo/  
Chikoo, a simple file organizer for the Mac. Download (30-day trial). Version 0.9.1 ...



orkut by Google™

# Bigtable: Data Model

“a sparse, distributed, persistent, multi-dimensional, sorted map.”

- **sparse**: not all values form a dense square
- **distributed**: lots of machines
- **persistent**: disk storage (GFS)
- **multi-dimensional**: values with columns
- **sorted**: sorting lexicographically by row key
- **map**: look up a key, get a value



# Bigtable: Sorted Keys

S  
O  
R  
T  
E  
D

Primary Key	capital		pop-value		pop-year	
Asia:Afghanistan	t <sub>1</sub>	Kabul	t <sub>1</sub>	31143292	t <sub>1</sub>	2009
			t <sub>2</sub>	31120978		
			t <sub>4</sub>	31108077	t <sub>4</sub>	2011
Asia:Azerbaijan	...	...	...	...	...	...
...	...	...	...	...	...	...
Europe:Albania	t <sub>1</sub>	Tirana	t <sub>1</sub>	2912380	t <sub>1</sub>	2010
			t <sub>3</sub>	3011405	t <sub>3</sub>	2013
Europe:Andorra	...	...	...	...	...	...
...	...	...	...	...	...	...

Benefits of sorted vs. hashed keys?



Range queries and ...

# Bigtable: Tablets

	Primary Key	capital		pop-value		pop-year	
ASIA	Asia:Afghanistan	t <sub>1</sub>	Kabul	t <sub>1</sub>	31143292	t <sub>1</sub>	2009
				t <sub>2</sub>	31120978		
				t <sub>4</sub>	31108077	t <sub>4</sub>	2011
	Asia:Azerbaijan	...	...	...	...	...	...
	...	...	...	...	...	...	...
EUROPE	Europe:Albania	t <sub>1</sub>	Tirana	t <sub>1</sub>	2912380	t <sub>1</sub>	2010
				t <sub>3</sub>	3011405	t <sub>3</sub>	2013
	Europe:Andorra	...	...	...	...	...	...
	...	...	...	...	...	...	...

Benefits of sorted vs. hashed keys?



Range queries and ...

... locality of processing

# A real-world example of locality/sorting



Primary Key	language		title		links	
com.imdb	t <sub>1</sub>	en	t <sub>1</sub>	IMDb Home	t <sub>1</sub>	...
			t <sub>2</sub>	IMDB - Movies		
			t <sub>4</sub>	IMDb	t <sub>4</sub>	...
com.imdb/title/tt2724064/	t <sub>1</sub>	en	t <sub>2</sub>	Sharknado	t <sub>2</sub>	...
com.imdb/title/tt3062074/	t <sub>1</sub>	en	t <sub>2</sub>	Sharknado II	t <sub>2</sub>	...
...	...	...	...	...	...	...
org.wikipedia	t <sub>1</sub>	multi	t <sub>1</sub>	Wikipedia	t <sub>1</sub>	...
			t <sub>3</sub>	Wikipedia Home	t <sub>3</sub>	...
org.wikipedia.ace	t <sub>1</sub>	ace	t <sub>1</sub>	Wikipèdia bahsa Acèh	...	...
...	...	...	...	...	...	...

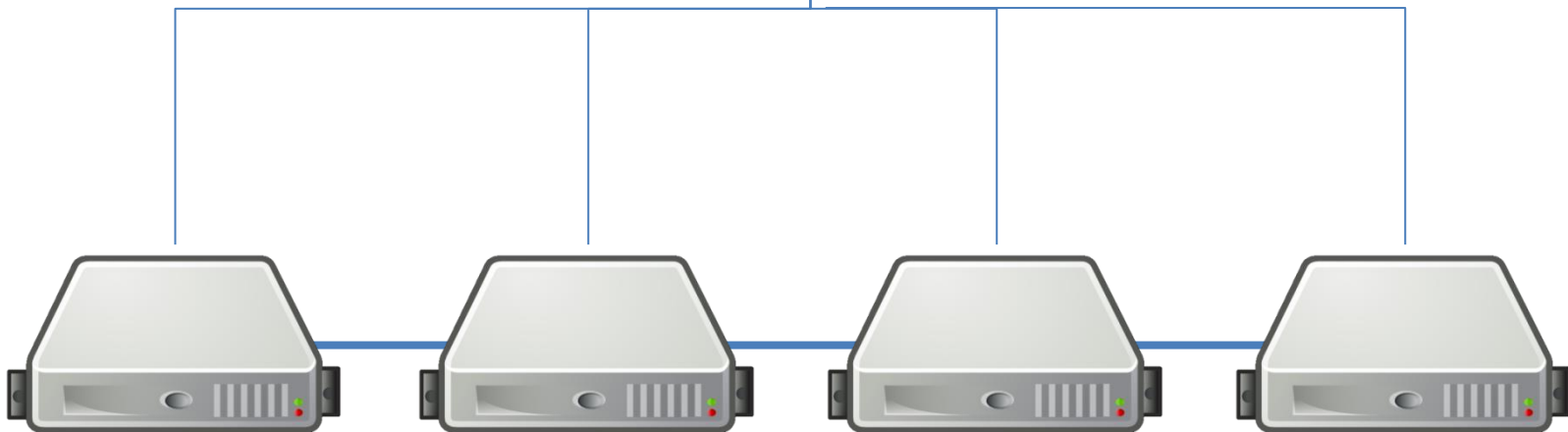


# Bigtable: Distribution

Asia:Afghanistan	t <sub>1</sub>	Kabul	t <sub>1</sub>	31143292	t <sub>1</sub>	2009
			t <sub>2</sub>	31120978		
			t <sub>4</sub>	31108077	t <sub>4</sub>	2011
Asia:Azerbaijan	...	...	...	...	...	...
...	...	...	...	...	...	...

Europe:Albania	t <sub>1</sub>	Tirana	t <sub>1</sub>	2912380	t <sub>1</sub>	2010
			t <sub>3</sub>	3011405	t <sub>3</sub>	2013
Europe:Andorra	...	...	...	...	...	...
...	...	...	...	...	...	...

Split by tablet



Horizontal range partitioning



# Bigtable: Column Families

Primary Key	pol:capital		demo:pop-value		demo:pop-year	
Asia:Afghanistan	t <sub>1</sub>	Kabul	t <sub>1</sub>	31143292	t <sub>1</sub>	2009
			t <sub>2</sub>	31120978		
			t <sub>4</sub>	31108077	t <sub>4</sub>	2011
Asia:Azerbaijan	...	...	...	...	...	...
...	...	...	...	...	...	...
Europe:Albania	t <sub>1</sub>	Tirana	t <sub>1</sub>	2912380	t <sub>1</sub>	2010
			t <sub>3</sub>	3011405	t <sub>3</sub>	2013
Europe:Andorra	...	...	...	...	...	...
...	...	...	...	...	...	...


- Group logically similar columns together
  - Accessed efficiently together
  - Access-control and storage: column family level
  - If of same type, can be compressed

# Bigtable: Versioning

- Similar to Apache Dynamo
  - Cell-level
  - 64-bit integer time stamps
  - Inserts push down current version
  - Lazy deletions / periodic garbage collection
  - Two options:
    - keep last  $n$  versions
    - keep versions newer than  $t$  time

# Bigtable: SSTable Map Implementation

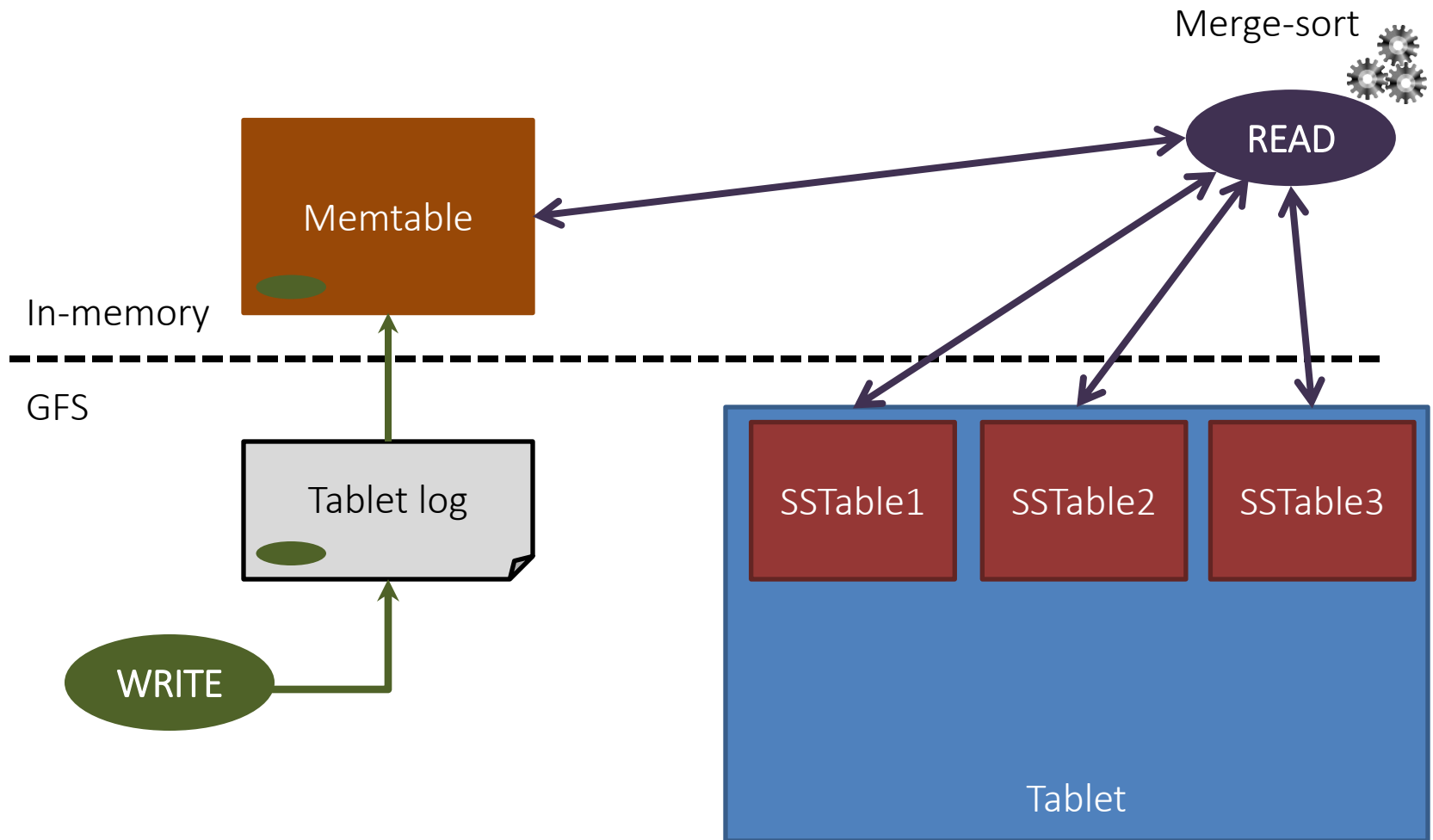
- 64k blocks (default) with index in footer (GFS)
- Index loaded into memory, allows for seeks
- Can be split or merged, as needed

Writes? 

	Primary Key	pol:capital	demo:pop-value		demo:pop-year		
0	Asia:Afghanistan	t <sub>1</sub>	Kabul	t <sub>1</sub>	31143292	t <sub>1</sub>	2009
				t <sub>2</sub>	31120978		
				t <sub>4</sub>	31108077	t <sub>4</sub>	2011
	Asia:Azerbaijan	...	...	...	...	...	...
...	...	...	...	...	...	...	
65536	Asia:Japan	...	...	...	...	...	...
	Asia:Jordan	...	...	...	...	...	...
	...	...	...	...	...	...	...
Index:	Block 0 / Offset 0 / Asia:Afghanistan						
	Block 1 / Offset 65536 / Asia: Japan						

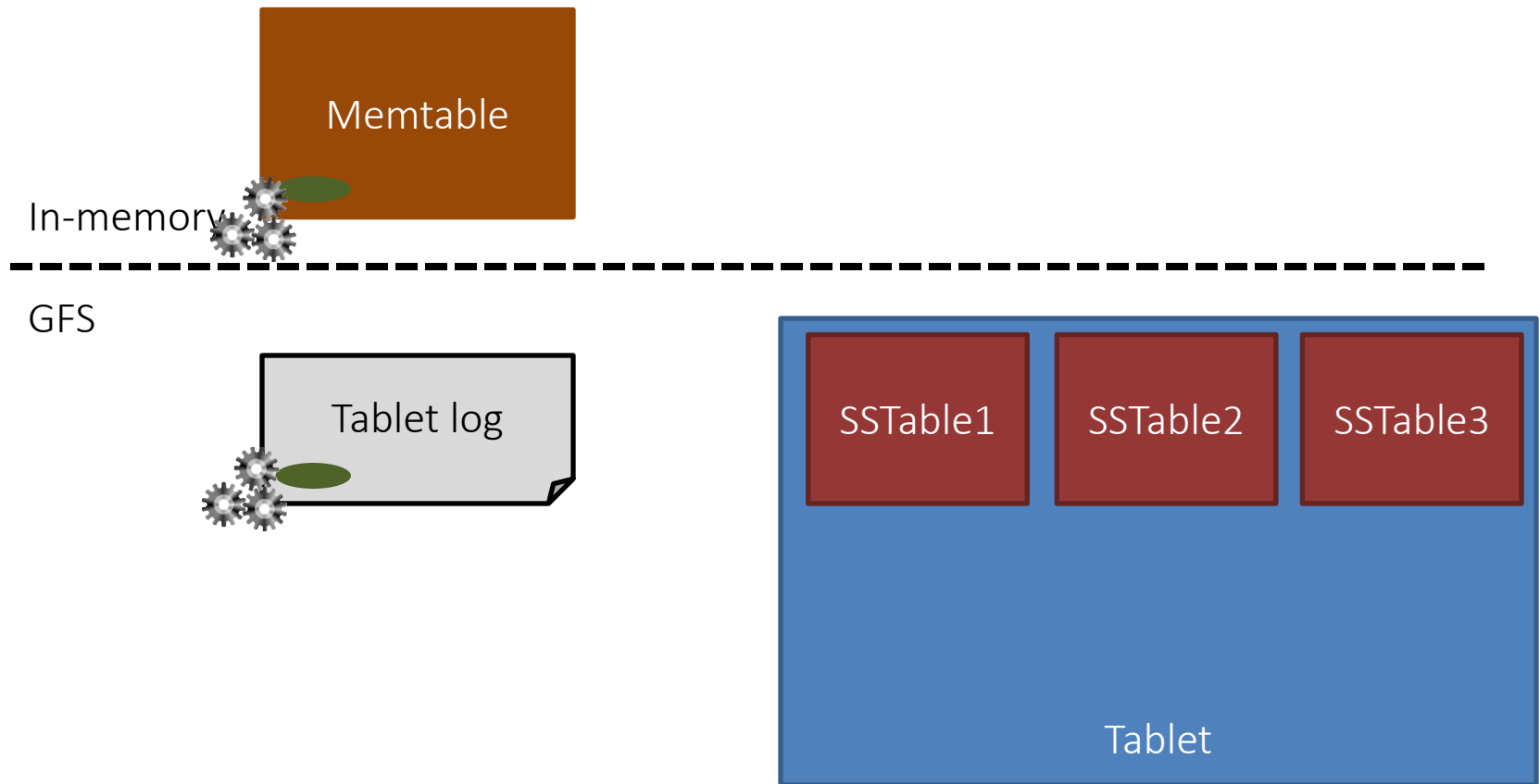
# Bigtable: Buffered/Batched Writes

What's the danger?



# Bigtable: Redo Log

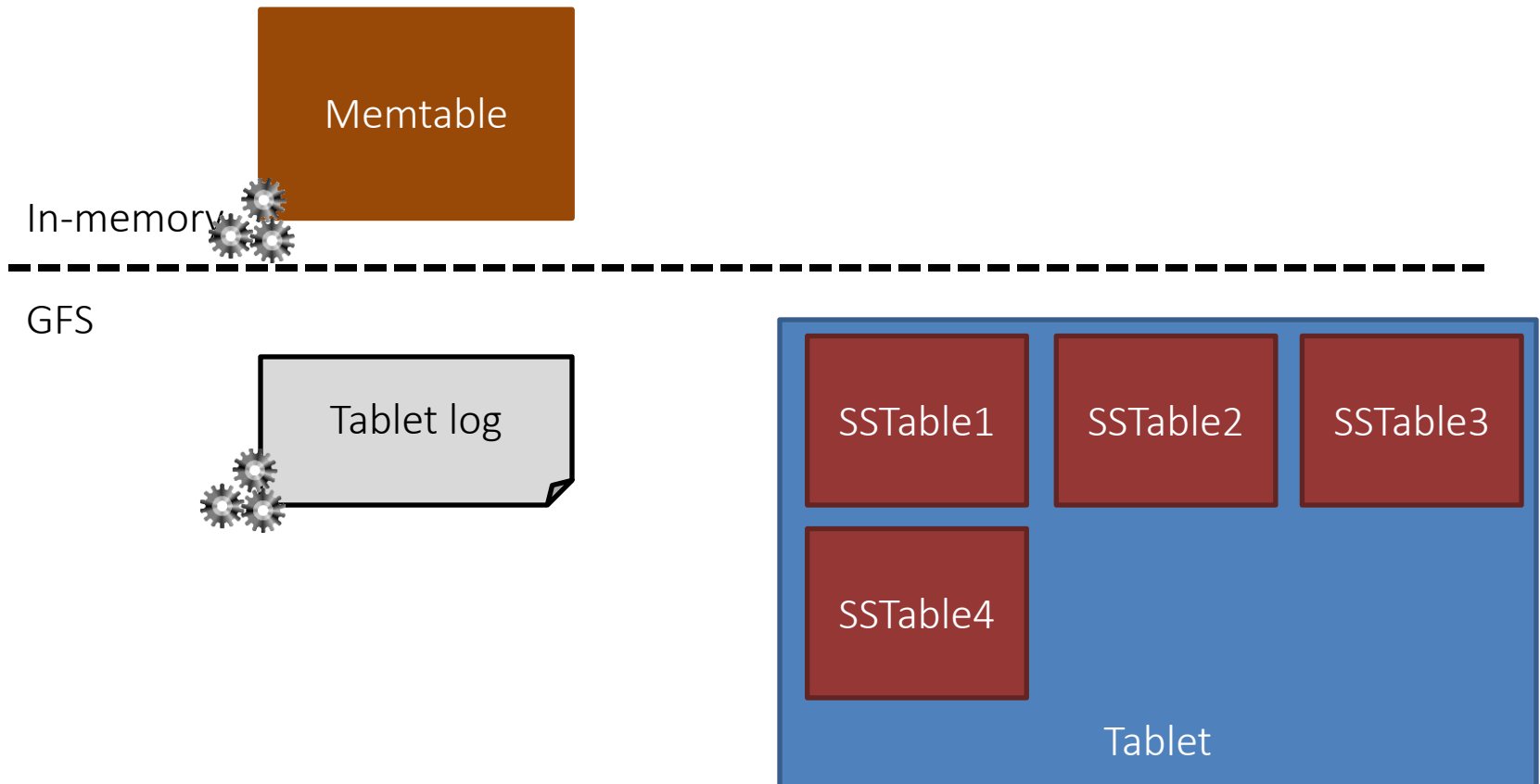
- If machine fails, Memtable redone from log



# Bigtable: Minor Compaction

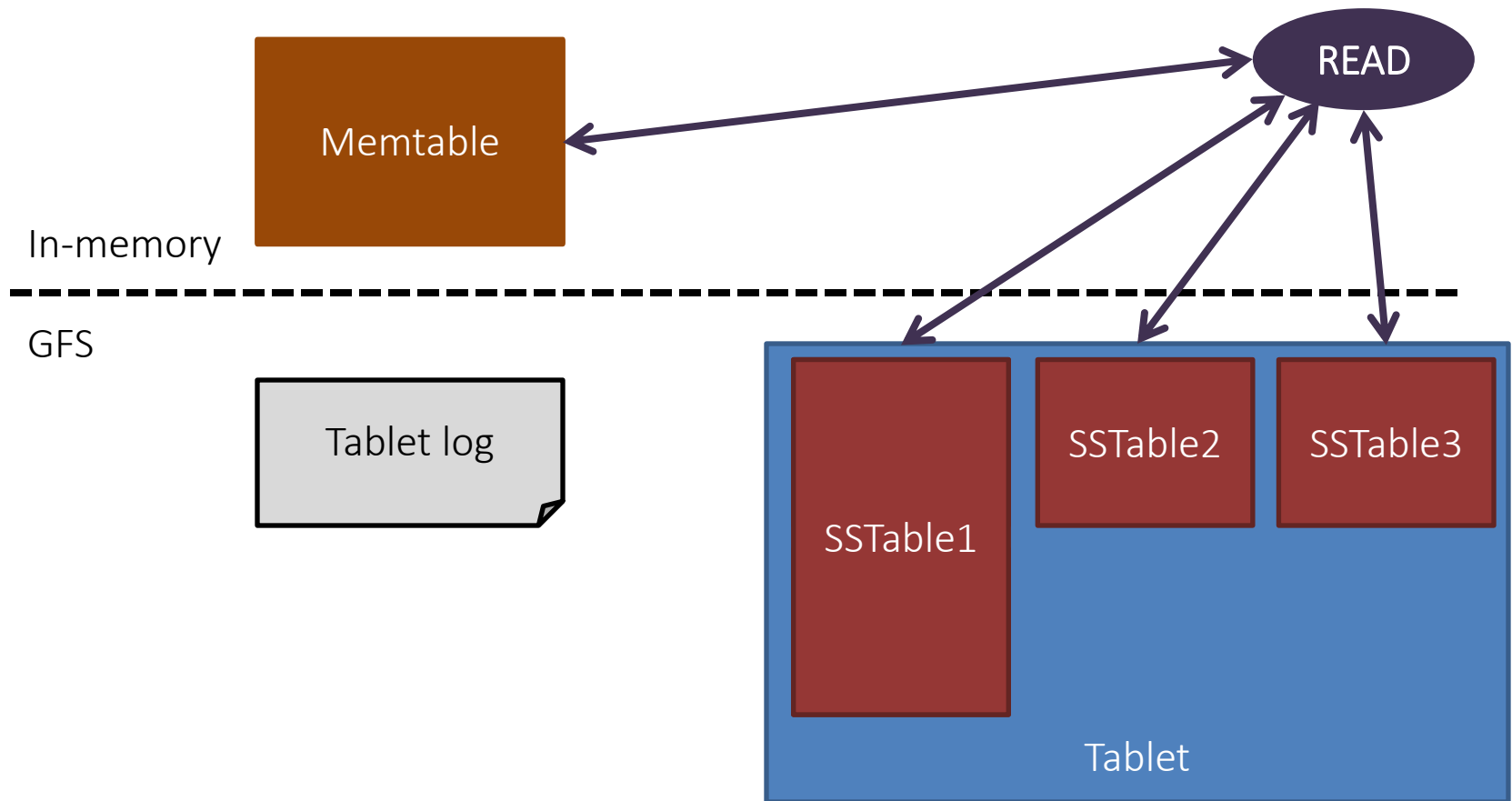
- When full, write Memtable as SSTable

Problem with performance?



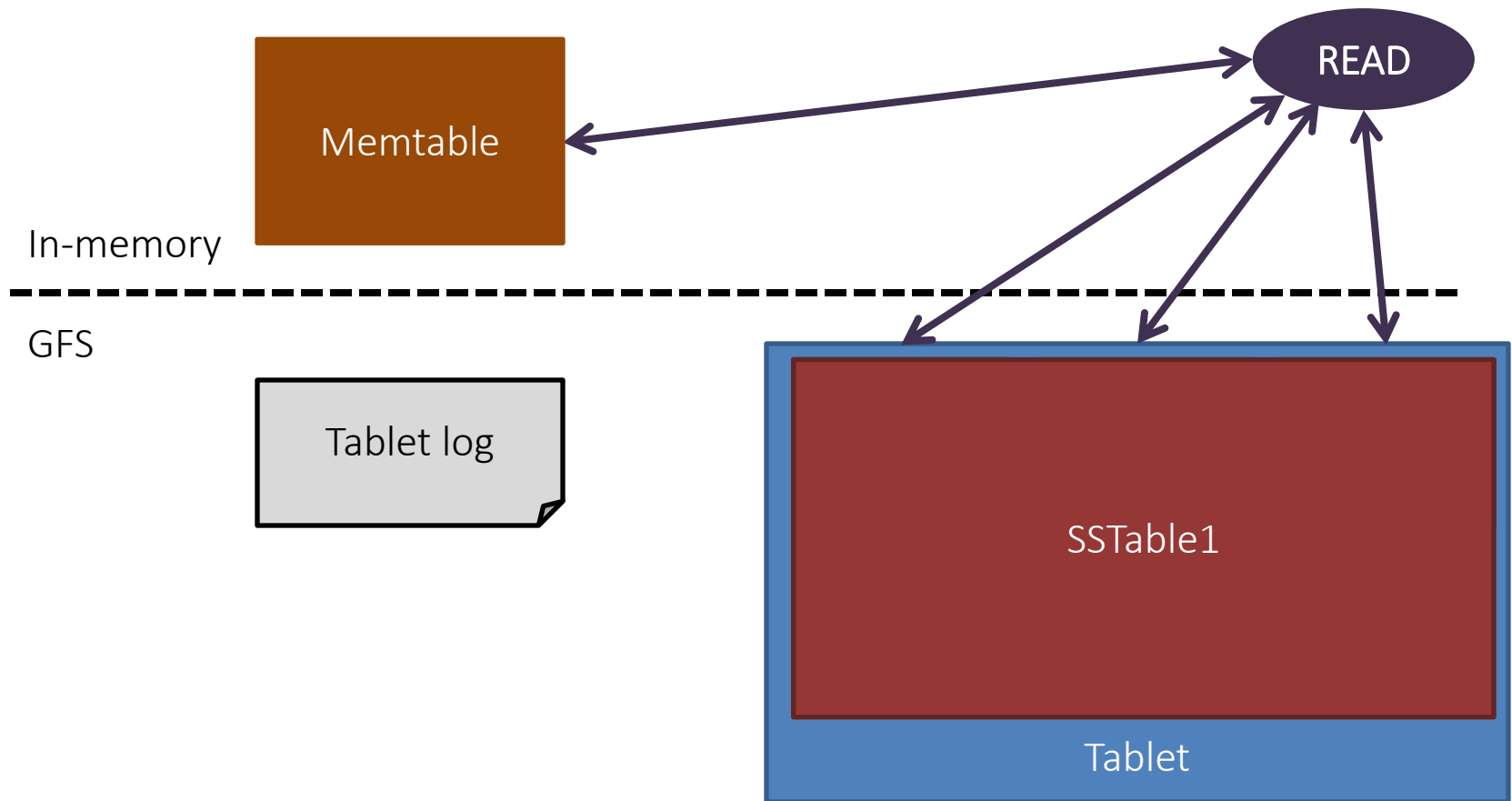
# Bigtable: Merge Compaction

- Merge **some of** the SSTables (and the Memtable)



# Bigtable: Major Compaction

- Merge **all** SSTables (and the Memtable)
- Makes reads more efficient!





# Bigtable: A Bunch of Other Things

- **Hierarchy and locks**: how to find and lock tablets
- **Locality groups**: Group multiple column families together; assigned a separate SSTable
- **Select storage**: SSTables can be persistent or in-memory
- **Compression**: Applied on SSTable blocks; custom compression can be chosen
- **Caches**: SSTable-level and block-level
- **Bloom filters**: Find negatives cheaply ...

Read More ...



## **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**

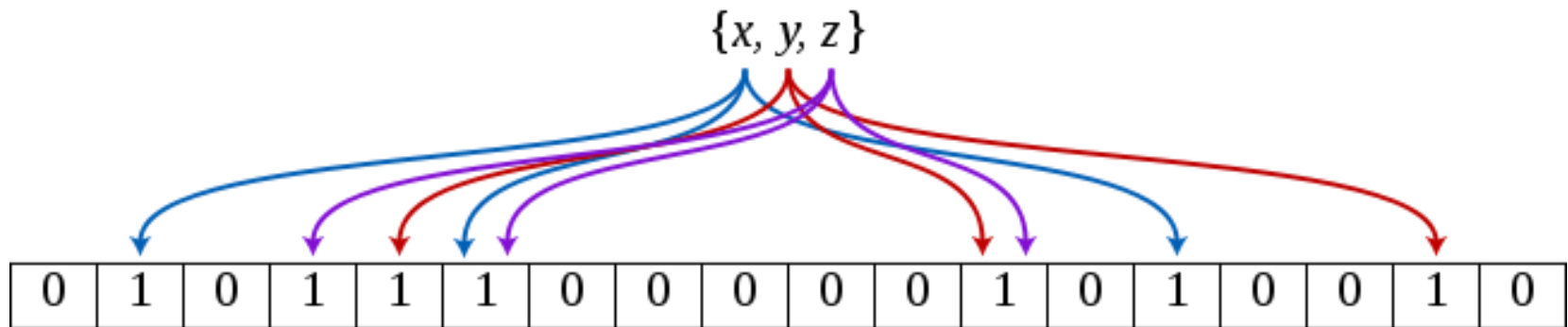
Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place 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

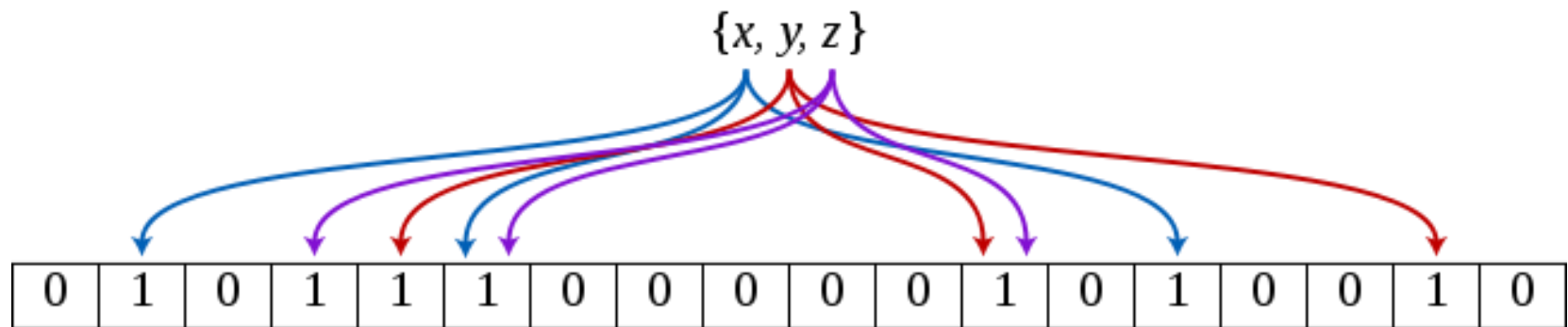
# Aside: Bloom Filter

- Create a bit array of length  $m$  (init to 0's)
- Create  $k$  hash functions that map an object to an index of  $m$
- **Index  $x$** : set  $m[\text{hash}_1(x)], \dots, m[\text{hash}_k(x)]$  to 1



# Aside: Bloom Filter

- Create a bit array of length  $m$  (init to 0's)
- Create  $k$  hash functions that map an object to an index of  $m$
- **Index  $x$ :** set  $m[\text{hash}_1(x)], \dots, m[\text{hash}_k(x)]$  to 1



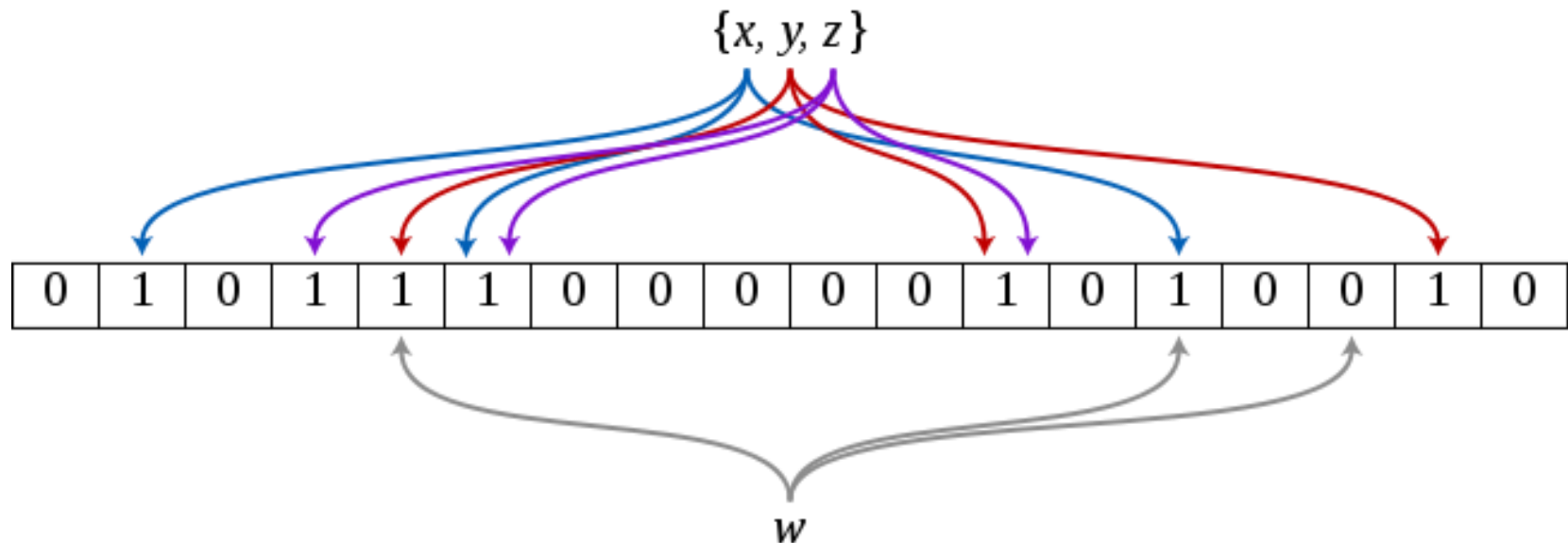
- **Query  $w$ :**
  - any  $m[\text{hash}_1(w)], \dots, m[\text{hash}_k(w)]$  set to 0  $\Rightarrow$
  - all  $m[\text{hash}_1(w)], \dots, m[\text{hash}_k(w)]$  set to 1  $\Rightarrow$



# Aside: Bloom Filter

Reject “empty” queries using very little memory!

- Create a bit array of length  $m$  (init to 0's)
- Create  $k$  hash functions that map an object to an index of  $m$
- **Index**  $x$ : set  $m[\text{hash}_1(x)], \dots, m[\text{hash}_k(x)]$  to 1

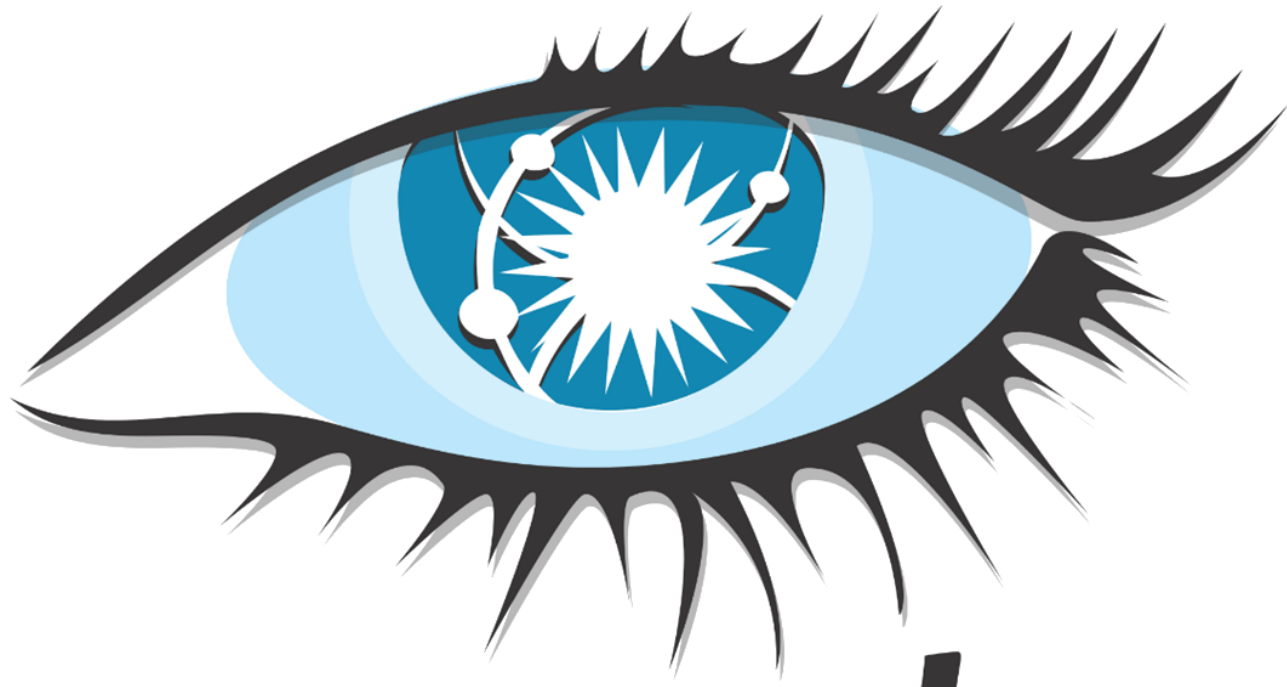


- **Query**  $w$ :
  - any  $m[\text{hash}_1(w)], \dots, m[\text{hash}_k(w)]$  set to 0  $\Rightarrow$  not indexed
  - all  $m[\text{hash}_1(w)], \dots, m[\text{hash}_k(w)]$  set to 1  $\Rightarrow$  **might** be indexed

# Tabular Store: Apache HBase

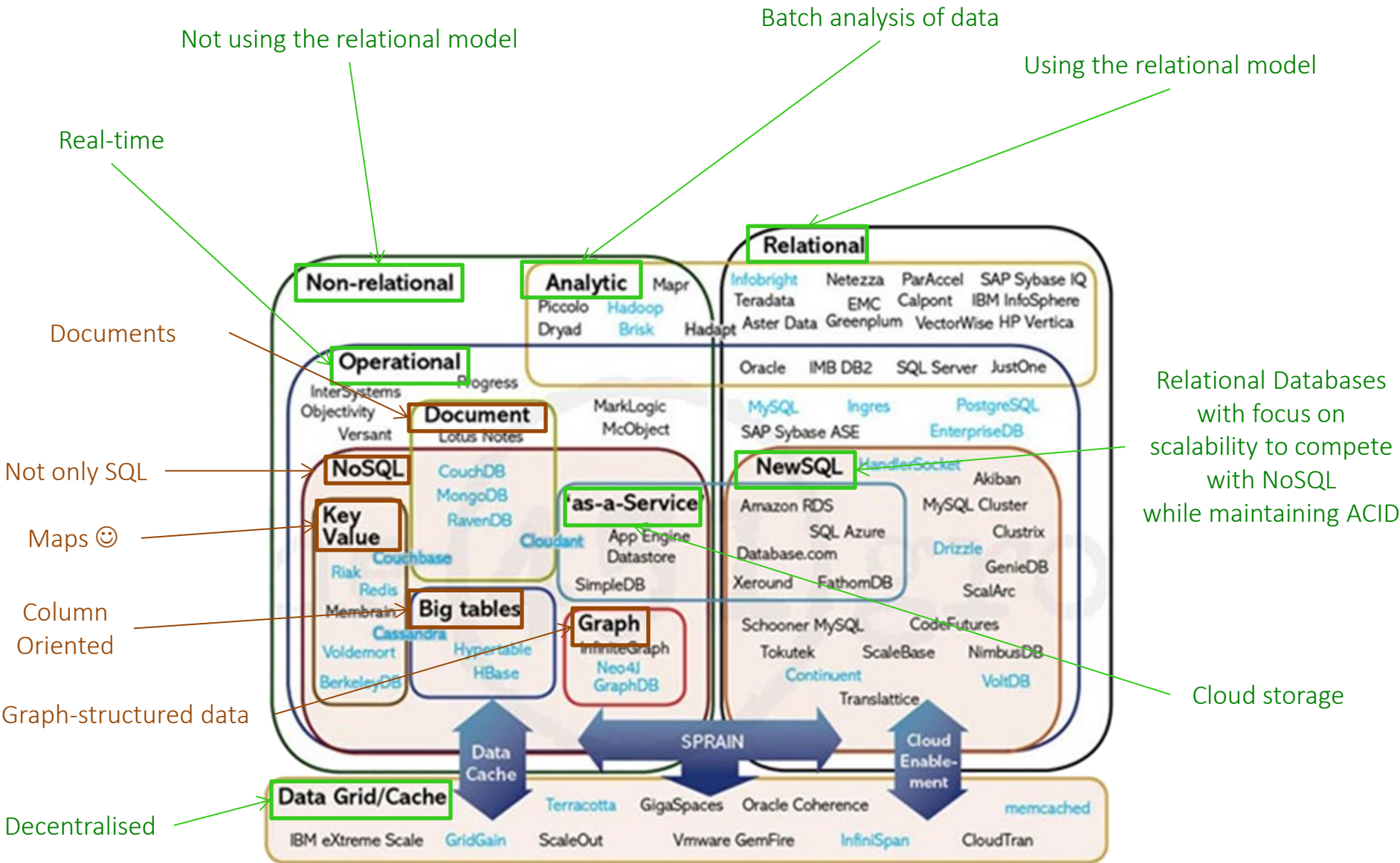


Tabular Store: Cassandra



***cassandra***

# Database Landscape







Questions?