The Relational Problem

Situation:

- Big Data is exploding all around us
- The explosion of Big Data is growing 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 wade through 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?
  - Why not?
- Can my existing staff (skills & tools) make the transition?

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
    - Consistency VS Fast Performance and Throughput
  - RDBMS Software Choices
- Traditional Database Server Architecture
  - Memory, CPU, Storage
  - The cost of scaling “UP”

The Relational Problem

Techniques to Help Handle Big Data in RDBMS

- Scaling – Up versus Out
  - UP: Add CPU, Memory, Storage
  - OUT: Add server nodes to a cluster
    - "Linear" scalability
- Clustering
  - "Commodity" Hardware
  - Introduces some overhead for inter-node communications
- Replication
- Parallelization
- Sharding
- Move the COMPUTE closer to the DATA

Side Notes -- Scaling Out

NOTE:

Be sure to watch the Google Container Data Center video. (2009, getting started)

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

Challenges:

- Heat
- Maintenance
- UPS – Uninterruptible Power Supply

Side Notes -- Scaling Out

NOTE:

Be sure to also watch the Google Data Center tours (9 years later)

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

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

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

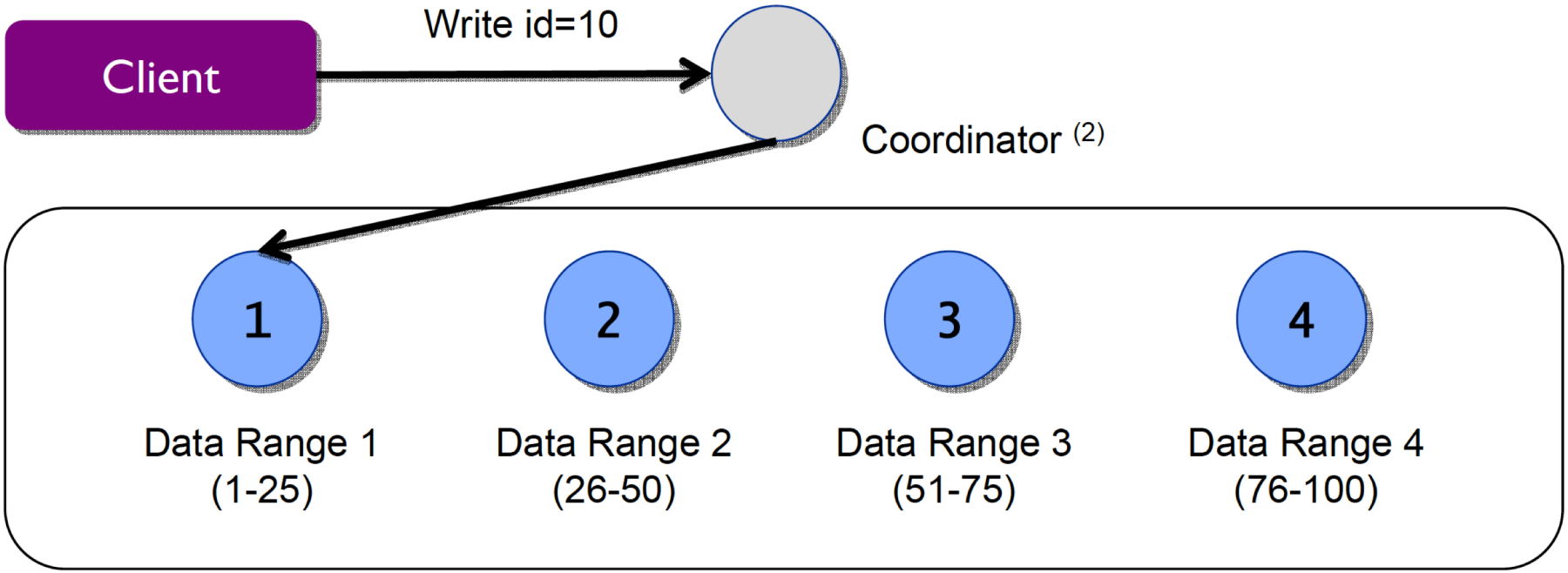
The Relational Problem

Techniques to Help Handle Big Data in RDBMS

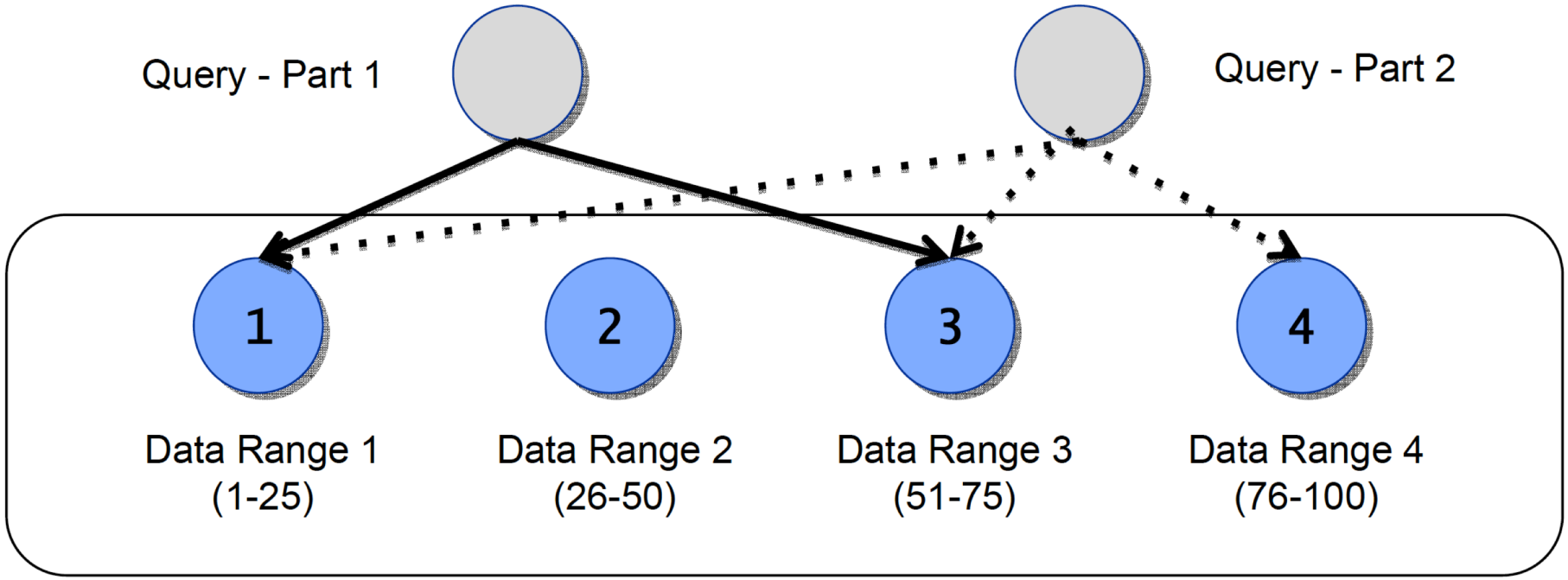
- Clustering without sharding for
  - Built Upon Simple Replication
  - HA - High Availability
  - No SPOF – Single Point of Failure
- Master-Master, Master-Slave Replication
  - The need for a “Watcher”
  - Problems with Geographically Distributed clusters

- **Scale Out**: Use large clusters of commodity hardware
  - As opposed to **scale up** (with heavy duty servers)
  - Often done in the cloud for ease of startup and growth
- **Distribute**: Spread data and processing across nodes
  - **Distribute the data**, increasing data capacity
  - **Move processing to the data**, increasing processing capacity
- **Replicate data**: For availability and processing capacity
- **Relax Restrictions**: In various ways
  - **Relax** transactional and consistency guarantees
  - **Denormalize** data - sometimes radically
- **Specialize**: Design for your data and needs
  - Use data models and tools that work well for them

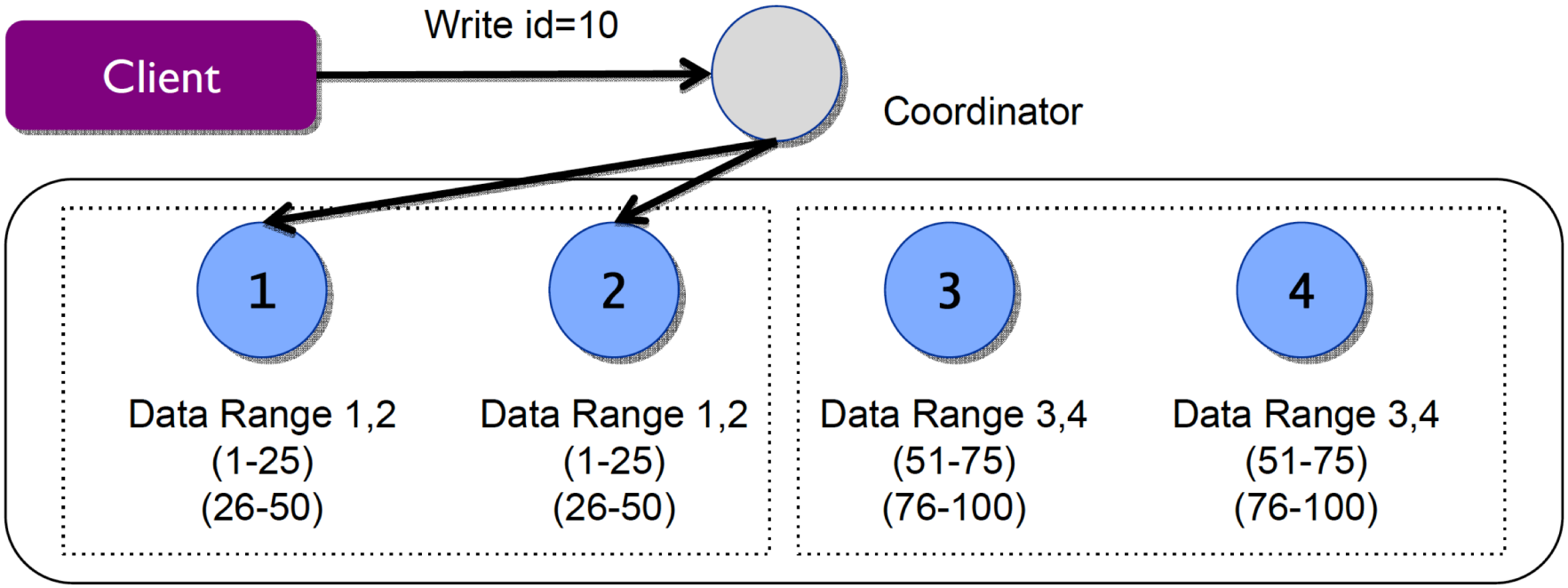
- **Sharding** partitions large data sets into smaller shards
  - Each shard hold parts of the data, so data is distributed
- Requires a partitioning scheme to shard data across nodes (1)
  - As well as a way to determine where sharded data goes to
- Below, 4 nodes each hold ¼ of a data range (1-100)
  - An incoming write with id=10 goes to node 1



- Consider a join that has to first read id=10 and id=70
  - And based on data in the read, reads data with id = 12, 55, and 80
    - Perhaps from another table
- You end up with many multiple reads across the cluster
  - Lots of network traffic and work to merge all the data into a result
  - Inherent in the relational model, and affects scalability [1]



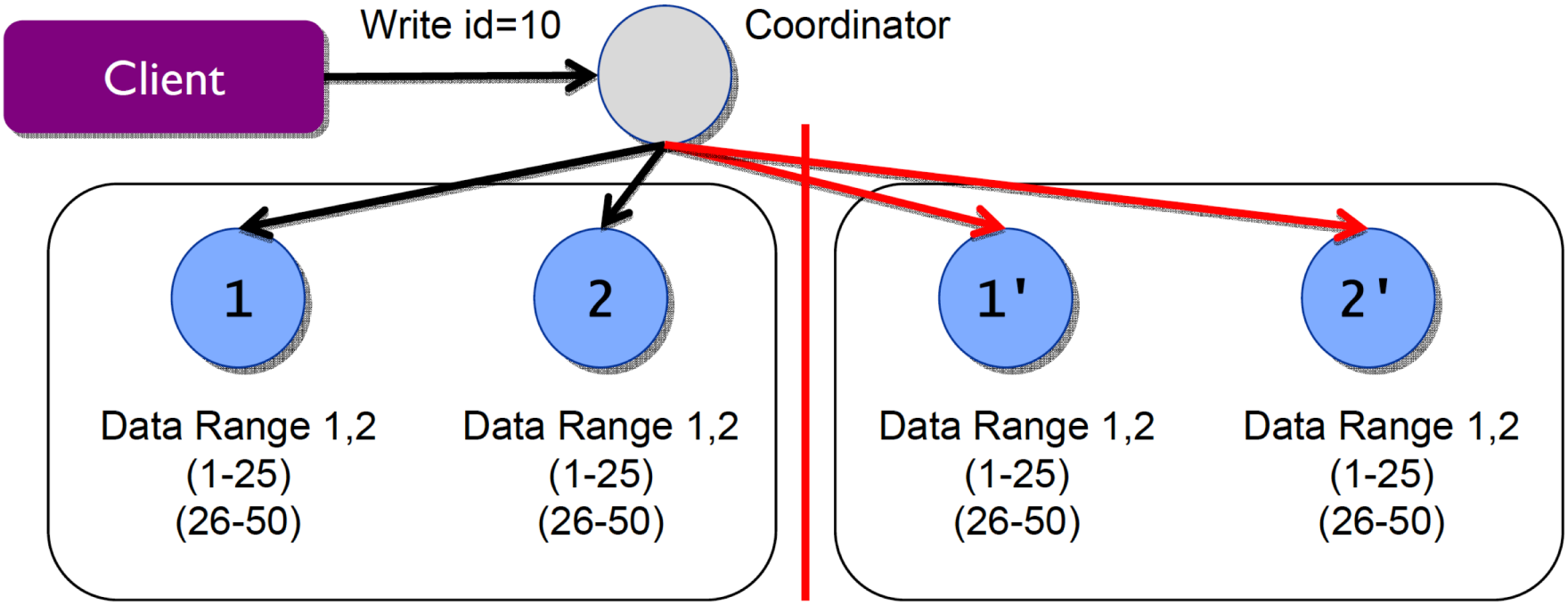
- Replication **duplicates data** to multiple nodes (1)
  - Increases scalability - operations can go to all replicas
  - Increases fault tolerance - operations continue if any replica is up
  - Two general types: **master/slave** and **peer-to-peer**
- Below, a write with id=10 goes to both nodes 1 and 2
  - The data is replicated to both nodes (2)



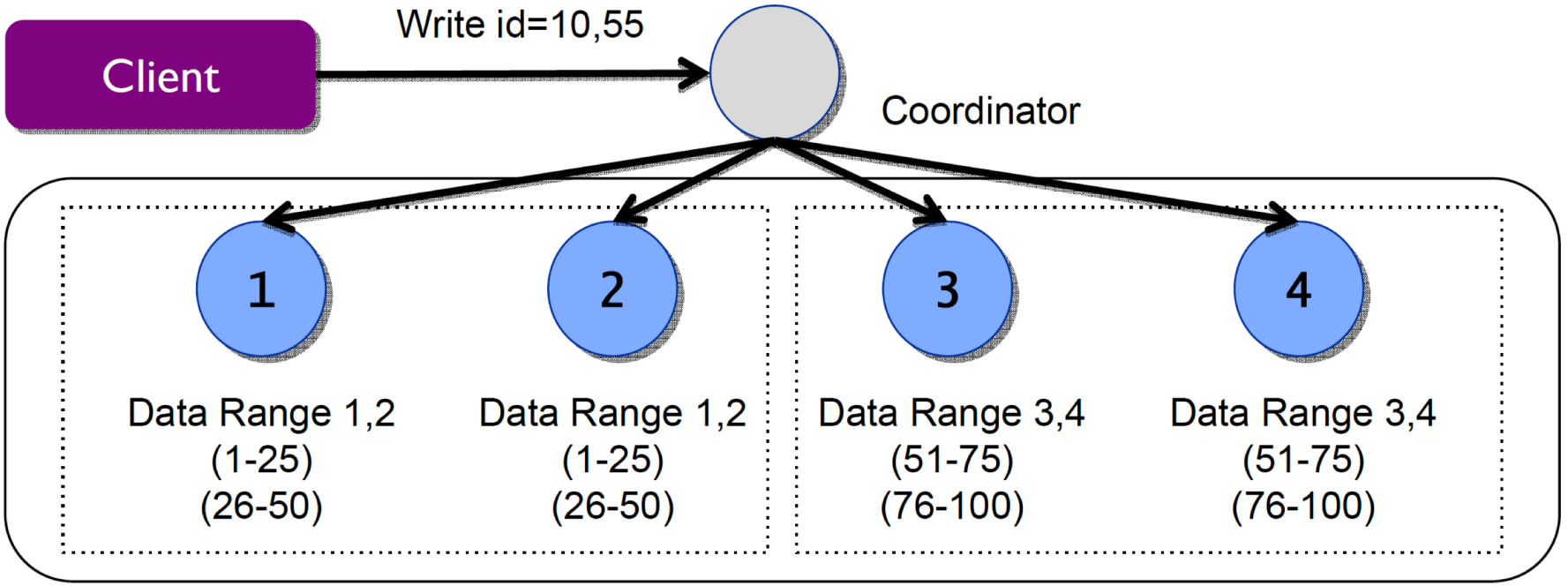
- **Master/Slave replication**: One node (master) is primary holder of data (1)
  - Updates go to master, and are then propagated to slaves
  - Reads can go to any node (master or slave)
  - Provides good scalability/availability for reads by adding slaves (2)
    - Since reads can go to any node with the data
- **Advantages**:
  - Reads scale well, and can easily be made highly available
  - Can give guarantees on update isolation (3)
- **Issues**:
  - Updates are limited by master's capabilities, limiting throughput
  - If master fails, failover to slave can occur, but may take time or manual intervention, limiting availability

- **Peer-to-peer replication**: All nodes are equal
  - Reads and writes can go to any node
  - Updates on any node are propagated to all replicas
  - Provides good scalability/availability for reads and writes
    - Both can go to any node with the data
- **Advantages**:
  - Reads and writes scale well and are highly available
  - Failure of a node doesn't necessarily prevent any reads/writes
- **Issues**:
  - These solutions are fairly complex
  - Hard to give any guarantees on updates (1)

- Replication **raises data consistency issues** among replicas
  - e.g. A read occurs before propagation of a write to all replicas (1)
- Consider two datacenters - with data replicated across them
  - What if the two datacenters can't communicate (a **partition**) (2)
  - You now face choices in how you continue to operate (or not)



- Consider a write that has to write to rows with ids=10 and 55
  - You'll write to all 4 nodes (because of replication and sharding)
  - For ACID guarantees, you'll have to use a complex TX protocol (e.g. 2 phase commit) to insure data integrity
  - The overhead of this protocol will affect scalability, and may add other complexities



Solving The Relational Problem

Compromise: Enhancing Relational to Handle Big Data

- Keep Using Relational 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, more throughput