VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

Partitioning and Replication

Lecturer* : *Dr. Pavle Mogin

SWEN 432
Advanced Database Design and Implementation

Plan for Data Partitioning and Replication

- Data partitioning and replication techniques
- Consistent Hashing
  - The basic principles
  - Workload balancing
- Replication
- Membership changes
  - Joining a system
  - Leaving a system

- ***Readings:*** Have a look *at Readings at the Course Home Page*

Data Partitioning and Replication

- Partitioning means storing different parts of a data base on different servers
- Replication means storing copies of the same data base on different machines
- There are thee reasons for storing a database on a number of machines (nodes):
  - The amount of data exceeds the capacity of a single machine (partitioning),
  - To allow scaling for load balancing (partitioning), and
  - To ensure reliability and availability by replication

Data Partitioning and Replication (2)

- There are a number of techniques to achieve data partitioning and replication:
  - Sharding (partitioning)
  - Consistent Hashing (partitioning)
  - Memory caches (replication and work load partitioning)
  - Separating reads from writes (replication),
  - HA Clustering (replication)

Cashes and Separating R and W

- ***Memory Caches*** keep most frequently requested parts of a database in main memories of different servers and send a client request to the server caching the data needed
  - Fast response to clients
  - Off-loading database servers (work load partitioning)
- ***Separating reads from writes***:
  - One or more servers are dedicated to writes (master(s)),
  - A number of replica servers are dedicated to satisfy reads (slaves),
  - The master replicates the updates to slaves
    - If the master crashes before completing replication to at least one slave, the write operation is lost
    - The most up to date slave undertakes the role of the master

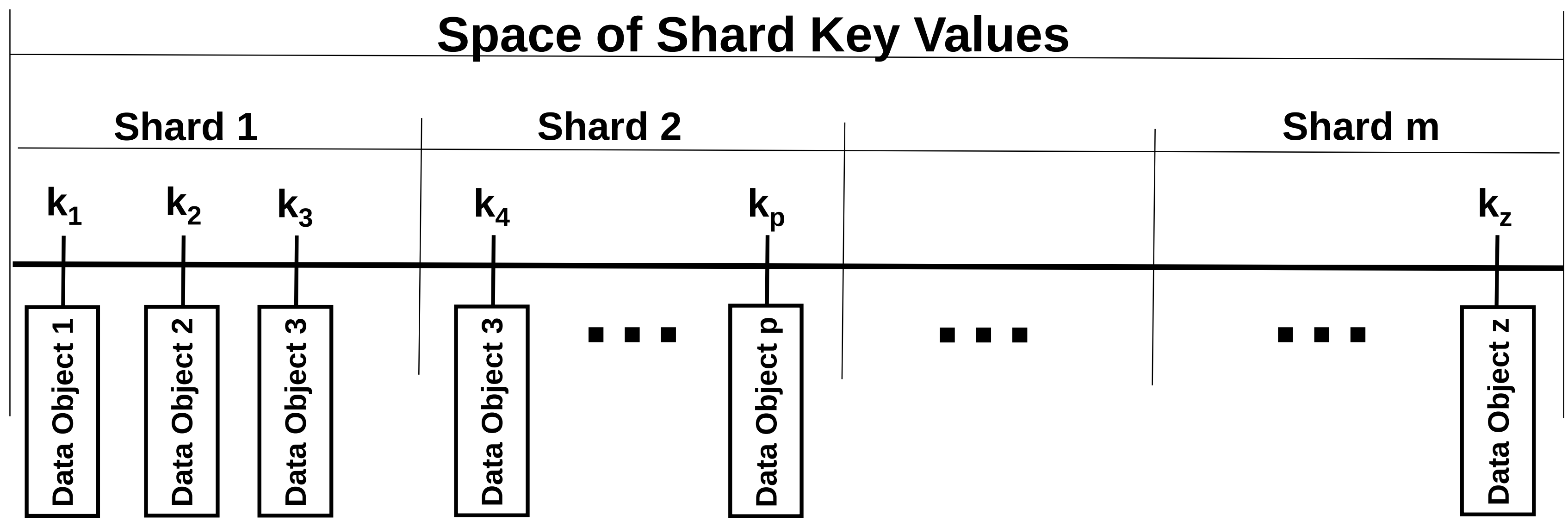
Clustering

- ***A High-Availability cluster*** (also known as **HA cluster** or **failover cluster**) is a group of servers that store the same database (or part of it)
  - One computer services client's request, the others replicate data, only
- When a HA cluster detects a hardware/software fault, it immediately restarts the application on another system without requiring administrative intervention
  - A process known as ***failover***
- HA clusters use redundancy to eliminate **single points of failure** (SPOF)
  - Example: Master/Slave mode of replication

Sharding (1)

- Database Sharding is a “shared-nothing” partitioning scheme for large databases across a number of servers, that enables higher levels of database performance and scalability
  - It is a horizontal partitioning schema where data objects having the neighbouring shard key values are stored in the same shard on the same node
  - It assumes that queries ask for a single data object or data objects having shard keys from an interval of values

Sharding (2)


Space of Shard Key Values
Shard 1
Shard 2
Shard m
k1
k2
k3
k4
kp
kz
Data Object 1
Data Object 2
Data Object 3
Data Object 3
Data Object p
Data Object z
...
...
...

Consistent Hashing

- Consistent hashing is a data partitioning technique
- An obvious (but naive) way to map a database object *o* to a partition *p* on a network node is to hash the object's primary key *k* to the set of *m* available nodes

$$p = (k) \bmod m$$

- In a setting where nodes may join and leave at runtime, the simple approach above is not appropriate since all keys have to be remapped and most objects moved to another node
- ***Consistent hashing*** is a special kind of hashing where only (*K* / *m*) keys need to be remapped, where *K* is the number of keys

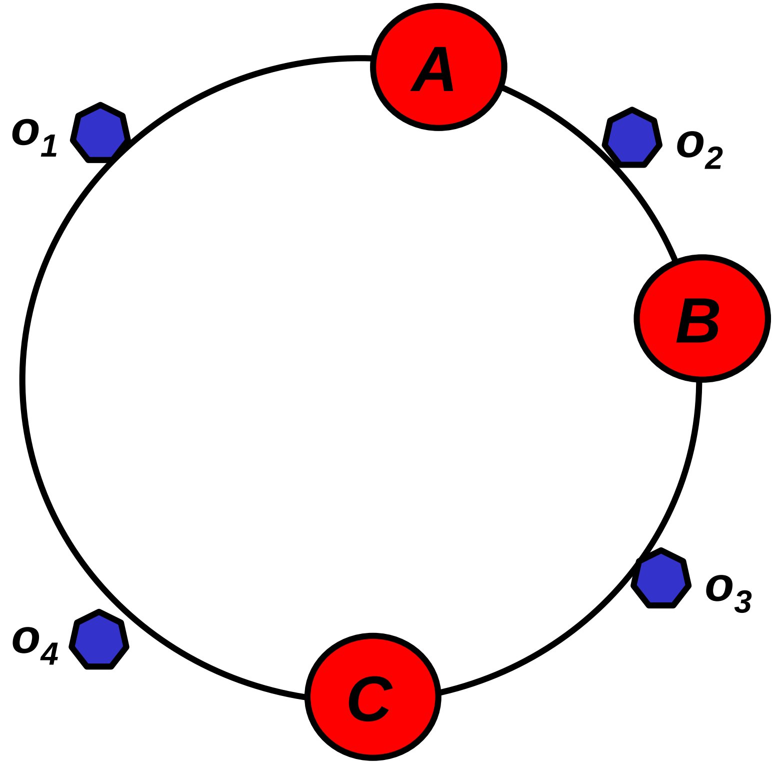
Consistent Hashing (The Main Idea)

- The main idea behind the consistent hashing is to associate each node with one or more hash value intervals where the interval boundaries are determined by calculating the hash of each node identifier
- If a node is removed, its interval is taken over by a node with an adjacent interval
- All the remaining nodes remain unchanged
- The hash function does not depend on the number of nodes m
- Consistent hashing is used in the partitioning component of a number of CDBMSs

Consistent Hashing (Basic Principles 1)

- Each database object is mapped to a point on the edge of a circle by hashing its key value
- Each available machine is mapped to a point on the edge of the same circle
- To find a node to store an object, the CDBMS:
  - Hashes the object’s key to a point on the edge of the circle,
  - Walks clockwise around the circle until it encounters the first node,
  - Each node contains objects that map between its and the previous node point

Consistent Hashing (Example 1)



Objects o_1 and o_4 are stored on the node A

Object o_2 is stored on the node B

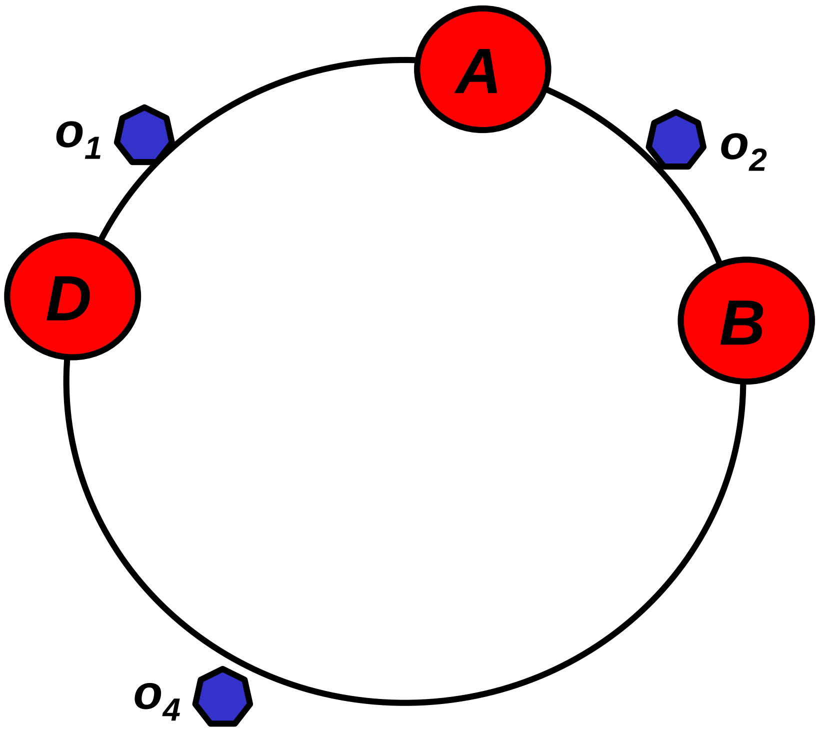
Object o_3 is stored on the node C

Consistent Hashing (Basic Principles 2)

- If a node leaves the network:
  - All data objects of the failed node are gone,
  - The next node in the clockwise direction stores all the new data objects that would belong to the failed node
- If a node is added to the network, it is mapped to a point and:
  - All the new data objects that map between the point of the new node and the first counter clock wise neighbour, map to the new node

Consistent Hashing (Example 2)

The node C has left and the node D has entered the network



Object o_1 is stored on the node A

Object o_4 is still stored on the node A, although now belongs to the node D

Object o_2 is stored on the node B

Object o_3 is gone, although now belongs to the node D

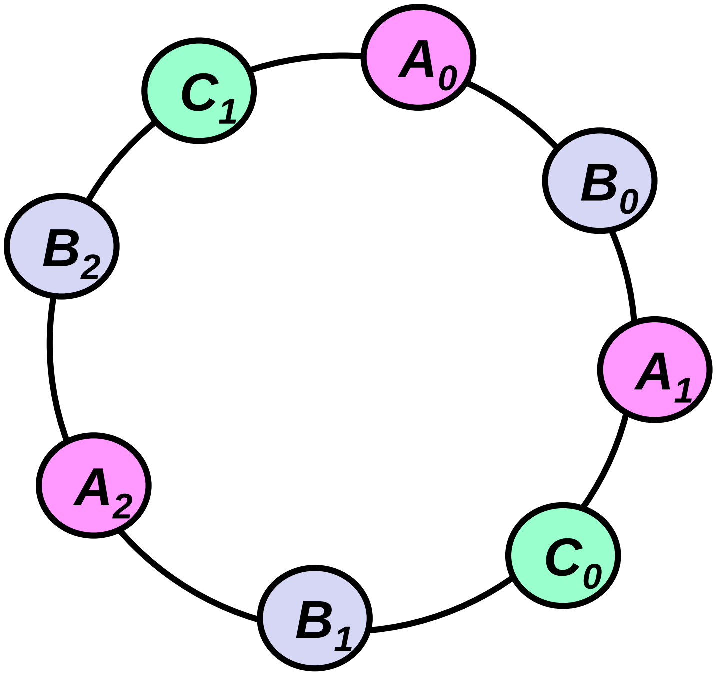
Consistent Hashing (Problems)

- The basic consistent hashing algorithm suffers a number of problems:
  1. Unbalanced distribution of objects to nodes due to different intervals of points belonging to nodes
     - It is the consequence of determining the position of a node as a random number by applying a hash function to its identifier
  2. If a node has left the network, objects stored on the node become unavailable
  3. If a node joins the network, the adjacent node still stores objects that now belong to the new node
     - But client applications ask the new node for these objects, not the old one (that actually stores objects)

Consistent Hashing (Solution 1)

- An approach to solving the unbalanced distribution of database object is to define a number of virtual nodes for each physical node:
  - The number of virtual nodes depends on the performance of the physical node (cpu speed, memory and disk capacity)
  - The identifier of a virtual node is produced by appending the virtual node's ordinal number to physical node's identifier
  - A point on the edge of the circle is assigned to each virtual node
  - This way database objects hashed to different parts of the circle may belong to the same physical node
  - Experiments show very good balancing after defining a few hundreds of virtual nodes for each physical node
- By introducing *k* virtual nodes, each physical node is given *k* random addresses on the edge of the circle
  - Virtual node addresses are called ***tokens***

Balancing Workload (Example)



Physical nodes A and B have three virtual nodes

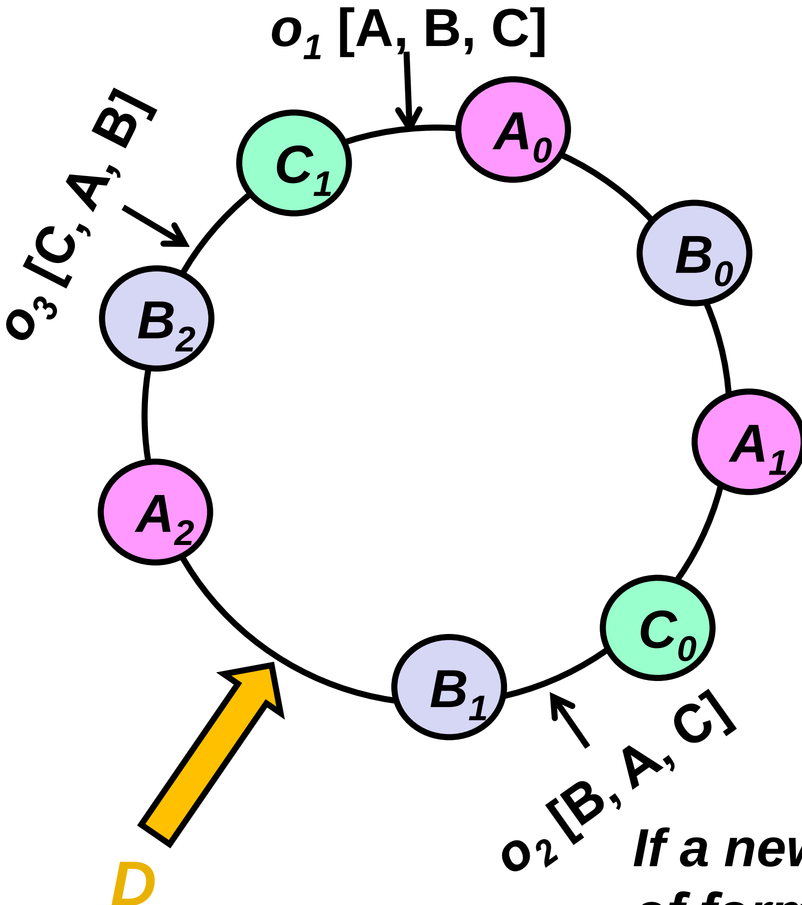
Physical node C has only two virtual nodes

Let the physical node A have k virtual nodes. Then, A_i for i = 0, 1,..., k-1 is the identifier of the virtual node i of the physical node A

Consistent Hashing (Solutions 2&3)

- Problems caused by leaving of an existing and joining of a new node are solved by introducing a replication factor ***n*** (> *1*)
  - This way, the same database object is stored on *n* consecutive physical nodes (the object's home node and *n* – *1* nodes that follow in a clock wise direction)
- Now, if a physical node leaves the network, its data objects still remain stored on *n* – *1* nodes following it on the ring and will be found by the basic search algorithm already described
- If a new node enters the network, some of its data objects will be found by accessing his first clockwise neighbour (the new search algorithm)

Replication (Example)



Assume replication factor n = 3

Object o_1 will be stored on physical nodes A, B, and C

Object o_2 will be stored on physical nodes B, A, and C

Object o_3 will be stored on physical nodes C, A, and B

If the node A leaves, object o_1 will still be accessible on the node B and node C

If a new node D enters the network, some of former node's A objects will be accessible on the node A via the node D

Optimistic Replication

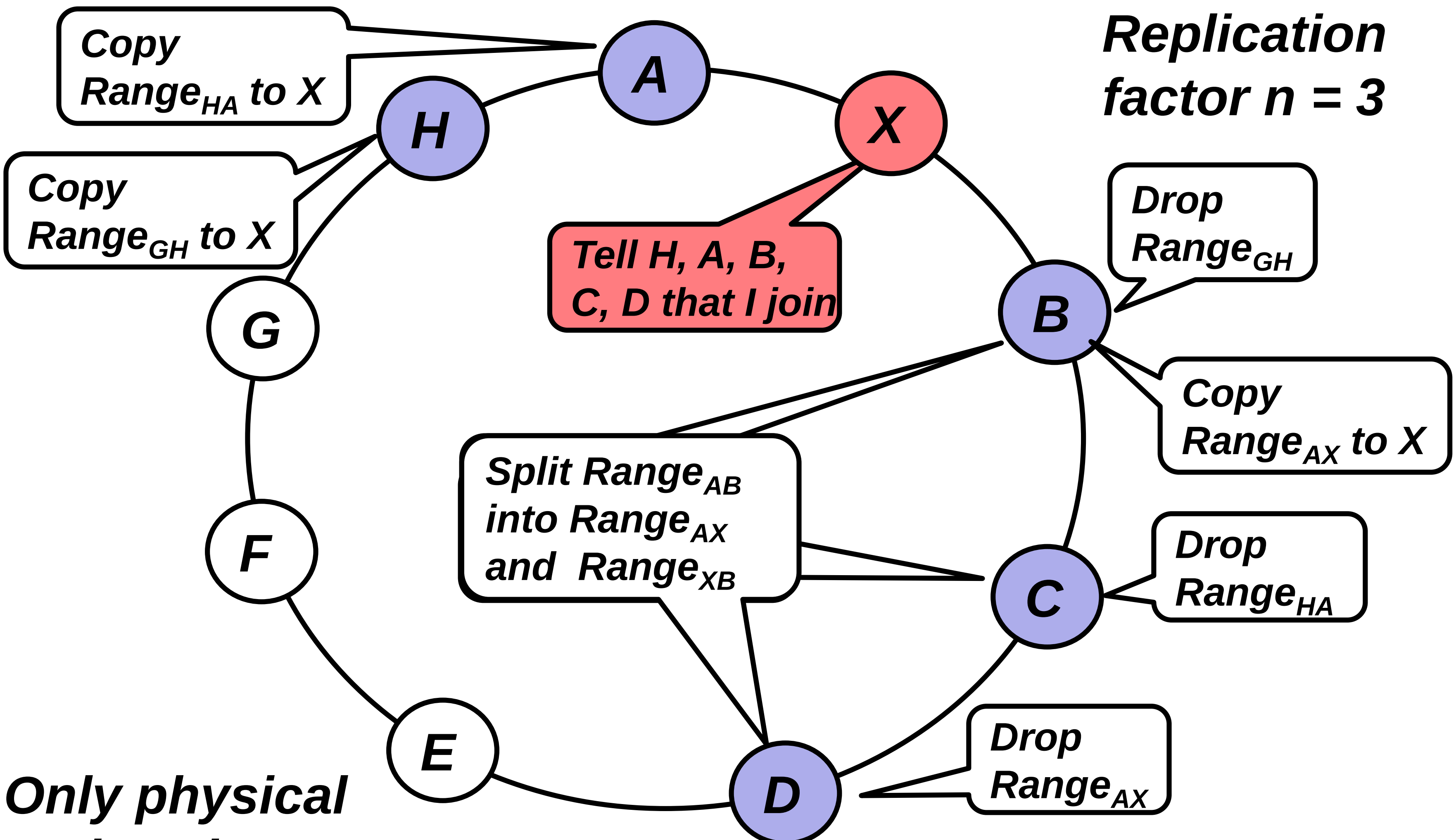
- **Optimistic replication** (also known as lazy replication) is a strategy in which replicas are allowed to diverge
  - Traditional pessimistic replication systems try to guarantee that all replicas are identical to each other all the time, as if there were only a single copy
  - Optimistic replication does away with this in favour of eventual consistency, meaning that replicas are guaranteed to converge only when the system has been temporarily inactive
- As a result there is no longer a need to wait for all of the copies to be synchronized when updating data, which helps concurrency and parallelism
- The trade-off is that different replicas may require explicit reconciliation later on, which might then prove difficult or even insoluble

Membership Changes

- The process of nodes leaving and joining the network is called ***membership changes***
- The following slides consider principles of memberships changes that may or may not apply to each Cloud DBMS using gossipping
- When a new node joins the system:
  1. The new node announces its presence and the identifier to adjacent nodes (or to all nodes) via broadcast
  2. The neighbours react by adjusting their object and replica ownerships
  3. The new node receives copies of datasets it is now responsible for from its neighbours

Node X Joins the System

Replication factor n = 3

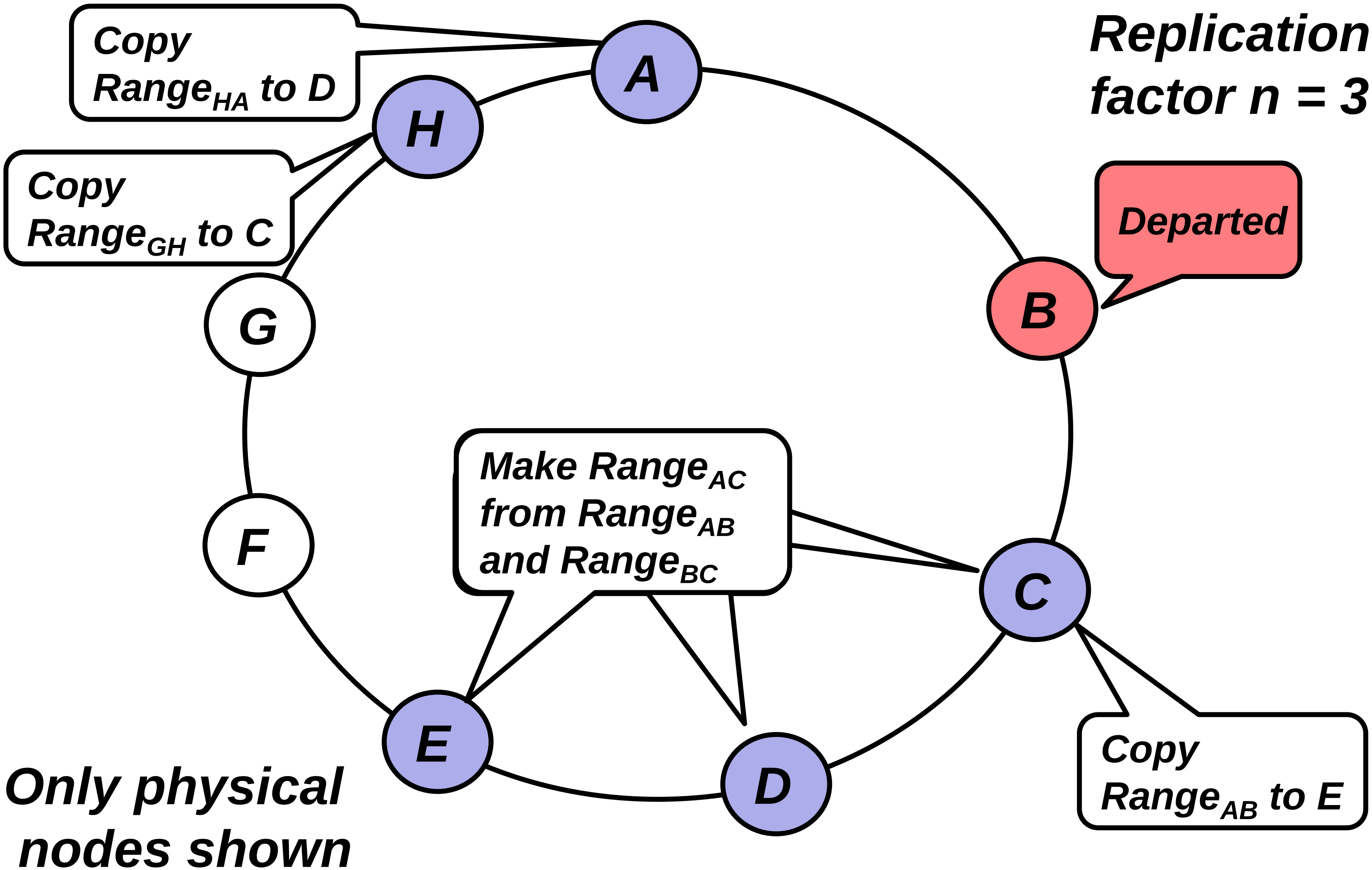


Only physical nodes shown

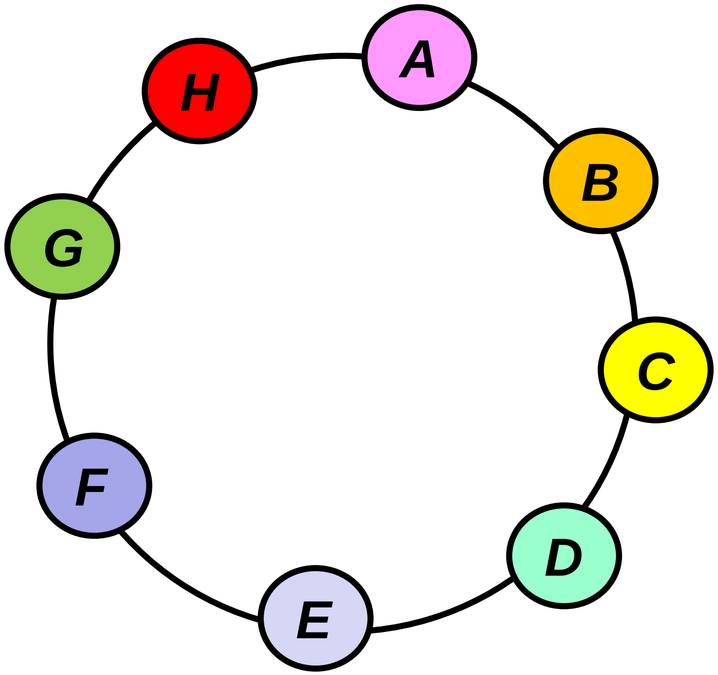
Membership Changes (Leaving)

- If a node departs the network (for any reason):
  1. The other nodes have to be able to detect its departure,
  2. When the departure has been detected, the neighbours have to exchange data with each other and to adjust their object and replica ownerships
- It is common that no notification is given if a node departs for any reason (crush, maintenance, decrease in the work load)
- Nodes within a system communicate regularly and if a node is not responding, it has departed
- The remaining nodes redistribute data of the departed node from replicas and combine ranges of the departed node and its clock wise neighbour

Node B Departs the System



Consistency and Availability Trade-offs



Worst case: 1 node

Best case: 2 nodes

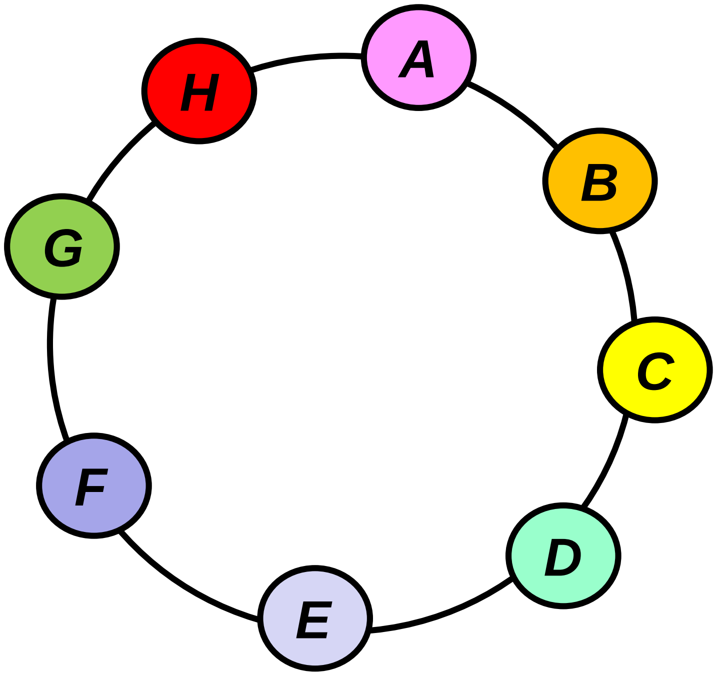
Assume, all failing nodes fail at the very same moment of time and there were no time for membership changes

Replication factor 3

Strong consistency under quorum required for 100% of data

How many nodes in total can get down (be not available)?

Consistency and Availability Trade-offs



Worst case: 2 nodes

Best case: 5 nodes

Assume, all failing nodes fail at the very same moment of time and there were no time for membership changes

Replication factor 3

Eventual consistency required for 100% of data

How many nodes can get down (be not available)?

Summary (1)

- Techniques to achieve data partitioning and replication are:
  - Memory caches,
  - Separating reads from writes,
  - Clustering, and
  - Sharding
- The main idea of consistent hashing is to associate each physical node with one or more hash value intervals where hash values of node identifiers represent interval boundaries
  - Introducing the virtual nodes solves the problem of unbalanced work load
  - Introducing replication solves the problems caused by nodes leaving and joining the network

Summary (2)

- The process of nodes leaving and joining the network is called membership changes
  - When a node leaves the network, other nodes combine its range and the range of its clockwise neighbor and redistribute data
  - If a node joins the network, the neighbors react by adjusting their object and replica ownerships and the new node receives copies of datasets it is now responsible for