The Relational Problem

Situation:

- Big Data is exploding all around us
- The explosion of Big Data is booming at an increasing rate

Problem:

Relational Database Systems cannot handle the ***volume***, ***velocity*** and ***variety*** of Big Data.

The Relational Problem

Challenges

- How can we collect, process and store so much data?
- How can we process so much data in a timely manner?
- How can we analyze so much data and gain meaningful insights?
- Can my existing systems/architectures handle this?
- Can my existing staff (skills & tools) make the transition?

The Relational Problem

The Relational Problem

The Relational Database: 40+ year-old technology

- Demands STRUCTURE: Tables, Rows, Columns, Keys, Indexes
- Demands ACID Transaction Compliance
  - Keep my data consistent across transactions
  - Data Consistency VS Fast Performance and Throughput

What if my Big Data is UNSTRUCTURED?
What if customers demand speed over consistency?

The Relational Problem

Database Professionals face a challenge:

Do we keep using our relational database systems?

Do we transition to newer NoSQL database systems that are designed to handle Big Data?

The Relational Problem

One approach:

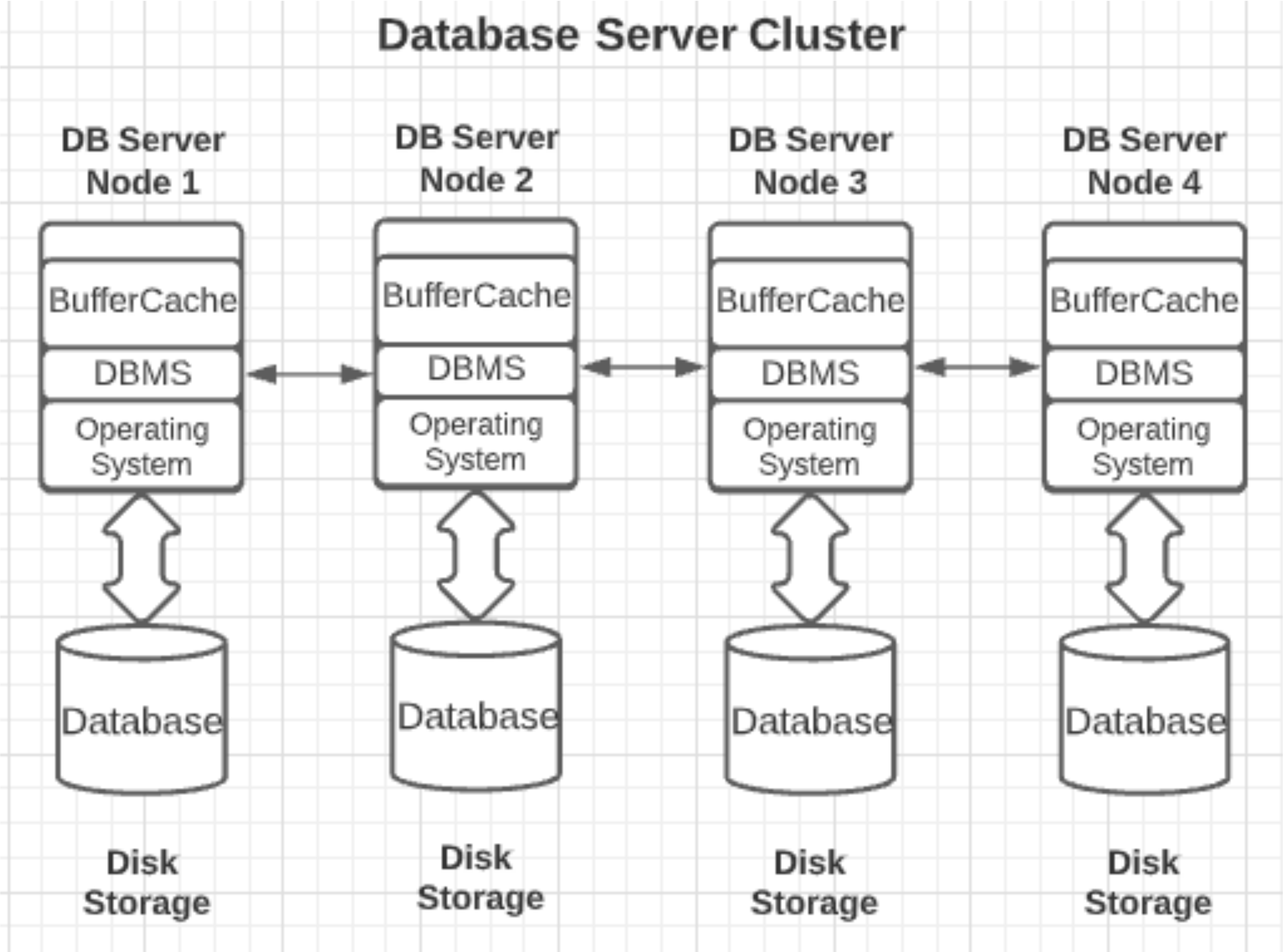
Keep our relational systems, but enhance them to better handle the demands of Big Data.

The Relational Problem

Techniques to Help RDBMS Handle Big Data

- Scaling – Out versus Up
  - UP: Add CPU, Memory, Storage – can be costly
  - OUT: Add server nodes to a cluster
- Clustering
  - "Commodity" Hardware
  - Introduces some overhead for inter-node communications

The Relational Problem



Multiple server nodes in a cluster work together as one

Linear scalability:

Can twice the number of nodes cut processing time in half ?

Side Notes -- Scaling Out

Google has been the pioneer in clustering solutions.

Watch the Google Container Data Center video. (2009)

https://www.youtube.com/watch?v=zRwPSFpLX8I

Challenges they faced and figured out:

- Managing Heat
- Server Maintenance
- UPS – Uninterruptible Power Supply

Side Notes -- Scaling Out

Follow up: Watch these Google Data Center tours (7 years later)

https://www.youtube.com/watch?v=zDAYZU4A3w0

https://www.youtube.com/watch?v=XZmGGAbHqa0

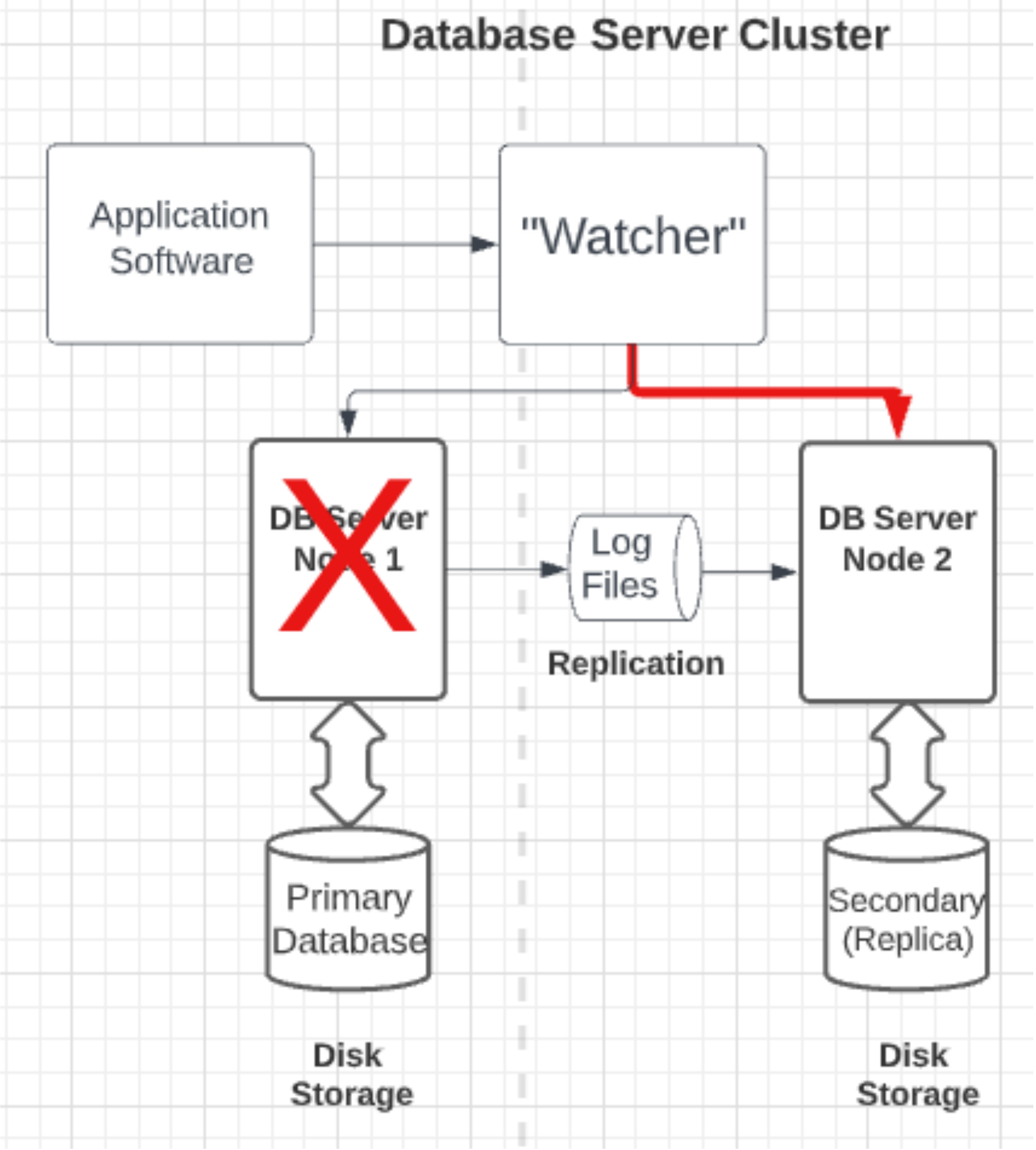
https://www.youtube.com/watch?v=avP5d16wEp0

The Relational Problem

Techniques to Help RDBMS Handle Big Data

- Replication
  - The foundation of clustering
- Parallelization
  - Computing processes are spread out among all the nodes
  - Allows work to be done in parallel
- Sharding
  - Subsets of data ("partitions") are spread across different nodes
  - Allows work to be done in parallel, increases throughput

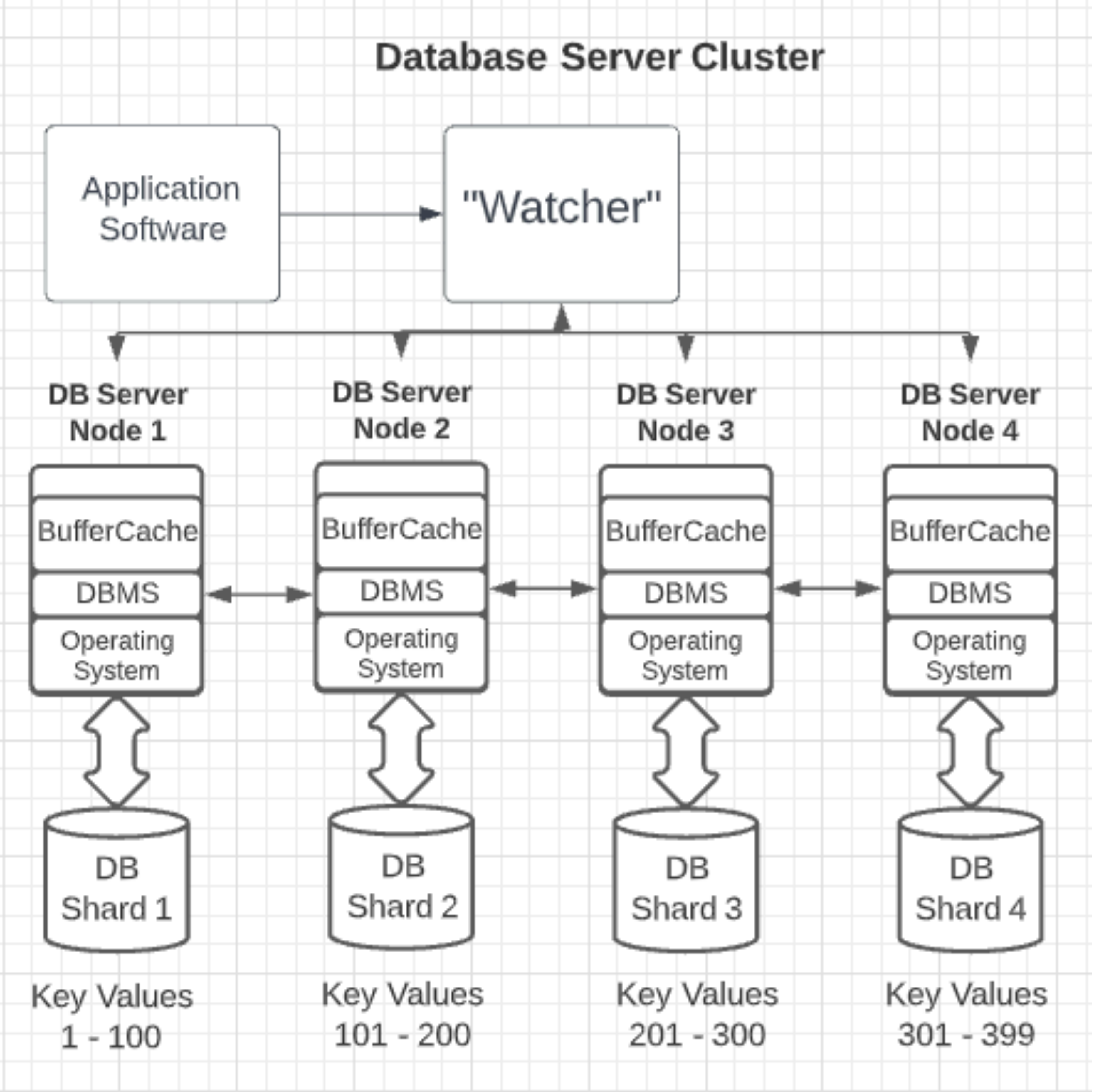
The Relational Problem



The role of the "Watcher" in Replication

- Sometimes called a "coordinator" node
- A component of the replication engine
- A special server node that manages the cluster
- Directs traffic from application software
- Detects if a node goes down, and initiates a FAIL OVER to a secondary node
- Provides HA (High Availability)
  - Eliminates any SPOF (Single Point Of Failure)

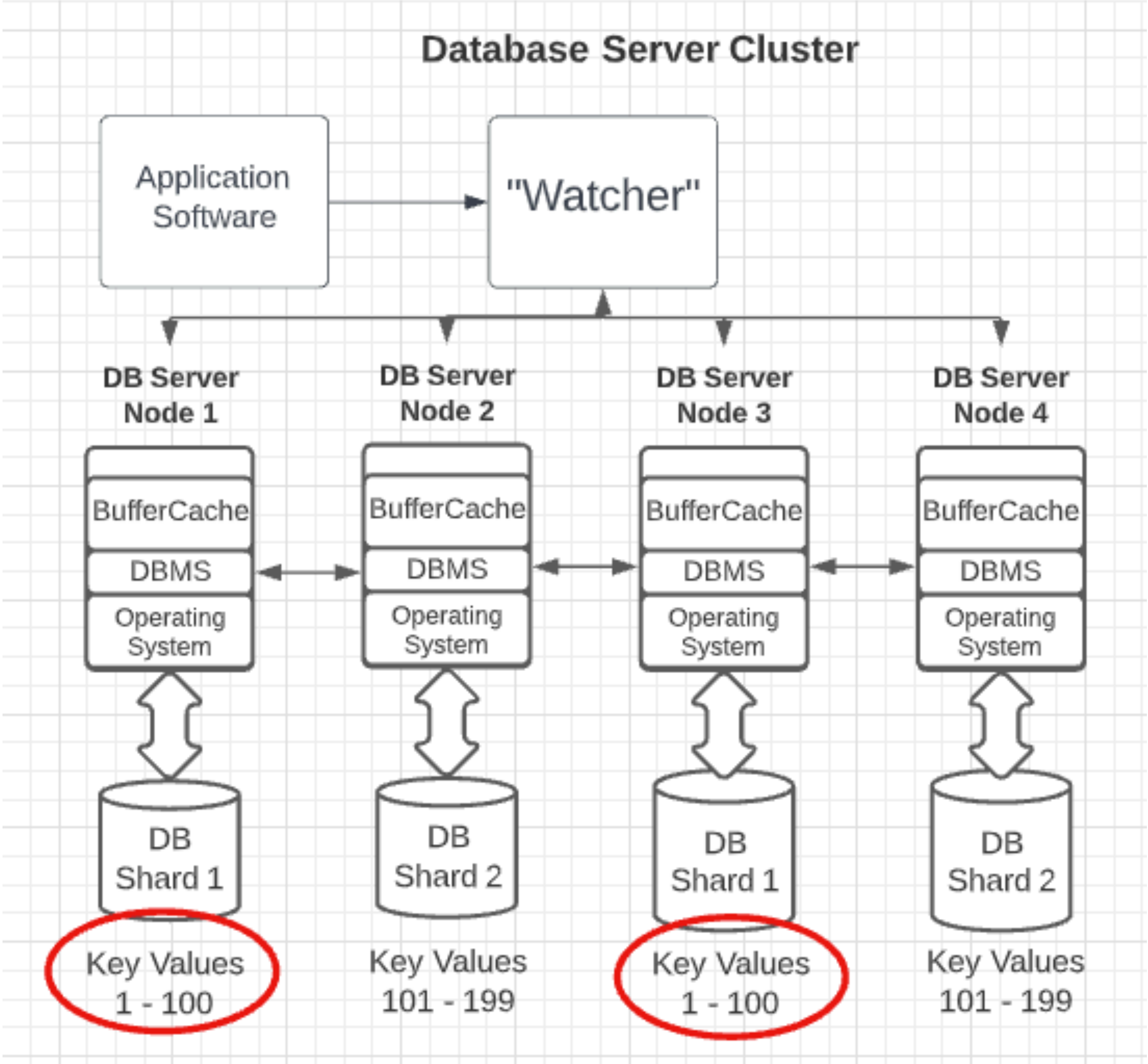
The Relational Problem



Sharding

- Distribute data partitions across nodes
- Based on key range values
- The watcher directs query traffic based upon key values
- Allows parallelization

The Relational Problem



Sharding + Replication

- Data is distributed via sharding

  AND

- Data is stored redundantly via replication
- Further supports parallelization and HA

The Relational Problem

Two modes of replication

- Primary-to-Secondary (also called "Master-Slave")
- Peer-to-Peer

The Relational Problem

Primary-to-Secondary Replication

- All UPDATES must go to Primary node
- READ activity can run against any node

Advantages

- Good READ Scalability – just add more secondary nodes
- Guarantees UPDATE isolation

Disadvantages

- Constrained by the capacity of the Primary node

The Relational Problem

Peer-to-Peer Replication

- UPDATES and READs can go to any node

Advantages

- Good Scalability – just add more nodes
- Provides robust HA in case of node failure

Disadvantages

- Update propogation is very complex
- Difficult to guarantee update isolation

The Relational Problem

Thoughts on Replication

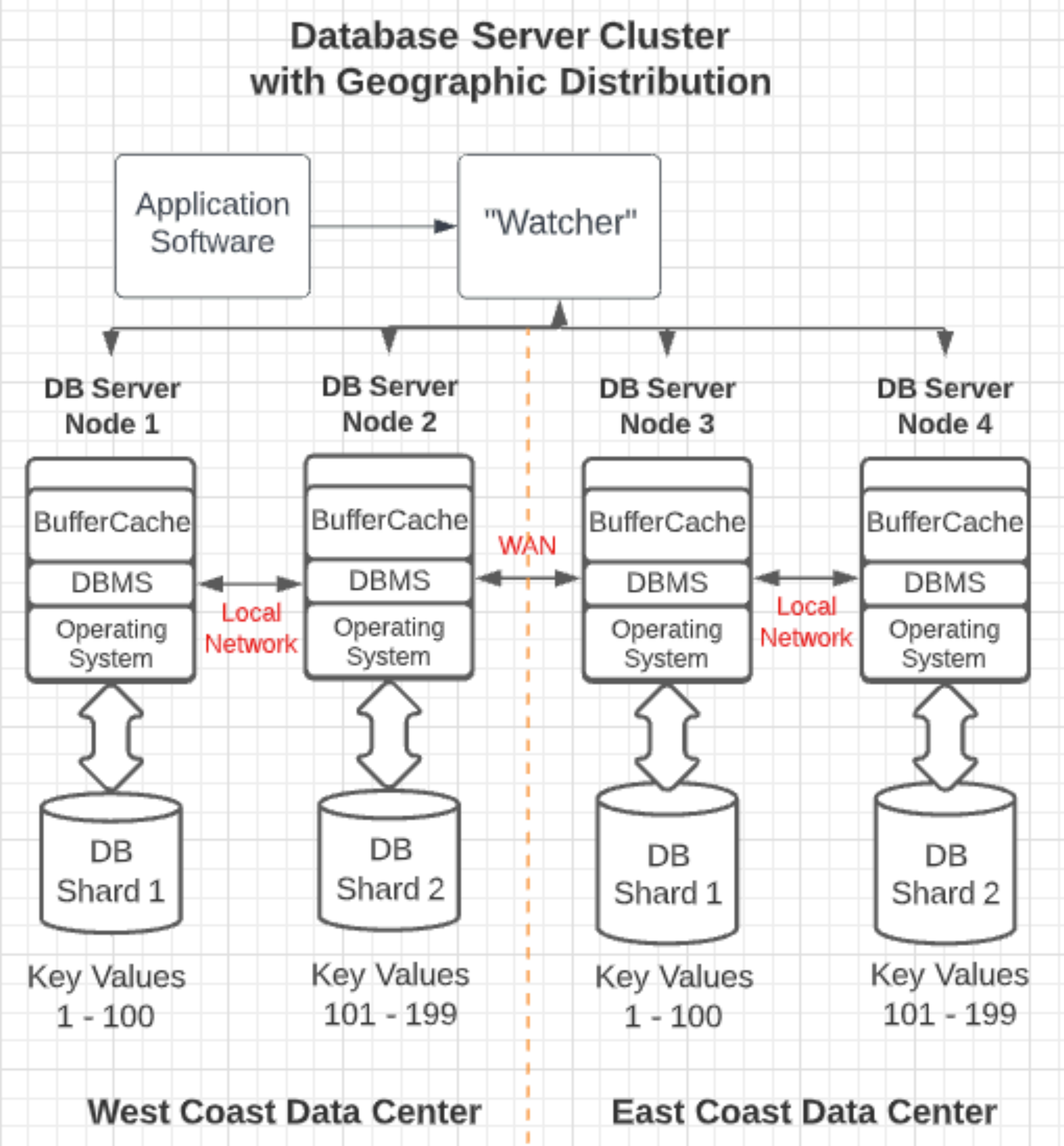
What if replicas are geographically distributed?

- There can be latency (delay) as updates are propogated across a network
- This could cause READ anomalies

Trade-off

- Data Consistency versus Processing Speed

The Relational Problem



Geographically distributed nodes

- Update on Node 1
- READ against Node 3 before the update is propogated
- What if the WAN is down?

Trade-off

- Do I require absolute Data Consistency?
- Do I want to delay remote READs while waiting for update propogation?
- Or, do I seek the fastest possible execution at the cost of perfect consistency?

Solving The Relational Problem

Compromise:

Enhance my RDBMS architecture to Handle Big Data

Or:

Abandon Relational DBMS and adopt NoSQL

Solving The Relational Problem

Pros & Cons

- Keep Using Relational Database Systems
  - Leverage Software Costs / Investment
  - Leverage existing Staff
  - Leverage existing Application Software/Code
- Expand by scaling out (horizontally)
  - Use the Cloud where possible
- Utilize Replication, Sharding
- Leverage Parallelization (queries run in parallel on different nodes)
- Relax ACID compliance where possible for faster throughput

Solving The Relational Problem

Pros & Cons

- Adopt a new NoSQL solution
  - Handles unstructured data
  - Re-train or replace staff
  - Rewrite existing Application Software/Code
- NoSQL Utilizes Replication, Sharding, Parallelization
- NoSQL Relaxes ACID compliance
- NoSQL opts for speed over consistency